Chapter 12

# Deterministic and Probabilistic Domain Coordination

Governing the Coexistence of Execution Regimes

proFQuansistor

**Abstract**

This document formalizes the coordination of deterministic and probabilistic execution domains within the Quantum Virtual Machine. While field-native execution admits intrinsic nondeterminism and concurrency, system-level operation often requires precise guarantees regarding reproducibility, bounded variability, and controlled interaction between execution regimes.

The chapter introduces coordination mechanisms through which QVM governs the coexistence of deterministic and probabilistic domains without altering underlying execution semantics. Determinism and probabilism are treated as governance-level execution modes rather than as properties of operators or fields themselves. QVM regulates their interaction through domain configuration, contract activation, and admissibility constraints.

Special attention is given to cross-domain interaction, where deterministic and probabilistic domains must cooperate without contaminating each other's guarantees. The document formalizes coordination patterns, boundary contracts, and audit implications arising from mixed-regime execution. By doing so, it establishes a rigorous foundation for hybrid field computing systems that combine exploration, optimization, and verified execution within a single governed infrastructure.

This whitepaper builds directly upon the isolation, composition, and audit mechanisms developed in previous chapters and prepares the ground for system-level field orchestration, economic governance models, and practical deployment on classical computational infrastructure.

## 1 Deterministic Execution Domains

Deterministic execution domains in QVM are governance constructs that impose reproducibility guarantees on field-native execution without modifying the underlying execution semantics. Determinism is not treated as a property of operators, fields, or physical realization, but as a domain-level obligation enforced through contracts and admissibility constraints.

A deterministic execution domain is defined as an execution domain $\mathcal{D}_{\text{det}}$ in which all admissible execution trajectories originating from an equivalent initial state class are required to be equivalent under a specified equivalence relation. This relation may identify configurations that differ in inessential structure while preserving the properties deemed relevant by governance policy.

Formally, let $[s_0]$ denote an equivalence class of initial execution states and let $\mathcal{T}(s_0)$ denote the set of all admissible execution trajectories originating from $s_0$. The domain $\mathcal{D}_{\text{det}}$ satisfies determinism if for all trajectories

$$\tau_1, \tau_2 \in \mathcal{T}(s_0),$$

their terminal states are equivalent under the domain's equivalence relation. This definition permits intrinsic nondeterminism at the execution level while enforcing determinism at the governance level.

Deterministic domains achieve this guarantee through contract activation rather than execution control. Contracts may constrain operator selection, bound admissible nondeterminism, enforce confluence properties, or require convergence to canonical representatives. These constraints are evaluated as part of admissibility and cannot be bypassed by scheduling modulation or concurrency.

Importantly, deterministic domains do not require serialization or centralized arbitration. Concurrent execution and parallel operator composition remain admissible as long as contract constraints ensure equivalence of outcomes. Determinism is thus compatible with intrinsic parallelism and does not imply a reduction to sequential execution.

Deterministic execution domains also define strong audit guarantees. Because outcome equivalence is contractually enforced, execution traces may be reconstructed up to the defined equivalence relation without ambiguity. This enables reproducible computation, verifiable results, and regulatory compliance in contexts where exact repeatability is required.

Transition into or out of a deterministic execution domain is a governed act. Domain admission requires verification that initial conditions satisfy all determinism contracts. Domain exit may require additional verification to ensure that determinism guarantees are preserved up to the transition boundary. Such transitions do not alter execution semantics but reconfigure governance obligations.

By defining deterministic execution domains as governance-enforced equivalence guarantees, this chapter establishes a precise and scalable notion of determinism for field-native computation. Determinism becomes an explicit contract rather than an implicit assumption, providing a robust foundation for coordination with probabilistic domains developed in subsequent chapters.

## 2 Probabilistic Execution Domains

Probabilistic execution domains in QVM are governance constructs that explicitly permit and regulate nondeterministic field evolution. Unlike deterministic domains, which enforce equivalence of outcomes, probabilistic domains treat variability as a first-class and intentional property of execution. Probability is not introduced as randomness injected into execution, but as a governed allowance of multiple admissible trajectories.

A probabilistic execution domain is defined as an execution domain $\mathcal{D}_{\text{prob}}$ in which admissible execution trajectories from a given initial state class are permitted to diverge without requiring convergence to an equivalent terminal configuration. The degree and nature of this divergence are bounded by governance policies and execution contracts rather than left unconstrained.

