TECHNICAL REFERENCE · F06

Technical architecture of the Dweve platform

Responsibilities, contracts, state, evidence, deterministic foundations, deployment topologies, failure, recovery, and a production-proof method.

ARCHITECTURE CONTRACTS PRODUCTS FOUNDATIONS EVIDENCE OPERATIONS

Read this when a product overview is not enough and the engineering, security, operations, and assurance teams need one shared technical model.

F06

CONTENTS

The whole technical system, from contract to proof

This reference deliberately keeps products, open foundations, deployment, evidence, and customer responsibilities in one sequence. Page ranges are fixed to this 118-page source edition.

PRIMARY READERS	READING DEPTH	DECISION THIS SUPPORTS
Architects, senior engineers and technical evaluators	Deep technical	A long-form architecture reference for evaluators who need to follow the system, challenge its boundaries, and turn public claims into installed evidence.

SOURCE DISCIPLINE

Dweve-specific facts are taken directly from the English website text audited 2026-07-20; web-navigation wording is humanised for print. Evaluation guidance is editorial, and scenarios and derived visuals are illustrative. Unsupported synthesis is replaced by canonical copy or omitted.

READ THIS FIRST · STATUS

Scope: architecture explanation, not an implementation specification

The website is the sole canonical source for this edition. This reference connects its published material into an evaluable model while keeping the boundary between public description and production proof explicit.

The document explains the public architecture through owned responsibilities, state, interfaces, artefacts, evidence, deployment, and failure. It follows a request across Fabric, Loom, Nexus, Spindle and deeper products or foundations where the selected workload requires them. It also treats customer systems, operators, human authority, and independent review as first-class parts of the system context.

It does not invent current API schemas, repository state, source rights, product availability, benchmark conditions, assurance reports, service commitments, or contract terms. Illustrative envelopes, identifiers, fields, scenarios, and acceptance procedures show the shape of a good implementation or evaluation; the installed platform and current documentation must supply exact names and supported versions.

Where Kera is not licensed, Core uses its Rust-native execution path. Where Kera is included, Core gains the complete Kera graph, compiler, and execution foundation. Do not collapse the products. Core owns AI operations, representations, training, inference, and serving. Kera owns graph semantics, effects, ownership, planning, compilation, and heterogeneous execution.

STATES DIRECTLY

Public mechanism

A recurring collection job should not behave like five unrelated scripts.

EXPLAINS

Architecture consequence

How responsibilities and artefacts should be evaluated together, labelled as synthesis or guidance.

ILLUSTRATES

Contract shape

Example fields and cases are visibly illustrative and not represented as a current schema.

REQUIRES

Installed proof

Versions, interfaces, source/binaries, harnesses, assurance, agreements, and customer acceptance.

PRIMARY READER

Enterprise, platform, security, data, ML, systems, and infrastructure architects; senior engineers; and technical evaluators deciding whether to run a production proof.

READ THIS FIRST · VOCABULARY

Terminology: five axes and six kinds of technical object

Precise nouns prevent the architecture from collapsing into one undifferentiated "AI platform" box.

The five architecture axes

A **product** owns an operating responsibility and state. A **layer** constrains where responsibility and dependencies sit. An **open foundation** implements an inspectable mechanism that can contribute an artefact. An **entry door** says whether a person uses Fabric, an application calls APIs, or an organisation licenses the platform. A **deployment posture** says who operates the runtime and where it is bounded.

The six object classes

Every operation, every policy check, every intermediate value, every sampled token is hashed, the hashes are assembled into a Merkle DAG, the root is signed with Ed25519, and the whole thing becomes a single AION proof artefact. Re-execute it on any platform (x86, ARM, RISC-V, WASM) and the output root hash matches or verification fails closed. No floating-point drift in hard mode. No non-deterministic ops in the hot path. The spec is public; the verifier is Apache 2.0.

TERM	ANSWERS	MUST NOT BE CONFUSED WITH
Product	Who owns the job/state?	Commercial tier or layer
Foundation	Which mechanism is inspectable?	Automatic release/right
Contract	What crosses with which meaning?	Transport endpoint alone
Manifest	Which versions actually apply?	Latest/current config
Evidence	Which proposition can be checked?	Every operational log
Replay	Which declared work is reconstructed?	A universal recreation of history

Most people open managed Fabric in a browser and start without an infrastructure project.

EDITORIAL CONVENTION

"Must" and "should" in evaluation guidance describe what a credible design or proof needs; they are not claims that the current product implements an unnamed schema or control.

READ THIS FIRST · CLAIM DISCIPLINE

How to classify and verify a technical claim

Architecture facts, measured results, assurance statements, availability, research, and commercial rights require different proof.

An **architecture fact** is tested through manifests, interfaces, state and dependency maps, installed behaviour, and failure paths. A **measured result** requires its workload or corpus, hardware, versions, configuration, warm/cold state, method, repetitions, raw output, comparison boundary, and non-SLA qualification. Counts must define what is counted and in which release.

An **assurance statement** needs a threat or governance boundary and applicable evidence. Integrity is not correctness; offline capability is not an exercised air gap; support for traceability is not certification. An **availability statement** needs current repository, release, delivery, licence, and product scope. A **research statement** must not enter the production design without a separately supported status.

A **commercial fact** such as included products, usage treatment, source access, support, update rights, or exit depends on the current written offer and agreement. Technical architecture can identify why a right matters; it cannot grant it. Every relied-on claim therefore ends in verified evidence, accepted limitation with owner, contractual commitment, remediation, or a decision not to proceed.

01 **Locate**
Like using the same edition of a map. Keep the map and the question fixed, and the route stays reproducible.

02 **Classify**
Architecture, measure, assurance, availability, research, or commercial.

03 **Scope**
Product/version, workload, hardware, condition, and limitation.

04 **Request**
Manifest, interface, source/binary, harness, report, or agreement.

05 **Reproduce**
Run the material claim in the customer proof boundary.

06 **Dispose**
Verify, narrow, accept risk, remediate, contract, or stop.

READING RULE

A number in a product or foundation page belongs to that published method. This technical reference explains how to reproduce it; it does not convert it into a universal platform SLA.

01

SYSTEM PRINCIPLES

The architecture starts with responsibility and inspectable joins

Before products and components, establish the rules by which identities, state, dependencies, versions, determinism, evidence, and deployment remain coherent.

The repeated pattern used throughout the reference.

SYSTEM MODEL · METHOD

Read the architecture by responsibility, not by logo

Products, layers, foundations, entry doors, and deployment postures describe different axes. A coherent review keeps them distinct before drawing their relationships.

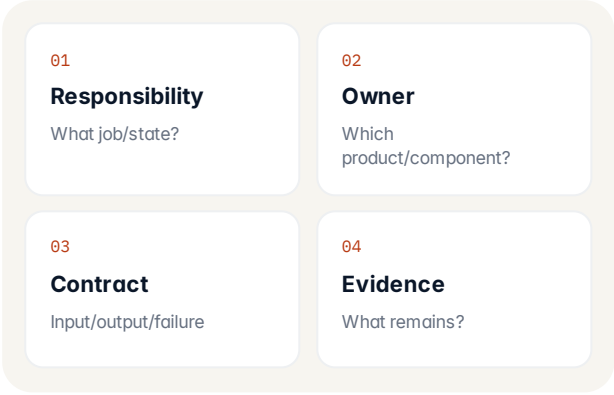
The same component can appear under a product, in the stack, and on an open-foundation description without those labels meaning the same thing. The technical model starts from owned state and contracts.

Mechanism. For each responsibility, identify the component that owns mutable state, the callers it serves, the lower dependencies it may use, and the evidence it emits. Products own operating jobs; foundations implement inspectable mechanisms; layers constrain dependency direction.

Contract. A system map records component/version, input/output schemas, state store, authority boundary, availability status, deployment placement, and replacement surface. Commercial inclusion and source rights remain separate attributes.

Failure and recovery. If two boxes both appear to own the same state, or no box owns a transition, stop the review. Duplication becomes drift; an unowned join becomes manual reconciliation and an incomplete audit trail.

Evidence and acceptance. Walk one work identifier through the map and retrieve every referenced artefact. Remove a component and identify the explicit contract that breaks rather than inventing an implicit dependency.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can the team name one owner for every mutable state and one versioned contract for every material crossing?

SYSTEM MODEL · DEPENDENCY DIRECTION

Downward dependencies are an operational constraint

The stack page presents higher responsibilities as consumers of lower substrate contracts. That direction limits how changes and failures can propagate.

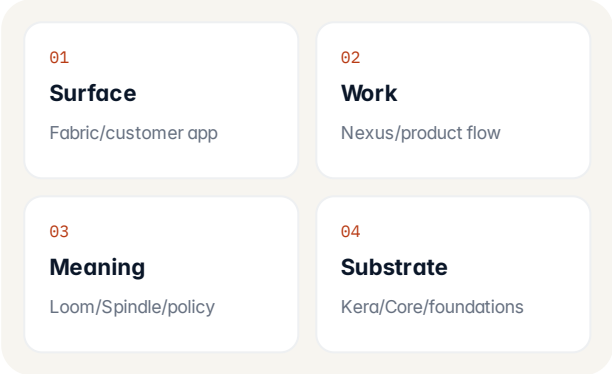
A layer boundary is useful only when a higher layer can be changed without rewriting the lower one and a lower implementation can be replaced without changing the higher meaning.

Mechanism. Maintain a directed dependency manifest from surface and workflow responsibilities toward cognition, knowledge, policy/evidence, compiler/runtime, and compute foundations. Cross-layer callbacks use declared event or capability interfaces rather than shared internals.

Contract. Every dependency declares semantic version, accepted artefact/schema range, capability negotiation, optional/required status, timeout and failure mapping, determinism effect, evidence returned, and compatibility tests.

Failure and recovery. A circular dependency can turn an isolated outage into a platform outage and make startup, update, and air-gap packaging ambiguous. A hidden dependency can invalidate no-outbound and replay claims.

Evidence and acceptance. Generate the dependency graph from the actual builddeployment page manifest, compare it with the published map, sever optional services, and prove the declared degraded or fail-closed behaviour.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can a reviewer derive startup order, failure blast radius, update compatibility, and offline package closure from one manifest?

SYSTEM MODEL · CONTEXT

The system context has more actors than “user and model”

Users, applications, source systems, models, agents, tools, operators, reviewers, auditors, and affected people cross different trust and authority boundaries.

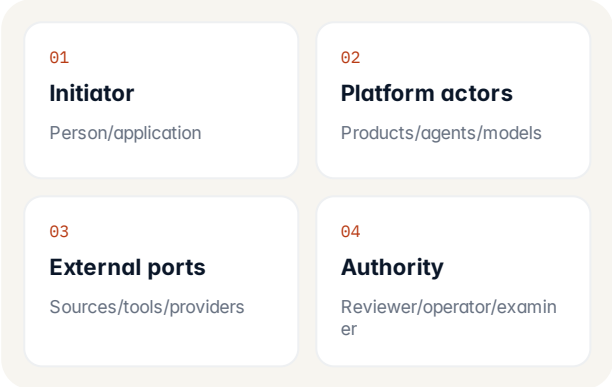
A two-party AI diagram hides the identities that select sources, grant capabilities, approve results, operate infrastructure, receive side effects, and later inspect evidence.

Mechanism. Model each actor as a port with purpose, identity, credentials, permitted objects, capabilities, data classes, authority ceiling, expected artefacts, and revocation path. Treat model providers and tools as external systems even when packaged nearby.

Contract. The context contract distinguishes caller identity from subject identity, human role from agent capability, service authentication from signing identity, and operational access from evidence disclosure.

Failure and recovery. Identity collapse creates confused-deputy paths: a platform service acts with customer authority, an agent inherits administrator scope, or a reviewer gains payload access simply because they can verify integrity.

Evidence and acceptance. Attempt cross-tenant, cross-purpose, expired, revoked, delegated, support, and emergency access. Retain both permitted and denied transitions with the policy/authority version that decided them.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Does every request, action, approval, signature, administration event, and evidence read resolve to the correct identity plane?

SYSTEM MODEL · IDENTITY

A work identity is the spine of the request

The work identifier connects user intent, tasks, artefacts, external actions, human decisions, evidence, and later replay without copying the whole payload everywhere.

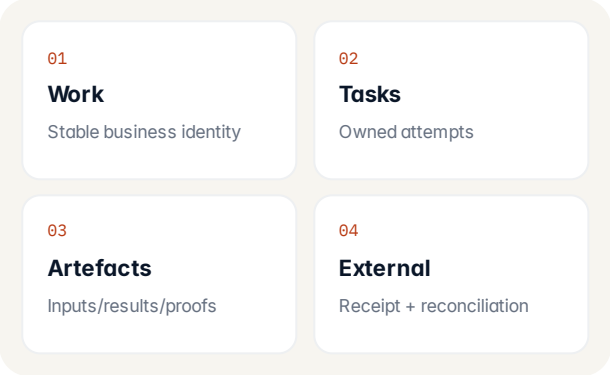
Correlation IDs help operators follow calls. A governed work identity additionally carries purpose, authority, lifecycle, retention, and the relationship between one business case and many technical attempts.

Mechanism. Create the work identity at the ingress boundary; derive task, attempt, artefact, trace, and external-correlation identities beneath it. Append state transitions and supersession links rather than reusing identifiers for different decisions.

Contract. The work envelope includes owner/tenant, initiating actor, subject/reference, purpose, created/effective time, workflow and policy selection, authority ceiling, result state, evidence reference, and idempotency/reconciliation keys.

Failure and recovery. Regenerating identity on retry fragments evidence; sharing one identity across two business decisions merges authority; exposing a predictable identifier leaks existence; losing the external correlation prevents reconciliation.

Evidence and acceptance. Trace a work item through retry, reassignment, human escalation, external write, correction, and replay. Verify every child resolves to one parent and every superseded decision remains distinguishable.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can an authorised examiner start from either the business-system record or platform work record and arrive at the same closed state?

SYSTEM MODEL · MOVEMENT

Control flow and data flow must be drawn separately

An orchestration call may cross the platform while sensitive data stays local, arrives by reference, or is reduced to a purpose-scoped projection.

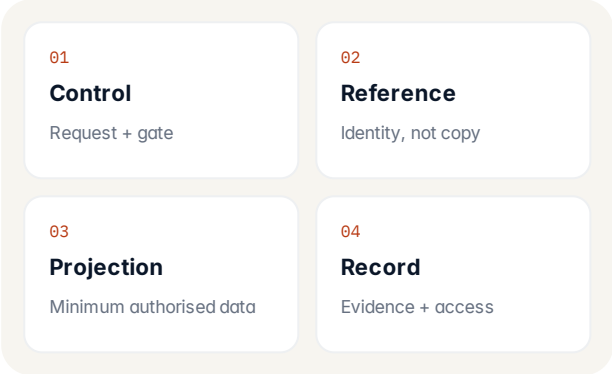
A single arrow labelled “AI request” conceals which component initiated work, which bytes moved, which store retained them, and whether the receiver gained authority over the source.

Mechanism. Draw a control-flow sequence for requests, gates, retries, cancellation, and callbacks. Overlay a data-flow map for payloads, references, embeddings/indexes, prompts, outputs, telemetry, evidence, backups, and support bundles.

Contract. At each hop record source and destination trust zones, identity, purpose, data class, transformation, encryption/key boundary, retention, model/tool/provider exposure, and evidence link.

Failure and recovery. A fallback can silently move data to a different provider; an error log can retain payloads outside policy; a replay service can retrieve more context than the original work; a support export can cross the air gap.

Evidence and acceptance. Instrument a representative workflow, capture allowed network/storage paths, compare them with the declared map, disable undeclared egress, inspect retained stores, and exercise deletion/export.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can the team answer where every sensitive byte can appear without relying on the high-level deployment label?

SYSTEM MODEL · REPLACEABILITY

Typed contracts make replacement a real property

A replaceable component is not one with an adapter slide. Callers must depend on stable meaning, explicit capabilities, and testable failure semantics.

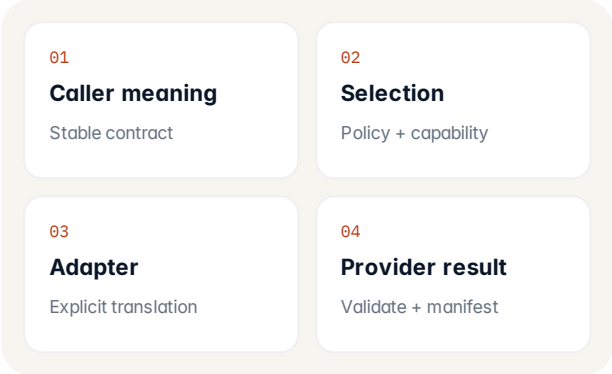
Model, tool, storage, backend, compiler-target, and deployment changes are normal. The platform claim is credible when these changes are visible and bounded rather than disguised as identical.

Mechanism. Define a semantic interface above provider-specific protocols. Negotiate capabilities, select an implementation by policy, record the chosen provider/version, translate inputs explicitly, validate outputs, and keep provider-specific artefacts behind a declared extension field.

Contract. Specify required/optional operations, types, determinism mode, streaming/order, errors, retries, limits, security classification, evidence, compatibility, and behaviour when no implementation satisfies the request.

Failure and recovery. Lowest-common-denominator interfaces can erase necessary capability; permissive translation can fabricate defaults; silent fallback changes data boundary or result; incompatible errors trigger unsafe retry.

Evidence and acceptance. Run the same acceptance corpus through two approved implementations, compare contract invariants, preserve declared variation, deny an unsupported capability, and inspect the manifest change.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can the component be replaced without changing the business contract, while still making every implementation difference visible?

SYSTEM MODEL · VERSIONS

Versioning is part of state, not release administration

Sources, policies, models, experts, solvers, agents, workflows, schemas, foundations, compiler plans, runtimes, keys, and deployment packages can all change a result.

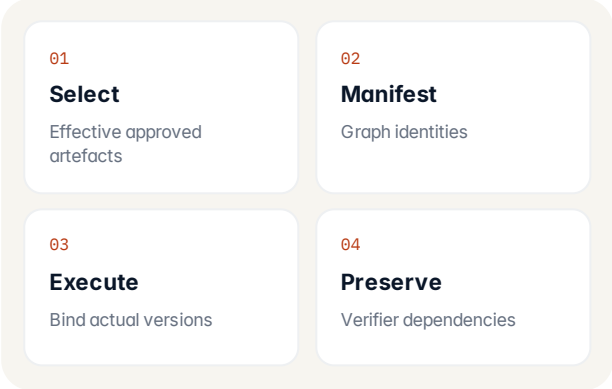
A platform version alone is insufficient for historical interpretation. The relevant configuration is a graph of artefact identities and compatibility edges.

Mechanism. Build a work manifest at execution time from immutable/content-addressed identities where possible and signed release/version identifiers otherwise. Record selection policy and effective time, not just what files happened to be installed.

Contract. Each artefact declares kind, owner/publisher, version/hash, schema, dependencies, effective/expiry interval, compatibility, signature/trust root, retrieval location, and retention requirement.

Failure and recovery. Mutable tags, overwritten source, expired download, lost key, missing schema, automatic model update, or incompatible verifier can make a historic trace intact but uninterpretable.

Evidence and acceptance. Retrieve an old manifest in an isolated environment, resolve every required artefact or retained substitute, verify signatures, execute or verify the supported path, and document irreproducible dependencies.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Does the retained manifest describe what actually ran, and can every identity needed for the promised historical check still be resolved?

SYSTEM MODEL · RUNTIME CONTRACTS

Three determinism modes serve different work

Hard, seeded, and creative behaviour should be requested and recorded explicitly. One vague “deterministic AI” label cannot cover them all.

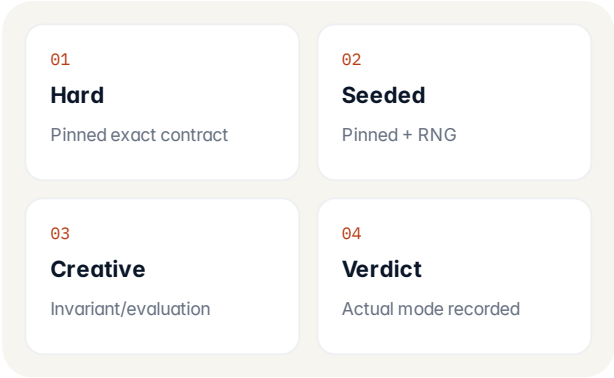
Exact policy, simulation, accounting, and verification work may require bit- or contract-identical results. Search/routing may be reproducible under a fixed seed and catalogue. Generative expression may permit bounded variation.

Mechanism. Bind hard mode to supported operations, numeric profiles, ordering, target contracts, artefact versions, and verification. Bind seeded mode additionally to RNG/seed and deterministic selection. Bind creative mode to invariants, safety/authority gates, and evaluation criteria rather than exact text.

Contract. The request declares mode and comparison rule; the result returns actual mode, pinned dependencies, seed when relevant, target/runtime, variation warnings, and proof/replay support.

Failure and recovery. An unsupported operator, provider, race, clock, external search result, unstable iteration order, floating backend, or changed context can violate replay. The system must degrade explicitly or refuse, not silently relabel.

Evidence and acceptance. Run repeated and cross-target tests, compare bytes or declared invariants, change one dependency, inspect the detected mismatch, and ensure a creative result is never advertised as bit-identical.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can every workload say what “same” means before execution and receive an honest verdict afterward?

SYSTEM MODEL · EVIDENCE FIRST

Evidence is produced by the work, not assembled after it

The architecture connects source, rule, model, agent, tool, human, action, and result as the workflow advances.

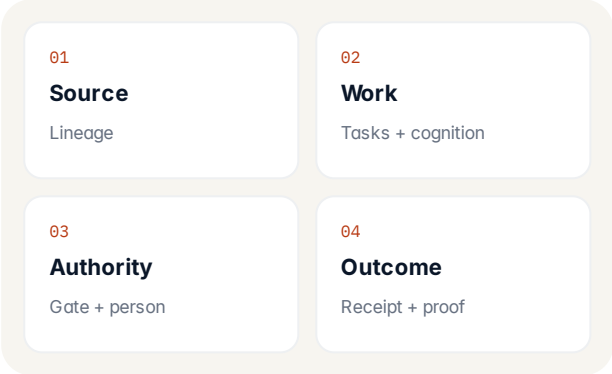
After-the-fact reconstruction depends on logs, screenshots, memory, and current configuration. It often cannot recover the exact source or authority state that existed at decision time.

An intelligence system needs more than a message history. Loom separates memory by purpose. Retrieval is scoped to the task and the active cognitive thread. The memory system can consolidate, decay, forget, replay, and inspect its own state without turning memory into an unbounded transcript.

