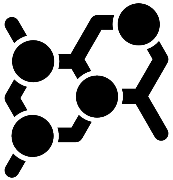
QOS
Quantum Operating System

Emmanouil (Manos) Giortamis, Francisco Romão,
Nathaniel Tornow, Pramod Bhatotia

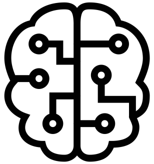
Technical University of Munich

USENIX OSDI 2025, Boston, USA

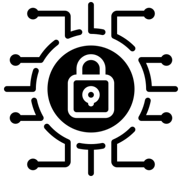
Towards a quantum computing era



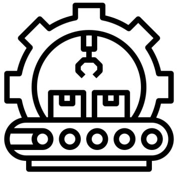
Chemistry



Machine learning



Cryptography



Material science

Promising applications

IBM Q™

Google AI Quantum

IONQ

aws Amazon Braket

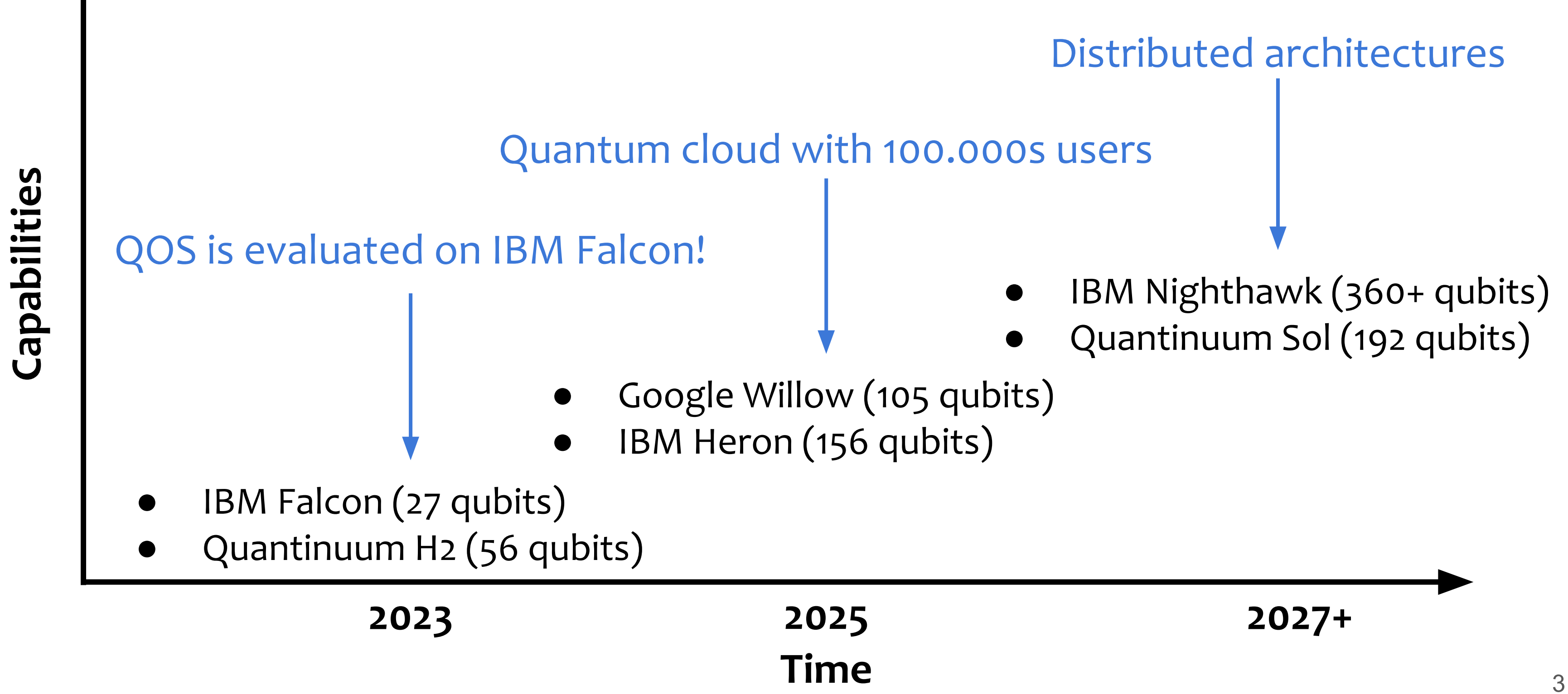
QUANTINUUM

Azure Quantum

Industry R&D and adoption

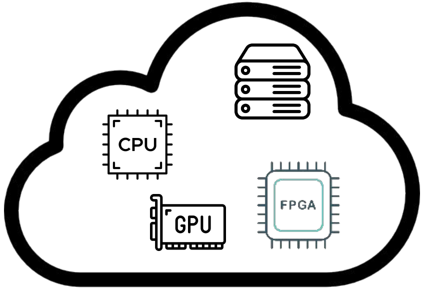
Quantum computers are a reality

There exists no OS support for quantum architectures!

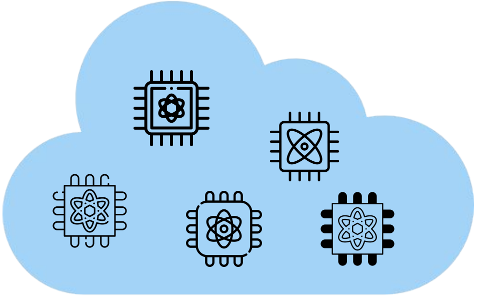


How to manage quantum computers?

Classical cloud	Quantum cloud



MESOS kubernetes



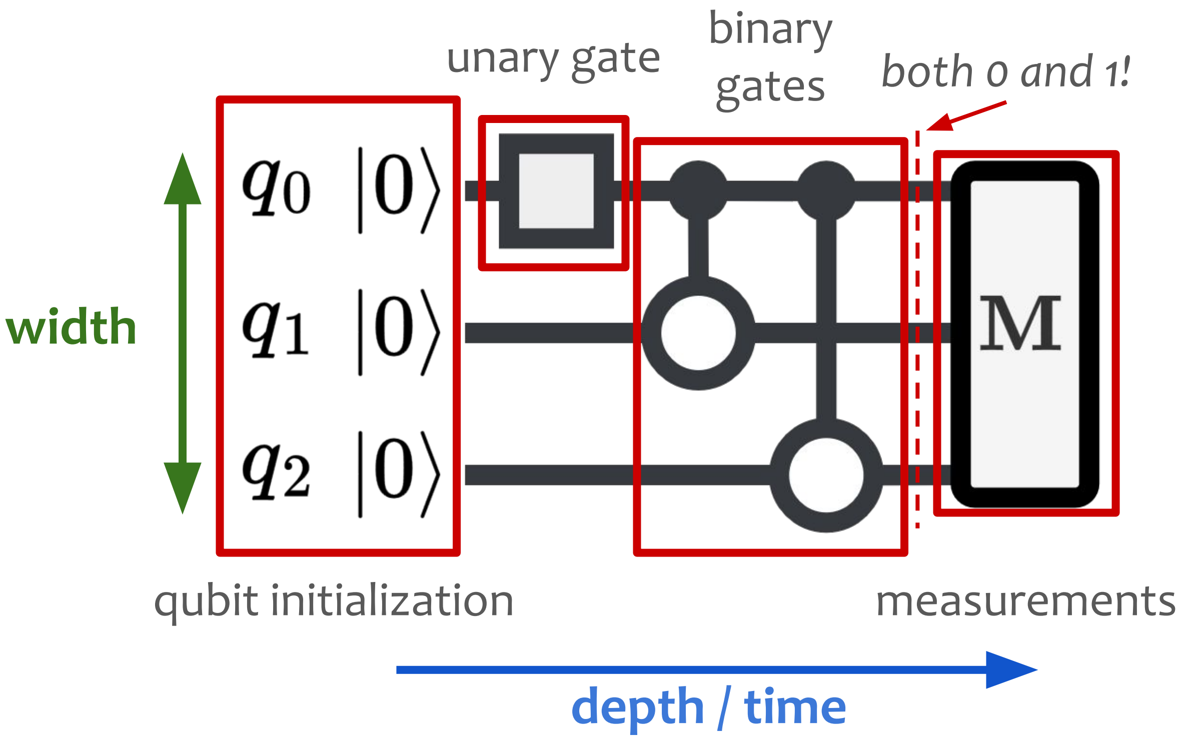
Can we use existing cloud OSes for quantum computers?

