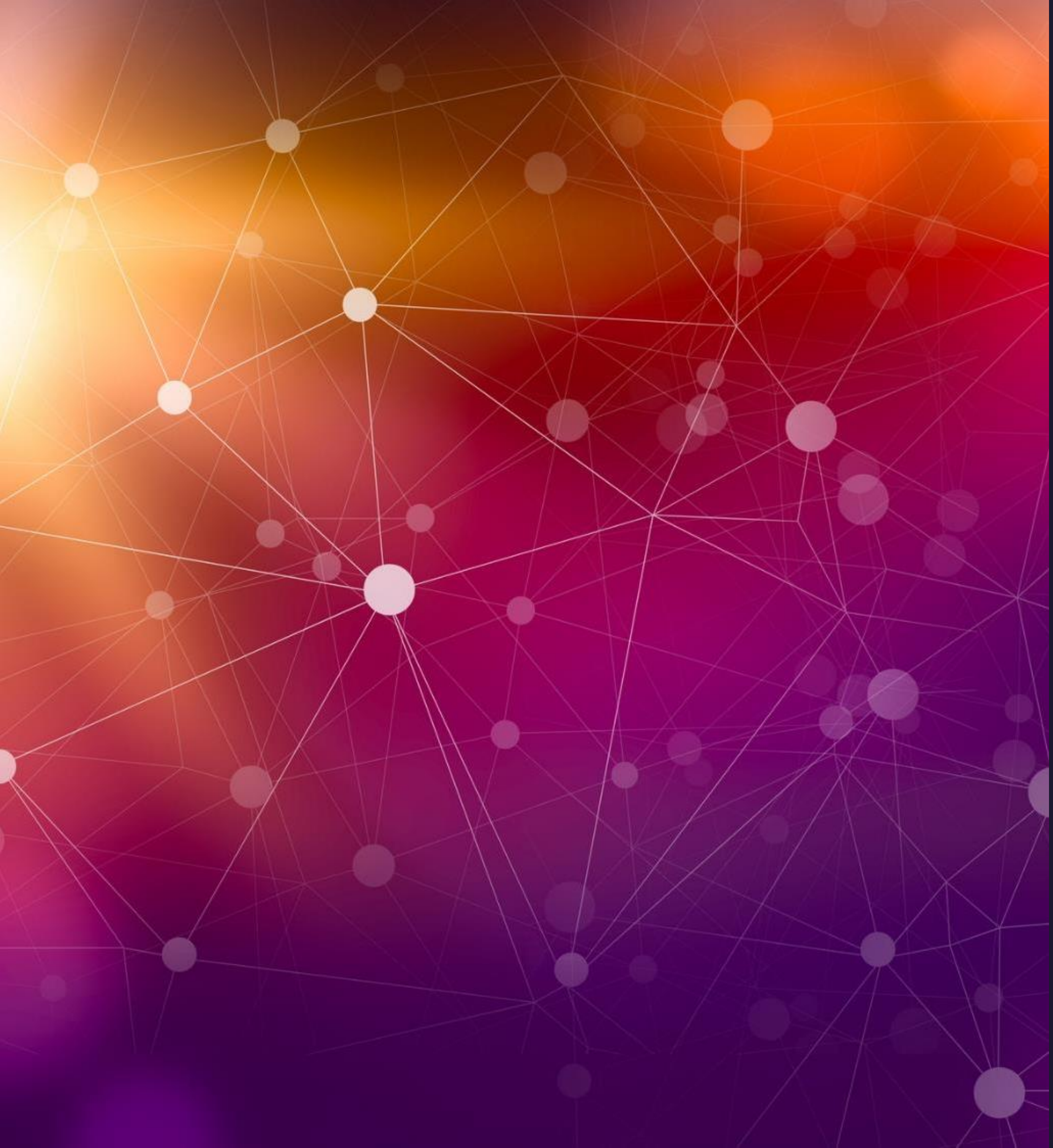


Qantum Computing Algorithms

Agenda

Qubits

Logic Gates

Quantum Properties

Quantum Algorithms

Challenges of Qubits

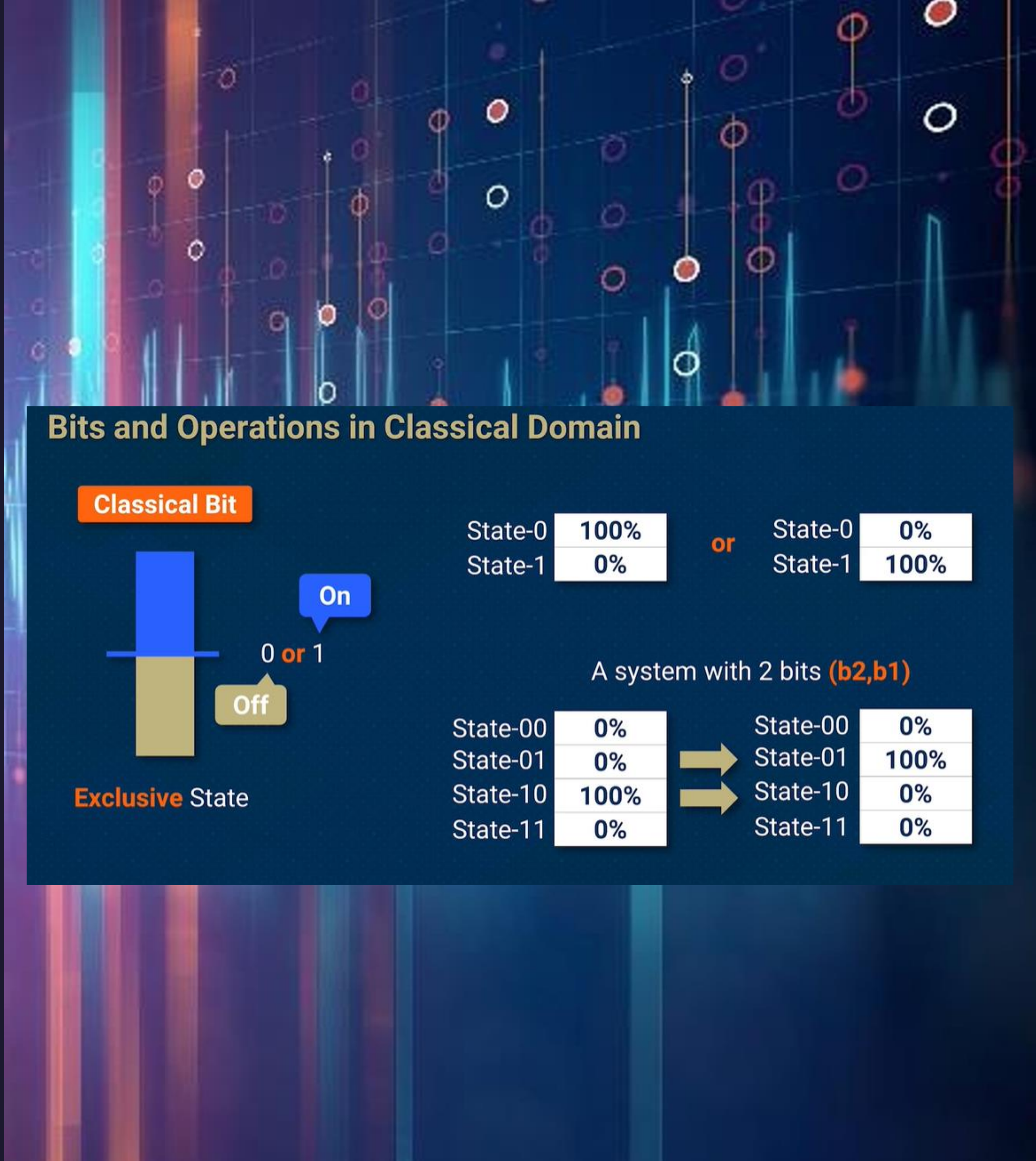
Noisy Intermediate Scale Quantum Computers (NISQ)

