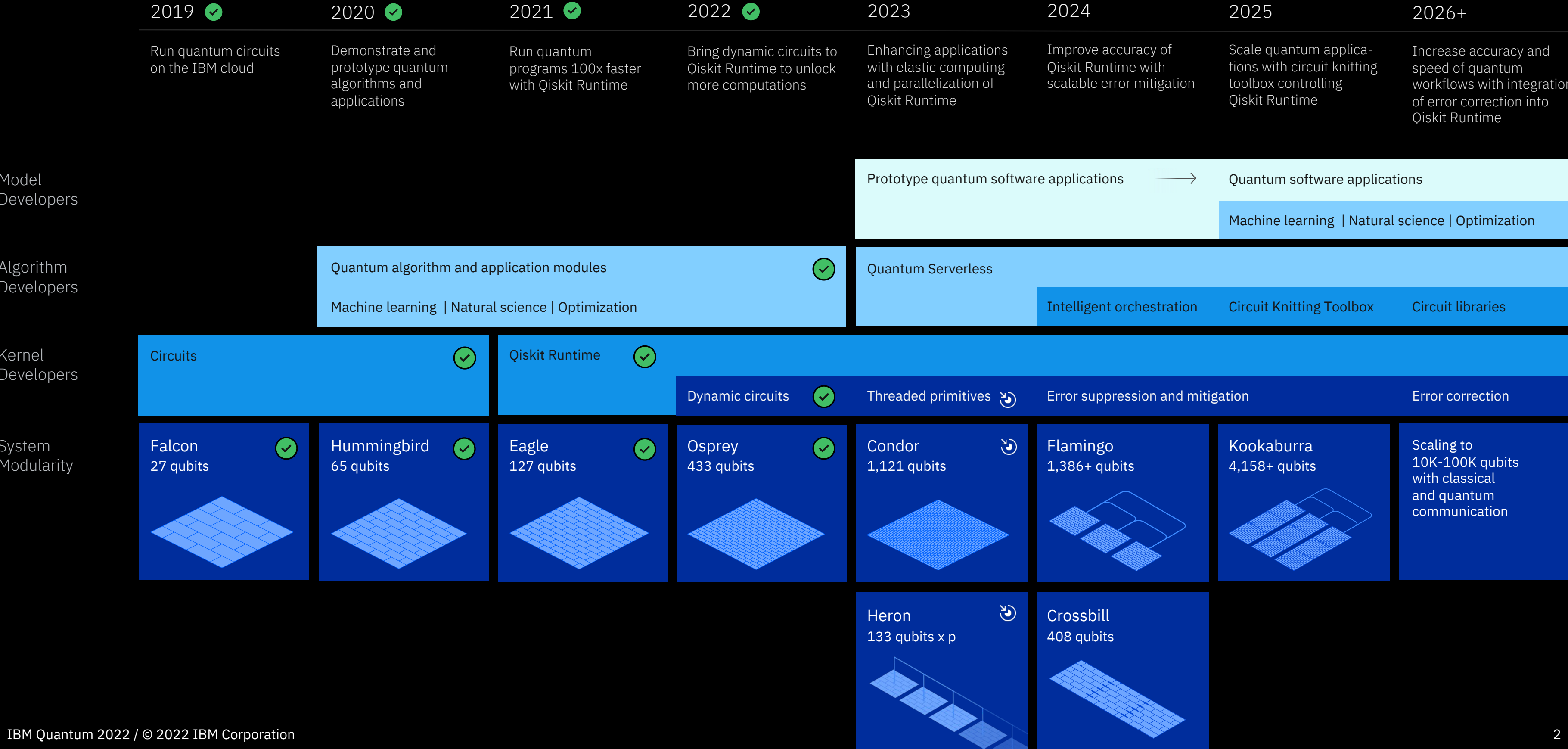






Quantum Centric Supercomputing

Ismael Faro

Distinguished Engineer, Tech Lead IBM Quantum Services



2019 	2020 	2021 	2022 	2023	2024	2025	2026+
Run quantum circuits on the IBM cloud	Demonstrate and prototype quantum algorithms and applications	Run quantum programs 100x faster with Qiskit Runtime	Bring dynamic circuits to Qiskit Runtime to unlock more computations	Enhancing applications with elastic computing and parallelization of Qiskit Runtime	Improve accuracy of Qiskit Runtime with scalable error mitigation	Scale quantum applications with circuit knitting toolbox controlling Qiskit Runtime	Increase accuracy and speed of quantum workflows with integration of error correction into Qiskit Runtime

Quantum-centric
supercomputing

Model
Developers

Algorithm
Developers

Kernel
Developer

System
Modularity

01 Modularity for quantum

02 Communication for quantum

03 Middleware for quantum

Prototype quantum software applications → Quantum software applications

Machine learning | Natural science | Optimization

Quantum Serverless

Intelligent orchestrationCircuit Knitting ToolboxCircuit libraries

Threaded primitivesError suppression and mitigationError correction

Condor
1,121 qubits





Flamingo
1,386+ qubits

Kookaburra
4,158+ qubits

Scaling to
10K-100K qubits
with classical
and quantum
communication

Heron
133 qubits x p

Crossbill
408 qubits

2019 	2020 	2021 	2022 	2023	2024	2025	2026+
Run quantum circuits on the IBM cloud	Demonstrate and prototype quantum algorithms and applications	Run quantum programs 100x faster with Qiskit Runtime	Bring dynamic circuits to Qiskit Runtime to unlock more computations	Enhancing applications with elastic computing and parallelization of Qiskit Runtime	Improve accuracy of Qiskit Runtime with scalable error mitigation	Scale quantum applications with circuit knitting toolbox controlling Qiskit Runtime	Increase accuracy and speed of quantum workflows with integration of error correction into Qiskit Runtime

