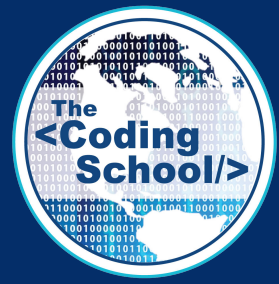
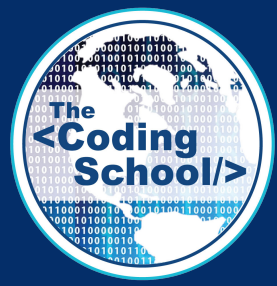




INTRO TO QUANTUM COMPUTING

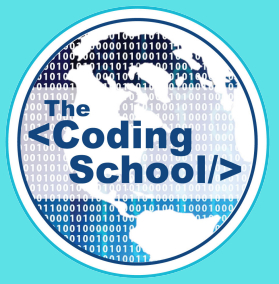
LECTURE #19

QUANTUM COMPUTATION PT. 4 :

INTRO TO QUANTUM ALGORITHMS

FRANCISCA VASCONCELOS

3/14/2021



ANNOUNCEMENTS

QUANTUM COMPUTATION LECTURE SERIES

Lecture 1 – The Quantum Circuit Model *How can we perform computation with quantum systems?*	CONCEPTS
Lecture 2 – Qiskit Tutorial *How can we program quantum circuits?*	PROGRAMMING
Lecture 3 – Quantum Circuit Mathematics *How can we represent quantum circuits mathematically?*	MATH
Lectures 4-6 – Introductory Quantum Protocols and Algorithms *How can we leverage quantum for cryptography, teleportation, and algorithms?*	APPLICATION

TODAY'S LECTURE

1. Algorithms Motivation
2. What is an Algorithm?
    a) Algorithmic Thinking
    b) Classical Algorithms
3. Characterizing Algorithmic Performance
    a) Big-O Notation
4. Quantum Algorithms Landscape
    a) Oracles
        a) Deutsch-Josza
    b) Quantum Fourier Transform
        a) Quantum Phase Estimation
        b) Shor's
    c) Amplitude Amplification
        a) Grover Search
    d) Near-Term Hybrid Algorithms
        a) Applications

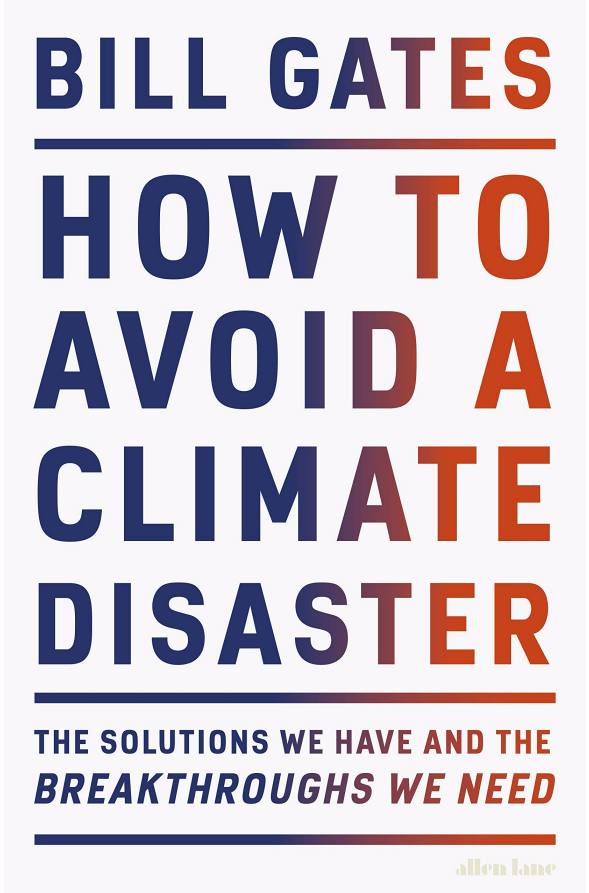
A STEP BACK...

We've spent a lot of time this semester learning about the technical details of quantum information and computation.

Why do we care so much about building quantum computers, though? What will they be useful for?

51 BILLION

How many tons of greenhouse gas the world currently adds to the atmosphere annually...



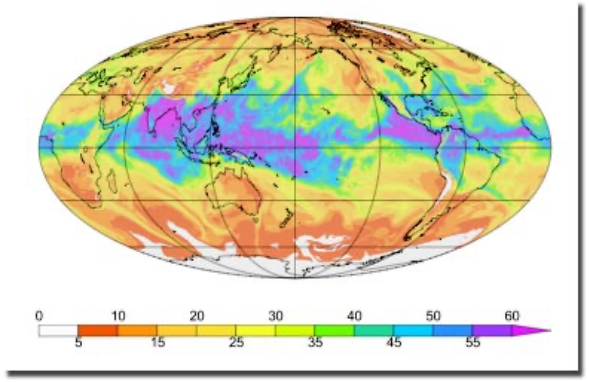
ZERO

What we need to aim for...

COMPUTATION FOR INNOVATION

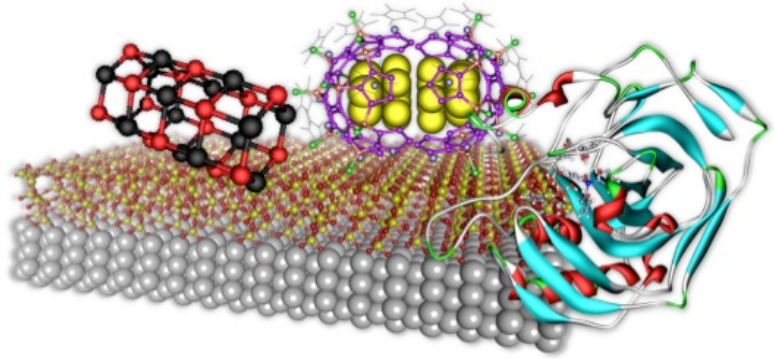
Climate change will require a lot of ***innovation***: in policy, in economics, institutionally, and in technology.

These problems are so hard that we can only feasibly solve them with the aid of ***computation***.



Modeling the atmosphere

Predicting energy usage



Solving for efficient material properties

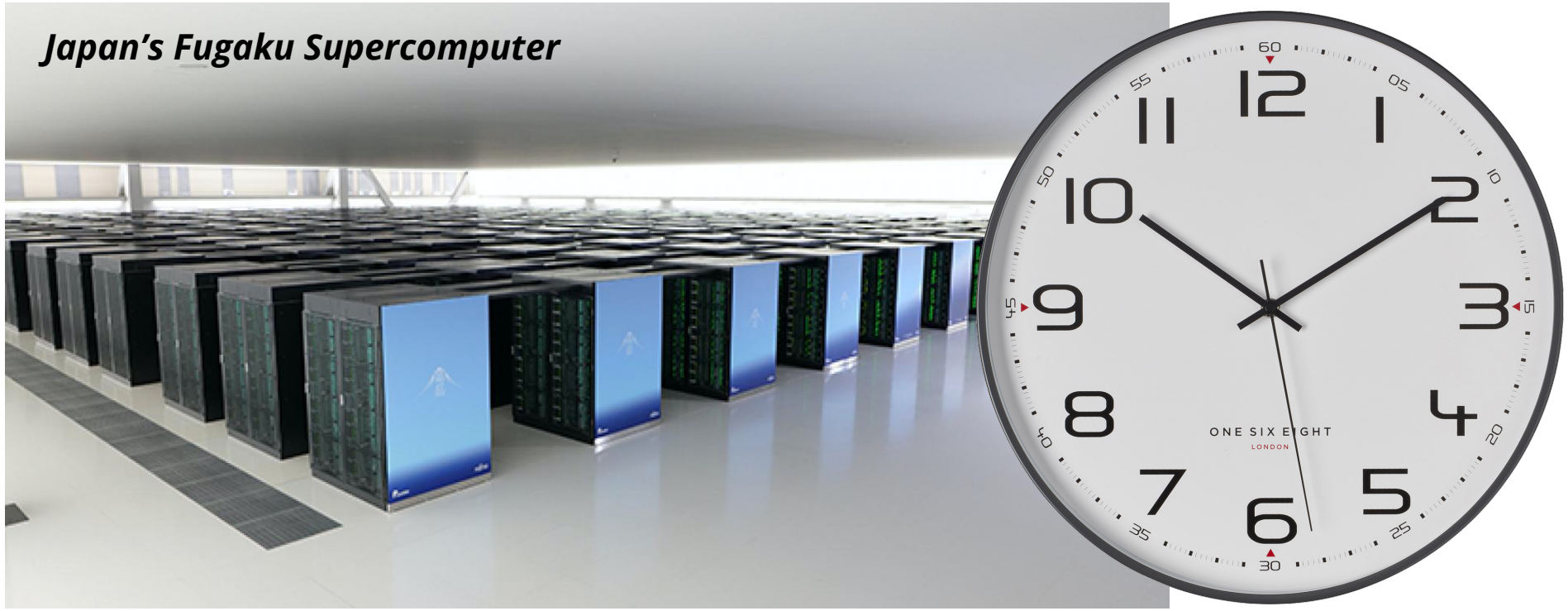
Strategic use of different renewables

Monitoring emissions

Calculating Green Premiums

COMPUTATIONAL CHALLENGES

Scientists have worked extremely hard for several decades to develop **algorithms**, which run on classical computers, to attempt to solve these types of hard problems. However, some of problems are computationally ***so hard*** that they would take the age of the universe for the world's largest supercomputer to solve...

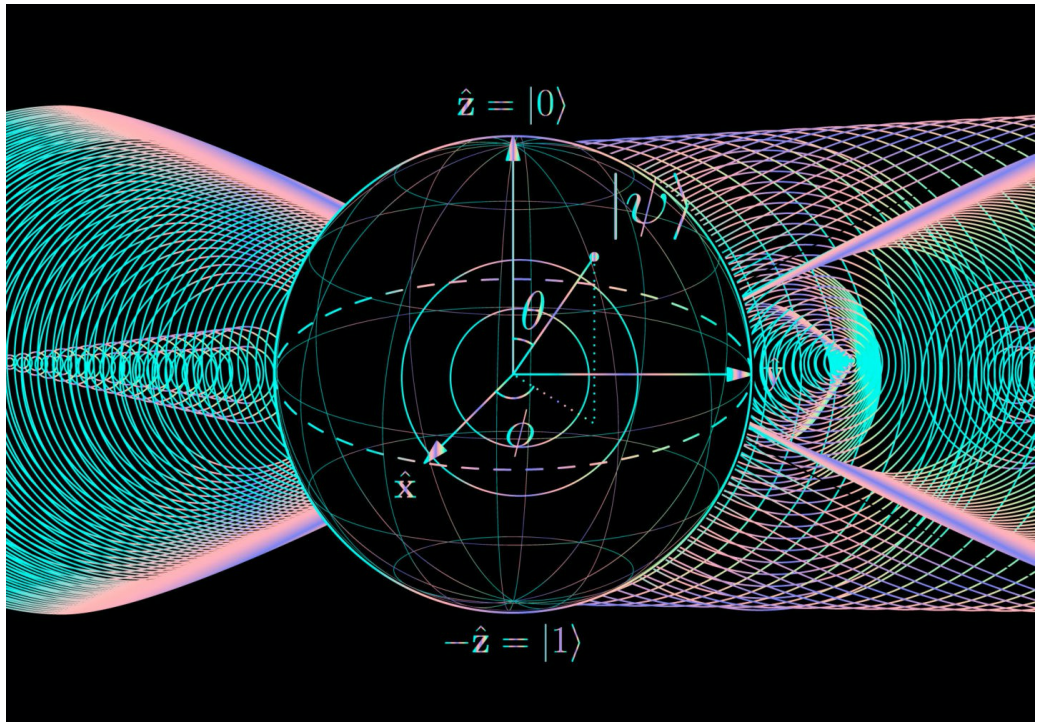


QUANTUM SPEEDUPS

A key goal of quantum computation is leveraging our added quantum resources (entanglement, superposition, and quantum interference) to design ***quantum algorithms***, which are more powerful than any classical algorithm and enable us to efficiently solve ***certain types of problems***!

Richard Feynman

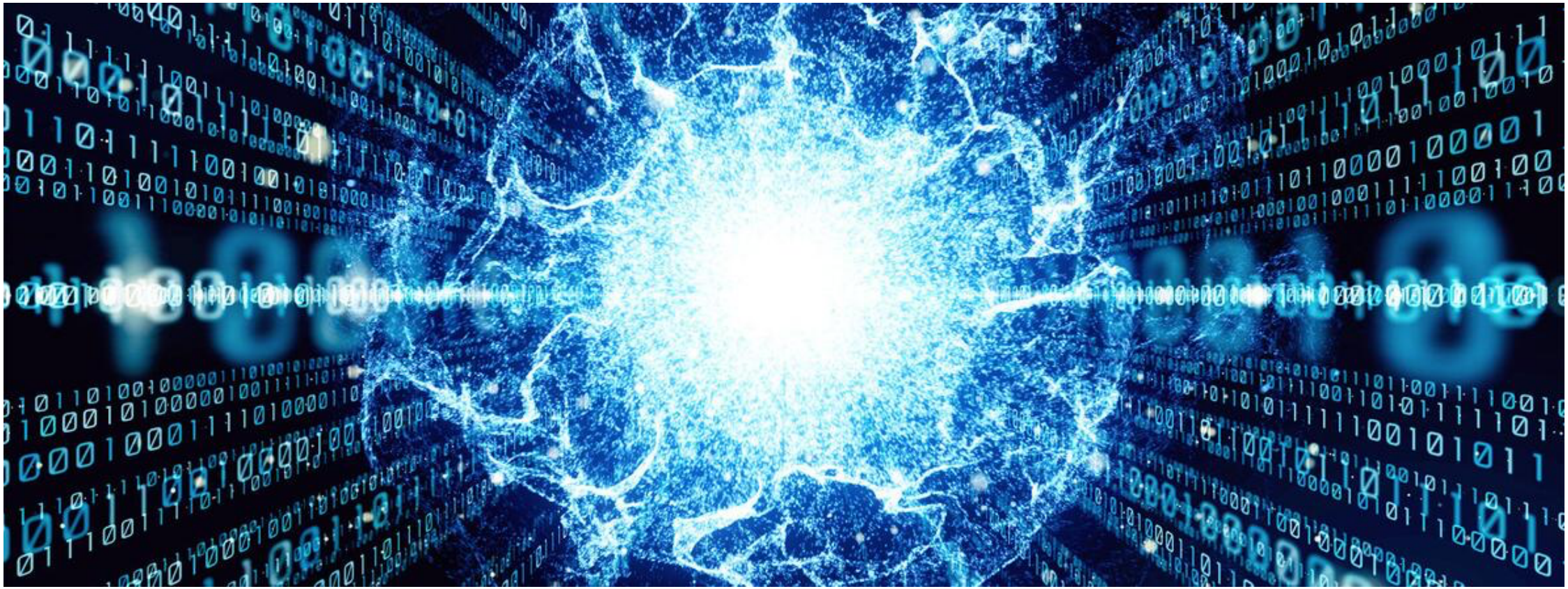
Let's simulate quantum on quantum!

