Practical HPCQC Integration with QDMI

A Real-Hardware Case Study with IQM Systems

Lukas Burgholzer CTO, MQSC · Senior Researcher, TUM CDA

Marcel Walter · Patrick Hopf · Álvaro Caride-Tabarés Sánchez
Teemu Mattsson · Bernd Hoffmann · Noora Färkkilä
Daniel Bulmash · Robert Wille · Eric Mansfield

Munich Quantum Software Company
Technical University of Munich
IQM Quantum Computers

QDMI-on-IQM

HPCQC Integration

Powered by QDMI

The Implementation

Source · Installation · Examples

github.com/iqm-finland/QDMI-on-IQM

The Paper

Burgholzer, Walter, et al., 2026

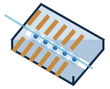
arxiv.org/abs/2604.19869

Software that connects the quantum ecosystem

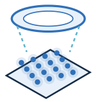
Quantum Processors



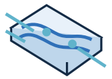
Superconducting



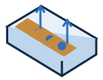
Trapped Ions



Neutral Atoms

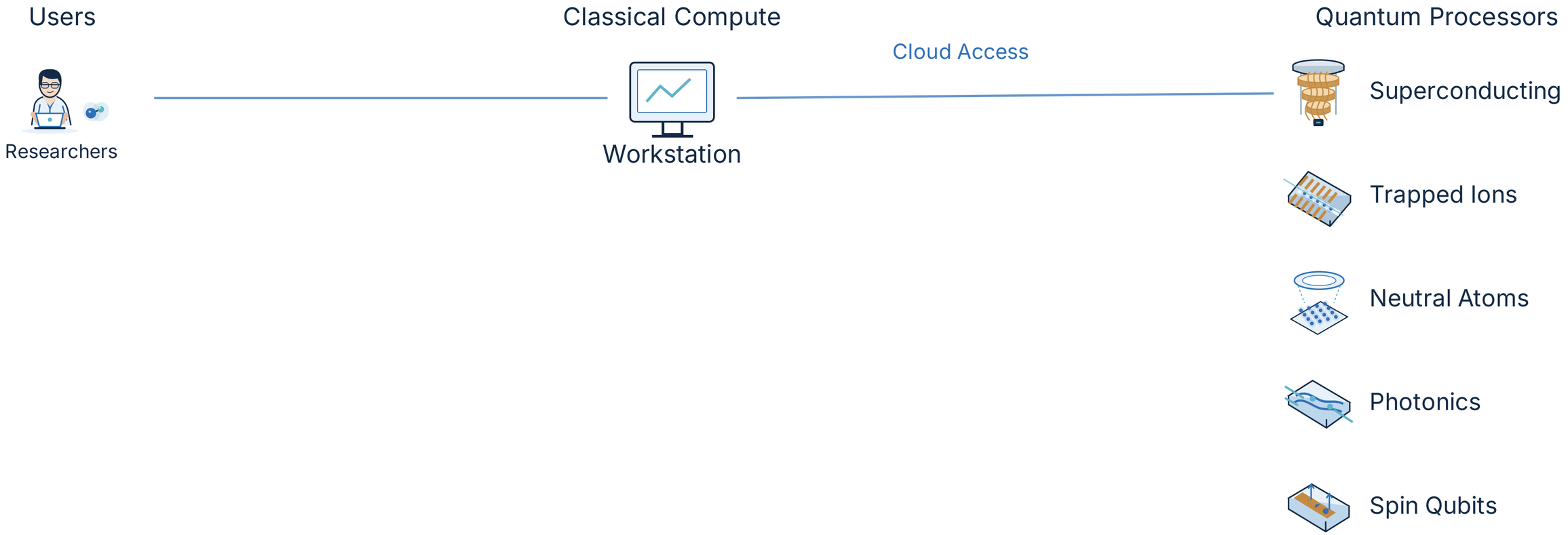


Photonics

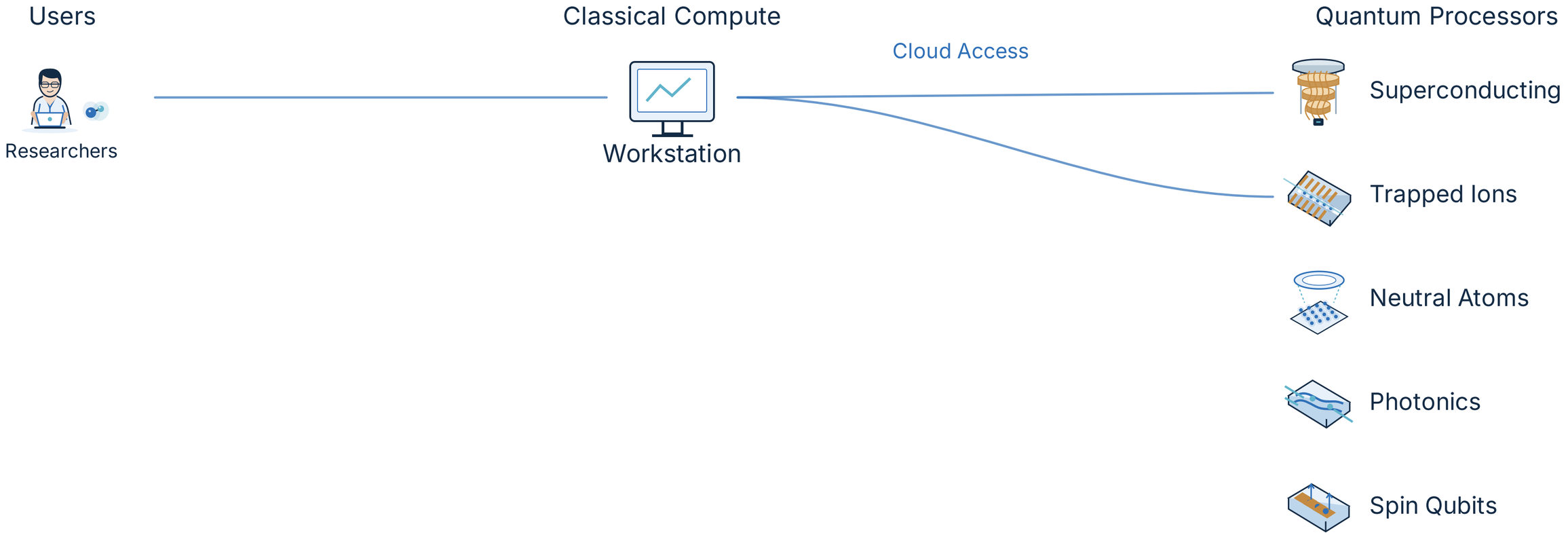


Spin Qubits

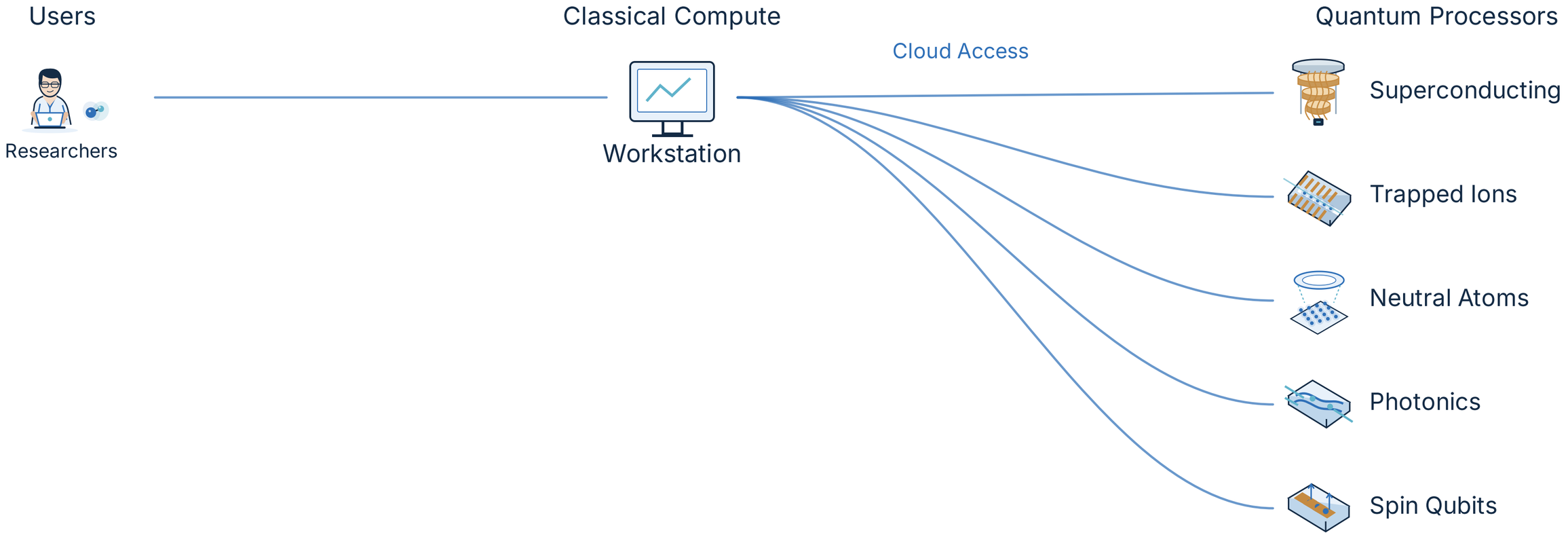
Software that connects the quantum ecosystem



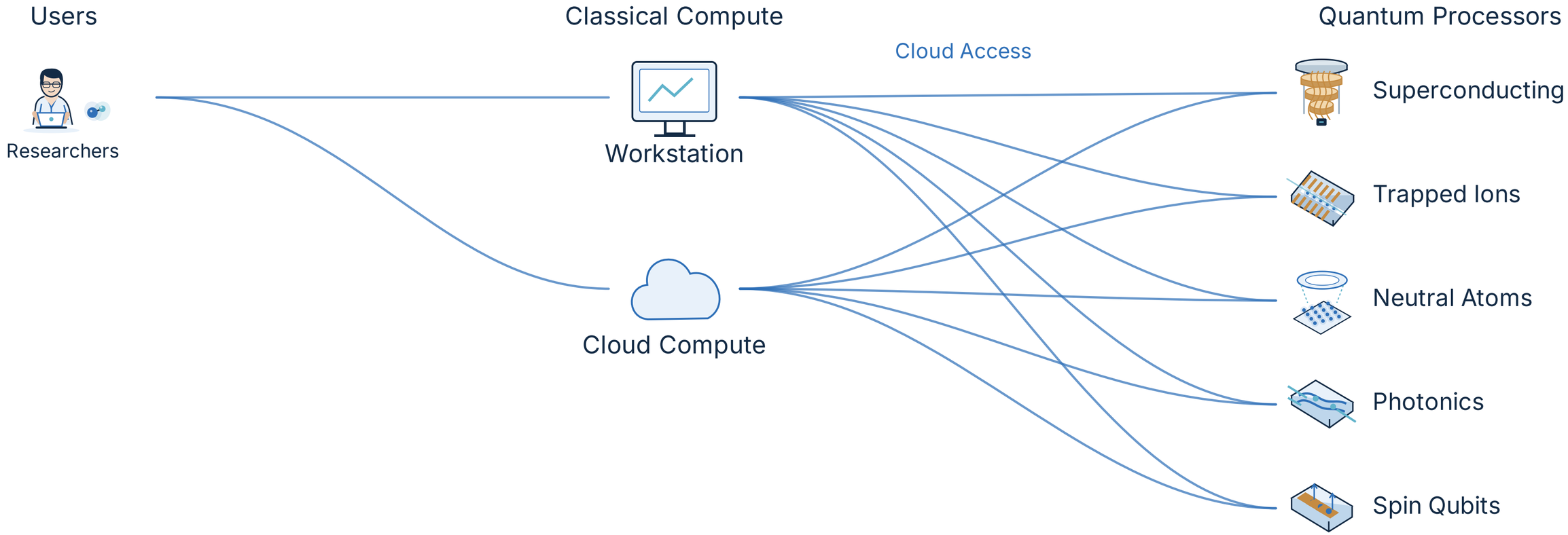
Software that connects the quantum ecosystem



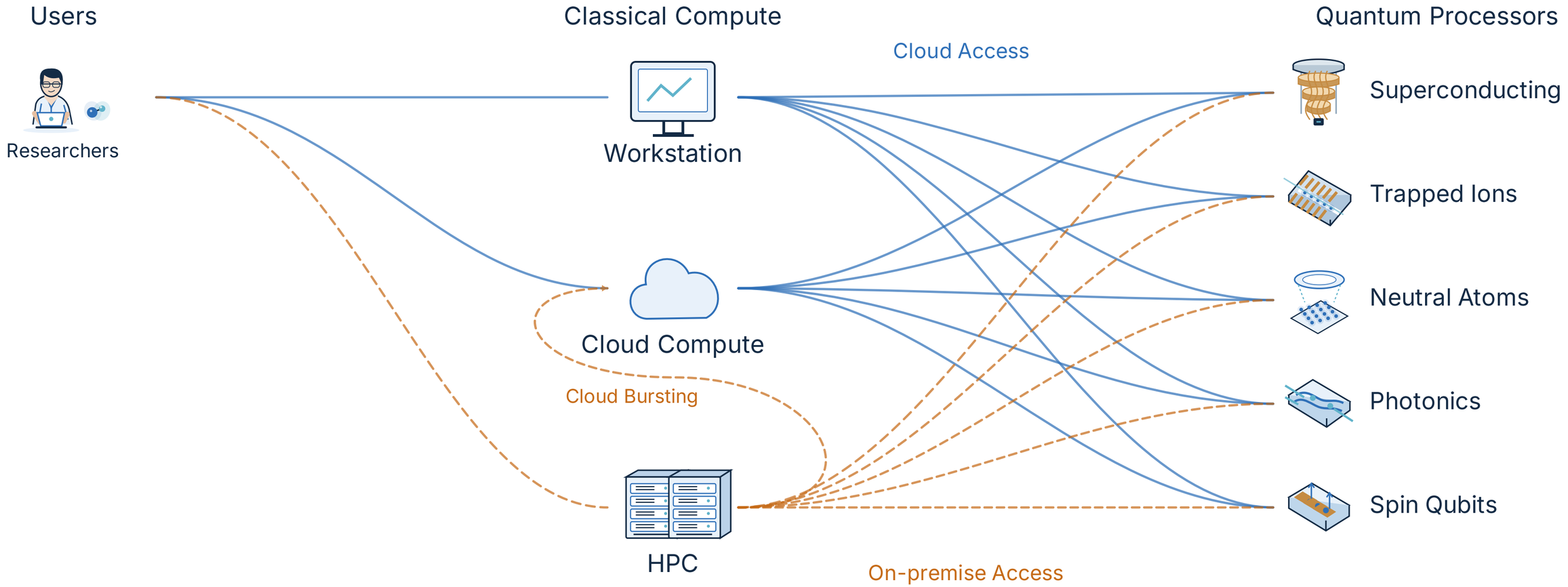
Software that connects the quantum ecosystem



