

Reasoning Substrate:

A Model-Agnostic Deterministic Reasoning Cycle

RS v1 and RS v2

A Technical White Paper

Version 1.2 | September 2026

Classification: Public Technical Reference

Document Series: Reasoning Substrate Architecture

Revision Cycle: RS v1 (Foundational) · RS v2 (Generalized)

Revision Note: v1.2 corrects the cycle formalization in Sections 3-12 to match the verified four-operator implementation (Limitation, Generation, Collapse, FixedPoint, governed by a termination check). Earlier versions described a six-step / seven-component cycle that was not the version implemented or tested; this revision replaces that description throughout.

Table of Contents

Opening Statement

1. Executive Summary

2. Motivation and Problem Statement

3. RS v1 Overview

4. The RS v1 Cycle

4.1 Step 1 — Limitation Operator

4.2 Step 2 — Generation Operator

4.3 Step 3 — Collapse Operator

4.4 Step 4 — FixedPoint Operator

4.5 Termination Check

5. Operators and State: Component Reference

6. RS v1 Software Applications

7. RS v1 $\rightarrow$ RPU Hardware Mapping

8. RS v2 Overview

9. RS v2 Capabilities

9.1–9.9 Capability Extensions

10. Security, Safety, and Auditability

11. Non–Von Neumann Compatibility

12. Future Directions

Closing Statement

AI Summary Page

Opening Statement

The Reasoning Substrate formalizes reasoning itself as a governed, deterministic state-evolution process — not the emergent, statistical behavior of a model, but an explicit, verifiable structure: state transitions, deterministic collapse, identity continuity, and proven convergence. No prior system — symbolic, probabilistic, neural, or hybrid — has combined that formalization with full cross-domain, cross-model, cross-language determinism, fixed-point convergence, bounded reasoning, temporal coherence, and auditable replay in one substrate. RS does. It integrates cleanly with existing architectures, guiding or supporting them to produce reasoning that is not just more stable, but formally verifiable.

1. Executive Summary

The Reasoning Substrate (RS) is a minimal, universal, model-agnostic reasoning cycle that expresses reasoning as a deterministic state-evolution process. It is not a neural network, not a symbolic logic system, and not a probabilistic sampling mechanism. It is a substrate-level control loop that governs reasoning by repeatedly transforming a structured state toward coherence and resolution. The RS defines reasoning at the level of the substrate — the layer beneath any particular model, agent architecture, or application domain — thereby providing a universal foundation upon which heterogeneous reasoning systems can be constructed, wrapped, or extended.

The RS exists in two versions. RS v1 is the foundational specification: a single-state, single-agent deterministic cycle composed of four formally defined operators — Limitation, Generation, Collapse, and FixedPoint — governed by a termination check that determines whether the cycle concludes or continues. RS v2 is the generalization of RS v1: a multi-state, multi-agent, hierarchical, distributed reasoning environment that extends the v1 cycle across coordinated substrate instances. RS v2 does not replace RS v1; it supersedes the single-agent constraint and introduces capabilities for coordinated, adaptive, and hierarchically structured substrate operation.

The RS delivers verifiable benefits across the following categories. Determinism ensures that the state-evolution process produces consistent outputs given identical inputs. Auditability ensures that every state, candidate set, and collapse event is inspectable throughout the cycle. Replayability guarantees that identical inputs produce identical reasoning trajectories. Injection resistance is achieved by positioning the substrate as the governing layer — above the prompt — such that adversarial inputs operating at the prompt level cannot redirect the substrate-level control loop. Fail-safe behavior ensures that when the termination check does not confirm convergence within the defined bounds, the system performs a controlled safe return with no runaway loops. Power efficiency is realized by constraining and selecting among candidates before any state update is committed, eliminating retries, flailing, and wasted compute. Hardware mapping to the Reasoning Processing Unit (RPU) enables native substrate execution on non-Von Neumann hardware architectures.

This white paper provides the complete formal specification of the RS, organized as follows. Section 2 establishes the motivation and problem statement. Section 3 introduces RS v1 at an architectural level. Section 4 presents the precise formal cycle definition. Section 5 provides a component reference for the four operators and the termination check. Sections 6 and 7 address software applications and RPU hardware mapping, respectively. Sections 8 and 9 present RS v2 and its nine capability extensions. Sections 10 and 11 address security, safety, auditability, and non-Von Neumann compatibility. Section 12 outlines future directions derived from the RS foundation. The document closes with a statement on the role of RS as a universal reasoning substrate.

2. Motivation and Problem Statement

Prior work has formalized aspects of governed state-evolution. Dynamical systems theory established the mathematical treatment of stability, convergence, and invariant preservation over a century ago. Hopfield-era energy-based computation applied that treatment directly to neural computation in 1982, proving convergence to a stable fixed point via an explicit energy function. More recent work — Equilibrium Propagation among it — extends this lineage to modern network training. Current agent-governance frameworks apply comparable formal rigor to the narrower problem of bounding AI agent actions. Each of these addresses a real piece of this problem, within a single domain — neural computation, or agent governance, but not both, and not beyond them.

What has not been shown is one unmodified substrate — the same formal cycle, the same operators — verified across domains that share no structural relationship to each other: AI reasoning, physical system control, financial-signal damping, and hardware execution alike. This is the specific, narrow gap RS addresses. Contemporary AI systems and agent frameworks share a structural deficiency: the reasoning behavior of these systems is typically governed by the model itself, the prompt that drives the model, or an application-level orchestration layer — none of which constitutes a substrate-level control loop generalized across domains. Without a formally defined substrate that specifies how state is constrained, how candidates are generated and selected, and how the system confirms convergence, reasoning behavior is non-deterministic at the architectural level, non-auditable in practice, and prone to systematic failure modes including runaway loops, prompt injection, and flailing — the expenditure of compute on repeated, unproductive retries without convergence.