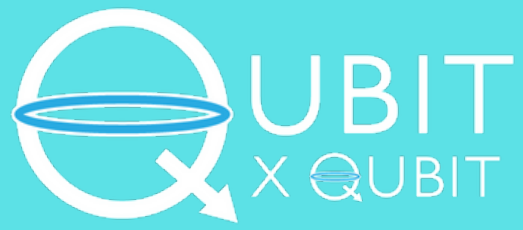


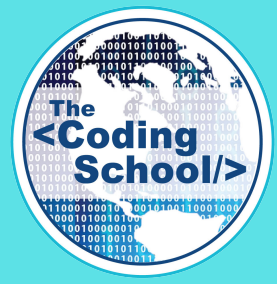
QUANTUM “COMPUTATION”

Thus far, we really have focused on quantum mechanics and quantum information.

In today’s lecture, we focus on ***quantum algorithms***, which is really the ”computation” in quantum computation…





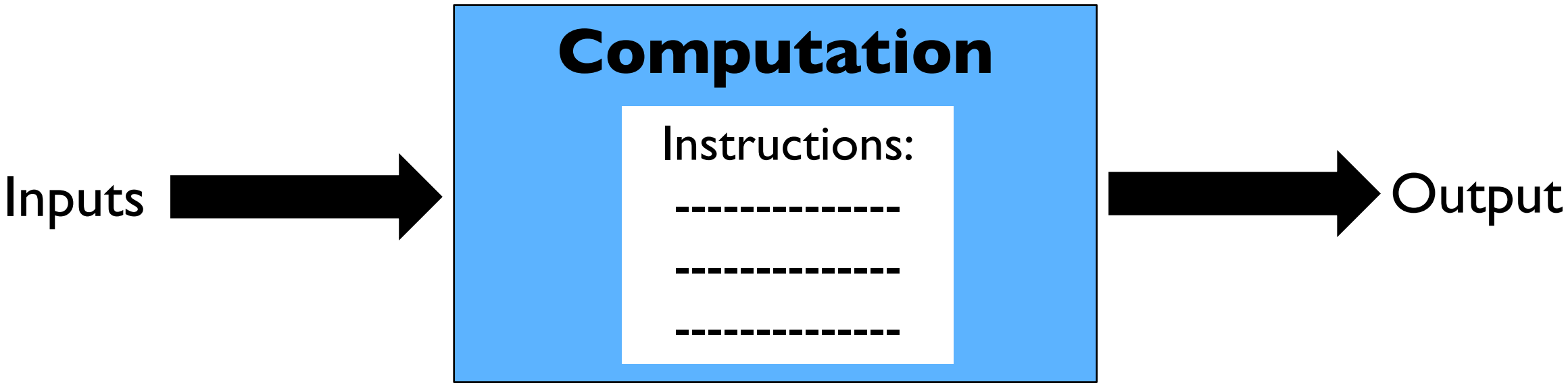


WHAT IS AN ALGORITHM?

ALGORITHM DEFINITION

An ***algorithm*** is a set of instructions or rules that, especially if given to a computer,

will help to calculate an answer to a problem.



REAL LIFE ALGORITHM

An example of an algorithm you have definitely come across is a ***recipe***.

"Inputs"

(ingredients)

- 1 red onion
- 2 red peppers
- 120g bacon
- 1 can tomatoes
- 1 cup water
- olive oil
- garlic
- oregano
- 50 g pasta per person

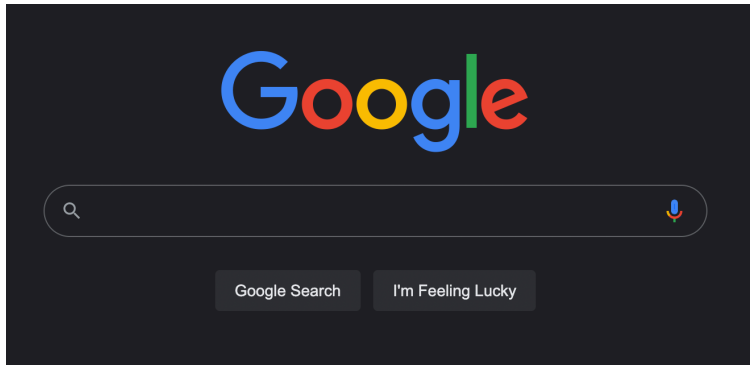
"Computation"

1. Cut the onion, bacon, and red pepper into small pieces.
2. Heat some olive oil in a pan and fry the onion, bacon, and pepper.
3. Add oregano, garlic, tomatos, and water and cook for 20 min.
4. Cook the pasta in a big pot of boiling water.
5. Put the sauce on the pasta and serve!

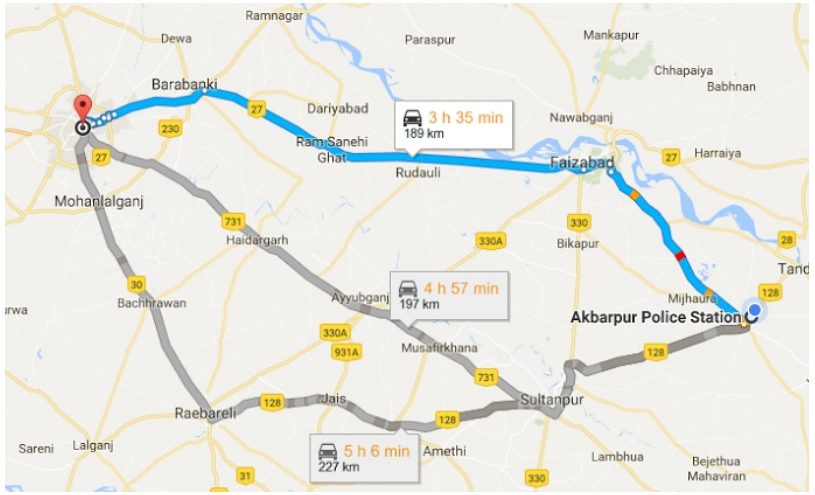
"Output"

COMPUTER SCIENCE ALGORITHMS

Most of computer science revolves around developing ***efficient*** algorithms. Turns out you use algorithms all the time...



Google search uses page rank algorithms



Maps uses path-finding algorithms

Spotify and Netflix use recommendation algorithms

Let's see an example of an algorithm implemented in Python!

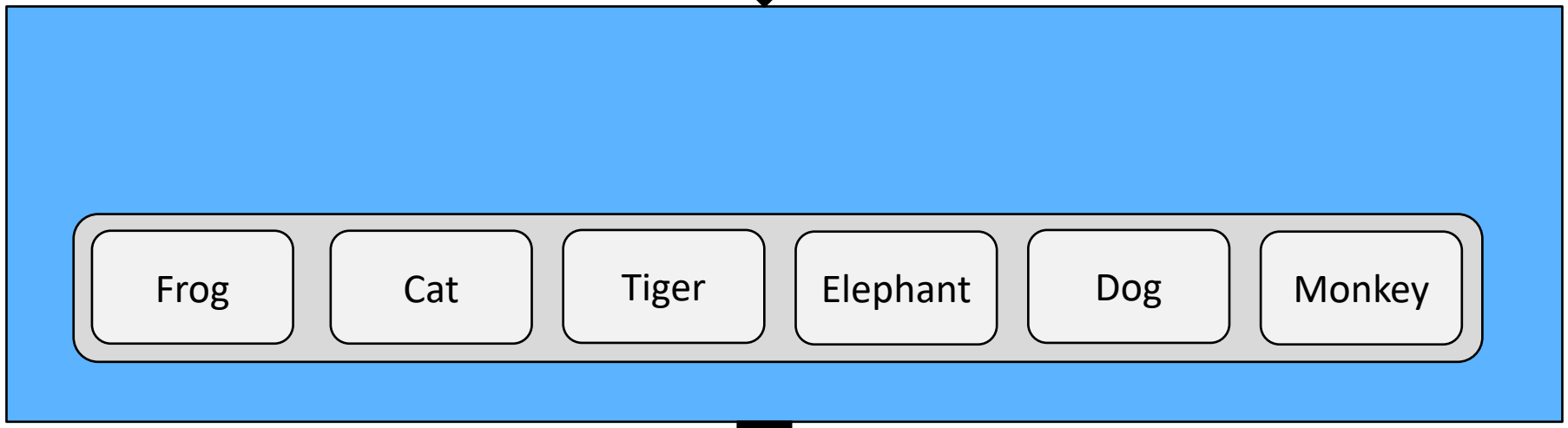
ALGORITHM EXAMPLE - SEARCH

Linear Search Algorithm:

Input:

```
animals = [ 'frog', 'cat', 'tiger', 'elephant', 'dog', 'monkey' ]
```

Computation:



Output:

```
dog_idx
```

ALGORITHM EXAMPLE - SEARCH

Python implementation of our linear search algorithm:

```python
# algorithm input
animals = [ 'frog', 'cat', 'tiger', 'elephant', 'dog', 'monkey' ]

# linear search algorithm - find 'dog'
dog_idx = -1
for i, animal in enumerate(animals):
    if animal == 'dog':
        dog_idx = i
        break

# verify outputs
print("Dog index:", dog_idx)
print("Animal at dog_idx:", animals[dog_idx])
```

Code output:

```
Dog index: 4
Animal at dog_idx: dog
```

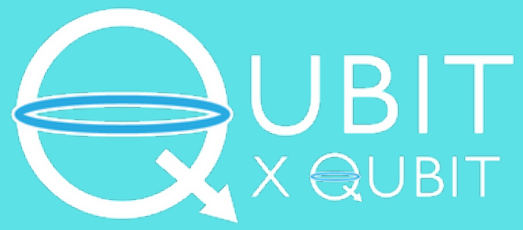
IS MY ALGORITHM GOOD?

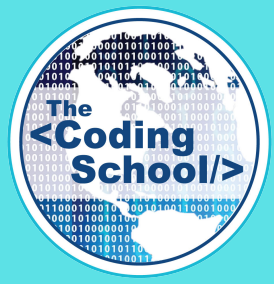
Now that we have a search algorithm, how can we tell if it is good?

What does ”good” even mean in the context of algorithms?

What are the characteristics of a good algorithm?

How can we compare different algorithms?





CHARACTERIZING ALGORITHM PERFORMANCE

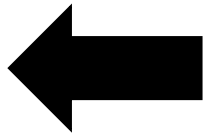
GOOD ALGORITHM DESIGN

We have two key goals in designing an algorithm.

Irrespective of the input length or difficulty, the algorithm should:

1. Solve the problem correctly. (***ACCURACY***)

2. Solve the problem as quickly as possible. (***EFFICIENCY***)



QUANTIFYING ALGORITHM EFFICIENCY

Assuming that my algorithm is correct, how can I quantify it's efficiency?

Naïve Idea: report the algorithm ***runtime*** on my computer for a specific input

```python
# algorithm input
animals = [ 'frog', 'cat', 'tiger', 'elephant', 'dog', 'monkey' ]

# linear search algorithm - find 'dog'
dog_idx = -1
for i, animal in enumerate(animals):
    if animal == 'dog':
        dog_idx = i
        break

# verify outputs
print("Dog index:", dog_idx)
print("Animal at dog_idx:", animals[dog_idx])
```

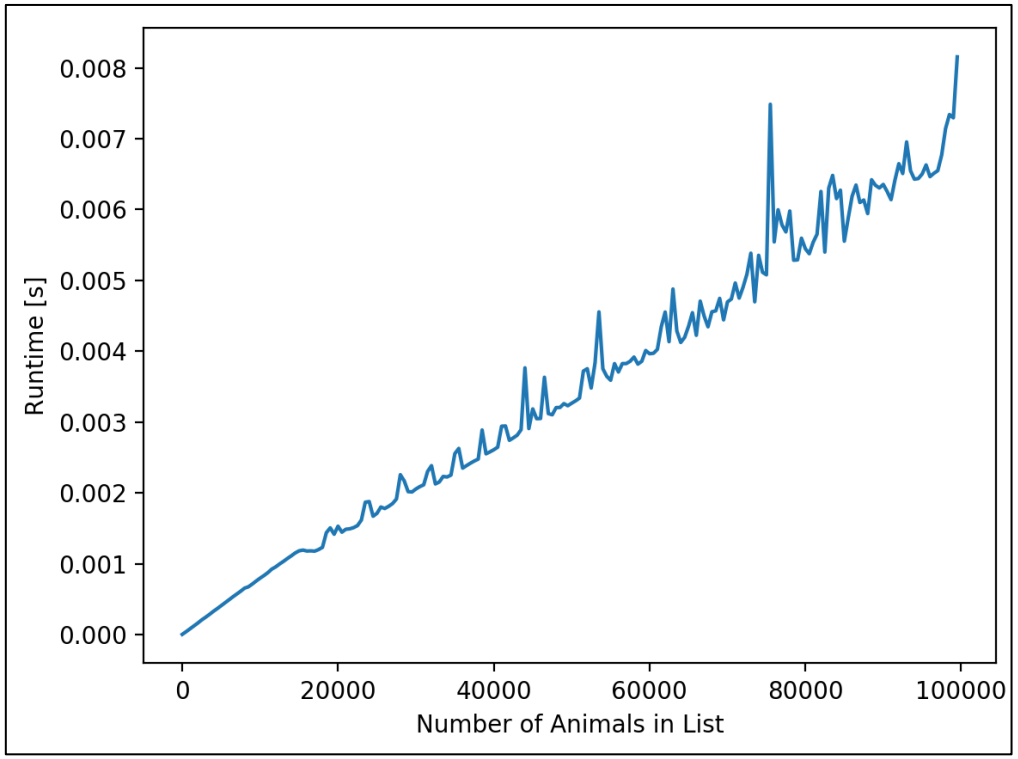
Running this code on my laptop took:

0.472 milliseconds

What are some potential problems in characterizing algorithms by their runtime?

NAÏVE RUNTIME ISSUES

Naïve Idea: report the algorithm ***runtime*** on my computer for a specific input



Issues:

- Runtime is **hardware dependent**

  (*a supercomputer can do it faster than your laptop!)*

- Runtime depends on the **input difficulty**

  (*if 'dog' came last in the list it would take longer*)

- Runtime depends on the **input size**

  (*if the list had 1,000 animals 'dog' would probably take longer to find*)

BETTER RUNTIME METRIC

Computer scientists have developed a really good metric for characterizing algorithmic efficiency, which addresses all these issues, known as ***Big-O notation***.

Naïve Runtime Issues:

- Runtime is ***hardware dependent***
- Runtime depends on the ***input difficulty***
- Runtime depends on the ***input size***

Big-O Solution:

- Big-O reports ***# of operations***, not actual runtime
- Big-O characterizes the ***worst-case*** performance
- Big-O is expressed as ***function of input size***

BIG-O NOTATION

Big-O Solution:

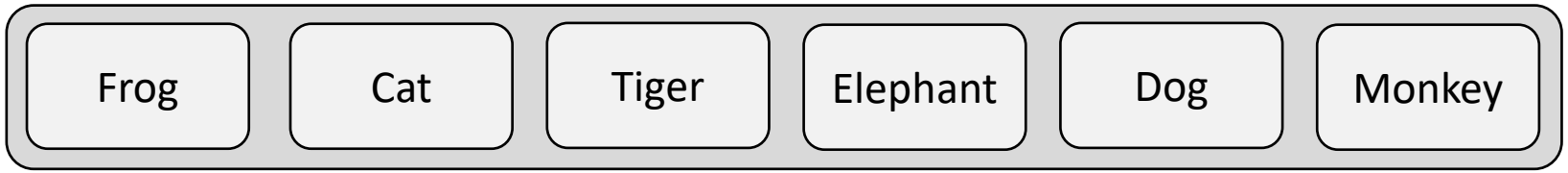
- Big-O reports ***# of operations***, not actual runtime
- Big-O characterizes the ***worst-case*** performance
- Big-O is expressed as ***function of input size***

The actual time it takes code to run depends on a lot of things that are independent of your algorithm, such as the hardware speed.

