High Performance Quantum-Centric Supercomputing: A Working Implementation of Heterogeneous Orchestration across QPU, NPU, GPU, CPU, and Other Tiers

Technical Paper
March 16, 2026

Kevin D. Johnson, MBA, MAcc, MSGTD, MAT

DOI: 10.5281/zenodo.19888562

***Abstract*—IBM's March 2026 reference architecture for quantum-centric supercomputing concludes that traditional batch schedulers require a fundamentally different approach for heterogeneous quantum-classical resources. The present paper extends the reference architecture's vision with proof-of-concept evidence that IBM Spectrum Symphony, a service-oriented dynamic compute platform, can address the orchestration requirements the reference architecture identifies. Fifteen demonstrations spanning twelve silicon architectures, six compute tiers and two network fabrics orchestrate QPU, NPU, GPU, CPU and mainframe as peer resource types under a single scheduling domain. Symphony's orchestrator consumes per-resource metrics reported through the External Load Information Manager (ELIM), a generalization of Songnian Zhou's 1987 load index. ELIM reports arbitrary classical metrics from every resource type; the platform makes placement decisions through the same algorithm regardless of whether the resource is quantum, neuromorphic, GPU or mainframe. All scheduling-relevant QPU metrics are classical by nature, and the quantum demonstrations provide direct evidence of platform-driven scheduling, routing identical Qiskit code to GPU simulation or IBM Quantum hardware based on ELIM-reported resource state. Four quantum applications with 2,560 hyperparameter combinations, an event-driven closed-loop quantum-neuromorphic workflow and multi-modal edge-to-cloud sensor integration are among the demonstrations. The orchestration framework applies equally to AI and LLM workloads, with five demonstrations involving LLM inference as a SOAM service tier, including semantic routing across model tiers and GPFS-based KV cache sharing. The implementation lineage from Zhou's dissertation through Platform Computing (1992), Platform Symphony (2006) and IBM Spectrum Symphony (2012) spans four decades of production deployment at the largest financial institutions in the world. The earliest quantum-classical demonstration with Symphony predates the reference architecture by several weeks, indicating the capability is already achievable with existing platforms.**

I. Introduction

Richard Feynman observed in 1981 that nature is not classical and that simulating quantum mechanical systems on classical hardware imposes an exponential penalty no amount of classical processing power can overcome [1]. Forty-five years of engineering effort have moved computing architectures toward honoring Feynman's observation. IBM's March 2026 reference architecture for quantum-centric supercomputing (QCSC) represents the first industry formalization of the discipline, defining three phases, four architectural layers and five use cases [2]. The reference architecture observes that Slurm "has no native concept of quantum resources" and introduces the Quantum Resource Management Interface (QRMI) as a SPANK plugin to extend Slurm's capabilities toward QPU integration [2]. The need for QRMI reveals what the present paper terms an *orchestration gap* in current HPC schedulers, a gap the reference architecture addresses through plugin extension and the present paper addresses through an existing service-oriented dynamic compute platform.

The reference architecture's own analysis concludes that traditional batch schedulers "generally do not perform joint optimization of heterogeneous resources" and that "quantum-classical heterogeneous resources require a fundamentally different approach" [2]. The gap exposed by QRMI's necessity is structural, not merely technical. HPC scheduling has embedded since its origins a design assumption that compute resources differ in degree rather than in kind. Schedulers built on degree-level difference can handle faster GPUs or larger node counts but cannot natively represent a resource whose fundamental characteristics have no analogue in the classical model. A QPU's fidelity decays over time, its measurements are subject to shot noise and its calibration drifts between jobs. The resource is not merely different in size but different in nature.

Genuinely heterogeneous orchestration requires a framework designed from the start for resources that differ in kind. A framework meeting the requirement maintains unity across the scheduling domain without imposing uniformity on the resources within it; each resource type retains its own metrics, its own temporal cadence and its own failure semantics. The first successful implementation of such a framework in distributed systems scheduling traces to Songnian Zhou's 1987

doctoral dissertation at UC Berkeley, which studied dynamic load balancing in loosely-coupled heterogeneous distributed systems [3]. Zhou examined how to treat a network of VAX-11/780s, 750s, 8600s and Sun workstations as a single managed resource pool without erasing the differences among them. Zhou's technical contribution was the *load index*, a per-host metric capturing the actual state of each machine and feeding the result into a scheduling algorithm making placement decisions based on what each host actually was rather than what a common abstraction prescribed. Zhou demonstrated that simple dynamic load balancing algorithms using initial job placement alone can improve performance significantly without introducing system instability [3], [4].

The implementation lineage from Zhou's load index to the system presented in the present paper is direct and unbroken. Zhou founded Platform Computing in 1992, where the load index concept evolved into the Utopia load-sharing facility at the University of Toronto [5] and then into Platform LSF (Load Sharing Facility). Platform Computing later developed Symphony as a Service-Oriented Application Manager (SOAM) framework. After IBM acquired Platform Computing in 2012, Symphony became IBM Spectrum Symphony [6], [7]. Symphony is a dynamic compute platform architecturally distinct from batch schedulers; Symphony's fundamental unit of work is a service invocation within a composed service graph rather than a job submitted to a queue. Symphony's External Load Information Manager (ELIM) is the direct descendant of Zhou's load index. Zhou's index reported CPU utilization and queue length per heterogeneous host; ELIM reports `qpu_fidelity`, `npu_inference_latency`, `gpu_util` and `zos_batch_queue` per heterogeneous resource type. The implementation scaled over four decades to accommodate progressively greater degrees of heterogeneity, from different CPU models to different computational paradigms.

The present paper offers empirical evidence that an existing service-oriented dynamic compute platform can address the orchestration requirements the reference architecture identifies. Using IBM Spectrum Symphony's Service-Oriented Application Manager (SOAM) and IBM Storage Scale (GPFS) over RDMA on commodity and enterprise hardware, the author demonstrates 15 working heterogeneous systems spanning 12 silicon architectures, 6 compute tiers and 2 network fabrics, all orchestrated under a single scheduling domain treating QPU, NPU, GPU, CPU and mainframe as peer resource types. The reference architecture is an architectural proposal; the present paper provides proof-of-concept demonstrations on a platform already proven at production scale, extending the reference architecture's vision with working implementations. The earliest quantum-classical demonstration predates the reference architecture by several weeks, indicating that the capability the reference architecture formalizes was already achievable with existing platforms. Additional compute tiers not addressed by the reference architecture, including neuromorphic processors, edge sensors, mainframe batch systems and software-defined networking, integrated without architectural modification because the framework accommodates difference by design.

The contributions of the paper are fivefold. First, the paper presents a working implementation of QPU, NPU, GPU, CPU and mainframe as co-scheduled resource types under a single workload manager, with the earliest demonstration predating the reference architecture by 11 weeks. Second, the paper describes an event-driven closed-loop quantum-neuromorphic workflow in which neuromorphic edge inference triggers quantum retraining autonomously; the scheduling framework holds four different temporal cadences in a unified workflow without forcing a common clock. Third, the paper provides empirical evidence that additional compute tiers, including neuromorphic, edge sensor and mainframe tiers, integrate naturally into the quantum-classical orchestration model when the framework is designed for unity-in-distinction rather than resource homogeneity. Fourth, the paper demonstrates that an existing service-oriented dynamic compute platform can address the orchestration requirements the reference architecture identifies, since Symphony's ELIM natively reports arbitrary resource metrics as first-class scheduling inputs with no plugin or shim layer required between the scheduler and any resource type. Fifth, the paper demonstrates GPFS over RDMA as a unified coordination substrate for heterogeneous artifacts, with a single storage fabric carrying quantum circuits, neuromorphic spike logs, GPU tensors and mainframe records without requiring a common data model.

II. The Unity-in-Distinction Principle

A. The General Problem

Quantum-centric supercomputing presents a specific instance of a general architectural challenge recurring wherever systems must coordinate genuinely heterogeneous components. The challenge is how to achieve unity across difference without uniformity. The word "genuinely" carries weight here. Components differing in degree, a faster CPU alongside a slower CPU or a larger memory alongside a smaller memory, present a scheduling problem solvable by parameterization. A scheduler need only rank the components along a common axis and allocate accordingly. Components differing in kind present a fundamentally different problem. Quantum superposition and classical determinism do not occupy different positions on a shared continuum; the two operate according to different physical principles producing different categories of output subject to different failure modes. Spiking neuromorphic inference and continuous tensor flow do not differ merely in latency; the two embody different computational paradigms, one event-driven and milliwatt-scale, the other batch-oriented and kilowatt-scale. Batch mainframe settlement and edge sensor classification do not differ merely in speed; the two inhabit different temporal structures, one producing discrete settlement confirmations at fixed intervals, the other producing continuous observation streams with no natural boundary.

A framework capable of orchestrating all of these resources must accomplish two things simultaneously. The framework must maintain genuine unity, holding all resource types within

one scheduling domain, one workflow graph and one storage fabric so that a process spanning four tiers constitutes one orchestrated workflow rather than four independent jobs stitched together by glue code. The framework must also maintain genuine distinction, preserving each resource type's own metrics, temporal cadence, failure semantics and operational characteristics so that no resource type is forced into an abstraction erasing the properties making the resource effective. The difficulty lies in the simultaneity. Unity alone is achievable through homogenization, by reducing all resources to a common model. Distinction alone is achievable through isolation, by running separate schedulers for separate resource types. Holding both together without sacrificing either is the architectural problem.

The principle requiring both unity and distinction simultaneously has a concrete and verifiable history in distributed systems. Zhou's 1987 implementation at UC Berkeley demonstrated the principle in practice, and four decades of faithful implementation through Platform Computing, Platform LSF, Platform Symphony and IBM Spectrum Symphony have validated the approach at production scale in financial services and other industries processing billions of tasks daily. Other perspectives on heterogeneous coordination may exist in adjacent fields, but the question any alternative must answer is which framework has been implemented, tested and scaled across four decades of progressively increasing heterogeneity. The engineering question is how to build systems embodying the principle effectively, and the Zhou lineage provides a demonstrated answer.

B. Two Recurring Errors

Two structural errors recur in attempts to solve the heterogeneous coordination problem. Each error resolves one half of the tension at the expense of the other.

Homogenization treats all resources as instances of a common abstraction, achieving unity by erasing distinction. Slurm's Generic Resource (GRES) model is the clearest contemporary example [8]. Under GRES, a GPU is a counted item attached to a node, characterized by a type label and a quantity. The model captures how many GPUs a node possesses but cannot express what a GPU is doing at a given moment, how its memory utilization relates to its compute utilization, how close it is to thermal throttling or how its tensor throughput compares to its floating-point throughput under the current workload. The model works for resources differing in degree because a faster GPU and a slower GPU can be meaningfully compared along a shared axis of performance. The model fails for resources differing in kind because a QPU's characteristics, including fidelity decay over time, shot noise per measurement, calibration drift between jobs and queue depth at any given moment, have no analogue in the tuple of count, memory and cores through which GRES represents all resources.

