PANTA RHEI
RESEARCH PROGRAM

ANCHOR CANON TECHNICAL OVERVIEW

TauLib
Technical Overview

Lean 4 formalization layer, release manifest, and trust budget

Thorsten Fuchs • Anna-Sophie Fuchs

May 8, 2026 • canonical v1.0 • White Paper

AUDIENCE
Reviewers, formal-methods readers, Lean users, technically oriented institutional readers, and program readers who need the formalization route rather than the theory synopsis

READING TIME
Fast route 10–15 min, full route 70–100 min

STATUS
Canonical v1.0

CANONICAL ROUTE
https://panta-rhei.site/publications/anchor-documents/wp003-taulib-technical-overview/

SHORT ROUTE
https://prrp.site/wp003

LICENSE

Correspondence: thorsten@panta-rhei.site

READER STANCE

How to read this overview

CLAIM DISCIPLINE

This Technical Overview explains the current TauLib Lean 4 formalization layer and its public inspection surfaces. It does not claim empirical validation, semantic adequacy of bridge principles, peer-review completion, complete formal coverage of the research program, product readiness, deployment readiness, policy adoption, or achieved impact. Formal claims are bounded by Lean's kernel, the stated custom axioms, the pinned toolchain, and the Release Manifest; non-formal claims must be inspected through Corpus, Results, Verify, Publications, and domain review surfaces.

READER ROUTE

Read this paper as the formalization companion to WP002. Begin with the claim boundary and the release snapshot. Then follow the architecture, codebase, trust-budget, and inspection-route chapters. Use the route index to inspect the live TauLib repository, generated docs, Release Manifest, Verify/TauLib page, and Corpus correspondence.

ROUTES AND INSPECTION

Canonical route: `https://panta-rhei.site/publications/anchor-documents/wp003-taulib-technical-overview/`

Short route: `https://prrp.site/wp003`

Inspection route: `https://panta-rhei.site/verify/taulib/`

CONTENTS

§ 1 | TauLib in One Page

Fast route for first-contact readers and reviewers

TauLib is the Lean 4 formalization layer of the Panta Rhei Research Program. It is a pinned, buildable, inspectable Lean library. This page is the fast route into the paper; the rest of the document is the full route.

Current snapshot

METRIC	CURRENT VALUE	MEANING
Lean toolchain	`v4.28.0-rc1`	Pinned Lean version used by the public release.
Mathlib snapshot	`85028a6`	Pinned Mathlib commit; imported for tactics only.
TauLib snapshot	`cb5e830`	Pinned TauLib commit imported into the public release.
Lean modules/files	`512`	Total Lean modules in the pinned snapshot.
Lean lines	`142,406`	Total Lean source lines.
Theorem/lemma records	`4,863`	4,857 theorems and 6 lemmas.
Definitions	`3,741`	Represented definitions in the source summary.
Declarations/evals	`14,601`	Aggregate declaration and evaluation count.
Custom axioms	`3`	All disclosed; all in Book III spectral/number-theoretic bridges.
`sorry` assignments	`0`	Across all seven books.

What Lean owns here

Lean's kernel owns the represented definitions, theorem statements and proofs, explicit dependencies, the custom axiom inventory, and the no-`sorry` status. The Release Manifest, generated documentation, `#print axioms` artifacts, and the no-`sorry` checker make that ownership inspectable.

What Lean does not own here

Lean's kernel does not own empirical truth, semantic adequacy of bridge claims, external scientific acceptance, peer-review completion, deployment readiness, or achieved impact. This separation is intentional and is restated as the Boundary Rule below.

Trust Budget at a Glance

LAYER	OWNED BY	BOUNDARY
Custom axiom budget	3 disclosed axioms	All in Book III: `bridge_functor_exists`, `spectral_correspondence_03`, `grand_grh_adelic`. Every theorem depending on any of them is a conditional result.
No-`sorry` status	0 `sorry` across all seven books	Books I–VI sorry-free since Wave 12; Book VII's prior `theorem : True := sorry` commitments retired into inspectable `def : Commitment` values in `peer-review-fixes-v1` (2026-04-19).
Checker policy	`expected-axioms 3, expected-sorry 0`	CI fails the build if either invariant is broken; locally reproduced via `scripts/check_no_sorry.py`.
Review route	Custom axiom inventory and `#print axioms` artifacts	Public route at `panta-rhei.site/verify/custom-axioms/` and per-module `#print axioms` reports.

Boundary Rule

> *Formal checking establishes internal formal status under the stated trust budget. It does not establish empirical truth, bridge adequacy, semantic correspondence, external scientific acceptance, peer-review completion, deployment readiness, or achieved impact.*

This Boundary Rule applies throughout the paper. Where later chapters reference it, the rule is named rather than restated in full. Where the boundary is load-bearing for a specific surface (Reader Stance, What TauLib Is, Trust Budget, Reviewer Protocols, Conclusion), it is restated in context.

What a reviewer can do in 15 minutes

The paper rewards a 70–100 minute close read, but a reviewer can extract the essential trust posture in 15 minutes by following this route:

1. Open the Release Manifest at `panta-rhei.site/verify/release-manifest/`. Confirm the pinned Lean version, Mathlib snapshot, TauLib snapshot, custom axioms (3), and `sorry` status (0).
2. Open Verify/TauLib at `panta-rhei.site/verify/taulib/`. Confirm the public verification surface matches what this paper describes.
3. Open the TauLib repository at `github.com/Panta-Rhei-Research/taulib`. Confirm the pinned commit `cb5e830` is reachable.
4. Open the generated documentation at `taulib.site`. Inspect one declaration page and follow its source link into the repo.
5. Pick one theorem route. Compare its statement against the relevant Registry entry, and read its `#print axioms` profile. Confirm that any axiom dependency is one of the three disclosed custom axioms.
6. Confirm the Boundary Rule. Internal formal status under this trust budget is what TauLib delivers. Empirical truth, bridge adequacy, and external acceptance are not.

This is the fast route. The rest of the paper is the full route.

§ 2 | Release Snapshot

This paper starts with the release facts because the TauLib white paper is not a literary description of an evolving repository. It is an anchor document for a pinned public release. The counts in this section are refreshed from the current Release Manifest and from the TauLib source state used by the public site projection. They supersede earlier TauLib v2 white-paper counts and must be read as a status snapshot, not as permanent claims about future releases.

METRIC	CURRENT VALUE	MEANING
Lean toolchain	`v4.28.0-rc1`	Pinned by the TauLib toolchain for the current public release.
Mathlib snapshot	`85028a6`	Short form of Mathlib commit `85028a69f0eaaa02bc92614d68698fb495bc6770`.
TauLib snapshot	`cb5e830`	Short form of pinned TauLib release snapshot `cb5e83015b54dd72eba560953fe2461820078757`.
Lean modules/files	`512`	Release Manifest count for TauLib Lean files and modules.
Lean lines	`142,406`	Source-line count for the pinned TauLib source snapshot.
Theorem/lemma records	`4,863`	Release Manifest count of theorem and lemma records.
Declarations/evals	`14,601`	Aggregate declaration and evaluation count.
Custom axioms	`3`	Expected custom axiom budget enforced by the no-sorry check.
`sorry`	`0`	Expected and observed sorry count in the public release.

These numbers establish three kinds of accountability.

First, they provide scale. TauLib is large enough that manual prose review alone cannot be the inspection surface. A reviewer needs module routes, search, generated documentation, manifest counts, and reproducible build commands.

Second, they provide constraint. If a later public page says that the current release has zero `sorry` assignments, that statement is not a rhetorical trust gesture. It is a build-time condition in the TauLib release workflow and a manifest datum in the public projection.

Third, they provide a boundary. The counts say how much Lean material is present and how it is checked. They do not say that the encoded mathematics is the only possible formalization, that every manuscript claim has been formalized, or that a physical bridge has empirical adequacy. TauLib's strongest public claim is narrower: the represented formal obligations compile under the pinned toolchain with the stated trust budget.

> **KEY CLAIM • RELEASE DISCIPLINE**
>
> A TauLib release fact is acceptable in this paper only when it is owned by a machine-readable manifest, an auditable source file, or a public Verify surface. The paper may explain those facts, but it does not become their source of truth.

2.1 What changed since the earlier TauLib white paper

The earlier TauLib v2.0 white paper remains part of the historical public record, but WP003 is the canonical anchor document. The central changes are not cosmetic. The paper now uses the Anchor Document design system, routes readers through the public Anchor Canon, treats the Release Manifest as the count authority, and makes the distinction between formal checking and empirical or semantic acceptance explicit on every major route.

The stale v2 count pattern is deliberately retired. In particular, WP003 does not carry forward earlier file totals, theorem totals, definition totals, or provisional Book VII sorry-budget language. The current public release snapshot used here is 512 modules/files, 4,863 theorem/lemma records, 3,741 definitions, three custom axioms, and zero `sorry` assignments.

2.2 The role of WP003 in the Anchor Canon

WP003 is the third anchor document because it answers a different question from WP001 and WP002. WP001 asks whether the program is structured enough to inspect. WP002 asks what the theory itself says. WP003 asks what has been formalized in Lean, how the formal layer is organized, and how a reviewer can inspect it without accepting more than the formal layer warrants.

That separation matters. A formalization paper can easily drift into a theory synopsis or a promotional evidence claim. WP003 avoids that drift. It uses technical documentation, repository structure, build workflow, and trust-budget surfaces to explain the formalization layer in its own terms.

§ 3 | What TauLib Is

TauLib is the Lean 4 formalization layer for the public Panta Rhei release. It is a code library, a documentation surface, a trust-budget object, and a route back into the Corpus. It is not the full Corpus and not the books. It is not the Results lane. It is not the Verify lane as a whole. It is the formal substrate on which selected definitions, constructions, lemmas, theorem records, examples, and executable checks are represented.

The simplest description is also the safest one:

> TauLib is a pinned Lean 4 library whose public release can be built and inspected. Its formal claims are bounded by the Lean kernel, the imported Mathlib snapshot, the TauLib source, three explicit custom axioms, and zero `sorry` assignments.

That description has four consequences.

3.1 TauLib is a formalization layer

TauLib turns part of the research program into Lean source. In Lean, a definition, theorem, structure, instance, evaluation, or namespace has a precise technical status. The compiler does not care whether the surrounding prose is persuasive. It checks whether the term inhabits the claimed type, whether imports resolve, whether definitions elaborate, whether theorem proofs close, and whether remaining axioms are explicit.

