**Absolute Relativity / Overall V2 Theory – v1.9**  
Document: (4) Bridge (V1–V2)  
Author: Kent Nimmo – absoluterelativityproject@gmail.com

Project tokens: “Absolute Relativity (AR)” – ETH contract 0xAacCd7bA616405C184335F193fEf080fC982921F, SOL mint ARafKuCqRgszXZWjYGWyBT7GnLZkyiaXQd1YjXC1x224

On-chain hash record (Ethereum): see (0) Front Matter & File Map.docx, §§0.4–0.5

Contents

[**Bridge Document: V1 Formal Core and V2 Present-Act Engine** 2](#_Toc215316880)

[**0. Orientation & How to Use This Document** 2](#_Toc215316881)

[**0.1 Purpose of the Bridge** 2](#_Toc215316882)

[**0.2 Relationship to V1 and V2** 2](#_Toc215316883)

[**0.3 How to Read and Cross-Reference This Bridge** 3](#_Toc215316884)

[**1. High-Level Architecture of the Unified Framework** 5](#_Toc215316885)

[**1.1 Two Layers: Formal Core (V1) and Engine (V2)** 5](#_Toc215316886)

[**1.2 Realization Under Constraint** 7](#_Toc215316887)

[**1.3 One Theory, Two Faces** 10](#_Toc215316888)

[**2. Ontology Crosswalk: Mapping Primitives Between V1 and V2** 12](#_Toc215316889)

[**2.1 Ticks ↔ Sites (k)** 12](#_Toc215316890)

[**2.2 PMS / Carrier ↔ (Wₖ, Qₖ)** 13](#_Toc215316891)

[**2.3 Abstract State hₖ ↔ Feature Alphabet Ξ** 15](#_Toc215316892)

[**2.4 IN / ON Split ↔ Inward / Outward Read** 17](#_Toc215316893)

[**2.5 Context Ladder n ↔ 6-Band Ladder (−2 … +3)** 19](#_Toc215316894)

[**2.6 L-Roles ↔ V1 Dynamic Roles** 22](#_Toc215316895)

[**3. Operator Algebra ↔ Engine Cycle** 25](#_Toc215316896)

[**3.1 Engine Pipeline Overview** 25](#_Toc215316897)

[**3.2 Renew F ↔ Enumeration** 27](#_Toc215316898)

[**3.3 Sink S ↔ Acceptance & Commit** 29](#_Toc215316899)

[**3.4 Trade T, Sync C, Framing CT ↔ Gates & Hinge** 32](#_Toc215316900)

[**3.5 Admissibility Predicates ↔ Gates** 35](#_Toc215316901)

[**3.6 Ledger (I, E, K) & Arrow of Time** 38](#_Toc215316902)

[**4. Geometry, Dimension, Scales and the Manifest** 40](#_Toc215316903)

[**4.1 Abstract Ladder D(n) + Pivot g(D) (V1)** 41](#_Toc215316904)

[**4.2 Concrete 6-Band Ladder and UGM (V2)** 43](#_Toc215316905)

[**4.3 Encoding D(n), g(D) in the Manifest (Curve-ban Compliant)** 45](#_Toc215316906)

[**4.4 Boundary Projector 𝔅 ↔ Hinge Equality** 48](#_Toc215316907)

[**4.5 Present Plane (𝒫, J) ↔ Phase Bins Ξ\_phase** 50](#_Toc215316908)

[**5. Emergent Physics in Both Views (SR, QM, Gravity)** 53](#_Toc215316909)

[**5.1 SR: Invariant Interval ↔ Typed Budgets** 53](#_Toc215316910)

[**5.2 QM: Structural Born Rule ↔ PF/Born Ties-Only** 56](#_Toc215316911)

[**5.3 Gravity: Pivot Profile ↔ ParentGate** 59](#_Toc215316912)

[**5.4 Fields and Gauge Structure via Features and Structural Gates** 62](#_Toc215316913)

[**6. Dynamics & Master Action as Algorithmic Preference** 65](#_Toc215316914)

[**6.1 V1 Master Action and Path-Sum Picture** 65](#_Toc215316915)

[**6.2 V2 Cycle as Realization of Action Minimization** 67](#_Toc215316916)

[**6.3 Quantization as Engine Sampling** 70](#_Toc215316917)

[**7. L-Roles, Ladder, and Phenomenology (“Dynamics of Becoming”)** 73](#_Toc215316918)

[**7.1 L-Roles in V2 & Their V1 Meaning** 73](#_Toc215316919)

[**7.2 Upward vs Downward View (Generic-down, Specific-up)** 76](#_Toc215316920)

[**7.3 Nested-Time Matter & Stability in Engine** 78](#_Toc215316921)

[**8. Constraints as Principles (From Implementation to Axioms)** 80](#_Toc215316922)

[**8.1 Finiteness & Discreteness** 80](#_Toc215316923)

[**8.2 Locality & the No-Skip Rule** 82](#_Toc215316924)

[**8.3 Curve-ban as Combinatorial Control Principle** 85](#_Toc215316925)

[**8.4 Auditability & Manifest as Part of the Theory** 87](#_Toc215316926)

[**9. Unified Hinge & Unified View (Space, Matter, Qualia)** 90](#_Toc215316927)

[**9.1 Triple Coincidence at the Hinge** 90](#_Toc215316928)

[**9.2 Space as Context Hierarchy** 92](#_Toc215316929)

[**9.3 Matter as Stable Engine Pattern** 95](#_Toc215316930)

[**9.4 Qualia as Qₖ Stream and Equivalence Classes** 97](#_Toc215316931)

[**10. Navigation Aids and Conclusion** 100](#_Toc215316932)

[**10.1 Where to Look in V1** 100](#_Toc215316933)

[**10.2 Where to Look in V2** 103](#_Toc215316934)

[**10.3 Closing Summary** 106](#_Toc215316935)

**Bridge Document: V1 Formal Core and V2 Present-Act Engine**

**0. Orientation & How to Use This Document**

**0.1 Purpose of the Bridge**

This document exists to do one very specific job: **show that there is only one theory here, expressed in two complementary languages**, and give you a clean way to walk back and forth between them.

On one side you have what we will call **V1**: the formal core of Absolute Relativity as it appears in your “volumes into one” manuscript. V1 is where the theory is developed in its most abstract form. It introduces the present-moment as the primitive, the IN/ON split, the context ladder, the operator algebra (Renew, Sink, Trade, Sync, framing), the invariant interval, the pivot function over dimension, and the derivations that show how relativity, quantum behaviour, and gravity emerge from that structure.

On the other side you have **V2**: the **Present-Act Engine** and the present-first ontology that sits around it. V2 takes the same ideas and makes them **executable**. It gives you concrete objects (sites (k), world and qualia records, a finite feature alphabet), a disciplined engine pipeline (selectors, hinge equality, feasibility gates, acceptance), typed budgets, and a set of auditing rules that make the whole thing implementable and testable as code. It also refines the ontology in cleaner language: nested presents, context roles (−2, −1, 0, +1, …), and process roles (L1, L2, L3).

The bridge is needed because neither side is sufficient on its own:

* If you read **V1 in isolation**, you see what the theory demands at a deep structural level, but you do not see a fully specified engine that could be run, audited, or reproduced in a computational setting.
* If you read **V2 in isolation**, you see a very strict engine design, but you do not see, in one place, how every design choice is anchored in the abstract proofs and constructions of the earlier work.

This bridge document is therefore a **reference map**. Its purpose is to:

* **Identify all the shared primitives** (ticks, presents, IN/ON, context ladder, hinge, invariant interval) and state explicitly how they are named and represented in V1 and in V2.
* **Show how the abstract operators and constructions in V1 are realized in the engine**: how the Renew/Sink cycle appears as candidate enumeration and commit, how the IN/ON boundary appears as hinge equality, how the ledger and the arrow of time appear as the irreversible record of committed acts, and so on.
* **Explain how geometric and scale information in V1 is encoded into V2’s configuration** (the manifest) without breaking the engine’s constraints (finiteness, discreteness, curve-ban).
* **Give paired readings for key physical results**: for each major theme (special relativity, quantum probabilities, gravity as feasibility geometry), the bridge lays out the V1 derivation side-by-side with the V2 engine interpretation, so you can see that they are two descriptions of the same mechanism.

You should not think of this document as a third version of the theory. It does not introduce new dynamics or new postulates. Instead, it **makes explicit equivalences that were previously implicit**, and it gathers those equivalences in one place so that you do not have to reconstruct them from memory every time you move between the abstract formalism and the engine.

In practice, this bridge will serve at least three roles:

* As a **Rosetta stone** when you are editing or extending V1 and V2: whenever you change a definition or notation on one side, you can check here to see what must change on the other.
* As a **technical reference** for future readers or collaborators who need to understand why the engine looks the way it does, or how a particular gate or budget rule traces back to an abstract property in the formal core.
* As the **source material** for shorter, more accessible bridge sections you will later embed in a defensive publication or a public exposition. Those shorter sections can be safely condensed from this document, knowing that the full mapping and justification are archived here.

The rest of the bridge is organized thematically rather than by referencing numbered sections. Each major heading corresponds to a conceptual cluster – ontology, operators, geometry and scale, emergent physics, implementation constraints – and under each heading you will find parallel descriptions in the “V1 language” and the “V2 language,” along with explicit statements of equivalence.

**0.2 Relationship to V1 and V2**

This bridge is meant to be read **alongside** two other documents that you already have in near-final form:

* your **V1 reference document** – the condensed “all volumes into one” formal core of Absolute Relativity, and
* your **V2 engine document** – the Present-Act Engine specification together with the V2 ontology.

It is helpful to be explicit about what each of those does, and what this bridge is doing in relation to them.

**The V1 reference document**

The V1 document is the **formal backbone** of the theory. It is where you:

* Introduce the present as the primitive and define the basic ontology in abstract terms (presents, IN/ON, context ladder, carriers, ledger, etc.).
* Develop the **operator algebra** – the Renew, Sink, Trade, Sync and framing operators – and show how they act on the abstract carriers and networks of presents.
* Define the **context ladder** as a sequence of roles or levels and introduce the dimension profile (D(n)) and the pivot function (g(D)).
* Prove the **invariant interval** and related geometric results: how the discrete tick structure gives rise to a Minkowski-like relation and the basic features of special relativity.
* Extend the same machinery to **quantum behaviour** (amplitudes, Born weights, collapse) and **gravity** (pivot profile, curvature, container hierarchy).

In other words, the V1 document tells you, in a reasonably compact way, **what the theory is saying at the mathematical and conceptual level**, with proofs and constructions laid out in a continuous narrative. It does *not* try to specify a particular implementation architecture; it stays at the level of formal structure.

**The V2 engine document**

The V2 document is the **realization** of that formal core as a concrete, auditable engine:

* It refines the ontology into a present-first, context-ladder narrative written in the language you want for the final theory: nested presents, inward depth vs outward standing, 0 inside +1, the six context bands (−2, −1, 0, +1, +2, +3), and the L1/L2/L3 process roles.
* It specifies the **engine pipeline**: sites (k), world and qualia records (W\_k) and (Q\_k), selectors that enumerate candidate next states, hinge equality on a finite feature alphabet, feasibility gates (time, granularity, structural, gravity-like, CRA), ratio-lexicographic acceptance, and PF/Born sampling on exact ties only.
* It defines the **typed budgets** ((\Delta \tau, \Delta t, \Delta x)), with a discrete Minkowski-type relation as a typing constraint, and shows how special relativistic behaviour emerges when you compose acts under these constraints.
* It includes **implementation constraints and audit rules**: locality (no-skip), finiteness (finite candidate sets, finite alphabet), the curve-ban (no continuous weights or scores in control), and explicit manifest/audit structures so that a concrete code implementation can be checked against the theory.

Where the V1 document tells you *what is required*, the V2 document tells you **how to implement those requirements** in a way that a simulation or physical architecture can actually follow.

**How the bridge stands between them**

This bridge document sits **between** these two and binds them together:

* It **does not replace** either V1 or V2. You still need the V1 document for the formal proofs and the deep structural picture, and you still need the V2 document for the precise engine specification and implementation rules.
* It **makes explicit the mapping** between them: which V1 primitive is represented by which V2 object, which abstract operator is realized by which part of the engine cycle, and how V1’s geometric and scale data (such as (D(n)), (g(D)), the UGM scale, and the temporal hinge) are encoded in V2’s manifest and gate configuration.
* It **organizes those mappings by theme** – ontology, operators, geometry and scales, emergent physics, implementation constraints – so that you can quickly move from a concept in the V1 reference to its concrete counterpart in the V2 engine, or vice versa.

Practically, when you later update the V1 and V2 documents, this bridge becomes the **source of truth about equivalence**. If you rename a concept, adjust an operator, or revise a piece of engine structure, this is the place where you will update the correspondence. That way, the formal core and the implementation remain synchronized, and anyone reading the theory as a whole can see clearly that **there is one framework, expressed in two tightly aligned ways**.

### **0.3 How to Read and Cross-Reference This Bridge**

This bridge is meant to be *used*, not just read once. It should feel like a map you can keep beside you while you move between the V1 reference and the V2 engine. A few simple conventions will make that easier.

**0.3.1 Parallel “V1 view / V2 view” structure**

Most of the core sections in this bridge will have an implicit or explicit **two-column logic**:

* a **V1 view**, which describes how a concept appears in the formal core, and
* a **V2 view**, which explains how the same concept is realized in the Present-Act Engine and ontology.

Sometimes that pairing will be explicit (for example, when a subsection is broken into “Abstract picture” and “Engine picture”). Other times it will be woven into the prose. When you see a concept introduced, expect the bridge to say, in effect, “here is how this shows up in the V1 formalism, and here is how it shows up in the V2 engine.”

You can treat those as **two lenses on the same object**. If you are more comfortable thinking in abstract operator language, start from the V1 view and then read the V2 realization. If you are thinking in terms of code or simulations, start from the V2 view and then glance back at the V1 justification.

**0.3.2 Referencing topics by title, not by section numbers**

Throughout this bridge, whenever there is a need to refer to another place in the text, the reference will be by **topic or heading title**, not by section number. For example:

* “the discussion of the ontology crosswalk” rather than “see section 2.1”,
* “the mapping between the operator algebra and the engine cycle” rather than “see section 3.2”.

This is intentional. Your V1 and V2 documents may evolve and be reorganized, and even this bridge may be rearranged. Referring by title makes those references more robust: the topic remains the same even if the numbering changes.

When you later integrate parts of this bridge into other documents, you can adopt the same practice there, or, if numbering has stabilized, you can add explicit numbered cross-references at that time.

**0.3.3 How to use this bridge while reading V1**

If you are walking through the **V1 reference document** and encounter a concept you want to see in its engine form, you can:

1. Identify the **topic** in V1 (for example, “invariant interval”, “Present Plane”, “pivot function and D(n)”, “operator Sink”).
2. Look for the corresponding heading in this bridge:
   * “geometry, dimension, scales and the manifest” for the invariant interval and pivot mapping,
   * “operator algebra and engine cycle” for the mapping of Renew/Sink/Trade/Sync and framing,
   * “ontology crosswalk” for basic objects like PMS, IN/ON, carriers versus W/Q records.
3. Read the V2 view there to see **how that abstract concept is encoded in the engine**.

You do not have to read the bridge straight through. It is completely acceptable to dip into it as a **lookup table** while you are working through V1.

**0.3.4 How to use this bridge while reading V2**

If you are focused on the **V2 engine document** and want to understand why a particular design choice is the way it is, you can:

1. Identify the **engine component** you are looking at (for example, “hinge equality on a finite alphabet”, “time gate Θ”, “ParentGate schedule”, “PF/Born ties-only”).
2. Go to the section of this bridge that corresponds to that type of structure:
   * “ontology crosswalk” for questions like “what is W/Q in abstract terms?”,
   * “operator algebra and engine cycle” for questions like “which abstract operator does this gate correspond to?”,
   * “geometry, dimension, scales and the manifest” for questions like “how does D(n) or g(D) show up here?”.
3. Read the V1 view there to see **what abstract property or theorem the engine rule is realizing**.

In other words, you can treat this bridge as the **rationale layer** for the engine: it tells you which abstract constraints each engine component is satisfying.

**0.3.5 Skimming versus deep reading**

There are two natural ways to move through this bridge:

* **Skimming for orientation**: read the main headings and the opening paragraphs of each major section (ontology, operators, geometry, emergent physics, constraints). This gives you a mental picture of how everything lines up, but does not require you to absorb every mapping detail.
* **Deep reading for alignment work**: when you are editing V1, revising V2, or preparing a defensive publication, read the relevant sections in depth and verify that the wording and notation in your other documents match the mappings stated here.

You do not need to resolve all details at once. The key is that this bridge becomes your **single reference point** for how concepts are supposed to correspond. If you keep that contract intact, V1 and V2 can evolve and be reorganized without drifting apart.

**1. High-Level Architecture of the Unified Framework**

**1.1 Two Layers: Formal Core (V1) and Engine (V2)**

At the highest level, the unified framework has **two tightly coupled layers**:

* a **formal core**, which is your V1 reference document, and
* an **engine layer**, which is your V2 Present-Act Engine and associated ontology.

They are not two different theories. The engine layer is the **realization of the formal core under explicit implementation constraints**. This subsection spells out what each layer is responsible for and how they fit together conceptually.

**The V1 formal core**

The V1 formal core plays the role of a **specification**:

* It starts from the ontological commitment that **presents** (experiences-of-time) are the primitives, and that everything else – objects, space, fields, “matter” – are how relations between presents appear from a chosen vantage.
* It introduces the IN/ON distinction and the context ladder, and defines abstract carriers consisting of:
  + a tick index (k),
  + an internal state (what the present already retains), and
  + outer associations (how that present stands among others).
* It builds a **tick-operator algebra**: operations like Renew, Sink, Trade, Sync, and framing that act on the carriers and their networks. These operators encode renewal of possibilities, commitment of outcomes, reallocation of structure between IN and ON, synchronization across many carriers, and the enforcement of coherent frames.
* It defines and analyses the **context ladder**: integer-labeled roles (… −2, −1, 0, +1, +2, +3 …) with an associated dimension profile (D(n)), together with a pivot function (g(D)) that controls how couplings and curvature behave across scales.
* It proves key **geometric and dynamical properties**:
  + that a discrete invariant interval emerges from the tick structure,
  + that certain configurations of operators give rise to special relativistic behaviour,
  + that interference and Born-like probabilities arise from the structure of the Present Plane and collapse,
  + that gravity can be read as the effect of a pivot profile across the outward bands.

In short, the V1 core is where you answer the questions:

“What are the primitive objects of the theory?”  
“What algebraic and geometric structure must they satisfy?”  
“What does that structure imply about relativity, quantum behaviour, and gravity?”

It is intentionally **implementation-agnostic**: it tells you what must be true of any correct engine, but not yet how to build that engine in detail.

**The V2 engine layer**

The V2 layer is the **concrete present-act engine** that realizes those abstract requirements:

* It gives you a **discrete worldsheet** of sites indexed by an integer “time” coordinate (k), with a world record (W\_k) and a qualia record (Q\_k) at each site. These records play the roles of ON and IN respectively, but in a way that is directly usable by an implementation.
* It formalizes the **engine pipeline**:
  + selectors that enumerate a finite set of candidate next states,
  + a hinge equality test on a finite feature alphabet (ensuring IN/ON agreement at the hinge),
  + a stack of boolean/ordinal feasibility gates (time windows, granularity levels, structural and context-resolved admissibility, and a gravity-like ParentGate),
  + a ratio-lexicographic acceptance rule and a PF/Born ties-only sampling step,
  + a commit step that updates the records and extends the history.
* It introduces **typed budgets** ((\Delta \tau, \Delta t, \Delta x)) associated with each committed act and enforces a discrete Minkowski-type relation between them as a typing constraint. Composition of these budgets and the no-skip rule give you the engine-side realization of light cones and time dilation.
* It encodes **scale and geometry** into a manifest:
  + a six-band context ladder tied to physically meaningful scales (nanometre band, micron band, UGM band, Earth band, galactic band, cosmic band),
  + per-band choices for feature resolution, gate thresholds, and schedules,
  + fixed conversion factors (like the mapping between inner and outer tick units) derived from the hinge identity.
* It imposes **implementation discipline**:
  + finiteness (finite candidate sets, finite feature alphabet),
  + locality (no long hops in (k)),
  + curve-ban (no continuous weights or scores in control; all geometry appears via configuration and diagnostics),
  + explicit manifest and audit structures so the engine can be checked and reproduced.

This layer answers the questions:

“How does one tick actually run, step by step?”  
“What are the concrete data structures and operations?”  
“How do we guarantee, in a strict combinatorial setting, that the engine respects the abstract constraints of the formal core?”

**Realization under explicit constraints**

The crucial architectural point is that **the engine is not free to do anything it likes**. Its design is shaped by the constraints coming from the formal core *and* by additional requirements you have imposed for implementability and auditability:

* The formal core dictates relationships like the invariant interval, the IN/ON split, the existence of a hinge band where geometry “turns neutral”, the necessity of an arrow of time, and the structure of collapse and Born weights.
* The implementation constraints dictate that:
  + all control logic must be discrete and finite,
  + feasibility decisions must be made with predicates and integer thresholds,
  + probabilistic behaviour must arise only from structural ties, not from injected randomness,
  + geometry and scale information must be encoded in static configuration (manifest) and diagnostics, never as hidden weights in the control path.

V2 is what you get when you **take V1’s formal requirements and insist on realizing them under those implementation constraints**. Every part of the engine has a “because” in V1, and every abstract structure in V1 has a “here is how we do it concretely” in V2.

The rest of this bridge document will unpack that relationship theme by theme: it will show, for each ontological primitive, operator, geometric construct, and emergent physical feature, how the formal definition in the V1 core is mirrored by a specific architectural choice in the V2 engine.

**1.2 Realization Under Constraint**

The engine layer is not just an arbitrary implementation sitting next to the formal core. It is the **image of the formal core after you impose a very specific set of realizability constraints**. This subsection spells out those constraints and how they shape the design of the Present-Act Engine.

You can think of the design problem this way:

“Given the V1 formal core, what is the simplest engine that  
– respects all of its structural requirements, and  
– can actually be run, checked, and reproduced in a discrete setting,  
while using only finite, combinatorial control?”

The answer to that question is V2.

**1.2.1 Finiteness and discreteness**

The first and most basic constraint is **finiteness**:

* Every present act must deal with **finite sets**: a finite set of candidate world updates, a finite set of qualia candidates, a finite alphabet of features.
* There must be **no hidden infinities** in the control path: no continuous parameter curves, no unbounded data structures, no implicit limits.

In the engine, this shows up as:

* A finite **neighbourhood** at each site (k): only a finite set of potential next sites are considered.
* A finite **feature alphabet** Ξ: all “information” that matters to the decision is encoded in a finite set of tags and phase bins.
* Finite **candidate sets** (W\_{k+1}) and (Q\_k): the engine enumerates a finite list of possible next-world and qualia pairs and decides among them.

From the V1 side, this is how you respect the fact that each tick is a **finite slice** of the infinite relational web. The formal core already assumes that each carrier has a finite interface to its neighbourhood; the engine makes this finiteness explicit and enforceable.

**1.2.2 Locality and the no-skip rule**

The second major constraint is **locality**:

* V1 assumes that each tick connects a present to its immediate successor, not to arbitrary distant instants.
* The engine enforces this as a **no-skip rule**: every commit advances the discrete “time” index by exactly one, and no control operation can directly consult or modify non-adjacent sites in the (k)-direction.

In the engine, this is realized by:

* Restricting the **candidate enumeration** to moves from ((x, k)) to a finite set of ((x', k+1)).
* Ensuring that all **gates and budgets** apply only to these local candidate transitions.

This is what keeps the emergent geometry causal in the familiar sense: light cones and relativistic constraints arise from composing these local steps, not from adding a metric by hand.

**1.2.3 Curve-ban: predicates, not weights**

A central design decision is the **curve-ban**: the engine’s control path may not use continuous weights, similarity scores, or fitted curves.

From the formal core, you know that:

* There is a **pivot function** (g(D)) and a dimension profile (D(n)) that modulate couplings and curvature.
* There is an invariant interval and various geometric relations that can be expressed in continuous form.

However, in the engine you insist that:

* Every feasibility decision is made by a **predicate** or an **integer comparison** – a gate that says “pass” or “fail”, or orders candidates lexicographically by simple integer ratios.
* All continuous or scale-dependent information is relegated to **configuration and diagnostics**:
  + into the **manifest**, where you choose band structure, feature resolutions, and gate thresholds once per run, and
  + into **diagnostic computations**, where you may fit curves or compute continuous quantities after the fact, without steering the engine.

The result is that geometry, scales, and pivots are expressed as **which predicates are present and how strict they are**, not as floating-point weight functions inside the control logic. This respects the formal core’s geometric structure while keeping the engine’s decision process purely combinatorial.

**1.2.4 Structured randomness, not arbitrary noise**

Another constraint is that any randomness must be **structural**, not injected ad hoc.

The formal core tells you that:

* Certain situations – genuine **ties** between alternatives – should give rise to Born-type probabilities.
* Quantum-style behaviour is not due to arbitrary noise, but to the structure of the relational possibilities at a hinge.

The engine realizes this by:

* Using a deterministic, lexicographic acceptance rule that reduces the candidate set as far as it can using only gates and residual ratios.
* Invoking randomness **only when there is an exact tie** after all deterministic criteria have been applied, and
* Deriving the probability weights from a **primitive graph structure** (a simple adjacency matrix on the tie set) via the Perron–Frobenius eigenvector, and then squaring its components to obtain Born-style weights.

In other words, the only randomness allowed in the engine is the randomness required by the formal core’s Born-rule structure, and it appears only in the precise place where the formal theory says it should: at a tie in the present plane, not as a general-purpose noise source.

**1.2.5 Typed budgets as built-in geometry**

Finally, the engine builds the **geometric constraints** of the formal core into **typed budgets** rather than adding them as external checks.

From the V1 point of view, you know that:

* There must be an **invariant interval** relating inner time-like separation, outer time-like separation, and space-like separation.
* There must be a way to read out special relativistic phenomena (cones, time dilation) from the tick structure.

The engine satisfies this by:

* Associating with each committed act a budget triple ((\Delta \tau, \Delta t, \Delta x)), typed so that only certain combinations are allowed.
* Requiring that these combinations satisfy a discrete analogue of the invariant interval as a **typing condition**: if a candidate act would violate this relation, it is simply not a valid act.
* Letting **composition of acts** and the no-skip rule generate cones and dilation automatically, as a by-product of these budget constraints.

Thus, the engine does not “compute geometry on the side”; it is **built so that only geometrically consistent acts are even expressible**.

**1.2.6 Summary: V2 as V1 under realizability constraints**

When you put these constraints together – finiteness, locality, curve-ban, structural randomness, and typed budgets – you get a very narrow design space. The Present-Act Engine is the outcome of taking the formal core seriously and then insisting on:

* fully discrete, finite control,
* explicit configuration of all scale-dependent structure,
* minimal and principled use of randomness, and
* hard-wired geometric typing.

This is what is meant by saying that **V2 is V1 realized under constraint**. The engine is not an embellishment; it is what the formal theory looks like when you force it into a concrete, auditable, combinatorial form.

**1.3 One Theory, Two Faces**

It is easy, especially when you are deep in the details, to start thinking of V1 and V2 as two separate constructions – an abstract “math version” and a concrete “engine version.” The core message of this subsection is that **there is only one theory**, and V1 and V2 are simply **two faces of that single framework**.

You can think of them as different coordinate charts on the same underlying structure:

* V1 is the **structural/variational chart**: it describes the geometry of the present, the operator algebra, the context ladder, and the global constraints that any realization must satisfy.
* V2 is the **algorithmic/constructive chart**: it describes, in explicit, implementable terms, how one present leads to the next under those constraints.

Neither is more “real” than the other. They are complementary views of the same object.

**1.3.1 Structural view (V1)**

In the structural view:

* The focus is on **relationships and invariants**:
  + the primitive status of present moments,
  + the distinction between inner retention and outer standing,
  + the ladder of context roles and how dimension changes across it,
  + the operator algebra that transforms carriers of experience.
* Relativity, quantum behaviour, and gravity are seen as **necessary consequences** of:
  + how presents must relate to one another if there is to be a coherent history, and
  + how the ladder geometry and pivot function constrain those relations.

This view is powerful for proving theorems, understanding global structure, and seeing **why** certain engine features are required. It is less concerned with step-by-step implementation details.

**1.3.2 Engine view (V2)**

In the engine view:

* The focus is on **how a single present act is actually decided**:
  + what data structures exist at each tick (world and qualia records),
  + how candidates for the next tick are enumerated,
  + which predicates are applied to test feasibility,
  + how acceptance and, when unavoidable, probabilistic selection are carried out,
  + how budgets are assigned and updated.
* The same physical phenomena – relativity, quantum behaviour, gravity – appear as:
  + the emergent behaviour of histories generated by this engine,
  + the consequences of picking particular manifest values (band structure, scales, thresholds)  
    that match the constraints uncovered in the structural view.

This view is powerful for coding, running simulations, designing experiments and diagnostics, and testing whether a concrete implementation actually respects the abstract principles.

**1.3.3 Mutual dependence**

The two faces are not independent; they **need each other**:

* Without the structural view:
  + the engine design would look arbitrary or overly restrictive,
  + there would be no clear reason for choices like PF/Born ties-only, or the particular form of budgets, or the strict curve-ban.
* Without the engine view:
  + the structural theory would remain a **specification without a machine**,
  + there would be no direct path to simulations, empirical tests, or code-level verification,
  + many of the theory’s claims would be harder to ground in something that can, in principle, be run.

The bridge document formalizes this mutual dependence. For each major concept, it shows:

* what it means in the structural view, and
* how it is implemented in the engine view.

In that sense, the bridge is a **pairing** of the two faces: each structural statement has an engine counterpart, and each engine rule has a structural justification.

