Reasoning Substrate: A Model-Agnostic Deterministic Reasoning Cycle

RS v1 and RS v2

A Technical White Paper

Version 1.0 | June 2026

Classification: Public Technical Reference

Document Series: Reasoning Substrate Architecture

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

Table of Contents

12. Future Directions

— Closing Statement

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 seven formally defined components. 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, plan, score, 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 condition triggers outside the stability threshold, the system performs a controlled safe return with no runaway loops. Power efficiency is realized by selecting plans via energy minimization before any state update is applied, 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 all seven RS v1 elements. 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

Contemporary AI systems and agent frameworks share a structural deficiency: they lack a universal, architecture-agnostic reasoning substrate. 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. Without a formally defined substrate that specifies how state evolves, how candidate actions are scored, and how the system terminates, 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, plan, score, 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 — forward prediction, plan generation, scoring, collapse, and state update — until the state achieves stability or the fail-safe termination condition is triggered. Any of the foregoing mechanisms may serve as a component within the RS cycle, 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 forward model computes, what the plan generation mechanism produces, or what domain the state represents. It prescribes only the cycle structure: how the state is predicted, how candidate plans are generated and scored, how the best plan is selected by the collapse operator, and how the state is updated by the apply operator. 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 step of the cycle is formally defined — including the state, the forward model, the plan set, the scoring function, the collapse operator,

the apply operator, and the termination condition — every invocation of the cycle with identical inputs produces identical intermediate states and identical outputs. Every intermediate state, plan, score, and collapse event is inspectable. These properties are not implementation choices; they are consequences of the formal cycle definition.

RS v1 consists of seven formally defined components, which together constitute the complete substrate specification: STATE (S), FORWARD MODEL (F), PLAN SET (P), SCORING FUNCTION (E), COLLAPSE OPERATOR (C), APPLY OPERATOR (A), and the REPEAT cycle governed by a TERMINATION condition. These seven 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 S_t denote the structured state at discrete time step t. The cycle proceeds through the following six steps at each time step, repeating until the termination condition is satisfied. The pseudocode that follows defines the abstract substrate cycle only. It does not specify the internal operators, hardware primitives, or proprietary update rules of any implementation.

4.1 Step 1 — Forward Model

The Forward Model F takes the current state S_t as input and produces a predicted near-future state $\hat{S}_{(t+1)}$ as output. This prediction represents the substrate's anticipation of where the state will be at the next step, given the current state and the dynamics encoded in F. The predicted state $\hat{S}_{(t+1)}$ serves as the conditioning context against which candidate plans are subsequently scored.

4.2 Step 2 — Generate Candidate Plans

A set of n candidate plans is generated. Each candidate plan p_i proposes a specific state update ΔS_i — that is, a transformation to be applied to the current state. The plan set P represents the space of possible next actions available to the substrate at time step t. The substrate does not apply any plan at this step; it generates the candidate set for evaluation by the scoring function.

4.3 Step 3 — Scoring Function

The Scoring Function E assigns a real-valued energy (or cost) to each candidate plan p_i, conditioned on the predicted future state $\hat{S}_{(t+1)}$. Lower energy values indicate plans that are more coherent with the predicted future state; higher energy values indicate plans that diverge from it. This energy assignment provides the quantitative basis upon which the Collapse Operator selects the best plan in the subsequent step.

4.4 Step 4 — Collapse Operator

The Collapse Operator C selects the plan p^* with the minimum energy score across all candidate plans in P. This operation collapses the plan set from n candidates to a single selected plan deterministically, without sampling or stochastic selection. The collapse is fully determined by the energy scores computed in Step 3; given identical inputs and an identical scoring function, the collapse always produces the same selected plan.

4.5 Step 5 — Apply

The Apply Operator A updates the current state by composing the current state S_t with the state update $\Delta S(p^*)$ proposed by the selected plan p^*. The composition operator $\oplus$ denotes the structured application of the update to the state; the specific semantics of $\oplus$ are determined by the state representation and are not fixed by the abstract cycle definition. The result is the new state $S_{(t+1)}$, which becomes the input to the next cycle iteration.

