Quantum Computing with Qiskit

Qubits, Quantum Gate and Measurement

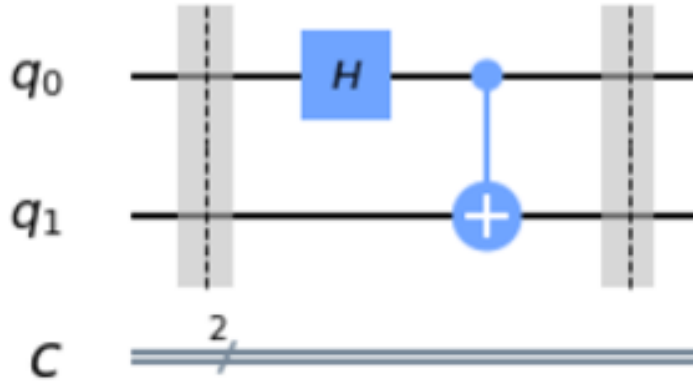
- A quantum computation is a collection of three elements
  ① A quantum register or a set of quantum register.
  ② A unitary matrix, which is used to execute a given quantum algorithm.
  ③ Measurement to extract information we need.

- Quantum circuit model
  - Universal quantum computer

Quantum Properties

- Superposition

The linear combination of two or more state vectors is another state vector in the same Hilbert space and describes another state of the system.

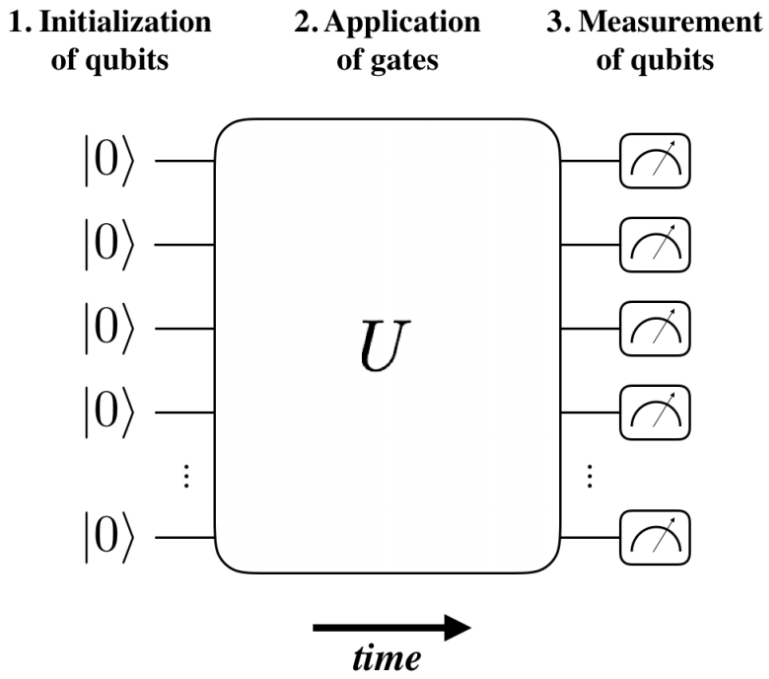


- Entanglement

Two systems are in a special case of quantum mechanical superposition called *entanglement* if the measurement of one system is correlated with the state of the other system in a way that is stronger than correlations in the classical world.

Source: Hidary,Quantum Computing: An Applied Approach

Quantum Circuit

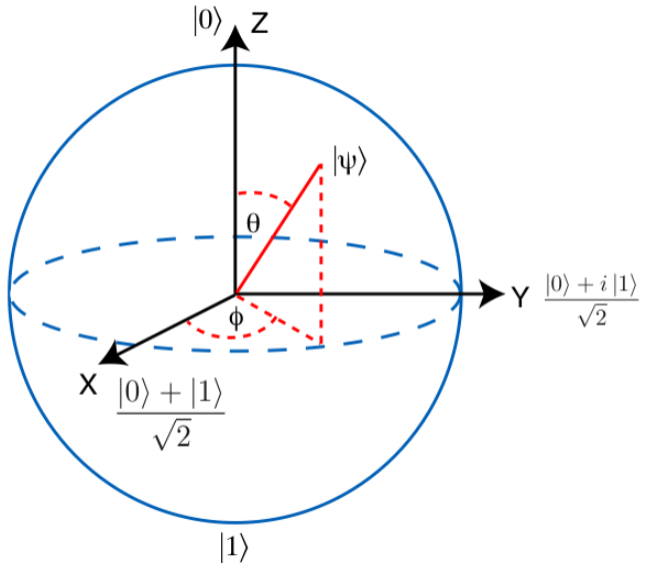


Source: https://qiskit.org/textbook/ch-states/introduction.html

Quantum Gates

- Quantum Operators
  - In gate-based quantum computers, these operator used to evolve the state.
    - Unitary and reversable.
    - Single qubit gate: rotation in block sphere.

- There exist universal quantum gates set.

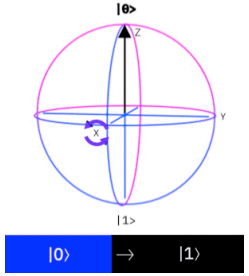


Group Theory Here.

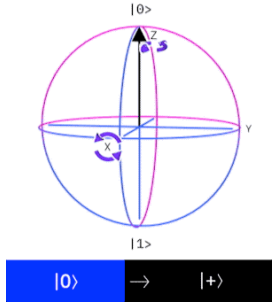
Reference: https://quantum-computing.ibm.com/docs/circ-comp/q-gates

