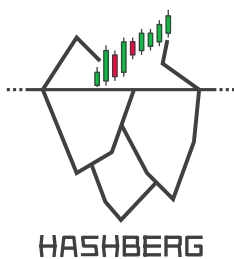


# 1. Gate-based QC

Dr. Stefano Gogioso



# Your Lecturer



Dr. Stefano Gogioso

Lecturer in  
Quantum  
Computing

@

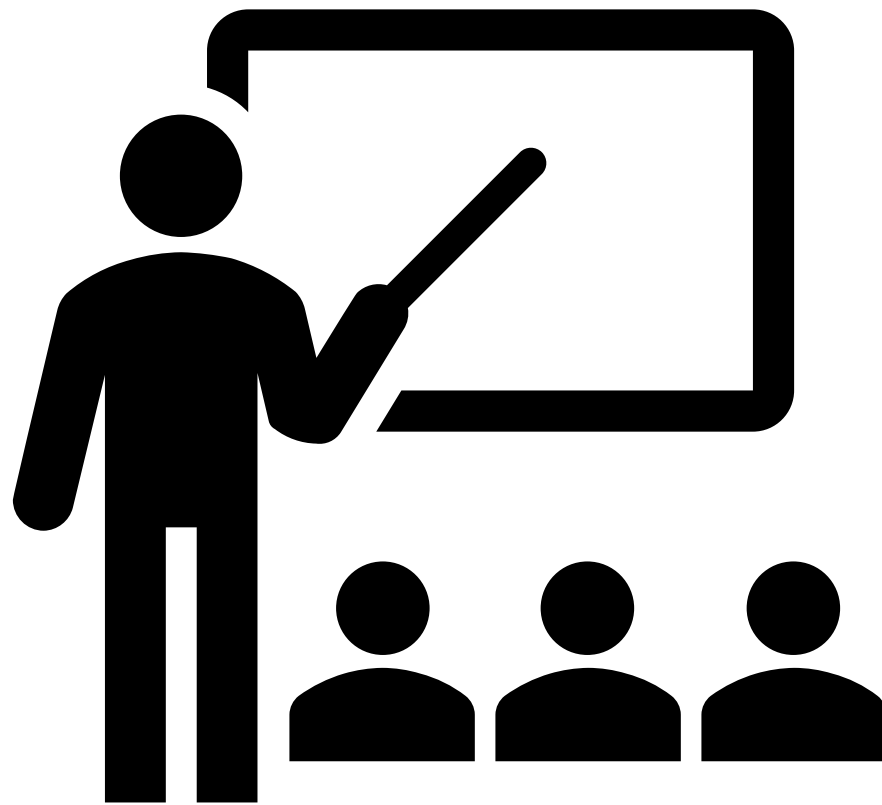


Quantum  
Software and  
Decentralised  
Finance

@



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.

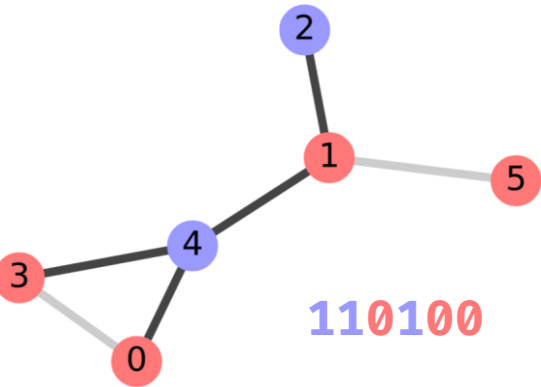
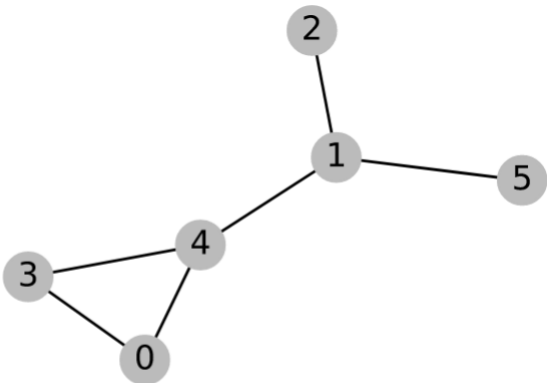
# The Goals of Quantum Computing (QC)

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.

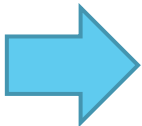
# QC Workflow

Problem data

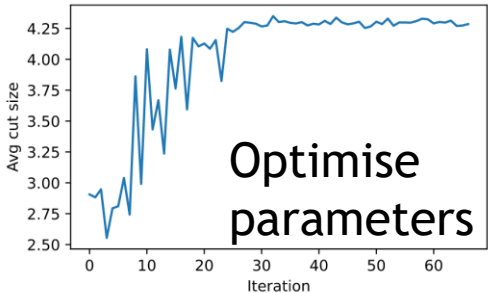
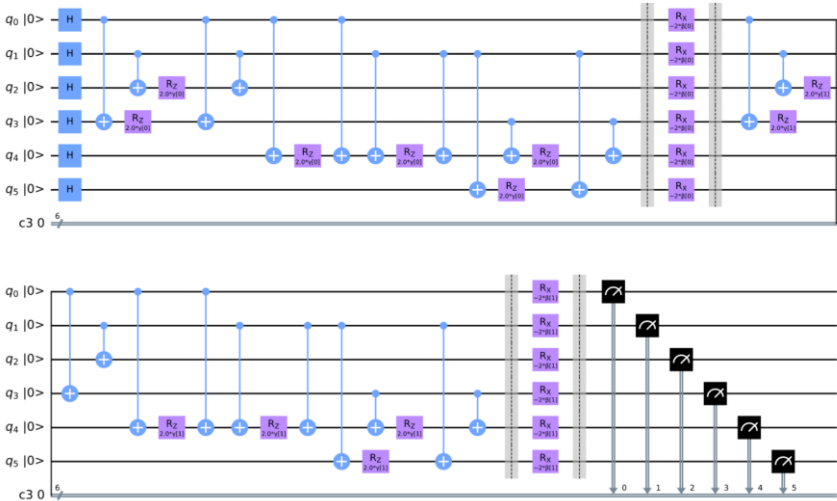


Candidate solutions

Encode



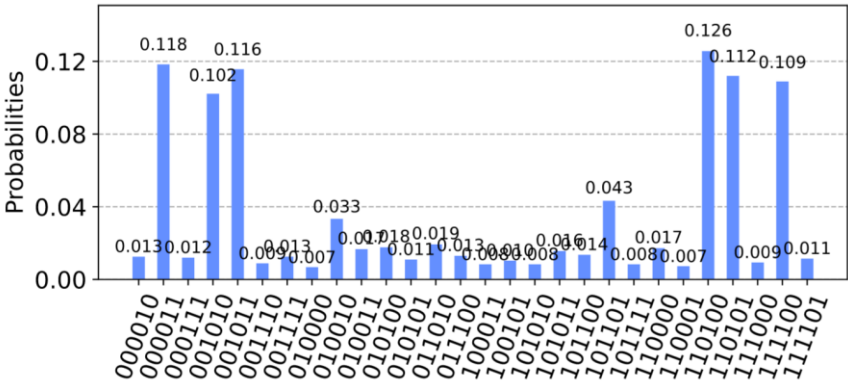
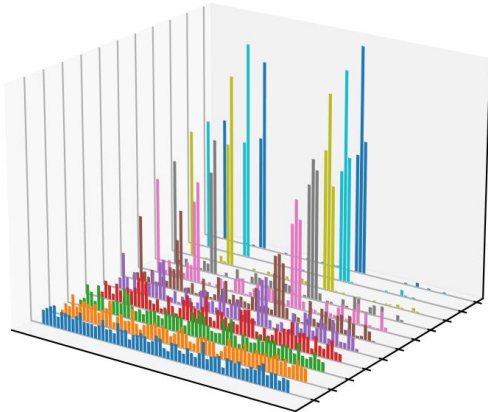
Quantum Circuit



Optimise parameters



Send to QPU

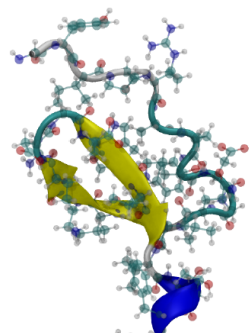


Sampled bitstrings

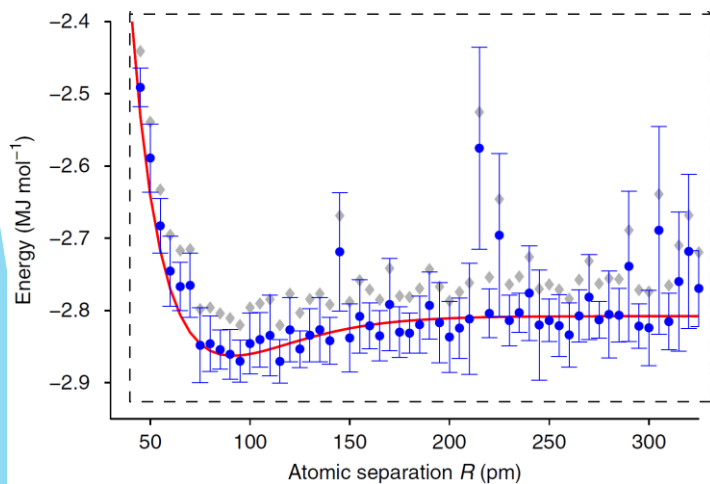
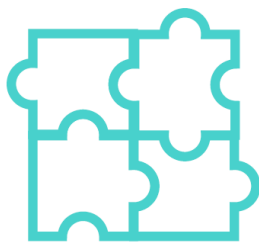
# QC Workflow

Molecular data

Encode



=



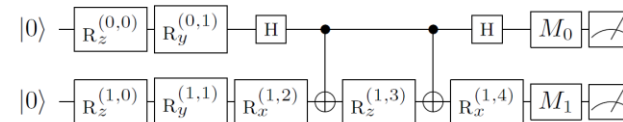
Candidate energy curves

Decode

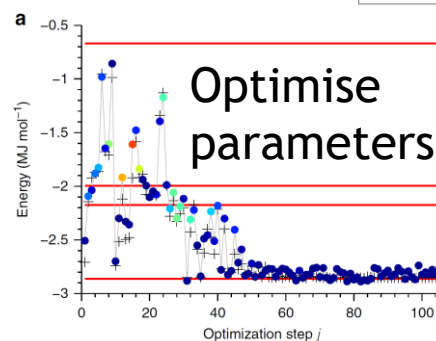
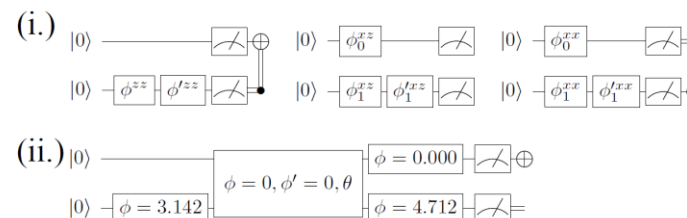


