Keeping Current Seminar

What Every Computer Scientist Should Know About Quantum Computing

March 5, 2025

Prof. Henry Dietz
Electrical & Computer Engineering

Aggregate.Org UNBRIDLED COMPUTING

University of Kentucky

Abstract

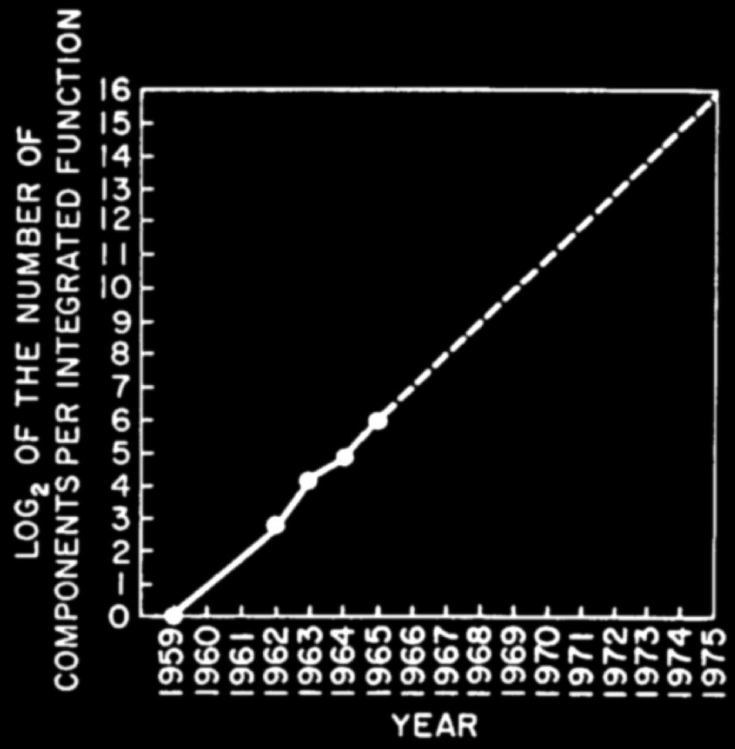
Last November, El Capitan officially became the world's fastest supercomputer. It claimed that record using 11,039,616 cores and about 30MW. About a month later, Google Quantum AI announced that their Willow quantum computer chip "performed a standard benchmark computation in under five minutes that would take one of today’s fastest supercomputers 10 septillion (that is, 10^25) years." One could scale-up a supercomputer like El Capitan to match the performance claimed by Willow, but that machine would require the energy output of billions of Suns! The catch is that most computations that you can do in under five minutes on a $3 microcontroller are not even theoretically possible on a Willow chip...

This talk will briefly explain what every computer scientist should know about quantum computing. We will do that without getting into details of quantum physics that Einstein called "spooky" and without discussing how Schrodinger contemplated abusing his cat. Instead, we will focus on what kinds of computational tasks quantum computers really can accomplish more efficiently than conventional machines, what they cannot do and why not, and what the main problems are in building practical quantum computers.

