LATTICE SUBSTRATE / POSITION PAPER

Lawful Substrates:

A Constitutional Eighth Strategy for Computational Worlds

From integrated media to invariant laws of identity, relation, authority, transition, lineage, closure, extension, and federation.

Mark Lenhardt

4 September 2026 Version 1.1.0

Abstract. The emerging Software Substrates community brings complete computational environments into view as coherent scientific objects rather than accidental stacks of applications, services, and file formats. Current work emphasizes liveness, persistence, direct manipulation, malleability, computational media, collaboration, and local agency. This paper develops the complementary constitutional layer required for generality. It proposes a law- and relation-centric eighth strategy in which plural typed subjects remain distinct while sharing invariant laws of identity, authority, admission, transition, observation, effects, causal lineage, finite execution closure, conservative extension, and federation. Documents, databases, spreadsheets, object worlds, and runtimes remain domain substrates and realizations, while the universal level is defined by the laws that govern their composition and consequences. The proposal is stated as a total evidenced transition judgment, a root-preservation law for extensions, and a set of falsifiable criteria for domain independence, independent realization, and federation without sovereignty collapse.

Keywords: software substrates; computational media; programming systems; formal methods; distributed systems; provenance; local authority; conservative extension.

Contents

A substrate should not be defined by the nouns visible in its demonstration.

A document is not a substrate law. Neither is a spreadsheet, database, canvas, notebook, object image, browser DOM, application model, or runtime. Each may be an excellent computational medium. Each may embody a coherent substrate for a domain. None is general merely because many things can be encoded inside it.

The deeper question is constitutional:

> **What remains invariant when computational worlds acquire new subjects, tools, domains, owners, implementations, and peers?**

This article proposes a law- and relation-centric answer. A **lawful substrate** is an underlying computational constitution whose laws govern identity, typed relations, authority, admission, transition, observation, isolation, effects, lineage, finite closure, extension, and federation. Domain systems elaborate their operations into that constitution. They do not rewrite it. Concrete implementations realize the constitution. They do not become its universal ontology.

Here, *lawful* means admitted under an explicit computational constitution. It does not mean merely compliant with external legislation, centrally administered, or globally owned. Governance is the mediation of commitment by stated law and authority; federation preserves multiple such authorities.

Documents, direct manipulation, liveness, persistence, and malleability are central research achievements and design goals. A constitutional account places them at the level of experience and realization, then asks the additional question: **under what laws can unlike realizations remain comparable, extensible, auditable, and federable?**

The proposed category adds a law- and relation-centered strategy to seven overlapping historical strategies. Its organizing center is unity by invariant law rather than unity by one privileged representation.

1 A converging research program

Substrate research draws together a long lineage of attempts to treat the complete computational environment as a coherent design object.

Engelbart treated computing as an environment for augmenting individual and collective intellect (Engelbart, 1962). Nelson sought structures adequate to information that is complex, changing, and indeterminate (Nelson, 1965). Kay and Goldberg described a personal dynamic medium broad enough to combine communication, memory, simulation, and authoring (A. Kay and Goldberg, 1977). Smalltalk made a live, inspectable object world into both programming environment and medium (A. C. Kay, 1993). Persistent programming later attacked the artificial boundary between short-lived computation and durable data (Atkinson, Bailey, et al., 1983; Atkinson and Morrison, 1995).

Across these lineages, the whole computational environment was already the design object. Their ideas subsequently continued through programming languages, operating systems, databases, distributed systems, security, hypermedia, provenance, and human-computer interaction, each of which developed a tractable part of the larger problem.

The contemporary Software Substrates program brings these strands into direct conversation. Jakubovic, Edwards, and Petricek (2023) argue that programming systems lack a common structure comparable to the theories used to compare programming languages. Their technical-dimensions framework seeks to make programming systems cumulative research rather than a succession of incomparable personal visions. The first

dedicated Software Substrates workshop in 2025 collected sharply different vision statements, and the 2026 workshop retained "substrate" as an umbrella over interactive authoring, computational media, malleable systems, local-first work, integration domains, and other traditions (Software Substrates, 2025; Software Substrates, 2026).

The proceedings identify the present research opportunity. The universal review of the 2025 contributions observes that working systems are valuable tests of feasibility, yet packaging ideas inside different demo systems makes abstract comparison difficult (Software Substrates, 2025). The 2026 call describes a spectrum running from accounts broad enough to find substrate-like qualities in nearly any runtime to visions that expect a successful substrate to converge on a canonical structure (Software Substrates, 2026).

