CPEN 400Q / EECE 571Q Lecture 01
Overview and intro to gate model quantum computing

Tuesday 11 January 2022

Course learning outcomes

Core goal: **learn how to program quantum computers** in a hands-on, software-focused setting.

- Describe the societal importance and implications of quantum computing
- Explain the theory and principles behind gate-model quantum computing
- Describe the operation of key quantum algorithms
- Implement basic and research-level quantum algorithms using Python and PennyLane

In this course you will implement everything you learn!

Covered in this course

- qubit states, operations, and measurements
- quantum circuits
- basic quantum algorithms
- quantum compilation
- Grover's algorithm
- Shor's algorithm
- variational quantum algorithms
- Hamiltonian simulation
- considerations for running on near-term hardware

Not covered in this course

- How the hardware actually works
- Non-gate-model quantum computing
  - adiabatic quantum computing
  - quantum annealing
  - measurement-based quantum computing
  - continuous-variable quantum computing
- Advanced theoretical topics (quantum information, complexity theory, quantum error correction)

Logistics

	Olivia Di Matteo	Gideon Uchehara (TA)
Office:	KAIS TBD	KAIS 4075
Office hrs:	W/F 13:00-14:00 or by appointment.	M 15:30-16:30
	Open-door policy after 12:00 weekdays.	Bimonthly tutorials W 15:00-16:00 (same room, dates TBD)
Email:	`olivia@ece.ubc.ca`	`gideon.uchehara@ece.ubc.ca`
GitHub:	`glassnotes` / `odimatte`	`gideonuchehara` / `gideonu`

All lecture slides/demos will be posted on the course GitHub:

`https://github.com/glassnotes/CPEN-400Q`

Textbook

Primary resource will be the **free**, **online** Xanadu Quantum Codebook. Use to learn about the content, PennyLane, and do practice programming exercises to supplement class work.

`codebook.xanadu.ai`

Secondary (optional) resource: *Introduction to Quantum Computation and Quantum Information*, Nielsen and Chuang.

Assignments and grading

Three components:

- 30% assignments
- 20% in-class quizzes
- 50% final group project

No exams!

Few extra things for EECE 571Q students.

Assignments and grading, cntd.

30% assignments (5-6; lowest grade dropped):

- Distributed via GitHub

Assignments and grading, cntd.

30% assignments (5-6; lowest grade dropped):

- Distributed via GitHub
- Implement algorithms and solve quantum computing programming problems in PennyLane

Assignments and grading, cntd.

30% assignments (5-6; lowest grade dropped):

- Distributed via GitHub
- Implement algorithms and solve quantum computing programming problems in PennyLane
- Work alone or together, but what you submit must be your own work (and cite your collaborators!)

Assignments and grading, cntd.

30% assignments (5-6; lowest grade dropped):

- Distributed via GitHub
- Implement algorithms and solve quantum computing programming problems in PennyLane
- Work alone or together, but what you submit must be your own work (and cite your collaborators!)
- *EECE 571Q students: +1 more challenging question*

Assignments and grading, cntd.

30% assignments (5-6; lowest grade dropped):

- Distributed via GitHub
- Implement algorithms and solve quantum computing programming problems in PennyLane
- Work alone or together, but what you submit must be your own work (and cite your collaborators!)
- *EECE 571Q students: +1 more challenging question*

Assignments and grading, cntd.

30% assignments (5-6; lowest grade dropped):

- Distributed via GitHub
- Implement algorithms and solve quantum computing programming problems in PennyLane
- Work alone or together, but what you submit must be your own work (and cite your collaborators!)
- *EECE 571Q students: +1 more challenging question*

You have received instructions to access "Assignment 0". This is meant to be a straightforward review of linear algebra, Python, and NumPy, and introduce you to the GitHub workflow.

Assignments and grading, cntd.

20% quizzes (weekly; lowest grade dropped):

- Starting next Tuesday 18 Jan; done at **end** of class

Assignments and grading, cntd.

20% quizzes (weekly; lowest grade dropped):

- Starting next Tuesday 18 Jan; done at **end** of class
- Solve a small programming problem (individually)

Assignments and grading, cntd.

20% quizzes (weekly; lowest grade dropped):

- Starting next Tuesday 18 Jan; done at **end** of class
- Solve a small programming problem (individually)
- Primary purpose is for reviewing content

Assignments and grading, cntd.

20% quizzes (weekly; lowest grade dropped):

- Starting next Tuesday 18 Jan; done at **end** of class
- Solve a small programming problem (individually)
- Primary purpose is for reviewing content
- Grading scale:
  - 0: incomplete
  - 1: (partially) complete / attempted but incorrect
  - 2: complete and correct

Assignments and grading, cntd.

50% final project:

- Implement the methods / algorithms in a research paper and reproduce the results.

Assignments and grading, cntd.

50% final project:

- Implement the methods / algorithms in a research paper and reproduce the results.
- Work in groups of $\sim$4 (*EECE 571Q students: in pairs*)

Assignments and grading, cntd.

50% final project:

- Implement the methods / algorithms in a research paper and reproduce the results.
- Work in groups of $\sim$4 (*EECE 571Q students: in pairs*)
- Three components:
  - 25-30 minute in-class lecture and demonstration
  - Fully-documented software implementation
  - Companion report
  - *EECE 571Q students: include complexity analysis, resource estimates, and run on hardware or justify why you can't*

Assignments and grading, cntd.

50% final project:

- Implement the methods / algorithms in a research paper and reproduce the results.
- Work in groups of $\sim$4 (*EECE 571Q students: in pairs*)
- Three components:
  - 25-30 minute in-class lecture and demonstration
  - Fully-documented software implementation
  - Companion report
  - *EECE 571Q students: include complexity analysis, resource estimates, and run on hardware or justify why you can't*
- Can use a different quantum programming framework if you like (Qiskit, Q#, Quantum++, Cirq, Yao, etc.)

Assignments and grading, cntd.

50% final project:

- Implement the methods / algorithms in a research paper and reproduce the results.
- Work in groups of $\sim$4 (*EECE 571Q students: in pairs*)
- Three components:
  - 25-30 minute in-class lecture and demonstration
  - Fully-documented software implementation
  - Companion report
  - *EECE 571Q students: include complexity analysis, resource estimates, and run on hardware or justify why you can't*
- Can use a different quantum programming framework if you like (Qiskit, Q#, Quantum++, Cirq, Yao, etc.)

Assignments and grading, cntd.

50% final project:

- Implement the methods / algorithms in a research paper and reproduce the results.
- Work in groups of $\sim$4 (*EECE 571Q students: in pairs*)
- Three components:
  - 25-30 minute in-class lecture and demonstration
  - Fully-documented software implementation
  - Companion report
  - *EECE 571Q students: include complexity analysis, resource estimates, and run on hardware or justify why you can't*
- Can use a different quantum programming framework if you like (Qiskit, Q#, Quantum++, Cirq, Yao, etc.)

More details (rubric, suggested papers, etc.) to follow.

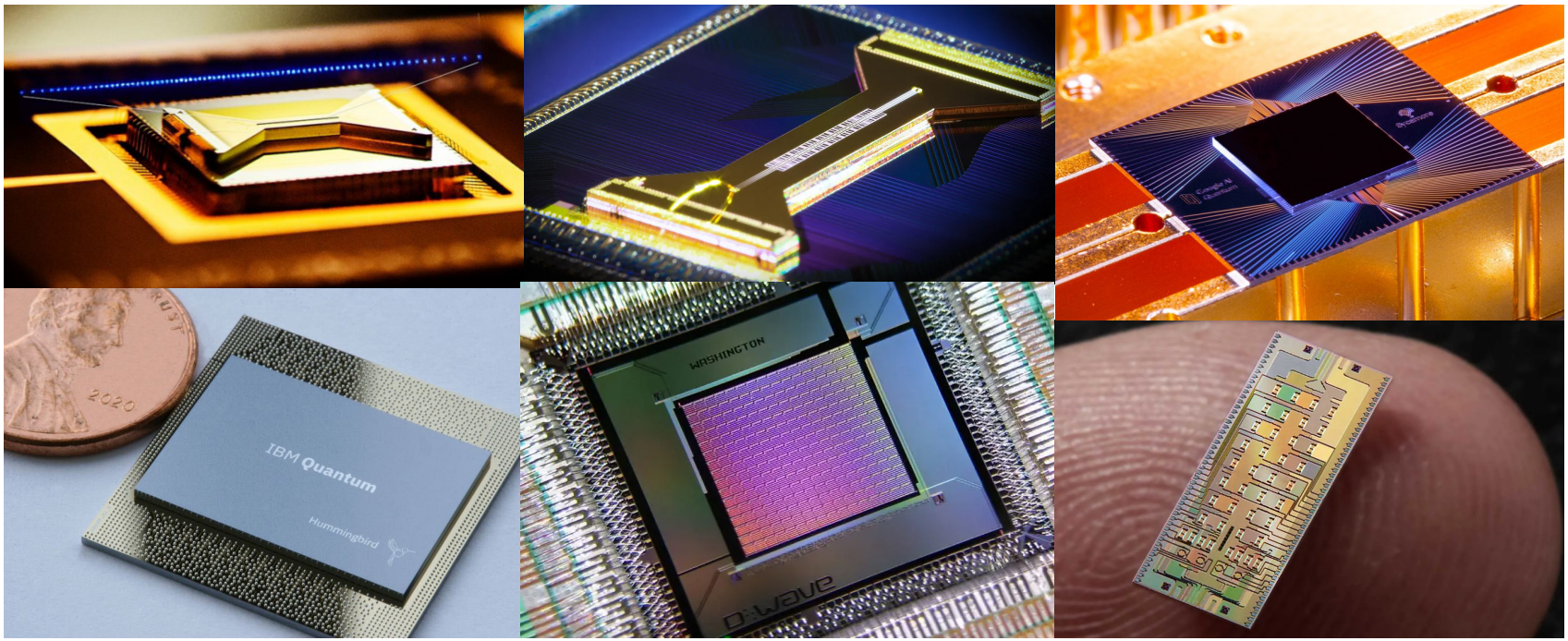
Today

Learning outcomes:

1. Define quantum computing, and explain its societal importance
2. Define a *qubit* both conceptually and mathematically
3. Simulate quantum computing on a single-qubit

Quantum computers are here

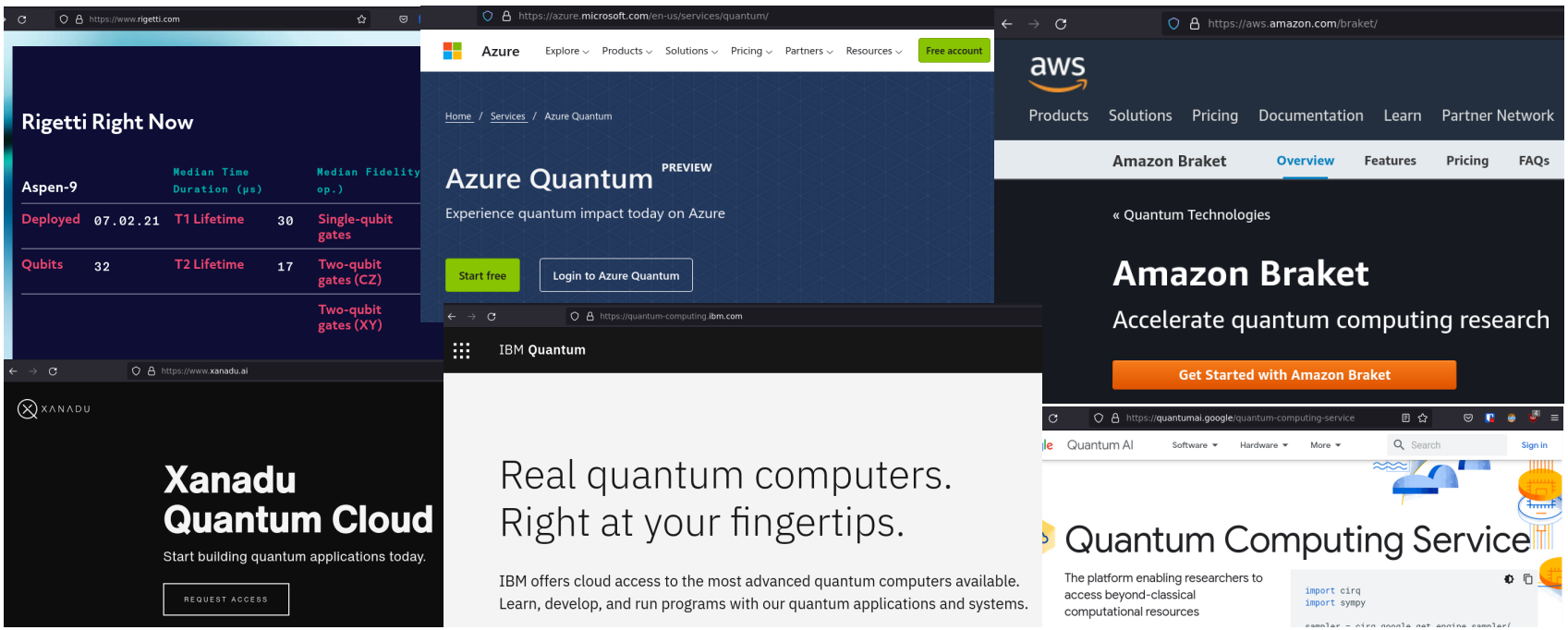
People are building them...



Commercial quantum processors (images clockwise from to left): IonQ (trapped-ion), Honeywell (trapped-ion), Google (superconducting), Xanadu (photonic), D-Wave (superconducting), IBM (superconducting).

Quantum computers are here

… and you can go use them *right now*!



But why? How? And for what?

Physical limitations

'Classical' computers are made more powerful by

- making smaller transistors
- putting more transistors onto a single chip
- using multiple processors in parallel

Physical limitations

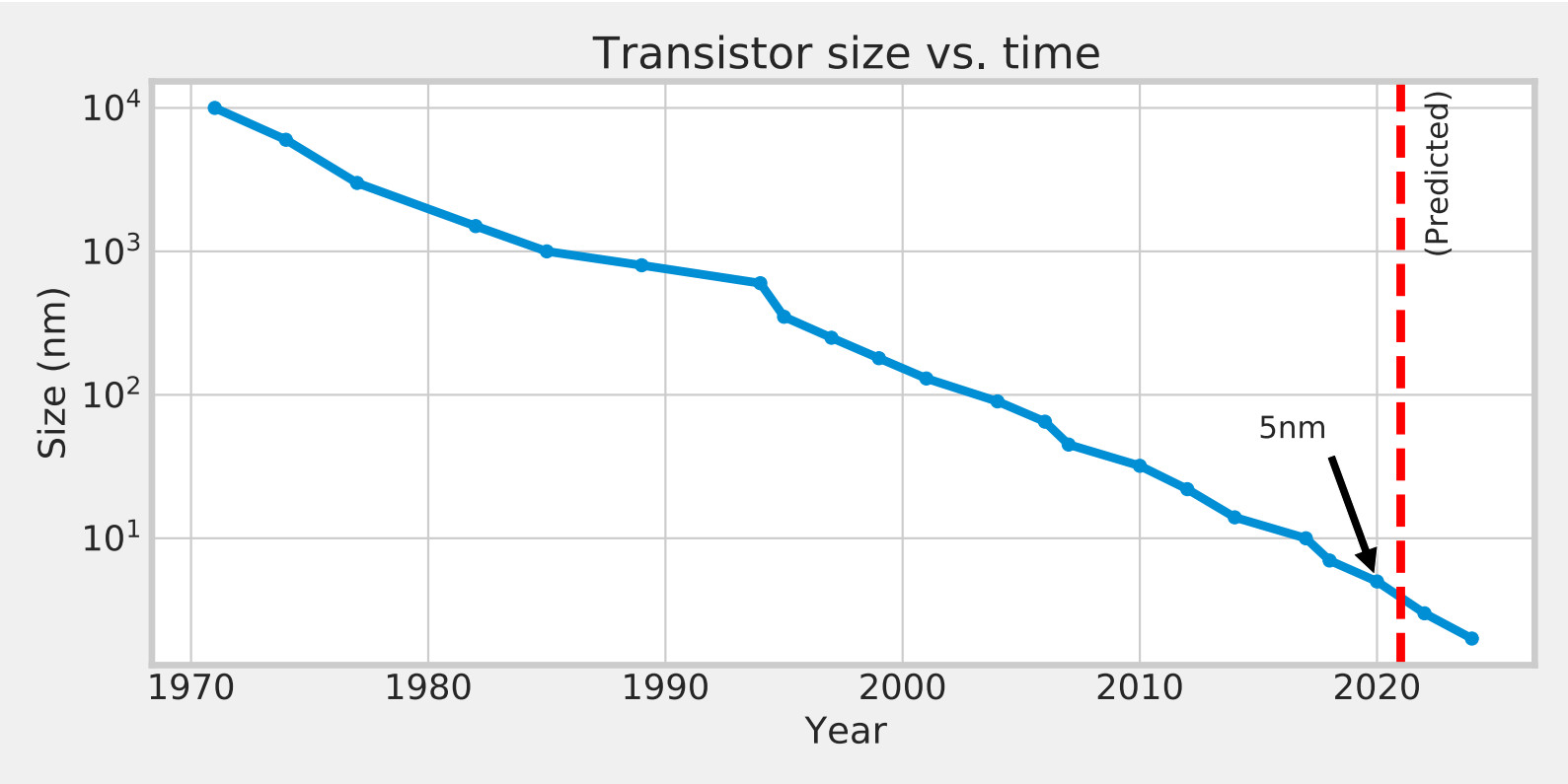
'Classical' computers are made more powerful by

- making smaller transistors
- putting more transistors onto a single chip
- using multiple processors in parallel

Moore's law suggests that every 18 months - 2 years the number of transistors on a chip will double, while the cost is halved.

Why quantum computing?

But we are approaching the physical limits of how small a transistor we can put on a chip before quantum effects will become a problem.



Data source: `https://en.wikipedia.org/wiki/Semiconductor_device_fabrication`

Computational limitations

Some problems will take an intractable amount of time to run on classical computers.

Computational limitations

Some problems will take an intractable amount of time to run on classical computers.

Parallelization can help, but we still cannot fully counteract the exponential complexity of some problems.

Computational limitations

Some problems will take an intractable amount of time to run on classical computers.

Parallelization can help, but we still cannot fully counteract the exponential complexity of some problems.

Sometimes that's a good thing:

- cryptographic infrastructure is built on such mathematically hard problems

But usually it just prevents us from doing interesting things.

Quantum computing

Nielsen & Chuang:

> *"Quantum computation and quantum information is the study of the information processing tasks that can be accomplished using quantum mechanical systems."*

Quantum computing involves manipulating and measuring the states of physical systems, most often **qubits** (quantum bits).

This course will teach you how to implement *gate-model* quantum algorithms that will be able to solve some computationally intractable problems, once we build a machine with enough high-quality qubits.