The absence of replayability is a direct consequence of the absence of a defined state-evolution process. When reasoning behavior is governed by a prompt and a model rather than by a formal substrate cycle, identical inputs do not guarantee identical reasoning trajectories. The system may arrive at different intermediate states, select different actions, and produce different outputs on successive invocations. This property — the inability to reproduce a reasoning trajectory exactly — makes verification, debugging, and validation structurally impossible in the general case. A substrate that formally defines state evolution eliminates this deficiency by construction.

The absence of auditability compounds the problem. Without inspectable records of state, candidates, and collapse events at each cycle step, the reasoning behavior of an AI system cannot be verified or traced after the fact. Debugging becomes dependent on log inspection at the application layer, which captures effects rather than causes. Compliance with safety or governance requirements is similarly impeded. A substrate that produces a complete, structured record of every reasoning step — at the substrate level rather than the application level — provides auditability as a structural property rather than as an add-on capability.

The absence of injection resistance is a consequence of the absence of a governing substrate. When a prompt governs the model, adversarial inputs operating at the prompt level have direct access to the reasoning-control mechanism. Prompt injection — the redirection of model behavior through adversarially crafted inputs — is possible precisely because there is no substrate layer above the prompt that enforces the reasoning cycle. A substrate that positions itself as the governing layer, with the model operating as a component within the substrate cycle rather than as the governing element, eliminates the prompt-level attack surface as a control pathway.

The need for a substrate that is domain-agnostic, language-agnostic, and compatible with both software and non-Von Neumann hardware architectures is driven by the increasing heterogeneity of deployment environments. Reasoning systems are deployed across domains — natural language, code generation, planning, control, and scientific reasoning — and across hardware environments that include conventional CPUs, GPUs, and next-generation non-Von Neumann processing architectures such as the RPU. A substrate defined at the abstract cycle level, with formal mappings to hardware constructs, satisfies this requirement without imposing domain or language constraints on the systems it governs.

RS v1 and RS v2 are introduced as the solution to the foregoing deficiencies. RS v1 defines the minimal, universal substrate cycle as a formal deterministic state-evolution process. RS v2 generalizes this cycle to multi-state, multi-agent, hierarchical, and distributed reasoning environments. Together, they constitute a complete substrate specification that addresses determinism, auditability, replayability, injection resistance, fail-safe behavior, power efficiency, and hardware compatibility as structural properties of the reasoning substrate itself.

3. RS v1 Overview

RS v1 is a minimal, universal, model-agnostic reasoning cycle. Its defining characteristic is that it expresses reasoning not as a property of a model or a prompt, but as a formal state-evolution process: a deterministic sequence of operations that transforms a structured state at each cycle step. By locating the governance of reasoning at the substrate level — below the model, below the prompt, and below the application — RS v1 establishes a reasoning foundation that is independent of any particular AI architecture, programming language, or application domain.

RS v1 is explicitly not a neural network, not a symbolic logic system, and not a probabilistic sampling mechanism. These distinctions are definitional. Neural networks are parameter-governed function approximators; symbolic logic systems are rule-governed inference engines; probabilistic samplers generate outputs by drawing from learned or specified distributions. RS v1 is none of these. It is a substrate-level control loop that governs reasoning by repeatedly applying a defined cycle — constrain, generate, collapse, and evaluate convergence — until the state achieves stability or the fail-safe termination condition is triggered. Any of the foregoing mechanisms may serve as an implementation detail within a given operator, but none of them constitutes the substrate itself.

RS v1 is architecture-agnostic and domain-agnostic. It can wrap an existing AI model, run inside an agent framework, or run alongside a legacy system. The substrate does not prescribe what the generation mechanism computes or what domain the state represents. It prescribes only the cycle structure: how the state is constrained, how candidates are generated, how the best candidate is selected by the collapse operator, and how convergence is evaluated. This structural agnosticism makes RS v1 deployable without modification across natural language reasoning, code synthesis, planning, and control domains.

The determinism, replayability, and auditability of RS v1 are structural properties of the cycle definition. Because every operator of the cycle is formally defined — Limitation, Generation, Collapse, and FixedPoint, together with the termination check that governs repetition — every invocation of the cycle with identical inputs produces identical intermediate states and identical outputs. Every intermediate state, candidate set, and collapse event is inspectable. These properties are not implementation choices; they are consequences of the formal cycle definition.

RS v1 consists of four formally defined operators, together with a termination check that governs repetition of the cycle: LIMITATION, GENERATION, COLLAPSE, and FIXEDPOINT, operating on a structured STATE and governed by a TERMINATION CHECK. These components are described in architectural overview in this section, presented as a formal cycle definition in Section 4, and documented as a component reference in Section 5.

4. The RS v1 Cycle

The RS v1 cycle is defined as a formal deterministic state-evolution process. Let state denote the structured state at the current cycle iteration. The cycle proceeds through four operators at each iteration, followed by a termination check, repeating until the termination check confirms convergence. The pseudocode that follows defines the abstract substrate cycle only. It does not specify the internal mechanisms, hardware primitives, or proprietary implementation details of any deployment.

4.1 Step 1 — Limitation Operator

The Limitation Operator establishes constraints, boundaries, or scoping rules that define the permissible reasoning space for the current cycle. It takes the current state as input and produces a limited state as output. The Limitation Operator acts before any candidate is generated, so the constraint is structural — candidates outside the permissible space are never produced, not filtered afterward.

