

# Quantum Computing (QUC)



UNIVERSITY OF  
**OXFORD**

# Your Lecturer



Stefano Gogioso

Faculty in the



Oxford  
Quantum  
Group

- Quantum Information
- Quantum Causality
- Foundations of Physics
- Quantum Software

External business interests:  
crypto & decentralised finance



# Your TA



Nicola Pinzani

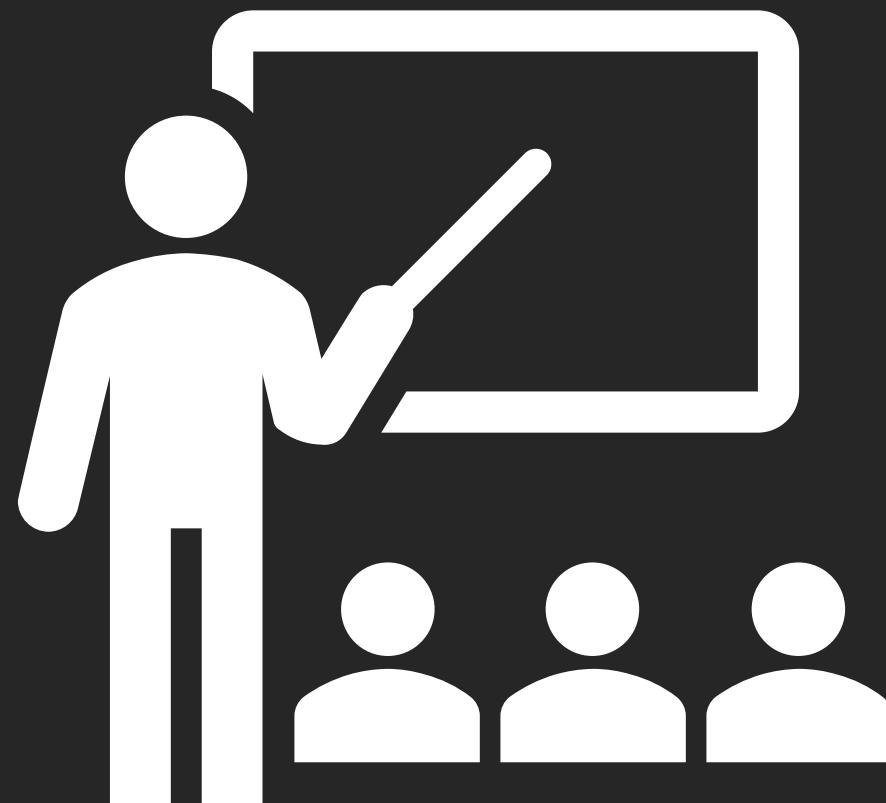
Student in the



Oxford  
Quantum  
Group

- Quantum Information
- Quantum Causality
- Foundations and Philosophy of Physics

# Tell us a little about you!



# Plan for this course

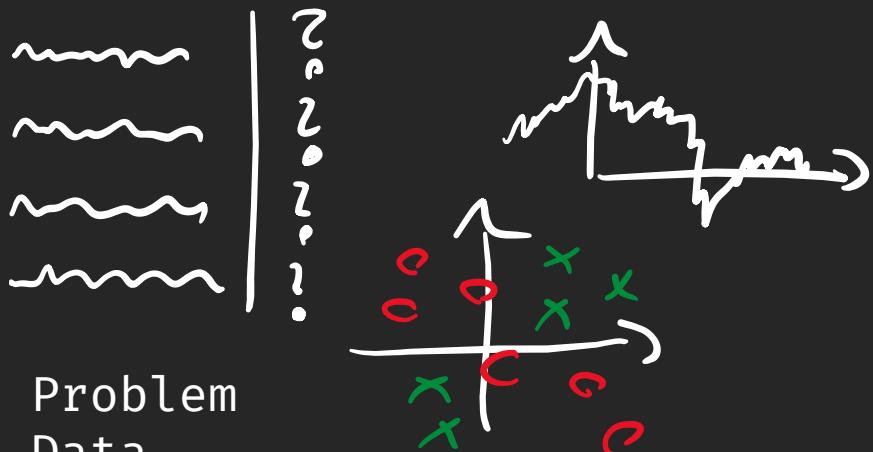
- Lectures Mon-Fri 9am-12:15pm
- Exercises Mon-Thu 2pm-5pm

# What is Quantum Computing?

- Quantum Computers as black-box devices used inside classical algorithms
- Quantum Computers as a programmable interface to the quantum world

# What is Quantum Computing?

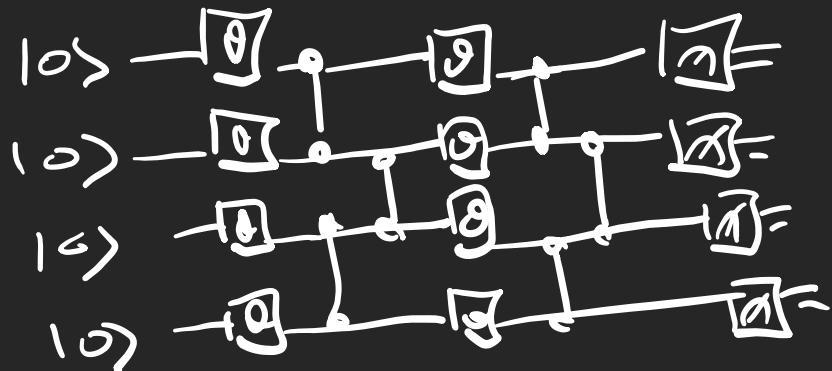
- Quantum Computers as black-box devices used inside classical algorithms
  - Breaking your Bitcoin wallet (one day)
  - Combinatorial Optimization
  - Solutions of Linear/non-linear Systems
  - Complex Physics Simulations (e.g. fluid dynamics)
  - Machine Learning
- Quantum Computers as a programmable interface to the quantum world



Iterate (?)

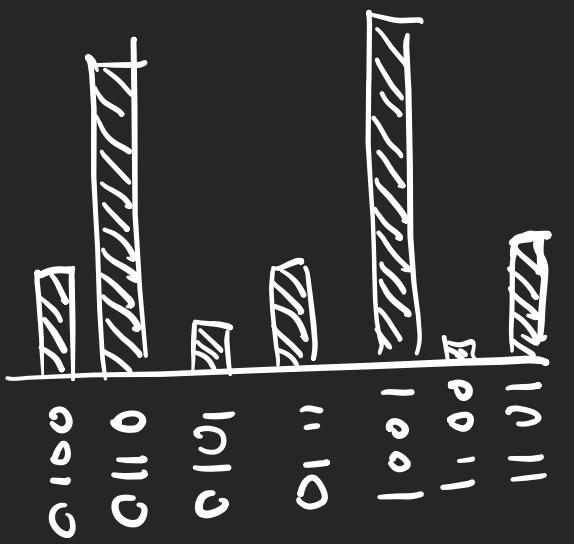


Pre-processing  
and/or encoding



Quantum Circuit(s)

Quantum Processing Unit

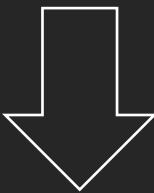
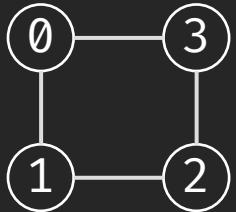


Post-processing  
and/or decoding

Measurement Samples

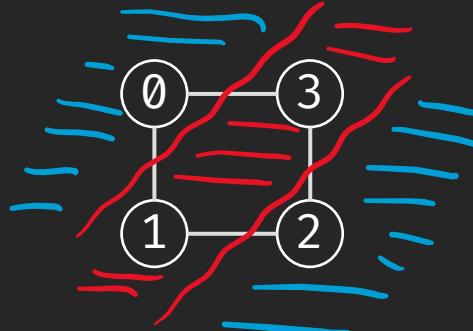
# Example: max-cut with QAOA

Graph



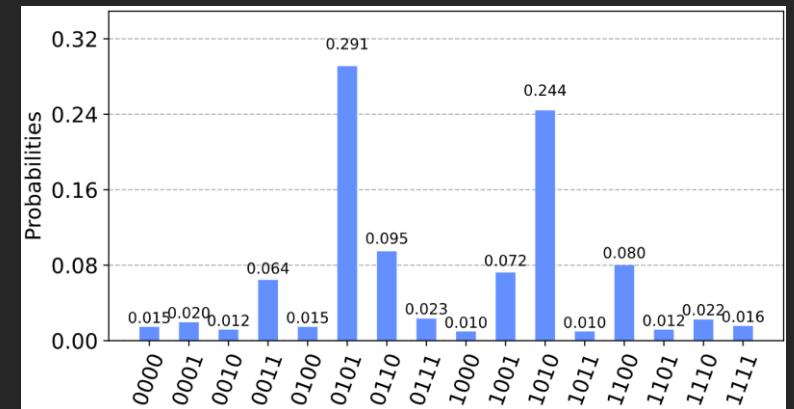
Encoding using  
2-qubit gates

Graph Cut

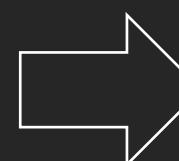
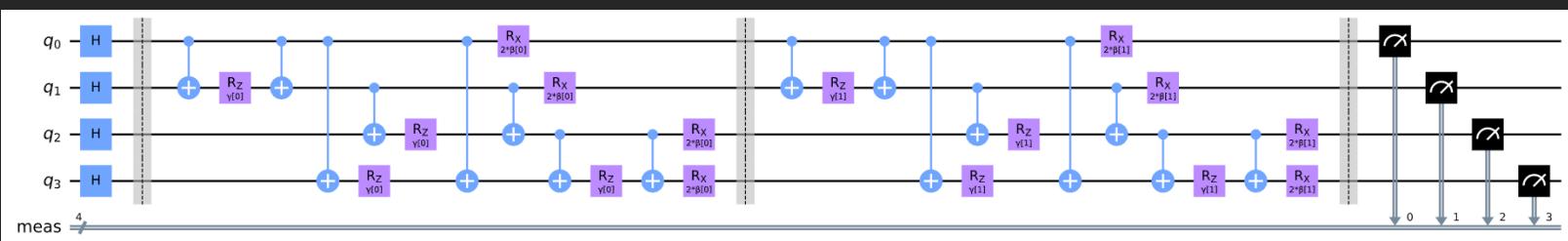


Max count  
solution

Samples



Parameter  
optimisation



ibmq\_santiago  
System status: Online  
Processor type: Falcon r4  
5 Qubits 32 Quantum volume

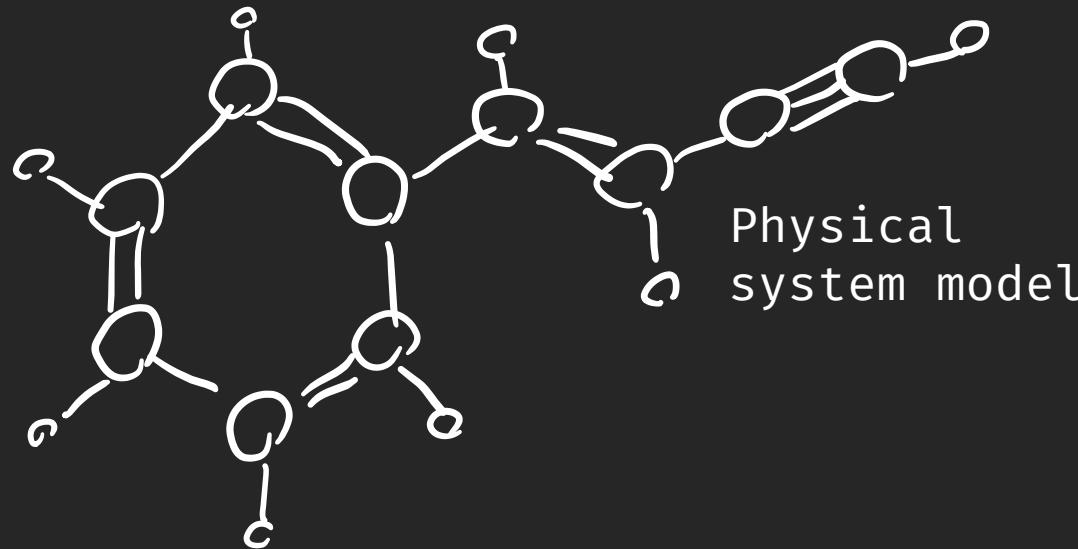


QPU

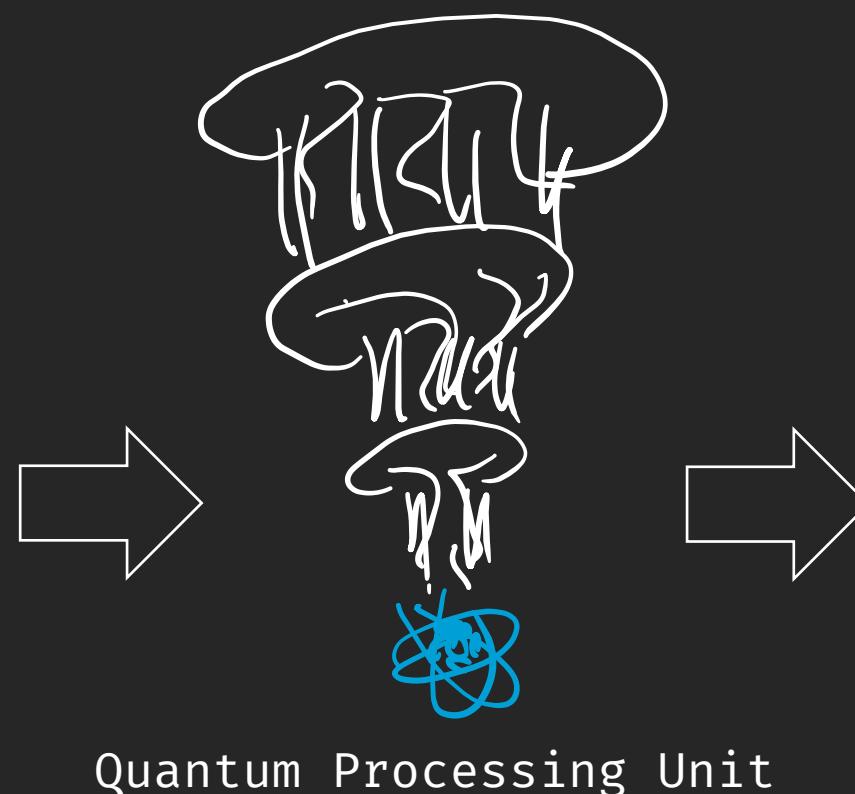
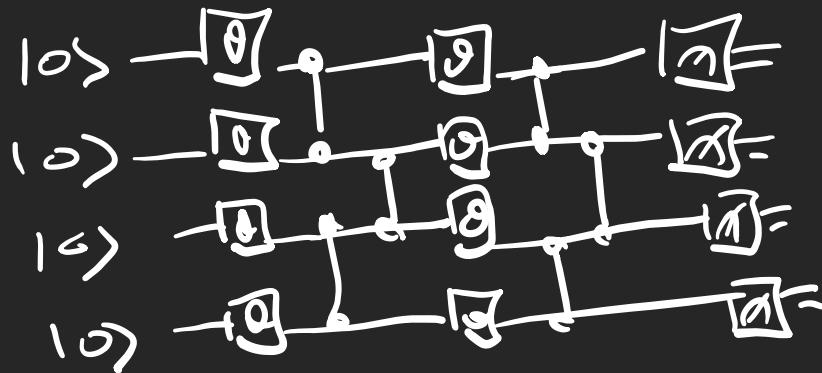
QAOA Circuit (parametric)

# What is Quantum Computing?

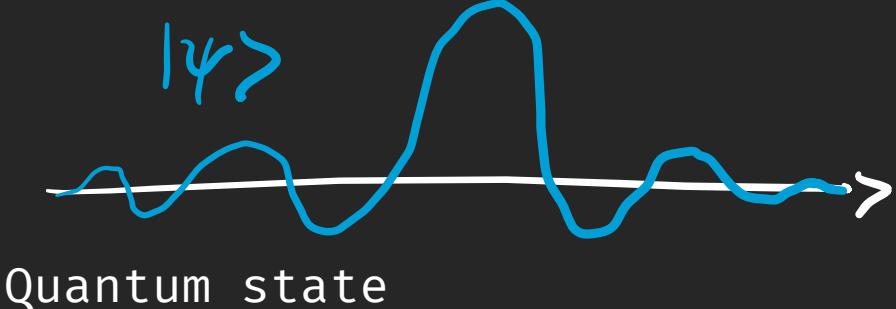
- Quantum Computers as black-box devices used inside classical algorithms
- Quantum Computers as a programmable interface to the quantum world
  - Quantum Chemistry Simulations
  - Quantum Physics Simulations
  - Quantum Information/Communication/Cryptography
  - Quantum Metrology
  - Programmable Quantum Experiments



Encoding



$$\psi^{(\alpha)}(x, t) = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \exp\left(-\frac{m\omega}{2\hbar}\left(x - \sqrt{\frac{2\hbar}{m\omega}}\Re[\alpha(t)]\right)^2 + i\sqrt{\frac{2m\omega}{\hbar}}\Im[\alpha(t)]x + i\theta(t)\right)$$

