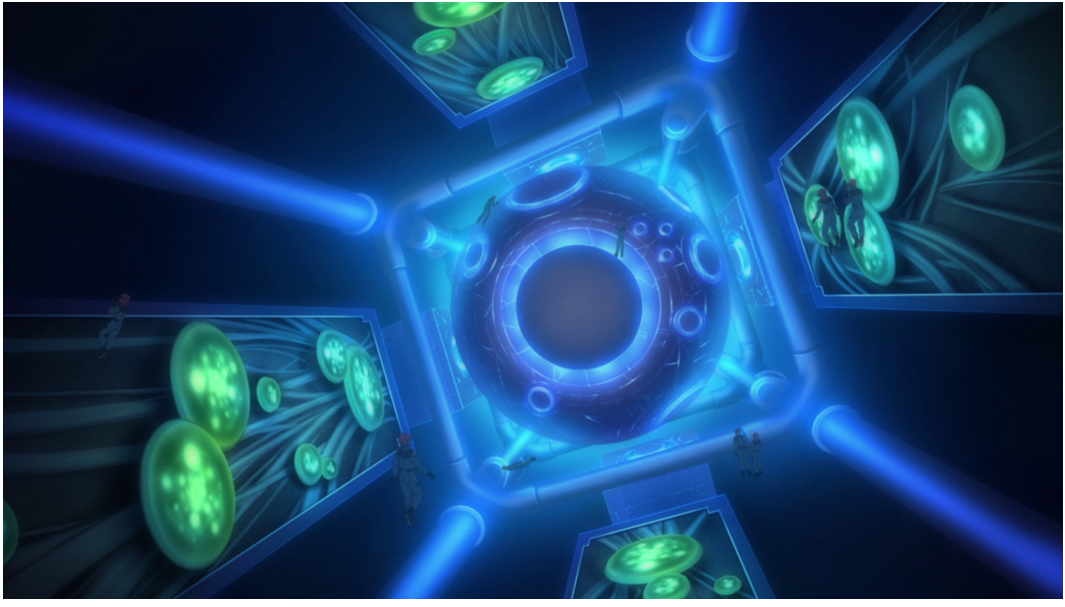
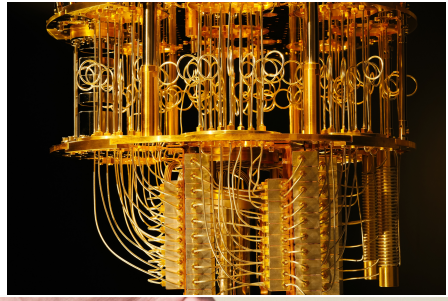
Open Source Quantum Computing

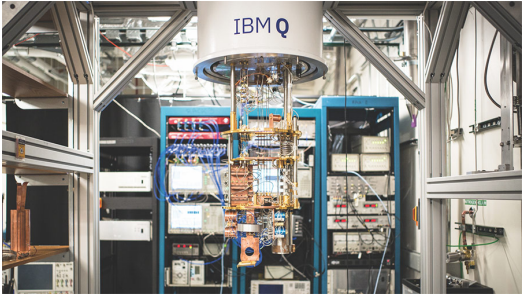
Matthew Treinish
Software Engineer - IBM Research
mtreinish@kortar.org
`mtreinish on Freenode`
https://github.com/mtreinish/open-source-quantum-computing