Contract. Evidence items carry work identity, type/schema, producer, subject, time/order, input/output references, version manifest, integrity fields, access classification, retention, and links to prior/parent material.

Failure and recovery. Asynchronous evidence can arrive late, fail, contain sensitive payload, or refer to a state rolled back elsewhere. The workflow needs an evidence-completeness policy and may have to fail closed before consequential publication/action.

Evidence and acceptance. Select a completed case, retrieve the packet without engineers reconstructing it, verify integrity offline, compare authority and target state, and identify which questions remain outside technical proof.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

What consequential result can currently leave the platform without its declared minimum evidence packet, and why?

SYSTEM MODEL · POSTURE

Deployment invariants need installed tests

Managed Fabric access, licensed customer operation, and air-gapped deployments move operational responsibility. The logical product and evidence contracts should remain recognisable.

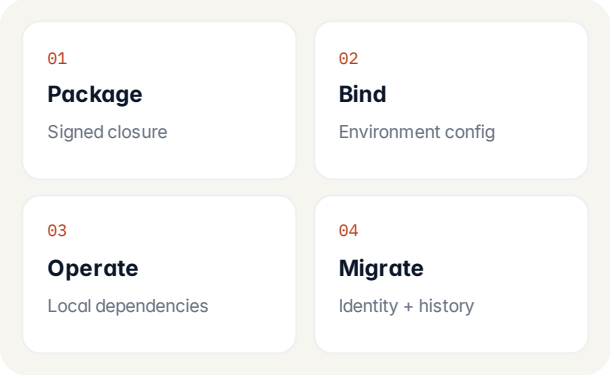
"Same binaries" and "no replatforming" are strong public statements. They become technical properties only when packaging, configuration, dependencies, data, keys, updates, and historical records survive the move.

Mechanism. Maintain one signed component manifest and posture-specific deployment descriptors. Externalise environment bindings such as identity, keys, endpoints, storage, model packages, network policy, observability, backups, and licence validation.

Contract. The posture specification names operator, infrastructure, trust boundary, egress, administrative access, key custody, dependencies, update/rollback, support path, recovery, evidence stores, export, and migration mapping.

Failure and recovery. A managed-only control plane, online licence check, external model, telemetry endpoint, time service, package registry, or support dependency can break licensed/air-gap claims. Hidden customer duties can break operation even when binaries start.

Evidence and acceptance. Install from signed media, sever external connectivity, run the workflow and verifier, restore locally, update and roll back, migrate one history and one new case, and reconcile both states.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

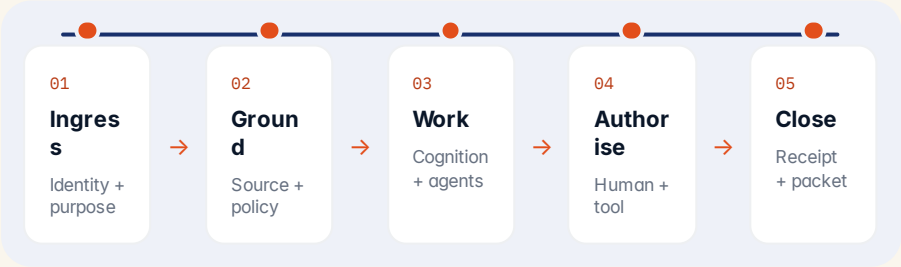
Can the chosen workflow cold-start, run, prove, recover, update, and roll back within the declared posture boundary?

02

REQUEST LIFECYCLE

Follow one work identity from ingress to history

The end-to-end sequence makes source selection, cognition, exact work, agents, tools, humans, external systems, closure, and replay one inspectable system.



Control and evidence move with the work; sensitive data may remain a scoped reference or projection.

REQUEST LIFECYCLE · INGRESS

Ingress validates identity, purpose, and workload shape

The platform should reject ambiguity before expensive cognition, agent work, or external action begins.

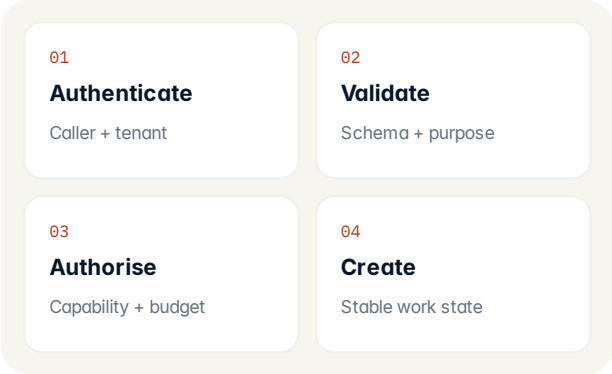
A raw prompt is not an enterprise request. It lacks stable identity, subject/purpose separation, source selection, authority, expected result, and failure semantics.

Mechanism. Authenticate the caller, resolve tenant/workspace, validate work/idempotency identifiers, classify purpose and data, select the workflow contract, check capability and budget, and reject unknown or conflicting fields.

Contract. Ingress returns accepted work identity and initial state, or a typed rejection covering authentication, authorisation, schema, purpose, duplication, limits, unavailable capability, and policy precondition.

Failure and recovery. Permissive defaults can choose the wrong tenant, current rather than effective policy, broader source scope, or execute an already completed request. Retries without idempotency can create a second case.

Evidence and acceptance. Retain the validated envelope, identity and policy decisions, rejected fields, deduplication result, selected workflow/version, and the caller-visible acceptance or refusal.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Send missing, stale, duplicated, over-broad, cross-tenant, and unsupported requests and verify none enter hidden partial work.

REQUEST LIFECYCLE · CONTEXT

Context projection is an access-control operation

The context a component receives should be the minimum authorised projection for its task, not a copy of the workspace or case.

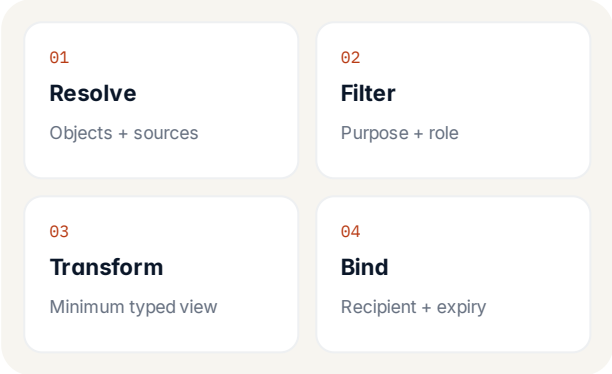
Source access by the orchestrator does not imply every model, agent, solver, tool, or reviewer may see the same fields. Purpose and role can narrow the same underlying object differently.

Mechanism. Resolve object and source rights, field/purpose policy, effective versions, confidentiality, model/provider boundary, task need, and retention before building a typed context projection with references back to origin.

Contract. The projection identifies work/task, subject/reference, purpose, included fields or atoms, excluded classes, source/version, transformations, recipient capability, expiry, and provenance.

Failure and recovery. Caching a broad projection, reusing it after role or purpose changes, prompt/log persistence, provider fallback, or downstream forwarding can bypass the original access decision.

Evidence and acceptance. Record projection policy and content identity without unnecessarily duplicating sensitive values; test forbidden fields, role change, revocation, provider switch, and replay under the historical policy.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can the platform prove what each participant was allowed to see and prevent later reuse outside that purpose?

REQUEST LIFECYCLE · GROUNDING

Source selection binds authority and effective time

Retrieval begins after the organisation decides which source may answer which question and which version applied at the event time.

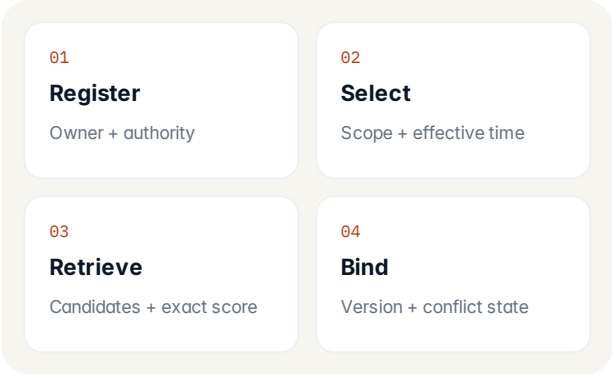
Similarity can find related text but cannot decide whether a draft, superseded policy, local note, foreign jurisdiction, or unauthorised copy is controlling.

The approval packet is created as the work moves, not rebuilt later when someone asks. Source, rule, owner, and a replay path attach while the decision is made, so a regulator question is a retrieval, not a reconstruction.

Contract. Return source/version identities, authority metadata, effective/expiry state, selection rule, query, retrieved atoms/passages, scores, conflicts, transformations, and exclusion reasons.

Failure and recovery. Missing owner, ambiguous precedence, withdrawn source, stale index, version mismatch, insufficient coverage, or source conflict should stop or escalate the dependent decision rather than be hidden by rank.

Evidence and acceptance. Reconstruct selection at the historical event time, show included and material excluded candidates, verify lineage to source bytes, and change the policy to prove a new decision gets a new binding.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can a reviewer answer not just “what was cited?” but “why was this the authorised source for this case then?”

REQUEST LIFECYCLE · REASONING

Cognitive work and exact work use different contracts

Interpretation, search, synthesis, and expression can coexist with deterministic rules, arithmetic, simulation, and optimisation without blurring their guarantees.

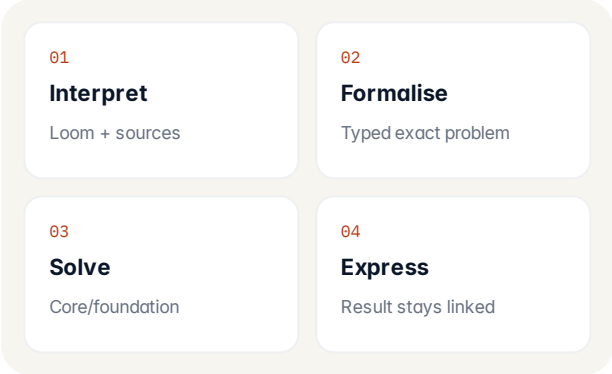
A language model should not improvise exact cents, constraints, eligibility thresholds, safety equations, or a reproducibility claim. A solver should not be asked to infer meaning the source does not formalise.

Mechanism. Loom decomposes the task, uses authorised knowledge, routes specialists, and calls typed solvers or foundations for exact subproblems. The solver receives declared inputs/profile and returns result, validity state, manifest, and evidence.

Contract. The hand-off defines problem type, units/types, constraints, source references, numeric/determinism profile, target, timeout, expected proof/verification, and how invalid/unknown results propagate.

Failure and recovery. Unit mismatch, incomplete constraint, unsupported operation, overflow, solver timeout, invalid certificate, or cognitive summary that changes the exact result must be detected before expression or action.

Evidence and acceptance. Retain the decomposition, exact input/output, solver/runtime identity, validation/proof, and expression mapping. Re-run the solver independently and compare the natural-language statement with its artefact.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can the exact artefact be verified without the prose, and can the prose be traced without asking the solver to explain business meaning?

REQUEST LIFECYCLE · MODELS

Expert routing is selection under a manifest

A sparse specialist architecture still needs catalogue, candidate, routing, constraint, and merge identities to make a result inspectable.

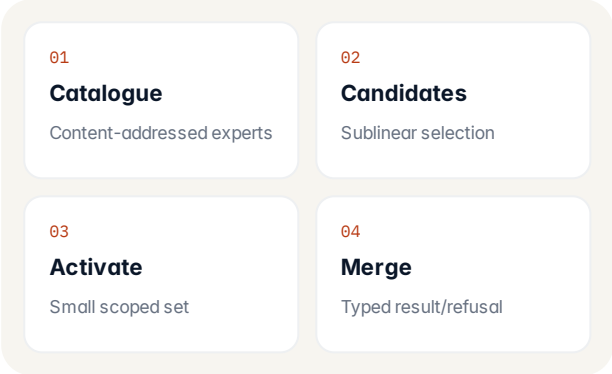
Naming a model is not enough when a request activates a subset of experts and may combine their artefacts with search, memory, or solvers.

Mechanism. Pin the catalogue and cluster signatures, compute or retrieve the query representation, build a candidate set, score/select the activated experts, enforce cluster constraints, run their work, challenge or merge typed artefacts, and express the result.

Contract. Record catalogue/version, router/version, query/context identity, candidate/activated set, scores or PAP breakdown where supported, constraints, per-expert artefacts, merge policy, seed/mode, and output.

Failure and recovery. Catalogue drift, missing expert, unstable tie order, constraint conflict, invalid expert artefact, over-broad activation, merge disagreement, or fallback model must remain explicit.

Evidence and acceptance. Repeat with fixed artefacts/seed, inspect the activated set, add an expert as a versioned change, force conflict, compare result and routing, and preserve the old catalogue for replay.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Does a replay pin the actual expert catalogue and routing state, or only the top-level model name?

REQUEST LIFECYCLE · TEAMWORK

Agent collaboration needs typed artefacts, not shared prose

Teams become accountable when every hand-off names an expected artefact, owner, validation, disagreement, and recovery path.

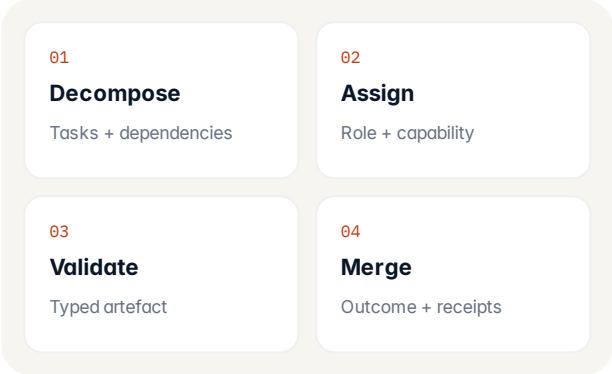
A multi-agent transcript can show messages while leaving the objective owner, task completion, authority, and final merge ambiguous.

Mechanism. Decompose the objective into tasks with dependencies and contracts; assign roles/capabilities; validate each returned artefact; retain disagreement; route challenge/review; gate action; and dissolve the team after one outcome.

Contract. Task state includes owner, inputs, dependencies, expected schema/quality, permitted tools, deadline/budget, policy, attempts, artefact identity, validator, next transition, and contribution to result.

Failure and recovery. Invalid or missing artefact, cyclic dependency, two owners, orphaned hand-off, duplicate execution, lost dissent, worker substitution, or failed final merge triggers retry/reassignment/escalation/stop under policy.

Evidence and acceptance. Inspect task graph and passports, force a participant failure and disagreement, verify recovery and authority, and confirm the final record links each material claim/action to a validated contribution.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Could an examiner tell who owned every task and why a rejected contribution did or did not affect the result?

REQUEST LIFECYCLE · ACTION

Tool execution crosses from proposal into effect

A tool call is where internal reasoning can change an external system. Capability, approval, idempotency, containment, and reconciliation meet here.

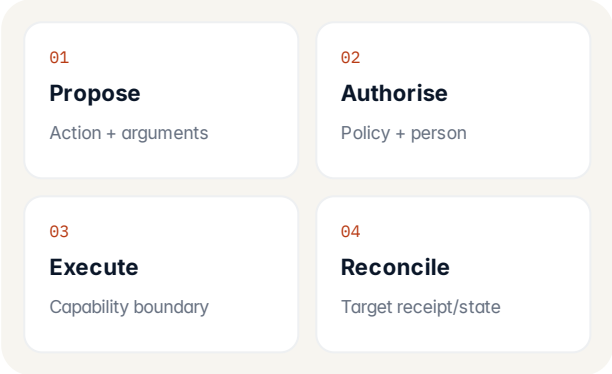
A plausible plan is not authority. Even a successful HTTP response is not necessarily an accepted or durable business action.

Mechanism. Resolve tool identity and schema, validate arguments and data class, check agent/user capability and policy, obtain required human approval, execute through a constrained boundary, capture result/receipt, and reconcile target state.

Contract. The action envelope includes work/task, requesting identity, tool/version, operation, argument commitment, purpose, authority state, limits, idempotency key, timeout/retry, expected receipt, and reconciliation rule.

Failure and recovery. Denied permission, malformed argument, sandbox limit, timeout after possible effect, duplicate, partial target update, stale approval, target conflict, or receipt mismatch requires a state other than “tool failed”.

Evidence and acceptance. Retain request, gate, human decision, execution transcript or action receipt, target response, follow-up query/reconciliation, retries, and final authoritative state.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can a timeout be resolved without either losing a completed action or executing it twice?

REQUEST LIFECYCLE · AUTHORITY

Human review is a state transition with information needs

A person remains meaningfully in control only when they can see the basis, choose real alternatives, and understand the consequence of the transition.

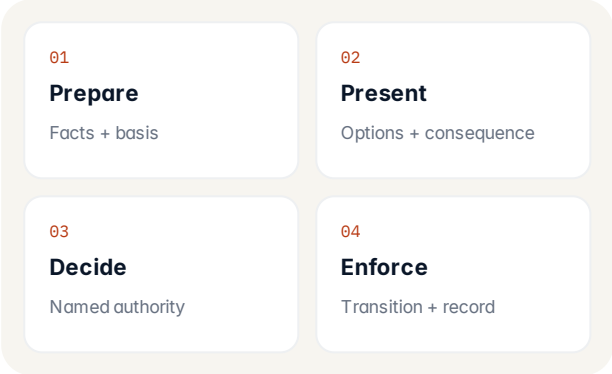
A generic approval button after an opaque answer turns the human into a ceremonial checkpoint and makes later accountability misleading.

Mechanism. Present material facts, source/rule versions, uncertainty/conflict, proposed result/action, affected object, limits, and available choices. Bind the reviewer identity and scope, enforce separation/expiry, and record reason plus transition.

Contract. Review request and response declare work/state/version, reviewer authority, information shown, permissible decisions, correction/escalation path, expiry, delegation rule, and downstream effect of each choice.

Failure and recovery. Reviewer absent, wrong role, stale information, expired approval, changed source/result after review, self-approval, bulk rubber-stamping, UI ambiguity, or downstream bypass should stop or reopen the gate.

Evidence and acceptance. Recreate the review screen from retained references, verify what the person could see and do, test source change and expiry, and compare approval state with the executed action.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Could the reviewer reasonably detect a bad recommendation and choose correction, rejection, or escalation before effect?

REQUEST LIFECYCLE · OUTCOME

Publication and execution are separate closure points

A result can be generated, reviewed, delivered, accepted by another system, communicated to an affected person, and finally closed at different times.

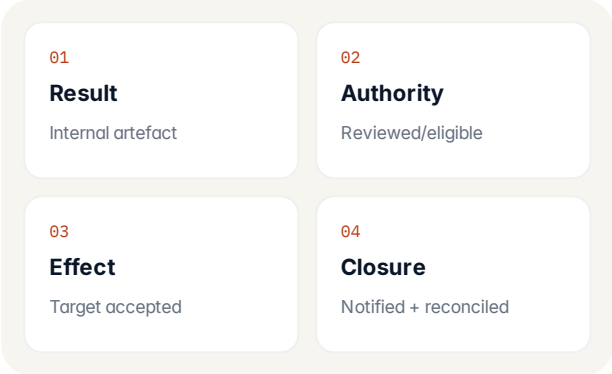
Collapsing those states causes false success, lost notices, duplicate side effects, and evidence packets that disagree with the business record.

Mechanism. Model internal result, review, action eligibility, execution attempt, target acceptance, reconciliation, notification, contestability window, correction, and closed/superseded state separately.

Contract. Every transition includes prior version, actor, time, required receipt, retry/idempotency behaviour, customer-system correlation, visibility, and evidence-completeness requirement.

Failure and recovery. Output formatting failure, notification bounce, target rejection, eventual-consistency delay, duplicate callback, correction after closure, or proof packet lag must not masquerade as one generic error.

Evidence and acceptance. Compare platform state, target state, communication delivery, human decision, and packet manifest; seed mismatches and prove the reconciliation queue exposes them.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Which state is the organisation legally or operationally relying on, and how is that state reconciled with the platform?

REQUEST LIFECYCLE · HISTORY

Replay reconstructs a declared contract, not the past universe

Meaningful replay pins or preserves the inputs and artefacts that affect a supported result, then distinguishes re-execution from verification and simulation.

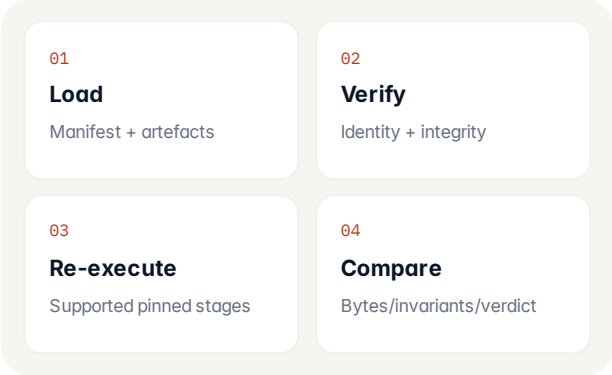
External sources, providers, clocks, people, physical systems, and creative generation may not be reproducible. The replay mode must state what is held constant and what is substituted.

Mechanism. Load the work manifest, verify identities and integrity, resolve retained source/context, policy, model/expert, solver, workflow, tools or transcripts, compiler/runtime, target, seed and keys, then execute or verify supported stages.

Contract. Return replay mode, dependency resolution, original and replay environment, comparison rule, stage verdicts, divergences, unsupported substitutions, and final relationship to the original outcome.

Failure and recovery. Missing artefact, unavailable verifier, provider drift, external side effect, key loss, schema incompatibility, unsupported target, or creative variation yields partial/unverifiable, not a fabricated success.

Evidence and acceptance. Replay one historic deterministic path offline, verify one proof without execution, simulate one external action without effect, and document every dependency not reproduced.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Does the replay result make its scope and missing dependencies explicit enough that a reviewer cannot confuse verification with a rerun?

09	**Kera**	the graph-native language, content-addressed IR, compiler, planner, and target emission path
08	**Core**	the binary AI framework and runtime operation matrix
07	**Charter**	the organisation's single governed operating graph
06	**Mesh**	distributed capacity and privacy-aware workload execution across an estate
05	**Aura**	governed software-development work through a goal, plan, tools, models, quality, and review loop
04	**Spindle**	the governed lifecycle from source material to reusable knowledge atoms
03	**Nexus**	executable organisations formed around an objective
02	**Loom**	cognition: perception, memory, search, reasoning, exact solver use, and expression
01	**Fabric**	the workspace and the stable work object

A responsibility register, not a UI menu or commercial entitlement statement.

