Quantum Compilers for People in a Hurry

Learning from the classical realm?

Florea Ioan-Albert, B.Sc.

Who am I?

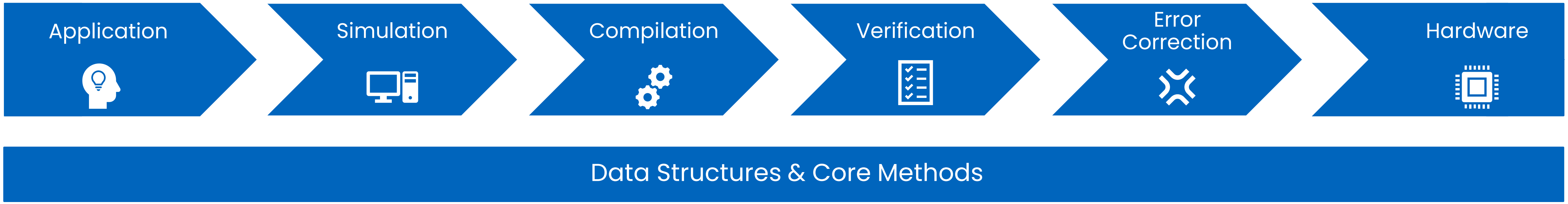
- Research Assistant in QC at the Chair for Design Automation (Prof. Robert Wille)

- Currently focusing on a joint Master's Thesis between TUM, LRZ and DLR
  Topic: Development of an Interface between two different quantum computing software stacks (MQSS – QCi Connect)

- Co-founder of RoQTeam

- Computer Scientist by training (B.Sc. also at TUM)

Software for Quantum Computing



Software for Quantum Computing

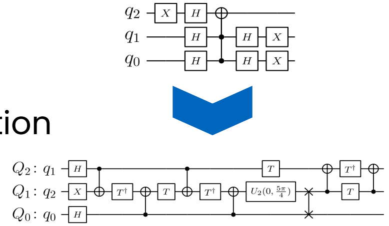
Munich Quantum Valley

Application

- Workflow from classical problem to quantum solution
- Automated encoding, execution & decoding

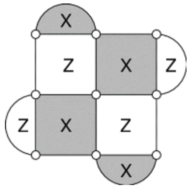
Compilation

- Automatic device selection
- Compiler optimization
- Technology-specific compilation
- Reversible synthesis



Error Correction

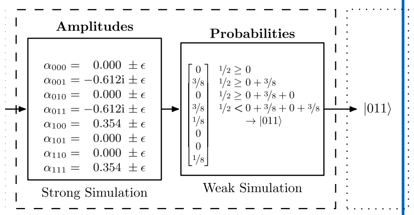
- Decoding algorithms
- Fault-tolerant state preparation
- Automated code construction and numerical simulations



Application → Simulation → Compilation → Verification → Error Correction → Hardware

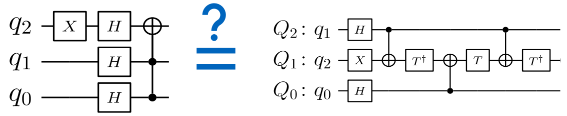
Simulation

- Classical simulation of quantum circuits based on decision diagrams
- Includes sampling, noise-aware simulation, Hybrid Schrödinger Feynman approaches, approximation strategies, expectation value computations, etc.



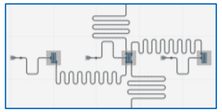
Verification

- Equivalence checking
- Verifying compilation results



Hardware

- Application specific physical design for superconducting platform



Data Structures & Core Methods

- Efficient data structures
- Dedicated core methods (optimal and heuristic)
- Based on C++ and Python

Decision Diagrams | Tensor Networks | ZX-Calculus

SAT/SMT Solvers | Machine Learning | Heuristics

Check it out!

https://mqt.readthedocs.io

The Munich Quantum Toolkit (MQT)

Munich Quantum Valley

All tools are available as open-source repositories on GitHub under the MIT license

MQT ProblemSolver — Application
A Tool for Solving Problems Using Quantum Computing
github.com/munich-quantum-toolkit/problemsolver
PyPI

MQT Bench — Application
Benchmarking Software and Tools for Quantum Computing
www.cda.cit.tum.de/mqtbench
github.com/munich-quantum-toolkit/bench
PyPI

MQT QUBOMaker — Application
A Framework for the Automatic Generation of QUBO Formulations
github.com/cda-tum/mqt-qubomaker
PyPI

MQT YAQS — Simulation
A Tool for Simulating Open Quantum Systems, Noisy Quantum Circuits, and Realistic Quantum Hardware
github.com/munich-quantum-toolkit/yaqs
PyPI

MQT DDSIM — Simulation
A Tool for Classical Quantum Circuit Simulation based on Decision Diagrams
github.com/munich-quantum-toolkit/ddsim
PyPI

MQT Predictor — Compilation
A Tool for Determining Good Quantum Circuit Compilation Options
github.com/munich-quantum-toolkit/predictor
PyPI

MQT IonShuttler — Compilation
A Tool for Generating Shuttling Schedules for QCCD Architectures
github.com/cda-tum/mqt-ion-shuttler
PyPI

MQT Qudits — Compilation
A Tool for Compiling High-Dimensional Quantum Systems
github.com/cda-tum/mqt-qudits
PyPI

