1. Gate-based QC

Dr. Stefano Gogioso

UNIVERSITY OF OXFORD

HASHBERG

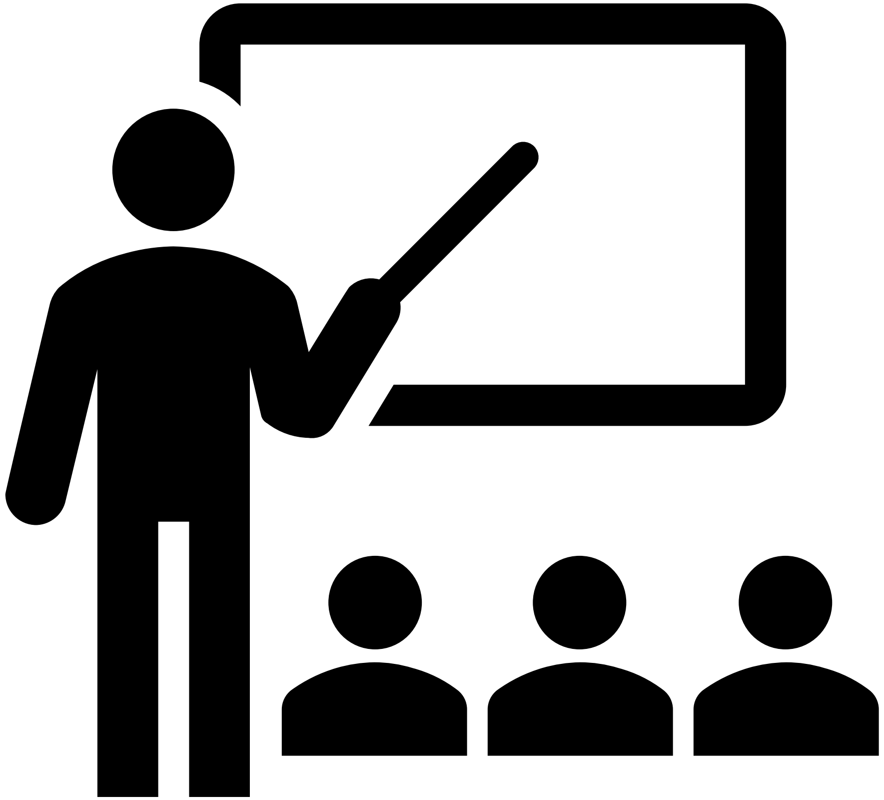
Your Lecturer

Dr. Stefano Gogioso

Lecturer in Quantum Computing @ UNIVERSITY OF OXFORD

Quantum Software and Decentralised Finance @ HASHBERG

Tell us a little about you!



The Goals of Quantum Computing (QC)

The purpose of gate-based quantum computing is simple: quantum circuits are used to create quantum states which, upon simple measurements, yield interesting quantities.

These quantities typically take two forms:

1. Probability distributions on strings of bits.
2. Average values for certain energy functions.

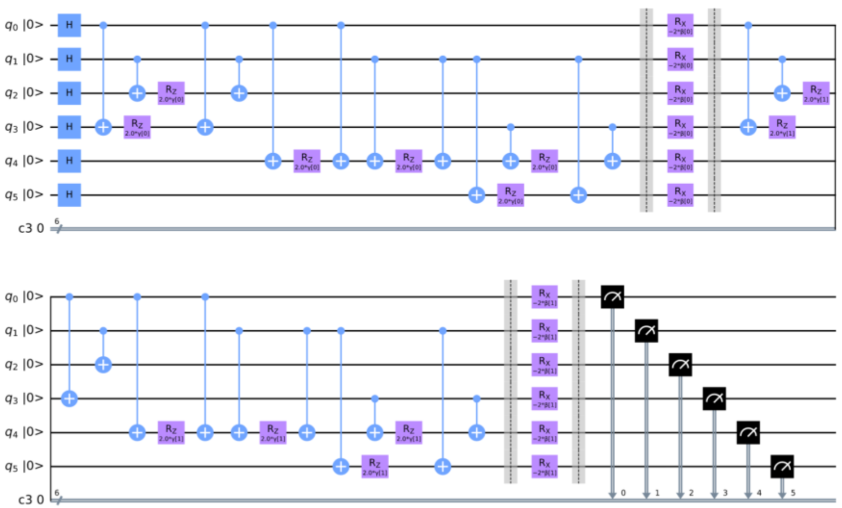
The Goals of Quantum Computing (QC)

Exactly **which** probability distributions or energy function are interesting, and **how** to create the necessary quantum states, is the core task faced by researchers in quantum computing.

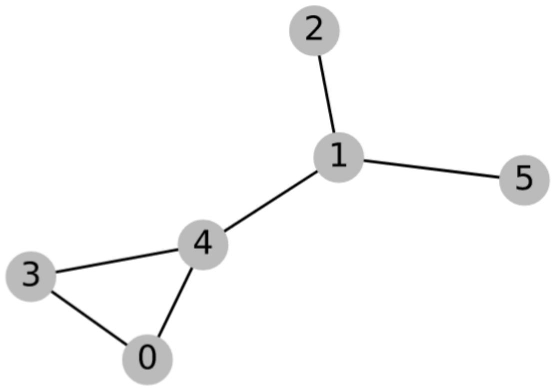
For practising quantum computer scientists, the task is simpler: implement the quantum circuits, perform the measurements and solve real-world problems.