FABRIC ARCHITECTURE · OWNERSHIP

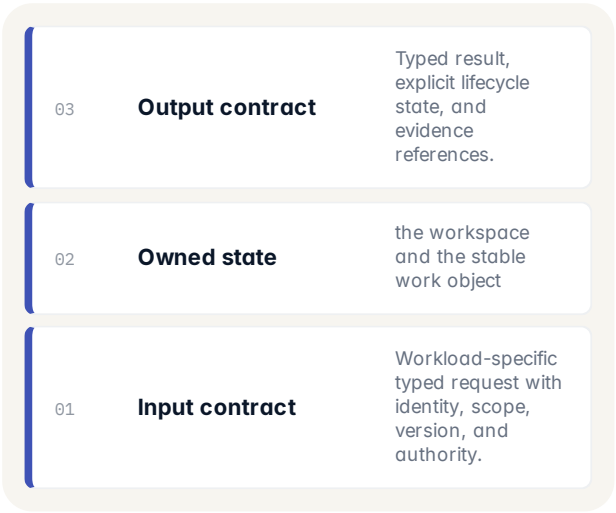
Fabric: responsibility, state, and boundary

Fabric owns the workspace and the stable work object. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes workspace graph, participants, roles, conversations, governed objects, model and agent registries, knowledge references, flow graphs, budgets, and the tenant audit chain. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is a user or API request with workspace identity, purpose, context, selected objects, and an authority ceiling. The principal output is a workspace-visible result with its source, participant, decision, and record references still attached. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Fabric owns the surface and workspace object graph. It does not absorb Loom cognition, Nexus organisation, Spindle source governance, customer policy authority, or the deployment operator's duties.



The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Fabric from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

FABRIC ARCHITECTURE · FLOW

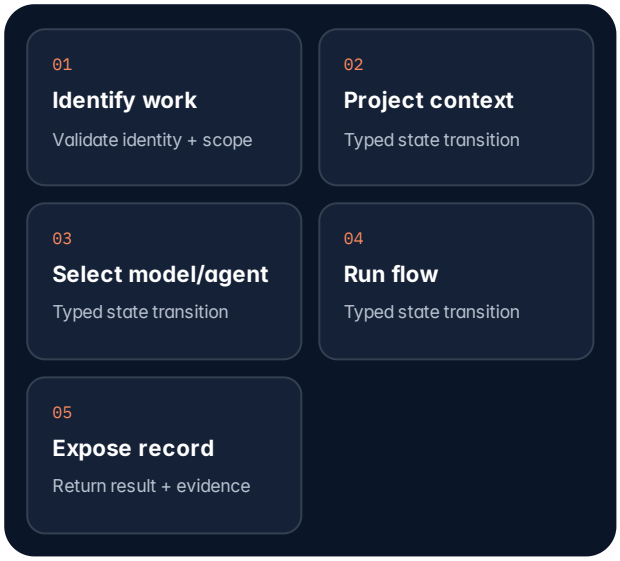
Fabric: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

The website describes one work object shared by dashboard, REST, WebSocket, and flows; workspace-scoped keys and versioned contracts write to one tenant audit chain. Participant ports distinguish people, models, agents, teams, and tools so a provider swap does not redefine the work.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Integration surfaces include Browser, REST, WebSocket, workspace-scoped keys, versioned objects, flow triggers, extensions, shared links, and the lower product contracts exposed through the workspace.. For each selected surface, document authentication and authorisation, tenancy or workload scope, schema/version negotiation, payload and reference limits, streaming or ordering behaviour, idempotency, error taxonomy, rate/resource limits, observability, and the point at which an external side effect becomes authoritative.



Conceptual Fabric path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

FABRIC ARCHITECTURE · ASSURANCE

Fabric: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Reject or surface missing workspace scope, invalid participant capability, unavailable model or agent, inaccessible object, exhausted budget, failed flow node, and inconsistent record linkage. The work object should remain stable across retries and provider changes. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain work identity, participant identity and role, projected context, selected source and object versions, model or agent selection, flow state, decision or approval, output, share state, and audit-chain reference. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Create the same work through UI and API; swap an approved model; add and revoke a participant; run a saved flow; share only the intended answer/source subset; replay the audit chain; and confirm the work identity remains one object. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

FABRIC ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Fabric owns the surface and workspace object graph. It does not absorb Loom cognition, Nexus organisation, Spindle source governance, customer policy authority, or the deployment operator's duties.

LOOM ARCHITECTURE · OWNERSHIP

Loom: responsibility, state, and boundary

Loom owns cognition: perception, memory, search, reasoning, exact solver use, and expression. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes content-addressed expert catalogue, domain clusters, routing signatures, constraint sets, corpus snapshots, memories, retrieved knowledge references, reasoning artefacts, solver manifests, seed, and expression state. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is a typed question or cognitive task, authorised context projection, source references, constraints, requested mode, and determinism parameters. The principal output is an answer or typed refusal with source references, expert/constraint identity, solver artefacts where used, and trace material. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Loom produces cognitive artefacts. It does not own team staffing and task recovery, source certification, human authority, downstream side effects, or the numeric/runtime implementation beneath exact work.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	cognition: perception, memory, search, reasoning, exact solver use, and expression
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Loom from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

LOOM ARCHITECTURE · FLOW

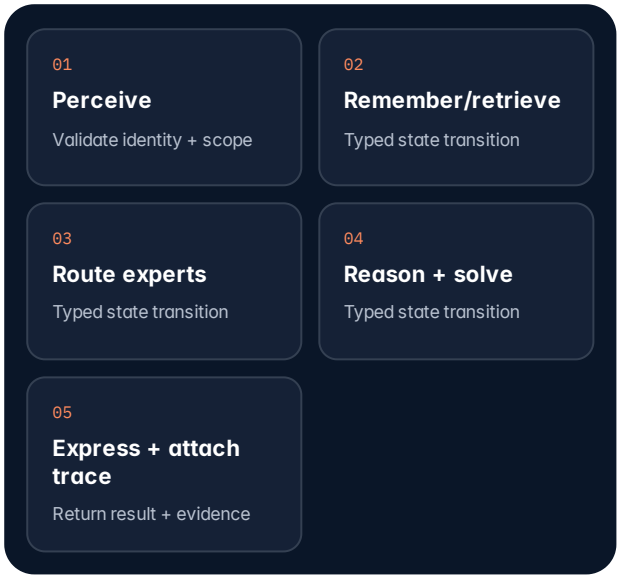
Loom: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

The route separates four weaves, perception, memory/search, reasoning/solvers, and expression. Experts carry training corpus, knowledge graph references, constraint sets, and a content-addressed version. Routing activates a small subset rather than merging every specialist into one global model.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Provider SDKs want to own the conversation, message format, tools, memory, streaming protocol, agent schema, and execution history. That is convenient for one integration and expensive for a platform. Fabric defines the durable system above those providers. The workspace owns the object graph. The work owns the context. Models, agents, teams, solvers, tools, and people participate through typed contracts. The UI, REST API, WebSocket stream, flow runtime, knowledge layer, model registry, and activity view operate on the same domain.



Conceptual Loom path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

LOOM ARCHITECTURE · ASSURANCE

Loom: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Surface conflicting constraints, insufficient source coverage, invalid solver output, unavailable expert, routing uncertainty, unsupported replay conditions, and expression that cannot remain faithful to the reasoning artefact. Do not average a hard conflict into fluent prose. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain expert catalogue and activated set, source and memory references, constraints, seed and routing state, reasoning and solver artefacts, expression result, refusal or uncertainty state, and the linked proof certificate where the published claims one. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Pin catalogue, source set, constraints, seed, solver and runtime; repeat the inference in supported environments; inspect activated experts and PAP/routing breakdown; force a constraint conflict; verify the certificate; and distinguish architectural claims from benchmark-specific numbers. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

LOOM ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Loom produces cognitive artefacts. It does not own team staffing and task recovery, source certification, human authority, downstream side effects, or the numeric/runtime implementation beneath exact work.

NEXUS ARCHITECTURE · OWNERSHIP

Nexus: responsibility, state, and boundary

Nexus owns executable organisations formed around an objective. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes objective, mission, team shape, roles, capabilities, task graph, dependencies, collaboration sockets, four memory classes, policy checkpoints, human gates, recovery state, and work receipts. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is an objective with acceptance conditions, source authority, available capabilities, policy, action limits, and named human authority. The principal output is one accountable outcome assembled from validated task artefacts, with organisation state, approvals, failures, and receipts retained. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Nexus owns organisational execution. It does not pretend a role is a model, rewrite a Loom result, determine source legitimacy, grant itself authority, or collapse the customer's business transaction into an internal task state.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	executable organisations formed around an objective
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Nexus from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

NEXUS ARCHITECTURE · FLOW

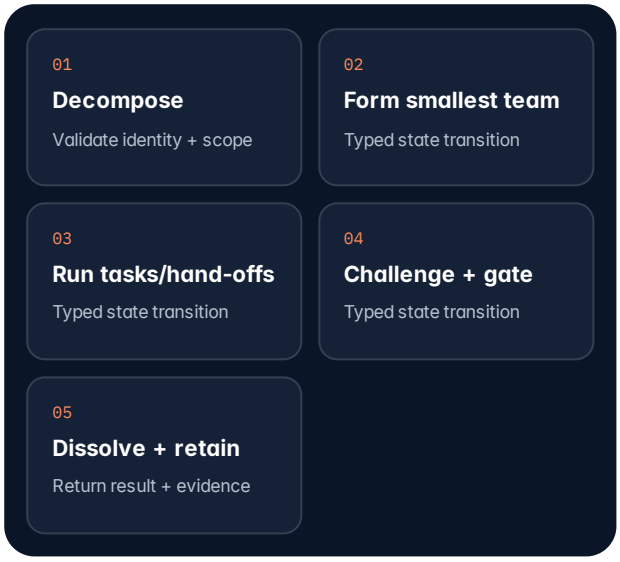
Nexus: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

Nexus treats roles and team shapes as runtime structure rather than assistant personas. A task has an owner, inputs, expected artefact, dependencies, permitted tools, policy checks, state, retry or escalation path, and contribution to the objective. Typed hand-offs preserve disagreement.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Integration surfaces include Fabric flows, Loom cognitive tasks, Spindle sources, customer tools, typed message/event surfaces, policy gates, human review, Ledger/Knot/Selvedge evidence, and authoritative-system reconciliation.. For each selected surface, document authentication and authorisation, tenancy or workload scope, schema/version negotiation, payload and reference limits, streaming or ordering behaviour, idempotency, error taxonomy, rate/resource limits, observability, and the point at which an external side effect becomes authoritative.



Conceptual Nexus path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

NEXUS ARCHITECTURE · ASSURANCE

Nexus: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Contain unavailable or invalid participants, rejected artefacts, failed hand-offs, source conflict, tool denial, model outage, reviewer absence, budget exhaustion, duplicate work, and unreconciled side effects. Recovery may retry, substitute, reassign, checkpoint, escalate, or fail closed. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain objective and team formation, role/capability assignments, task transitions, artefact contracts, collaboration messages, policy and human gates, tool requests/results, recovery decisions, final merge, dissolution, and work receipt. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Run a multi-role objective with a deliberate disagreement, invalid artefact, denied tool, missing reviewer, and downstream timeout; verify task ownership, recovery, no duplicated side effect, retained dissent, final authority, and one closed receipt. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

NEXUS ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Nexus owns organisational execution. It does not pretend a role is a model, rewrite a Loom result, determine source legitimacy, grant itself authority, or collapse the customer's business transaction into an internal task state.

SPINDLE ARCHITECTURE · OWNERSHIP

Spindle: responsibility, state, and boundary

Spindle owns the governed lifecycle from source material to reusable knowledge atoms. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

Every decision Dweve makes, every output Loom generates, every action Nexus takes, every fact Spindle ingests, leaves a content-addressed proof trace. The trace is cryptographically anchored, digitally signed, and replayable bit-for-bit on any machine. When your DPIA, your DORA incident response, or your internal-audit team asks "how did the model reach that conclusion?", you hand them the trace and they replay it. No screenshots, no approximations.

The principal input is approved sources with owner, scope, rights, effective version, collection/parse method, transformation policy, quality threshold, and withdrawal rule. The principal output is queryable governed atoms and indexes whose path can be followed backward to source and forward into dependent work. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Spindle makes knowledge governed and reusable. It does not declare a source legally authoritative, decide a policy conflict without an owner, perform general cognition, or convert quality scores into business authority.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	the governed lifecycle from source material to reusable knowledge atoms
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Spindle from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

SPINDLE ARCHITECTURE · FLOW

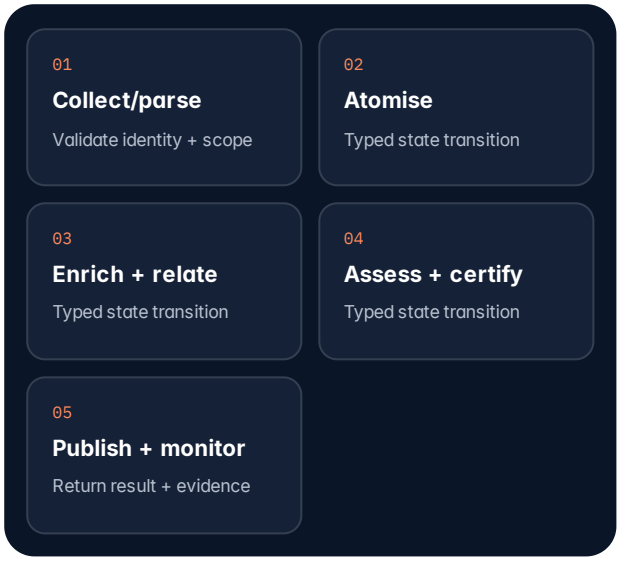
Spindle: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

the published material describes seven stages, a canonical atom model, hierarchy, typed events, multi-store persistence, quality dimensions, bias categories, and forward/backward lineage. Existing systems connect as scoped source classes; atom promotions, threshold breaches, bias flags, and conflicts can emit events.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Integration surfaces include Winnow collection, Bindery/Reed parsing, BitWeave retrieval, Fabric knowledge objects, Loom context, typed events/webhooks, distributed graph, atom store, time series, vector index, and customer repositories.. For each selected surface, document authentication and authorisation, tenancy or workload scope, schema/version negotiation, payload and reference limits, streaming or ordering behaviour, idempotency, error taxonomy, rate/resource limits, observability, and the point at which an external side effect becomes authoritative.



Conceptual Spindle path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

SPINDLE ARCHITECTURE · ASSURANCE

Spindle: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Quarantine unreadable or unauthorised material, preserve parse/transform errors, stop promotion below configured thresholds, expose contradictory sources, retain disputed tags and rebuttal, prevent a withdrawn source from silently feeding new work, and identify dependent atoms and cases. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain source/version identity, collection and parse record, every material transformation, atom and hierarchy versions, quality observations, bias/compliance state, conflict and rebuttal, promotion authority, downstream use, and withdrawal effects. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Ingest one changing and one conflicting source; inspect the atoms and lineage; fail a quality gate; rebut a bias tag; withdraw a version; prove dependent retrieval changes; and retrieve an old case with the old source path intact. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

SPINDLE ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Spindle makes knowledge governed and reusable. It does not declare a source legally authoritative, decide a policy conflict without an owner, perform general cognition, or convert quality scores into business authority.

AURA ARCHITECTURE · OWNERSHIP

Aura: responsibility, state, and boundary

Aura owns governed software-development work through a goal, plan, tools, models, quality, and review loop. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes goal and success condition, plan/DAG, steering, context budget, merged configuration, agent/skill/plugin definitions, tool registry and permissions, model registry, memory, event bus, diffs, quality stages, and audit record. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is a repository-scoped goal, project instructions, resolved configuration, permitted models/tools, success checks, and human steering. The principal output is a reviewable change or analysis with diffs, tool/model history, quality results, decisions, and an audit chain. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Aura governs development work. It does not own the customer repository's business authority, treat a passing test as proof of product correctness, bypass project instructions, or make an external MCP server trusted merely because it connected.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	governed software-development work through a goal, plan, tools, models, quality, and review loop
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Aura from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

AURA ARCHITECTURE · FLOW

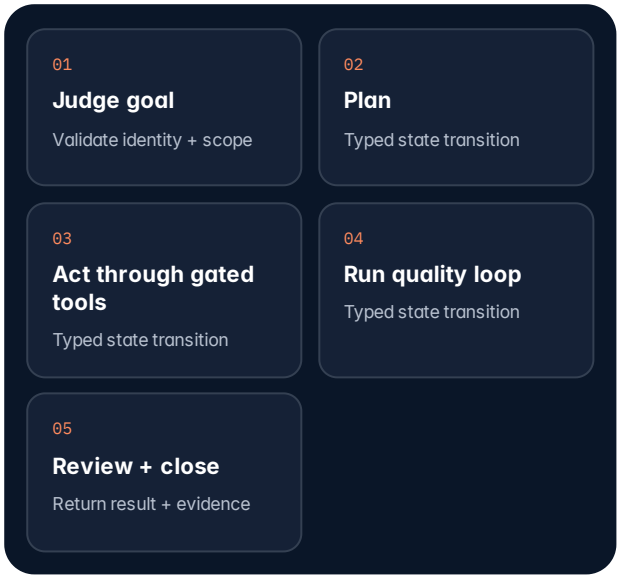
Aura: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

Aura resolves built-in, user, project, and environment configuration layers; markdown/YAML agents and workflows can hot-reload; MCP servers join the typed registry through a capability handshake; built-in and external tools pass the same permission, audit, filtering, and circuit-breaker path.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Integration surfaces include Static binary/TUI, repository files, Rust trait tools, shell and quality tools, MCP over stdio JSON-RPC, typed event bus, model backends, CI, configuration files, and the shared evidence substrate.. For each selected surface, document authentication and authorisation, tenancy or workload scope, schema/version negotiation, payload and reference limits, streaming or ordering behaviour, idempotency, error taxonomy, rate/resource limits, observability, and the point at which an external side effect becomes authoritative.



Conceptual Aura path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

AURA ARCHITECTURE · ASSURANCE

Aura: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Stop on permission denial, invalid plan step, context exhaustion, tool or MCP failure, unsafe output, quality-gate failure, model outage, repeated circuit-breaker event, repository drift, or a goal that cannot be judged complete. Preserve steering and partial work for review. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain resolved configuration and source layers, goal and plan versions, active agent/model, context decisions, tool requests/results, file diffs, events, steering, permission prompts, quality pipeline, security findings, output filters, and closure judgement. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Resolve all four config layers; hot-reload an agent; connect a constrained MCP server; deny a dangerous tool; run the six-stage quality path; steer mid-plan; recover from a failed tool; compare the audit chain with the actual diff; and close only against the goal. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

AURA ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Aura governs development work. It does not own the customer repository's business authority, treat a passing test as proof of product correctness, bypass project instructions, or make an external MCP server trusted merely because it connected.

MESH ARCHITECTURE · OWNERSHIP

Mesh: responsibility, state, and boundary

Mesh owns distributed capacity and privacy-aware workload execution across an estate. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes nodes, authenticated topology, capabilities, workload/job/task graph, scheduler policy, priority and budget, checkpoints, consistency path, federated rounds, privacy budget, proposals, receipts, faults, and settlement state. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is a typed workload with data locality, capability, target, resource, privacy, determinism, deadline, priority, and failure requirements. The principal output is a workload result and receipt that identify placement, participants, budget, faults, aggregation or consensus state, and ownership return. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Mesh owns distributed execution. It does not make a workload lawful, permit data to cross a declared locality boundary, guarantee Byzantine consensus merely because Raft is present, or turn a measured energy comparison into a universal estate saving.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	distributed capacity and privacy-aware workload execution across an estate
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Mesh from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

MESH ARCHITECTURE · FLOW

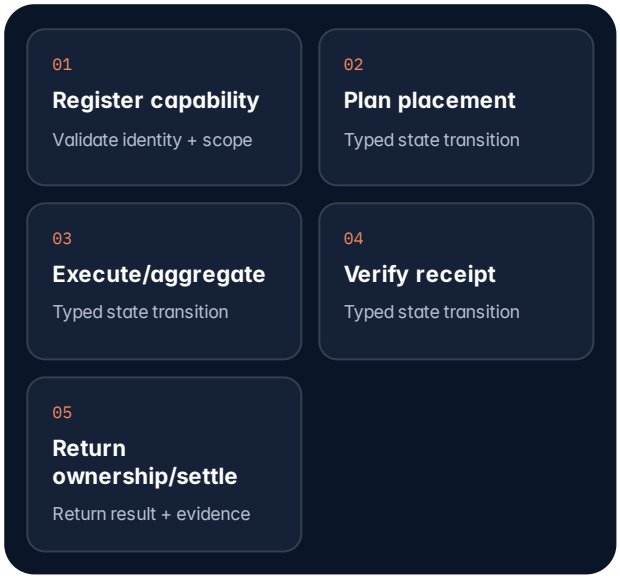
Mesh: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

The route presents peer discovery, scheduling, gRPC and REST job/model/node/metric surfaces, generated Protocol Buffers, bidirectional streaming, operator-tunable task/job limits, strong and eventual consistency paths, and federated privacy mechanisms including HE, secret sharing, ZKP, and tracked differential-privacy budgets.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Integration surfaces include gRPC/REST, generated Protobuf clients, streaming inference, node/operator APIs, Prometheus/OTLP telemetry, Core/Kera workloads, federated learning, customer schedulers, identity and key infrastructure, and receipt consumers.. For each selected surface, document authentication and authorisation, tenancy or workload scope, schema/version negotiation, payload and reference limits, streaming or ordering behaviour, idempotency, error taxonomy, rate/resource limits, observability, and the point at which an external side effect becomes authoritative.



Conceptual Mesh path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

MESH ARCHITECTURE · ASSURANCE

Mesh: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Handle node loss, stale capability, deadline or budget breach, manipulated federated update, quorum loss, partition, failed checkpoint, retry exhaustion, duplicate work, invalid receipt, privacy-budget exhaustion, and result that cannot satisfy the requested determinism or ownership boundary. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain workload and placement plan, node identities/capabilities, scheduler and consistency choice, privacy parameters, task/checkpoint states, input/output commitments, aggregation decisions, faults/retries, resource/cost measurements, receipt, and settlement or ownership return. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Partition nodes, drain one during work, exceed a privacy and resource budget, inject a bad federated update, lose quorum, resume from checkpoint, abort a stream, verify receipts, reconcile duplicated delivery, and show data/local-model material stayed inside the declared boundary. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

MESH ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Mesh owns distributed execution. It does not make a workload lawful, permit data to cross a declared locality boundary, guarantee Byzantine consensus merely because Raft is present, or turn a measured energy comparison into a universal estate saving.