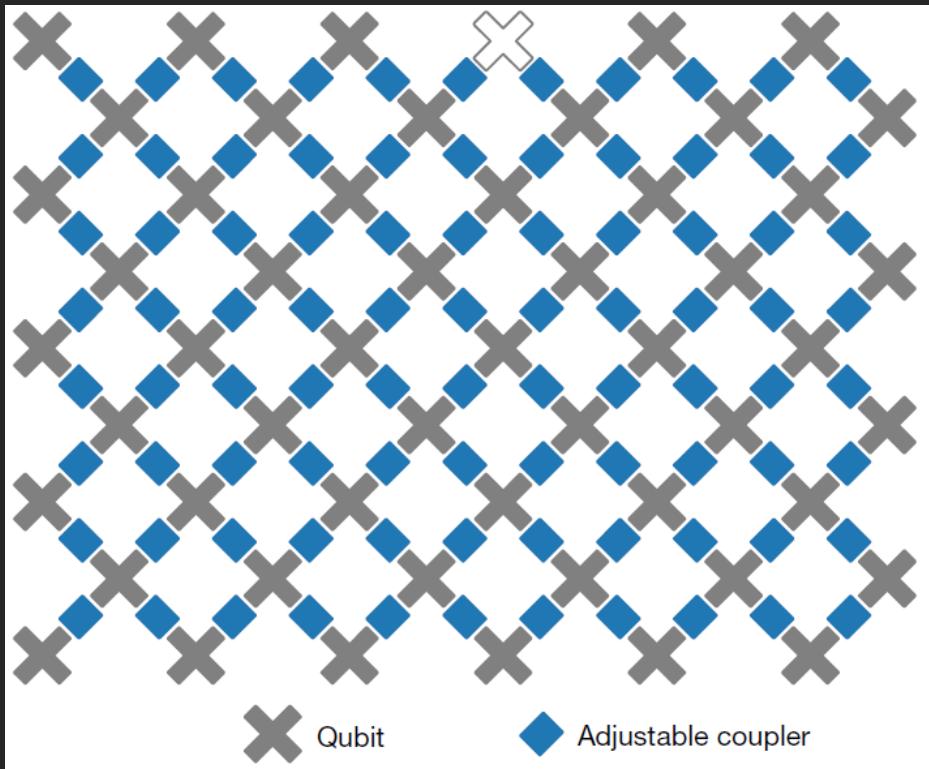


Tomography

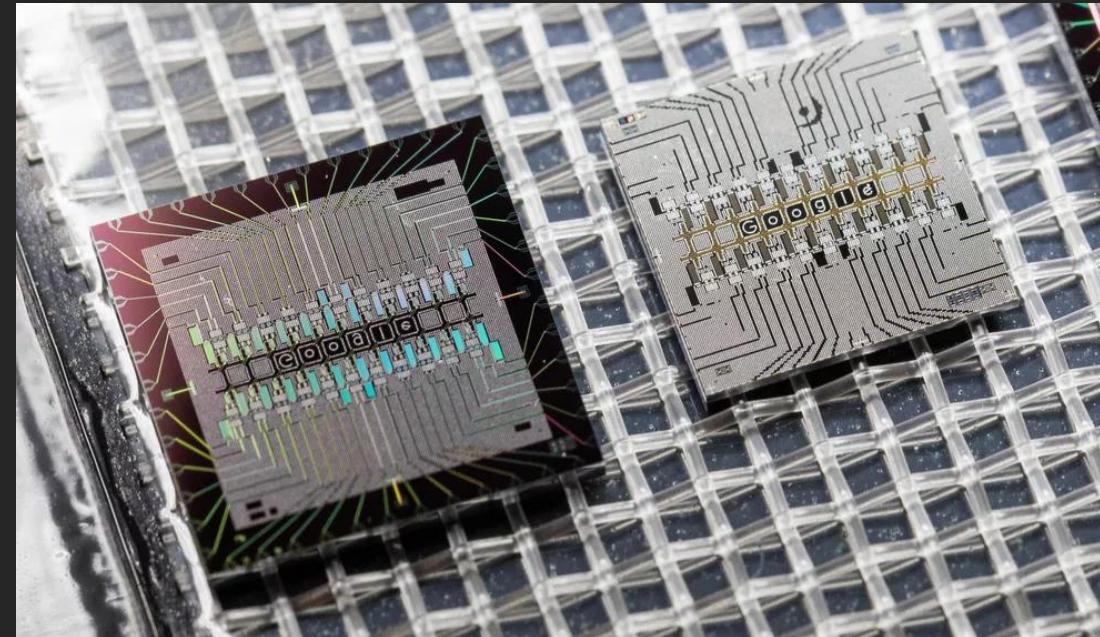


# Hardware Providers

# Google Sycamore (54 qubits)

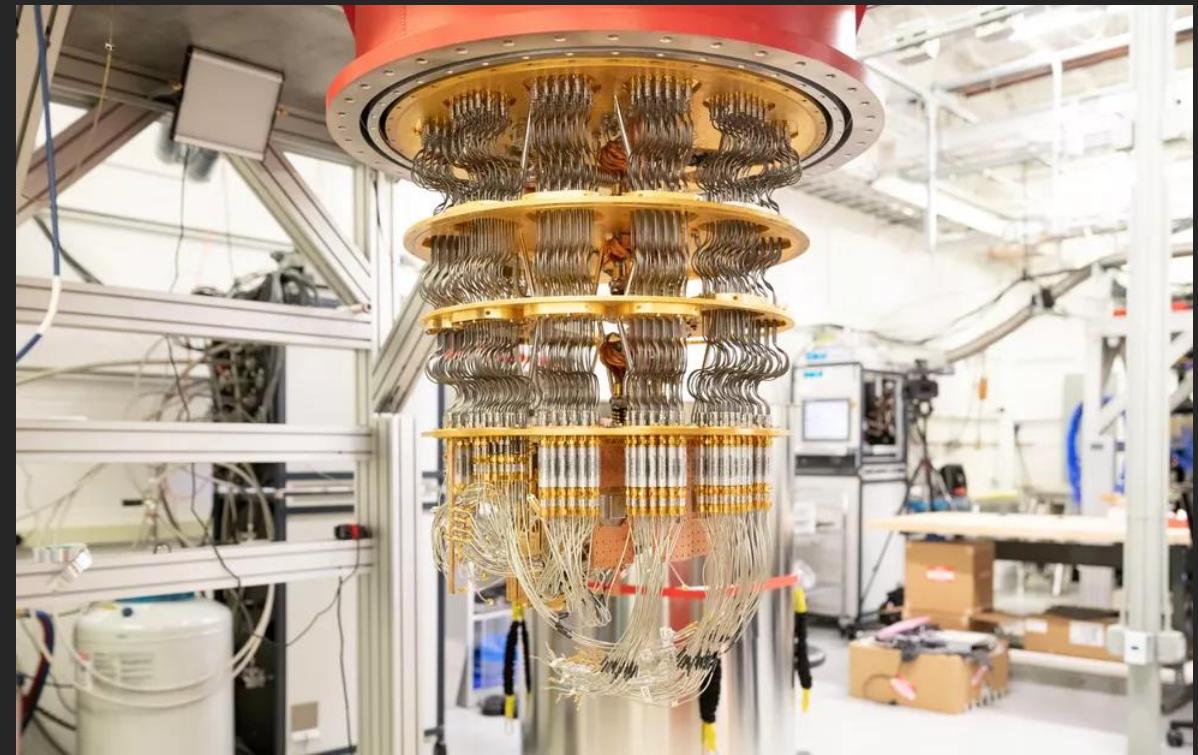
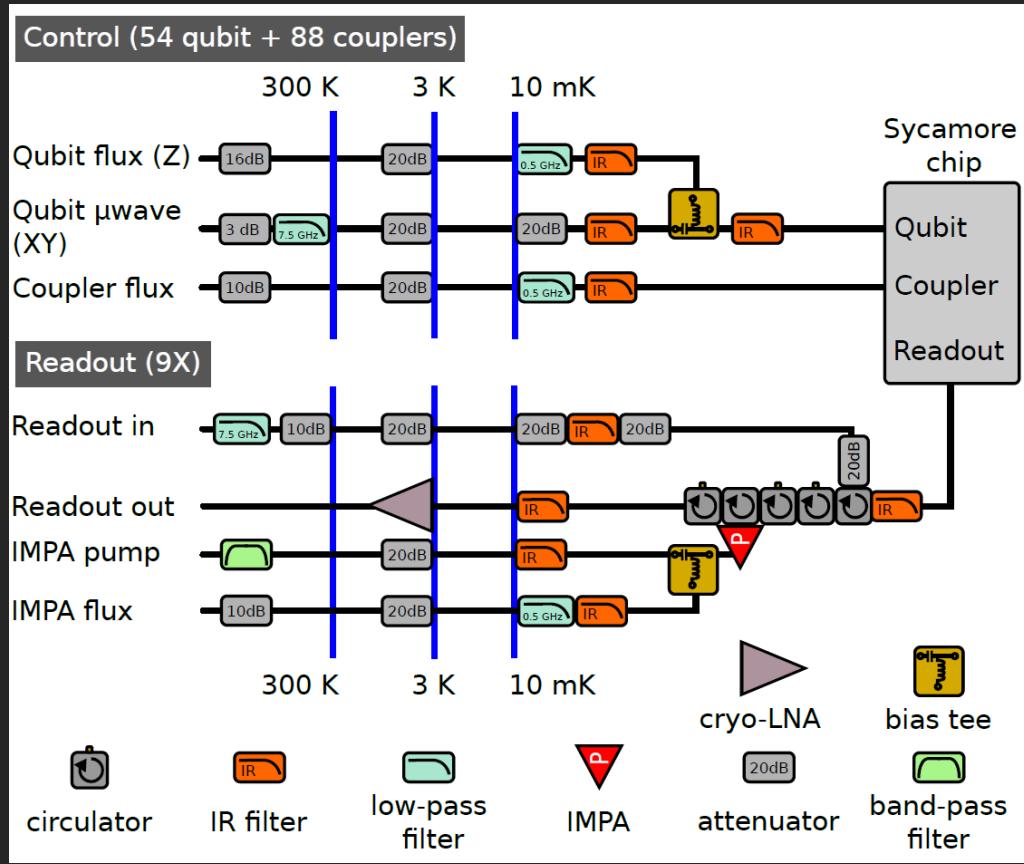


Credit: Nature/Google  
<https://doi.org/10.1038/s41586-019-1666-5>



Credit: Stephen Shankland/CNET

# Google Sycamore (54 qubits)

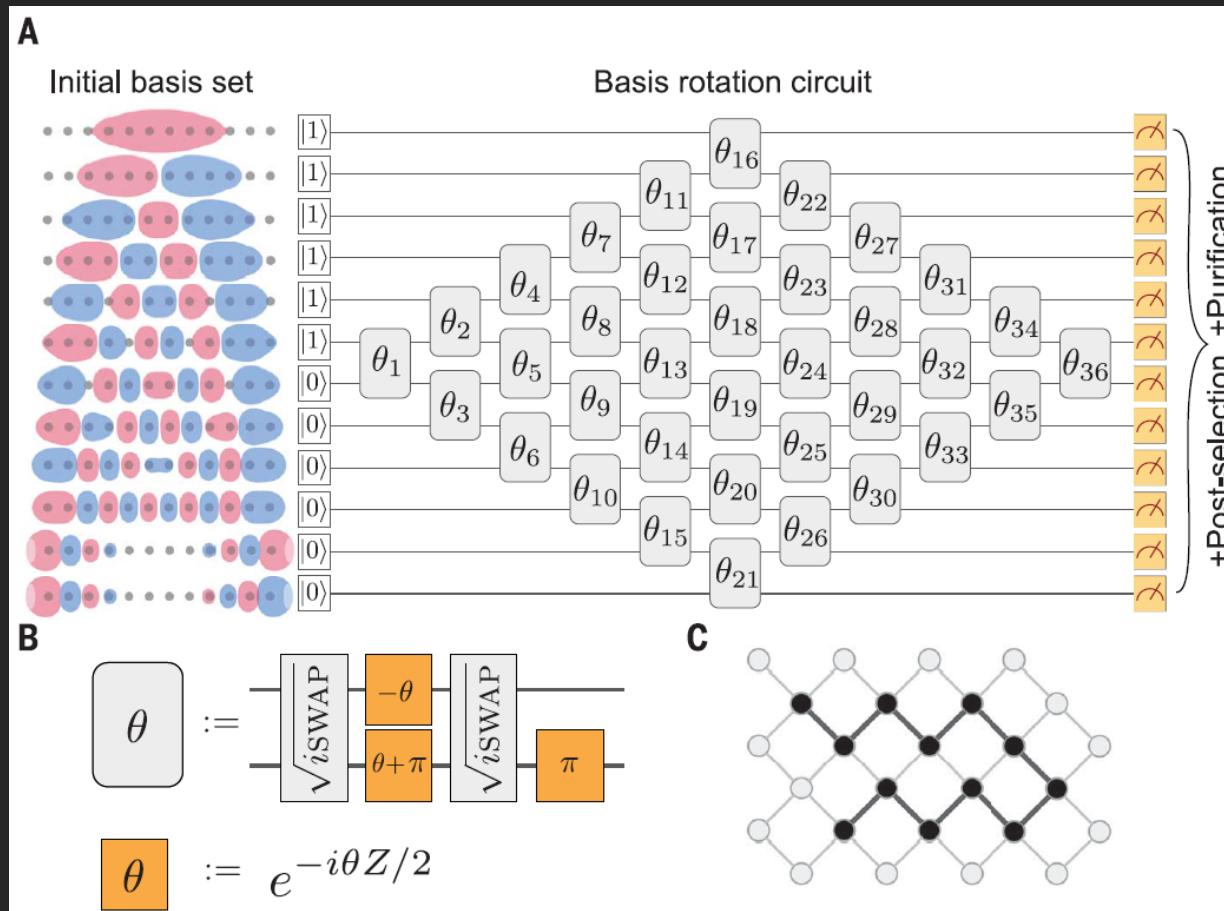


Credit: Stephen Shankland/CNET

Credit: Nature/Google

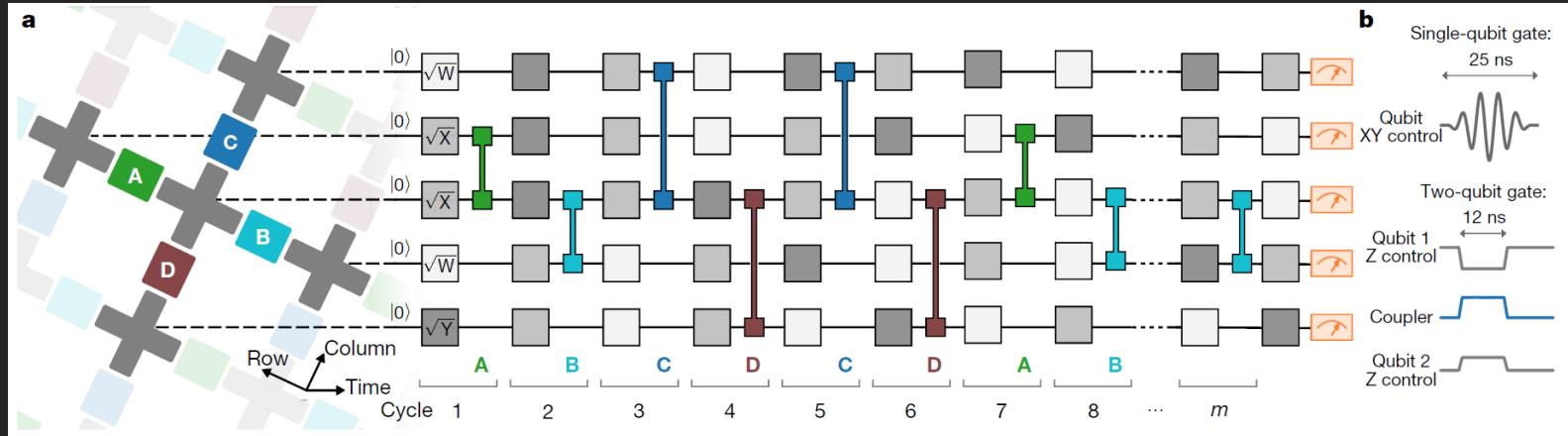
<https://doi.org/10.1038/s41586-019-1666-5>

# Quantum Chemistry Simulations



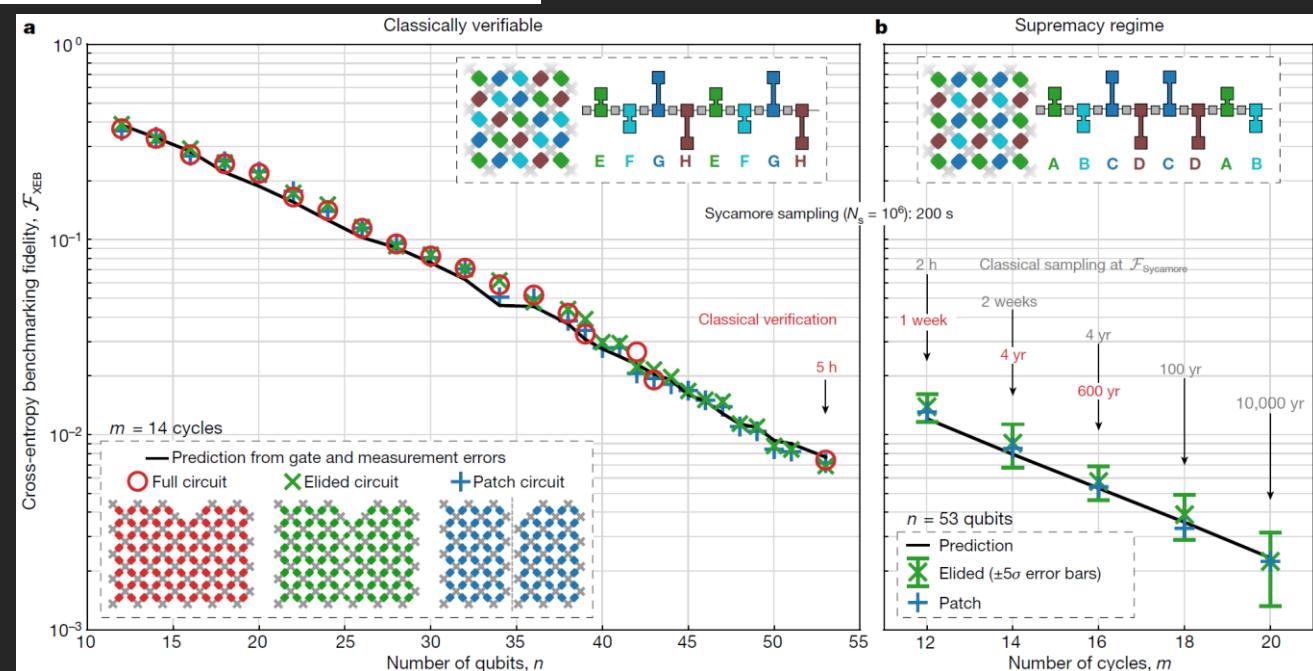
Credit: Science/Google  
<https://doi.org/10.1126/science.abb9811>

# Randomized Benchmarking (XEB)

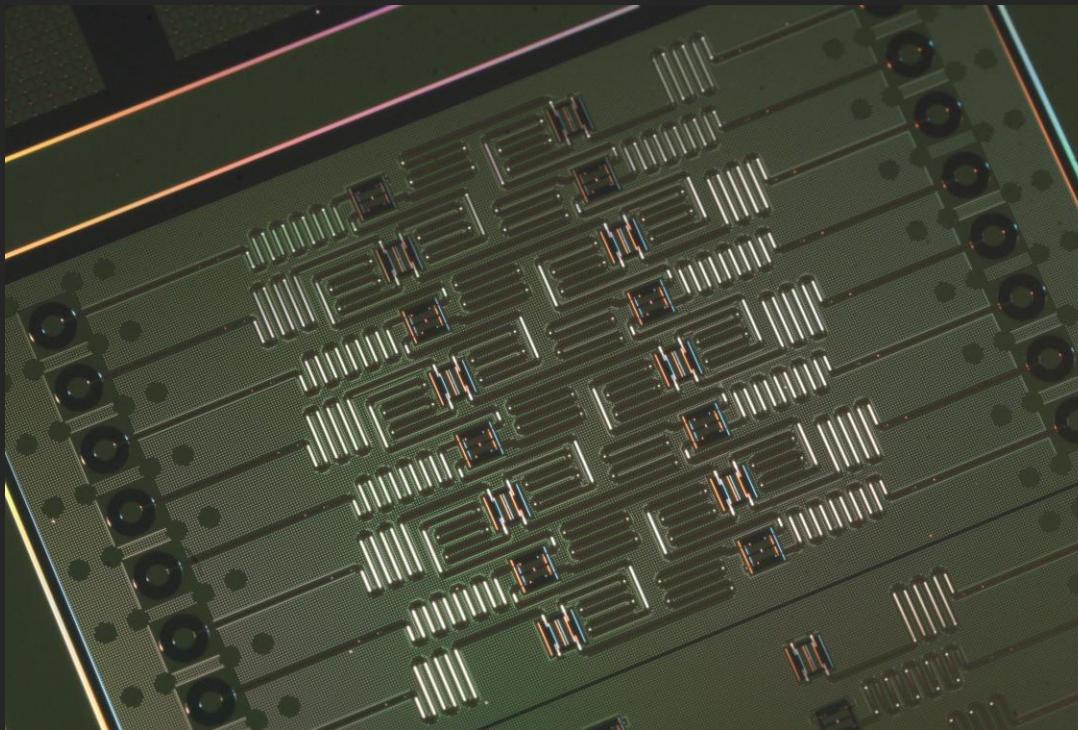


Credit: Nature/Google

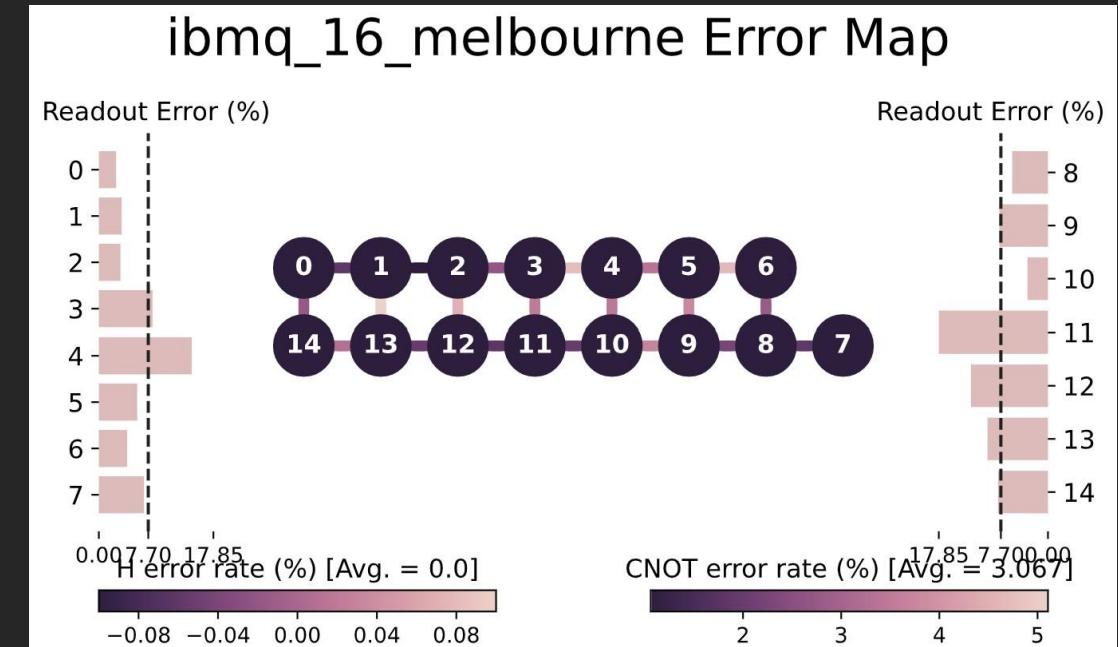
<https://doi.org/10.1038/s41586-019-1666-5>



