

QISKit

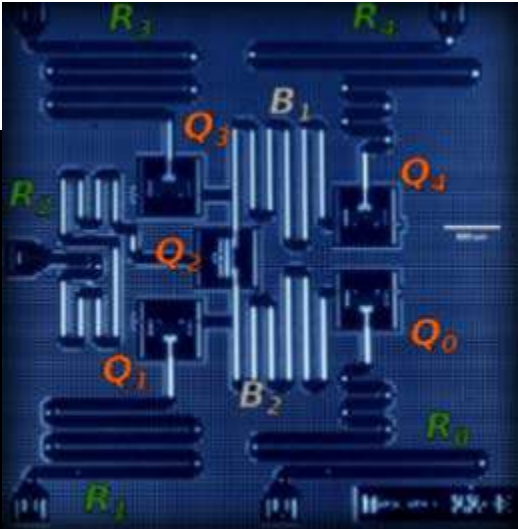
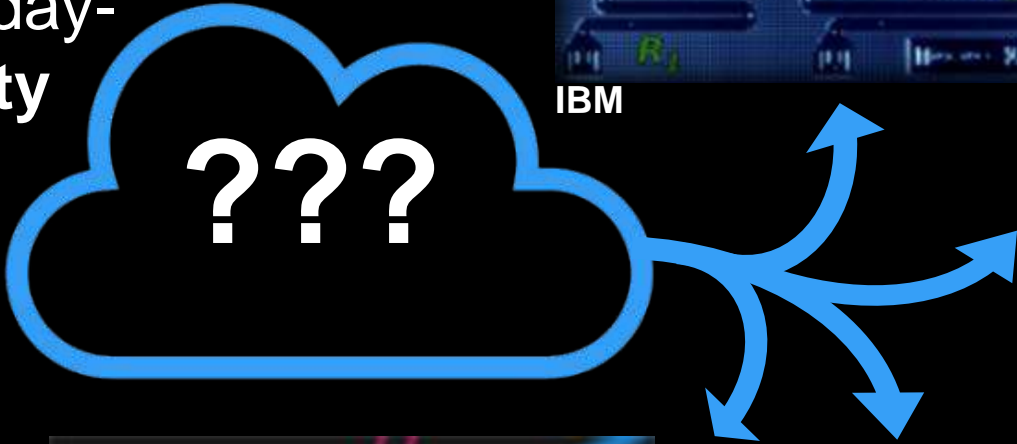
Jay Gambetta



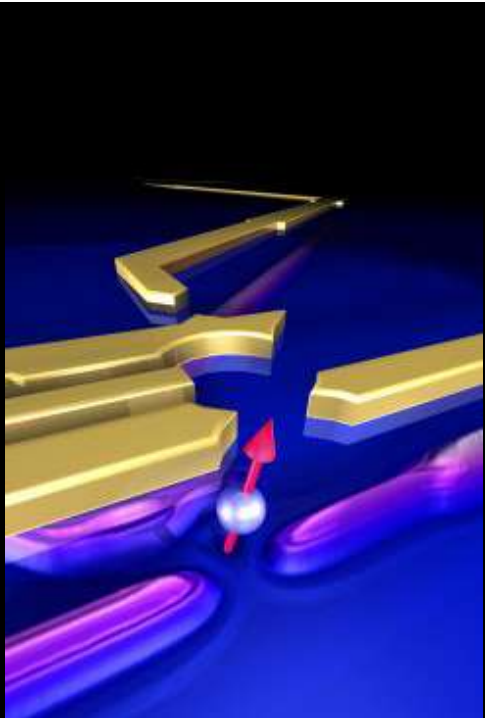
Exploring Quantum Possibilities

Quantum hardware is **evolving rapidly**, with **many approaches** being pursued

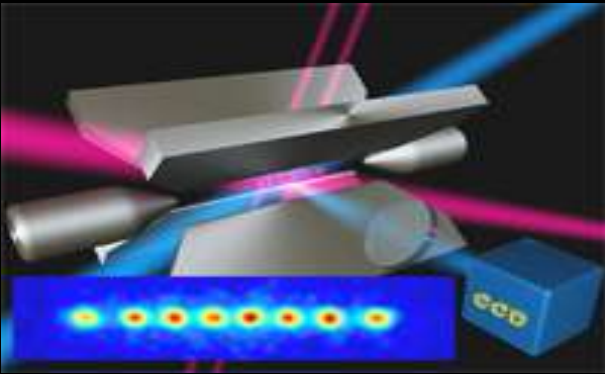
Even individual qubits exhibit day-to-day **performance variability**



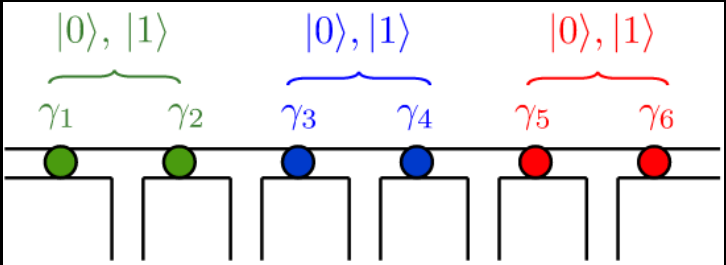
IBM



<http://vandersypenlab.tudelft>



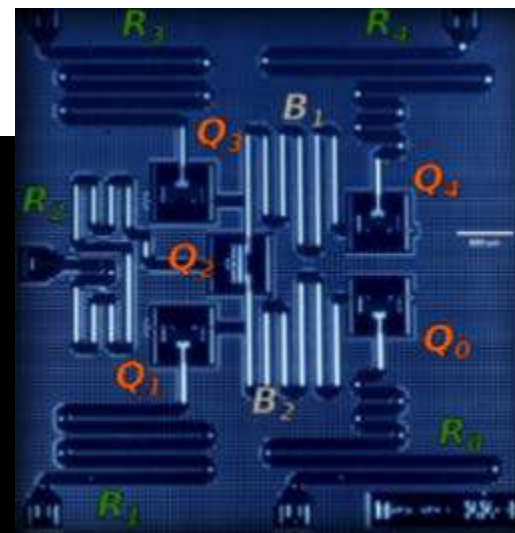
<http://www.quantumoptics.at>



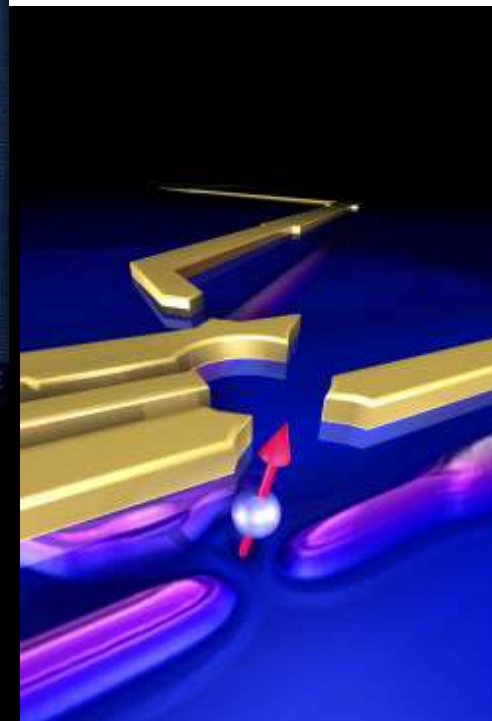
http://topocondmat.org/w2_majorana/braiding.html

Exploring Quantum Possibilities

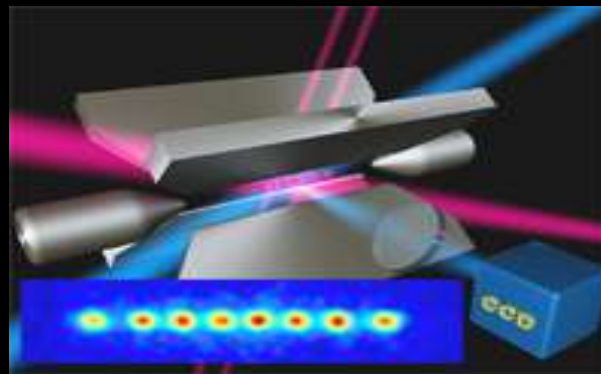
QISKit goal is to provide an open-source platform for building quantum programs that can **keep up with changing hardware**



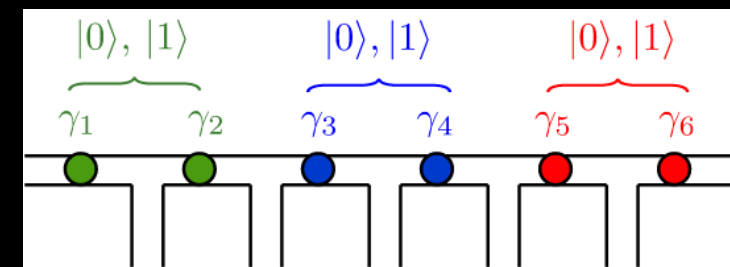
IBM



<http://vandersypenlab.tudelft.nl>



<http://www.quantumoptics.at>



http://topocondmat.org/w2_majorana/braiding.html



Quantum Information Software **Kit**.

Is an open source software development kit for writing quantum computing experiments, programs, and applications.



This organization

Search

Pull requests

Issues

Marketplace

Explore



QISKit

Quantum Information Software Kit

<http://www.qiskit.org>

qiskit@qiskit.org

Repositories 9

People 33

Teams 4

Projects 0

Settings

Pinned repositories

[Customize pinned repositories](#)

[qiskit-sdk-py](#)

Software development kit for writing quantum computing experiments, programs, and applications.

Python ★ 1.1k 🍴 310

[qiskit-tutorial](#)

A collection of Jupyter notebooks using QISKit

Jupyter Notebook ★ 173 🍴 75

[openqasm](#)

Gate and operation specification for quantum circuits

TeX ★ 152 🍴 42

[ibmqx-backend-information](#)

Information about the different backends on the IBM Q experience

★ 54 🍴 26

[ibmqx-user-guides](#)

The users guides for the IBM Q experience

HTML ★ 28 🍴 18



This repository

Search

Pull requests

Issues

Marketplace

Explore



QISKit / qiskit-sdk-py

Unwatch

172

Unstar

1,112

Fork

310

Code

Issues 18

Pull requests 7

Projects 0

Wiki

Insights

Settings

Software development kit for writing quantum computing experiments, programs, and applications. <http://www.qiskit.org>

Edit

quantum-computing

qiskit

sdk

python

quantum-programming-language

Manage topics

1,175 commits

2 branches

20 releases

42 contributors

Apache-2.0

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download



atilag Bug fix: Remove static compilation of the simulator because it's

Latest commit 0a312fe an hour ago

.github

Add templates for issues and pull requests

6 months ago

doc

Add change log for release 0.4.0. (#229)

6 hours ago

examples

Style and fixes for initializer

13 days ago

images

Update Sphinx theme color and diagrams

3 months ago

qiskit

Merge pull request #194 from chriselectic/cpp-simulator

6 hours ago

src/cpp-simulator

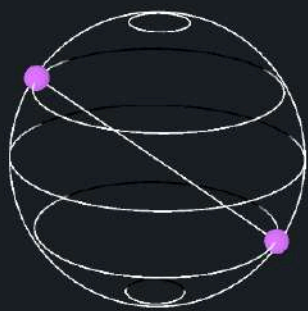
Bug fix: Remove static compilation of the simulator because it's

an hour ago

test

Merge pull request #194 from chriselectic/cpp-simulator

6 hours ago



QISKit

Quantum Information Software Kit

Join our Slack community

Approximate Quantum Computing: From advantage to applications
Recordings now available!

Latest version pypi v0.4.3

The Quantum Information Software Kit (QISKit for short) is a software development kit (SDK) for working with OpenQASM and the IBM Q experience (QX).

GitHub

Road map

Learn

Use QISKit to create quantum computing programs, compile them, and execute them on one of several backends (online Real quantum processors, and simulators).

Tutorials

Documentation

IBM Q experience

Run a quantum program

```
[python3] $ pip install qiskit
```

```
from qiskit import QuantumProgram
qp = QuantumProgram()
qr = qp.create_quantum_register('qr', 2)
cr = qp.create_classical_register('cr', 2)
qc = qp.create_circuit('Bell', [qr], [cr])
qc.h(qr[0])
qc.cx(qr[0], qr[1])
qc.measure(qr[0], cr[0])
qc.measure(qr[1], cr[1])
result = qp.execute('Bell')
print(result.get_counts('Bell'))
```

Python 3.5+ required, see more **in the docs**

- # device-file-schema
- # documentation
- # general
- # ibmqx5
- # latex_drawer
- # openpulse
- # openqasm
- # qobj-interface
- # random
- # release0_4
- # sdk
- # simulators
- # sympy-backends
- # tools
- # transpiler
- # tutorials

Direct Messages

- slackbot
- jaygambetta (you)
- apurva
- Diego Moreda
- erick winston

#release0_4

8 | 0 | to discuss the 0.4 release

Wednesday, January 3rd



Jay Gambetta

@Chris Wood it looks good to me but i agree with comments @Juan Gómez made do you think you can address them today

Posted in #release0_4 | Jan 3rd at 10:15 AM

@Juan Gómez @erick winston do you know if there is anyway in the build to get it to try and build the simulator with gcc if available on macOS, and fall back to xcode/clang? (the gcc I use personally is the commented out g++-7). The gcc build is needed to use all the multithreading / parallelisation options

Thursday, January 4th



Juan Gómez 6:00 AM

@Chris Wood yep, I think we could try different compilers in the Makefile before building, I'll take a look



Andreas 7:15 PM
joined #release0_4.

Today

new messages



Juan Gómez 9:57 AM

Release 0.4 is out! 😊



Our next release will be 0.5, so the master branch points to it.



Message #release0_4





Why is quantum computing exciting?

Why Now?

To solve problems that are intractable for classical computers

Fault-tolerant vs approximate quantum computing

What does QC look like today?

