Interference and Collapse-Based Security Model

Security-Native Execution Architecture

Honza Rožek

Contents

1 Introduction

Security in contemporary computational systems is predominantly enforced through external mechanisms such as access control, privilege separation, sandboxing, and monitoring. These approaches treat security as an overlay applied to an otherwise neutral execution model.

As computational systems become increasingly distributed, concurrent, and adaptive, external security enforcement introduces complexity, attack surfaces, and failure modes that are difficult to reason about formally.

This document introduces a security-native execution architecture in which security properties are enforced intrinsically through execution dynamics rather than through external control layers. The proposed model leverages interference and controlled state collapse as first-class mechanisms for enforcing isolation, containment, and finalization.

Rather than preventing disallowed behavior through access checks, the system renders such behavior computationally ineffective or self-terminating through structured interference and irreversible collapse boundaries.

The architecture is intentionally platform-neutral and does not rely on any specific hardware features, operating systems, instruction sets, or execution environments. It is applicable to a broad range of computational substrates, including centralized, distributed, and adaptive systems.

The primary contributions of this work include:

1. the formulation of interference as a security control mechanism,
2. the definition of controlled state collapse as an enforcement primitive,
3. an execution model in which security is intrinsic rather than external,
4. a formal foundation for containment and fault isolation.

Subsequent sections formalize interference-based control, state collapse mechanisms, and security-native enforcement, and analyze formal properties, failure modes, and example embodiments.

2 Interference as a Control Mechanism

This section formalizes interference as a first-class control mechanism within the proposed security-native execution architecture. Unlike traditional systems in which interference is treated as an error or side effect, the proposed model treats interference as an intentional and controllable means of regulating execution behavior.

2.1 Reframing Interference

In conventional execution models, interference between concurrent operations is typically undesirable and mitigated through synchronization, isolation, or serialization. Such mitigation increases system complexity and introduces additional attack surfaces.

In the proposed architecture, interference is reframed as a constructive mechanism through which execution may be shaped, constrained, or neutralized. Rather than preventing interference, the system leverages it to enforce security and control properties.

2.2 Interference Domains

Interference occurs when multiple execution influences act upon overlapping regions of execution state or context. The architecture defines explicit interference domains within which such interactions are permitted and regulated.

An interference domain may be defined by:

- state region boundaries,
- operator scope,
- contextual or policy constraints.

By controlling the structure of interference domains, the system constrains how execution effects may combine or conflict.

2.3 Constructive and Destructive Interference

Interference may be constructive or destructive depending on system objectives.

Constructive interference amplifies or reinforces desired execution behavior, while destructive interference suppresses, attenuates, or neutralizes undesired behavior. The architecture supports both modes as intentional control tools.

Destructive interference is particularly relevant for security enforcement, as it allows disallowed or anomalous execution paths to be rendered ineffective without explicit denial or interruption.

2.4 Policy-Governed Interference

Interference behavior may be governed by policy rules embedded within the execution context. Such policies may specify:

- which interactions are permitted,
- which interactions result in suppression,
- thresholds beyond which interference escalates.

Policies may be static or dynamically adjustable, enabling adaptive security responses without altering execution structure.

2.5 Interference as an Alternative to Access Control

Traditional access control mechanisms operate by permitting or denying actions before execution. In contrast, interference-based control allows actions to proceed but neutralizes their effects when they violate security constraints.

This approach:

- reduces reliance on privileged decision points,
- limits the impact of incorrect permission checks,
- transforms violations into non-effective execution.

Security is thus enforced through execution dynamics rather than external checks.

2.6 Containment Through Interference

Interference may be used to contain execution effects within defined boundaries. When execution attempts to propagate beyond permitted domains, destructive interference prevents further influence.

Such containment limits fault propagation and restricts the blast radius of erroneous or malicious behavior.

2.7 Deterministic Interference Resolution

Although interference introduces interaction between concurrent effects, its resolution is governed by deterministic rules defined within the execution context. Given fixed policies and context configuration, interference resolution is reproducible.

This determinism supports auditability and formal reasoning about security behavior.

2.8 Role Within the Security-Native Architecture

Interference-based control forms the first layer of security enforcement within the proposed architecture. It enables regulation, containment, and suppression of execution behavior prior to or in conjunction with state collapse mechanisms, which are introduced in the following section.

3 State Collapse Mechanisms

This section introduces state collapse as a fundamental enforcement mechanism within the security-native execution architecture. State collapse provides an irreversible transition that terminates, isolates, or finalizes execution in response to defined conditions.

