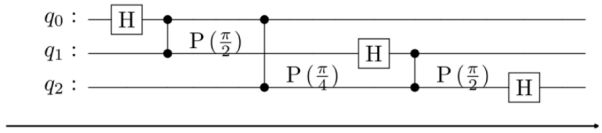
It's Elementary: Meet Quantum Computing

Outcome	Binary	Amplitude	Direction	Amplitude Bar
0	000	0.71	0.00	
1	001	0.00		
2	010	0.00		
3	011	0.00		
4	100	-0.65 - 0.27i	-157.50	
5	101	0.00 - 0.00i		
6	110	0.00 - 0.00i		
7	111	0.00 - 0.00i		



Outcome	Binary	Amplitude	Direction	Amplitude Bar
0	000	0.02 - 0.10i	-78.75	
1	001	0.15 - 0.23i	-56.25	
2	010	0.35 - 0.23i	-33.75	
3	011	0.48 - 0.10i	-11.25	
4	100	0.48 + 0.10i	11.25	
5	101	0.35 + 0.23i	33.75	
6	110	0.15 + 0.23i	56.25	
7	111	0.02 + 0.10i	78.75	

When explaining new ideas, it's helpful to start with what people already know. But what common knowledge is useful in learning quantum computing? Many think you need to study high-level linear algebra before understanding quantum computing's strange nature. This isn't true. Basic math (like binary numbers, trigonometry, and probabilities) is enough. By implementing quantum computing concepts, you can gain a deeper understanding of them, and you can even create a simple quantum simulator using just a few lines of code.

Quantum computers have two main features that regular computers don't: quantum parallelism and quantum measurement. Classical computers have limited ways of parallel computing (like vector processing and SIMD), which are like quantum parallelism, but only on a small scale. They can also take samples from probability distributions, similar to quantum measurement, but again, they can only do this efficiently on a small scale.

Building quantum computers needs deep knowledge of quantum physics, but you don't need to know this to use them. The same goes for traditional computers. This introduction shows that quantum computing is actually quite easy to understand.

From Classical To Quantum Systems

You're used to using numbers to understand systems that matter to you, like the stuff in your fridge, your account balances, or your investment portfolio. In portfolios, percentages have to add up to 100, and you can change your allocations by moving money between investments.

Quantum systems also have attributes with values that are constrained as a total, and you can change them with quantum transformations, which essentially move values between attributes.

The big difference is what happens when we observe (measure) a quantum system's state. The system's state collapses into one with just one attribute, like putting all your investments into one instrument. We only see one attribute at random, with a likelihood that depends on its value. This is the only strange thing about quantum systems, but once you understand it, it is not hard to use it.

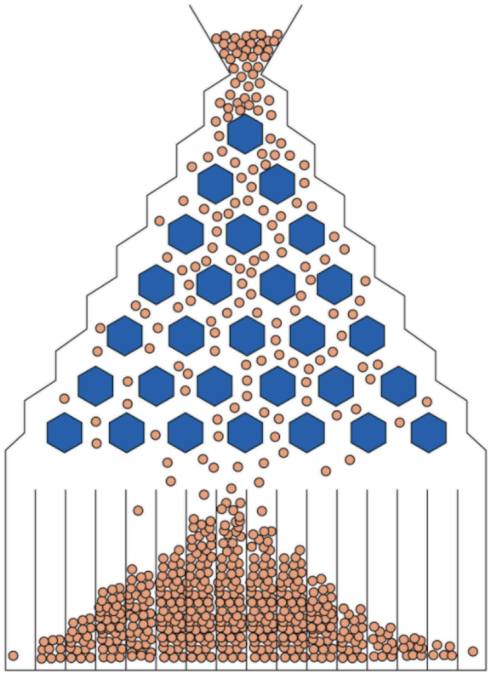
Philosophers of science often engage in discussions about the existence of these values prior to observation. When it comes to designing quantum programs, these values are relevant, as they must be taken into consideration. Therefore, from the perspective of quantum program developers, these values are considered to exist. However, they are not stored anywhere, like we are used to values in classical computing. They are similar to attributes like the speed or the rotations per minute desribing the state of a car. We know how to change and measure them without directly writing and reading them.

Quantum Programs, Outcomes and Probabilities

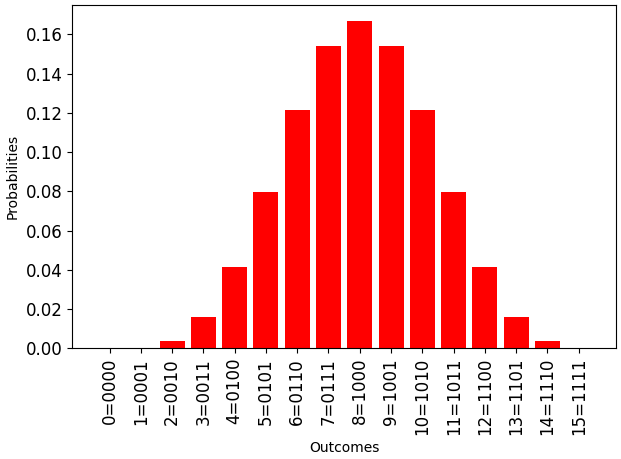
In a quantum computation, the state of the underlying quantum system is altered, and there are various possible outcomes, each with a particular probability. These potential outcomes are attributes of the quantum system's state.

When executing a quantum program on a quantum computer, a single outcome is observed upon measuring the state of the quantum system. To obtain valuable information, the same program must be run multiple times, and the results of each run must be collected. A single execution is commonly referred to as a shot, while the number of times an outcome is observed is called a count.

The behavior of a quantum program, when observed multiple times, is analogous to a Galton board, also known as a bean machine. This device consists of a vertical board with rows of pegs that are alternately positioned. Beads dropped from the top of the board bounce left or right as they encounter the pegs, ultimately landing in bins at the bottom. The resulting distribution of accumulated bead columns in these bins closely resembles a bell curve. By adjusting the arrangement of the pegs, one can generate different probability distributions.



By converting the counts into percentages of the total number of runs, a probability histogram can be created, with each outcome being assigned a percentage based on its count, sometimes called a quasi-probability.

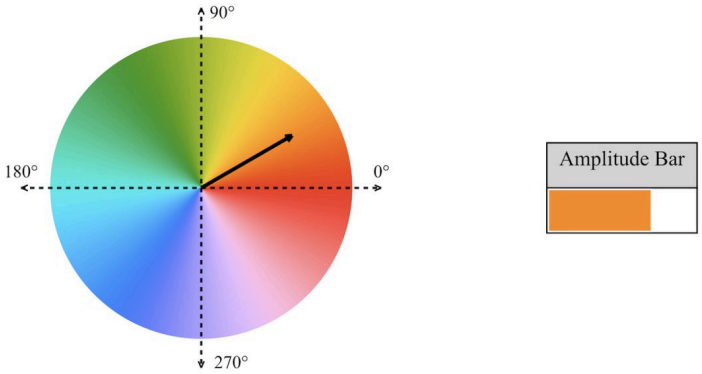


Quantum Probabilities Are More Complex

While we can only count outcomes of a quantum computation and approximate their probabilities, the values comprising a quantum state (one per possible outcome) are not straight probabilities. They are complex numbers, called probability amplitudes, or simply amplitudes. We can think of an amplitude as an arrow, or force, having a magnitude and a direction. The probability of an outcome is actually the squared magnitude of its amplitude. This is one of the fundamental facts of quantum theory that we just accept without trying to explain.