## Quantum Circuit

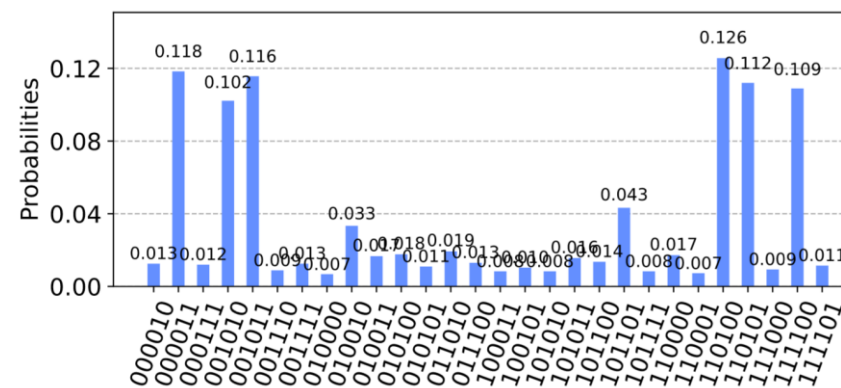
(1) Circuit compilation



(2) Circuit optimization



Send to QPU



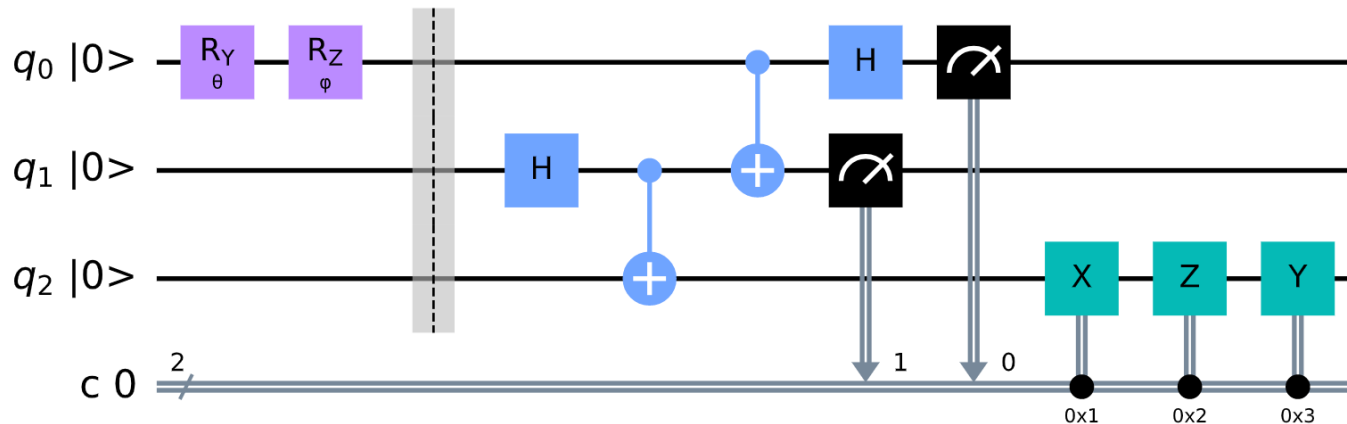
Sampled bitstrings

arXiv:2102.07045

doi:10.1038/nature5213

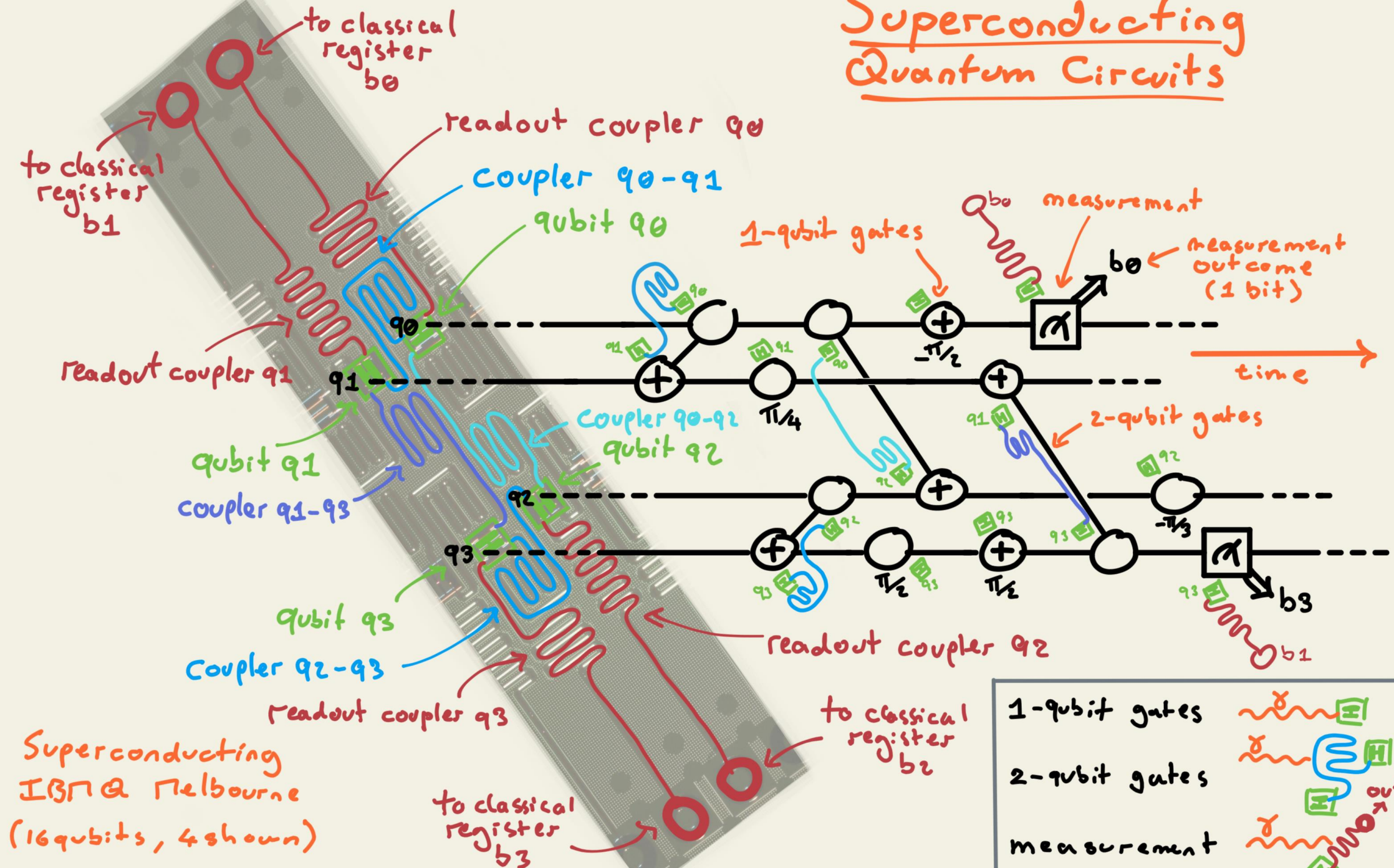
# Quantum Circuits

```
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)
```

