Software for Quantum Computing

Florea Ioan-Albert

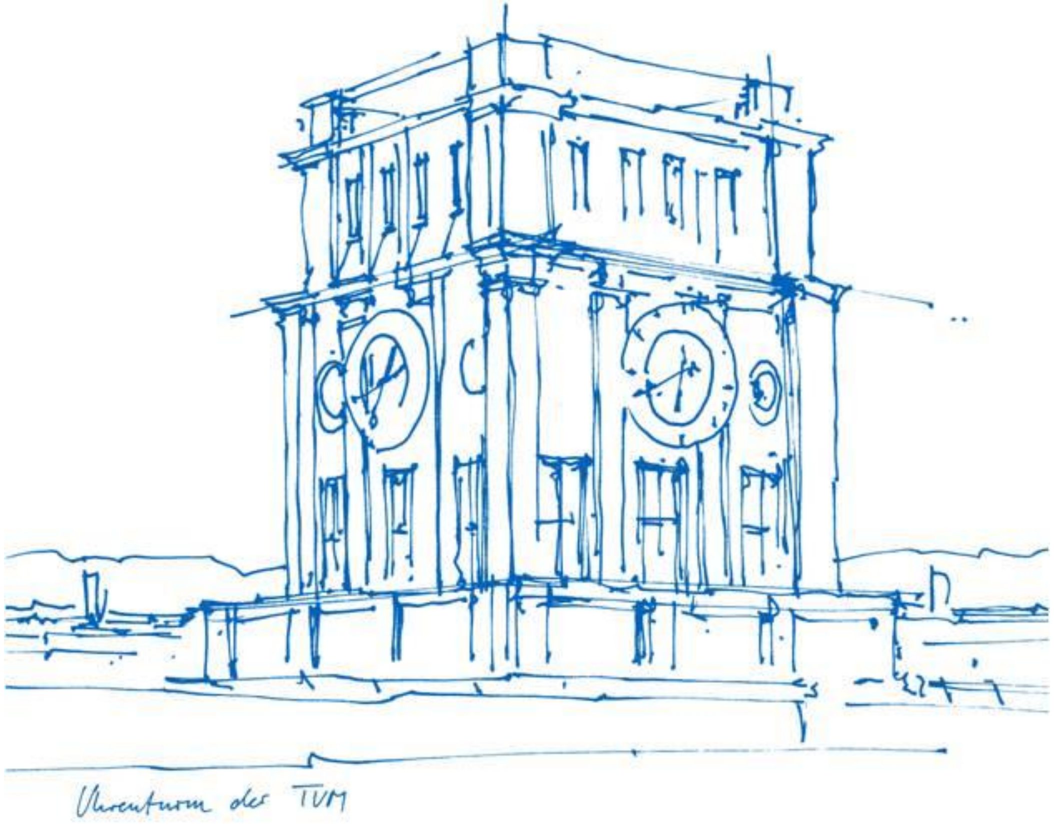
About myself…

- M.Sc. Candidate in Computer Science at TUM (attended several lectures in QC)

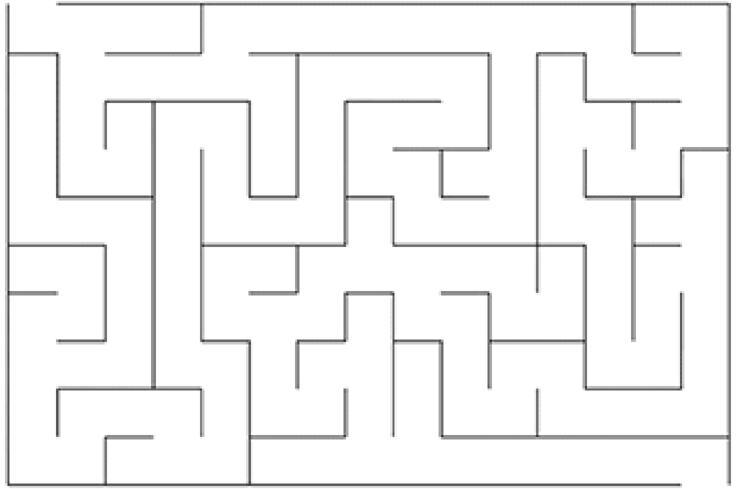
- Interdisciplinary work in Quantum Computing at the Chair of Design Automation (Prof. Robert Wille)

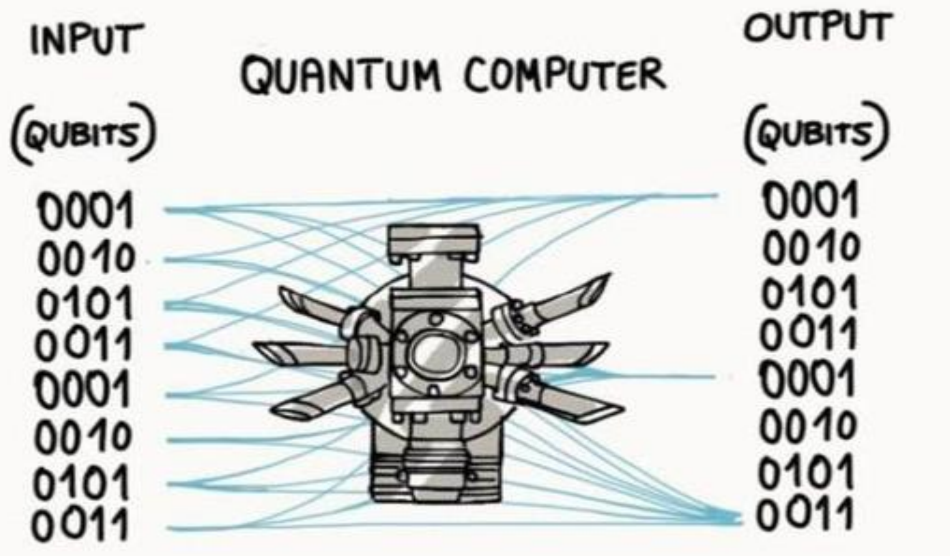
- Contributed to the open-source MUNICH QUANTUM TOOLKIT

- Co-authored a paper rewarded with a Best Paper Award at the 2024 IEEE International Conference on Quantum Software (QSW) (https://arxiv.org/abs/2406.11959)