<u>Unfortunately, no!</u>

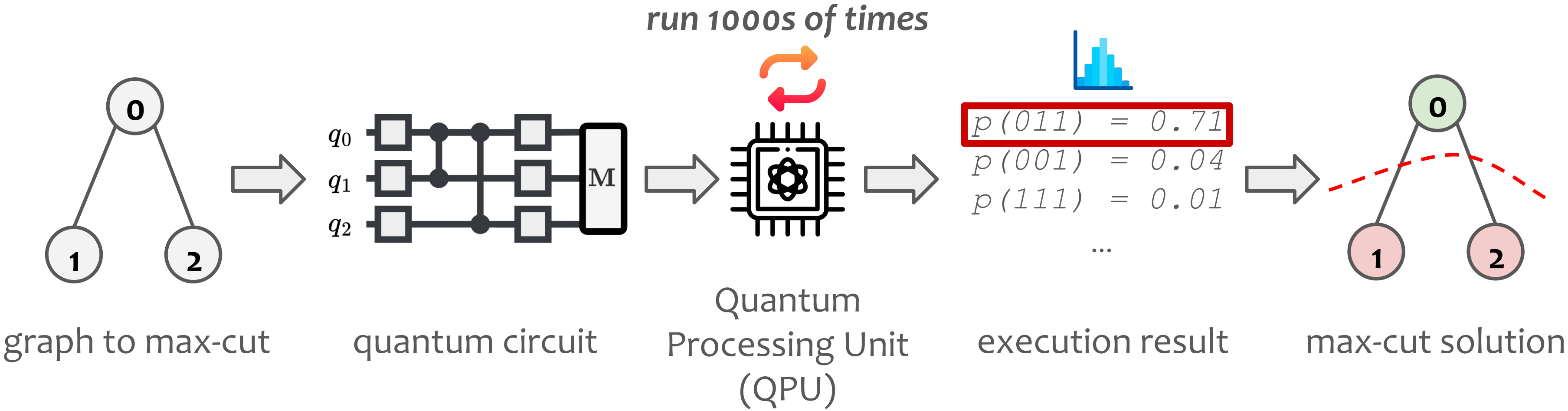
Quantum computers are fundamentally different compared to classical computers

Quantum 101: Programs are represented as circuits

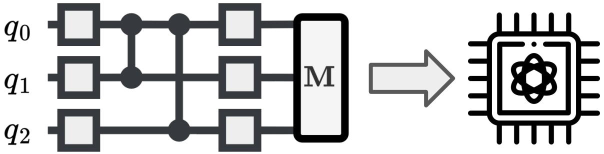
```
qreg q[3];

h q[0];
cx q[0], q[1];
cx q[0], q[2];

creg c[3];
measure q -> c;
```



An example: Max-cut on a graph



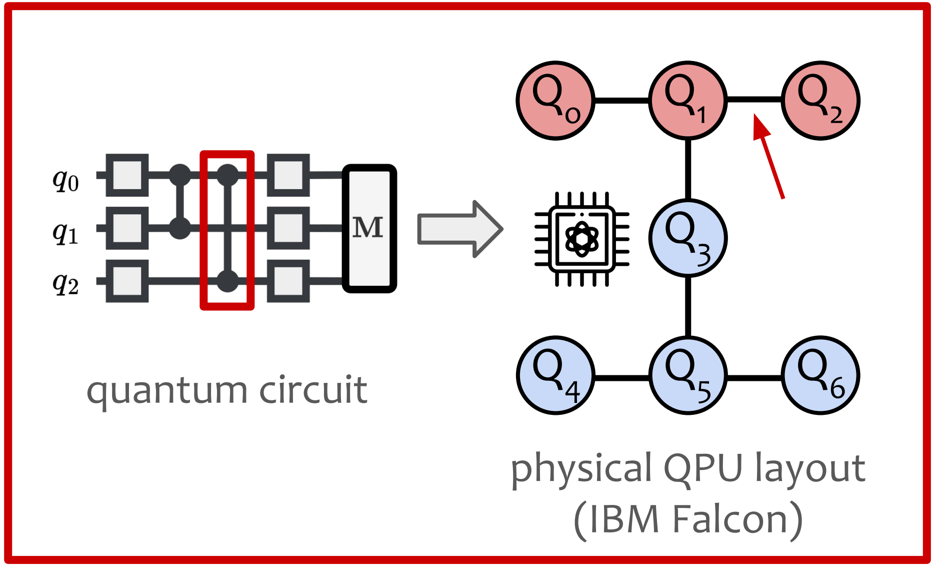
Quantum circuit execution



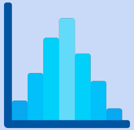
quantum circuit

QPU

Quantum circuit execution



#1 Fidelity



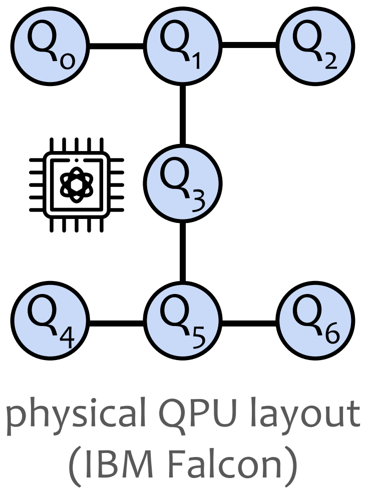
#2 Utilization

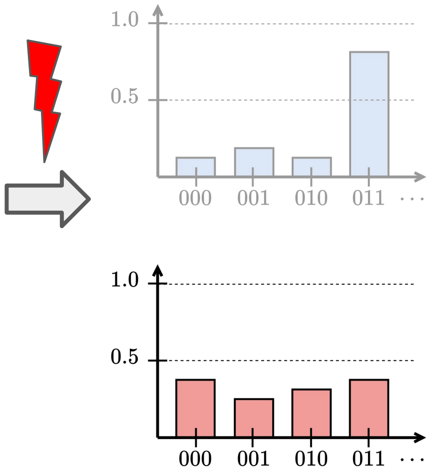
#3 Heterogeneity

#4 Load imbalance

Challenge #1: Fidelity

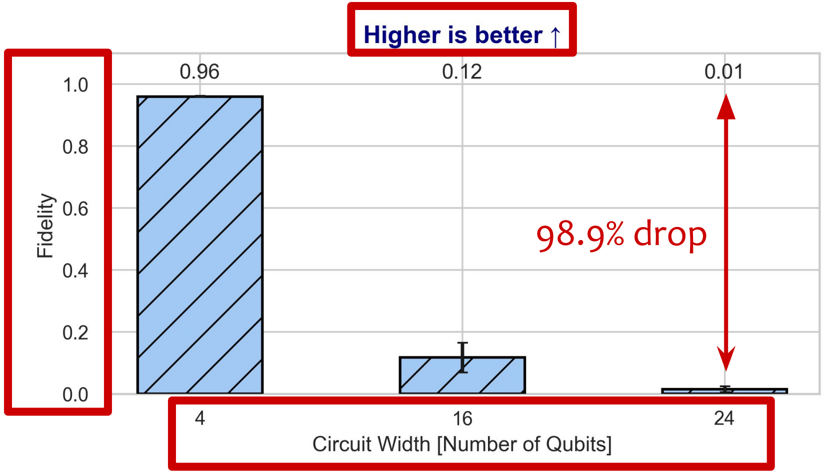
- QPUs are susceptible to noise errors
- Circuit operations add noise errors
- The execution results are noisy





