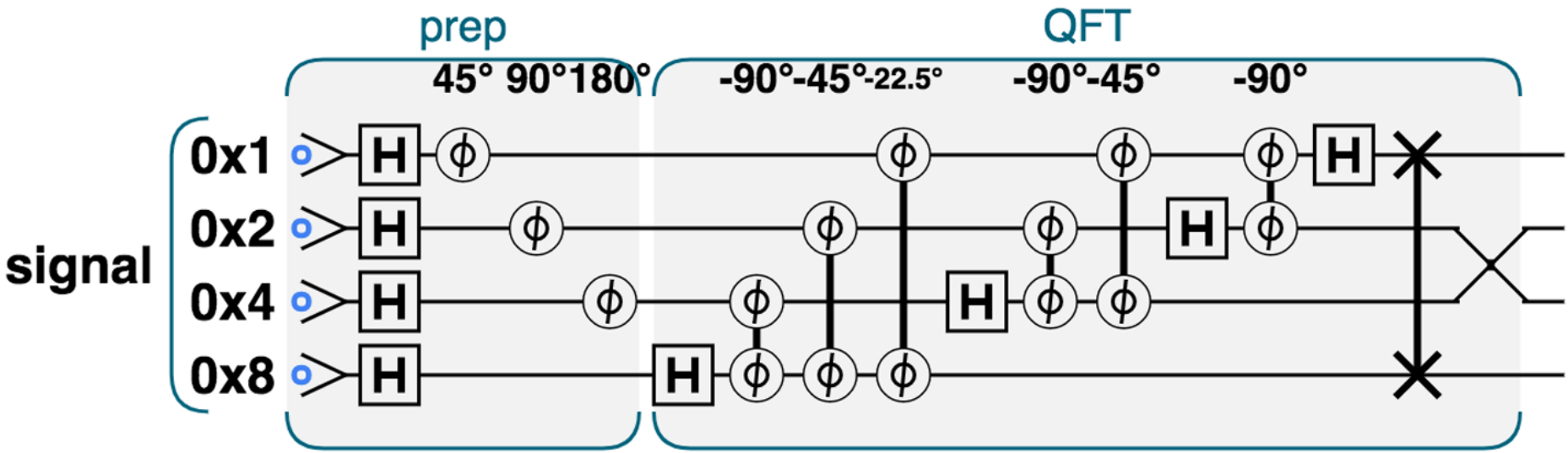
Ch10: The Quantum Fourier Transform

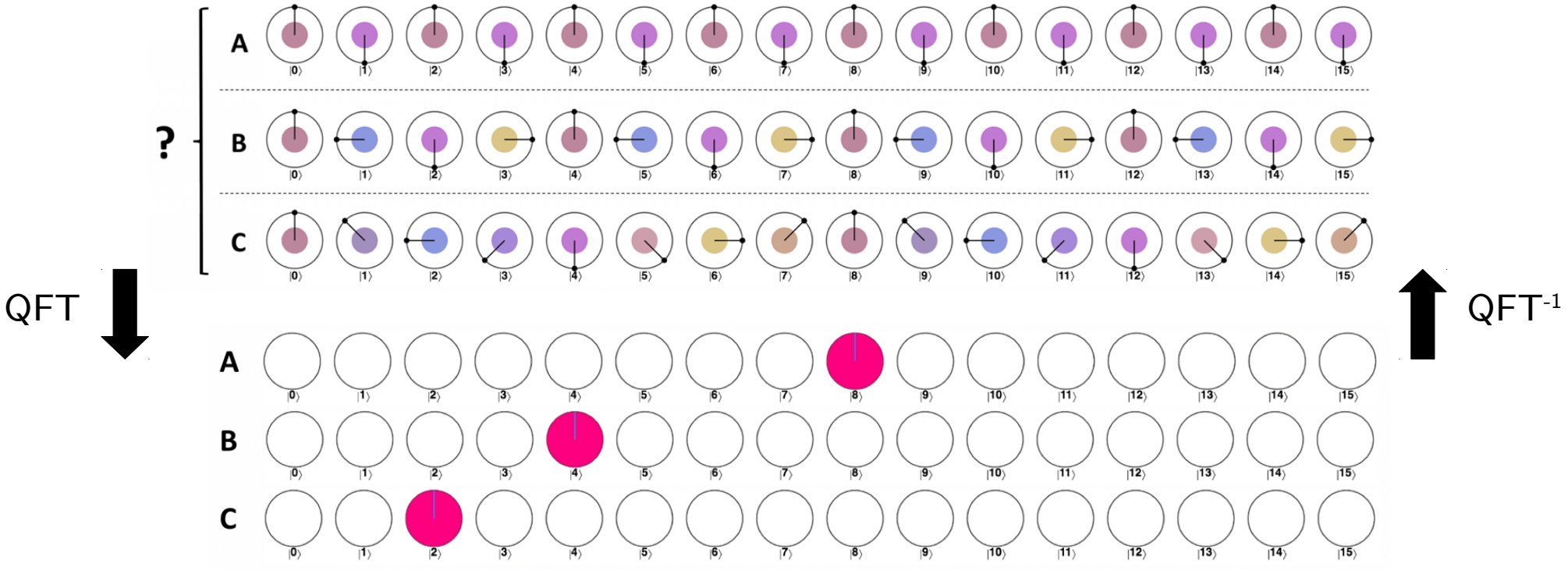
Hidden patterns

The quantum Fourier transform (QFT) is a QPU primitive allowing us to READ out the frequency with which values vary in a quantum register.