This is why TauLib has a privileged role in the public research observatory. It gives critics a direct route into the formal layer. A reviewer can ask: which modules exist, what imports they use, what theorems they expose, what axioms a theorem depends on, whether the CI build reproduces locally, and whether a claimed correspondence to a Registry object is actually present.

3.2 TauLib is not the theory itself

TauLib is not identical with τ-Theory. The theory also has manuscript sources, Registry objects, conceptual definitions, mathematical constructions, physical bridges, predictions, falsification tests, and interpretation layers. Some of those layers are formalizable. Some are partially formalized. Some are not yet formalized. Some are deliberately outside Lean's role because they belong to empirical measurement, source policy, or domain review.

The paper therefore avoids the phrase "TauLib proves τ-Theory". The better sentence is: TauLib represents a formal layer of the current theory release and makes that layer mechanically inspectable under a declared trust budget.

3.3 TauLib is an inspection route

TauLib has public value even for readers who never run Lean. The generated documentation exposes modules, namespaces, theorem names, source links, and searchable pages. The Verify lane exposes the no-sorry status, custom-axiom inventory, trusted computing base disclosure, release manifest, and filter rules. The Corpus and Registry expose how formal objects correspond to program objects.

SURFACE	ROUTE	USE
TauLib repository	`https://github.com/Panta-Rhei-Research/taulib`	Public source repository for the formalization layer.
Generated docs	`https://taulib.site`	Rendered module and declaration documentation for public inspection.
Verify/TauLib	`https://panta-rhei.site/verify/taulib/`	Public verification status and trust-budget route.
Release Manifest	`https://panta-rhei.site/verify/release-manifest/`	Machine-readable release facts and source ownership.
Custom axioms	`https://panta-rhei.site/verify/custom-axioms/`	Public inventory of the explicit custom axiom budget.
TCB disclosure	`https://panta-rhei.site/verify/tcb/`	Trusted computing base and formal-verification boundary.

3.4 TauLib is an accountability constraint

The strongest reason to maintain TauLib is not that every reader will enjoy Lean. It is that formal source creates resistance against imprecise release language. If a theorem is said to exist, it should have a route. If a no-sorry status is public, it should be checked. If a custom axiom is necessary, it should be named. If a bridge claim is not formally owned by Lean, it should not be smuggled into the formal layer.

TauLib therefore functions as a discipline device for the whole public release. It forces a distinction among:

- mechanically checked formal material;
- explicit assumptions and custom axioms;
- manuscript mathematics that has not yet been fully formal-

ized;

- semantic correspondence claims between symbols and intended objects;
- empirical or physical bridge claims;
- external review, adoption, or impact claims.

WP003 is written to preserve that grammar.

§ 4 | Architecture

TauLib's architecture is organized around a disciplined import and ownership model. The public repository is a Lean package with a pinned toolchain, Mathlib import boundary, book-aligned module tree, generated documentation path, CI build, no-sorry check, and release metadata projection. The architecture is not merely a filesystem arrangement. It is the mechanism that keeps the formal layer inspectable.

4.1 Package boundary

The package boundary begins with the Lean toolchain and Lake build. The repository is meant to be reproducible: install the pinned toolchain, obtain the Mathlib cache for the pinned commit, and run the package build. This does not eliminate all environmental risk, but it fixes the formal package against the toolchain and dependency state stated in the release manifest.

At the package level, the important technical boundary is the Mathlib import seam. TauLib may use Lean and Mathlib, but it must not blur where external library facts enter. The CI workflow and architecture documentation preserve that seam so a reader can distinguish TauLib source from imported foundations.

4.2 Book-aligned module tree

The module tree mirrors the seven-book construction and adds meta/tour surface modules. The Release Manifest records the current distribution as 147 Book I modules, 66 Book II modules, 71 Book III modules, 90 Book IV modules, 81 Book V modules, 31 Book VI modules, 9 Book VII modules, and 17 meta, tour, or root modules. The total is 512.

METRIC	CURRENT VALUE	MEANING
Book I	147	Foundations, kernel, logic, sets, topos, addressability, boundary, coordinates, and related modules.
Book II	66	Holomorphy, enrichment, geometry, closure, mirror, topology, and central theorem surfaces.
Book III	71	Spectrum, arithmetic, bridge, computation, doors, hinge, spectral, sector, and physics-oriented modules.
Book IV	90	Microcosm surfaces including arena, calibration, particles, quantum mechanics, strong sector, and mass derivation.
Book V	81	Macrocosm surfaces including cosmology, gravity, thermodynamics, fluid macro, temporal, and astrophysics.
Book VI	31	Life, agency, closure, persistence, mind, consumer, source, and sector modules.
Book VII	9	Metaphysics, ethics, logos, social, final, and related commitment surfaces.
Meta/tour/root	17	Reader tours, root imports, release inspection, and documentation helper surfaces.

This arrangement is a readability choice, not a proof of completeness. A book may contain manuscript arguments that have no Lean counterpart. Conversely, a Lean module may exist to support a formalized construction without being a standalone public doctrine page. The correspondence is therefore routed through the Corpus and Registry rather than asserted by folder names alone.

4.3 Kernel and bridge separation

The architecture's most important doctrinal split is kernel versus bridge. Kernel-side material is material that Lean can check as formal content: definitions, structures, theorems, lemmas, and computations. Bridge-side material links formal symbols to intended mathematical, physical, or empirical meaning. Bridge claims are necessary for the research program, but they are not made true merely because a Lean theorem compiles.

This is why TauLib's architecture refuses a common shortcut. It does not say: formalization implies reality. It says: formalization

makes part of the argument explicit, typechecked, and auditable. Semantic and empirical adequacy remain separate obligations.

4.4 Generated documentation

Generated documentation is part of the architecture because the public release is intended to be inspectable without cloning the repository first. The docs site exposes module pages, declaration pages, namespace pages, and source links. It also provides a bridge between the public website and Lean source: an expert can move from a Verify or Corpus page into generated docs, and from there into the repository.

Generated docs are not a separate authority. They are a projection of source. If source, docs, manifest, and Verify pages disagree, the disagreement is a release defect. One purpose of the current anchor work is to reduce those defects by routing stable statements back to the same source-owned facts.

4.5 CI as a release surface

The GitHub Actions workflow is not just a build convenience. It is part of the published trust surface. The relevant workflow performs the Lean build, runs the no-sorry and expected-axiom check, and harvests a structured `#print axioms` report from Lean's own elaborator path. The public statement "zero `sorry`" is therefore backed by a script invocation: `python3 scripts/check_no_sorry.py --root TauLib --expected-axioms 3 --expected-sorry 0`.

That command is intentionally narrow. It does not verify empirical truth. It does not inspect every possible semantic interpretation. It verifies that the source tree has the expected no-sorry and axiom profile under the current repository rules.

§ 5 | Codebase Walkthrough

This chapter gives a reader's map of the TauLib repository. It does not replace the generated documentation or a local clone. Its purpose is to make the source tree legible before the reader enters it.

5.1 Root files

The root package files define the formal environment. The Lean toolchain pins the compiler version. The Lake file specifies the package and dependency configuration. The README gives build and inspection routes. The workflow directory defines CI. The scripts directory contains checks such as the no-sorry/expected-axiom verifier. The docs directory contains human-readable architecture and formalization-status documentation.

SURFACE	ROUTE	USE
Toolchain	`lean-toolchain`	Pins Lean `v4.28.0-rc1`.
Package configuration	`lakefile.lean`	Defines the Lean package and dependency relationship to Mathlib.
Build workflow	`.github/workflows/lean-build.yml`	Runs the public CI build and verification checks.
No-sorry checker	`scripts/check_no_sorry.py`	Checks expected custom axioms and zero `sorry`.
Architecture docs	`docs/ARCHITECTURE.md`	Explains the import model, module organization, and public inspection stance.
Status docs	`docs/FORMALIZATION_STATUS.md`	Summarizes the current formalization status and known boundaries.

5.2 Book I: foundations

Book I modules are the largest family. They carry foundational material: kernel notions, addressability, boundary, denotation, logic, sets, topos language, coordinates, polarity, and orbit-like constructions. In a technical overview, the important point is not that every Book I manuscript paragraph is formalized. The important point is that the formal layer has enough foundational surface to make later modules refer back to explicit Lean objects instead of free-floating prose labels.

Book I is also where reviewers should expect to find many of the local naming conventions. Namespaces, import paths, and foundational definitions in later books often inherit their grammar from this layer.

5.3 Book II: holomorphy and central theorem surfaces

Book II modules carry holomorphy, enrichment, closure, geometry, topology, mirror, and central theorem surfaces. In the theory synopsis, Book II belongs to the mathematical construction of enriched internal structure. In TauLib, the corresponding modules expose what has been encoded and what depends on which foundational surfaces.

For reviewers, Book II is a useful test of the architecture because it sits between abstract foundations and later bridge-oriented material. If the import graph becomes opaque here, the formal layer becomes harder to inspect later.

5.4 Book III: spectrum and computation

Book III modules include arithmetic, bridge, computation, doors, hinge, spectral, spectrum, sector, and physics-oriented surfaces. This family is where the formalization begins to carry more of the spectral and computational vocabulary that later physical routes depend on.

The formal status must remain exact. A Book III Lean theorem is a theorem in the encoded formal environment. It is not automatically a validated claim about physical spectra. When a Book III object is relevant to a prediction or falsification route, the Results and Verify surfaces own the bridge status.

5.5 Book IV and Book V: physical grammar and world readouts

Book IV and Book V contain many of the modules most likely to interest physicists: arena, calibration, particles, quantum mechanics, strong sector, mass derivation, cosmology, gravity, thermodynamics, fluid macro, temporal, and astrophysics surfaces. These modules can make internal formal structure inspectable, but their interpretation requires careful separation from empirical adequacy.

In practical review terms, this means that a reader should ask three questions in sequence:

1. Does the Lean object compile and have the claimed formal dependencies?

2. Does the Corpus or Registry identify the intended research object?
3. Does the Results or Verify lane state the status of the empirical bridge?

Only the first question is owned by TauLib directly. The second and third are owned by adjacent public surfaces.