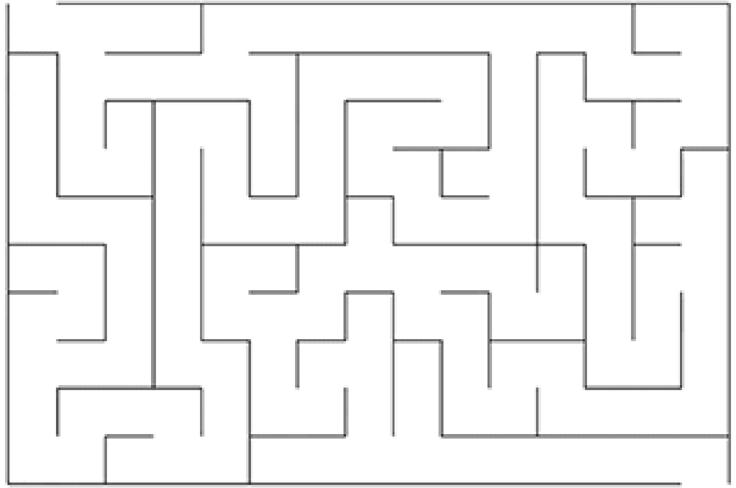


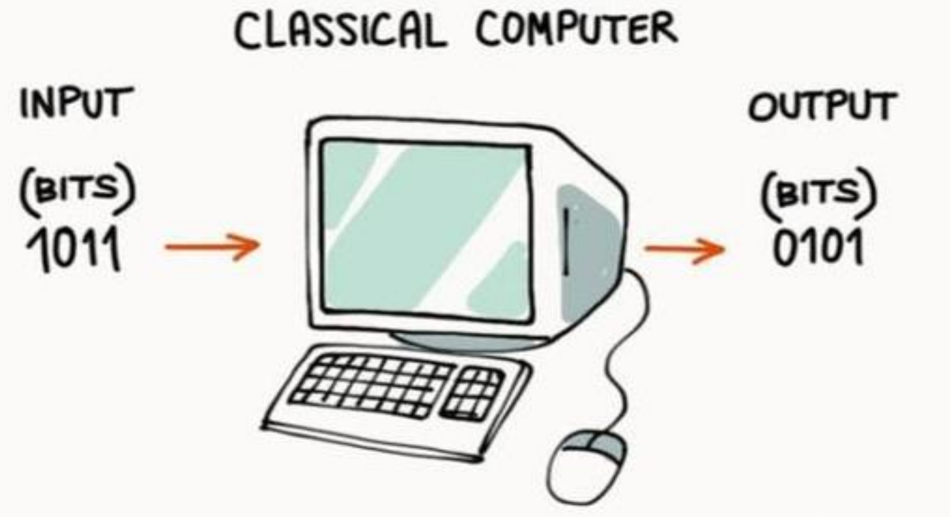
Quantum computers





Classical computers





Conventional vs. Quantum Physics

- Thus far: Newtonian Physics
  - Describes "our world" (larger than the subatomic level)
  - Based on particles
  - Completely deterministic

- Now: Quantum Mechanics
  - Describes the smallest "things" in our universe
  - Described by waves
  - Completely probabilistic

  - Allows to be in more than one state at once
  - Measurement collapses back to a single state (back to "our" world)
  - True random
  - Entanglement

Conventional vs. Quantum Physics

- Thus far: Newtonian Physics
  - Describes “our world” (larger than the subatomic level)
  - Based on particles
  - Completely deterministic

$$\mathbf{0} \qquad \text{or} \qquad \mathbf{1}$$

- Now: Quantum Mechanics
  - Describes the smallest “things” in our universe
  - Described by waves
  - Completely probabilistic

$$\alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle$$
$$\text{with} \quad |\alpha_0|^2 + |\alpha_1|^2 = 1$$

  - Allows to be in more than one state at once
  - Measurement collapses back to a single state (back to “our” world)
  - True random

Conventional vs. Quantum Computing

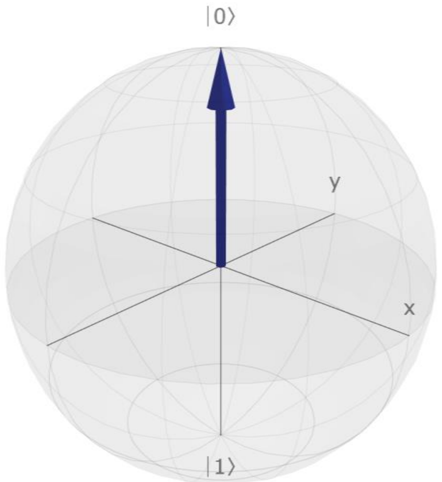
- **Classical bit:** 0 or 1
- **Qubit:** Superposition of two basis states

  $\alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle$ with $|\alpha_0|^2 + |\alpha_1|^2 = 1$
- Measurement leads to 0 or 1 with probability $|\alpha_0|^2$ or $|\alpha_1|^2$
- Represented in terms of **vectors**

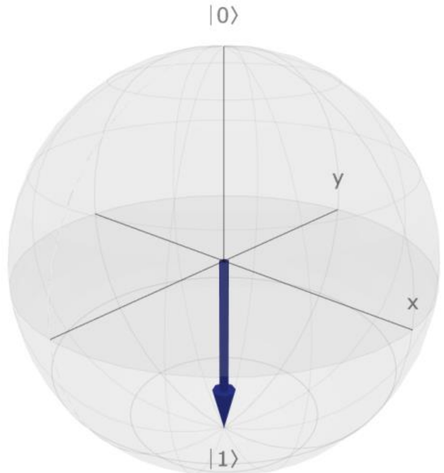
$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- **Operations** represented through unitary matrices,

  e.g. $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$



$|0\rangle$ Qubit



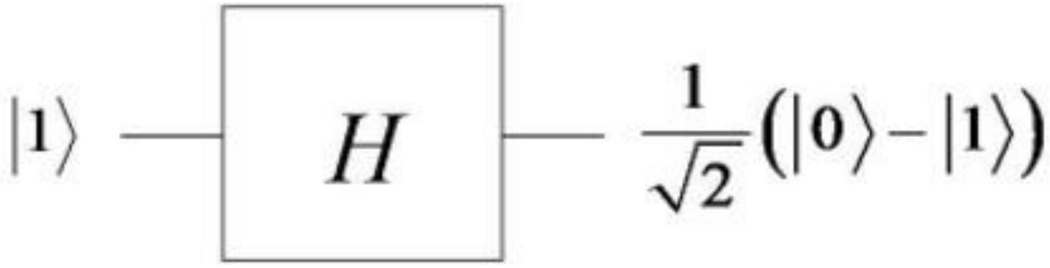
$|1\rangle$ Qubit

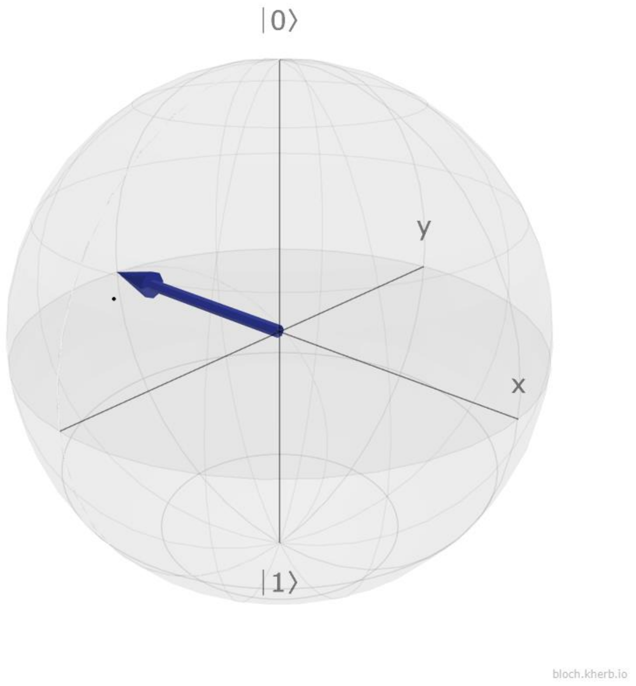
Example of Applying an Operation to a Qubit

- Applying the H gate to the |1> basis state would result in the following state:

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

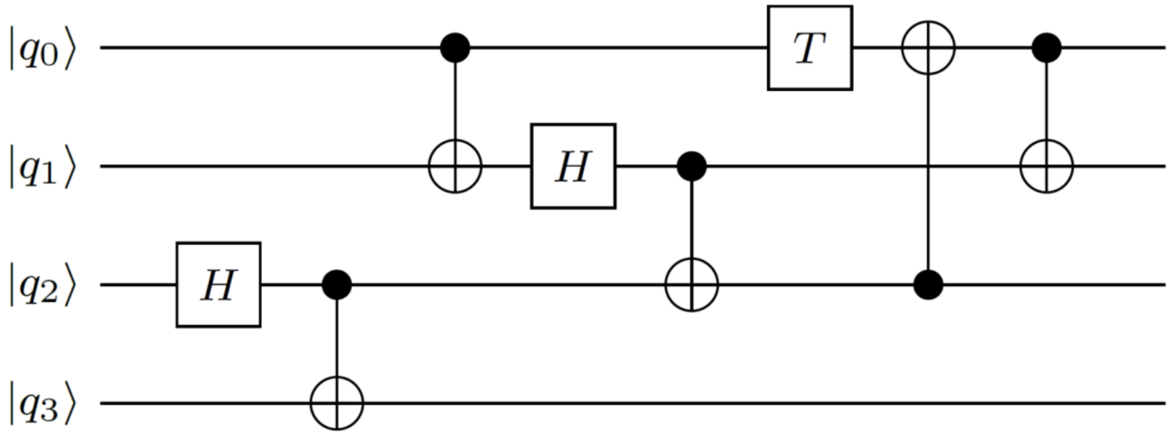
- The quantum circuit for the above computation looks as follows:



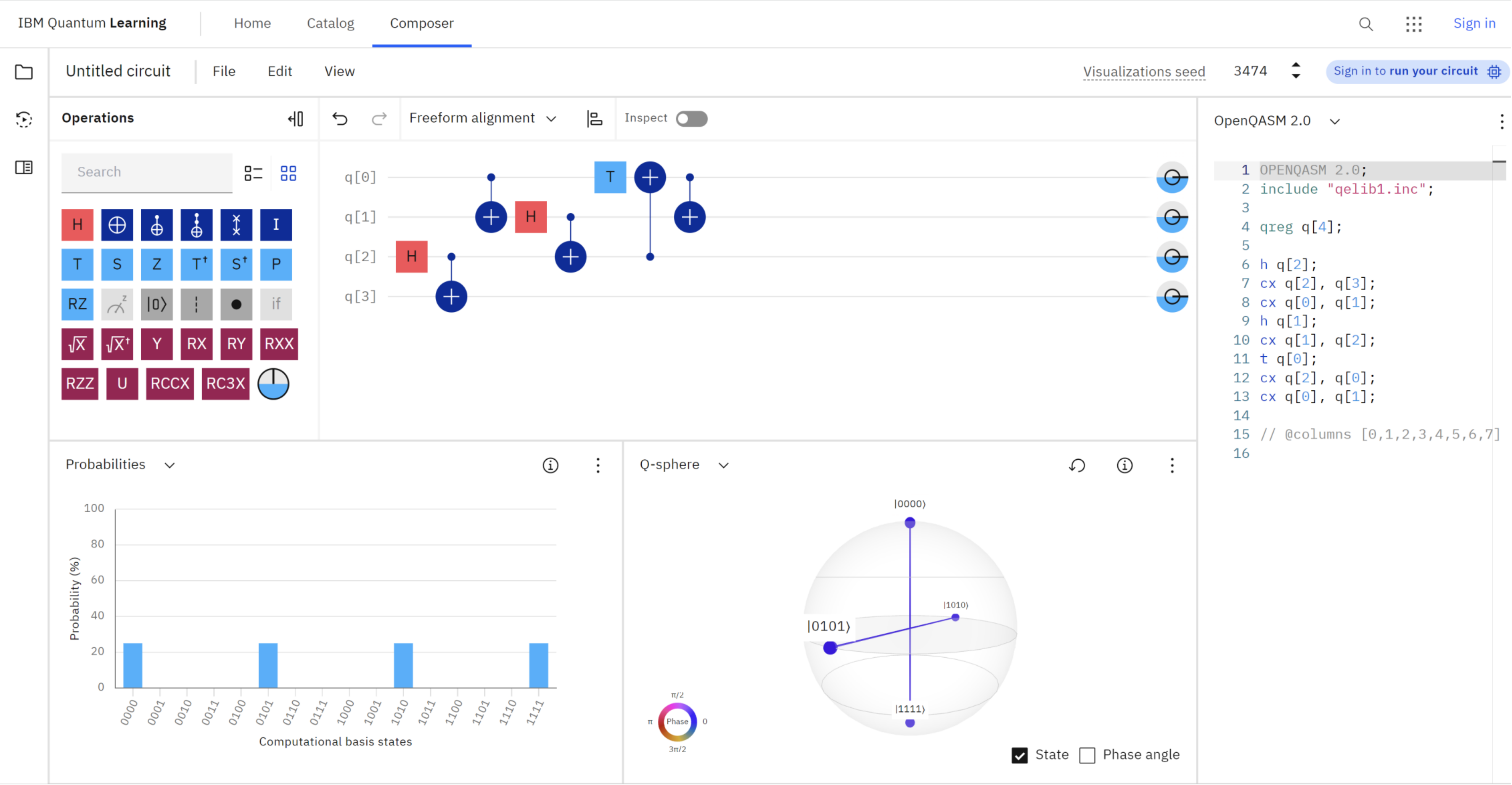


Quantum Circuits / Quantum Software

- Cascades of quantum gates working on qubits
- Quantum gates may either be unary gates or working on two gates with a control qubit (black dot) and a target qubit




```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[4];
h q[2];
cx q[2],q[3];
cx q[0],q[1];
h q[1];
cx q[1],q[2];
t q[0];
cx q[2],q[0];
cx q[0],q[1];
```


IBM Quantum Learning
Home
Catalog
Composer
Sign in
Untitled circuit
File
Edit
View
Visualizations seed
3474
Sign in to run your circuit
Operations
Freeform alignment
Inspect
Search
H
I
T
S
Z
P
RZ
|0⟩
if
√X
√X†
Y
RX
RY
RXX
RZZ
U
RCCX
RC3X
q[0]
q[1]
q[2]
q[3]
OpenQASM 2.0
OPENQASM 2.0;
include "qelib1.inc";
qreg q[4];
h q[2];
cx q[2], q[3];
cx q[0], q[1];
h q[1];
cx q[1], q[2];
t q[0];
cx q[2], q[0];
cx q[0], q[1];
// @columns [0,1,2,3,4,5,6,7]
Probabilities
Probability (%)
100
80
60
40
20
0
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
Computational basis states
Q-sphere
|0000⟩
|1010⟩
|0101⟩
|1111⟩
π/2
π
Phase
0
3π/2
State
Phase angle