The task is to develop a vocabulary strong enough to compare complete computational worlds without prematurely selecting one realization as universal.

2 Current substrate foundations

Current substrate research contributes essential design principles for complete computational environments.

Edwards (2025) defines a substrate through a complete and self-sufficient programming system, persistent code and data, direct manipulation, live programming, continuity between use and programming, and conceptual unity. The compact slogan is a "WYSIWYG document, DB, & PL in one." This is a strong realization strategy. It attacks impedance mismatches directly and places the user inside a coherent computational world rather than outside a stack of opaque services.

Klokmose (2025) develops another powerful lineage from instrumental interaction and Webstrates. Its principles reject a hard technical boundary between tools and data, demand manipulation across devices and users, use transclusion for composition, seek modification during use, permit semantically distinct substrate layers, and make edit and interaction history first-class. Webstrates operationalizes much of this by turning the browser DOM into collaboratively editable, persistent computational media (Klokmose et al., 2015). A decade of work around that platform has also documented the trade between general principle and the pragmatics of building real systems (Borowski et al., 2022).

Mackay and Beaudouin-Lafon (2025) distinguish information substrates from interaction substrates and study how interaction can gain power without forcing all complexity directly onto the user. Local-first software restores local primacy, ownership, offline operation, longevity, and collaboration (Kleppmann et al., 2019). CRDTs provide mathematically defined convergence for classes of replicated data (Shapiro et al., 2011). These are substantial parts of any serious future substrate.

The constitutional proposal complements these lines of work by supplying a level at which their agreements and disagreements can be stated without requiring any one artifact model to become universal.

The distinction is:

Question	Realization-centered answer	Constitutional answer
What does the user manipulate?	Documents, cells, objects, views, instruments	Whatever a domain law admits as subjects and relations
How does computation feel?	Live, direct, observable, malleable	A property to be realized and empirically evaluated
What persists?	An image, document graph, database, operation history	Identities and relations whose persistence conditions are explicit
What may happen?	Whatever the implementation exposes	Only transitions admitted under stated law and authority

Question	Realization-centered answer	Constitutional answer
What crosses machines or owners?	Replicated state, operations, documents, messages	Claims whose meaning, authority, lineage, and local admission remain distinct
How does the system grow?	Plugins, packages, code, schemas, documents	Conservative extension under a non-self-amending root law

A document/database/programming-language synthesis may be an exceptionally good substrate. It answers the realization question by selecting an integrated medium. A constitutional kernel answers the generality question by stating the laws under which integrated media can differ without losing meaning.

3 Seven historical strategies and a constitutional eighth

The following classification is a comparative map of what each lineage tends to make primary. Real systems frequently cross categories.

Strategy	Primary source of unity	Representative lineage	Characteristic strength	Characteristic limit for a general substrate
1. Image- and object-centered	One live object world or system image	Smalltalk and related image systems (A. C. Kay, 1993)	Immediate inspection, modification, and continuity between system and program	Tends to privilege one object ontology and one world boundary
2. Persistent-world	Orthogonal persistence or a durable address space	Persistent programming and persistent object systems (Atkinson, Bailey, et al., 1983; Atkinson and Morrison, 1995)	Removes accidental boundaries between transient computation and durable data	Persistence alone does not settle authority, effects, lineage, or federation
3. Document- and hypermedia-centered	Addressable, linked, editable media	Nelsonian hypertext, computational media, Webstrates (Nelson, 1965; Klokmose et al., 2015)	Unifies content, computation, interaction, linking, and authoring	A document model can silently become the universal ontology
4. Data-model- and database-centered	A formal data model and query/derivation operations	Relational database systems (Codd, 1970)	Stable relations, declarative manipulation, and durable shared state	Data semantics do not by themselves govern arbitrary external execution or authority
5. Protocol- and narrow-waist-centered	A small set of interface constraints	End-to-end design and REST (Saltzer, Reed, and Clark, 1984; Fielding, 2000)	Independent implementation and large-scale interoperability	The waist governs exchange, not the complete semantics of a personal computational world
6. Coordination- and component-centered	Communication or composition among autonomous parts	Linda tuple spaces and component/integration systems (Gelernter, 1985)	Decoupling, coordination, and plural execution loci	Coordination does not automatically preserve exact identity, admission, or causal evidence
7. Provenance- and content-graph-centered	Identified artifacts and derivation graphs	PROV-O, SWHIDs, and in-toto (Lebo, Sahoo, and McGuinness, 2013; Software Heritage, 2021; Torres-Arias et al., 2019)	Durable identity, derivation, attribution, and supply-chain evidence	Evidence models often describe results after the fact rather than constituting the law of every transition