IBM Q superconducting hardware

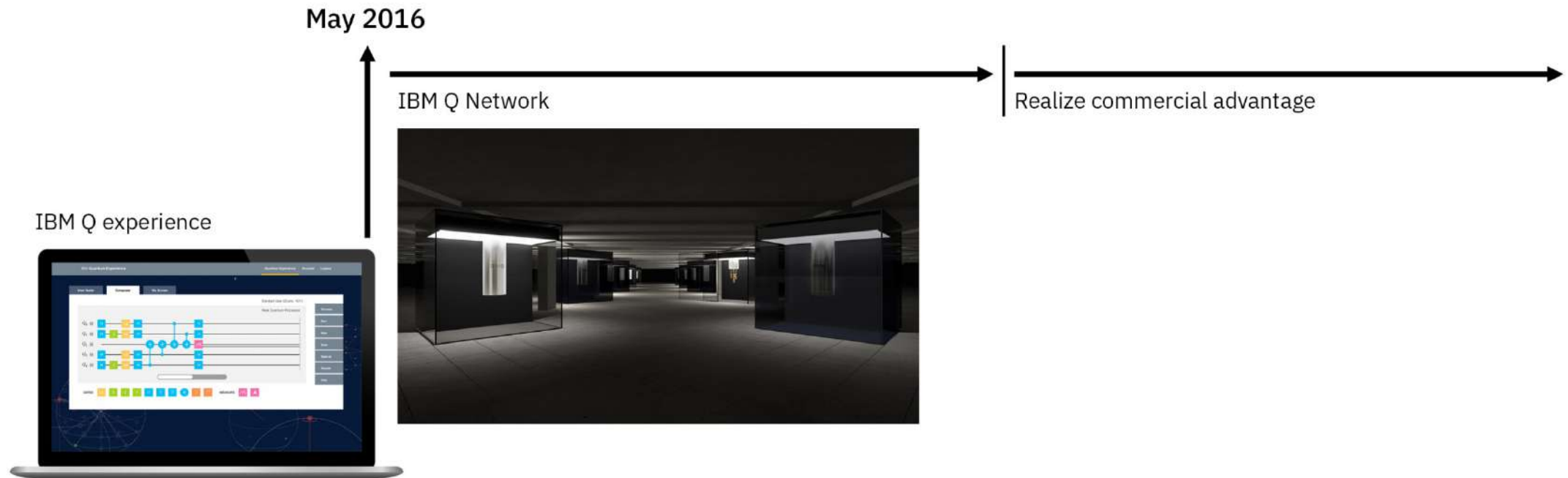
QISKit

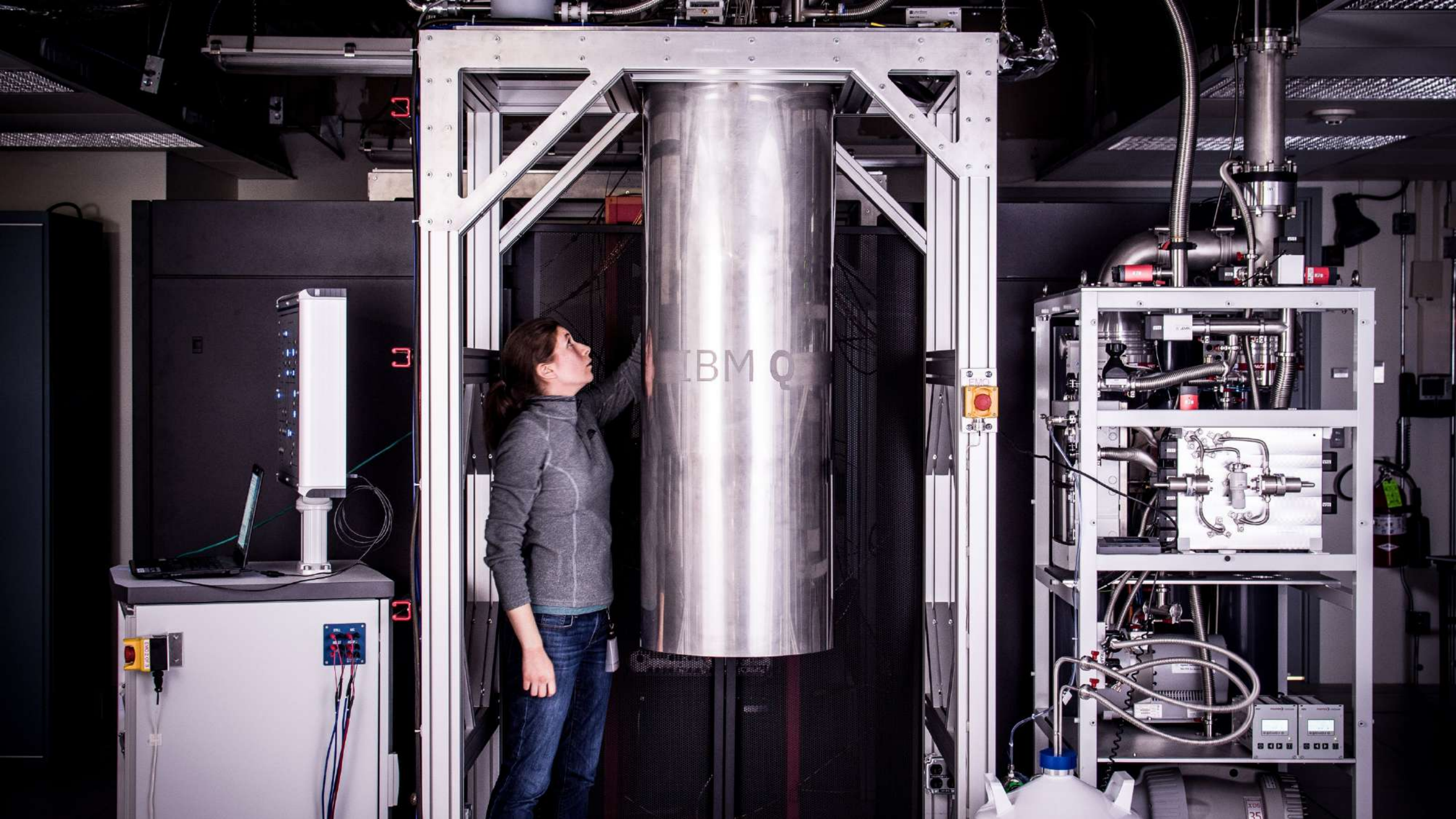
The Road to Quantum Advantage

Quantum
Science

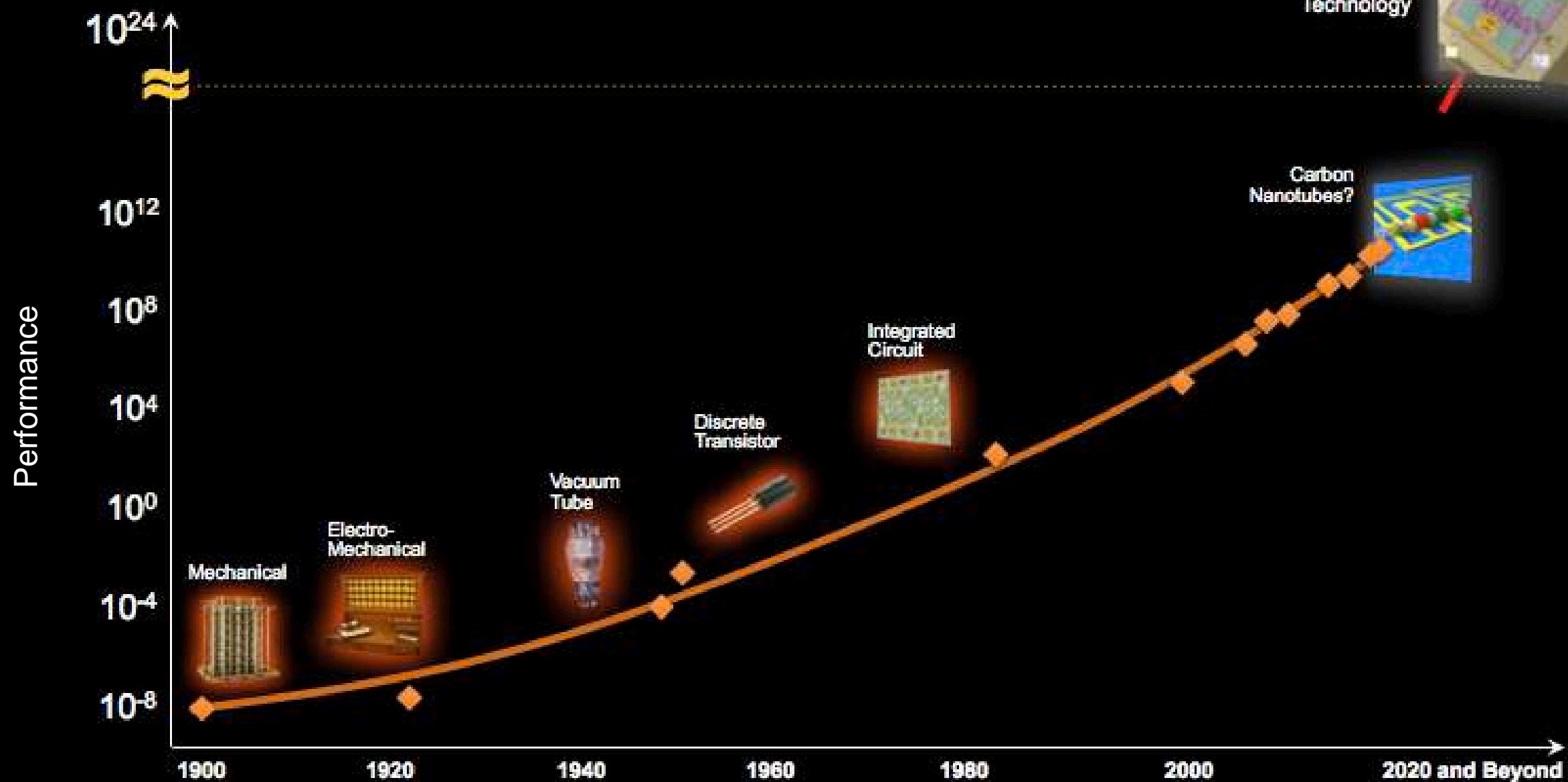
Quantum
Ready

Quantum
Advantage



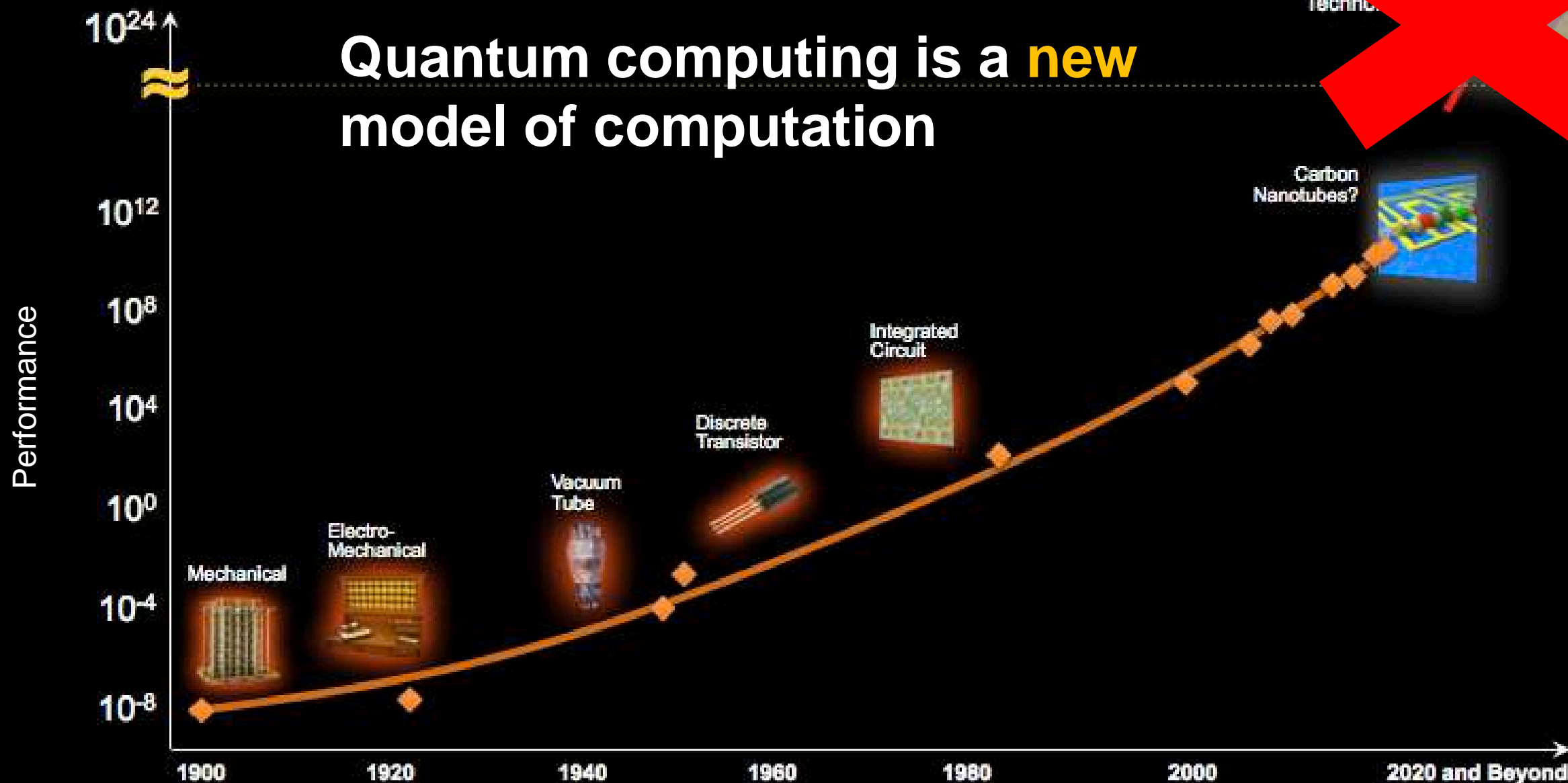


Moore's Law



Moore's Law

Quantum computing is a **new** model of computation



Exponential scaling



There's a famous legend outlined in ["IBM Mathematics Peep Show"](#)

The inventor of chess showed it to the emperor of India, and the emperor was so impressed he said *"Name your reward!"*

The man responded. *"Oh emperor, my wishes are simple. I only wish for this. For the next 64 days I will come back and for the first day please only give me one grain of rice for the first square of the chessboard, on the second day two grains for the next square, four for the next, eight for the next and so on for all 64 squares, with each square having double the number of grains as the square before."*

The emperor agreed, amazed that the man had asked for such a small reward - or so he thought.

..



On the first day...



After one week...127 grains of rice



After one month... 1,070,000,000 grains of rice

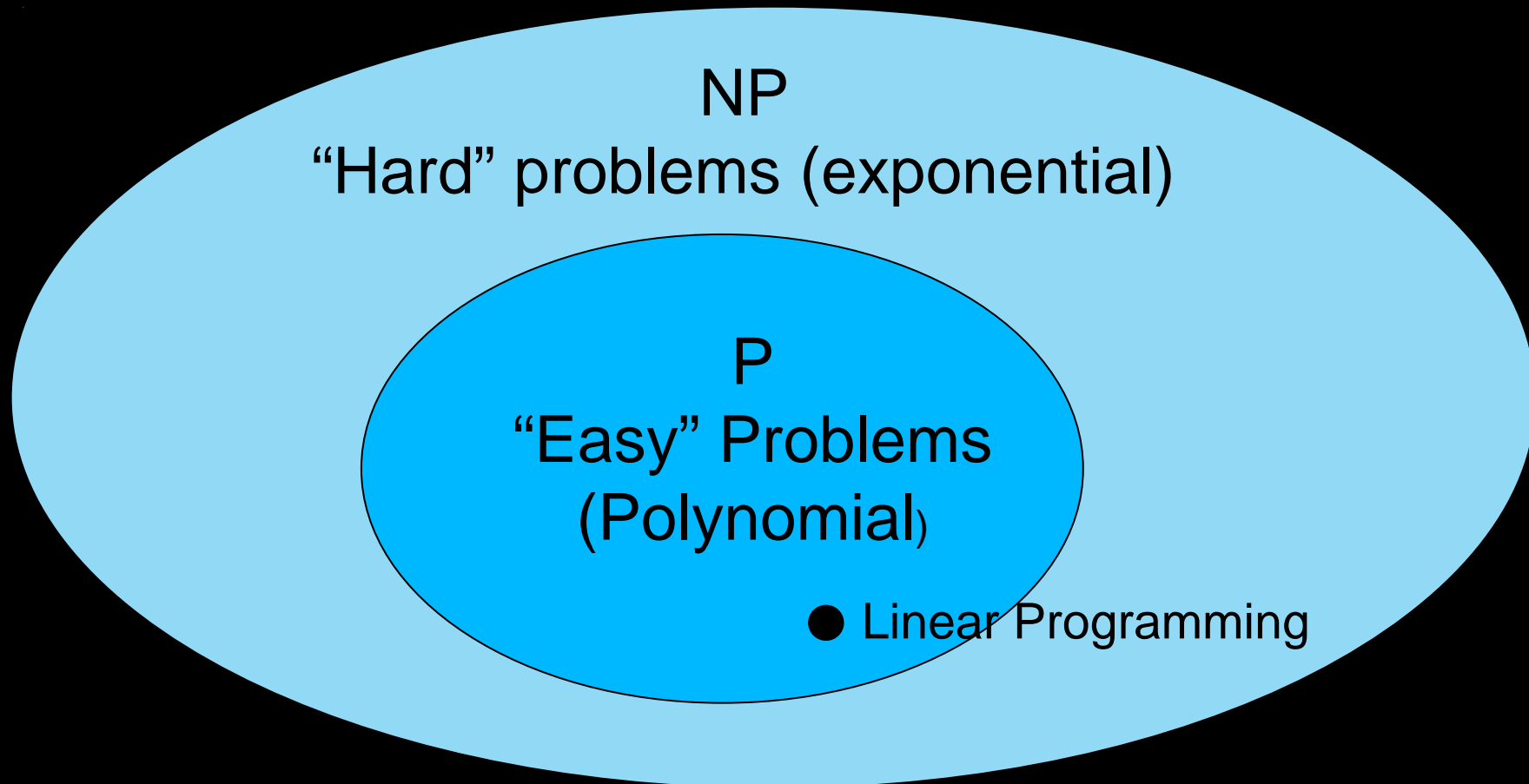


After 64 days ... 535 billion metric tons of rice



Limitations of conventional computers

There are many **intractable problems** where the best known algorithm has runtime that scales exponentially with input size



A new model of computation

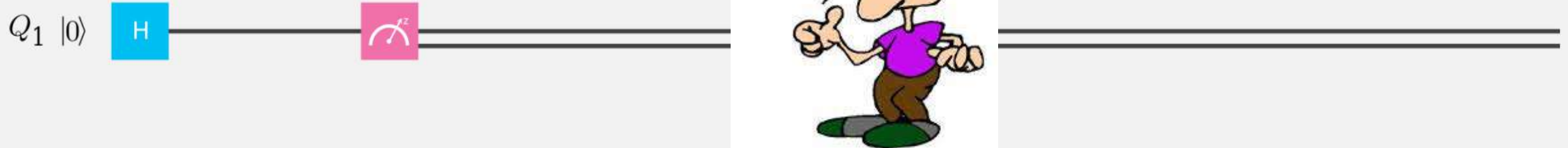


a physical system in a perfectly definite state can still behave randomly

two systems that are too far apart to influence each other can nevertheless behave in ways that, though individually random, are somehow strongly correlated.

Quantum Applications is about working out how to use these two principles in a new model of computation

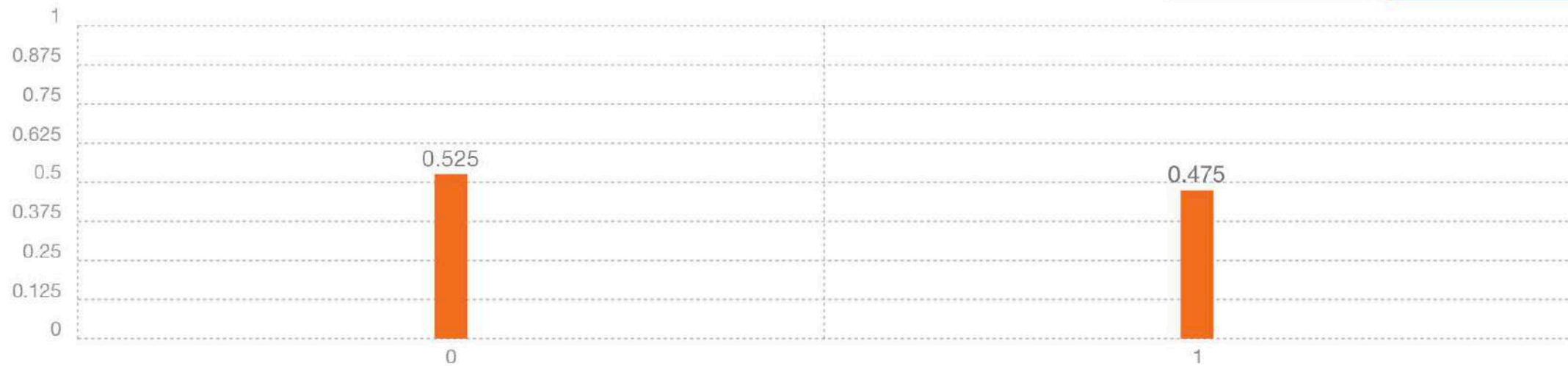
Superposition



Runs 1024

Show Qsphere

Download CSV



The outcomes appear to be random

Superposition

$Q_1 \quad |0\rangle$



Superposition

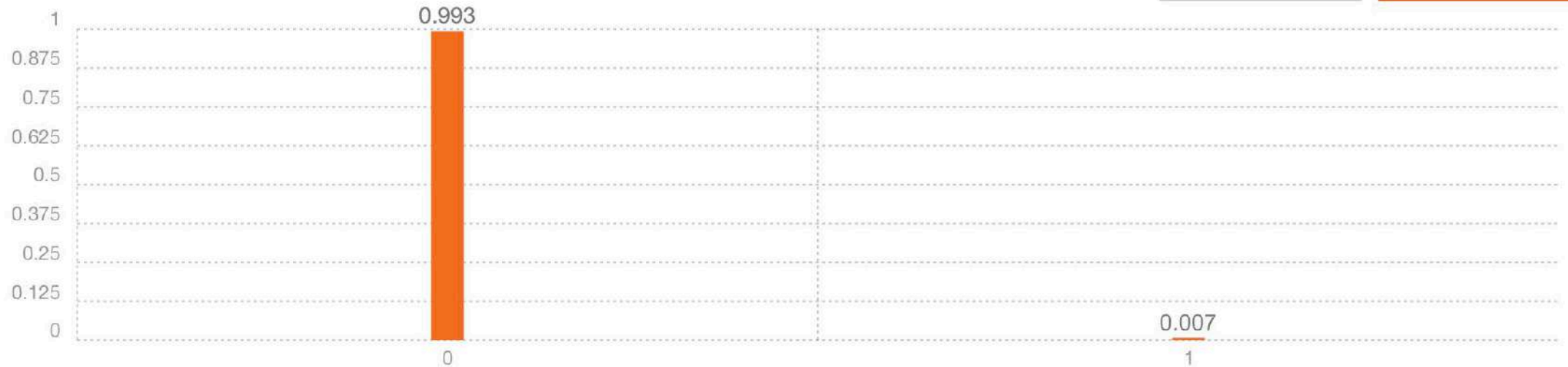
Q_1 $|0\rangle$



Runs 1024

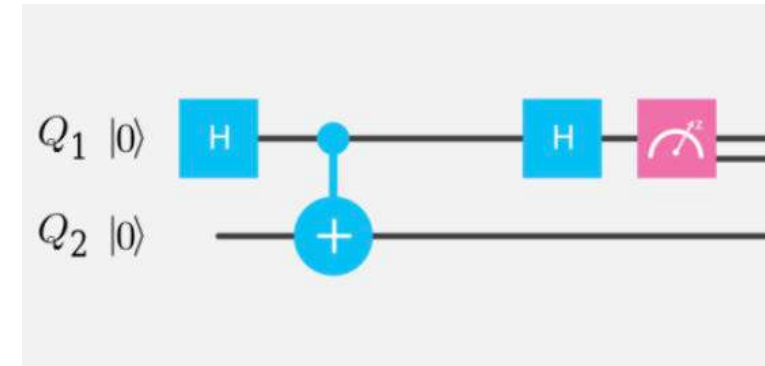
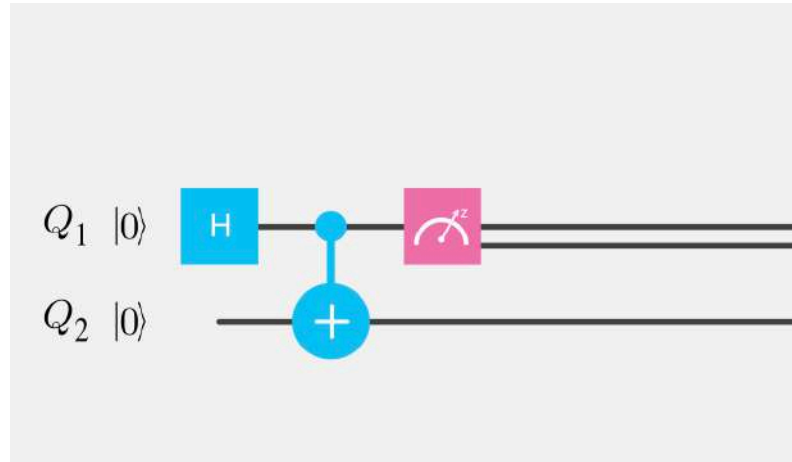
Show Qsphere

Download CSV

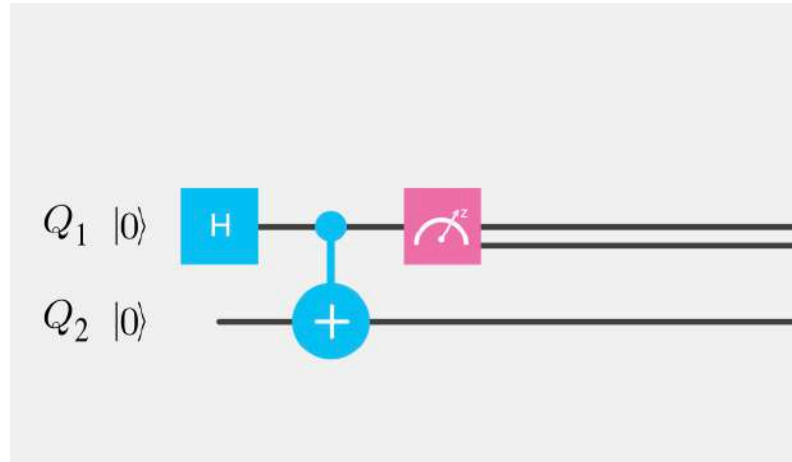


Doing it twice makes it certain again. Can not be a **classical random mixture**

Entanglement

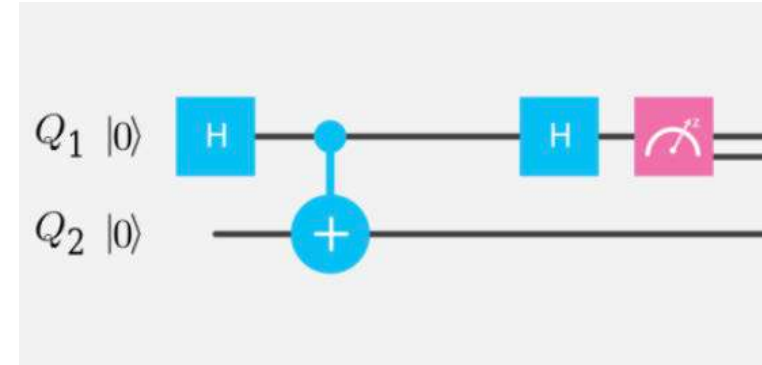


Entanglement



Runs 4096

[Download CSV](#)

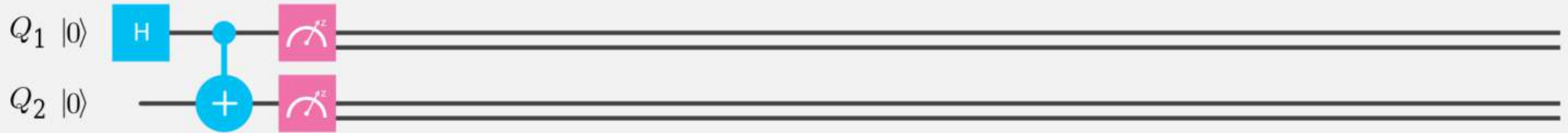


[Download CSV](#)



qubit 1 it has **no information** about the **quantum state**. It is not superposition or a computational state.

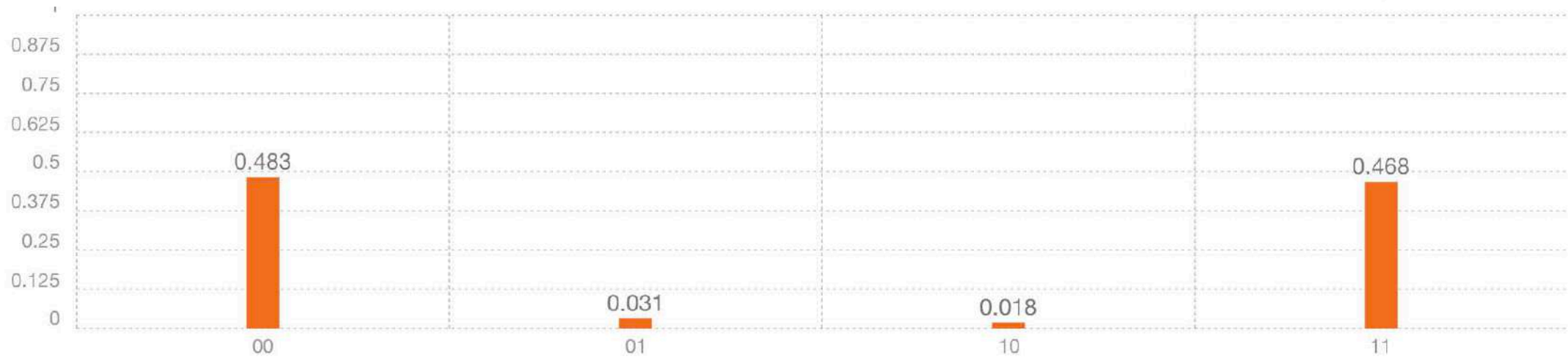
Entanglement



Runs 4096

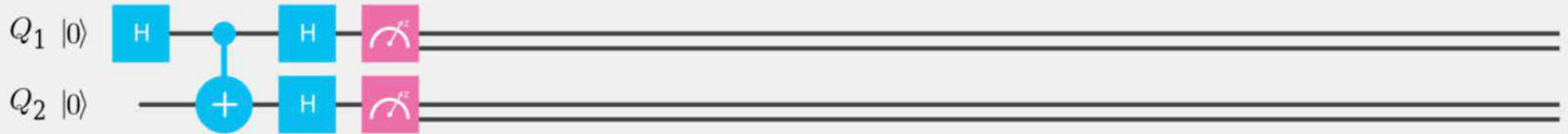
Show Qsphere

Download CSV



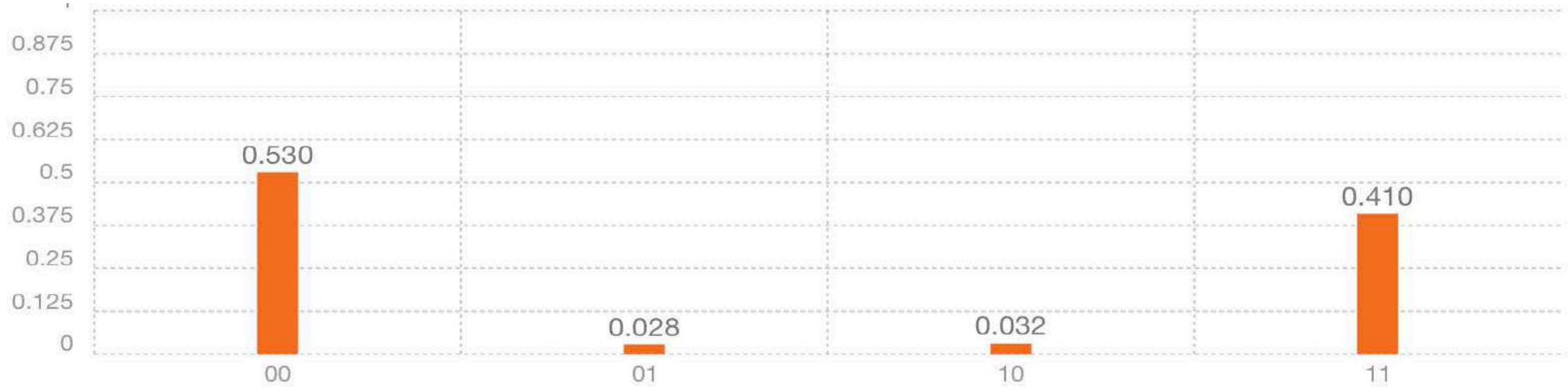
Correlated in the computational basis

Entanglement



Runs 4096

[Download CSV](#)



Correlated in the superposition basis

Quantum Computing and Quantum Circuits I

My First Score

Add a description

NewSaveSave as

< >

Switch to Qasm Editor

Backend: Custom TopologyMy Units: 56 Experiment Units: 3

Simulate

q[0] $|0\rangle$

q[1] $|0\rangle$

q[2] $|0\rangle$

q[3] $|0\rangle$

q[4] $|0\rangle$

H

id

S†

+

id

+

+

X

Z

S

+

S

+

+

H

S

Y

X

X

H

Y

Z

S

+

Z

Z

Z

S†

id

H

S†

S†

S†

c 0 $\frac{5}{7}$

012

GATES

☐ Advanced

idXYSZH

SS†+TT†

BARRIER

OPERATIONS

Quantum Computing and Quantum Circuits II

A quantum computation: $|\psi'\rangle = U|\psi\rangle$ for all $U \in SU(2^n)$

The 2 – qubit CNOT gate $\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$



$U(\theta, \phi, \lambda) := \begin{pmatrix} e^{-i(\phi+\lambda)/2} \cos(\theta/2) & -e^{-i(\phi-\lambda)/2} \sin(\theta/2) \\ e^{i(\phi-\lambda)/2} \sin(\theta/2) & e^{i(\phi+\lambda)/2} \cos(\theta/2) \end{pmatrix}$
with all single

U3

▪ A universal set of gates

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



$S = \sqrt{Z} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$



$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$



$T = \sqrt{S} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$



Quantum math in a single slide

Probability Theory:

$$\begin{pmatrix} t_{11} & \dots & t_{12^n} \\ \vdots & \ddots & \vdots \\ t_{2^n 1} & \dots & t_{2^n, 2^n} \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_{2^n} \end{pmatrix} = \begin{pmatrix} q_1 \\ \vdots \\ q_{2^n} \end{pmatrix}$$

$$p_i \geq 0 \quad \sum_{i=1}^{2^n} p_i = 1$$

Linear transformations that conserve 1-norm of probability vectors: **Stochastic matrices**

There **exists** efficient ways to simulate this **exponential**

Quantum Theory:

$$\begin{pmatrix} u_{11} & \dots & u_{12^n} \\ \vdots & \ddots & \vdots \\ u_{2^n 1} & \dots & u_{2^n 2^n} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{2^n} \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_{2^n} \end{pmatrix}$$

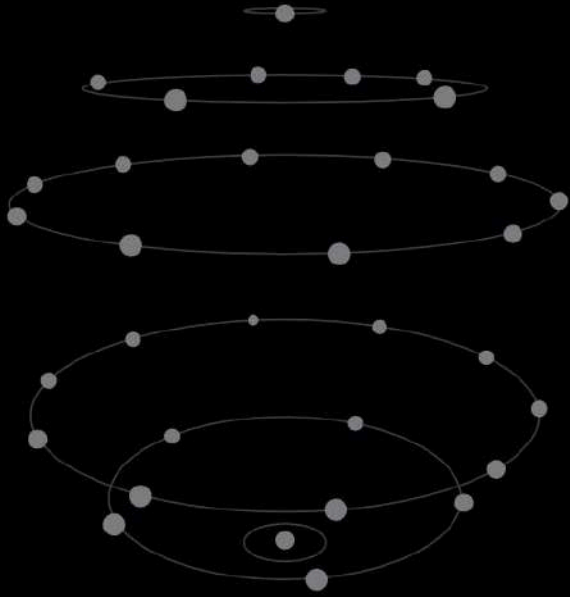
$$\alpha_i \in \mathbb{C} \quad \sum_{i=1}^{2^n} |\alpha_i|^2 = 1$$

Linear transformations that conserve 2-norm of amplitude vectors: **Unitary matrices**

There does **not exist** efficient ways to simulate this **exponential** (negative sign)

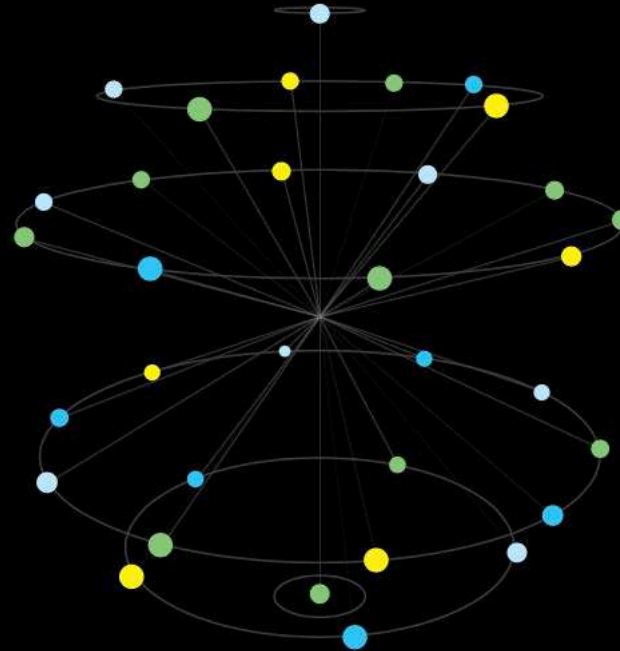
It is like “Probability theory with Minus Signs”

A Quantum Algorithm



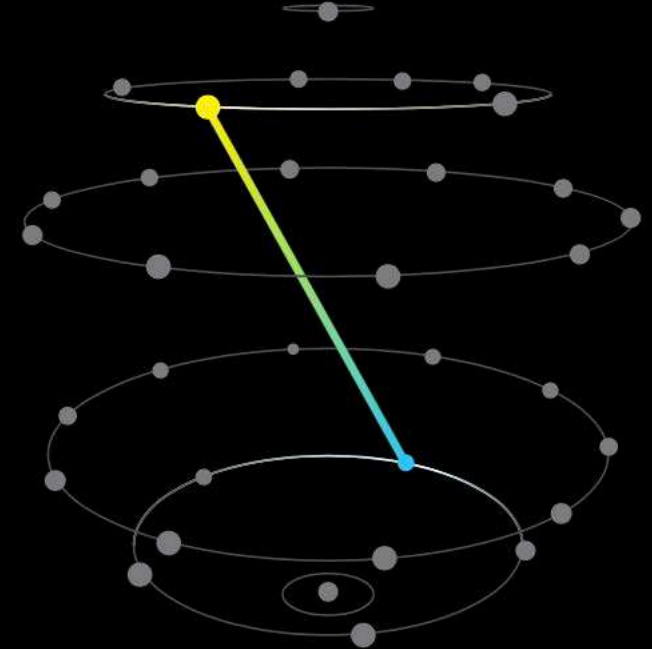
The spread

First part of the algorithm is make a equal superposition of all 2^n states. Apply H gates



The problem

The second part is to encode the problem into this states (phases on the on the all 2^n states.



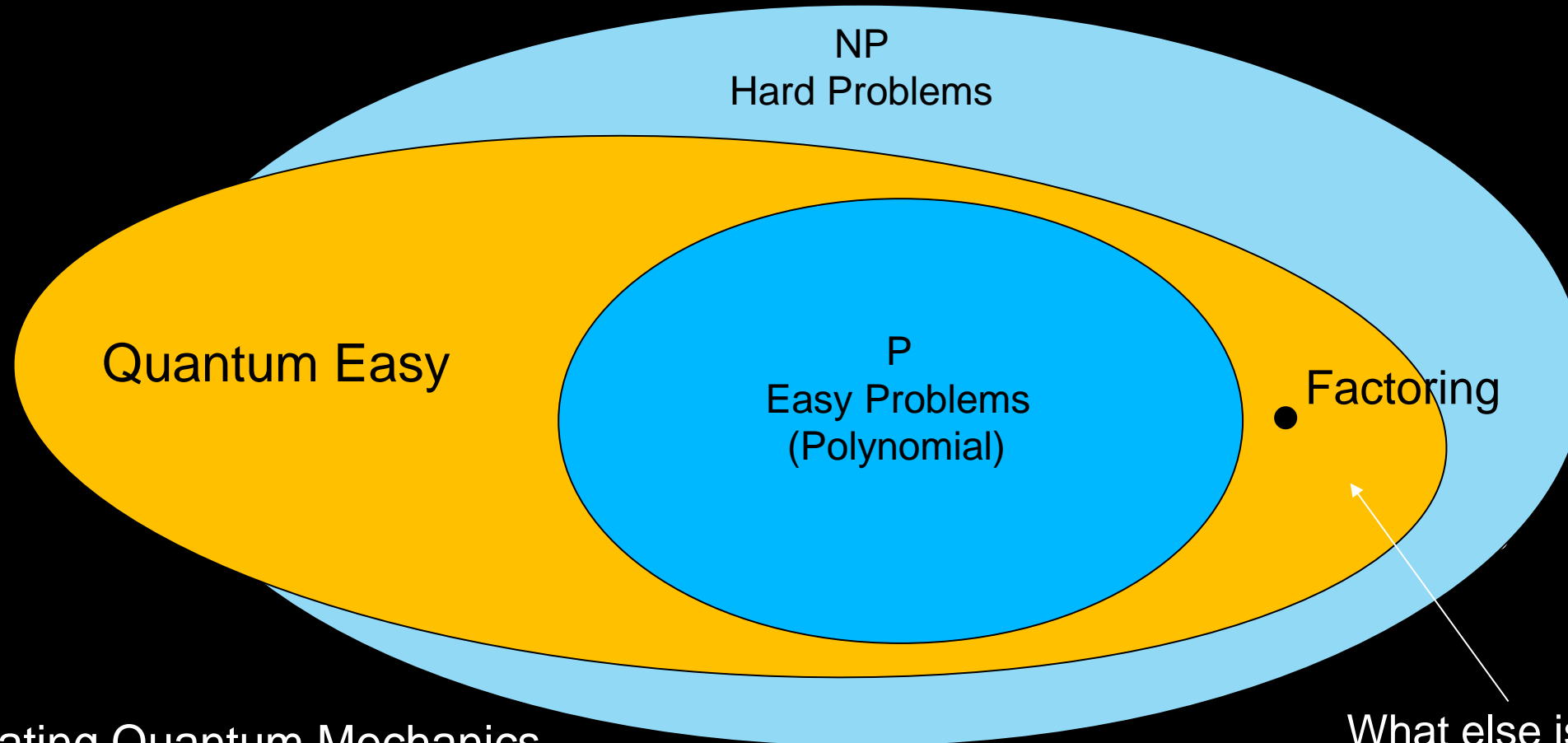
The magic

The magic of quantum algorithms is to interfere all these states back to a few outcomes containing the solution

Quantum speedups



...Quantum computers are the **only** novel hardware which changes the game



Simulating Quantum Mechanics

What else is in here?

Types of quantum computing



Universal fault-tolerant quantum computer

The holy grail of quantum information science. Allows one to run **useful** quantum algorithms which achieve **exponential speed ups** over their classical counterparts. However the over head of quantum error correction estimates **1M-5M qubits**

Approximate quantum computer

A quantum device which does not have fault tolerance, with the goal of demonstrating a useful application by interacting with a classical computing system, e.g. quantum chemistry, optimization. Estimate **1K-5K qubits**

Analog / quantum Annealing

A special built system which uses **quantum effects** to solve/emulate a specific problem. It has limited programmability and unclear if and when it has a speed up over conventional computers.

Types of quantum computing



Universal fault-tolerant quantum computer

The holy grail of quantum information science. Allows one to run **useful** quantum algorithms which achieve **exponential speed ups** over their classical counterparts. However the over head of quantum error correction estimates **1M-5M qubits**

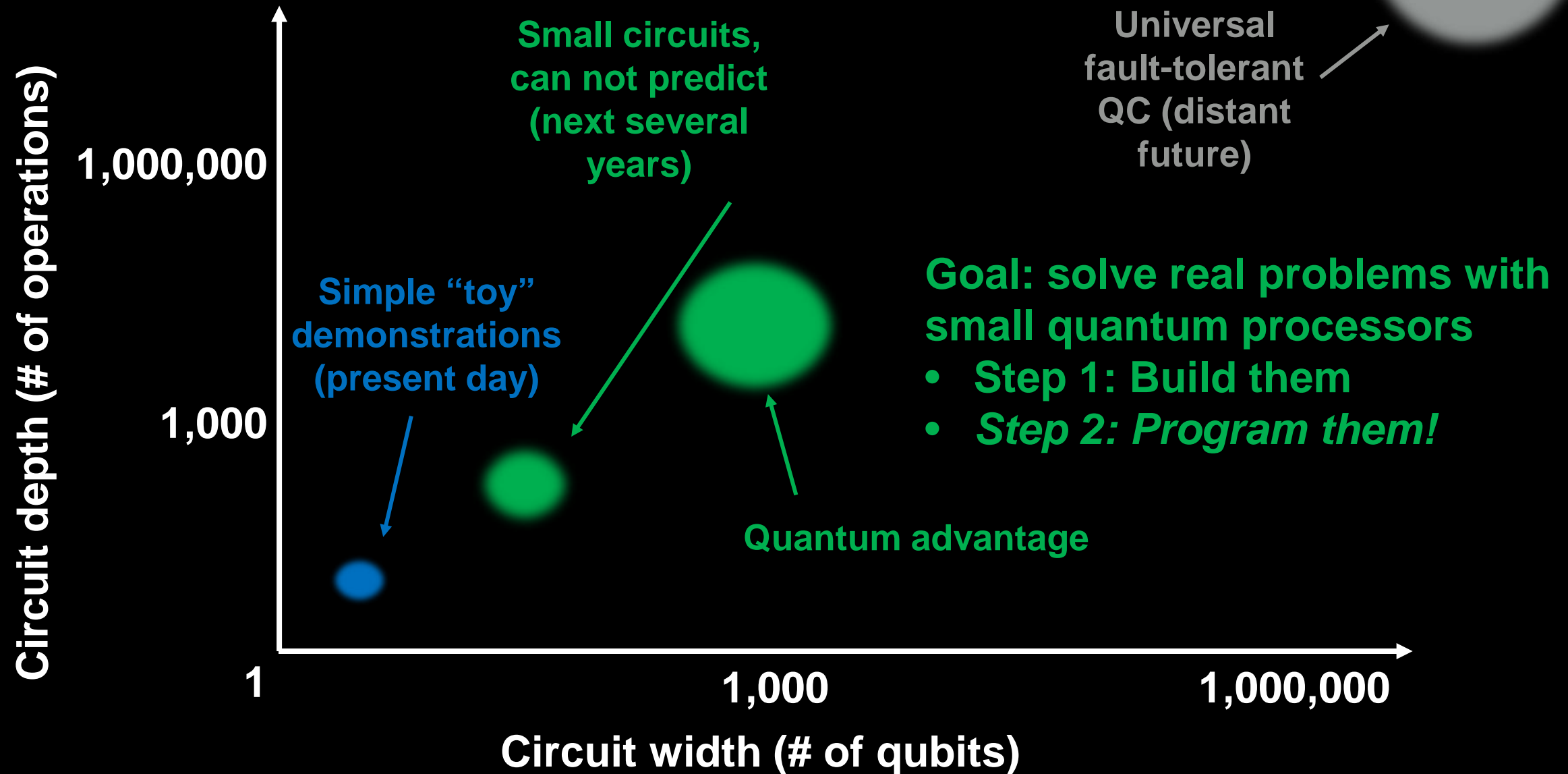
Approximate quantum computer

A quantum device which does not have fault tolerance, with the goal of demonstrating a useful application by interacting with a classical computing system, e.g. quantum chemistry, optimization. Estimate **1K-5K qubits**

Analog / quantum Annealing

A special built system which uses **quantum effects** to solve/emulate a specific problem. It has limited programmability and unclear if and when it has a speed up over conventional computers.

Toward a Quantum Approximate computer



Technical goals for QISKit software platform



- Plan for continued improvement of quantum devices
 - Increasing size, capability, and fidelity
- Build software tools for working with these near-term quantum computing systems
 - *Short-depth circuits*: enable investigation of algorithms toward quantum advantage
 - *Pre-fault-tolerance*: enable exploration of broad error mitigation techniques
- Create a framework for experiments, simulations, and analysis
 - Backend-independent interface for running experiments
 - Multiple simulators and analysis tools
 - Circuit rewriting infrastructure
 - Optimization, scheduling, hardware mapping
- Increase capabilities and add features over time
 - Expose lower level control interfaces and extend mapping framework
 - Access to timing and pulse shape
 - Introduce higher level abstractions



IBM Quantum Experience

Anatomy of a superconducting qubit device

Qubits:

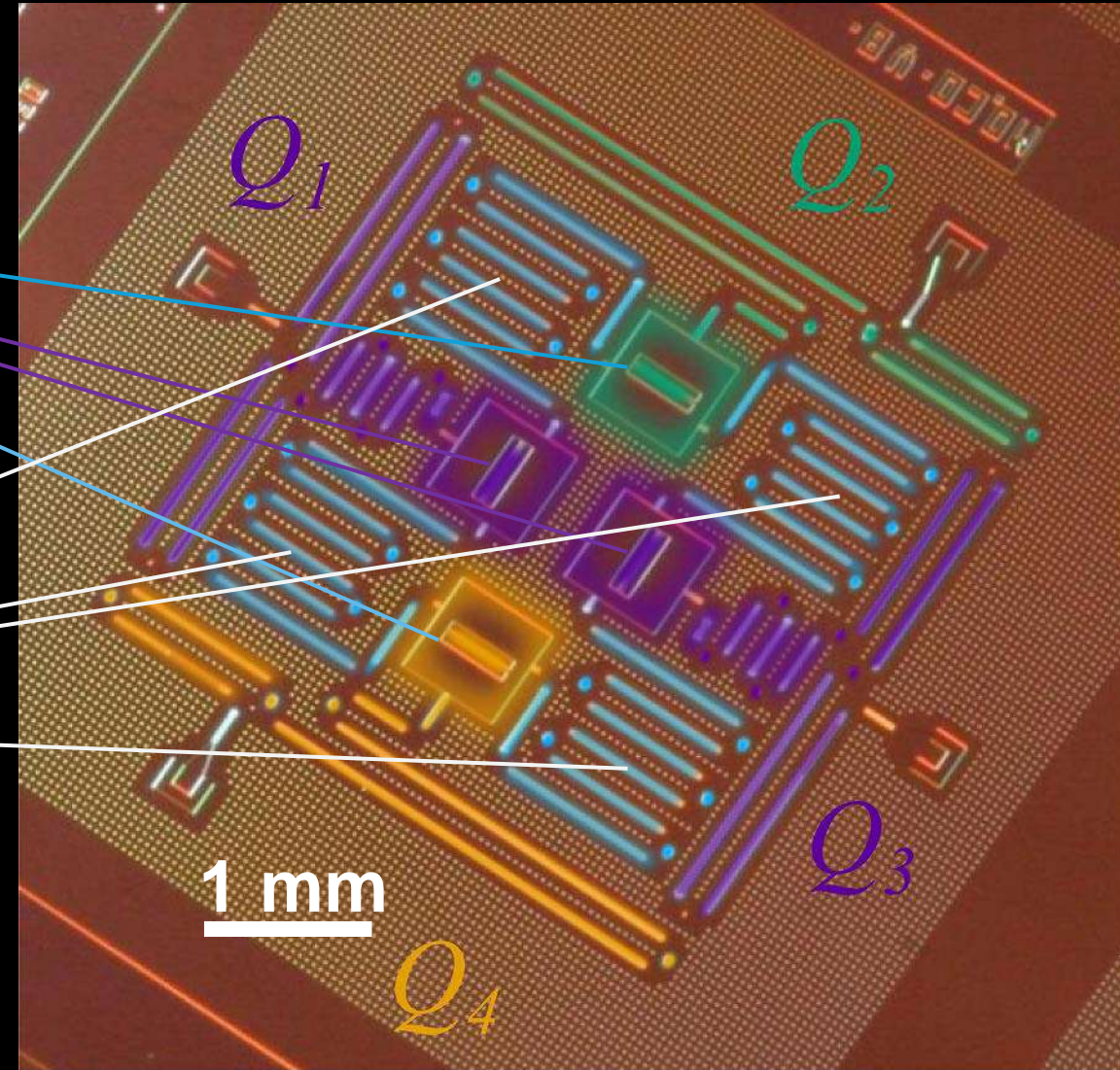
Single-junction transmon
Frequency ~ 5 GHz
Anharmonicity ~ 0.3 GHz

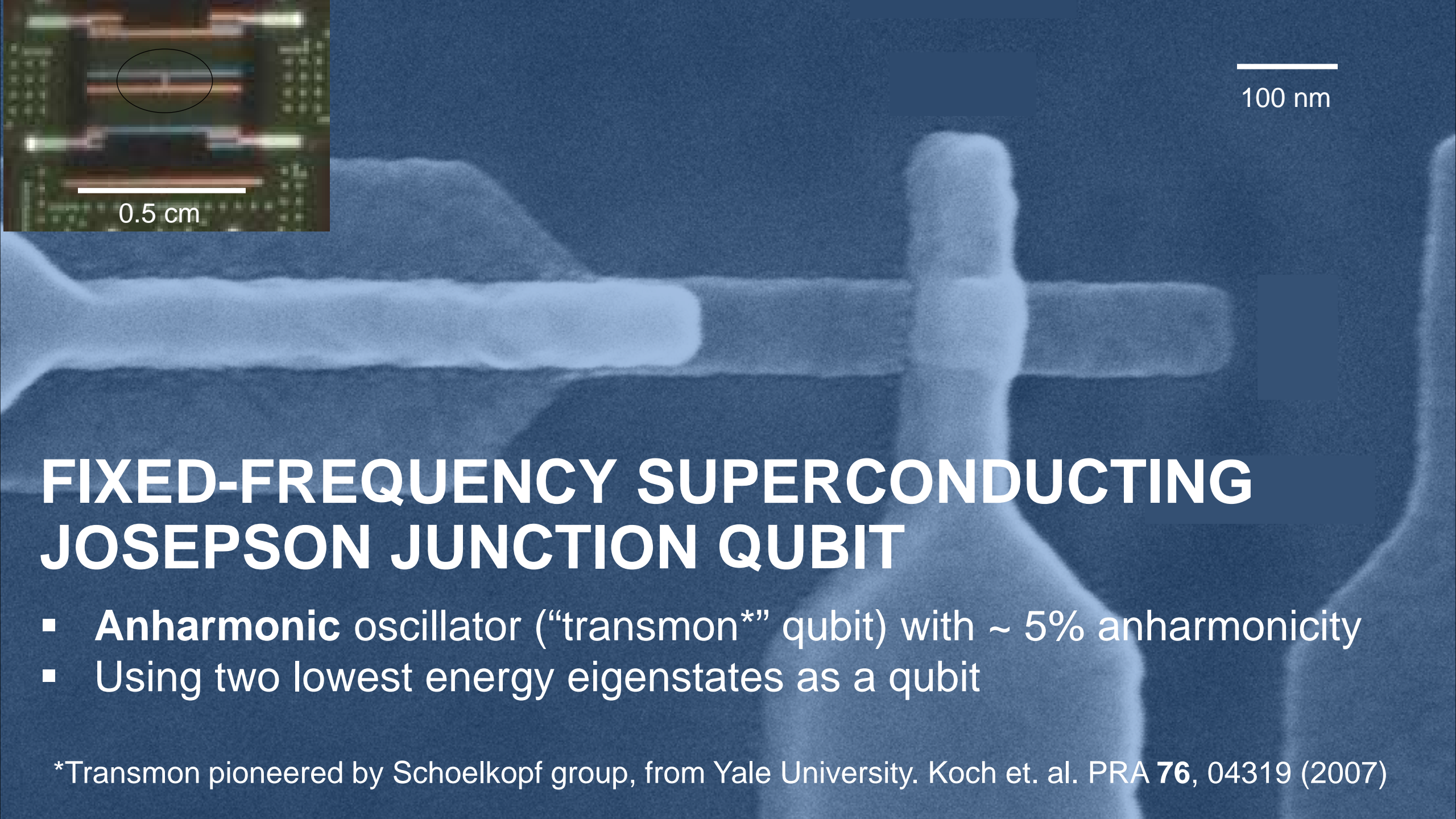
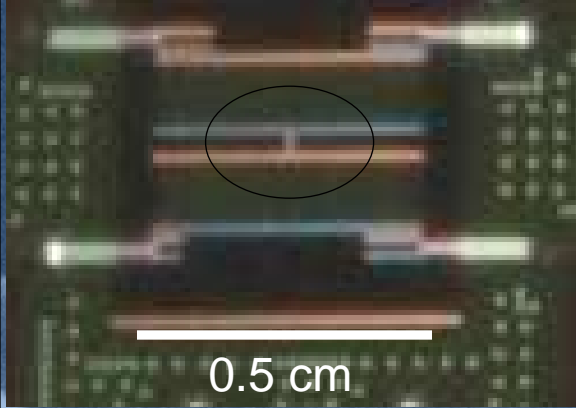
Resonators:

Co-planar waveguide
Frequency $\sim 6 - 7$ GHz

Roles:

Individual qubit readout
Qubit coupling (“bus”)





100 nm

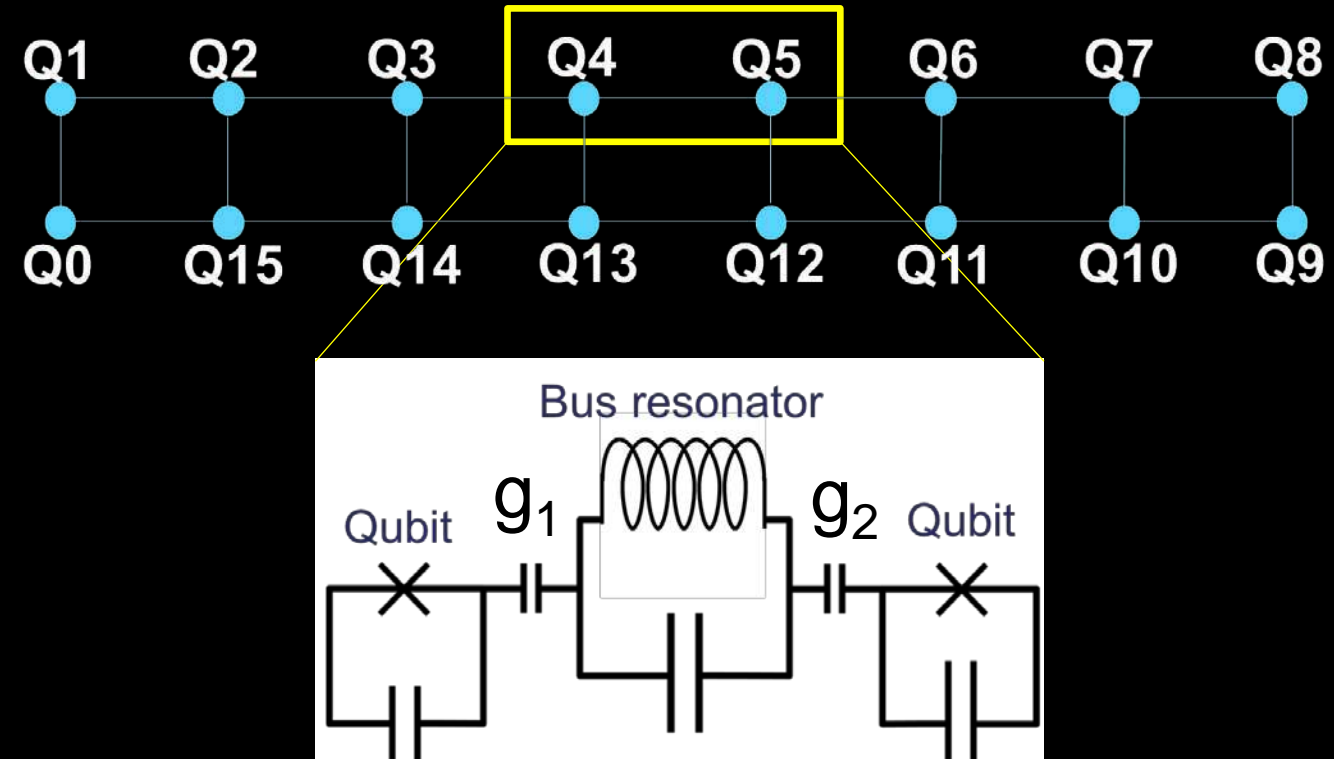
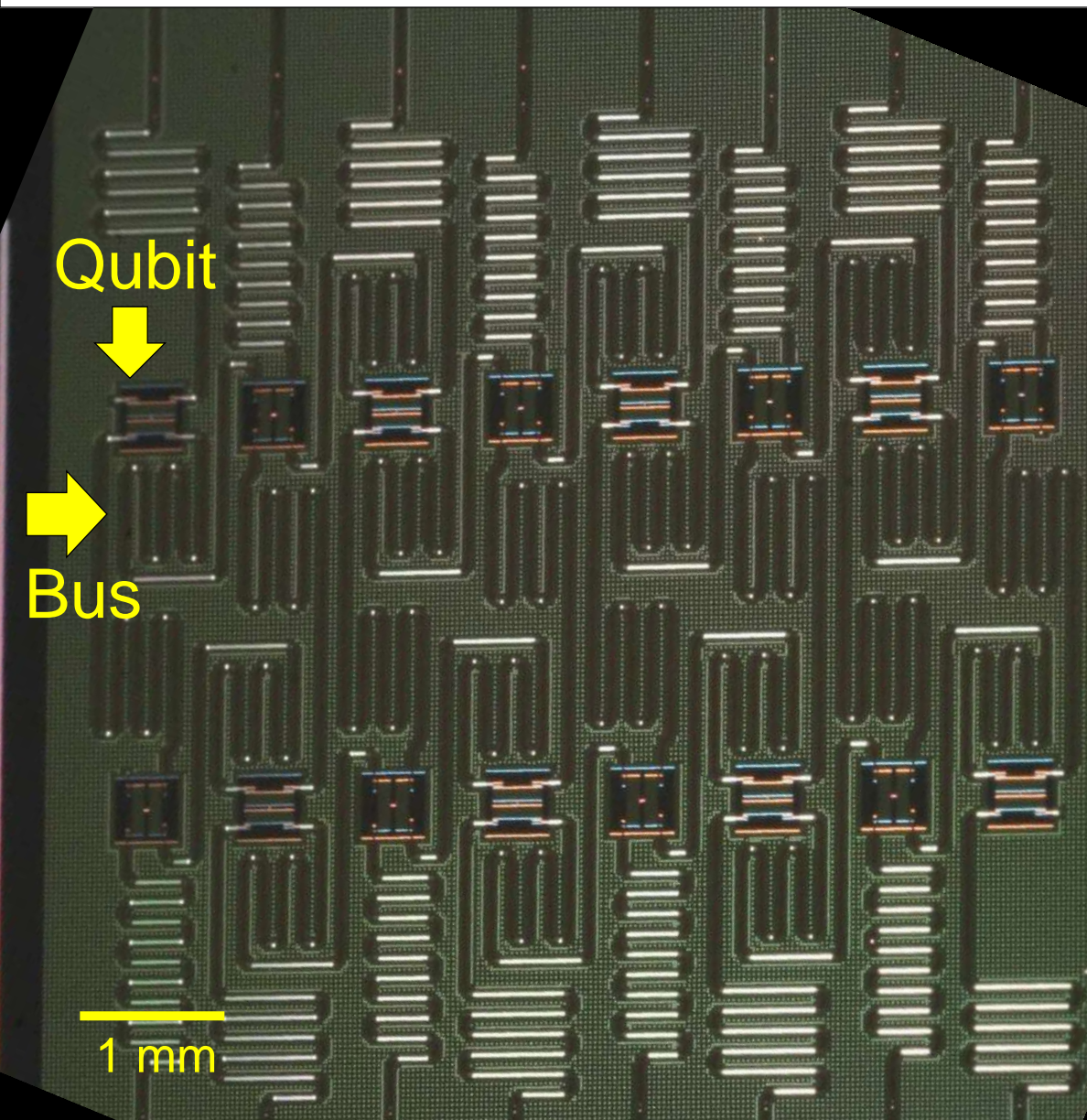
A scanning electron micrograph (SEM) of a superconducting circuit. The image shows a complex pattern of thin, dark lines (superconducting wires) on a lighter background. A white scale bar at the bottom right indicates a length of 100 nm.

FIXED-FREQUENCY SUPERCONDUCTING JOSEPHSON JUNCTION QUBIT

- Anharmonic oscillator (“transmon*” qubit) with $\sim 5\%$ anharmonicity
- Using two lowest energy eigenstates as a qubit

*Transmon pioneered by Schoelkopf group, from Yale University. Koch et. al. PRA **76**, 04319 (2007)

COUPLING QUBITS BY BUS RESONATOR



- Bus frequency detuned from qubit frequencies. $|f_{\text{qubit}} - f_{\text{bus}}| \gg g$
- Two-qubit exchange interaction J via virtual photons is mediated by the bus resonator.

The image panel with colonoscopy (C) still was not found in the file

Properties of current devices:

Single-junction transmon qubits

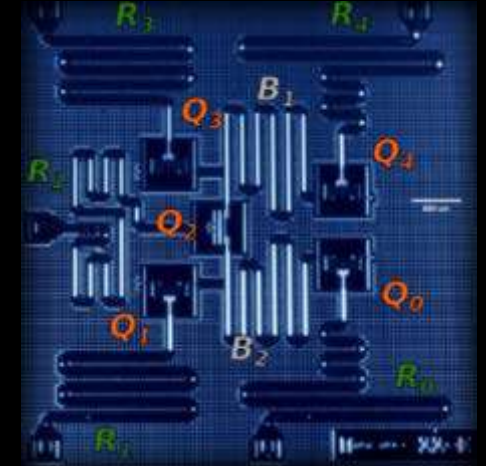
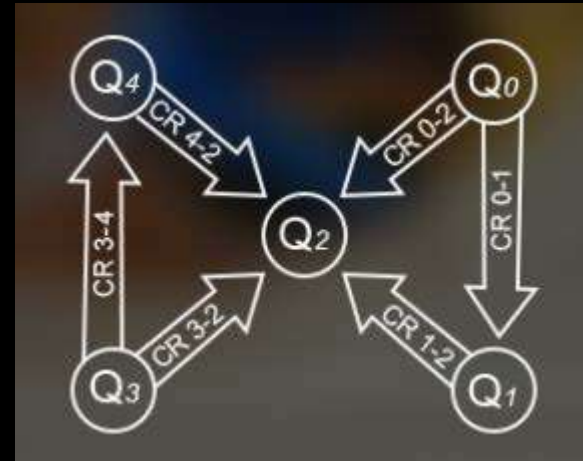
$$T_1 \sim T_2 \sim 50 \mu s$$

1Q gate fidelities > 99%

2Q gate fidelities > 95%

Measurement fidelities > 93%

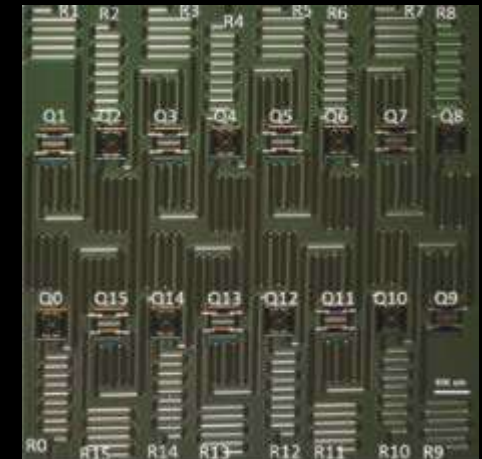
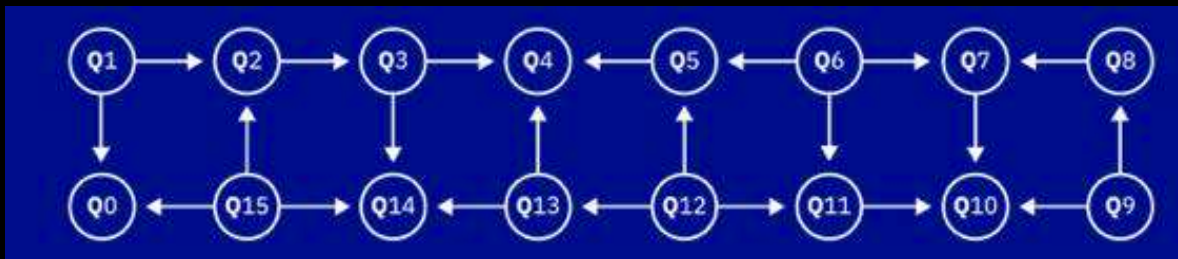
Nearest-neighbor couplings



Current device offerings:

5-qubit device ("ibmqx4"): access via web GUI and QISKit API

16-qubit device ("ibmqx5"): access via QISKit API only



Circuit building blocks: single-qubit basis gates

Goals: achieve universal qubit control, maximize fidelity, minimize need for calibrations

Arbitrary rotation on the Bloch sphere = up to 3 successive rotations around fixed axes:

$$U(\theta, \phi, \lambda) = R_z(\phi)R_y(\theta)R_z(\lambda)$$

Good news: arbitrary R_z can be done instantaneously and exactly

Just adjust phase of carrier to change reference frame

Bad news: can't calibrate $R_y(\theta)$ to high fidelity for arbitrary θ

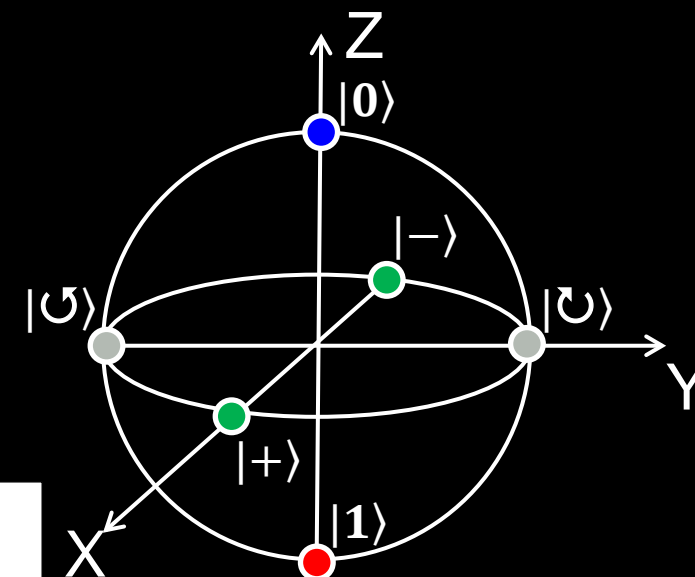
Solution: re-write arbitrary gates to use N frame changes plus

$N - 1$ well-calibrated gates with fixed axis and angle ($N = 1, 2, \text{ or } 3$)

$$u_1(\lambda) = U(0, 0, \lambda) = R_z(\lambda) = \omega_q \text{ --- FC } (-\lambda) \text{ ---}$$

$$u_2(\phi, \lambda) = U(\pi/2, \phi, \lambda) = \omega_q \text{ --- FC } (-\lambda) \text{ --- } \overset{Y_{\pi/2}}{\text{GD } (\pi/2, \pi/2)} \text{ --- FC } (-\phi) \text{ ---}$$

$$u_3(\theta, \phi, \lambda) = U(\theta, \phi, \lambda) = \omega_q \text{ --- FC } (-\lambda) \text{ --- } \overset{X_{\pi/2}}{\text{GD } (\pi/2, 0)} \text{ --- FC } (-\theta) \text{ --- } \overset{X_{-\pi/2}}{\text{GD } (\pi/2, \pi)} \text{ --- FC } (-\phi) \text{ ---}$$



FC = frame change
GD = Gaussian w/ DRAG

Circuit building blocks: two-qubit basis gate

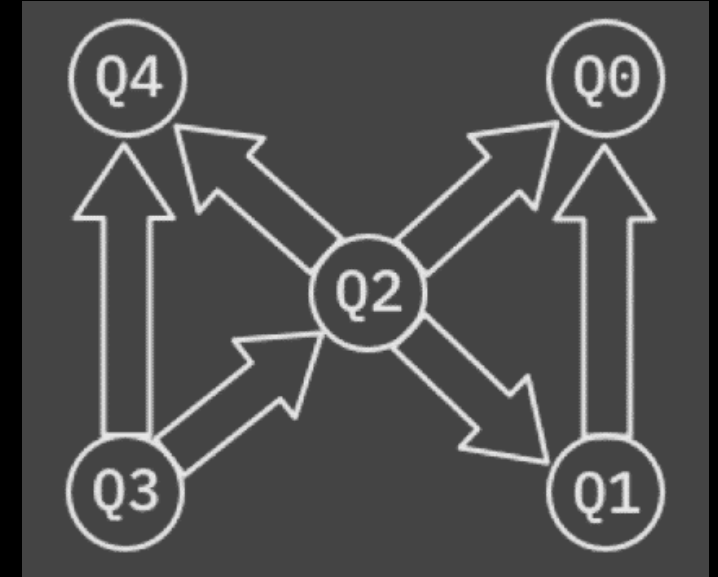
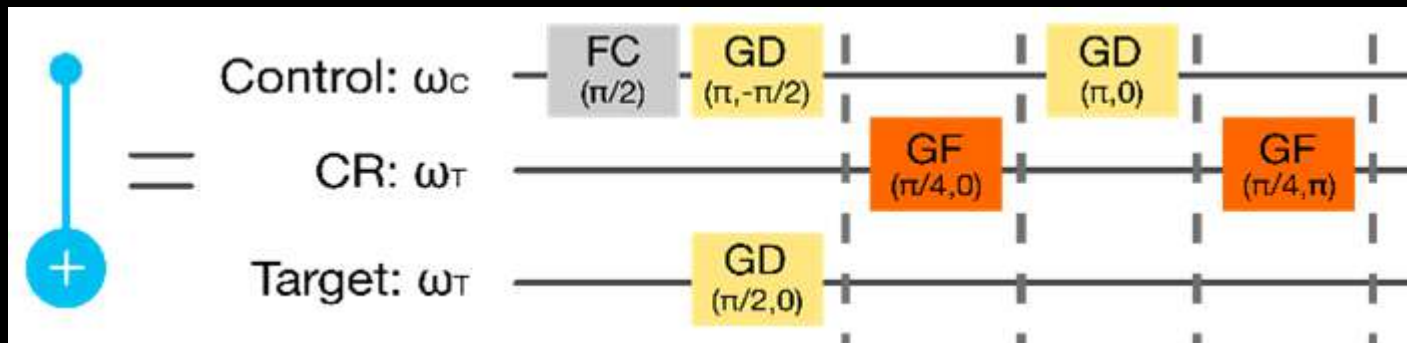
In conjunction with the single-qubit gates, need just a single entangling gate for universal control (ideally well-calibrated)

Due to physics of the cross resonance effect, optimal direction of CNOT (i.e. which qubit is control vs. target) depends on frequency and anharmonicity of the qubits involved

Implement CNOT in only one direction for each qubit pair

Include directionality in coupling map given to user

In practice, cross-resonance tone is split in half, with a refocusing pulse in the middle to cancel slow fluctuations



CNOT gates on a 5-qubit chip Arrows go from control to target

> Backend: QS1_1 (20 Qubits)

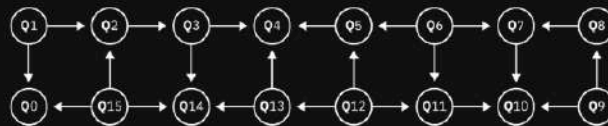
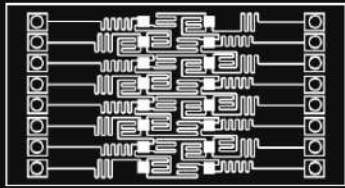
ACTIVE

AVAILABLE TO HUBS, PARTNERS, AND MEMBERS OF THE IBM Q NETWORK

✓ Backend: ibmqx5 (16 Qubits)

ACTIVE

AVAILABLE ON QISKIT



Frequency (GHz)

T1 (μ s)

T2 (μ s)

Gate error (10^{-3})

Readout error (10^{-2})

MultiQubit gate error (10^{-2})

	Q0	Q1	Q2	Q3	Q4	Q5	Q6
Frequency (GHz)	5.26	5.40	5.28	5.08	4.98	5.15	5.31
T1 (μ s)	45.00	38.30	48.40	48.20	41.70	35.30	43.10
T2 (μ s)	36.40	50.20	60.30	82.90	98.10	41.60	68.20
Gate error (10^{-3})	1.82	3.64	3.71	2.08	1.42	2.06	1.62
Readout error (10^{-2})	5.85	7.21	4.03	5.14	7.51	5.07	5.09
MultiQubit gate error (10^{-2})		CX1_0 4.18	CX2_3 3.96	CX3_4 3.76		CX5_4 3.74	CX6_5 3.74
		CX1_2 4.19		CX3_14 3.34			CX6_7 3.23
							CX6_11 6.10

Date Calibration: 2018-01-08 07:30:27

Fridge Temperature: 0.0136409 K

More details

> Backend: ibmqx4 (5 Qubits)

MAINTENANCE

AVAILABLE ON QISKIT

> Backend: ibmqx2 (5 Qubits)

ACTIVE

AVAILABLE ON QISKIT

✓ Backend: ibmqx_qasm_simulator

ACTIVE

SIMULATOR

AVAILABLE ON QISKIT

Number of qubits

20

Conditionals (if)

Yes

✓ Backend: ibmqx_hpc_qasm_simulator

ACTIVE

SIMULATOR

AVAILABLE ON QISKIT

Number of qubits

32

Conditionals (if)

No

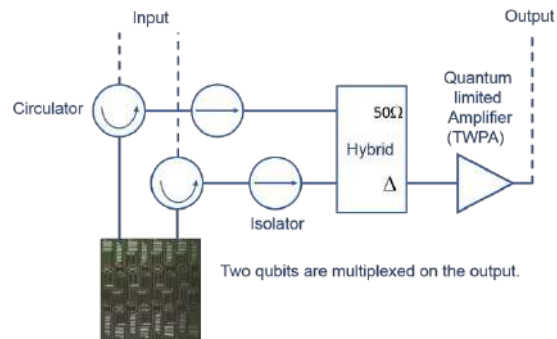
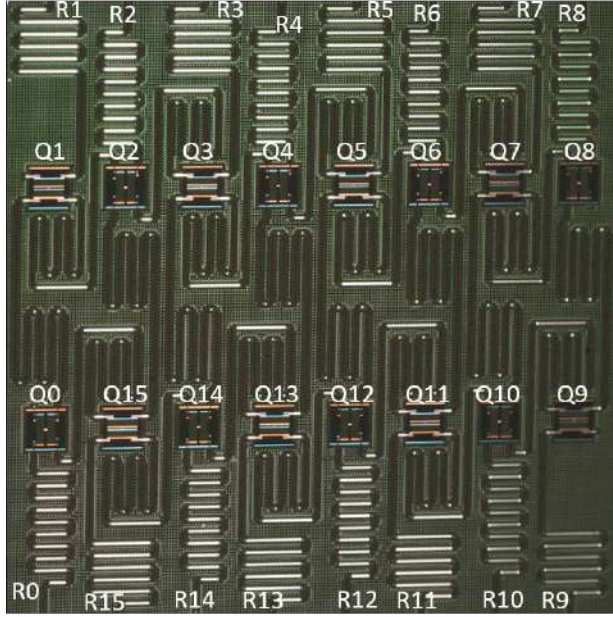


dark

<https://quantumexperience.ng.bluemix.net/qx/devices>

Additional backend information

<https://github.com/QISKit/ibmqx-backend-information>



IBM QX3

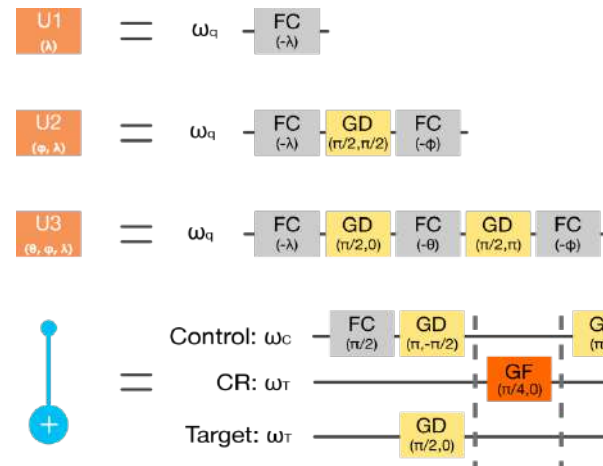
This document contains information about the IBM Quantum Experience `ibmqx3` backend.

Contributors (alphabetical)

Baleegh Abdo, Vivekananda Adiga, Lev Bishop, Markus Brink, Nicholas Bronn, Jerry Chow, Antonio Córcoles, Andrew Cross, Jay M. Gambetta, Jose Chavez-Garcia, Jared Hertzberg, Oblesh Jinka, George Keefe, David McKay, Salvatore Olivadese, Jason Orcutt, Hanhee Paik, Jack Rohrs, Sami Rosenblatt, Jim Rozen, Martin Sandberg, Dongbing Shao, Sarah Sheldon, Firat Solgun, Maika Takita

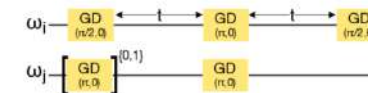
Status History

This device went online June 2017.



Crosstalk, which we parameterize by $\zeta_{ij} = (E_{i1} - E_i - E_j + E_{j1})/h$ is measured using a Joint Amplification of ZZ (JAZZ) experiment, which is a modified Bilinear Rotational Decoupling (BIRD) [46n1]. The standard BIRD sequence used in nuclear magnetic resonance (NMR) is a Ramsey experiment on one qubit with echo pulses on both the measured qubit (Q_1) and the coupled qubit (Q_2). In the JAZZ experiment, this sequence is performed twice, for each initial state of the coupled qubit. Additionally, the phase of the final $\pi/2$ -rotation is varied in order to detect an oscillating signal. ζ_{ij} is equal to the frequency difference found between the two experiments. The JAZZ experiment is shown in the figure below, and the measurements of all ζ_{ij} are in the following table. The GD pulse notation is defined below in the Gate Specification section.

[*fn1]: J.R. Garbow, D.P. Weitekamp, A. Pines, Bilinear rotation decoupling of homonuclear scalar interactions, Chemical Physics Letters, Volume 93, Issue 5, 1982, Pages 504-509.



In the crosstalk matrix, the error bar is less than 1 kHz for all J_{ij} and a dash indicates an interaction strength for that pair < 5 kHz.

[illegible]

Much more than superconducting qubits on the cloud!

IBM Q > Experience

Community of over 60,000 users

Home Composer Devices Community GitHub Jay Gambetta

Forum

News

Videos

Papers

Awards

Post to forum

Search for...



All Categories

Quick links

- FAQ
- Beginner's Guide
- Full User Guide

General

1
comments
45

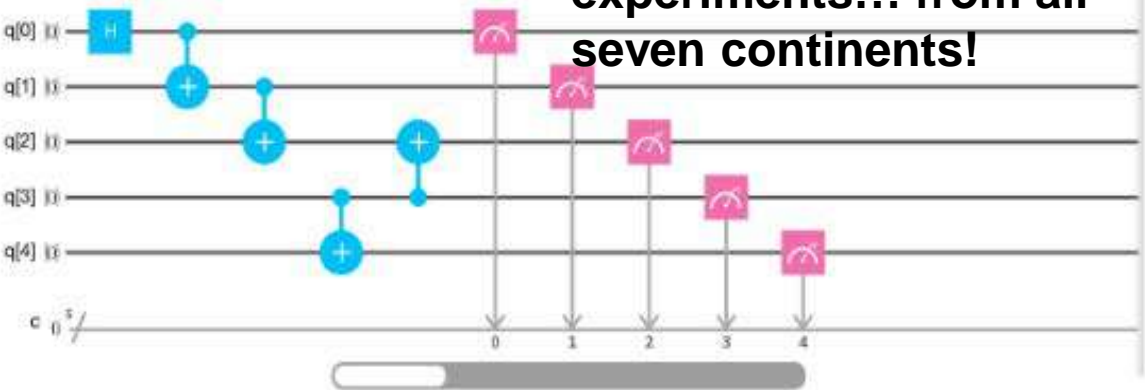
one question about Rx, Rz, and Ry gate in ibmqx

Hi everyone. In qelib1.inc, I found that Rx, Ry, Rz are implemented by U3, U2 and U1 gate. $Rx(\theta) = U3(\theta, -\pi/2, \pi/2)$ $Ry(\theta) = U3(\theta, 0, 0)$ $Rz(\theta) = U1(\dots$

New experiment New Save Save as

ibmqx2

Over a million experiments... from all seven continents!



Run Simulate

Gates Properties QASM

GATES ? Advanced

id	X	Y
Z	H	S
S†	+	T
T†		

Explanatory videos

A Qubit in the Making
Dr Markus Brink discusses how to make a qubit.
3A jaygambetta IBM Staff
Published 2 months ago
quantum computing

Go Behind-the-Scenes of a Quantum Experiment
A video with our very own Dr. David McKay @davemckay explaining how the IBM Q experience works.
3A jaygambetta IBM Staff
Published 4 months ago
quantum computing

A Benchmarking Metric for a Quantum Computer: the Quantum Volume
Learn more about the quantum volume metric for short-depth quantum computing from our very own Dr. Lev Bishop (@levsbishop)
3M jmchow IBM Staff
Published 4 months ago
quantum volume benchmarking

A Mechanical Qubit
Mun Kunch
Published 4 months ago
quantum computing

The qubit
Published 4 months ago
quantum computing

Quantum Gates
Published 4 months ago
quantum computing



OpenQASM

OpenQASM (Quantum Assembly Language)

```
1 include "qelib1.inc";
2 qreg q[5];
3 creg c[5];
4
5 x q[0];
6 x q[2];
7 ccx q[0],q[2],q[3];
8 measure q[3] -> c[0];
9 if(c==1) x q[3];
10
```



Import QASM

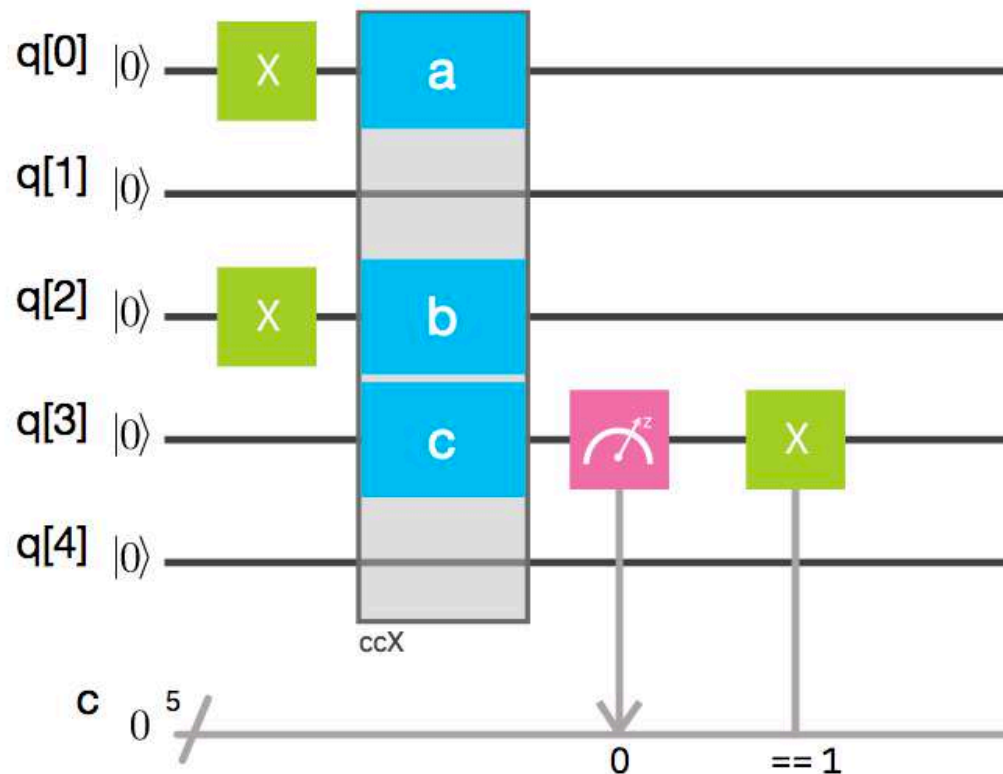


Download QASM

Express *data dependency* but not explicit timing of instructions; separation of quantum and classical processing; hardware agnostic

<https://arxiv.org/abs/1707.03429>

<https://github.com/QISKit/openqasm>



Open to discussions about extensions and modifications!

OpenQASM features



- Define quantum and classical **registers**: `qreg qr[8]; creg cr[8];`
- Apply **built-in unitary** operations **U** and **CX**: `U(pi/2,0,pi) qr[0]; CX qr[0],qr[1];`
- Define additional gates as **subroutines** using combinations of **U** and **CX**:

```
gate swap a,b {      //swap the quantum states of qubits a and b
    CX a,b;
    CX b,a;
    CX a,b;
}
```
- **Include** subroutines defined in other files: `include "qelib1.inc"`
- Perform register-level operations: `h qr; CX qra,qrb;`
- Measure qubits: `measure qr[0] -> cr[0];`
- Use barriers to limit compiler optimizations: `x qr[0]; barrier qr[0]; x qr[0];`
- Apply classically conditioned operations: `if (cr[0]==1) { x qr[1]; }`



QISKIT API

QISKit API: an interface to quantum hardware

▪ **Submit requests** for processing by quantum hardware

- Name of quantum backend to use
- Quantum circuit(s) to run, in QASM format
- # of trials (“shots”) to run

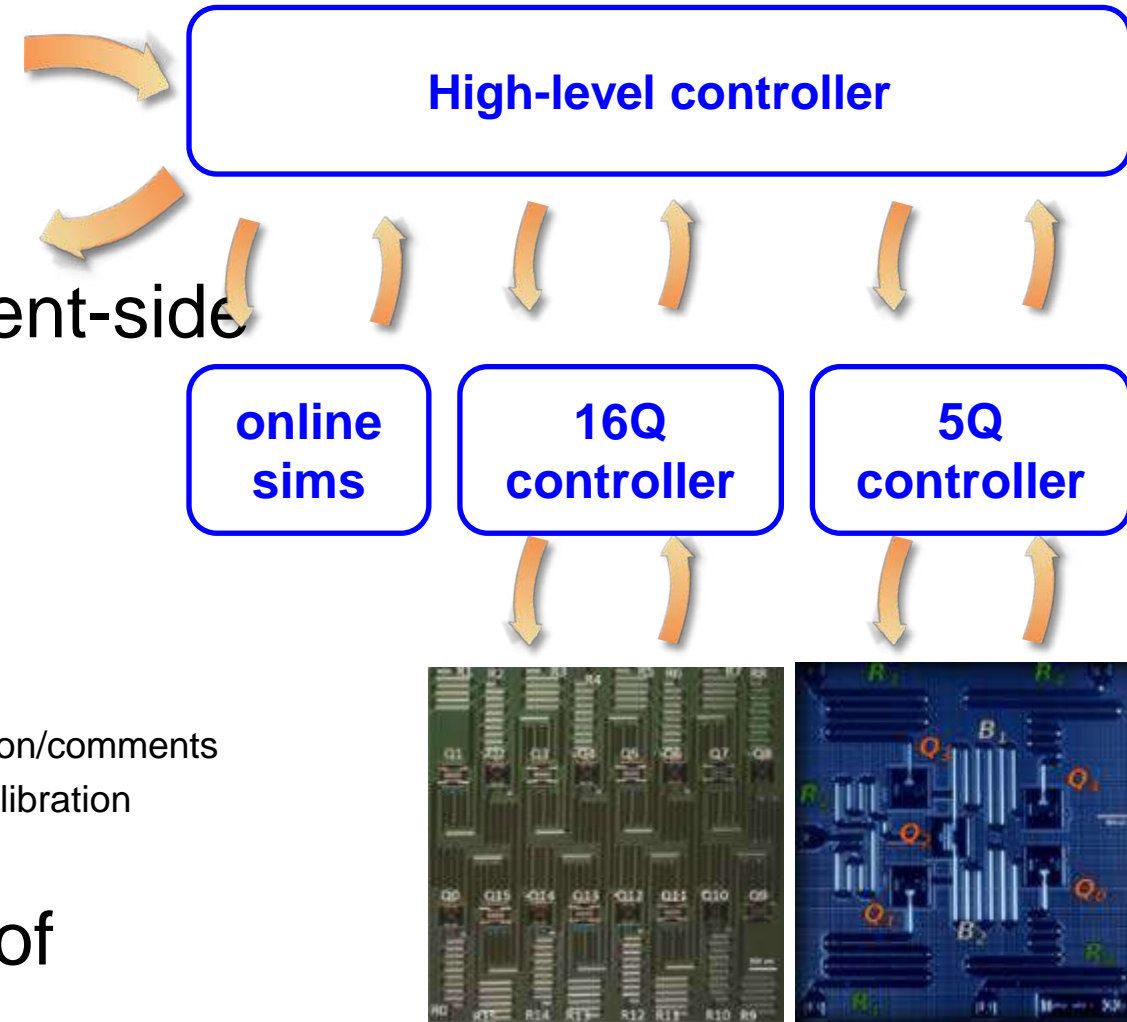
▪ **Retrieve results** and metadata for client-side processing

- Probability of each outcome
- Execution time and duration
- Most recent calibration data at time of execution

▪ **Get backend details**

- Static properties: device type, qubit coupling map, basis gates, description/comments
- Dynamic properties: coherence times, operation fidelities, time of last calibration
- Status: availability, length of job queue

▪ **Token-based authentication, tracking of “credits”**



QISKit API: connecting to the IBM Q Experience

- **Connect** the QISKit API (see github.com/QISKit/qiskit-api-py) to the IBM Q Experience

```
In [1]: import Qconfig
```

```
In [2]: from IBMQuantumExperience.IBMQuantumExperience import IBMQuantumExperience
```

```
In [3]: api = IBMQuantumExperience(Qconfig.APIToken,Qconfig.config)
```

- Get a list of IBM Q Experience **backends** currently online

```
In [4]: backends = api.available_backends()
```

```
In [5]: [backend['name'] for backend in backends if  
....: api.backend_status(backend['name'])['available'] is True]
```

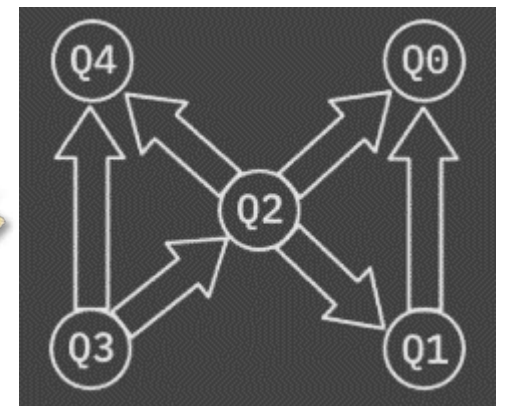
```
Out[5]: ['ibmqx4', 'ibmqx5', 'ibmqx_qasm_simulator']
```

- Get **details** about a given backend

```
In [6]: backends['name']=='ibmqx4']
```

```
Out[6]:
```

```
{'basisGates': 'SU2+CNOT',  
 'chipName': 'Raven',  
 'couplingMap': [[1, 0], [2, 0], [2, 1], [2, 4], [3, 2], [3, 4]],  
 'description': '5 qubit transmon bowtie chip 3',  
 'id': 'c16c5ddebbf8922a7e2a0f5a89cac478',  
 'nQubits': 5, ...}
```



QISKit API: making a Bell state

■ Submit job with QASM for making Bell state

```
In [8]: api.run_job([{'qasm': Bell_QASM}], backend = 'ibmqx4', shots = 1000)
Out[8]: {'backend': {'id': 'c16c5ddebbf8922a7e2a0f5a89cac478', 'name': 'ibmqx4'},
         'creationDate': '2017-11-26T05:23:54.746Z',
         'deleted': False,
         'id': 'ebdaee522b43b171dee105262ad6cc2e',
         'infoQueue': {'status': 'EXECUTING'},
         ...}
```

```
In [7]: Bell_QASM = input()

....: OPENQASM 2.0;
....: include "qelib1.inc";
....: qreg q[5];
....: creg c[5];
....: h q[1];
....: cx q[1],q[0];
....: measure q[0] -> c[0];
....: measure q[1] -> c[1];
```

■ Check job status

```
In [9]: api.get_job('ebdaee522b43b171dee105262ad6cc2e')['status']
Out[9]: 'COMPLETED'
```

■ Retrieve results

```
In [10]: api.get_job('ebdaee522b43b171dee105262ad6cc2e')['qasms'][0]['data']['counts']
Out[10]: {'00000': 500, '00001': 26, '00010': 64, '00011': 410}
```

→ As expected, 00 and 11 appear with similar probabilities
and 01 and 10 are suppressed

That's great, but...

- What if I want to work with many qubits?
- What if I want to optimize my program to maximize fidelity?
- What if I want to assemble complex circuits from simple ones?
- What if I want to use a high level language to construct my circuits?
- What if I want to avoid certain qubits based on device calibration data?
- What if I want to run the same circuit on completely different quantum hardware?
- What if ?



QISKIT SDK

QISKit Documentation

Quantum Information Software Kit (QISKit), SDK Python version for working with [OpenQASM](#) and the IBM Q experience (QX).

Table Of Contents

Installation and setup
Getting started
QISKit overview
Developer documentation
SDK reference

Next topic

Installation and setup

This Page

Show Source

Table of Contents

- [Installation and setup](#)
 - [Installation](#)
 - [Install Jupyter-based tutorials](#)
 - [FAQ](#)
- [Getting started](#)
 - [Quantum Chips](#)
 - [Project Organization](#)
- [QISKit overview](#)
 - [Philosophy](#)
 - [Project Overview](#)
- [Developer documentation](#)
 - [Programming interface](#)

Documentation from getting
started to developing

 [QISKit / qiskit-sdk-py](#) 

 Watch

151

 Star

909

 Fork

261

 Code

 Issues 15

 Pull requests 8

 Boards

 Reports

 Projects 0

 Wiki

 Insights

Python software development kit for writing quantum computing experiments, programs, and applications.

<http://www.qiskit.org>

[quantum-computing](#)

[qiskit](#)

[sdk](#)

[python](#)


[quantum-programming-language](#)

Open development process

 1,080 commits

 5 branches

 10 releases

 35 contributors

 Apache-2.0

Branch: **master** ▾

[New pull request](#)

[Find file](#)

[Clone or download](#) ▾

 **westurner** committed with **diego-plan9** Update README.md Jupyter Notebook(s) (#176) ...

Latest commit d2e9c2d 3 hours ago

 [.github](#)

[Add templates for issues and pull requests](#)

5 months ago

PYTHON

qiskit-sdk-py

Python software development kit for writing quantum computing experiments, programs, and applications

python sdk quantum-computing quantum-programming-language

Python ★ 905 258 Apache-2.0 Updated 4 hours ago

qiskit.github.io

QISKit landing page

HTML ★ 2 5 Updated 4 hours ago

qiskit-sdk-swift

Swift ★ 8 1 Apache-2.0 Updated 6 hours ago

openqasm

Gate and operation specification for quantum circuits

quantum-computing quantum-information qiskit ibmqx

TeX ★ 134 39 Apache-2.0 Updated 5 days ago

qiskit-sdk-js

Quantum Information Software Kit in pure JavaScript.

JavaScript ★ 2 Apache-2.0 Updated 7 days ago

Top languages

Python HTML JavaScript
TeX Swift

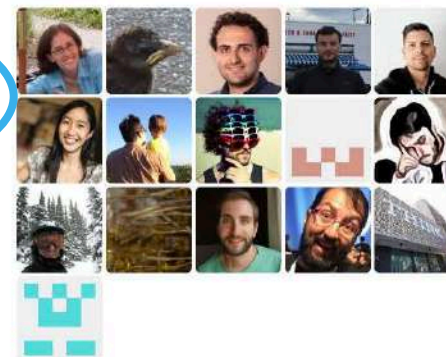
Most used topics

Manage

quantum-computing qiskit
ibmqx python

People

30 >



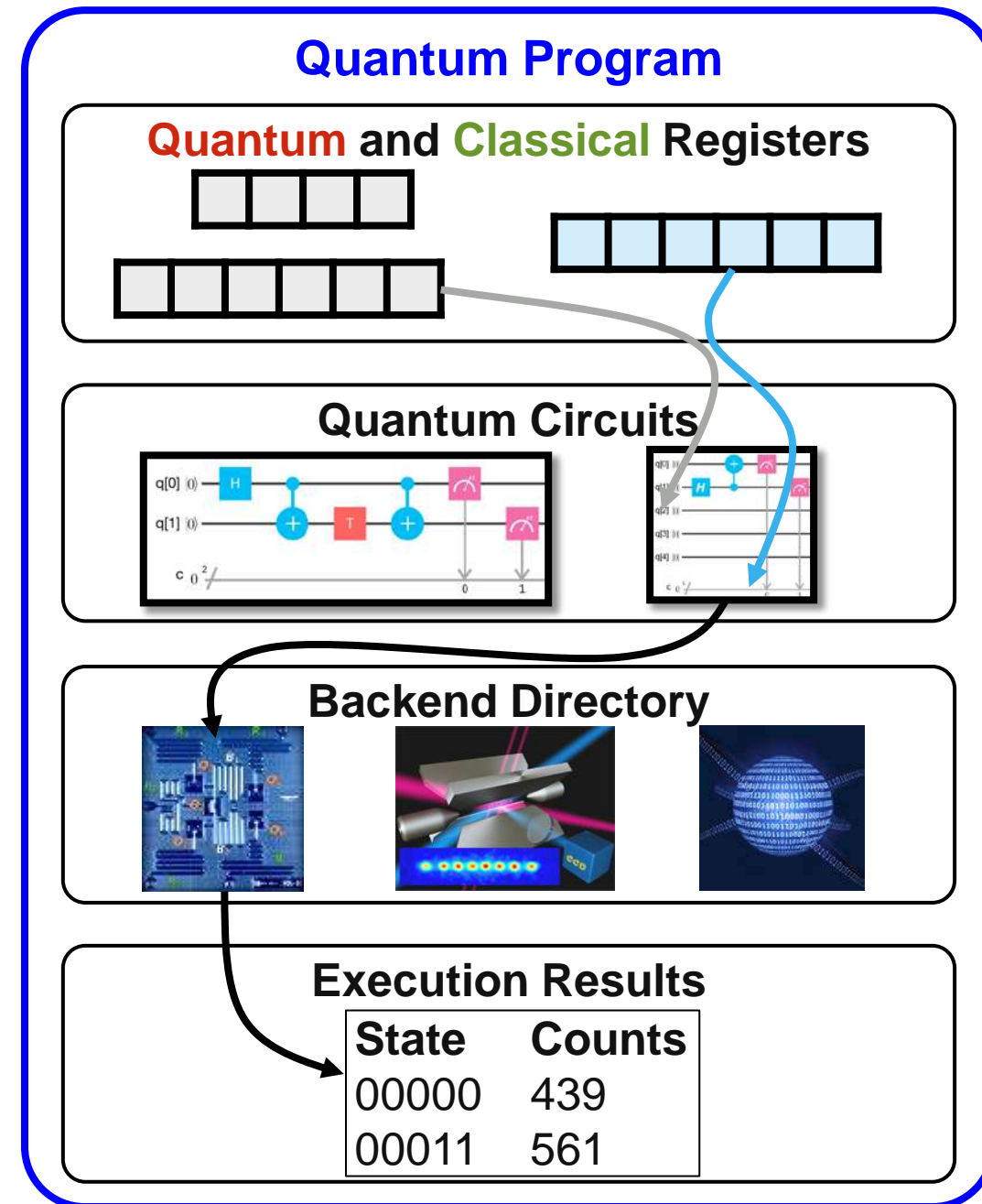
SWIFT

JAVASCRIPT

QISKit SDK

- Goal: enable research and further development of applications for near-term quantum backends
- Central components:
 - **Quantum Program** class (see illustration)
 - Quantum circuit **transcompiler**
 - Quantum circuit **backends**
- Typical workflow:
 1. Initialize quantum program
 2. Define quantum and classical registers
 3. Build quantum circuits
 4. Rewrite circuits to run on target backend
 5. Execute job
 6. Analyze results

github.com/QISKit/qiskit-sdk-py



QISKit: Getting started

Download qiskit-tutorial from <https://github.com/QISKit/qiskit-tutorial>

Install qiskit (optionally download SDK from <https://github.com/QISKit/qiskit-sdk-py>)

Navigate to qiskit-tutorial folder and launch Jupyter notebook

```
1. cjwood@christophers-MacBook-Pro: ~/Documents/IBM-Git/qiskit-tutorial  
→ qiskit-tutorial git:(master) x pip install qiskit; jupyter notebook
```

Create a new Python 3 notebook and import qiskit

```
In [1]: # Import QISKit  
import qiskit  
from qiskit import QuantumProgram # basic QISKit object  
  
# Add IBMQX API token and URL. Needed for online access  
API_TOKEN = "your_quantum_experience_api_token_here"  
API_URL = 'https://quantumexperience.ng.bluemix.net/api'
```

Initializing a quantum program

The main interface to QISKit is the **QuantumProgram** class.

- Collection of quantum circuits and methods to interact with them
- Build and store quantum circuits
- Import or export OpenQASM text circuits
- Interface with backends to run experiments (on real hardware or simulators)

Basic steps to initialize a new program

1. Create a new QuantumProgram
2. Add 1 or more quantum registers
3. Add 1 or more classical registers

```
In [2]: # Initialize a new quantum program
qp = QuantumProgram()

# Add a 2-qubit quantum register "qr"
qr = qp.create_quantum_register("qr", 2)
|
# Add a 2-bit register "cr" to record results
cr = qp.create_classical_register("cr", 2)
```

Bell state with QISKit: building a basic circuit

Create quantum program and associated registers

Define a circuit to **prepare** a Bell state

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Define a circuit to **measure** both qubits in the default (Z) basis

```
In [1]: from qiskit import QuantumProgram
```

```
In [2]: qp = QuantumProgram()
```

```
In [3]: qr = qp.create_quantum_register('qr',2)
```

```
In [4]: cr = qp.create_classical_register('cr',2)
```

```
In [5]: bell = qp.create_circuit('Bell',[qr],[cr])
```

```
In [6]: bell.h(qr[0])
```

```
In [7]: bell.cx(qr[0],qr[1])
```

```
In [8]: meas_z = qp.create_circuit('measZ',[qr],[cr])
```

```
In [9]: meas_z.measure(qr,cr)
```

qiskit.extensions.standard
defines methods for common operations (similar to **qelib1.inc**).
Need others? Add an extension!

What's in the standard extension?

Available circuit operation methods:

- Single qubit gates
 - *iden, x, y, z, h, s, sdg, t, tdg, u1, u2, u3, rx, ry, rz*
- Two qubit gates (*cx, cy, cz, cu1, cu2*)
- Three qubit gates (*ccx, cswap*)
- Measurement, reset, and barrier (*measure, reset, barrier*)

Additional circuit construction methods:

- Invert gates with *.inverse*
 - *mycirc.u1(pi/8).inverse()*
- Add a classical control with *.c_if*
 - *mycirc.x(q[0]).c_if(outcome, 1)*



Bell state with QISKit: getting some information

Get quantum circuit and program register names

```
In [8]: qp.get_circuit_names()
```

```
Out[8]: dict_keys(['Bell', 'measZ'])
```

```
In [9]: qp.get_quantum_register_names()
```

```
Out[9]: dict_keys(['qr'])
```

```
In [10]: qp.get_classical_register_names()
```

```
Out[10]: dict_keys(['cr'])
```

Get OpenQASM text for the quantum circuits

```
In [11]: for qasm in qp.get_qasms(qp.get_circuit_names()):  
         print(qasm)
```

```
OPENQASM 2.0;  
include "qelib1.inc";  
qreg qr[2];  
creg cr[2];  
h qr[0];  
cx qr[0],qr[1];
```

```
OPENQASM 2.0;  
include "qelib1.inc";  
qreg qr[2];  
creg cr[2];  
measure qr[0] -> cr[0];  
measure qr[1] -> cr[1];
```

Bell state with QISKit: harnessing circuit modularity

- For more evidence the qubits are actually entangled, **measure in a different basis** and **verify that the correlation persists**
- Make a circuit for measuring in the X basis:

```
In [10]: meas_x = qp.create_circuit('measX',[qr],[cr])
```

```
In [11]: meas_x.h(qr)
```

```
In [12]: meas_x.measure(qr,cr)
```

- Now add two new circuits to our Quantum Program, each made by combining the Bell state preparation circuit with one of the measurement circuits:

```
In [13]: qp.add_circuit('Bell_measZ',bell + meas_z)
```

```
In [14]: qp.add_circuit('Bell_measX',bell + meas_x)
```

Bell state with QISKit: inquiring about backends

Request backend names from the program object

```
In [12]: qp.available_backends()
```

```
Out[12]: ['local_qasm_cpp_simulator', 'local_qasm_simulator', 'local_unitary_simulator']
```

Configure the API to access the online backends

```
In [14]: qp.set_api(Qconfig.APIToken, Qconfig.config["url"])
qp.available_backends()
```

```
Out[14]: ['ibmqx4',
          'ibmqx5',
          'ibmqx2',
          'ibmqx_qasm_simulator',
          'local_qasm_cpp_simulator',
          'local_qasm_simulator',
          'local_unitary_simulator']
```

```
In [15]: qp.online_backends()
```

```
Out[15]: ['ibmqx4', 'ibmqx5', 'ibmqx2', 'ibmqx_qasm_simulator']
```

Bell state with QISKit: execution

- Send both circuits to the IBM Q Experience's 5-qubit chip for execution:

```
In [15]: qp.set_api(API_TOKEN, 'https://quantumexperience.ng.bluemix.net/api')
```

```
In [16]: cmap = qp.get_backend_configuration('ibmqx4')['coupling_map']
```

```
In [17]: result = qp.execute(['Bell_measZ', 'Bell_measX'], backend='ibmqx4',  
    ...: coupling_map=cmap, shots=1000)
```

- ... and in a couple minutes:

```
In [18]: result.get_counts('Bell_measZ')
```

```
Out[18]: {'00000': 490, '00001': 32, '00010': 65, '00011': 413}
```

```
In [19]: result.get_counts('Bell_measX')
```

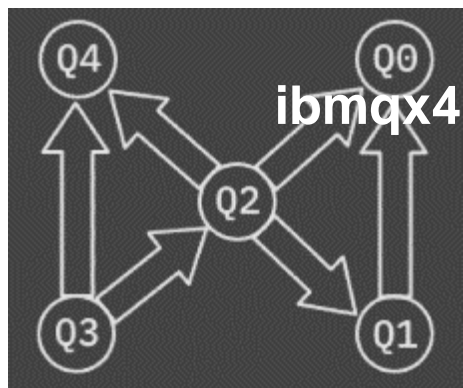
```
Out[19]: {'00000': 501, '00001': 45, '00010': 56, '00011': 398}
```

- Indeed, outcomes are correlated regardless of the choice of measurement basis



Bell state with QISKit: circuit rewriting

■ Q: But how did these circuits even run at all?!



Wrong direction!

From qiskit.extensions.standard.cx:

```
def cx(self, ctl, tgt):  
    """Apply CNOT from ctl to tgt."""
```

```
In [7]: bell.cx(qr[0],qr[1])
```

■ A: Execute = **compile** + run

```
In [20]: print(qp.get_qasm('Bell_measZ'))
```

```
OPENQASM 2.0;
```

```
include "qelib1.inc";
```

```
qreg qr[2];
```

```
creg cr[2];
```

```
h qr[0];
```

```
cx qr[0],qr[1];
```

```
measure qr[0] -> cr[0];
```

```
measure qr[1] -> cr[1];
```

Rewrite gates in backend's basis

Re-map circuit given coupling map

```
In [21]: print(result.get_ran_qasm('Bell_measZ'))
```

```
OPENQASM 2.0;
```

```
include "qelib1.inc";
```

```
qreg q[2];
```

```
creg cr[2];
```

```
u2(0.0,3.141592653589793) q[1];
```

```
cx q[1],q[0];
```

```
measure q[0] -> cr[1];
```

```
measure q[1] -> cr[0];
```

■ Transcompiler goals:

- **Map** a given circuit into one that can be run on target backend
- **Optimize** circuit performance by eliminating redundancies in instruction sequences

Nuts and bolts: basic circuit rewriting

- The circuit rewriting methods (mapper module) execute a few simple fixed passes
 - 1. “*unroll*”: expands gate definitions to some level, expands loops
 - 2. “*swap_mapper*”: selects a layout and inserts SWAP gates as needed
 - 3. “*cx_cancellation*”: removes even runs of CNOT gates
 - 4. “*optimize_1q_gates*”: simplifies runs of single qubit gates
- SWAP insertion algorithm solves using a randomized, greedy layer-by-layer approach
- Single qubit gate optimization attempts to minimize number of pulses
- **Modular framework in development to enable extensibility and research**

Bell state with QISKit: local Python simulators

Often we would like to examine the expected quantum state

- Using the simulator in QISKit we may “cheat” and ask directly for the state

Run these examples on circuits that *don't contain measurement*

```
In [18]: results = qp.execute('Bell', backend='local_qasm_simulator', shots=1)
data = results.get_data('Bell')
print(data)

{'counts': {'00': 1}, 'quantum_state': array([ 0.70710678+0.j,  0.00000000+0.j,  0.00000000+0.j,  0.70710678+0.j]), '
classical_state': (0,)}
```

```
In [19]: results = qp.execute('Bell', backend='local_unitary_simulator', shots=1)
data = results.get_data('Bell')
print(data)

{'unitary': array([[ 0.70710678 +0.00000000e+00j,  0.70710678 -8.65956056e-17j,
                    0.00000000 +0.00000000e+00j,  0.00000000 +0.00000000e+00j],
                  [ 0.00000000 +0.00000000e+00j,  0.00000000 +0.00000000e+00j,
                    0.70710678 +0.00000000e+00j, -0.70710678 +8.65956056e-17j],
                  [ 0.00000000 +0.00000000e+00j,  0.00000000 +0.00000000e+00j,
                    0.70710678 +0.00000000e+00j,  0.70710678 -8.65956056e-17j],
                  [ 0.70710678 +0.00000000e+00j, -0.70710678 +8.65956056e-17j,
                    0.00000000 +0.00000000e+00j,  0.00000000 +0.00000000e+00j]])}
```

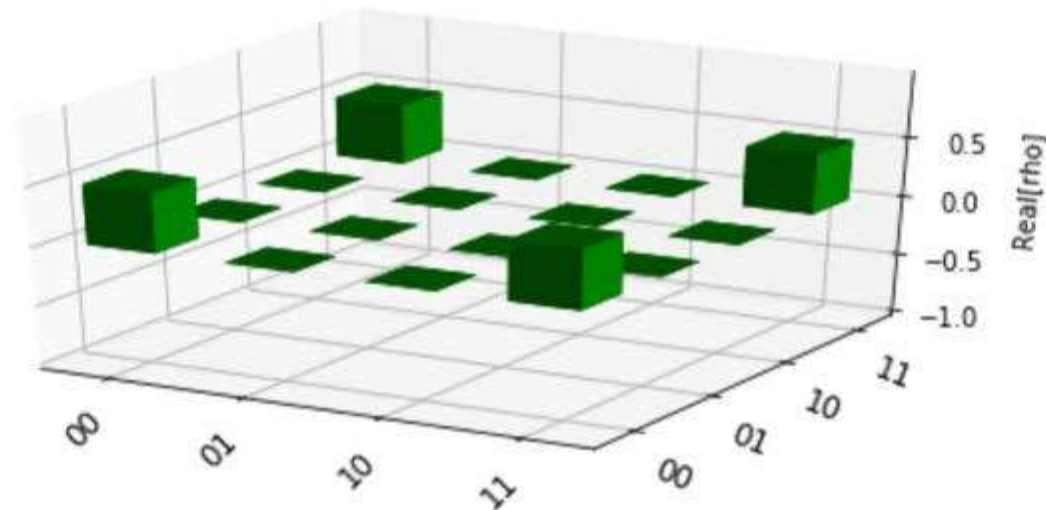
Bell state with QISKit: plotting states

Plotting a state using the visualization module:

- The **qiskit.tools.visualization** model contains several methods of visualizing quantum states:

```
In [10]: # Import QISKit visualization library
from qiskit.tools.visualization import plot_state
from qiskit.tools.qi.qi import outer

# Plot the density matrix of the state
rho = outer(data['quantum_state']) # convert to density matrix
plot_state(rho, method='city')
```



Example of advanced QISKit features

- To fully determine a 2-qubit state experimentally, we need to obtain counts from 9 different measurement circuits corresponding to the Pauli group elements, i.e. we need to do **quantum state tomography**.

Is there an easier way to implement this?

- Yes! We have the full power of Python, so write functions that automate necessary steps
- Many useful routines are already implemented in QISKit modules, and more are coming!

