Munich Quantum Valley

Connecting the HPC and the Quantum Community

A Tutorial on High-Performance Software for Quantum Computing

Lukas Burgholzer, Patrick Hopf, Robert Wille, and Team

Technical University of Munich, Germany

lukas.burgholzer@tum.de

https://www.cda.cit.tum.de/research/quantum/

@burgholzer

MUNICH QUANTUM SOFTWARE COMPANY

CHAIR FOR DESIGN AUTOMATION

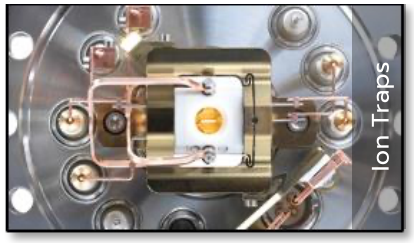
Munich Quantum Valley

The Big Picture

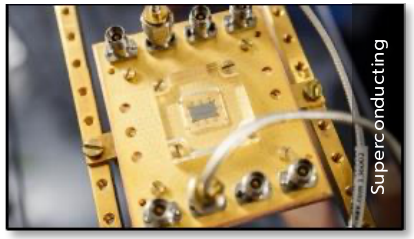
Domain Experts

Quantum Devices

AQT @ LRZ



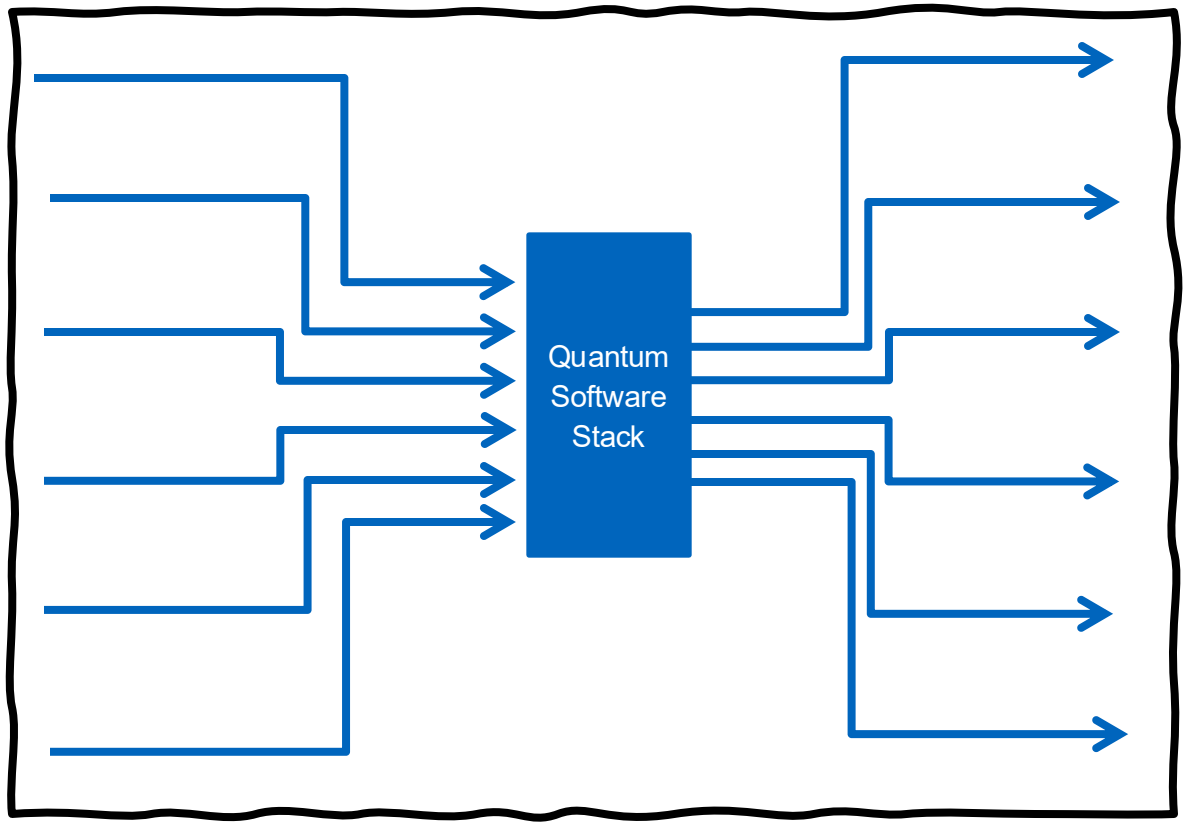
WMI @ LRZ



planqc @ LRZ

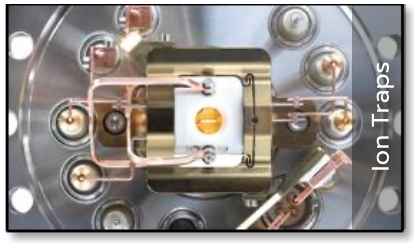
The Big Picture

Domain Experts

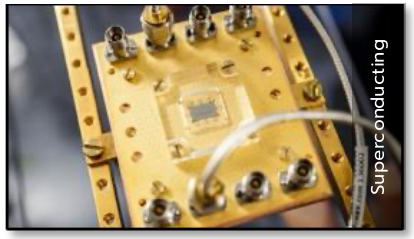


Quantum Devices

AQT @ LRZ



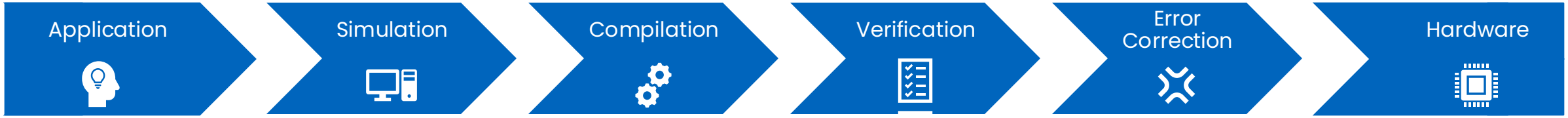
WMI @ LRZ



planqc @ LRZ

Software for Quantum Computing

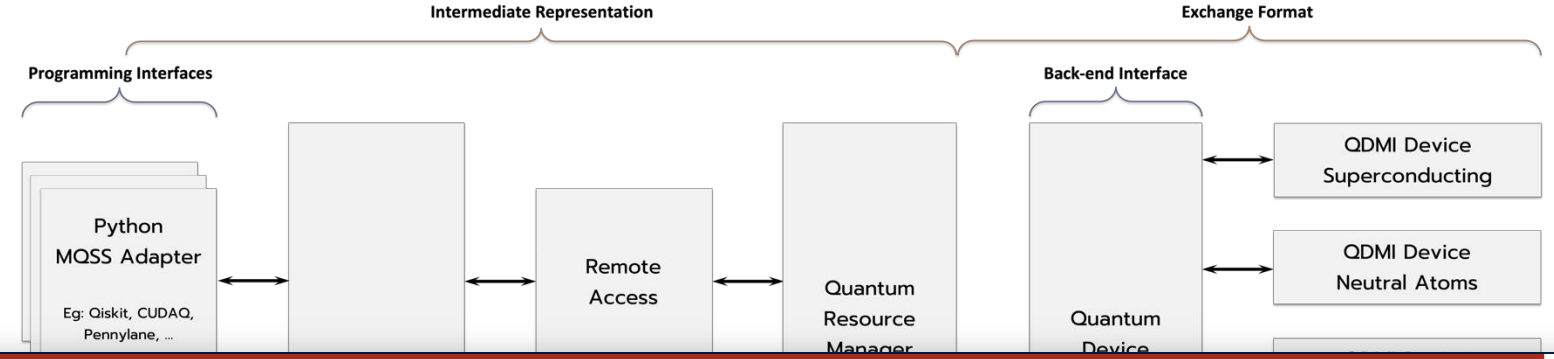
Needed across the whole spectrum…



Data Structures & Core Methods

MQSS
Munich Quantum Software Stack

Overview Paper
arxiv.org/abs/2509.02674



arXiv > quant-ph > arXiv:2509.02674

Search... Help | Adva

Quantum Physics

[Submitted on 2 Sep 2025]

The Munich Quantum Software Stack: Connecting End Users, Integrating Diverse Quantum Technologies, Accelerating HPC

Lukas Burgholzer, Jorge Echavarria, Patrick Hopf, Yannick Stade, Damian Rovara, Ludwig Schmid, Ercüment Kaya, Burak Mete, Muhammad Nufail Farooqi, Minh Chung, Marco De Pascale, Laura Schulz, Martin Schulz, Robert Wille

MQT Repositories

Leave a ★

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
munich-quantum-toolkit/problemsolver

MQT Bench — Application
Benchmarking Software and Tools for Quantum Computing
mqt-bench.app
munich-quantum-toolkit/bench

MQT QuSAT — Core Methods
A Tool for Encoding Quantum Computing using Satisfiability Testing (SAT) Techniques
$F \wedge (x_1 \wedge \neg x_2)$
$F \wedge (x_3 \wedge x_2)$
munich-quantum-toolkit/qusat

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

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

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

MQT IonShuttler — Compilation
A Tool for Generating Shuttling Schedules for QCCD Architectures
munich-quantum-toolkit/ionshuttler

MQT Qudits — Compilation
A Tool for Compiling High-Dimensional Quantum Systems
munich-quantum-toolkit/qudits

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

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

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

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

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
munich-quantum-toolkit/core

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

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

https://mqt.readthedocs.io

https://github.com/munich-quantum-toolkit

Agenda

- First Session (13:30 – 15:00)
  - Quantum Computing 101
  - Simulation of Quantum Circuits
  - Practical I: Simulation
  - Compilation of Quantum Circuits
  - Practical II: Compilation

- Second Session (15:30 – 17:00)
  - Verification of Quantum Circuits
  - Practical III: Verification
  - Quantum Computing Applications and End User Support
  - Practical IV: Device Selection
  - HPC+QC Integration
  - Summary and Discussion

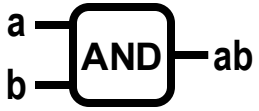
Classical Computing

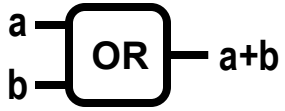
ASM

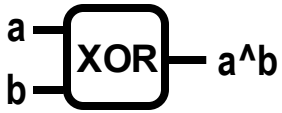
```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

1110
0110
0001
1001

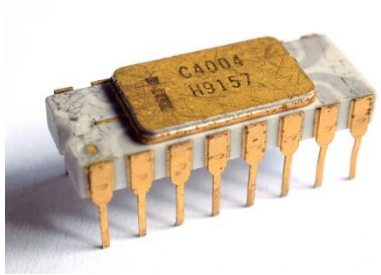
Classical Computing

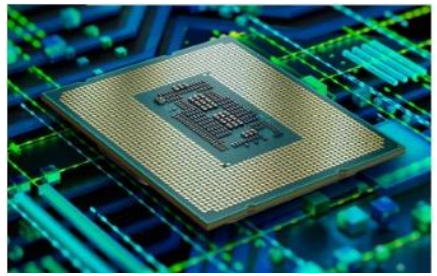


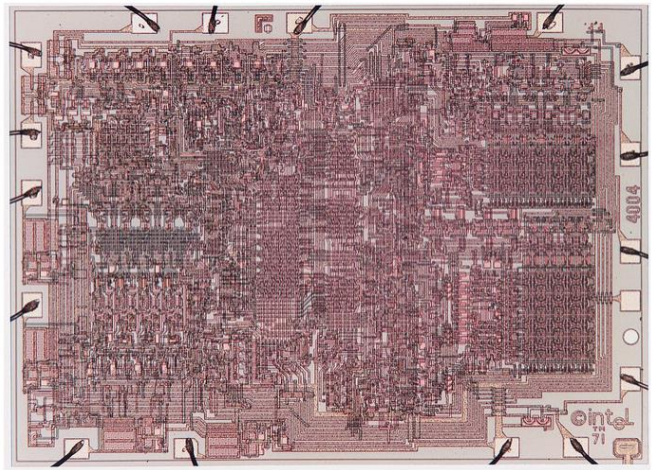


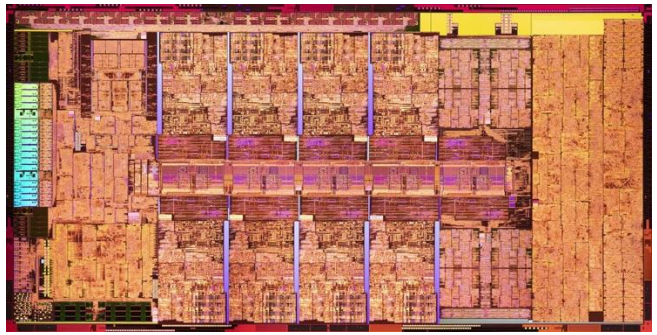


1971 Intel® 4004		2021 12th Generation Intel® Core™
2300	Number of Transistors	Billions
16	Number of Pins	1700
750 kHz	Frequency	5.2 GHz
4-bit	Instruction Set	64-bit
1	Number of Cores	16









HOW? Design Automation!

Classical Computing

ASM

```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

1110
0110
0001
1001

Quantum Computing

ASM

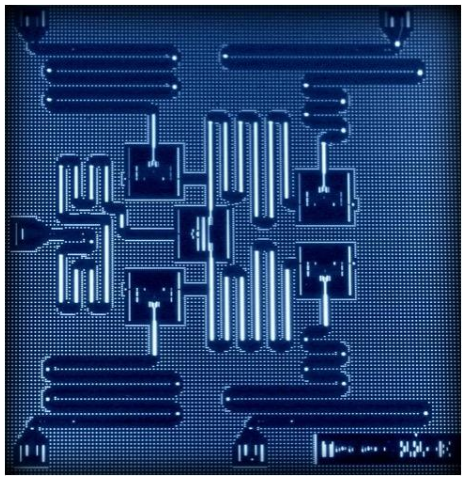
```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

1110
0110
0001
1001

Quantum Computing

ASM