Challenge #1: Fidelity

- QPUs are susceptible to noise errors
- Circuit operations add noise errors
- The execution results are noisy



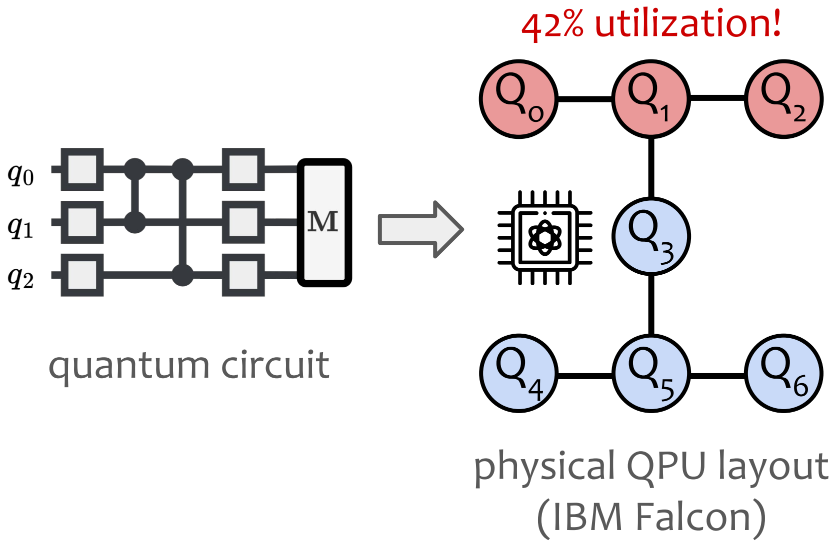
Executed on IBM Kolkata, a 27-qubit Falcon QPU

Problem: Large circuits suffer from low execution fidelity

Key idea: Optimize the circuits to be smaller in width and depth

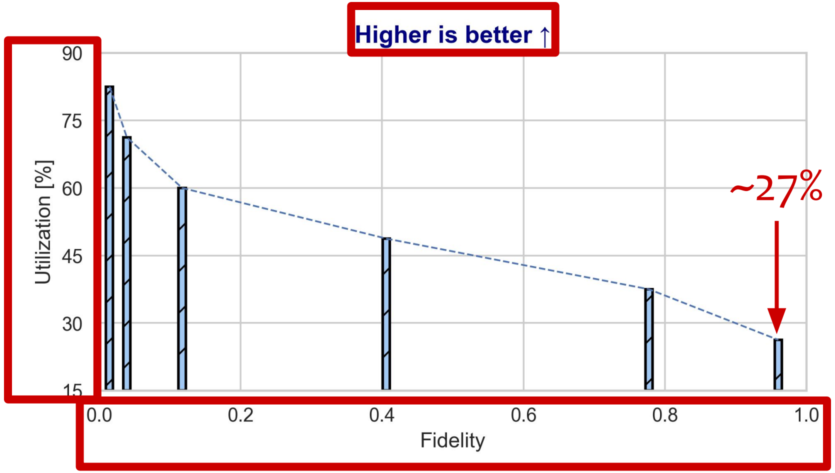
Challenge #2: Utilization

- Large circuits → low fidelity (#1)
- Small circuits → low utilization
- Circuits cannot be preempted



Challenge #2: Utilization

- Large circuits → low fidelity (#1)
- Small circuits → low utilization
- Circuits cannot be preempted



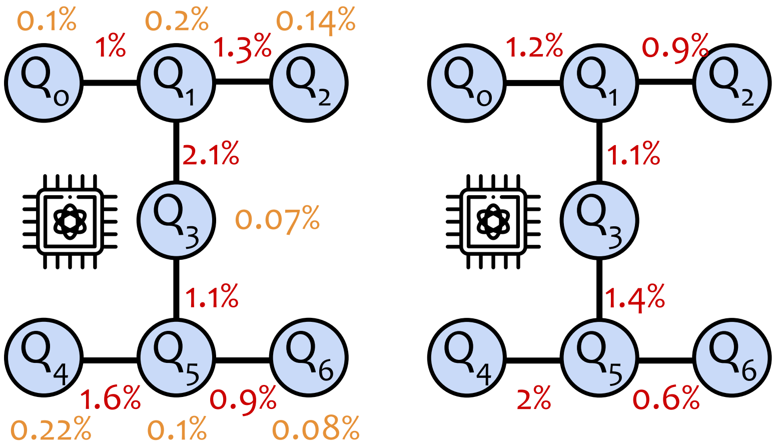
Executed on IBM Kolkata, a 27-qubit Falcon QPU

Problem: Solo circuit execution sacrifices either fidelity or QPU utilization

Key idea: Multi-program *small* circuits on a QPU to increase utilization

Challenge #3: Heterogeneity

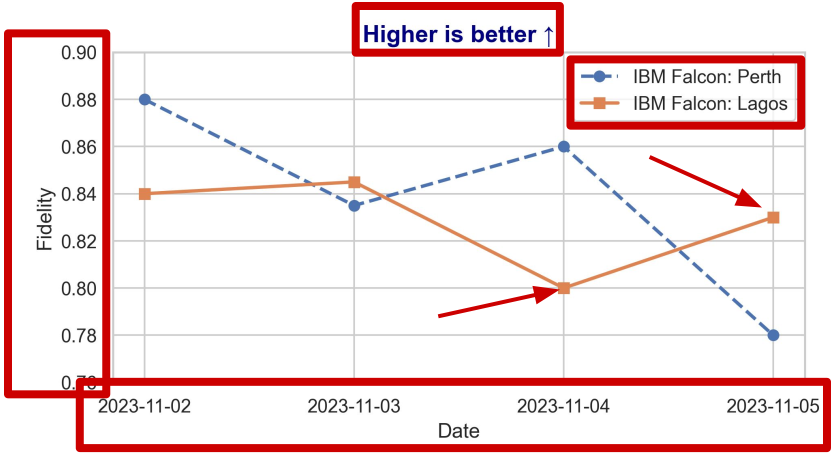
- QPU qubits have different error rates
- QPUs have different error rates
- QPU error rates change over time



IBM Falcon: Perth IBM Falcon: Lagos

Challenge #3: Heterogeneity

- QPU qubits have different error rates
- QPUs have different error rates
- QPU error rates change over time

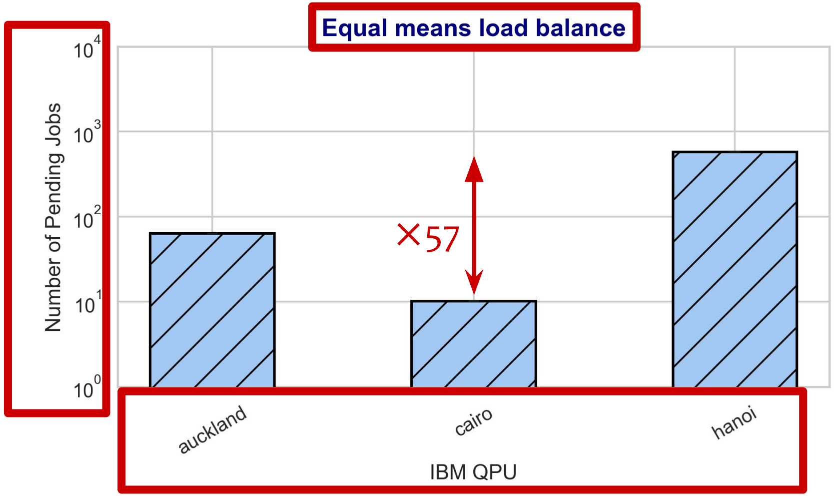


Problem: QPU fidelity varies across space and time

Kea idea: We can estimate the execution fidelity of a circuit a priori

Challenge #4: Load imbalance

- QPUs are vastly heterogeneous
- Users want high fidelity
- Providers want resource efficiency



Problem: Tradeoff between high fidelity and high resource efficiency

Key idea: Scheduling that optimizes both fidelity and resource efficiency

The need for a quantum operating system

QPUs suffer from low fidelity performance, low utilization, spatio-temporal heterogeneity, and vast load imbalance

How to design a **unified** operating system that manages heterogeneous QPUs with **high fidelity** and **high resource efficiency**?

