Connecting the HPC and the Quantum Community: A Tutorial on High-Performance Software for Quantum Computing

Lukas Burgholzer, Nils Quetschlich, Robert Wille, and Team

Technical University of Munich, Germany

lukas.burgholzer@tum.de

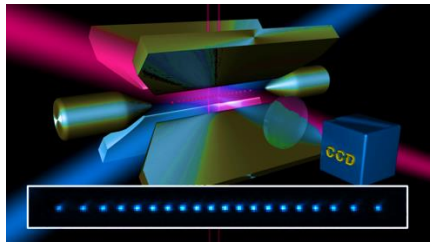
https://www.cda.cit.tum.de/research/quantum/

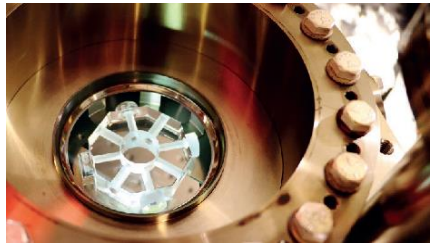

Munich
Quantum
Valley

The Big Picture

Domain Experts

Quantum Devices

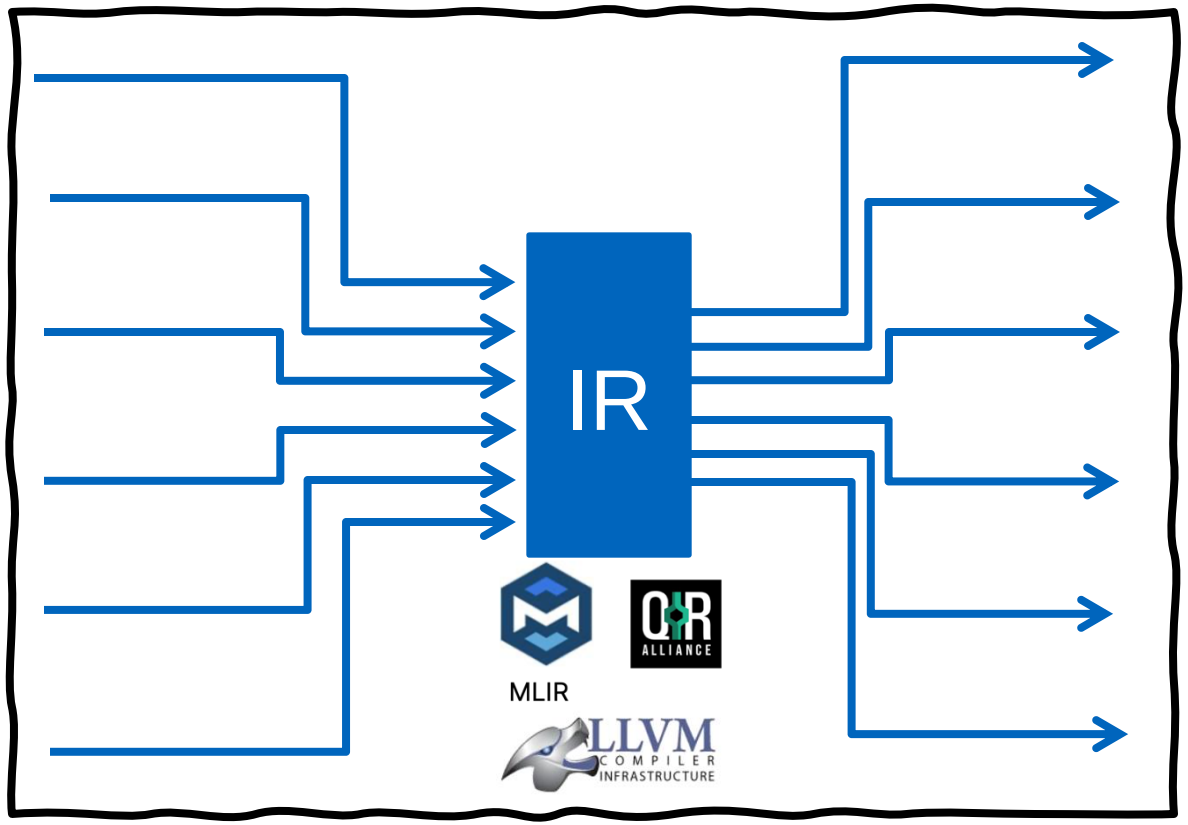




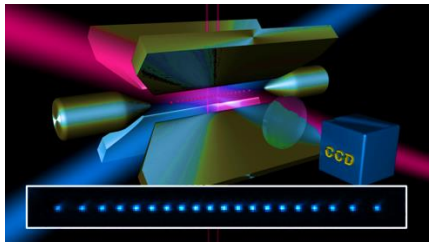
The Big Picture

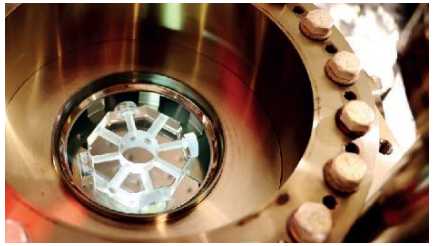
Domain Experts

Quantum Computing Software Stack



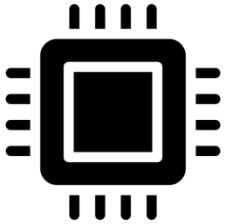
Quantum Devices





Chair for Design Automation

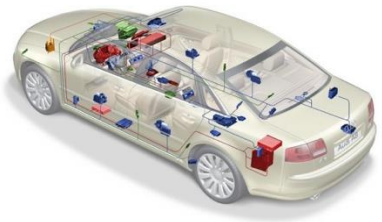
Designing Systems



- **Classical Systems**
  - Are enormously **complex**

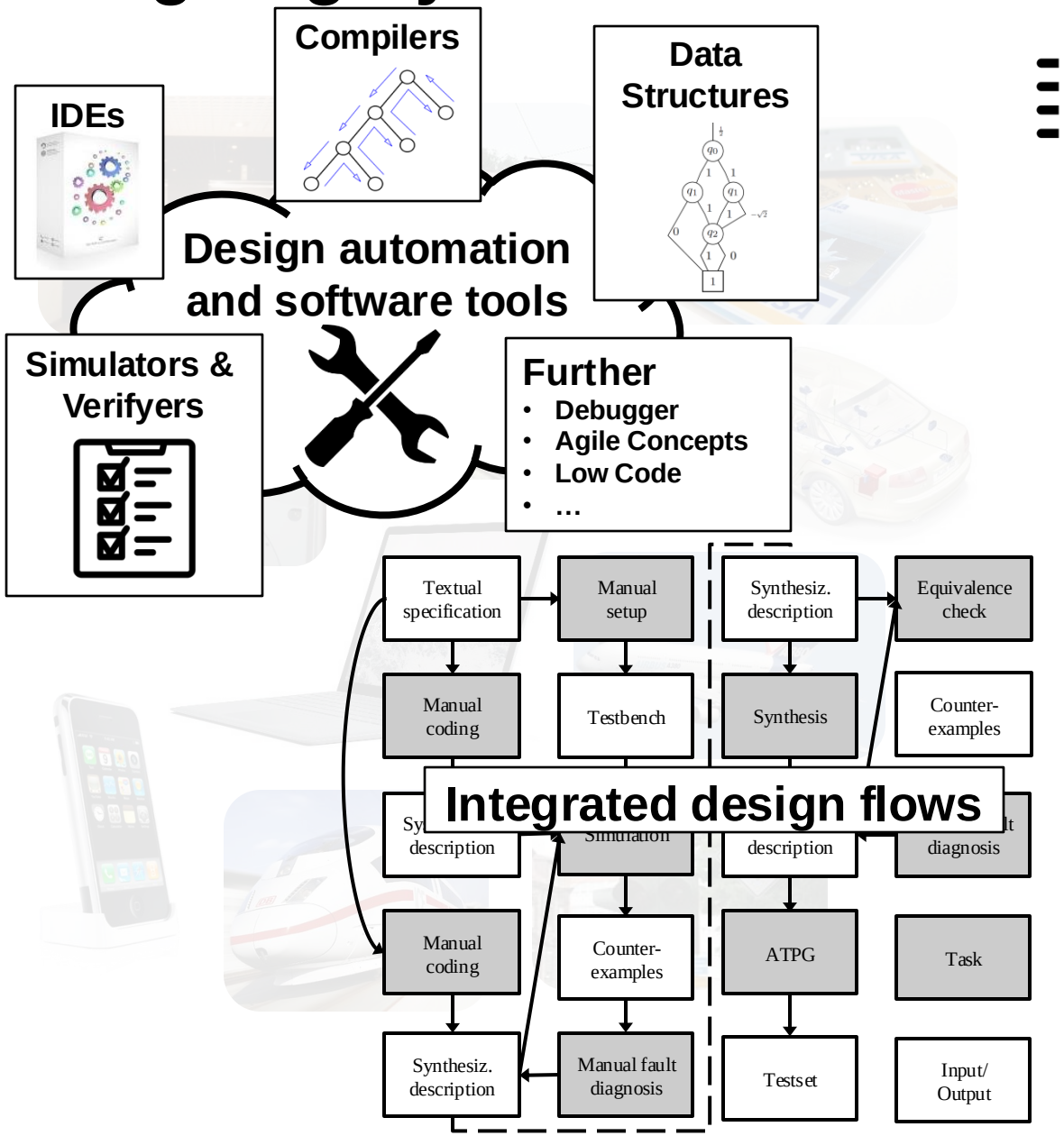






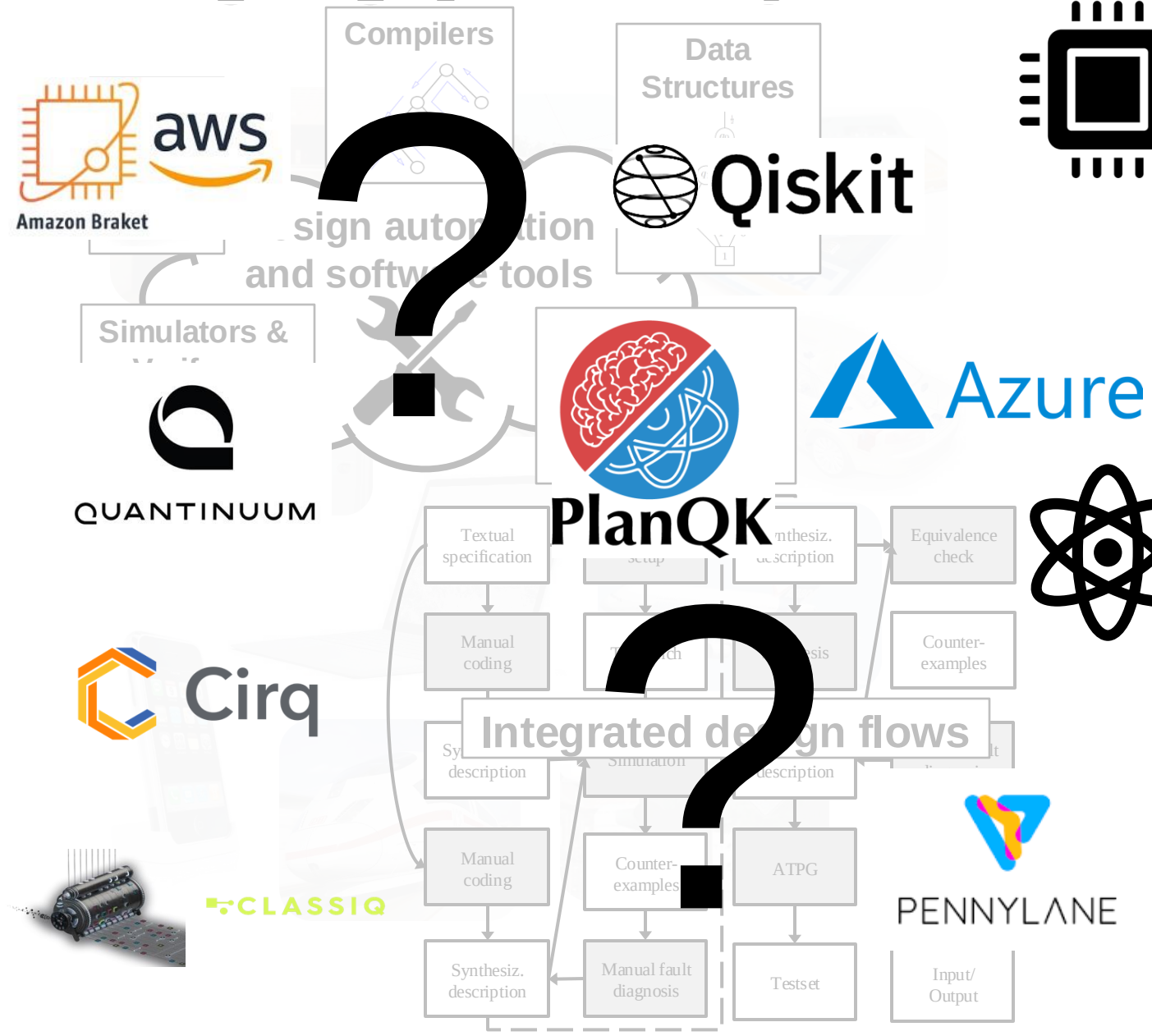



Designing Systems



- **Classical Systems**
  - Are enormously **complex**
  - Rely on efficient and scalable **design automation methods and software**
  - Integrated into powerful **design flows**
  - Have been developed and optimized **over the past decades**

Designing Quantum Systems



- **Classical Systems**
  - Are enormously **complex**
  - Rely on efficient and scalable **design automation methods and software**
  - Integrated into powerful **design flows**
  - Have been developed and optimized **over the past decades**

- **Quantum Systems**
  - Similar tasks (simulation, compilation, etc.), but **substantially different** paradigm
  - **Initial solutions** are available
  - **Do not** fully exploit the decades of experiences in **design automation**

We may end up in a situation where we have quantum computers but no methods to use them

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

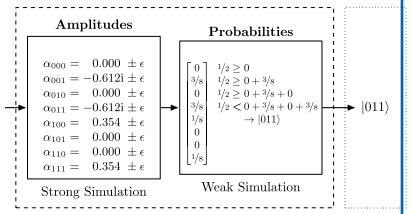
Error Correction

- 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

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/cda-tum/mqt-problemsolver

MQT Bench — Application
A Quantum Circuit Benchmark Suite
www.cda.cit.tum.de/mqtbench
github.com/cda-tum/mqt-bench

MQT Quantum Auto Optimizer — Application
An Automatic Framework for Solving Optimization Problems
github.com/cda-tum/mqt-qao

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

MQT DDSIM — Simulation
A Tool for Classical Quantum Circuit Simulation based on Decision Diagrams
github.com/cda-tum/mqt-ddsim

MQT Predictor — Compilation
A Tool for Determining Good Quantum Circuit Compilation Options
github.com/cda-tum/mqt-predictor

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

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

MQT SyReC — Compilation
A Tool for the Synthesis of Reversible Circuits/Quantum Computing Oracles
github.com/cda-tum/mqt-syrec

MQT QMAP — Compilation
A Tool for Quantum Circuit Mapping And Clifford Circuit Optimization/Synthesis
github.com/cda-tum/mqt

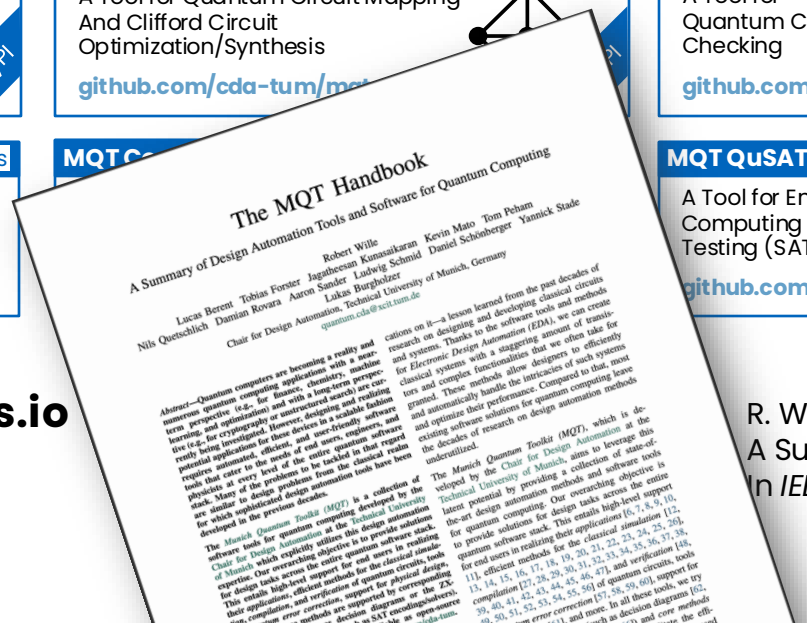
MQT QCEC — Verification
A Tool for Quantum Circuit Equivalence Checking
github.com/cda-tum/mqt-qcec

MQT DASQA — Hardware
A Tool for Designing Alternative Superconducting Quantum Architectures
github.com/cda-tum/mqt-dasqa