Depending on the context, it is useful to visualize an amplitude (complex number) in two ways:

1. An arrow starting from the origin of the plane and pointing to the corresponding point of the complex number.
2. A colored bar with a length equal to the magnitude (absolute value) of the complex number, and a color determined by the phase (direction) of the complex number, as depicted on the color wheel below.



The amplitudes comprising the state of a qubit system cannot have magnitudes larger than 1, since the squared magnitudes are probabilities.

Complex Number Refresher

This is the most advanced math needed in this article. Recall that a complex number has an algebraic form:

$$z = a + ib, \tag{1}$$

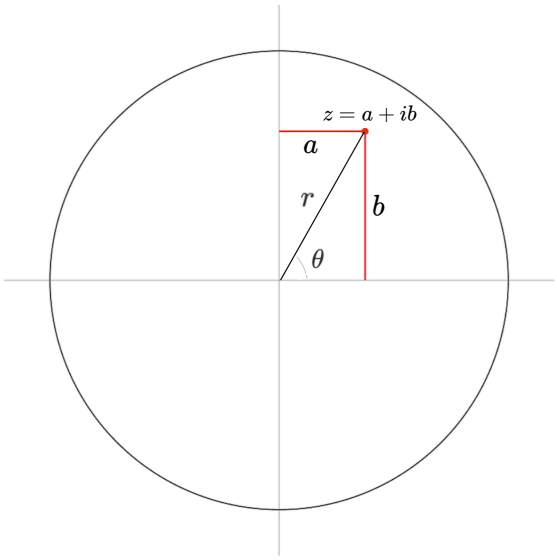
and a polar form

$$z = r(\cos\theta + i\sin\theta), \tag{2}$$

Where *i* is the imaginary unit, with the property

$$i^2 = -1. \tag{3}$$

A complex number corresponds to a point in the plane.



We can convert from the algebraic to the polar using simple formulas:

$$r = \sqrt{a^2 + b^2} \tag{4}$$
$$\theta = \operatorname{atan2}(b, a) \tag{5}$$

Quantum State

As mentioned, each possible outcome of a quantum computation has a probability amplitude associated to it, and the probability of an outcome is the squared magnitude of its corresponding amplitude. The sum of the probabilities of all possible outcomes add up to 1.

We can use table "comprehensions" (term borrowed from Python's list comprehensions) to visualize quantum states algebraically.

Outcome	Amplitude
k	$a_k + ib_k$

Such a comprehension table can be seen as a bridge between formal math and code. Python code to express the same state looks like this:

```
state = [a[k] + 1j*b[k] for k in range(N)]
```

You don't need to understand the following formula, but this is how a quantum state is typically expressed mathematically:

$$|\psi\rangle = \sum_{k=0}^{N} (a_k + ib_k)|k\rangle \tag{6}$$

In our experience, using tables is very effective in expressing quantum states, for all audiences.

We can use the polar form in tables as well:

Outcome	Direction	Probability
k	θ_k	p_k

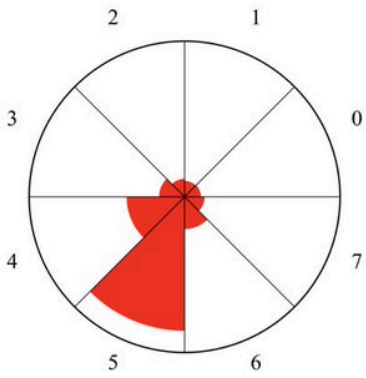
It's sometimes useful to include more information in table rows, to get a deeper understanding of a quantum state:

Outcome	Amplitude	Direction	Magnitude	Probability
k	$a_k + ib_k$	$\text{atan2}(b_k, a_k)\frac{180}{\pi}$	$\sqrt{a_k^2 + b_k^2}$	$a_k^2 + b_k^2$

State Tables and Rose Charts

Comprehension tables are used to show formulas or expressions in quantum states. For a concrete state, we can use a state table with one row of expanded information for each possible outcome of a quantum computation, or a simpler polar area diagram where the outcomes are represented by circle sectors, and their probabilities are sub-sectors whose area is proportional to the probability.

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar	Probability	Probability Bar
0	000	0.0986 + 0.0364i	20.2510°	0.1051		0.0110	
1	001	0.0748 + 0.0691i	42.7476°	0.1018		0.0104	
2	010	0.0485 + 0.1053i	65.2525°	0.1159		0.0134	
3	011	0.0064 + 0.1632i	87.7510°	0.1633		0.0267	
4	100	-0.1290 + 0.3495i	110.2502°	0.3726		0.1388	
5	101	0.5840 - 0.6318i	-47.2500°	0.8604		0.7403	
6	110	0.1880 - 0.0867i	-24.7510°	0.2070		0.0428	
7	111	0.1287 - 0.0051i	-2.2520°	0.1288		0.0166	



These two representations are versions of a bar plot, and can be preferred in different contexts.

Qubits

A qubit is a digit in the binary representation of outcomes. A quantum system with a single qubit has two possible outcomes, 0 or 1. A two-qubit system has four possible outcomes, consisting of all two-digit binary numbers:

$$0 = 00_2 \tag{7}$$
$$1 = 01_2 \tag{8}$$
$$2 = 10_2 \tag{9}$$
$$3 = 11_2 \tag{10}$$

Similarly, a three-qubit system has eight possible outcome, consisting of all three-digit binary numbers:

$$0 = 000_2 \tag{11}$$
$$1 = 001_2 \tag{12}$$
$$2 = 010_2 \tag{13}$$
$$3 = 011_2 \tag{14}$$
$$4 = 100_2 \tag{15}$$
$$5 = 101_2 \tag{16}$$
$$6 = 110_2 \tag{17}$$
$$7 = 111_2 \tag{18}$$