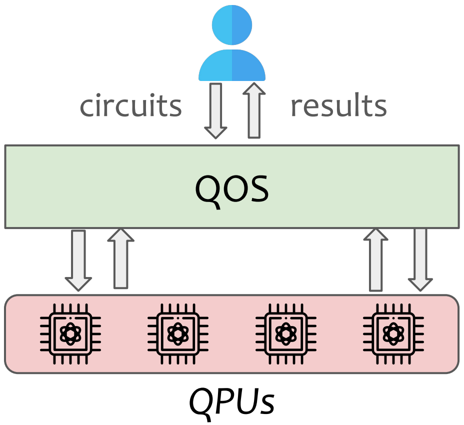
Our proposal: QOS

Quantum Operating System (QOS)

The *first* unified system stack for managing QPUs while mitigating their limitations

<u>**Core contributions:**</u>

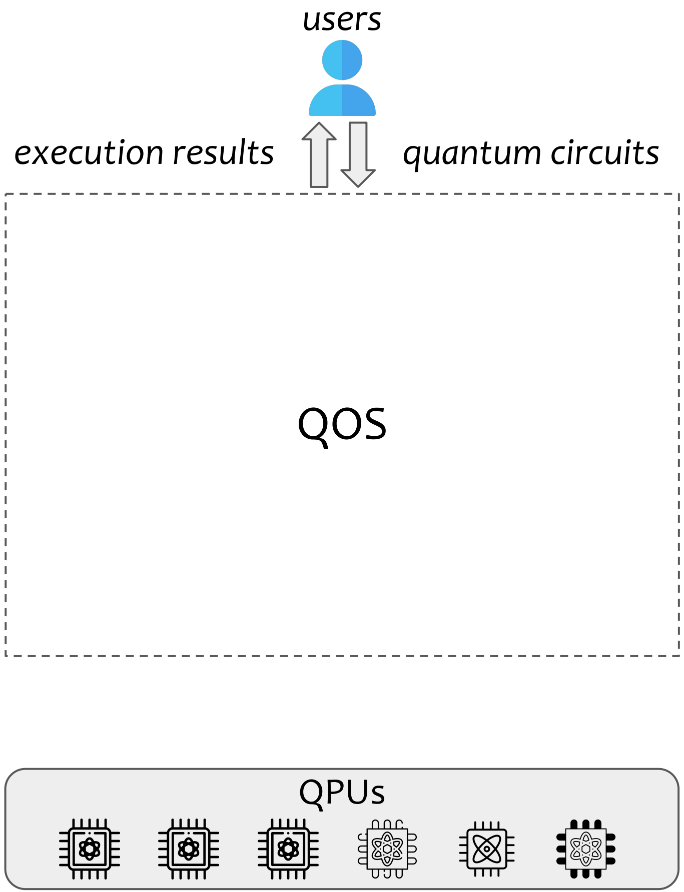
- High fidelity and high resource efficiency
- Systematic trade-off management
- Intra- and cross-stack optimizations



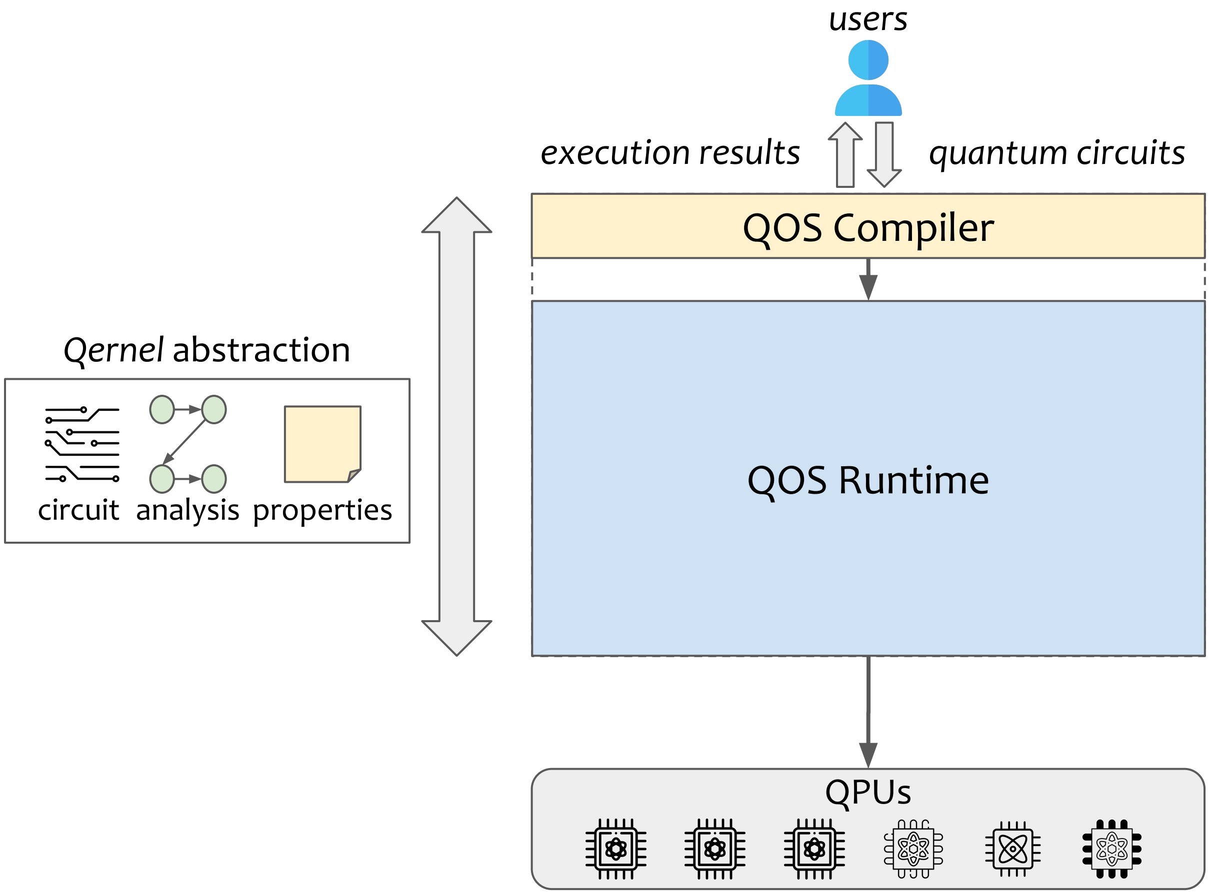
Outline

- ~~Introduction & motivation~~
- System design
  - System overview
  - System Components
- Evaluation