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

MQT QuSAT — Core Methods
A Tool for Encoding Quantum Computing using Satisfiability Testing (SAT) Techniques
$F \wedge (x_1 \wedge \neg x_2)$
$F \wedge (x_3 \wedge x$
github.com/cda-tum/mqt-qusat

MQT QECC — QECC
A Tool for Quantum Error Correcting Codes
github.com/cda-tum/mqt-qecc


The MQT Handbook
A Summary of Design Automation Tools and Software for Quantum Computing

https://mqt.readthedocs.io

R. Wille et al. The MQT Handbook:
A Summary of Design Automation Tools and Software for Quantum Computing.
In *IEEE International Conference on Quantum Software (QSW)*. 2024

Agenda

- First Session (13:30 – 15:00)
  - Intro to quantum computing
  - Classical simulation of quantum circuits
  - Hands-on

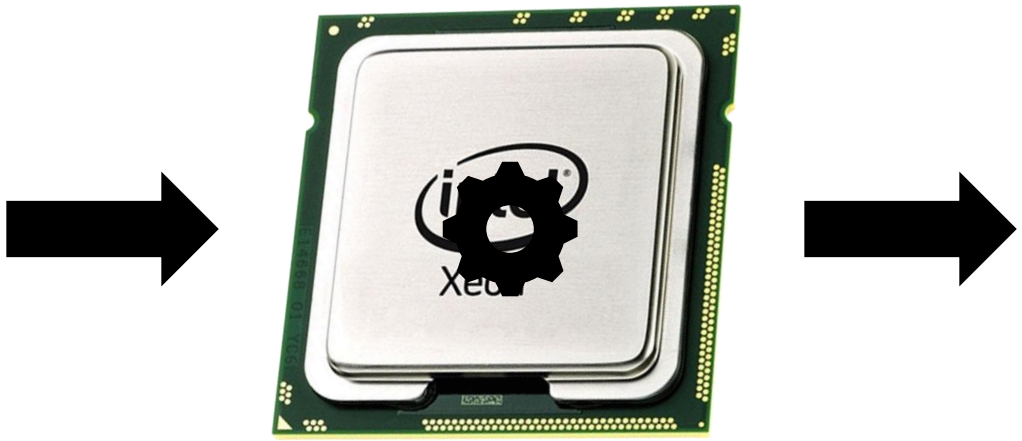
- Second Session (15:30 – 17:00)
  - Compilation of quantum circuits
  - Hands-on
  - Verification of quantum circuits
  - Hands-on
  - Conclusion & Wrap-Up

Classical Computing

Classical Computing

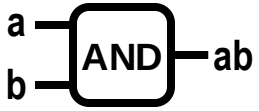
ASM

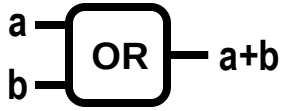
```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

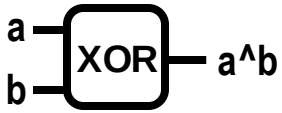


1110
0110
0001
1001

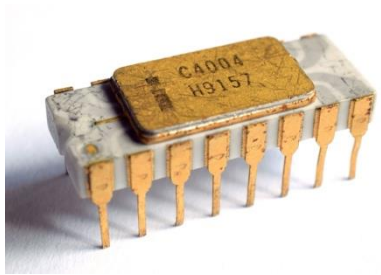
Classical Computing

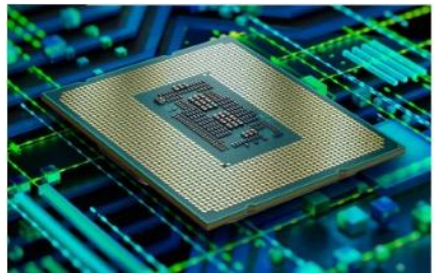


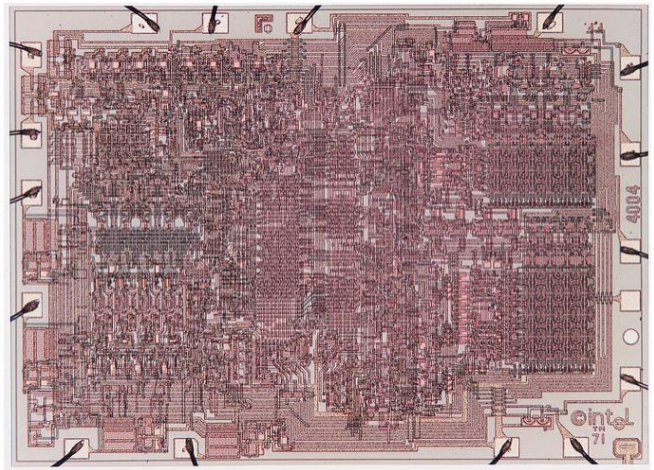


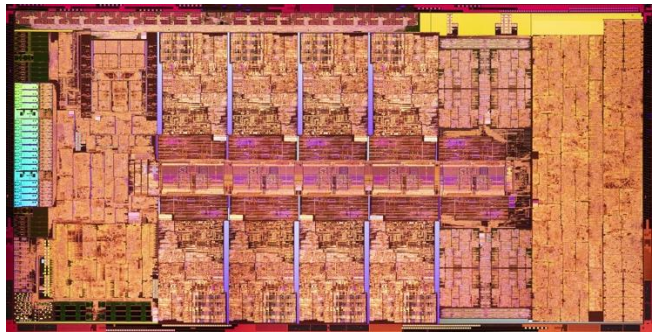


1971 Intel® 4004		2021 12th Generation Intel® Core™
2300	Number of Transistors	Billions
16	Number of Pins	1700
750 kHz	Frequency	5.2 GHz
4-bit	Instruction Set	64-bit
1	Number of Cores	16









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

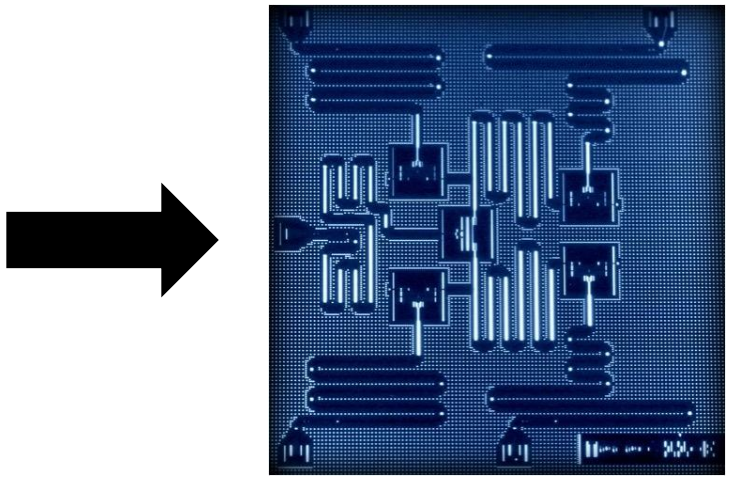
```
mov r0, #1
mov r1, #1
l:
add r2, r0, r1
str r2, [r3]
add r3, #4
mov r0, r1
mov r1, r2
b l
```

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
```

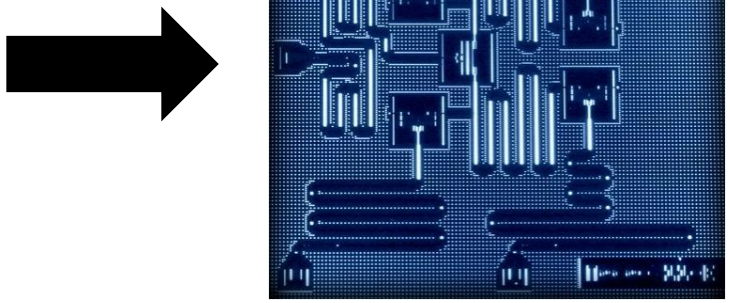


1110
0110
0001
1001

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

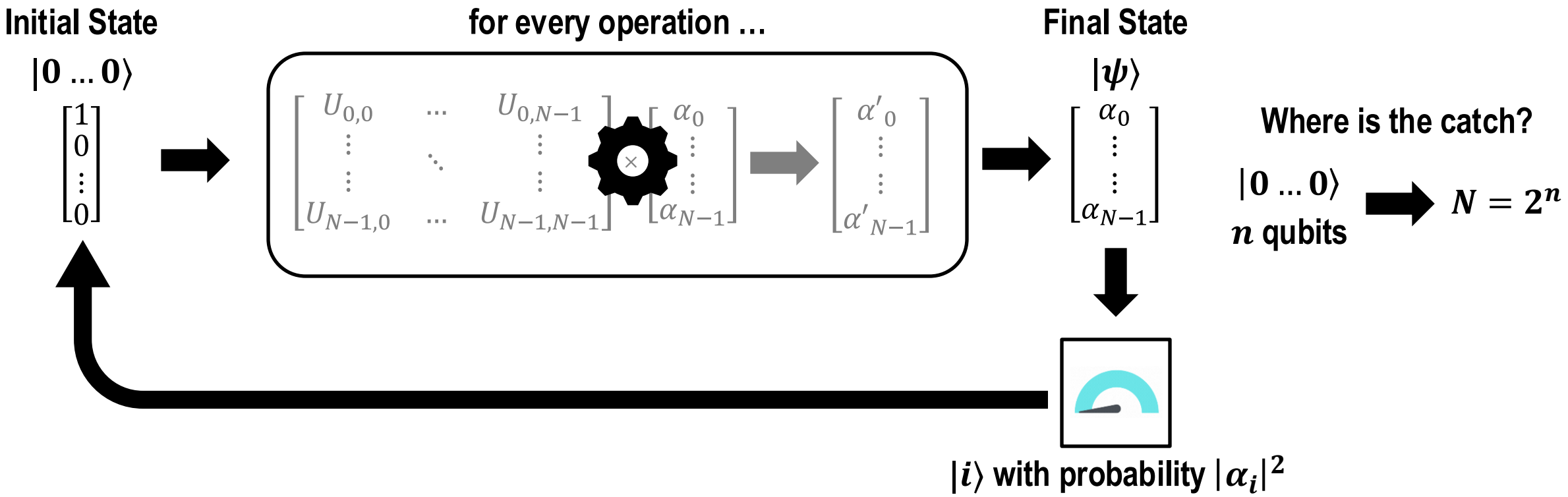
Quantum Operation | Quantum State | Evolved Quantum State

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

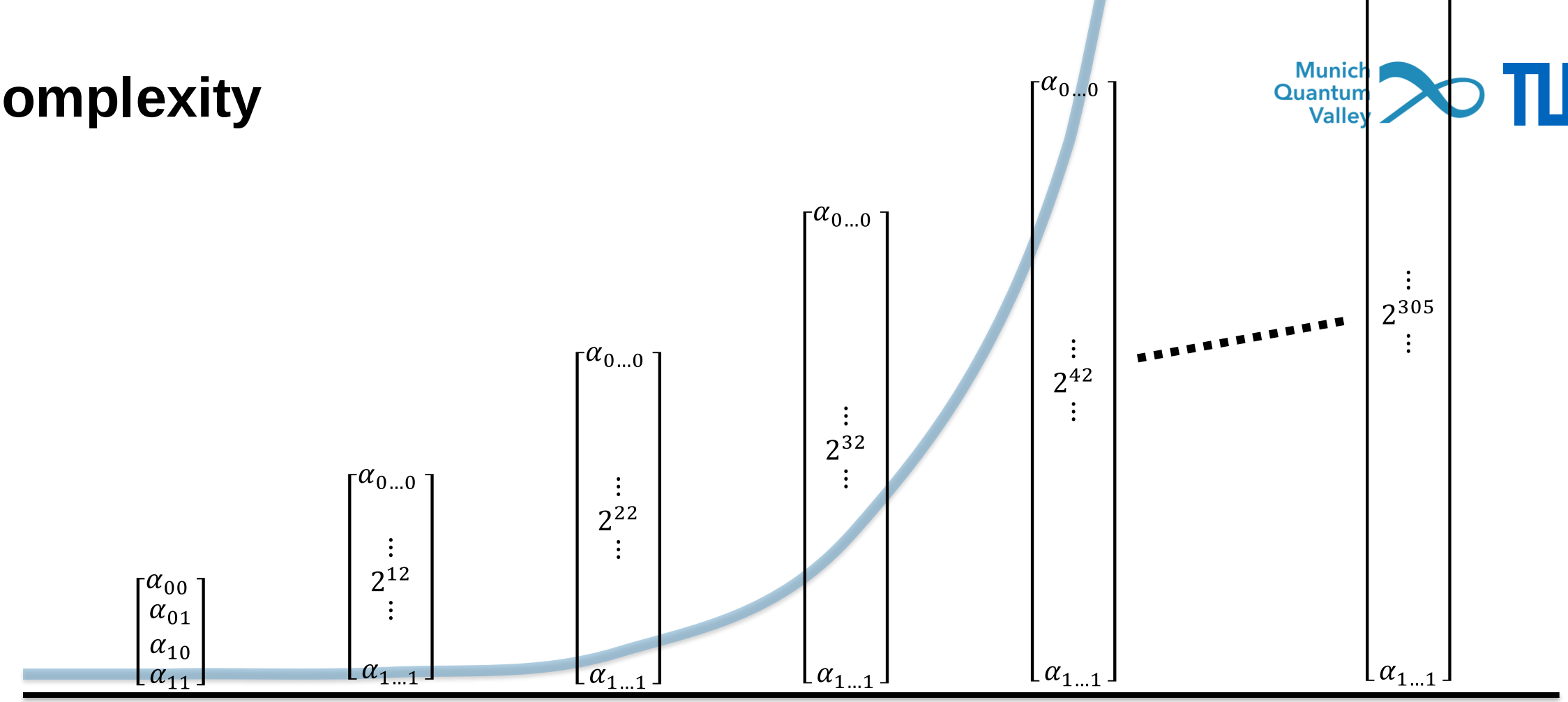
$$U_{i,j} \in \mathbb{C} \qquad \alpha_i \in \mathbb{C}$$