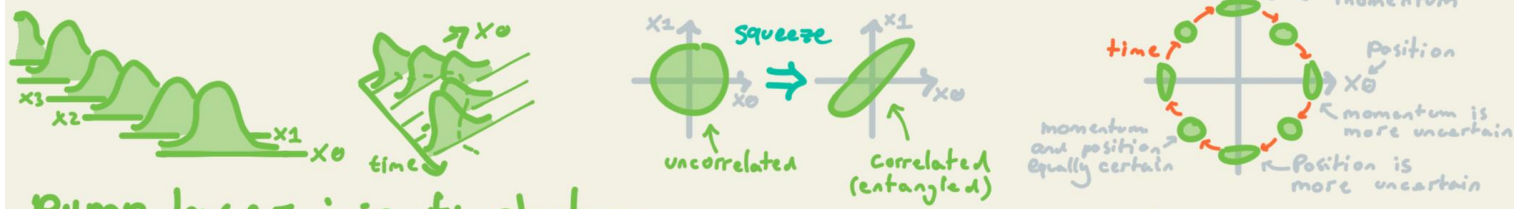




# Superconducting Quantum Circuits

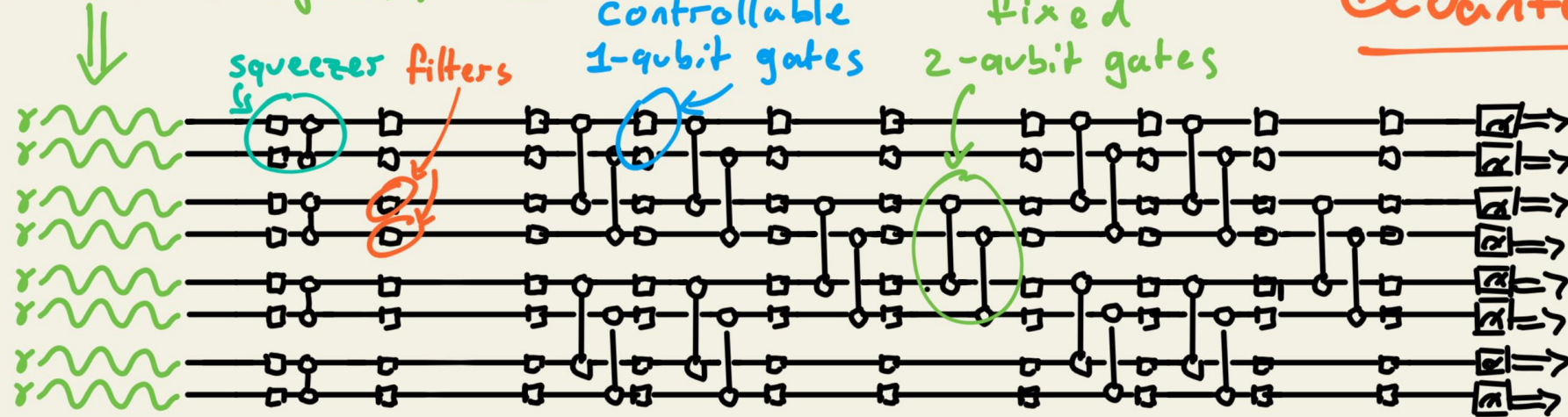




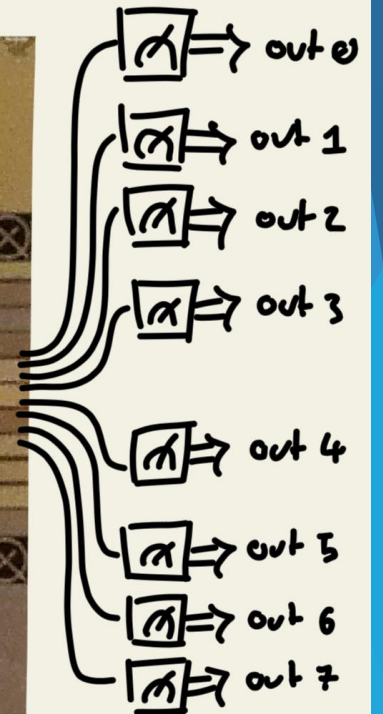
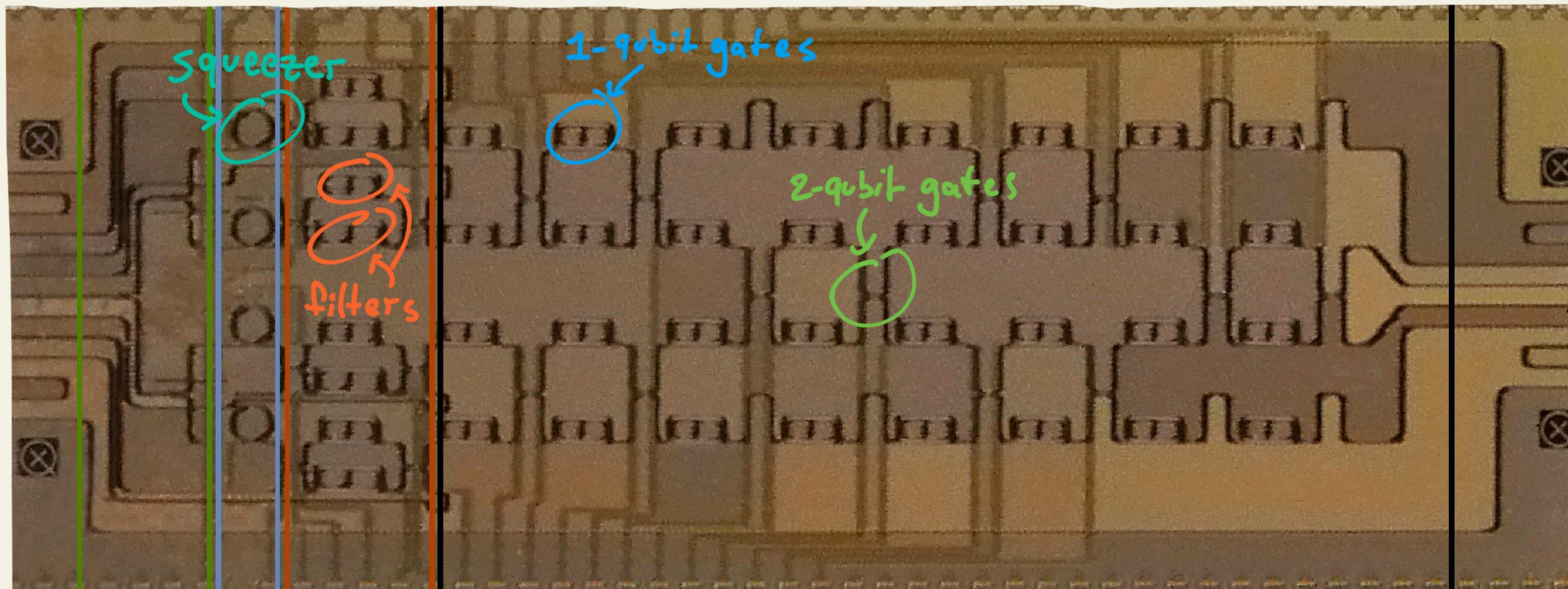


# Photonic Quantum Circuits

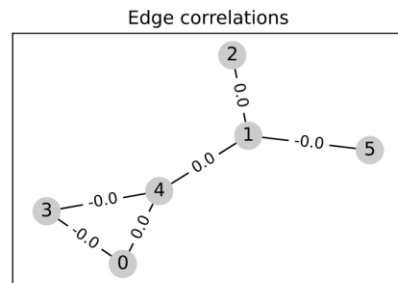
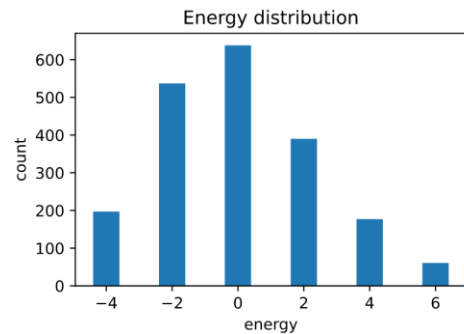
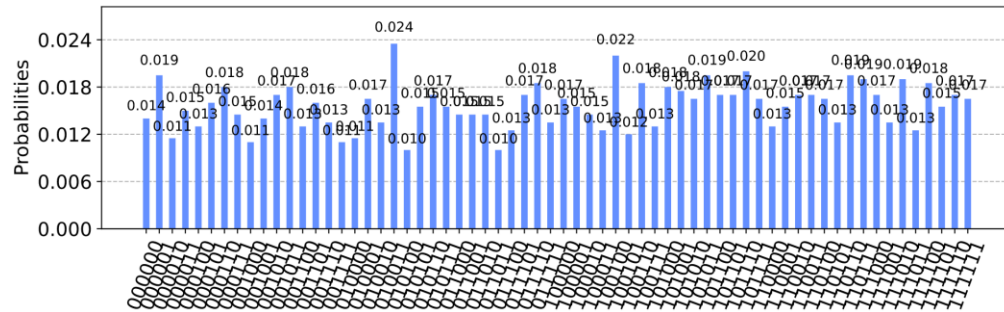
pump laser injects photons



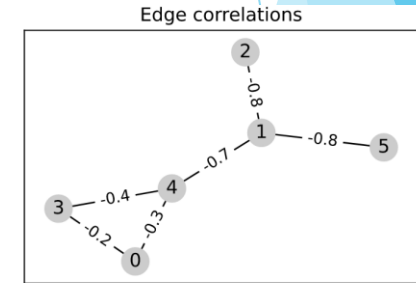
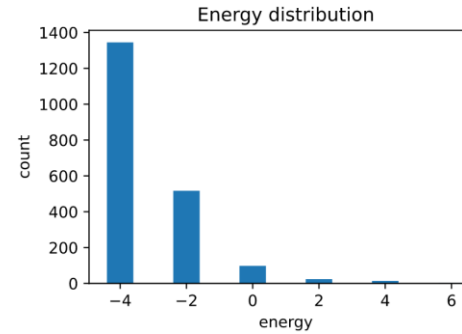
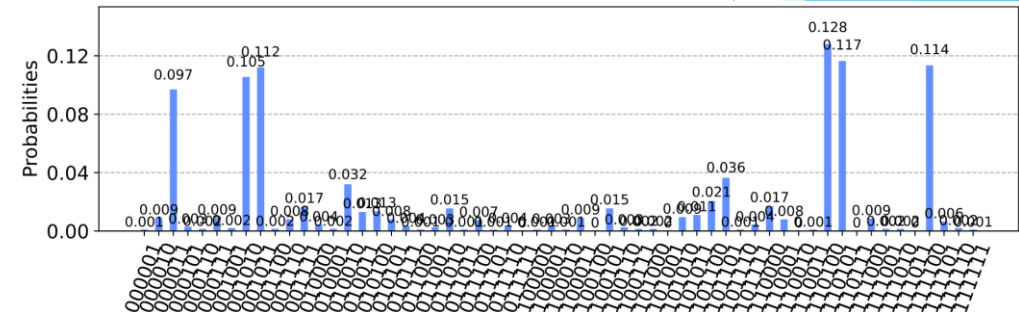
measurements performed off-chip.  
- homodyne  
- photon number