Strategy	Primary source of unity	Representative lineage	Characteristic strength	Characteristic limit for a general substrate
8. Law- and relation-centered	**An invariant constitution over plural typed subjects and total evidenced transitions**	**Proposed here**	**Unity without ontological flattening; governed extension and federation**	**Must prove that the law is implementable, conservative, and useful across genuinely unrelated domains**

The eighth strategy incorporates the first seven as possible realizations, domain substrates, or imported law families. Their favored nouns become explicit domain commitments rather than implicit universal ontology.

The organizing principle is:

> **Unity without ontological flattening.**

A file need not become a process. A process need not become a theorem. A theorem need not become a document. A document need not become a database row. They may remain distinct subjects while sharing laws of identity, relation, admission, consequence, evidence, extension, and federation.

4 A substrate constitution

Let the invariant kernel be:

```
K := <Subj, Id, Eq, Rel, Auth, Admit, Step,
      Obs, Iso, Emit, Lin, Close, Ext, Fed>
```

The components are law families:

- `Subj` governs the formation and admission of subject kinds without enumerating every domain noun or requiring one universal data structure.
- `Id` and `Eq` govern identity, equality, succession, aliasing, and distinction.
- `Rel` governs how relation kinds are formed and admitted, including endpoint conditions, direction, authority, and consequence; overlays supply domain meanings.
- `Auth` identifies who or what may propose, attest, admit, delegate, or revoke.
- `Admit` governs entry into the committed world.
- `Step` governs lawful state transition and refusal.
- `Obs`, `Iso`, and `Emit` govern captured inputs, isolation assumptions, visible outputs, ambient influence, and external effects.
- `Lin` governs causal lineage rather than merely chronological logging.
- `Close` requires the dependency and execution closure of a transition to be finite and admitted before commitment.
- `Ext` governs overlays, capabilities, succession, and promotion without silent root mutation.
- `Fed` governs exchange among independently governed worlds.

A domain overlay `O`, a local policy `P`, and an admitted finite capability closure `A` specialize but do not replace `K`. “Owner” denotes the local locus of authority, which may be a person, group, institution, or formally constituted process. For current state `s`, proposal `q`, and captured boundary `c`, the core judgment is:

```
decide_(K,O,P,A) : S × Q × C
                 -> Admit(S × R × E)
                  + Reject(S × F × E)
```

R is a result, F is a first lawful failure, and E is an evidence object whose required contents and adequacy are themselves fixed by the active law. The sum is disjoint. The judgment has no silent third outcome.

Let `X_(K,O,P,A)` be the declared boundary request space. If a concrete boundary accepts bytes, decoding and malformed-input rejection belong inside this space or in an immediately preceding total judgment. Admission is an outcome, never a precondition. The minimal totality and determinism obligation is:

```
for every boundary request x, there exists exactly one y
such that decide_(K,O,P,A)(x) = y.
```

In notation:

```
∀x ∈ X_(K,O,P,A). ∃! y. decide_(K,O,P,A)(x) = y
```

A rejection preserves the committed state. It may produce evidence explaining refusal, but it does not smuggle in a partial successor:

```
decide(s,q,c) = Reject(s,f,e)
```

An admission identifies the successor, result, and evidence together:

```
decide(s,q,c) = Admit(s',r,e)
```

This makes enforcement part of the denotation of the law. “Persistent,” “malleable,” “open,” “local,” “secure,” and “observable” are not sufficient as values. A law needs a scope, a satisfaction condition, a violation condition, a deciding authority, an enforcement point, a total outcome, and evidence adequate to audit the decision.

5 Conservative narrowing under extension

Within a fixed observable alphabet Σ, additional law narrows behavior:

```
Beh_(i+1) ⊆ Beh_i
```

An extension may introduce new subject kinds or operations, so its behavior lives over an expanded alphabet. Let `Comp(K)` be the triples `(O,P,A)` lawfully composable with `K`; let ⊕ denote that governed composition, not arbitrary union; let `Beh(X)` be the observable behaviors admitted by `X`; let `Σ_K` be the root observable alphabet; and let `π_(Σ_K)` project an extended behavior onto that alphabet. The root-preservation obligation is:

```
∀(O,P,A) ∈ Comp(K).
  π_(Σ_K)(Beh(K ⊕ O ⊕ P ⊕ A)) ⊆ Beh(K)
```