In Big-O we count the number of operations, so that we can focus purely on the algorithm's efficiency.

For example: In our linear search algorithm,

```
dog_idx = -1
for i, animal in enumerate(animals):
    if animal == 'dog':
        dog_idx = i
        break
```



BIG-O NOTATION

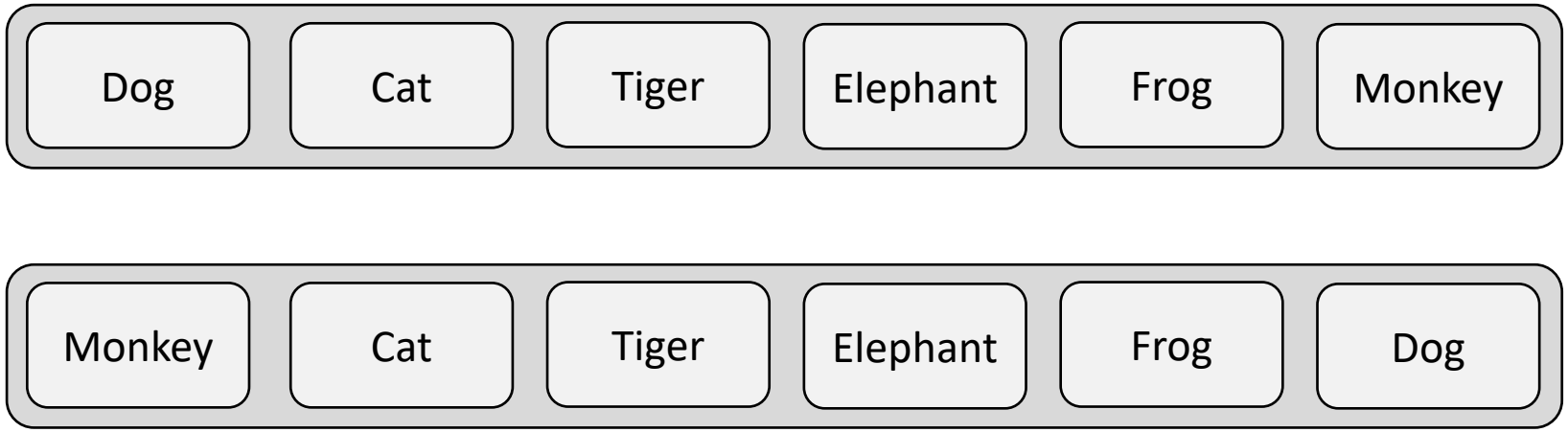
Big-O Solution:

- Big-O reports ***# of operations***, not actual runtime
- Big-O characterizes the ***worst-case*** performance
- Big-O is expressed as ***function of input size***

Dependent on how the input is structured, our algorithm could take more or less operations to solve.

In Big-O, we only care about the worst-case (hardest input, longest runtime).

For example: In our linear search algorithm,



BIG-O NOTATION

Big-O Solution:

- Big-O reports ***# of operations***, not actual runtime
- Big-O characterizes the ***worst-case*** performance
- Big-O is expressed as ***function of input size***

We want to know how our algorithm will perform for any input. So,

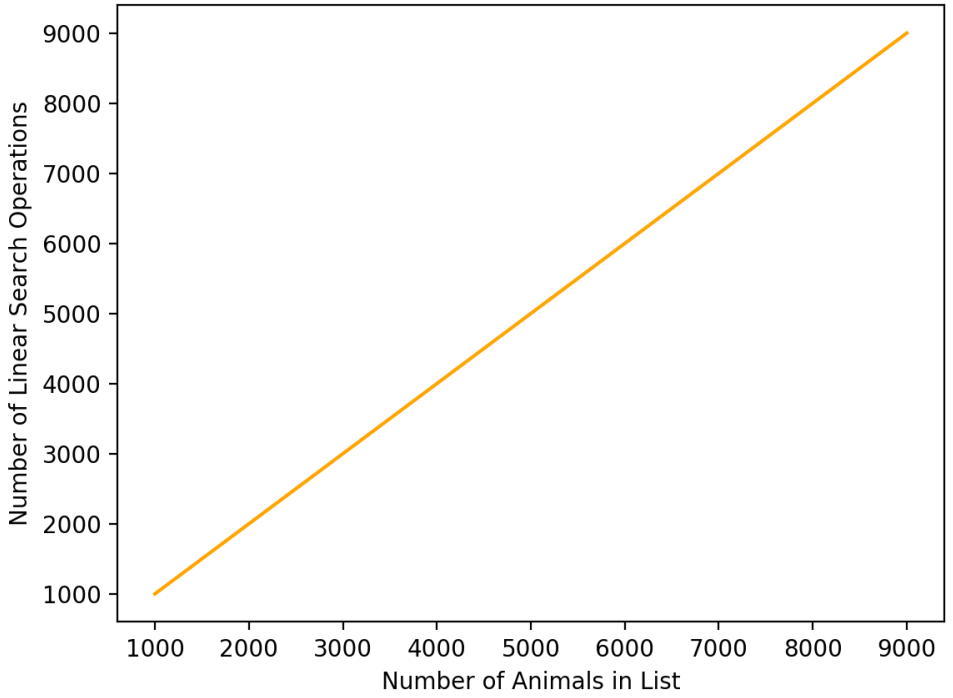
Big-O is expressed as a function of input length (n).

For example: In our linear search algorithm,

In the worst-case, we will need to check every list item, meaning we need to perform as many operations as there are list items.

So, given a list of length n, we will need to perform n operations.

Using Big-O notation, we say that linear search is $O(n)$



COMPARING BIG-O

Different algorithms will perform differently. To say whether one algorithm is better or worse, we need to know how to different Big-Os ***compare***.

$O(1)$ *Constant*

$O(\log(n))$ *Logarithmic*

$O(\sqrt{n})$ *Square-Root*

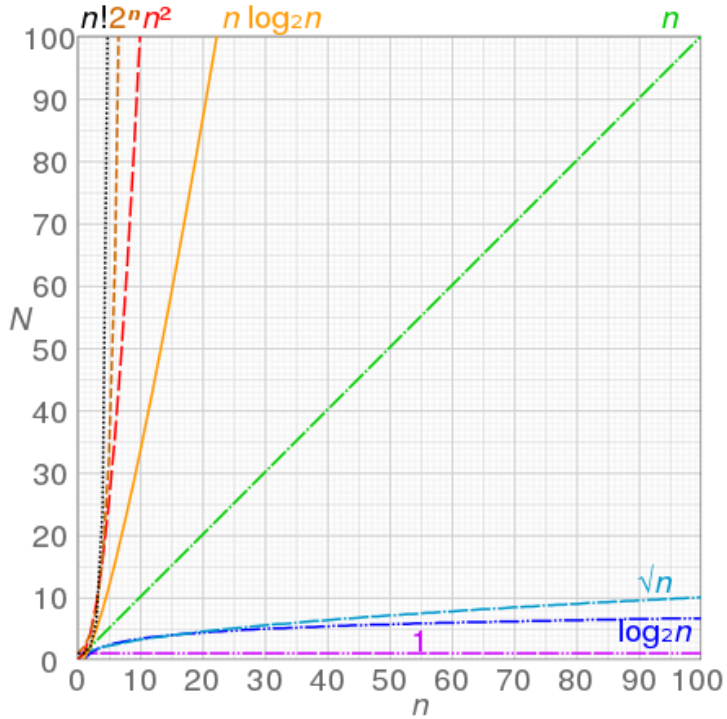
$O(n)$ *Linear*

$O(n^2)$ *Quadratic*

$O(2^n)$ *Exponential*

$O(n!)$ *Factorial*

More efficient *algorithms have Big-Os that are* ***smaller functions*** *of input size*



COMPARING BIG-O

For large input sizes, the largest Big-O term in our function will dominate.

Thus, Big-O ignores smaller terms and multiplicative constants.

Examples:

$$O(10) = O(1)$$

$$O(2n + 3) = O(n)$$

$$O(3^n + 7n^3) = O(3^n)$$

$$O(n^4 + 3{,}000n^3) = O(n^4)$$

QUANTUM PRACTICE TIME!

State whether the following Big-O times are =, <, or >.

(1) $O(2n + 1)$ $O(n^3)$

(2) $O(1000n + 30)$ $O(3n)$

(3) $O(\log(n))$ $O(\sqrt{n})$

(4) $O(6^n)$ $O(2^n)$

(5) $O(10^n)$ $O(3^n)$

(6) $O(3n^5 + 200n^2 + 4e^n)$ $O(e^n)$

QUANTUM PRACTICE SOLUTION

State whether the following Big-O times are =, <, or >.

(1) $O(2n + 1)$ $O(n^3)$

(2) $O(1000n + 30)$ $O(3n)$

(3) $O(\log(n))$ $O(\sqrt{n})$

(4) $O(6^n)$ $O(2^n)$

(5) $O(10^n)$ $O(3^n)$

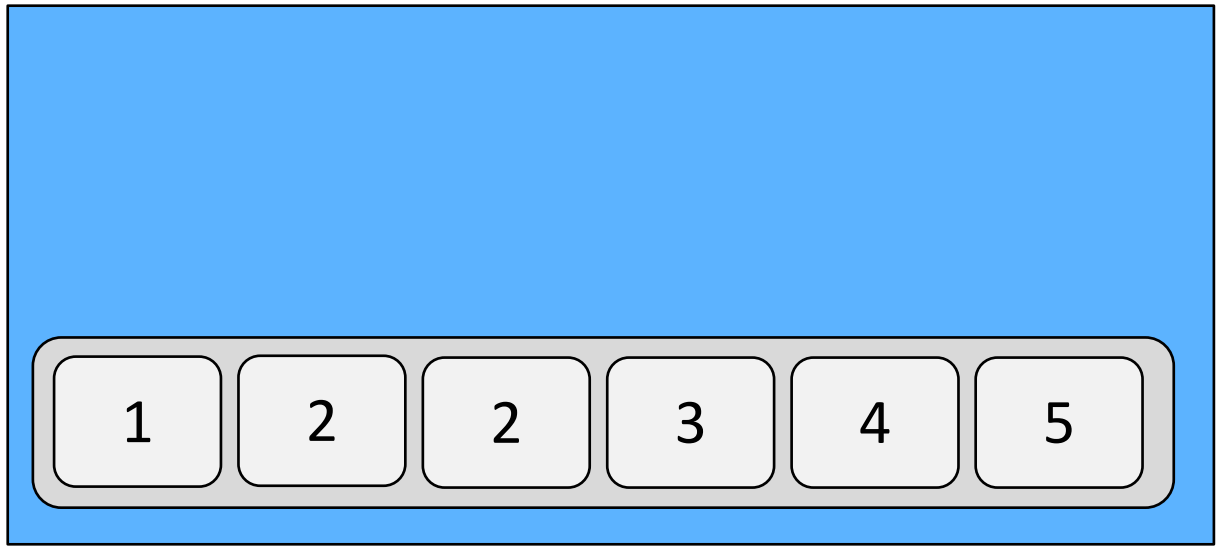
(6) $O(3n^5 + 200n^2 + 4e^n)$ $O(e^n)$

IMPROVING ALGORITHM BIG-O

Key Idea: The better the algorithm, the smaller it's Big-O!

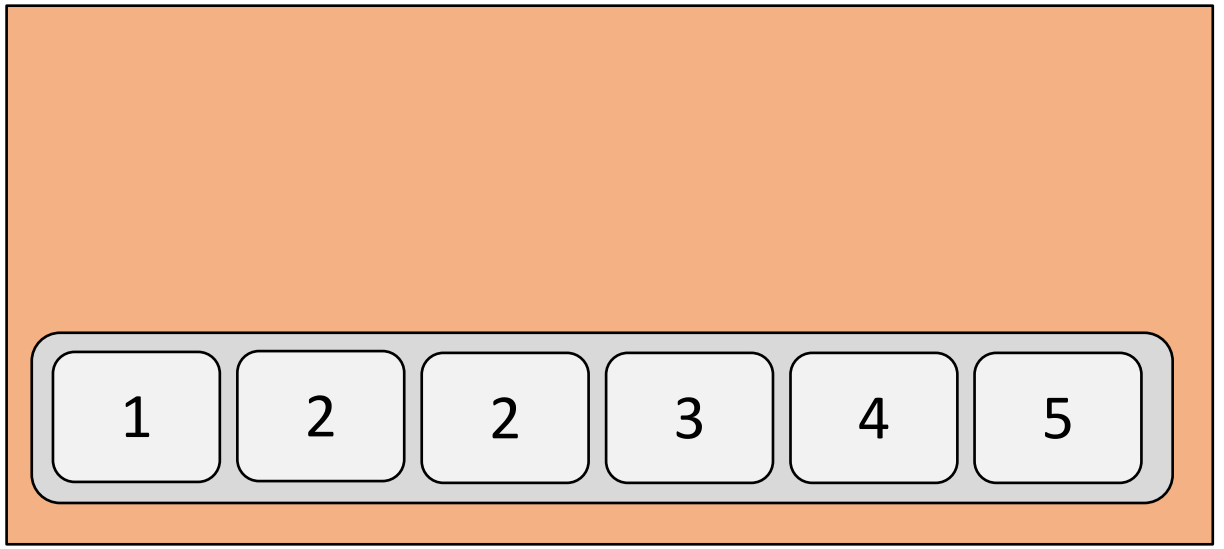
Given a ***sorted*** list of numbers, is there a more efficient algorithm than linear search for finding, say, the number 5?