5.6 Book VI and Book VII: life, reflection, and commitment layers

Book VI and Book VII are smaller in module count but important in boundary discipline. Life, agency, mind, persistence, logos, ethics, social, and final commitment surfaces are easy places for a formalization paper to overclaim. WP003 keeps them in the formalization lane: TauLib may encode selected structures and commitments, but it does not claim phenomenological adequacy, ethical authority, social adoption, or metaphysical completion.

The Book VII status is especially important historically because earlier release language carried provisional sorry-budget language. The current release status used by WP003 is zero `sorry`. The paper therefore records the current state while avoiding the suggestion that zero `sorry` is identical with completed philosophical assessment.

§ 6 | Trust Budget

Formal verification always has a trust budget. The honest question is not whether a system has one, but whether it names it. TauLib's trust budget is public by design. It includes the Lean kernel and elaboration environment, Mathlib at the pinned commit, the TauLib source snapshot, the explicit custom axiom budget, repository scripts and CI configuration, and the generated release projection.

6.1 Trusted computing base

The trusted computing base is the part of the system a reader accepts in order to treat checked Lean theorems as checked formal theorems. For TauLib, that base includes Lean itself, the pinned Mathlib dependency, and any custom axioms introduced by TauLib. It also includes ordinary practical assumptions about the build environment and source integrity.

LAYER	OWNED BY	BOUNDARY
Lean kernel and elaborator	Lean `v4.28.0-rc1`	A theorem is accepted because Lean elaborates it and the kernel checks the term. Bugs in Lean are outside TauLib's direct proof obligation.
Mathlib	Pinned commit `85028a6`	Imported library facts are part of the formal environment; TauLib does not reprove Mathlib.
TauLib source	Pinned snapshot `cb5e830`	The formal claims in this paper refer to this release snapshot, not arbitrary future source.
Custom axioms	TauLib	The current public release budget is three custom axioms. They must remain explicit and inspectable.
No-sorry policy	CI and checker script	The current expected and observed `sorry` count is zero.
Bridge interpretation	Corpus, Results, Verify	Interpretive and empirical adequacy are not discharged by Lean alone.

6.2 Custom axioms

The current release permits three custom axioms. That number is not hidden in the prose and not left as folklore. It is the expected axiom count in the no-sorry checker and appears in the public custom-axiom inventory. A reader who cares about proof foundations should inspect those axioms before relying on downstream formal theorems.

The right attitude is neither alarm nor complacency. Custom axioms are common in formal projects when a project deliberately introduces assumptions, interfaces, or bridge placeholders. The question is whether they are explicit, stable, justified, and separated from theorem claims that are meant to be axiom-free. TauLib's public release makes that inventory part of the Verify surface.

6.3 No-sorry status

A `sorry` in Lean is a placeholder proof. A sorry-free release matters because it says that theorem proofs are not being bypassed by placeholder terms. The current public release records zero `sorry` assignments.

This does not mean every desired theorem exists. It means that, for theorem claims currently represented in the source, the release does not rely on `sorry`. Missing formal coverage should be discussed as missing coverage, not disguised as a proof gap inside an accepted theorem.

6.4 `#print axioms` artifacts

The CI workflow harvests a structured `#print axioms` report. This is important because no-sorry status and axiom dependency are different questions. A theorem can have no `sorry` and still depend on axioms. The `#print axioms` route is therefore a finer inspection surface: it helps a reader see which assumptions enter a theorem.

6.5 What formal verification does not buy

TauLib's formal layer buys typechecking, proof checking, explicit dependency routes, and source inspectability. It does not buy empirical truth. It does not buy uniqueness of formalization. It does not buy external acceptance. It does not make a physical bridge adequate. It does not make a publication peer-reviewed. It does not turn a public-good scenario into achieved impact.

This is not a weakness of TauLib. It is the reason WP003 exists. By naming the formal boundary, the program prevents Lean from being used as a rhetorical shield for claims that Lean cannot own.

§ 7 | What Is Formalized

The question "what is formalized?" has to be answered at several levels. TauLib formalizes source files, declarations, definitions, theorem and lemma records, structures and inductive types, computations, and registry-linked objects. But those categories alone do not tell the reader what the formal layer contributes. The more useful answer distinguishes representation, derivation, computation, correspondence, and coverage.

7.1 Representation

Representation is the first layer. A concept becomes inspectable in Lean when it is represented as a definition, structure, inductive type, namespace, notation, theorem statement, or computational artifact. Representation does not by itself prove the concept true. It makes the concept precise enough for Lean to process and for reviewers to inspect.

TauLib's current release includes 3,741 definitions and 1,700 structures plus inductive types. Those counts indicate substantial representational surface. They also warn against casual reading. A project of this size must be studied through routes, not scanned linearly.

7.2 Derivation

Derivation is the theorem and lemma layer. The current public release records 4,857 theorems and 6 lemmas, for 4,863 theorem/lemma records. A theorem record means that a proposition is represented and its proof term is accepted under the current formal environment. It does not mean that every informal theorem in the books has a corresponding Lean theorem, nor that each theorem is axiom-free.

The right review question is therefore local: for a particular theorem, what is its statement, what modules does it import, what axioms does it use, and which Registry or manuscript object does it claim to correspond to?

7.3 Computation and evaluation

TauLib also contains computational surfaces. The Release Manifest records 3,923 computations and 1,900 `native_decide` mentions. These surfaces matter because the program is not only theorem-proving prose encoded into Lean. It also uses executable checks, finite examples, and computational probes where appropriate.

Computations should be read with the same discipline as theorems. They are useful when the domain is finite, symbolic, or explicitly computational. They do not replace theoretical proof or empirical measurement when those are the real obligations.

7.4 Registry correspondence

The release records 3,323 TauLib registry links and 3,091 Registry items with TauLib links. This is one of the most important inspection bridges in the program. It says that many public Registry objects are not isolated labels: they have formal routes into TauLib.

Registry correspondence should not be confused with full formal coverage. A link can mean a definition route, theorem route, module route, support lemma, or formal witness. The Corpus and Registry must say what the link means. WP003 uses the correspondence counts to establish that the bridge exists at scale, not to claim that every linked item is fully proved in Lean.

7.5 Formalization by construction region

The book-aligned tree is only one view. Another view is the Construction Spine from WP002. TauLib supports the Spine most clearly in its early mathematical and internal-structure stages, then increasingly acts as a formal companion to physical, life, reflective, and metaphysical routes whose bridge obligations remain outside Lean alone.

LAYER	OWNED BY	BOUNDARY
Kernel and core mathematics	Book I, Book II	Strong formal role: definitions, structures, logical grammar, and theorem support.
Internal enrichment and spectrum	Book II, Book III	Formal role plus bridge-sensitive interpretation for spectral and computational surfaces.
Physical carrier and grammar	Book III, Book IV, Book V	Formal route for represented structures; empirical adequacy belongs to Results and Verify.
Life and reflection	Book VI, Book VII	Selective formal representation; philosophical and domain adequacy remain separately reviewable.
Self-hosting and closure tests	Meta, Tour, Book VII	Inspection and commitment surfaces; not a declaration that final-theory status has been achieved.

7.6 Known incompleteness

Every serious formalization project has incompleteness. TauLib's honest status is that it is a large, sorry-free, custom-axiom-bounded formalization layer for selected public-release obligations. It is not a complete formalization of all books, all research notes, all Results records, all physical bridge arguments, or all interpretive claims.

The research program should therefore treat incompleteness as an audit object, not a hidden embarrassment. Missing formal coverage can be converted into a Registry task, a TauLib issue, a Results status note, a Verify disclosure, or a future release target.

§ 8 | Inspection Routes

TauLib inspection is deliberately multi-surface. No single page can carry the whole review workflow. The repository carries source. Generated docs carry rendered declarations. The Release Manifest carries release facts. Verify carries trust-budget and status disclosures. Corpus and Registry carry program-object correspondence. Results carries prediction and falsification status when formal material enters empirical routes.

8.1 Route grammar

The route grammar is simple:

1. start from a public claim or object;
2. find its owning surface;
3. follow the formal route into TauLib when the claim has a Lean component;
4. inspect source, generated docs, imports, theorem statement, and axiom dependencies;
5. return to Verify or Results for non-formal status.

This workflow prevents two common errors. The first error is to treat a public claim as formal merely because a nearby Lean module exists. The second is to ignore formal work because the ultimate theory has empirical or philosophical claims. The route grammar lets both layers be inspected without collapsing them.

8.2 Repository inspection

Repository inspection is for readers who want source-level accountability. They should inspect `lean-toolchain`, `lakefile.lean`, the module tree under `TauLib/`, the CI workflow, and the verification scripts. For a local reproduction attempt, the central commands are a Lake build and the no-sorry checker with expected axiom and sorry counts.

```
lake build

python3 scripts/check_no_sorry.py --root TauLib --expected-
axioms 3 --expected-sorry 0
```

The exact build environment can change outside the paper, so readers should follow the repository README for operational details. The paper's role is to identify the reproducibility contract, not to become the build manual.

8.3 Generated documentation inspection

Generated documentation is the fastest route for many technical readers. A reviewer can search for modules, navigate namespaces, and inspect declarations without setting up Lean locally. This is especially useful for public readers who want to verify that a formal object exists before deciding whether to clone the repository.

The generated-docs route also protects public pages from becoming code dumps. The site can cite a declaration route or module route, while the docs page shows the technical object in its own medium.

8.4 Verify inspection

Verify is the route for trust claims. It answers questions such as:

- What release facts are currently public?
- How many custom axioms are allowed?
- What does the no-sorry status mean?
- What is in the trusted computing base?
- Which counts are generated rather than hand-written?

This is why WP003 routes to Verify/TauLib rather than attempting to make the PDF the final authority. A PDF can be cited. A live Verify surface can be updated when a release changes.

8.5 Corpus and Registry inspection

Corpus and Registry inspection is needed whenever a Lean object is said to correspond to a research object. The Registry gives the atomic program-object view. TauLib gives the formal-source view. A correspondence link is strongest when both directions are clear: from Registry object to Lean route, and from Lean declaration back to the intended Registry object or construction surface.

This bidirectional discipline is one of the main reasons the program maintains TauLib and the Registry separately. If everything lived only inside Lean, non-formal program objects would be distorted. If everything lived only inside the Registry, formal obligations would be too easy to state loosely.