Quantum-centric
supercomputing

Model
Developers

Algorithm
Developers

Prototype quantum software applications → Quantum software applications

Machine learning | Natural science | Optimization

Quantum Serverless

Intelligent orchestrationCircuit Knitting ToolboxCircuit libraries








Kernel
Developers

System
Modularity

01 Modularity for quantum

02 Communication for quantum

03 Middleware for quantum

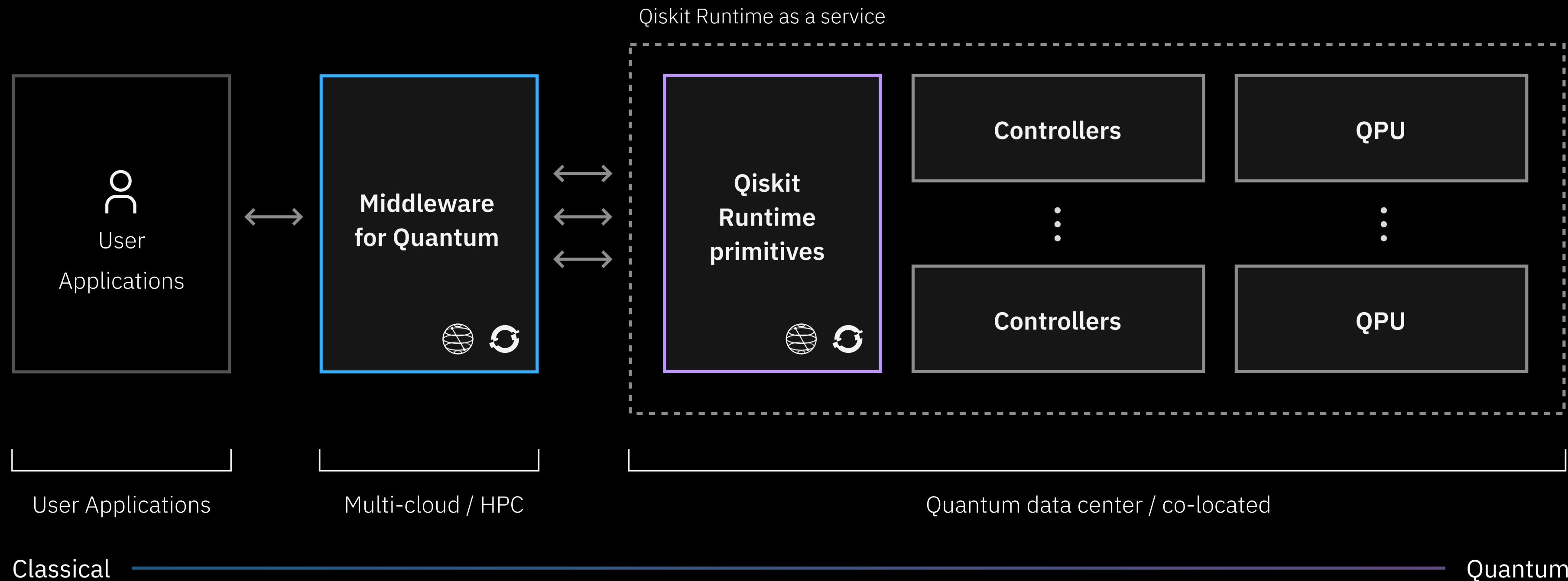
Circuits 	Qiskit Runtime 					
	Dynamic circuits  Threaded primitives Error suppression and mitigation Error correction					
Falcon 27 qubits 	Hummingbird 65 qubits 	Eagle 127 qubits 	Osprey 433 qubits 	Condor 1,121 qubits	Flamingo 1,386+ qubits	Kookaburra 4,158+ qubits
						Scaling to 10K-100K qubits with classical and quantum communication
				Heron 133 qubits x p	Crossbill 408 qubits	