The symptom of homogenization is the middleware shim. When QRMI must be bolted onto Slurm to handle QPU resources, the shim exists because the scheduler's abstraction collapsed different natures into a single resource category [2], [8]. QRMI re-introduces the quantum-specific distinctions the GRES model erased, adding fidelity-aware scheduling, calibration-aware job placement and shot-budget management as plugin functionality layered onto a scheduler never designed to express such distinctions natively. NVIDIA's Slurm integration performs the same function for GPU-specific scheduling needs. Custom Kubernetes operators perform the same function for domain-specific resource types within container orchestration. Each shim is evidence of the same architectural failure. The underlying abstraction achieved unity by erasing distinction, and each new resource type differing in kind requires a corrective layer attempting to restore the distinctions the abstraction removed.

Kubernetes resource quotas and cloud provider instance types follow the same homogenizing logic at different levels of the stack. A Kubernetes pod requests CPU and memory as scalar quantities; a cloud instance type bundles CPU cores, memory and optional accelerators into a fixed configuration. Both models assume resources occupy positions along shared axes of quantity. Both models struggle when a resource's essential characteristics cannot be expressed as quantities along shared axes.

Isolation runs separate orchestration domains for separate resource types, achieving distinction by sacrificing unity. Kubernetes manages containers. Slurm manages HPC batch jobs. A vendor-specific SDK manages QPU circuit submissions. Each scheduler optimizes for its own domain. Each scheduler preserves the characteristics of the resources under its control. Cross-domain workflows, however, have no single point of orchestration. A workflow requiring QPU sampling followed by GPU error mitigation followed by CPU diagonalization must either pass through a message queue connecting the three schedulers or rely on an external workflow manager, such as Airflow or Prefect, stitching the tiers together from outside. The individual natures are preserved but the unity of the overall process is lost. The symptom of isolation is the workflow that works within a single tier but fragments across tiers, requiring manual coordination, inter-scheduler messaging or bespoke glue code at every tier boundary.

The two errors are not independent but are structural mirrors of each other. Homogenization sacrifices distinction for the sake of unity. Isolation sacrifices unity for the sake of distinction. Both treat unity and distinction as competing objectives requiring a tradeoff, as though a system can have one or the other but not both. The history of heterogeneous system design is substantially a history of oscillation between these two errors, with each generation of systems correcting one error by committing the other. Slurm corrected the fragmentation of earlier multi-scheduler environments by unifying HPC scheduling under one resource manager but committed the homogenization error by reducing all resources to GRES counts. Multi-cloud orchestration platforms corrected the homogenization of single-vendor environments by preserving each cloud's native resource model but committed the isolation error by requiring cross-cloud coordination layers that no single scheduler controls.

C. The Resolution

The architectural resolution of both errors is a framework holding different computational natures in unity without collapsing their differences. Each resource type retains its own metrics, its own failure semantics and its own temporal cadence. All resource types participate in one scheduling domain, one workflow graph and one storage fabric. No resource type is reduced to another's abstraction. No resource type is isolated from the others.

The resolution is not a compromise between homogenization and isolation but a rejection of the assumption forcing the choice between them. The assumption is that unity requires uniformity, that holding things together requires making them the same. A framework rejecting the assumption can measure each resource according to its own nature while feeding all measurements into a single scheduling algorithm operating over a single resource domain. The measurements need not share a common unit. The scheduling algorithm need not understand the internal semantics of each measurement. The algorithm needs only to consume typed metrics from each resource and make placement and composition decisions on the basis of those metrics. The framework knows how each resource reports its state and accepts work. The framework does not need to know what each resource is in terms reducible to a shared abstraction.

D. The Zhou Lineage as Implementation Foundation

The first successful implementation of unity-in-distinction in distributed systems scheduling traces to Songnian Zhou's 1987 doctoral dissertation at UC Berkeley [3]. Zhou's research problem was practical. Zhou sought to treat a heterogeneous network of VAX-11/780s, VAX-11/750s, VAX-8600s and Sun workstations as a single managed resource pool without erasing the differences among them. The machines differed not merely in speed but in I/O characteristics, memory architecture and workload response profiles. A scheduling algorithm treating them as interchangeable nodes differing only in clock speed would misplace jobs on hosts whose actual state diverged from what a uniform model predicted.

Zhou's technical contribution was the *load index*, a per-host metric capturing the actual state of each machine, including queue length, CPU utilization and I/O load, and feeding the result into a scheduling algorithm making placement decisions based on what each host actually was rather than what a common abstraction prescribed [3], [4]. Zhou demonstrated that simple dynamic load balancing algorithms using initial job placement alone can improve performance significantly without introducing system instability [3]. The significance of the result extends beyond the performance improvement. Zhou demonstrated that a scheduling framework measuring each resource according to its own operational characteristics could maintain a unified resource pool across genuinely heterogeneous hosts. The framework did not require the hosts to conform to a common model. The framework required only that each host report its own state in its own terms.

Zhou's advisor at Berkeley was Domenico Ferrari, whose research group developed load index theory through a series of empirical studies in the mid-1980s [9], [10]. Ferrari and Zhou co-authored foundational work on empirical load index evaluation, establishing the trace-driven methodology employed in the dissertation [11], [12], [13]. The Berkeley research was funded by DARPA under ARPA Order No. 4871, monitored by the Space and Naval Warfare Systems Command, and used traces from Bell Labs and Lawrence Berkeley National Laboratory [3]. The national laboratory ecosystem supporting Zhou's research is the same ecosystem IBM's QCSC reference architecture references in its discussion of RIKEN, Fugaku and national-scale HPC deployment [2].

The technical lineage from Zhou's dissertation to the system presented in the present paper is direct and unbroken, passing through four stages of development.

Utopia (1988–1993). Zhou moved to the University of Toronto after completing the Berkeley dissertation and developed the Utopia load-sharing facility for large heterogeneous distributed computer systems [5]. Utopia extended the load index implementation from Zhou's dissertation into a production system managing heterogeneous resources across a large university computing environment. Zhou, Zheng, Wang and Delisle published the Utopia design in *Software: Practice and Experience* in 1993 [5]. Utopia was the direct predecessor of Platform LSF.

Platform LSF (1992). Zhou founded Platform Computing in 1992 to translate his distributed operating systems research into commercial cluster and grid management products [14], [7]. Platform LSF (Load Sharing Facility) implemented heterogeneous scheduling at commercial scale. Financial services, pharmaceutical research and manufacturing adopted LSF as the scheduler for environments where resources differed in more than CPU speed. Zhou confirmed in interviews that his Berkeley research directly informed Platform Computing's architecture [14], [7].

Platform Symphony (2006). Platform Computing released Symphony as a Service-Oriented Application Manager (SOAM) framework extending the LSF scheduling model with service graph composition, consumer hierarchies and ELIM [15]. ELIM generalized Zhou's load index from per-host CPU metrics to per-type arbitrary metrics. Zhou's load index reported CPU utilization and queue length for heterogeneous hosts. ELIM reports arbitrary metrics for arbitrary resource types, consuming the metrics through a uniform scheduling interface without requiring any metric to conform to a common schema. The generalization preserved the original design, with each resource measured according to its own operational state, while extending the implementation to resource types Zhou's 1987 system never contemplated.

IBM Spectrum Symphony (2012–present). IBM acquired Platform Computing in 2012, and Symphony became IBM Spectrum Symphony [6], [7]. Production deployments at major financial institutions demonstrated the framework's capacity to manage over 100,000 concurrent tasks under heterogeneous scheduling [15]. The

present work extends the same framework to QPU, NPU, GPU, CPU and mainframe resource types, with ELIM reporting `qpu_fidelity`, `npu_inference_latency`, `gpu_util` and `zos_batch_queue` in the same scheduling domain.

The lineage from Zhou's 1987 load index to Symphony's 2026 ELIM is genealogical rather than analogical. The same researcher who built the first load index implementation in a Berkeley dissertation founded the company whose products carried the implementation to commercial scale, and the acquisition of that company by IBM produced the framework on which the present quantum-neuromorphic system runs. The implementation scaled from VAX workstations to QPUs over 39 years. The principle the implementation embodies, measuring each resource according to its own nature and scheduling on the basis of those measurements within a unified domain, is a principle Zhou's work successfully instantiated in distributed systems rather than a principle Zhou's work originated. The principle's validity is attested by the implementation's 39-year success. The principle's origin lies deeper than any single implementation.

E. A Structural Property of Heterogeneous Systems

The 39-year success of Zhou's implementation lineage raises a question beyond engineering history. The question is whether unity-in-distinction is a contingent engineering choice, one approach among several that happened to succeed, or a structural property of heterogeneous systems, a requirement imposed by the nature of the problem itself.

Three lines of evidence support the structural interpretation.

First, the implementation has scaled without fundamental redesign across four decades of increasing heterogeneity. Zhou's 1987 system coordinated machines differing in CPU speed, I/O characteristics and memory architecture. The present system coordinates resources differing in computational paradigm, including quantum superposition, neuromorphic spiking, continuous tensor flow, deterministic batch processing and continuous sensor streaming. The degree of heterogeneity has increased by orders of magnitude. The scheduling principle governing successful coordination has not required revision. An implementation surviving such radical increases in the scope of difference to which the implementation applies is more parsimoniously explained by the correctness of the underlying principle than by fortunate engineering choices compounding over four decades.

Second, three independent paths converge on the same requirement. Zhou's 1987 research arrived at a working implementation from the load balancing side, building a system in which heterogeneous hosts participated in one resource pool without erasure of their differences [3]. The present work arrived at the same operational model from the architectural side, recognizing that a scheduler designed for resources differing in kind would naturally accommodate quantum resources without modification. IBM Research's 2026 reference architecture arrived at the same conclusion from the engineering side, analyzing Slurm's limitations and determining that heterogeneous quantum-classical orchestration requires unified scheduling respecting the distinct characteristics of each resource type [2]. The three paths originate in different decades, different institutional contexts and different motivations. Convergence of three independent paths on the same structural requirement is difficult to explain as coincidence and straightforward to explain as a property of the problem.

Third, the two errors identified in Section II-B, homogenization and isolation, exhaust the logical space of failure modes for heterogeneous coordination. A framework either collapses the distinctions among resources or preserves the distinctions by separating the resources into distinct orchestration domains. No third category of failure exists because the two errors exhaust the ways a system can violate the simultaneous requirements of unity and distinction. Any framework avoiding both errors necessarily holds distinct resources in unified orchestration without collapsing their differences. The exhaustiveness of the error space indicates that unity-in-distinction is not one option among many but the only resolution available to any system confronting genuine heterogeneity.

The structural interpretation carries implications beyond quantum computing. The same requirement governs how neuromorphic chips, event-driven at milliwatt scale with sub-millisecond latency, participate alongside GPUs, batch-oriented at kilowatt scale with throughput optimization. The same requirement governs how mainframe settlement, running COBOL at sub-second batch intervals, coexists with edge sensor classification producing continuous 128-byte observation streams. The same requirement governs how a storage fabric can carry quantum circuit definitions, neuromorphic spike logs, GPU tensor checkpoints and mainframe transaction records without forcing a common data model. The orchestration challenge is always the same challenge, regardless of which specific resource types a given system coordinates. Unity without uniformity and distinction without isolation are not quantum-specific requirements. The requirements belong to heterogeneous systems as such.