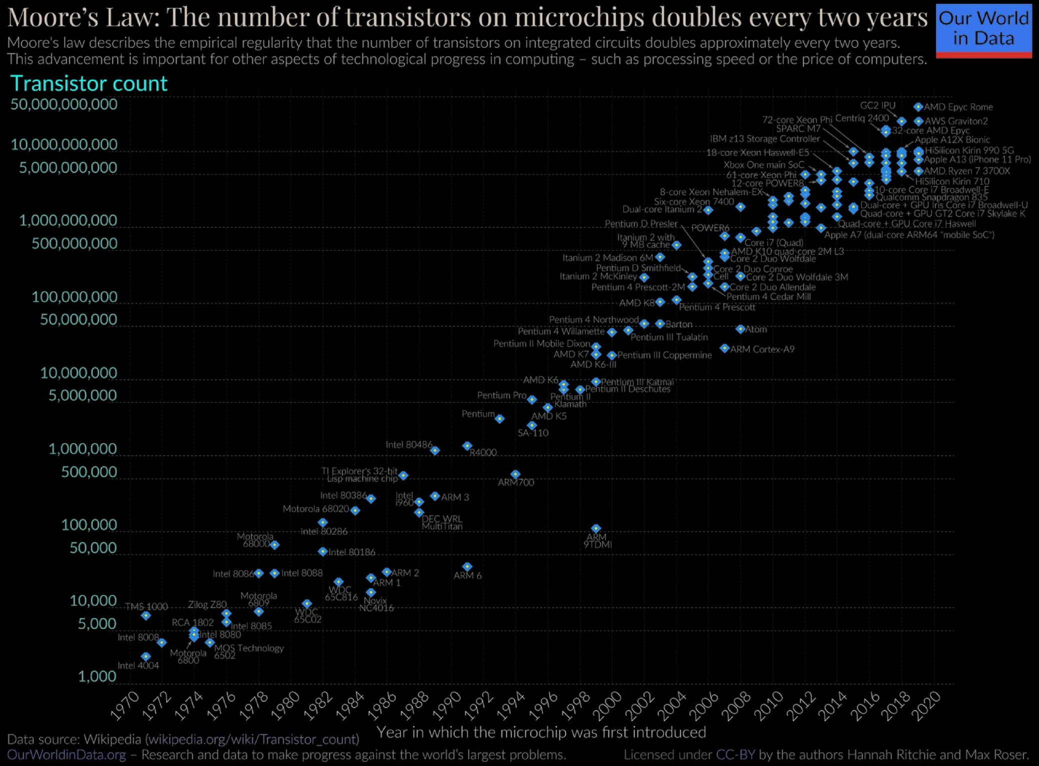
University of Kentucky

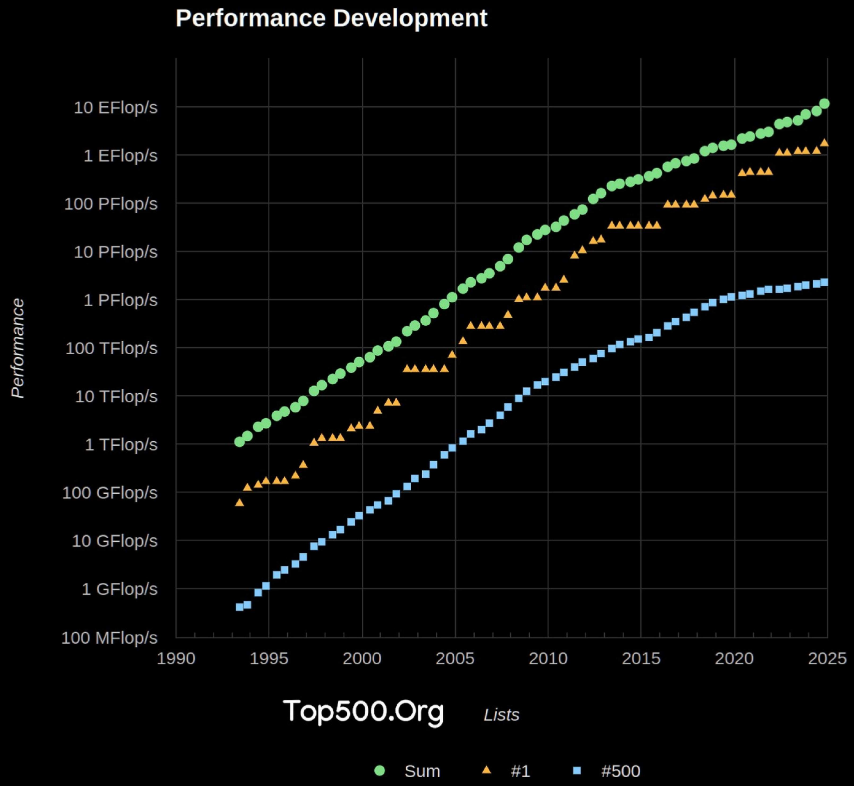
How Computers Get Faster: Moore’s Law

- 1965 prediction
  - Not about chip speed
  - Circuit complexity 2X every 18-24 months
- Speedup is mostly about **parallel processing**



Moore’s Law is still sort-of OK...





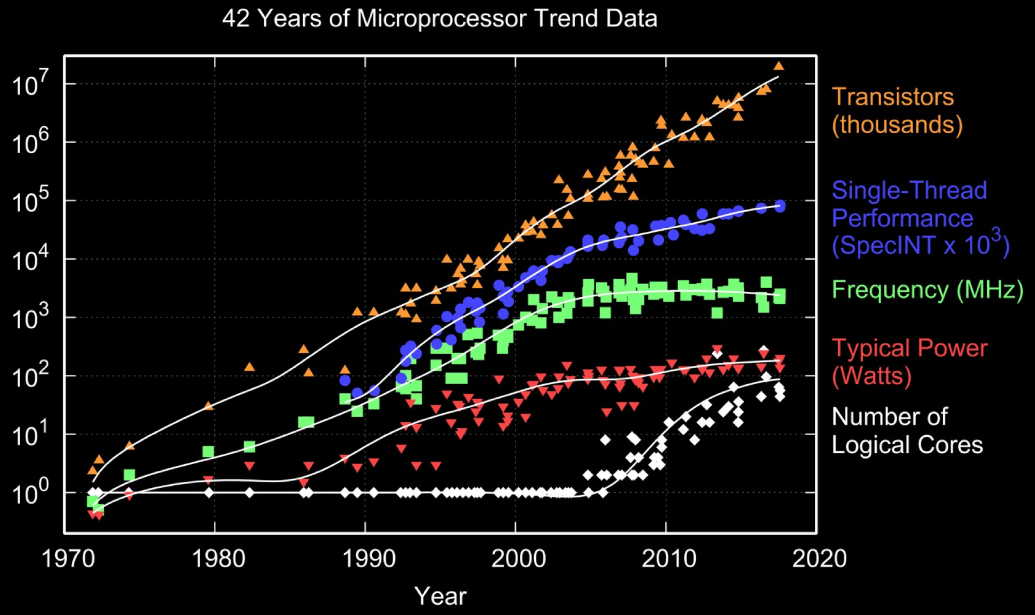
Parallel Processing

- Break program into ***N*** pieces that can execute simultaneously
  - **Scalable**: bigger ***N***, more speedup
  - **Modular hardware**
  - Can be **fault tolerant** using redundancy
- This scales up forever, right?

El Capitan supercomputer:
11,039,616 cores, 2.746 Exaflop/s
Cost approx. $600M, 29.6 MW

All The Bad News

- Moore's Law slowing
- Power/transistor ▼ slower than transistors/chip ▲
- Individual ops not getting much faster



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp


QUANTUM OF
SOLACE

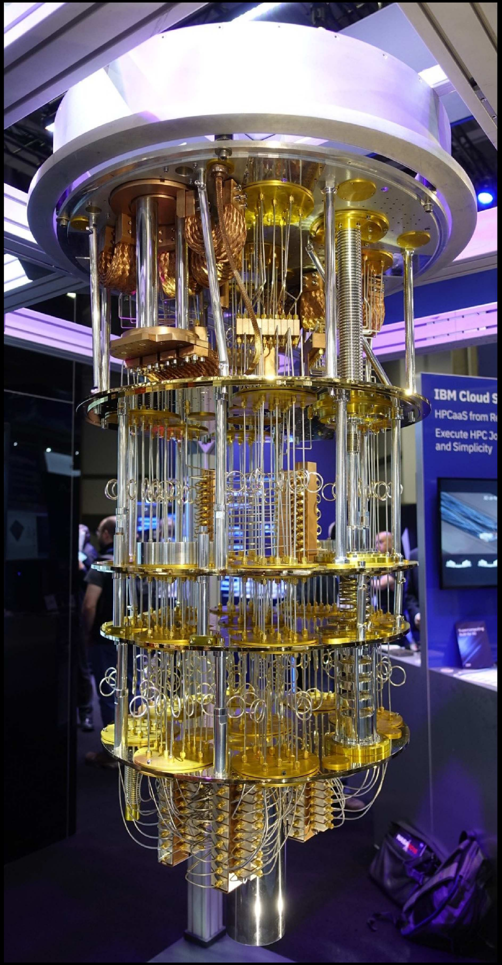
The Solace of Quantum

- Massively-parallel processing ***without*** massively parallel hardware
- Not limited by continuation of Moore’s Law; efficiently solve NP-complete problems?
- Potentially very low power consumption per unit computation performed

