The Portable Warrant
Evidence at the Boundary

Flyxion
Independent Researcher

September 2026

Abstract

Modern software transports conclusions more readily than the conditions that justify them. A verifier returns success, an agent reports that a task is verified, or a theorem prover establishes a property of a translated representation; downstream components often receive only the conclusion. This paper develops Scope-Carrying Certificates (SCC) as an architecture for preserving the scope of such warrants across computational boundaries. The central criterion is property-relative: for a transformation $T : D \to Y$ to carry warrant for $\Phi : D \to \{0, 1\}$, Φ must be constant on the relevant fibers of T. We use this criterion to place temporal history truncation, structural graph flattening, and neuro-formal translation under one account of warrant non-preservation. An implementation discipline, the Anti-Assertion Mandate, requires authorization to consume independently resolvable evidence objects rather than epistemically meaningful Boolean claims. A cross-representation implementation experiment then reconstructs the same kernel in typed Python, asynchronous orchestration, Unix-style byte streams and stores, constrained BASIC profiles, and Forth stack machines. The experiment does not establish universal correctness; it identifies a small interface that survives substantial changes in representation. The resulting principle is simple: a warrant should not cross a distinction-erasing transformation unless independently checkable evidence establishes that the erased distinction is irrelevant to the property being warranted.

1 Introduction

Modern software engineering moves conclusions with extraordinary speed while often leaving behind the conditions that justified them. A verifier returns a green checkmark. An agent declares a task "verified." A theorem prover discharges a proof over an abstract translation. A downstream system receives a Boolean. The representation travels across computational boundaries; its epistemic boundary does not necessarily travel with it.

This asymmetry matters because verification is always verification of something, under some observation model, within some domain, using some proof obligation and trusted machinery. Once the conclusion is detached from those conditions, a locally justified statement can acquire an unjustified global reading. We call the operational pattern *certainty laundering*: the progressive migration of a scoped conclusion beyond the evidence that warranted it.

The Scope-Carrying Certificate (SCC) project began as an attempt to make this boundary explicit. It subsequently became an implementation experiment spanning a typed Python reference runtime, asynchronous multi-agent orchestration, Unix command-line filters and content-addressed stores, a deliberately fictional Research Unix/XENIX personality, constrained BASIC profiles, and Forth stack machines. These environments differ enough that conveniences available in one

disappear in another. What survived was not a particular class hierarchy or serialization format. It was a smaller rule: a warrant may cross a transformation only where the transformation preserves the distinctions on which the warranted property depends.

This paper develops that rule as an interface principle. The paper is explanatory rather than normative: the versioned SCC specifications in the accompanying software distribution define the software interface, while this text develops the calculus, rationale, limitations, and portability experiment.

2 Portable Certitude and the Fiber Criterion

Let D be an admitted domain, let $\Phi : D \to \{0, 1\}$ be a property of interest, and let $T : D \to Y$ be a transformation observed by a downstream component. The downstream component receives $T(z)$ rather than z itself.

Definition 1 (Property-relative warrant preservation)**.** *A transformation T preserves the observational information required for Φ on D when*

$$\forall z_1, z_2 \in D, \qquad T(z_1) = T(z_2) \Longrightarrow \Phi(z_1) = \Phi(z_2).$$

Each $y \in Y$ induces a fiber $T^{-1}(y) = \{z \in D : T(z) = y\}$. The condition therefore says that Φ is constant on every relevant fiber. Equivalently, there exists a function $\widehat{\Phi}$ on the image of T such that

$$\Phi = \widehat{\Phi} \circ T$$

on the admitted domain. The criterion is deliberately property-relative. A projection may erase distinctions irrelevant to one property and indispensable to another.

This observation separates information loss from failure. Abstraction is useful precisely because it discards information. The problem begins when a system carries a conclusion across an abstraction without establishing that the discarded information is irrelevant to that conclusion.

3 The Warrant Reconstruction Limit

Proposition 1 (Warrant Reconstruction Limit)**.** *Given $T : D \to Y$ and $\Phi : D \to \{0, 1\}$, if there exist $z_1, z_2 \in D$ such that $T(z_1) = T(z_2)$ and $\Phi(z_1) \neq \Phi(z_2)$, then no deterministic decision procedure whose only observation is $T(z)$ can correctly recover $\Phi(z)$ for every $z \in D$. A randomized procedure cannot be correct with probability one on both such inputs without additional information.*

Proof. Let $T(z_1) = T(z_2) = y$. A deterministic procedure receiving only y must return the same value in both cases, while the required values differ; it therefore fails on at least one. A randomized procedure receives the same observational input and therefore induces the same output distribution in both cases. No single distribution can assign probability one to two different required outputs. Randomization may alter error rates, but it cannot reconstruct the erased distinction. □

The obstruction is informational rather than computational. Increasing downstream model capacity cannot recover a distinction absent from the observation unless additional evidence enters through another channel.

4 Three Ways to Lose a Distinction

The project encountered three failure classes that initially appeared unrelated.

4.1 Temporal truncation

A system compresses a history $H_{\leq t}$ into an operational state M_t. If two histories produce the same retained state while a relevant property differs, then the retained state is insufficient:

$$M(H_1) = M(H_2), \qquad \Phi(H_1) \neq \Phi(H_2).$$

This includes cases where current behavior depends on trajectory, prior delegation, accumulated commitments, or historical context omitted by a rolling summary.

4.2 Structural flattening

Let a workflow be a directed labeled graph $G = (V, E, \lambda_V, \lambda_E)$. A monitor that preserves events while discarding incidence can map distinct workflows to the same flattened representation $\widehat{G}$. Reachability, cycles, delegation depth, fan-out, path multiplicity, and indirect flows are properties of relations among events, not merely of isolated event records. Local checks can therefore all succeed while a path-level invariant fails.

4.3 Representational translation

Suppose a translator maps a source object into a formal target,

$$X \xrightarrow{\tau} Y.$$

A terminal verifier may establish $P_Y(\tau(x))$. Migrating that result back to a source-language claim $P_X(x)$ requires an independently justified bridge, for example

$$P_Y(\tau(x)) \Longrightarrow P_X(x)$$

over a declared source domain. Sound verification of Y does not by itself establish faithful translation from X.