CHARTER ARCHITECTURE · OWNERSHIP

Charter: responsibility, state, and boundary

Charter owns the organisation's single governed operating graph. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes party spine, source tree, domain-engine records, permissions and purpose projection, workflow graph, proposed actions, policy gates, run state, events/outbox, provider registry, country/domain packs, evidence, and lifecycle history. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is a business event or user request bound to a party, purpose, domain, current record version, permitted fields, applicable pack/policy, and authority. The principal output is an updated company record or explicit no-change result with proposal, human authority, event, domain state, and audit linkage. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Charter owns the company record and lifecycle. AI proposes inside the permissions of that graph; it does not self-authorise. A single data model does not remove domain ownership, legal duties, external provider constraints, or the need to isolate tenants and deployments.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	the organisation's single governed operating graph
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Charter from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

CHARTER ARCHITECTURE · FLOW

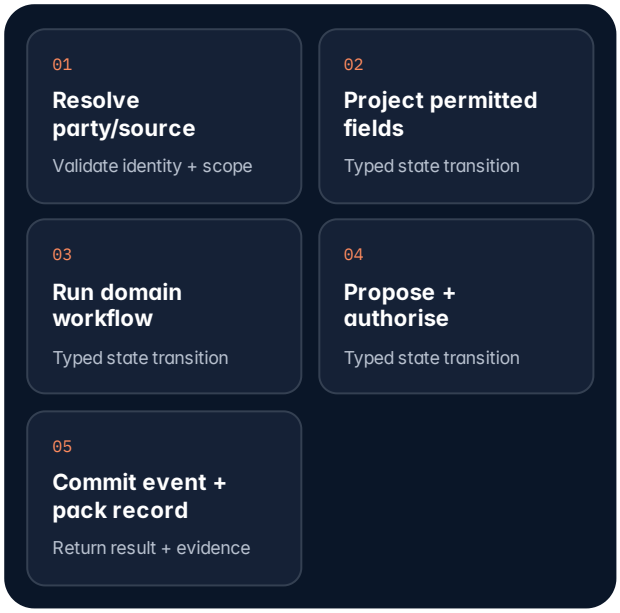
Charter: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

Charter's route describes one party graph seen through multiple domain lenses, four planes on one substrate, field- and purpose-scoped projections, proposed actions that are signed and expire, human authorisation before commit, a run machine, event/outbox fan-out, provider adapters, and portable country/domain packs.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Loom is not a model router with a larger catalogue. It is a runtime for composing heterogeneous cognitive components into one typed execution graph. A node can be a perception transform, a memory operation, a micro model, an embedding or retrieval stage, a reasoning mode, a constraint set, a native or integrated solver, a knowledge operation, a proof checker, a domain expert, or an expression stage. The planner resolves the graph from the task, policy, available capability, evidence requirements, and execution budget. Only the required graph becomes active.



Conceptual Charter path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

CHARTER ARCHITECTURE · ASSURANCE

Charter: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Reject stale record versions, over-broad field projection, missing purpose, invalid pack, expired proposal, unauthorised approval, failed policy gate, duplicate outbox delivery, provider rejection, partial domain transition, erasure conflict, and an event that cannot reconcile with the persistent party state. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain party/source identity, projected fields and purpose, domain/pack/policy versions, workflow/run transitions, assistant proposal, signature/expiry, human decision, committed event, provider receipt, outbox delivery, erasure or correction effects, and audit reference. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Follow one party from first signal through quote, signed acceptance, invoice/project, support, and renewal; attempt a forbidden field read and expired proposal; fail a provider; replay events; reconcile outbox delivery; run a rights request; and export the owned record. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

CHARTER ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Charter owns the company record and lifecycle. AI proposes inside the permissions of that graph; it does not self-authorise. A single data model does not remove domain ownership, legal duties, external provider constraints, or the need to isolate tenants and deployments.

CORE ARCHITECTURE · OWNERSHIP

Core: responsibility, state, and boundary

Core owns the binary AI framework and runtime operation matrix. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes model/graph representation, operation and algorithm identity, datatype and width, packing, accumulator, numeric profile, backend and instruction set, dispatch plan, runtime mode, kernel identity, training/inference state, and verification result. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is a versioned model or graph with shapes, operations, formats, target constraints, determinism mode, resource limits, and requested training or inference path. The principal output is a result plus the operation, format, backend, dispatch, kernel, mode, and verification identities required to interpret or reproduce it. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Core implements execution. It does not make a model semantically correct, select lawful data, decide business authority, prove a benchmark generalises, or turn approximate/creative work into hard determinism without the required numeric and artefact constraints.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	the binary AI framework and runtime operation matrix
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Core from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

CORE ARCHITECTURE · FLOW

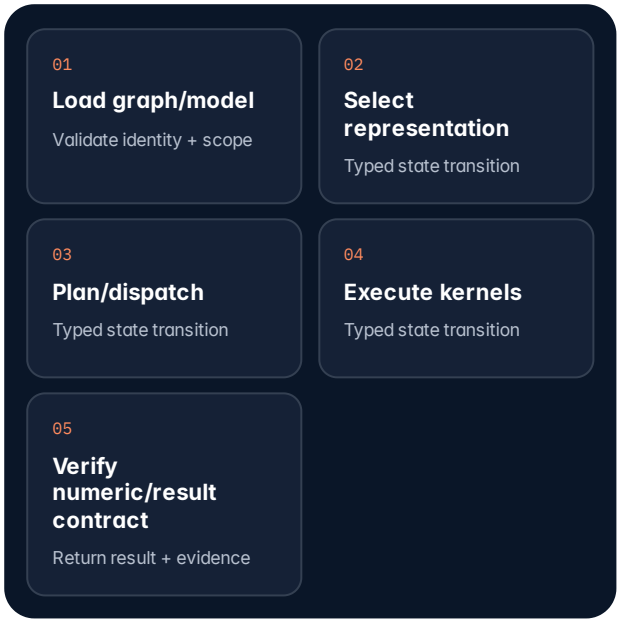
Core: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

The route makes the combined coverage matrix the product: algorithms, datatypes, widths, packing, accumulator types, backends, instruction sets, dispatch, and determinism modes. It separates storage format from accumulator type and hard, seeded, and creative runtime contracts.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Kera is a statically typed systems language with a graph-native, content-addressed IR. Programs compile to .keg files, directed acyclic graphs of operations, and execute across CPU, GPU, FPGA, and WASM. No LLVM: custom code generation for x86-64, ARM64, and RISC-V with register allocation and SIMD kernel selection. Effect-tracked side effects. Rust-style ownership across host, device, pinned, and unified memory. A Rust workspace with distributed execution, capability-based security, and LSP tooling.



Conceptual Core path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

CORE ARCHITECTURE · ASSURANCE

Core: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Reject unsupported operation/format/target combinations, invalid shape or pack, unsafe accumulator, unavailable backend, parity violation, verification mismatch, out-of-resource execution, unsupported determinism request, corrupt artefact, and migration that would silently change the numeric contract. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain model/graph hash, operation set, shapes, format and accumulator, backend/ISA/kernel, dispatch policy, runtime mode, seed where used, compiler/runtime versions, resource context, result, cross-backend comparison, and verification suite reference. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Select representative operations across required formats and backends; compare supported-target results under each determinism mode; force unsupported coordinates; inspect accumulator and dispatch choices; verify model/graph identity; measure on customer hardware; and retain raw harness output. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

CORE ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Core implements execution. It does not make a model semantically correct, select lawful data, decide business authority, prove a benchmark generalises, or turn approximate/creative work into hard determinism without the required numeric and artefact constraints.

KERA ARCHITECTURE · OWNERSHIP

Kera: responsibility, state, and boundary

Kera owns the graph-native language, content-addressed IR, compiler, planner, and target emission path. Technical evaluation begins by keeping that responsibility separate from the products, foundations, customer systems, and human authority around it.

The stable state for this boundary includes source programme, typed DAG, operation and shape identity, effect set, ownership and memory spaces, .keg content address, legal transforms, fusion/allocation/dispatch plan, target artefact, distributed/checkpoint plan, and tool diagnostics. That list is intentionally broader than a feature menu: it names the material that has to survive a retry, provider change, deployment move, audit request, or historical replay. If one of those identities lives only in a transient prompt, local variable, or operator memory, the boundary is not yet technically inspectable.

The principal input is a typed graph programme with shapes, effects, ownership, host/device memory, target, flags, capability, resource, movement, and determinism requirements. The principal output is a versioned target-specific plan or artefact whose graph identity, transformations, effects, ownership, and execution requirements remain inspectable. Production integration should express both as versioned, typed contracts with explicit error and authority states. A caller must be able to tell the difference between completed work, a refusal, an incomplete dependency, work awaiting a person, and an internal result that has not yet been reconciled with the business system.

Kera preserves and lowers programme meaning. Core owns runtime operation execution. Kera does not prove a programme's business purpose, make every target currently supported, erase backend-specific performance, or guarantee that a legal optimisation is beneficial on every workload.

03	**Output contract**	Typed result, explicit lifecycle state, and evidence references.
02	**Owned state**	the graph-native language, content-addressed IR, compiler, planner, and target emission path
01	**Input contract**	Workload-specific typed request with identity, scope, version, and authority.

The product boundary expressed as input, owned state, and output. Exact production schemas come from current interfaces.

BOUNDARY TEST

Remove Kera from the design. The architecture should name exactly which state and responsibility disappear, which caller breaks, and which other component does not silently inherit the job.

KERA ARCHITECTURE · FLOW

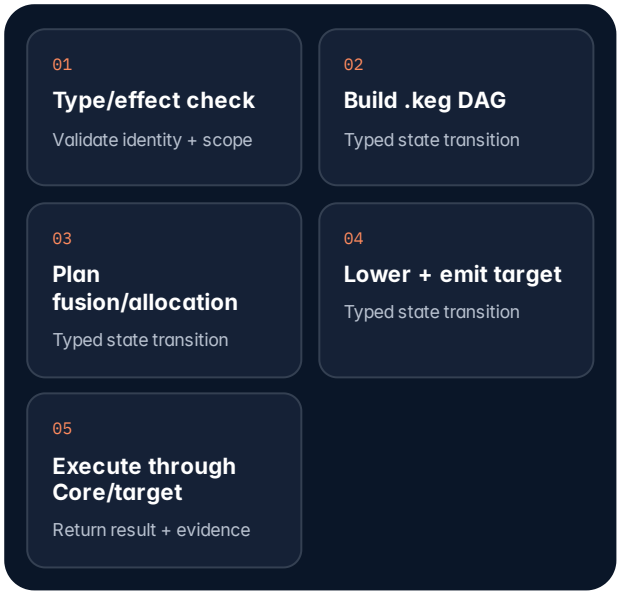
Kera: data path and contracts

The useful technical view follows identities and artefacts through the product. A box-and-arrow diagram becomes architecture only when each transition has a contract and an owner.

the published material describes a statically typed systems language with content-addressed .keg DAGs, effect-tracked side effects, Rust-style ownership across memory spaces, custom planning and code generation rather than LLVM at the centre, register allocation, SIMD selection, direct target emission, distributed execution, and LSP/tooling.

The normal control path is shown at right. Data should not automatically follow the same route as control: a work request can carry content-addressed source references, capability references, or a minimal projection rather than copying the entire upstream object. Each transition should state validation, version compatibility, timeout, retry, cancellation, access, retention, and the evidence returned to its caller.

Integration surfaces include Kera source/tooling, .keg artefacts, x86-64/ARM64/RISC-V and other published targets, GPU/FPGA/WASM paths where scoped, Core operations, distributed ring/checkpoint machinery, CI, and deployment manifests.. For each selected surface, document authentication and authorisation, tenancy or workload scope, schema/version negotiation, payload and reference limits, streaming or ordering behaviour, idempotency, error taxonomy, rate/resource limits, observability, and the point at which an external side effect becomes authoritative.



Conceptual Kera path. The implementation contract must replace these labels with current interface and artefact versions.

AT EVERY TRANSITION	REQUIRED ANSWER
Identity	Which work, actor, source, artefact, and version?
Authority	Who may request, inspect, approve, or act?
State	What entered, changed, failed, or completed?
Compatibility	Which schema and product versions interoperate?
Evidence	Which receipt, event, trace, or proof returns?

Inspect the mechanism first; only then judge the product claim built on top.

KERA ARCHITECTURE · ASSURANCE

Kera: failure, evidence, and acceptance

A product boundary is credible when failure remains explicit, recovery does not erase history, and a buyer can test the public mechanism on a representative workload.

Failure design: Reject type/shape mismatch, undeclared effect, illegal ownership or movement, unsupported target/capability, impossible allocation, invalid fusion, non-deterministic transform under a hard contract, code-generation mismatch, distributed/checkpoint error, and artefact whose identity cannot be reconciled with its graph. Recovery must keep the original work identity and append state transitions rather than replacing the record with the eventual success. Where an action may have reached another system, retry is unsafe until reconciliation or idempotency proves that repeating it cannot duplicate the effect.

Evidence design: Retain source and graph hash, type/effect result, ownership/movement decisions, applied transforms, fusion/allocation/dispatch plan, target/features, generated artefact identity, distributed/checkpoint metadata, Core/runtime binding, diagnostics, and result verification. Operational logs may reference this packet, but they should not become the only store for material needed by an authorised examiner. Retention and access should preserve verification dependencies while limiting disclosure to the purpose and role of the reviewer.

Acceptance exercise: Compile a representative graph; inspect types, effects, movement and .keg identity; compare target plans; reject an undeclared effect and illegal move; verify hard-mode output across supported targets; benchmark emitted paths against the declared baseline; and reproduce from pinned artefacts. Run the exercise with versions, configuration, hardware or service posture, raw results, reviewer, exception, and decision retained. Public measurements or availability statements remain qualified by their published source and current product status.

KERA ACCEPTANCE	representative-workload	
NORMAL	**contract + expected result**	RUN
BOUNDARY	**limits + version edge**	RUN
REFUSAL	**authority or policy denied**	RUN
FAILURE	**dependency + recovery**	RUN
HISTORY	**retrieve + verify/replay**	RUN
DECISION	**gap, owner, disposition**	RECORD

A feature demonstration is not this acceptance exercise.

DO NOT OVERCLAIM

Kera preserves and lowers programme meaning. Core owns runtime operation execution. Kera does not prove a programme's business purpose, make every target currently supported, erase backend-specific performance, or guarantee that a legal optimisation is beneficial on every workload.

04

OPEN FOUNDATIONS

Inspect the component, artefact, and reproduction method

Two pages per current canonical foundation page: how the mechanism joins the platform, then how failure and reproducibility should be tested.

Source and release availability still require component-level verification.

OPEN FOUNDATION · PORTABLE DECISION EVIDENCE

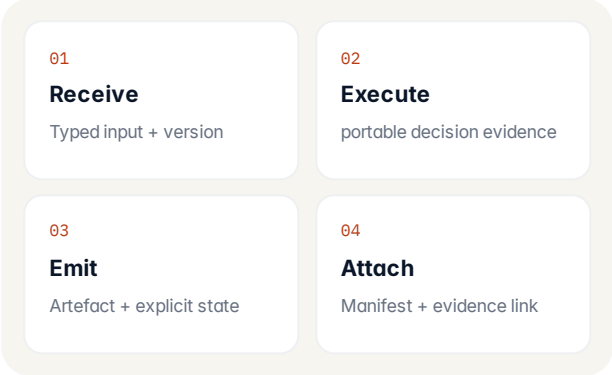
AION: mechanism and place in the stack

AION is the inspectable component for portable decision evidence. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

AION packages a decision and the premises needed to check it into a certificate that a compatible checker can validate step by step. The route distinguishes this reasoning/decision proof from a system event history or a sandbox execution transcript.

The material interface includes decision identity, premises, rule or constraint references, proof steps, certificate, Merkle or commitment material where used, signature/public-key material, checker result, and format/version manifest. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include library integration, command-line checking, standard proof formats, compatible external checkers, offline review service, and isolated archive verification. Integration points include Loom emissions and solver results, Lattice answers, Reed roots, Numerus/FMI exact calculations, platform trace packets, and an auditor-facing evidence bundle. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual AION component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned AION artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

AION: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Malformed proof, missing premise, unsupported format/version, signature or root mismatch, incompatible checker, altered artefact, or a logically valid certificate attached to the wrong business identity must produce an explicit invalid or unverifiable verdict. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Produce a certificate, move it with only its declared dependencies to an isolated machine, check it with the documented verifier and a compatible alternative where supported, alter one committed byte, rotate keys without losing historic verification, and record the verdict. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: A valid proof establishes the declared relationship between retained premises and conclusion. It does not prove the premises were true, complete, fair, lawful, or used for a legitimate purpose. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

AION REPRODUCTION	component-evidence	
COMPONENT	**source/binary + version**	PIN
INPUTS	**corpus + identities**	PIN
CONFIG	**profile + target + limits**	PIN
METHOD	**harness + comparison**	PIN
RESULT	**raw output + verdict**	RETAIN
LIMITS	**scope + exceptions**	RETAIN

Run an example certificate before integrating AION into a live decision path.

EXIT TEST

Archive the AION artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · UNIFIED OFFICE-DOCUMENT PARSING AND QUERYING

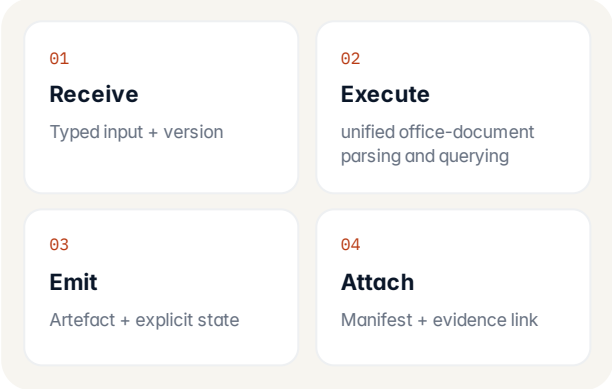
Bindery: mechanism and place in the stack

Bindery is the inspectable component for unified office-document parsing and querying. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Bindery detects and parses the website's supported office, document, archive, spreadsheet, presentation, iWork, ODF, RTF, PDF, and encrypted-format paths into one structured document model. Formula and DocQL surfaces make the parsed content computable rather than a bag of extracted strings.

The material interface includes input content identity, detected format, reader/writer and parser version, structural tree/model, cells/formula results where applicable, embedded-object state, warnings, query, output format, and content-address reference. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include Rust crate, PyO3 Python API, CLI convert/query/inspect commands, native embedding, shell pipelines, and downstream Spindle/BitWeave/Reed integration. Integration points include Spindle knowledge stages, BitWeave indexing, Reed content-addressed parsing, Fabric uploads, archive/compliance workflows, and customer document stores. "Open" does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual Bindery component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Bindery artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Bindery: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Unsupported or malformed container, wrong password, formula incompatibility, corrupt embedded object, reader/writer loss, zip-bomb/resource condition, structure ambiguity, or partial conversion should remain a typed parse result rather than silently returning incomplete text. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Assemble a representative format corpus including large, encrypted, malformed, formula-heavy, and round-trip cases; pin reader/version; compare structure and formulas; enforce size-tier budgets on customer hardware; inspect warnings; and follow one node into the downstream index. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Successful parsing proves that a document was interpreted under a reader contract. It does not make its content authoritative, grant rights to use it, or guarantee semantic equivalence for every unsupported application feature. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

BINDERY REPRODUCTION	component-evidence	
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities	PIN
CONFIG	profile + target + limits	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN

Before the computer can answer, something has to open that sheet and read the real numbers inside. That something is Bindery. It does this quietly, in the background.

EXIT TEST

Archive the Bindery artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · DETERMINISTIC BINARY SEMANTIC ENCODING AND RETRIEVAL

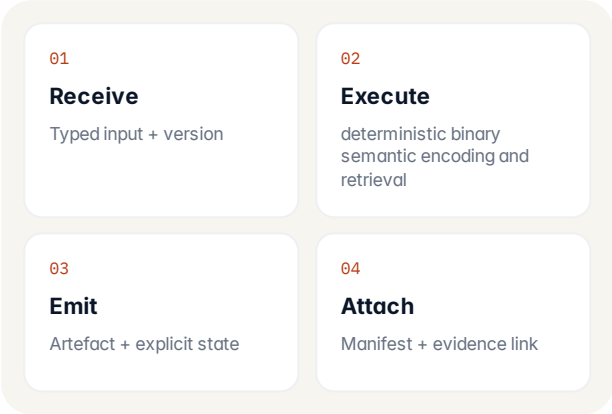
BitWeave: mechanism and place in the stack

BitWeave turns text, images, audio, and existing vectors into fixed-width binary hypervectors: a native semantic type with an algebra you can compose, compare, index, filter, rank, store, and run anywhere. Search is one operation. RAG is one application.

BitWeave encodes documents, passages, queries, or symbols as fixed-dimensional binary hypervectors, using bundle, bind, and permute operations. Locality-sensitive hashing narrows candidates and exact Hamming rescoring preserves the final comparison contract.

The material interface includes encoder/version, dimension and seed, source identity, hypervector, index/.bwks identity, band configuration, query vector, candidate set, exact scores, ranking, runtime mode, and benchmark/harness reference. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include in-process Rust/C/Python use, standalone TCP or Unix-socket server, compatible WASM state, one portable index format, and benchmark API. Integration points include Spindle governed atoms, Bindery/Reed outputs, Loom retrieval, Fabric knowledge, local browser/edge search, and Numerus-supported exact arithmetic where described. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual BitWeave component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned BitWeave artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

BitWeave: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Encoder/version drift, seed mismatch, incompatible state, corrupt index, unsupported dimension, candidate loss from a changed band policy, resource exhaustion, or failure to exact-rescore must be observable. An approximate candidate stage must not be reported as exact final scoring. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Rebuild the same source set twice, compare hypervectors and index identity, query through each runtime, reproduce the published corpus/method on the stated hardware class, vary bands and candidate budgets, confirm final score/recall behaviour, and test a corrupted state. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Deterministic similarity is not semantic truth. Retrieval recall claims belong to their corpus, configuration, hardware, and method; a retrieved passage still needs source authority and use-right checks. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

BITWEAVE REPRODUCTION	component-evidence	
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities	PIN
CONFIG	profile + target + limits	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN

BitWeave rows from the repo benchmark report. TurboVec is a public competitor baseline. Reproduce them with the bundled example.

