بسم الله الرحمن الرحیم

The HHL Algorithm

for Solving System of Linear Equations

Azim Farghadan

October 2024

مرکز تحقیقات
فناوری‌های
کوانتومی ایران

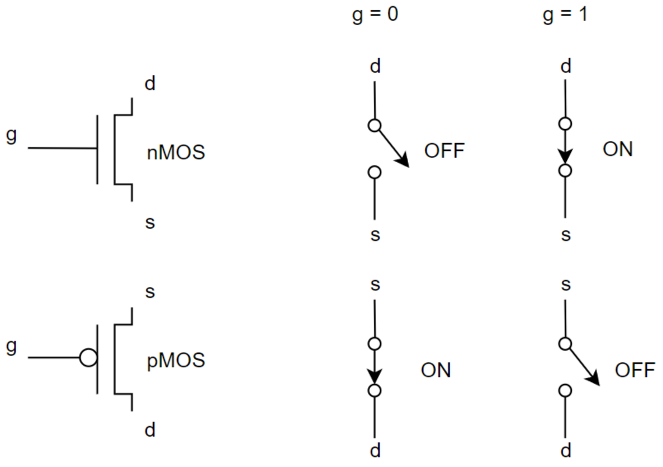
Objectives of the Lecture

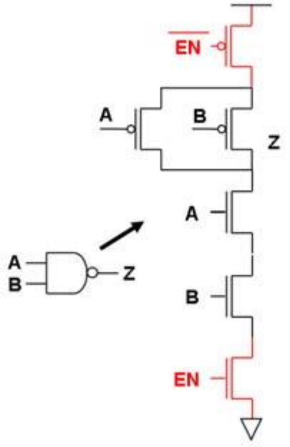
- ✓ Why quantum computing is necessary?
- ✓ What is the HHL algorithm?
- ✓ Simulation of quantum algorithms.

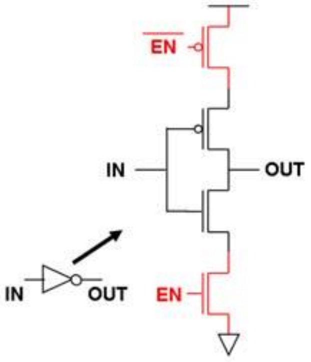
Why quantum computing is necessary?