The three axes can therefore be written

$$H \to M, \qquad G \to \widehat{G}, \qquad X \to Y.$$

Their mechanisms differ; their failure condition is the same. A transformation has identified states that the warranted property still needs to distinguish.

5 A Calculus of Warrant Preservation

The fiber criterion describes observational adequacy, but an operational system needs a vocabulary for the warrant that is being transported. We model a warrant at a boundary as a tuple

$$W = \langle \Phi, T, D, E, P, C, t, \nu \rangle,$$

where Φ is the property being relied upon, T is the observation or transformation through which the consumer sees the source state, D is the admitted domain, E is a set of environmental assumptions, P identifies independently checkable proof material, C records compositional and delegation constraints, t is the validity interval, and ν identifies the relevant implementation or specification version. The tuple is not intended to imply that every implementation must use this

exact record layout. It identifies the semantic coordinates that disappear when a rich verification result is reduced to a bare success bit.

A warrant is therefore not simply a proposition. It is a proposition together with an account of why a particular consumer may rely on it at a particular boundary. This distinction is useful because logical truth and operational admissibility need not coincide. A theorem may remain true while the certificate that once permitted a particular action has expired, crossed its delegation limit, or arrived at a consumer outside its declared scope.

Definition 2 (Boundary-relative warrant)**.** *Let b be an execution boundary and a a proposed action. A warrant W is* admissible at b for a *only if its evidence is authentic, its current state lies in D, its assumptions E are supported, its temporal and version constraints hold, and the derivation from its issuing boundary to b is permitted by C.*

This makes explicit two implications that are often conflated. First, evidence may establish a proposition without establishing authorization to act on it at every later boundary. Second, failure to derive authorization says less than falsity of the proposition. The calculus is intentionally asymmetric: positive authorization requires a derivation; non-derivation requires no counter-proof.

5.1 Restriction and monotonicity

If $D' \subseteq D$, a proof of fiber constancy on D also establishes fiber constancy on D'. Domain restriction can therefore repair an otherwise invalid transformation, provided the restriction is itself explicit and checkable. The reverse does not hold: a bridge proved only on D' cannot be silently widened to D. This simple asymmetry is one reason SCC binds domain semantics into proof identity rather than treating a domain name as administrative metadata.

The same principle applies to consumers and delegation depth. Attenuating scope is generally safe with respect to the certificate's own authority: a certificate valid for consumers $\{A, B\}$ can be restricted to $\{A\}$ without acquiring new authority. Widening the consumer set, extending validity, increasing delegation depth, or replacing the admitted domain requires a new derivation. SCC therefore treats scope widening as evidence-producing work rather than as record editing.

5.2 Composition is a new proof obligation

Suppose $T_1 : D \to Y$ preserves Φ and $T_2 : Y \to Z$ preserves some property Ψ. It does not follow merely from those two facts that $T_2 \circ T_1$ preserves Φ. The second proof may concern a different property or a domain that does not contain the image relevant to the first transformation. Composition requires compatibility among the property, domains, and bridge obligations.

A sufficient condition is the existence of a property Φ_Y such that

$$\Phi = \Phi_Y \circ T_1$$

on D, together with a property Φ_Z satisfying

$$\Phi_Y = \Phi_Z \circ T_2$$

on $T_1(D)$. Then

$$\Phi = \Phi_Z \circ T_2 \circ T_1.$$

The point is elementary but operationally important: compositional warrant is derived from compatible witnesses, not from the fact that two local checks independently returned success.

5.3 Joins and relational properties

Parallel workflows introduce another distinction. Certificates W_A and W_B can each justify local propositions while failing to justify a proposition about their combination. In general,

$$W_A \models P, \qquad W_B \models Q$$

does not by itself yield an authorization for a join whose required property is relational in the two branches. The join may depend on whether the branches share provenance, whether their outputs are mutually consistent, or whether combining them creates a path forbidden by a structural invariant. SCC therefore models a join as a fresh boundary rather than as bookkeeping around two already-approved actions.

6 From Claims to Evidence Objects

Implementation exposed a second-order form of laundering. Fields such as `proof_verified=true`, `translation_faithful=true`, or `composition_valid=true` appear to attach assurance to an object, but each field is itself an epistemically meaningful assertion. If authorization trusts the field without resolving what established it, the original problem has merely moved into metadata.

SCC therefore adopts the *Anti-Assertion Mandate* as an architectural invariant:

> Authorization consumes evidence objects, not claims about evidence.

A proof obligation resolves to an artifact binding a property identity, projection identity, admitted domain, verifier implementation, and proof material. A composition claim resolves to a composition artifact. A representational claim resolves through translation, terminal-proof, and property-relative bridge artifacts. A simplified neuro-formal chain is

$$h(x) \xrightarrow{C_\tau} h(y) \xrightarrow{C_V} P_Y(y) \xrightarrow{P_{bridge}} P_X(x).$$

The runtime authenticates the links rather than accepting a Boolean summary of them.

This also motivates non-transitive positive authorization. A certificate sufficient for one consumer at one boundary does not automatically become sufficient for a downstream consumer. Delegation and joining are new transformations and may require new evidence.

7 The Three-Gate Runtime

The reference runtime separates three questions that are easy to collapse:

1. Is the proof artifact authentic and bound to the property, projection, domain, and verifier identity claimed by the certificate?
2. Is the current state inside the admitted runtime domain and are declared environmental assumptions presently supported?
3. Is the proposed delegation, join, or composition authorized by independently resolvable evidence?

Only after these gates succeed is a positive disposition derivable. This matters because proof validity, current applicability, and compositional inheritance are different propositions.

The runtime also records granular reasons for non-derivation, including domain mismatch, stale evidence, unauthorized consumer, delegation-depth exhaustion, missing composition evidence, and unverified translation faithfulness. Diagnostics are not merely ergonomic: they prevent distinct epistemic failures from being collapsed into an opaque rejection bit.

8 Evidence Lifecycle and Boundary Semantics

A scope-carrying certificate is useful only if its lifecycle is as explicit as its contents. The reference architecture distinguishes issuance, resolution, consumption, derivation, expiration, and replay. Each stage answers a different question.

Issuance. An issuer constructs a certificate that names already-existing evidence. Issuance does not make the evidence true. In particular, an issuer cannot create a valid proof merely by embedding the hash of a nonexistent or untrusted artifact. The trusted registry resolves the identifier and checks its semantic bindings.