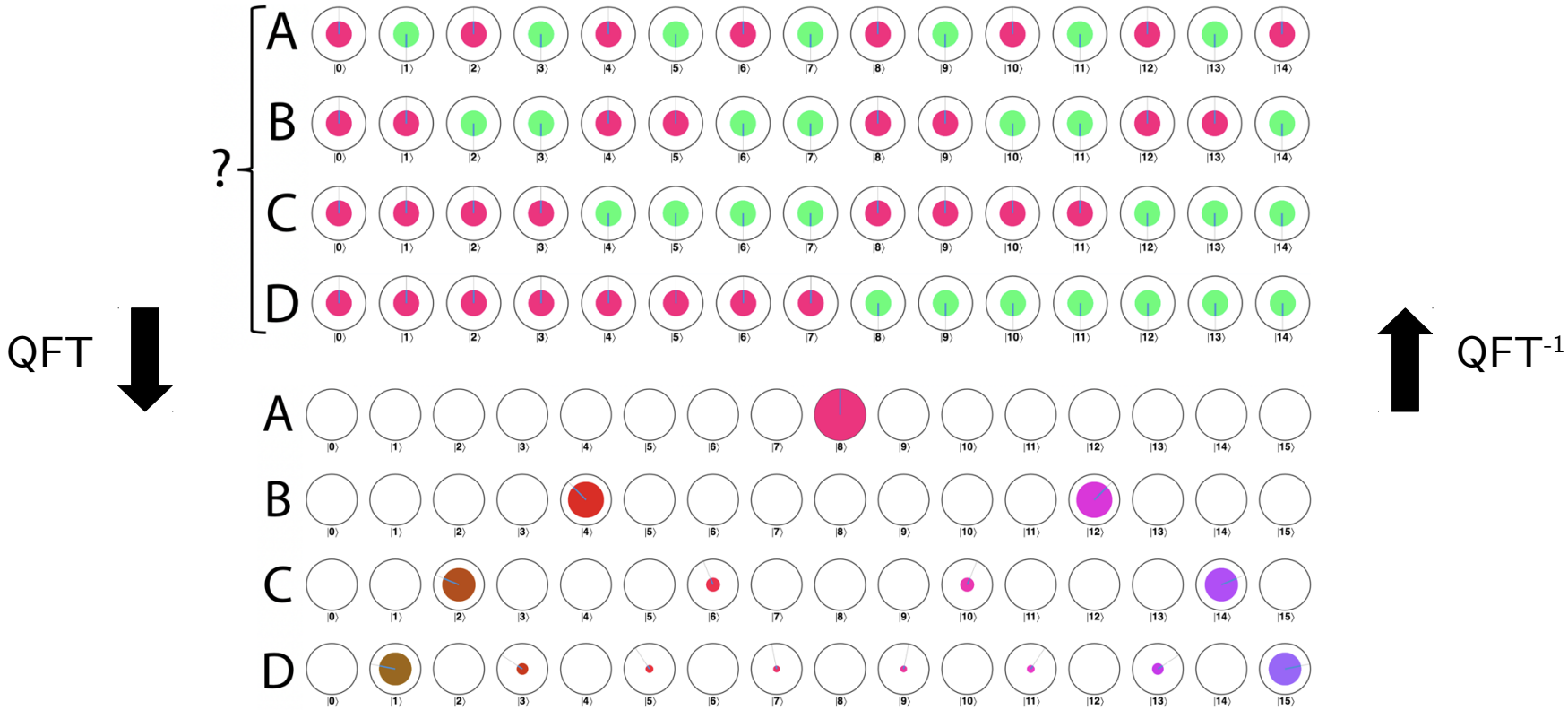
Example applications of QFT

Let's illustrate the concept with few examples:



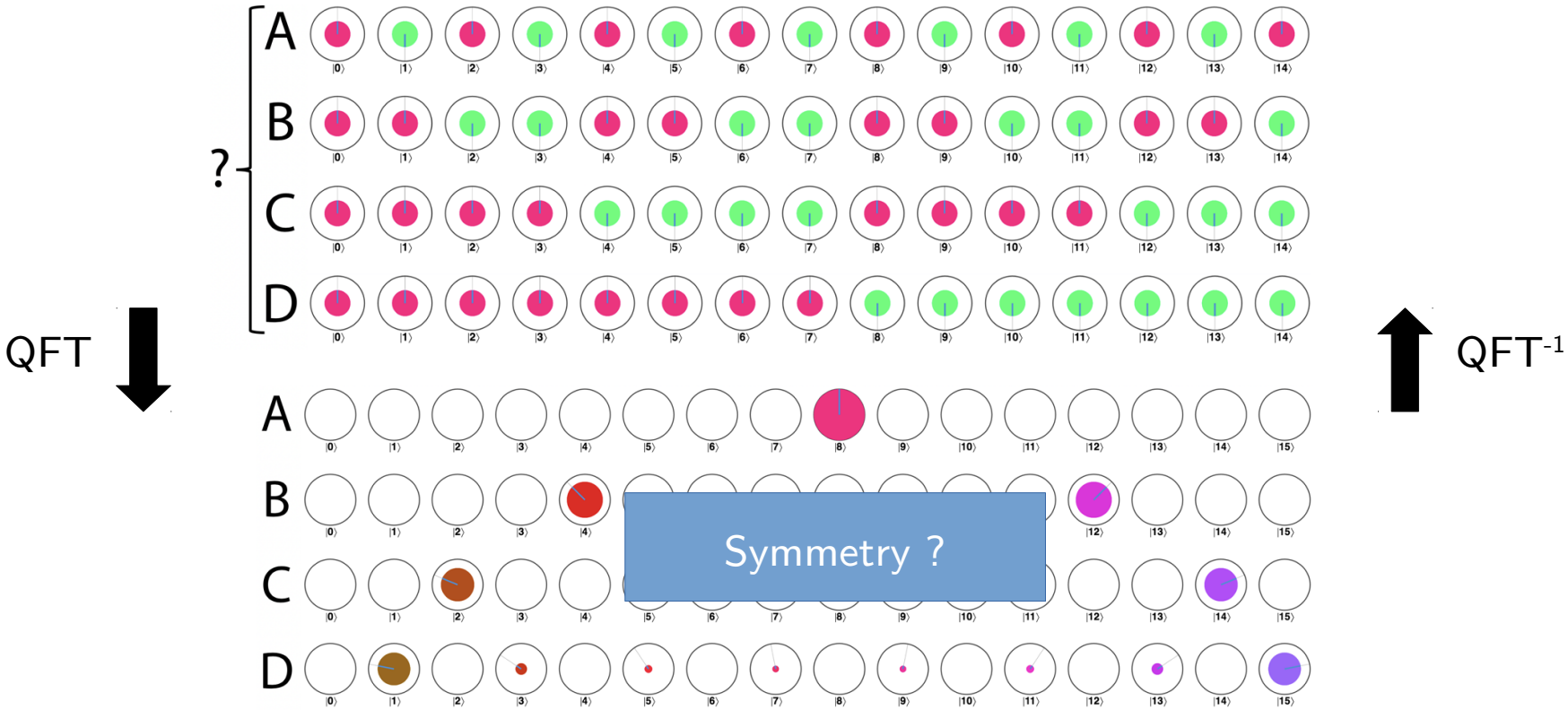
Example applications of QFT

Many periodic signals don't give us such nice QFTs:



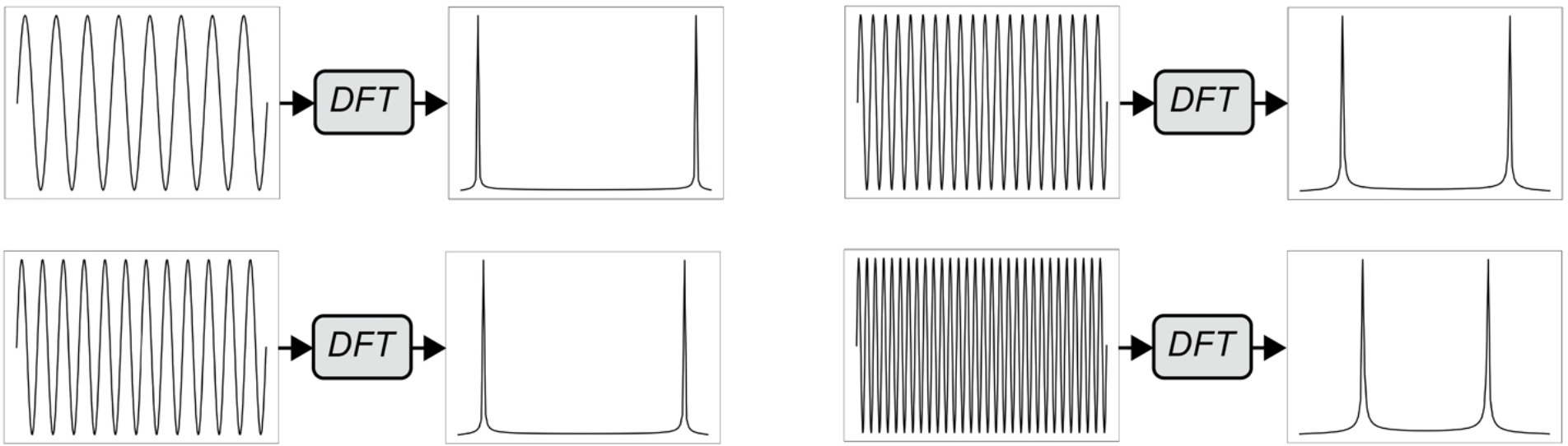
Example applications of QFT

Many periodic signals don't give us such nice QFTs:



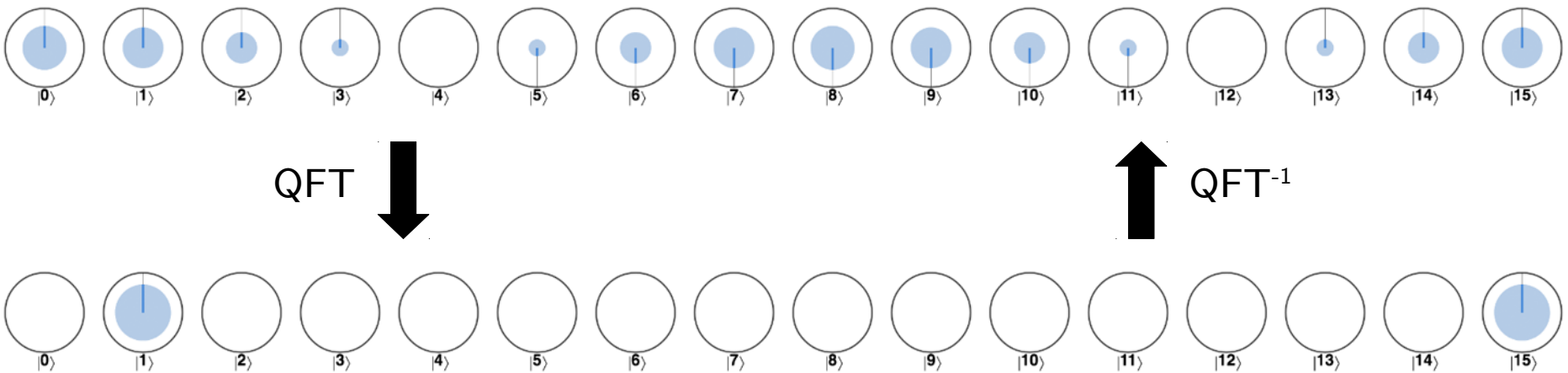
Real and complex DFT inputs

A property of the DFT of any real signal (one where the samples are all real numbers, as is the case for most conventional signals). In such cases, only the first half of the DFT result is actually useful:



Real and complex QFT inputs

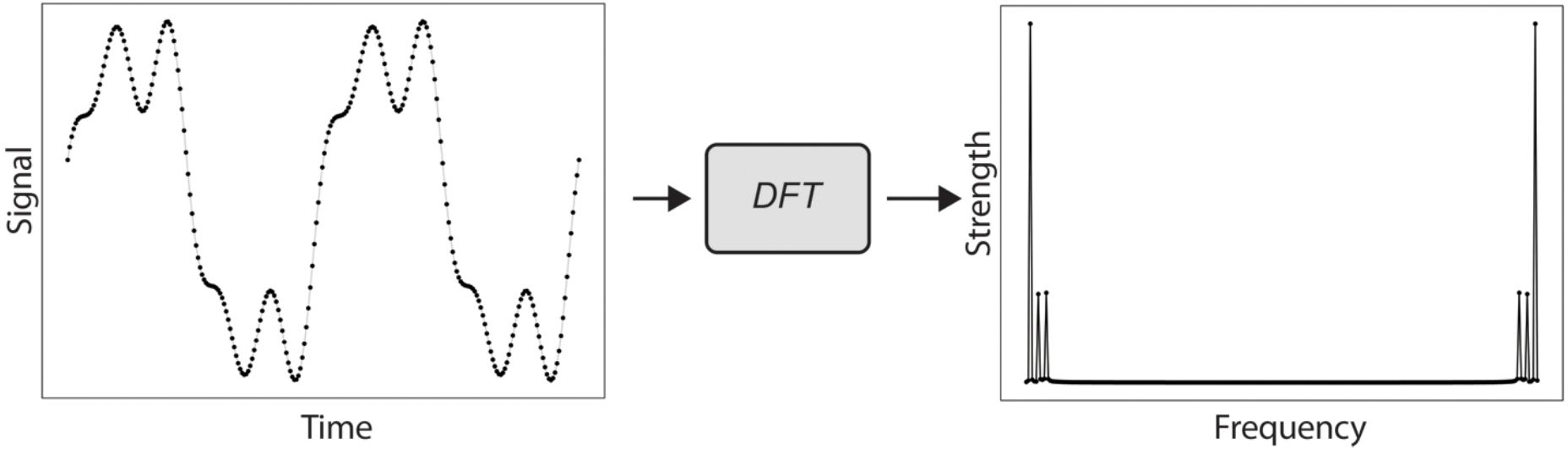
The same caveat on interpreting the output of real versus complex input signals also holds true for the QFT. It is nevertheless possible to prepare entirely real signals in our QFT register. Suppose that the signal we planned to QFT was encoded in the magnitudes of our input QPU register, instead of the relative phases:



Consequently, we need to take care of how we interpret QFT results dependent on whether our input register is encoding information in phases or magnitudes.