```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

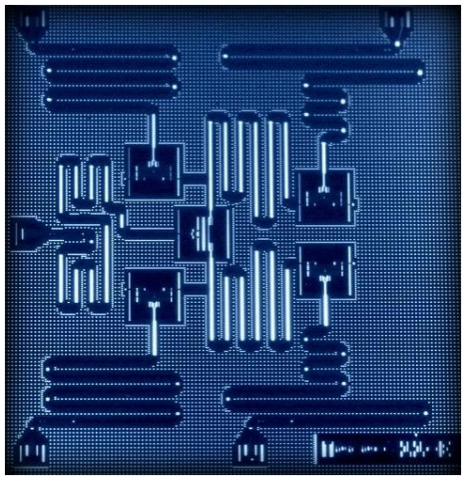


1110
0110
0001
1001

Quantum Computing

QASM

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];
h q[2];
cx q[2], q[1];
cx q[2], q[0];
measure q[2] -> c[2];
```



1110

0110

0001

1001

Quantum Computing

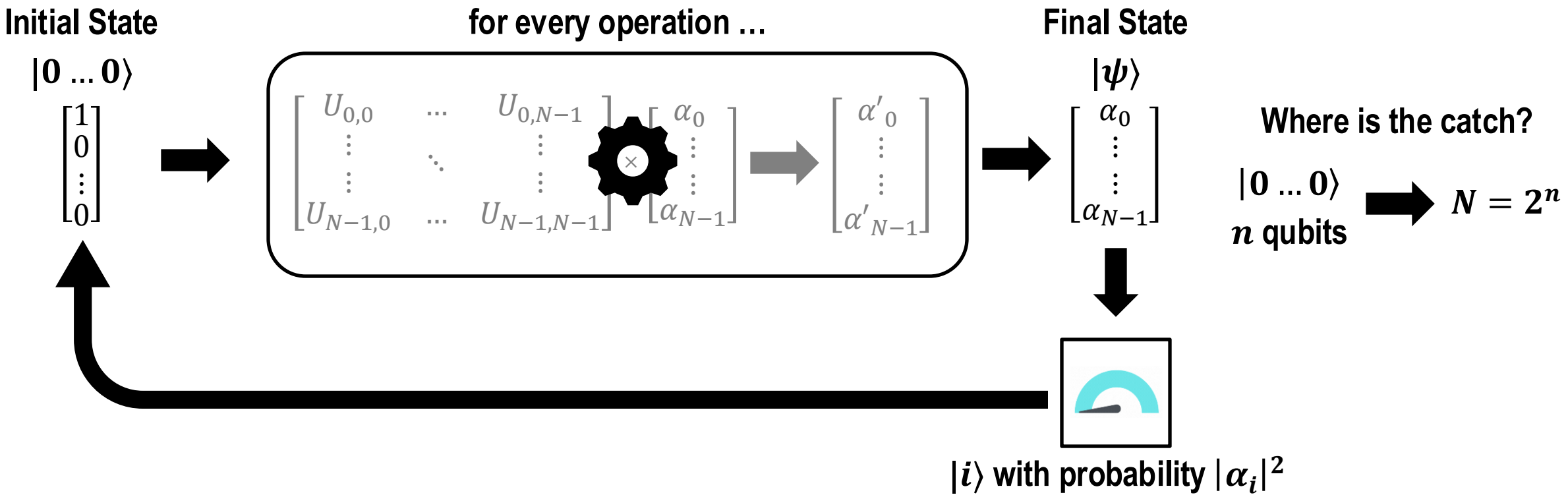
Quantum Operation | Quantum State | Evolved Quantum State

$$\begin{bmatrix} U_{0,0} & \cdots & U_{0,N-1} \\ \vdots & \ddots & \vdots \\ U_{N-1,0} & \cdots & U_{N-1,N-1} \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{N-1} \end{bmatrix} \rightarrow \begin{bmatrix} \alpha'_0 \\ \vdots \\ \alpha'_{N-1} \end{bmatrix}$$

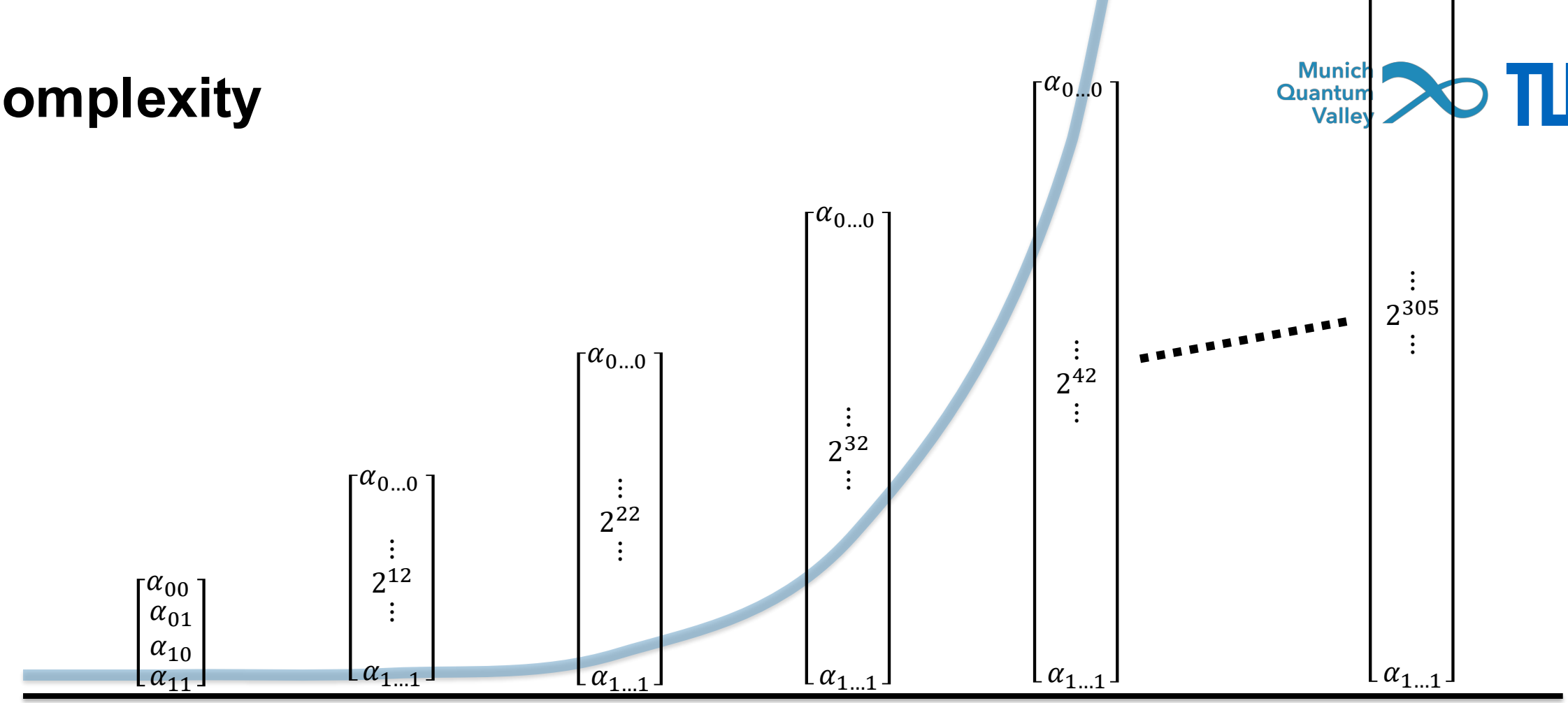
$$U_{i,j} \in \mathbb{C} \qquad \alpha_i \in \mathbb{C}$$

$$U^\dagger U = I \qquad \sum |\alpha_i|^2 = 1$$

Quantum Computing

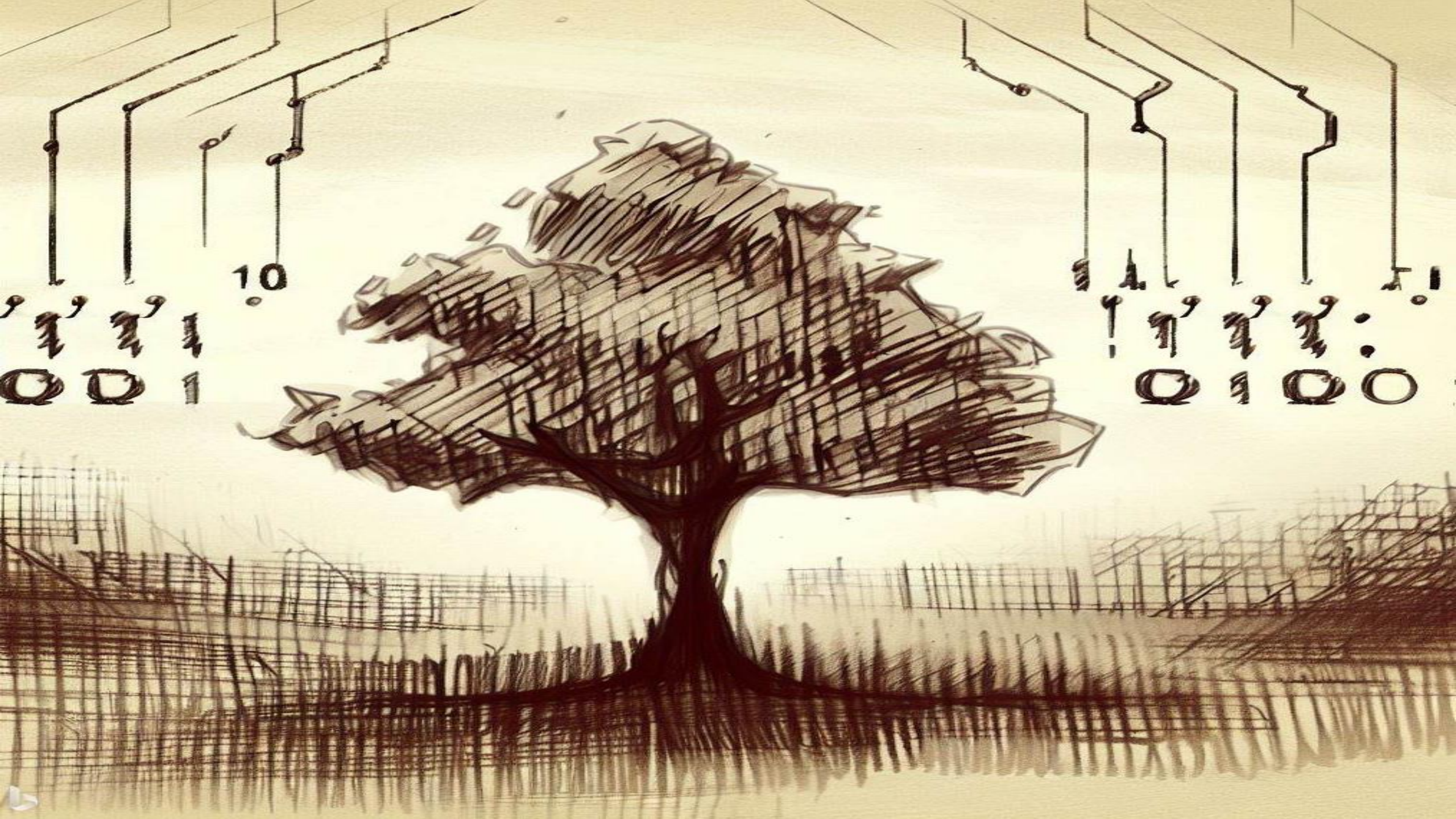


Complexity



#qubits	2	12	22	32	42	305
memory	64B	64KiB	64MiB	64GiB	64TiB	
	by hand	L1 cache	L3 cache	RAM	HPC	

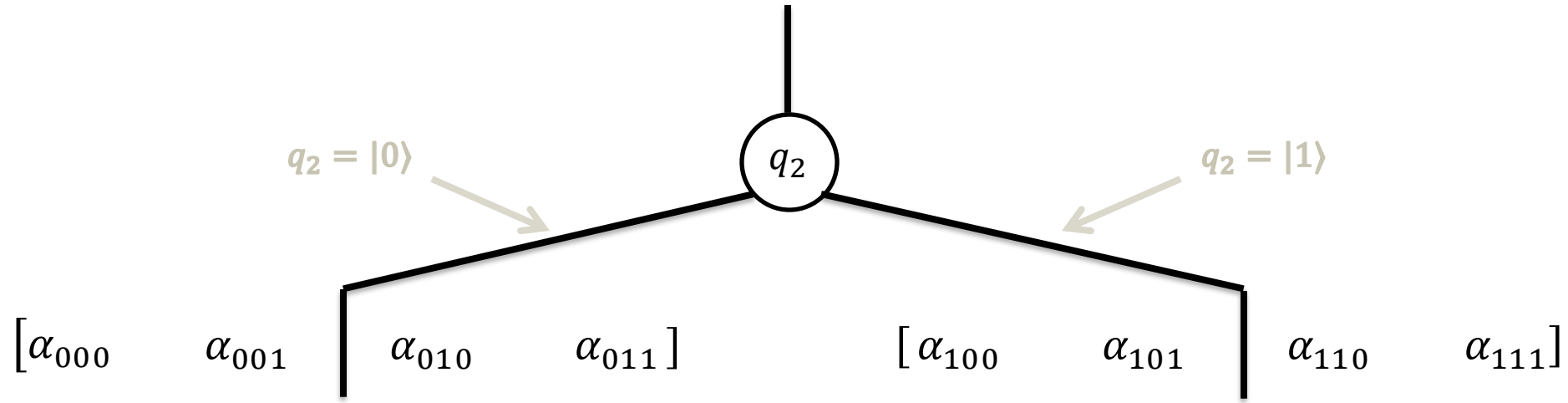
HOW? Design Automation!



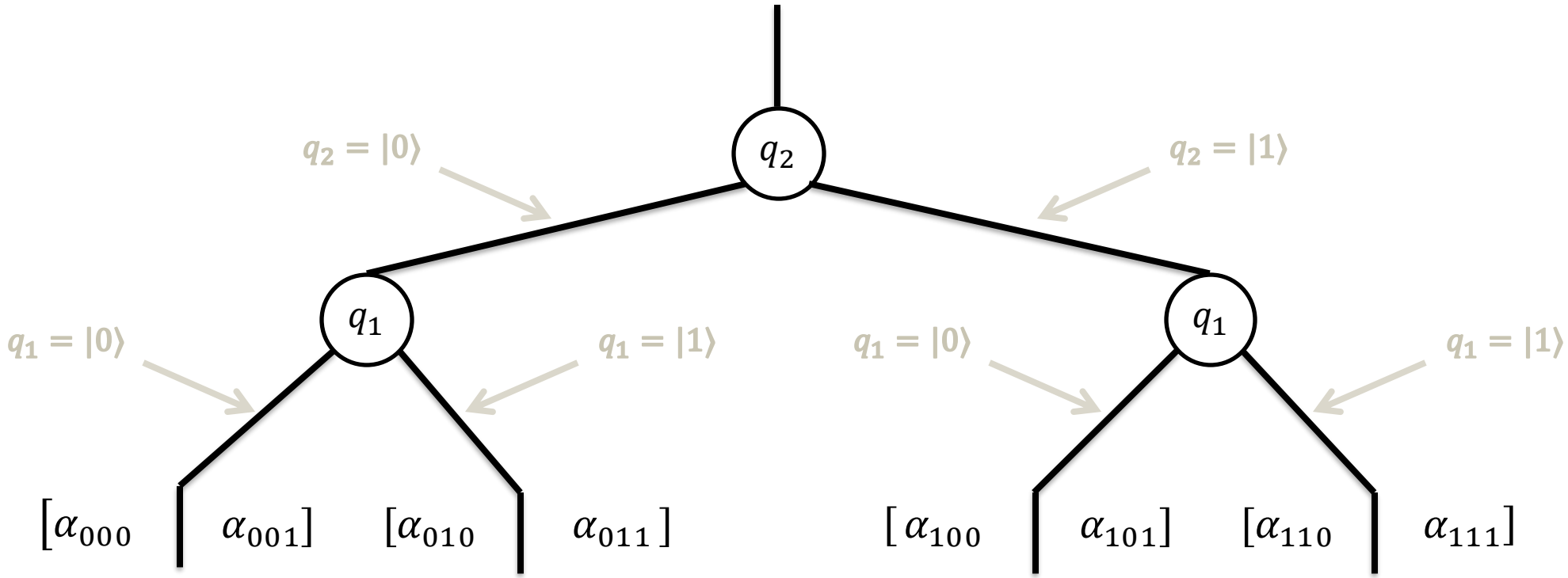
Structure

$$\left[\alpha_{000} \quad \alpha_{001} \quad \alpha_{010} \quad \alpha_{011} \;\middle|\; \alpha_{100} \quad \alpha_{101} \quad \alpha_{110} \quad \alpha_{111}\right]$$

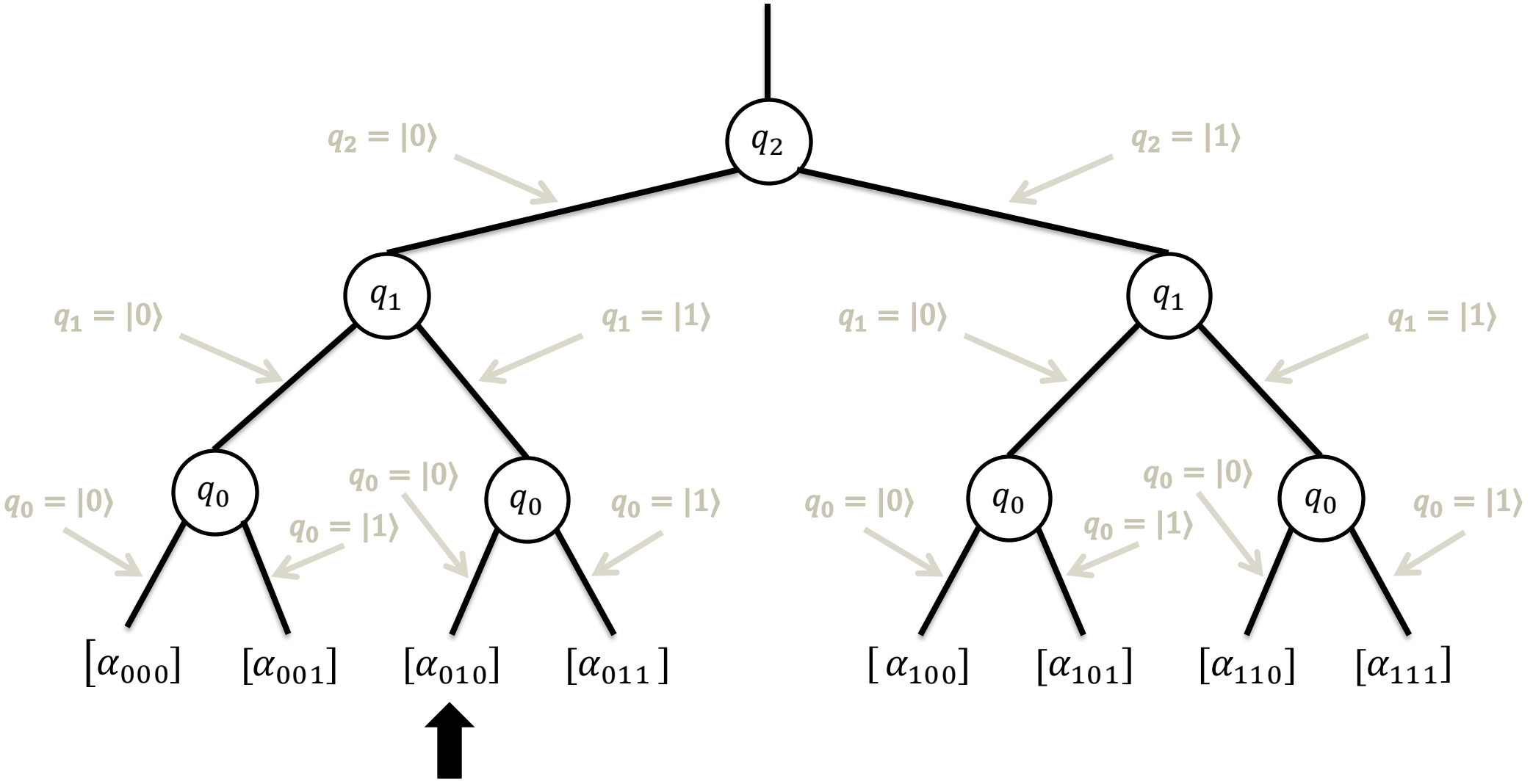
Structure


$q_2 = |0\rangle$
q_2
$q_2 = |1\rangle$
$[\alpha_{000}$ α_{001} α_{010} $\alpha_{011}]$ $[\alpha_{100}$ α_{101} α_{110} $\alpha_{111}]$

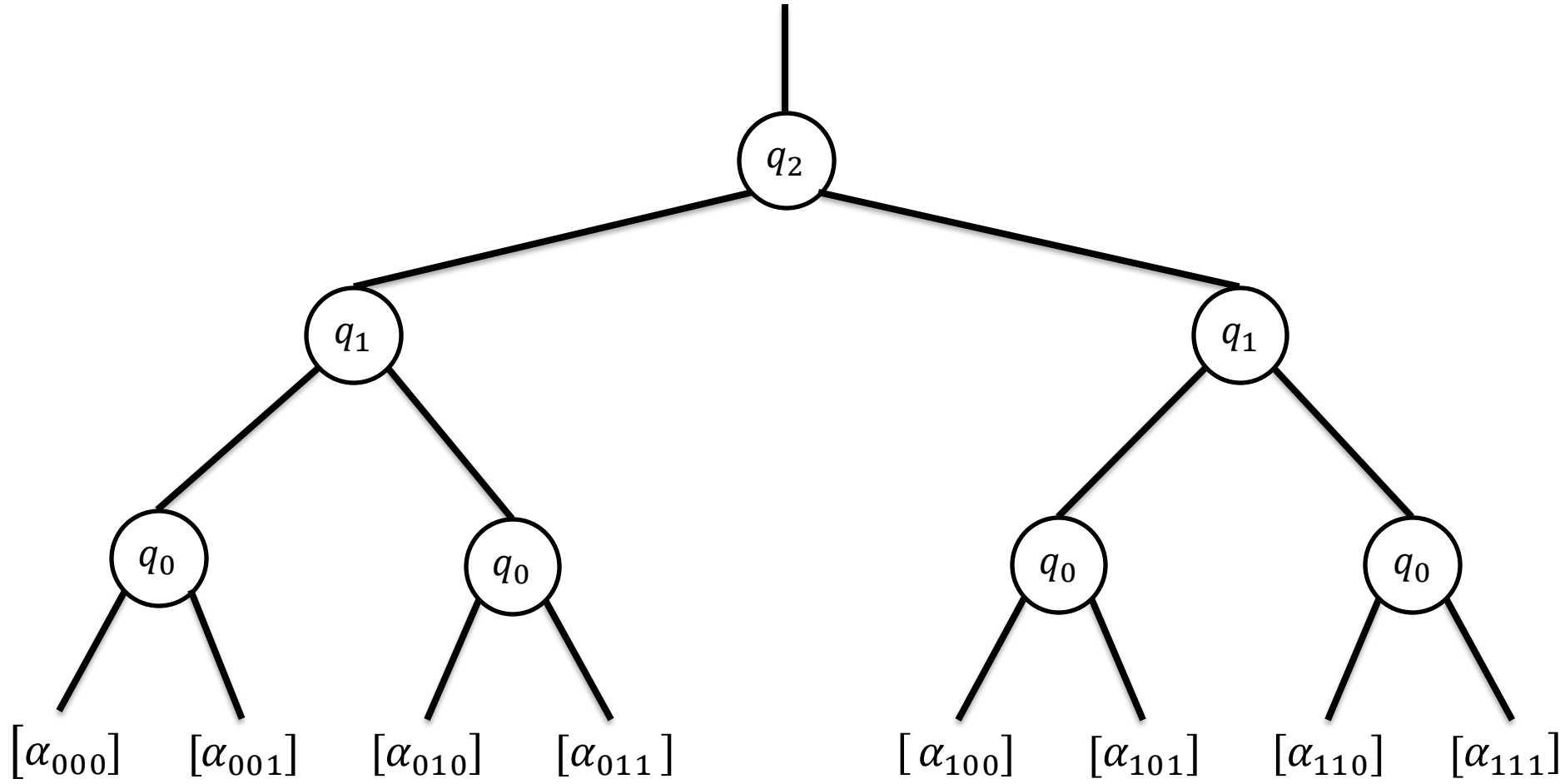
Structure


q_2
$q_2 = |0\rangle$
$q_2 = |1\rangle$
q_1
$q_1 = |0\rangle$
$q_1 = |1\rangle$
q_1
$q_1 = |0\rangle$
$q_1 = |1\rangle$
$[\alpha_{000}$
$\alpha_{001}]$
$[\alpha_{010}$
$\alpha_{011}]$
$[\alpha_{100}$
$\alpha_{101}]$
$[\alpha_{110}$
$\alpha_{111}]$

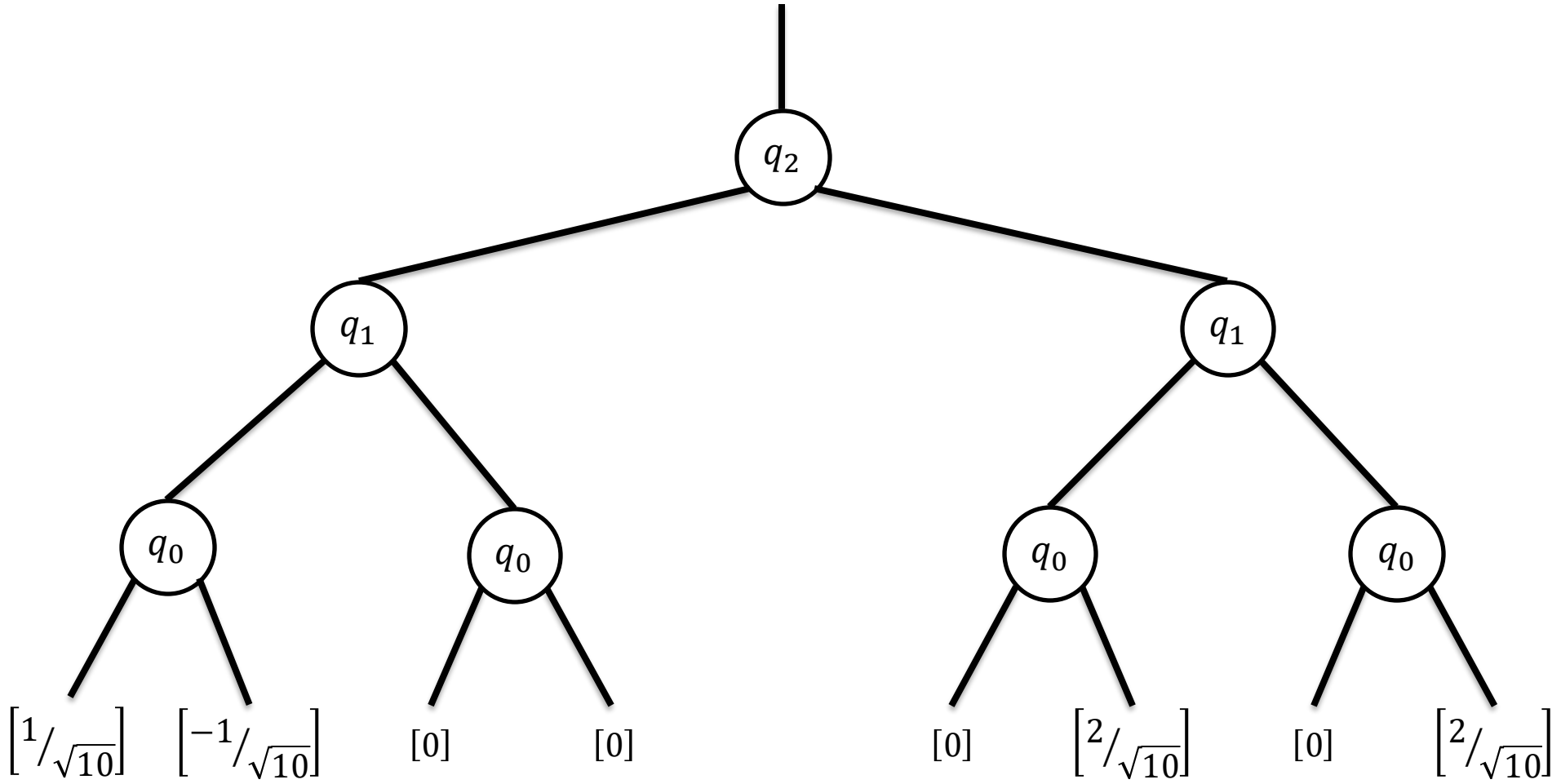
Structure


q_2
$q_2 = |0\rangle$
$q_2 = |1\rangle$
q_1
$q_1 = |0\rangle$
$q_1 = |1\rangle$
q_1
$q_1 = |0\rangle$
$q_1 = |1\rangle$
q_0
$q_0 = |0\rangle$
$q_0 = |1\rangle$
q_0
$q_0 = |0\rangle$