In general, an *n*-qubit quantum system supports quantum computations with

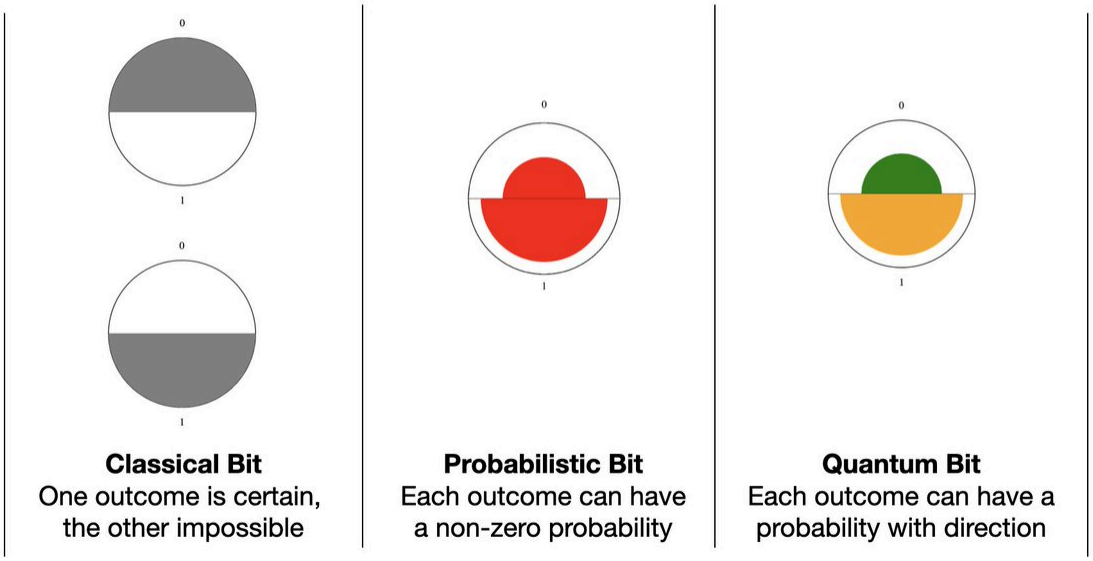
$$N = 2^n \tag{19}$$

outcomes.

One-Qubit Systems

How do we explain the difference between a classical and a quantum bit? A quantum bit, or better said, a one-qubit system, has a probability with direction for each of the two possible outcomes, while in a classical bit, exactly one outcome is possible. We represent outcomes with circle sectors, and directions with colors on a color wheel. For two outcomes, the sectors are semi-circles. Note that the comparison works only for one-(qu)bit systems, not for a (qu)bit in a larger system. Many comparisons in the quantum computing literature are incorrect or incomplete, especially those saying that a qubit can have any value between 0 and 1.

Classical vs Probabilistic vs Quantum Bits



Single-Qubit Gates

We visualize the effect of applying quantum gates using table comprehensions, state tables, and color wheels.

The X-gate, also called the NOT gate, swaps the amplitudes corresponding to a pair of outcomes.

X Gate

Matrix: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Table comprehension:

Outcome	Amplitude
0	z_0
1	z_1

$\xrightarrow{X}$

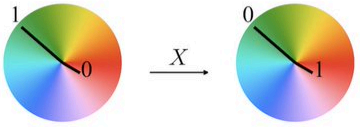
Outcome	Amplitude
0	z_1
1	z_0

Visual example:

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	-0.7073 + 0.6160i	138.9°	0.9379	

$\xrightarrow{X}$

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	-0.7073 + 0.6160i	138.9°	0.9379	
1	1	0.2994 - 0.1750i	-30.3°	0.3468	



Applying a Y-gate to a single-qubit state changes both the directions and swaps the probabilities, as shown in the diagram.

Y Gate

Matrix: $\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$

Table comprehension:

Outcome	Amplitude
0	z_0
1	z_1

$\xrightarrow{Y}$

Outcome	Amplitude
0	$-iz_1$
1	iz_0

Outcome	Direction	Probability
0	θ_0	p_0
1	θ_1	p_1

$\xrightarrow{Y}$

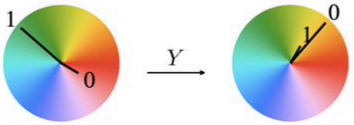
Outcome	Direction	Probability
0	$\theta_1 - 90°$	p_1
1	$\theta_0 + 90°$	p_0

Visual example:

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	-0.7073 + 0.6160i	138.9°	0.9379	

$\xrightarrow{Y}$

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.6160 + 0.7073i	48.9°	0.9379	
1	1	0.1750 + 0.2994i	59.7°	0.3468	



This gate multiplies the 1-side of a pair of amplitudes by -1. It changes the signs of both the real and imaginary parts of the amplitude of the 1-side of the pair.

Z Gate

Matrix: $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Table comprehension:

Outcome	Amplitude
0	z_0
1	z_1

$\xrightarrow{Z}$

Outcome	Amplitude
0	z_0
1	$-z_1$

Outcome	Direction	Probability
0	θ_0	p_0
1	θ_1	p_1

$\xrightarrow{Z}$

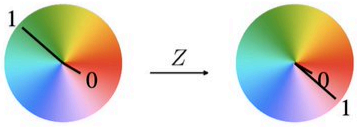
Outcome	Direction	Probability
0	θ_0	p_0
1	$180° + \theta_1$	p_1

Visual example:

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	-0.7073 + 0.6160i	138.9°	0.9379	

$\xrightarrow{Z}$

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	0.7073 - 0.6160i	-41.1°	0.9379	



The Hadamard gate (or H-gate), named after the famous mathematician Jacques Hadamard, replaces an amplitude pair with their sum and difference divided by the square root of 2.

Hadamard Gate

Matrix: $\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$

Table comprehension:

Outcome	Amplitude
0	z_0
1	z_1

$\xrightarrow{H}$

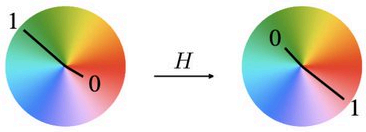
Outcome	Amplitude
0	$\frac{1}{\sqrt{2}}(z_0 + z_1)$
1	$\frac{1}{\sqrt{2}}(z_0 - z_1)$

Visual example:

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	-0.7073 + 0.6160i	138.9°	0.9379	

$\xrightarrow{H}$

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	-0.2884 + 0.3119i	132.8°	0.4248	
1	1	0.7119 - 0.5593i	-38.2°	0.9053	



The Phase gate rotates the 1-side of a pair of amplitudes by a given angle.

Phase Gate

Matrix: $P(\varphi)$

$$\begin{bmatrix}1 & 0\\ 0 & \cos\varphi + i\sin\phi\end{bmatrix}$$

Table comprehension:

Outcome	Amplitude
0	z_0
1	z_1

$\xrightarrow{P(\varphi)}$

Outcome	Amplitude
0	z_0
1	$\mathrm{cis}(\varphi)z_1$

Outcome	Direction	Probability
0	θ_0	p_0
1	θ_1	p_1

$\xrightarrow{P(\varphi)}$

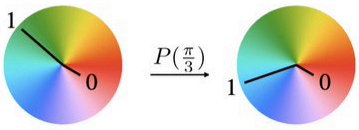
Outcome	Direction	Probability
0	θ_0	p_0
1	$\varphi + \theta_1$	p_1

Visual example:

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	-0.7073 + 0.6160i	138.9°	0.9379	

$\xrightarrow{P(\frac{\pi}{3})}$

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	-0.8871 - 0.3045i	-161.1°	0.9379	



For a given angle θ, the RZ(θ)-gate rotates the 0-side of a pair of amplitudes counterclockwise by θ/2, and the 1-side of a pair of amplitudes clockwise by θ/2.