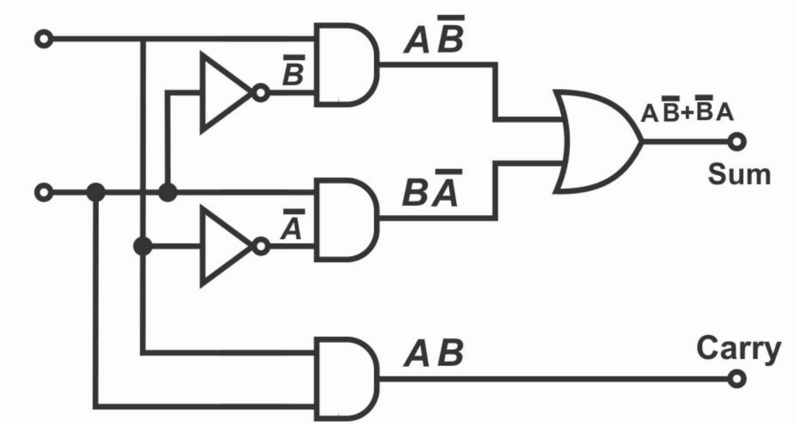

PROJECTS
MOONSHOT
PROJECTS
MOONSHO
PROJECTS
HACKATHO

The Need for Quantum Computation

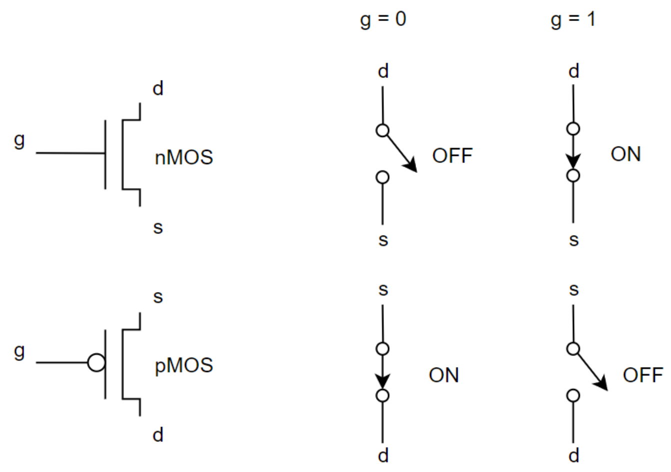


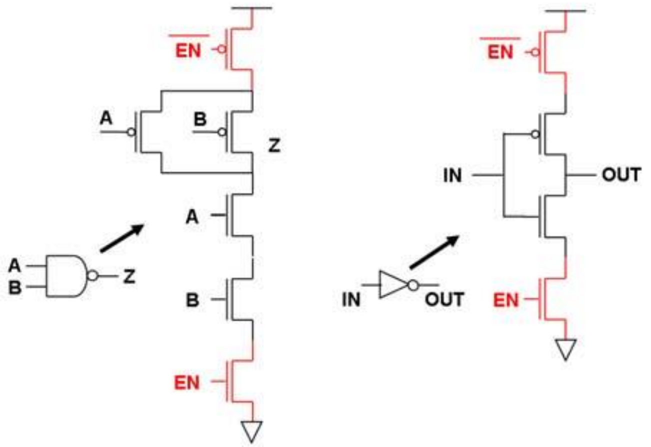


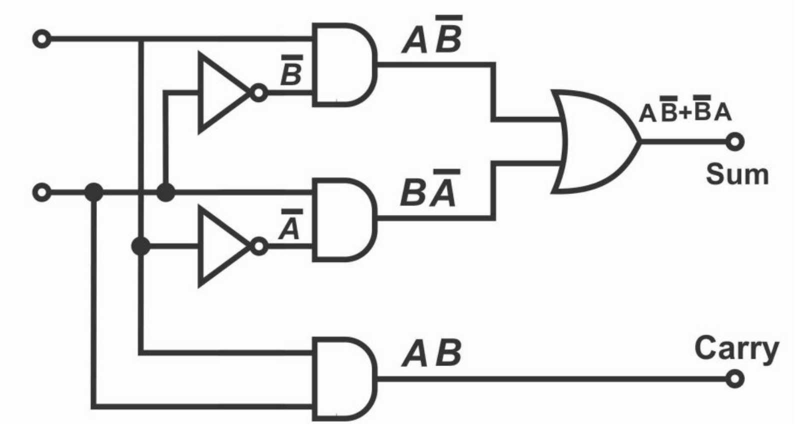


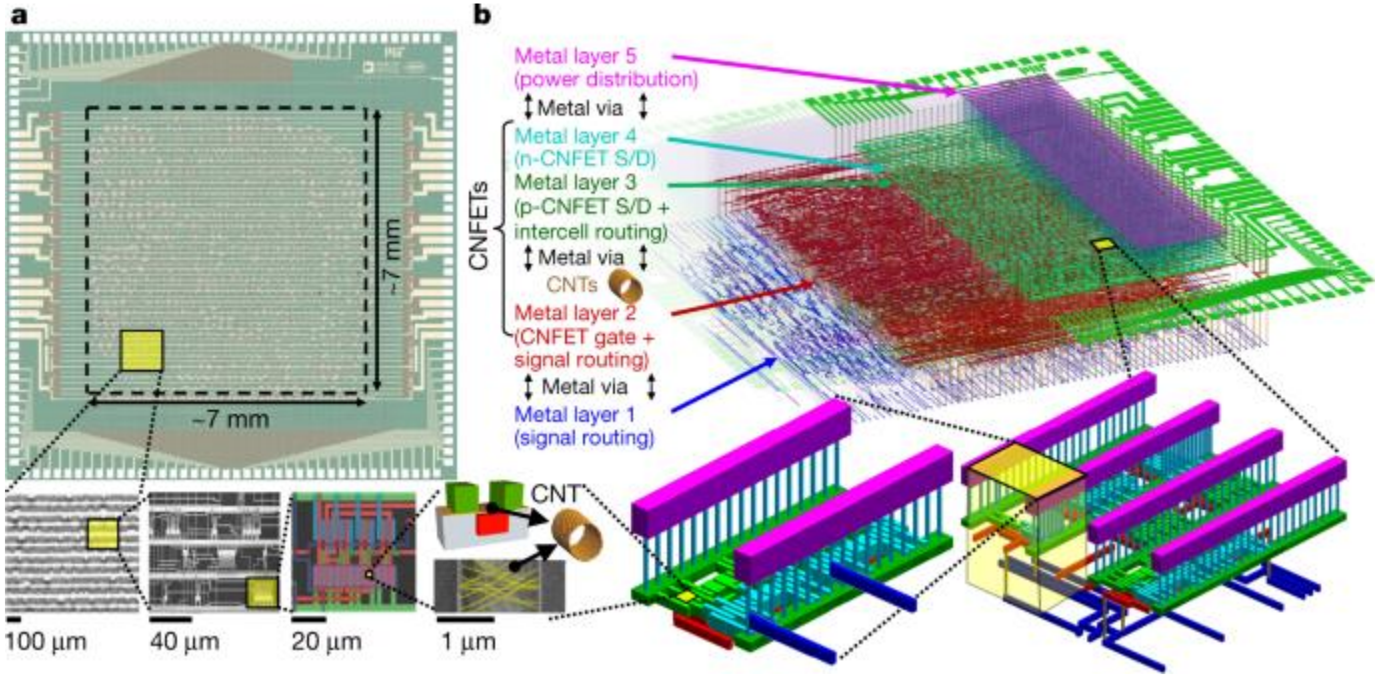


The Need for Quantum Computation

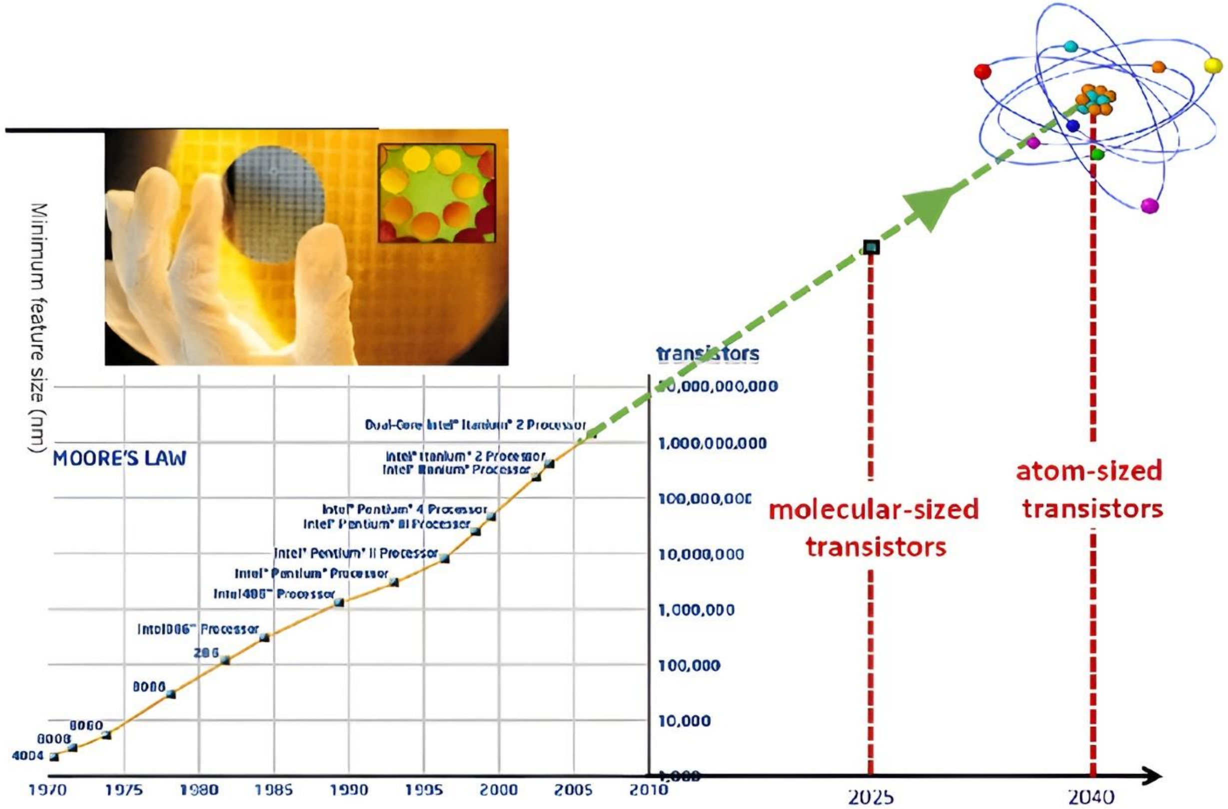


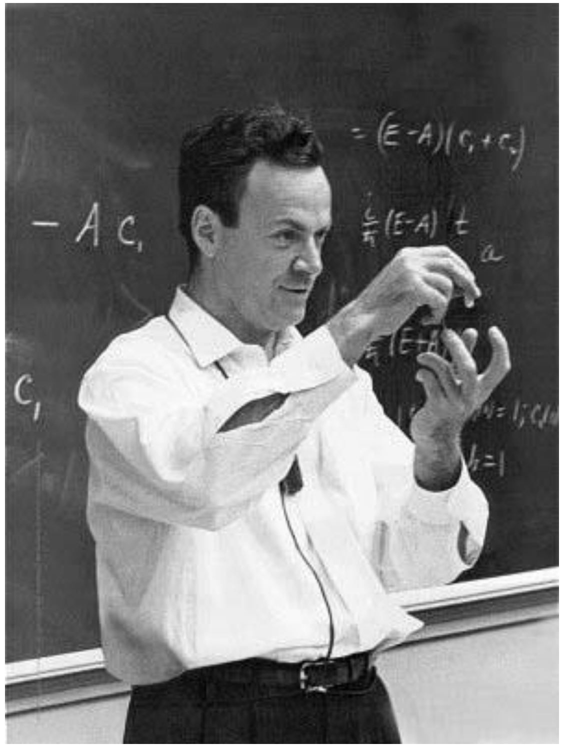




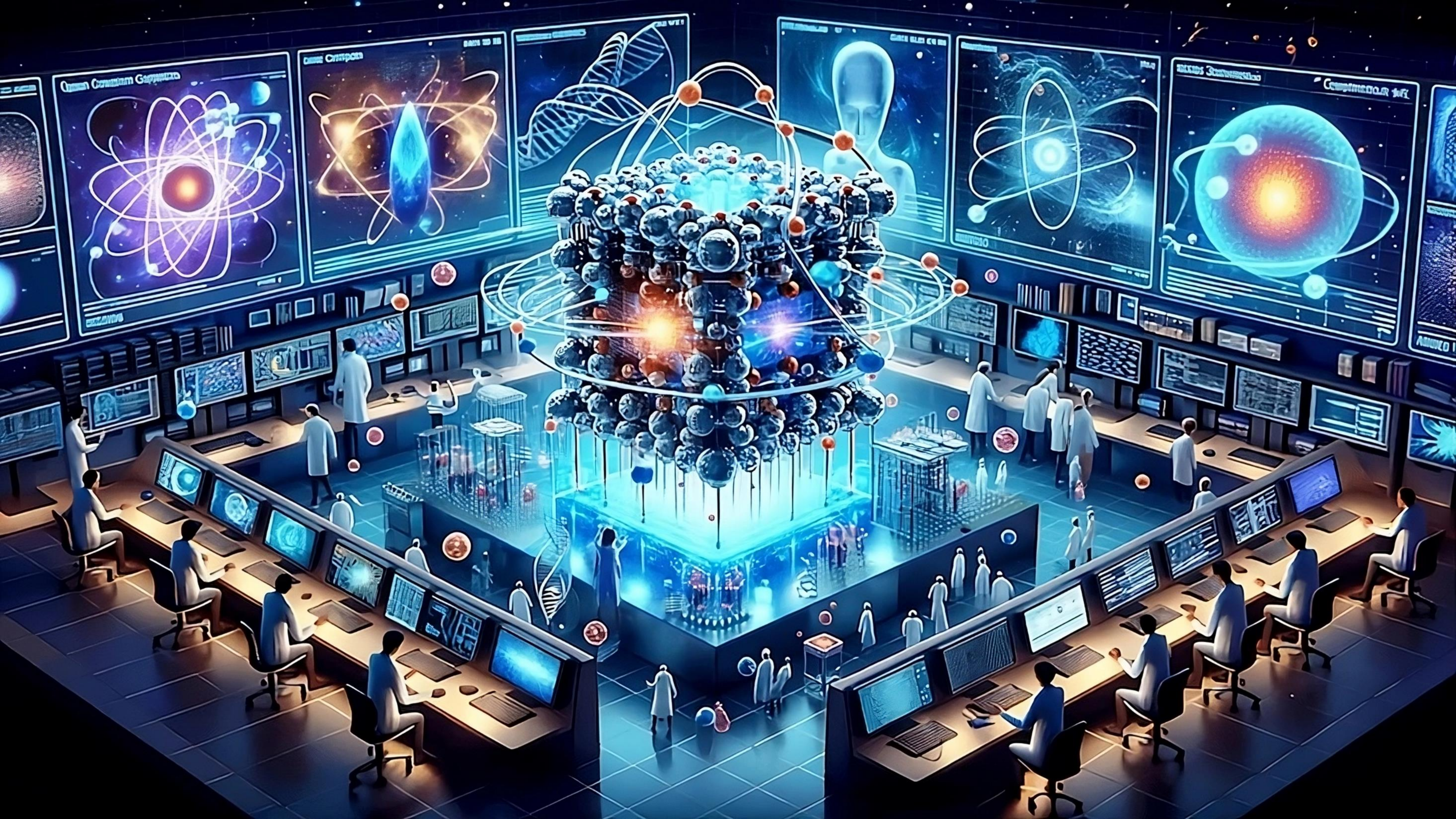


The Need for Quantum Computation

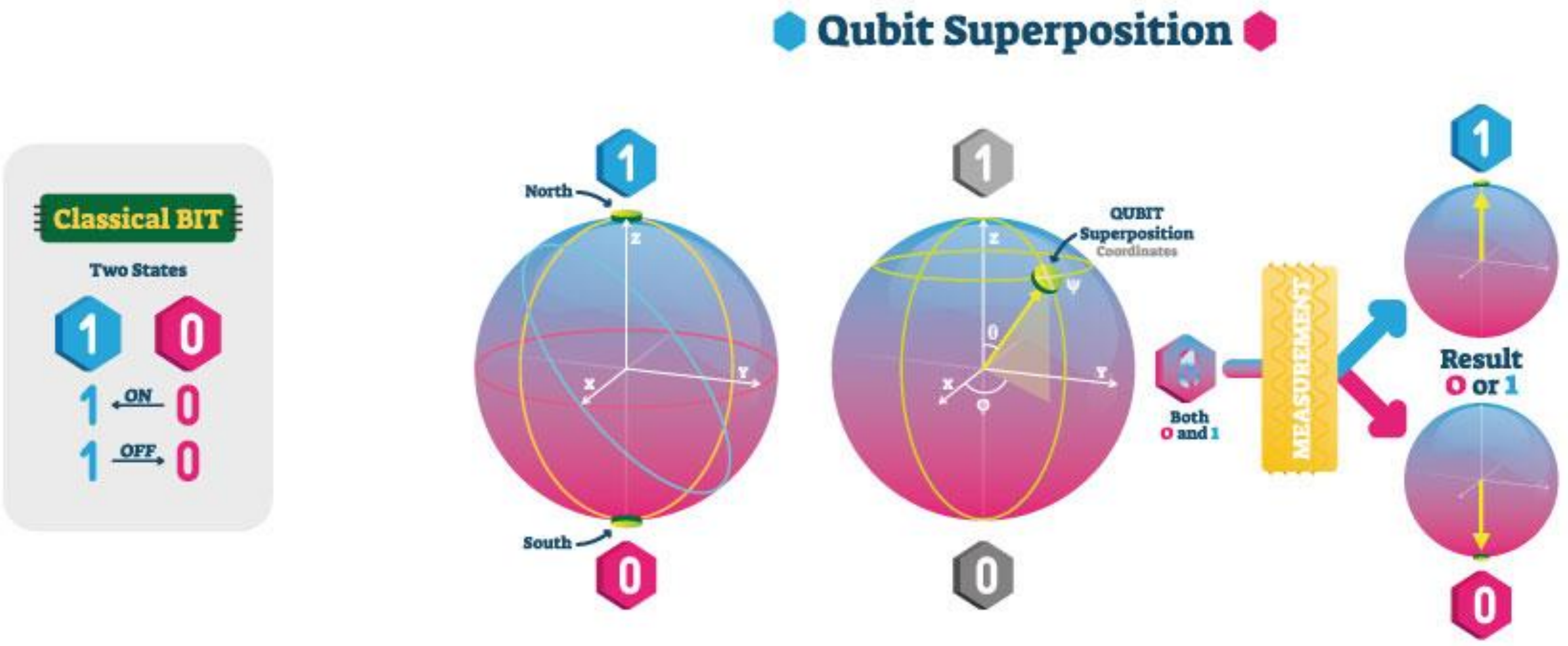




- 1981: Richard Feynman proposed the idea of creating machines based on the laws of quantum mechanics instead of the laws of classical physics.



