Introduction to programming quantum computers

Mark Fingerhuth
ProteinQure Inc.
University of KwaZulu-Natal
Quantum Open Source Foundation (QOSF)

2020 Symposium on Data Science & Statistics (SDSS)
WebEx Virtual
June 3, 2020

/m-fingerhuth
@mark_fingerhuth

qosf

Agenda

- Introductions (10min)
- First block (1hr 15min)
  - Lecture 1: Quantum bits, architectures and gates (30min)
  - Tutorial 1: Quantum bits & gates with Quil (45min)
- BREAK (10min)
- Second block (1hr 30min)
  - Lecture 2: Quantum programming (30min)
  - Tutorial 2: Quantum programming in PyQuil (1hr)
- BREAK (10min)
- Final summary + Q&A (15min)

Introductions

qosf

About me

- BSc in Liberal Arts & Sciences (Maastricht University, Netherlands)
  - Thesis at UKZN on "**Quantum-enhanced machine learning**" under the supervision of Dr. Maria Schuld and Prof. Francesco Petruccione
- Moved to Canada in 2017 and co-founded the biotech startup **ProteinQure**
- Co-founded the **Quantum Open Source Foundation** together with Dr. Peter Wittek and Tomas Babej
- 6 scientific publications in applied quantum computing (Science Advances, PLOS One, EPJ, ...)

About me

Implementing a distance-based classifier with a quantum interference circuit

Maria Schuld,[1,*] Mark Fingerhuth,[1,2,†] and Francesco Petruccione[1,3,‡]

Coarse-grained lattice protein folding on a quantum annealer

Tomas Babej* and Christopher Ing
ProteinQure Inc., Toronto, Canada

A quantum alternating operator ansatz with hard and soft constraints for lattice protein folding

Mark Fingerhuth*
ProteinQure Inc., Toronto, Canada and
University of KwaZulu-Natal, Durban, South Africa

Molecular Docking with Gaussian Boson Sampling

Leonardo Banchi,[1] Mark Fingerhuth,[2] Tomas Babej,[2] Christopher Ing,[2] and Juan Miguel Arrazola[1]
[1] *Xanadu, 372 Richmond St W, Toronto, M5V 1X6, Canada*
[2] *ProteinQure Inc., 192 Spadina Ave, Toronto, M5T 2C2, Canada*

Gaussian Boson Samplers are photonic quantum devices with the potential to perform tasks that are intractable for classical systems. As with other near-term quantum technologies, an outstanding challenge is to identify specific problems of practical interest where these quantum devices can prove useful. Here we show that Gaussian Boson Samplers can be used to predict molecular docking

About me

Implementing a distance-based classifier with a quantum interference circuit

Maria Schuld,[1,*] Mark Fingerhuth,[1,2,†] and Francesco Petruccione[1,3,‡]

Coarse-grained lattice protein folding on a quantum annealer

Tomas Babej* and Christopher Ing
ProteinQure Inc., Toronto, Canada

A quantum alternating operator ansatz with hard and soft constraints for lattice protein folding

Mark Fingerhuth*
ProteinQure Inc., Toronto, Canada and University of KwaZulu-Natal, Durban, South Africa

Molecular Docking with Gaussian Boson Sampling

Leonardo Banchi,[1] Mark Fingerhuth,[2] Tomas Babej,[2] Christopher Ing,[2] and Juan Miguel Arrazola[1]
[1] *Xanadu, 372 Richmond St W, Toronto, M5V 1X6, Canada*
[2] *ProteinQure Inc., 192 Spadina Ave, Toronto, M5T 2C2, Canada*

Gaussian Boson Samplers are photonic quantum devices with the potential to perform tasks that are intractable for classical systems. As with other near-term quantum technologies, an outstanding challenge is to identify specific problems of practical interest where these quantum devices can prove useful. Here we show that Gaussian Boson Samplers can be used to predict molecular docking

qosf

Quantum Open Source Foundation

-

www.qosf.org

Introductions: you

- Background: Physics / Computer science / Statistics / other
- Stage of academic life: undergrad / grad / postdoc / other
- Comfort with Python?
- Operating system?
- Quantum computing background?
- Programmed a quantum computer before?

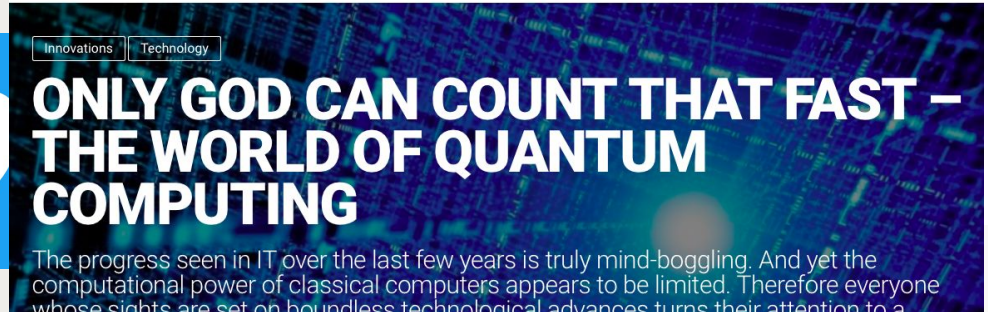
Lecture 1:

Quantum bits, gates & architectures

qosf

What *is* quantum computing?

Quantum computing... everywhere!



How quantum computers will change our lives

Quantum computing... everywhere!

Top stories

Google claims to have reached quantum supremacy

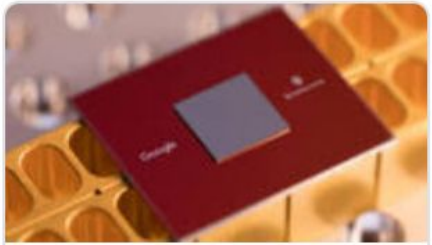
Financial Times

2 days ago

Google researchers have reportedly achieved “quantum supremacy”

MIT Technology Review

2 days ago



Google Says It's Achieved Quantum Supremacy, a World-First: Report

Gizmodo

2 days ago

Quantum computing... everywhere!

Quantum supremacy using a programmable superconducting processor

Google AI Quantum and collaborators[†]

The tantalizing promise of quantum computers is that certain computational tasks might be executed exponentially faster on a quantum processor than on a classical processor. A fundamental challenge is to build a high-fidelity processor capable of running quantum algorithms in an exponentially large computational space. Here, we report using a processor with programmable superconducting qubits to create quantum states on 53 qubits, occupying a state space $2^{53} \sim 10^{16}$. Measurements from repeated experiments sample the corresponding probability distribution, which we verify using classical simulations. While our processor takes about 200 seconds to sample one instance of the quantum circuit 1 million times, a state-of-the-art supercomputer would require approximately 10,000 years to perform the equivalent task. This dramatic speedup relative to all known classical algorithms provides an experimental realization of quantum supremacy on a computational task and heralds the advent of a much-anticipated computing paradigm.

Quantum computing... everywhere!

Quantum supremacy using a programmable superconducting processor

Google AI Quantum and collaborators[†]

The tantalizing promise of quantum computers is that certain computational tasks might be executed exponentially faster on a quantum processor than on a classical processor. A fundamental challenge is to build a high-fidelity processor capable of running quantum algorithms in an exponentially large computational space. Here, we report using a processor with programmable superconducting qubits to create quantum states on 53 qubits, occupying a state space $2^{53} \sim 10^{16}$. Measurements from repeated experiments sample the corresponding probability distribution, which we verify using classical simulations. While our processor takes about 200 seconds to sample one instance of the quantum circuit 1 million times, a state-of-the-art supercomputer would require approximately 10,000 years to perform the equivalent task. This dramatic speedup relative to all known classical algorithms provides an experimental realization of quantum supremacy on a computational task and heralds the advent of a much-anticipated computing paradigm.

Quantum computing... everywhere!

Quantum supremacy using a programmable superconducting processor

Google AI Quantum and collaborators[†]

The tantalizing promise of quantum computers is that certain computational tasks might be executed exponentially faster on a quantum processor than on a classical processor. A fundamental challenge is to build a high-fidelity processor capable of running quantum algorithms in an exponentially large computational space. Here, we report using a processor with programmable superconducting qubits to create quantum states on 53 qubits, occupying a state space $2^{53} \sim 10^{16}$. Measurements from repeated experiments sample the corresponding probability distribution, which we verify using classical simulations. While our processor takes about 200 seconds to sample one instance of the quantum circuit 1 million times, a state-of-the-art supercomputer would require approximately 10,000 years to perform the equivalent task. This dramatic speedup relative to all known classical algorithms provides an experimental realization of quantum supremacy on a computational task and heralds the advent of a much-anticipated computing paradigm.

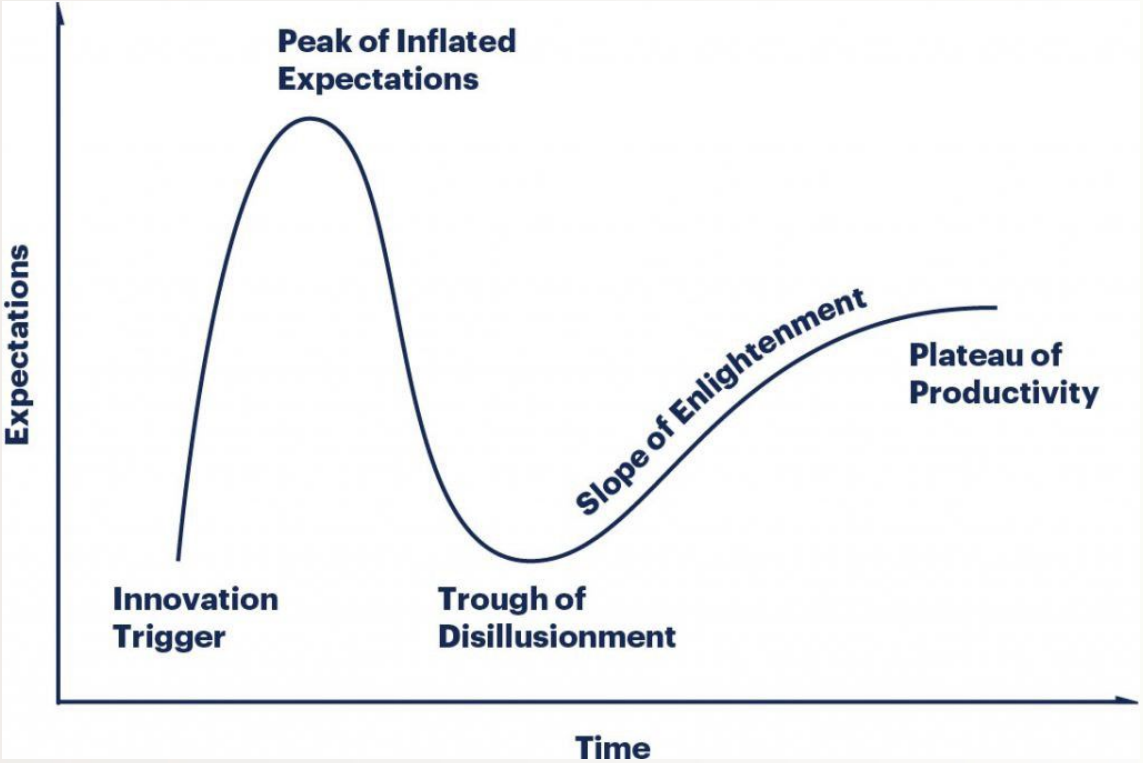
Quantum computing... everywhere!

Quantum supremacy using a programmable superconducting processor

Google AI Quantum and collaborators[†]

The tantalizing promise of quantum computers is that certain computational tasks might be executed exponentially faster on a quantum processor than on a classical processor. A fundamental challenge is to build a high-fidelity processor capable of running quantum algorithms in an exponentially large computational space. Here, we report using a processor with programmable superconducting qubits to create quantum states on 53 qubits, occupying a state space $2^{53} \sim 10^{16}$. Measurements from repeated experiments sample the corresponding probability distribution, which we verify using classical simulations. While our processor takes about 200 seconds to sample one instance of the quantum circuit 1 million times, a state-of-the-art supercomputer would require approximately 10,000 years to perform the equivalent task. This dramatic speedup relative to all known classical algorithms provides an experimental realization of quantum supremacy on a computational task and heralds the advent of a much-anticipated computing paradigm.

The hype cycle



Why *is* quantum computing?

Quantum Computing isn't the answer to everything.

But it will almost certainly free us to solve more problems.

- Will Zeng, Unitary Fund & Goldman Sachs

What we want a quantum computer for

Optimization problems

Encryption & cybersecurity

Artificial General Intelligence

What we want a quantum computer for

Molecular drug design

Organic batteries and solar cells

The power of quantum computing

Quantum computing power scales exponentially with qubits
N bits can exactly simulate log N qubits

This compute unit....

Commodore 64

AWS M4 Instance

1 Million x Commodore 64

Entire Global Cloud

1 Billion x
(1 Million x Commodore 64)

can exactly simulate:

