Chapter 13

System-Level Field Orchestration

QVM as a Field Operating System and Control Plane

proFQuansistor

Abstract

This document introduces system-level field orchestration as the highest governance layer of the Quansistor Field Computing architecture. Building upon execution domains, isolation, execution regimes, and hybrid workflows, it formalizes the Quantum Virtual Machine (QVM) as a field operating system and control plane.

System-level orchestration does not execute computation and does not alter field-native execution semantics. Instead, it coordinates the lifecycle of execution domains, assigns execution modes, governs transitions, manages composition, and enforces system-wide policies across heterogeneous field executions. Orchestration operates over governed structures—domains, workflows, contracts, and interfaces—rather than over operators or execution states.

The chapter establishes orchestration primitives for admission, scheduling of workflows, domain lifecycle management, and inter-domain coordination at scale. It treats QVM as an infrastructural layer analogous to an operating system, while preserving the fundamentally non-procedural and non-projective nature of field-native computation.

By formalizing QVM as a field OS and control plane, this document completes the transition from isolated field execution to fully orchestrated field computing systems. It provides the conceptual foundation for deploying, managing, and governing large-scale field-native infrastructures on both native and classical computational substrates.

1 Orchestration Primitives and Responsibilities

System-level field orchestration defines the highest layer of governance within the Quansistor Field Computing architecture. Its purpose is not to execute computation, nor to influence operator-level behavior, but to coordinate, configure, and sustain field-native execution at scale. This chapter introduces the core orchestration primitives and clarifies the responsibilities of QVM as a field operating system and control plane.

Orchestration primitives are governance constructs that operate over execution domains, workflows, and policies rather than over execution states or operators. They define how execution domains are admitted into the system, how they are configured, how they interact, and how they evolve over time. These primitives do not introduce new execution semantics; they act exclusively on the governance structures established in previous layers.

The primary orchestration responsibilities of QVM include domain admission and retirement, execution mode assignment, lifecycle management, workflow coordination, and enforcement of

system-wide policies. QVM determines *which* domains may exist, *under what conditions* they operate, and *how* they may be composed, but it does not determine *how execution proceeds* within those domains.

A fundamental orchestration primitive is domain admission. Admission is the act by which a new execution domain is instantiated with a specified governance configuration, including isolation mode, active contracts, execution regime, and projection permissions. Admission is subject to system-level constraints such as capacity, policy compliance, and compatibility with existing domains.

Lifecycle management is another central responsibility. Execution domains may be created, suspended, resumed, reconfigured, composed, or terminated through orchestrated acts. These acts are discrete and declarative, and they must preserve all active invariants. Orchestration does not intervene in execution trajectories; it manages the existence and configuration of execution contexts.

Workflow coordination operates at a higher temporal and structural level. Orchestration arranges how multiple domains and execution regimes participate in a larger computational process. It governs phase ordering, inter-domain dependencies, and execution mode transitions without resolving nondeterminism or selecting execution outcomes. Workflows are thus sequences of governed configurations rather than sequences of executed instructions.

Importantly, orchestration does not equate to resource scheduling in the classical sense. QVM does not allocate CPU cycles or memory pages. Instead, it governs admissibility of execution contexts with respect to available infrastructure and policy constraints. Resource considerations influence admission and configuration decisions, not execution behavior.

By defining orchestration primitives and responsibilities in this manner, this chapter establishes QVM as a non-intrusive but authoritative system layer. QVM coordinates field-native computation without collapsing it into procedural control, enabling scalable, auditable, and policy-aware field computing systems. This foundation prepares subsequent chapters on domain lifecycle management, workflow scheduling, and system stability.

2 Domain Lifecycle Management

Domain lifecycle management is a core responsibility of system-level field orchestration. While execution semantics determine how computation evolves within a domain, lifecycle management determines when domains come into existence, how long they persist, and under what conditions they may change or cease to exist. This chapter formalizes the lifecycle of execution domains as a sequence of governed states managed by QVM.

The lifecycle of an execution domain begins with admission. Domain admission is a governed act that instantiates a new domain with a specified governance configuration. This configuration includes isolation mode, active execution contracts, execution regime, admissibility modulation parameters, and projection permissions. Admission is permitted only if the proposed configuration is compatible with system-wide policies and does not violate existing isolation or capacity constraints.

Once admitted, a domain enters an active state in which field-native execution proceeds under its configured governance envelope. QVM does not intervene in execution trajectories during this phase. Its role is limited to monitoring compliance at the governance boundary, ensuring that the domain remains within the scope of its declared obligations.