Unlike traditional security mechanisms that rely on preventing execution, collapse-based enforcement renders certain execution states unrecoverable or non-propagating, thereby enforcing security properties intrinsically.

3.1 Definition of State Collapse

State collapse is defined as an irreversible transition of execution state or context beyond which prior states cannot be reconstructed through normal execution dynamics.

Formally, let $C(t)$ denote an execution context state. A collapse transition maps:

$$C(t) \longrightarrow C^{\dagger},$$

where $C^{\dagger}$ denotes a collapsed state that:

- does not admit inverse transitions,
- restricts or terminates further execution,
- enforces defined security boundaries.

Collapse is therefore distinct from rollback, exception handling, or error states.

3.2 Collapse Triggers

Collapse may be triggered by a variety of conditions, including but not limited to:

- violation of interference policies,
- detection of disallowed operator interaction,
- exceeding stability or resource thresholds,
- completion or finalization of execution.

Triggers may be deterministic and context-dependent, ensuring predictable and auditable behavior.

3.3 Partial and Localized Collapse

Collapse need not affect the entire execution context. In many embodiments, collapse may be partial or localized to specific regions of state or context.

Localized collapse enables:

- isolation of faulty or malicious execution paths,
- preservation of unaffected computation,
- containment of security incidents.

The scope of collapse is defined by context topology and policy constraints.

3.4 Collapse as a Security Primitive

State collapse functions as a first-class security primitive. Rather than attempting to block disallowed behavior preemptively, the system permits execution to proceed until collapse conditions are met, at which point execution is irreversibly constrained.

This approach:

- minimizes reliance on privileged checks,
- reduces attack surfaces associated with access control,
- ensures enforcement even under partial system compromise.

3.5 Interaction with Interference

Interference and collapse operate in complementary layers. Interference regulates and suppresses execution behavior dynamically, while collapse enforces hard boundaries when regulation is insufficient or violations persist.

In some embodiments, escalating interference may precede collapse, providing a graded response mechanism.

3.6 Determinism and Auditability

Collapse behavior is governed by deterministic rules defined within the execution context. Given identical context configuration and execution history, collapse events occur reproducibly.

This determinism enables:

- forensic analysis,
- security auditing,
- replay-based verification.

3.7 Finalization and Commitment

Beyond security enforcement, collapse may serve as a mechanism for finalization or commitment of execution results. Once a collapse boundary is crossed, results may be considered finalized and immutable.

This dual use supports both security and correctness guarantees.

3.8 Role Within the Security-Native Architecture

State collapse provides the irreversible enforcement layer of the proposed architecture. Together with interference-based control, it enables a security model in which disallowed behavior is rendered computationally ineffective or irreversible without reliance on external enforcement mechanisms.

Subsequent sections examine how collapse and interference combine to form a security-native execution model and how isolation and containment are achieved.

4 Security-Native Enforcement

This section describes how security enforcement is realized intrinsically within the execution model through the combined use of interference and state collapse. Security is not treated as an external constraint or monitoring layer, but as a native property of execution dynamics.

4.1 Security as an Execution Property

In the proposed architecture, security properties are embedded directly into the rules governing execution behavior. Enforcement does not rely on external access checks, privilege levels, or supervisory control paths.

Instead, execution is structured such that disallowed behavior either:

- fails to produce effective state transitions due to interference, or
- triggers irreversible collapse that constrains further execution.

Security is therefore enforced by the same mechanisms that govern computation.

4.2 Elimination of Privileged Enforcement Paths

Traditional systems rely on privileged enforcement paths, such as kernel-mode checks or trusted monitors, which constitute high-value attack targets. The security-native model eliminates the need for such privileged paths.

Because enforcement arises from execution dynamics, no single component holds exclusive authority over security decisions. This distributed enforcement reduces single points of failure and privilege escalation risks.

4.3 Policy Integration into Execution Context

Security policies are expressed as part of the execution context rather than as external rule sets. Policies may define:

- permissible operator interactions,
- interference thresholds,
- collapse trigger conditions,
- containment boundaries.

By embedding policy into the execution context, enforcement becomes inseparable from execution behavior.

4.4 Progressive Enforcement

The architecture supports progressive enforcement, in which security responses escalate gradually rather than abruptly. For example:

1. initial suppression through destructive interference,
2. containment within restricted context regions,
3. localized collapse upon persistent violation,
4. global collapse in extreme cases.