RZ Gate

Matrix: $R_Z(\theta)$

$$\begin{bmatrix} \cos\frac{\theta}{2} - i\sin\frac{\theta}{2} & 0 \\ 0 & \cos\frac{\theta}{2} + i\sin\frac{\theta}{2} \end{bmatrix}$$

Table comprehension:

Outcome	Amplitude
0	z_0
1	z_1

$\xrightarrow{R_Z(\theta)}$

Outcome	Amplitude
0	$\mathrm{cis}(-\frac{\theta}{2})z_0$
1	$\mathrm{cis}(\frac{\theta}{2})z_1$

Outcome	Direction	Probability
0	θ_0	p_0
1	θ_1	p_1

$\xrightarrow{R_Z(\theta)}$

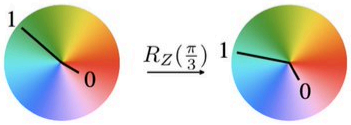
Outcome	Direction	Probability
0	$\theta_0 - \frac{\theta}{2}$	p_0
1	$\theta_1 + \frac{\theta}{2}$	p_1

Visual example:

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.2994 - 0.1750i	-30.3°	0.3468	
1	1	-0.7073 + 0.6160i	138.9°	0.9379	

$\xrightarrow{R_Z(\frac{\pi}{3})}$

Outcome	Binary	Amplitude	Direction	Magnitude	Amplitude Bar
0	0	0.1718 - 0.3013i	-60.3°	0.3468	
1	1	-0.9205 + 0.1798i	168.9°	0.9379	



It turns out we only need two single-qubit gates: Hadamard and Phase. The rest can be built from them. You basically need to recombine pairs of magnitudes somehow, and freedom to rotate amplitudes. And you also need CX (or CNOT), a two-qubit gate, for entanglement, to restrict the vast impact of gates on amplitudes.

Quantum Transformations

An elementary quantum transformation is a single-qubit gate applied to a target qubit.

We can visualize and explain quantum state evolution with elementary quantum gates using the important, but often ignored fact that amplitudes change in pairs, with the pairing depending on the target of a gate application. If the target qubit is *t*, two outcomes are paired if their binary representations differ only in the digit *t*. This means that their difference is

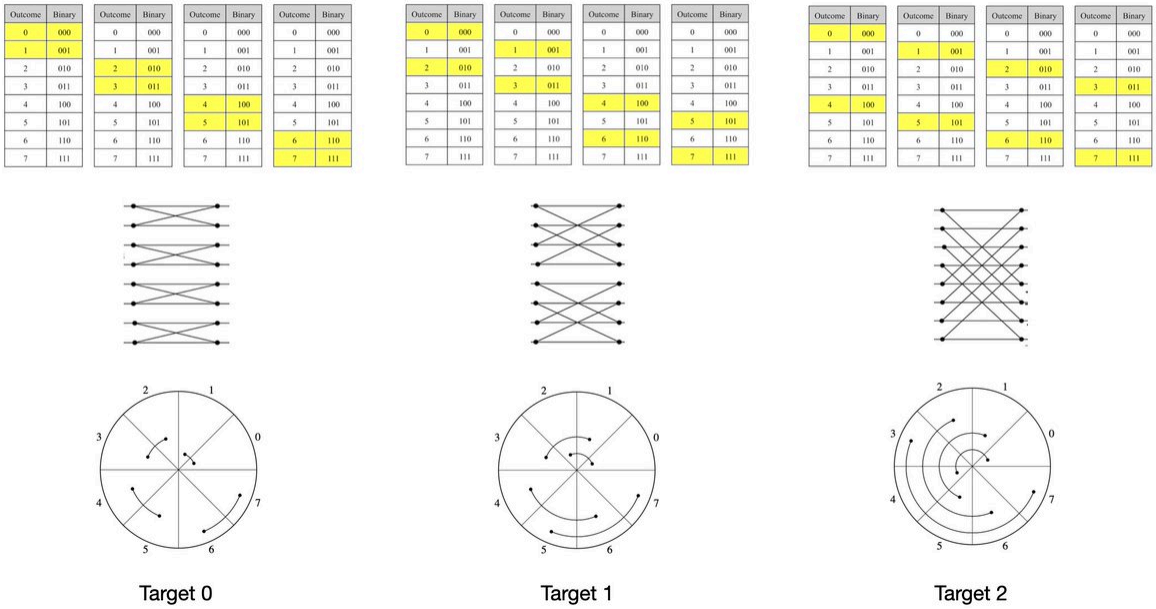
$$2^t. \tag{20}$$

We can represent the pairing of outcomes and amplitudes with two generic rows in a table:

Outcome	Amplitude
k_0	z_{k_0}
$k_1 = k_0 + 2^t$	z_{k_1}

We can use state tables and polar (circular) representations, but we can also reference the "butterfly" diagrams used in the Fast Fourier Transform literature. In fact, the process of applying an elementary quantum gate is virtually the same as a stage in the (most common implementation of the) Fast Fourier Transform algorithm.

Amplitudes (probabilities) change in pairs selected by target



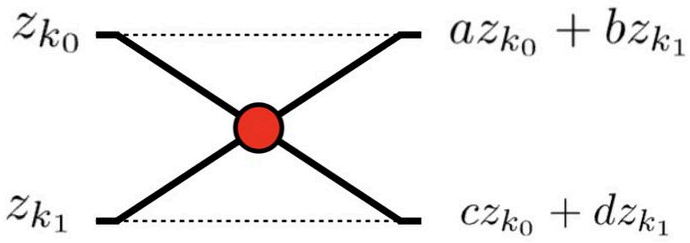
Each gate comes with its own formula, which is applied to all pairs of amplitudes.

The same formula is applied to all pairs

Outcome	Amplitude
k_0	z_{k_0}
$k_1 = k_0 + 2^t$	z_{k_1}

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Outcome	Amplitude
k_0	$az_{k_0} + bz_{k_1}$
$k_1 = k_0 + 2^t$	$cz_{k_0} + dz_{k_1}$



As an example, let's look at a visualization of the effect of applying the X-gate on qubit 0 of a three-qubit system. The amplitudes in each pair are swapped. For other gates, a different formula is applied to recombine the amplitudes in each pair, but the pairing mechanism is the same.

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0748 + 0.0691i	
1	001	0.0986 + 0.0364i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0748 + 0.0691i	
1	001	0.0986 + 0.0364i	
2	010	0.0064 + 0.1632i	
3	011	0.0485 + 0.1053i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0748 + 0.0691i	
1	001	0.0986 + 0.0364i	
2	010	0.0064 + 0.1632i	
3	011	0.0485 + 0.1053i	
4	100	0.5840 - 0.6318i	
5	101	-0.1290 + 0.3495i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0748 + 0.0691i	
1	001	0.0986 + 0.0364i	
2	010	0.0064 + 0.1632i	
3	011	0.0485 + 0.1053i	
4	100	0.5840 - 0.6318i	
5	101	-0.1290 + 0.3495i	
6	110	0.1287 - 0.0051i	
7	111	0.1880 - 0.0867i	

Here is the visualization for target qubit 1.