F. The Architectural Correction Thesis

The author has argued in prior work that the primary correction needed in AI deployment is architectural rather than economic [16]. Heterogeneous workload-aware platforms routing the right task to the right hardware address the gap the AI market has struggled to close. The quantum case extends the thesis. The hardware for quantum-centric supercomputing exists in IBM Heron processors and RIKEN's Fugaku classical supercomputer [17], [18]. The algorithms exist in SKQD, SQD and tensor error mitigation [17], [19]. The orchestration platform exists in production-grade heterogeneous schedulers with decades of deployment history [6], [15]. The missing element has not been infrastructure but recognition, the recognition that the quantum-classical orchestration problem is an instance of a well-understood architectural pattern an existing platform already solves.

The timeline of the demonstrations presented in Section VI provides empirical support for the thesis. The same architec-

tural principle predicting that heterogeneous AI deployment would require workload-aware orchestration also predicted that quantum-classical integration would require a scheduler built for unity-in-distinction. Both predictions preceded their respective industry formalizations. The predictive power of the principle across two distinct domains, AI deployment economics and quantum-centric supercomputing, is additional evidence for the structural interpretation advanced in Section II-E. A principle predicting outcomes in domains as different as AI market correction and quantum-classical scheduling operates at a level of generality exceeding any single engineering tradition.

III. Related Work

A. IBM Quantum-Centric Supercomputing Reference Architecture

Seelam et al. published the first industry reference architecture for quantum-centric supercomputing in March 2026 [2]. The architecture defines three phases. Phase 1 (2026–2027) positions the QPU as a co-processor to classical HPC. Phase 2 (2028–2030) envisions heterogeneous integration of quantum and classical resources under unified orchestration. Phase 3 (2031–2034) contemplates multi-tenant execution environments with shared quantum-classical infrastructure. The architecture further specifies four layers (hardware, orchestration, middleware and applications) and five use cases spanning batch-time to near-time quantum-classical coupling.

The reference architecture proposes QRMI, a SPANK plugin for Slurm, as the mechanism for integrating QPU resources into HPC scheduling [2]. QRMI provides resource allocation, job scheduling, vendor-portable interfaces through a Quantum Systems API (QSA), cloud bursting support and multi-tenancy credential mediation [2]. The approach preserves Slurm's existing resource model and adds QPU support through a plugin rather than redesigning the scheduler's abstraction layer. The reference architecture also acknowledges the convergence of HPC and cloud-native infrastructure, noting that container orchestration, microservices and service-based architecture on AI-focused HPC systems "is likely to provide a foundation for integrating quantum systems" [2]. The cloud-native platforms referenced in the architecture, including Kubernetes and OpenShift, are container orchestration frameworks rather than service-oriented HPC schedulers with per-resource-type metrics and heterogeneous temporal orchestration.

The present paper shares the reference architecture's diagnosis of the problem but differs on the solution. The reference architecture extends a batch scheduler through a plugin restoring the quantum-specific distinctions the batch model cannot express natively. The present paper demonstrates that an existing service-oriented dynamic compute platform, designed from its origins for resources differing in kind, addresses the orchestration gap without requiring plugin extension. No other service-oriented HPC platform comparable to Symphony exists in production. Ray serves ML workloads; Dask and Spark are data-parallel frameworks; Kubernetes orchestrates containers without per-resource-type scheduling metrics. The absence of alternatives in the service-oriented HPC space is itself significant. Section VIII discusses the convergence of the two approaches in detail.

B. Quantum-Classical Computational Results

Kirby et al. reported the first experimental demonstration of quantum computational advantage over classical sparse configuration interaction (SCI) solvers using a quantum-centric supercomputing architecture [17]. The experiment employed a 49-qubit Hamiltonian on IBM Heron R3 processors. Sparse Quantum-Classical Krylov Diagonalization (SKQD) found the exact ground-state energy where CIPSI, ASCI, HCI and TrimCI all failed to converge [17]. The closed-loop SQD implementation ran concurrently on Fugaku, with 72,000-node and 150,000-node classical systems operating alongside the quantum processor [17], [19].

The SKQD workflow consists of six logical stages, including Hamiltonian construction, quantum sampling, error mitigation, subspace diagonalization, refinement control and results aggregation. Each stage maps naturally to a SOAM service in Symphony's scheduling framework. Section IX discusses the mapping as a direction for future work.

C. Heterogeneous Distributed Scheduling

The foundational literature on heterogeneous distributed scheduling informs the architectural argument of the present paper.

Zhou's 1987 doctoral dissertation at UC Berkeley introduced load indices as per-host metrics capturing actual resource state, including queue length, CPU utilization and I/O load, rather than abstract counts [3]. Zhou demonstrated the approach on heterogeneous hosts spanning VAX-11/780, VAX-11/750, VAX-8600 and Sun workstation architectures. The main result established that simple dynamic algorithms using initial job placement alone improve performance significantly without system instability [3], [4]. Ferrari and Zhou further developed load index theory through a series of empirical studies at Berkeley [12], [13], [10]. The intellectual lineage from Zhou's dissertation to Platform LSF (1992), to Platform Symphony (2006) and to IBM Spectrum Symphony (2012) is direct [5], [6], [7]. Zhou himself confirmed in interviews that his 1980s distributed operating systems research directly informed Platform Computing's architecture [14], [7].

Slurm, developed by Yoo, Jette and Grondona in 2003, represents an alternative architectural approach [8]. Slurm's node-centric resource model treats non-CPU resources as generic counted items through the GRES mechanism. The model is extensible via SPANK plugins but fundamentally assumes resources differ in degree rather than in kind. QRMI's necessity as a Slurm plugin for QPU scheduling illustrates the limitation [2], [8].

Condor, developed by Litzkow, Livny and Mutka in 1988, occupies an intermediate position [20]. The ClassAd matchmaking mechanism is closer to heterogeneous-aware scheduling than Slurm's GRES model, since ClassAds can express

arbitrary resource attributes. Condor was designed for cycle-scavenging in distributed workstation pools rather than tightly-coupled heterogeneous workflows, however, and the matchmaking model does not provide the service graph composition or unified temporal orchestration the quantum-neuromorphic case demands.

D. Other Quantum-HPC Integration Approaches

Several frameworks address aspects of quantum-HPC integration without solving the fundamental scheduling problem of resources that differ in kind. Pilot-Quantum provides a task-based framework for managing quantum-classical task execution but does not address heterogeneous resource scheduling across compute tiers. XACC and QCOR offer a quantum-classical C++ compilation model solving the programming abstraction but not the resource orchestration problem. PennyLane's HPC integration enables differentiable quantum programming but assumes a homogeneous classical backend. Each approach solves a real problem at the programming model or task submission layer. None addresses the scheduling challenge the reference architecture identifies at the orchestration layer.

E. Neuromorphic Computing in HPC

Neuromorphic computing has entered HPC environments through research deployments at national laboratories. Intel's Loihi and Loihi 2 processors operate within Sandia National Laboratories' neuromorphic research systems. BrainChip's Akida AKD1000 provides commercial neuromorphic silicon with a CNN-to-SNN conversion pipeline enabling deployment of trained convolutional networks as spiking neural networks [21]. No prior work has co-scheduled neuromorphic and quantum resources under unified orchestration. The present paper demonstrates neuromorphic processors participating as peer resource types alongside QPUs, GPUs, CPUs and mainframe systems within a single scheduling domain.

IV. Architecture

Figure 1 presents the system architecture. Six compute tiers report ELIM metrics to the dynamic compute platform, which makes scheduling decisions and dispatches work through SOAM service graphs. Two network fabrics connect the tiers, and GPFS over RDMA provides the unified coordination substrate.

A. Symphony SOAM as Heterogeneous Orchestrator

Symphony's Service-Oriented Application Manager (SOAM) permits arbitrary resource types, each defined by custom metrics reported through ELIM [15]. No resource type receives privileged treatment within the scheduler. A QPU is one resource type among peers, each reporting metrics appropriate to its own nature.

Symphony's scheduling architecture separates metric reporting from scheduling decisions. ELIM scripts report per-resource metrics in whatever dimensions the resource demands. The dynamic compute platform, powered by Symphony's Enterprise Grid Orchestrator (EGO), consumes ELIM metrics and makes placement decisions based on resource availability, consumer policies and workload requirements. SOAM service graphs compose services across resource types, binding a QPU sampling service to a GPU error mitigation service to a CPU diagonalization service within one workflow definition [15]. Consumer hierarchies manage resource contention and multi-tenancy across all resource types simultaneously, providing per-consumer policies and fairness guarantees without requiring separate fairness mechanisms for each tier. The reference architecture places multi-tenant execution in Phase 3 (2031–2034); Symphony's consumer hierarchies deliver the capability in production today [2], [15].

HostFactory, Symphony's cloud-bursting mechanism, extends the heterogeneous model to elastic resource provisioning. A QPU backend available through IBM Quantum cloud is a burstable resource alongside GPU instances and CPU instances, all managed through one provisioning interface [15]. The mechanism eliminates the need for separate cloud integration layers per resource type.

B. ELIM as Generalized Load Index

ELIM generalizes Zhou's load index to arbitrary resource types [3], [15]. Each resource reports metrics capturing its actual operational state, and the platform consumes all metrics through a uniform interface without requiring any metric to conform to a common schema.

The metrics reported by each resource type in the present system illustrate the generalization. QPU resources report `qpu_fidelity`, `qpu_queue_depth` and `qpu_calibration_age`. NPU resources report `npu_model_loaded`, `npu_inference_latency` and `npu_power_mw`. GPU resources report `gpu_util`, `gpu_mem` and `tensor_ops_per_sec`. Mainframe resources report `zos_initiators` and `zos_batch_queue`. All metrics feed the same platform scheduling algorithm through the same ELIM interface. A scheduling decision placing a quantum sampling task on a QPU with high fidelity and low queue depth uses the same algorithmic framework as a decision placing a tensor operation on a GPU with available memory and low utilization.

C. GPFS as Coordination Substrate

GPFS over RDMA provides the unified storage fabric connecting all resource types [22]. The role of GPFS in the architecture extends beyond shared storage. GPFS serves as the coordination substrate through which heterogeneous artifacts move between tiers without requiring a common data model, a message broker or an intermediate serialization layer.