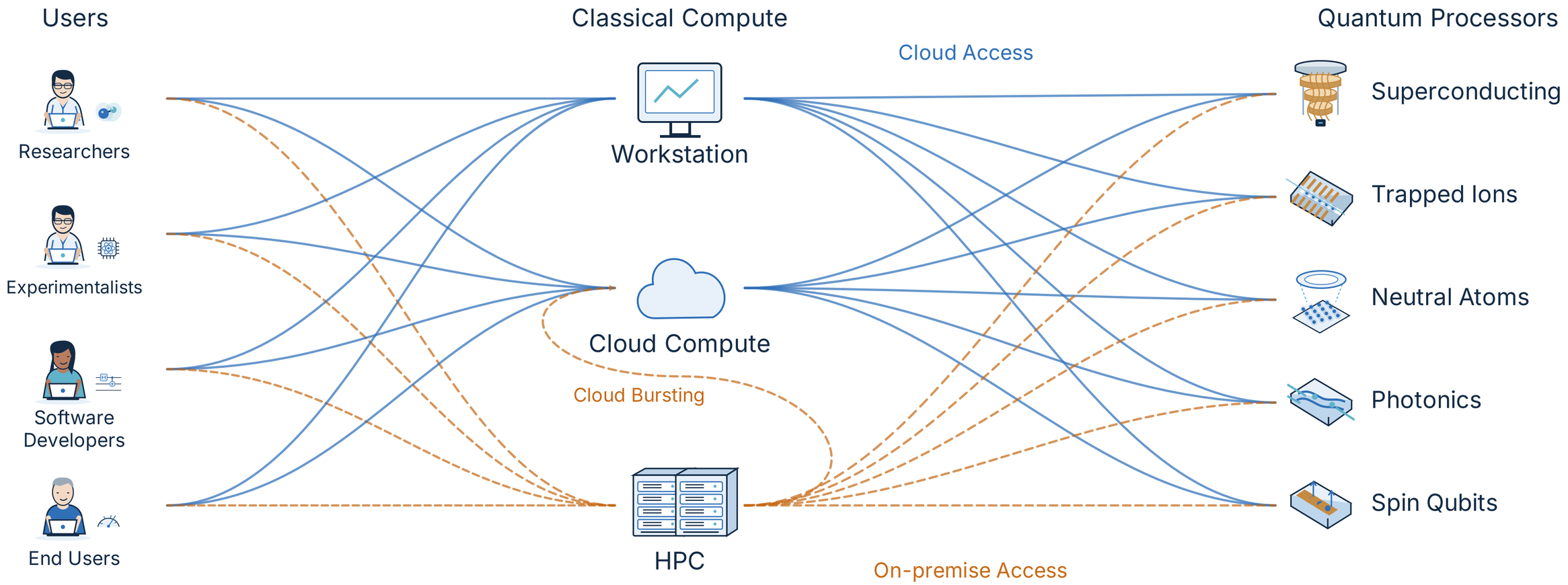
Software that connects the quantum ecosystem