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

Quantum Computing

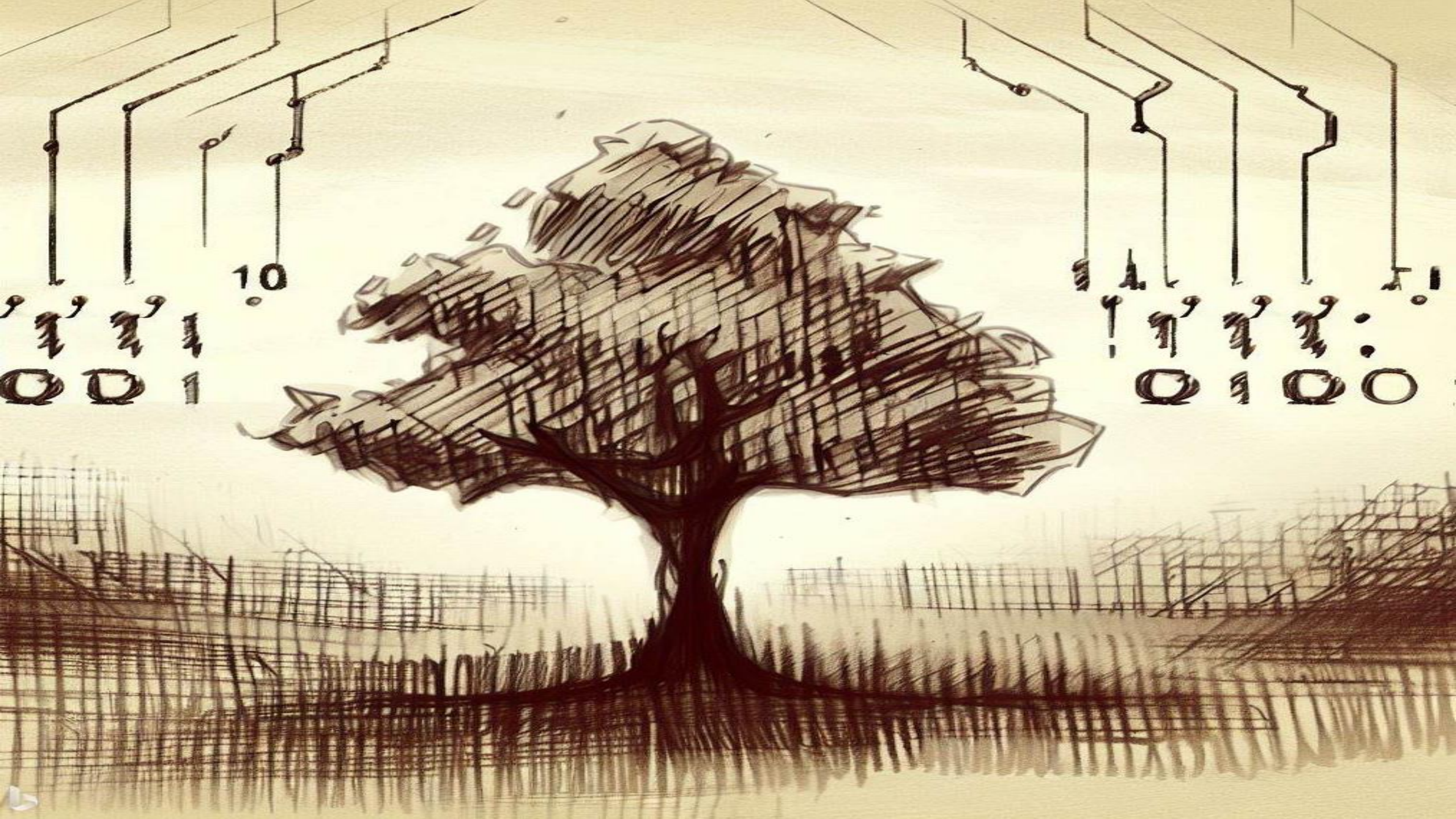


Complexity



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

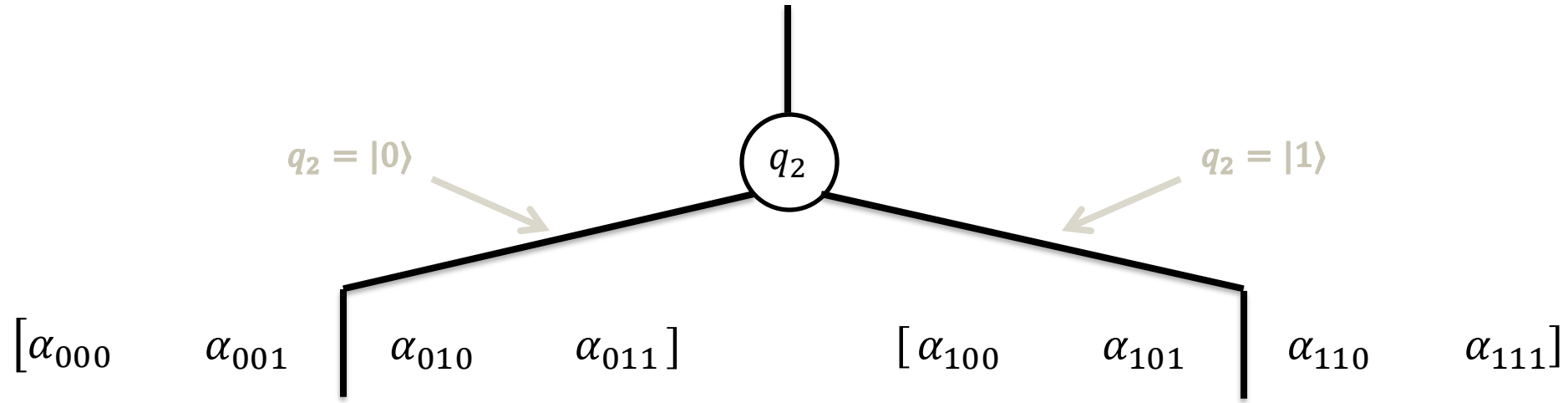
HOW? Design Automation!