Resolution. The consumer resolves proof, domain, translation, and composition identities against trusted stores. Content addressing is useful here because it turns accidental mutation into identity change, but hashing is not itself verification. A digest establishes byte identity relative to a hash function; it does not establish that those bytes prove the advertised theorem.

Consumption. Consumption occurs at a named boundary with a current state, consumer identity, clock, and environmental evidence. This is where a certificate can be authentic yet inapplicable. The distinction between authenticity and applicability is essential for stale certificates and domain escape.

Derivation. When a warrant is delegated or composed, the downstream certificate records parent identities and a derivation artifact. Positive dispositions are non-transitive by default. A downstream component may inherit data without inheriting the authority to rely on the upstream conclusion.

Expiration and revocation. Temporal validity is represented as an applicability condition rather than a metaphysical claim that a theorem stops being true. Expiration says that the evidence package no longer warrants this operational use without refresh. A production system may additionally support revocation; the reference calculus requires only that whatever revocation mechanism is declared become part of the evidence-resolution boundary rather than an invisible side channel.

Replay. A recorded decision is reproducible only to the extent that its relevant epistemic state was recorded. The Research Unix tools therefore distinguish replay of a decision from re-execution of the world that originally produced it. A ledger can demonstrate that the same recorded evidence yields the same gate result; it cannot prove that omitted external facts were unchanged.

This lifecycle motivates a useful conservation rule: no stage may increase the semantic scope of a warrant merely by copying, serializing, renaming, or transporting it. Scope can be attenuated mechanically. Scope expansion requires evidence.

9 Threat Model and Non-Goals

SCC is designed against a family of architectural errors rather than against an omnipotent adversary. The relevant failures include stale or mismatched proof objects, certificates presented outside their admitted domain, downstream consumers treating local authorization as transitive, joins performed without relational evidence, lossy translations whose terminal proofs are migrated back to richer source claims, and metadata that asserts verification without making its basis resolvable.

The framework assumes that the trusted checker, canonicalizer, cryptographic primitives, and configured roots of trust behave according to their specifications. It does not defend against arbitrary compromise of the entire trusted computing base. Nor does it infer the complete set of transformations occurring in a system. An undeclared preprocessing step can invalidate the analysis while remaining invisible to SCC. The architecture therefore improves the explicitness of trust; it does not abolish trust.

SCC is also not a general hazard classifier. Its runtime can establish that an action lacks a declared warrant without deciding that the action is physically, morally, or legally unsafe. Conversely, a valid SCC establishes only the property encoded by its proof obligations. If the chosen property is too weak, the certificate can be perfectly valid and operationally inadequate. The specification of Φ is therefore part of the trusted problem statement.

Finally, SCC is not intended to replace proof assistants, information-flow systems, capability mechanisms, policy engines, or provenance stores. It can consume evidence produced by such systems and constrain how conclusions derived from them cross later boundaries. Its target is the seam between verification contexts.

10 The Portable Kernel

The Python implementation uses typed models, registries, asynchronous execution, structured serialization, and cryptographic libraries. The portability exercise asked which of these facilities are essential. The resulting kernel can be summarized as

$$\boxed{\text{canonicalize} \rightarrow \text{resolve} \rightarrow \text{check domain} \rightarrow \text{check ancestry} \rightarrow \text{derive} \rightarrow \text{act} \rightarrow \text{record.}}$$

Hashing accompanies the sequence by binding identities to exact representations.

The table is not evidence that all environments provide equivalent assurance. It records what each reconstruction makes explicit. The BASIC profile, for example, deliberately refuses to simulate a missing cryptographic primitive. A constrained implementation can report:

```
CRYPTOGRAPHIC PRIMITIVE NOT RESIDENT
EXTERNAL VERIFIER REQUIRED
WARRANT NOT DERIVED
NULL ACTION
```

This is not a hidden deficiency. It is the intended epistemic behavior: an implementation unable to establish required evidence does not manufacture permission.

Table 1: Cross-representation portability matrix.

Environment	Representation	Missing convenience	Preserved obligation
Python	Typed evidence objects	—	Full reference semantics
Async orchestration	Concurrent events	Global sequential state	Ancestry and composition
Research Unix personality	Byte streams and files	In-process object identity	Canonical evidence and replay
BASIC/16 profile	Flat records and arrays	Resident cryptography	Explicit abstention
Forth profile	Stack and words	High-level object model	Warrant-before-execute

11 Forth and the Shape of Authorization

Forth makes the execution boundary unusually visible. The essential stack effect can be written

```
( certificate state -- warranted? reason )
```

Verification words transform evidence into an authorization result; `EXECUTE` remains outside that pure derivation and is unreachable through the guarded path until the warrant succeeds. The port therefore strips away an object model while retaining the distinction between warrant derivation and action execution.

The significance is methodological rather than nostalgic. Reconstructing the same semantics in a stack language tests whether SCC depends on conveniences of its reference representation. It does not prove universality, but it makes accidental dependencies easier to see.

12 The Operating-System Interpretation

The Research Unix personality organizes evidence beneath virtualized `/etc/scc`, `/var/lib/scc`, and `/var/log/scc` hierarchies and exposes inspection through filters such as `scc-check`, `scc-lint`, `scc-fsck`, `scc-dump`, and `scc-replay`. This suggests an operating-system analogy:

$$\text{file descriptor : resource access} \quad :: \quad \text{SCC : warranted action.}$$

The analogy is limited. SCC is not a conventional operating-system capability. The useful resemblance is that a name or object is not identical to unrestricted authority; access is mediated by a scoped intermediary whose interpretation depends on context.

The distribution also contains an explicitly fictional XENIX/386 personality. It is a presentation and compatibility metaphor, not a claim of derivation from Microsoft or SCO source code and not evidence of execution on historical XENIX. Its "scope fault" vocabulary treats a certificate as if it were a logical segment descriptor

$$S = \langle base, limit, rights, generation \rangle,$$

where the fields denote originating evidence, scope constraints, permitted use, and freshness. Crossing that logical boundary without a derivation produces a scope fault handled as warrant non-derivation rather than a processor exception.