Software that connects the quantum ecosystem



Software that connects the quantum ecosystem



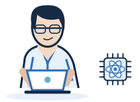
A Shared Software Stack

Modular · Efficient · Extensible

Users



Researchers



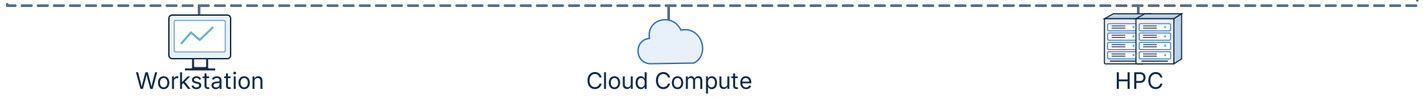
Experimentalists



Software Developers



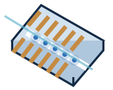
End Users



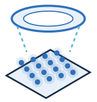
Quantum Processors



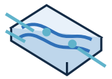
Superconducting



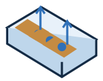
Trapped Ions



Neutral Atoms



Photonics



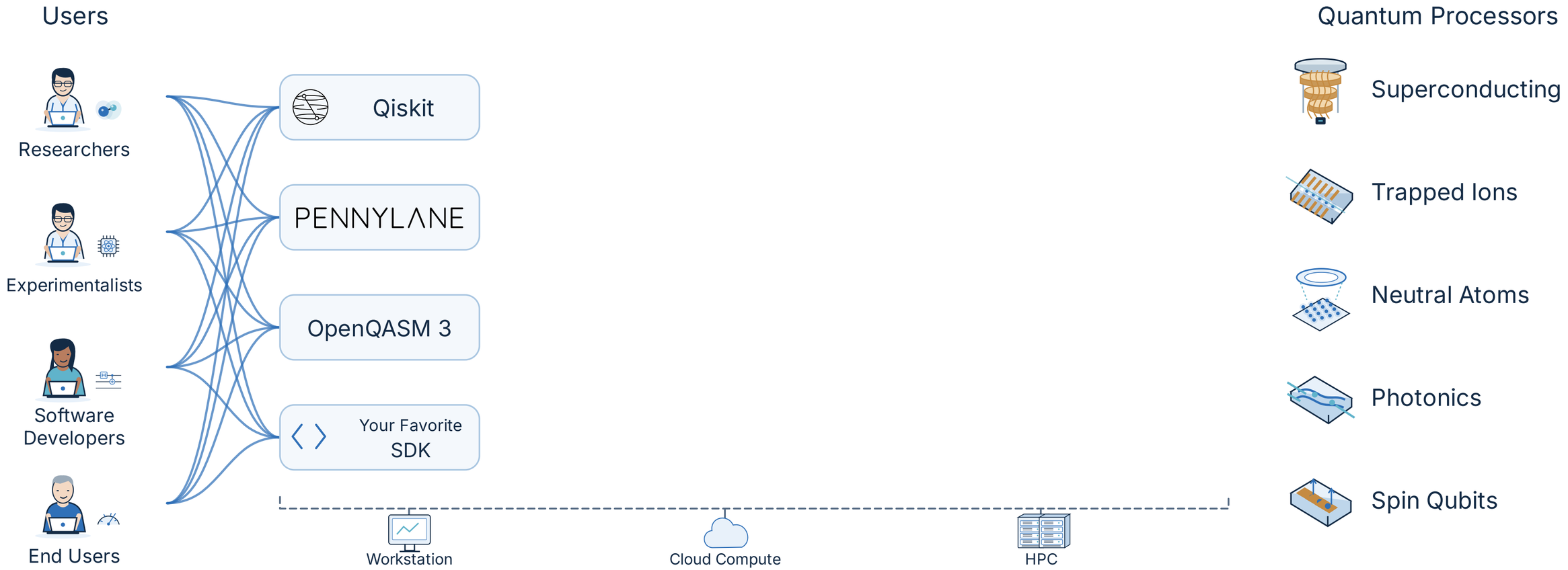
Spin Qubits

MQSS Munich Quantum Software Stack

Burgholzer, Echavarria, et al., 2026
doi.org/10.1145/3773656.3773669

A Shared Software Stack

Modular · Efficient · Extensible

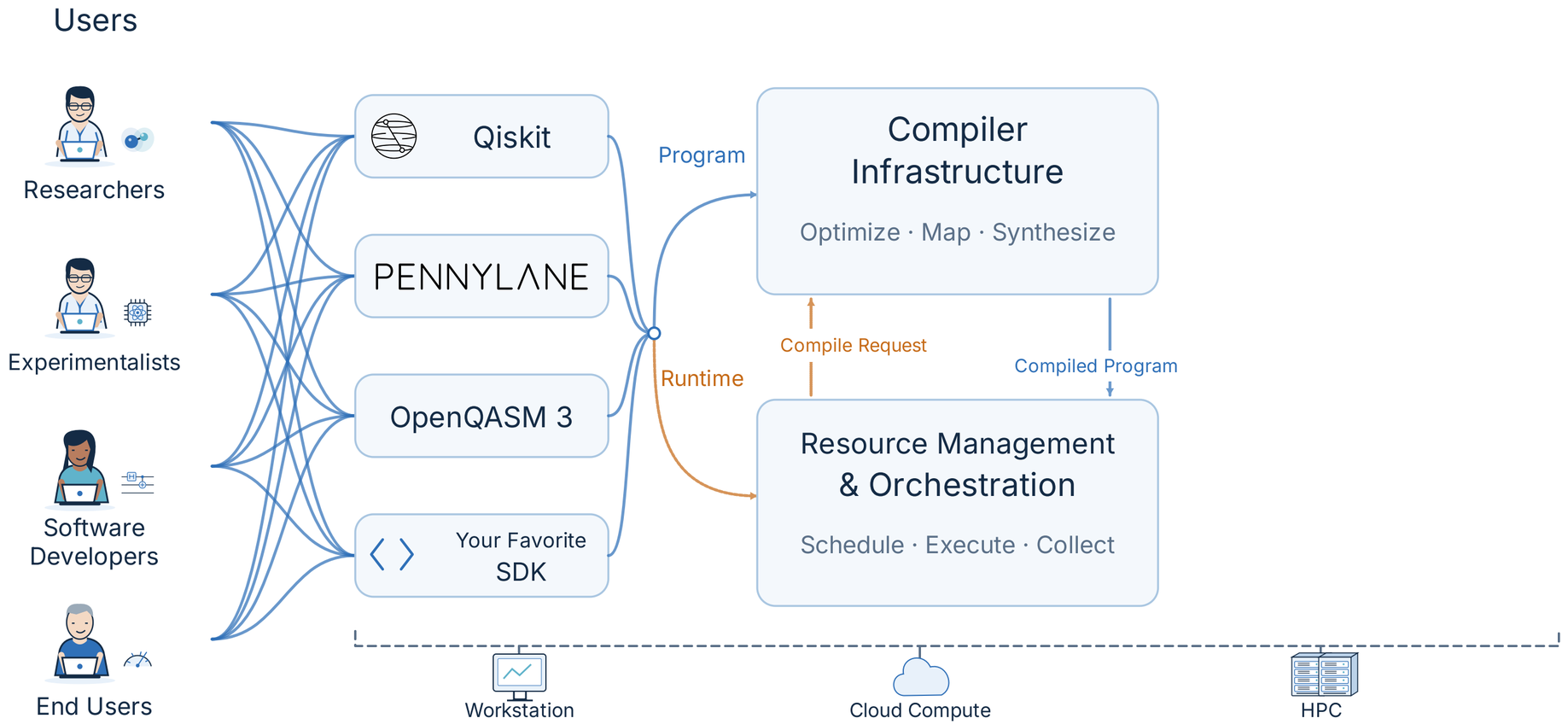


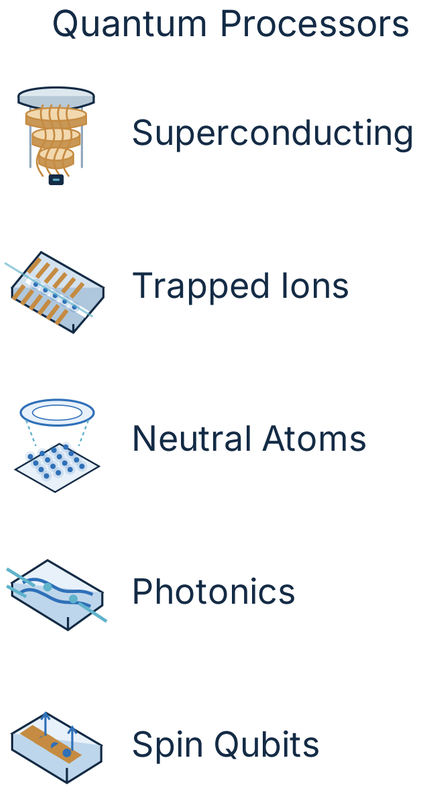
MQSS Munich Quantum Software Stack

Burgholzer, Echavarria, et al., 2026
doi.org/10.1145/3773656.3773669

A Shared Software Stack

Modular · Efficient · Extensible



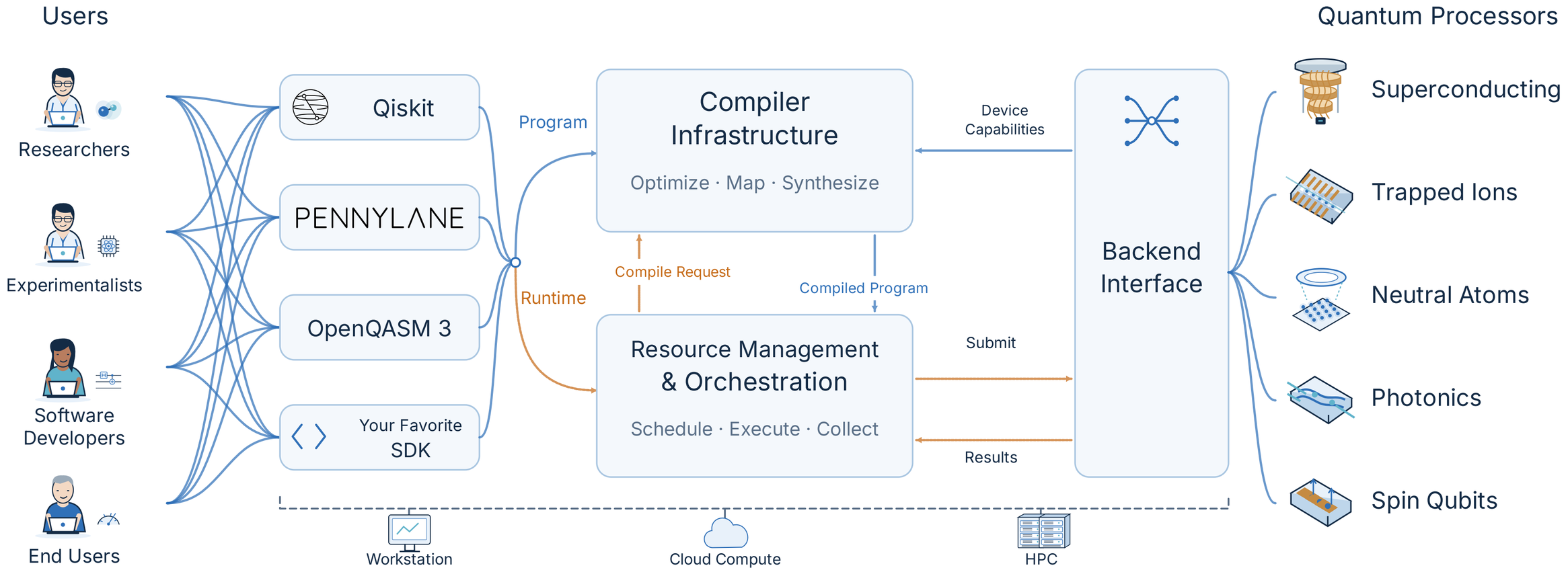


MQSS Munich Quantum Software Stack

Burgholzer, Echavarria, et al., 2026
doi.org/10.1145/3773656.3773669

A Shared Software Stack

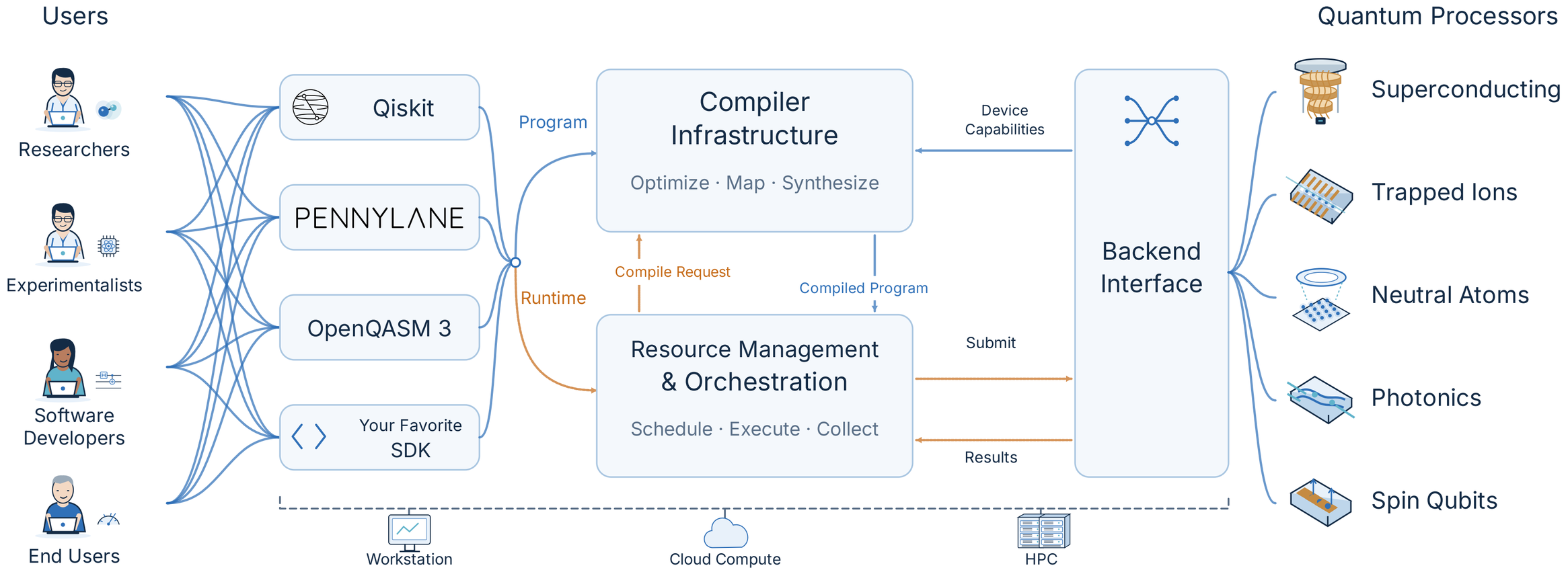
Modular · Efficient · Extensible



MQSS Munich Quantum Software Stack
Burgholzer, Echavarria, et al., 2026
doi.org/10.1145/3773656.3773669

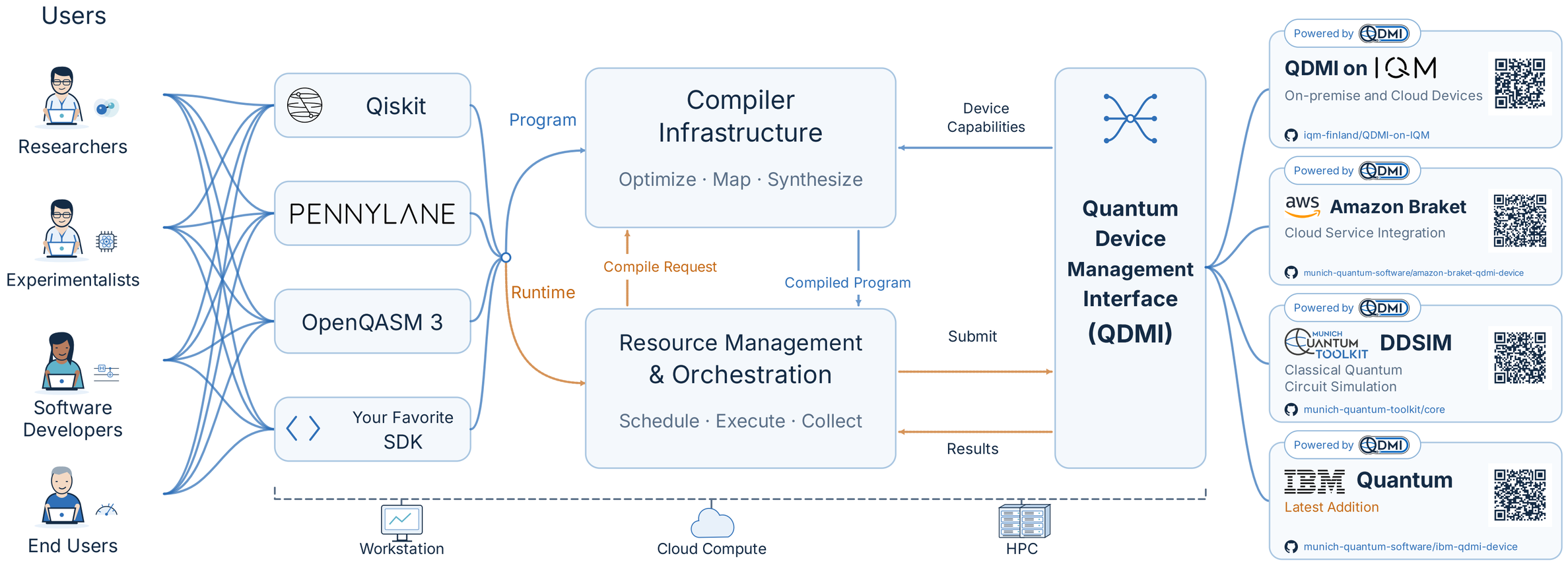
Our Approach: From the Metal to the User

Modular · Efficient · Extensible



Our Approach: From the Metal to the User

Modular · Efficient · Extensible



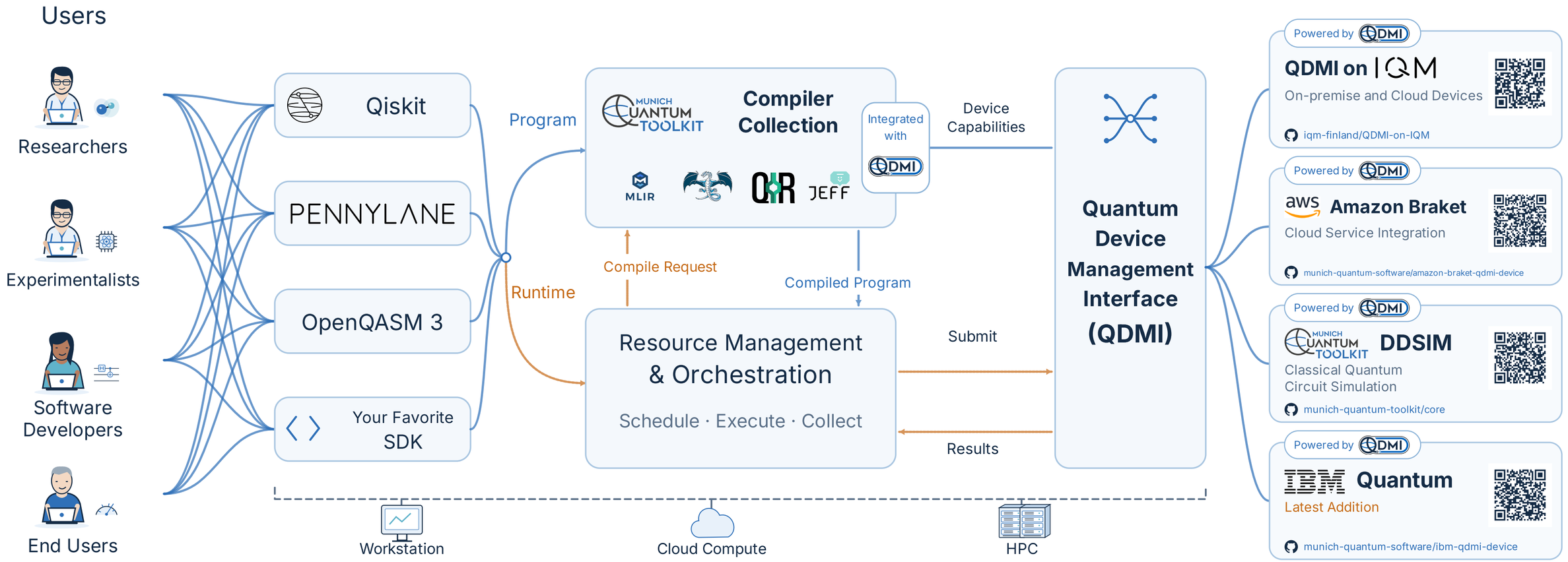
QDMI

Created by TUM CDA, TUM CAPS & LRZ · Maintained by MQV gGmbH & MQSC

Munich-Quantum-Software-Stack/QDMI ★ 89

Our Approach: From the Metal to the User

Modular · Efficient · Extensible



MQT Core

Created and Maintained by MQSC · TUM CDA · MQT Community

munich-quantum-toolkit/core ★ 136 977k PyPI downloads

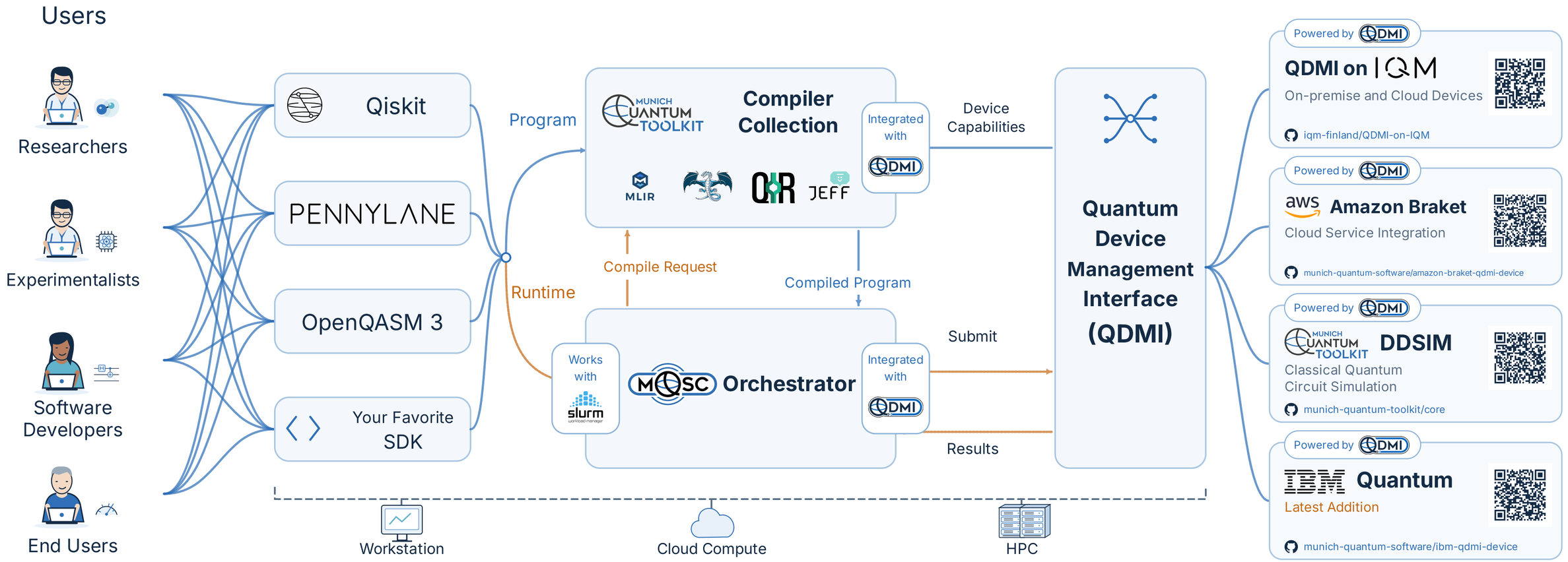
QDMI

Created by TUM CDA, TUM CAPS & LRZ · Maintained by MQV gGmbH & MQSC

Munich-Quantum-Software-Stack/QDMI ★ 89

Our Approach: From the Metal to the User

Modular · Efficient · Extensible



MQT Core MQSC TUM

Created and Maintained by MQSC · TUM CDA · MQT Community

munich-quantum-toolkit/core ★ 136 977k PyPI downloads

QDMI TUM lrz MQV MQSC

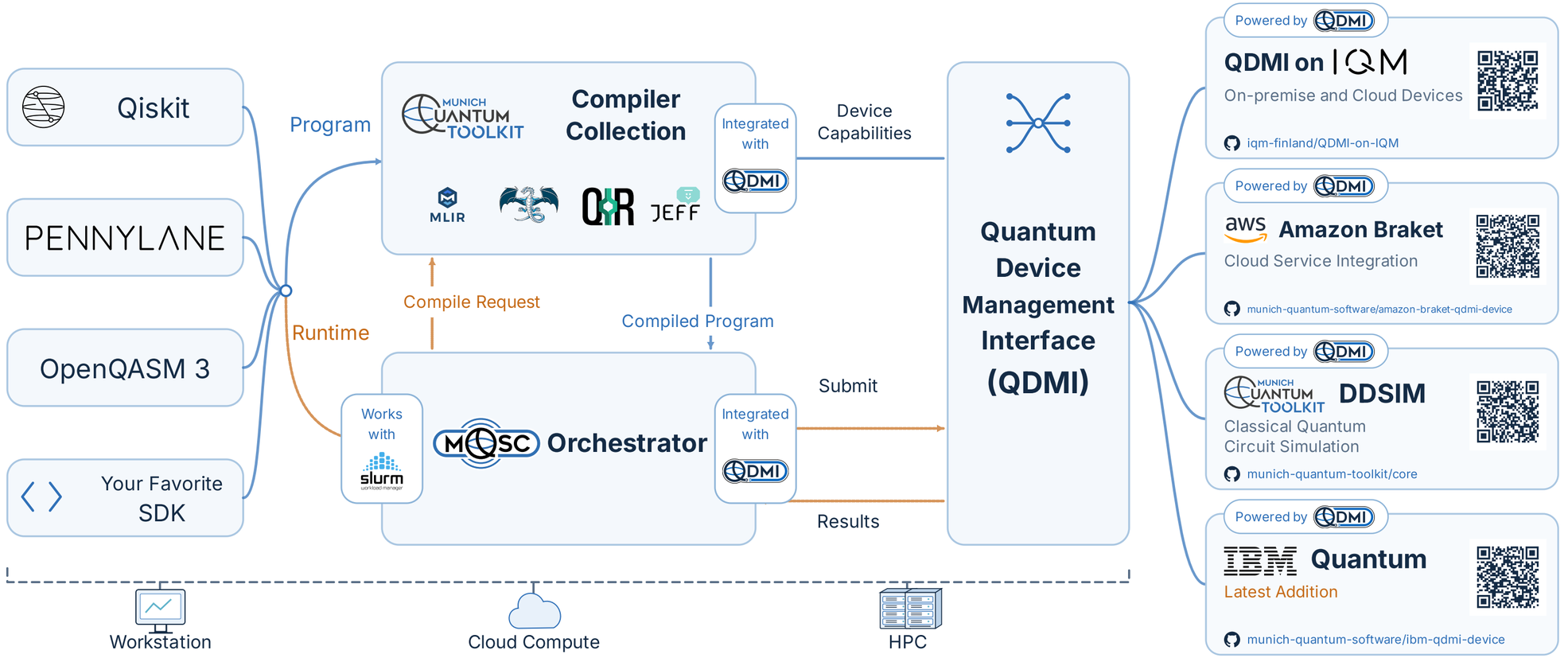
Created by TUM CDA, TUM CAPS & LRZ · Maintained by MQV gGmbH & MQSC