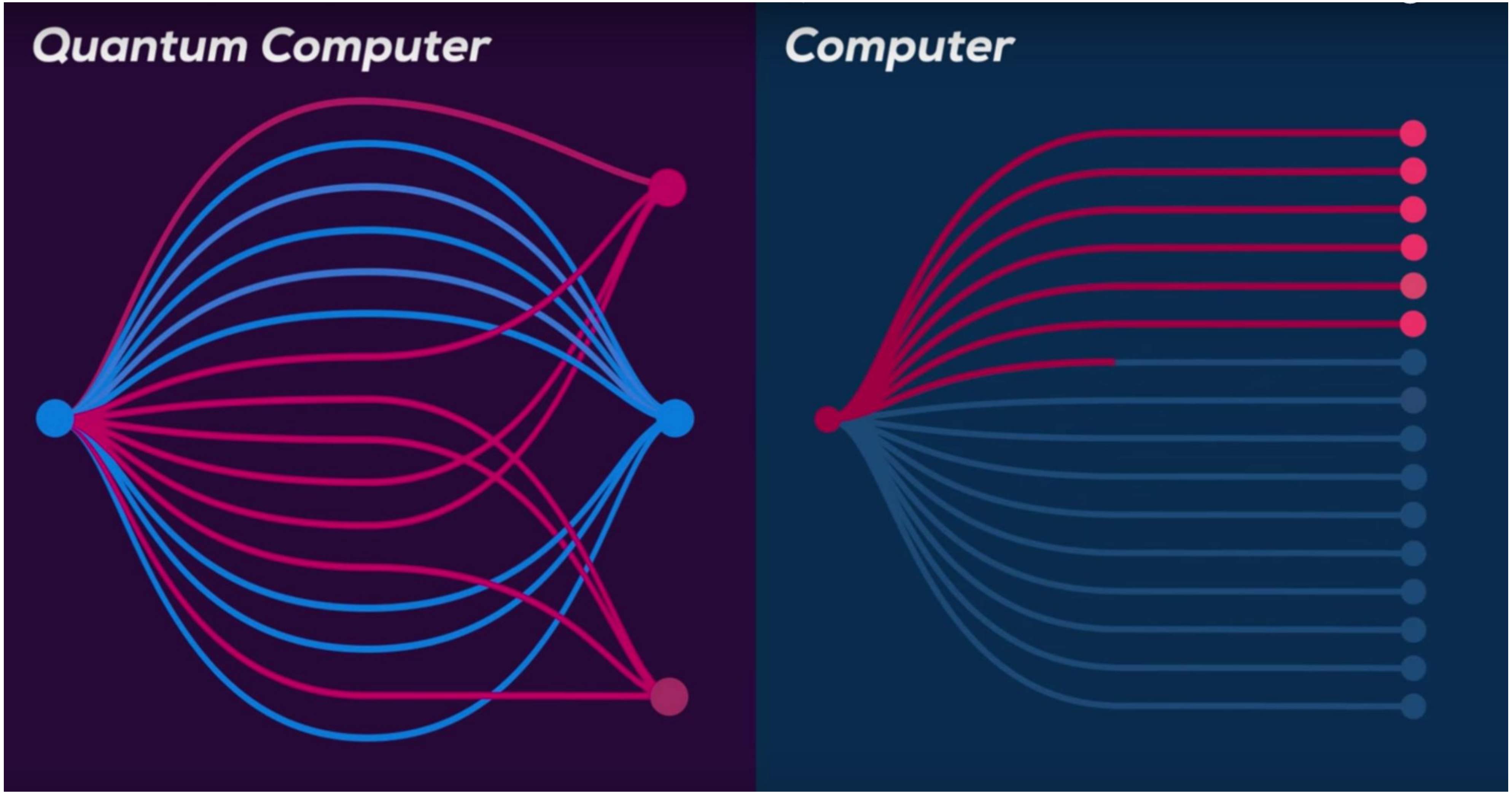
Key Concepts in Quantum Computing



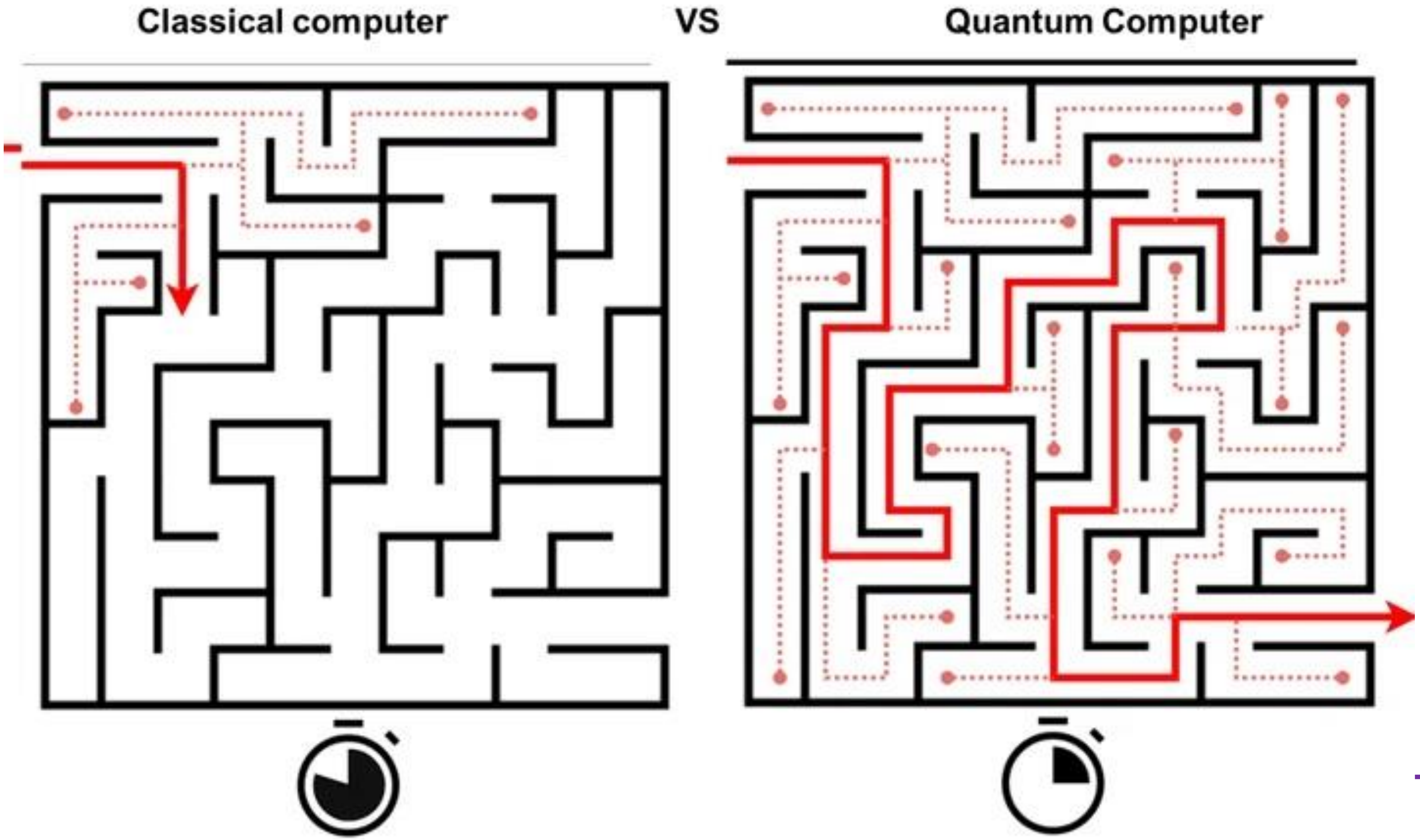
$$|\psi> = \alpha_1|0> + \alpha_2|1>$$

Where α_1 and α_2 are complex numbers and $|\alpha_1|^2 + |\alpha_2|^2 = 1$

Key Concepts in Quantum Computing

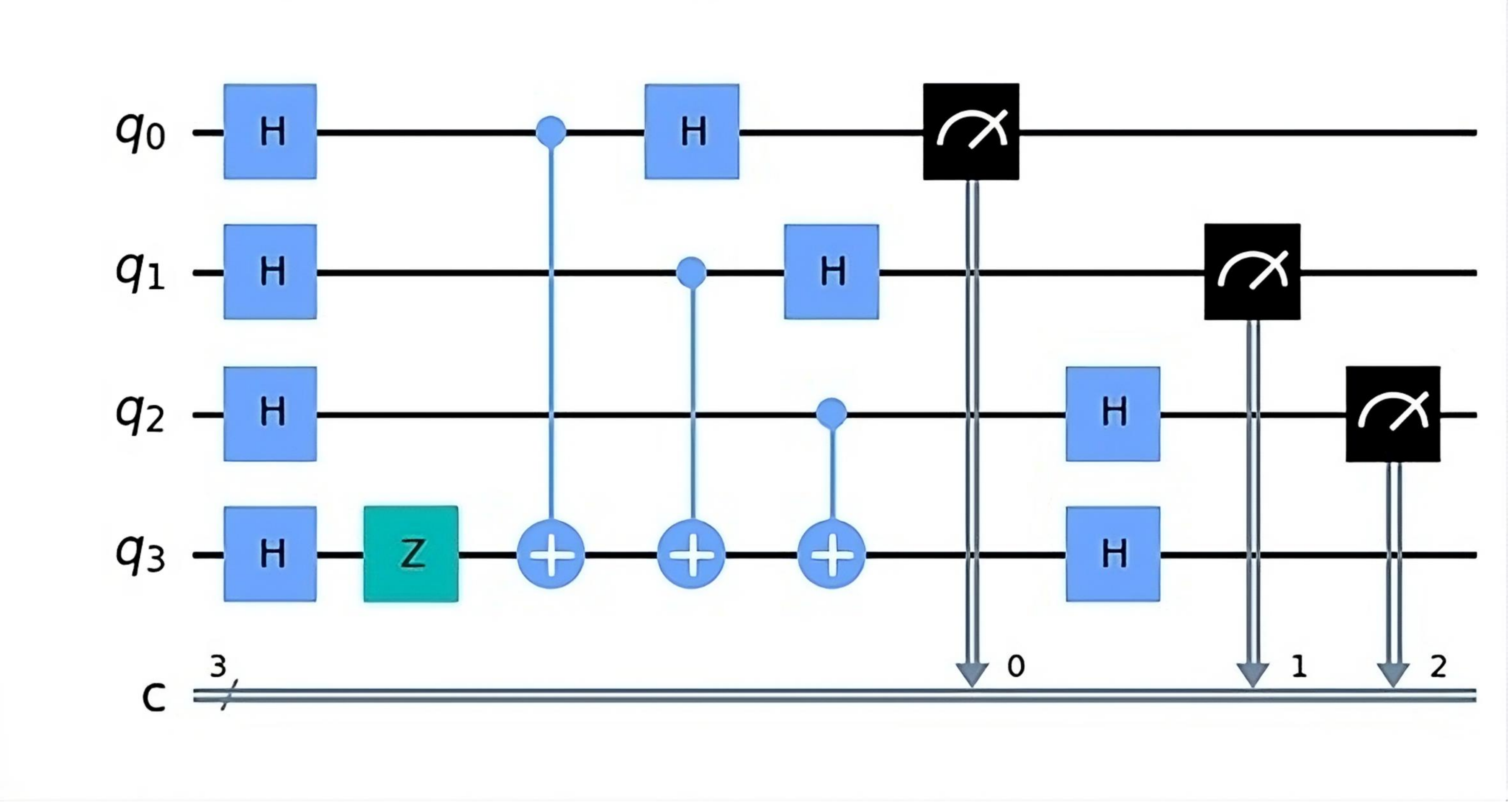


Key Concepts in Quantum Computing



Quantum circuits and Quantum Algorithm

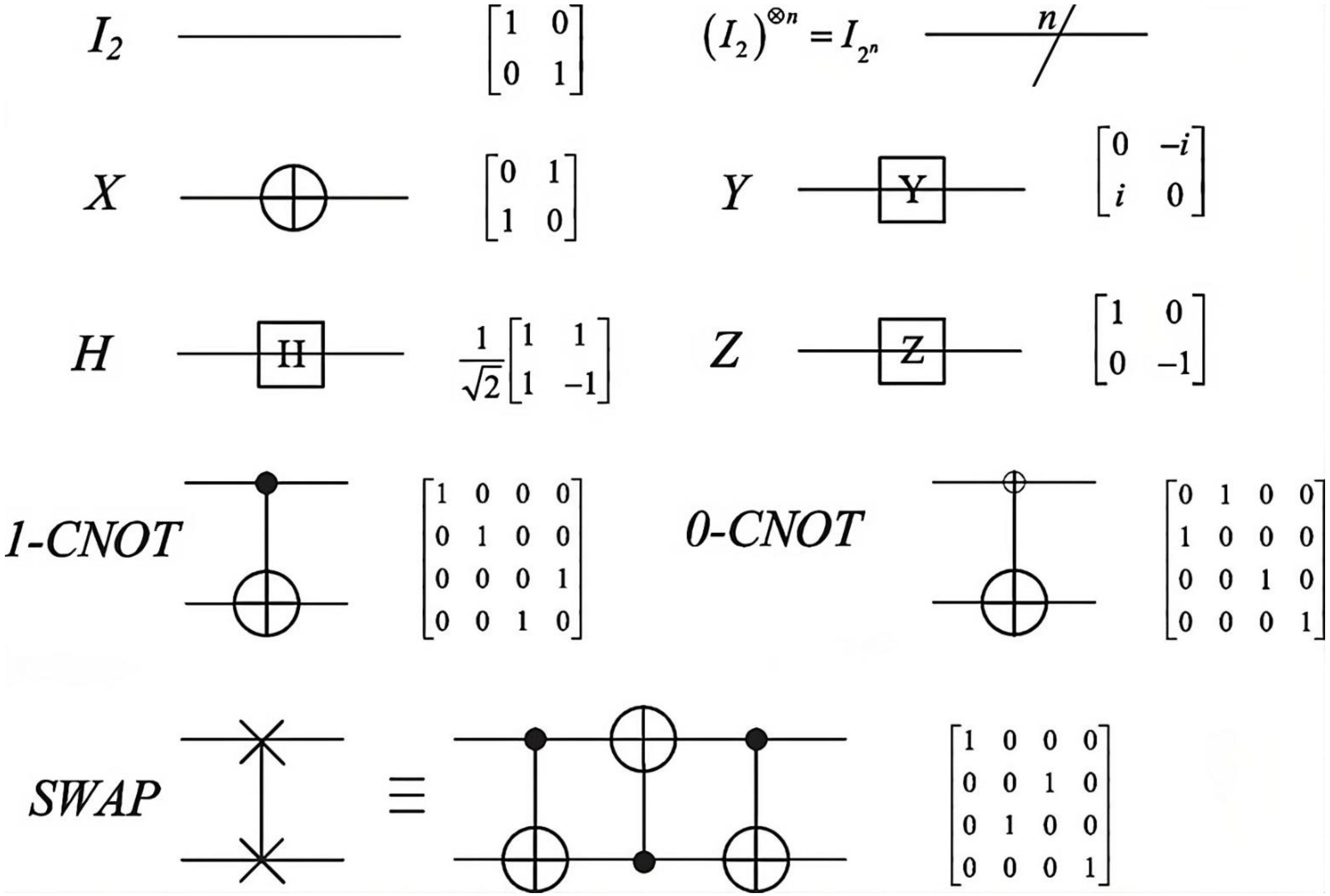
- In the *quantum circuit* model, the wires represent qubits and the gates represent both unitary operations and measurements.



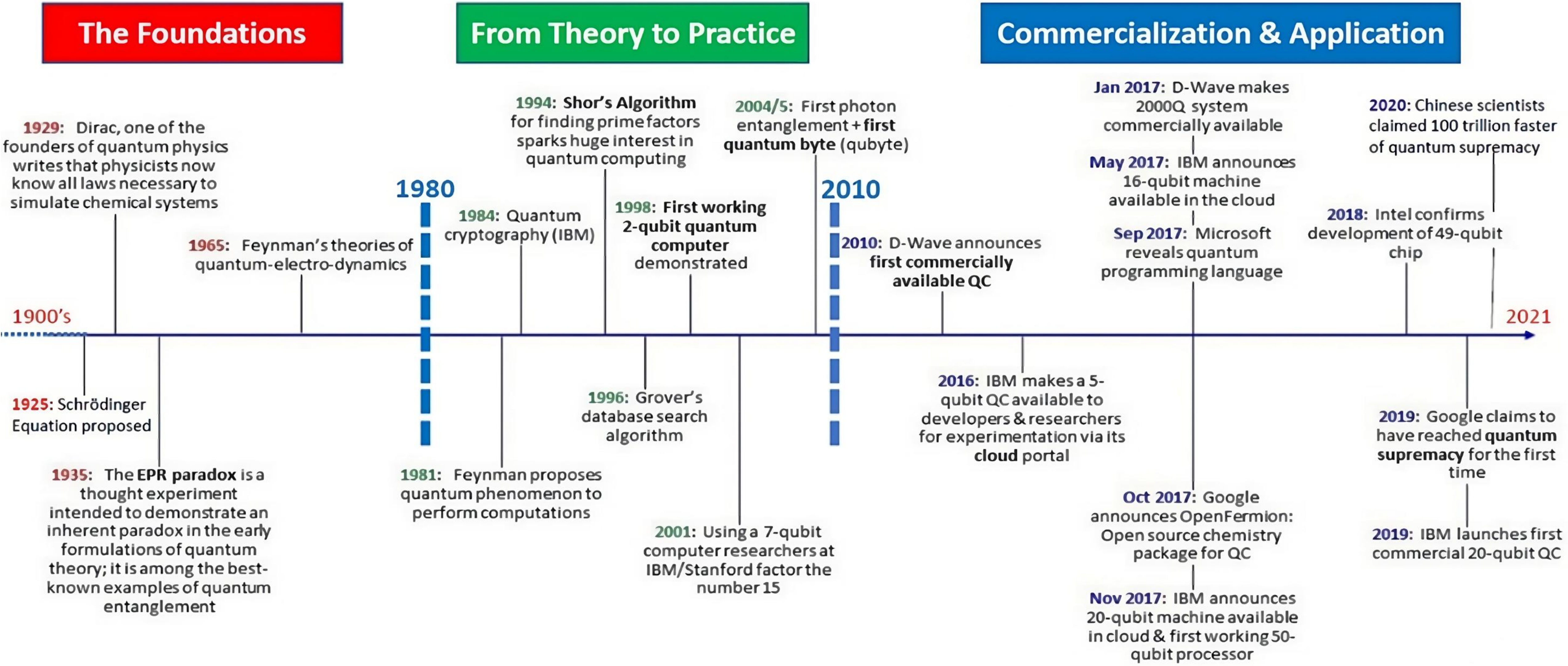
مرکز تحقیقات
فناوری‌های
کوانتومی ایران

Quantum circuits and Quantum Algorithm

- In the *quantum circuit* model, the wires represent qubits and the gates represent both unitary operations and measurements.



Quantum physics timeline



The Foundations

From Theory to Practice

Commercialization & Application

1900's

1925: Schrödinger Equation proposed

1929: Dirac, one of the founders of quantum physics writes that physicists now know all laws necessary to simulate chemical systems

1935: The **EPR paradox** is a thought experiment intended to demonstrate an inherent paradox in the early formulations of quantum theory; it is among the best-known examples of quantum entanglement