13 NULL_ACTION as Operational Abstention

Many safety architectures blur three propositions:

$$\neg Warranted(a), \qquad Unsafe(a), \qquad \neg Execute(a).$$

SCC separates them. In particular,

$$\neg Warranted(a) \not\Rightarrow Unsafe(a),$$

while an execution boundary may enforce

$$\neg Warranted(a) \Rightarrow DoNotExecute(a).$$

The distinction gives NULL_ACTION a positive operational role. It can acquire telemetry, refresh expired evidence, request a composition proof, reconstruct missing history, obtain a richer representation, or wait for an assumption to become observable. The proposed action may then be reconsidered. Abstention is therefore a continuation under insufficient evidence, not necessarily a declaration of intrinsic danger.

14 Warrant Algebra

The preceding calculus treats a warrant as a structured object whose applicability depends on a property, transformation, domain, evidence basis, consumer, time, and implementation identity. That structure admits a small algebra. The purpose of the algebra is not to assign a scalar "strength" to evidence. It is to distinguish operations that can only reduce an existing scope from operations that create a genuinely new proof obligation.

Definition 3 (Scope preorder)**.** *For warrants W_1 and W_2 concerning the same underlying property family, write $W_1 \preceq W_2$ when every use authorized by W_1 is also within the declared scope of W_2, and W_1 introduces no weaker evidence requirement than W_2. The relation compares authorization scope, not truth.*

The simplest operation is *attenuation*. Restricting a domain, shortening a validity interval, reducing the permitted consumer set, or lowering the allowed delegation depth yields a warrant no broader than its parent:

$$W' \preceq W.$$

Such an operation may be mechanically checkable because it removes possible uses. By contrast, *widening* is not derivable from possession of the narrower warrant alone. If a transformation proposes $W \prec W'$, some new evidence must justify the additional cases. Copying, serialization, renaming, or repeated delegation cannot supply that evidence.

This gives SCC a conservation principle:

Proposition 2 (No authority by transport)**.** *Let $\mathcal{T}$ be a sequence consisting only of representation-preserving transport operations—copying, serialization, storage, retrieval, and identity-preserving forwarding. If no step introduces independently authenticated evidence, then $\mathcal{T}$ cannot soundly derive a warrant whose declared scope strictly contains that of its input warrant.*

Proof. The transport operations introduce no observation capable of distinguishing a newly admitted case from one previously outside scope. A strict scope expansion would therefore authorize at least one use not justified by the input evidence. The expansion requires an additional premise and cannot follow from transport alone. □

14.1 Composition

Sequential composition is not ordinary transitivity. Suppose $T_1 : D \to Y$ preserves Φ on D and $T_2 : Y \to Z$ preserves some target property on a domain $D_2 \subseteq Y$. A downstream warrant for $T_2 \circ T_1$ requires more than two independently valid certificates. The image of the first transformation must fall inside the admitted domain of the second, property identities must align, environmental assumptions must remain compatible, and the composition must preserve the distinctions required by the final claim.

We write an authenticated composition operation as

$$W_2 \odot_P W_1 \Downarrow W_{21},$$

where P is the composition artifact. The notation is intentionally partial: the operation is undefined when the proof artifact is absent or its bindings do not match.

Proposition 3 (Property-aligned sequential composition)**.** *Let $T_1 : D \to Y$ and $T_2 : Y \to Z$. Suppose there exist functions $\Phi_1 : D \to \{0,1\}$, $\Phi_2 : Y \to \{0,1\}$, and $\Phi_3 : Z \to \{0,1\}$ such that*

$$\Phi_1 = \Phi_2 \circ T_1 \quad \textit{and} \quad \Phi_2 = \Phi_3 \circ T_2$$

on the admitted domains, with $T_1(D) \subseteq D_2$. Then

$$\Phi_1 = \Phi_3 \circ (T_2 \circ T_1)$$

on D.

Proof. For every $z \in D$, substitution gives $\Phi_1(z) = \Phi_2(T_1(z)) = \Phi_3(T_2(T_1(z)))$. □

The proposition is deliberately elementary. Operational difficulty lies in establishing that the two certificates actually refer to these same functions and domains. SCC therefore treats the theorem as a condition to be evidenced rather than inferred from matching human-readable labels.

14.2 Joins

Joins are more dangerous because the property of a pair need not decompose into properties of its members. Let W_A warrant $P(a)$ and W_B warrant $Q(b)$. In general,

$$W_A, W_B \nvdash W_{A \bowtie B}$$

for a relational property $R(a, b)$. Two individually admissible records can jointly disclose a restricted fact; two independently authorized actions can create a forbidden path; two harmless partial observations can identify a person when combined. The join is therefore itself a transformation with its own fibers.

A join certificate must bind the parent warrants, the join operator, the relational property, and the admitted pair-domain. This is why SCC separates delegation proofs from join proofs: a certificate may be validly forwarded and still be invalidly combined.

14.3 Algebraic non-laws

Several tempting algebraic laws are intentionally absent. Warrant composition is not generally commutative because transformation order can matter. It is not generally idempotent because replay at a later time may change applicability. It is not total because missing evidence makes derivation undefined. And it is not globally transitive because consumer, domain, and property bindings can change at every boundary. These non-laws are not inconveniences to be optimized away; they encode where new evidence is required.

15 Temporal Warrant and Historical Extension

The temporal axis deserves separate treatment because time changes both the world and the evidence available about it. Let $H_{\leq t}$ denote a history prefix and let a be a proposed action. An operational warrant is a relation

$$\mathcal{W}(H_{\leq t}, a, E_t) = 1,$$

where E_t is the evidence package available at time t. A warrant at one prefix need not persist under extension:

$$\mathcal{W}(H_{\leq t}, a, E_t) = 1 \not\Rightarrow \mathcal{W}(H_{\leq t+\Delta}, a, E_{t+\Delta}) = 1.$$

The implication can fail even when every previously recorded fact remains true. A credential can expire, an environmental assumption can change, a delegation can be revoked, or a new event can make a previously irrelevant historical distinction operationally relevant.

15.1 Archival truth and operational sufficiency

SCC therefore distinguishes an archival proposition from its current operational use. A theorem proved yesterday does not become false merely because a certificate expires. Expiration instead says that the recorded proof package is no longer sufficient for this action under the declared freshness policy. The same distinction applies to telemetry. "Sensor s reported value v at t" can remain archivally true while ceasing to warrant an action at $t + \Delta$.