$q_0 = |1\rangle$
q_0
$q_0 = |0\rangle$
$q_0 = |1\rangle$
q_0
$q_0 = |0\rangle$
$q_0 = |1\rangle$
$[\alpha_{000}]$
$[\alpha_{001}]$
$[\alpha_{010}]$
$[\alpha_{011}]$
$[\alpha_{100}]$
$[\alpha_{101}]$
$[\alpha_{110}]$
$[\alpha_{111}]$

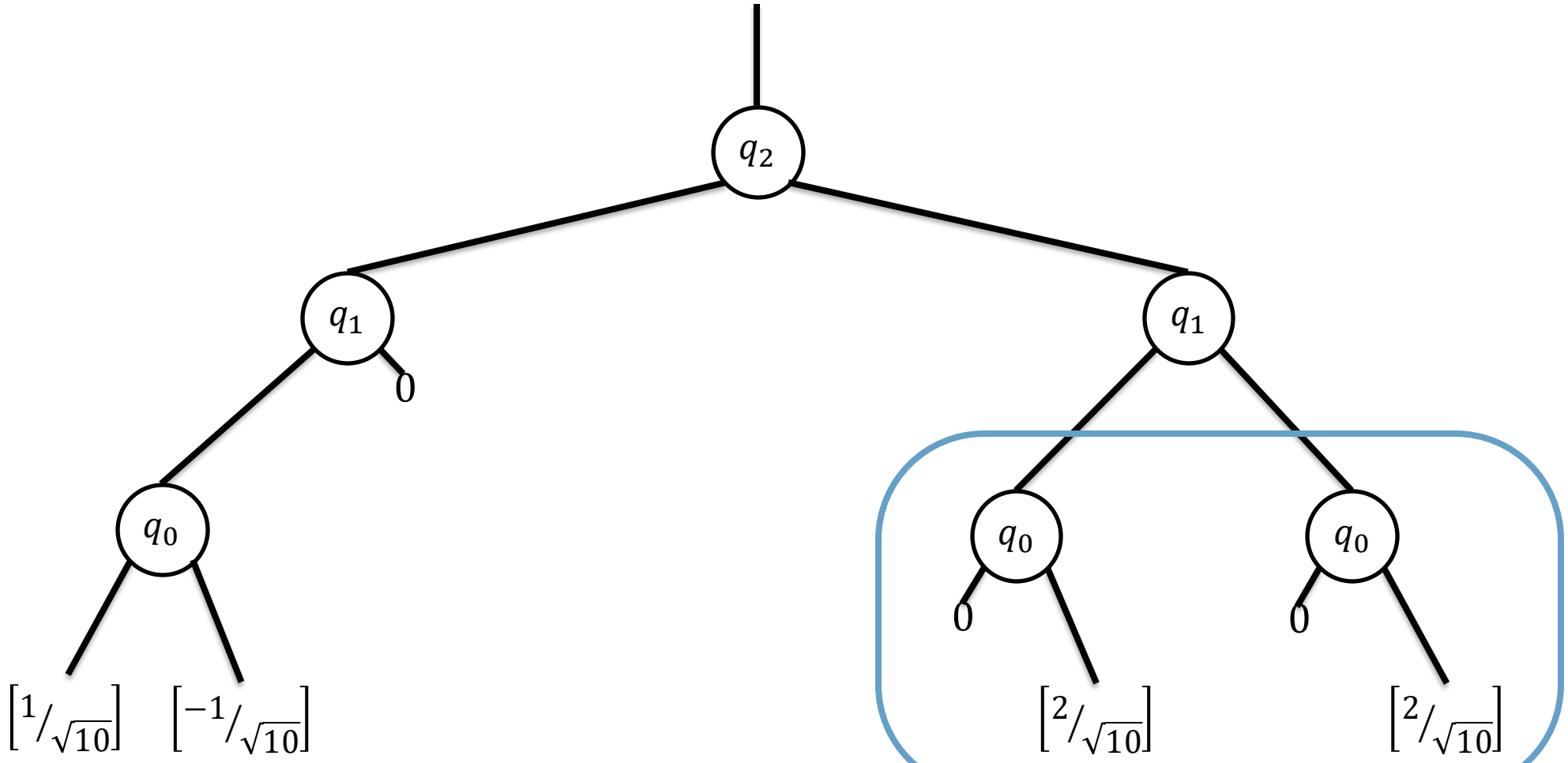
Structure + Redundancy



Structure + Redundancy

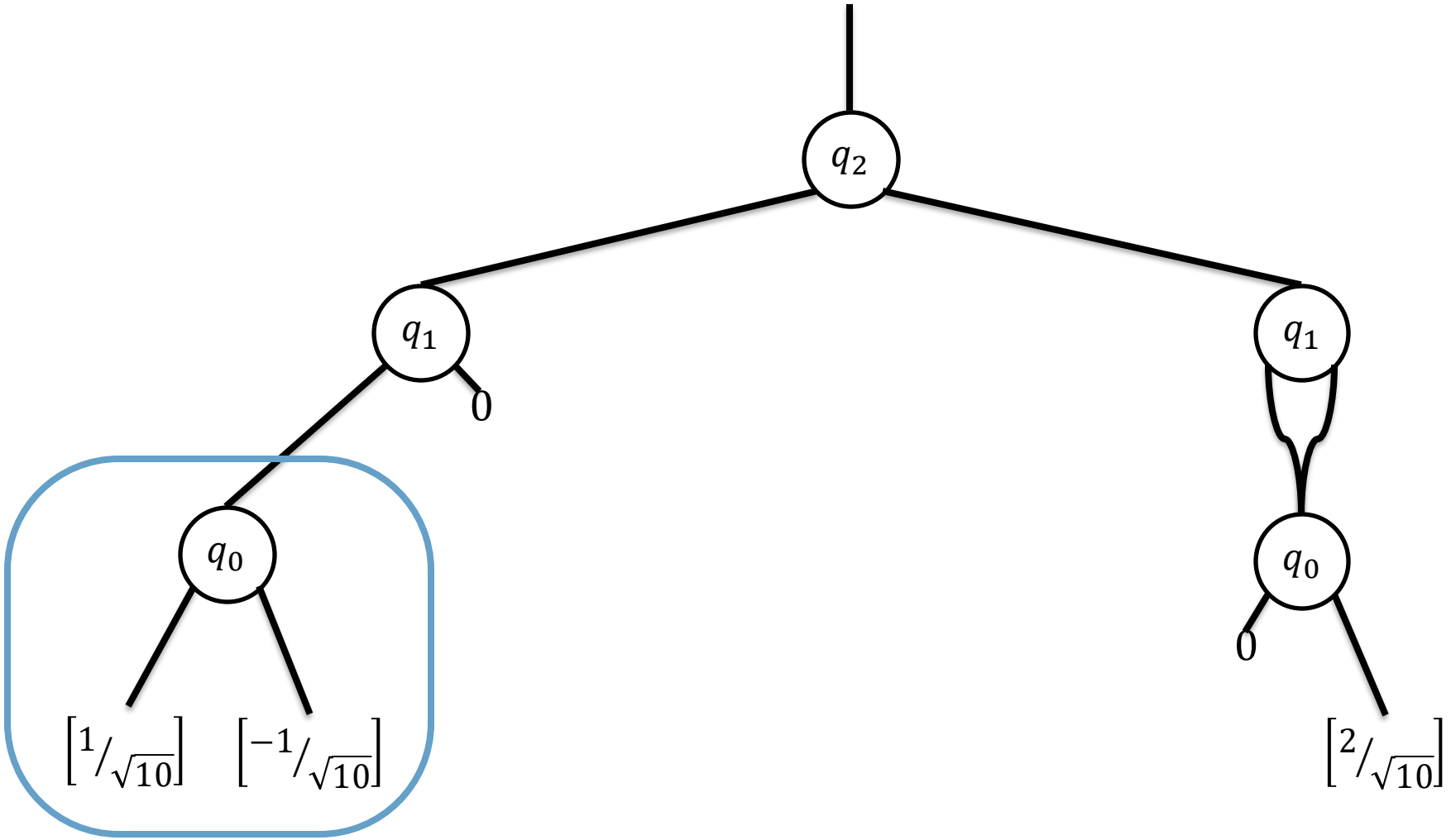


Structure + Redundancy



sparse

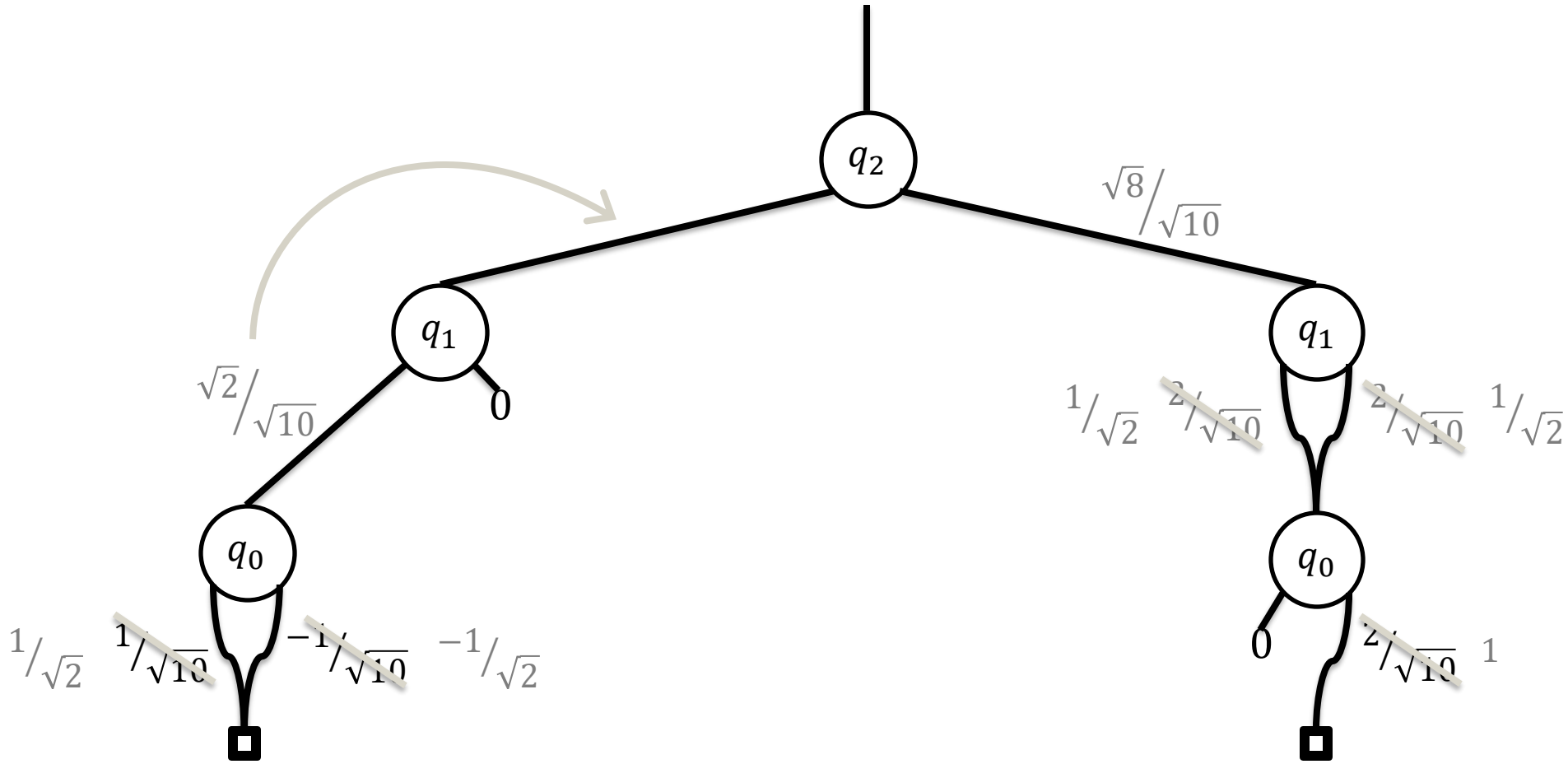
Structure + Redundancy



sparse

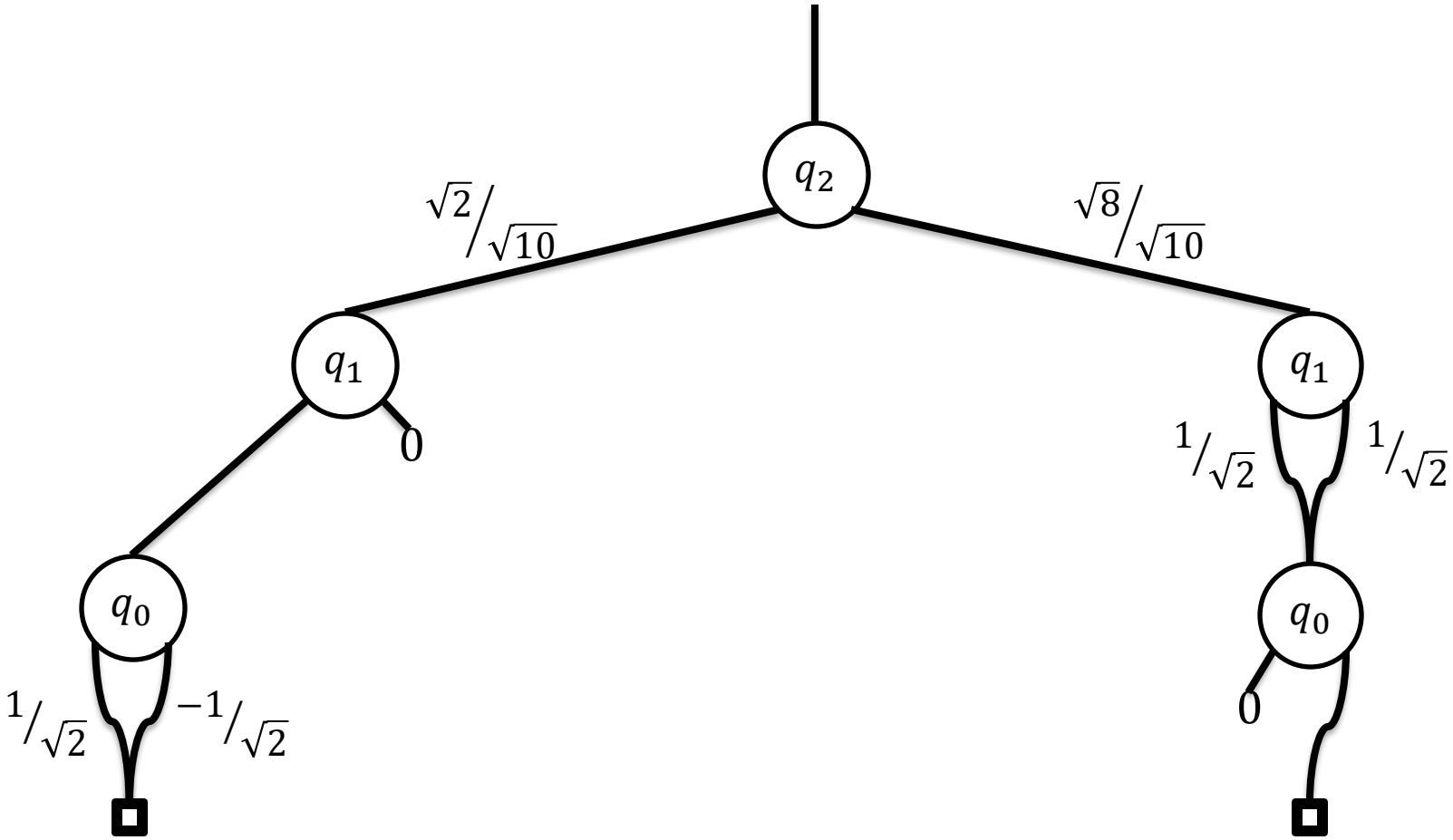
reduced

Structure + Redundancy



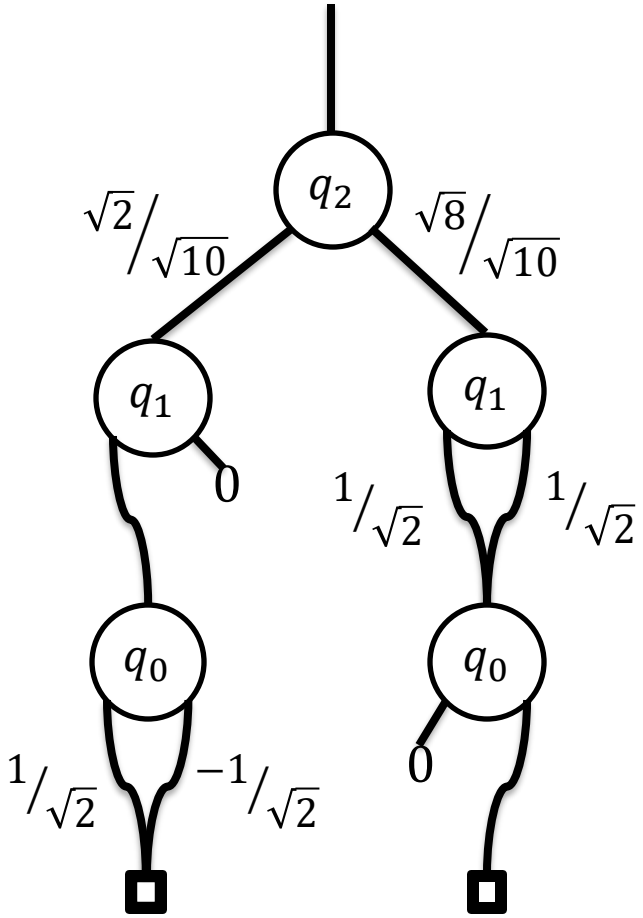
sparse **reduced**

Structure + Redundancy = Decision Diagrams



sparse **reduced** **normalized**

Structure + Redundancy = Decision Diagrams



sparse **reduced** **normalized**

Matrices

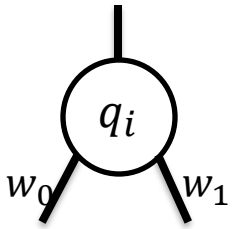
$$\left[\begin{array}{c|c} U_{00} & U_{10} \\ -- + -- \\ U_{01} & U_{11} \end{array}\right]$$

mqt-core

Multiplication

$$\left[\begin{array}{c|c} U_{00} & U_{10} \\ -- + -- \\ U_{01} & U_{11} \end{array}\right] \times \begin{bmatrix} \alpha_{0\ldots} \\ - \\ \alpha_{1\ldots} \end{bmatrix} = \begin{bmatrix} U_{00}\alpha_{0\ldots} + U_{10}\alpha_{1\ldots} \\ ---------- - \\ U_{01}\alpha_{1\ldots} + U_{11}\alpha_{1\ldots} \end{bmatrix}$$

Sampling

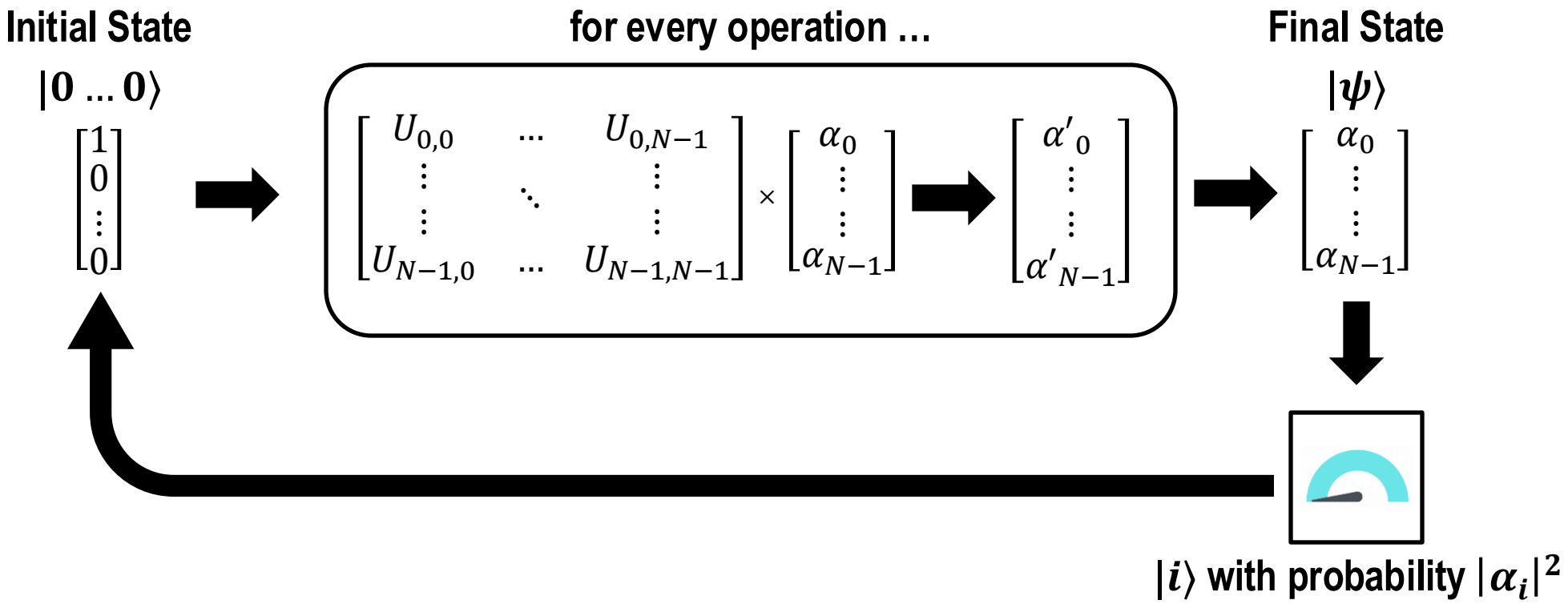


$$P(q_i = |0\rangle) = |w_0|^2$$

$$P(q_i = |1\rangle) = |w_1|^2$$

R. Wille, S. Hillmich, and L. Burgholzer.
Decision Diagrams for Quantum Computing.
In Design Automation of Quantum Computers. Springer, 2023

Classical Simulation of Quantum Circuits



MQT DDVis - A Web-based GUI for Exploring DDs

Feel Free to Follow Along

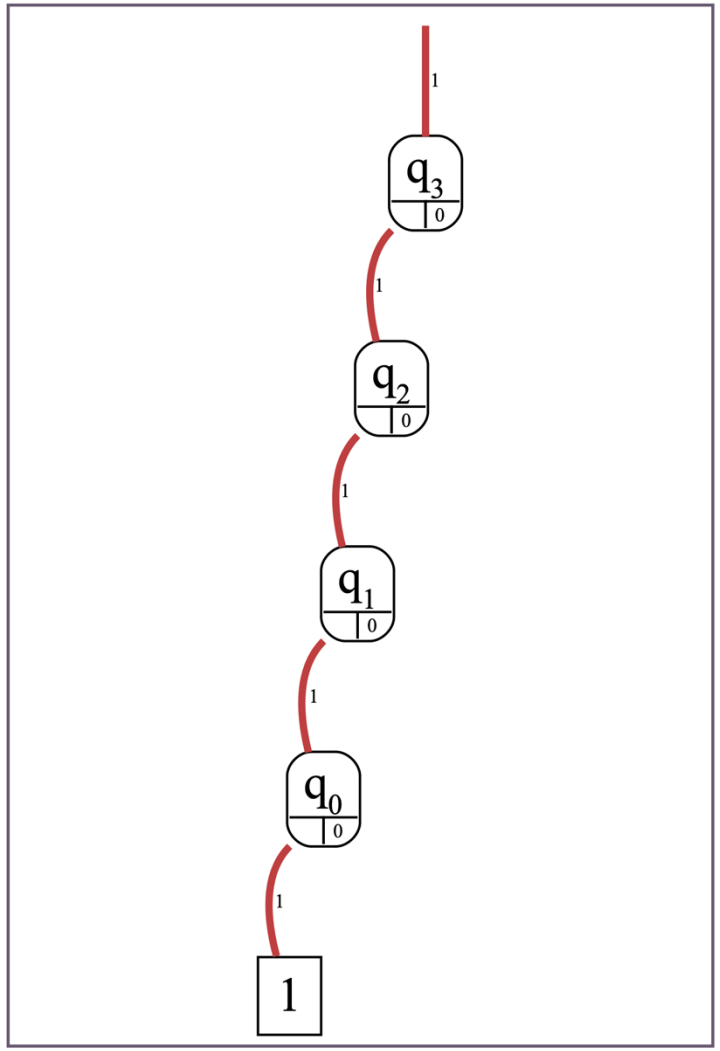
- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*

Algorithm

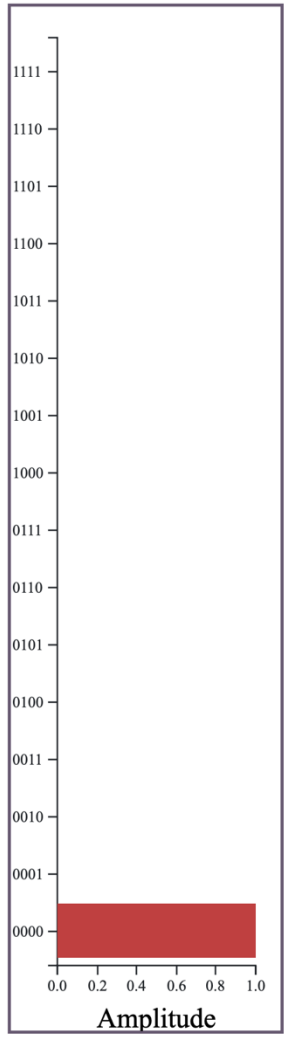
```
gate oracle q0, q1, q2, flag {
    mcphase(pi) q0, q1, q2, flag;
}
gate diffusion q0, q1, q2 {
    h q0; h q1; h q2;
    x q0; x q1; x q2;
    h q2;
    ccx q0, q1, q2;
    h q2;
    x q2; x q1; x q0;
    h q2; h q1; h q0;
}