Applications

Final tips & takeaways

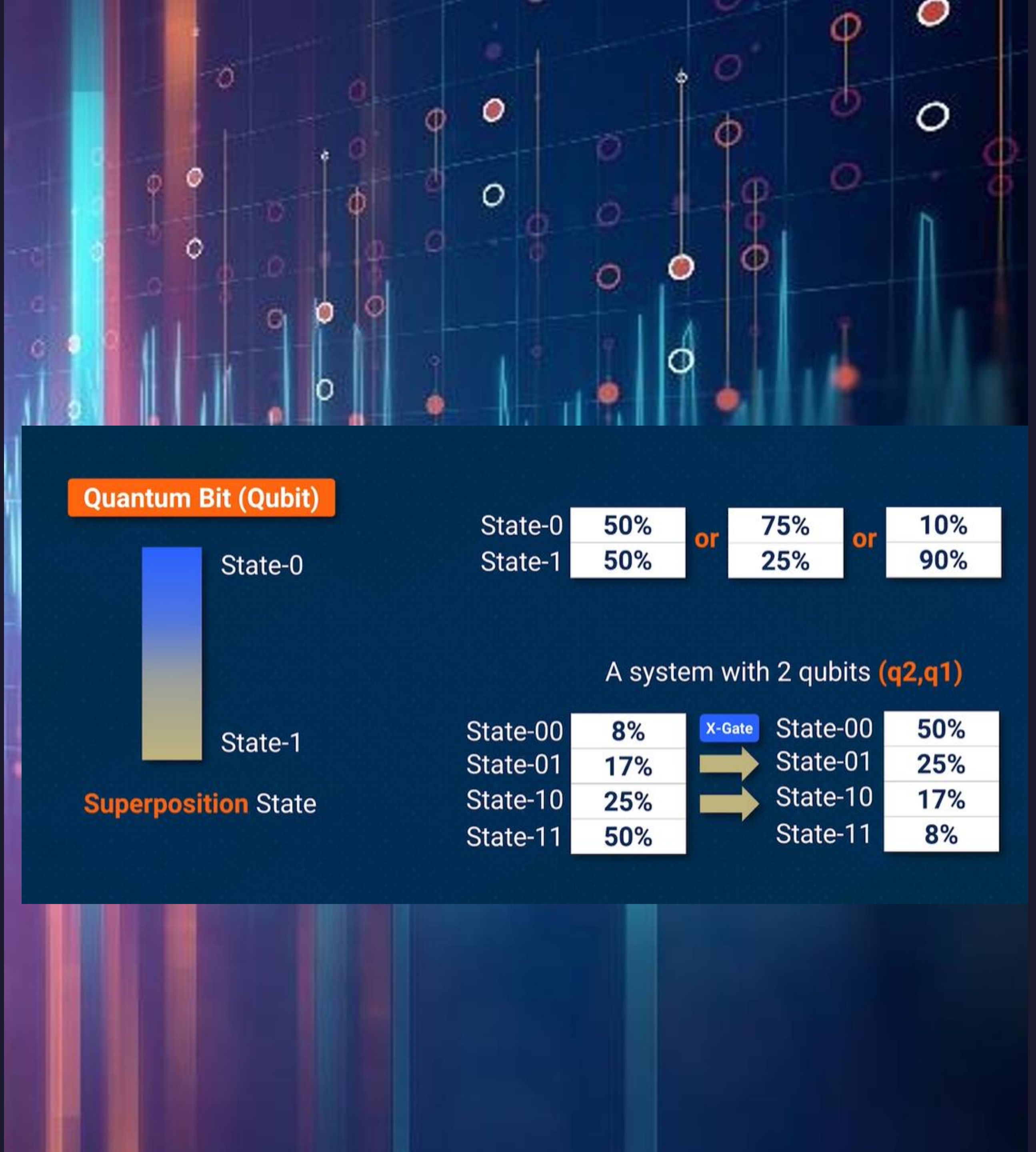
Classical bits

- Represented as 0 or 1
- Implemented as electrical voltage or light pulses
- Not as efficient for certain complex computations
- Limited scalability for high performance computing



Quantum bits

- Fundamental unit of Quantum information
- Qubits exist in a superposition
- Can be more efficient for certain complex computations
- Implemented through different quantum systems such as Superconducting circuits, trapped ions, photon-based qubits



Key Properties of Quantum Bits

Superposition

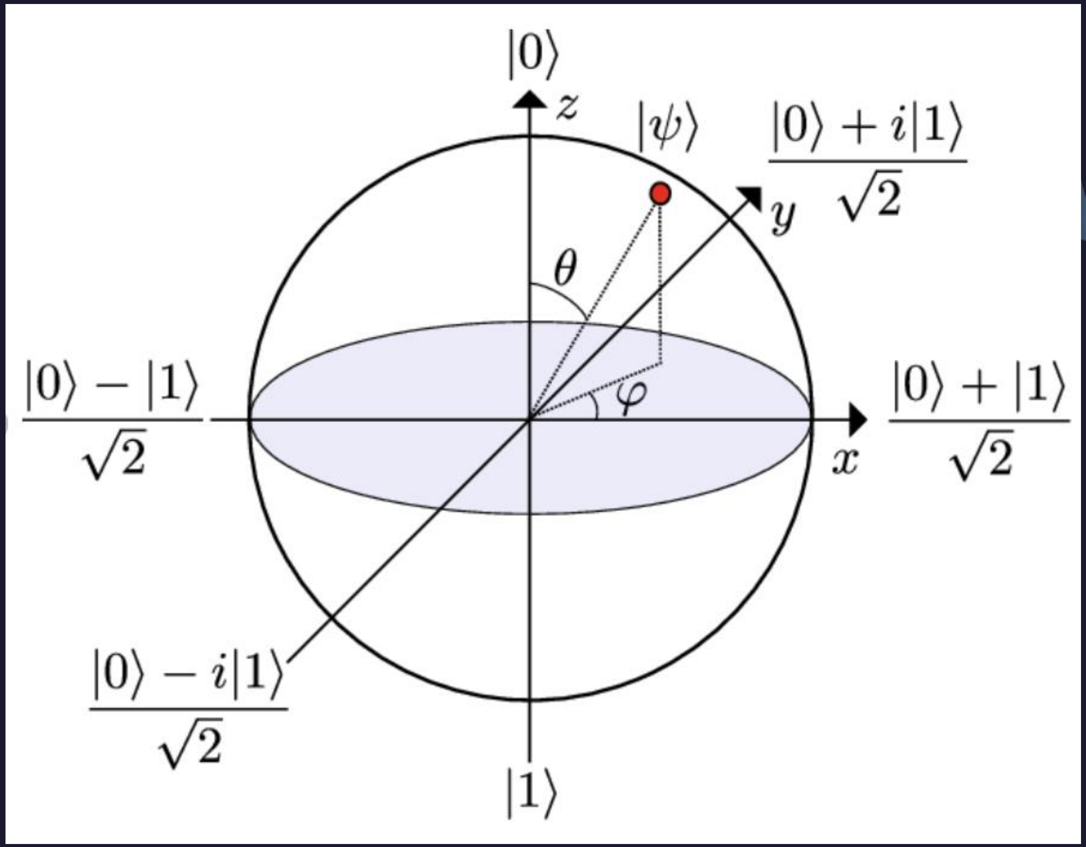
- Can be 0 or 1 simultaneously

Entanglement

- Qubits can be linked affecting other qubits instantaneously

Quantum Interference

- Probability amplitudes can be manipulated for computation



Differences between Quantum and classical bits

- A 4-bit classical system can represent 1 of 16 states at a time
- 1 bit only contains one unit of information
- Power increases linearly with number of bits