Key applications

- Searching large spaces

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6
- Simulating and calculating properties of physical systems

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks $\sim$8-9

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks $\sim$8-9
  - Hamiltonian simulation, week $\sim$11

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks $\sim$8-9
  - Hamiltonian simulation, week $\sim$11
- Optimization

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks $\sim$8-9
  - Hamiltonian simulation, week $\sim$11
- Optimization
  - QAOA, week $\sim$8

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks $\sim$8-9
  - Hamiltonian simulation, week $\sim$11
- Optimization
  - QAOA, week $\sim$8
- Machine learning

Key applications

- Searching large spaces
  - Grover's algorithm, week $\sim$3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks $\sim$4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks $\sim$8-9
  - Hamiltonian simulation, week $\sim$11
- Optimization
  - QAOA, week $\sim$8
- Machine learning
  - e.g., variational quantum classifier, week $\sim$9

Key applications

- Searching large spaces
  - Grover's algorithm, week ∼3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks ∼4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks ∼8-9
  - Hamiltonian simulation, week ∼11
- Optimization
  - QAOA, week ∼8
- Machine learning
  - e.g., variational quantum classifier, week ∼9
- Things we haven't thought of yet!

Key applications

- Searching large spaces
  - Grover's algorithm, week ∼3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks ∼4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks ∼8-9
  - Hamiltonian simulation, week ∼11
- Optimization
  - QAOA, week ∼8
- Machine learning
  - e.g., variational quantum classifier, week ∼9
- Things we haven't thought of yet!

Key applications

- Searching large spaces
  - Grover's algorithm, week ∼3
- Discrete mathematics, (breaking) cryptography
  - Shor's algorithm, weeks ∼4-6
- Simulating and calculating properties of physical systems
  - variational algorithms, weeks ∼8-9
  - Hamiltonian simulation, week ∼11
- Optimization
  - QAOA, week ∼8
- Machine learning
  - e.g., variational quantum classifier, week ∼9
- Things we haven't thought of yet!

We are already capable of doing some of these on today's devices.

The NISQ era

Today's quantum computers are known as **noisy, intermediate scale quantum** (NISQ) devices.

Quantum Computing in the NISQ era and beyond

John Preskill

Institute for Quantum Information and Matter and Walter Burke Institute for Theoretical Physics, California Institute of Technology, Pasadena CA 91125, USA

Published: 2018-08-06, volume 2, page 79

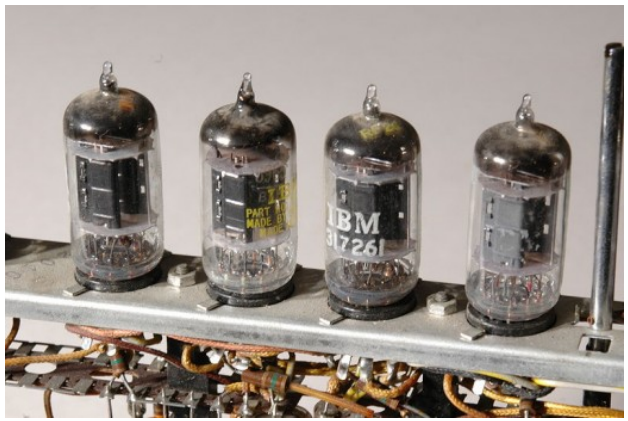
Eprint: arXiv:1801.00862v3

Doi: https://doi.org/10.22331/q-2018-08-06-79

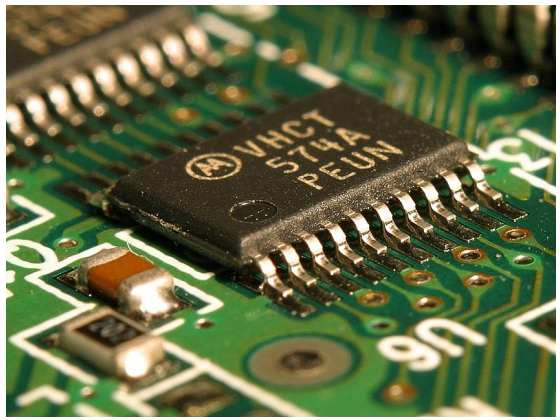
Citation: Quantum 2, 79 (2018).

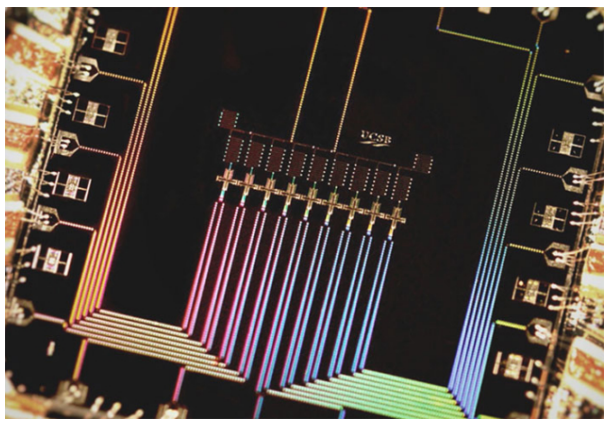
The NISQ era

Many competing technologies; still too early to know which (combination?) of them, if any, will win out long term.



vs.





vs.

???

Image credits:
https://hubpages.com/business/What-Is-a-Transistor-and-Why-is-it-Important
https://en.wikipedia.org/wiki/Solid-state_electronics
https://physicsworld.com/a/google-gains-new-ground-on-universal-quantum-computer/

The NISQ era

Largest device (IBM's "Eagle") has 127 superconducting qubits.

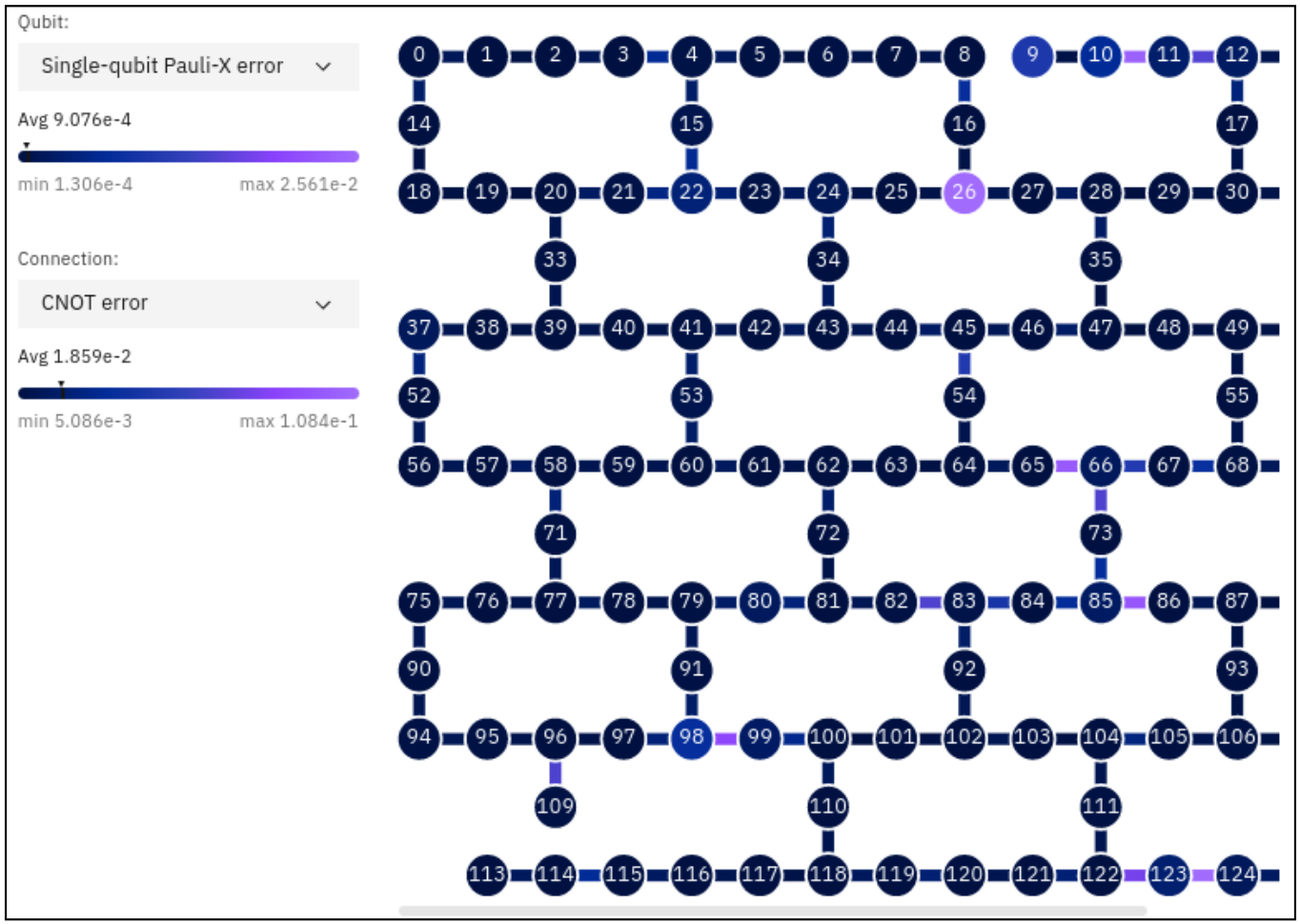


Image source: screencap 2022-01-11 from `https://quantum-computing.ibm.com/services?services=systems`

The NISQ era

Important metrics:

- Coherence times
- Gate error rates (single-qubit, two-qubit operations)
- Readout error rates
- Qubit connectivity
- Gate operation time
- More comprehensive benchmarks (e.g., quantum volume)

Huge variability in these metrics depending on the physical implementation. We will cover noisy hardware in weeks $\sim$6 and 10.

