DOTNEXT

Programming Quantum Computers in .NET using Q#

Rolf Huisman

Software Development Enablement Engineer

Rolf.Huisman@InfoSupport.com

@rlrhuisman

Agenda

Why do you need it ?

What is it ?

How can I program it ?

How would a Quantum Computer improve your life ?

Picture by Rolf Huisman / CC0 Creative Commons

Picture by Rolf Huisman / CC0 Creative Commons

Picture by Rolf Huisman / CC0 Creative Commons

Picture by Rolf Huisman / CC0 Creative Commons

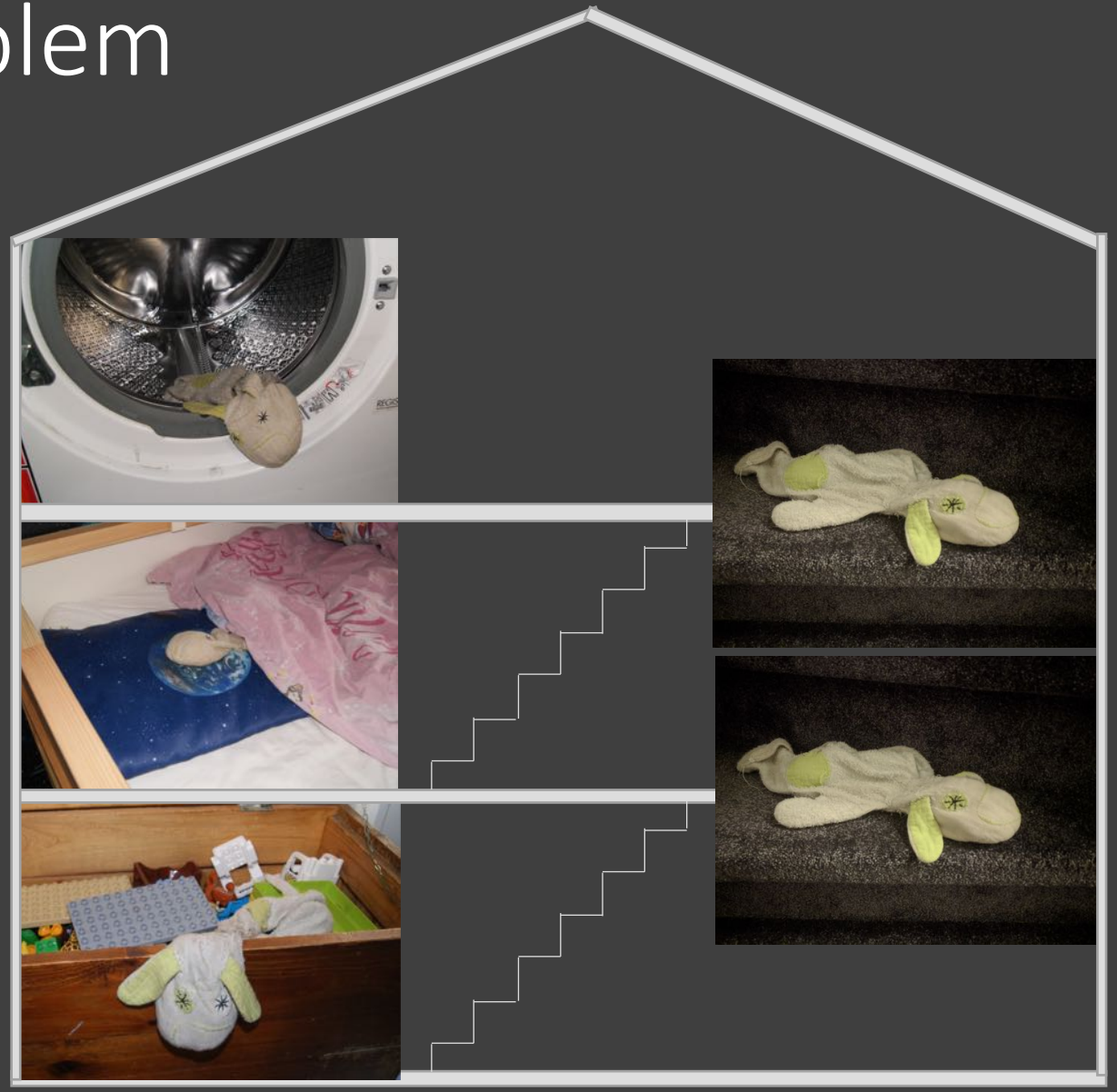
Internet Of Stuffed Animals (IOSA)

Stuffed animal problem

Attic

Bedroom

Living Room



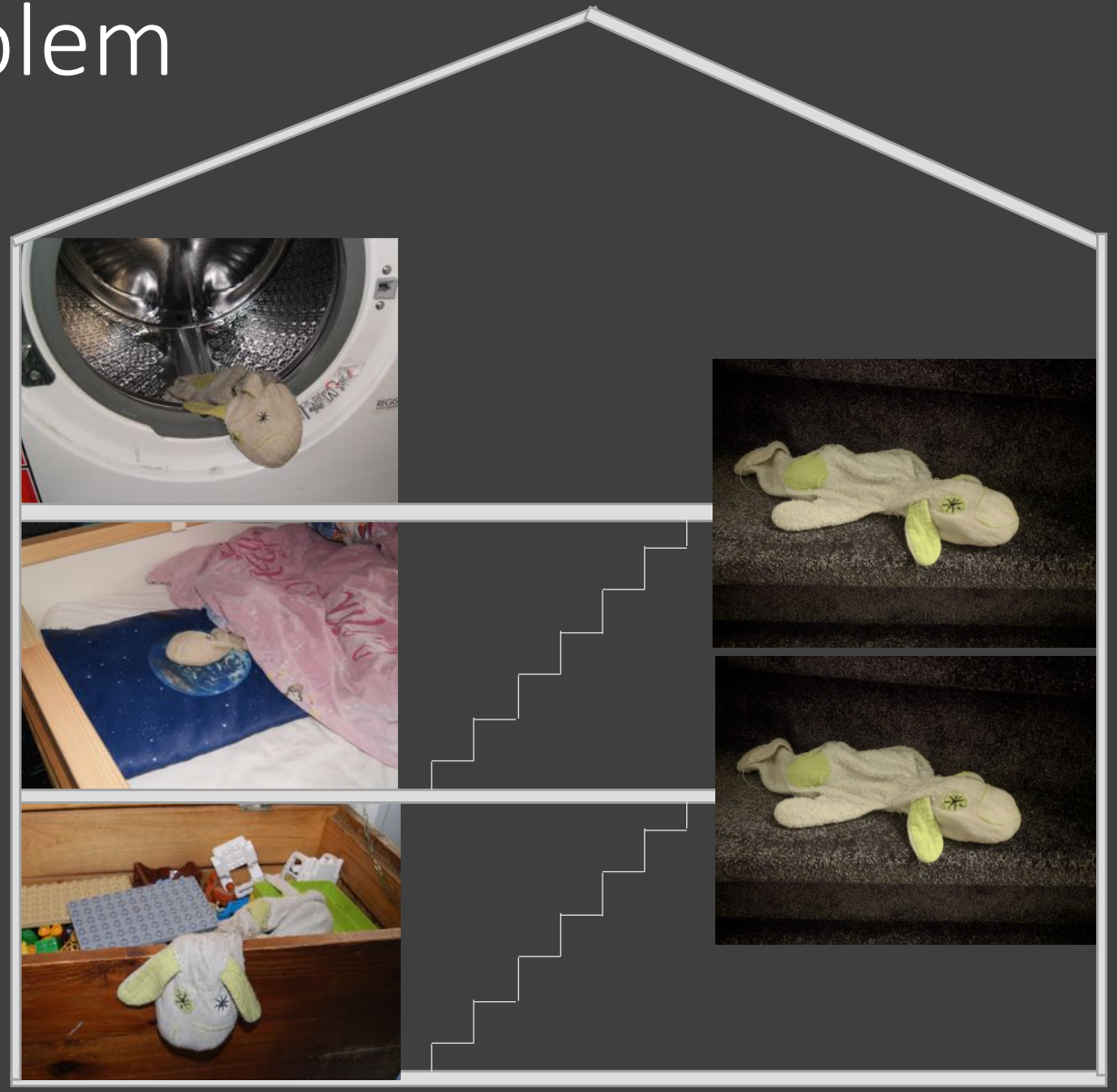
Picture by Rolf Huisman / CC0 Creative Commons

Stuffed animal problem

Attic

Bedroom

Living Room



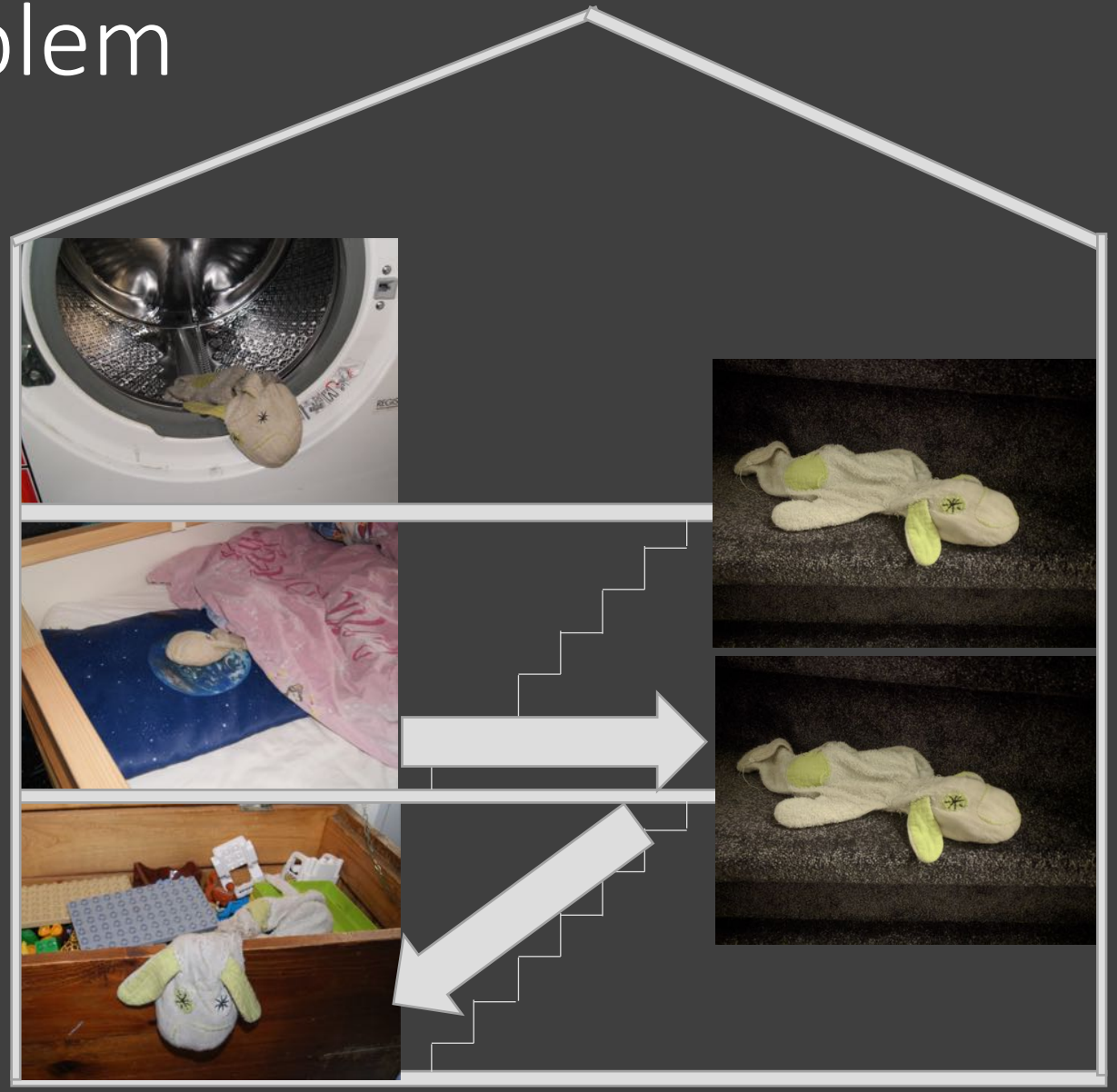
Picture by Rolf Huisman / CC0 Creative Commons

Stuffed animal problem

Attic

Bedroom

Living Room



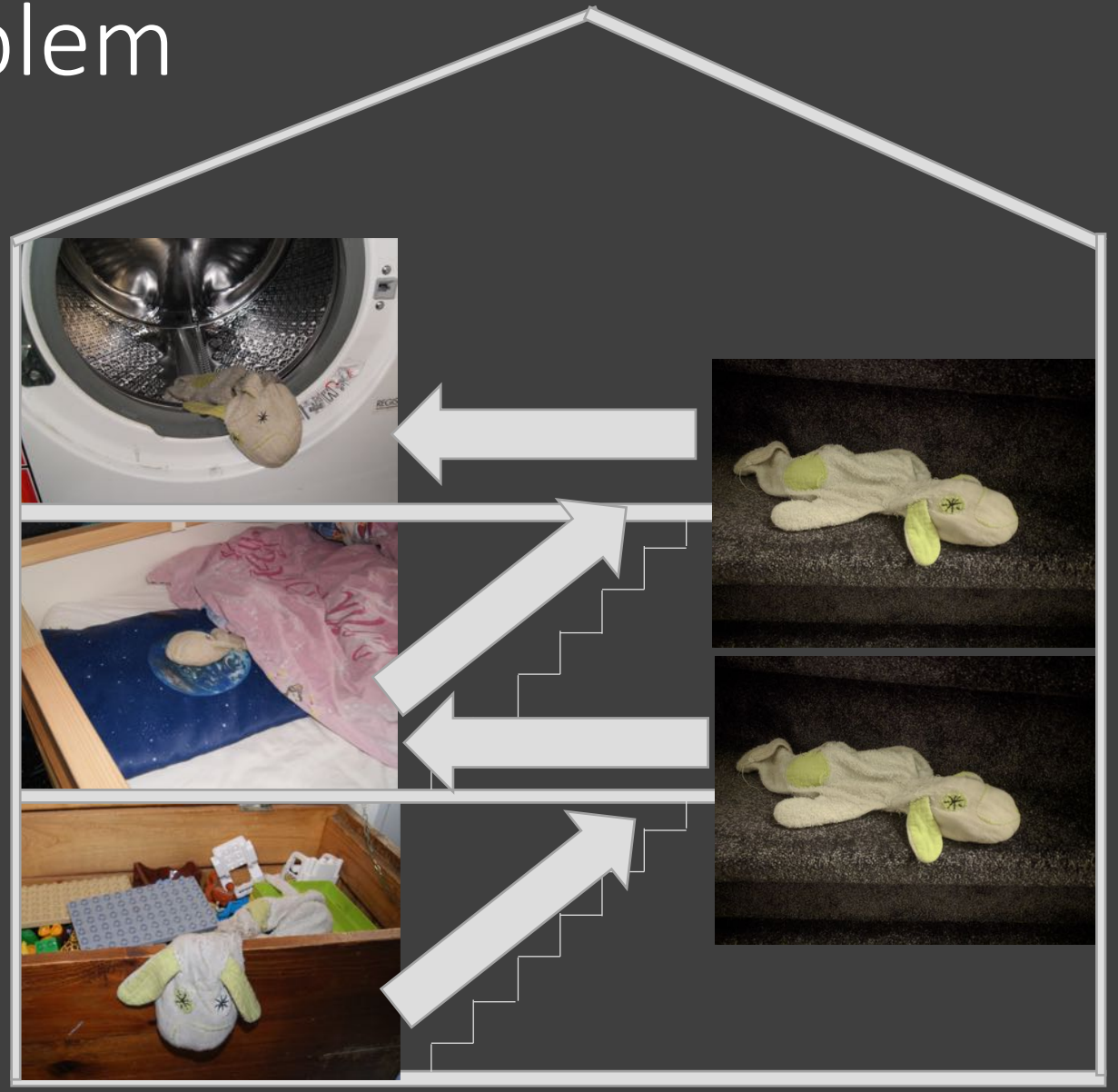
Picture by Rolf Huisman / CC0 Creative Commons

Stuffed animal problem

Attic

Bedroom

Living Room



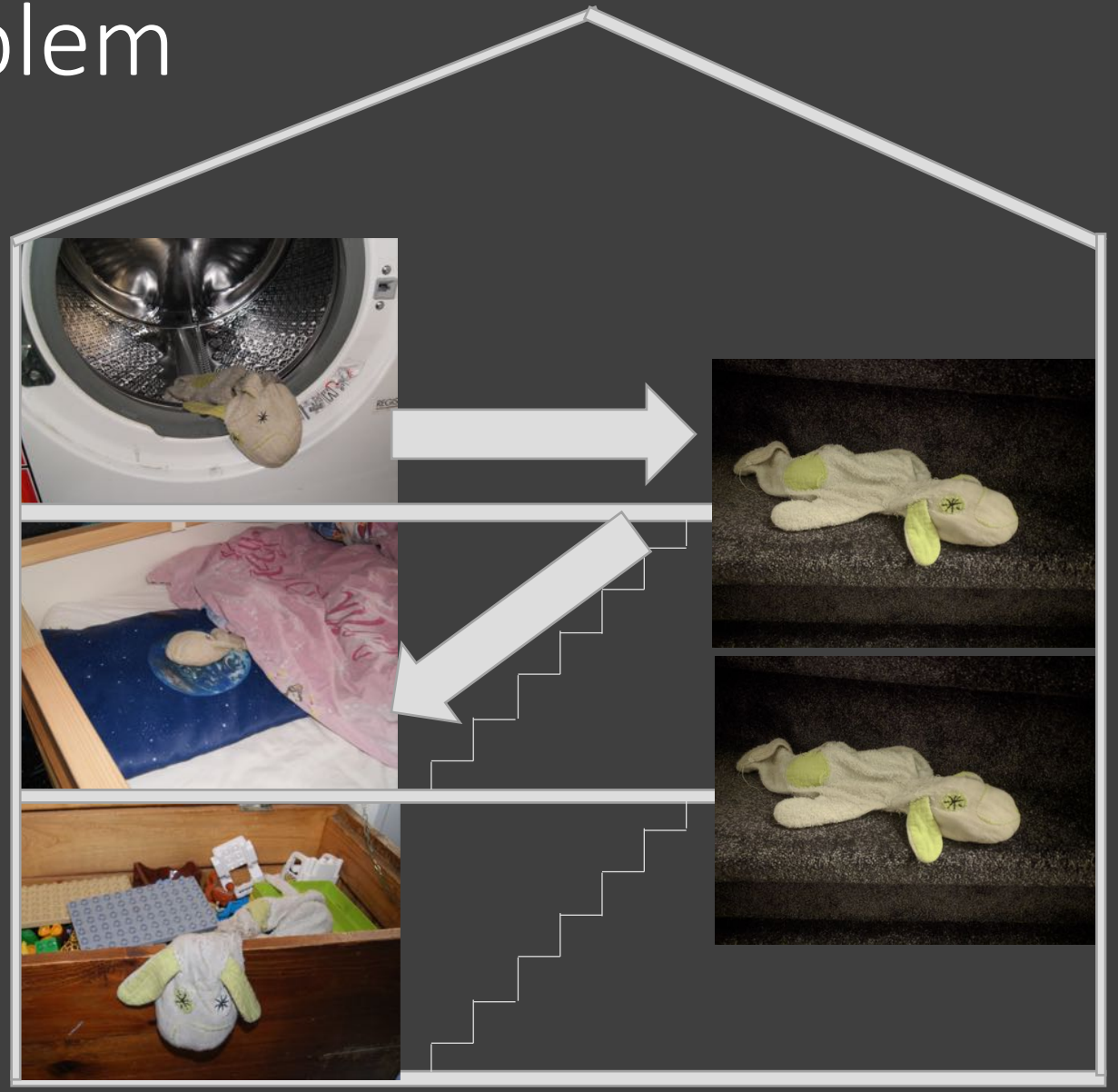
Picture by Rolf Huisman / CC0 Creative Commons

Stuffed animal problem

Attic

Bedroom

Living Room



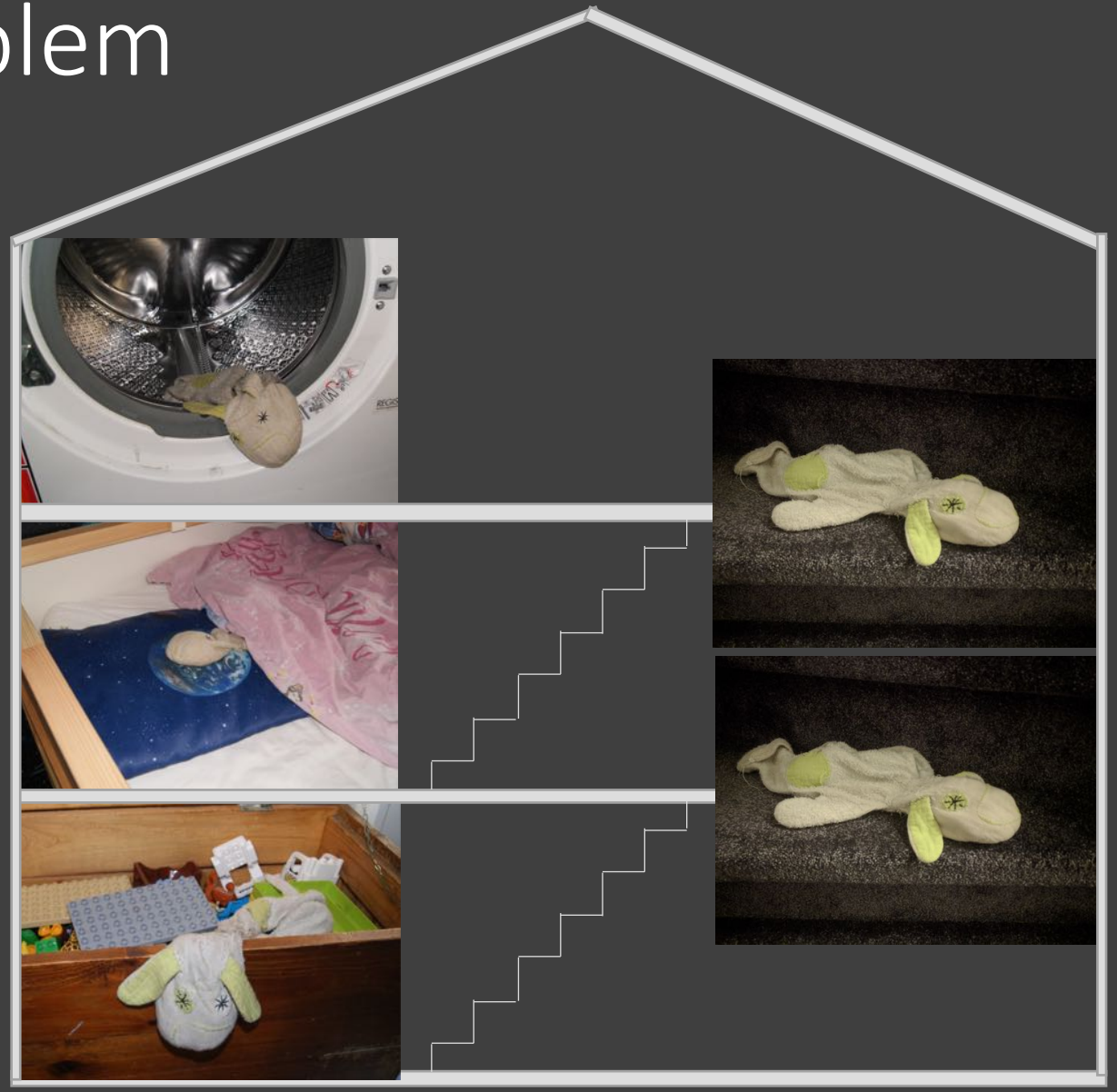
Picture by Rolf Huisman / CC0 Creative Commons