Middleware for quantum

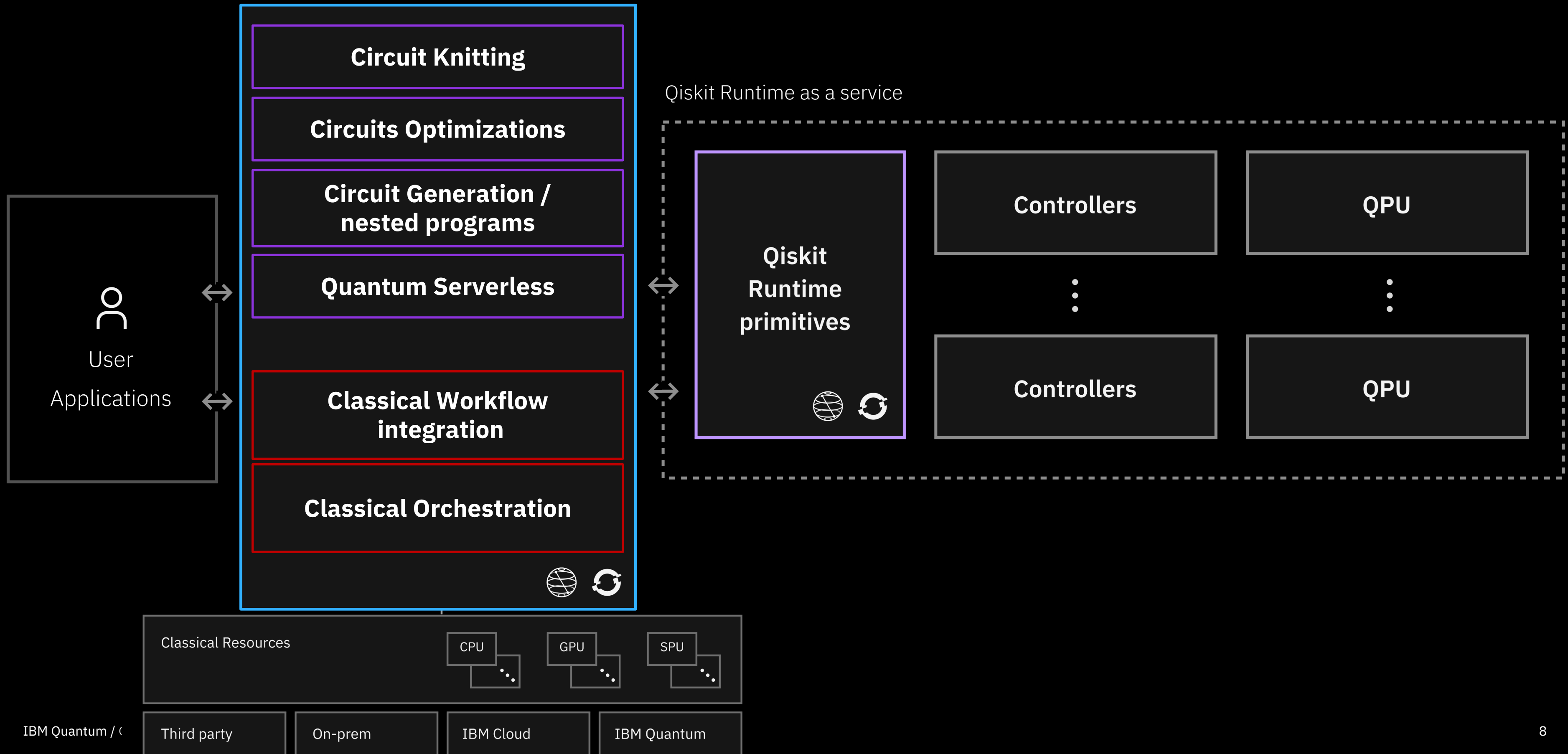
Middleware for quantum

is the abstraction layer to enable a seamless integration between classical ecosystems of applications and quantum resources, based in open-source software and services in multi-cloud/classical environments.

Quantum + Classical Integration



Middleware for quantum



Middleware for quantum

⁰¹ Quantum Serverless

⁰² Circuit Knitting toolbox

⁰¹ Quantum Serverless

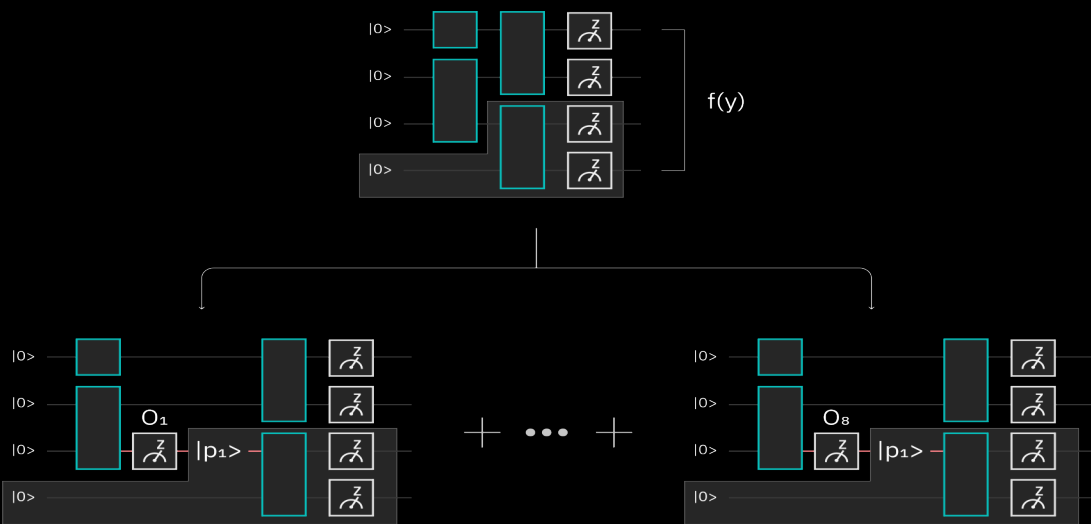
- Enabling flexible quantum-classical resource orchestration
- Integration with other circuit tools such Circuit Knitting, Nested Program, Compilation, Synthesis, Layout & Routing, Optimization
- Multi-cloud/HPC (Ray.io, Knative, CodeEngine, CodeFlare, OpenShift, LSF, Slurm)

02 Circuit Knitting toolbox

Circuit cutting

Simulate a large quantum circuit using small QPUs by cutting the circuit into subcircuits, which are then sent to QPUs

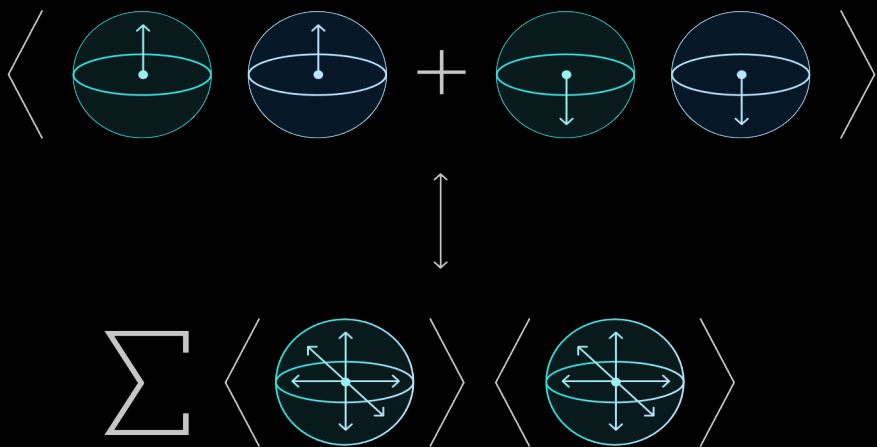
The output of the original circuit is built from classical post-processing of the subcircuits outputs



Entanglement forging

Break down a correlated system into smaller subsystems which can be tackled by smaller QPUs.