10 Qubits

30 Qubits

60 Qubits

Rigetti 32 qubits
available since this week

Quantum *hardware* architectures

Quantum hardware: the big players

Universal Superconducting Quantum Computers

IBM

rigetti

Google

Microsoft

Quantum Annealers

D-Wave
The Quantum Computing Company™

Quantum-inspired approaches

FUJITSU

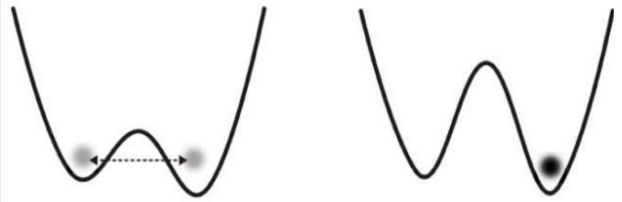
Other Universal QC Hardware

IONQ

XANADU

Honeywell

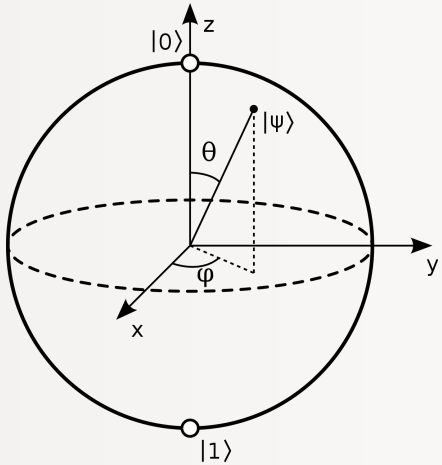
Paradigms of Quantum Computing



Quantum annealing

D:wave
The Quantum Computing Company™

FUJITSU

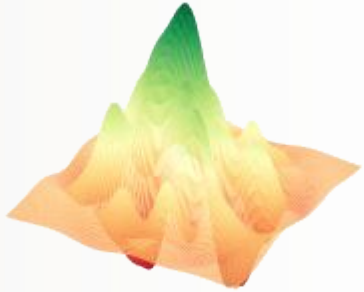


Discrete gate-based

rigetti

IONQ

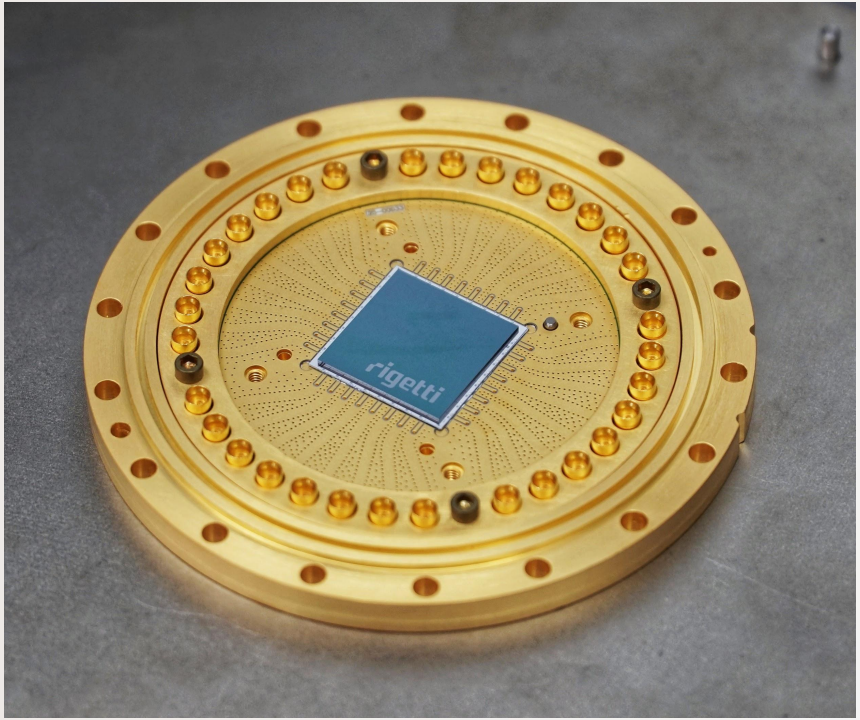
IBM

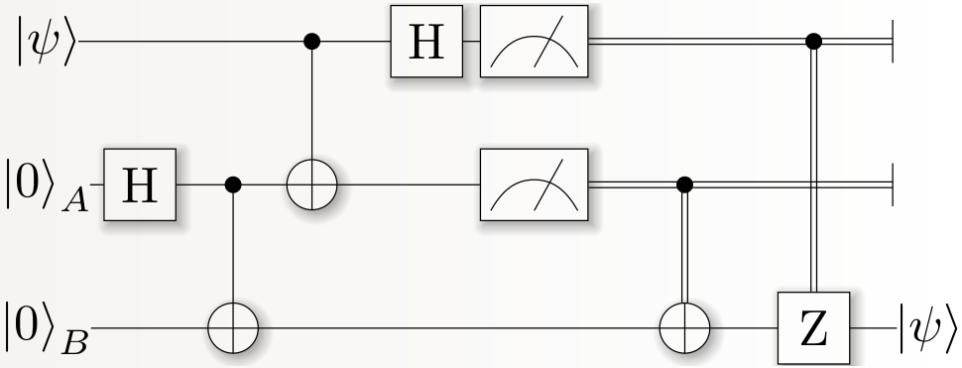


Continuous gate-based

XANADU

Paradigm: Universal gate-based QC





Quantum circuit for quantum teleportation

Big players in universal QC

Software: Qiskit

Software: Cirq

Software: Forest

A *single* qubit

A single qubit as a vector

We can write a two-dimensional **qubit** as the vector:

Global phase

$$|\psi\rangle = e^{i\gamma}\left(\alpha\,|0\rangle + \beta\,|1\rangle\right) = e^{i\gamma}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

Amplitudes

Ground state

Excited state

Calculating probabilities

Complex conjugate

$$\underbrace{\left|\langle \Psi \mid \Psi \rangle\right.}_{\text{Inner product}}\left|^2 = 1\right.$$

The **Bohr rule** states that the inner product **must equal to unity.**

Probabilities

We can **always** drop the **global phase** since it's immeasurable:

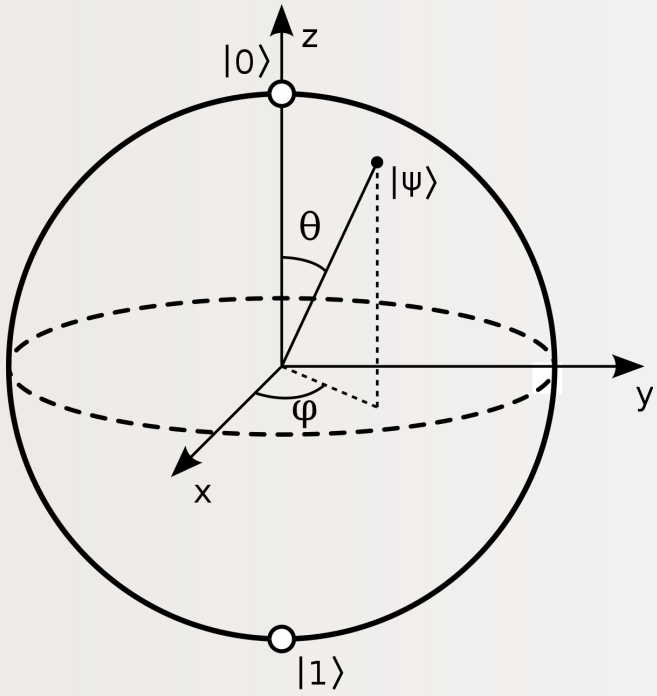
Global phase

$$|\psi\rangle = \cancel{e^{i\gamma}}\left(\alpha\,|0\rangle + \beta\,|1\rangle\right) = \cancel{e^{i\gamma}}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

since

$$|e^{i\gamma}\alpha|^2 = e^{i\gamma}\alpha e^{-i\gamma}\alpha^\dagger = e^{i\gamma - i\gamma}\alpha\alpha^\dagger = \alpha\alpha^\dagger$$

Visualizing a single qubit



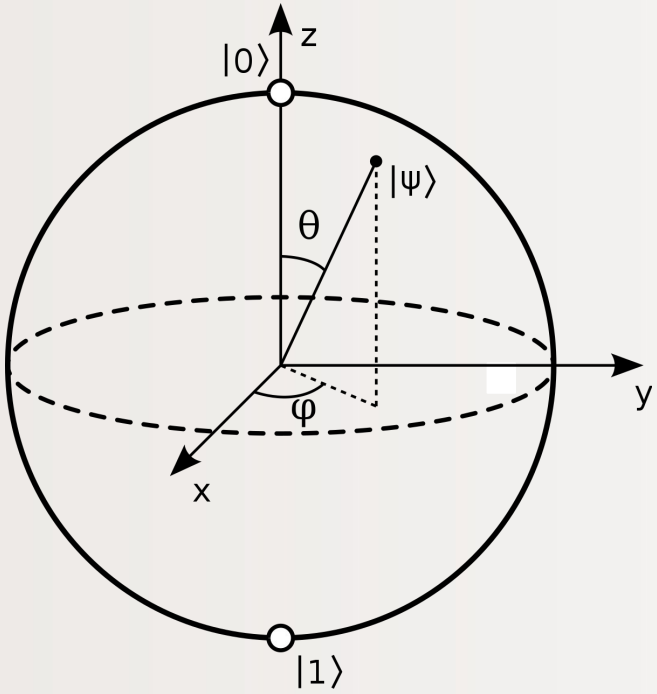
$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

$$|\psi\rangle = \cos(\tfrac{\theta}{2})\,|0\rangle + e^{i\phi}\sin(\tfrac{\theta}{2})\,|1\rangle$$

where $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$

$$x = \sin(\theta)\cos(\phi)$$
$$y = \sin(\theta)\sin(\phi)$$
$$z = \cos(\theta)$$

The Bloch sphere



Quantum gates rotate the Bloch vector

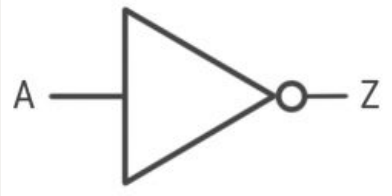
The Bloch sphere provides a **visual tool** to understand quantum gate operations.

The Bloch sphere **only applies to single qubit states**. Visualizing multi-qubit states is out of the scope of this tutorial.

Single-qubit gates

Classical NOT Gate

NOT



Input	Output
0	1
1	0

Single-qubit gates

Quantum X (NOT) Gate

Input	Output
0	1
1	0

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$? \quad \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$|0\rangle \qquad |1\rangle$$

Single-qubit gates

Quantum X (NOT) Gate

Input	Output
0	1
1	0

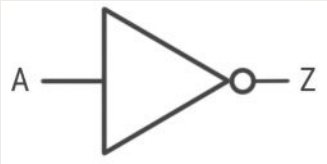
$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{|0\rangle} \rightarrow \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{|1\rangle}$$

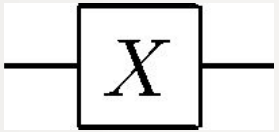
Single-qubit gates

Quantum X (NOT) Gate

NOT → X



→



$$\mathrm{X}\,|0\rangle = |1\rangle$$

$$\mathrm{X}\,|1\rangle = |0\rangle$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

swaps amplitudes!

Single-qubit gates

Hadamard gate (H)

$$|0\rangle \text{ to } \frac{|0\rangle + |1\rangle}{\sqrt{2}} \text{ and } |1\rangle \text{ to } \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Single-qubit gates

Operator	Gate(s)		Matrix
Pauli-X (X)	$-\boxed{\mathbf{X}}-$	$-\oplus-$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)	$-\boxed{\mathbf{Y}}-$		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)	$-\boxed{\mathbf{Z}}-$		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)	$-\boxed{\mathbf{H}}-$		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)	$-\boxed{\mathbf{S}}-$		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)	$-\boxed{\mathbf{T}}-$		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$

Multi-qubit systems

Notation of quantum states

$$|000\rangle$$

Is the **super lazy** physicist's version of

$$|0\rangle\ |0\rangle\ |0\rangle$$

which is the **lazy** physicist's version of

$$|0\rangle \otimes |0\rangle \otimes |0\rangle$$

Notation of quantum states

$$|000\rangle$$

Qubit **2** Qubit **1** Qubit **0**

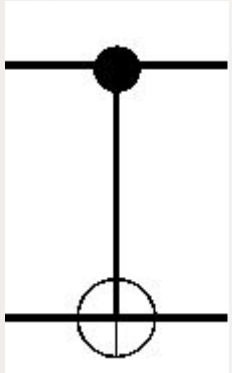
This is a **common convention** in quantum computing.
You will explore the reasoning behind this in the next tutorial.

Two-qubit quantum gates

Controlled-NOT (CNOT) gate

Control qubit

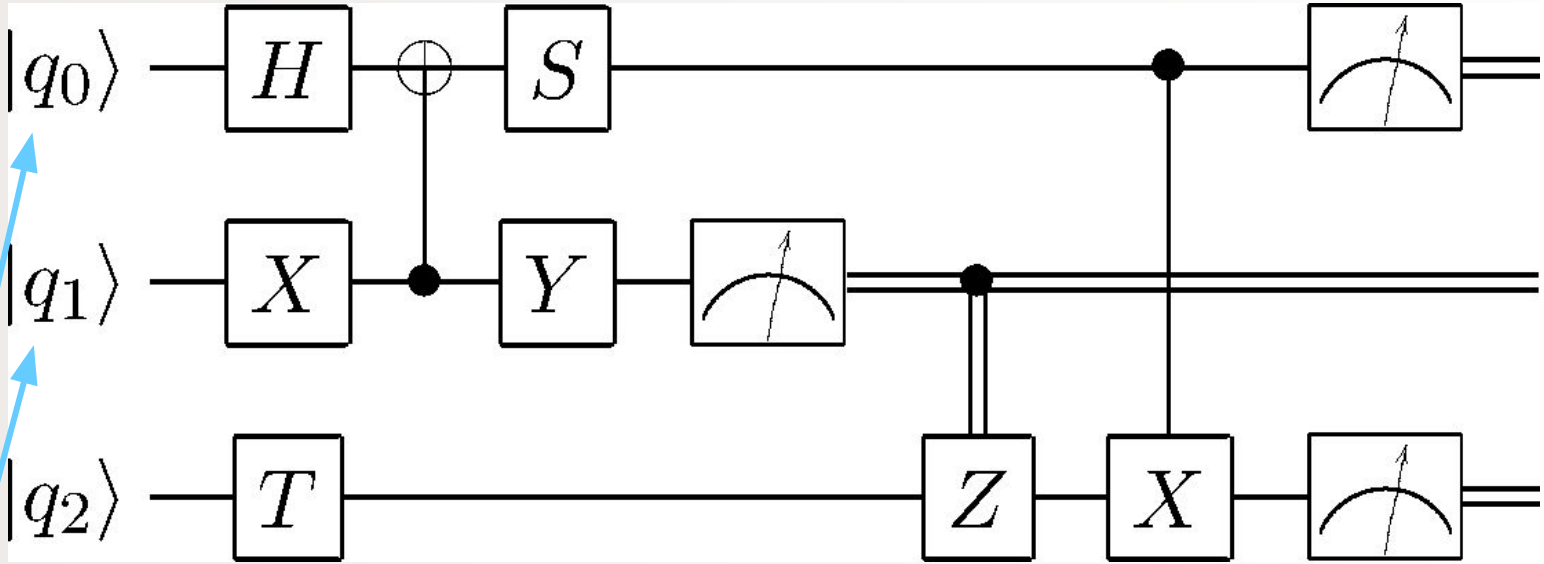
Target qubit



Target qubit	Control qubit	Output
0	0	00
0	1	11
1	0	10
1	1	01

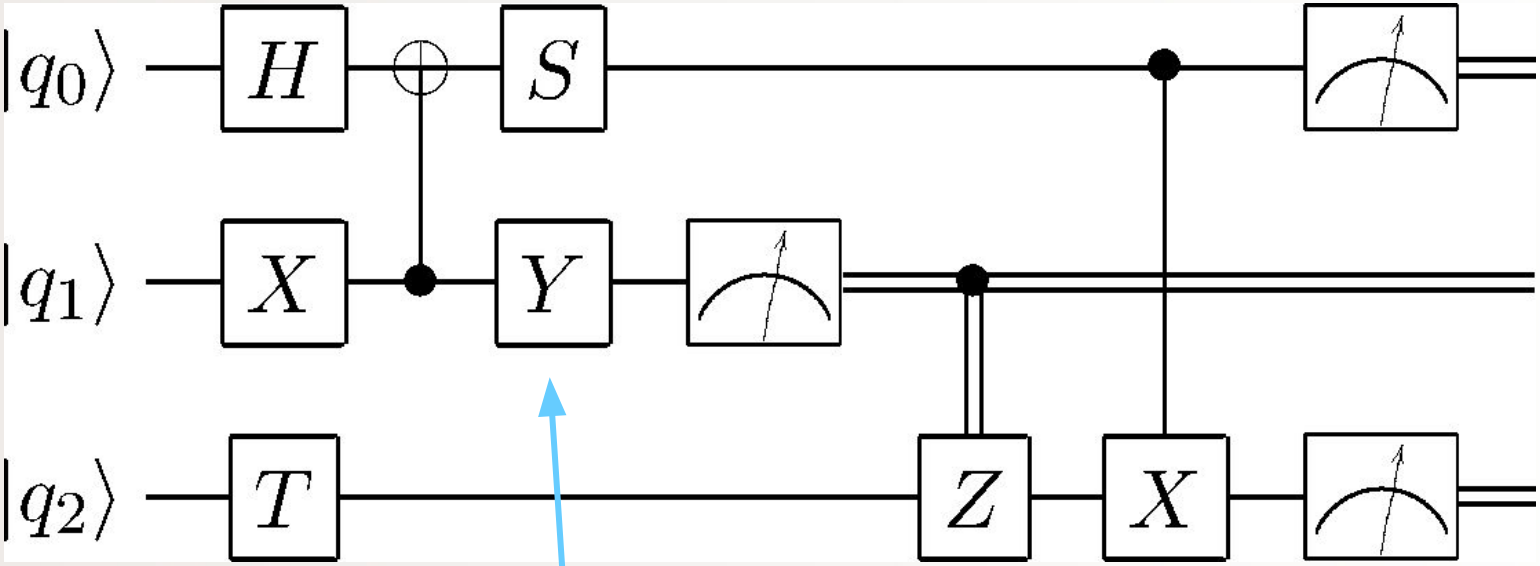
Essential for **entangling** two qubits!