This distinction prevents a common category error in revocation systems: revocation does not retroactively falsify history. It changes which historical evidence is admissible for a present decision.

15.2 Monotone and non-monotone evidence

Some evidence is monotone with historical extension. A content hash of an immutable artifact, once correctly computed, need not become stale merely because time passes. Other evidence is intrinsically non-monotone: current location, process ownership, resource availability, membership in a revocable role, or absence of a later contradictory event. A certificate should therefore not use a single undifferentiated timestamp as if all premises aged identically.

Let $E = E^+ \cup E^\pm$, where E^+ denotes premises whose applicability is monotone under the declared model and $E^\pm$ denotes premises requiring freshness or revalidation. Then historical extension may preserve E^+ while invalidating $E^\pm$. A temporal SCC can record these classes separately, making refresh obligations explicit.

15.3 Causal markers and sufficient summaries

A system need not retain complete history if it can prove that a summary is sufficient. Let $M_t = m(H_{\leq t})$. The temporal fiber condition is

$$m(H_1) = m(H_2) \Longrightarrow \Phi(H_1, a) = \Phi(H_2, a)$$

for all admitted histories. When this holds, M_t is a sufficient operational summary for the property. When it does not, the system may preserve selected causal markers rather than the entire trace. The design problem is therefore not "store everything" but "retain or certify every distinction on which the declared action property depends."

15.4 Replay

Replay has two meanings that should not be conflated. *Decision replay* re-evaluates a recorded epistemic state and asks whether the same checker derives the same disposition. *World replay* would require recreating the external conditions that generated that state. SCC's ledger supports the first. It cannot guarantee the second. Deterministic replay is consequently evidence about the implementation's treatment of recorded premises, not evidence that the premises were complete descriptions of the historical world.

16 A Worked Neuro-Formal Counterexample

The representational axis becomes clearest when the terminal proof is entirely correct. Consider a source language whose objects describe both a data origin and a destination:

$$X = \{(o, d) : o \in \{public, restricted\},\ d \in \{internal, external\}\}.$$

The source property is

$$P_X(o, d) = \neg(o = restricted \wedge d = external).$$

Now define a deliberately lossy translator

$$\tau(o, d) = d.$$

The formal target remembers only whether the destination is internal or external. Let the target verifier prove the perfectly valid proposition

$$P_Y(y) = (y = external).$$

For

$$x_0 = (public, external), \qquad x_1 = (restricted, external),$$

we have

$$\tau(x_0) = \tau(x_1) = external,$$

but

$$P_X(x_0) = 1, \qquad P_X(x_1) = 0.$$

A proof assistant can prove $P_Y(external)$ with complete rigor. That proof does not select between x_0 and x_1, because the translation has erased the origin field before formal verification begins.

The failure is not "the theorem prover made a mistake." The theorem prover established exactly the proposition it was given. Nor is the failure necessarily "the translator is bad." The translator may be adequate for properties depending only on destination. The failure occurs when a warrant about P_Y is migrated into a claim about P_X without a bridge theorem whose domain makes the implication valid.

Suppose we restrict the admitted source domain to

$$D_{public} = \{(public, internal), (public, external)\}.$$

On this domain, origin is constant. A property such as

$$P'_X(o, d) = (d = external)$$

factors through τ, and a bridge theorem can legitimately establish

$$P_Y(\tau(x)) \Longrightarrow P'_X(x), \qquad x \in D_{public}.$$

The same translator has moved from inadequate to adequate because the property and admitted domain changed. Faithfulness is therefore neither an intrinsic Boolean attribute of τ nor a global endorsement of the translator. It is a relation among transformation, property, and domain.

The SCC evidence chain makes these bindings explicit:

$$h(x) \xrightarrow{C_\tau} h(y) \xrightarrow{C_V} P_Y(y) \xrightarrow{P_{bridge}} P_X(x).$$

C_τ binds source and target hashes, translator identity and configuration, property identities, domain identity, and the faithfulness artifact. C_V binds the target artifact to the terminal verifier and theorem. P_{bridge} establishes the direction in which the target result may be reused. If any identity differs at consumption time, the source warrant is not derived even though the terminal proof remains valid.

This example illustrates why "formal verification after translation" is not a single verification event. It is a chain containing at least two logically different obligations: correctness of the target proof and adequacy of the representation for transporting the source claim.

17 Evidence Is Not Truth

The Anti-Assertion Mandate can be misunderstood as saying that replacing booleans with authenticated artifacts turns evidence into truth. It does not. SCC distinguishes at least four notions:

$$\text{authentic evidence} \neq \text{sound evidence} \neq \text{complete evidence} \neq \text{truth}.$$

Authenticity concerns identity and provenance: is this the artifact whose hash, issuer, verifier, and parentage are claimed? *Soundness* concerns whether the inference encoded by the artifact is valid under its assumptions. *Completeness* concerns whether the evidence captures every premise relevant to the requested conclusion. *Truth* concerns the world described by those premises and conclusions. These relations can support one another, but none collapses automatically into the next.

A signed sensor record can be authentic and false because the sensor is defective. A mechanically checked proof can be sound relative to axioms that poorly model the intended system. A complete derivation of a weak property can be irrelevant to the stronger property a consumer actually needs. A registry can faithfully resolve an artifact that was maliciously admitted as a root of trust.

This hierarchy clarifies the role of content addressing. A cryptographic digest can establish that two parties refer to the same byte sequence with overwhelming practical confidence under the hash assumption. It cannot establish that the bytes are a correct specification, that the program they describe is free of defects, or that an observation encoded in them was true. Hashing stabilizes reference; it does not upgrade semantics.

The same point applies to formal methods. A theorem prover can provide exceptionally strong evidence that a formal statement follows from formal premises. The remaining boundary is correspondence: whether those premises and statements represent the intended external system. SCC does not solve that correspondence problem universally. It makes the correspondence assumption nameable, bindable, and rejectable when absent.

The practical consequence is a hierarchy of failure diagnostics. `UNVERIFIED_PROOF_OBLIGATION` means that the required proof artifact was not authenticated; it does not mean the proposition

is false. `DOMAIN_MISMATCH` means the presented state lies outside the admitted domain; it does not mean the action is unsafe. `UNVERIFIED_TRANSLATION_FAITHFULNESS` means the bridge is underived; it does not indict the terminal verifier. Precise failure language prevents the diagnostic layer from laundering uncertainty into a stronger negative claim.