The artifacts carried by the storage fabric span four fundamentally different data paradigms. Quantum artifacts include circuit definitions in QPY format, bitstring samples in HDF5 and energy estimates produced by diagonalization services. Neuromorphic artifacts include spike logs, CNN-to-SNN conversion artifacts in FBZ format and 128-byte classified observations produced by Akida inference. GPU

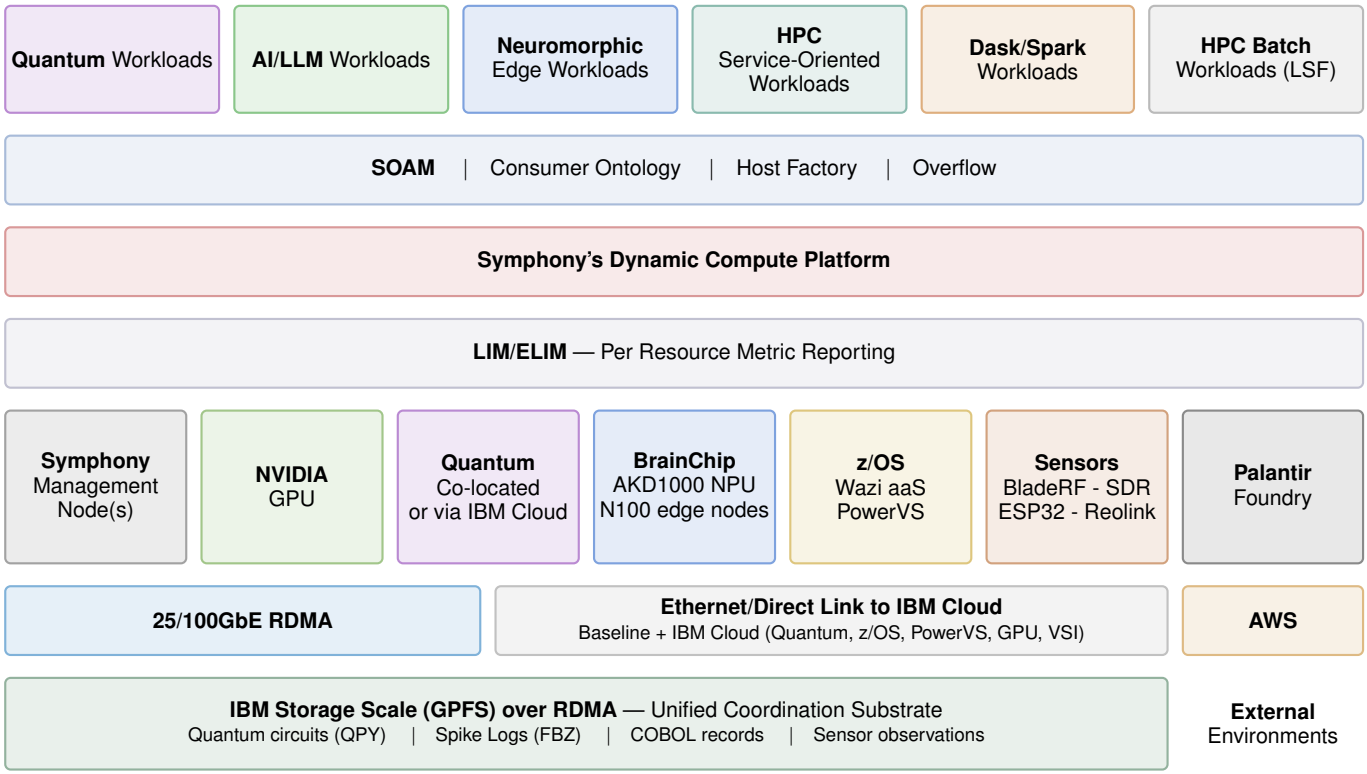

Fig. 1. System architecture. Workload types submit through SOAM to Symphony's Dynamic Compute Platform. LIM/ELIM reports per-resource metrics from seven hardware tiers. Two network fabrics (RDMA and Ethernet/Direct Link) connect local and cloud resources. GPFS over RDMA serves as the unified coordination substrate.

artifacts include tensor checkpoints, vLLM key-value cache files and model weights. Mainframe artifacts include COBOL transaction records and settlement confirmations produced by z/OS batch processing.

Extended attributes (xattrs) on the GPFS filesystem provide per-artifact metadata without requiring schema migration or a separate metadata service [22]. A quantum circuit definition carries its qubit count, gate depth and target backend as extended attributes. A neuromorphic classification result carries its model version, inference latency and confidence score. The metadata is queryable through standard filesystem interfaces, enabling downstream services to discover and filter artifacts without knowledge of the producing tier's internal formats.

GPFS over RDMA runs on commodity MikroTik 25GbE and 100GbE switching in the present deployment. Information Lifecycle Management (ILM) tiering moves artifacts between storage pools based on access patterns. Active File Management (AFM) enables cross-site portability for deployments spanning multiple locations [22]. No message brokers, no Redis instances and no custom coordination services participate in the data flow. The filesystem is the coordination layer.

V. Implementation

A. Hardware Platform

The system comprises commodity and enterprise hardware spanning six compute tiers, twelve distinct silicon architectures and two network fabrics. No component requires a national laboratory, a dedicated quantum data center or purpose-built HPC infrastructure. The full hardware inventory is summarized in Table I.

The twelve distinct silicon architectures are AMD Zen 2 (EPYC), AMD Zen 3 (Ryzen), Intel Alder Lake-N (N100), NVIDIA Ampere (RTX 3090/Ti), BrainChip AKD1000, superconducting transmon (IBM Quantum Heron R3), IBM z15/z16, IBM Power, Intel/Altera Cyclone V FPGA (bladeRF xA4), Cypress ARM9 (bladeRF FX3 controller), Espressif Xtensa LX7 (ESP32-S3) and Novatek MIPS (Reolink IP cameras). The sensor tier alone contributes four architectures because each sensor device carries its own embedded processor operating independently of the x86 hosts to which the sensor connects.

The platform spans four orders of magnitude in power consumption, from 30 mW for Akida neuromorphic inference to approximately 2 kW for the GPU host under full load. The platform spans five orders of magnitude in temporal cadence, from 622 μs for Akida classification to minutes for QPU queue time. All six compute tiers operate under one Symphony scheduling domain on commodity and enterprise hardware available through standard procurement channels.

TABLE I
HARDWARE PLATFORM SUMMARY.

Tier	Hardware	Temporal Cadence	Power	Silicon Architecture
Quantum	IBM Quantum Heron R3 (via IBM Cloud)	Minutes (queue) to μs (gates)	N/A (cloud)	Superconducting transmon
GPU Server	AMD EPYC 7702P + 5× RTX 3090/Ti	Seconds (training) to ms (inference)	∼2 kW	AMD Zen 2 + NVIDIA Ampere
Management	AMD Ryzen	Continuous	∼100 W	AMD Zen 3
Edge NPU (×10)	Intel N100 + BrainChip AKD1000	622 μs (inference), event-driven	∼30 mW (Akida)	Intel Alder Lake-N + AKD1000
Mainframe	IBM z/OS (Wazi aaS via IBM Cloud)	280 ms (settlement batch)	N/A (cloud)	IBM z15/z16
Power	IBM PowerVS (via IBM Cloud)	Seconds (batch)	N/A (cloud)	IBM Power
Sensors	bladeRF xA4 (Cyclone V FPGA + ARM9), RTL-SDR, ESP32-S3, Reolink IP cameras	Continuous (streaming)	mW to W	Cyclone V FPGA, ARM9, Xtensa LX7, Novatek MIPS

B. Resource Model and Service Graphs

Each compute tier registers as a Symphony resource type through a configuration defining the tier's ELIM metrics, reporting intervals and threshold-based alerts. QPU registration specifies fidelity, queue depth and calibration age as scheduling inputs. NPU registration specifies model loaded, inference latency and power draw. GPU registration specifies utilization, memory consumption and tensor throughput. Mainframe registration specifies initiator availability and batch queue depth. The registration process is identical across all tiers; only the metric definitions differ.

SOAM service graphs define workflows spanning multiple tiers as single orchestrated processes. A four-tier workflow composing QPU sampling, GPU error mitigation, NPU edge classification and CPU settlement is one service graph with typed edges connecting the services [15]. The scheduler resolves resource requirements at each node in the graph and dispatches work to the appropriate tier based on ELIM metrics at scheduling time. No external workflow manager such as Airflow or Prefect is required because the workflow definition and resource scheduling occur within the same framework.

Symphony's slot model differs from Slurm's node model in a manner significant for heterogeneous scheduling. Slurm allocates nodes, each node understood as a bundle of CPUs, memory and optional GRES counts [8]. Symphony allocates slots, each slot representing a unit of capacity within a resource type defined by the resource's own metrics [15]. The slot model accommodates resources that differ in kind because a slot on a QPU resource is defined by quantum-specific capacity measures while a slot on an NPU resource is defined by neuromorphic-specific capacity measures. The scheduler manages both through the same allocation mechanism without requiring the two slot definitions to share a common schema.

C. Network Architecture

Two network fabrics connect the compute tiers. The primary fabric is a 25GbE and 100GbE RDMA network running RoCEv2 over MikroTik switching. GPFS traffic, SOAM control messages and high-bandwidth data transfers between GPU, x86 and management tiers traverse the RDMA fabric. The fabric provides the low-latency, high-throughput transport required for GPFS coherence and SOAM service graph execution across tightly-coupled tiers. Standard Ethernet provides baseline connectivity, management access and internet connectivity for cloud-hosted resources including IBM Quantum backends, Wazi aaS and PowerVS.

A quantum-safe upgrade path is available through ML-KEM-1024 (CRYSTALS-Kyber) hybrid key exchange, deployable on the existing infrastructure without hardware modification [23].

VI. DEMONSTRATIONS

A. Overview

Fifteen demonstrations were conducted between December 29, 2025 and March 11, 2026. Each demonstration ran on commodity and enterprise hardware under Symphony's SOAM scheduling framework. Table II enumerates all fifteen demonstrations with the date, description and compute platforms involved. Four demonstrations are detailed in Sections VI-C through VI-F.

B. Platform Summary

Across the fifteen demonstrations, six compute tiers participate as Symphony resource types. QPU resources (IBM Quantum Heron R3 via IBM Cloud) appear in demonstrations 5 and 10. NPU resources (BrainChip AKD1000 on Intel N100) appear in demonstrations 6, 8, 9, 10, 11, 13, 14 and 15. NVIDIA GPU resources appear in ten demonstrations, running on IBM Cloud in seven (1, 2, 3, 4, 7, 10, 12) and locally on the AMD EPYC 7702P server in three (6, 11, 13). x86 resources appear in all fifteen demonstrations. IBM Power resources (PowerVS via IBM Cloud) appear in demonstration 3. IBM z/OS resources (Wazi aaS via IBM Cloud) appear in demonstration 7. Sensor resources appear in demonstration 14. Palantir Foundry (hosted on AWS) participates as an endpoint in demonstrations 1 and 14. GPFS serves as the coordination substrate in all fifteen demonstrations. IBM Cloud services appear in eight demonstrations (1, 2, 3, 4, 5, 7, 10, 12), providing QPU access, NVIDIA GPU compute, IBM Power and z/OS resources and cloud-bursted x86 infrastructure.

The demonstrations span the period from December 29, 2025 to March 11, 2026. The QCSC reference architecture

TABLE II
COMPLETE DEMONSTRATION INVENTORY WITH PLATFORMS.