Reading quantum circuit diagrams



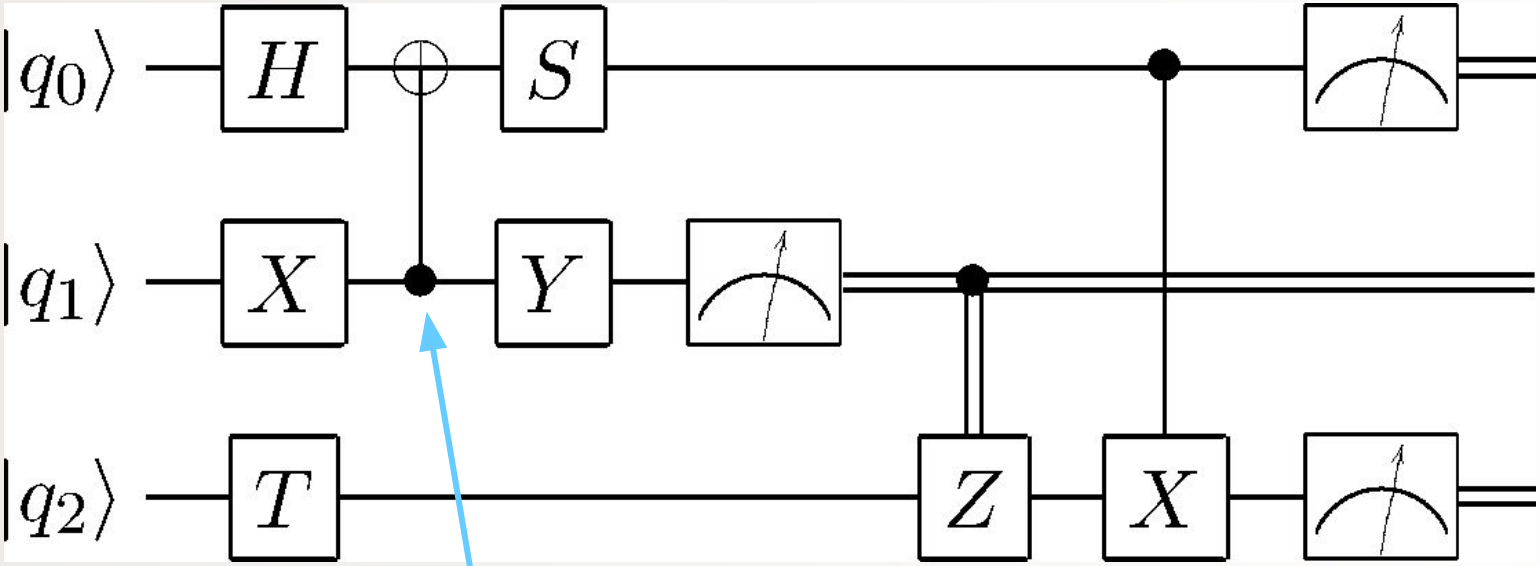
Initial qubit state ($|000\rangle$)

Reading quantum circuit diagrams



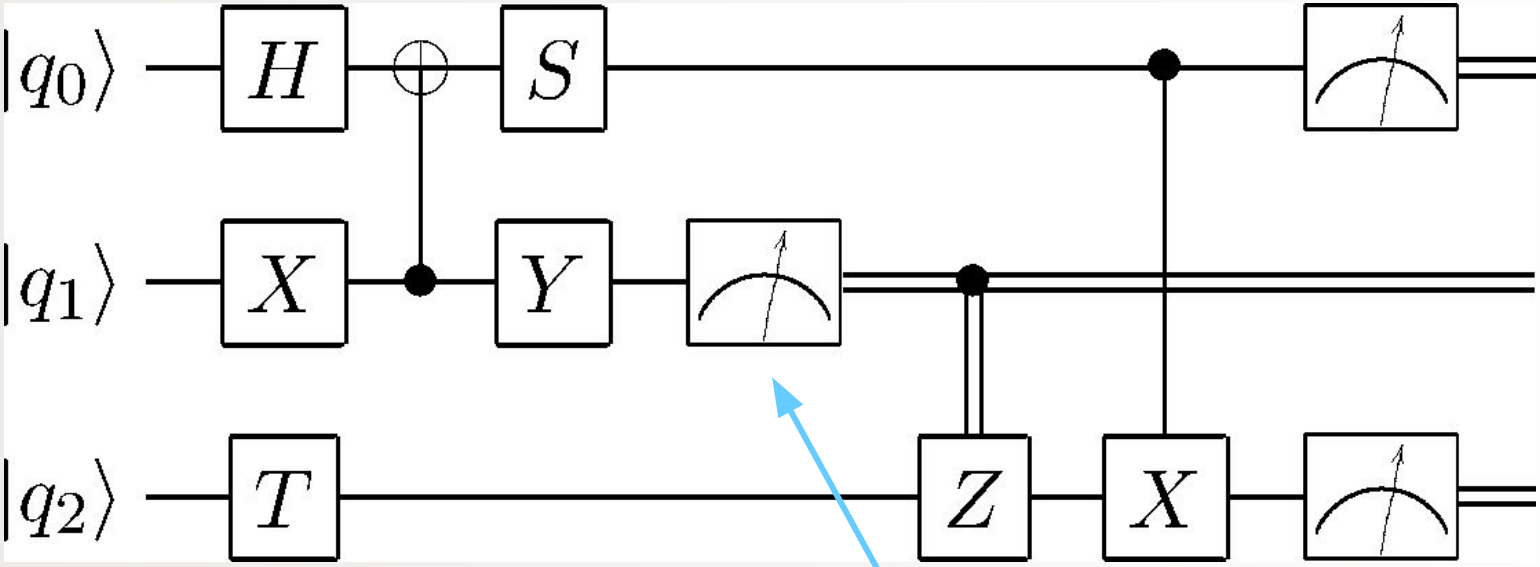
Single-qubit gates

Reading quantum circuit diagrams



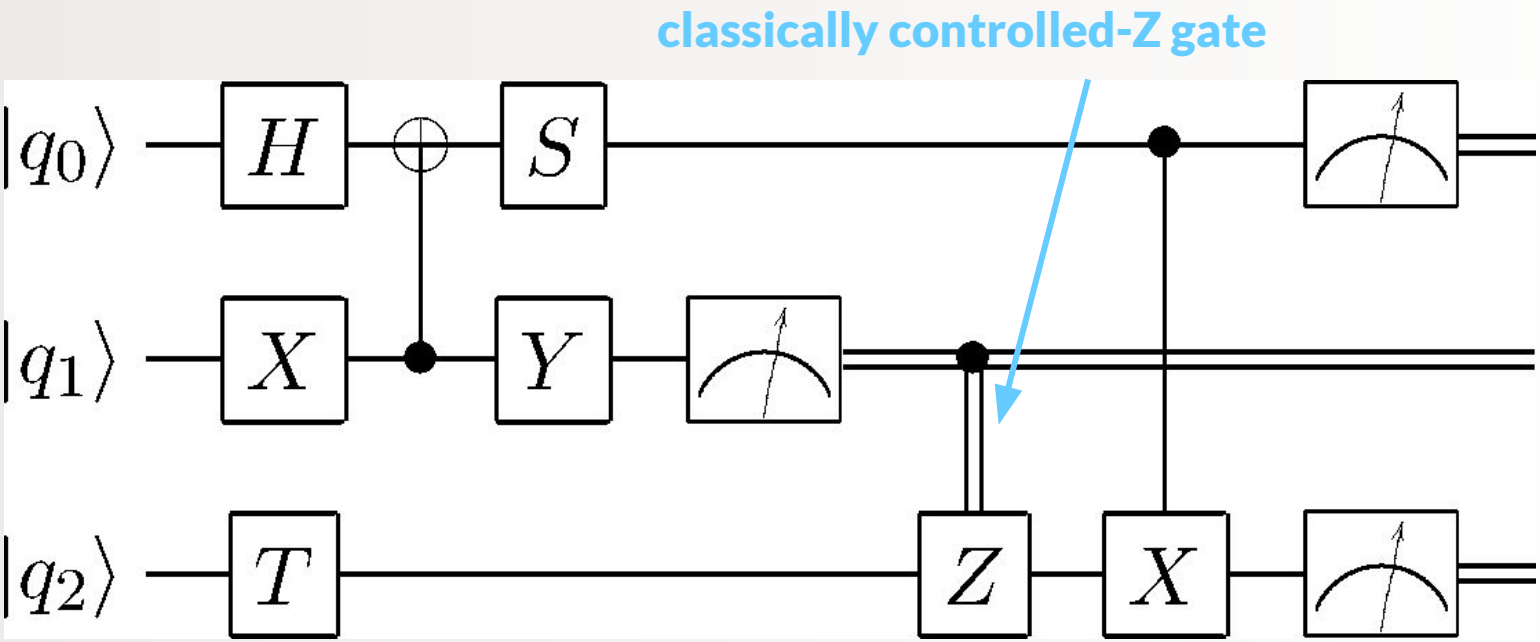
controlled-NOT (CNOT)

Reading quantum circuit diagrams



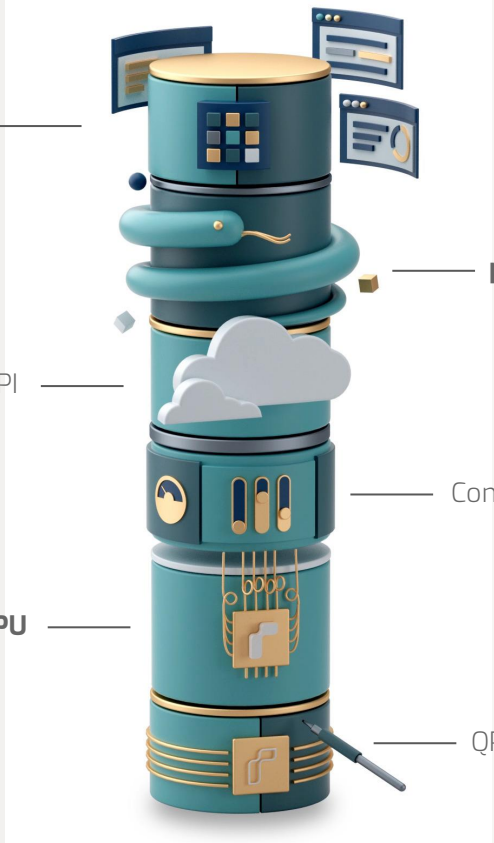
Measurements

Reading quantum circuit diagrams

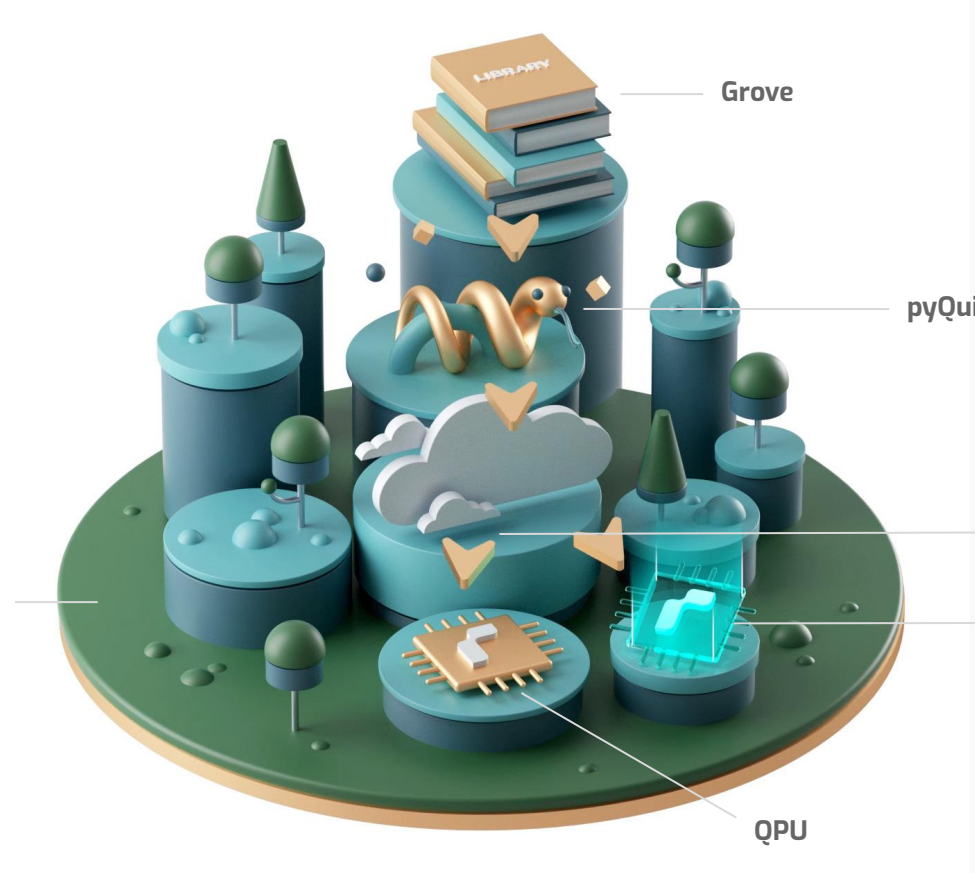


Rigetti's shared memory makes such operations very easy!