1965: Feynman's theories of quantum-electro-dynamics

1980

1981: Feynman proposes quantum phenomenon to perform computations

1984: Quantum cryptography (IBM)

1994: **Shor's Algorithm** for finding prime factors sparks huge interest in quantum computing

1996: Grover's database search algorithm

1998: **First working 2-qubit quantum computer** demonstrated

2001: Using a 7-qubit computer researchers at IBM/Stanford factor the number 15

2004/5: First photon entanglement + **first quantum byte** (qubyte)

2010

2010: D-Wave announces **first commercially available QC**

2016: IBM makes a 5-qubit QC available to developers & researchers for experimentation via its **cloud** portal

Jan 2017: D-Wave makes 2000Q system commercially available

May 2017: IBM announces 16-qubit machine available in the cloud

Sep 2017: Microsoft reveals quantum programming language

Oct 2017: Google announces OpenFermion: Open source chemistry package for QC

Nov 2017: IBM announces 20-qubit machine available in cloud & first working 50-qubit processor

2018: Intel confirms development of 49-qubit chip

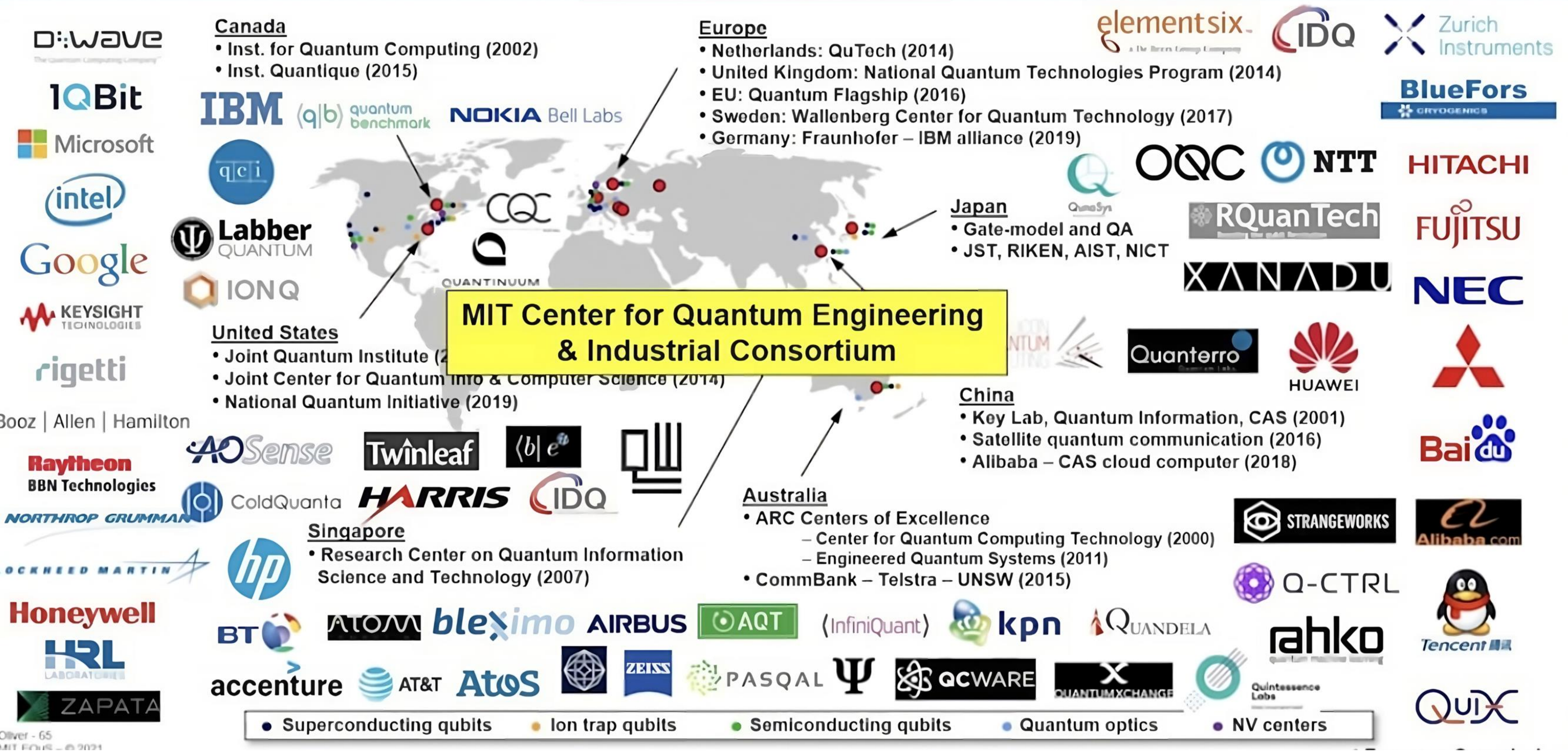
2019: Google claims to have reached **quantum supremacy** for the first time