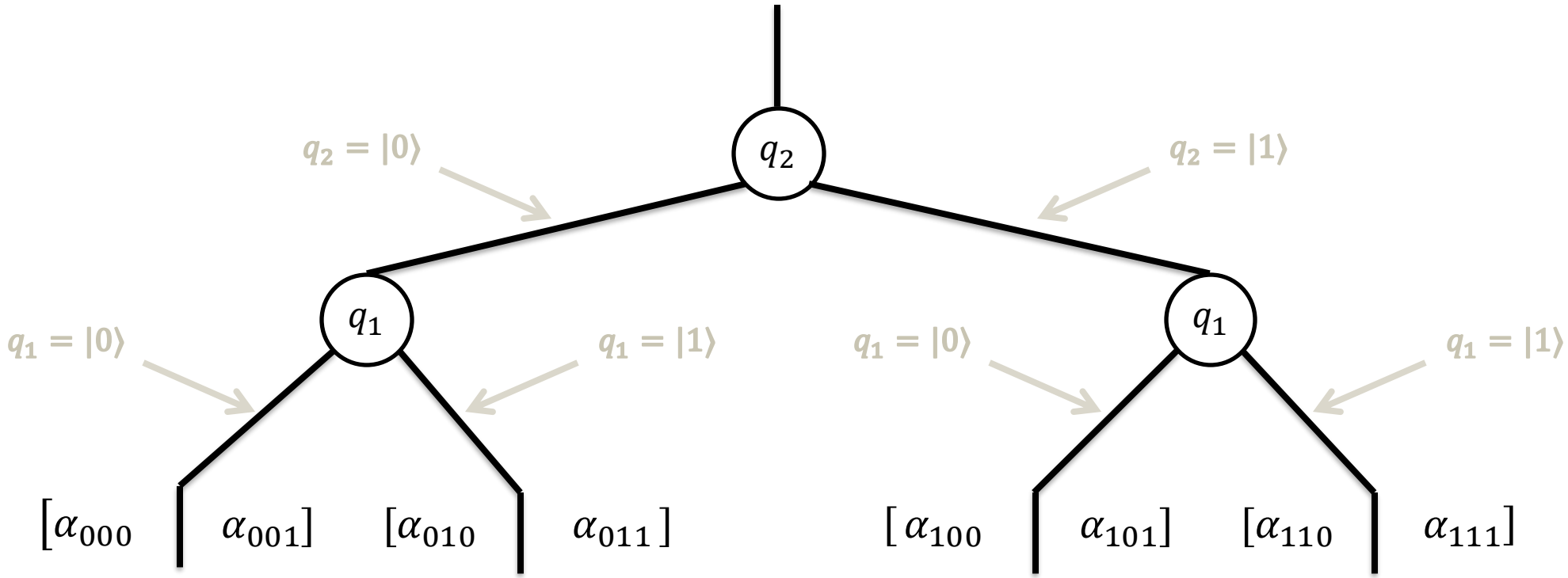
Structure

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

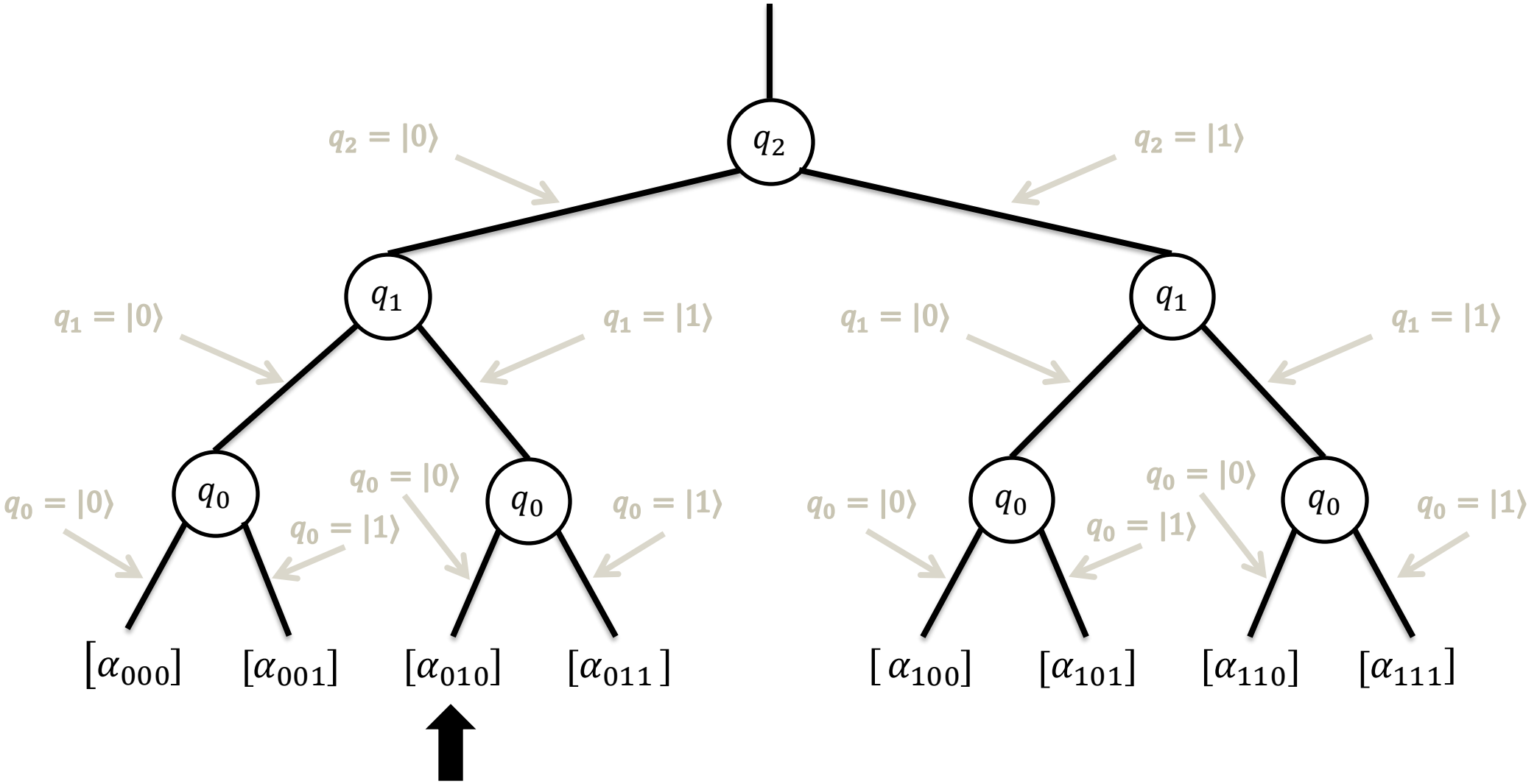
Structure


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

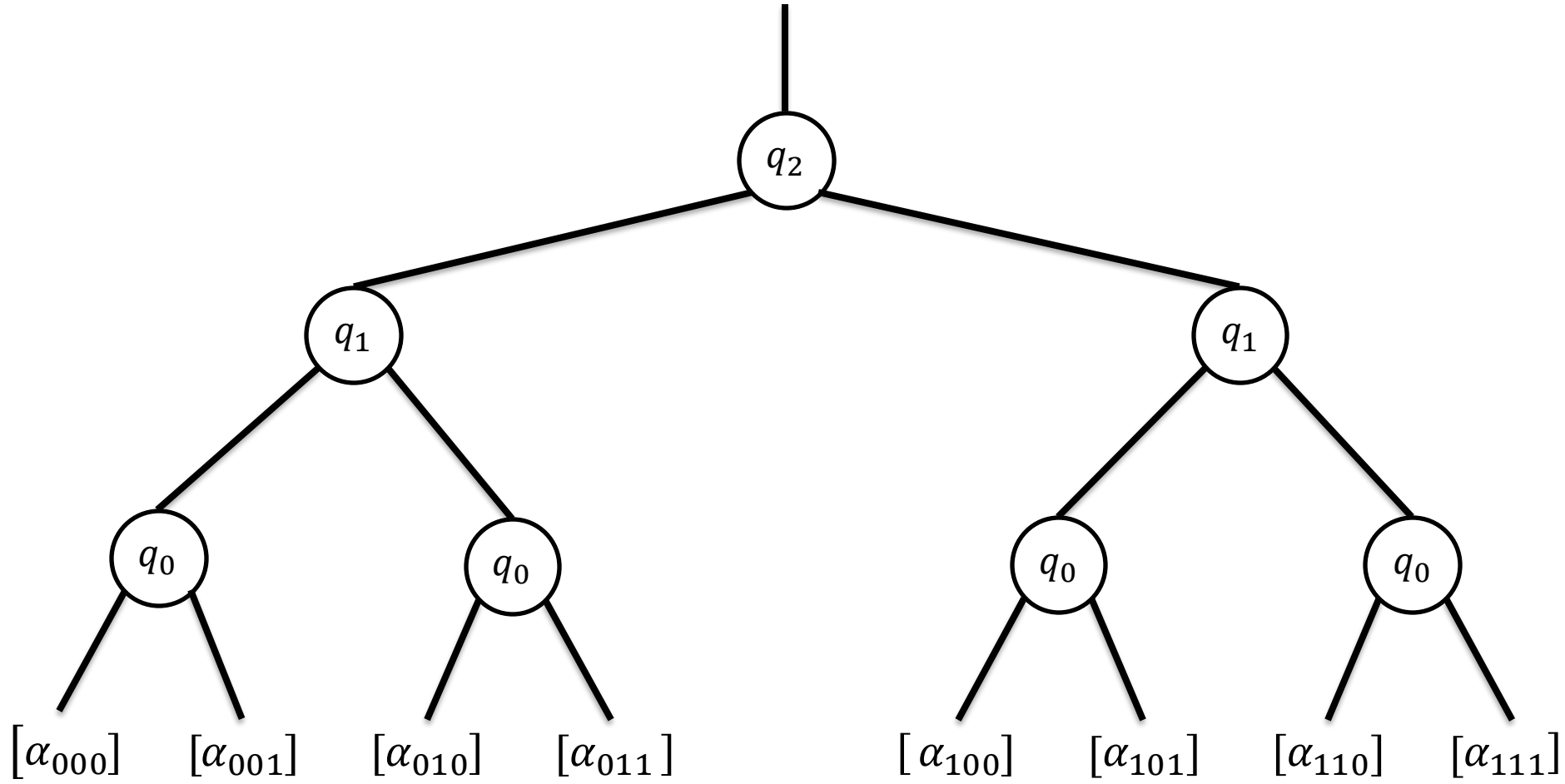
Structure


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

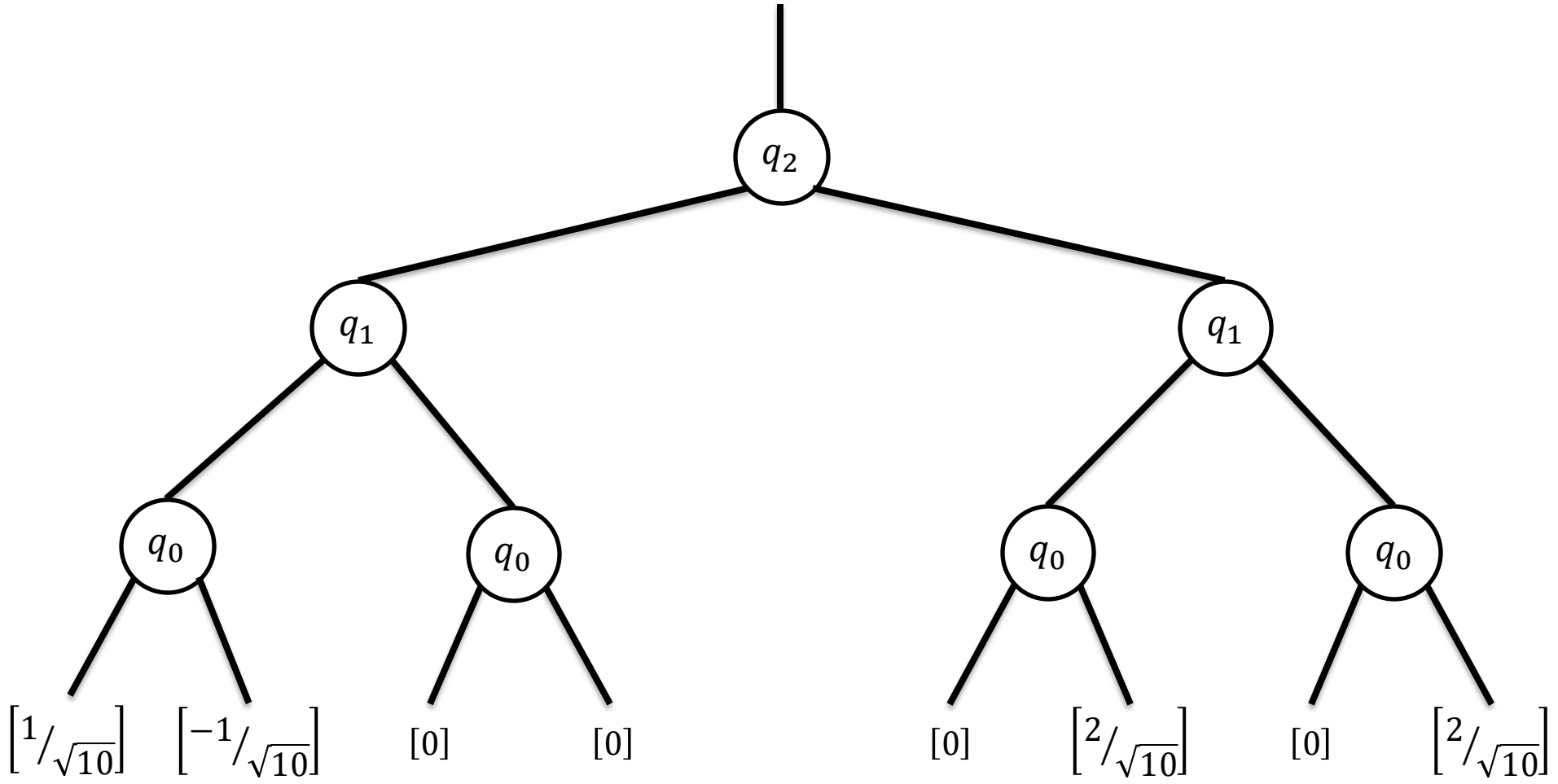
Structure


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

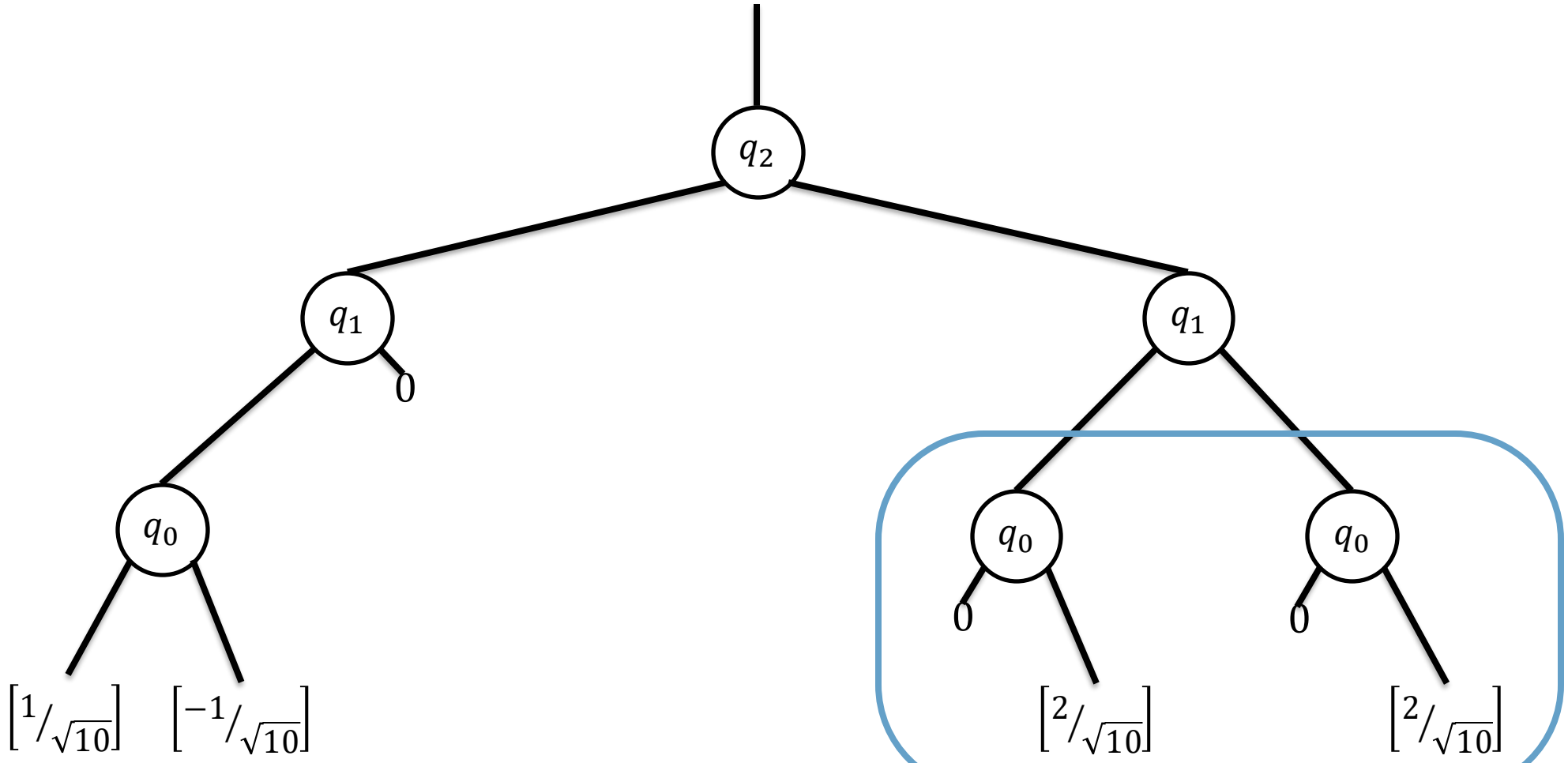
Structure + Redundancy



Structure + Redundancy

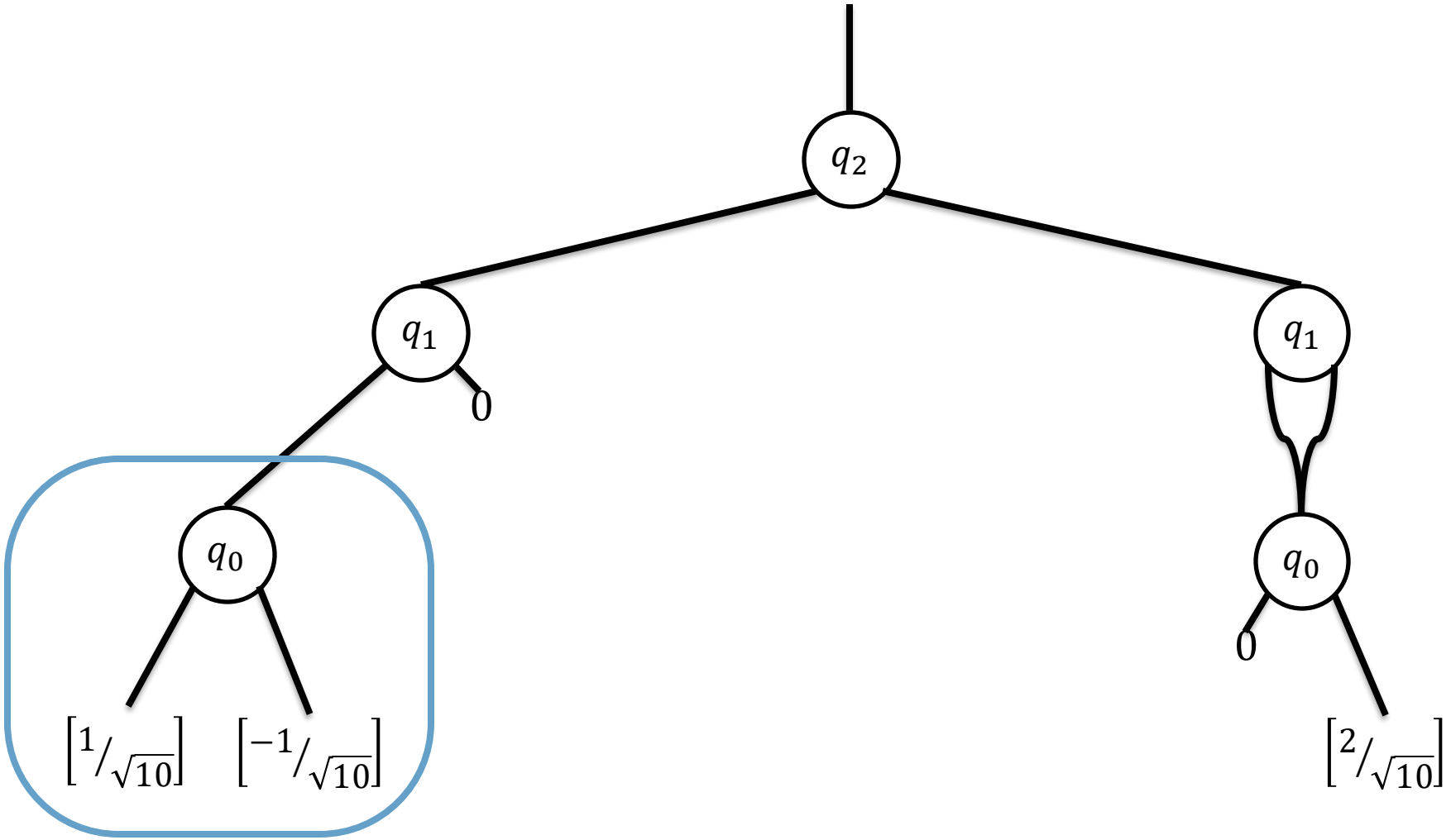


Structure + Redundancy



sparse

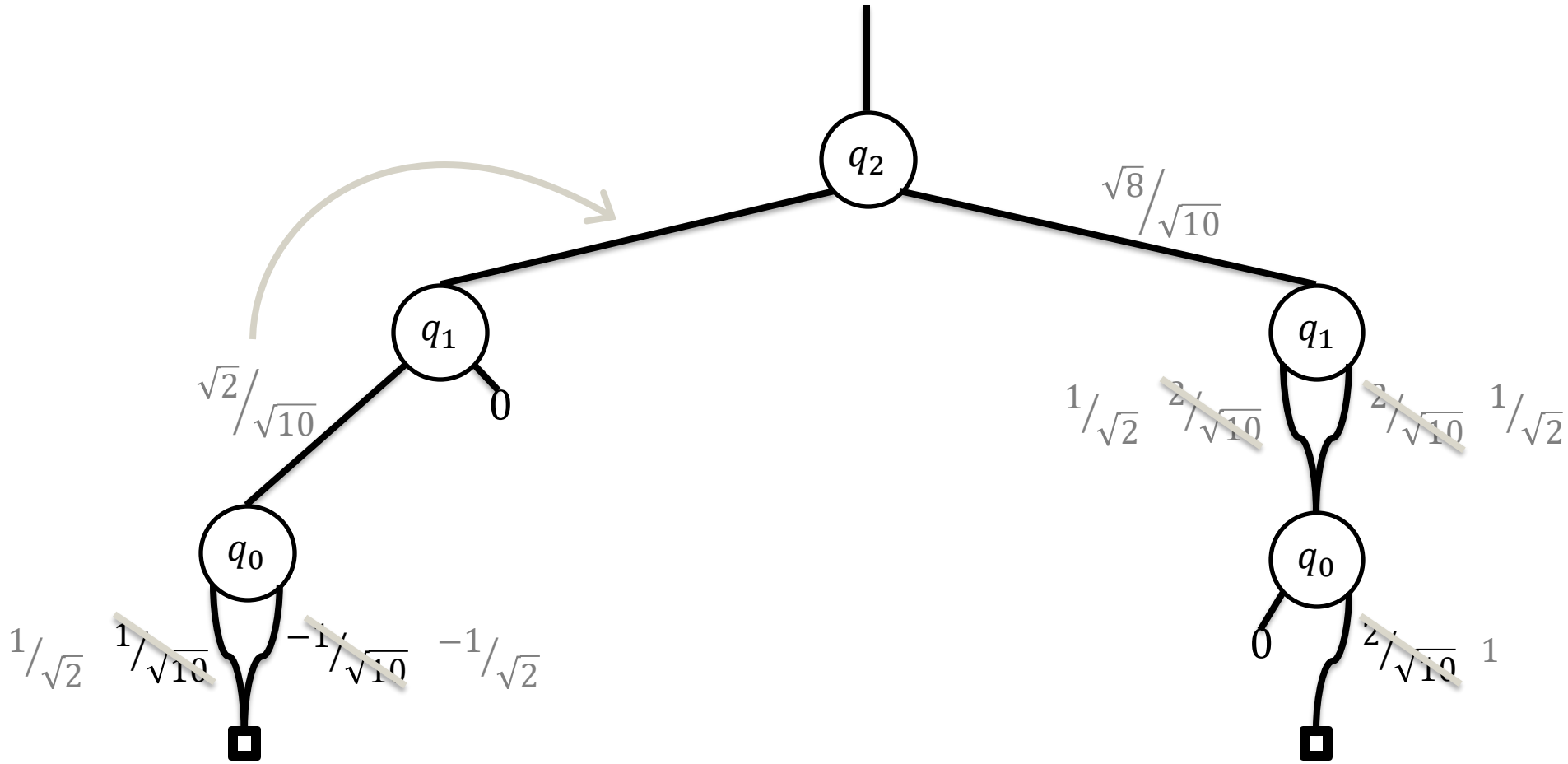
Structure + Redundancy



sparse

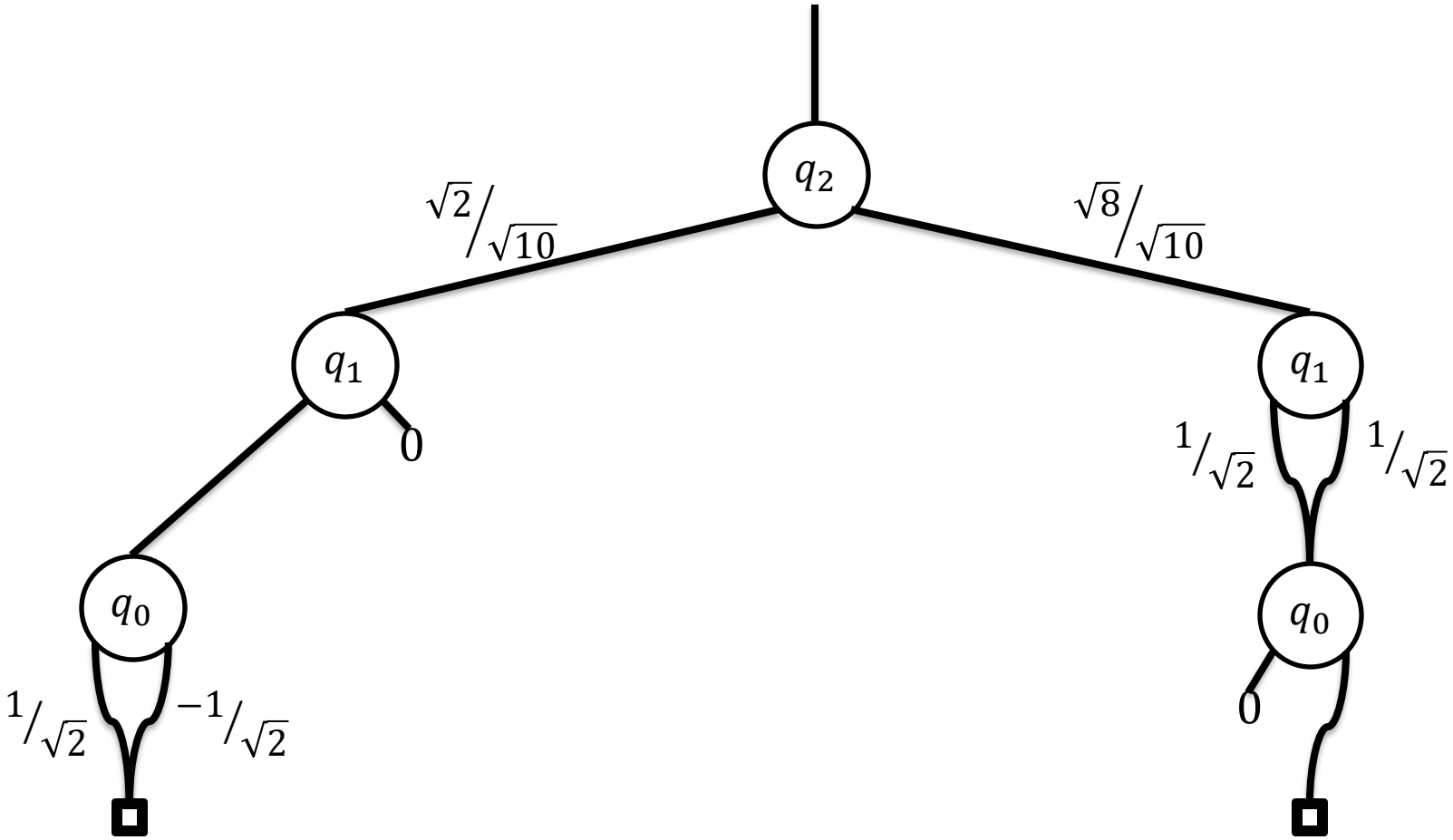
reduced

Structure + Redundancy



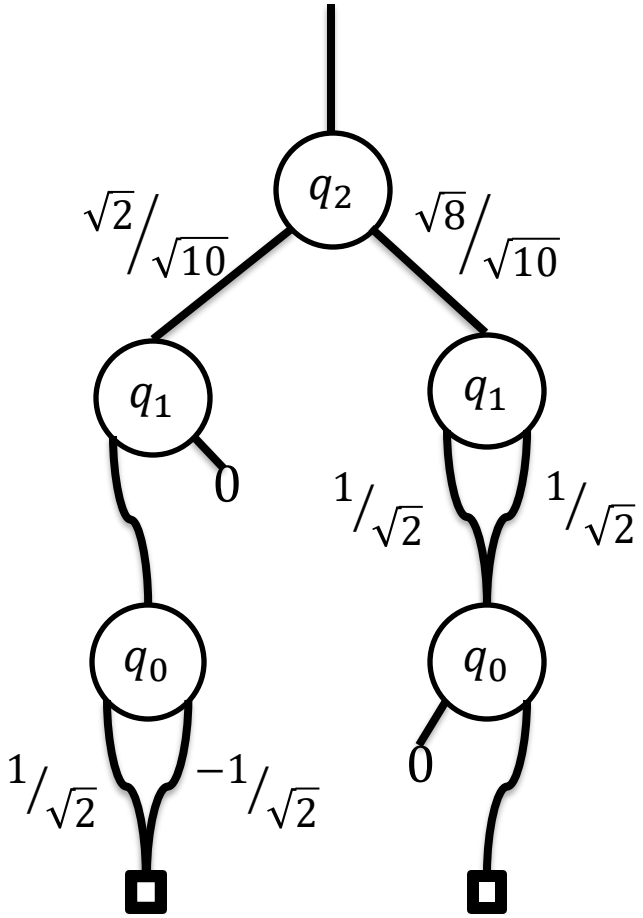
sparse **reduced**

Structure + Redundancy = Decision Diagrams



sparse **reduced** **normalized**

Structure + Redundancy = Decision Diagrams



sparse **reduced** **normalized**

Matrices

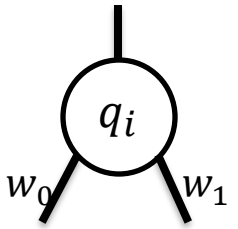
$$\left[\begin{array}{c|c} U_{00} & U_{10} \\ \hline U_{01} & U_{11} \end{array}\right]$$