**1.3.4 A single acceptance test for consistency**

Thinking in terms of “two faces of one theory” also gives you a practical consistency test:

* When you modify something in the **formal core**, you can ask:  
  *“What does this change imply for the engine? Which gate, budget, or data structure is affected?”*  
  If there is no clear answer, the change might be under-specified or inconsistent with the existing realization.
* When you modify something in the **engine**, you can ask:  
  *“What abstract property or theorem is this supposed to realize? Does it still do so?”*  
  If there is no clear structural story backing the new engine rule, it may violate the underlying theory.

The bridge is where those questions get answered. It makes the “two faces” picture **explicit and checkable**, so that as you refine V1 and V2 over time, you can keep them firmly aligned.

In summary, there is one underlying AR framework. V1 expresses it as a network of relational axioms, operators and invariants. V2 expresses it as a present-act engine with concrete data and rules. This bridge, and especially the themes that follow, are written to keep that unity front and centre while giving you the practical tools to move comfortably between both faces.

**2. Ontology Crosswalk: Mapping Primitives Between V1 and V2**

**2.1 Ticks ↔ Sites (k)**

One of the most basic correspondences between V1 and V2 is the relationship between a **tick** in the formal core and a **site index (k)** in the engine. At first glance they look similar – both are integers that step forward – but it is worth being very clear about what each of them means and why they can be identified.

**V1 view: ticks as atomic present updates**

In the V1 formal core, a **tick** is the **atomic unit of change** in the web of presents:

* Each tick marks the transition from one present to the next along a given experiential thread.
* The formal core does not need to assume a numerical label at first; it merely requires that there is a well-ordered sequence of presents along each “worldline” and that this sequence can be treated as discrete steps.
* When things are made more explicit, you label these steps by an integer (k), so that carrier (\mathcal{C}*k) represents the state of a given present at tick (k), and (\mathcal{C}*{k+1}) is its successor after one application of the operator algebra (Renew, Sink, Trade, Sync, framing).
* The **arrow of time** in the formal core is expressed as the fact that ticks only advance in one direction (there is no operator that undoes a tick and rewinds to (\mathcal{C}\_{k-1})). The ledger construction then shows that the record component grows monotonically with (k).

In other words, in V1 a tick is “whatever it takes” to get from one carrier to the next along a given line of experience, under the prescribed operators and constraints.

**V2 view: sites as engine positions in the outward index**

In the V2 engine, the integer (k) appears explicitly as an **outward index** on a discrete lattice or worldsheet:

* Each pair ((x, k)) represents a location in a grid where:
  + (k) is the discrete “time-like” coordinate that always advances by +1 at each commit, and
  + (x) is a discrete “space-like” index (or multi-index) labelling positions in whatever lattice or graph the engine is running over.
* At each (k), the engine maintains:
  + a world record (W\_k) – a finite set of outward proposals or environment states, and
  + a qualia record (Q\_k) – a finite set of inward/pixel-like proposals.
* The **engine cycle** takes you from site index (k) to (k+1): selectors enumerate a finite candidate set for (k+1), hinge equality and gates test feasibility, acceptance chooses one winning candidate, and then the world and qualia records at (k+1) are updated accordingly.
* The engine enforces the **no-skip rule**: there is no control path that jumps from (k) to (k+2) directly; all transitions go through (k+1). This mirrors the idea in the formal core that there is no such thing as “skipping a present.”

So in V2, the integer (k) labels the **steps of the engine’s outward evolution**. A single pass through the engine pipeline, resulting in a commit, corresponds to advancing (k) by one.

**Identifying ticks with site indices**

With that in mind, the crosswalk is straightforward:

* A **tick** in the V1 sense – an atomic update of a carrier’s state under the operator algebra – corresponds, in the engine, to a **single successful cycle** that advances the outward index from (k) to (k+1).
* The **sequence of ticks** along a given experiential thread in V1 corresponds to following a single history (a chosen path of commits) through the lattice in V2, stepping along the (k) axis one site at a time.
* The **irreversibility** of ticks in the formal core is realized by two engine constraints:
  + the no-skip rule (you never decrease (k)), and
  + the absence of any operation that undoes a commit (once a winner is chosen at (k\to k+1), that decision is recorded and not re-opened).

There is, therefore, **no extra degree of freedom** associated with V2’s site index beyond what V1’s tick index already implied. The engine simply makes the tick index explicit and couples it to spatial indices and data structures in a way that the formal core left abstract.

In practical terms, whenever V1 talks about “at tick (k)” or “the transition from tick (k) to (k+1),” you can read that as referring to **one full engine cycle from site index (k) to site index (k+1)**. And whenever V2 speaks about the sequence of sites ((x\_0, k\_0), (x\_1, k\_1), \dots) along a committed history, you can read that as the sequence of carriers (\mathcal{C}*{k\_0}, \mathcal{C}*{k\_1}, \dots) in the formal core.

The rest of the ontology crosswalk will build on this identification: with ticks and site indices aligned, we can now map the contents of carriers (IN/ON) to the engine’s world and qualia records, and then map abstract operators to the steps of the engine pipeline.

**2.2 PMS / Carrier ↔ (Wₖ, Qₖ)**

Once ticks and site indices are aligned, the next step is to map the **state carried at each tick** in V1 onto the **records held at each site** in V2. In the formal core, this state is represented by a **Present-Moment Sphere (PMS)** or **carrier**. In the engine, it is represented by the pair of records ((W\_k, Q\_k)) at site (k). This subsection explains that correspondence.

**V1 view: Present-Moment Sphere and carriers**

In the V1 formal core, each tick (k) is associated with a **carrier** (\mathcal{C}\_k) that packages together everything relevant about that present:

* The carrier has an **inner face (IN)**: this is what the present has *already retained* – the nested structure that encodes its past. In different parts of the formalism this is described as:
  + the IN network or inner graph of relations,
  + the record component in the ledger,
  + the inward-looking side of the Present-Moment Sphere.
* The carrier has an **outer face (ON)**: this is how the present **stands among other presents** – the outward associations or “exposure” of the present to its environment. This includes:
  + the ON network or outer graph of relational links to other carriers,
  + what the present offers outward as a possible influence or participation in larger contexts.
* The Present-Moment Sphere language emphasizes that there is a **boundary** between IN and ON:
  + the PMS boundary is where inner and outer must fit together consistently,
  + the boundary projector in the formal core is the operation that enforces this fit, selecting the allowed boundary configurations.

From this perspective, the carrier (\mathcal{C}\_k) at tick (k) is **one object with two faces**: an inward record (IN) and an outward stance (ON), separated and connected by a hinge-like boundary.

**V2 view: world and qualia records (W\_k) and (Q\_k)**

In the V2 engine, the same idea is realized using **two finite records at each site**:

* The **world record (W\_k)** is a finite set of outward proposals or “environmental hypotheses” at site (k):
  + each element of (W\_k) represents a possible configuration of the external world, as the engine is currently treating it,
  + it is the outward-facing content: what the system takes to be available in its environment at this tick.
* The **qualia record (Q\_k)** is a finite set of inward or pixel-like proposals:
  + each element of (Q\_k) represents a possible configuration of what is being “experienced” or registered internally at this tick,
  + it is the inward-facing content: how the system’s inner structure is presenting itself in discrete, addressable units.

If you look at the engine pipeline:

* Candidates for **next-world** configurations come from manipulating and extending (W\_k).
* Candidates for **qualia** at the current or next tick come from manipulating and extending (Q\_k).
* The hinge equality and gates compare elements of (W\_{k+1}) against elements of (Q\_k) in terms of their features, deciding which pairings are admissible for the next act.

In other words, at each site (k), the engine explicitly holds **two finite, typed sets**, one outward and one inward, that are the operational counterparts of ON and IN.

**Mapping IN/ON to world/qualia**

With that in mind, the crosswalk can be stated simply:

* The **IN** part of the V1 carrier (\mathcal{C}\_k) corresponds, operationally, to the **qualia record (Q\_k)** in V2:
  + both are the locus of inward, retained structure,
  + both encode “what has already been gathered and is now being taken as given” at the present.
* The **ON** part of (\mathcal{C}\_k) corresponds to the **world record (W\_k)**:
  + both describe outward, environment-facing content,
  + both represent “how this present stands among others” as far as the engine is concerned.

The difference is that in the formal core, IN and ON are described as parts of one abstract object, whereas in the engine they are **two explicit records with finite content** that can be enumerated, gated, and updated.

From the standpoint of the theory, however, they are playing the same roles: **one inward, one outward**, both bound together at each tick.

**Hinge equality as the PMS boundary**

The V1 PMS boundary is where IN and ON must be consistent. In the abstract description, this is expressed by:

* a boundary projector or hinge operator that selects combinations of IN and ON that satisfy certain consistency conditions at the interface,
* the idea that only certain joint configurations of inner and outer content are allowed to co-exist at a given present.

In V2, this is realized concretely by **hinge equality on features**:

* The engine constructs feature maps:
  + from candidate world elements (w \in W\_{k+1}) into a feature alphabet, and
  + from current qualia elements (q \in Q\_k) into the same alphabet.
* It then forms the **hinge set** as pairs ((w, q)) whose features exactly match:
  + this is the operational definition of “IN and ON are in agreement at the boundary,”
  + any candidate pair that fails this equality test is not even considered further by the gates and acceptance logic.

So where the formal core says “the PMS boundary identifies which IN/ON combinations are allowed,” the engine says “the hinge equality identifies which (W\_{k+1}/Q\_k) pairs are even admissible for the next act.” It is the same concept translated into the language of finite records and feature maps.

**What this means in practice**

Putting this together:

* The V1 carrier (\mathcal{C}\_k) at tick (k) maps to the **pair of records ((W\_k, Q\_k))** at site (k) in V2.
* The **IN** side of (\mathcal{C}\_k) maps to (Q\_k); the **ON** side maps to (W\_k).
* The PMS boundary maps to the combination of:
  + feature maps from world and qualia to a shared alphabet, and
  + a strict equality condition on those features at the hinge.

Whenever V1 speaks about “inner network, outer network, and their boundary at a present,” you can read that as a statement about **what is stored in (Q\_k), what is stored in (W\_k), and which pairs ((w, q)) are allowed to contribute to the next act** in the engine.

Conversely, whenever V2 refers to world and qualia records and to hinge equality, you can see that as the engine’s way of maintaining the **IN/ON split and PMS boundary** that the formal core requires.

**2.3 Abstract State hₖ ↔ Feature Alphabet Ξ**

At each tick, the formal core treats the present not only as IN and ON networks but also as having an **internal state** (h\_k) that summarizes “where we are” in a more compact way. In the engine, this role is taken by the **feature alphabet Ξ** together with the feature values assigned to elements of (W\_k) and (Q\_k). This subsection explains how those two ideas line up.

**V1 view: the abstract state (h\_k)**

In the V1 picture, a carrier (\mathcal{C}\_k) is often thought of in terms of three components:

* IN: the inward record (what has been retained),
* ON: the outward associations (how it stands among others), and
* an **abstract state (h\_k)** that captures, in a compressed, algebraic way, the “type” or “condition” of the present at tick (k).

Depending on context, (h\_k) can be:

* a point in some abstract state space (e.g., a state in a Hilbert-like construction),
* a label that determines which operators are active or which symmetries apply,
* a summary of the boundary configuration that will matter for the next act.

The important thing is that (h\_k) is **not the full IN/ON network**; it is a **finite descriptor** that the operator algebra uses to decide how to update the carrier. It encodes the aspects of the present that are relevant to the next transformation, in a way that is stable under the relevant symmetries and pivots.

**V2 view: the feature alphabet Ξ and feature assignments**

In the V2 engine, there is no separate symbol (h\_k), but the role it plays is taken over by the **feature alphabet Ξ** and the feature values assigned to elements of (W\_k) and (Q\_k):

* Ξ is a **finite set of feature types and allowed values**:
  + discrete tags describing band, shell, pattern type, orientation bin, lane, role, etc.,
  + phase bins for the cyclic part of the state (e.g., a finite partition of the unit circle).
* For each candidate world element (w \in W\_k) or qualia element (q \in Q\_k), the engine computes a **feature vector**:
  + (f\_k(w) \in Ξ) for world-side candidates,
  + (g\_k(q) \in Ξ) for qualia-side candidates.

These feature vectors are what:

* the **hinge equality** compares,
* the **gates** inspect (for example, checking whether a candidate has the right band, shell, or pattern flags),
* the **acceptance logic** uses when computing ratio residuals and deciding between alternatives.

In other words, the engine never needs to inspect the full raw content of the IN/ON networks; it only needs to inspect **feature assignments in Ξ**. Those assignments are the engine’s operational notion of “state.”

**Mapping (h\_k) to feature assignments in Ξ**

With this in mind, the crosswalk can be expressed as:

* The abstract state (h\_k) at tick (k) in the formal core corresponds, in the engine, to the **collection of feature assignments in Ξ for the active elements of (W\_k) and (Q\_k)**.
* Instead of a single symbol (h\_k), the engine holds:
  + a finite set of feature vectors on the world side, and
  + a finite set of feature vectors on the qualia side.

Taken together, these assignments carry the same information that (h\_k) is intended to summarize:

* They determine **which symmetries and constraints** apply (e.g., which band we are in, which gates are active, which shell schedule we are using).
* They determine **which candidates are compatible with the current hinge configuration**.
* They determine the **equivalence classes** of candidates that should be treated as “the same” for the purposes of collapse or selection (for example, candidates with identical feature vectors at the hinge).

Put differently, (h\_k) is the abstract label of “where we are” in a combined state space; the engine realizes that label as **a structured pattern of tags and phase bins in Ξ**.

**Why finiteness of Ξ is compatible with the abstract picture**

The formal core does not require that the state space be infinite in any operational sense; it requires that:

* the tick-operator algebra can distinguish the relevant situations, and
* the invariants (like the interval and pivot behaviour) are respected.

The engine’s choice to use a **finite alphabet Ξ per run** is a realizability constraint:

* For any particular manifest, you decide upfront **which features are relevant** at the scales and bands you care about.
* Those choices determine **which distinctions the engine can make** at runtime.
* If, in a future incarnation, more detail is needed (for example, more refined band distinctions or additional pattern types), you extend Ξ and adjust the manifest accordingly.

From the bridge perspective, this means:

* The abstract state space of V1 can be thought of as **the closure of all feature assignments in all possible Ξ choices** across all manifests.
* A given run of the engine picks out a **finite slice of that space** by specifying a particular Ξ and using it as the operational state representation.

Thus, the finiteness of Ξ **does not contradict** the abstract picture; it is how you make a finite, combinatorial engine that still respects the abstract constraints of (h\_k). The mapping simply says: **for this manifest and this run, the bits of (h\_k) that matter are exactly the bits encoded in Ξ and its feature assignments.**

**Practical reading of the crosswalk**

In practice, when V1 refers to “the state (h\_k) at tick (k)” you can read it, in engine terms, as:

“the pattern of feature assignments in the chosen alphabet Ξ across the relevant elements of (W\_k) and (Q\_k).”

And when V2 talks about “features” or “feature vectors” driving gates and acceptance, you can understand that as:

“the concrete, finite representation of the abstract state that the operator algebra is acting on at this tick.”

This identification is the basis for later sections that will show how **geometric and scale-dependent structure** (captured abstractly by (D(n)) and (g(D))) is encoded into **choices of feature types and thresholds in Ξ**, and how the **Present Plane structure** is realized as phase bins within the same alphabet.

**2.4 IN / ON Split ↔ Inward / Outward Read**

The IN/ON split is one of the central structural ideas in the V1 formal core: each present has an **inner face** (what it retains) and an **outer face** (how it stands among others). In the V2 engine and ontology, the same idea appears as the distinction between **inward depth** and **outward standing**, or informally, the **inward read** and the **outward read** of a present. This subsection shows how those two descriptions match.

**V1 view: IN and ON as two faces of the present**

In the formal core:

* **IN** (inner network, inner face):
  + is the side of the present that looks **inward**,
  + contains the nested record of past experiences and relations that have already been condensed,
  + is where the ledger’s record component lives – the accumulation of “what has been taken in” so far.
* **ON** (outer network, outer face):
  + is the side that looks **outward**,
  + encodes how this present stands among other presents in its environment,
  + carries the “exposure” part of the ledger – how this present is available to contribute to larger contexts.

The key is that IN and ON are not two separate objects: they are **two roles** of the same present at a given tick. One role points inward, to nested history; the other points outward, to the larger coordination it participates in.

This is also why the formal core puts so much emphasis on the **IN/ON boundary**: it is the locus where the inward and outward faces of a present must fit together coherently.

**V2 view: inward depth and outward standing**

In the V2 ontology and engine, you describe the same situation using the language of **inward depth** versus **outward standing**, and in practice via the two records (Q\_k) and (W\_k):

* The **inward read** at tick (k):
  + treats the present as a center looking **inward** into what it contains,
  + is realized concretely as the qualia record (Q\_k), which holds the inward/pixel-like structure,
  + corresponds to the narrative that “the present sinks its past into itself” – inward depth increases as more structure is retained.
* The **outward read** at tick (k):
  + treats the present as a center looking **outward** toward the context that contains it,
  + is realized concretely as the world record (W\_k), which holds outward hypotheses about environment and coordination,
  + corresponds to the narrative that “the present stands amid many possible futures and contexts” – outward standing describes its relationships to other centers and containers.

In the qualitative V2 language, this is often described as:

* **“depth”** = how much the present has taken in and can re-relate internally,
* **“standing”** = how the present is positioned in the larger network of presents at its own level and above.

The engine’s world/qualia split and the ontology’s inward/outward language are thus the operational expression of the IN/ON split.

**The crosswalk: IN ↔ inward read, ON ↔ outward read**

The correspondence can now be stated succinctly:

* V1 **IN** ↔ V2 **inward read**:
  + inner network, record component, nested past  
    ↔ inward depth, qualia record (Q\_k), re-relatable internal structure.
* V1 **ON** ↔ V2 **outward read**:
  + outer network, exposure, environment-facing links  
    ↔ outward standing, world record (W\_k), participation in larger contexts.

In both pictures:

* The inward side is about **memory, coherence, and internal plexity**.
* The outward side is about **participation, coordination, and external structure**.

The difference lies only in emphasis: V1 describes this structurally in terms of networks and carriers; V2 describes it functionally in terms of “what the present sees when it looks in” versus “what it sees when it looks out,” and implements that as two explicit records.

**Re-reading the ontology in IN/ON terms**

This mapping also clarifies some of the V2 ontological language:

* When the V2 ontology says that **a present is defined by its inward depth and its outward standing**, you can read that as saying that **a carrier is defined by its IN and ON faces**.
* When it says that **“space” is the outward read of a context and “biology/plexity” is the inward read**, you can see that as:
  + ON being read as spatial layout (how many peers and containers stand side-by-side), and
  + IN being read as the internal, often non-spatial network of relations that gives rise to life-like plexity.

Conversely, when the V1 formal core talks about IN and ON as abstract structures, the V2 ontology gives you an **intuitive, physical-language gloss**: inward for “what we are”, outward for “where we are.”

**Why this matters for the rest of the bridge**

Keeping this crosswalk in mind will simplify later mappings:

* When the bridge discusses the **context ladder**, you can think of each rung as having its own IN and ON, and in V2 as having its own notion of inward depth and outward standing.
* When you look at the **geometry and scale encoding**, you can see that a lot of the work is about how outward standing (ON/world) is shaped by container geometry, while inward depth (IN/qualia) is shaped by nested plexity at smaller scales.
* When you look at **emergent physics**, you will constantly see the inward/outward pairing: inner coherence and outward feasibility are two sides of the same act.

Summarized in one sentence: **the IN/ON split of the formal core and the inward/outward read of the ontology are the same relational distinction**, expressed once in abstract mathematical language and once in implementation-ready terms.

**2.5 Context Ladder n ↔ 6-Band Ladder (−2 … +3)**

The formal core organizes reality into a **context ladder** indexed by integers (n). The V2 ontology and engine make that ladder concrete as a **six-band ladder** of scale roles (−2, −1, 0, +1, +2, +3) tied to physical ranges (nanometre, micron, UGM, Earth, galactic, cosmic). This subsection explains how those two ladders align.

**V1 view: abstract context ladder (n) and dimension profile (D(n))**

In the V1 formal core, the context ladder is introduced as an **integer-labelled hierarchy of roles**:

* Each (n) labels a **context role** relative to some chosen center:
  + (n = 0) marks the **hinge level**: the role corresponding to “this present” as the contextual center.
  + (n < 0) indexes **inner contexts**: progressively more “inside” roles (purer, deeper, more fine-grained structure).
  + (n > 0) indexes **outer contexts**: progressively more “outside” roles (larger, slower, more encompassing coordination).
* The ladder comes equipped with a **dimension profile (D(n))**:
  + for each context role (n), (D(n)) describes the effective fractal dimension of the structures that dominate at that level,
  + the ladder is not arbitrary; there are constraints on how (D(n)) changes as you move inward or outward.
* There is a special **pivot at (n=0)**:
  + (D(0)) is constrained to be exactly 2 in the formal construction,
  + at this pivot, the hinge behaves “neutrally”: certain geometric factors drop out, and the Present-Moment Sphere boundary acts like a 2-sphere.

The ladder is meant to be **role-relative** rather than absolute: if you change the present you are centering on, you re-label the ladder accordingly. But at any given choice of center, the structure “inside” and “outside” that present is organized by this integer index (n) and its associated dimension profile (D(n)).

**V2 view: six-band ladder and physical scale ranges**

In the V2 ontology, this abstract ladder is instantiated as a **six-band ladder** of context roles, each tied to a characteristic physical scale:

* **−2 band (nanoband)**:
  + associated roughly with **1–200 nm** scales,
  + typical representatives: DNA, chromatin, molecular clusters, nanoscale material structures.
* **−1 band (micron band)**:
  + associated roughly with **0.2–50 μm** scales,
  + typical representatives: cells, small multicellular clumps, tissue micro-architecture.
* **0 band (UGM band)**:
  + associated with **around 0.1–0.2 mm**,
  + this is the **UGM (Universal Geometric Mean)** scale – the hinge grain where inner relations first reliably present as “parts” in the outward read.
* **+1 band (Earth band)**:
  + associated with **kilometre-scale structures on the Earth’s surface** (and more broadly the Earth-surface environment as a context),
  + this is the immediate outward container for individual organisms at our vantage.
* **+2 band (galactic band)**:
  + associated with **kiloparsec-scale galactic structures**,
  + typical representatives: Milky Way-like disks and their structural features.
* **+3 band (cosmic band)**:
  + associated with **Gpc-scale shells**, such as cosmic microwave background structures and large-scale sky-shell patterns.

These bands are not just labels; they are fixed into:

* the **feature alphabet Ξ** (with per-band feature types and resolutions),
* the **manifest** (with per-band gate thresholds and schedules), and
* the **interpretation of evidence** (e.g. GM clustering at specific bands).

In other words, the V2 six-band ladder is the **canonical, evidence-tied instantiation** of the abstract context ladder at our vantage.

**The crosswalk: (n) ↔ band index and physical scale**

The mapping between the two ladders can be summarized as follows, for a typical center (an organism with a nervous system, at our vantage):

* (n = -2) ↔ **−2 band (nanoband)**:
  + V1: very inner role, where structures are highly relational, quantum-like, and barely represented in the outward read.
  + V2: nanoscale structures (DNA, chromatin, nanomaterials) that sit at the boundary between classical and quantum behaviour in the +1 picture.
* (n = -1) ↔ **−1 band (micron band)**:
  + V1: inner role where nested plexity begins to cohere into usable “parts” for the hinge.
  + V2: cellular and tissue micro-scales that function as building blocks for the UGM band.
* (n = 0) ↔ **0 band (UGM band)**:
  + V1: hinge role with D(0)=2, pivot behaviour, PMS boundary.
  + V2: UGM scale (~0.1–0.2 mm), the smallest grain where inner structure reliably presents as distinct parts in the outward read.
* (n = +1) ↔ **+1 band (Earth band)**:
  + V1: first outer role containing the hinge; environment that bundles many 0-centers at once.
  + V2: Earth-surface context, effectively a 2D-like surface container for organisms.
* (n = +2) ↔ **+2 band (galactic band)**:
  + V1: next outer role, where larger containers (like galaxies) become the relevant environment.
  + V2: Milky Way-scale disks and related galactic structures.
* (n = +3) ↔ **+3 band (cosmic band)**:
  + V1: still further outer role, associated with global or horizon-scale behaviour.
  + V2: cosmic shells, CMB-scale structures, and the effective “sky-shell” context at our vantage.

Thus, the V2 six-band ladder is a **finite, physically grounded slice** of the abstract context ladder, focusing on the roles and scales that matter at our human vantage.

**Dimension profile (D(n)) and evidence**

In the formal core, the dimension profile (D(n)) is a theoretical object. In the V2 setting, it is tied to **empirical observations** of fractal dimension and geometric-mean clustering at each band:

* At the **0 band (UGM)**, multiple independent contexts show that boundary-only structures collapse to an effective dimension 2 at around 0.1–0.2 mm, matching (D(0) = 2).
* At the **+1 band (Earth)**, fractal analyses of Earth-surface features show GM clustering in the 1–100 km range and boundary-like behaviour, consistent with a quasi-2D container at that band.
* At the **+2 band (galactic)** and **+3 band (cosmic)**, fractal windows and GM pivots cluster at kpc and Gpc scales with effective dimensions near 2, matching the idea of surface-like outer containers at those roles.
* At the **−2 and −1 bands**, GM clustering in the nanometre and micron ranges shows characteristic behaviours (windows, plateaus, activation thresholds) that match the expected seams between inner roles.

In the bridge context, you can read this as saying:

* (D(n)) is the **abstract dimension profile**;
* the six-band ladder and associated evidence are one **concrete instantiation** of that profile at our vantage, giving content to statements like “D(0)=2 at the hinge” and “outer bands behave surface-like.”

**Role-relativity and re-centering**

Finally, both ladders are role-relative:

* In the formal core, if you pick a different center, you reassign the meaning of (n) (what was +1 might become 0, etc.).
* In the V2 ontology, if you re-center on a different system (for example, a galaxy rather than an organism), the assignment of physical scales to band indices shifts accordingly.

This means the crosswalk is not an absolute statement about “the one true ladder of the universe,” but a statement about **how the abstract context ladder and the six-band ladder line up for a chosen vantage**. The bridge document fixes that vantage as “our present inside the Earth band,” and all mappings are to be read in that light.

In practice, when V1 refers to “moving inward from (n=0) to (n=-1),” you can think “moving from UGM-band behaviour down to micron-scale, cellular behaviour” in the V2 language. When it refers to “moving outward from (n=0) to (n=+1, +2, +3),” you can read that as “moving from the hinge to Earth-surface, galactic, and cosmic container roles” in the six-band ladder.

**2.6 L-Roles ↔ V1 Dynamic Roles**

The L-roles in V2 – **L1, L2, L3** – are a way of naming the different *jobs* a present is doing during a single act: handling past-units, handling outward alternatives, and unifying to one realized outcome. In the V1 formal core, those jobs are already present in the behaviour of the operator algebra and the carrier network, but they were not given the simple L1/L2/L3 labels. This subsection shows how the L-roles line up with V1’s dynamic roles.

**V2 view: L1, L2, L3 as process roles around one act**

In the V2 ontology and engine, you describe each present act in terms of three process roles:

* **L1 – past-units / candidate futures**
  + L1 names the role that deals with **what’s available to be used** for the next act.
  + Inwardly, it is the collection of “pixels from the past” – the inward pieces that can be recombined.
  + Outwardly, it is the cloud of **co-feasible candidate futures** – the different ways the next act could unfold.
* **L2 – outward experiences / environment**
  + L2 names the role that treats each candidate future as a **full experience of time** from the next outward vantage.
  + From this viewpoint, each admissible alternative is no longer “just a possibility,” but an entire potential world-thread at the next level out.
  + L2 is where you see “many possible environments” that could be co-present from the outside.
* **L3 – unifier between alternatives**
  + L3 names the role that **chooses** one outcome so that, at this tick, there is a single realized present.
  + In the engine, this is the combination of:
    - ratio-lexicographic acceptance that narrows the set of candidates, and
    - PF/Born sampling when there is an exact tie.
  + L3 is responsible for the fact that we do not experience a many-worlds superposition at the hinge; we experience just one present outcome.

These are not three separate things in time; they are three **roles** that the same present plays while an act is being decided. They describe the “fan-in” from past, the “fan-out” to futures, and the “collapse” to a single realized path.

**V1 view: implicit L-roles in the operator algebra and carrier dynamics**

In the V1 formal core, the same three roles are encoded in the way carriers and operators behave:

* There is a role that deals with **past-units and potential futures**:
  + The carrier’s IN network holds a collection of **discrete past-units** that can be re-used and re-related,
  + The Renew operator exposes ON as a cloud of possible outward relations that could be realized next,
  + Together, these supply exactly what L1 is supposed to represent: “what can be used to construct the next act.”
* There is a role that treats outward alternatives as **full experiences**:
  + From one step further out, each of those possibilities (when realized) would itself be a new carrier with its own IN/ON structure,
  + The formal core’s notion of a “world-line” or outer environment is precisely this: a sequence of carriers at the next context level that see each alternative as a full present.
* There is a role that **unifies** to one outcome:
  + The Sink operator commits some subset of the ON possibilities into IN,
  + The framing and boundary conditions ensure that the result is a single coherent carrier at each tick,
  + The constructions associated with collapse and Born weights (in the Present Plane language) describe how ambiguity at the hinge is resolved to one realized present.

You did not label these roles as L1, L2, L3 in the V1 text, but they are there as different aspects of what a tick does:

* “Gathering and recombining discrete past-units and possibilities” → L1.
* “Presenting alternatives as full outer experiences from a one-step-out vantage” → L2.
* “Selecting one realized outcome and extending the record” → L3.

The V2 L-role naming is a way of **making these implicit roles explicit** and giving them short, reusable labels.