For a purely conservative extension that adds vocabulary without restricting old behavior, the stronger equality holds:

```
π_(Σ_K)(Beh(K ⊕ 0)) = Beh(K)
```

This distinction matters. A spreadsheet overlay may introduce cells, ranges, formulas, recalculation, and workbook-specific failures. A theorem overlay may introduce terms, goals, elaboration, proof objects, and kernel rejection. A build overlay may introduce artifacts, dependency edges, compilers, and link steps. Each expands the vocabulary. None may redefine root identity, invent unrecorded authority, conceal effects, escape closure, or erase lineage.

Owner-reserved freedom is itself part of the constitution. A law should classify each dimension as one of:

Status	Meaning
invariant	every realization must preserve it
required	the realization must provide it
prohibited	no lawful realization may exhibit it
owner-selected	the local owner chooses within an admitted space
overlay-selected	a domain law chooses within an admitted space
variable-but-evidenced	realizations may differ, but the selected value and consequence must be recorded

Explicit reservation distinguishes owner discretion from underspecification. Without it, one implementation may treat silence as freedom while another treats it as prohibition or obligation.

6 The law families

6.1 Identity before storage

Most software lets its storage mechanism answer identity questions by accident. A pathname, memory address, database key, URL, object pointer, row number, hash, or UI position becomes “the thing” because that is what the implementation happens to expose.

A lawful substrate separates at least:

- subject identity from representation;
- equality from byte identity;
- succession from mutation;
- aliases from identities;
- local names from global references;
- content identity from contextual identity;
- imported identity from locally admitted identity.

Content-addressed identifiers such as SWHIDs solve an important subset: stable reference to source-code artifacts by intrinsic structure (Software Heritage, 2021). They do not alone state whether two executions are the same act, whether a derived result is authorized, whether a foreign identifier is locally admitted, or whether a successor preserves the identity of its predecessor. Those are additional laws.

6.2 Typed relations before a universal container

A lawful substrate is relation-centric, but it is not committed to the relational model as the single representation of everything. It requires relations to have admitted kinds, endpoint conditions, direction, authority, and consequences.

The root may know that a relation is:

```
produced-by(result, execution)
used-input(execution, subject)
authorized-by(proposal, authority)
succeeds(new, old)
interprets(view, subject)
imports(local, foreign)
```

The domain may know that another relation is:

```
cell-depends-on(cell_7, cell_2)
theorem-uses-lemma(T, L)
document-transcludes(D1, D2)
object-instantiates-schema(x, S)
```

The first group can be constitutional. The second group remains domain law. Generality comes from preserving the distinction, not from encoding every domain noun as a generic node and then pretending the semantics survived.

6.3 Authority and admission before capability

The existence of a tool or operation does not authorize its use. The ability to parse a foreign claim does not require accepting it. The possession of a capability does not permit that capability to admit itself.

Security research has long separated mechanism from policy and emphasized complete mediation (Saltzer and Schroeder, 1975). Enforceable-policy work asks which policies a monitor can actually enforce (Schneider, 2000). Proof-carrying code makes acceptance conditional on checkable evidence (Necula, 1997). SPKI's local authorization model demonstrates that useful trust need not begin with one universal naming sovereign (Ellison et al., 1999).

A substrate constitution should absorb the lesson directly:

```
Propose(x) ≠ Authorize(x) ≠ Admit(x) ≠ Execute(x)
```

and:

```
origin(proposal) cannot be the sole source of the authority
required to admit that proposal.
```

This is the non-self-admission rule. A capability may propose its successor. It cannot manufacture the root authority that makes the successor lawful.

6.4 Total transition and first refusal

Partial functions are often convenient inside an implementation. They are a poor external constitution. At the commitment boundary, every admitted request should yield exactly one admitted successor or one explicit rejection.

A first-failure order is not merely an error-message preference. It makes refusal deterministic and prevents implementations from exposing different partial interpretations of the same invalid proposal. Where several conditions fail, the law states which failure is authoritative.

This supports:

- reproducible conformance testing;
- stable audit evidence;
- cross-implementation comparison;
- denial without partial effects;
- reasoning about adversarial inputs;
- exact correspondence between abstract law and concrete behavior.

6.5 Observation, isolation, and effects

A transition cannot be explained by naming only its explicit arguments. External computation may observe environment variables, files, clocks, randomness, process state, network replies, inherited descriptors, locale, kernel behavior, hardware features, or previous mutable state. It may emit files, packets, subprocesses, UI changes, or other effects.

