Solving the Deutsch's oracle problem

The answer to the first question is apparent

- two queries are enough to determine what the function does
  - there are only two possible alternatives for the input bit

What does it happen using a quantum computer?

Is quite natural to forecast that just one query is required!

QC: Much Ado About Nothing?

As a matter of fact, that is a wrong intuition.

- remember that quantum computers **don't** really compute with all values simultaneously
- in the end, a qubit collapses to a single bit of information:
  - a single bit of information is not enough to uniquely identify one out of four functions
- since we need two bits, this problem actually requires two queries also on a quantum computer!

End of the game?

A problem more *suitable* to QC

Suppose we are interested to know whether the function is

- constant: constant 0 or constant 1 (regardless of the input)

or

- variable: identity or negation

Since there are two categories, a single bit is enough to identify the answer

- nevertheless, this problem takes two queries on a classical computer.
- how many on a quantum computer?

Here, at last, QC advantage

On a QC a single query is enough to tell whether the function is constant or variable

- this undeniably outperforms a classical computer

Finally, we are going to leverage the features of the superposition!

Let's start defining each of the four functions acting on a single bit on a quantum computer

- first issue: we are using a QC model in which computations must be *reversible*

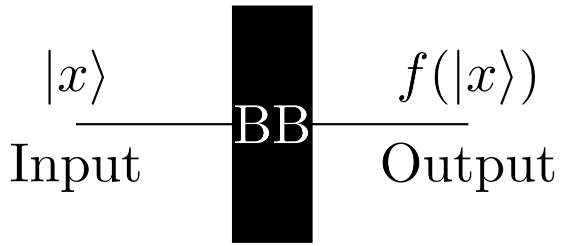
  (https://physics.stackexchange.com/questions/704625/quantum-and-classical-physics-are-reversible-yet-quantum-gates-have-to-be-rever);

- the constant functions are **not** *reversible*.

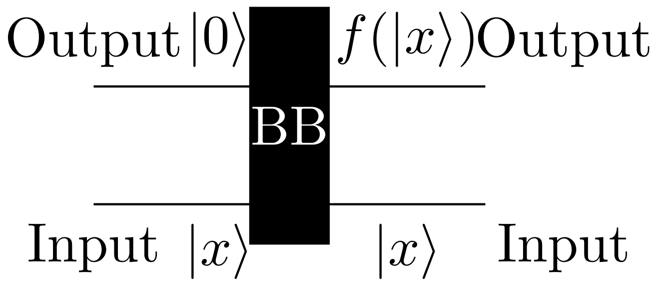
Making QC operations reversible

An additional *output bit* to which the function action is applied provides what we need

From



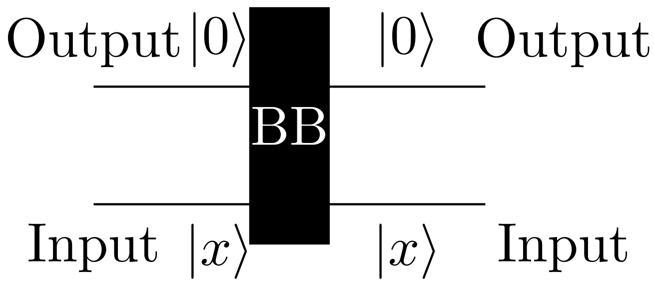
to



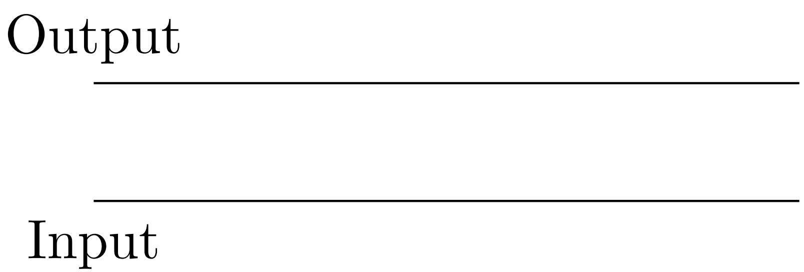
- now there are two qubits and it is necessary to rewire the black box;
- the input qubit is unchanged;
- the value of the function on the input qubit is written to the output qubit;
- the black box assumes that the output input is zero.