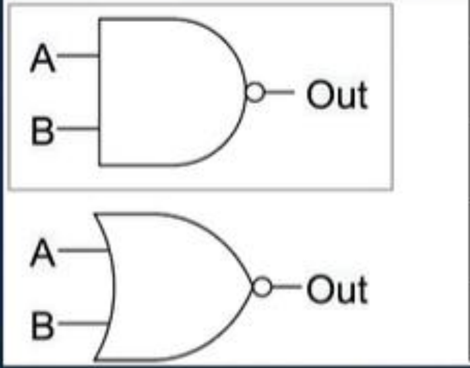
- A 4-bit Quantum system can represent all 16 states at simultaneously
- Due two entanglement one qubit can equal 2 bits of classical information
- Power increases exponentially with number of qubits

Quantum Logic gates

Quantum vs Classical Gates

Logic Gates Compared to Quantum Gates

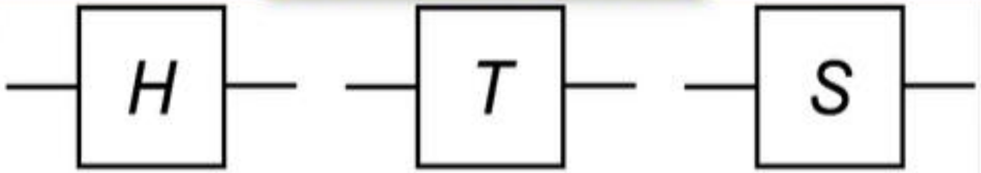
LOGIC GATES



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

- Operation on Boolean values
- Truth tables
- Most of them only run forward

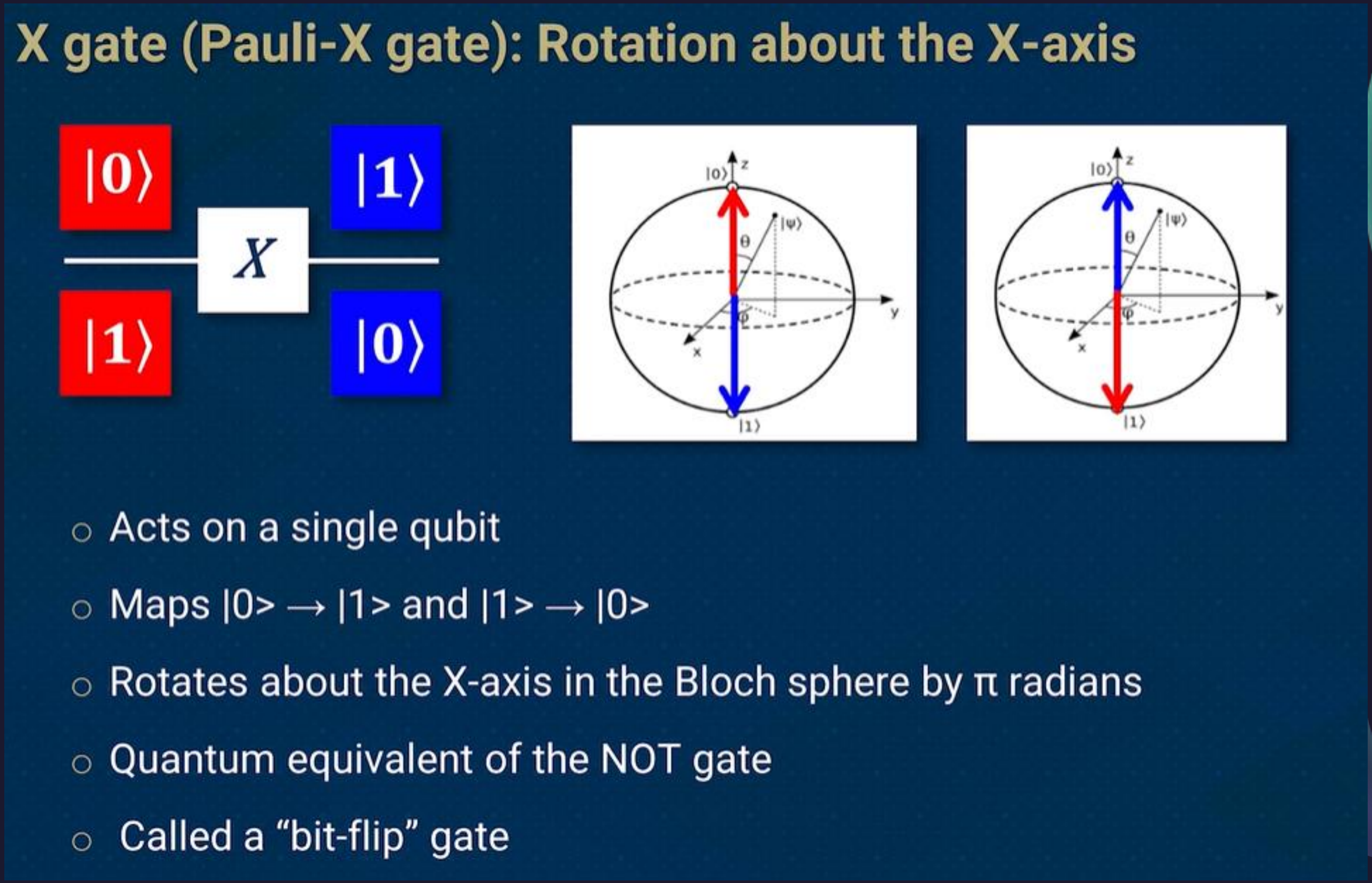
QUANTUM GATES



$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \quad \begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix} \quad \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}$$

- Operation on complex state
- Unitary matrix
- They are reversible

Pauli – X Gate

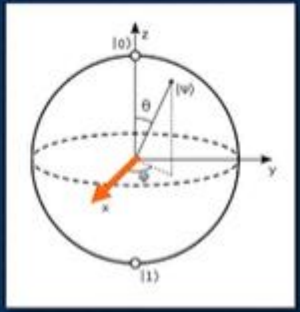

X gate (Pauli-X gate): Rotation about the X-axis
|0⟩
|1⟩
X
|1⟩
|0⟩
Acts on a single qubit
Maps |0> → |1> and |1> → |0>
Rotates about the X-axis in the Bloch sphere by π radians
Quantum equivalent of the NOT gate
Called a “bit-flip” gate

Pauli – Z Gate

Pauli-Z: Rotation about the Z-axis

Pauli-Z

$$\alpha_0|0\rangle + \alpha_1|1\rangle \; \text{—}\boxed{Z}\text{—} \; \alpha_0|0\rangle - \alpha_1|1\rangle$$

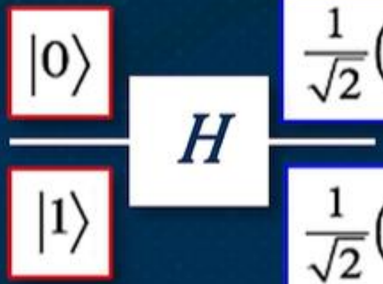


$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ -\alpha_1 \end{pmatrix}$$

- Maps |+> → |-> and |-> → |+>
- Rotates about the z-axis in the Bloch sphere by π radians
- Called a “phase-flip” gate