A lawful substrate therefore makes the observation boundary and effect boundary explicit. Isolation is not a slogan that "nothing else mattered." It is evidence of what was excluded, admitted, or captured.

At minimum, evidence for an externally realized transition should bind:

```
proposal identity
law and overlay versions
authority and admission decision
exact tool identity
finite dependency/execution closure
captured observations and inputs
isolation conditions
termination and failures
outputs and emissions
committed successor state
```

A log stating that an event happened is weaker. A provenance graph stating that one entity was derived from another is stronger. A lineage-bearing execution record binds the exact admitted act to the exact causal and authority closure that made its result possible.

PROV-O gives a cross-domain vocabulary for entities, activities, agents, generation, use, attribution, and derivation (Lebo, Sahoo, and McGuinness, 2013). in-toto binds authorized supply-chain steps, materials, products, and attestations (Torres-Arias et al., 2019). A lawful substrate treats such evidence not as optional reporting attached after execution, but as part of the transition result itself.

6.6 Finite admitted closure

A computation whose operative closure is "whatever happens to be available" is not governed. The closure need not be small, and its internals need not be implemented in one language. It must be finite for the act being admitted, identifiable, and closed under the dependencies that can influence the result.

Formally, for an admitted execution `a`:

```
Close(a) is finite
and every member of Close(a) is admitted under the active law.
```

Libraries and abstractions can serve as governed compression. At the commitment boundary, their identities, versions, authorities, and causal contributions must be admitted rather than ambient.

> **Ungoverned abstraction is the failure. Governed abstraction is compression.**

Conceptual unity is compatible with implementation heterogeneity. A realization may contain compilers, stores, processes, protocols, proof kernels, UI frameworks, and hardware layers. The constitutional requirement is **no ungoverned semantic seams**.

6.7 Lawful extension without root mutation

A substrate that cannot grow becomes a museum. A substrate whose extensions may silently redefine its law becomes an application platform with privileged plugins.

Lawful extension requires at least:

- explicit extension identity and version;
- declared new vocabulary and operations;
- proof, decision, or test evidence for root preservation;
- explicit migration or succession when old behavior is narrowed;
- no self-admission;
- revocation and rollback semantics;
- visible consequences for existing subjects and relations.

The root can permit new domains without knowing their nouns. It can permit new tools without granting those tools constitutional authority. Recursive growth is therefore possible without recursive surrender.

7 Four levels of substrate architecture

A constitutional account distinguishes four levels of commitment.

Level	Proper contents	Category error to avoid
Constitutional substrate law	Identity, relation, authority, admission, transition, observation, effects, lineage, closure, extension, federation	Treating cells, documents, rows, windows, notebooks, or a DOM as universal by default

Level	Proper contents	Category error to avoid
Concrete realization	Runtime, process model, storage, transport, cryptography, proof machinery, operating-system interface	Treating one mechanism as the only possible meaning of the law
Domain substrate or overlay	Documents, spreadsheets, proofs, builds, experiments, media, workflows, simulations	Granting a domain privileged power to rewrite the root constitution
Application and interface	Editor, notebook, spreadsheet view, IDE, inspector, collaborative workspace	Presenting a product workflow as a universal substrate law

A document can be a lawful subject. A spreadsheet can be a domain substrate. A database can realize persistence and query. A browser DOM can be a pragmatic common medium. Each belongs at an explicit architectural level.

Putting a domain noun into the root immediately imports assumptions about granularity, identity, transactions, conflict resolution, presentation, ownership, synchronization, addressability, and what counts as an operation. Those assumptions may be excellent for the intended world. They cease to be general when they become invisible.

Generality depends on preserving both plurality and level distinctions. Unlike substrate realizations become comparable only when root law, domain law, concrete realization, and interface are stated separately.

8 Elaboration: rich domains, small constitutional judgments

The useful analogy is a language elaborator and a small trusted core. Rich surface operations are translated into a smaller judgment that the kernel can check. The kernel does not need every surface construct as a primitive.

A substrate operation follows the same shape:

```
rich domain operation
        ↓ domain elaboration
identified subjects + typed relations + proposed transition
+ authority + finite closure + captured boundary
        ↓ constitutional decision
admitted successor or evidenced rejection
```

Consider four unrelated operations.