**Crosswalk: L1 ↔ F/ON and IN-units, L2 ↔ outer carriers, L3 ↔ S/CT**

The crosswalk can be made more concrete by aligning each L-role with specific V1 structures:

* **L1 ↔ Renew F acting on ON + IN past-units**
  + In V1, the Renew operator and the ON side of the carrier encode the outward cloud of potential relations,
  + The IN side encodes the discrete past-units that can be used as building blocks,
  + Taken together, these form the L1 set of “available options” for the next tick.
  + In V2, this role is realized by:
    - the candidate enumeration (selectors over (W\_k) and (Q\_k)), and
    - the set of feasible candidates (F\_k) that survive the early gates.
* **L2 ↔ carriers one level out (outer context / CS)**
  + In V1, if you look at the next context level out (for example, from 0 to +1), each potential future at the 0-level corresponds to a full carrier at the +1 level – an entire “experience-of-time” for the environment.
  + This is what the collective sphere and outer network constructions are about: how many 0-level lines of experience are synchronized into a single higher-level context.
  + In V2, L2 is the role where:
    - each candidate branch in (F\_k) is considered as a **full engine history segment**, and
    - you ask which subsets of those branches are jointly coherent as one environment in the +1 band (this is what the L2 coherence checks and context graph logic capture).
* **L3 ↔ Sink S + framing CT**
  + In V1, the Sink operator and the framing operations define which alternatives are actually committed into the record, and under what conditions there is a single coherent carrier after the tick.
  + This is where you see the formal version of “collapse” and the requirement that at each tick you land in a single carrier, not in a superposition of incompatible carriers.
  + In V2, L3 is realized by:
    - the ratio-lexicographic comparison of residuals, which deterministically narrows the candidate set as much as possible, and
    - the PF/Born ties-only selection when multiple candidates remain exactly equal after all deterministic criteria,
    - followed by committing the winner and discarding the others.

In short:

* **L1** is the engine’s name for “the Renew/ON + IN past-units role; the many local possibilities”.
* **L2** is the engine’s name for “the outer-carrier role; the way those possibilities appear as full experiences one level out”.
* **L3** is the engine’s name for “the Sink/Framing role; the act that unifies alternatives into one realized present”.

**Why naming the roles matters**

Giving these roles explicit names (L1, L2, L3) does more than just tidy up the language:

* It makes it easier to talk about **what the engine is doing** at each part of the pipeline, without having to constantly unpack the full operator algebra.
* It clarifies how **branching, environment, and unification** are distinct but tightly coupled aspects of one act, which helps when you are designing or analysing simulations.
* It provides a **natural bridge** between the formal theory and the ontology:
  + L1 resonates with the idea of “past pixels” and “branching futures” in the present-first narrative,
  + L2 resonates with the idea of “environment” and “shared objective world,”
  + L3 resonates with the idea of “the single realized now” that we actually experience.

From the bridge perspective, the L-role terminology is simply an explicit naming of dynamic roles that V1 already required. It allows you to state, in a single sentence, how the engine realizes each piece of the formal tick dynamics, and it sets up later sections that will use L1/L2/L3 statistics (like multiplicity, coherence, tie-rate) as diagnostic signatures for different context bands and physical regimes.

**3. Operator Algebra ↔ Engine Cycle**

**3.1 Engine Pipeline Overview**

In the V1 formal core, a single tick is described as the action of the **operator algebra** on a carrier: Renew, Sink, Trade, Sync, and framing act (in a particular pattern) to transform one present into the next. In the V2 engine, the same transformation is realized as a **pipeline** of discrete steps that takes you from site (k) to site (k+1). This subsection gives a high-level overview of that pipeline so we can later map each part of the operator algebra to the corresponding engine step.

**V2 view: one tick as a five-stage pipeline**

In the V2 engine, a single “tick” from (k) to (k+1) can be summarized as the following stages:

1. **Enumeration (candidate generation)**
   * Starting from the current world and qualia records ((W\_k, Q\_k)), the engine uses **selectors** to enumerate a finite set of candidate next-world records and candidate pairings with the current qualia.
   * Conceptually, this builds the “cloud of possibilities” for what might happen next – both outwardly (how the world could change) and inwardly (how the present might experience that change).
2. **Hinge equality (boundary match)**
   * For each candidate pair ((w\_{k+1}, q\_k)) – where (w\_{k+1}) is a proposed next-world element and (q\_k) is a current qualia element – the engine computes feature vectors in the alphabet Ξ and compares them.
   * Only pairs that **match exactly** in their features at the hinge are retained. All others are discarded.
   * This is how the engine enforces the requirement that inner and outer faces of the present “fit together” at the boundary.
3. **Feasibility gates (Θ, κ, structural, ParentGate, CRA)**
   * The surviving candidates pass through a stack of **boolean/ordinal gates**:
     + a **time gate** (Θ) that checks stability over a ladder of outward windows,
     + a **part/granularity gate** (κ) that ensures coherence and persistence at appropriate scales,
     + **structural gates** that enforce contiguity, minimal degree, orientation uniqueness, and other local consistency properties,
     + the **gravity-like gate** (ParentGate) that encodes feasibility gradients tied to containers (for example, inward monotone schedules in shells),
     + **context-resolved admissibility gates** (CRA) that forbid certain common-mode commits that would collapse distinct contexts.
   * Each gate is purely predicate-based: it either passes or filters out a candidate; no weights or soft scores are used.
   * The result of this stage is a set of **feasible candidates**, those that are structurally allowed and context-consistent.
4. **Acceptance (ratio-lex and PF/Born ties-only)**
   * The engine then compares the feasible candidates using a **ratio-lexicographic rule**:
     + candidates are assigned integer or ratio-valued residuals describing how well they satisfy outward, inward, and cross conditions,
     + these residuals are compared lexicographically in a fixed priority order,
     + a “fewest-acts” rule may break ties at the residual level by preferring simpler histories.
   * If this deterministic process leaves **exactly one best candidate**, that candidate is accepted.
   * If it leaves a **true tie** – multiple candidates with exactly the same residual profile and minimal act count – the engine constructs a primitive adjacency structure over the tied set, computes a Perron–Frobenius eigenvector, and uses the squared components as Born-like probabilities to pick a single winner at random.
   * This is the unique place where randomness enters, and it is entirely tied to structural ties at the hinge.
5. **Commit and update (record and budgets)**
   * Once a winner is chosen, the engine **commits**:
     + the world record is updated to (W\_{k+1}) based on the chosen candidate,
     + the qualia record is updated to (Q\_{k+1}) accordingly,
     + the tick index advances from (k) to (k+1).
   * At the same time, the engine assigns **typed budgets** ((\Delta \tau, \Delta t, \Delta x)) to the act, obeying the invariant-interval-like typing constraint.
   * The history (ledger) is extended: a new entry is added that cannot be undone, and various diagnostics (counts, tie-rates, band statistics) can be recorded for later analysis.

From a purely engine-oriented perspective, that is what “one act” or “one tick” looks like: enumerate, check boundary, test feasibility, accept via deterministic rule (with a tie-resolving Born step if necessary), then commit and log.

**V1 view: one operator cycle on a carrier**

In the V1 formal core, a single tick is described more abstractly as:

* Starting from a carrier (\mathcal{C}\_k) with IN/ON structure and state (h\_k),
* Applying operators like:
  + **Renew**, which exposes new ON relations (potential futures),
  + **Trade**, which reallocates structure between IN and ON and across contexts,
  + **Sync**, which enforces alignment with other carriers and with the collective context,
  + **Sink**, which commits certain ON possibilities into IN,
  + **framing operations**, which ensure that the result is a coherent carrier with a well-defined boundary.
* Ending at a new carrier (\mathcal{C}*{k+1}), with updated IN/ON structure and state (h*{k+1}), and with the ledger updated accordingly.

The explicit details of how these operators are sequenced can vary depending on which part of the theory is being emphasized, but the conceptual structure is always the same: **many potential ways forward, constraints from geometry and context, one realized outcome and an updated record**.

**High-level correspondence**

At this overview level, the correspondence between the operator cycle and the engine pipeline looks like this:

* **Renew** ↔ **enumeration** of candidates from (W\_k) and (Q\_k).
* **Boundary projector / hinge** ↔ **hinge equality** on features in Ξ.
* **Trade / Sync / context constraints** ↔ **Θ, κ, structural, ParentGate, CRA gates** enforcing feasibility and coherence.
* **Sink + framing** ↔ **acceptance (ratio-lex and PF/Born) + commit**, producing one new present and extending the ledger.

The rest of the operator-bridge section will take this overview and unpack it in more detail, mapping each named operator and each kind of constraint in the formal core to the corresponding engine-step, gate, or budget rule. Here, the important point is simply that **both V1 and V2 agree about what one tick does**:

* a cloud of possibilities is generated,
* those possibilities are tested against structural and contextual constraints,
* one outcome is chosen (deterministically when possible, probabilistically when necessary),
* and the present is updated with a new IN/ON split, a new state, and an extended record.

**3.2 Renew F ↔ Enumeration**

The **Renew** operator in the V1 formal core is the part of the algebra that exposes **new outward possibilities** – it is what makes a present “open” to multiple ways of continuing. In the V2 engine, this role is taken by the **enumeration stage**, where selectors generate a finite set of candidate next-world configurations and their pairings with current qualia. This subsection shows how these two notions align.

**V1 view: Renew as exposure of potential futures**

In the structural picture:

* Starting from a carrier (\mathcal{C}\_k), the Renew operator acts on its ON side to **generate or expose a cloud of possible outward relations**:
  + new links that could be formed to other carriers,
  + new ways in which this present could participate in larger contexts,
  + new configurations of the boundary that would be compatible with its current state (h\_k).
* Conceptually, Renew:
  + does not yet decide which of these possibilities will be realized,
  + does not commit them into the record (that is the job of Sink),
  + simply lays out the **menu of potential futures** that are compatible with the current carrier before constraints and other operators are applied.

You can think of Renew as the formal core’s way of saying: “from here, these are the possible ways the world could go next, as seen from the outward face of this present.”

Renew is also closely associated with the **L1 role** in the ontology: it generates the outward half of the L1 cloud of candidated futures that can then be filtered and selected.

**V2 view: enumeration via selectors**

In the engine:

* The enumeration stage is where the engine, at site (k), uses **selectors** to construct a **finite set of candidate next-world states**:
  + starting from the current world record (W\_k), it applies discrete rules (selectors) to propose a set of possible (W\_{k+1}) configurations,
  + each candidate in this set represents a “possible way the world could look” after the next tick.
* In parallel or in coordination, the engine constructs or identifies **candidate pairings with the current qualia**:
  + elements of (Q\_k) are matched with these candidate (W\_{k+1}) states to form potential ((w\_{k+1}, q\_k)) pairs,
  + this is the raw material for the hinge equality step – the engine is constructing all the pairs that might be admissible.
* The key points about enumeration:
  + it is **finite** – selectors produce a finite candidate set,
  + it is **structured** – the rules for generating candidates are part of the manifest and respect locality and the no-skip rule,
  + it does not yet apply gates, budgets, or probabilities – it just builds the L1 cloud the engine will work over.

From the engine’s perspective, enumeration answers the question: “given ((W\_k, Q\_k)), what are the possible ((W\_{k+1}, Q\_k)) pairings that we should even consider for the next act?”

**Crosswalk: Renew ↔ selectors and candidate sets**

The correspondence between Renew and enumeration can be stated as:

* **Renew in V1** corresponds to the **selector-driven enumeration of candidate next-world states in V2**:
  + both are concerned with **opening up possibilities**,
  + both act on the outward face of the present (ON in V1, (W\_k) in V2),
  + both generate a cloud of potential futures without yet deciding which will occur.
* The **L1 cloud** of candidates in the engine is the concrete representation of the **Renewed ON possibilities** in the formal core:
  + in V1, you talk about ON being expanded or renewed into many possible outward relations,
  + in V2, you talk about selectors generating a finite list of candidate (W\_{k+1}) states and their pairings with current qualia.

Schematically:

* V1:  
  [  
  (\mathcal{C}\_k, \text{ON}\_k) \xrightarrow{\text{Renew F}} { \text{ON}*k^{(i)} }*{i}  
  ]
* V2:  
  [  
  (W\_k, Q\_k) \xrightarrow{\text{selectors}} { (W\_{k+1}^{(i)}, Q\_k) }\_{i}  
  ]

In both cases, you start from one outward configuration and end up with **many candidate outward configurations**.

**What Renew does *not* do (and enumeration does not either)**

An important aspect of the crosswalk is clarifying what Renew and enumeration **do not** do:

* Renew **does not**:
  + enforce all structural or geometric constraints,
  + resolve conflicts between possibilities,
  + pick a single outcome.
* Enumeration **does not**:
  + apply the hinge equality (that is the next stage),
  + apply Θ, κ, structural, ParentGate, or CRA gates (those are later),
  + compute ratio residuals or invoke PF/Born (that belongs to acceptance).

Both Renew and enumeration are **pre-selection** operations. They expand the space of potential futures but leave the actual choice and consistency checks to later steps in the tick.

**Relation to L1 and later diagnostics**

Because the enumeration stage realizes Renew in engine terms, it is natural that:

* L1 statistics in later diagnostics (like L1 multiplicity – how many feasible candidates exist at a tick) are derived from the **size and structure of the enumerated candidate set and its survivors after minimal gating**.
* When the bridge later talks about **L1 multiplicity being higher at some bands (e.g., quantum-like regimes)**, it is describing a Renew/Enumeration behaviour: more candidate futures are opened up and survive initial tests at those bands.

Thus, whenever you see V1 statements about the Renew operator populating a cloud of ON possibilities, you can read that in engine terms as “the selectors at this scale and band will generate a large, structured candidate set at the enumeration stage.” And whenever V2 documentation talks about how selectors are designed or why the candidate sets look the way they do, you can trace that back to the Renew operator’s role in the formal core via this crosswalk.

**3.3 Sink S ↔ Acceptance & Commit**

In the V1 formal core, the **Sink** operator is responsible for turning potential into actuality: it is the step that takes some of the outward possibilities exposed by Renew and **commits them into the inner record** of the carrier. In the V2 engine, this role is carried by the **acceptance and commit stages**, where the engine chooses a single winning candidate and updates the records at site (k+1). This subsection explains how Sink and acceptance/commit correspond.

**V1 view: Sink as commitment into the record**

From the structural perspective:

* After Renew has exposed a cloud of outward possibilities (different ways ON could be configured), and after other operators and constraints have acted, the Sink operator:
  + **selects which outward relations are actually realized**, and
  + **incorporates them into IN**, the inward record of the next carrier.
* Conceptually, Sink:
  + is the point where “what could happen” becomes “what did happen,”
  + updates the ledger’s record component,
  + produces a new carrier (\mathcal{C}\_{k+1}) that includes the committed content as part of its inner structure.
* Sink also enforces that **each tick lands in a single coherent carrier**:
  + it is not allowed to produce a superposition of incompatible carriers at the same tick,
  + it must produce one well-defined IN/ON split and state (h\_{k+1}) for the next step.

In the formal core, the details of how the choice is made can be discussed via the Present Plane, amplitudes, and Born weights, but the essence is that Sink is the **actualization operator**.

**V2 view: acceptance and commit**

In the engine, the same job is carried out by two tightly connected stages: **acceptance** and **commit**.

* The **acceptance stage**:
  + takes the set of feasible candidates that have passed all gates at tick (k),
  + assigns to each candidate a set of integer/ratio-valued residuals (how well it satisfies outward, inward, and cross conditions),
  + orders these candidates lexicographically by those residuals, with a fixed priority ordering and “fewest-acts” tie-breaking when applicable,
  + and then:
    - if the ordering yields a **unique best candidate**, that candidate is accepted deterministically, or
    - if there is a **true tie** (multiple candidates with identical residual profiles and act counts), it constructs a primitive adjacency structure on the tie set, computes a Perron–Frobenius eigenvector, and uses its squared components as Born-like weights to randomly select one winner.
* The **commit stage**:
  + takes the chosen winner and **updates the engine state**:
    - the world record is set to the chosen (W\_{k+1}),
    - the qualia record is updated to (Q\_{k+1}), reflecting the new inward configuration,
    - budgets ((\Delta \tau, \Delta t, \Delta x)) are assigned in accordance with the invariant-interval typing rule,
    - diagnostic counts and logs are updated,
    - the tick index advances from (k) to (k+1).

After commit, all other candidates are discarded. They remain “what could have happened,” but they do not enter the record.

From the engine’s perspective, acceptance and commit together answer: “which candidate becomes the next actual present, and how do we update the records to reflect that?”

**Crosswalk: Sink ↔ (ratio-lex + PF/Born) + commit**

With these pictures in mind, the correspondence is:

* **Sink S in V1** corresponds to the combination of:
  + **acceptance**: the **selection** of one candidate from the feasible set, via deterministic lexicographic ordering plus PF/Born ties-only sampling when necessary, and
  + **commit**: the **incorporation** of that candidate into the engine’s record as the new ((W\_{k+1}, Q\_{k+1})), with updated budgets and logs.

More concretely:

* In V1, Sink:
  + takes a cloud of ON possibilities (already constrained by geometry, context, and other operators),
  + chooses one outward configuration to actually realize,
  + folds that configuration into IN to form the next carrier’s record.
* In V2, acceptance and commit:
  + take a finite set of feasible ((W\_{k+1}, Q\_k)) candidates (already filtered by hinge equality and gates),
  + choose exactly one of them using deterministic comparison and, if needed, structurally derived probabilities,
  + store the chosen world as (W\_{k+1}) and update qualia to (Q\_{k+1}), thus building the next inward record.

Schematically:

* V1:  
  [  
  {\text{ON}\_k^{(i)}}\_i \xrightarrow{\text{constraints}} {\text{admissible ON}\_k^{(j)}}*j \xrightarrow{\text{Sink S}} \text{IN}*{k+1}  
  ]
* V2:  
  [  
  {(W\_{k+1}^{(i)}, Q\_k)}*i \xrightarrow{\text{gates}} { \text{feasible } (W*{k+1}^{(j)}, Q\_k)}*j \xrightarrow{\text{ratio-lex + PF/Born}} W*{k+1} \xrightarrow{\text{commit}} (W\_{k+1}, Q\_{k+1})  
  ]

In both views, this is the step where **one and only one outcome** is chosen and integrated into the next present.

**Determinism vs probabilistic collapse**

The Sink ↔ acceptance mapping also clarifies the relationship between determinism and probabilistic behaviour:

* In V1:
  + the formal core allows for both **deterministic** ticks (when the structure singles out one outcome) and **probabilistic** ticks (when the structure yields a Present Plane with multiple equal-amplitude options).
  + the Born rule determines the probabilities of different outcomes in those ambiguous cases.
* In V2:
  + the acceptance stage is **deterministic** as far as possible: ratio-lex reduces the candidate set to the smallest possible subset using only structural predicates and integer-comparison logic,
  + **only when a true tie remains** does the engine invoke PF/Born sampling, deriving weights from a primitive adjacency structure on the tie set,
  + this is how the engine realizes the idea that randomness appears only where the abstract structure demands it – at ties in the present plane – and nowhere else.

In both pictures, Sink is the point where ambiguity (if any) is resolved, and the record is advanced.

**Updating the record and budgets**

Another part of the crosswalk is how Sink’s effect on the ledger is realized:

* In V1, Sink:
  + updates the **record component** (I) of the ledger: one more act has been integrated, and the inner network has deepened accordingly,
  + implicitly updates **exposure** and **capacity** components (how much the present is still open to further relations, how much it can still hold).
* In V2, commit:
  + updates the **world and qualia records**: these are the concrete carriers of IN and ON content for the next tick,
  + assigns **typed budgets** ((\Delta \tau, \Delta t, \Delta x)) that reflect how much inner and outer separation this act carried,
  + appends to diagnostic logs and counters (which are the engine’s way of keeping track of (I), (E), (K) over many ticks).

So when the formal core talks about Sink “deepening the record,” the engine’s translation is “committing a chosen candidate and updating (W\_{k+1}), (Q\_{k+1}) and the budgets accordingly.”

**Summary**

In summary:

* Renew and enumeration open up the space of possible futures.
* Gates and other operators enforce structure and feasibility.
* **Sink in the formal core** and **acceptance/commit in the engine** are the **actualization step**: they decide which possible future actually becomes the next present and embed that decision into the inner record.

The mapping Sink ↔ (acceptance + commit) is one of the most central in the bridge: it is how the abstract notion of “one realized present per tick” is made concrete in the engine, and it is where the Born-rule structure of the formal theory finds its operational home in PF/Born ties-only sampling and the irreversible advance of the engine’s history.

**3.4 Trade T, Sync C, Framing CT ↔ Gates & Hinge**

Between “open up possibilities” (Renew/enumeration) and “actualize one outcome” (Sink/acceptance+commit), there is a middle layer of structure in the V1 formal core: the operators **Trade T**, **Sync C**, and **Framing CT**. They decide *which* possibilities are even legitimate, *how* they must fit together across contexts, and *when* a context is “locked in” enough to allow Sink to act.

In the V2 engine, these roles are realized by the combination of:

* **hinge equality** (feature match at the boundary), and
* the suite of **feasibility gates** (Θ, κ, structural gates, ParentGate, CRA).

This subsection explains how those pieces correspond.

**V1 view: Trade, Sync, and framing as middle-layer structure**

In the structural picture, after Renew has exposed a cloud of ON possibilities, the operators Trade, Sync, and CT act to shape and constrain that cloud:

* **Trade T**
  + Trade is responsible for **reallocating structure between IN and ON and between neighbouring contexts** while preserving overall capacity and consistency.
  + Intuitively, it moves “where the detail lives”: some detail may be taken off ON and pushed into IN (or vice versa), or shifted between adjacent context levels, as long as the hinge conditions and pivot constraints are honoured.
  + Trade enforces **granularity and persistence**: it prevents pathological configurations where a present would suddenly gain or lose fine-grained structure in a way that violates the ladder’s rules.
* **Sync C**
  + Sync enforces **coherence and alignment across multiple carriers** and between a carrier and its outer context.
  + It ensures that, for a given tick, carriers that are supposed to be co-present (e.g., many 0-level streams inside a +1 context) have compatible boundary states and respect shared constraints.
  + Sync is what ties individual carriers into a **collective sphere** at the next context level: a coherent environment seen from the “outside”.
* **Framing CT**
  + Framing (often expressed in compositions like C∘T) is the operation that **locks in a particular context frame** for the act:
    - it decides which context (which container, which environment) is relevant for this tick,
    - it ensures that the chosen frame is internally consistent and compatible with the pivot structure,
    - it prevents “frame drift” during the act.
  + In measurement-like situations, framing plays the role of **selecting and stabilizing a measurement context** so that collapse (Sink) has a well-defined meaning.

Together, these operators:

* “massage” the raw possibilities from Renew into a set of **frame-consistent, context-consistent, granularly appropriate candidates**,
* enforce that each candidate respects both **internal** (IN-side) and **external** (ON-side) constraints before Sink is allowed to act.

**V2 view: hinge equality and feasibility gates**

In the engine, the same middle-layer structure is implemented by:

* a sharp **hinge equality** step, and
* several layers of **feasibility gates**:

1. **Hinge equality (boundary match)**
   * This is the first “middle” operation after enumeration:
     + the engine maps world candidates (W\_{k+1}^{(i)}) and current qualia elements (Q\_k) into feature vectors in Ξ,
     + it keeps only those pairs ((w\_{k+1}^{(i)}, q\_k)) whose features match exactly.
   * This is the concrete version of **boundary compatibility**:
     + IN and ON must present the same state at the hinge,
     + mismatched combinations are discarded immediately.
2. **Time gate Θ**
   * Θ enforces a **stability condition over outward windows**:
     + it checks that a candidate’s behaviour is stable across a ladder of outward time windows,
     + if a candidate’s residuals fluctuate too much across those windows, it is rejected.
   * This realizes, in engine form, Sync’s requirement that the act be **coherent over the relevant outward span**: you cannot call something “one act” if its outward presentation is wildly unstable over the window.
3. **Part/granularity gate κ**
   * κ enforces **minimal persistent granularity**:
     + it checks that the candidate’s structure persists over some minimal spatial/scale window,
     + it ensures you are not trying to treat a fleeting micro-pattern as a robust part at the wrong level.
   * This is the engine’s way of implementing Trade-like behaviour:
     + if a pattern is too fine or too fragile, κ effectively trades it back into “texture” rather than letting it count as a part at this band,
     + stable patterns survive as parts; unstable ones remain as inward plexity.
4. **Structural gates**
   * These include checks like contiguity, minimal degree, orientation uniqueness, and local history coherence.
   * They enforce that the **shape and connectivity** of candidate configurations are compatible with the structural axioms:
     + no “holes” or disconnected fragments where there should be continuous structure,
     + no impossible orientation patterns,
     + no contradictory local histories.
   * These are Sync/Trade-like constraints at a more “geometric” level.
5. **ParentGate (gravity-like feasibility gate)**
   * ParentGate encodes **feasibility gradients tied to containers**:
     + it enforces an inward-monotone schedule on strictness in radial shells (e.g., closer to a mass, fewer candidates survive),
     + it is rotation-invariant, acting only on radial structure in expectation.
   * This realizes part of the framing behaviour:
     + it “tunes” which candidates are even possible in a given container (Earth band, galactic band, etc.),
     + it ensures the local frame reflects the larger container’s pivot profile.
6. **CRA (Context-Resolved Admissibility)**
   * CRA forbids **common-mode commits** that would collapse distinct contexts inappropriately:
     + if two branches would share a “rail” in a way that makes them context-indistinguishable, CRA can reject one or both,
     + this protects the integrity of context separation.
   * This is another framing function:
     + it keeps distinct contexts from being blended in the same act,
     + it helps fix “which frame we are in” for the purposes of the tick.

Taken together, hinge equality plus Θ, κ, structural gates, ParentGate, and CRA **shape the candidate space** in exactly the way Trade, Sync, and framing do in the formal core: they turn a big, raw cloud of possibilities into a set of candidates that are:

* boundary-compatible,
* temporally stable over the relevant windows,
* spatially and granularly appropriate,
* structurally consistent,
* and contextually well-framed.

**Crosswalk: operators ↔ engine components**

The correspondence can be summarized as:

* **Trade T** ↔ **κ and certain structural gates**
  + Trade reallocates structure between IN and ON and across scales; κ and structural gates decide what counts as a part at this band and what remains as texture or inward plexity.
  + If something fails κ or key structural checks, it is effectively “traded away” from the candidate set at that scale.
* **Sync C** ↔ **Θ and coherence-related structural gates**
  + Sync enforces alignment across carriers and consistency with the outer context; Θ enforces temporal stability, and structural gates enforce spatial and relational coherence.
  + Together, they ensure that what the engine is about to call “one act” is genuinely one coherent event relative to the relevant context.
* **Framing CT** ↔ **combination of hinge equality, ParentGate, CRA, and the gate stack as a whole**
  + Framing selects and stabilizes a context frame in the formal core; the hinge equality plus ParentGate and CRA do exactly this in the engine:
    - hinge equality fixes what the boundary looks like,
    - ParentGate encodes which container we are in and how strict the feasibility gradient is,
    - CRA prevents context mixing by disallowing certain ambiguous or common-mode configurations,
    - the full gate stack ensures all of this is in place before Sink/acceptance.act.

Schematically:

* V1:
  + Renew exposes ON possibilities.
  + Trade / Sync / CT shape them into frame-consistent, context-consistent, scale-appropriate options.
  + Sink chooses and commits.
* V2:
  + Enumeration exposes candidate ((W\_{k+1}, Q\_k)) pairs.
  + Hinge equality + Θ + κ + structural gates + ParentGate + CRA shape them into a feasible, well-framed candidate set.
  + Acceptance and commit choose and store one.

**Why this middle layer matters**

This middle layer is critical because it is **where the abstract geometric and contextual constraints actually bite**:

* It is where the dimension profile, pivot behaviour, and container hierarchy are translated into concrete constraints on which candidates can exist at all.
* It is where “one act” is distinguished from a merely local fluctuation: candidates that do not have the right temporal footprint, spatial coherence, or contextual framing are removed before they can be considered for Sink.
* It is also where **context-sensitive phenomena** (like gravitational feasibility gradients or the difference between quantum-like and classical-like regimes at different bands) are enforced via manifest choices.

When you later see the bridge talk about how **band structure**, **pivots**, or **gravity** are encoded into the engine, it will almost always be via this middle layer of hinge equality and gates – the concrete realization of Trade, Sync, and framing CT in the tick-operator algebra.

**3.5 Admissibility Predicates ↔ Gates**

Both V1 and V2 need a way to say “this candidate is allowed” versus “this candidate is forbidden.” In the V1 formal core, this is expressed in terms of **admissibility predicates** on carriers: conditions that must hold for a carrier or a transformation to be considered valid. In the V2 engine, the same idea is implemented as a collection of **boolean/ordinal gates** that accept or reject candidate acts. This subsection shows how these two ways of talking about admissibility are the same thing viewed from different angles.

**V1 view: admissible carriers and admissible transformations**

In the structural picture, you can distinguish between:

* **Admissible carriers**:
  + A carrier (\mathcal{C}\_k) is admissible if its IN/ON structure, its boundary configuration, and its state (h\_k) satisfy certain **predicates**.
  + These predicates express things like:
    - “the hinge lies in the pivot band,”
    - “the dimension profile around this present is within the allowed range,”
    - “the carrier’s IN/ON split respects the context ladder rules,”
    - “there is no violation of causal or capacity constraints in the way this carrier stands among others.”
* **Admissible transformations**:
  + A transformation from (\mathcal{C}*k) to (\mathcal{C}*{k+1}) is admissible if it respects:
    - the operator algebra’s structural rules (how Renew, Trade, Sync, Sink, CT may be combined),
    - the pivot structure (e.g., not crossing forbidden bands in one tick),
    - and the ledger constraints (monotone record, bounded exposure, capacity).
  + In practice, this means:
    - not all formally imaginable futures are allowed,
    - only those that satisfy the relevant predicates on both the starting and resulting carriers are considered legitimate ticks.

In V1, these predicates are often left implicit (“we require that carriers satisfy conditions A, B, C”), but conceptually you can think of them as **yes/no tests** that encode the physical and geometric constraints the theory imposes.