Stuffed animal problem

Attic

Search Path Optimization

Living Room

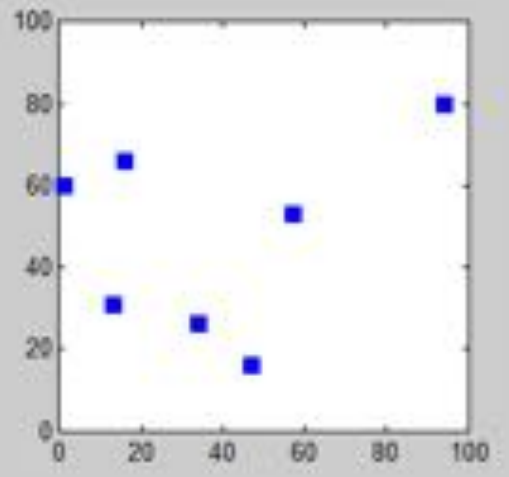


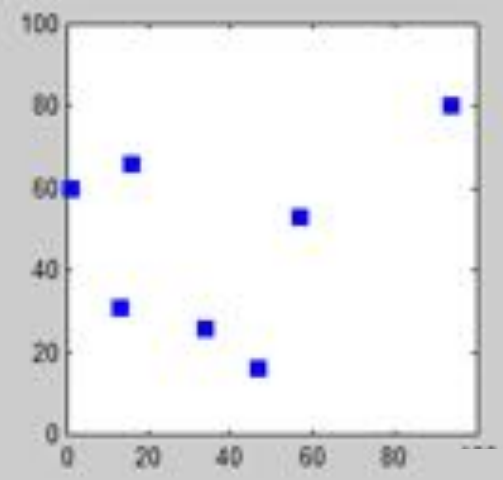
Picture by Rolf Huisman / CC0 Creative Commons

Same problem as other fields

Picture by Geralt / CC0 Creative Commons

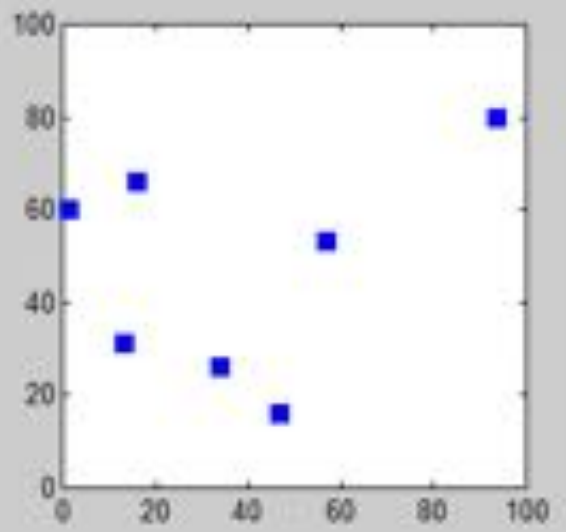
Traveling Salesman problem

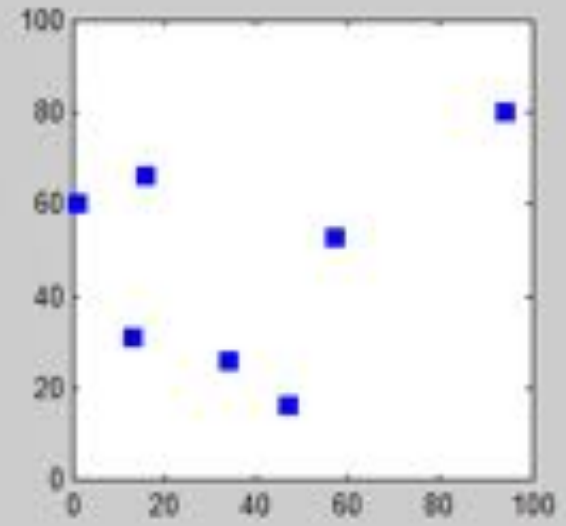




NP Hard

$O(2n*2^n)$





Add more hardware

No problem,

Can scale using cloud providers

Right ?

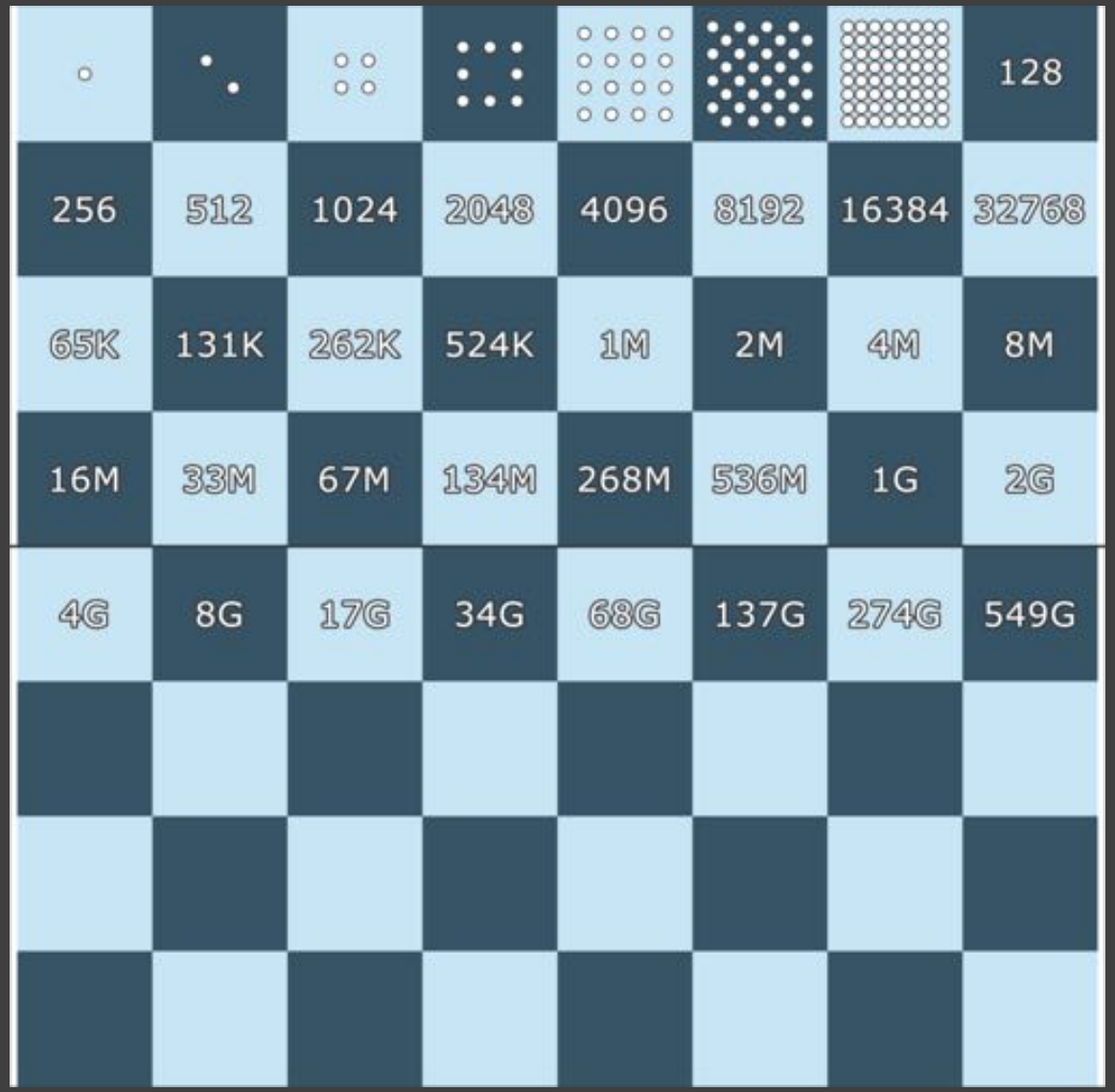
Google Cloud Platform

The power of exponents

$O(2n*2^n)$

Using 1000 machines, only makes it 1000 times quicker,

if you're lucky



And there are more fields that have this issue

Artificial Intelligence

Weather Prediction

DNA Simulations

New plan

Quantum computing

The computer that uses Qubits

New plan

Quantum computing

The computer that uses Qubits

Qubit can take value |0>, |1>, or **both at once**,

a phenomenon known as **superposition**

Speed

A Computer with N Qubits,
can be in a superposition of 2^n logical states.

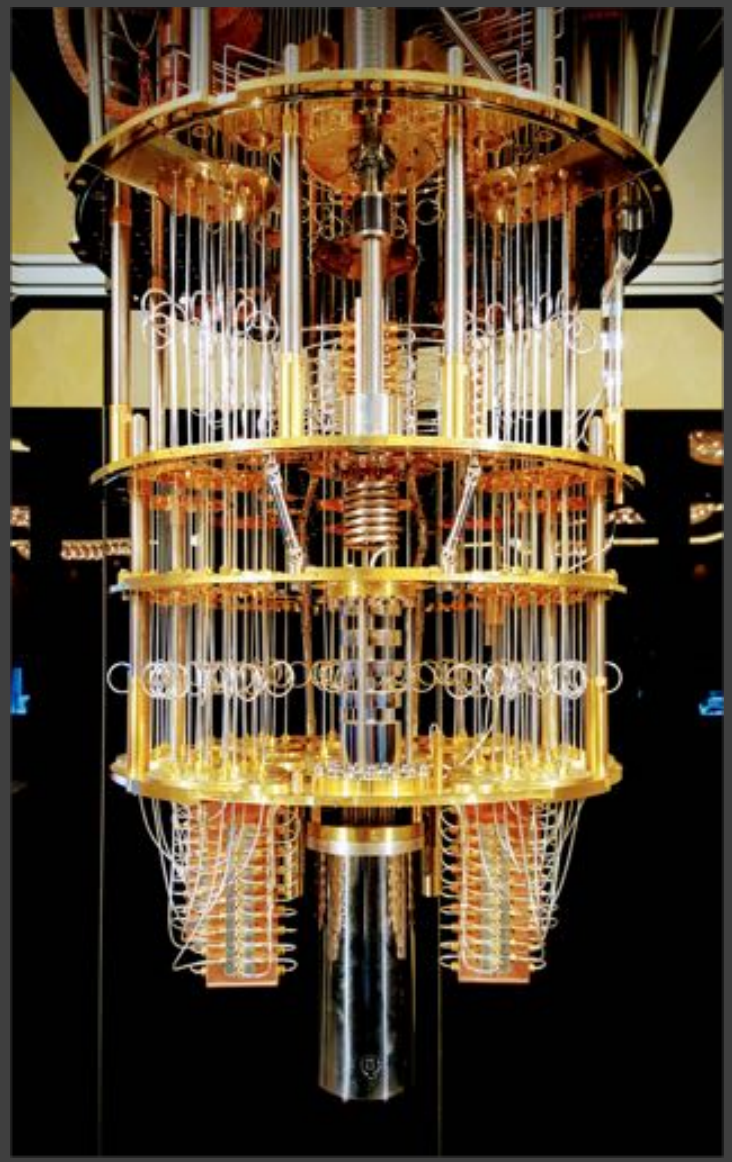
Speed

A Computer with N Qubits,
can be in a superposition of 2^n logical states.

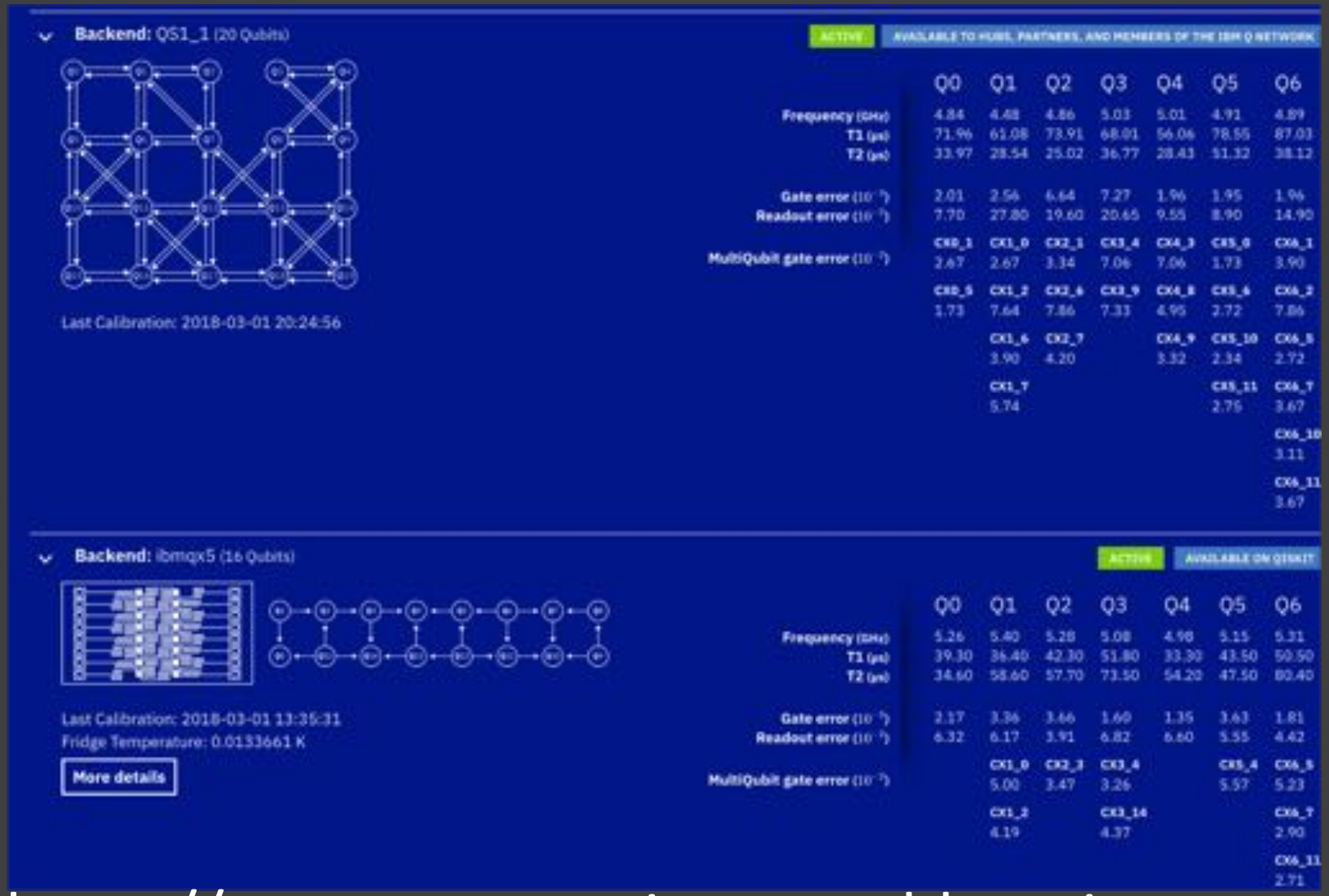
So can work with these at the same time.

Multi-Billion Investments

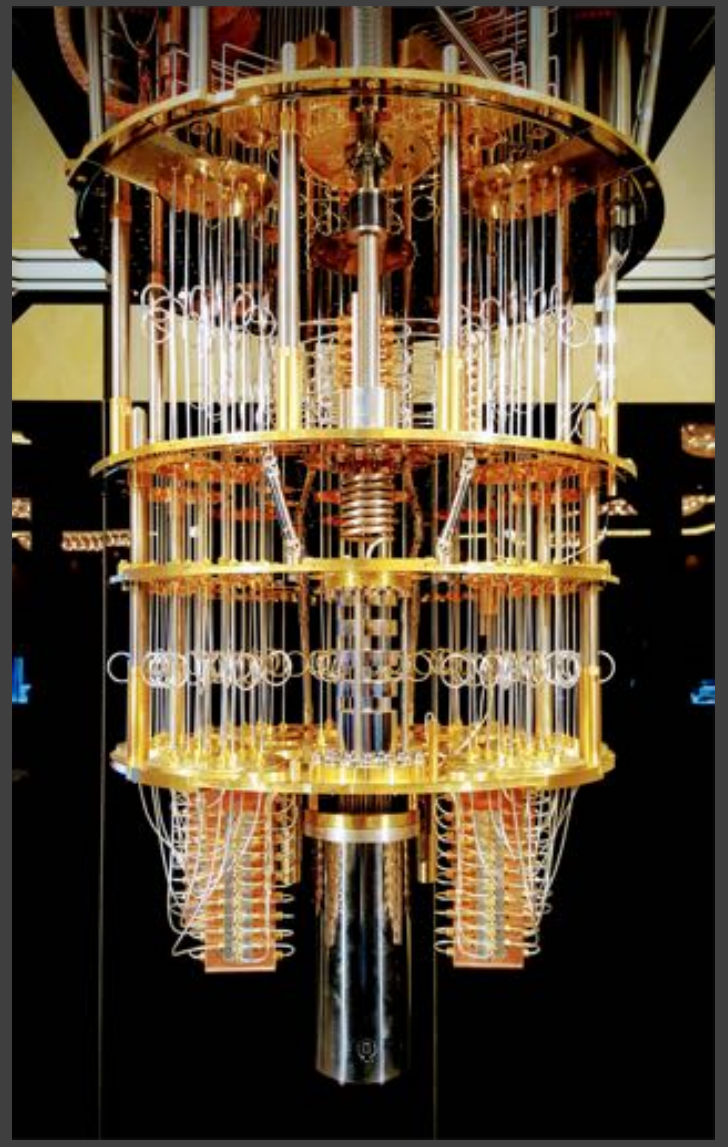
What does a Quantum Computer look like ?