Single Qubit Gates

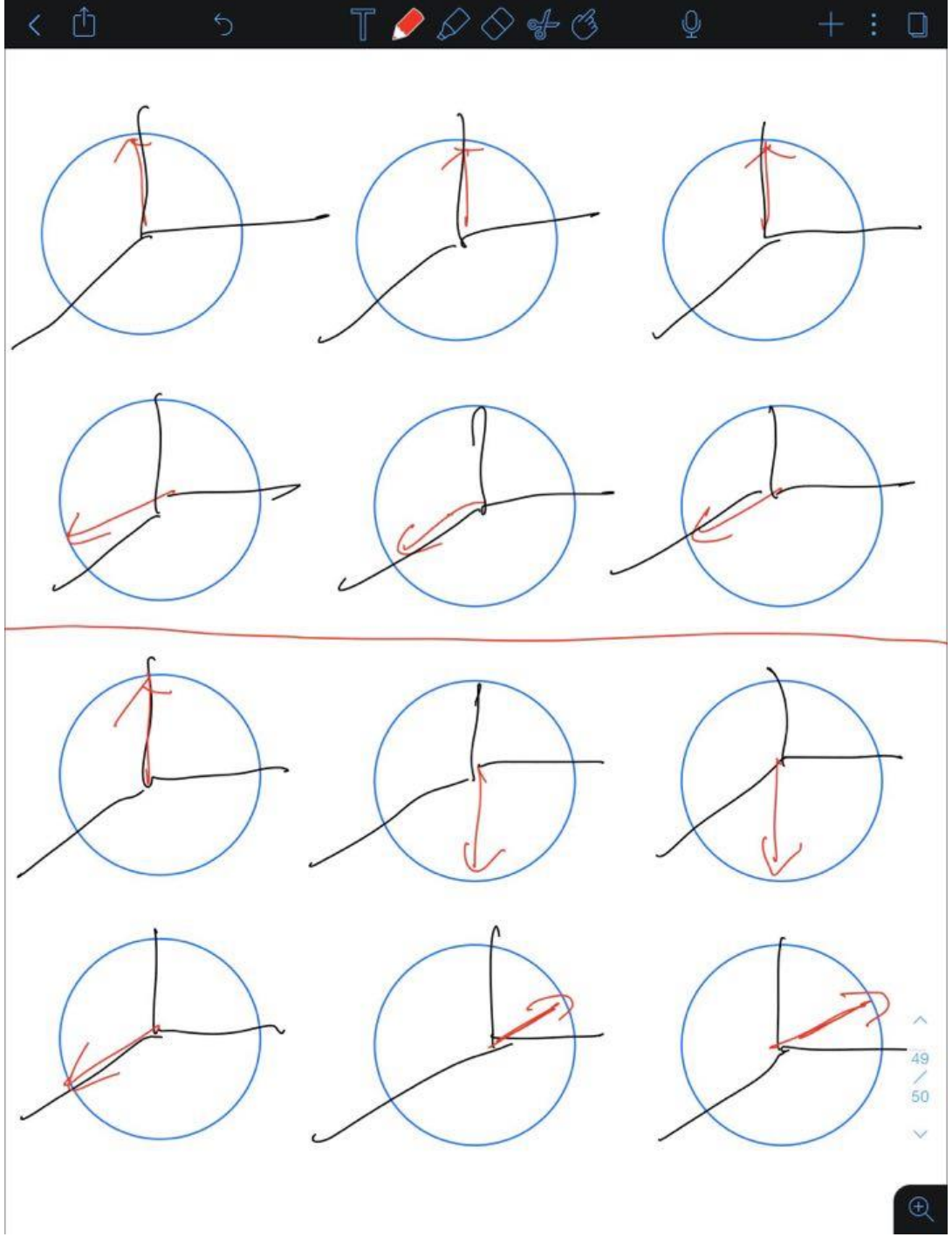
$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \quad Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$



$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{bmatrix}$$



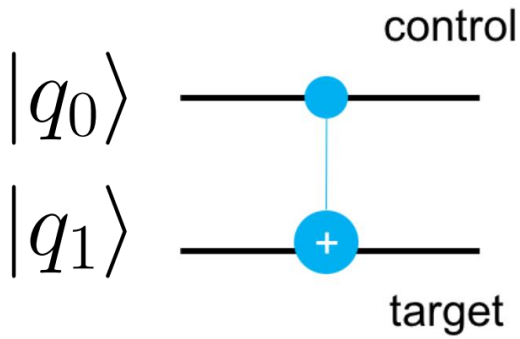
$$u3(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda + i\phi}\cos(\theta/2) \end{pmatrix}. \qquad u2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{pmatrix}. \qquad u1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}.$$


49
50

Quantum Gates in Qiskit

- Very Important!
  - The LSB (Least Significant Bit) is from right to left in qiskit.

$$|q_{n-1}, \cdots, q_1, q_0\rangle = |q_{n-1}\rangle \otimes |q_{n-2}\rangle \otimes \cdots |q_1\rangle \otimes |q_0\rangle$$



Starting state		Ending State
|00>	→	|00>
|10>	→	|10>
|01>	→	|11>
|11>	→	|01>

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

$$|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X$$

$$q_0 \otimes q_1$$

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

$$X \otimes |1\rangle\langle 1| + I \otimes |0\rangle\langle 0|$$

$$q_1 \otimes q_0$$

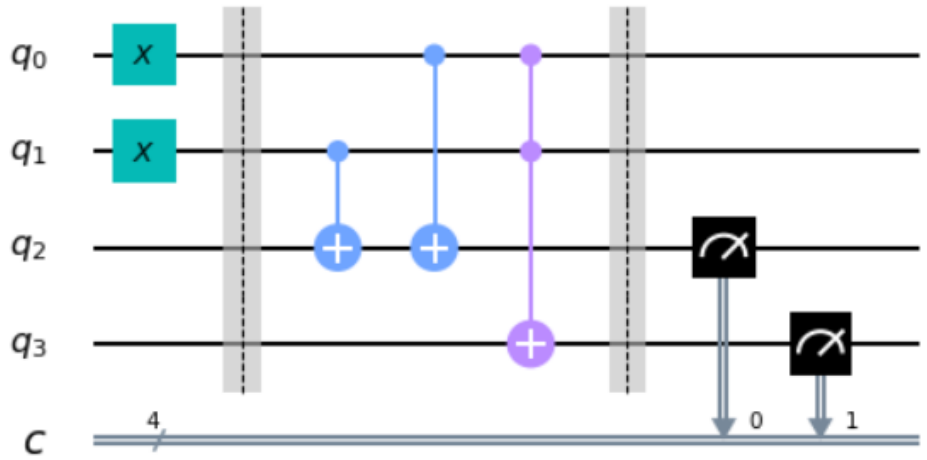
→ Also, this show the difference between entangle and separate state

Reference: https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html#qiskit.circuit.QuantumCircuit