Munich-Quantum-Software-Stack/QDMI ★ 89

Our Approach: From the Metal to the User

Modular · Efficient · Extensible

- Open by Design
- Community-Driven
- No Lock-Ins
- From Research to Practice
- Performance at Every Layer



Your Device Could Be Next

MQT Core

Created and Maintained by MQSC · TUM CDA · MQT Community

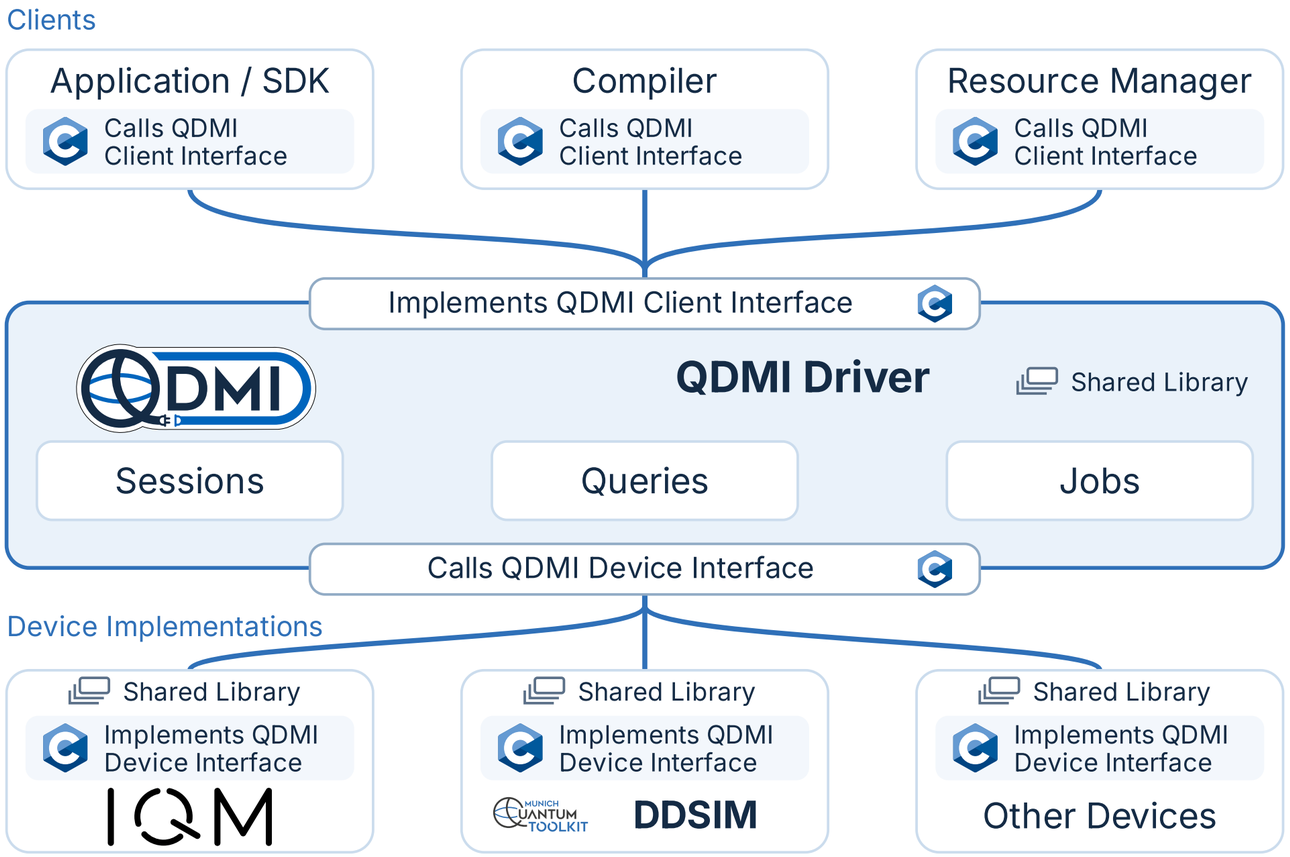
munich-quantum-toolkit/core ★ 136 977k PyPI downloads

QDMI

Created by TUM CDA, TUM CAPS & LRZ · Maintained by MQV gGmbH & MQSC

Munich-Quantum-Software-Stack/QDMI ★ 89

The Quantum Device Management Interface



QDMI

Open, Vendor-Neutral Interface

Device Discovery

Access devices through a QDMI driver

Capabilities and Calibration

Query sites, operations, and device properties

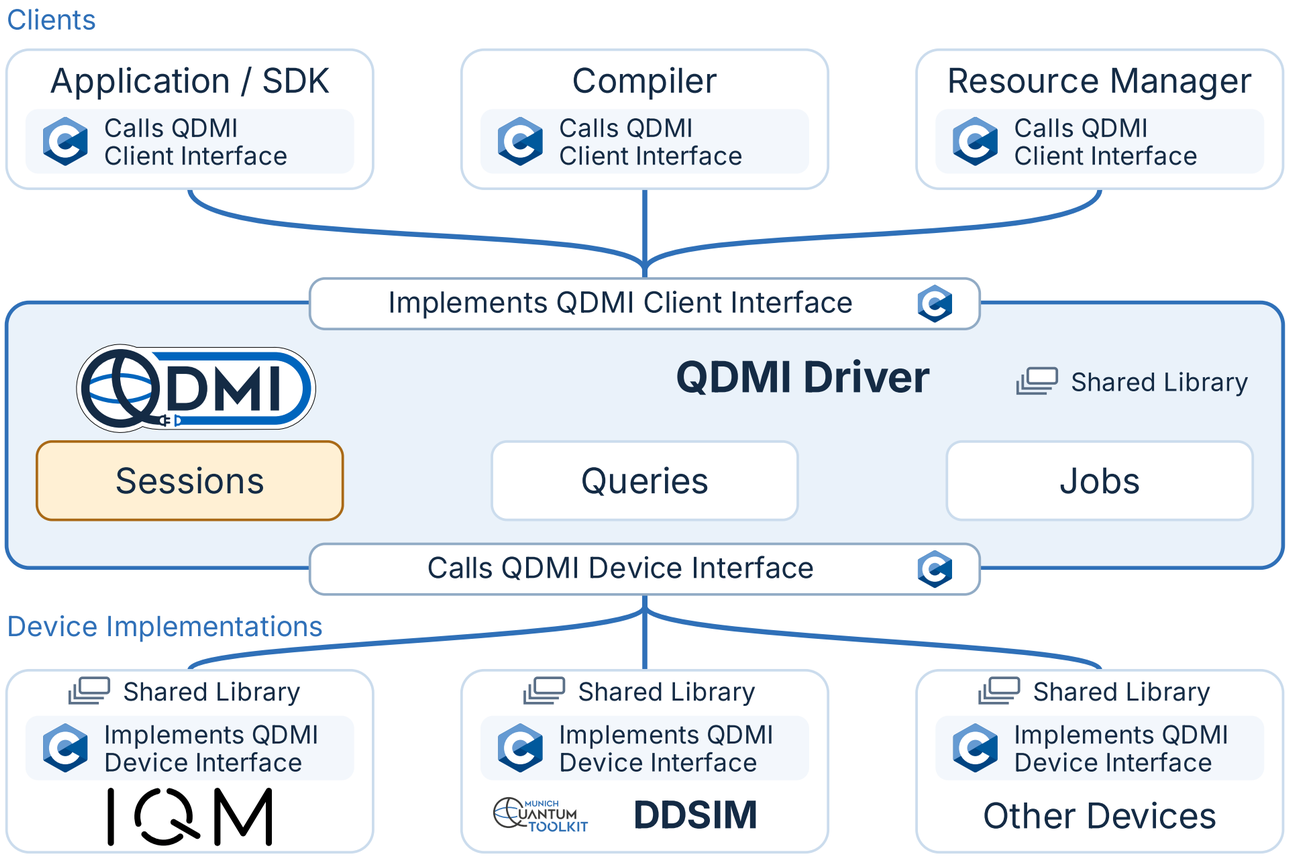
Job Execution

Submit programs and retrieve results

Specification & Ecosystem

github.com/Munich-Quantum-Software-Stack/QDMI

The Quantum Device Management Interface



1 · SESSIONS

Session Configuration

Allocate

Create a Session with the driver

Configure

Set endpoint and authentication parameters

Initialize

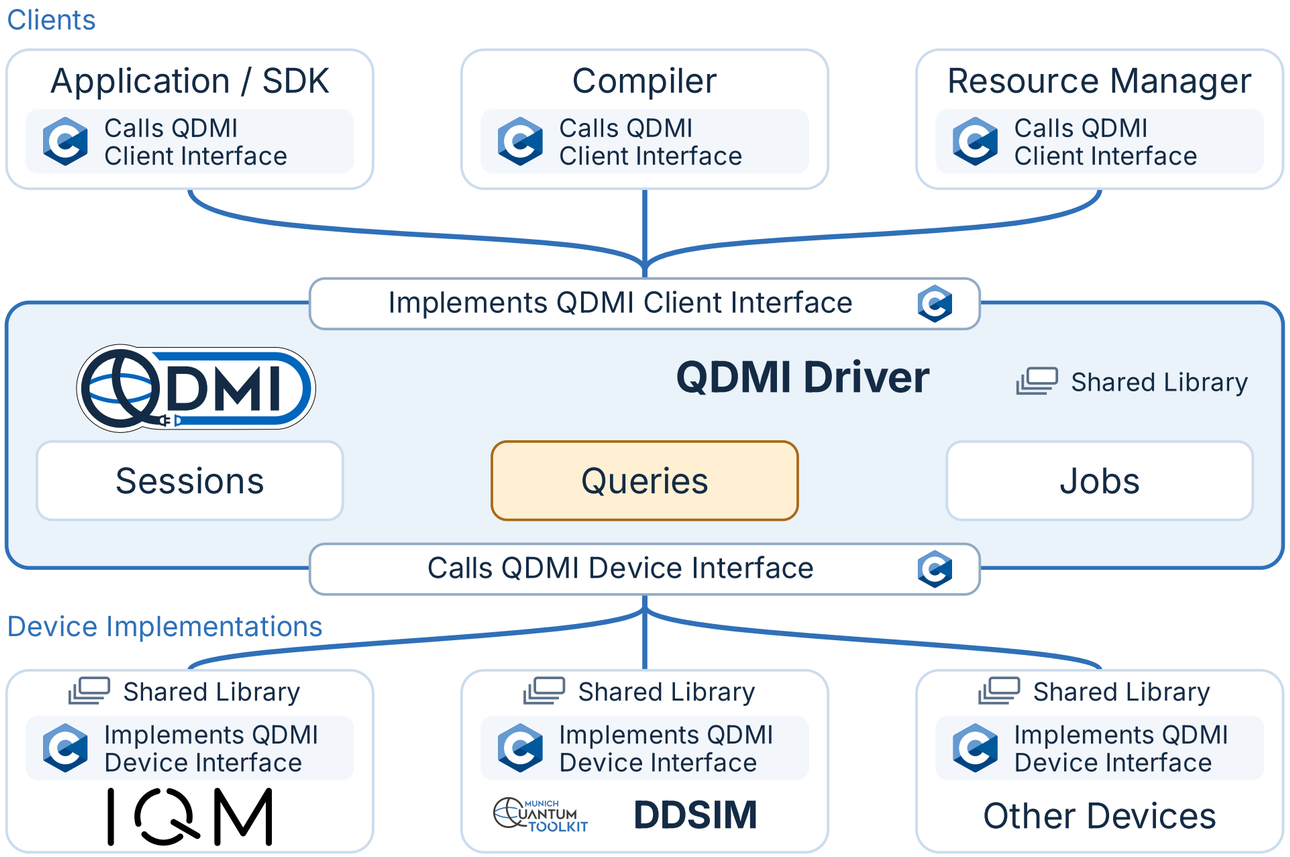
Access the configured devices

Device Sessions connect the driver to device implementations.