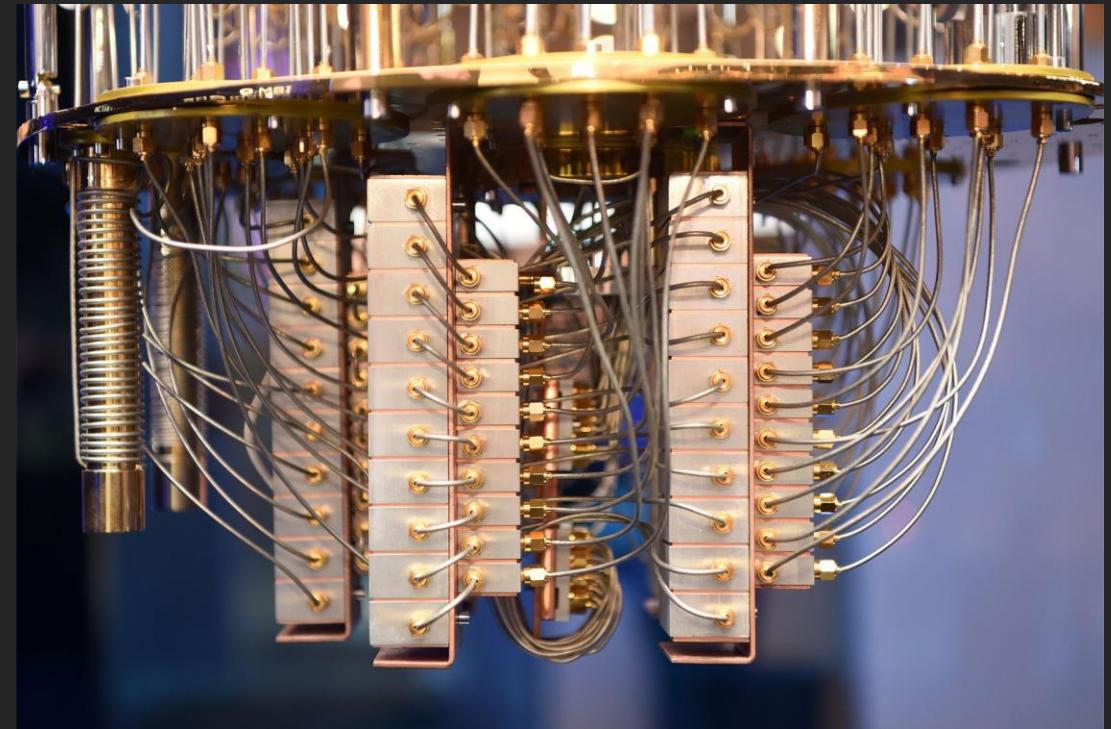
# IBMQ Melbourne (16 qubits)



Credit: IBM

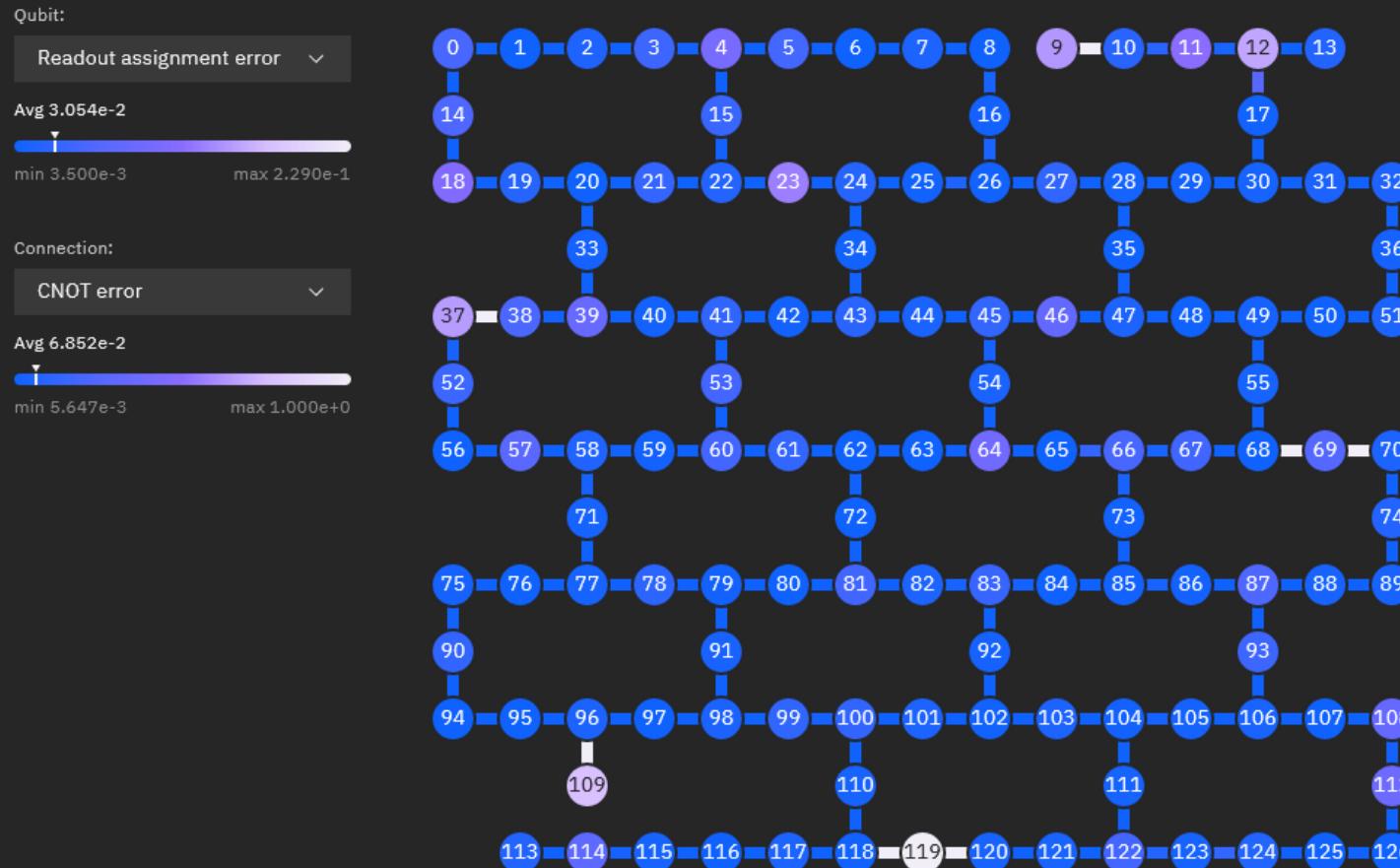


# IBMQ System One (20 qubits)

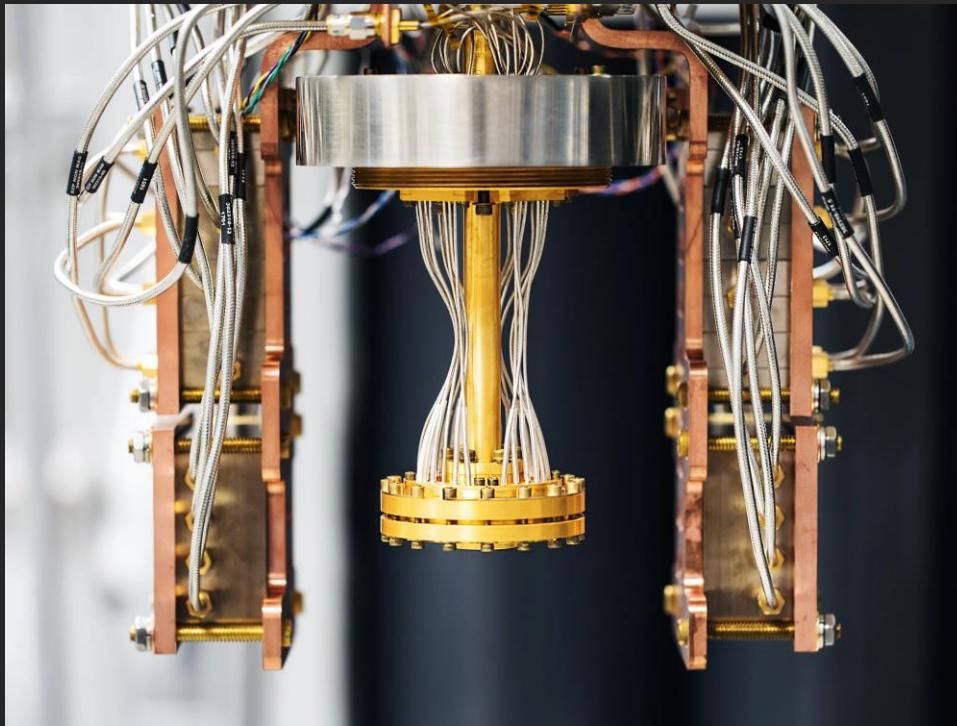


Credit: IBM

# IBM Washington (127 qubits)

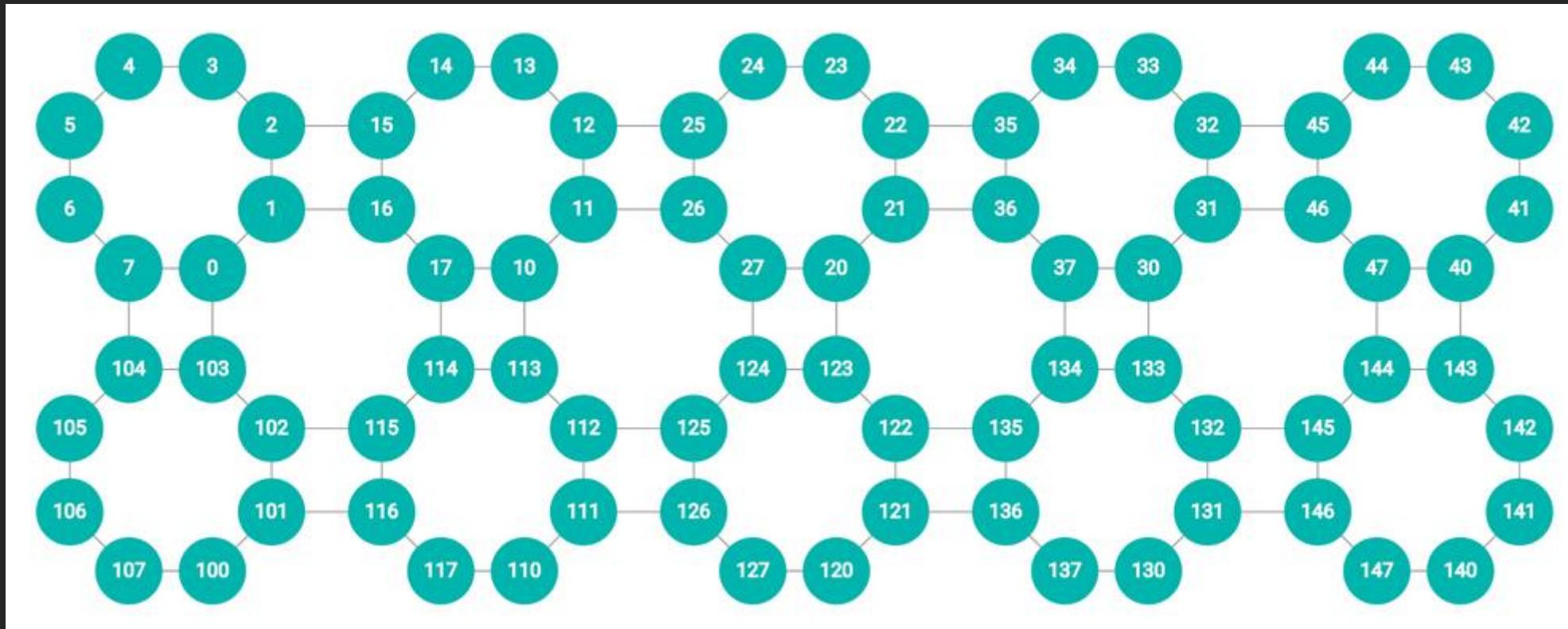


# Rigetti Acorn (19 qubits)



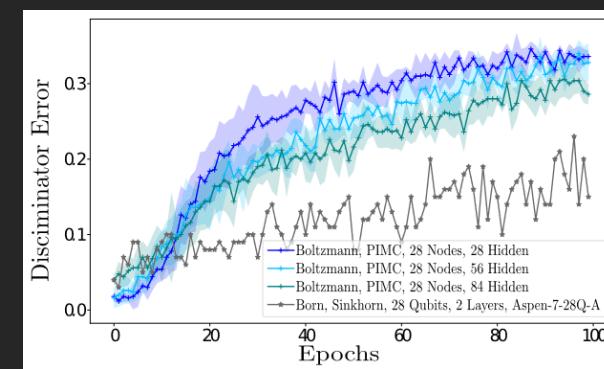
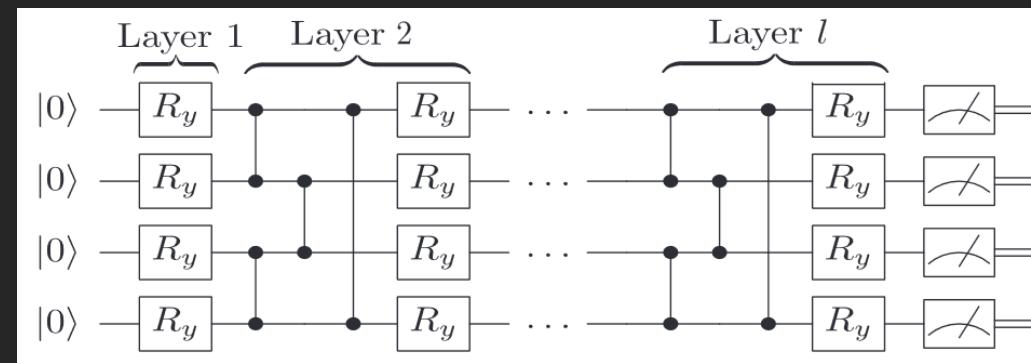
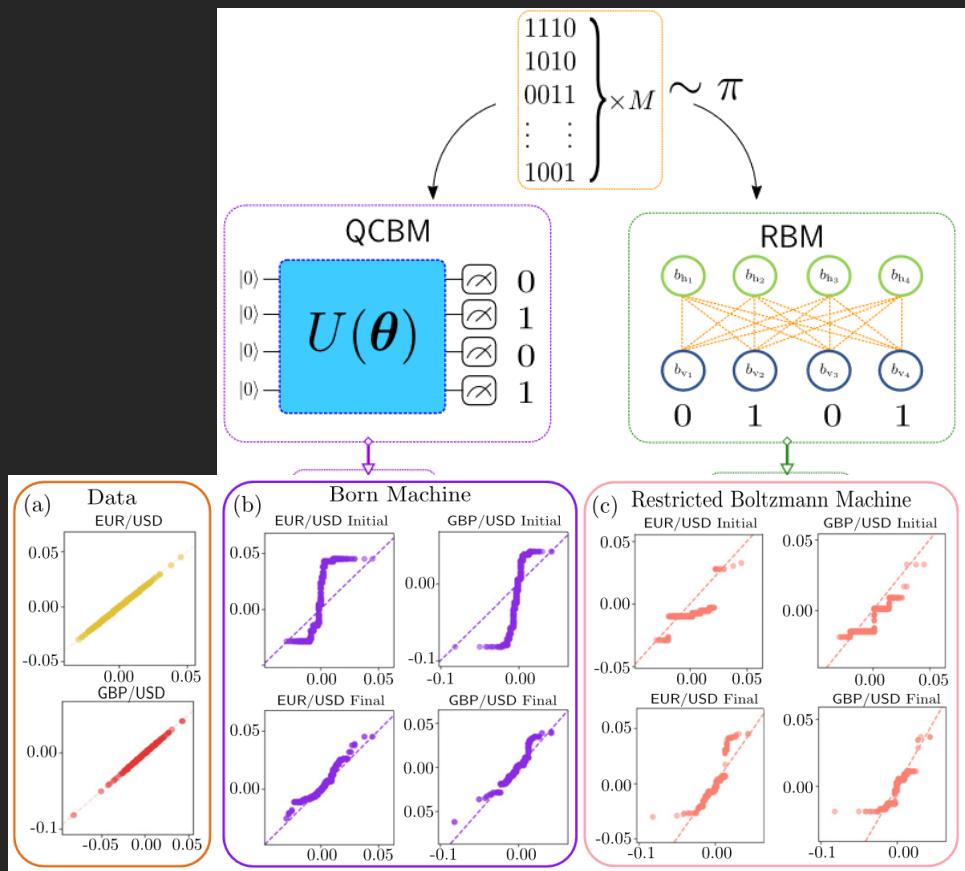
Credit: Rigetti

# Rigetti Aspen-M (80 qubits)



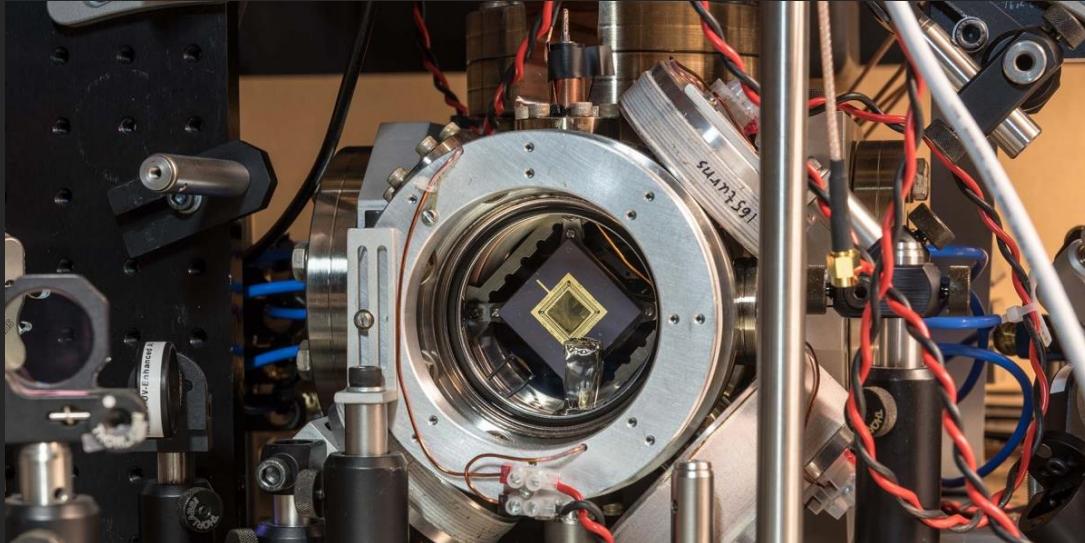
Credit: Amazon AWS/Rigetti

# Generative Financial Modelling

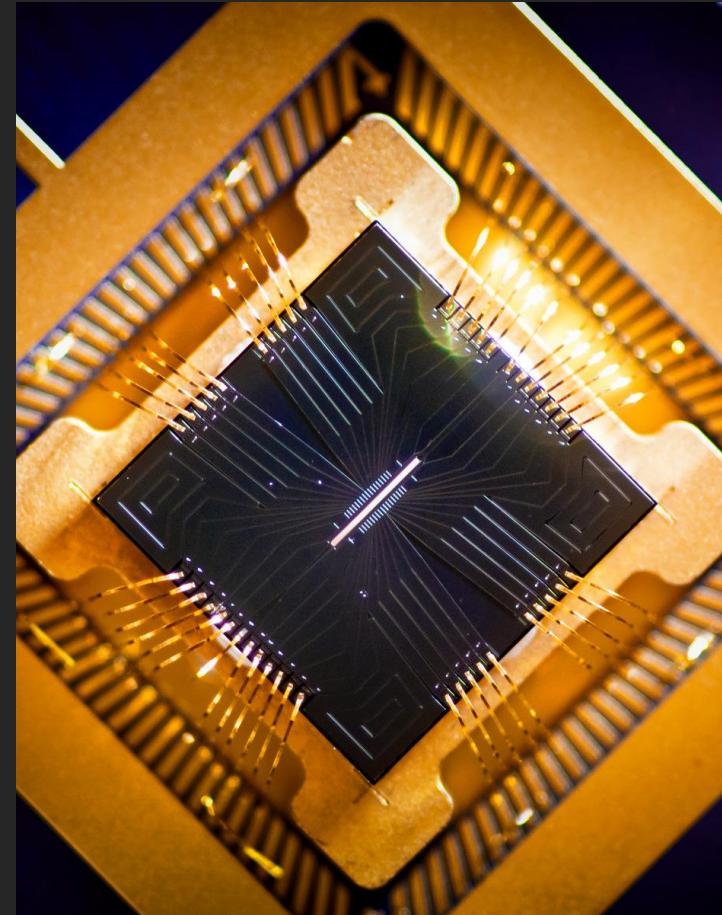


Credit: IOP/various institutions (on Rigetti HW)  
<https://doi.org/10.1088/2058-9565/abd3db>