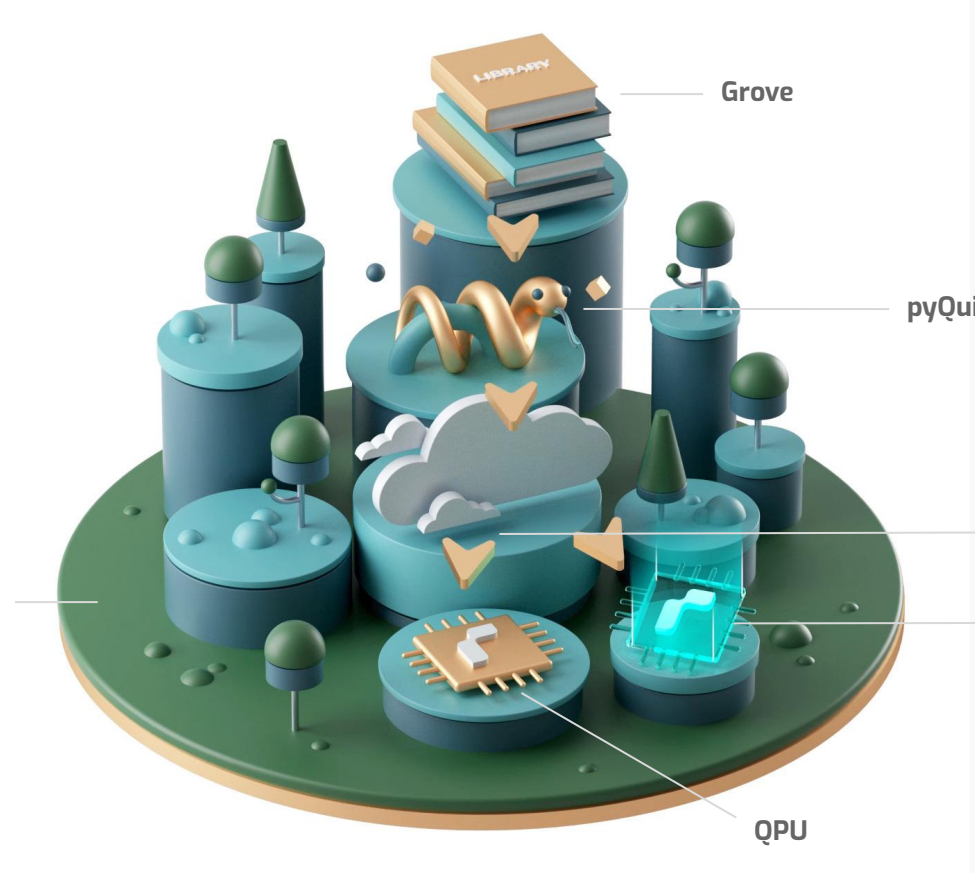
The Rigetti ecosystem



The Rigetti ecosystem



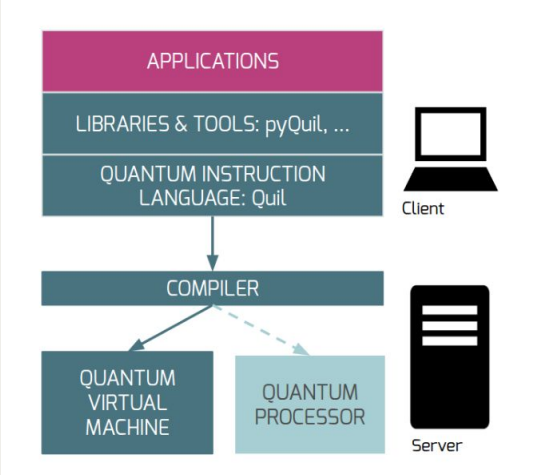
Quantum assembly language (Quil)



Quantum assembly language (Quil)

A Practical Quantum Instruction Set Architecture

Robert S. Smith, Michael J. Curtis, William J. Zeng
Rigetti Computing
775 Heinz Ave.
Berkeley, California 94710
Email: {robert, spike, will}@rigetti.com




```
>>>
H 0
CNOT 0 1
DECLARE ro BIT[2]
MEASURE 0 ro[0]
MEASURE 1 ro[1]

[[1 1]]
```

Enough talking!

Let’s get started with **tutorial #1**.

qosf/SDSS2020_quantum_workshop

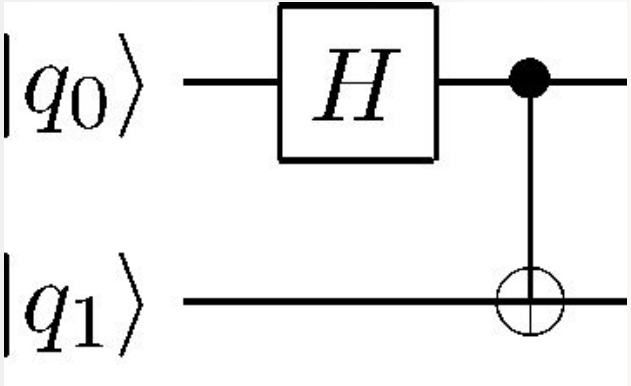
Lecture 2:

Quantum programming

qosf

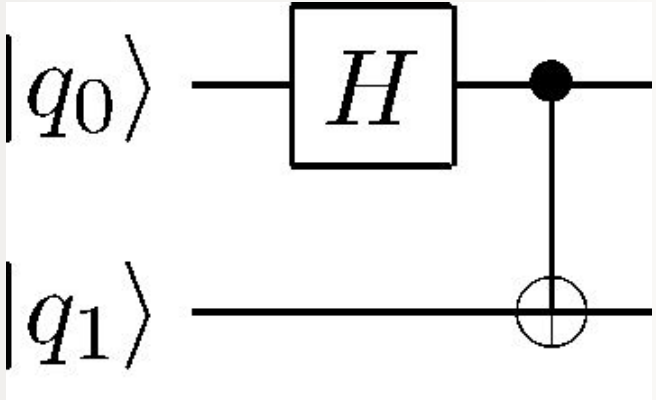
Pen & paper quantum computing

Pen & paper quantum computing



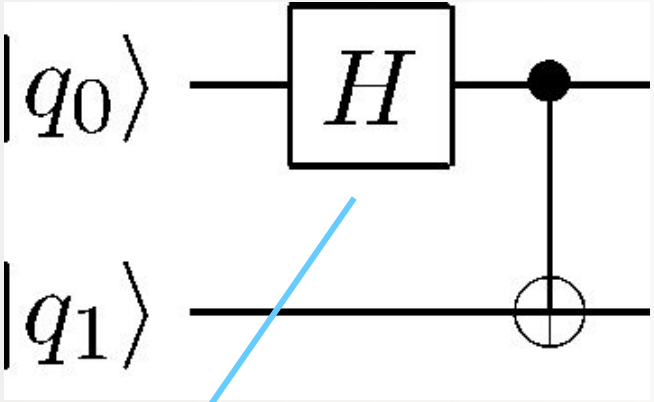
To finish it up, let's put it all together and
walk through a **quantum circuit with pen & paper!**

Pen & paper quantum computing



$|00\rangle$

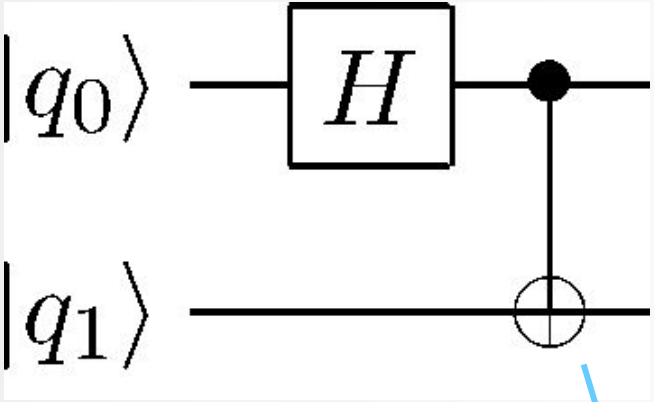
Pen & paper quantum computing



$$|00\rangle \quad \rightarrow \quad (\mathbb{I} \otimes H)\,|00\rangle$$

In a multi-qubit system don't forget the **tensor product with the identity matrix!**

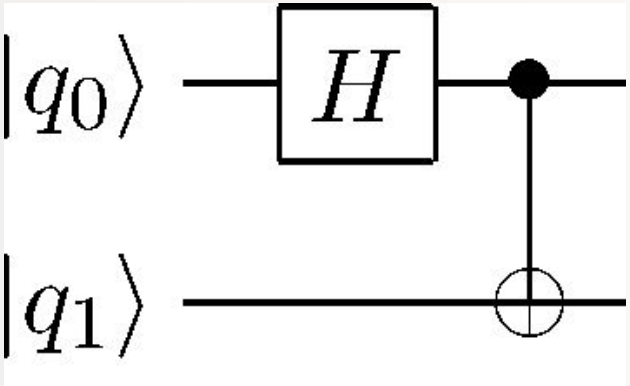
Pen & paper quantum computing



$$|00\rangle \quad \rightarrow \quad (\mathbb{I} \otimes H)\,|00\rangle \quad \rightarrow \quad \mathrm{CNOT}(0,1)(\mathbb{I} \otimes H)\,|00\rangle$$

Pay attention to the **order of operations!**

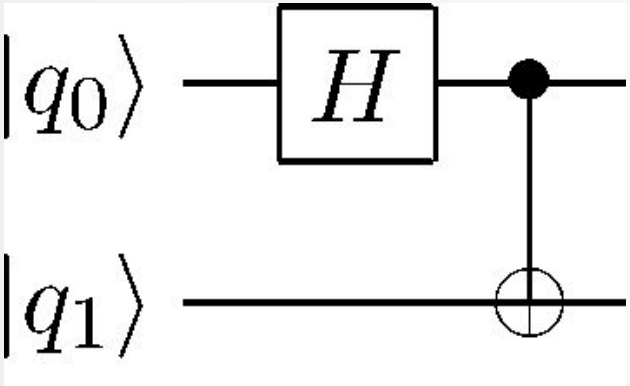
Pen & paper quantum computing



$$\mathrm{CNOT}(0,1)(\mathbb{I} \otimes H)\,|00\rangle = \mathrm{CNOT}(0,1)\left[\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle\right]$$

Apply the Hadamard to the 0th qubit only and don't apply CNOT yet.

Pen & paper quantum computing



$$\mathrm{CNOT}(0,1)(\mathbb{I}\otimes H)\,|00\rangle = \mathrm{CNOT}(0,1)\left[\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle\right]$$

$$= \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle = |\Phi^+\rangle$$

Applying the CNOT results in a maximally entangled state.

It's often called the **first Bell state.**

Step-by-step Quil code

Step by step Quil code

Quantum Processor

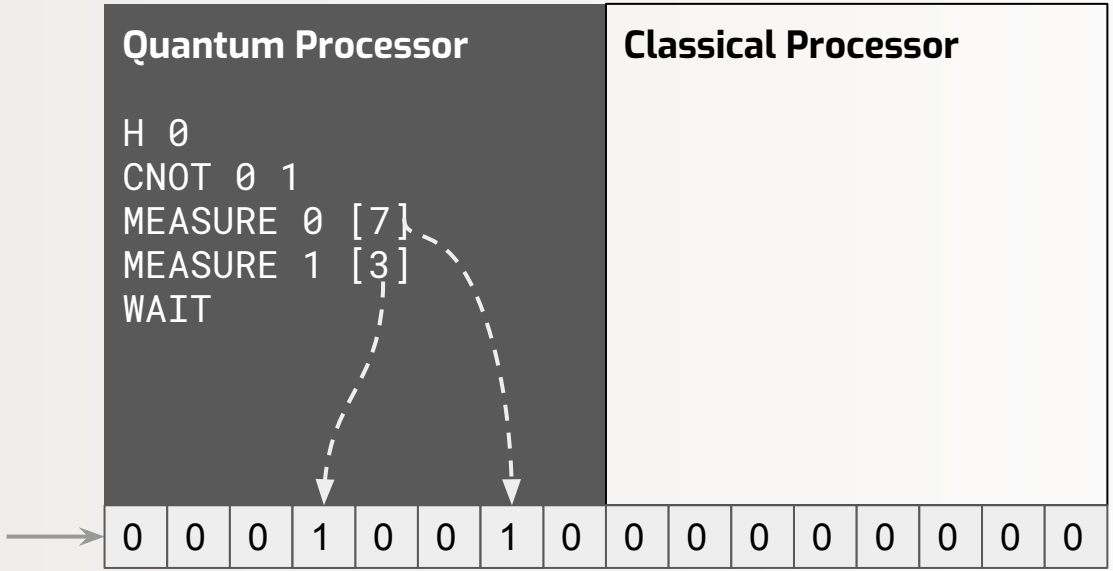
```
H 0
CNOT 0 1
MEASURE 0 [7]
MEASURE 1 [3]
WAIT
```

Classical Processor

C: Classical Shared Memory (bits)

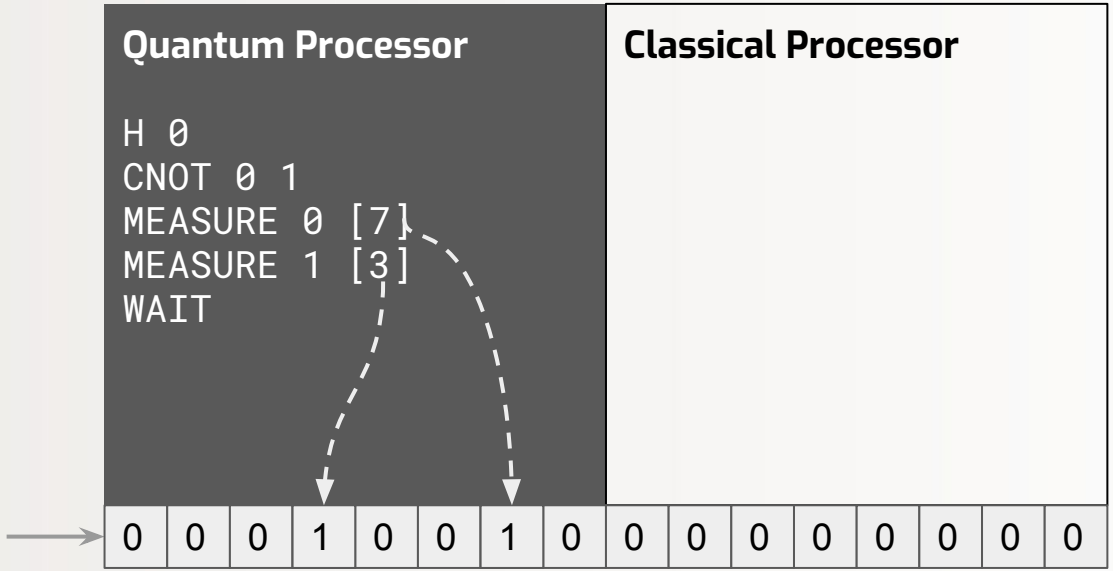
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Step by step Quil code



C: Classical Shared Memory (bits)

Step by step Quil code



C: Classical Shared Memory (bits)

You can define the classical address! It doesn't need to be the qubit index.