System overview



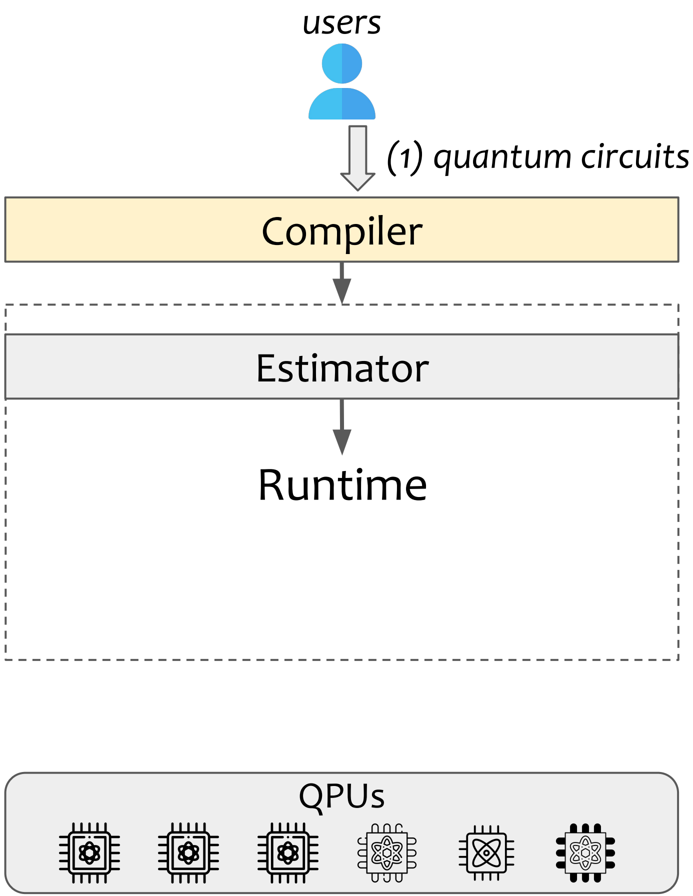
System overview



Optimizes circuits to increase execution fidelity

Multiplexes circuits across space and time to increase resource efficiency

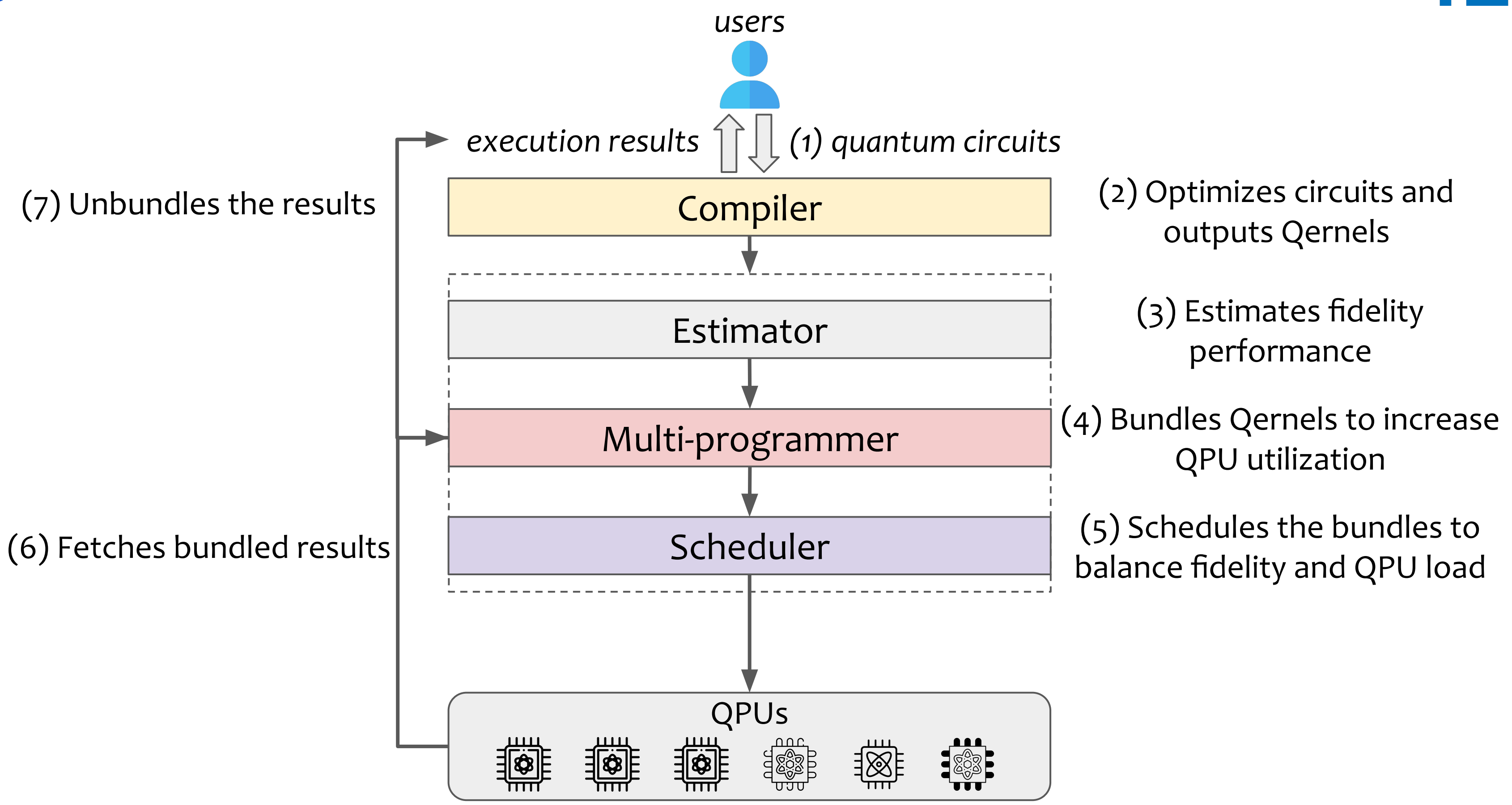
System overview and workflow



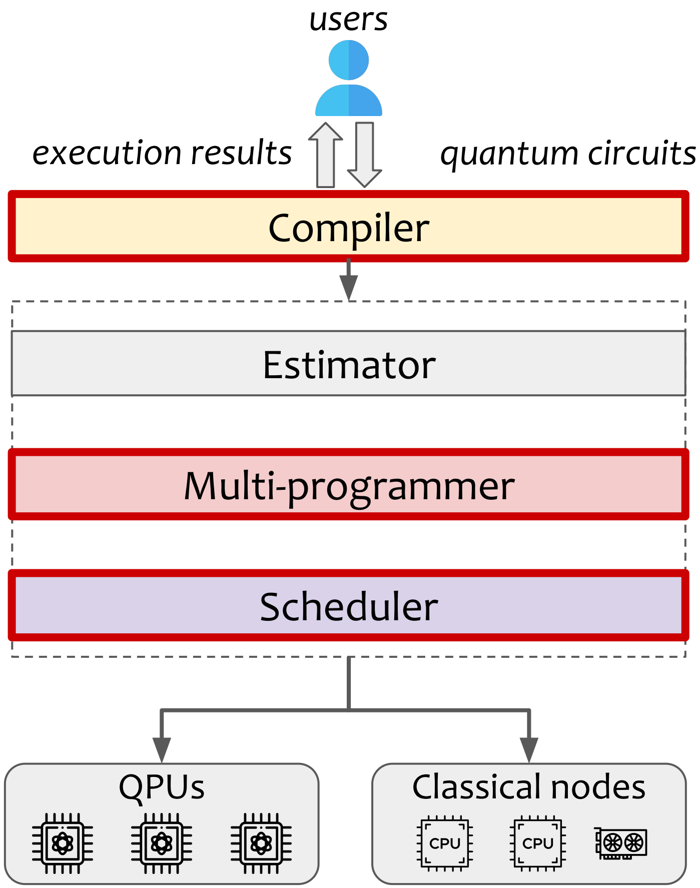
(2) Optimizes circuits and outputs Qernels

