CPEN 416 / EECE 571O

Lecture 02: Single-qubit systems; introducing PennyLane

Tuesday 09 September 2025

Announcements

- Practice assignment 1 available later today
- Quiz 1 today

Recap from last time

Qubits are physical quantum systems with two **basis states**:

Arbitrary states are complex-valued linear combinations

where $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha, \beta \in \mathbb{C}$.

Qubit states live in **Hilbert space**.

Recap from last time

Unitary matrices (gates/operations) modify a qubit's state.

A matrix U is unitary if

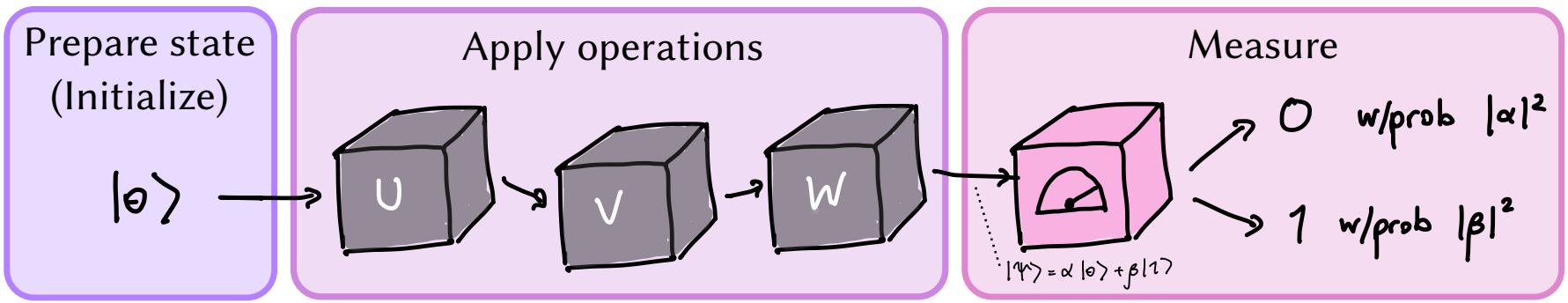
They preserve normalization of state vectors (and angles between them - can prove on practice assignment 1).

We saw three examples:

Recap from last time

After applying gates, we measure and observe the qubit in either state $|0\rangle$ or $|1\rangle$.

Measurement is probabilistic: the outcome probabilities depend on the amplitudes.

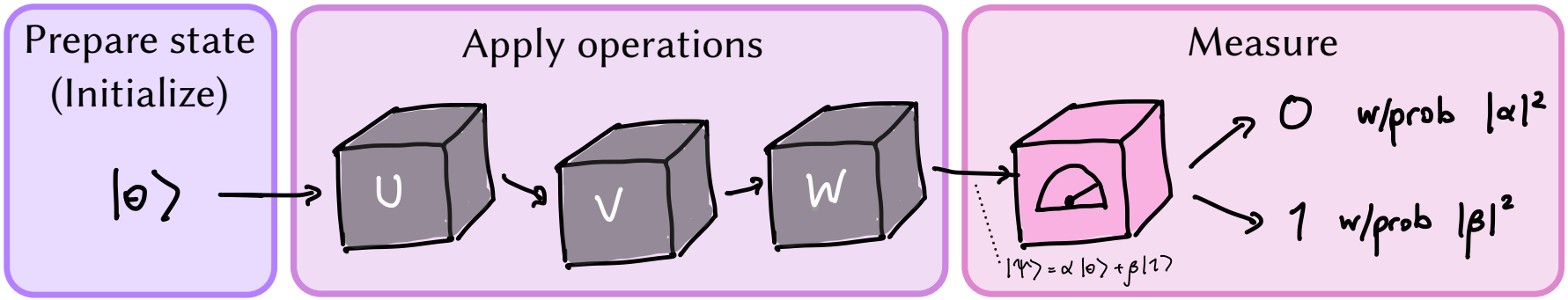


Learning outcomes

1. Express quantum computations as quantum circuits, and as quantum functions in PennyLane
2. Represent the state of a single qubit on the Bloch sphere
3. Describe the behaviour of common single-qubit gates

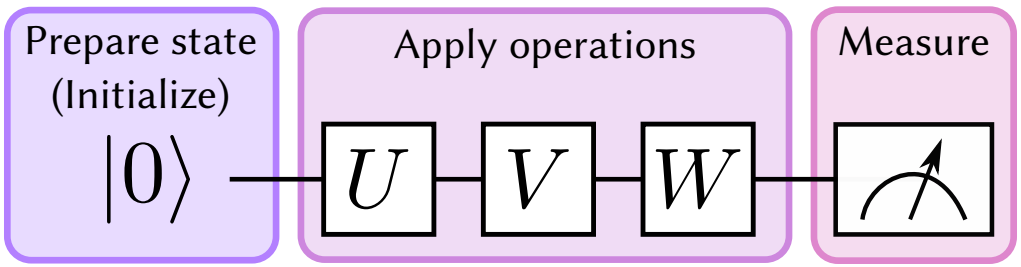
Quantum circuits

This representation is cumbersome:



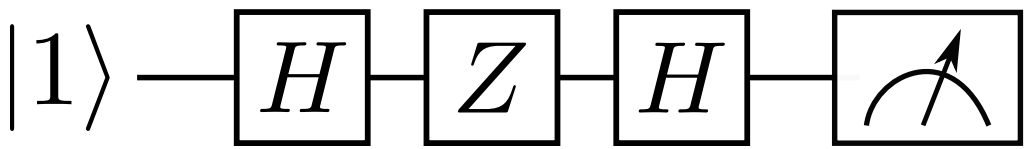
So is the matrix representation:

Alternative: quantum circuits!



Quantum circuits

Exercise: perform the quantum computation in the circuit below. What is the probability of observing the qubit in state $|0\rangle$ after measurement?



Programming quantum computers

Everything we've done so far is just matrix-vector multiplication:

```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

... better to use real quantum software instead.

PennyLane

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

```python
import pennylane as qml

H = qml.Hamiltonian(...)

dev = qml.device('default.qubit', wires=2)

@qml.qnode(dev)
def quantum_circuit(params):
    qml.RY(params[0], wires=0)
    qml.RY(params[1], wires=1)
    qml.CNOT(wires=[0, 1])
    qml.RY(params[2], wires=0)
    return qml.expval(H)

quantum_circuit([0.1, 0.2, 0.3])
```

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/`

Quantum functions

We can express circuits as **quantum functions** in PennyLane. They are similar to normal Python functions, except they

1. apply one or more quantum operations to qubits (0-indexed)
2. must return a quantum measurement

```
import pennylane as qml

def my_quantum_function():
    qml.Hadamard(wires=0) # Apply Hadamard gate to qubit 0
    qml.PauliZ(wires=0)   # Apply Pauli Z gate to qubit 0
    qml.PauliX(wires=0)   # Apply Pauli X gate to qubit 0
    return qml.probs()    # Return measurement probs
```

Q: Why `wires`? A: Lines in a circuit diagram are called wires, and PennyLane can be used for continuous-variable quantum computing, which isn't qubit-based. (Note: keyword argument no longer needed.)

Devices

Quantum functions are executed on **devices**.

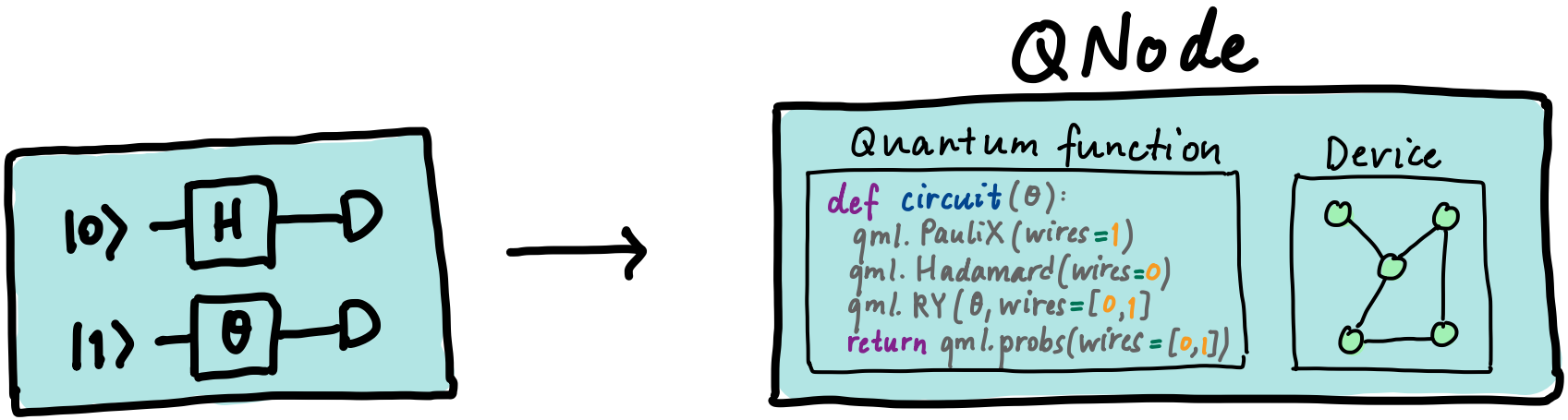
```
# Example 1: 5 qubits, analytic
dev = qml.device("default.qubit", wires=5)

# Example 2: 1 qubit, probabilistic (100 measurements)
dev = qml.device("default.qubit", wires=1, shots=100)
```

Devices can be either *simulators*, or *actual quantum hardware*.

Qubits on a device are initialized in $|0\rangle$.

Quantum nodes

A **QNode (quantum node)** is an object that binds a quantum function to a device, and executes it.

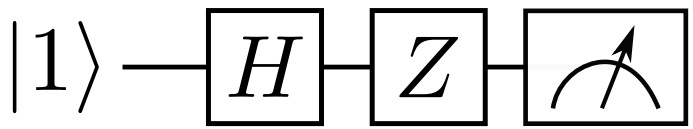



```
# Create a QNode (version 1)
my_qnode = qml.QNode(my_quantum_function, dev)

# Execute the QNode
result = my_qnode()
```

Image: https://pennylane.ai/qml/glossary/quantum_node.html

Executing quantum computations with PennyLane

Exercise: perform the quantum computation below in PennyLane on a device with 1000 shots. What is the probability of observing the qubit in state $|0\rangle$ after measurement?