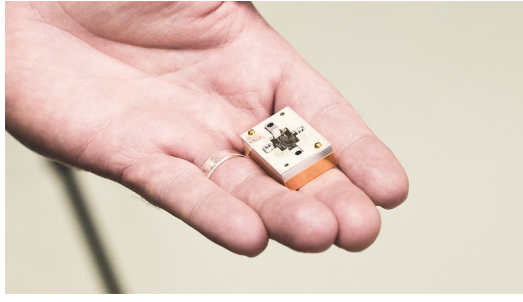
January 23, 2019



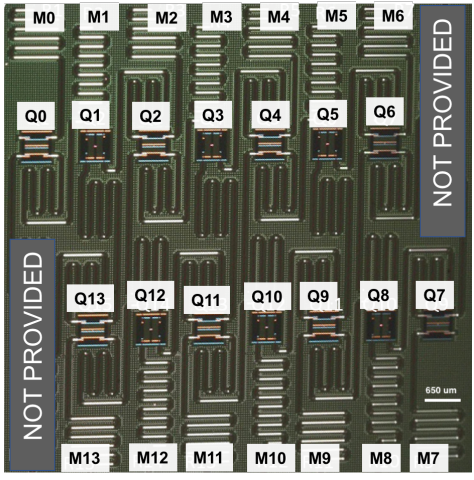
Real Quantum Computer

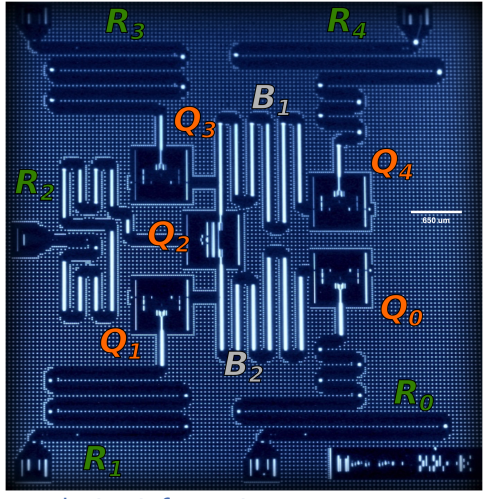






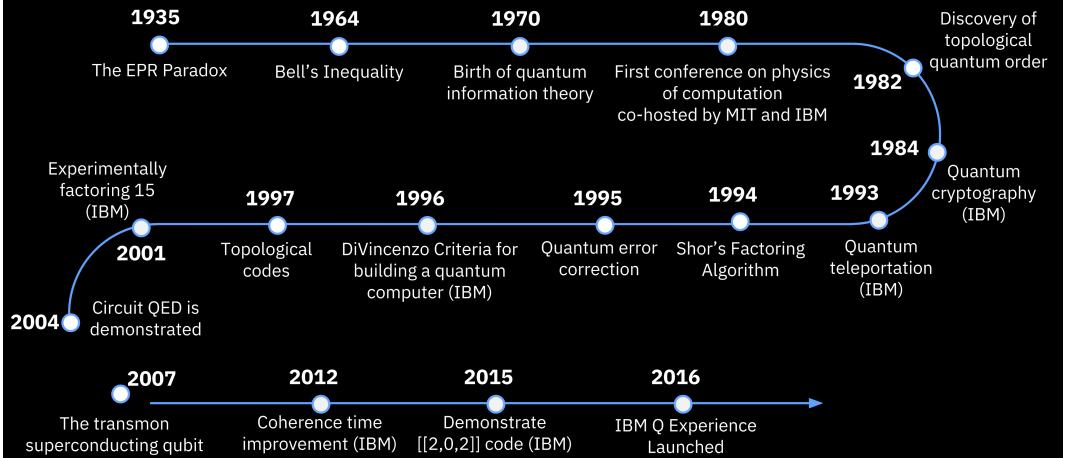
Quantum Chips





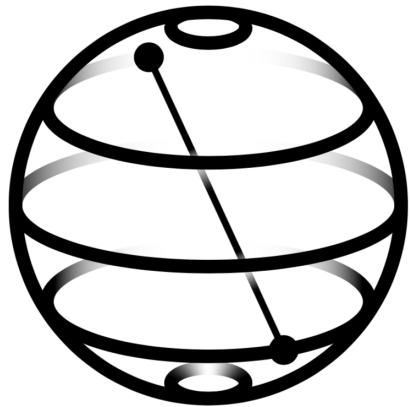
https://github.com/Qiskit/ibmq-device-information

History of Quantum Computing

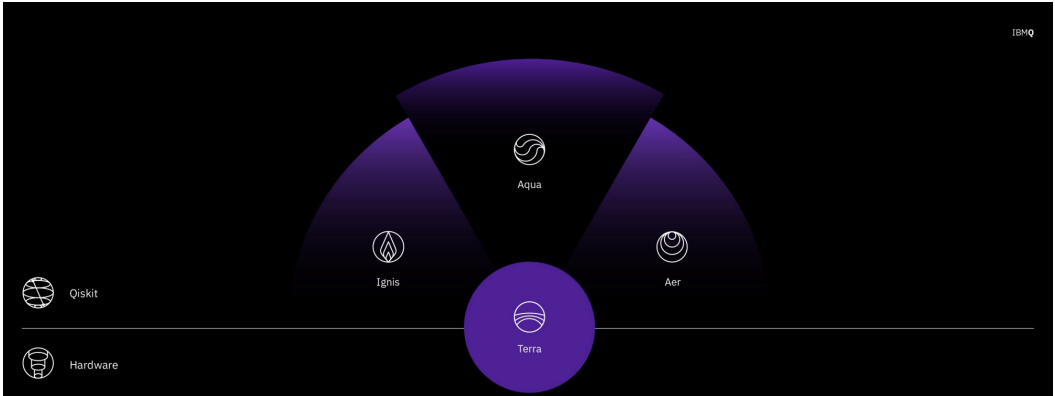


What is Qiskit?