# Ion-trap Quantum Computers

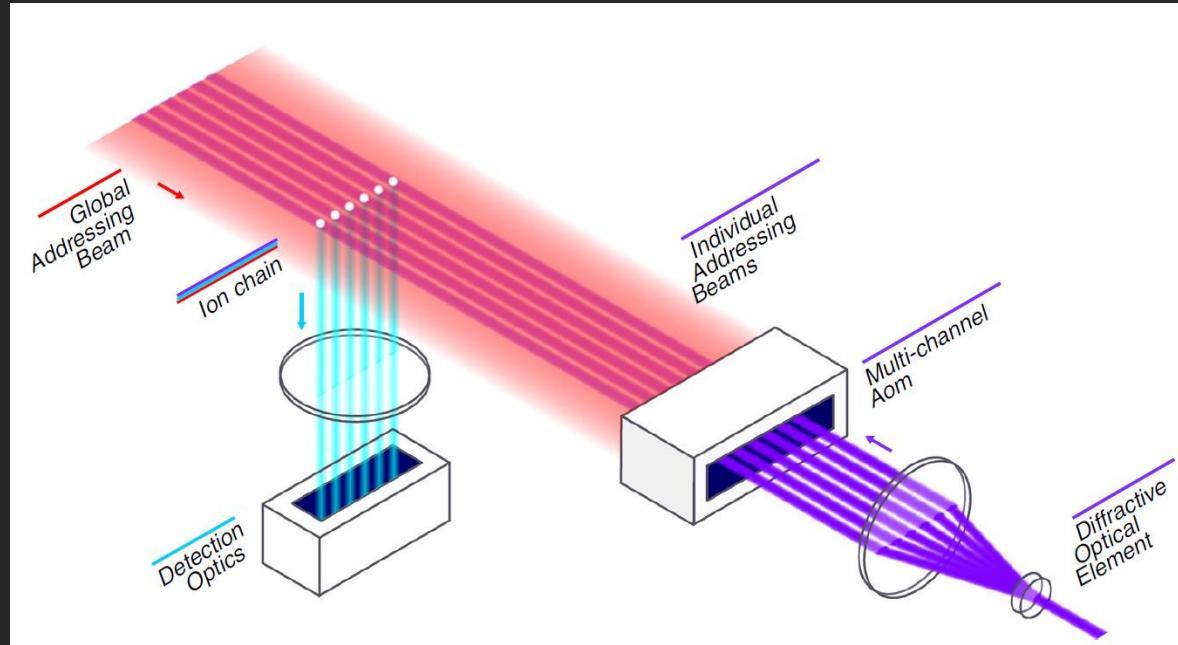


Credit: NQIT/Stuart Bebb



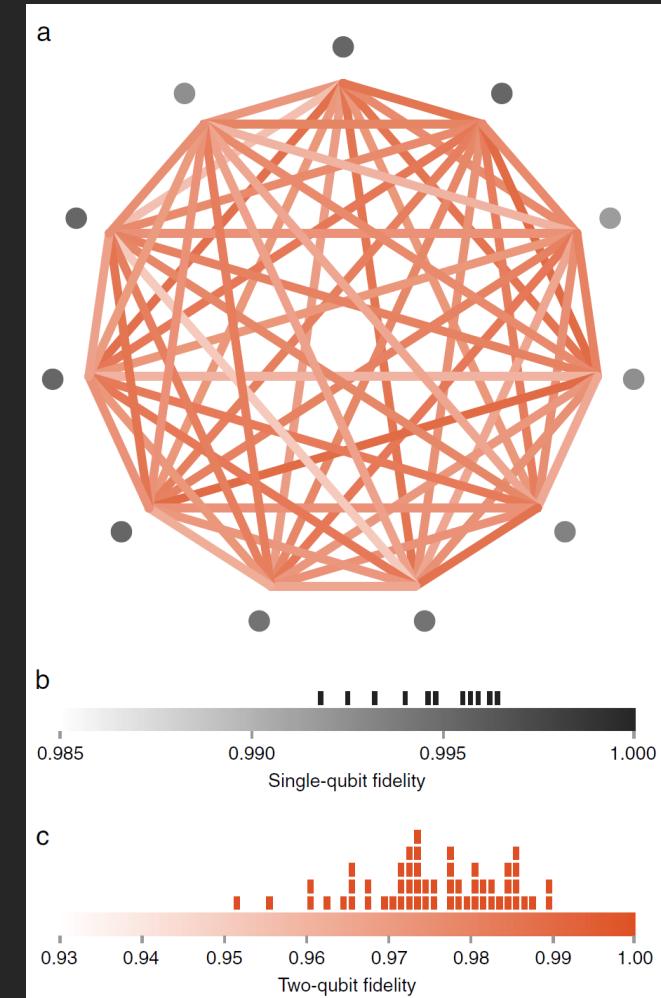
Credit: JQI

# IonQ 171Yb+ (11 qubits)

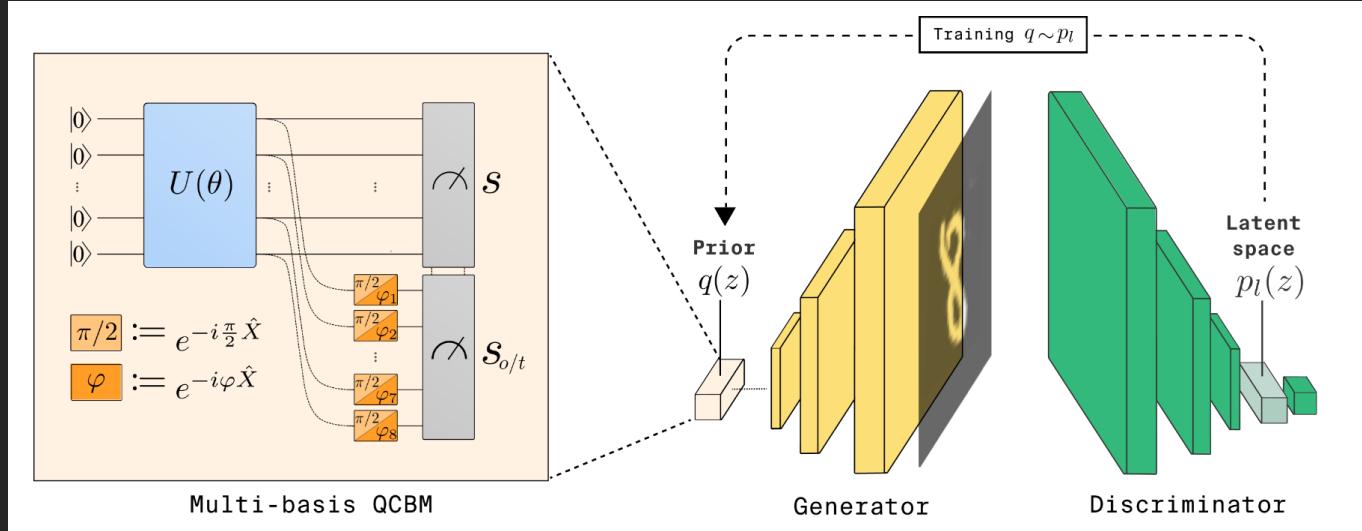


Credit: Nature/IonQ

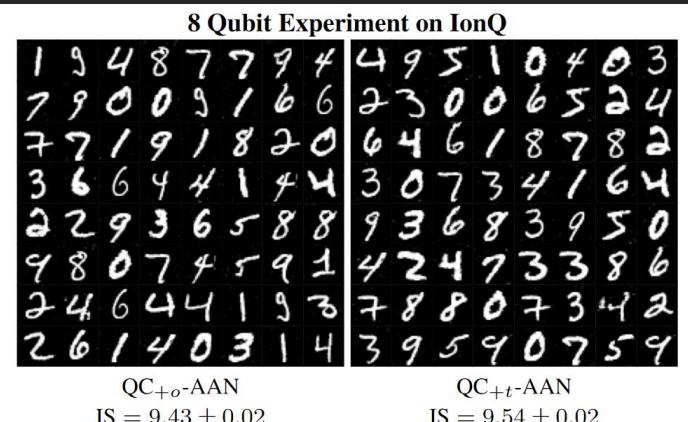
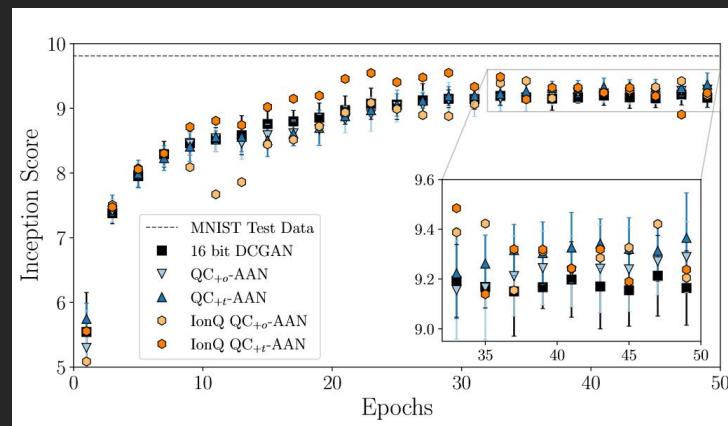
<https://doi.org/10.1038/s41467-019-13534-2>



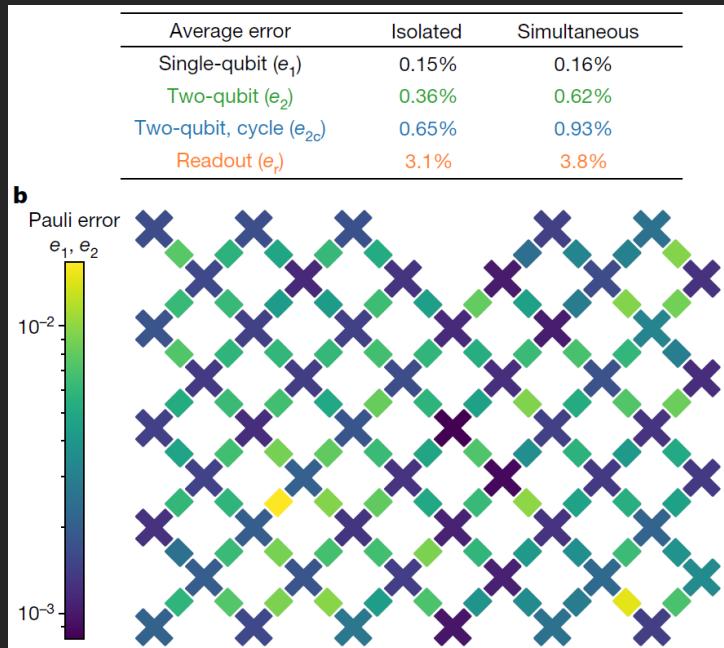
# Quantum Generative Adversarial Networks



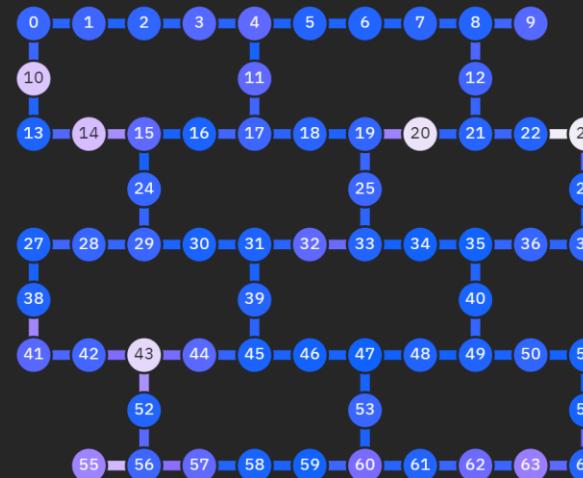
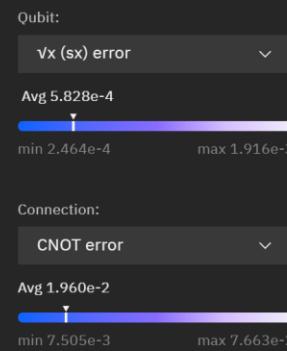
Credit: Zapata Computing/IonQ  
<https://arxiv.org/abs/2012.03924>



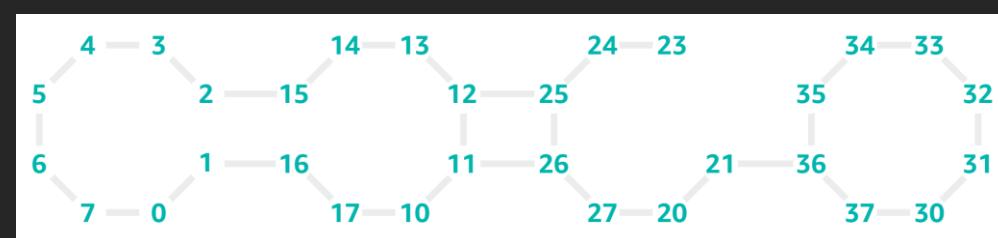
# Superconducting vs Ion-trap



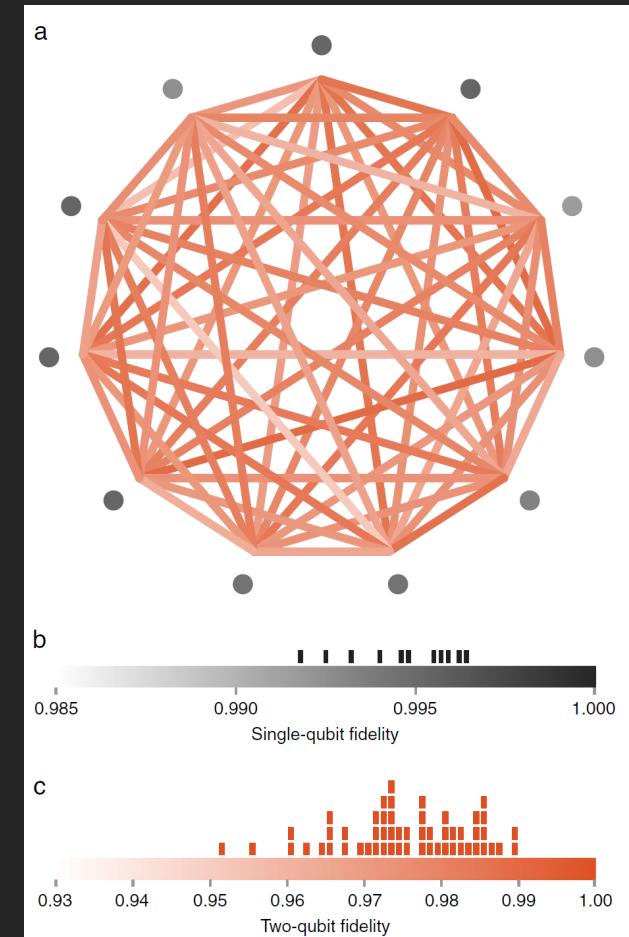
Credit: Nature/Google  
<https://doi.org/10.1038/s41586-019-1666-5>



Credit: IBMQ

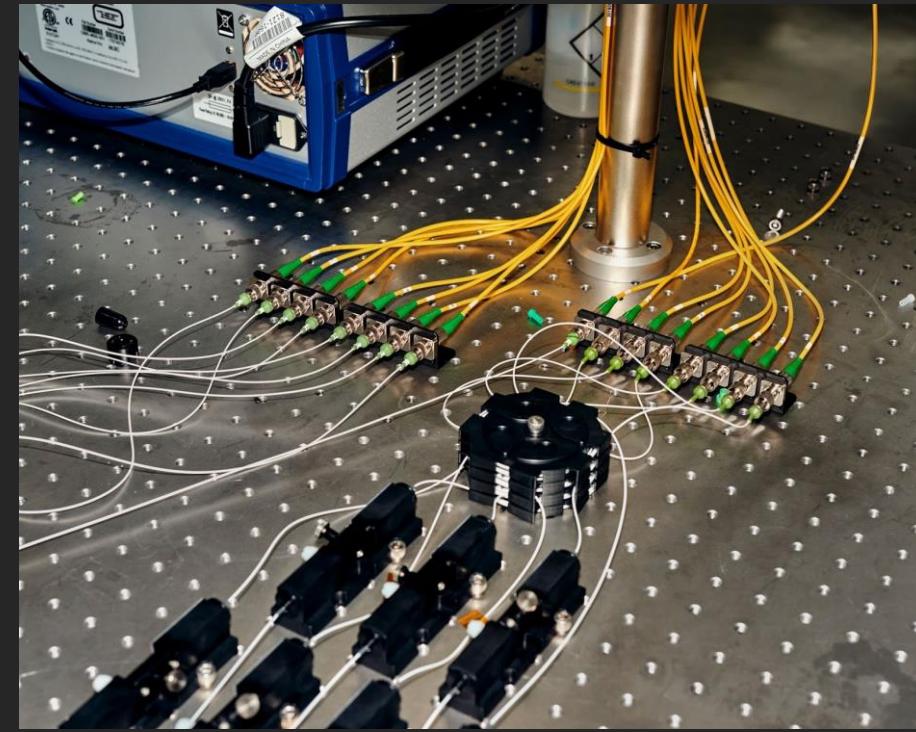
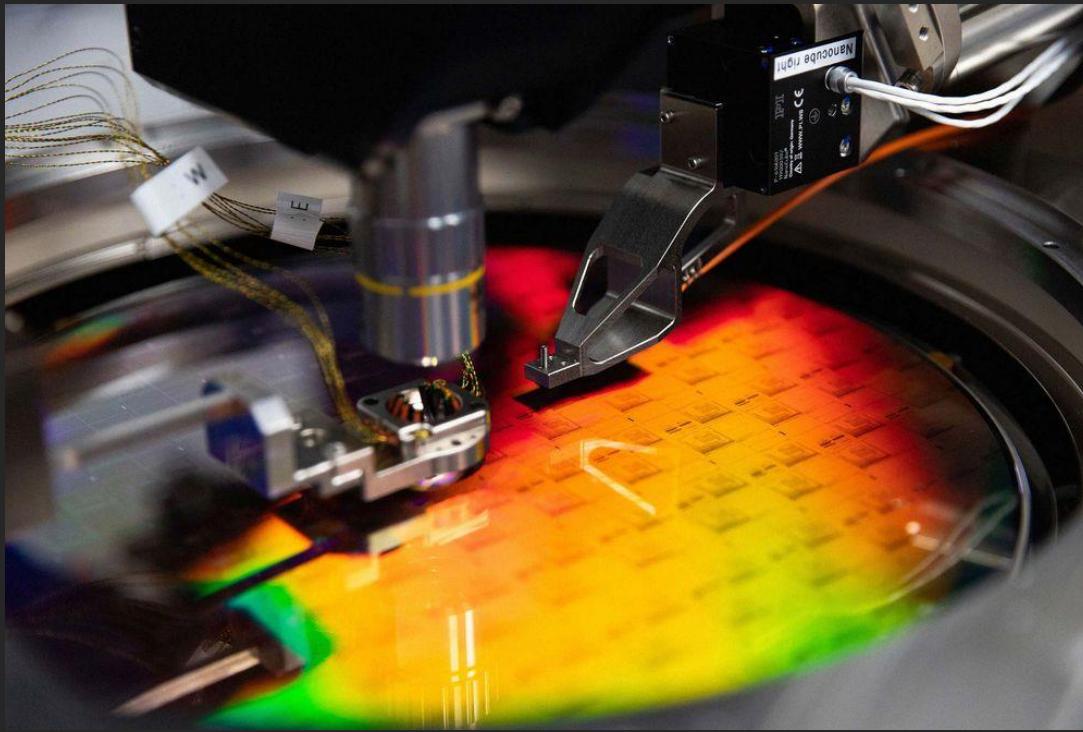


Credit: Amazon AWS/Rigetti



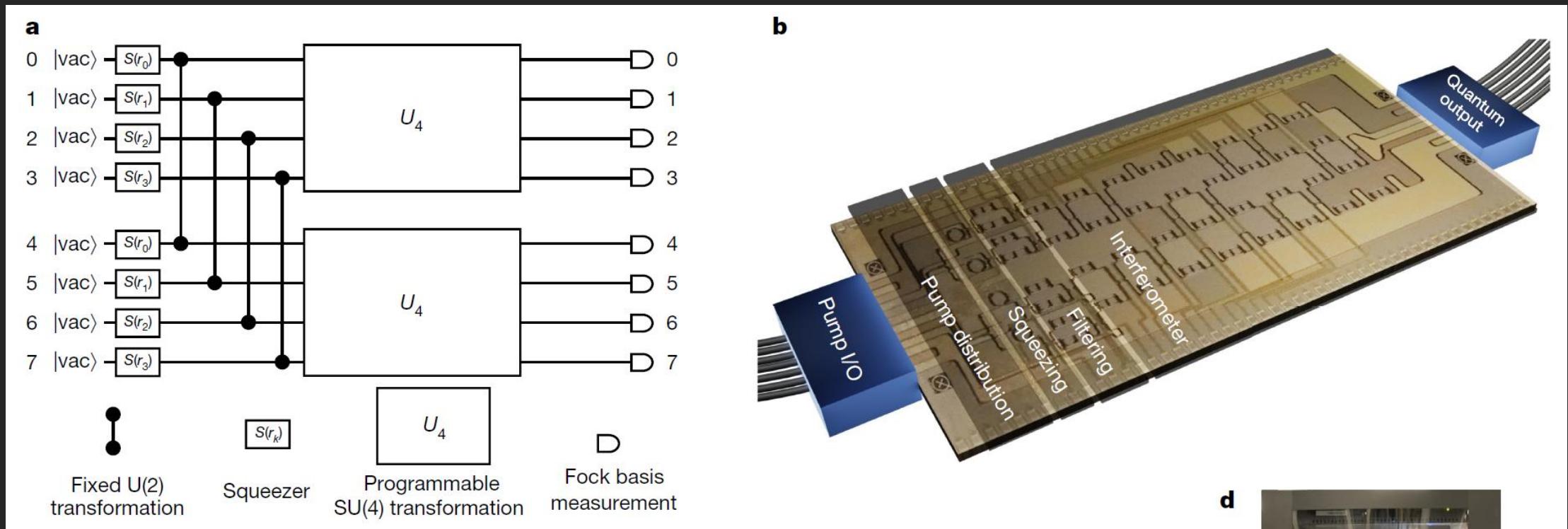
Credit: Nature/IonQ  
<https://doi.org/10.1038/s41467-019-13534-2>