mqt-core

Multiplication

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

Sampling

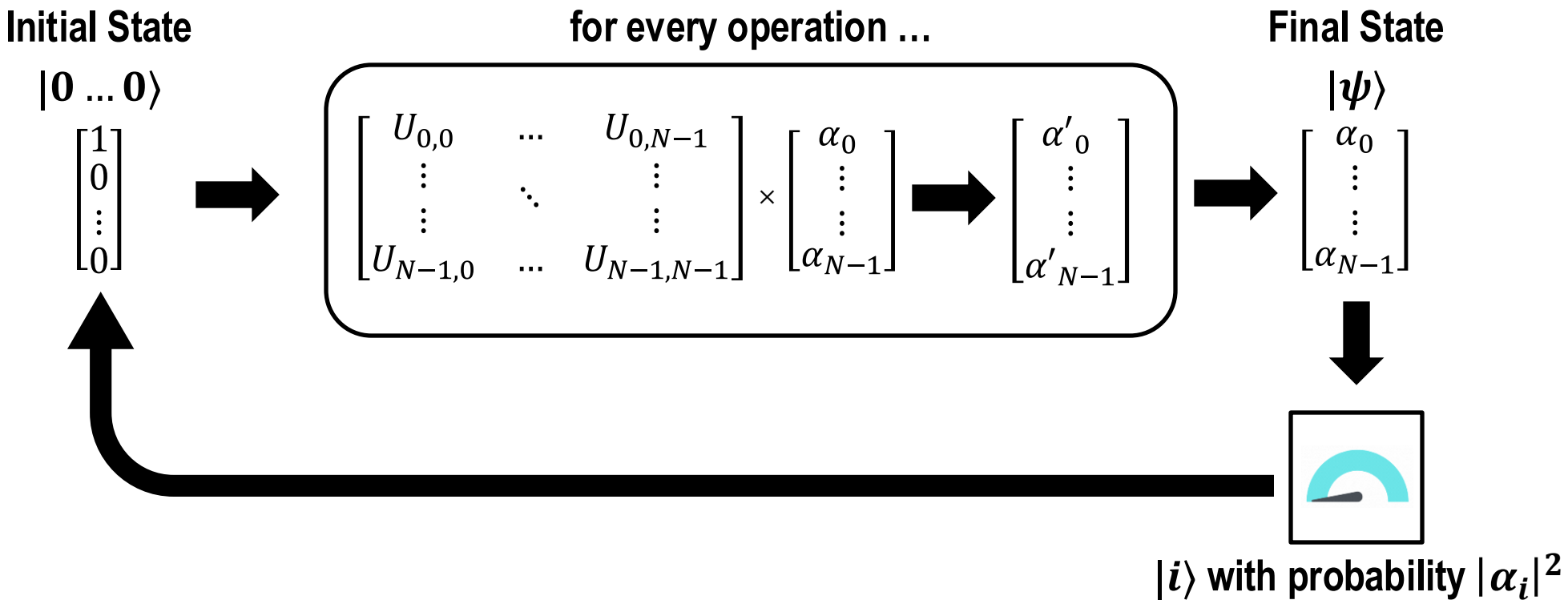


$$P(q_i = |0\rangle) = |w_0|^2$$

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

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

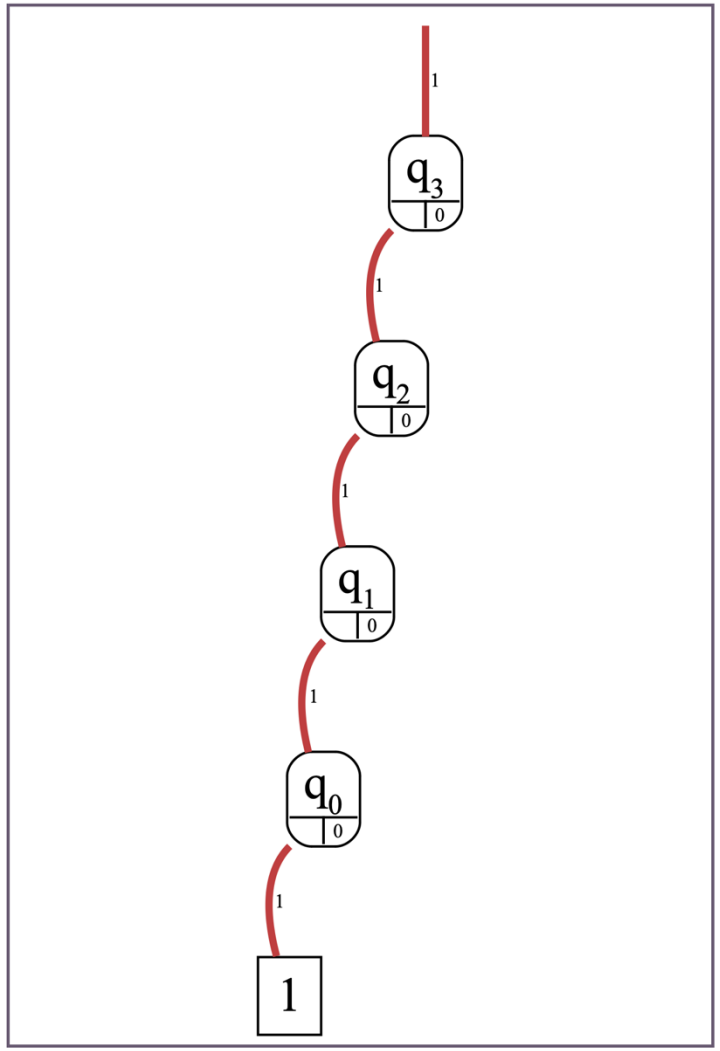
- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*

Algorithm

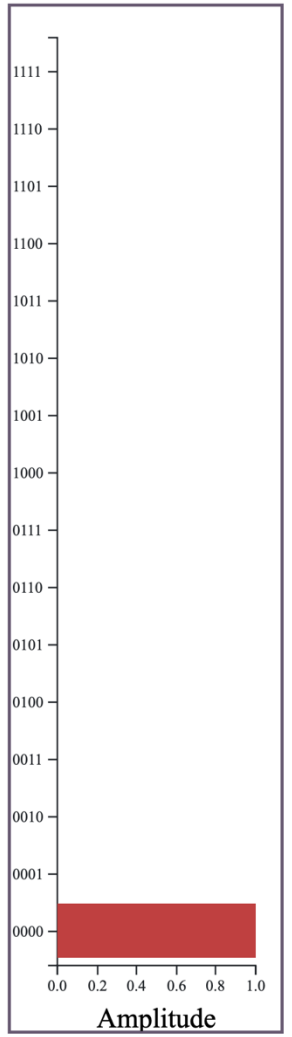
```
gate oracle q0, q1, q2, flag {
    mcphase(pi) q0, q1, q2, flag;
}
gate diffusion q0, q1, q2 {
    h q0; h q1; h q2;
    x q0; x q1; x q2;
    h q2;
    ccx q0, q1, q2;
    h q2;
    x q2; x q1; x q0;
    h q2; h q1; h q0;
}

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

Feel Free to Follow Along

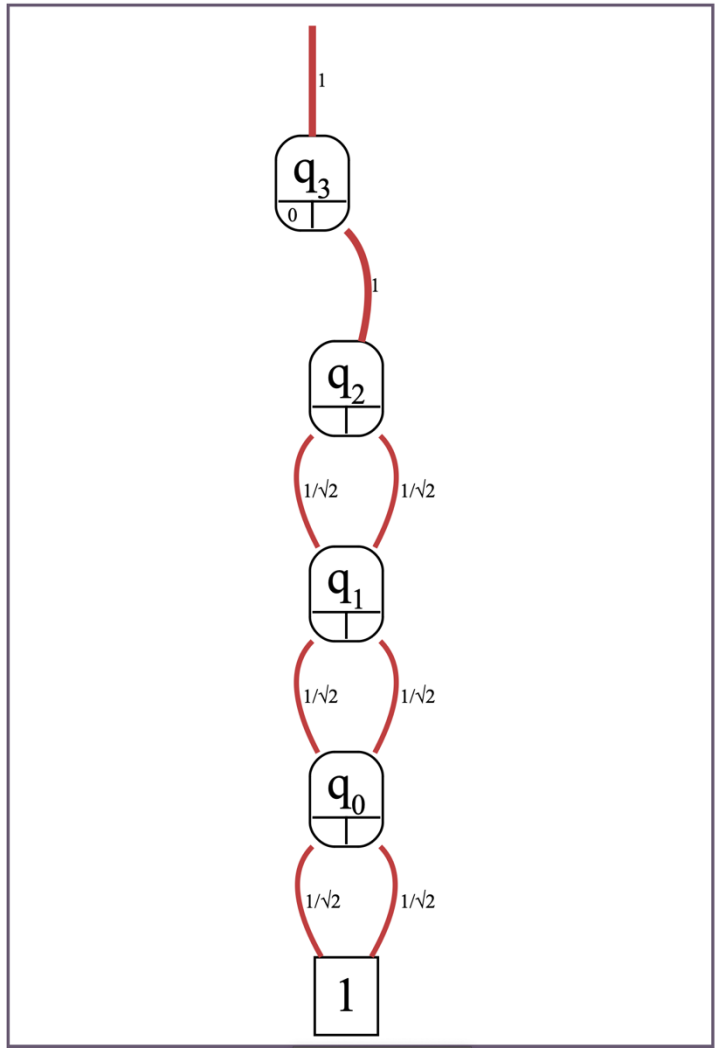
- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*

Algorithm

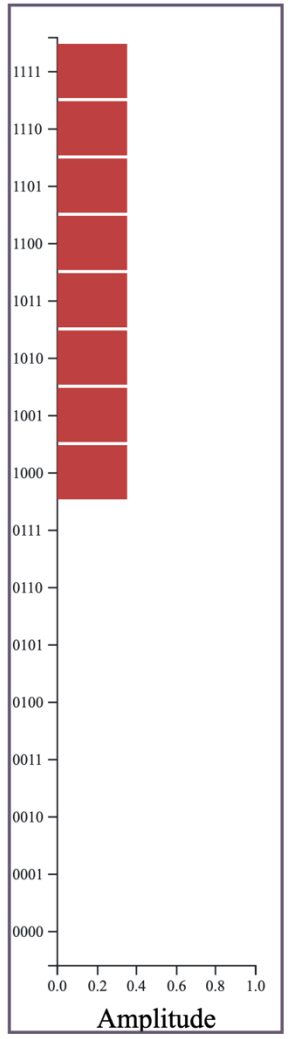
```
gate oracle q0, q1, q2, flag {
    mcphase(pi) q0, q1, q2, flag;
}
gate diffusion q0, q1, q2 {
    h q0; h q1; h q2;
    x q0; x q1; x q2;
    h q2;
    ccx q0, q1, q2;
    h q2;
    x q2; x q1; x q0;
    h q2; h q1; h q0;
}

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

Feel Free to Follow Along

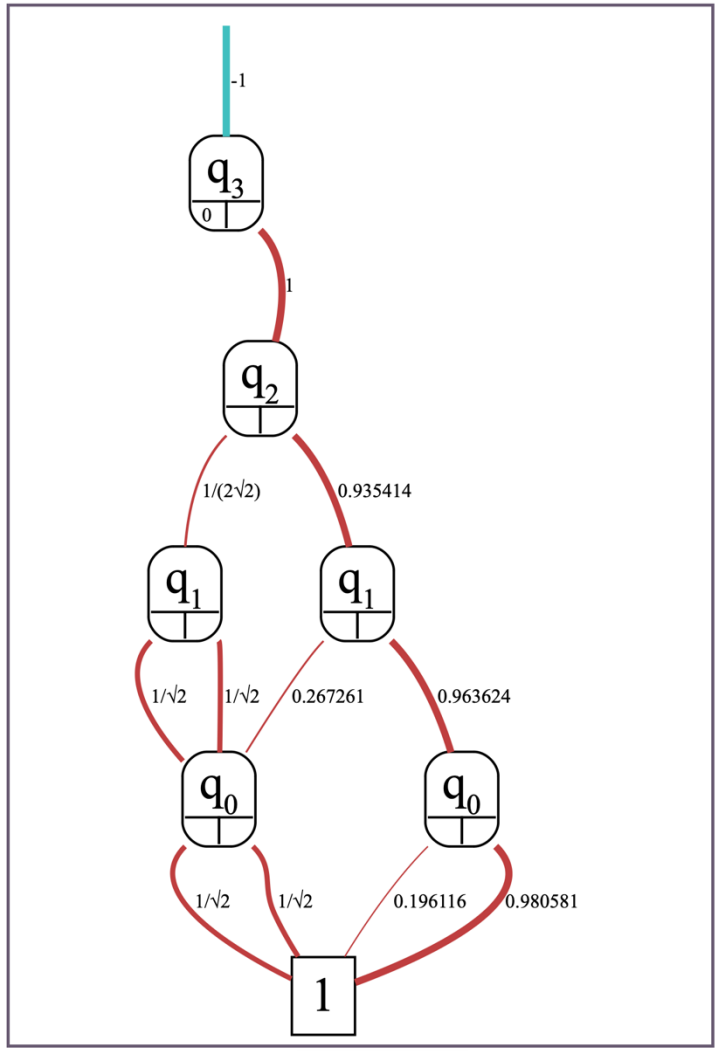
- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*

Algorithm

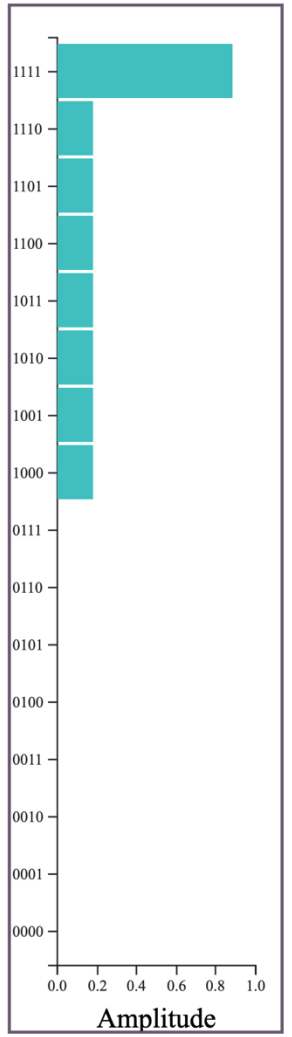
```
gate oracle q0, q1, q2, flag {
    mcphase(pi) q0, q1, q2, flag;
}
gate diffusion q0, q1, q2 {
    h q0; h q1; h q2;
    x q0; x q1; x q2;
    h q2;
    ccx q0, q1, q2;
    h q2;
    x q2; x q1; x q0;
    h q2; h q1; h q0;
}

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

Feel Free to Follow Along

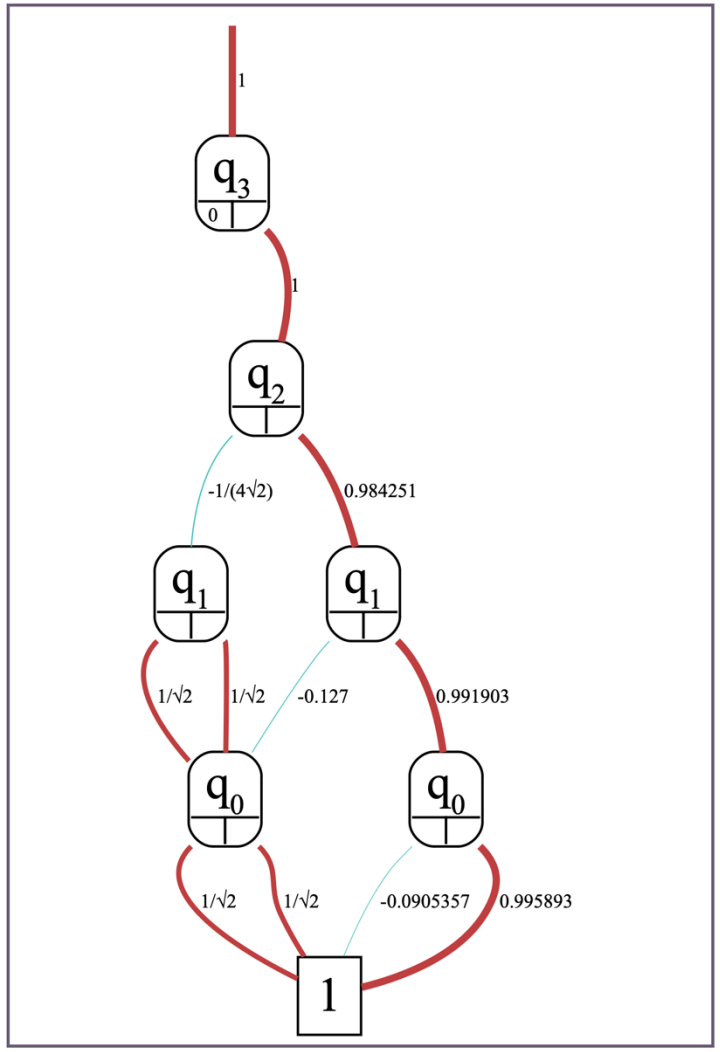
- *Go to the website below*
- *Select the "Simulation" tab*
- *Choose the "Grover (4 qubits)" example*

Algorithm

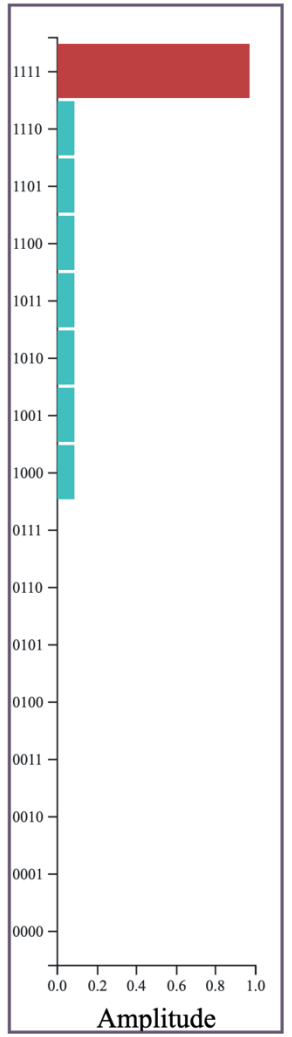
```
gate oracle q0, q1, q2, flag {
    mcphase(pi) q0, q1, q2, flag;
}
gate diffusion q0, q1, q2 {
    h q0; h q1; h q2;
    x q0; x q1; x q2;
    h q2;
    ccx q0, q1, q2;
    h q2;
    x q2; x q1; x q0;
    h q2; h q1; h q0;
}