- SDK for working with Noisy Intermediate-Scale Quantum (NISQ) computers
- Apache 2.0 License
- Designed to be backend agnostic
- Includes out-of-the-box local simulators and support for running on IBMQ



Qiskit Elements



Terra

Terra is a collection of core, foundational tools for communicating with quantum devices and simulators. Users can write quantum circuits, and address real hardware constraints with Terra. Its modular design simplifies adding extensions for quantum circuit optimizations and backends.

Ignis

Controlling fire was a turning point in human evolution. Learning how to fix or control quantum errors will be a turning point in the evolution of quantum computing. Users can access better characterization of errors, improve gates, and compute in the presence of noise with Ignis. It is designed for researching and improving errors or noise in near-term quantum systems.

Aqua

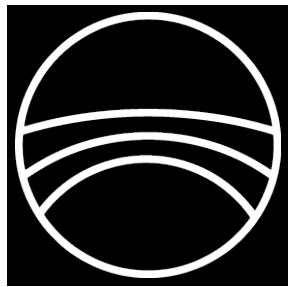
Aqua is a modular and extensible library for experimenting with quantum algorithms on near-term devices. Users can build domain-specific applications, such as chemistry, AI and optimization with Aqua. It bridges quantum and classical computers by enabling classical programming to run on quantum devices.

Aer

Aer permeates all other Qiskit elements. Users can accelerate their quantum simulator and emulator research with Aer, which helps to better understand the limits of classical processors by demonstrating their ability to mimic quantum computation. Users can also verify current and near-term quantum computer functionality with Aer.

Qiskit Terra

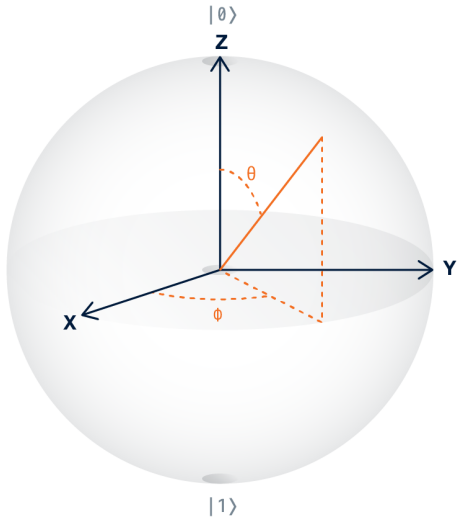
- Is the base layer for applications, provides interface to hardware and simulators
- Provides an SDK for working with quantum circuits
- Compiles circuits to run on different backends
- Written in Python



The Qubit

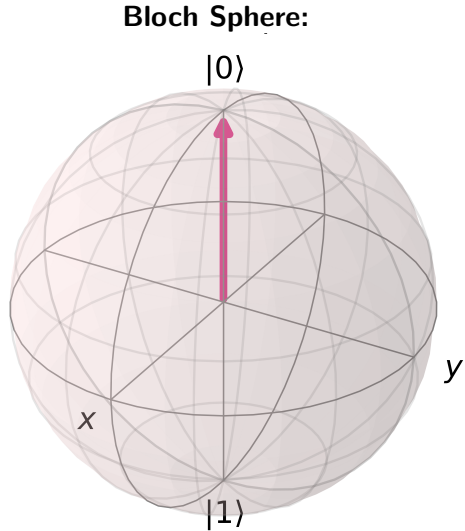
- The bloch sphere provides a representation of qubit state
- State can be at any point along surface of sphere
- Measuring a qubit occurs along the Z axis. (also called basis states)
- Measuring a qubit is irreversible and will either be 0 or 1

Bloch Sphere:



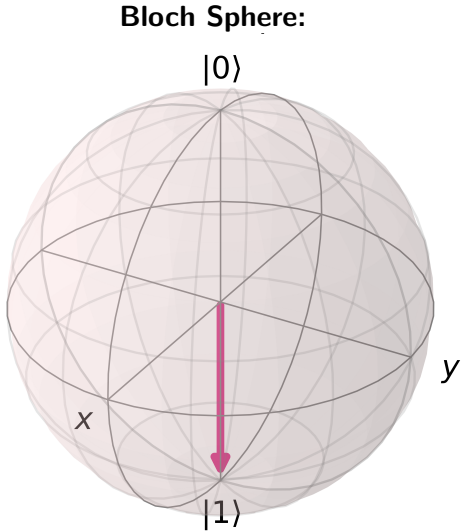
The Qubit

- The bloch sphere provides a representation of qubit state
- State can be at any point along surface of sphere
- Measuring a qubit occurs along the Z axis. (also called basis states)
- Measuring a qubit is irreversible and will either be 0 or 1



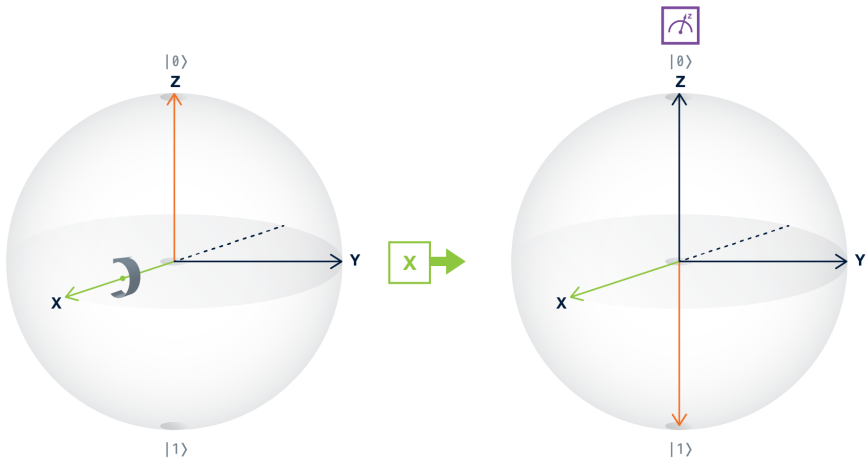
The Qubit

- The bloch sphere provides a representation of qubit state
- State can be at any point along surface of sphere
- Measuring a qubit occurs along the Z axis. (also called basis states)
- Measuring a qubit is irreversible and will either be 0 or 1



Quantum Gates

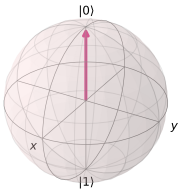
- Quantum Logic Gates are used perform operations on qubits
- Gates are reversible
- Gates can be represented as unitary matrices



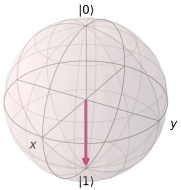
Superposition

- Identically prepared qubits can still behave randomly
- The randomness is inherent in nature