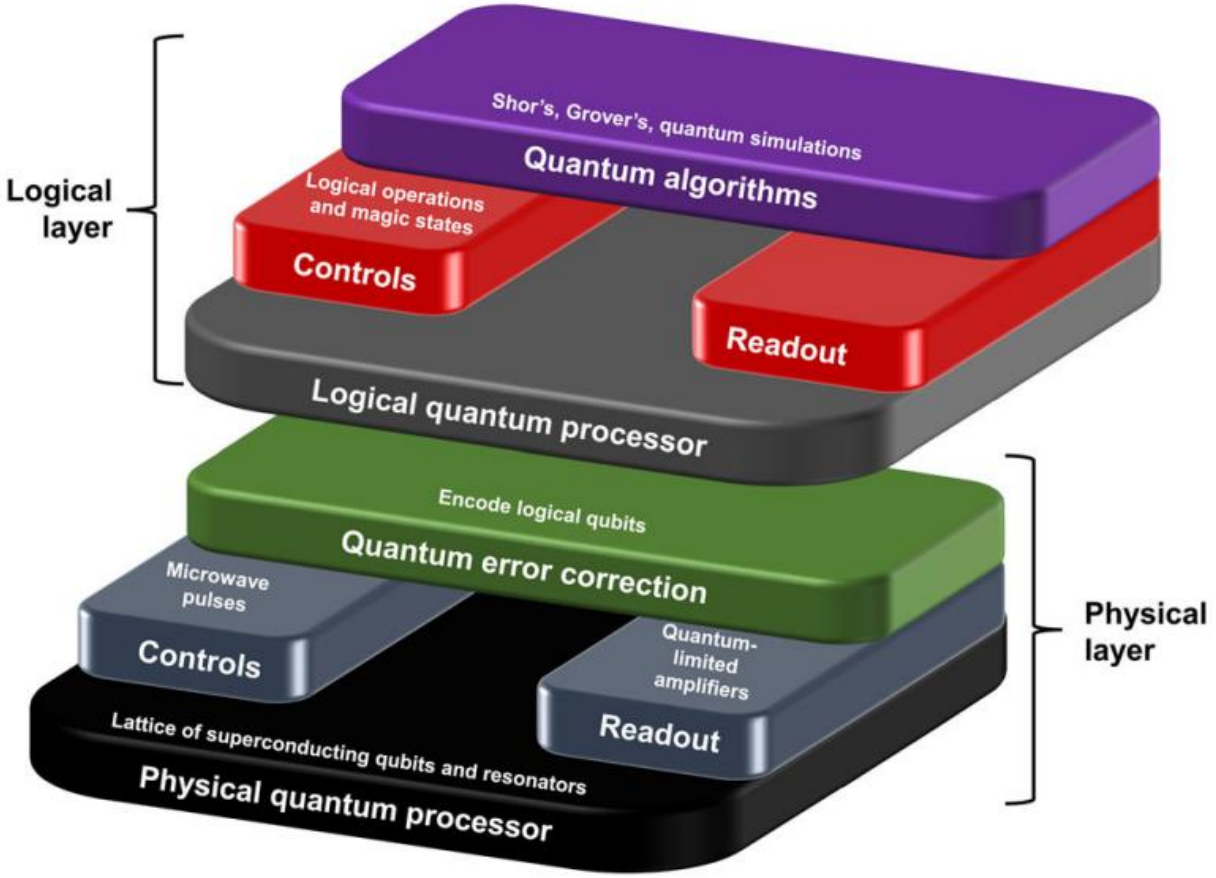
Qiskit Circuit Example

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

# Create quantum circuit with 4 qubits and 4 classical bits
q_circ = QuantumCircuit(4, 4)
q_circ.x([0,1])      # x gate on q0 & q1
q_circ.barrier()
q_circ.cx(1,2)       # control not gate  (control,target)
q_circ.cx(0,2)
q_circ.ccx(0,1,3)    # control control not gate (control1,control2,target)
q_circ.barrier()
q_circ.measure([2, 3], [0, 1])      # measurement    (target_qubitlist,classical_bitlist)

q_circ.draw(output='mpl',plot_barriers=True)  # draw circuit  you can also use: print(circuit_name)
```

Out[1]:



Quantum Computing Stack



Source: Gambetta, J.M., Chow, J.M. & Steffen, M. Building logical qubits in a superconducting quantum computing system. *npj Quantum Inf* **3,** 2 (2017).

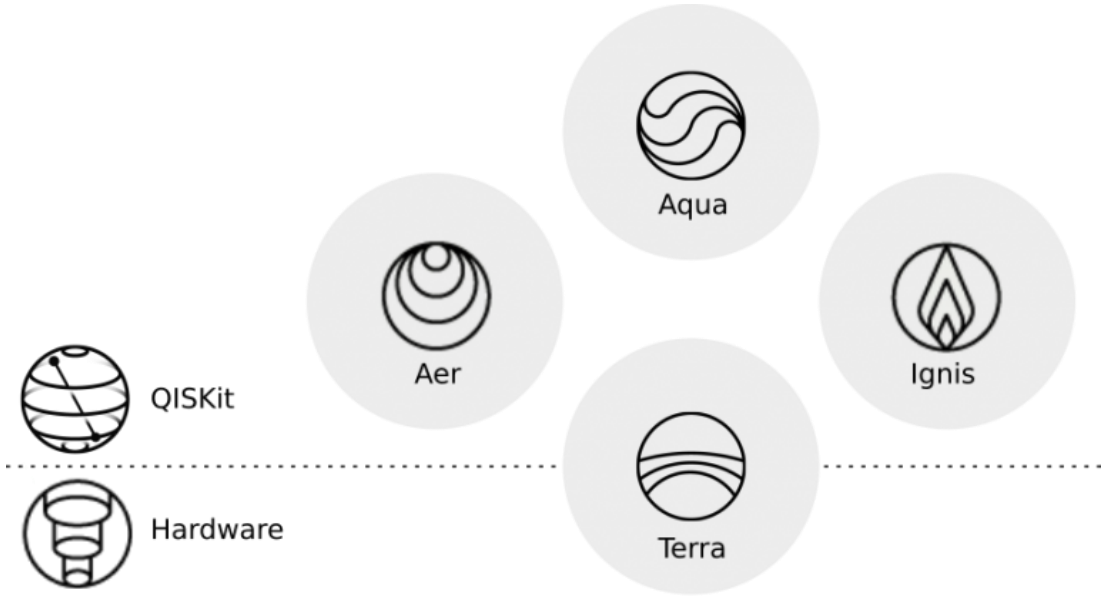
Qiskit Overview

Institution	IBM
First Release	0.1 on March 7, 2017
Open Source	Yes
License	Apache-2.0
HomePage	https://qiskit.org/
Github	https://github.com/Qiskit
Documentation	https://qiskit.org/documentation/
OS	Mac, Windows, Linux
Language	Python
Quantum Language	OpenQASM

Version Information

Qiskit Software	Version
Qiskit	0.17.0
Terra	0.12.0
Aer	0.4.1
Ignis	0.2.0
Aqua	0.6.5
IBM Q Provider	0.6.0

The Qiskit Elements



Terra, the ‘earth’ element, is the foundation on which the rest of the software lies.

Aer, the ‘air’ element, permeates all Qiskit elements. For accelerating development via simulators, emulators and debuggers

Aqua, the ‘water’ element, is the element of life. For building algorithms and applications.

Ignis, the ‘fire’ element, is dedicated to fighting noise and errors and to forging a new path

Source: https://qiskit.org/documentation/the_elements.html

Different Simulator

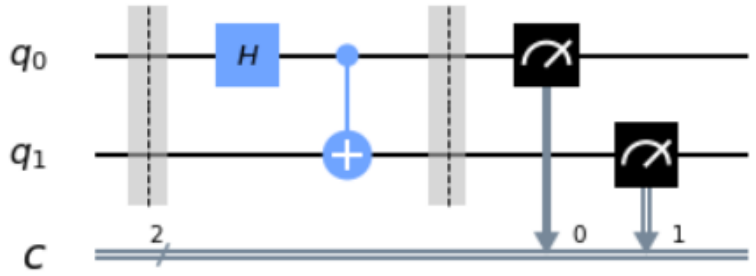
Simulator Backends

`QasmSimulator` ([configuration, provider])	Noisy quantum circuit simulator backend.
`StatevectorSimulator` ([configuration, provider])	Ideal quantum circuit statevector simulator
`UnitarySimulator` ([configuration, provider])	Ideal quantum circuit unitary simulator.
`PulseSimulator` ([configuration, provider])	Pulse schedule simulator backend.

Qiskit Code Example

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

q_bell = QuantumCircuit(2, 2)
q_bell.barrier()
q_bell.h(0)
q_bell.cx(0, 1)
q_bell.barrier()
q_bell.measure([0, 1], [0, 1])


q_bell.draw(output='mpl',plot_barriers=True)
```