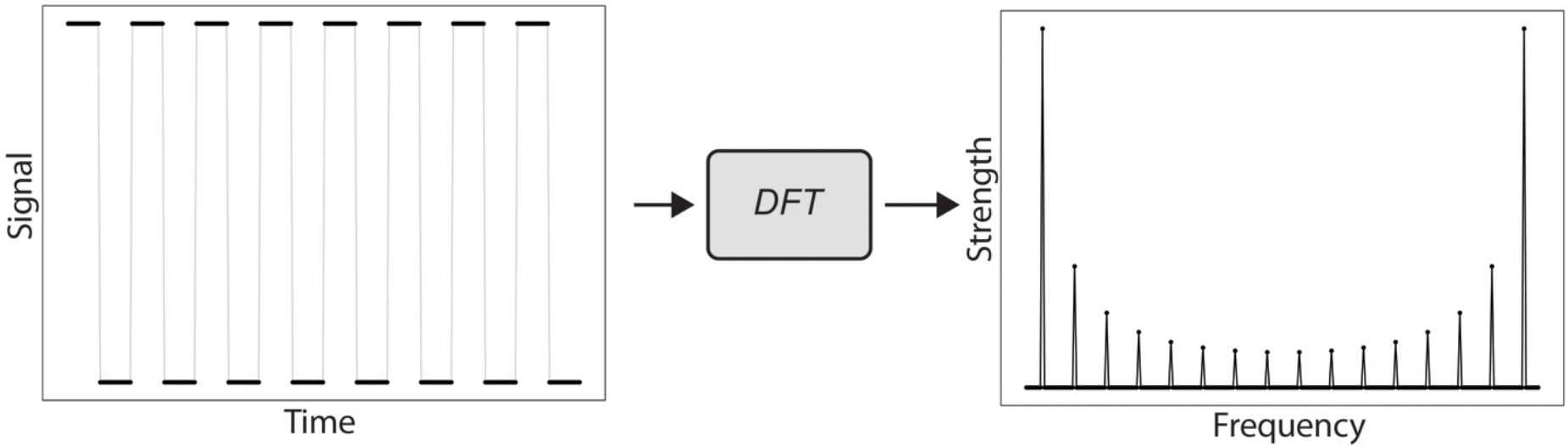
DFT everything

So far we've vaguely talked about the DFT (and QFT) showing us what frequencies a signal contains. To be slightly more specific, these transforms are actually telling us the frequencies, proportions, and offsets of simple sinusoidal components that we could combine to produce the input signal.



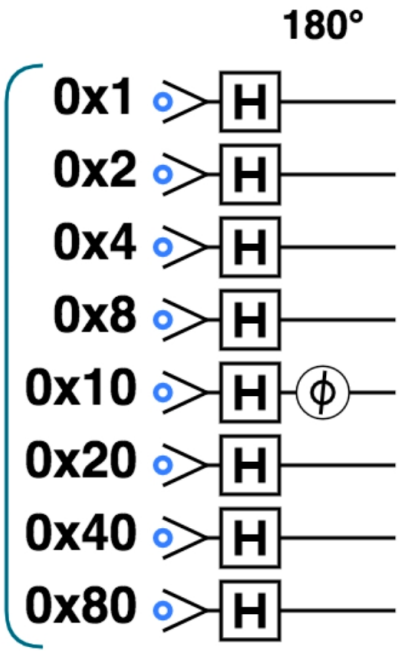
DFT everything

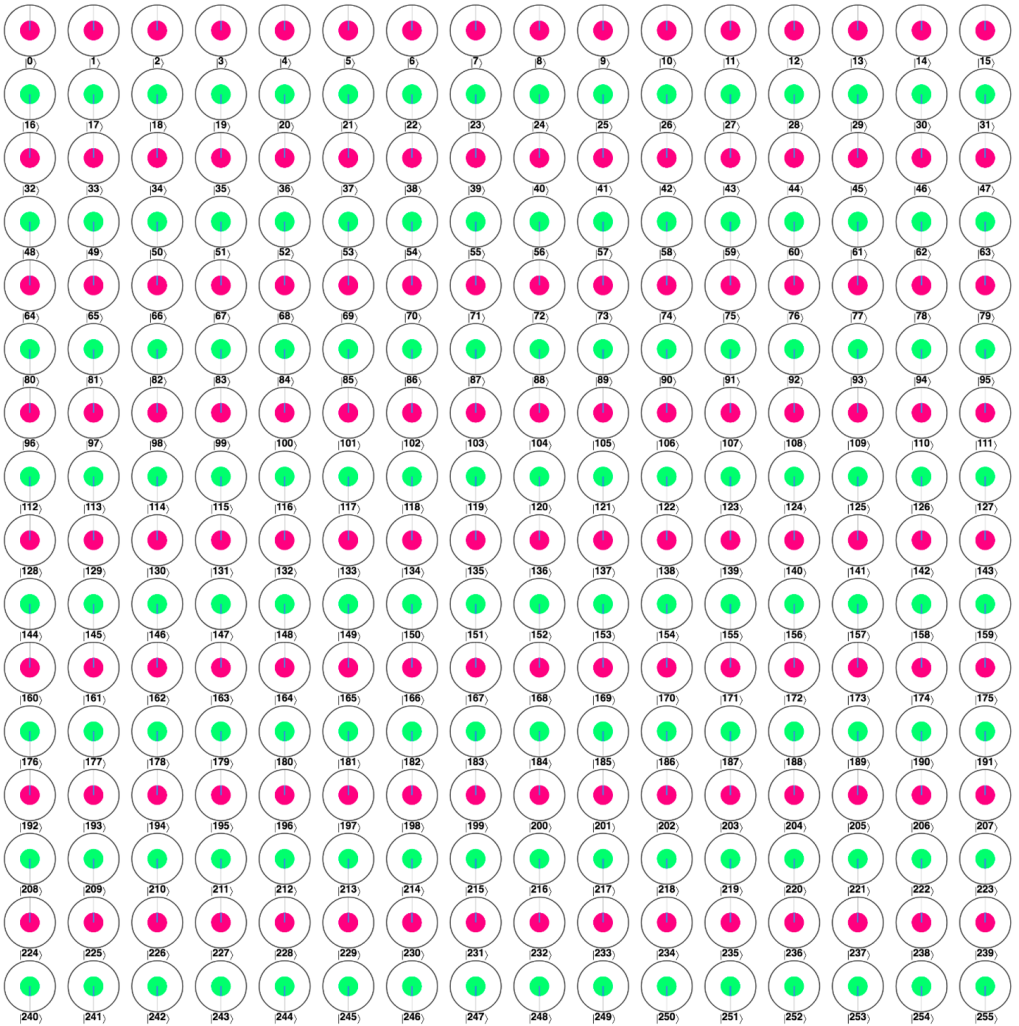
One particularly common and useful class of input signals that don't look at all sinusoidal are square waves.