**V2 view: gates as operational admissibility tests**

In the V2 engine, these admissibility predicates are **made explicit as gates**:

* Each gate is a **boolean or ordinal operator** on candidates:
  + it takes as input a candidate act (usually represented by a pair ((W\_{k+1}, Q\_k)) plus additional local state),
  + it returns “pass” or “fail”, or an integer/ordinal that can be compared in a lexicographic scheme.
* The main gates collectively implement all the admissibility constraints:
  + **Hinge equality** ensures boundary compatibility – the candidate respects the IN/ON boundary condition.
  + **Time gate Θ** ensures that the candidate’s behaviour is stable over the declared outward windows, matching the notion of an “act” rather than a series of incompatible shifts.
  + **Granularity gate κ** enforces band-appropriate “what counts as a part” rules.
  + **Structural gates** (contiguity, degree, orientation, history coherence) enforce local geometric and network predicates.
  + **ParentGate** encodes admissibility relative to containers (e.g. which radial shells and strictness levels a candidate is allowed to inhabit).
  + **CRA gates** enforce context separation, saying “this candidate would blur contexts that must remain distinct; therefore it is inadmissible.”

No continuous weights are involved in these decisions. Each gate either **allows** a candidate to proceed to the next stage or **filters it out** based on simple, checkable conditions. In this sense, each gate is a literal implementation of an admissibility predicate.

**Crosswalk: abstract predicates ↔ concrete gates**

The crosswalk can be summarized as:

* Every **admissibility predicate** in the V1 formal core corresponds to at least one **gate** (or combination of gates) in the engine:
  + if V1 says “carriers must satisfy property P to be admissible,”
  + then V2 implements “property P” as a gate that rejects any candidate that lacks P.
* Conversely, each gate in the engine can be traced back to a **predicate in the formal theory**:
  + hinge equality ↔ “IN and ON agree at the PMS boundary,”
  + Θ ↔ “this candidate counts as one act across the outward window,”
  + κ ↔ “this structure is a part at this band rather than mere texture,”
  + ParentGate ↔ “this candidate respects the container’s feasibility gradient and pivot profile,”
  + CRA ↔ “this candidate respects context separation and does not collapse distinct rails.”

This means that:

* The engine’s **control logic** is nothing more than the V1 admissibility predicates made explicit and made executable.
* If a candidate would violate a V1 admissibility condition, there must be (and in the design there is) a gate that filters it out before it gets anywhere near acceptance.

**Admissibility before selection**

Another important aspect of the mapping is the **order** in which admissibility and selection occur:

* In V1, the conceptual order is:
  + first, apply admissibility predicates to carriers and transformations,
  + only then consider which admissible transformations could actually be realized (e.g. via Sink and the Born structure where needed).
* In V2, the engine pipeline follows the same pattern:
  + enumeration creates a raw set of possibilities,
  + **gates enforce admissibility**, producing a set of feasible candidates that respect all predicates,
  + only then does the acceptance stage compare the feasible candidates and possibly invoke PF/Born for ties.

From the bridge perspective, this is crucial: it means **no candidate that fails a V1 predicate can ever be chosen by the engine**, because it will be filtered out at the gate stage. The acceptance stage only operates over candidates that already satisfy the formal core’s admissibility requirements.

**Practical implication for future changes**

Because gates are the operational form of admissibility predicates, the bridge also tells you how to manage changes:

* If you add or refine an admissibility condition in the formal core (for example, a new constraint on how carriers can relate at a certain band), you must:
  + either show that it is already implicitly enforced by existing gates, or
  + add or modify a gate in the engine to make that condition explicit.
* If you propose a new gate in the engine, you should be able to say:
  + “This gate is the engine-side implementation of predicate P in the formal core,”
  + or, if not, you need to formulate the corresponding predicate in the V1 language.

In this way, the set of gates in V2 and the set of admissibility predicates in V1 remain in one-to-one conceptual correspondence, keeping the engine and the abstract algebra tightly synchronized.

**3.6 Ledger (I, E, K) & Arrow of Time**

In the V1 formal core, the **ledger** ((I, E, K)) is a way to track how the present evolves:

* (I) for **record** (what has been taken in),
* (E) for **exposure** (how much the present is still “standing out” toward others), and
* (K) for **capacity** (how much potential it still has to form new relations).

A key structural result is that the **record component (I)** is **monotone**: it never decreases with tick (k). This monotonicity is what anchors the **arrow of time** in the formal theory. In the V2 engine, these ideas are realized as the **irreversible commit of acts**, the **growth of the world/qualia history**, and the behaviour of the **typed budgets and diagnostics**. This subsection explains the mapping.

**V1 view: ledger and monotone record**

In the structural description:

* The ledger ((I, E, K)) is an abstract device attached to the evolution of carriers. At each tick (k), you can, in principle, assign:
  + (I\_k) – a measure of how much inner structure has been committed so far,
  + (E\_k) – a measure of how much outward exposure or “standing among others” the carrier still has,
  + (K\_k) – a measure of the remaining capacity for forming new relations.
* The formal core shows that, under the tick-operator algebra:
  + (I) is **non-decreasing**: each Sink operation can only add to the record; there is no inverse operation that removes committed structure,
  + (E) and (K) may fluctuate under Renew, Trade, and Sync, but in controlled ways dictated by the ladder and pivot constraints.
* The **arrow of time** is then not an extra postulate but a **consequence**:
  + The fact that (I\_{k+1} \ge I\_k) for all (k) means there is a well-defined direction “from less record to more record,”
  + At the level of lived experience, this is exactly the direction in which we experience “more having happened” – more nested structure, more “past” being retained in the present.

So in V1, the ledger formalizes the idea that:

“Presents can only gain record; they cannot lose it. That is why we have a consistent sense of time’s direction.”

**V2 view: history, budgets, and diagnostics**

In the engine, there is no single variable named (I), but the same structure is present in the way:

* **Histories are built**:
  + Each commit at tick (k) appends a concrete state ((W\_{k+1}, Q\_{k+1})) to the evolving history.
  + Once appended, that state is not removed or overwritten; subsequent ticks build on it.
  + This is the **irreversibility of commit**: there is no operation that “de-commits” a tick and erases its contribution to the history.
* **World and qualia records accumulate structure**:
  + Over time, (Q\_k) can be thought of as containing more and more nested structure: new committed patterns, new persistent parts, new relations that have survived κ and structural gates at various bands.
  + Similarly, (W\_k) reflects a progressively refined picture of the environment – new stable structures, new container relationships encoded via ParentGate and other gates.
  + Even though specific transient candidates are discarded, the overall “depth” of what has been committed grows.
* **Typed budgets ((\Delta \tau, \Delta t, \Delta x))** are assigned and summed:
  + For each committed act, budgets are added to cumulative totals or used to derive diagnostics (e.g., cumulative proper time, cumulative displacement, etc.).
  + These cumulative quantities also only **increase in magnitude** as more ticks occur; they do not rewind.
* **Diagnostics and logs are monotone in information content**:
  + The engine writes out diagnostic records (counts, tie-rates, band statistics, path properties) as ticks accumulate.
  + These logs and counters grow; they are not rolled back when future ticks occur.
  + From the standpoint of the engine’s own self-description, more ticks means more log entries and more data – another reflection of increasing “record.”

All of these elements together form the **engine’s realization of the ledger**:

* The world and qualia histories ({(W\_j, Q\_j)}\_{j\le k}),
* The cumulative budgets per history,
* The diagnostic logs and counters per run.

They are the concrete carriers of “how much has been recorded” at any given tick.

**Crosswalk: ((I, E, K)) ↔ history, exposure, and capacity in the engine**

The bridge can therefore state:

* (I) (record) ↔ **the growing history of committed states** and nested internal structure:
  + Each commit adds to what is retained; no commit is undone.
  + The growth of this history, and of the complexity of (Q\_k) in particular, is the engine’s manifestation of monotone record.
* (E) (exposure) ↔ **how rich the candidate clouds and world relations remain**:
  + In the engine, this is reflected in:
    - the size and diversity of candidate sets at each tick,
    - the behaviour of outward-facing structures in (W\_k),
    - the strictness of gates like ParentGate and CRA that can reduce exposure as the context becomes more constrained.
* (K) (capacity) ↔ **the available combinatorial degrees of freedom**:
  + For the engine, this is seen in:
    - how many distinct features and patterns are encoded in Ξ,
    - how permissive or restrictive the manifest is (which gates are active at which bands and thresholds),
    - how much “space” remains for new patterns to form without collapsing into saturated configurations.

While (E) and (K) require more context to define numerically, the important structural point is that:

* (I) maps to **the monotone growth of the committed history and associated diagnostics**,
* (E) and (K) map to **the richness and constraints of candidate spaces and patterns over time**.

**Arrow of time in engine terms**

Once the crosswalk is made, the arrow-of-time statement in V1 translates directly to engine language:

* In V1: time has a direction because the **record cannot shrink**.
* In V2:
  + There is a **distinguished direction** in the (k) index: commits move (k \to k+1), never backwards.
  + The history of states, budgets, and diagnostics **grows** as (k) increases.
  + There is no operation that moves the engine from a “later” state to an “earlier” one, or that erases the already committed history.

From the experiential perspective, this is exactly what you expect:

* As more ticks happen, the engine has more data, deeper internal structure, and longer histories – it literally “remembers” more.
* That accumulation of structure is what you experience as “more past behind you,” and the one-way extension of history is what you experience as the **direction** of time.

**Practical reading of the crosswalk**

In practice:

* When the formal core talks about the ledger ((I, E, K)) and its monotone record, you can read that in engine terms as:

“the cumulative, irreversible growth of the committed history and its internal complexity.”

* When the engine documentation refers to:
  + the no-skip rule,
  + the impossibility of undoing a commit,
  + the cumulative nature of diagnostics and logs,  
    you can see that as the implementation of:

“ticks only move forward, and the record only increases.”

This completes the mapping between the **abstract ledger and arrow of time** in V1 and the **concrete, irreversible evolution** of the engine in V2.

**4. Geometry, Dimension, Scales and the Manifest**

**4.1 Abstract Ladder D(n) + Pivot g(D) (V1)**

Before we can map geometry and scale into the engine manifest, it is useful to restate the **abstract picture** from the V1 formal core: the **context ladder (n)**, the **dimension profile (D(n))**, and the **pivot function (g(D))**. These are the main structural tools V1 uses to talk about “where geometry lives” in the theory.

**The context ladder and dimension profile**

As described earlier in the ontology crosswalk, V1 organizes roles around a chosen present using an integer context ladder (n):

* (n = 0) is the **hinge role** – the role of the present as contextual center.
* (n < 0) are **inner roles** – “deeper inside” relative to that center.
* (n > 0) are **outer roles** – “further outside” relative to that center.

Attached to this ladder is a **dimension profile** (D(n)):

* For each role (n), (D(n)) is the **effective fractal dimension** of the structures that dominate at that context:
  + inner roles tend to have more “volumetric” or plexity-heavy behaviour (higher or more complex effective dimensions in the inward sense),
  + outer roles often show “boundary-like” behaviour (dimensions closer to 2 when seen as surfaces).
* The profile is not arbitrary: V1 imposes constraints on how (D(n)) can change as you move inward or outward, reflecting the requirement that the overall relational structure remain self-consistent and compatible with the ladder of containers and seams.

The context ladder and dimension profile give you a way to say:

“At this role, the dominant structures look like this kind of fractal, with this effective dimension, and that matters for how couplings and feasibility behave.”

**The pivot at D(0) = 2**

A particularly important constraint in the formal core is the **pivot at the hinge**:

* At the hinge role (n = 0), the effective dimension is fixed to  
  [  
  D(0) = 2.  
  ]
* This is not a cosmetic choice; it is a structural requirement that:
  + the **boundary at the hinge behaves like a 2-sphere**,
  + certain geometric factors simplify exactly at this level (for example, coupling factors associated with surface behaviour become neutral),
  + the Present-Moment Sphere boundary acts, in a precise sense, like a 2D surface across which inner and outer faces are matched.

This “D=2 at the hinge” condition is central. It means that:

* whatever the detailed dimension profile inward and outward, it must **pass through a 2D pivot** at the present,
* the hinge is a place where geometry has a special neutrality: it neither adds nor removes certain kinds of geometric weighting.

In V2 language, this will later correspond to the observation that there is a **scale band** (the UGM band) where boundary structures empirically collapse to D≈2 across many contexts, and to the way the engine treats that band as the pivot grain for parts.

**The pivot function g(D)**

Alongside (D(n)), V1 introduces a **pivot function** (g(D)):

* (g(D)) is a function that **modulates couplings or weights** based on the effective dimension of the context:
  + certain processes are more or less “strong” depending on D,
  + transitions between contexts with different D values can be described in terms of how (g(D)) varies with D.
* At the hinge, where (D = 2), the function has a **special value**:
  + often normalized so that (g(2) = 1),
  + meaning that at the hinge pivot, the geometric factor is neutral.

Conceptually, (g(D)) is:

* a way to encode **how geometry affects feasibility and dynamics** without introducing a full-blown metric or field in the usual sense,
* a way to talk about **“where geometry is doing work”**: when D is above or below the pivot, the function deviates from its neutral value and that deviation shows up in things like effective couplings, delays, or curvature.

Crucially, in the **formal core**, (g(D)) and (D(n)) are defined at the level of the abstract relational structure. They tell you how the **tick-operator algebra and the ladder of roles** must behave, but they **do not specify** any particular implementation mechanics.

**Geometry without a background metric**

The presence of (D(n)) and (g(D)) allows V1 to:

* derive **geometric results** (like an invariant interval, surface-like behaviour at certain roles, and volume-like behaviour at others),
* talk about **curvature and gravity** as manifestations of how the pivot profile changes across the ladder,
* do all of this **without postulating an external background metric** in the usual physics sense.

Instead:

* geometry is **relational**: it lives in the structure of the ladder, the dimension profile, and how operators behave across it,
* the pivot function is the **bridge between structural properties and effective couplings**.

From the perspective of the bridge, this is exactly the information that V2 must encode into its manifest and gates:

* which bands correspond to which roles on the ladder,
* what effective D and g(D) look like at those bands,
* how that then constrains the engine’s admissible configurations and feasibility gradients.

The rest of the geometry-and-manifest section will show how:

* the abstract ladder ((n, D(n), g(D))) is mapped to the **six-band ladder with empirical GM pivots**, and
* how the pivot function is realized **not as a runtime weight**, but as **choices of feature resolution, gate thresholds, and schedules** in the engine’s configuration.

**4.2 Concrete 6-Band Ladder and UGM (V2)**

On the V2 side, the abstract ladder and pivot from the formal core are made concrete as a **six-band ladder** tied to specific physical scale ranges, and a special hinge-scale called the **UGM (Universal Geometric Mean)**. This subsection restates that construction in a way that makes its role as the realization of (D(n)) and the hinge pivot explicit.

**The six bands as concrete context roles**

In the V2 ontology, you describe the context ladder relative to “our” present (an organism with a nervous system on the Earth’s surface) in terms of the following six bands:

* **−2 band (nanoband)**
  + Scale range: roughly **1–200 nm**.
  + Typical representatives: DNA, chromatin, molecular complexes, nanostructured materials.
  + Qualitative character:
    - quantum-regime signatures in the +1 picture are common here (tunnelling, coherence, spin effects),
    - structures are highly relational and not yet reliably presentable as classical “parts” at our scale.
* **−1 band (micron band)**
  + Scale range: roughly **0.2–50 μm**.
  + Typical representatives: cells, organelles, small multicellular aggregates, tissue micro-architecture.
  + Qualitative character:
    - nested plexity begins to look like usable “parts” from the vantage of the UGM band,
    - fractal GM pivot analyses from diverse domains point to strong clustering here, suggesting a genuine seam.
* **0 band (UGM band)**
  + Scale range: around **0.1–0.2 mm**.
  + Typical representatives: small anatomical and material features at the edge of direct perception; many cross-domain GM pivots accumulate here.
  + This is the **UGM scale**:
    - a geometric-mean pivot between an inner and outer scale range,
    - empirically, a scale at which boundary-only contexts consistently collapse to an effective dimension ~2 across many systems,
    - interpreted as the **hinge grain** where inward plexity first reliably presents outward as distinct “parts” at our vantage.
* **+1 band (Earth band)**
  + Scale range: roughly **1–100 km** on the Earth’s surface.
  + Typical representatives: coastlines, rivers, topographic features, atmospheric structures, ecological patterns.
  + Qualitative character:
    - the Earth’s surface behaves effectively like a 2D container for our experiences: a “sheet” on which objects and environments are laid out,
    - fractal GM pivot and dimension analyses show clustering around these kilometric scales with effective dimensions near 2 when analysed as surfaces.
* **+2 band (galactic band)**
  + Scale range: **hundreds of parsecs to a few kiloparsecs**.
  + Typical representatives: galactic disks, spiral arms, ISM distributions at disk scales.
  + Qualitative character:
    - structures at this band (e.g., HI maps, HII region distributions) often show GM pivots in the ~0.3–1 kpc range with effective projected dimensions near 2,
    - interpreted as a disk-like, surface-like context container at the +2 role.
* **+3 band (cosmic band)**
  + Scale range: **Gpc-scale shells** associated with the observable universe.
  + Typical representatives: CMB maps, full-sky source distributions, large-scale sky-shell statistics.
  + Qualitative character:
    - fractal analyses at this band highlight GM pivots on the order of the horizon scale and effective dimensions ~2 on the sky sphere,
    - interpreted as the outermost container relevant to our vantage: the “sky-shell” boundary.

These bands are not arbitrary bins. They are **empirically motivated** partitions where:

* finite fractal windows and their GM pivots tend to cluster, and
* effective dimensions often line up with the surface-like behaviour predicted by the formal ladder at relevant outward roles.

**UGM as the hinge-scale in V2**

Within the six-band ladder, the 0 band – the UGM band – plays the role of the **hinge**:

* It is defined as a **geometric mean** between an inner and outer scale range:
  + for example, between a microscopic inner bound and an Earth-scale outer bound,
  + yielding a length on the order of **0.1–0.2 mm**.
* Across a wide variety of systems and domains:
  + surface roughness studies,
  + fracture and flow in materials,
  + biological tissue structure,
  + certain perceptual thresholds,  
    you see **scaling behaviour changing** or **fractal windows cutting off** near this scale.

From the engine’s perspective, UGM is:

* the **smallest outward grain** at which the engine treats structures as **parts** in the +1 outward read,
* the scale at which:
  + the IN/ON split is stable for parts (below it you mainly have inward plexity),
  + the hinge conditions can be enforced reliably in terms of discrete spatial units,
  + the Θ and κ gates can distinguish “one act with parts” from “finer-scale inner dynamics.”

In the bridge language, this is exactly the **V2 realization of the V1 hinge at (n=0)** where (D(0) = 2) and (g(D)) is neutral. UGM is the concrete scale at which:

* the boundary behaves like a 2-sphere (in the empirical sense: D≈2 for boundary-only contexts),
* inward and outward faces can meet without additional geometric weighting.

**How the six-band ladder appears in the manifest**

In the engine configuration, the six-band ladder and UGM are encoded into the **manifest**:

* Each band has:
  + its own **feature types** in Ξ (e.g., band labels, shell indices, pattern signatures appropriate to that scale),
  + its own **Θ and κ ladder entries** (time windows and granularity thresholds tuned to that scale),
  + its own **ParentGate schedule entries** (radial shells and strictness levels appropriate to the band’s containers).
* UGM specifically appears as:
  + a **cutoff or hinge parameter** in the way spatial coordinates are discretized and interpreted,
  + the reference scale used when deciding:
    - what counts as “microscopic” versus “mesoscopic” in κ,
    - how to bucket spatial separations into bands in the feature alphabet,
    - which band a particular pattern belongs to when gates and diagnostics operate.

This is how the abstract statements “there is a hinge role with D=2 and a pivot g(D)” and “there are inner and outer context roles” are turned into:

* a **finite, declarative configuration** (manifest) that the engine can read at run start, and
* a set of **band-aware gate behaviours** that enforce the intended geometry.

**Summary of the crosswalk at this level**

At the geometric-and-scale level:

* The abstract ladder (n) ↔ the **six-band ladder** (−2 to +3) in V2.
* The formal hinge at (n=0), D(0)=2 ↔ the **UGM band** at ~0.1–0.2 mm, where empirical D≈2 and parts first reliably appear in the +1 read.
* Outer roles with surface-like behaviour ↔ the **Earth, galactic, and cosmic bands**, where boundaries again show D≈2 when analysed appropriately.
* The pivot function and dimension profile ↔ the **per-band configuration** of feature types, gate ladders, and ParentGate schedules in the manifest.

The remaining subsections in this geometry-and-manifest section will make this mapping more explicit for D(n) and g(D), showing exactly how geometry is moved out of the control path and into **configuration and diagnostics** in the engine.

**4.3 Encoding D(n), g(D) in the Manifest (Curve-ban Compliant)**

The formal core uses the **dimension profile (D(n))** and **pivot function (g(D))** to encode how geometry affects feasibility and couplings at different context roles. The engine, however, must obey the **curve-ban**: no continuous weight functions or fitted curves are allowed in the control path. This subsection explains how the information in (D(n)) and (g(D)) is moved into the **manifest and gate configuration** so that geometry shapes the engine’s behaviour **without** violating the curve-ban.

**From abstract profiles to per-band configuration**

In V1, you can think of (D(n)) and (g(D)) as high-level statements like:

* “At this role, structures are effectively D-dimensional,”
* “At this D, the geometric factor (g(D)) should be such-and-such,”
* “Transitions between roles with different D should be favoured or suppressed in particular ways.”

In the engine, these statements are turned into **per-band configuration choices** in the manifest:

* For each band (−2, −1, 0, +1, +2, +3), you decide:
  + which **feature types** in Ξ are relevant (e.g., band labels, shell indices, pattern flags that reflect the kind of fractal structure you expect at that D),
  + which **Θ windows** and **κ thresholds** apply (e.g., how long an act must be stable, how coarse/fine a part must be to count at that band),
  + which **ParentGate schedules** to use (e.g., how inward strictness changes with radius in that band’s containers),
  + which **structural gates** are active or emphasized (e.g., specific contiguity or degree rules appropriate to 2D-like vs 3D-like behaviour).

These are all **static configuration decisions**: they are chosen once per manifest, offline, based on your understanding of (D(n)), (g(D)) and the evidence, and then used as integer parameters and feature definitions during runs. There is **no runtime call to (g(D))** inside the control logic.

**Geometry as gate structure, not weights**

The pivot function and dimension profile influence the engine through **which predicates exist and how strict they are**, not through runtime multipliers:

* If the formal core says “at this role, geometry wants to suppress certain patterns,” you implement that by:
  + adding or tightening **gates** that reject those patterns (for example, κ thresholds that require higher persistence or contiguity),
  + sharpening **ParentGate** schedules (inward shells where fewer candidates are allowed),
  + adjusting **Θ windows** so that only appropriately stable patterns count as one act.
* If the formal core says “at the hinge (D(0)=2), geometry should be neutral,” you reflect that by:
  + choosing gate settings at the UGM band that are **balanced** (for example, not favouring any particular orientation or shell beyond what is required for basic coherence),
  + using **isotropic** structural predicates (rotation-invariant in expectation),
  + ensuring that ParentGate at that band is configured so that it does not introduce additional asymmetry beyond the abstract pivot.

In this way, geometry is **implemented as a pattern of yes/no tests and discrete thresholds**. The engine never uses a floating-point value of D or a continuous curve (g(D)) to tilt acceptance; instead, those functions have been “compiled” into the structure and parameters of the gates.

**Encoded once, checked many times**

An important aspect of encoding (D(n)) and (g(D)) into the manifest is that it is done **once**, at configuration time:

* You use the abstract theory and empirical evidence to decide:
  + which bands correspond to which roles on the context ladder,
  + what effective dimensions and pivot behaviours you need at those bands,
  + how strict gates and schedules should be to reflect those behaviours.
* You then **fix** these choices in the manifest:
  + band definitions and mappings from physical units to bands,
  + Θ and κ ladders per band,
  + ParentGate shell structure and strictness per container and per band,
  + any band-specific structural gate toggles.

Once the manifest is loaded, the engine:

* only sees **integers, enums, and feature tags**,
* never recomputes (D(n)) or evaluates (g(D)),
* simply executes its gate pipeline using the parameters that, by design, already encode the desired geometric behaviour.

This is what makes the implementation **curve-ban compliant**: all continuous or curve-based thinking has been pushed into **pre-run configuration**, and the runtime control path remains purely combinatorial.

**Geometry in diagnostics, not control**

While the control path cannot use curves or continuous weights, **diagnostics** are free to do so:

* After a run, you can:
  + compute effective fractal dimensions (e.g., via scaling of counts) to see if the engine’s behaviour matches the expected (D(n)) at each band,
  + fit envelopes (like 1/r or log patterns) to measured quantities to see if they match the pivot-informed predictions,
  + check whether empirical GM pivots and plateau behaviours line up with the band definitions implied by the manifest.

This split is deliberate:

* **Geometry in control**: represented indirectly via configuration (gates, thresholds, schedules) derived from (D(n)) and (g(D)).
* **Geometry in diagnostics**: measured and fitted after the fact, used to validate and adjust the manifest for future runs.

If a diagnostic reveals that behaviour at a given band does not align with the intended (D(n)) or (g(D)), you update the manifest (for example, change certain thresholds or feature definitions) and rerun. The theory remains consistent: the formal geometric structure tells you **what you are aiming for**, the manifest and gates encode your current approximation to that, and diagnostics tell you **how close you are**.

**Summary at this level**

At the level of the bridge:

* The **dimension profile and pivot function** from the formal core are **not** used as continuous runtime weights;
* instead, they inform **what the manifest looks like**:
  + how the six bands are defined,
  + which predicates exist at each band,
  + how strict those predicates are,
  + what the container schedules look like.

This ensures that **V2 respects the geometric content of V1** while still obeying the engine’s core discipline: no curves, no continuous weights, only combinatorial gates and finite feature alphabets in the control path.

**4.4 Boundary Projector 𝔅 ↔ Hinge Equality**

In the V1 formal core, the **boundary projector** (often denoted 𝔅 or as part of the Present-Moment Sphere construction) is the operation that selects those IN/ON combinations that are allowed to coexist at a present. In the V2 engine, this role is carried by **hinge equality** on the feature alphabet: only those world/qualia pairs whose features match at the hinge are considered admissible. This subsection explains how these two notions are the same structure in different guises.

**V1 view: boundary projector as IN/ON consistency filter**

In the structural picture:

* The **Present-Moment Sphere** has a boundary where IN (inner network) and ON (outer network) meet.
* Not every arbitrary combination of IN and ON is allowed. There are **compatibility conditions** that must be satisfied at the boundary.
* The **boundary projector 𝔅** expresses this by:
  + acting on candidate IN/ON configurations,
  + enforcing the hinge conditions (for example, that certain quantities match, that the dimension and pivot conditions at (n=0) are respected, that the sphere behaves like a D=2 boundary),
  + discarding those combinations that violate these rules and keeping only those that are **boundary-consistent**.

You can think of 𝔅 as a **filter** at the present-moment boundary:

* It “looks” at the IN/ON pair and decides whether this pair is **admissible as one present**.
* If the conditions are satisfied, the combination passes and can be used in further operator algebra (Renew, Trade, Sync, Sink, etc.).
* If not, that combination is simply not part of the theory’s set of allowable carriers.

The boundary projector is thus the abstract expression of the idea that “a present must have a consistent inner and outer face.”

**V2 view: hinge equality as feature-level boundary test**

In the engine, the same concept is implemented via **hinge equality**:

* For each candidate pair ((w\_{k+1}, q\_k)) – a proposed next-world element and a current qualia element – the engine computes:
  + a feature vector (f(w\_{k+1})) in the alphabet Ξ for the world side, and
  + a feature vector (g(q\_k)) in the same alphabet Ξ for the qualia side.
* The **hinge test** then says:
  + keep only those pairs ((w\_{k+1}, q\_k)) for which  
    [  
    f(w\_{k+1}) = g(q\_k)  
    ]  
    as elements of Ξ, possibly including a phase component modulo the chosen phase resolution,
  + discard all other pairs.

Operationally:

* This is the first serious consistency test between inner and outer faces in the engine pipeline.
* It enforces that **the way the world is presented outward at the next tick** is compatible with **the way the current qualia record is presenting inward** at the hinge.
* This is done on a finite, discrete feature set, but conceptually it is enforcing the same kind of matching that the formal core’s boundary projector encodes.

From the engine’s point of view, hinge equality is the step where it says:

“Only world/qualia combinations that present the **same boundary state** are even candidates for the next act.”

**Crosswalk: 𝔅 ↔ hinge equality on Ξ**

With these descriptions in mind, the crosswalk is straightforward:

* The boundary projector 𝔅 in V1 corresponds to **hinge equality on the feature alphabet Ξ in V2**:
  + both act at the **present boundary**,
  + both enforce **consistency between inner and outer faces**,
  + both act as **filters**: incompatible combinations are removed, compatible ones proceed.
* In abstract terms, 𝔅 is defined in whatever mathematical language the formal core is using (e.g., as a projection onto a subspace of boundary-consistent states).
* In engine terms, hinge equality is implemented as:
  + a pair of deterministic feature maps (f) and (g),
  + a set of equality checks on the resulting discrete tags and phase bins,
  + a simple rule: if they match, keep the candidate; if they do not, discard it.

The boundary projector’s high-level condition “IN and ON agree at the hinge” becomes, in the engine, “world and qualia features match at the hinge.”

**Neutrality at the hinge and D=2**

The formal core gives special status to the hinge role (n=0), where the effective dimension is D=2 and the pivot function is neutral. At that pivot:

* The boundary projector is, in a sense, **simplest**:
  + no extra geometric weighting is applied at this band,
  + the condition for boundary consistency is “pure” equality or equivalence on the hinge quantities.

The engine reflects this in its hinge equality implementation:

* At the UGM band (the V2 realization of the hinge), the feature mapping and equality conditions:
  + are **isotropic** (no azimuthal bias),
  + do not depend on additional continuous weights or band-specific multipliers,
  + encode neutrality: the engine does not favour certain boundary configurations via hidden weights; it only checks that the two sides match within the declared feature resolution.