This graded response enables resilience and reduces unnecessary termination of benign computation.

4.5 Fail-Safe Behavior

Security-native enforcement is inherently fail-safe. When uncertainty, inconsistency, or undefined behavior is detected, execution dynamics favor suppression or collapse rather than permissive continuation.

This bias toward safety ensures that failures degrade execution capability rather than expanding it.

4.6 Resistance to Circumvention

Because enforcement mechanisms are intrinsic to execution, attempts to bypass security controls necessarily alter execution dynamics. Such alterations are themselves subject to interference and collapse.

As a result, circumvention attempts either:

- fail to produce meaningful execution effects, or
- accelerate enforcement through collapse.

4.7 Auditability and Verification

Security-native enforcement is deterministic under fixed context configuration and execution history. Enforcement actions, including interference suppression and collapse events, are reproducible and inspectable.

This determinism enables formal verification, audit trails, and forensic analysis without reliance on external logging mechanisms.

4.8 Role Within the Overall Architecture

Security-native enforcement integrates interference-based control and state collapse into a unified security model. This integration transforms security from an external constraint into an inherent property of execution.

The next section examines how isolation and containment are achieved using these mechanisms to limit fault propagation and execution impact.

5 Isolation and Containment

This section describes mechanisms for isolating execution behavior and containing faults, anomalies, or malicious activity within the security-native execution architecture. Isolation and containment are achieved intrinsically through execution dynamics rather than through external sandboxing or privilege separation.

5.1 Contextual Isolation

Isolation in the proposed architecture is defined contextually rather than structurally. Execution behavior is isolated by constraining how operator effects may propagate through the execution context.

Contextual isolation may be achieved by:

- limiting operator scope to defined context regions,
- defining interference boundaries between regions,
- restricting cross-region interaction policies.

This approach enables fine-grained isolation without duplicating execution environments.

5.2 Containment Boundaries

Containment boundaries define limits beyond which execution effects may not propagate. Such boundaries are enforced through destructive interference and, if necessary, localized state collapse.

Containment boundaries may be:

- static, defined at system configuration time,
- dynamic, adjusted in response to execution behavior,
- hierarchical, supporting nested containment regions.

These boundaries constrain the impact of faults and violations.

5.3 Localized Failure Containment

When anomalous or disallowed behavior is detected, containment mechanisms prevent its effects from influencing unrelated regions of execution.

Localized containment enables:

- continued execution of unaffected computation,
- targeted enforcement without global disruption,
- reduction of system-wide failure risk.

Containment is therefore proportional to the scope of the detected issue.

5.4 Isolation Without Sandboxing

Traditional isolation mechanisms rely on sandboxing, virtualization, or hardware enclaves, which introduce performance overhead and complexity. The proposed architecture achieves isolation without such constructs.

Because execution effects are regulated by interference and collapse, isolation emerges naturally from execution dynamics rather than being imposed externally.

5.5 Adaptive Containment

Containment behavior may adapt dynamically based on execution context, threat assessment, or policy changes. For example, containment regions may be expanded or contracted in response to observed behavior.

Adaptive containment allows the system to respond proportionally to evolving conditions while minimizing unnecessary restriction.

5.6 Containment Escalation

In cases of persistent or severe violations, containment may escalate from interference-based suppression to localized collapse and, if required, broader collapse of execution regions.

Escalation paths are defined by policy and context configuration, enabling predictable and auditable responses.

5.7 Deterministic Containment Behavior

Containment and isolation decisions are governed by deterministic rules within the execution context. Given fixed configuration and execution history, containment outcomes are reproducible.

This determinism supports auditability and formal analysis of containment behavior.

5.8 Role Within the Security-Native Architecture

Isolation and containment mechanisms limit the blast radius of faults and violations, ensuring that execution remains robust and controlled even under adverse conditions.

The next section compares the security-native approach to traditional access control and sandbox-based security models and identifies resulting technical effects.

6 Comparison to Access Control Models and Technical Effects

This section compares the proposed security-native execution architecture to traditional access control and sandbox-based security models and identifies the technical effects arising from the intrinsic enforcement mechanisms described herein.

6.1 Traditional Access Control Models

Conventional security architectures enforce restrictions through access control mechanisms such as access control lists (ACLs), role-based access control (RBAC), capability systems, or mandatory access control frameworks.

These models operate by:

- evaluating permissions prior to execution,
- granting or denying access to resources or operations,
- relying on trusted enforcement components.

Security is thus external to execution semantics and depends on correct and complete policy evaluation.