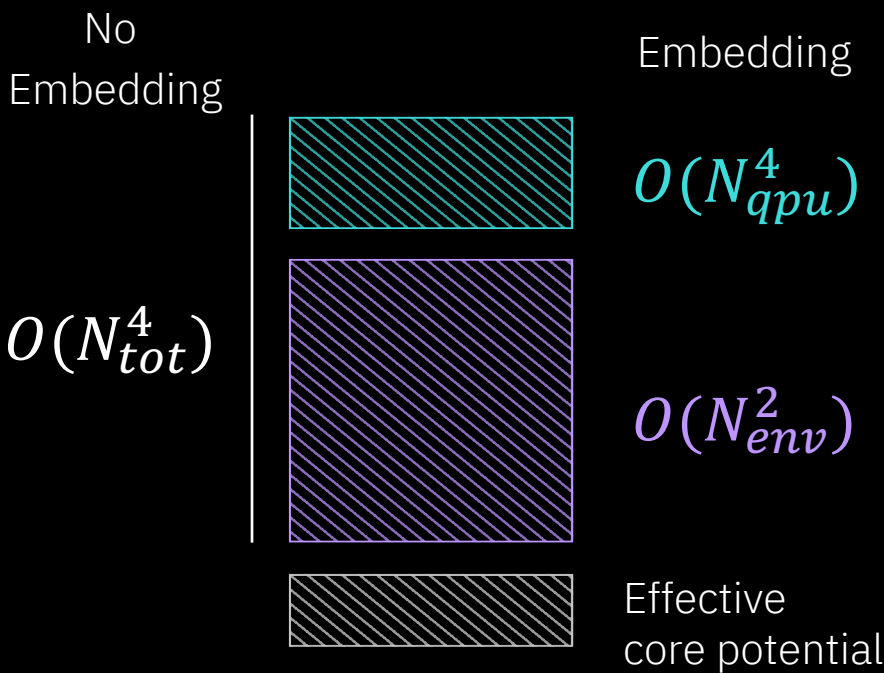
Recover the lost correlations with classical post-processing of the QPUs outputs



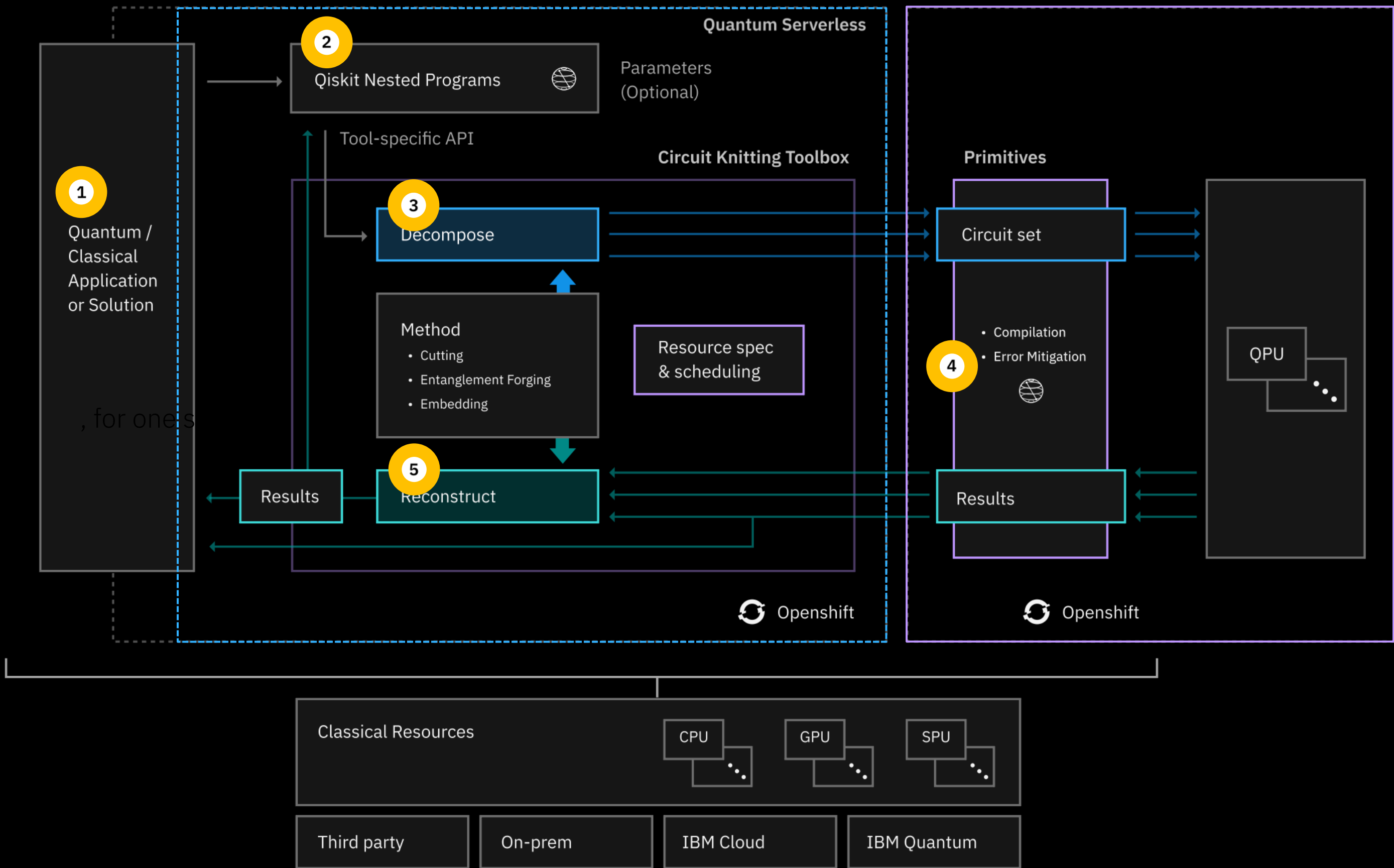
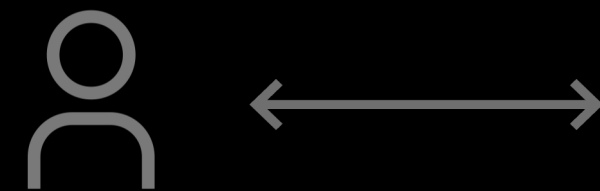
Embedding

Effective leveraging of QPU resources: Only the part of the problem which most benefits from exploiting entanglement is undertaken by the QPU