Linear Search



Big-O: $O(n)$

Binary Search



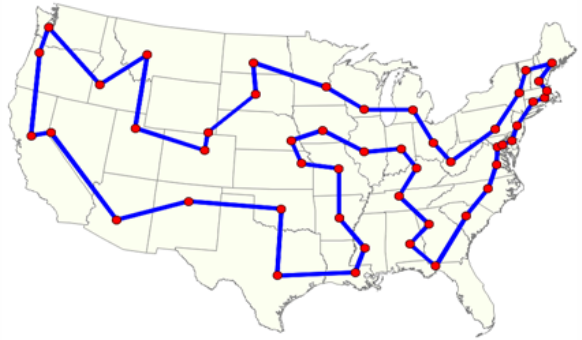
Big-O: $O(\log(n))$

QUANTUM BIG-O

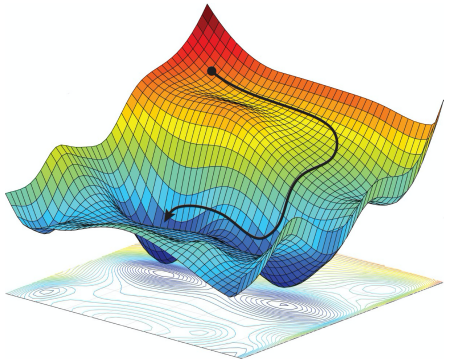
We will also use Big-O to characterize the efficiency of ***quantum algorithms***! The goal in designing a good quantum algorithm is to achieve a Big-O smaller than that of any classical algorithm for the same task...

$$O(Quantum) < O(Classical)$$

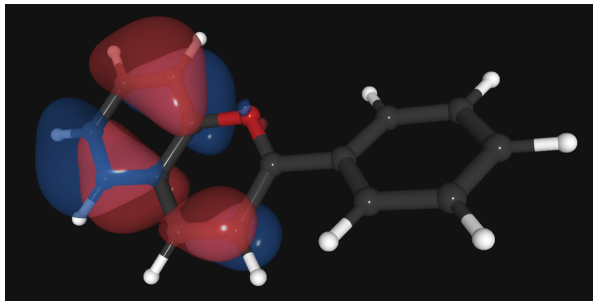
It turns out that a lot of important problems have large classical Big-O's:



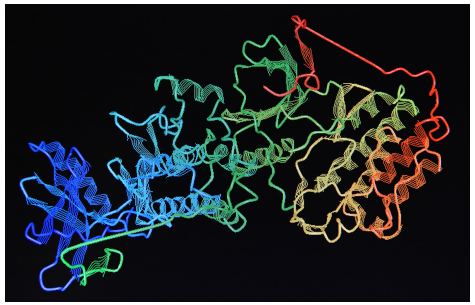
Traveling Salesman



Optimization / Machine Learning

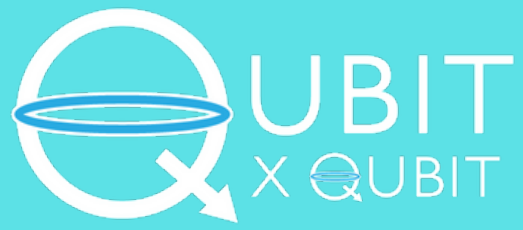


Quantum Chemistry / Simulation



Protein Folding / Drug Design

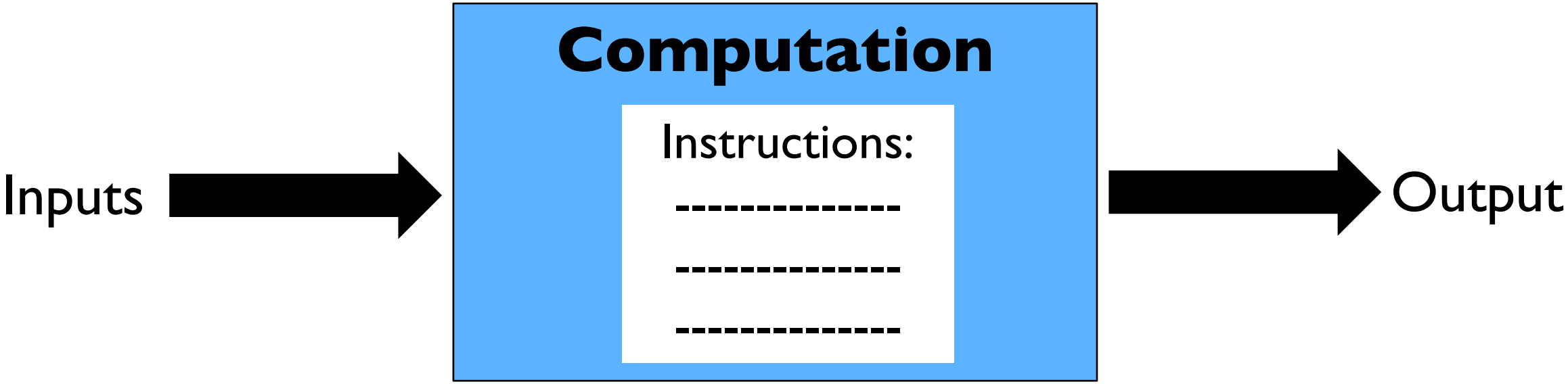
And we think quantum can help us solve them!



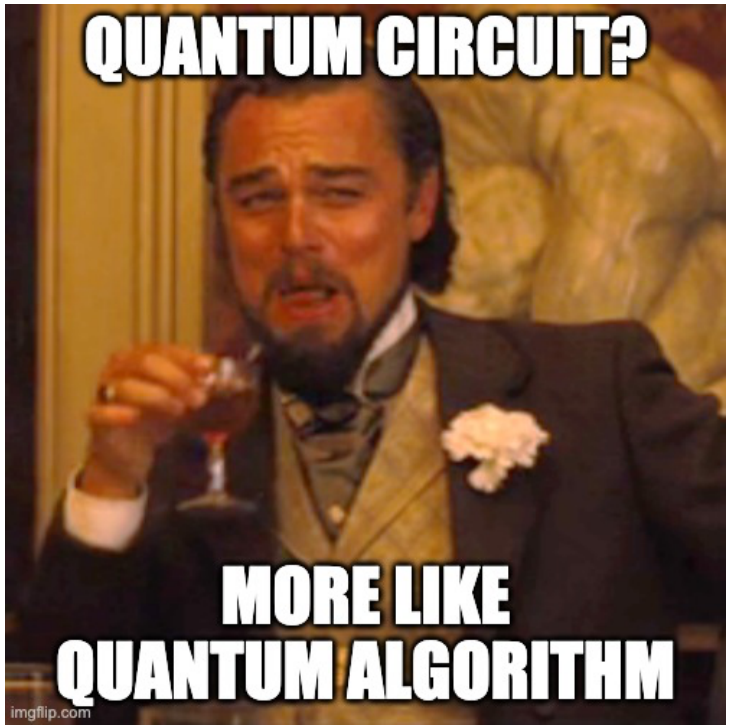
THE QUANTUM ALGORITHMS LANDSCAPE

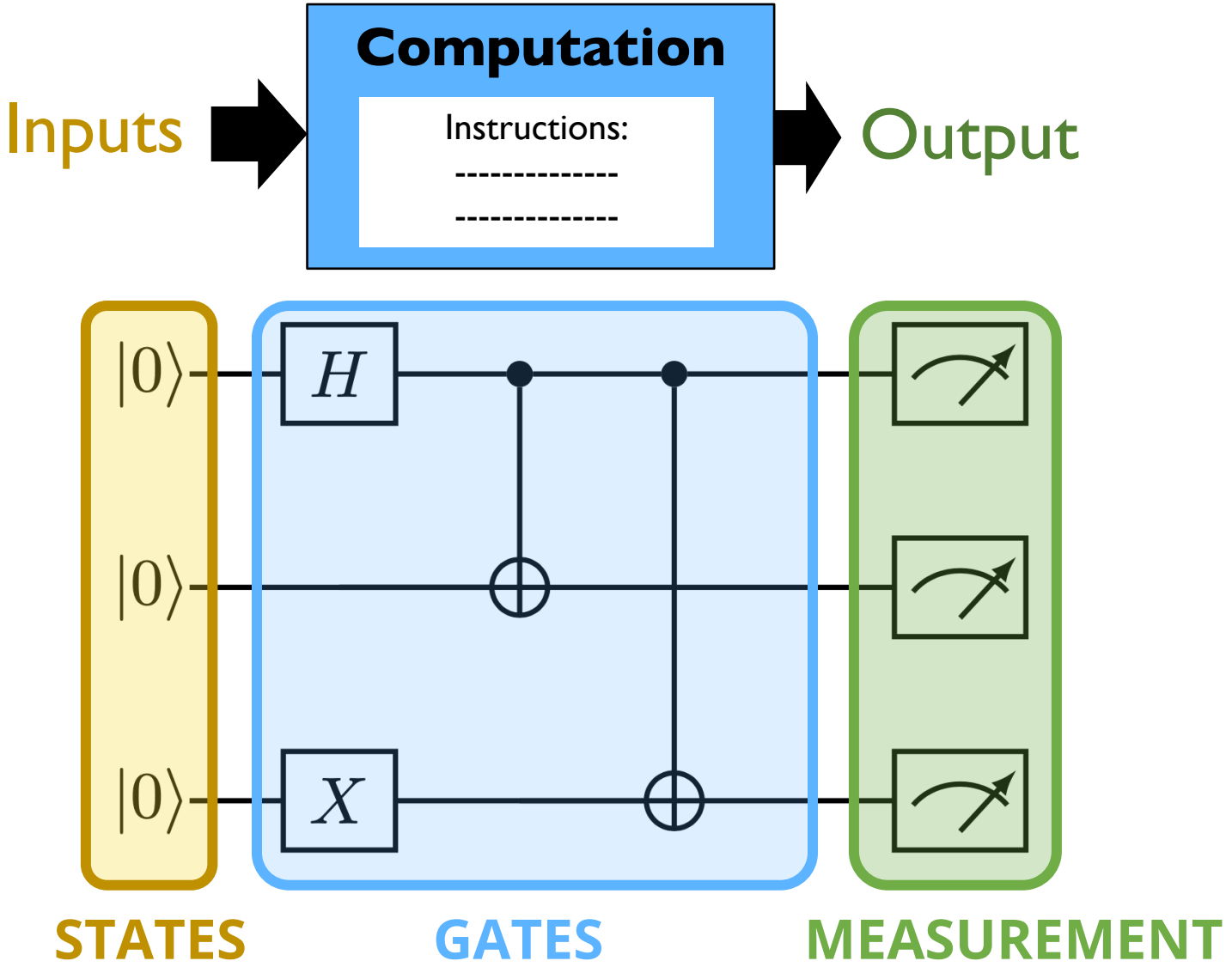
WHAT IS A QUANTUM ALGORITHM?

Doesn't this structure look familiar?



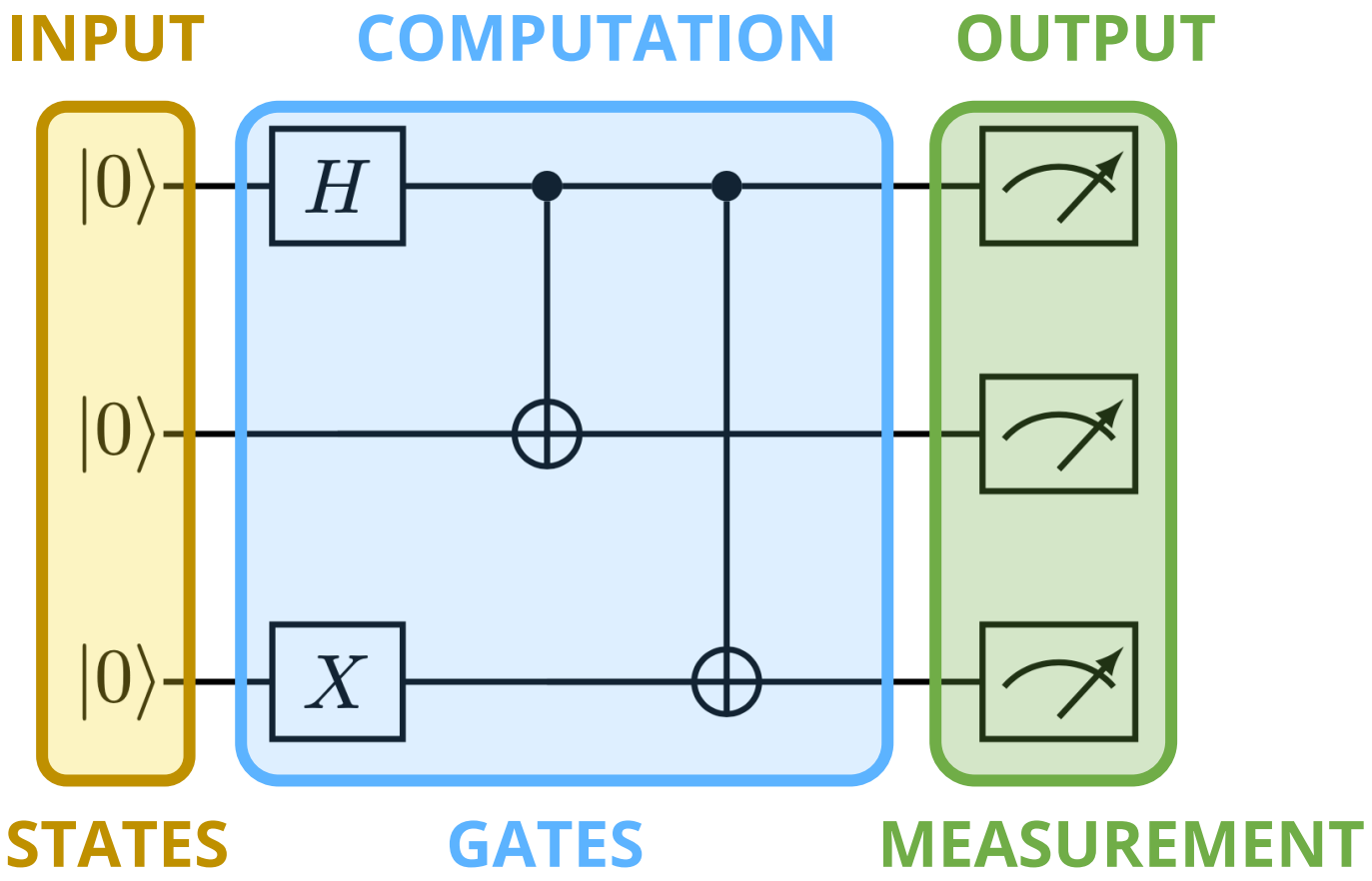
WHAT IS A QUANTUM ALGORITHM?





WHAT IS A QUANTUM ALGORITHM?

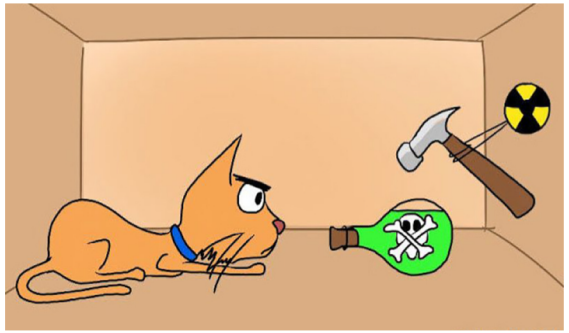
All quantum algorithms boil down to quantum circuits, but not every quantum circuit is a ***useful*** quantum algorithm...

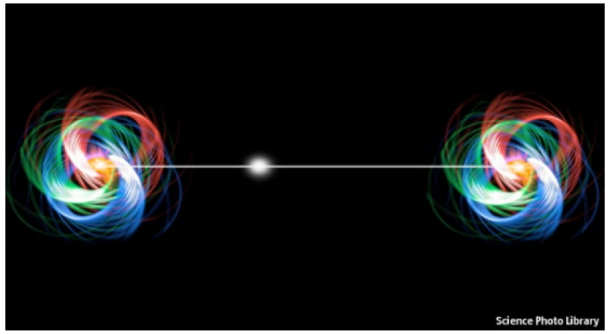


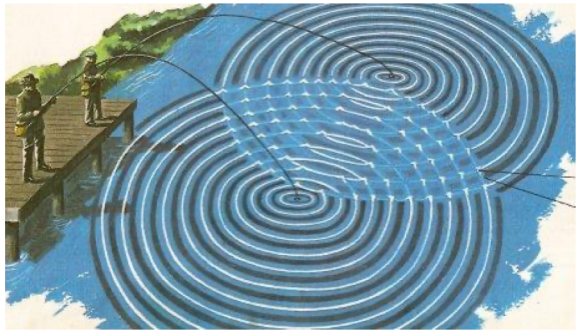
"GOOD" QUANTUM ALGORITHMS

We believe there is a lot of potential for using quantum computers to get big computational advantages over classical systems, but this requires clever algorithm design....

Quantum algorithms need to strategically leverage our 3 quantum resources:







Superposition, ***Entanglement***, and ***Quantum Interference***

to outperform their classical counterparts.

It turns out that designing such algorithms is ***challenging***! You can't just throw together any quantum circuit and expect ***quantum advantage***....