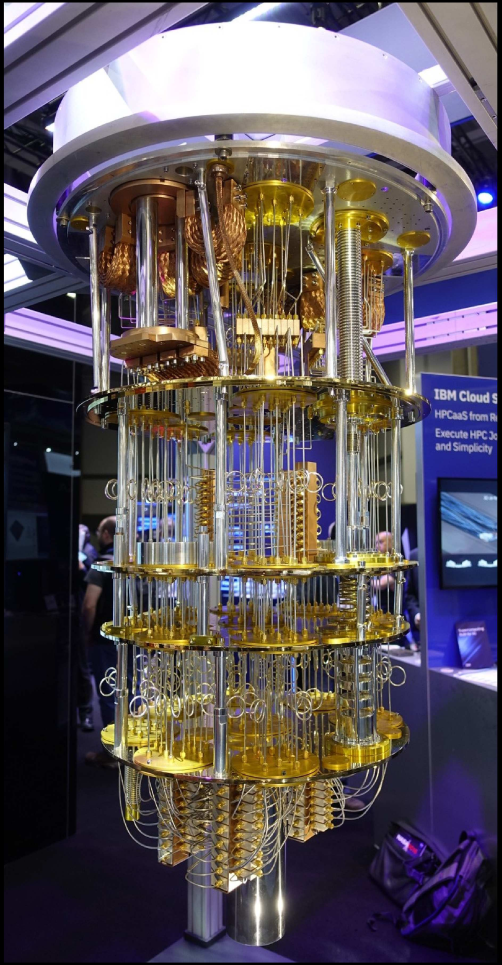
Quantum Computing

- State-of-the-art conventional processor chips already depend on quantum phenomena, but just implement conventional logic with it
- **Quantum Computing** is about using quantum phenomena to ***implement a different model***
  - **Quantum gates**
  - Other models (e.g., Adiabatic optimization)

Is This A Quantum Computer?



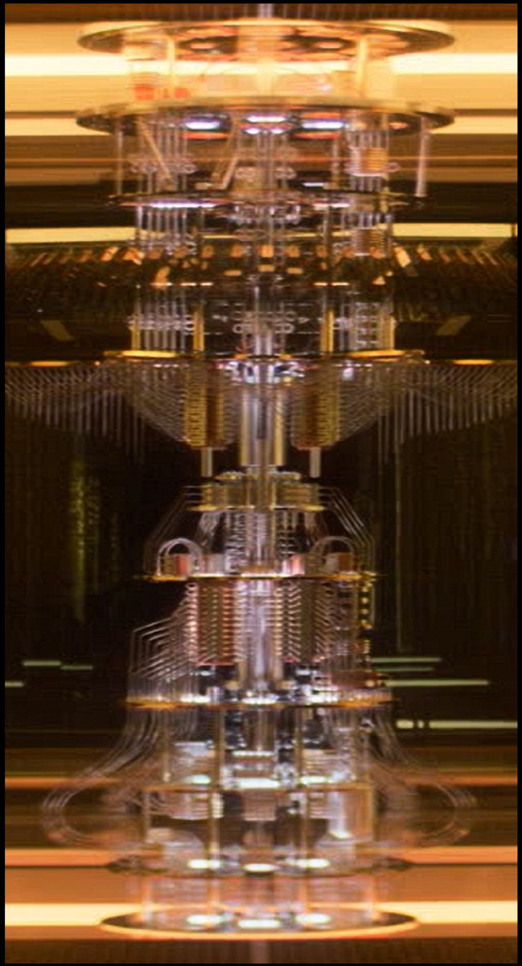
Is This A Quantum Computer?



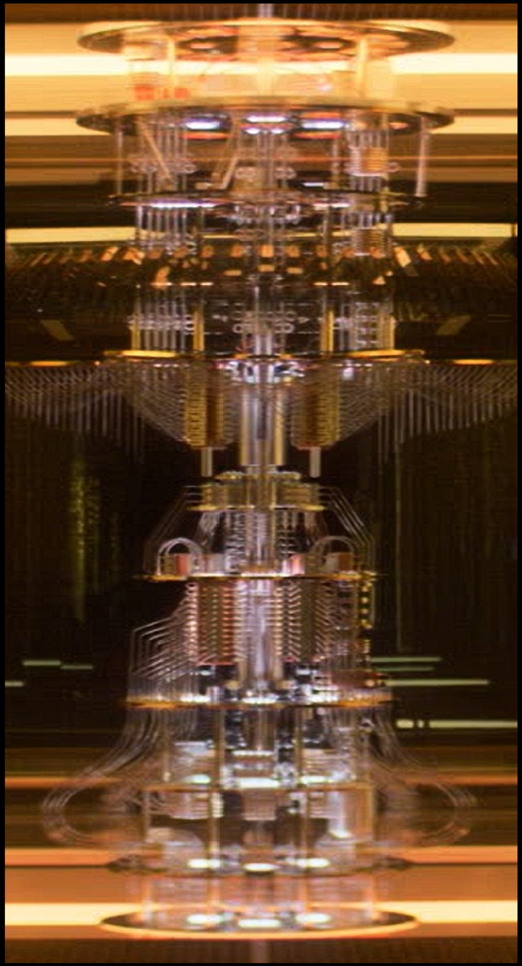
Yup!

IBM Q

Is This A Quantum Computer?



Is This A Quantum Computer?



Nope!

TV show:
DEVS

Is This A Quantum Computer?

Is This A Quantum Computer?

Yup!

Google Sycamore

Is This A Quantum Computer?

Is This A Quantum Computer?

Yup!

SpinQ
Gemini Mini
~$8K

Is This A Quantum Computer?

Is This A Quantum Computer?

Nope.