The four one-bit operations

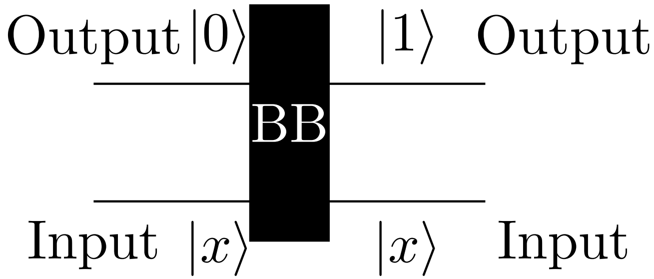
constant 0:



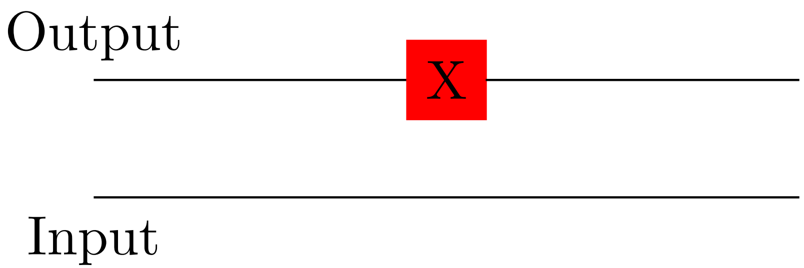
basically this corresponds to a *void black box*



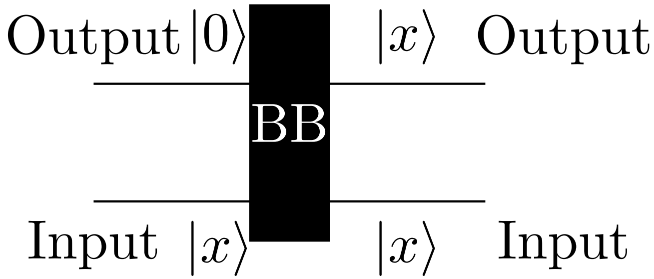
Moving to constant 1 we have:



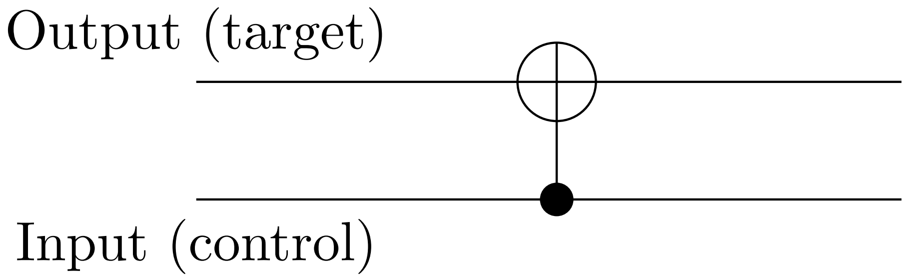
corresponding to a single negation

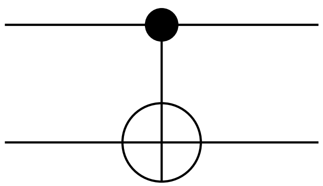


The identity is a bit more complex:

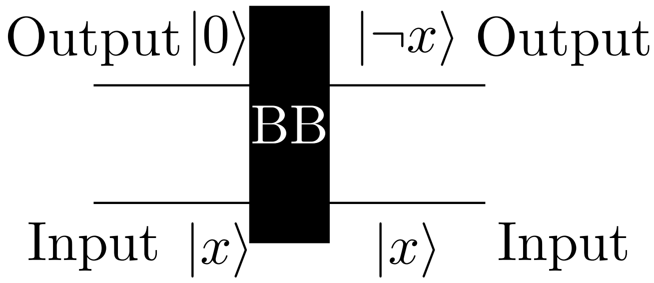


here the input bit plays the role of *control* bit;

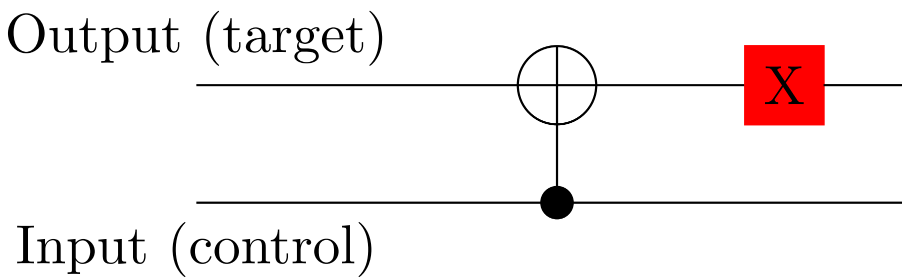


Where  represents a CNOT gate.