2019: IBM launches first commercial 20-qubit QC

2020: Chinese scientists claimed 100 trillion faster of quantum supremacy

2021

Which companies



Canada
- Inst. for Quantum Computing (2002)
- Inst. Quantique (2015)

Europe
- Netherlands: QuTech (2014)
- United Kingdom: National Quantum Technologies Program (2014)
- EU: Quantum Flagship (2016)
- Sweden: Wallenberg Center for Quantum Technology (2017)
- Germany: Fraunhofer – IBM alliance (2019)

Japan
- Gate-model and QA
- JST, RIKEN, AIST, NICT

MIT Center for Quantum Engineering & Industrial Consortium

United States
- Joint Quantum Institute (2
- Joint Center for Quantum Info & Computer Science (2014)
- National Quantum Initiative (2019)

China
- Key Lab, Quantum Information, CAS (2001)
- Satellite quantum communication (2016)
- Alibaba – CAS cloud computer (2018)

Singapore
- Research Center on Quantum Information Science and Technology (2007)

Australia
- ARC Centers of Excellence
  - Center for Quantum Computing Technology (2000)
  - Engineered Quantum Systems (2011)
- CommBank – Telstra – UNSW (2015)

- ● Superconducting qubits
- ● Ion trap qubits
- ● Semiconducting qubits
- ● Quantum optics
- ● NV centers

Oliver - 65
MIT EQuS – © 2021

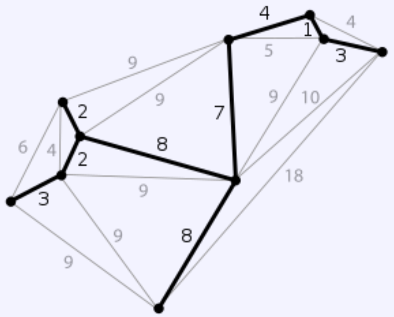
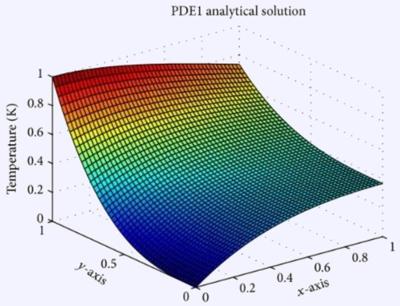
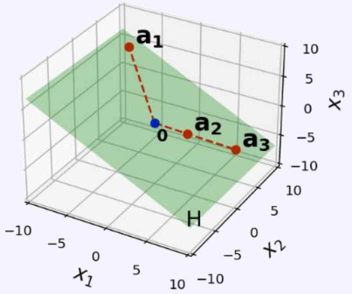
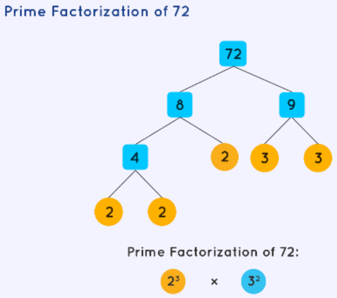
What does a quantum computer look like?

Chinese 76-qubit photon-based quantum computer

IonQ, ion-trap-based 32-qubit quantum computer

IBM 53-qubit superconductor-based quantum computer

The classification of different types of problems and their applications.

Type of problem		Useful for	Industrial applications
	Combinatorial optimization	Minimizing or maximizing an objective function, such as finding the most efficient allocation of resources or the shortest distance between a set of points (e.g., the Traveling Salesman Problem).	- Network optimization - Supply chain optimization - Portfolio optimization
	Differential equation	Modeling the behavior of complex systems involving fundamental physical laws (e.g., Navier-Stokes equations for fluid dynamics and chemistry).	- Computational fluid dynamics simulation - Molecular simulation for the discovery of specialized materials and drugs.
	Linear algebra	Machine learning techniques such as clustering, pattern matching, and principal component analysis, as well as support vector machines, which have widespread applications in industry.	- Risk management in finance - DNA sequence classification - Marketing
	Factoring	Cryptography and computer security, where today's most common protocols (such as RSA) rely on the feasibility (for classical computers) of factoring the product of two large prime numbers.	- Codebreaking and cryptanalysis (e.g., for government agencies).

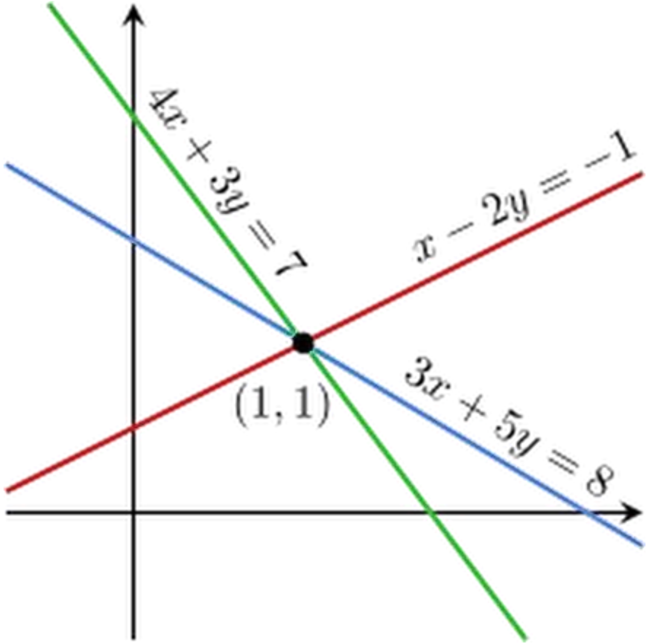
The impact of quantum computing on various industrial problems

	Automotive	Aerospace	Chemical Materials	Financial Services	Biological Sciences
					
Leading use cases of quantum computing being explored by industry.	• Traffic flow management • Automotive design optimization • Crash simulation • Battery manufacturing • Industrial efficiency • Supply chain optimization	• Air traffic control •Aircraft design optimization •Fleet, crew, and fuel optimization •Cargo loading optimization •Supply chain optimization	•Chemical reaction modeling and optimization •Battery manufacturing •Molecular simulation and discovery	•Risk management •Dynamic portfolio management •Derivatives pricing •Detection of financial data manipulation	•Biological target identification •Evidence synthesis for identification and optimization •Drug interaction detection •Disease diagnosis •Clinical trial optimization
The added value generated by quantum computing is estimated to be $700 billion.	$80 billion.	…	$300 billion.	$120 billion.	$200 billion.
Potential Benefits	•Efficient automotive production and sales •Designing better materials •Entering new markets	•Efficient aircraft and satellite manufacturing	•Entering new markets through new materials •Producing efficient products	•Better understanding of risk exposure •Improved portfolio returns •Lower fraud risk	•Faster drug production •Efficient drug development •Higher return on investment •Entering new markets

What is the HHL algorithm?

Applications of System of Linear Equations

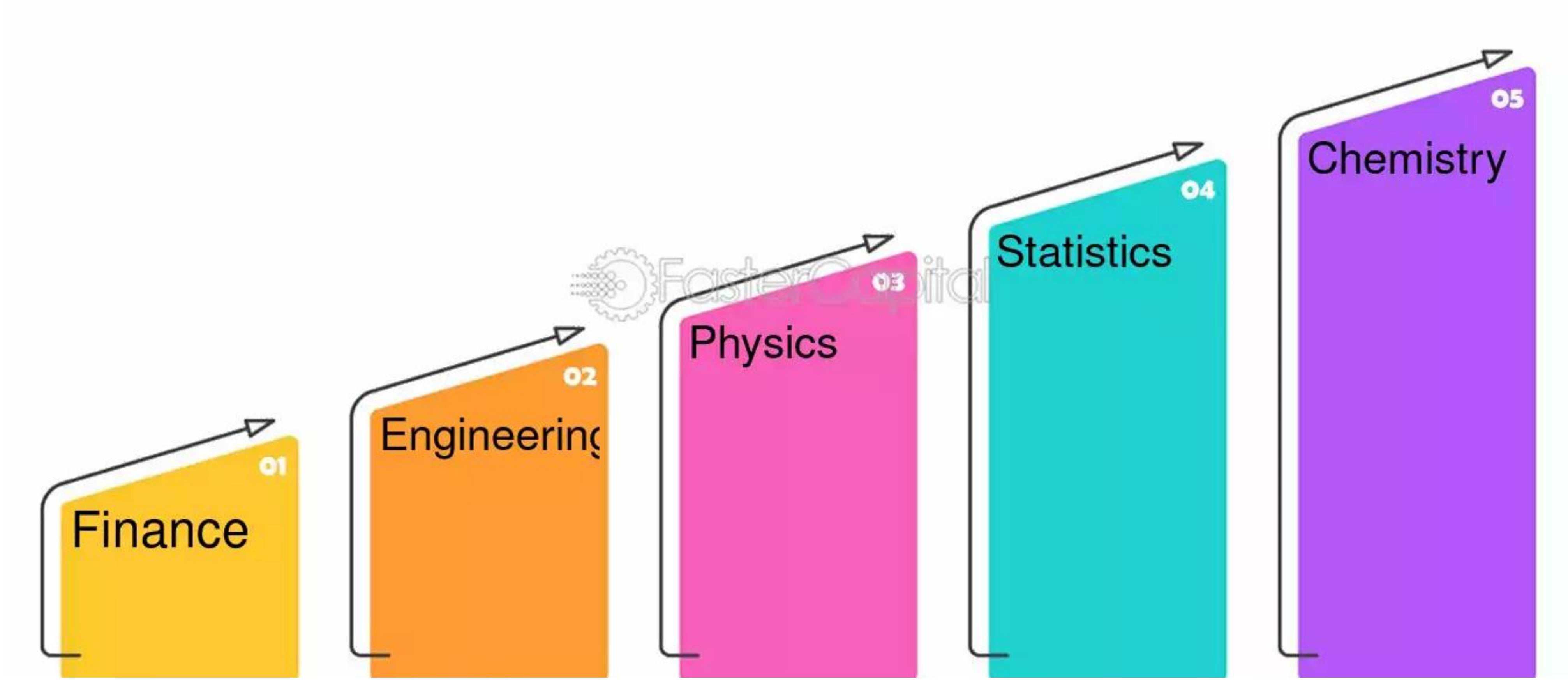
$$\begin{cases} 4x + 3y = 7 \\ x - 2y = -1 \\ 3x + 5y = 8 \end{cases}$$