PBP

Not quantum, but similar properties

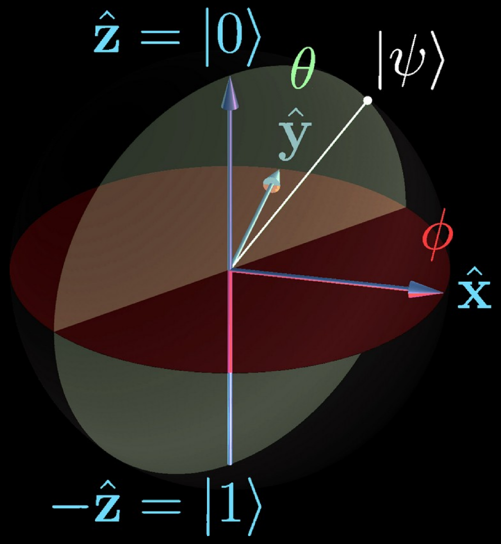
Conventional Computing

- **Memory** is made of **Bits**, each holding 0 or 1
  - Bit values reliably persist ***forever***
  - **Every bit** can be accessed by addressing
- **Processor** (perhaps one of many in a system)
  - Gates: **`AND`**, **`OR`**, **`XOR`**, **`NOT`**, **`NAND`**, **`NOR`**, **`MUX`**…
  - **Fanout** is allowed (e.g., FOF = fanout of 4)

Quantum Computing

- **Memory** is made of **Qubits**, each holding a ***probability density function*** over 0 and 1
  - Qubit value collapses to 0 or 1 when read
  - Values have a limited lifespan (decoherence)
- **Processor** (really PIM: processors in memory)
  - Gates: **`NOT`**, **`CNOT`**, **`CCNOT`**, **`SWAP`**, **`CSWAP`**…
  - **Fanout** is **not** allowed

Bloch Sphere Qubit Model



$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle$$
$$= \cos(\theta/2)|0\rangle + (\cos\phi + i\sin\phi)\sin(\theta/2)|1\rangle$$

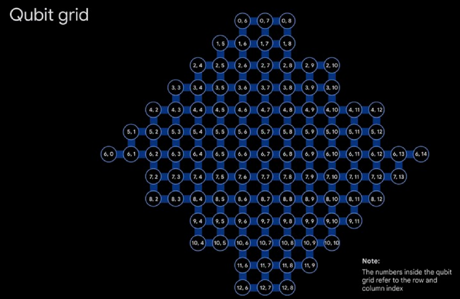
where $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$

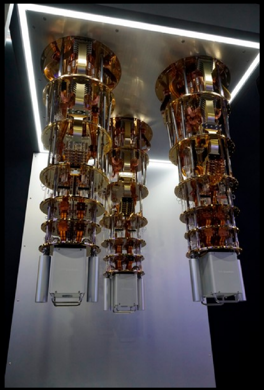
- Probability by coordinates on sphere surface
- Value of a Qubit is really a ***wave function***

But Where’s The Parallelism?

- Bloch sphere is one qubit in **superposition**
- Multiple qubits can be **entangled** so that ***E* Qubits** hold a ***probability density function*** over **all 2^E-bit values**
  - Each qubit holds up to 2^E bits
  - One operation on one qubit gives 2^E results

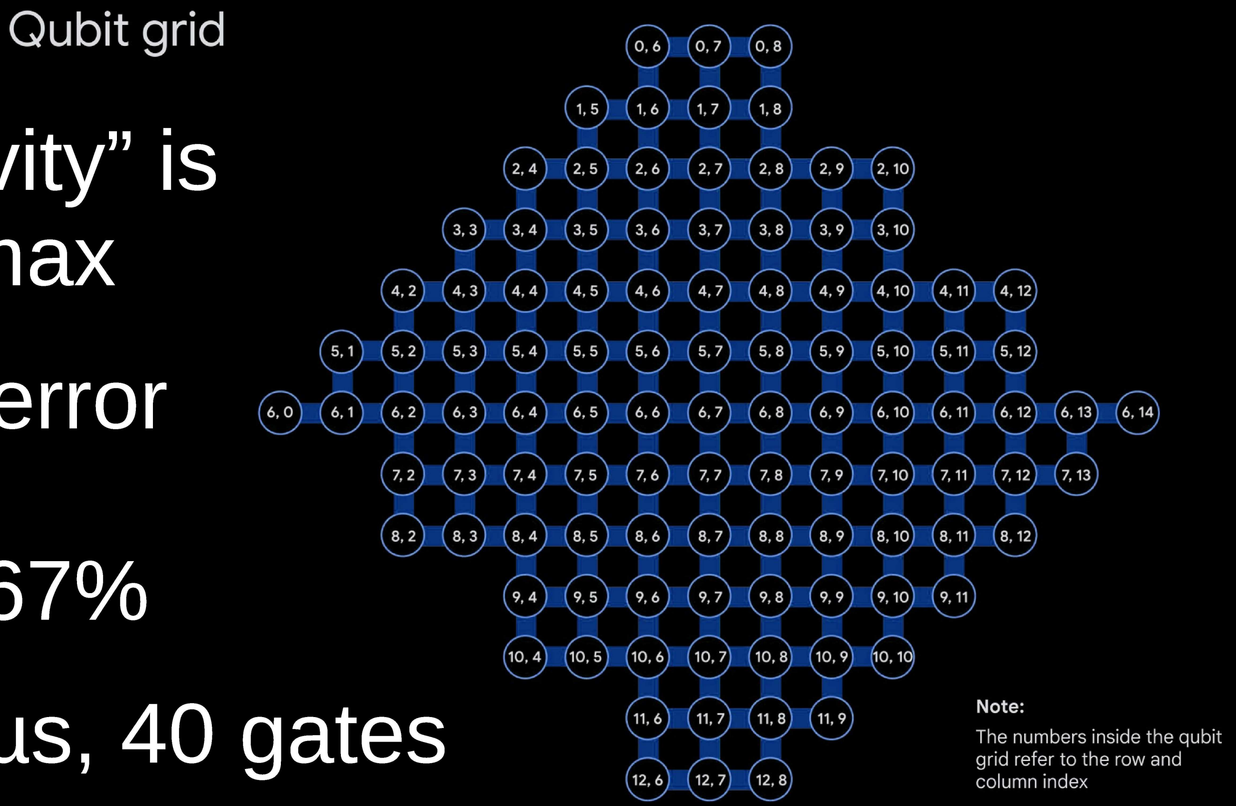
Some Quantum Computers





- IBM Condor: 1,121 superconducting
- China’s Tianyan-504: 504 superconducting
- Google Willow: 105 superconducting
- Rigetti Ankaa-3: 84 superconducting
- Fujitsu: 64 superconducting, 10 spin

Some Details About Willow...