The NISQ era

We can still use NISQ devices to do interesting things; but to solve problems "at scale" we need **fault-tolerant quantum computing**. We are starting to see error-corrected qubits in the wild...

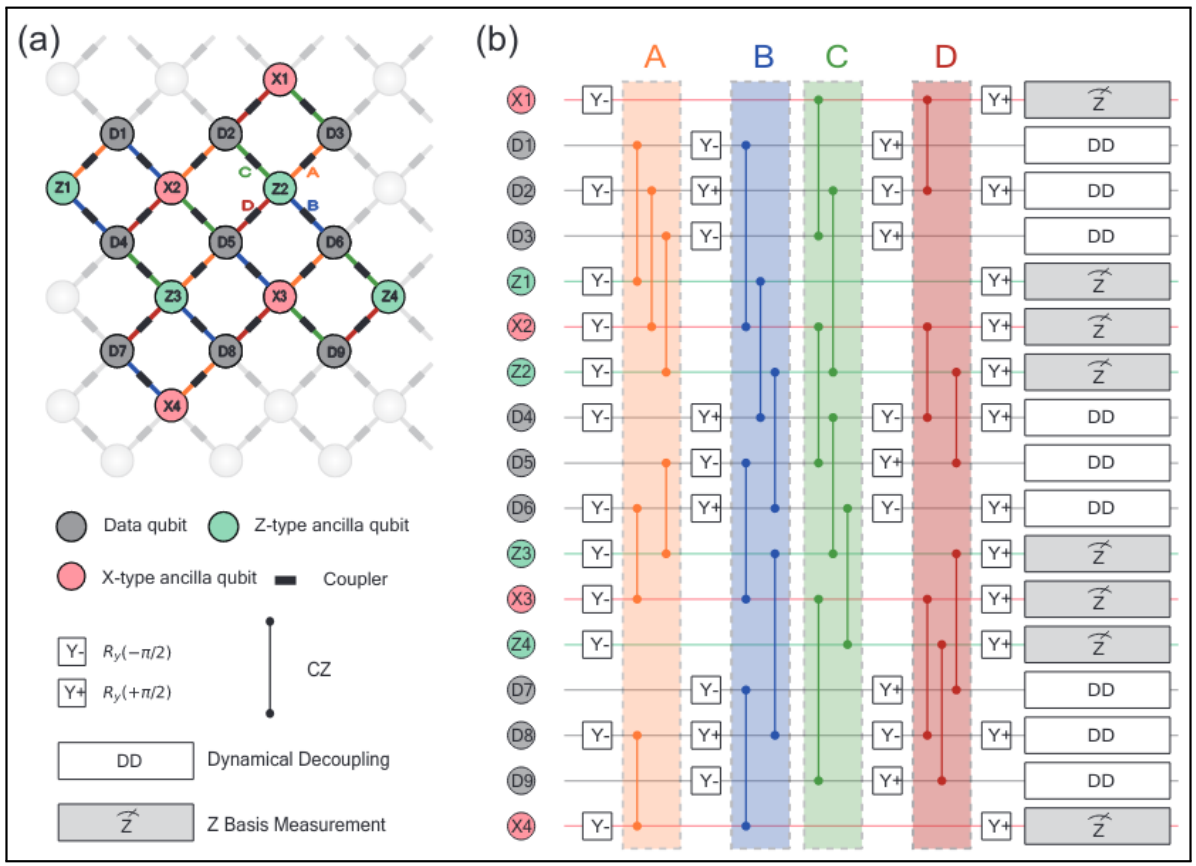


Image: 17-qubit surface code, Zhao et al. `https://arxiv.org/abs/2112.13505v1`

The NISQ era

Some applications will require *millions* of qubits to run an algorithm on noisy qubits with full error correction.

RSA-2048	**Old estimates**				**Current estimates**			
p_g	n_ℓ	n_p	quantum resources	time	n_ℓ	n_p	quantum resources	time
10^{-3}	6190	19.20	1.17	1.46	8194	22.27	0.27	0.34
10^{-5}	6190	9.66	0.34	0.84	8194	8.70	0.06	0.15

Table 2. RSA-2048 security estimates. Here n_ℓ denotes the number of logical qubits, n_p denotes the number of physical qubits (in millions), time denotes the expected time (in hours) to break the scheme, and *quantum resources* (*quantum resources*) are expressed in units of megaqubitdays. The corresponding classical security parameter is 112 bits.

How do we figure out how many? **Quantum resource estimation** (∼week 4).

Image: V. Gheorghiu and M. Mosca, *A resource estimation framework for quantum attacks against cryptographic functions - recent developments.* Feb. 15 2021.

Mathematical representation of qubits

Bits

All computation in our computers today is done with bits.

The state of a bit is *either* 0 or 1.

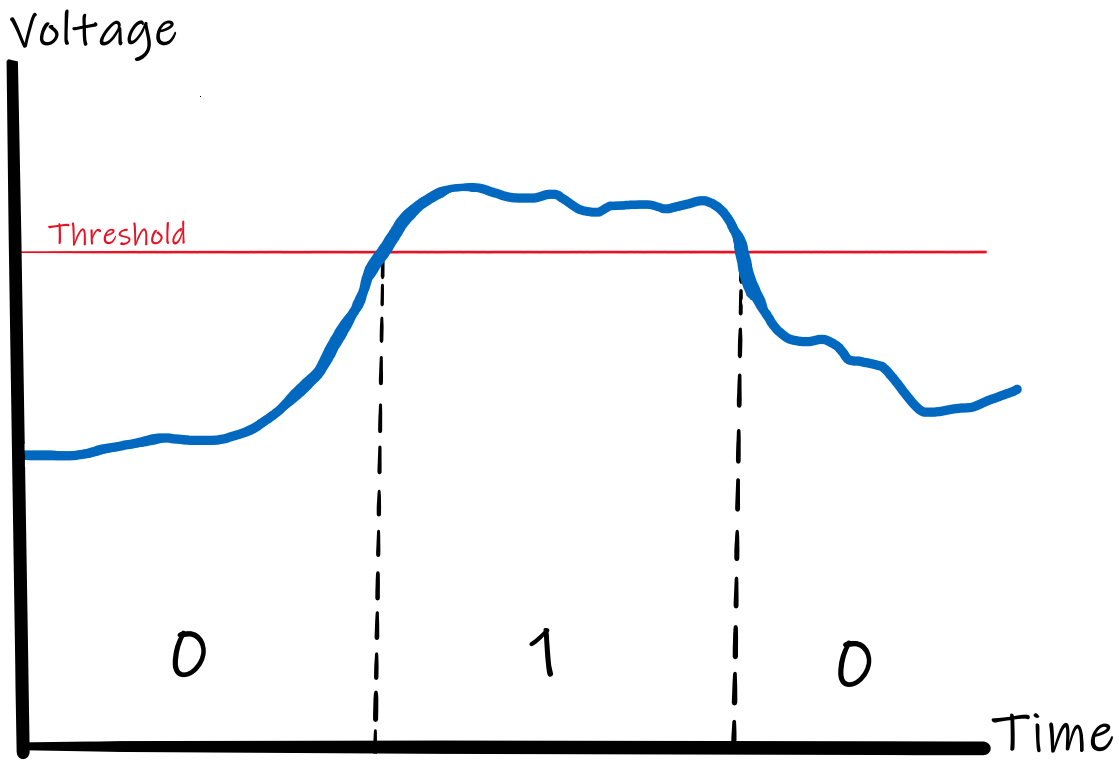
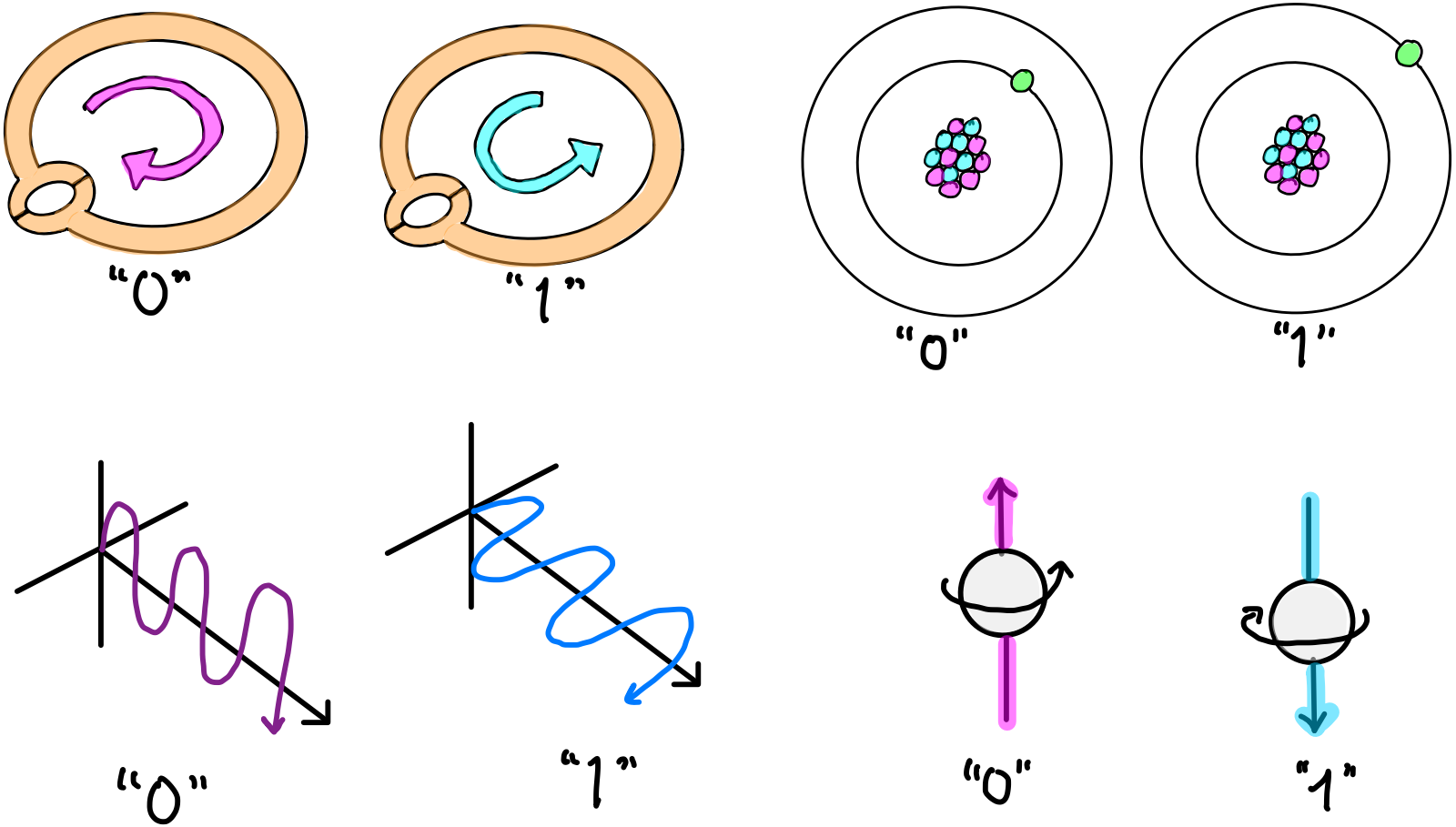