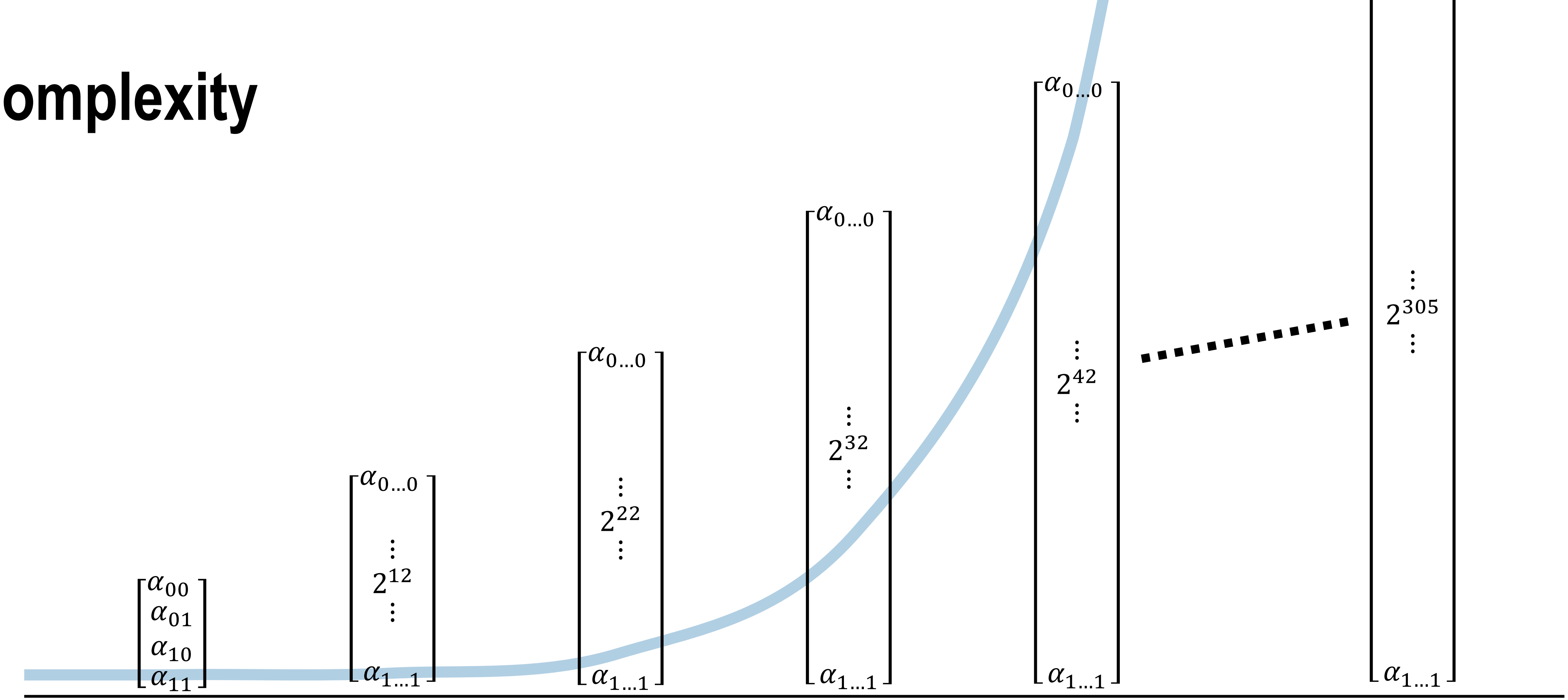
The Basis of the Software for Quantum Computing…

- …are vectors and matrices!
- Can be represented by 1- or 2-dimensional arrays (problem: exponential complexity)
- Solution: Decision Diagrams

$$\begin{bmatrix} m_{11} & m_{12} & \dots & m_{12^n} \\ m_{21} & m_{22} & \dots & m_{22^n} \\ m_{31} & m_{32} & \dots & m_{32^n} \\ m_{41} & m_{42} & \dots & m_{42^n} \\ \dots & \dots & \dots & \dots \\ m_{2^n1} & m_{2^n2} & \dots & m_{2^n2^n} \end{bmatrix}$$

$$\begin{bmatrix} \alpha_{0\dots000} \\ \alpha_{0\dots001} \\ \alpha_{0\dots010} \\ \alpha_{0\dots011} \\ \dots \\ \alpha_{1\dots111} \end{bmatrix}$$