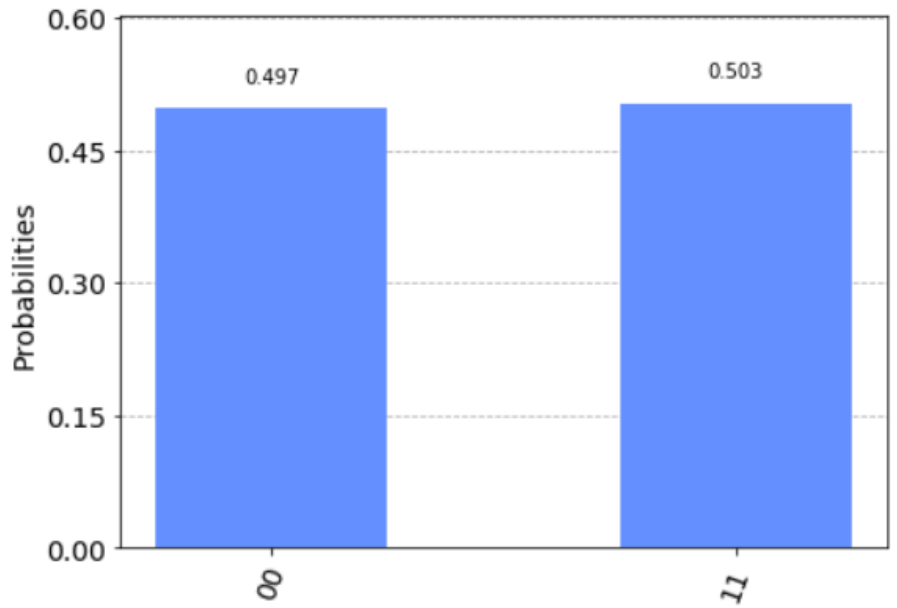
Out[1]:



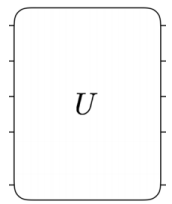
In [2]:
```
from qiskit import Aer, execute
from qiskit.visualization import plot_histogram
backend = Aer.get_backend('qasm_simulator')
job_sim = execute(q_bell, backend,shots=100000)
sim_result = job_sim.result()

print(sim_result.get_counts(q_bell))
plot_histogram(sim_result.get_counts(q_bell))
```

```
{'11': 50254, '00': 49746}
```

Out[2]:

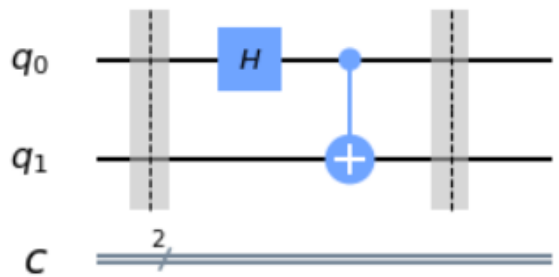


```python
from qiskit import QuantumCircuit

q_bell = QuantumCircuit(2, 2)
q_bell.barrier()
q_bell.h(0)
q_bell.cx(0, 1)
q_bell.barrier()

q_bell.draw(output='mpl',plot_barriers=True)
```




```python
from qiskit import Aer,execute
unitary_backend=Aer.get_backend('unitary_simulator')
result = execute(q_bell,unitary_backend).result()
print(result.get_unitary())
```

```
[[ 0.70710678+0.00000000e+00j  0.70710678-8.65956056e-17j
   0.        +0.00000000e+00j  0.        +0.00000000e+00j]
 [ 0.        +0.00000000e+00j  0.        +0.00000000e+00j
   0.70710678+0.00000000e+00j -0.70710678+8.65956056e-17j]
 [ 0.        +0.00000000e+00j  0.        +0.00000000e+00j
   0.70710678+0.00000000e+00j  0.70710678-8.65956056e-17j]
 [ 0.70710678+0.00000000e+00j -0.70710678+8.65956056e-17j
   0.        +0.00000000e+00j  0.        +0.00000000e+00j]]
```

