Quantum Computing with the **IBM Quantum Experience** with the **Quantum Information Software Toolkit (QISKit)**

Nick Bronn
Research Staff Member
IBM T.J. Watson Research Center

ACM Poughkeepsie Monthly Meeting, January 2018

Overview

Part 1: Quantum Computing

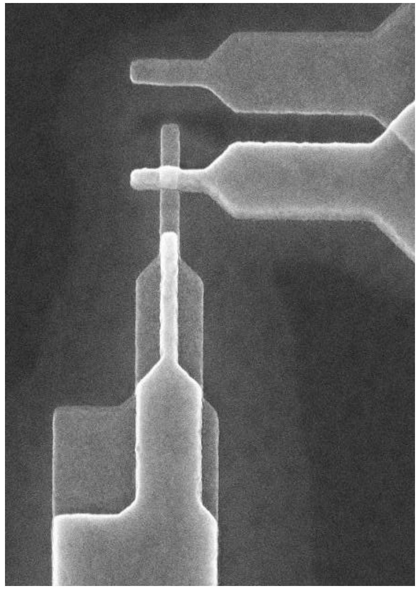
- What, why, how
- Quantum gates and circuits

Part 2: Superconducting Qubits

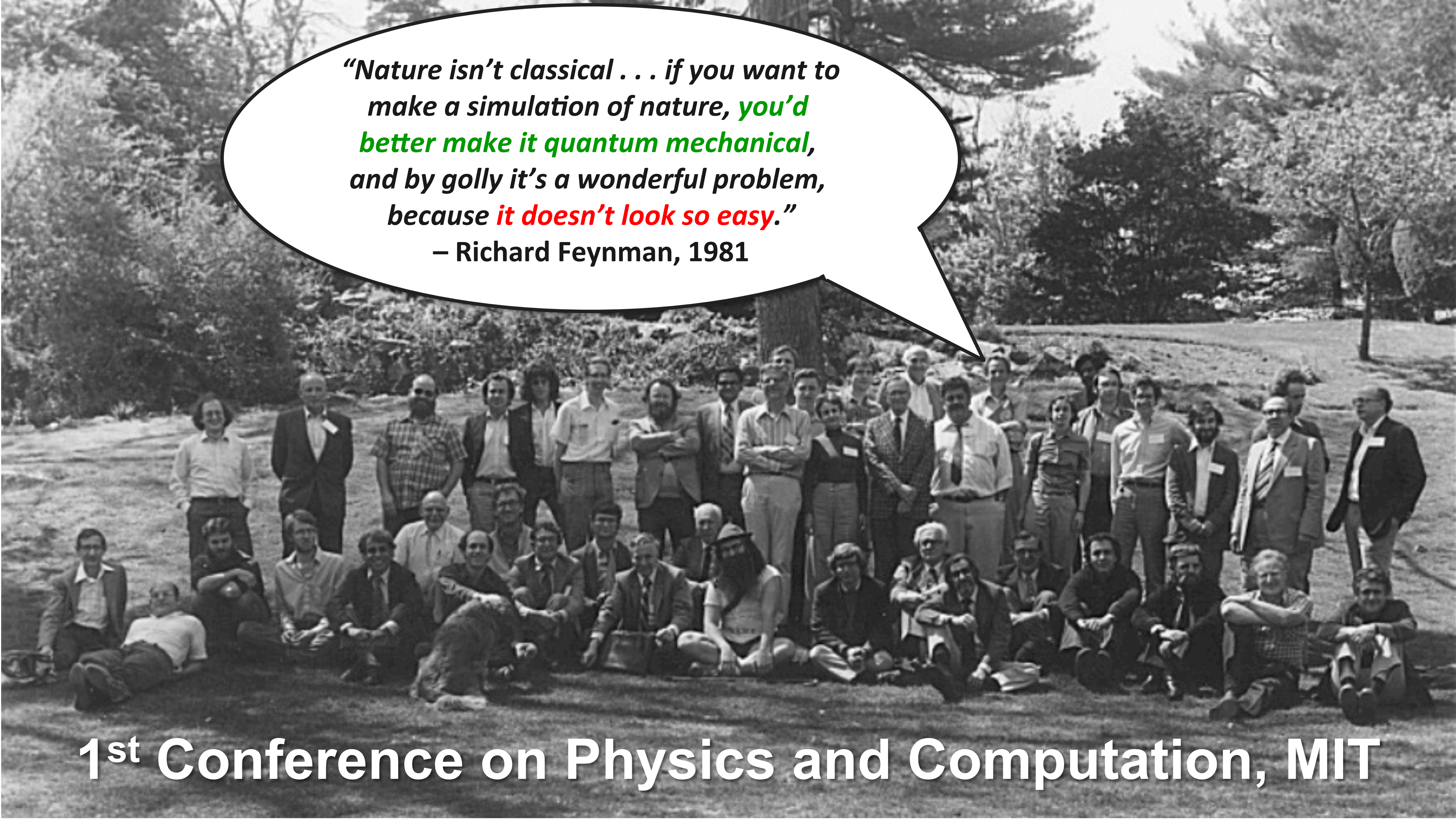
- Device properties
- Control and performance

Part 3: IBM Quantum Experience

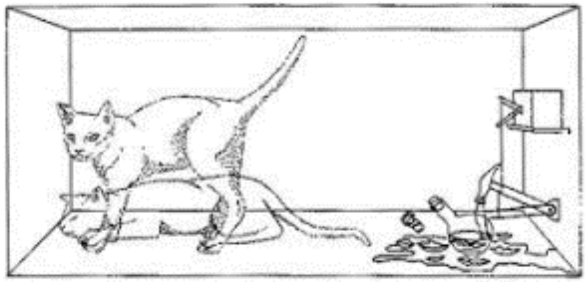
- Website: GUI, user guides, community
- QISKit: API, SDK, Tutorials

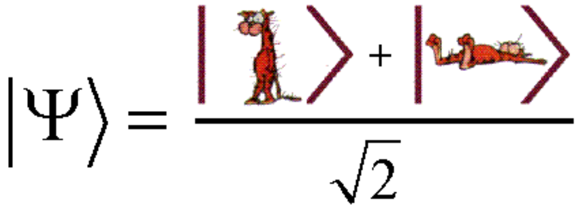


Quantum computing: what, why, how


"Nature isn't classical . . . if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy."
– Richard Feynman, 1981
1st Conference on Physics and Computation, MIT

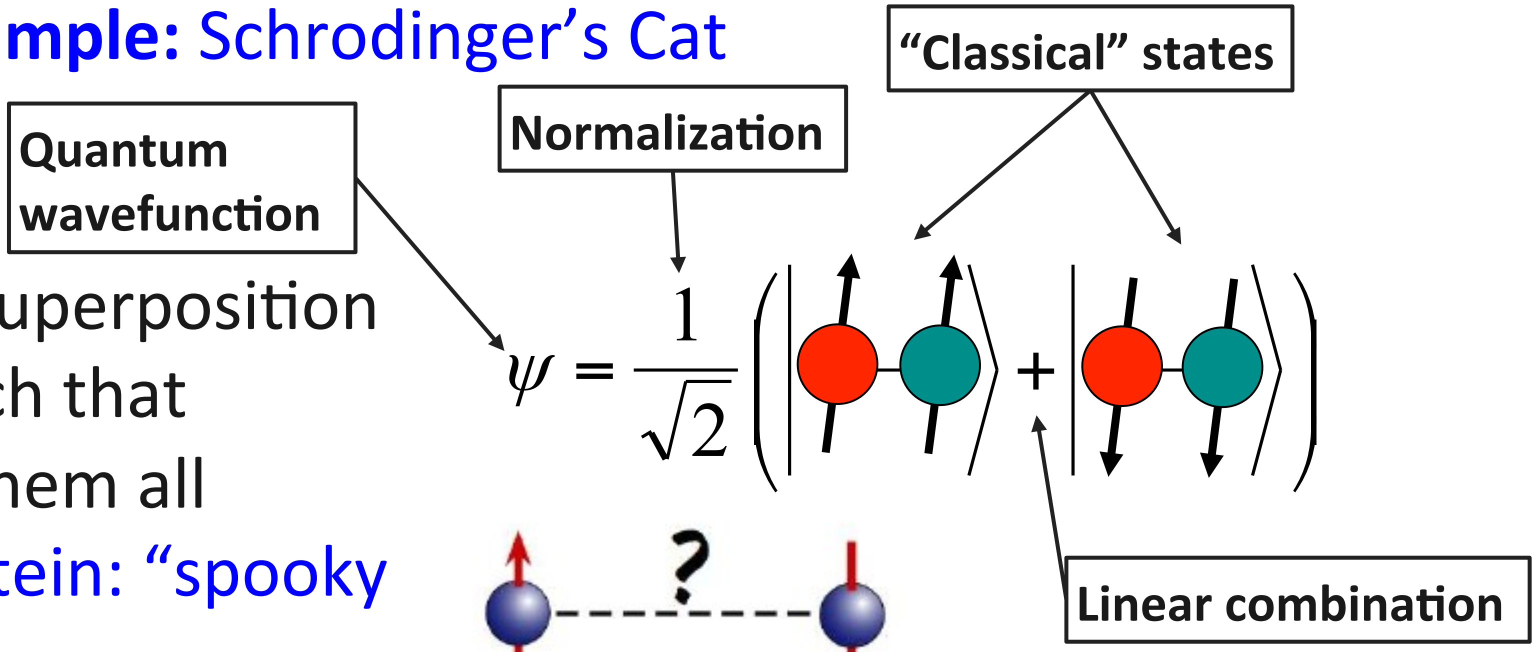
Computing with Quantum Mechanics: Features





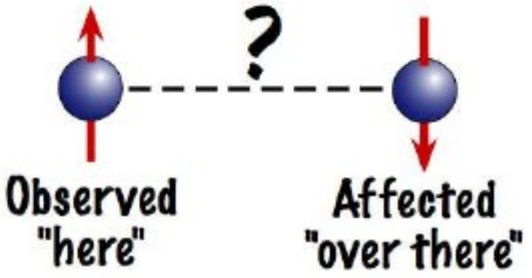
Superposition: a system's state can be any linear combination of classical states *…until it is measured, at which point it collapses to one of the classical states*

Example: Schrodinger's Cat

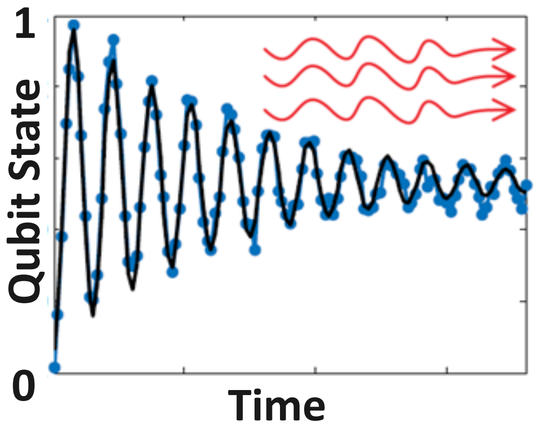


Entanglement: particles in superposition can develop correlations such that measuring just one affects them all

Example: EPR Paradox (Einstein: "spooky action at a distance")



Computing with Quantum Mechanics: Drawbacks



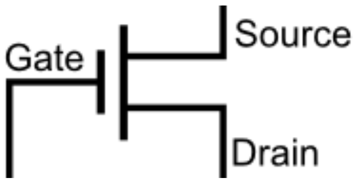
Decoherence: a system is gradually measured by residual interaction with its environment, killing quantum behavior

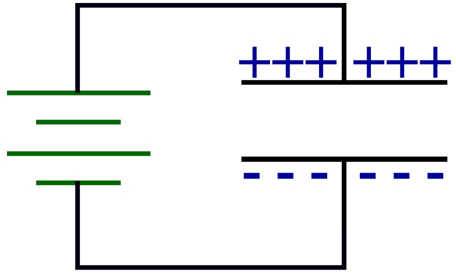
Consequence: quantum effects observed only in well-isolated systems (so not cats… yet)

Uncertainty principle: measuring one variable (e.g. position) disturbs its conjugate (e.g. momentum)

Consequence: complete knowledge of an arbitrary quantum state is impossible.
→ "No-Cloning Theorem"

What does a quantum bit look like?



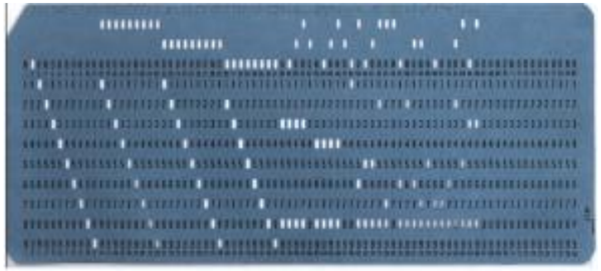


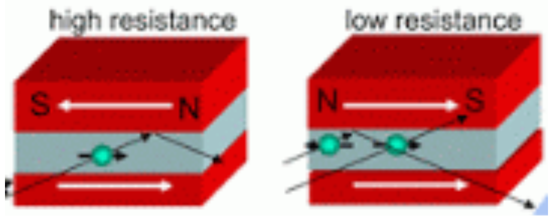
Classical bit

Physical systems: capacitor charge, transistor state, magnetic polarization, presence or absence of a punched hole, etc.

Logical states: **just 0 and 1**

Multi-bit effects: **none**



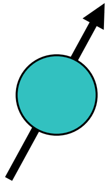


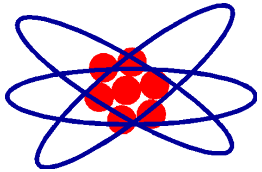
Quantum bit ("qubit")

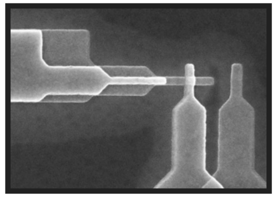
Physical systems: electron spins, atomic states, *superconducting circuit states*

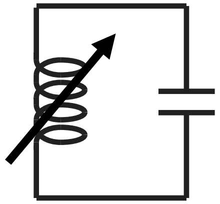
Logical states: |0>, |1>, ***superpositions***

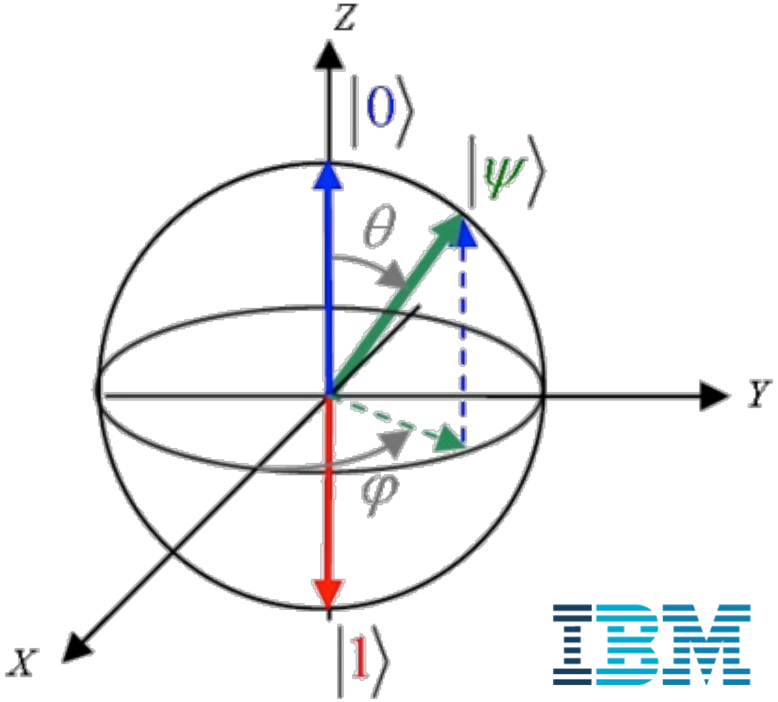
Multi-qubit effects: ***entanglement***



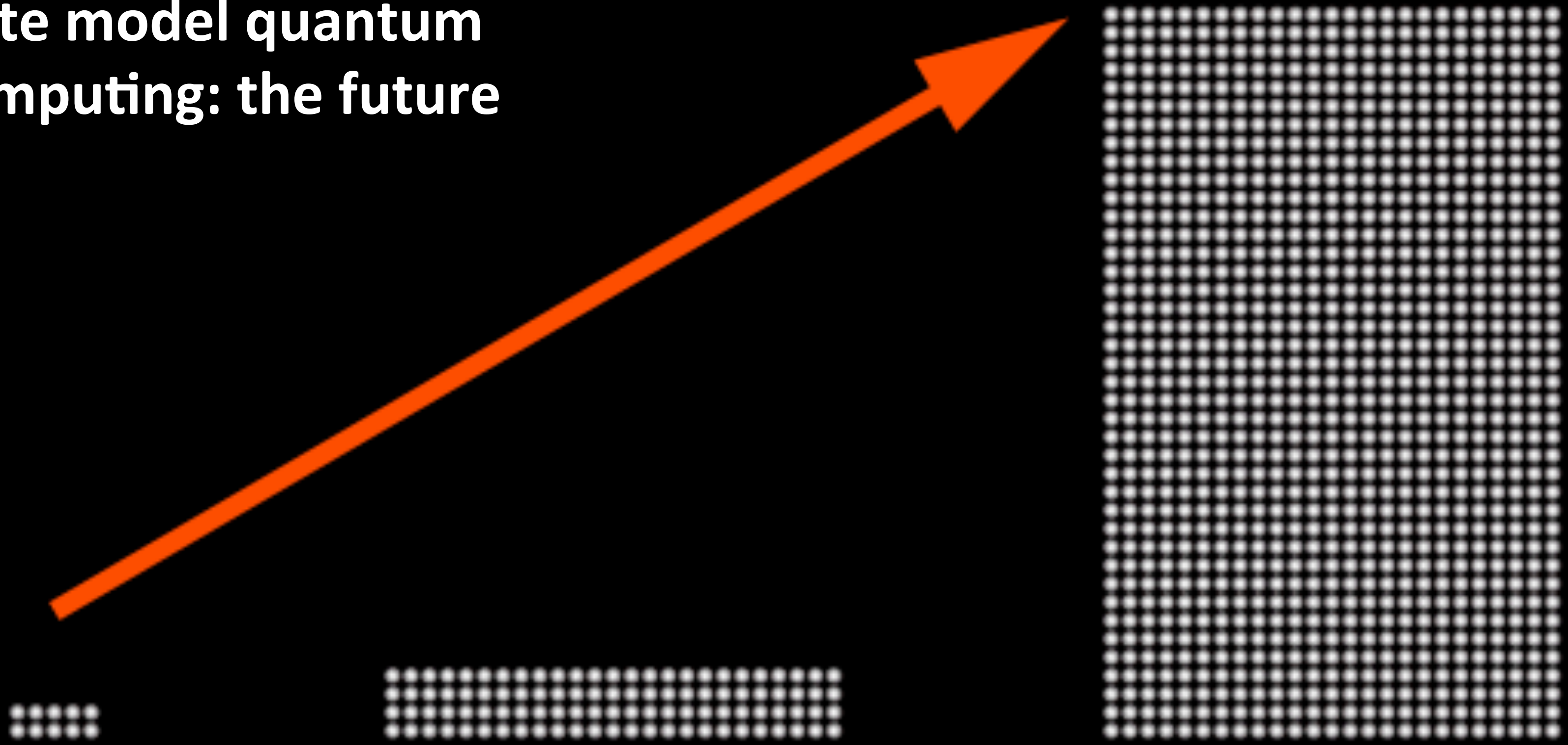








Gate model quantum computing: the future



Fault-Tolerant QC

Today
~ 10 Qubits

Near future:
50-100 qubits too big to simulate!

Future:
Millions of qubits fully fault-tolerant

How powerful is a quantum computer: *quantum volume*

IBM Q

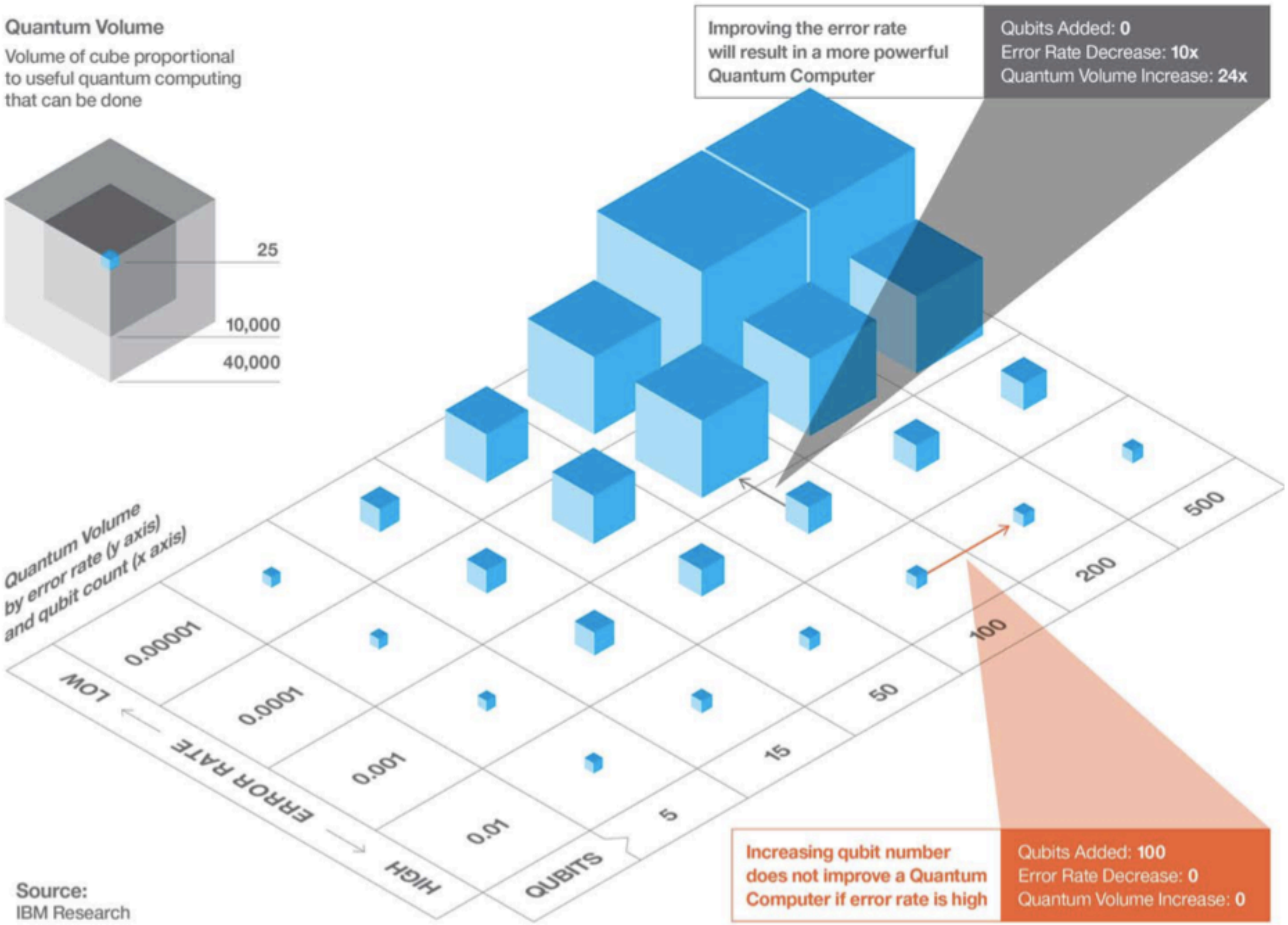
Quantum Volume

Number of **qubits** (more is better)

Errors (fewer is better)

Connectivity (more is better)

Gate set (more is better)

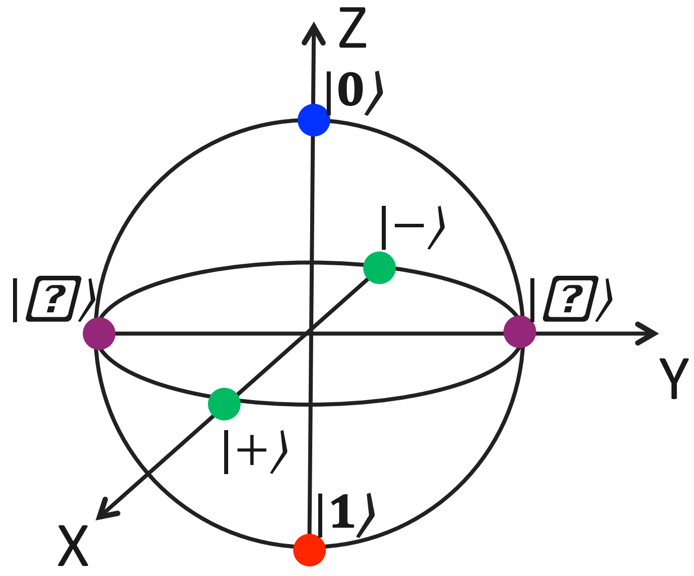


Quantum computing: quantum operations and circuits

Single-qubit gates

- **Gates are described by one or more rotations about an axis or set of axes**
  - **Pauli X, Y, Z gates:**
    - Rotate π radians about specified axis
    - X and Y gates equivalent to classical NOT
      - -Transform **|0>** to **|1>** and vice versa
  - **Clifford gates:**
    - Permute states identified at right (includes Pauli gates)
  - **Arbitrary gates:**
    - Map any point on sphere to any other
    - Typically implemented with a small set of well-calibrated gates, e.g. ***Clifford group*** plus one additional gate

***Clifford group*: permutes the states $0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$, $|\boxed{?}\rangle$, and $|\boxed{?}\rangle$, identified below**

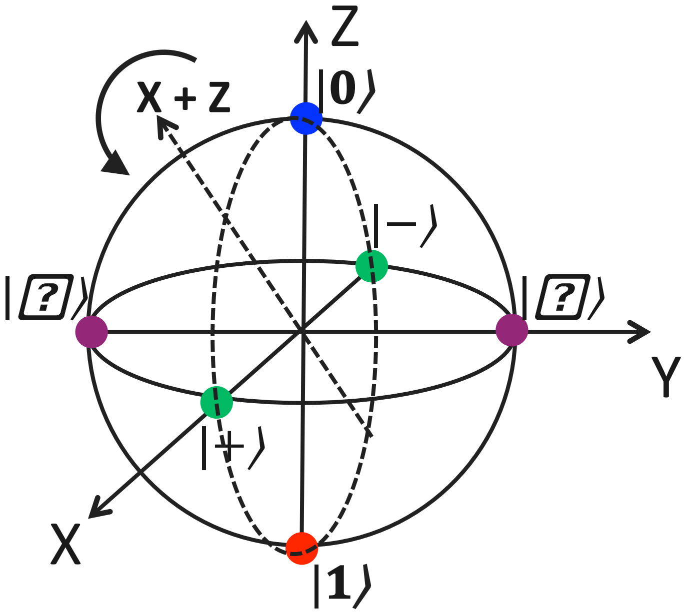


$|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$

$|\boxed{?}\rangle = (|0\rangle + i|1\rangle)/\sqrt{2}$ $|\boxed{?}\rangle = (|0\rangle - i|1\rangle)/\sqrt{2}$

Key single-qubit gate: Hadamard (*H*)

- **Hadamard** gate: rotate 180°about X+Z axis
  - Exchanges Z and X axes
  - Takes classical states to equal-weighted superposition states and vice versa
    - $|0\rangle \rightarrow |+\rangle$ $\quad |+\rangle \rightarrow |0\rangle$
    - $|1\rangle \rightarrow |-\rangle$ $\quad |-\rangle \rightarrow |1\rangle$
  - Used in almost every quantum algorithm



- Performs the ***quantum Fourier transform*** of a single qubit
  - Classical Fourier transform: exchange conjugate variables describing a *signal* (e.g. time domain → frequency domain)
  - Quantum Fourier transform: exchange conjugate variables describing a *state*

Matrix representation of Hadamard acting on

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle$$

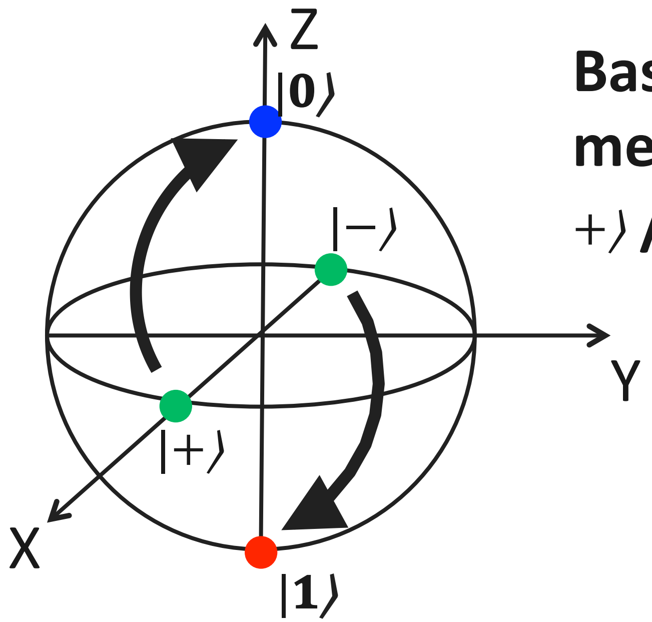
Qubit measurements

- **Standard measurement in the computational basis:**
  - Collapses any superposition into one of the two classical states: $|\mathbf{0}\rangle$ or $|\mathbf{1}\rangle$
- **Measurement in other bases:**
  - Measurement itself is only sensitive to $|\mathbf{0}\rangle$ vs $|\mathbf{1}\rangle$
  - To measure in other bases, rotate first
  - Example: to distinguish $|+\rangle$ from $|-\rangle$, apply Hadamard before measuring
    - If state was $|+\rangle$, measure $|\mathbf{0}\rangle$
    - If state was $|-\rangle$, measure $|\mathbf{1}\rangle$

Measurement icon used in the IBM QX

<u>Initial state</u> <u>Possible outcomes</u>

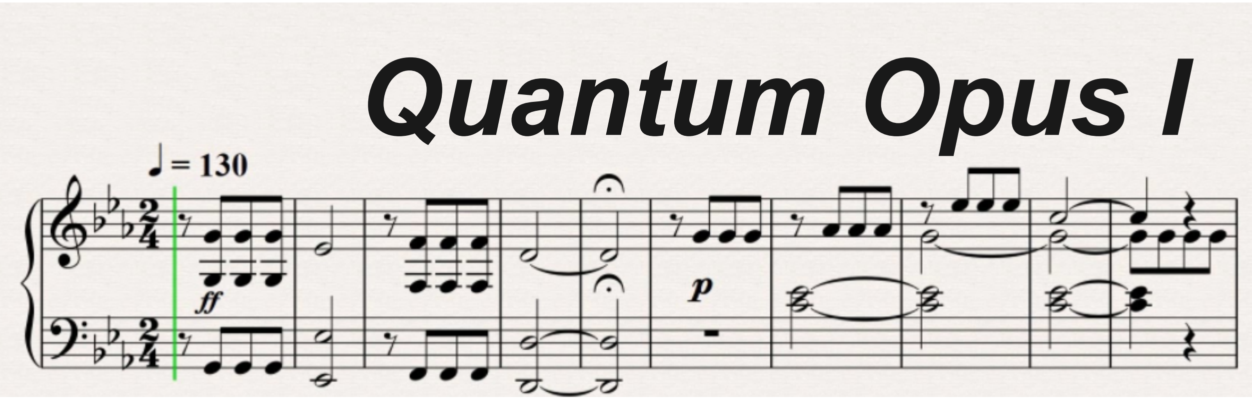
$$\alpha|0\rangle + \beta|1\rangle \Rightarrow \begin{cases} |0\rangle \text{ with probability } \alpha^2 \\ |1\rangle \text{ with probability } \beta^2 \end{cases}$$

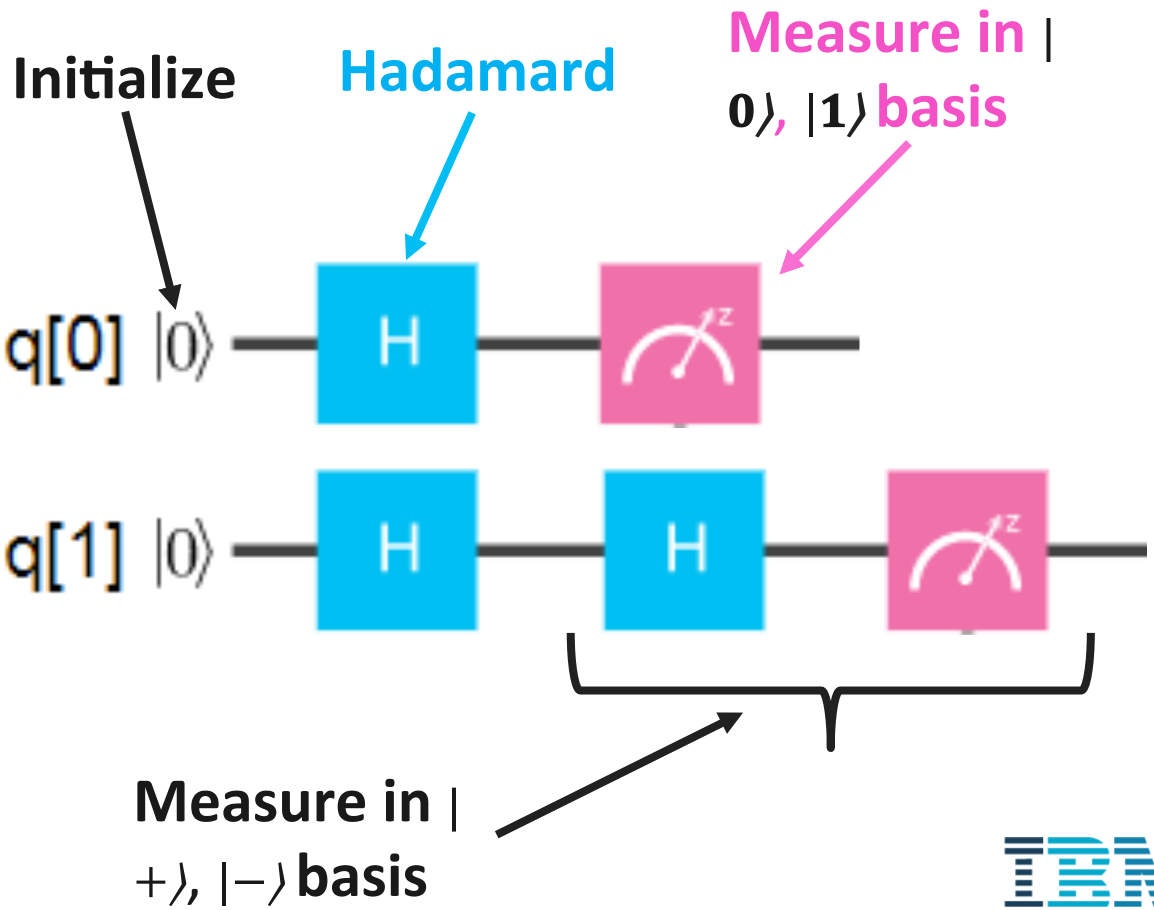


Basis change for measuring in $|+\rangle$ / $|-\rangle$ basis

A simple "quantum score"

- Visual representation of a series of operations performed on a ***quantum register*** (a set of qubits grouped together)
- N-qubit quantum register: qubits q[0] – q[N-1]
- After measurement, results stored in *classical register* as c[0] – c[N-1]
- Example quantum score on 2-qubit register:
  - Initialize both qubits in $|\mathbf{0}\rangle$
  - Apply Hadamard (*H*) to each qubit
  - Measure q[0] in the $|\mathbf{0}\rangle$, $|\mathbf{1}\rangle$ basis
  - Measure q[1] in the $|+\rangle$, $|-\rangle$ basis
- Results:
  - q[0] measurement gives either $|\mathbf{0}\rangle$ or $|\mathbf{1}\rangle$, each with 50% probability
  - q[1] measurement always gives $|\mathbf{0}\rangle$
    - Infer that q[1] was in $|+\rangle$ prior to 2nd *H*



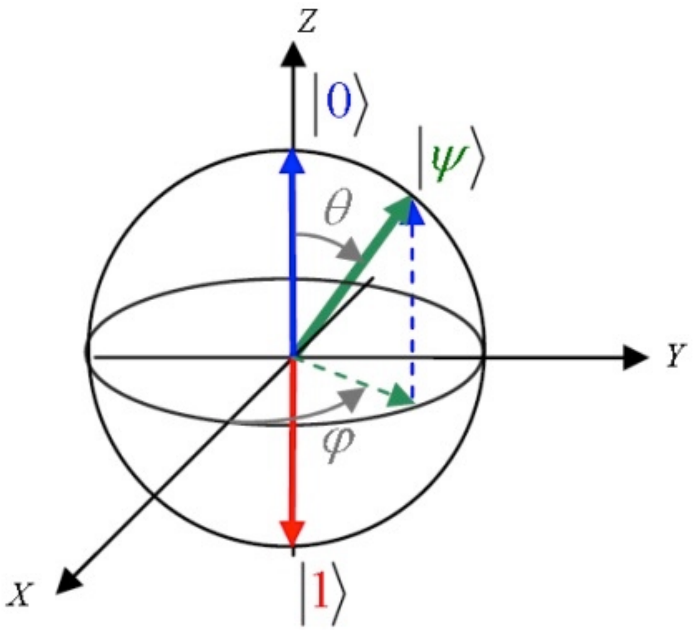


Multi-qubit operations

- Two-qubit operations:
  - Controlled not (CNOT):
    - Classical behavior: flip target *iff* control is 1

Initial State		Final State	
Control Q	Target Q	Control Q	Target Q
|	|	|	|
|	|	|	|
α| + β|	|	α| + β|	

Entangled state!

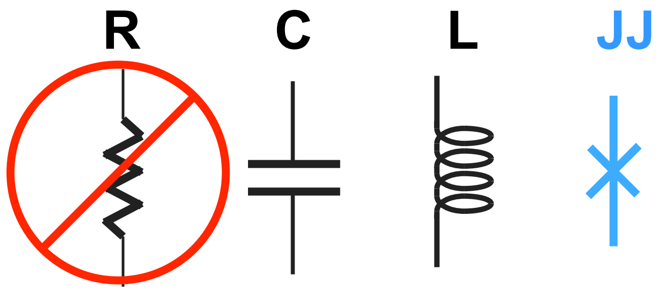


  - Controlled phase (CPhase)
    - Same idea but target qubit is flipped around the Z axis (instead of X)
    - Equivalent to CNOT up to single-qubit gates

Superconducting qubits: device properties

Superconducting qubit building blocks

Circuit element toolbox

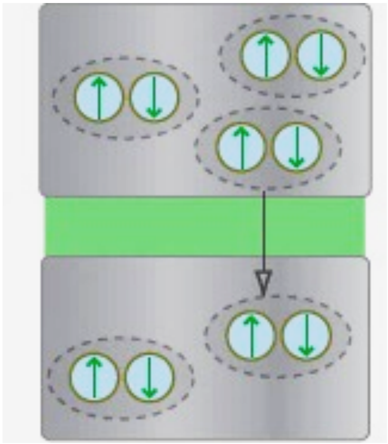


Josephson Junction:

- **Weak link between two superconductors**
- **Typically Al / AlOx / Al**

Key features:

- **non-linear inductance**
- **dissipationless operation**

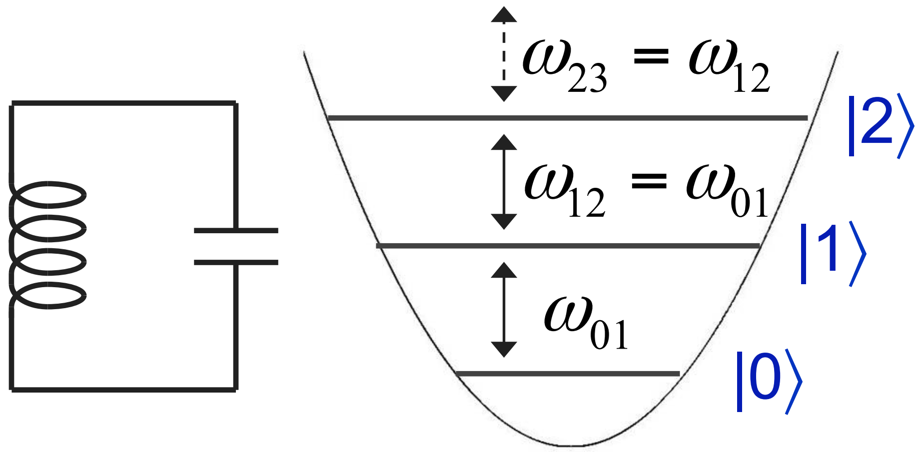


$$\frac{dI}{dt} = \frac{1}{L} V(t)$$

$$L(\delta) = \frac{\Phi_0}{2\pi I_0 \cos(\delta)}$$

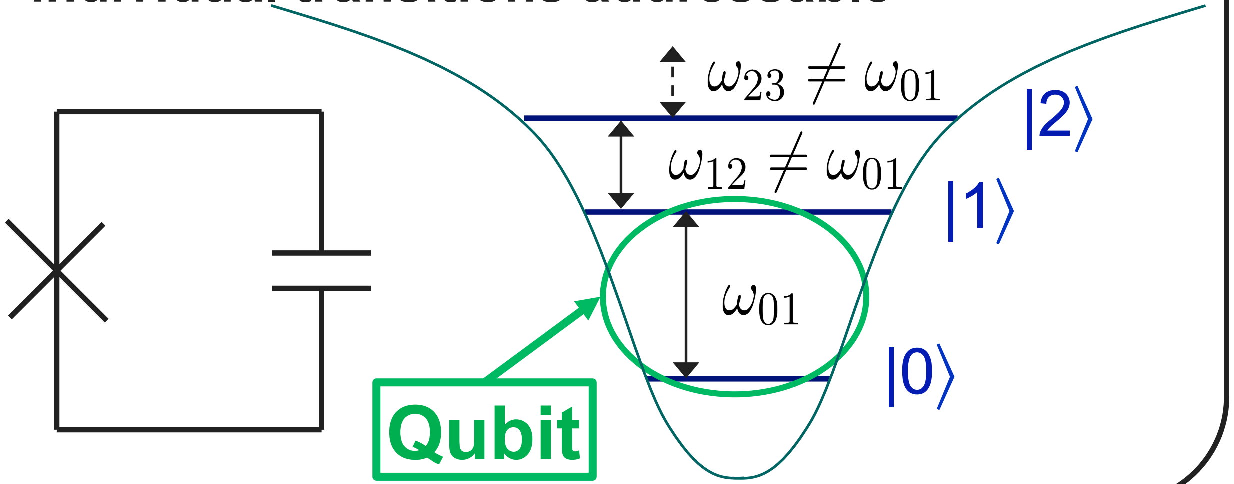
L-C Oscillator: ***harmonic***

→ can't address individual transitions



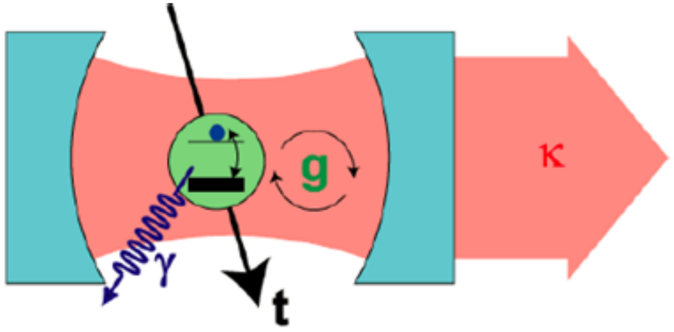
JJ-C Oscillator: ***anharmonic***

→ individual transitions addressable



Qubit coupling via resonators: circuit QED (cQED)

- **Qubit interacts with environment via a resonator**
- **Analogous to an atom in an optical cavity**



2g = vacuum Rabi freq.

κ = cavity decay rate

γ = “transverse” decay rate

Jaynes-Cummings Hamiltonian

$$\hat{H} = \hbar\omega_r(a^\dagger a + \tfrac{1}{2}) - \frac{\hbar\omega_0}{2}\hat{\sigma}_z - \hbar g(a^\dagger\sigma^- + \sigma^+ a) + H_\kappa + H_\gamma$$

Quantized Field

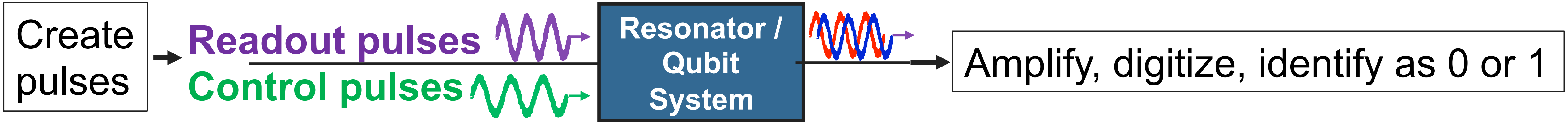
2-level system

Electric dipole Interaction

Dissipation

Wallraff et al., Nature 431, 162 (2004)

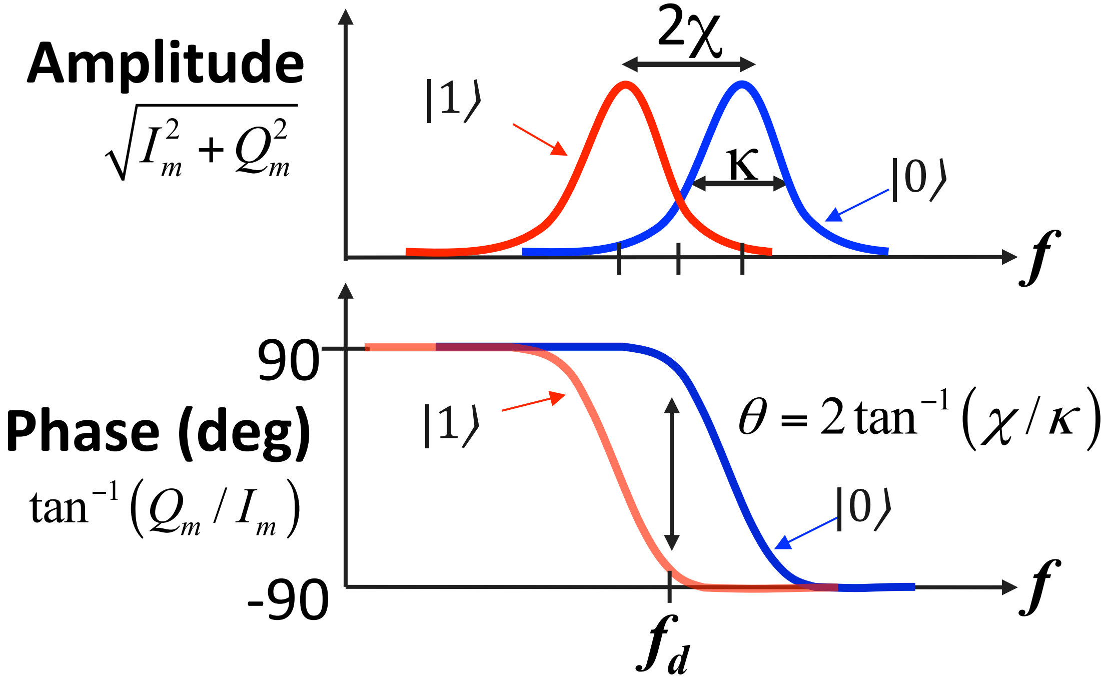
Qubit Readout in cQED

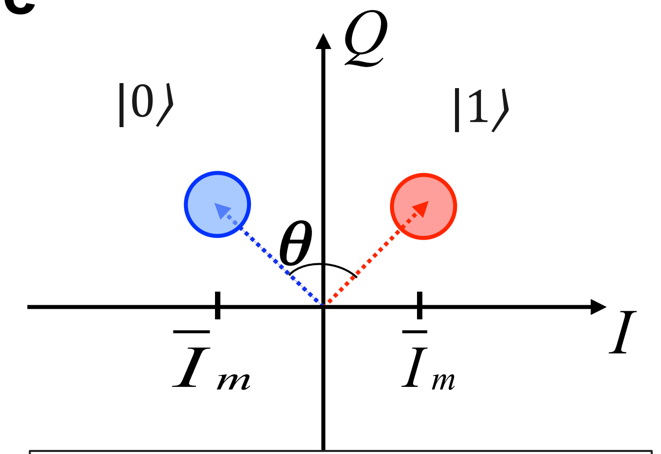


Readout freq. near ω_r; control freq. at ω_0

Resonator frequency depends on qubit state
→ Infer qubit state from resonator response

I = in-phase
Q = out-of-phase

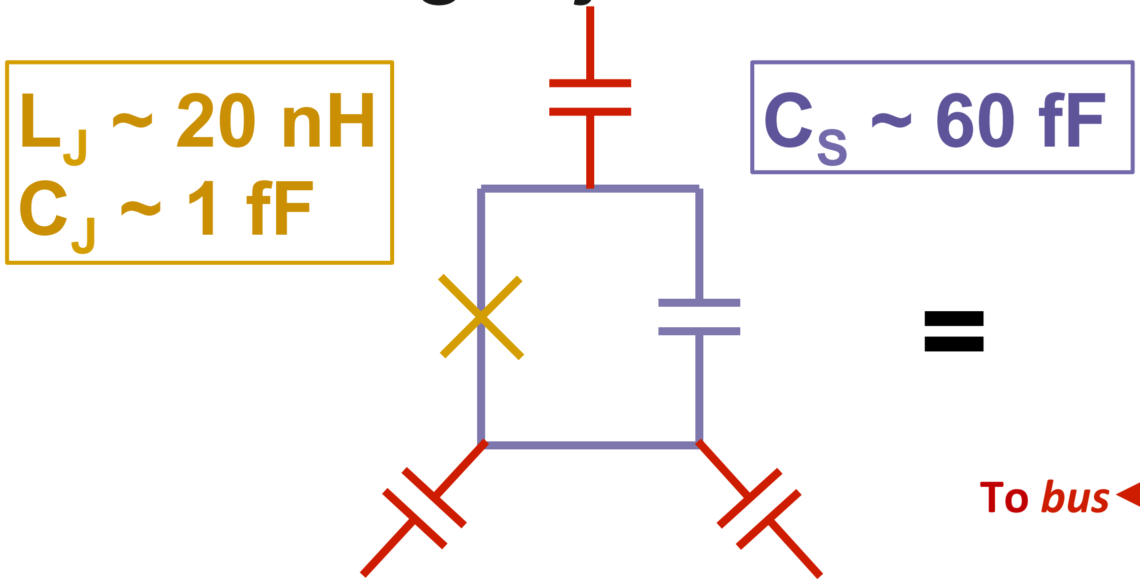


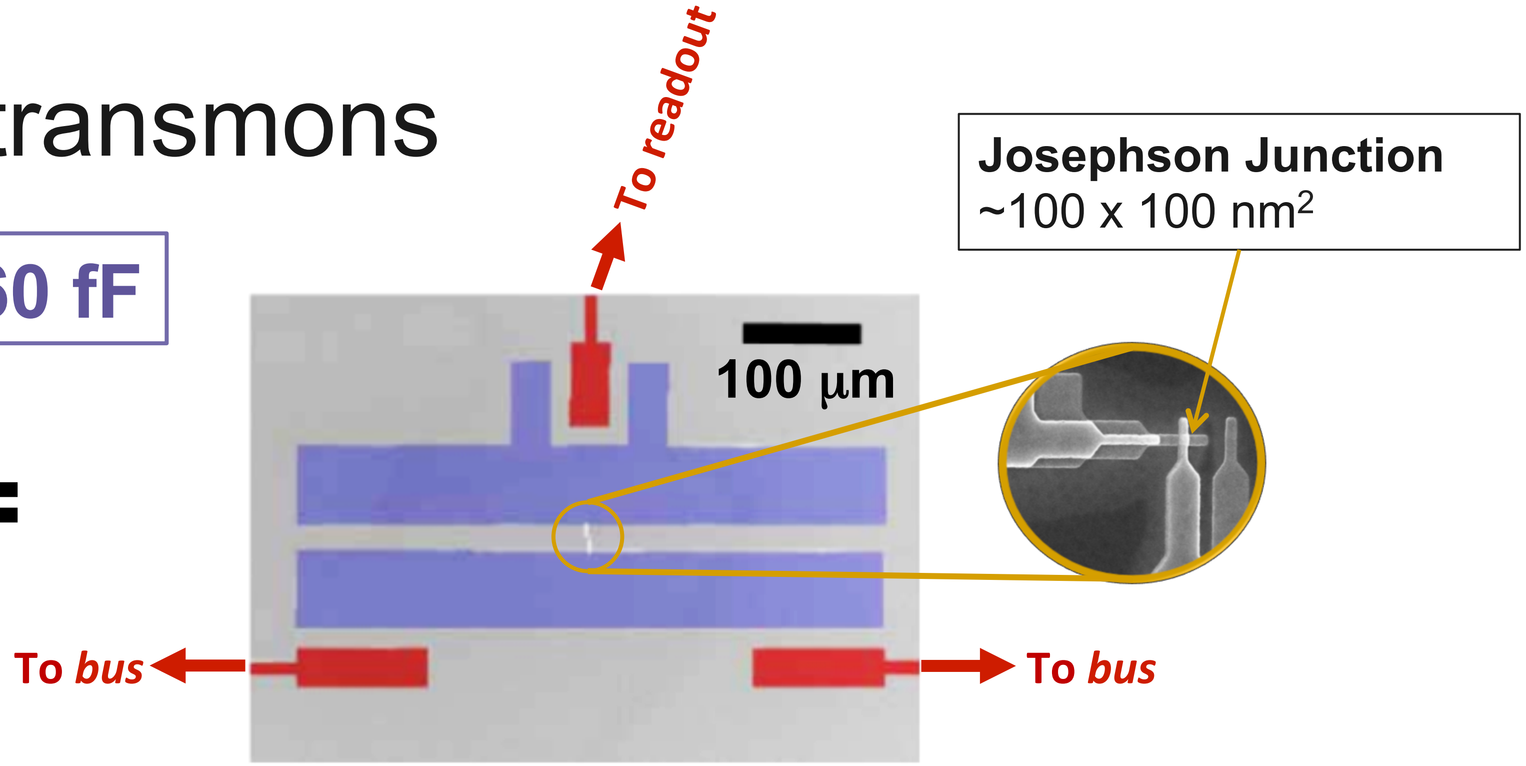


For $2\chi = \kappa$, $\theta = 90°$

Gambetta et al., PRA 77, 012112 (2008)
Jeffrey et al., PRL 112, 190504 (2014)
Magesan et al., PRL 114, 200501 (2015)

IBM single-junction transmons





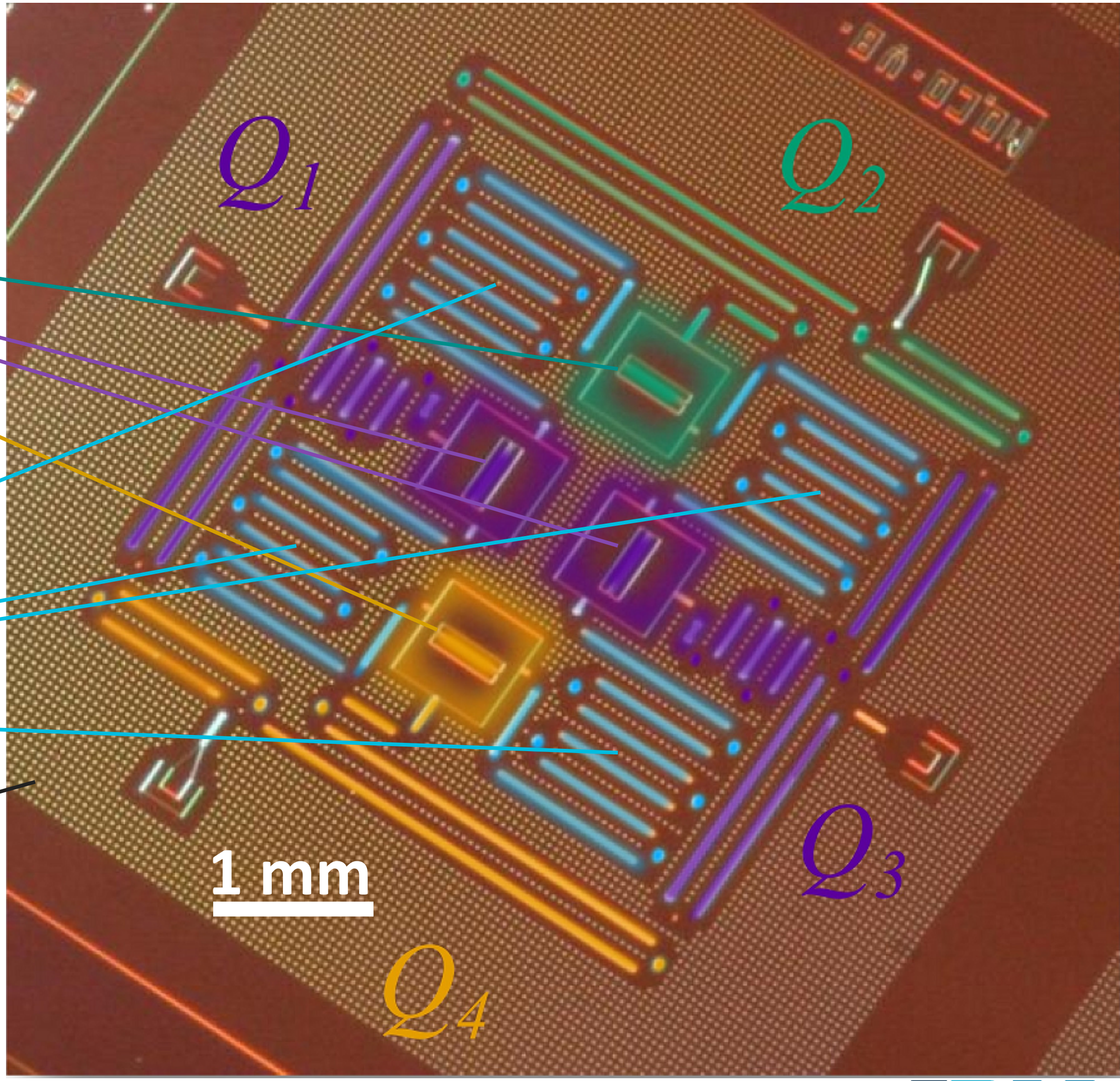
- Patterned superconducting metal (**niobium** + **aluminum**) on silicon
  – Qubit capacitance dominated by **shunting capacitance C_S**
- Resonant frequency ~ **5 GHz** → energy splitting ~ 20 μeV, or 240 mK
  → Cool in a dilution refrigerator (~ 10 mK) to reach ground state
- Interactions mediated by **capacitively coupled co-planar waveguide resonators** (circuit QED)

Anatomy of a multi-qubit device

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

Resonators:
Co-planar waveguide
Frequency ~ 6 – 7 GHz
Roles:
Individual qubit readout
Qubit coupling ("bus")

Ground plane
Periodic holes prevent stray magnetic field from hurting superconductor performance



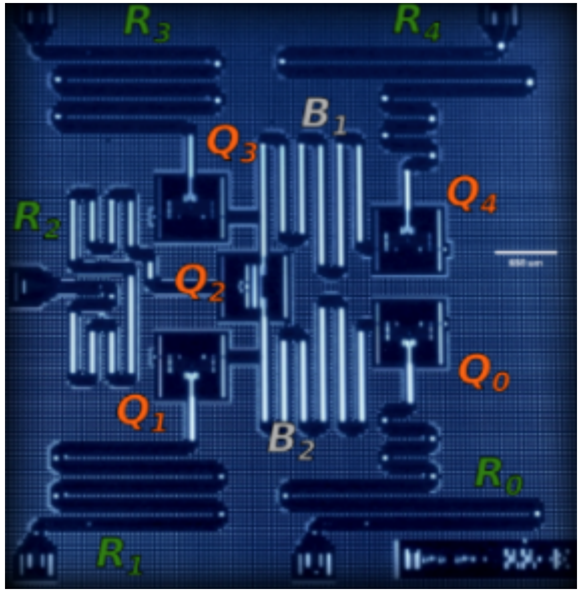
Corcoles et al., Nat. Commun. 6, 6979 (2015)

IBM Quantum Experience

IBM Quantum Experience (IBMQX)

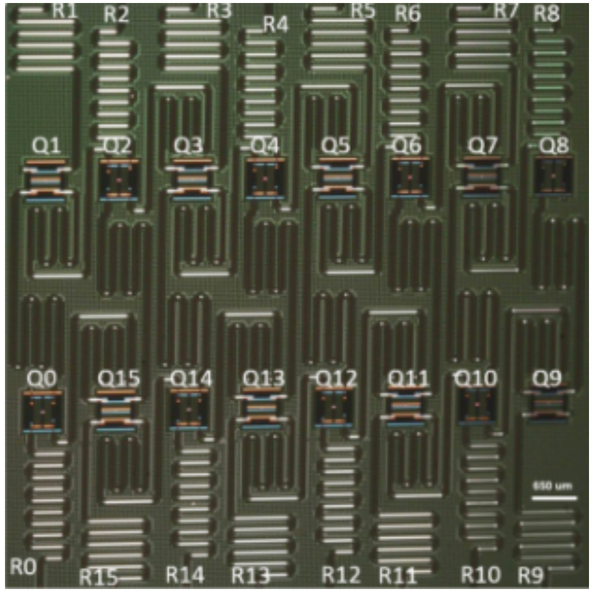
- **Free cloud based quantum computing platform**
  - 5-qubit quantum processor (real hardware)
  - 20-qubit quantum simulator
  - 16-qubit quantum processor (access through QISKit: www.qiskit.org)

Cloud
research.ibm.com/ibm-q/



IBM QX2: 5-qubit

Quantum Simulator



IBM QX3: 16-qubit

A quantum teleportation experiment for undergraduate students

S. Fedortchenko*
Laboratoire Matériaux et Phénomènes Quantiques, Sorbonne Paris Cité, ...ot, CNRS UMR 7162, 75013, Paris, France

Braiding Majoranas in a five qubit experiment

James R. Wootton
...nt of Physics, University of Basel, Klingelbergstrasse 82, CH-4056 Basel, Switzerland

Performing Quantum Computing Experiments in the Cloud

Simon J. Devitt
Center for Emergent Matter Science, RIKEN, Wakoshi, Saitama 315-0198, Japan.

PHYSICAL REVIEW A 94, 012314 (2016)

Experimental test of Mermin inequalities on a five-qubit quantum computer

Daniel Alsina and José Ignacio Latorre
Departament Física Quàntica i Astrofísica, Universitat de Barcelona, Diagonal 645, 080...
and Institut de Ciències del Cosmos (ICCUB), Martí i Franquès 1, 08028 Barc...

(Received 25 May 2016; published 11 July 2016)

Homomorphic Encryption Experiments on IBM's Cloud Quantum Computing Platform

He-Liang Huang,[1,2] You-Wei Zhao,[2,3] Tan Li,[1,2] Feng-Guang Li,[1,2] Yu-Tao Du,[1,2] Xiang-Qun Fu,[1,2] Shuo Zhang,[1,2] Xiang Wang,[1,2] and Wan-Su Bao[1,2,*]

New Journal of Physics
The open access journal at the forefront of physics
IOP Institute of Physics | DPG
Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

PAPER

Entropic uncertainty and measurement reversibility

Mario Berta[1], Stephanie Wehner[2] and Mark M Wilde[3,4]

Experimental Comparison of Two Quantum Computing Architectures

N. M. Linke,[1] D. Maslov,[2,3] M. Roetteler,[4] S. Debnath,[1] C. Figgatt,[1] K. A. Landsman,[1] K. Wright,[1] and C. Monroe[1,3,5]
[1] *Joint Quantum Institute and Department of Physics,*

Demonstration of *entanglement assisted invariance* on IBM's Quantum Experience

Sebastian Deffner
Department of Physics, University of Maryland Baltimore County, Baltimore, MD 21250, USA

Compressed quantum computation using the IBM Quantum Experience

M. Hebenstreit,[1] D. Alsina,[2,3] J. I. Latorre,[2,3] and B. Kraus[1]
[1] *Institute for Theoretical Physics, University of Innsbruck,*
[2] *Dept. Física Quàntica i Astrofísica, Universitat de Barcelona, Diagon...*
[3] *Institut de Ciències del Cosmos, Universitat de Barcelona, Diagonal...*

Leggett-Garg test of superconducting qubit addressing the clumsiness loophole

Emilie Huffman[1,2] and Ari Mizel[1]
...oratory for Physical Sciences, College Park, Maryland 20740, USA
...ment of Physics, Duke University, Durham, North Carolina 27708, USA

O Computador Quântico da IBM e o *IBM Quantum Experience*

IBM Quantum Computer and the IBM Quantum Expe...

Alan C. Santos*[1]
...de Física, Universidade Federal Fluminense, Niterói, Rio de Ja...

Quantum state reconstruction made easy: a direct method for tomography

R. P. Rundle,[1] Todd Tilma,[2] J. H. Samson,[1] and M. J. Everitt[1,*]
[1] *Quantum Systems Engineering Research Group & Department of Physics, Loughborough University, Leicestershire LE11 3TU, United Kingdom*
[2] *Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8550, Japan*

ProjectQ: An Open Source Software Framework for Quantum Computing

Damian S. Steiger,* Thomas Häner,* and Matthias Troyer*
Institute for Theoretical Physics, ETH Zurich, 8093 Zurich, Switzerland

(Dated: December 28, 2016)

Quintuple: a Python 5-qubit quantum computer simulator to facilitate cloud quantum computing

Christine Corbett Moran[a,b,*]
[a] *NSF AAPF California Institute of Technology, TAPIR, 1207 E. California Blvd. Pasadena, CA 91125*
[b] *University of Chicago, 2016 SPT Winterover Scientist, Amundsen-Scott South Pole Station, Antarctica*

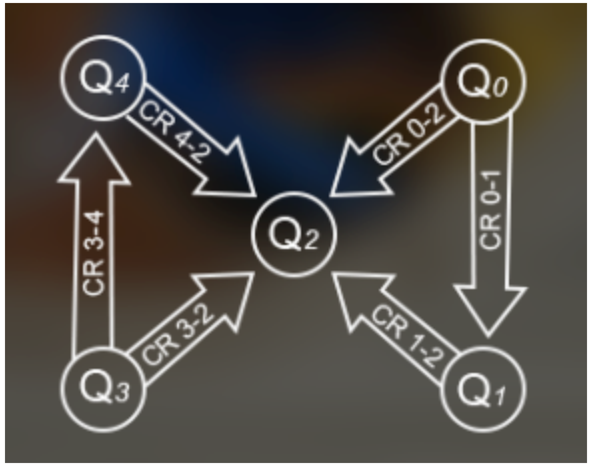
Approximate Quantum Adders with Genetic Algorithms: An IBM Quantum Experience

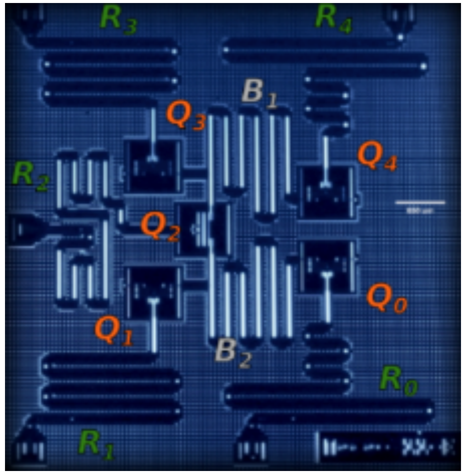
Rui Li[1], Unai Alvarez-Rodriguez[2], Lucas Lamata[2], and Enrique Solano[2,3]
[1] Department of Physics, Zhejiang University, Hangzhou 310027, China
[2] Department of Physical Chemistry, University of the Basque Country UPV/EHU, Apartado 644, 48080 Bilbao, Spain
[3] IKERBASQUE, Basque Foundation for Science, Maria Diaz de Haro 3, 48013 Bilbao, Spain

ABSTRACT

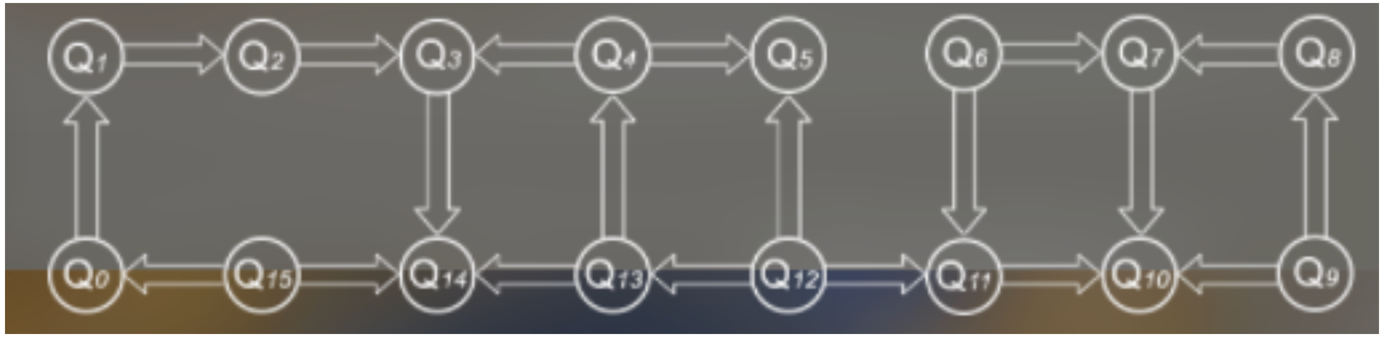
Real Quantum Processor: Device Details

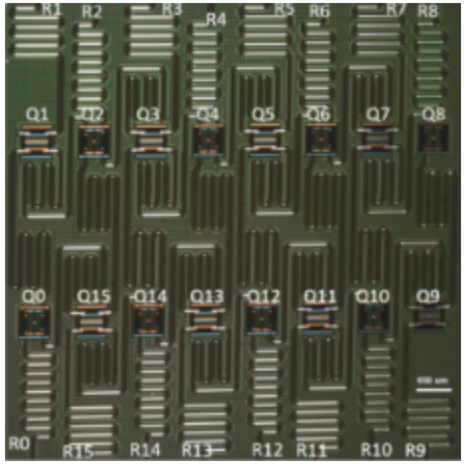
- **5-qubit device**
  - Single-junction transmons
  - $T_1 \sim T_2 \sim 50 - 100\ \mu s$
  - 1Q gate fidelities > 99%
  - 2Q gate fidelities > 95%
  - Measurement fidelities > 93%
  - Connectivity: 6 CNOTs available





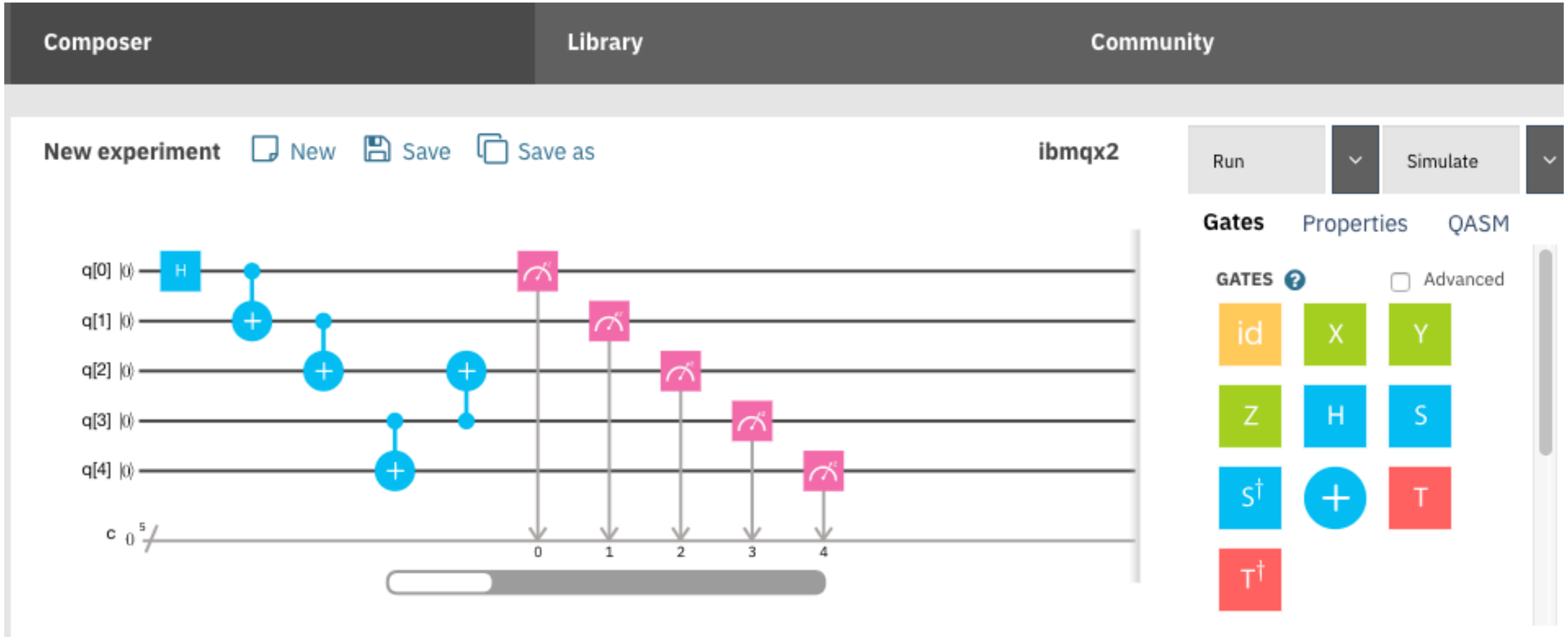
- **16-qubit device (NEW!)**
  - Access through QISKit API only





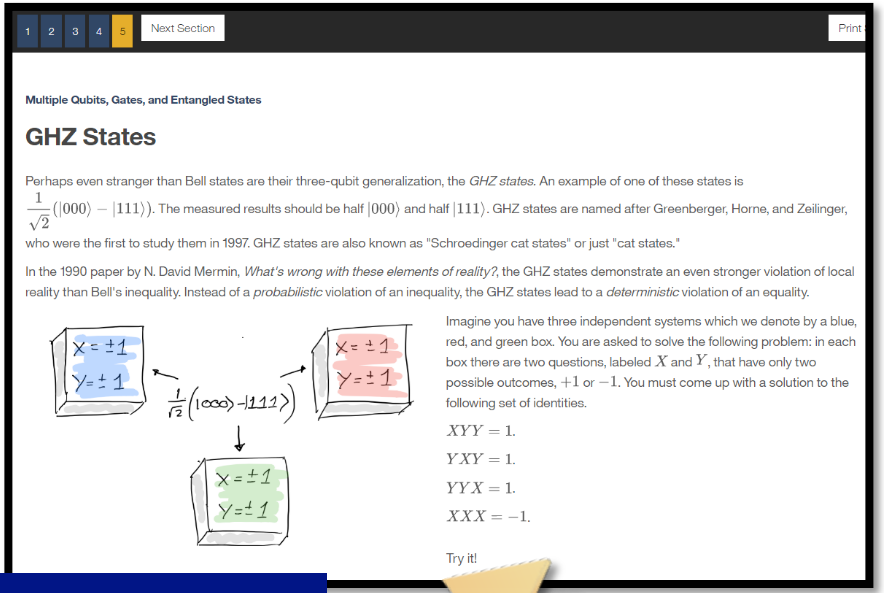
IBM QX: Web Interface

- https://quantumexperience.ng.bluemix.net
- **Graphical composer**
  - Compose quantum circuits using drag and drop interface
  - Save circuits online or as QASM text, and import later
  - Run circuits on real hardware and simulator



IBM QX: Web Interface

- https://quantumexperience.ng.bluemix.net
- **Library**
  – User guides for all levels (beginner, advanced, developer)
  – Run examples in composer



Multiple Qubits, Gates, and Entangled States

GHZ States

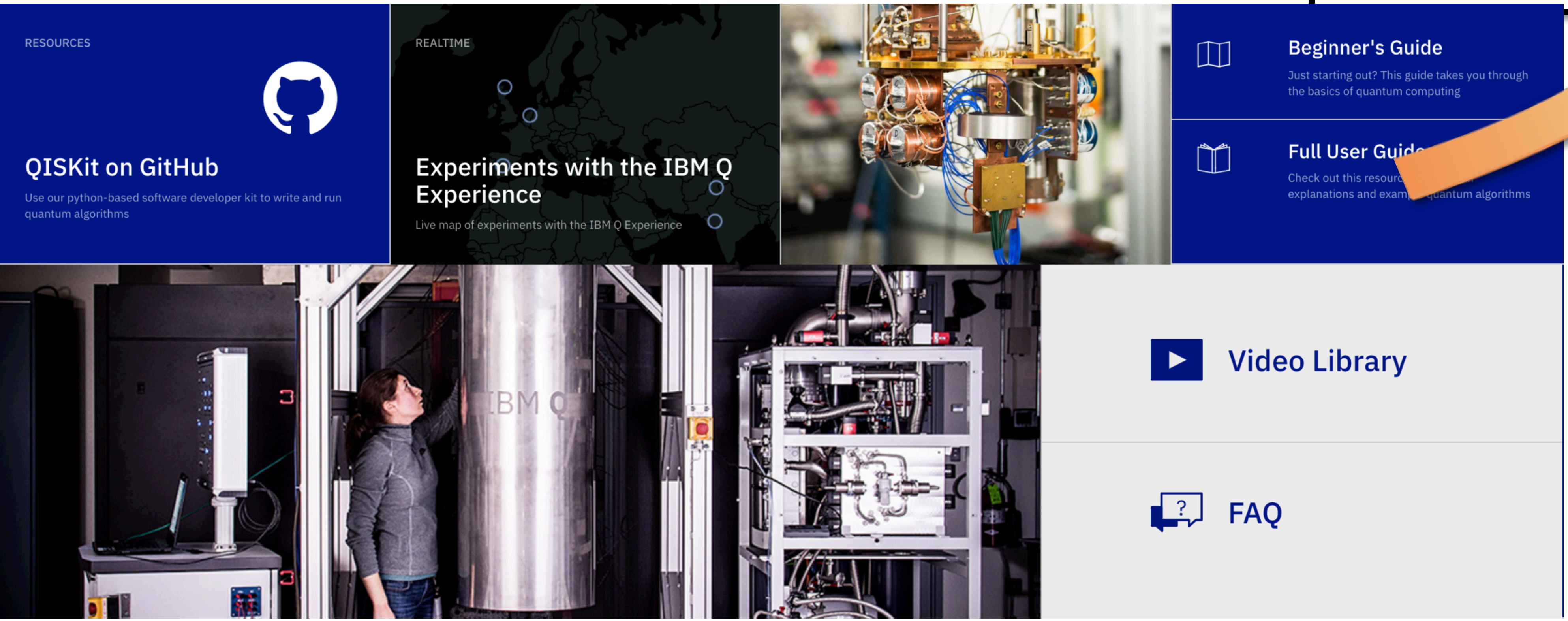
Perhaps even stranger than Bell states are their three-qubit generalization, the *GHZ states*. An example of one of these states is $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. The measured results should be half $|000\rangle$ and half $|111\rangle$. GHZ states are named after Greenberger, Horne, and Zeilinger, who were the first to study them in 1997. GHZ states are also known as "Schroedinger cat states" or just "cat states."

In the 1990 paper by N. David Mermin, *What's wrong with these elements of reality?*, the GHZ states demonstrate an even stronger violation of local reality than Bell's inequality. Instead of a *probabilistic* violation of an inequality, the GHZ states lead to a *deterministic* violation of an equality.

Imagine you have three independent systems which we denote by a blue, red, and green box. You are asked to solve the following problem: in each box there are two questions, labeled X and Y, that have only two possible outcomes, $+1$ or -1. You must come up with a solution to the following set of identities.

$$XYY = 1.$$
$$YXY = 1.$$
$$YYX = 1.$$
$$XXX = -1.$$

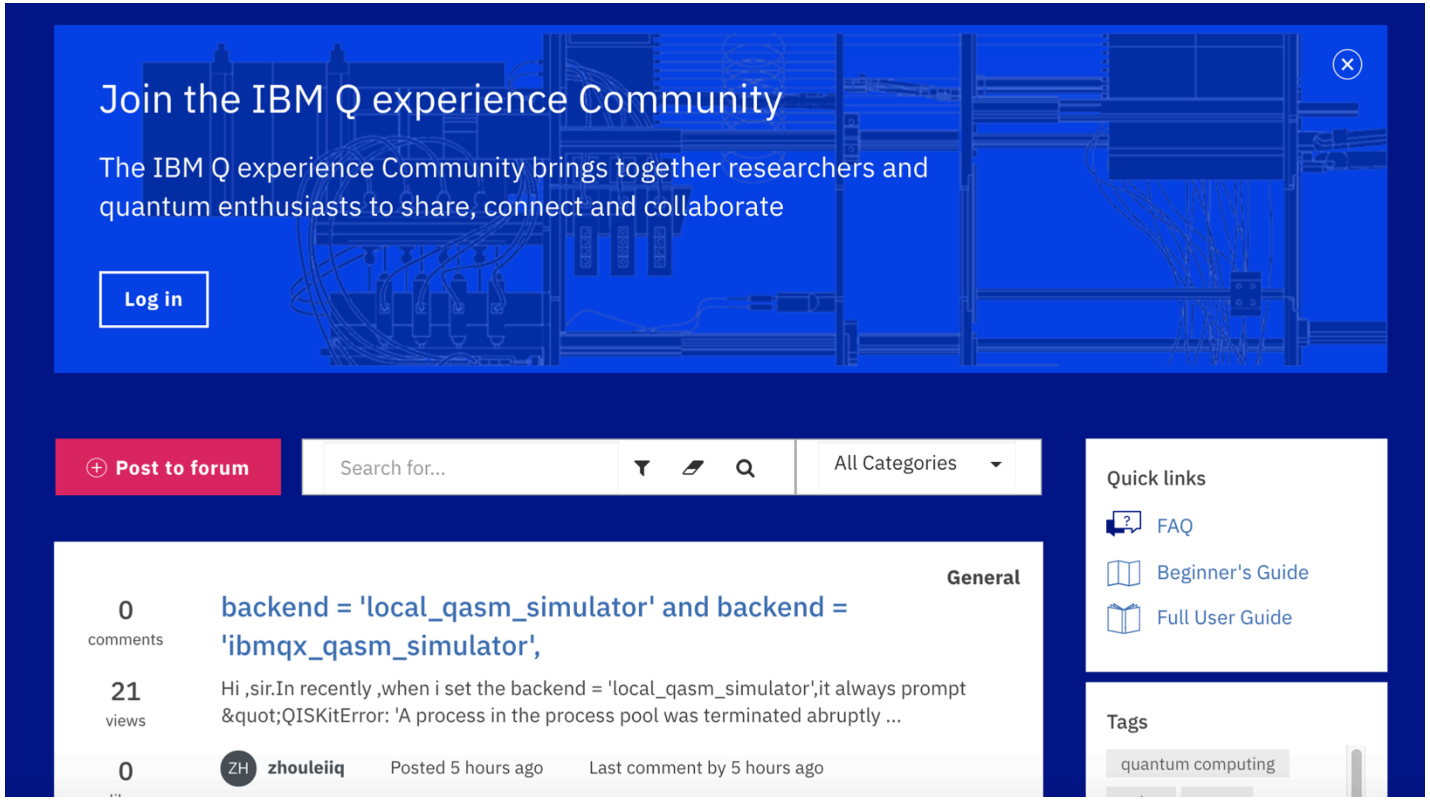
Try it!



IBM QX: Web Interface

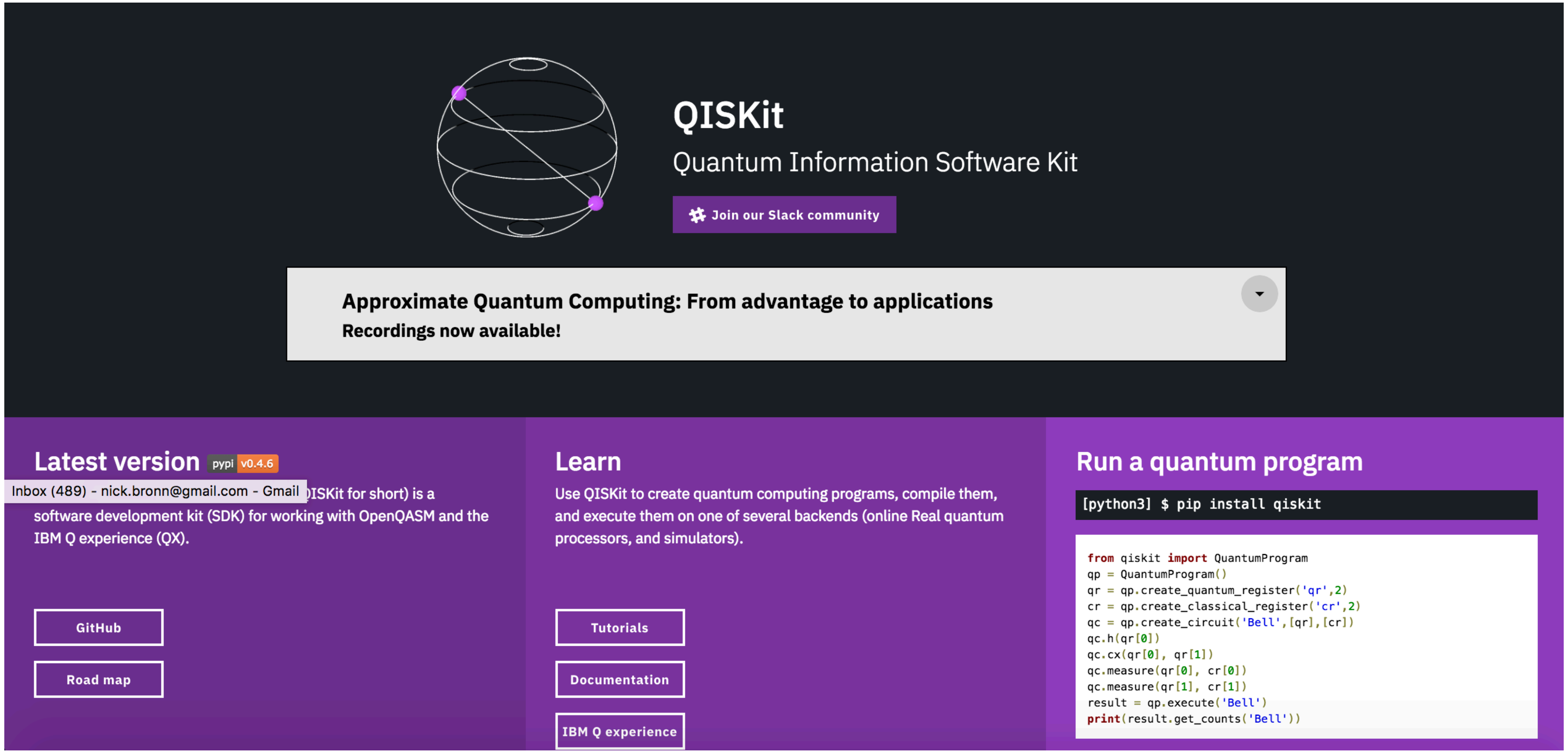
- https://quantumexperience.ng.bluemix.net

- **Community forum**
  - Ask questions, discuss ideas
  - Receive answers from IBM staff and community members
  - Keep up to date with announcements and news



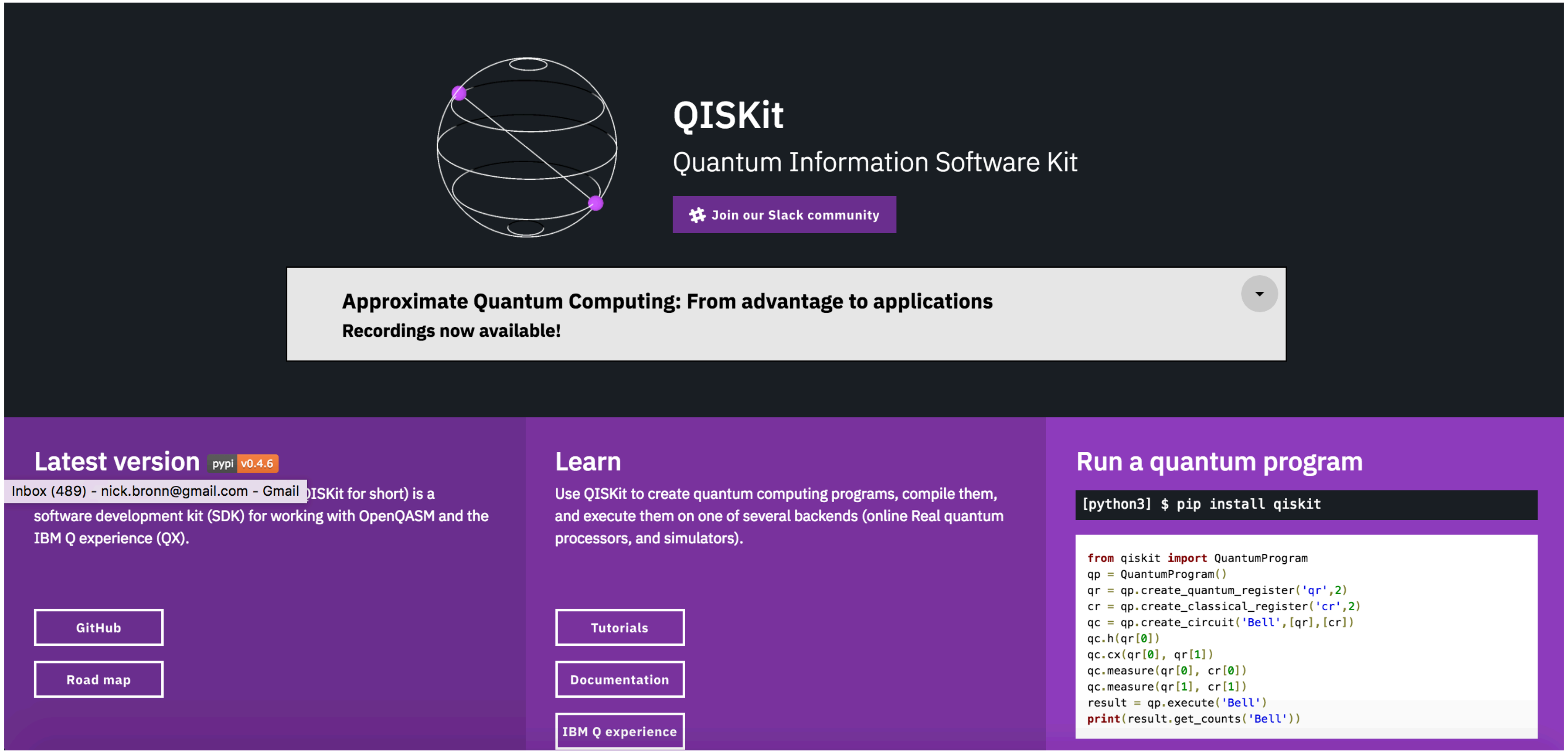
IBM QX: QISKit Interface

- www.qiskit.org
- Open source project for quantum software development tools



IBM QX: QISKit Interface

- www.qiskit.org
- **GitHub: Python SDK**
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs



IBM QX: QISKit Interface

- www.qiskit.org
- **GitHub: Python SDK**
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs
- **GitHub: Tutorial Notebooks**
  - Interactive Jupyter notebooks demonstrating a variety of topics

jupyter index (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Markdown

1. Introducing the tools

In this first topic, we break down the tools in the QISKit SDK, and introduce all the different parts to make this useful. Our list of introductory notebooks:

- Getting started with QISKit SDK - how to use QISKit SDK.
- Understanding the different backends - how to get information about the connected backends.
- Compiling and running a quantum program - how to rewrite circuits to different backends.
- Running a quantum program on IBM DSX - how to run a quantum notebook directly using IBM Data Science Experience (i.e. without installing any dependencies locally!)
- Loading and Saving a Quantum Program [coming soon].
- Visualizing a quantum state - illustrates the different tools we have for visualizing a quantum state.
- Quantum gates and linear algebra - list all basic gates and their definitions

2. Exploring quantum information concepts

The next set of notebooks shows how you can explore some simple concepts of quantum information science.

IBM QX: QISKit Interface

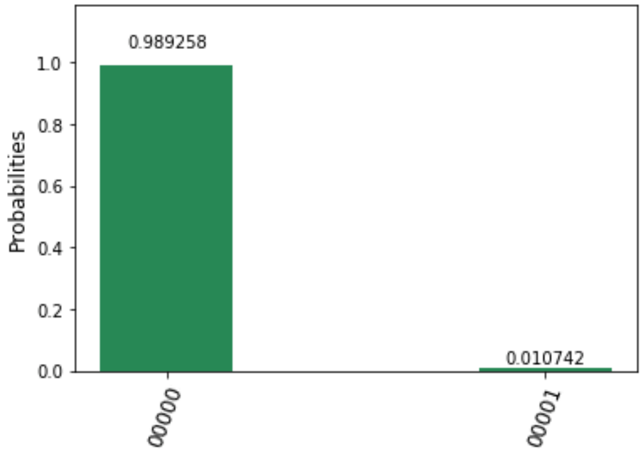
- www.qiskit.org
- **GitHub: Python SDK**
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs
- **GitHub: Tutorial Notebooks**
  - Interactive Jupyter notebooks demonstrating a variety of topics

In [5]:
```
result = Q_program.execute(circuits, backend=backend, shots=shots, max_credits=3, wait=10,
timeout=240, silent=False)
```

```
running on backend: ibmqx2
status = RUNNING (10 seconds)
status = RUNNING (20 seconds)
```

After the run has been completed, the data can be extracted from the API output and plotted.

In [6]:
```
plot_histogram(result.get_counts('ground'))
```



IBM QX: QISKit Interface

- www.qiskit.org
- **GitHub: Python SDK**
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs
- **GitHub: Tutorial Notebooks**
  - Interactive Jupyter notebooks demonstrating a variety of topics
- **Advanced documentation**

Table Of Contents

Getting Started

The starting point for writing code is the QuantumProgram object. The QuantumProgram is a collection of circuits, or scores if you are coming from the Quantum Experience, quantum register objects, and classical register objects. The QuantumProgram methods can send these circuits to quantum hardware or simulator backends and collect the results for further analysis.

To compose and run a circuit on a simulator, which is distributed with this project, one can do,

```
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'))
```

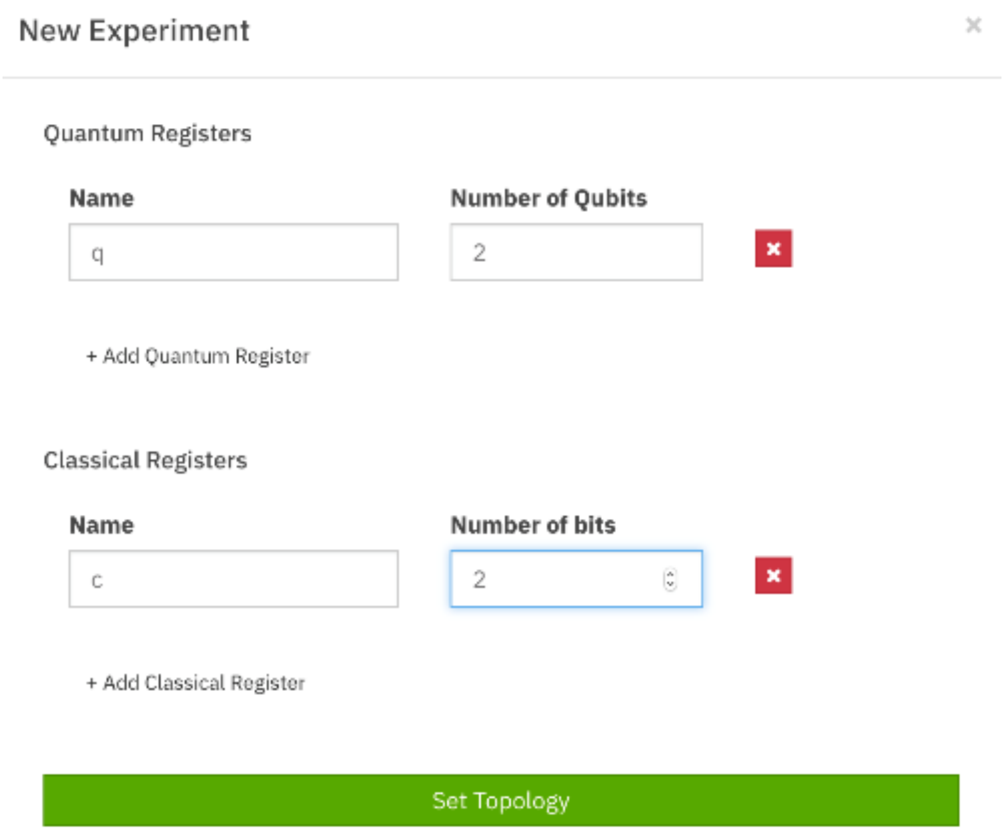
The `get_counts` method outputs a dictionary of state:counts pairs;

```
{'00': 531, '11': 493}
```


Using the Web Interface

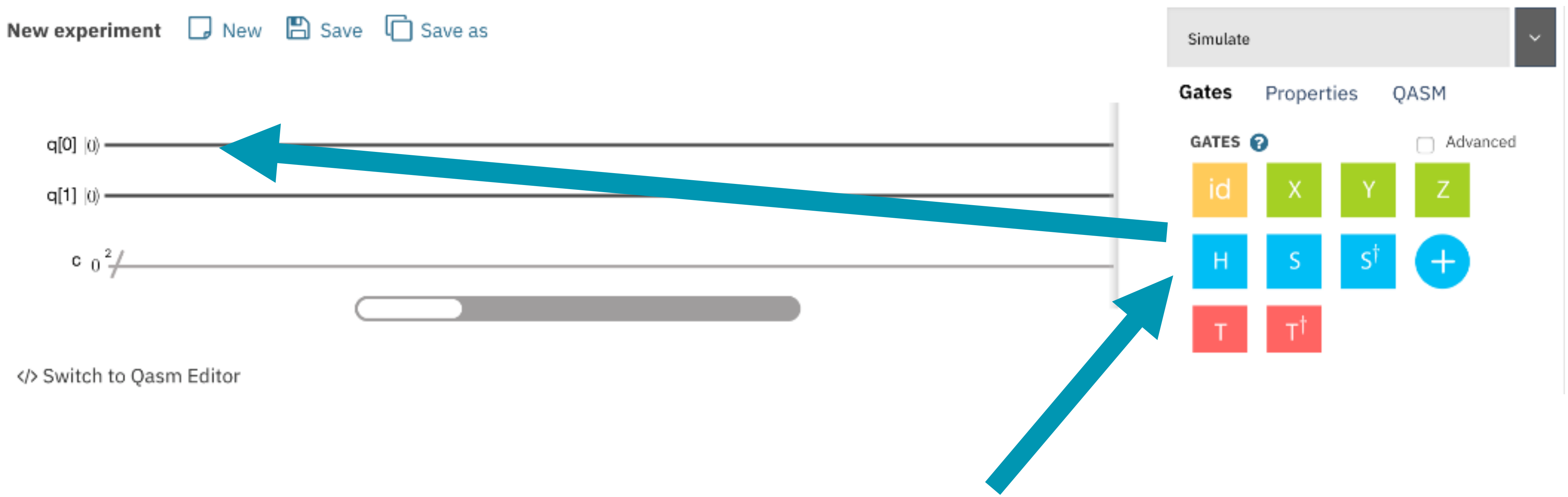
IBMQX: Getting started

- Create account at https://quantumexperience.ng.bluemix.net
- Create a new experiment: our example is 2-qubits



IBMQX: Getting started

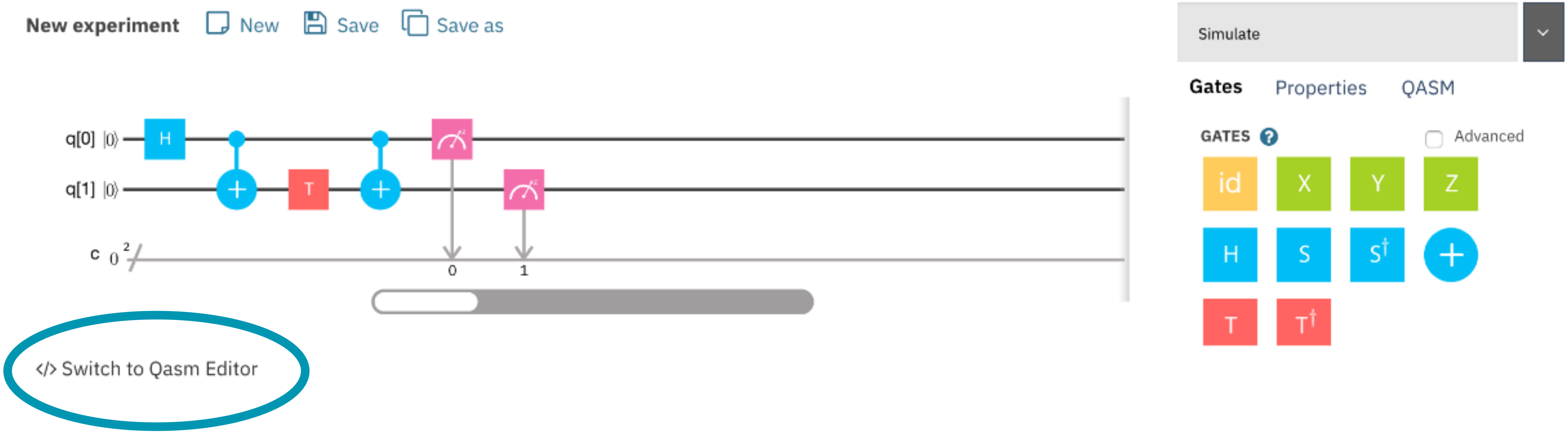
- Create account at https://quantumexperience.ng.bluemix.net
- Create a new experiment: our example is 2-qubits



Drag and drop gates onto the score

IBMQX: Getting started

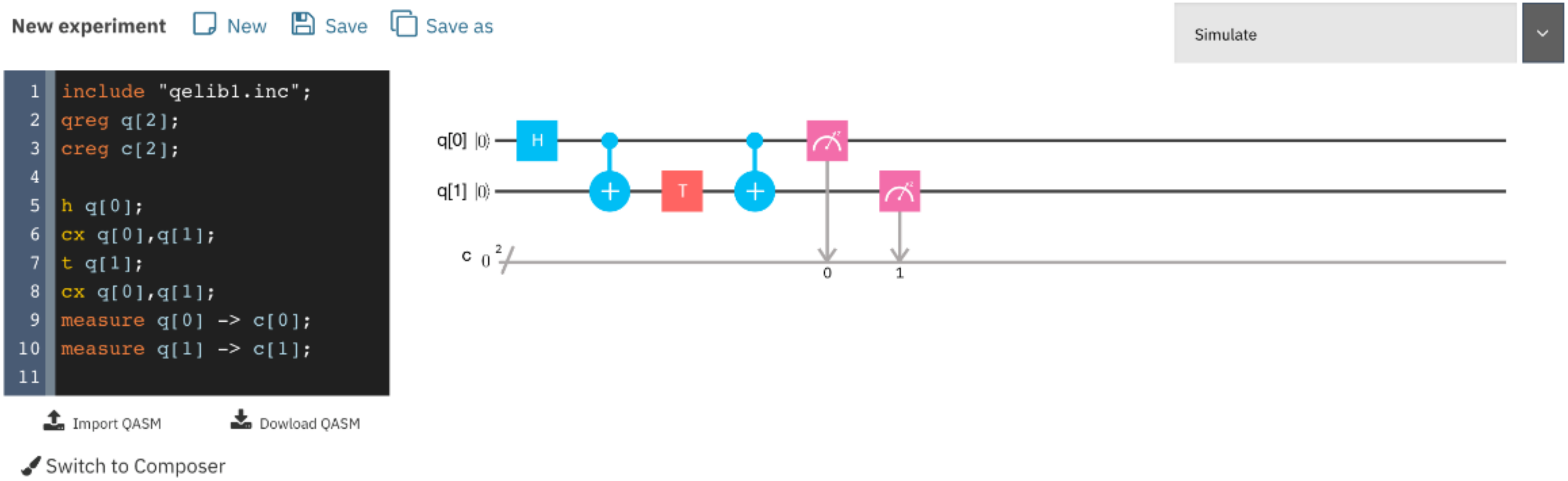
- Create account at https://quantumexperience.ng.bluemix.net
- Create a new experiment: our example is 2-qubits



- Score is translated into OPENQASM (a Quantum Assembly Language) behind the scene

IBMQX: Getting started

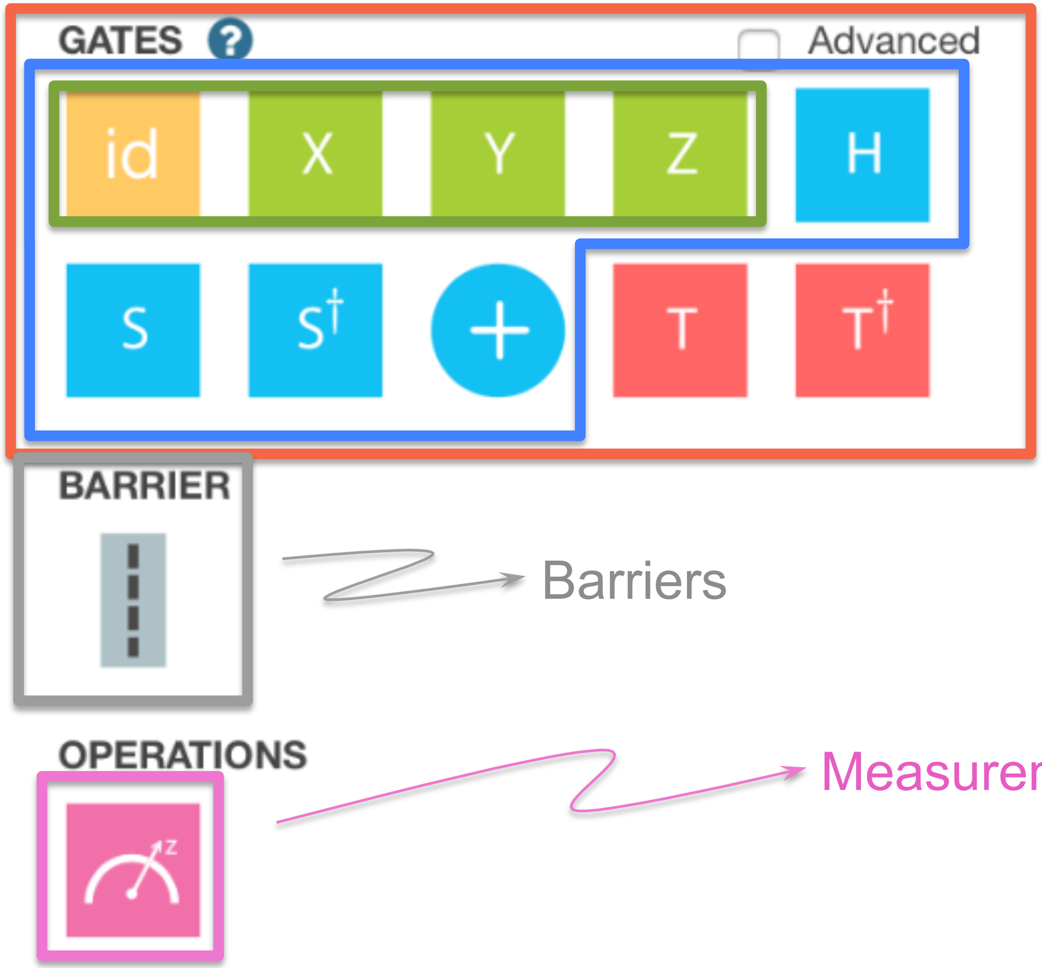
- Create account at https://quantumexperience.ng.bluemix.net
- Create a new experiment: our example is 2-qubits



- Score is translated into OPENQASM (a Quantum Assembly Language) behind the scene

Basic Operation

- Universal gate set is available



Pauli gates

Clifford gates

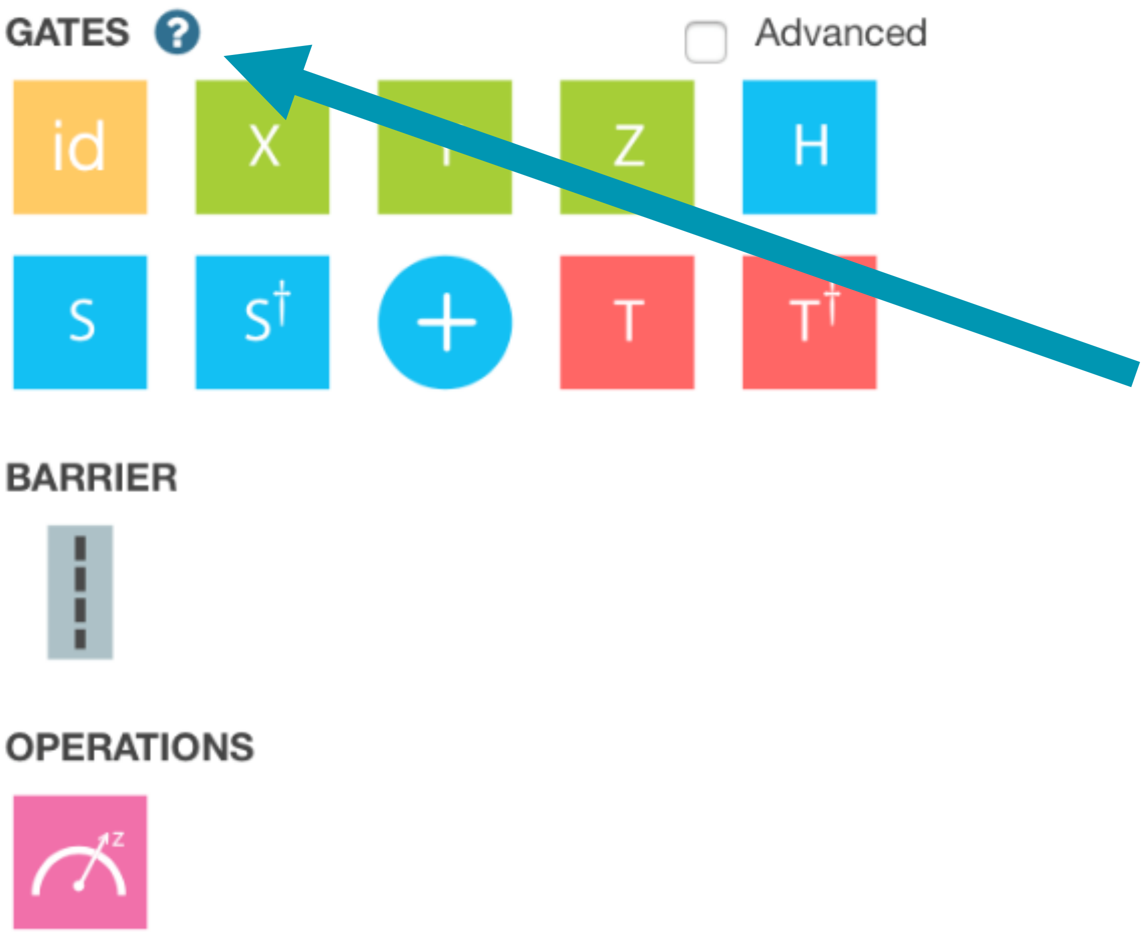
Universal
gate et

Barriers

Measurements

Basic Operation

- Universal gate set is available



Get additional information about gates by clicking here

Basic Operation

Help

×

U1

The first physical gate of the Quantum Experien
It is a one parameter single-qubit phase gat
with zero duration.

QASM Matrix

```
gate u2(phi,lambda) q {
 U(pi/2,phi,lambda) q;
 }
```

The second physical gate

QASM Matrix

U3

The third physical gate of the Quantum Experience.

```
[[cos(theta/2),-exp(1i*lambda)
```

QASM Matrix

id

The identity gate performs an idle operation on the qubit for a time equal to one unit of time.

QASM Matrix

X

The Pauli X gate is a π-rotation around the X axis and has the property that $X \rightarrow X, Z \rightarrow -Z$. Also referred to as a bit-flip.

QASM Matrix

Y

The Pauli Y gate is a π-rotation around the Y axis and has the property that $X \rightarrow -X, Z \rightarrow -Z$. This is both a bit-flip and a phase-flip, and satisfies $Y = XZ$.

QASM Matrix

Z

The Pauli Z gate is a π-rotation around the Z axis and has the property that $X \rightarrow -X, Z \rightarrow Z$. Also referred to as a phase-flip.

QASM Matrix

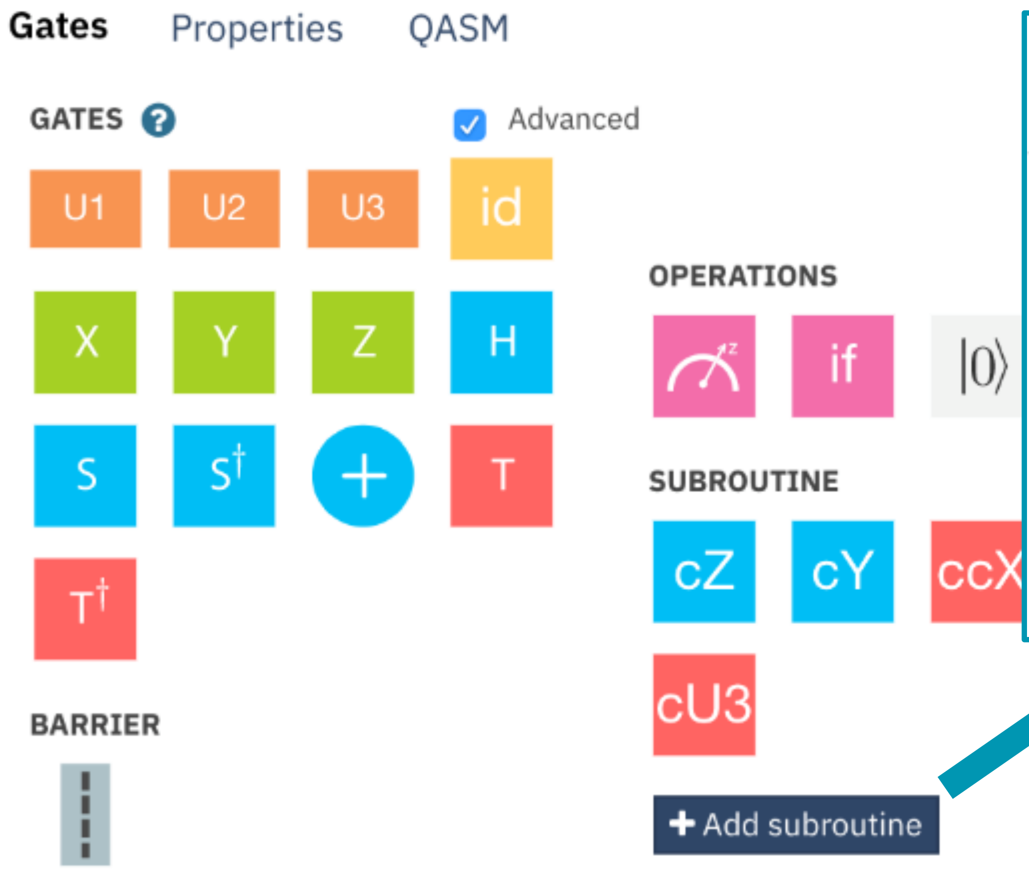
H

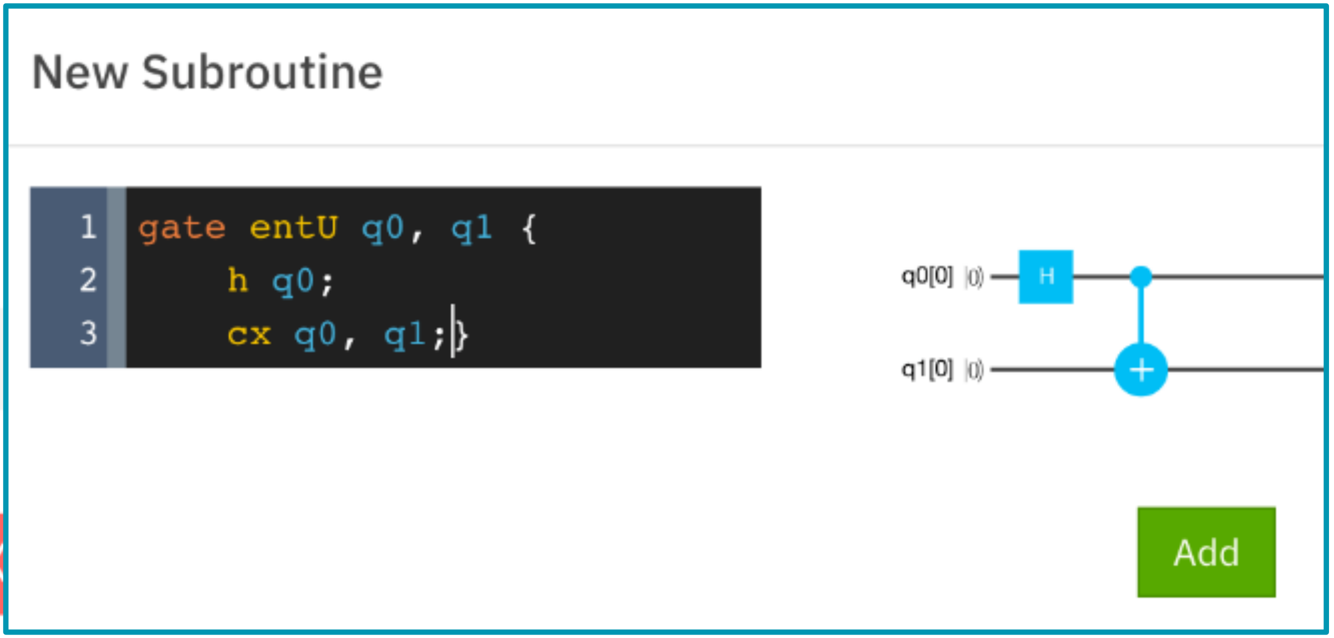
The Hadamard gate has the property that it maps $X \rightarrow Z$, and $Z \rightarrow X$. This gate is required to make superpositions.

QASM Matrix

Advanced Operations

- Advanced operations give access to arbitrary single qubit gates (u1, u2, u3)
- Advanced 2-qubit gate subroutines

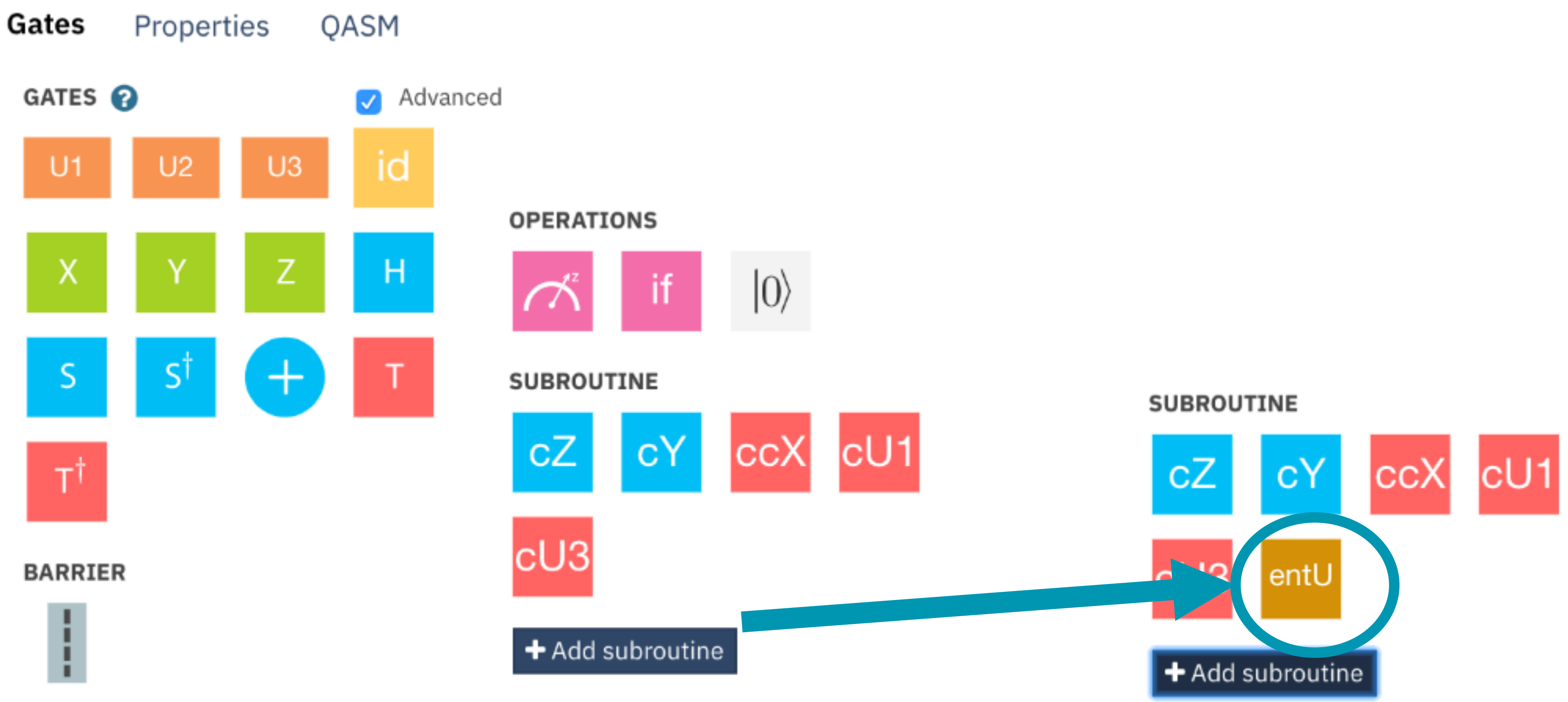




This gate generates a maximally entangled Bell state from the initial state

Advanced Operations

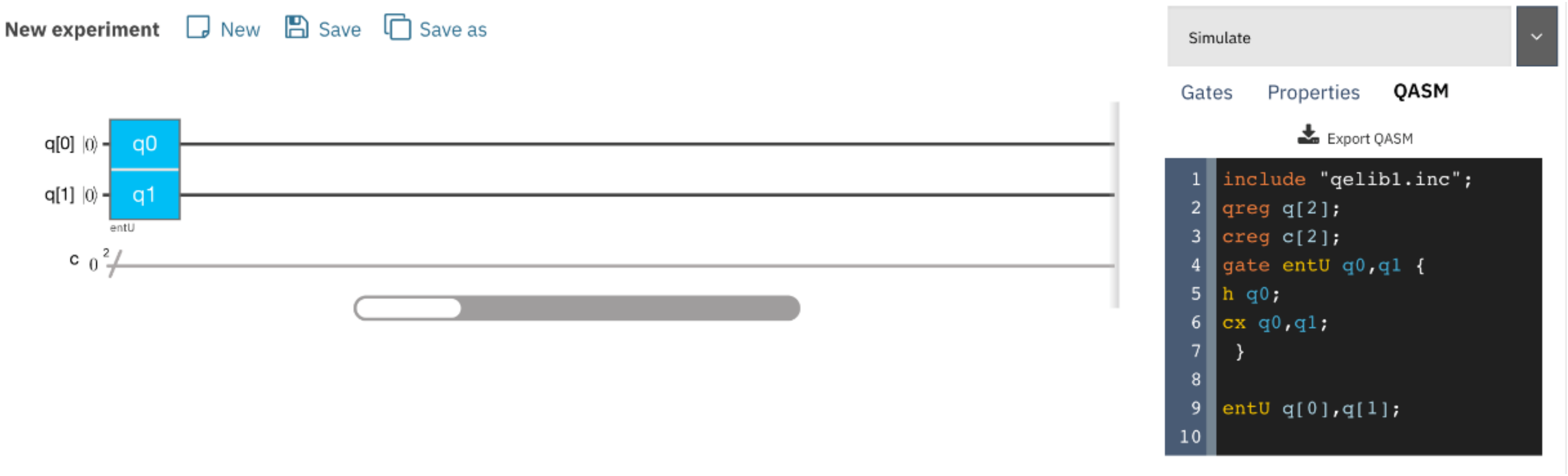
- Advanced operations give access to arbitrary single qubit gates (u1, u2, u3)
- Advanced 2-qubit gate subroutines



  25 January 2018

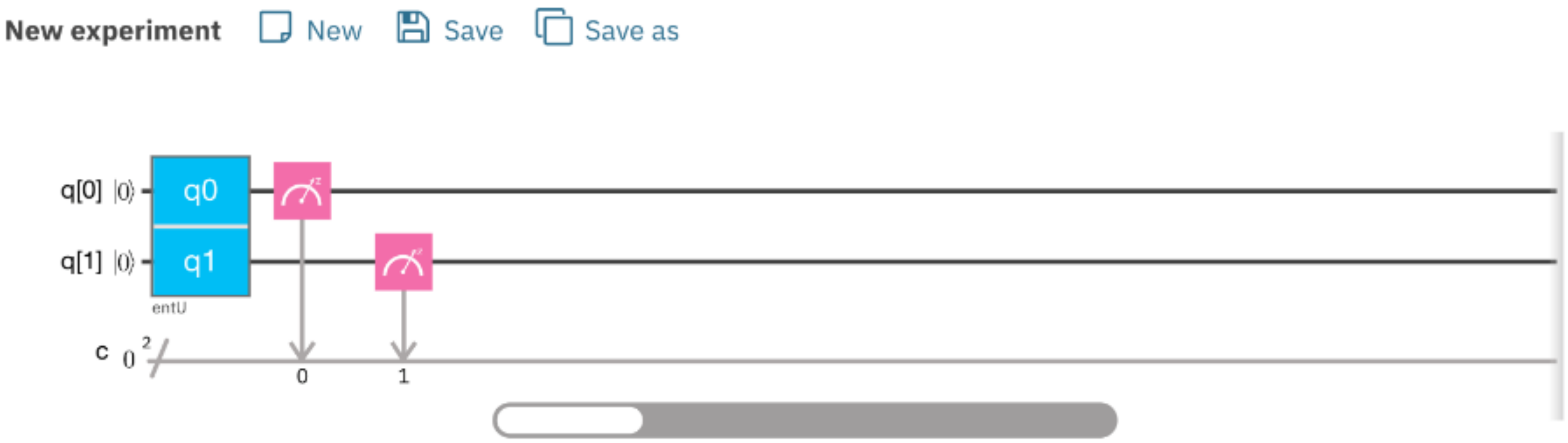
Generating an entangled state

- Lets use the gate to make a maximally entangled state.
- We clear the score and drag the new subroutine onto score



Generating an entangled state

- Lets use the gate to make a maximally entangled state.
- We clear the score and drag the new subroutine onto score
- **Next we add measurements**



Simulate

Gates Properties **QASM**

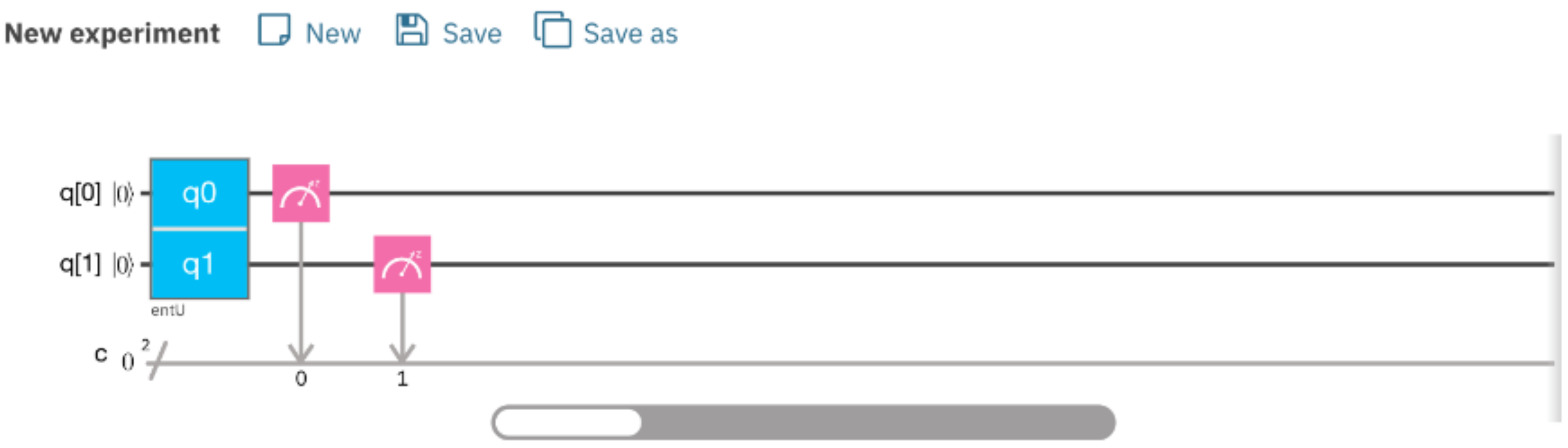
Export QASM

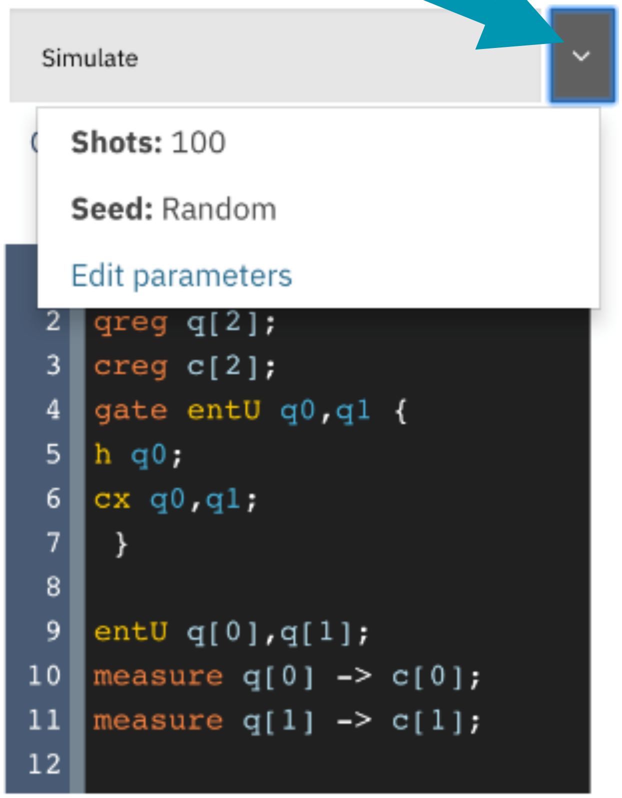
```
include "qelib1.inc";
qreg q[2];
creg c[2];
gate entU q0,q1 {
h q0;
cx q0,q1;
 }

entU q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];

```

Generating an entangled state

- Now we choose the simulation or experiment parameters
- Choose number of shots

Click here to choose number of shots for simulation or experiment

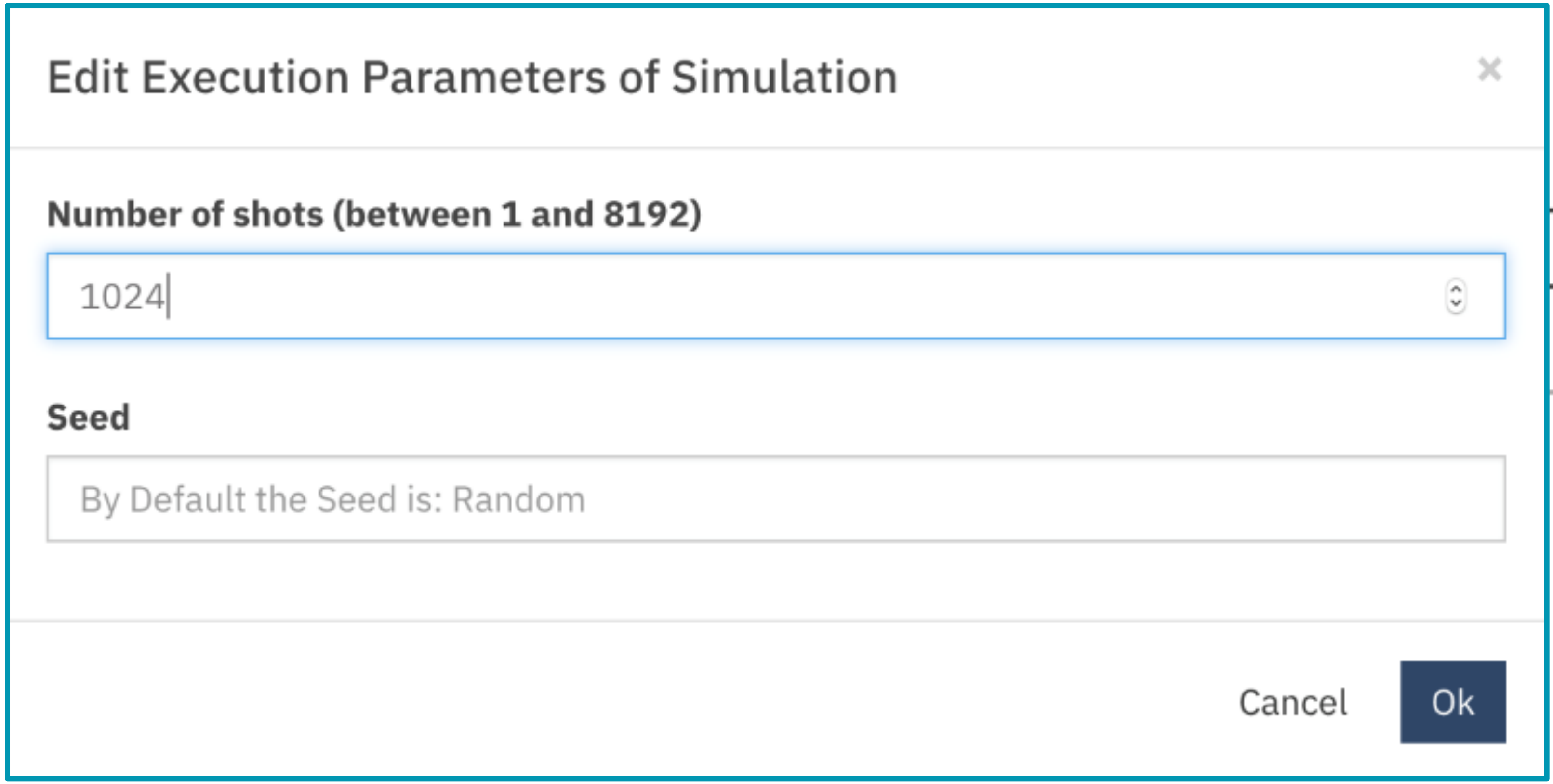


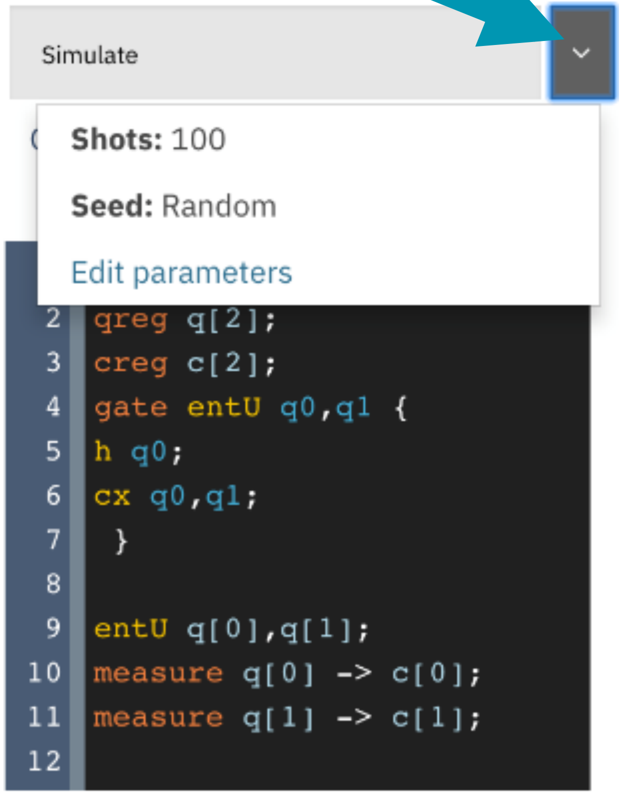


Generating an entangled state

- Now we choose the simulation or experiment parameters
- Choose number of shots

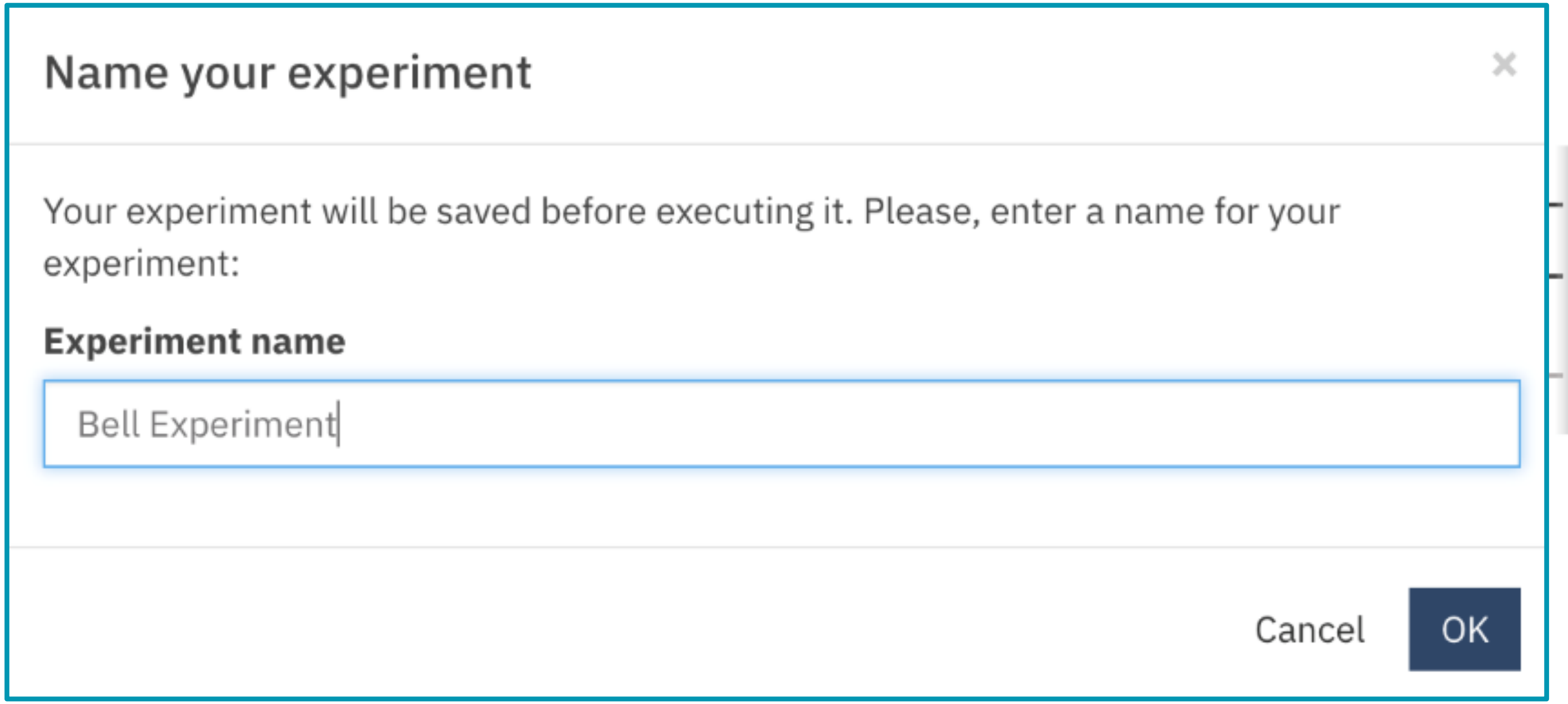
Click here to choose number of shots for simulation or experiment



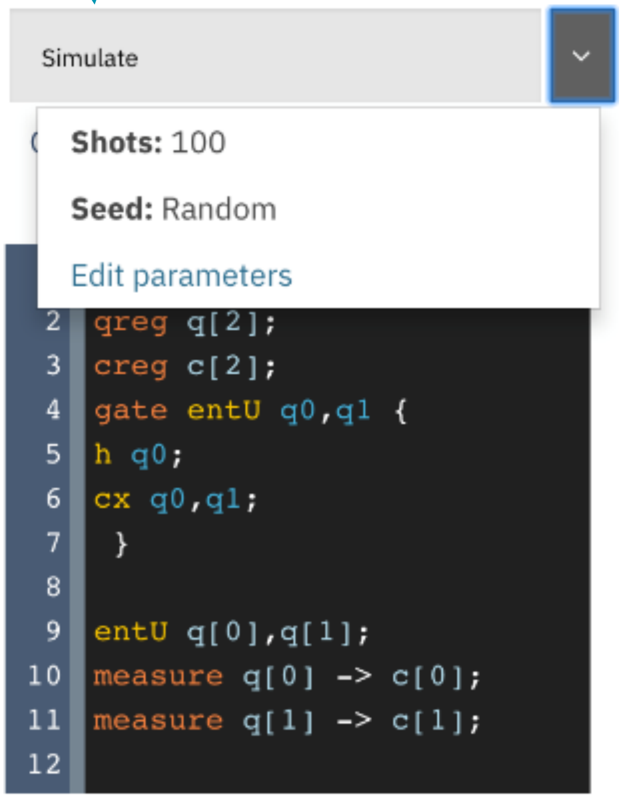


Generating an entangled state

- Now we choose the simulation or experiment parameters
- Choose number of shots
- Click simulate to run simulation



Click simulate to run simulation



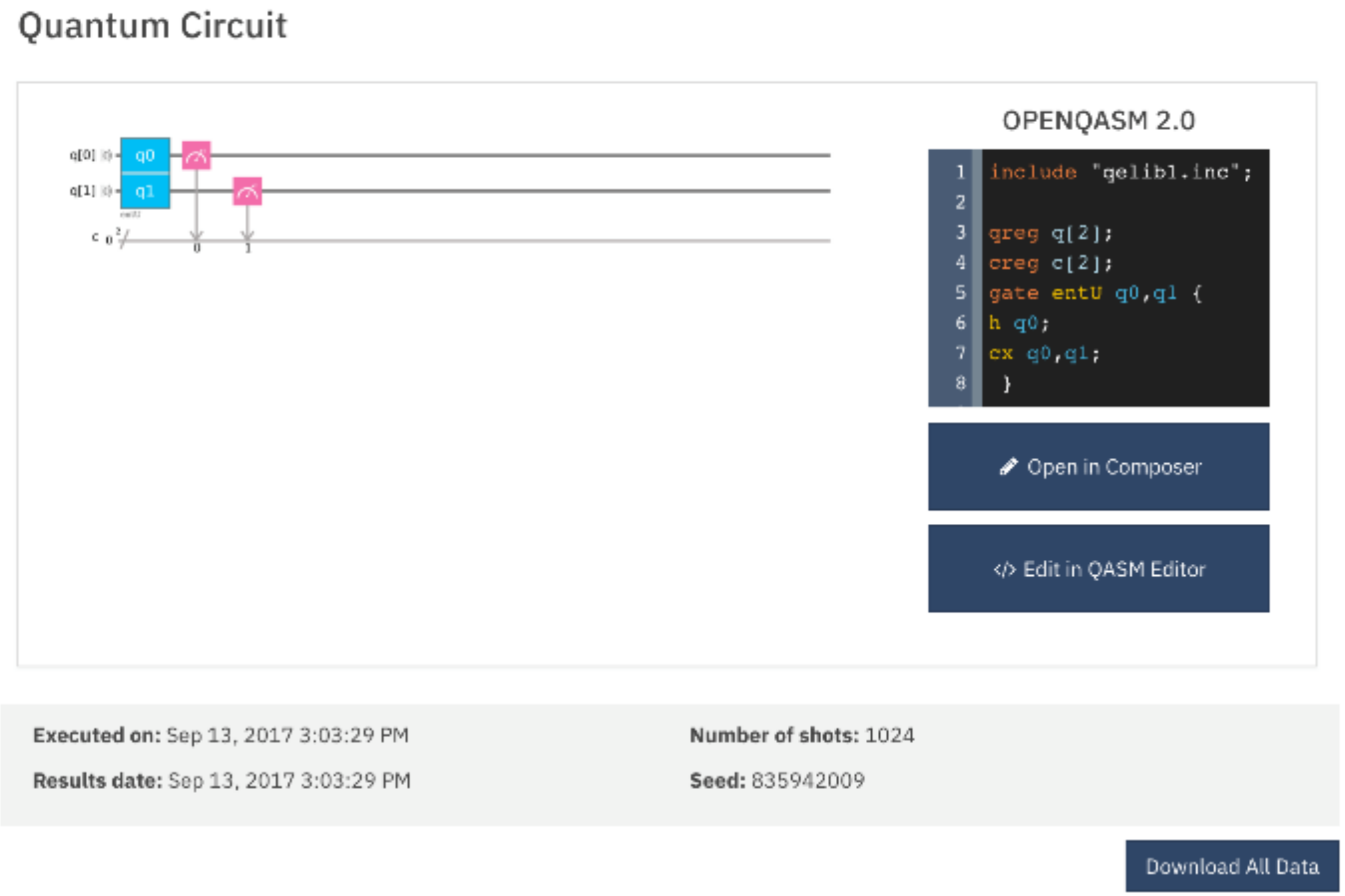
Experiment Results

- After running we may view the experiment results

Count data can be exported as CSV file

Experiment Results

- After running we may view the experiment results
- Results are saved to your account to view or run again later



Using the QISKit SDK

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) ✗ 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'
```


Programming a Quantum Experiment

The most important part of QISKit is the **QuantumProgram** class.

- Roughly equivalent to the score on web interface
- Used to build and store quantum circuits
- Import or export QASM
- Interface with backends to run experiments (on real hardware or simulators)

Designing an experiment

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)
```


Programming a Quantum Experiment

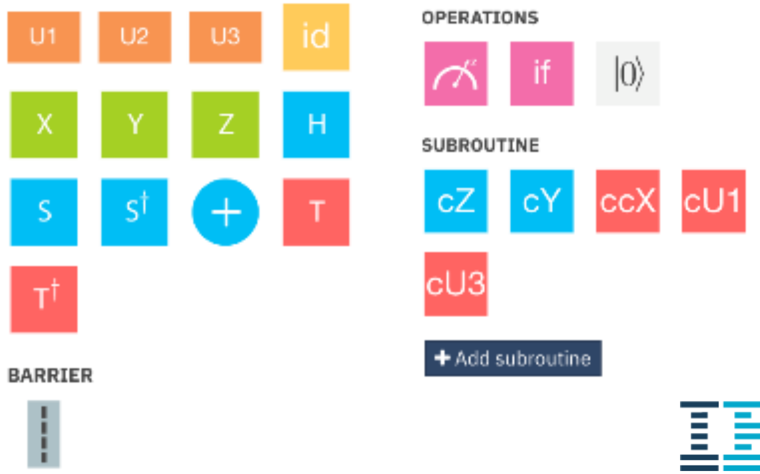
Adding a circuit to a QuantumProgram

- Next we create a new circuit to prepare a 2-qubit entangled state: $|\psi\rangle = \dfrac{|00\rangle + |11\rangle}{\sqrt{2}}$
- We must first create an empty circuit with a name (label). We use *"example"*
- Use circuit methods to add gates to the circuit:

```
In [3]: # Create a new empty circuit which uses these registers
        circ = qp.create_circuit('example', [qr], [cr])
        circ.h(qr[0])  # add Hadamard gates to qubit-0
        circ.cx(qr[0], qr[1])  # CNOT between qubit-0 and qubit-1
        circ.measure(qr, cr)  # measure qubits
```

Available circuit operation methods:

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



Programming a Quantum Experiment

- The quantum program now contains a single circuit that we may view:

```
In [4]: qp.get_circuit_names()

Out[4]: dict_keys(['example'])
```

- We may also view the QASM for this circuit:

```
In [5]: qasm = qp.get_qasm('example')
        print(qasm)

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];
```


Programming a Quantum Experiment

Executing the circuit on a simulator

- We may view available backends for running a circuit:

```
In [6]: qp.available_backends()

Out[6]: ['local_qasm_cpp_simulator', 'local_qasm_simulator', 'local_unitary_simul
        ator']
```

- To use online backends we must set our API token and URL as follows:

```
In [7]: qp.set_api(API_TOKEN, API_URL)
        qp.available_backends()

Out[7]: ['ibmqx3',
         'ibmqx2',
         'ibmqx_qasm_simulator',
         'local_qasm_cpp_simulator',
         'local_qasm_simulator',
         'local_unitary_simulator']
```

Programming a Quantum Experiment

Executing the circuit on a simulator

- We will run on the *'local_qasm_simulator'* which is an offline Python simulator.
- This is done using the ***execute*** command and returns a dictionary containing results:

```
In [7]: backend = 'local_qasm_simulator'
        shots = 1024
        results = qp.execute('example', backend='local_qasm_simulator', shots=1024)
        data = results.get_data('example')
        print(data)

        {'counts': {'00': 509, '11': 515}}
```

- The results contain a list of counts.
- Counts can also be accessed directly by method: ***results.get_counts('example')***
- **Note:** Different backends may return different types of results in the data dictionary
- **Note:** A list of many circuits can be submitted at once by the execute command

Simulator Features

We claimed that we prepared an entangled state? How can we verify this?

- Using the simulator in QISKit we may cheat and look directly at the state:
- **To do this create new circuit to prepare the state *without measurement:***

```
In [8]: # Create a new empty circuit which uses these registers
        circ = qp.create_circuit('bell', [qr], [cr])
        circ.h(qr[0])  # add Hadamard gates to qubit-0
        circ.cx(qr[0], qr[1])  # CNOT between qubit-0 and qubit-1
```

- **Execute:** using shots = 1 to obtain the quantum state vector

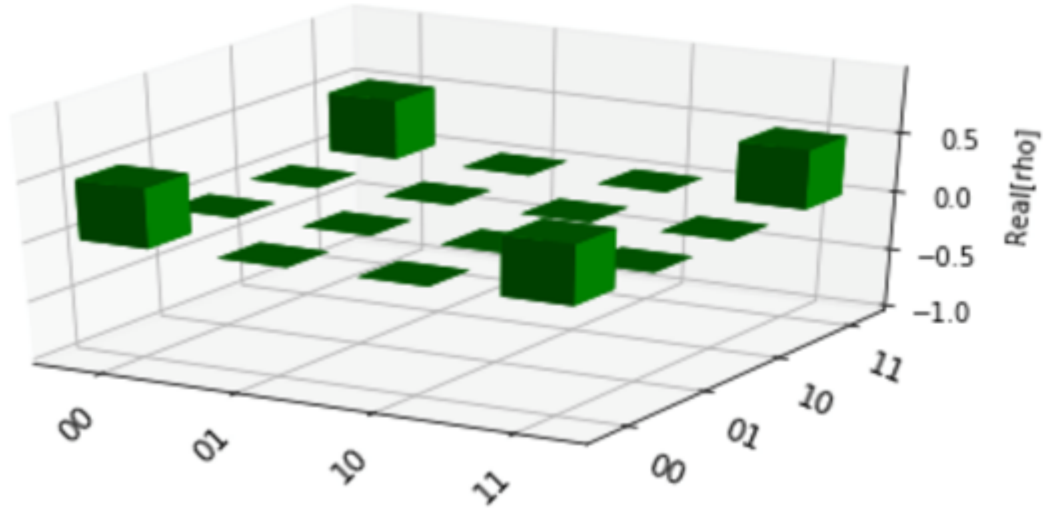
```
In [9]: # Execute on simulator for 1 shot
        backend = 'local_qasm_simulator'
        shots = 1
        results = qp.execute('bell', backend='local_qasm_simulator', shots=shots)
        data = results.get_data('bell')
        print(data)

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


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')
```



Combining Circuits

How would we verify the state is entangled on a real experiment?

- We need to measure the state in different bases.
- Create a new measurement circuit

```
In [11]: measZZ = qp.create_circuit('measZZ', [qr], [cr])
         measZZ.measure(qr, cr)
         print(qp.get_qasm('measZZ'))

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


Combining Circuits

How would we verify the state is entangled on a real experiment?

- We need to measure the state in different bases.
- Create a new measurement circuit
- The measurement circuit can be appended to another circuit using the **+** operator
- This new circuit can be added to the quantum program using the **add_circuit** method

```
In [12]: qp.add_circuit('example_mZZ', circ + measZZ)
         print(qp.get_qasm('example_mZZ'))

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];
```


Combining Circuits

How would we verify the state is entangled on a real experiment?

- We need to measure the state in different bases.
- ***We can repeat this for additional measurement circuits in different bases***

In [13]:
```
# Create circuit to measure both qubits in X basis
measXX = qp.create_circuit('measXX', [qr], [cr])
measXX.h(qr)
measXX.measure(qr, cr)

# Create circuit to measure both qubits in Y basis
measYY = qp.create_circuit('measYY', [qr], [cr])
measYY.h(qr)
measYY.s(qr)
measYY.measure(qr,cr)

# Add circuits to QuantumProgram
qp.add_circuit('example_mXX', circ + measXX)
qp.add_circuit('example_mYY', circ + measYY)
print(qp.get_circuit_names())
```

```
dict_keys(['example', 'bell', 'measZZ', 'example_mZZ', 'measXX', 'measY
Y', 'example_mXX', 'example_mYY'])
```


Combining Circuits

How would we verify the state is entangled on a real experiment?

- We need to measure the state in different bases.
- ***Run these circuits on a backend and get the counts:***

```
In [14]: backend = 'local_qasm_simulator'
         shots = 1024
         meas_circs = ['example_mZZ', 'example_mXX', 'example_mYY']
         meas_res = qp.execute(meas_circs, backend=backend, shots=shots)

         for c in meas_circs:
             print('Measured counts:', c)
             print(meas_res.get_counts(c))

Measured counts: example_mZZ
{'11': 517, '00': 507}
Measured counts: example_mXX
{'11': 520, '00': 504}
Measured counts: example_mYY
{'00': 498, '11': 526}
```


Explore QISKit

- **What next?**
- Explore the QISKit tutorial Jupyter notebooks. A good start are the ones in Section 2:

2. Exploring quantum information concepts

The next set of notebooks shows how you can explore some simple concepts of quantum information science.

- Superposition and Entanglement - how to make simple quantum states on one and two qubits, and demonstrates concepts such as quantum superpositions and entanglement.
- Single-qubit States: Amplitude and Phase - discusses more complicated single-qubit states.
- Single-qubit Quantum Random Access Coding - how superpositions of one-qubit quantum states can be used to encode two and three bits into one qubit, and how measurements can be used to decode any one bit with a success probability of more than half.
- Two-qubit Quantum Random Access Coding - how superposition and entanglement can be used to encode seven bits of information into two qubits, such that any one of seven bits can be recovered probabilistically.
- Entanglement Revisited - the CHSH inequality, and extensions for three qubits (Mermin).
- Quantum Teleportation - introduces quantum teleportation.
- Quantum Superdense Coding - introduces the concept of superdense coding.
- Quantum Fourier Transform - introduces the quantum Fourier transform.
- Vaidman Detection Test - demonstrates interaction free measurement through the Vaidman bomb detection test.