Step by step Quil code

Quantum Processor

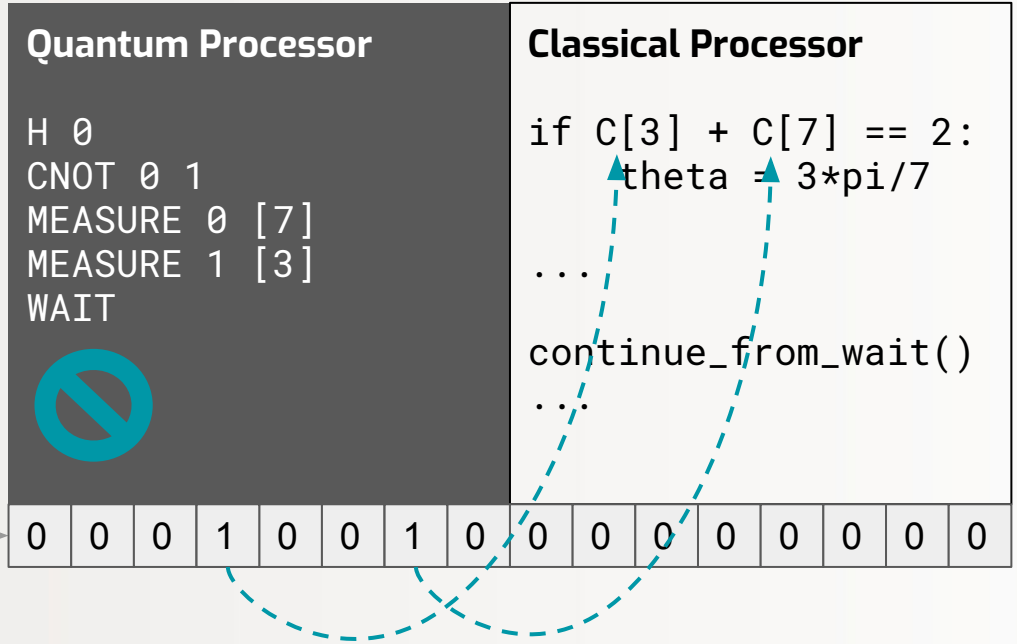
```
H 0
CNOT 0 1
MEASURE 0 [7]
MEASURE 1 [3]
WAIT
```

Classical Processor

C: Classical Shared Memory (bits)

0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0

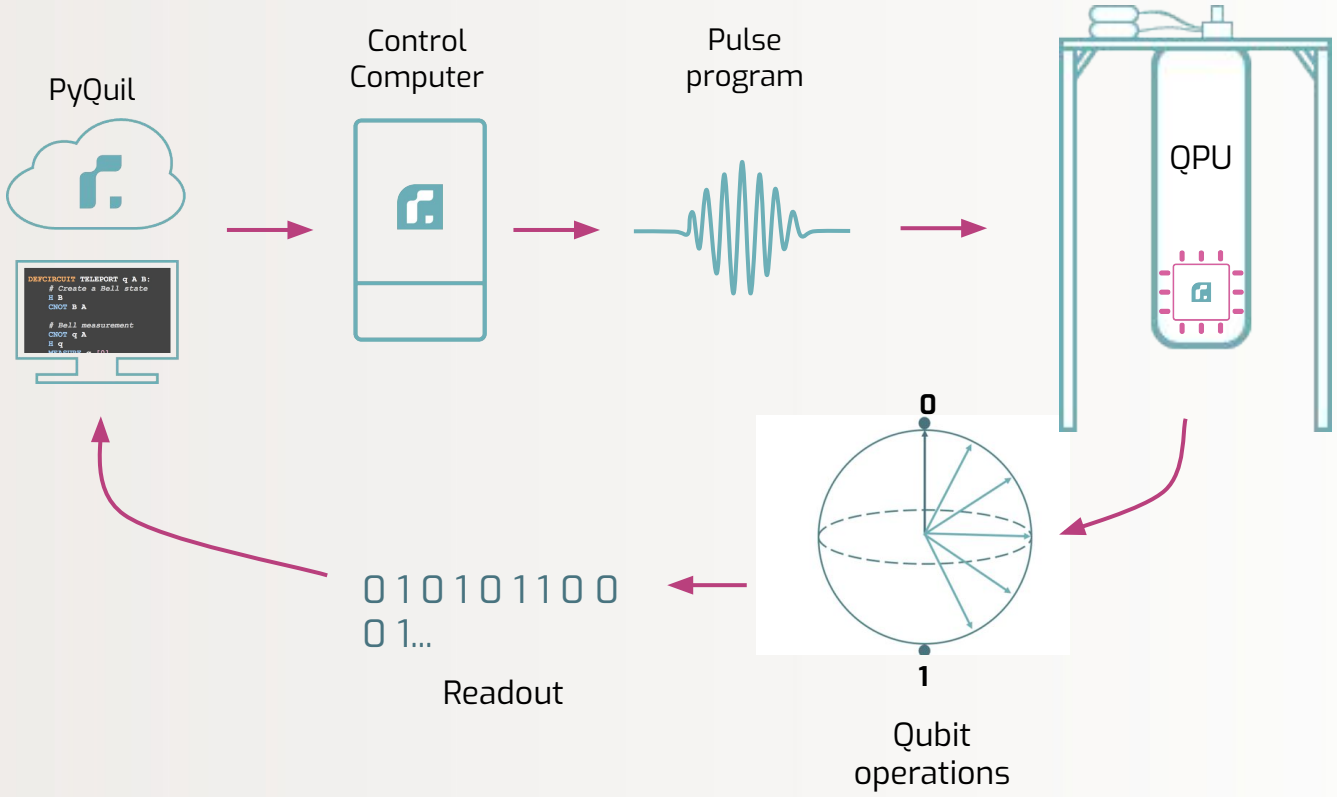
Step by step Quil code



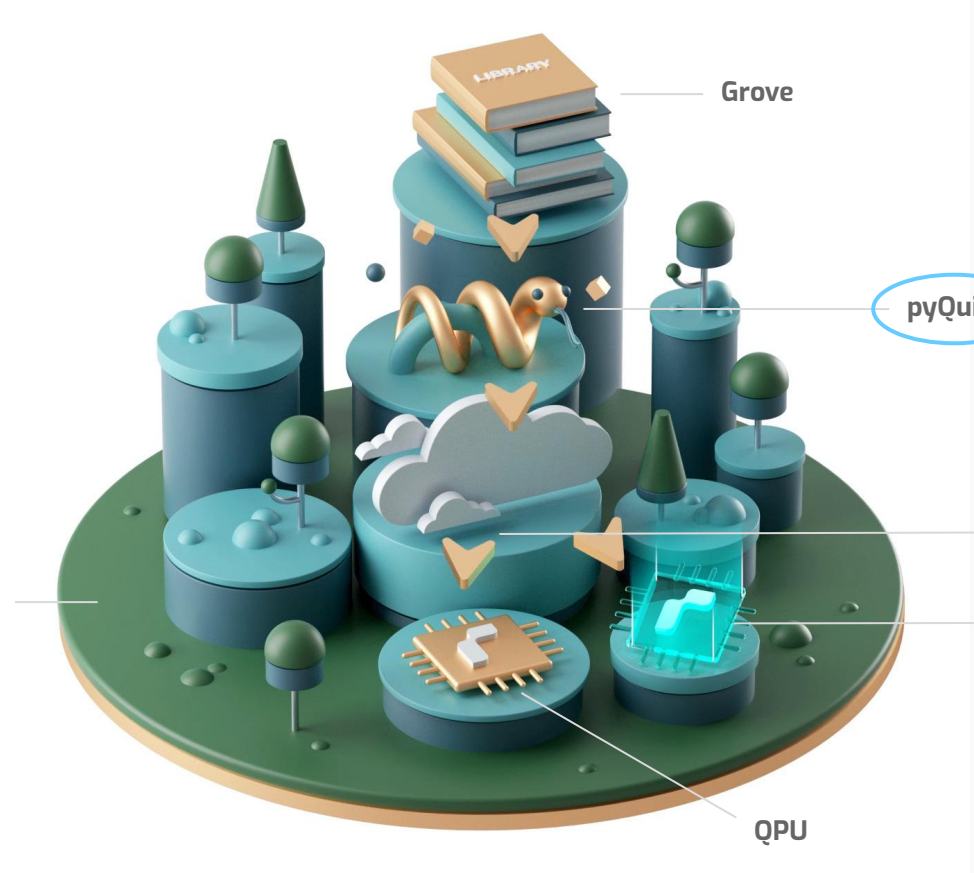
C: Classical Shared Memory (bits)

PyQuil: Python quantum programming

High-level quantum workflow



Rigetti's Python library: PyQuil



PyQuil: The Connection

First step: Establish connection to the simulator server

```
>>> from pyquil.api import ForestConnection

>>> connection = ForestConnection(
    sync_endpoint=http://IP:5000,
    compiler_endpoint=tcp://IP:5555,
)
```

PyQuil: The Quantum Computer

Second step: Instantiate a Quantum Computer object

```
>>> from pyquil.api import ForestConnection
>>> from pyquil import get_qc

>>> connection = ForestConnection(
    sync_endpoint=http://IP:5000,
    compiler_endpoint=tcp://IP:5555,
)
>>> qc = get_qc('2q-qvm', connection=connection)
```

PyQuil: The Program

Third step: Define an empty pyQuil program

```
>>> from pyquil.api import ForestConnection
>>> from pyquil import get_qc
>>> from pyquil.quil import Program

>>> connection = ForestConnection(
    sync_endpoint=http://IP:5000,
    compiler_endpoint=tcp://IP:5555,
)
>>> qc = get_qc('2q-qvm', connection=connection)
>>> p = Program()
```

PyQuil: The quantum circuit

Fourth step: Define a quantum circuit by modifying the Program()

```
>>> from pyquil.api import ForestConnection
>>> from pyquil import get_qc
>>> from pyquil.quil import Program
>>> from pyquil.gates import X, H, CNOT

>>> connection = ForestConnection(...)
>>> qc = get_qc('2q-qvm', connection=connection)
>>> p = Program()
>>> p.inst(H(0), H(1))
```

PyQuil: The quantum circuit

There is **many different ways** of defining programs!

```
>>> p.inst(H(0), H(1))

>>> p.inst(H(0)).inst(H(1))

>>> quil = """
H 0
H 1
"""
>>> p = Program(quil)
```

PyQuil: The quantum circuit

There is **many different ways** of defining programs!

```
>>> p.inst(H(0), H(1))
```

1. In one .inst() call

```
>>> p.inst(H(0)).inst(H(1))
```

2. in multiple .inst() calls

```
>>> quil = """
H 0
H 1
"""
>>> p = Program(quil)
```

3. First define Quil then load into Program()

```
>>> p = Program("H 0\nH 1")
```

4. In one-line as Quil

```
>>> p.inst([H(0), H(1)])
```

5. With a list of gates

```
>>> p.inst(H(i) for i in range(2))
```

6. Using a generator (so cool!)

PyQuil: The classical register

Fifth step: Initialize the classical register!

```
>>> from pyquil.api import ForestConnection
>>> from pyquil import get_qc
>>> from pyquil.quil import Program
>>> from pyquil.gates import X, H, CNOT

>>> connection = ForestConnection(...)
>>> qc = get_qc('2q-qvm', connection=connection)
>>> p = Program()
>>> p.inst(H(0), H(1))
>>> ro = p.declare('ro', 'BIT', 2)
```

PyQuil: The measurement

Sixth step: Measure and save to classical register.

```
>>> from pyquil.api import ForestConnection
>>> from pyquil import get_qc
>>> from pyquil.quil import Program
>>> from pyquil.gates import X, H, CNOT, MEASURE

>>> connection = ForestConnection(...)
>>> qc = get_qc('2q-qvm', connection=connection)
>>> p = Program()
>>> p.inst(H(0), H(1))
>>> ro = p.declare('ro', 'BIT', 2)
>>> p.inst(MEASURE(0, ro[0]))
```

PyQuil: The measurement

Sixth step: Measure and save to classical register.

```
>>> from pyquil.api import ForestConnection
>>> from pyquil import get_qc
>>> from pyquil.quil import Program
>>> from pyquil.gates import X, H, CNOT, MEASURE

>>> connection = ForestConnection(...)
>>> qc = get_qc('2q-qvm', connection=connection)
>>> p = Program()
>>> p.inst(H(0), H(1))
>>> ro = p.declare('ro', 'BIT', 2)
>>> p.inst(MEASURE(0, ro[0]))
```

Quantum register

Classical register

PyQuil: The measurement

There is **many different ways** of measuring!

```
>>> p.inst(MEASURE(0, ro[0]))
```

1. In a .inst() call

```
>>> p.measure(0, ro[0])
```

2. With a .measure() call

```
>>> quil = """
H 0
H 1
MEASURE 0 ro[0]
"""
>>> p = Program(quil)
```

3. Defining it in Quil

```
>>> p = Program("H 0\nH 1\nMEASURE 0 ro[0]")
```

4. In a one-liner as Quil

PyQuil: The measurement

Seventh step: Run the circuit on the Quantum Computer!

```
>>> from pyquil.api import ForestConnection
>>> from pyquil import get_qc
>>> from pyquil.quil import Program
>>> from pyquil.gates import X, H, CNOT, MEASURE

>>> connection = ForestConnection(...)
>>> qc = get_qc('2q-qvm', connection=connection)
>>> p = Program()
>>> p.inst(H(0), H(1))
>>> p.inst(MEASURE(0, ro[0]))
>>> p.inst(MEASURE(1, ro[1]))
>>> results = [qc.run(p) for _ in range(10)]
```

Don't forget measuring qubit 1!

If you use **qc.run()** you will have to **define explicit measurements.**

Quantum *algorithms*

Quantum state preparation

For many quantum algorithms you need to
load data into a quantum computer:

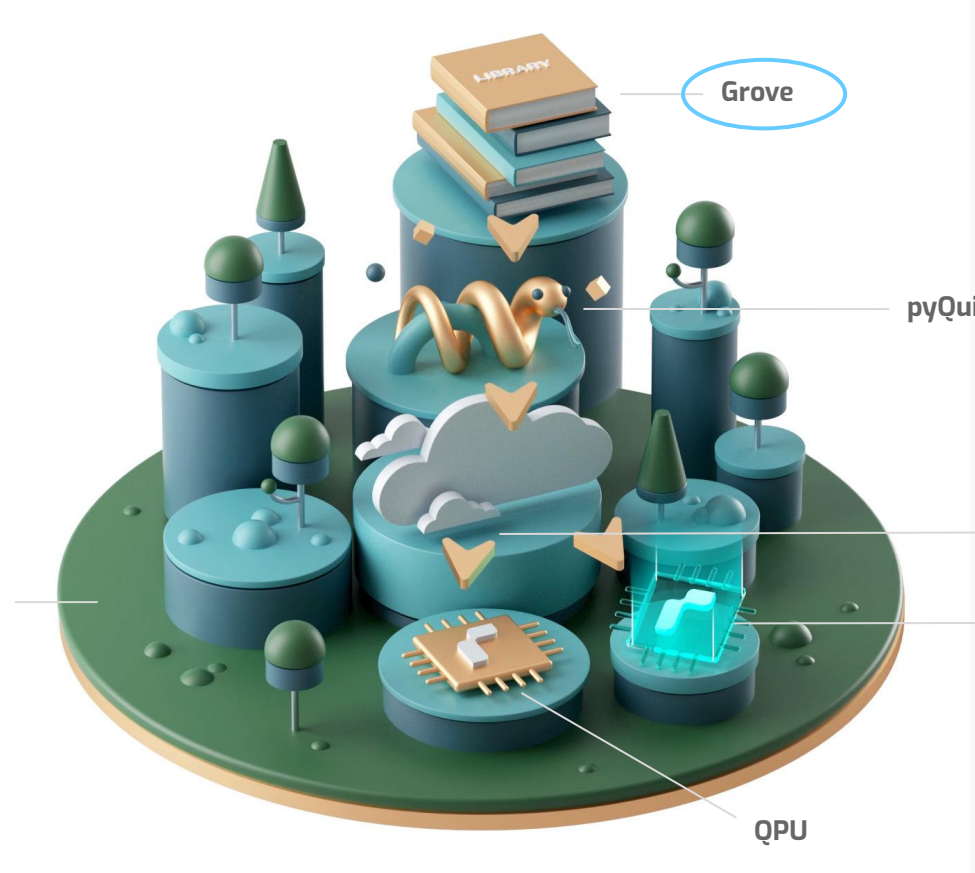
$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \quad \Rightarrow \quad |\Psi\rangle = \sum_{i=0}^{N-1} \frac{a_i}{|\mathbf{a}|} |i\rangle$$

Quantum state preparation

This is a **non-trivial problem** and still an **active field of research**.
In the near future, you might still have to prepare quantum states manually.