# Photonic Quantum Computers



Credit: PsiQuantum

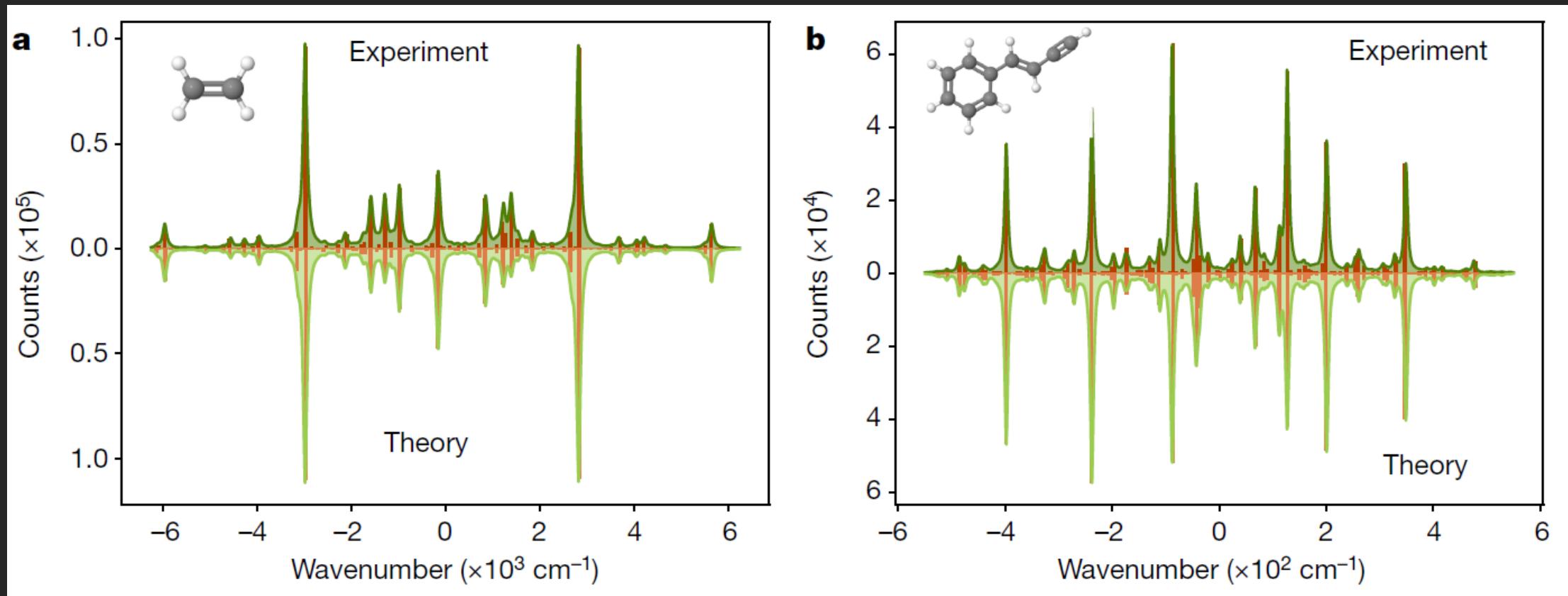
# Xanadu 8-mode photonic chip



Credit: Nature/Xanadu

<https://doi.org/10.1038/s41586-021-03202-1>

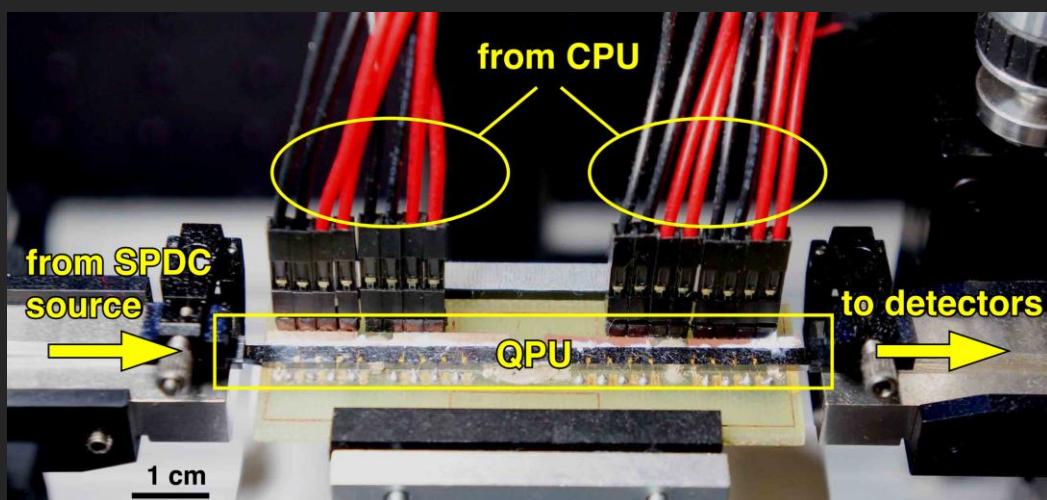
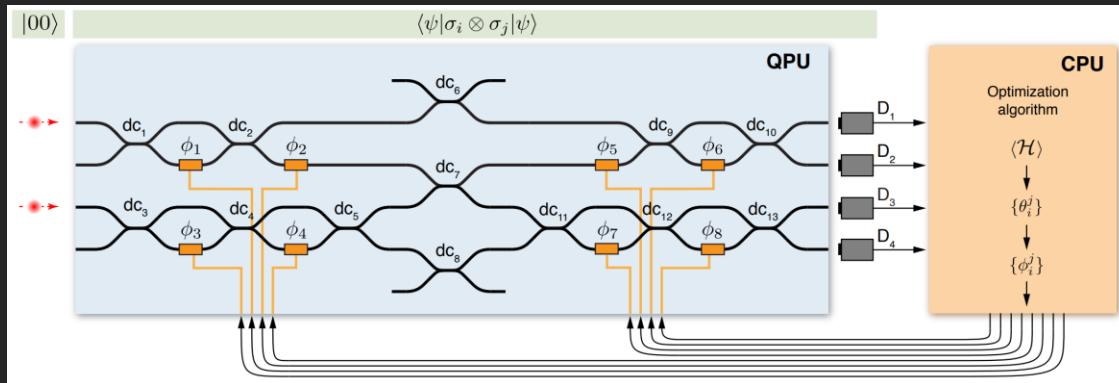
# Quantum Chemistry Simulations



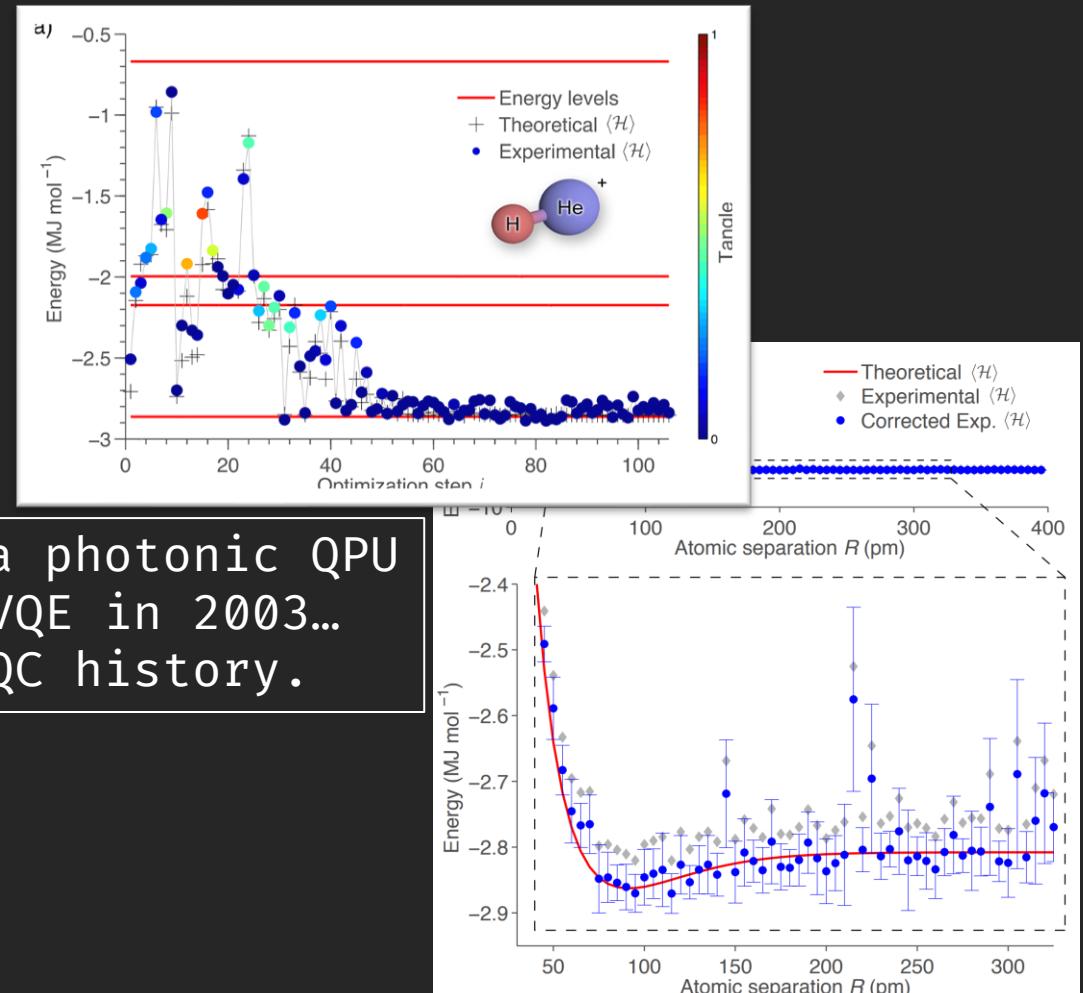
Credit: Nature/Xanadu

<https://doi.org/10.1038/s41586-021-03202-1>

# Quantum Chemistry Simulations



This is a photonic QPU  
running VQE in 2003...  
Ancient QC history.

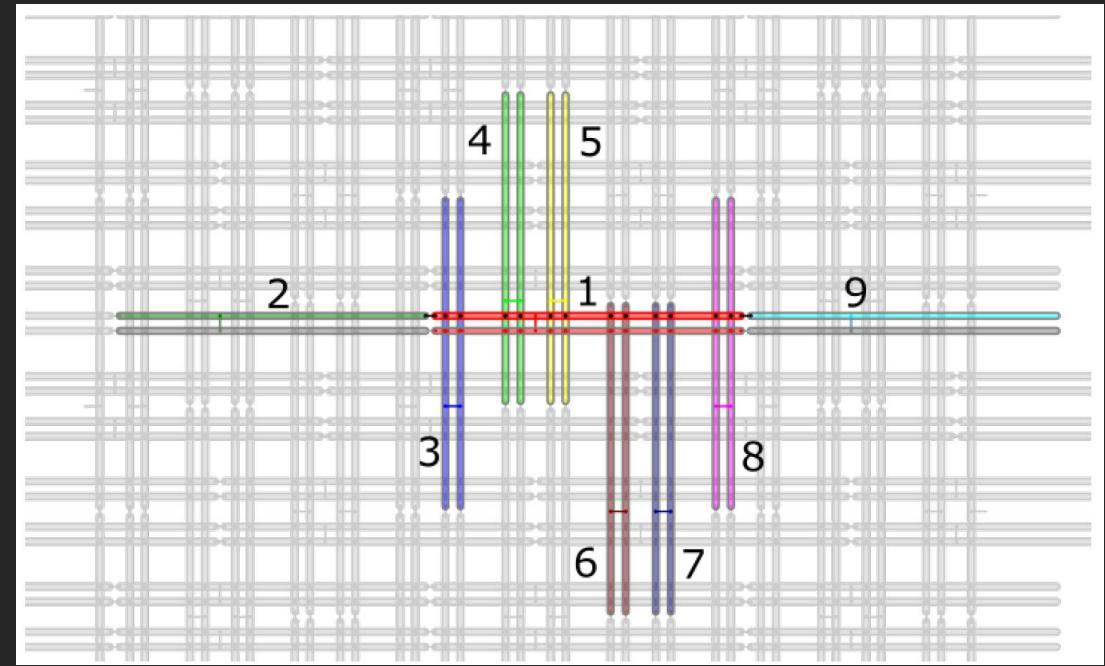
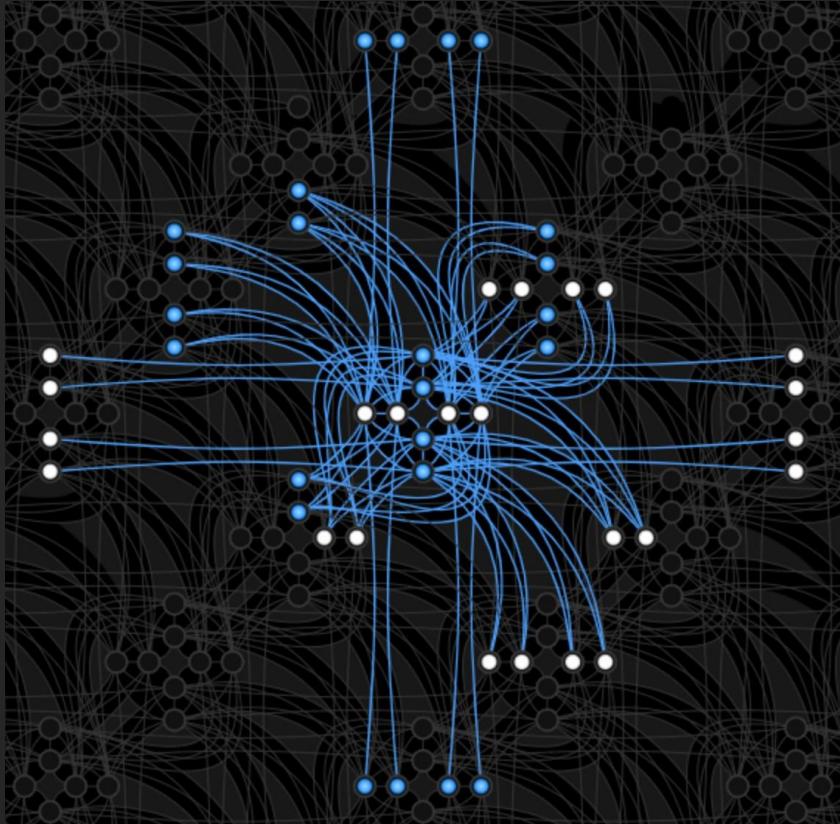


# Quantum Annealers



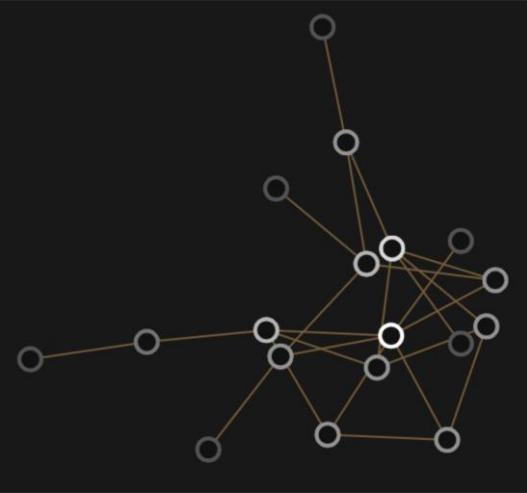
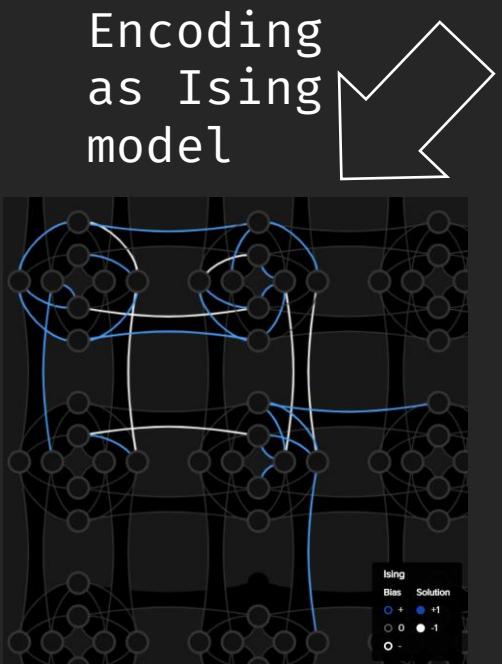
Credit: D-Wave

# D-Wave Advantage (5760 qubits, Pegasus topology)



Credit: D-Wave

# Example: Max-cut with QA

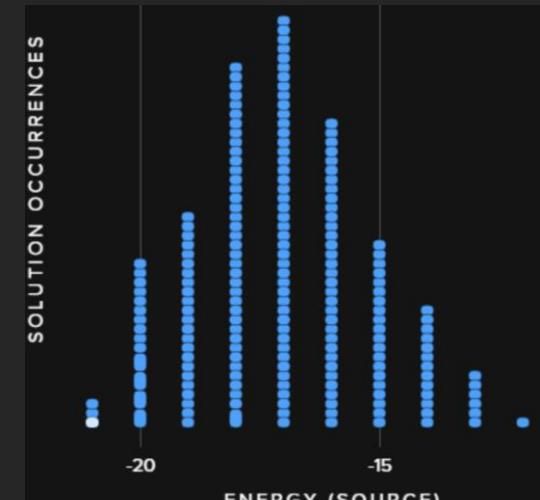
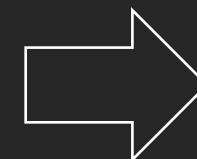


Min energy solution

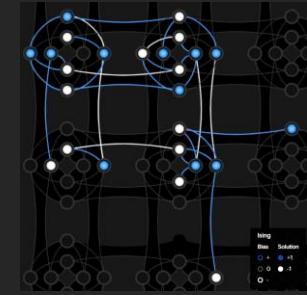


NAME (CHIP ID)	DESCRIPTION
DW_2000Q_6	D-Wave 2000Q lower-noise system
QUBITS	SUPPORTED PROBLEM TYPES
2048	ising, qubo
TOPOLOGY	TAGS
[16,16,4] chimera	lower_noise