Domains may transition into suspended states. Suspension is a governance-level act that temporarily halts admission of new execution trajectories without altering the internal field configuration. Suspension preserves auditability and isolation guarantees while allowing system-

level reconfiguration, resource rebalancing, or policy evaluation. Importantly, suspension does not imply rollback or partial execution; it freezes admissibility rather than execution state.

Reconfiguration is a controlled lifecycle transition in which the governance parameters of a domain are modified. This may include changes to execution mode, contract activation, admissibility constraints, or interface permissions. Reconfiguration is admissible only if all invariants required by both the previous and updated configurations are preserved. Boundary contracts ensure continuity across reconfiguration events.

Domain composition and decomposition are also lifecycle acts. Composition creates a new governed domain that subsumes existing ones under a higher-level governance context, while decomposition dissolves such a composite into its constituents. These operations are declarative and preserve isolation and audit guarantees through contract enforcement.

Termination marks the end of a domain's lifecycle. Termination is a governed act that withdraws admissibility for further execution within the domain. Terminated domains may produce terminal field configurations subject to projection or archival policies, but no further execution occurs. Termination does not invalidate auditability; completed domains remain verifiable after their operational lifetime.

Throughout the lifecycle, responsibility and accountability are attached to domain configuration rather than to individual execution events. Lifecycle transitions are auditable governance actions, providing a clear record of how and why domains were created, modified, or retired.

By formalizing domain lifecycle management, this chapter establishes QVM as a system-level steward of field-native execution contexts. Lifecycle control enables long-lived, adaptive, and policy-aware field computing systems without intruding into execution semantics. This foundation prepares the next chapter on workflow scheduling and inter-domain coordination at system scale.

3 Workflow Scheduling Across Domains

Workflow scheduling in QVM operates at the level of governance rather than execution. Unlike classical schedulers, which assign processor time or order instructions, QVM scheduling coordinates the temporal and structural arrangement of execution domains and their governance configurations. This chapter formalizes workflow scheduling as orchestration of domain phases rather than control of computational steps.

A workflow is defined as an ordered or partially ordered structure of governed execution phases spanning one or more execution domains. Each phase corresponds to a domain operating under a specific governance configuration, including execution mode, active contracts, and admissibility constraints. Scheduling determines when phases become admissible, how they relate temporally, and under what conditions transitions between phases may occur.

Workflow scheduling does not select execution trajectories. It does not resolve nondeterminism, prioritize operators, or intervene in field evolution. Instead, it regulates the admissibility of phases and their transitions. A phase becomes active when its governance conditions are satisfied and inactive when those conditions are withdrawn or superseded.

Cross-domain scheduling coordinates dependencies between domains. Such dependencies may include prerequisite completion of phases, satisfaction of boundary contracts, or availability of projected results from other domains. These dependencies are expressed declaratively as governance constraints rather than as procedural synchronization.

Parallel workflows are first-class citizens. Multiple workflows may be active concurrently, each spanning distinct or overlapping sets of domains. Isolation guarantees ensure that concurrent workflows do not interfere unless explicitly composed through governed interfaces. Scheduling therefore scales naturally with intrinsic parallelism without introducing contention at the execution

level.

Conditional scheduling is also supported. Governance policies may specify that activation of a workflow phase depends on the satisfaction of audit conditions, equivalence checks, or probabilistic criteria. These conditions are evaluated at governance checkpoints and do not require runtime monitoring of execution internals.

Importantly, workflow scheduling is reversible and adaptive. Phases may be suspended, reordered, or reconfigured through governance acts without invalidating execution semantics or auditability guarantees. Adaptation occurs by modifying admissibility envelopes rather than by manipulating execution state.

By defining workflow scheduling as orchestration of domain phases, this chapter establishes a system-level notion of scheduling compatible with field-native computation. Scheduling becomes a matter of structuring time and dependency at the governance layer, enabling complex, adaptive, and auditable computational processes without reintroducing procedural control. This formulation prepares the next chapter on resource governance and system stability.

4 Resource Governance without Execution Control

Resource governance in QVM addresses how field-native computation coexists with finite physical and infrastructural resources without introducing execution-level control. This chapter formalizes resource governance as a boundary condition on domain admission and configuration rather than as a mechanism for regulating execution behavior.