That is, the engine’s hinge equality is designed to be the **neutral matching operation** appropriate to the D=2 pivot: geometry has already been encoded into the manifest and gates; the equality itself is just “do these boundary descriptions agree?”

**Hinge equality as the first gate**

In the engine pipeline, hinge equality is typically the **first gate** after enumeration:

* It shrinks the raw candidate set down to those candidates that have a coherent boundary.
* Only then are other gates (Θ, κ, structural, ParentGate, CRA) applied to these boundary-consistent candidates.

This mirrors the formal idea that **boundary consistency is a precondition** for everything else:

* In V1, 𝔅 is often treated as a prior or separate step: you talk about admissible carriers as those that already satisfy the boundary conditions.
* In V2, hinge equality takes on that role: any candidate that fails the boundary condition never reaches the rest of the control logic.

Thus, in both views:

* You first ensure “this is even a legitimate present (IN and ON can meet here),”
* and only then worry about timing, granularity, structure, container gradients, and collapse.

**Practical reading of the mapping**

In practical terms, whenever the formal core mentions:

* “the PMS boundary,”
* “boundary consistency between IN and ON,”
* “projecting onto the subspace of allowed boundary states,”

you can read that in engine language as:

“the hinge equality step that checks whether feature maps on world and qualia match in Ξ, and discards any mismatches.”

Conversely, whenever the engine documentation talks about:

* feature-level matching at the hinge,
* the initial filter that prunes incompatible world/qualia pairings,

you can understand that as the concrete, finite-alphabet realization of:

“the boundary projector constrained to D=2 at the hinge, ensuring IN/ON compatibility before any other dynamics are applied.”

This completes the mapping for the boundary operator: the abstract 𝔅 at the present-moment sphere corresponds directly to hinge equality on the feature alphabet in the engine.

**4.5 Present Plane (𝒫, J) ↔ Phase Bins Ξ\_phase**

In the V1 formal core, the **Present Plane** ((\mathcal{P}, J)) is the structure that carries **phase-like information** at the hinge. It is where amplitudes live and where the Born rule is formulated. In the V2 engine, this role is taken over by the **phase component of the feature alphabet**, Ξ\_phase: a finite set of phase bins that are used in hinge equality and in tie resolution. This subsection shows how these two structures correspond.

**V1 view: the Present Plane and complex structure J**

In the abstract picture:

* The **Present Plane (\mathcal{P})** is a space associated with the hinge where:
  + states relevant to a particular act are represented,
  + interference and collapse are described,
  + amplitudes are assigned and manipulated.
* It is equipped with a **complex structure (J)**:
  + (J) acts like multiplication by (i) in an ordinary complex vector space,
  + this allows the formalism to encode phase information, interference patterns, and the distinction between amplitude and probability (via squaring).

On this Present Plane:

* **Amplitudes** are vectors in a complexified space;
* The **Born rule** is expressed as “probabilities are given by the squared modulus of amplitudes”;
* **Collapse** can be described as selecting one ray or component from a superposition, consistent with the probabilities derived from these amplitudes.

In other words, the Present Plane is the formal core’s way of representing **phase-coherent structure** at the hinge, where many potential next states coexist and must be compared.

**V2 view: phase bins Ξ\_phase in the feature alphabet**

In the engine, there is no explicit complex vector space visible in the control path, but there is a **phase component** of the feature alphabet Ξ:

* The alphabet Ξ is typically split conceptually into:
  + a **phase part**, Ξ\_phase, and
  + a **discrete tag part**, Ξ\_disc (for band labels, shell indices, pattern flags, etc.).
* Ξ\_phase is a **finite partition of the unit circle**:
  + the continuum of phases (\theta \in [0, 2\pi)) is discretized into bins,
  + each bin corresponds to an element of Ξ\_phase,
  + a candidate’s phase information is mapped to one of these bins by an engine-defined function.

In practice:

* When constructing feature vectors for world and qualia candidates, the engine includes a **phase bin** component:
  + for example, (f(w\_{k+1}) = (\text{phase\_bin}, \text{tags})),
  + (g(q\_k) = (\text{phase\_bin}, \text{tags})).
* Hinge equality compares both the tag part and the phase bin:
  + two candidates can only match at the hinge if they lie in the same phase bin,
  + this enforces a **phase-consistency condition** at the level of finite resolution.

Additionally, when the engine constructs the **tie adjacency structure** for PF/Born sampling:

* It uses the phase-bin structure in conjunction with the adjacency pattern to define a **primitive “phase-coherent” connectivity** among tied candidates.
* The Perron–Frobenius eigenvector of that adjacency acts as the engine’s **amplitude vector**, and its squared components give the sampling probabilities.

Thus, Ξ\_phase is the engine’s operational representation of “phase information on the present plane,” albeit in a discretized form.

**Crosswalk: ((\mathcal{P}, J)) ↔ finite phase bins and adjacency**

The crosswalk can be stated like this:

* The **Present Plane ((\mathcal{P}, J))** in the formal core corresponds to:
  + the **phase-bin component Ξ\_phase** of the feature alphabet, and
  + the **adjacency structure** among tied candidates at the hinge.
* The complex structure (J) corresponds to:
  + the **cyclic nature** of the phase bins (mod (2\pi)), and
  + the way the engine treats these bins as encoding the “direction” or “phase” of a candidate in a finite approximation to a complex plane.

When the engine:

* groups candidates by **identical feature vectors**, including phase bins,
* builds an adjacency matrix on that group,
* computes the Perron–Frobenius eigenvector and then squares its components,

it is performing, in finite-alphabet form, the operations that:

* the formal core describes as assigning amplitudes on the Present Plane and then applying the Born rule.

The key idea is that **amplitudes live in the structure of the tie-set adjacency**, and **phase information is stored in Ξ\_phase**, so the combination of the two is the engine’s version of ((\mathcal{P}, J)).

**Why discretizing phase is compatible with the formal picture**

The formal core assumes a richer, potentially continuous Present Plane, but nothing in the structure requires that:

* an implementation must use infinite phase resolution, or
* amplitudes must be represented as exact complex numbers in the engine.

Instead, from the bridge perspective:

* The **abstract Present Plane** is the limit of all possible finite approximations;
* A given engine manifest chooses a particular **phase resolution** (number and arrangement of bins in Ξ\_phase), which is appropriate to the scales and phenomena being modeled;
* The **PF/Born step** ensures that probabilistic behaviour respects the abstract structure (probabilities derived from “amplitudes squared”) even when amplitudes are represented implicitly via eigenvectors of discrete adjacency matrices.

If more refined phase behaviour becomes important in a future application, you can:

* increase the **number of phase bins** in Ξ\_phase,
* adjust the way adjacency is defined among tied candidates,
* and thereby get a finer approximation to the abstract Present Plane without changing the underlying theory.

**Present Plane as a hinge structure in both views**

In both V1 and V2, the Present Plane (abstract) and the phase-bin structure (concrete) play their roles **only at the hinge**:

* They are used when multiple candidates are both:
  + structurally admissible, and
  + boundary-consistent (via the boundary projector / hinge equality).
* They provide the **extra structure needed** to:
  + meaningfully define interference and coherence in the abstract,
  + compute Born-rule probabilities over tied candidates in the engine.

So, whenever V1 talks about:

* amplitudes on the Present Plane,
* the action of the complex structure (J),
* squaring amplitudes to get probabilities,

you can read that, in engine terms, as:

“discrete phase bins in Ξ\_phase combined with adjacency on the tie set define an implicit amplitude vector, which is then squared to give PF/Born sampling weights.”

Conversely, when V2 describes:

* phase bins,
* tie graphs,
* PF/Born selection,

you can understand that as the **finite, combinatorial realization of the Present Plane and its complex structure** in the formal core.

**5. Emergent Physics in Both Views (SR, QM, Gravity)**

**5.1 SR: Invariant Interval ↔ Typed Budgets**

One of the most important checks of coherence between V1 and V2 is **special relativity**: in the formal core, you derive an **invariant interval** and the usual light-cone structure from the tick algebra; in the engine, you enforce a **typed budget relation** between (\Delta \tau), (\Delta t), and (\Delta x) for each act, and then recover cones and time dilation by composing acts. This subsection explains how those two pictures are the same.

**V1 view: invariant interval from relational structure**

In the V1 formal core, special relativity is not assumed as a background structure; it is **derived** from the way the tick-operator algebra and the context ladder behave. The key points are:

* Each tick carries both an **inner, time-like separation** (how much the present changes internally) and an **outer, time/space separation** (how much it changes relative to its context and neighbours).
* Under suitable symmetry assumptions and pivot conditions (especially at the hinge), you can show that **there exists a quadratic invariant** of the form  
  [  
  (\Delta t)^2 = (\Delta \tau)^2 + \frac{(\Delta x)^2}{c^2},  
  ]  
  where:
  + (\Delta \tau) is a measure of inner, proper-time-like separation,
  + (\Delta x) is a measure of spatial separation in the outward read,
  + (\Delta t) is the “environmental” time separation as seen from the outer context,
  + (c) is a conversion factor connecting inner and outer tick units.
* By composing ticks and imposing the no-skip and monotonicity constraints, you recover:
  + **light-cone structure**: paths cannot exceed a slope set by (c) when plotted in the appropriate coordinates,
  + **time dilation and length contraction**: different “worldlines” experience different amounts of (\Delta \tau) for the same (\Delta t) and (\Delta x), consistent with relativistic kinematics.

In the abstract view, the invariant interval is a **relational fact** about the tick algebra and the hinge geometry, not a separately imposed metric field.

**V2 view: typed budgets and per-act constraints**

In the engine, this structure is made concrete as **typed budgets** associated with each committed act:

* For each successful commit from (k) to (k+1), the engine assigns a triple  
  [  
  (\Delta \tau, \Delta t, \Delta x),  
  ]  
  where:
  + (\Delta \tau) is the **inner-tick budget**, counting how much “internal advance” this act represents along the relevant 0-context’s own line,
  + (\Delta t) is the **outer-tick budget**, counting how much “environmental time” this act advances along the outward index,
  + (\Delta x) is a measure of the **space-like displacement** associated with this act in the outward lattice (for example, how far the worldline moves in the spatial coordinates).
* The engine enforces a **typing constraint** that mirrors the invariance relation:
  + only triples that satisfy the appropriate discrete analogue of the invariant interval are even **allowed** as candidate budgets,
  + if a proposed act would violate that relation, it is treated as an invalid act and never reaches acceptance.
* The constant (c) appears as a **unit map** in the configuration:
  + it is declared in the manifest as the conversion between inner and outer time units, often derived from hinge and container considerations (for example, from the relation between a characteristic spatial scale and the temporal hinge),
  + it is not tuned per run or per scene; it is fixed in the configuration consistent with the abstract hinge identity.

By design, the engine cannot produce acts that deviate from the invariant-interval-type relation; those acts are simply ill-typed and therefore impossible.

**Crosswalk: invariant interval ↔ budget typing and composition**

The mapping between the abstract and engine pictures can be stated as:

* The **invariant interval** of the formal core corresponds to the **typing rule for budgets** in the engine:
  + In V1: “there is a relational invariant combining inner and outer separations” →
  + In V2: “only acts whose ((\Delta \tau, \Delta t, \Delta x)) satisfy this invariant are legal acts.”
* The **relativistic kinematics** (light cones, time dilation) correspond to the **composition of budgeted acts under the no-skip rule**:
  + In V1: composing ticks that obey the invariant interval yields:
    - a maximum effective speed (c) (nothing can move outside the light cone),
    - different proper-time experiences along different worldlines.
  + In V2: summing budget triples along a history:
    - produces effective worldlines in the space of ((t, x)) that cannot exceed a slope determined by (c),
    - yields different accumulated (\tau) for different histories that share the same outer ((\Delta t, \Delta x)), matching the notion of time dilation.

In both views:

* The constant (c) is **not a field**; it is a **conversion factor** fixed by the hinge and container choices.
* Geometry is not added on top of the dynamics; it is built into the way acts are defined and composed.

**Engine-level emergent SR**

Seen purely from the engine side, the story looks like this:

* You run the engine with the manifest that fixes:
  + the band structure,
  + the unit map (c),
  + the gate parameters,
  + and the budget typing rule.
* You then:
  + examine histories of committed acts,
  + plot their cumulative (\Delta x) versus (\Delta t),
  + and observe that:
    - no history goes outside the “cone” determined by (c),
    - histories that move faster in space have less accumulated (\tau) for the same (\Delta t), matching the relativistic pattern.

From the bridge perspective, this is exactly what the formal core predicted: once you enforce the invariant-interval-like typing and the no-skip rule, the **engine cannot help but exhibit special relativity** in its emergent behaviour.

**Practical reading of the mapping**

In practical terms:

* When you see the V1 theory speak about:
  + “deriving an invariant interval from the tick algebra,”
  + “light cones emerging from feasibility and composition,”
  + “time dilation as a consequence of relational structure,”

you can translate that, in engine language, as:

“the engine enforces a typing relation between (\Delta \tau), (\Delta t), (\Delta x) for each act, with a fixed unit map (c), and the composition of such acts under the no-skip rule generates the familiar relativistic behaviour.”

* Conversely, when the V2 documentation discusses:
  + the unit map (c) in the manifest,
  + the budget typing rule,
  + and the emergent cones and time dilation in simulations,

you can understand that as the **concrete realization of the invariant-interval structure** that V1 derives at the level of the abstract relational geometry.

This completes the special-relativity part of the emergent physics crosswalk: the invariant interval in the formal core and the typed budgets in the engine are two ways of stating the same constraint, one in abstract algebra and one in executable combinatorics.

**5.2 QM: Structural Born Rule ↔ PF/Born Ties-Only**

In the V1 formal core, **quantum behaviour** – superposition, interference, and the Born rule – is expressed in terms of **amplitudes on the Present Plane** and a **structural collapse** at the hinge. In the V2 engine, quantum-like probabilities appear only in one place: the **PF/Born ties-only step** that resolves exact structural ties between candidates. This subsection shows how those two pictures align.

**V1 view: Born rule from Present Plane structure**

In the abstract picture, the quantum side of the theory looks like this:

* At the hinge, a set of **potential next outcomes** is represented on the Present Plane (\mathcal{P}):
  + each outcome corresponds to a ray or vector in the state space associated with the present,
  + these vectors can interfere and form superpositions.
* A **complex structure** (J) on (\mathcal{P}) allows you to represent **amplitudes**:
  + components of the state are complex numbers,
  + relative phases between components matter for interference.
* The **Born rule** arises as a **structural statement**:
  + given a normalized state vector with components (a\_i), the probability of realizing outcome (i) is (|a\_i|^2),
  + this is not an extra axiom; it is tied to how you interpret squared norms in the Present Plane and to consistency requirements on how probabilities compose.
* **Collapse** is described as:
  + selecting a single outcome at the hinge,
  + resetting the state onto the corresponding ray in (\mathcal{P}),
  + updating the inner record accordingly.

The essential point is that, in V1, quantum probabilities are **not** arbitrary frequencies added by hand; they are **derived from the structure of amplitudes** and the way the Present Plane is interpreted.

**V2 view: PF/Born ties-only in the engine**

In the engine, the only place where probabilistic behaviour enters is the **tie-resolution step** in acceptance:

* After enumeration, hinge equality, and all gates have been applied, the engine has a set of **feasible candidates**:
  + typically, the ratio-lexicographic rule orders these candidates and selects a unique best one deterministically.
* However, in some ticks there remains a **true tie**:
  + multiple candidates have exactly the same residuals (outer, inner, cross metrics) under the lexicographic ordering,
  + they also satisfy all gates equally,
  + structurally, the engine cannot distinguish between them using any of its deterministic predicates.
* In that situation, the engine:
  + **Constructs an adjacency structure** on the tie set:
    - each tied candidate is a node,
    - edges represent a primitive notion of “phase-coherent connectivity” (how candidates relate under the engine’s adjacency rules, often derived from local structure).
  + **Forms a primitive Markov kernel** or adjacency matrix for this graph and computes its **Perron–Frobenius eigenvector** (v):
    - (v) has nonnegative components and describes the leading mode of this structure.
  + **Derives weights** by squaring the components: (w\_i \propto v\_i^2).
  + **Samples one candidate** from the tie set using these weights:
    - a random selection, but with probabilities determined entirely by the graph structure of the tie set.
* After this PF/Born step, the chosen candidate is then **committed** like any other: the world and qualia records are updated, budgets are assigned, and the other candidates are discarded.

This is the only probabilistic element in the engine. Everywhere else, behaviour is fully deterministic given the manifest, the gates, and the initial conditions.

**Crosswalk: amplitudes and Born ↔ eigenvector and PF/Born sampling**

The bridge between the two pictures can be described as:

* In V1, each potential outcome at the hinge has an **amplitude** (a\_i) on the Present Plane, and the Born rule says probability (\propto |a\_i|^2).
* In V2, each tied candidate in a tie set has a component (v\_i) in the **Perron–Frobenius eigenvector** of the tie adjacency:
  + you can view this eigenvector as the engine’s implicit **amplitude vector** for the tie set,
  + squaring its components, (w\_i \propto v\_i^2), is then the engine’s direct analogue of (|a\_i|^2).

Thus:

* **Amplitude assignment** in V1 ↔ **leading-eigenvector components** in V2.
* **Born probability** in V1 ↔ **squared eigenvector components used as sampling weights** in V2.
* **Collapse** in V1 ↔ **randomly selecting one candidate from the tie set using PF/Born weights, and then committing it** in V2.

Both pictures agree on the deeper structure:

* Probabilities arise **only** where there is **structural ambiguity** (ties at the hinge).
* The distribution of probabilities is determined **entirely by relational structure** (the geometry of the Present Plane in V1; the adjacency graph in V2), not by arbitrary noise.

**Why “ties-only” is important**

The “ties-only” rule in the engine is crucial for preserving the structural interpretation of the Born rule:

* It ensures that **probability is not used as a general decision tool**:
  + if the engine can distinguish candidates structurally (via gates, residuals, or acts), it does so deterministically,
  + randomness is reserved only for those cases where, from the engine’s own perspective, alternatives are genuinely indistinguishable.
* From the V1 side, this matches the idea that:
  + the Born rule applies **only** when the theory presents you with a true superposition at the hinge – multiple possibilities with equal structural status,
  + you do not invoke Born-like probabilities to resolve decisions that could have been determined by more structure or more information.

This alignment is what makes the PF/Born step a **faithful realization** of the structural Born rule in the formal core, rather than a generic stochastic add-on.

**Interference and phase in both views**

Interference between alternatives, in the abstract theory, is encoded in the **phases and amplitudes** on the Present Plane, and in how superpositions evolve. In the engine:

* Interference behaviour is reflected in:
  + how phase bins in Ξ\_phase group candidates,
  + how adjacency is defined on the tie graph (which candidates are considered “coherent” with which others),
  + how sequences of ticks redistribute support among candidates before the tie step.

When you design the tie adjacency and the phase-bin structure to reflect the same interference logic that the Present Plane carries in V1:

* the engine’s PF/Born step reproduces the **same qualitative interference patterns**,
* probabilities over outcomes depend on structural relationships among candidates, not just on counts.

The details of interference implementation belong to engine design and diagnostics, but at the bridge level the key message is:

* **Quantum structure** (superposition, interference, Born rule) in the formal theory  
  ↔
* **PF/Born ties-only selection** using eigenvectors on a phase-aware tie graph in the engine.

**Practical reading of the mapping**

In practice:

* When V1 talks about:
  + amplitudes on (\mathcal{P}),
  + collapse,
  + Born probabilities,

you can translate that into engine terms as:

“when multiple candidates survive every structural and feasibility test and are still exactly tied, build an adjacency graph on them, compute its Perron–Frobenius vector, square its components, and use that as the probability distribution for selecting the winner.”

* When V2 documentation describes:
  + the tie graph,
  + PF computation,
  + and the rule “randomness only at exact ties,”

you can read that as the concrete realization of **the Present Plane and structural Born rule** from the formal core.

This completes the quantum part of the emergent physics crosswalk: the formal amplitudes and Born rule in V1 correspond, one-to-one, to the engine’s PF/Born ties-only step acting on discrete tie sets and phase bins in Ξ.

**5.3 Gravity: Pivot Profile ↔ ParentGate**

In the V1 formal core, gravity is not introduced as a separate field living on top of spacetime. Instead, it appears as a consequence of the **pivot profile** (g(D(n))) across the context ladder and the way that profile shapes which relations are feasible in different regions and at different roles. In the V2 engine, this same idea is realized by the **ParentGate** (and related schedule configuration): a purely structural, boolean gate that thins out candidates according to radial shells and container roles, producing the same kinds of redshift, delay, and deflection that are usually attributed to a metric. This subsection explains how those pictures match.

**V1 view: gravity as pivot-driven feasibility geometry**

In the structural view:

* The context ladder and dimension profile (D(n)) are not just labels; they control how **feasible** certain relations are at different roles and locations.
* The **pivot function (g(D(n)))** describes how “strong” geometric effects are at each role:
  + near the hinge (D=2), (g) is neutral; geometry does not favour or suppress particular directions in a special way,
  + as you move outward or inward, deviations in (D(n)) and (g(D(n))) encode how containers and seams shape relational possibilities.

Gravity appears when you consider:

* **Outer containers** (like the Earth, galaxies, or cosmic shells) as particular bands on the ladder,
* and how the pivot profile behaves across the outer roles associated with those containers.

The key V1 idea is that:

* A **massive container** (e.g., an Earth-like or galaxy-like structure) is associated with a particular pivot profile in its region: relational possibilities are **more or less feasible** depending on radial position and context role.
* This **feasibility geometry** leads to:
  + **redshift-style effects**: clocks deeper in the container’s influence effectively accumulate record at different rates,
  + **delay and deflection**: paths that pass near a container must respect a different feasibility landscape than paths far away.

In other words, gravity is the **effect of the pivot profile on which relational paths are allowed**, not a force field pushing on objects.

**V2 view: ParentGate as structural feasibility gradient**

In the engine, this abstract idea becomes the **ParentGate**:

* ParentGate is a **boolean gate** that acts on candidates based on:
  + their **radial position** relative to a container center (e.g., shells around an Earth-like body),
  + their **band membership** (e.g., whether they belong to the Earth band, galactic band, etc.),
  + and sometimes additional discrete features in Ξ that encode container-related structure.

ParentGate is configured via the manifest as:

* A set of **radial shells** around one or more centers:
  + each shell corresponds to a range of radii in the underlying lattice,
  + shells are typically arranged so that they approximate isotropic containers (e.g. concentric rings or spheres).
* A **strictness schedule**:
  + each shell has an associated “strictness level” – for example, how many candidates are allowed to survive the gate in that shell, or what extra structural predicates must be satisfied there,
  + the schedule is **inward-monotone**: shells closer to the center have greater strictness (fewer candidates survive), shells further away are more permissive.
* A **rotation-invariant design**:
  + ParentGate is constructed to be **isotropic in expectation**: it treats directions around a center symmetrically,
  + any anisotropy must come from other structure, not from the gate itself.

Operationally, ParentGate:

* Declares some candidates **inadmissible** purely on the basis of where (and in which band) they are in relation to a container,
* Ensures that:
  + paths passing deeper into a container will be **thinned** more heavily,
  + paths staying far from containers will be **less affected** by this gate.

The engine never multiplies candidate scores by a continuous gravitational potential; it simply **removes** candidates according to these integer-based shell schedules.

**Crosswalk: pivot profile ↔ shell strictness schedule**

The crosswalk between V1 and V2 can be summarized like this:

* The **pivot profile (g(D(n)))** in the formal core corresponds to the **shell strictness schedule** in ParentGate:
  + abstractly, “geometry is stronger/weaker here”  
    ↔  
    concretely, “the gate is stricter/looser in this shell or at this role.”
* The **container roles** in the context ladder (e.g., Earth band, galactic band, cosmic band) correspond in V2 to:
  + specific **centers** and **sets of shells** in the manifest,
  + with ParentGate schedules that reflect the way (D(n)) and (g(D(n))) behave across those roles.

When the V1 formal core says:

* “Inside this radius, the effective geometry is different; clocks tick differently; paths bend and accumulate extra delay,”

the engine has a translation:

* “Inside this radius (identified by shell), ParentGate will be stricter; fewer candidate acts survive; the sequences of acts that do survive will show effective redshift, delay, and deflection when you look at their accumulated budgets and lattice paths.”

No explicit metric is needed. All “gravity” appears as **differences in allowed versus disallowed candidates** across shells and roles.

**Emergent gravitational effects in the engine**

This mapping becomes concrete when you look at the engine’s behaviour:

* **Redshift-style effects**:
  + signals emitted from deeper shells must pass through regions where ParentGate is stricter; fewer histories connect them to outer detectors,
  + the surviving histories, when interpreted via budgets or event rates, show an effective “slowing” or redshift relative to histories in less strict shells.
* **Shapiro-like delay**:
  + paths that pass near a strong container center (where strictness is high) may require more ticks (more acts) to reach a given outward position,
  + this shows up as **extra time-of-flight** compared to paths that stay in more permissive regions.
* **Deflection**:
  + the set of feasible candidate paths is biased away from certain directions and toward others near strong shells,
  + when you look at the aggregate behaviour of rays or particles, they appear to **bend** around the container as in weak-field lensing.

All of this is achieved using only:

* ParentGate’s shell strictness schedule,
* the other gates and budgets,
* and the standard engine pipeline.

No continuous gravitational potential or metric field appears in the control path. Gravity is, in engine terms, a **feasibility gradient**.

**Containers and re-centering**

In the formal core, you can re-center the context ladder on different systems: an organism, a planet, a galaxy. V2 respects this by:

* allowing **multiple ParentGate configurations**:
  + one for an Earth-like container,
  + another for a galactic disk container,
  + another for a cosmic shell, etc.
* letting you switch or compose these schedules depending on which container you are treating as the relevant +1 / +2 / +3 context for a particular run.

This preserves the **role-relativity** of the pivot picture:

* The same engine, with different manifests, can be “re-centered” on different containers,
* The shell strictness schedules then encode the corresponding pivot profiles for those containers without changing the core logic.

**Practical reading of the mapping**

In practical terms:

* When the V1 formal core discusses:
  + “gravity as pivot-driven feasibility geometry,”
  + “containers as surface-like roles with dimension 2 and specific (g(D)) behaviour,”
  + “redshift, delay, and deflection emerging from relational structure,”

you can read that, in engine language, as:

“ParentGate is configured with radial shells and inward-monotone strictness to encode container geometry; this gate thins candidate acts in a way that produces redshift, delay, and deflection in the surviving histories.”

* When the V2 documentation talks about:
  + “rotation-invariant, inward-monotone ParentGate schedules,”
  + “gravity-like scenes,”
  + “horizon-like regions where no commits occur,”

you can read that as the **concrete realization of the pivot profile** from the formal core. The patterns you measure in such runs (e.g., redshift ratios, timing delays, deflection angles) are the engine’s way of showing you (g(D(n))) in action, through purely combinatorial feasibility constraints.

**5.4 Fields and Gauge Structure via Features and Structural Gates**

Beyond relativity, quantum probabilities, and gravity, the V1 formal core also has room for **fields and gauge structure**: ways in which local symmetries and “charges” show up in the relational network. In the V2 engine, these are realized not as continuous fields but as **discrete tags in the feature alphabet** and as **structural gate rules** that control how those tags may change. This subsection explains that correspondence in broad strokes.

**V1 view: gauge structure and charges in the relational network**

In the abstract picture, you can think of gauge structure and charges like this:

* The underlying relational network is not just a bare graph; it supports **local symmetries**:
  + you can change certain internal labels or “phases” on carriers and relations without changing the physically relevant content,
  + equivalence under these transformations defines a **gauge symmetry**.
* **Gauge fields** appear as:
  + additional structure attached to links or nodes (connection-like data),
  + rules for how those attachments transform when you move along the network.
* **Charges** appear as:
  + patterns of inclusion or exclusion relative to certain contexts or symmetries,
  + constraints on how carriers can be connected or how operators may act when those patterns are present.

In the formal core, you often talk about these things in terms of:

* abstract **link variables**,
* transformation rules,
* and invariance of the action or master relation under certain local group actions.

The specific details depend on which part of the full theory you are focusing on (electromagnetism, other forces, etc.), but the common theme is that gauge structure is **encoded in relational tags and symmetry rules**, not in an independent, substance-like field.

**V2 view: discrete features and structural gate rules**

In the engine, these ideas are implemented using:

* **Discrete tags in the feature alphabet Ξ**, and
* **Structural gates** that specify how those tags may appear and change.

Concretely:

* The feature alphabet Ξ includes:
  + **band and role tags** (e.g., which context band a candidate belongs to),
  + **container tags** (e.g., which ParentGate or schedule a candidate is subject to),
  + **“charge-like” tags** (e.g., inclusion patterns such as “inwards/outwards inclusion through a given context,” which can play the role of electric charge or other quantum numbers),
  + **directional and topological tags** (e.g., lane, orientation, cycle labels) that encode discrete symmetry-related structure.
* Structural gates then enforce rules like:
  + “a candidate with tag A cannot simultaneously have tag B,”
  + “along this path, the orientation or lane tag must change in a specific pattern,”
  + “charge-like tags must be conserved across a local operation,”
  + “certain configuration changes are only allowed if tags transform under a prescribed rule.”

In this way, what the formal core describes as a gauge symmetry becomes, in the engine:

* a **set of equivalence relations** on feature tags, and
* a **set of allowed and forbidden transitions** encoded as structural gate rules.

**Crosswalk: gauge fields ↔ tag structure, charges ↔ inclusion patterns**

The bridge can phrase the mapping as:

* **Gauge fields in V1** ↔ **structured patterns of tags and adjacency in Ξ and in the candidate graph**:
  + the abstract link variables that transform under a gauge group correspond to discrete tags on edges and nodes in the engine’s internal representation,
  + the way those tags are allowed to change under the engine’s rules is the discrete version of the gauge transformation laws.
* **Charges in V1** ↔ **patterns of inclusion and exclusion encoded in tags and gates**:
  + for example, “charge” can be represented as:
    - whether a candidate’s state includes a particular kind of inward/outward context inclusion (such as a particular 0↔+1 inclusion pattern),
    - how that pattern is allowed to flow through the network over ticks.
  + structural gates enforce conservation or transformation rules for these patterns, mirroring the formal statements about charge conservation and gauge invariance.