# Quantum circuits are used to create complex correlation patterns



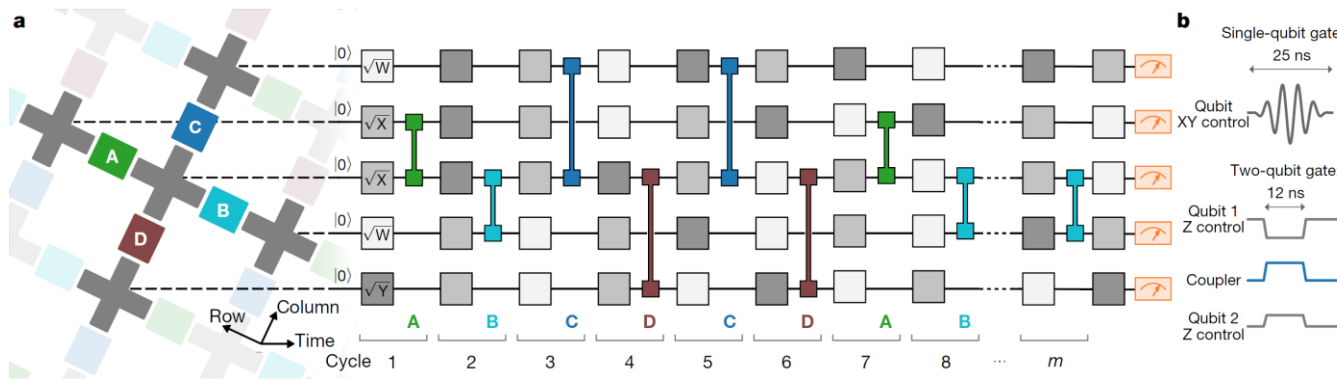
Optimise



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

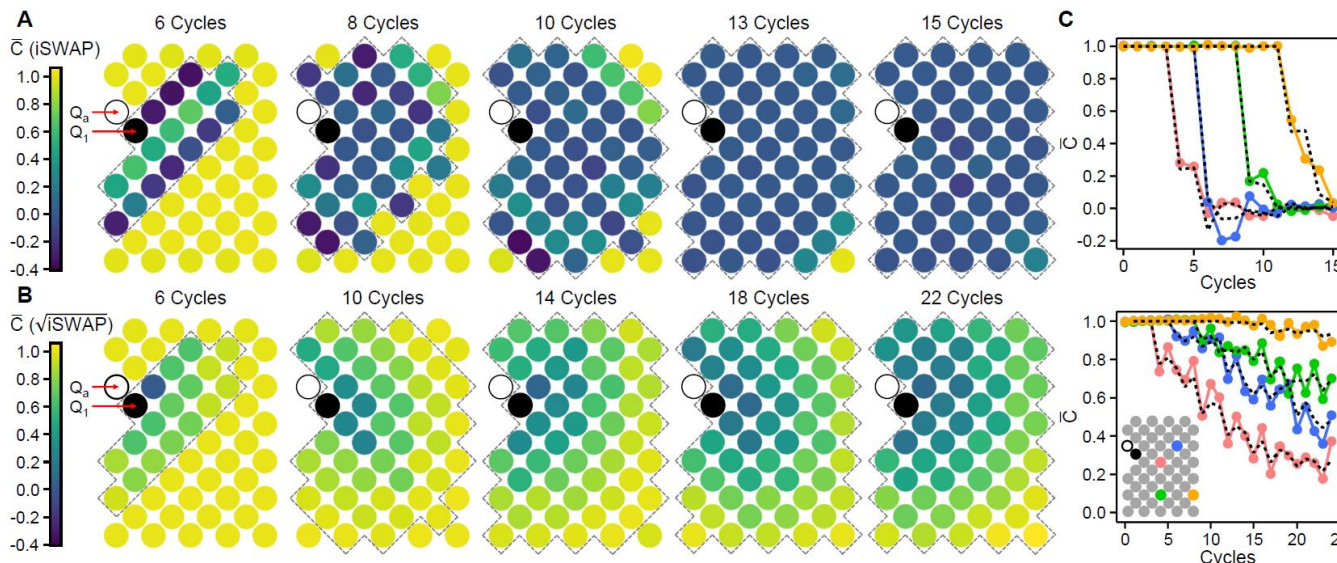


# Quantum circuits are used to create complex correlation patterns



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**

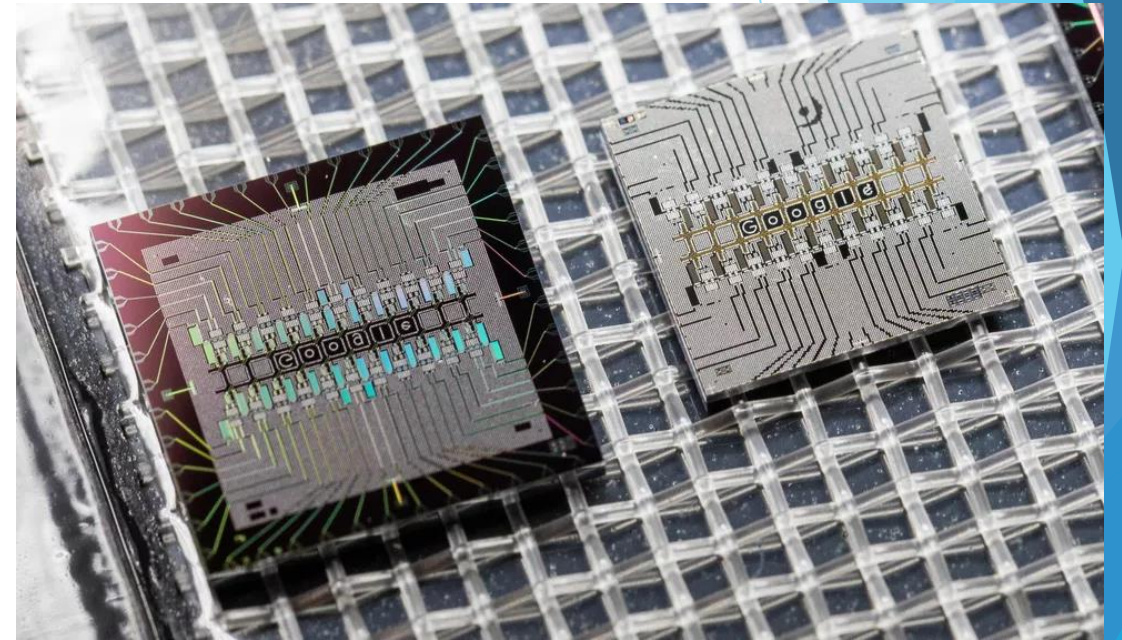
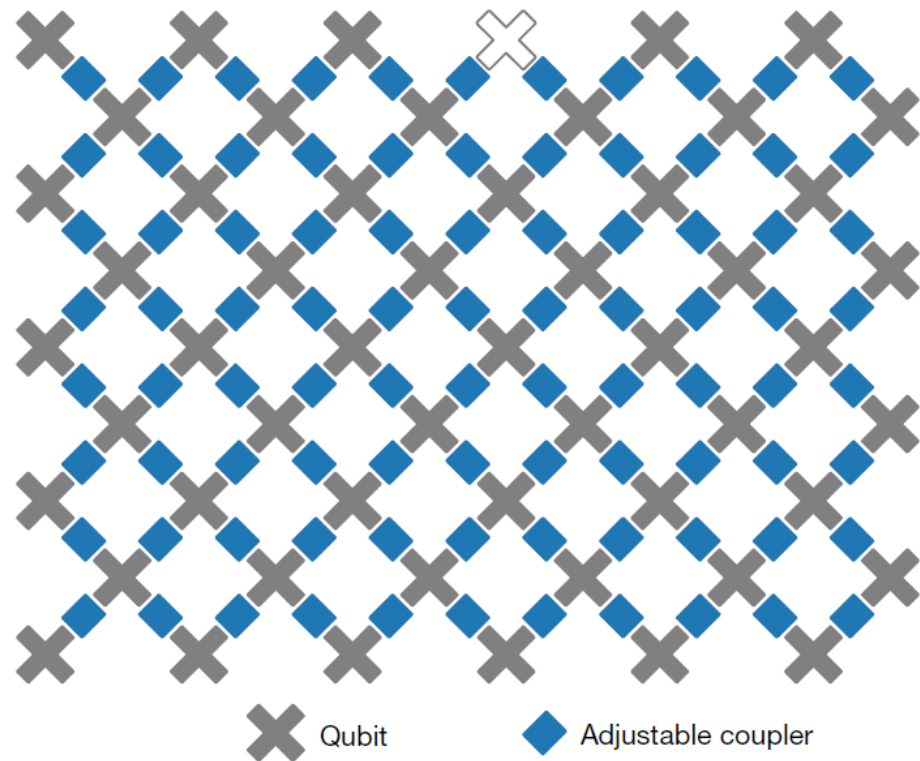
 **IONQ**

 **PASQAL**

**D::wave**

 **XANADU**

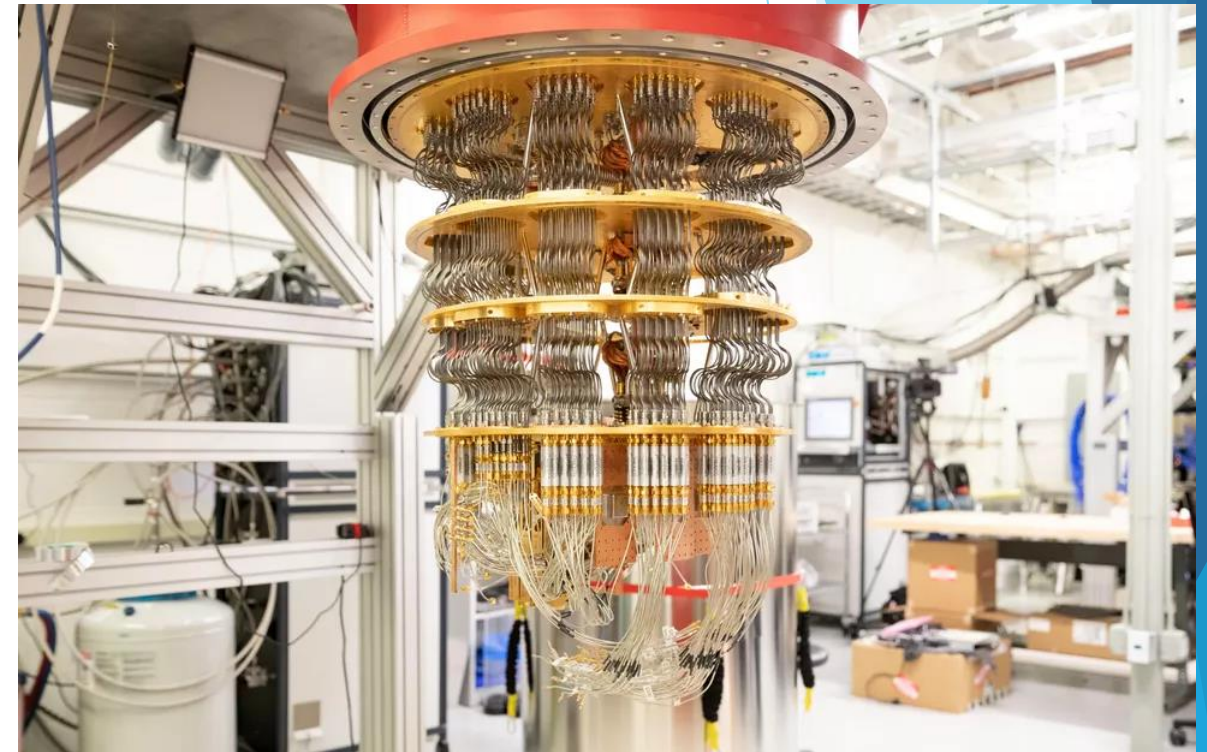
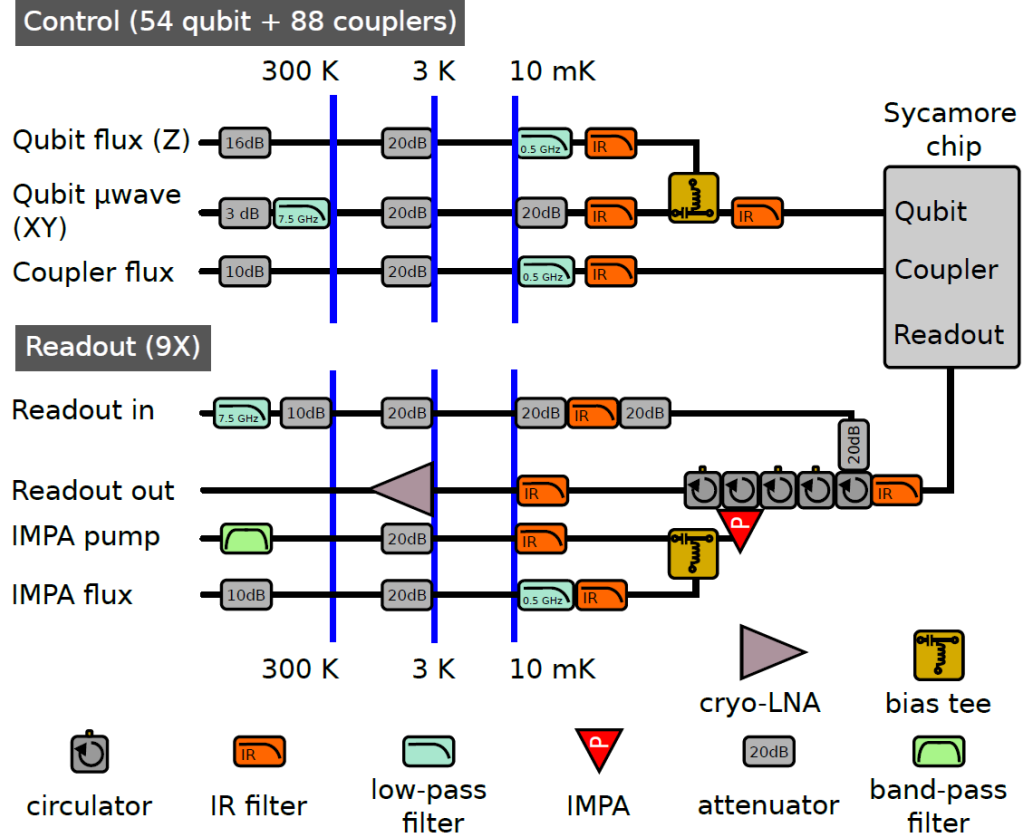
# Superconducting QC