4.6 Step 6 — Termination

$$\text{Stop when: stability} \quad \text{OR} \quad \text{fail-safe triggered}$$

The cycle repeats from Step 1 with $S_{(t+1)}$ as the new current state unless one of two termination conditions is met. The primary termination condition is that the stability measure of the current state meets or exceeds a defined threshold, indicating that the state has reached sufficient coherence and resolution. The secondary termination condition is the triggering of the fail-safe mechanism, which initiates a controlled safe return. When the fail-safe is triggered, the system does not continue cycling; it returns without further state updates, eliminating the possibility of runaway loops.

Scope Note

The foregoing pseudocode defines only the abstract substrate cycle. It does not describe the internal operators, hardware primitives, domain-specific plan generators, scoring implementations, or proprietary update rules of any specific RS deployment. The cycle structure is fixed; the component implementations are substrate-agnostic by design.

5. Operators, State, Energy, and Collapse: Component Reference

The following section provides a structured technical reference for each of the seven formally defined components of the RS v1 cycle. Each entry specifies the component's formal name and symbol, its role within the cycle, and its input-output signature as derived from the mathematical definitions in Section 4.

1. STATE — Symbol: S

- **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 time.
- **Input:** Initialized from the problem context; updated at each cycle step by the Apply Operator.
- **Output:** S_t at time t — the current state consumed by all other components in the cycle.
- **Notes:** The state is structured and inspectable at every cycle step. Its internal representation is not fixed by the abstract cycle definition.

2. FORWARD MODEL — Symbol: F

- **Role:** The Forward Model predicts the near-future state of the system, providing the conditioning context for plan scoring.
- **Input:** S_t — the current structured state.
- **Output:** $\hat{S}_{(t+1)}$ — the predicted near-future state.

- **Notes:** F may be implemented using any mechanism — learned, rule-based, or analytic — that produces a prediction of the next state given the current state. The abstract cycle does not constrain the internal implementation of F.

3. PLAN SET — Symbol: P

- **Role:** The Plan Set is the set of candidate state updates available to the substrate at each cycle step. It represents the space of possible next actions.
- **Input:** Generated from the current context; conditioned on S_t and $\hat{S}_{(t+1)}$.
- **Output:** $P = \{p_1, p_2, \ldots, p_n\}$, where each p_i proposes a state update ΔS_i.
- **Notes:** The cardinality n of the plan set is not fixed by the abstract cycle definition. The mechanism by which candidate plans are generated is not constrained by the substrate specification.

4. SCORING FUNCTION — Symbol: E

- **Role:** The Scoring Function assigns a real-valued energy to each candidate plan, conditioned on the predicted future state. It provides the quantitative basis for plan selection.
- **Input:** $p_i \in P$ and $\hat{S}_{(t+1)}$.
- **Output:** $E(p_i \mid \hat{S}_{(t+1)}) \to \mathbb{R}$ — a real-valued energy score for each plan.
- **Notes:** Lower energy indicates greater coherence between the proposed update and the predicted future state. The internal computation of E is not fixed by the abstract cycle definition.

5. COLLAPSE OPERATOR — Symbol: C

- **Role:** The Collapse Operator deterministically selects the single best plan from the candidate set by minimizing energy. It collapses the plan set from n candidates to one selected plan.
- **Input:** The full plan set P with associated energy scores $E(p_i \mid \hat{S}_{(t+1)})$.
- **Output:** $p^* = \text{argmin}_i \, E(p_i \mid \hat{S}_{(t+1)})$ — the selected plan.
- **Notes:** The collapse is deterministic and requires no sampling. Given identical inputs, C always produces the same selected plan.