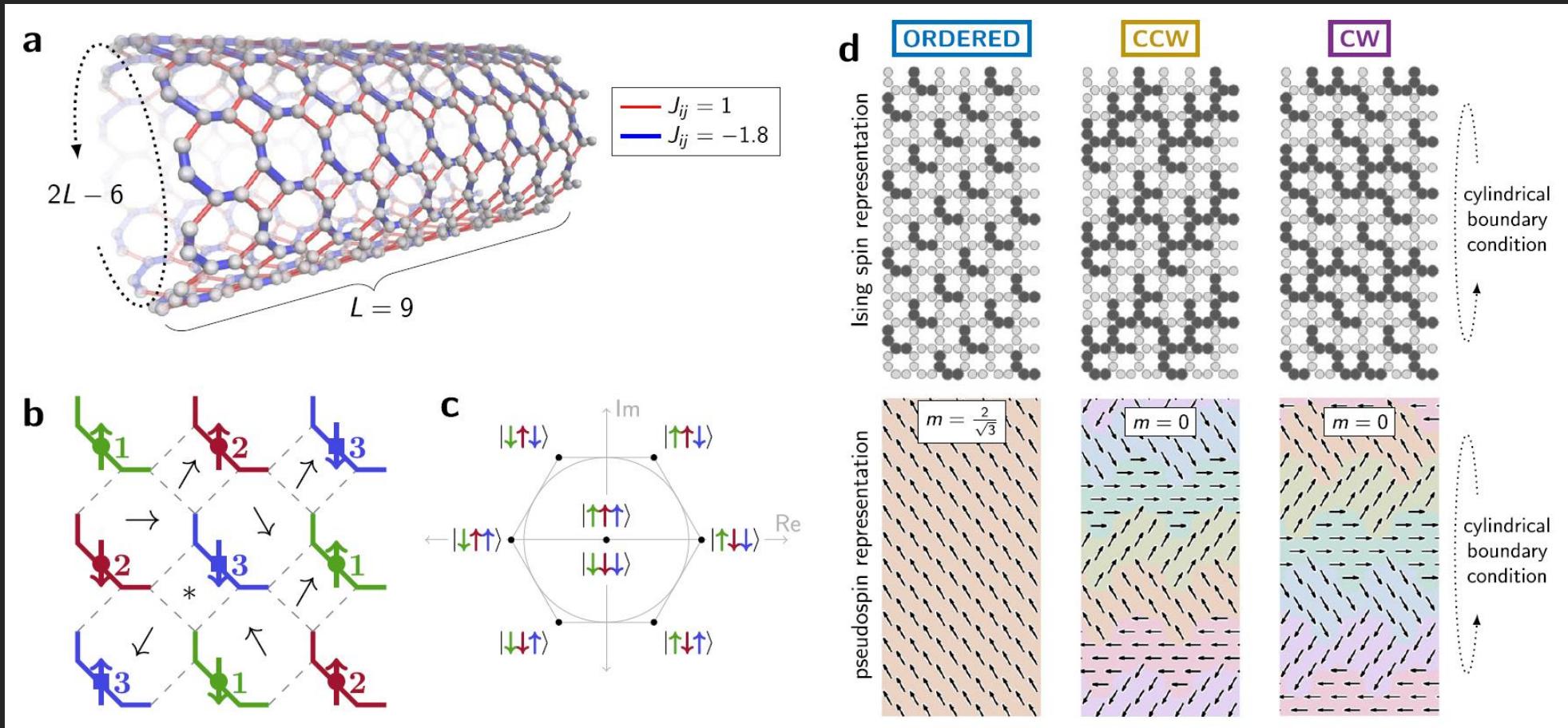
QPU



Samples

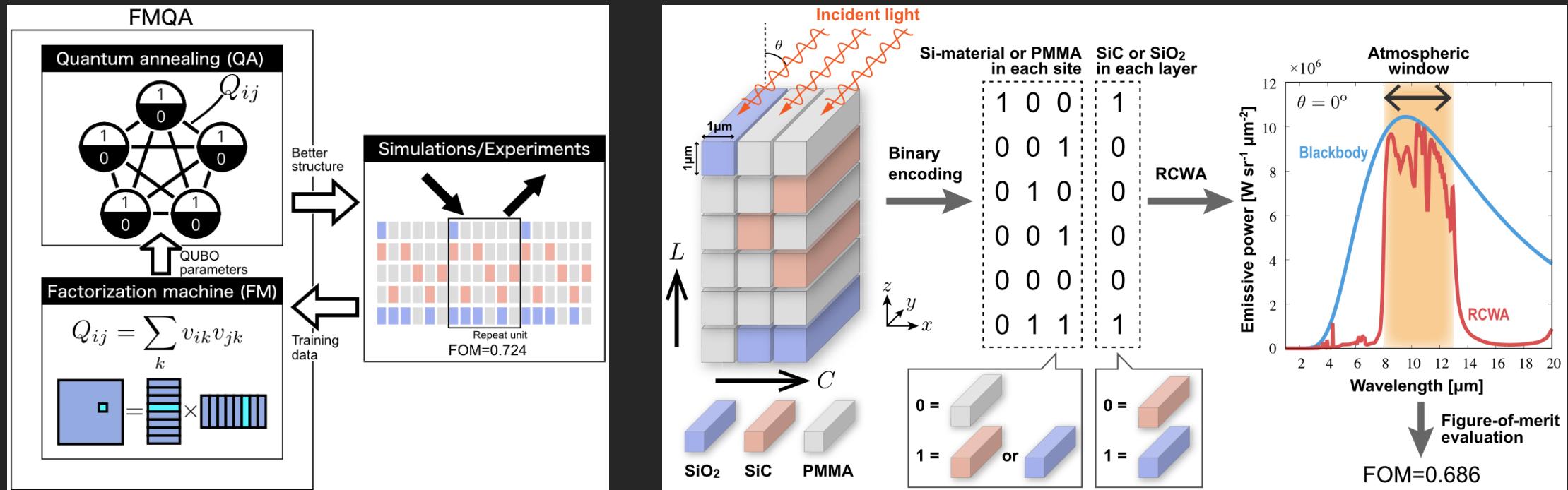


# Spin-glass Simulations



Credit: Nature/D-Wave  
<https://doi.org/10.1038/s41467-021-20901-5>

# Metamaterial Design



Credit: APS/various institutions (on D-Wave HW)  
<https://doi.org/10.1103/PhysRevResearch.2.013319>

# QKD and QRNG (not QC)



Credit: ID Quantique

# Software Providers

# IBM Quantum

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 →](#)

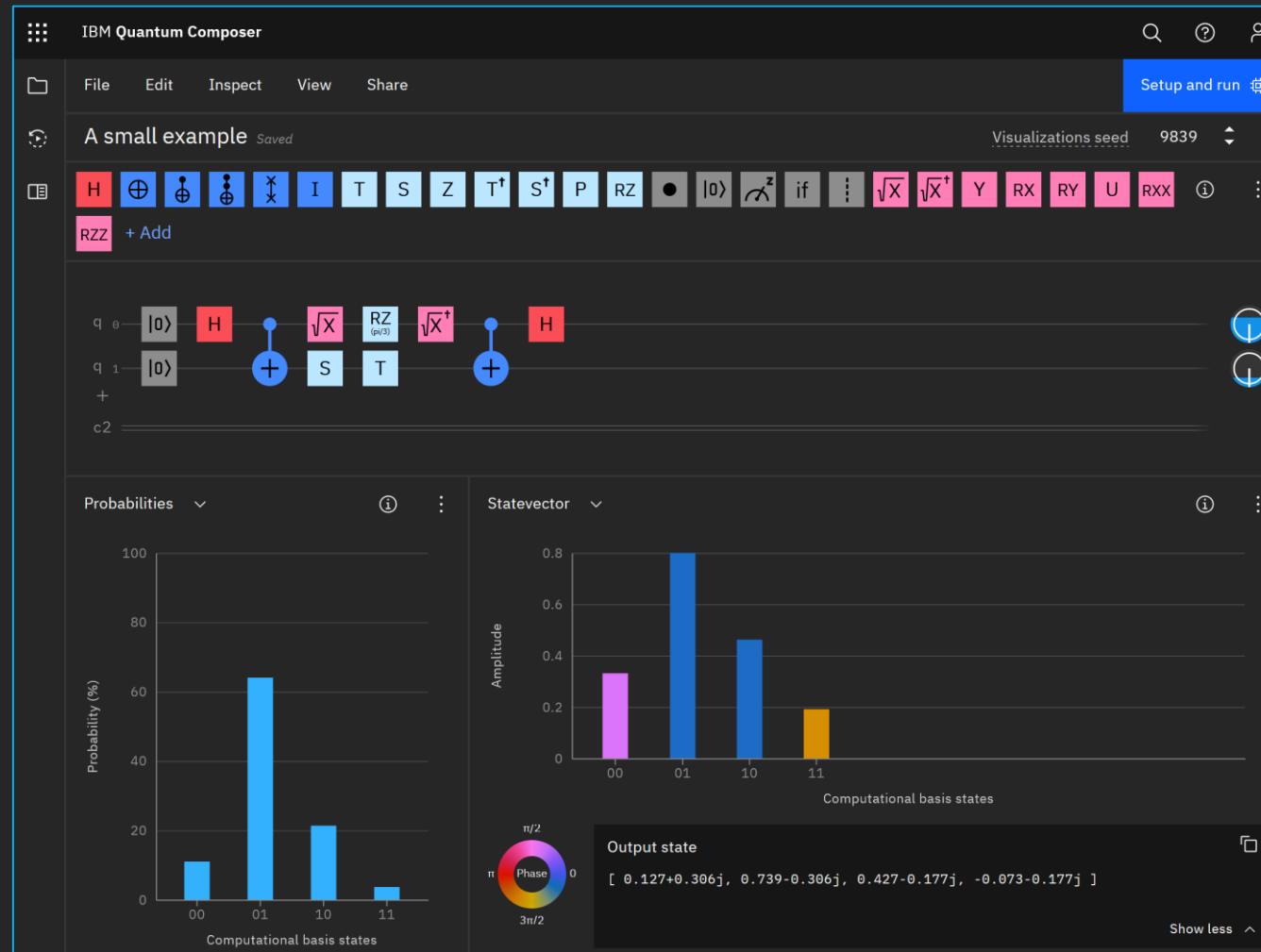
New reservation Card Table

Search by system name All systems (23) ▾

<a href="#">ibmq_montreal</a> System status: Online Processor type: Falcon r4  27 Qubits 128 Quantum volume	<a href="#">ibmq_kolkata</a> System status: Offline Processor type: Falcon r5.11  27 Qubits 128 Quantum volume	<a href="#">ibmq_mumbai</a> System status: Offline Processor type: Falcon r5.1  27 Qubits 128 Quantum volume	<a href="#">ibmq_dublin</a> System status: Online Processor type: Falcon r4  27 Qubits 64 Quantum volume
<a href="#">ibm_hanoi</a> System status: Online - Queue paused Processor type: Falcon r5.11  27 Qubits 64 Quantum volume	<a href="#">ibm_cairo</a> System status: Online Processor type: Falcon r5.11  27 Qubits 64 Quantum volume	<a href="#">ibmq_manhattan</a> System status: Offline Processor type: Hummingbird r2  65 Qubits 32 Quantum volume	<a href="#">ibmq_brooklyn</a> System status: Online - Queue paused Processor type: Hummingbird r2  65 Qubits 32 Quantum volume
<a href="#">ibmq_toronto</a> System status: Online Processor type: Falcon r4  27 Qubits 32 Quantum volume	<a href="#">ibmq_sydney</a> System status: Online Processor type: Falcon r4  27 Qubits 32 Quantum volume	<a href="#">ibmq_guadalupe</a> System status: Online Processor type: Falcon r4P  16 Qubits 32 Quantum volume	<a href="#">ibmq_casablanca</a> System status: Online Processor type: Falcon r4H  7 Qubits 32 Quantum volume

Credit: [IBM - Quantum Services](#)

# IBM Quantum



Credit: [IBM - Quantum Composer](#)

# IBM Quantum

IBM Quantum Lab

New file +

Filter files by name

Lab files / ... / qiskit / algorithms /

Name	Last Modified
01_algorithms_introduction.ipynb	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 01\_algorithms\_introduction.ipynb 07\_grover\_examples.ipynb

Markdown

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)
```

Bitstring	Probability
0000	0.045
0001	0.028
0010	0.028
0011	0.031
0100	0.031
0101	0.031
0110	0.031
0111	0.031
1000	0.029
1001	0.029
1010	0.029
1011	0.029
1100	0.029
1101	0.035
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

# IBM Quantum

The screenshot shows the Qiskit website homepage. At the top, there is a navigation bar with links for Overview, Learn, Community (with a dropdown arrow), and Documentation. Below the navigation bar, there is a purple header section containing the Qiskit logo and the text "qiskit 0.24.0 see release notes". The main content area features a large illustration of two people working on a complex quantum computing system, which includes several large cylindrical components and a ladder. To the left of the illustration, the text "Open-Source Quantum Development" is displayed. Below this text, a description of Qiskit is provided: "Qiskit [kiss-kit] is an open source SDK for working with quantum computers at the level of pulses, circuits and application modules." At the bottom left, there is a purple button labeled "Get started" with a white arrow pointing right. At the bottom right, there is a small icon of a person with a gear inside a circle.

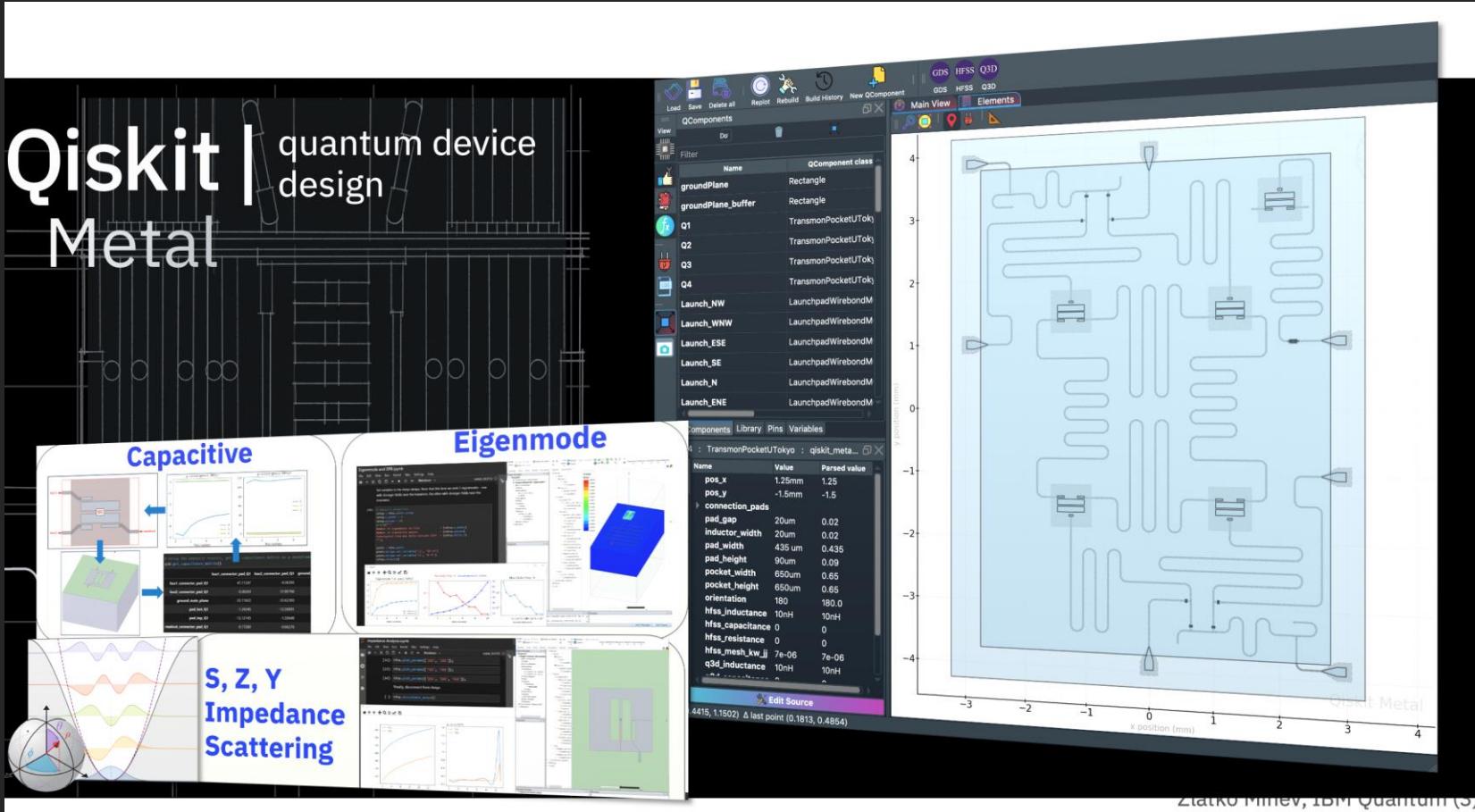
Credit: [IBM - Qiskit](#)

# IBM Quantum

The screenshot shows the Qiskit documentation homepage. At the top, there's a navigation bar with links for Getting started, Tutorials, Partners, Applications, Experiments, Resources, and Github. Below that is a language dropdown set to English, a search bar labeled "Search Docs", and a breadcrumb trail "Docs > Qiskit 0.31.0 documentation". To the right of the search bar is a "Qiskit 0.31.0 documentation" section with three bullet points: "Interested in applications of quantum comp.", "Interested in running experiments on real qu.", and "Interested in quantum hardware design?". The main content area features a quantum circuit diagram with six qubits (q<sub>0</sub> to q<sub>5</sub>) and various orange operations. Below the circuit, the title "Qiskit 0.31.0 documentation" is displayed with a small icon. A paragraph explains Qiskit's purpose: "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." Another paragraph states: "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." At the bottom, there are two columns: "What is quantum computing?" (with a link to "Get cracking") and "Access to quantum systems" (with a link to "Qiskit Partners"). On the left sidebar, there are sections for Frontmatter (with links to Quantum computing in a nutshell, Introduction to Qiskit, Release Notes, Contributing to Qiskit, Local Configuration, and Frequently Asked Questions), Libraries (with a link to Circuit Library), and API References (with links to Qiskit Terra, Qiskit Aer, Qiskit Ignis (deprecated), Qiskit Aqua (deprecated), and Qiskit IBM Quantum Provider).

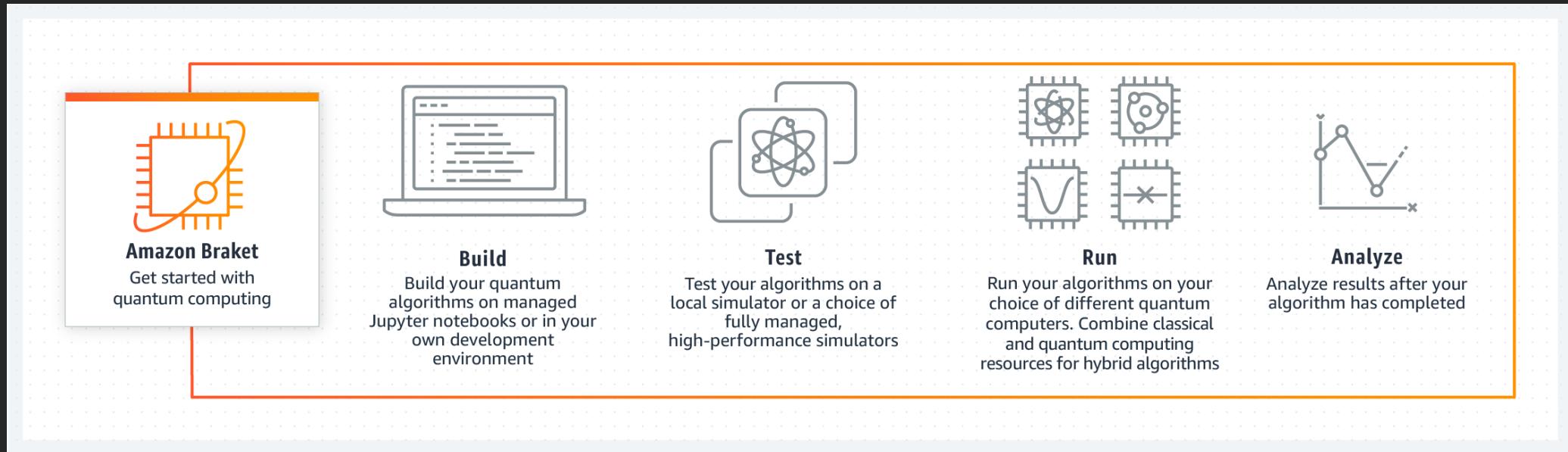
Credit: [IBM – Qiskit documentation](#)

# IBM Quantum



Credit: [IBM - Qiskit Metal](#)

# AWS Braket



Credit: [Amazon AWS - Braket](#)

# 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.



The Quantum Computing Company™

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](#)

# PennyLane

The screenshot shows the official website for PennyLane. At the top, there's a navigation bar with links for 'PENNY LANE' (with a logo), 'Quantum machine learning', 'Install', 'Plugins', 'Documentation', 'QHACK' (highlighted in red), 'FAQ', 'Support', and 'GitHub'. Below the navigation is a large teal banner with the word 'PENNY LANE' in white. Underneath, a sub-headline reads: 'A cross-platform Python library for differentiable programming of quantum computers. Train a quantum computer the same way as a neural network.' Three main sections are displayed: 'Learn' (with a thumbnail of a video player showing a quantum circuit), 'Play' (with a snippet of Python code from a Jupyter notebook), and 'Hack' (with a purple button labeled 'Sign up >>'). A dark banner at the bottom lists various partners and ecosystem members: Xanadu (with logo), AWS (with logo), IBM (with logo), Google (with logo), Rigetti (with logo), Microsoft (with logo), Zapata (with logo), QAT (with logo), TensorFlow (with logo), and PyTorch (with logo).

PENNY LANE

Quantum machine learning

Install

Plugins

Documentation

QHACK

FAQ

Support

Github

# PENNY LANE

A cross-platform Python library for differentiable programming of quantum computers. Train a quantum computer the same way as a neural network.

**Learn**

Sit back and learn about the field of quantum machine learning, explore key concepts, and view our selection of curated videos.

Quantum machine learning >>

**Play**

Tutorials to introduce core QML concepts, including quantum nodes, optimization, and devices, via easy-to-follow examples.

Demos >>

**Hack**

Join us for QHACK, the quantum machine learning hackathon. Feb 17-26th 2021.

Sign up >>

PennyLane supports a growing ecosystem, including a wide range of quantum hardware and machine learning libraries

XANADU

aws

IBM

Google

rigetti

Microsoft

ZAPATA

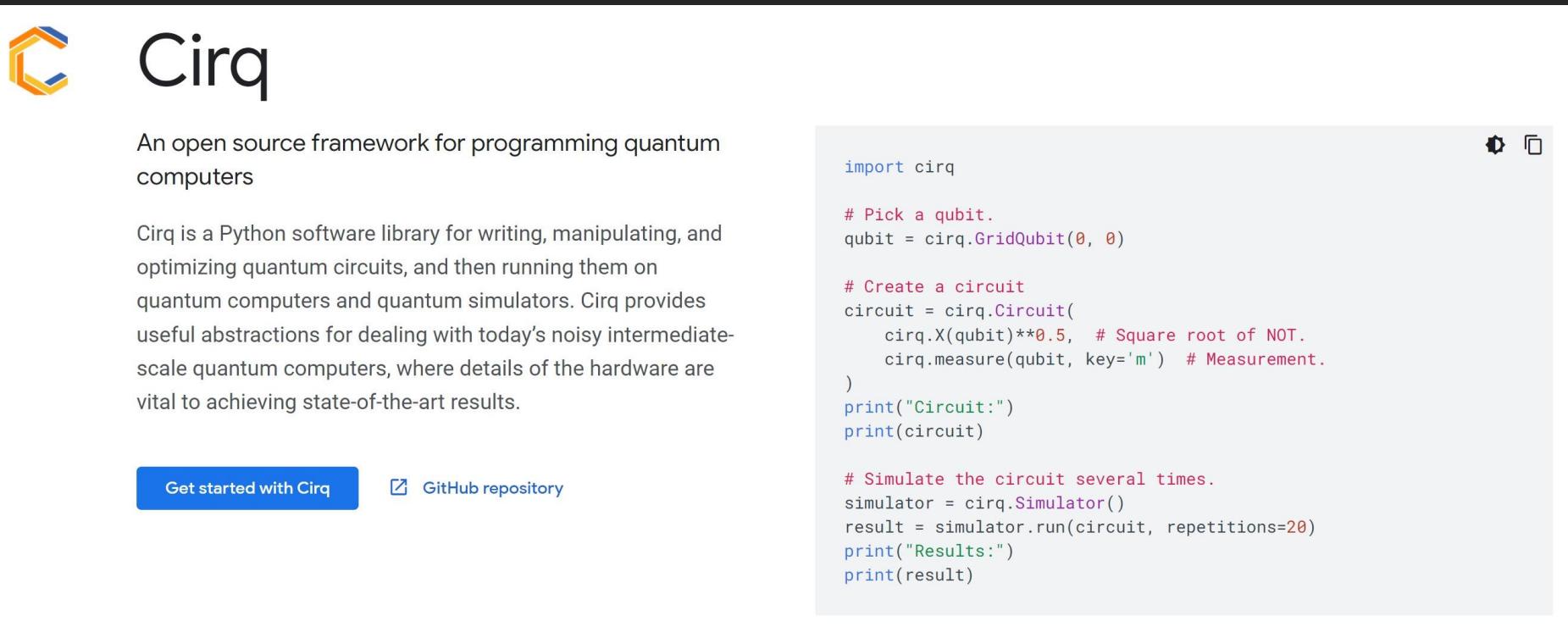
QAT

TensorFlow

PyTorch

Credit: [Xanadu - PennyLane](#)

# Google Cirq



The image shows a screenshot of the Cirq website on the left and a code editor window on the right.

**Cirq Website:**

- Logo:** A stylized orange and blue 'C' logo.
- Title:** Cirq
- Text:** An open source framework for programming quantum computers
- Description:** 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.
- Buttons:** Get started with Cirq, GitHub repository

**Code Editor:**