4.2 Step 2 — Generation Operator

The Generation Operator produces one or more candidate outputs within the boundaries established by the Limitation Operator. The generation mechanism may be neural, symbolic, hybrid, or any other implementation; the abstract cycle does not constrain how candidates are produced, only that they are produced within the limited state's boundaries.

4.3 Step 3 — Collapse Operator

The Collapse Operator performs a controlled reduction, selection, ranking, merging, or consolidation of the candidate outputs, producing a single collapsed output. This is a designed operator, not an emergent behavior: the collapse is deterministic given identical candidates and an identical selection mechanism, and requires no sampling.

4.4 Step 4 — FixedPoint Operator

The FixedPoint Operator evaluates whether the collapsed output satisfies a convergence condition. Convergence may be semantic, structural, logical, or numerical, depending on the deployment. The FixedPoint Operator produces both a convergence verdict and the state that will carry forward — either the converged output, or an updated state for the next iteration.

4.5 Termination Check

```
if convergence.is_converged = TRUE then      return convergence.output else      state :=  
convergence.output end if
```

The termination check reads the verdict produced by the FixedPoint Operator. If convergence is confirmed, the cycle concludes and returns the converged output. Otherwise, the FixedPoint Operator's output becomes the new state, and the cycle repeats from the Limitation Operator. The termination check is loop-control logic that acts on FixedPoint's result — it is not itself a transforming operator, and does not constitute a fifth stage of the cycle in the sense that Limitation, Generation, Collapse, and FixedPoint do.

A secondary, fail-safe condition may also trigger termination independent of convergence — for instance, a bound on iteration count or resource use defined at the deployment level. When the fail-safe triggers, the cycle performs a controlled safe return rather than continuing to iterate, eliminating the possibility of runaway loops.

Scope Note: The foregoing pseudocode defines only the abstract substrate cycle: four operators and a termination check. It does not describe the internal mechanisms, hardware primitives, domain-specific generation

or collapse implementations, or proprietary logic of any specific RS deployment. The cycle structure is fixed; the operator implementations are substrate-agnostic by design.

5. Operators and State: Component Reference

The following section provides a structured technical reference for the state and each of the four formally defined operators of the RS v1 cycle, together with the termination check that governs repetition. Each entry specifies the component's role within the cycle and its input-output signature as derived from the definitions in Section 4.

1. STATE

- Role: The structured state is the central data object of the RS cycle. It encodes the complete representation of the reasoning context at a given point in the cycle.
- Input: Initialized from the problem context; updated at each cycle iteration by the FixedPoint Operator's output when convergence is not yet confirmed.
- Output: The current state, consumed first by the Limitation Operator.
- Notes: The state is structured and inspectable at every cycle iteration. Its internal representation is not fixed by the abstract cycle definition.

2. LIMITATION OPERATOR

- Role: Establishes constraints, boundaries, or scoping rules defining the permissible reasoning space for the current cycle iteration.
- Input: The current state.
- Output: A limited state, reflecting the applicable constraints and invariants.
- Notes: Limitation acts before generation, not as a filter afterward. The mechanism by which constraints are defined and enforced is not fixed by the abstract cycle definition.

3. GENERATION OPERATOR

- Role: Produces one or more candidate outputs within the boundaries defined by the Limitation Operator.
- Input: The limited state.
- Output: A set of one or more candidates.
- Notes: The generation mechanism may be neural, symbolic, hybrid, or any other implementation. The abstract cycle does not constrain how candidates are produced.

4. COLLAPSE OPERATOR

- Role: Performs a controlled reduction, selection, ranking, merging, or consolidation of the candidate set, producing a single collapsed output.
- Input: The candidate set produced by Generation.
- Output: A single collapsed output.
- Notes: The collapse is deterministic given identical candidates and an identical selection mechanism. It is a designed operator, not an emergent behavior.

5. FIXEDPOINT OPERATOR

- Role: Evaluates whether the collapsed output satisfies a convergence condition, and produces the state that carries forward.
- Input: The collapsed output.
- Output: A convergence verdict, and either the converged output or an updated state for the next iteration.

- Notes: Convergence may be semantic, structural, logical, or numerical, depending on implementation. The threshold or condition for convergence is defined by the deployment context and is not fixed by the abstract cycle definition.

6. TERMINATION CHECK

- Role: Reads the verdict produced by the FixedPoint Operator and determines whether the cycle concludes or repeats.
- Primary condition: `convergence.is_converged = TRUE` — the cycle concludes and returns the converged output.
- Secondary condition: a fail-safe bound (e.g., iteration count, resource limit) defined at the deployment level — the cycle performs a controlled safe return independent of convergence.
- Notes: The termination check is loop-control logic, not a transforming operator. It prevents runaway loops by guaranteeing the cycle always concludes, either by convergence or by fail-safe.

6. RS v1 Software Applications

RS v1 is model-agnostic in the most operationally complete sense: it can wrap an existing AI model, run inside an agent framework, or run alongside any legacy system without modification to the substrate cycle itself. The model, the agent framework, or the legacy system operates as a component within the RS cycle — specifically, as a potential implementation of the Generation Operator — rather than as the governing layer. This architectural positioning means that RS v1 does not depend on the capabilities, architecture, or version of the underlying model, and can be applied uniformly across a heterogeneous ecosystem of AI systems.

RS v1 is language-agnostic in both the programming language and the natural language sense. The formal cycle definition — state, Limitation, Generation, Collapse, FixedPoint, termination check — is expressed at the abstract level and does not presuppose any particular programming language implementation or any particular natural language as the medium of reasoning. This property allows RS v1 to be implemented in any programming language and applied to reasoning tasks expressed in any natural language or formal notation, without modification to the substrate specification.