8.6 Results and falsification inspection

Results inspection is required when a formal object is used in a prediction, calibration, falsification, or world-readout claim. Lean can support the internal mathematics of a bridge, but the Results lane owns status labels such as prediction record, falsification test, calibration cascade, or problem mirror. WP003 therefore treats Results as adjacent, not subordinate.

> **KEY CLAIM • REVIEWER PATH**
>
> For a strong technical claim, start with the landing page, open the Verify route, inspect the Release Manifest, follow the TauLib source or generated-docs route, then return to Results only if the claim has empirical or falsification content.

§ 9 | Formalization Patterns

TauLib is easier to inspect when the reader knows the recurring patterns. A large Lean library can look like a wall of names from the outside. Inside the source, however, several stable patterns repeat: representation by structures, boundary definitions, theorem statements, computation surfaces, registry links, import seams, and documentation carriers. This chapter explains those patterns without trying to replace generated docs.

9.1 Representation pattern

The representation pattern is the first step in nearly every formalization thread. An informal object becomes a Lean object. A space becomes a structure. A relation becomes a predicate or field. A named construction becomes a definition. A family of objects becomes a namespace, typeclass, or indexed structure. A theorem from the manuscript becomes a Lean theorem statement only when the premises and conclusion can be represented precisely enough for Lean.

This sounds elementary, but it is one of the most important editorial filters in the project. A term that works in prose can hide ambiguity. Lean forces the ambiguity to appear. Which type does the object inhabit? Which parameters are explicit? Which assumptions are fields, which are typeclass arguments, and which are external axioms? What is definitionally equal, and what requires a theorem?

In TauLib, representation is not merely local code style. It is a translation discipline from the books and Registry into a formal environment. Every representation choice either reduces ambiguity or moves it to a more explicit place.

9.2 Boundary-definition pattern

Many TauLib modules use definitions to create a boundary around an intended object before proving downstream facts. This pattern matters because it keeps the library from making every theorem carry the whole conceptual payload of the theory. A boundary definition can say: within this formal file, this is the object we are studying; here are the fields or assumptions needed; here is the type of the object; here is what later theorems may use.

Boundary definitions are especially important in bridge-sensitive regions. For example, a formal object may represent a spectral, physical, or commitment-level construction. The definition can be fully formal while the claim that it captures the intended real-world or metaphysical object remains outside Lean. This lets TauLib make formal progress without pretending that semantic interpretation has vanished.

9.3 Theorem-record pattern

A theorem record is not only a theorem statement. It is a route through the import graph, a dependency profile, a proof term, and a potential Registry correspondence. When a theorem is cited in public prose, the safest public claim is not "the theory proves X" but "the TauLib release contains a Lean theorem with this statement under these dependencies".

The theorem-record pattern encourages reviewers to inspect several layers:

1. the exact statement;
2. the namespace and module;
3. the imports;
4. the theorem proof;
5. the `#print axioms` dependencies;
6. any Registry or Corpus correspondence;
7. any Results or Verify surface that interprets the theorem beyond Lean.

Only the first five layers are primarily TauLib-owned. The last two are cross-surface program obligations.

9.4 Computational-check pattern

Computational checks play a different role from theorem proofs. A `native_decide` check, finite search, executable equality, or evaluation can give strong evidence within a finite or decidable formal setting. It can also make examples and sanity checks public. But a computation must be read in its domain. A finite check over a represented object is not a universal theorem unless the theorem says so. A computed value is not an empirical measurement unless it is connected to measurement through a separate bridge.

TauLib's 3,923 computation/evaluation surfaces and 1,900 `native_decide` mentions are therefore important, but they are not a separate proof of the entire program. They show that the formal layer contains executable material. Reviewers should inspect which computations support which formal or bridge claims.

9.5 Import-seam pattern

The import seam is the place where external formal mathematics enters. Lean itself and Mathlib are not rewritten by TauLib. TauLib imports them. That means the project must preserve the distinction between a theorem proved inside TauLib and a theorem imported from Mathlib. Both are part of the formal environment once imported, but they have different maintenance and review owners.

This is why the pinned Mathlib commit matters. The public release does not claim compatibility with every Mathlib state. It claims the release under Mathlib `85028a6`. If Mathlib changes, a future TauLib release must update, rebuild, and republish status.

9.6 Documentation-carrier pattern

A generated documentation page is a carrier, not an authority. It carries module information from source into a public reading surface. It gives users links, declaration pages, namespaces, and search. But it should not be used to state independent release metrics or status claims that disagree with the source and Release Manifest.

The docs route is therefore most powerful when paired with source and Verify. Generated docs help a reader find a theorem; source helps inspect it; Verify helps inspect the status claims around it.

9.7 Registry-link pattern

Registry links are the translation interface between formal objects and program objects. The current public release records 3,323 TauLib registry links and 3,091 Registry items with TauLib links. The existence of a link means the program has made a correspondence claim. The quality of the link depends on its semantics.

A mature correspondence model distinguishes several link types:

- definition witness: the Lean object represents the Registry definition;
- theorem witness: the Lean theorem formalizes a registered theorem or claim;
- support witness: the Lean object supports a larger registered item;
- module witness: the module is the owning formal route for a cluster;
- computational witness: an executable check supports a finite claim;
- route witness: the link is navigational but not a full formal witness.

WP003 does not assert that every current link has this full semantic classification. It names the target discipline so future releases can improve the correspondence layer.

9.8 Bridge-hygiene pattern

Bridge hygiene is the practice of keeping formal statements from quietly absorbing non-formal obligations. A theorem about a represented physical grammar may be formally valid while the bridge between that grammar and physical measurement remains unproven or under review. A theorem about a commitment structure may compile while philosophical interpretation remains open. A theorem about a computational finite model may be exact while the model's relation to a natural system is only conditional.

The pattern is not defensive. It is liberating. When bridge hygiene is clear, the formalization layer can be praised for what it actually does without being burdened by claims it cannot discharge.

9.9 Maintenance pattern

TauLib maintenance is not a single act of coding. It is a cycle:

1. identify a source obligation in Corpus, Registry, or manuscript;
2. represent the object in Lean;
3. prove or compute what is appropriate;
4. inspect imports, axioms, and sorry status;
5. update generated docs;
6. update correspondence metadata;
7. update release manifest facts;
8. update public Verify and publication surfaces.

This cycle is the reason WP003 treats source, manifest, docs, and Verify as a single public accountability system.

§ 10 | Reviewer Protocols

Different readers inspect TauLib for different reasons. A Lean expert wants proof dependencies and build reproducibility. A mathematician wants statement precision and assumptions. A physicist wants to know where formal structure stops and bridge adequacy begins. A program reviewer wants to know whether the public site is overstating what the formal layer owns. This chapter gives protocols for those readers.

10.1 Protocol A: reproduce the formal build

The build-reproduction protocol is the most direct. It asks whether the repository builds under the pinned toolchain and dependency state. The reviewer should clone the public TauLib repository, confirm the toolchain, obtain or build the Mathlib dependency, run the Lake build, and then run the no-sorry checker with the expected budget.

The expected outcome for the current public release is a successful build, three custom axioms, and zero `sorry` assignments. A failure can have many causes: local environment mismatch, dependency drift, cache issue, repository checkout mismatch, or a real release defect. The protocol's value is that failures become concrete.

10.2 Protocol B: inspect a theorem

The theorem-inspection protocol begins with a theorem name or generated-docs route. The reviewer opens the declaration, reads the statement, follows the source link, inspects imports, and then checks dependencies. The critical question is not whether the theorem title sounds impressive. It is whether the formal statement says what public prose claims it says.

This protocol often reveals productive mismatches. Public prose may be too strong. A theorem may formalize a narrower object. A bridge premise may be implicit in prose but explicit in Lean. Those mismatches are not failures if they are corrected. They are precisely the kind of defects a formalization layer is supposed to expose.

10.3 Protocol C: inspect an axiom dependency

The axiom protocol asks which assumptions a theorem depends on. The reviewer uses the generated docs, source, and `#print axioms` artifacts to identify dependencies. If a theorem depends on one of the three custom axioms, the reviewer should inspect the custom-axiom inventory and decide whether the dependency is expected, acceptable, or surprising.

The most valuable outcome is not necessarily axiom elimination. Sometimes an axiom is the correct explicit boundary. The problem is hidden dependency, not declared dependency.

10.4 Protocol D: inspect a Registry correspondence

The Registry protocol starts from a public object rather than a Lean theorem. The reviewer opens the Registry object, follows the TauLib link, and asks whether the linked formal object has the advertised role. If the Registry says the Lean route is a theorem witness, the reviewer should see a theorem. If it says the route is a definition witness, the reviewer should see a definition. If the current Registry projection does not yet classify the link precisely, the reviewer should treat the link as correspondence evidence rather than full formalization evidence.

This protocol is especially useful for public readers because it begins in the site's conceptual vocabulary and moves toward code only when needed.

10.5 Protocol E: inspect a bridge claim

The bridge protocol is for claims that pass from formal structure into physics, measurement, prediction, falsification, or public impact. The reviewer should not stop at TauLib. A bridge claim requires a route through Corpus or Registry, then Results or Verify, and sometimes Publications or Impact. The central question becomes: what part is formal, what part is translation, what part is empirical, and what status label owns each part?

This is the protocol that keeps WP003 from becoming WP002 or WP005. Formal source may support a bridge, but it does not own the whole bridge.

10.6 Protocol F: inspect public wording

The public-wording protocol asks whether a page, paper, or release note uses TauLib language correctly. Good wording says that TauLib represents and checks formal obligations under a pinned trust budget. Risky wording says or implies that TauLib validates the theory, proves empirical adequacy, guarantees application impact, or replaces external review.

This protocol should be applied not only to TauLib pages but to the whole site. Formalization status can leak into stronger claims if public prose is not disciplined.

10.7 Protocol G: report a defect

A high-quality TauLib defect report should identify the surface, the source route, the expected claim, the observed issue, and the layer. The layer matters because different issues have different owners. A broken source link is a docs or site defect. A theorem statement mismatch is a formalization defect. A bridge overclaim is a Results or Verify wording defect. A stale count is a release-manifest projection defect.