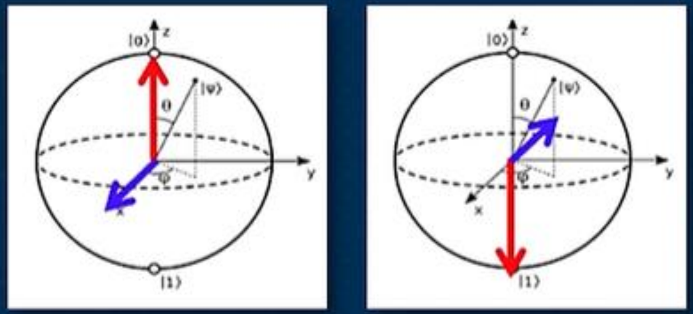
Hadamard Gate

Hadamard Gate: Creates Superposition



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

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



- Maps |0> → |+> and |1> → |->
- Given Standard basis states, generates superposition

Hadamard

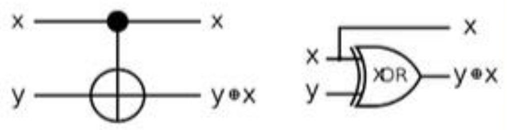
$$H \quad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$1/\sqrt{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = 1/\sqrt{2}\begin{pmatrix} \alpha_0+\alpha_1 \\ \alpha_0-\alpha_1 \end{pmatrix}$$

CNOT Gate

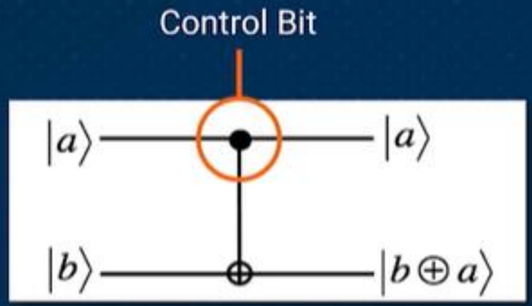
CNOT (Controlled NOT) or CX Gate

- **CNOT gate**: performs X gate on second qubit iff the first qubit is |1>



input x	input y	output x	output y+x
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

input x	input y	output x	output y+x
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0



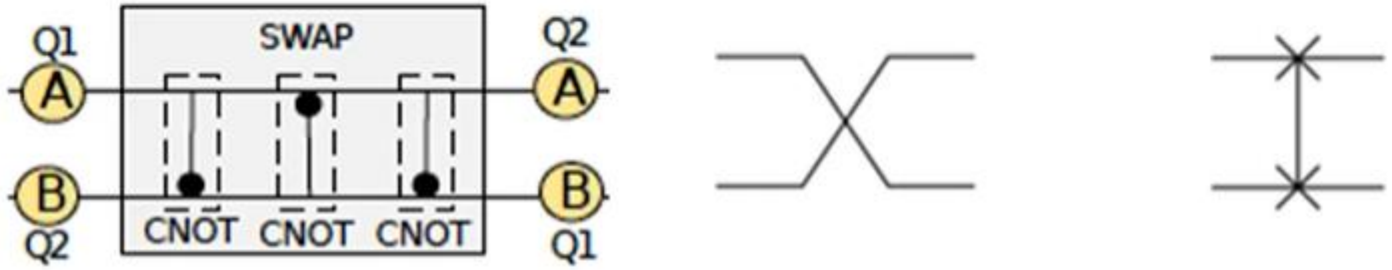
Unitary Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- **Controlled gates**: One or more qubits act as a control for some operation.
- Inputs are complex numbers; classical truth tables are for understanding only

Swap Gate

SWAP Gate: Swaps Two Cubits

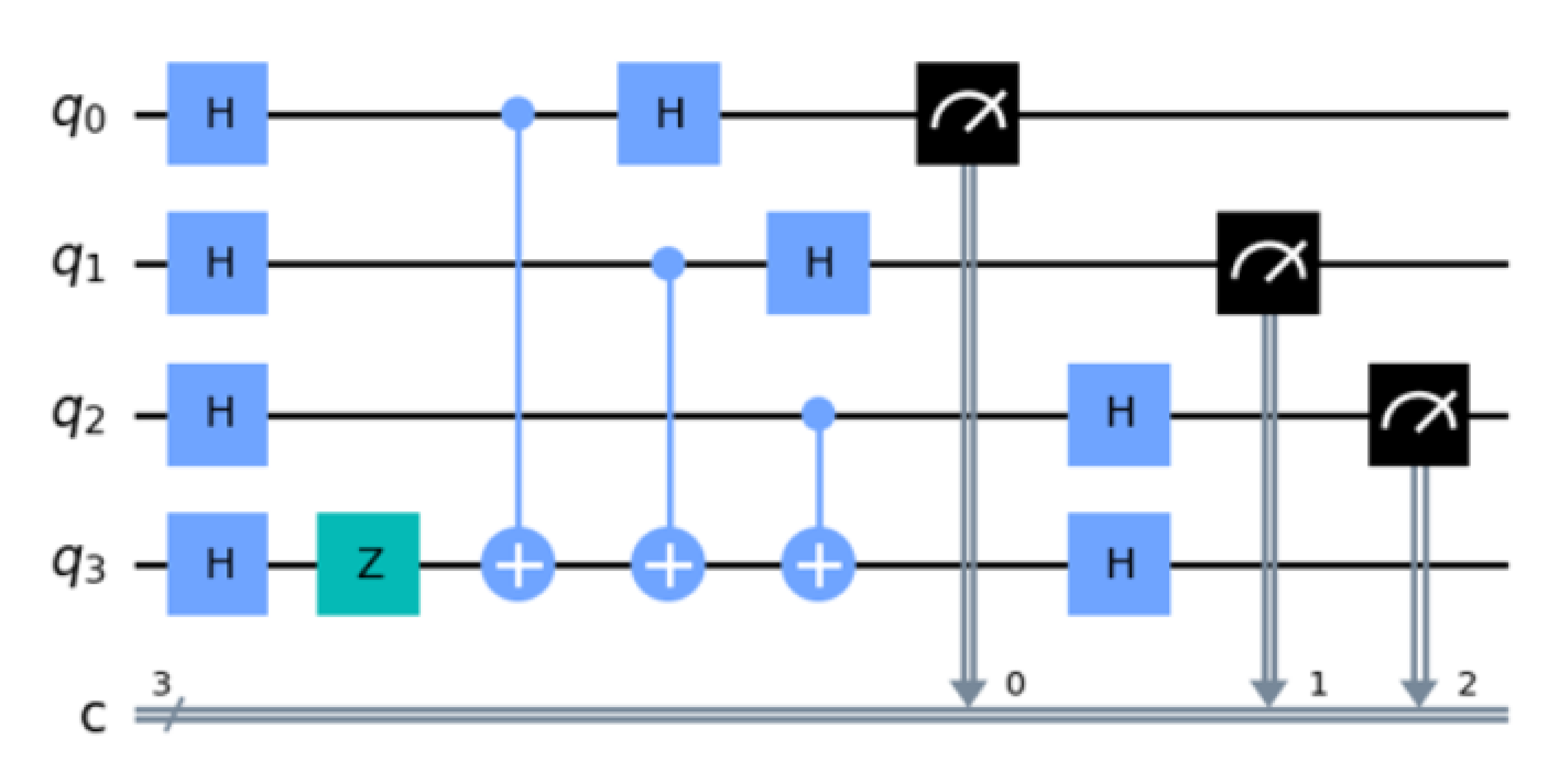


Unitary Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Typically implemented as 3 CNOTs
- Used to move qubits under restricted connectivity
- Recall that cloning of qubit state is not allowed