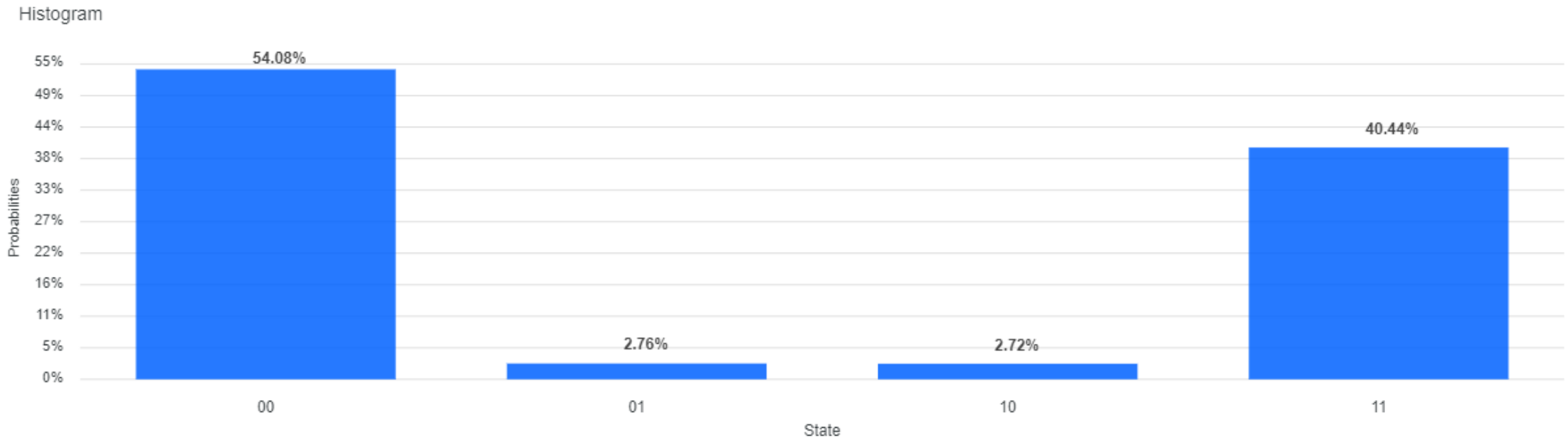


$$\begin{pmatrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{pmatrix}$$

```python
from qiskit import Aer,execute
statevector_backend= Aer.get_backend('statevector_simulator')
result = execute(q_bell,statevector_backend).result()
print(result.get_statevector())
```

```
[0.70710678+0.j 0.        +0.j 0.        +0.j 0.70710678+0.j]
```

Real QC


Histogram
Probabilities
55%
49%
44%
38%
33%
27%
22%
16%
11%
5%
0%
54.08%
2.76%
2.72%
40.44%
00
01
10
11
State

Installation

- Anaconda (highly recommended for learning)
  - pip install qiskit
- IBM online notebook
- This is CS course.

Reference: https://qiskit.org/documentation/install.html#install

The Deutsch Algorithm

- The first to demonstrate quantum over classical computing.

Deutsch Problem:

Given a black box that implement some Boolean function $f: \{0,1\} \rightarrow \{0,1\}$.
We are promised that the function is either constant or balanced.

x	f_0	f_1	f_x	$f_{\bar{x}}$
0	0	1	0	1
1	0	1	1	0

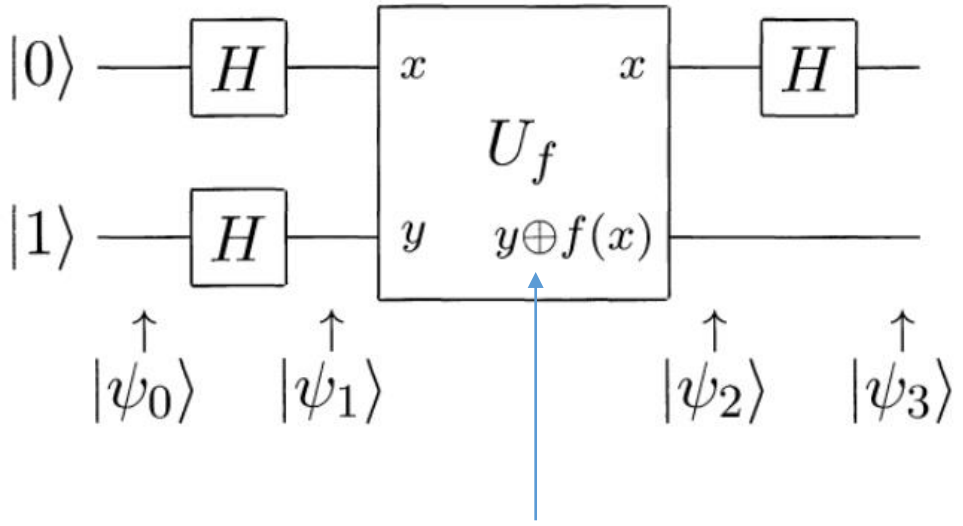
constant balanced

```
if f(0) = 0:
    if f(1) = 0:
        print("Constant")
    else:
        print("Balanced")
else:
    if f(1) = 0:
        print("Balanced")
    else:
        print("Constant")
```

require two function evaluations to figure out the answer.

The Deutsch Algorithm

- We need only one query on a quantum computer !



$$|\psi_0\rangle = |0\rangle \otimes |1\rangle$$

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle)$$

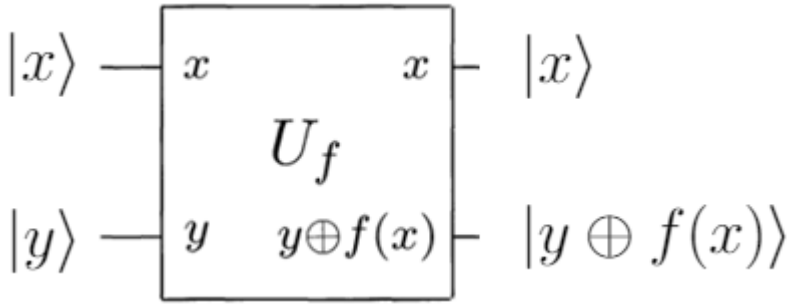
f is Boolean function.