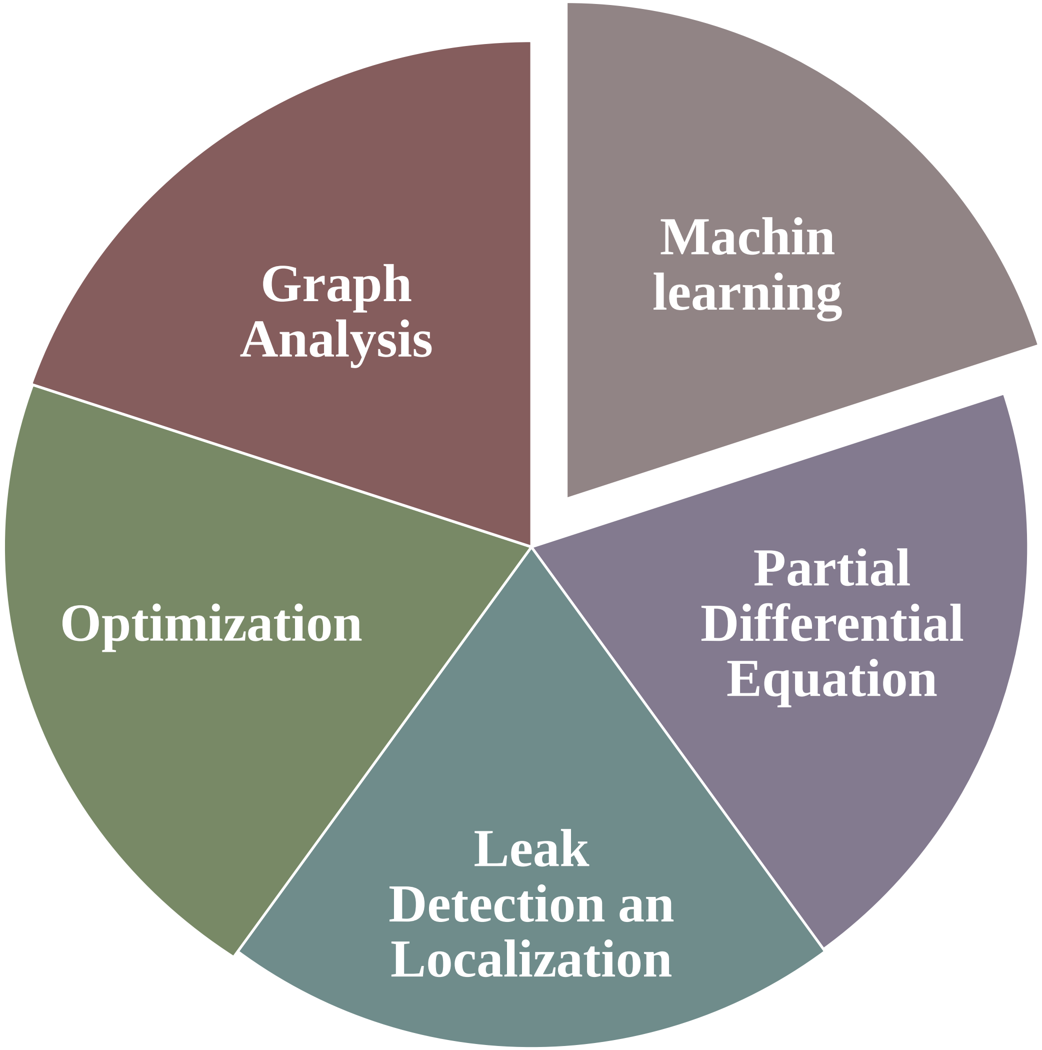
- ✓ As a result linear system problem (LSP) can be represented as the following:

$$\boldsymbol{A\vec{x} = \vec{b}}$$

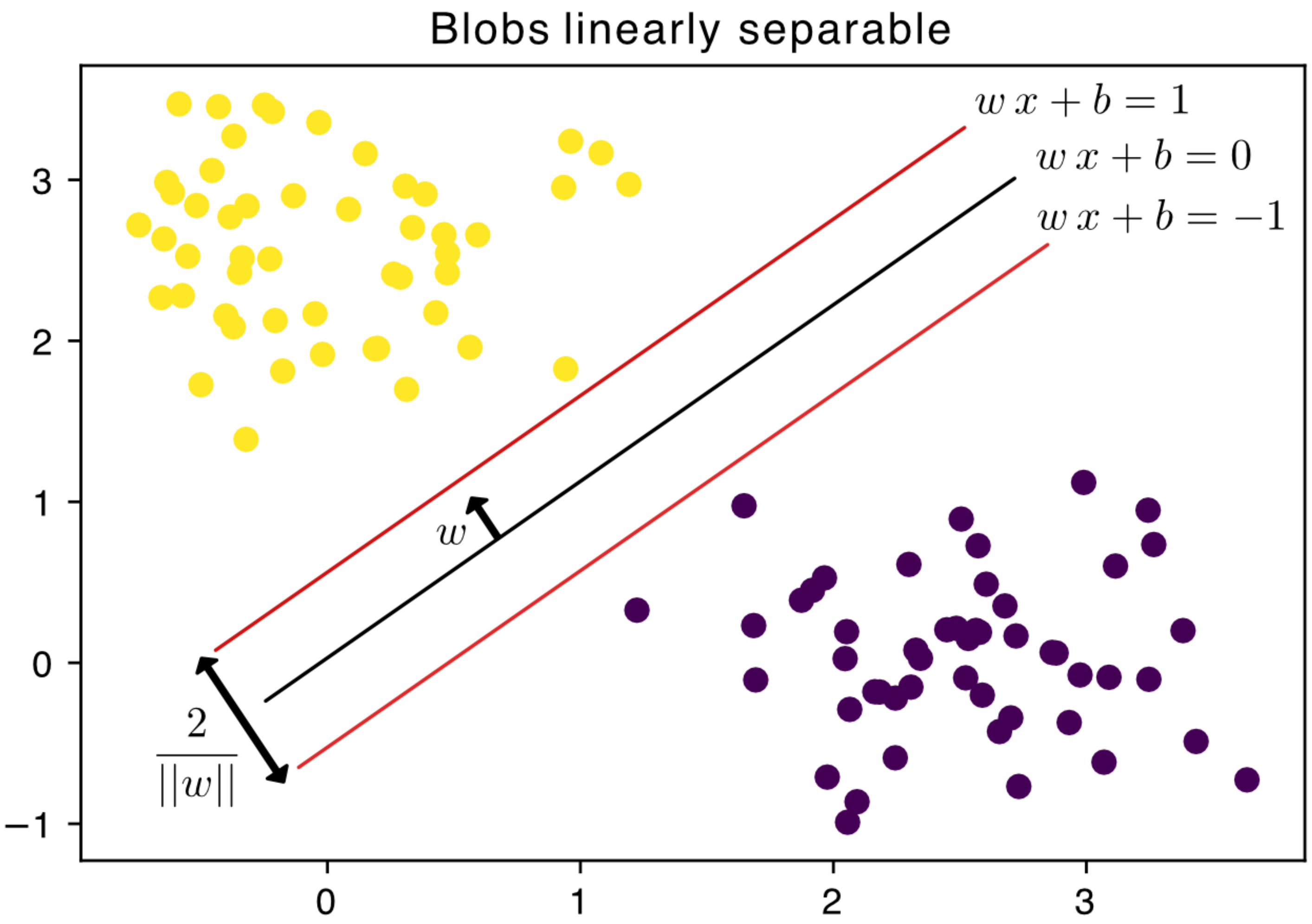
Applications of System of Linear Equations



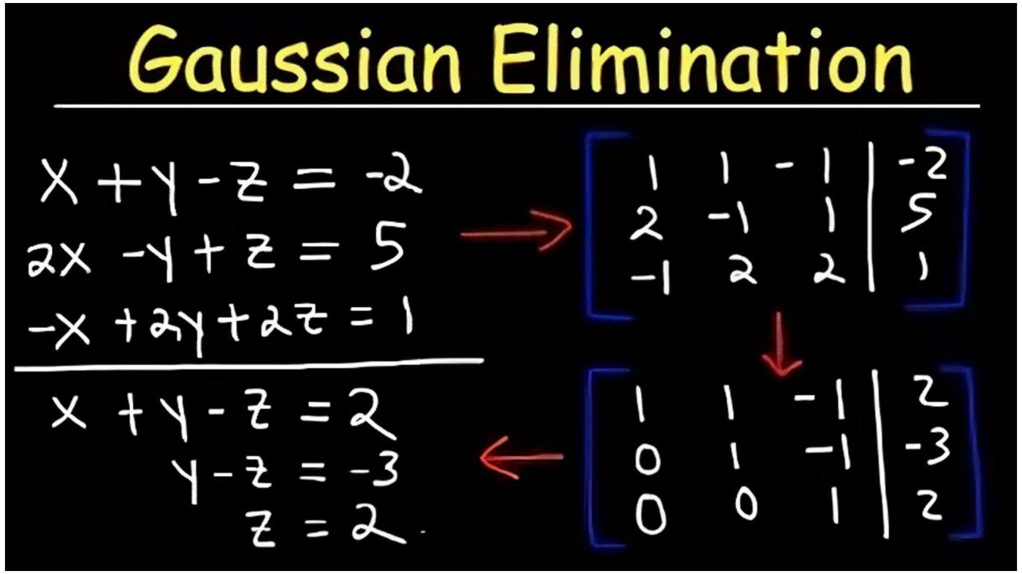
Applications of System of Linear Equations