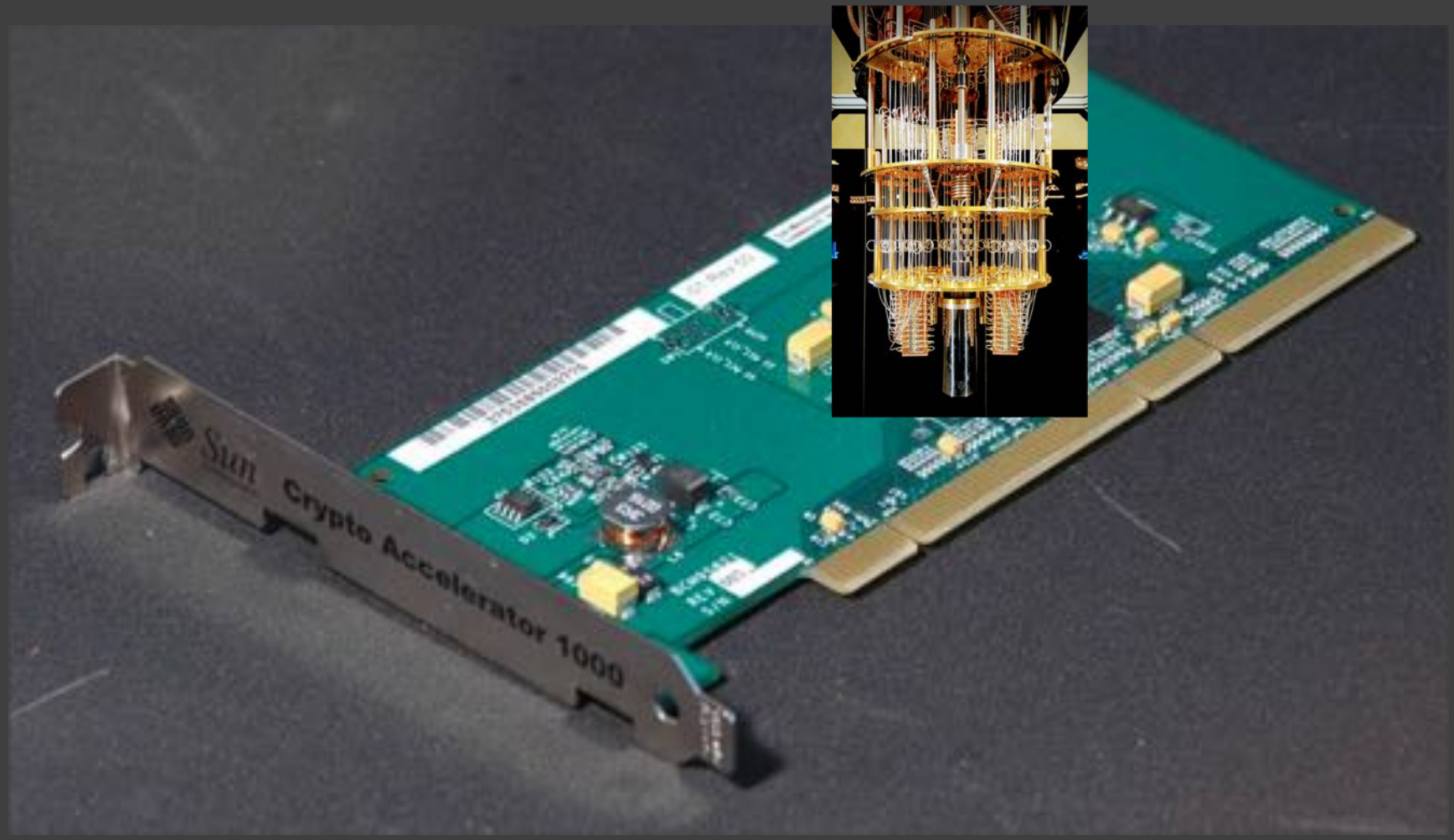
IBM Q



https://quantumexperience.ng.bluemix.net

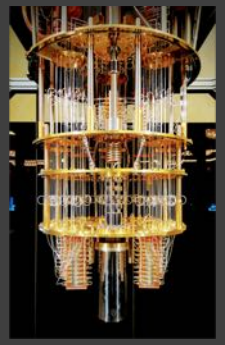


“Quantum Accelerator card”



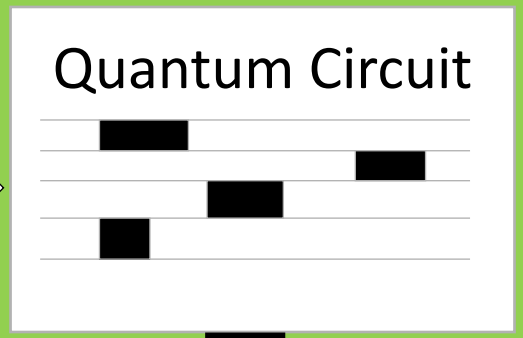
Quantum Computing Workflow

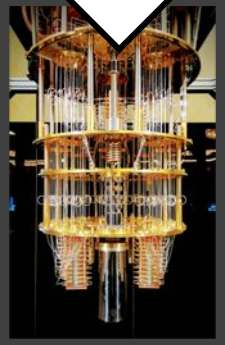
Traditional
Computer



Quantum Computing Workflow

Traditional Computer

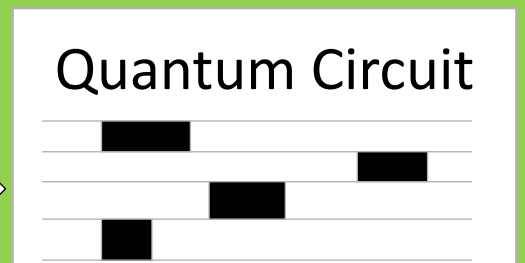


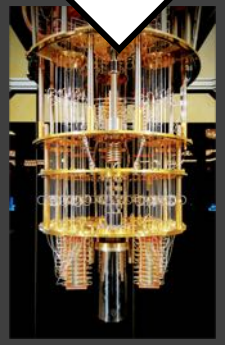


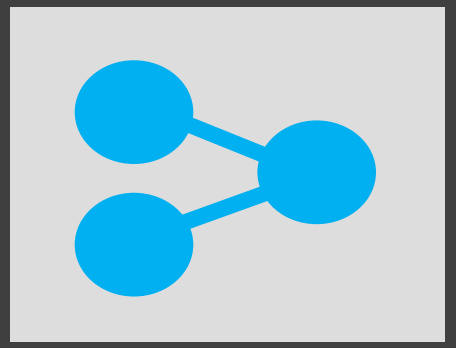
Quantum Computing Workflow

Traditional Computer

Quantum Circuit







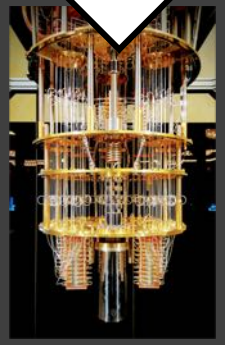
Quantum Computing Workflow

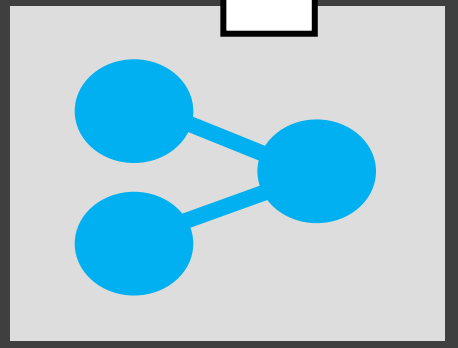
Traditional Computer

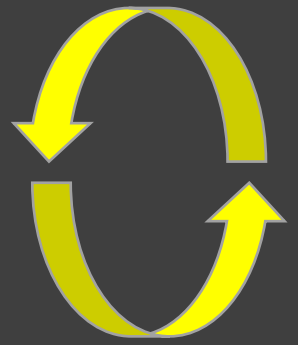
Quantum Circuit

Measurements

11 10 01 00







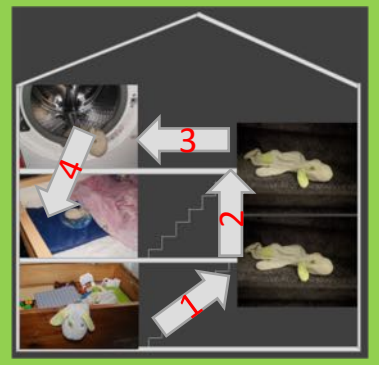
Quantum Computing Workflow

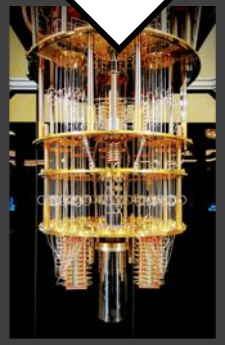
Traditional Computer

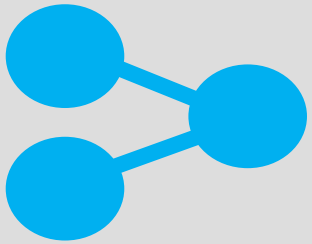
Quantum Circuit

Measurements

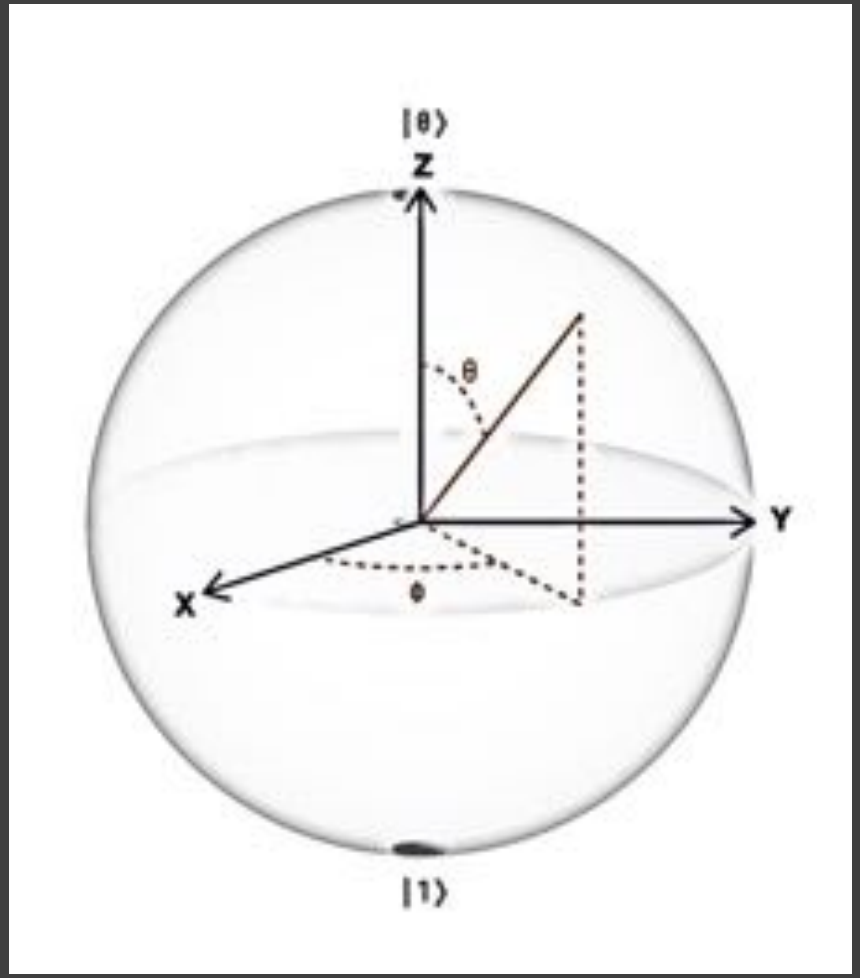
11 10 01 00







The Qubit in more detail



Bloch Sphere

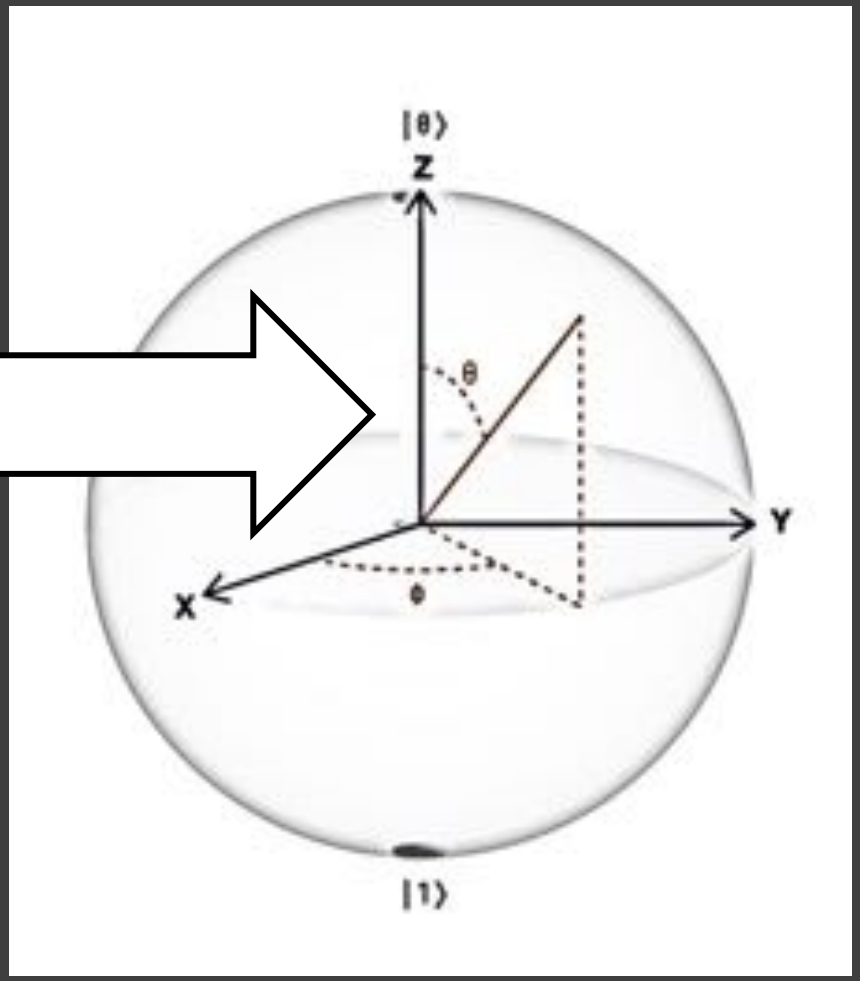
Picture by IBM / Licensed Use

The Qubit in more detail

Angle Θ

|0> 0.75

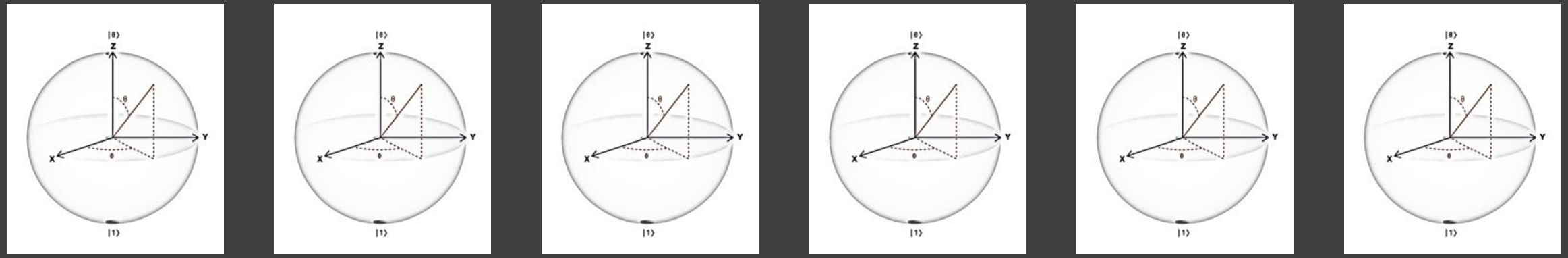
|1> 0.25



Bloch Sphere

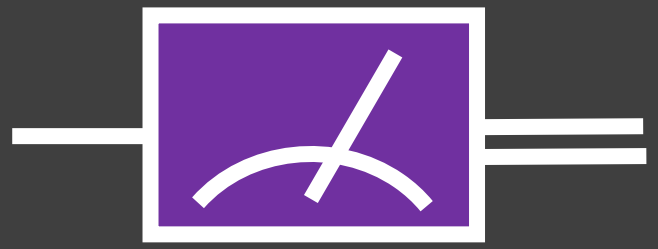
Picture by IBM / Licensed Use

M (Measurement Gate)



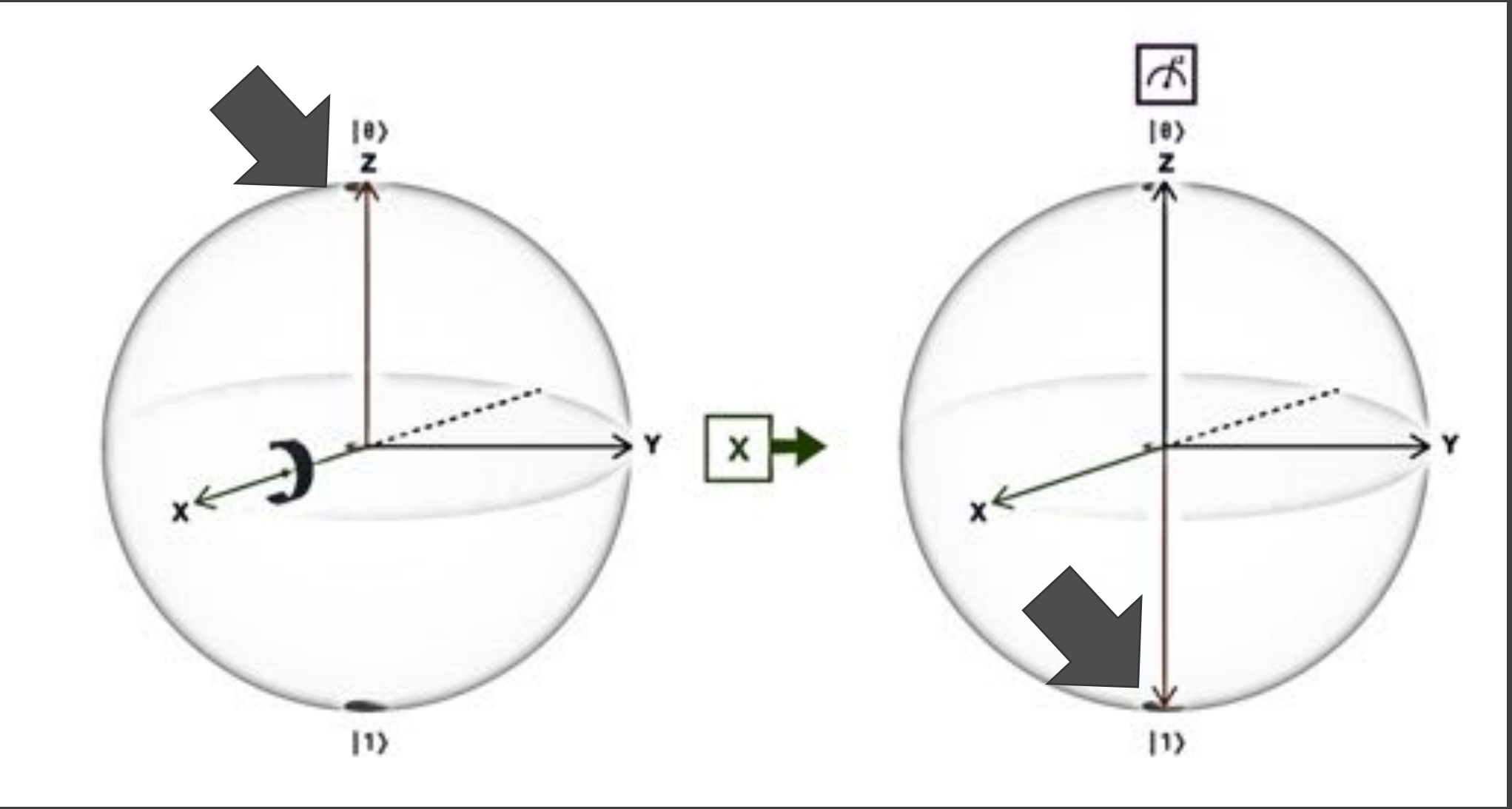
|0> |0> |1> |0> |0> …

|0> ~75%
|1> ~25%



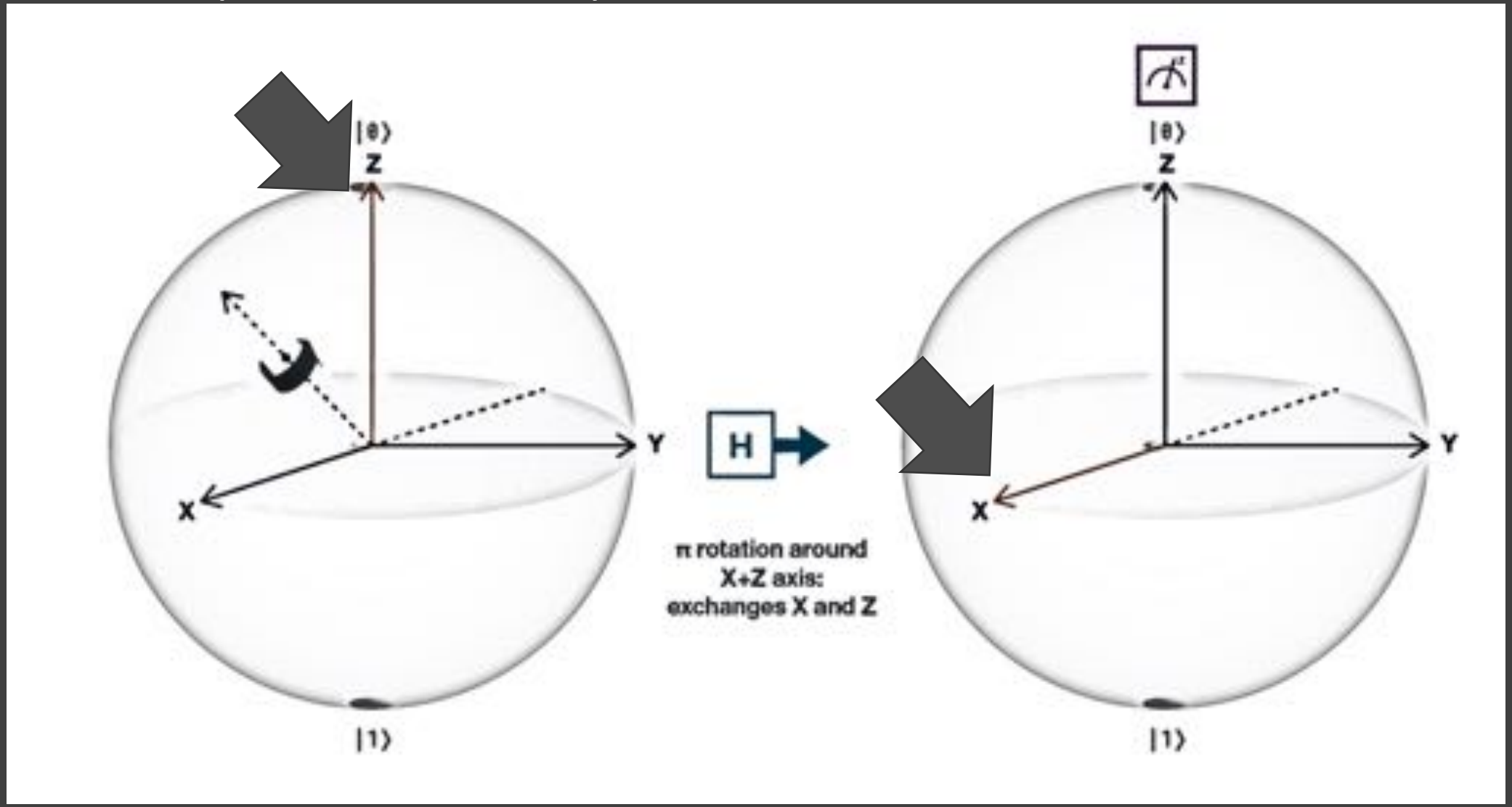
Picture by IBM / Licensed Use

X Gate (Not)



Picture by IBM / Licensed Use

H Gate (Hadamard)



Picture by IBM / Licensed Use

One Qubit Gates

X-Gate (Bit-flip, Not Gate)

Y-Gate

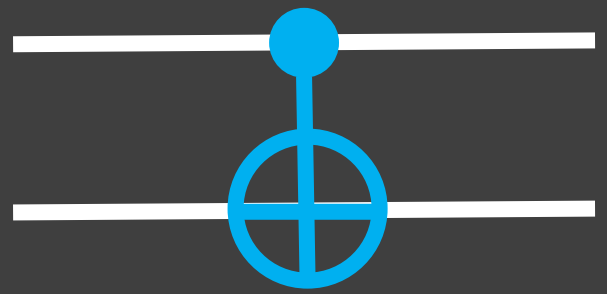
Z-Gate (Phase flip)