MQT SyReC — Compilation
A Tool for the Synthesis of Reversible Circuits/Quantum Computing Oracles
github.com/munich-quantum-toolkit/syrec
PyPI

MQT QMAP — Compilation
A Tool for Quantum Circuit Mapping And Clifford Circuit Optimization/Synthesis
github.com/munich-quantum-toolkit/qmap
PyPI

MQT QCEC — Verification
A Tool for Quantum Circuit Equivalence Checking
github.com/munich-quantum-toolkit/qcec
PyPI

MQT Debugger — Verification
A semi-automated tool for debugging quantum programs
github.com/munich-quantum-toolkit/debugger
PyPI

MQT DDVis — Data Structures
A Web-Application visualizing Decision Diagrams for Quantum Computing
www.cda.cit.tum.de/app/ddvis

MQT Core — Data Structures
The Backbone of the MQT Intermediate Representation (IR) Decision Diagram and ZX Package
github.com/munich-quantum-toolkit/core
PyPI

MQT NAViz — Core Methods
A visualization software for neutral atom quantum computers.
github.com/munich-quantum-toolkit/naviz
PyPI

MQT QECC — QECC
A Tool for Quantum Error Correcting Codes
github.com/munich-quantum-toolkit/qecc
PyPI

https://mqt.readthedocs.io

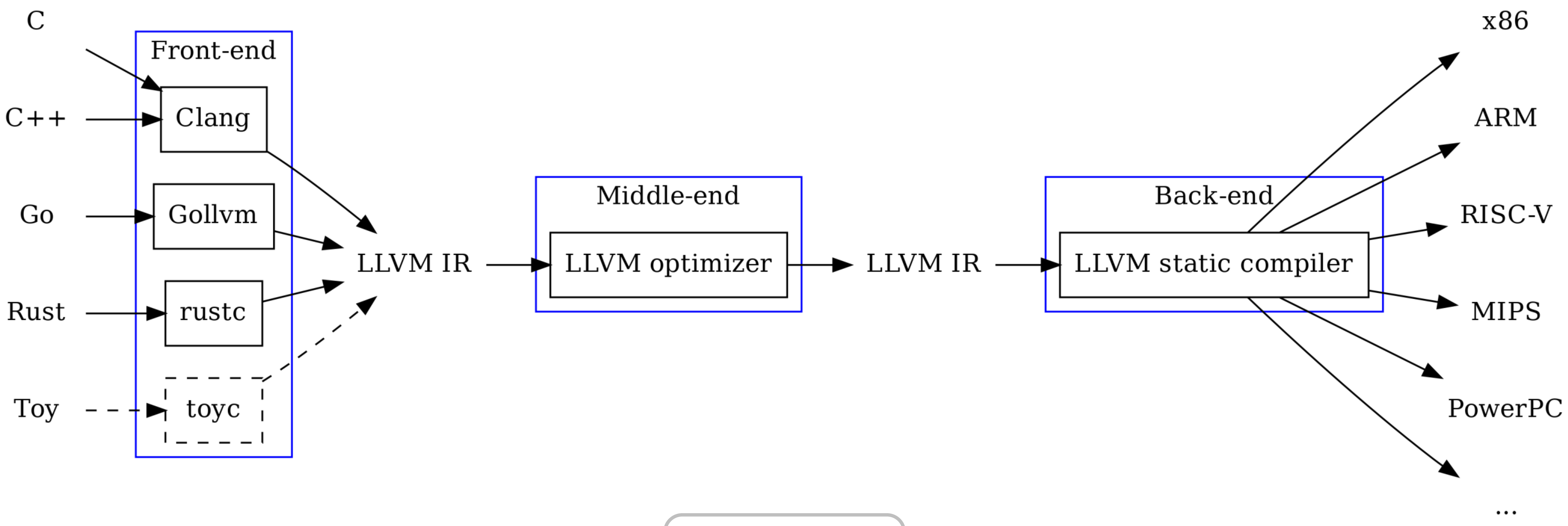
Over 1k ★ on GitHub

MUNICH QUANTUM TOOLKIT @ GitHub

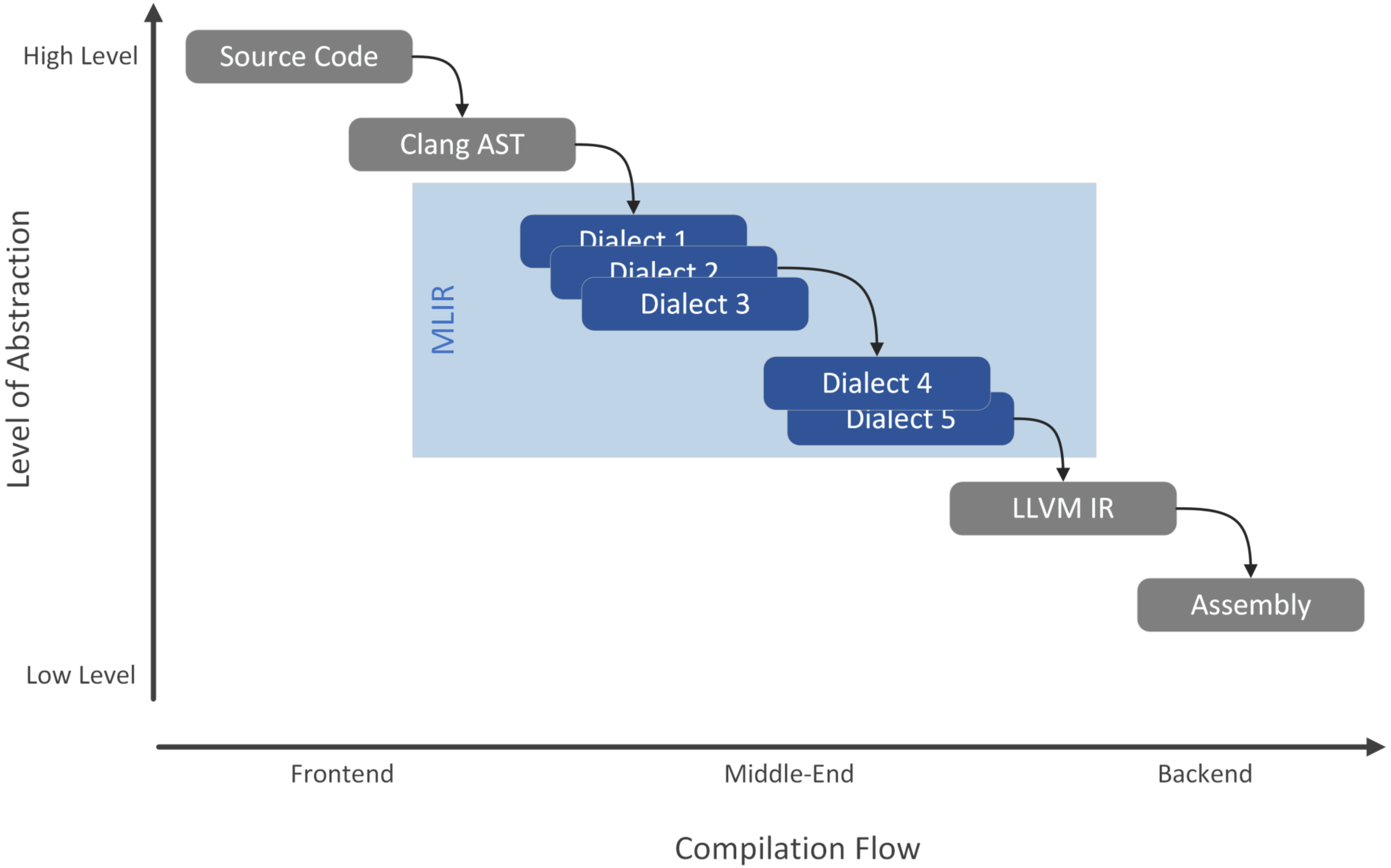
Over 2 Million Downloads on PyPI