qreg q[3];
qreg flag[1];
creg c[3];

1  h q;
2  x flag;
3  barrier q;

4  oracle q[0], q[1], q[2], flag;
5  diffusion q[0], q[1], q[2];
6  barrier q;
7  oracle q[0], q[1], q[2], flag;
8  diffusion q[0], q[1], q[2];

9  measure q -> c;
```

Go to line 0

Quantum Decision Diagram



State Vector



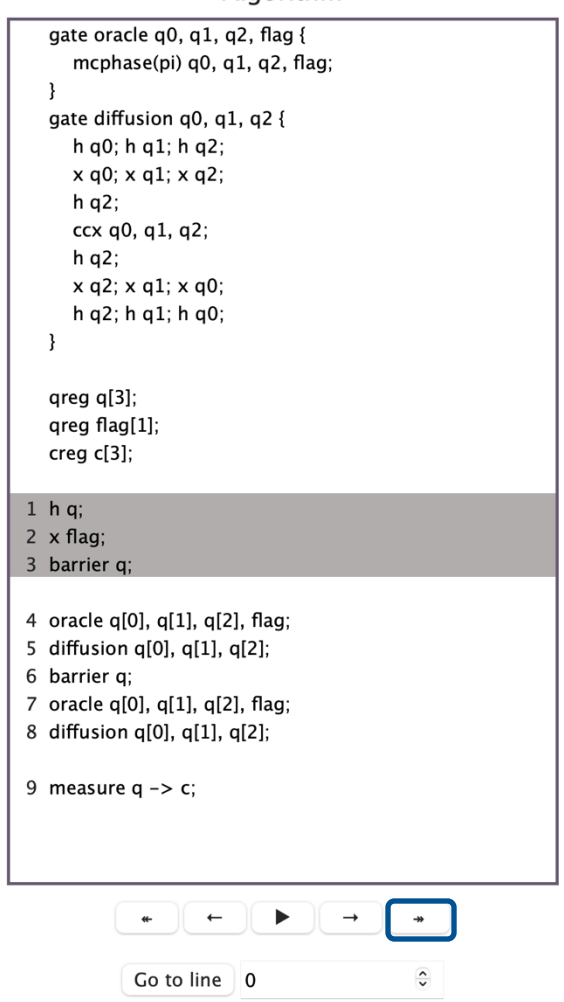
https://www.cda.cit.tum.de/app/ddvis/

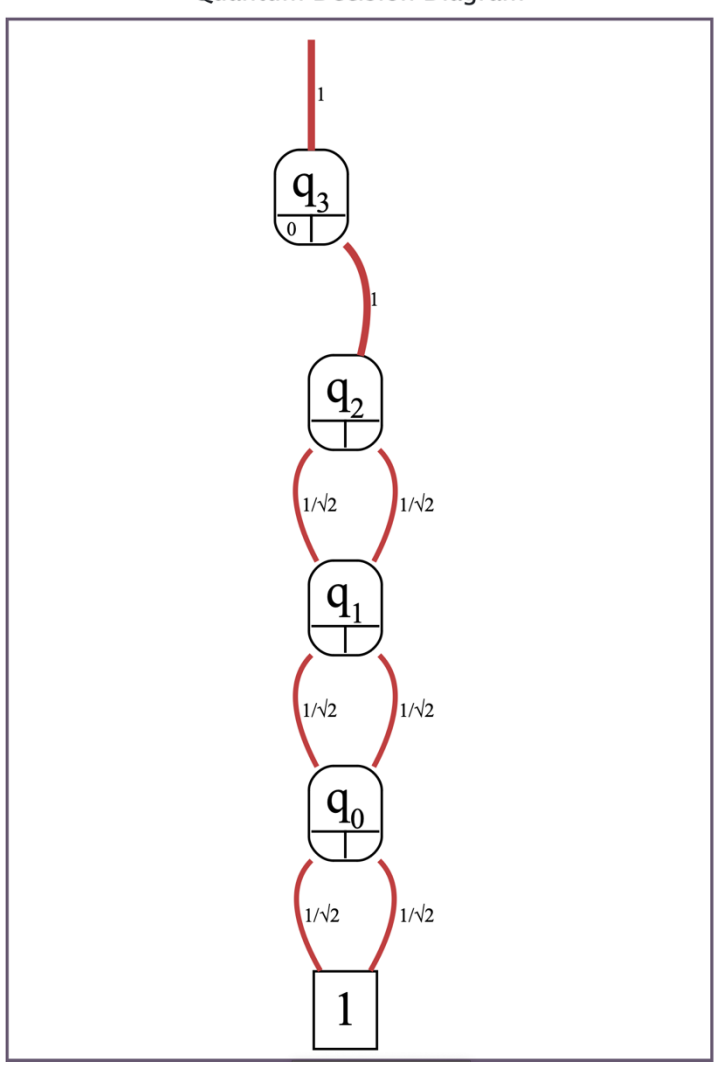
MQT DDVis - A Web-based GUI for Exploring DDs

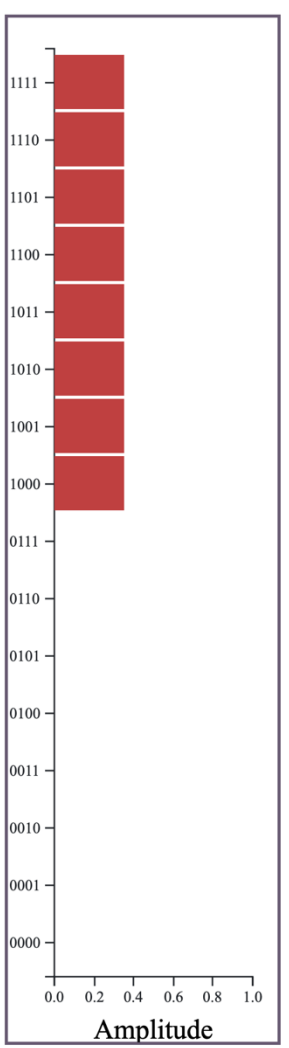
Munich Quantum Valley

Feel Free to Follow Along

- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*







https://www.cda.cit.tum.de/app/ddvis/

MQT DDVis - A Web-based GUI for Exploring DDs

Feel Free to Follow Along

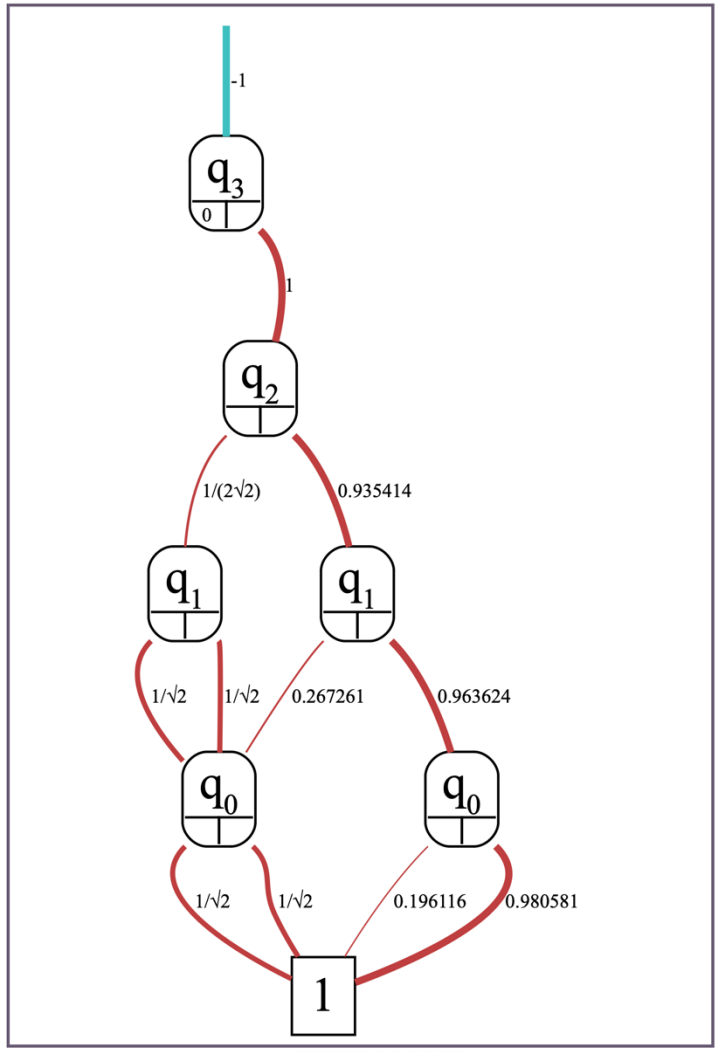
- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*

Algorithm

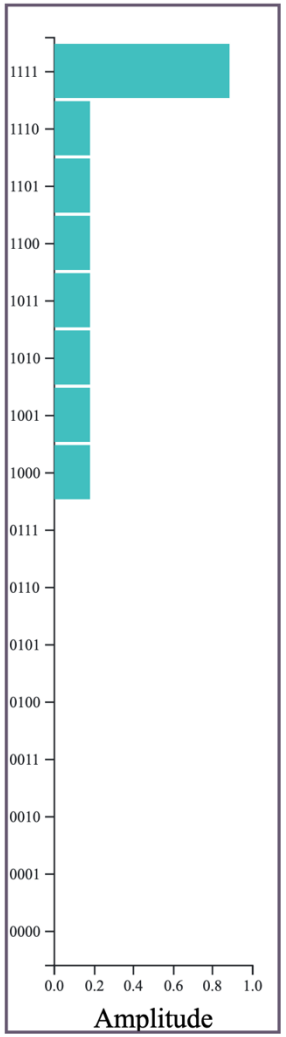
```
gate oracle q0, q1, q2, flag {
    mcphase(pi) q0, q1, q2, flag;
}
gate diffusion q0, q1, q2 {
    h q0; h q1; h q2;
    x q0; x q1; x q2;
    h q2;
    ccx q0, q1, q2;
    h q2;
    x q2; x q1; x q0;
    h q2; h q1; h q0;
}

qreg q[3];
qreg flag[1];
creg c[3];

1 h q;
2 x flag;
3 barrier q;

4 oracle q[0], q[1], q[2], flag;
5 diffusion q[0], q[1], q[2];
6 barrier q;
7 oracle q[0], q[1], q[2], flag;
8 diffusion q[0], q[1], q[2];

9 measure q -> c;
```

Go to line 0

Quantum Decision Diagram



State Vector



https://www.cda.cit.tum.de/app/ddvis/

MQT DDVis - A Web-based GUI for Exploring DDs

Munich Quantum Valley

Feel Free to Follow Along

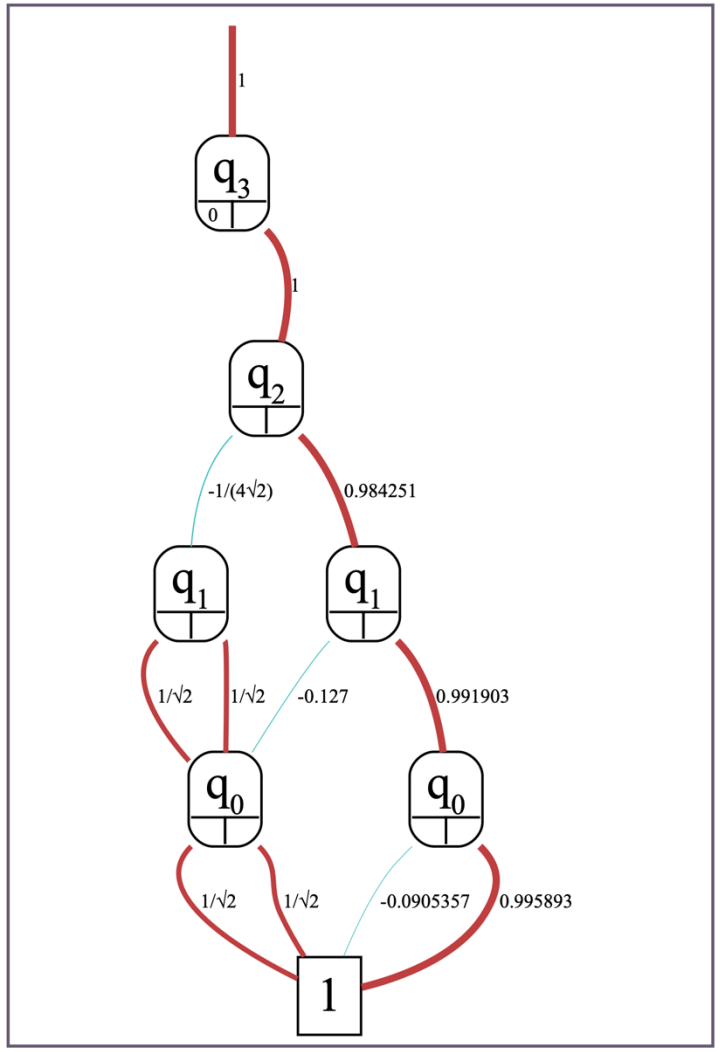
- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*

Algorithm

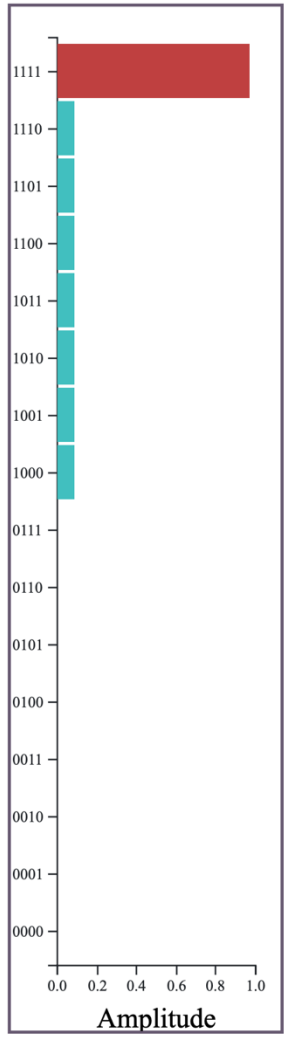
```
gate oracle q0, q1, q2, flag {
    mcphase(pi) q0, q1, q2, flag;
}
gate diffusion q0, q1, q2 {
    h q0; h q1; h q2;
    x q0; x q1; x q2;
    h q2;
    ccx q0, q1, q2;
    h q2;
    x q2; x q1; x q0;
    h q2; h q1; h q0;
}

qreg q[3];
qreg flag[1];
creg c[3];

1  h q;
2  x flag;
3  barrier q;

4  oracle q[0], q[1], q[2], flag;
5  diffusion q[0], q[1], q[2];
6  barrier q;
7  oracle q[0], q[1], q[2], flag;
8  diffusion q[0], q[1], q[2];

9  measure q -> c;
```

Go to line 0

Quantum Decision Diagram



State Vector



https://www.cda.cit.tum.de/app/ddvis/

MQT DDSIM

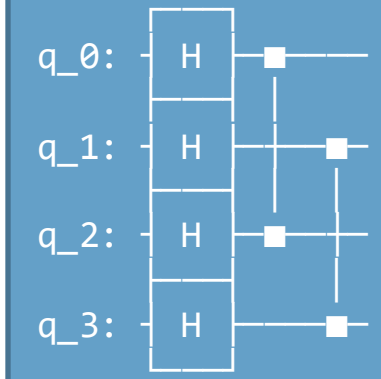
https://github.com/munich-quantum-toolkit/ddsim or simply **`pip install mqt.ddsim`**

hybrid_sim.py

```python
from qiskit import *
from mqt.ddsim import DDSIMProvider

q = QuantumRegister(4, 'q')
circ = QuantumCircuit(q)
circ.h(q)
circ.cz(q[0], q[2])
circ.cz(q[3], q[1])
print(circ.draw())

provider = DDSIMProvider()
backend = provider.get_backend('hybrid_statevector_simulator')
job = backend.run(circ, mode='amplitude') # or mode='dd'

print(job.result().get_statevector(circ))
```

Terminal

```
$ python3 hybrid_sim.py
```




```
[ 0.25+0.j  0.25+0.j  0.25+0.j  0.25+0.j
0.25+0.j -0.25+0.j  0.25+0.j -0.25+0.j
0.25+0.j  0.25+0.j -0.25+0.j -0.25+0.j
0.25+0.j -0.25+0.j -0.25+0.j  0.25+0.j]
```

*"… allowed to perform simulations **within 20 min** where the **state-of-the-art** decision diagram simulator **did not finish within a whole day.**"*

Qiskit Ecosystem

PLANQK Beta

155 ★ on GitHub, **500k Downloads** from PyPI

Practical Part I: Simulation

Setup

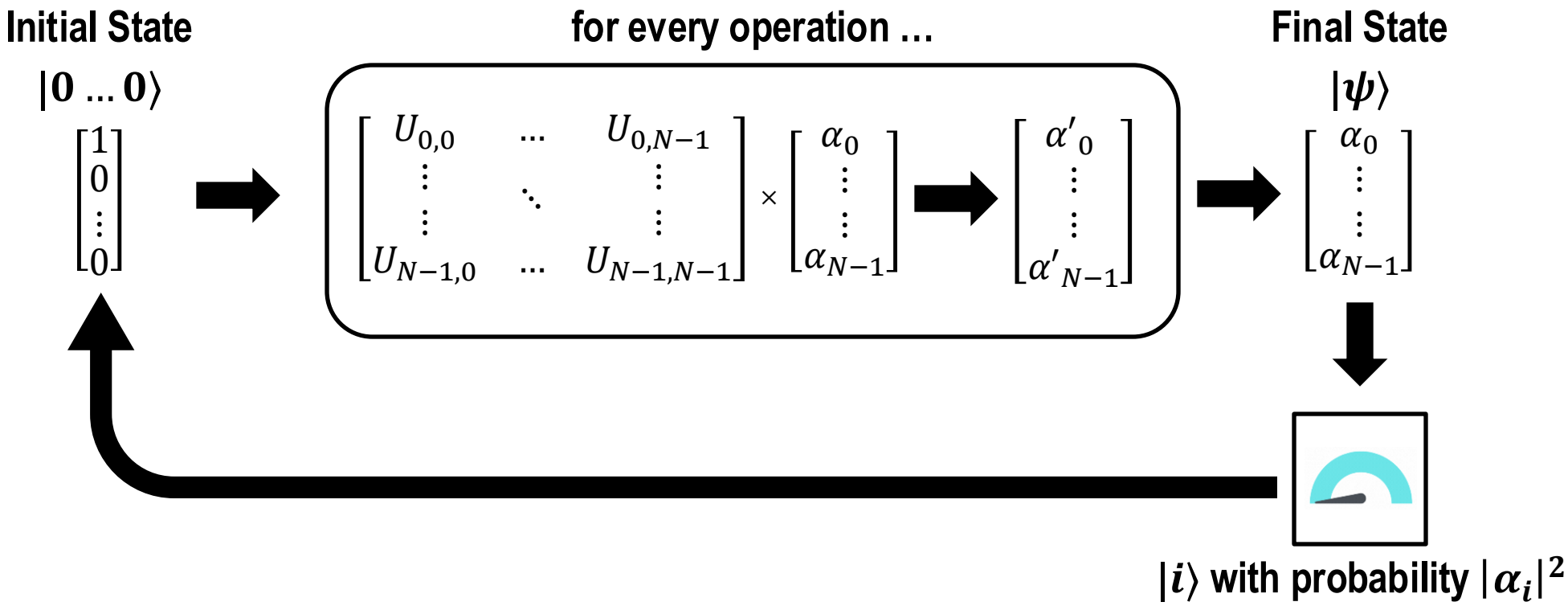
- Clone the repository
  - git clone https://github.com/munich-quantum-toolkit/.github.git
  - cd mqt

- Check out the **tutorial-SC25** branch
  - git switch tutorial-SC25

- Follow the README in the **tutorial** folder
  - cd tutorial

github.com/munich-quantum-toolkit/.github.git

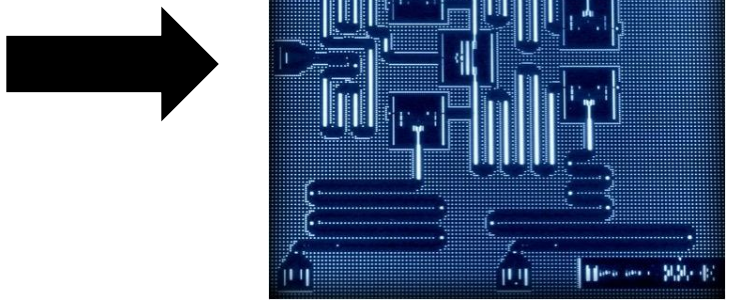
Quantum Computing



Quantum Computing

QASM

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];
h q[2];
cx q[2], q[1];
cx q[2], q[0];
measure q[2] -> c[2];
```



1110

0110

0001

1001

Classical Computing

ASM

```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

1110
0110
0001
1001

Classical Computing

C++

```
int i = 1;
int j = 1;
while (true) {
  *val++ = i + j;
  j = i + (i = j);
}
```

Compiler

gcc
clang
icc

ASM

```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

1110
0110
0001
1001

Quantum Computing

Python

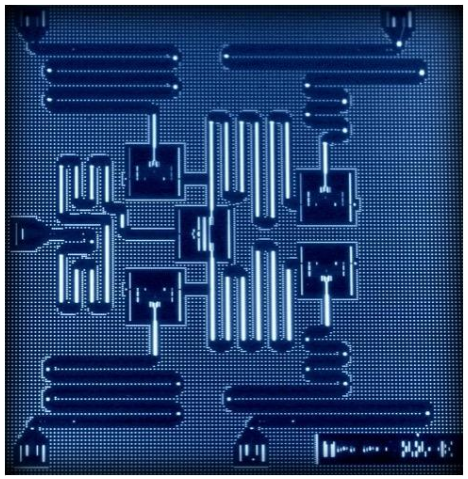
```
from qiskit import ↩
QuantumCircuit

qc = ↩
QuantumCircuit(3, 3)
qc.h(2)
qc.cx(2, 1)
qc.cx(2, 0)
qc.measure(2, 2)
```