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \quad \Rightarrow \quad |\Psi\rangle = \sum_{i=0}^{N-1} \frac{a_i}{|\mathbf{a}|} |i\rangle$$

Quantum algorithms: Grove



Quantum algorithms: Grove

Grove is a library of quantum algorithms implemented in **pyQuil**:

- Variational Quantum Eigensolver (**VQE**)
- Quantum Approximate Optimization Algorithm (**QAOA**)
- Quantum Fourier Transform (**QFT**)
- **Phase Estimation** Algorithm
- **Grover** Search
- Arbitrary **State Generation**

Enough talking!

Let’s get started with **tutorial #2**.

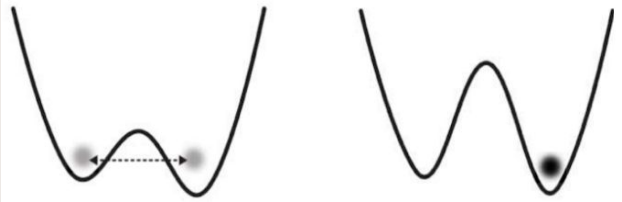
qosf/SDSS2020_quantum_workshop

Lecture 3:

Summary & additional resources

qosf

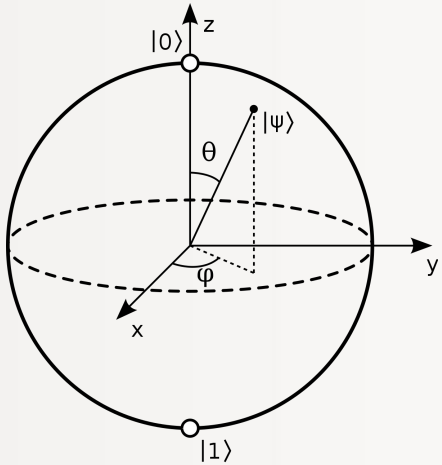
Paradigms of Quantum Computing



Quantum annealing

D:wave
The Quantum Computing Company™

FUJITSU

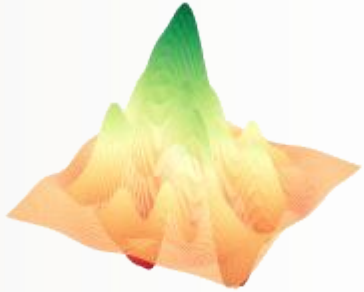


Discrete gate-based

rigetti

IONQ

IBM

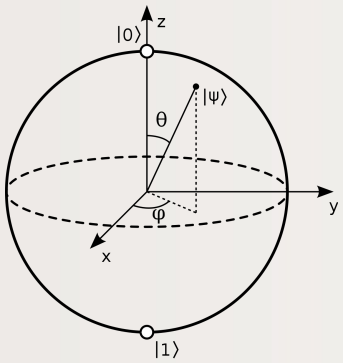


Continuous gate-based

XANADU

A single qubit as a vector

We can write a two-dimensional **qubit** as the vector:



Global phase

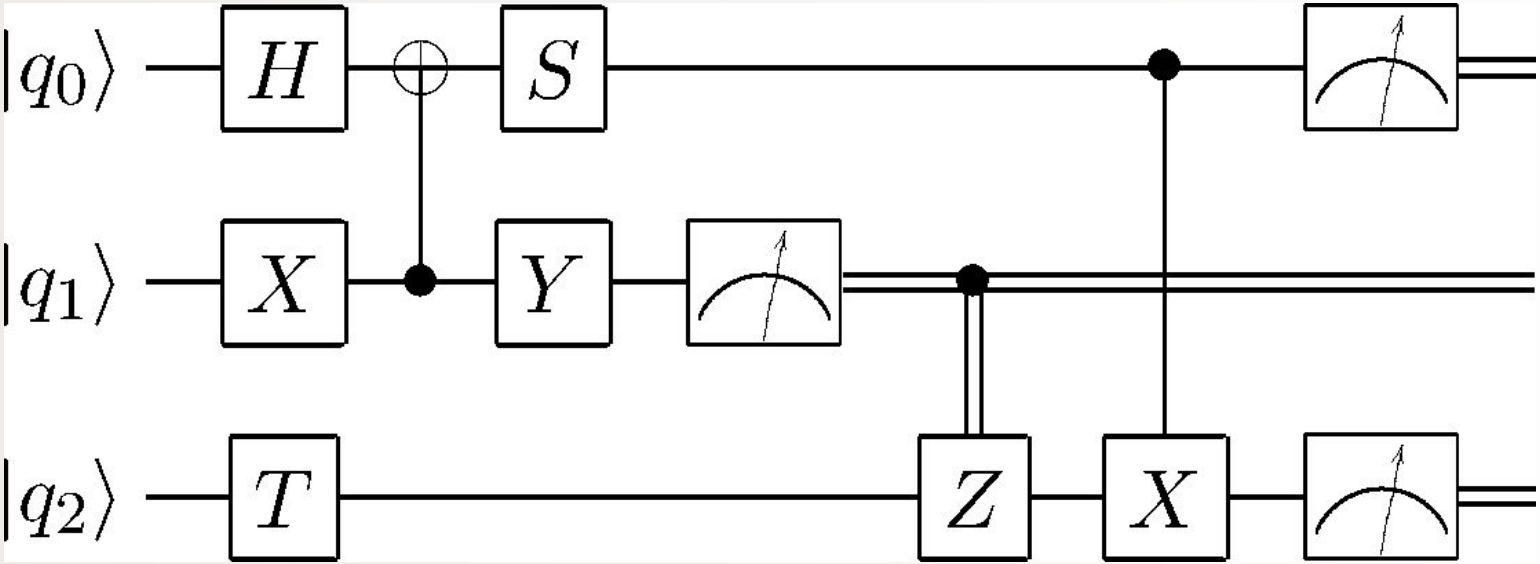
$$|\psi\rangle = e^{i\gamma}\left(\alpha\,|0\rangle + \beta\,|1\rangle\right) = e^{i\gamma}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

Amplitudes

Ground state

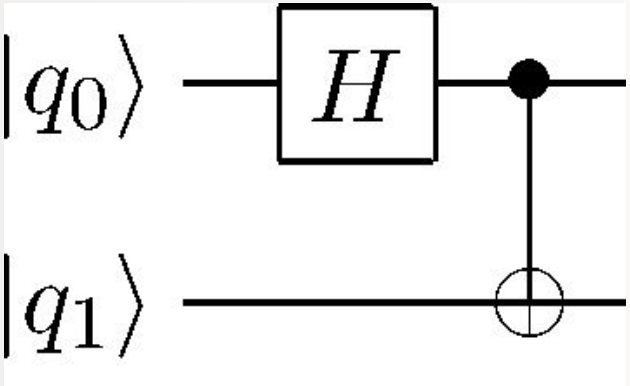
Excited state

Reading quantum circuit diagrams



$|000\rangle$

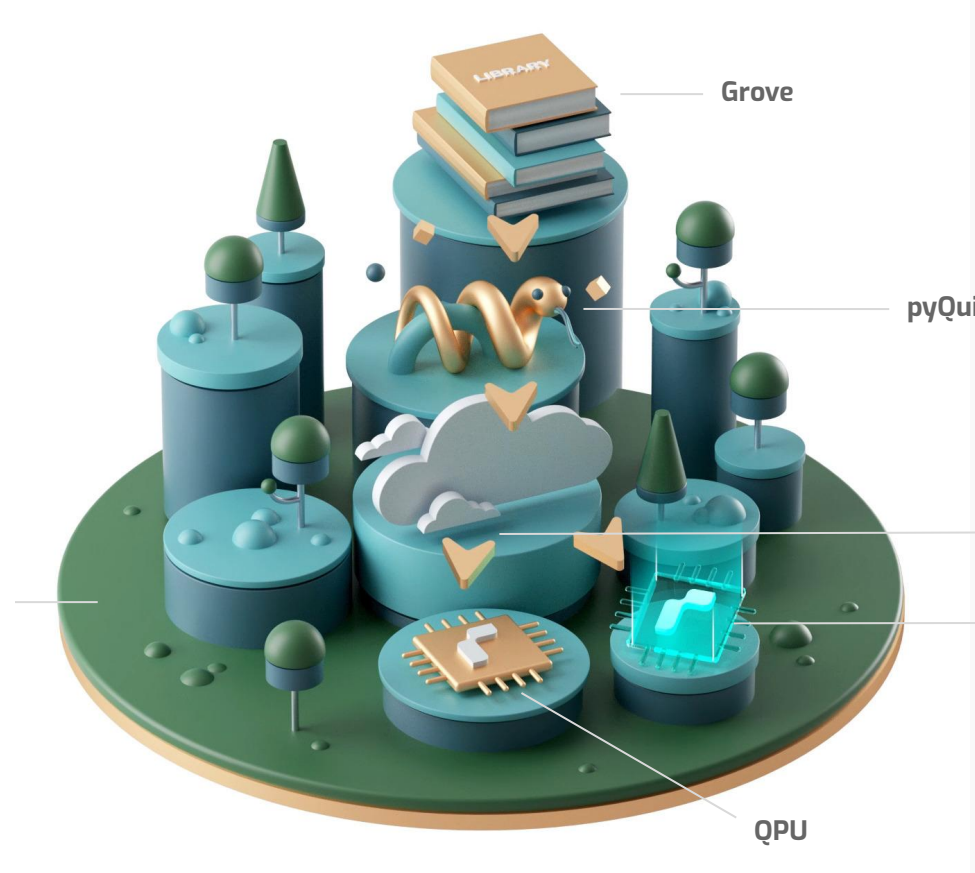
Pen & paper quantum computing



$$\mathrm{CNOT}(0,1)(\mathbb{I}\otimes H)\,|00\rangle = \mathrm{CNOT}(0,1)\left[\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle\right]$$

$$= \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle = |\Phi^+\rangle$$

The Rigetti ecosystem

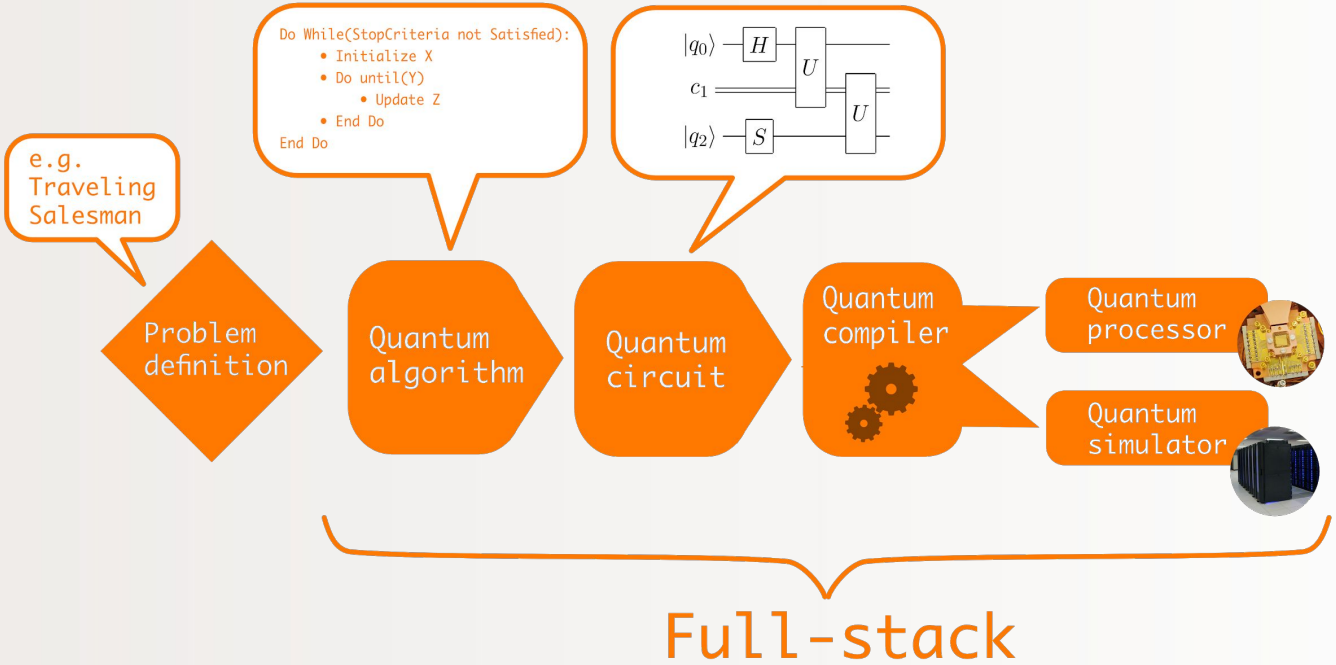


Paradigm: Universal gate-based QC

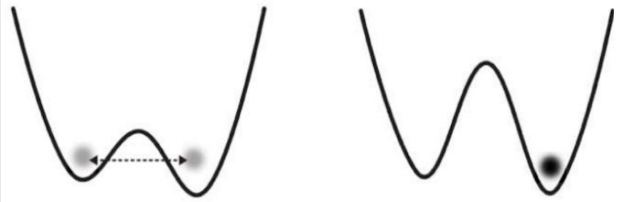
```python
from pyquil import Program
from pyquil.gates import H, CNOT, MEASURE
from pyquil import get_qc

program = Program(H(0), CNOT(0, 1))
classical_register = program.declare('ro', 'BIT', 2)
program += MEASURE(0, classical_register[0])
program += MEASURE(1, classical_register[1])
print(program)

qc = get_qc('2q-qvm')  # We need 2 qubits
executable = qc.compile(program)
result = qc.run(executable)
print(result)
```

```
>>>
H 0
CNOT 0 1
DECLARE ro BIT[2]
MEASURE 0 ro[0]
MEASURE 1 ro[1]

[[1 1]]
```