MUNICH QUANTUM TOOLKIT @ python Package Index

(Modern) Classical Compilers



Multi Level Intermediate Representation - MLIR



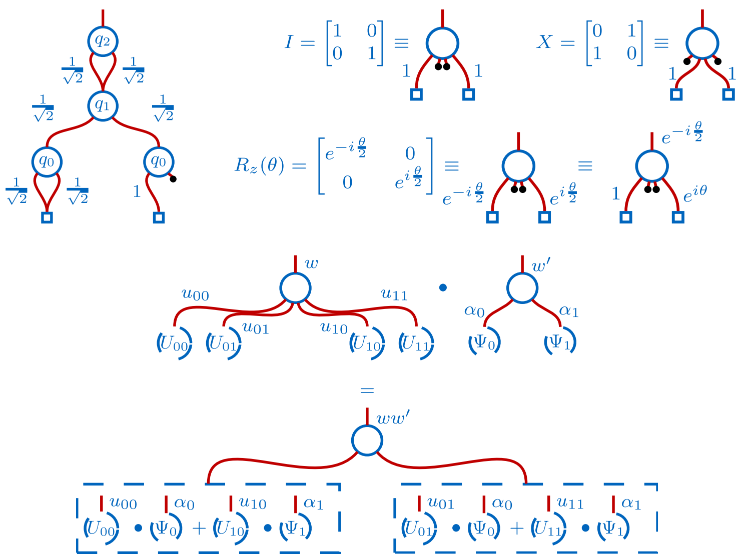
MQT Core – The Backbone of the MQT

C++20 and Python library for quantum computing

IR – QuantumComputation

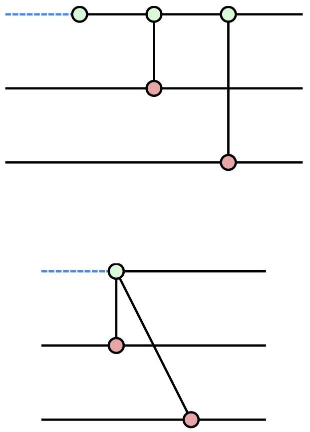
```python
from mqt.core.ir import QuantumComputation
from mqt.core.ir.operations import OpType

from math import pi

theta = 3 * pi / 8
precision = 3

# Create an empty quantum computation
qc = QuantumComputation()

# Counting register
q = qc.add_qubit_register(1, "q")

# Eigenstate register
psi = qc.add_qubit_register(1, "psi")

# Classical register for the result, the estimated phase is `0.c_2 c_1 c_0 * pi`
c = qc.add_classical_register(precision, "c")

# Prepare psi in the eigenstate |1>
qc.x(psi[0])

for i in range(precision):
  # Hadamard on the working qubit
  qc.h(q[0])

  # Controlled phase gate
  qc.cp(2**(precision - i - 1) * theta, q[0], psi[0])

  # Iterative inverse QFT
  for j in range(i):
    qc.if_(op_type=OpType.p, target=q[0], control_bit=c[j], params=[-pi / 2**(i - j)])
  qc.h(q[0])

  # Measure the result
  qc.measure(q[0], c[i])

  # Reset the qubit if not finished
  if i < precision - 1:
    qc.reset(q[0])
```