Specification & Ecosystem

github.com/Munich-Quantum-Software-Stack/QDMI

The Quantum Device Management Interface



2 · QUERIES

Device Properties, Sites, and Operations

Compiler

Connectivity and supported operations

Calibration

Coherence and operation quality

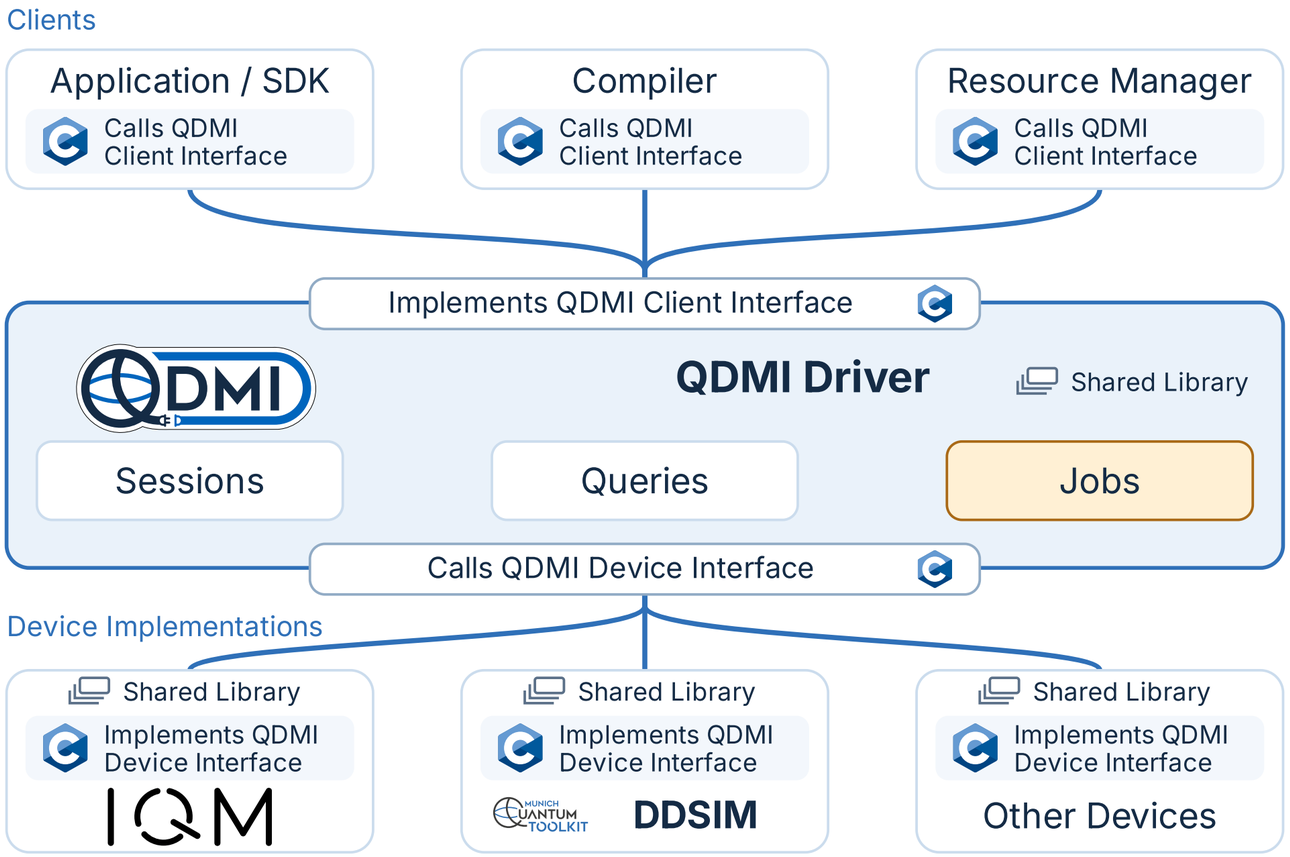
Resource Manager

Device status and queue information

Specification & Ecosystem

github.com/Munich-Quantum-Software-Stack/QDMI

The Quantum Device Management Interface



3 · JOBS

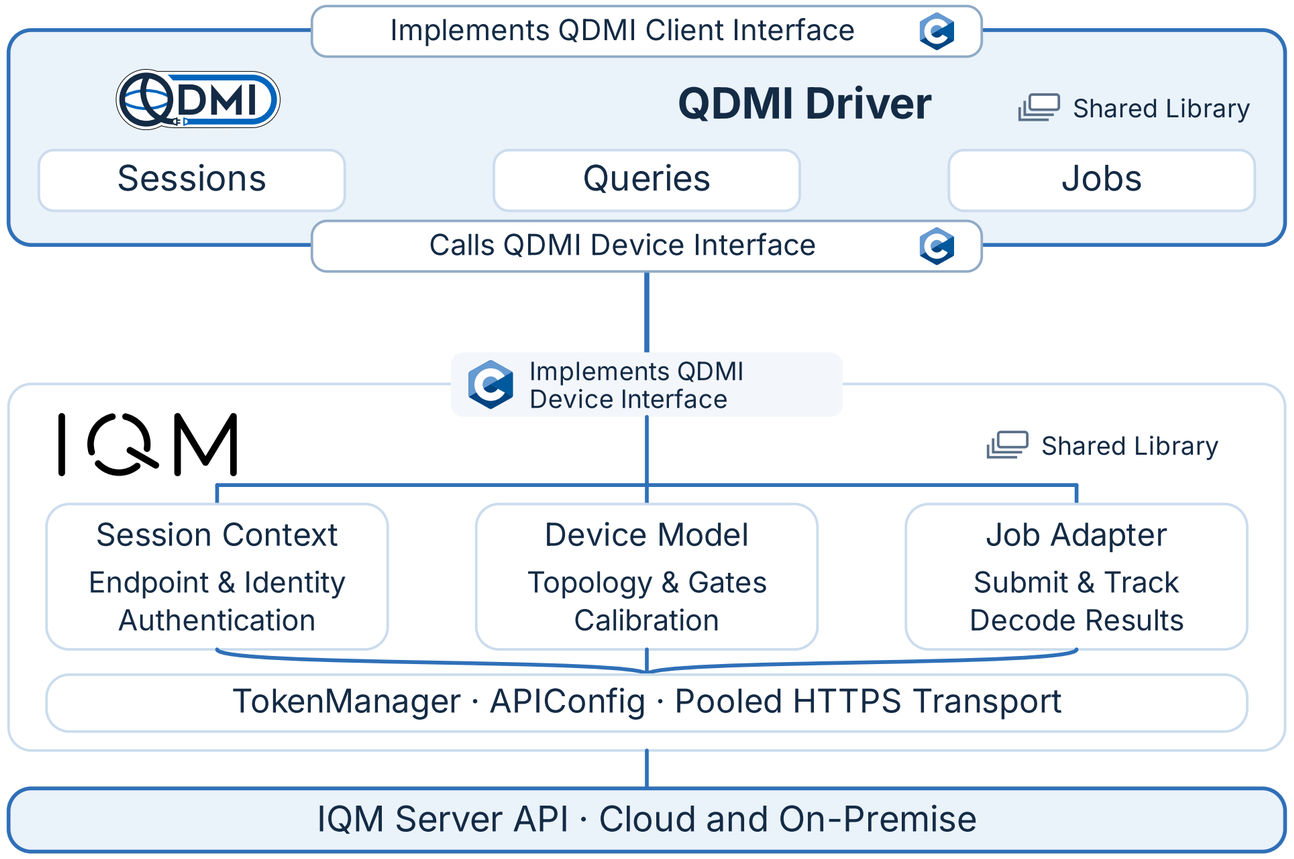
Job Execution and Results

1 Create	2 Configure
3 Submit	4 Check / Wait
5 Results	6 Free

One or more programs per Job.

Specification & Ecosystem
github.com/
Munich-Quantum-Software-Stack/QDMI

Connecting QDMI to IQM Systems



IMPLEMENTATION

QDMI-on-IQM

QDMI Device Interface

Sessions, queries, Jobs, and results

Shared Support Layer

Token handling, API routes, connection reuse

Configured Deployment

IQM endpoint and selected system

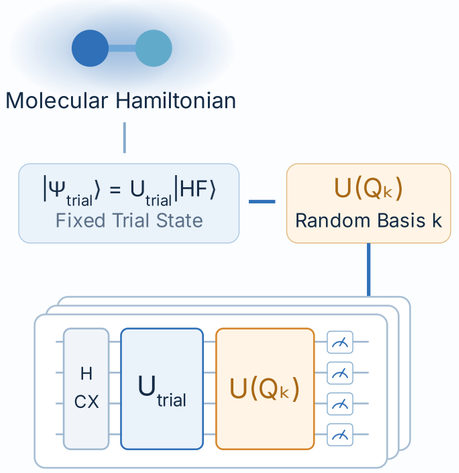
QDMI-on-IQM
github.com/iqm-finland/QDMI-on-IQM

Quantum-Classical Auxiliary-Field Quantum Monte Carlo · QC-AFQMC

A Classical–Quantum Workflow

Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

Classical Preparation



One Random Basis Q_k per Program

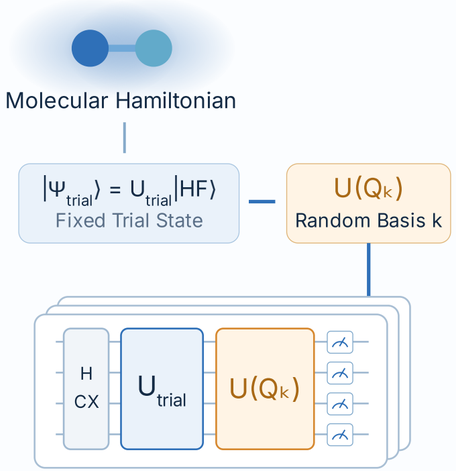
PENNYLANE

arxiv.org/abs/2506.22408
QC-AFQMC · Algorithm and Analysis

amazon-braket/amazon-braket-examples
Amazon Braket · Public AFQMC Example

Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

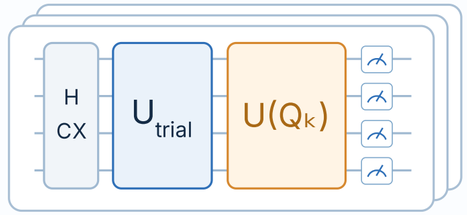
Classical Preparation



PENNYLANE

Quantum Shadow Collection

4,096 Circuits × 1,024 Shots



One Random Basis Q_k per Program

Program 0	Program 1	Program 2	Program 3
0000	1110	1000	1001
1001	1011	0111	1111
0000	1101	0111	1010
⋮	⋮	⋮	⋮

...

MUNICH QUANTUM TOOLKIT Compiler Collection

QDMI

arxiv.org/abs/2506.22408
QC-AFQMC · Algorithm and Analysis

aws

amazon-braket/amazon-braket-examples
Amazon Braket · Public AFQMC Example

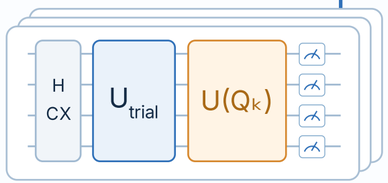
Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

Classical Preparation

Molecular Hamiltonian

$|\Psi_{trial}\rangle = U_{trial}|HF\rangle$
Fixed Trial State

$U(Q_k)$
Random Basis k

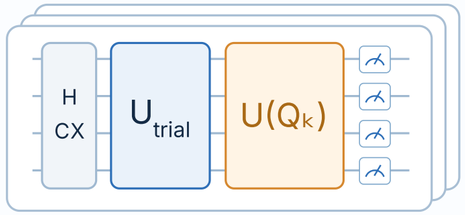


One Random Basis Q_k per Program

PENNYLANE

Quantum Shadow Collection

4,096 Circuits × 1,024 Shots



One Random Basis Q_k per Program

Program 0	Program 1	Program 2	Program 3	
0000	1110	1000	1001	…
1001	1011	0111	1111	
0000	1101	0111	1010	
⋮	⋮	⋮	⋮	

MUNICH QUANTUM TOOLKIT Compiler Collection

QDMI

Classical Monte Carlo

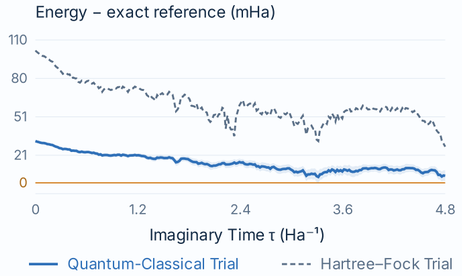
Propagate → Overlap → Reweight → Reduce



τ = 4.78 Ha⁻¹ · filled area ∝ weight

$$\widehat{E}(\tau) = \frac{\sum_w W_w(\tau)\,\mathrm{Re}\,E_L[\phi_w(\tau)]}{\sum_w W_w(\tau)}$$

Weighted Mean



MQSC Orchestrator

arxiv.org/abs/2506.22408
QC-AFQMC · Algorithm and Analysis

aws

amazon-braket/amazon-braket-examples
Amazon Braket · Public AFQMC Example

Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

Classical Preparation

PENNYLANE

Trial State and Gaussian Rotations

```python
def hydrogen_trial() -> None:
    import pennylane as qml
    qml.CNOT(wires=[0, 1])
    qml.SingleExcitation(float(TRIAL_PARAMETERS[0]), wires=[0, 2])
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[2, 3])

def apply_gaussian_givens(thetas, schedule: List[Tuple[int, int]]) -> None:
    thetas = np.asarray(thetas)
    for k, (i, _) in enumerate(schedule):
        angle = thetas[..., k]          # scalar, or one value per batch element
        if i % 2 == 0:
            qp.RZ(angle, wires=i // 2)
        else:
            qp.IsingXX(angle, wires=[(i - 1) // 2, (i + 1) // 2])
```

0 / 4,096 Programs Prepared

Shadow Circuit and Batch

```python
def apply_pauli_layer(pauli_vector) -> None:
    pauli_vector = np.asarray(pauli_vector)
    num_modes = pauli_vector.shape[-1] // 2
    for p in range(num_modes):
        qp.RX(np.pi * (pauli_vector[..., p] % 2), wires=p)
        qp.RZ(np.pi * (pauli_vector[..., p + num_modes] % 2), wires=p)

@qml.set_shots(shots=args.shots)
@qml.qnode(device)
def shadow_circuit(thetas: NDArray, paulis: NDArray) -> tuple:
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0, 1])
    hydrogen_trial()
    apply_gaussian_givens(thetas, schedule)
    apply_pauli_layer(paulis)
    return qml.sample(wires=range(4)), qml.counts(wires=range(4))

rotations = [shadow.random_signed_permutation(8) for _ in range(max(32, args.snapshots))]
compiled = [matchgate.compile_gaussian_givens(rotation) for rotation in rotations]
schedule = compiled[0][0]
angles = -np.stack([entry[1] for entry in compiled])
paulis = np.stack([entry[2] for entry in compiled])
samples_batch, counts_batch = shadow_circuit(angles, paulis)
```

Prepare Compile · Run · Retrieve Propagate on Slurm

0 s · Recorded Application Clock

4,096 Programs · 4,194,304 Shots · 128 Walkers per Trial

IQM Mock Target · Ideal DDSIM · Time-Scaled Replay

Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

Classical Preparation

PENNYLANE

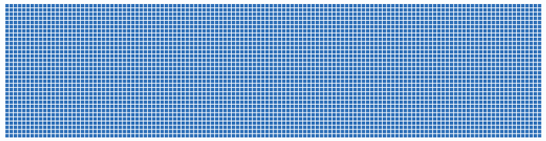
Generate random measurement bases 0–0.04 s
`rotations = [shadow.random_signed_permutation(8) for _ in range(max(32, args.snapshots))]`

Compile Gaussian rotations 0.04–2.88 s
`compiled = [matchgate.compile_gaussian_givens(rotation) for rotation in rotations]`

Capture Check: Majorana Convention 2.88–23.45 s
`for index, rotation in enumerate(rotations):`

Construct and preprocess the PennyLane broadcast 23.45–24.99 s
`device = qml.device(device_id, wires=4)`

Convert each tape to OpenQASM 24.99–26.4 s
`program = original_convert(tape)`



4,096 / 4,096 Programs Prepared

Shadow Circuit and Batch

```python
def apply_pauli_layer(pauli_vector) -> None:
    pauli_vector = np.asarray(pauli_vector)
    num_modes = pauli_vector.shape[-1] // 2
    for p in range(num_modes):
        qp.RX(np.pi * (pauli_vector[..., p] % 2), wires=p)
        qp.RZ(np.pi * (pauli_vector[..., p + num_modes] % 2), wires=p)

@qml.set_shots(shots=args.shots)
@qml.qnode(device)
def shadow_circuit(thetas: NDArray, paulis: NDArray) -> tuple:
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0, 1])
    hydrogen_trial()
    apply_gaussian_givens(thetas, schedule)
    apply_pauli_layer(paulis)
    return qml.sample(wires=range(4)), qml.counts(wires=range(4))

rotations = [shadow.random_signed_permutation(8) for _ in range(max(32, args.snapshots))]
compiled = [matchgate.compile_gaussian_givens(rotation) for rotation in rotations]
schedule = compiled[0][0]
angles = -np.stack([entry[1] for entry in compiled])
paulis = np.stack([entry[2] for entry in compiled])
samples_batch, counts_batch = shadow_circuit(angles, paulis)
```