In practice, you can define:

* a particular **charge tag** in Ξ,
* rules in the structural gates for how that tag may appear, transform, and combine,
* and interpret those rules as the engine-side realization of a gauge symmetry and conserved quantity in the abstract theory.

**Electromagnetism and other specific structures**

For concrete examples like electromagnetism:

* In the formal core, you might describe:
  + an **electric charge** as a particular relational inclusion between inner and outer contexts,
  + an **electromagnetic field** as a pattern of relational biases or curvature in configurations around those charges,
  + and gauge invariance as the freedom to shift potentials without changing physical content.
* In the engine, this becomes:
  + a **charge-like tag** associated with certain candidates (for example, “inclusion sign” or similar),
  + **E-like and B-like diagnostics** built from count patterns and loop circulation around those tagged candidates,
  + structural gate rules that prevent certain impossible configurations (for instance, that would violate charge conservation or create inconsistent E/B patterns around a candidate).

Other gauge-like structures (if introduced) can be encoded similarly:

* add tags to Ξ for the relevant quantum numbers,
* define structural gates that allow only those transitions that respect the intended symmetry,
* use diagnostics to reconstruct the effective field behaviour.

The key is that **nothing in the engine control path uses continuous gauge fields directly**; it only uses discrete tags and predicate rules.

**Role of diagnosis versus control**

Just as with geometry, gauge-like behaviour in the engine is split between:

* **Control**, which uses:
  + tags and gates to enforce structural constraints and symmetry rules,
  + but never continuous weights or potentials.
* **Diagnostics**, which can:
  + reconstruct effective field behaviour (e.g., E/B-like patterns, charge distributions) from statistics of acts and tag configurations,
  + fit discrete data to familiar continuous field models for interpretation and comparison.

From the bridge perspective, this means:

* The formal gauge fields are expressed, in the engine, as **rules on tags and allowed transitions**,
* The usual continuous field equations are something you can **infer in diagnostics** (as an approximate description of emergent behaviour), rather than embed directly in control.

**Summary at this level**

At the level of the crosswalk:

* **Local symmetries and gauge fields** in the abstract theory are implemented as **tag structures and structural gate rules** in the engine.
* **Charges** are realized as **patterns of inclusion or discrete quantum numbers** in Ξ, with conservation or transformation rules enforced by the gates.

This ensures that V2 can reproduce the **symmetry and field content** of V1’s relational physics **without introducing continuous gauge fields into the control path**, staying faithful to the combinatorial and curve-ban constraints while still reflecting the same underlying structures.

**6. Dynamics & Master Action as Algorithmic Preference**

**6.1 V1 Master Action and Path-Sum Picture**

In the V1 formal core, the dynamics of the theory are not given by differential equations on a background spacetime, but by a **master action** defined over the relational network of ticks. This action assigns “costs” or “weights” to sequences of present-acts and encodes which histories are preferred, allowed, or effectively suppressed. Before we map this to the engine, it is useful to restate the V1 picture on its own terms.

**The master action over tick histories**

At a high level, the V1 construction does the following:

* It considers **histories** as sequences of carriers:  
  [  
  \mathcal{H} = (\mathcal{C}\_0, \mathcal{C}\_1, \mathcal{C}\_2, \dots),  
  ]  
  where each (\mathcal{C}\_k) is a carrier at tick (k), with its IN/ON split, state (h\_k), and ledger values.
* For each elementary step (\mathcal{C}*k \to \mathcal{C}*{k+1}), the tick-operator algebra (Renew, Trade, Sync, Sink, framing) can be associated with a **local contribution** to an action:
  + these contributions reflect:
    - how much the step “stretches” inner or outer separation (relativity side),
    - how much it deviates from pivot conditions or dimension profiles (D(n)) (geometry side),
    - how well it respects gauge-like symmetries and admissibility predicates (field side).
* The **master action** (S[\mathcal{H}]) for a history (\mathcal{H}) is then built as a sum (or more generally, an aggregate) of these local contributions over ticks:  
  [  
  S[\mathcal{H}] = \sum\_k S\_k(\mathcal{C}*k \to \mathcal{C}*{k+1}),  
  ]  
  where each (S\_k) is a function of the operators applied, the context roles involved, and the hinge/ledger state.

The details of how each term is defined depend on the part of the theory you are looking at (e.g., purely relativistic dynamics, inclusion of gravity, gauge structure), but the conceptual picture is simple: **each tick contributes to a global “score” or “cost” for a history.**

**Extremal histories and action principles**

Once the master action is defined, the formal core uses a familiar idea from physics: **extremal action principles**:

* Among many possible histories consistent with the admissibility predicates, the ones that matter physically are:
  + those that **extremize** the action (e.g., least action, stationary action), or
  + those that are **weighted more heavily** in a path-sum (for quantum-like descriptions).
* In a purely classical or “most probable” sense:
  + the theory singles out histories that minimize or stationarize (S[\mathcal{H}]),
  + these histories obey discrete analogues of geodesic or field equations in the relational network.
* In a quantum-like sense:
  + you can imagine a **path-sum representation**, where a state is built from contributions of many histories, each weighted by an amplitude related to (\exp(i S[\mathcal{H}] / \hbar)) or a similar function,
  + interference between these contributions leads to suppression of some histories and enhancement of others, consistent with the structural Born rule.

The key point is that the master action provides a **unified criterion** for which histories are dynamically preferred, without introducing an external “force law” on top of the relational structure.

**Relation to the operator algebra and admissibility**

The master action is not independent of the operator algebra and admissibility predicates; it is built from them:

* **Admissibility predicates** determine the **domain** of the action:
  + histories that violate admissibility at any tick are simply not allowed; their action is either undefined or effectively infinite,
  + the action only has meaning on histories constructed from admissible transitions.
* **Operators and their composition** determine the **form** of the action contributions:
  + Renew and Trade affect how much inner and outer structure changes at a tick, influencing terms related to interval and geometry,
  + Sync and framing influence how various carriers are aligned with containers and gauge structure, influencing terms related to curvature and fields,
  + Sink is where actualization happens; it can induce contributions related to collapse, information gain, or decoherence-like effects.

In other words, the master action **summarizes the cumulative effect** of the operator algebra and admissibility rules along a history. It is not a separate layer of physics, but rather a **compact encoding** of the dynamics already implicit in the relational structure.

**Path-sum picture and quantization**

When you want to talk about **quantum behaviour** at the level of histories, the master action also provides the natural foundation:

* Each admissible history (\mathcal{H}) is assigned an **amplitude** related to its action:
  + for example, something like (A[\mathcal{H}] \propto \exp(i S[\mathcal{H}] / \hbar)) in a continuum-inspired analogy,
  + the exact form can be adapted to the discrete relational context, but the core idea is that histories with different actions acquire different complex phases.
* The **path-sum** (or sum over histories) then combines these amplitudes:
  + when computing an effective amplitude for a coarse-grained outcome, you sum over amplitudes for all fine-grained histories that lead to that outcome,
  + interference between these contributions, governed by their relative phases, yields patterns consistent with the Born rule and the Present Plane description at the hinge.

This connects:

* local **tick-level structure** (operator algebra and admissibility) to
* global **probabilistic behaviour** (quantum interference and collapse), via the master action and path-sum.

**Why the master action matters for the engine**

Although the engine does not literally compute (S[\mathcal{H}]) or sum over all histories at runtime, the master action is important for two reasons when you bridge to V2:

1. It tells you **what kind of histories you want the engine’s deterministic preferences to approximate**:
   * the engine should behave as if, locally, it is favouring steps that align with extremal action paths and suppressing steps that would lie far from those paths.
2. It tells you **how to interpret PF/Born sampling and emergent statistics**:
   * the distribution over realized histories generated by the engine should match, in the appropriate limit, the distribution you would obtain by weighting histories with their action-derived amplitudes and applying the structural Born rule.

The next subsection will describe how the engine pipeline can be viewed as a **local, greedy implementation of “action minimization/preference”**: the gates and ratio-lex acceptance effectively steer the system toward low-action histories, while PF/Born sampling over ties ensures the correct probabilistic weighting when multiple histories are equally favoured by the structural criteria.

**6.2 V2 Cycle as Realization of Action Minimization**

The V1 master action and path-sum picture say, in effect, “some histories are dynamically preferred over others; the preferred ones are those that extremize a certain relational action.” The V2 engine does not literally compute a global action or scan all histories, but its **local decision rules** are designed so that, tick by tick, it behaves as if it were following an **action-minimizing (or extremizing) preference**. This subsection explains how the engine cycle realizes that idea.

**Local preference instead of explicit S**

The engine never carries around an explicit function (S[\mathcal{H}]). Instead, its pipeline implements the **same preferences locally** via:

* **gates**, which enforce admissibility and coarse geometric constraints, and
* **ratio-lex acceptance**, which orders feasible candidates according to simple, local “cost” measures.

You can think of the engine as implementing a **greedy algorithm** for history construction:

* at each tick, it looks at a finite set of admissible local moves and chooses the one that is, in a well-defined sense, “best” according to structural criteria,
* over many ticks, this local “best choice” rule guides the system toward histories that line up with the extremal paths of the master action.

The key is that the structural criteria used by the engine – stability, coherence, minimal deviation from certain norms – are directly grounded in the same relational constraints that generated the master action in the formal core.

**Residuals as local action surrogates**

In the acceptance stage, each feasible candidate at a tick is assigned a **residual vector**:

* components of this vector may include:
  + outward residuals (how much the candidate deviates from ideal outward constraints),
  + inward residuals (how much it deviates from inward consistency or record-coherence),
  + cross residuals (how much mismatch remains at the hinge or between bands).

These residuals are:

* entirely **integer or ratio-valued**,
* constructed from gate-level information (which predicates passed, which barely passed, which were at thresholds),
* ordered in a **fixed priority scheme** (e.g., outward residual first, then inward, then cross).

From a bridge perspective, these residuals are **surrogates for local action contributions**:

* a candidate with smaller residuals is one that:
  + better respects the invariant interval typing,
  + better aligns with the pivot structure and dimension profile as encoded in the manifest,
  + better obeys structural and gauge-like constraints.

Choosing a candidate with **minimal residuals** is the engine’s way of approximating “choose the step with minimal local action contribution.”

**Ratio-lex acceptance as discrete extremal principle**

The ratio-lex ordering rule can be understood as a discrete extremal principle:

* First, candidates are **filtered**:
  + only those that satisfy all admissibility gates (the engine’s implementation of formal predicates) are allowed in.
* Then, among those admissible candidates:
  + the engine picks the one with the **lowest outward residual**; if tied, the one with the lowest inward residual; if still tied, the one with the lowest cross residual; if still tied, the one with **fewest acts**, or other structural tie-breakers.

This procedure has several important properties:

1. It is **deterministic** as far as structure allows:
   * as long as the residual vector distinguishes candidates, there is a unique “best” candidate.
2. It is **order-consistent** with the formal core’s priorities:
   * outward constraints (e.g., invariant interval, container geometry) get resolved first,
   * inward and cross constraints follow, reflecting the way the master action weighs different contributions.
3. It is **local**:
   * the engine does not need to know future consequences; it only needs local data at this tick and what has been carried in the record.

Together, these make ratio-lex acceptance a **discrete implementation of an extremal principle**: out of a locally admissible set, pick the one that best minimizes a structured “cost” vector.

**Gates as hard constraints, residuals as soft preferences**

Viewed through the lens of the master action:

* **Gates** correspond to **hard constraints**:
  + they enforce that a candidate has **finite or acceptable action**; those that would effectively correspond to “infinite cost” or forbidden configurations are removed outright.
* **Residuals and ratio-lex ordering** correspond to **soft preferences**:
  + among the candidates that satisfy hard constraints, residuals encode which ones are closer to the “ideal” action pattern,
  + choosing the candidate with the smallest residuals is akin to picking the one with minimal or stationary local action.

So the engine realizes a two-tier action logic:

1. **Admissibility gates** ensure “no illegal moves” – only histories within the allowed relational class are built at all.
2. **Residual-based acceptance** expresses a **preference for minimal or extremal local action** within that allowed class.

**Consistency over many ticks**

The single-tick extremal choice must, over many ticks, line up with the global action structure. Several design decisions ensure this:

* **No-skip rule**:
  + ensures that histories are built out of **chains** of these local choices, with no jumps that would break the action composition picture.
* **Monotone record and typed budgets**:
  + ensure that the consequences of earlier choices are carried forward, influencing residuals at later ticks in a way consistent with accumulating action contributions.
* **Manifest and band structure**:
  + guarantee that local choices at different bands are informed by the same global ladder (D(n)) and pivot structure that defined the master action in V1.

This means that, as you compose many engine cycles, you are effectively composing local action-like decisions into a global history that approximates an extremal path for the master action.

**How this informs engine design**

From the bridge point of view, the master action is not something the engine has to compute explicitly; **it is the design guide** for:

* what residuals to define,
* how to order them,
* how to steer gates and manifests.

Whenever you refine the action or add new terms in the formal core, you ask:

* “What local constraint or preference does this new term correspond to?”
* “Which gate or residual component needs to be updated so that the engine continues to favour the right histories?”

In this way, the master action and path-sum are **conceptual tools** that keep the engine pointed in the right direction: every engine cycle is a discrete, local realization of “prefer lower action” in the space of allowed transitions, ensuring that the histories the engine actually produces are the same ones the formal theory singles out as dynamically natural.

**6.3 Quantization as Engine Sampling**

In the V1 formal core, once you have a master action, **quantization** can be framed as a **path-sum**: amplitudes are assigned to entire histories, and interference between those histories yields the probabilities you see at a coarse-grained level. In the V2 engine, you do not explicitly sum over all histories. Instead, you run the engine many times with the same configuration and observe the **distribution of realized histories**. This subsection explains how those two pictures correspond: quantization in V1 is realized as **engine sampling** in V2.

**V1 view: path-sum and amplitudes over histories**

From the formal perspective:

* Each **admissible history** (\mathcal{H}) – a sequence of tick-to-tick transitions – is assigned a **complex amplitude** (A[\mathcal{H}]) derived from the master action:
  + typically, (A[\mathcal{H}]) is some function of (S[\mathcal{H}]), such as (\exp(i S[\mathcal{H}]/\hbar)) in a continuum-inspired analogy,
  + the details of the function matter less than the fact that different histories acquire different phases and magnitudes based on their action.
* To compute the amplitude for a **coarse-grained outcome** (e.g., “the system is found in configuration X at time T”), you conceptually:
  + sum over all admissible histories (\mathcal{H}) that lead to that outcome,
  + combine their amplitudes with signs and phases,
  + let **interference** between them enhance some outcomes and suppress others.
* Probabilities for outcomes are then given by:
  + applying the **Born rule** to these coarse-grained amplitudes,
  + i.e., taking squared norms after summing over histories.

The result is a **distribution over outcomes** that reflects both:

* the structural constraints encoded in admissibility and the master action, and
* the quantum nature of superposition and interference encoded in the amplitudes and their phases.

**V2 view: distributions from repeated engine runs**

In the engine, there is no explicit summation over all histories. Instead:

* You fix a **manifest**:
  + band structure,
  + gates and thresholds,
  + ParentGate schedules,
  + budget typing,
  + phase-bin structure and adjacency rules for PF/Born.
* You choose **initial conditions**:
  + initial world and qualia records,
  + any explicit context labels or charge-like tags.
* You then **run the engine many times** with:
  + the same manifest and initial conditions,
  + but different realizations of randomness in the PF/Born ties-only step (i.e., different random seeds).

Each run produces:

* a single **realized history**, a sequence of committed states ((W\_k, Q\_k)) and associated budgets,
* one **actualized path** through the space of possibilities.

By collecting many runs and examining:

* the frequency of different coarse-grained outcomes (e.g., measurement results, path shapes),
* the frequencies of different types of histories,

you obtain an **empirical distribution** over outcomes and histories.

From the bridge standpoint, this empirical distribution is the engine’s way of **sampling from the path-sum ensemble** that the formal core describes.

**How PF/Born ties the sampling to the action structure**

The crucial link between the engine’s sampling and the formal path-sum is the **PF/Born ties-only rule**:

* At each tick where a tie occurs, the engine uses **Perron–Frobenius eigenvectors** of the tie adjacency (and their squared components) to assign probabilities to the tied candidates.
* These probabilities are entirely determined by:
  + the **structural context** (which candidates are connected to which others in the tie graph),
  + the **phase-bin structure** in Ξ\_phase,
  + and ultimately, the relational constraints that gave rise to those adjacency rules and phase assignments in the first place.

Because of this:

* The choices the engine makes at each tie are **not arbitrary**; they reflect the same relational structure that, in the formal core, informs the master action and Present Plane amplitudes.
* Over many ticks and many runs, these local PF/Born choices accumulate into a distribution over histories that is **consistent with the path-sum weights** implied by the action.

In this sense:

* The engine is a **stochastic dynamical system** whose randomness is carefully constrained so that its long-run statistics match the distribution defined by the formal action-and-amplitude picture.

**From local choices to global distributions**

The engine builds histories one tick at a time:

* At ticks with no ties, the evolution is fully deterministic given configuration and past history – these acts correspond to history segments that are strongly preferred by the action structure (unique minimal residual).
* At ticks with ties, PF/Born introduces **branch points**:
  + different runs will take different branches at those points,
  + the relative frequencies of those branches across many runs match the PF/Born probabilities.

Over many runs:

* Each complete history (\mathcal{H}) accumulates a **realization probability** (P\_{\text{engine}}[\mathcal{H}]) determined by:
  + the product of PF/Born probabilities at each tie along that history,
  + combined with the fact that non-admissible histories have zero probability (they are never realized).
* For coarse-grained questions (like measurement outcomes), you can:
  + count frequencies of those outcomes,
  + interpret them as sums over the probabilities of underlying histories that realize them.

From the bridge perspective, this is the engine’s operational translation of:

* summing over histories with action-derived amplitudes,
* and then applying the Born rule at the coarse-grained level.

The difference is **procedural**:

* V1: think in terms of a formal sum over all histories, weighted by amplitudes.
* V2: let the engine’s dynamics and PF/Born ties-only rule **sample histories** directly, and read off frequencies.

But the underlying structure determining which histories are more or less likely is the same.

**Why this counts as “quantization” in the engine sense**

Calling this “quantization as engine sampling” emphasizes:

* You do not bolt quantum noise onto a classical engine; you build an engine whose **stochastic structure is already dictated by the relational and action principles**.
* The engine’s random steps:
  + occur only in structurally justified contexts (ties at the hinge),
  + use a probability assignment derived from adjacency and phase, which reflects the same relational geometry that the master action and Present Plane capture.

In the appropriate limit:

* the **ensemble of histories** generated by the engine reproduces the **same distribution of outcomes** that the formal path-sum and Born rule assign,
* the engine thus acts as a **Monte Carlo-like realization** of the quantized dynamics implied by the master action.

**Practical reading of the mapping**

In practical terms, when you are thinking in V1 language:

* “Sum over all admissible histories, weight them by (\exp(i S/\hbar)), then apply Born,”

you can translate that to engine language as:

“Run the V2 engine many times with the same manifest and initial conditions, using PF/Born ties-only at each structural tie. The empirical frequencies of outcomes across runs will approximate the path-sum/Born probabilities defined by the master action.”

Conversely, when you see in V2 discussions about:

* repeated runs,
* statistics of outcomes,
* agreement with quantum-like predictions,

you can understand that as:

“quantization in this framework is the **distribution over histories** that the engine samples when it follows its PF/Born-informed, action-respecting dynamics.”

This completes the dynamics-and-quantization part of the bridge: the master action and path-sum picture in the formal core correspond directly to the V2 engine’s local preference rules and its sampling over histories, with PF/Born ties-only providing the structurally grounded link between the two.

**7. L-Roles, Ladder, and Phenomenology (“Dynamics of Becoming”)**

**7.1 L-Roles in V2 & Their V1 Meaning**

The L-roles (L1, L2, L3) were introduced in the V2 ontology as a way to talk about what a present is *doing* during a single act: gathering usable structure from the past, standing among possible futures, and unifying into one realized outcome. In the V1 formal core, these roles were already there implicitly in the way the carrier and operators behave, but they were not given short names. This subsection makes the connection explicit.

**L-roles in V2: what the present is doing**

In the V2 ontology, the L-roles are process labels attached to a single tick:

* **L1 – Past-units and candidate futures**
  + At L1, the present is:
    - looking inward at **discrete units of structure** that have already been formed and retained (past pixels in the qualia record), and
    - looking outward at a **cloud of candidate futures** that can be assembled from those units and the current environment.
  + In engine terms:
    - enumeration of candidates + minimal gating produce the L1 set of **co-feasible candidates** for the next act,
    - these candidates are built from the current (Q\_k) and (W\_k), and from whatever patterns have survived κ and structural gates so far.
* **L2 – Outward experiences / environment of those candidates**
  + At L2, the same set of candidates is now being read from **one step out**:
    - each candidate is treated as a **full experience of time** from the vantage of a higher context (for instance, from +1 looking at 0),
    - the question becomes: “which of these candidate futures can co-exist as one coherent environment at the next context level?”
  + In engine terms:
    - L2 is realized by **coherence tests** over candidate families (for example, using context graphs or L2-coherence diagnostics),
    - it is where you determine which combinations of L1 candidates can be treated as a single outer world in the sense of the container.
* **L3 – Unifier / selector**
  + At L3, the present takes on the job of **choosing one realized outcome**:
    - structurally, this is the move from “many possible futures” to “the one that actually happened,”
    - phenomenologically, it is what turns potential into the next definite now.
  + In engine terms:
    - L3 is the ratio-lex acceptance and, if necessary, PF/Born ties-only step,
    - followed by commit, which writes the chosen future into the world and qualia records as the new present.

Together, L1, L2, and L3 describe the **full dynamics of becoming** at a single tick: from available past-units and candidate futures, through environmental coherence, to one realized act.

**How these roles appear in V1**

In the V1 formal core, you can see the same roles in the carrier and operator structure:

* **L1 in V1: Renew and IN/ON as sources of possibilities**
  + The **IN** side of the carrier holds discrete, nested units of past structure – exactly what L1 calls “past pixels.”
  + The **ON** side, under **Renew**, exposes a cloud of possible outward relations – exactly what L1 calls “candidate futures.”
  + This is where the formal core talks about:
    - “the collection of IN-units available to be used,” and
    - “the renewed ON configurations that could be realized next.”
* **L2 in V1: outer carriers and the collective sphere**
  + From one step further out (e.g. at +1 looking at 0), each potential 0-level future is seen as a **full carrier** in its own right.
  + The **Sync** operator and the **collective sphere** construction describe exactly how:
    - different 0-level candidates can or cannot be assembled into a single +1-level environment,
    - which combinations of 0-level worldlines are compatible as one coherent outer context.
  + This is L2’s “environment of candidates” role in formal language.
* **L3 in V1: Sink and framing**
  + The **Sink** operator and framing operations decide:
    - which outward possibilities are actually realized,
    - how they are incorporated into IN as the next carrier’s record.
  + The formal collapse, interpreted via the Present Plane and Born rule, is exactly L3’s “unifier/selector” job:
    - structurally, the algebra must produce one coherent carrier per tick,
    - probabilistically, when there are multiple structurally identical options, the Born structure decides which one becomes actual.

So, although V1 does not use L1/L2/L3 labels, the algebra already **distinguishes these roles by what each operator and construction is doing**.

**Why naming L-roles helps**

The bridge benefits from the L-role naming for several reasons:

* It gives a **shared vocabulary** for dynamic roles across V1 and V2:
  + You can say “L1 behaviour at the −2 band” or “L3 statistics at +1” instead of repeatedly describing “the phase where past-units and candidate futures are in play” or “the unifying collapse phase.”
* It makes it easier to organize **phenomenology and diagnostics**:
  + For example, you can talk about:
    - “L1 multiplicity” (how many candidates exist at a tick),
    - “L2 coherence” (how many candidates can form one environment),
    - “L3 tie-rate” (how often probabilistic selection is needed).
  + These quantities correspond directly to structural features in V1 (size of Renewed ON set, coherence of outer carriers, presence of Present Plane superpositions).
* It clarifies how **different context bands** behave dynamically:
  + At some bands (e.g., near the quantum-like −2 seam), you may see high L1 multiplicity and high L3 tie-rates – many possibilities, frequent structural ties.
  + At others (e.g., macro-like +1 band), you may see low L1 multiplicity and almost no L3 ties – few possibilities, mostly deterministic evolution.

From the formal core’s point of view, you can think of the L-roles as simply **naming different slices of the operator algebra’s work** in a way that is friendly to the ontology and engine, while still being anchored in the V1 structure of carriers, operators, and collapse.

**7.2 Upward vs Downward View (Generic-down, Specific-up)**

The context ladder is not just a stack of “levels”; it is a **relational hierarchy around a chosen present**. How things look depends on whether you are looking **downward** (toward inner roles −1, −2, …) or **upward** (toward outer roles +1, +2, +3, …). The V2 ontology talks about this as “generic-down, specific-up”: inner contexts look more generic and plexity-like, outer contexts look more specific and world-like. This subsection connects that intuition to the V1 IN/ON structure and the L-roles.

**Downward view: inner roles as generic plexity**

When you look **downward** from a present (from 0 toward −1, −2, …):

* In V2 language:
  + you are looking at **what the present contains**,
  + deeper roles (−1, −2) present as **dense, highly relational “texture”** rather than as discrete, named objects,
  + the further inward you go, the less things look like familiar “things” and the more they look like **plexity** – intertwined patterns and processes.
* In V1 language:
  + you are seeing ever more of the **IN side** of the carrier’s network,
  + the inner roles correspond to increasingly **fine-grained nested structure** that is already part of the record,
  + from the vantage of 0, these inner roles are typically **indistinguishable in detail**: you know that the carrier’s inward depth is rich, but you do not track each individual inner relation as a separate “object.”

From the L-role perspective:

* L1, looking inward, sees **past units** at various scales:
  + at −1, these may look like “cells” or micro-parts,
  + at −2, they look like “molecular networks” or other fine-grained plexity.
* But from the 0 vantage, the inward side is mostly about **what is usable as parts**, not about naming each inner relational thread separately.

This is why the downward view has the flavour of “generic”:

* you see **aggregated capacity** and **nested depth**,
* you treat inner structures more as **resources** (things you can draw on) than as fully separate entities in their own right at your level.

**Upward view: outer roles as specific environment**

When you look **upward** from a present (from 0 toward +1, +2, +3, …):

* In V2 language:
  + you are looking at **the containers that include you**,
  + outer roles (+1, +2, +3) present as **specific environments**:
    - Earth-surface as a shared world with concrete locations and objects,
    - galactic disk as a defined astronomical context,
    - cosmic shell as the sky and horizon.
* In V1 language:
  + you are seeing more of the **ON side** of the carrier’s network,
  + the outer roles correspond to the **contexts that coordinate many carriers** at your level,
  + these contexts are experienced as **particular and structured**:
    - there is “this planet,” “this galaxy,” “this sky,” rather than a generic “outer network.”

From the L-role perspective:

* L2 sees **candidate futures as full experiences** in these outer contexts:
  + different ways your present could unfold are tied to **different configurations of the environment**,
  + the environment at +1 or +2 appears as a specific arrangement of places, objects, and large-scale structures.
* L3 then chooses one such configuration as the realized environment for the next tick.

This is why the upward view has the flavour of “specific”:

* you see **definite locations**, **definite containers**, and **definite configurations** that many 0-level presents share as a common world,
* outer roles are what make the world feel **objective and named** (“Earth,” “Milky Way,” “cosmic background”), rather than a generic outside.

**Generic-down, specific-up as IN/ON asymmetry**

The “generic-down, specific-up” pattern is just the IN/ON asymmetry viewed through the ladder:

* **Downward (IN)**:
  + dominated by **plexity** and **many-to-one condensation**,
  + internal structure is rich but mostly anonymous from the 0 vantage,
  + we experience it as “complexity inside” rather than as discrete external entities.
* **Upward (ON)**:
  + dominated by **coordination** and **one-to-many branching**,
  + containers are relatively few and easily named,
  + we experience them as “the specific world we’re in.”

The L-roles sit on top of this:

* L1 uses the **generic inner plexity** as a pool of possible building blocks.
* L2 expresses how those blocks appear in a **specific outer context**.
* L3 selects one specific outer configuration while quietly updating the inner depth.

**Implications for phenomenology and diagnostics**

Understanding this asymmetry is useful when interpreting both:

* **Phenomenology** (how the world appears):
  + inner complexity appears as “me” – my biology, my mind, my inner processes – rich but not fully objectified,
  + outer containers appear as “world” – the environment, planet, cosmos – distinct and seemingly fixed.
* **Diagnostics** (how the engine behaves at different bands):
  + at inner bands (−2, −1), you expect:
    - high **L1 multiplicity** and deep plexity,
    - but relatively coarse, generic outer descriptions from the 0 vantage.
  + at outer bands (+1, +2, +3), you expect:
    - more constrained L1 multiplicity at the 0-level,
    - but highly **specific, structured environments** that show up clearly in L2 and L3 statistics.

From the bridge perspective, “generic-down, specific-up” is simply the **operational expression** of the formal statement that:

* IN is where many inner relations are collapsed into one present,
* ON is where one present is embedded in a few large, shared contexts.

The L-roles, the ladder, and the engine’s gates all respect this structure; they just make it **explicit and usable** when you design simulations, interpret results, or explain the phenomenology of time and space in the language of the theory.

**7.3 Nested-Time Matter & Stability in Engine**

In the formal core, “matter” is not a separate substance; it is **stable nested-time structure** – patterns that persist across ticks and across context roles, behaving as coherent units relative to a chosen present. In the engine, the same idea appears as **stable motifs** that repeatedly survive κ and structural gates, maintain their identity across many acts, and behave like parts in the outward read. This subsection makes that connection explicit.