In contrast to classical systems, where resource management is tightly coupled to execution scheduling, QVM decouples resource considerations from execution semantics. QVM does not allocate processor cycles, memory pages, or bandwidth to individual operations. Instead, it governs the admissibility and configuration of execution domains with respect to available resources and system-level policies.

Resources are treated as governance constraints. An execution domain may be admitted, configured, or expanded only if sufficient resources are available to sustain its declared governance obligations. These obligations may include isolation guarantees, auditability requirements, execution mode constraints, or projection fidelity. Resource sufficiency is evaluated at admission and reconfiguration points, not continuously during execution.

This approach prevents resource contention from influencing execution trajectories. Because execution proceeds independently within admitted domains, resource scarcity cannot cause partial execution, forced preemption, or priority inversion at the execution level. If resources become insufficient, QVM responds by restricting admissibility—suspending domain admission, delaying workflow phase activation, or denying reconfiguration requests—rather than by intervening in execution.

Resource governance also enables predictable system behavior. By ensuring that admitted domains are provisioned according to their declared obligations, QVM guarantees that execution semantics remain stable throughout a domain's lifecycle. Deterministic domains retain their guarantees, probabilistic domains retain their exploratory freedom, and auditability contracts remain enforceable.

Importantly, resource governance is policy-driven and declarative. Governance policies may encode priorities, quotas, or fairness constraints at the level of domain admission and workflow scheduling. These policies shape which domains and workflows are permitted to exist concurrently, not how execution unfolds within them.

Resource release is handled through lifecycle transitions. Suspension, reconfiguration, or termination of domains frees governance capacity, enabling new admissions or workflow activations.

Resource accounting is therefore tied to domain existence rather than to execution events.

By governing resources at the boundary of execution rather than within it, QVM achieves a clean separation between physical constraints and computational semantics. Resource governance becomes a system-level responsibility that preserves the integrity, auditability, and composability of field-native execution. This formulation prepares the final chapter on fault tolerance and system stability, completing the orchestration layer.

5 Fault Tolerance and System Stability

Fault tolerance in QVM-governed field systems is not achieved through recovery mechanisms, redundancy protocols, or execution-level intervention. Instead, it emerges as a structural property of governed execution domains, isolation guarantees, and lifecycle orchestration. This chapter formalizes fault tolerance and system stability as consequences of governance design rather than as reactive control strategies.

A fault is defined as any event that prevents an execution domain or workflow phase from continuing under its declared governance obligations. Such events may include contract incompatibility, resource insufficiency, policy violation, or infrastructural failure. Crucially, faults do not manifest as corrupted execution states or partial computations. They appear as governance-level conditions that withdraw admissibility.

Because execution is preventive rather than corrective, faults do not require rollback or checkpoint restoration. Forbidden or unsupported transitions simply do not occur. When a fault condition arises, QVM responds by adjusting governance structures: suspending domains, denying reconfiguration, isolating affected workflows, or terminating execution contexts. Execution semantics remain intact and lawful at all times.

Isolation guarantees play a central role in fault containment. Faults arising within one execution domain cannot propagate to others unless explicitly permitted by compositional interfaces. Even in composed systems, conjunctive contract enforcement ensures that faults are blocked at governance boundaries. This containment property holds independently of concurrency, execution mode, or nondeterminism.

System stability is achieved through admission control and lifecycle governance. By ensuring that only domains with compatible configurations and sufficient resources are admitted, QVM prevents overload-induced instability. Stability is preserved by refusing to admit or activate execution contexts that would compromise existing guarantees, rather than by degrading execution quality or forcing preemption.

Graceful degradation is expressed through governance adaptation rather than execution manipulation. When system conditions change, QVM may suspend workflows, delay phase transitions, or reassign governance priorities. These adaptations occur at discrete governance checkpoints and do not interfere with ongoing execution trajectories within admitted domains.

Importantly, fault tolerance does not imply fault invisibility. All lifecycle transitions, suspensions, and terminations are auditable governance events. System operators and higher-level institutions may analyze fault patterns, responsibility attribution, and policy effectiveness without inspecting execution internals or compromising execution integrity.

By defining fault tolerance and system stability as governance properties, this chapter completes the orchestration layer of QFC. Field-native computation becomes resilient not because it can be repaired mid-flight, but because it is governed in a way that prevents unlawful states from arising. QVM thus fulfills the role of a field operating system: sustaining long-lived, adaptive, and trustworthy computational systems without collapsing execution into procedural control.