Once the identity is well understood, the negation becomes simple:

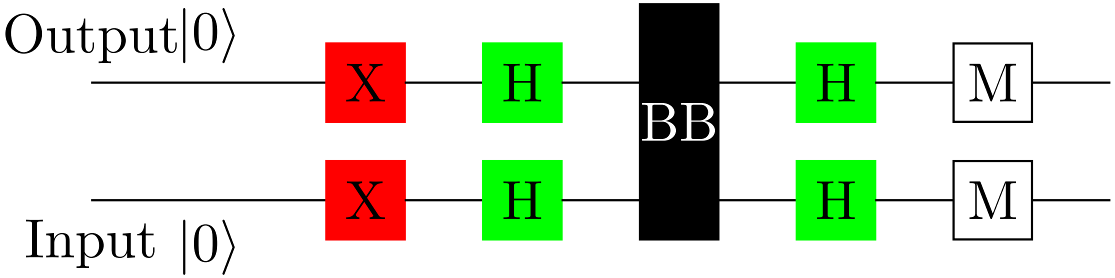


again the input bit plays the role of *control* bit



And the one-query solution to the Deutsch's oracle problem is...

A quantum circuit solving the Deutsch's problem



where $\boxed{\text{M}}$ represents a *measurement* gate.

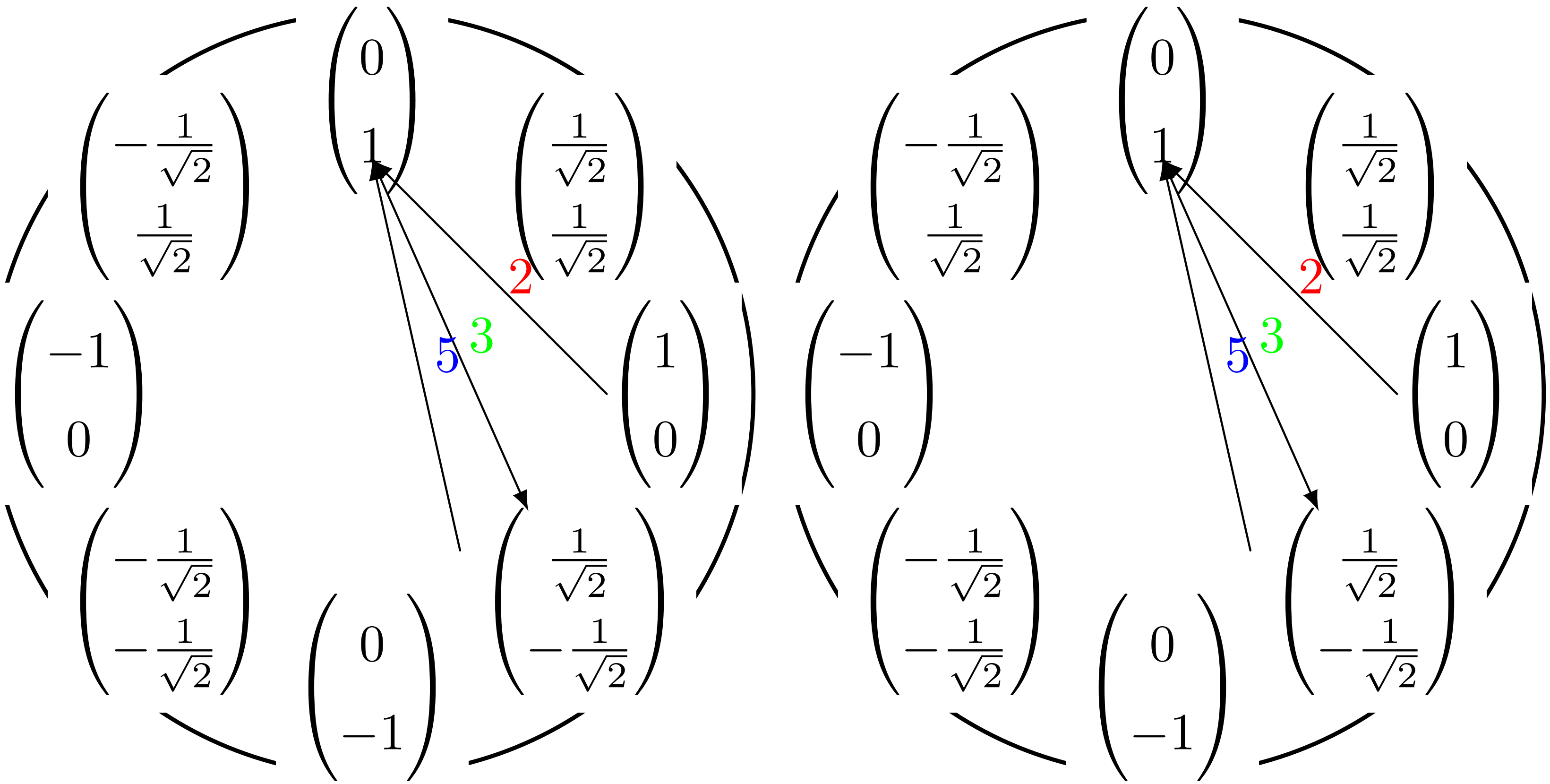
We are going to show that:

- if the black-box function is constant, the measure returns $|11\rangle$;
- if the black-box function is variable, the measure returns $|01\rangle$.

The *algorithm* is the following:

1. initialize qubits to $|0\rangle$;
2. bit flip them: both become $|1\rangle$;
3. apply the Hadamard gate to put them into equal superposition;
4. send them into the black-box.
   One of the four circuits we described above is applied.
5. apply again the Hadamard gate;
6. finally measure them.

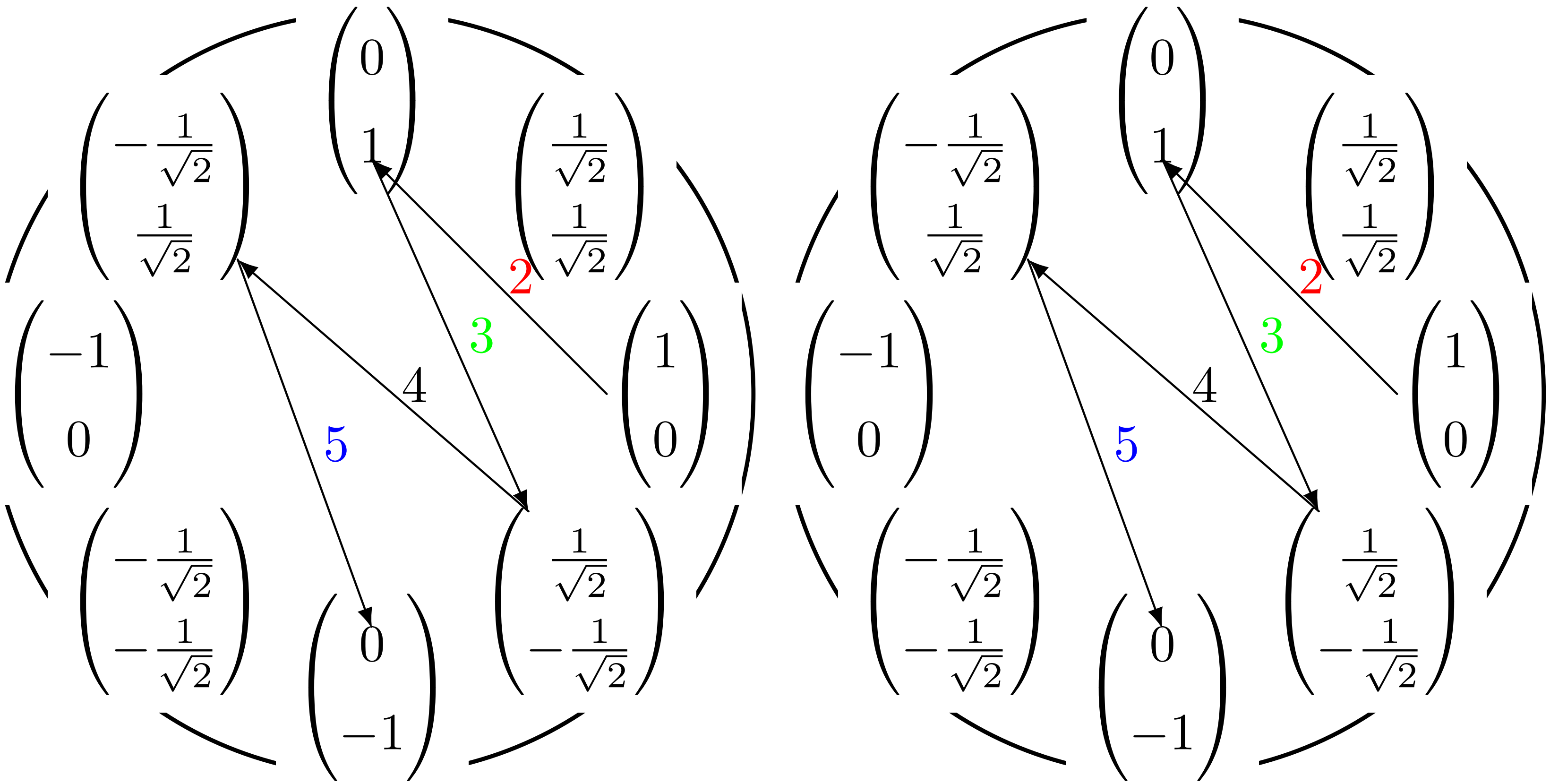
Steps two and three correspond to two moves along the unit circle.