(3) Estimates fidelity performance

System overview and workflow

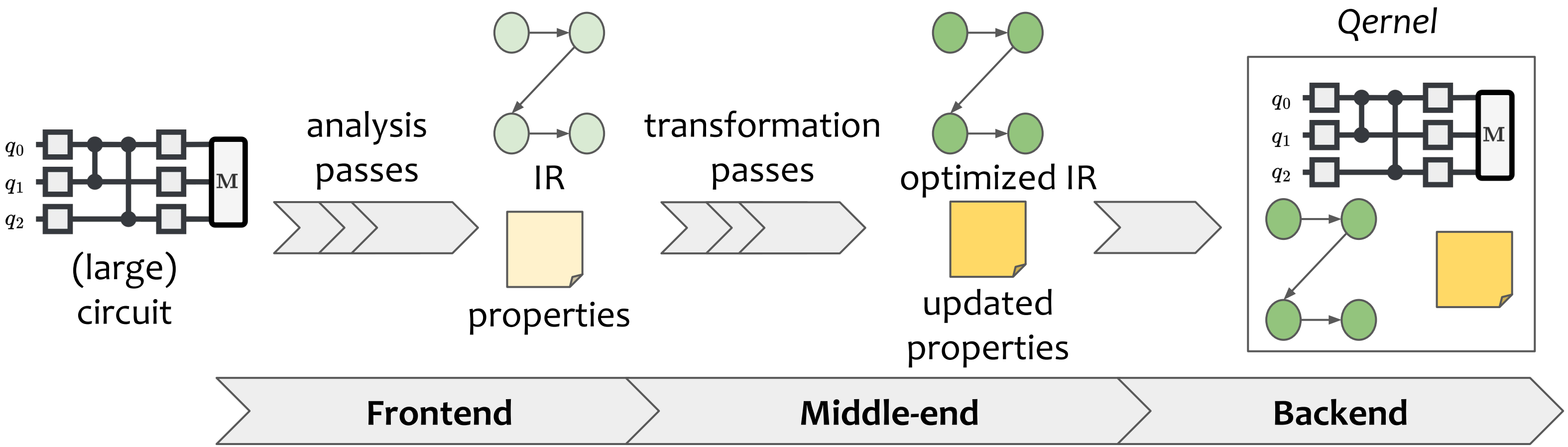


System overview



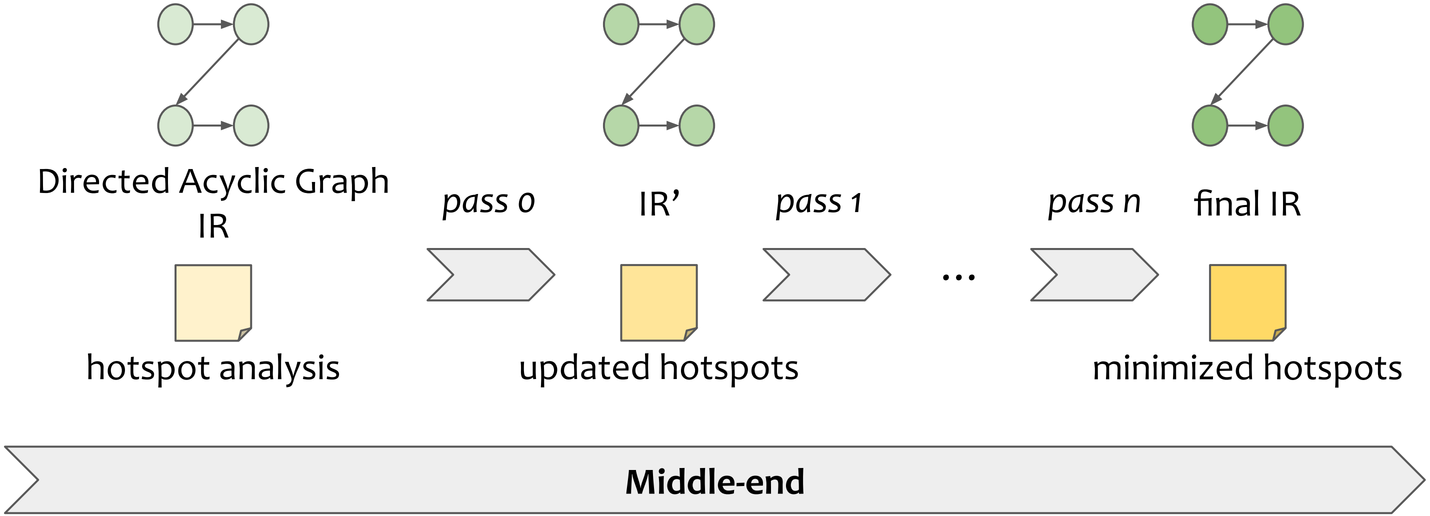
Compiler

- How to increase the execution fidelity on noisy QPUs?
  - Compose complementary optimization techniques in a compilation pipeline
- Challenge: How to compose techniques that work on different abstractions?
- Approach: LLVM-like design with IR, analysis, and transformation passes

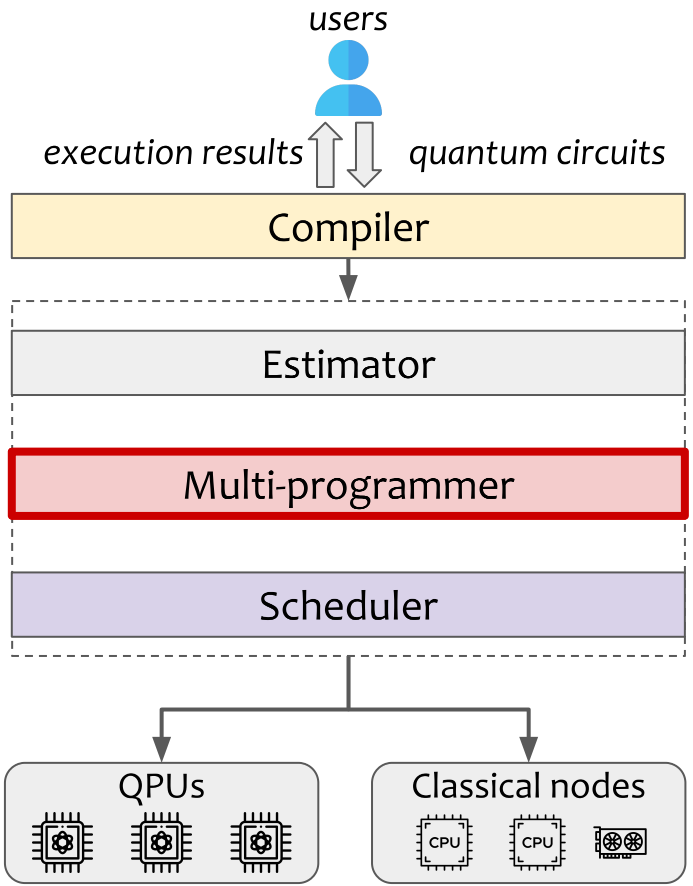


Compiler: Middle-end

- Challenge: Which optimization techniques to apply and in what order?
- Approach: Identify *hotspot* operations that impact fidelity the most
  - Greedy algorithm that applies techniques in order of hotspot elimination efficiency