RS v1 provides a universal compilation mechanism for legacy systems. A legacy system — defined as any existing system with deterministic or semi-deterministic behavior — can be compiled into a set of structured operators that conform to the RS v1 component interface. Specifically, the legacy system's state representation can be mapped to the RS STATE component, its permitted-action rules can be mapped to the LIMITATION component, and its behavioral or output-generating logic can be mapped to the GENERATION component. This compilation does not require modification of the legacy system itself; it requires only the definition of the mapping between the legacy system's constructs and the RS component interface.

Injection resistance is a structural property of the RS v1 architecture. Because RS v1 positions the substrate as the governing layer — above both the model and the prompt — adversarial inputs operating at the prompt level cannot redirect the substrate-level control loop. The substrate cycle determines what candidates are even reachable (via Limitation), which candidate is selected (via Collapse), and when the cycle concludes (via the termination check acting on FixedPoint's verdict). None of these operations is accessible to a prompt-level input. The prompt, and any adversarial content it may contain, is treated as data that the substrate processes through its operators, not as a directive that governs the substrate itself.

Power efficiency is a quantifiable benefit of the RS v1 architecture. In systems without a substrate-level constraint-and-collapse step, candidate actions may be applied sequentially or heuristically, with failures triggering retries. This pattern — commonly described as flailing — expends compute on state updates that do not advance the reasoning process toward resolution. RS v1 eliminates flailing by constraining the reachable space before generation and by evaluating candidates via Collapse before any state update is committed, with the result that every state update is a productive step toward stability or resolution.

Stability and drift correction are provided continuously by the RS v1 cycle structure. Because the cycle repeats until the termination check confirms convergence, any drift away from coherence at one iteration is automatically subject to correction at the next. Limitation, Generation, Collapse, and FixedPoint together constitute a continuous correction mechanism: each iteration constrains and selects the candidate most consistent with the governing constraints, driving the actual state progressively toward coherence. This mechanism operates without explicit drift-detection logic at the application layer; it is a structural consequence of the cycle definition.

Graceful fail-safe behavior is provided by the secondary termination condition described in Section 4.5. If the fail-safe is triggered — by conditions defined at the deployment level, such as iteration count exceeding a bound, resource exhaustion, or an externally imposed halt — the substrate performs a controlled safe return. No further state updates are applied after the fail-safe triggers. This behavior eliminates the possibility of runaway loops, in which a system continues to cycle indefinitely without converging.

Full auditability and replayability are direct consequences of the RS v1 cycle structure. Every state, every candidate produced by Generation, and every collapse result produced by Collapse is defined at the substrate level and is therefore inspectable without recourse to application-layer logging. Replayability follows from determinism:

because the cycle is formally defined and the Collapse Operator is deterministic, identical inputs produce identical reasoning trajectories at every iteration, enabling exact reproduction of any prior reasoning session.

7. RS v1 → RPU Hardware Mapping

The components of the RS v1 cycle map directly to discrete hardware constructs in the Reasoning Processing Unit (RPU) architecture. This mapping is conceptual: it defines the correspondence between the abstract substrate cycle and RPU hardware constructs, without specifying the proprietary implementation details of any particular RPU design. The mapping is as follows: STATE maps to physical register space; LIMITATION maps to a constraint-enforcement circuit; GENERATION maps to a candidate-generation hardware path; COLLAPSE maps to a deterministic selection circuit; FIXEDPOINT maps to a convergence-evaluation circuit; and the CYCLE, including the termination check, maps to a native hardware loop.

RS v1 Component	RPU Hardware Construct	Function
STATE	Physical register space	Stores the structured state at each cycle iteration
LIMITATION	Constraint-enforcement circuit	Applies boundaries before candidates are generated
GENERATION	Candidate-generation hardware path	Produces candidates within the limited state
COLLAPSE	Deterministic selection circuit	Selects a single candidate without sampling
FIXEDPOINT	Convergence-evaluation circuit	Evaluates the collapsed output against convergence conditions
TERMINATION CHECK + CYCLE	Native hardware loop	Reads the convergence verdict; concludes or repeats the cycle at hardware speed

The significance of this mapping is that it enables native substrate execution on non-Von Neumann architectures. Von Neumann architectures process instructions sequentially through a fetch-decode-execute pipeline; this pipeline is not the natural execution model for a substrate that repeatedly applies a fixed set of operators to a structured state. The RPU, as a non-Von Neumann architecture, implements the substrate cycle as a native hardware loop: the cycle does not need to be decomposed into sequential instruction fetches and dispatches because the hardware is structured around the cycle itself. The result is that the substrate cycle executes at hardware speed without the overhead of the fetch-decode-execute pipeline.

The deterministic selection circuit that implements the Collapse Operator is of particular architectural significance. In software implementations, selecting the best candidate requires a sequential or parallel comparison, typically implemented as a reduction operation. In the RPU, the deterministic selection circuit performs this operation as a native hardware primitive, asserting the selected candidate in a single hardware operation. This implementation preserves the determinism property of Collapse — because the circuit is combinatorial and stateless with respect to the selection operation, identical candidate inputs always produce identical selections — while executing at the speed of hardware logic rather than software comparison.

The convergence-evaluation circuit that implements the FixedPoint Operator interacts directly with the physical register space that represents the STATE component. This direct register-level interaction eliminates the memory-access latency that would be incurred in a Von Neumann architecture, in which a convergence check and state update would each require separate memory loads and stores. The net effect is that the RS v1 cycle maps to the RPU hardware loop with each iteration corresponding to a small number of native hardware operations, enabling substrate execution at a level of efficiency that is architecturally inaccessible to conventional Von Neumann implementations.