```
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

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 Ocean/Leap

The screenshot shows the D-Wave Leap IDE interface. On the left, a code editor displays a Python script named `maximum_cut.py`. The script defines a graph, initializes a QUBO matrix, and runs a sampler on a QPU to find a maximum cut. The right side features a "PROBLEM INSPECTOR" window with two tabs: "Source - Force Directed" and "Target - QPU". The Source tab shows a small graph with nodes and edges. The Target tab shows a larger, more complex graph with nodes and connections. Below the inspector is a "Console" window showing the command run and its output, which includes a table of sets and their energies, and a message about a saved plot. The bottom status bar provides system information.

```
maximum_cut.py
29     # Create empty graph
30     G = nx.Graph()
31
32     # Add edges to the graph (also adds nodes)
33     G.add_edges_from([(1,2),(1,3),(2,4),(3,4),(3,5),(4,5)])
34
35     # ----- Set up our QUBO dictionary -----
36
37     # Initialize our Q matrix
38     Q = defaultdict(int)
39
40     # Update Q matrix for every edge in the graph
41     for i, j in G.edges:
42         Q[(i,i)]+= -1
43         Q[(j,j)]+= -1
44         Q[(i,j)]+= 2
45
46     # ----- Run our QUBO on the QPU -----
47     # Set up QPU parameters
48     chainstrength = 8
49     numruns = 10
50
51     # Run the QUBO on the solver from your config file
52     sampler = EmbeddingComposite(DWaveSampler({"qpu": True}))
53     response = sampler.sample_qubo(Q,
54                                     chain_strength=chainstrength,
55                                     num_reads=numruns,
56                                     label='Example - Maximum Cut')
57     dwave.inspector.show(response)
```

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

Your plot is saved to maxcut\_plot.png

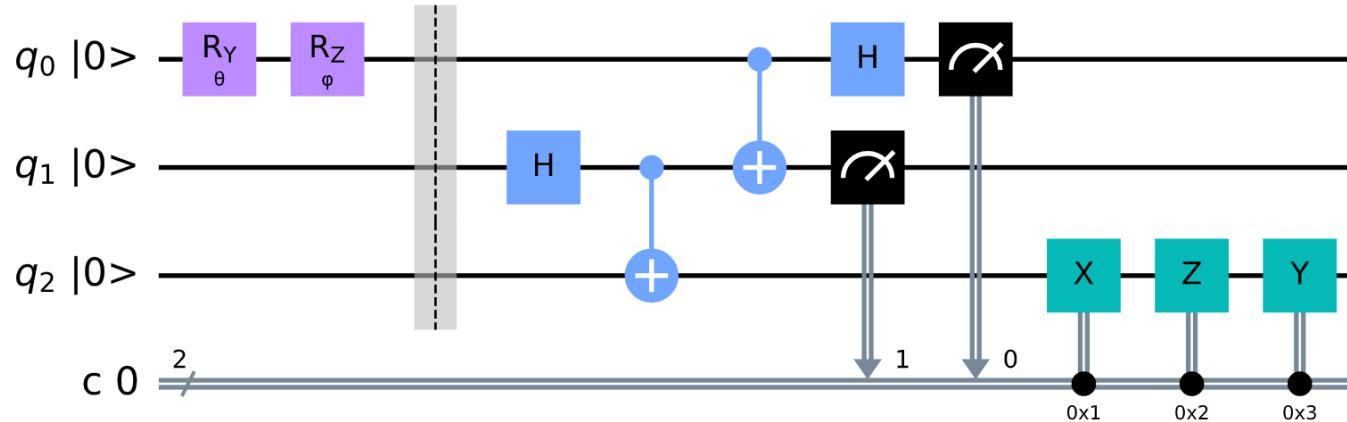
dwave-examples/maximum-cut Python 3.7.10 64-bit master\* 0 0 ▲ 0

Ln 57, Col 33 LF UTF-8 Spaces: 4 No open ports Python

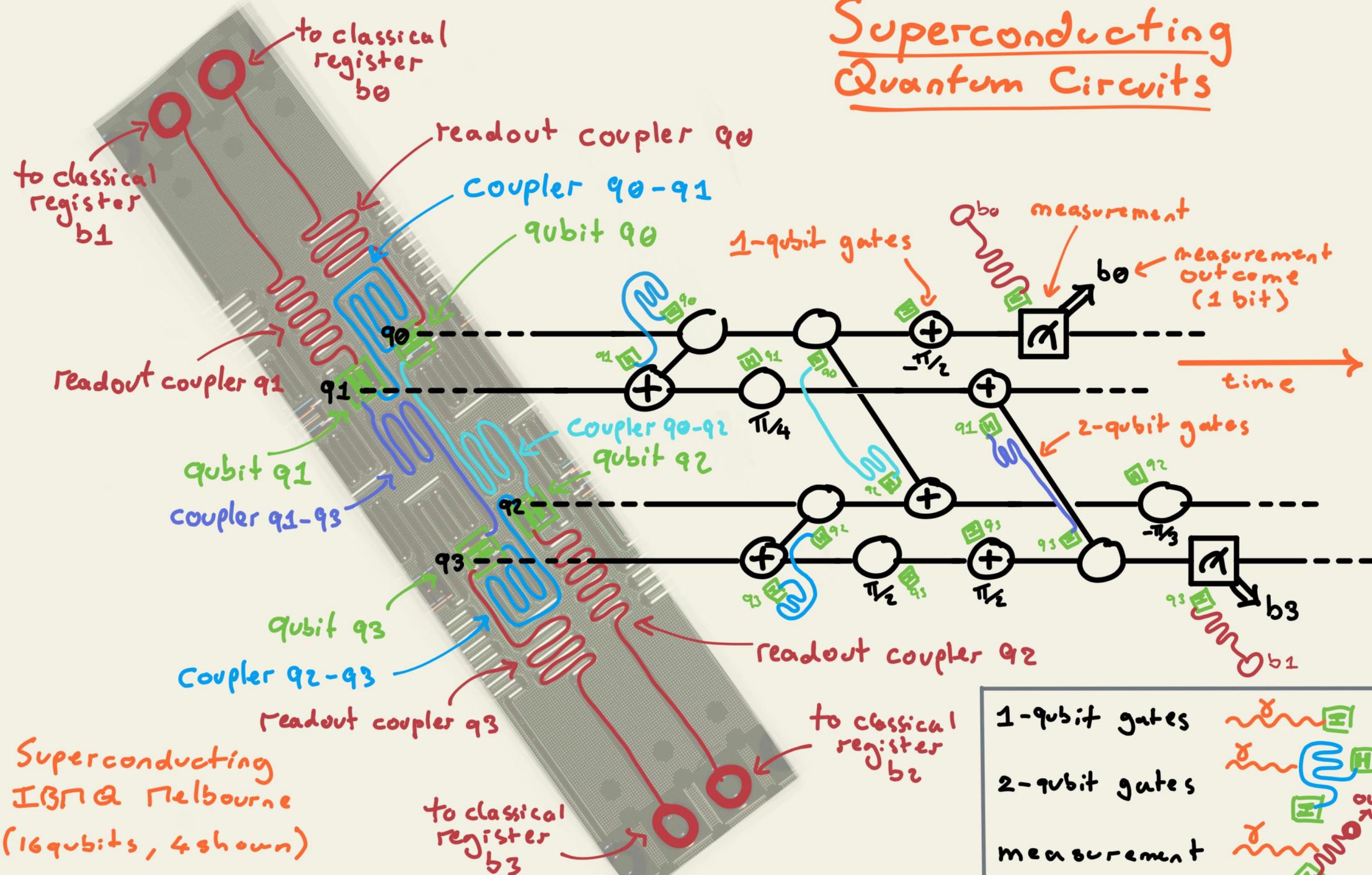
Credit: [D-Wave - Leap](#)