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

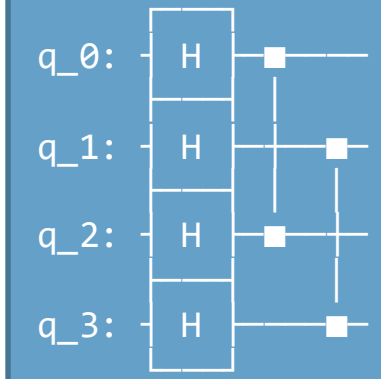
https://github.com/cda-tum/mqt-ddsim or simply **`pip install mqt.ddsim`**

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.**"*

142 ★ on GitHub, **308k Downloads** from PyPI

Practical Part I: Simulation

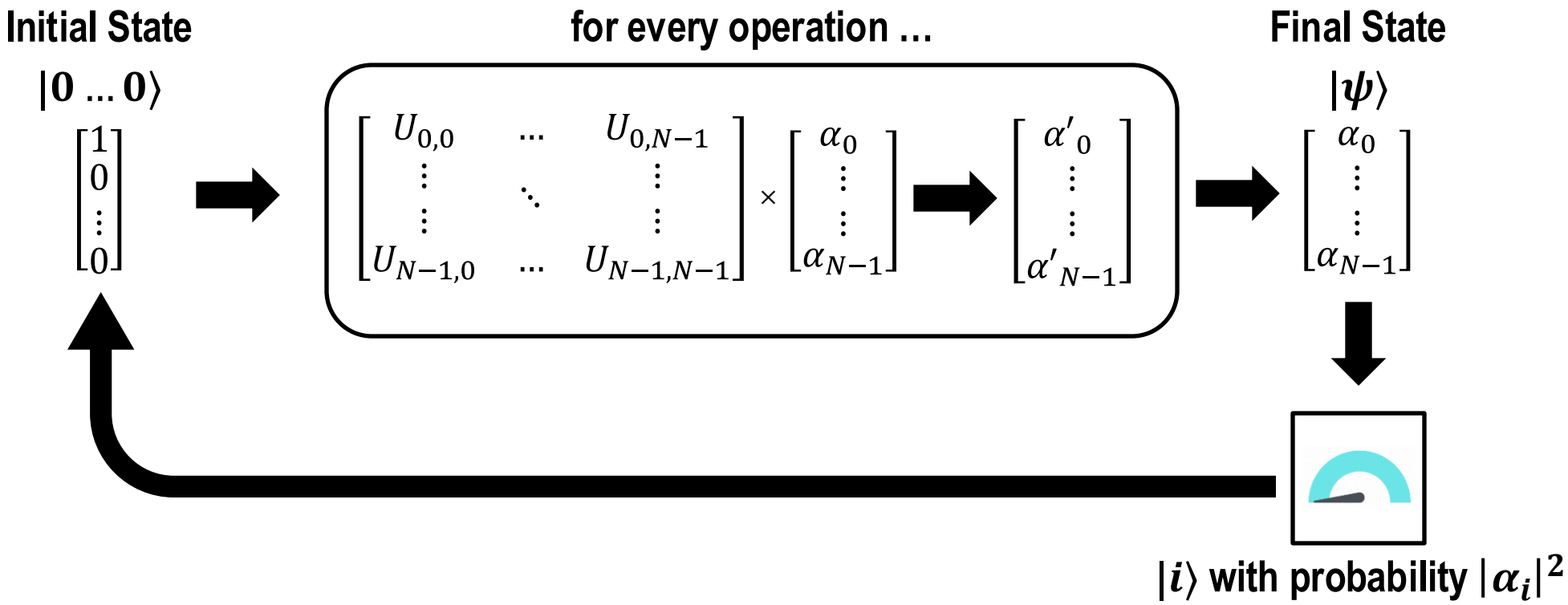
Tutorial Setup

- Clone the repository: **https://github.com/cda-tum/mqt**
  - git clone https://github.com/cda-tum/mqt
  - cd mqt