Image credit: Xanadu Quantum Codebook I.1

Qubits

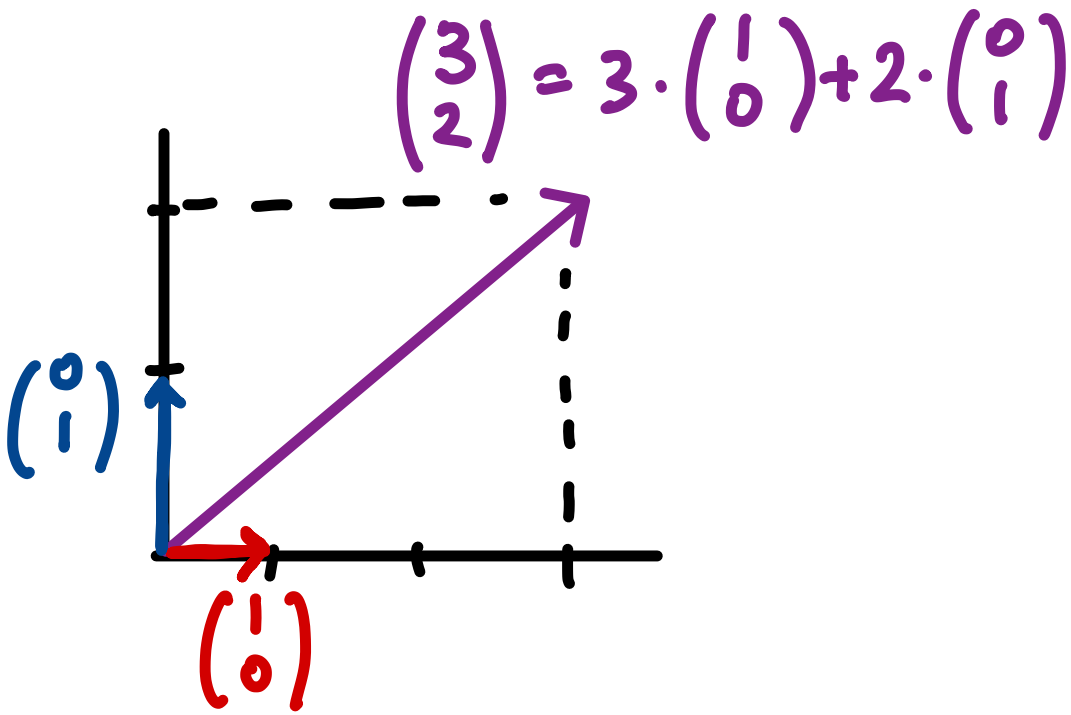
Qubits are *physical quantum systems* described by a state.



Like bits, there are two states labelled 0 and 1.

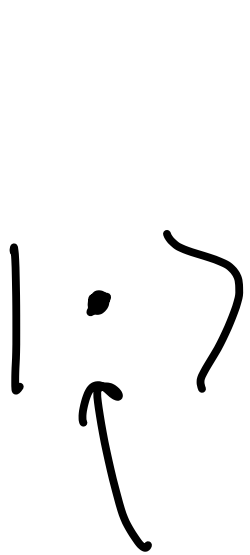
Review: vector spaces

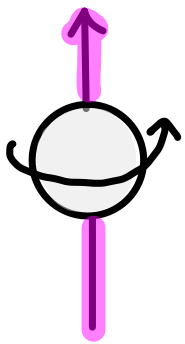
We can represent the state of a qubit mathematically as a vector in a vector space.



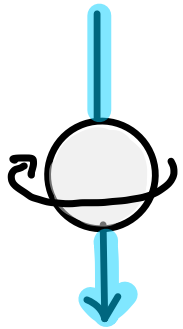
Mathematical representation of qubits

The vector space where qubits live is called **Hilbert space**.





"0"



"1"

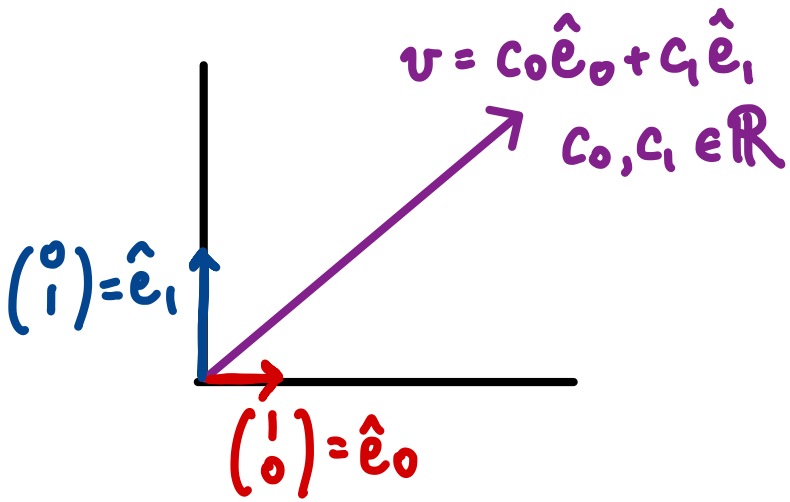
$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

These two vectors together form the **computational basis**.

The notation $|\cdot\rangle$ is called Dirac, or **bra-ket notation**. $|\cdot\rangle$ is a ket.

Mathematical representation of qubits

Qubit states are written as a linear combination of these two basis states. Unlike regular 2D vectors, the coefficients can be *complex*.



$\Rightarrow$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$
$$\alpha, \beta \in \mathbb{C}$$

The coefficients α and β are called **amplitudes**. "Valid" qubit state vectors have unit length:

$$|\alpha|^2 + |\beta|^2 = \alpha\alpha^* + \beta\beta^* = 1$$

(complex conj)

Such states are called **normalized**.

Mathematical representation of qubits

A qubit in a linear combination

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

is in a **superposition** of the two basis states.

Superposition is one of the unique features that makes quantum computing very powerful.

Note: a qubit in superposition is not *"in both states at the same time"*!

What's so good about qubits?

- We can make linear combinations of all possible basis states, i.e., we can put them in **superposition**

What's so good about qubits?

- We can make linear combinations of all possible basis states, i.e., we can put them in **superposition**
- The size of the mathematical space grows **exponentially** in the number of qubits

What's so good about qubits?

- We can make linear combinations of all possible basis states, i.e., we can put them in **superposition**
- The size of the mathematical space grows **exponentially** in the number of qubits
- We can **entangle** multiple qubits, and use these states as a resource in many algorithms

Using qubits for computation

1. Put qubits into superposition
2. Apply operations to qubits and manipulate the amplitudes
3. Measure qubits to extract an answer
4. Profit