QC Workflow

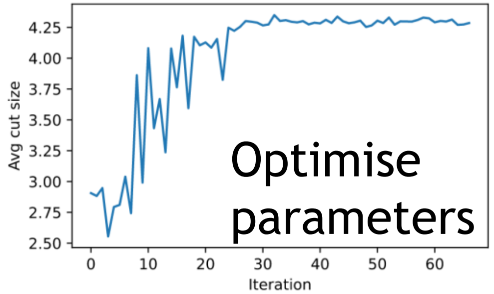
Quantum Circuit



Problem data

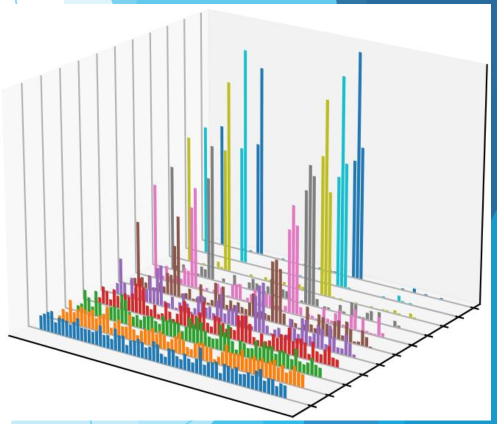


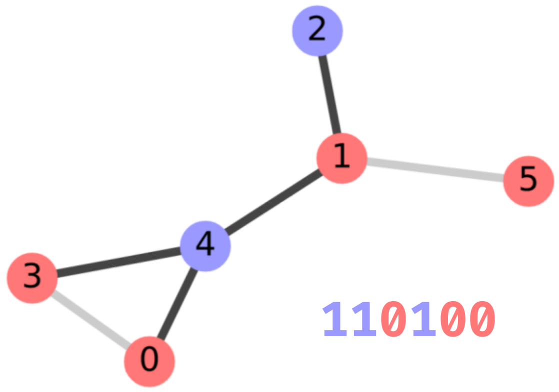
Encode



Optimise parameters

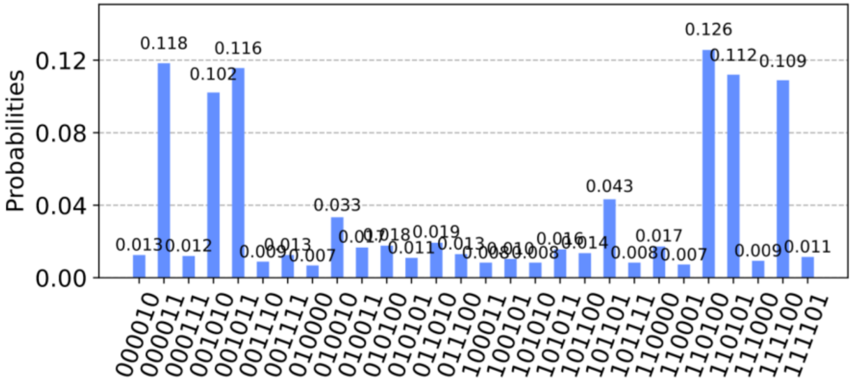
Send to QPU





Candidate solutions

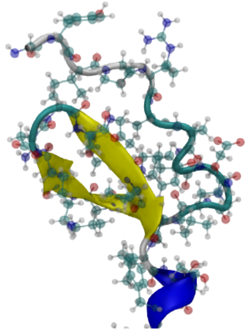
Decode



Sampled bitstrings

QC Workflow

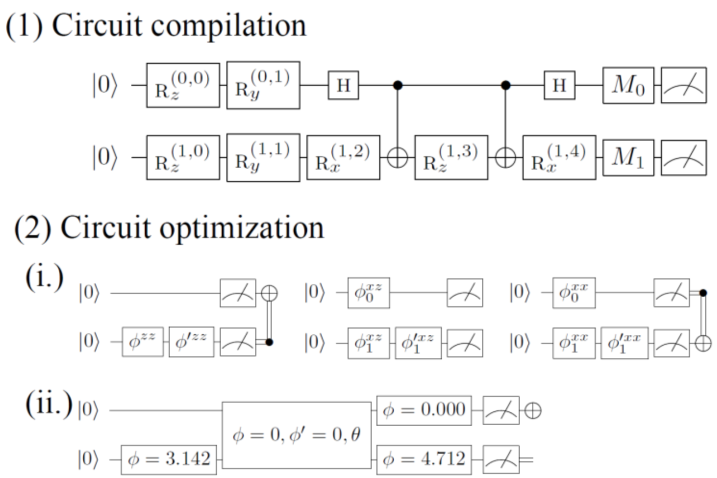
Molecular data

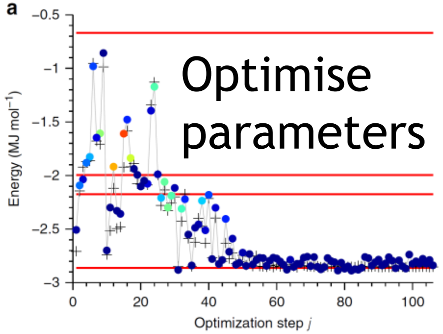


=

Encode

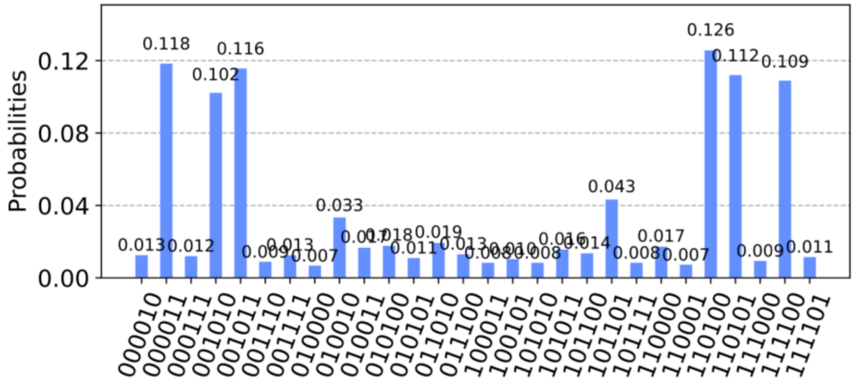
Quantum Circuit



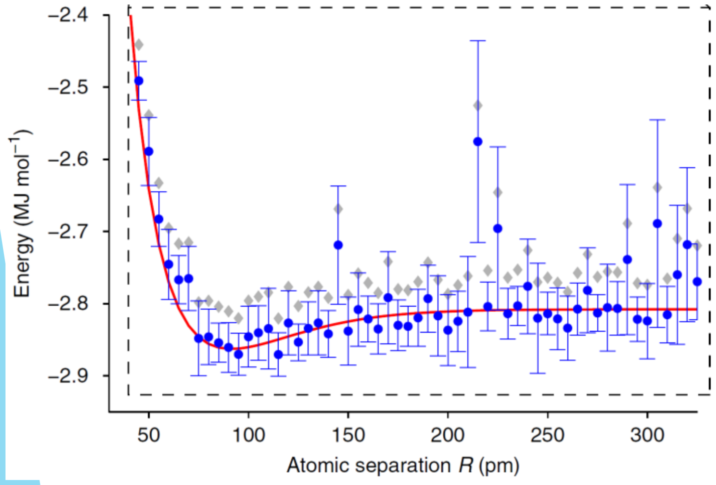


Optimise parameters

Send to QPU



Sampled bitstrings



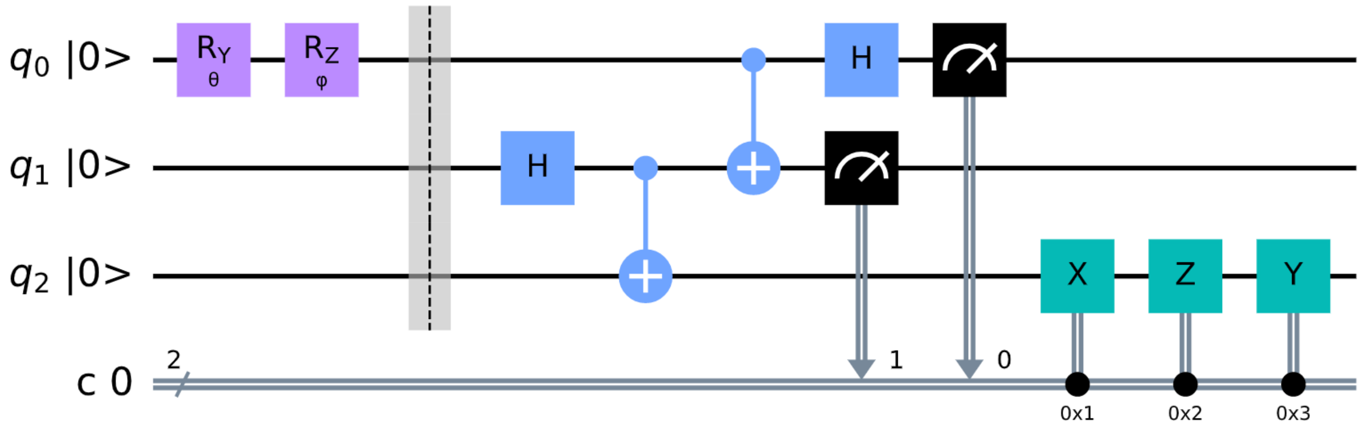
Decode

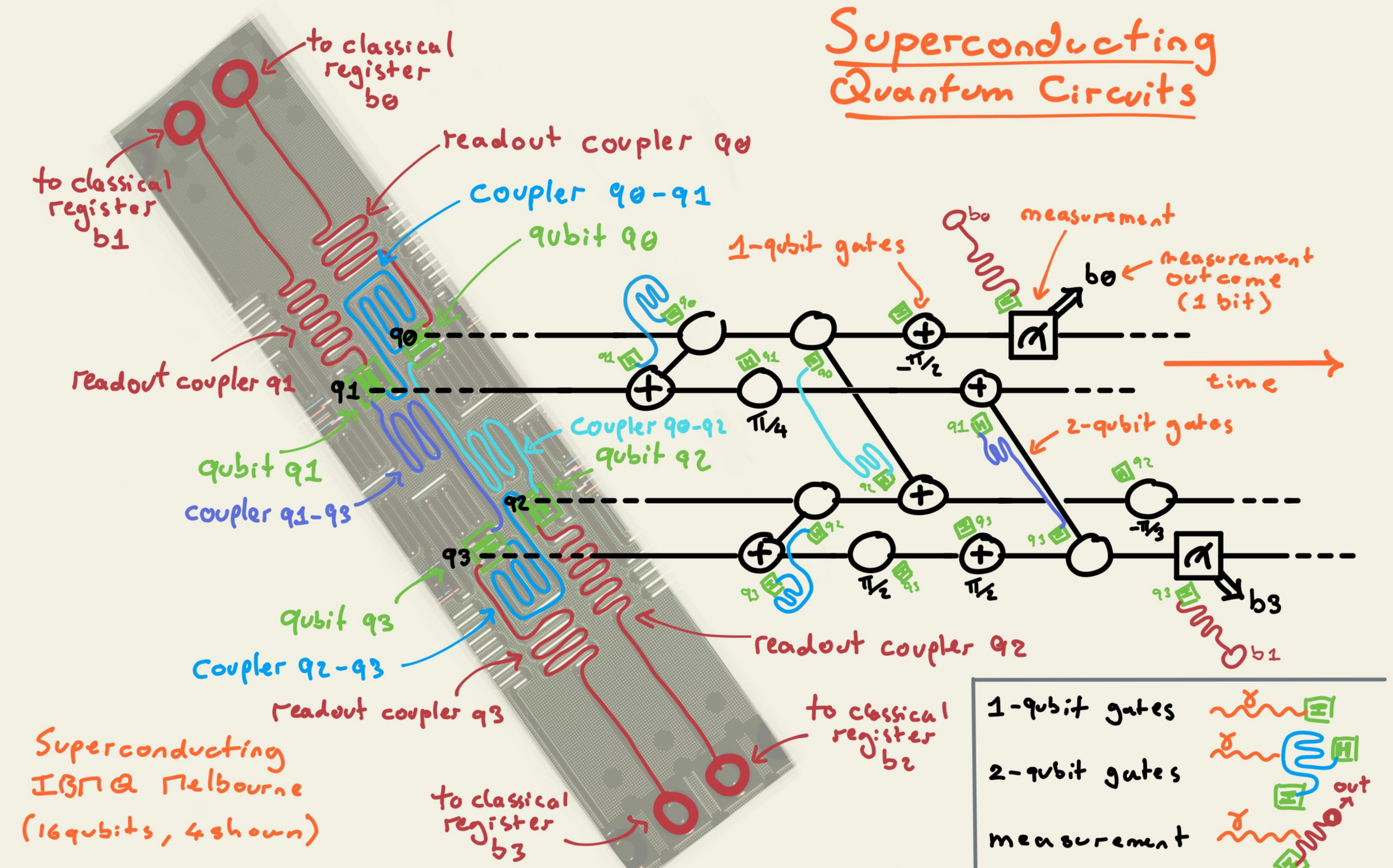
Candidate energy curves

arXiv:2102.07045
doi:10.1038/nature5213

Quantum Circuits

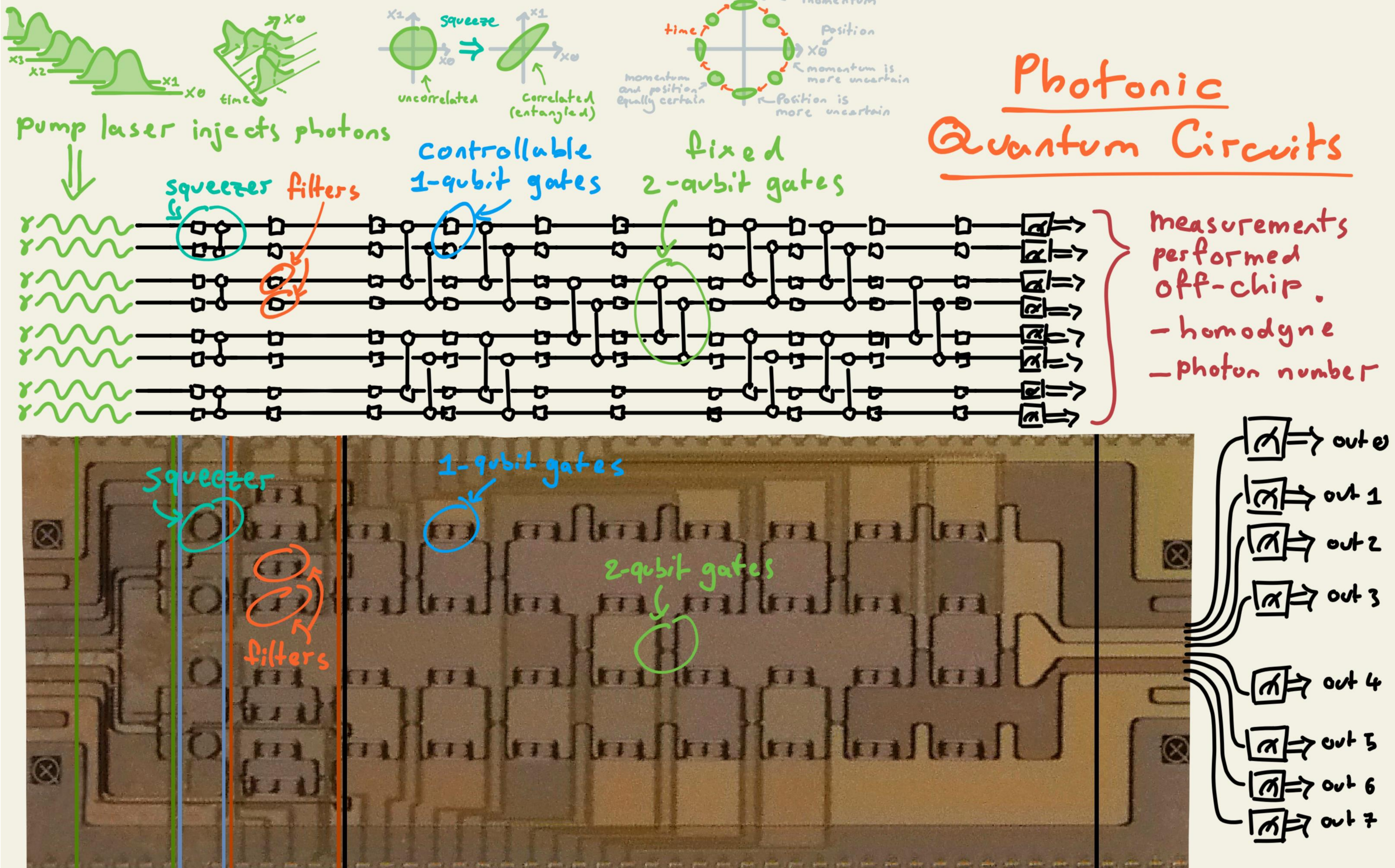
```python
from qiskit import QuantumCircuit
from qiskit.circuit import Parameter
# create a new circuit (3 qubits, 2 bits)
circ = QuantumCircuit(3, 2)
# custom input state on q0:
circ.ry(Parameter("θ"), 0)
circ.rz(Parameter("φ"), 0)
circ.barrier()
# quantum teleportation circuit:
circ.h(1)
circ.cx(1, 2)
circ.cx(0, 1)
circ.h(0)
circ.measure([0, 1], [0, 1])
circ.x(2).c_if(circ.cregs[0], 0b01)
circ.z(2).c_if(circ.cregs[0], 0b10)
circ.y(2).c_if(circ.cregs[0], 0b11)
# draw the circuit:
circ.draw("mpl", initial_state=True)
```



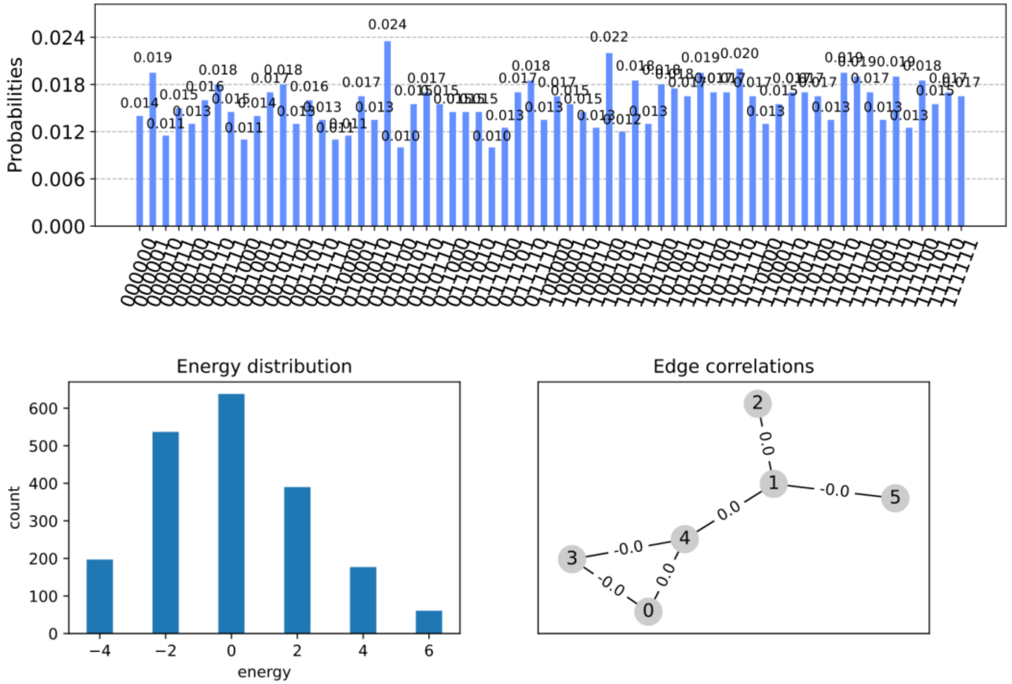


Superconducting Quantum Circuits
to classical register b0
to classical register b1
readout coupler q0
coupler q0-q1
qubit q0
readout coupler q1
q0
q1
q2
q3
qubit q1
coupler q1-q3
coupler q0-q2
qubit q2
qubit q3
coupler q2-q3
readout coupler q3
readout coupler q2
to classical register b2
to classical register b3
Superconducting IBMQ Melbourne
(16 qubits, 4 shown)
1-qubit gates
measurement
measurement outcome (1 bit)
b0
b1
b3
2-qubit gates
time
π/4
-π/2
π/2
π/2
-π/3
1-qubit gates
2-qubit gates
measurement
out