LAYER	OWNED BY	BOUNDARY
Build failure	TauLib repository	Report checkout, toolchain, command, and log.
Stale count	Release Manifest or downstream projection	Report source of the stale public number and expected manifest value.
Broken docs route	Generated docs or site	Report the page, expected declaration, and source route.
Axiom surprise	Custom axiom inventory and theorem route	Report theorem, `#print axioms` output, and expected dependency profile.
Bridge overclaim	Results, Verify, or public prose	Report wording and clarify which layer cannot own the claim.

10.8 What a good review does not require

A useful TauLib review does not require accepting τ-Theory. It does not require reading all seven books. It does not require endorsing the program's metaphysics or public-good aspirations.

A reviewer can contribute at the formal layer by checking source, build, imports, statements, dependencies, axioms, docs, and wording. This is one of the program's core public benefits: criticism can be layer-specific.

§ 11 | Release Governance

TauLib is a living codebase, but WP003 is an anchor document. Release governance is the discipline that lets both be true. The repository can evolve while the public release remains citeable. A future release can supersede current counts while the current anchor PDF preserves a dated snapshot.

11.1 Source ownership

The release system separates source ownership from public projection. TauLib owns formal source. The Release Manifest owns public quantitative release facts. Verify owns public status interpretation. The site presents reader routes. Publications carries citeable artifacts. No single prose paragraph should silently become the source of truth for a release fact.

This separation is particularly important for metrics. Counts such as 512 modules, 4,863 theorem/lemma records, and zero `sorry` assignments should not be hand-maintained in multiple pages without a check. The manifest policy points in the right direction: public quantitative release facts should come from generated or verified sources wherever possible.

11.2 Canonical versus development state

The public release snapshot and the live development repository can differ. That is normal. The important rule is that public pages must say which state they refer to. WP003 refers to the May 8, 2026 anchor release and the pinned TauLib snapshot named in the Release Manifest. A later repository commit may add modules, remove modules, change proofs, or alter documentation. That does not retroactively change WP003.

When development state becomes release state, the Release Manifest and downstream public pages should be regenerated. This is the clean handoff from code evolution to public status.

11.3 Legacy artifact handling

The existing 2026-05-01 TauLib white paper is not deleted from the public history. It is treated as a legacy/superseded archival artifact. The reason is not embarrassment; it is route clarity. A first-time reader should land on WP003 for the canonical TauLib technical overview. Historical readers can still inspect earlier artifacts to see how the program evolved.

This policy protects the short route https://prrp.site/wp003. WP003 owns the canonical TauLib anchor route. The legacy white paper should not compete for that short-route meaning.

11.4 Release gates

A TauLib-facing release should pass several gates before public citation:

1. the Lean build succeeds under the pinned toolchain;
2. the no-sorry checker reports the expected custom axiom and sorry counts;
3. generated docs build and link to current source;
4. release manifest metrics are regenerated or verified;
5. Verify pages agree with the manifest;
6. public PDF and site pages carry the same checksum and page count;
7. link checks confirm that public routes resolve.

These gates do not turn formal checking into empirical validation. They make the release internally consistent and inspectable.

11.5 Public language policy

Public language about TauLib should use a small controlled vocabulary. Strong phrases such as “proved”, “verified”, and “formalized” should specify their owner and scope. Safer phrases include “Lean-checked theorem”, “formalized in the current TauLib release”, “represented formal object”, “custom-axiom-dependent theorem”, “sorry-free release”, and “pinned toolchain”.

Unsafe phrases include “TauLib validates the theory”, “Lean proves the physics”, “the formalization proves impact”, and “the library is complete”. Those phrases collapse layers and should be removed or rewritten.

11.6 Why governance belongs in a technical overview

It may seem unusual to discuss governance in a technical white paper. For TauLib it is necessary. A formal library becomes a public trust surface only when its release facts, source links, documentation, and status language are governed. Without that governance, formal source can be impressive but confusing. With it, the formal layer becomes genuinely inspectable.

§ 12 | Roadmap

TauLib's roadmap is a formalization roadmap, not a promise that the theory will be accepted or that downstream applications will follow. The roadmap should be evaluated by formal-methods criteria: coverage, import hygiene, documentation, axiom reduction, reproducibility, generated artifact quality, and correspondence with the Corpus and Registry.

12.1 Coverage deepening

The first continuing task is coverage deepening. Coverage deepening means turning manuscript obligations into formal obligations where doing so is mathematically appropriate. It does not mean forcing every sentence into Lean. Some manuscript material is conceptual, historical, interpretive, empirical, or editorial. The goal is not maximum encoding; the goal is correct ownership.

Good coverage tasks have a clear source object, a formal target, an inspection route, and a status label. They should say what becomes formal and what remains non-formal.

12.2 Axiom pressure

The second task is axiom pressure. A custom axiom budget is acceptable only when it is kept under review. Future releases should ask whether each custom axiom can be eliminated, weakened, localized, justified by a better interface, or separated more clearly from theorem records that do not need it.

Axiom pressure is not the same as pretending there are no assumptions. It is a method for making assumptions more visible and less casually inherited.

12.3 Documentation and source links

The third task is documentation quality. Generated docs are useful only when they remain connected to public routes and current source links. Stale links, ambiguous module names, and missing route metadata create friction for expert review. The v2 public site already corrected earlier stale source-link patterns, and WP003 continues that discipline by refusing old repository-owner links.

12.4 Registry correspondence

The fourth task is correspondence normalization. Registry-to-TauLib links at scale are already present, but the public program benefits when link semantics become sharper. A link should ideally distinguish definition witness, theorem witness, support lemma, module owner, computational check, or route-only reference.

12.5 CI and release manifest tightening

The fifth task is release-manifest tightening. The ideal release surface is one in which all public quantitative status claims are generated or verified from machine-readable sources. Manual prose should explain, not originate, release metrics. This is the same discipline already expressed in the Release Manifest policy.

12.6 External review readiness

External review readiness does not mean external review has occurred. It means the formal layer is organized so an outside Lean or formal-methods reviewer can reproduce the build, inspect the trust budget, identify key modules, and write useful criticism without reverse-engineering the whole research program first. WP003 is one part of that readiness layer.

§ A | Appendix: Current Formalization Tables

This appendix collects compact status tables for readers who want a single place to compare the main release metrics, codebase families, and inspection surfaces. The tables are descriptive snapshots, not independent source data.

A.1 Release metrics

METRIC	CURRENT VALUE	MEANING
Modules/files	512	Total Lean modules/files in the pinned TauLib snapshot.
Lean lines	142,406	Total Lean source lines reported by the release projection.
Definitions	3,741	Definitions represented in the source summary.
Theorems	4,857	Theorem records represented in the source summary.
Lemmas	6	Lemma records represented in the source summary.
Theorems plus lemmas	4,863	Combined theorem/lemma release count.
Structures plus inductives	1,700	Structural and inductive surfaces.
Computations	3,923	Evaluation and computation surfaces.
`native_decide` mentions	1,900	Finite or decidable computational proof/check surfaces.
Registry links	3,323	TauLib links recorded in Registry correspondence data.
Registry items with TauLib links	3,091	Registry items that have TauLib correspondence.
Custom axioms	3	Expected custom axiom budget.
`sorry`	0	Expected and observed placeholder-proof count.

A.2 Public route index

SURFACE	ROUTE	USE
WP003 landing page	https://panta-rhei.site/publications/anchor-documents/wp003-taulib-technical-overview/	Canonical landing page for this paper.
Short route	https://prrp.site/wp003	Short citation route for the canonical anchor document.
Mnemonic route	https://prrp.site/wp-taulib	Human-readable short route for the TauLib technical overview.
TauLib repo	https://github.com/Panta-Rhei-Research/taulib	Public Lean source repository.
Generated docs	https://taulib.site	Rendered TauLib documentation.
Verify/TauLib	https://panta-rhei.site/verify/taulib/	Verification status and trust budget.
Release Manifest	https://panta-rhei.site/verify/release-manifest/	Release facts and source ownership.
Filter rules	https://panta-rhei.site/verify/filter-rules/	Public metric filter rules.
TCB disclosure	https://panta-rhei.site/verify/tcb/	Trusted computing base disclosure.

A.3 Audit checklist

A technical reviewer can use the following lightweight checklist before opening issue-level criticism.

1. Confirm that the local clone uses Lean `v4.28.0-rc1`.
2. Confirm that Mathlib resolves to the pinned commit family `85028a6`.
3. Run `lake build`.
4. Run the no-sorry checker with expected custom axiom count 3 and expected `sorry` count 0.
5. Inspect the custom axiom inventory.
6. For a selected theorem, inspect the theorem statement, imports, and `#print axioms` output.
7. If the theorem is tied to a Registry object, inspect the Corpus or Registry correspondence route.
8. If the theorem enters a prediction or falsification claim, inspect the Results and Verify status routes separately.

A.4 Module family map

The following map is intentionally directory-level rather than file-level. Its purpose is to orient the reviewer before they enter generated docs or source. Counts and exact files are owned by the Release Manifest and repository; the map below names the main inspection neighborhoods.

LAYER	OWNED BY	BOUNDARY
Book I / Addressability	Foundation route	Addressability and reference grammar for formal objects.
Book I / Boundary	Foundation route	Boundary surfaces for object separation and formal contexts.
Book I / CF	Foundation route	Core category or construction-family support surfaces.
Book I / Coordinates	Foundation route	Coordinate and representational support.
Book I / Denotation	Foundation route	Denotational interfaces between names and formal objects.
Book I / Holomorphy	Foundation route	Early holomorphic or structural language carried into later books.
Book I / Kernel	Kernel route	Kernel-side primitives and formal support.
Book I / KernelFoundation	Kernel route	Foundational kernel construction surfaces.
Book I / Logic	Logic route	Logical and proof-theoretic support.
Book I / MetaLogic	Logic route	Meta-logical support where needed.
Book I / Orbit	Foundation route	Orbit and action-like formal structures.
Book I / Polarity	Foundation route	Polarity grammar and formal sign/dual structures.
Book I / Sets	Foundation route	Set-level support.
Book I / Topos	Foundation route	Topos-inspired and categorical environment surfaces.