if the *black box* is constant 0, it does nothing.

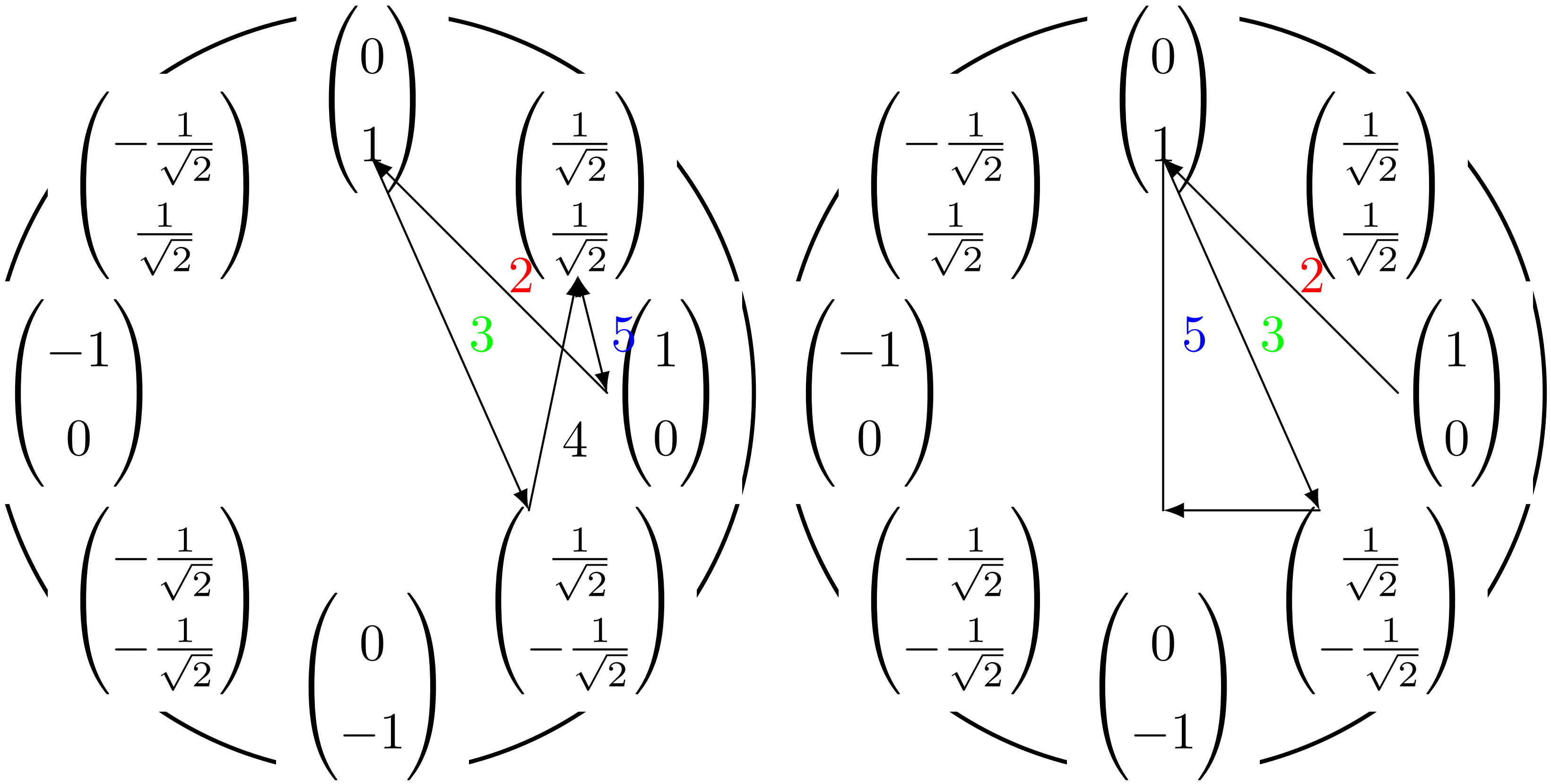
The post-processing is a single Hadamard gate. We are back to $|11\rangle$.