QFT everything

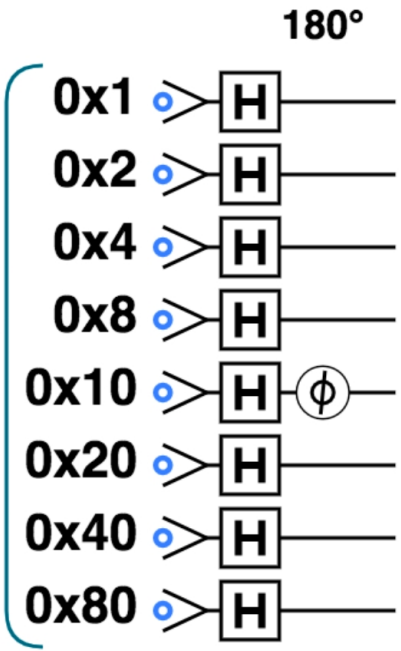
One particularly common and useful class of input signals that don't look at all sinusoidal are square waves.



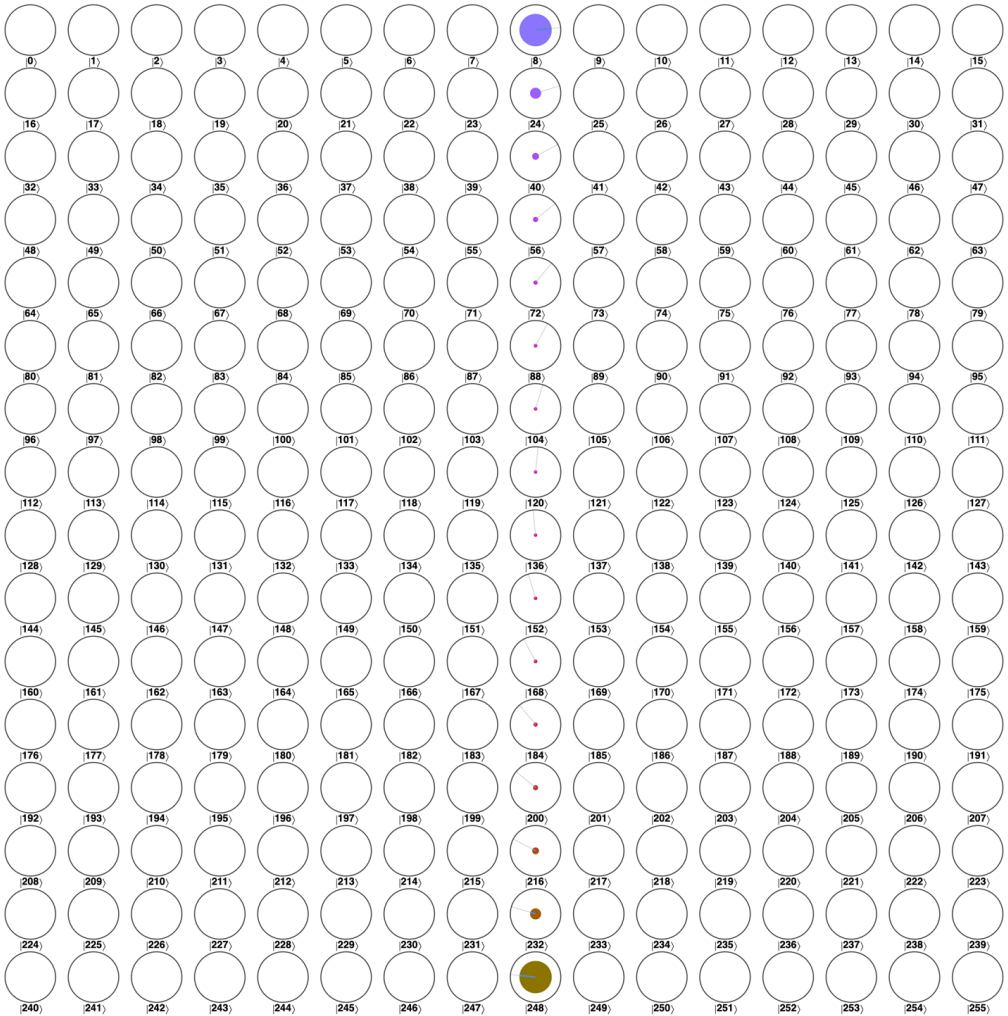


QFT everything

One particularly common and useful class of input signals that don't look at all sinusoidal are square waves.