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0485 + 0.1053i	
1	001	0.0748 + 0.0691i	
2	010	0.0986 + 0.0364i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0485 + 0.1053i	
1	001	0.0064 + 0.1632i	
2	010	0.0986 + 0.0364i	
3	011	0.0748 + 0.0691i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0485 + 0.1053i	
1	001	0.0064 + 0.1632i	
2	010	0.0986 + 0.0364i	
3	011	0.0748 + 0.0691i	
4	100	0.1880 - 0.0867i	
5	101	0.1287 - 0.0051i	
6	110	-0.1290 + 0.3495i	
7	111	0.5840 - 0.6318i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0485 + 0.1053i	
1	001	0.0064 + 0.1632i	
2	010	0.0986 + 0.0364i	
3	011	0.0748 + 0.0691i	
4	100	0.1880 - 0.0867i	
5	101	0.1287 - 0.0051i	
6	110	-0.1290 + 0.3495i	
7	111	0.5840 - 0.6318i	

And the visualization for target qubit 2.

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	-0.1290 + 0.3495i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	0.0986 + 0.0364i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	-0.1290 + 0.3495i	
1	001	0.5840 - 0.6318i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	0.0986 + 0.0364i	
5	101	0.0748 + 0.0691i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	-0.1290 + 0.3495i	
1	001	0.5840 - 0.6318i	
2	010	0.1880 - 0.0867i	
3	011	0.0064 + 0.1632i	
4	100	0.0986 + 0.0364i	
5	101	0.0748 + 0.0691i	
6	110	0.0485 + 0.1053i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	-0.1290 + 0.3495i	
1	001	0.5840 - 0.6318i	
2	010	0.1880 - 0.0867i	
3	011	0.1287 - 0.0051i	
4	100	0.0986 + 0.0364i	
5	101	0.0748 + 0.0691i	
6	110	0.0485 + 0.1053i	
7	111	0.0064 + 0.1632i	

Controlled Transformations

A gate-based quantum transformation applies a quantum gate to a target qubit and has a number of control qubits (different from the target). The same gate formula is used to process amplitude pairs, but only the paired outcomes that have 1 in the control positions get their amplitudes processed. For example, in a three-qubit system, if the middle qubit is a control qubit, the outcomes that have 0 as the middle digit in their binary representation are left unchanged. They are shown grayed out in the image that shows the pairs corresponding to target qubit 0. The paired outcomes differ only in the last digit (qubit 0).

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0064 + 0.1632i	
3	011	0.0485 + 0.1053i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0064 + 0.1632i	
3	011	0.0485 + 0.1053i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1287 - 0.0051i	
7	111	0.1880 - 0.0867i	

Here is the visualization for target 2, control 1.

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.1880 - 0.0867i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.0485 + 0.1053i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.1880 - 0.0867i	
3	011	0.1287 - 0.0051i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.0485 + 0.1053i	
7	111	0.0064 + 0.1632i	

Here is the visualization for target 0, control 2.

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	0.5840 - 0.6318i	
5	101	-0.1290 + 0.3495i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	0.5840 - 0.6318i	
5	101	-0.1290 + 0.3495i	
6	110	0.1287 - 0.0051i	
7	111	0.1880 - 0.0867i	

Here is the visualization for target 1, control 2.

Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	-0.1290 + 0.3495i	
5	101	0.5840 - 0.6318i	
6	110	0.1880 - 0.0867i	
7	111	0.1287 - 0.0051i	

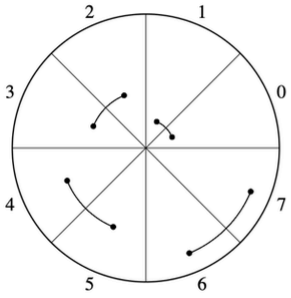
Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	0.1880 - 0.0867i	
5	101	0.5840 - 0.6318i	
6	110	-0.1290 + 0.3495i	
7	111	0.1287 - 0.0051i	

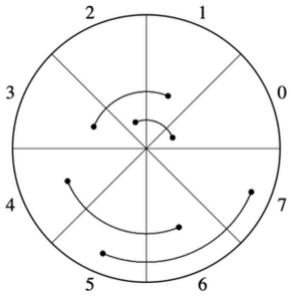
Outcome	Binary	Amplitude	Amplitude Bar
0	000	0.0986 + 0.0364i	
1	001	0.0748 + 0.0691i	
2	010	0.0485 + 0.1053i	
3	011	0.0064 + 0.1632i	
4	100	0.1880 - 0.0867i	
5	101	0.1287 - 0.0051i	
6	110	-0.1290 + 0.3495i	
7	111	0.5840 - 0.6318i	

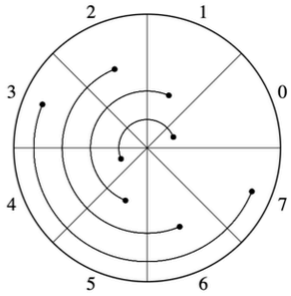
Each control qubit cuts the number of affected pairs in half. This is great for a simulator, but expensive on real quantum computers.

Quantum Parallelism

The behavior of quantum gates is a fascinating blend of the familiar and the unexpected. Just as we're accustomed to transfers between investments, quantum gates act similarly, transferring amplitudes (balances) between outcomes (accounts). However, here's where the twist lies: unlike in our everyday world, these "transfers" occur simultaneously, governed by the same mathematical formula. This is called quantum parallelism.







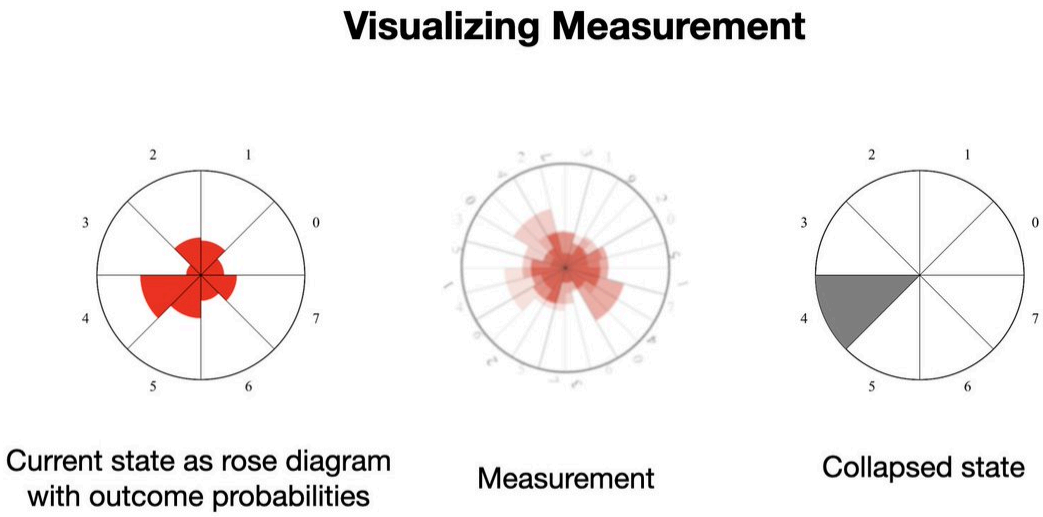
This simultaneous operation is the secret sauce of quantum computing, offering unparalleled advantages in certain applications. Yet, this very feature also restricts the types of problems that can benefit from quantum magic.