LAYER	OWNED BY	BOUNDARY
Book II / CentralTheorem	Mathematics route	Formal route for central-theorem style surfaces.
Book II / Closure	Mathematics route	Closure and completion constructions.
Book II / Domains	Mathematics route	Domain-level formal grammar.
Book II / Enrichment	Mathematics route	Self-enrichment and enriched structure.
Book II / Geometry	Mathematics route	Geometric support objects.
Book II / Hartogs	Mathematics route	Ordinal/cardinal or Hartogs-style support.
Book II / Interior	Mathematics route	Interior and internality structures.
Book II / Mirror	Mathematics route	Mirror and duality surfaces.
Book II / Prologue	Mathematics route	Book-level entry and scaffolding.
Book II / Regularity	Mathematics route	Regularity and stability surfaces.
Book II / Topology	Mathematics route	Topological support.
Book II / Transcendentals	Mathematics route	Transcendental or analytic support routes.

LAYER	OWNED BY	BOUNDARY
Book III / Arithmetic	Spectrum route	Arithmetic support for spectrum and hinge constructions.
Book III / Bridge	Bridge route	Formal objects that require careful semantic interpretation.
Book III / Computation	Computation route	Computational and executable formal surfaces.
Book III / Doors	Spectrum route	Door or transition structures in the spectral construction.
Book III / Enrichment	Spectrum route	Enrichment surfaces inherited into the spectrum.
Book III / Hinge	Hinge route	Foundational hinge support objects.
Book III / Mirror	Spectrum route	Mirror/duality surfaces in spectrum context.
Book III / Physics	Bridge route	Physics-facing formal grammar requiring separate bridge status.
Book III / Sectors	Spectrum route	Sector decomposition and classification surfaces.
Book III / Spectral	Spectrum route	Spectral formalization route.
Book III / Spectrum	Spectrum route	Core spectrum route.

LAYER	OWNED BY	BOUNDARY
Book IV / Arena	Microcosm route	Arena and state-space style surfaces.
Book IV / Calibration	Microcosm route	Calibration-facing formal surfaces.
Book IV / Coda	Microcosm route	Closing and integration surfaces.
Book IV / Electroweak	Microcosm route	Electroweak-facing formal vocabulary.
Book IV / ManyBody	Microcosm route	Many-body and collective formal support.
Book IV / MassDerivation	Microcosm route	Mass-derivation route, with bridge status owned elsewhere.
Book IV / Particles	Microcosm route	Particle-facing formal objects.
Book IV / Physics	Microcosm route	General physics-facing formal grammar.
Book IV / QuantumMechanics	Microcosm route	Quantum-mechanical formal support.
Book IV / Sectors	Microcosm route	Sector grammar in the microcosm layer.
Book IV / Strong	Microcosm route	Strong-sector formal vocabulary.

LAYER	OWNED BY	BOUNDARY
Book V / Astrophysics	Macrocosm route	Astrophysics-facing formal support.
Book V / Coda	Macrocosm route	Closing and integration surfaces.
Book V / Cosmology	Macrocosm route	Cosmological formal grammar.
Book V / FluidMacro	Macrocosm route	Fluid and macro-scale support surfaces.
Book V / Gravity	Macrocosm route	Gravity-facing formal objects.
Book V / GravityField	Macrocosm route	Gravity-field route and support structures.
Book V / Orthodox	Macrocosm route	Orthodox reference/comparison surfaces where represented.
Book V / Prologue	Macrocosm route	Book-level entry and scaffolding.
Book V / Temporal	Macrocosm route	Temporal grammar and formal support.
Book V / Thermodynamics	Macrocosm route	Thermodynamic formal support.

LAYER	OWNED BY	BOUNDARY
Book VI / Agency	Life route	Agency-facing formal surfaces.
Book VI / Closure	Life route	Closure surfaces in the life layer.
Book VI / Consumer	Life route	Consumer or uptake-facing formal grammar.
Book VI / CosmicLife	Life route	Cosmic-life support surfaces.
Book VI / LifeCore	Life route	Core life formalization route.
Book VI / Mind	Life route	Mind-facing formal support with careful bridge boundaries.
Book VI / Persistence	Life route	Persistence and continuity support.
Book VI / Sectors	Life route	Sector grammar in life route.
Book VI / Source	Life route	Source-oriented support structures.
Book VII / Ethics	Metaphysics route	Ethics-facing commitment surfaces.
Book VII / Final	Metaphysics route	Final commitment or closure surfaces.
Book VII / Logos	Metaphysics route	Logos-facing formal route.
Book VII / Meta	Metaphysics route	Meta-level support and reflection surfaces.
Book VII / Social	Metaphysics route	Social-facing commitment surfaces.

A.5 Release-metric provenance map

The release metric layer matters because public pages should not manually invent or drift from status counts. The table below maps each WP003 headline metric to the owning source category.

METRIC	CURRENT VALUE	MEANING
Lean version	`v4.28.0-rc1`	Owned by `lean-toolchain` and projected through the Release Manifest.
Mathlib commit	`85028a6`	Owned by the package dependency state and projected through the Release Manifest.
TauLib snapshot	`cb5e830`	Owned by the Release Manifest source snapshot.
Modules/files	`512`	Owned by source summary and public filter rules.
Book module counts	`147/66/71/90/81/31/9/17`	Owned by public registry filter rules and release projection.
Lean lines	`142,406`	Owned by source summary.
Theorems	`4,857`	Owned by source summary.
Lemmas	`6`	Owned by source summary.
Definitions	`3,741`	Owned by source summary.
Structures/inductives	`1,700`	Owned by source summary.
Computations/evals	`3,923`	Owned by source summary.
Declarations/evals	`14,601`	Owned by source summary.
Custom axioms	`3`	Owned by checker expectation and custom-axiom inventory.
`sorry`	`0`	Owned by checker expectation and public Verify status.

A.6 Status vocabulary

TauLib status language should remain small and exact. The following vocabulary is recommended for public use.

LAYER	OWNED BY	BOUNDARY
Lean-checked	Formal layer	The declaration elaborates and is accepted under the pinned formal environment.
Sorry-free	Formal layer	The release contains no `sorry` assignments under the checker policy.
Custom-axiom-dependent	Formal layer	The theorem or route depends on one or more explicit custom axioms.
Imported	Formal layer	A fact enters through Lean or Mathlib rather than TauLib source.
Represented	Formal layer	A concept has a Lean definition, structure, theorem statement, or related formal object.
Linked	Corpus/Registry	A public Registry object has a TauLib correspondence route.
Bridge-sensitive	Results/Verify	The formal object has interpretation obligations outside Lean.
Empirically open	Results/Verify	Formal material exists, but empirical adequacy is not established.
Publication-ready	Publications	The artifact is downloadable and internally consistent; it does not claim peer review.
Canonical v1.0	Anchor Canon	The document is the current public anchor route for its role.

A.7 Technical glossary

Bridge A route from formal objects toward intended mathematical, physical, empirical, or interpretive meaning. Bridges are not discharged by Lean alone.

Custom axiom An explicit axiom introduced by TauLib rather than proved or imported from the standard formal environment.

Generated docs Rendered documentation pages generated from Lean source. They are public inspection carriers.

Import seam The boundary between TauLib source and imported Lean or Mathlib material.

Kernel The trusted Lean component that checks proof terms once elaboration has produced them.

Lake Lean's build and package-management system used by the TauLib repository.

Mathlib The community mathematical library for Lean, pinned to a specific commit in the public release environment.

No-sorry A release property stating that theorem proofs are not completed by placeholder `sorry` assignments.

Registry correspondence A public link from Registry objects to TauLib formal routes.

Release Manifest The machine-readable public release source for status facts and counts.

Semantic adequacy The claim that a formal representation captures the intended meaning. This is not established by type-checking alone.

TCB Trusted computing base: the components and assumptions trusted for formal checking.

Theorem record A Lean theorem plus its statement, module, imports, dependencies, proof,

and possible correspondence route.

A.8 Worked inspection narratives

The following narratives show how the review protocols in this paper should feel in practice. They are deliberately generic. They do not introduce new theorem claims; they describe repeatable review motions.

Narrative 1: from a public theorem claim to Lean source

A public page says that a construction has a TauLib theorem route. The reader does not begin by judging the philosophical importance of the construction. They first ask for the formal owner. The page should expose either a generated docs route, a module route, a Registry object with TauLib correspondence, or a Verify route. The reader opens the generated docs page and confirms that the named theorem exists.

The next step is statement inspection. The reader asks whether the theorem statement is the same as the public prose claim or a narrower formal witness. If the statement is narrower, the public prose may still be acceptable, but it must not present the narrower theorem as owning the wider claim. The reader then follows the source link, inspects imports, and checks axiom dependencies.

The useful output of this review is a short note: public claim, theorem route, statement summary, dependencies, axiom profile, and any mismatch between prose and formal statement. That note can be used to improve either TauLib, the Registry correspondence, or the public wording.

Narrative 2: from a Registry object to a formal witness

A reader starts from the Registry rather than from TauLib. The Registry object has a TauLib link. The reader asks what the link means. Is the Lean object a definition witness, theorem witness, support lemma, module owner, computational check, or navigational route? If the current public projection does not yet classify it, the reader treats the link as correspondence rather than proof.

This route is important because it preserves the Registry as the program spine. The formal layer should not be an isolated island of names. It should help a reader inspect the public research graph. A good Registry-to-TauLib route therefore closes the loop: Registry object to Lean route to generated docs to source and back to the public object.

Narrative 3: from a no-sorry claim to a release check

A public reader sees the phrase "zero `sorry`". The phrase is meaningful only if it is connected to a release check. The reader opens the Verify/TauLib page or Release Manifest, confirms that the expected value is zero, then inspects the workflow or script invocation that checks it. The relevant command in the current workflow expects three custom axioms and zero `sorry`.

The review result should not be "therefore all claims are true". The result is narrower: the represented theorem layer is not using placeholder proof terms under the current checker policy. That is a significant formal status claim and should be protected from rhetorical inflation.

Narrative 4: from a bridge-sensitive theorem to Results

A physics-facing page cites a formal construction. The reader confirms that the construction has a Lean route, but then asks what the construction is being used for. If it supports a prediction, calibration, or falsification route, the reader must leave TauLib and inspect Results and Verify. The Lean route may be necessary, but it is not sufficient.

This narrative is one of the most important in the entire program. It stops the phrase "formalized in Lean" from becoming a substitute for empirical accountability. It also stops empirical uncertainty from erasing the value of formal work. The two layers can be assessed separately.