```python
import pennylane as qml

dev = qml.device("default.qubit", wires=?, shots=?)

def my_quantum_function():
    # Your gates here
    return qml.probs()

my_qnode = qml.QNode(my_quantum_function, dev)
my_qnode()
```

Run locally, or try our interactive debugger, CircInspect: `circinspect.ece.ubc.ca`

You probably have some questions...

1. Where's the state?
   - Inside the device
2. How do gates get applied?
   - Operations are recorded onto a "tape"
   - The QNode constructs the tape when it is called
   - The tape is then executed on the device.

Single-qubit quantum states

So far we know the following 3 gates:

But, a general qubit state looks like

What about the rest?

1. First, we'll explore a way to visualize arbitrary quantum states
2. Next we'll learn new unitary operations to create them with

Parametrization of qubit states

Exercise (part A): Consider the states

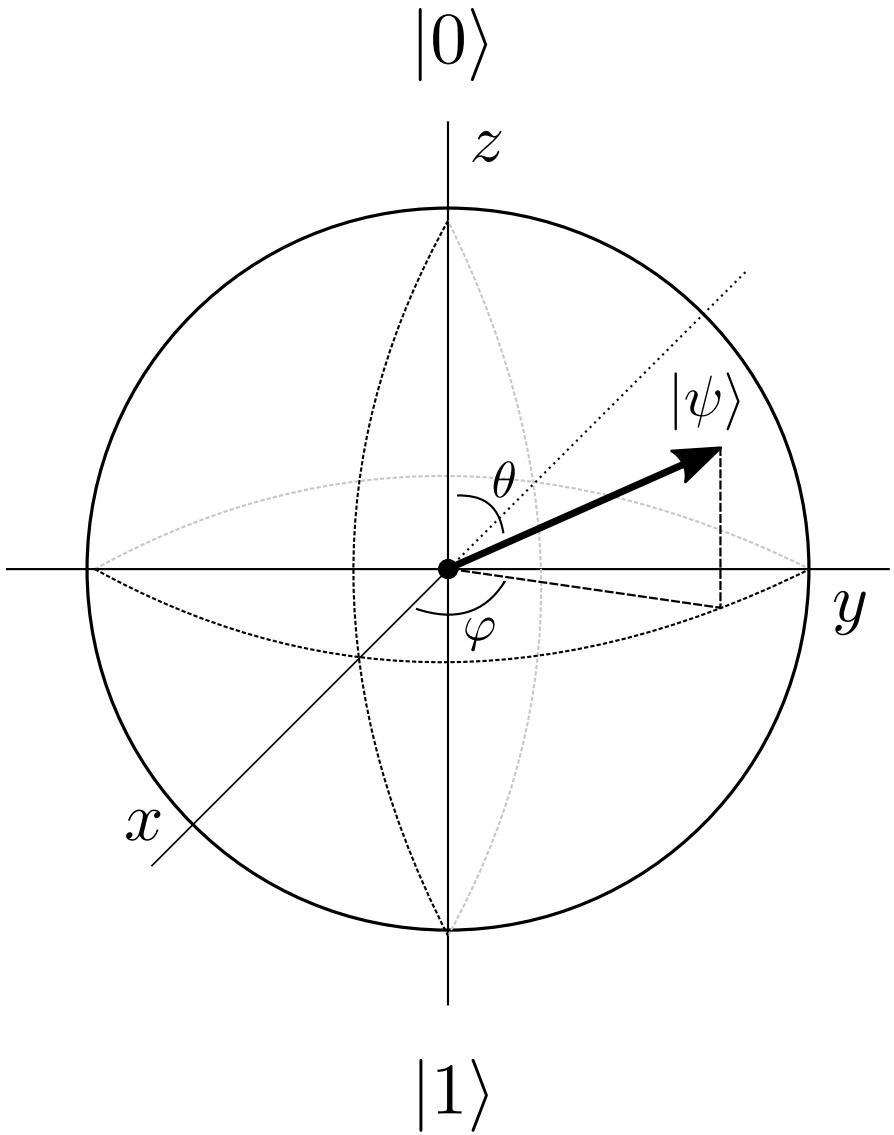
$$|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\psi_2\rangle = \alpha e^{i\phi}|0\rangle + \beta e^{i\phi}|1\rangle$$

How does $e^{i\phi}$ affect the measurement outcome probabilities of $|\psi_2\rangle$ compared to $|\psi_1\rangle$?