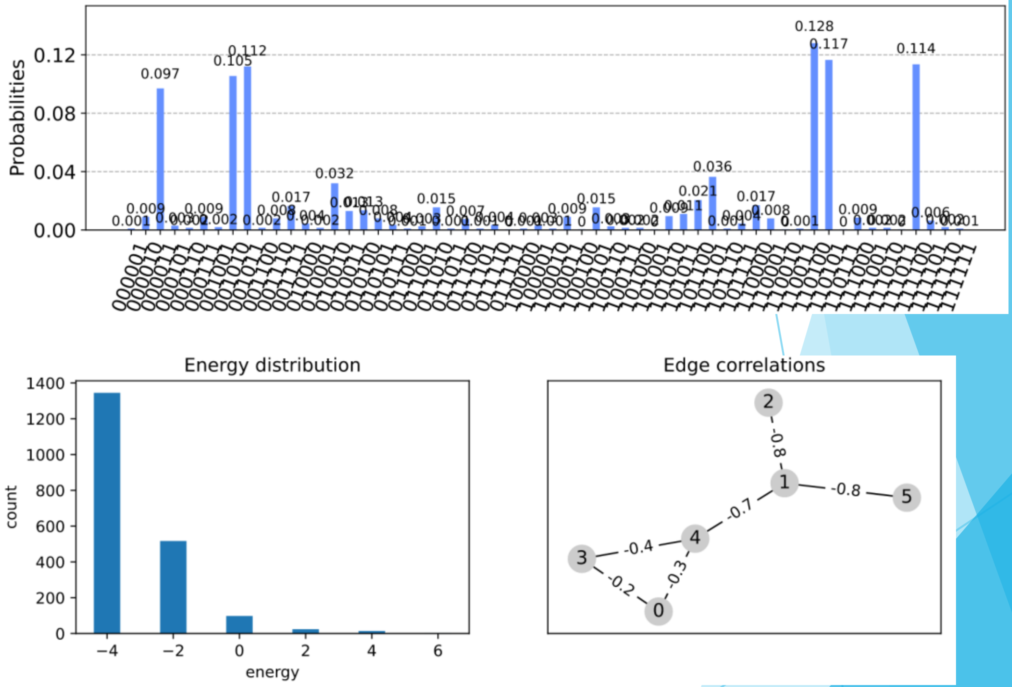
Photonic Quantum Circuits



Quantum circuits are used to create complex correlation patterns

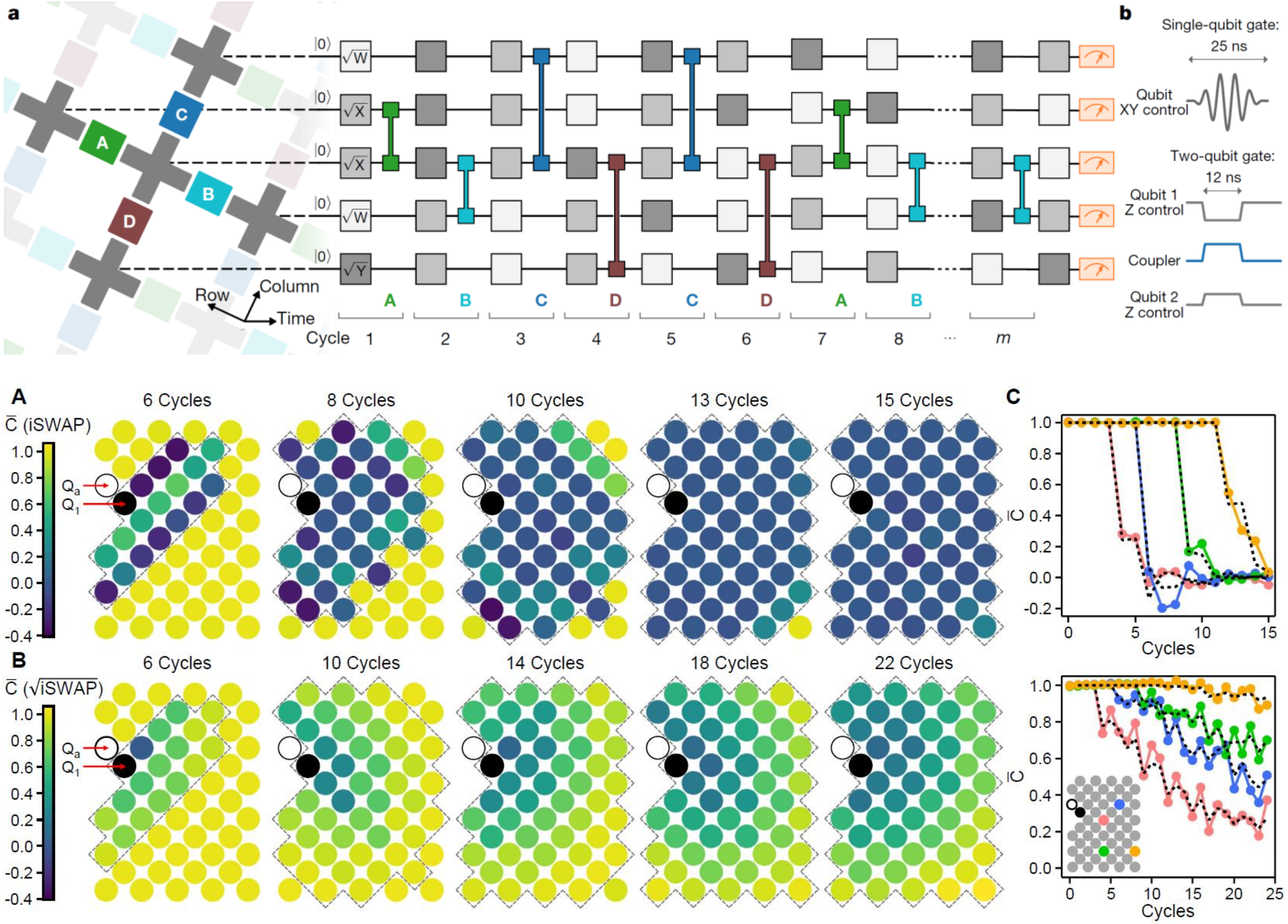


Optimise



Strong multi-partite correlations increase the probability of good (low energy) problem solutions.

Quantum circuits are used to create complex correlation patterns



As more entangling gates are applied (more cycles), strong correlations spread across the qubits. Here, we see the 53 qubits of Google Sycamore, in a 2021 paper.

https://arxiv.org/abs/2101.08870

The $iSWAP$ gate (top) is more entangling than the $\sqrt{iSWAP}$ gate (bottom), so it spreads correlations faster across the 53-qubit lattice of Sycamore.