Superposition And Entanglement

These quantum mechanics concepts are crucial to the implementation of quantum computers, but are not directly relevant to quantum computing. Quantum superposition expresses the uncertainty of what outcome will be observed in a measurement. Quantum entanglement refers to the fact that individual qubit measurements are not always independent of each other. It is the implementation mechanism for controlled transformations.

Quantum Measurement

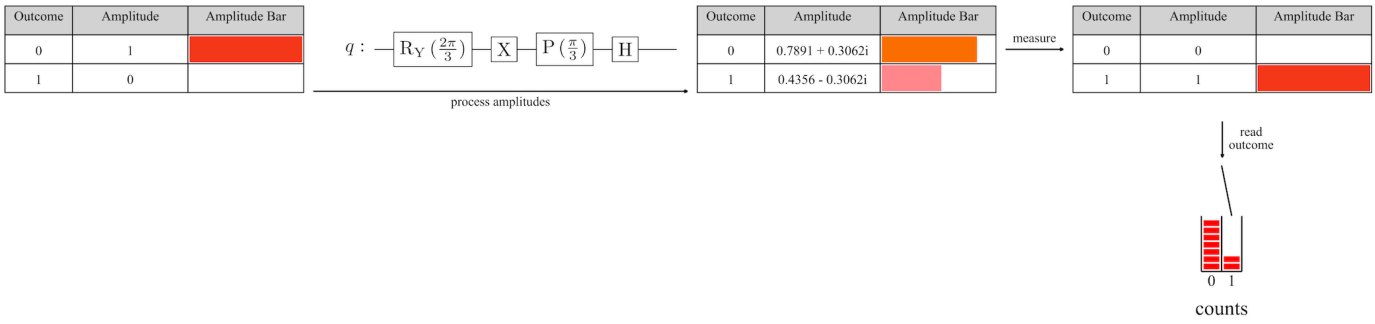
How can we visualize the measurement operation of a quantum state? In the measurement context, a quantum state is essentially a probability distribution. The probability of an outcome is the squared magnitude of its amplitude. Measurement is sampling from this probability distribution, and the measurement outcome is the sample. This outcome receives probability 1 after measurement. We visualize the state as a rose (polar area) diagram, which is the circular version of a bar chart, and the measurement as spinning this diagram. The resulting collapsed state is a rose diagram where the measurement outcome has probability 1 (and the rest of the outcomes have probability 0).



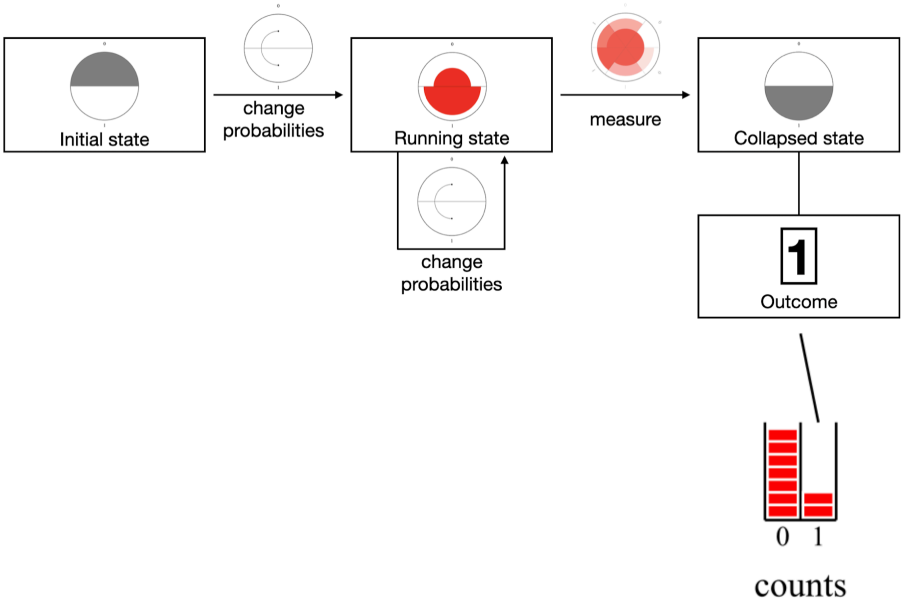
The Structure of Quantum Computations

Putting the fundamentals together: start with the default quantum state, change it with quantum transformations, measure it, record the outcome and repeat.

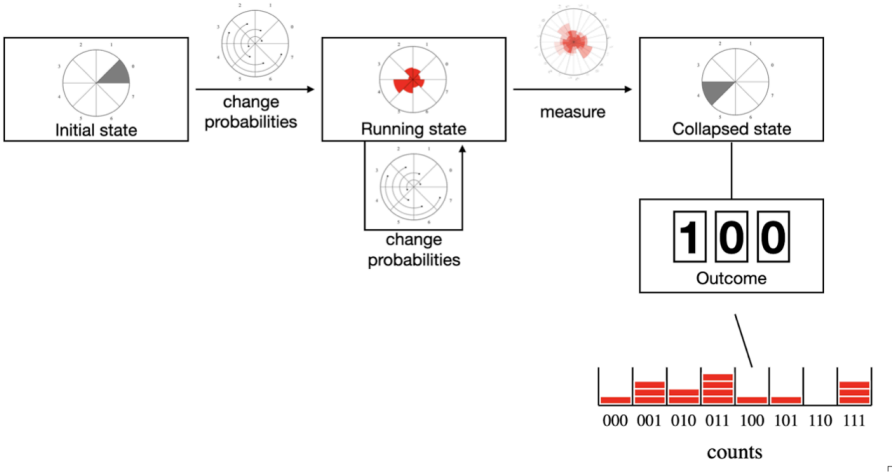
Here is an example of a one-qubit system evolution and measurement.



We can visualize the general case of running a one-qubit quantum computation as follows.

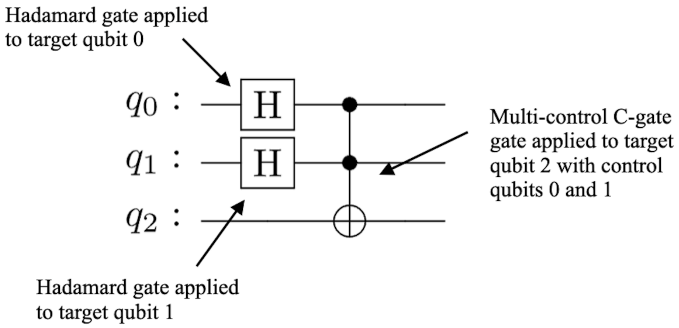


similarly, we can visualize the process of running a multi-qubit computation.