Compiler

qiskit
tket
…

QASM

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];
h q[2];
cx q[2], q[1];
swap q[2], q[1];
cx q[1], q[0];
measure q[1] -> c[2];
```



1110
0110
0001
1001

Compilation of Quantum Circuits

Conceptional algorithm

Limited Gate Set

Synthesis

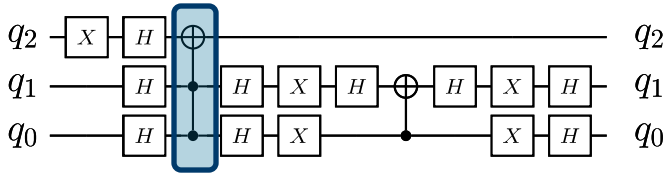
Limited Connectivity

Mapping

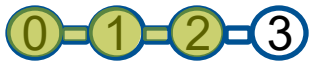
Limited Fidelity and Coherence

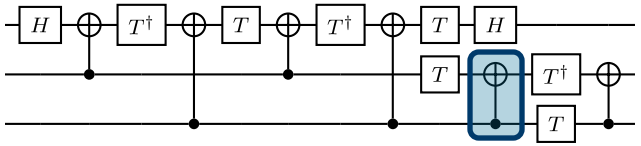
Optimizations

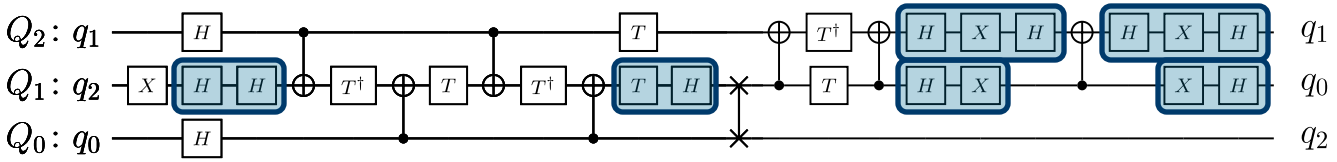
Actual Realization

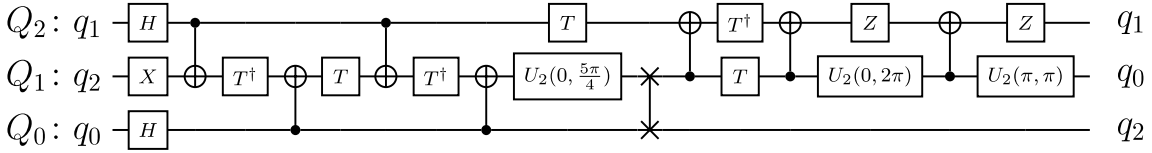


Target Device









Compilation of Quantum Circuits

Conceptional algorithm

Limited Gate Set

Synthesis

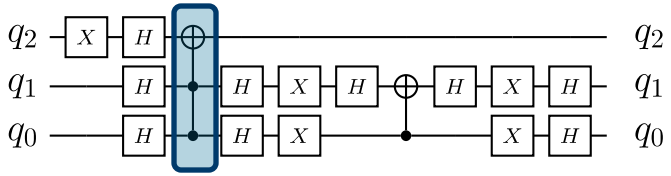
Limited Connectivity

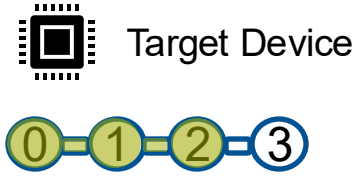
Mapping

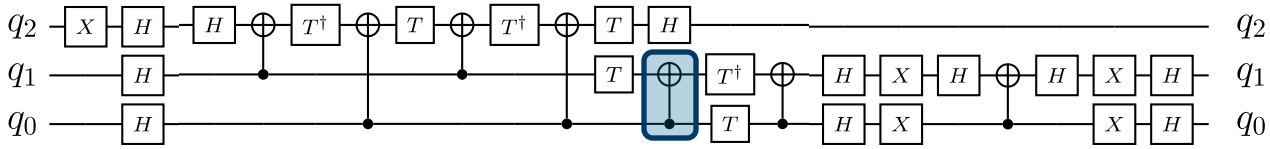
Limited Fidelity and Coherence

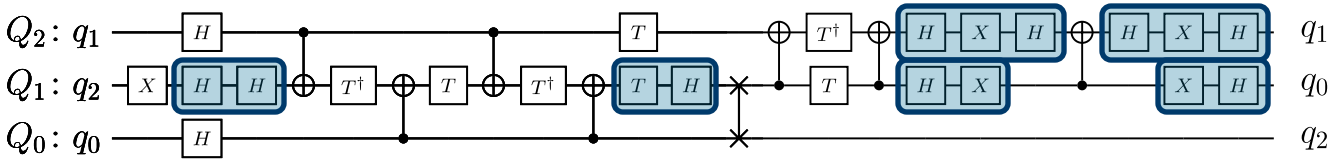
Optimizations

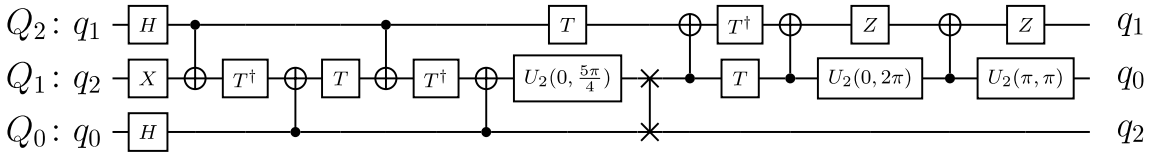
Actual Realization



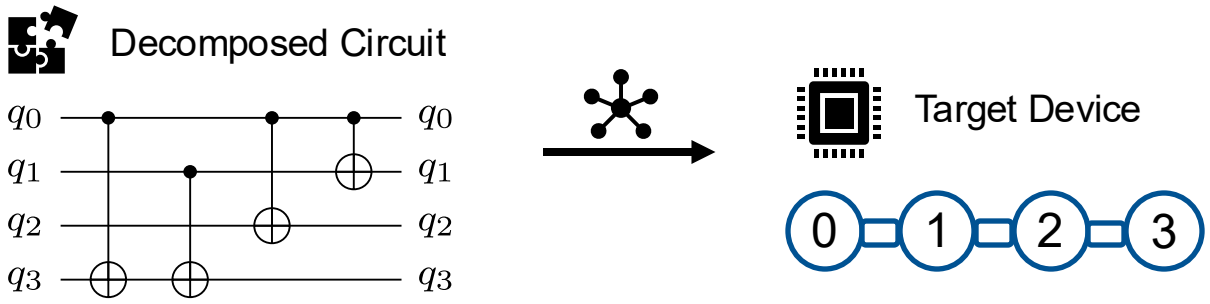






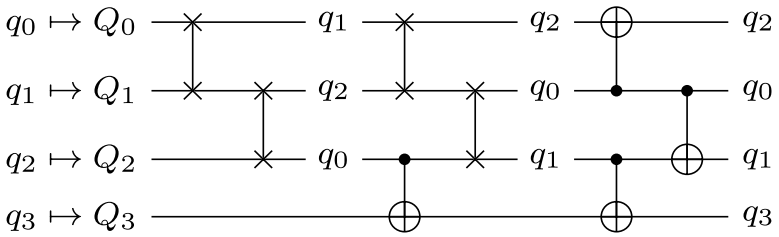


Mapping of Quantum Circuits

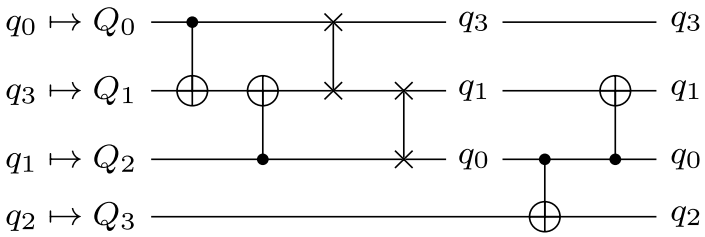


- Different approaches for mapping offer trade-off between runtime vs quality of result

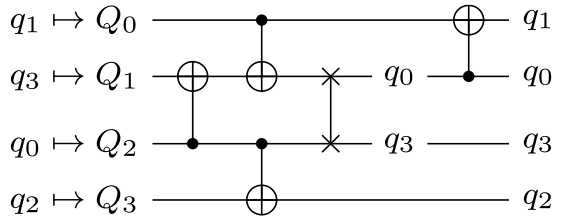
Naive Approach



Heuristic Approach



Optimal Approach



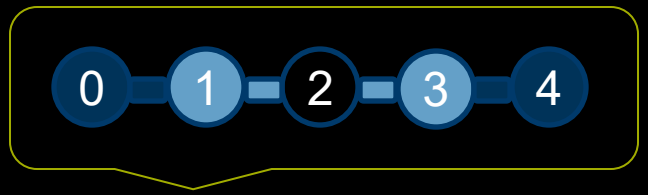
Finding an optimal solution is NP-complete!

MQT QMAP

https://github.com/munich-quantum-toolkit/qmap or simply **`pip install mqt.qmap`**

map.py

```python
from qiskit import *
from qiskit.providers.fake_provider import GenericBackendV2
from mqt import qmap

q = QuantumRegister(4, 'q')
circ = QuantumCircuit(q)
circ.cx(q[0], q[3])
circ.cx(q[1], q[3])
circ.cx(q[0], q[2])
circ.cx(q[0], q[1])
print(circ.draw())

arch = GenericBackendV2(num_qubits=5, coupling_map=
[[0, 1], [1, 0], [1, 2], [2, 1], [2, 3], [3, 2], [3, 4], [4, 3]])
m_circ, results = qmap.compile(circ, arch=arch, method='exact')

print(m_circ.draw())
```



Terminal

```
$ python3 map.py
q_0: ──■──────■──┤X├
q_1: ──┼──■───┼──┤X├
q_2: ──┼──┼──┤X├──┼──
q_3: ┤X├─┤X├──┼──┼──

q_0: ──■──┤X├
q_1: ┤X├─┤X├─X──■──
q_2: ──■──■──X─────
q_3: ─────┤X├──────
q_4: ──────────────
```

"… one of the **first optimal solutions** to the problem of mapping quantum circuits to architectures with limited connectivity … showed that **circuits mapped by Qiskit** required **twice as many additional gates** as necessary."

Qiskit Ecosystem

PLANQK Beta

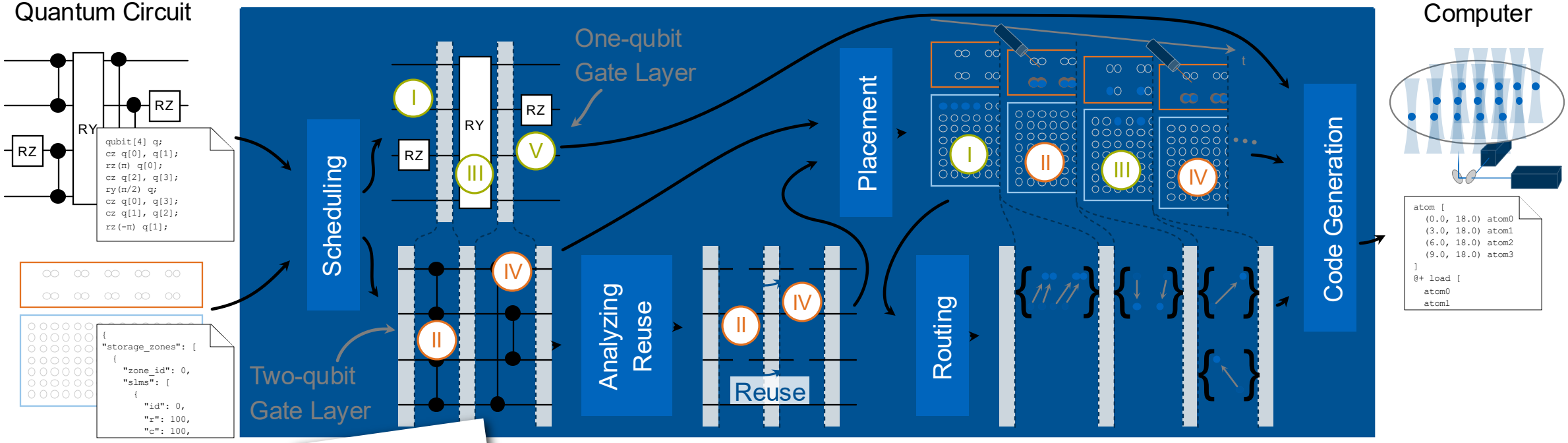
126 ★ on GitHub, **528k Downloads** from PyPI

Compilation for Next Generation Hardware

Quantum Circuit

Compiler

Neutral Atom Quantum Computer



Architecture Specification

Read the Article on arXiv
arxiv.org/abs/2505.22715

IEEE/ACM ICCAD 2025 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN 44th Edition

Try It Out

Python

```
>>> from mqt.qmap.na.zoned import *
>>> arch = ZonedNeutralAtomArchitecture.from_json_file(\
"arch.json")
>>> compiler = RoutingAwareCompiler(arch)
>>> from mqt.core import load
>>> circ = load(your_qiskit_circuit)
>>> code = compiler.compile(circ)
>>> print(code)
atom (0.000, 57.000) atom0
atom (3.000, 57.000) atom1
...
```

MQT QMAP

126 ★ 528k

Shell

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

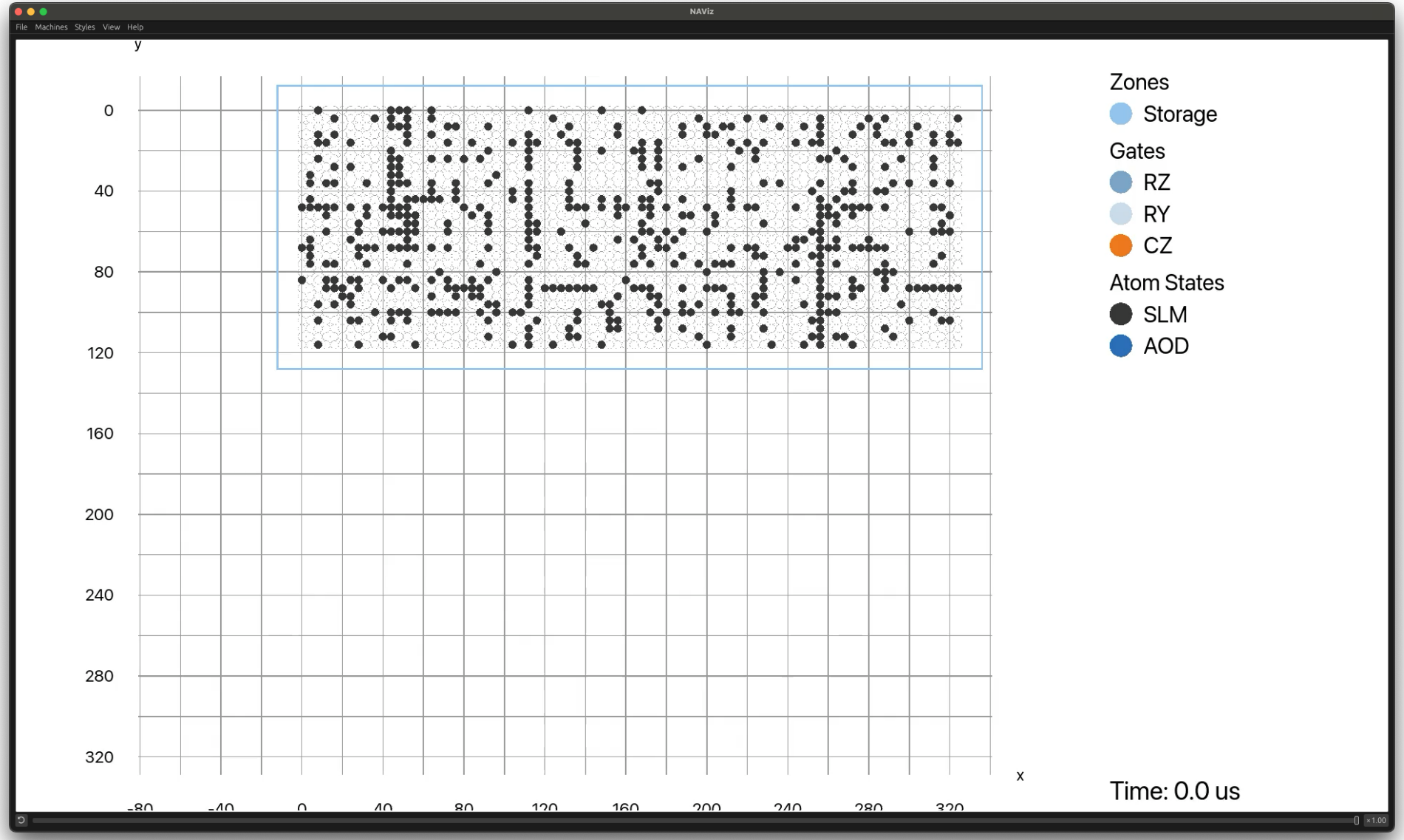
Qiskit Ecosystem

PLANQK

MQT QMAP Documentation with Example
mqt.readthedocs.io/projects/qmap/en/latest/

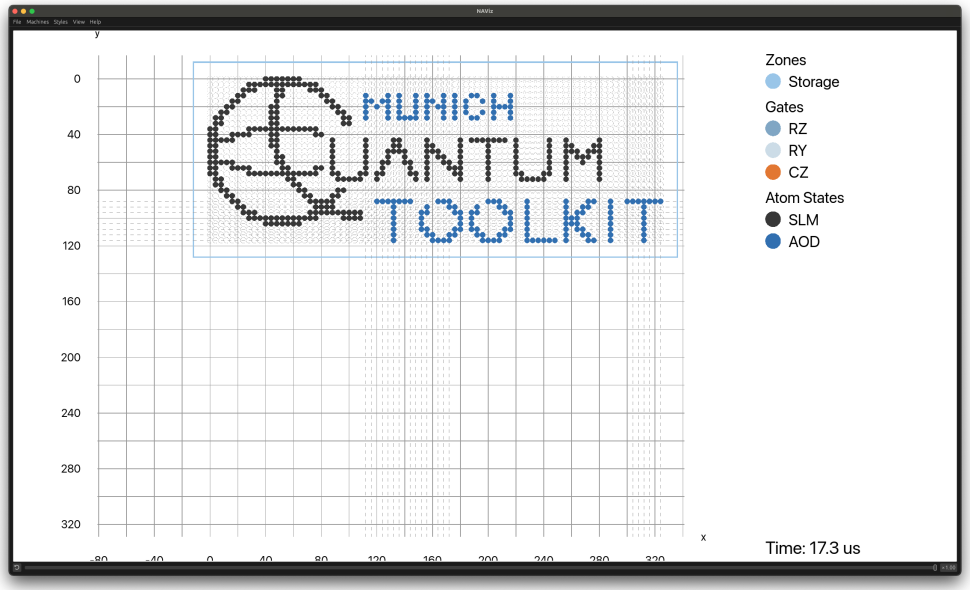
MQT QMAP GitHub
github.com/munich-quantum-toolkit/qmap

Animate the Output



MQT NAViz

- **Instant Playback**
- **Video Export**
- **Scrubbable Timeline**



Check out the Code
github.com/munich-quantum-toolkit/naviz

Read the Docs
mqt.readthedocs.io/projects/naviz/en/latest/

Practical Part II: Compilation

Quantum Computing

Python

```
from qiskit import ↩
Quantu
qc = ↩
Quantu
qc.h(2)
qc.cx(2
qc.cx(2, 0)
qc.measure(2, 2)
```

QASM