Some Quantum Hardware Manufacturers

IBM

PsiQuantum

Google

QUANTINUUM

rigetti

QUANDELA

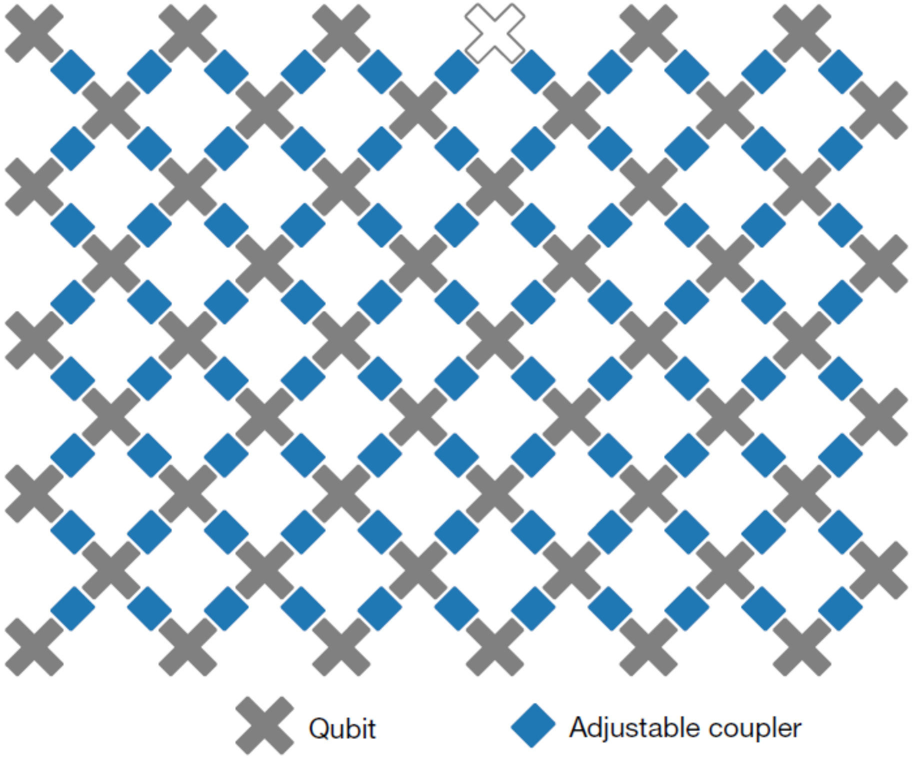
IONQ

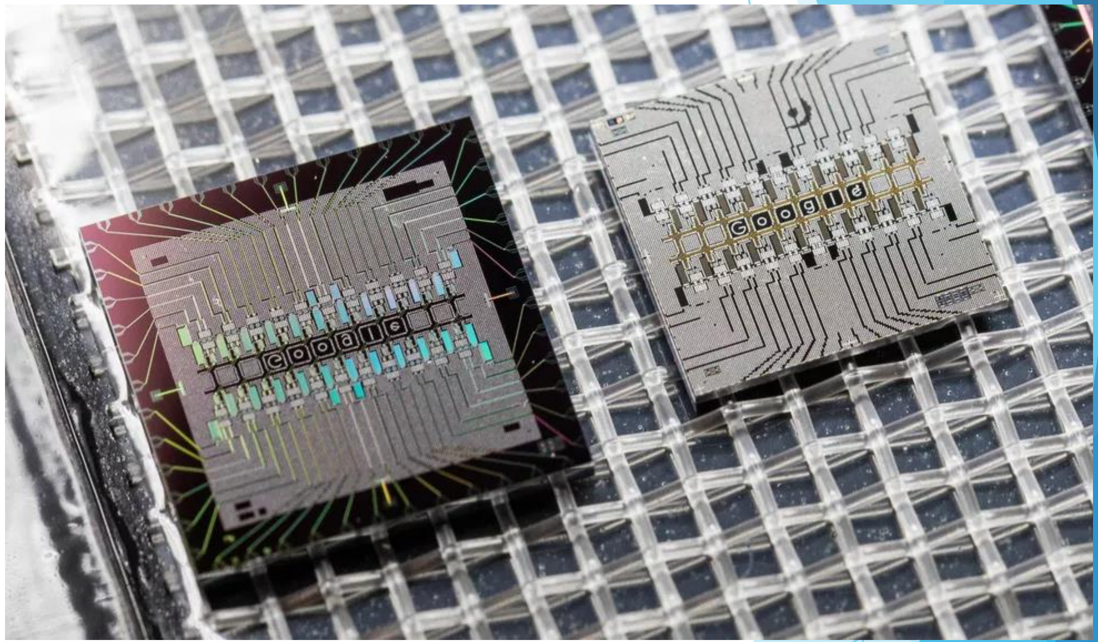
PASQAL

D:WAVE

XANADU

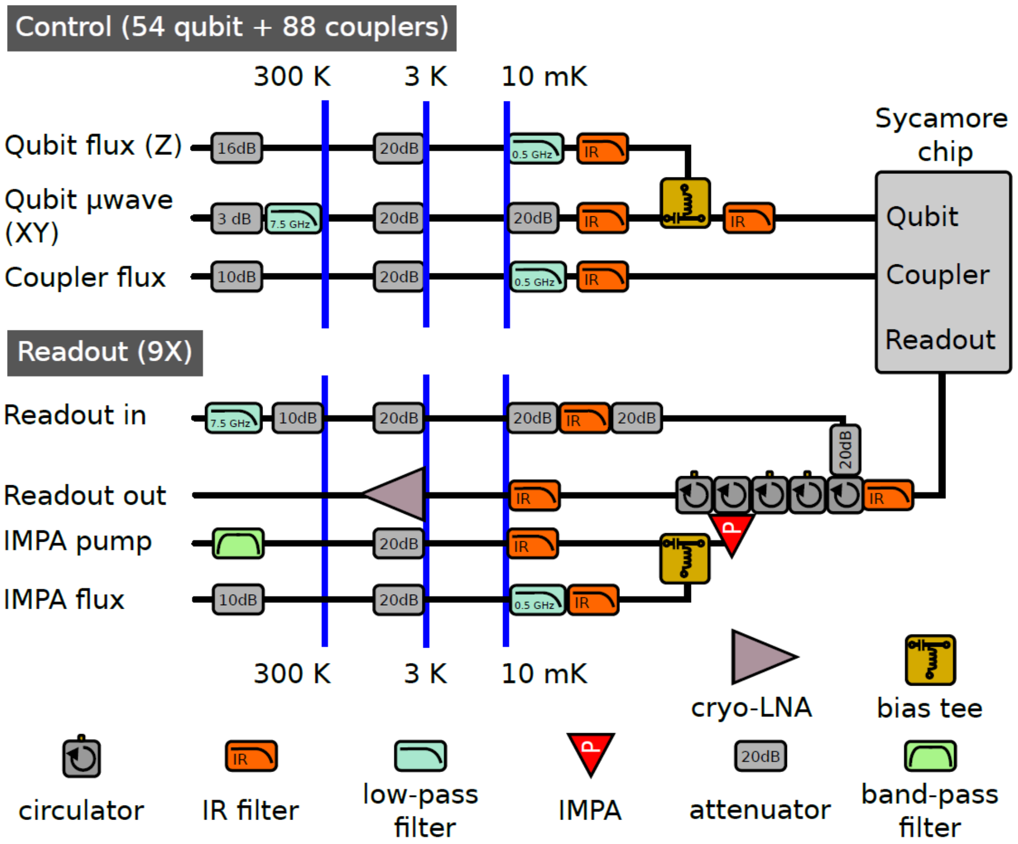
Superconducting QC

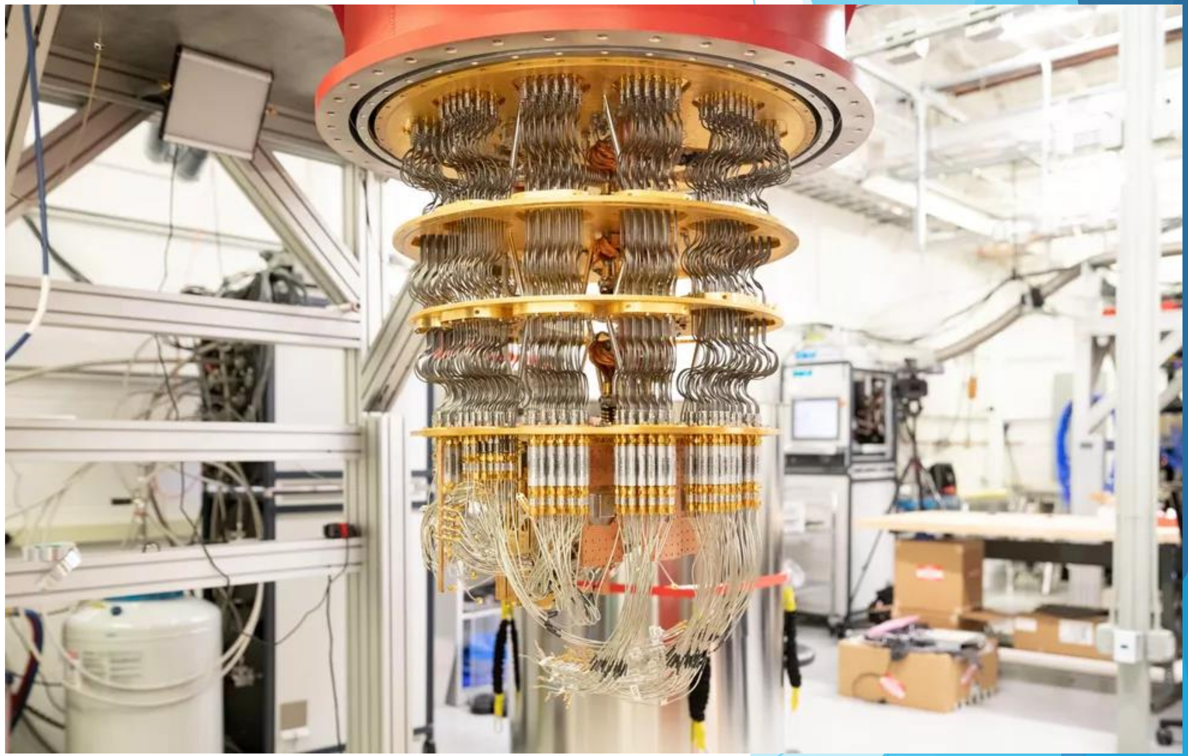



Credit: Stephen Shankland/CNET

Google

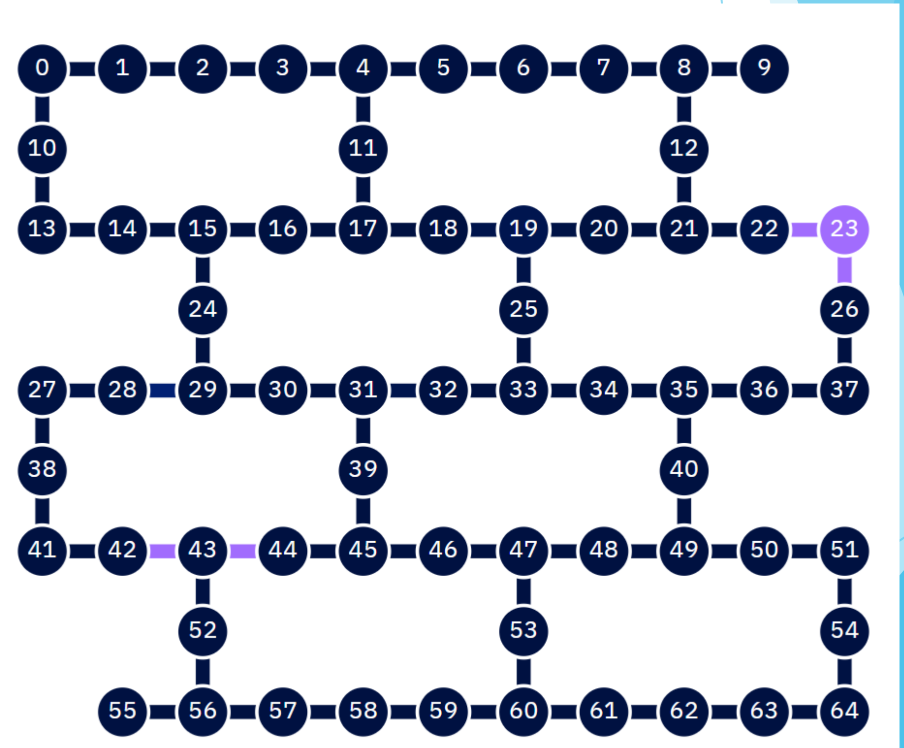
Superconducting QC



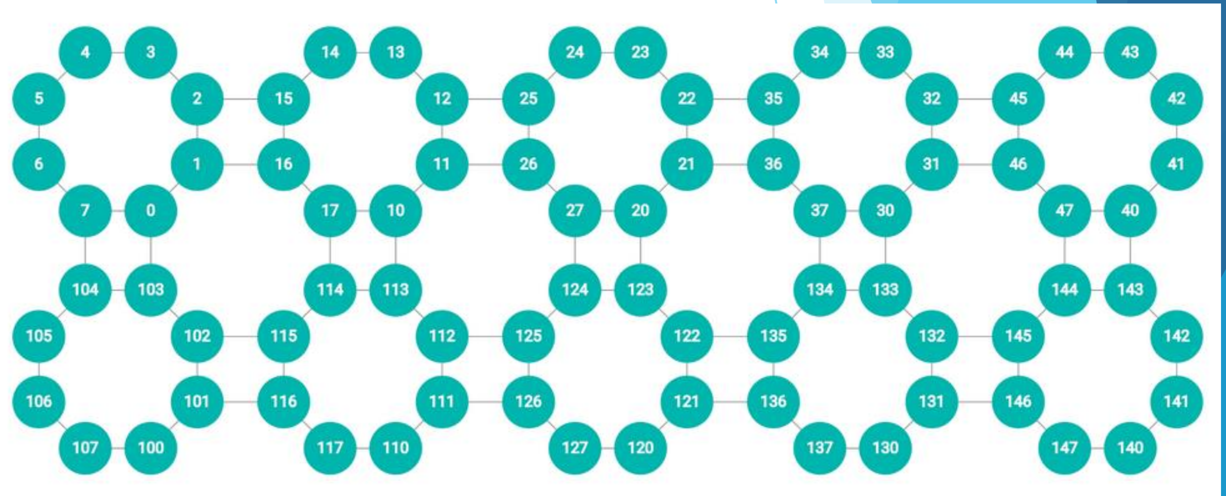