QUANTUM ADVANTAGE

Demonstrating that a quantum device can solve a problem significantly faster than any classical device is known as ***quantum advantage*** (traditionally ***quantum supremacy***).

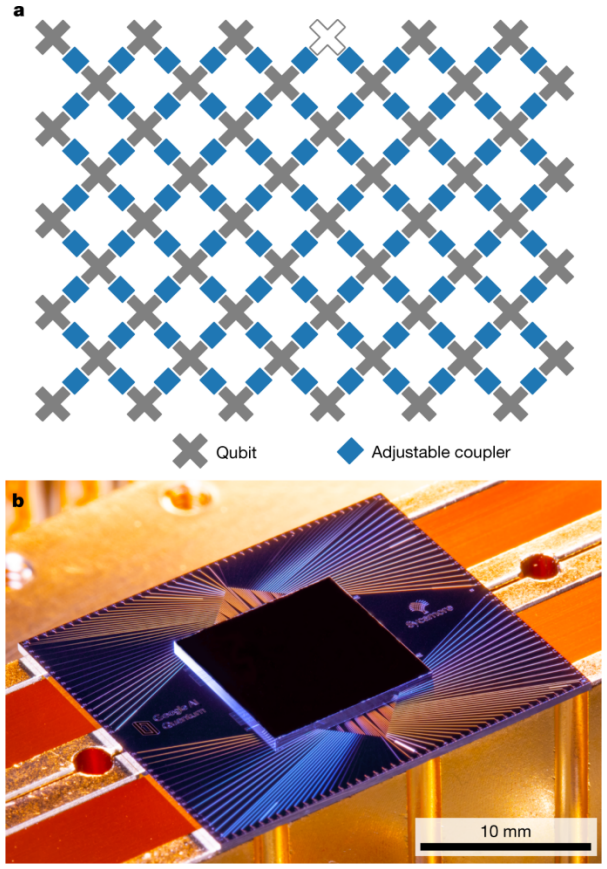
Article | Published: 23 October 2019

Quantum supremacy using a programmable superconducting processor

Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven & John M. Martinis ✉ -Show fewer authors

Nature **574**, 505–510(2019) | Cite this article

807k Accesses | **768** Citations | **6010** Altmetric | Metrics



In October 2019 the first ever experimental demonstration of quantum advantage was achieved by the Google AI Quantum team. With a noisy 53-qubit device they calculated a result in a few minutes that would take the world's largest supercomputer multiple days!!

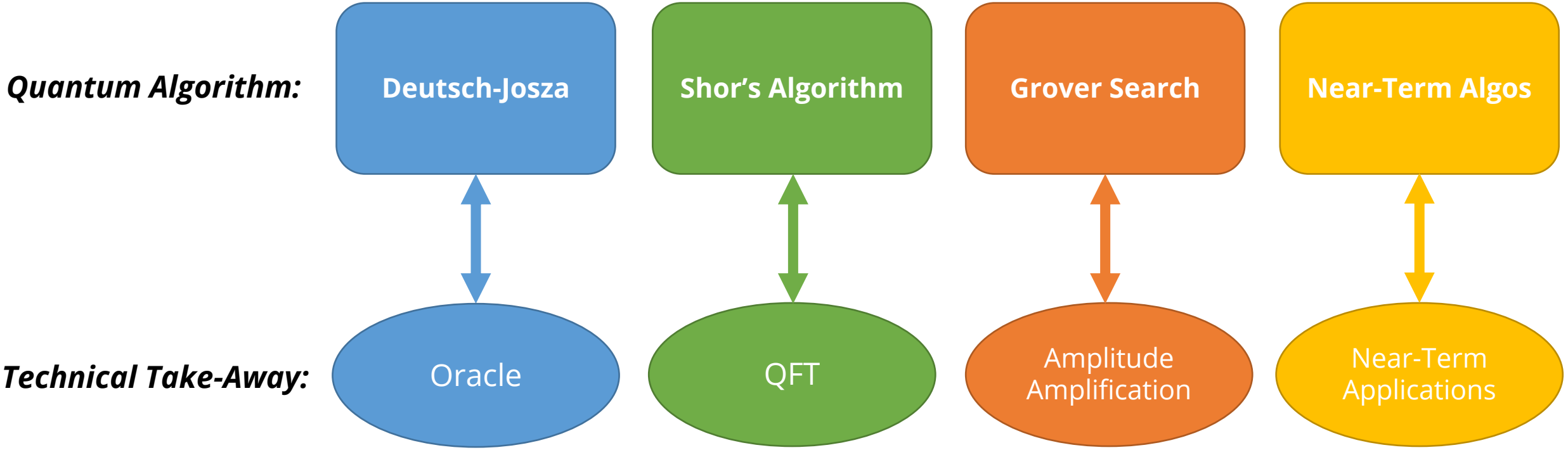
#QuantumAdvantage

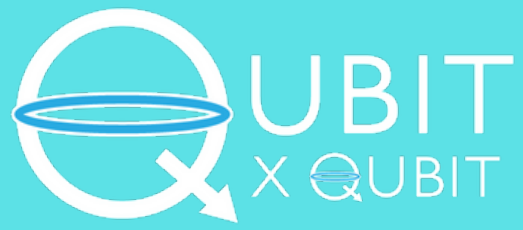
THE QUANTUM ALGOS LANDSCAPE

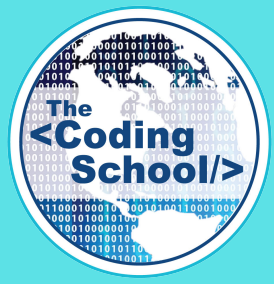
Currently, there exist only a few core quantum algorithms which offer big computational advantages.

We are going to spend the rest of looking at these core algorithms and what makes them interesting.

Since these algorithms are complicated, we will also focus a key-technical take-away.







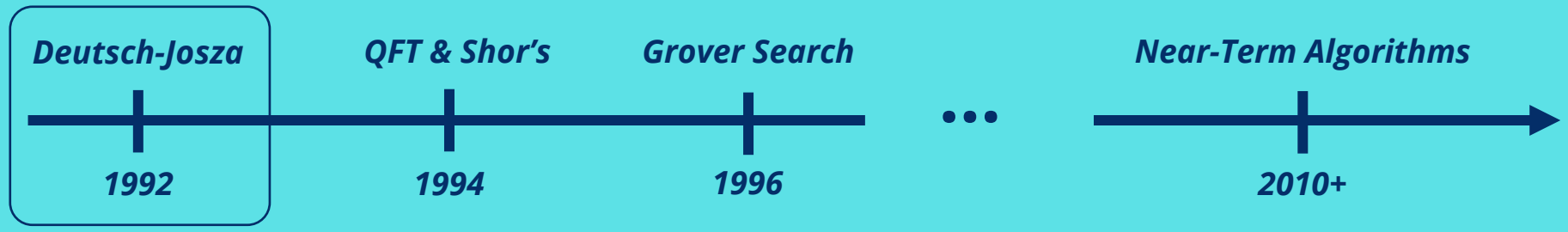
BREAK TIME!

QUBIT X QUBIT

The <Coding School/>

DEUTSCH-JOSZA & ORACLES

QUANTUM ALGORITHMS CRASH COURSE



DEUTSCH-JOZSA: OVERVIEW

Inventors: David Deutsch and Richard Jozsa

Year: 1992

Problem: The algorithm is not practically useful - it solves a "toy problem".

Why do we care?: It was the first theoretical demonstration of quantum advantage!

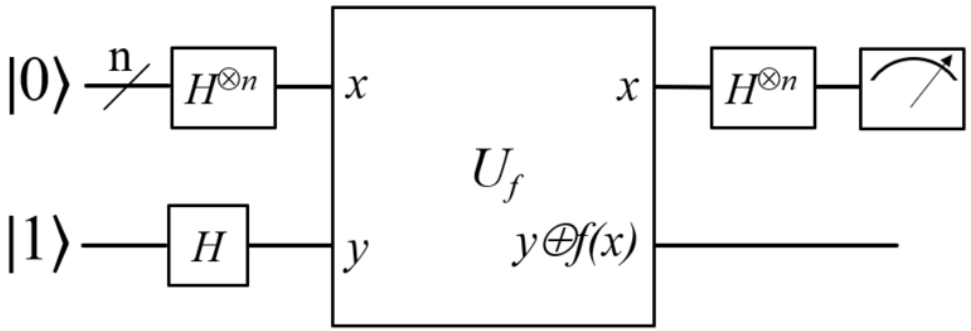
Classical Efficiency: $O(2^n)$

Quantum Efficiency: $O(1)$

Exponential Speedup!

Technical Take-Away: Deutsch-Jozsa uses an ***oracle***

Quantum Circuit:



WHAT'S AN ORACLE?

The notion of an oracle comes from complexity theory.

You can think of an oracle as a ***genie*** that grants you a computational wish.

There are two different types of wishes the oracle genie can grant:

(1) **Decision Problem**: given a problem (specified by a bit-string), the genie will tell you whether the answer is "NO" (0) or "YES" (1)

(2) **Function Problem**: given an input, bit-string x, the genie will calculate the output, bit-string $f(x)$, for some unknown but desired function f

Key Idea: you give the genie (oracle) an input and he performs some computation. You do not know or understand his computation (it's a black-box!), but he always returns the correct output.

DEUTSCH-JOZSA: OVERVIEW

Inventors: David Deutsch and Richard Jozsa

Year: 1992

Problem: The algorithm is not practically useful - it solves a "toy problem".

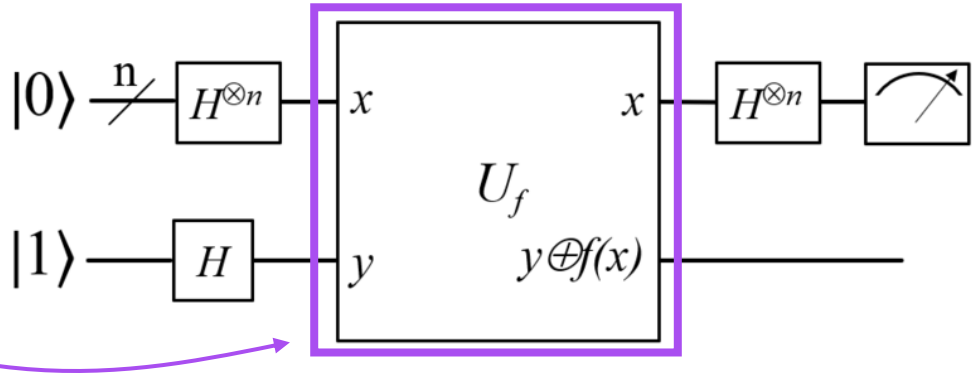
Why do we care?: It was the first theoretical demonstration of quantum advantage!

Classical Efficiency: $O(2^n)$

Quantum Efficiency: $O(1)$

Exponential Speedup!

Quantum Circuit:



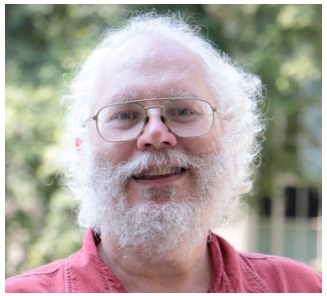
Technical Take-Away: Deutsch-Jozsa uses an ***oracle***

SHOR'S ALGORITHM & QFT

QUANTUM ALGORITHMS CRASH COURSE

SHOR'S ALGORITHM: OVERVIEW

Inventors: Peter Shor (Shor's Algo), Don Coppersmith (QFT)



Year: 1994

Problem: Factor large integers (period finding).

Why do we care?: RSA encryption is only secure if large integers are hard to factor!

RSA SECURITY™

Classical Efficiency: $\sim O(n^{1.9})$

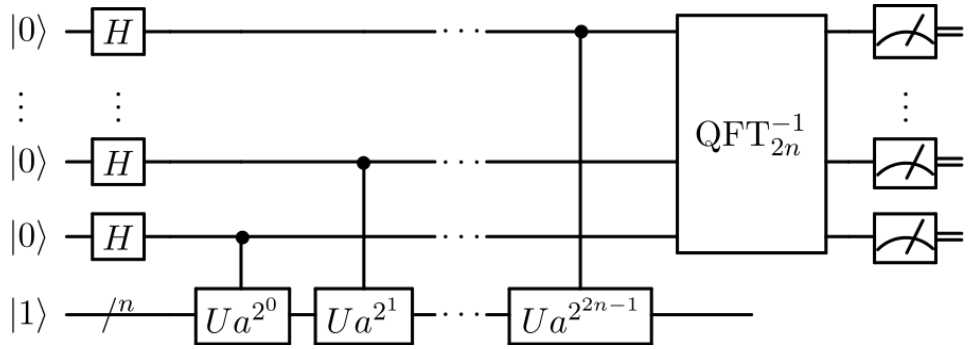
Quantum Efficiency: $\sim O(\log(n)^3)$

Super-Polynomial Speedup!

(note n is the # to factorize)

Technical Take-Away: Shor's algorithm uses the ***QFT***

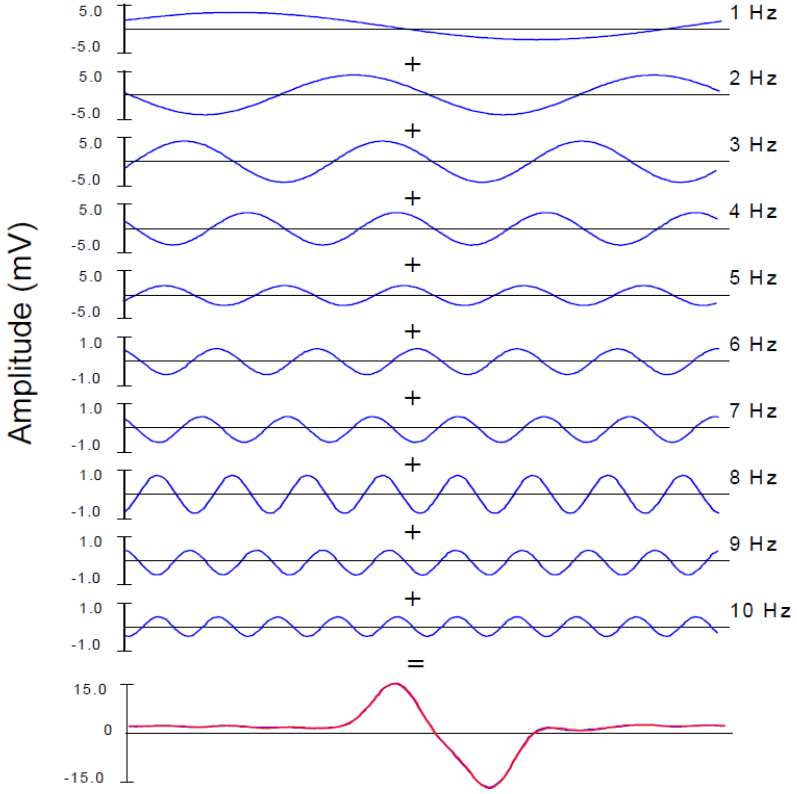
Quantum Circuit:



FOURIER ANALYSIS

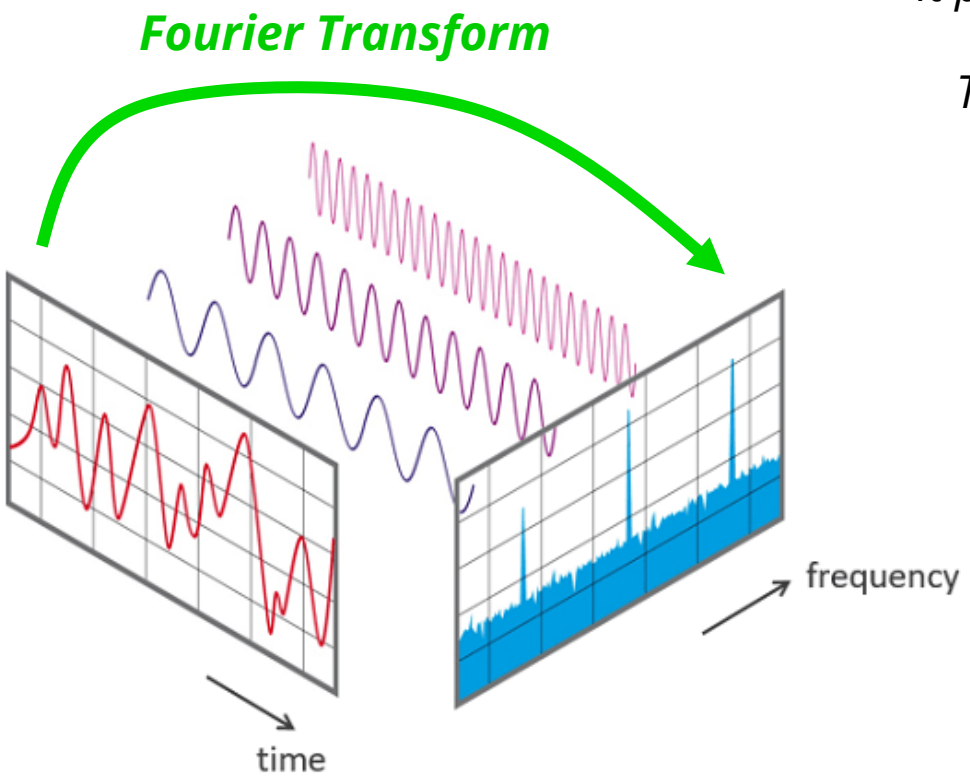
In 1822, French mathematician Joseph Fourier showed that any function can be represented (or very-well approximated) with a ***linear combination*** of trig-functions with varying ***frequencies***.