6.2 Sandboxing and Isolation-Based Models

Sandboxing and virtualization-based security models enforce isolation by executing code within restricted environments. Such models rely on:

- structural separation of execution environments,
- privileged monitoring or mediation layers,
- explicit containment boundaries enforced externally.

While effective, these approaches introduce performance overhead, complexity, and additional attack surfaces.

6.3 Security-Native Execution Model

The proposed architecture differs fundamentally in that security enforcement is intrinsic to execution dynamics. Rather than denying execution or isolating it structurally, the system regulates execution effects through interference and state collapse.

Execution that violates security constraints:

- fails to produce effective state transitions due to destructive interference, or
- triggers irreversible collapse that constrains further execution.

Security is thus enforced by computation itself rather than by external checks.

6.4 Elimination of Pre-Execution Permission Checks

Unlike access control models, the security-native approach does not require pre-execution permission evaluation. Actions are not blocked in advance; instead, their effects are neutralized if disallowed.

This eliminates:

- dependency on correct permission resolution,
- race conditions between permission checks and execution,
- privileged decision points susceptible to attack.

6.5 Technical Effects of Security-Native Enforcement

The architectural differences described above give rise to several concrete technical effects:

1. **Intrinsic Enforcement**: security properties are enforced by execution dynamics rather than external mechanisms.
2. **Reduced Attack Surface**: elimination of privileged enforcement paths reduces opportunities for escalation.
3. **Fine-Grained Containment**: interference and collapse enable localized containment without global disruption.
4. **Fail-Safe Behavior**: violations result in suppression or collapse rather than unintended execution.
5. **Deterministic Auditability**: enforcement behavior is reproducible under fixed configuration.

These effects arise directly from the intrinsic integration of security into the execution model.

6.6 Scalability and Adaptability

Access control and sandbox-based models often struggle to scale in highly dynamic or distributed environments due to policy synchronization and boundary management.

The security-native model scales naturally with execution context, as interference and collapse operate locally and adaptively without centralized coordination.

6.7 Resistance to Misconfiguration

In traditional models, misconfigured permissions may silently grant excessive privilege. In contrast, the security-native model biases execution toward suppression and collapse when uncertainty or inconsistency is detected.

This bias provides inherent resistance to configuration errors.

6.8 Summary of Technical Advantages

Compared to traditional access control and sandbox-based security models, the security-native execution architecture provides:

1. enforcement intrinsic to execution,
2. reduced reliance on privileged components,
3. localized and adaptive containment,
4. deterministic and auditable security behavior,
5. improved robustness against misconfiguration and circumvention.

These advantages demonstrate that the proposed architecture constitutes a substantive technical improvement rather than a policy-layer variation of existing security models.

7 Formal Properties and Invariants

This section defines formal properties and invariants of the interference- and collapse-based security architecture. These properties hold independently of implementation details and provide guarantees regarding correctness, containment, and enforceability of security behavior.

7.1 Context Closure Invariant

Let $\mathcal{C}$ denote the execution context and $\mathcal{S}$ the associated state space. A fundamental invariant of the architecture is that all execution evolution preserves membership in $\mathcal{C}$.

For any valid context state $C(t)$ and admissible execution influences, the resulting state satisfies:

$$C(t+1) \in \mathcal{C}.$$

This closure property ensures that execution remains well-defined even under interference and collapse transitions.

7.2 Interference Determinacy

Given a fixed execution context configuration and policy set, the resolution of interference between concurrent execution influences is deterministic.

That is, identical context states and interaction rules yield identical interference outcomes. This invariant enables reproducible security behavior and formal reasoning about enforcement.

7.3 Collapse Irreversibility Invariant

State collapse transitions are irreversible by construction. Once an execution context enters a collapsed state $C^{\dagger}$, no sequence of admissible execution dynamics can reconstruct a prior non-collapsed state.

This irreversibility invariant enforces hard security boundaries and prevents rollback-based circumvention.

7.4 Containment Preservation

Containment boundaries defined within the execution context are preserved under execution evolution. Execution effects originating within a containment region cannot propagate beyond its boundary unless explicitly permitted by policy.

This invariant ensures that containment remains effective even under adversarial or anomalous execution behavior.

7.5 Locality of Enforcement

Security enforcement actions, including interference suppression and localized collapse, are confined to the minimal execution context regions necessary to enforce policy.

This locality invariant prevents unnecessary global disruption and supports graceful degradation.