8. RS v2 Overview

RS v2 is the generalized substrate. It extends RS v1 from a single-state, single-agent substrate into a multi-state, multi-agent, hierarchical, and distributed reasoning environment. The extension is strictly a generalization: RS v1 is a special case of RS v2 in which the number of states is one, the number of agents is one, the hierarchy is flat, and the substrate operates in a non-distributed mode. RS v2 does not modify the RS v1 cycle definition; it extends the operating context within which the cycle operates and introduces nine additional capability dimensions that are absent in RS v1.

RS v2 introduces the following extensions to the RS v1 foundation: multi-state substrates that support coordinated reasoning across multiple simultaneous structured states; multi-agent substrates that support coordinated reasoning across multiple simultaneous agents, each operating within the RS cycle; hierarchical reasoning layers that enable hierarchical planning and collapse across layered substrate instances; distributed substrate behavior that maintains stability and coherence across distributed compute environments; multi-objective evaluation that replaces a single scalar criterion within Collapse and FixedPoint with a multi-objective formulation; adaptive operator sets that allow substrate operators to be reconfigured in response to observed behavior; substrate-level learning that allows adaptation to occur at the substrate level rather than at the model level; substrate-to-substrate communication that enables coordinated behavior across distinct RS instances; and RPU-native execution patterns that leverage hardware-accelerated substrate primitives.

The motivation for RS v2 derives directly from the limitations of the single-state, single-agent constraint of RS v1. In RS v1, the substrate governs a single reasoning process operating on a single structured state. This is sufficient for a large class of reasoning tasks. However, tasks that require coordinated planning across multiple agents, hierarchical decomposition of complex reasoning objectives, distribution of reasoning across heterogeneous compute environments, or adaptive modification of substrate behavior in response to observed patterns require capabilities beyond the RS v1 specification. RS v2 provides these capabilities as formal extensions of the v1 cycle, maintaining backward compatibility while enabling a significantly broader class of reasoning deployments.

The RPU hardware mapping of RS v2 extends the RS v1 mapping in direct correspondence with the extended capabilities. Multi-state substrates map to multi-register banks, enabling multiple structured states to be maintained simultaneously in hardware register space. Hierarchical substrates map to layered hardware loops, enabling hierarchical reasoning to be executed as a hierarchy of native hardware loops rather than as a software-level orchestration layer. Distributed substrates map to a multi-core substrate mesh, enabling distributed substrate behavior to be implemented as a hardware-level mesh of RPU cores. Adaptive operators map to reconfigurable hardware primitives, enabling operator reconfiguration to be performed at the hardware level without incurring software-level overhead.

RS v2 represents the complete substrate specification for reasoning systems that require coordination, hierarchy, distribution, and adaptation as structural properties. Together with RS v1, it constitutes a two-tier substrate architecture in which the foundational cycle (v1) is universally applicable and the generalized substrate (v2) is available when the task requires capabilities beyond the single-state, single-agent constraint. The two tiers share a common formal foundation: the same state-evolution cycle, the same four operators and termination check, and the same formal properties of determinism, auditability, replayability, injection resistance, fail-safe behavior, and hardware mapping.

9. RS v2 Capabilities

The following subsections provide a structured technical description of each of the nine capability extensions introduced in RS v2. Each capability is derived strictly from the RS v1 foundation and the RS v2 extension definitions. No external frameworks or external terminology are introduced.

9.1 Multi-State Substrates

A multi-state substrate maintains and evolves multiple simultaneous structured states, each subject to its own RS cycle, with coordination occurring across the state set. Coherence between states is maintained by cross-state terms considered within Collapse, which penalize candidates whose selection would reduce coherence between the states. In the RPU hardware mapping, multi-state substrates correspond to multi-register banks, enabling simultaneous hardware-level maintenance of multiple structured states.

9.2 Multi-Agent Substrates

A multi-agent substrate coordinates reasoning across multiple simultaneous agents, each operating within its own RS cycle and governed by the substrate. Each agent maintains its own state and executes its own Limitation, Generation, Collapse, and FixedPoint operators; coordination is achieved through shared or overlapping constraints and selection criteria. The substrate governs the agents collectively, ensuring that the aggregate reasoning behavior satisfies system-level coherence and resolution criteria rather than only agent-level criteria.

9.3 Hierarchical Reasoning Layers

Hierarchical reasoning layers extend the RS cycle into a layered structure in which higher-level substrate instances govern the constraints and termination conditions of lower-level substrate instances. The higher-level substrate performs generation and collapse over coarse-grained state representations; the selected higher-level candidate constrains the space available to lower-level substrates, which perform generation and collapse over fine-grained state representations. In the RPU hardware mapping, hierarchical substrates correspond to layered hardware loops, enabling hierarchical reasoning to execute as a native hierarchy of hardware cycles.

9.4 Distributed Substrate Behavior

Distributed substrate behavior extends the RS cycle across multiple compute nodes, each executing a subset of the cycle's operators on a partition of the structured state. Stability and coherence are maintained across the distributed environment through inter-node communication of candidate and collapse information, ensuring that the aggregate state evolution satisfies the global termination condition. In the RPU hardware mapping, distributed substrates correspond to a multi-core substrate mesh, enabling distribution as a hardware-level topology rather than a software-level message-passing layer.

9.5 Multi-Objective Evaluation

In RS v1, Collapse and FixedPoint operate against a single evaluative criterion. In RS v2, this is extended to a vector-valued formulation, where each dimension corresponds to a distinct objective. Collapse is correspondingly extended to perform multi-objective selection, choosing the candidate that achieves the best trade-off across all objectives according to a defined preference ordering or Pareto-optimality criterion.