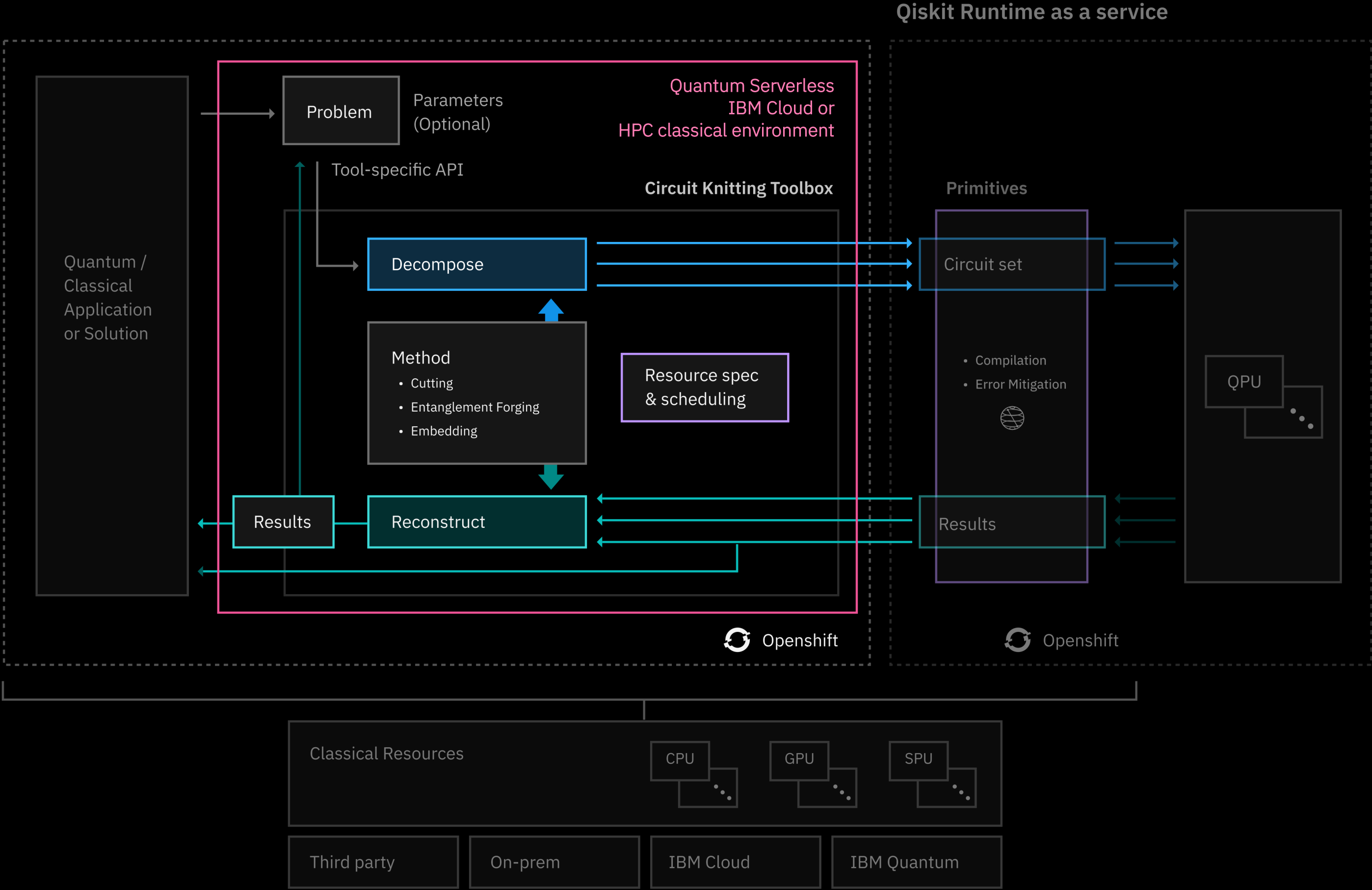
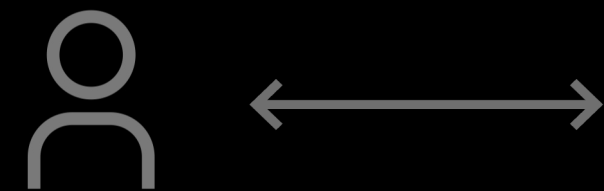
The CPU is efficient in tackling the remaining of the problem



Middleware
for quantum



Demo 01
Easy integration
using Quantum
Serverless



Demo 01 | Easy integration using Quantum Serverless

```
from quantum_serverless import run_qiskit_remote

@run_qiskit_remote()
def electronic_structure_problem(molecule, backend,
                                service):
    ...
    problem = ElectronicStructureProblem(driver)
    op = QubitConverter.convert(problem, ...)
    ...
    with Session(service, backend) as session:
        estimator = Estimator(session)
        vqe = EstimatorVQE(estimator, ansatz, ...)
        energy = vqe.compute_minimum_eigenvalue(op)
    return energy
```

Execute VQEs in parallel on the cloud:

```
electronic_structure_problem(molecule_0, backend_0)
```

```
electronic_structure_problem(molecule_1, backend_1)
```

•
•
•

```
electronic_structure_problem(molecule_N, backend_M)
```

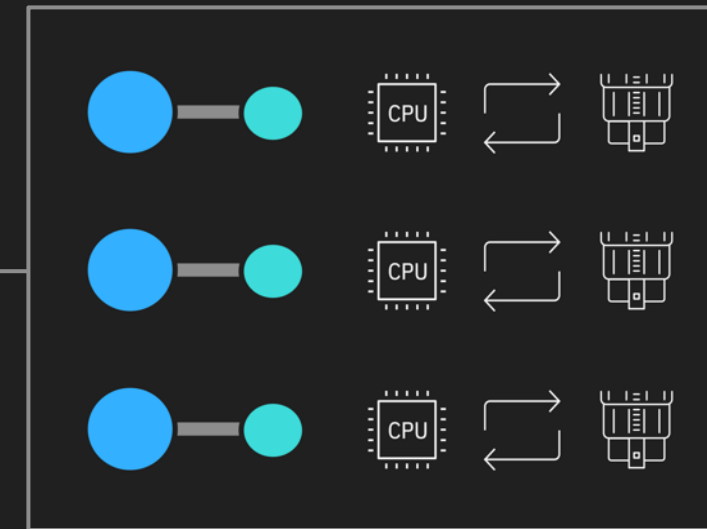
Demo 01 | Configuring a parallel workload

```
from qiskit_ibm_runtime import QiskitRuntimeService
from quantum_serverless import QuantumServerless
```