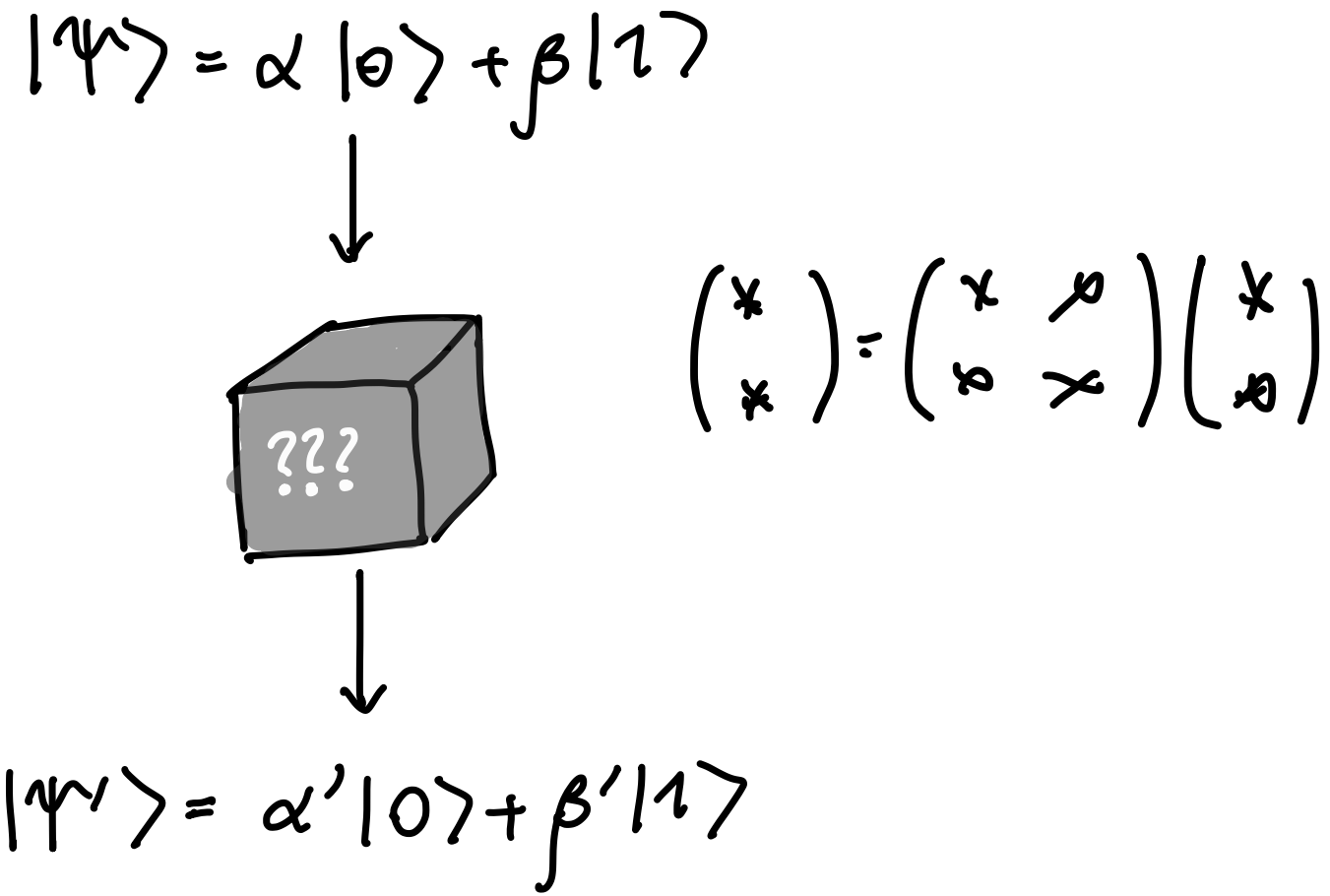
Using qubits for computation

1. Put qubits into superposition
2. Apply operations to qubits and manipulate the amplitudes
3. Measure qubits to extract an answer
4. Profit

...how do we do this?

Manipulating qubits

Manipulating a qubit changes its state:



We need a mechanism for sending valid quantum state vectors to other valid quantum state vectors.

Manipulating qubits: unitary operations

Single-qubit states are manipulated by 2×2 unitary matrices. A matrix U is unitary if

$$U^\dagger U = U U^\dagger = \mathbb{1}.$$

The $\dagger$ means "conjugate transpose":

$$U^\dagger = (U^*)^T.$$

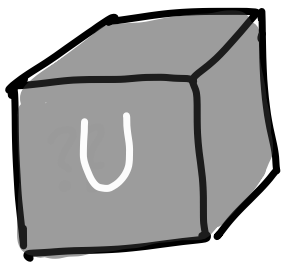
Unitary operations, or **gates**, are **reversible**: if we apply U, we can undo it by applying $U^\dagger$.

Manipulating qubits: unitary operations

Unitary operations preserve the length of vectors, i.e., *maintain the normalization of qubit states*.

$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

$|\alpha|^2 + |\beta|^2 = 1$



$UU^\dagger = U^\dagger U = \mathbb{1}$

$|\psi'\rangle = \alpha'|0\rangle + \beta'|1\rangle$

$= U|\psi\rangle$

$|\alpha'|^2 + |\beta'|^2 = 1$

Manipulating qubits: unitary operations

Unitary operations act *linearly* on superpositions:

$$U(\alpha|0\rangle + \beta|1\rangle) = \alpha U|0\rangle + \beta U|1\rangle$$

This is a key contributor to the power of quantum computing!

Example: X

The matrix

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

is unitary.

What does it do? Check action on basis states...

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

This is the qubit version of a `NOT` gate.

Example: X

If

$$X^\dagger = X$$

$$\begin{aligned} X|0\rangle &= |1\rangle, \\ X|1\rangle &= |0\rangle, \end{aligned}$$

then by linearity,

$$X(\alpha|0\rangle + \beta|1\rangle) = \alpha X|0\rangle + \beta X|1\rangle$$
$$= \alpha|1\rangle + \beta|0\rangle$$

X is also known as Pauli X, or the "bit flip" gate. It is *self-inverse*.

Example: H

Perhaps the most important unitary in quantum computing is the Hadamard gate (H):

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

This gate creates a *uniform superposition*:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) = |+\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) = |-\rangle$$

Example: H

The Hadamard is also self-inverse:

$$HH|0\rangle = H|+\rangle = H\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)$$

$$= \frac{1}{\sqrt{2}} H|0\rangle + \frac{1}{\sqrt{2}} H|1\rangle$$

$$= \frac{1}{\sqrt{2}}\left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] + \frac{1}{\sqrt{2}}\left[\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right]$$

$$= \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|0\rangle - \frac{1}{2}|1\rangle$$

$$= |0\rangle$$

Similarly, $H|-\rangle = |1\rangle$.

Example: Z

The gate

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

does something a little different.

Apply to basis states:

$$Z|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

$$Z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = -|1\rangle$$

Just changes the sign. (This will be important later!)

Single-qubit gates: applying sequences of gates

We apply *products* of single-qubit operations to represent performing gates in sequence. For example,

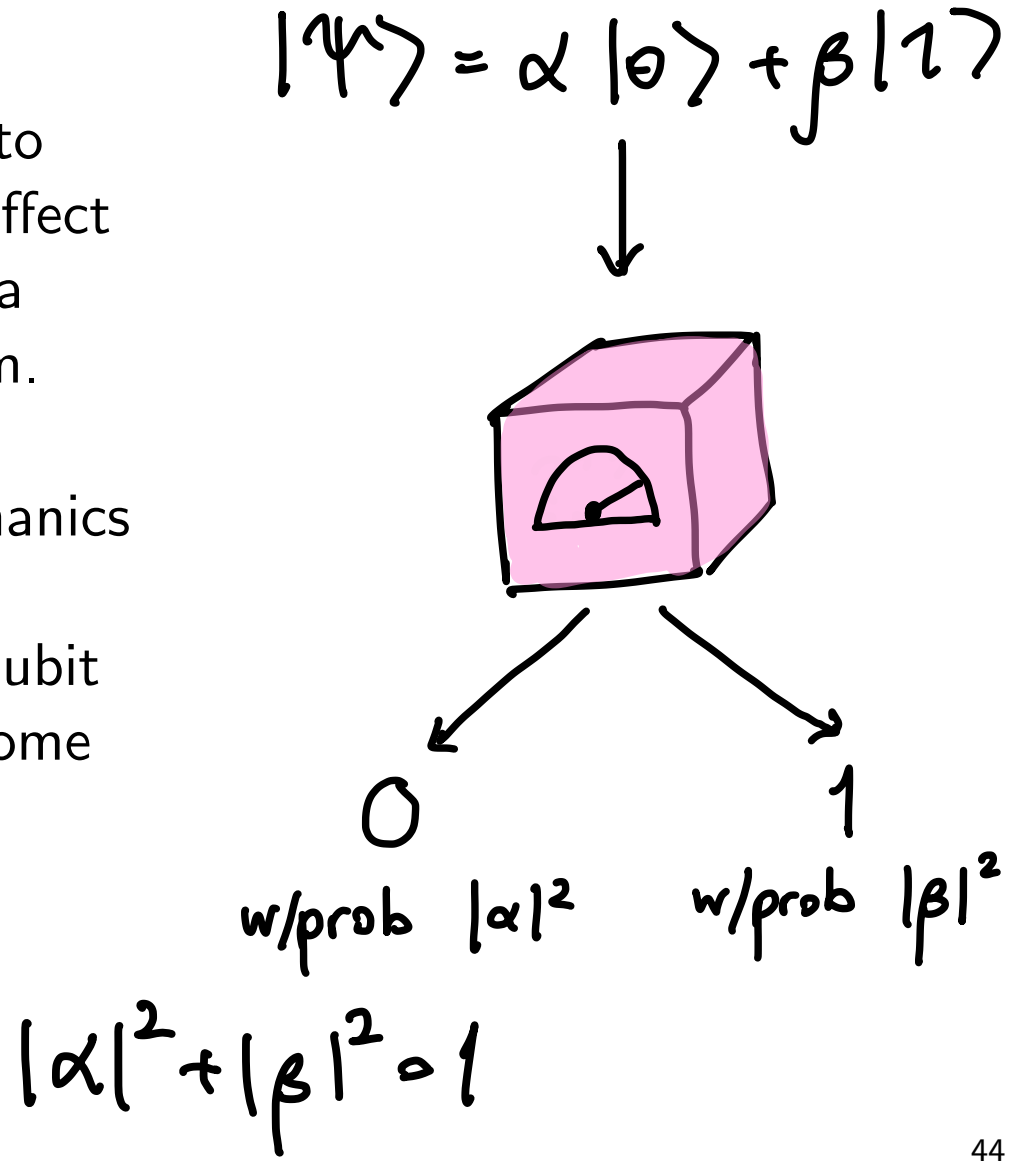
$$X(Z(H|0\rangle)) = X\left(Z\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)\right)$$
$$= X\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)$$
$$= \frac{1}{\sqrt{2}}|1\rangle - \frac{1}{\sqrt{2}}|0\rangle$$

Products are applied from right to left.