- Check out the **tutorial-SC24** branch
  - git switch tutorial-SC24

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

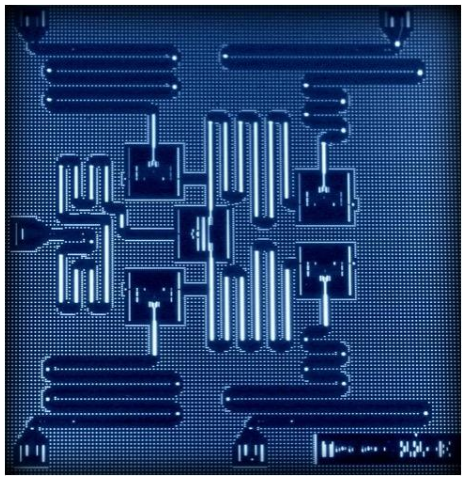
Quantum Computing



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

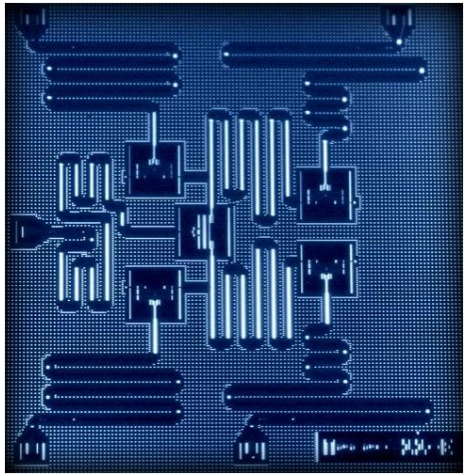
```
from qiskit import ↩
QuantumCircuit

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

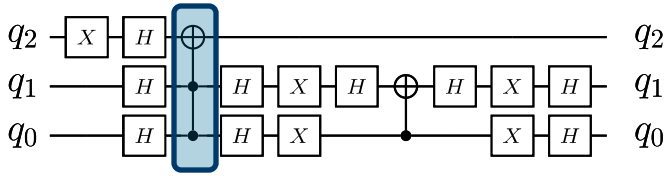
Limited Connectivity

Mapping

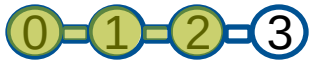
Limited Fidelity and Coherence

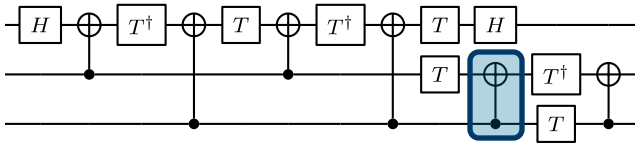
Optimizations

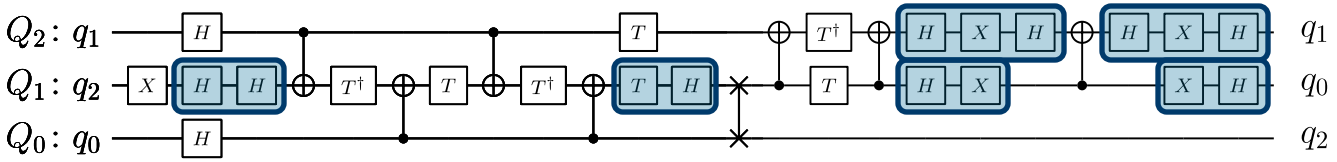
Actual Realization

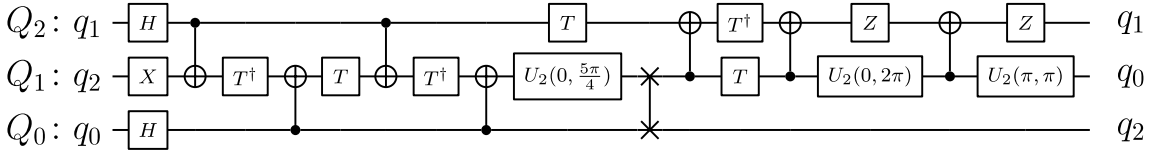


Target Device



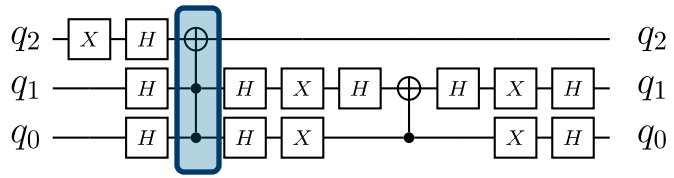


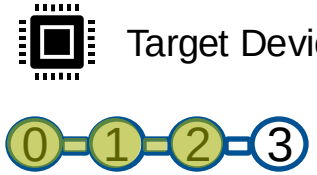


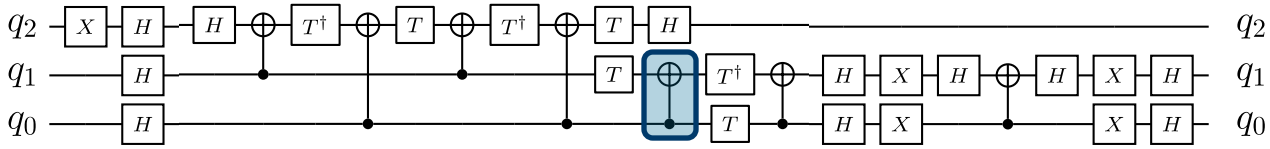


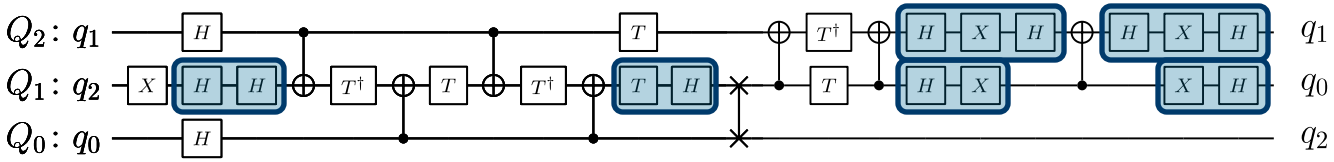
Compilation of Quantum Circuits

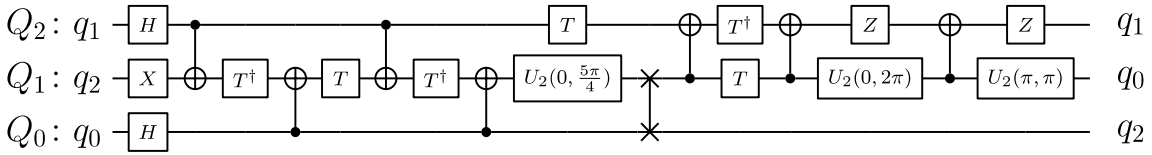
- Conceptional algorithm
- Limited Gate Set
- Synthesis
- **Limited Connectivity**
- **Mapping**
- Limited Fidelity and Coherence
- Optimizations
- Actual Realization



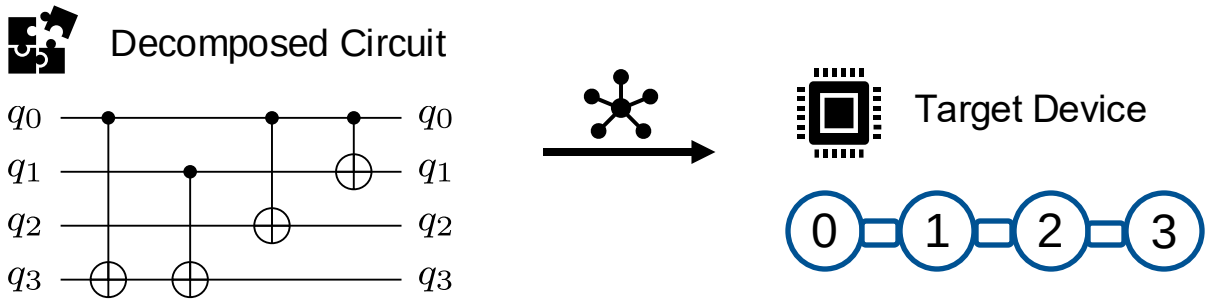






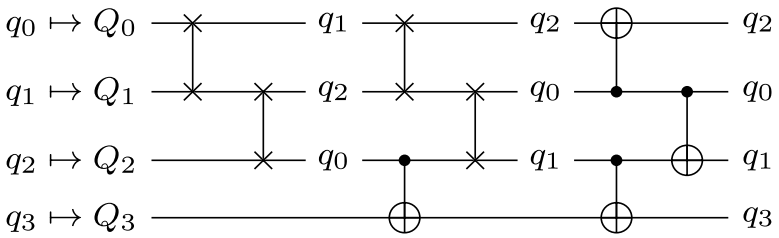


Mapping of Quantum Circuits

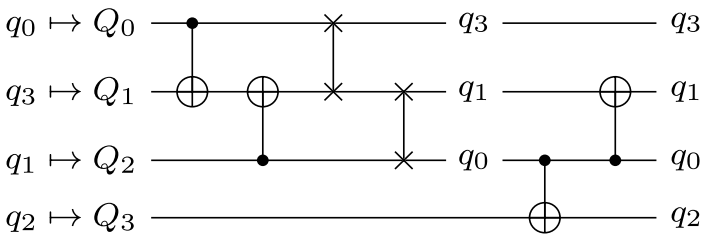


- Different approaches for mapping offer trade-off between runtime vs quality of result

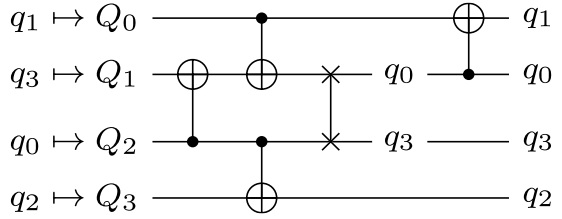
Naive Approach



Heuristic Approach



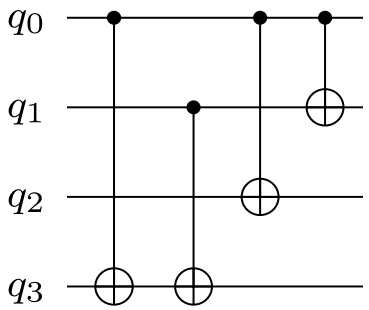
Optimal Approach

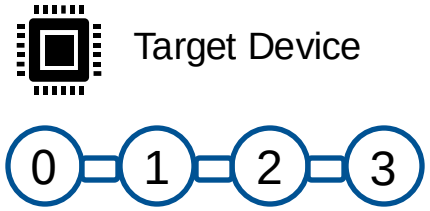


Finding an optimal solution is NP-complete!

Optimal Mapping of Quantum Circuits

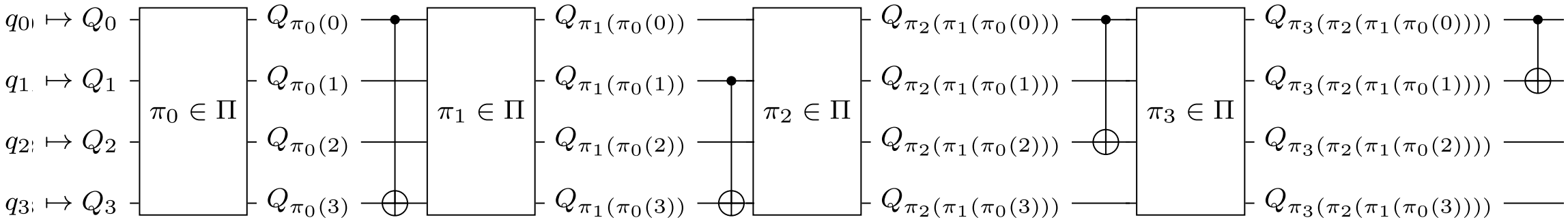
- Different Objectives: gate-count, depth, fidelity

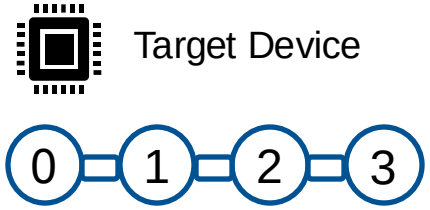




Optimal Mapping of Quantum Circuits

- Different Objectives: gate-count, depth, fidelity

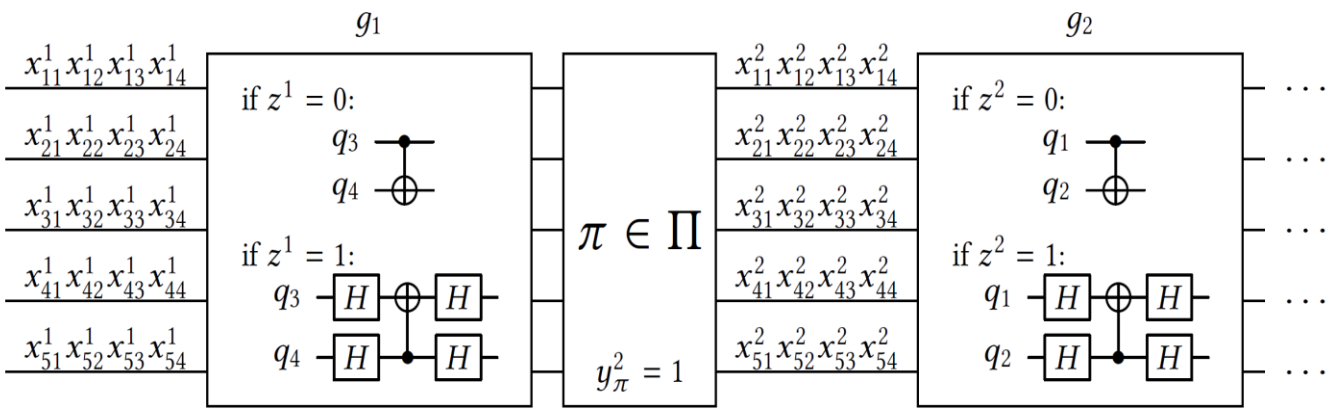




- Valid mapping makes every gate executable
- Permutations are eventually realized as sequences of swaps
- Goal: minimize the number of swaps

Optimal Mapping of Quantum Circuits

Symbolic Encoding and Powerful Reasoning Engines



$$\bigwedge_{k=1}^{|G|}\left(\bigwedge_{j=1}^{n}\left(\sum_{i=1}^{m} x_{ij}^k = 1\right) \wedge \bigwedge_{i=1}^{m}\left(\sum_{j=1}^{n} x_{ij}^k \leq 1\right)\right)$$

$$\bigwedge_{g_k = CNOT_k(q_c,q_t)\in G}\left(\bigvee_{(p_i,p_j)\in CM}(x_{ic}^k \wedge x_{jt}^k) \vee (x_{it}^k \wedge x_{jc}^k)\right)$$

$$\bigwedge_{k=2}^{|G|}\left(\bigwedge_{\pi\in\Pi}\left(\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n}\left(x_{ij}^{k-1} = x_{\pi(i)j}^k\right)\right) \Leftrightarrow y_\pi^k\right)$$

$$\mathcal{F} = \sum_{k=2}^{|G|}\sum_{\pi\in\Pi}(7 \cdot \text{swaps}(\pi)\, y_\pi^k) + \sum_{k=1}^{|G|}(4 \cdot z^k)$$

- Valid mapping makes every gate executable
- Permutations are eventually realized as sequences of swaps
- Goal: minimize the number of swaps

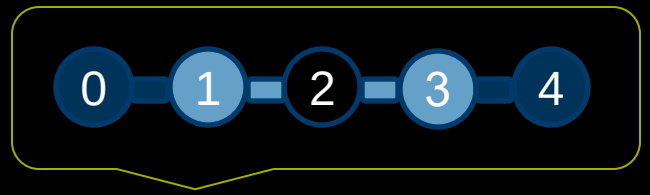
R. Wille, L. Burgholzer, and A. Zulehner.
Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations.
In Design Automation Conference (DAC). 2019.

MQT QMAP

https://github.com/cda-tum/mqt-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."

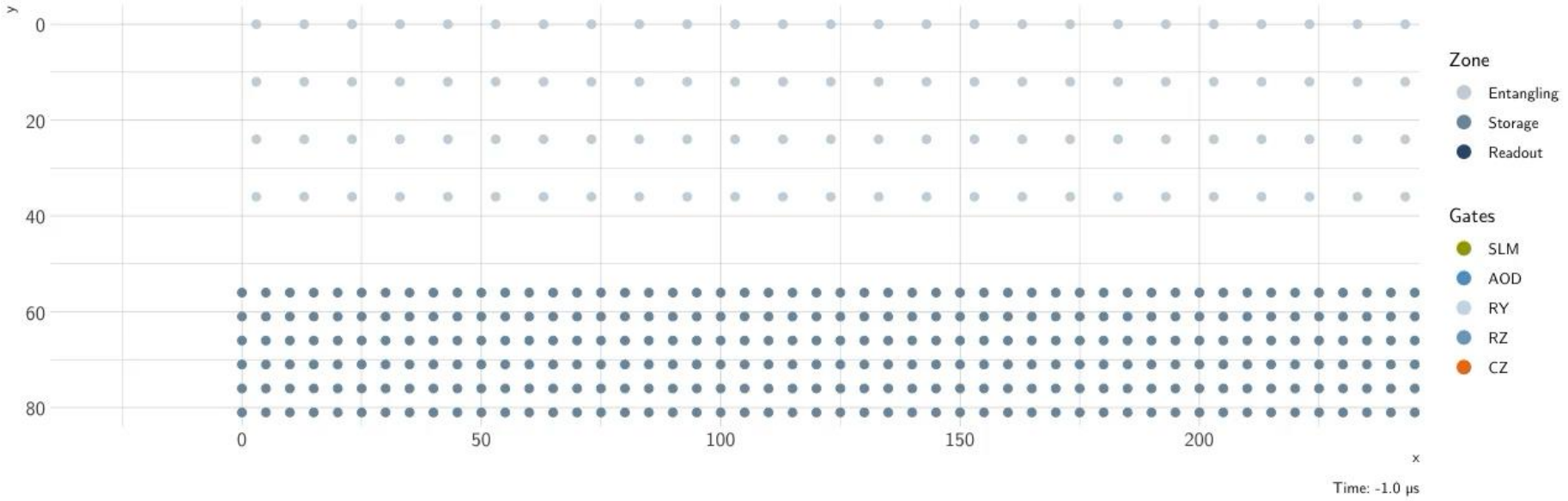
MUNICH QUANTUM TOOLKIT

PlanQK

99 ★ on GitHub, **315k Downloads** from PyPI

Practical Part II: Compilation

Compilation for Zoned Neutral Atom Architectures



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

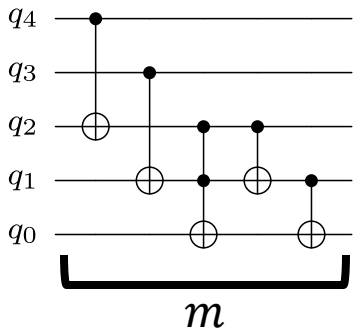
"...a **significant fraction** of these bugs (**39.9%**) are **quantum-specific**… bugs occur particularly often in components that **represent, compile, and optimize** quantum programming abstractions."

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

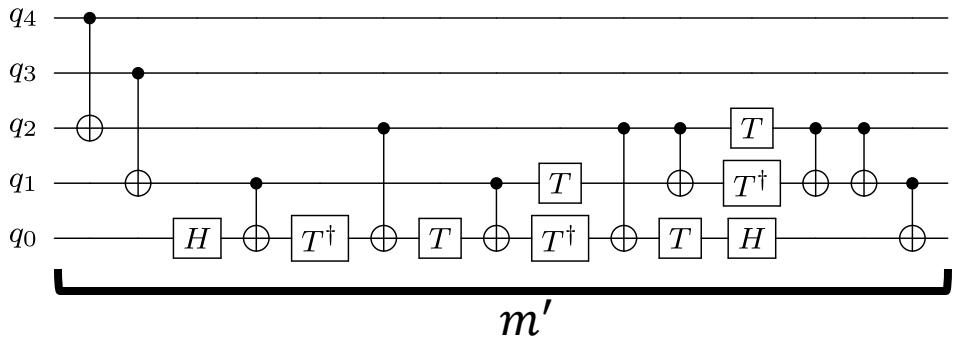
How do you verify that the resulting circuit still realizes the originally intended functionality?

Verification of Quantum Circuits

Conceptional algorithm G



Actual Realization G'

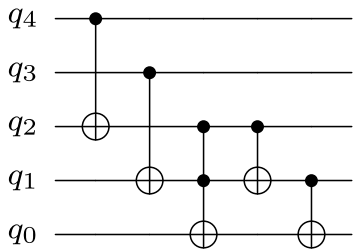


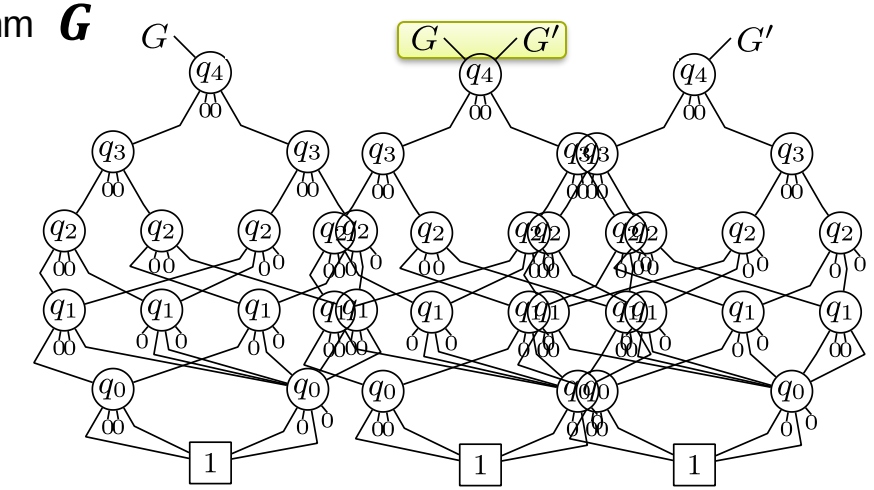
How do you verify that the resulting circuit still realizes the originally intended functionality?

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

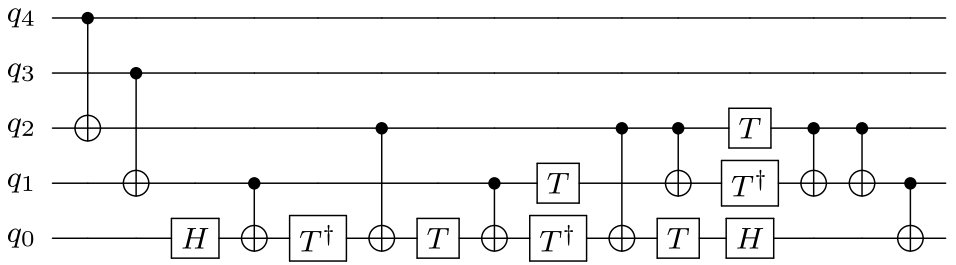
Verification of Quantum Circuits

Conceptional algorithm G





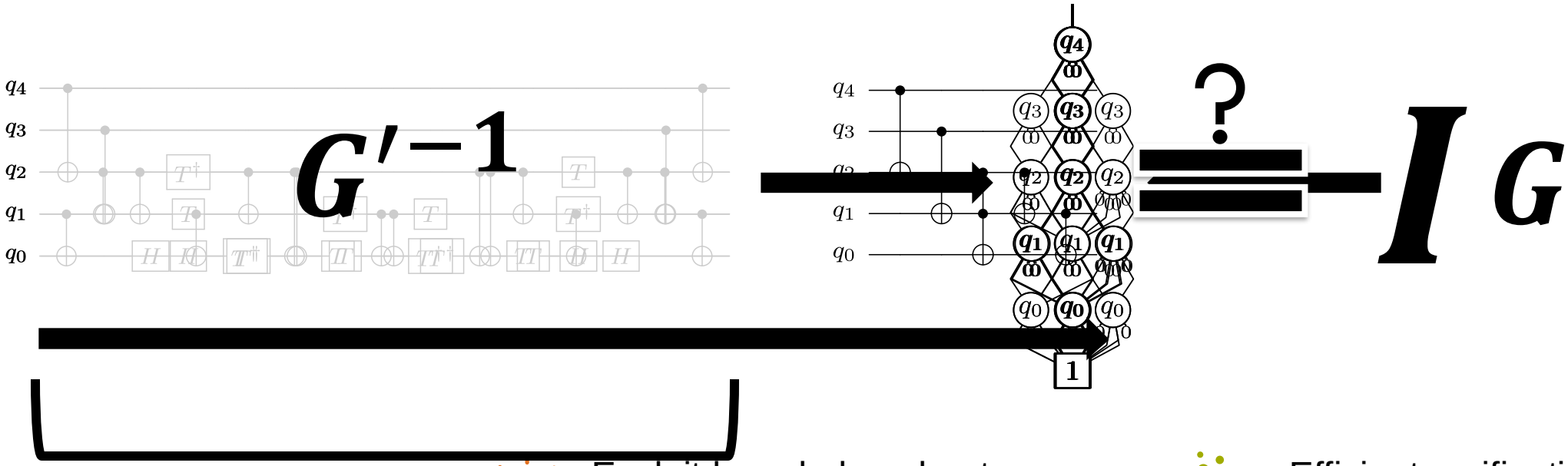
Actual Realization G'



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



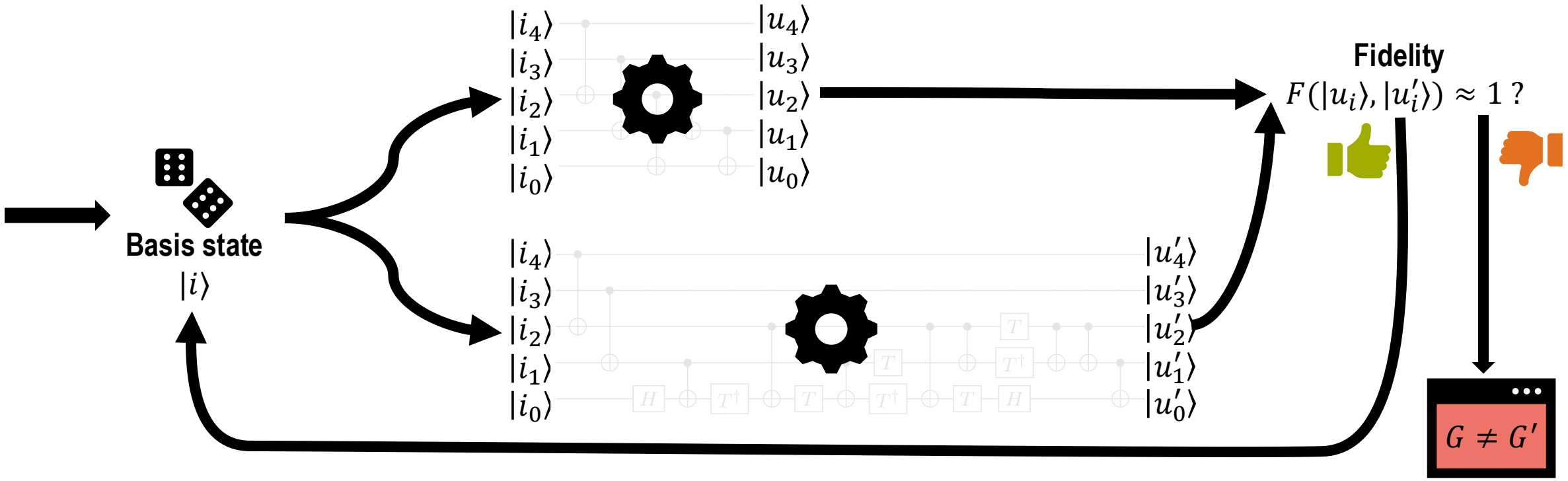
Strategies?

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 👎 ?

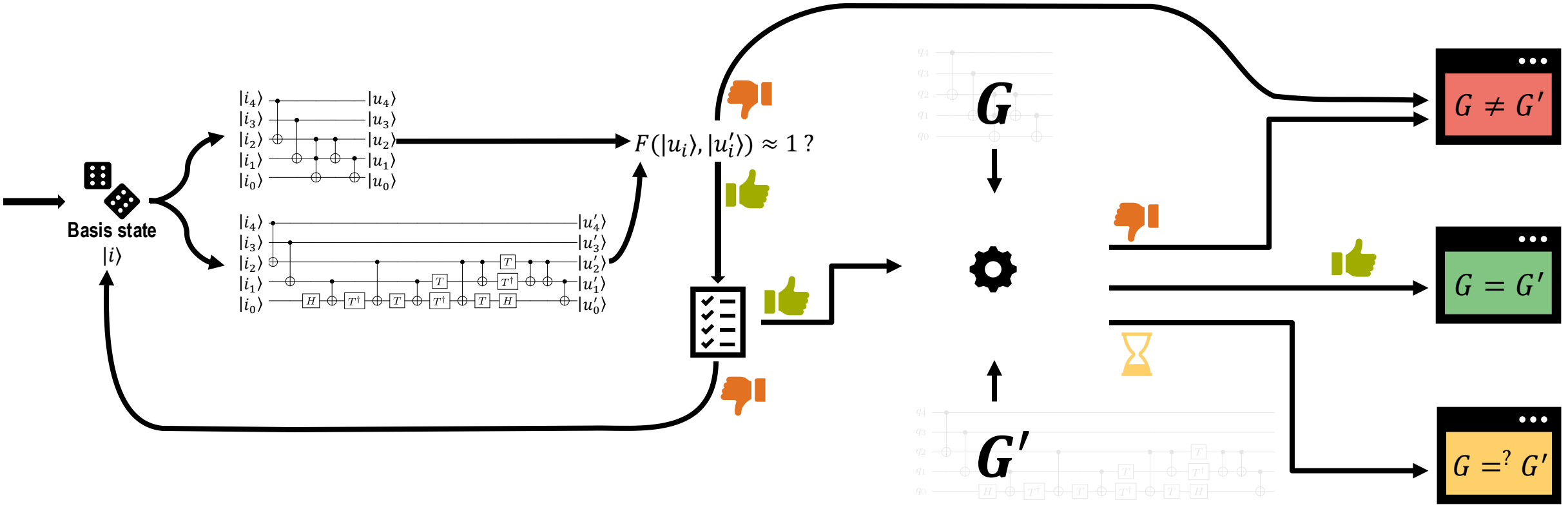
Reversibility reduces masking effects drastically

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

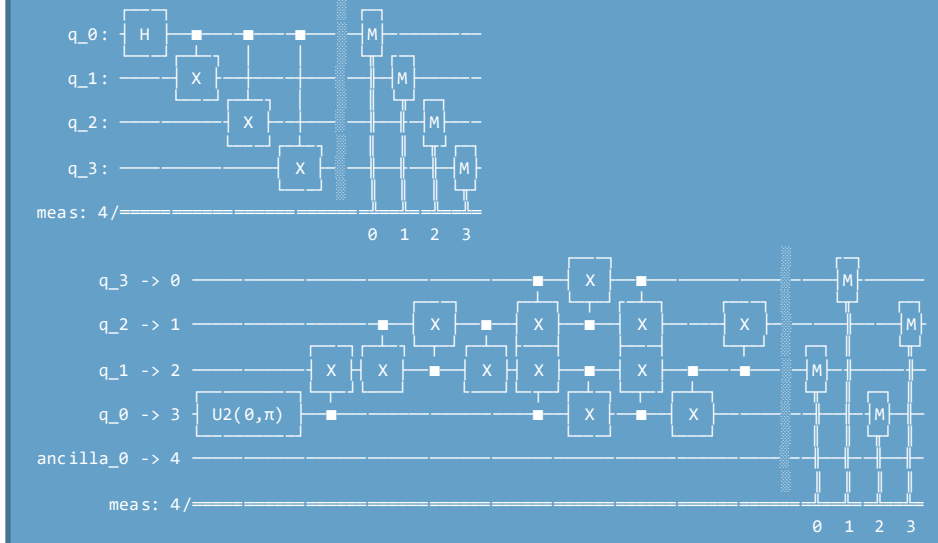
github.com/cda-tum/mqt-qcec or simply **`pip install mqt.qcec`**

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."

95 ★ on GitHub, **505k Downloads** from PyPI

Practical Part III: Verification

Quantum Computing

Core Methods and Data Structures

mqt-core

Python