Prepare Compile · Run · Retrieve Propagate on Slurm

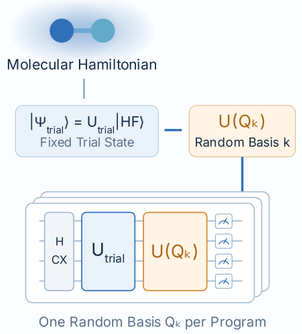
26.4 s · Recorded Application Clock

4,096 Programs · 4,194,304 Shots · 128 Walkers per Trial

IQM Mock Target · Ideal DDSIM · Time-Scaled Replay

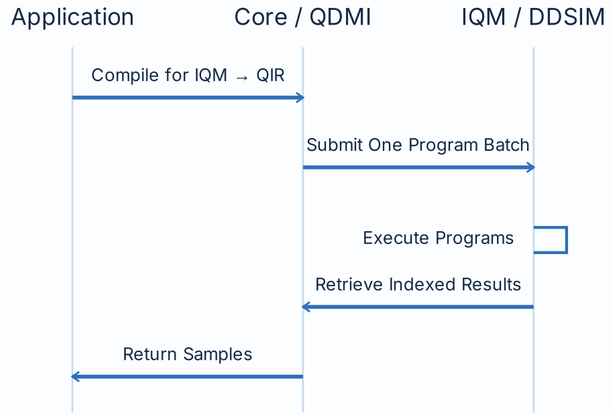
Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

Classical Preparation

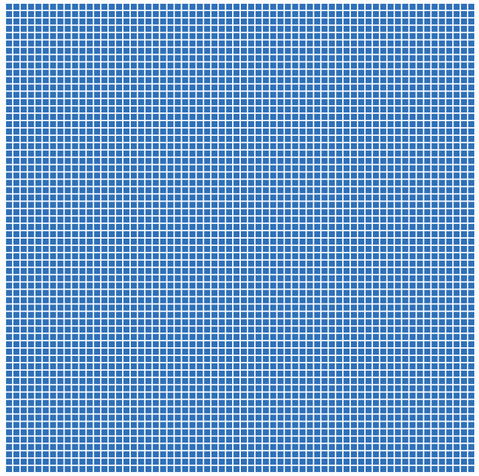


PENNYLANE

Quantum Shadow Collection



One QDMI Job · 4,096 Programs



4,096 Compiled · 4,096 Completed · 4,096 Retrieved

MUNICH QUANTUM TOOLKIT Compiler Collection

QDMI

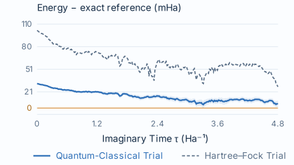
Classical Monte Carlo

Propagate → Overlap → Reweight → Reduce



$$\widehat{E}(\tau) = \frac{\sum_w W_w(\tau)\,\mathrm{Re}\,E_L[\phi_w(\tau)]}{\sum_w W_w(\tau)}$$

Weighted Mean



MQSC Orchestrator

Prepare **Compile · Run · Retrieve** Propagate on Slurm

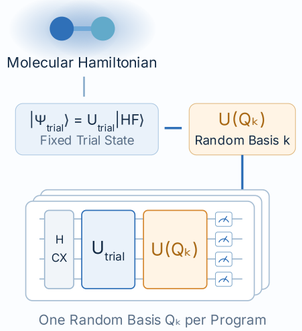
61.14 s · Recorded Application Clock

4,096 Programs · 4,194,304 Shots · 128 Walkers per Trial

IQM Mock Target · Ideal DDSIM · Time-Scaled Replay

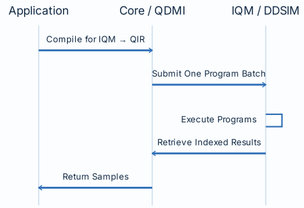
Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

Classical Preparation

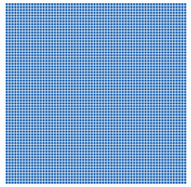


PENNYLANE

Quantum Shadow Collection



One QDMI Job · 4,096 Programs



4,096 Compiled	4,096 Completed	4,096 Retrieved

MUNICH QUANTUM TOOLKIT Compiler Collection

QDMI

Classical Monte Carlo

slurm workload manager

4 Compute Nodes · 4 Slurm Ranks

Quantum-Classical Trial

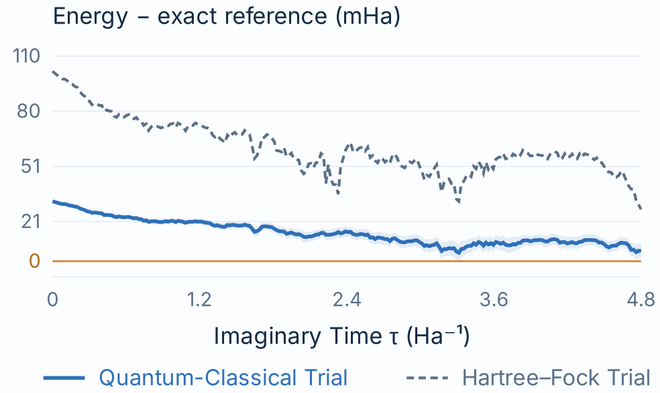


τ = 4.78 Ha^{-1} · filled area ∝ weight

Hartree–Fock Reference



τ = 4.78 Ha^{-1} · filled area ∝ weight



MQSC Orchestrator

Prepare Compile · Run · Retrieve **Propagate on Slurm**

141.89 s · Recorded Application Clock

4,096 Programs · 4,194,304 Shots · 128 Walkers per Trial

IQM Mock Target · Ideal DDSIM · Time-Scaled Replay

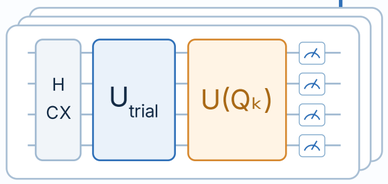
Quantum-Classical Auxiliary-Field Quantum Monte Carlo QC-AFQMC

Classical Preparation

Molecular Hamiltonian

$|\Psi_{trial}\rangle = U_{trial}|HF\rangle$
Fixed Trial State

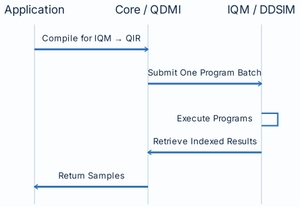
$U(Q_k)$
Random Basis k



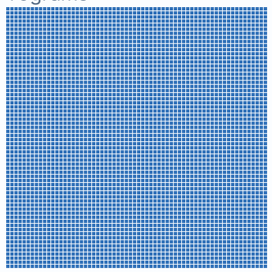
One Random Basis Q_k per Program

PENNYLANE

Quantum Shadow Collection



One QDMI Job · 4,096 Programs



4,096 Compiled 4,096 Completed 4,096 Retrieved

MUNICH QUANTUM TOOLKIT Compiler Collection

QDMI

Classical Monte Carlo

slurm workload manager

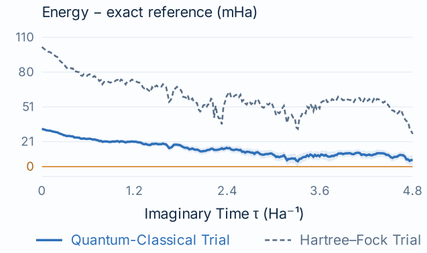
4 Compute Nodes · 4 Slurm Ranks

Quantum-Classical Trial



Hartree–Fock Reference





MQSC Orchestrator

Prepare Compile · Run · Retrieve **Propagate on Slurm**

141.89 s · Recorded Application Clock

4,096 Programs · 4,194,304 Shots · 128 Walkers per Trial

IQM Mock Target · Ideal DDSIM · Time-Scaled Replay

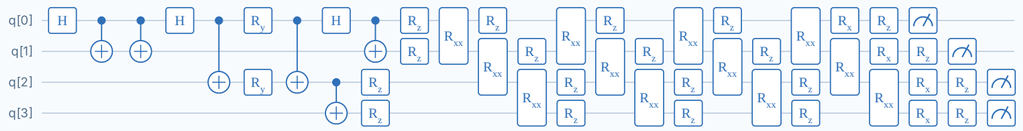
Targeting a Quantum Device

From Quantum Program to Executable

From Quantum Program to Executable

1 Input Program | 2 Optimization | 3 Device Specification | 4 Placement & Routing | 5 Native Synthesis | 6 Executable

1 · Input Program

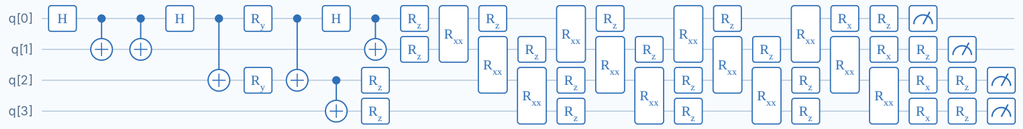


29 1Q
18 2Q
25 D

From Quantum Program to Executable

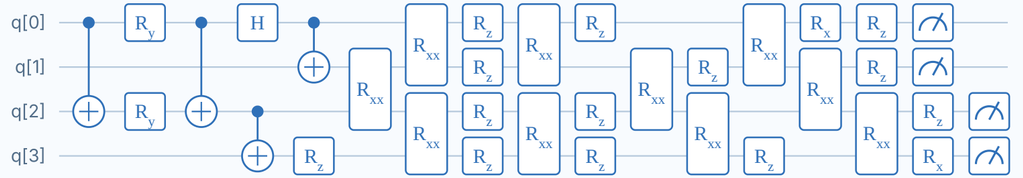
1 Input Program | 2 Optimization | 3 Device Specification | 4 Placement & Routing | 5 Native Synthesis | 6 Executable

1 · Input Program



29 1Q
18 2Q
25 D

2 · Optimization

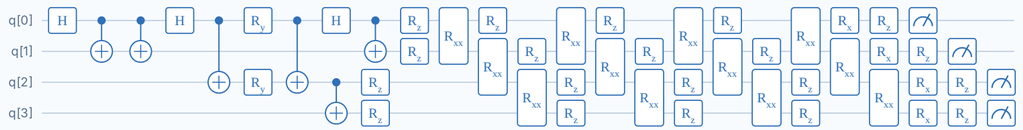


18 1Q
14 2Q
17 D

From Quantum Program to Executable

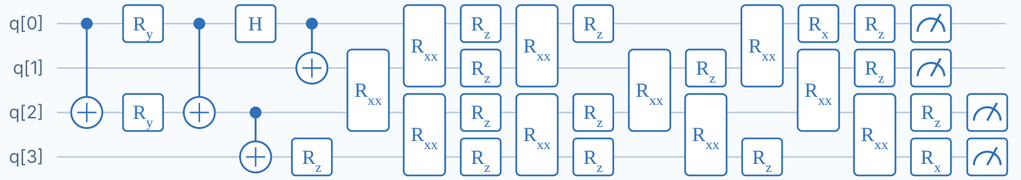
1 Input Program | 2 Optimization | 3 Device Specification | 4 Placement & Routing | 5 Native Synthesis | 6 Executable

1 · Input Program



29 1Q
18 2Q
25 D

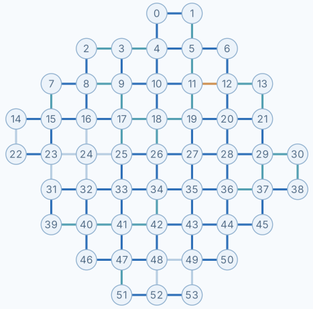
2 · Optimization



18 1Q
14 2Q
17 D

3 · Device Specification

Powered by QDMI



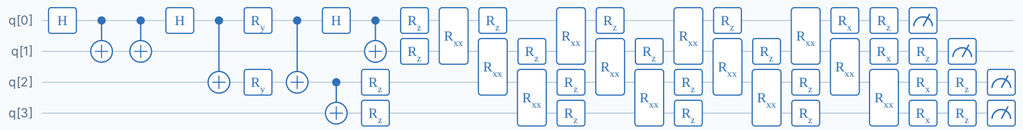
Emerald · 54 Qubits · QIR · IQM-JSON · R · CZ

iqm-finland/QDMI-on-IQM

From Quantum Program to Executable

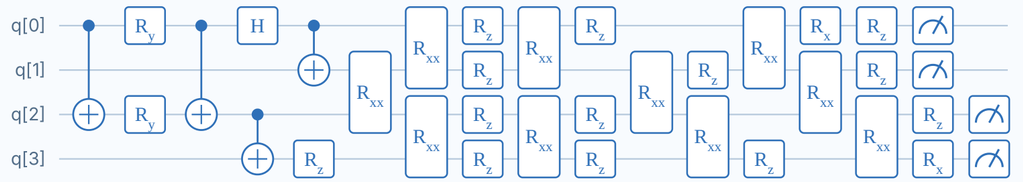
1 Input Program | 2 Optimization | 3 Device Specification | 4 Placement & Routing | 5 Native Synthesis | 6 Executable

1 · Input Program



29 1Q
18 2Q
25 D

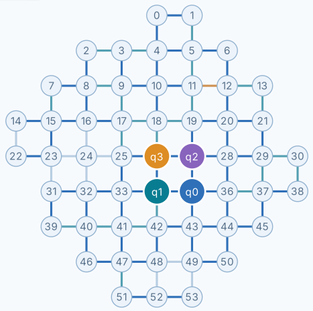
2 · Optimization



18 1Q
14 2Q
17 D

3 · Device Specification

Powered by QDMI



Emerald · 54 Qubits · QIR · IQM-JSON · R · CZ

iqm-finland/QDMI-on-IQM

4 · Placement & Routing

$q_3 \mapsto 26$

$q_2 \mapsto 27$

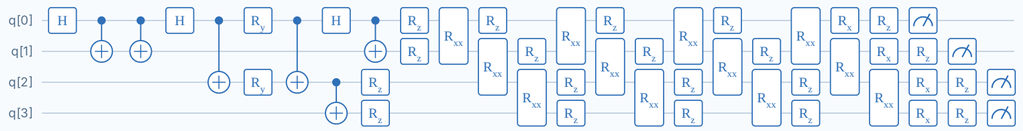
$q_1 \mapsto 34$

$q_0 \mapsto 35$

From Quantum Program to Executable

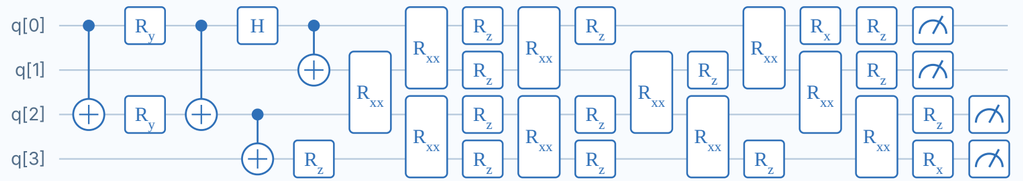
1 Input Program | 2 Optimization | 3 Device Specification | 4 Placement & Routing | 5 Native Synthesis | 6 Executable