6. APPLY OPERATOR — Symbol: A

- **Role:** The Apply Operator updates the current state by composing it with the state update proposed by the selected plan. It produces the state for the next cycle iteration.
- **Input:** S_t and $\Delta S(p^*)$ — the current state and the selected plan's proposed update.
- **Output:** $S_{(t+1)} = S_t \oplus \Delta S(p^*)$ — the updated state.
- **Notes:** The composition operator $\oplus$ is defined by the state representation and is not fixed at the abstract cycle level. No state update is applied unless a plan has been selected by the Collapse Operator.

7. TERMINATION

- **Role:** The Termination condition governs when the RS cycle stops. It defines two distinct stopping criteria: stability attainment and fail-safe activation.
- **Primary condition:** $\text{stability}(S_t) \geq \text{threshold}$ — the state has reached sufficient

coherence and resolution.

- **Secondary condition:** Fail-safe triggered — the system performs a controlled safe return, bypassing further state updates.
- **Notes:** The fail-safe condition prevents runaway loops. The threshold value for stability is defined by the deployment context and is not fixed by the abstract cycle definition.

---

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 Forward Model, the plan generator, or both — 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, forward model, plan set, scoring function, collapse, apply, termination — is expressed at the abstract mathematical 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 behavioral rules or model can be mapped to the FORWARD MODEL component, and its action space can be mapped to the PLAN SET 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 which plan is selected (via the Collapse Operator acting on energy scores) and when the cycle terminates (via the Termination condition). Neither 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 components, not as a directive that governs the substrate itself.

Power efficiency is a quantifiable benefit of the RS v1 energy-minimization architecture. In systems without a substrate-level scoring 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 performing energy-based plan evaluation before any state update is applied. Only the plan with the minimum energy score — the plan most coherent with the predicted future state — is applied at each cycle step, 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 $\text{stability}(S_t) \geq \text{threshold}$, any drift away from coherence at one cycle step is automatically subject to correction at the next. The Forward Model, Scoring Function, and Collapse Operator together constitute a continuous correction mechanism: each cycle step selects the plan most consistent with the predicted future state, 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. If the fail-safe is triggered — by conditions defined at the deployment level, such as cycle 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. The safe return is a structural guarantee of the RS v1 termination specification, not an application-layer exception handler.

Full auditability and replayability are direct consequences of the RS v1 cycle structure. Every state S_t, every plan $p_i \in P$, every energy score $E(p_i \mid \hat{S}_{(t+1)})$, and every collapse result p^* 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 step, enabling exact reproduction of any prior reasoning session.

---

7. RS v1 → RPU Hardware Mapping

The seven 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; OPERATORS map to hardware primitives; the FORWARD MODEL maps to the predictive hardware path; ENERGY maps to hardware scoring logic; COLLAPSE maps to a deterministic selection circuit; APPLY maps to the state-update circuit; and the CYCLE maps to a native hardware loop.

RS v1 Component	RPU Hardware Construct	Function
STATE (S)	Physical register space	Stores the structured state at each cycle step
OPERATORS (F, E, C, A)	Hardware primitives	Execute each substrate operation natively
FORWARD MODEL (F)	Predictive hardware path	Computes the predicted near-future state
ENERGY (E)	Hardware scoring logic	Assigns energy scores to candidate plans
COLLAPSE (C)	Deterministic selection circuit	Selects the minimum-energy plan without sampling
APPLY (A)	State-update circuit	Applies the selected plan's state update
CYCLE (repeat)	Native hardware loop	Executes the substrate 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 operations 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, argmin over an energy vector requires a sequential or parallel comparison of n values, typically implemented as a reduction operation. In the RPU, the deterministic selection circuit performs this operation as a native hardware primitive: the circuit is designed to identify the minimum-energy input and assert the corresponding plan selection in a single hardware operation. This implementation preserves the determinism property of the Collapse Operator — because the circuit is combinatorial and stateless with respect to the selection operation, identical energy inputs always produce identical plan selections — while executing at the speed of hardware logic rather than software comparison.