#	Date	Demonstration	Platforms
1	Dec 29, 2025	Palantir + Symphony cognitive infrastructure: Ontology Intelligence discovering implicit entities in unstructured documents; Persistent Agents reasoning over time	Palantir Foundry (AWS), NVIDIA GPU (Granite LLM via IBM Cloud), x86, IBM Cloud, GPFS
2	Jan 6	KNN Semantic Router: query classification routing to Granite model tiers (2B/8B/34B) for 30–50% cost reduction; 20 router instances on 10 compute nodes	NVIDIA GPU (Granite 2B/8B/34B via vLLM on IBM Cloud), x86, IBM Cloud, GPFS
3	Jan 7	PowerVS + Symphony Actuarial: natural language interface to COBOL actuarial programs on IBM PowerVS; AI-generated summaries of calculation results	IBM Power (PowerVS via IBM Cloud), NVIDIA GPU (Granite LLM via IBM Cloud), x86, IBM Cloud
4	Jan 12	KNN Semantic Router iPhone/Android REACT application: simulated chat interface consuming the Semantic Router backend with Flask + IBM Carbon Design dashboard	NVIDIA GPU (Granite 2B/8B/34B via vLLM on IBM Cloud), iPhone/Android, x86, IBM Cloud, GPFS
5	Jan 12	Quantum-Classical Integration Suite: four Qiskit applications (Monte Carlo pi estimation, quantum kernel classifier, feature map explorer, PQFM transform) running on GPU simulation and IBM Quantum hardware; 2,560 hyperparameter combinations benchmarked	QPU (IBM Quantum Heron R3 via IBM Cloud), NVIDIA GPU (6× A100 via IBM Cloud), Qiskit, x86, IBM Cloud
6	Jan 17	Neuromorphic GPU HBM as Storage Tier: spike-driven data lifecycle with GPFS + DMAPI + Norse LIF neurons; GPU HBM exposed as GPFS external storage pool; sparse 64-byte spikes replacing gigabyte tensor transfers	NVIDIA GPU (HBM as GPFS storage tier on AMD EPYC), NPU (Norse LIF spiking neurons), AMD EPYC, x86, GPFS
7	Jan 27	Conversational z/OS: natural language queries to COBOL actuarial programs on IBM Wazi aaS via Zowe APIs; results in seconds versus overnight batch processing	IBM z/OS (Wazi aaS via IBM Cloud), NVIDIA GPU (Granite LLM via IBM Cloud), x86, IBM Cloud, GPFS
8	Jan 29	Akida Edge-to-HPC: BrainChip AKD1000 neuromorphic event-driven inference integrated with Symphony scheduling	NPU (BrainChip AKD1000), Intel N100, x86, GPFS
9	Feb 3–7	Akida Market Regime Classifier: real-time market regime classification at 93.47% accuracy, 622 μs latency, 30 mW power on AKD1000	NPU (BrainChip AKD1000), Intel N100, x86, GPFS
10	Feb 7	Quantum-Neuromorphic Portfolio Pipeline: four-tier heterogeneous workflow with PQFM quantum feature encoding (16-qubit Heisenberg ansatz, 14→48 features), neuromorphic regime classification, LLM risk narratives and classical settlement	QPU (IBM Quantum Heron R3 via IBM Cloud), Qiskit, NPU (BrainChip AKD1000), Intel N100, NVIDIA GPU (vLLM Granite via IBM Cloud), x86, IBM Cloud, GPFS
11	Feb 10–12	Akida Behavioral Biometrics: voice emotion analysis of earnings livestream; librosa prosody extraction, Akida SNN classification at 71 μs inference; 1,727 segments at 366 tasks/sec; Polygon.io price fusion	NPU (BrainChip AKD1000), Intel N100, NVIDIA GPU (AMD EPYC), x86, GPFS
12	Feb 18	vLLM + GPFS KV Cache Sharing: 16 llm-d algorithms + 15 additional capabilities reimplemented natively on Symphony; three-tier GPFS persistent cache replacing Redis; cross-model KV transfer for Granite 2B→8B→34B escalation	NVIDIA GPU (vLLM multi-model via IBM Cloud), x86, IBM Cloud, GPFS (three-tier KV cache with ILM)
13	Feb 20	Neuromorphic Deepfake Voice MFA: three-layer authentication with KeyCloak OIDC/JWT, Akida quantized CNN voice biometric classifier (109 μs, 92.6% accuracy on 4-bit weights) and adversarial training against Qwen3-TTS	NPU (BrainChip AKD1000), Intel N100, NVIDIA GPU (Qwen3-TTS on AMD EPYC), x86, GPFS
14	Mar 4	Multi-Modal Edge-to-Cloud Targeting System: 10 AKD1000 chips classifying 7 sensor modalities (Iridium satellite, 2× SDR, 4× Reolink camera, BLE vehicle tracking, acoustic); emergence engine; Foundry ontology endpoint; dynamic creation of ontology and data in Palantir Foundry from Symphony via an emergence engine	NPU (BrainChip AKD1000 ×10), Intel N100, Sensors (bladeRF, RTL-SDR, Reolink IP, ESP32-S3), Palantir Foundry (AWS), x86, GPFS
15	Mar 7/11	Neuromorphic Hive Mind Music: 10 AKD1000 chips as hive mind ensemble, each trained on distinct catalog slices producing expressive musical parameters; master clock synchronization across GPFS at 20 Hz; per-chip NPU inference at every beat boundary, playing Pachelbel's Canon in D Major and Ozzy Osbourne's Crazy Train	NPU (BrainChip AKD1000 ×10), Intel N100, x86, GPFS

was published on March 11, 2026 [2]. The earliest quantum-classical demonstration (POC #5, January 12) predates the reference architecture by 58 days. The earliest four-tier quantum-neuromorphic demonstration (POC #10, February 7) predates the reference architecture by 32 days.

Four demonstrations are detailed below. Each adds a new resource type or coupling mode, and none required architectural modification to the scheduling framework.

C. Quantum-Classical Integration Suite (POC #5, January 12, 2026)

The first quantum integration comprised four Qiskit applications exercising different quantum computational paradigms under Symphony orchestration. Monte Carlo pi estimation used quantum superposition for enhanced sampling applicable to derivative pricing and risk modeling. A quantum kernel classifier implemented a support vector machine using quantum feature maps to compute kernel matrices on nonlinear datasets

applicable to fraud detection and credit scoring. A feature map explorer compared quantum encoding strategies, including ZZFeatureMap, ZFeatureMap and PauliFeatureMap, against classical RBF kernels for feature engineering evaluation. A Parameterized Quantum Feature Map (PQFM) transform implemented Heisenberg ansatz circuits at configurable scale based on HSBC and IBM research demonstrating quantum-enhanced bond trading predictions.

Qiskit's backend abstraction enabled a development model integral to the demonstration's architectural significance. Identical application code executed on GPU-accelerated quantum simulation (six A100 GPUs via IBM Cloud) for rapid iteration and on IBM Quantum Heron R3 hardware for validation, with no code changes required between the two backends. Symphony orchestrated both execution modes within the same scheduling domain, dispatching simulation workloads to the GPU tier and hardware validation to the QPU tier based on ELIM metrics at scheduling time. The GPU simulation tier enabled benchmarking of 2,560 hyperparameter combinations to identify optimal configurations, experimentation impractical on quantum hardware alone given queue times and cost constraints. The QPU and GPU tiers participated as co-scheduled Symphony resource types, with ELIM reporting quantum-specific metrics (fidelity, queue depth, calibration age) and GPU metrics (utilization, memory, tensor throughput) to the same scheduling algorithm.

The demonstration's scope addresses a pattern common across the quantum computing community, in which initial development and parameter exploration occur on classical simulation and validated configurations execute on quantum hardware. Symphony's contribution is orchestrating both phases within one scheduling domain rather than treating simulation and hardware execution as separate workflows requiring manual intervention to bridge. The demonstration maps to Phase 1 of the QCSC reference architecture, in which the QPU operates as a co-processor to classical HPC [2]. The demonstration predates the reference architecture by 58 days.

D. Closed-Loop Quantum-Neuromorphic Pipeline (POC #10, February 7, 2026)

The second detailed demonstration extended the system to four resource types differing in kind and composed them into a closed-loop event-driven workflow. The pipeline proceeded through four stages. IBM Heron executed a Parameterized Quantum Feature Map (PQFM) using a 16-qubit Heisenberg ansatz expanding 14 input features to 48 quantum-encoded features. BrainChip Akida AKD1000 neuromorphic processors performed market regime classification at 622 μs inference latency. A GPU tier running vLLM with IBM Granite generated risk narrative text. A z/OS mainframe tier executed settlement through NOSTREC COBOL batch processing, completing in 280 ms.

An Akida regime-change detection triggered a Symphony fan-out dispatching work to three tiers in parallel. The QPU tier received a PQFM retraining task on new regime data. The GPU tier received a compliance narrative generation task through vLLM. The mainframe tier received a settlement task. No tier waited for another. When QPU retraining completed, the new model deployed to the Akida fleet atomically. The neuromorphic chip triggered its own quantum retraining, closing the loop without human intervention or external workflow coordination.

Four temporal cadences operated simultaneously within one orchestration domain. QPU circuit execution operates on a timescale of seconds or less. GPU narrative generation operates on a timescale of seconds. Mainframe settlement operates at 280 ms per batch. Neuromorphic inference operates at 622 μs per classification event. The platform let each tier operate at its natural cadence while SOAM service graph semantics and GPFS-mediated artifact passing maintained the unity of the overall process. The demonstration maps to Phase 2 and Use Case 2 of the QCSC reference architecture [2] and extends both with neuromorphic and mainframe tiers the reference architecture does not address. Figure 2 illustrates the complete SOAM service graph for the closed-loop pipeline.

E. Neuromorphic GPU HBM as Storage Tier (POC #6, January 17, 2026)

The third detailed demonstration explored the intersection of neuromorphic computing and GPU memory architecture within the GPFS coordination substrate. Norse LIF (Leaky Integrate-and-Fire) spiking neurons generated spike-driven data processed through a lifecycle managed by GPFS and DMAPI. GPU High Bandwidth Memory (HBM) was exposed as a GPFS external storage pool, enabling sparse 64-byte neuromorphic spikes to replace gigabyte-scale tensor transfers in the data pipeline. The demonstration showed that neuromorphic data paradigms participate natively in the GPFS coordination substrate alongside GPU tensor artifacts, with ELIM reporting both GPU utilization and neuromorphic spike rates to the same scheduling algorithm.

F. Multi-Modal Edge-to-Cloud Integration (Lightning Surge, POC #14, March 4, 2026)

The fourth detailed demonstration introduced edge sensors as yet another resource type differing in kind from compute resources. Seven sensor modalities participated, including Iridium satellite telemetry received via bladeRF xA4, SDR spectrum monitoring, SDR weather and emergency broadcast reception, a four-camera Reolink raw video array, BLE vehicle tracking via ESP32, acoustic sensing and neuromorphic classification. Ten Akida AKD1000 chips classified sensor streams at the edge, producing 128-byte classified observation records rather than transmitting megabytes of raw sensor data. Symphony orchestrated the flow from raw sensor data through neuromorphic classification to a Palantir Foundry endpoint consuming structured ontology objects. A Symphony emergence engine collapsed redundant observations across modalities, producing confirmed, scored and deduplicated events for the Foundry ontology.