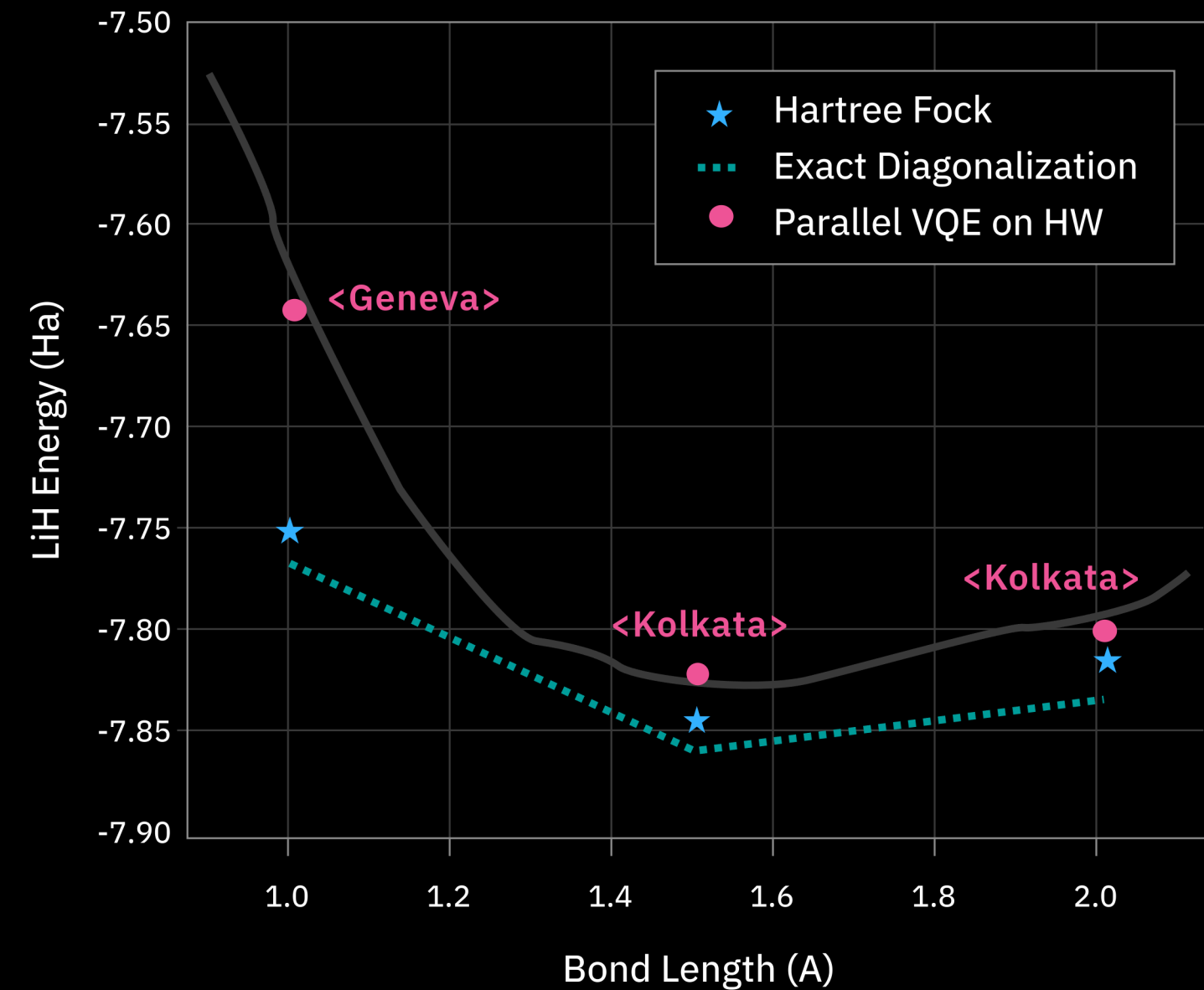
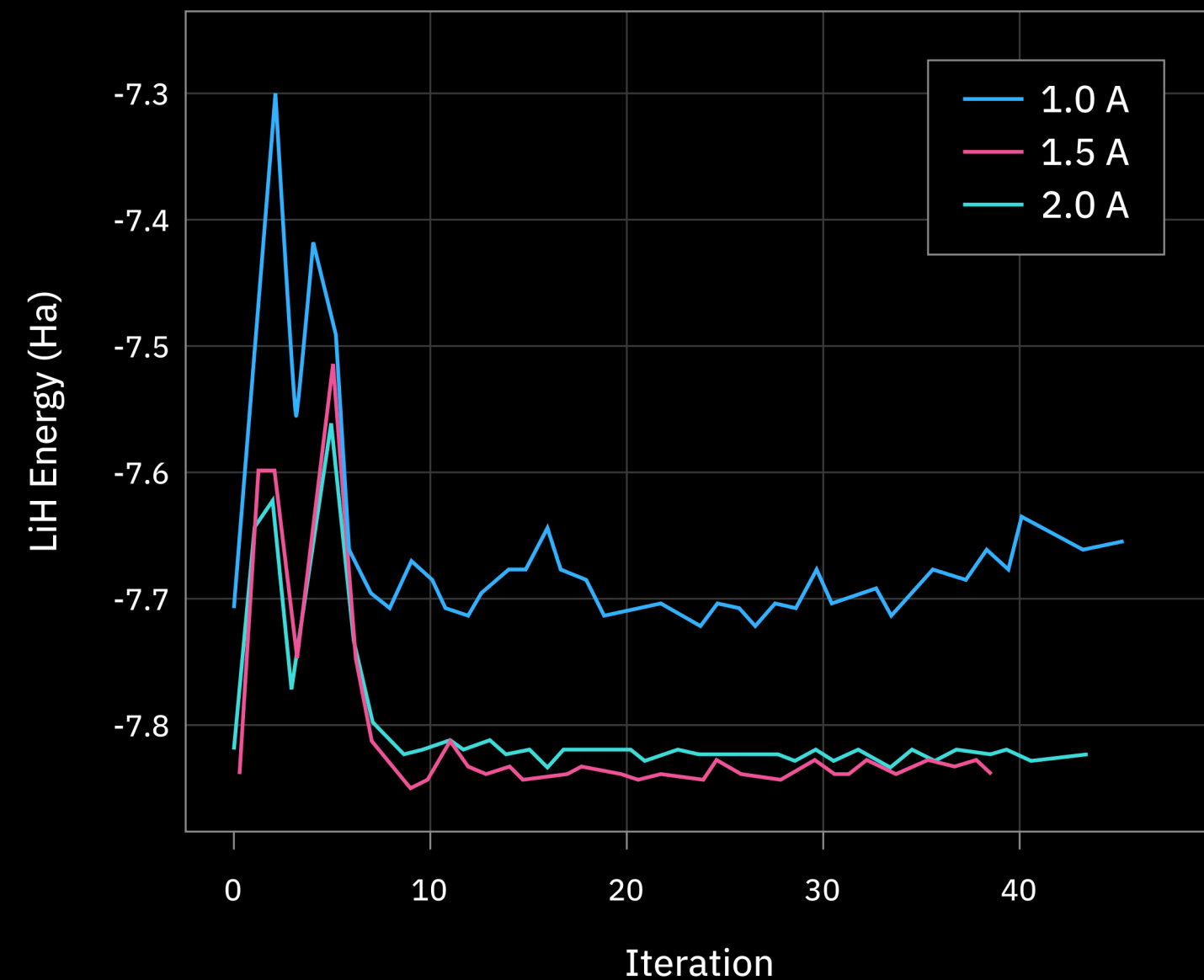
```
service = QiskitRuntimeService()
serverless = QuantumServerless(...)
```

```
with serverless.provider('ibm_cloud'): → Create serverless context
                                         with a cloud or HPC provider
```