Decision Diagram (DD) Package



Basis for DDSIM and QCEC

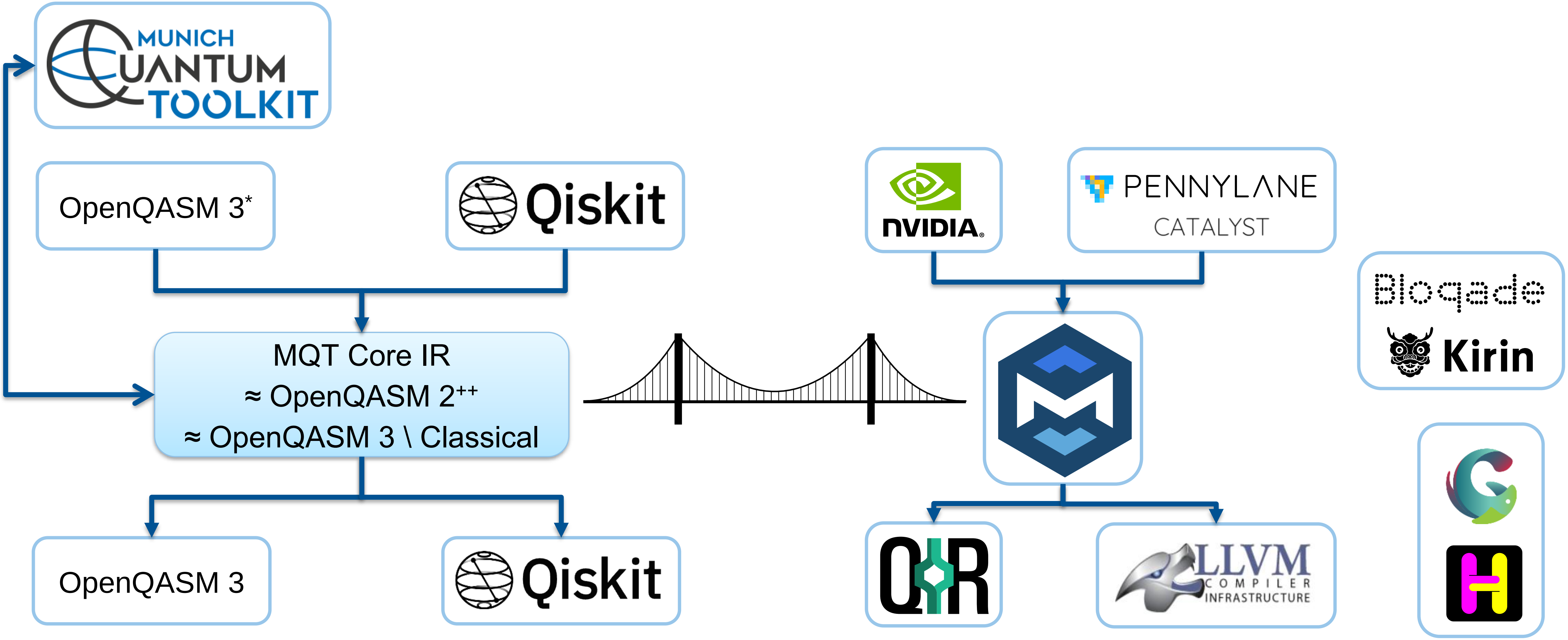
ZX-Calculus Package



https://github.com/munich-quantum-toolkit/core
or simply **(uv)** **`pip install mqt.core`**

84 ★ on GitHub, **370k Downloads** from PyPI

MQT Core – The Backbone of the MQT



"quantum-first"

"classical-first"

Building Bridges

PENNYLANE

```python
import pennylane as qml
from pennylane.tape import QuantumTape

# Define GHZ circuit
def ghz_circuit():
  qml.Hadamard(wires=0)
  qml.CNOT(wires=[0, 1])
  qml.CNOT(wires=[1, 2])

# Convert to OpenQASM
with QuantumTape() as tape:
  ghz_circuit()
qasm = tape.to_openqasm()

# Map circuit with QMAP
mapped_qasm = map_circuit(qasm)

# Convert back to PennyLane
mapped_ghz = qml.from_Qasm(mapped_qasm)
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[0];
cx q[0], q[1];
cx q[1], q[2];
measure q -> c;
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[1];
cx q[1], q[0];
cx q[1], q[2];
measure q -> c;
```

MUNICH QUANTUM TOOLKIT

```python
from mqt.core import QuantumComputation
import mqt.qmap as qmap

def map_circuit(qasm):
  # Convert OpenQASM to MQT Core IR
  mqt_qc = \
QuantumComputation.from_qasm_str(qasm)

  n_qubits = 3
  cm = set([(0,1),(1,0),(1,2),(2,1)])
  arc = qmap.Architecture(n_qubits, cm)
  default_config = qmap.Configuration()

  # Map circuit with QMAP
  mapped_qc, _ = \
qmap.map(mqt_qc, arc, default_config)

  # Convert to OpenQASM
  return \
QuantumComputation.qasm2_str(mapped_qc)
```