Recalculate a spreadsheet. The overlay identifies formula subjects and dependency relations, elaborates recalculation into a derivation proposal, supplies the admitted evaluator and finite input closure, and declares the expected result kind. The root decides identity, authority, closure, observation, effects, and commitment. It does not need a universal `Cell` primitive.

Check a theorem. The overlay identifies terms, declarations, dependencies, and the proof-checking operation. The constitutional layer governs which checker and environment are admitted, what exact closure was used, and whether the accepted proof object may enter committed state. It does not decide the theorem's mathematical meaning for the domain.

Produce a build artifact. The overlay supplies source and dependency relations, toolchain actions, and expected artifacts. The constitutional layer binds exact tool identities, observations, isolation, emissions, and lineage. It does not need `Compiler` to be a root subject kind.

Capture a scientific observation. The overlay owns the scientific schema, units, calibration rules, and domain validity. The constitutional layer governs authority, capture identity, observation boundaries, transformations, and evidentiary succession. It must not infer a scientific conclusion the domain has not authorized.

The root is general because it remains ignorant of domain semantics that it has no right to invent, while remaining strict about consequences that no domain may conceal.

9 Federation and semantic continuity

The field of distributed systems provides mature theories of event order, replication, convergence, consensus, failure, and serializability. Lamport's causal ordering and state-machine reasoning made the absence of a single global time an explicit scientific problem in 1978 (Lamport, 1978). CRDTs formalize convergence for replicated data under specified conditions (Shapiro et al., 2011).

For lawful substrates, those mechanisms sit inside a broader federation problem:

> **How can a complete computational world preserve semantic continuity across independently governed machines, people, organizations, and law versions?**

A federated lawful substrate carries the established distributed-systems questions into the constitutional domain:

Distributed mechanism asks	Substrate federation must also ask
Did replicas converge?	Did they converge on a state each owner was authorized to admit?
Which event happened first?	Which law and authority made each event meaningful?
Can nodes agree?	Does agreement preserve local refusal and local ownership?
Is a message authentic?	Is its asserted meaning understood, and is it locally authorized?
Was data replicated?	Were identity, lineage, restrictions, and succession preserved?
Can an operation be merged?	May its consequences cross this governance boundary?

Two laws follow.

First, intelligibility is not authority:

```
Understand_j(x) does not imply Authorize_j(x)
```

A world may decode a foreign subject, validate its evidence, and understand the law under which it was produced while still refusing to admit, execute, promote, or trust it.

Second, trust is not silently transitive:

```
Trust_i(j) and Trust_j(k) do not imply Trust_i(k)
```

Transitivity may be explicitly authorized by local policy, but it cannot be smuggled in as a property of connectivity.

A distributed substrate therefore includes more than synchronized documents or replicated state. Local-first systems restore the local copy as primary and show that collaboration can preserve data ownership (Kleppmann et al., 2019). A constitutional substrate generalizes the boundary: local authority remains local even when meanings and evidence are globally traversable.

> **Global intelligibility does not imply global authorization.**

10 Formal and empirical obligations

A law-centered account separates what formal law can establish from what human experience must demonstrate.

A proof can show that an extension is conservative, a transition deterministic, a closure finite, or an authority check complete relative to a model. Empirical study determines whether a person can discover a capability, understand a consequence, comfortably manipulate a representation, or appropriate a tool during use.

Claims such as malleability, directness, conviviality, learnability, and human-scale coherence require empirical evaluation. Accordingly, current substrate work treats interaction as first-class (Edwards, 2025; Klokmose, 2025; Mackay and Beaudouin-Lafon, 2025). The constitutional category gives those systems a common formal floor alongside human-computer interaction research.

The division of labor is:

```
formal and executable law:
    what may happen, under whose authority, with what evidence

human-centered realization:
    what can be perceived, understood, discovered, and appropriated

domain law:
    what the operation means in the subject matter
```

A serious substrate science coordinates all three while preserving their distinct evidentiary roles.

11 What would make the category scientific

A position becomes a research program only when it can fail. The law- and relation-centric category should be judged by at least the following tests.

11.1 Domain independence

The same unchanged root law must govern several genuinely unrelated domains. Demonstrating three document variants is insufficient. A stronger experiment would combine, for example, a proof development, a build, a scientific capture, and a collaborative document without adding their nouns to the kernel.

Failure condition: the root must be edited whenever a new domain is introduced.

11.2 Conservative extension

An overlay or capability must either preserve existing root-observable behavior or declare an explicit restrictive succession. Projection onto the old vocabulary must satisfy the conservation equation.