The state-update circuit that implements the Apply 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 state update would require a load from memory, an arithmetic or logical operation, and a store back to memory. In the RPU, the state-update circuit operates on register space directly, applying $\Delta S(p^*)$ to S_t in a single register-level operation. The net effect is that the RS v1 cycle maps to the RPU hardware loop with each cycle step corresponding to a native hardware operation, 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 energy landscapes that replace the single-scalar energy function of RS v1 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 seven components, 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 $\{S^{(1)}_t, S^{(2)}_t, \ldots, S^{(k)}_t\}$, each subject to its own RS cycle, with coordination occurring across the state set. Coherence between states is maintained by cross-state energy

terms in the Scoring Function, which penalize plans whose proposed state updates 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 forward-model, scoring, and collapse operations; coordination is achieved through shared or overlapping components of the energy landscape. 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 plan sets and termination conditions of lower-level substrate instances. The higher-level substrate performs planning and collapse over coarse-grained state representations; the selected higher-level plan constrains the plan sets available to lower-level substrates, which perform planning 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 operations on a partition of the structured state. Stability and coherence are maintained across the distributed environment through inter-node communication of energy scores and collapse results, 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 Energy Landscapes

In RS v1, the Scoring Function assigns a single real-valued energy to each candidate plan. In RS v2, the Scoring Function is extended to assign a vector-valued energy $\mathrm{E}(\mathrm{p_i} \mid \hat{\mathrm{S}}_{(t+1)}) \rightarrow \mathbb{R}^{m}$, where each dimension of the energy vector corresponds to a distinct objective. The Collapse Operator is correspondingly extended to perform multi-objective minimization over the energy vector, selecting the plan 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 — the Forward Model, the Scoring Function, the Collapse Operator, and the Apply Operator — 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 the model that may serve as an operator 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 the Forward Model F, the Scoring Function E, or the composition semantics of $\oplus$ may be updated based on the history of state trajectories, energy scores, and collapse results 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, energy scores, 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 condition, 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 operations — Forward Model, Scoring, Collapse, Apply — 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 candidate plans generated in Step 2 of the cycle, but the selection of the plan to apply is determined entirely by the energy scores computed by the Scoring Function and the deterministic collapse performed by the Collapse Operator. The adversarial input cannot modify the Scoring Function, the Collapse Operator, or the Termination condition through the prompt channel.

Auditability is provided as a complete structural record of reasoning behavior at the substrate level. Every state S_t maintained by the Apply Operator, every candidate plan p_i in the Plan Set P, every energy score $E(p_i \mid \hat{S}_{(t+1)})$ computed by the Scoring Function, and every collapse result p^* produced by the Collapse Operator 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, plan set, energy assignment, and plan 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 plans deterministically by energy minimization — without sampling or stochastic selection — identical inputs to the cycle produce identical outputs at every step. This property holds across all six steps of the cycle and through every iteration of the cycle from initialization to termination. Replayability enables exact reproduction of any prior reasoning session: given the initial state, the Forward Model, the plan generation

mechanism, and the Scoring Function used in a prior session, the substrate produces an identical reasoning trajectory without deviation.

Fail-safe behavior is defined by the secondary termination condition. The fail-safe mechanism is triggered by conditions defined at the deployment level — cycle count bounds, resource limits, or externally imposed halt signals — that are independent of the stability threshold. 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. This behavior is guaranteed by the Termination specification: the fail-safe condition is an OR operand in the termination clause, meaning that its satisfaction unconditionally stops the cycle. 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 the Forward Model, the plan generation mechanism, the Scoring Function, or the Apply Operator. 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 set of fixed operations 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 seven components of the cycle — STATE, FORWARD MODEL, ENERGY, COLLAPSE, APPLY, OPERATORS, and the CYCLE itself — correspond to discrete hardware constructs in the RPU architecture: physical register space, predictive hardware path, hardware scoring logic, deterministic selection circuit, state-update circuit, hardware primitives, and 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 step 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 — the Forward Model, the Scoring Function, and the composition semantics of the Apply Operator — 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, energy scores, 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. The development of a formal substrate communication protocol would enable the construction of reasoning networks with formally specified coordination behavior, extending the auditability and replayability properties of RS v1 to multi-substrate networks.