Joseph Fourier



THE FOURIER TRANSFORM

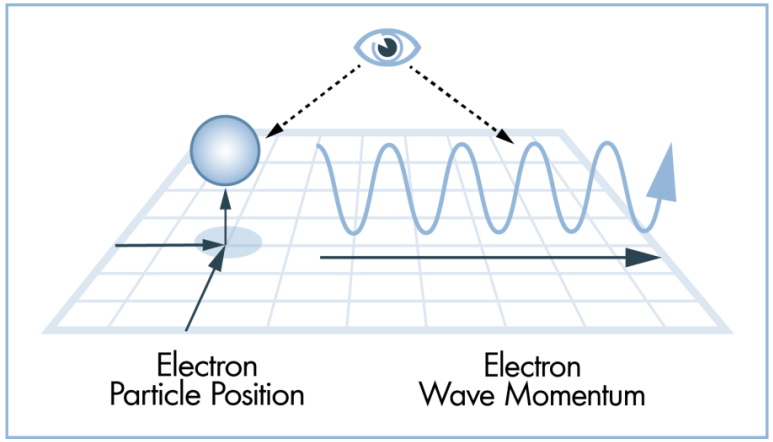
This led him to develop one of the most important equations in science and engineering: the ***Fourier Transform (FT)***.



*It provides a mapping from the **time domain** to the **frequency domain**.*

This enables us to find the most prominent frequencies in our signal.

FUN FACT:

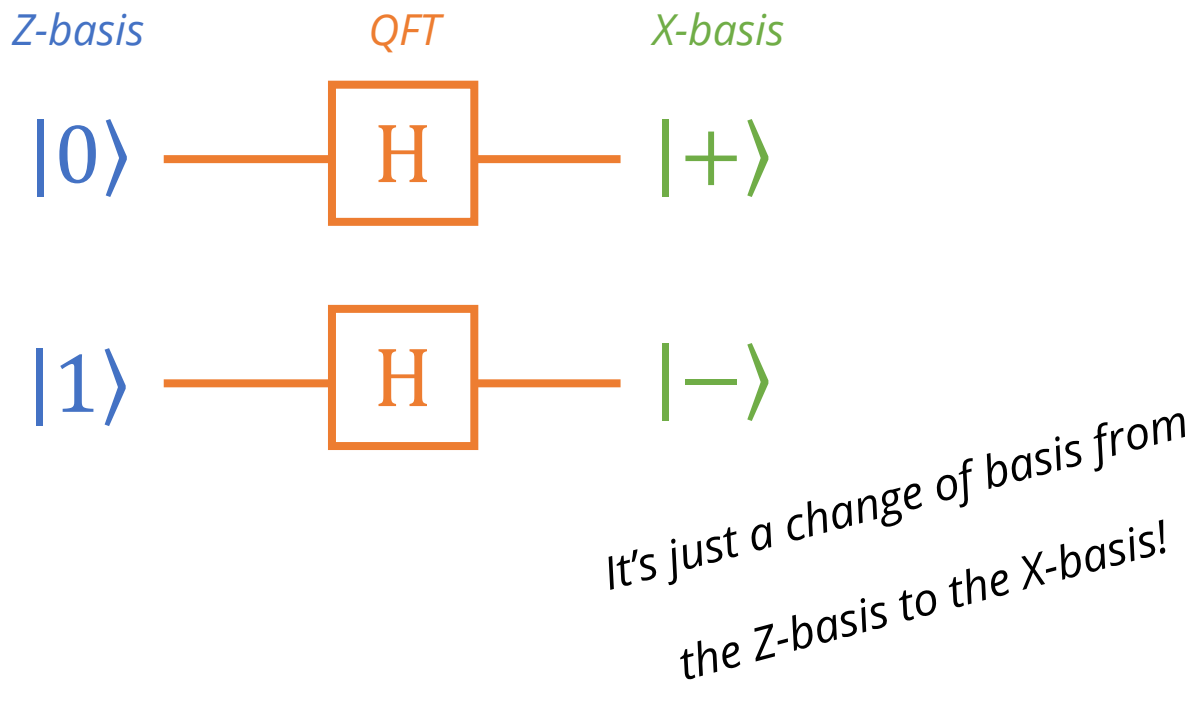


The momentum-position relationship in quantum mechanics is mathematically governed by the FT!

THE QUANTUM FOURIER TRANSFORM

In 1994, Don Coppersmith developed the ***Quantum Fourier Transform (QFT)***, which encodes frequency information in the phase of our quantum states. At the end of the day, however, the FT and QFT are just performing a ***change of basis***...

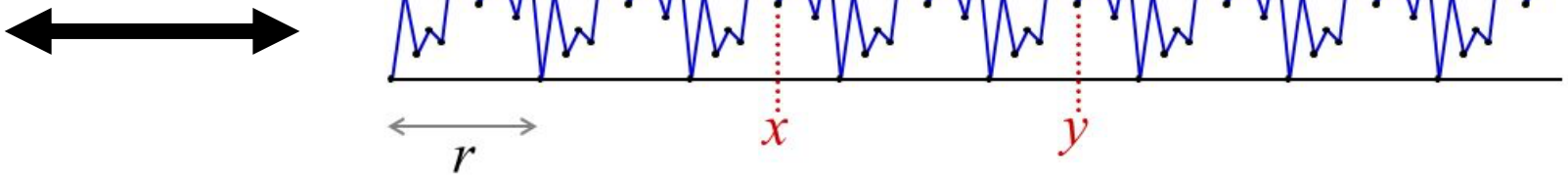
In fact, for a single qubit, the Hadamard is the QFT!



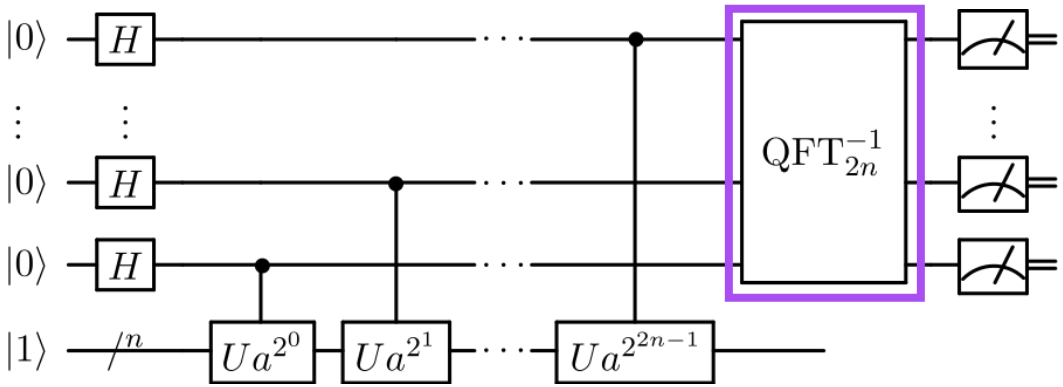
SHOR'S ALGORITHM

Shor cleverly realized that the problem of integer factorization is a problem of ***period-finding***.

RSA SECURITY™



which can be solved with a quantum subroutine known as ***phase estimation***, that leverages the ***QFT*** !



SHOR'S ALGORITHM

*How extreme is this **super-polynomial quantum speedup** for factoring integers?*

Classical Efficiency:

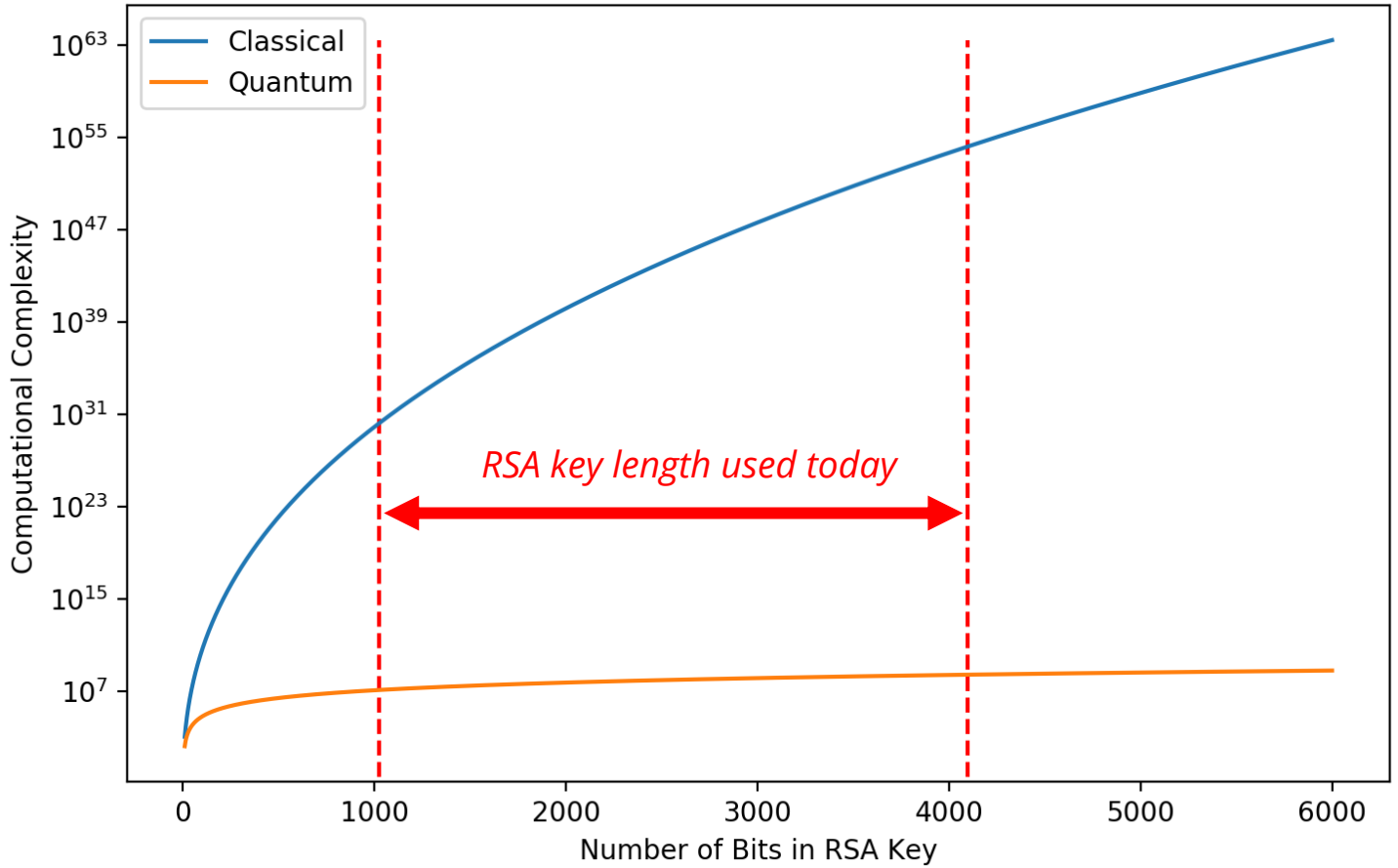
$$O\left(e^{1.9\,\log(n)^{1/3}\,\log(\log(n))^{2/3}}\right)$$

$$\sim O(n^{1.9})$$

Quantum Efficiency:

$$O(\log(n)^2\,\log(\log(n))\,\log(\log(\log(n))))$$

$$\sim O(\log(n)^3)$$



QUBIT X QUBIT

The <Coding School/>

AMPLITUDE AMPLIFICATION & GROVER SEARCH

QUANTUM ALGORITHMS CRASH COURSE

GROVER SEARCH: OVERVIEW

Inventor: Lov Grover

Year: 1996

Problem: Searching assuming ***no*** data structure

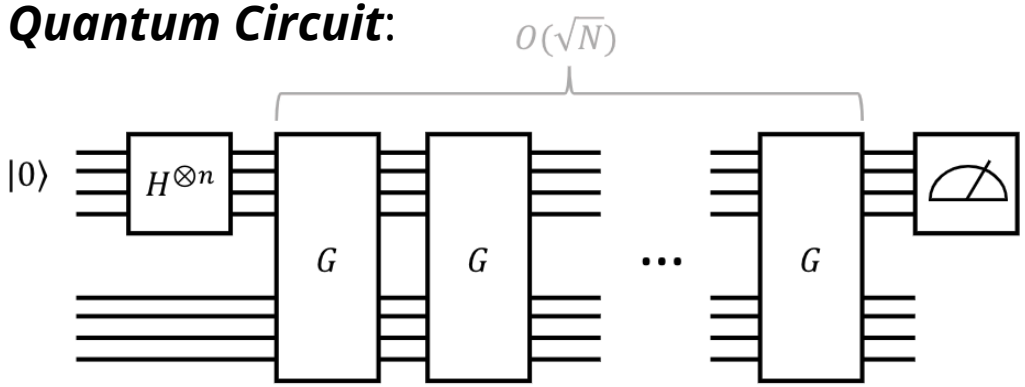
Why do we care?: Search is everywhere – i.e. databases

Classical Efficiency: $O(n)$

Quantum Efficiency: $O(\sqrt{n})$

Quadratic Speedup!