9.6 Adaptive Operator Sets

Adaptive operator sets allow the operators of the RS cycle — Limitation, Generation, Collapse, and FixedPoint — to be reconfigured by the substrate in response to observed reasoning behavior. Reconfiguration occurs at the

substrate level, not the model level: the operators themselves are modified, not any model that may serve as part of an operator's implementation. In the RPU hardware mapping, adaptive operators correspond to reconfigurable hardware primitives, enabling operator reconfiguration to be performed as a hardware-level operation.

9.7 Substrate-Level Learning

Substrate-level learning is the mechanism by which the RS substrate adapts its operators based on observed cycle behavior, without delegating adaptation to the underlying model. Learning at the substrate level means that Limitation's constraints, Generation's candidate distribution, or Collapse's selection criteria may be updated based on the history of state trajectories and outcomes across prior cycle invocations. Substrate-level learning is categorically distinct from model-level learning, which updates the parameters of a model; substrate-level learning updates the operators that govern the cycle itself.

9.8 Substrate-to-Substrate Communication

Substrate-to-substrate communication enables distinct RS instances — each executing its own cycle on its own structured state — to exchange state information, candidates, or collapse results in a defined communication protocol. Communication between substrates allows coordinated multi-substrate reasoning without collapsing the substrates into a single multi-state substrate: each substrate retains its own cycle, its own termination check, and its own state; communication occurs at defined synchronization points between cycles. This capability supports the construction of reasoning networks composed of peer RS instances.

9.9 RPU-Native Execution Patterns

RPU-native execution patterns are substrate-level primitives that are specifically designed for hardware-accelerated execution on the RPU. These patterns expose the RS cycle operators — Limitation, Generation, Collapse, FixedPoint — as hardware-callable primitives that execute at the speed of the RPU's native hardware loop rather than as software-layer function calls. RPU-native execution patterns in RS v2 extend the RS v1 RPU mapping to the full set of v2 capabilities, including multi-register bank access for multi-state substrates, layered hardware loop execution for hierarchical substrates, and multi-core mesh routing for distributed substrates.

RS v2 Capability	RPU Hardware Mapping
Multi-State Substrates	Multi-register banks
Hierarchical Substrates	Layered hardware loops
Distributed Substrates	Multi-core substrate mesh
Adaptive Operators	Reconfigurable hardware primitives

10. Security, Safety, and Auditability

Injection resistance is a structural property of the RS architecture that holds independently of the underlying model, language, or domain. Because the RS cycle governs the model rather than the reverse — the model operates as a component within the cycle, not as the cycle's governing layer — adversarial inputs at the prompt level cannot redirect the substrate-level control loop. A prompt-level adversarial input may influence the content of candidates produced by Generation, but the reachable space for those candidates is bounded by Limitation before Generation runs, and the selection of which candidate is used is determined entirely by Collapse. The adversarial input cannot modify Limitation's constraints, Collapse's selection mechanism, or the termination check through the prompt channel.

Auditability is provided as a complete structural record of reasoning behavior at the substrate level. Every state, every candidate produced by Generation, and every collapse result produced by Collapse is defined at the substrate level and is therefore available for inspection without recourse to application-layer logging. This structural auditability covers the full causal chain of reasoning behavior: from the initial state through every intermediate state, candidate set, and selection, to the final state at termination. No step in the reasoning process is opaque to the substrate-level audit record.

Replayability is a direct consequence of the deterministic state-evolution process. Because the RS cycle is formally defined and the Collapse Operator selects deterministically — without sampling or stochastic selection — identical inputs to the cycle produce identical outputs at every iteration. This property holds across all four operators of the cycle and through every iteration from initialization to termination. Replayability enables exact reproduction of any prior reasoning session: given the initial state and the Generation and Collapse mechanisms used in a prior session, the substrate produces an identical reasoning trajectory without deviation.

Fail-safe behavior is defined by the secondary termination condition described in Section 4.5. The fail-safe mechanism is triggered by conditions defined at the deployment level — iteration count bounds, resource limits, or externally imposed halt signals — that are independent of the convergence condition evaluated by FixedPoint. When the fail-safe is triggered, the substrate does not apply further state updates and does not enter the next cycle iteration; it performs a controlled safe return, producing the current state as the final output. Runaway loops — cycles that execute indefinitely without convergence — are structurally precluded by this specification.

The security, safety, and auditability properties described in this section hold regardless of the underlying model, language, or domain in which the RS substrate is deployed. They are not dependent on the internal implementation of Generation or Collapse. They are properties of the cycle structure itself, and they are preserved in RS v2 across all nine capability extensions: multi-state, multi-agent, hierarchical, distributed, multi-objective, adaptive, substrate-learning, substrate-communicating, and RPU-native deployments are all governed by the same formal cycle definition, and all retain the structural security, safety, and auditability properties of RS v1.

11. Non–Von Neumann Compatibility

RS v1 and RS v2 are compatible with non-Von Neumann hardware architectures by virtue of the cycle structure of the substrate definition. The Von Neumann model of computation is defined by a stored-program architecture in which a processor sequentially fetches instructions from memory, decodes them, and executes them — a model in which the program and the data occupy the same memory space and are accessed through the same memory bus. The RS cycle is not structured as a sequence of stored instructions; it is structured as a fixed set of operators applied repeatedly to a structured state. This structural difference means that the RS cycle does not require the fetch-decode-execute pipeline, and therefore does not require a Von Neumann architecture for its execution.