7.6 Fail-Safe Bias

In the presence of ambiguity, inconsistency, or undefined interaction, the architecture biases execution toward suppression or collapse rather than permissive continuation.

This fail-safe invariant ensures that uncertainty reduces execution capability rather than expanding it.

7.7 Deterministic Auditability

Given fixed context configuration, policy definitions, and execution history, the occurrence and scope of enforcement actions are uniquely determined.

This invariant supports deterministic replay, auditability, and forensic analysis of security behavior.

7.8 Compositional Security Invariance

Execution subsystems with defined interference and collapse rules may be composed into larger systems without violating security invariants.

Composed systems preserve containment, irreversibility, and enforcement locality, enabling hierarchical and modular security design.

7.9 Bounded Propagation Invariant

All execution influence propagation is bounded by interference domains and containment boundaries. No execution effect may propagate unboundedly through the execution context.

This invariant constrains blast radius and supports predictable system behavior.

7.10 Summary of Formal Properties

The security-native execution architecture satisfies the following invariant properties:

1. execution context closure,
2. deterministic interference resolution,
3. irreversible collapse transitions,
4. preservation of containment boundaries,
5. localized and proportional enforcement,
6. fail-safe behavior under uncertainty,
7. deterministic auditability,
8. compositional security guarantees.

These invariants establish the architecture as a formally well-defined and robust security model intrinsic to execution dynamics rather than external control mechanisms.

8 Failure Modes and System Limits

This section identifies potential failure modes of the interference- and collapse-based security architecture and analyzes how such failures manifest and are contained within the execution model. Explicit modeling of failure modes is essential for safe deployment and formal reasoning about system limits.

8.1 Policy Misconfiguration

A primary failure mode arises from misconfigured security policies governing interference and collapse. Policies that are overly permissive may fail to suppress disallowed execution, while overly restrictive policies may trigger premature suppression or collapse.

Policy misconfiguration may result in:

- unintended execution suppression,
- excessive containment,
- unnecessary collapse of benign computation.

Such failures are configuration errors rather than architectural flaws and are detectable through deterministic audit mechanisms.

8.2 Over-Aggressive Interference

Excessive use of destructive interference may prevent meaningful execution progress. Over-aggressive interference may manifest as:

- persistent suppression of operator effects,
- oscillatory execution behavior,
- inability to reach stable configurations.

This failure mode highlights the need for balanced interference thresholds.

8.3 Premature State Collapse

Collapse mechanisms that trigger too early may irreversibly terminate execution before intended computation is complete. Premature collapse may:

- destroy intermediate results,
- prevent recovery or replay,
- reduce system availability.

Because collapse is irreversible, careful definition of collapse triggers is essential.

8.4 Insufficient Collapse Escalation

Conversely, failure to escalate from interference to collapse when required may allow persistent violations or anomalous behavior to continue.

This failure mode may result in:

- prolonged containment without resolution,
- accumulation of unstable execution regions,
- degraded system integrity.

8.5 Containment Boundary Leakage

Improperly defined containment boundaries may allow execution effects to propagate beyond intended regions. Leakage may occur due to:

- incorrect context topology definition,
- unintended operator scope overlap,
- policy inconsistencies.

Such leakage constitutes a containment failure and is mitigated through boundary auditing and deterministic replay.

8.6 Resource Exhaustion Under Enforcement

Security enforcement itself consumes execution resources. Excessive interference or repeated collapse events may lead to resource exhaustion, manifesting as:

- degraded performance,
- delayed enforcement responses,
- forced termination of execution.

This failure mode emphasizes the need for resource-aware enforcement policies.

8.7 Loss of Audit Fidelity

Auditability relies on accurate recording of context state, interference resolution, and collapse events. Loss of audit fidelity may arise from:

- incomplete execution traces,
- inconsistent context snapshots,
- external corruption of audit data.

Loss of audit fidelity undermines verification but does not compromise intrinsic enforcement behavior.

8.8 Adversarial Stress Conditions

Under adversarial or pathological execution patterns, the system may experience high rates of interference and collapse. Such stress conditions may degrade throughput or availability but do not bypass enforcement guarantees.

Graceful degradation under stress is an expected behavior rather than a failure of security enforcement.

8.9 Boundaries of Applicability

The security-native execution architecture is not intended to eliminate all possible failures. Instead, it aims to ensure that failures are:

- localized,
- deterministic,
- observable,
- non-escalating beyond defined boundaries.

Understanding these boundaries informs appropriate configuration and deployment scenarios.