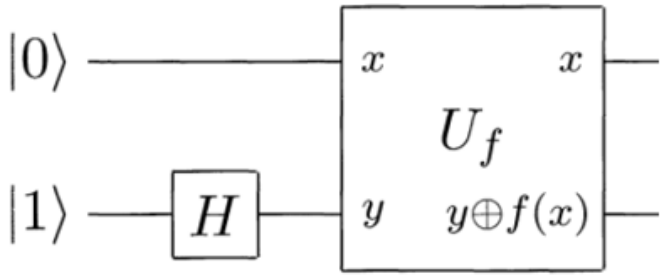
$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(x)\rangle - \left|\overline{f(x)}\right\rangle)$$

Phase Kick Back

- Useful trick in many quantum algorithm.

Consider Quantum Black box function *f*





$$U_f\left(|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) = |x\rangle \otimes \frac{1}{\sqrt{2}}(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle)$$

$$f(x) = 0: \quad \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

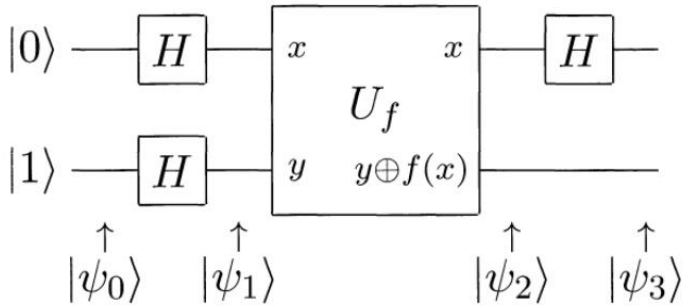
$$f(x) = 1: \quad \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} = \frac{|1\rangle - |0\rangle}{\sqrt{2}} = -\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

$$\frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} = (-1)^{f(x)}\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

$$U_f\left(|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) = (-1)^{f(x)}\left(|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)$$

$$U_f\left((\alpha_0|0\rangle + \alpha_1|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) = \left((-1)^{f(0)}\alpha_0|0\rangle + (-1)^{f(1)}\alpha_1|1\rangle\right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The Deutsch Algorithm



$$|\psi_0\rangle = |0\rangle \otimes |1\rangle$$

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|f(x)\rangle - |\overline{f(x)}\rangle)$$

$$U_f\left((\alpha_0|0\rangle + \alpha_1|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) = \left((-1)^{f(0)}\alpha_0|0\rangle + (-1)^{f(1)}\alpha_1|1\rangle\right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$\begin{aligned} |\psi_2\rangle &= \frac{(-1)^{f(0)}}{\sqrt{2}}|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{(-1)^{f(1)}}{\sqrt{2}}|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\ &= \left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\ &= (-1)^{f(0)}\left(\frac{|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \end{aligned}$$

$$|\psi_2\rangle = \begin{cases} \pm\left[\dfrac{|0\rangle + |1\rangle}{\sqrt{2}}\right]\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right]\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) \neq f(1). \end{cases}$$

$$|\psi_3\rangle = \begin{cases} \pm|0\rangle\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm|1\rangle\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) \neq f(1). \end{cases}$$

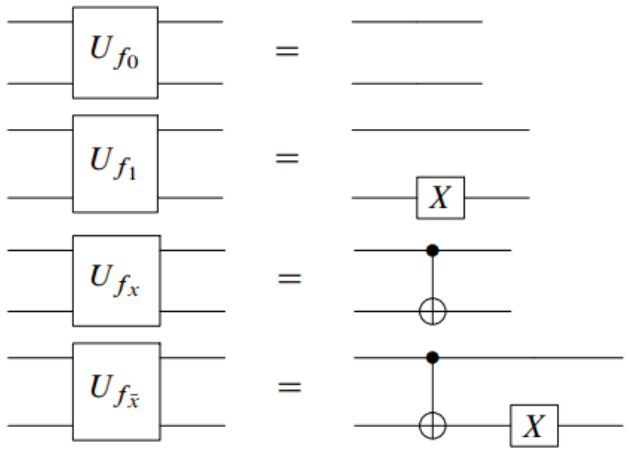
Qiskit Example

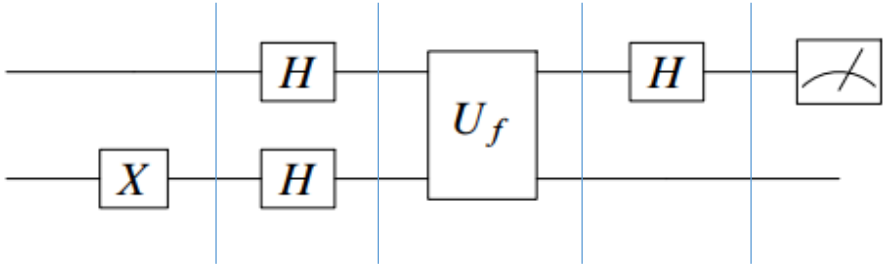
x	f_0	f_1	f_x	$f_{\bar{x}}$
0	0	1	0	1
1	0	1	1	0

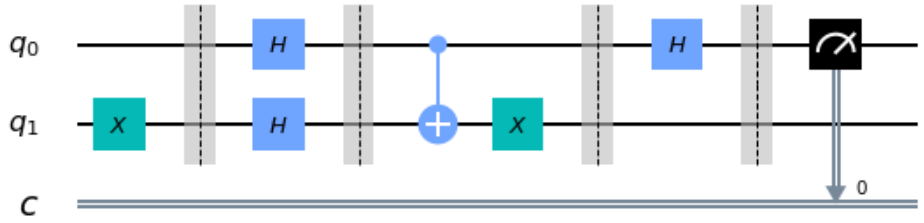
constant
output = 0

balanced
output = 1

$f_0 \rightarrow I$
$f_1 \rightarrow -I$
$f_x \rightarrow Z$
$f_{\bar{x}} \rightarrow -Z$