If the *black box* is constant 1, there is an additional negation



Still $|11\rangle$ at the end.

The first variable function is identity (based on the CNOT gate).



The final result is $|01\rangle$.

Let's look at the details of step 4 in matrix form.

$$C\left(\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix}\otimes\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix}\right)=C\begin{pmatrix}\frac{1}{2}\\ \frac{-1}{2}\\ \frac{-1}{2}\\ \frac{1}{2}\end{pmatrix}=\frac{1}{2}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}1\\-1\\-1\\1\end{pmatrix}=\frac{1}{2}\begin{pmatrix}1\\-1\\1\\-1\end{pmatrix}=\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}\otimes\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix}$$

When we finally apply the Hadamard operator we obtain

$|0\rangle$ (starting from $\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}$)

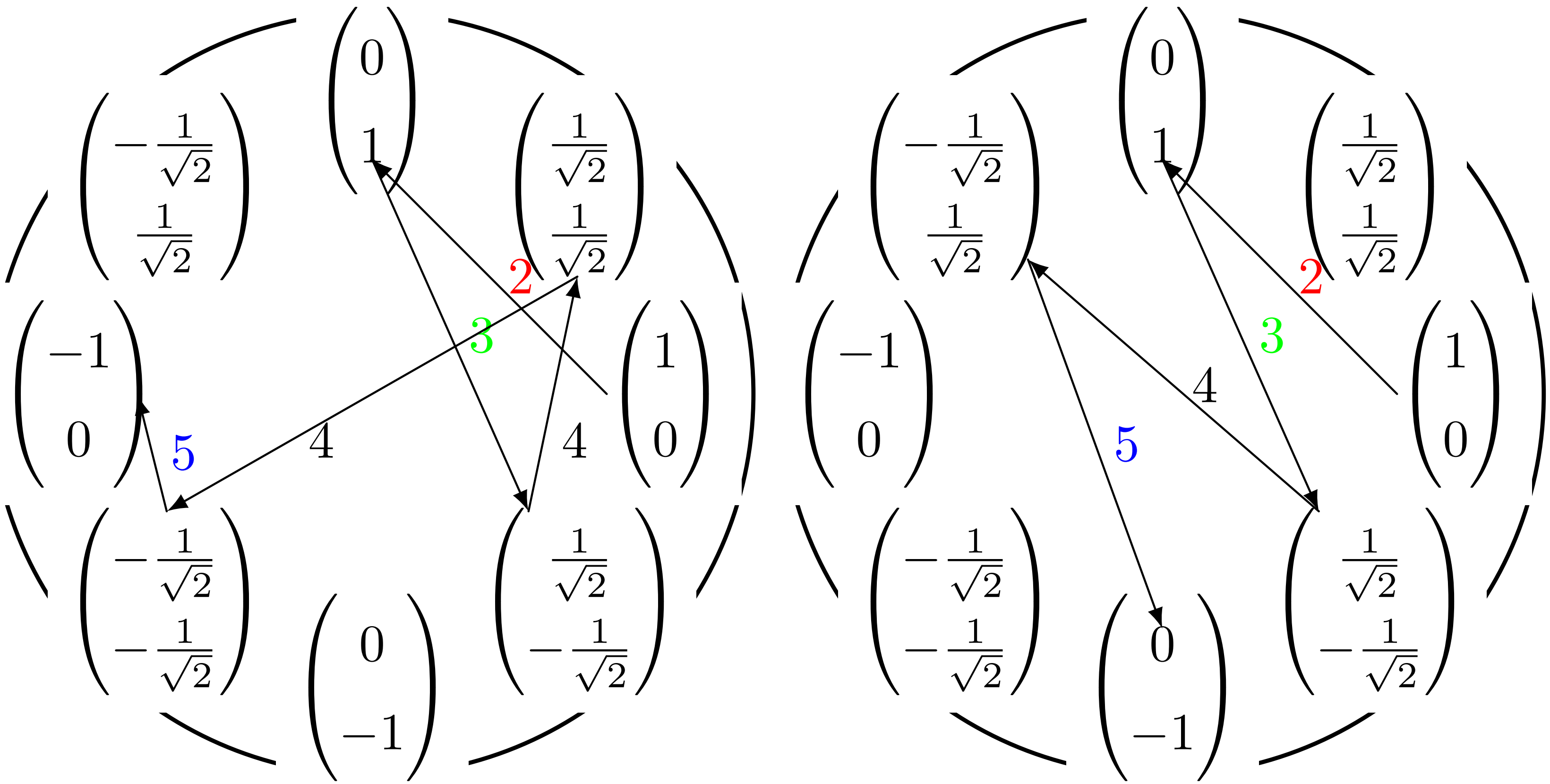
and

$|1\rangle$

(starting from $\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix}$)

so the final state is $|01\rangle$.

negation requires an additional bit flip after the CNOT



When we finally apply the Hadamard operator we obtain

$|0\rangle$ (starting from $\begin{pmatrix} \frac{-1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{pmatrix}$)

and

$|1\rangle$ (starting from $\begin{pmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$), so the final state is again $|01\rangle$.

It is possible to determine whether the function is constant or variable by using a **single** query in a QC setting!

A *layman* explanation

- The difference *within* the categories (a negation) is neutralized:
- the only difference between constant zero and constant one is a single negation gate;
- when the negation gate is applied in a superposed state, it does not really have any effect;
- the effect of the negation gate is neutralized and this, in turn, neutralizes the difference within the categories.

The difference *between* the categories (a CNOT) is magnified:

- the variable functions have a CNOT and the constant functions do not

Most of the power of QC comes from the chance of changing the action of various logic gates by leveraging suitable superposition states.

What next?

The solution to the Deutsch's problem appears, somehow, contrived and over-killing but...

- it can be *easily* extended to the case of a $n-$bit black box (Deusch-Josza problem);
- on a classic computer the problem requires a number of queries that is $O(2^n)$;
- with a quantum computer it can be solved again with a *single* query

so it shows the route to an *exponential* speedup!

From Deutsch to Shor

- A further variant is the Simon's periodicity problem
  - again there is a black box and the problem is to figure out some properties of the function that the black box implements.
- The famous Shor's algorithm for integer factorization, is built on the idea of finding the period of a sequence
  - the problem may be formulated in terms of a *decision* problem exactly like determining whether the black box has a particular property or not.

Integer Factorization

Given an integer N, find two integers $1 < P, Q$ such that $N = P \times Q$.

N requires n bits to be represented
($N = 4294967296$ requires $n = 32$ bits)

- More difficult when P and Q are primes with roughly the same number of bits.

No algorithm with polynomial-in-n time complexity is known
(but there is no proof that it does not exist!)

The straightforward algorithm that tries all factors from 2 to $\sqrt{N}$ takes time exponential in n.

The most efficient algorithm has a complexity $O(\exp\left(\sqrt[3]{\frac{64}{3}n(\log n)^2}\right))$

It is not feasible to factor integer with more than 1000 bits.

A different viewpoint

Let's start with a "guess" $1 < g < N$

There are two alternatives:

1. g is a factor of N **or** it shares a common factor with N, that is the g.c.d.$(g, N) > 1$
   (we are lucky and the problem is solved!)

2. g is neither a factor of N nor shares a common factor
   This is the interesting part...