EXIT TEST

Archive the BitWeave artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · BIT-EXACT DETERMINISTIC SIMULATION ACROSS SUPPORTED TARGETS

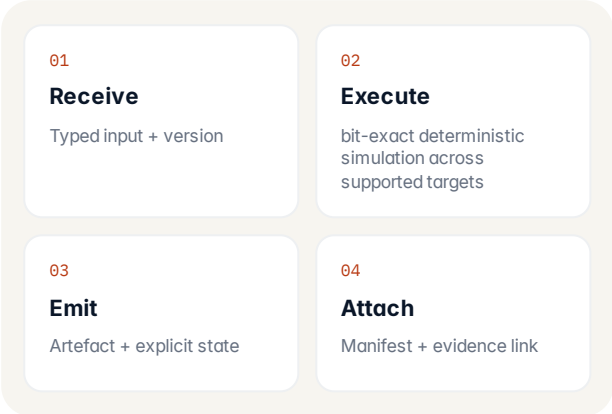
FMI: mechanism and place in the stack

FMI is the inspectable component for bit-exact deterministic simulation across supported targets. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

FMI binds a simulation model to fixed-point profiles and a result contract, then compares supported-target execution against high-precision reference checks. The published description positions platform parity and MPFR-backed validation as evidence for one safety case rather than repeated reconciliation.

The material interface includes model/version, parameter and initial-state snapshot, numeric profile, step/integration contract, target/backend identity, result series, cross-target comparison, MPFR oracle result, validation record, and supported-target matrix. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include library/runtime use in simulation workloads, customer hardware, on-premises or EU-region deployment, example models, validation harness, and integration with Numerus/BitWeave/AION. Integration points include Numerus arithmetic, Core/Kera execution, digital twins, Loom solvers, safety-case evidence, AION certificates, and operational models retained for replay. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual FMI component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned FMI artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

FMI: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Unsupported operation/profile/target, overflow, unbounded error, different initial state, step-order drift, non-equal result, oracle divergence, invalid model interchange, or missing validation dependency must block a bit-exact claim. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Pin model, state, profile, step contract, backend and versions; execute on two supported targets; compare result bytes; run the documented high-precision oracle and tolerance rule; alter one parameter; and retain the target matrix and raw validation output. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Bit-exact simulation does not prove that the model represents reality, that parameters are correct, or that a safety case is adequate. It removes one reproducibility variable within the declared profile. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

FMI REPRODUCTION	component-evidence	
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities	PIN
CONFIG	profile + target + limits	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN

MPFR-backed checks catch divergence before use

EXIT TEST

Archive the FMI artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · COMPACT, LOCAL, LOSSLESS EXCHANGE FOR MODEL AND AGENT DATA PATHS

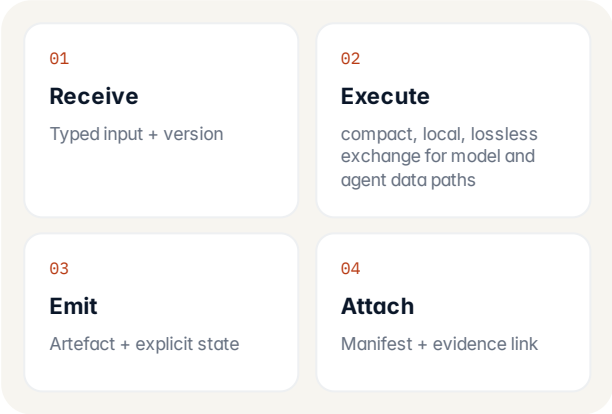
HEDL: mechanism and place in the stack

HEDL adds one compact exchange layer before the model while every database, report, and application continues speaking JSON. Run conversion locally, measure it on one workflow, and remove it without a data migration if the numbers do not hold.

HEDL removes syntactic overhead while preserving the supported data semantics and restoring the original exchange shape. Conversion is local and bidirectional; a proxy can speak HEDL upstream and JSON downstream while recording token counts around the same payload.

The material interface includes input/output format and schema, converter version, canonical payload identity, encoded bytes, restored payload, round-trip verdict, token measurement, model/task/dataset context, and interface/runtime identity. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include Rust core/crate, CLI, MCP server, and WASM component with local conversion and a zero-runtime-dependency core as described in the published material. Integration points include Nexus agent messages, Aura MCP tools, retrieval and extraction calls, batch jobs, customer APIs, and any model path where payload tokens are billed by volume. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual HEDL component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned HEDL artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

HEDL: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Unsupported value or schema, ambiguous positional encoding, loss on round trip, ordering/canonicalisation mismatch, proxy framing error, model behaviour change, unavailable downstream JSON consumer, or measurement without task/model context invalidates the “same data” claim. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Round-trip the production payload corpus byte/semantic-equivalently under the documented contract; compare schema edge cases; run direct and proxy paths; reproduce token and accuracy measurements with the named models/tasks; remove the proxy; and confirm the application returns to JSON. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Fewer tokens do not guarantee lower total cost, latency, or equal model behaviour for every provider. Published percentages remain tied to the documented benchmark datasets, models, and extraction method. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

HEDL REPRODUCTION	component-evidence	
COMPONENT	**source/binary + version**	PIN
INPUTS	**corpus + identities**	PIN
CONFIG	**profile + target + limits**	PIN
METHOD	**harness + comparison**	PIN
RESULT	**raw output + verdict**	RETAIN
LIMITS	**scope + exceptions**	RETAIN

Drop in the MCP server and your agent stack speaks HEDL upstream and JSON downstream with no application code change.

EXIT TEST

Archive the HEDL artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · SEALED ACTION-AND-RESULT RECEIPTS FOR AGENTS AND TOOLS

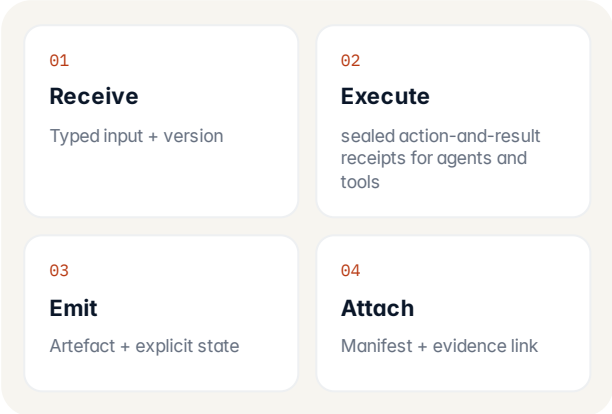
Knot: mechanism and place in the stack

Knot is the inspectable component for sealed action-and-result receipts for agents and tools. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Knot captures each action beside its result while work runs, masks supported sensitive fields before storage, orders the entries, signs or seals the chain, and exports a portable bundle for independent checking. Its subject is agent/tool activity, not the internal logical proof of a decision.

The material interface includes work and actor identity, capability, action request, arguments after policy-aware masking, time/order, result or error, gate state, previous-entry reference, root/signature, public key, export manifest, and verification verdict. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include agent runtime integration, tool boundary, incident and audit export, portable verifier, on-premises or EU deployment, and offline verification with the public key. Integration points include Nexus tasks, Aura tools, Fabric work, Selvedge execution transcripts, Ledger events, platform trace packets, incident review, and external assurance hand-off. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual Knot component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Knot artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Knot: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Uncaptured side effect, result without action, secret masking after persistence, broken order, reused identity, missing key, signature mismatch, partial export, or inaccessible verifier must be surfaced as a coverage or integrity gap. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Run success, denial, timeout, retry, partial result, and duplicate action; inspect pre-storage masking; export the bundle; verify offline; alter order and payload; correlate with external receipts; and confirm that a reviewer sees only authorised fields. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: An honest receipt can show what the agent/tool boundary recorded. It does not prove the tool's external side effect completed unless the authoritative target receipt and reconciliation are attached. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

KNOT REPRODUCTION	component-evidence	
COMPONENT	**source/binary + version**	PIN
INPUTS	**corpus + identities**	PIN
CONFIG	**profile + target + limits**	PIN
METHOD	**harness + comparison**	PIN
RESULT	**raw output + verdict**	RETAIN
LIMITS	**scope + exceptions**	RETAIN

A reviewer can reconstruct and verify the run offline with the public key.

EXIT TEST

Archive the Knot artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · COMPILED DETERMINISTIC POLICY AND CONSTRAINT EVALUATION

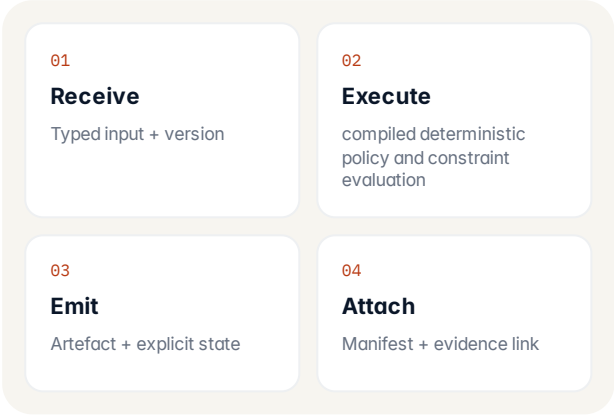
Lattice: mechanism and place in the stack

Lattice is the inspectable component for compiled deterministic policy and constraint evaluation. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Lattice compiles rules and constraints into a memory-mappable artefact evaluated inside the application. A named rulebook checksum and deterministic query path make the same decision reproducible across the route's supported targets without an outside policy service in the critical path.

The material interface includes source rulebook/version, compiler identity, compiled artefact/checksum, schema, query facts, target/runtime identity, evaluated clauses, result/refusal, explanation or replay path, and optional AION certificate. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Charter is organized as four planes over one operating graph. The experience plane is the web app that gives people a coherent interface. The policy plane is the API and policy layer that validates every request and applies permissions. The operating graph stores the durable business state. The execution plane handles events, workflows, jobs, realtime collaboration, AI actions and external integrations. It holds together because domains share spines: a support ticket can see subscription status because support and billing share a customer, and a project inherits context from a won deal because delivery and CRM agree on the company.



Conceptual Lattice component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Lattice artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Lattice: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Invalid rule, ambiguous precedence, incompatible schema, missing fact, stale artefact, checksum mismatch, unsupported target, numeric divergence, unavailable owner for a policy conflict, or rulebook change after review must stop or create a new decision state. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Compile the approved rulebook twice, compare artefact identity, run boundary and missing-fact cases on supported targets, switch failover hardware, reproduce the benchmark in its scope, replay a historical query with its old artefact, and alter the checksum. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Deterministic rule evaluation does not decide that the rule is fair, lawful, current, or applicable. Those facts remain with the policy owner and source-governance process. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

LATTICE REPRODUCTION	component-evidence	
COMPONENT	**source/binary + version**	PIN
INPUTS	**corpus + identities**	PIN
CONFIG	**profile + target + limits**	PIN
METHOD	**harness + comparison**	PIN
RESULT	**raw output + verdict**	RETAIN
LIMITS	**scope + exceptions**	RETAIN

lattice build / lattice verify

EXIT TEST

Archive the Lattice artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · TYPED, HASH-CHAINED SYSTEM EVENT PROVENANCE

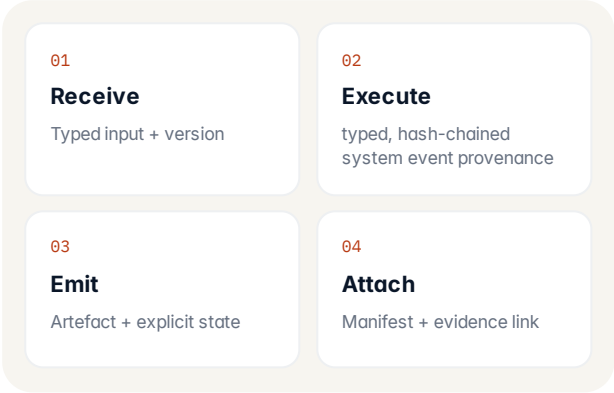
Ledger: mechanism and place in the stack

Ledger is the inspectable component for typed, hash-chained system event provenance. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Ledger appends typed events to a hash chain that can be queried and replayed. The same event shape can use memory, JSONL, SQLite, Postgres, or archival storage, and can be embedded through a crate/C ABI or operated through sidecar/service forms.

The material interface includes stream and event identity, typed schema/version, actor/source, timestamp/order, payload or protected reference, previous hash, event hash, signature/attestation where configured, checkpoint, storage backend, replay cursor, and verification result. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include Rust crate, C ABI, sidecar, service binary, five documented storage tiers, lifecycle/tool/gate/data-right/incident/attestation events, query and replay. Integration points include Fabric tenant audit, Nexus organisation history, Aura receipts, Spindle knowledge events, Charter outbox/lifecycle, Knot actions, Selvedge transcripts, and external archival/WORM storage. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual Ledger component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Ledger artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Ledger: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Schema incompatibility, missing event, order break, hash mismatch, clock anomaly, duplicate append, partial backend migration, corrupt checkpoint, unauthorised payload access, or replay that applies a side effect twice must be detected and contained. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Append each selected event category, migrate between two storage tiers without re-encoding, verify chain and schema, alter/delete/reorder events, replay from checkpoint into a read-only projection, test duplicate delivery, and retrieve a rights or incident chronology. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

The procurement pack is a retrieval, not a scramble, because the frameworks are design constraints rather than questionnaires answered after the fact. GDPR and the EU AI Act, including general-purpose AI, are core compliance. NIS2, DORA, and the Cyber Resilience Act are met architecturally, built to satisfy security-by-design, vulnerability handling, and operational-resilience requirements from the ground up.

LEDGER REPRODUCTION	component-evidence	
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities	PIN
CONFIG	profile + target + limits	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN

Figures are an in-memory benchmark snapshot from the Ledger repository, not a published SLA. Run the included harness before sizing a workload.

EXIT TEST

Archive the Ledger artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · DETERMINISTIC NUMERIC PROFILES AND VERIFIED ARITHMETIC

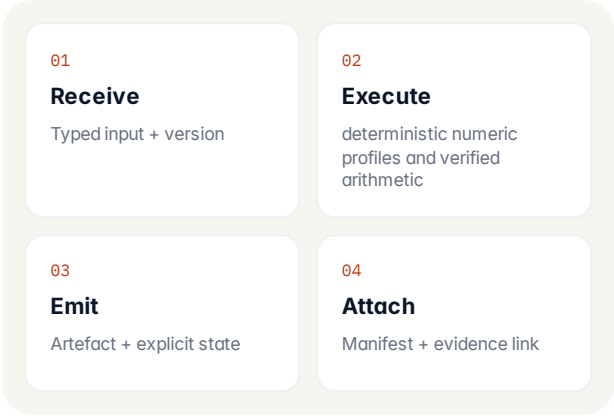
Numerus: mechanism and place in the stack

Numerus is the inspectable component for deterministic numeric profiles and verified arithmetic. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Numerus exposes Q-format binary fixed point, base-10 Decimal, and AXIOM adaptive-exponent integer-only arithmetic through a common contract. Operations and supported transcendental implementations are checked against high-precision references in documented domains and tolerances.

The material interface includes numeric family/profile, scale/exponent, value and operation types, rounding/overflow policy, implementation/version, target, inputs/result, tolerance/domain, property-test seed, MPFR or high-precision reference, and verification outcome. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include Rust crate on documented desktop/server/WASM/embedded targets, C-compatible interface for other languages, scoped component crates, per-call profile selection, and verification suite. Integration points include Core numeric formats and accumulators, Kera type/target plan, FMI simulation, Lattice numeric rules, Loom solvers, finance/science/control workloads, and proof records. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual Numerus component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Numerus artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Numerus: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Profile mismatch, overflow or saturation, unsupported range, tolerance breach, target divergence, fused-operation difference, reordered accumulation, compiler/ISA drift, ambiguous decimal scale, or unrecorded dynamic exponent invalidates deterministic replay. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Select profiles from workload semantics; run corners, ranges and property tests; compare high-precision references; execute on every required supported target; test FMA/reordering and migration cases; inspect error bounds; and retain exact profile/type/version metadata. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Deterministic arithmetic guarantees a declared numeric behaviour, not that the chosen profile has enough range or precision for the domain. An owner still approves units, tolerance, rounding, and error consequences. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

NUMERUS REPRODUCTION	component-evidence	
COMPONENT	**source/binary + version**	PIN
INPUTS	**corpus + identities**	PIN
CONFIG	**profile + target + limits**	PIN
METHOD	**harness + comparison**	PIN
RESULT	**raw output + verdict**	RETAIN
LIMITS	**scope + exceptions**	RETAIN

cargo test -p numerus-verify

EXIT TEST

Archive the Numerus artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · CONTENT-ADDRESSED PARSING WITH VERIFIABLE SYNTAX TREES

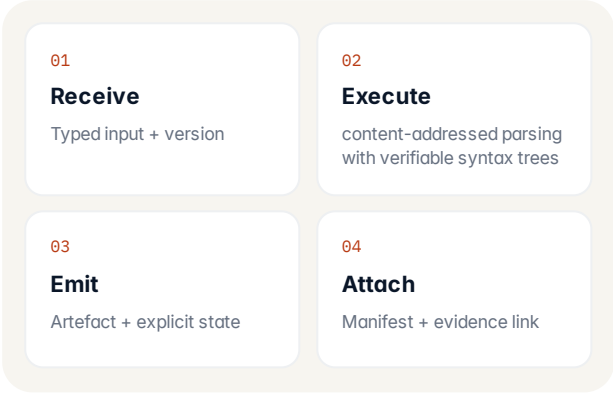
Reed: mechanism and place in the stack

Reed is the inspectable component for content-addressed parsing with verifiable syntax trees. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Reed offers fast structural scanning, table-driven and general parsing paths, ambiguity support, syntax forests/trees, query patterns, external scanners, and certificate hashing. Parsed nodes can become content-addressed inputs to knowledge and proof workflows.

The material interface includes input bytes/hash, grammar and parser-engine version, scanner registry, tokens, syntax tree or forest, ambiguity/error state, query/pattern, match set, Merkle/certificate root, bundle format, and replay result. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include native Rust, CLI parse/query/certify, editor/LSP diagnostics, C ABI for Python/Go/WASM, scanner extension, tree-pattern/RQL query, and RGB1-style bundle described in the published material. Integration points include Bindery document flow, Spindle atoms, source-code analysis, BitWeave indexing, AION certificates, Selvedge verified runtime, build pipelines, and customer language formats. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



reed @ your-project

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Reed artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Reed: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Grammar/version drift, lexical ambiguity, unsupported external scanner, malformed input, recovery that changes tree meaning, non-deterministic traversal, query mismatch, certificate-root change, or parser-engine parity failure must be explicit. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Pin grammar, scanner, engine and source; parse with relevant fast/general paths; compare accepted tree meaning; exercise ambiguity and malformed input; run stable queries; certify the root; replay the bundle; alter a node; and reproduce performance only with the published file/workload method. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: A verifiable syntax tree proves how bytes were parsed under a grammar. It does not establish that the document statement is true, the code is safe, or the grammar captures every intended semantic. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

REED REPRODUCTION	component-evidence	
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities	PIN
CONFIG	profile + target + limits	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN

CI publishes the certificate root of the parsed source. Consumers verify against it before they deploy.

EXIT TEST

Archive the Reed artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · CAPABILITY-LIMITED EXECUTION WITH A SEALED TRANSCRIPT

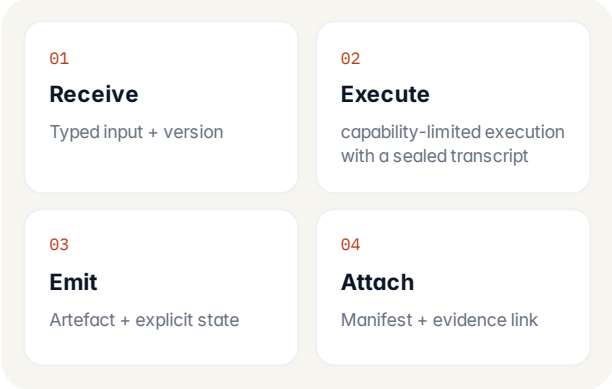
Selvedge: mechanism and place in the stack

Selvedge is the inspectable component for capability-limited execution with a sealed transcript. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Selvedge runs untrusted code through a default-deny sandbox boundary, grants capabilities per execution, limits memory/time/fuel or equivalent controls, and seals the execution transcript for later verification and replay. The route distinguishes containment from the evidence envelope.

The material interface includes module/graph identity, engine/runtime/version, declared capabilities, granted resources, input commitments, host calls, gate results, stdout/stderr or structured result, errors, time/resource measurements, transcript hash/signature, and replay verdict. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include embedded crate, CLI build/verify/run/replay, long-running service, MCP agent-tool server, Wasmtime component model, Kera engine, and portable proof envelope. Integration points include Aura/Nexus tools, third-party plugins, Kera/Core graph execution, Knot action receipts, Ledger system events, AION proof material, regulated data work, CI, and isolated deployments. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual Selvedge component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Selvedge artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Selvedge: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Undeclared capability, filesystem/network escape, memory/time/fuel breach, host-call denial, engine mismatch, non-deterministic dependency, transcript gap, signature mismatch, replay difference, or external side effect outside reconciliation must block a verified result. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Run allowed and denied modules, attempt each forbidden capability, exhaust limits, compare engine paths in scope, inspect transcript coverage, sign and verify offline, alter the module and transcript, replay in isolation, and reconcile every permitted external effect. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Containment limits what a module can touch; a transcript records the declared boundary. Neither proves the code's purpose is legitimate or that every host/environment behaviour is deterministic. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

SELVEDGE REPRODUCTION	component-evidence	
COMPONENT	**source/binary + version**	PIN
INPUTS	**corpus + identities**	PIN
CONFIG	**profile + target + limits**	PIN
METHOD	**harness + comparison**	PIN
RESULT	**raw output + verdict**	RETAIN
LIMITS	**scope + exceptions**	RETAIN

Your team can evaluate and run Selvedge on its own infrastructure before committing to an outside service contract.

EXIT TEST

Archive the Selvedge artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · EVENT-SOURCED DIGITAL-TWIN STATE THROUGH APPEND, FOLD, AND REPLAY

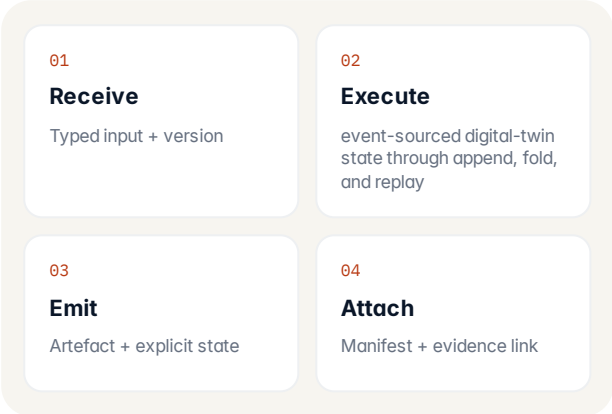
Twin: mechanism and place in the stack

Twin is the inspectable component for event-sourced digital-twin state through append, fold, and replay. Its technical value is the artefact and contract it contributes, not the fact that another product can mention its name.