Credit: Stephen Shankland/CNET



# Superconducting QC

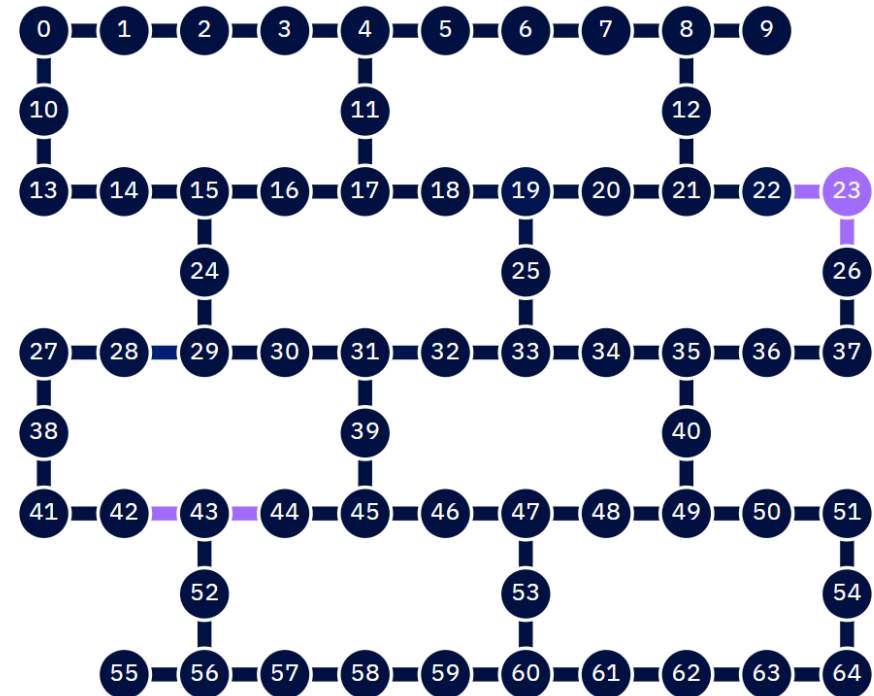


Credit: Stephen Shankland/CNET

# Superconducting QC

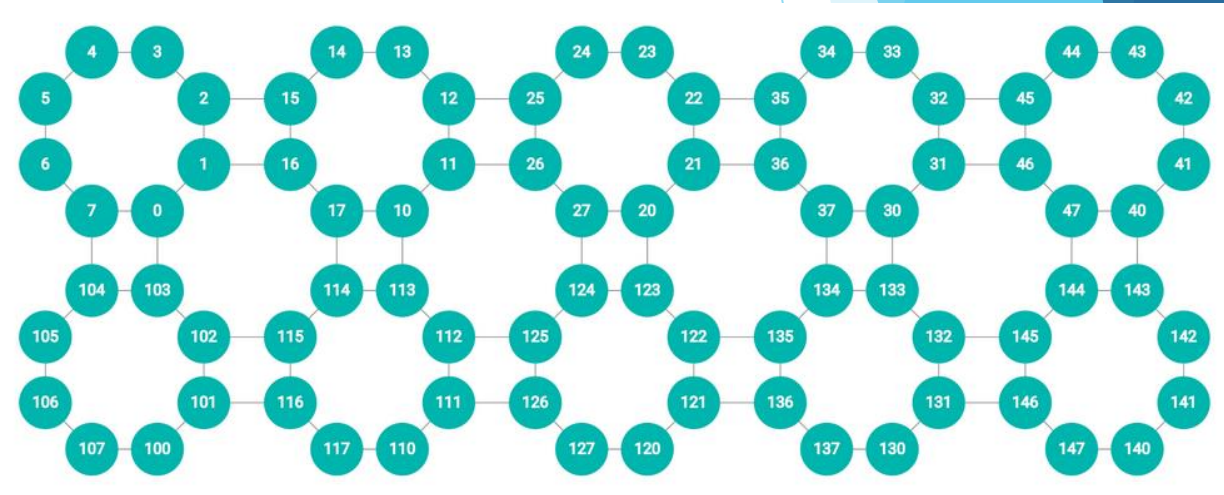


**IBM.**



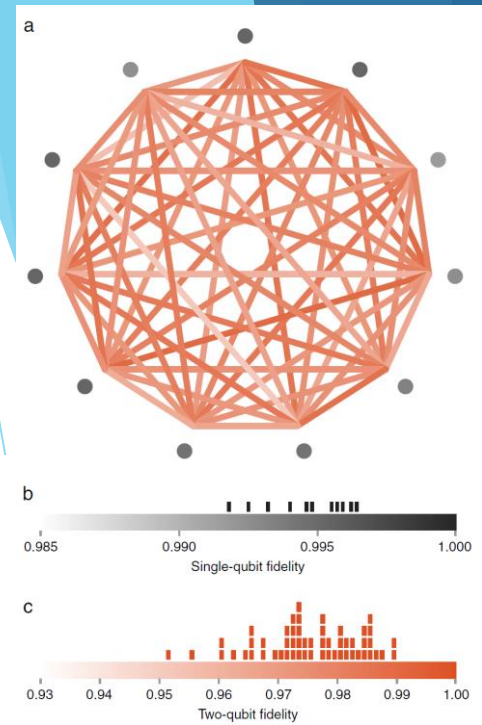
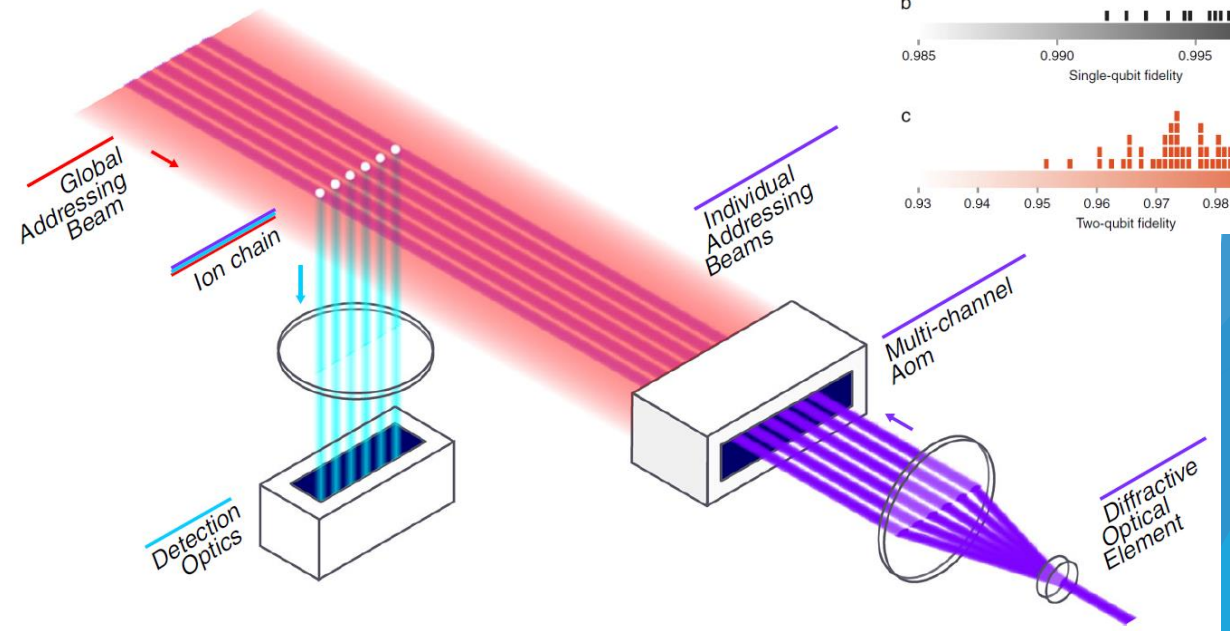
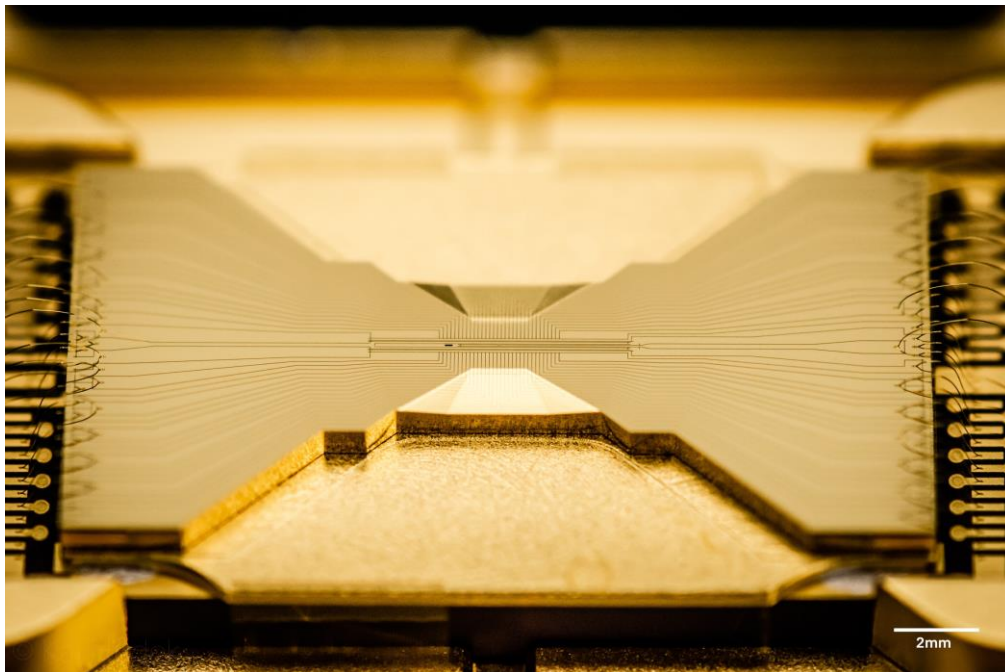