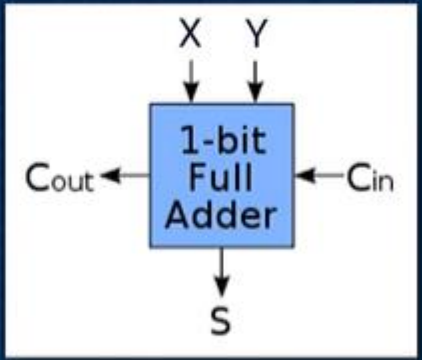
Representing a Quantum Circuit

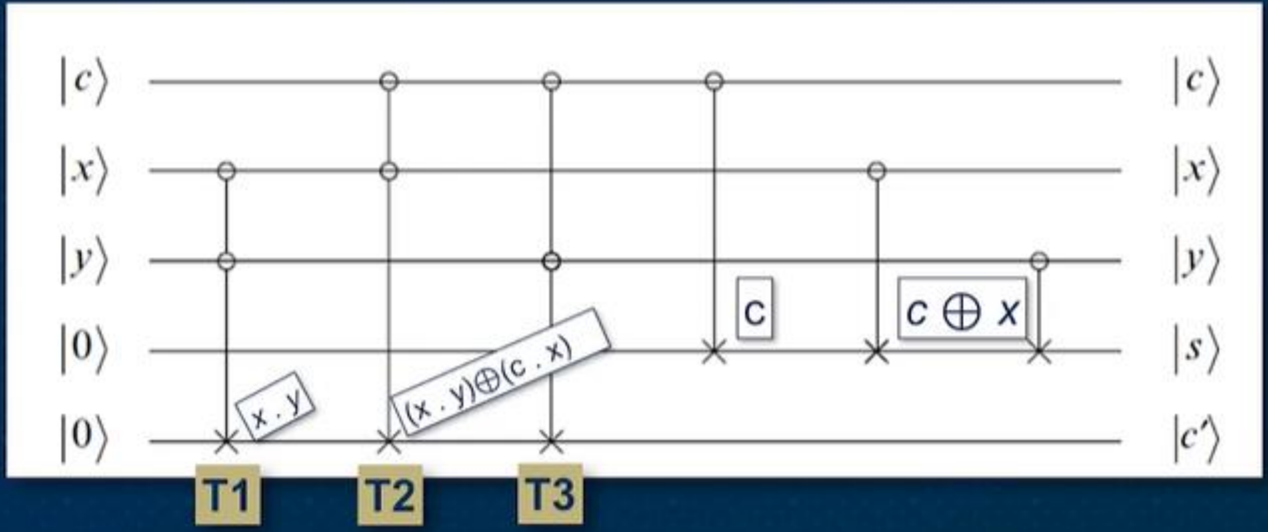


Adder circuit

Quantum ADDER

Cin	X	Y	T1	T2	T3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	1	0	1

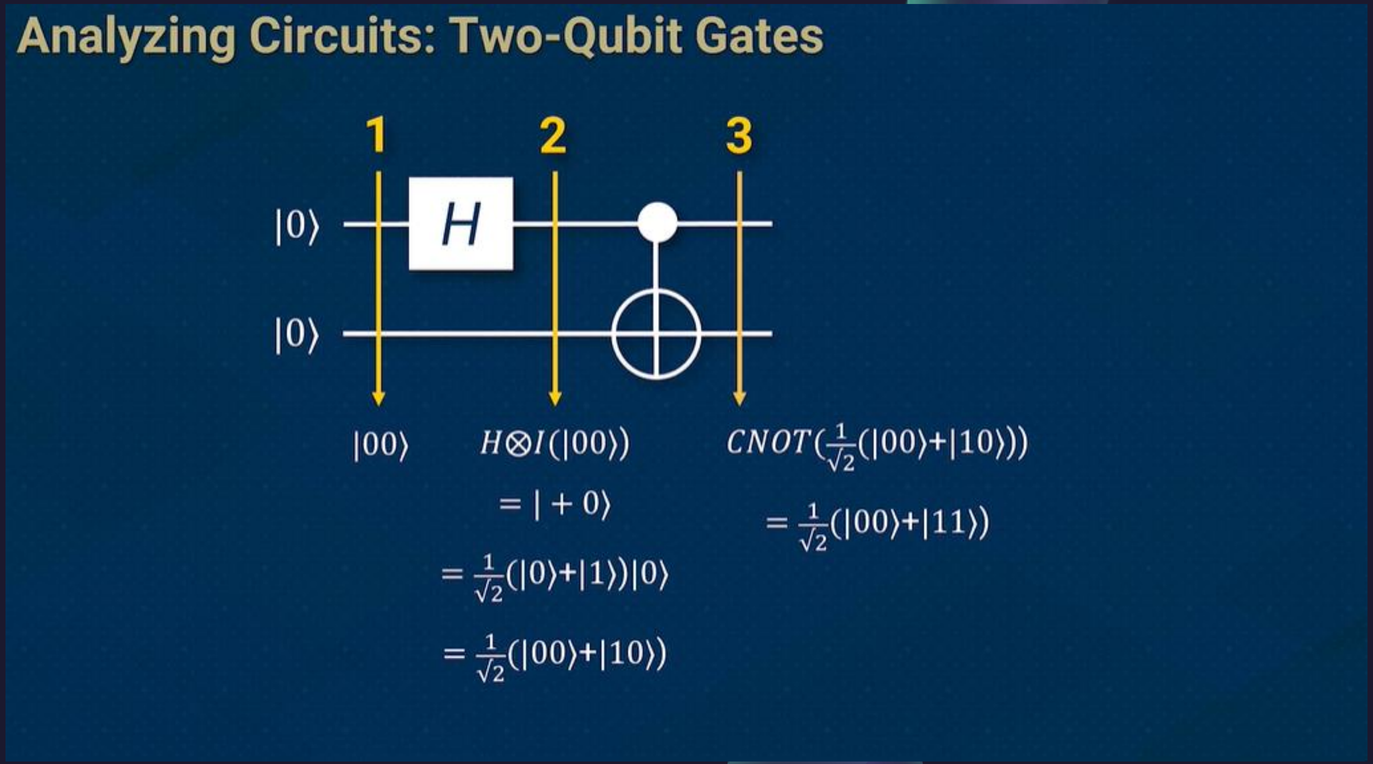




Quantum Properties

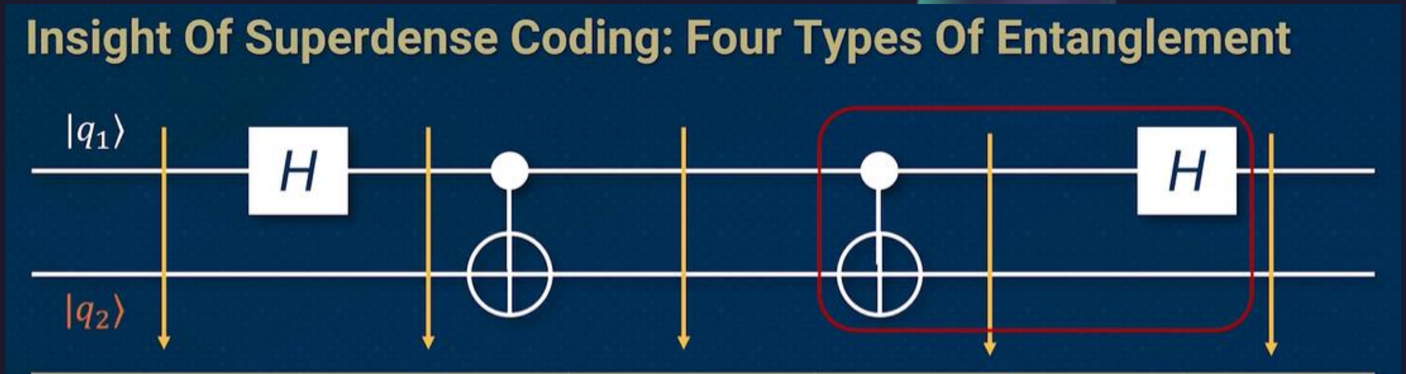
Entanglement

- Entanglement is generated using **microwave pulses** applied through control lines.
- A **Controlled-NOT (CNOT) gate** links two qubits, making them entangled.
- measuring one **instantly sets** the other's state
- Entanglement is fragile and can be lost due to environmental interference