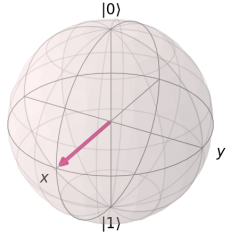
$|0\rangle$



$|1\rangle$

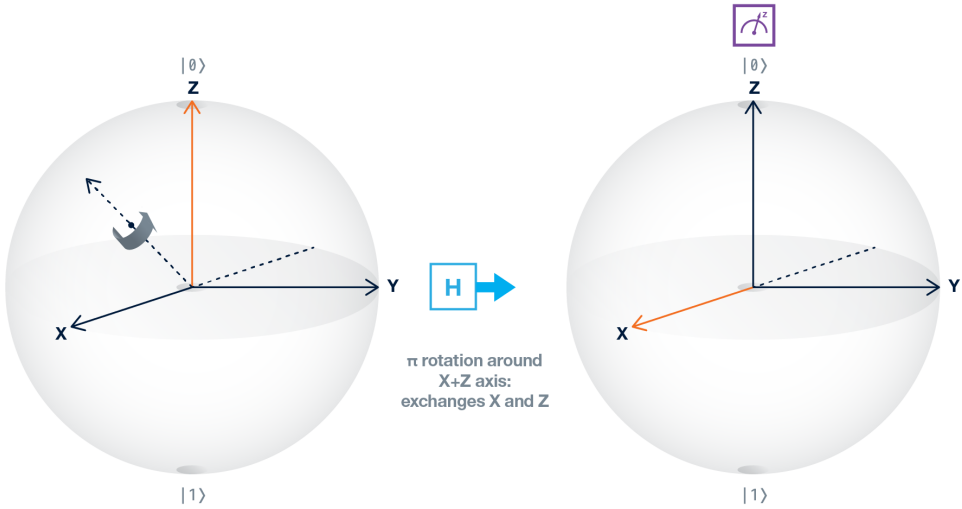


$|0\rangle + |1\rangle$



$\sim$50/50 chance of being $|0\rangle$ or $|1\rangle$

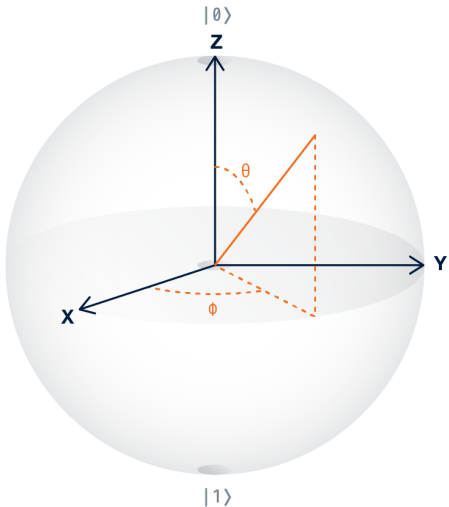
Hadamard Gate



Qubit Phase

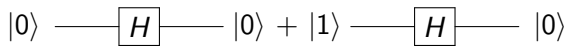
- While qubits are read along the basis vectors you can still use the other dimensions
- The phase can be leveraged to encode more information in the qubit

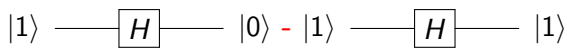
Phase is ϕ:



Hadamard and Phase

Hadamard gates are self inverses:

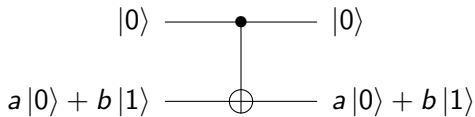


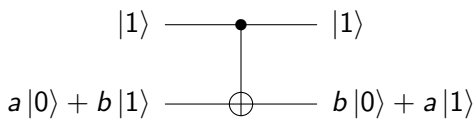


Phase

Controlled Not Gate

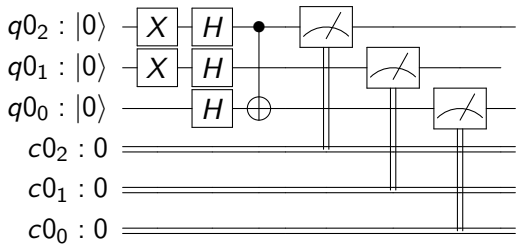
CNOT flips the *target* bit if the *control* bit is 1





Quantum Circuits

Putting it together you build a circuit like:



- Each row represents a bit, either quantum or classical
- The operations are performed each qubit left to right
- Shows dependencies of operations

Bernstein-Vazirani Algorithm[1]

The Oracle

Input (query)
$\Longleftarrow X_{n-1} \cdots X_1 X_0$

Secret Bitstring
$\boxed{S_{n-1} \cdots S_1 S_0}$

Output (result)
$\Longrightarrow X_{n-1} S_{n-1} \oplus \cdots X_1 S_1 \oplus X_0 S_0$

[1] E. Bernstein & U. Vazirani, STOC, 93

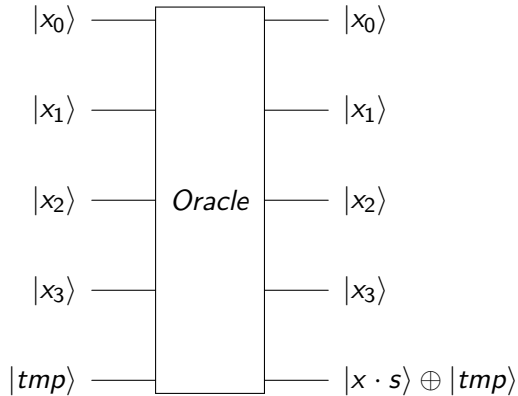
Optimal Classical Oracle

Loop over each bit!