Failure condition: an extension silently changes the meaning of existing identity, authority, transition, or evidence.

11.3 Total and ordered refusal

Every admitted input must yield one admitted successor or one explicit first failure, with no partial commitment and no implementation-dependent error race.

Failure condition: malformed or conflicting proposals produce ambient exceptions, partial effects, or different authoritative failures across conforming implementations.

11.4 Closure and causal evidence

An externally executed transition must identify a finite admitted closure and bind its inputs, observations, isolation assumptions, outputs, effects, and termination to the committed result.

Failure condition: the result depends on an unrecorded ambient resource, or the evidence cannot distinguish two causally different executions.

11.5 Independent realization

At least two materially different implementations must realize the same abstract law and agree on admitted observations. Otherwise the "law" may be only a description of one program.

Failure condition: correspondence requires importing hidden behavior from the reference implementation.

11.6 Federation without sovereignty collapse

Two independently governed worlds must exchange understandable claims while retaining local admission, local refusal, and explicit trust boundaries.

Failure condition: interoperability requires one global owner, one mutable service, or automatic transitive trust.

11.7 Non-self-admission

A capability may propose an extension but cannot create the authority by which that extension is admitted.

Failure condition: executing an untrusted extension is necessary to decide whether it was safe to admit.

11.8 Empirical validation of experience

Where a realization claims malleability, direct manipulation, understandability, or lower cognitive burden, those claims must be evaluated with people and tasks appropriate to the intended users.

Failure condition: a formal property is presented as evidence of a human outcome it does not entail.

Together, these tests make the category a refutable technical claim.

12 The case in one judgment

A lawful substrate can be summarized as:

```
K; O; P; A ⊢ <s,q,c>
    ⇓ Admit<s',r,e>
    + Reject<s,f,e>
```

subject to:

```
∀(O,P,A) ∈ Comp(K).
π_(Σ_K)(Beh(K ⊕ O ⊕ P ⊕ A)) ⊆ Beh(K)
```

The first line says every proposed consequence is totally decided under explicit root law, domain law, owner policy, admitted capability closure, current state, and captured boundary. The result is either a committed successor with evidence or an evidenced refusal that preserves committed state.

The second line says domains, owners, and capabilities may add vocabulary and narrow permission, but they may not exhibit root-observable behavior forbidden by the constitution.

Everything else follows:

- documents and databases may be worlds without becoming universal law;
- implementations may be heterogeneous without hiding semantic seams;
- capability may grow without self-authorization;
- provenance may become causal lineage rather than decorative metadata;
- distribution may preserve local authority rather than dissolve it;
- owners may retain explicit freedom without relying on underspecification;
- unlike domains may remain unlike while participating in one governed computational world.

The field has already recovered the ambition to treat complete computational environments as research objects. The next step is to distinguish the medium from its constitution.

Open authoring; governed commitment.

References

Atkinson, M. P., P. J. Bailey, K. J. Chisholm, W. P. Cockshott, and R. Morrison (1983). "An Approach to Persistent Programming". In: *The Computer Journal* 26.4, pp. 360–365. DOI: 10.1093/comjnl/26.4.360.

Atkinson, M. P. and R. Morrison (1995). "Orthogonally Persistent Object Systems". In: *The VLDB Journal* 4, pp. 319–401. DOI: 10.1007/BF01231642.

Borowski, M., B. V. Fog, C. F. Griggio, J. R. Eagan, and C. N. Klokmose (2022). "Between Principle and Pragmatism: Reflections on Prototyping Computational Media with Webstrates". In: *ACM Transactions on Computer-Human Interaction* 30.4. DOI: 10.1145/3569895.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks". In: *Communications of the ACM* 13.6, pp. 377–387. DOI: 10.1145/362384.362685.

Edwards, J. (2025). *Substrate Vision Statement*. Substrates 2025 Proceedings. URL: https://software-substrates.github.io/proceedings/2025/statements/Paper%203%20-%20Jonathan%20Edwards%20-%20Substrate%20vision%20statement.pdf.

Ellison, C. M., B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen (1999). *SPKI Certificate Theory*. Experimental RFC RFC 2693. RFC Editor. DOI: 10.17487/RFC2693. URL: https://www.rfc-editor.org/rfc/rfc2693.

Engelbart, D. C. (1962). *Augmenting Human Intellect: A Conceptual Framework*. Tech. rep. AFOSR-3223. Stanford Research Institute. URL: https://www.dougengelbart.org/pubs/augment-3906.html.