8.10 Summary

The failure modes described in this section delineate the operational limits of the security-native execution model. By explicitly modeling failure, the architecture ensures that security enforcement remains robust, auditable, and controllable even under adverse conditions.

9 Example Embodiments

This section presents non-limiting example embodiments illustrating practical realizations of the interference- and collapse-based security architecture. These embodiments are provided for explanatory purposes and do not limit the scope of the invention.

9.1 Single-Context Secure Execution

In one embodiment, a computational system comprises a single execution context in which interference and collapse rules are defined as part of the execution dynamics. Operators interact within this context, and security enforcement is achieved through destructive interference and controlled state collapse.

Disallowed execution behavior is suppressed or terminated intrinsically, without external access control checks or sandboxing.

9.2 Distributed Secure Execution Environment

In another embodiment, the execution context is distributed across multiple computational nodes. Each node maintains a partial or replicated context state and enforces interference and collapse rules locally.

Security enforcement occurs independently on each node, while containment boundaries prevent propagation of disallowed effects across the distributed context. This embodiment supports scalable and fault-tolerant secure execution.

9.3 Localized Containment Zones

In a further embodiment, the execution context is partitioned into multiple containment zones, each governed by its own interference policies and collapse thresholds.

When anomalous behavior is detected within a zone, enforcement actions are confined to that zone through localized interference or collapse, preserving execution in unaffected zones.

9.4 Progressive Enforcement Configuration

In one embodiment, security enforcement is configured progressively. Initial violations result in destructive interference that suppresses execution effects. Persistent or severe violations escalate to localized collapse, and global collapse is reserved for extreme cases.

This embodiment enables graded security response and minimizes unnecessary disruption.

9.5 Audit-Driven Secure Execution

In another embodiment, the execution system records context evolution, interference resolution, and collapse events as part of an audit trace. This information is sufficient to deterministically reproduce and analyze security enforcement behavior.

Such an embodiment supports compliance, forensic analysis, and post-execution verification.

9.6 Adaptive Security Policies

In a further embodiment, interference and collapse policies adapt dynamically based on execution context, observed behavior, or external input. Policy updates do not require modification of execution structure and are applied through context reconfiguration.

Adaptive policies enable responsive security behavior in evolving execution environments.

9.7 Security-Oriented Finalization

In some embodiments, state collapse is used not only for enforcement but also for finalization of execution results. Once a finalization collapse boundary is crossed, results become immutable and further execution is constrained.

This embodiment supports secure commitment and result integrity.

9.8 Implementation Flexibility

The foregoing embodiments may be realized using software-based execution, hardware-assisted execution, or hybrid approaches. No embodiment depends on any specific instruction set, hardware architecture, operating system, or execution platform.

Variations and combinations of these embodiments fall within the scope of the security-native execution architecture.

9.9 Non-Limiting Nature of Embodiments

The example embodiments described herein are illustrative rather than exhaustive. Additional embodiments consistent with interference-based control and collapse-based enforcement may be realized without departing from the scope of the invention as defined by the claims.

10 Conclusion

This document has presented a security-native execution architecture in which security enforcement is intrinsic to execution dynamics rather than imposed as an external control layer. The proposed model departs fundamentally from traditional access control and sandbox-based approaches by embedding enforcement directly into the mechanisms governing execution behavior.

Security is achieved through the intentional use of interference and controlled state collapse. Interference regulates and suppresses execution effects that violate policy or stability constraints, while collapse provides irreversible enforcement boundaries that ensure containment, finalization, and isolation. Together, these mechanisms render disallowed behavior computationally ineffective or unrecoverable.

The architecture has been formally described, including:

- interference as a first-class control mechanism,
- state collapse as an irreversible enforcement primitive,
- security-native enforcement without privileged control paths,
- intrinsic isolation and containment of execution effects,
- formal properties and invariants guaranteeing determinism and auditability,
- explicit identification of failure modes and system limits,
- non-limiting example embodiments demonstrating realizability.

Crucially, the proposed security model is independent of any specific hardware, software platform, instruction set, or execution environment. This implementation-agnostic design ensures broad applicability across centralized, distributed, and adaptive computational systems, including future architectures not yet conceived.

By transforming security from an external policy overlay into an inherent property of execution, the architecture reduces attack surface, limits fault propagation, and enables deterministic auditability. These properties constitute concrete technical effects that address fundamental limitations of conventional security models.

The scope of the invention is defined by the claims and is not limited by the specific examples or embodiments described herein.