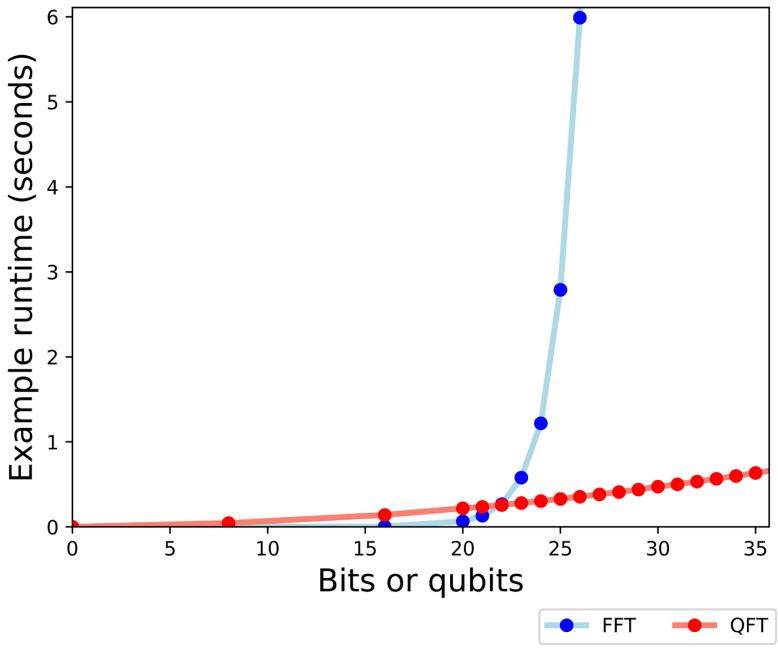


QFT



The QFT is fast

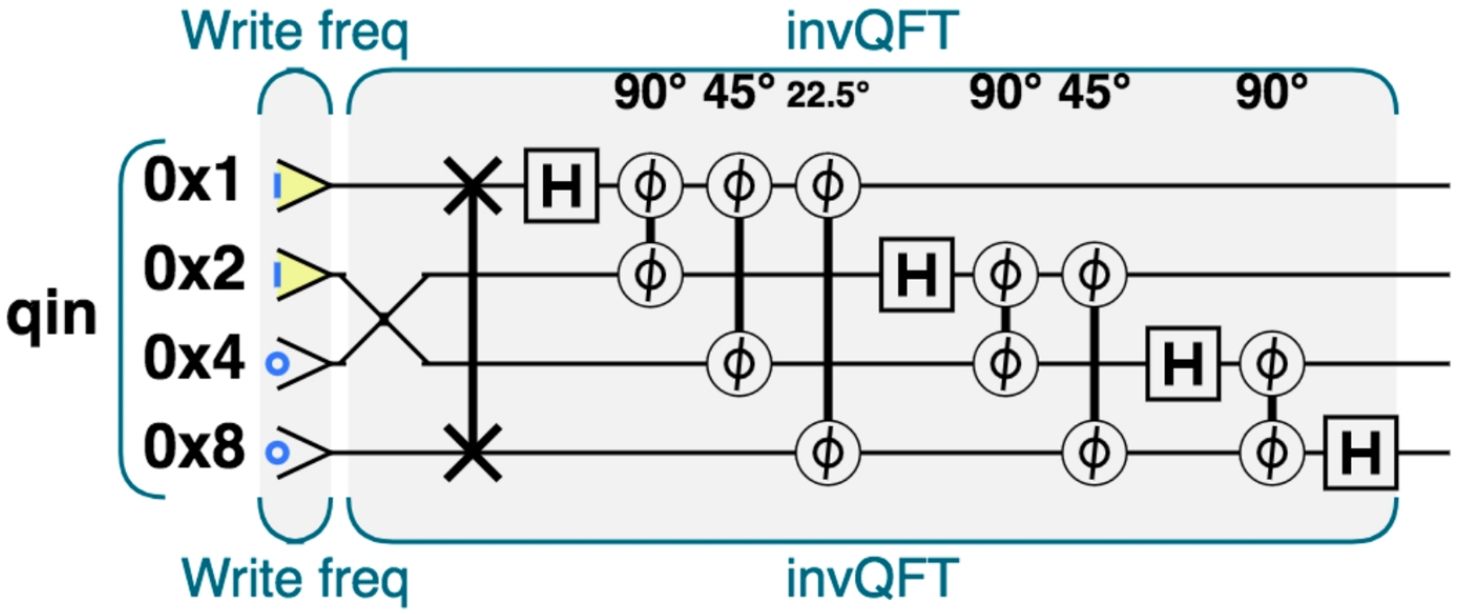
The FFT requires a number of operations that grows with the number of input bits n as $O(n2^n)$. The QFT, however - by leveraging the 2m states available in an m-qubit register- uses a number of gates that grows only as $O(m^2)$.



This means that for small signals (less than 22 bits, or about 4 million samples) the conventional FFT will be faster, even using just a laptop. But as the size of the input problem grows, the exponential advantage of the QFT becomes clear.

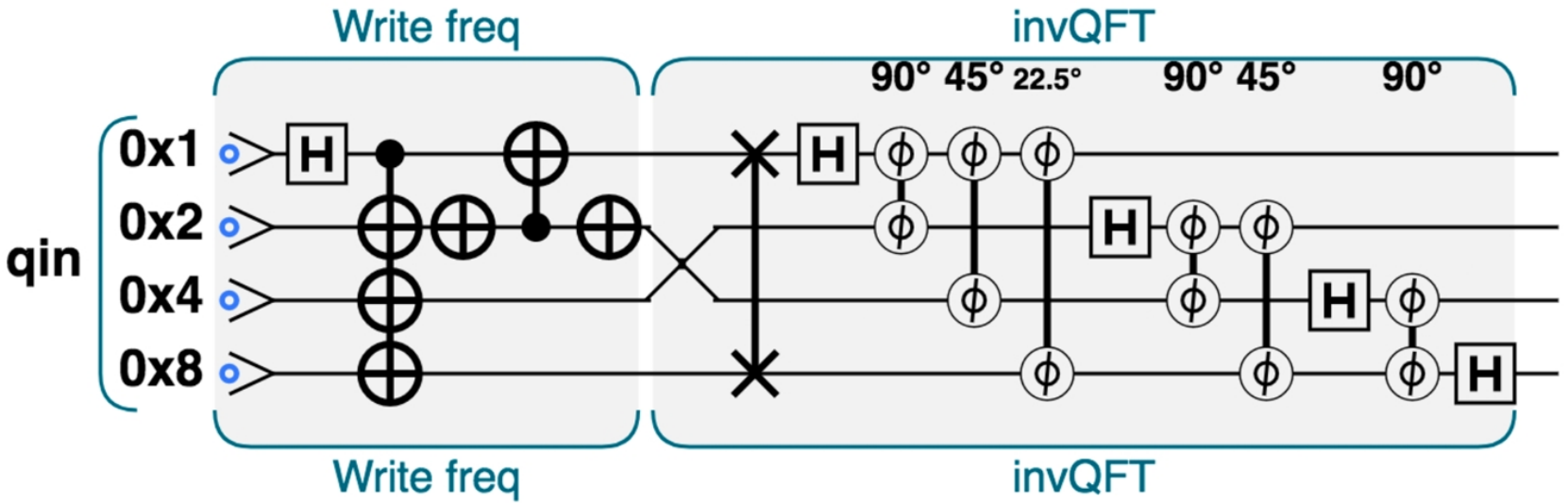
Preparing superpositions with the inverse QFT

We can use the invQFT to easily prepare periodically varying register superpositions:



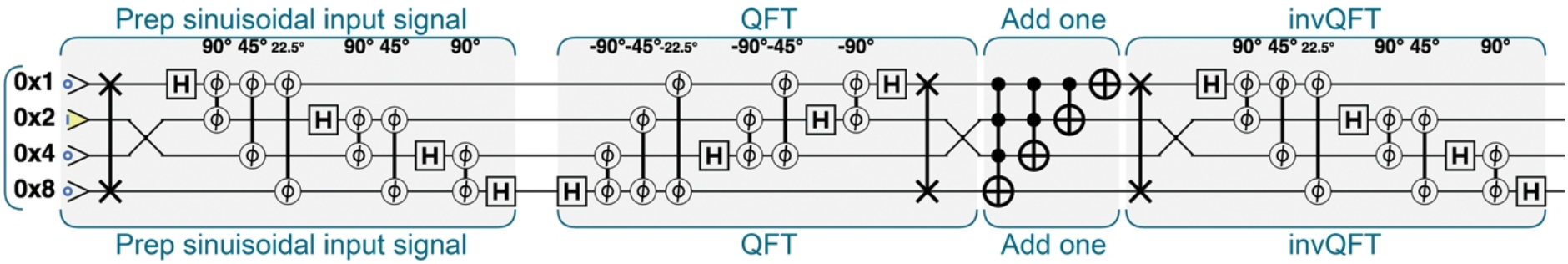
Preparing superpositions with the inverse QFT