It is "well known" (Euler) that given A and B (both integers), there exists a power p and a multiple m (both integers) such that

$$A^p = m \times B + 1$$

For instance, suppose we take 3 and 7:

$3^2 = 9 = 1 \times 7 + 2$

$3^3 = 27 = 3 \times 7 + 6$

$3^4 = 81 = 11 \times 7 + 4$

$3^5 = 243 = 34 \times 7 + 5$

$3^6 = 729 = 104 \times 7 + 1$

So we can write

$$g^p = m \times N + 1$$

then

$$g^p - 1 = m \times N$$

Assume (1) p is even then

$$(g^{\frac{p}{2}} + 1) \times (g^{\frac{p}{2}} - 1) = m \times N$$

$$((g^{\frac{p}{2}} + 1) \times (g^{\frac{p}{2}} - 1)) \mod N = 0$$

Assume (2) neither $(g^{\frac{p}{2}} + 1)$ nor $(g^{\frac{p}{2}} - 1)$ is a multiple of N
(see at the bottom for both "assumptions")
Then we have learned a factor of N. Why?

- $N = P \times Q$
- either P divides $(g^{\frac{p}{2}} + 1)$ and Q divides $(g^{\frac{p}{2}} - 1)$ or
  the other way around
  Q divides $(g^{\frac{p}{2}} + 1)$ and P divides $(g^{\frac{p}{2}} - 1)$

Number theory says that: for any N, if g is relatively prime to N
then, with probability **at least** $\frac{3}{8}$

- p is even;
- neither $(g^{\frac{p}{2}} + 1)$ nor $(g^{\frac{p}{2}} - 1)$ is a multiple of N;

(don't ask me to demonstrate that!)

So the problem is finding p.

Consider the modular exponentiation sequence $a_k = g^k \mod N$
obviously a_k may assume only the values $0, ..., N-1$.
So there are i and j such that $a_i = a_j$

assuming $j > i$, a_k has a period $r = j - i$ that (if it is even) corresponds to the p we are looking for. Why?

$$g^r \mod N = 1$$

(because $g^0 = 1$)

$$(g^r - 1) \mod N = 0$$

So, if we find the period r we can *easily* solve the integer factorization problem. Sounds good...

Unfortunately, finding the period of the sequence g_k is **not** easier than directly searching for factors of N on a **classic** computer!
(do you have a clue about the reason?)

But a Quantum Fourier Transform (QFT) allows to find the period in polynomial time!

The magic (Q)FT

(an absolutely **not** rigorous reminder...)

- In general a Fourier transform maps from the time domain to the frequency domain;
- Fourier transforms map functions of period r to functions which have non-zero values only at multiples of the frequency $\frac{2\pi}{r}$;
- the Discrete Fourier Transform (DFT) operates on N equally spaced samples in the interval $[0, 2\pi)$ and can be implemented as a (symmetric) matrix-vector product where the Fourier matrix F_n is defined as

$$F_N = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_N^1 & \omega_N^2 & \dots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \dots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \dots & \omega_N^{(N-1)^2} \end{pmatrix}$$

with

$$\omega_N = e^{-i2\pi/N}$$

- The DFT of a (sampled) function of period r is a function concentrated near multiples of $\frac{N}{r}$
  - if the period r divides N evenly, the result is a function having non-zero values **only** at multiples of $\frac{N}{r}$
  - otherwise there will be non-zero terms at integers **close** to multiples of $\frac{N}{r}$

Suppose $f(x) = \sin(3 * x)$ and $N = 9$.

Index	**f**	**DFT**
0	0	(0, 0i)
1	0.866	(0,0i)
2	-0.866	(0,0i)
3	0	(0,-4.5i)
4	0.866	(0,0i)
5	-0.866	(0,0i)
6	0	(0,4.5i)
7	0.866	(0,0i)
8	-0.866	(0,0i)

- When N is a power of 2, the DFT may be computed in a very efficient way becoming a *Fast* Fourier Transform

  - a clever *classic* recursive algorithm exploits the special structure of the matrix
  - the computational cost drops from $O(N^2)$ to $O(N \log N)$

- The Quantum Fourier transform (QFT) is a variant of the DFT
  - the Fourier matrix F_n is unitary, so it is a quite natural consider it a *quantum operation*;

$$\mathcal{QFT}(\sum_{k=0}^{N-1} x_k |k\rangle) = \sum_{k=0}^{N-1} c_k |k\rangle$$

  - however this quantum operation does something different from the classical Fourier transform because it operates on the amplitude of the quantum state;
  - the QFT gives the amplitudes of the resulting state.
  - It is not trivial to implement the QFT!