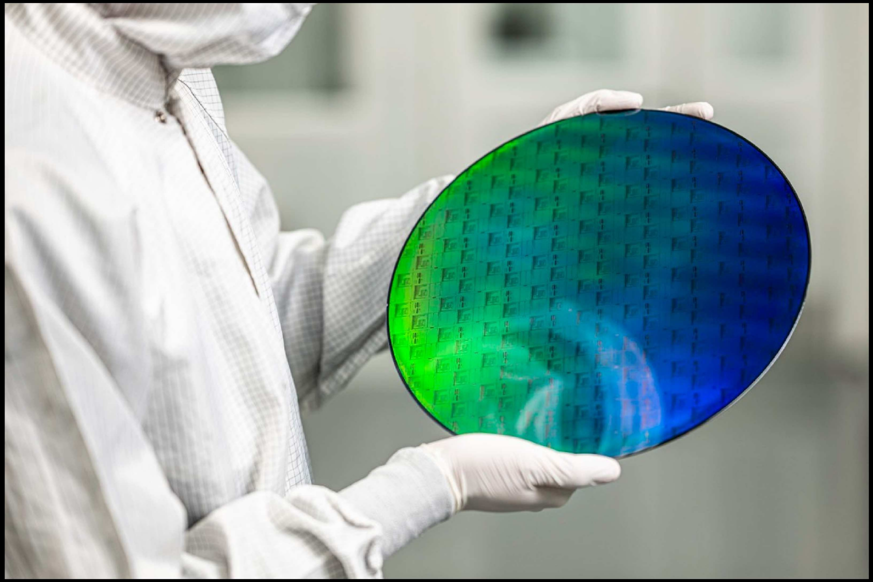
- **Willow** “connectivity” is 3.47 average, 5 max
- Gate(q): 0.036% error
  Gate(q,q): 0.14%
  Measurement: 0.67%
- Coherent for ~98µs, 40 gates

More Quantum Computers

- Intel Tunnel Falls: 12 spin
- Microsoft Majorana 1: 8 topological
- Quantinuum H2-1: 56 trapped ion
- IonQ Forte Enterprise: 36 trapped ion
- SpinQ: 20 superconducting, 2 spin (NMR)

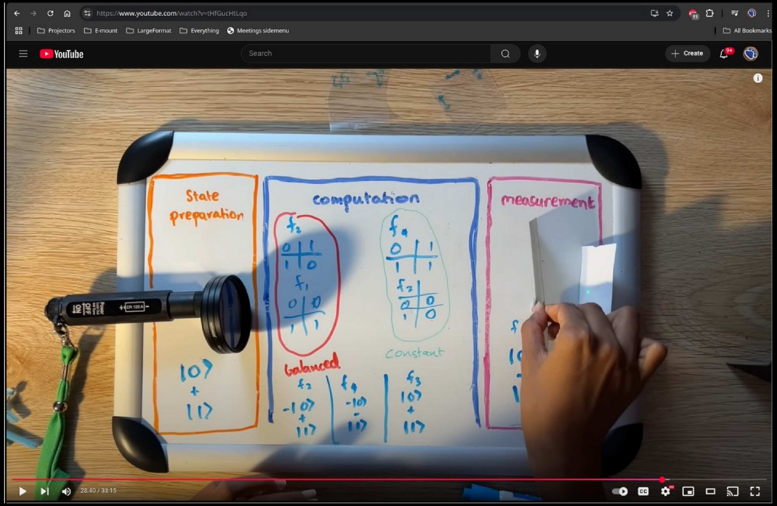
Photonic Quantum Computers

- PsiQuantum: photonic chips
- QCI: photonic chips



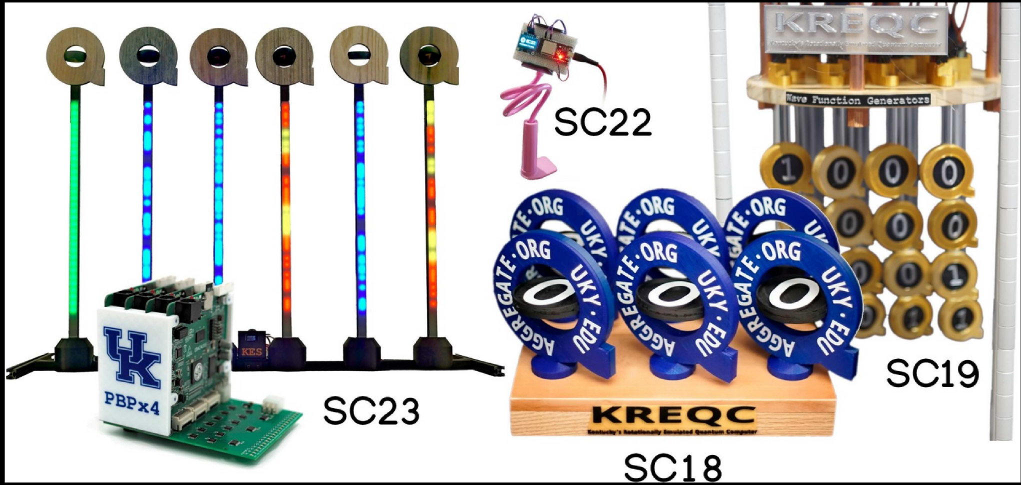
Not As Programmable...

- DWave: 5000+ adiabatic quantum annealing
- Xanadu: 12 boson sampling photonic
- Homemade: 1 photonic

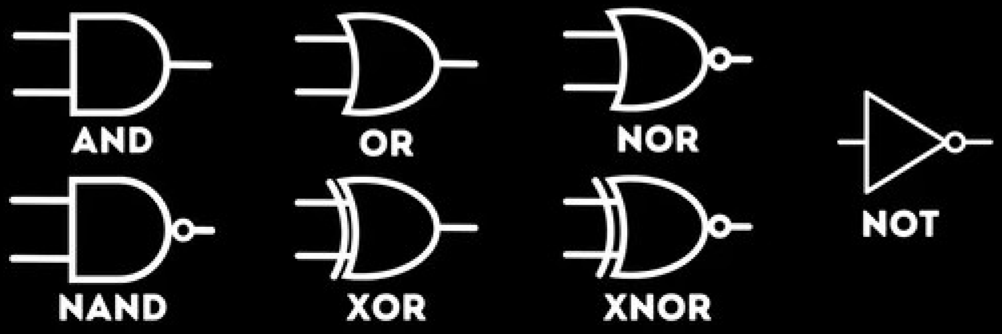


Not Quantum

- **PBP: Parallel Bit Pattern**
  - Entangled superposition via **symbolic comp.**
  - Qubit ⇒ pbit (well, sort-of...)
  - **1024+ pbits**
  - 6 to **32-way** entangled
  - 10-way 1024 pbits on a $3 micro!