Quantum Circuit:

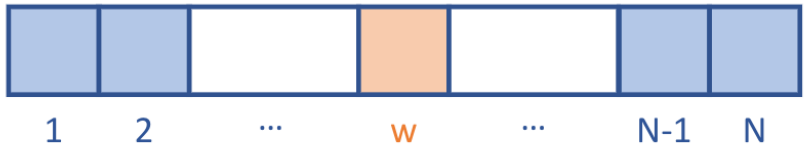


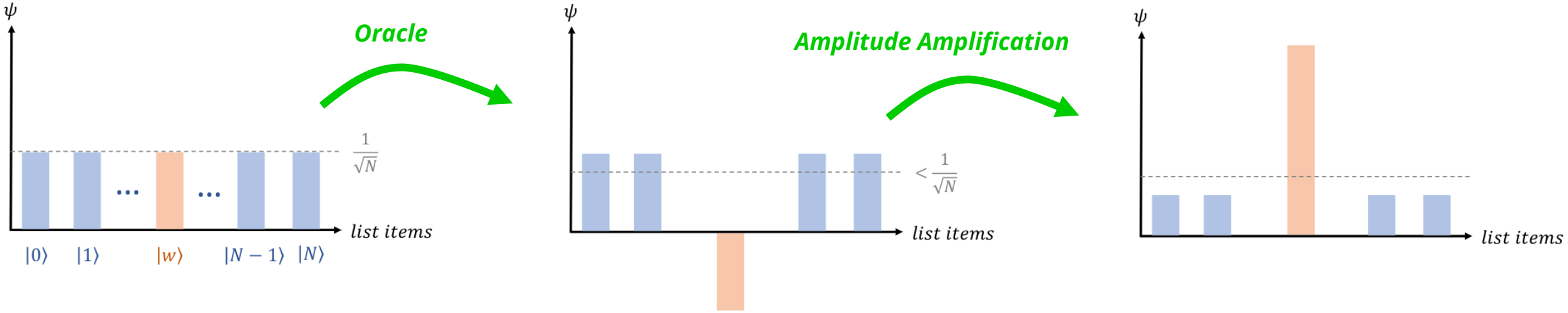
Technical Take-Away: Grover's algorithm uses ***amplitude amplification***

AMPLITUDE AMPLIFICATION

Grover search introduced a technique later formalized as ***amplitude amplification***. Through clever circuit design (using an ***oracle***!), you can ***amplify*** the probability of measuring the bit-string corresponding to the item you are searching for!

Goal: Find the index of w in your list





You start off totally uncertain about where w is, you are in an equal ***superposition*** over all indices.

Apply an ***oracle*** which flips the amplitude of the probability of w's index.

Perform ***amplitude amplification*** by applying an operator which reflects all the amplitudes about the average.

QUBIT X QUBIT

The <Coding School/>

NEAR-TERM HYBRID ALGORITHMS

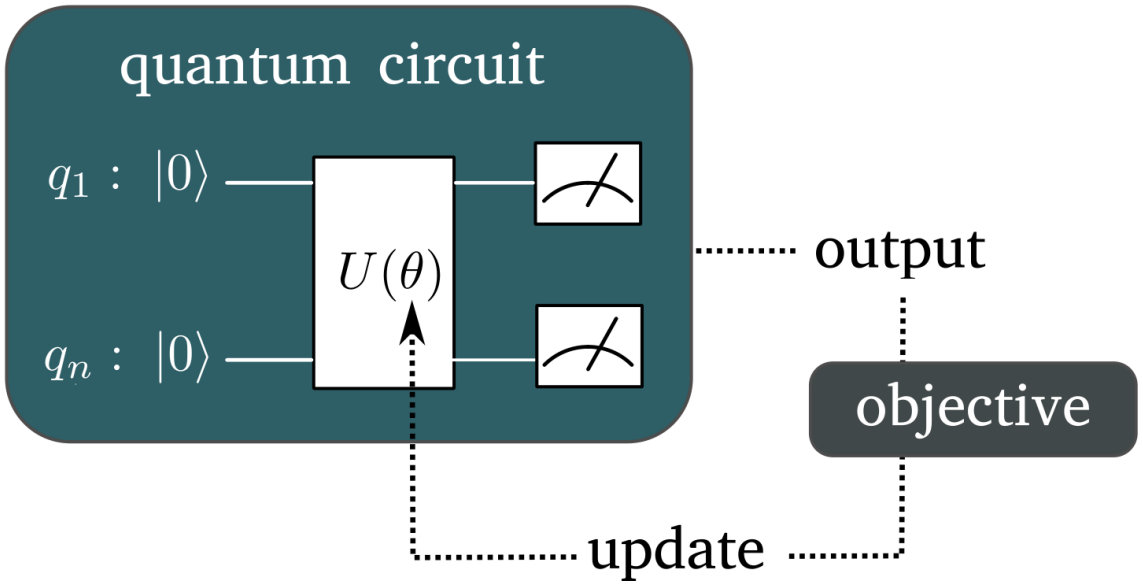
QUANTUM ALGORITHMS CRASH COURSE

NEAR-TERM HYBRID ALGORITHMS

“Near-Term”: make use of the small noisy quantum devices we will have access to in the near-term

“Hybrid”: since the quantum devices are limited, let’s have them work alongside classical devices

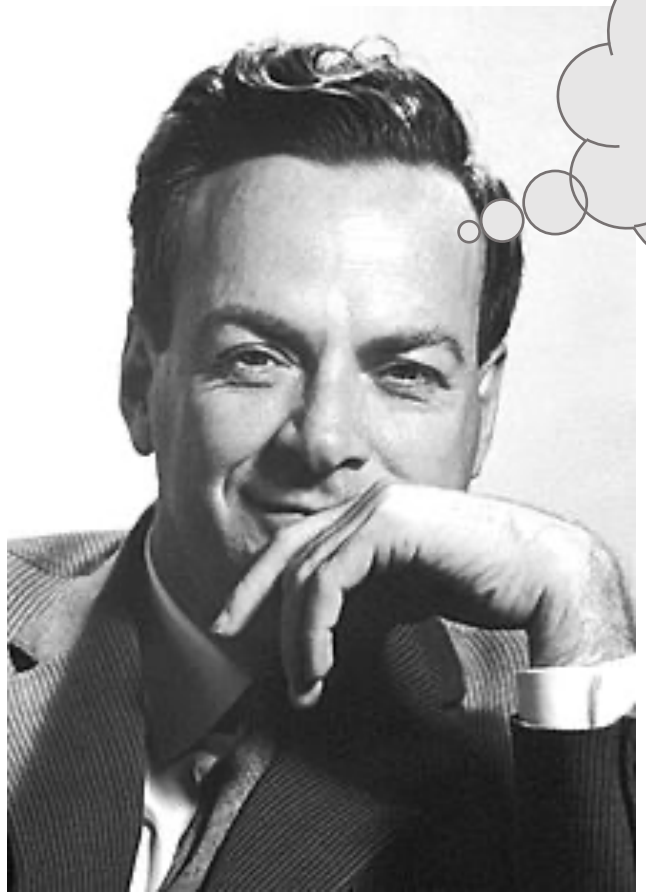
Current near-term hybrid quantum algorithms research is primarily focused on ***variational quantum circuits***...



Key Idea: *The gates in a quantum circuit correspond to rotations, so we can parametrize the angles of rotation. A classical computer can then be used to optimize the values of these parameters, to design a good quantum circuit!*

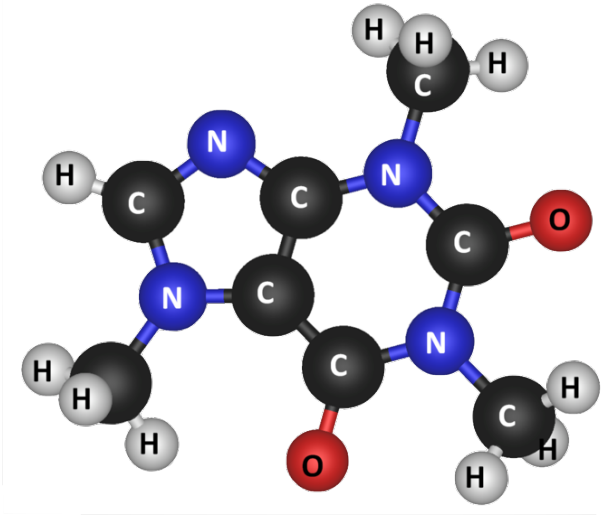
Let’s look at some examples of ***current*** use cases for these near-term algorithms...

QUANTUM SIMULATION



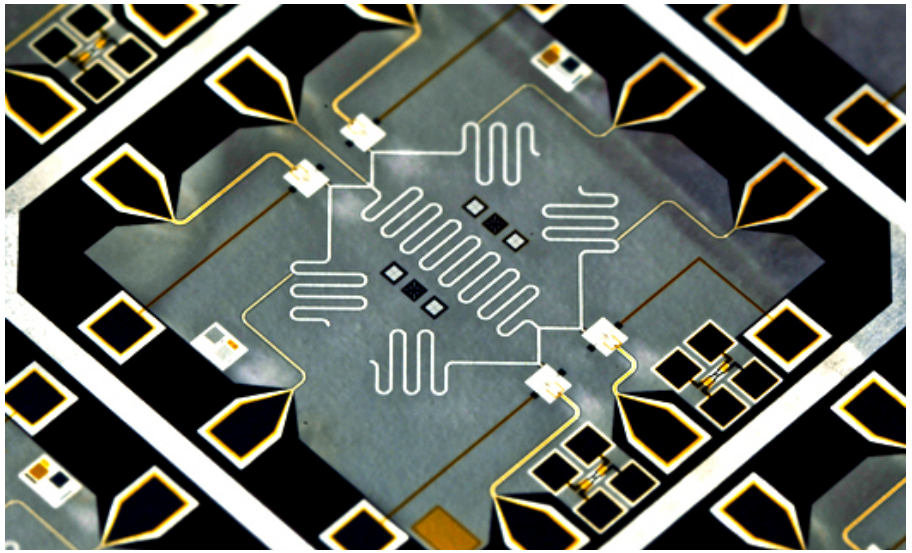
Let's simulate quantum on quantum!

Richard Feynman



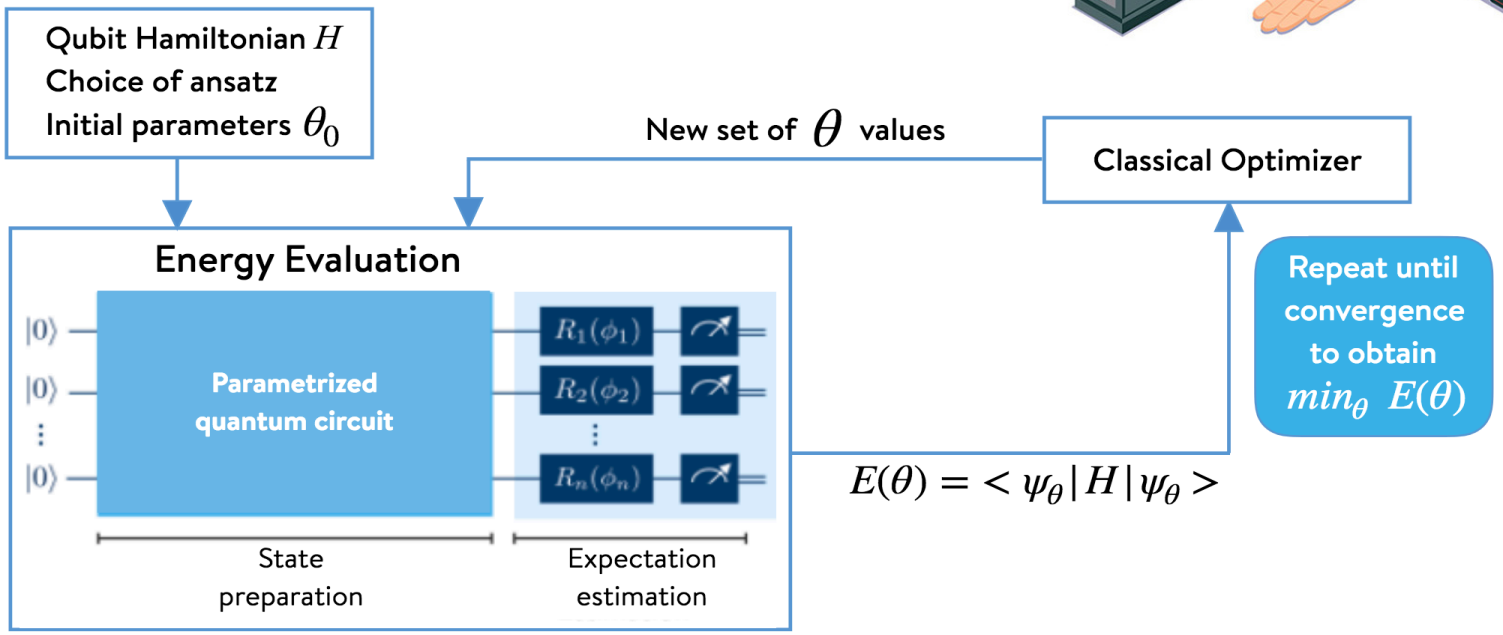
Real Atoms

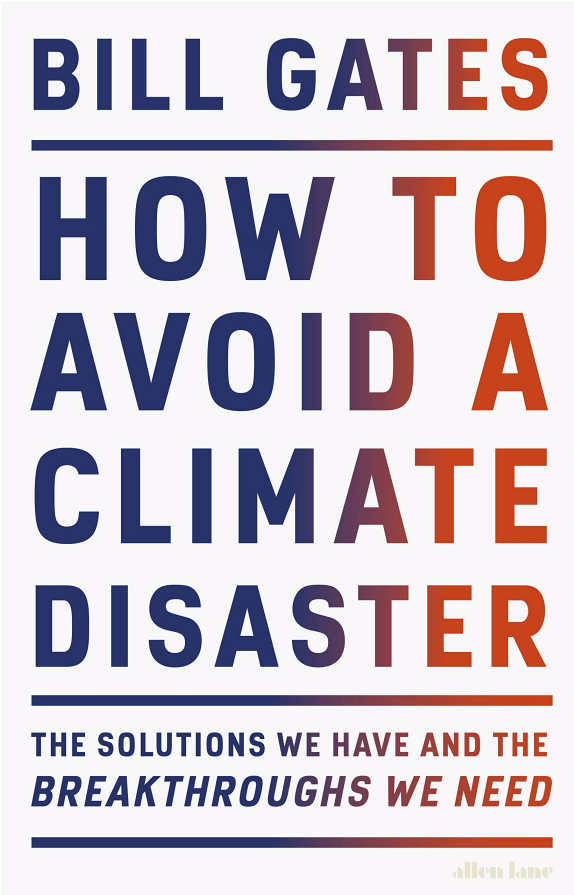
"Artificial Atoms"