Complexity



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

Classical Simulation of Quantum Circuits

Quantum Operation **Quantum State** **Evolved Quantum State**

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

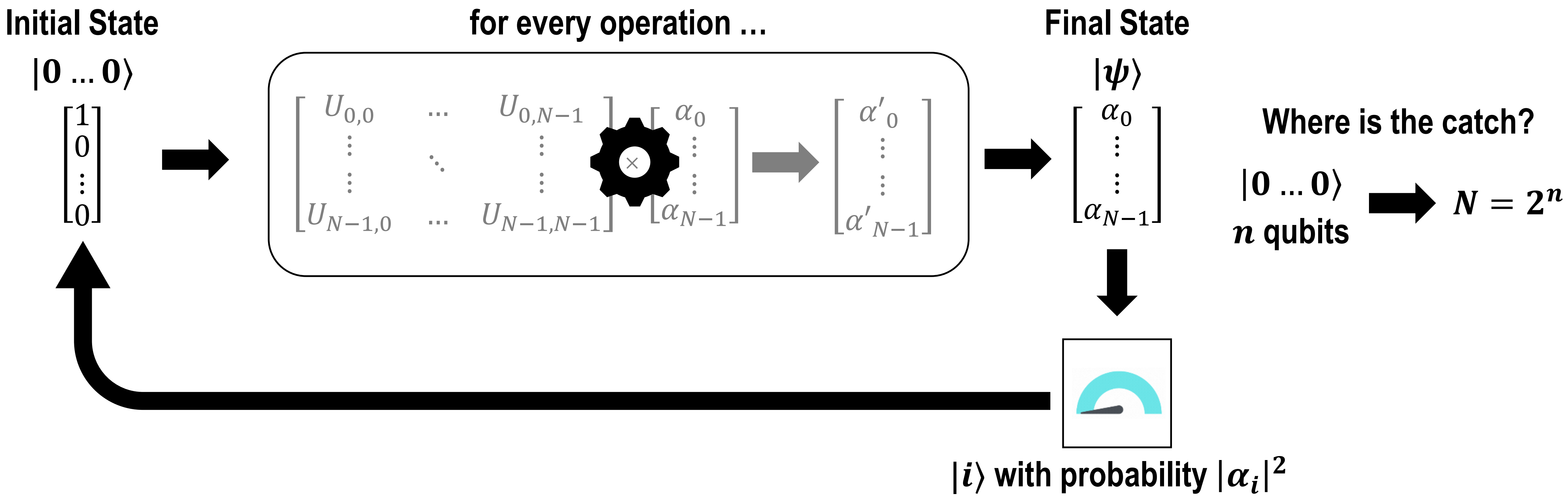
$U_{i,j} \in \mathbb{C}$

$U^\dagger U = I$

$\alpha_i \in \mathbb{C}$

$\sum |\alpha_i|^2 = 1$

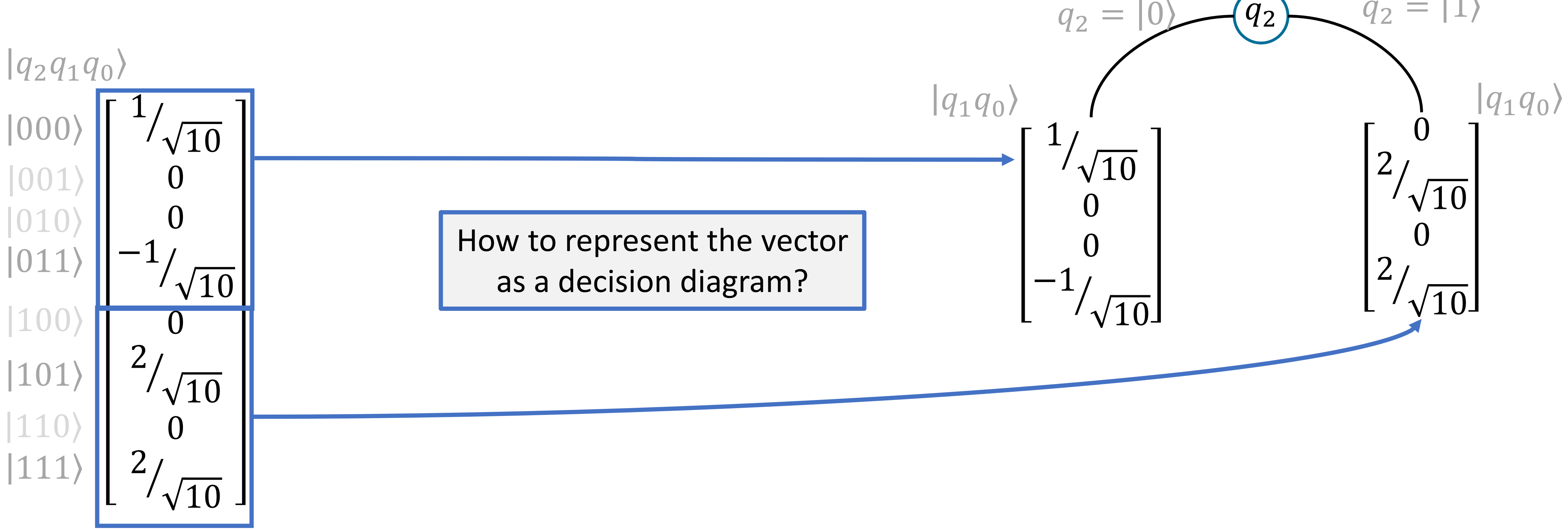
Classical Simulation of Quantum Circuits



Representation of Quantum States

- Quantum states are commonly represented as vectors

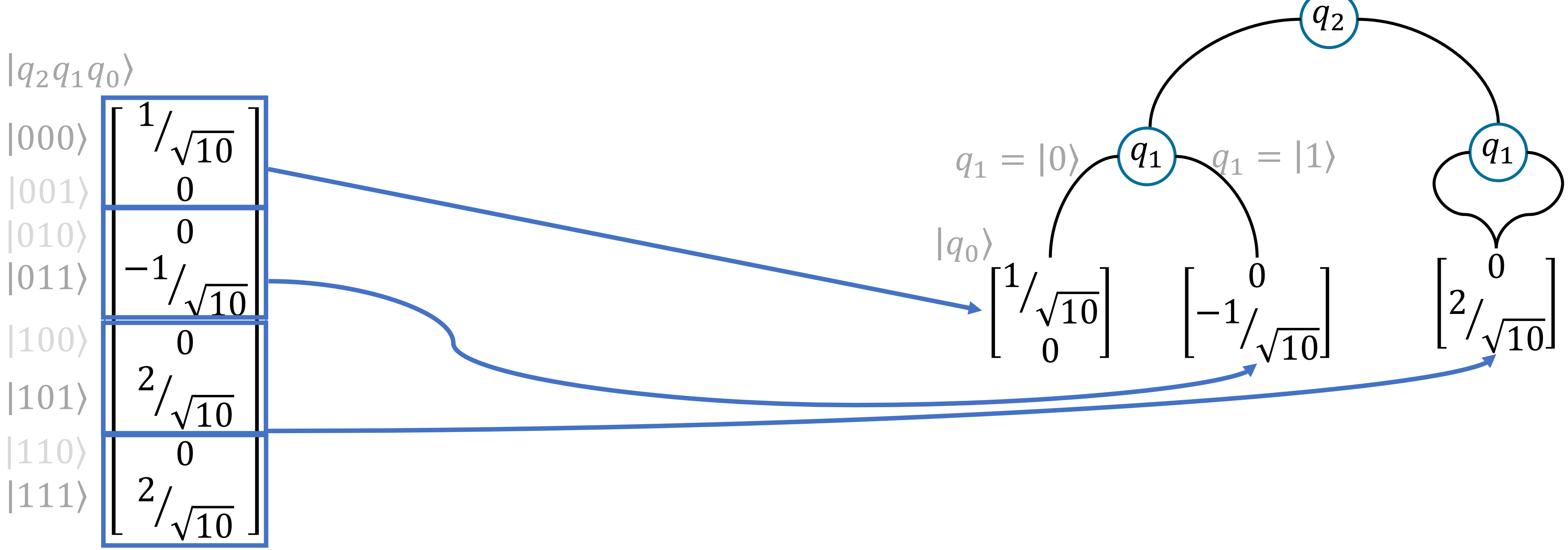
  Consider state $\frac{1}{\sqrt{10}}|000\rangle - \frac{1}{\sqrt{10}}|011\rangle + \frac{2}{\sqrt{10}}|101\rangle + \frac{2}{\sqrt{10}}|111\rangle$



Representation of Quantum States

- Quantum states are commonly represented as vectors

  Consider state $\frac{1}{\sqrt{10}}|000\rangle - \frac{1}{\sqrt{10}}|011\rangle + \frac{2}{\sqrt{10}}|101\rangle + \frac{2}{\sqrt{10}}|111\rangle$



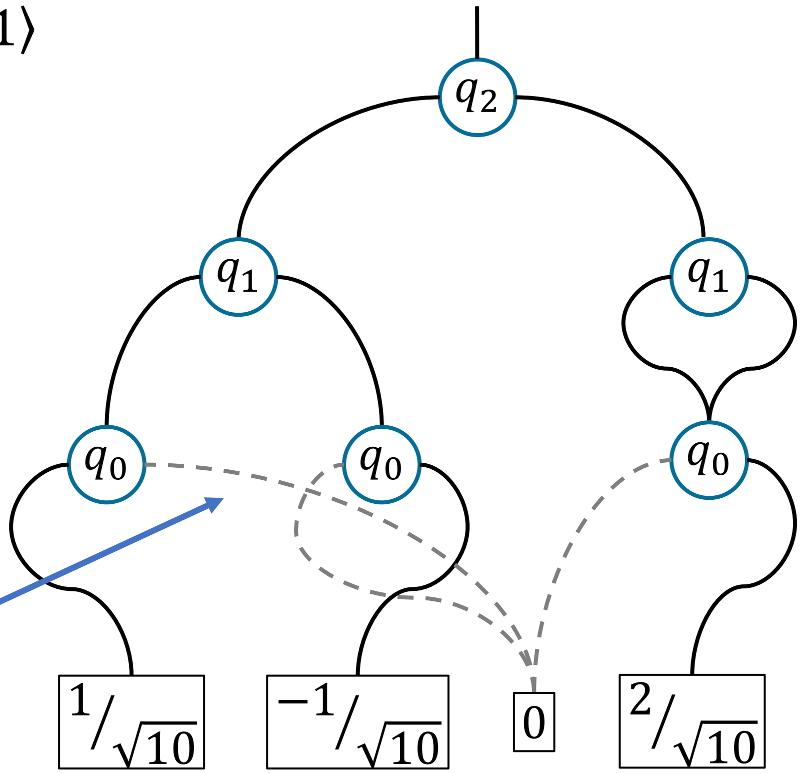
Representation of Quantum States

- Quantum states are commonly represented as vectors

  Consider state $\frac{1}{\sqrt{10}}|000\rangle - \frac{1}{\sqrt{10}}|011\rangle + \frac{2}{\sqrt{10}}|101\rangle + \frac{2}{\sqrt{10}}|111\rangle$

$$
\begin{array}{c} |q_2q_1q_0\rangle \\ |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{array}
\begin{array}{c} \\ \left[\begin{array}{c} 1/\sqrt{10} \\ 0 \\ 0 \\ -1/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \end{array}\right] \end{array}
$$