```
from qiskit import ↩
QuantumCircuit

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

Compilation of Quantum Circuits

mqt-qmap

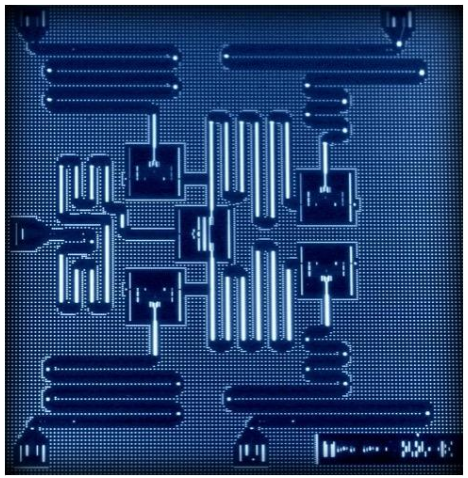
Compiler

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];
```



Classical Simulation of Quantum Circuits

mqt-ddsim

1110

0110

0001

1001

Verification of Quantum Circuits

mqt-qcec

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

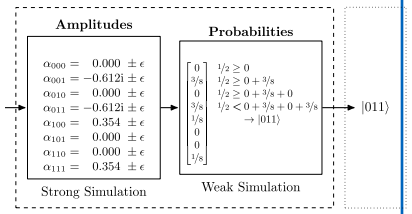
Error Correction

- 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

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/cda-tum/mqt-problemsolver

MQT Bench — Application
A Quantum Circuit Benchmark Suite
www.cda.cit.tum.de/mqtbench
github.com/cda-tum/mqt-bench

MQT Quantum Auto Optimizer — Application
An Automatic Framework for Solving Optimization Problems
github.com/cda-tum/mqt-qao

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

MQT DDSIM — Simulation
A Tool for Classical Quantum Circuit Simulation based on Decision Diagrams
github.com/cda-tum/mqt-ddsim

MQT Predictor — Compilation
A Tool for Determining Good Quantum Circuit Compilation Options
github.com/cda-tum/mqt-predictor

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

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

MQT SyReC — Compilation
A Tool for the Synthesis of Reversible Circuits/Quantum Computing Oracles
github.com/cda-tum/mqt-syrec

MQT QMAP — Compilation
A Tool for Quantum Circuit Mapping And Clifford Circuit Optimization/Synthesis
github.com/cda-tum/mqt-qmap

MQT QCEC — Verification
A Tool for Quantum Circuit Equivalence Checking
github.com/cda-tum/mqt-qcec

MQT DASQA — Hardware
A Tool for Designing Alternative Superconducting Quantum Architectures
github.com/cda-tum/mqt-dasqa

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/cda-tum/mqt-core

MQT QuSAT — Core Methods
A Tool for Encoding Quantum Computing using Satisfiability Testing (SAT) Techniques
$F \wedge (x_1 \wedge \neg x_2)$
$F \wedge (x_3 \wedge x_$
github.com/cda-tum/mqt-qusat

MQT QECC — QECC
A Tool for Quantum Error Correcting Codes
github.com/cda-tum/mqt-qecc

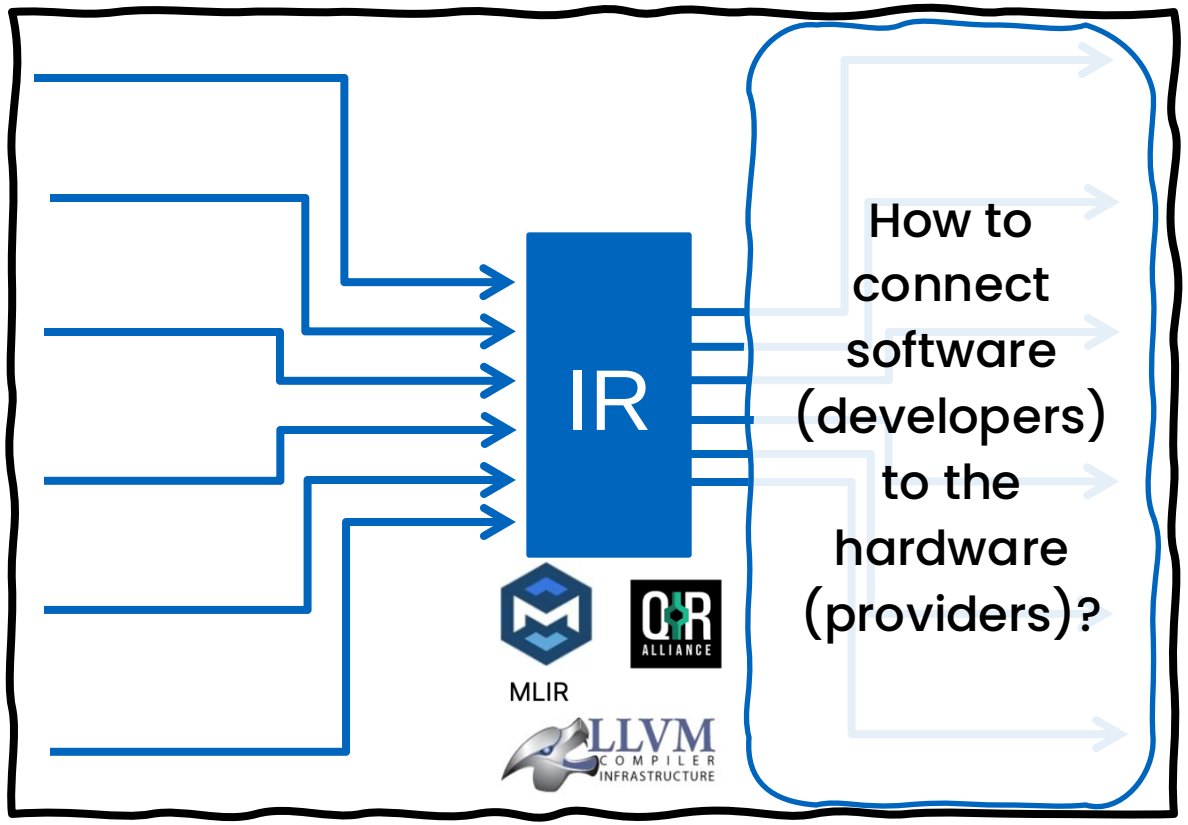
https://mqt.readthedocs.io

get involved and contribute

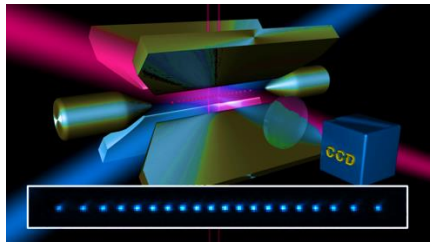
The Big Picture

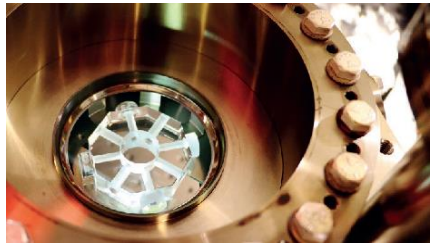
Domain Experts

Quantum Computing Software Stack

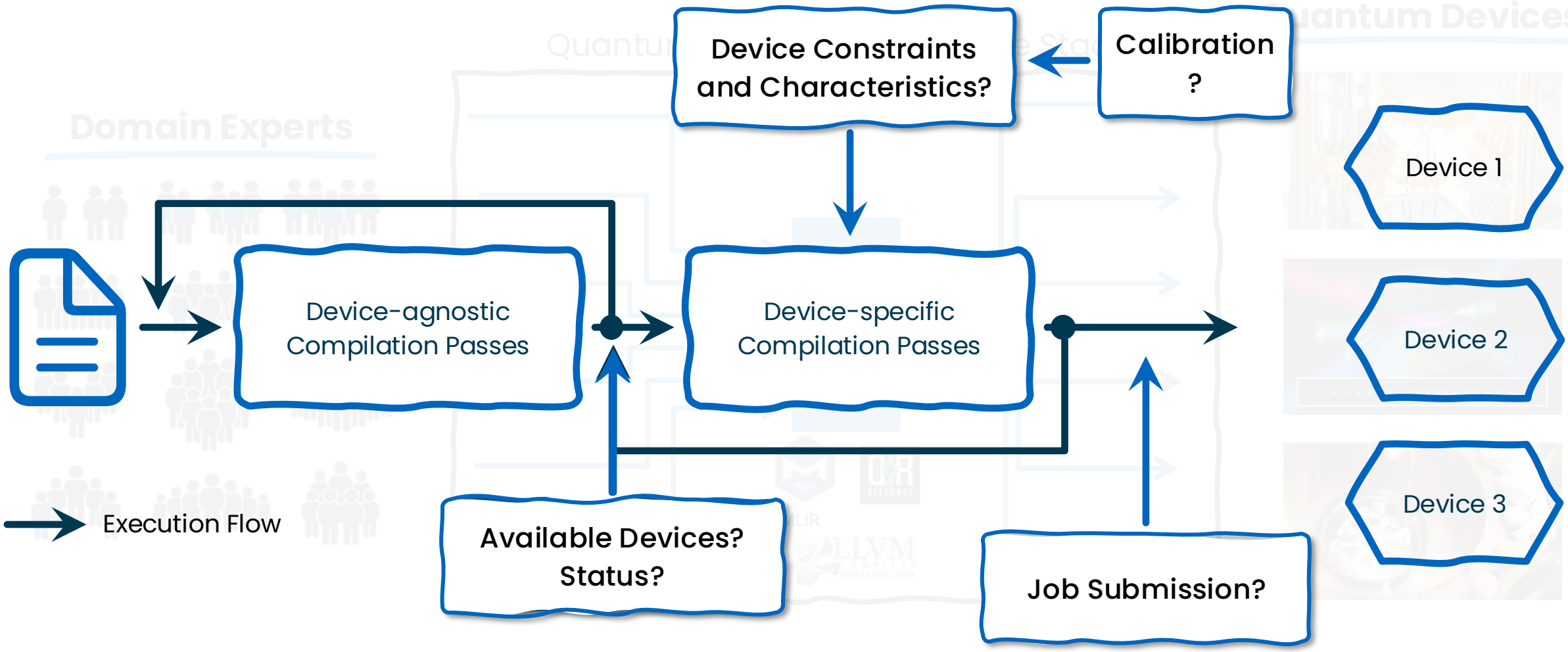


Quantum Devices

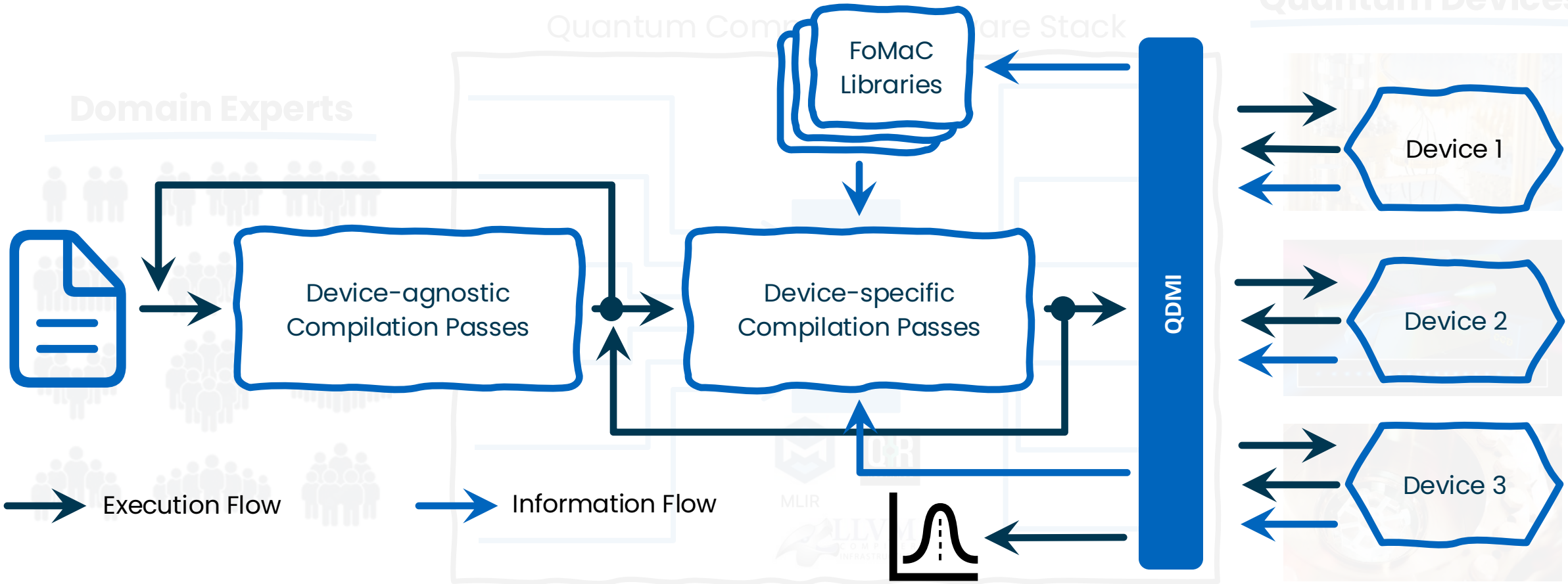




Software 🤝 Hardware



Software 🤝 Hardware



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

Munich Quantum Valley

TUM

MUNICH QUANTUM SOFTWARE

munichquantum.software