# 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)
```



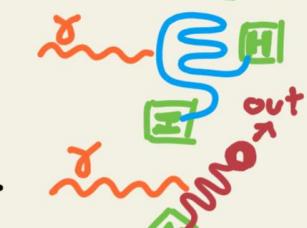
# Superconducting Quantum Circuits



1-qubit gates

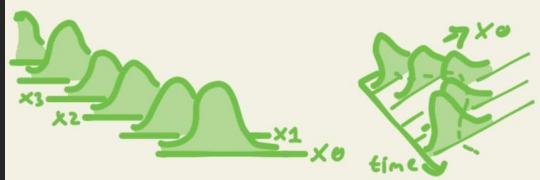


2-qubit gates

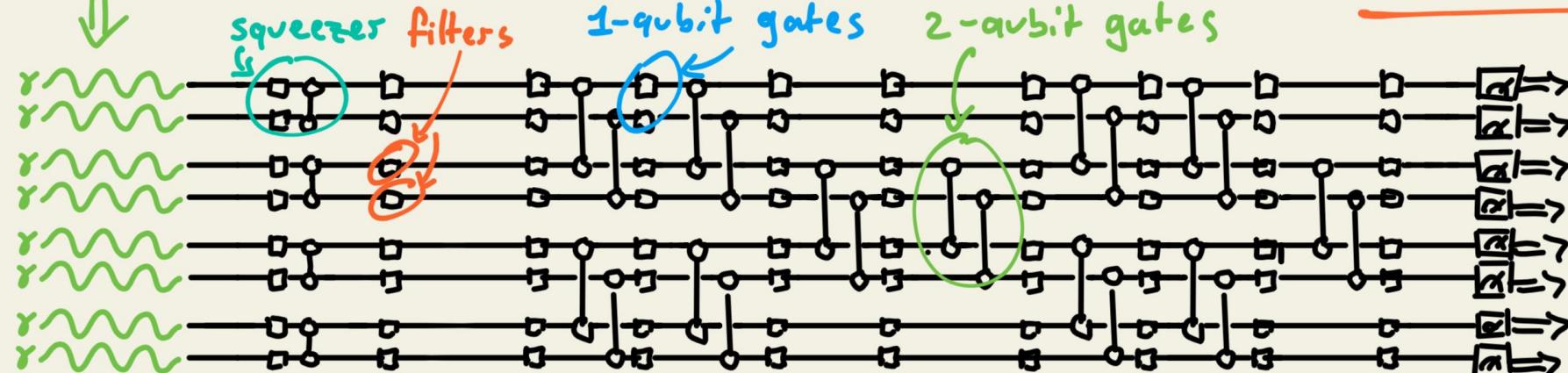
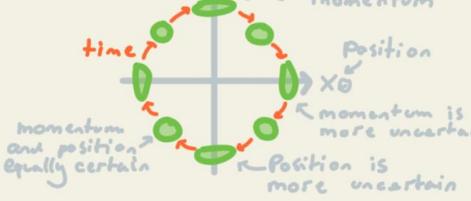
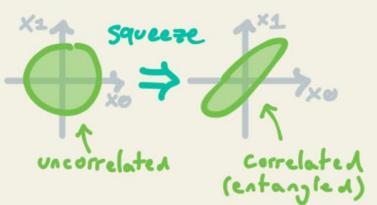


measurement



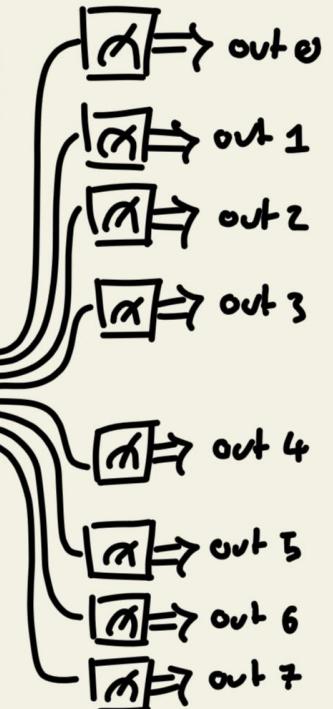
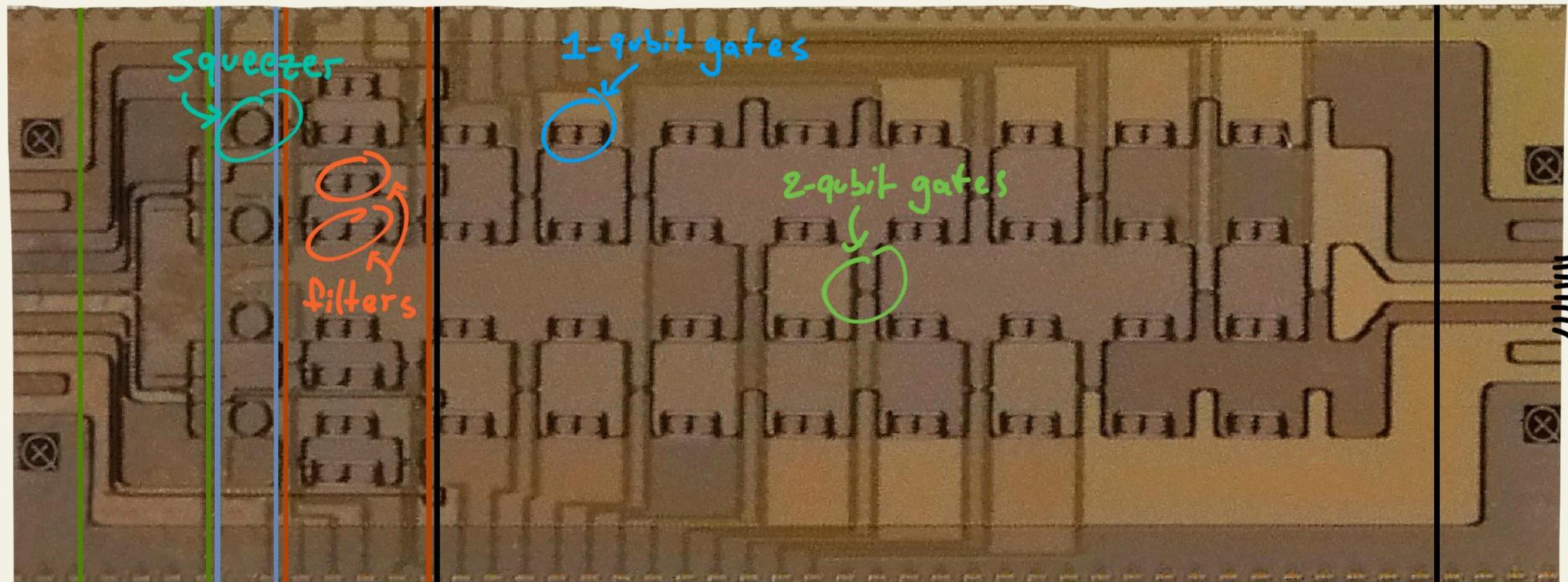


pump laser injects photons

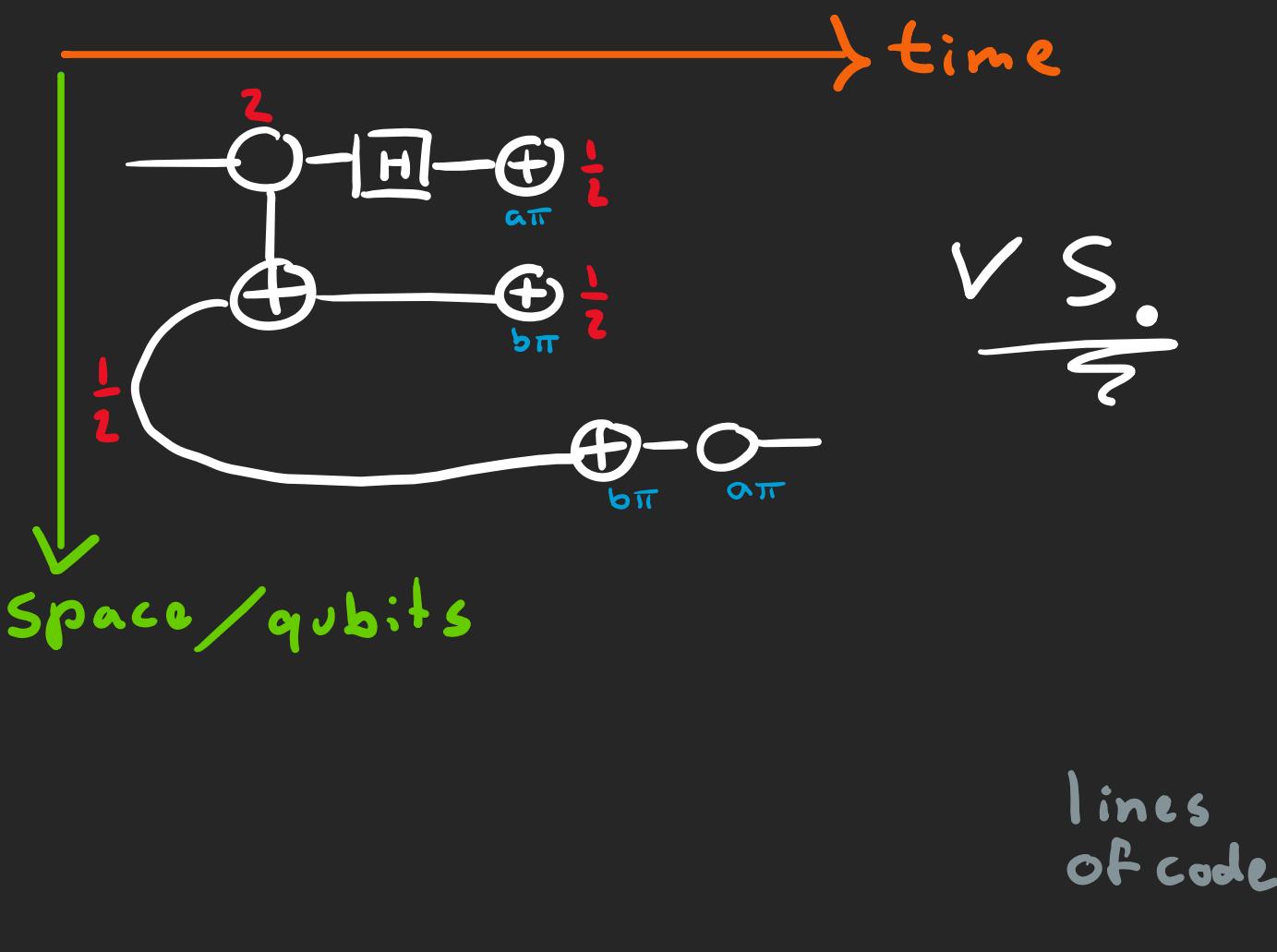


## Photonic Quantum Circuits

measurements performed off-chip.  
- homodyne  
- photon number



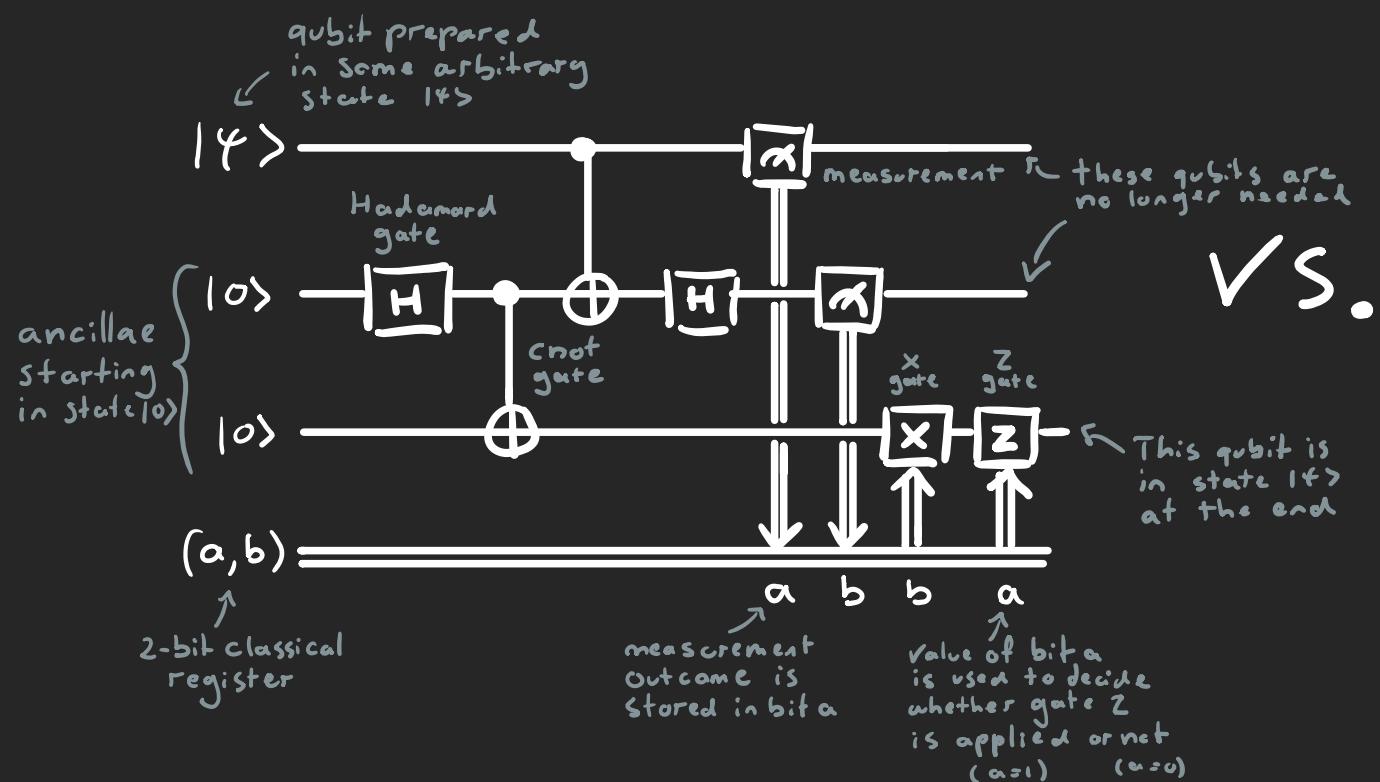
# Diagrams as code



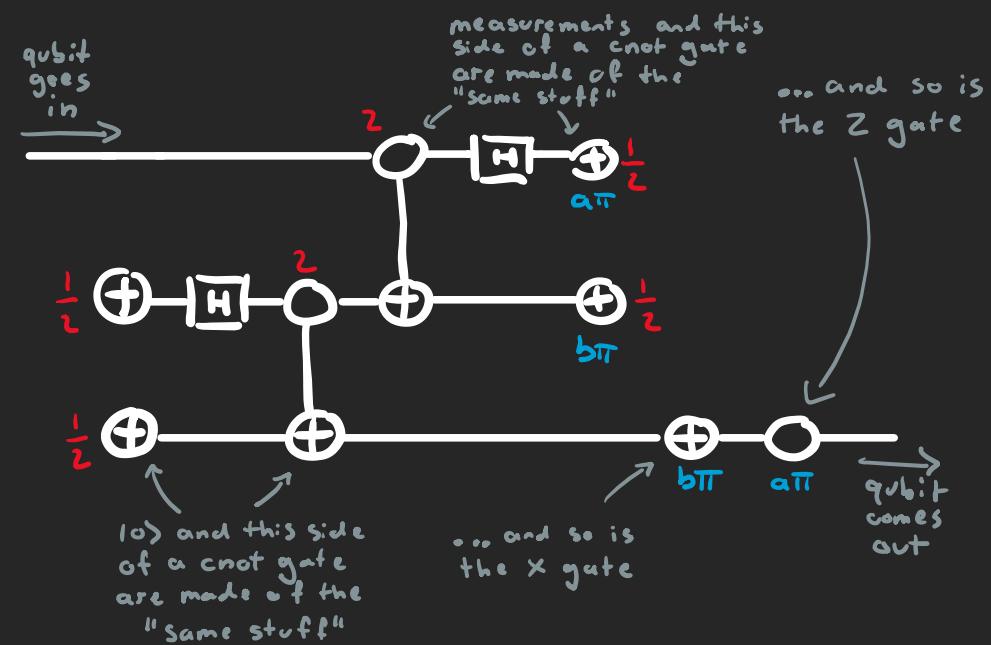
```
1  from pyquil.quil import Program
2  from pyquil import api
3  from pyquil.gates import X, Z, H, CNOT
4
5
6  def make_bell_pair(q1, q2):
7      """Makes a bell pair between qubits q1 and q2
8      """
9      return Program(H(q1), CNOT(q1, q2))
10
11
12 def teleport(start_index, end_index, ancilla_index):
13     """Teleport a qubit from start to end using an ancilla qubit
14     """
15     program = make_bell_pair(end_index, ancilla_index)
16
17     # do the teleportation
18     program.inst(CNOT(start_index, ancilla_index))
19     program.inst(H(start_index))
20
21     # measure the results and store them in classical registers [0] and [1]
22     program.measure(start_index, 0)
23     program.measure(ancilla_index, 1)
24
25     program.if_then(1, X(2))
26     program.if_then(0, Z(2))
27
28     program.measure(end_index, 2)
29
30     return program
31
32
33 if __name__ == '__main__':
34     qvm = api.QVMConnection()
35
36     # initialize qubit 0 in |1>
37     teleport_demo = Program(X(0))
38     teleport_demo += teleport(0, 2, 1)
39     print("Teleporting |1> state: {}".format(qvm.run(teleport_demo, [2])))
40
41     # initialize qubit 0 in |0>
42     teleport_demo = Program()
43     teleport_demo += teleport(0, 2, 1)
44     print("Teleporting |0> state: {}".format(qvm.run(teleport_demo, [2])))
45
46     # initialize qubit 0 in |+>
47     teleport_demo = Program(H(0))
48     teleport_demo += teleport(0, 2, 1)
49     print("Teleporting |+> state: {}".format(qvm.run(teleport_demo, [2], 10)))
50
51     print(Program(X(0)).measure(0, 0).if_then(0, Program(X(1))))
```

# Circuits vs Diagrams

## Quantum Circuit

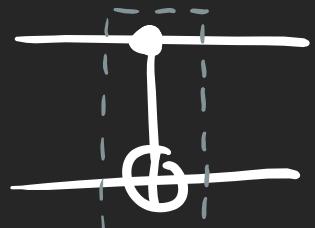


## ZX Diagram



In this course, the two words will be used interchangeably, always meaning a ZX diagram.

# Circuits vs Diagrams



Cnot gate is just one symbol

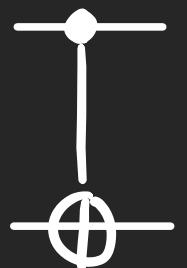
vs.



Cnot gate is made of two separate pieces, connected by a "virtual" qubit



$|0\rangle$

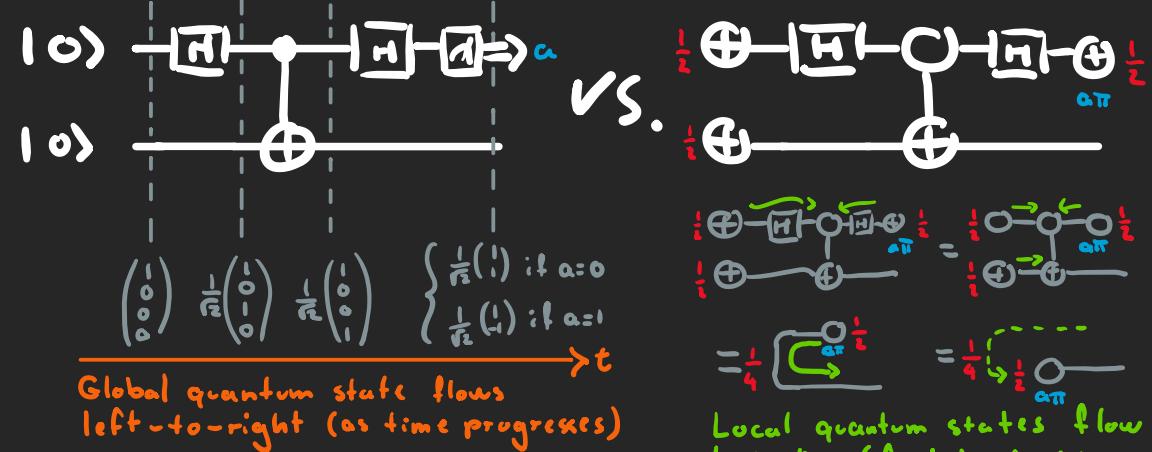


vs.

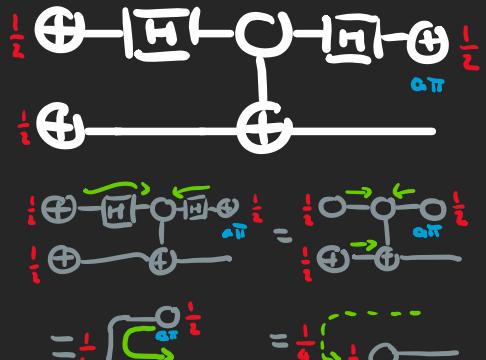
$|0\rangle$



These are five unrelated symbols

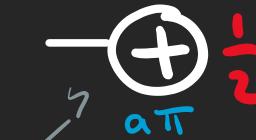
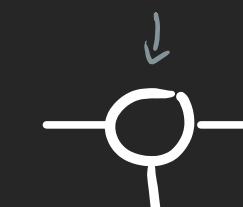
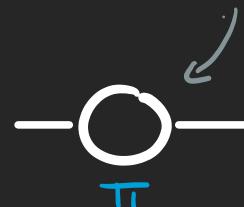


Global quantum state flows left-to-right (as time progresses)



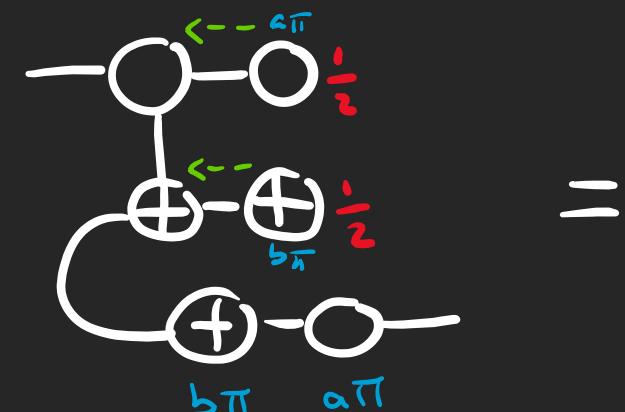
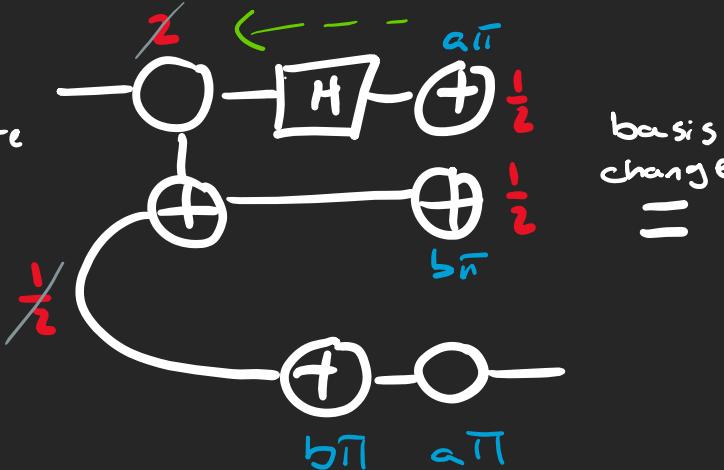
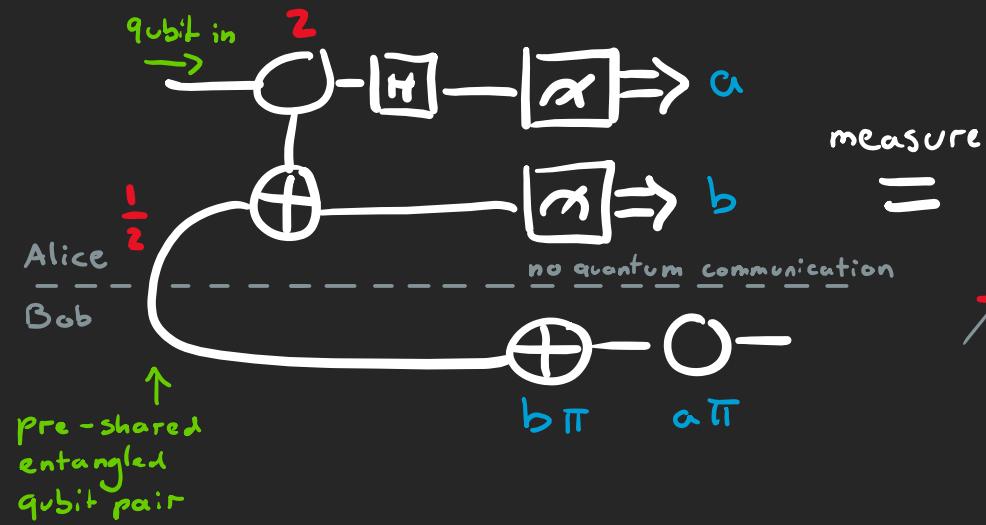
Local quantum states flow logically (up, down, sideways)

made of the "same stuff" (Z spiders)



made of the "same stuff" (X spiders)

# Diagrammatic Reasoning



$$\text{fusion } \times 2 = \frac{1}{4}$$

Diagram illustrating the fusion of two loops with a total phase of  $2\pi$ . The result is a single loop with a phase of  $\pi$ .

$$\frac{1}{4} = \frac{1}{4} \text{ fusion} = \frac{1}{4}$$

Diagram illustrating the fusion of three loops with a total phase of  $3\pi$ . The result is a single loop with a phase of  $\pi$ .

$$P(a,b) = \frac{1}{4}$$

Diagram illustrating the probability  $P(a,b) = \frac{1}{4}$ . A wavy line represents a path from Alice's qubit to Bob's qubit.

Annotations:

- Alice's qubit is "teleported" to Bob
- Alice Bob
- qubit in
- qubit out

# ZX Calculus



Bob Coecke  
Chief Scientist



Ross Duncan  
Head of Quantum Software



Aleks Kissinger  
Professor of Quantum  
Computation

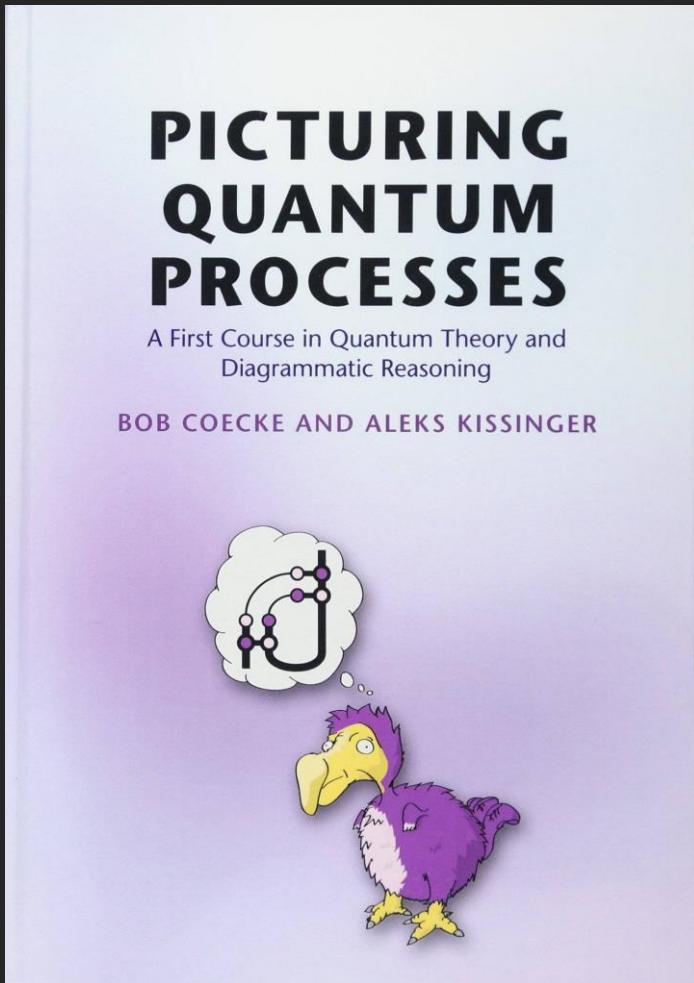


QUANTINUUM



UNIVERSITY OF  
OXFORD

# ZX Calculus



Since its invention in 2007, the ZX calculus has seen more than 150 publications!

You can find them listed [here](#).

## ZX-calculus for the working quantum computer scientist

John van de Wetering<sup>1,2</sup>

<sup>1</sup>Radboud Universiteit Nijmegen

<sup>2</sup>Oxford University

December 29, 2020

## Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus

Ross Duncan<sup>1,2</sup>, Aleks Kissinger<sup>3</sup>, Simon Perdrix<sup>4</sup>, and John van de Wetering<sup>5</sup>

<sup>1</sup>University of Strathclyde, 26 Richmond Street, Glasgow G1 1XH, UK

<sup>2</sup>Cambridge Quantum Computing Ltd, 9a Bridge Street, Cambridge CB2 1UB, UK

<sup>3</sup>Department of Computer Science, University of Oxford

<sup>4</sup>CNRS LORIA, Inria-MOCQUA, Université de Lorraine, F 54000 Nancy, France

<sup>5</sup>Institute for Computing and Information Sciences, Radboud University Nijmegen

May 27, 2020

## PyZX: Large Scale Automated Diagrammatic Reasoning

Aleks Kissinger

University of Oxford

[aleks.kissinger@cs.ox.ac.uk](mailto:aleks.kissinger@cs.ox.ac.uk)

John van de Wetering

Radboud University

[john@vdwetering.name](mailto:john@vdwetering.name)

Most ZX calculus literature uses a different notation (colour-coded):

$$\begin{array}{c} -G \sim \leftrightarrow \\ -\oplus \sim \times \end{array} \begin{array}{c} -\text{green circle} \\ -\text{red circle} \end{array}$$



[PyZX on GitHub](#)

LET'S DIVE IN!