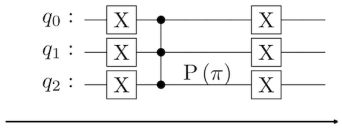
Quantum Circuits

A quantum program, or circuit, consists of a sequence of quantum transformations. A typical transformation in a qubit-based quantum system is defined by a single-qubit gate, a target qubit, and a number of other control qubits.



We can show circuits together with the states before and after their application.

Outcome	Binary	Amplitude	Direction	Amplitude Bar
0	000	-0.3789 - 0.2656i	-144.9717	
1	001	0.1811 + 0.0821i	24.4015	
2	010	0.0625 + 0.2799i	77.4126	
3	011	-0.3653 - 0.0663i	-169.7189	
4	100	0.6282 - 0.0000i	0.0000	
5	101	-0.0825 - 0.0962i	-130.5955	
6	110	-0.2684 - 0.0613i	-167.1357	
7	111	-0.0345 + 0.1965i	99.9532	



Outcome	Binary	Amplitude	Direction	Amplitude Bar
0	000	0.3789 + 0.2656i	35.0283	
1	001	0.1811 + 0.0821i	24.4015	
2	010	0.0625 + 0.2799i	77.4126	
3	011	-0.3653 - 0.0663i	-169.7189	
4	100	0.6282 - 0.0000i	0.0000	
5	101	-0.0825 - 0.0962i	-130.5955	
6	110	-0.2684 - 0.0613i	-167.1357	
7	111	-0.0345 + 0.1965i	99.9532	

Each Circuit Is Reversible

An important property of quantum transformations is that they are reversible: each has an inverse that cancels its action. The inverse of a transformation is obtained by using the inverse of its underlying elementary gate, and keeping the target and controls as they are.

Gate	Inverse
X	X
Y	Y
Z	Z
H	H
$P(\phi)$	$P(-\phi)$
$R_X(\theta)$	$R_X(-\theta)$
$R_Y(\theta)$	$R_Y(-\theta)$
$R_Z(\theta)$	$R_Z(-\theta)$

The inverse of a circuit is obtained by applying the inverses of its transformations in reversed order.

Quantum Advantage

Quantum advantage comes from exploiting quantum parallelism and measurement. In order to exploit parallelism, we need to counter its side effect: all amplitudes change in pairs according to a common formula. This is done through interference, where simultaneous changes to the numbers comprising a quantum state have to be balanced with the right types of changes. There are a few sources of useful natural interference, mostly present in the Phase Estimation and Amplitude Amplification algorithms. Since interference is hard, people try to find it using brute force search (variational algorithms). This is similar to finding weights for a neural network, but much more complex, hence the "brute force" term.

> "The goal in devising an algorithm for a quantum computer is to choreograph a pattern of constructive and destructive interference so that for each wrong answer the contributions to its amplitude cancel each other out, whereas for the right answer the contributions reinforce each other. If, and only if, you can arrange that, you'll see the right answer with a large probability when you look."
>
> -- Scott Aaronson, What Makes Quantum Computing So Hard to Explain? June 8, 2021, Quanta Magazine, https://www.quantamagazine.org/why-is-quantum-computing-so-hard-to-explain-20210608/

Looking for quantum advantage is essentially looking for sources of interference.

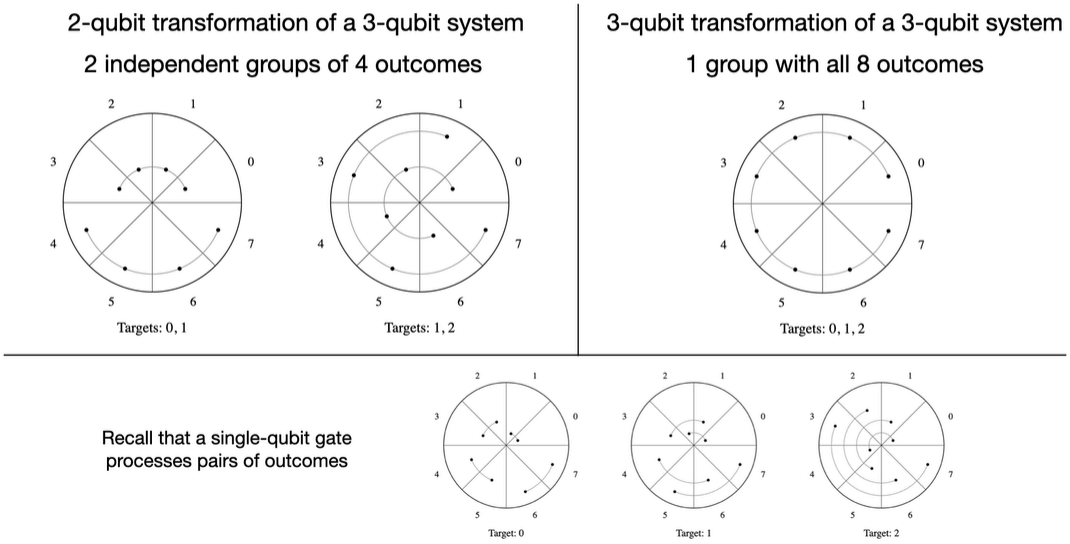
Conclusion

The concepts introduced in this document give enough information to define the implementation of a simple quantum simulator. We include such a simple implementation in Appendix B, where we also include a simple test, and its equivalent in Qiskit, IBM's quantum framework. The same concepts have been used in both a learning version of a simulator, called Hume: https://learnqc.com, and a high-performance simulator written in Rust, called Spinoza: https://github.com/QuState/spinoza.

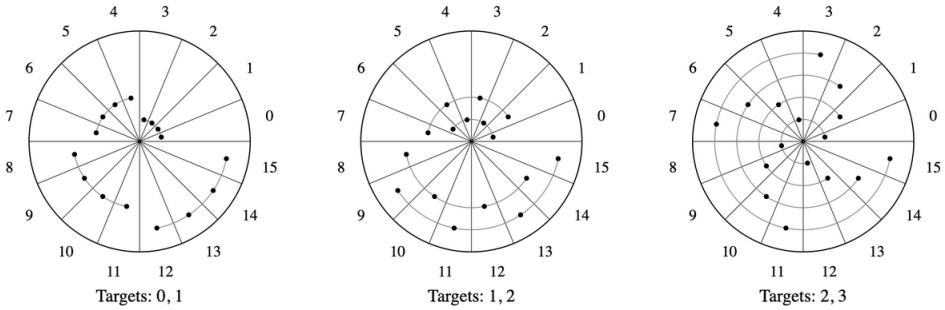
Appendix A: Multi-Target Transformations

How do we visualize multi-target (sometimes called multi-qubit) transformations? We showed how single-qubit gates recombine pairs of amplitudes. It turns out that multi-target transformations process groups of amplitudes, and preserve the combined probability of each group. We can use the same type of visualization for the groups, where each group is represented by an arc, and each outcome belongs to exactly one group. The way the grouping is done depends on the target qubits. A single maximum size unitary processes all amplitudes as a group. While there are theoretical reasons to use such unitaries, most of the time we don't need to. It is more efficient to use a for loop to process the groups than to use a single giant unitary, and then rely on other tools to avoid multiplications by 0 and 1.