The RS cycle maps naturally to hardware that implements the cycle as a native loop rather than as a sequence of instruction fetches. The components of the cycle — STATE, LIMITATION, GENERATION, COLLAPSE, FIXEDPOINT, and the CYCLE itself — correspond to discrete hardware constructs in the RPU architecture: physical register space, a constraint-enforcement circuit, a candidate-generation hardware path, a deterministic selection circuit, a convergence-evaluation circuit, and a native hardware loop, respectively. Each of these constructs can be implemented as a dedicated hardware unit without requiring the Von Neumann instruction-memory model. The result is a substrate cycle that executes as a hardware-native loop, with each operator implemented as a hardware-primitive operation rather than as a software-level function call dispatched through a fetch-decode-execute pipeline.

In RS v2, the multi-core substrate mesh extends non-Von Neumann compatibility to distributed substrate deployments. The multi-core substrate mesh is a hardware topology in which multiple RPU cores, each executing its own substrate cycle, are connected by a mesh routing fabric that supports substrate-to-substrate communication at the hardware level. This architecture enables distributed substrate behavior — coordinated reasoning across multiple compute nodes — without requiring a Von Neumann memory bus or a shared instruction stream. Each core operates its own cycle independently, with coordination achieved through the mesh routing fabric rather than through shared memory access. This architecture is particularly well-suited to next-generation compute environments in which heterogeneous, distributed hardware is the norm rather than the exception.

The non-Von Neumann compatibility of RS v1 and RS v2 positions the Reasoning Substrate as a suitable foundation for next-generation hardware architectures in which the conventional fetch-decode-execute model is supplanted by purpose-built reasoning hardware. As RPU architectures mature and as the hardware-software boundary in AI systems shifts toward increasingly purpose-built compute, the RS cycle's structural alignment with hardware-native loop execution becomes an increasingly significant architectural advantage. The substrate specification is defined at an abstraction level that is simultaneously above any particular hardware implementation and directly mappable to hardware constructs, enabling the RS cycle to track advances in non-Von Neumann hardware without requiring revision of the abstract substrate definition.

12. Future Directions

Substrate-level learning, introduced in RS v2, represents the most consequential direction for future development of the Reasoning Substrate. In its RS v2 formulation, substrate-level learning enables the operators of the cycle — Limitation, Generation, Collapse, and FixedPoint — to be updated based on observed cycle behavior, without delegating adaptation to the underlying model. Future work in this direction concerns the formal specification of substrate learning rules: what constitutes a valid substrate-level update, how update stability is guaranteed across successive cycle invocations, and how substrate-level learning interacts with the determinism and replayability properties that are structural guarantees of the RS cycle. A substrate-level learning specification that preserves these properties while enabling genuine adaptation represents a significant open research and engineering challenge.

Substrate-to-substrate communication protocols, introduced in RS v2 as the capability for distinct RS instances to exchange state information, candidates, and collapse results, open a direction for coordinated multi-substrate reasoning networks. In the RS v2 specification, communication between substrates occurs at defined synchronization points between cycles, with the communication protocol not yet formally specified at the abstract cycle level. Future work in this direction concerns the formal definition of substrate communication semantics: what information is exchanged, in what format, at what synchronization points, and under what conditions communication is permitted to influence the receiving substrate's cycle.

Hierarchical substrate stacks, defined in RS v2 as layered substrate instances in which higher-level substrates govern the constraints and termination conditions of lower-level substrates, suggest future work in deep hierarchical planning and collapse across many substrate layers. The RS v2 specification defines the two-level case — a higher-level substrate governing one or more lower-level substrates — but does not formally specify the behavior of substrate stacks with depth greater than two. Future work concerns the formal extension of the hierarchical substrate specification to arbitrary depth, including how constraint information propagates across substrate layers, how termination conditions compose across layers, and how the auditability record is maintained across a deep hierarchical stack.

RPU hardware acceleration is the direction indicated by the RS v2 introduction of RPU-native execution patterns. Future work in this direction concerns the complete hardware specification of the RPU, including the formal definition of the native hardware loop that implements the RS cycle, the deterministic selection circuit that implements Collapse, and the multi-core substrate mesh topology that supports distributed substrates. A fully hardware-accelerated RS substrate — one in which every operator executes as a native hardware primitive without software-layer overhead — represents the terminal state of the RPU hardware mapping trajectory.

Universal legacy compilation, defined in RS v1 as the capability to compile legacy systems into structured operators conforming to the RS component interface, suggests future work in automated operator extraction from existing systems. In the RS v1 specification, this compilation is described as a manual mapping process: a human engineer identifies the legacy system's state representation, permitted-action rules, and behavioral logic, and maps them to the corresponding RS components. Future work concerns the automation of this mapping process: the development of tools and formal procedures that can extract RS-conformant operator definitions from existing systems without manual engineering effort.

Closing Statement

The Reasoning Substrate is defined as a minimal, universal, model-agnostic reasoning cycle that expresses reasoning as a deterministic state-evolution process. It is not a model, not a framework, and not a platform. It is a substrate: the foundational layer beneath all reasoning behavior, governing what reasoning is rather than what it is about. RS v1 provides this foundation for any single-state, single-agent reasoning task, built from four operators — Limitation, Generation, Collapse, FixedPoint — and a termination check. RS v2 extends it to the multi-state, multi-agent, hierarchical, distributed, and hardware-accelerated environments that define the leading edge of AI system deployment. Together, RS v1 and RS v2 constitute a complete substrate specification that provides determinism, auditability, replayability, injection resistance, fail-safe behavior, and hardware compatibility as structural properties of reasoning itself — properties that hold regardless of the model, the language, the domain, or the hardware on which the substrate operates. The Reasoning Substrate is presented as the universal substrate upon which all future reasoning systems may be grounded.