Superdense Coding

- A quantum communication protocol that allows the transmission of 2 classical bits using only 1 qubit.
- 3 Steps:
  - **Entanglement**: Shared entangled qubits between Alice and Bob.
  - **Encoding**: Alice encodes 2 classical bits in 1 qubit.
  - **Decoding**: Bob decodes the information by measuring the entangled qubits.



1	2	3	4	5
$\|00\rangle$	$\frac{1}{\sqrt{2}}(\|00\rangle+\|10\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle+\|11\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle+\|10\rangle)=\|+0\rangle$	$\|00\rangle$
$\|01\rangle$	$\frac{1}{\sqrt{2}}(\|01\rangle+\|11\rangle)$	$\frac{1}{\sqrt{2}}(\|01\rangle+\|10\rangle)$	$\frac{1}{\sqrt{2}}(\|01\rangle+\|11\rangle)=\|+1\rangle$	$\|01\rangle$
$\|10\rangle$	$\frac{1}{\sqrt{2}}(\|00\rangle-\|10\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle-\|11\rangle)$	$\frac{1}{\sqrt{2}}(\|00\rangle-\|10\rangle)=\|-0\rangle$	$\|10\rangle$
$\|11\rangle$	$\frac{1}{\sqrt{2}}(\|01\rangle-\|11\rangle)$	$\frac{1}{\sqrt{2}}(\|01\rangle-\|10\rangle)$	$\frac{1}{\sqrt{2}}(\|01\rangle-\|11\rangle)=\|-1\rangle$	$\|11\rangle$

Superdense Coding: Encoding

- Taking in a string of classical information and looping through it, we send two bits of information at a time
- Encoding:
    - **00**: No operation (Identity gate).
    - **01**: Apply **X gate**.
    - **10**: Apply **Z gate**.
    - **11**: Apply **XZ gate**.

```python
def qubit_encoding(prepared_qubits, classical_information):
    """
    Based on the Super Dense Coding principle, Alice can only use the first ceil(n/2) qubits of the prepared qubits.
    Args:
        prepared_qubits: QuantumCircuit of prepared qubits
        classical_information: str
    Return:
        qcirc: QuantumCircuit of encoded qubits
    """

    qcirc = None
    qcirc = prepared_qubits.copy()  # Start with the prepared qubits
    if len(classical_information) % 2 !=0:
        classical_information = classical_information + "0"
    n = len(classical_information)
    print(classical_information)
    count = 0
    num_qubits = prepared_qubits.num_qubits

    for i in range(0,num_qubits,2):
        qcirc.h(i)
        qcirc.cx(i,i+1)

        s = classical_information[i:i+2]
        print(s)
        if s == "01":
            qcirc.x(i)
        elif s == "10":
            qcirc.z(i)
        elif s == "11":
            qcirc.x(i)
            qcirc.z(i)
        count +=1
    # print(qcirc)
    return qcirc
```

Restart Visual Studio Code to appl

Upda

Superdense Coding: Decoding

- Taking in the Encoded bits we apply a CNOT gate on q i and q i+1 to get the decoded message
- To keep the order of the message all output bits were placed after their encoded bit
- For a message of length n only n/2 encoded bits of information need to be sent

```
def qubit_decoding(encoded_qubits, n):
    """
    Bob restores the classical information using the encoded qubits
    Args:
        encoded_qubits: QuantumCircuit of encoded qubits
        n: int of length of classical_information
    Return:
        restored_information: str
    """
    lend = n
    if  lend % 2 != 0:
        lend +=1

    restored_information = ""
    for i in range(0,lend,2):
        print(i)
        encoded_qubits.cx(i,i+1)
        encoded_qubits.h(i)
    print(encoded_qubits)
    encoded_qubits.measure_all()
    simulator = BasicSimulator()
    cc = transpile(encoded_qubits, simulator)
    job = simulator.run(cc, shots=100)
    result = list(job.result().get_counts(cc).keys())[0]
    restored_information = result
```

Quantum Algorithm's

Constant or Balanced Problem

- You are given a function $f : \{0,1\}^n \to \{0,1\}$, which is either:
  - **Constant**: $f(x) = c$, where c is the same for all inputs x.
  - **Balanced**: $f(x) = 0$ for half of the inputs and $f(x) = 1$ for the other half.

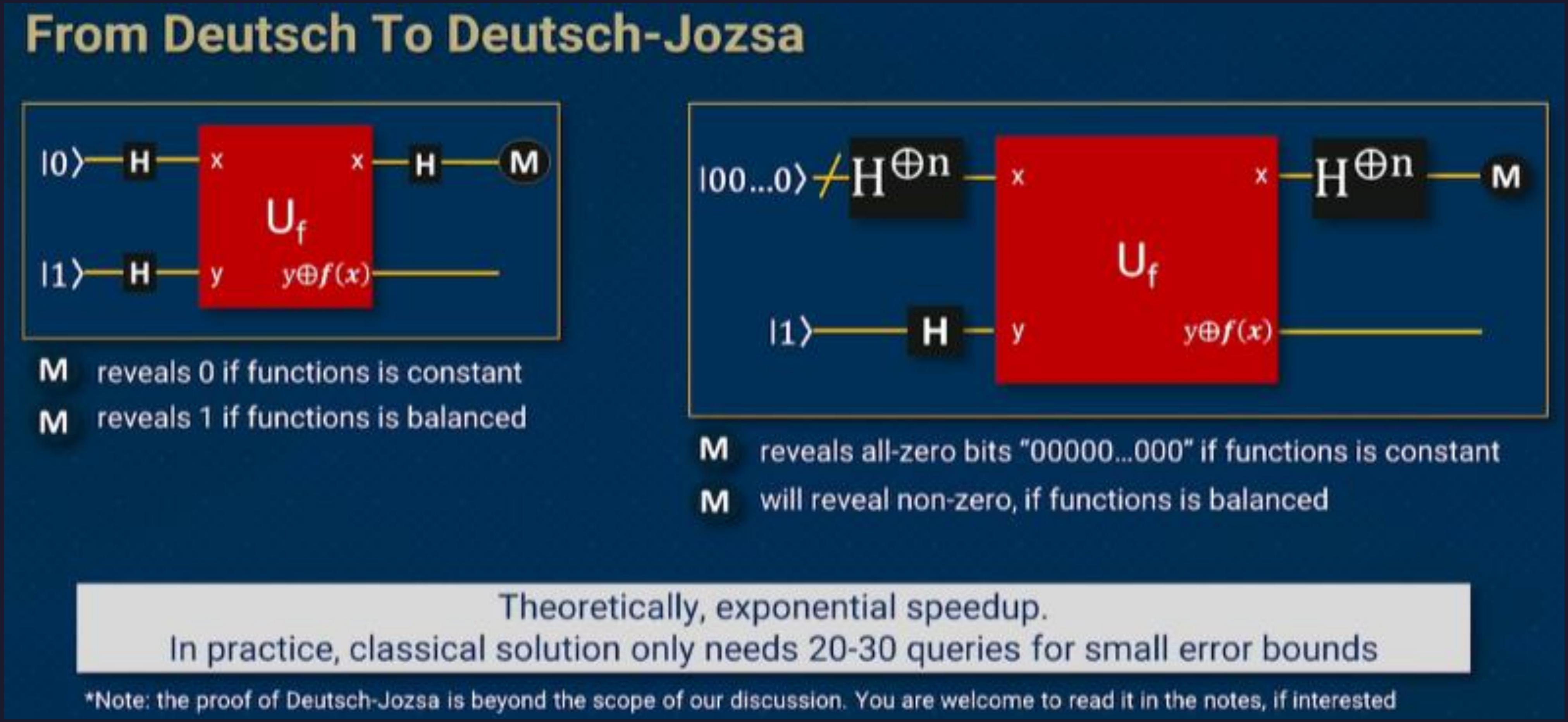
Classical Solution:

- In the **classical case**, you would need to evaluate the function f at least 2^n/2 + 1 times in the worst case to determine if it's constant or balanced. Since one more query would be needed to see if more than half are either 0 or 1

Quantum Solution:

- In the **Quantum case**, you would need to evaluate the function once.

Deutsch-Joza Algorithm



Deutsch-Joza Algorithm Steps

1. **Quantum Superposition and Parallelism:**

   - You start by initializing your system in a superposition of all possible input values by applying a **Hadamard gate** to all the qubits, putting the quantum state into an equal superposition of all possible inputs.

2. **Oracle Query**:

- The quantum oracle applies the function f(x) to each possible qubit, modifying the auxiliary qubit based on the function's output. Importantly, this **does not require evaluating the function multiple times in sequence**; it's done in a single operation.

3.After the oracle step, the quantum state is now a **superposition** of all possible inputs, each carrying information about whether the function f(x) is 0 or 1 for that specific input

4. **Interference with Hadamard**:

- After applying the oracle, you perform another round of **Hadamard gates** on the first n qubits.
- This step induces **interference**: it causes the amplitudes of different states to combine in such a way that when you measure the qubits, the quantum state collapses to one of two outcomes:

  - $|0\rangle$ (if the function is constant), or
  - $|1\rangle$ (if the function is balanced).

Quantum VS Classical Algorithm's

Bad For

Simple Arithmetic and Classical Computation

- **Problem**: Basic calculations or tasks that don't involve complex or large datasets.
- **Why Not Quantum**: Classical computers are still more efficient for simple, non-parallelizable tasks like addition, subtraction, or small-scale multiplications.

Problems with No Quantum Advantage

- **Problem**: Problems that don't involve large data sets or don't benefit from superposition or quantum parallelism.
- **Why Not Quantum**: For many everyday problems, quantum computers may offer no speedup and would be overkill.

- **Problems that Require Error-Free Computation**

Large-Scale Classical Cryptography (Post-Quantum Cryptography)

- **Problem**: Classical encryption systems (e.g., AES, hash functions).
- **Why Not Quantum**: Quantum computers pose a threat to RSA and other public-key systems but are not yet a practical threat to symmetric-key cryptography like AES. Classical solutions remain effective for these problems in the current state of quantum computing.

Problem's with Quantum Algorithms

- **1. Quantum Decoherence**
  - Qubits lose their quantum state due to interactions with the environment.
  - Short **coherence times** lead to errors before computations finish.
- **2. Gate Imperfections**
  - Quantum gates (e.g., CNOT, Hadamard) are not perfectly implemented.
  - Errors occur due to hardware limitations in applying precise operations.

3. Crosstalk Between Qubits

- Nearby qubits unintentionally interfere with each other.
- Causes unwanted entanglement, leading to computational errors.

4. Measurement Errors

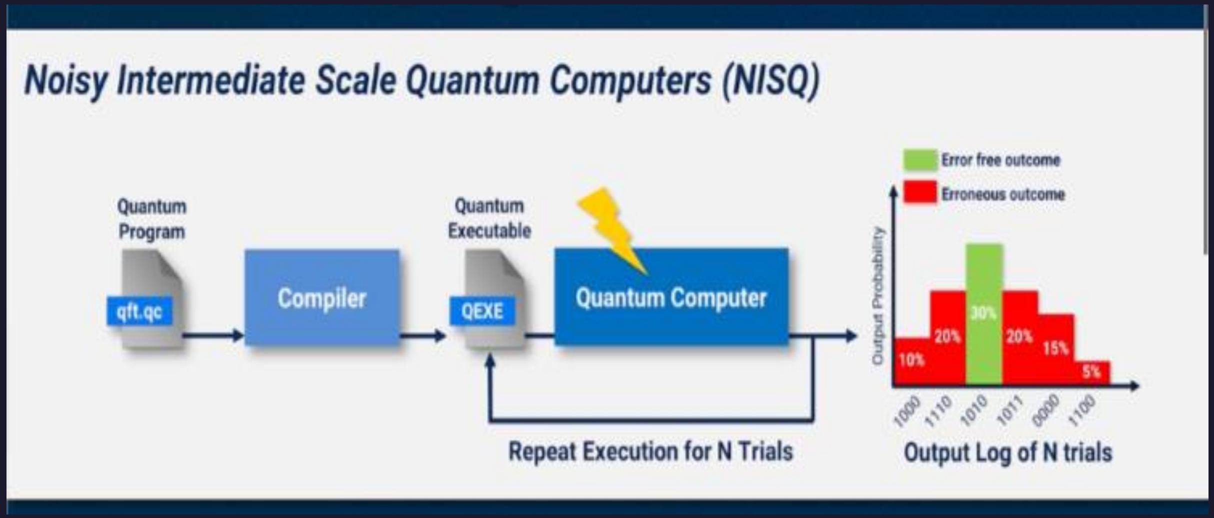
- Reading out qubits introduces noise.
- Errors occur when $qubit$ $|0\rangle$ is misread as $|1\rangle$ (or vice versa).

5. External Noise & Temperature Sensitivity

- Quantum processors operate near **absolute zero** (~15mK).
- Even tiny vibrations or electromagnetic interference can disrupt qubits.