$$\begin{cases} X = 1\,0 \cdots 0\,0 & (2^{n-1}) \\ X = 0\,1 \cdots 0\,0 & (2^{n-2}) \\ \vdots & \\ X = 0\,0 \cdots 1\,0 & (2) \\ X = 0\,0 \cdots 0\,1 & (1) \end{cases}$$

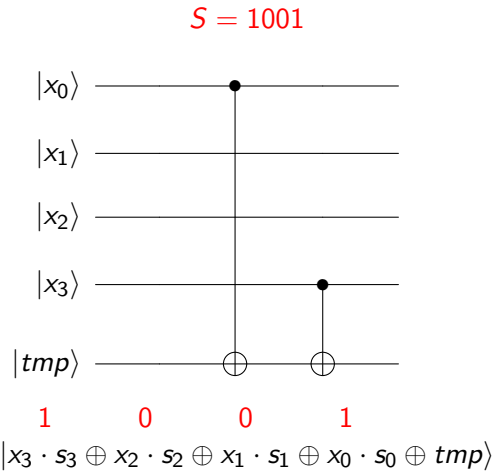
The ideal classical oracle is $\mathcal{O}(n)$

Quantum Oracle



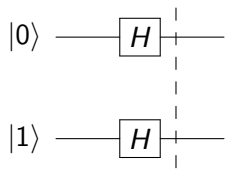
A quantum oracle is $\mathcal{O}(1)$

Quantum Oracle Implementation



Phase Kickback

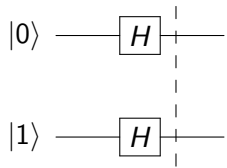
$$(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$$



Phase Kickback

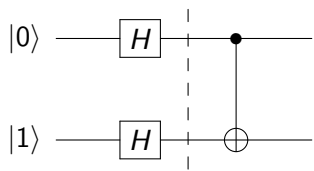
$$(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = |00\rangle - |01\rangle + |10\rangle - |11\rangle$$

Expand it



Phase Kickback

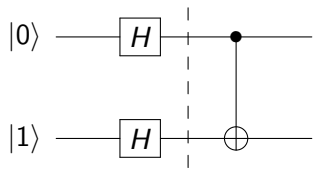
$$(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = |00\rangle - |01\rangle + |10\rangle - |11\rangle$$



$$|00\rangle - |01\rangle - |10\rangle + |11\rangle$$

Phase Kickback

$$(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = |00\rangle - |01\rangle + |10\rangle - |11\rangle$$

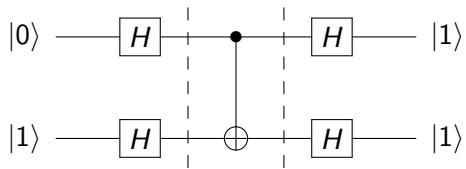


$$|00\rangle - |01\rangle - |10\rangle + |11\rangle = (|0\rangle - |1\rangle)(|0\rangle - |1\rangle)$$

Phase Kickback

Phase Kickback

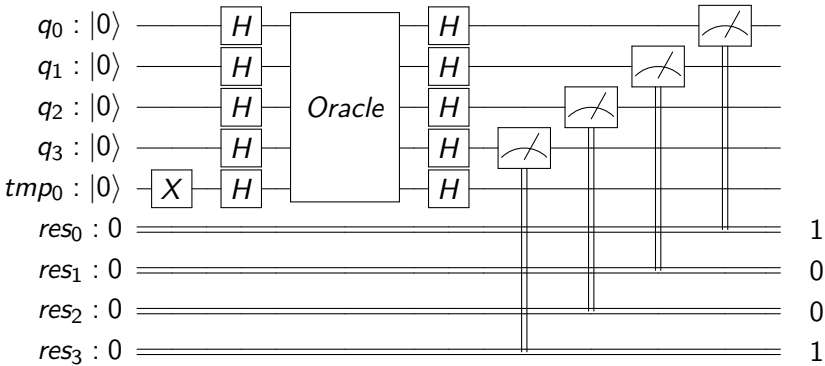
$$(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = |00\rangle - |01\rangle + |10\rangle - |11\rangle$$