Building Bridges

PENNYLANE

```python
import pennylane as qml
from pennylane.tape import QuantumTape

# Define GHZ circuit
def ghz_circuit():
  qml.Hadarmard(wires=0)
  qml.CNOT(wires=[0, 1])
  qml.CNOT(wires=[1, 2])

# Convert to OpenQASM
with QuantumTape() as tape:
  ghz_circuit()
qasm = tape.to_openqasm()

# Map circuit with QMAP
mapped_qasm = map_circuit(qasm)

# Convert back to PennyLane
mapped_ghz = qml.from_Qasm(mapped_qasm)
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[0];
cx q[0], q[1];
cx q[1], q[2];
measure q -> c;
```

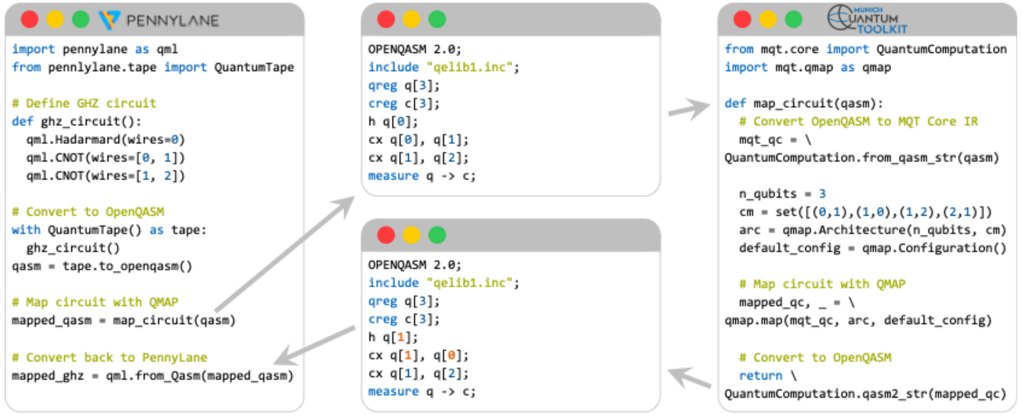
MUNICH QUANTUM TOOLKIT

```python
from mqt.core import QuantumComputation
import mqt.qmap as qmap

def map_circuit(qasm):
  # Convert OpenQASM to MQT Core IR
  mqt_qc = \
QuantumComputation.from_qasm_str(qasm)

  n_qubits = 3
  cm = set([(0,1),(1,0),(1,2),(2,1)])
  arc = qmap.Architecture(n_qubits, cm)
  default_config = qmap.Configuration()

  # Map circuit with QMAP
  mapped_qc, _ = \
qmap.map(mqt_qc, arc, default_config)

  # Convert to OpenQASM
  return \
QuantumComputation.qasm2_str(mapped_qc)
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[1];
cx q[1], q[0];
cx q[1], q[2];
measure q -> c;
```

loose integration

Building Bridges



loose integration

PENNYLANE

```python
import pennylane as qml
from catalyst import measure
from mqt_plugin import QMAP, plugin

@qml.qjit(pass_plugins={plugin}, dialect_plugins={plugin})
@QMAP({"cMap": [(0,1),(1,0),(1,2),(2,1)]})
@qml.qnode(qml.device("lightning.qubit", wires=3))
def ghz_circuit():
  qml.Hadamard(wires=[0])
  qml.CNOT(wires=[0, 1])
  qml.CNOT(wires=[1, 2])
  return [measure(i) for i in range(3)]
```

tight integration

MLIR Support in MUNICH QUANTUM TOOLKIT

- Over 30 closed Issues and PRs
  - github.com/munich-quantum-toolkit/core/milestone/8
- Over 200 commits just on the plugin itself
  - github.com/munich-quantum-toolkit/core/pull/881

Key Challenges and Obstacles

Steep Learning Curve
Complex MLIR concepts
C++

Version Compatibility
Can we align LLVM versions?

Packaging & Distribution
How to ship all of this?

QIR Support
Support beyond the base profile

Uniform Exchange Format
Can we agree on some common format?

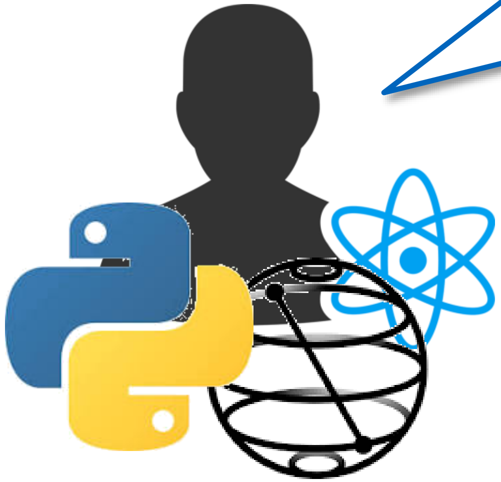
JEFF BRIDGES COMPILATION

github.com/unitaryfoundation/jeff

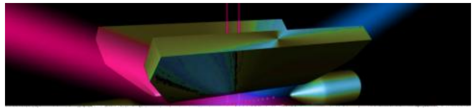
The Current State

Individual Physics Researcher

I want to tune the pulse to execute an X-gate!