# Superconducting QC



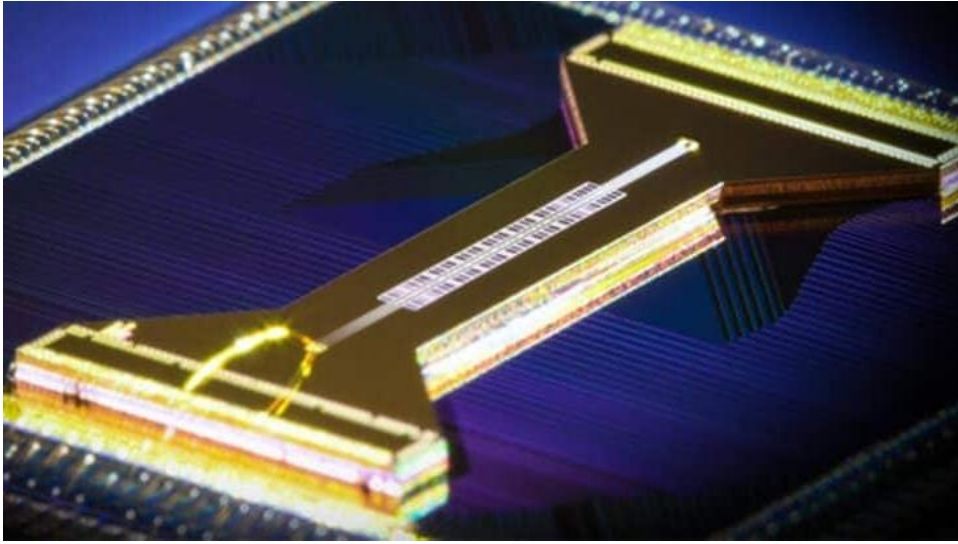
**rigetti**

# Ion Trap QC

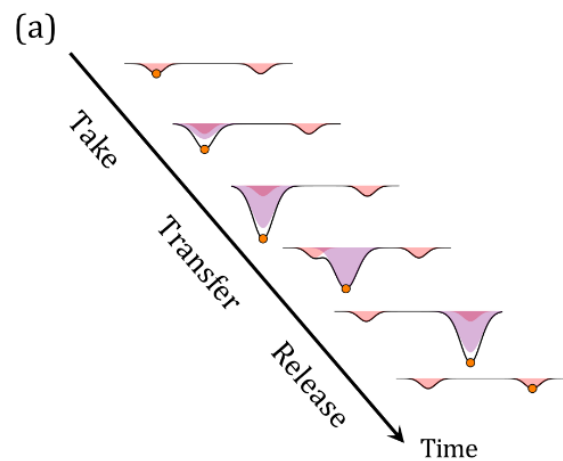




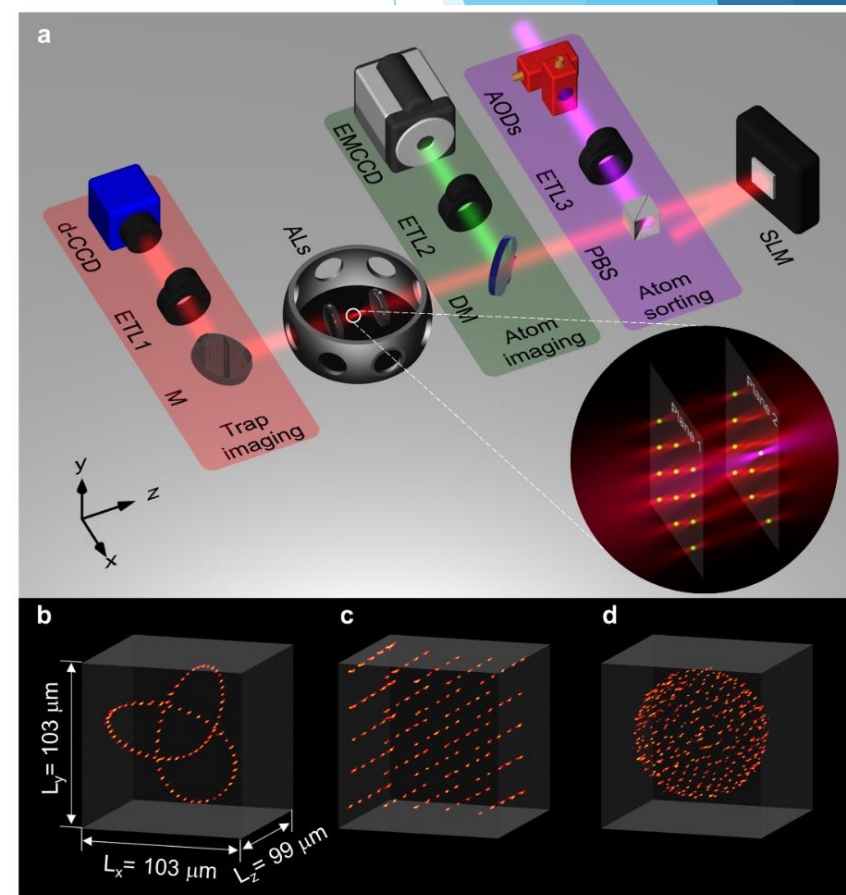
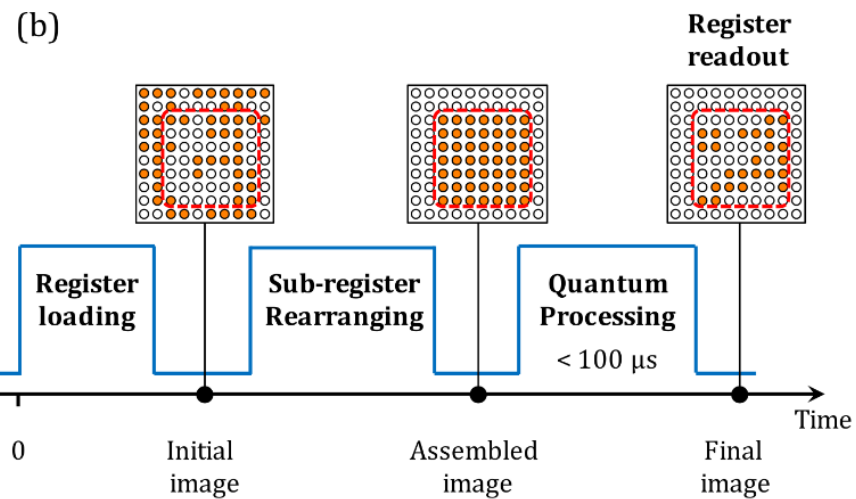
# Ion Trap QC