ENVIRONMENTAL SCIENCE

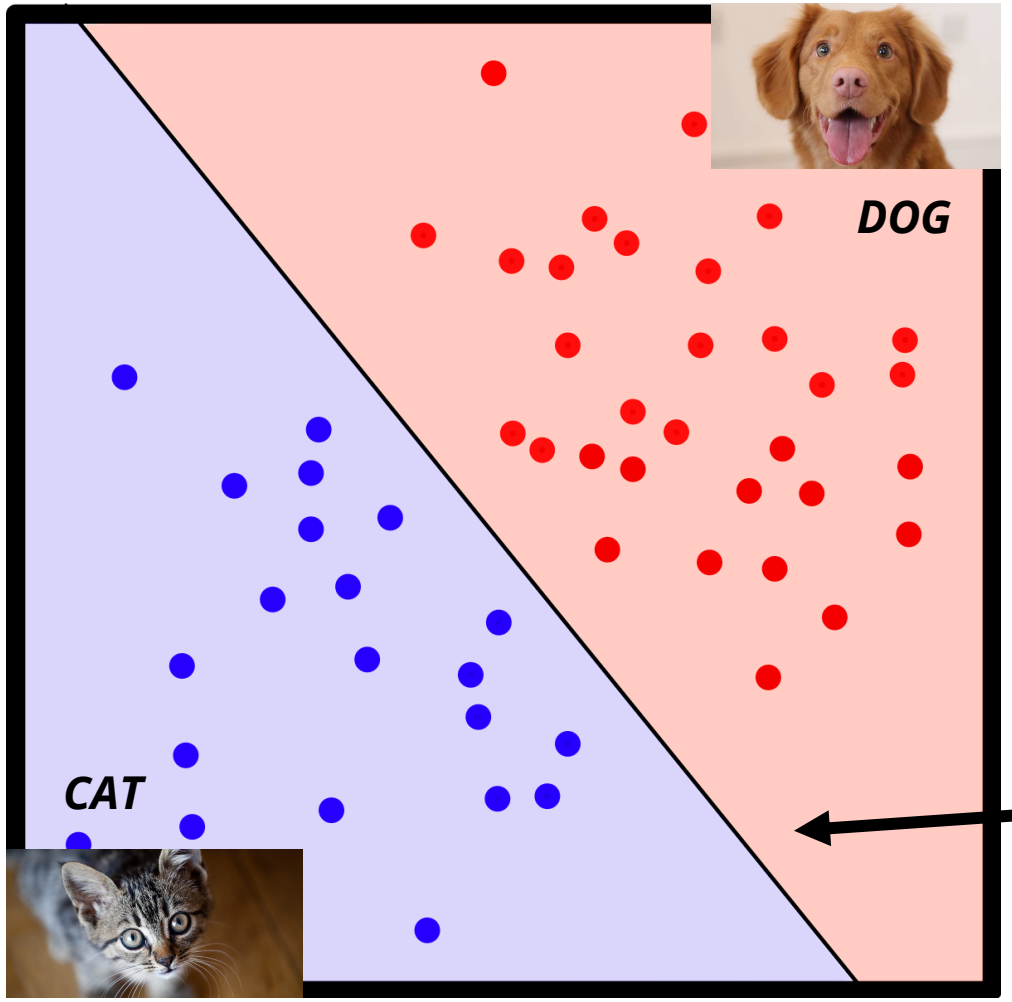
The **Variational Quantum Eigensolver (VQE)** algorithm is already being used to solve several problems in quantum chemistry and material science, such as finding the ground state energy of a molecular system.





MACHINE LEARNING

Classification in a nutshell...



LINEARLY SEPARABLE

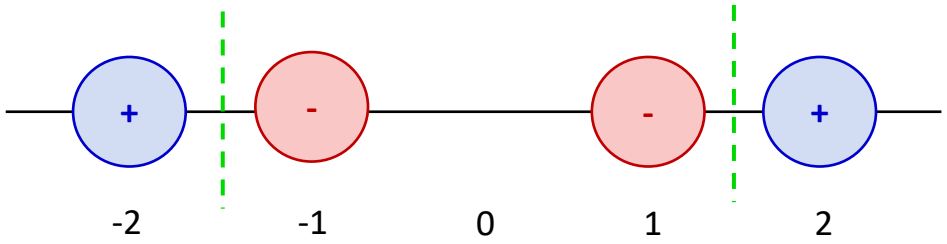
(can be separated with a straight line)

MACHINE LEARNING

Typically data is not linearly separable and it is hard to make a function to perform classification.

However, often we can apply functions to our data to make it linearly separable.

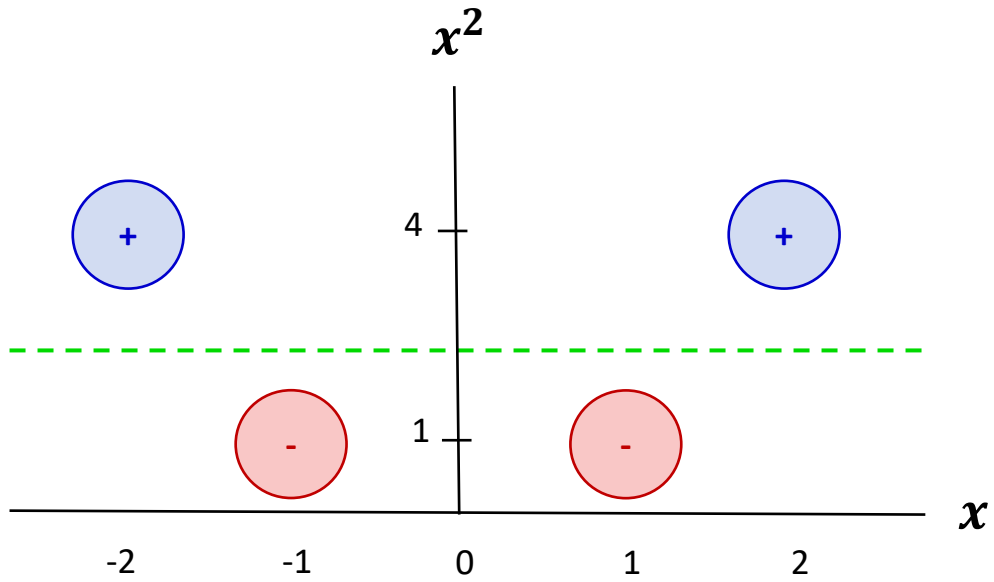
This is known as a **feature mapping**...



*The data **is not** linearly separable.*

$$\phi(x):\ x \rightarrow (x, x^2)$$

*Apply a **feature mapping** to the data.*



*The data **is** linearly separable.*

QUANTUM MACHINE LEARNING

Computationally finding an optimal feature mapping for a dataset is really hard!

Variational quantum algorithms have been proposed to learn powerful feature mappings!

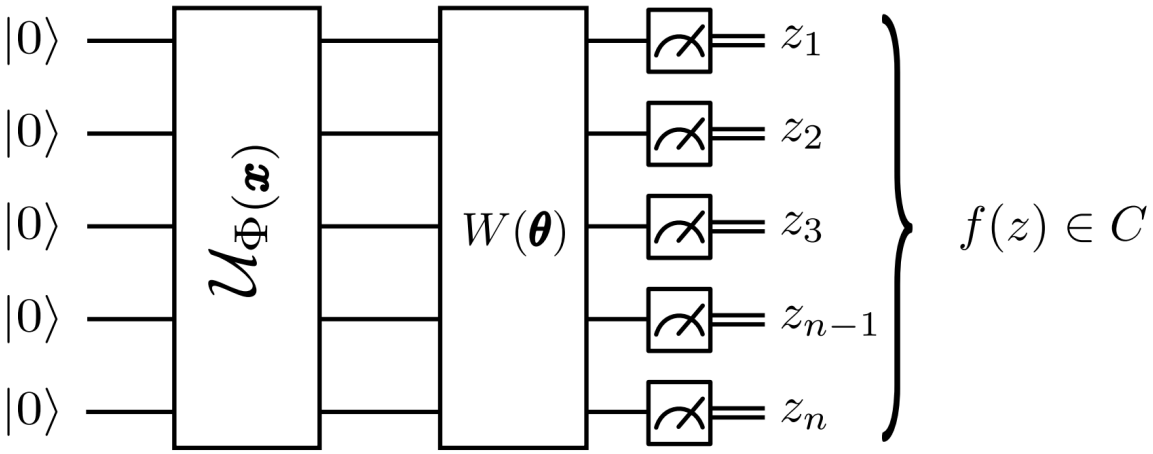
Letter | Published: 13 March 2019

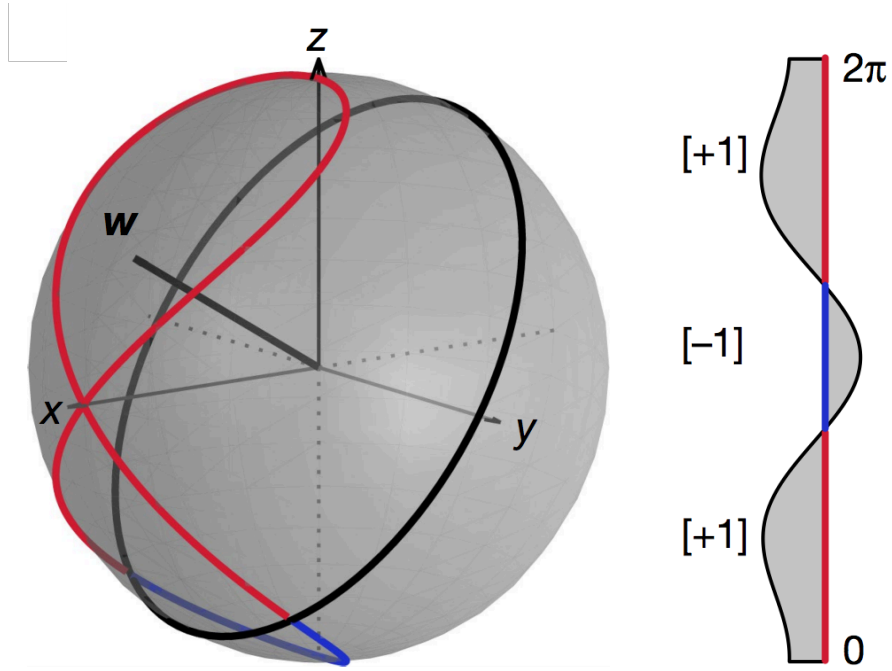
Supervised learning with quantum-enhanced feature spaces

Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow & Jay M. Gambetta

Nature **567**, 209–212(2019) | Cite this article

26k Accesses | **134** Citations | **261** Altmetric | Metrics





A quantum feature mapping and classification!

THE QUANTUM ALGOS LANDSCAPE

Deutsch-Josza

First theoretical demonstration of quantum advantage!

Uses an Oracle

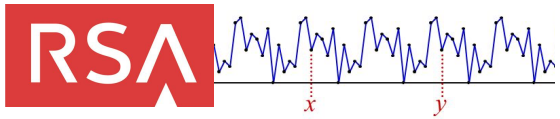
$$O(1) \ll O(2^n)$$

Exponential Quantum Speedup!

QUANTUM ADVANTAGE

Shor's Algorithm

Super-polynomial speedup for factoring using the QFT!



Cracking RSA Requires Factoring & Period Finding

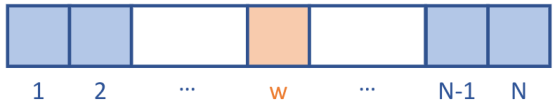
The Quantum Fourier Transform Encodes Frequency in Phase

$$O(\log(n)^3) \ll O(n^{1.9})$$

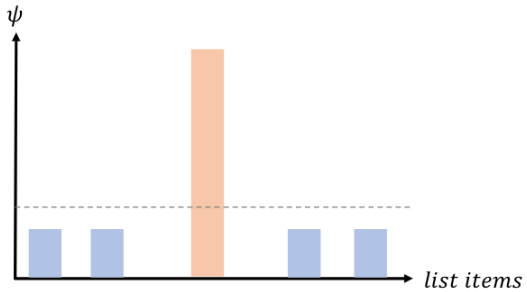
Super-Polynomial Quantum Speedup!

Grover Search

Quadratic speedup for search using amplitude amplification!



Unstructured Search



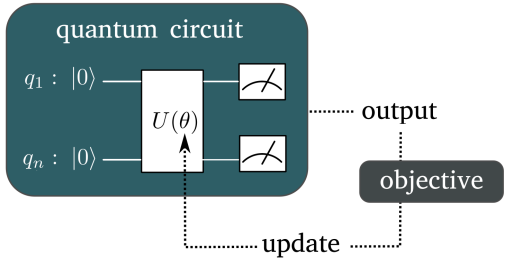
Leverages Amplitude Amplification

$$O(\sqrt{n}) \ll O(n)$$

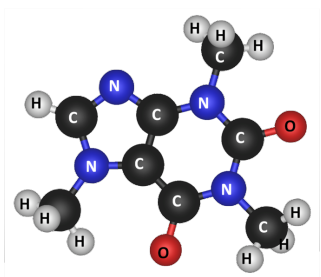
Quadratic Quantum Speedup!

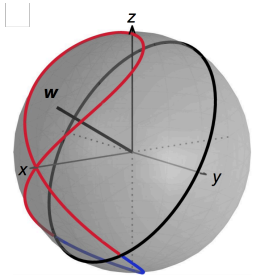
Near-Term Algos

Applications of noisy, small available quantum devices!



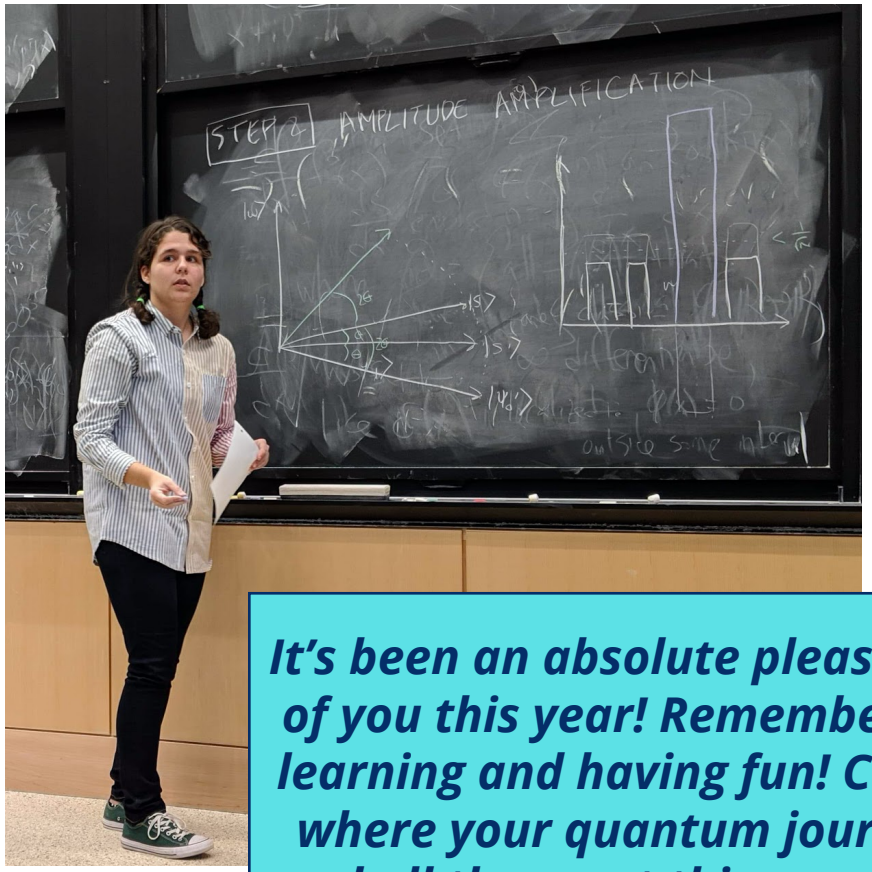
Hybrid Quantum-Classical Algos





Using Quantum to Solve Important Problems!

PEACE OUT, QUANTUM HOMIES



It's been an absolute pleasure teaching all of you this year! Remember to never stop learning and having fun! Can't wait to see where your quantum journeys take you, and all the great things you will achieve...

- Fran