1 · Input Program



29 1Q
18 2Q
25 D

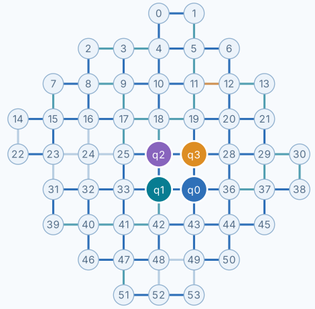
2 · Optimization



18 1Q
14 2Q
17 D

3 · Device Specification

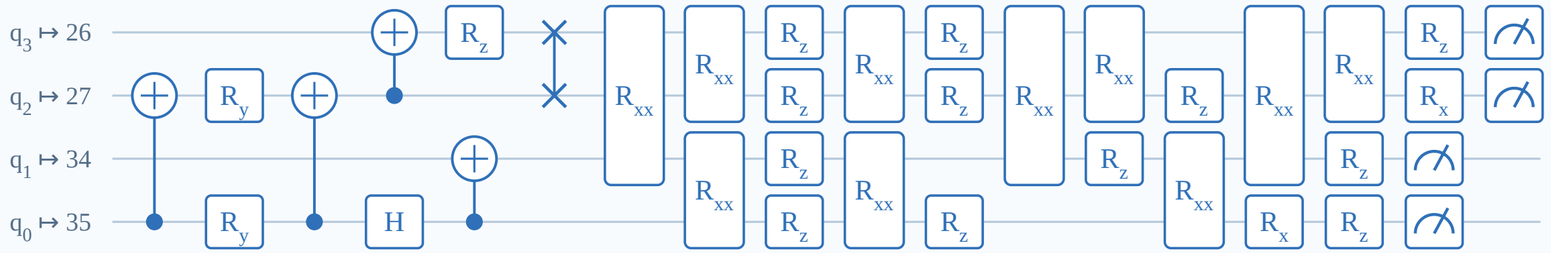
Powered by QDMI



Emerald · 54 Qubits · QIR · IQM-JSON · R · CZ

iqm-finland/QDMI-on-IQM

4 · Placement & Routing

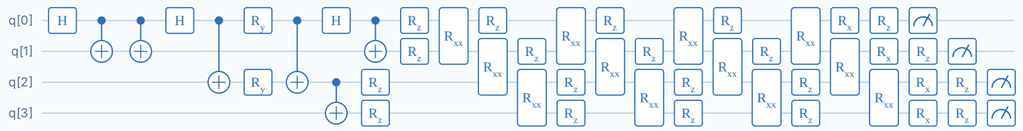


18 1Q
15 2Q
18 D

From Quantum Program to Executable

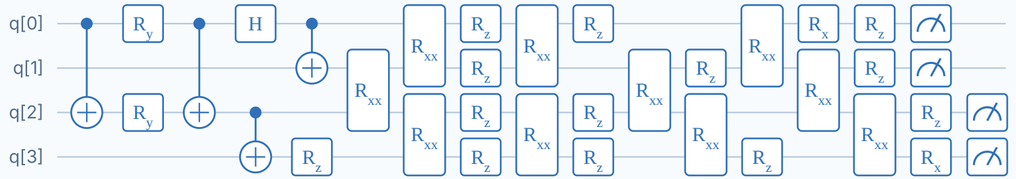
1 Input Program | 2 Optimization | 3 Device Specification | 4 Placement & Routing | 5 Native Synthesis | 6 Executable

1 · Input Program



29 1Q
18 2Q
25 D

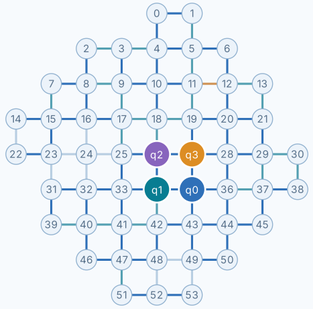
2 · Optimization



18 1Q
14 2Q
17 D

3 · Device Specification

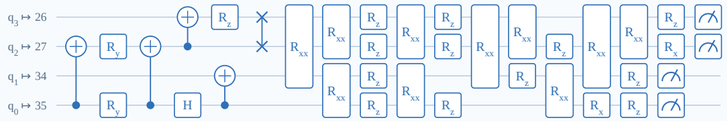
Powered by QDMI



Emerald · 54 Qubits · QIR · IQM-JSON · R · CZ

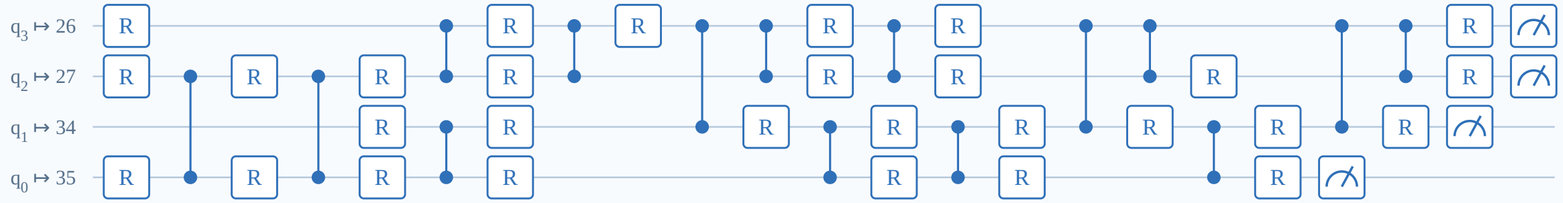
iqm-finland/QDMI-on-IQM

4 · Placement & Routing



18 1Q
15 2Q
18 D

5 · Native Synthesis

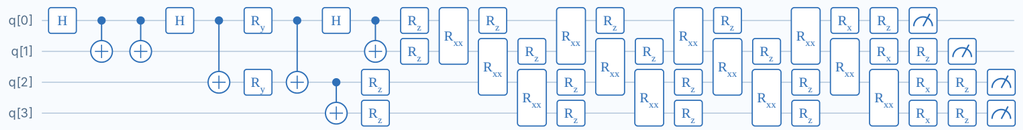


29 1Q
15 2Q
23 D

From Quantum Program to Executable

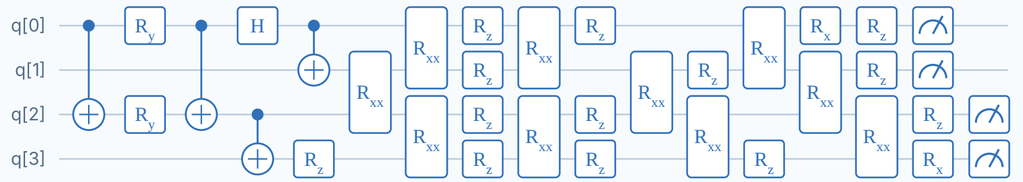
1 Input Program | 2 Optimization | 3 Device Specification | 4 Placement & Routing | 5 Native Synthesis | 6 Executable

1 · Input Program



29 1Q
18 2Q
25 D

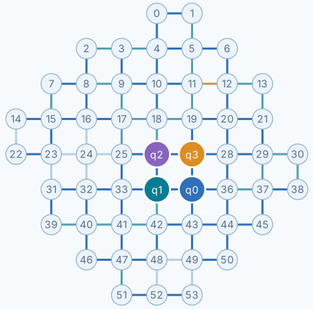
2 · Optimization



18 1Q
14 2Q
17 D

3 · Device Specification

Powered by QDMI



Emerald · 54 Qubits · QIR · IQM-JSON · R · CZ

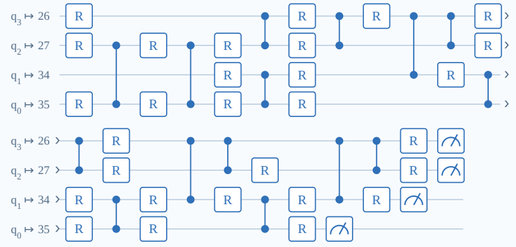
iqm-finland/QDMI-on-IQM

4 · Placement & Routing



18 1Q
15 2Q
18 D

5 · Native Synthesis



29 1Q
15 2Q
23 D

6 · QIR 2.1 Base Profile

```
define i64 @main() #0 {
  call void @__quantum__rt__initialize(ptr null)
  br label %1

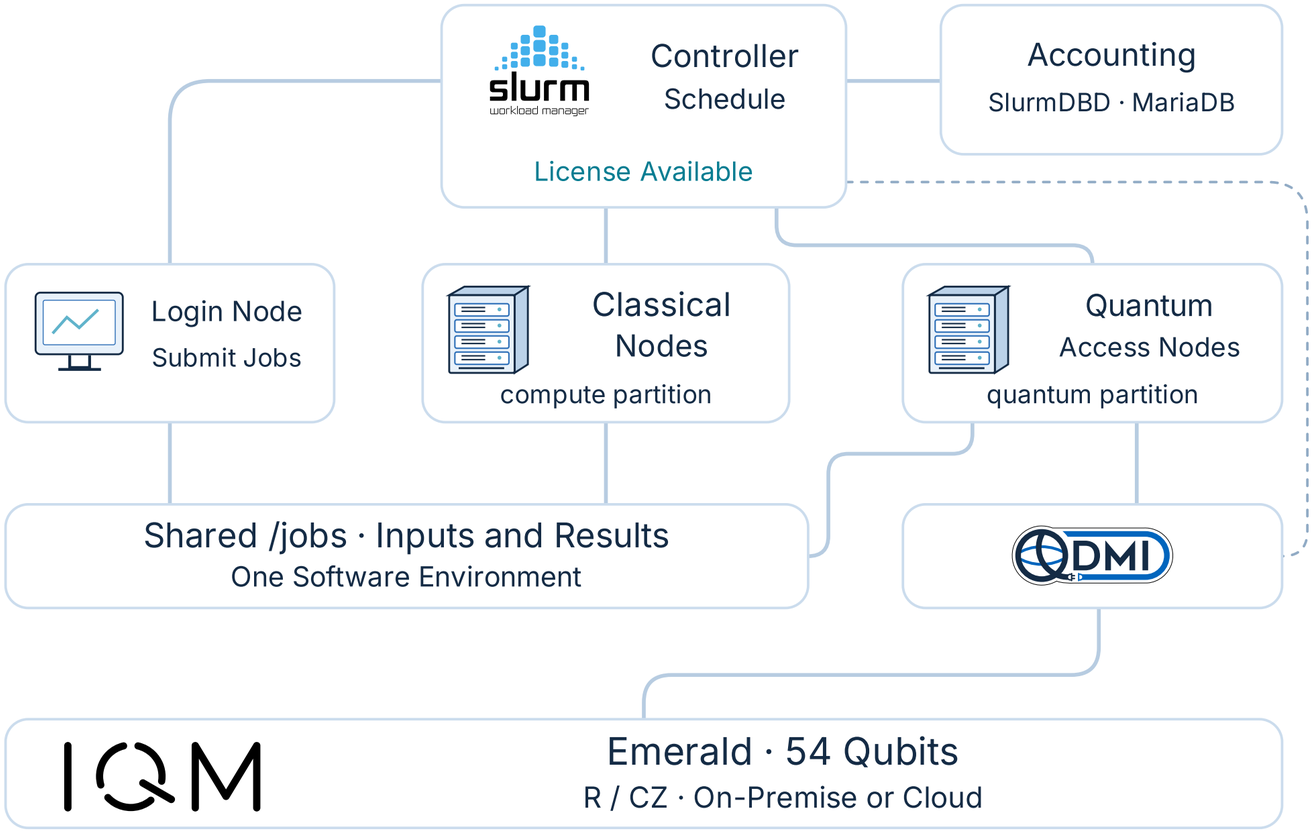
1:                                                ; preds = %0
  call void @__quantum__qis__prx__body(double f0x400921FB54442D18, double
f0x400921FB54442D18, ptr inttoptr (i64 35 to ptr))
  call void @__quantum__qis__prx__body(double f0x3FF921FB54442D18, double
f0x3FF921FB54442D18, ptr inttoptr (i64 27 to ptr))
  call void @__quantum__qis__cz__body(ptr inttoptr (i64 27 to ptr), ptr
inttoptr (i64 35 to ptr))
  call void @__quantum__qis__prx__body(double f0x3FD3333333333336, double
f0xBFF921FB54442D1C, ptr inttoptr (i64 35 to ptr))
  call void @__quantum__qis__prx__body(double f0x3FD3333333333334, double
f0x401F6A7A2955385E, ptr inttoptr (i64 27 to ptr))
```

⋮ Continued in the complete file

Explore HPCQC Integration with Slurm

From a Workflow to a Cluster

Explore HPCQC Integration with Slurm



EMERALD DEPLOYMENT

Two Dependent Slurm Jobs

Quantum Stage

Reserve one Emerald license and a quantum access node

Shared Results

Write classical shadows to /jobs

Classical Stage

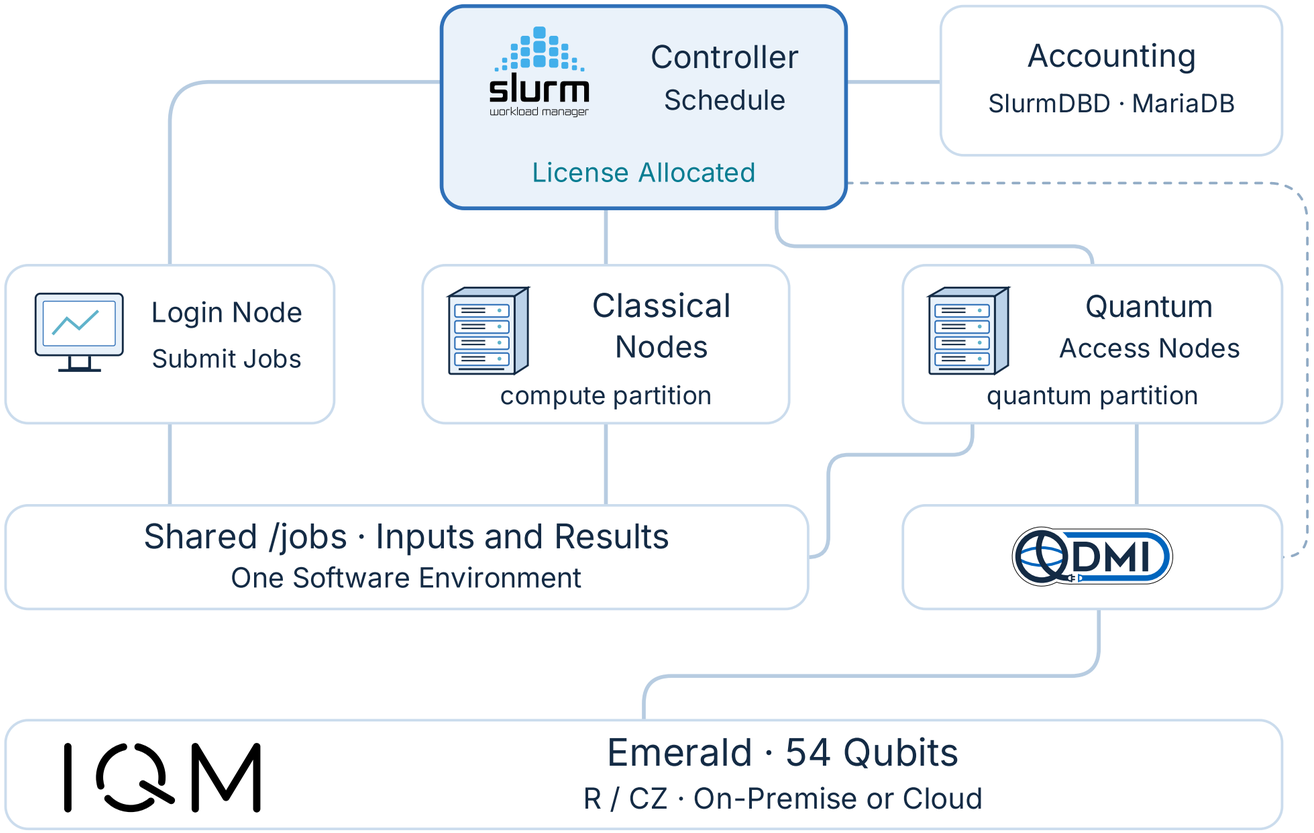
Start propagation after successful measurement

Device access fits the existing scheduler.