The invQFT can also be used to prepare a QPU register that varies periodically in magnitude rather than relative phase. To do this we need only recall that a register with periodically varying magnitudes is a real signal, and so in frequency space we need a symmetric representation for the invQFT to act on:



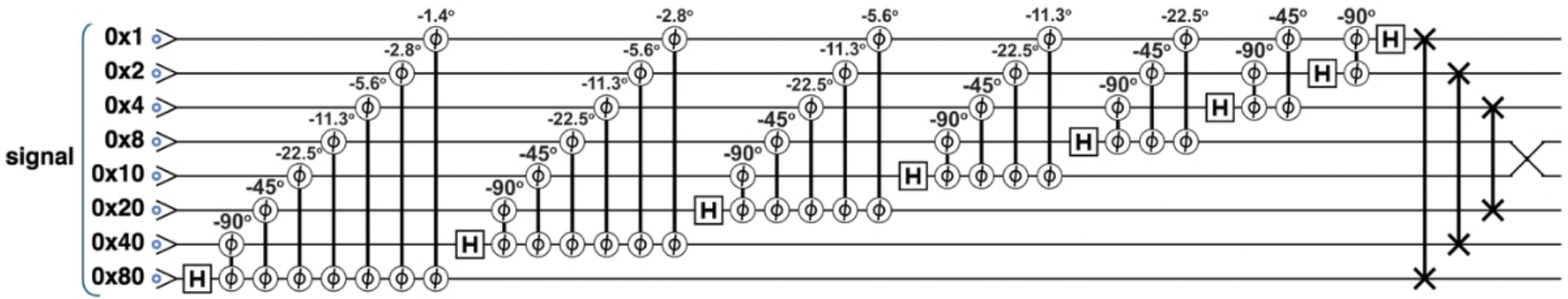
Alter frequency information with the inverse QFT

As well as preparing states with given frequencies, the invQFT also allows us to easily alter their frequency information. Suppose that at some point during an algorithm we want to increase the frequency with which the relative phases oscillate in our QPU register:



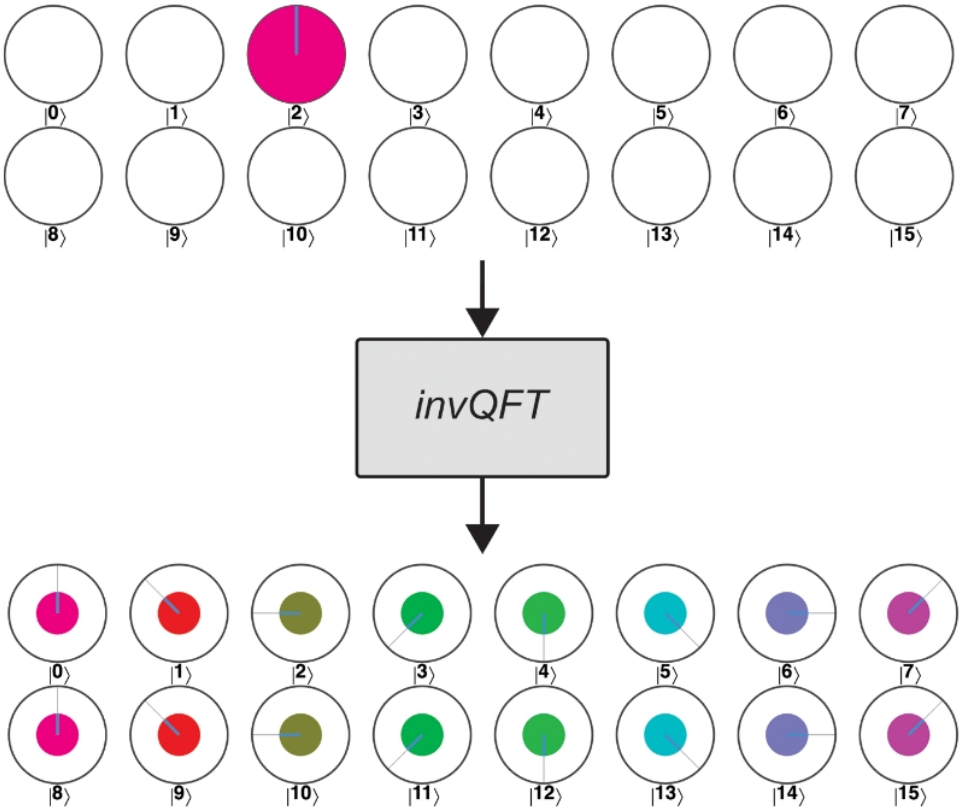
Inside the QPU

How the arrangement of simple QPU operations can extract the frequency components from a signal in the input register ?



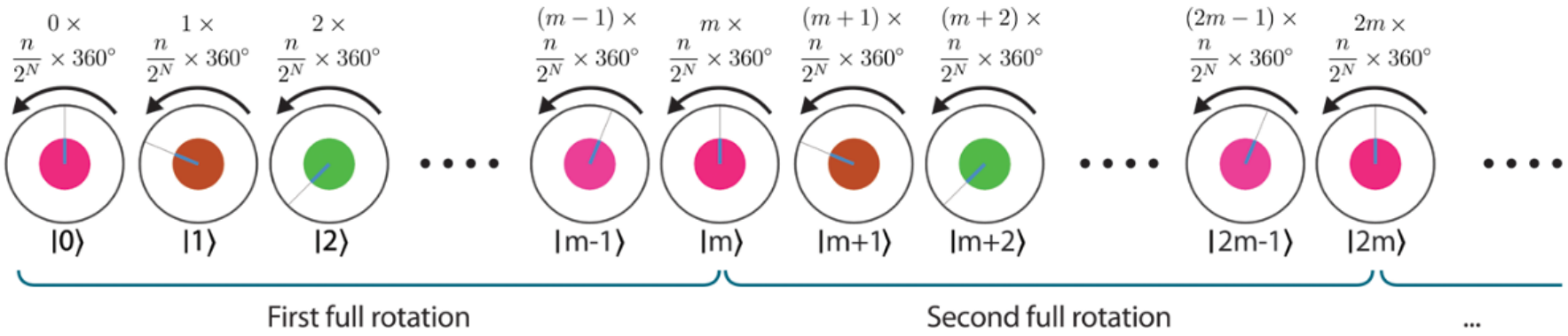
Understanding the inverse QFT

Trying to explain the inverse QFT is easier.