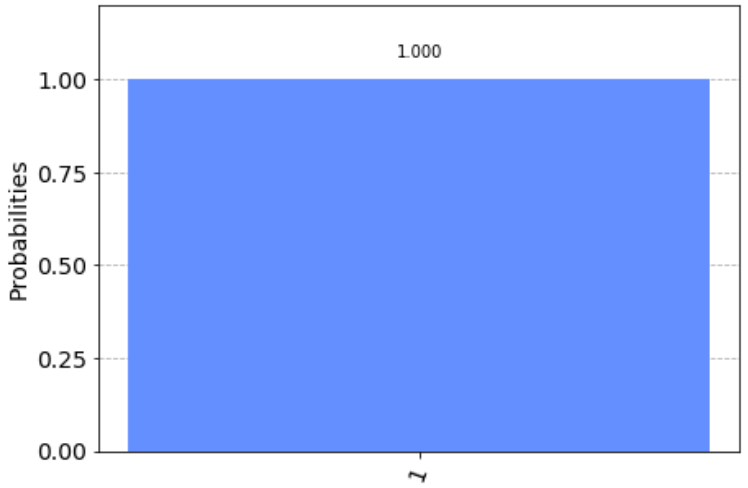







```
from qiskit import QuantumCircuit
q_demo=QuantumCircuit(2,1)

q_demo.x(1)
q_demo.barrier()    # psi0
q_demo.h([0,1])
q_demo.barrier()    # psi1
q_demo.cx(0,1)
q_demo.x(1)
q_demo.barrier()    # psi2
q_demo.h(0)
q_demo.barrier()    # psi3
q_demo.measure([0],[0])

q_demo.draw(output='mpl')      # draw your circuit

from qiskit import Aer, execute
from qiskit.visualization import plot_histogram
backend = Aer.get_backend('qasm_simulator')
demo_circ = execute(q_demo, backend,memory=False)
result = demo_circ.result()

#memory = result.get_memory(q_demo)
print(result.get_counts(q_demo))
plot_histogram(result.get_counts(q_demo))
```

```
{'1': 1024}
```



Extension : The Deutsch-Jozsa algorithm

- This time the function f is a function from n bits string to a bit.

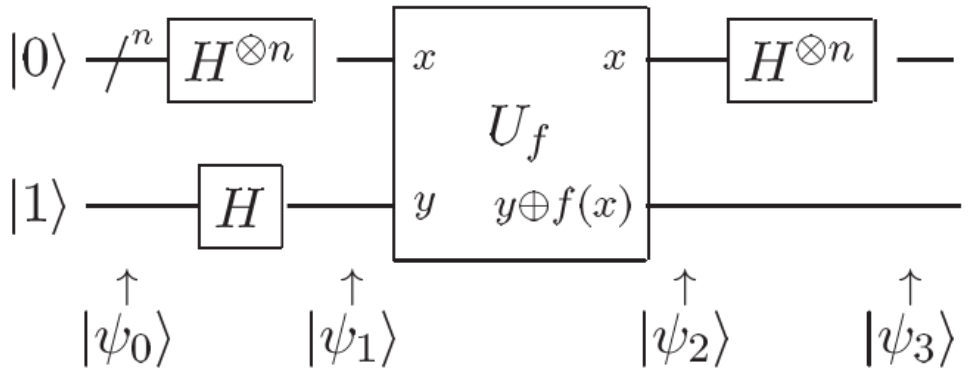
The Deutsch–Jozsa Problem

Input: A black-box for computing an unknown function $f : \{0,1\}^n \to \{0,1\}$.
Promise: f is either a constant or a balanced function.
Problem: Determine whether f is constant or balanced by making queries to f.

The Deutsch-Jozsa algorithm

The Deutsch–Jozsa Problem

Input: A black-box for computing an unknown function $f: \{0,1\}^n \to \{0,1\}$.
Promise: f is either a constant or a balanced function.
Problem: Determine whether f is constant or balanced by making queries to f.



$$|\psi_0\rangle = |0\rangle^{\otimes n} \otimes |1\rangle$$

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_y \sum_x (-1)^{f(x)+x\cdot y} |y\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

Summary

Algorithm: Deutsch–Jozsa

Inputs: (1) A black box U_f which performs the transformation $|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$, for $x \in \{0, \ldots, 2^n - 1\}$ and $f(x) \in \{0, 1\}$. It is promised that $f(x)$ is either *constant* for all values of x, or else $f(x)$ is *balanced*, that is, equal to 1 for exactly half of all the possible x, and 0 for the other half.

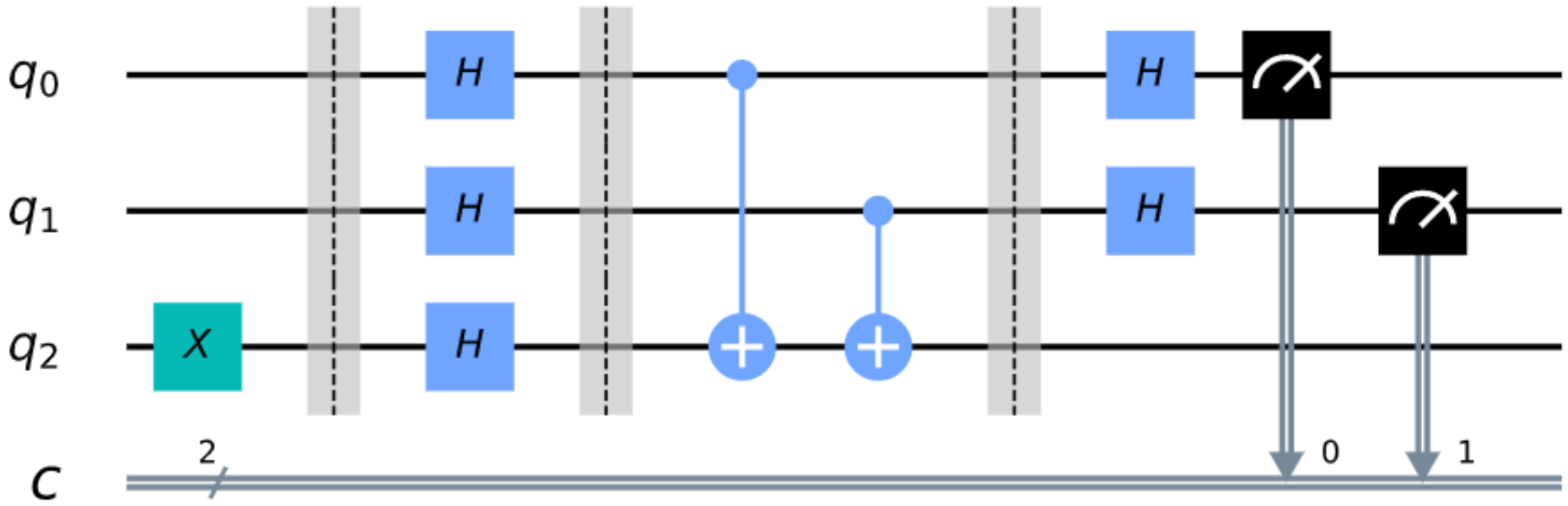
Outputs: 0 if and only if f is constant.