Conventional Logic Gates



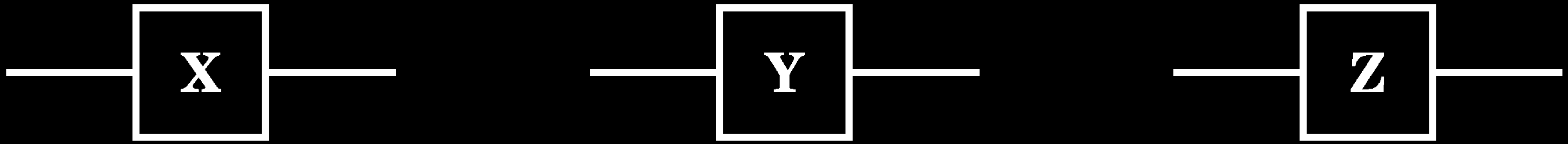
NOT gate

A	$\bar{A}$
0	1
1	0

Inputs		Outputs					
A	B	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

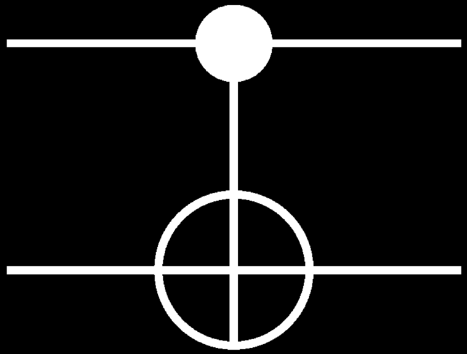
- Inputs are **absorbed**
- Output is **generated**, can **drive multiple inputs**

Quantum Gate Types: **Pauli**



- **Pauli `X`** is also known as **`NOT`**
  - Rotates Bloch Sphere around X by π radians
  - Functions like conventional **`NOT`**
  - **`NOT`** is its own inverse
- **Pauli `Y`** rotates around Y and **Pauli `Z`** around Z

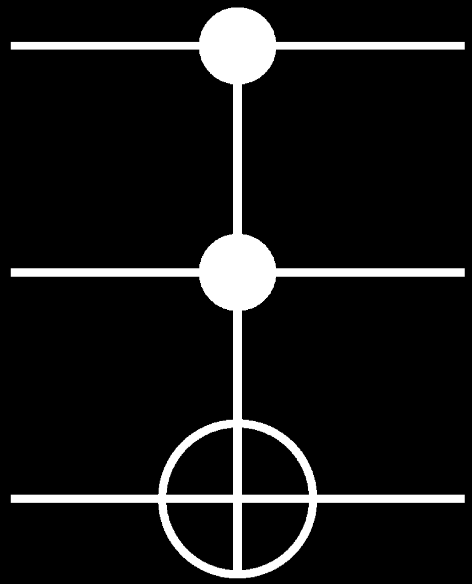
Quantum Gate Types: CNOT



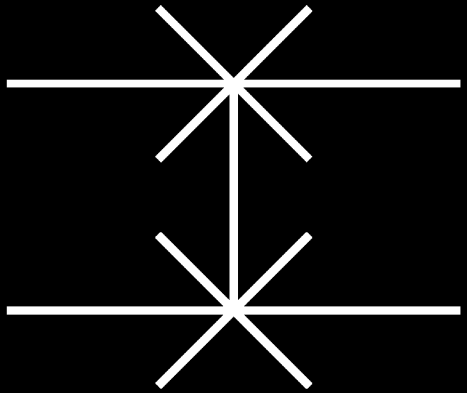
- CNOT is the **Controlled NOT** gate
  - Top input is control, passes thru unchanged
  - Bottom input is inverted where control is 1
  - Both inputs can't be the same Qubit
  - Similar to conventional XOR gate

Quantum Gate Types: Toffoli

- **Toffoli** is also known as `CCNOT`, **Controlled Controlled** `NOT`
  - A classical universal gate
  - Top two inputs pass unchanged
  - Bottom input is inverted where both control inputs are 1
  - Behaves like `C = (A AND B) XOR C`



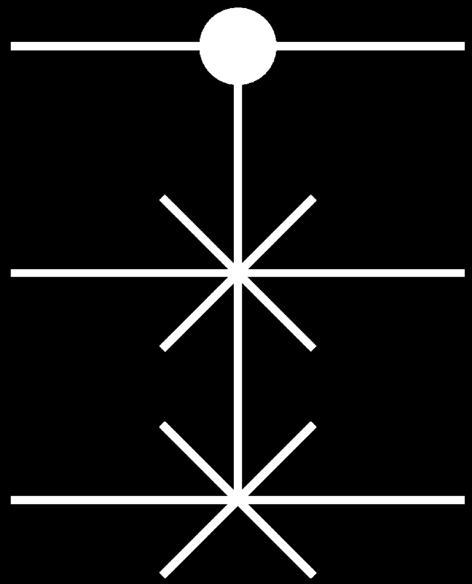
Quantum Gate Types: SWAP



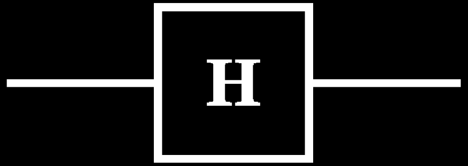
- **SWAP** exchanges values of two Qubits
  - Seems pointless...
    but this is a reversible assignment

Quantum Gate Types: Fredkin

- **Fredkin** is also known as **`CSWAP`**, **Controlled `SWAP`**
  - A classical universal gate… and *billiard-ball* conservative
  - Top input passes unchanged
  - Bottom inputs are swapped where top control input is 1
  - Behaves like paired conventional `MUX`es



Quantum Gate Types: Hadamard



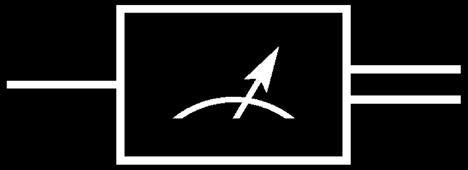
- **Hadamard** is not like any conventional gate
  - A Qubit can only be initialized to 0 or 1
  - Hadamard operator converts that into the equiprobable superposed state: 50% 0, 50% 1
- If *applied in parallel* to ***E*** Qubits, the result is the equiprobable ***E***-way entangled superposition

Equiprobable *E*-Way Entangled Superposition?