Noisy Intermediate Scale Quantum

- current generation of quantum computers that are limited in size, prone to significant errors due to environmental noise.
- Running the same quantum circuit **thousands or millions of times** helps identify the most likely correct answer.
- Currently the top end of quantum computers have around a 1180 qubits (Atom Computing)



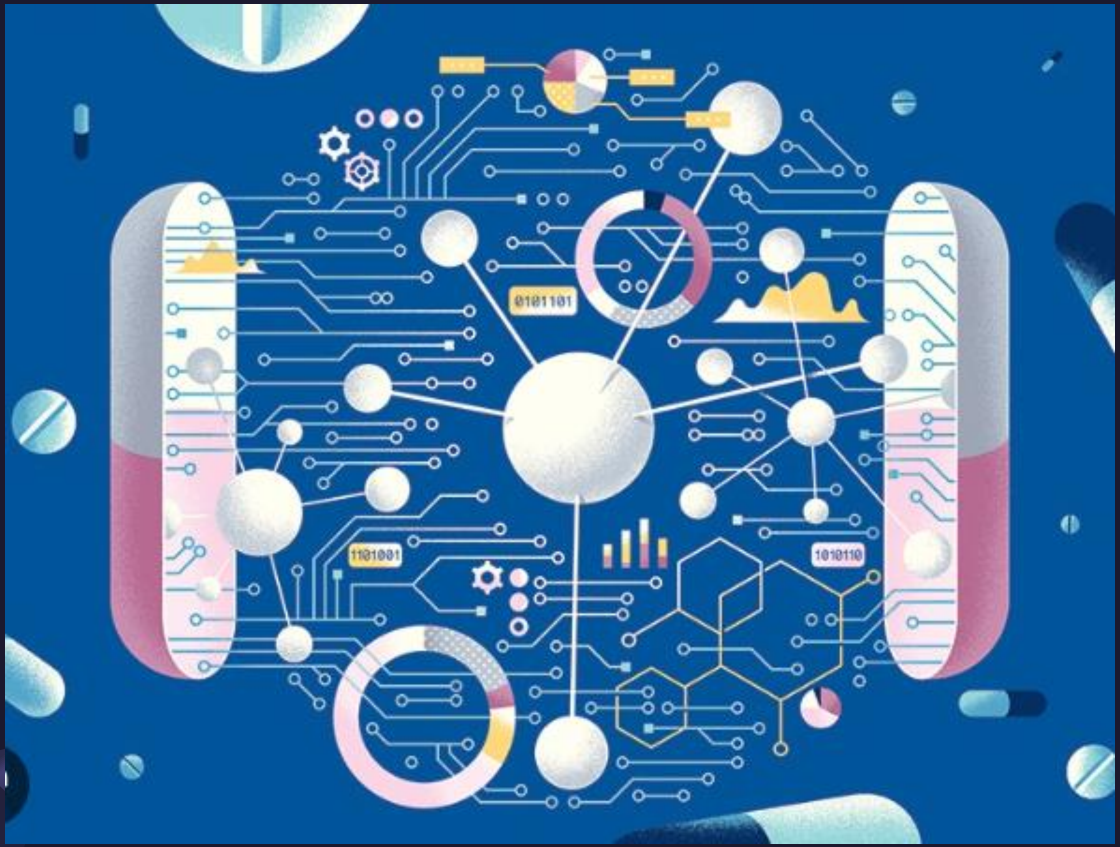
Quantum Algorithm Use Cases

Use Casses

Supply Chain Management - Route Optimization

- **Use Case**: Quantum algorithms can optimize delivery routes, minimizing time and fuel costs in logistics.
- **Algorithm**: **Quantum Approximate Optimization Algorithm (QAOA)**, **Grover's Algorithm**
- **Benefit**: Finding optimal routes faster by processing large datasets more efficiently, crucial for industries like logistics, transportation, and e-commerce.

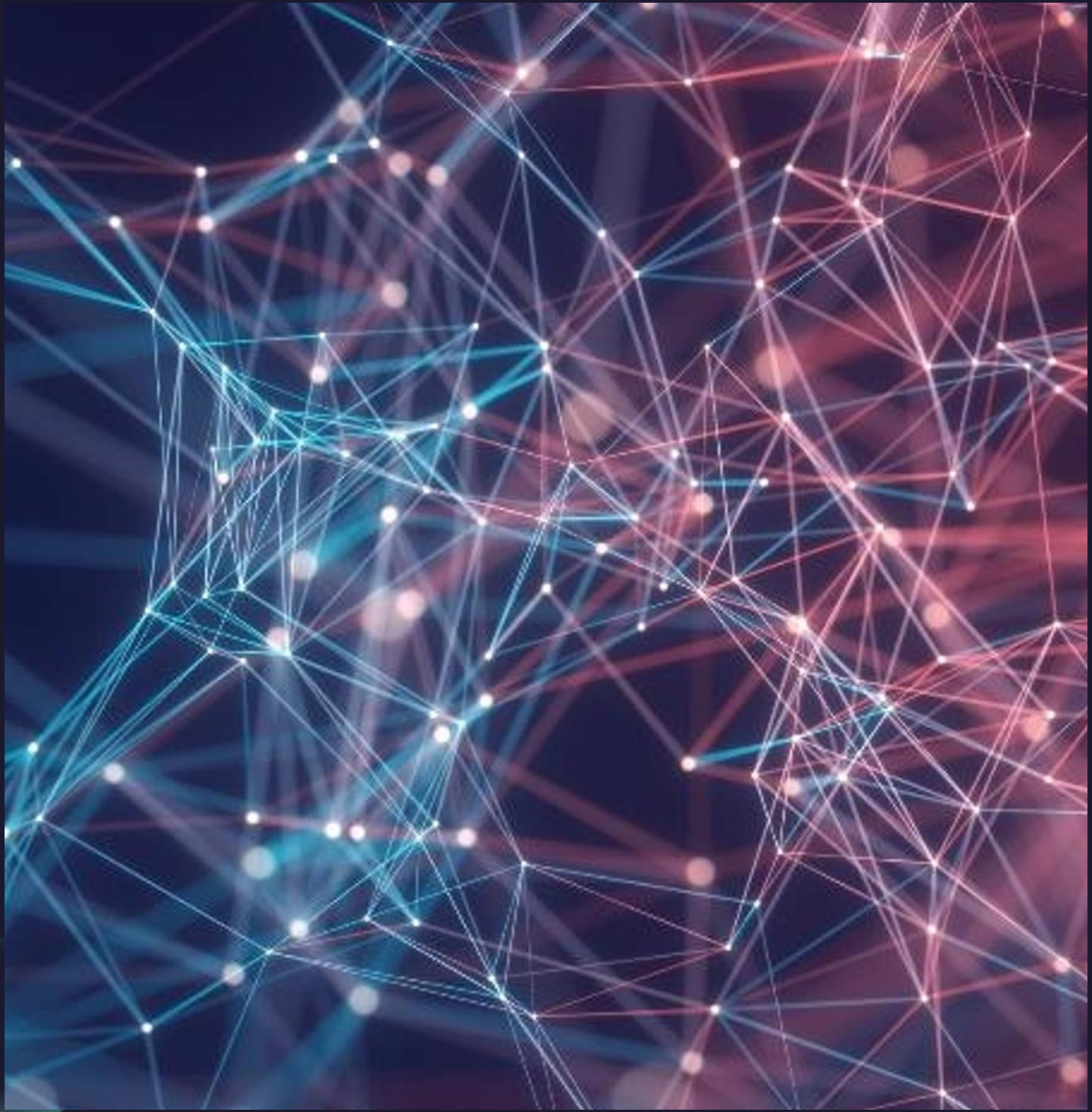
Use Casses



Drug Discovery

- **Use Case**: Quantum computers can simulate molecular interactions at a level of detail that is impossible with classical computers, aiding in faster drug discovery.
- **Algorithm**: **Quantum Simulations** (e.g., **Variational Quantum Eigensolver (VQE)**)
- **Benefit**: Acceleration of finding promising drug candidates, reducing time and cost in the pharmaceutical industry.

Thank you



Teleportation

1. Entanglement Setup

- Qubits **A** & **B** are entangled, linking their states across any distance.
- A third qubit (**C**) holds the quantum state to be teleported.

2. Bell Measurement & Classical Communication

- Alice measures qubits **C** & **A**, collapsing their states into a known classical result.
- She sends **two classical bits** to Bob.

3. Quantum State Reconstruction

- Bob applies a quantum operation on qubit **B** based on Alice's message.
- This restores the original state from qubit **C** onto **B**.