Edges to zero can be omitted



Representation of Quantum States

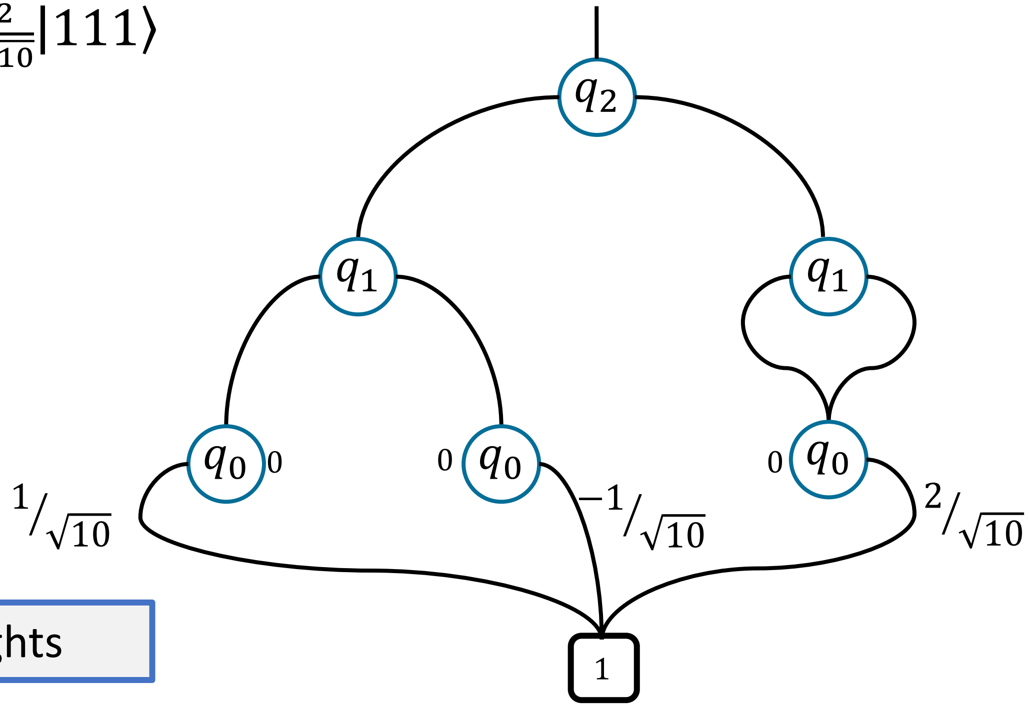
- Quantum states are commonly represented as vectors

  Consider state $\frac{1}{\sqrt{10}}|000\rangle - \frac{1}{\sqrt{10}}|011\rangle + \frac{2}{\sqrt{10}}|101\rangle + \frac{2}{\sqrt{10}}|111\rangle$

$|q_2 q_1 q_0\rangle$

$$\begin{array}{c} |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{array} \begin{bmatrix} 1/\sqrt{10} \\ 0 \\ 0 \\ -1/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \end{bmatrix}$$

Coefficients stored in edge weights



Representation of Quantum States

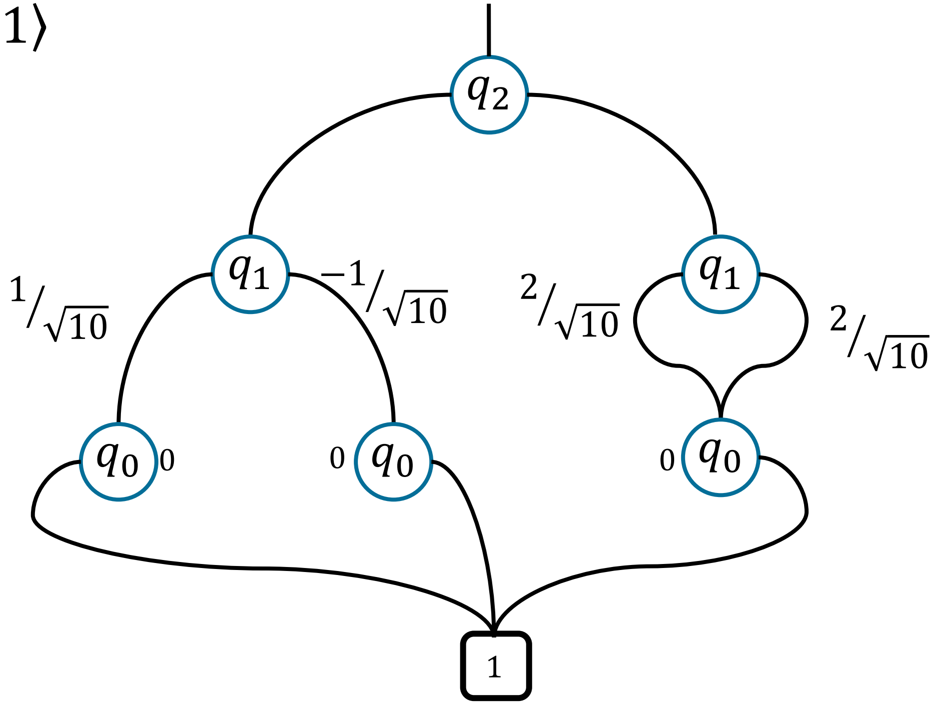
- Quantum states are commonly represented as vectors

  Consider state $\frac{1}{\sqrt{10}}|000\rangle - \frac{1}{\sqrt{10}}|011\rangle + \frac{2}{\sqrt{10}}|101\rangle + \frac{2}{\sqrt{10}}|111\rangle$

$|q_2 q_1 q_0\rangle$

$$\begin{matrix} |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{matrix} \begin{bmatrix} 1/\sqrt{10} \\ 0 \\ 0 \\ -1/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \end{bmatrix}$$

Normalize DD
to guarantee canonicity



Representation of Quantum States

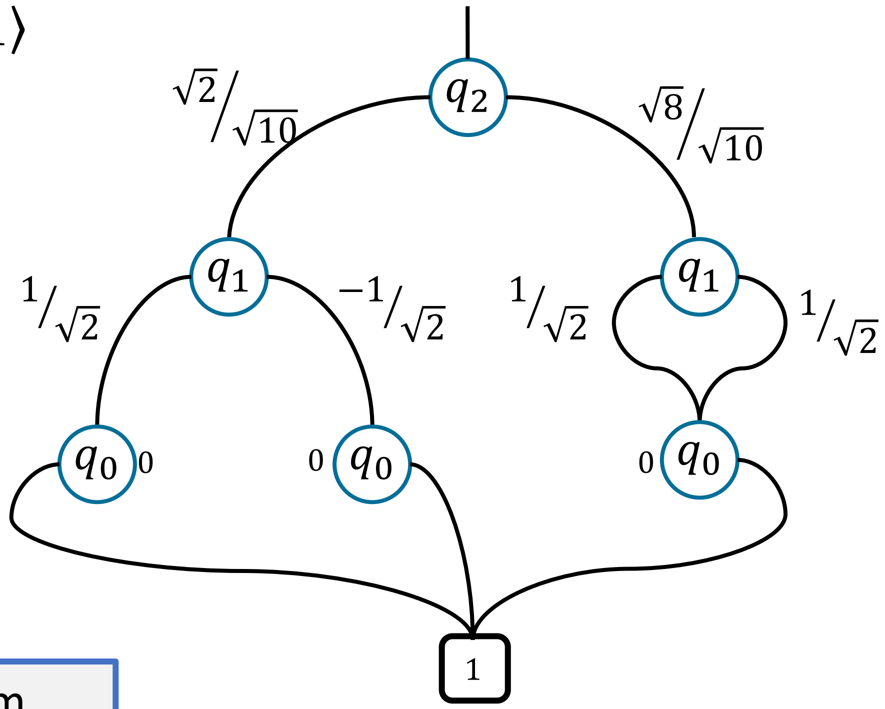
- Quantum states are commonly represented as vectors