This is also why SCC's own evidence package cannot certify SCC universally. The package can authenticate a source tree, record a test run, and bind a paper to a particular reference release. It cannot turn finite tests into a proof of implementation correctness or turn implementation correctness into a proof that every real deployment has modeled every relevant transformation. The evidence is valuable precisely when its limits remain visible.

18 Sound Forgetting and Portable Warrant

18.1 Abstraction as Disciplined Information Loss

Abstract interpretation provides the most developed mathematical precedent for reasoning soundly after deliberate loss of information. In its classical form, a concrete semantic domain C is related to an abstract domain A so that computations in A conservatively approximate facts about computations in C [4]. Useful abstractions normally erase concrete distinctions; the achievement is preservation of enough semantic structure to justify a declared class of conclusions.

Writing $\alpha : C \to A$ for an abstraction map, the relevant question is property-relative. For a Boolean property $\Phi : C \to \{0, 1\}$, the fiber criterion asks whether

$$\alpha(c_1) = \alpha(c_2) \Longrightarrow \Phi(c_1) = \Phi(c_2)$$

on the admitted domain. When this holds, Φ factors through the abstraction. Abstract interpretation supplies far richer machinery than this elementary criterion: Galois connections, fixpoint transfer, widening and narrowing, semantic approximation, and calculational derivation of analyzers. The SCC criterion should therefore not be read as a rediscovery or replacement of abstraction soundness. It is the deliberately small condition needed to state what an authorization boundary must know about a particular information-losing map.

The breadth of the abstract-interpretation program makes this distinction important. Cousot's constructive hierarchy of transition-system semantics treats semantic descriptions themselves as systematically related abstractions [9]; program-transformation frameworks can likewise be derived by abstract interpretation [10]. Later work derives semantic dependency analyses from trace semantics [12], applies abstract interpretation to analyses themselves in A^2I [11], and relates hybrid trajectory semantics whose temporal observations need not align synchronously [13]. Recent work continues this calculational view into Event-B [14]. These results substantially precede SCC's use of transformation-relative preservation and constrain what novelty can reasonably be claimed here.

18.2 Where SCC Begins

SCC addresses a later boundary in the lifecycle of a result. Suppose an abstraction and analyzer soundly establish an abstract fact:

$$C \xrightarrow{\alpha} A, \qquad A \models \widehat{\Phi}.$$

The analysis may be impeccable. A distributed system can nevertheless destroy the warrant by exporting only a token such as `VERIFIED`, by changing the admitted domain, by delegating the

conclusion to a consumer outside the certified scope, or by joining it with an independently derived conclusion whose relational preconditions were never established. The failure occurs after, or around, the sound analysis rather than inside it.

It is useful to separate four stages:

$$\underbrace{C \xrightarrow{\alpha} A}_{\text{abstraction}} \quad \underbrace{A \models \widehat{\Phi}}_{\text{analysis}} \quad \underbrace{E_A \to E_B \to E_C}_{\text{warrant transport}} \quad \underbrace{a}_{\text{authorization}} .$$

Abstract interpretation principally supplies disciplined relationships for the first two stages. SCC principally specifies evidence obligations at the third stage and the conditions under which the fourth may consume the result. This is a division of labor, not a hierarchy of generality. A certified abstraction, simulation argument, refinement theorem, or abstract-interpretation proof can be exactly the evidence object that an SCC boundary resolves.

This separation also explains why two consumers can receive the same sound abstract result yet differ in authorization status. Let $B = (T, D, \Phi, E, C, t, \nu)$ denote the boundary context introduced earlier. It is consistent to have the same abstraction and abstract result while

$$W(B_1) = 1, \qquad W(B_2) = 0.$$

The second boundary may be outside the certified domain, temporally stale, reached through an uncertified delegation, combined through an unproved join, or asking a different property of the same abstraction. Soundness of α has not failed. The scope in which its result may be reused has changed.

18.3 Sound Forgetting, Admissible Loss, and Warrant Transport

This gives a useful three-level vocabulary. *Information loss* asks which distinctions disappeared. *Sound abstraction* asks which conclusions remain justified despite that loss. *Warrant-preserving transport* asks under what evidence, provenance, temporal, domain, and compositional conditions another component may reuse such a conclusion. The third question is contextual in a way that the abstraction relation alone need not be.

The connection is especially close to admissible degradation. Degradation is not a defect merely because representation becomes poorer. A transformation may discard enormous amounts of concrete detail while preserving every distinction relevant to the declared operational property. Conversely, a nearly lossless transformation can erase the single distinction on which authorization depends. SCC therefore treats the amount of information lost as secondary to which fiber was formed relative to which property.

This also sharpens the Anti-Assertion Mandate. An SCC need not reproduce an abstract-interpretation proof internally. It may identify and authenticate a proof artifact whose semantics establish the required relation. What it may not do is replace that artifact with the assertion that such a relation was checked. Abstraction soundness is a possible source of warrant; a Boolean report of abstraction soundness is not, by itself, a portable warrant.

18.4 Reflexive Analysis and the Trusted Boundary

A^2I is particularly relevant because it makes analyses themselves objects of abstraction and analysis [11]. SCC's reflexive requirement has a related motivation but a different target: the canonicalizer, registry, certificate issuer, translator, and runtime are themselves transformations and evidence consumers. Their adequacy cannot be made universal merely by placing them inside the trusted

computing base. Trust marks an assumption boundary; it does not convert the assumption into a theorem.

The consequence is methodological. SCC should not claim novelty from the bare fact that transformations, analyzers, or abstractions can themselves be analyzed. Its contribution is the explicit treatment of the *migration and consumption of warrant* as a versioned boundary protocol. The protocol records which property, domain, transformation, evidence, consumer, time interval, and implementation identity license reuse of an already established conclusion.

The portability experiment can itself be read through this lens. The reconstruction

$$I_{\text{Python}} \to K_{\text{portable}} \to I_{\text{BASIC/Forth}}$$

deliberately forgets implementation structure while testing whether the warrant-before-execute obligation survives. It is not a proof that the implementations are equivalent. It is an abstraction experiment over the implementation architecture, with shared conformance vectors acting as finite observations of the claimed correspondence.