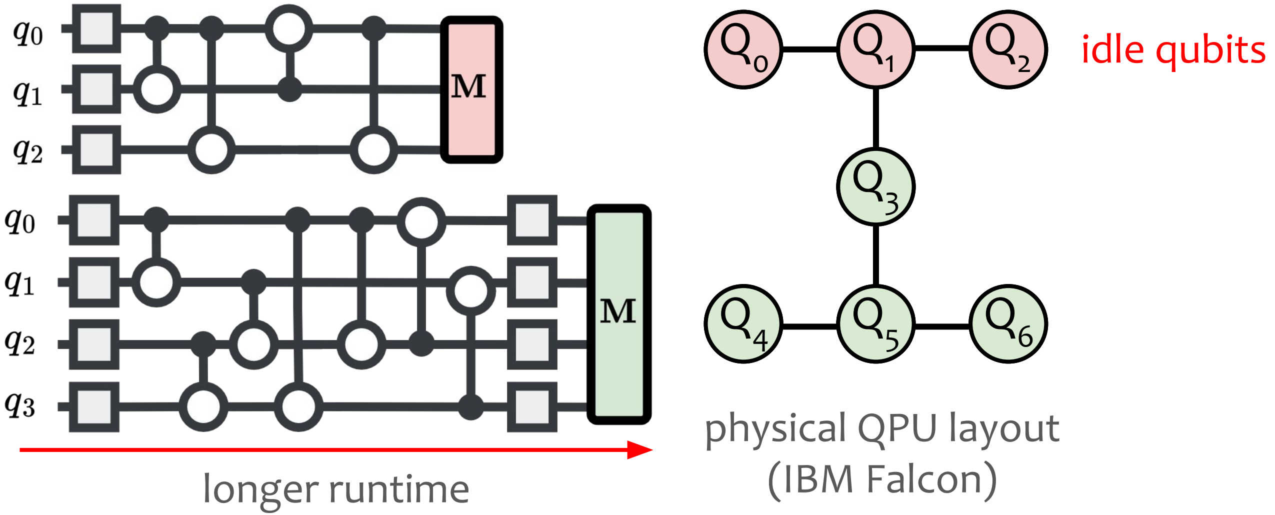


System overview



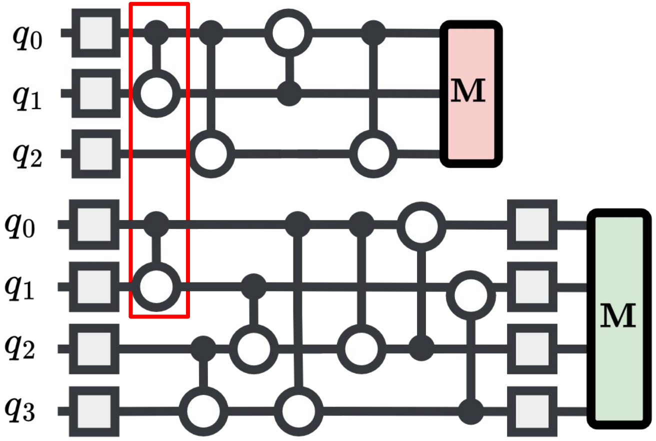
Multi-programmer

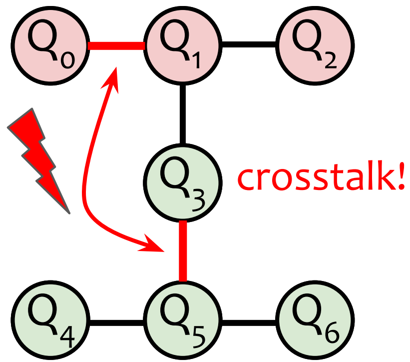
- How to increase QPU utilization?
  - Bundle multiple Qernels on the same QPU
- Challenge #1: Temporal utilization
  - Unequal Qernel runtimes reduce temporal QPU utilization



Multi-programmer

- Challenge #2: Destructive interference due to co-location
  - *Crosstalk* noise across Qernels lowers fidelity
  - Crosstalk depends on the physical distance

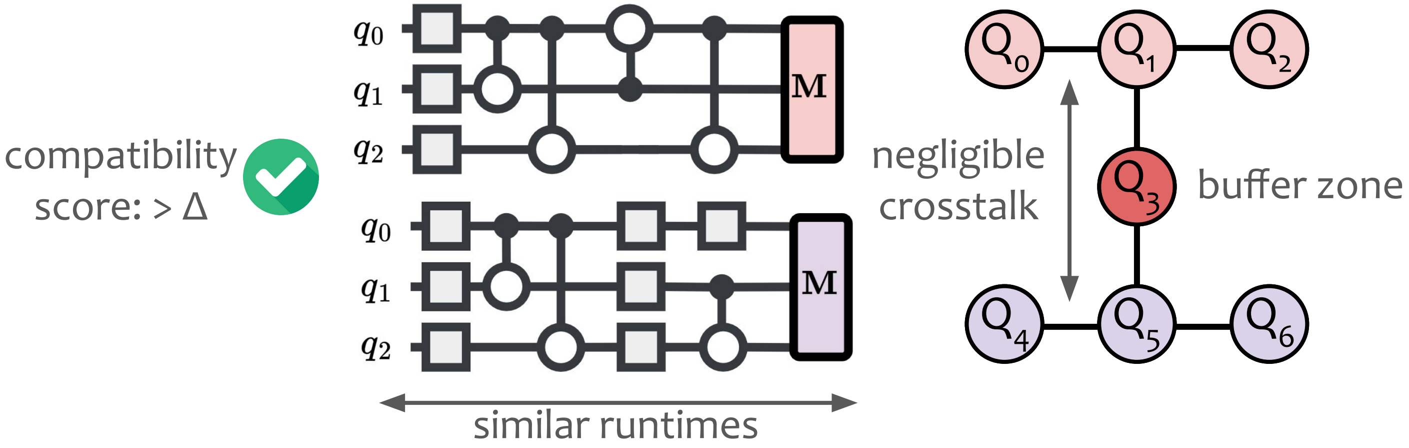




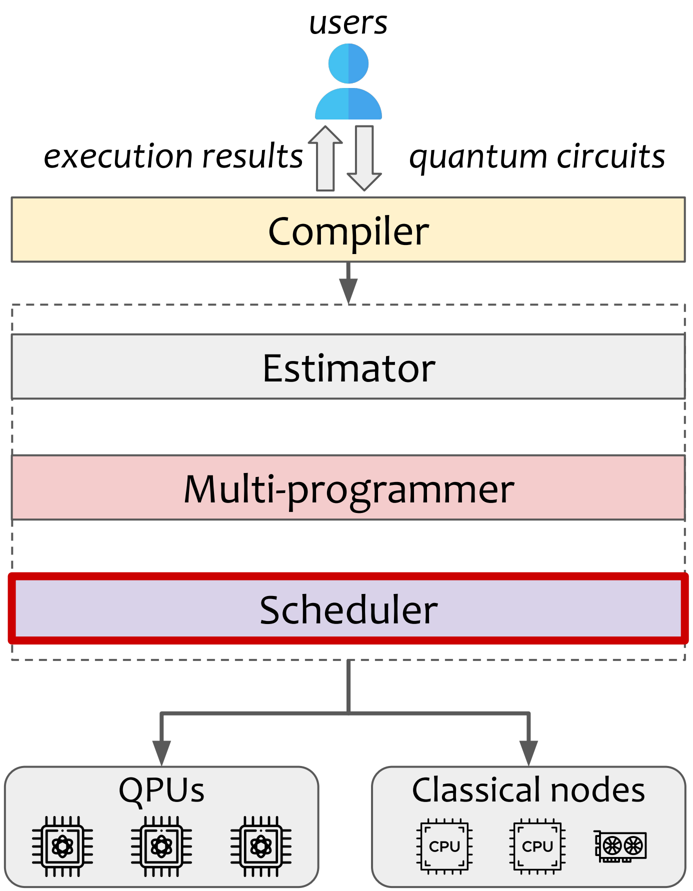
physical QPU layout
(IBM Falcon)

Multi-programmer

- Approach: Compatibility score & buffer zone
    - *Effective* utilization: Spatial + temporal utilization
    - Compatibility score: Quantifies crosstalk chance
    - Buffer zone: Unallocated qubits between Qernels