  Consider state $\frac{1}{\sqrt{10}}|000\rangle - \frac{1}{\sqrt{10}}|011\rangle + \frac{2}{\sqrt{10}}|101\rangle + \frac{2}{\sqrt{10}}|111\rangle$

$|q_2 q_1 q_0\rangle$

$$\begin{matrix} |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{matrix} \begin{bmatrix} 1/\sqrt{10} \\ 0 \\ 0 \\ -1/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \\ 0 \\ 2/\sqrt{10} \end{bmatrix}$$

Normalize DD
to guarantee canonicity

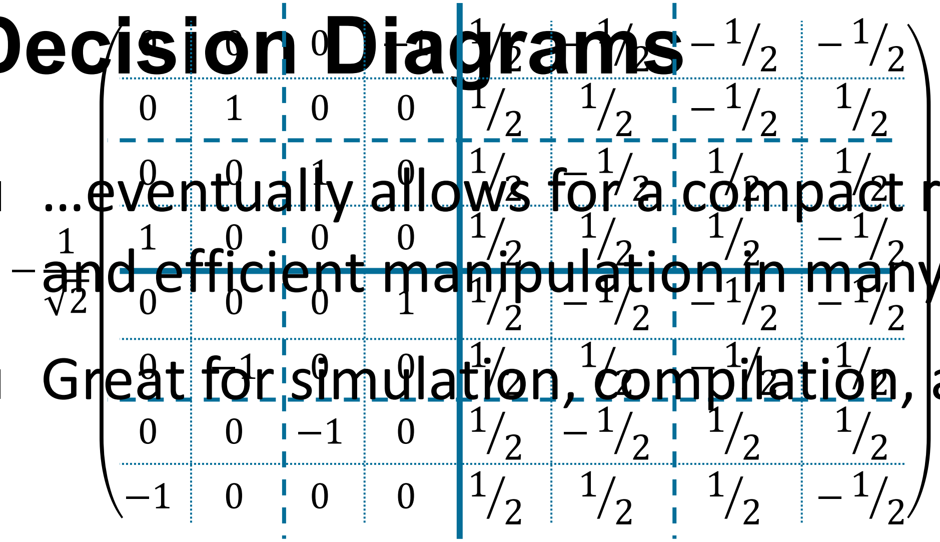


Representation of quantum
operations works similarly

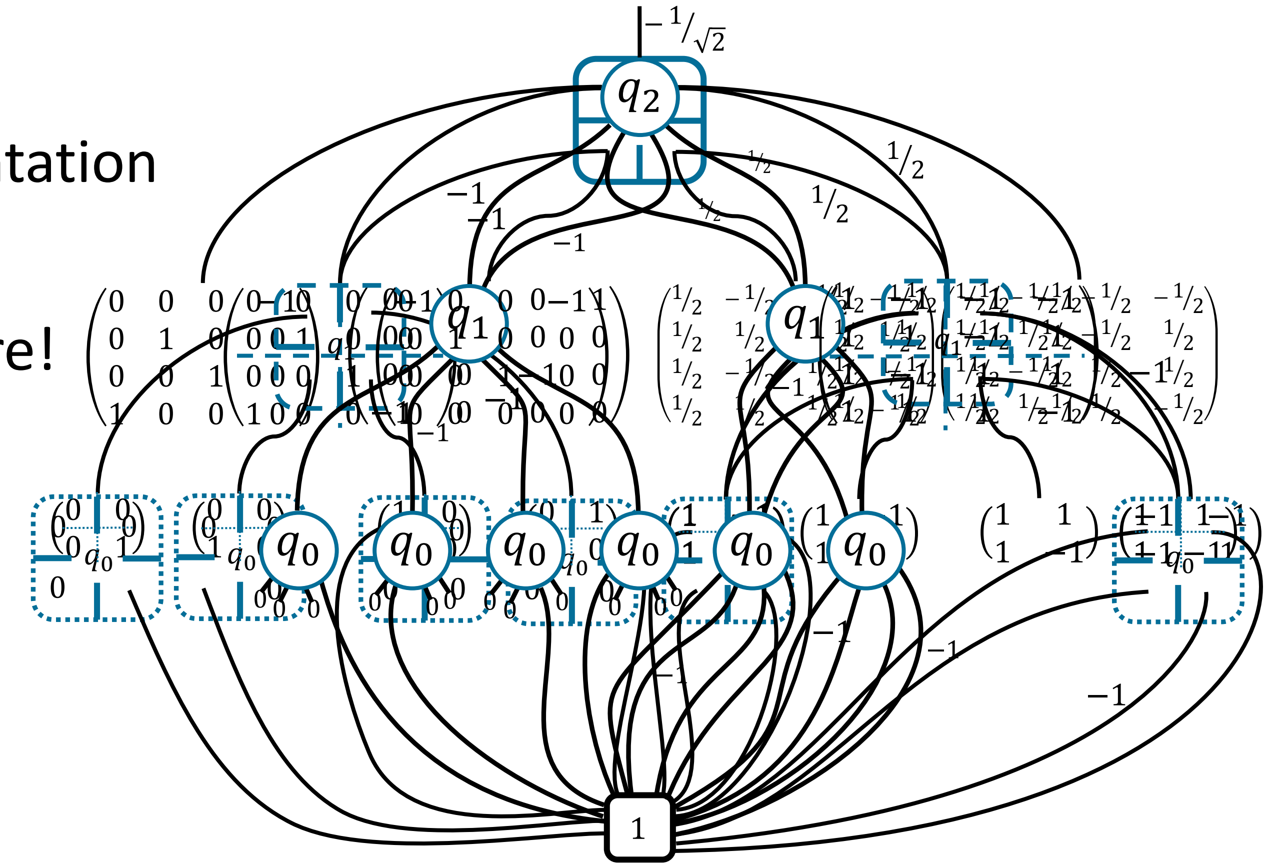
Functionality

Decision Diagram

Decision Diagrams

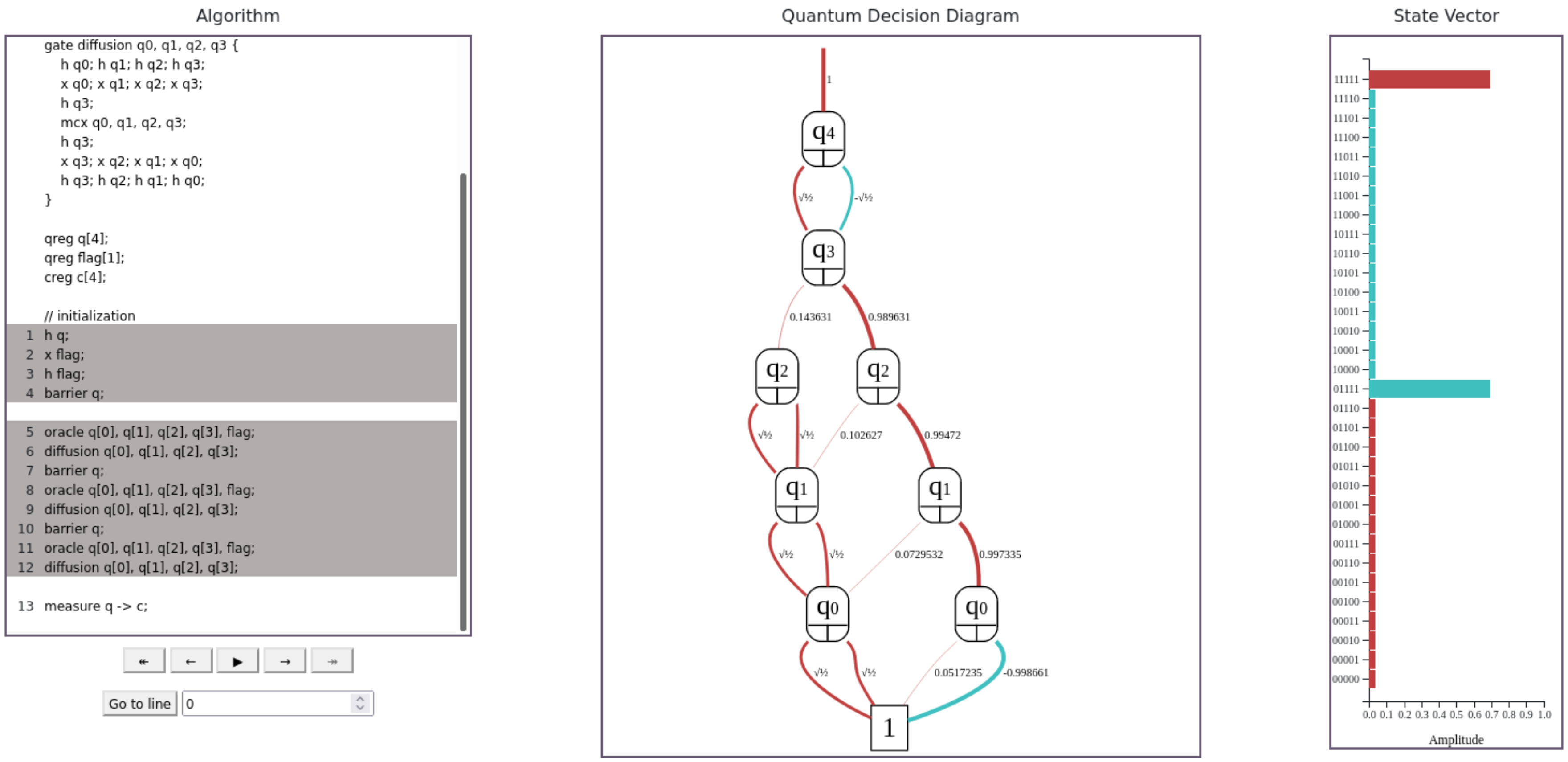


- …eventually allows for a compact representation and efficient manipulation in many cases.
- Great for simulation, compilation, and more!
- …are a canonical representation!
- Great for verification!



Decision Diagrams – MQT DDVis Web-based GUI

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

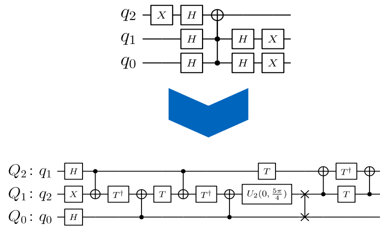


Overview of MQT

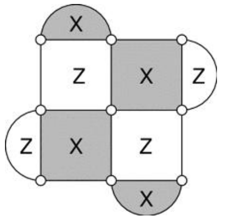
MUNICH QUANTUM TOOLKIT

Application
- Classical problem encoding
- Quantum solution execution
- Solution Decoding

Compilation
- Best compilation options
- Reversible synthesis
- Technology-specific Mapping
- Qudit Compilation

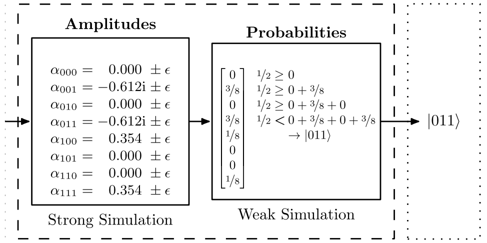


Error Correction
- Decoding algorithms
- Automated code construction and numerical simulations

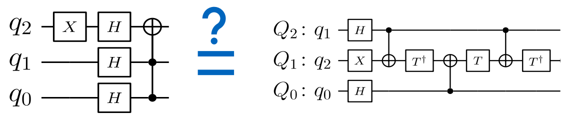


Application → Simulation → Compilation → Verification → Error Correction → Hardware

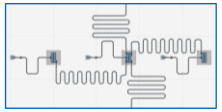
Simulation
- Decision Diagram based simulation of gate-based quantum circuits
- Strong & weak simulation
- Noise-aware simulation



Verification
- Equivalence checking of quantum circuits



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

MQT DDSIM

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

(Fully compatible with Qiskit 1.0)

ghz_3.py

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