Credit: Stephen Shankland/CNET

Superconducting QC

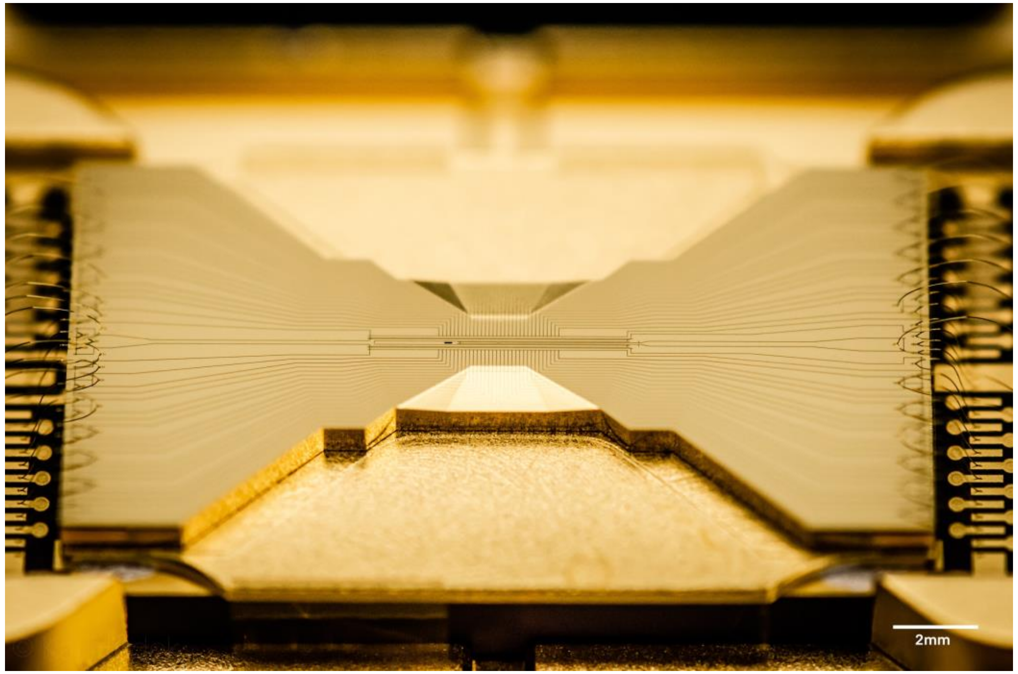


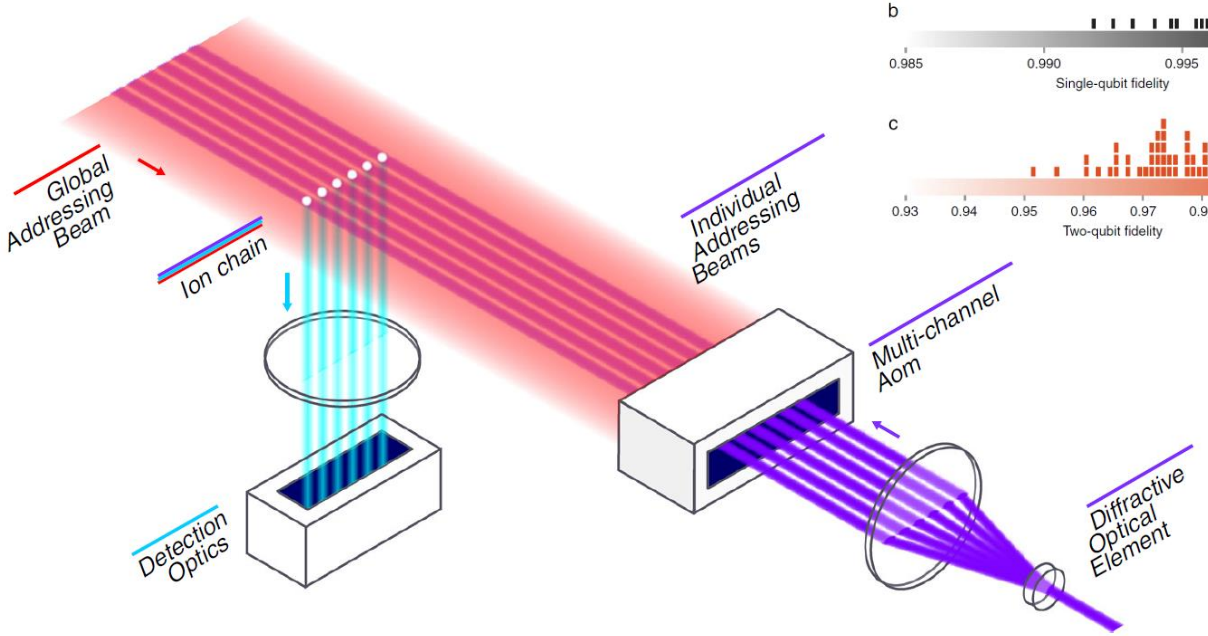
Superconducting QC

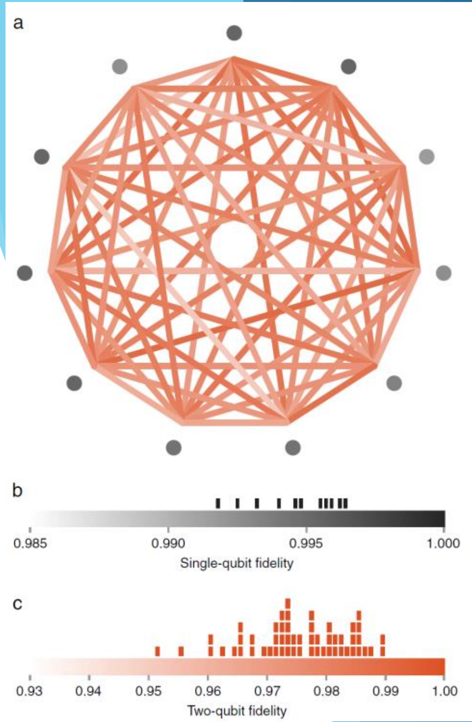


rigetti

Ion Trap QC

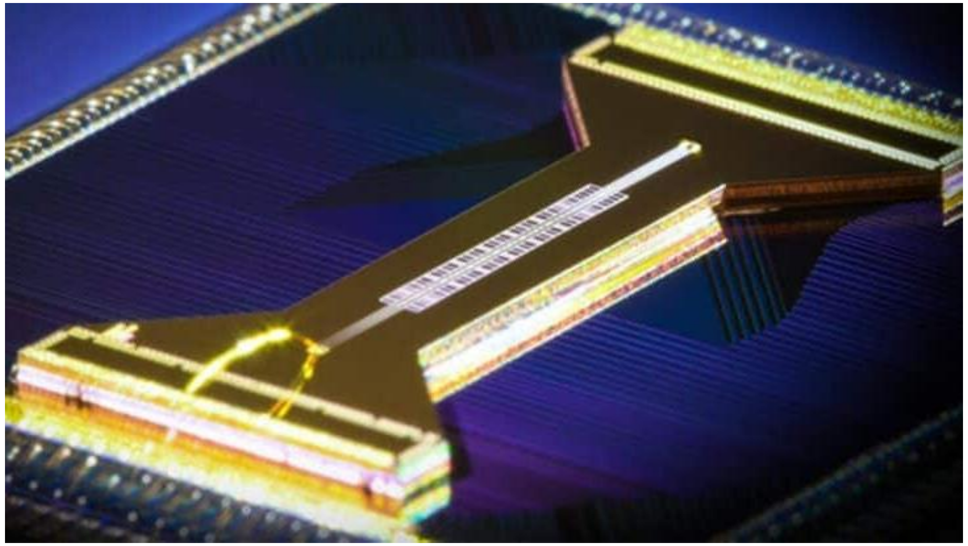






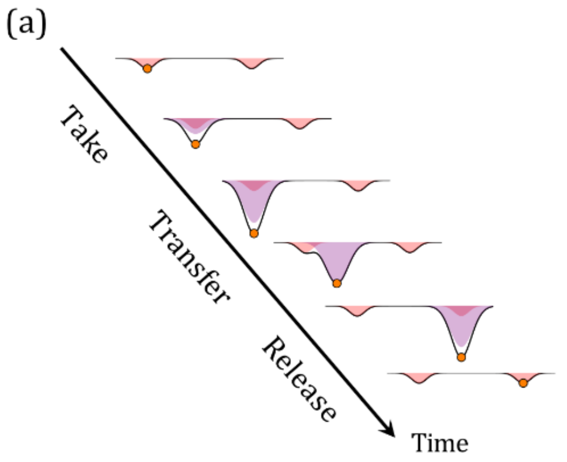
IONQ

Ion Trap QC

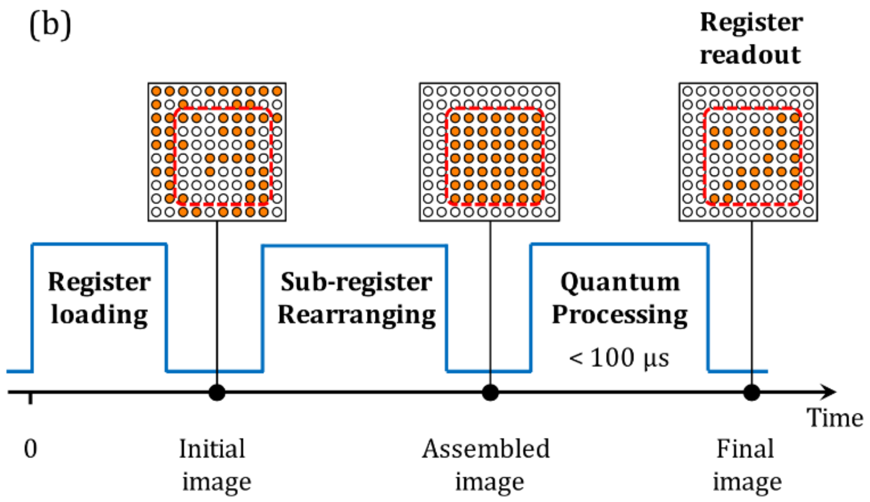


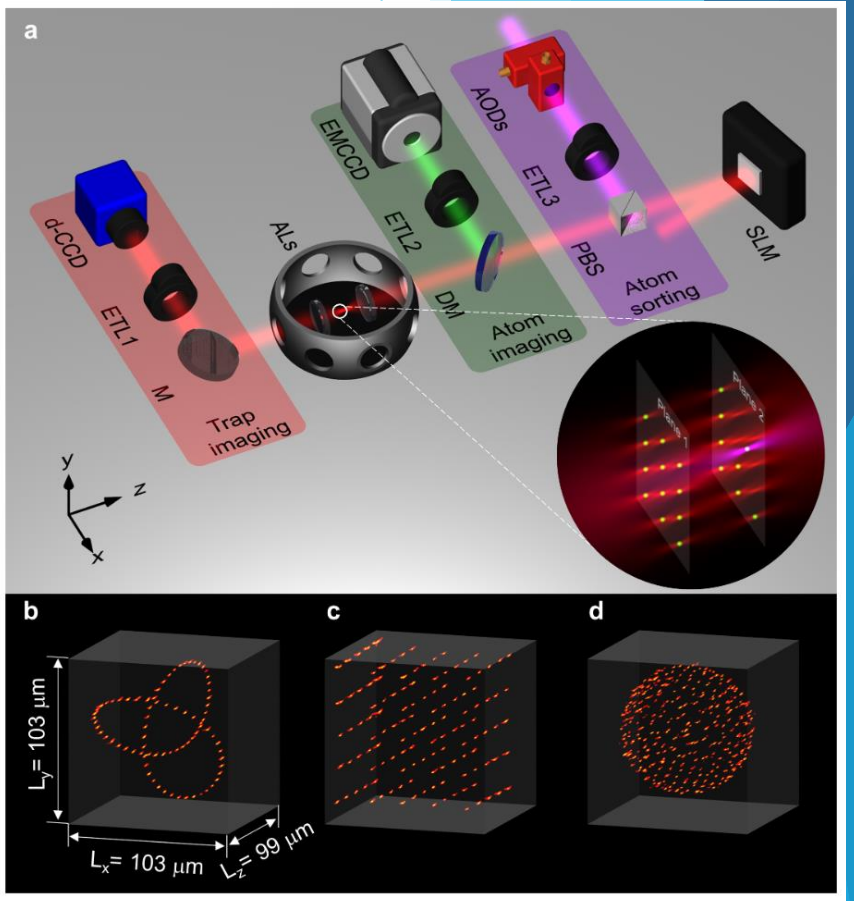
QUANTINUUM

Neutral Atoms QC