# Neutral Atoms QC



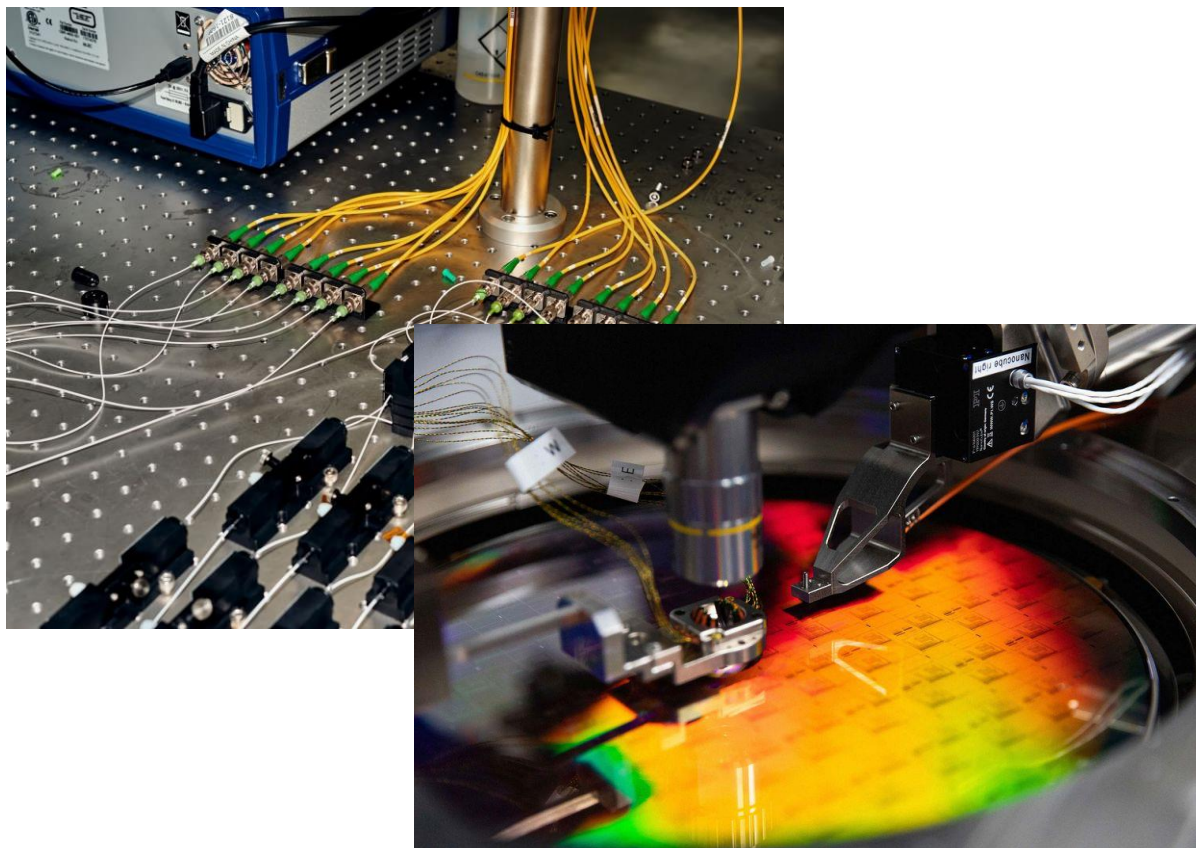
arXiv:2006.12326



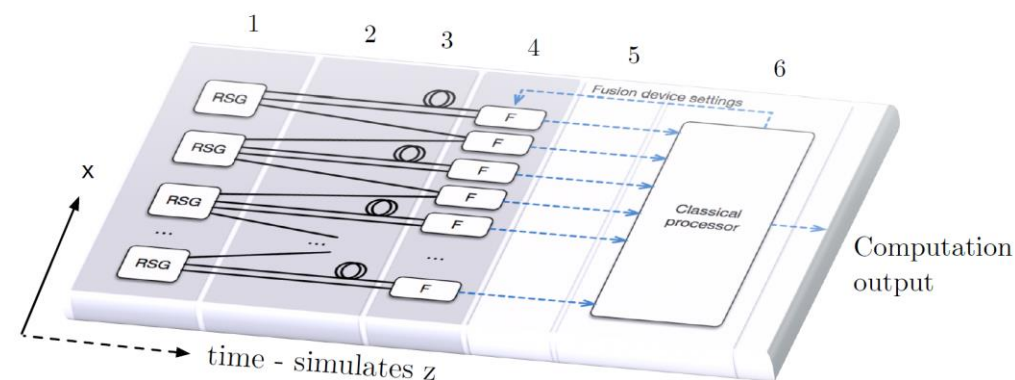
arXiv:1712.02727









# Photonic QC



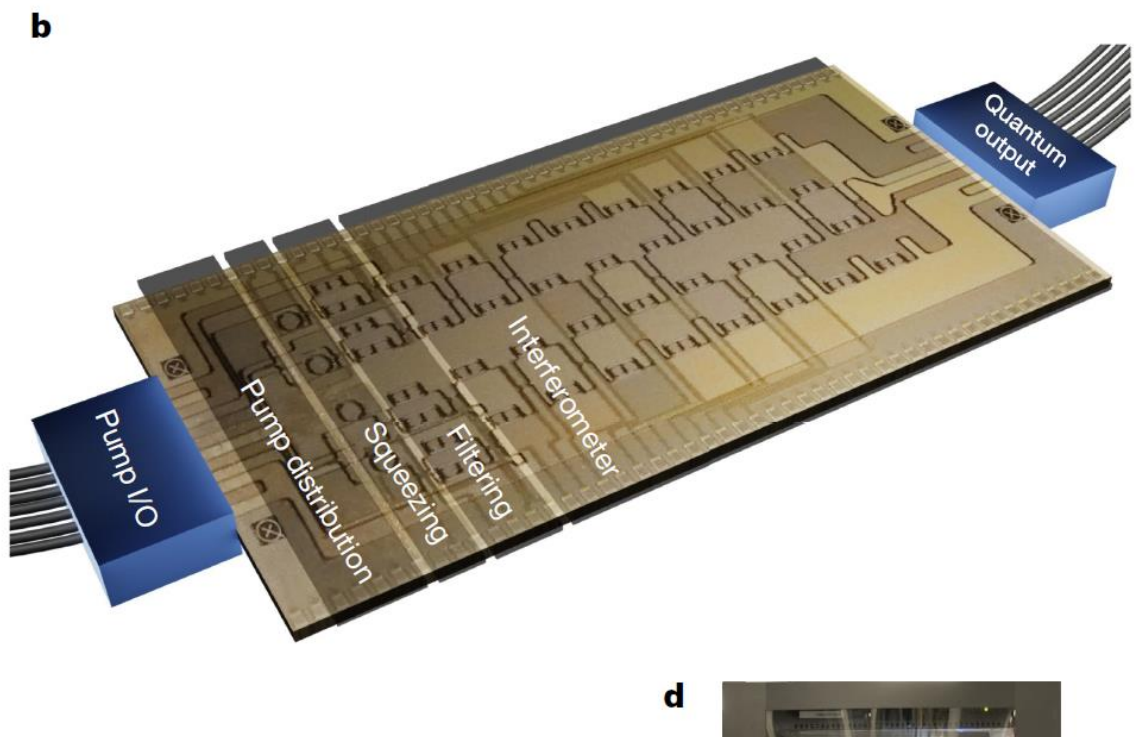
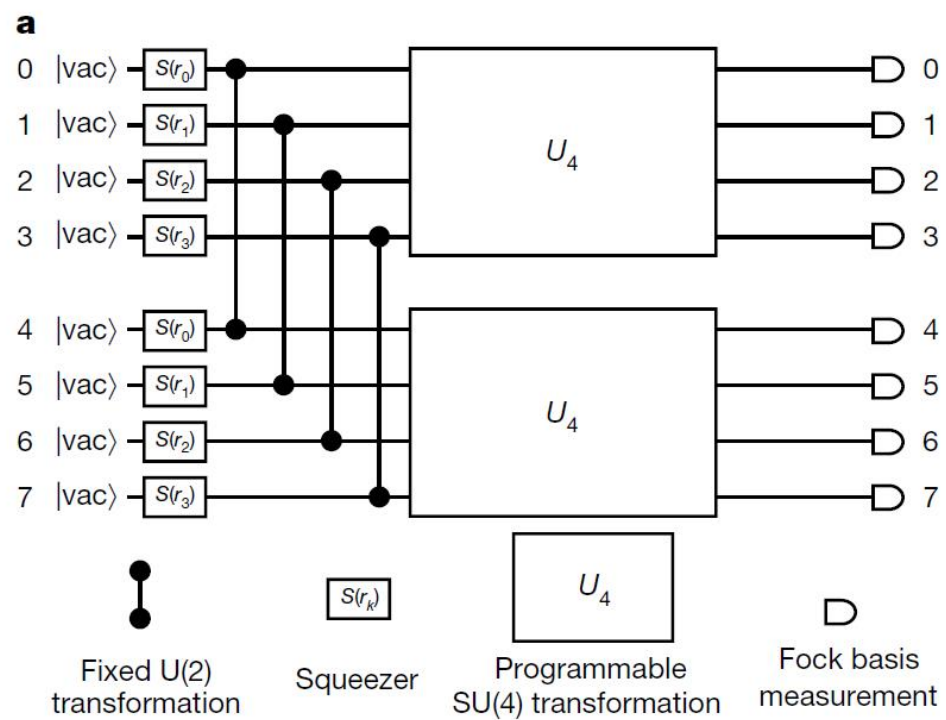
b) Fusion based quantum computing architecture



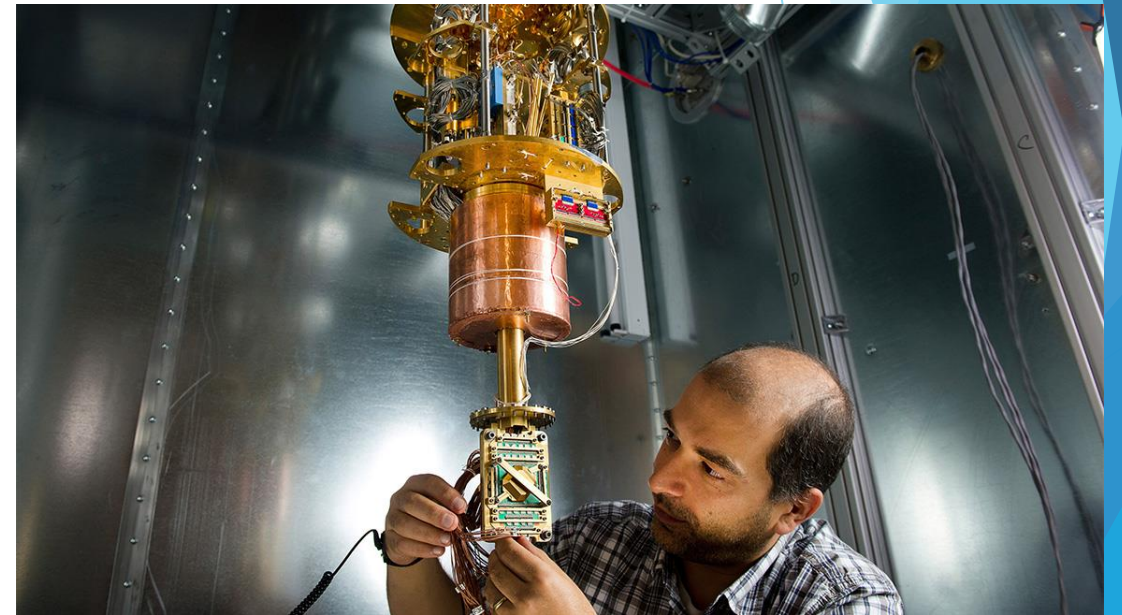
- |   |  |   |  |
|---|--|---|--|
|    | <b>1.</b> Resource state generators repeatedly create entangled states |    | <b>4.</b> Fusion devices perform reconfigurable measurements       |
|    | <b>2.</b> Fusion network router sends qubits to fusion locations.      |    | <b>5.</b> Classical signal transmission                            |
|  | <b>3.</b> Delays - delay a qubit for one or more clockcycles.          |  | <b>6.</b> Classical Processor - decoding and algorithm feedforward |

arXiv:2101.09310

# Photonic QC



# Quantum Annealing (QA)

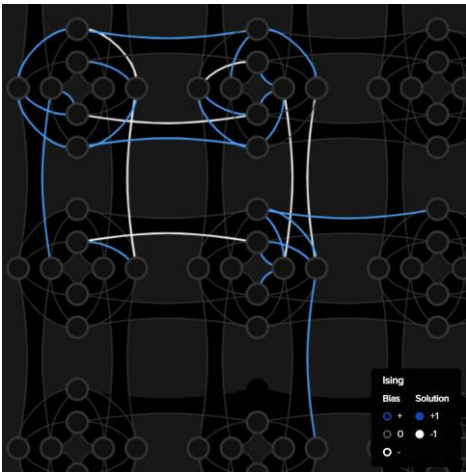


**D-Wave**

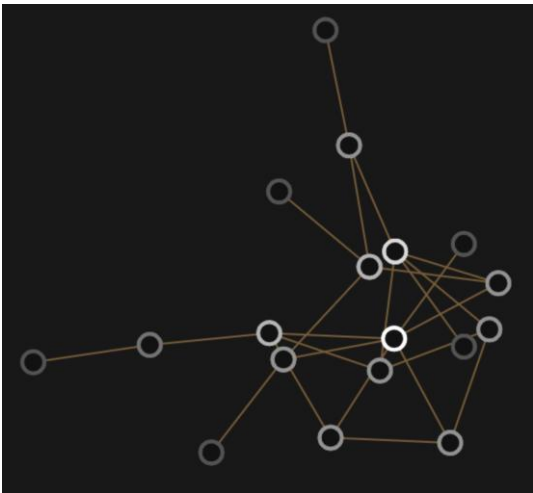


# QA Workflow

Encode



Ising model

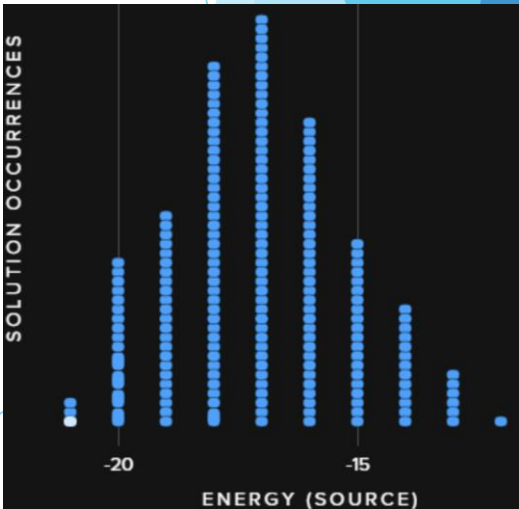
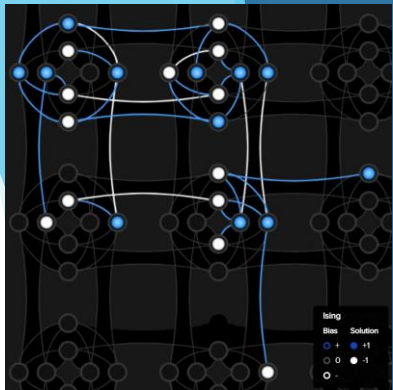


Problem data



Candidate solutions

Decode min energy bitstrings



Sampled bitstrings



NAME (CHIP ID)	DESCRIPTION
DW_2000Q_6	D-Wave 2000Q lower-noise system
QUBITS	SUPPORTED PROBLEM TYPES
2048	ising, qubo
TOPOLOGY	TAGS
[16,16,4] chimera	lower_noise

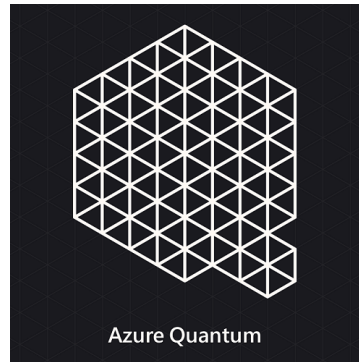
Send to quantum annealer





# Some Quantum Computing Providers

IBM.



Google



aws

D::wave  
(quantum annealing)



IBM Quantum Services

Services

View the availability and details of IBM Quantum programs, systems, and simulators.