Applications of System of Linear Equations



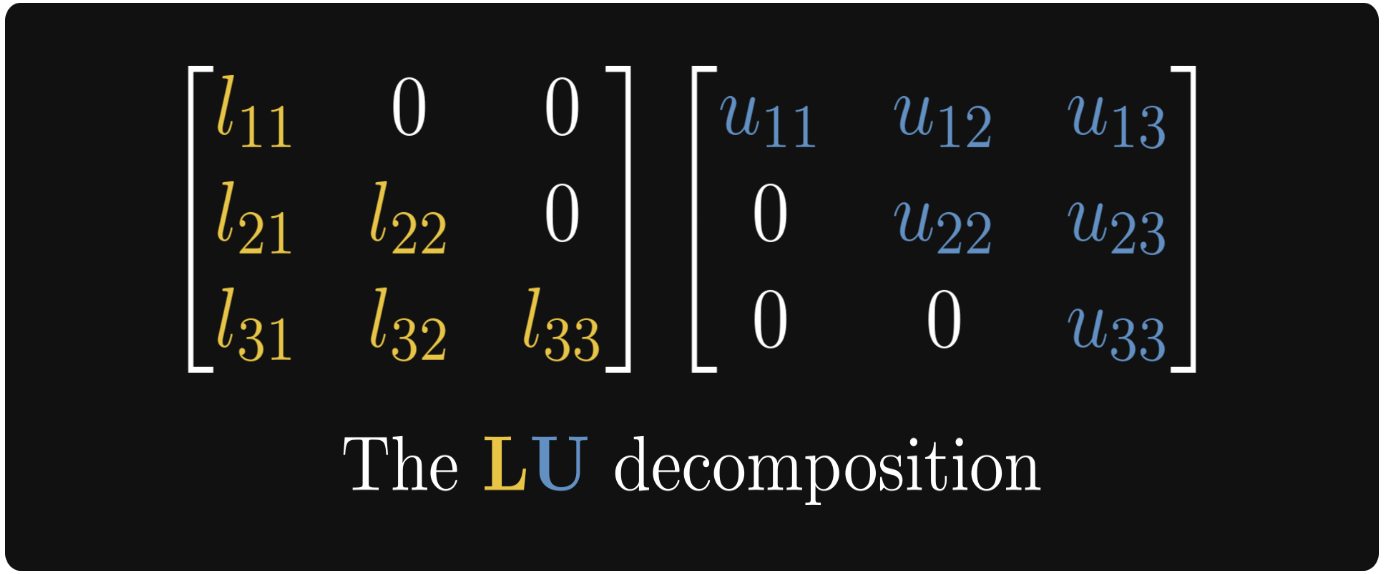
System of Linear Equations


Gaussian Elimination

$$\begin{aligned} x+y-z&=-2\\ 2x-y+z&=5\\ -x+2y+2z&=1 \end{aligned} \rightarrow \left[\begin{array}{ccc|c} 1 & 1 & -1 & -2\\ 2 & -1 & 1 & 5\\ -1 & 2 & 2 & 1 \end{array}\right]$$

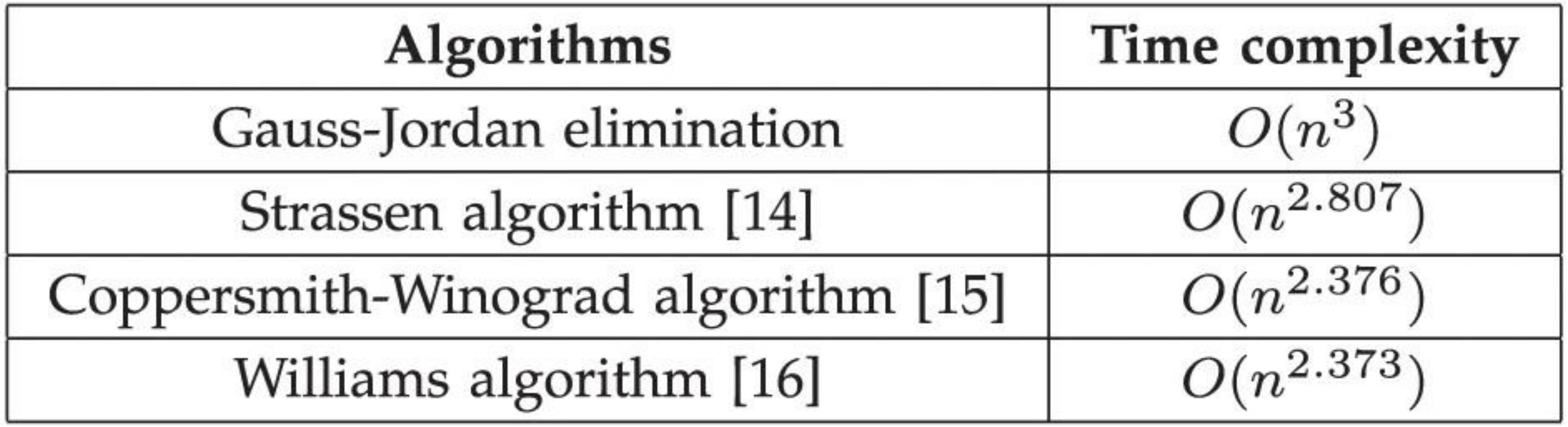
$$\downarrow$$

$$\begin{aligned} x+y-z&=2\\ y-z&=-3\\ z&=2 \end{aligned} \leftarrow \left[\begin{array}{ccc|c} 1 & 1 & -1 & 2\\ 0 & 1 & -1 & -3\\ 0 & 0 & 1 & 2 \end{array}\right]$$



$$\begin{bmatrix} l_{11} & 0 & 0\\ l_{21} & l_{22} & 0\\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13}\\ 0 & u_{22} & u_{23}\\ 0 & 0 & u_{33} \end{bmatrix}$$

The **LU** decomposition



Algorithms	Time complexity
Gauss-Jordan elimination	$O(n^3)$
Strassen algorithm [14]	$O(n^{2.807})$
Coppersmith-Winograd algorithm [15]	$O(n^{2.376})$
Williams algorithm [16]	$O(n^{2.373})$

Iterative Method for Solving System of Linear Equations

$$\sum_j E_{i,j} \boldsymbol{x}_j^{(k+1)} = \sum_j F_{i,j} \boldsymbol{x}_j^{(k)} + \boldsymbol{b}_i .$$

Various selections of the matrices *E* and *F* lead to different iterative methods.

$$\begin{aligned} E_{i,j} &= \frac{1}{\omega} D_{i,j} + L_{i,j} \\ F_{i,j} &= \left(\frac{1}{\omega} - 1\right) D_{i,j} - U_{i,j} \end{aligned}$$

$$D_{i,j} = \begin{cases} A_{i,j}, & \text{if } i = j. \\ 0, & \text{otherwise.} \end{cases}$$

$$U_{i,j} = \begin{cases} A_{i,j}, & \text{if } i < j. \\ 0, & \text{otherwise.} \end{cases}$$

$$L_{i,j} = \begin{cases} A_{i,j}, & \text{if } i > j. \\ 0, & \text{otherwise.} \end{cases}$$

The Harrow-Hassidim-Lloyd Algorithm (HHL)

Harrow-Hassidim-Lloyd (HHL)

- This is a quantum algorithm for solving **a system of linear equations**.
- This algorithm is designed based on **quantum computer**.
- This algorithm uses the quantum **phase estimation** and **Fourier transfer.**
- This algorithm provides an **exponential speedup.**

The HHL Algorithm

- ✓ A linear system problem (LSP) can be represented as the following:

$$\boldsymbol{A\vec{x} = \vec{b}}$$

- ✓ Where A is a $Nb * Nb$ Hermitian matrix.

$$\boldsymbol{\vec{x} = A^{-1}\vec{b}}$$

$$\begin{bmatrix} \mathbf{0} & \boldsymbol{A} \\ \boldsymbol{A}^{\dagger} & \mathbf{0} \end{bmatrix}$$

- ✓ For simplicity, it is assumed $Nb = 2^{\mathrm{nb}}$.

The HHL Algorithm Main Idea

- Since A is a Hermitian matrix, it has a spectral decomposition as follows:

$$A = \sum_{i=0}^{2^{n_b}-1} \lambda i|ui\rangle\langle ui| \quad ,\lambda_i \in R \qquad \rightarrow \qquad A^{-1} = \sum_{i=0}^{2^{n_b}-1} \lambda_i{}^{-1}|u_i\rangle\langle u_i|$$

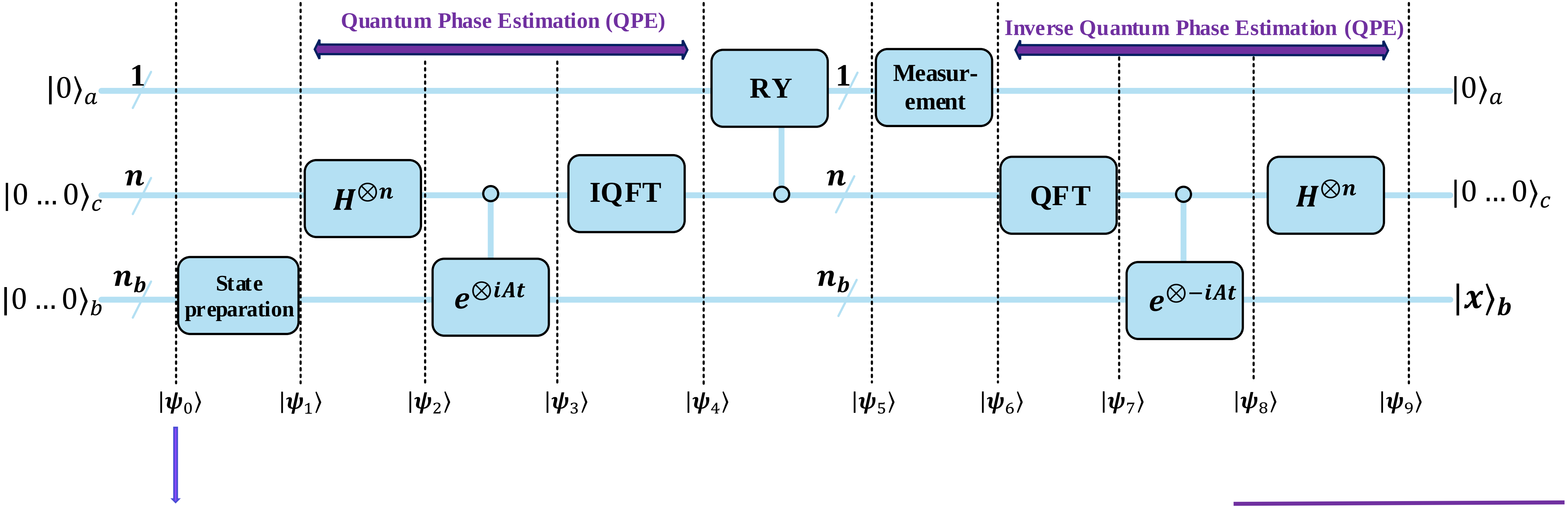
- Accordingly, the right side of the equation can be written as follows based on the eigenvalues:

$$|b\rangle = \sum_{j=0}^{2^{n_b}-1} b_j|u_j\rangle, \qquad \sum_{j=0}^{2^{n_b}-1} |b_j|^2 = 1$$

$$|x\rangle = A^{-1}|b\rangle = \sum_{i=0}^{2^{n_b}-1} \lambda_i{}^{-1}b_i|u_i\rangle \ , \qquad \sum_{i=0}^{2^{n_b}-1} |\lambda_i{}^{-1}b_j|^2 = 1$$