- Up to this point, nothing about Quantum Computing sounded better than conventional…
- Suppose we apply **H** in parallel to 16 Qubits?
  - Those 16 Qubits will hold all 65,536 possible 16-bit values with equal probabilities
  - Any single operation on any of those Qubits will effectively operate on all 65,536 values

Parallel processing without parallel hardware!

Quantum Gate Types: **Measurement**

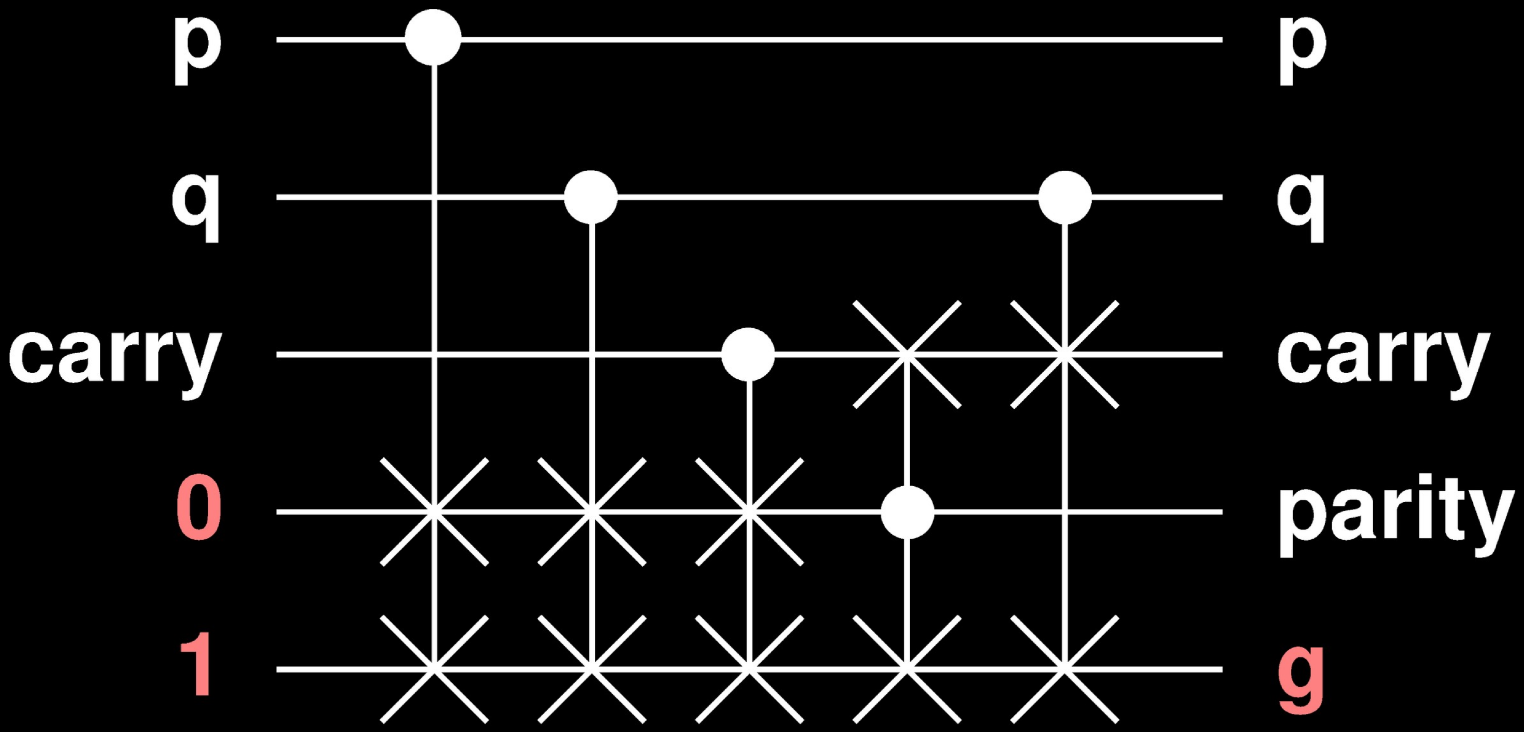


- **Measurement** collapses a superposition
  - Superposed Qubit becomes either 0 or 1
  - Superposed probability density function is randomly sampled, determines odds of 0 vs. 1

Exponentially cheap parallel computation…

but you only get to read-out one answer per run

Let’s Build A 1-Bit Full Adder



{carry, parity} = p + q + carry

Let’s Build A 1-Bit Full Adder

KREQC Program

```
// 1-bit full adder
p=1;
q=1;
carry=0;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);
```

Simulation Output

```
QUBIT                           g   parity    carry        q        p
                 32 |     64 |       0 |       0 |      64 |      64 |
CSWAP               |        x-------x-------|-------|-------@
                 32 |      0 |      64 |       0 |      64 |      64 |
CSWAP               |        x-------x-------|-------@       |
                 32 |     64 |       0 |       0 |      64 |      64 |
CSWAP               |        x-------x-------@       |       |
                 32 |     64 |       0 |       0 |      64 |      64 |
CSWAP               |        x-------@-------x       |       |
                 32 |     64 |       0 |       0 |      64 |      64 |
CSWAP               |        x-------|-------x-------@       |
                 32 |      0 |       0 |      64 |      64 |      64 |
                    1        0       0       1       1       1

                                g   parity    carry        q        p
     64/64                      0        0        1        1        1
```

Now Give It Superposed Input

KREQC Program

```
// 1-bit full adder
p=1;
q=0;
carry=?;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);
```

Simulation Output

```
QUBIT                  g  parity   carry       q       p
            32 |    64 |     0 |    32 |     0 |    64 |
CSWAP          |       x-------x-------|-------|-------@
            32 |     0 |    64 |    32 |     0 |    64 |
CSWAP          |       x-------x-------|-------@       |
            32 |     0 |    64 |    32 |     0 |    64 |
CSWAP          |       x-------x-------@       |       |
            32 |    32 |    32 |    32 |     0 |    64 |
CSWAP          |       x-------@-------x       |       |
            32 |    32 |    32 |    32 |     0 |    64 |
CSWAP          |       x-------|-------x-------@       |
            32 |    32 |    32 |    32 |     0 |    64 |
               0       1       0       1       0       1
```

	g	parity	carry	q	p
32/64	0	1	0	0	1
32/64	1	0	1	0	1

KREQC Program

```
// 1-bit full adder
p=?;
q=?;
carry=?;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);
```