Fielding, R. T. (2000). "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. University of California, Irvine. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Gelernter, D. (1985). "Generative Communication in Linda". In: *ACM Transactions on Programming Languages and Systems* 7.1, pp. 80–112. DOI: 10.1145/2363.2433.

Jakubovic, J., J. Edwards, and T. Petricek (2023). "Technical Dimensions of Programming Systems". In: *The Art, Science, and Engineering of Programming* 7.3, 13:1–13:49. DOI: 10.22152/programming-journal.org/2023/7/13. URL: https://programming-journal.org/2023/7/13/.

Kay, A. and A. Goldberg (1977). "Personal Dynamic Media". In: *Computer* 10.3, pp. 31–41. DOI: 10.1109/C-M.1977.217672.

Kay, A. C. (1993). "The Early History of Smalltalk". In: *ACM SIGPLAN Notices* 28.3, pp. 69–95. DOI: 10.1145/155360.155364.

Kleppmann, M., A. Wiggins, P. van Hardenberg, and M. McGranaghan (2019). "Local-First Software: You Own Your Data, in Spite of the Cloud". In: *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp. 154–178. DOI: 10.1145/3359591.3359737.

Klokmose, C. N., J. R. Eagan, S. Baader, W. E. Mackay, and M. Beaudouin-Lafon (2015). "Webstrates: Shareable Dynamic Media". In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, pp. 280–290. DOI: 10.1145/2807442.2807446.

Klokmose, C. N. (2025). *Substrates: From Webstrates and Beyond*. Substrates 2025 Proceedings. URL: https://software-substrates.github.io/proceedings/2025/statements/Paper%205%20-%20Clemens%20Klokmose%20-%20Substrates,%20From%20Webstrates%20and%20Beyond.pdf.

Lamport, L. (1978). "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Communications of the ACM* 21.7, pp. 558–565. DOI: 10.1145/359545.359563.

Lebo, T., S. Sahoo, and D. McGuinness (2013). *PROV-O: The PROV Ontology*. W3C Recommendation, 30 April 2013. URL: https://www.w3.org/TR/prov-o/.

Mackay, W. E. and M. Beaudouin-Lafon (2025). "Interaction Substrates: Combining Power and Simplicity in Interactive Systems". In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–16. DOI: 10.1145/3706598.3714006.

Necula, G. C. (1997). "Proof-Carrying Code". In: *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 106–119. DOI: 10.1145/263699.263712.

Nelson, T. H. (1965). “Complex Information Processing: A File Structure for the Complex, the Changing and the Indeterminate”. In: *Proceedings of the 20th ACM National Conference*, pp. 84–100. DOI: 10.1145/800197.806036.

Saltzer, J. H., D. P. Reed, and D. D. Clark (1984). “End-to-End Arguments in System Design”. In: *ACM Transactions on Computer Systems* 2.4, pp. 277–288. DOI: 10.1145/357401.357402.

Saltzer, J. H. and M. D. Schroeder (1975). “The Protection of Information in Computer Systems”. In: *Proceedings of the IEEE* 63.9, pp. 1278–1308. DOI: 10.1109/PROC.1975.9939.

Schneider, F. B. (2000). “Enforceable Security Policies”. In: *ACM Transactions on Information and System Security* 3.1, pp. 30–50. DOI: 10.1145/353323.353382.

Shapiro, M., N. Preguica, C. Baquero, and M. Zawirski (2011). “Conflict-Free Replicated Data Types”. In: *Stabilization, Safety, and Security of Distributed Systems*. Vol. 6976. Lecture Notes in Computer Science, pp. 386–400. DOI: 10.1007/978-3-642-24550-3_29. URL: https://inria.hal.science/inria-00609399.

Software Heritage (2021). *SoftWare Heritage Persistent IDentifiers (SWHIDs)*. Living specification, accessed 4 September 2026. URL: https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html.

Software Substrates (2025). *Substrates 2025 Proceedings*. Initial workshop at the International Conference on the Art, Science, and Engineering of Programming, Prague, 3 June 2025. URL: https://software-substrates.github.io/proceedings/2025/.

— (2026). *Substrates-26: Call for Contributions and Program*. International Conference on the Art, Science, and Engineering of Programming, Munich. URL: https://2026.programming-conference.org/home/substrates-2026.

Torres-Arias, S., H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos (2019). “in-toto: Providing Farm-to-Table Guarantees for Bits and Bytes”. In: *28th USENIX Security Symposium*, pp. 1393–1410. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias.