

Week 9

Lecture 17 – IBM and Optimisations

14.1 QUI compared to IBM

14.2 QASM and QISKit

14.3 Optimizing circuits

Lecture 18 – IBM Quantum Experience

Introduction to IBM Quantum Experience: Guest Lecture

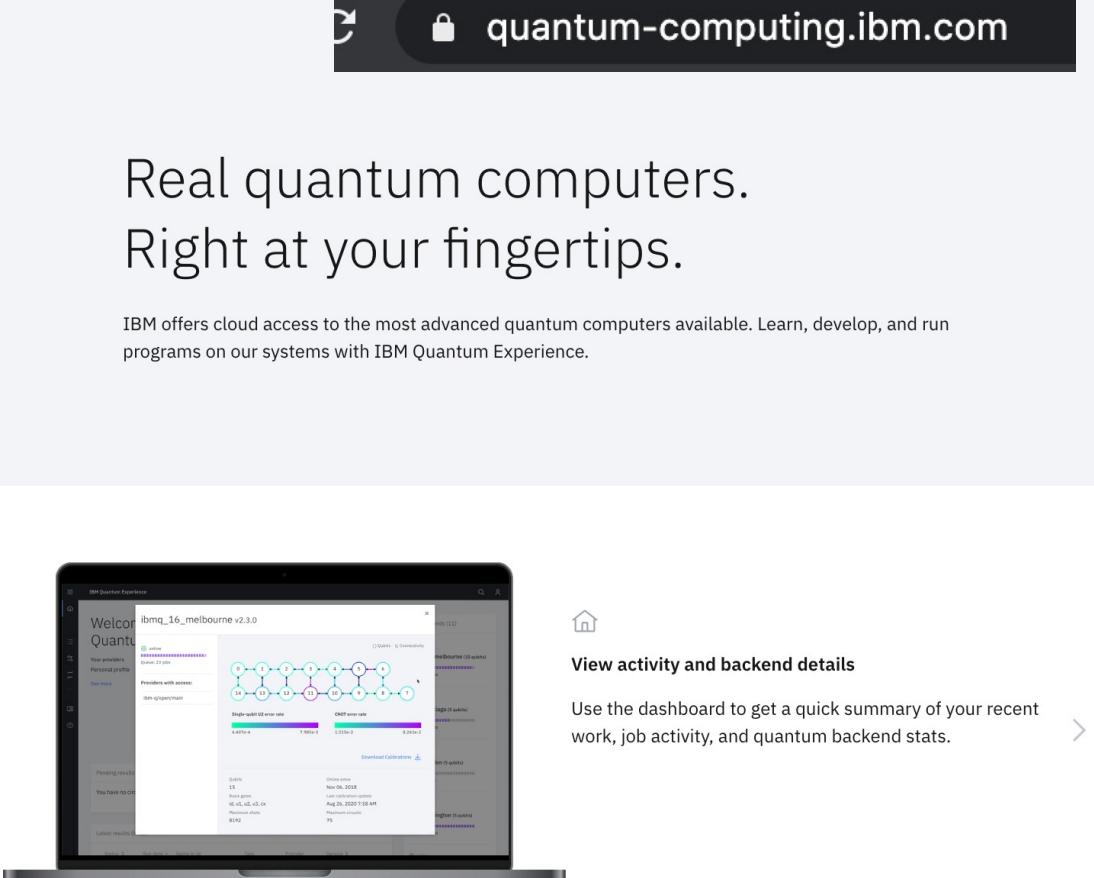
Lab 9

Using the IBM system

IBM and Optimisation

Lecture 17

The IBM Quantum Computing System



Real quantum computers.
Right at your fingertips.

IBM offers cloud access to the most advanced quantum computers available. Learn, develop, and run programs on our systems with IBM Quantum Experience.

Sign in to get started

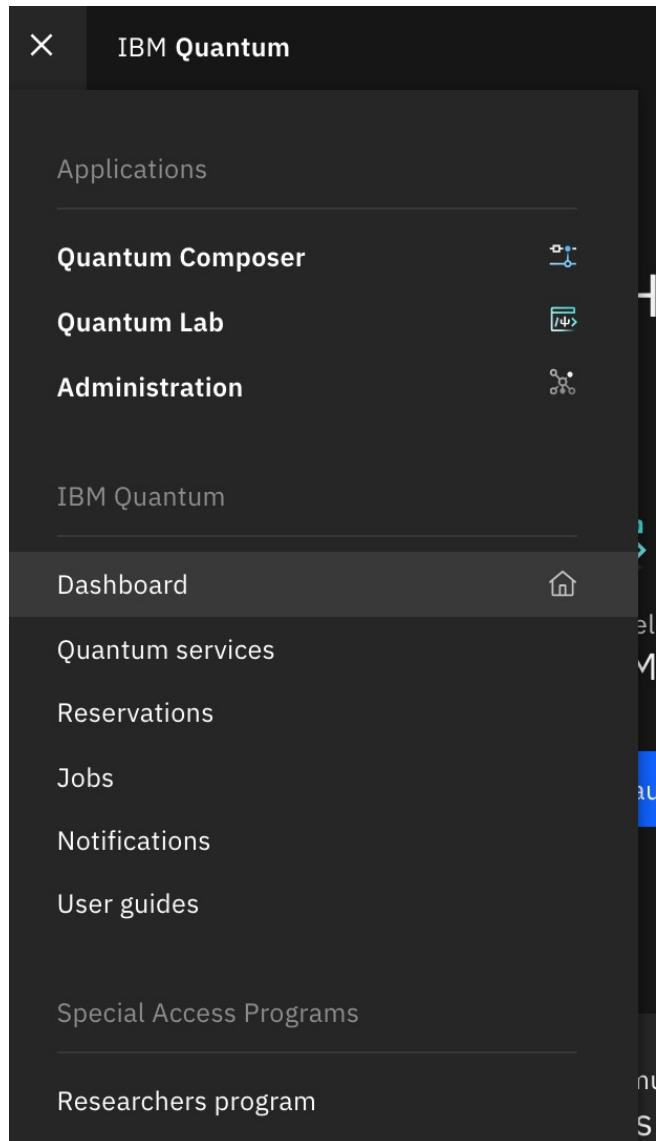
IBMid

G O in Twitter Email

New to IBM Quantum Experience?
Create an IBMid account.

Sign up using your university email before Thursday!

IBM's Circuit Composer



The screenshot shows the IBM Quantum interface. At the top left is a close button (X) and the title "IBM Quantum". Below the title is a sidebar with the following items:

- Applications
- Quantum Composer** (selected, indicated by a blue icon)
- Quantum Lab
- Administration

Below the sidebar is a main content area titled "IBM Quantum". It contains the following items:

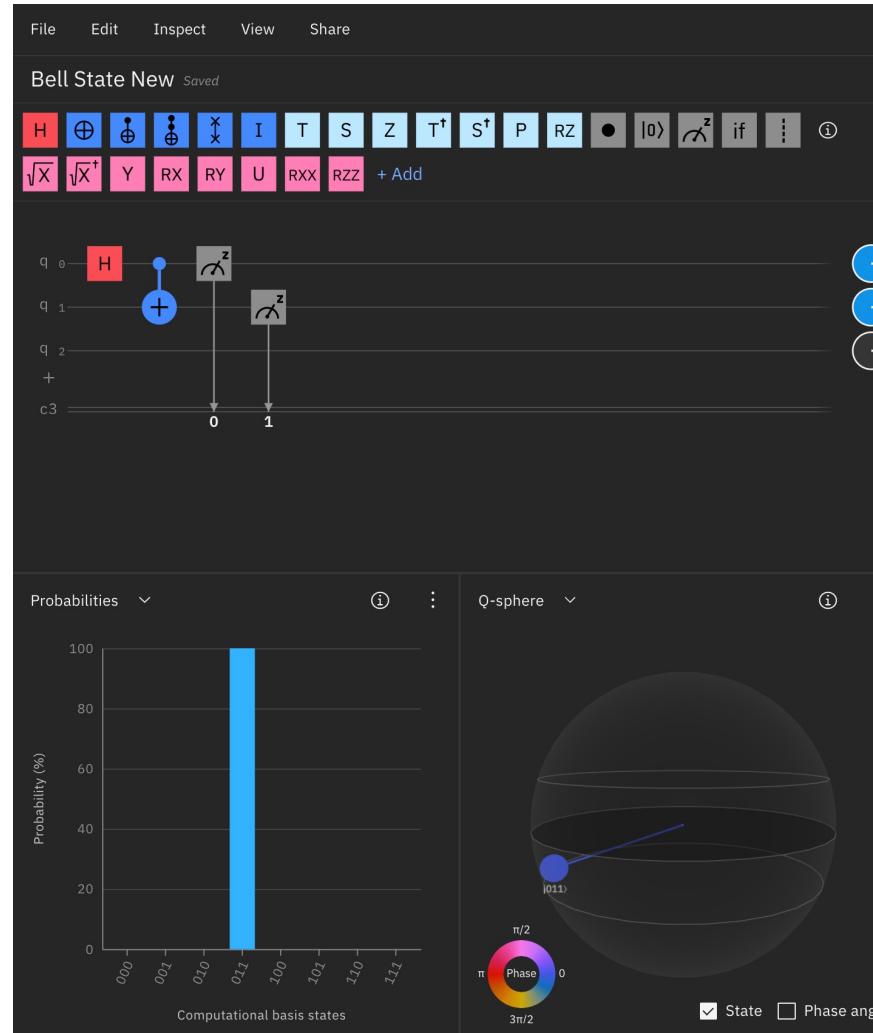
- Dashboard (selected, indicated by a blue icon)
- Quantum services
- Reservations
- Jobs
- Notifications
- User guides

At the bottom of the content area is a section titled "Special Access Programs".

Access through the left sidebar

Creating a Bell State Circuit

Drag and drop gates onto your circuit:

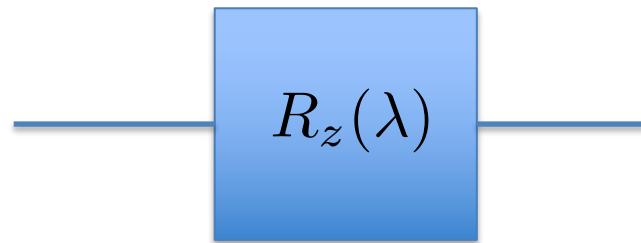


U1

U1 is a rotation around Z by angle lambda, which is equivalent to a rotation around the z-axis by an angle lambda

$$U_1 = \begin{bmatrix} 1 & 0 \\ 0 & \exp i\lambda \end{bmatrix}$$

Most easily understood as:



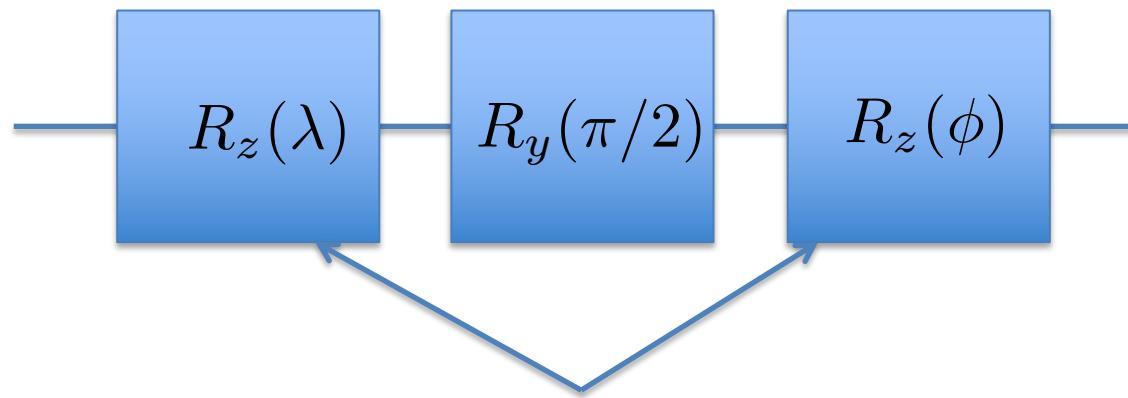
In the QUI, to emulate these z-rotations, use a global phase of $\lambda/2$.
No global phase for the y-rotation.

U2

The U2 operation is given by

$$U_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -\exp(i\lambda) \\ \exp(i\phi) & \exp(i\lambda + i\phi) \end{bmatrix}$$

Which can be represented as:



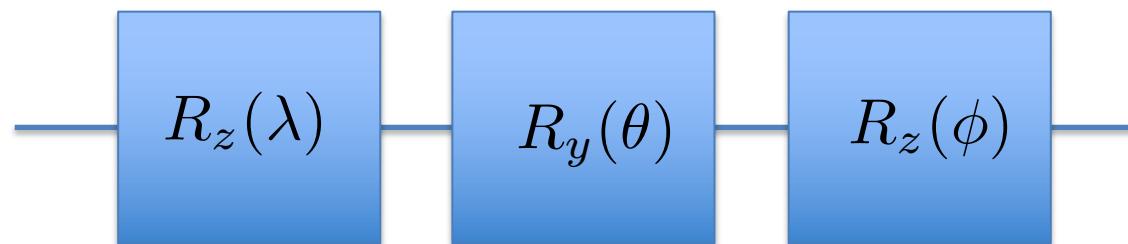
In the QUI, to emulate these z-rotations, use a global phase of theta/2.
No global phase for the y-rotation.

U3

The matrix of a U3 rotation is:

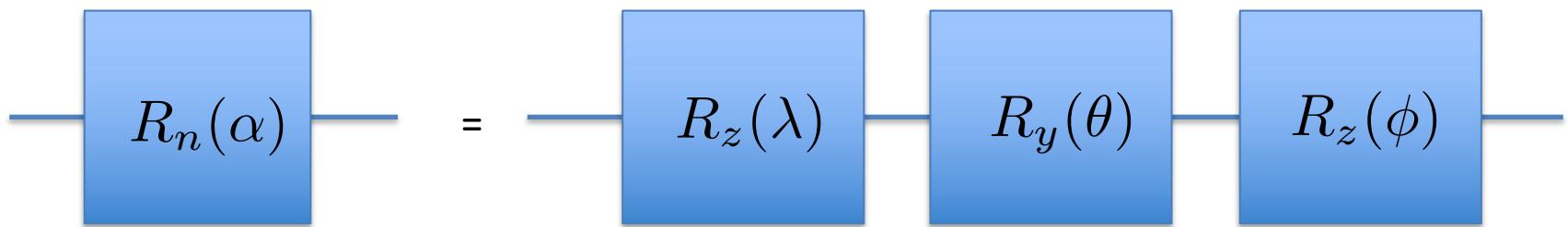
$$U_3 = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos \theta/2 & -\exp(i\lambda) \sin(\theta/2) \\ \exp(i\phi) \sin(\theta/2) & \exp(i\lambda + i\phi) \cos(\theta/2) \end{bmatrix}$$

As a circuit:



Euler Angle Decomposition

Any rotation can be represented as a rotation around orthogonal axes:



QUI

IBM Quantum Experience

Converting to and from Euler angles

General form of arbitrary rotation about an unit axis $\mathbf{n}=(n_x, n_y, n_z)$:

$$\begin{aligned}
 R_n(\alpha) &= \cos \frac{\alpha}{2} I - i \sin \frac{\alpha}{2} \hat{n} \cdot \sigma \\
 &= \begin{bmatrix} \cos \frac{\alpha}{2} - i n_z \sin \frac{\alpha}{2} & \sin \frac{\alpha}{2} (-i n_x - n_y) \\ \sin \frac{\alpha}{2} (-i n_x + n_y) & \cos \frac{\alpha}{2} + i n_z \sin \frac{\alpha}{2} \end{bmatrix}
 \end{aligned}$$

Euler angle rotations (with global phase = 0):

$$U_3 = \begin{bmatrix} e^{-i(\lambda+\phi)/2} \cos(\theta/2) & -e^{i(\lambda-\phi)/2} \sin(\theta/2) \\ e^{i(-\lambda+\phi)/2} \sin(\theta/2) & e^{i(\lambda+\phi)/2} \cos(\theta/2) \end{bmatrix}$$

Write out the matrix (with determinant 1) and equate elements.

Equating angles

$$\cos \frac{\alpha}{2} = \cos \frac{\lambda + \phi}{2} \cos \frac{\theta}{2}$$

$$\sin \left(\frac{\alpha}{2} \right) n_x = \sin \frac{\lambda - \phi}{2} \sin \frac{\theta}{2}$$

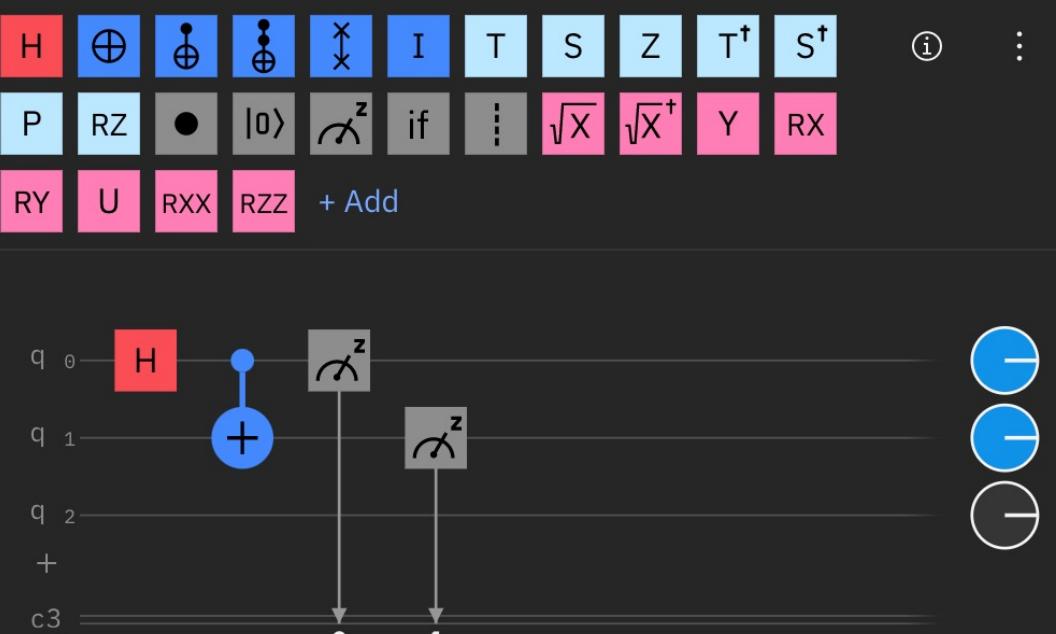
$$\sin \left(\frac{\alpha}{2} \right) n_y = \cos \frac{\lambda - \phi}{2} \sin \frac{\theta}{2}$$

$$\sin \left(\frac{\alpha}{2} \right) n_z = \sin \frac{\lambda + \phi}{2} \cos \frac{\theta}{2}$$

QASM – Quantum Assembly language

Bell State New *Saved*

Visualizations [se](#)



OpenQASM 2.0 ▾

[Open in Quantum Lab](#)

```

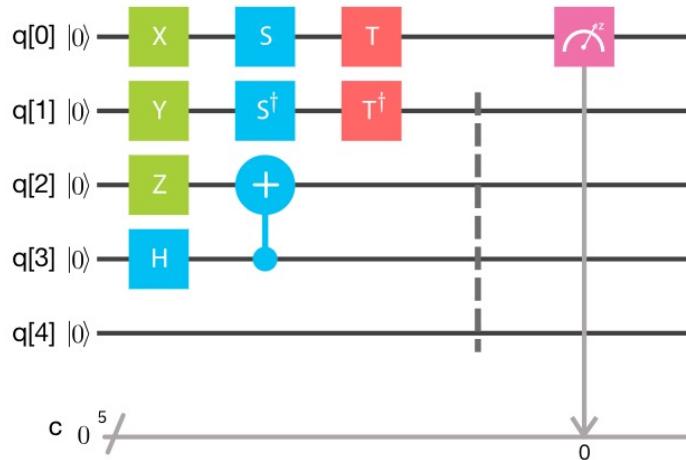
1 OPENQASM 2.0;
2 include "qelib1.inc";
3
4 qreg q[3];
5 creg c[3];
6
7 h q[0];
8 cx q[0],q[1];
9 measure q[0] -> c[0];
10 measure q[1] -> c[1];

```

QASM Syntax

```
1 include "qelib1.inc";  
2  
3 // This is a comment  
4 qreg q[5];  
5 creg c[5];  
6  
7 x q[0];  
8 y q[1];  
9 z q[2];  
10 h q[3];  
11  
12 s q[0];  
13 sdg q[1];  
14  
15 cx q[3],q[2];  
16 t q[0];  
17 tdg q[1];|  
18  
19 barrier q[1],q[2],q[3],q[4];  
20 measure q[0] -> c[0];  
21
```

Semi-Colons



Comment

QASM

Hidden first line: **OPENQASM 2.0;**

```
1 include "qelib1.inc";           Include standard definitions
2 qreg q[5];                    Declare quantum register
3 creg c[5];                    Declare classical register
4
5 x q[0];
6 y q[1];
7 z q[2];                      Single qubit gates
8 h q[3];
9
10 s q[0];
11 sdg q[1];
12 cx q[3],q[2];                CNOT gate
13                                         (control first parameter, target second)
14 t q[0];
15 tdg q[1];                    Dagger indicated by "dg"
16
17 barrier q[1],q[2],q[3],q[4];  Barrier (don't optimize across it)
18 measure q[0] -> c[0];        Measure qubits to classical register
19
```

Qiskit

There is also a Python interface to IBM Quantum Experience.

It is required to make use of the larger machines.

You can:

- Authenticate with the system
- Construct circuits (ie. python which translates to QASM)
- Submit jobs, and check for results
- Receive the results of jobs

Python works well with Jupyter interface.

We will use this later when we use the 16 qubit quantum computer.

Qiskit for the Bell State Circuit

Read only

Open in Quantum Lab

H \oplus \ominus $\oplus\ominus$ \otimes I T S Z T^\dagger S^\dagger

① ⋮

Qiskit

P RZ ● $|0\rangle$ \curvearrowright^z if ⋮ \sqrt{X} \sqrt{X}^\dagger Y RX

RY U RXX RZZ + Add



```

1 from qiskit import QuantumRegister, ClassicalRegister,
2   QuantumCircuit
3
4 qreg_q = QuantumRegister(3, 'q')
5 creg_c = ClassicalRegister(3, 'c')
6 circuit = QuantumCircuit(qreg_q, creg_c)
7
8 circuit.h(qreg_q[0])
9 circuit.cx(qreg_q[0], qreg_q[1])
10 circuit.measure(qreg_q[0], creg_c[0])
11 circuit.measure(qreg_q[1], creg_c[1])

```

Qiskit

Secure | https://qiskit.org

Qiskit™ Terra Aqua Tools Fun



Qiskit

An open-source quantum computing framework for leveraging today's quantum processors and conducting research

 GitHub

 Join the Slack community

Try it out

Introducing VSCode extension!

Simplifying Qiskit to make developing quantum circuits and applications faster

[More information](#)

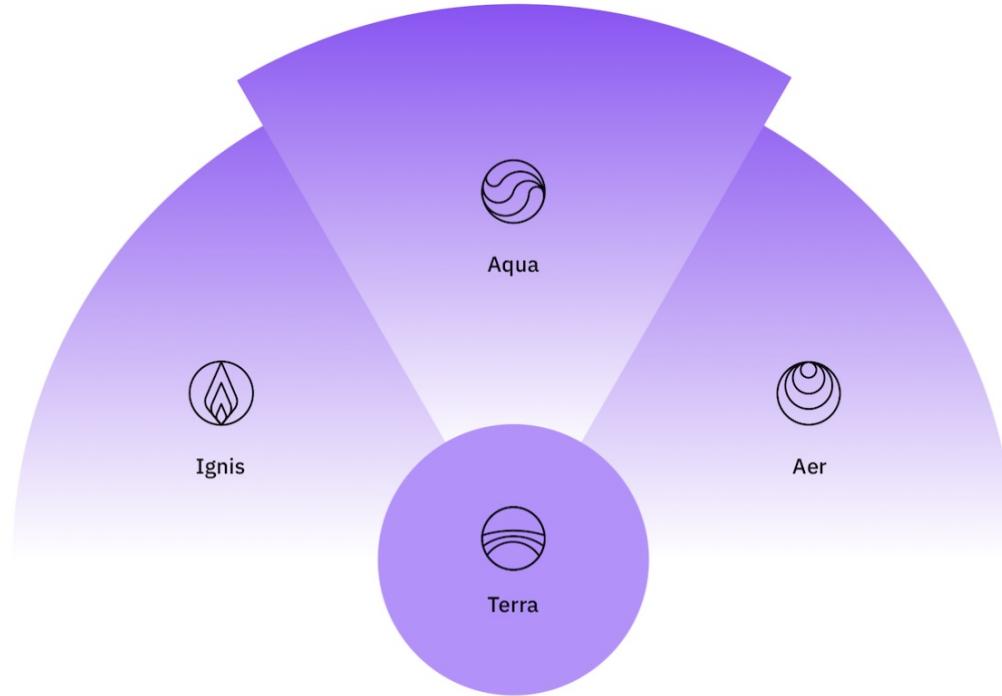
Getting started with Qiskit

In this episode Doug McClure, Qiskitter at IBM, introduces us to Qiskit and its functions. You'll learn all about how to run your first quantum program on real IBM Q hardware.

Lots of examples in the github repository.

IBM's Qiskit

Terra, Aer, Ignis, Aqua



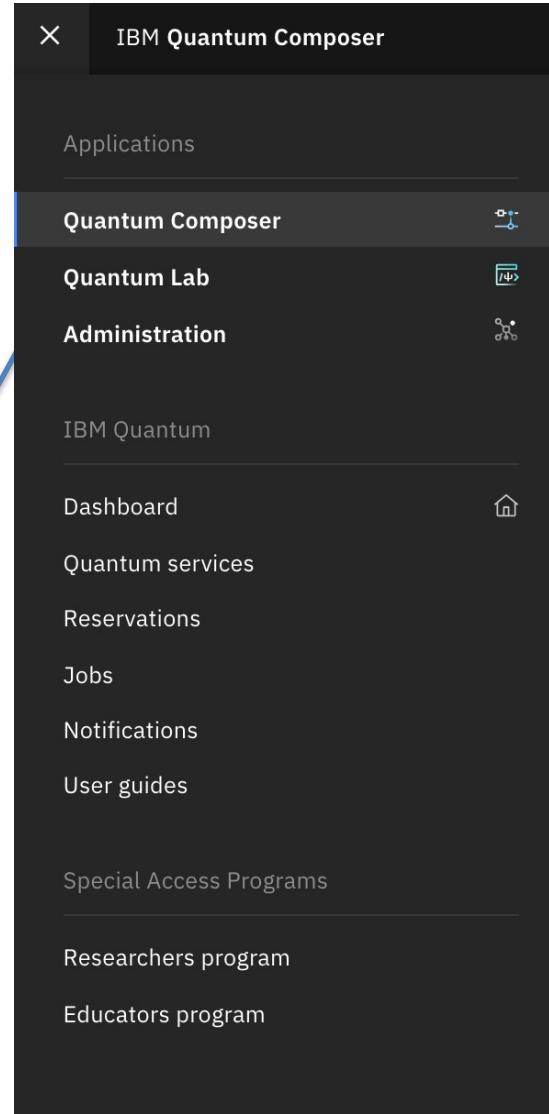
Tera (Earth): Access to IBM Q Devices through python interface

Aer (Air): Classical simulation of quantum algorithms/circuits

Ignis (Fire): Characterisation of errors, tomography

Aqua (Water): Large selection of quantum algorithms

Starting a new Workbook



Select Quantum Lab in menu

We will loosely follow the “Getting Started with Qiskit” tutorial from IBM,



Getting Started with Qiskit

Here, we provide an overview of working with Qiskit. Qiskit provides the basic building blocks necessary to program quantum computers. The fundamental unit of Qiskit is the **quantum circuit**. A workflow using Qiskit consists of two stages: **Build** and **Execute**. **Build** allows you to make different quantum circuits that represent the problem you are solving, and **Execute** allows you to run them on different backends. After the jobs have been run, the data is collected. There are methods for putting this data together, depending on the program. This either gives you the answer you wanted, or allows you to make a better program for the next instance.

```
In [1]: import numpy as np
from qiskit import *
%matplotlib inline
```

Python Primer (if required)

Some Python Basics

```
In [2]: a=6
         b=7
         life = a*b
         life
```

```
Out[2]: 42
```

Similar to many other imperative languages you may know for numerical work: (C/C++, MATLAB, R, FORTRAN, Julia) and often used for data processing.

Defining and calling Functions

def keyword indicates a new function

No types on parameters

```
def square(x):  
    # This is a comment  
    return x*x
```

Colon

Comment

Whitespace is significant in python.
Indentation indicates a new block.

No semicolons.
Newline is the end of a statement

Calling a function:

```
square(4)
```

```
square(x=4)
```

Named parameters

Lists and for loops

Lists store a sequence of values. Square brackets indicate a list:

```
[ "This", "is", "a", "list"]  
  
primes = [2, 3, 5, 7, 11]
```

Eg. For loops often use lists:

```
for p in primes:  
    print(p)
```

2
3
5
7
11

Accessing an individual element.
0-based!

```
primes[2]
```

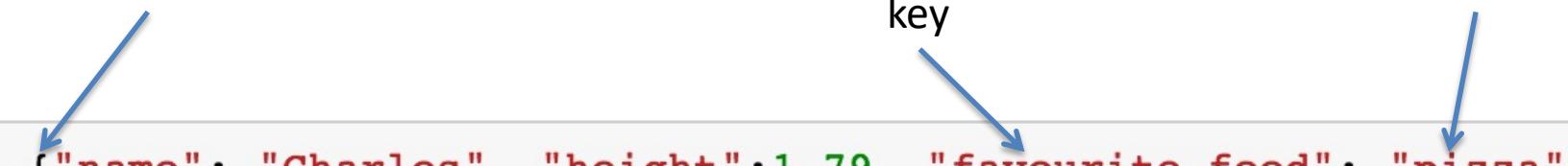
5

Dictionaries

Dictionaries store key-value pairs.

Curly braces indicate a dictionary

```
me = {"name": "Charles", "height": 1.79, "favourite_food": "pizza"}  
me["favourite_food"]  
  
'pizza'
```



```
me["favourite_food"] = "sweet and sour pork"  
me["favourite_food"]  
  
'sweet and sour pork'
```

Importing other libraries

```
import numpy as np
x = np.matrix([[0,1],
               [1,0]])
```

Importing a module (“as np” is optional).
numpy gives similar functionality to MATLAB

Calling functions from that module.
Here creating an X matrix.

Or import individual functions and classes:

```
from qiskit import QuantumProgram
from qiskit import available_backends, execute, get_backend, compile
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, QISKitError
```

qiskit is an Python library/API for interacting with IBM’s quantum computers remotely.

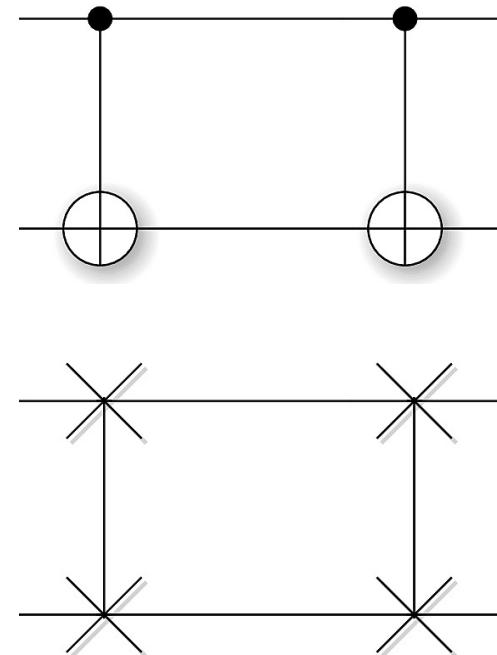
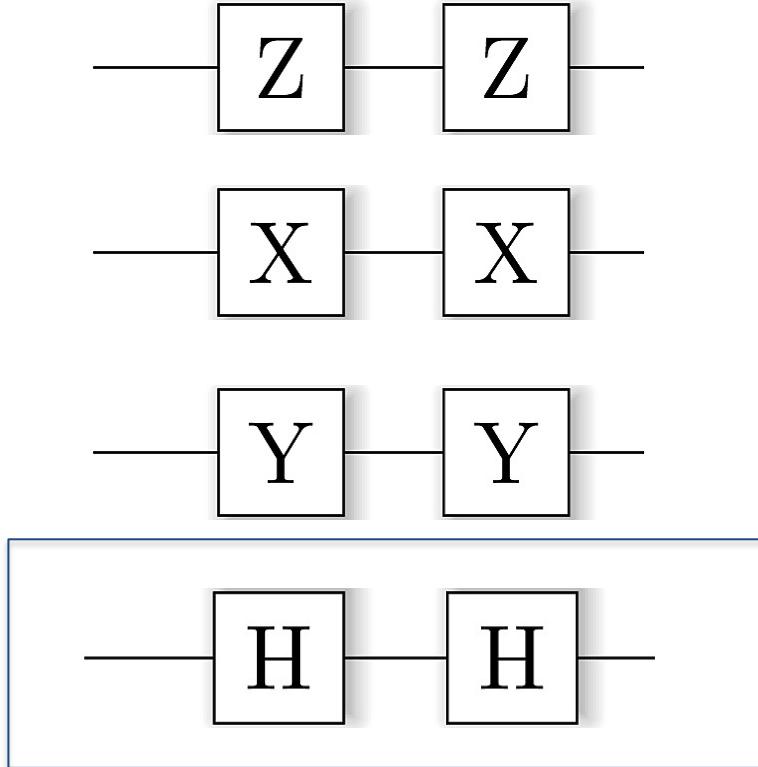
Qiskit Demonstration

(in browser)

Circuit Optimisation

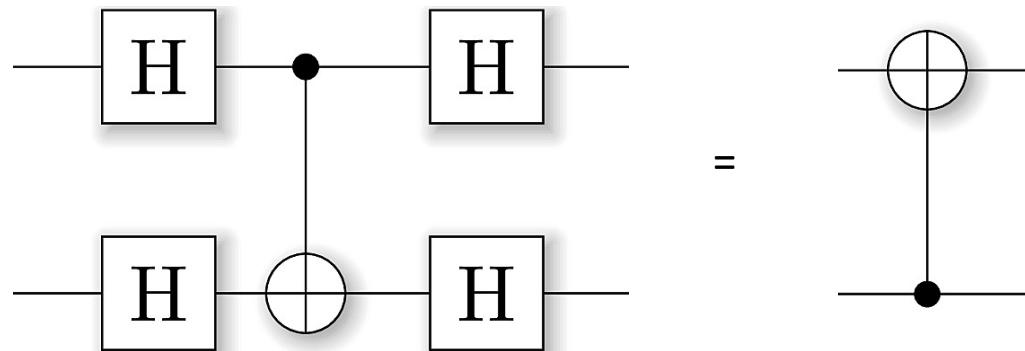
Many gates square to identity

Pro Tip: Most physicists looking at quantum circuit diagrams aren't multiplying matrices in their head. They're identifying common patterns.



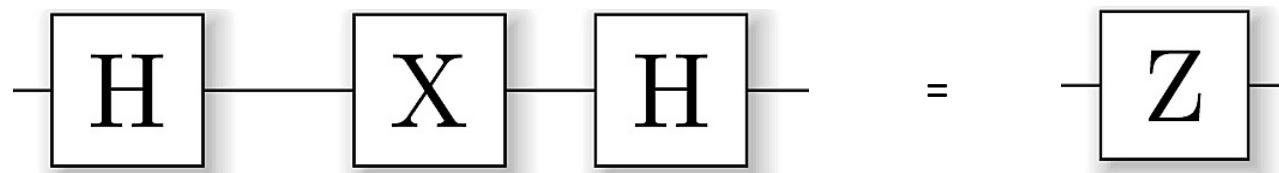
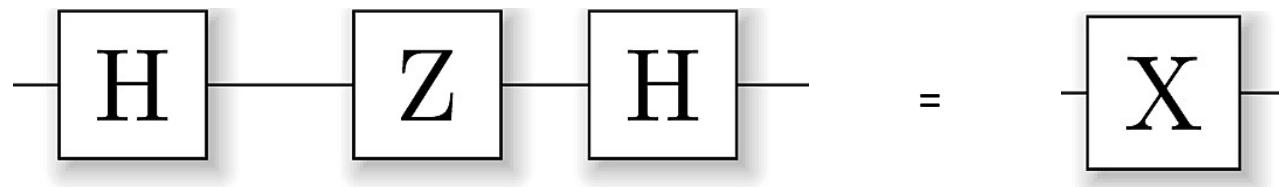
All of these combinations square to the identity (do nothing)

Circuit identity: Inverted CNOT

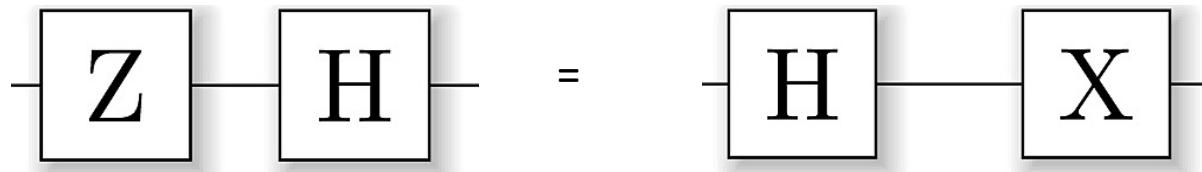
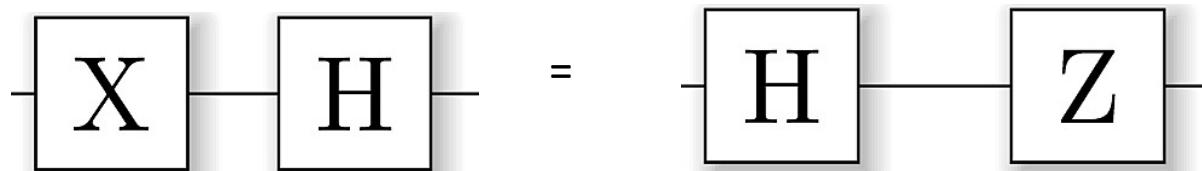


Exercise: You can verify this by writing out the matrices and multiplying!