19 Related Work

SCC sits near several established traditions without replacing them. Proof-carrying code associates untrusted code with proofs checked against a host safety policy [1]. Foundational and abstraction-carrying variants further investigate what evidence a consumer must check and how certificate information can be reconstructed efficiently. SCC borrows the broad architectural idea that acceptance should depend on checkable evidence, but moves the question outward: after one verifier has established a claim, what permits another component to transport that claim through a different observation, domain, consumer, or representation?

Proof-carrying and code-carrying authorization are especially close neighbors. Code-Carrying Authorization moves part of authorization decision-making into dynamically verified code supplied at run time [5], while proof-carrying authorization systems require requests to carry logical evidence. A Proof-Carrying File System combines proof verification with conditional capabilities for dynamic access policies whose consequences can depend on time and state [6]. SCC shares the insistence that authorization be justified by inspectable evidence, but concentrates on preservation across transformations and on the non-transitivity of a warrant whose observational basis changes.

Information-flow control provides formal models for restricting flows among security classes [2]. SCC asks a complementary question: if a monitor sees only a projection of a richer workflow, is that projection sufficient for the property being enforced? A graph can preserve every event label while erasing the reachability relation that makes an indirect flow visible. In that sense, SCC is concerned not only with which flows are permitted but with whether the enforcement representation retains enough structure to decide the declared policy.

Capability-system literature emphasizes protected references, authority, attenuation, and mediation rather than a simple equation of possession with unconditional access [3]. SCC's scope objects are only analogous. A capability primarily conveys authority over an object; an SCC primarily conveys evidence that a particular conclusion may be relied upon under declared conditions. The analogy becomes useful at delegation boundaries, where both traditions resist authority silently becoming broader merely because a token was copied.

Abstract interpretation is treated separately above because its relationship to SCC is foundational rather than incidental. Abstraction-carrying code makes an abstract model itself serve as a certificate whose validity can be checked by the consumer [7]. Work on certificate translation

likewise studies preservation of certificates across program transformations [8]. These traditions make clear that abstraction and transformation need not destroy assurance when appropriate simulation, reconstruction, or preservation results are available. SCC's fiber formulation is intentionally elementary: it isolates a property-relative observational condition and allows stronger mechanisms—including abstract-interpretation arguments—to supply the evidence establishing that condition in a concrete system.

Provenance, refinement, proof assistants, authorization logics, and epistemic logic provide additional neighboring vocabularies. SCC's narrower contribution is to make the migration of a *warrant* itself the object of analysis: what evidence permits a conclusion to survive a particular transformation and be consumed at a later boundary? The answer may be a formal proof, a certified abstraction, a provenance chain, a composition theorem, or a domain restriction. SCC is the boundary protocol around that evidence, not a replacement for the mechanism that produced it.

20 Methodology and Conformance

The reference distribution is a cross-representation implementation experiment, not a proof of the calculus by testing. The Python reference suite exercises root proof validation, domain escape, composition failure, environmental drift, stale evidence, fully certified chains, neuroformal counterexamples, orchestration, Research Unix utilities, and portability fixtures. At the archival snapshot used for this paper, the Python suite reports 34 passing tests. The BASIC and Forth source profiles and shared conformance vectors are included as portability artifacts; native FreeBASIC and Gforth execution is not claimed by this paper unless separately recorded in the artifact manifest.

The distinction between theory, specification, and implementation is deliberate:

$$\text{theory} \longleftrightarrow \text{versioned specification} \longleftrightarrow \text{reference implementation}.$$

The paper develops the first, the repository's `spec/` directory defines the normative software interface, and executable tests provide evidence about the third.

21 The System Must Certify Less Than It Knows

A framework concerned with scope must apply the same criticism to itself. SCC has a trusted computing base. Its canonicalizer may contain defects; a domain predicate may omit relevant states; a registry may be corrupted or misconfigured; a hash may faithfully identify an inadequately specified object; a transformation may occur that the model never declared.

Consequently SCC does not establish that a system is safe, nor that certainty laundering has been eliminated universally. Its narrower claim is that, for declared transformations, admitted domains, specified properties, authenticated evidence mechanisms, and the assumptions of the trusted computing base, the architecture can refuse classes of warrant transfer for which the required preservation relationship is not established.

This limitation is not ancillary. It is the framework applying its own rule to its own claims.

22 Conclusion

The Python runtime, asynchronous orchestrator, Unix filters, fictional XENIX personality, BASIC profiles, and Forth implementation function as experimental apparatus. Each reconstruction removes conveniences supplied by another environment. Much of the machinery changes; the preservation obligation remains recognizable.

For a property Φ, an information-losing transformation T can carry the relevant warrant only where distinctions erased by T are irrelevant to Φ:

$$\forall z_1, z_2 \in D, \qquad T(z_1) = T(z_2) \Longrightarrow \Phi(z_1) = \Phi(z_2).$$

Operationally: do not transport a warrant across a distinction-erasing transformation unless independently checkable evidence establishes that the erased distinction is irrelevant to the property being warranted.

Everything else in the reference implementation—certificates, registries, proof objects, content-addressed stores, manual pages, scope faults, BASIC records, and Forth words—is machinery for making that constraint executable.

The portable object is therefore not certainty. It is the boundary around what the evidence permits us to claim.

A Finite Fiber Counterexamples

A minimal structural witness uses source states $z_0 = (\text{public}, \text{external})$ and $z_1 = (\text{restricted}, \text{external})$, projection $T(o, d) = d$, and property $\Phi(o, d) = \neg(o = \text{restricted} \wedge d = \text{external})$. Then $T(z_0) = T(z_1)$ while $\Phi(z_0) \neq \Phi(z_1)$. Any authorization procedure receiving only the destination therefore lacks information required to distinguish the two cases. The `scc-fiber` utility operationalizes this finite-domain search by grouping cases by transformed value and reporting fibers on which the property varies.

The same construction applies to histories and translations. For histories, choose two trajectories with identical retained memory but different path-dependent obligations. For translation, choose two source specifications mapped to one target representation while differing on the source property. These are counterexamples to a particular preservation claim, not declarations that the transformation is globally defective.