H-Gate (Hadamard, Creates Superposition)

S-Gate, T-Gate (Phase shift)

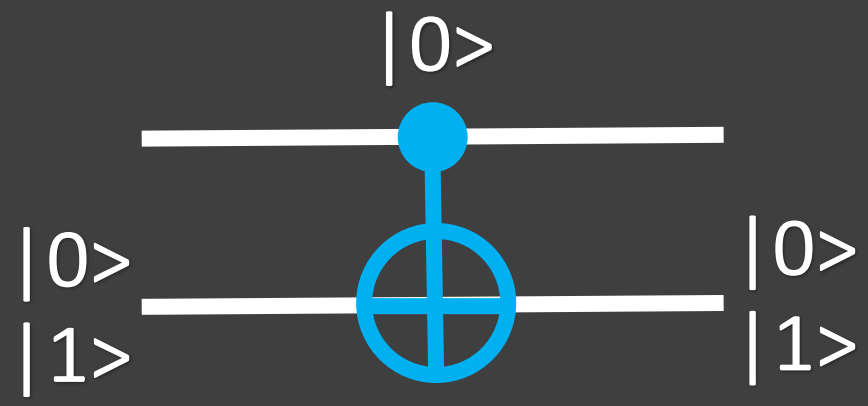
Multiple Qubit Gates

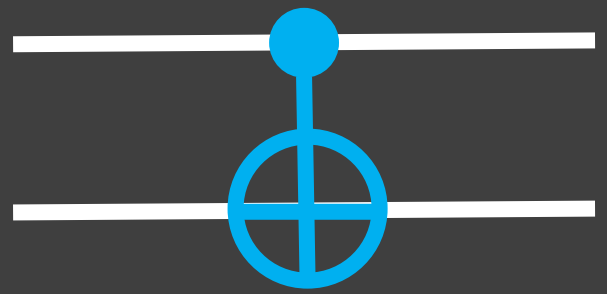
CNOT (Controlled Not Gate)



Multiple Qubit Gates

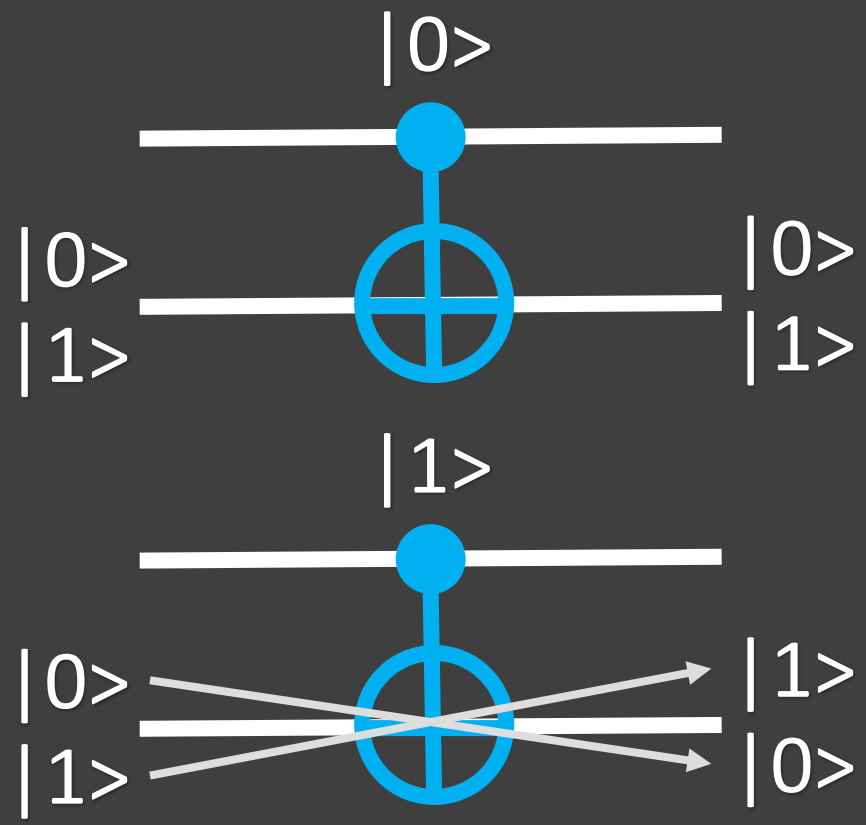
CNOT (Controlled Not Gate)

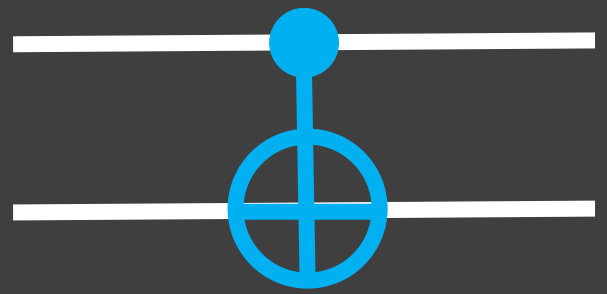




Multiple Qubit Gates

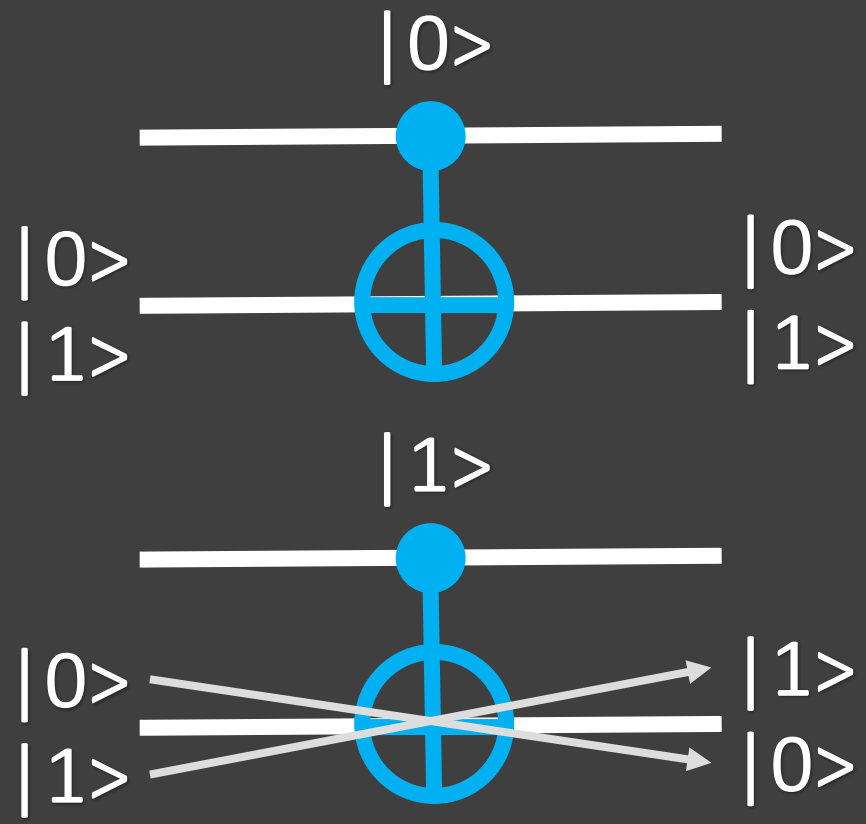
CNOT (Controlled Not Gate)

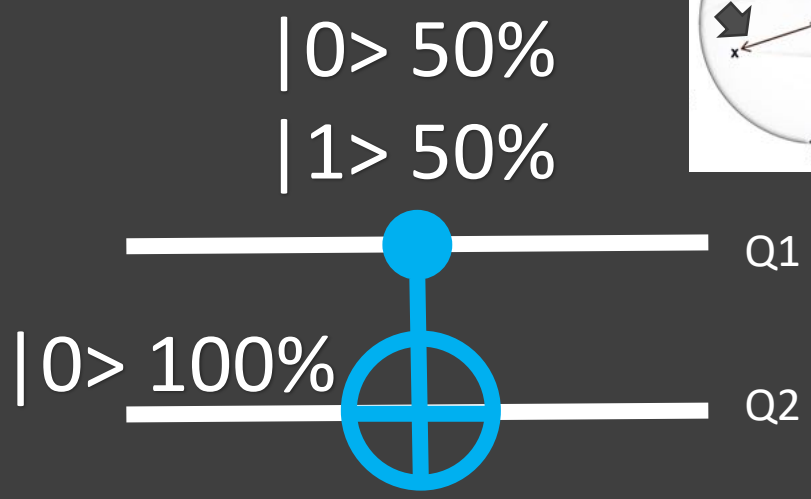


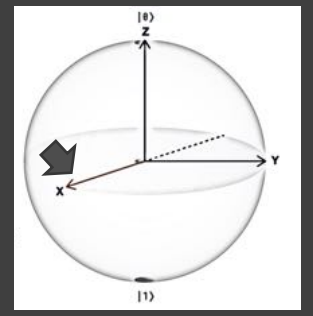


Multiple Qubit Gates

CNOT (Controlled Not Gate)

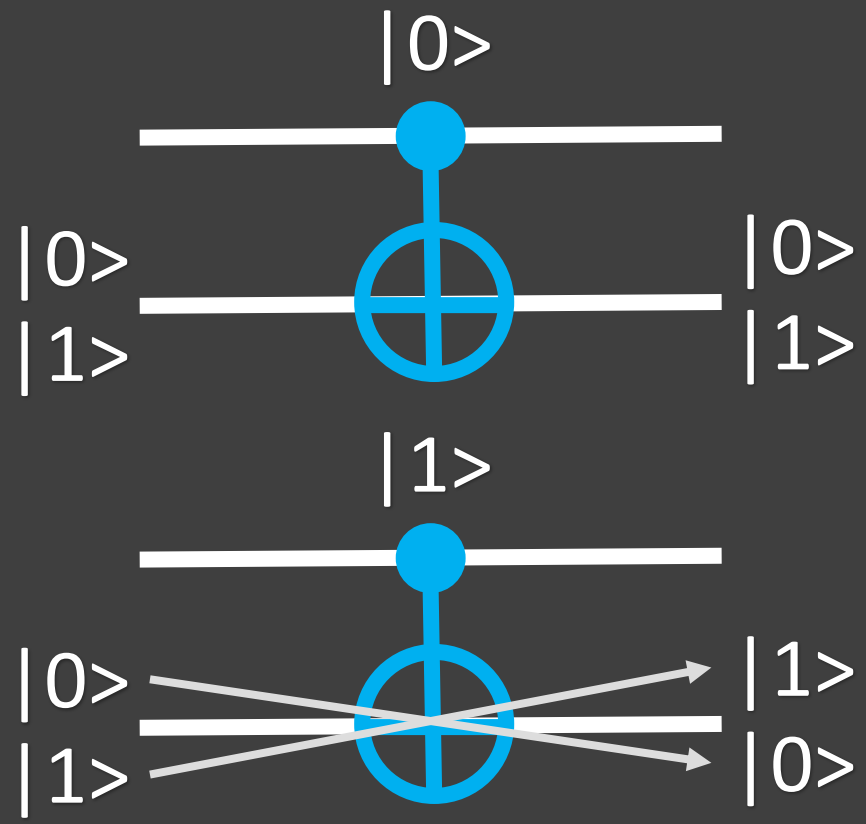


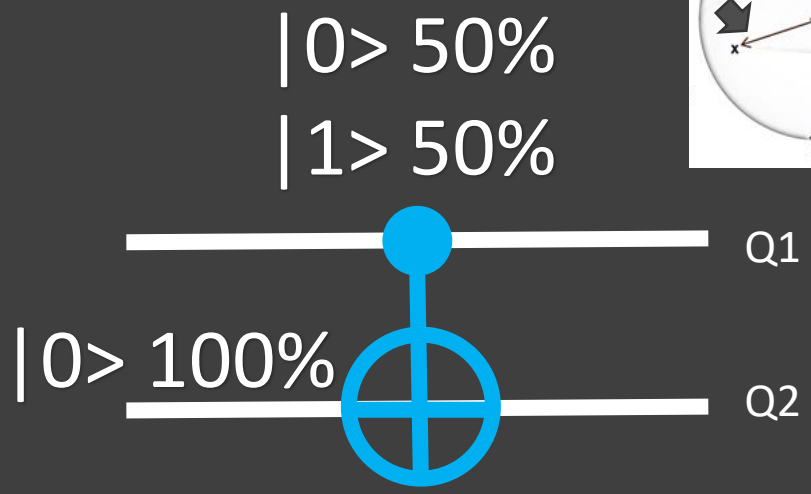


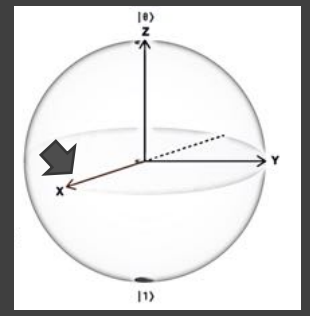


Multiple Qubit Gates

CNOT (Controlled Not Gate)







		Q1	
		|0>	|1>
Q2	|0>	50%	0%
	|1>	0%	50%

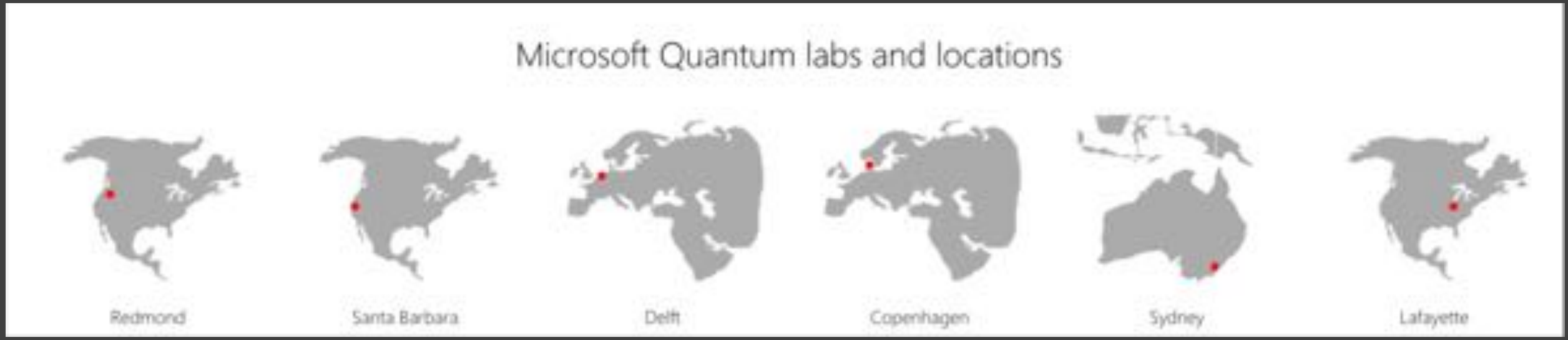
How can I program for this ?

Microsoft Q#

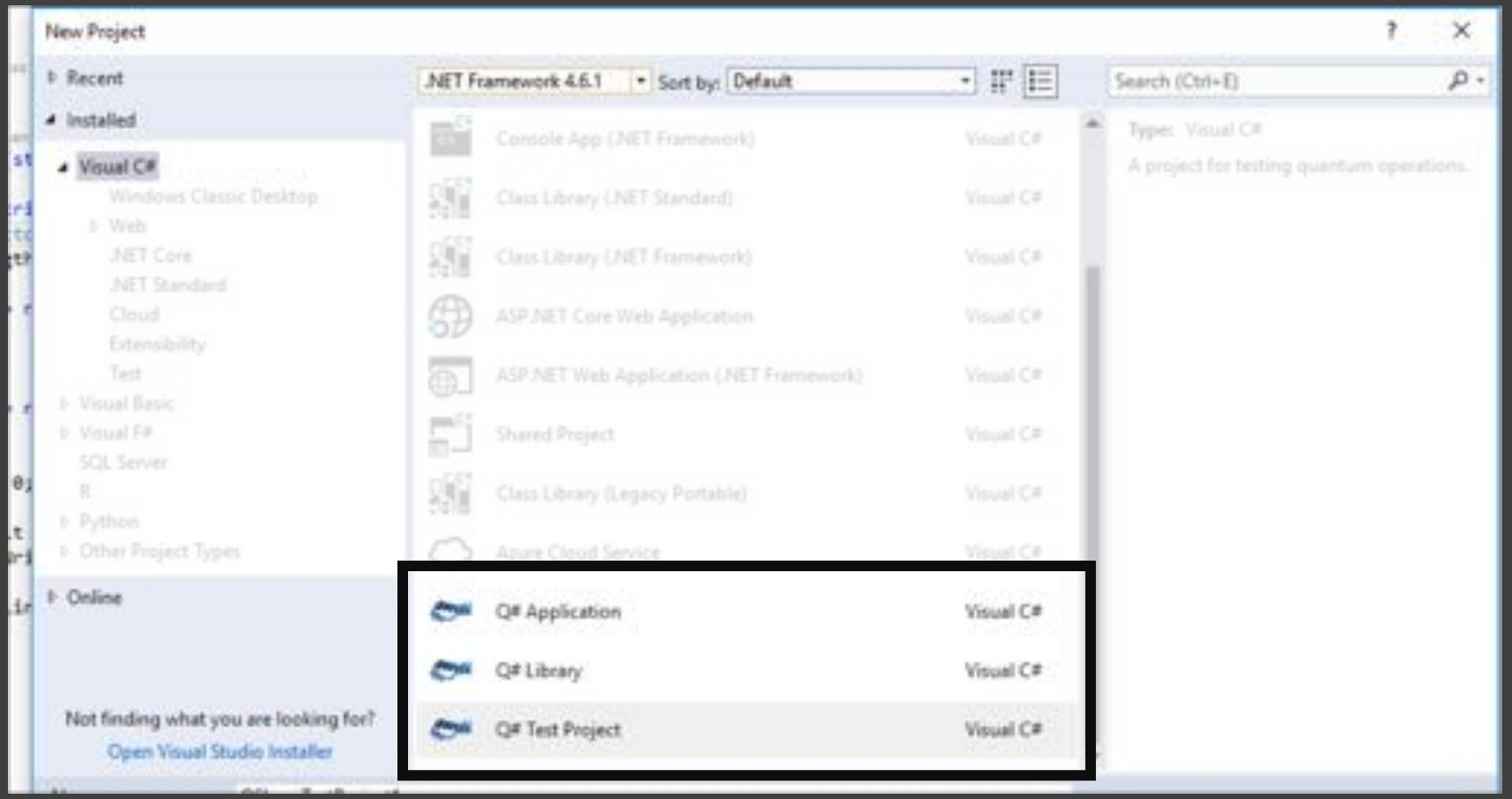
Revealed at Ignite 2017

Microsoft Quantum (former Station Q)

Previously known as F# extension **LIQUi|>**

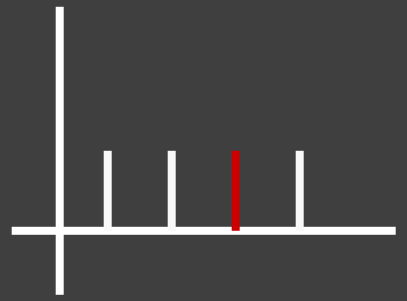


Microsoft Q#

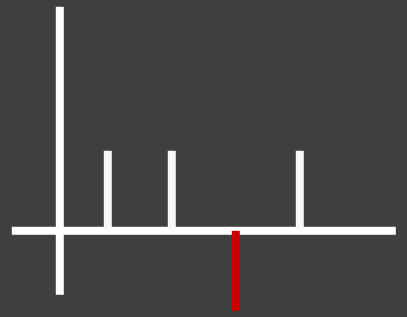

New Project
Recent
Installed
Visual C#
Windows Classic Desktop
Web
.NET Core
.NET Standard
Cloud
Extensibility
Test
Visual Basic
Visual F#
SQL Server
R
Python
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
.NET Framework 4.6.1
Sort by: Default
Search (Ctrl+E)
Console App (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (.NET Framework) Visual C#
ASP.NET Core Web Application Visual C#
ASP.NET Web Application (.NET Framework) Visual C#
Shared Project Visual C#
Class Library (Legacy Portable) Visual C#
Q# Application Visual C#
Q# Library Visual C#
Q# Test Project Visual C#
Type: Visual C#
A project for testing quantum operations.

DEMO

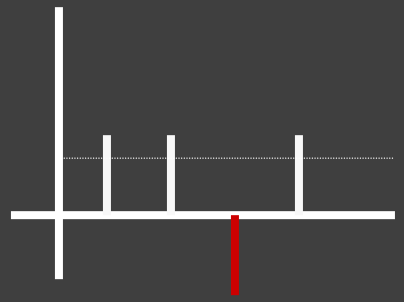
Grover (and other amplifications)



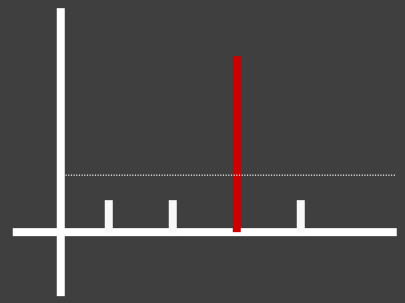
Original Amplitudes



Negate Amplitude



Average of all Amplitudes



Flip all Amplitudes around Avg

Grover execution: informally $\sqrt{states}$

ReflectionOracle user defined type

Namespace: Microsoft.Quantum.Canon

Represents a reflection oracle O, where the inputs are

- The phase ϕ by which to rotate the reflected subspace.
- The qubit register on which to perform the given reflection.