**V1 view: matter as stable nested-time patterns**

In the V1 picture, you can think of “nested-time matter” like this:

* A piece of matter is a **pattern in the relational network** that:
  + persists over many ticks,
  + maintains a recognizable identity when seen from the hinge,
  + has **inner nested time** (its own internal dynamics) that is consistent with the outer time of the context.
* This pattern spans **multiple roles on the context ladder**:
  + inward roles encode how the matter’s internal structure behaves (e.g., molecular, cellular, or sub-cellular dynamics),
  + outward roles encode how the matter fits into larger containers (e.g., as part of an organism, a planet, etc.).
* From the formal standpoint, matter is characterized by:
  + **stability under the operator algebra**:
    - repeated application of Renew, Trade, Sync, Sink, framing still yields a recognizably “the same” entity,
  + **compatibility with the pivot and dimension profiles**:
    - the pattern lives in the right bands for its scale and role,
    - it respects admissibility predicates across inner and outer roles.

In short, “matter” in V1 is **a self-consistent cluster of relations that survives the tick dynamics**, not a primitive, irreducible thing.

**V2 view: stable engine motifs as matter-like units**

In the engine, this idea is realized as **patterns in (W\_k) and (Q\_k)** that:

* Survive the **κ gate and structural gates** across many ticks:
  + they repeatedly pass granularity tests – they are large and coherent enough, at the relevant band, to be treated as parts rather than texture,
  + they repeatedly pass contiguity, degree, and orientation checks – they remain connected and well-formed.
* Persist across many acts without losing their identity:
  + the same configuration (or a close variant) appears in successive (W\_k) and (Q\_k),
  + the engine’s view of this pattern as a “part” is stable: it continues to be presented outward as a unit and inward as a coherent substructure.
* Show up as **L1 building blocks**:
  + at L1, these motifs are the “past-units” that can be used again and again to construct different futures,
  + the fact that they can be reused reliably is part of what makes them matter-like.
* Are **tracked in features**:
  + they have consistent tags in Ξ (band labels, pattern types, roles),
  + their tags and relations evolve in a controlled way, following rules encoded in structural gates and context graphs.

From the engine perspective, when a pattern keeps appearing as a **coherent, admissible, and reusable unit** across ticks, you treat it as a “piece of matter” at that band and vantage.

**Crosswalk: nested-time matter ↔ stable, reusable engine patterns**

The mapping between the two views is then:

* **Nested-time matter in V1** ↔ **stable patterns in the engine that:**
  + repeatedly pass κ and structural gates at their band,
  + maintain their identity as parts in (W\_k) and (Q\_k),
  + participate consistently in the operator algebra (enumeration, hinge equality, gates, acceptance) without being torn apart or dissipated.

In more detail:

* The **inner nested-time aspect**:
  + in V1: matter has internal time – processes that unfold within it, consistent with the outer time,
  + in V2: the engine can represent internal dynamics of a pattern as:
    - changes in its internal tags and substructure in (Q\_k),
    - while still presenting the pattern as “the same part” at the band appropriate to its outward role.
* The **outer stability aspect**:
  + in V1: matter retains an identity across ticks and roles,
  + in V2: the pattern keeps:
    - appearing in the world record (W\_k) at similar positions or roles,
    - passing the same set of gates,
    - being recognized by the engine’s feature extraction as a part of a given type.

Put simply, a matter-like object is a **relational pattern that the engine keeps re-using without breaking it**, because all the structural and temporal tests say “yes, this is still a coherent unit.”

**Stability criteria in L-role terms**

Seen through the L-role lens:

* **L1 stability**:
  + the pattern frequently appears in L1 candidate sets as a **unit** – it is not constantly being decomposed into fragments at that band,
  + it contributes many times to candidate futures, showing that it is a reliable past-unit.
* **L2 stability**:
  + the pattern fits consistently into outer environments as a specific kind of object (e.g., a particle, a cell, a macro-object),
  + different L1 candidates that include this pattern can still cohere at L2 into a single environment – the pattern does not cause context-breaking anomalies.
* **L3 stability**:
  + when multiple candidates involving this pattern are in play, L3 selection often **retains** the pattern,
  + it is not systematically eliminated by the acceptance and PF/Born steps – which would indicate that it is dynamically disfavoured or short-lived.

Over many ticks and runs, patterns with these stability signatures are the engine’s **matter-like structures** at the various bands.

**Implications for phenomenology and simulations**

From a phenomenological and simulation standpoint:

* When you simulate scenes in the engine:
  + objects that we would ordinarily call “particles,” “cells,” or “macro-objects” are those patterns that show strong **stability and reuse**:
    - they keep reappearing,
    - they interact with each other in structured ways,
    - they are recognized by diagnostic tools as persistent motifs.
* When you look at diagnostics:
  + you can measure **stability** by:
    - the lifetime of patterns across ticks,
    - the frequency with which they pass κ and structural gates,
    - the consistency of their tags in Ξ across time and contexts.
  + you can use these measures to distinguish:
    - short-lived, flux-like structures (background plexity),
    - from long-lived, matter-like structures (nested-time matter).

From the bridge perspective, this ties the abstract notion of **nested-time matter** directly to **what the engine is doing**: matter is not something you add by hand; it is **whatever patterns the engine’s relational and temporal constraints cause to be stable and reusable**.

**8. Constraints as Principles (From Implementation to Axioms)**

**8.1 Finiteness & Discreteness**

The V2 engine is built on a handful of very strict implementation constraints. At first they might look like “just coding choices” – finite alphabets, finite candidate sets, no infinite loops – but from the bridge’s perspective they are more than that. They are **concrete expressions of deeper principles** that were already implicit in the V1 formal core. This subsection starts with the most basic of these: **finiteness and discreteness**.

**Engine side: everything is finite and discrete**

On the V2 side, finiteness and discreteness show up everywhere:

* **Finite candidate sets per tick**
  + At each site (k), the engine only considers a **finite** set of candidate next acts:
    - only finitely many neighbours in the lattice or graph,
    - only finitely many ways of updating the world and qualia records that the selectors generate.
* **Finite feature alphabet Ξ**
  + All information that matters for control is encoded in a **finite alphabet**:
    - a finite set of tag types (band, role, container, pattern type, lane, etc.),
    - a finite set of values for each tag,
    - a finite set of phase bins in Ξ\_phase.
  + The engine never needs an unbounded set of symbols to decide feasibility or selection.
* **Discrete ticks and indices**
  + Time-like evolution is represented by an integer index (k) that advances by +1 per commit.
  + Spatial structure is represented by discrete indices (graph nodes, lattice sites).
  + There is no continuous parameter that the engine uses directly in control; all such structure is mediated via discrete tags and thresholds.
* **Finite histories per run (at any finite k)**
  + For any finite simulation horizon, you only have finitely many ticks and a finitely extended history of states and logs.

This finiteness is not just a practical limitation; it is part of the engine’s **contract**: all control decisions must be made on the basis of discrete, finite data.

**Formal side: why finiteness was implicit from the start**

In the V1 formal core, finiteness and discreteness were already baked in, even if not always spelled out in implementation language:

* A **tick** is a discrete unit of change; there is no “half-tick” in the core relational picture.
* Each **carrier** at a tick has:
  + a specific IN/ON split,
  + a finite number of relations to other carriers at that tick,
  + a finite description of its boundary state, at least from the vantage of the chosen present.
* The **operator algebra** (Renew, Trade, Sync, Sink, framing) acts on these discrete carriers, not on a continuum of states with uncountably many degrees of freedom per tick.
* The **ledger idea** (record, exposure, capacity) assumes that at each tick:
  + a finite amount of new record can be added,
  + capacity is limited,
  + exposure is finite.

In other words:

* Even when V1 uses continuum-inspired language for convenience (e.g., integrals, continuous profiles), the underlying ontology is **discrete ticks and finite relations**.
* The Present-Moment Sphere itself is **a single, finite “now”**, not a continuum of sub-moments.

The engine simply **commits fully** to this discrete footing.

**Crosswalk: finiteness as a shared axiom, not just a coding constraint**

The crosswalk can be stated like this:

* V1’s commitment to “present-only, discrete ticks, finite relations at each tick”  
  ↔
* V2’s insistence that the engine only operate with **finite candidate sets, finite feature alphabets, and discrete time/space indices**.

From the bridge’s standpoint:

* What looks like an implementation detail in the engine is actually the **natural operational form** of the formal core’s ontology:
  + a present is a finite whole,
  + it relates to finitely many others in the relevant sense,
  + it can only process finitely many alternatives at a time.

When you design the engine, this principle becomes a **hard rule**:

* If a proposed modification would require:
  + infinite candidate sets,
  + unbounded feature alphabets,
  + or control logic that depends on continuous parameters,  
    then it is not just “inefficient”; it is **out of line with the theory’s ontology**.

**Practical consequences for theory and simulations**

Taking finiteness and discreteness as a principle rather than an accident has several practical consequences:

* It keeps the **engine analysable and auditable**:
  + every step can, in principle, be enumerated and checked,
  + there is no hidden “infinite reservoir” where unexplained behaviour can lurk.
* It shapes how you think about **approximation**:
  + if you need to approximate a continuum, you do it via **better discretization schemes** and **richer but still finite** feature sets,
  + you do not abandon the discrete structure; you refine it.
* It fits the **nested-present ontology**:
  + each present is a finite object with finite inner and outer relations at its hinge,
  + larger-scale “continua” are emergent descriptions of many discrete relations, not something you ever give directly to the engine as a control object.

From the bridge perspective, finiteness and discreteness are therefore **not negotiable implementation quirks**; they are the **engine translation of the theory’s basic stance**: reality is built from discrete present-acts, and any faithful engine must reflect that in its core mechanics.

**8.2 Locality & the No-Skip Rule**

A second core implementation constraint in the V2 engine is **locality**: every act only connects a present to its immediate successor, and only to nearby positions in whatever space or graph you are using. This is enforced as the **no-skip rule**. From the bridge’s perspective, this is not just a coding habit; it is the concrete realization of V1’s commitment to **neighbor-only tick relations** and a **local** notion of causality.

**Engine side: no skipping in time, finite neighborhood in space**

In the engine, locality appears in two main ways:

* **No skip in the tick index**
  + Every commit advances the tick index from (k) to (k+1).
  + There is **no control path** that allows:
    - jumping directly from (k) to (k+2) without passing through (k+1),
    - going backward from (k) to (k-1),
    - or updating multiple future ticks in a single act.
  + All decisions about the next state are made at (k), and only the next state ((k+1)) is written.
* **Finite spatial/graph neighborhood**
  + For each site ((x, k)), the engine only considers **neighboring positions** ((x', k+1)) as candidates:
    - for example, immediate lattice neighbours, or nodes within a fixed finite hop-count on a graph,
    - there is no control logic that directly connects ((x, k)) to arbitrarily remote spatial positions in a single tick.
  + The selectors enumerate candidates only within this **finite neighborhood**.

Taken together, these rules enforce that **every act is a local step**:

* Local in time: one tick at a time, no jumps.
* Local in space: limited to a finite, physically meaningful neighborhood.

**Formal side: neighbor-only relations and causal structure**

In the V1 formal core, this locality was always implicit in the way the theory is formulated:

* A **tick** is the update from one present to its immediate successor along a given thread:
  + the operator algebra acts on carriers in a sequential way,
  + there is no formal operation that says “skip two presents ahead” without passing through the intermediate ones.
* The **relational network** at a tick connects carriers to their **immediate neighbors** in the outer context:
  + the ON side describes relations to nearby contexts, not arbitrary long-range jumps,
  + updating those relations from tick to tick is done via local operator actions, not by global rewrites.
* The **invariant interval and causal cones** derived in the formal theory rely on this locality:
  + you derive light cones and causal structure by composing many small steps that each respect relational constraints,
  + you do not assume a continuum where events are directly related across arbitrarily large separations in a single operation.

So even when the abstract language may speak of “paths” or “geodesics” through the network, these are understood as sequences of **local** relational changes, one tick at a time.

**Crosswalk: locality as a shared structural commitment**

The crosswalk is straightforward:

* V1’s “each tick links only to its immediate successor and to local neighbors in the relational network”  
  ↔
* V2’s no-skip rule (only (k \to k+1)) and finite neighborhood selection (only nearby spatial/graph moves).

From the bridge perspective:

* The engine’s locality constraints are the **operational form** of the formal core’s relational locality:
  + they ensure that causal structure, cones, and the geometric picture built from many small steps are preserved,
  + they prevent any hidden, nonlocal “shortcuts” that would break the theoretical picture.

If an engine modification tried to introduce:

* direct jumps in (k) (e.g., (k \to k+2) in one act), or
* unbounded spatial jumps (e.g., connecting arbitrary distant sites in a single tick),

it would not just be a different implementation; it would be **incompatible** with the assumptions behind the V1 derivations of the invariant interval and causal structure.

**Why locality matters for emergent physics**

Locality in the engine is what makes many of the emergent physics results meaningful:

* **Cones and signal speeds**:
  + because you can only move one tick at a time and only within a finite neighborhood, there is a natural maximum effective speed for propagation,
  + this underwrites the emergence of light cones when budgets are applied correctly.
* **Accumulated effects (gravity, fields)**:
  + gravitational and gauge-like effects are encoded in **local feasibility gradients** (e.g., ParentGate shells, structural gate rules),
  + their influence builds up over many ticks as paths repeatedly encounter these local constraints,
  + if you allowed nonlocal jumps, you would spoil that accumulated picture.
* **Stability and nested-time matter**:
  + the persistence of matter-like motifs depends on consistent local evolution;
  + if an act could arbitrarily reach into distant parts of the network, stable patterns would be much harder to define and track.

In short, locality is what allows you to go from **local relational rules** to **global geometric and dynamical behaviour** in a controlled way.

**Practical consequences for engine design and analysis**

Treating locality and the no-skip rule as principles rather than conveniences has clear practical implications:

* Any proposed change to the engine must be checked for compliance with:
  + **no-skip in k**: every update is still one tick, one step forward,
  + **finite neighborhood selection**: candidates remain local in the underlying spatial/graph structure.
* When you design new scenes or experiments:
  + you think in terms of **local rules and interactions**,
  + you look at how repeated local acts give rise to the macroscopic behaviour you care about,
  + you avoid designs that implicitly require “instantaneous” or “global” operations in control.

From the bridge standpoint, locality and the no-skip rule are therefore **axioms in implementation form**: they are how the engine embodies the formal theory’s insistence that reality is built from **local, sequential present-acts**, not from global, all-at-once updates.

**8.3 Curve-ban as Combinatorial Control Principle**

Another core engine constraint is the **curve-ban**: the control path may not use **continuous weights, fitted curves, or floating-point scoring functions** to decide feasibility or select outcomes. At first glance this can look like “just a coding rule” (for reproducibility or robustness), but from the bridge’s perspective it is much deeper than that. It is the concrete expression of the theory’s commitment to **purely combinatorial control** at the present, with all continuous-looking structure emerging only at the level of configuration and diagnostics.

**Engine side: what the curve-ban actually forbids**

In V2, the curve-ban forbids any use of:

* **Continuous weight functions** inside control:
  + no functions like (w(x) = e^{-x^2}) or (w(x) = g(D)) multiplying candidate scores,
  + no smooth potentials used to bias selection.
* **Fitted curves or regression-derived functions** as part of acceptance:
  + the engine cannot, for example, fit a 1/r curve to candidate properties and then use the resulting coefficients as weights in control,
  + regression, fitting, and other continuous modeling are allowed only in diagnostics, never in the decision logic.
* **Arbitrary floating-point scoring** in the gate and acceptance pipeline:
  + gates return booleans or integers/ratios, not arbitrary real-valued “scores,”
  + ratio-lex acceptance compares integer/ratio residuals; it does not aggregate continuous scores into a single “fitness” value via weights.

More generally, the curve-ban insists that:

* **All control decisions** – what is admissible, how candidates are ordered, when ties occur – must be expressible in terms of:
  + finite feature tags,
  + boolean predicates,
  + integer or rational comparisons,
  + and fixed, discrete orderings (like lexicographic rules).

Any continuous structure (like a gravitational 1/r profile, a smooth field, or a fitted relationship) is pushed **out of the control path** and into configuration or analysis.

**Formal side: why purely combinatorial control fits the ontology**

In the V1 formal core, the present is always treated as a **discrete, finite entity**:

* It is defined by **which relations exist and which do not**, not by continuous weights on those relations.
* The tick-operator algebra:
  + either allows a relation to be part of the next carrier or it does not – there is no halfway membership with a real-valued coefficient in the ontologically primitive description,
  + acts on carriers via discrete operations (Renew, Trade, Sync, Sink, framing), not via continuous rescaling of weights.

The presentation of geometry, fields, and pivot structure in V1 uses continuous language (dimensions, profiles, couplings) as a **summary of patterns over many relations**, not as a literal per-relation weight attached at the hinge:

* The dimension profile (D(n)) and pivot (g(D)) describe **how many and which relations are feasible** at each role, not with what continuous weight each relation should be chosen at the present.
* The invariant interval and geodesic notions come from discrete **counting and composition of ticks**, not from assigning a continuous length to each tick and then optimizing an integral in the control path.

From this standpoint, the curve-ban is a natural consequence:

* If the present is fundamentally a **combinatorial object** (a finite pattern of relations and tags), then the control mechanics at the present should also be combinatorial.
* Continuous quantities belong to **summaries of many presents** and to **emergent descriptions**, not to the primitive act of deciding a single present.

**Crosswalk: curve-ban as “no hidden metric in control”**

The bridge can phrase the correspondence like this:

* V1’s insistence that:
  + the present is a discrete whole,
  + relational suitability is encoded in admissibility predicates,
  + geometry and fields are relational summaries over many acts,

↔

* V2’s rule that:
  + the engine must not smuggle in a **hidden metric or field** in the form of continuous weights in control,
  + all admissibility and ordering decisions must be made using **tags, predicates, and integer/ratio comparisons**.

In other words, the curve-ban is the engine’s way of saying:

“If there is a ‘metric’ or ‘potential’ influencing acts, it must appear as a pattern in which acts are allowed or disallowed and how discrete residuals are defined, not as a numeric weight that secretly steers the decision.”

This keeps the control logic **transparent** and **faithful to the ontology**.

**Geometry and fields via configuration, not runtime weights**

The bridge discussion of geometry and gauge structure already showed how:

* (D(n)) and (g(D)) are realized via:
  + band definitions,
  + Θ/κ threshold ladders,
  + ParentGate shell schedules,
* charges and symmetries are realized via:
  + discrete tags in Ξ,
  + structural gate rules on allowed transitions.

The curve-ban is what forces this design:

* Instead of computing (g(D)) at runtime and multiplying candidate scores by it, you:
  + decide **before running** how the pivot profile should affect which patterns are allowed,
  + encode that in discrete configuration knobs (thresholds, on/off switches, shell strictness),
  + and then let the engine enforce those choices via predicates and residual comparisons.

Likewise, instead of using a smooth potential in control, you:

* define discrete **“higher effort” regions** via tighter gates,
* or discrete **“zero feasibility” regions** via absolute gate failures.

This is what it means to keep geometry and fields in **configuration and diagnostics** while keeping control combinatorial.

**Benefits for transparency and falsifiability**

Treating the curve-ban as a principle has important benefits:

* **Transparency**:
  + it is always clear *why* a candidate was accepted or rejected – you can point to specific predicates and residual comparisons,
  + there is no opaque combination of real weights where a tiny change in a parameter, or a misfit curve, silently flips decisions.
* **Falsifiability and reconfiguration**:
  + if a pattern of behaviour does not match the theory’s predictions or empirical evidence, you know the only levers are:
    - feature definitions,
    - gate predicates,
    - manifest parameters.
  + you do not have “hidden degrees of freedom” in the form of arbitrary continuous weight functions to tune until the data fits.
* **Alignment with the relational stance**:
  + keeping control combinatorial emphasises that the **fundamental content of the theory is relational and discrete**,
  + smooth behaviour in experiments is then something you **derive** statistically from many discrete acts, not something you **impose** as a control law.

From the bridge’s point of view, the curve-ban is therefore a **core conceptual bridge**:

* It ties the abstract commitment to present-only, relational discreteness in V1 to a very concrete design rule in V2: **no curves or continuous weights in control; only combinatorial logic at the present.**

**8.4 Auditability & Manifest as Part of the Theory**

The last major implementation constraint is **auditability**: the engine must be configured and run in a way that is **explicit, inspectable, and reproducible**. In V2, this shows up as the requirement that every run is governed by a **manifest** (a declarative configuration) and that a suite of **audits** must pass for claims based on the run to be taken seriously. From the bridge’s perspective, this is not just good engineering practice; it is the operational expression of the V1 formal core’s demand for **transparent relational consistency**.

**Engine side: manifest and audits**

On the V2 side, two things are central:

* **The manifest**
  + The manifest is a **declarative configuration** that specifies:
    - the unit map between inner and outer time (the effective “c”),
    - band definitions and mappings from physical scales to context bands,
    - Θ and κ ladder settings (time windows, granularity thresholds),
    - ParentGate schedules (shell structure and strictness per container),
    - which structural gates are active and how they are parameterized,
    - the feature alphabet Ξ structure (available tags, phase bins),
    - random seed policies and any other discrete control inputs.
  + It is loaded **once** at the start of a run and treated as **read-only** during control:
    - gates and selection logic refer only to manifest-defined parameters,
    - no hidden tuning or mid-run changes are allowed in the core control path.
* **Audits**
  + Audits are **post-run and runtime checks** that enforce theoretical hygiene:
    - **curve-ban compliance** (no prohibited continuous weights in control),
    - **no-skip and locality** checks,
    - **measure invariance** and single-read of hinge-related configuration,
    - **isotropy and monotonicity** for ParentGate schedules,
    - **no-signalling** in correlation-type scenes,
    - **SR compatibility**, **UGM/hinge integrity**, and other domain-specific checks.
  + If audits fail beyond specified tolerances:
    - the run is treated as **diagnostic only**, not as evidence for the theory’s predictions,
    - configuration or implementation must be revisited.

Together, the manifest and audits make the engine **inspectable**: given a run and its manifest, another agent can reconstruct what the engine did and verify that the choices were consistent with the theory’s constraints.

**Formal side: transparent relational consistency**

In the V1 formal core, there is an analogous demand, stated in more abstract terms:

* The theory insists on **relational consistency**:
  + the same operator algebra must apply in the same way wherever its conditions are met,
  + there should be no hidden “different rules” in different parts of the network that are not accounted for in the ontological structure.
* It treats **hinges, pivots, and context roles** as:
  + well-defined structural positions in the relational web,
  + not as ad hoc knobs that can be tuned arbitrarily from one case to another.
* It expects **derivations and identities** (like the invariant interval, pivot normalizations, Born-rule structure) to be:
  + globally applicable wherever their hypotheses hold,
  + not broken by hidden exceptions or unarticulated adjustments.

In effect, the formal core is saying:

“If you claim this is the operator algebra and these are the ladders and pivots, then the system must behave accordingly everywhere those constructs apply. Any deviations must be identifiable and attributable to changed conditions, not to hidden parameters.”

The manifest and audits are V2’s way of making that **explicit and checkable**.

**Crosswalk: manifest and audits ↔ explicit instantiation of relational structure**

The bridge can state the correspondence this way:

* V1’s requirement that:
  + the operator algebra, context ladder, and pivots be **globally coherent** and **transparent**,
  + no extra, unexpressed degrees of freedom be smuggled in,

↔

* V2’s design where:
  + **all control-relevant configuration** lives in the manifest,
  + **audits** verify that the engine’s behaviour respects the structural constraints implied by the formal core,
  + **no control parameter** affecting admissibility or selection is permitted to live outside the manifest or to change mid-run.

The manifest is thus not just a configuration file – it is the **engine-side representation of the theory’s global relational structure** for that scenario. Audits are the engine’s way of checking that this representation has been adhered to.

**Manifest as part of the theory, not just “code config”**

Treating the manifest as part of the theory has several consequences:

* **The manifest is a first-class theoretical object**:
  + it encodes how abstract objects like (D(n)), (g(D)), context bands, and containers are instantiated in a particular scenario,
  + choices in the manifest are constrained by the formal theory – not arbitrary; they must reflect ladder roles and pivot identities.
* **Differences between manifests correspond to different theoretical conditions**:
  + for example, a manifest representing “Earth-only” versus “Earth + galactic container” are different instantiations of the same underlying relational framework,
  + each can be evaluated against the formal expectations (e.g., predicted redshift patterns, context-level behaviour).
* **Publishing a manifest and its audits is part of publishing a result**:
  + to claim “this engine run probes such-and-such prediction of the theory,” you also publish:
    - the manifest (so others know exactly how the abstract structure was instantiated),
    - the audit results (so others can see the structure was respected in execution),
    - any diagnostics derived from the run.

In that sense, manifest and audits live **between** theory and implementation: they are where abstract commitments meet concrete code.

**Practical implications for working with V1 and V2**

Operationally, this principle means:

* When you change something in the **formal core**:
  + you must be able to point to where the manifest will change to reflect the new structure,
  + and to which audits will need to be updated or extended.
* When you change something in the **engine**:
  + you must ensure that the manifest captures any new control parameters,
  + you must add or adjust audits to guarantee the new engine behaviours remain consistent with the theory.

If those links cannot be made, the bridge would consider the change suspect: an engine behaviour or parameter that cannot be traced back to – or forward to – a clear formal statement is likely an unprincipled hack rather than a genuine extension of the theory.

From the bridge’s perspective, **auditability and the manifest** are therefore part of the **axiomatic content** of the unified framework:

* They guarantee that the engine is not a black box but an open, checkable instantiation of V1’s relational structure.
* They ensure that future work – whether simulations, experiments, or refinements of the formal theory – can be tied back to a well-defined configuration and a set of structural checks.

In summary: manifest and audits are the **engine-level axioms about transparency**. They are how the combinatorial present-act engine keeps faith with the formal core’s demand for clear, relationally grounded dynamics, and how the unified theory remains falsifiable and reproducible rather than drifting into an opaque, parameter-tuned model.

**9. Unified Hinge & Unified View (Space, Matter, Qualia)**

**9.1 Triple Coincidence at the Hinge**

One of the most important pieces of the unified framework is that three things that, in ordinary treatments, are kept separate all **land on the same structure** here:

1. The **geometric hinge** in the V1 formal core (D=2 at the context pivot).
2. The **engine hinge** in V2 (the UGM grain and the present-act cycle at 0↔+1).
3. The **phenomenological hinge** (“this now” – the specious present where space, matter, and experience show up together).

This subsection makes that “triple coincidence” explicit.

**Geometric hinge: D = 2 at the context pivot**

From the V1 side:

* The context ladder has a special role at (n=0), where:
  + the effective dimension is exactly (D(0) = 2),
  + the pivot function is neutral (no extra geometric weighting),
  + the boundary behaves like a 2-sphere.
* At this role, the Present-Moment Sphere boundary:
  + is where inner and outer faces meet,
  + is where the invariant interval construction is anchored,
  + is the place where geometry is “balanced” between inward and outward.

This is the **geometric hinge**: a privileged role in the ladder where the formal core says “this is the pivot between inner and outer.”

**Engine hinge: UGM grain and 0↔+1 act**

From the V2 side:

* The UGM band (around 0.1–0.2 mm) is the **spatial grain** at which:
  + inward plexity first reliably appears as **distinct outward parts**,
  + κ and structural gates start treating structures as units in the outward read,
  + the engine’s hinge equality and per-band gate settings are tuned to be neutral in the sense of the D=2 pivot.
* The 0↔+1 act – the present inside its immediate container – is the **time-like hinge**:
  + the engine’s typed budgets are defined relative to this coupling (inner ticks vs outward ticks),
  + the no-skip rule and budget typing ensure that the invariant-interval-like relation is enforced at this hinge,
  + every present act is a 0↔+1 decision: what the inner network will retain and how it will stand in the outer context.

This is the **engine hinge**: the specific grain and role combination at which the engine’s combinatorial machinery is calibrated to the geometry of the V1 pivot.

**Phenomenological hinge: the lived “now”**

From the vantage of experience:

* We do not live in an abstract ladder or a code base; we live in **presents**:
  + each present is a “now” with an inner sense of “me” and an outer sense of “world,”
  + each present carries a specious duration (on the order of a tenth of a second) within which many inner and outer changes are experienced as one act.
* Empirically, several facts line up:
  + the **specious present** in human experience (integration window of ~0.1 s),
  + the **UGM scale** of around 0.1–0.2 mm as a natural “pixel” for our direct perception and many cross-domain structural pivots,
  + the **Earth-surface context** as an effectively 2D environment where our space is laid out.

When you stand at your own present, you naturally experience:

* **space** as the outward read of the +1 context at roughly the UGM grain and above,
* **matter** as stable patterns at and around that grain that behave as parts in that space,
* **qualia** as the inner stream that retains and re-relates these patterns.

All three – geometry, engine, phenomenology – **pick out the same hinge**: the grain, role, and time span where the inward network, outward container, and lived “now” cohere.

**Why the triple coincidence matters**

The triple coincidence is not an extra assumption; it is a consistency check and an anchor:

* It says that the scale and role where:
  + the formal core’s geometry is neutral (D=2 pivot),
  + the engine’s combinatorial rules are calibrated (UGM grain, 0↔+1 budgets),
  + and our lived present is organized (specious present, bodily and perceptual resolution),

are **one and the same structure** in the unified theory.

From the bridge perspective, this means:

* When you talk about “the present” in any register – abstract, engine, or experiential – you are always pointing back to **the same hinge**, not three different things.
* It provides a natural anchor for:
  + defining context bands (−2, −1 inward; +1, +2, +3 outward),
  + tying simulation scales to phenomenological scales,
  + interpreting stability and matter-like behaviour in the engine as the things we actually perceive as objects in space.

The remaining subsections in this “Unified Hinge” section will build on this:

* Space as the outward hierarchy of containers anchored at this hinge,
* Matter as stable patterns at this hinge and across nearby bands,
* Qualia as the 0-level stream of Qₖ states that, at the hinge, present these patterns as a world.

**9.2 Space as Context Hierarchy**

In this framework, **space is not a pre-existing container** that everything sits in. Instead, space is how the outward context hierarchy **presents itself from the hinge**. The V1 formal core describes this in terms of context roles and their geometry; the V2 engine realizes it as the way **outer bands and containers** are configured and read out; phenomenologically, it is what you experience as “the world around you.” This subsection ties those together.