Narrative 5: from a stale count to a manifest defect

A reader finds a public count that differs from WP003 or from the Release Manifest. The correct response is not to choose the more flattering number. The reader identifies the owning source. If the Release Manifest is current and the page is stale, the page projection should be fixed. If the manifest is stale, the release generation path should be fixed. If the PDF is a dated anchor document, it should not be silently mutated; a future release can supersede it.

This narrative is why WP003 names its date and release status. A citation artifact should be stable, while live release surfaces should be able to move.

A.9 Release invariants

The following invariants should hold for the WP003-era TauLib public release. They are written as editorial and technical release rules rather than as Lean theorems.

1. A public TauLib count should have a manifest or generated-data owner.
2. A public no-sorry claim should route to a checker policy and expected value.
3. A custom axiom should be explicit, inspectable, and counted.
4. A generated documentation page should route back to current public source, not to a stale personal fork or obsolete repository owner.
5. A theorem claim should not be stronger in public prose than in its Lean statement unless the additional strength is owned by another surface.
6. A physical or empirical bridge claim should route to Results or Verify.
7. A Registry correspondence link should not be presented as full formal coverage unless its semantics support that claim.
8. A canonical PDF should carry its release date, status, short route, and canonical route.
9. A legacy artifact should remain accessible as history but not compete with the canonical short route.
10. A future TauLib release should be allowed to supersede WP003 without rewriting what WP003 said about May 2026.

A.10 Layer-specific critique forms

One practical outcome of this paper is that criticism can be layer-specific. The following forms show the minimum information a useful critique should include.

LAYER	OWNED BY	BOUNDARY
Formal source critique	Module, declaration, statement, imports, proof issue	Use when a Lean object is malformed, misnamed, too weak, too strong, or dependency-heavy.
Axiom critique	Theorem, axiom dependency, expected profile	Use when an axiom dependency is surprising or undocumented.
Build critique	Checkout, toolchain, command, log excerpt	Use when local or CI reproduction fails.
Documentation critique	Docs route, source route, broken link, stale title	Use when generated docs or source links mislead.
Registry critique	Registry ID, TauLib link, claimed link semantics	Use when correspondence is missing, ambiguous, or overstated.
Bridge critique	Formal route, Results route, status language	Use when a formal object is being used beyond Lean's ownership.
Release critique	Manifest field, public page, expected value	Use when counts or release facts drift across public surfaces.
Editorial critique	Quoted public wording and suggested replacement	Use when prose collapses formal checking into validation, acceptance, deployment, or impact claims.

A.11 Suggested public wording replacements

This final appendix is intentionally practical. It gives safer replacements for language that often becomes too strong.

LAYER	OWNED BY	BOUNDARY
Avoid: "TauLib proves the theory"	Use: "TauLib formalizes selected obligations in Lean"	The former overclaims theory-level truth; the latter names the formal layer.
Avoid: "Lean validates the physics"	Use: "Lean checks the represented formal construction; bridge adequacy is inspected separately"	The replacement preserves the kernel/bridge boundary.
Avoid: "fully formalized"	Use: "formalized in the current TauLib release where represented"	The replacement avoids global coverage overclaim.
Avoid: "no assumptions"	Use: "three explicit custom axioms and zero `sorry` assignments"	The replacement names the actual trust budget.
Avoid: "verified impact"	Use: "formal route only; impact scenarios are conditional and owned by Impact/Results surfaces"	The replacement prevents public-good overclaim.
Avoid: "peer reviewed"	Use: "publicly inspectable; external review not claimed in this release"	The replacement separates openness from external validation.

§ B | Conclusion

TauLib gives the Panta Rhei Research Program a public formal layer that can be inspected with ordinary formal-methods discipline. That is already a strong public asset. It is also a bounded one.

The current release records Lean `v4.28.0-rc1`, Mathlib `85028a6`, TauLib `cb5e830`, 512 modules/files, 142,406 Lean lines, 4,863 theorem/lemma records, 14,601 declarations and evaluations, three custom axioms, and zero `sorry`. Those numbers make the formal layer substantial. They do not make it omnipotent. They tell readers where the Lean surface stands and how to inspect it.

The central lesson of WP003 is therefore architectural. A research program can use formalization without overclaiming formalization. It can state what Lean checks, expose what Lean assumes, route what Lean cannot own, and invite technical criticism at the correct layer. That is the standard TauLib is meant to serve.

Future TauLib work should deepen coverage, reduce or justify axioms, improve correspondence metadata, strengthen generated documentation, and keep release metrics source-owned. But the release already establishes the public grammar: formal claims belong to the formal layer; bridge claims belong to bridge surfaces; empirical accountability belongs to Results and Verify; public impact claims belong elsewhere entirely.

For the anchor canon, that means WP003 is not the proof of WP002. It is the technical overview of the formalization layer that helps make WP002 inspectable. That distinction is the paper's main contribution.

How to Cite & References

Citation and source list

HOW TO CITE

Fuchs, Thorsten, and Anna-Sophie Fuchs. *TauLib Technical Overview*. Panta Rhei Research Program, WP003, canonical v1.0, May 8, 2026. Canonical route: https://panta-rhei.site/publications/anchor-documents/wp003-taulib-technical-overview/. Short route: https://prrp.site/wp003. Mnemonic route: https://prrp.site/wp-taulib.

DOI:

References

[1] Archive of Formal Proofs Editors. The archive of formal proofs. https://www.isa-afp.org/, 2024.

[2] Ofer Arieli and Arnon Avron. Reasoning with logical bilattices. *Journal of Logic, Language and Information*, 5(1):25–63, 1996.

[3] Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich. Theorem proving in Lean 4. https://leanprover.github.io/theorem_proving_in_lean4/, 2024. Online textbook; Lean Prover community.

[4] Jeremy Avigad and Patrick Massot. Mathematics in Lean. https://leanprover-community.github.io/mathematics_in_lean/, 2024. Online tutorial; Lean Prover community.

[5] Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009.

[6] Nuel D. Belnap. A useful four-valued logic. In J. Michael Dunn and George Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 5–37. Reidel, Dordrecht, 1977.

[7] Michael V. Berry and Jonathan P. Keating. The Riemann zeros and eigenvalue asymptotics. *SIAM Review*, 41(2):236–266, 1999.

[8] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004.

[9] Bryan J. Birch and Peter Swinnerton-Dyer. Notes on elliptic curves. II. *Journal für die reine und angewandte Mathematik*, 218:79–108, 1965.

[10] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *Journal of Automated Reasoning*, 51:109–128, 2013.

[11] Jasmin Christian Blanchette, Maximilian Haslbeck, Daniel Matichuk, and Tobias Nipkow. Mining the Archive of Formal Proofs. In *Intelligent Computer Mathematics (CICM 2015)*, volume 9150 of *Lecture Notes in Artificial Intelligence*, pages 3–17. Springer, 2015.

[12] Robert S. Boyer and J Strother Moore. *A Computational Logic*. Academic Press, 1979.

[13] Joachim Breitner. Loogle: A search engine for Mathlib4. https://loogle.lean-lang.org/, 2023.

[14] Daniel Bump. *Automorphic Forms and Representations*, volume 55 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1997.

[15] Kevin Buzzard, Johan Commelin, and Patrick Massot. Formalising perfectoid spaces. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pages 299–312. ACM, 2020.

[16] Dustin Clausen and Peter Scholze. Condensed mathematics and complex geometry. *Lecture notes, Universität Bonn*, 2022.

[17] Johan Commelin and Adam Topaz. Abstraction boundaries and spec driven development in pure mathematics. *Notices of the American Mathematical Society*, 71(2):240–246, 2024.

[18] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2–3):95–120, 1988.

[19] Sander R. Dahmen, Johannes Hölzl, and Robert Y. Lewis. Formalizing the solution to the cap set problem. In *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *LIPIcs*, pages 15:1–15:19, 2019.

[20] Harold Davenport. *Multiplicative Number Theory*, volume 74 of *Graduate Texts in Mathematics*. Springer, 3rd edition, 2000. Standard graduate reference for Dirichlet density on primes in arithmetic progressions; cited for the density-$1/2$ result on the B/C partition.

[21] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021.

[22] Leonardo de Moura and Sebastian Ullrich. Lake: A build system and package manager for Lean 4. https://github.com/leanprover/lean4/tree/master/src/lake, 2022.

[23] Persi Diaconis and Frederick Mosteller. Methods for studying coincidences. *Journal of the American Statistical Association*, 84(408):853–861, 1989.

[24] Peter Gustav Lejeune Dirichlet. Beweis des satzes, dass jede unbegrenzte arithmetische progression …. *Abhandlungen der Königlichen Preußischen Akademie der Wissenschaften*, pages 45–81, 1837. The original proof of Dirichlet's theorem on primes in arithmetic progressions; underwrites the natural-density-$1/2$ claim for the B/C polarity partition of TauLib Hinge 2 via the residue classes $\pm 1 \pmod 8$ and $\pm 3 \pmod 8$.

[25] J. Michael Dunn. Intuitive semantics for first-degree entailments and 'coupled trees'. *Philosophical Studies*, 29(3):149–168, 1976.

[26] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. A metaprogramming framework for formal verification. In *Proceedings of the ACM on Programming Languages (ICFP)*, volume 1, pages 34:1–34:29, 2017.

[27] Edward Frenkel. Recent advances in the Langlands program. *Bulletin of the American Mathematical Society*, 41(2):151–184, 2004.

[28] Edward Frenkel. *Langlands Correspondence for Loop Groups*, volume 103 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2007.

[29] Thorsten Fuchs and Anna-Sophie Fuchs. Address resolution, not calculation: The genealogical DAG, cayley metric, and why arithmetic in category τ has no equations. Internal preprint, Panta Rhei Research; `papers/research-papers/address-resolution/`, 2026. Hinge 7 of the eight-paper bundle.

[30] Thorsten Fuchs and Anna-Sophie Fuchs. *Categorical Foundations*, volume I of *Panta Rhei*. Panta Rhei Research Program, second edition, 2026.

[31] Thorsten Fuchs and Anna-Sophie Fuchs. *Categorical Holomorphy*, volume II of *Panta Rhei*. Panta Rhei Research Program, second edition, 2026.