Q# Copy

```
newtype ReflectionOracle = ((Double, Qubit[]) => () : Adjoint, Controlled);
```

AmpAmpByOracle function

Namespace: Microsoft.Quantum.Canon

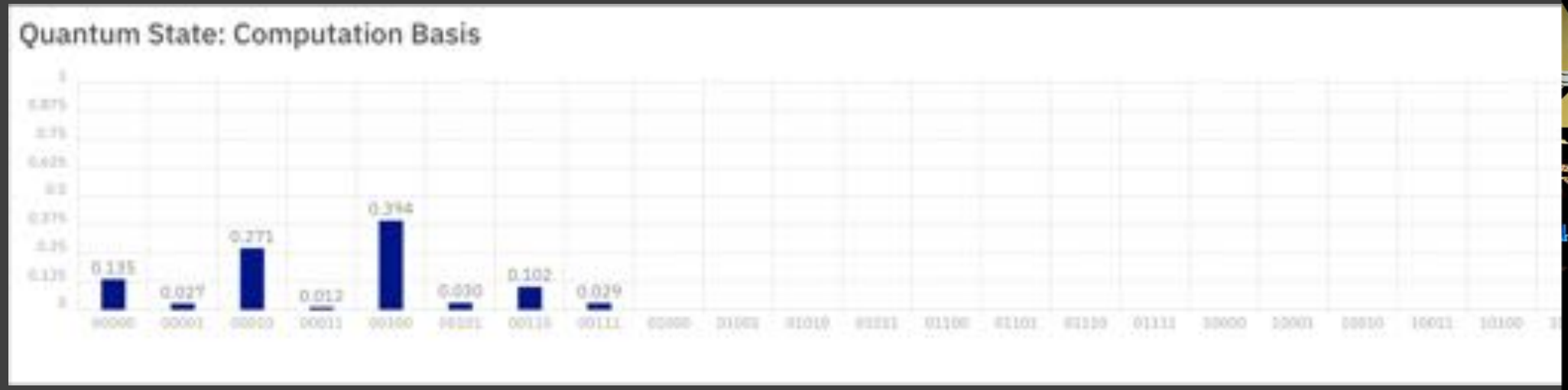
Standard Amplitude Amplification algorithm

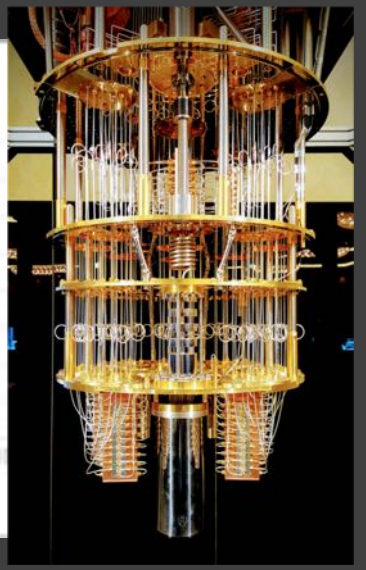
Q# Copy

```
function AmpAmpByOracle (nIterations : Int, stateOracle : StateOracle, idxFlagQubit : Int) : (Qubit[] => () : Adjoint,
Controlled)
```

Reversing on the real device

Real vs Simulated for x + y = 1



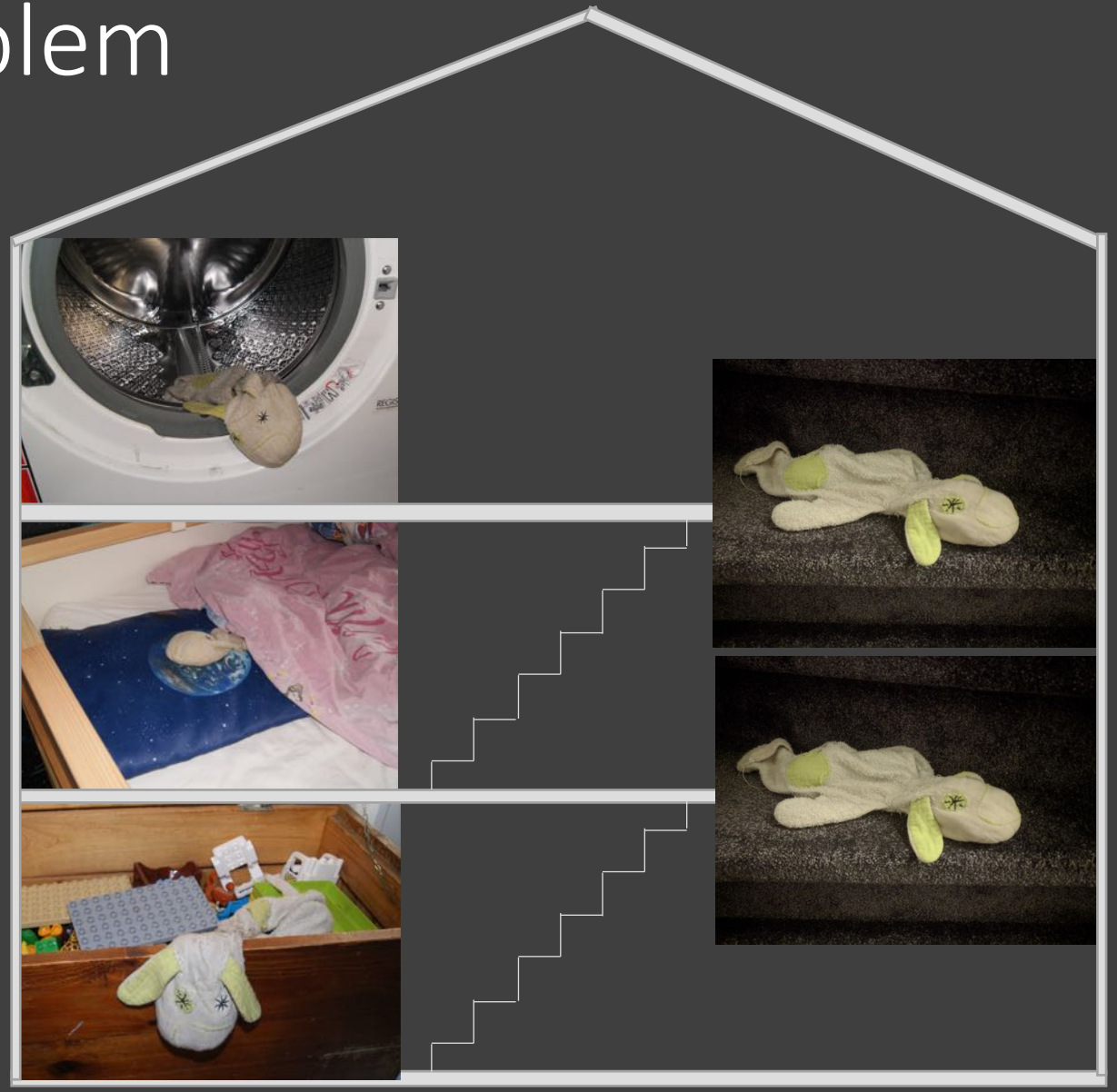


Stuffed animal problem

Attic

Bedroom

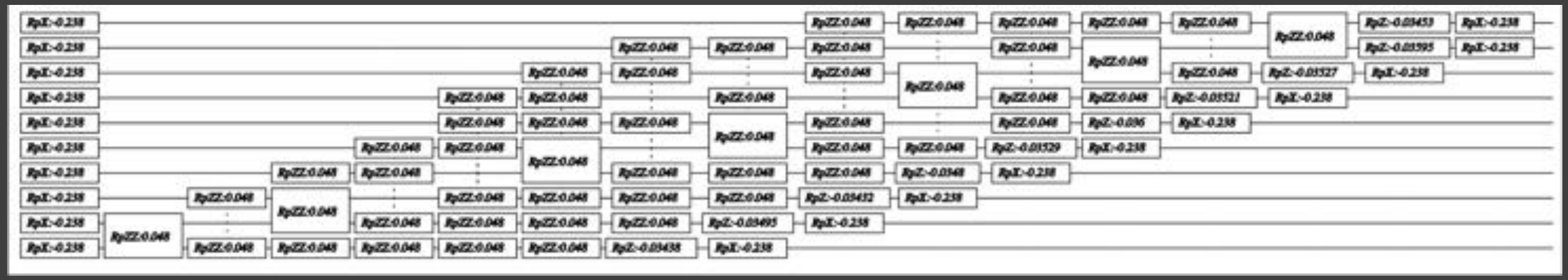
Living Room



Picture by Rolf Huisman / CC0 Creative Commons

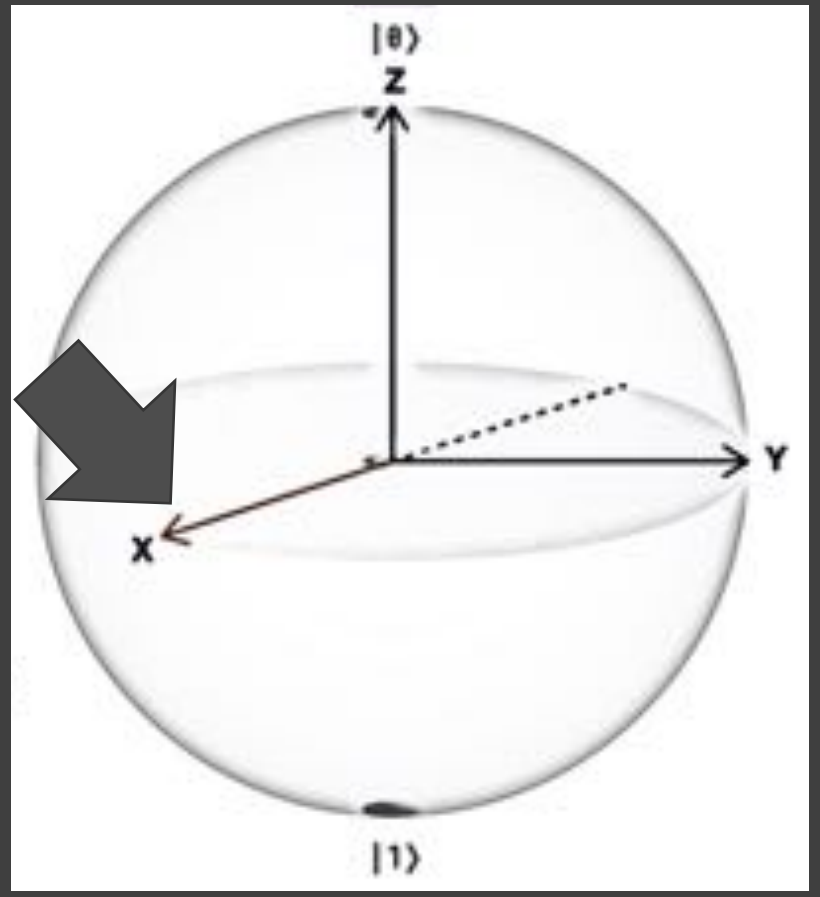
Using the gates to describe

Optimized Traveling Salesman Problem 5 Cities:



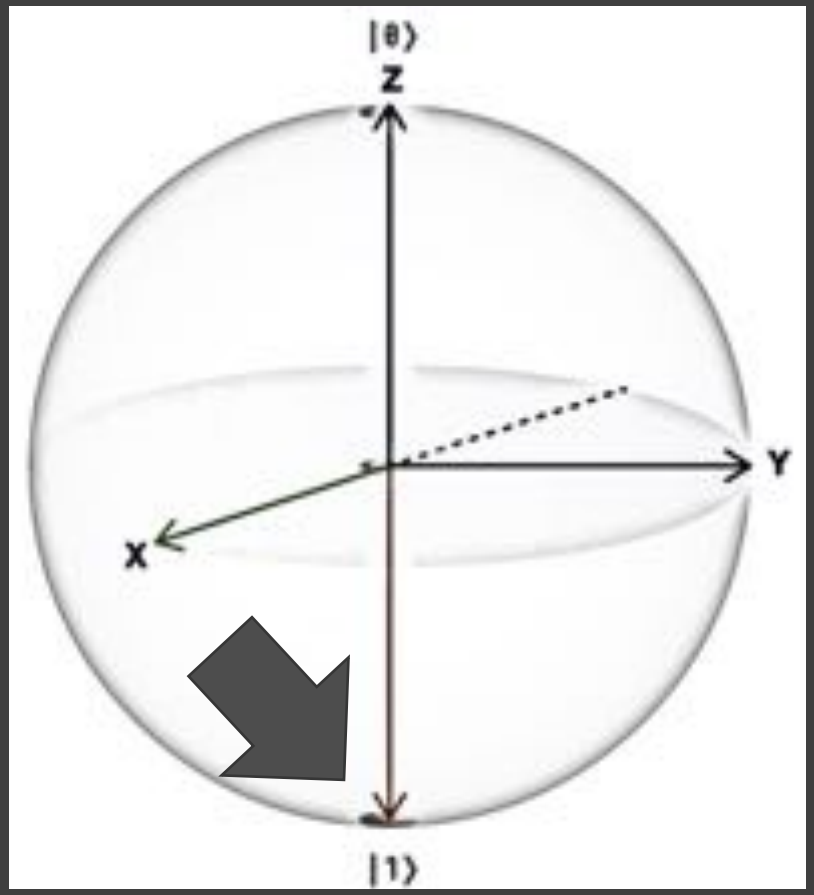
*Details: Phys. Rev. A **95**, 032323 – Published 22 March 2017*

Oracle Function



One Stair

Oracle (State)



Two Stairs

Many hours and code lines later

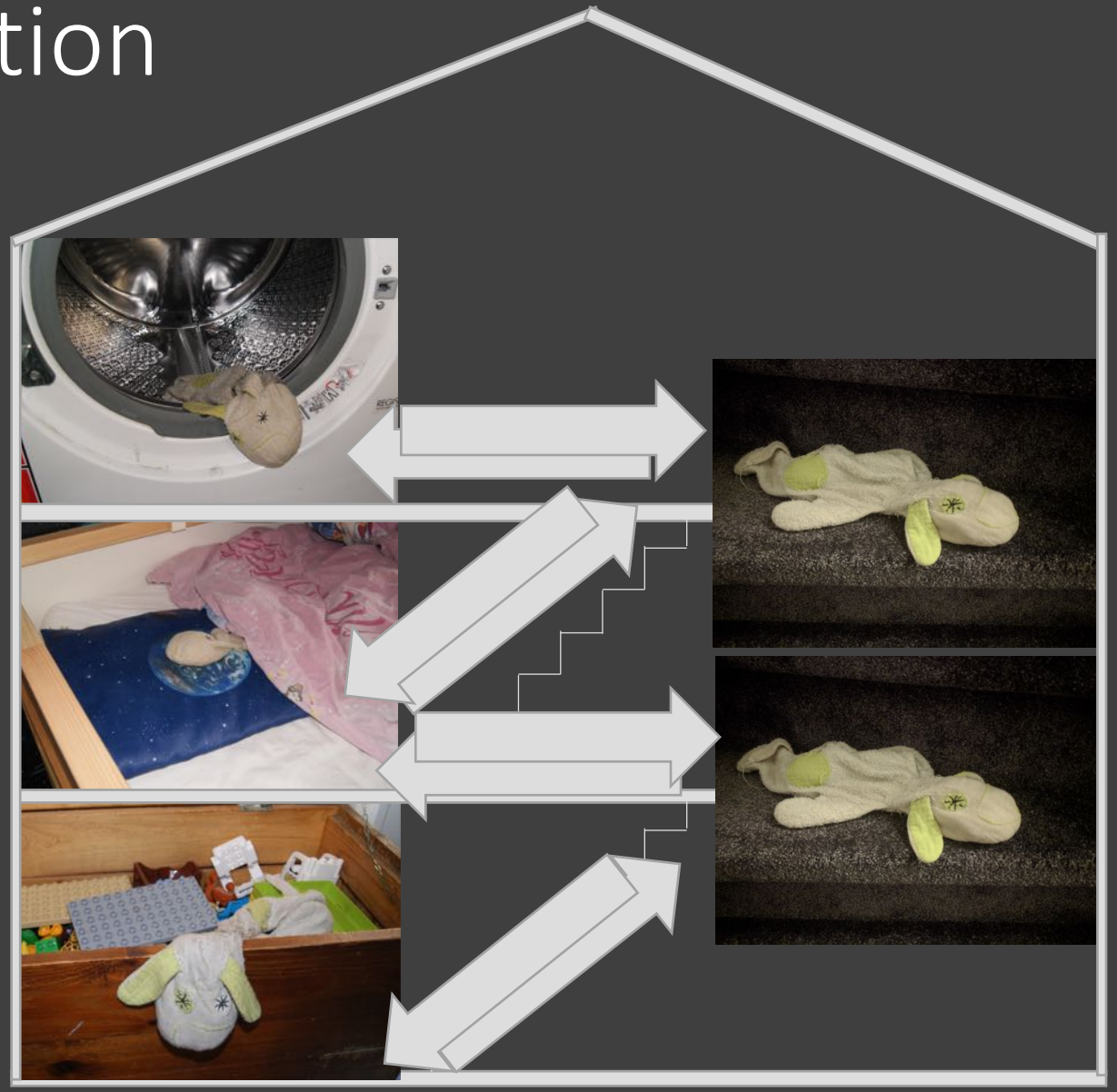
	Traditional	**Quantum**
Naive	$O(n!)$	$O(\sqrt{n!})$
With effort	$O(2n*2^n)$	$O((\sqrt{n})!)$

Stuffed animal solution

Attic

Bedroom

Living Room



Picture by Rolf Huisman / CC0 Creative Commons

Low-Tech Solution

Conclusions

Quantum Computing

1. Its not simple
2. But its powerful
3. It will change the world
4. Is currently unstable
5. Is cool to program for
6. One can use existing tooling
7. And it works on real hardware ☺

DOTNEXT

So be part of the quantum future !

Questions ?

Rolf Huisman

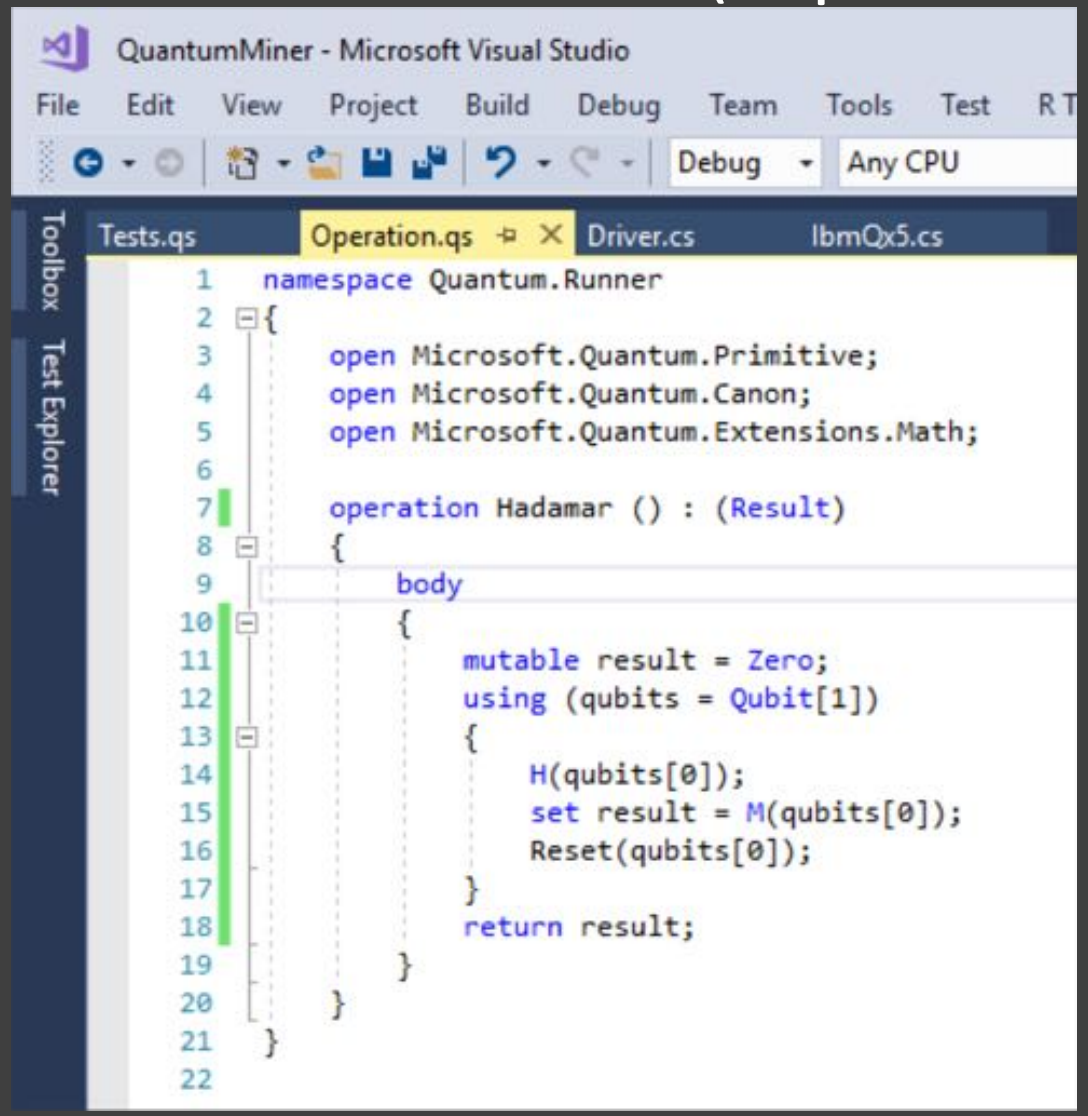
Rolf.Huisman@infosupport.com

@rlrhuisman

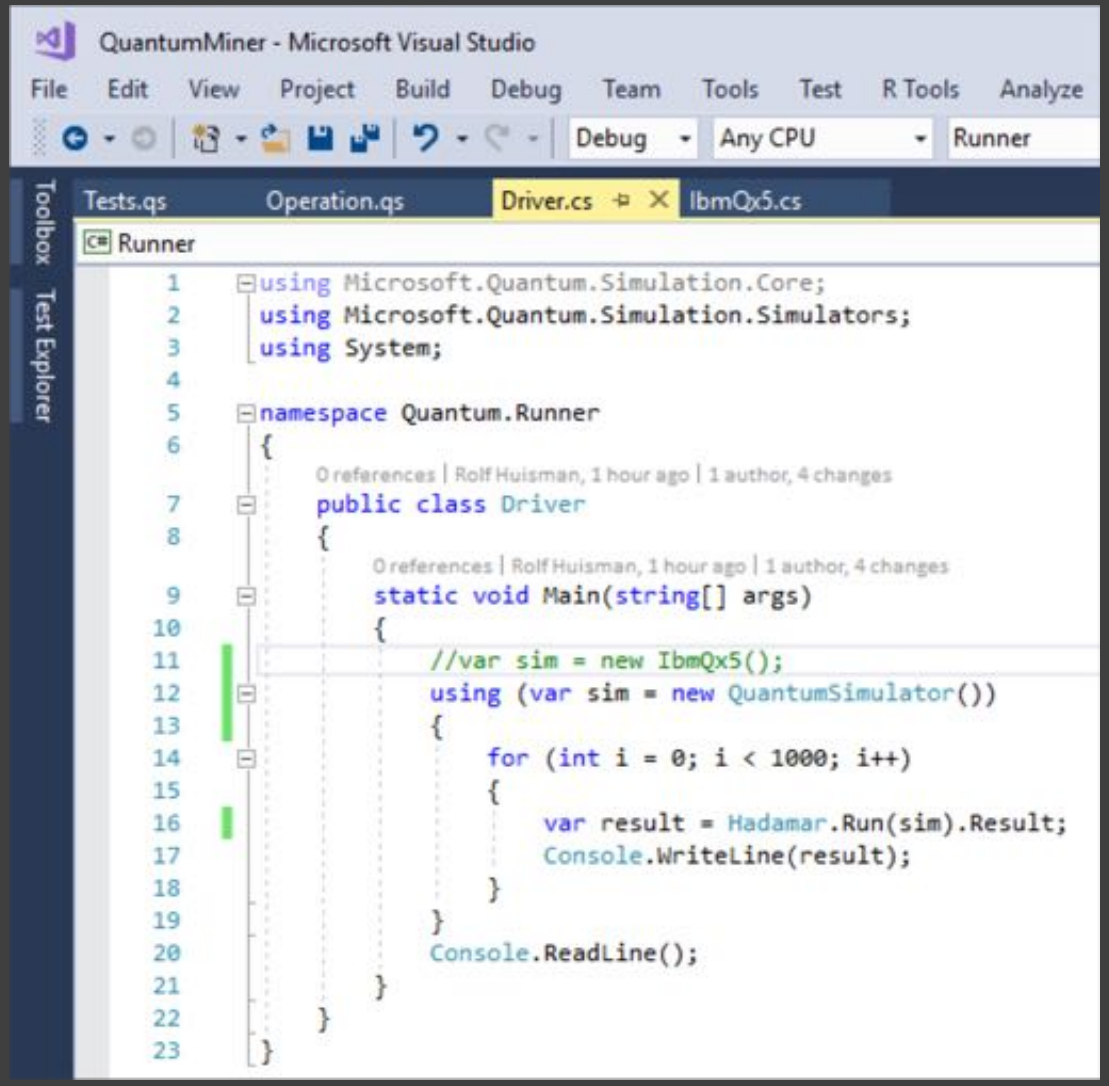
Microsoft® MVP Most Valuable Professional