Twin retains an append-only sequence of asset events and folds them into current state. Replaying the same retained event history reconstructs a prior or current twin state, supporting incident and audit questions without treating a mutable snapshot as the whole record.

The material interface includes asset/twin identity, event type/schema/version, source/time/order, payload or reference, append result, fold/projection version, current state, checkpoint, replay range/result, correction/forgetting event, and integrity link. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include licensed deployment on customer hardware, managed-service or disconnected postures described in the published material, connectors to assets, event append/query/replay, and FMI/Spindle/Ledger integration. Integration points include physical assets and sensors, FMI deterministic simulations, Spindle governed knowledge, Ledger software events, maintenance/incident processes, customer operational systems, and proof/audit packets. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.



Conceptual Twin component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Twin artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Twin: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Late/out-of-order event, duplicate, incompatible schema, corrupt checkpoint, impossible state transition, missing physical measurement, clock drift, correction/forgetting conflict, replay mismatch, or side effect executed during reconstruction must be handled explicitly. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Append representative and out-of-order events, rebuild from zero and checkpoint, compare state, replay an incident window, apply a correction under policy, disconnect the supplier/network path, restore the history, and reconcile the twin with a sampled physical asset. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: An exact replay of retained events proves the software state derived from those events. It does not prove sensors were correct or that the digital model perfectly reflects the physical asset. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

TWIN REPRODUCTION	component-evidence	
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities	PIN
CONFIG	profile + target + limits	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN

The implementation is inspectable line by line, so teams can verify behaviour before production.

EXIT TEST

Archive the Twin artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

OPEN FOUNDATION · GOVERNED, QUESTION-LED COLLECTION ACROSS EXTERNAL AND LOCAL SOURCES

Winnow: mechanism and place in the stack

Pre-built scrapers for the sources analysts already use, from government and science portals to news, finance, and open data. Each scraper is a typed Rust struct. Custom scrapers drop in beside the built-ins and follow the same output contract.

Winnow turns a question or collection instruction into a repeatable route that discovers, fetches, parses, normalises, and emits typed results. Each request retains source, URL, time, status, and content identity while per-domain limits and private-network protections constrain collection.

The material interface includes question/run identity, engine/collector/scraper version, source and URL, request policy, time/status, response/content hash, parse/normalise result, output schema/format, error/retry, rate-limit state, and audit record. A production manifest should pin the component, schema/format, runtime or checker, relevant configuration, input identities, output identity, and compatibility requirements. That makes a downstream trace precise enough to retrieve or reproduce the component result.

Supported surfaces described in the published material include single binary, Rust crates, axum HTTP service, browser/edge/WASM or PyO3 surfaces described in the published material, extensible typed scrapers, search engines, APIs, feeds, websites, and files. Integration points include Spindle source intake, Reed/Bindery parsing, BitWeave indexing, Fabric/Loom research, analyst workflows, customer proxies, and local data-governance controls. “Open” does not make every surface, version, licence, repository state, or commercial right interchangeable; confirm the current source/release status and the exact artefact used in the installed platform.

Conceptual Winnow component contract. Exact names and formats follow the current project surface.

ARCHITECTURAL JOIN

The upstream component should pass the minimum sufficient input. The downstream component should consume a versioned Winnow artefact or reference, not reinterpret hidden internal state.

OPEN FOUNDATION · VERIFY IT

Winnow: failure and reproducibility contract

Reproducibility starts by specifying what must remain equal, what may vary, which dependencies are pinned, and which verdict the component returns when the contract cannot be met.

Failure model: Disallowed/private-network URL, robots or policy restriction, authentication failure, rate limit, source format change, malicious content, redirect escape, partial crawl, duplicate or stale result, parser failure, or missing rights/owner state must be recorded and contained. The caller must receive a typed result that distinguishes invalid input, unsupported configuration, dependency or resource failure, integrity failure, verification mismatch, and a valid negative/refusal result. Treating all of them as an empty response destroys both recovery and assurance.

Verification exercise: Run a bounded approved source set; inspect every request and content hash; test SSRF/private ranges and redirects; enforce per-domain limits; change a scraper; reproduce output from retained inputs where lawful; withdraw a source; and trace a downstream answer back to the collection run. Capture the source or delivered component identity, build and runtime environment, inputs/corpus, configuration, method, raw outputs, repetitions where measured, comparison rule, limitations, reviewer, and date. A benchmark requires its hardware and method; a deterministic claim requires every identity that can alter the result.

Boundary: Technical collection capability does not grant a lawful basis, licence, permission, or authority to use a source. The operator defines those conditions and the downstream knowledge process preserves them. This limitation belongs next to the mechanism because downstream teams otherwise promote a narrow technical property into a broad business, security, or compliance conclusion.

WINNOW REPRODUCTION	component-evidence	
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities	PIN
CONFIG	profile + target + limits	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN

Winnow follows configured rate limits and robots rules before it visits a page. Those safeguards are part of the run rather than something you have to remember each time.

EXIT TEST

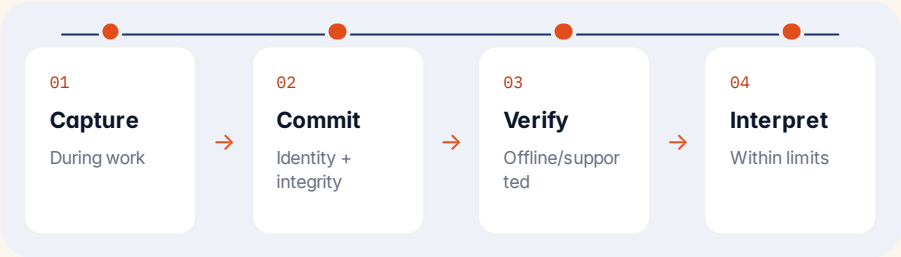
Archive the Winnow artefact, manifest, and verifier dependencies; then reproduce the declared check without a Dweve-operated service. Record exactly what still depends on Dweve.

05

ASSURANCE ARCHITECTURE

Proof is a system of bounded propositions

Evidence becomes useful when each artefact states what it can establish, how it is verified, who may see it, and what remains a human or institutional judgement.



Integrity and reproducibility enable accountability; they do not replace it.

ASSURANCE ARCHITECTURE · VOCABULARY

Evidence types answer different questions

Source lineage, work trace, event ledger, action receipt, execution transcript, proof certificate, signature, and replay packet overlap without being substitutes.

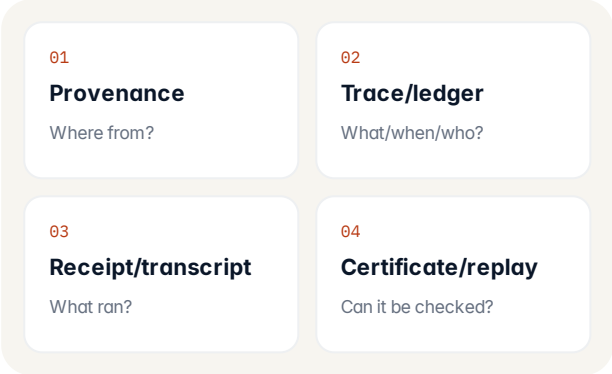
A team that calls every retained object an audit log cannot say which object establishes order, execution, reasoning, authority, integrity, or reproducibility.

A procurement review asks two plain questions: what does this save, and what does it de-risk? Ledger keeps teams from stitching evidence together across partial logs during an incident. Its append-only chain also exposes any attempt to alter history after the fact.

Contract. Every type declares producer, subject, semantic claim, schema, integrity method, required inputs, verifier, access class, retention, portability, and relationships to other evidence.

Failure and recovery. Using a log as proof, a certificate as source truth, a transcript as business reconciliation, or a signature as correctness creates an assurance claim the artefact cannot support.

Evidence and acceptance. Take one case and label every artefact by the question it can answer, the dependency it needs, and the inference it cannot justify. Reject duplicates with no distinct purpose.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can an examiner request the minimum evidence for one question without receiving an indiscriminate archive?

ASSURANCE ARCHITECTURE · PACKET

The evidence packet is a graph, not a zip file

A manifest should resolve identities and relationships while allowing policy-specific retrieval of sensitive artefacts.

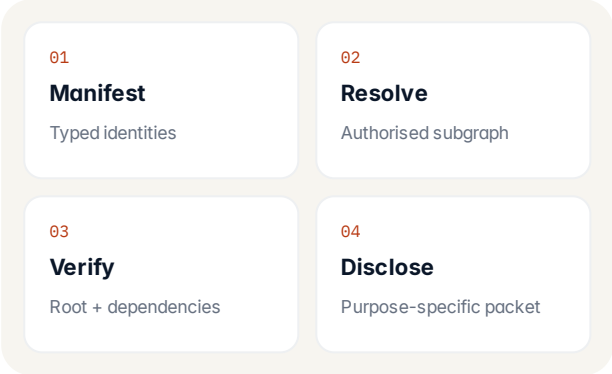
Copying every source, prompt, message, log, and output into one bundle creates retention, privacy, disclosure, and consistency problems while still omitting version dependencies.

Mechanism. Create a packet manifest whose nodes are typed artefact identities and whose edges express derived-from, used-by, approved-by, emitted, committed-in, supersedes, verifies, and reconciles-with relationships.

Contract. The manifest includes work and packet version, node type/hash/location/access/retention, edge semantics, completeness policy, root/commitment, signatures, verifier requirements, export profile, and unresolved dependency state.

Failure and recovery. Dangling reference, mutable location, access mismatch, cyclic derivation, missing schema, mixed packet versions, incorrect root, expired key, or partial export must yield an incomplete/invalid packet verdict.

Evidence and acceptance. Resolve a random historical packet under officer, examiner, operator, and affected-person views; verify the same root, different disclosures, complete required subgraphs, and recorded access.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can one integrity identity support multiple legitimate views without duplicating or over-sharing the underlying evidence?

ASSURANCE ARCHITECTURE · INTEGRITY

Merkle commitments detect change; they do not explain meaning

Tree or DAG commitments make a large evidence set tamper-evident and support selective paths, provided canonicalisation and identity rules are pinned.

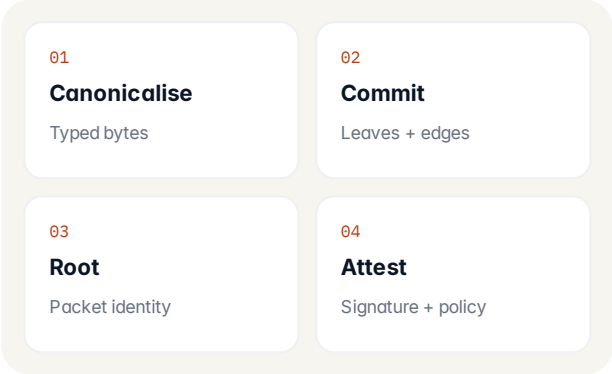
Hashing arbitrary serialisation is fragile: field order, whitespace, timestamps, omitted nodes, and algorithm migration can change roots without a semantic change, or preserve a root for the wrong identity graph.

Mechanism. Canonicalise typed artefacts, hash leaves with domain separation, order or address children deterministically, build the tree/DAG, bind root to packet/work identity, sign when attestation is required, and retain inclusion paths.

Contract. Pin canonical schema/version, hash algorithm, domain tags, node/edge encoding, ordering, absent/null treatment, root construction, signer/key id, signature algorithm, and verification policy.

Failure and recovery. Ambiguous canonicalisation, hash-algorithm confusion, duplicate leaf identity, omitted required node, wrong packet binding, compromised key, or unverifiable historic algorithm must fail explicitly.

Evidence and acceptance. Recompute roots independently, verify inclusion/exclusion paths, permute non-semantic input, alter one byte, remove a required node, rotate signers, and verify old packets under the archived trust policy.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Does the commitment unambiguously bind the intended work graph, or only a convenient collection of bytes?

ASSURANCE ARCHITECTURE · KEYS

Signing establishes an attester, not automatic trust

The signature is useful only when identity, key purpose, custody, validity, revocation, and historical verification policy are explicit.

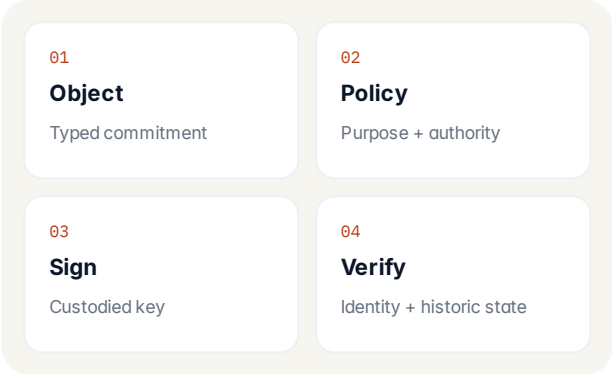
A cryptographically valid signature from an unknown, over-privileged, expired, or compromised key can still be operationally meaningless.

Mechanism. Separate release, service, work, human approval, evidence-root, and external-receipt signing purposes. Generate and store keys within the selected custody boundary; bind policy and time; publish verification material; rotate without destroying history.

Contract. Signature metadata includes algorithm, key id, signer identity/role, purpose, signed object type/hash, signing time, validity policy, certificate/attestation chain where used, and revocation/historic-verification state.

Failure and recovery. Key reuse across purposes, silent rotation, unavailable revocation data in air gap, lost historic public key, clock uncertainty, stolen operator credentials, or signature without authority must be contained.

Evidence and acceptance. Sign and verify each selected object class, attempt cross-purpose use, rotate and revoke, restore key-verification material, verify historic work offline, and simulate compromise response.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

What exact proposition and authority does each signature carry, and can that proposition still be checked after rotation and exit?

ASSURANCE ARCHITECTURE · VERIFIER

Offline verification is a supply-chain requirement

A portable certificate or transcript is not independently checkable if its schema, checker, keys, runtime, or dependencies disappear with the supplier.

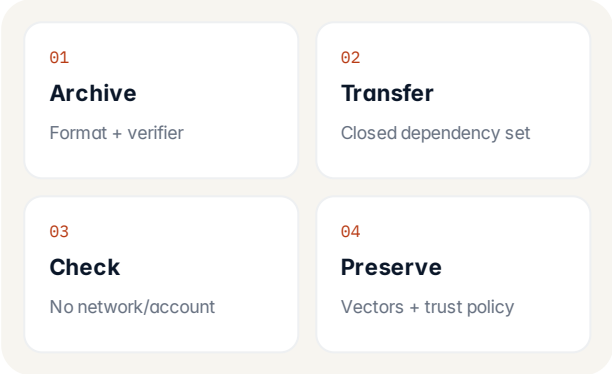
"No network required" describes runtime connectivity. Long-lived verification additionally requires preservable formats, implementations, trust material, documentation, and test vectors.

Mechanism. Package the verifier or its buildable source, format/schema, dependency manifest, keys/trust policy, sample valid/invalid vectors, command or API, expected verdict semantics, and migration guidance for algorithms/formats.

Contract. The offline verification profile states supported artefact versions, operating/runtime requirements, network prohibition, dependency hashes, trust-store snapshot, result codes, resource limits, and unsupported/expired handling.

Failure and recovery. Dynamic package fetch, remote licence/account, hosted key lookup, current-only schema, undocumented exit code, unavailable CPU feature, expired certificate without historic policy, or unbounded input can defeat independent checking.

Evidence and acceptance. Move only the declared package and evidence to a clean disconnected machine, build/run it, check positive and negative vectors, remove a dependency, cross a key/format boundary, and record reproducibility.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can a third party verify a retained case years later without access to the production environment or a Dweve service?

ASSURANCE ARCHITECTURE · NUMERIC/RUNTIME

Determinism begins at representation and ordering

Integer-oriented profiles, fixed rounding, stable traversal, seeded randomness, pinned kernels, and target contracts remove distinct sources of divergence.

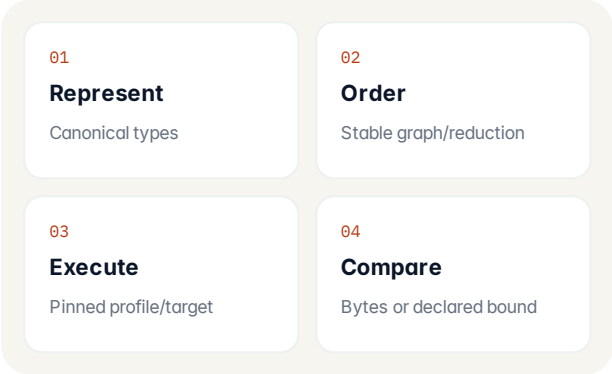
Two runs with the same high-level model can differ through input normalisation, map iteration, parallel reduction, FMA, accumulator width, compiler flags, instruction set, backend, and race timing.

Mechanism. Specify canonical input representation, operation semantics, numeric family/profile, accumulator, rounding/overflow, reduction order, RNG/seed, scheduler constraints, compiler/runtime versions, target/backend/ISA, and comparison.

Contract. The result manifest exposes each dimension that may alter bits or accepted invariants and rejects a hard-mode request when any executed operation lacks a supported deterministic path.

Failure and recovery. Fallback kernel, reordered graph, dynamic provider, unknown CPU feature, mixed numeric profile, parallel race, external time/randomness, or compiler optimisation can invalidate equivalence.

Evidence and acceptance. Use boundary/corner/property cases, multiple supported targets, repeated runs, changed ISA/backend, MPFR/high-precision oracle where applicable, and mutation tests for every pinned dimension.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can the manifest explain every source of allowed and disallowed variation rather than merely recording a seed?

ASSURANCE ARCHITECTURE · PARITY

Cross-backend parity needs an operation-level oracle

A final-output comparison can hide compensating errors and offers little help when a new backend diverges.

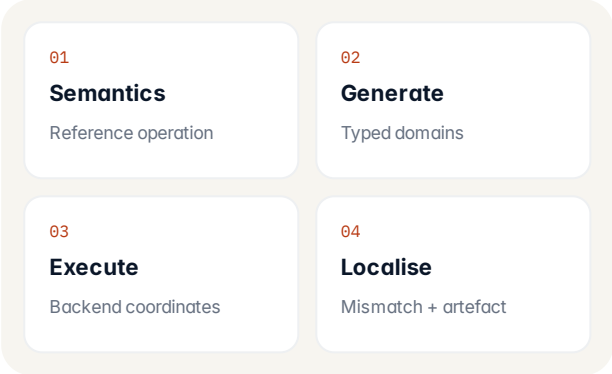
Core and Kera span representations, kernels, numeric types, accumulators, dispatch, and targets. Parity must be tested at supported coordinates in that matrix, not inferred from one model.

Mechanism. Define reference semantics per operation/profile, generate typed test domains including edges, compare intermediate and final artefacts, test graph transforms and fused paths, and maintain backend capability/exception matrices.

Contract. A parity record binds operation/version, shapes, datatype/width, packing, accumulator, profile, target/backend/ISA, kernel/dispatch, input generator/corpus, oracle, tolerance/equality, and raw result.

Failure and recovery. Unsupported coordinate, hidden promotion, saturation, denormal/NaN semantics where relevant, alignment/tail bug, fusion difference, race, target-specific fast path, or oracle error must localise to a coordinate.

Evidence and acceptance. Run conformance and property suites per backend, differential-test pairs, compare with high-precision/reference paths, retain failing seeds, and prevent unsupported cells from dispatch.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Which exact operation/profile/target cells are proven, excluded, or still research, and can dispatch enforce that matrix?

ASSURANCE ARCHITECTURE · ACCESS

Security boundaries follow identities and capabilities

Network location alone does not constrain a model, agent, tool, source, operator, verifier, or support process.

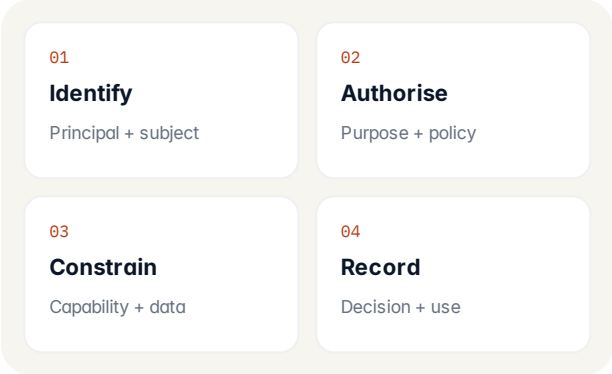
Zero-trust and default-deny claims become concrete only when every call identifies actor, subject, purpose, object, capability, policy, and evidence.

Mechanism. Authenticate each human/service/agent; authorise a scoped capability at every material hop; project minimum data; separate administration, policy, signing, and review; enforce limits at tool/sandbox and storage boundaries; record decisions.

Contract. Access decision includes principal and delegation chain, tenant/workspace, purpose, resource/field, operation, condition, policy/version, expiry, decision/reason, and downstream capability token or denial.

Failure and recovery. Ambient service credentials, confused deputy, stale capability, role explosion, cross-tenant cache, policy bypass through fallback, support access, or evidence export can cross the intended boundary.

Evidence and acceptance. Threat-model and test every trust transition; attempt escalation, replay, delegation abuse, cross-purpose access, revoked identity, break-glass use, and bypass through an alternate surface.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Is the same control enforced through UI, API, agent, tool, replay, administration, and support paths?

ASSURANCE ARCHITECTURE · AI BOUNDARY

Model and source content are untrusted inputs

Prompt injection, poisoned knowledge, unsafe output, and tool manipulation are data/control-confusion problems, not merely bad wording.

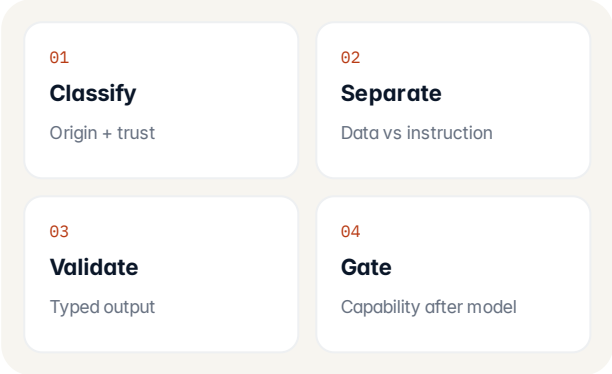
A source can contain instructions, a model can emit a tool-shaped string, and a retrieved passage can exploit trust granted to the orchestrator. None should acquire authority by appearing in context.

Mechanism. Label origin and trust, separate instructions from data structurally, restrict context projection, validate model outputs into typed artefacts, apply policy/capability after generation, sandbox tools, minimise secrets, and challenge anomalous source/action combinations.

Contract. The boundary records source/content identity, trust class, parser/transform, model/provider, prompt/instruction version, output schema, validation, policy decision, permitted tool operation, redaction, and incident signal.