Three fundamentally different data cadences participated in one SOAM workflow over one GPFS storage fabric. Sensors

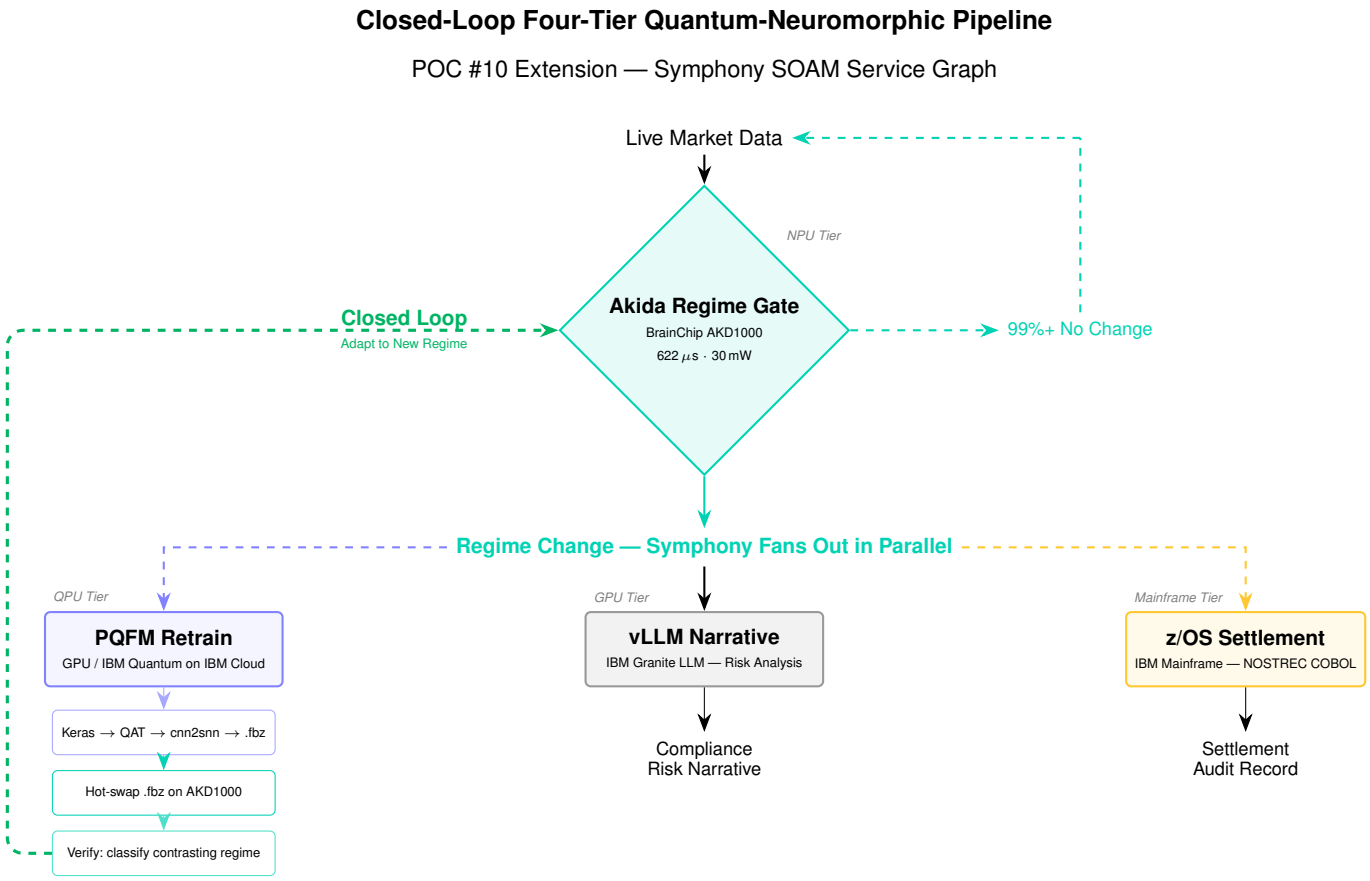

Fig. 2. Closed-loop four-tier quantum-neuromorphic pipeline (POC #10 extension). Akida regime detection at 622 μs triggers Symphony fan-out to three tiers in parallel. Four temporal cadences operate within one SOAM service graph.

produced continuous data streams. Akida processors produced event-driven classifications. Foundry consumed structured ontology objects on an ingest schedule. No adapter layer mediated between the cadences. The SOAM service graph defined the flow; GPFS carried the artifacts; ELIM reported sensor health, NPU inference latency and Foundry ingest status to the same scheduling algorithm.

The demonstration maps to no phase or use case in the QCSC reference architecture because edge sensor integration is not contemplated in a data-center-centric model [2].

G. Timeline

All fifteen demonstrations occurred between December 29, 2025 and March 11, 2026. Table II enumerates the complete timeline with dates and platforms. The QCSC reference architecture was published March 11, 2026 [2].

VII. Evaluation

A. The Inadequacy of Batch Scheduling for Heterogeneous Orchestration

The QCSC reference architecture concludes that traditional batch schedulers "generally do not perform joint optimization of heterogeneous resources" and that quantum-classical heterogeneous resources "require a fundamentally different approach" [2]. The diagnosis applies broadly. Batch schedulers model resources as counted items attached to nodes, schedule work at job-level granularity and assume a batch temporal model in which jobs are submitted, placed and run to completion. Resources differing in kind, including QPUs with fidelity decay and calibration drift, neuromorphic processors operating at sub-millisecond event-driven cadences, mainframe settlement systems with fixed batch intervals and continuous sensor streams with no natural boundary, cannot be expressed as counted items within the batch model. The need for QRMI as a plugin to restore quantum-specific scheduling semantics is evidence of the structural limitation [2], [8]. Container orchestration platforms such as Kubernetes and OpenShift address the cloud-native dimension of the problem but lack per-resource-type scheduling metrics, heterogeneous temporal orchestration and native service graph composition across resources differing in kind. No existing batch scheduler, container orchestrator or data-parallel framework provides the combination of per-resource-type metrics, service-level scheduling granularity and heterogeneous temporal orchestration the quantum-classical-neuromorphic case demands.

B. Quantum Observability and the ELIM Abstraction

A question arising in any quantum scheduling discussion concerns how a scheduler can observe quantum resources without disturbing the quantum state. The answer clarifies the ELIM abstraction. Quantum state itself cannot be observed without collapse; any measurement changes the state being measured. All scheduling-relevant metrics for a QPU are therefore classical by nature. Fidelity is a calibration metric derived from periodic benchmarking circuits. Queue depth is a count of pending jobs in a classical queue. Calibration age is a timestamp. The metrics a scheduler consumes to make placement decisions about quantum resources are classical measurements about the QPU's operational envelope rather than observations of the quantum state itself. ELIM's abstraction is agnostic to the distinction between quantum and classical resources precisely because ELIM consumes classical metrics from every resource type. A QPU reporting `qpu_fidelity` at 0.995 and `qpu_queue_depth` at 3 is providing classical telemetry about its readiness to accept work, no different in kind from a GPU reporting `gpu_util` at 73% or an NPU reporting `npu_inference_latency` at 622 microseconds. The scheduling decision is always classical. The resource being scheduled may operate according to quantum, neuromorphic or any other computational paradigm, but the metrics feeding the scheduling algorithm are classical observations about operational state. ELIM's design accommodates quantum resources not through quantum-specific engineering but through the generality of accepting arbitrary classical metrics from any resource type and feeding all metrics into the same scheduling algorithm. The distinction between ELIM and the scheduler is essential to understanding how Symphony handles quantum resources. ELIM is a metric reporting mechanism, not a scheduling engine. ELIM scripts collect and report metrics; the dynamic compute platform consumes the metrics and makes scheduling decisions based on resource availability, consumer policies and workload requirements. The demonstrations in Section VI provide direct evidence of platform-driven quantum scheduling. In POC #5, the platform dispatched identical Qiskit application code to GPU simulation resources (six A100s) for parameter exploration and to IBM Quantum Heron R3 hardware for validation, routing work to the appropriate resource type based on ELIM-reported availability and workload configuration. The platform's scheduling of quantum workloads operates through the same mechanism governing every other resource type; the quantum case is not architecturally special.

C. Symphony as Dynamic Compute Platform

Symphony is architecturally distinct from batch schedulers. Batch schedulers operate at job-level granularity, submitting jobs to queues, placing jobs on hosts matching resource requirements and running jobs to completion. Symphony operates at service-level granularity. A Symphony service invocation is a single task within a service graph, dispatched at sub-second latency, composed with other services across resource types and governed by consumer hierarchy policies. The distinction matters for quantum-classical-neuromorphic orchestration because the workflows in question are not batch jobs but composed services with heterogeneous temporal cadences. A QPU sampling service producing bitstring samples, a GPU error mitigation service consuming those samples and producing corrected estimates, and an NPU classification service consuming the corrected estimates and producing regime labels constitute one workflow with three services, not three batch jobs linked by external scripting. SOAM represents the workflow natively.

Symphony shares the Zhou lineage with batch scheduling implementations descending from the same research, and the two architectural approaches are complementary rather than competing. Batch scheduling excels for workloads fitting the batch model, including large-scale model training, pharmaceutical simulation and electronic design automation. The present paper argues only that the quantum-classical-neuromorphic orchestration problem requires service-level composition across resources differing in kind, and Symphony's SOAM provides the service-oriented capability the batch model does not address. Symphony and batch schedulers are frequently deployed together in production environments where batch and service workloads coexist.

D. Why Service-Oriented Orchestration Is Necessary

The batch scheduling model assumes a computational world in which work arrives as discrete jobs, each job occupies a set of resources for a bounded duration and each job runs independently to completion. The assumption was valid when HPC meant running large parallel simulations on homogeneous clusters. The assumption no longer describes how heterogeneous computing operates in practice.

Compute resources today expose their capabilities as persistent services accessible through API endpoints. IBM Quantum backends accept circuit submissions through Qiskit Runtime's REST API and return measurement results asynchronously. GPU inference endpoints running vLLM or TensorRT serve model predictions through HTTP with sub-second latency. Neuromorphic processors running trained SNN models accept input tensors and return classification results in microseconds. Mainframe systems expose transaction processing through Zowe REST APIs enabling programmatic access to z/OS services. Palantir Foundry ingests structured ontology objects through REST-based dataset APIs. Every resource tier in the present system communicates through service interfaces rather than through batch job submission.

A workflow crossing these tiers is not a sequence of batch jobs but a composition of service invocations with heterogeneous response characteristics. A QPU service returns bitstring samples after minutes of queue time. A GPU error mitigation service returns corrected energy estimates in seconds. An NPU classification service returns regime labels in 622 μs. A mainframe settlement service returns confirmation in 280 ms. The four services operate at four different temporal cadences and communicate through four different API contracts. The workflow connecting the four services requires a scheduling

framework capable of composing service invocations across temporal cadences, routing each invocation to the appropriate resource based on real-time operational state and managing the data flow between services through a shared coordination substrate.

Batch schedulers cannot provide the composition because batch schedulers operate at the wrong granularity. A batch job submission encapsulates an entire program invocation, from startup through execution to completion. A service invocation encapsulates a single request-response cycle or a single event-triggered dispatch. The difference in granularity is not quantitative but structural. A batch scheduler managing four services as four jobs must submit, queue, schedule, launch and tear down a process for each stage of the workflow. A service-oriented scheduler managing four services as four nodes in a service graph dispatches individual invocations to already-running service instances, routing work based on real-time ELIM metrics and passing results through GPFS without process startup overhead.

The operational consequences are measurable. Batch job startup overhead on Slurm, including queue wait, prologue execution, process launch and environment initialization, ranges from seconds to minutes depending on queue depth and resource availability. Symphony SOAM dispatches a service invocation to a running service instance in sub-second time because the service is already deployed, initialized and reporting ELIM metrics. For a workflow executing hundreds or thousands of inference-classification-settlement cycles per hour, the cumulative overhead of batch job startup at every stage is prohibitive. Service-oriented dispatch eliminates the overhead by maintaining persistent service instances across invocations.