Formally, let $[s_0]$ denote an equivalence class of initial execution states and let $\mathcal{T}(s_0)$ denote the set of admissible trajectories originating from $s_0$. In a probabilistic domain, $\mathcal{T}(s_0)$ is intentionally non-singleton under the domain's equivalence relation, and governance policies may attach weights, bounds, or admissibility preferences to subsets of trajectories.

Probabilistic domains do not rely on external randomness sources or stochastic primitives. Variability arises intrinsically from concurrent admissible operator compositions, structural nondeterminism of the field, or exploratory admissibility modulation. Governance may optionally

introduce controlled stochastic selection mechanisms, but such mechanisms operate at the domain boundary and do not alter execution semantics.

Contracts play a central role in regulating probabilistic execution. Rather than enforcing outcome equivalence, contracts bound divergence, restrict pathological trajectories, and ensure that all admissible outcomes satisfy required safety, structural, and auditability invariants. Probabilistic freedom exists only within these normative boundaries.

Auditability in probabilistic domains is defined differently than in deterministic ones. Reconstruction does not aim to identify a unique execution trace, but to verify that an observed outcome lies within the admissible outcome space defined by active contracts and governance policies. Probability distributions, when specified, are subject to verification rather than assumption.

Probabilistic execution domains are particularly suited for exploratory computation, optimization, learning, and search processes. They enable the system to explore a space of possible field evolutions without prematurely collapsing execution into a single trajectory. Importantly, such exploration remains governed, auditable, and isolated from deterministic domains unless explicitly composed.

Transition into or out of a probabilistic execution domain is a governed act. Entry requires activation of contracts that permit bounded divergence, while exit may require projection or consolidation mechanisms if interaction with deterministic domains is anticipated. These transitions preserve execution semantics while reconfiguring governance obligations.

By defining probabilistic execution domains as governed spaces of admissible variability, this chapter establishes probability as a controlled execution regime rather than an implementation artifact. This formulation enables rigorous coordination with deterministic domains and prepares the ground for hybrid execution patterns and cross-domain interaction developed in subsequent chapters.

# 3 Coordination Between Deterministic and Probabilistic Domains

The coexistence of deterministic and probabilistic execution domains within a single governed field system requires explicit coordination mechanisms. Without such mechanisms, interaction between regimes would risk undermining determinism guarantees or trivializing probabilistic exploration. This chapter formalizes coordination as a governance-layer process that preserves the integrity of both execution regimes.

Coordination is defined as the regulated interaction between execution domains operating under different execution modes. Deterministic and probabilistic domains are never implicitly composable. Any interaction between them must be explicitly authorized through governance configuration, compositional interfaces, and boundary contracts.

The fundamental principle of coordination is non-contamination. Probabilistic variability must not propagate into deterministic domains in a way that violates their equivalence guarantees. Conversely, deterministic constraints must not collapse probabilistic domains into degenerate execution regimes. Coordination mechanisms ensure that each domain retains its normative identity.

One common coordination pattern is asymmetric interaction. Probabilistic domains may supply candidate field configurations, optimization results, or exploratory outcomes to deterministic domains through projection or selection interfaces. Deterministic domains may consume such results only after they have been filtered, validated, or canonicalized under deterministic contracts.

Another pattern is staged execution. A system may alternate between probabilistic and determin-

istic phases, where exploration occurs in probabilistic domains and verification or consolidation occurs in deterministic domains. Transitions between phases are governed acts that activate or deactivate relevant contract sets and redefine admissibility constraints.

Boundary contracts play a central role in coordination. These contracts specify admissible transformations at the interface between deterministic and probabilistic domains. They may enforce selection criteria, equivalence reduction, bounded variability, or statistical constraints. Boundary contracts ensure that interaction preserves the guarantees of both sides.

Auditability across coordinated domains is preserved by maintaining explicit separation of regimes. Execution traces within each domain remain reconstructible according to their respective audit definitions. Cross-domain interaction is auditable at the boundary by verifying that all transfers satisfied active boundary contracts.