Measuring qubits

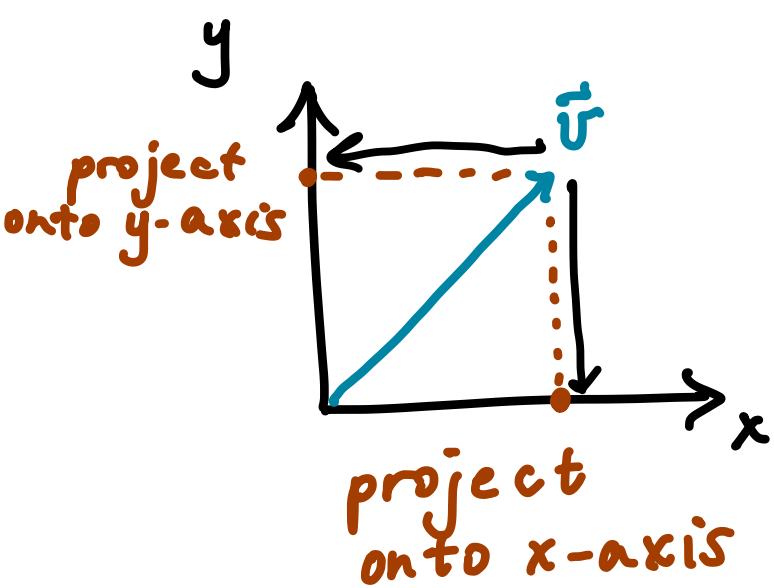
We manipulate qubits in order to change their amplitudes; they affect what we see when we measure a qubit at the end of an algorithm.

Measurement in quantum mechanics (and quantum computing) is **probabilistic**: we observe the qubit in one of its basis states with some probability.



Projective measurements

Measurement is performed with respect to a basis; we compute **projections** of qubit states onto the basis to determine the overlap with a given basis state.



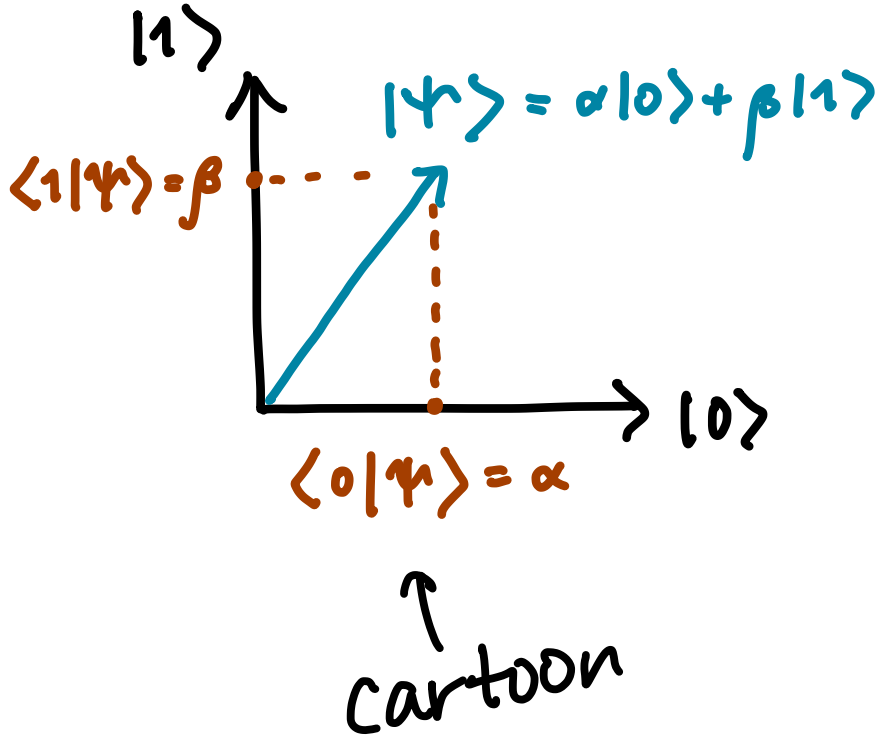
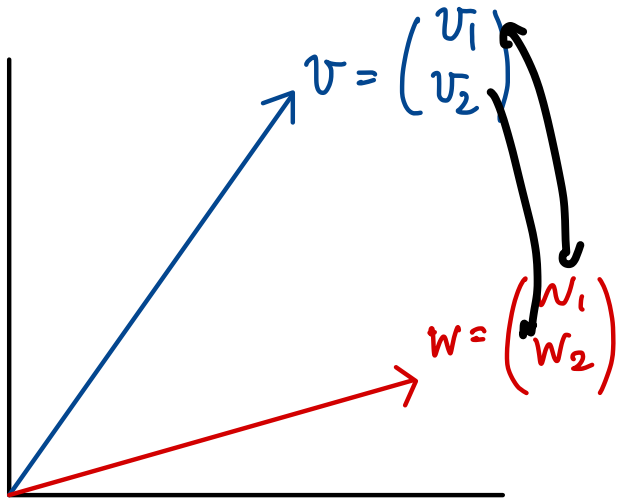


Image credit: Xanadu Quantum Codebook I.9

Inner products

The amount of overlap between two states can be computed by taking their inner product:

$|\cdot\rangle = \begin{pmatrix} x \\ x \end{pmatrix}$ $\langle\cdot| = (\quad)$



$$|\psi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} = \psi_1|0\rangle + \psi_2|1\rangle$$

$$|\varphi\rangle = \begin{pmatrix} \varphi_1 \\ \varphi_2 \end{pmatrix} = \varphi_1|0\rangle + \varphi_2|1\rangle$$

$$\langle v, w\rangle = v \cdot w = (v_1 \; v_2)\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \sum_{i=1}^{2} v_i w_i = v_1 w_1 + v_2 w_2$$

$$\langle \psi, \varphi\rangle = \langle\psi||\varphi\rangle = \langle\psi|\varphi\rangle = (\psi_1^* \; \psi_2^*)\begin{pmatrix} \varphi_1 \\ \varphi_2 \end{pmatrix} = \sum_{i=1}^{2} \psi_i^* \varphi_i = \psi_1^* \varphi_1 + \psi_2^* \varphi_2$$

Inner products

Special cases:

1. $\langle\varphi|\psi\rangle = 0$: states are orthogonal
2. $\langle\varphi|\psi\rangle = 1$: $|\varphi\rangle = |\psi\rangle$

The second case also tells us a state is normalized. If $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$,

$$\langle\psi|\psi\rangle = \alpha^*\alpha + \beta^*\beta = |\alpha|^2 + |\beta|^2 = 1.$$

If two states are both normalized, and orthogonal, they constitute an **orthonormal basis**. We can write any qubit state in terms of those basis states, and performa measurement with respect to it.

Projective measurements

When we measure state $|\varphi\rangle$ with respect to basis $\{|\psi_i\rangle\}$, the probability of obtaining outcome i is

$$\Pr(\text{outcome } i) = |\langle\psi_i|\varphi\rangle|^2$$

If we observe outcome i, following the measurement the system will be left in state $|\psi_i\rangle$.

Example: measurement in computational basis

$$\langle 0|\psi\rangle = \alpha\langle 0|0\rangle + \beta\langle 0|1\rangle$$

$\langle 0|0\rangle = 1$, $\langle 0|1\rangle = 0$

$$= \alpha$$

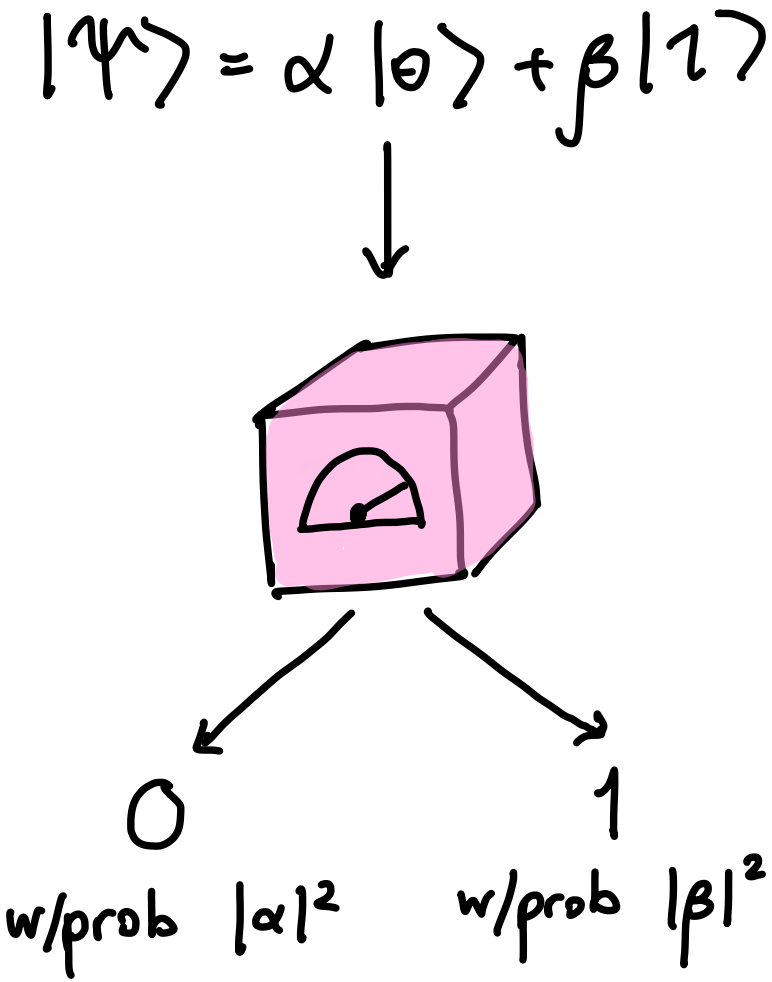
Let $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$.