Bell state with QISKit: state tomography

- We can implement quantum state tomography using the QISKit tomography module:

```
In [15]: import qiskit.tools.qcvtv.tomography as tomo|
```

- This automates generating of measurement circuits, and reconstructing the density matrix:
- **Generate measurement circuits:**

```
In [17]: meas_qubits = [0,1]
         tomo_circs = tomo.build_state_tomography_circuits(qp, 'bell',
                                                         meas_qubits, qr, cr)
         print(tomo_circs)|

>> created state tomography circuits for "bell"
['bell_measX0X1', 'bell_measX0Y1', 'bell_measX0Z1', 'bell_measY0X1', 'bell_measY0Y1', 'bell_measY0Z1', 'bell_measZ0X1', 'bell_measZ0Y1', 'bell_measZ0Z1']
```


Bell state with QISKit: state tomography

- Execute the circuits on a simulator to obtain count results:

```
In [19]: backend = 'local_qasm_simulator'
shots = 1024
tomo_res = qp.execute(tomo_circs, backend=backend, shots=shots)

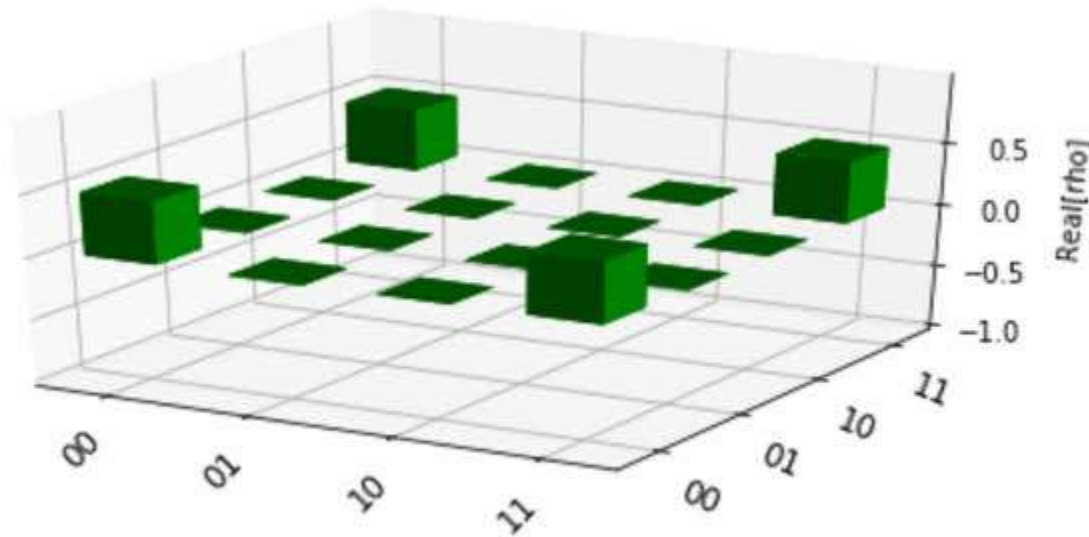
for c in tomo_circs:
    print(tomo_res.get_counts(c))

{'11': 532, '00': 492}
{'01': 243, '11': 255, '00': 263, '10': 263}
{'11': 256, '00': 234, '10': 261, '01': 273}
{'01': 268, '00': 260, '10': 258, '11': 238}
{'10': 519, '01': 505}
{'11': 267, '00': 274, '10': 251, '01': 232}
{'00': 256, '11': 259, '01': 247, '10': 262}
{'10': 252, '01': 249, '00': 259, '11': 264}
{'11': 494, '00': 530}
```

Bell state with QISKit: state tomography

- Post-process data to reconstruct state

```
In [20]: # extract state tomography data
tomo_data = tomo.state_tomography_data(tomo_res, 'bell', meas_qubits)
# Fit the state using Maximum Likelihood estimation
rho_fit = tomo.fit_tomography_data(tomo_data)
# Plot the state
plot_state(rho_fit, method='city')
```

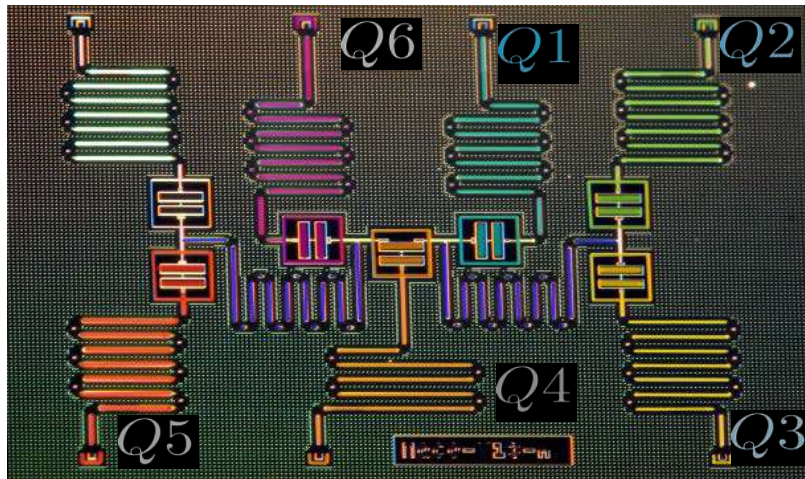




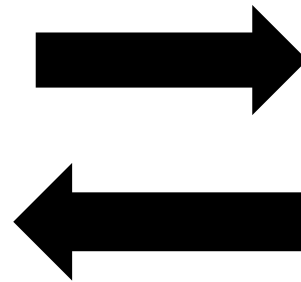
VQE algorithm: Application to quantum chemistry

The latest version of this notebook is available on <https://github.com/QISKit/qiskit-tutorial>.

For some physical Hamiltonian H , find the smallest eigenvalue E_G , such that $H|\psi_G\rangle = E_G|\psi_G\rangle$, where $|\psi_G\rangle$ is the eigenvector corresponding to E_G .



**Prepare a trial state $|\psi(\theta)\rangle$
and compute its energy $E(\theta)$**



**Use classical optimizer to choose
a new value of θ to try**

Approximate universal quantum computing for quantum chemistry problems

In order to find the optimal parameters θ^* , we set up a closed optimization loop with a quantum computer, based on some stochastic optimization routine. Our choice for the variational ansatz is a deformation of the one used for the optimization of classical combinatorial problems, with the inclusion of Z rotation together with the Y ones. The optimization algorithm for fermionic Hamiltonians is similar to the one for combinatorial problems, and can be summarized as follows:

1. Map the fermionic Hamiltonian H to a qubit Hamiltonian H_P .
2. Choose the maximum depth of the quantum circuit (this could be done adaptively).
3. Choose a set of controls θ and make a trial function $|\psi(\theta)\rangle$. The difference with the combinatorial problems is the insertion of additional parametrized Z single-qubit rotations.
4. Evaluate the energy $E(\theta) = \langle \psi(\theta) | H_P | \psi(\theta) \rangle$ by sampling each Pauli term individually, or sets of Pauli terms that can be measured in the same tensor product basis.
5. Use a classical optimizer to choose a new set of controls.
6. Continue until the energy has converged, hopefully close to the real solution θ^* and return the last value of $E(\theta)$.

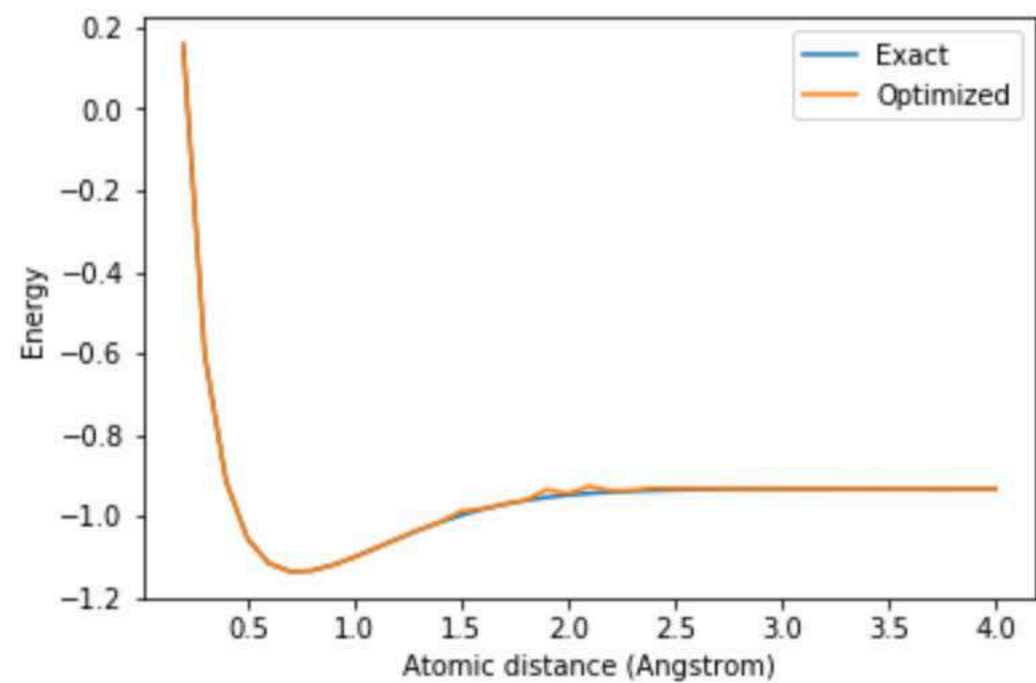
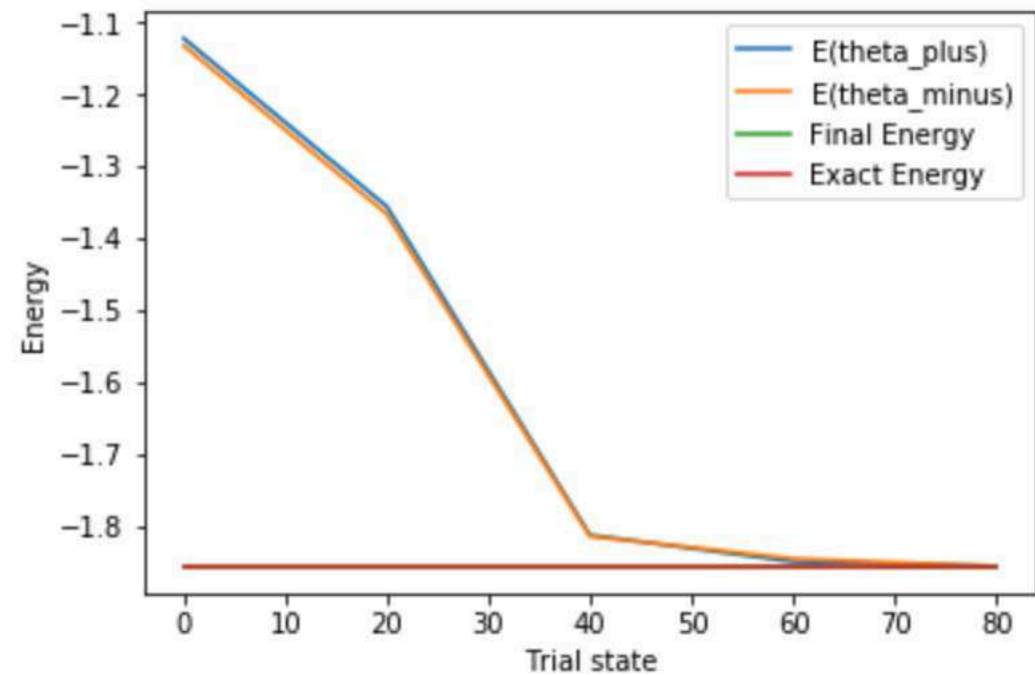
Note that, as opposed to the classical case, in the case of a quantum chemistry Hamiltonian one has to sample over non-computational states that are superpositions, and therefore take advantage of using a quantum computer in the sampling part of the algorithm. Motivated by the quantum nature of the answer, we also define a variational trial ansatz in this way:

$$|\psi(\theta)\rangle = [U_{\text{single}}(\theta)U_{\text{entangler}}]^m|+\rangle$$

where $U_{\text{entangler}}$ is a collection of cPhase gates (fully entangling gates), $U_{\text{single}}(\theta) = \prod_{i=1}^n Y(\theta_i)Z(\theta_{n+i})$ are single-qubit Y and Z rotation, n is the number of qubits and m is the depth of the quantum circuit.

References and additional details:

[1] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, *Hardware-efficient Variational Quantum Eigensolver for Small Molecules and Quantum Magnets*, Nature 549, 242 (2017), and references therein.



Experimental Results (2 qubits): Hydrogen Molecule

$H_A: 1s^1 \quad H_B: 1s^1$

4 spin orbitals mapped to 2 qubits

Equilibrium $d = 0.735 \text{ \AA}$

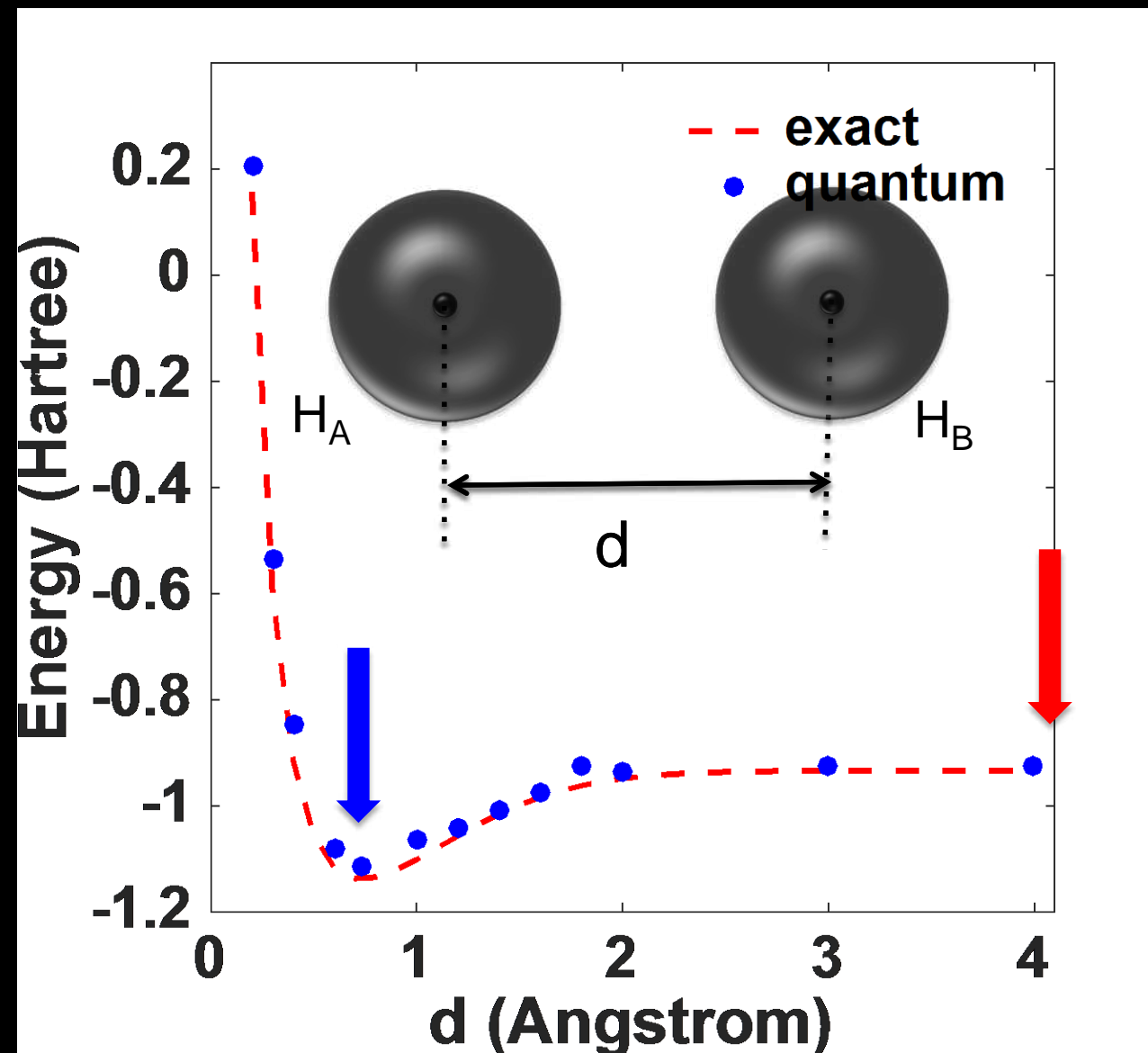
$H = (-1.05237)II + (0.39735)ZI + (0.39735)IZ +$
 $(0.11279)ZZ + (0.18093)XX$

Dissociation $d = 4 \text{ \AA}$

$H = (-0.70461)II + (0.00012)ZI + (0.00012)IZ +$
 $(1.6673e-10)ZZ + (0.33438)XX$

	Equilibrium ($d=0.735 \text{ \AA}$)	Dissociation ($d=4 \text{ \AA}$)
Exact	-1.858	-1.0655
1 U_{ENT}	-1.8365	-1.0595
2 U_{ENT}	-1.8229	-1.0586

arXiv:1704.05018



Toolboxes and tutorials

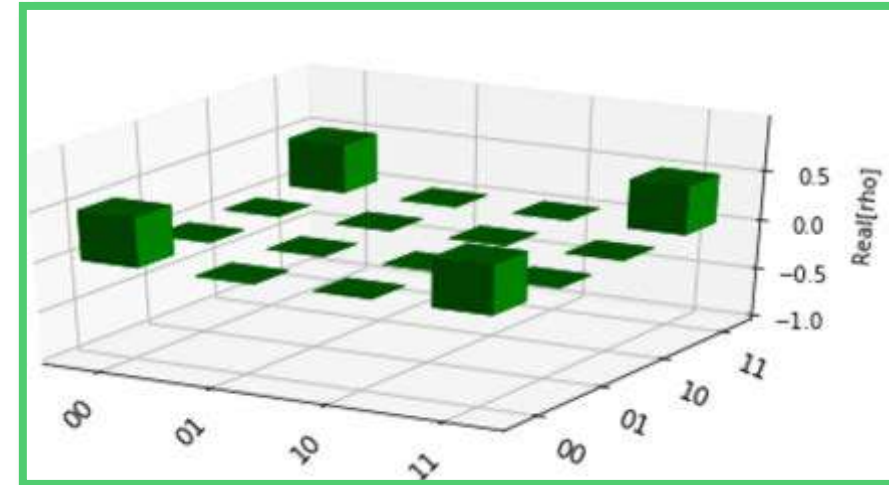
■ **Toolboxes** (part of SDK): libraries of helper functions for...

- Quantum state visualization and analysis
- Optimization problems
- Quantum chemistry problems
- Verification and validation
- File I/O

■ **Tutorials** (github.com/QISKit/qiskit-tutorial): Jupyter notebooks illustrating concepts, usage, applications, etc.

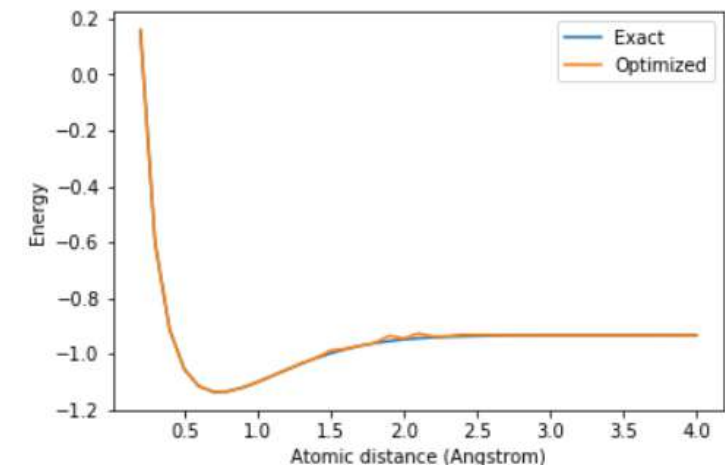
- Currently 24 notebooks spanning 5 categories:
 1. **Introduction** to the tools
 2. Exploring quantum information **concepts**
 3. **Verification** tools for quantum information science
 4. **Applications** of short-depth quantum circuits
 5. Quantum **games**

```
import qiskit.tools.qcqv.tomography as tomo
```



```
In [6]: plt.plot(mol_distance, electr_energy+coulomb_repulsio
if run_optimization:
    plt.plot(mol_distance, electr_energy_optimized+coulomb_repulsio
plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy')
plt.legend()
```

Out[6]: <matplotlib.legend.Legend at 0x1112bfc50>





- Interface for experiments or simulations to enable research and applications pre-fault-tolerance
- Growing software stack, including higher-level tutorials and examples
- Interface to quantum devices: IBM QX and local devices
- Open source development
 - New releases every few months
 - Ongoing projects to improve circuit rewriting architecture, simulators, visualizers, backend interfaces

explore <https://www.qiskit.org/>

contribute <https://github.com/QISKit>

help define <https://qiskit.slack.com/>