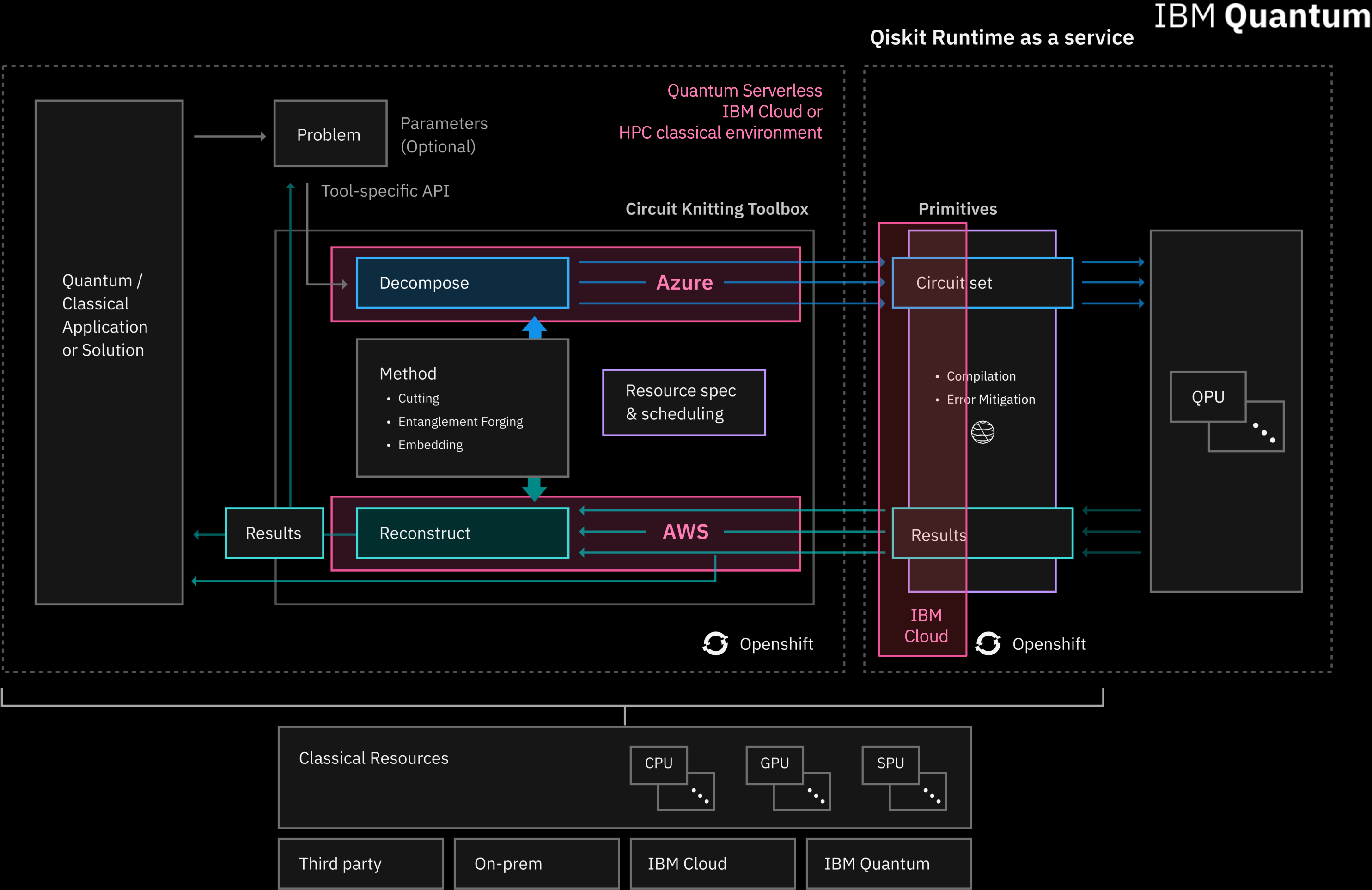
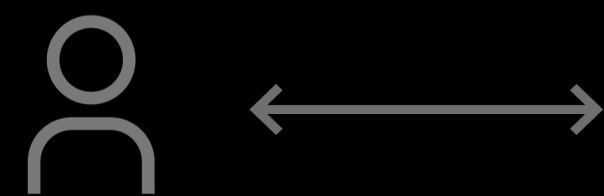
```
energies = electronic_structure_problem(
    molecules = [
        Molecule(geometry = [{"Li", [0, 0, 0]}, [{"H", [0, 0, 1.0]}]),
        Molecule(geometry = [{"Li", [0, 0, 0]}, [{"H", [0, 0, 1.5]}]),
        Molecule(geometry = [{"Li", [0, 0, 0]}, [{"H", [0, 0, 2.0]}])
    ],
    backends = [
        service.backend('ibmq_geneva'),
        service.backend('ibmq_kolkata'),
        service.backend('ibmq_auckland')
    ],
    ...
)
```