```
OPENQASM 2.0;
swap q[2], q[1];
cx q[1], q[0];
measure q[1] -> c[2];
```

1110
0110
0001
1001

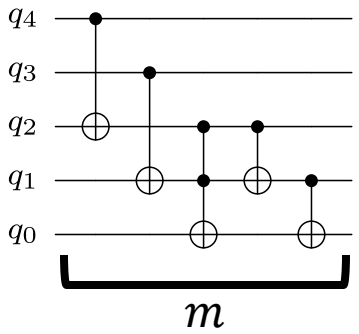
"...a **significant fraction** of these bugs (**39.9%**) are **quantum-specific**… bugs occur particularly often in components that **represent, compile, and optimize** quantum programming abstractions."

Paltenghi, M. and Pradel, M., "Bugs in Quantum Computing Platforms: An Empirical Study", *arXiv: 2110.14560*, 2021.

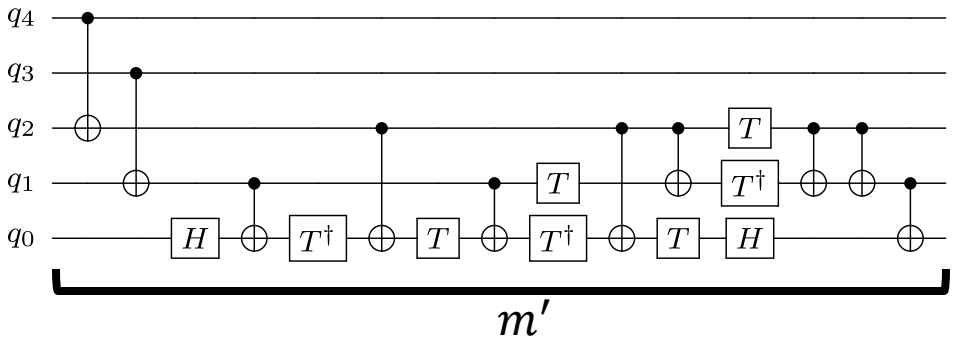
How do you verify that the resulting circuit still realizes the originally intended functionality?

Verification of Quantum Circuits

Conceptional algorithm G



Actual Realization G'

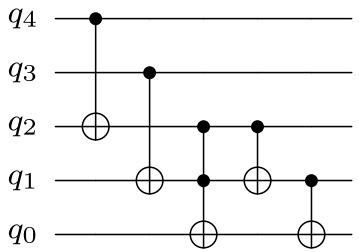


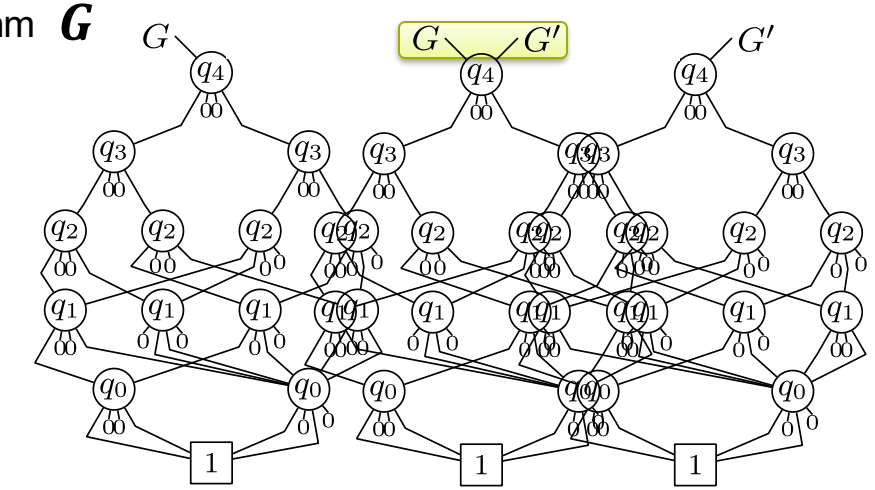
How do you verify that the resulting circuit still realizes the originally intended functionality?

$$[U_{m-1}\cdots U_0] \stackrel{?}{=} [U'_{m'-1}\cdots U'_0]$$

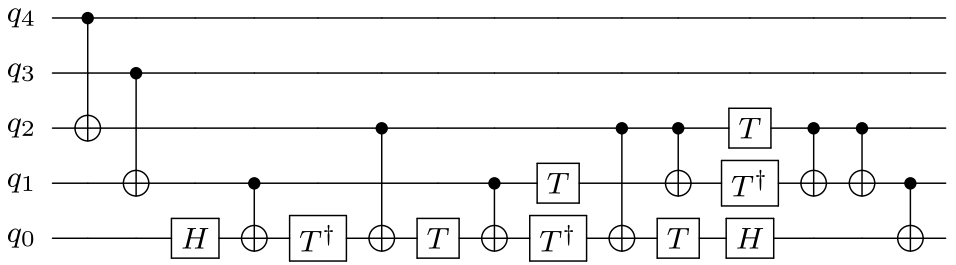
Verification of Quantum Circuits

Conceptional algorithm G





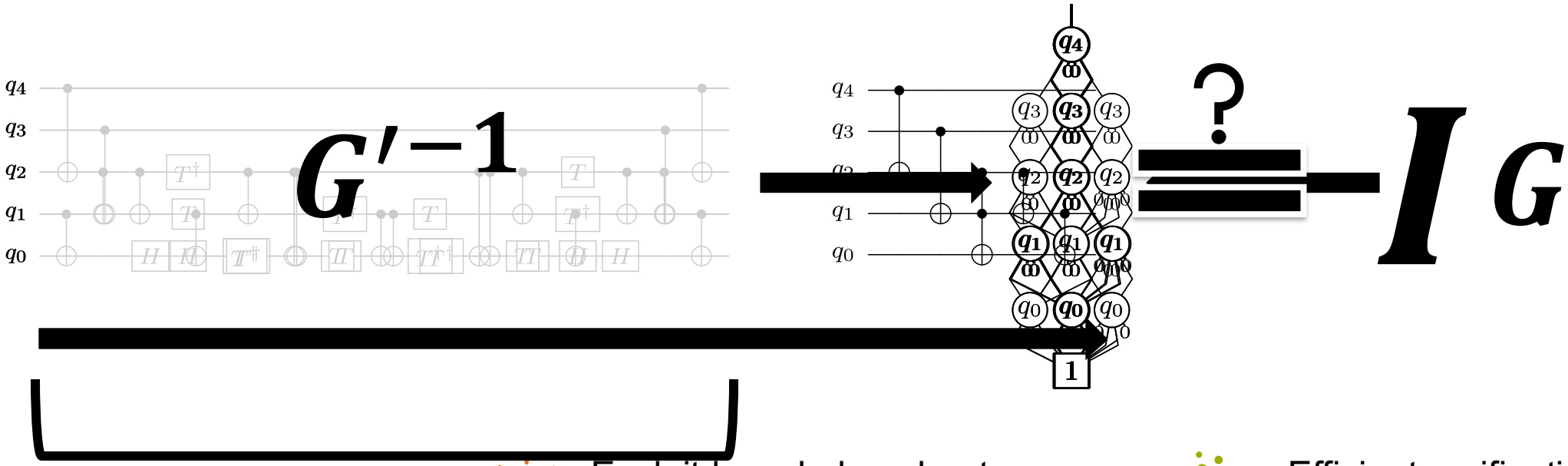
Actual Realization G'



Construction of two large DDs!

Verification of Quantum Circuits

- If $G \equiv G'$, *then* $G'^{-1} \cdot G \equiv I$
- G'^{-1} easy due to reversibility



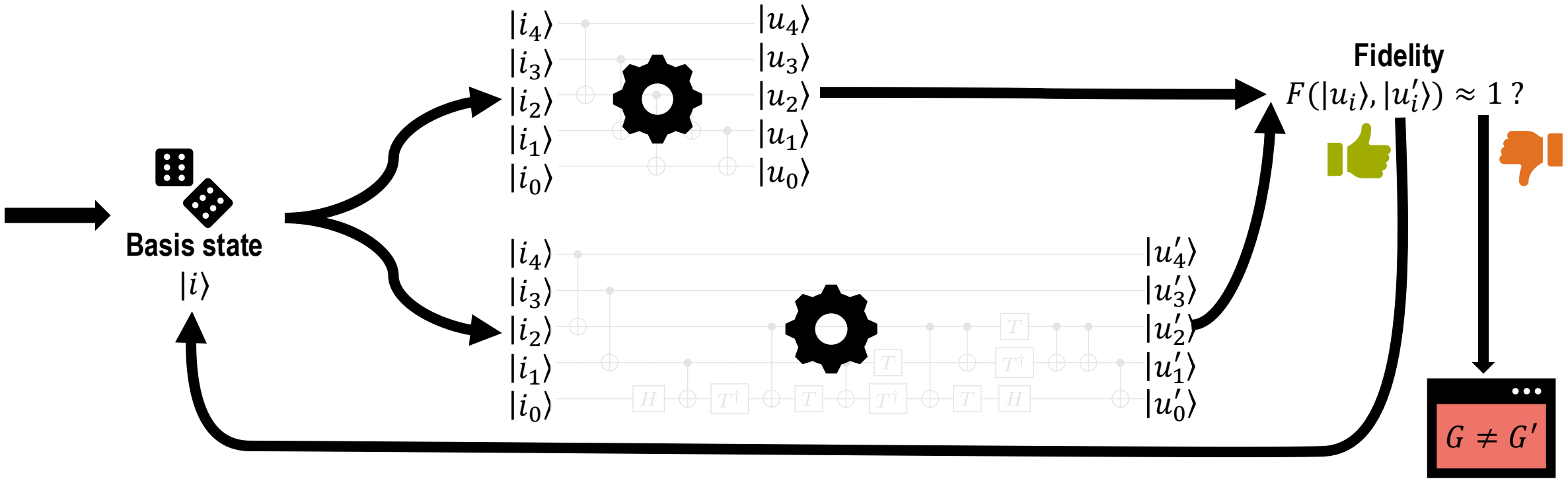
Strategies?

Complete DD for G'^{-1}

Exploit knowledge about relation between G and G'

Efficient verification of compilation results

Verification of Quantum Circuits



How likely is 👎 ?

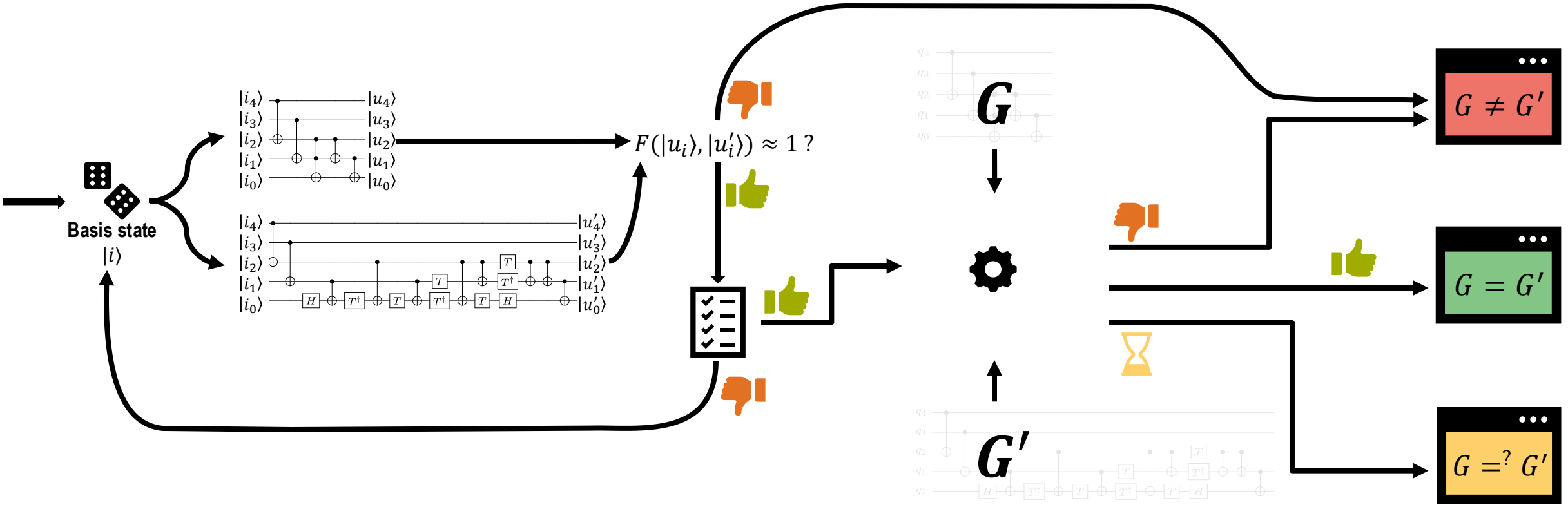
Reversibility reduces masking effects drastically

Slightest changes frequently affect entire functionality

L. Burgholzer and R. Wille. The Power of Simulation for Equivalence Checking in Quantum Computing. In Design Automation Conference (DAC), 2020

L. Burgholzer, R. Kueng, and R. Wille. Random Stimuli Generation for the Verification of Quantum Circuits. In Asia and South Pacific Design Automation Conference (ASP-DAC), 2021

Verification of Quantum Circuits



Simulations provide a highly probable estimate even when reference methods yield no conclusive answer!

L. Burgholzer and R. Wille. Advanced Equivalence Checking for Quantum Circuits.
IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD), 2021

MQT QCEC

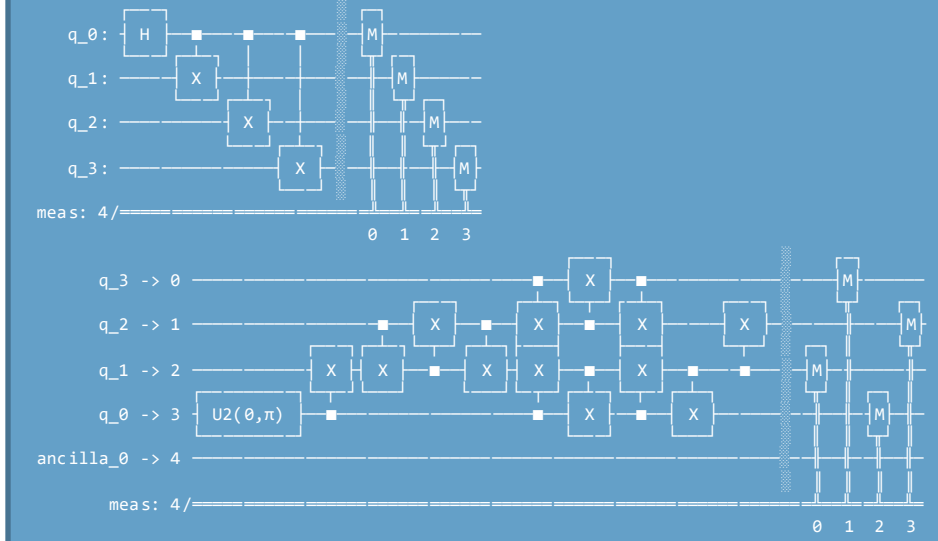
github.com/cda-tum/mqt-qcec or simply **`pip install mqt.qcec`**

ghz_4.py

```python
from qiskit import *
from qiskit.providers.fakeprovider import GenericBackendV2
from mqt import qcec

circ = QuantumCircuit(4)
circ.h(0)
circ.cx(0, 1)
circ.cx(0, 2)
circ.cx(0, 3)
circ.measure_all()
print(circ.draw())

backend = GenericBackendV2(num_qubits=5, coupling_map=
[[0, 1], [1, 0], [1, 2], [2, 1], [2, 3], [3, 2], [3, 4], [4, 3]])

m_circ = transpile(circ, backend=backend, optimization_level=2)
print(m_circ.draw())

result = qcec.verify_compilation(circ, m_circ, 2)
print(result.equivalence)
```

Terminal

```
$ python3 ghz_4.py
```




```
Equivalence.equivalent
```

"… an important step towards **avoiding a situation** where we have **powerful quantum computers**, but **no means to design and/or verify** suitable applications for them."

MUNICH QUANTUM TOOLKIT

PlanQK

105 ★ on GitHub, **822k Downloads** from PyPI

Practical Part III: Verification

Benchmarking

Munich Quantum Valley

Software Tool Evaluation

Measure compiler performance, validate optimization passes, stress-test simulators, and more.

Hardware Performance

Benchmark real quantum devices on diverse applications—compare noise resilience, fidelity and execution speed.

AI Training Datasets

Access a library of scalable quantum circuits to power machine learning models and data-driven optimizations.

MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

103 ★

112k

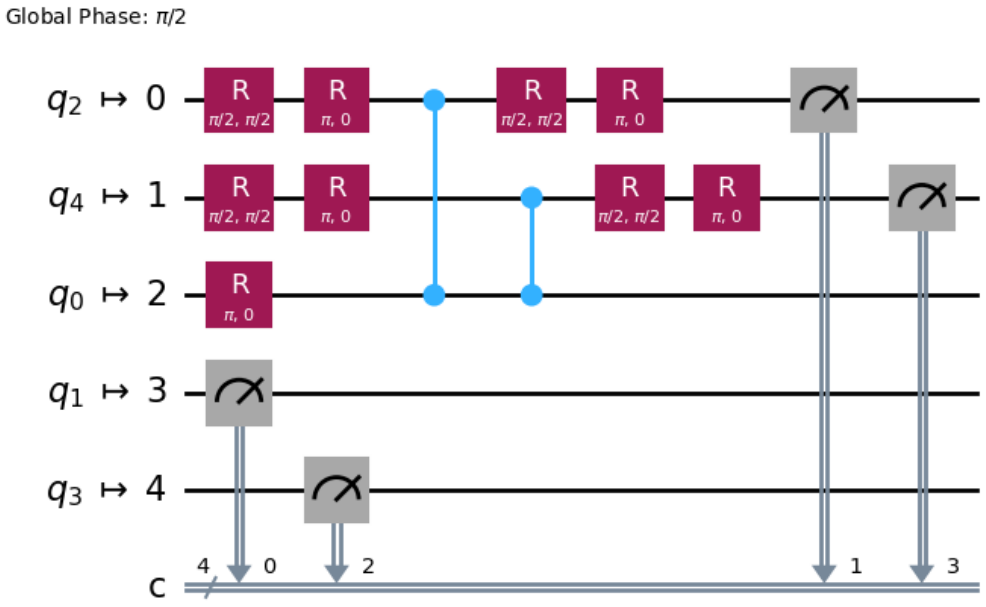
Qiskit Ecosystem

PLANQK76

Benchmarking: Software Tool Evaluation

optimizer_eval.py

```python
from mqt.bench import BenchmarkLevel, get_benchmark
from mqt.bench.targets import get_device
from your.package import awesome_optimizer

# Bernstein-Vazirani circuit on the algorithmic level
circuit = get_benchmark(benchmark="bv", level=BenchmarkLevel.ALG, circuit_size=5)
circuit.draw()

# Optimize using your developed algorithm
optimized_circuit = awesome_optimizer(circuit)
optimized_circuit.draw()

# Map to IQM Crystal (5 Qubits) to evaluate
mapped_circuit = get_benchmark(
    benchmark=optimized_circuit,
    level=BenchmarkLevel.MAPPED,
    circuit_size=5,
    target=get_device("iqm_crystal_5"),
)
mapped_circuit.draw()
```



MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

103 ★

112k

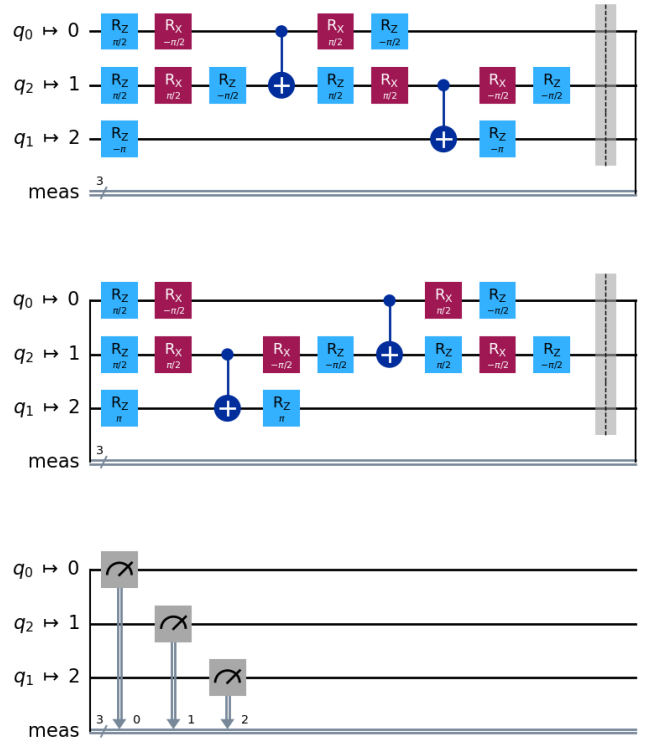
Qiskit Ecosystem

PLANQK

Benchmarking: Hardware Performance

hardware_performance.py

```python
from mqt.bench import BenchmarkLevel, get_benchmark
from qiskit.transpiler import InstructionProperties, Target
from qiskit.circuit import Parameter
from qiskit.circuit.library import RXGate, RZGate

# Custom target
custom_target = Target(num_qubits=3, description="custom_target")
# Single qubit properties
single_qubit_props = InstructionProperties(duration=1e-3, error=1e-4)
properties = {(0,): single_qubit_props, (1,): single_qubit_props,
              (2,): single_qubit_props}
custom_target.add_instruction(RXGate(Parameter("alpha")), properties=properties)
custom_target.add_instruction(RZGate(Parameter("beta")), properties=properties)
# Add two qubit properties, measurement etc.
...

# Map Deutsch-Jozsa to custom target and append mirrored version
mapped_circuit = get_benchmark("dj", BenchmarkLevel.MAPPED, circuit_size=3,
    opt_level=3, target=custom_target, generate_mirror_circuit=True)
mapped_circuit.draw()
```



MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

103 ★

112k

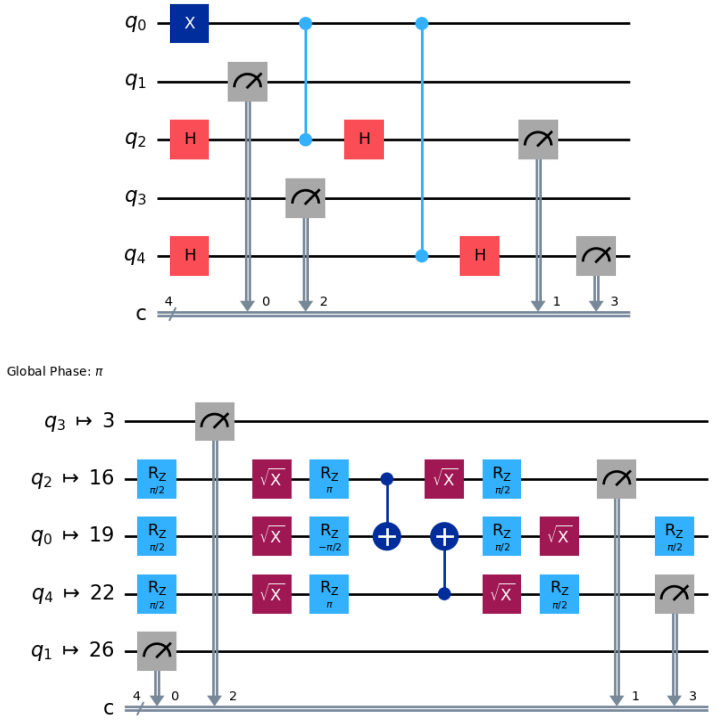
Qiskit Ecosystem

PLANQK78

Benchmarking: AI Training Datasets

ai_training.py

```python
from mqt.bench import BenchmarkLevel, get_benchmark
from mqt.predictor import qcompile

# Get uncompiled Bernstein-Vazirani circuit
qc = get_benchmark("bv", level=BenchmarkLevel.INDEP, circuit_size=5)
qc.draw()

# Compile using MQT Predictor (which has been trained using circuits from MQT Bench)
qc_compiled, compilation_information, quantum_device = qcompile(qc)