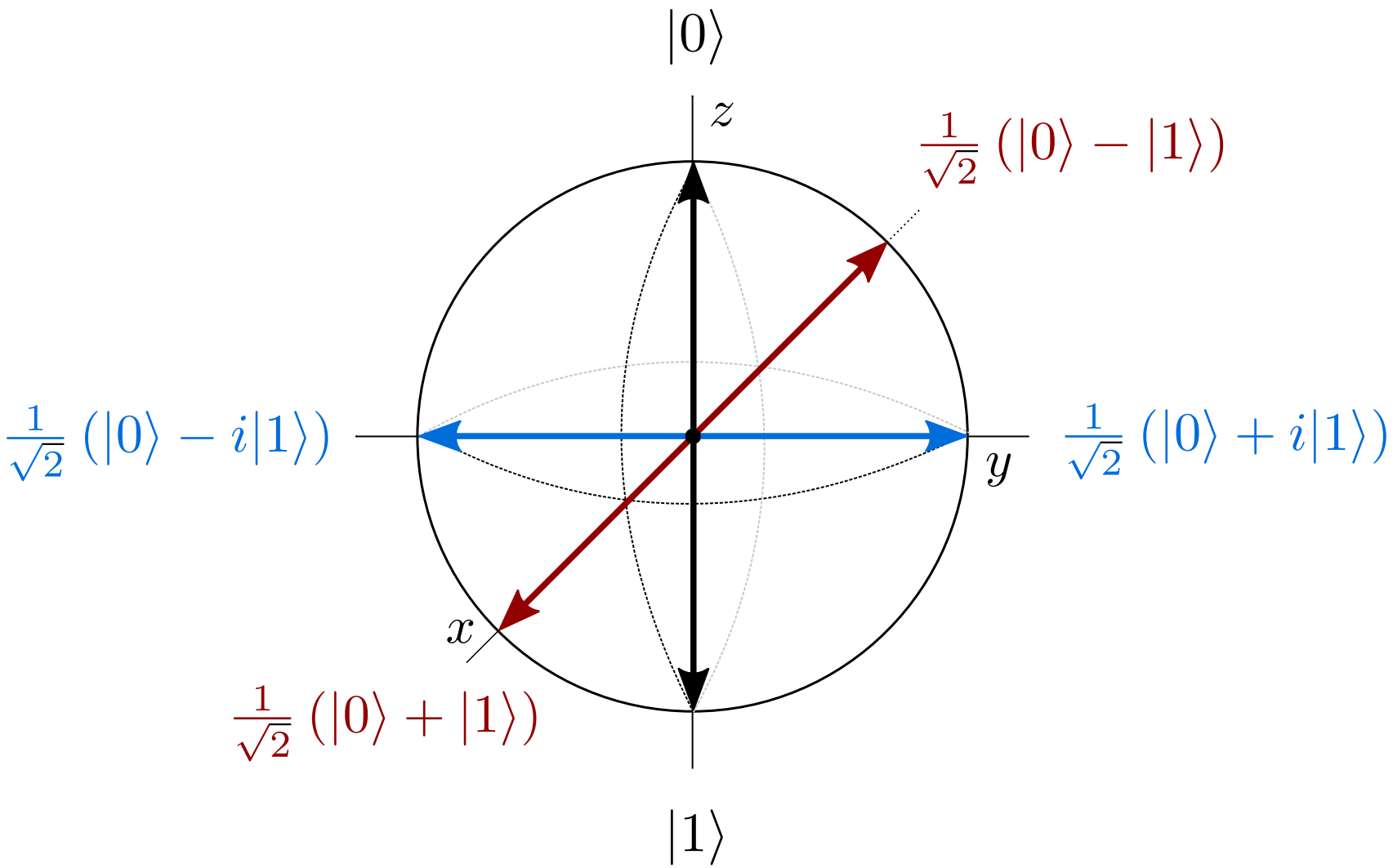
Parametrization of qubit states

Exercise (part B): How many real numbers are required to *fully specify* a single-qubit state vector?

The Bloch sphere

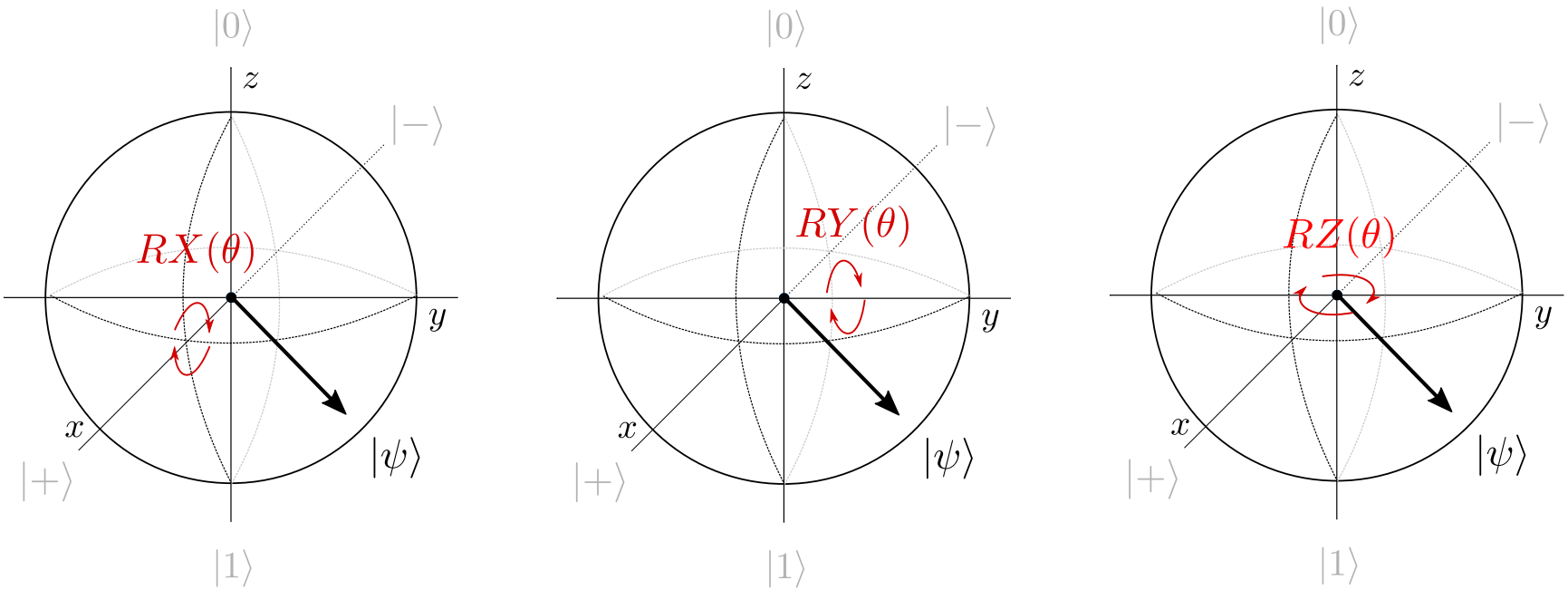


The Bloch sphere



Rotations: the Bloch sphere

Unitary operations rotate the state vector on the Bloch sphere.

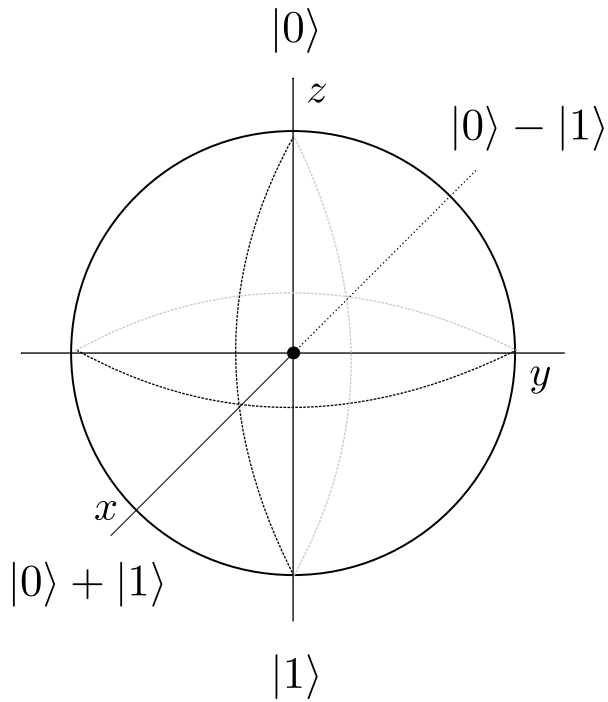


`https://bloch.kherb.io/`

Image credit: Codebook node I.6

Z rotations (RZ)

$$RZ(\phi) = e^{-i\frac{\phi}{2}Z} = \begin{pmatrix} e^{-i\frac{\phi}{2}} & 0 \\ 0 & e^{i\frac{\phi}{2}} \end{pmatrix}$$



In PennyLane:

```
qml.RZ(phi, wires=wire)
```

Try at home: directly expand $e^{-i\frac{\phi}{2}Z}$ to obtain the matrix representation.

Z rotations (RZ)

$$RZ(\phi) = e^{-i\frac{\phi}{2}Z} = \begin{pmatrix} e^{-i\frac{\phi}{2}} & 0 \\ 0 & e^{i\frac{\phi}{2}} \end{pmatrix}$$

Apply to a general state:

S and T

Two other special cases: $\phi = \pi/2$, and $\phi = \pi/4$.

$$S = RZ(\pi/2) = \begin{pmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

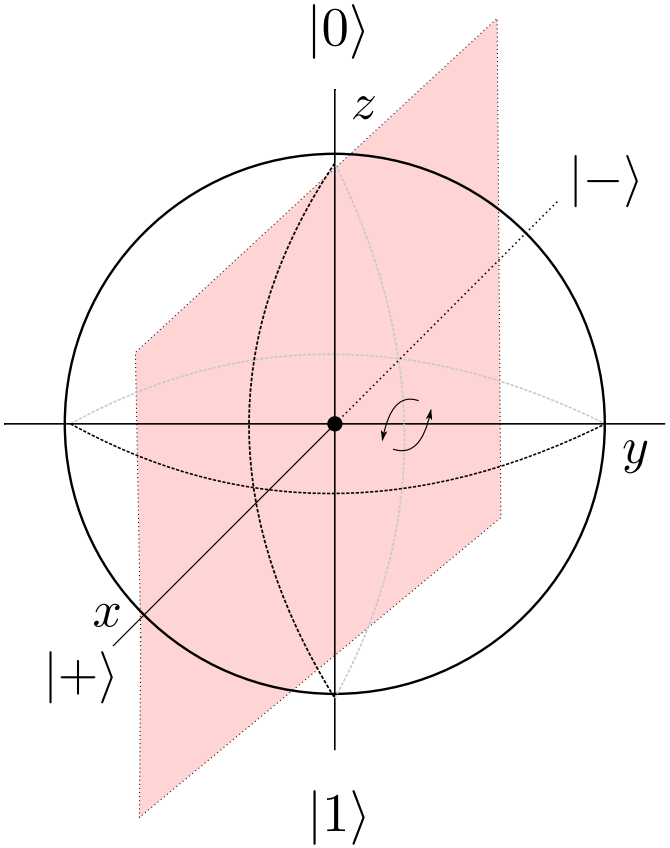
$$T = RZ(\pi/4) = \begin{pmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$$

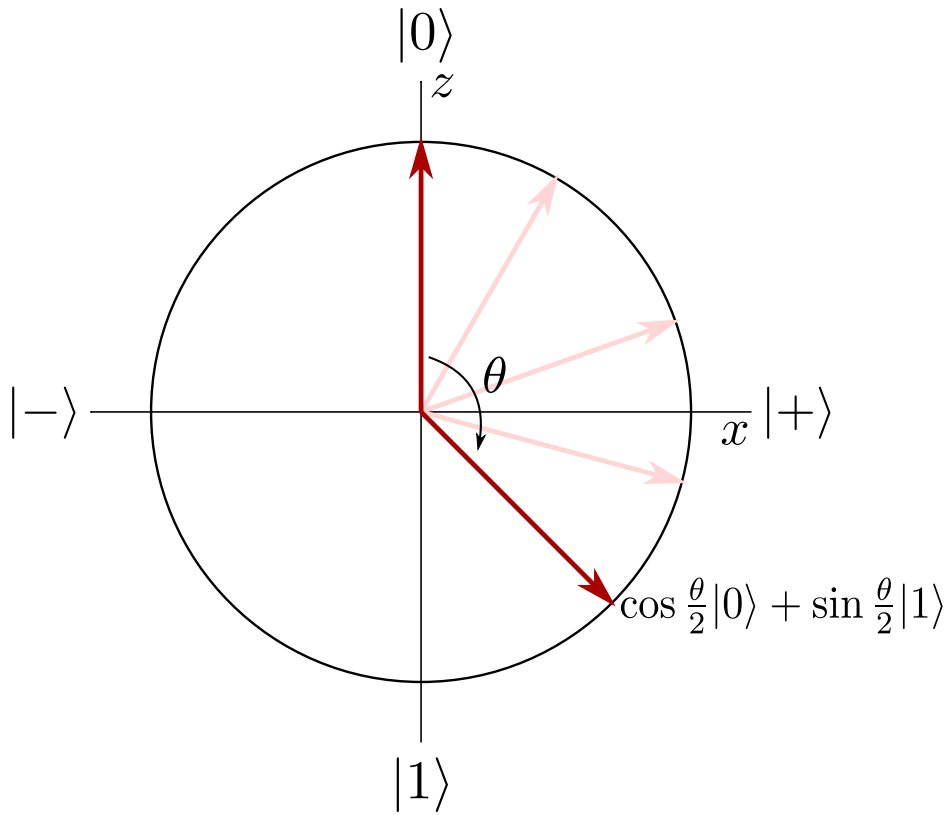
In PennyLane:

```
qml.PauliZ(wires=wire)
qml.S(wires=wire)
qml.T(wires=wire)
```

S is part of a special group called the **Clifford group**.
T is used in universal gate sets for fault-tolerant QC.

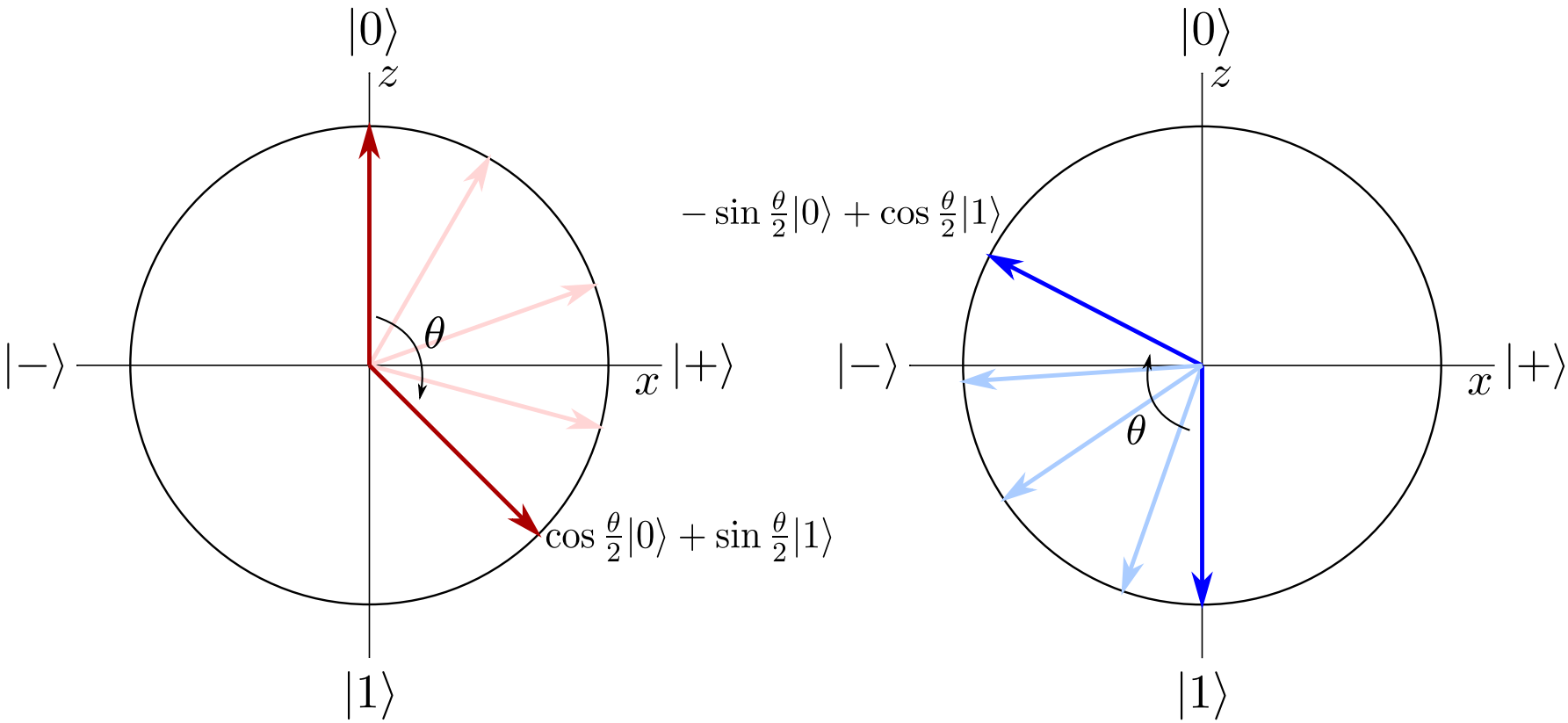
Y rotations (RY)





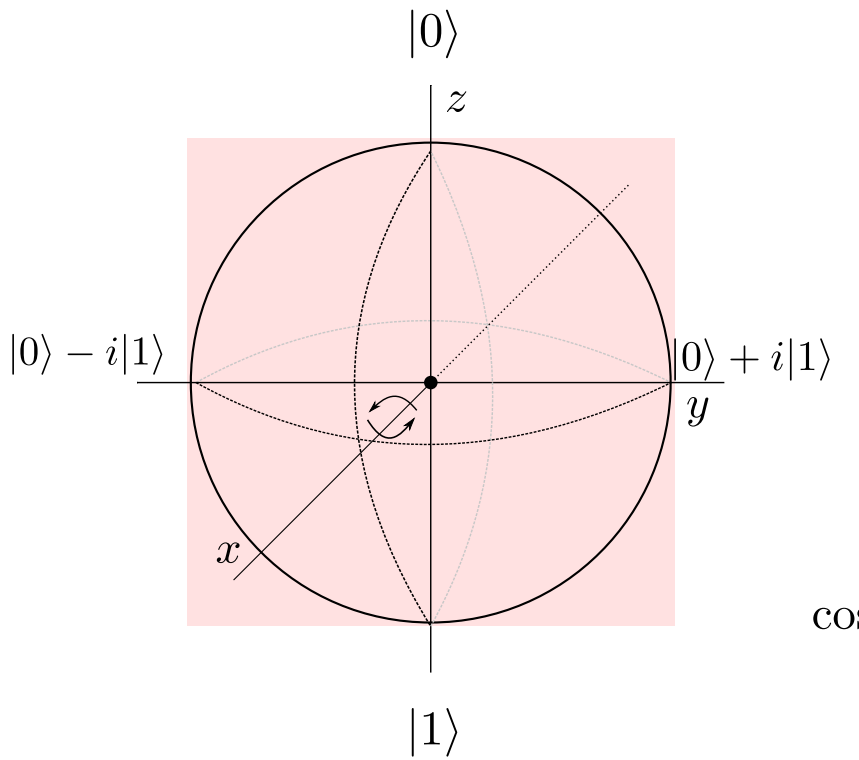
Y rotations (RY)

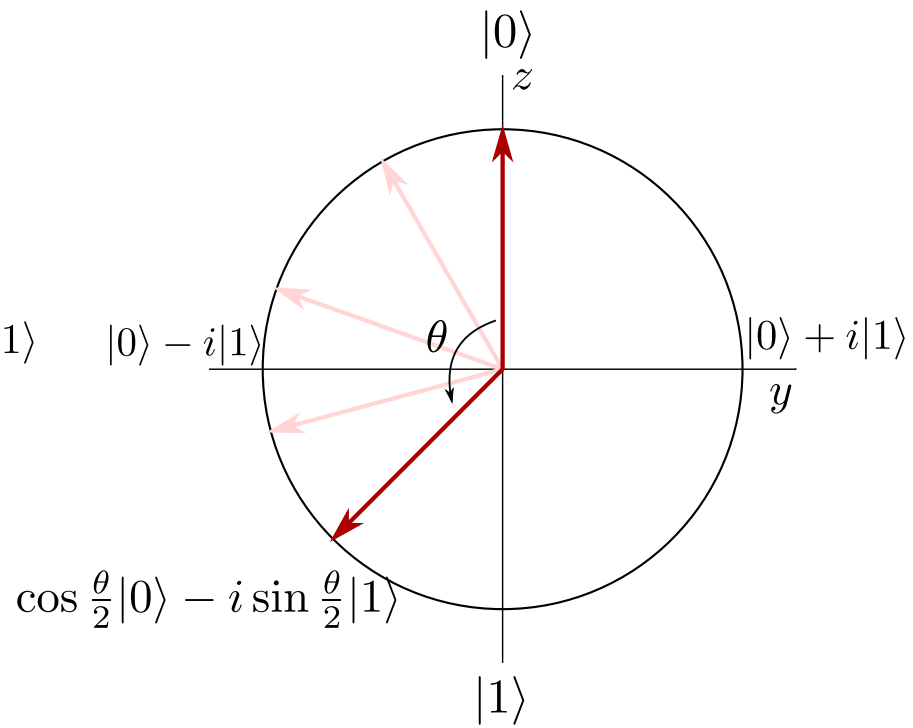
$$RY(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$



X rotations (RX)

$$RX(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$





Pauli rotations

These unitary operations are called **Pauli rotations**.

	Math	Matrix	Code	Special cases (θ)
RZ	$e^{-i\frac{\theta}{2}Z}$	$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$	`qml.RZ`	$Z(\pi), S(\pi/2), T(\pi/4)$
RY	$e^{-i\frac{\theta}{2}Y}$	$\begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$	`qml.RY`	$Y(\pi)$
RX	$e^{-i\frac{\theta}{2}X}$	$\begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$	`qml.RX`	$X(\pi), SX(\pi/2)$