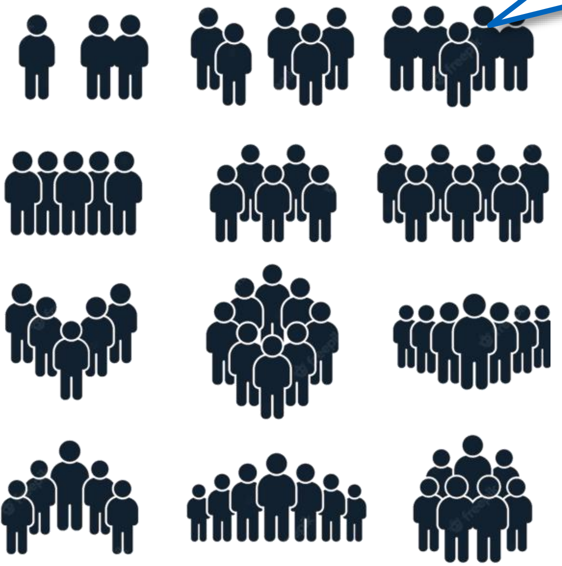


Quantum Device



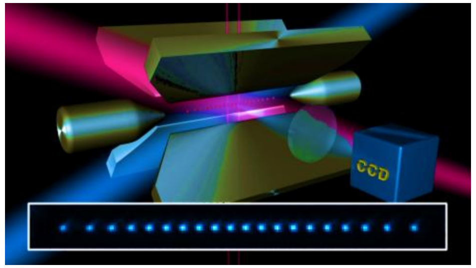
The Vision

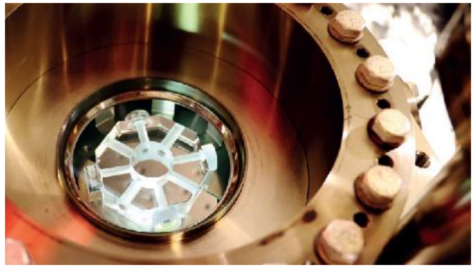
Domain Experts



Comprehensive Software Stack

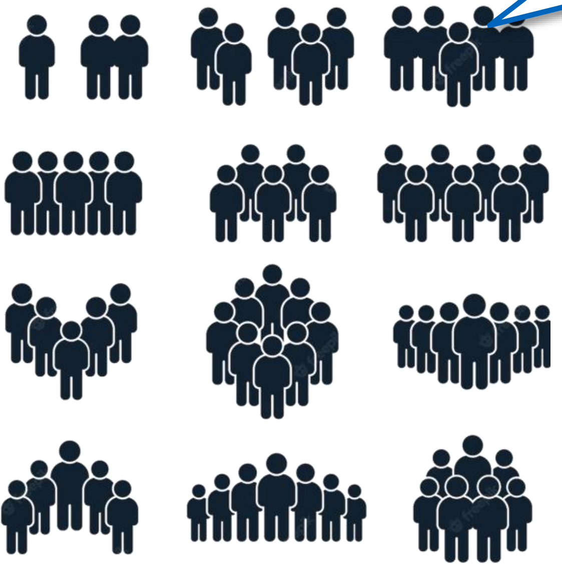
Many/Different Quantum Devices



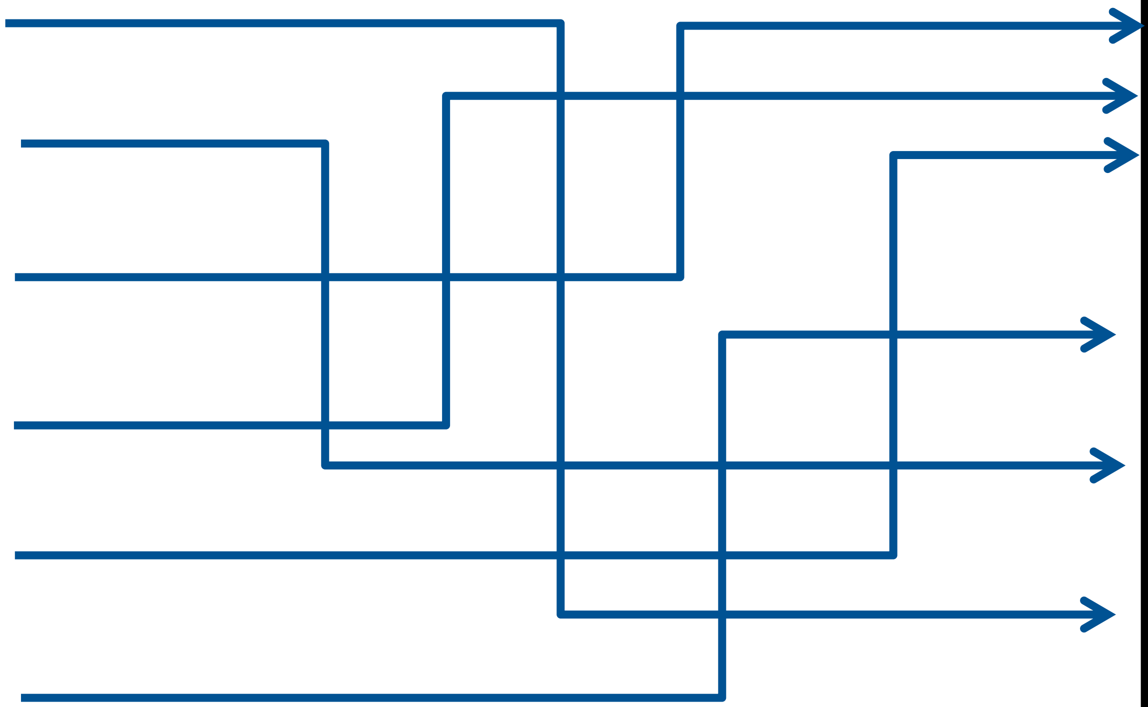


The Vision

Domain Experts

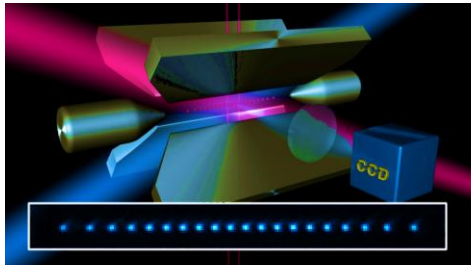


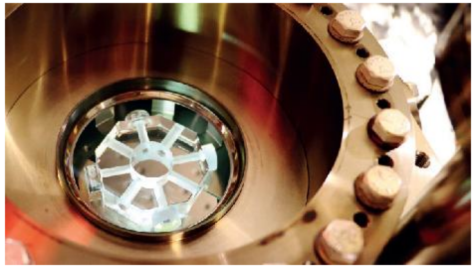
Comprehensive Software Stack



❌ **What we do not want**