An m-qubit transformation (2^m x 2^m unitary) applies to groups of 2^m amplitudes

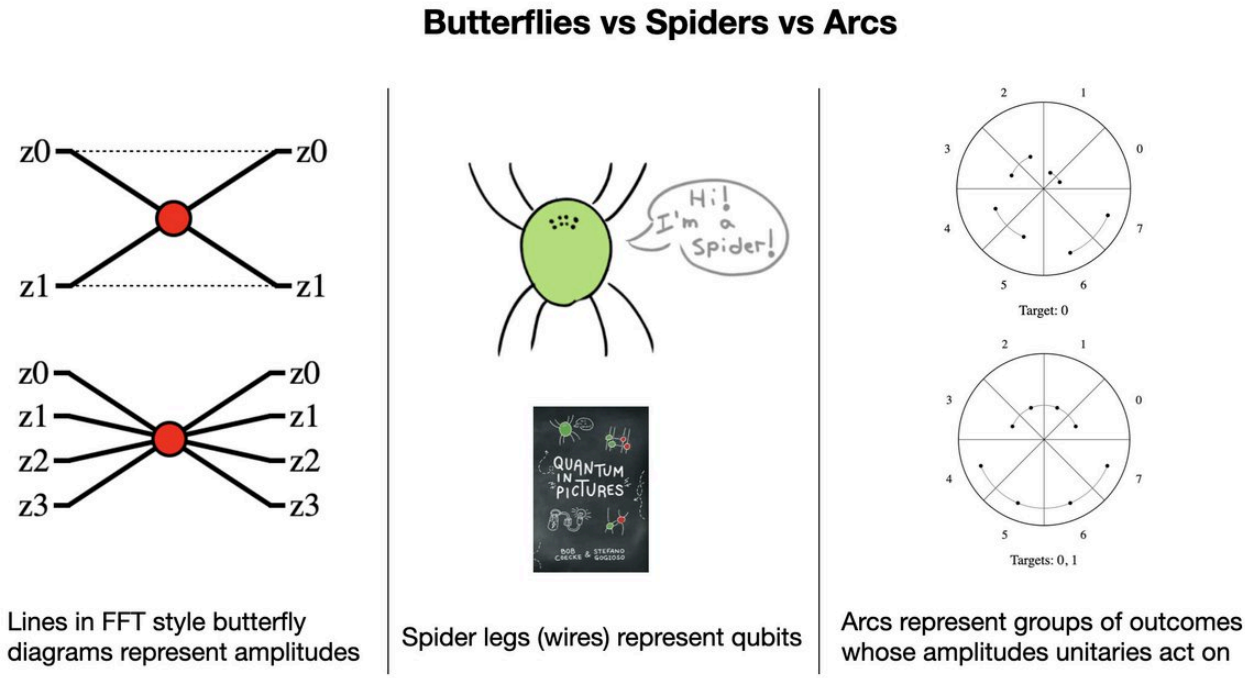


We can see that the arc diagrams have an advantage over the butterfly diagrams used in the Fast Fourier Transform literature. The arcs don't intersect, and they can capture all members of a group nicely.



Also note that control qubits eliminate some groups from the processing. Only groups of outcomes having 1 as a binary digit in the control positions are processed. In a simulator, this significantly speeds up computations. On real computers, control qubits are expensive.

You may say: "The butterfly diagrams look more like spiders than butterflies". Indeed. The "butterfly" name is established in the Fast Fourier Transform literature. Butterfly diagrams show how complex numbers are recombined, while spider diagrams are wire diagrams that focus on qubits, with a complete formal theory behind them. Arc diagrams show the groups of outcomes whose amplitudes are transformed by a unitary.



Appendix B: Code Implementation

```
from collections import Counter
from math import cos, sin, sqrt, log2
from random import choices

x = [[0, 1], [1, 0]]

z = [[1, 0], [0, -1]]


def phase(theta):
    return [[1, 0], [0, complex(cos(theta), sin(theta))]]


h = [[1/sqrt(2), 1/sqrt(2)], [1/sqrt(2), -1/sqrt(2)]]


def rz(theta):
    return [[complex(cos(theta / 2), -sin(theta / 2)), 0], [0, complex(cos(theta / 2), sin(theta /
2))]]


y = [[0, complex(0, -1)], [complex(0, 1), 0]]


def rx(theta):
```

```
    return [[cos(theta/2), complex(0, -sin(theta/2))], [complex(0, -sin(theta/2)), cos(theta/2)]]


def ry(theta):
    return [[cos(theta/2), -sin(theta/2)], [sin(theta/2), cos(theta/2)]]


def init_state(n):
    state = [0 for _ in range(2 ** n)]
    state[0] = 1
    return state


def is_bit_set(m, k):
    return m & (1 << k)


def pair_generator(n, t):
    distance = int(2 ** t)

    for j in range(2 ** (n-t-1)):
        for k0 in range(2*j*distance, (2*j+1)*distance):
            k1 = k0 + distance
            yield k0, k1


def process_pair(state, gate, k0, k1):
    x = state[k0]
    y = state[k1]
    # new amplitudes
    state[k0] = x * gate[0][0] + y * gate[0][1]
    state[k1] = x * gate[1][0] + y * gate[1][1]


def transform(state, t, gate):
    n = int(log2(len(state)))
    for (k0, k1) in pair_generator(n, t):
        process_pair(state, gate, k0, k1)


def c_transform(state, c, t, gate):
    n = int(log2(len(state)))
    for (k0, k1) in filter(lambda p: is_bit_set(p[0], c), pair_generator(n, t)):
        process_pair(state, gate, k0, k1)


def mc_transform(state, cs, t, gate):
    assert t not in cs
    n = int(log2(len(state)))
    for (k0, k1) in filter(lambda p: all([is_bit_set(p[0], c) for c in cs]), pair_generator(n, t)):
        process_pair(state, gate, k0, k1)


def measure(state, shots):
```

```
    samples = choices(range(len(state)), [abs(state[k])**2 for k in range(len(state))], k=shots)
    counts = {}
    for (k, v) in Counter(samples).items():
        counts[k] = v
    return counts


def test_simulator():
    state = init_state(3)
    transform(state, 0, h)
    transform(state, 1, h)
    mc_transform(state, [0, 1], 2, x)

    samples = measure(state, 1000)

    print('\n', state)
    # [0.4999999999999999, 0.4999999999999999, 0.4999999999999999, 0.0, 0.0, 0.0, 0.0,
0.4999999999999999]
    print('\n', samples)
    #  {2: 2595, 1: 2470, 7: 2474, 0: 2461}


def test_qiskit():
    from qiskit import QuantumRegister, QuantumCircuit
    from qiskit_aer.backends.compatibility import Statevector
    q = QuantumRegister(3)
    qc = QuantumCircuit(q)

    qc.h(0)
    qc.h(1)

    qc.mcx([0, 1], 2)

    print('\n', Statevector(qc).data)

    # [0.5+0.j 0.5+0.j 0.5+0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0.5+0.j]
```