**V1 view: outer roles as geometric containers**

From the formal side, the outer part of the context ladder – the positive roles – are **containers**:

* At **+1**, you have the immediate container of the present:
  + the environment that coordinates many 0-level presents at once,
  + where geometry appears as an effectively two-dimensional surface (D≈2) from the hinge’s viewpoint.
* At **+2**, you have a larger container:
  + for example, a galactic disk,
  + again exhibiting surface-like behaviour in appropriate analyses,
  + coordinating many +1 contexts (planets and their environments).
* At **+3**, you have an even larger container:
  + cosmic shells like the last-scattering surface or full-sky source distributions,
  + still presenting effective D≈2 when viewed as a spherical boundary,
  + coordinating many +2 contexts.

In V1 language:

* “Space” is how **the ON side of the carrier** looks when you project outward along these container roles:
  + containers define **where** and **how far** you can stand relative to other carriers,
  + the **invariant interval** and pivot structure tell you how separation is measured and how it is constrained.
* The context ladder is a **hierarchy of containers**:
  + +1, +2, +3 are not just abstract numbers; they represent successive “layers” of outward coordination that together define the large-scale layout in which carriers find themselves.

From this point of view, space is **the relational geometry of the outer context hierarchy** as seen from the hinge.

**V2 view: banded containers and outward read**

In the engine, this hierarchical view of space appears as:

* **Band structure**:
  + the six-band ladder associates each outward role with a physical scale range and type of container (Earth, galactic disk, cosmic shell),
  + features in Ξ include band tags and container identifiers,
  + gates and schedules are defined per band and per container.
* **ParentGate configuration**:
  + for each container, radial shells are defined (e.g., around an Earth-like object),
  + strictness schedules are set so that feasibility reflects the container’s expected geometry and pivot profile,
  + rotation invariance ensures the container is treated as a **surface-like environment** in expectation.
* **Outward read at 0**:
  + from the 0-level present, the engine’s outward read uses:
    - the world record (W\_k),
    - band tags,
    - and container structure  
      to determine **how the environment presents itself**:
    - as a locally near-2D surface at +1,
    - embedded in larger structures at +2 and +3.

In this picture, space is:

* the result of **reading the configuration of outer bands and containers** from the 0-level present,
* **encoded** in:
  + which containers are active in the manifest,
  + how ParentGate schedules and structural gates shape feasible configurations,
  + what pattern of positions and distances the engine presents in (W\_k).

From the engine’s point of view, space is not a continuous coordinate system it was given; it is a **pattern in which world records, bands, and containers are organized**.

**Hierarchy: from local surface to cosmic shell**

Put together, V1 and V2 tell essentially the same story in different languages:

* The **local surface**:
  + at +1, the Earth band, you have:
    - a contiguous, roughly planar surface from the hinge’s vantage,
    - where everyday objects and movements happen,
    - where the invariant interval and budgets give you relativistic structure in a 2D+time presentation.
* The **intermediate container**:
  + at +2, galactic-scale bands, you have:
    - a disk-like context that shapes large-scale structure,
    - which enters the engine only through manifest and gates (it is not “felt” directly in every act, but its existence shapes feasible configurations and long-scale behaviour).
* The **outer shell**:
  + at +3, cosmic bands, you have:
    - a sky-shell or horizon context,
    - which provides the boundary conditions and long-range pivot profile for everything inside.

Space, in this unified picture, is **this hierarchy of containers** as they appear to a present:

* Each container role contributes a layer to the outward read.
* The combination of these layers gives you the full sense of “where things are and how far apart they are.”

**Phenomenological reading: why space feels like “where we are”**

From the standpoint of experience:

* You normally identify “space” with:
  + the Earth-surface environment (your immediate +1 context),
  + extended by visual, symbolic, and measurement-based inferences to:
    - global and cosmic structures (+2, +3),
    - sometimes down to local micro-structure (−1, −2) when using instruments.
* What makes space feel **objective and stable** is:
  + the consistency of the +1 container (Earth band) across many 0-level presents – it is the **shared environment** that many agents inhabit,
  + the persistent patterns of +2 and +3 containers as revealed by astronomy and cosmology,
  + the fact that all of this is anchored at the hinge where your present meets the context hierarchy.

The engine reproduces this by:

* making +1, +2, +3 band structure and containers explicit,
* using them to shape feasible world records and budgets,
* presenting space as **whatever the outward read at 0 sees when containers are configured as they are**.

**Summary at this level**

At this stage of the bridge:

* V1 describes space as the **geometry of the outward context ladder**, anchored at the D=2 hinge and structured by containers at +1, +2, +3.
* V2 implements space as:
  + **banded container configuration** in the manifest,
  + **ParentGate and structural gates** that realize the corresponding feasibility gradients,
  + **outward reads** of (W\_k) that present this hierarchy as a world.

Phenomenologically, space is what you experience when:

* your present (0) is embedded in the +1 container (Earth band),
* that container in turn sits in +2 and +3,
* and the engine’s combinatorial dynamics present all of this as a coherent, navigable environment.

**9.3 Matter as Stable Engine Pattern**

In this unified framework, “matter” is not a primitive substance. It is **whatever remains stable as a pattern across ticks and across context bands**, and shows up as a part in the outward read at the hinge. The V1 formal core describes this as **nested-time structure** that persists under the operator algebra; the V2 engine realizes it as **stable motifs** in (W\_k) and (Q\_k) that repeatedly survive gates and act as reusable L1 units. This subsection pulls those threads together and ties them to our everyday sense of “physical objects.”

**V1 view: matter as persistent nested structure**

From the formal side:

* Matter is a **pattern in the relational network** that:
  + persists through many ticks (it is not destroyed by the usual action of Renew, Trade, Sync, and Sink),
  + maintains a recognizable identity when seen from the hinge,
  + has **inner dynamics** (nested time) that are compatible with the outer context’s time.
* This pattern spans multiple context roles:
  + inward roles (−1, −2) encode how the pattern’s internal structure evolves: molecules, cells, tissue structure, etc.,
  + outward roles (+1, +2, +3) encode how the pattern is embedded in containers (organism, planet, galaxy, cosmos).
* The formal requirements for something to count as matter-like are:
  + **stability under the operator algebra**: repeated ticks act on it but do not disperse it into generic plexity,
  + **compatibility with pivots and bands**: it lives consistently at certain scales and roles without constantly violating admissibility predicates.

In this language, a “particle,” a “cell,” or a “macro-object” is just a **long-lived, multi-band relational motif** that fits the theory’s stability and admissibility structure.

**V2 view: matter as persistent, reusable motifs in (W\_k) and (Q\_k)**

In the engine, the same idea appears as **persistent patterns** in the world and qualia records:

* A matter-like pattern is one that:
  + appears as a **coherent, contiguous structure** in (W\_k) and (Q\_k) (e.g., a cluster of sites, tags, and relations),
  + **reappears across ticks** at similar locations or roles,
  + survives κ and structural gates that would otherwise treat small or incoherent patterns as mere “texture.”
* These patterns are:
  + **granular at the right band**: κ says “this is a part,” not “this is sub-part texture,”
  + **structurally regular**: contiguity, degree, orientation, and history checks consistently pass for them.
* In L-role terms:
  + At **L1**, they are the “past-units” the engine keeps reusing to build candidate futures,
  + At **L2**, they slot into outer environments as recognizable objects (e.g., a particle in a field, a cell in tissue, a car on a road),
  + At **L3**, they are frequently retained through acceptance: candidates involving them often win, meaning the pattern remains in the realized history rather than being systematically eliminated.

Operationally, matter-like patterns are those that the engine’s gates and selection rules **keep bringing back as units**, tick after tick.

**Crosswalk: stable relational motifs ↔ stable engine patterns**

We can now state the mapping:

* **V1 nested-time matter** ↔ **V2 stable motifs in (W\_k)/(Q\_k) that:**
  + pass κ and structural gates consistently at their band,
  + maintain their identity (same tags, similar structure) across many ticks,
  + function as reusable L1 units and coherent L2 objects.

More concretely:

* Inner nested structure:
  + In V1: matter has internal time and complexity at inner roles (−1, −2),
  + In V2: the pattern has rich internal tagging and substructure in (Q\_k), which evolves over ticks but preserves an identifiable whole at its outward band.
* Outer embedding:
  + In V1: matter fits into containers and obeys pivot and context rules at +1, +2, +3,
  + In V2: the pattern consistently occupies roles and positions in (W\_k) that obey band-specific gates and ParentGate schedules, so it behaves properly in the container hierarchy.

Put simply: a matter-like thing is what the engine **cannot get rid of** without breaking its own structural rules—because those rules encode the stability that the formal core demands of matter.

**Phenomenological reading: why matter feels “solid”**

From our lived perspective, this stability is what makes matter feel **solid and reliable**:

* The patterns that satisfy the engine’s stability and admissibility conditions are exactly the ones that:
  + we can track over time (a rock, a table, a body),
  + we can interact with and expect consistent responses from,
  + we label with persistent names and treat as “objects.”
* These are not “substances” sitting in an external space; they are **persistent modes of the engine’s relational activity**:
  + stable sequences of (Q\_k) (inner) and (W\_k) (outer) passing the same gates over and over,
  + nested across bands in a way that matches our multiscale experience (micro → macro → cosmic).

When the engine’s design and the V1 core agree on what counts as a long-lived pattern, the things that persist in (W\_k)/(Q\_k) are precisely the things we recognize as **matter** in space.

**Summary at this level**

In this unified view:

* Matter is **neither purely “in the world” nor purely “in the mind”**; it is a stable relational pattern that:
  + the formal core identifies as nested-time structure,
  + the engine continually re-uses as a unit,
  + and the present experiences as a stable object in its outward space.

The next subsection will complete this triad by looking at **qualia**: how the stream of (Q\_k) states and their equivalence classes encode the inner side of this picture—how experience and content relate to the same underlying present-act engine that is producing space and matter.

**9.4 Qualia as Qₖ Stream and Equivalence Classes**

The last piece of the triad is **qualia** – the “what-it-is-like” content of experience. In ordinary language, qualia are often treated as something fundamentally different from “matter” and “space.” In this framework, they are **another face of the same present-act structure**: the **inner stream** of states at the hinge. The V1 formal core talks about qualia as patterns in the present-moment network; the V2 engine implements them as the evolving **Qₖ stream and its equivalence classes**. This subsection connects those descriptions.

**V1 view: qualia as patterns on the inner side of the present**

In the formal core:

* A present is not just a point; it has **content**:
  + an inner network (IN) encoding what has been retained,
  + a boundary configuration encoding how that inner network meets the outer context.
* Qualia are:
  + the **patterns of relation** on that inner side,
  + as they are accessed by a given carrier at its hinge.

Key features:

* **Qualia are relational**:
  + they are not “little mental atoms,”
  + they are the way certain IN patterns show up to a carrier as “this colour,” “this sound,” “this feeling” when combined with a particular boundary and context.
* **Qualia are present-only**:
  + the theory does not posit qualia “stored somewhere” as separate objects,
  + each tick, the present’s IN/ON structure gives rise to a particular “what-it-is-like” content, and that is all there is ontically.
* **Equivalence of experience**:
  + two present-moment states can be considered **phenomenally equivalent** if:
    - their relevant IN patterns and boundary conditions are isomorphic in the right sense,
    - they would drive the same responses at the next act in the same context.
  + This gives a formal way to talk about “the same experience” across different carriers or times.

So, in V1 language, qualia are **particular equivalence classes of inner patterns at the hinge**, not a separate substance.

**V2 view: Qₖ as the inner state stream**

In the engine, the qualia side is realized as the **Qₖ records** at each tick:

* At each site (k), (Q\_k) is the **inward record**:
  + a finite set of qualia candidates (or “pixels”) representing what the present is taking in,
  + organized by feature tags in Ξ and structured according to κ and structural gates.
* As the engine runs:
  + (Q\_0, Q\_1, Q\_2, \dots) form a **stream of inner states**,
  + each commit updates (Q\_{k+1}) based on:
    - the chosen world/qualia candidate pair at that tick,
    - the application of gates and budgets,
    - any band- and container-specific rules.

In addition:

* The engine’s feature extraction on (Q\_k):
  + groups qualia pixels into **equivalence classes**:
    - by tags (e.g., same colour-channel, same spatial location relative to the hinge, same pattern-type),
    - by relational role (e.g., forming part of the same object pattern, part of the same temporal pattern).
  + These equivalence classes are, operationally, what the engine treats as “the same inner content” across ticks or across different carriers.

From the engine’s perspective, “what-it-is-like” at a given present is just **whatever pattern (Q\_k) presents when read through the feature alphabet and L-roles**, at the hinge, in that manifest.

**Crosswalk: qualia ↔ Qₖ stream and equivalence classes**

The mapping between the two views is:

* **Qualia in V1** ↔ **patterns in the Qₖ stream that belong to the same equivalence classes** under the engine’s feature and relational structure.

Concretely:

* The **inner network IN** at tick (k) is encoded in (Q\_k).
* The **“what-it-is-like” content** is:
  + the **configuration of Qₖ** at the hinge,
  + as interpreted by:
    - band and container tags,
    - spatial/temporal layout,
    - structural relations among qualia pixels,
    - and L1/L2 roles (how those patterns are being used in candidate futures).
* Two experiences are **qualitatively the same** (for the purposes of the theory) if:
  + their Q-stream segments (e.g., around ticks (k) and (k')) fall into the **same equivalence classes** under:
    - feature-based grouping,
    - relational structure (who connects to whom),
    - gate behaviour (which acts they enable or block).

In practice, “the same red patch” across time or across observers is something like:

* “Qₖ contains a cluster of qualia pixels with tags and relations that, under the engine’s equivalence criteria, match the cluster from Qₗ in another tick or agent.”

**Matter vs qualia vs space at the hinge**

Putting this together with the previous subsections:

* **Space** is the outward read of the context hierarchy (the layout of containers, as seen in (W\_k) and budgets).
* **Matter** is stable patterns that exist both in (W\_k) (as outward objects) and in (Q\_k) (as inward parts of experience), surviving gates and presenting as reusable L1 units.
* **Qualia** are the inner patterns in (Q\_k), grouped into equivalence classes by:
  + their tags,
  + their relational roles,
  + and their effects on future acts (L1/L2/L3 behaviour).

Crucially:

* All three are **different readings of the same present-act engine**:
  + the outer read (space),
  + the stability of patterns across acts (matter),
  + the inner read and its equivalence classes (qualia).

At the hinge, the same engine state is giving you:

* a **world** of containers and objects,
* a **body** and its stable internal structures,
* a **stream of “what-it-is-like”** tied to the patterns in Qₖ.

**Phenomenological reading: why qualia are not “extra stuff”**

From the standpoint of experience:

* It might seem natural to think of qualia as **something over and above** the physical world.
* In this unified picture:
  + qualia are not “extra stuff”; they are **how the inner face of the present reads the same relational structure** that, when read outward, gives you space and matter.
  + the engine does not have a separate module for qualia; it has a Q-stream that:
    - is structurally isomorphic to part of the relational network,
    - is read in a particular way at the hinge to yield “what-it-is-like.”

Thus, the bridge makes explicit:

* V1’s claim that qualia are patterns in the present, not a separate substance,
* V2’s implementation where Qₖ and its equivalence classes are the inner side of each present-act.

Together, they close the circle:

* **Space**, **matter**, and **qualia** are three views of the same present-act dynamics:
  + space = outward container hierarchy as seen at the hinge,
  + matter = stable, multi-band patterns that persist under those dynamics,
  + qualia = inner Q-stream patterns and their equivalence classes, as they are “seen from inside” at 0.

This is the unified hinge and unified view: one engine, one formal core, three faces of the same present.

**10. Navigation Aids and Conclusion**

**10.1 Where to Look in V1**

Because this bridge document sits between your **V1 reference** and your **V2 engine** documents, it is useful to have a quick guide for “where to look in V1” when you want to see the formal core behind something described here. This subsection gives that guide in terms of **topics and themes**, not section numbers, so it remains valid even if you later reorganize the V1 manuscript.

**Ontology and carriers**

When this bridge talks about:

* the **present as primitive**,
* the **IN/ON split**,
* **carriers** and the **Present-Moment Sphere**,

the corresponding V1 material lives in the parts of the reference document where you:

* introduce the **basic ontology** of Absolute Relativity – the present, the inner and outer networks, and the idea that “only presents and relations among them exist,”
* define **carriers** as objects that package IN, ON, and state (h\_k),
* describe the **PMS boundary** and how IN and ON meet there.

In V1, this is typically in the early chapters that motivate the theory and formalize the **present-only, IN/ON, carrier** picture.

**Tick algebra and operators**

When this bridge talks about:

* the **operator algebra** (Renew, Sink, Trade, Sync, framing),
* the **tick cycle** as applying these operators to carriers,
* **admissibility predicates** and the **ledger** ((I, E, K)),

you can find the formal definitions and proofs in the parts of V1 that:

* introduce the **tick-operator set** and specify how each operator acts on carriers,
* define what it means for a carrier or transition to be **admissible**,
* develop the **ledger construction** and show that the record component (I) is monotone (the arrow-of-time result).

Look for the sections where you first formalize the “**step of becoming**” in algebraic terms and then prove properties of repeated application.

**Context ladder, dimension profile, pivot function**

When this bridge discusses:

* the **context ladder** (n),
* the **dimension profile** (D(n)),
* the **pivot function** (g(D)) and the special role of **D=2 at the hinge**,

the corresponding V1 material is in the part of the reference where you:

* define the **integer-labelled ladder of roles** (inner, hinge, outer),
* construct and analyse the **dimension profile** across that ladder,
* introduce the **pivot function** (g(D)) and fix its behaviour at D=2,
* derive the **surface-like nature** of the hinge boundary and outer containers from that structure.

These are the chapters where geometry and scale first enter the theory as properties of the relational network rather than assumed background.

**Invariant interval and relativistic structure**

When this bridge talks about:

* the **invariant interval** relating (\Delta \tau, \Delta t, \Delta x),
* **light-cone structure** and **special relativity** as emergent,

the formal derivations are in the V1 sections where you:

* show that, given the tick-operator algebra and pivot structure, a **quadratic invariant** exists,
* derive the **discrete version of the Minkowski relation** and interpret it in terms of inner vs outer separation,
* discuss how **composing ticks** yields cones and relativistic kinematics.

Look for the parts where you explicitly move from “purely relational ticks” to statements resembling “no path can exceed speed c” and “different worldlines accumulate different proper time.”

**Present Plane, amplitudes, and Born rule**

When this bridge refers to:

* the **Present Plane** ((\mathcal{P}, J)),
* **amplitudes**, **phase structure**, and the **Born rule**,

the V1 reference contains the formal story in the parts where you:

* define the **state space at the hinge** and equip it with a **complex structure** (J),
* introduce **amplitude assignments** for potential outcomes,
* derive or justify the **Born rule** as (|a\_i|^2) from structural or consistency arguments,
* describe **collapse at the hinge** in terms of selecting a ray in this space.

Those chapters make explicit the link between relational structure at the present and quantum-style probabilities.

**Gravity as pivot-driven feasibility**

When this bridge talks about:

* gravity as a **feasibility geometry** driven by the pivot profile (g(D(n))),
* **containers** (planetary, galactic, cosmic) as outer roles with specific geometric behaviour,
* **redshift**, **delay**, and **deflection** emerging from relational constraints,

the formal core for this is in the V1 material where you:

* analyse how changes in (D(n)) and (g(D(n))) across outward roles affect what relations are feasible,
* define **container roles** and their geometric signatures,
* connect these to **time dilation, path bending, and horizon effects** without introducing a separate gravitational field.

Look for the parts that explicitly reconstruct gravitational-like phenomena from the ladder and pivot, rather than from metric postulates.

**Dynamics, master action, and path-sum picture**

When this bridge mentions:

* the **master action** (S[\mathcal{H}]) over histories,
* **extremal principles** for dynamics,
* the **path-sum** picture for quantization,

you can find the formal references in the V1 chapters where you:

* define the **local action contributions** for tick transitions,
* assemble these into a **global action** over histories,
* discuss how **extremal histories** correspond to classical-like behaviour,
* and how **amplitudes and path-sums** provide the quantum-like description.

These are the sections that connect the operator algebra and admissibility predicates to global dynamical preferences and quantum interference.

**Qualia, nested-time matter, and phenomenology**

When this bridge talks about:

* **nested-time matter** as stable relational structure,
* **qualia** as patterns on the inner side of the present,
* the unified picture where **space, matter, and qualia** are three faces of the same present-act,

the V1 side of that story is in the conceptual and philosophical chapters where you:

* interpret carriers and IN/ON structure as the basis for **experience**,
* argue that **matter is nested-time pattern**, not substance,
* explain how **present-only relational structure** can account for both physical behaviour and phenomenal content.

Those chapters provide the philosophical framing that the engine and this bridge render operational.

As you keep working, you can treat this subsection as a **topic index** into V1: whenever a part of the bridge raises a concept you want to see formally proved or defined in its original language, use these topic labels to locate the corresponding passages in your V1 reference document.

**10.2 Where to Look in V2**

Just as the previous subsection gave you a topic-based guide into the V1 reference, this one does the same for your **V2 engine document**. Whenever the bridge talks about some engine-side structure, you can use the themes below to locate the corresponding material in your Present-Act Engine specification, without relying on fixed section numbers.

**Engine pipeline and control path**

When this bridge talks about:

* the **engine pipeline** (enumeration → hinge equality → gates → acceptance → commit),
* the **no-skip rule**,
* the idea of “one tick as one full pipeline pass,”

look in the V2 document where you:

* introduce the **overall engine architecture**,
* describe the per-tick **control loop** from (k) to (k+1),
* explain how selectors, hinge equality, gates, acceptance, and commit are arranged in order.

This is the part of V2 that gives the “from start to finish, this is what happens in one act” story.

**Manifest and configuration**

When the bridge refers to:

* the **manifest** as the declarative configuration,
* **unit maps** (like the effective “c”),
* band definitions, Θ and κ ladders, ParentGate schedules, and feature alphabet structure,

you can find the details in the parts of V2 where you:

* define the **manifest format** (what fields it contains, how they are interpreted),
* describe how **band structure** and **context roles** are encoded there,
* specify how gates and budgets pull their parameter values from the manifest.

That material is the concrete home of everything this bridge calls “configuration” or “encoded geometry.”

**Gates and admissibility**

When this bridge talks about:

* **time gate Θ**,
* **granularity gate κ**,
* **structural gates** (contiguity, degree, orientation, history),
* **CRA gates** (Context-Resolved Admissibility),

you should look in your V2 specification where you:

* define each gate’s **input**, **predicate**, and **outcome**,
* explain how gate results are recorded (e.g., pass/fail, contributions to residuals),
* describe how gates are sequenced and combined to form the full feasibility phase.

Those sections are where the engine-side admissibility predicates live.

**Hinge equality and features**

When the bridge mentions:

* **hinge equality** between world and qualia features,
* the **feature alphabet Ξ**,
* **phase bins** and discrete tags used in equality tests,

consult the parts of V2 where you:

* define the **feature extraction functions** from (W\_k) and (Q\_k) into Ξ,
* describe the structure of **Ξ\_phase** and **Ξ\_disc** (phase component and discrete tags),
* spell out the exact **equality conditions** used at the hinge (including phase tolerance or bin matching).

This is the engine-side realization of the boundary projector and Present Plane mapping.

**Residuals and ratio-lex acceptance**

When this bridge talks about:

* **residual vectors** (outward, inward, cross),
* **ratio-lexicographic acceptance**,
* “fewest-acts” or similar structural tie-breakers,

look into the V2 document where you:

* define the **residual components** and how they are computed from gate outcomes,
* describe the **ordering rule** for comparing candidates via these residuals,
* explain how the engine decides when a tie is genuine and when it is broken deterministically.

That is where the practical details of “local action preference” and “deterministic selection as far as possible” are laid out.

**PF/Born ties-only**

When the bridge refers to:

* the **PF/Born ties-only step**,
* **tie adjacency graphs**,
* **Perron–Frobenius eigenvectors** and squaring components to get weights,

the corresponding engine details are in the parts of V2 where you:

* explain how the engine detects a **true tie** after ratio-lex acceptance,
* define how it builds a **graph or kernel** over the tied candidates,
* describe how the **leading eigenvector** is computed and how its squared components are used as selection probabilities,
* specify how random sampling is actually performed (RNG choices, seeding).

Those sections are the concrete implementation of Born-rule-like behaviour.

**ParentGate and containers**

When this bridge discusses:

* **ParentGate** as the gravity-like feasibility gate,
* **radial shells**,
* **inward-monotone strictness schedules**,
* **rotation-invariance**,

you should look at the parts of V2 where you:

* define the **shell structure** around container centers (indices or radii),
* describe the **strictness schedule** per shell,
* explain how ParentGate’s predicates are computed using shell indices and tags,
* specify how rotation symmetry is maintained (e.g., random phase shifts per shell, symmetric neighborhoods).

Those sections are where container geometry is wired into the engine.

**Typed budgets and SR behaviour**

When this bridge talks about:

* **budget triples** ((\Delta \tau, \Delta t, \Delta x)),
* the **typing rule** that enforces an invariant-interval-like relation,
* emergent **cones and time dilation** in the engine,

refer to the parts of V2 where you:

* define **budget types** and how they are attached to each committed act,
* describe how (\Delta \tau), (\Delta t), and (\Delta x) are constrained and checked,
* discuss how budget accumulation and the no-skip rule lead to observed relativistic behaviour in test scenes.

Those passages make explicit how special relativity is implemented combinatorially.

**Diagnostics and audits**

When this bridge mentions:

* **diagnostics** (L1/L2/L3 statistics, band occupancy, tie-rates, redshifts, delays, deflections, etc.),
* **audits** (curve-ban checks, no-skip/locality checks, isotropy, no-signalling, SR-compatibility),

look into the V2 material where you:

* define the **diagnostic outputs** the engine produces (CSV logs, histograms, panels),
* describe the **audit tests** the engine runs automatically or that you run as part of analysis,
* specify thresholds and accept/reject rules for calling a run “clean” versus “diagnostic only.”

This is where the engine makes its behaviour measurable and checkable against the theory’s requirements.

As you refine V2, this subsection can act as a **topic-level map**: whenever a piece of the bridge mentions an engine concept, you can refer here to quickly find the corresponding part of the Present-Act Engine specification, ensure naming and definitions match, and update both the engine and the bridge in sync.

**10.3 Closing Summary**

This bridge document has one core message: **there is only one theory here**. What you previously called “V1” and “V2” are not two competing frameworks; they are **two faces of the same present-act, relational structure**:

* V1 is the **formal core**: a precise ontology of presents, IN/ON structure, carriers, context ladder, dimension profile, pivot function, invariant interval, master action, and a relational account of quantum and gravitational phenomena.
* V2 is the **engine realization**: a concrete, combinatorial present-act engine with sites (k), world and qualia records, finite feature alphabet, hinge equality, boolean/ordinal gates, ratio-lex acceptance, PF/Born ties-only, typed budgets, manifest, and audits.

This bridge has:

* **Mapped the ontology**: carriers and PMS ↔ ((W\_k, Q\_k)); IN/ON ↔ inward/outward read; abstract state (h\_k) ↔ feature assignments in a finite alphabet; context ladder (n) ↔ the six-band ladder (−2 … +3) pinned to physical scales; implicit dynamic roles ↔ L1/L2/L3.
* **Mapped the operator algebra to the engine cycle**: Renew ↔ enumeration; Sink ↔ acceptance and commit; Trade, Sync, framing ↔ hinge equality plus Θ, κ, structural, ParentGate, and CRA gates; admissibility predicates ↔ gates; ledger and arrow of time ↔ irreversible commit and growing histories.
* **Mapped geometry and scale into the manifest**: abstract (D(n)) and (g(D)) ↔ band definitions, gate ladders, ParentGate schedules; boundary projector ↔ hinge equality; Present Plane and complex structure ↔ phase bins and tie adjacency.
* **Mapped emergent physics**: invariant interval ↔ typed budgets and no-skip composition; structural Born rule ↔ PF/Born ties-only; pivot-driven gravity ↔ ParentGate feasibility gradients; gauge structure ↔ discrete tags and structural gate rules.
* **Linked dynamics and quantization**: master action and path-sum ↔ local gate and residual preferences plus engine sampling over histories.
* **Clarified phenomenology**: space as outward context hierarchy; matter as stable engine patterns across bands; qualia as the Q-stream and its equivalence classes; all three as different reads of the same present-act.
* **Elevated engine constraints to principles**: finiteness and discreteness, locality and no-skip, curve-ban, manifest and audits are not arbitrary implementation choices but **operational forms of the theory’s ontological commitments**.

This means that, going forward:

* When you **work in V1**, you are specifying the **structural constraints** that any valid engine must realize.
* When you **work in V2**, you are designing and tuning an engine that **faithfully instantiates** those constraints under a strict combinatorial discipline.
* When you **update either**, this bridge is your **contract** for keeping them aligned: every concept should have a place on both sides, and any change should be able to be traced through the mappings here.

In practical terms, this bridge now gives you:

* A **shared vocabulary** for talking about the theory – you can move fluidly between abstract notions (carriers, ladder, action, Present Plane) and engine notions (records, bands, gates, budgets, PF/Born).
* A **reference backbone** for the next stages:
  + updating your V1 reference to use the same clarified ontology and naming,
  + updating your V2 engine spec to explicitly cite the corresponding V1 structures,
  + crafting a concise **defensive publication** that presents the core framework and references this longer bridge and the full theory archive (for example, via a blockchain timestamp).

The final picture is this:

* **One present-act engine** at the core, described in two complementary languages.
* **One hinge** where geometry, engine, and experience meet (UGM / D=2 / specious present).
* **One relational ontology** that yields space, matter, and qualia as three faces of the same underlying process.

Everything else – simulations, empirical tests, future elaborations – sits on top of that unity. This bridge document is here so you do not have to re-derive that unity every time you extend, explain, or defend the theory.