Hierarchical substrate stacks, defined in RS v2 as layered substrate instances in which higher-level substrates govern the plan sets 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 the definition of how energy information propagates across substrate layers, how termination conditions compose across layers, and how the auditability record is maintained across a deep hierarchical stack. Deep hierarchical substrate stacks represent a substrate-level approach to the class of reasoning problems that require both high-level strategic planning and low-level operational execution.

RPU hardware acceleration is the direction indicated by the RS v2 introduction of RPU-native execution patterns. In its RS v2 formulation, RPU-native execution patterns expose the RS cycle operations as hardware-callable primitives for single-substrate deployments and for the four RS v2 RPU mappings: multi-register banks, layered hardware loops, multi-core substrate mesh, and reconfigurable hardware primitives. 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 the Collapse Operator, and the multi-core substrate mesh topology that supports distributed substrates. A fully

hardware-accelerated RS substrate — one in which every cycle step 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, behavioral model, and action space, 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. Automated legacy compilation would enable RS to be deployed as a universal substrate retrofit for a broad class of existing AI and non-AI systems, extending the RS security, safety, and auditability properties to systems that were not originally designed with a substrate-level control loop.

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. 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 — Reasoning Substrate (RS v1 & RS v2)

RSFAM::ΣCORE={
RSv1:Σ[DetCycle+StateEvo+ForwardModel+PlanSet+EnergyScore+Collapse+Apply+Termination],
RSv2:Σ[MultiState+MultiAgent+Hierarchy+Distributed+MultiObjective+AdaptiveOps+SubstrateLearning+SubstrateComm+RPUNative],
Safety:Σ[Determinism+Replay+Audit+InjectionResist+FailSafe+PowerEfficient]
}::

RSv1_CYCLE={
Forward:"Ŝ(t+1)=F(S_t)",
Plans:"P={p1..pn}, each ΔS_i",
Energy:"E(p_i|Ŝ(t+1))→ℝ",
Collapse:"p* = argmin_i E(p_i|Ŝ(t+1))",
Apply:"S(t+1)=S_t ⊕ ΔS(p*)",
Terminate:"stability(S_t)≥threshold OR fail_safe"
}::

COMPONENTS={
STATE:"structured, inspectable, domain-agnostic",
FORWARD:"predictive model, any mechanism",
PLANSET:"candidate updates ΔS_i",
ENERGY:"real-valued coherence metric",
COLLAPSE:"deterministic argmin",
APPLY:"structured state update",
TERMINATION:"stability or fail-safe"
}::

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:TerminationFailSafe}},
{ID:RS6,NAME:"Flailing",FM:[retries, wasted compute],CORR:{RSv1:EnergyMinBeforeApply}}
]::

RSv2_EXTENSIONS=[
MultiState,
MultiAgent,
Hierarchy,
DistributedSubstrates,
MultiObjectiveEnergy,
AdaptiveOperatorSets,
SubstrateLevelLearning,
SubstrateToSubstrateComm,
RPUNativeExecution
]::

RPU_MAPPING={
STATE:"register_space",
FORWARD:"predictive_path",
ENERGY:"hardware_scoring",
COLLAPSE:"deterministic_selection_circuit",
APPLY:"state_update_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::RSv1_is_minimal_cycle::RSv2_generalizes_to_multi-agent_distributed_substrates::substrate-level_governance_eliminates_prompt-governed_failure_modes"::

SUMMARY={
Architecture:"RSv1_and_RSv2_define_universal_reasoning_substrate_beneath_models_and_agents",
Pattern:"reasoning_is_state_evolution_not_sampling",
Scope:"AI_safety|planning|control|multi-agent|hardware",
Refs:"RS_white_paper_2026",
KeyInsight:"substrate-governed_reasoning_provides_determinism_auditability_replayability_and_injection_resistance"
}::END