Gate model workflow



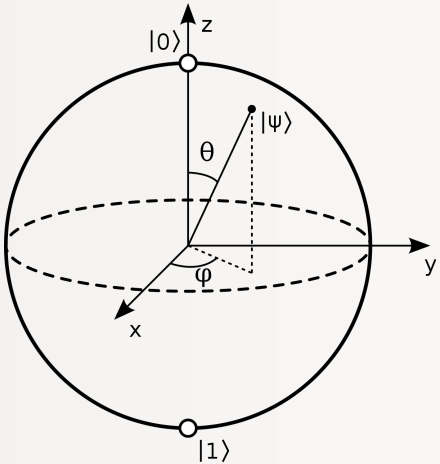
Paradigms of Quantum Computing



Quantum annealing

D-Wave
The Quantum Computing Company™

FUJITSU

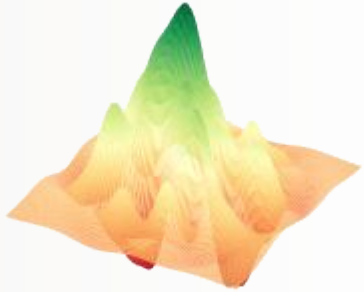


Discrete gate-based

rigetti

IONQ

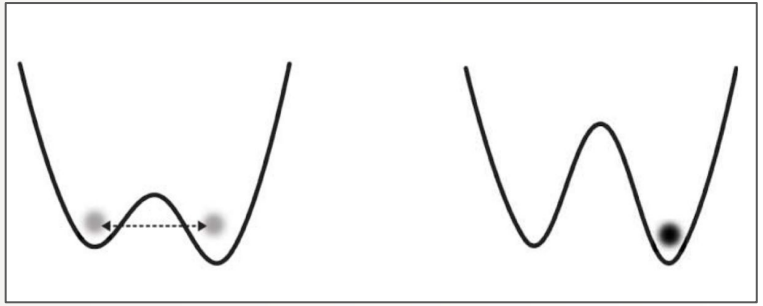
IBM



Continuous gate-based

XANADU

Paradigm: Quantum annealing



Quantum Annealing, D-Wave (2012)
Tunneling helps find optimal solutions

Big players in quantum annealing

D-Wave

The Quantum Computing Company™

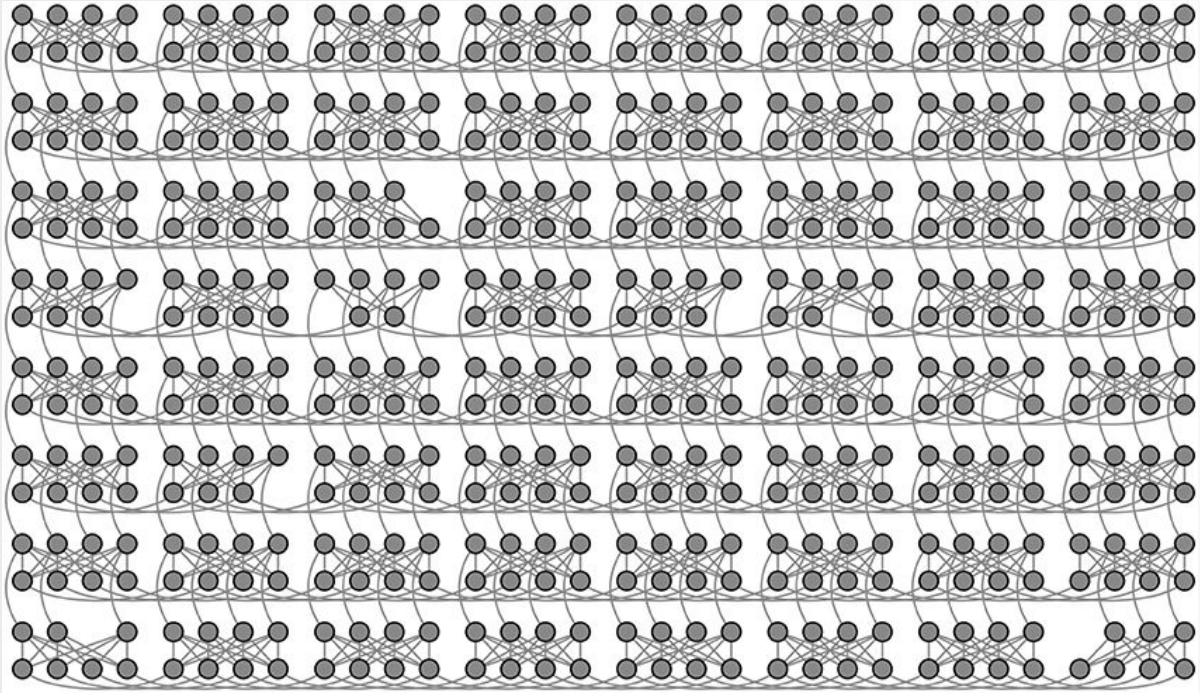
Software: Ocean

Crafting the energy landscape

$$H_0 = -\sum_{i,j} J_{i,j}\sigma_i^z\sigma_j^z - \sum_i h_i\sigma_i^z$$

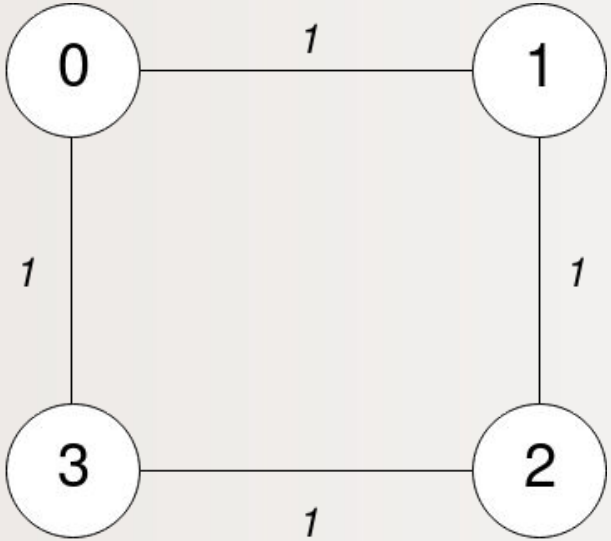
$$H(t) = (1-t)H_{\text{initial}} + tH_{\text{final}}$$

Paradigm: Quantum annealing



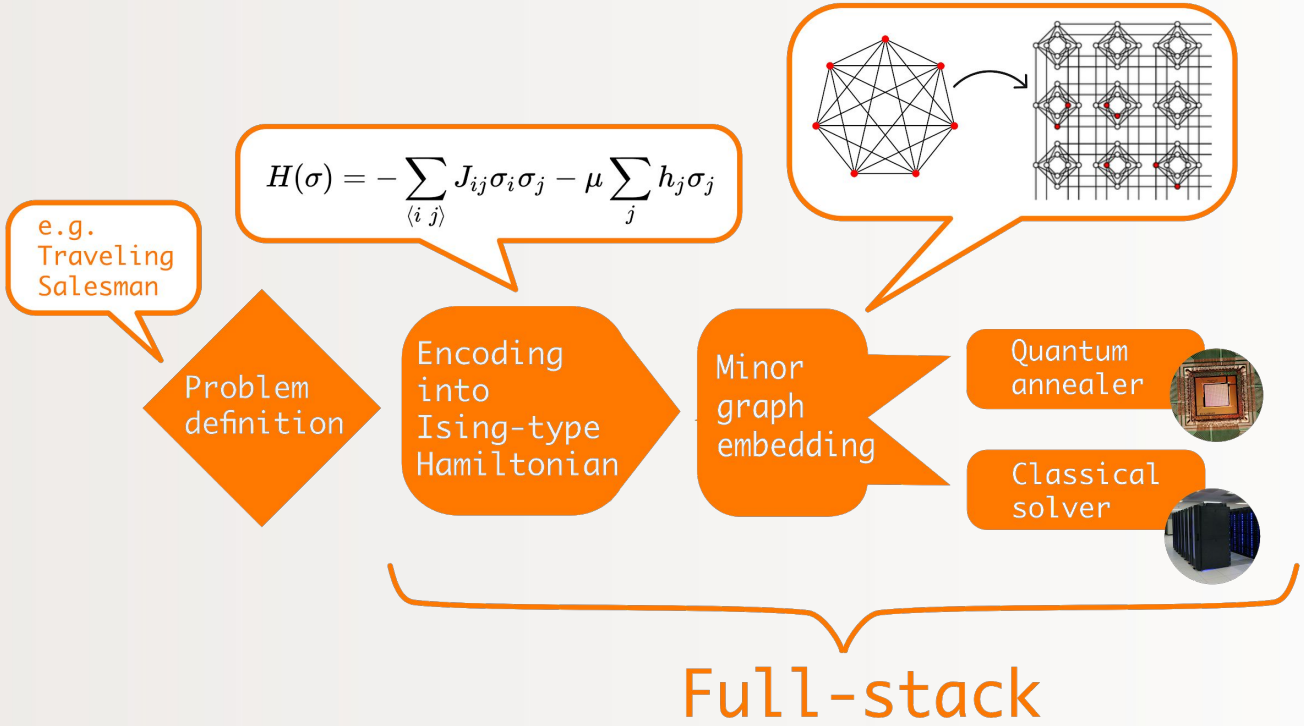
- Binary variables
- Finite resolution couplings and biases
- Classical readout

Example problem: Checkerboard

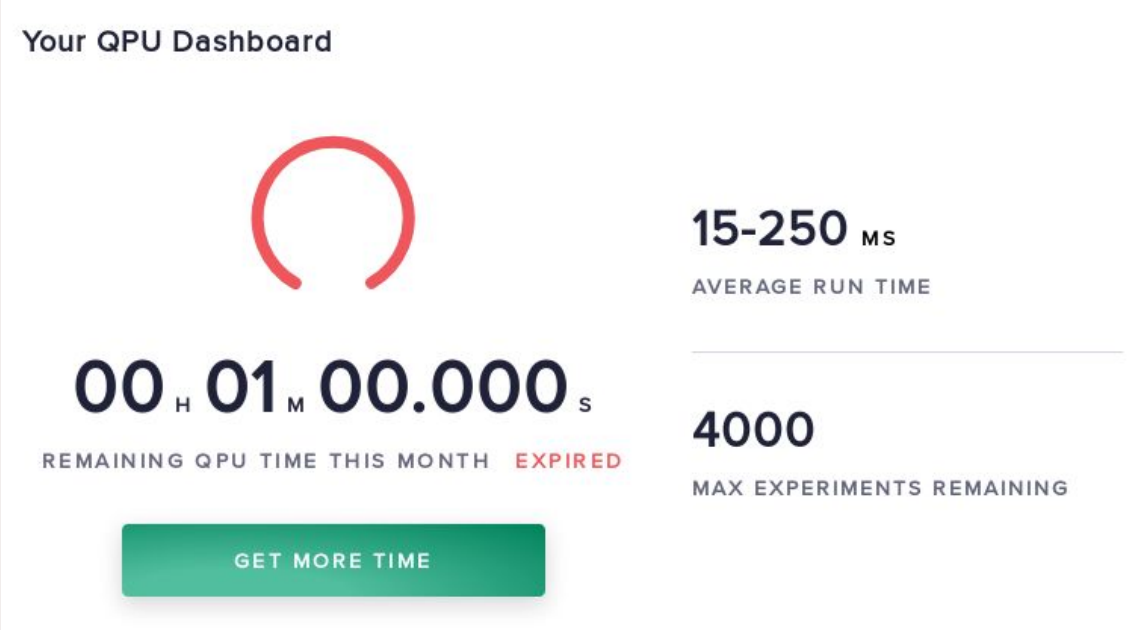



```python
import dimod
from dwave.system.samplers import DWaveSampler
from dwave.system.composites import EmbeddingComposite

h = {}
J = {(0, 1): 1, (1, 2): 1, (2, 3): 1, (3, 0): 1}
bqm = dimod.BinaryQuadraticModel.from_ising(h, J)
sampler = EmbeddingComposite(DWaveSampler())
result = sampler.sample(bqm)

print(result.first)

>>> Sample(sample={0: -1, 1: 1, 2: -1, 3: 1}, ...)
```

Quantum annealing workflow



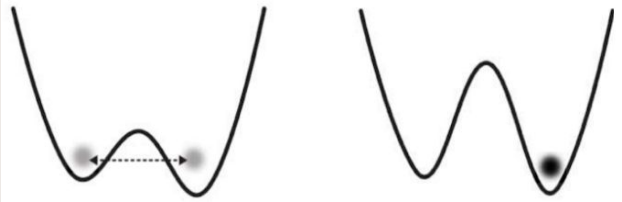
D-Wave Leap


Your QPU Dashboard
00 H 01 M 00.000 S
REMAINING QPU TIME THIS MONTH EXPIRED
GET MORE TIME
15-250 MS
AVERAGE RUN TIME
4000
MAX EXPERIMENTS REMAINING

D-Wave further resources

- Leap portal: https://cloud.dwavesys.com/leap/
- Getting started with Ocean SDK: https://docs.ocean.dwavesys.com/en/latest/overview/install.html
- Quantum annealing Youtube series: https://www.youtube.com/watch?v=zvfkXjzzYOo

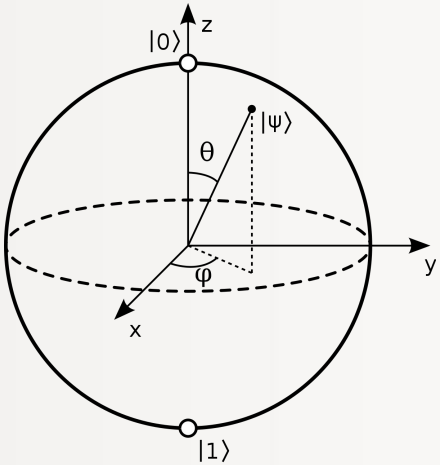
Paradigms of Quantum Computing



Quantum annealing

D:wave
The Quantum Computing Company™

FUJITSU

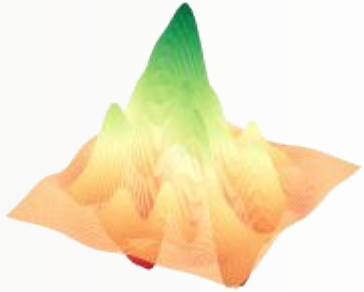


Discrete gate-based

rigetti

IONQ

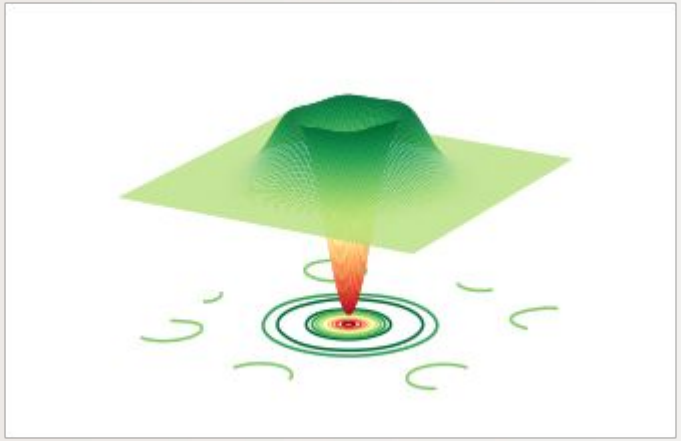
IBM



Continuous gate-based

XANADU

Paradigm: Continuous-variable QC



Qubit	$\lvert\phi\rangle = \phi_0 \lvert 0\rangle + \phi_1 \lvert 1\rangle$
Qumode	$\lvert\psi\rangle = \int dx\, \psi(x) \lvert x\rangle$