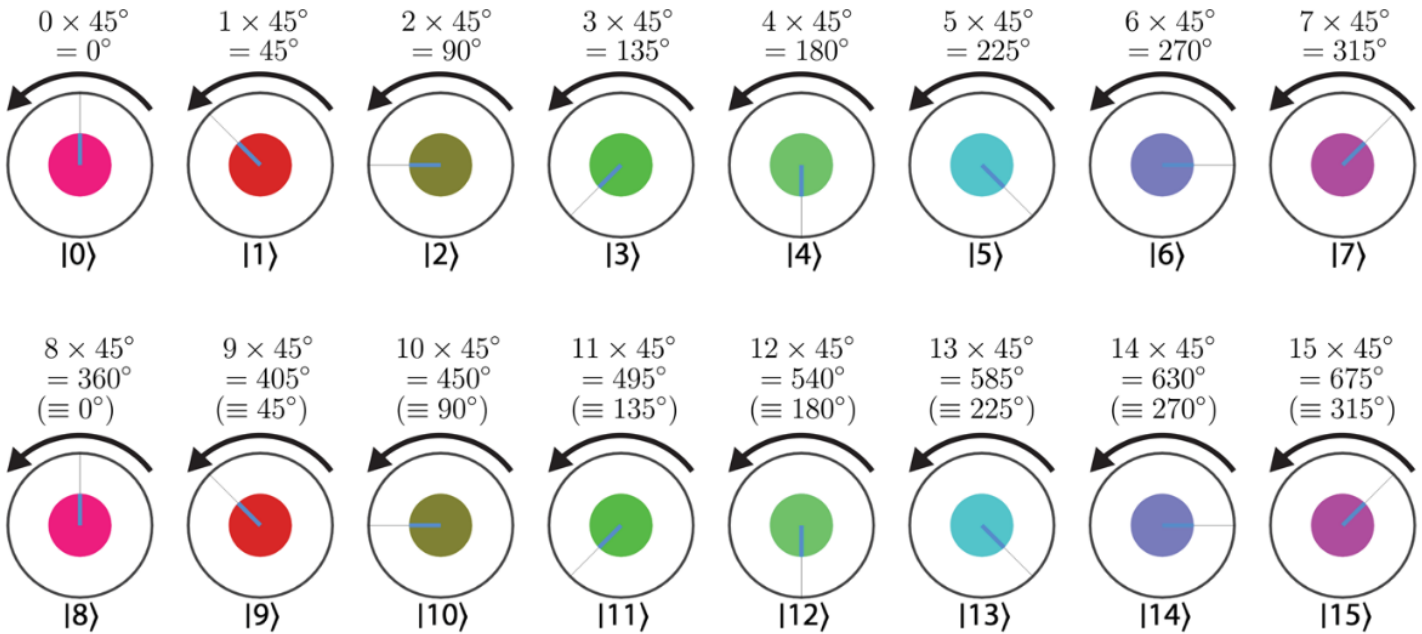
The intuition

Suppose we provide the inverse QFT with an N-qubit input register encoding some integer n. Scanning all along the output register 2^N values, we need each consecutive value's relative phase to rotate enough to perform a full $360° \times n/2^N$:



The intuition

For instance, with $N = 4$ and $n = 2$:



Requierements

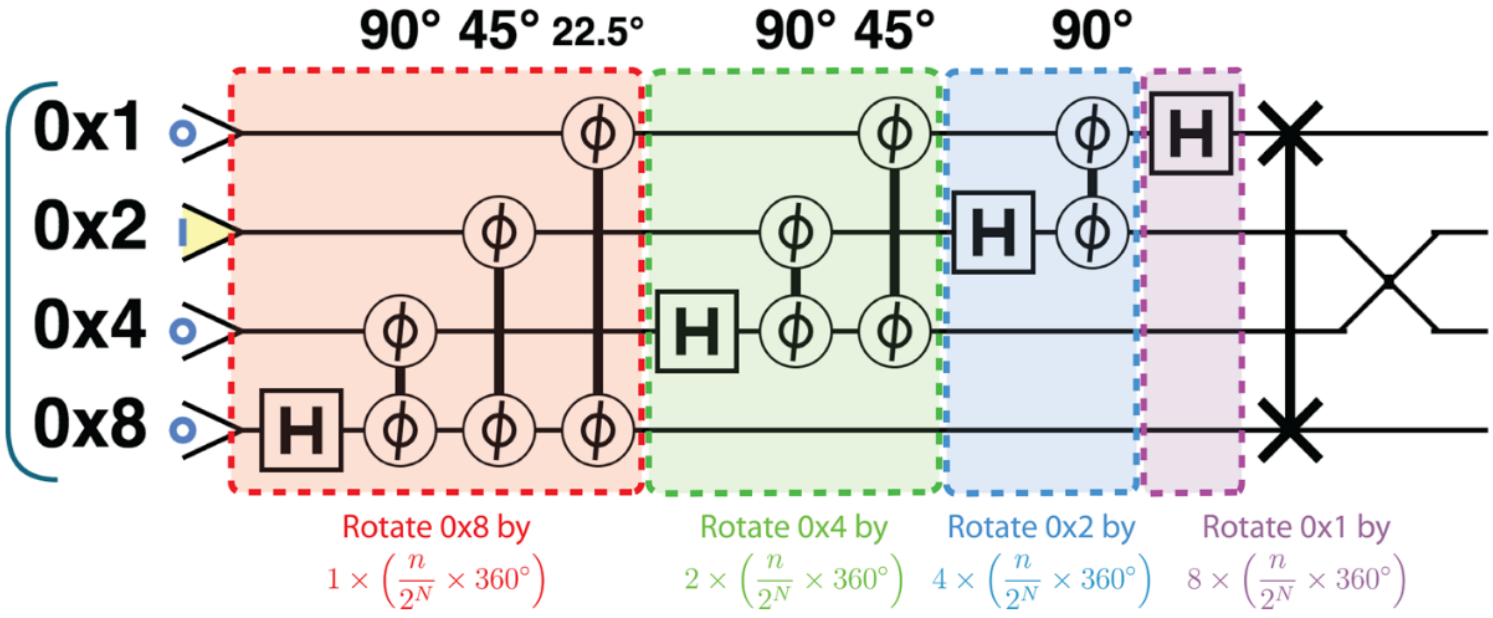
The inverse QFT is a cleverly cmpact way of implementing those two steps:

1) Determine the value of theta = 360 * (n/2), where n is the value initially stored in the register).

2) Rotate the phase of each circle in the register by a multiple of the angle found in the previous step, where the multiple is the circle's decimal value.

The inverse QFT subroutines

We see that the inverse QFT consists of four subroutines:



Each of these subroutines performs precisely the multiples rotations of increasing weighted that we mentioned

SWAP gates

If you're carrefully following you'll notice that this actually applies the rotations that we previously specified in the reverse order, that's why we apply SWAP gate at the end of the circuit.