arXiv:2006.12326

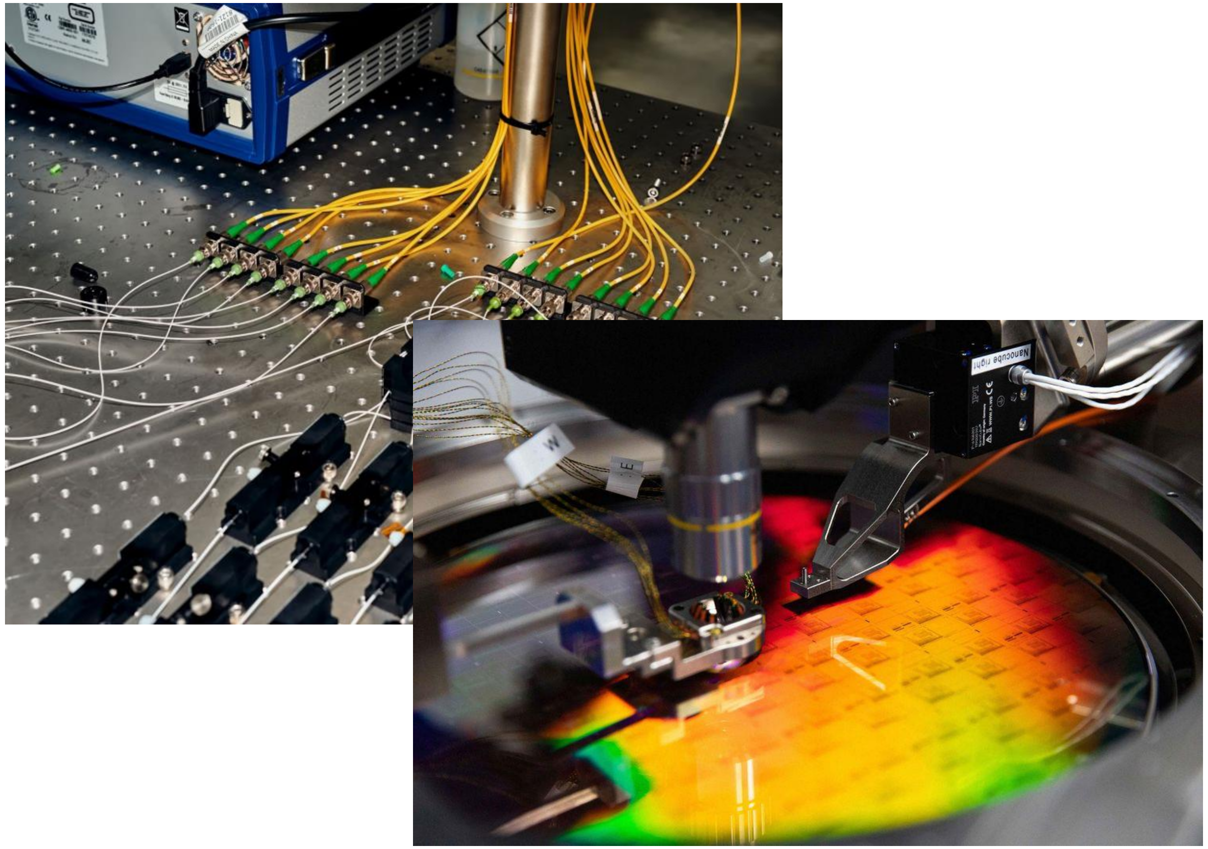


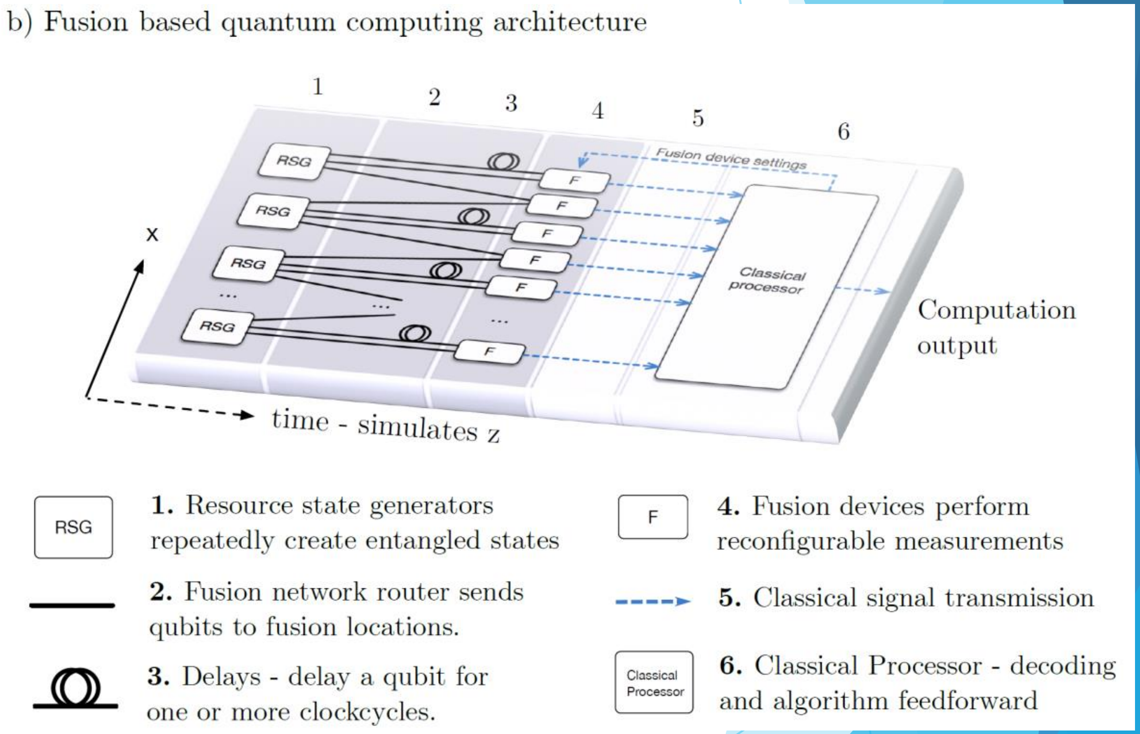


arXiv:1712.02727

PASQAL

Photonic QC

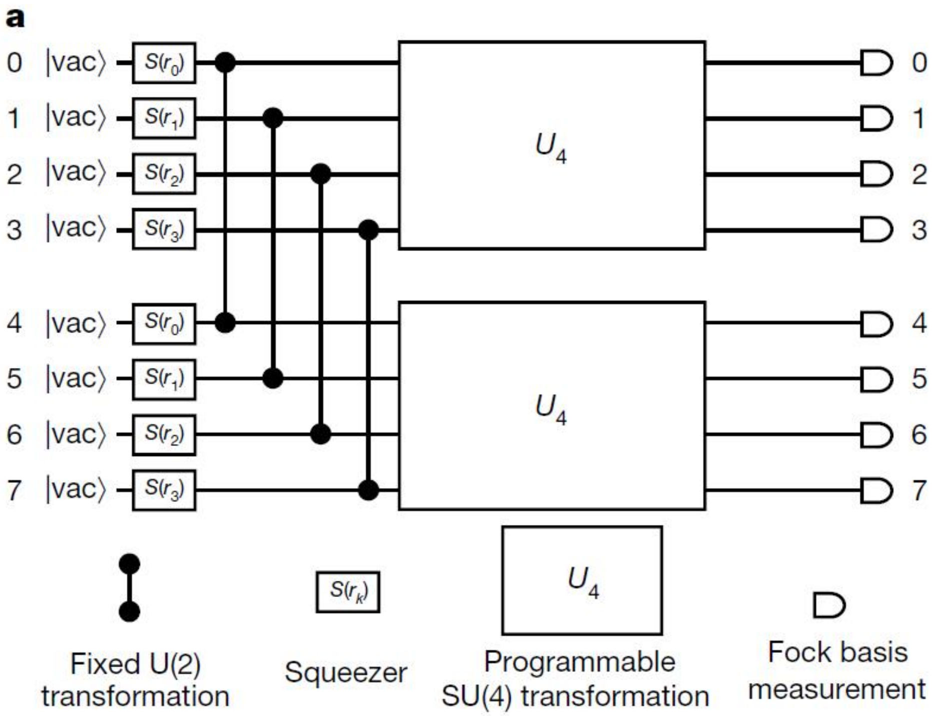


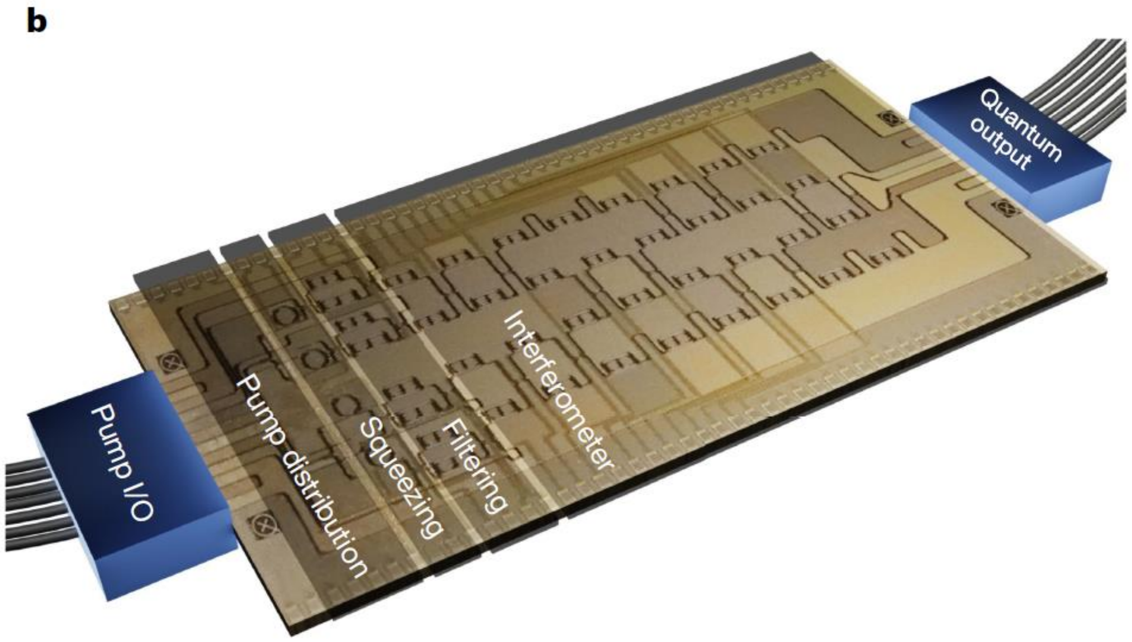


arXiv:2101.09310

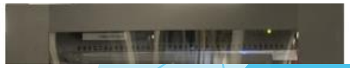
Ψ PsiQuantum

Photonic QC



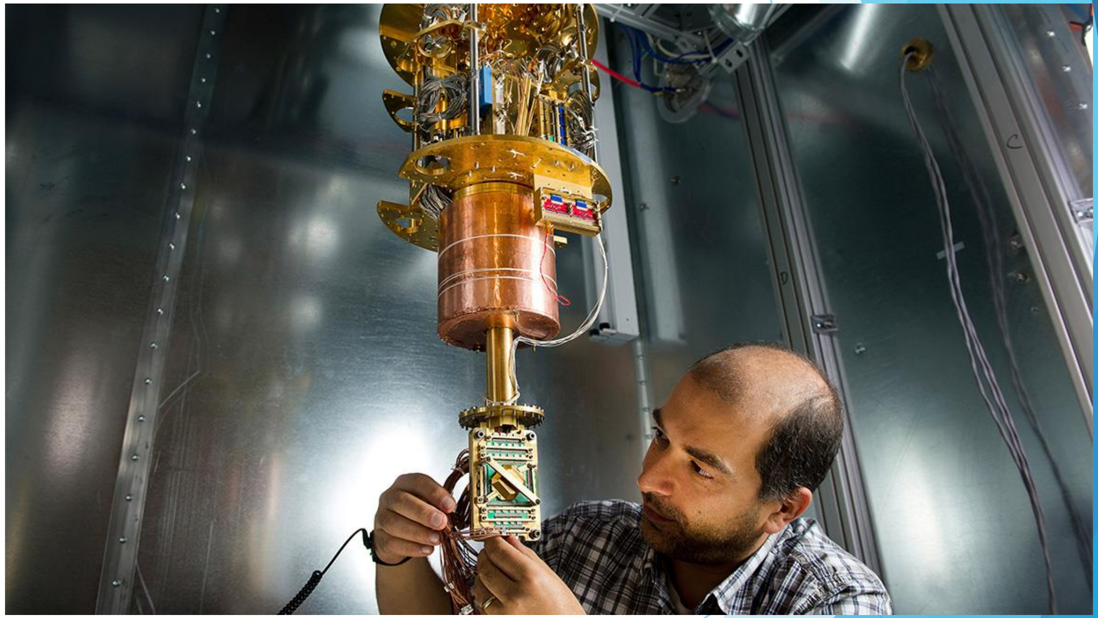


d



XANADU

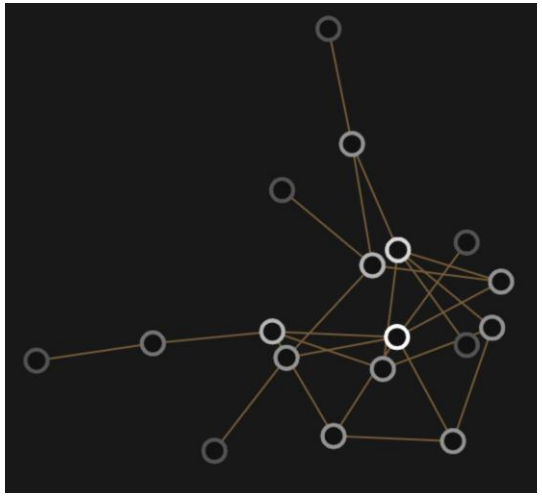
Quantum Annealing (QA)