[32] Thorsten Fuchs and Anna-Sophie Fuchs. *Categorical Life*, volume VI of *Panta Rhei*. Panta Rhei Research Program, second edition, 2026.

[33] Thorsten Fuchs and Anna-Sophie Fuchs. *Categorical Macrocosm*, volume V of *Panta Rhei*. Panta Rhei Research Program, second edition, 2026.

[34] Thorsten Fuchs and Anna-Sophie Fuchs. *Categorical Metaphysics*, volume VII of *Panta Rhei*. Panta Rhei Research Program, second edition, 2026.

[35] Thorsten Fuchs and Anna-Sophie Fuchs. *Categorical Microcosm*, volume IV of *Panta Rhei*. Panta Rhei Research Program, second edition, 2026.

[36] Thorsten Fuchs and Anna-Sophie Fuchs. *Categorical Spectrum*, volume III of *Panta Rhei*. Panta Rhei Research Program, second edition, 2026.

[37] Thorsten Fuchs and Anna-Sophie Fuchs. Hyperfactorization: Unique tower-atom decomposition and the ABCD chart. Internal preprint, Panta Rhei Research; `papers/research-papers/hyperfactorization/`, 2026. Hinge 1 of the eight-paper standalone-hinge bundle; awaiting external preprint server submission.

[38] Thorsten Fuchs and Anna-Sophie Fuchs. The master constant $\iota_\tau = 2/(\pi + e)$: Structural derivation via the crossing-point defect germ. Internal preprint, Panta Rhei Research; `papers/research-papers/iota-tau/`, 2026. Hinge 3 of the eight-paper bundle. Lean-side structural derivation in flight per `ROADMAP-3-HINGES.md`; current Lean module ships fiat constants.

[39] Thorsten Fuchs and Anna-Sophie Fuchs. The panta rhei foundational bundle: Research memo and reading guide for an eight-paper standalone-hinge peer-review package. Internal preprint, Panta Rhei Research; `papers/research-papers/bundle-memo/`, 2026. Framing/integration memo for the eight-paper hinge bundle; not itself a hinge.

[40] Thorsten Fuchs and Anna-Sophie Fuchs. Panta Rhei Research Program site. https://panta-rhei.site/, 2026.

[41] Thorsten Fuchs and Anna-Sophie Fuchs. Prime polarity: The B/C dichotomy and legendre density. Internal preprint, Panta Rhei Research; `papers/research-papers/prime-polarity/`, 2026. Hinge 2 of the eight-paper bundle; classical references in this paper: Gauss Disquisitiones 1801 (second supplementary law) and Dirichlet 1837 / Davenport 2000 (density 1/2).

[42] Thorsten Fuchs and Anna-Sophie Fuchs. The split-complex boundary algebra $\mathbb{H}[j] = \mathcal{R}_\partial[j]/(j^2 - 1)$: Canonical uniqueness and four-atom dictionary. Internal preprint, Panta Rhei Research; `papers/research-papers/boundary-algebra/`, 2026. Hinge 4 of the eight-paper bundle.

[43] Thorsten Fuchs and Anna-Sophie Fuchs. τ-holomorphy on the boundary algebra: ω-germ transformers, wave-equation cauchy–riemann, earned categorical machine. Internal preprint, Panta Rhei Research; `papers/research-papers/holomorphy-first/`, 2026. Hinge 5 of the eight-paper bundle.

[44] Thorsten Fuchs and Anna-Sophie Fuchs. The τ-kernel as foundational architecture: Ontic identity, linear discipline, and the star-autonomous path. Internal preprint, Panta Rhei Research; `papers/research-papers/kernel-foundation/`, 2026. Hinge 8 of the eight-paper bundle (the architectural capstone).

[45] Thorsten Fuchs and Anna-Sophie Fuchs. The τ-topos and its four-valued internal logic: Resolving paraconsistent semantic circularity via split-complex idempotents and ω-germ stabilization. Internal preprint, Panta Rhei Research; `papers/research-papers/tau-topos/`, 2026. Hinge 6 of the eight-paper bundle.

[46] Thorsten Fuchs and Anna-Sophie Fuchs. TauLib: A Mathlib-free Lean 4 library for category τ. https://github.com/Panta-Rhei-Research/taulib, 2026. Release manifest: https://panta-rhei.site/verify/release-manifest/.

[47] Rudolf Fueter. Die Funktionentheorie der differentialgleichungen $\delta u = 0$ und $\delta\delta u = 0$ mit vier reellen variablen. *Commentarii Mathematici Helvetici*, 7:307–330, 1934.

[48] Carl Friedrich Gauss. *Disquisitiones Arithmeticae*. Gerh. Fleischer, Leipzig, 1801. Original source for the second supplementary law of quadratic reciprocity, $(2/p) = (-1)^{(p^2-1)/8}$, which yields the $p \equiv \pm 1 \pmod 8$ vs. $p \equiv \pm 3 \pmod 8$ partition cited by the Prime Polarity Theorem (TauLib Hinge 2). English translation: Yale University Press, 1965.

[49] Graziano Gentili, Caterina Stoppato, and Daniele C. Struppa. *Regular Functions of a Quaternionic Variable*. Springer Monographs in Mathematics. Springer, 2013.

[50] Georges Gonthier. Formal proof—the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[51] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In *Interactive Theorem Proving (ITP 2013)*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

[52] Irving John Good. The surprise index for the multivariate normal distribution. *Annals of Mathematical Statistics*, 27(4):1130–1135, 1956.

[53] W. Timothy Gowers, Ben Green, Freddie Manners, and Terence Tao. On a conjecture of Marton. *arXiv preprint*, 2023.

[54] Thomas Hales, Mark Adams, Gertrud Bauer, et al. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017.

[55] Jesse Michael Han and Floris van Doorn. A formal proof of the independence of the continuum hypothesis. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pages 353–366, 2020.

[56] John Harrison. HOL Light: A tutorial introduction. In *Formal Methods in Computer-Aided Design (FMCAD 1996)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer, 1996.

[57] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[58] William A. Howard. The formulae-as-types notion of construction. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[59] Kenneth Ireland and Michael Rosen. *A Classical Introduction to Modern Number Theory*, volume 84 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1990. Classical reference for quadratic reciprocity, the supplementary laws, and elementary analytic number theory; standard graduate textbook.

[60] Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1999.

[61] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.

[62] Alex Kontorovich, Terence Tao, and the PrimeNumberTheorem+ contributors. PrimeNumberTheorem+: A formal proof of the prime number theorem in Lean 4. https://github.com/AlexKontorovich/PrimeNumberTheoremAnd, 2024.

[63] Lean Prover Community. The Lean 4 trusted computing base: `Lean.ofReduceBool` and `Lean.trustCompiler`. Lean 4 source repository, `src/Init/Core.lean` and related; https://github.com/leanprover/lean4, 2024. The extension axioms that appear in `#print axioms` output for theorems closed by `native_decide`.

[64] Robert Y. Lewis and Minchao Wu. A bi-directional extensible ad-hoc interface between Lean and Mathematica. *Journal of Automated Reasoning*, 66:423–452, 2022.

[65] Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, second edition, 1998.

[66] Per Martin-Löf. An intuitionistic theory of types: Predicative part. *Studies in Logic and the Foundations of Mathematics*, 80:73–118, 1975.

[67] Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

[68] Norman D. Megill and David A. Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, second edition, 2019.

[69] Hugh L. Montgomery. The pair correlation of zeros of the zeta function. *Proceedings of Symposia in Pure Mathematics*, 24:181–193, 1973.

[70] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[71] Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Chalmers University of Technology and Göteborg University, 2007.

[72] Christine Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In *Typed Lambda Calculi and Applications (TLCA 1993)*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993.

[73] Planck Collaboration. Planck 2018 results. VI. cosmological parameters. *Astronomy & Astrophysics*, 641:A6, 2020. Reference cosmology paper for the CMB acoustic-peak measurements, including $\ell_1 = 220.0 \pm 0.5$, against which the TauLib Book V Hinge prediction of $\ell_1^{\tau} = 220.63$ is compared in §6.4. Fixes the data side of the falsification protocol.

[74] Peter Scholze. Lectures on condensed mathematics. *Lecture notes, Universität Bonn*, 2019.

[75] Peter Scholze. Liquid tensor experiment. Xena Project blog, 2020.

[76] Terence Tao, Timothy Gowers, Ben Green, Freddie Manners, Floris van Doorn, and the PFR contributors. A formalization of the Polynomial Freiman-Ruzsa conjecture in Lean 4. https://github.com/teorth/pfr, 2023.

[77] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pages 367–381. ACM, 2020.

[78] The mathlib Community. Mathlib4: The Lean 4 port of the Lean mathematical library. https://github.com/leanprover-community/mathlib4, 2023.

[79] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, Princeton, 2013.

[80] Sebastian Ullrich. *An Extensible Theorem Proving Frontend*. PhD dissertation, Karlsruhe Institute of Technology (KIT), 2023.

[81] Floris van Doorn, Lars Becker, and the Carleson contributors. A formal proof of Carleson's theorem in Lean 4. https://github.com/fpvandoorn/carleson, 2024.

[82] Floris van Doorn, Patrick Massot, and Oliver Nash. Formalising the h-principle and sphere eversion. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023)*, pages 121–134, 2023.

[83] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath: A computer-checked library of univalent mathematics. https://github.com/UniMath/UniMath, 2014.

[84] Philip Wadler. Propositions as types. *Communications of the ACM*, 58(12):75–84, 2015.

[85] Freek Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Artificial Intelligence*. Springer, 2006.

[86] Andrew Wiles. Modular elliptic curves and Fermat's last theorem. *Annals of Mathematics*, 141(3):443–551, 1995.

WHITE PAPER • WP003 • canonical v1.0

TauLib
Technical Overview

Lean 4 formalization layer, release manifest, and trust budget

The Panta Rhei Research Program is an independent open research program dedicated to building a coherent theory of reality.

CANONICAL ROUTE

https://panta-rhei.site/publications/anchor-documents/wp003-taulib-technical-overview/

SHORT ROUTE

https://prrp.site/wp003

CODE

github.com/Panta-Rhei-Research

CORRESPONDENCE

thorsten@panta-rhei.site

Canonical v1.0 • Released May 8, 2026 • CC-BY-4.0