Simulation Output

```
QUBIT                     g  parity   carry        q        p
            32 |     64 |      0 |     32 |     32 |     32 |
CSWAP          |        x-------x-------|-------|-------@
            32 |     32 |     32 |     32 |     32 |     32 |
CSWAP          |        x-------x-------|-------@         |
            32 |     32 |     32 |     32 |     32 |     32 |
CSWAP          |        x-------x-------@        |        |
            32 |     32 |     32 |     32 |     32 |     32 |
CSWAP          |        x-------@-------x        |        |
            32 |     48 |     32 |     16 |     32 |     32 |
CSWAP          |        x-------|-------x-------@         |
            32 |     32 |     32 |     32 |     32 |     32 |
               1        1       0       0       0       0
```

	g	parity	carry	q	p
8/64	0	0	1	1	1
8/64	0	1	0	0	1
8/64	0	1	0	1	0
8/64	0	1	1	1	1
8/64	1	0	0	0	0
8/64	1	0	1	0	1
8/64	1	0	1	1	0
8/64	1	1	0	0	0

Programming: IBM Qiskit

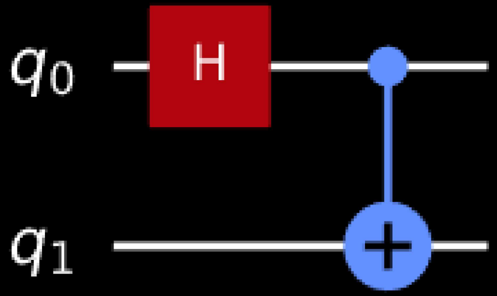
```python
from qiskit import QuantumCircuit
from qiskit.quantum_info import SparsePauliOp
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_ibm_runtime import EstimatorV2 as Estimator

# Create a new circuit with two qubits
qc = QuantumCircuit(2)

# Add a Hadamard gate to qubit 0
qc.h(0)

# Perform a controlled-X gate on qubit 1, controlled by qubit 0
qc.cx(0, 1)

# Return a drawing of the circuit using MatPlotLib ("mpl"). This is the
# last line of the cell, so the drawing appears in the cell output.
# Remove the "mpl" argument to get a text drawing.
qc.draw("mpl")
```



Programming: Microsoft Q#

Q# Copy

```
// The Q# compiler automatically detects the Main() operation as the entry point.

operation Main() : Result {
    // Allocate a qubit. By default, it's in the 0 state.
    use q = Qubit();
    // Apply the Hadamard operation, H, to the state.
    // It now has a 50% chance of being measured as 0 or 1.
    H(q);
    // Measure the qubit in the Z-basis.
    let result = M(q);
    // Reset the qubit before releasing it.
    Reset(q);
    // Return the result of the measurement.
    return result;
}
```

Programming: PBP C++

```
void pbitripple() {
 pbit a0(0), a1(0), a2(0), a3(0);
 pbit b0(1), b1(0), b2(0), b3(0);
 pbit z(0), x(0);
 H(a0, 0); // unlike Qubits,
 H(a1, 1); // must specify groups of
 H(a2, 2); // entanglement channels
 H(a3, 3); // for Hadamard gates
 CNOT(a1,b1); CNOT(a2,b2);
 CNOT(a3,b3); CNOT(a1,x);
 CCNOT(a0,b0,x); CNOT(a2,a1);
 CCNOT(x,b1,a1); CNOT(a3,a2);
 CCNOT(a1,b2,a2); CNOT(a3,z);
 CCNOT(a2,b3,z); NOT(b1);
 NOT(b2); CNOT(x,b1);
 CNOT(a1,b2); CNOT(a2,b3);
 CCNOT(a1,b2,a2);
 CCNOT(x,b1,a1);
 CNOT(a3,a2); NOT(b2);
 CCNOT(a0,b0,x); CNOT(a2,a1);
 NOT(b1); CNOT(a1,x);
 CNOT(a0,b0); CNOT(a1,b1);
 CNOT(a2,b2); CNOT(a3,b3);
 SETMEAS(); // pick random channel
 printf("a=%d b=%d\n",
  MEAS(a0)+(MEAS(a1)<<1) +
   (MEAS(a2)<<2)+(MEAS(a3)<<3),
  MEAS(b0)+(MEAS(b1)<<1)+
   (MEAS(b2)<<2)+(MEAS(b3)<<3));
}
```

```
void pintsqrt(int val){
 pint a(val); // 8-bit number
 pint b = pint(0).Had(4); // dim 0-3
 pint c = (b * b); // square them
 pint d = (c == a); // select answer
 int pos = d.First();
 printf("Square root of %d is %d\n",
  val, pos);
}
```

```
#include "pbp.h"

void pintfactor(int val) {
 pint a(val); // 8-bit number
 pint b = pint(0).Had(4); // dim 0-3
 pint c = pint(0).Had(4,4); // dim 4-7
 pint d = b * c; // multiply 'em
 pint e = (d == a); // which were val?
 pint f = e * b; // zero non-answers
 int spot = f.First(); // factors
 int one = c.Meas(spot);
 int two = b.Meas(spot);
 printf("%d, %d are factors of %d\n",
  one, two, val);
}
```

Quantum Supremacy or Advantage

Solving a useful problem faster than any classical computer could

- 2019 Google's 53-qubit **Sycamore**
- 2020 China's 113-qubit **Jiuzhang**
- 2021 IBM's 127-qubit **Eagle**
- 2024 Google's 105-qubit **Willow ...**

Is **Random Circuit Sampling useful?**

So, What Is Quantum Good For?

- Problems where:
  - You need to try all possible values
  - You don’t need all answers, just one or some
  - You don’t mind occasionally wrong answers
  - Combinatorial logic operating on few qubits

- **Quantum computers are special-purpose attached accelerators**, sort-of like GPUs

Conclusions

- Quantum computing is a way past Moore's Law, for very specific types of computations
- Quantum computers still have a long way to go
  - Quantum hardware *might* never get there
  - Thinking about quantum algorithms often yields faster algorithms for conventional computers
- RSA cryptography uses 2048-bit keys, **Shor's Algorithm** would need >1M qubits to break it