Importantly, coordination mechanisms do not arbitrate execution outcomes. They do not choose which probabilistic trajectory is "best" or enforce convergence within probabilistic domains. Selection, optimization, or decision-making occurs outside coordination itself, typically at higher governance or orchestration layers.

By formalizing coordination between deterministic and probabilistic execution domains, this chapter establishes a rigorous framework for hybrid field-native computation. Determinism and probabilism become complementary execution regimes whose interaction is explicit, governed, and auditable. This framework enables complex computational workflows that combine exploration, optimization, and verification without compromising foundational guarantees.

# 4 Execution Mode Transitions and Hybrid Workflows

Hybrid field-native systems often require execution to move between deterministic and probabilistic regimes over time. This chapter formalizes execution mode transitions as governed acts and introduces hybrid workflows as structured compositions of execution phases operating under distinct governance obligations.

An execution mode transition is defined as a governance-level reconfiguration of an execution domain that changes the active execution mode, contract set, or admissibility constraints without altering underlying execution semantics. Transitions do not interrupt execution dynamics; they redefine the normative envelope within which execution proceeds.

Transitions are admissible only if they preserve all invariants required by both the source and target execution modes. This typically involves boundary verification steps such as equivalence reduction, canonicalization, or consolidation of field configurations. These steps ensure that obligations imposed by the target mode are satisfied at the moment of transition.

Hybrid workflows are defined as ordered or structured compositions of execution phases, each phase operating under a distinct execution mode. A common pattern consists of an exploratory probabilistic phase followed by a deterministic consolidation phase. Other patterns include iterative alternation between modes, parallel probabilistic exploration feeding a deterministic verifier, or staged execution with progressively tightening determinism constraints.

Governance policies specify which transitions are permitted, under what conditions they may occur, and which boundary contracts must be enforced. Policies are declarative and evaluated at designated governance checkpoints. There is no implicit or automatic transition between execution modes; all transitions are explicit and auditable.

Hybrid workflows preserve auditability by maintaining regime separation across phases. Each phase is auditable according to its execution mode, and transitions are auditable as boundary events. The resulting system-level audit trace consists of phase-local traces linked by verified transition points rather than a single monolithic execution history.

Importantly, execution mode transitions do not resolve nondeterminism or select outcomes. They regulate when and how such resolution may occur by activating contracts that impose equivalence or convergence requirements. Decision-making remains external to execution governance and may be performed by orchestration layers or human operators.

By formalizing execution mode transitions and hybrid workflows, this chapter establishes a temporal coordination framework for field-native computation. Deterministic and probabilistic regimes become phases of a coherent computational process rather than competing models. This framework prepares the ground for system-level orchestration, where multiple hybrid workflows are coordinated across domains and infrastructure.

## From Execution Regimes to System Orchestration

The preceding chapters have established deterministic and probabilistic execution domains as governed execution regimes, formalized their coordination, and defined explicit mechanisms for regime transitions and hybrid workflows. Together, these constructs elevate execution modes from local execution properties to system-level governance instruments.

At this point, execution regimes are no longer isolated concerns of individual domains. They become components of larger computational processes that unfold across time, domains, and infrastructures. Coordinating such processes requires a perspective that extends beyond individual domains or workflows and addresses system-wide structure, lifecycle, and interaction.

This transition marks the boundary between execution coordination and system orchestration. Execution regimes define *how* computation may evolve within a domain. System orchestration defines *how multiple governed executions coexist, interact, and progress collectively* toward higher-level objectives. Orchestration operates over domains, workflows, and governance configurations rather than over operators or execution states.

The Quantum Virtual Machine thus emerges not merely as a governance engine, but as a control plane for field-native computation. It admits domains, assigns execution modes, governs transitions, and composes hybrid workflows into coherent system behavior. Importantly, orchestration does not subsume execution; it arranges and coordinates it without altering its intrinsic semantics.

By moving from execution regimes to system orchestration, the architecture of QFC gains temporal, structural, and organizational coherence. Deterministic verification, probabilistic exploration, auditability, and isolation become orchestrated capabilities rather than isolated features. This shift prepares the ground for treating QVM as a field operating system—a layer responsible for system-scale coordination, resource governance, and lifecycle management of field-native computation.

The next document therefore addresses system-level field orchestration explicitly, formalizing QVM as a control plane that coordinates governed execution at scale while preserving the foundational guarantees established throughout this book.