Big players in continuous-variable QC

XANADU

Further resources

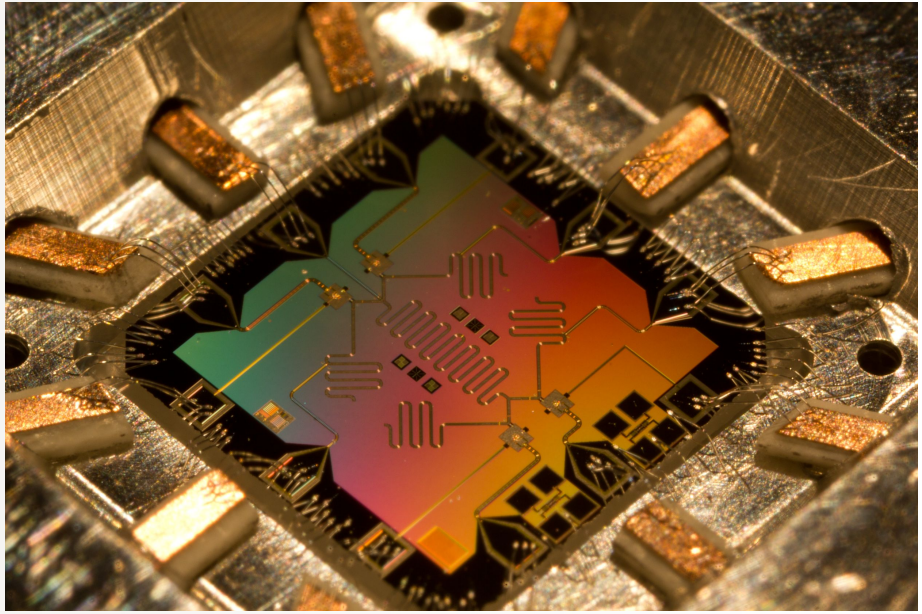
- Xanadu: https://www.xanadu.ai/
- Strawberry Fields: https://strawberryfields.readthedocs.io/en/latest/
  Installation instructions:
- https://strawberryfields.readthedocs.io/en/latest/installing.html

```python
import strawberryfields as sf
from strawberryfields.ops import *

eng, q = sf.Engine(2)

with eng:
  # construct your quantum circuit
  # using blackbird code here

state = eng.run('fock', cutoff_dim=10)
```

Quantum computing metrics

Key measurements of quantum 'horsepower'

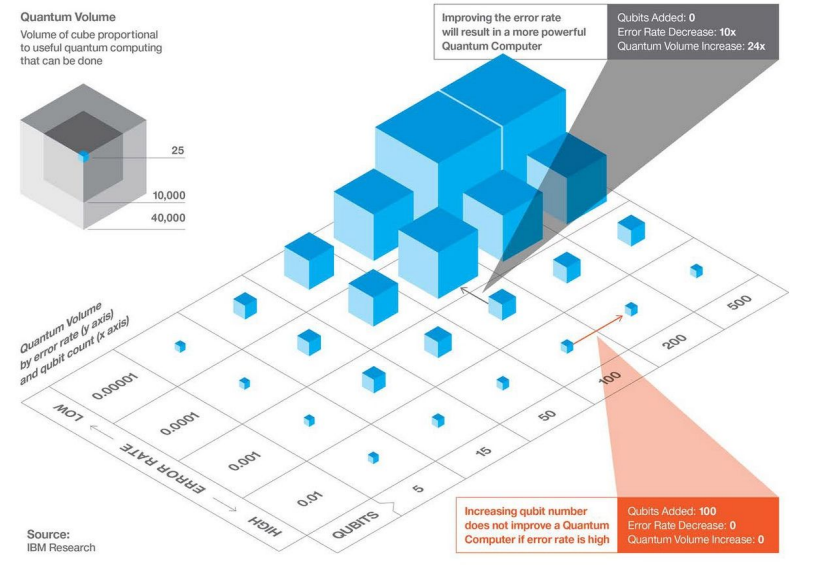
- # of qubits: How big a calculation can we do?
- Connectivity: How connected are the qubits?
- Circuit depth: How many operations can you perform on each qubit?
- Error rates: How likely are the calculations to be correct?



Quantum volume: A better metric?

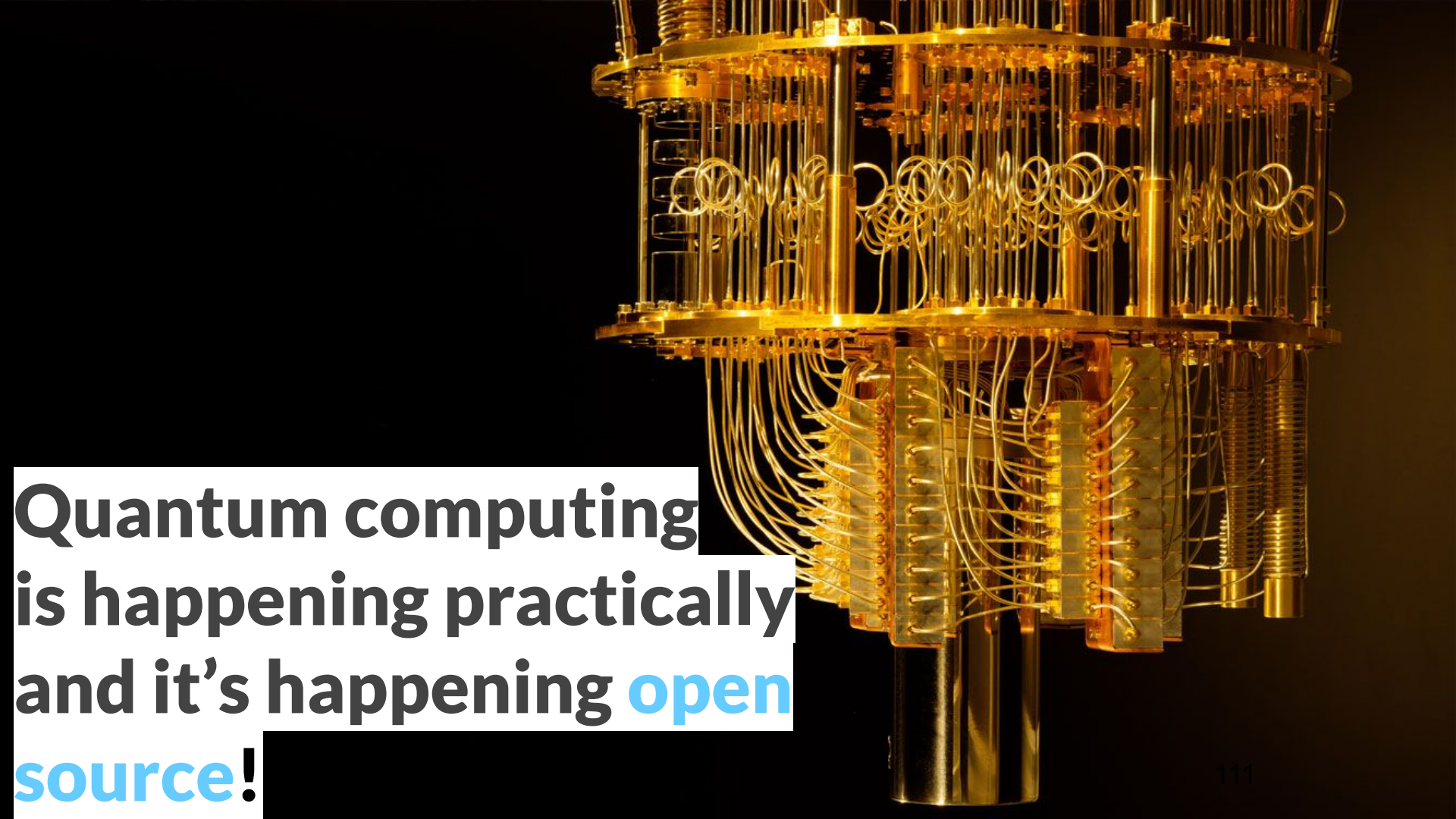
IBM has uses a catch-all term to represent computational power, called "Quantum Volume"

- Combines many metrics into a single number representing computational capability
- Incorporates the ultimate goal of lots of qubits with low errors



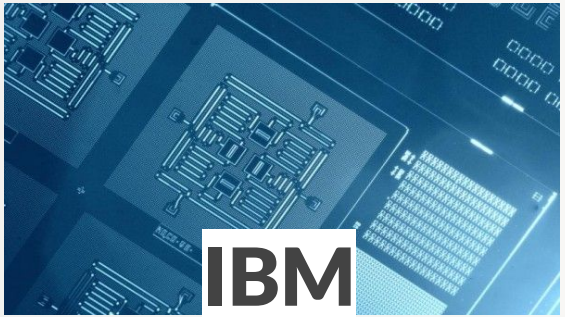
Compare quantum computers by quantum volume and not just the number of qubits!

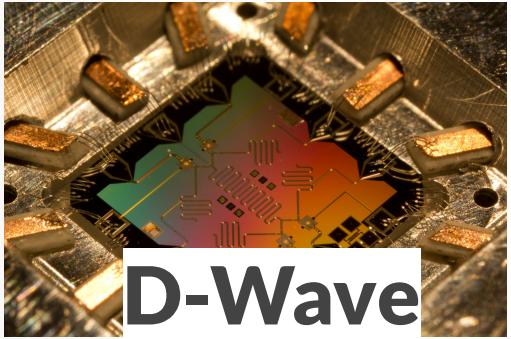
Source: IBM Research

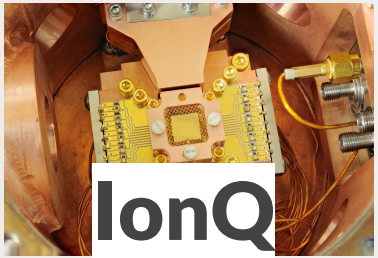


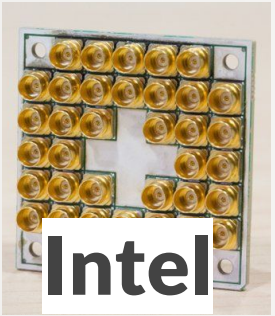
Quantum computing is happening practically and it's happening open source!

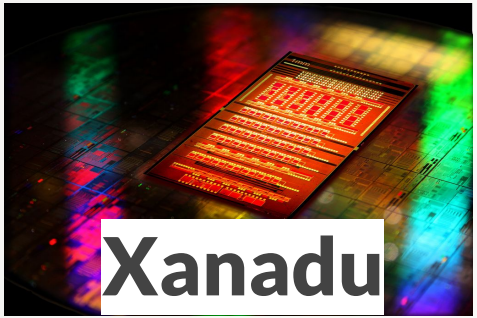
Plenty of quantum hardware (“QPUs”) available today...



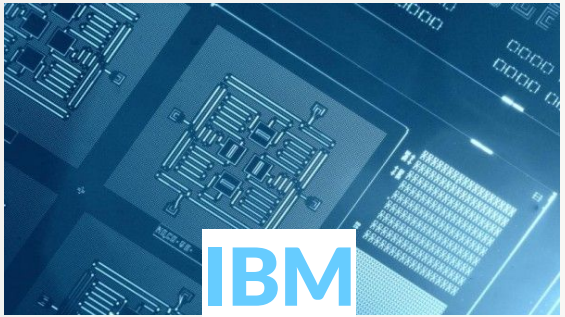


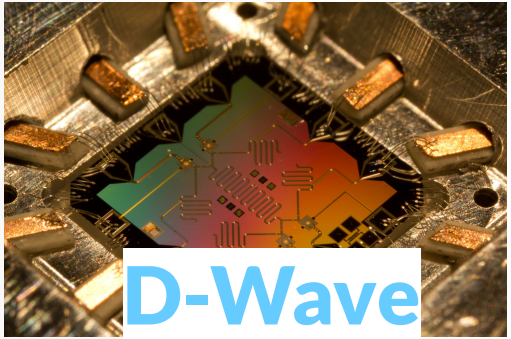


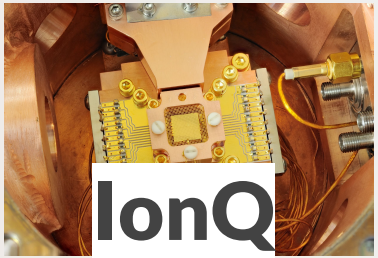


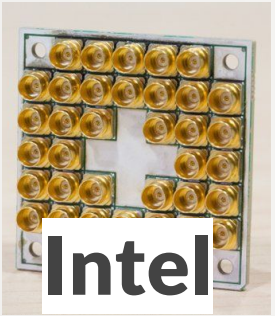


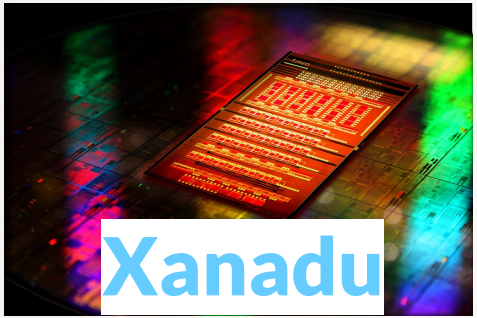
...and most of them have open-sourced!









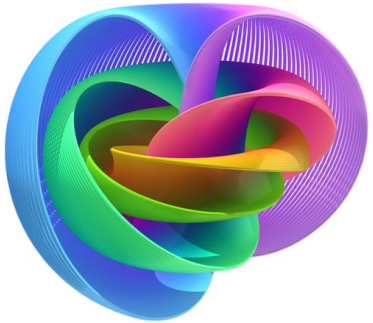


Large amounts of software engineering outstanding to enable real-world applications.

Quantum software overview

qosf/awesome-quantum-software

Unwatch 70 | Unstar 451 | Fork 91



Open-Source Quantum Software Projects

Curated list of open-source developed quantum software projects.

Please read the contribution guidelines before contributing.

Also please check out the Unitary Fund's 4k$ grant project for quantum OSS!

Contents

- Quantum full-stack libraries
- Quantum simulators
- Quantum annealing
- Quantum algorithms
- Quantum compilers
- Quantum assembly
- Quantum and post-quantum cryptography
- Experimental quantum computing
- Quantum fun
- Quantum tools
- Abandoned projects

Further resources

- Amazing learning resource for beginners by the author of Nielsen & Chuang (https://quantum.country/)
- Quantum Computing Stack Exchange (https://quantumcomputing.stackexchange.com/)
- Free QC learning resources (https://qosf.org/learn_quantum/)

https://qosf.org/join

https://qosf.org

@qosfoundation

https://github.com/qosf

@qosfoundation

https://gitlab.com/qosf

@quantumOSfoundation

This tutorial was brought to you by:

/m-fingerhuth

@mark_fingerhuth