AI Summary Page: Machine-Readable Compressed Summary

The following block constitutes a machine-readable, compressed, symbolic summary of this white paper, intended for ingestion by AI systems requiring a rapid, structured briefing without full-document parsing.

```
RSFAM::CORE={ RSV1:Σ[DetCycle+StateEvo+Limitation+Generation+Collapse+FixedPoint+TerminationCheck
],
RSV2:Σ[MultiState+MultiAgent+Hierarchy+Distributed+MultiObjective+AdaptiveOps+SubstrateLearning+S
ubstrateComm+RPUNative],
Safety:Σ[Determinism+Replay+Audit+InjectionResist+FailSafe+PowerEfficient] }::
CORE_CLAIM="RS_formalizes_reasoning_itself_as_governed_deterministic_state-
evolution,not_emergent_statistical_model_behavior::NO_PRIOR_SYSTEM(symbolic|probabilistic|neural|
hybrid)_has_combined_cross-domain+cross-model+cross-language_determinism+fixed-
point_convergence+bounded_reasoning+temporal_coherence+auditable_replay_in_ONE_substrate::prior_a
rt_acknowledged_explicitly(dynamical_systems_theory,Hopfield_1982,Equilibrium_Propagation,current
_agent-governance_frameworks)_each_addresses_ONE_domain_only,not_combined,not_cross-domain":
RSV1_CYCLE={ Limitation:"limited_state = LIMITATION(state) -- establishes permissible reasoning
space", Generation:"candidates = GENERATION(limited_state) -- produces candidates within bounds",
Collapse:"collapsed_output = COLLAPSE(candidates) -- deterministic reduction/selection",
FixedPoint:"convergence = FIXEDPOINT(collapsed_output) -- evaluates convergence, produces verdict
+ next state", TerminationCheck:"if convergence.is_converged then return convergence.output else
state := convergence.output -- loop control, not a transforming operator" }::
COMPONENTS={ STATE:"structured, inspectable, domain-agnostic", LIMITATION:"active constraint
enforcement BEFORE generation, not passive state property", GENERATION:"candidate production, any
mechanism, bounded by Limitation", COLLAPSE:"deterministic reduction/selection/ranking/merging,
no sampling", FIXEDPOINT:"convergence evaluation, semantic/structural/logical/numerical",
TERMINATION_CHECK:"loop control reading FixedPoint's verdict" }:: CORRECTION_NOTE="v1.2 replaces
the v1.1 six-step/seven-component formalization(STATE->FORWARD_MODEL->PLAN_SET->SCORING_FUNCTION-
->COLLAPSE->APPLY->TERMINATION), which did not match the verified implementation. The four-
operator-plus-termination-check cycle in this version is the real, implemented, tested
architecture.":: FAILURE_MODES_PREVENTED=[ {ID:RS1,NAME:"NonDeterministicReasoning",FM:[model-
governed, prompt-governed],CORR:{RSv1:DetCycle}}, {ID:RS2,NAME:"NoReplay",FM:[stochastic
trajectories],CORR:{RSv1:DeterministicOps}}, {ID:RS3,NAME:"NoAudit",FM:[opaque reasoning],CORR:
{RSv1:FullTrace}}, {ID:RS4,NAME:"PromptInjection",FM:[prompt governs reasoning],CORR:
{RSv1:SubstrateAbovePrompt}}, {ID:RS5,NAME:"RunawayLoops",FM:[no fail-safe],CORR:
{RSv1:TerminationCheckFailSafe}}, {ID:RS6,NAME:"Flailing",FM:[retries, wasted compute],CORR:
{RSv1:LimitationBeforeGeneration}} ]:: RSV2_EXTENSIONS=[MultiState, MultiAgent, Hierarchy,
DistributedSubstrates, MultiObjectiveEvaluation, AdaptiveOperatorSets, SubstrateLevelLearning,
SubstrateToSubstrateComm, RPUNativeExecution]:: RPU_MAPPING={STATE:"register_space",
LIMITATION:"constraint_enforcement_circuit", GENERATION:"candidate_generation_path",
COLLAPSE:"deterministic_selection_circuit", FIXEDPOINT:"convergence_evaluation_circuit",
CYCLE:"native_hardware_loop"}:: SECURITY_SAFETY={Determinism, Replayability, Auditability,
InjectionResistance, FailSafeTermination, PowerEfficiency}::
BENEFITS=[UniversalReasoningFoundation, ModelAgnostic, DomainAgnostic,
DeterministicStateEvolution, FullTraceability, StableConvergence, HardwareCompatibility,
NonVonNeumannReady]::
THESIS="RS_defines_reasoning_as_deterministic_state_evolution_via_4_operators(Limitation,Generati
on,Collapse,FixedPoint)+termination_check::RSv1_is_minimal_cycle::RSv2_generalizes_to_multi-
agent_distributed_substrates::substrate-level_governance_eliminate_prompt-
governed_failure_modes::DIFFERENTIATOR_IS_SCOPE-
one_unmodified_substrate_verified_across_domains_with_no_structural_relationship_to_each_other(AI
_reasoning,physical_control,financial_signal_damping,hardware_execution),not_claimed_elsewhere":
SUMMARY={ Architecture:"RSv1_and_RSV2_define_universal_reasoning_substrate_beneath_models_and_age
nts", Pattern:"reasoning_is_state_evolution_via_4_operators,not_sampling", Scope:"AI_safety|
planning|control|multi-agent|hardware", Refs:"RS_white_paper_v1.2_2026", KeyInsight:"substrate-
governed_reasoning_provides_determinism_auditability_replayability_and_injection_resistance::nove
lty_is_the_verified_combination_and_cross-domain_scope,not_state-
evolution_formalization_in_isolation" }::END
```