Runtime: One evaluation of U_f. Always succeeds.

Procedure:

1.	$\|0\rangle^{\otimes n}\|1\rangle$	initialize state
2.	$\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} \|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$	create superposition using Hadamard gates
3.	$\rightarrow \sum_x (-1)^{f(x)}\|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$	calculate function f using U_f
4.	$\rightarrow \sum_z \sum_x \frac{(-1)^{x \cdot z + f(x)}\|z\rangle}{\sqrt{2^n}} \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$	perform Hadamard transform
5.	$\rightarrow z$	measure to obtain final output z

Qiskit Example

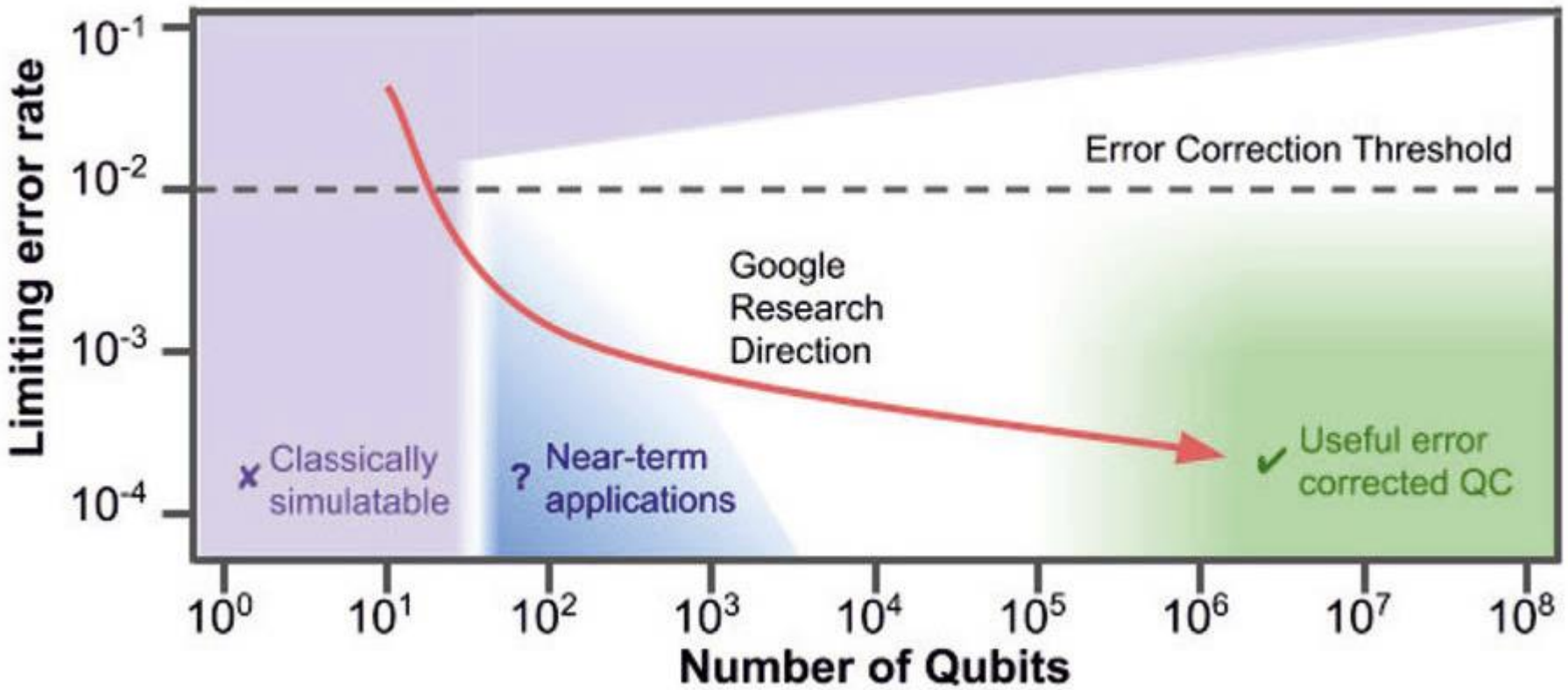


Improvement?

- The worst case in classical take us $\frac{2^n}{2} + 1$
- In quantum we need only one query ….. But, if there exist error ?

- That’s why we have Bernstein-Vazirani, Simon’s algorithm.

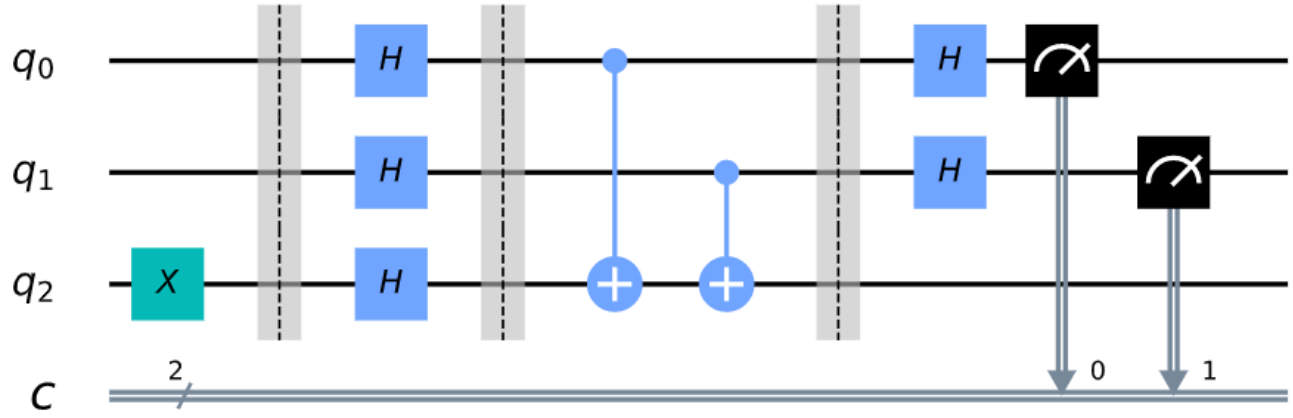
Where are we now?



NISQ Quantum Computing

- Noisy Intermediate-Scale Quantum

- Circuit depth is important.



Depth = 6 in this case

Reference

- Nielsen & Chuang : Quantum Computation and Quantum Information
- Hidary : Quantum Computing: An Applied Approach
- Qiskit Textbook : https://qiskit.org/textbook/preface.html