D:wave

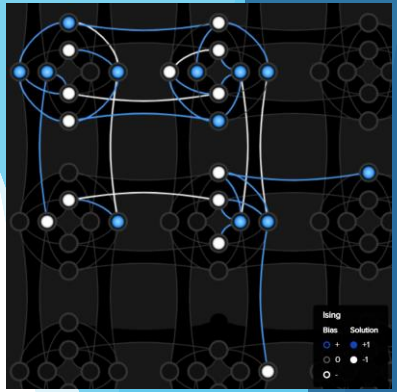
QA Workflow

Encode

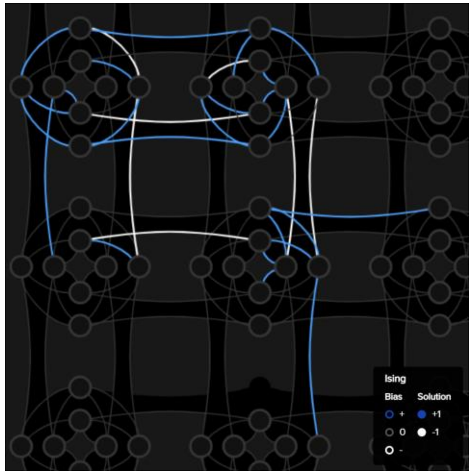


Problem data

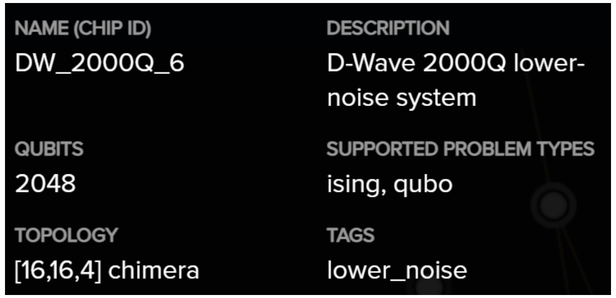
Candidate solutions



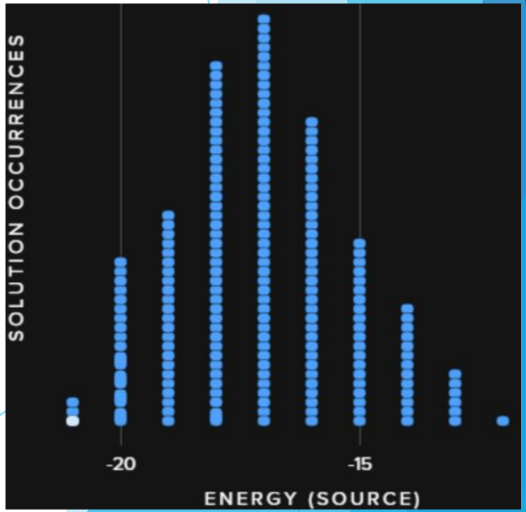
Decode min energy bitstrings



Ising model



Send to quantum annealer



Sampled bitstrings

Some Quantum Computing Providers

IBM

Qiskit

Azure Quantum

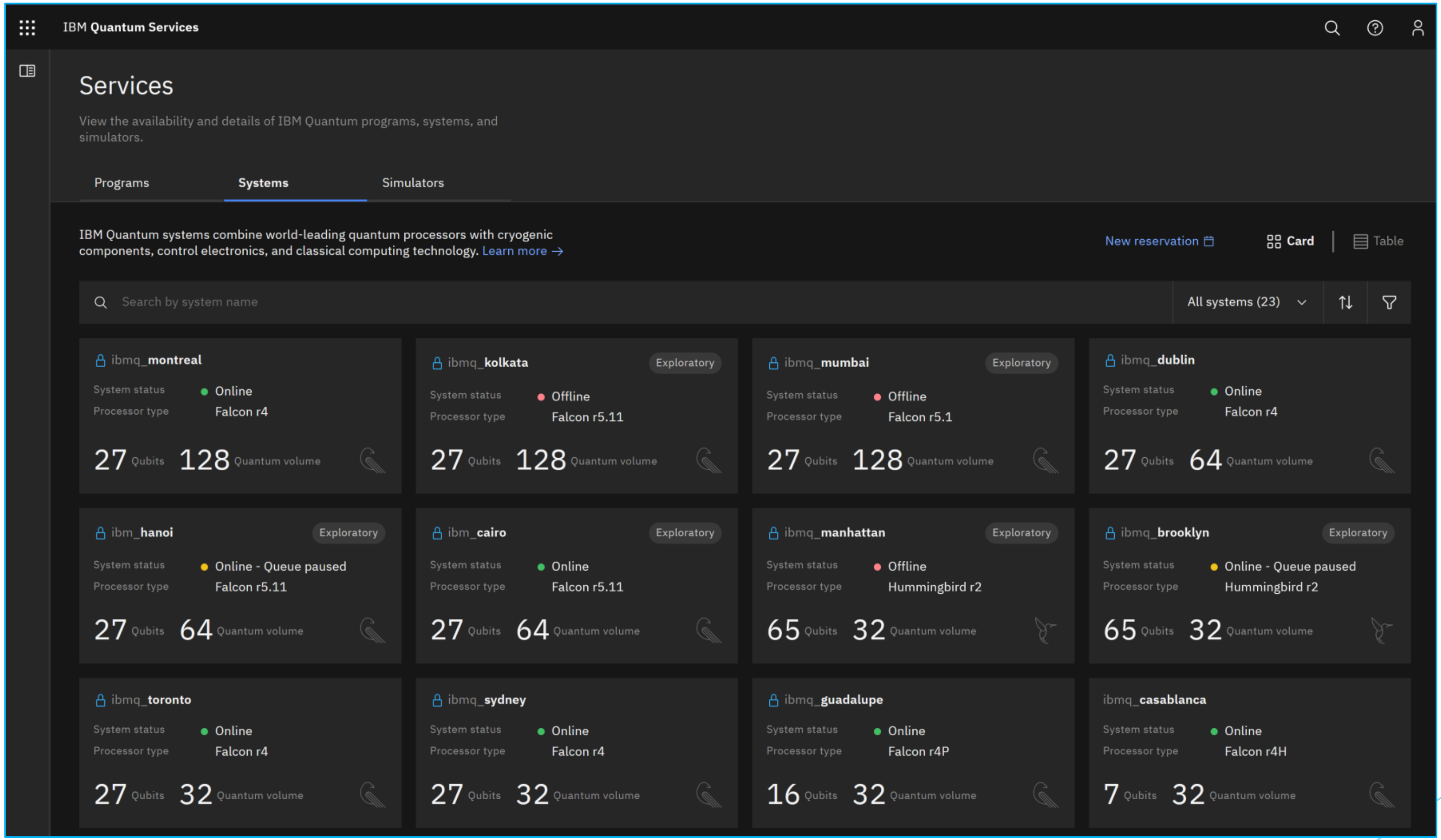
Google

TensorFlow

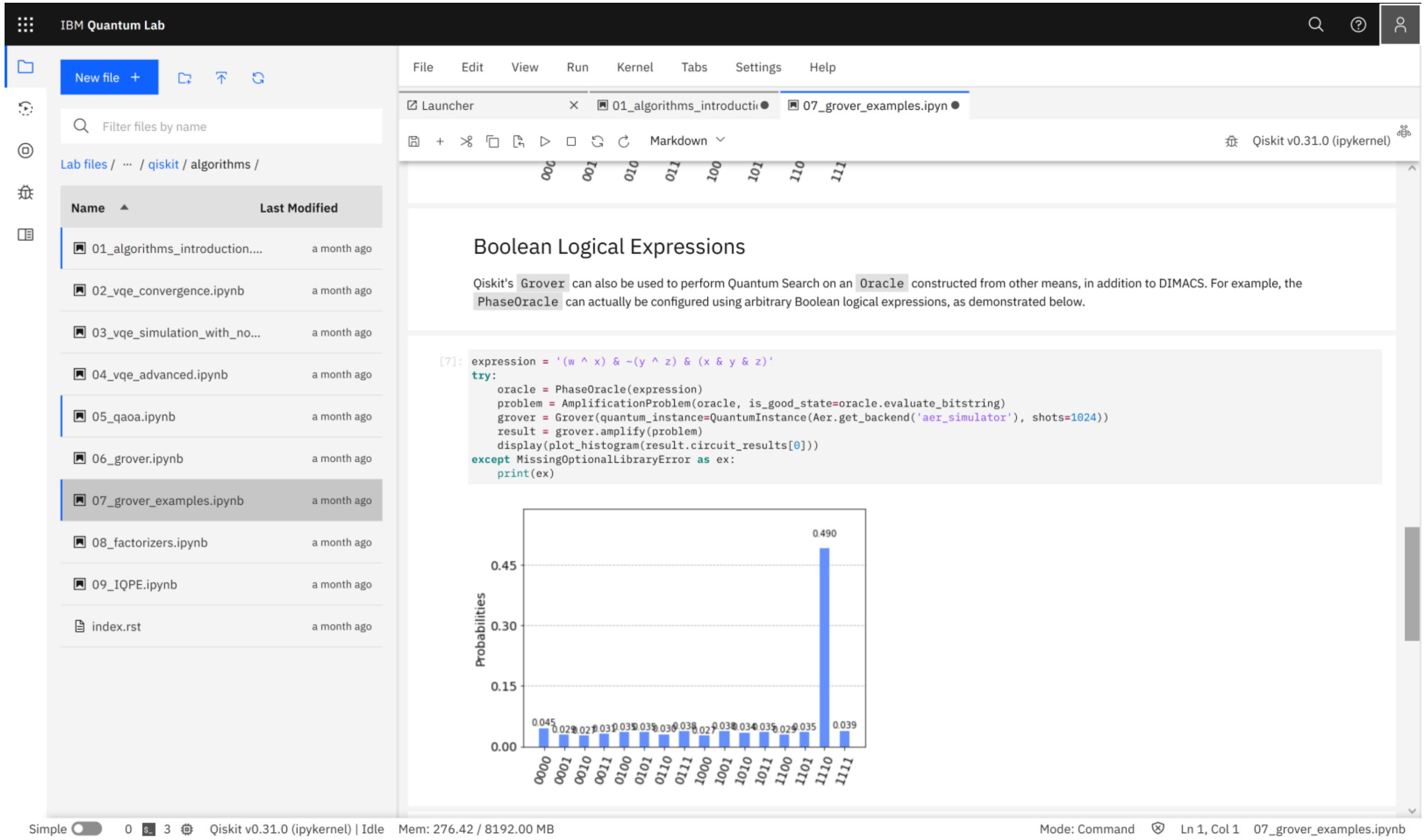
aws

D-Wave

(quantum annealing)

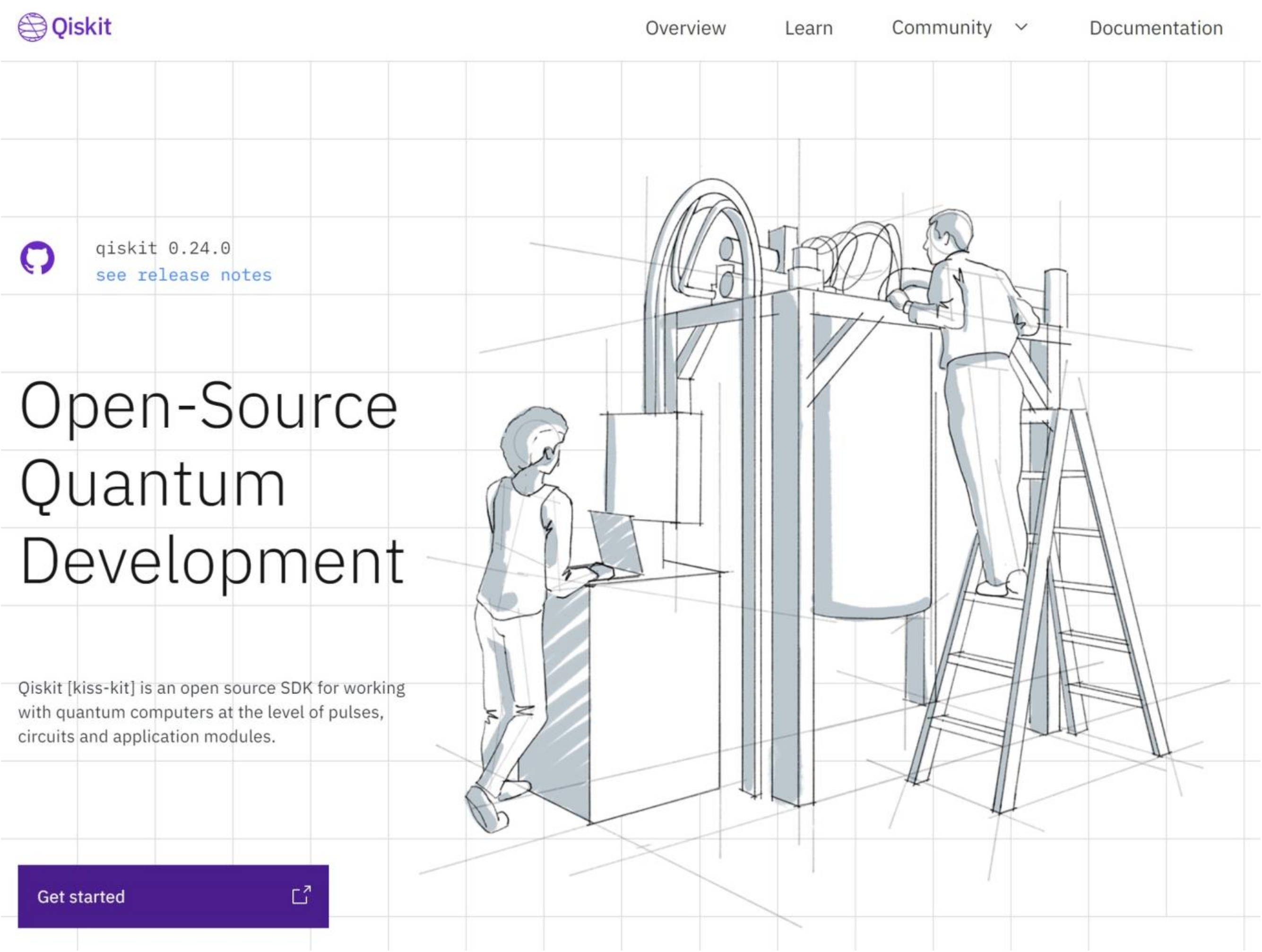


Credit: IBM - Quantum Services



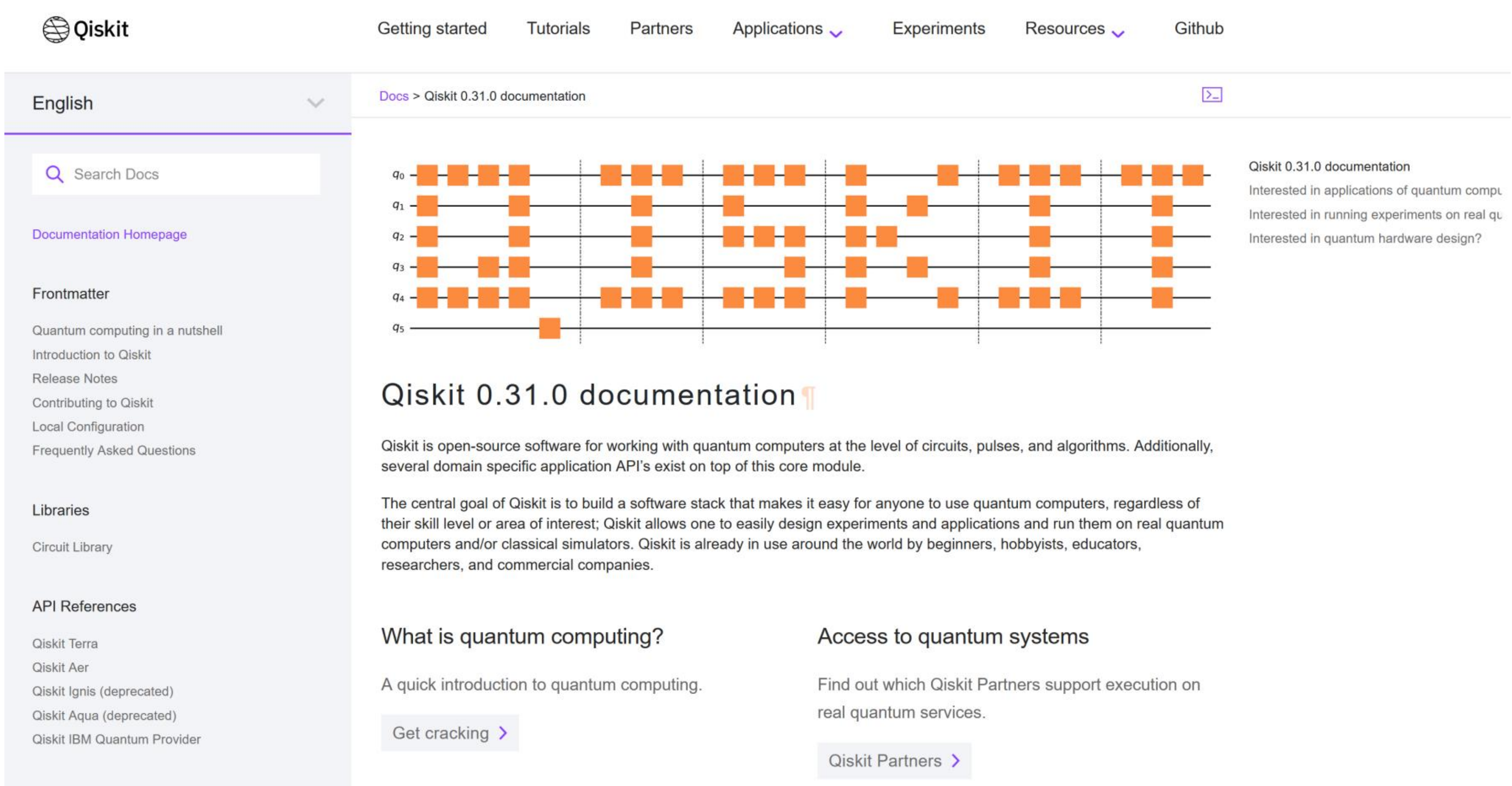
Credit: IBM - Quantum Lab

IBM Qiskit



Credit: [IBM - Qiskit]

IBM Qiskit



Qiskit

Getting started | Tutorials | Partners | Applications | Experiments | Resources | Github

English

Search Docs

Documentation Homepage

Frontmatter

- Quantum computing in a nutshell
- Introduction to Qiskit
- Release Notes
- Contributing to Qiskit
- Local Configuration
- Frequently Asked Questions

Libraries

- Circuit Library

API References

- Qiskit Terra
- Qiskit Aer
- Qiskit Ignis (deprecated)
- Qiskit Aqua (deprecated)
- Qiskit IBM Quantum Provider

Docs > Qiskit 0.31.0 documentation

q_0 q_1 q_2 q_3 q_4 q_5

Qiskit 0.31.0 documentation

Qiskit is open-source software for working with quantum computers at the level of circuits, pulses, and algorithms. Additionally, several domain specific application API's exist on top of this core module.