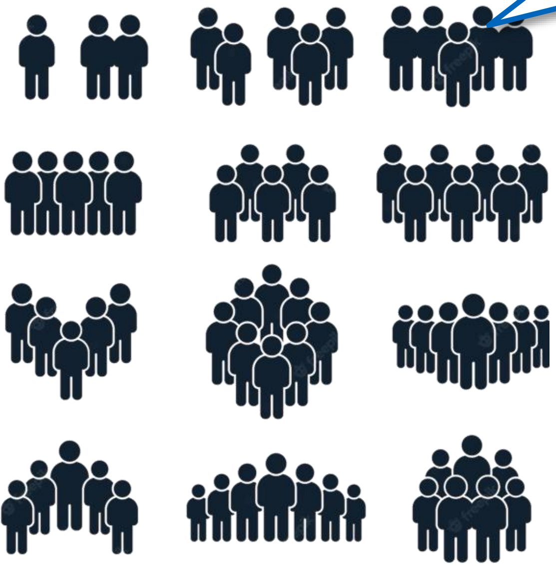
Many/Different Quantum Devices



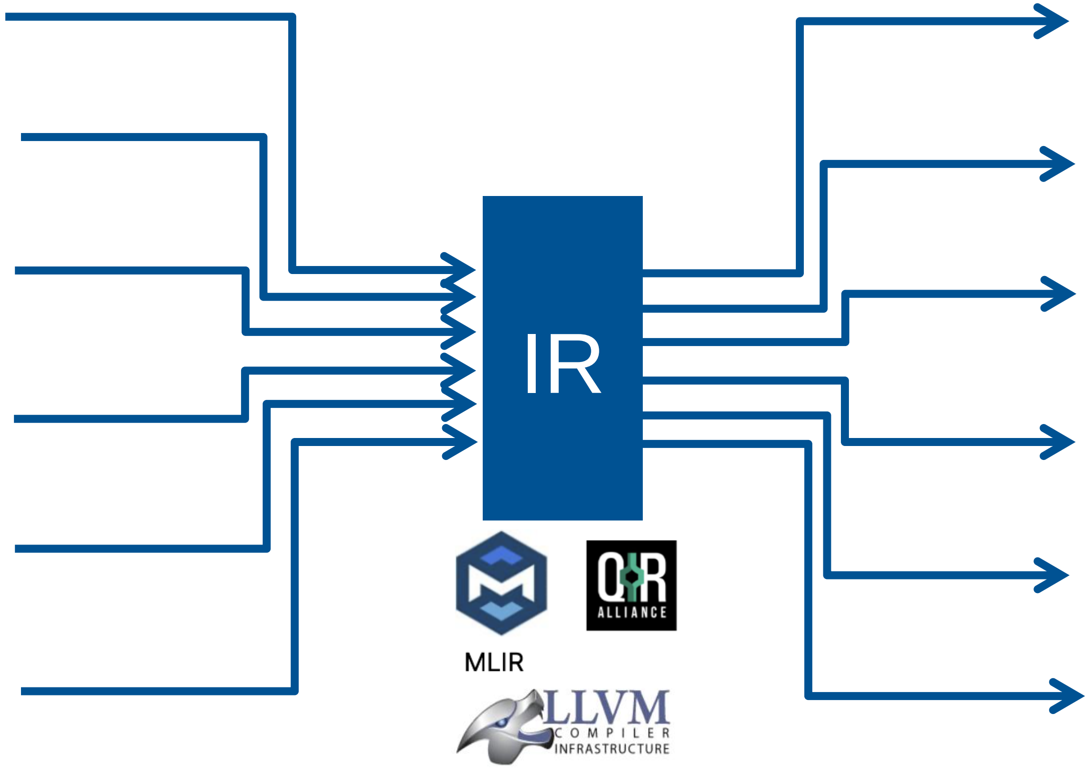


The Vision

Domain Experts

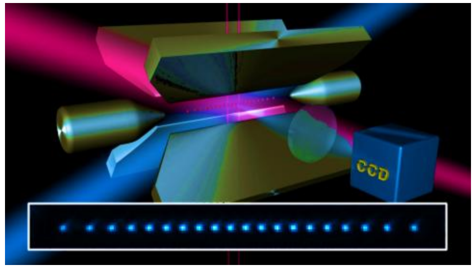


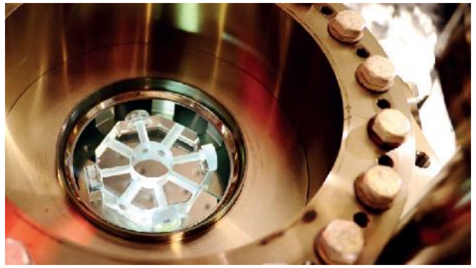
Comprehensive Software Stack



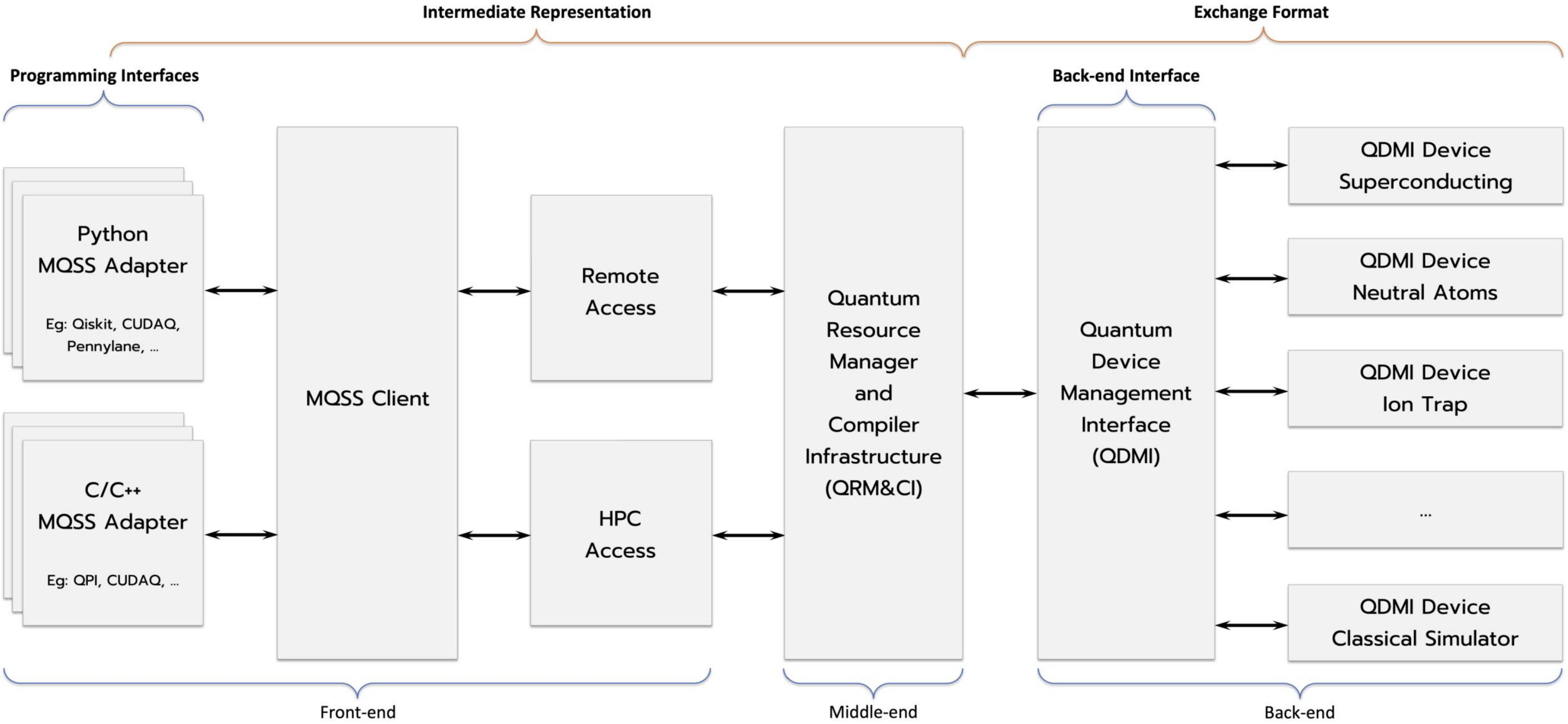
☑ **What we want**

Many/Different Quantum Devices





Quantum Compilers in HPCQC



Burgholzer et al., *The Munich Quantum Software Stack*, arXiv:2509.02674 (2025)

What's Next

Try out our Tools

Shell

```
$ (uv) pip install mqt.core
$ (uv) pip install mqt.qmap
```

Leave a Star on GitHub

Chair for Design Automation @ TUM
cda.cit.tum.de/

Check out MQSS
github.com/Munich-Quantum-Software-Stack

MQT Core GitHub
github.com/munich-quantum-toolkit/core

MQT Core Documentation
mqt.readthedocs.io/projects/core/en/latest/

MQT QMAP GitHub
mqt.readthedocs.io/projects/qmap/en/latest/

MQT QMAP Documentation
github.com/munich-quantum-toolkit/qmap

Thank you for your attention!