Then if we measure $|\psi\rangle$ is the computational basis,

$$\Pr(0) = |\langle 0|\psi\rangle|^2 = |\alpha|^2$$

$$\Pr(1) = |\langle 1|\psi\rangle|^2 = |\beta|^2$$

Measurement



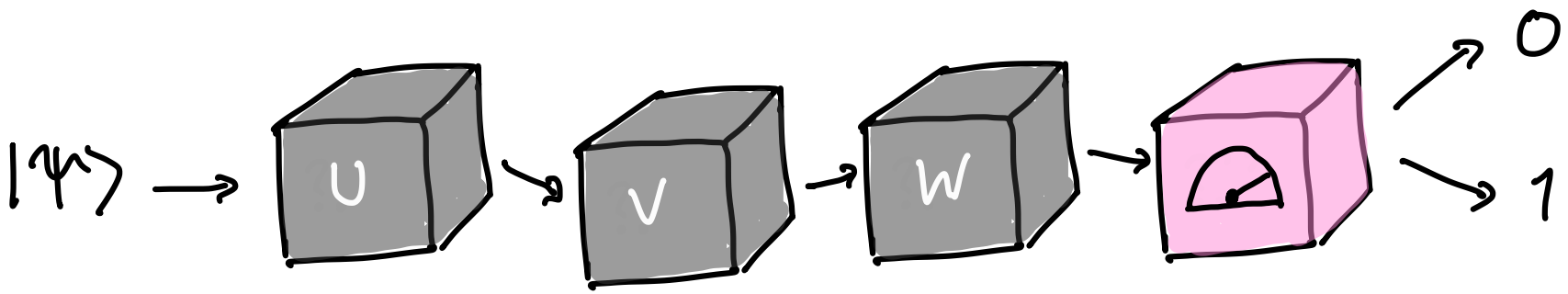
Running an algorithm and measuring gives us only a single bit of information (0 or 1).

In order to get more information, we need to run the algorithm multiple times (multiple **shots**).

Quantum computing

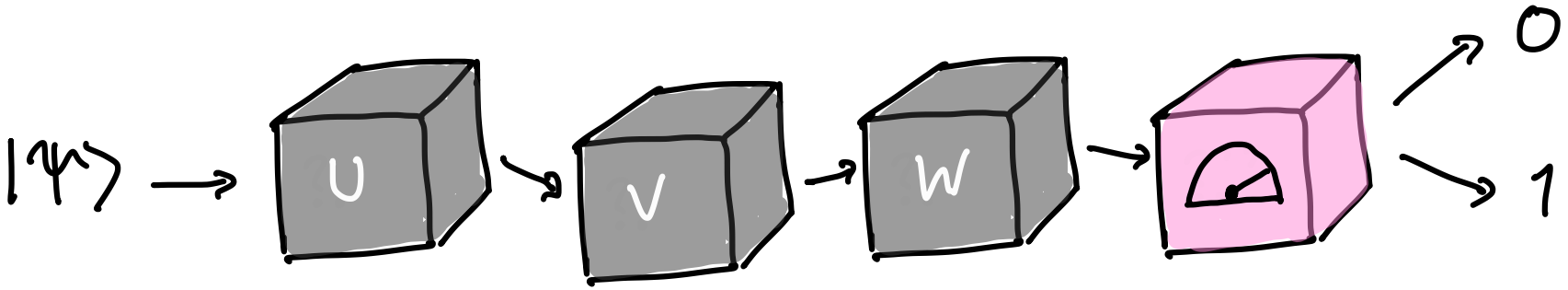
Quantum computing is the act of manipulating the state of qubits in a way that represents solution of a computational problem:

1. Put qubits into superposition
2. Apply operations to qubits and manipulate the amplitudes
3. Measure qubits to extract an answer
4. Profit

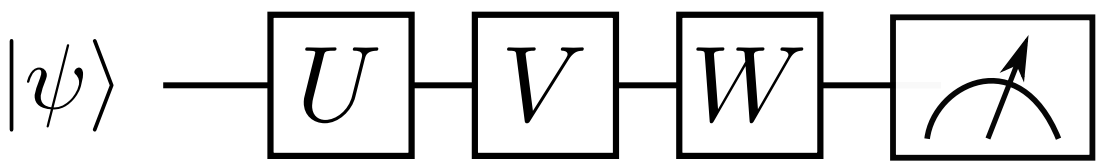


Quantum circuits

This is cute, but impractical:



Instead, we use **quantum circuits**.



Hands-on: manipulating qubits in NumPy

PennyLane

That was tedious—let's never do that again.

Let's use some real quantum software instead:

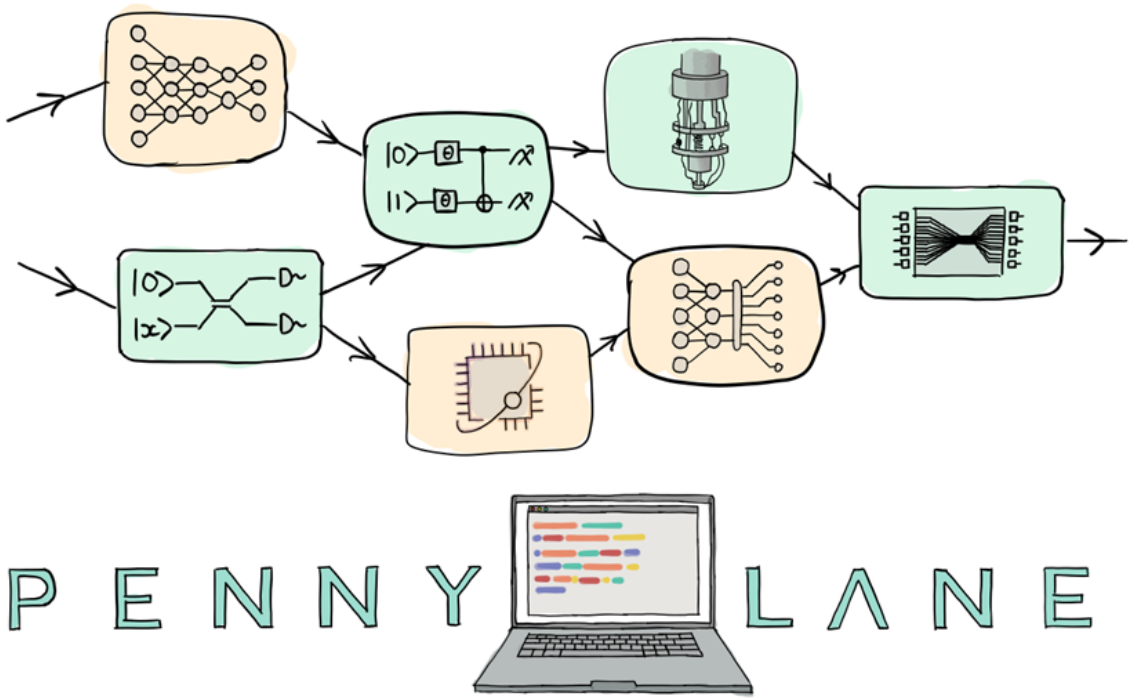


Image credit: `https://pennylane.ai/`

PennyLane

PennyLane is an open source Python framework developed by **Xanadu** (a Toronto-based quantum startup).

GitHub: `https://github.com/PennyLaneAI/PennyLane`
Documentation: `https://pennylane.readthedocs.io/en/stable/`
Demonstrations: `https://pennylane.ai/qml/demonstrations.html`
Discussion Forum: `https://discuss.pennylane.ai/`

Its key use case is **differentiable quantum programming** and quantum machine learning; it is also a valuable tool for quantum computing algorithms and applications.

PennyLane

```python
def ket_0():
    return np.array([1.0, 0.0])

def apply_ops(ops, state):
    for op in ops:
        state = np.dot(op, state)
    return state

def measure(state, num_samples=100):
    prob_0 = state[0] * state[0].conj()
    prob_1 = state[1] * state[1].conj()

    samples = np.random.choice(
        [0, 1], size=num_samples, p=[prob_0, prob_1]
    )
    return samples

H = (1/np.sqrt(2)) * np.array([[1, 1], [1, -1]])
X = np.array([[0, 1], [1, 0]])
Z = np.array([[1, 0], [0, -1]])

input_state = ket_0()
output_state = apply_ops([H, X, Z], input_state)
results = measure(output_state, num_samples=10)

print(results)
```

```
[1 0 0 1 0 0 0 1 1 0]
```

Sample NumPy

```python
dev = qml.device('default.qubit', wires=1, shots=10)

@qml.qnode(dev)
def my_circuit():
    qml.Hadamard(wires=0)
    qml.PauliX(wires=0)
    qml.PauliZ(wires=0)
    return qml.sample()

print(my_circuit())
```

```
[1 0 1 0 1 1 0 1 0 0]
```

PennyLane

Recap

Today's learning outcomes were:

1. Define quantum computing, and explain its societal importance
2. Define a *qubit* both conceptually and mathematically
3. Simulate quantum computing on a single-qubit

What topics did you find unclear today?

For next time

Content:

- More unitary operations
- Types of quantum measurements
- Getting started with PennyLane

Action items:

1. Follow instructions on Canvas to sign up for a UBC GitHub account
2. Fork the Assignment 0 repo and start working on it (due next Tuesday)
3. Install and configure your programming environment prior to next class (come to office hours if you need help)

Recommended reading: Codebook nodes I.1-I.5.