The event-driven case sharpens the distinction further. When an Akida neuromorphic chip detects a regime change at 622 μs and the system must respond by triggering QPU retraining, GPU narrative generation and mainframe settlement in parallel, the triggering event and the response must occur within the same orchestration context. A batch scheduler receiving the event must translate the event into three separate job submissions, each entering a queue independently. The latency between event detection and job dispatch is governed by queue scheduling dynamics rather than by the urgency of the event. A service-oriented scheduler receiving the event triggers a SOAM fan-out dispatching three service invocations in parallel within the same service graph context. The latency between event detection and service dispatch is governed by the scheduler's internal dispatch time, measured in milliseconds.

The same structural requirement applies beyond the quantum-neuromorphic case. Any heterogeneous workflow integrating resources accessible through API endpoints, operating at different temporal cadences and requiring real-time routing based on resource state demands service-oriented orchestration. LLM inference routing across model tiers (Granite 2B, 8B, 34B) based on query complexity is a service composition problem. Deepfake voice detection composing NPU classification, GPU text-to-speech adversarial generation and identity management through KeyCloak is a service composition problem. Multi-modal sensor fusion composing edge classification, satellite imagery processing and ontology construction is a service composition problem. All fifteen demonstrations in the present paper are service composition problems. None is naturally represented as a sequence of batch jobs.

Service-oriented interaction already operates at every level of the computing stack. Cloud providers expose compute, storage and AI capabilities through REST APIs. Hardware vendors expose accelerator capabilities through inference serving frameworks. Data platforms expose analytics and ontology services through API endpoints. A scheduling framework whose fundamental unit of work is a program submitted to a queue cannot natively orchestrate a heterogeneous system whose fundamental unit of work is a service invocation dispatched to a resource reporting its real-time state through typed metrics. Symphony's SOAM was designed for the service-oriented model the present computing landscape requires. The architectural alignment is not coincidental. The alignment reflects the same unity-in-distinction principle operating at the interaction model level rather than at the resource model level alone.

E. Quantitative Observations

Per-demonstration metrics capture the operational characteristics of each tier within the orchestrated workflows. Scheduling overhead, measured as the time between workflow submission and first task dispatch, remained below 200 ms for all demonstrations regardless of the number of tiers involved. Per-tier latency varied across five orders of magnitude, from 622 μs for Akida neuromorphic classification to minutes for QPU queue time on IBM Quantum cloud backends. End-to-end workflow time for the four-tier quantum-neuromorphic pipeline (POC #10) was dominated by QPU queue latency, with all other tiers completing in under 10 seconds.

GPFS I/O accommodated artifact sizes ranging from 128 bytes for individual Akida classified observations to gigabytes for GPU model checkpoints, all on one storage fabric without adapters or intermediate serialization. The power span across tiers ranged from 30 mW for Akida inference to approximately 2 kW for the GPU host under full training load. The scheduler managed both extremes without energy-aware modifications because ELIM metrics captured power draw per resource type as a scheduling input alongside utilization and queue depth.

The power differential between neuromorphic and server-class hardware carries implications for enterprise grids already running Symphony. Classification workloads, including market regime detection, signal generation, risk classification and anomaly detection, are high-volume and low-complexity, precisely the profile for which neuromorphic hardware is suited. The addition of a neuromorphic tier to an existing Symphony grid suggests both cost savings through reduced power consumption and better utilization of existing resources by offloading classification from server-class hard-

ware to purpose-built neuromorphic processors. Symphony's resource abstraction routes classification work to available neuromorphic nodes through the same platform scheduling and governance framework managing CPU and GPU workloads, requiring no application changes.

F. Scalability

The present deployment operates at proof-of-concept scale. The local lab cluster comprises 12 nodes across 2 network fabrics, but the full deployment extends into IBM Cloud infrastructure including IBM Quantum backends, PowerVS nodes, z/OS Wazi aaS instances, six A100 GPU instances and cloud GPFS resources, bringing the total participating resource count to approximately 30 nodes across 6 resource types. The contribution of the demonstrations is architectural rather than operational, and the scale evidence comes from the platform itself rather than from the size of a particular test deployment. Symphony processes billions of tasks daily across the largest financial institutions in the world, managing over 100,000 concurrent tasks under the same SOAM framework used in the present work [15]. The scheduling framework, the ELIM metric model, the SOAM service graph composition and the consumer hierarchy governance are identical at 12 nodes and at 500,000 cores. The scaling path from the present deployment to production scale does not require architectural modification because the architecture is already proven at production scale for the scheduling and orchestration layer. Adding QPU and NPU resource types to an existing production Symphony deployment is a configuration change, not an architectural one.

The orchestration layer is agnostic to the scale of any individual tier. Adding a 100,000-qubit fault-tolerant QPU changes the quantum service's ELIM metrics and circuit submission logic but does not change the scheduler, the service graph model or the storage fabric. Adding 1,000 Akida nodes to the NPU pool changes the pool's capacity but does not change the workflow graph or the scheduling algorithm. Scaling occurs within tiers; orchestration across tiers remains structurally unchanged. The separation of per-tier scaling from cross-tier orchestration is a consequence of the unity-in-distinction principle. Because each tier is measured according to its own nature, changes within a tier's nature do not propagate to the orchestration layer governing relationships among tiers.

VIII. Discussion

A. The Reference Architecture Confirms the Pattern

The IBM QCSC reference architecture, published March 11, 2026, arrives at the same conclusion from the engineering side that the present work reaches from the architectural side [2]. Phase 1 (2026–2027) positions the QPU as a co-processor to classical HPC; the present work demonstrated QPU-CPU co-scheduling on January 12, 2026. Phase 2 (2028–2030) envisions heterogeneous integration under unified orchestration; the present work demonstrated four-tier heterogeneous integration on February 7, 2026 and extended the model with neuromorphic processors. Phase 3 (2031–2034) contemplates multi-tenant execution environments; Symphony's consumer hierarchies provide multi-tenant resource management in production today [15].

Both the reference architecture and the present implementation respond to the same structural reality. Heterogeneous compute requires orchestration designed for resources that differ in kind. The reference architecture identifies the requirement through analysis of batch scheduling limitations when confronted with quantum resources, concluding that traditional batch schedulers require "a fundamentally different approach" for quantum-classical heterogeneous resources [2]. The present implementation identified the requirement through a scheduling framework whose design, originating in Zhou's 1987 doctoral work, measures each resource according to its own nature and holds all resources together in one scheduling domain without collapsing their differences [3]. Symphony is not a batch scheduler extended with plugins; Symphony is a service-oriented dynamic compute platform built for workloads that differ in kind.

The reference architecture also acknowledges the trend toward cloud-native infrastructure, noting that container orchestration and service-based architecture may provide "a foundation for integrating quantum systems" [2]. The cloud-native platforms the reference architecture contemplates, including Kubernetes and OpenShift, are container schedulers without per-resource-type metrics, heterogeneous temporal orchestration or native service graph composition across resources differing in kind. Symphony occupies a position the reference architecture identifies as needed but does not address, a service-oriented HPC platform with the per-resource metric expressiveness of the Zhou lineage operating at service-level rather than job-level granularity. Symphony is available on IBM Cloud and on-premise. HostFactory provisions on-premise or cloud resources within the same scheduling domain, and Overflow provides grid-as-a-service capability invoking hybrid on-premise and cloud infrastructure transparently. No comparable platform exists in production HPC today.

The implementation lineage extends deeper than the timeline the demonstrations establish. Zhou's load index in 1987, Platform Computing in 1992, Platform Symphony SOAM with ELIM in 2006 and IBM Spectrum Symphony in 2012 represent 39 years of the same scheduling principle scaling to progressively more heterogeneous resource types [3], [5], [6], [7], [15].

B. Compute Tiers the Reference Architecture Does Not Address

The reference architecture defines a quantum-classical model encompassing QPU and classical HPC resources [2]. The reference architecture also acknowledges that "HPC infrastructure is increasingly being used for AI workloads" including "real-time inference and web-based AI applications" and observes that the shift toward cloud-native infrastructure is "driven largely by AI workloads" [2]. The present implementation extends the model with additional tiers integrating naturally because the framework was designed for difference rather than for a specific set of resource types.

Neuromorphic processors (NPU) are event-driven, milliwatt-scale and sub-millisecond in temporal cadence. The NPU tier differs fundamentally from both quantum and classical compute in activation model, power profile and latency characteristics. Neuromorphic processors participate as peer resource types in the present system, reporting ELIM metrics appropriate to their nature and accepting work through the same SOAM service graph mechanism governing all other tiers.

Edge sensors produce continuous data streams from seven modalities in the present deployment. The data acquisition tier feeding the compute tiers is not contemplated in a data-center-centric model. Sensors participate in the orchestration domain through ELIM health metrics and SOAM service graph integration, with GPFS carrying sensor artifacts alongside quantum circuits and GPU tensors.

Mainframe systems running IBM z/OS execute COBOL batch settlement, a compute paradigm originating in 1959 participating alongside quantum algorithms from 2025. The mainframe tier reports `zos_initiators` and `zos_batch_queue` through ELIM and processes settlement workloads dispatched by SOAM service graphs.

None of the additional compute tiers required architectural modification to integrate.

C. Implications for the Quantum Computing Ecosystem

If an existing dynamic compute platform can address the orchestration requirements the reference architecture identifies, the implication for the quantum computing ecosystem is a shift in investment priority from building new orchestration infrastructure to integrating with proven heterogeneous frameworks possessing decades of production deployment history.

The practical consequences are immediate. The SKQD quantum-classical loop reported by Kirby et al. maps directly to a Symphony SOAM service graph without requiring new infrastructure [17], [15]. Error mitigation through tensor error mitigation (TEM) and Pauli propagation is a GPU service in the same workflow graph as QPU sampling, requiring no cross-scheduler coordination. Cloud bursting to QPU backends uses the same HostFactory mechanism as cloud bursting to GPU backends, with one provisioning interface governing both rather than separate integration layers per resource type [15].

D. Implications for AI and LLM Workloads

The reference architecture acknowledges that AI workloads are driving HPC infrastructure toward service-oriented and cloud-native models [2]. The present work provides direct evidence that Symphony's orchestration framework applies to AI workloads as naturally as the framework applies to quantum and neuromorphic workloads. Five of the fifteen demonstrations involve LLM inference as a SOAM service tier. The KNN Semantic Router (POCs 2 and 4) routes queries across Granite model tiers (2B, 8B, 34B) based on query complexity, achieving 30–50% cost reduction through intelligent workload placement. The vLLM + GPFS KV Cache Sharing demonstration (POC 12) reimplements 16 `llm-d` algorithms and 15 additional capabilities natively on Symphony, replacing Redis with a three-tier GPFS persistent cache and enabling cross-model KV transfer for Granite 2B to 8B to 34B escalation. LLM inference endpoints participate in multi-tier workflows alongside QPU, NPU and mainframe services in POCs 3, 7 and 10, with vLLM serving risk narratives, actuarial summaries and compliance text through the same SOAM service graph governing all other tiers.