# Predicted device
print(quantum_device)    # IBM Falcon (27 Qubits)

# Print compiled circuit
qc_compiled.draw()
```



MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

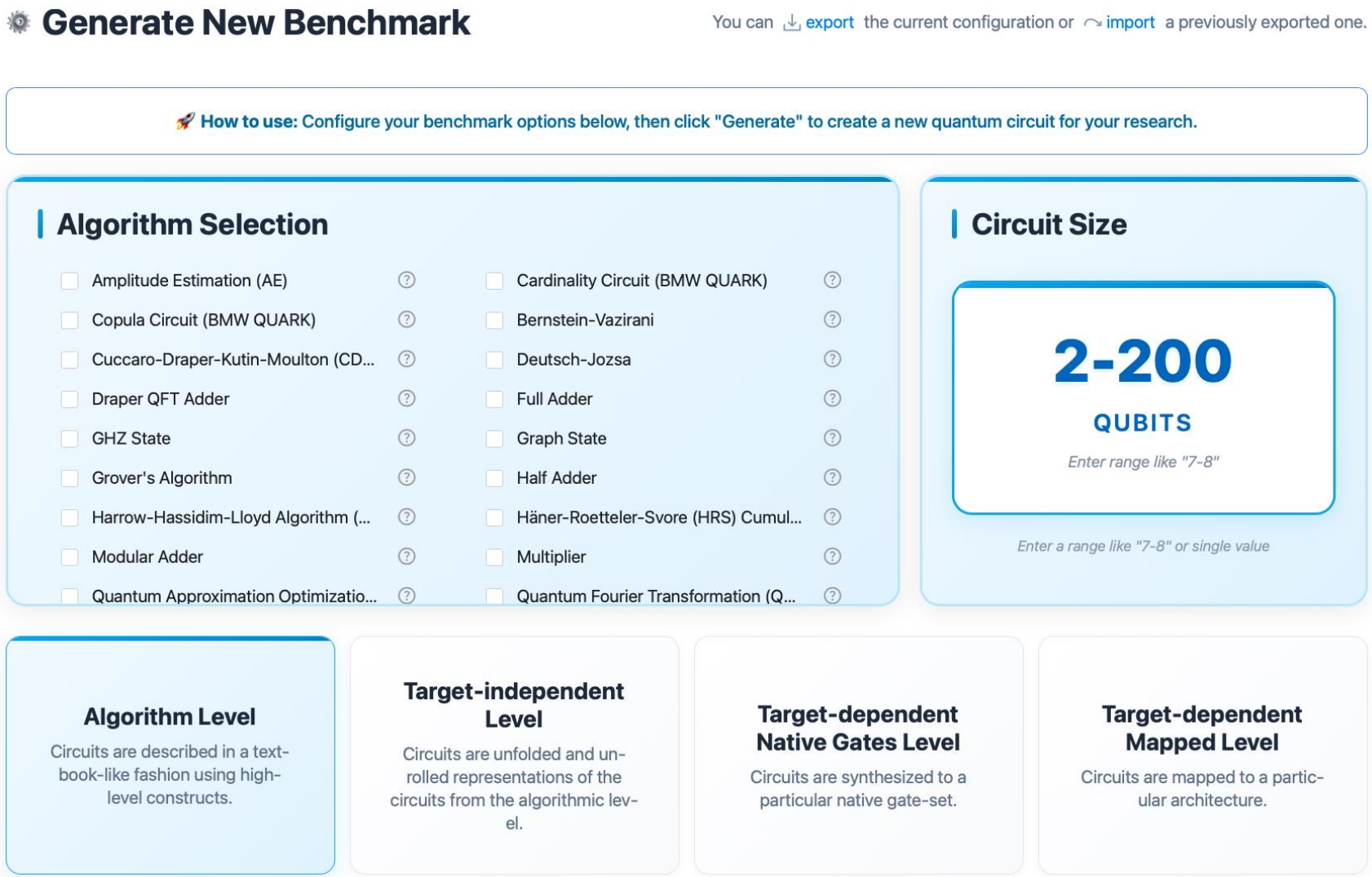
103 ★

112k

Qiskit Ecosystem

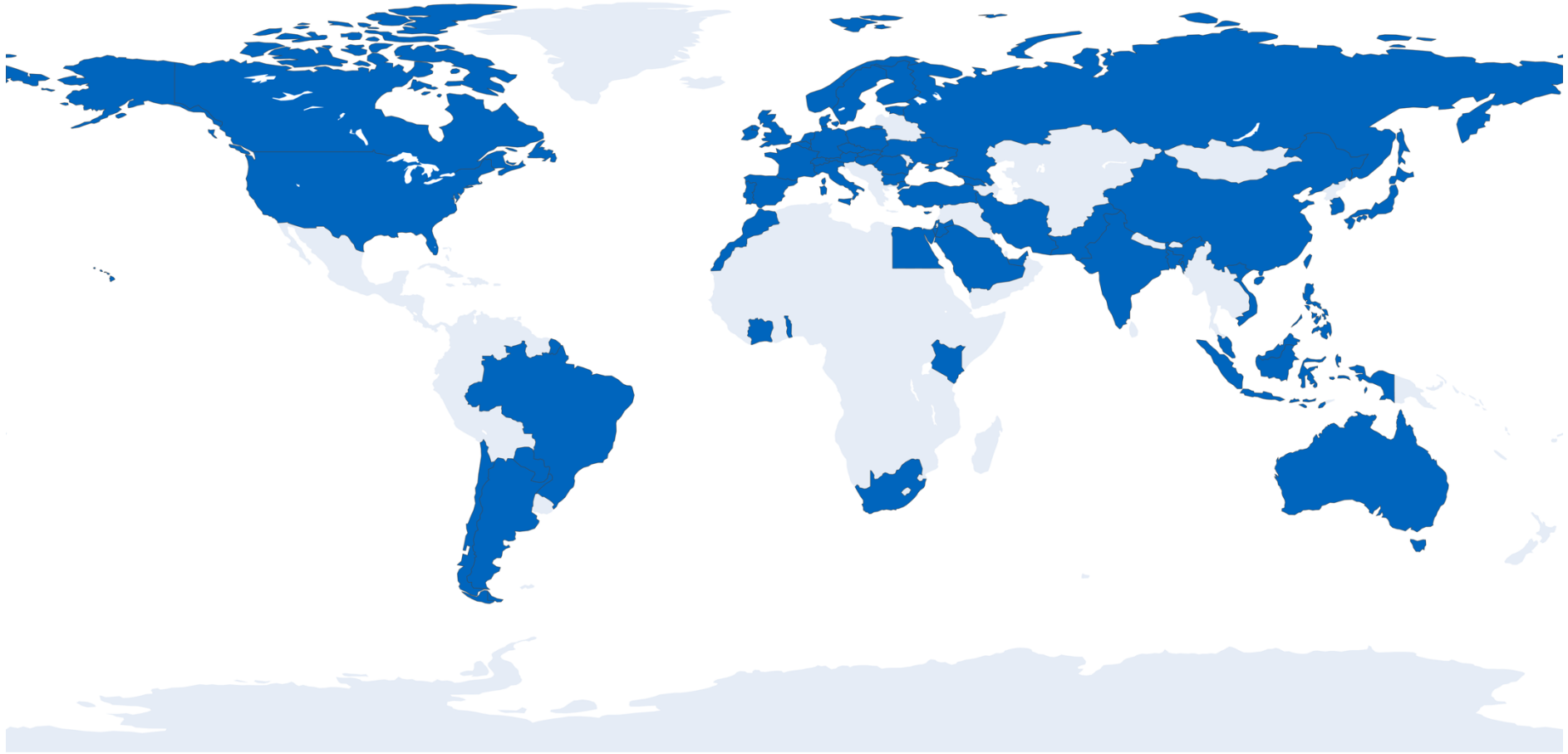
PLANQK79

MQT Bench



Mqt-bench.app

MQT Bench

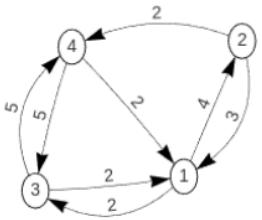


Mqt-bench.app

Automated QUBO encoding

MQT QUBOMaker: Pathfinder

Legal Information



Help | Change Graph | Generate

Encoding

One-Hot | Domain Wall | Binary

Number of Paths: 1

Max Path Length: 4

Path is Loop | Minimize Weight | Maximize Weight

The following vertices must appear exactly once in each path

ALL | 1 | 2 | 3 | 4

The following vertices must appear at least once in each path

ALL | 1 | 2 | 3 | 4

The following vertices must appear at most once in each path

ALL | 1 | 2 | 3 | 4

The following paths may not share vertices

ALL | 1

The following paths may not share edges

ALL | 1

Vertex positions

Path 1 position 1 is 3 Add

Path 1 position 1 is: 3

The following edges must appear exactly once in each path

2 → 1 Add

2 → 1

The following edges must appear at least once in each path

1 → 1 Add

The following edges must appear at most once in each path

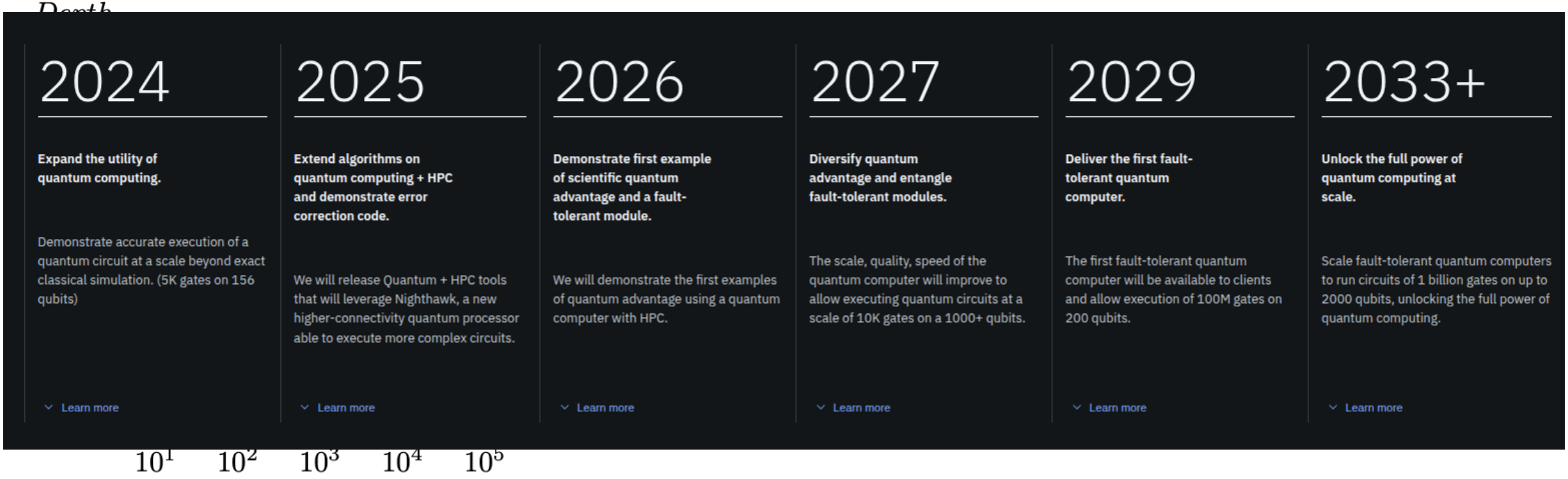
1 → 1 Add

The following precedence constraints must be fulfilled

4 → 1 Add

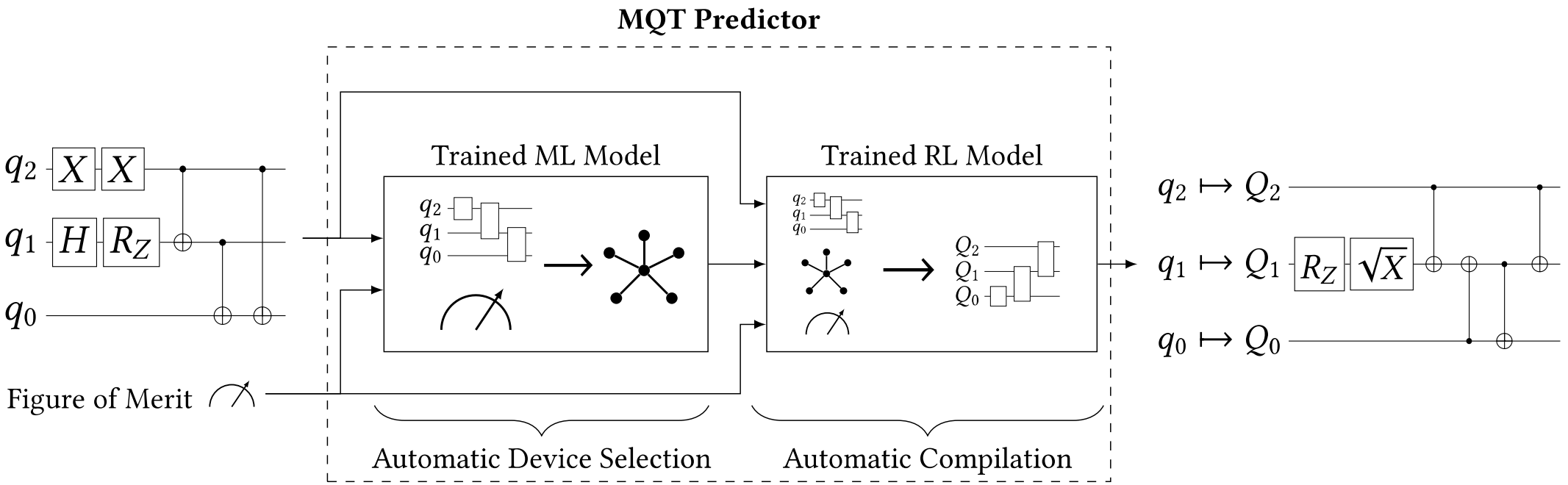
4 → 1

Resource Estimation



IBM Quantum Roadmap

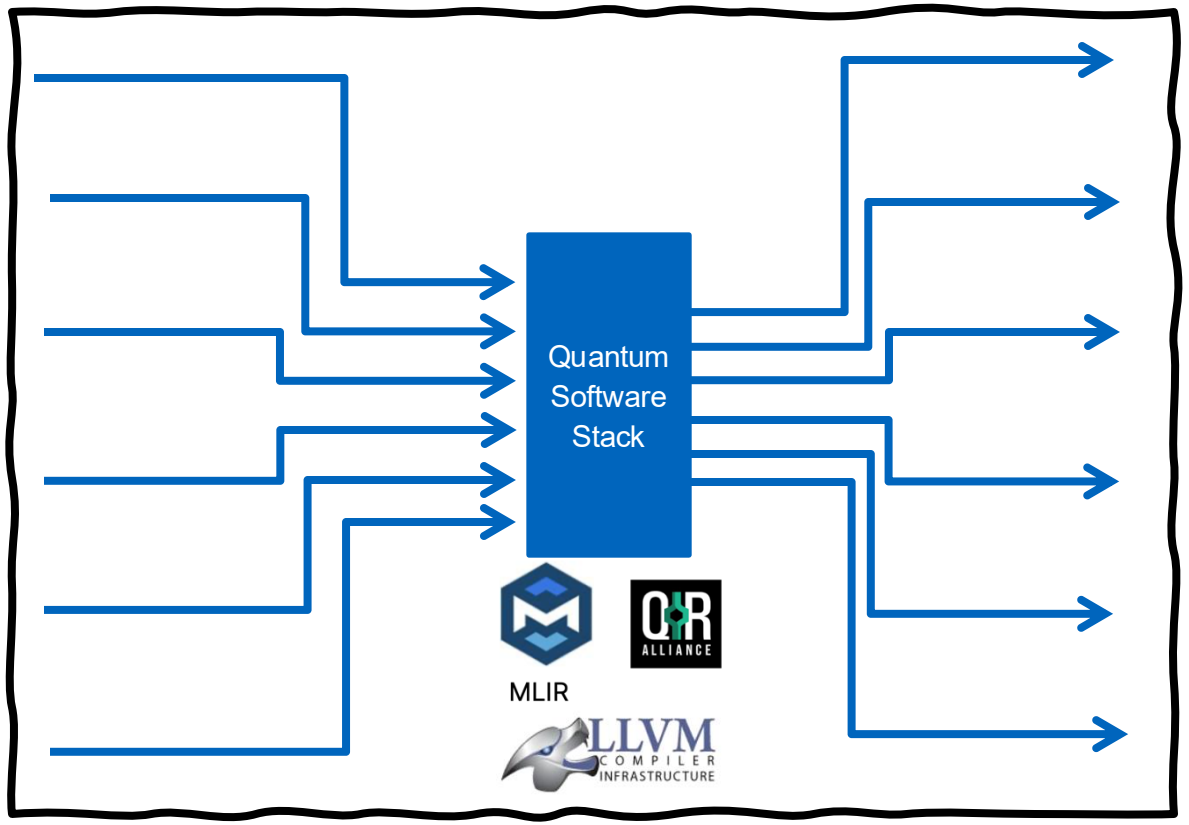
RL-based Circuit Compilation



Practical Part IV: Device Selection

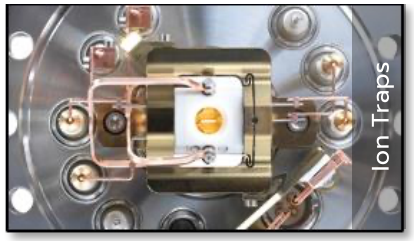
The Big Picture

Domain Experts

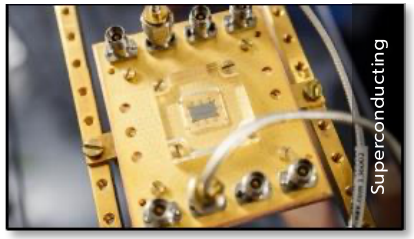


Quantum Devices

AQT @ LRZ



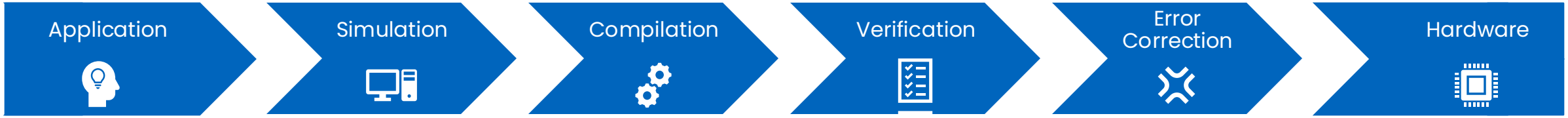
WMI @ LRZ



planqc @ LRZ

Software for Quantum Computing

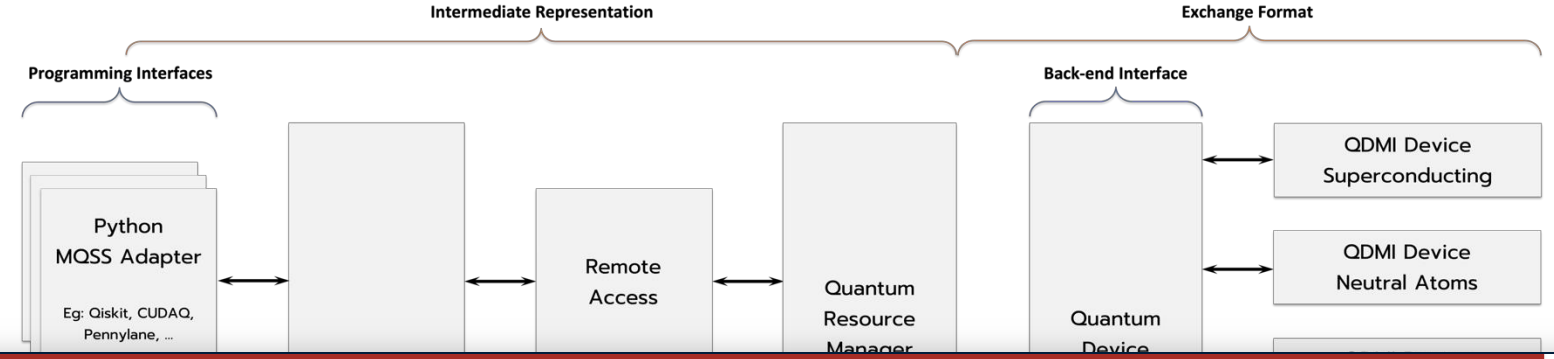
Needed across the whole spectrum…



Data Structures & Core Methods

MQSS
Munich Quantum Software Stack

Overview Paper
arxiv.org/abs/2509.02674



arXiv > quant-ph > arXiv:2509.02674

Search... Help | Adva

Quantum Physics

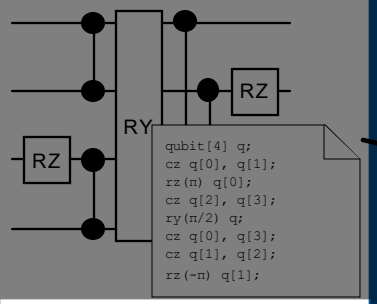
[Submitted on 2 Sep 2025]

The Munich Quantum Software Stack: Connecting End Users, Integrating Diverse Quantum Technologies, Accelerating HPC

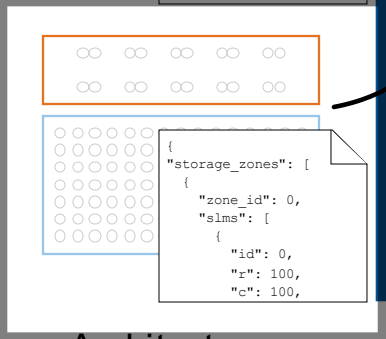
Lukas Burgholzer, Jorge Echavarria, Patrick Hopf, Yannick Stade, Damian Rovara, Ludwig Schmid, Ercüment Kaya, Burak Mete, Muhammad Nufail Farooqi, Minh Chung, Marco De Pascale, Laura Schulz, Martin Schulz, Robert Wille

Compiling for Next Generation Hardware

Quantum Circuit




```
qubit[4] q;
cz q[0], q[1];
rz(π) q[0];
cz q[2], q[3];
ry(π/2) q;
cz q[0], q[3];
cz q[1], q[2];
rz(-π) q[1];
```

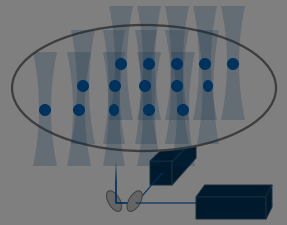


```
{
"storage_zones": [
  {
    "zone_id": 0,
    "slms": [
      {
        "id": 0,
        "r": 100,
        "c": 100,
```

Architecture Specification

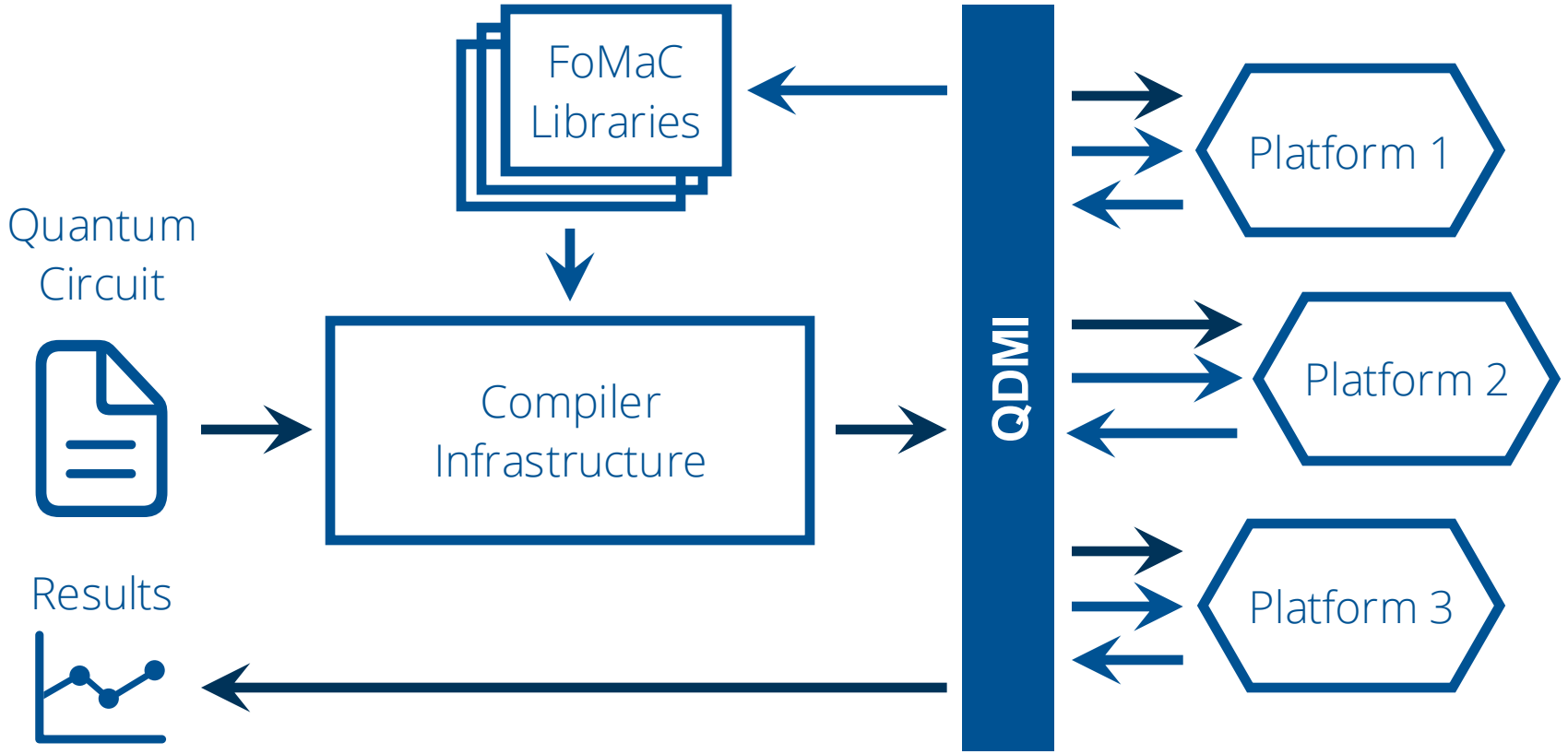
Compiler

Neutral Atom Quantum Computer