Try the Slurm Playground

MQT Core · examples/slurm

Explore HPCQC Integration with Slurm



EMERALD DEPLOYMENT · SCHEDULE

Submit the Quantum Stage

```
measurement=$(sbatch --parsable \
  --partition=quantum \
  --licenses=iqm.emerald:1 measure.sh)
```

1 Queue	2 Probe	3 Allocate

Device Availability

The monitor reserves licenses before probing; IDLE or BUSY permits allocation.

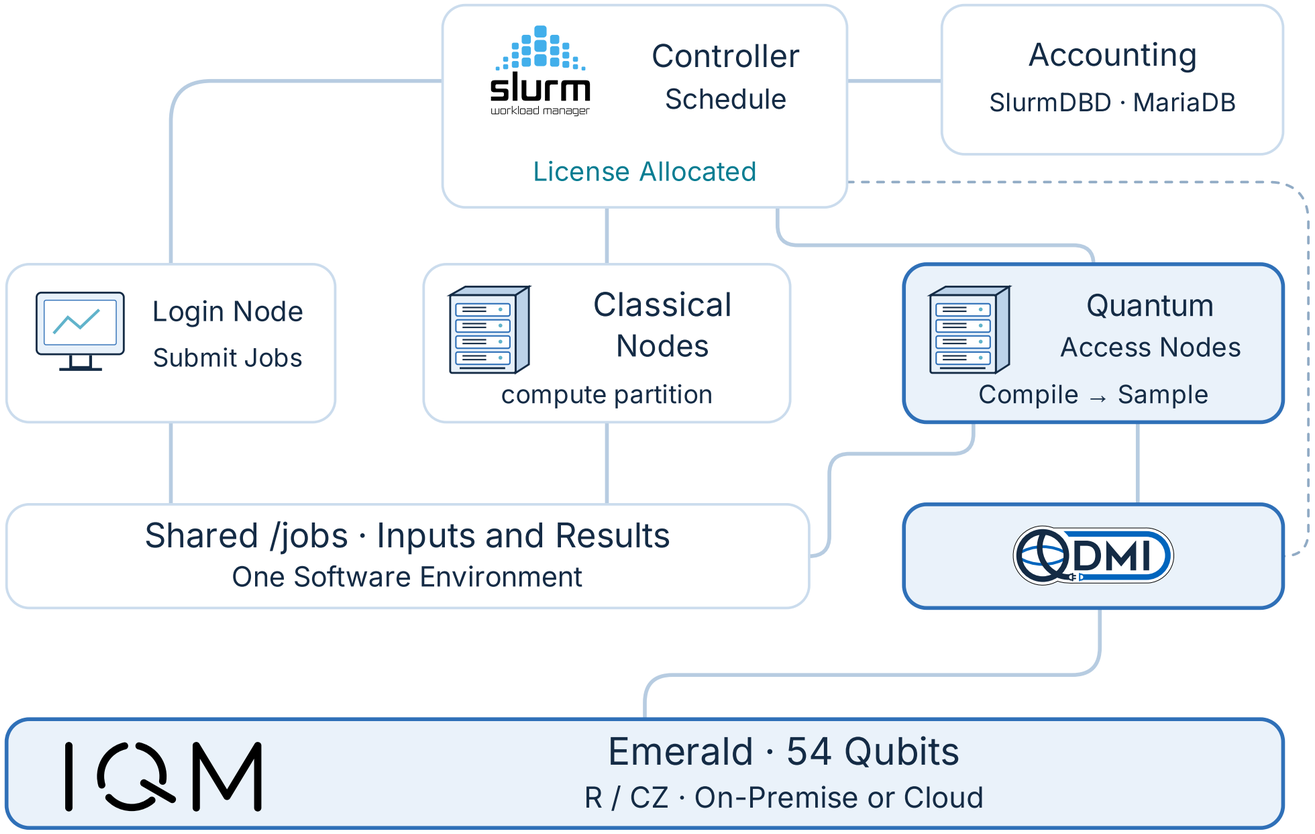
One license · One concurrent quantum allocation.

MUNICH QUANTUM TOOLKIT

Try the Slurm Playground

MQT Core · examples/slurm

Explore HPCQC Integration with Slurm



EMERALD DEPLOYMENT · EXECUTE

Compile and Sample on a QAN

Compile for IQM Emerald
Use the device topology and native operations

Submit through QDMI
Submit and track IQM Jobs

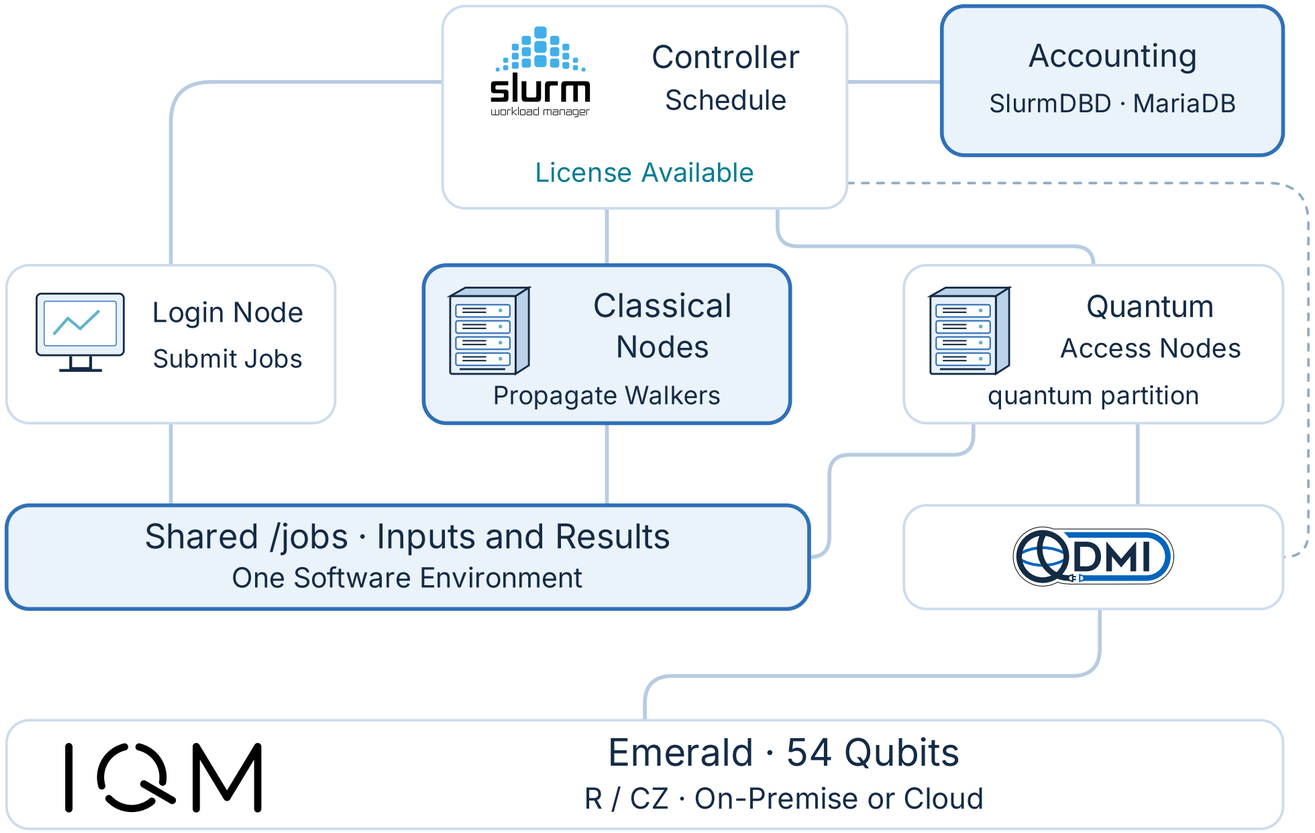
Retrieve Classical Shadows
Return samples to the application

The quantum stage holds one device license.

MUNICH QUANTUM TOOLKIT

Try the Slurm Playground
MQT Core · examples/slurm

Explore HPCQC Integration with Slurm



EMERALD DEPLOYMENT · CONTINUE

Propagate on Classical Nodes

```
sbatch --partition=compute \
  --dependency=afterok:"$measurement" \
  propagate.sh
```

Pass Results through /jobs

Measurement completes and returns its license.

Run Classical Propagation

4 Slurm ranks · 128 walkers per trial

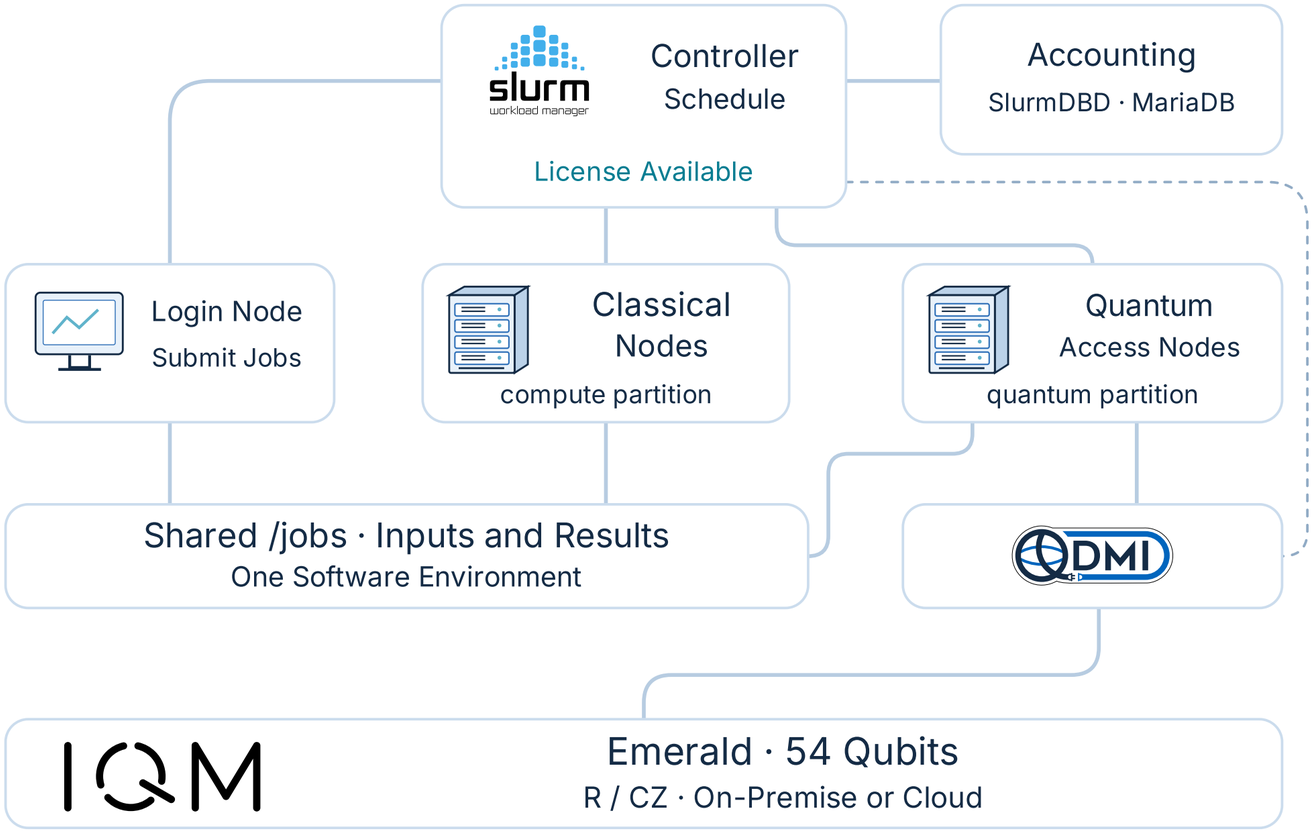
afterok requires success; no quantum license is held.

MUNICH QUANTUM TOOLKIT

Try the Slurm Playground

MQT Core · examples/slurm

Explore HPCQC Integration with Slurm



EMERALD DEPLOYMENT · SUMMARY

One Workflow, Two Allocations

```
measurement=$(sbatch --parsable \
  --partition=quantum \
  --licenses=iqm.emerald:1 measure.sh)

sbatch --partition=compute \
  --dependency=afterok:"$measurement" \
  propagate.sh
```

Scheduler-Enforced Access

One device license, returned after measurement

Independent Classical Resources

4 ranks · 128 walkers per trial · Shared /jobs

MUNICH QUANTUM TOOLKIT

Try the Slurm Playground

MQT Core · examples/slurm

From Interface to Integration

1. **A Shared Device Boundary**
   Keep frontend and HPC integration reusable.

2. **A Concrete IQM Implementation**
   Sessions, capabilities, jobs, and results
   across cloud and on-premise deployments.

3. **Practical Hybrid Workflows**
   Connect device-aware compilation,
   quantum execution, and classical computation.

Application ↔ slurm workload manager ↔ QDMI ↔ IQM

The Paper
Burgholzer, Walter, et al., 2026
arxiv.org/abs/2604.19869

The Implementation
Source · Installation · Examples
github.com/iqm-finland/QDMI-on-IQM

The Interface
Open Specification · Ecosystem
github.com/Munich-Quantum-Software-Stack/QDMI