Pauli rotations

Exercise: design a quantum circuit that prepares

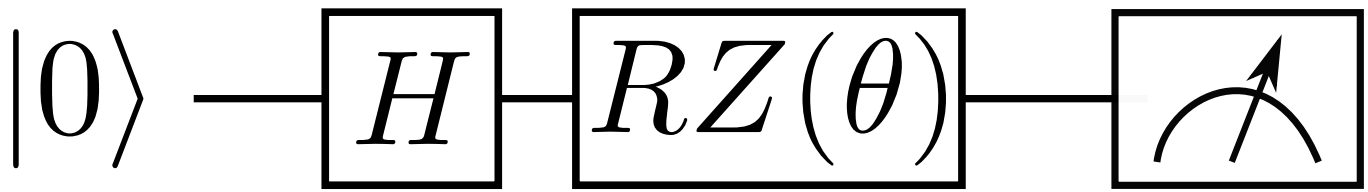
$$|\psi\rangle = \frac{\sqrt{3}}{2}|0\rangle - \frac{1}{2}e^{i\frac{5\pi}{4}}|1\rangle$$

Hint: you can return the state directly in PennyLane:

```
@qml.qnode(dev)
def some_circuit():
    # Gates...
    return qml.state()
```

Pauli rotations

Exercise: implement the circuit below in PennyLane.



Run your circuit with two different values of θ and take 1000 shots.

How does θ affect the measurement outcome probabilities?

Next time

Content:

- The theory of projective measurements
- Measuring in different bases

Action items:

1. Work on problems in practice Assignments 0 and 1
2. Explore the PennyLane docs and learn about the different kinds of operations and measurements
3. Visualize combinations of operations on the Bloch sphere

Recommended reading:

- Codebook modules IQC, SQ
- Nielsen & Chuang 4.2, 2.2.3, 2.2.5, 2.2.7