Failure and recovery. Instruction/data crossover, malicious document, retrieved secret, schema escape, encoded payload, tool argument injection, provider leakage, unsafe fallback, or output filter bypass must fail before side effect.

Evidence and acceptance. Use a maintained adversarial corpus across source, retrieval, prompt, model output, tool, and multi-agent hand-offs; inspect denials, false positives, bypass attempts, and retained incident artefacts.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can any untrusted content directly expand capability, change policy, select a hidden tool, or expose a broader context?

ASSURANCE ARCHITECTURE · BOUNDARY

Technical proof does not settle legitimacy

Integrity, reproducibility, source provenance, policy replay, and human signatures make scrutiny possible; they cannot decide whether the purpose and outcome are acceptable.

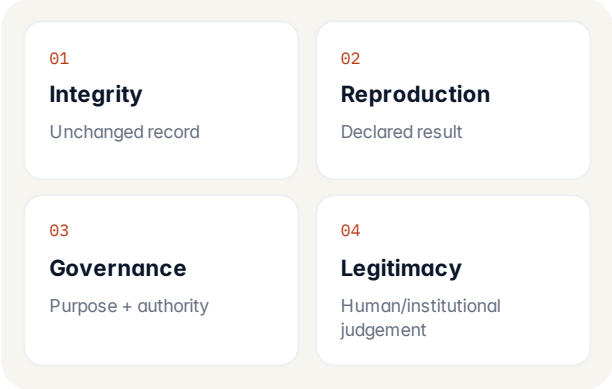
A perfectly reproducible harmful policy is still harmful. A signed approval may be uninformed. A cited source may be biased. A complete trace may document an unlawful or disproportionate process.

Mechanism. Pair technical evidence with organisational governance: purpose and impact assessment, source/policy ownership, affected-person information and contestability, competence and meaningful review, independent assurance, incident response, and authority to stop.

Contract. The decision record should link, not conflate, technical validity, domain correctness, policy approval, legal/risk assessment, human authority, impact findings, exceptions, appeals, and remediation.

Failure and recovery. Teams may use a green verifier as a universal approval, turn “aligned” into certified, or claim human oversight from a click. The governance design must reject these inference leaps.

Evidence and acceptance. For a sample case, list what every artefact proves and does not prove; obtain the relevant owners’ judgements; test explanation and contestability; and track accepted limitations.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

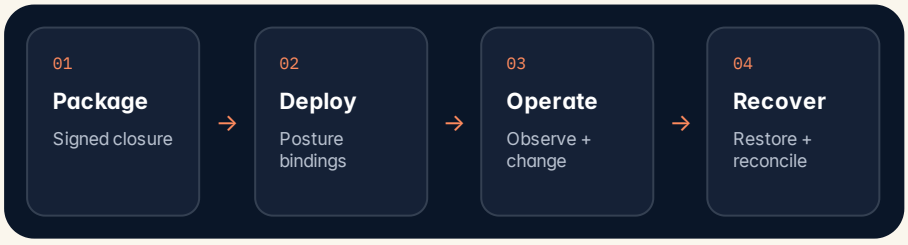
Can the organisation explain why the workflow should exist, not only prove that it ran as recorded?

06

DEPLOYMENT AND OPERATION

The workload is not production until it can be operated and recovered

Posture, packaging, updates, telemetry, continuity, interfaces, sizing, and exit decide whether the architecture survives outside a demonstration.



Every operational promise needs an owner, runbook, and installed exercise.

DEPLOYMENT AND OPERATION · MANAGED

Managed Fabric access still needs a precise boundary

The managed posture transfers infrastructure operation to Dweve within the described service boundary; it does not transfer customer purpose, source, authority, or outcome ownership.

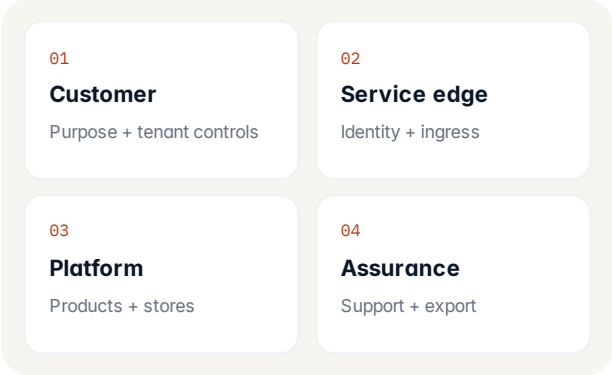
A European processing perimeter answers location at a high level. Architecture still needs tenant isolation, identities, data paths, model/provider routes, administration, retention, backups, support, evidence access, and exit.

Mechanism. Map browser/API/connectors to tenant/workspace controls, product services, stores, model/tool boundaries, observability, evidence, backups, administration, and exports. Name Dweve and customer owners at every boundary.

Contract. The current service description and agreement should settle region, isolation, encryption/key model, administrators/support, subprocessors/providers, retention/deletion, availability, incident, change, recovery, export, and customer obligations.

Failure and recovery. Unsupported data class, provider route, tenant-control error, service outage, unavailable support, retention mismatch, export gap, or customer misconfiguration must map to a runbook and notification path.

Evidence and acceptance. Use a representative tenant to test role removal, key/credential rotation, model/provider selection, deletion, export, incident escalation, evidence retrieval, restore commitments, and exit.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can both parties point to the same data/control diagram and responsibility matrix for the selected managed workload?

DEPLOYMENT AND OPERATION · LICENSED

Licensed operation is a production service the customer owns

Customer-controlled infrastructure and keys bring sovereignty only when the organisation can install, secure, observe, update, recover, and support the platform.

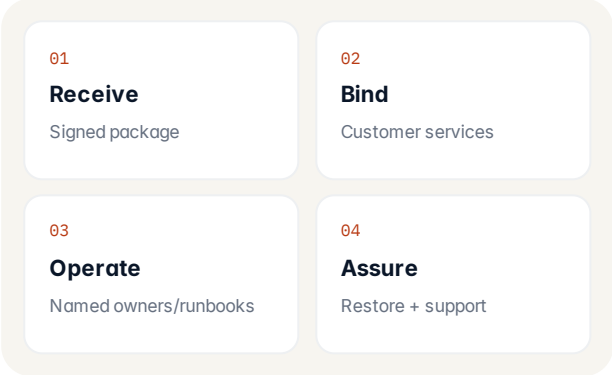
A delivered binary does not supply capacity planning, identity integration, backup, change control, incident ownership, evidence custody, or operational competence.

Mechanism. Define a supported environment profile; deploy signed packages with customer bindings; integrate identity/keys/storage/network/telemetry; assign product/source/policy/evidence owners; and operate against service objectives.

Contract. The deployment manifest covers component versions, infrastructure dependencies, sizing, network/ports, stores, keys, licences, models, configuration, health, logs/evidence, backup, update/rollback, support, and customer modifications.

Failure and recovery. Unsupported topology, missing local dependency, capacity exhaustion, certificate/key event, customer change, failed upgrade, backup gap, or unclear support boundary can invalidate platform claims despite successful installation.

Evidence and acceptance. Perform install, restart, load, key rotation, least-privilege administration, monitoring alert, incident, clean restore, update/rollback, verifier replay, supplier disconnect, and decommission.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Which operational duties move to the customer, and has each one passed an installed exercise rather than a document review?

DEPLOYMENT AND OPERATION · DISCONNECTED

Air-gapped operation closes the dependency set

No outbound dependency means licence, inference, work, evidence, verification, administration, recovery, and updates must have local or controlled-media paths.

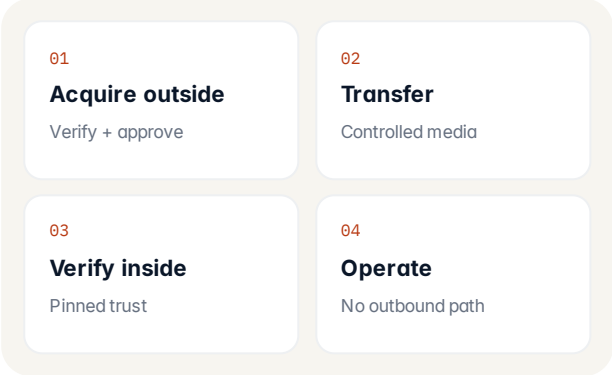
A firewall rule is insufficient if the runtime expects remote identity, time, model, telemetry, package, key, licence, support, or control-plane services.

Mechanism. Create an external acquisition zone and internal intake zone; verify signed manifests on both sides; package all runtime dependencies; use offline-signed licence material; operate local trust/time/identity; and control outbound diagnostics.

Contract. The air-gap profile lists every required package, model, policy, key/trust root, runtime, schema, verifier, licence artefact, infrastructure service, port, media workflow, update order, rollback, support bundle, and renewal condition.

Failure and recovery. Hidden DNS/NTP/OCSP/telemetry/model/package call, licence expiry path, absent trust update, clock drift, incompatible media, local capacity failure, unsupported diagnostics, or recovery requiring vendor access breaks closure.

Evidence and acceptance. Cold-start with cables absent; cross licence/time boundaries; run representative work and replay; restore locally; shuttle and roll back an update; export a minimised support bundle; and continue after Dweve disconnect.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can the complete installed workload operate through its failure and recovery cases with physically unavailable external services?

DEPLOYMENT AND OPERATION · SUPPLY CHAIN

Packaging is part of the architecture

Binaries, containers, models, policies, schemas, foundations, verifiers, licences, configuration, and documentation form one deployable and supportable closure.

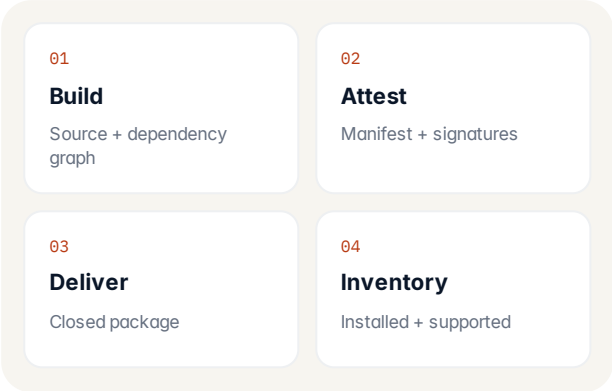
A signed application package can still pull mutable dependencies, unpinned images, model weights, compilers, system libraries, or verification tools after installation.

Mechanism. Build a release bill/manifest from source and dependency identities, sign immutable artefacts, record provenance and supported environment, scan and attest, package offline where needed, and publish compatibility and migration notes.

Contract. Each release item declares publisher, version/hash, format, signature, licence/rights, dependency graph, platform/CPU requirements, configuration migration, data/evidence compatibility, vulnerabilities/exceptions, rollback, and support status.

Failure and recovery. Tag mutation, missing transitive component, vulnerable or revoked artefact, incompatible model/schema, unsigned customer extension, build/non-build mismatch, or unpreserved old verifier can compromise operation and replay.

Evidence and acceptance. Rebuild or verify provenance where available, resolve the dependency graph with network disabled, compare installed inventory, scan/attest, reject a modified item, and roll back while retaining historic verification.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can an operator account for every executable, model, policy, schema, and verifier byte in the selected posture?

DEPLOYMENT AND OPERATION · CHANGE

Updates must preserve behavioural and evidential compatibility

A release can alter interfaces, model behaviour, rules, knowledge indexes, numeric results, evidence schemas, operational dependencies, and historic verification.

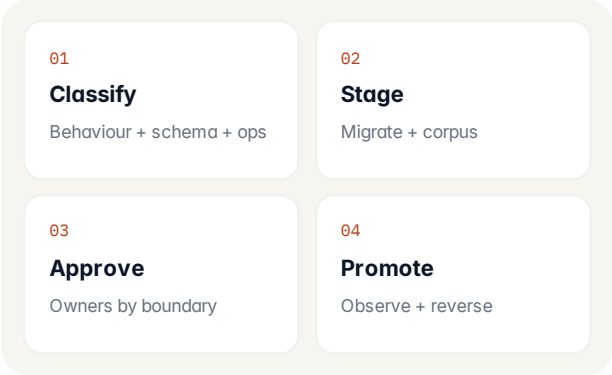
A green deployment health check says little about the customer workflow or retained record. Promotion requires a workload corpus and compatibility decisions.

Mechanism. Classify the delta; stage the full package; migrate copies; run normal/boundary/refusal/failure/replay/recovery cases; compare under declared modes; obtain owner approvals; promote reversibly; observe and reconcile.

Contract. The change record binds release and old/new manifests, migrations, acceptance corpus/version, results, accepted differences, security findings, approvals, window, operator, rollback point, and evidence-compatibility statement.

Failure and recovery. Schema drift, policy/model change, new dependency, performance regression, replay failure, irreversible migration, mixed-version queue, partial rollout, or rollback that loses new work requires a controlled hold/recovery.

Evidence and acceptance. Re-run the customer corpus, verify historic packets, canary by scoped workload, force rollback after new work exists, reconcile state, and retain the complete promotion/rollback record.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

What evidence shows the new release preserves the promises the organisation relied on, and which differences were consciously accepted?

DEPLOYMENT AND OPERATION · TELEMETRY

Observability must not become a shadow evidence store

Metrics, logs, traces, alerts, and profiles support operation; governed work evidence supports explanation and verification. Correlate them without copying sensitive payloads everywhere.

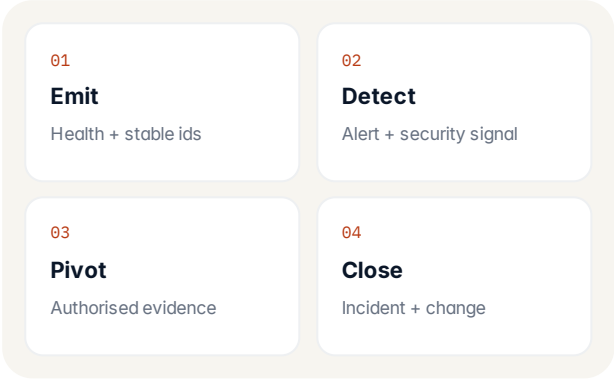
Payload-rich logging can breach purpose, retention, and air-gap boundaries. Payload-poor logging without stable identity can make incidents impossible to reconcile with work.

Mechanism. Emit service telemetry with work/task/component identifiers, state/error classes, latency/resource/budget, security and administrative signals; keep detailed work artefacts in purpose-controlled evidence stores; gate pivots between them.

Contract. Telemetry schema defines identifier, tenant/privacy handling, metric/logtrace and proof page class, severity, sampling, payload prohibition/redaction, retention, clock/order, exporter, access, alert, and evidence-reference policy.

Failure and recovery. PII/secret leakage, high-cardinality collapse, sampling of critical denial, clock drift, lost correlation, exporter outage, cross-tenant dashboard, alert flood, or inaccessible local telemetry can hide or create incidents.

Evidence and acceptance. Seed service, security, workflow, and evidence faults; follow alerts to authorised records; inspect payload absence; disable exporter; operate locally; test retention/access; and reconcile incident timeline.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can operations diagnose and contain failure without reading unrestricted work content or losing the link to the governed record?

DEPLOYMENT AND OPERATION · CONTINUITY

Backup, restore, and evidence preservation are different

Returning a database to service does not guarantee that historical work can be interpreted, verified, or reconciled.

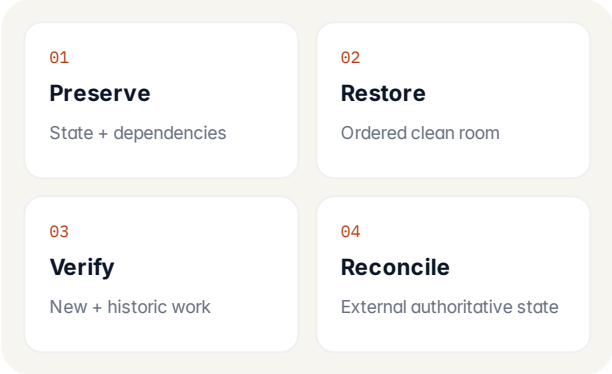
Recovery spans configuration and identities, workflow state, knowledge and indexes, model/policy artefacts, evidence and signatures, schemas/verifiers, external receipts, and queues.

Mechanism. Classify state and recovery order; set workload-level RPO/RTO; capture consistent checkpoints or reconciliation markers; retain verification dependencies; restore to clean infrastructure; replay safely; reconcile authoritative systems.

Contract. The continuity plan lists stores and owners, consistency boundary, backup frequency/immutability/encryption/key, dependency versions, restore sequence, queue/idempotency treatment, verification tests, external reconciliation, and evidence retention.

Failure and recovery. Inconsistent snapshot, lost key, restored stale policy, duplicate queue, missing index lineage, unavailable model/verifier, expired trust, external side effect, or untested manual step can return a misleading partial service.

Evidence and acceptance. Run clean-room restore with no undocumented operator state; process a safe new case, retrieve/verify a historical case, reconcile external systems, measure achieved RPO/RTO, and record exceptions.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

After restore, can the organisation both continue work safely and answer questions about work completed before the failure?

DEPLOYMENT AND OPERATION · INTERFACES

Integration errors need business-safe semantics

REST, gRPC, WebSocket, event, sidecar, CLI, and file surfaces differ in transport while sharing identity, authority, version, state, and evidence obligations.

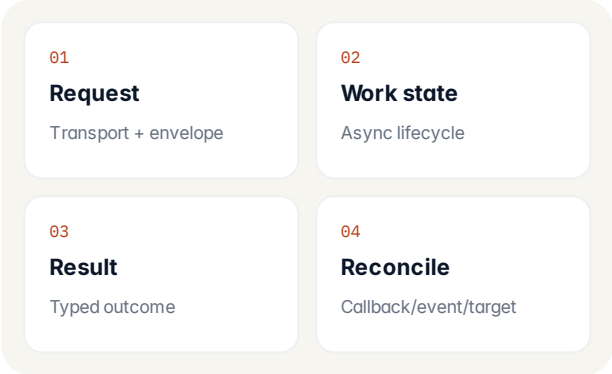
A stream is interrupted between tool calls, the pending step is replaced, and the audit record notes the steering event. The correction enters the next decision while the relevant context is still active.

Mechanism. Define a transport-independent result envelope and map it deliberately to status codes, stream frames, events, retries, dead-letter/repair queues, callbacks, and operator views.

Contract. Specify authentication, authorisation, schema/version, request/work/idempotency identities, sync/async lifecycle, ordering, pagination/streaming, timeouts/cancellation, typed errors, retry safety, evidence reference, and reconciliation.

Failure and recovery. Network timeout after effect, duplicate event, out-of-order callback, partial stream, schema downgrade, poison message, backpressure, lost cancellation, or dead-letter abandonment must produce an operable state.

Evidence and acceptance. Contract-test every surface with success/refusal/failure/duplicate/order/timeout/cancel/version cases; compare resulting work/evidence state; and exercise operator repair without data editing.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can a caller recover correctly from every exposed error without guessing whether the work or external effect happened?

DEPLOYMENT AND OPERATION · PERFORMANCE

Performance claims require workload-shaped sizing

Published counts and benchmarks can guide questions; production capacity follows the selected workflow, hardware, versions, concurrency, evidence, and service objectives.

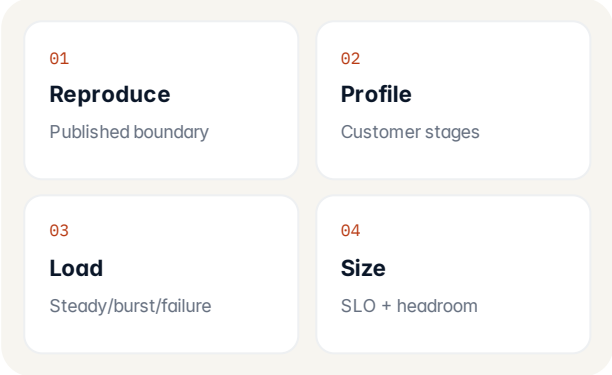
A retrieval QPS number, parser size tier, operation count, cold-start result, or energy-per-operation comparison measures a specific component boundary, not end-to-end throughput.

Mechanism. Model demand and case mix; profile stage latency/resource and queues; include models, solvers, parsing/indexing, tools, human waits, evidence, stores, network, and recovery headroom; test steady/burst/degraded states.

Contract. A measurement record binds claim, component/workload/corpus, hardware/ISA/backend, software/configuration, warm/cold state, concurrency/batch, duration/repetitions, metric/percentile, resource/energy method, raw output, and non-SLA limitation.

Failure and recovery. Baseline mismatch, cache/warmup, selective corpus, omitted preprocessing/evidence, average-only latency, failed-request exclusion, thermal/power difference, or benchmark-to-SLA promotion invalidates comparison.

Evidence and acceptance. Reproduce relied-on published figures in their stated scope, then run the customer corpus under expected and degraded load, inspect bottlenecks/queues/cost, and size from service objectives with margin.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Which measured component is the actual limiting resource for this workflow, and what happens when it saturates?

DEPLOYMENT AND OPERATION · EXIT

Portability includes history, proof, and operating knowledge

Moving binaries or exporting rows is not enough when workflows depend on sources, policies, model/agent manifests, evidence, keys, verifiers, and external mappings.

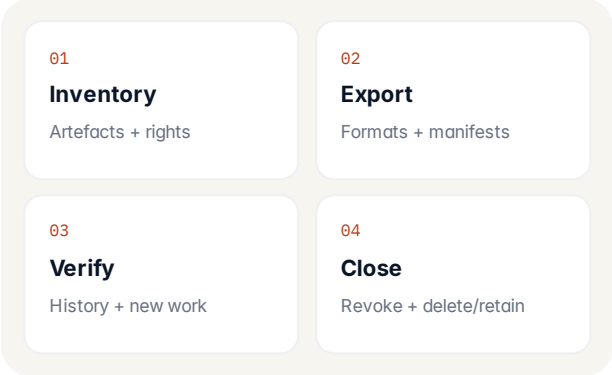
A platform can run under a customer licence yet remain difficult to leave if formats, schemas, licences, support rights, operational procedures, or verification dependencies are unavailable.

Mechanism. Inventory workload artefacts and rights; export documented formats plus manifests; validate completeness/integrity; remap environment bindings; import to destination; verify history; run shadow/new work; cut over and revoke old access.

Contract. The exit specification covers product/configuration, data/knowledge, workflows/policies, models/agents/tools, event/evidence/proof, keys/trust, schemas/verifiers, external identities/correlations, rights, support, deletion, and acceptance.

Failure and recovery. Proprietary mutable format, missing source right, unavailable historic verifier, lost identity mapping, external connector lock, incompatible schema, expired licence, incomplete deletion, or untrained operators blocks practical exit.