infoSupport Solid Innovator

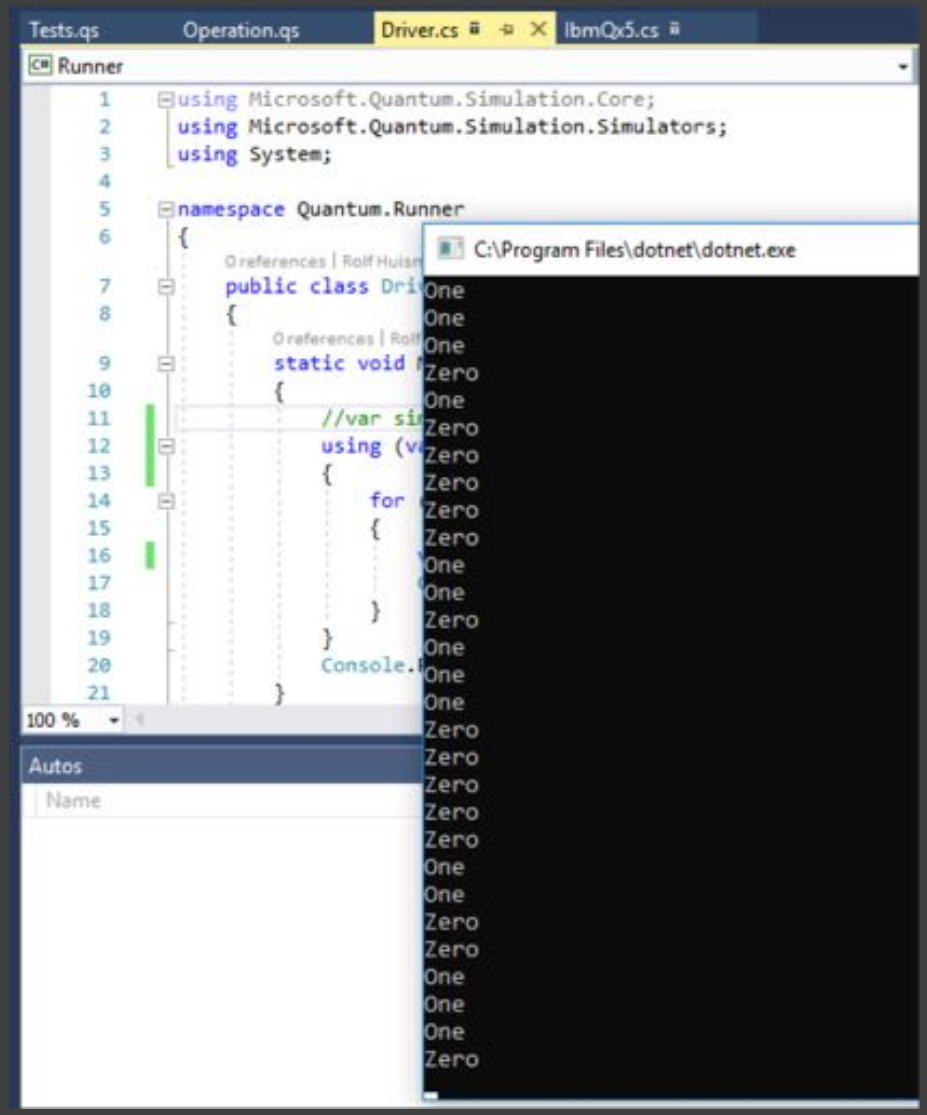
Microsoft Q# (Operators)



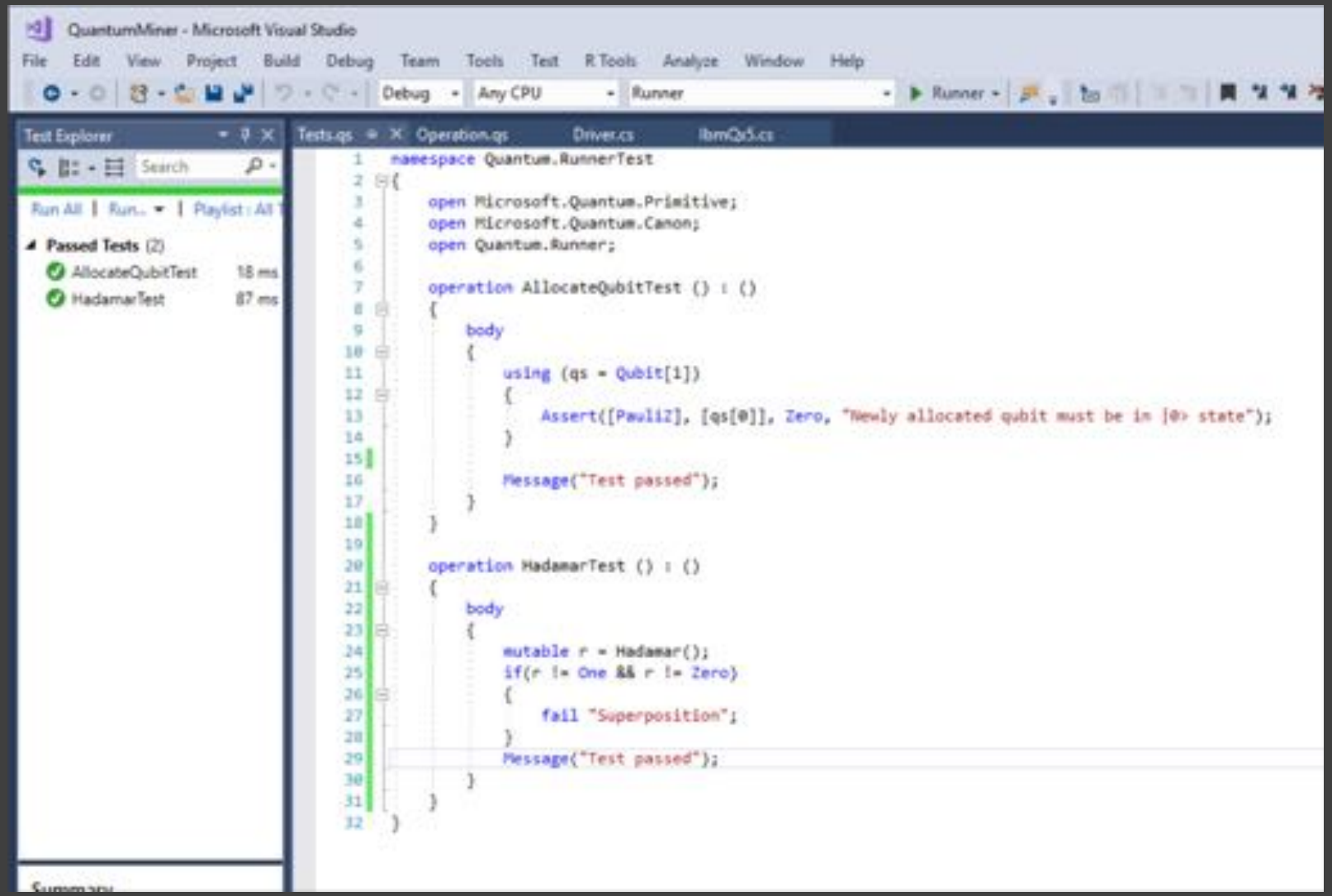
Microsoft Q# (Driver)



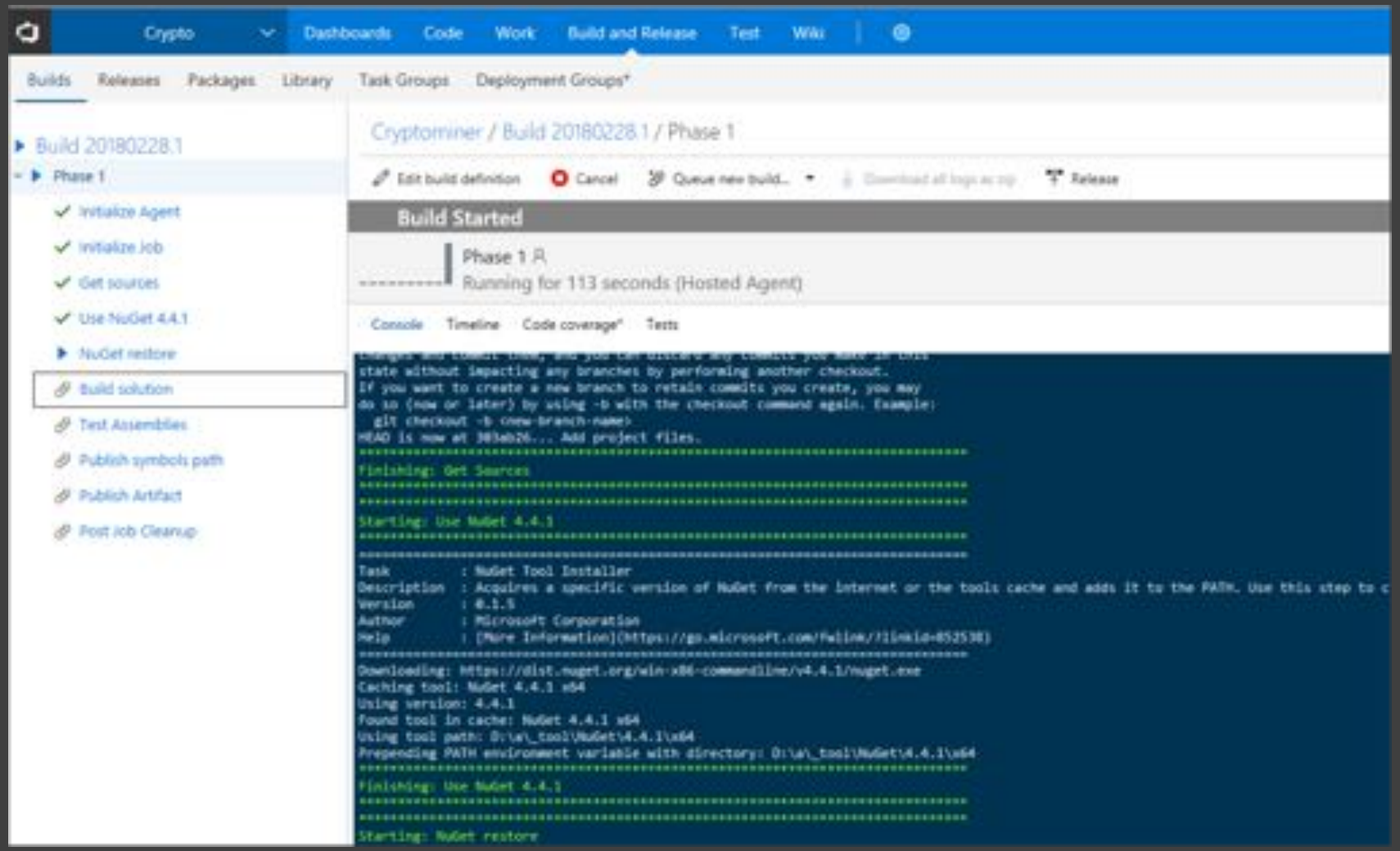
Microsoft Q# (Emulated Run)



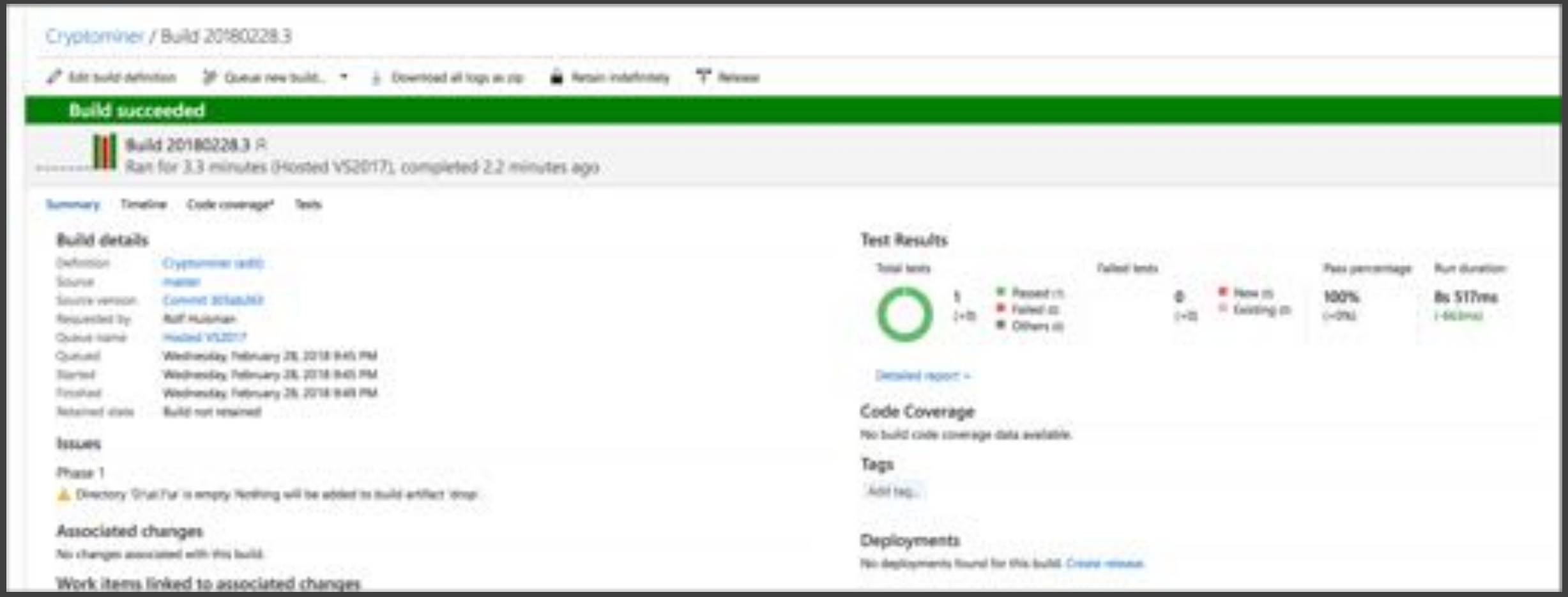
Microsoft Q# (Unit Test)



Microsoft Q# (VSTS Build)



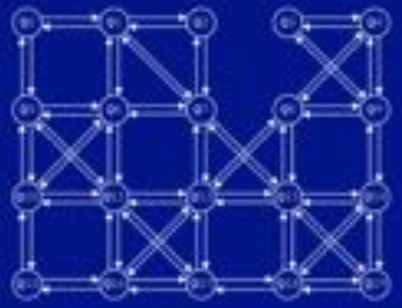
Microsoft Q# (VSTS Test)



Production (IBM Q)

Backend: QS1_1 (20 Qubits)

ACTIVE | AVAILABLE TO HUBS, PARTNERS, AND MEMBERS OF THE IBM Q NETWORK



Last Calibration: 2018-03-01 20:24:56

	Q0	Q1	Q2	Q3	Q4	Q5	Q6
Frequency (GHz)	4.84	4.48	4.86	5.03	5.01	4.91	4.89
T1 (µs)	71.96	61.08	73.91	68.01	56.06	78.55	87.03
T2 (µs)	33.97	28.54	25.02	36.77	28.43	51.32	38.12
Gate error (10^{-3})	2.01	2.56	6.64	7.27	1.96	1.95	1.96
Readout error (10^{-2})	7.70	27.80	19.60	20.65	9.55	8.90	14.90
MultiQubit gate error (10^{-2})	CX0_1 2.67	CX1_0 2.67	CX2_1 3.34	CX3_4 7.06	CX4_3 7.06	CX5_0 1.73	CX6_1 3.90
	CX0_5 1.73	CX1_2 7.64	CX2_6 7.86	CX3_9 7.33	CX4_8 4.95	CX5_6 2.72	CX6_2 7.86
		CX1_6 3.90	CX2_7 4.20		CX4_9 3.32	CX5_10 2.34	CX6_5 2.72
		CX1_7 5.74				CX5_11 2.76	CX6_7 3.67
							CX6_10 3.11
							CX6_11 3.67

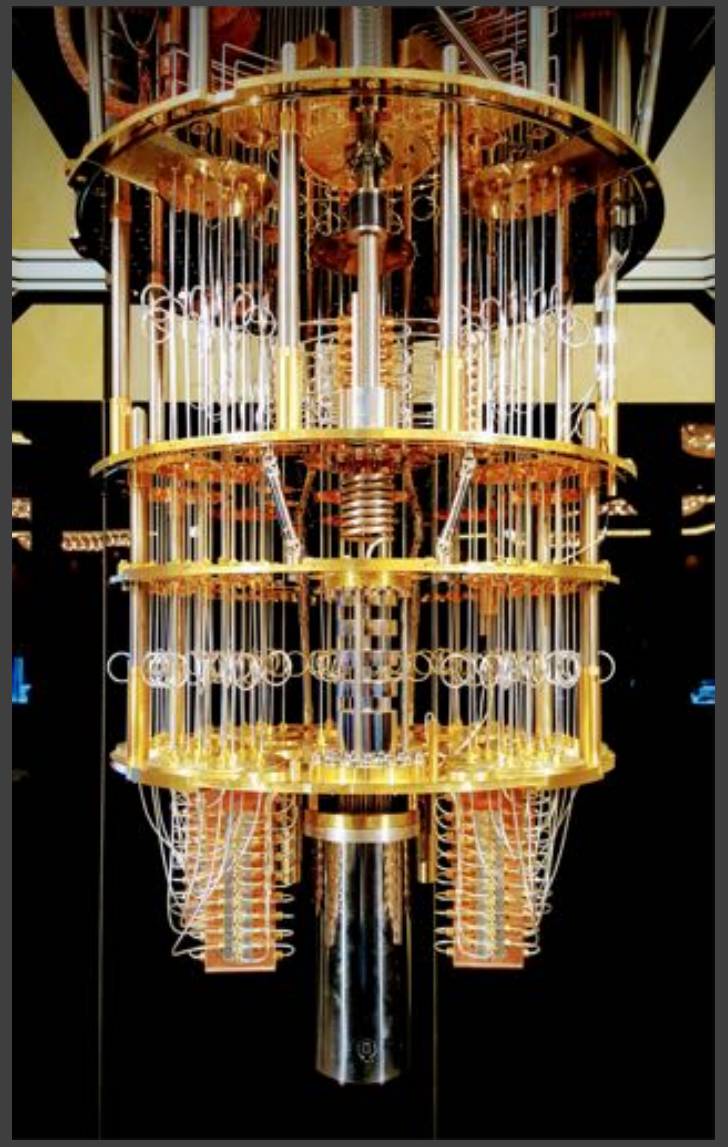
Backend: ibmqx5 (16 Qubits)