```
atom [
  (0.0, 18.0) atom0
  (3.0, 18.0) atom1
  (6.0, 18.0) atom2
  (9.0, 18.0) atom3
]
@+ load [
  atom0
  atom1
```

QDMI The Quantum Device Management Interface



QDMI Documentation

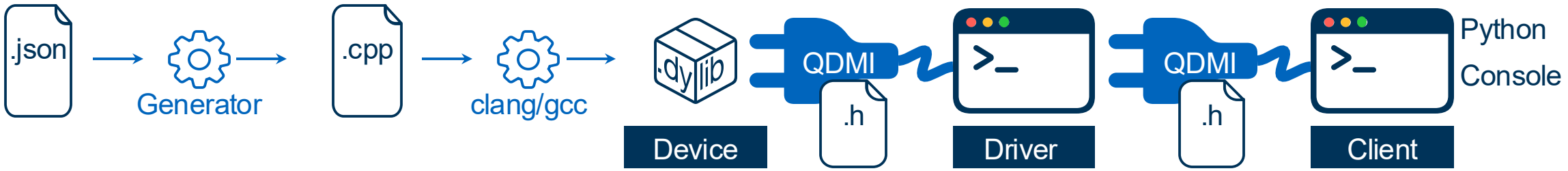
munich-quantum-software-stack.github.io/QDMI

QDMI GitHub

github.com/munich-quantum-software-stack/QDMI

open-source, openly-developed, multi-modality, HPC-compatible

Extending QDMI Device for Neutral Atoms




```
int MQ
int MQ
int MQ
```

Python

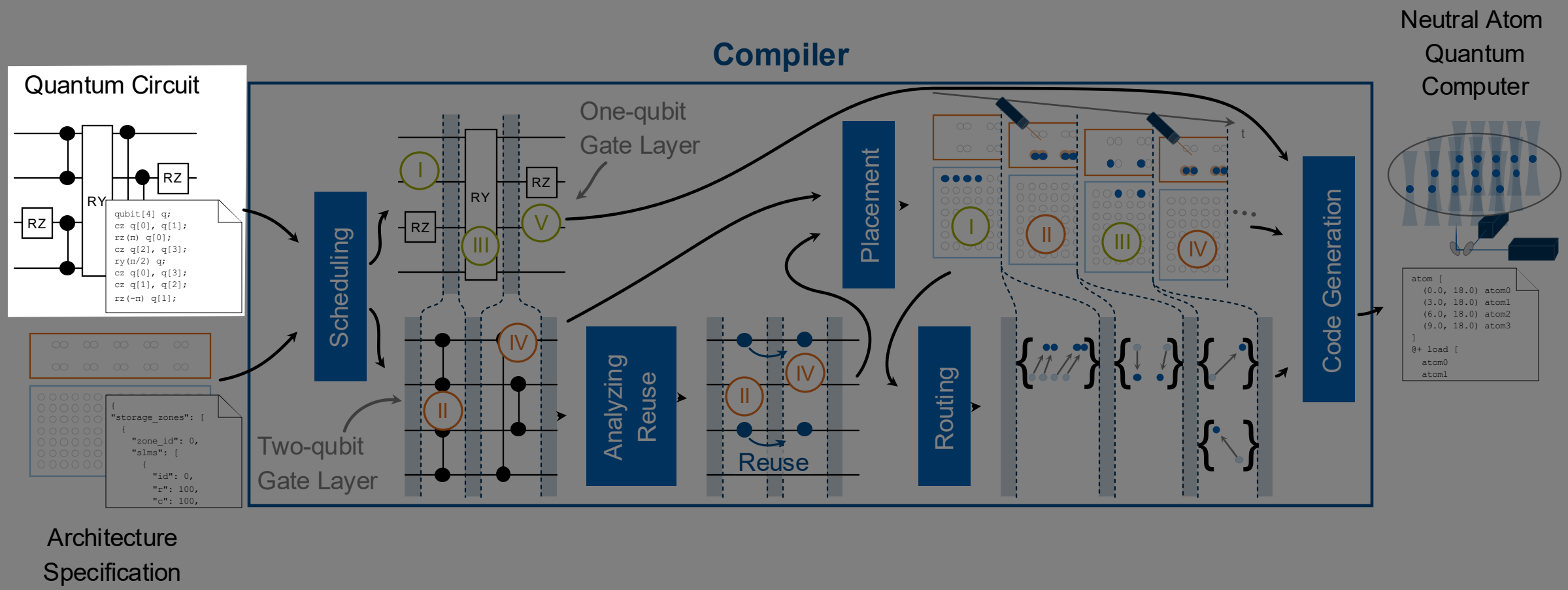
```
>>> from mqt.core.fomac import *
>>> device = devices()[0]
>>> device.name()
'MQT NA Default QDMI Device'
>>> site = device.sites()[24]
>>> site.x_coordinate(), site.y_coordinate()
(1, 2)
>>> [op.name() for op in device.operations()]
['ry', 'cz', 'rz', 'cz', 'load<0>', 'move<0>', 'store<0>']
```

MUNICH QUANTUM TOOLKIT

MQT Core

100 ★ 528k

Compiling for next generation hardware



IRs and Compiler Infrastructure

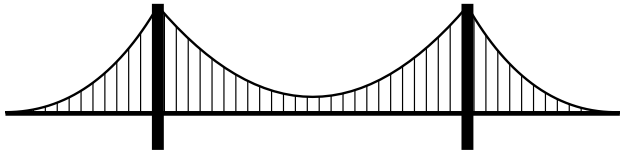
“quantum-first”

OpenQASM 2

OpenQASM 3

Qiskit

MUNICH QUANTUM TOOLKIT



“classical-first”

LLVM COMPILER INFRASTRUCTURE

QIR

Kirin Bloqade

PENNYLANE CATALYST

NVIDIA.

Building Bridges

PENNYLANE

```python
import pennylane as qml
from pennlylane.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)
```

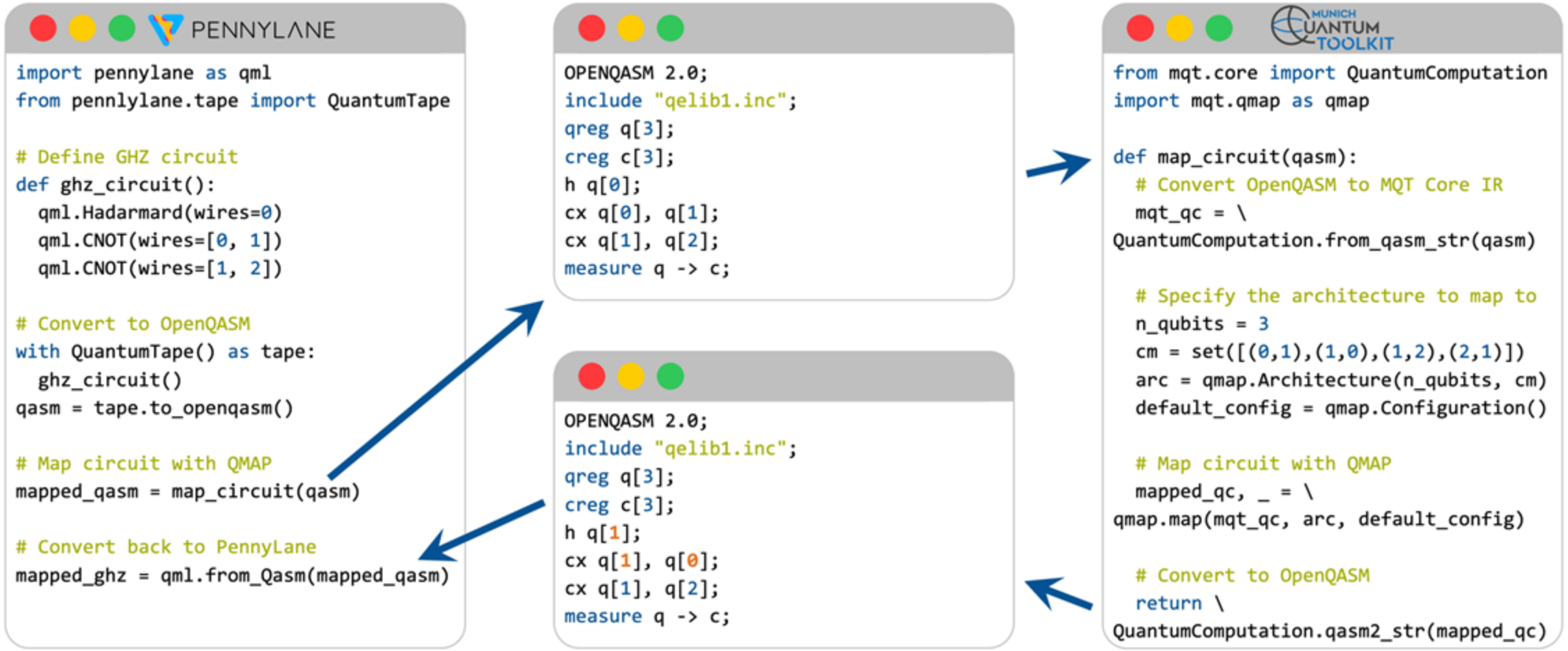
```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[0];
cx q[0], q[1];
cx q[1], q[2];
measure q -> c;
```

```
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;
```

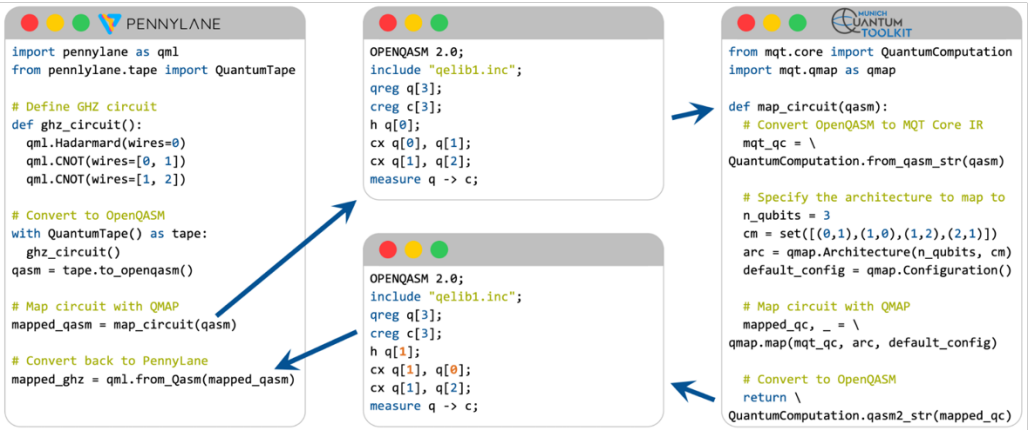
MUNICH QUANTUM TOOLKIT

```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)

  # Specify the architecture to map to
  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



Building Bridges



- Over 250 commits just on the plugin itself
  github.com/munich-quantum-toolkit/core/pull/881
- Over 55 closed Issues and PRs
  github.com/munich-quantum-toolkit/core/milestone/8

PENNYLANE

```
import pennylane as qml
from catalyst import measure
from catalyst.passes import apply_pass

@apply_pass(mqt.qmap{"cMap": [(0,1),(1,0),(1,2),(2,1)]})
@qml.qjit(target="mlir")
@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)]
```

MLIR Support in MUNICH QUANTUM TOOLKIT

MQT Core

100 ★ 528k 📈

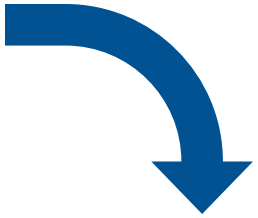
MQT Core
github.com/munich-quantum-toolkit/core

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

Behind the Scenes

PENNYLANE

```python
import pennylane as qml
from catalyst import measure
from catalyst.passes import apply_pass

@apply_pass(mqt.qmap{"cMap": [(0,1),(1,0),(1,2),(2,1)]})
@qml.qjit(target="mlir")
@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
```



MQTRef

```
func.func @bellState() {
  %qreg = "mqtref.allocQubitRegister"() <{size_attr = 3 : i64}> : () -> !mqtref.QubitRegister
  %q0 = "mqtref.extractQubit"(%qreg) <{index_attr = 0 : i64}> : (!mqtref.QubitRegister) -> !mqtref.Qubit
  %q1 = "mqtref.extractQubit"(%qreg) <{index_attr = 1 : i64}> : (!mqtref.QubitRegister) -> !mqtref.Qubit
  %q2 = "mqtref.extractQubit"(%qreg) <{index_attr = 2 : i64}> : (!mqtref.QubitRegister) -> !mqtref.Qubit
  mqtref.h() %q0
  mqtref.x() %q1 ctrl %q0
  mqtref.x() %q2 ctrl %q1
  %m0 = mqtref.measure %q0
  %m1 = mqtref.measure %q1
  %m2 = mqtref.measure %q2
  "mqtref.deallocQubitRegister"(%qreg) : (!mqtref.Qubi
  return
}
```

Reference Semantics

⇒ Similar to OpenQASM, QIR, ...

⇒ Good fit for In/Output

Behind the Scenes

MQTOpt

```
func.func @ghzState() {
  %reg_0 = "mqtopt.allocQubitRegister"() <{size_attr = 3 : i64}> : () -> !mqtopt.QubitRegister
  %reg_1, %q0_0 = "mqtopt.extractQubit"(%reg_0) <{index_attr = 0 : i64}> : (!mqtopt.QubitRegister) -> (!mqtopt.QubitRegister, !mqtopt.Qubit)
  %reg_2, %q1_0 = "mqtopt.extractQubit"(%reg_1) <{index_attr = 1 : i64}> : (!mqtopt.QubitRegister) -> (!mqtopt.QubitRegister, !mqtopt.Qubit)
  %reg_3, %q2_0 = "mqtopt.extractQubit"(%reg_2) <{index_attr = 2 : i64}> : (!mqtopt.QubitRegister) -> (!mqtopt.QubitRegister, !mqtopt.Qubit)
  %q0_1 = mqtopt.h() %q0_0 : !mqtopt.Qubit

  %q1_1, %q0_2 = mqtopt.x() %q1_0 ctrl %q0_1 : !mqtopt.Qubit ctrl !mqtopt.Qubit

  %q2_1, %q1_2 = mqtopt.x() %q2_0 ctrl %q1_1 : !mqtopt.Qubit ctrl !mqtopt.Qubit

  %q0_3, %m0_0 = "mqtopt.measure"(%q0_2) : (!mqtopt.Qubit) -> (!mqtopt.Qubit, i1)

  %q1_3, %m1_0 = "mqtopt.measure"(%q1_2) : (!mqtopt.Qubit) -> (!mqtopt.Qubit, i1)

  %q2_2, %m1_0 = "mqtopt.measure"(%q2_1) : (!mqtopt.Qubit) -> (!mqtopt.Qubit, i1)
  %reg_4 = "mqtopt.insertQubit"(%reg_3, %q0_3) <{index_attr = 0 : i64}> : (!mqtopt.QubitRegister, !mqtopt.Qubit) -> !mqtopt.QubitRegister
  %reg_5 = "mqtopt.insertQubit"(%reg_4, %q1_3) <{index_attr = 1 : i64}> : (!mqtopt.QubitRegister, !mqtopt.Qubit) -> !mqtopt.QubitRegister
  %reg_6 = "mqtopt.insertQubit"(%reg_5, %q2_2) <{index_attr = 2 : i64}> : (!mqtopt.QubitRegister, !mqtopt.Qubit) -> !mqtopt.QubitRegister
 "mqtopt.deallocQubitRegister"(%reg_6) : (!mqtopt.QubitRegister) -> ()
  return
}
```

Value Semantics

⇒ Makes Dataflow apparent

⇒ Good for Optimizations

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?

Structured Programs
Support beyond the base profile of QIR

Uniform Exchange Format
Can we agree on some common format?

JEFF BRIDGES COMPILATION

github.com/unitaryfoundation/jeff

XANADU

Unitary Foundation

QUANTINUUM

BERKELEY LAB

UBC

The Munich Quantum Toolkit (MQT)

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

MQT Repositories

Leave a ★

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
munich-quantum-toolkit/problemsolver

MQT Bench — Application
Benchmarking Software and Tools for Quantum Computing
mqt-bench.app
munich-quantum-toolkit/bench

MQT QuSAT — Core Methods
A Tool for Encoding Quantum Computing using Satisfiability Testing (SAT) Techniques
$F \wedge (x_1 \wedge \neg x_2)$
$F \wedge (x_3 \wedge x_2)$
munich-quantum-toolkit/qusat

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

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

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

MQT IonShuttler — Compilation
A Tool for Generating Shuttling Schedules for QCCD Architectures
munich-quantum-toolkit/ionshuttler

MQT Qudits — Compilation
A Tool for Compiling High-Dimensional Quantum Systems
munich-quantum-toolkit/qudits

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

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

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

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

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
munich-quantum-toolkit/core

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

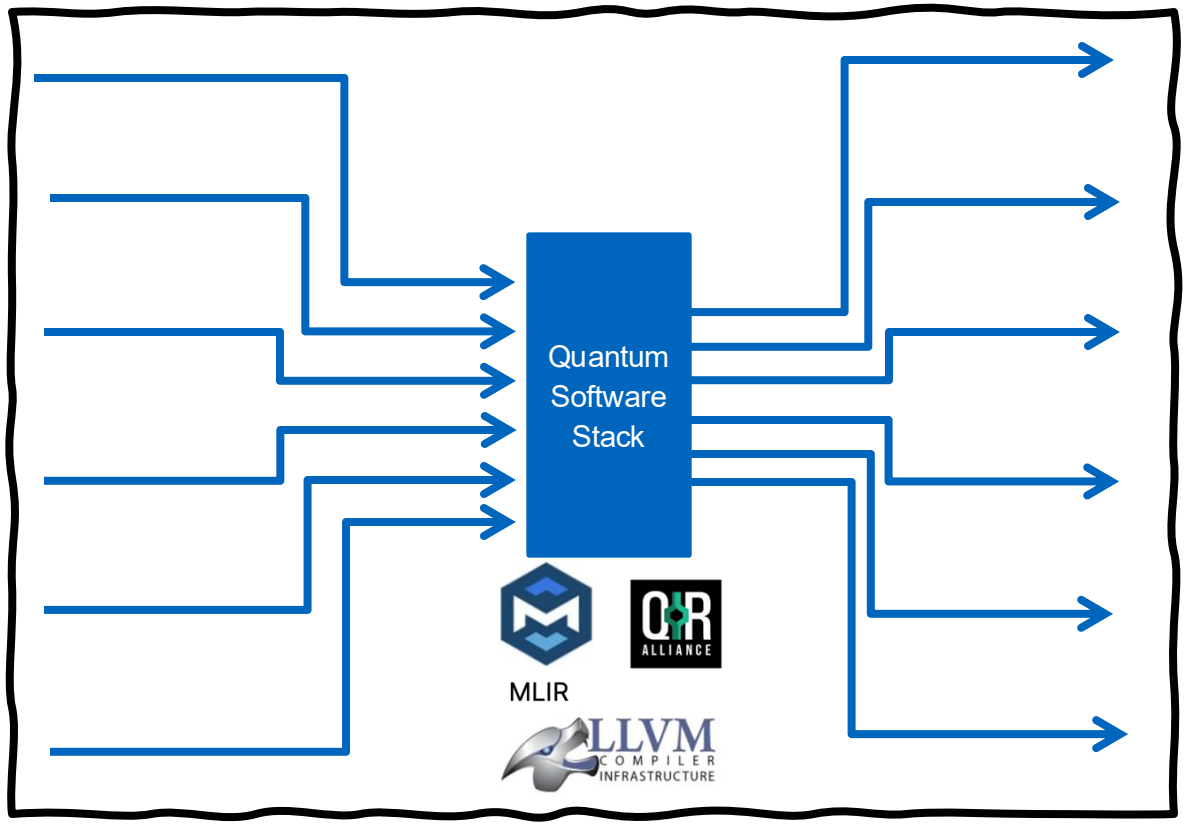
MQT QECC — QECC
A Tool for Quantum Error Correcting Codes
munich-quantum-toolkit/qecc

https://mqt.readthedocs.io

https://github.com/munich-quantum-toolkit

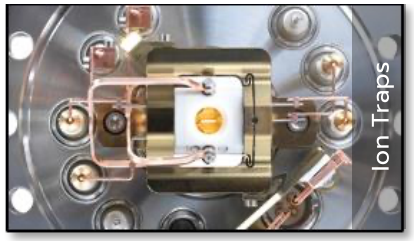
The Big Picture

Domain Experts

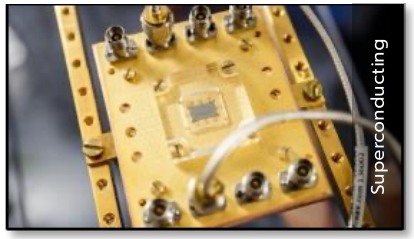


Quantum Devices

AQT @ LRZ



WMI @ LRZ



planqc @ LRZ


BENCHMARKS
PROGRAMS
QDMI
MUNICH QUANTUM SOFTWARE COMPANY
MQSS
Simulation
MLIR
MQT
VERIFICATION
Munich Quantum Valley
munichquantum.software