Programs

Systems

Simulators

IBM Quantum systems combine world-leading quantum processors with cryogenic components, control electronics, and classical computing technology. [Learn more →](#)

New reservation

Card

Table

Search by system name

All systems (23)

<div><div>ibmq_montreal</div><div>System status <span>Online</span></div><div>Processor type Falcon r4</div><div>27 Qubits 128 Quantum volume</div></div>	<div><div>ibmq_kolkata</div><div>System status <span>Offline</span></div><div>Processor type Falcon r5.11</div><div>27 Qubits 128 Quantum volume</div></div>	<div><div>ibmq_mumbai</div><div>System status <span>Offline</span></div><div>Processor type Falcon r5.1</div><div>27 Qubits 128 Quantum volume</div></div>	<div><div>ibmq_dublin</div><div>System status <span>Online</span></div><div>Processor type Falcon r4</div><div>27 Qubits 64 Quantum volume</div></div>
<div><div>ibmq_hanoi</div><div>System status <span>Online - Queue paused</span></div><div>Processor type Falcon r5.11</div><div>27 Qubits 64 Quantum volume</div></div>	<div><div>ibmq_cairo</div><div>System status <span>Online</span></div><div>Processor type Falcon r5.11</div><div>27 Qubits 64 Quantum volume</div></div>	<div><div>ibmq_manhattan</div><div>System status <span>Offline</span></div><div>Processor type Hummingbird r2</div><div>65 Qubits 32 Quantum volume</div></div>	<div><div>ibmq_brooklyn</div><div>System status <span>Online - Queue paused</span></div><div>Processor type Hummingbird r2</div><div>65 Qubits 32 Quantum volume</div></div>
<div><div>ibmq_toronto</div><div>System status <span>Online</span></div><div>Processor type Falcon r4</div><div>27 Qubits 32 Quantum volume</div></div>	<div><div>ibmq_sydney</div><div>System status <span>Online</span></div><div>Processor type Falcon r4</div><div>27 Qubits 32 Quantum volume</div></div>	<div><div>ibmq_guadalupe</div><div>System status <span>Online</span></div><div>Processor type Falcon r4P</div><div>16 Qubits 32 Quantum volume</div></div>	<div><div>ibmq_casablanca</div><div>System status <span>Online</span></div><div>Processor type Falcon r4H</div><div>7 Qubits 32 Quantum volume</div></div>

Credit: [IBM - Quantum Services](#)



IBM Quantum Lab

New file +

Filter files by name

Lab files / ... / qiskit / algorithms /

Name	Last Modified
01_algorithms_introduction....	a month ago
02_vqe_convergence.ipynb	a month ago
03_vqe_simulation_with_no...	a month ago
04_vqe_advanced.ipynb	a month ago
05_qaoa.ipynb	a month ago
06_grover.ipynb	a month ago
07_grover_examples.ipynb	a month ago
08_factorizers.ipynb	a month ago
09_IQPE.ipynb	a month ago
index.rst	a month ago

File Edit View Run Kernel Tabs Settings Help

Launcher x 01\_algorithms\_introducti 07\_grover\_examples.ipyn

Qiskit v0.31.0 (ipykernel)

Boolean Logical Expressions

Qiskit's `Grover` can also be used to perform Quantum Search on an `Oracle` constructed from other means, in addition to DIMACS. For example, the `PhaseOracle` can actually be configured using arbitrary Boolean logical expressions, as demonstrated below.

```
[7]: expression = '(w ^ x) & ~(y ^ z) & (x & y & z)'
try:
    oracle = PhaseOracle(expression)
    problem = AmplificationProblem(oracle, is_good_state=oracle.evaluate_bitstring)
    grover = Grover(quantum_instance=QuantumInstance(Aer.get_backend('aer_simulator'), shots=1024))
    result = grover.amplify(problem)
    display(plot_histogram(result.circuit_results[0]))
except MissingOptionalLibraryError as ex:
    print(ex)
```

8-bit string	Probability
0000	0.045
0001	0.021
0010	0.021
0011	0.039
0100	0.039
0101	0.039
0110	0.039
0111	0.039
1000	0.039
1001	0.039
1010	0.039
1011	0.039
1100	0.039
1101	0.039
1110	0.490
1111	0.039

Simple 0 3 Qiskit v0.31.0 (ipykernel) | Idle Mem: 276.42 / 8192.00 MB

Mode: Command Ln 1, Col 1 07\_grover\_examples.ipynb

Credit: [IBM - Quantum Lab](#)

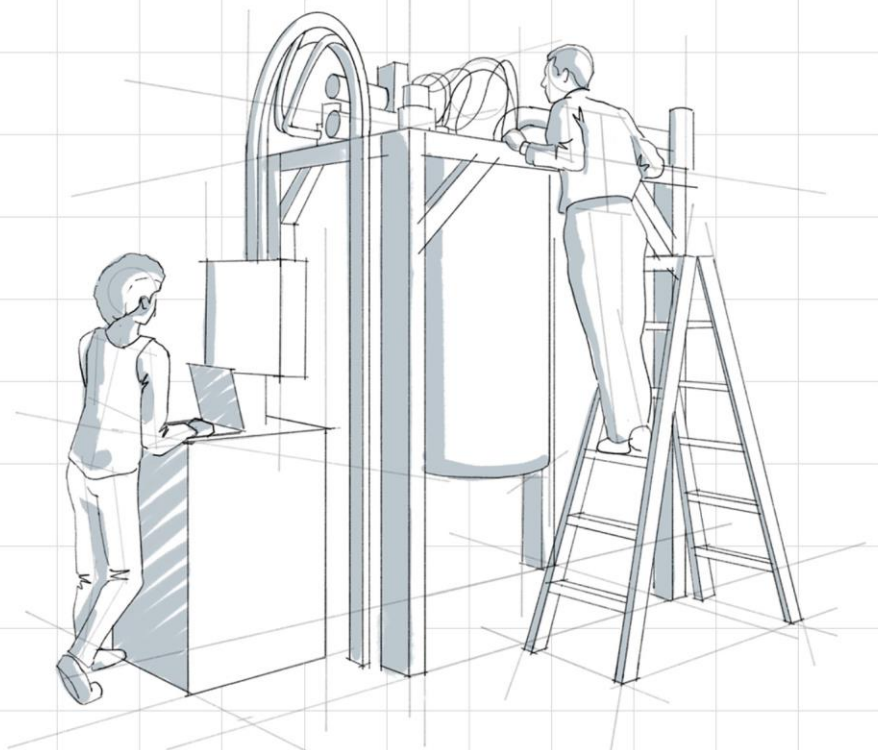
[Overview](#)[Learn](#)[Community](#) [Documentation](#)

qiskit 0.24.0

[see release notes](#)

## Open-Source Quantum Development

Qiskit [kiss-kit] is an open source SDK for working with quantum computers at the level of pulses, circuits and application modules.

[Get started](#)

Credit: [IBM - Qiskit](#)



# Qiskit

[Getting started](#)[Tutorials](#)[Partners](#)[Applications](#)[Experiments](#)[Resources](#)[Github](#)

English

[Docs](#) > [Qiskit 0.31.0 documentation](#)



Search Docs

[Documentation Homepage](#)

#### Frontmatter

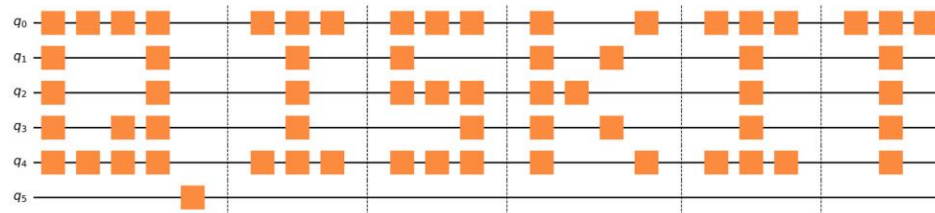
[Quantum computing in a nutshell](#)  
[Introduction to Qiskit](#)  
[Release Notes](#)  
[Contributing to Qiskit](#)  
[Local Configuration](#)  
[Frequently Asked Questions](#)

#### Libraries

[Circuit Library](#)

#### API References

[Qiskit Terra](#)  
[Qiskit Aer](#)  
[Qiskit Ignis \(deprecated\)](#)  
[Qiskit Aqua \(deprecated\)](#)  
[Qiskit IBM Quantum Provider](#)



#### Qiskit 0.31.0 documentation

Interested in applications of quantum computing?  
Interested in running experiments on real quantum hardware?  
Interested in quantum hardware design?

## 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.

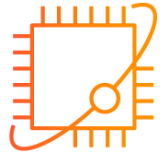
[Get cracking](#)

### Access to quantum systems

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

[Qiskit Partners](#)

Credit: [IBM - Qiskit documentation](#)



### Amazon Braket

Get started with quantum computing



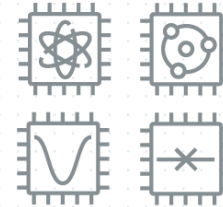
### Build

Build your quantum algorithms on managed Jupyter notebooks or in your own development environment



### Test

Test your algorithms on a local simulator or a choice of fully managed, high-performance simulators



### Run

Run your algorithms on your choice of different quantum computers. Combine classical and quantum computing resources for hybrid algorithms



### Analyze

Analyze results after your algorithm has completed

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 »](#)

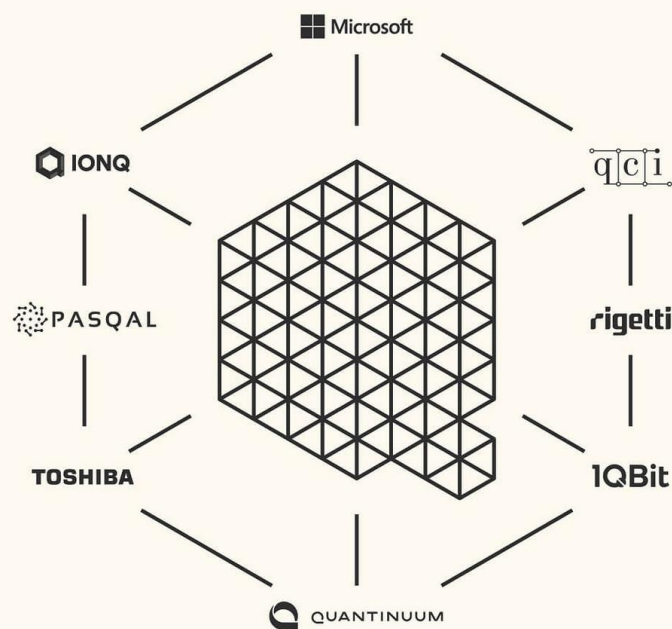
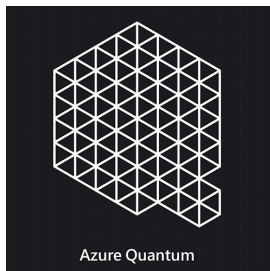


Rigetti quantum processors are universal, gate-based machines based on superconducting qubits. The Rigetti Aspen series of chips feature tileable lattices of alternating fixed-frequency and tunable superconducting qubits within a scalable architecture.

[Learn more »](#)

Credit: [Amazon AWS - Braket](#)





## 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](#)





# 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](#)

```
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 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](#)

# D-Wave

The screenshot displays the Leap IDE interface. On the left, a Python script named `maximum_cut.py` is shown, which creates a graph, adds edges, and solves a QUBO problem using the D-Wave sampler. The script includes comments and a loop to update the Q matrix for each edge. The bottom of the script shows the execution of the sampler and the display of the response.

On the right, the **PROBLEM INSPECTOR** panel is visible. It has two tabs: **Source - Force Directed** and **Target - QPU**. The **Source** view shows a graph with 5 nodes and 8 edges. The **Target** view shows the QPU architecture with nodes and connections. Below these views are legends for **QUBO** and **Ising** representations.

At the bottom, the **Console** panel shows the output of the script, including a table of results for Set 0 and Set 1.

Set 0	Set 1	Energy	Cut Size
[1, 4, 5]	[2, 3]	-5.0	5
[2, 3, 5]	[1, 4]	-5.0	5
[1, 4]	[2, 3, 5]	-5.0	5
[2, 3]	[1, 4, 5]	-5.0	5
[4, 5]	[1, 2, 3]	-3.0	3
[3]	[1, 2, 4, 5]	-3.0	3
[1, 3, 5]	[2, 4]	-3.0	3
[1, 3]	[2, 4, 5]	-3.0	3

The console also shows the message: "Your plot is saved to maxcut\_plot.png".

Credit: [D-Wave - Leap](#)