ACTIVE | AVAILABLE ON QISKIT

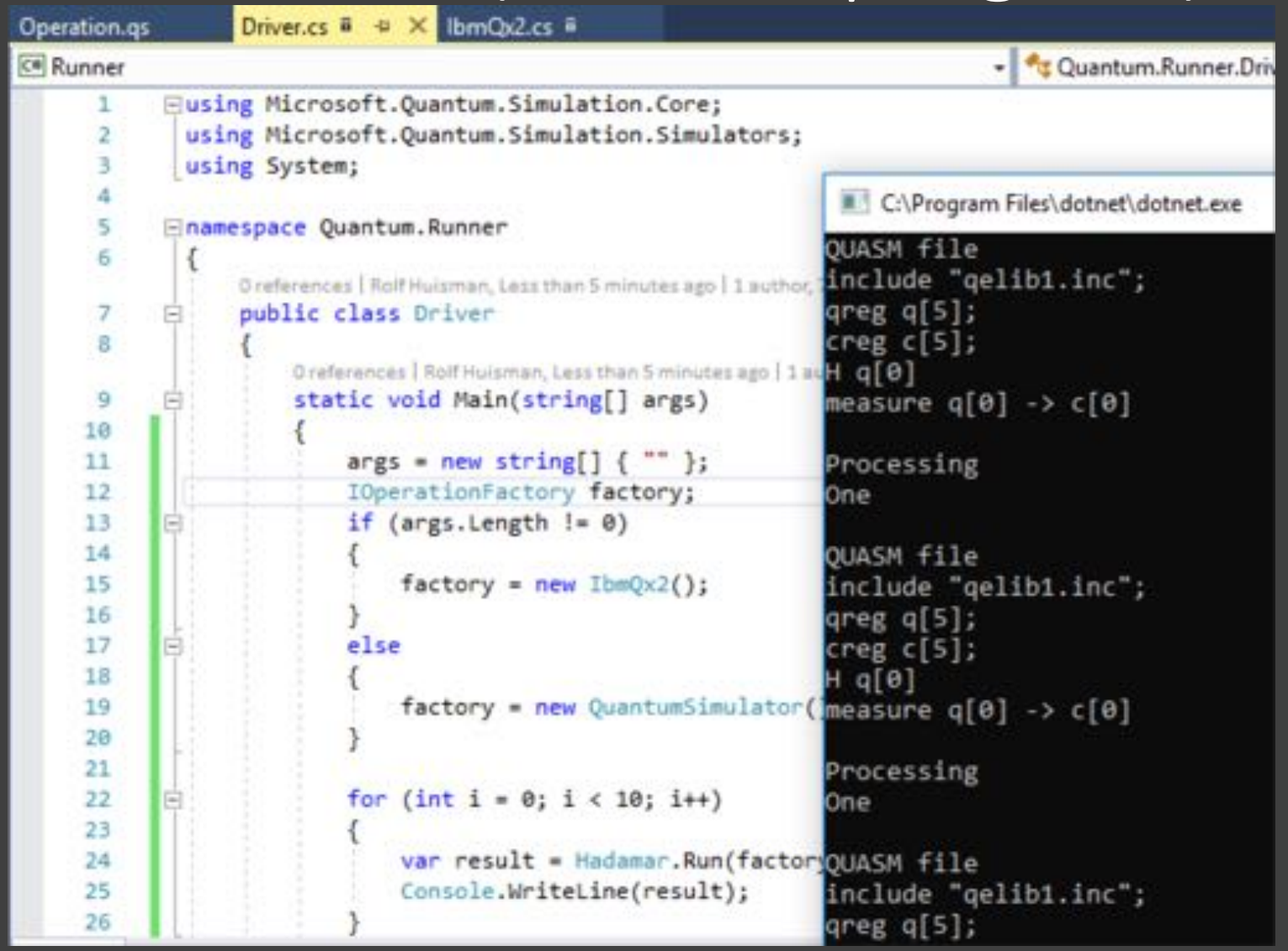
Last Calibration: 2018-03-01 13:35:31
Fridge Temperature: 0.0133661 K

More details

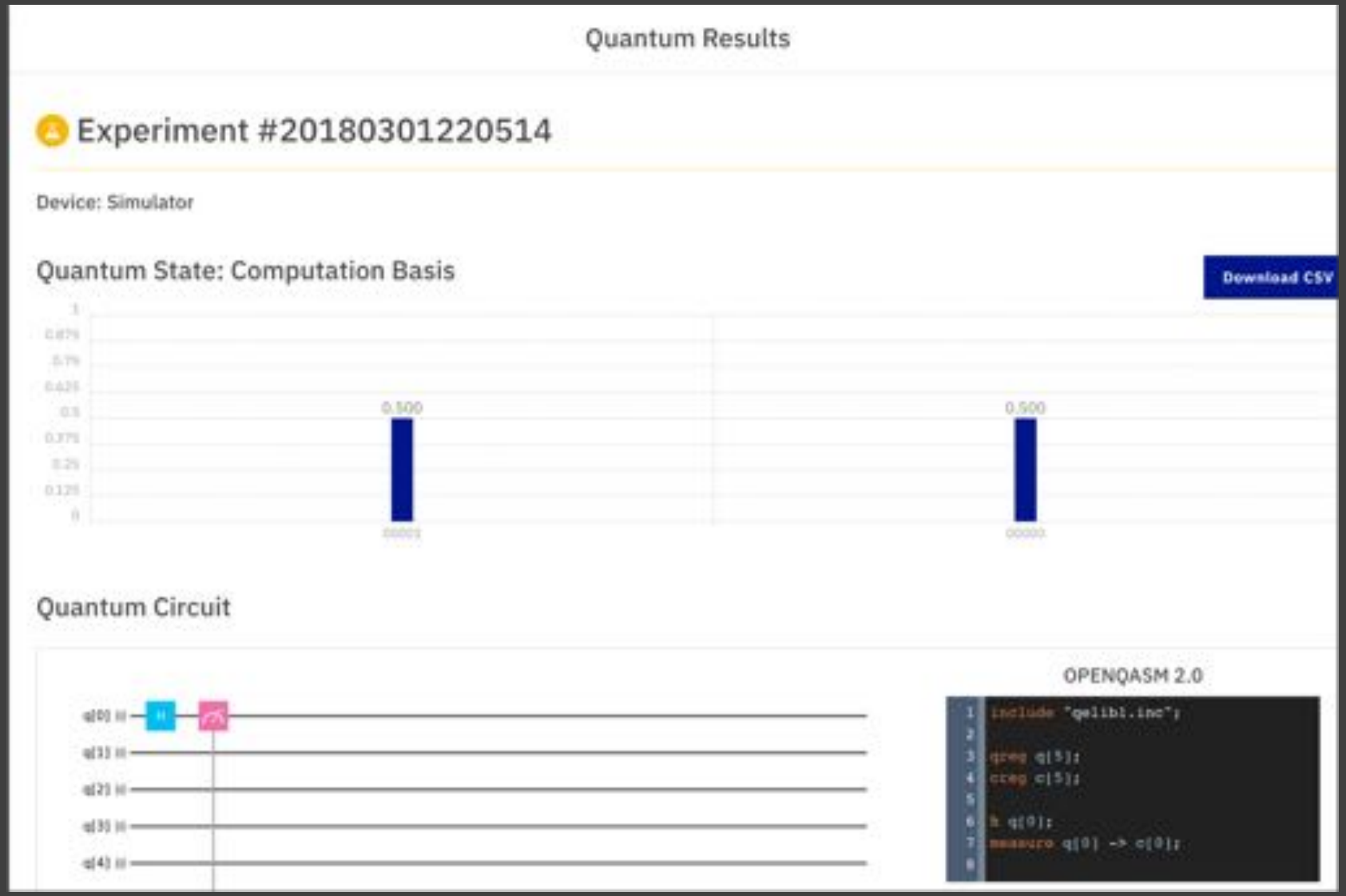
	Q0	Q1	Q2	Q3	Q4	Q5	Q6
Frequency (GHz)	5.26	5.40	5.28	5.08	4.98	5.15	5.31
T1 (µs)	39.30	36.40	42.30	51.80	33.30	43.50	50.50
T2 (µs)	34.60	58.60	57.70	73.50	54.20	47.50	80.40
Gate error (10^{-3})	2.17	3.36	3.66	1.60	1.35	3.63	1.81
Readout error (10^{-2})	6.32	6.17	3.91	6.82	6.60	5.55	4.42
MultiQubit gate error (10^{-2})		CX1_0 5.00	CX2_3 3.47	CX3_4 3.26		CX5_4 5.57	CX6_5 5.23
		CX1_2 4.19		CX3_14 4.37			CX6_7 2.90
							CX6_11 2.71



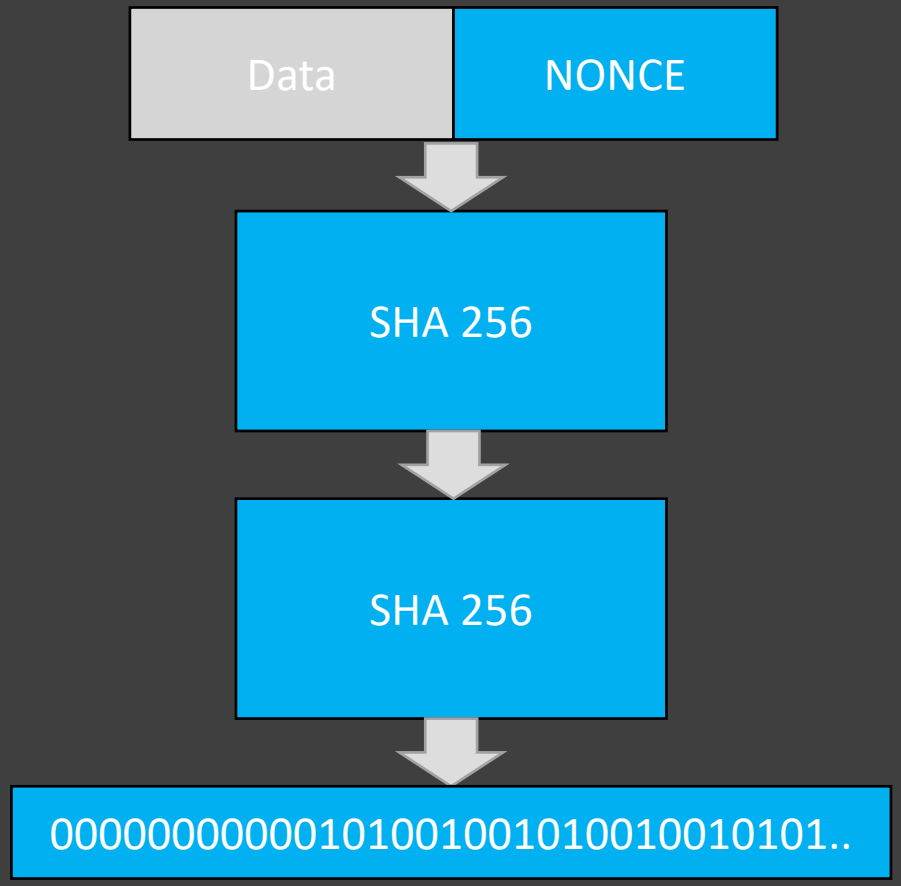
Production (Work in progress)



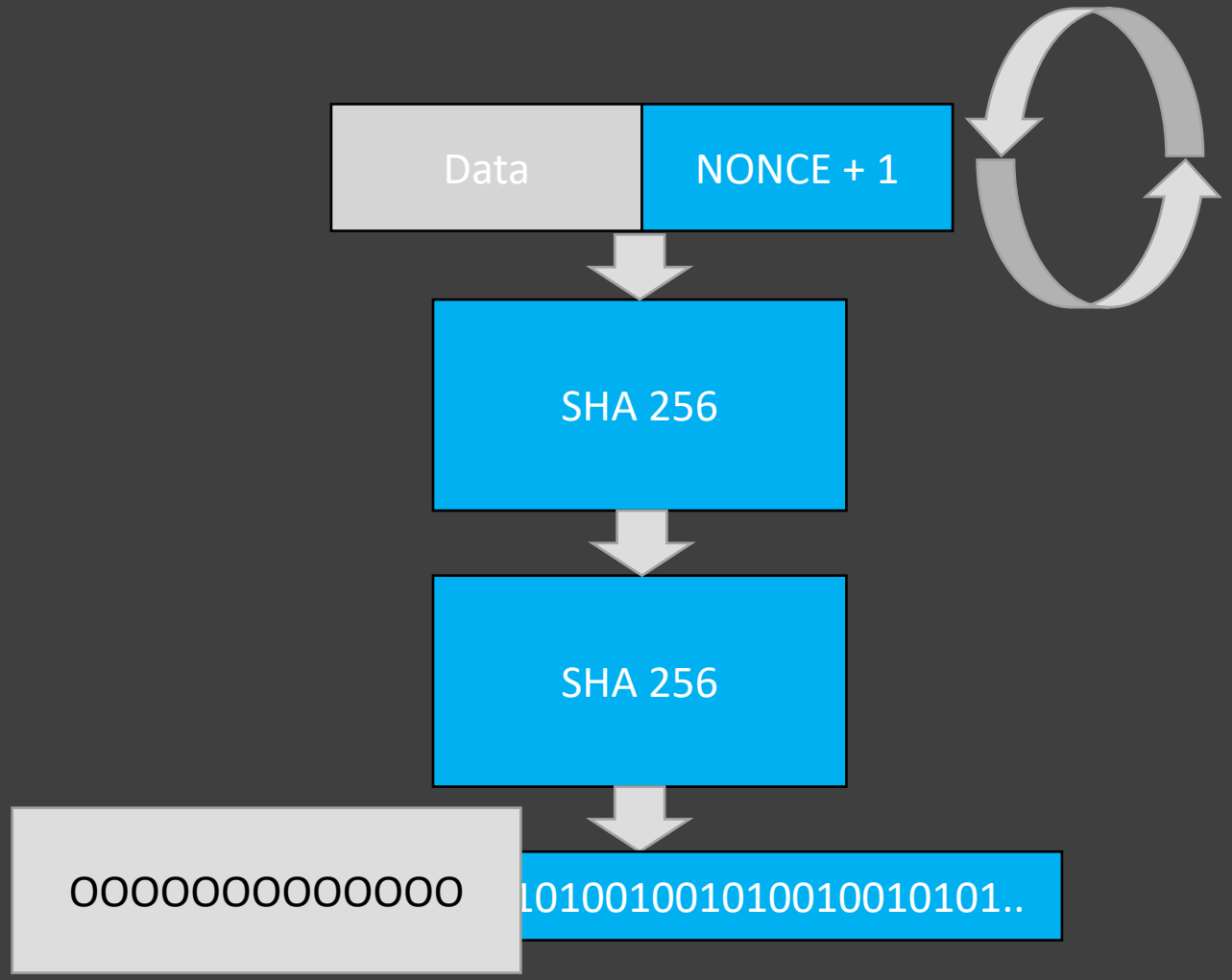
Production (Work in progress)