The central goal of Qiskit is to build a software stack that makes it easy for anyone to use quantum computers, regardless of their skill level or area of interest; Qiskit allows one to easily design experiments and applications and run them on real quantum computers and/or classical simulators. Qiskit is already in use around the world by beginners, hobbyists, educators, researchers, and commercial companies.

What is quantum computing?

A quick introduction to quantum computing.

Get cracking >

Access to quantum systems

Find out which Qiskit Partners support execution on real quantum services.

Qiskit Partners >

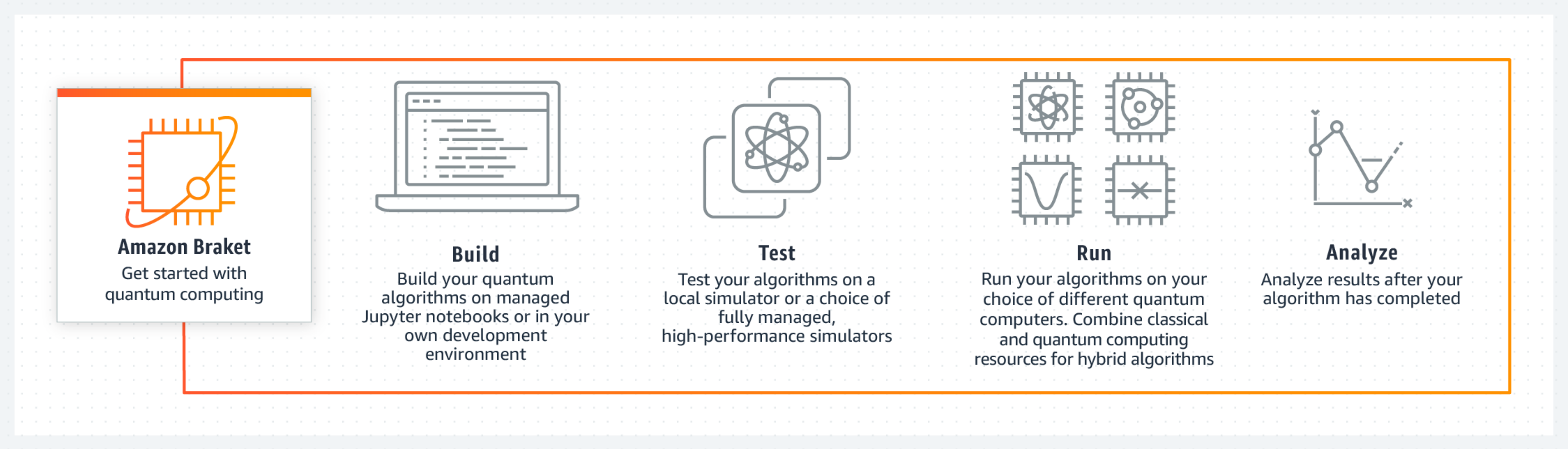
Qiskit 0.31.0 documentation

Interested in applications of quantum compu

Interested in running experiments on real qu

Interested in quantum hardware design?

Credit: IBM – Qiskit documentation



Credit: Amazon AWS - Braket

Amazon Braket Hardware Providers

Amazon Braket provides AWS customers access to multiple types of quantum computing technologies from quantum hardware providers, including gate-based quantum computers and quantum annealing systems. Learn more about these quantum hardware providers below.

D-Wave's technology uses quantum annealing to solve problems represented as mathematical functions (resembling a landscape of peaks and valleys). Their QPUs are built from a network of interconnected superconducting flux qubits. Each qubit is made from a tiny loop of metal interrupted by a Josephson Junction.

Learn more »

IonQ's trapped-ion approach to quantum computing starts with ionized ytterbium atoms. Two internal states of these identical atoms make up the qubits, the basic unit of quantum information. The execution of computational tasks is accomplished by programming the sequence of laser pulses used to implement each quantum gate operation.

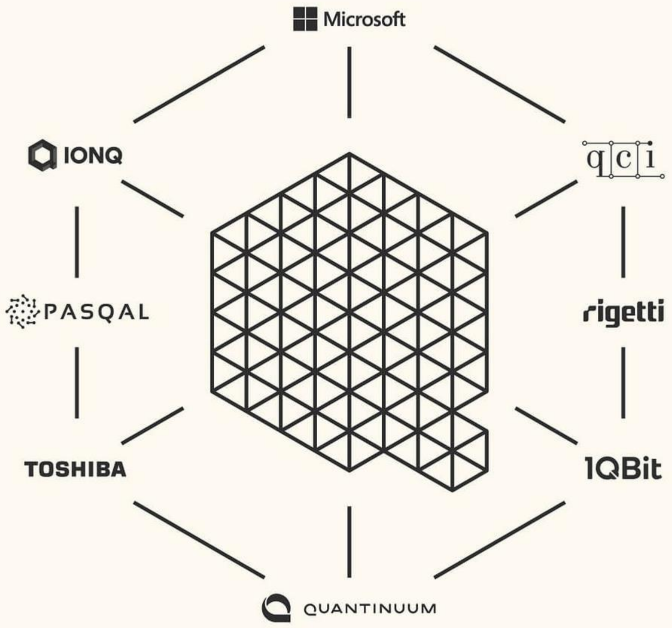
Learn more »

Rigetti quantum processors are universal, gate-based machines based on superconducting qubits. The Rigetti Aspen series of chips feature tileable lattices of alternating fixed-frequency and tunable superconducting qubits within a scalable architecture.

Learn more »

Credit: Amazon AWS - Braket

Azure Quantum



Richest development environment

Enjoy the richest development environment for quantum computing:

- Support for the most popular quantum SDKs: Q#, Qiskit and Cirq.
- Write once and run on multiple hardware architectures.
- Send native circuits to QPUs.
- World-class samples and curriculum.
- Free hosted Jupyter notebooks to get started within minutes.
- Full state and open systems and stabiliser simulators.
- Noisy simulator (Quantinuum).
- High-performance hybrid quantum computing with quantum intermediate representation (QIR).

Explore the [Quantum Development Kit]

Credit: [Microsoft Azure Quantum]

Google Quantum AI

Cirq

An open source framework for programming quantum computers

Cirq is a Python software library for writing, manipulating, and optimizing quantum circuits, and then running them on quantum computers and quantum simulators. Cirq provides useful abstractions for dealing with today's noisy intermediate-scale quantum computers, where details of the hardware are vital to achieving state-of-the-art results.

Get started with Cirq

GitHub repository

```python
import cirq

# Pick a qubit.
qubit = cirq.GridQubit(0, 0)

# Create a circuit
circuit = cirq.Circuit(
    cirq.X(qubit)**0.5,  # Square root of NOT.
    cirq.measure(qubit, key='m')  # Measurement.
)
print("Circuit:")
print(circuit)

# Simulate the circuit several times.
simulator = cirq.Simulator()
result = simulator.run(circuit, repetitions=20)
print("Results:")
print(result)
```

Credit: [Google Quantum AI - Cirq]

TensorFlow

TensorFlow Quantum is a library for hybrid quantum-classical machine learning.

TensorFlow Quantum (TFQ) is a quantum machine learning library for rapid prototyping of hybrid quantum-classical ML models. Research in quantum algorithms and applications can leverage Google's quantum computing frameworks, all from within TensorFlow.

TensorFlow Quantum focuses on *quantum data* and building *hybrid quantum-classical models*. It integrates quantum computing algorithms and logic designed in Cirq, and provides quantum computing primitives compatible with existing TensorFlow APIs, along with high-performance quantum circuit simulators. Read more in the TensorFlow Quantum white paper.

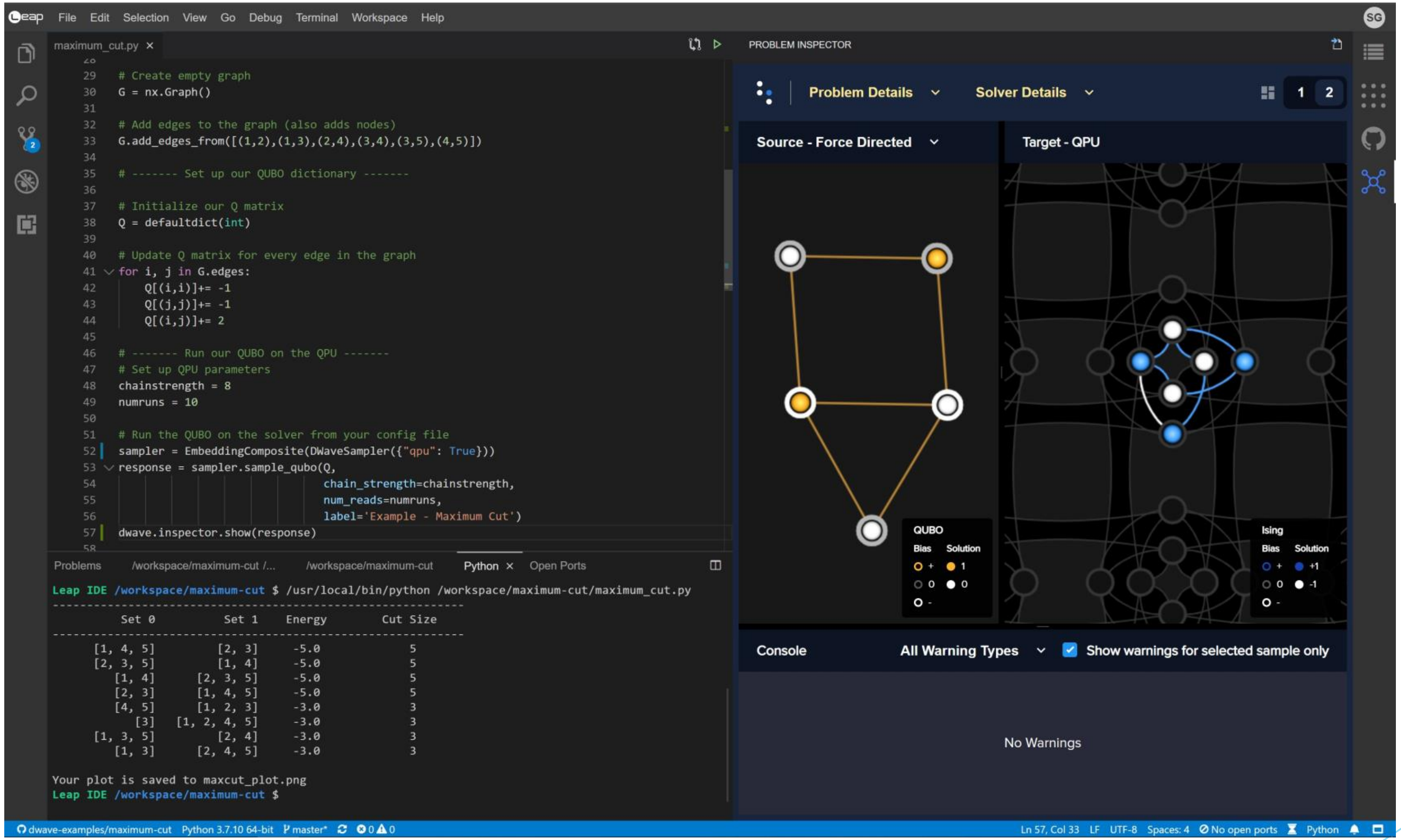
Start with the overview, then run the notebook tutorials.

```
# A hybrid quantum-classical model.
model = tf.keras.Sequential([
    # Quantum circuit data comes in inside of tensors.
    tf.keras.Input(shape=(), dtype=tf.dtypes.string),

    # Parametrized Quantum Circuit (PQC) provides output
    # data from the input circuits run on a quantum computer.
    tfq.layers.PQC(my_circuit, [cirq.Z(q1), cirq.X(q0)]),

    # Output data from quantum computer passed through model.
    tf.keras.layers.Dense(50)
])
```

Credit: Google – TensorFlow Quantum



Credit: [D-Wave - Leap]