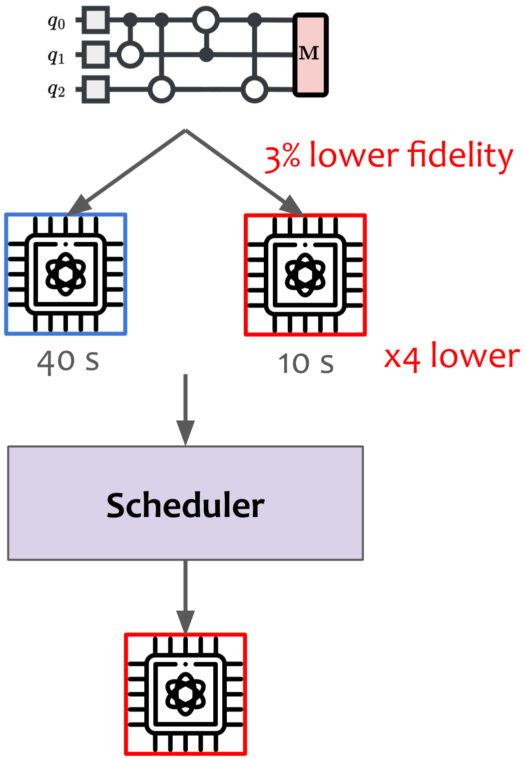


System overview



Scheduler

- How to balance QPU load?
  - Evenly distribute Qernels across QPUs
- Challenge: Fidelity-load balance tradeoff
  - Reason: QPU heterogeneity
- Approach: Multi-objective optimization
  - Criteria: fidelity or load balance



Outline

- ~~Introduction & Motivation~~
- ~~System Design~~
    - ~~System Overview~~
    - ~~System Components~~
- Evaluation

Evaluation

- Implemented in Python based on the Qiskit quantum SDK

- **Main research questions (RQs):**
  - **RQ1:** What is the compiler's impact on fidelity?
  - **RQ2:** What is the multi-programmer's impact on fidelity with increasing utilization?

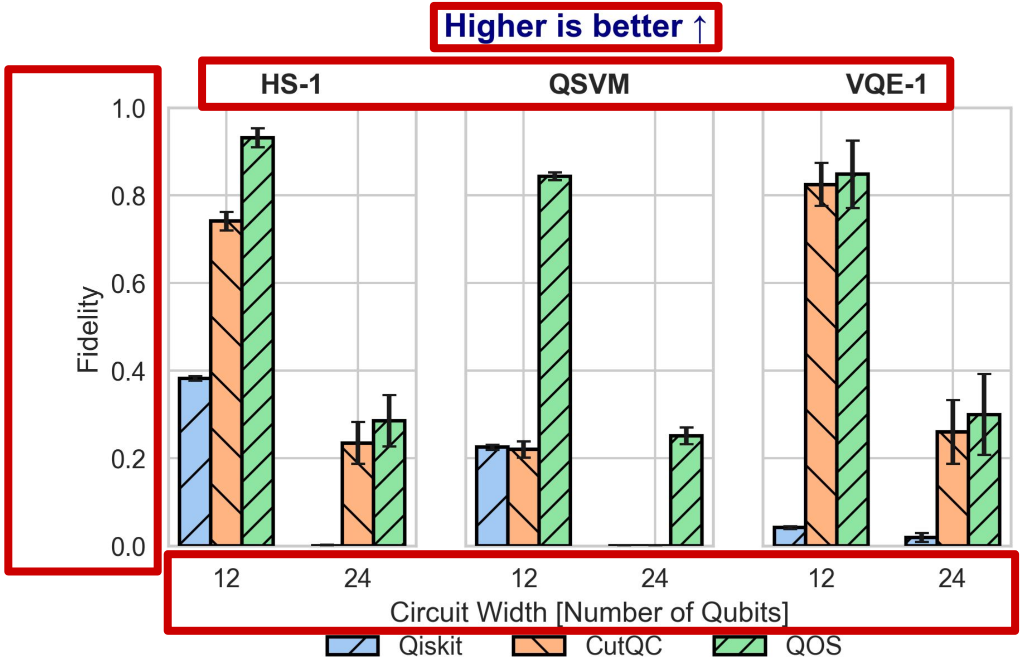
Many more results in the paper!

Evaluation methodology

- **Setup:** IBM 27-qubit Falcon QPUs, 64-core AMD EPYC 7713P
- **Benchmarks:** State-of-the-art quantum applications
- **Baselines**
  - IBM's Qiskit compiler, CutQC [1], A Case for Multi-programming [2]
- **Metrics**
  - **Fidelity** (Higher is better)
  - **Utilization** (Higher is better)

[1] CutQC: using small Quantum computers for large Quantum circuit evaluations, ASPLOS '21
[2] A Case for Multi-Programming Quantum Computers, MICRO '19

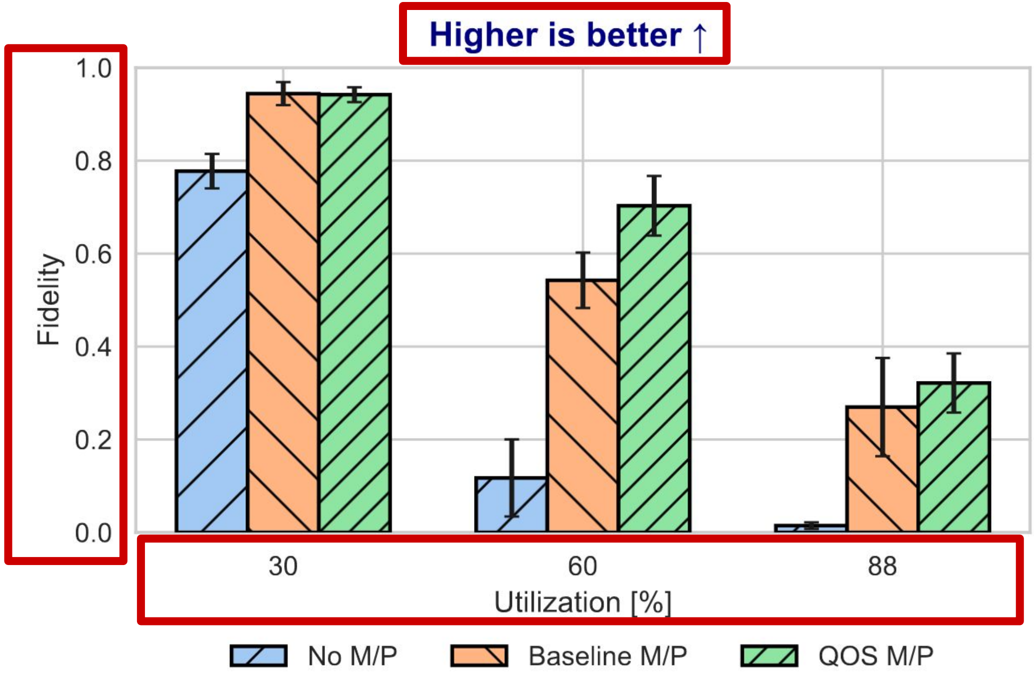
RQ #1: Compiler performance



Executed on IBM Kolkata, a 27-qubit Falcon QPU

QOS achieves **2.5x** and **430x** higher fidelity compared to CutQC and Qiskit

RQ #2: Multi-programmer performance



Executed on IBM Kolkata, a 27-qubit Falcon QPU

QOS achieves **9.6x** and **15%** higher fidelity than solo execution and the baseline

Conclusion

- Quantum computers face unique challenges:
  - Low fidelity performance
  - Low utilization
  - Vast heterogeneity
  - Vast load imbalance

- Quantum Operating System (QOS):
  - Improved fidelity with complementary compilation passes
  - Improved resource efficiency with multi-programming and scheduling

emmanouil.giortamis@tum.de

github.com/TUM-DSE/QOS