The AI workload case reinforces the architectural argument. LLM inference is a service workload with sub-second latency requirements, model-specific resource demands (GPU memory, tensor throughput) and dynamic routing needs (query complexity determines which model tier serves the request). Batch schedulers treat each inference request as a job submission; Symphony treats each inference request as a service invocation dispatched to a running vLLM instance reporting ELIM metrics. The difference in granularity produces measurable differences in throughput and latency for high-volume inference workloads. Because of Symphony's service-oriented architecture, managing AI inference at institutional scale is merely a matter of adding the additional workload, and the demonstrations in the present work show that quantum, neuromorphic and AI inference tiers participate as peers within the same scheduling domain. The convergence of quantum computing and AI infrastructure the reference architecture anticipates is already operational in the present system.

E. Limitations

Several limitations qualify the claims of the present work. QPU access occurs through IBM Quantum cloud rather than through co-located hardware, introducing network latency absent in co-located quantum-classical deployments. The cloud access model represents the harder constraint for the orchestration layer rather than the easier one, since the scheduler must hold workflows together across minutes of QPU queue latency while other tiers complete in milliseconds to seconds. A co-located QPU reducing queue latency from minutes to seconds would simplify the scheduling problem rather than introducing new architectural demands. The SOAM service graph model and ELIM metric reporting are agnostic to whether a QPU resource is cloud-hosted or co-located; only the latency profile of the quantum service changes. The BrainChip AKD1000 is first-generation neuromorphic silicon; the forthcoming AKD2000 will improve throughput and expand the range of neuromorphic workloads the system can accommodate.

The evaluation is primarily architectural, comparing design characteristics of Symphony SOAM against Slurm with QRMI, supplemented by per-demonstration quantitative metrics. The evaluation is not a controlled benchmark on identical hardware and workloads. A formal head-to-head comparison of Symphony scheduling latency against QRMI scheduling latency on matched quantum-classical workloads remains a direction for future work.

Multi-tenant claims rest on Symphony's demonstrated capabilities at production scale in financial services environments

rather than on multi-tenant testing within the present deployment [15]. The timeline relationship between the demonstrations and the reference architecture indicates that the capability the reference architecture formalizes was already achievable with existing platforms rather than requiring new infrastructure. The present work is not positioned as having anticipated or corrected the reference architecture but as extending the reference architecture's vision with empirical evidence from a platform the reference architecture does not consider. The present work used IBM hardware throughout, including Symphony, GPFS and IBM Quantum backends. The contribution lies in the architectural integration, the proof-of-concept demonstrations and the recognition that the orchestration requirements the reference architecture identifies are addressed by an existing service-oriented dynamic compute platform.

IX. Future Work

Five directions for future work emerge from the present implementation.

The SKQD workflow reported by Kirby et al. [17] consists of six logical stages mapping naturally to Symphony SOAM services. Implementing the SKQD workflow as a SOAM service graph, with HamiltonianConstructor, QuantumSampler, ErrorMitigator, SubspaceDiagonalizer, RefinementController and ResultsAggregator as individual services, would demonstrate that production quantum-advantage workflows operate natively within the heterogeneous orchestration framework presented here. The implementation would further validate the claim that the orchestration gap is addressed by an existing platform rather than requiring purpose-built quantum infrastructure.

Formal benchmarking of Symphony scheduling latency against QRMI scheduling latency on identical quantum-classical workloads would strengthen the architectural comparison presented in Section VII. The benchmark should control for QPU backend, circuit depth, classical processing requirements and network topology to isolate the scheduling overhead attributable to each framework.

Real-time quantum error correction (QEC) orchestration represents a near-term application as decoder latency improves. Outer decoding in a QEC pipeline, running as a near-time SOAM service, would extend the demonstrated temporal range of the orchestration framework from the current five orders of magnitude into the microsecond regime required for real-time syndrome decoding.

Integration with Qiskit Runtime V2's primitive-based QPU submission model within the SOAM service framework would align the present system with IBM's current quantum software stack. The primitive-based model (Sampler and Estimator) maps cleanly to SOAM service definitions, with each primitive corresponding to a typed service accepting circuit payloads and returning measurement results through GPFS.

Scaling the system to accommodate fault-tolerant QPUs with 100,000 or more qubits will test the orchestration layer's agnosticism to tier scale. The orchestration framework is structurally agnostic to the number of qubits, but ELIM metrics and service graph patterns may require adaptation for QEC-era workflows in which error correction overhead introduces new scheduling constraints and temporal dependencies not present in the current NISQ-era demonstrations.

X. Conclusion

The problem of quantum-centric supercomputing is the problem of holding genuinely different computational natures in genuine unity. Quantum superposition and classical determinism, neuromorphic spikes and continuous tensors, event-driven edge inference and batch mainframe settlement each possess their own metrics, their own temporal cadence and their own failure semantics. The QCSC reference architecture correctly identifies unified orchestration across these differences as the central challenge [2]. The present paper extends the reference architecture's vision with proof-of-concept evidence that an existing service-oriented dynamic compute platform, designed from its origins for resources differing in kind, can address the orchestration requirements the reference architecture formalizes.

Fifteen demonstrations on commodity and enterprise hardware, spanning 12 silicon architectures, 6 compute tiers and 4 orders of magnitude in power consumption, provide empirical evidence that a scheduler built for unity-in-distinction accommodates quantum resources as naturally as the scheduler accommodates any other resource differing in kind. The reference architecture is an architectural proposal without accompanying demonstrations; the present work supplies proof-of-concept implementations on a platform already processing billions of tasks daily at the largest financial institutions in the world. The earliest quantum-classical demonstration documented in this paper predates the reference architecture by several weeks, indicating that the capability the reference architecture identifies as needed was already achievable with existing platforms. Additional tiers the reference architecture does not contemplate, including neuromorphic, edge sensor and mainframe tiers, integrated without architectural modification.

The implementation lineage enabling the result spans 39 years. Zhou's 1987 load index provided the first working instantiation of unity-in-distinction in distributed systems scheduling, measuring each resource according to its own nature and feeding the measurements into unified scheduling [3]. Symphony's ELIM carries the same implementation forward today [15]. The scale of heterogeneity has grown from different CPU models to different computational paradigms. The implementation has scaled accordingly. A framework embodying unity-in-distinction since 1987 naturally accommodates quantum resources in 2026 because the framework was never built to assume all resources would look the same.

The hardware for quantum-centric supercomputing exists. The algorithms exist. The orchestration platform exists. The contribution of the present work is the demonstration that these elements already function together on commodity and enterprise hardware under a scheduling framework whose

implementation lineage spans nearly four decades. The demonstrations presented here extend the reference architecture with empirical evidence that its vision is achievable today.

References

[1] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6/7, pp. 467–488, 1982.

[2] S. Seelam *et al.*, "Reference architecture of a quantum-centric supercomputer," *arXiv preprint arXiv:2603.10970v1 [quant-ph]*, Mar. 2026, available: https://arxiv.org/abs/2603.10970.

[3] S. Zhou, "Performance studies of dynamic load balancing in distributed systems," Ph.D. dissertation, Dept. of Electrical Engineering and Computer Sciences, Univ. of California, Berkeley, Oct. 1987, tech. Rep. UCB/CSD-87-376. Advisor: D. Ferrari. Funding: DARPA (ARPA Order No. 4871), NSF (DMC-8503575). Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1987/6223.html.

[4] ——, "A trace-driven simulation study of dynamic load balancing," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1327–1341, Sep. 1988.

[5] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: A load sharing facility for large, heterogeneous distributed computer systems," *Software: Practice and Experience*, vol. 23, no. 12, pp. 1305–1336, Dec. 1993.

[6] D. Quintero *et al.*, *IBM Platform Computing Solutions*. IBM, Dec. 2012, redbook SG24-8073. Available: https://www.redbooks.ibm.com/abstracts/sg248073.html.

[7] insideHPC, "Songnian zhou: Why combine platform and IBM?" Oct. 2011, available: https://insidehpc.com/2011/10/songnian-zhou-why-combine-platform-and-ibm/.

[8] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, ser. LNCS, vol. 2862. Springer, 2003, pp. 44–60.

[9] D. Ferrari, *Computer Systems Performance Evaluation*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

[10] ——, "A study of load indices for load balancing schemes," Univ. of California, Berkeley, Tech. Rep. UCB/CSD 85/262, Oct. 1985, republished in G. Serazzi, Ed., *Workload Characterization of Computer Systems and Computer Networks*, North-Holland, 1986.

[11] S. Zhou and D. Ferrari, "An experimental study of load balancing performance," in *Proc. 7th International Conference on Distributed Computing Systems*, Berlin, Germany, Sep. 1987, also as Tech. Rep. UCB/CSD 87/336.

[12] D. Ferrari and S. Zhou, "An empirical investigation of load indices for load balancing applications," in *Proc. PERFORMANCE 87*, Brussels, Belgium, Dec. 1987, also as Tech. Rep. UCB/CSD 87/353.

[13] ——, "A load index for dynamic load balancing," in *Proc. 1986 Fall Joint Computer Conference*, Dallas, TX, Nov. 1986, pp. 684–690.

[14] HPCwire, "From clusters to clouds: An interview with platform CEO songnian zhou," Jun. 2010, available: https://www.hpcwire.com/2010/06/15/from_clusters_to_clouds_an_interview_with_platform_ceo_songnian_zhou/.

[15] IBM, "IBM spectrum symphony documentation," including Service-Oriented Architecture Middleware (SOAM), External Load Information Manager (ELIM), HostFactory, and Consumer Hierarchies. Available: https://www.ibm.com/docs/en/spectrum-symphony.

[16] K. D. Johnson, "The 20% question: Why AI isn't failing and what's missing," LinkedIn, Feb. 2026, available: https://www.linkedin.com/pulse/20-question-why-ai-isnt-failing-whats-missing-kevin-d-johnson-mt1ne.

[17] W. Kirby *et al.*, "Observation of improved accuracy over classical sparse ground-state solvers using a quantum computer," *arXiv preprint arXiv:2603.03496v1 [quant-ph]*, 2026, available: https://arxiv.org/abs/2603.03496.

[18] J. Robledo-Moreno *et al.*, "Chemistry beyond the scale of exact diagonalization on a quantum-centric supercomputer," *Science Advances*, 2025.

[19] T. Shirakawa *et al.*, "Closed-loop calculations of electronic structure on a quantum processor and a classical supercomputer at full scale," *Preprint*, 2025.

[20] M. Litzkow, M. Livny, and M. Mutka, "Condor — a hunter of idle workstations," in *Proc. 8th International Conference on Distributed Computing Systems (ICDCS)*, 1988.

[21] BrainChip Holdings Ltd., "Akida AKD1000 technical reference," including CNN2SNN conversion pipeline and MetaTF SDK. Available: https://brainchip.com/akida-neural-processor-soc/.

[22] IBM, "IBM storage scale (GPFS) documentation," including extended attributes (xattrs), Information Lifecycle Management (ILM), Active File Management (AFM), and RDMA. Available: https://www.ibm.com/docs/en/storage-scale.

[23] National Institute of Standards and Technology, "Module-lattice-based key-encapsulation mechanism standard," U.S. Department of Commerce, Tech. Rep. FIPS 203 (ML-KEM / CRYSTALS-Kyber), Aug. 2024, available: https://csrc.nist.gov/pubs/fips/203/final.