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

provider = DDSIMProvider()
backend = provider.get_backend('qasm_simulator')
job = backend.run(circ, shots=100000)
result = job.result()
counts = result.get_counts(circ)
print(counts)
```

Terminal

```
$ python3 ghz_3.py
     ┌───┐           ░ ┌─┐      
q_0: ┤ H ├──■────■───░─┤M├──────
     └───┘┌─┴─┐  │   ░ └╥┘┌─┐   
q_1: ─────┤ X ├──┼───░──╫─┤M├───
          └───┘┌─┴─┐ ░  ║ └╥┘┌─┐
q_2: ──────────┤ X ├─░──╫──╫─┤M├
               └───┘ ░  ║  ║ └╥┘
meas: 3/════════════════╩══╩══╩═
                        0  1  2 
{'000': 50149, '111': 49851}
```

MUNICH QUANTUM TOOLKIT

PlanQK

600+ ⭐ on GitHub, **1M+ Downloads** on PyPI

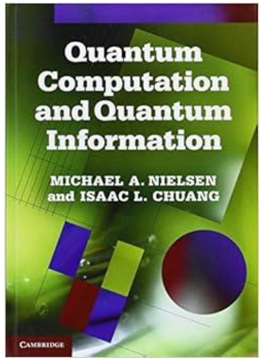
Star

Overview: Software Tooling for Quantum

- Programming Language(s): Python, (C++, Rust, etc.)
- Different categories of SDK providers:
  - Quantum SDK and Quantum Computers
    (e.g. IBM/Qiskit, Google/Cirq, Quantinuum/Pytket, Xanadu/Pennylane, …)
  - Quantum SDK and platform
    (Microsoft Azure, AWS Braket, …)
  - Only Quantum SDK
    (e.g. MQT, Classiq, …)
- Functionalities of SDKs similar with differences in the details, such as syntax, supported simulators or quantum computers
- Simulator vs. actual Quantum Computer

How to get started?

- Quantum Computation and Quantum Information – Nielsen & Chuang



- Quantum Mechanics | The Theoretical Minimum
  (by Leonard Susskind, Stanford YT Channel)

- IBM Quantum Learning platform and Qiskit Youtube channel
  (Qiskit Summer Camp, Understanding Quantum Information and Computation Series, Coding with Qiskit, etc.)

- Codebook and Challenges from Pennylane

Thank you for your time!

References and Bibliography

- https://bloch.kherb.io/
- https://review.mastersunion.org/wp-content/uploads/2022/06/Quantum-and-classical-computer-1.gif
- https://devopedia.org/images/article/68/4882.1613047200.jpg
- https://quantum.ibm.com/composer/files/new
- https://www.cda.cit.tum.de/research/quantum/
- https://www.cda.cit.tum.de/app/ddvis/