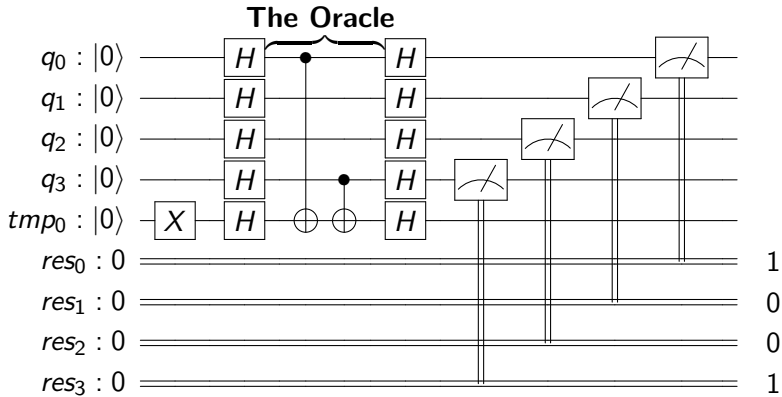
$$|00\rangle - |01\rangle - |10\rangle + |11\rangle = (|0\rangle - |1\rangle)(|0\rangle - |1\rangle)$$

Phase Kickback

Full Circuit for a Quantum Oracle



Full Circuit for a Quantum Oracle



Where there is a CNOT phase kickback will set the control qubit to state $|1\rangle$

Live Demo

Open Source in Quantum Computing

- Many of the tools for developing quantum programs are open source
- Open source is used to help fosters collaboration
- Learning from the history of development of classical computers

Other Open Source Tools

- https://github.com/rigetticomputing/pyquil
- https://github.com/ProjectQ-Framework/ProjectQ
- https://github.com/quantumlib/Cirq
- https://github.com/qutip/qutip
- https://github.com/XanaduAI/strawberryfields

A lot more out there: https://github.com/topics/quantum-computing

Conclusions

- Quantum Computing is about solving problems that we can't with classical computers
- It's still very early for quantum computers
- Not just in labs anymore, quantum computing is accessible by everyone now
- Open source software is playing a key role in the development of quantum computers

Where to get more information

- These Slides: https://github.com/mtreinish/open-source-quantum-computing
- Qiskit: https://qiskit.org/
- Qiskit Terra on Github: https://github.com/Qiskit/qiskit-terra
- IBM Q Experience: https://quantumexperience.ng.bluemix.net/qx
- Tutorials on Quantum Computing and Qiskit: https://github.com/Qiskit/qiskit-tutorials

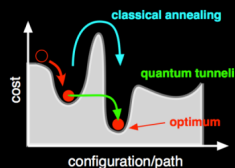
BACKUP SLIDES

Types of Quantum Computing

Quantum Annealing

Optimization Problems

- Machine learning
- Fault analysis
- Resource optimization
- etc…

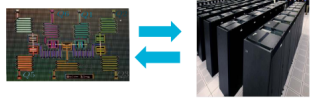


Many ‘noisy’ qubits can be built;
large problem class in optimization;
amount of quantum speedup unclear

Approximate Q-Comp.

Simulation of Quantum Systems, Optimization

- Material discovery
- Quantum chemistry
- Optimization
  (logistics, time scheduling,…)
- Machine Learning

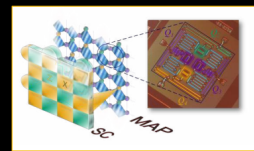


Hybrid quantum-classical approach;
already 50-100 “good” physical qubits
could provide quantum speedup.

Fault-tolerant Universal Q-Comp.

Execution of Arbitrary Quantum Algorithms

- Algebraic algorithms
  (machine learning, cryptography,…)
- Combinatorial optimization
- Digital simulation of quantum systems



Surface Code: Error correction in a Quantum Computer

Proven quantum speedup;
error correction requires significant qubit overhead.

Ivano Tavernelli - ita@zurich.ibm.com

IBM

Entanglement

- 2 qubits too far apart to influence each other can behave in a way that are **individually random** but are **strongly correlated**
- The state of an n-qubit system cannot (in general) be written as the state of its individual components.
- To simulate entanglement you need exponential resources
- CNOT gates are often used to setup entanglements

One Qubit:

$$a|0\rangle + b|1\rangle$$

Two Qubits:

$$a|00\rangle + b|01\rangle + c|11\rangle$$

Three Qubits:

$$a|000\rangle + b|001\rangle + c|010\rangle + d|011\rangle + e|100\rangle + f|101\rangle + g|110\rangle + h|111\rangle$$