Demo 01 | Parallelized LiH ground state computations



Demo 02
Multicloud
workflows +
Circuit Knitting
toolbox



Demo 02 | Multicloud workflows + Circuit Knitting toolbox

```
service = QiskitRuntimeService(...)
backends = ["ibmq_kolkata", "ibmq_auckland"]

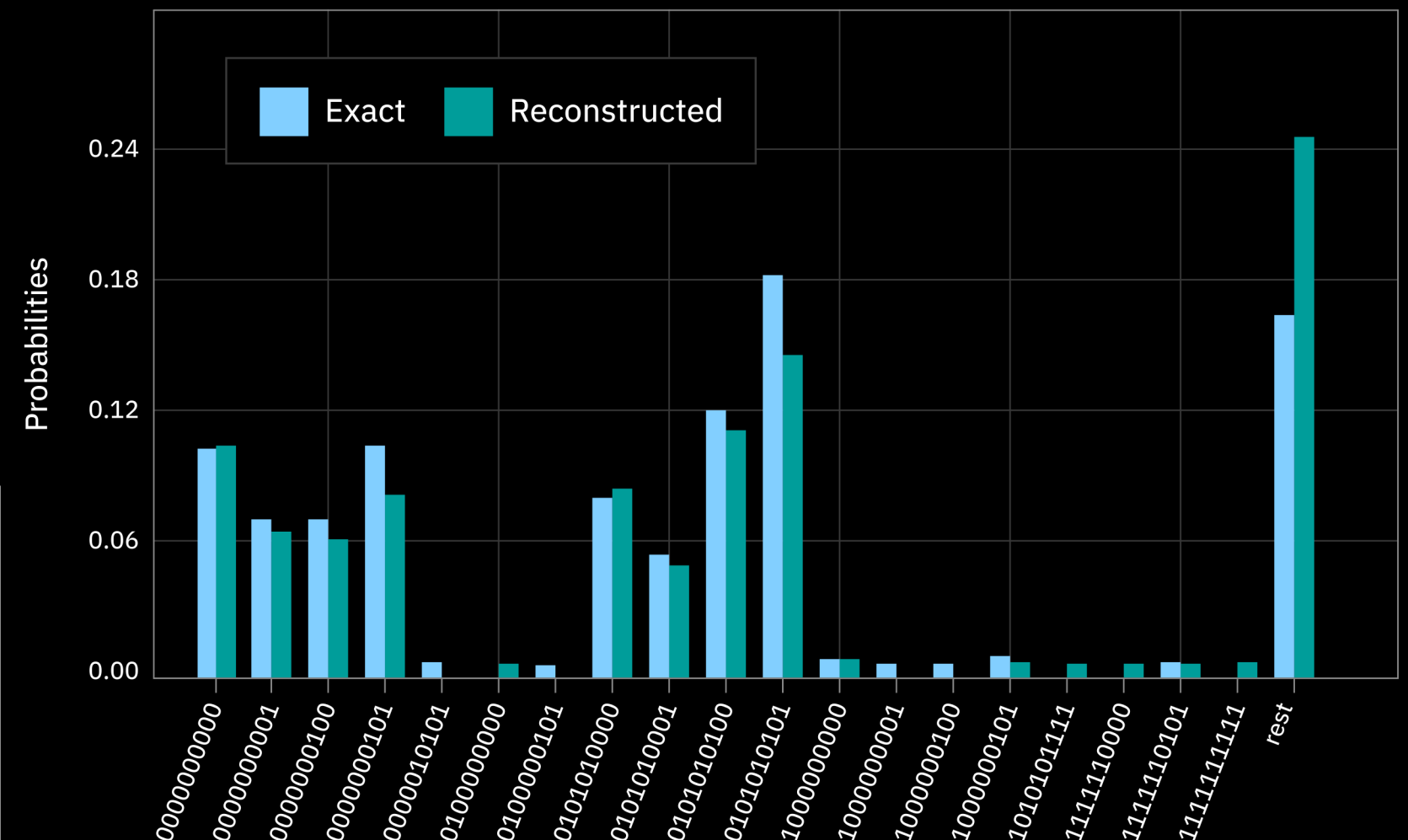
from circuit_knitting_toolbox.circuit_cutting
import WireCutter
cutter = WireCutter(circuit, service, backends)
```

```
# cut (decompose) the circuit on Azure
with serverless.provider('azure'):
    cuts = cutter.decompose("automatic", ...)
```

```
# execute subcircuits using Qiskit Runtime Primitives
with serverless.provider('ibmq_cloud'):
    subcircuit_results = cutter.evaluate(cuts)
```

```
# reconstruct probabilities on AWS
with serverless.provider('aws'):
    reconstructed_result =
cutter.recompose(subcircuit_results, cuts)
```

Reconstructed result



Multi-cloud or HPC solutions

01 Quantum Serverless

<https://github.com/Qiskit-Extensions/quantum-serverless>

02 Circuit Knitting toolbox

<https://github.com/Qiskit-Extensions/circuit-knitting-toolbox>

IBM Quantum