Evidence and acceptance. Perform an export/import rehearsal, count and hash artefacts, retrieve/verify a historical case at the destination, run a new case, reconcile systems, revoke source environment, and document residual dependencies.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

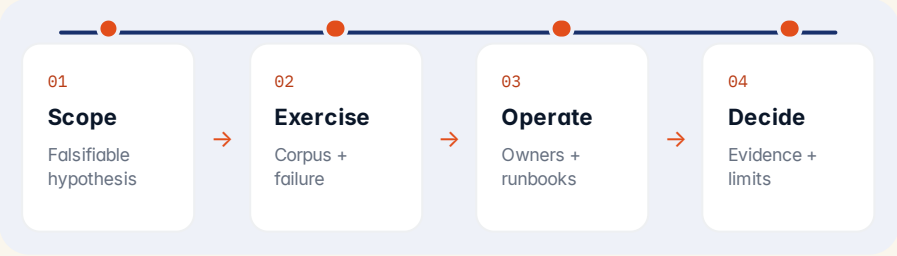
Could the customer continue and explain the selected workflow if Dweve were unavailable at the start of the exercise?

07

EVALUATION AND DELIVERY

A production proof is an evidence programme, not a feature tour

The final chapter turns the architecture into a scoped hypothesis, acceptance corpus, worked system, failure/security exercises, readiness gate, and diligence record.



The proof succeeds when it enables a defensible go, narrow, hold, or stop decision.

EVALUATION AND DELIVERY · SCOPE

Start with an architecture hypothesis that can fail

A proof should test whether a bounded workflow fits the platform and operating model, not demonstrate that a model can produce an attractive response.

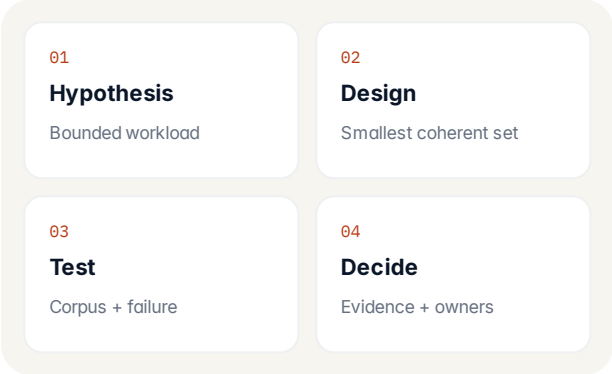
The architecture hypothesis names users, source/policy, cognition or exact work, team/action, human authority, systems, evidence, deployment, service objectives, and the reason each selected product is required.

Mechanism. Write the current-state problem and target outcome; draw system/context/data/control maps; select the minimum product/foundation set; list assumptions and rejected alternatives; translate public claims into customer acceptance tests.

A real trace verifies offline. You do not have to trust us to check it; you do not have to keep an account active to keep your records valid. Take the trace file, run the open-source verifier on your laptop, and you have your proof. We could disappear and the trace would still verify.

Failure and recovery. Expanding to a universal platform, using synthetic happy paths only, changing success criteria after results, hiding custom work, or omitting customer operating duties turns the proof into sales theatre.

Evidence and acceptance. Retain charter and changes, raw corpus/results, design decisions, defects/limitations, operating effort, accepted risks, and the production/hold/stop judgement by named owners.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

What result would cause the team to reject or narrow the architecture, and is that result actually tested?

EVALUATION AND DELIVERY · CORPUS

The acceptance corpus is an executable specification

Representative cases plus deliberately difficult boundaries define the workflow more reliably than feature requirements alone.

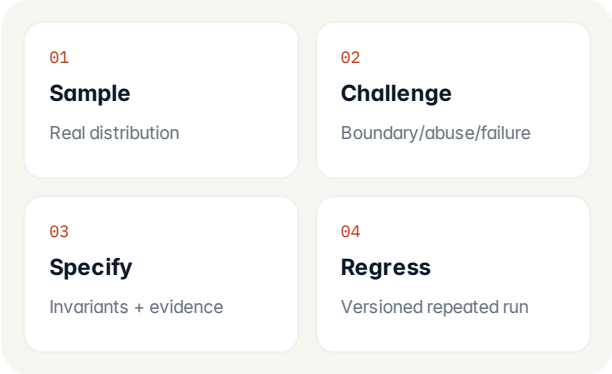
Historic normals show distribution; boundary, missing, conflicting, denied, malicious, failed, corrected, and appealed cases show whether the control model holds.

Mechanism. Select and minimise cases; classify risk/requirement; pin expected invariants/outcome classes; protect identities; separate configuration from evaluation; version and review; add incidents and newly accepted edge cases.

Contract. Each case records purpose, source/policy versions, input identity, permitted data, expected state/invariants, authority/tool outcome, evidence requirements, comparison rule, reviewer, provenance, and use restrictions.

Failure and recovery. Leakage into prompts/configuration, unrepresentative sampling, one exact prose expectation, ignored negative cases, stale policy, privacy breach, or changing labels to fit output invalidates evaluation.

Evidence and acceptance. Run unchanged corpus before and after material changes; retain outputs and manifests; adjudicate disagreement; report by case/risk segment; and keep failures visible with disposition.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Does the corpus test the architecture's refusal, authority, recovery, and evidence promises as thoroughly as its normal answer?

EVALUATION AND DELIVERY · WORKED SYSTEM

One regulated workflow should exercise the whole stack

An illustrative declined-credit explanation makes identity, policy time, exact calculation, specialist reasoning, human authority, notice, and evidence concrete without claiming a customer outcome.

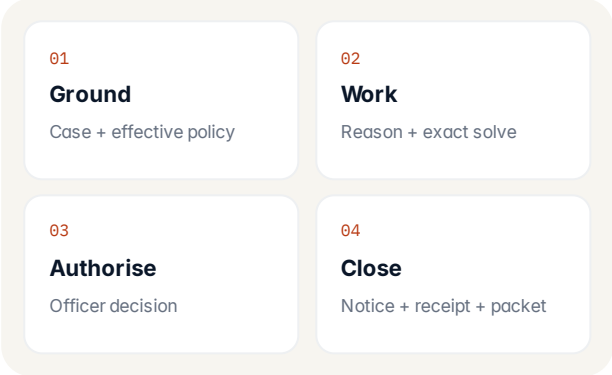
A case snapshot arrives with application identity and decision time. The workflow must explain a consequential outcome using the policy and facts that applied then, not current knowledge.

Loom can work with more than typed text. It can inspect documents, tables, images, audio, video, and structured information where the product supports them. It can notice layout, find names and dates, recognise patterns, inspect fields, and turn the input into something the next part of the system can use.

Contract. Bind applicant/case references, purpose, minimal projection, source/policy versions, solver/profile, expert/team/workflow manifests, officer authority, notice schema, tool/idempotency, target correlation, packet/evidence, retention/contestability.

Failure and recovery. Missing fact, conflicting policy, invalid solver proof, agent disagreement, denied tool, absent officer, target timeout, changed case, or replay mismatch routes to explicit stop/escalation/reconciliation states.

Evidence and acceptance. Retrieve case/policy lineage, solver artefact, cognitive and task records, officer view/decision, notice, target receipt, packet root and verifier; replay exact stages and present a plain reason/contest route.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Can a different authorised reviewer reconstruct the decision without production engineers or current policy replacing historical state?

EVALUATION AND DELIVERY · RESILIENCE

Failure injection reveals the real architecture

Deliberately break sources, models, agents, tools, storage, network, reviewers, targets, evidence, and recovery to observe containment and ownership.

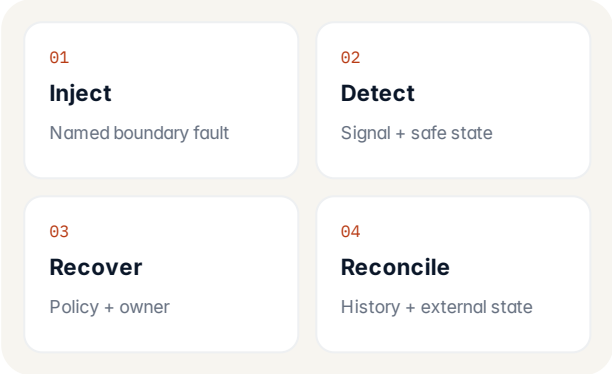
Happy-path sequence diagrams omit ambiguity after timeouts, partial writes, retries, failovers, stale versions, partitions, and human unavailability, the conditions that cause real incidents.

Mechanism. Build a failure matrix by boundary; inject one fault at a time and then combinations; observe state/error/evidence; verify retry/idempotency/compensation/escalation; reconcile targets; restore service and retained history.

Contract. Each experiment declares fault, location/time, expected detection, safe state, prohibited behaviour, owner/alert, recovery steps, evidence, maximum loss/outage, reconciliation, and acceptance threshold.

Failure and recovery. A test is invalid if it bypasses the production path, relies on hidden operator correction, measures only availability, or cleans state before reconciliation and evidence inspection.

Evidence and acceptance. Retain experiment and environment manifest, injected event, telemetry and work packet, operator decisions, actual recovery, residual state, target reconciliation, and runbook/change actions.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

When the component fails ambiguously, does the organisation know whether to retry, stop, escalate, compensate, or investigate?

EVALUATION AND DELIVERY · THREAT VALIDATION

Security testing follows abuse paths through products

Test end-to-end escalation routes from malicious source or caller to model, agent, tool, evidence, administrator, and deployment boundary.

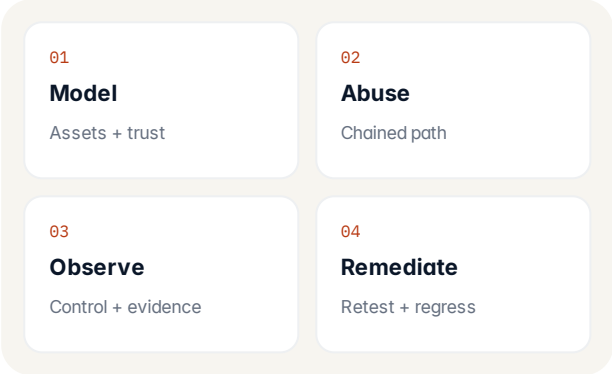
Component scanning and generic prompt tests miss chained attacks: source injection changes a plan, an agent selects a broad tool, a tool leaks a secret, and logging exports it.

Mechanism. Threat-model assets/actors/trust boundaries; build misuse cases; test identity/delegation, source poisoning, prompt/schema escape, tool capability, sandbox, model/provider, evidence disclosure, admin/support, supply chain, and egress.

Contract. Every finding records affected boundary/version/configuration, precondition, technique, observed state, data/action impact, detection/evidence, severity, owner, remediation, retest, and accepted residual risk.

Failure and recovery. Testing only the UI, using unrestricted test identities, excluding managed/support paths, or deleting attack traces can produce false confidence and prevent control improvement.

Evidence and acceptance. Demonstrate blocked and successful paths, confirm telemetry/evidence without secret spread, rotate/revoke affected credentials, retest correction, and add durable corpus/regression coverage.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Which untrusted input can reach the highest-consequence capability, and what independent controls stop that chain?

EVALUATION AND DELIVERY · OUTCOMES

Measure value beside quality, safety, and operating cost

Throughput, latency, token reduction, and hardware efficiency are useful only when paired with task correctness, correction, denial, impact, evidence, and total operation.

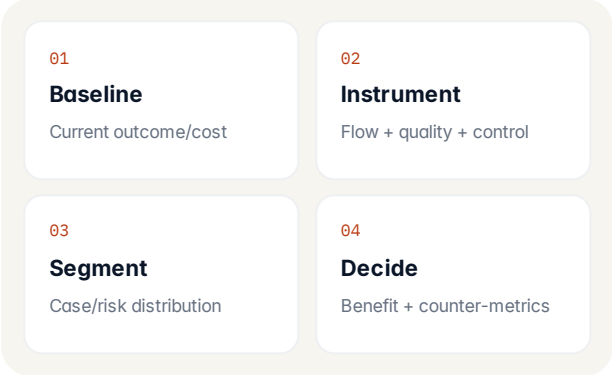
A platform can make work faster while shifting rework to reviewers, increasing unsupported denials, creating hidden operational toil, or harming a segment averaged out of headline results.

Mechanism. Baseline current flow and case mix; select outcome and counter-metrics; instrument platform and human stages; segment by risk/group/case type where legitimate; annotate changes; combine quantitative trends with case review.

Contract. The measurement plan defines numerator/denominator, population, time, owner, data source, collection method, exclusions, uncertainty, target and guardrail, segmentation, privacy, change annotations, and decision use.

Failure and recovery. Cherry-picked period, changed demand, unmeasured correction, survivor bias, missing failure cases, proxy target gaming, causality overclaim, or infrastructure/customer labour omission invalidates ROI.

Evidence and acceptance. Publish baseline and post-stage raw definitions, exceptions and operating effort; sample cases; compare shadow/assist/action stages; and let risk/domain owners decide whether change is beneficial.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Would the decision change if correction, reviewer time, incidents, evidence work, and customer infrastructure were fully counted?

EVALUATION AND DELIVERY · GATE

Production readiness is an ownership test

A workload is ready when technical, domain, security, operations, assurance, support, and affected-person responsibilities have owners and exercised procedures.

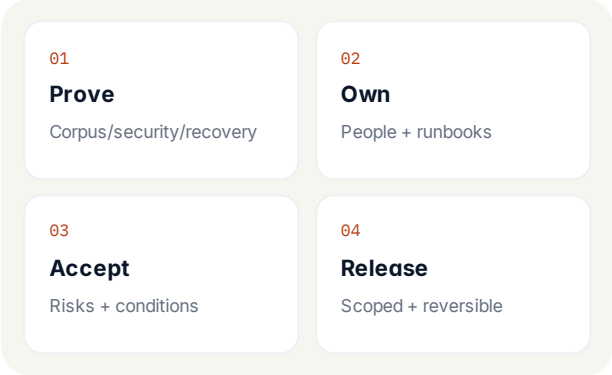
Passing functional tests while sources, policy changes, incident response, evidence disclosure, recovery, or user support remain unowned creates a brittle launch.

Mechanism. Review requirements and risks; close or accept gaps; verify owners/on-call/escalation; run corpus/load/security/failure/restore/replay; train reviewers/operators; validate notices/contestability; approve posture and commercial rights.

Contract. The readiness record lists scope/version, owners, test/evidence links, open risks, compensating controls, runbooks, service objectives, capacity, support, rollback/manual path, stop conditions, approvals, expiry and next review.

Failure and recovery. Conditional approvals with no owner/date, untested runbook, unavailable manual path, missing corpus coverage, capacity unknown, unresolved source right, or evidence packet gap means not ready.

Evidence and acceptance. Conduct a tabletop and live exercise spanning technical failure and domain consequence; retrieve a sample packet; execute rollback/manual route; and record named approval by each boundary owner.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Who has authority to stop production, and have they seen the evidence and failure modes needed to exercise it?

EVALUATION AND DELIVERY · SCALE

Expansion is a new architecture decision

More users, sources, jurisdictions, data, autonomy, tools, products, providers, targets, or deployments reopen different contracts and risks.

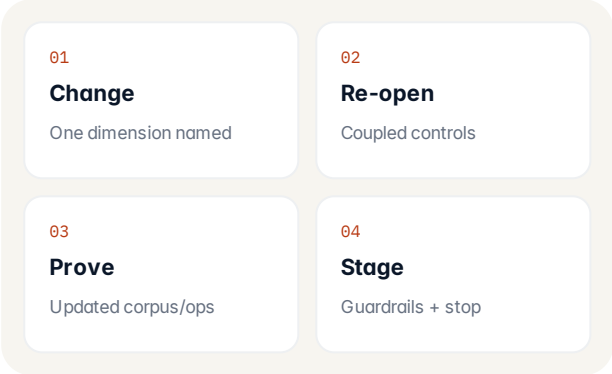
A pilot proves only its declared boundary. Scaling volume may need capacity; scaling autonomy changes authority; scaling sources changes governance; scaling posture changes operations.

Mechanism. Describe the changed dimension; perform impact/dependency analysis; re-open affected threat/source/policy/data/authority/evidence/operation contracts; update corpus and sizing; release segmentally with stop/rollback.

Contract. The expansion request names from/to scope, affected artefacts/owners, new data/action/consequence, required changes/tests, capacity/support, accepted risks, staged population, guardrails, stop conditions, and review date.

Failure and recovery. Treating expansion as configuration can inherit excessive permissions, stale policy, ungoverned sources, unsupported target, inadequate evidence/access, or operator overload.

Evidence and acceptance. Compare old/new architecture and manifests; run differential corpus/security/load/recovery; inspect early cases; track counter-metrics and incidents; and retain hold/narrow/reverse decisions.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Which assumption from the successful scope no longer holds after this expansion?

EVALUATION AND DELIVERY · BUYER EVIDENCE

Due diligence turns every strong claim into an artefact

Architecture, performance, security, availability, deployment, open-source, commercial, and compliance statements require different evidence and decision owners.

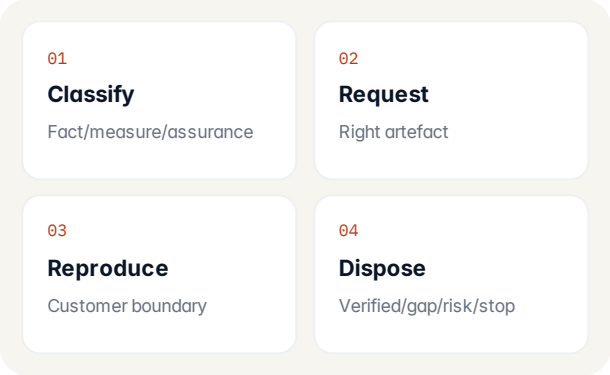
A broad request for “technical documentation” produces volume without resolving the workload-specific questions that determine production fitness.

Supported transcendental implementations are checked against high-precision references in documented domains. Accuracy claims are tied to operation-specific tolerances and property-based tests, so the proof is the harness, not a paragraph of marketing.

Contract. The claim register includes exact statement, class, website source/date, product/component/version, scope/conditions, decision relevance, evidence requested/received, reproduction result, conflict, owner, disposition, expiry and recheck trigger.

Failure and recovery. Screenshots without version, benchmark summaries without harness, “open” without repository/licence/release, “aligned” read as certification, or public price read as final rights leaves evidence weak.

Evidence and acceptance. For every relied-on claim, retain the direct artefact or installed test that proves it; mark contradicted/missing/accepted-risk items; and prevent launch conditions from disappearing into meeting notes.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Could an independent reviewer follow the decision from public claim to evidence, test result, limitation, and owner disposition?

EVALUATION AND DELIVERY · SOURCE HYGIENE

Known public-source conflicts must remain visible

Current website sources differ on some product/foundation counts, page placement, source-release status, and potentially volatile measurements or commercial language.

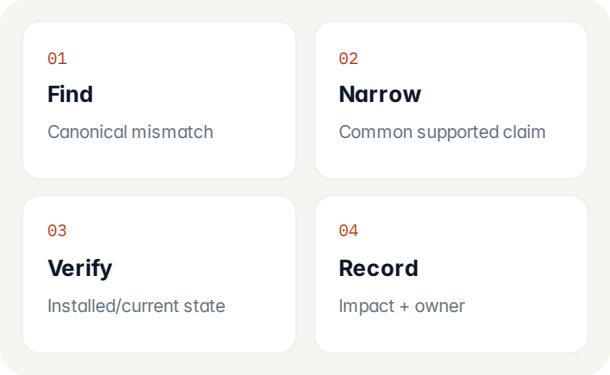
Silently choosing the more attractive number makes the handbook tidy while making diligence less reliable. The architecture usually works without an unqualified portfolio count.

Mechanism. Record both canonical locations; avoid the disputed claim or use the narrower common statement; name current products and components individually; bind benchmark to its route/method; recheck availability, source, pricing, assurance, and research status at publication/decision.

Contract. The conflict register records topic, page/source file/version/date, competing wording, impact, editorial treatment, product owner, required website reconciliation, and current decision assumption.

Failure and recovery. A count can alter licensed scope, an availability statement can be mistaken for shipping software, a benchmark can move outside its method, and a roadmapresearch overview item can enter production architecture.

Evidence and acceptance. Compare website page map, loaded English namespaces, installed/delivered manifests, repositories and current offer; retain unresolved conflicts and require written scope for the decision.



A technical reasoning path. Replace illustrative labels with current installed interfaces during design.

ACCEPTANCE QUESTION

Which unresolved website ambiguity could materially change architecture, rights, or acceptance, and who must reconcile it before production?

FIND A SUBJECT

Subject index

Page references point to the substantive explanation, worked example, control, boundary, or decision record, not merely to a passing label.

SUBJECT INDEX CONTINUED

Subject index · D–M

Terms are grouped alphabetically. Consecutive page references are collapsed into ranges.

SUBJECT INDEX CONTINUED

Subject index · M–R

Terms are grouped alphabetically. Consecutive page references are collapsed into ranges.

SUBJECT INDEX CONTINUED

Subject index · S–W

Terms are grouped alphabetically. Consecutive page references are collapsed into ranges.

CLOSING ARCHITECTURE RECORD

The next technical action is one bounded proof

Choose a representative workflow and require every responsibility, contract, state, failure, authority transition, artefact, and deployment claim in its path to become observable.

Start with the work identity and draw two diagrams: control flow from trigger to closure, and data flow from source to every store or recipient. Mark product owners, component dependencies, source and policy versions, model/expert/solver, agent tasks, tool capability, human authority, external receipt, evidence packet, and replay boundary.

Select the smallest coherent product and foundation set. Request current manifests and interfaces. Build the acceptance corpus with normal, boundary, refusal, malicious, failure, recovery, historical verification, and reconciliation cases. Run it in the intended deployment posture on representative infrastructure. Preserve raw results and limitations.

The production decision is then technical and organisational: does the workload meet its result, safety, authority, evidence, performance, continuity, rights, and operating requirements; do named owners accept the residual gaps; and can the system be stopped, recovered, inspected, and exited without inventing a story after the fact?

PROOF PACKAGE	one-bounded-workflow	
ARCHITECTURE	**maps + manifests + contracts**	INSPECT
WORK	**sources + policy + authority**	BIND
CORPUS	**normal + boundary + abuse**	RUN
RESILIENCE	**failure + restore + replay**	RUN
OPERATION	**owners + runbooks + capacity**	PROVE
EVIDENCE	**packet + offline check**	VERIFY
DECISION	**go/narrow/hold/stop**	RECORD

That record, not a slide deck, is the technical basis for production.

DWEVE.COM

Use the current website, installed artefacts, source/binaries where applicable, benchmark harnesses, assurance materials, and written agreement to settle the decision-date facts.