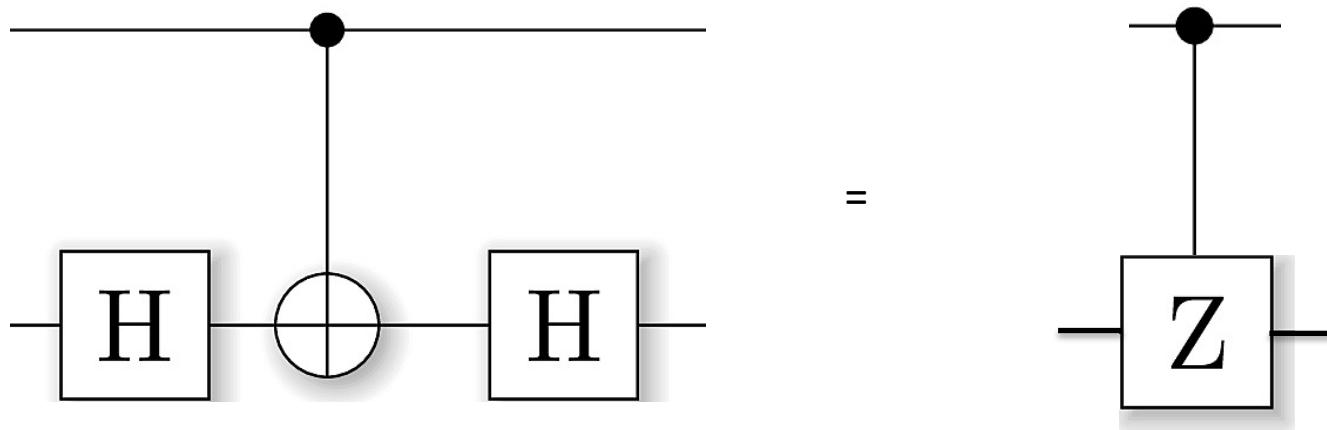
Conjugating with Hadamard



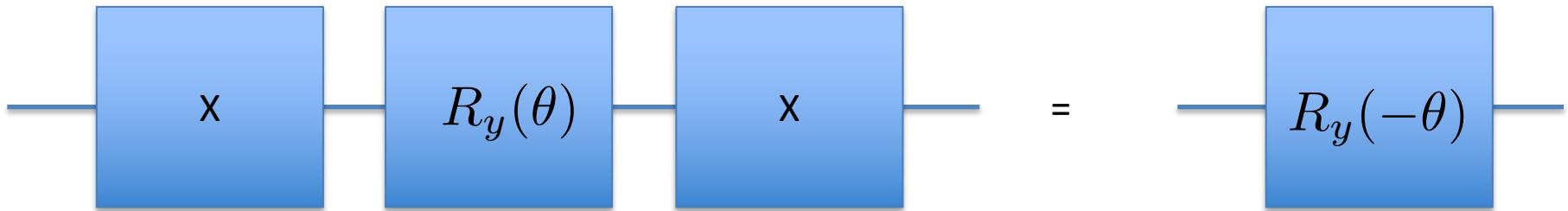
Commuting through Hadamard



Control-Z from CNOT



Conjugation with Pauli

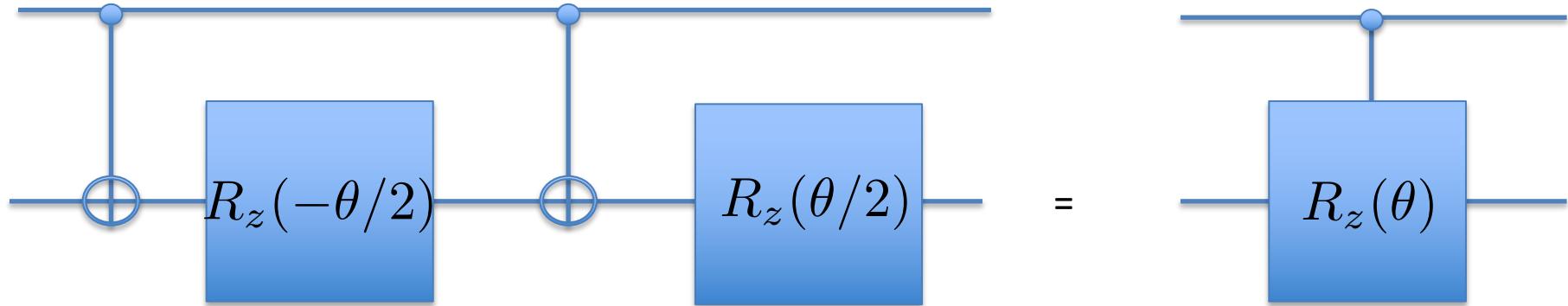


$$\begin{aligned} X R_y(\theta) X &= X \left(\cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Y \right) X \\ &= \cos \frac{\theta}{2} XX - i \sin \frac{\theta}{2} XYX \\ &= \cos \frac{\theta}{2} I + i \sin \frac{\theta}{2} Y \\ &= R_y(-\theta) \end{aligned}$$

Paulis **anticommute**
 $XY = -YX$

Works with any two orthogonal axes.

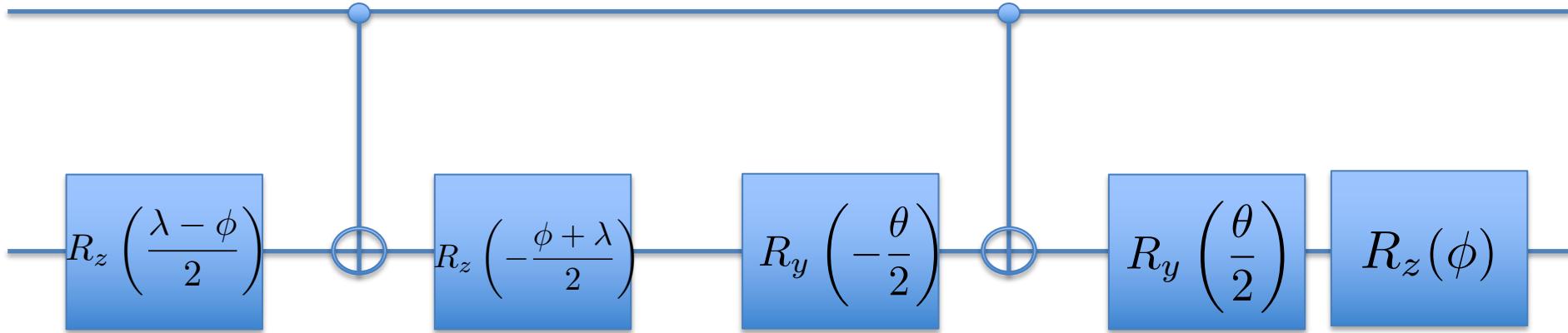
Controlled Angle Rotation



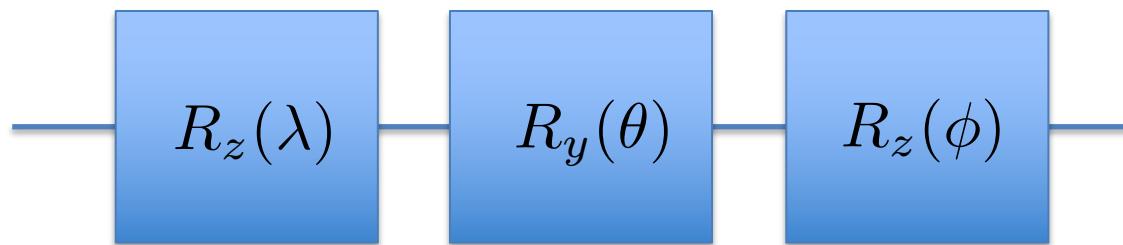
If the control is a zero, the rotations cancel.
If the control is one, the rotations add.

Any Controlled U

For U_3 Euler angle rotation (on IBM's system):

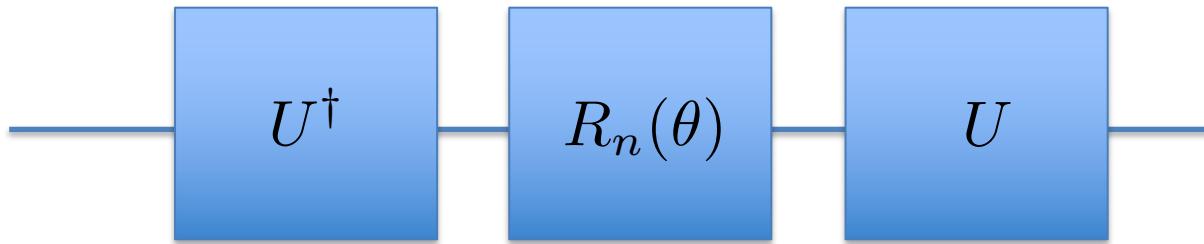


Controlled version of a U3 gate:



Conjugation with Rotation

Conjugation with a rotation:



Changes the axis of rotation, but not the rotation angle.

This rotates the axis itself

$$\begin{aligned} S R_x(\theta) S^\dagger &= \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right) S X S^\dagger \\ &= \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right) Y \\ &= R_y(\theta) \end{aligned}$$

A blue arrow points from the text "This rotates the axis itself" to the term $S X S^\dagger$ in the first line of the equation.

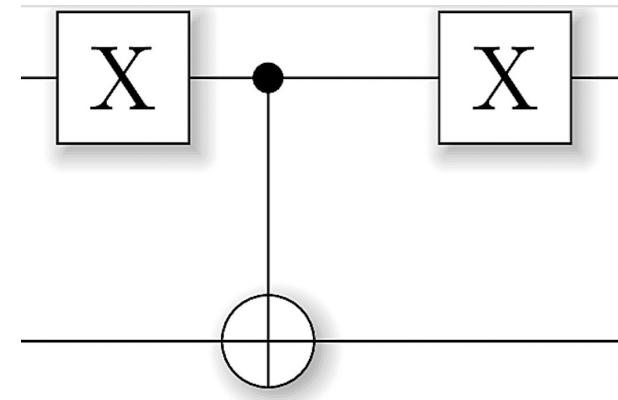
Conjugation with Hadamard is a special case of this.

Control from “0” state

Open circle =
Only apply
when the
control is “0”

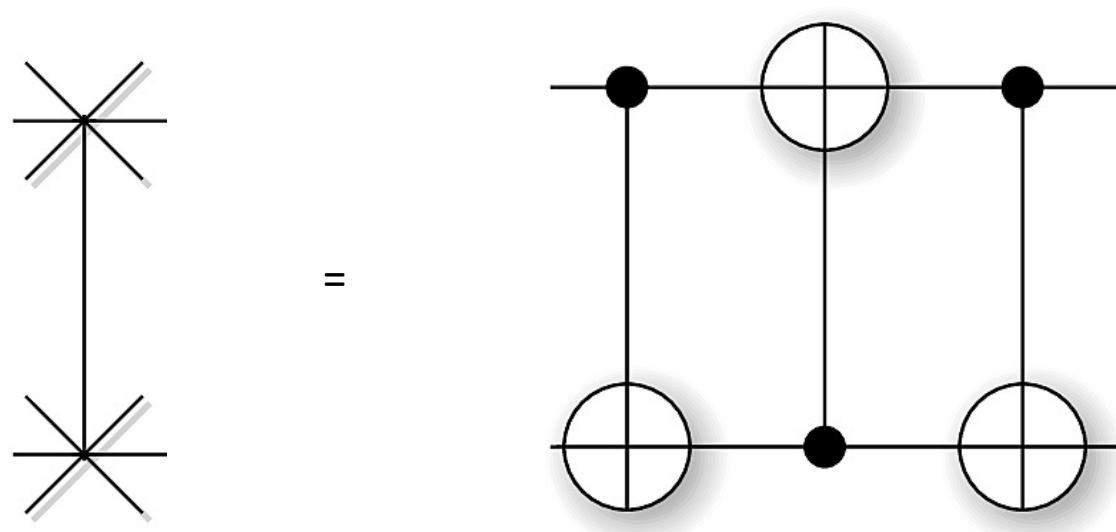


=



We've seen this trick in labs: for example in the oracle for Grover's algorithm.

Swap gate from three CNOTs



Let's check:

$$00 \rightarrow 00$$

$$01 \rightarrow 10$$

$$10 \rightarrow 01$$

$$11 \rightarrow 11$$

Square root of SWAP

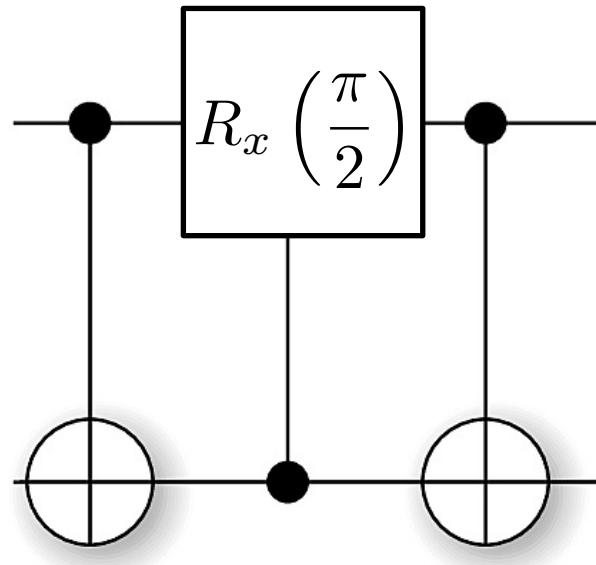
Swap

$$U_{Swap} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Square root of swap

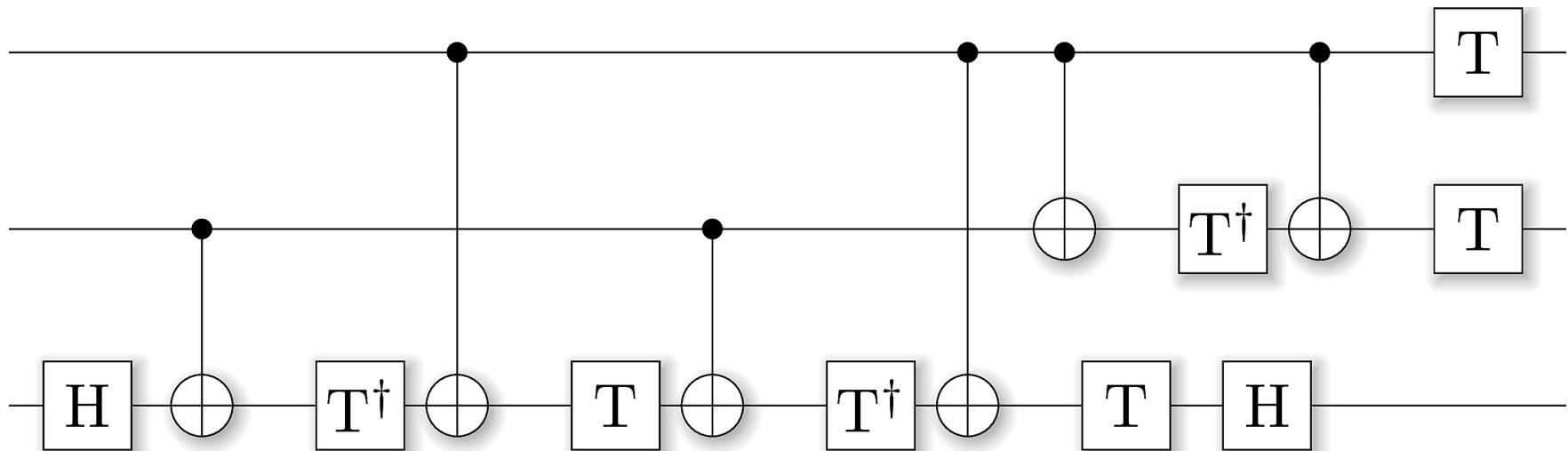
$$U_{SS} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1+i}{2} & \frac{1-i}{2} & 0 \\ 0 & \frac{1-i}{2} & \frac{1+i}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Square Root Swap Construction

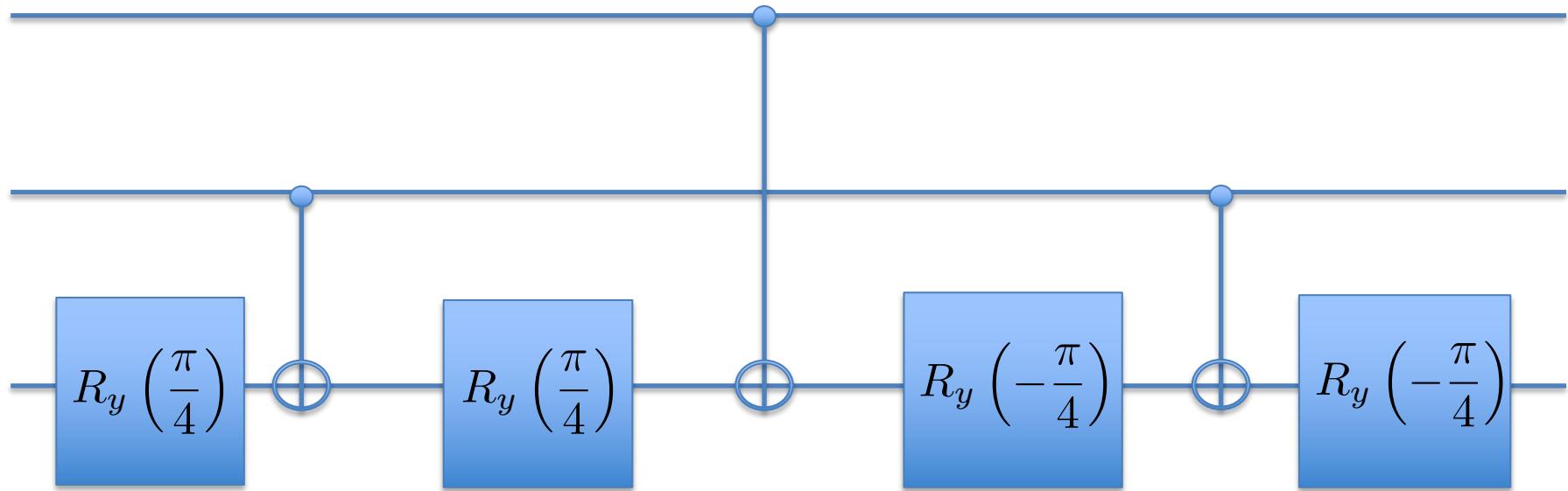


Similar to swap
Global phase of $\pi/4$

Toffoli from CNOTS



Toffoli with Incorrect Phase



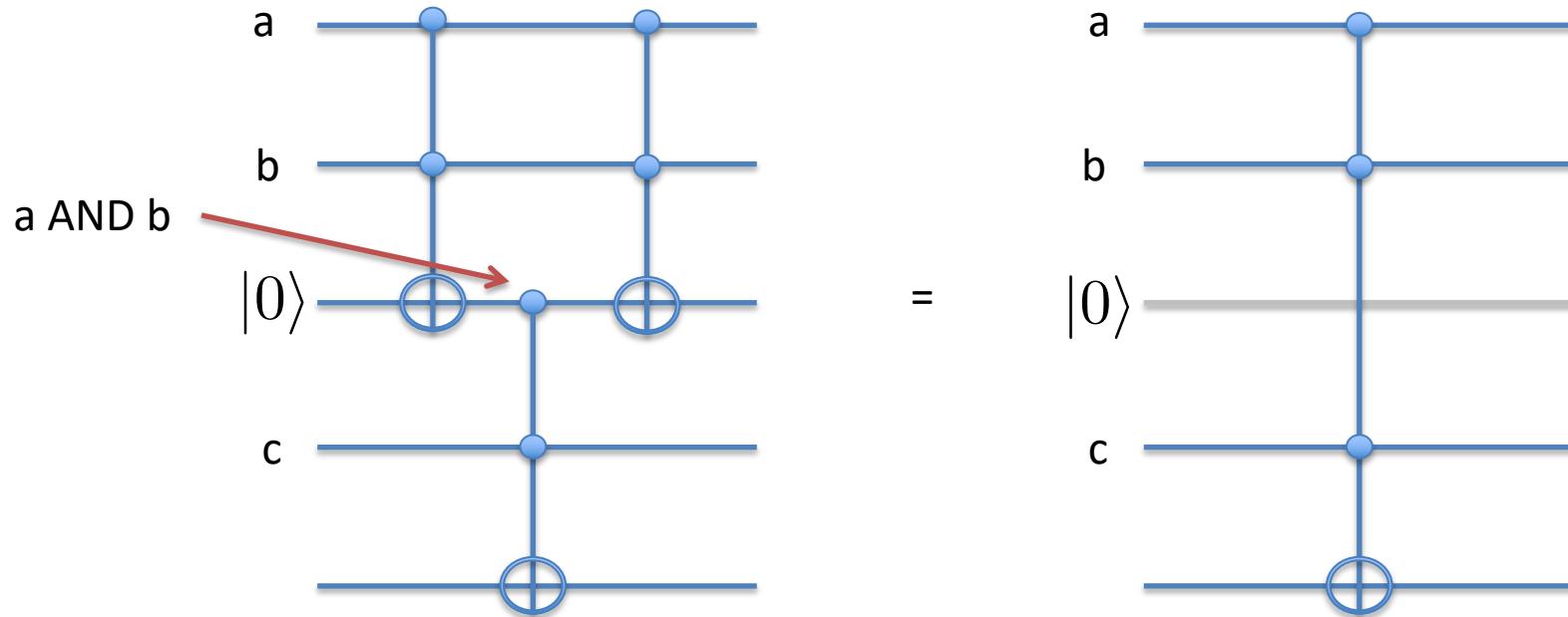
If neither control is 1, then no net rotation.

If only first control is a 1, then becomes Z rotation (Ry by pi).

If only second control qubit is a 1, then both halves cancel.

If both controls are 1, then bottom qubit flips because of middle CNOT.

Multiply Controlled Gates



We can use Toffoli and an ancilla qubit to define multiply controlled gates.

Mocking up gates

If you use controlled operations on all qubits except the target, then you isolate a single 2x2 subspace, with the rest of the matrix untouched.

$$CU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

Or controlled off the zero state:

$$C_0U = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

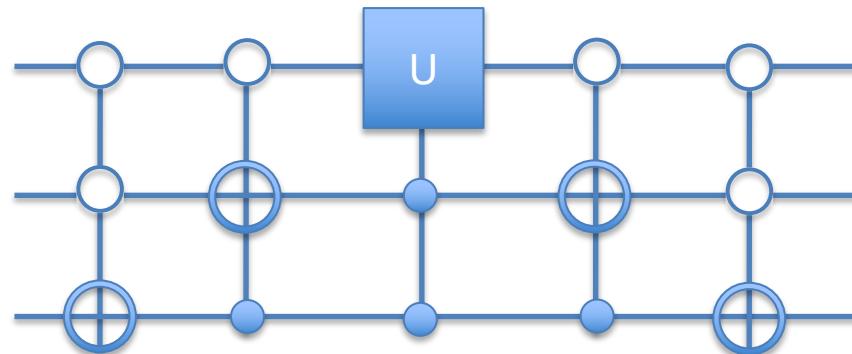
Using Gray codes

Imagine we wanted a 2x2 matrix between the 000 and 111 states:

A	B	C
0	0	0
0	0	1
0	1	1
1	1	1

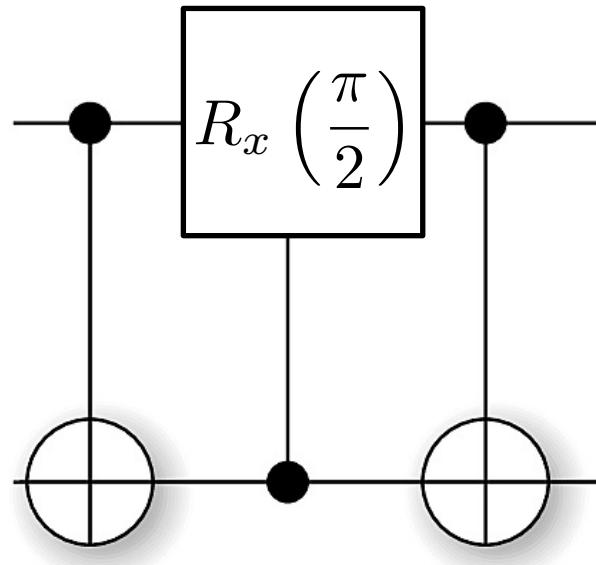
Gray code

Corresponding sequence:



In this way, complicated multi-qubit gates can be built, piece by piece.

Square Root Swap Construction



Similar to swap
Global phase of $\pi/4$

Week 9

Lecture 17 – Introduction to IBM Quantum Experience

Introduction to IBM Quantum Experience: Guest Lecture

Lecture 18 – IBM and Optimisations

14.1 QUI compared to IBM

14.2 QASM and QISKit

14.3 Optimizing circuits

Lab 9

Using the IBM system

Defining a new Function/Gate

Keyword: gate

```
// Controlled Phase gate
gate cz a,b
{
    h b;
    cx a,b;
    h b;
}
```

“cz” is name of gate

a and b are parameters

This gate can then be used like a native gate:

```
cz q[3],q[2];
```

QASM Header File

```

// Quantum Experience (QE) Standard Header
// file: qelib1.inc

// --- QE Hardware primitives ---

// 3-parameter 2-pulse single qubit gate
gate u3(theta,phi,lambda) q { U(theta,phi,lambda) q; }
// 2-parameter 1-pulse single qubit gate
gate u2(phi,lambda) q { U(pi/2,phi,lambda) q; }
// 1-parameter 0-pulse single qubit gate
gate u1(lambda) q { U(0,0,lambda) q; }
// controlled-NOT
gate cx c,t { CX c,t; }
// idle gate (identity)
gate id a { U(0,0,0) a; }

// --- QE Standard Gates ---

// Pauli gate: bit-flip
gate x a { u3(pi,0,pi) a; }
// Pauli gate: bit and phase flip
gate y a { u3(pi,pi/2,pi/2) a; }
// Pauli gate: phase flip
gate z a { u1(pi) a; }
// Clifford gate: Hadamard
gate h a { u2(0,pi) a; }
// Clifford gate: sqrt(Z) phase gate

```

Also defines:

Rotations

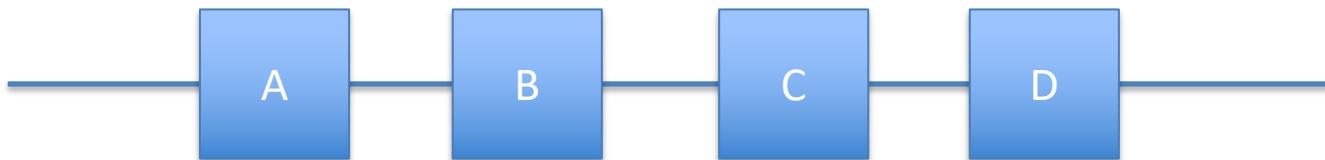
rx, ry, rz

Toffoli

CCX

Controlled rotations
cu1, cu2, cu3, crz, ch

Product of single qubit unitaries



Euler angle rotations (with global phase = 0):

$$U_3 = \begin{bmatrix} e^{-i(\lambda+\phi)/2} \cos(\theta/2) & -e^{i(\lambda-\phi)/2} \sin(\theta/2) \\ e^{i(-\lambda+\phi)/2} \sin(\theta/2) & e^{i(\lambda+\phi)/2} \cos(\theta/2) \end{bmatrix}$$

Write out the matrix and equate elements.