B Evidence-Chain Structure

A Scope-Carrying Certificate is not treated as self-authenticating. Conceptually it binds a property identity, observational projection, admitted domain, proof reference, environmental assumptions, temporal validity, composition scope, disposition, issuer, and provenance. The runtime resolves referenced proof material through a trusted registry and checks that the resolved artifact binds the same semantic identities.

For representational verification, the chain separates translation evidence from terminal proof evidence. A translation certificate binds exact source and target hashes, translator identity and configuration, source and target property identities, admitted domain, and a faithfulness proof reference. A terminal proof artifact establishes a target proposition about the translated object. A bridge artifact supplies the property-relative implication required to return from target to source. No field such as `translation_faithful=true` substitutes for these objects.

C Portable Kernel

The portable kernel contains seven conceptual operations: canonicalize an object into a stable byte representation; hash or otherwise bind that representation; resolve referenced evidence; test current domain membership and assumptions; validate ancestry and composition; derive a disposition; and append an audit record. Execution remains outside the pure derivation step.

Profiles may delegate unavailable primitives to a trusted external service, but they must expose that dependency. A constrained implementation that cannot verify a cryptographic binding cannot replace verification with a local Boolean. It must return warrant non-derivation or explicitly invoke the external verifier under a declared interface.

D Conformance Snapshot

The archival snapshot accompanying this paper identifies `scc-reference 0.4.2 (Portable Kernel)`. On 25 September 2026 the Python test suite completed with 34 passing tests. The distribution records the captured test output and the SHA-256 digest of the reference ZIP in `artifacts/`. These results are evidence about that specific snapshot, not a proof of the general calculus.

The BASIC and Forth profiles are included with shared semantic vectors. Native execution under FreeBASIC and Gforth is not claimed in this snapshot because those runtimes were not present in the build environment. This negative statement is intentional: portability artifacts should not acquire a stronger warrant merely by inclusion in a passing Python distribution.

E Warrant Algebra Reference Laws

This appendix collects the operational laws used by the reference implementation. They are deliberately weaker than a total algebra.

Operation	Form	Evidence requirement
Restriction	$W' \preceq W$	Mechanical proof that domain, time, consumers, or delegation scope only narrows.
Widening	$W \prec W'$	Fresh authenticated evidence for every newly admitted use.
Delegation	$W \xrightarrow{c} W_c$	Consumer and depth bindings; positive disposition is non-transitive.
Composition	$W_2 \odot_P W_1$	Authenticated P binding both parents, properties, domains, and transformation order.
Join	$W_A \bowtie_J W_B$	Relational artifact J for the pair-domain and joined property.
Refresh	$W_t \rightsquigarrow W_{t'}$	Revalidation of every non-monotone premise; immutable premises may be reused.
Replay	$(E, a) \mapsto d$	Recorded epistemic state sufficient to recompute disposition d; no claim of world replay.

The absence of a derivation is represented as non-derivation rather than as a synthetic negative theorem. Implementations may map non-derivation to NULL_ACTION, DEFER, or ABORT according to the declared execution policy, but they must not silently widen the warrant to avoid the missing proof.

F Machine-Readable Neuro-Formal Counterexample

The worked example in the main text can be encoded as the following finite vector. It is intentionally simple enough to reproduce in Python, BASIC, Forth, or by hand.

```
source-domain:
  x0 = { origin: public,     destination: external }
  x1 = { origin: restricted, destination: external }

translation tau(x):
  return x.destination

source-property Phi(x):
  return not (x.origin == restricted
              and x.destination == external)

observations:
  tau(x0) = external
  tau(x1) = external
  Phi(x0) = true
  Phi(x1) = false
```

```
result:
  FIBER VIOLATION
  source warrant not reconstructible from tau(x) alone
```

A conforming finite-domain checker should report at least one witness pair containing `x0` and `x1`. If the admitted domain is restricted so that only public-origin states remain, this particular counterexample disappears. That change does not certify every property of the translation; it establishes only that this witness no longer violates the declared property on the restricted domain.

References

[1] G. C. Necula, "Proof-Carrying Code," in *Proc. 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 106–119, 1997. doi:10.1145/263699.263712.

[2] D. E. Denning, "A Lattice Model of Secure Information Flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976. doi:10.1145/360051.360056.

[3] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975. doi:10.1109/PROC.1975.9939.

[4] P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," in *Proc. 4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 238–252, 1977. doi:10.1145/512950.512973.

[5] S. Maffeis, M. Abadi, C. Fournet, and A. D. Gordon, "Code-Carrying Authorization," in *Computer Security – ESORICS 2008*, LNCS 5283, pp. 563–579, 2008.

[6] D. Garg, L. Bauer, K. D. Bowers, F. Pfenning, and M. K. Reiter, "A Proof-Carrying File System," in *2010 IEEE Symposium on Security and Privacy*, 2010. doi:10.1109/SP.2010.28.

[7] E. Albert, P. Arenas, G. Puebla, and M. Hermenegildo, "Certificate Size Reduction in Abstraction-Carrying Code," *Theory and Practice of Logic Programming*, 2011.

[8] G. Barthe and C. Kunz, "An Abstract Model of Certificate Translation," *ACM Transactions on Programming Languages and Systems*, 2011. doi:10.1145/1985342.1985344.

[9] P. Cousot, "Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation," *Theoretical Computer Science*, vol. 277, nos. 1–2, pp. 47–103, 2002.

[10] P. Cousot and R. Cousot, "Systematic Design of Program Transformation Frameworks by Abstract Interpretation," in *Proc. 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 178–190, 2002.

[11] P. Cousot, R. Giacobazzi, and F. Ranzato, "A^2I: Abstract^2 Interpretation," *Proceedings of the ACM on Programming Languages*, vol. 3, POPL, Article 42, 2019. doi:10.1145/3290355.

[12] P. Cousot, "Abstract Semantic Dependency," in *Static Analysis*, LNCS 11822, pp. 389–410, Springer, 2019. doi:10.1007/978-3-030-32304-2_6.

[13] P. Cousot, "Asynchronous Correspondences Between Hybrid Trajectory Semantics," in *Principles of Systems Design*, LNCS 13660, Springer, 2023.

[14] P. Cousot, "The Essence of Event-B from an Abstract Interpretation Perspective," in *Gedenkschrift for Jean-Raymond Abrial*, Springer, 2026.