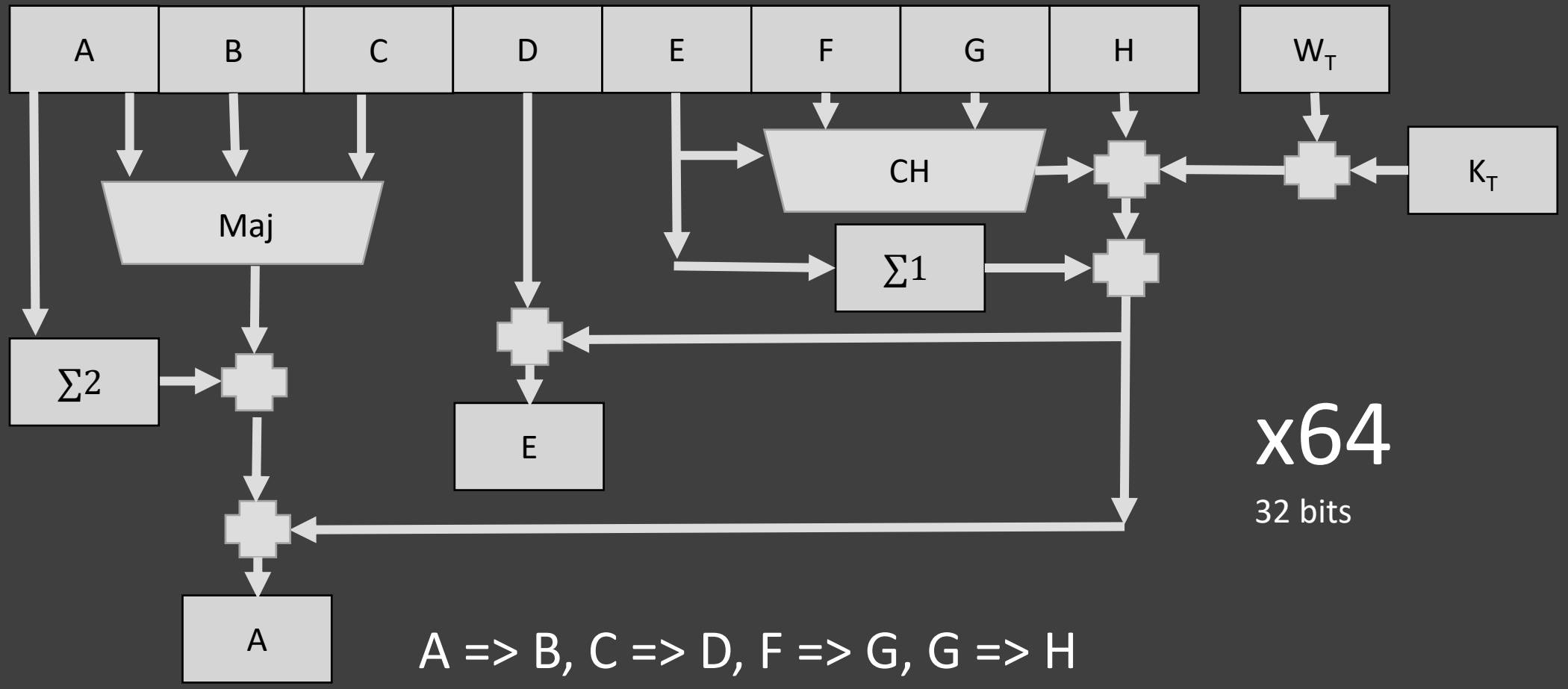
~~Bitcoin~~ Mining 101



~~Bitcoin~~ Mining 101



SHA 256



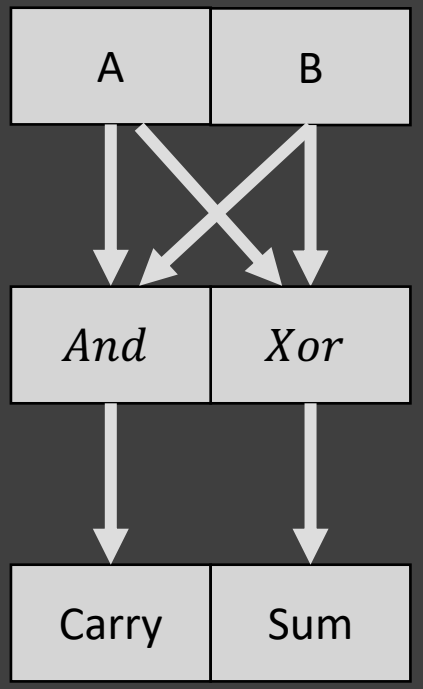
x64

32 bits

A => B, C => D, F => G, G => H

Reduced Adder

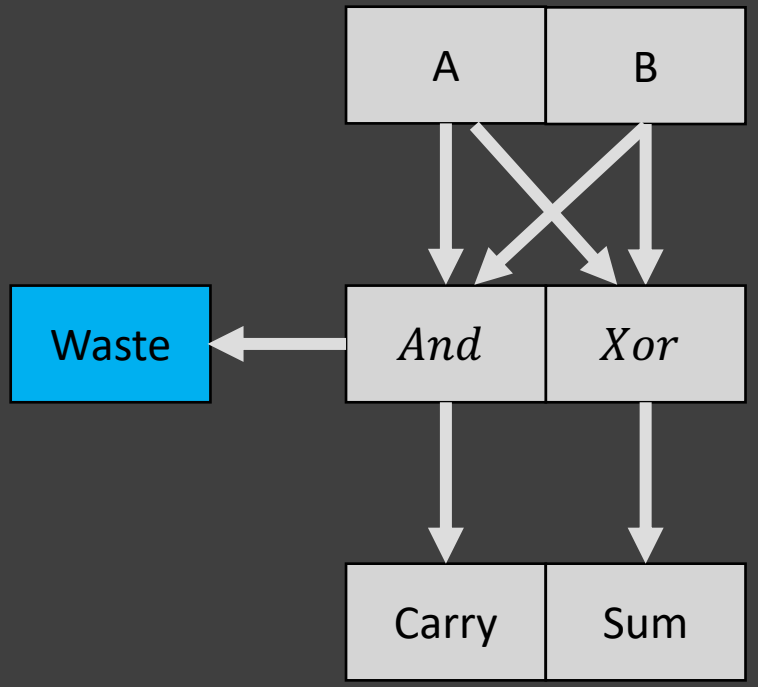
Oversimplified



x1

1 bit

Quantum Adder

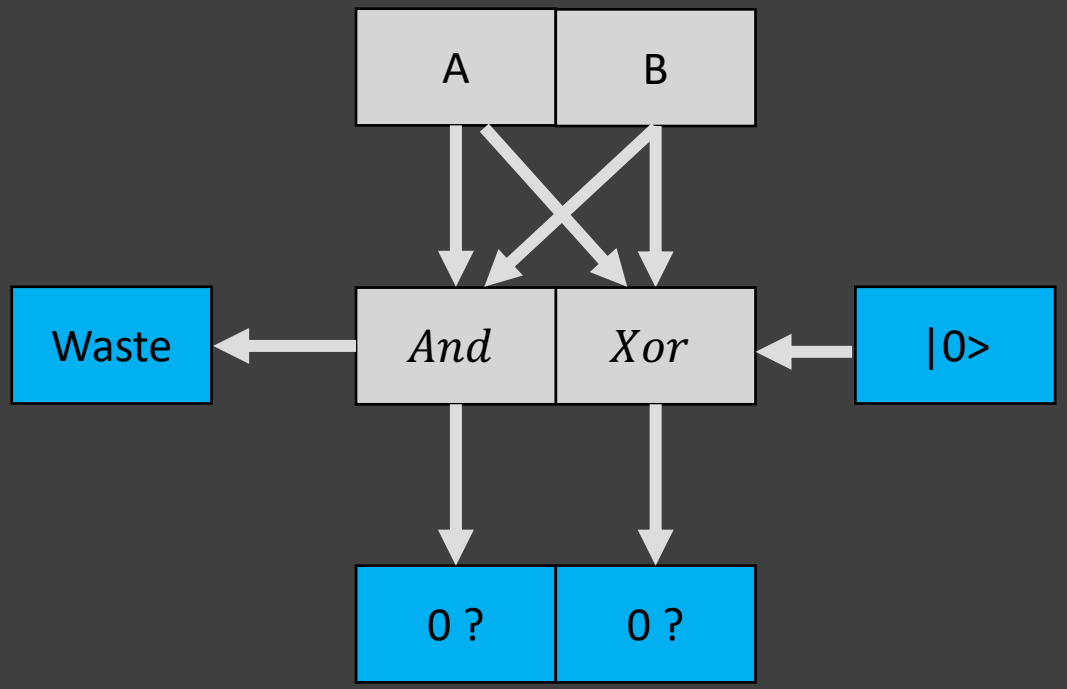


Quantum Adder

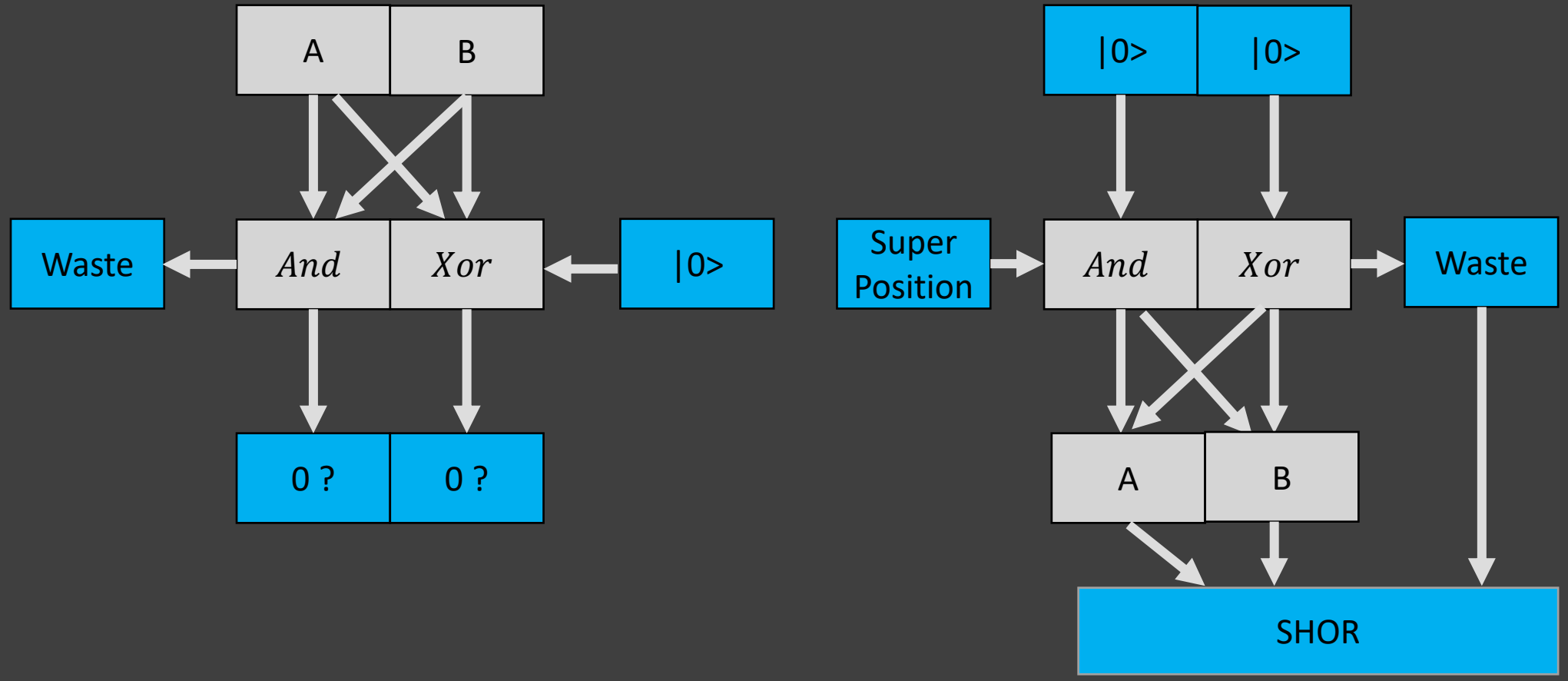
```
22      operation Adder (first: Bool, second: Bool) : (Result, Result)
23      {
24          body
25          {
26              mutable a = Zero;
27              mutable b = Zero;
28              using (qubits = Qubit[3])
29              {
30                  let inA = qubits[0];
31                  let inB = qubits[1];
32                  let outA = qubits[2];
33
34                  //Data
35                  if(first){  X(inA); } //Invese A to expect the first bit to be one
36                  if(second){ X(inB); } //Inverse B to expect the last bit to one
37
38                  //perform addition in forward
39                  CCNOT(inA, inB, outA); //And a and b in outB
40                  let outB = inB;
41                  CNOT(inA, outB); //XOR with B in out A
42
43                  //Read the outputs
44                  set a = M(outA);
45                  set b = M(outB);
46
47                  //Cleanup
48                  Reset(qubits[0]);
49                  Reset(qubits[1]);
50                  Reset(qubits[2]);
51              }
52              return (a,b);
53          }
54      }
55  }
56
```

```
            One
            r

            1 + 1 = (One, Zero)
            0 + 1 = (Zero, One)
            1 + 0 = (Zero, One)
        }   0 + 0 = (Zero, Zero)
        Consc
    }
```

Adder Algorithm



Inverted Adder Algorithm



Reverse Adder

```
operation ReverseAdder (first: Bool, second: Bool) : (Result, Result, Result)
{
    body
    {
        mutable a = Zero;
        mutable b = Zero;
        mutable error = Zero;
        using (qubits = Qubit[3])
        {
            let outA = qubits[0]; //end garbage
            let outB = qubits[1]; //also inB
            let inA = qubits[2];

            //Data
            if(first){  X(outA); } //Invese A to expect the first bit to be one
            if(second){ X(outB); } //Inverse B to expect the last bit to one

            // Put waste into superposition
            H(inA);

            //perform addition in reverse
            CNOT(inA, outB); //XOR with B in out A
            let inB = outB;
            CCNOT(inA, inB, outA); //And a and b in outB

            //Read the outputs
            set a = M(inA);
            set b = M(inB);
            set error = M(outA);

            //Cleanup
            Reset(qubits[0]);
            Reset(qubits[1]);
            Reset(qubits[2]);
        }
        return (a,b, error);
    }
}
```

```
? + ? = 2(One, One, Zero)
? + ? = 2(Zero, Zero, One)
? + ? = 2(One, One, Zero)
? + ? = 2(Zero, Zero, One)
? + ? = 2(Zero, Zero, One)
? + ? = 1(Zero, One, Zero)
? + ? = 1(One, Zero, Zero)
? + ? = 1(One, Zero, Zero)
? + ? = 1(Zero, One, Zero)
? + ? = 1(Zero, One, Zero)
? + ? = 0(Zero, Zero, Zero)
? + ? = 0(Zero, Zero, Zero)
? + ? = 0(One, One, One)
? + ? = 0(One, One, One)
? + ? = 0(Zero, Zero, Zero)
```

Reverse Adder

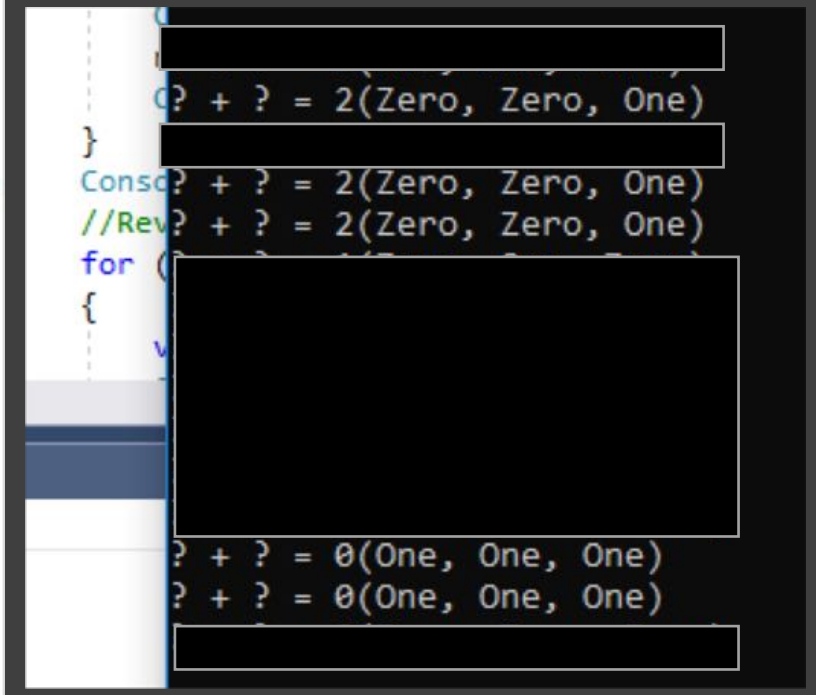
```
operation ReverseAdder (first: Bool, second: Bool) : (Result, Result, Result)
{
    body
    {
        mutable a = Zero;
        mutable b = Zero;
        mutable error = Zero;
        using (qubits = Qubit[3])
        {
            let outA = qubits[0]; //end garbage
            let outB = qubits[1]; //also inB
            let inA = qubits[2];

            //Data
            if(first){  X(outA); } //Invese A to expect the first bit to be one
            if(second){ X(outB); } //Inverse B to expect the last bit to one

            // Put waste into superposition
            H(inA);

            //perform addition in reverse
            CNOT(inA, outB); //XOR with B in out A
            let inB = outB;
            CCNOT(inA, inB, outA); //And a and b in outB

            //Read the outputs
            set a = M(inA);
            set b = M(inB);
            set error = M(outA);

            //Cleanup
            Reset(qubits[0]);
            Reset(qubits[1]);
            Reset(qubits[2]);
        }
        return (a,b, error);
    }
}
```

```
? + ? = 2(One, One, Zero)
[redacted]
? + ? = 2(One, One, Zero)
[redacted]
? + ? = 1(Zero, One, Zero)
? + ? = 1(One, Zero, Zero)
? + ? = 1(One, Zero, Zero)
? + ? = 1(Zero, One, Zero)
? + ? = 1(Zero, One, Zero)
? + ? = 0(Zero, Zero, Zero)
? + ? = 0(Zero, Zero, Zero)
[redacted]
? + ? = 0(Zero, Zero, Zero)
```

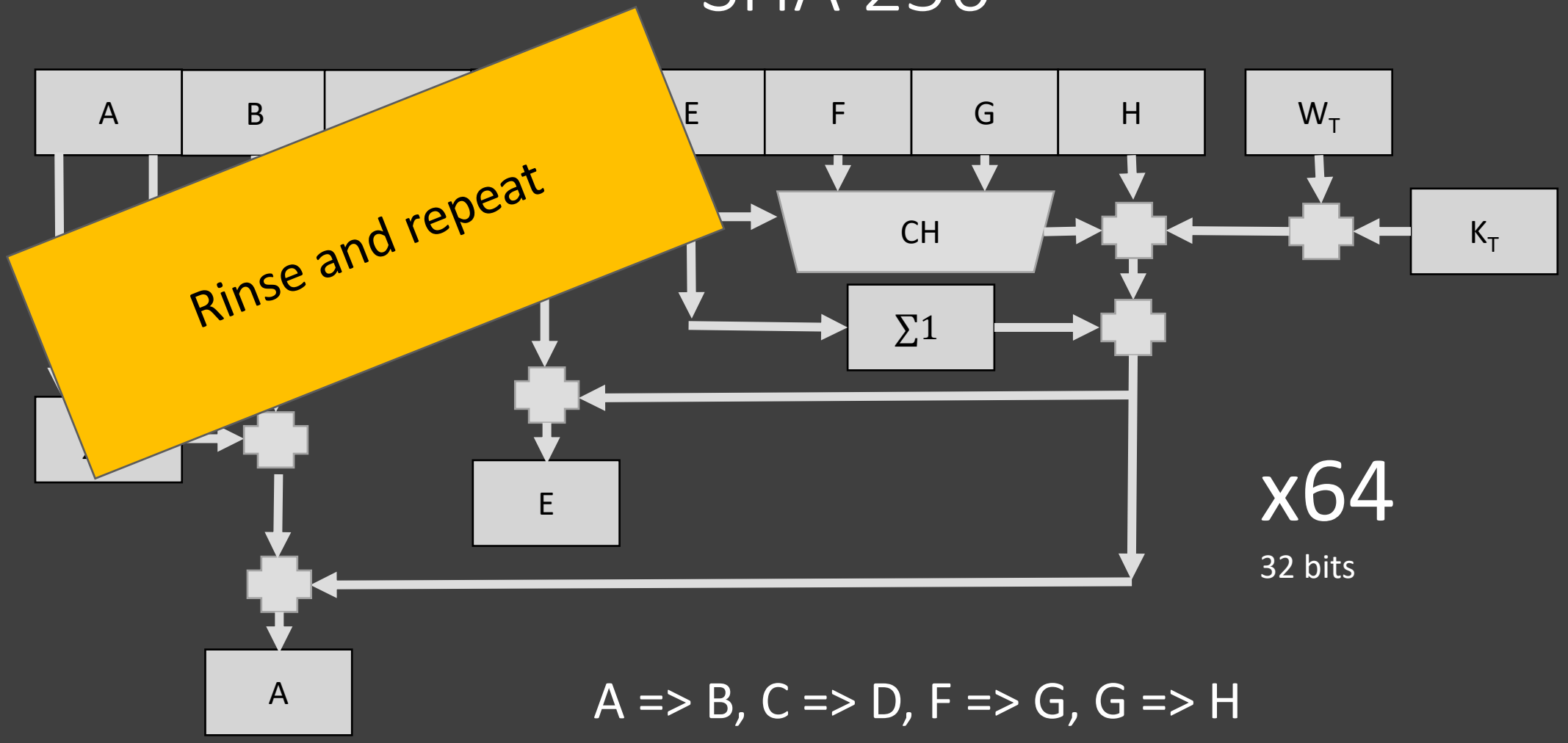
Reverse Adder

```
operation ReverseAdder (first: Bool, second: Bool) : (Result, Result, Result)
{
    body
    {
        mutable a = Zero;
        mutable b = Zero;
        mutable error = Zero;
        using (qubits = Qubit[3])
        {
            let outA = qubits[0]; //end garbage
            let outB = qubits[1]; //also inB
            let inA = qubits[2];

            //Data
            if(first){  X(outA); } //Invese A to expect the first bit to be one
            if(second){ X(outB); } //Inverse B to expect the last bit to one

            // Put waste into superposition
            H(inA);

            Reset(qubits[0]);
            Reset(qubits[1]);
            Reset(qubits[2]);
        }
        return (a,b, error);
    }
}
```

This is why you need to add Grover



SHA 256

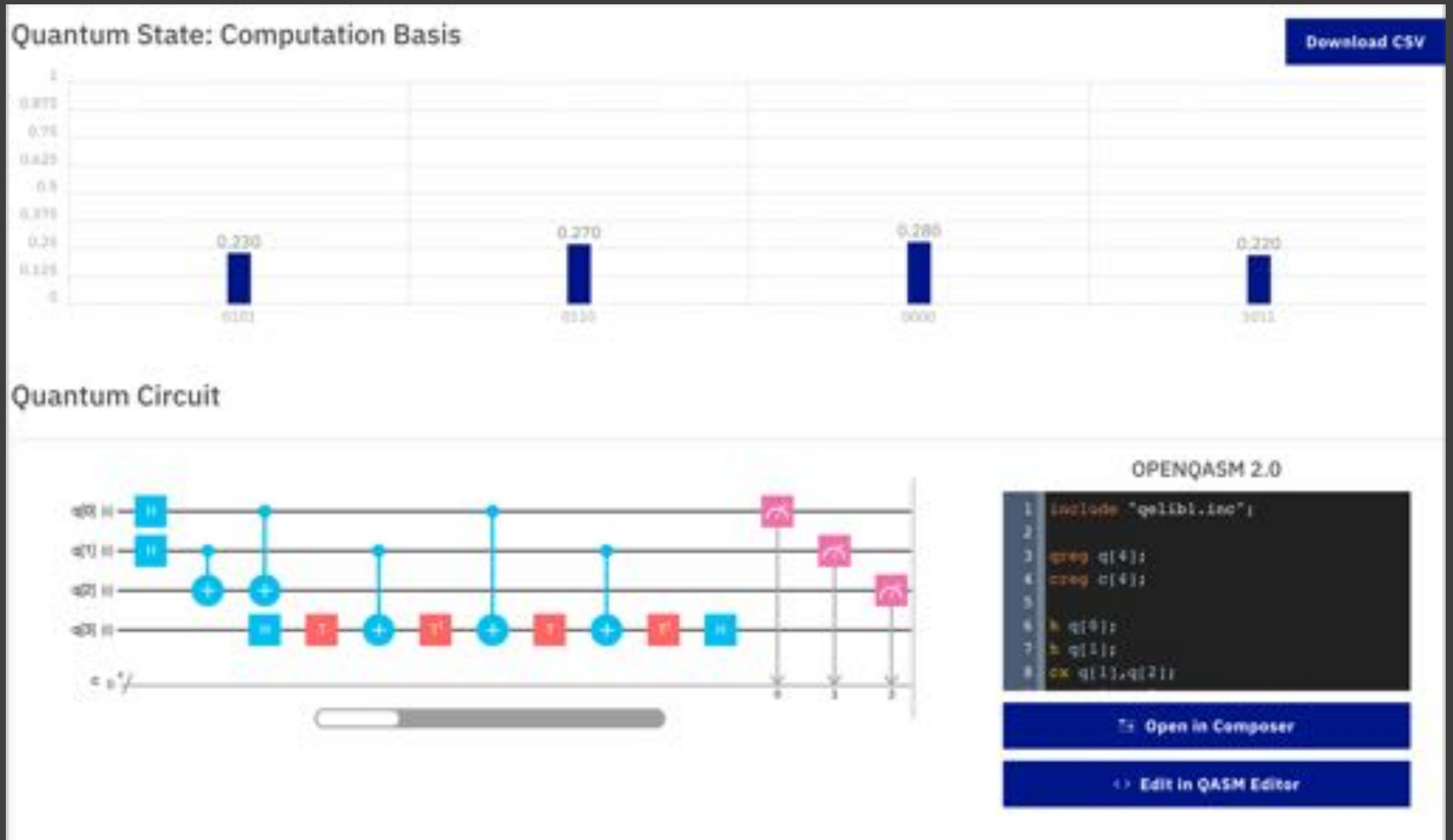


x64

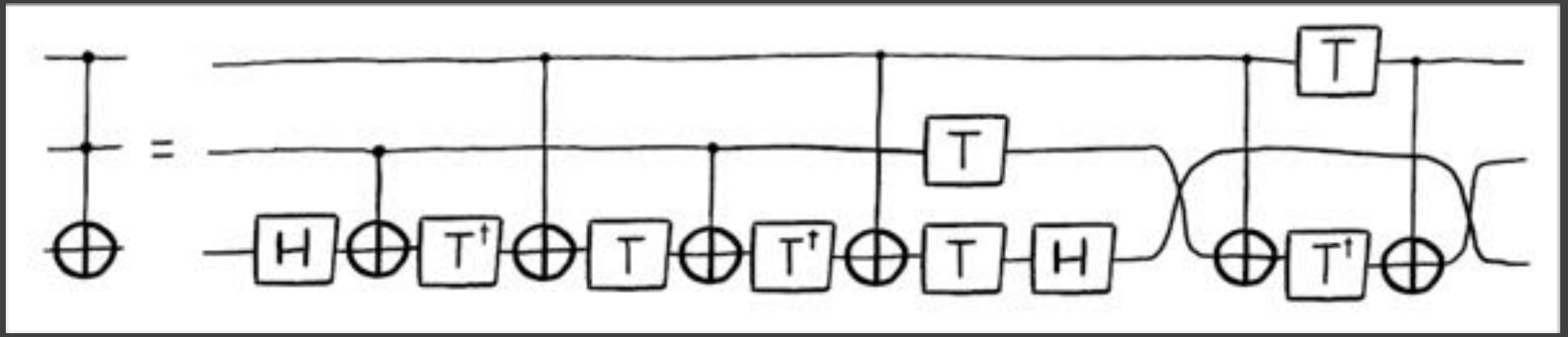
32 bits

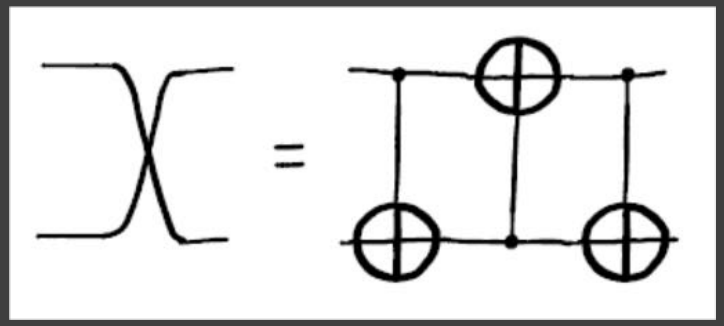
A => B, C => D, F => G, G => H

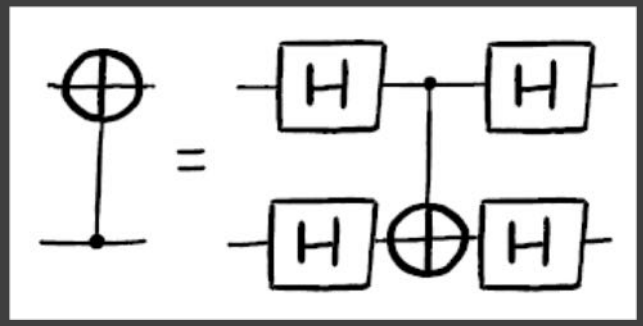
Dirty Adder (Not Reduced)



Tolifo Gate







Adder (Not Reduced)