The HHL Algorithm

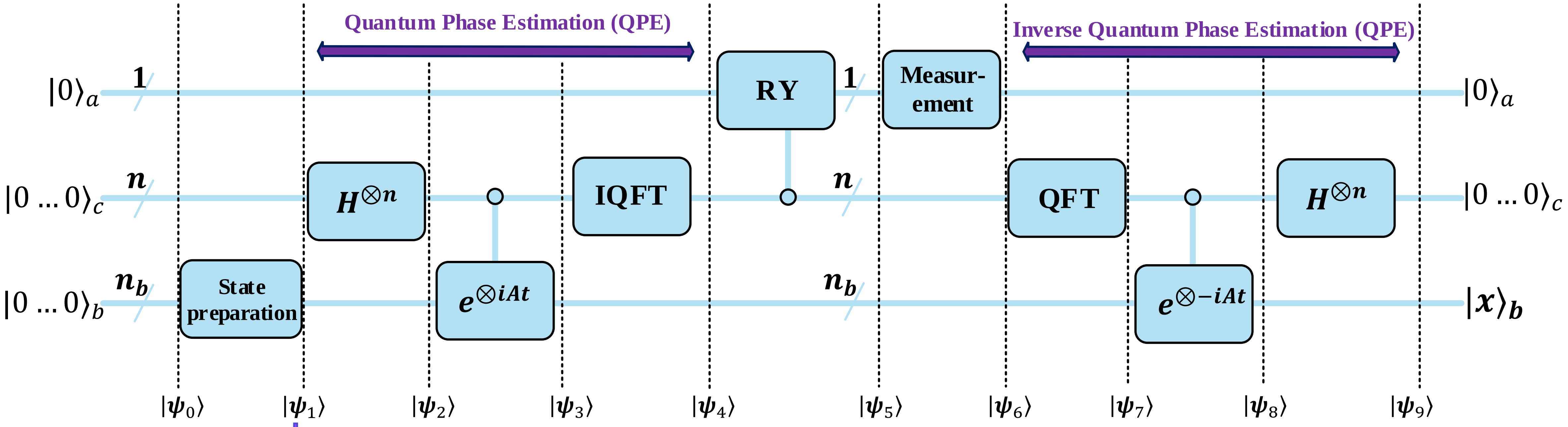
- ✓ The below figure shows the schematic of the HHL.



$$|\psi_0\rangle = |0\ldots0\rangle_b|0\ldots0\rangle_c|0\rangle_a = |0\rangle^{\otimes n_b}|0\rangle^{\otimes n_c}|0\rangle$$

The HHL Algorithm (Loading)

$$\vec{b} = \begin{pmatrix}\beta_0\\ \beta_1\\ \vdots\\ \beta_{n_b-1}\end{pmatrix} \Leftrightarrow \beta_0\,|0\rangle + \beta_1\,|1\rangle + \dots + \beta_{n_b-1}\,|n_b-1\rangle = |b\rangle$$



$$|\psi_1\rangle = |b\rangle_b|0\dots0\rangle_c|0\rangle_a$$

مرکز تحقیقات
فناوری‌های
کوانتومی ایران

The HHL Algorithm (Phase Estimation)

Therefore, in QPE, qubits of stability c are used to represent the phase information U and the accuracy depends on the number of qubits n.

$$\boldsymbol{U|b\rangle = e^{2\pi i\emptyset}|b\rangle}$$

Since the relationship between U and A is U= e^{iAt}, assuming that $|b\rangle$ is the eigenvector of U:

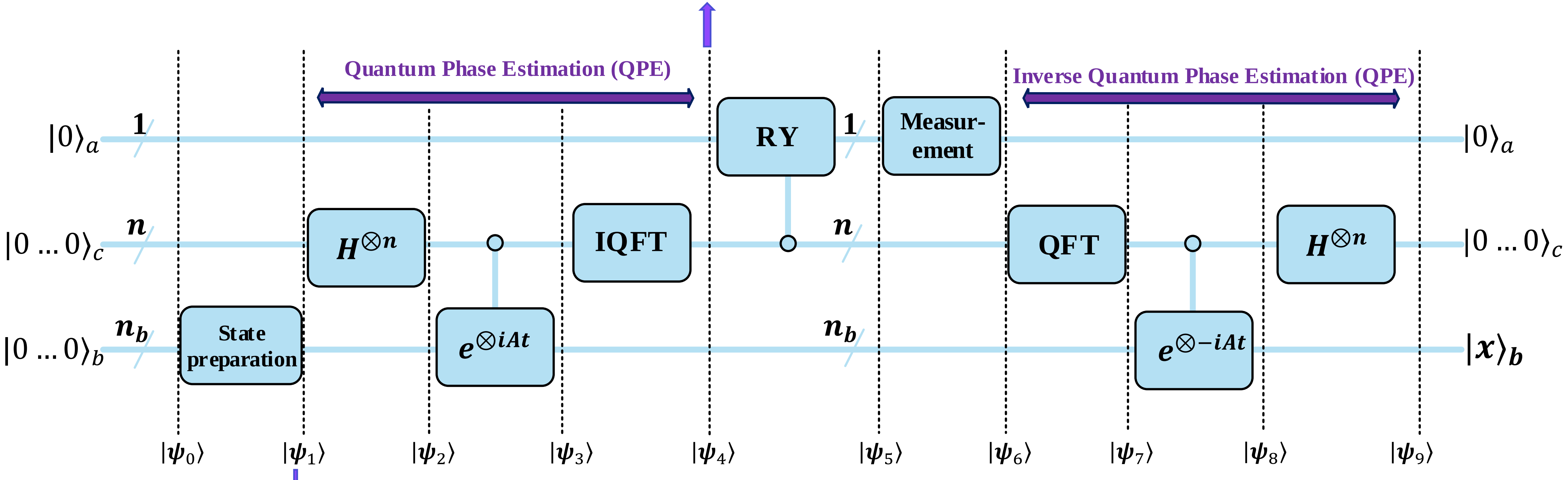
$$\boldsymbol{U|u_j\rangle = e^{i\lambda_j t}|u_j\rangle}$$

By equalizing $2\pi i\emptyset$ and $i\lambda_j t$ angle $\emptyset = \lambda_j t/2\pi$ as a result by considering $\tilde{\lambda}_j = \lambda_j t/2\pi$:

$$|\boldsymbol{\psi}_4\rangle = |u_j\rangle|\tilde{\lambda}_j\rangle|0\rangle_a$$

The HHL Algorithm (Phase Estimation)

$$|\boldsymbol{\psi}_4\rangle = \sum_{j=0}^{2^{n_b-1}} b_j|u_j\rangle|\tilde{\lambda}_j\rangle\,|0\rangle_a$$



$$|\boldsymbol{\psi}_1\rangle = |b\rangle_b|0 \dots 0\rangle_c|0\rangle_a$$

The HHL Algorithm (Eigen Value Inversion)

By applying the rotation gate along the y axis, we have:

$$|\boldsymbol{\psi}_5\rangle = \sum_{j=0}^{2^{n_b-1}} b_j|u_j\rangle|\tilde{\lambda}_j\rangle \left(\sqrt{1-\frac{C^2}{\tilde{\lambda}_j^{\,2}}}\,|0\rangle_a + \frac{C}{\tilde{\lambda}_j}|1\rangle_a\right)$$

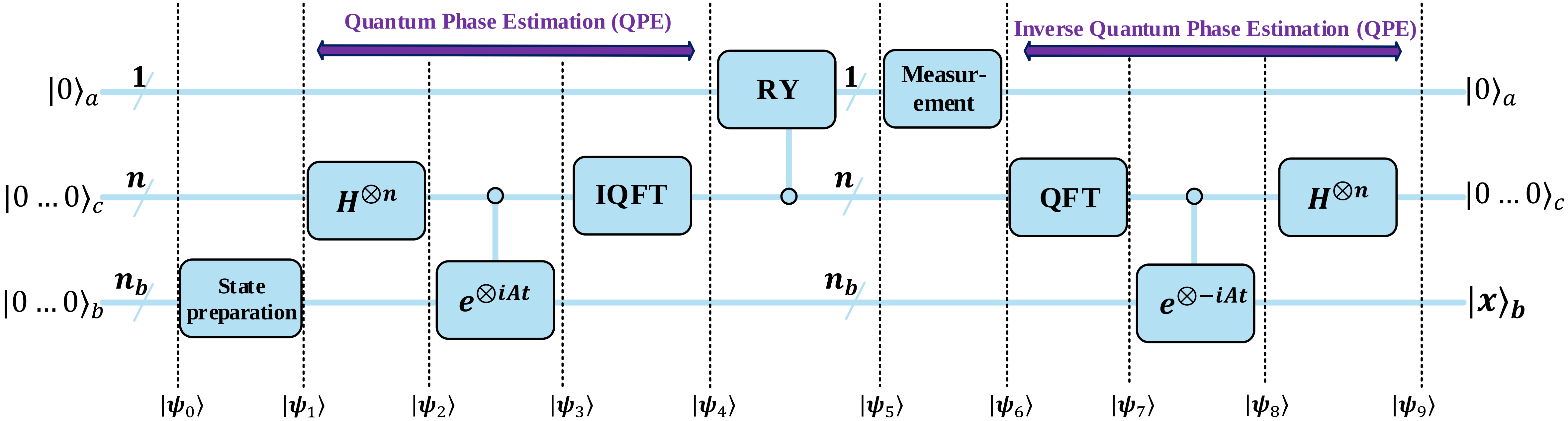
If the ancilla qubit is $|0\rangle$, the result is discarded and the calculation is repeated until the measurement is $|1\rangle$. Therefore, the desired final wave function is as follows:

$$|\boldsymbol{\psi}_6\rangle = \frac{1}{\sqrt{\sum_{j=0}^{2^{n_b-1}} \left|\frac{b_j C}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^{n_b-1}} b_j|u_j\rangle|\tilde{\lambda}_j\rangle \frac{C}{\tilde{\lambda}_j}|1\rangle_a$$

The HHL Algorithm (Eigen Value Inversion)

$$|\psi_4\rangle = \sum_{j=0}^{2^{n_b}-1} b_j |u_j\rangle |\tilde{\lambda}_j\rangle\, |0\rangle_a$$

$$|\psi_6\rangle = \frac{1}{\sqrt{\sum_{j=0}^{2^{n_b}-1} \left|\frac{b_j C}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^{n_b}-1} b_j |u_j\rangle |\tilde{\lambda}_j\rangle \frac{C}{\tilde{\lambda}_j} |1\rangle_a$$



$$|\psi_1\rangle = |b\rangle_b |0 \ldots 0\rangle_c |0\rangle_a$$

The HHL Algorithm (Uncommuted Step)

Apply this stage according to the entanglement of the b and $|\tilde{\lambda}_j\rangle$:

$$|\boldsymbol{\psi}_9\rangle = \frac{1}{2^{\frac{n}{2}}\sqrt{\sum_{j=0}^{2^{n_b}-1}\left|\frac{b_j}{\lambda_j}\right|^2}}|x\rangle|0\rangle_0^{\otimes n}|1\rangle_a$$

Since:

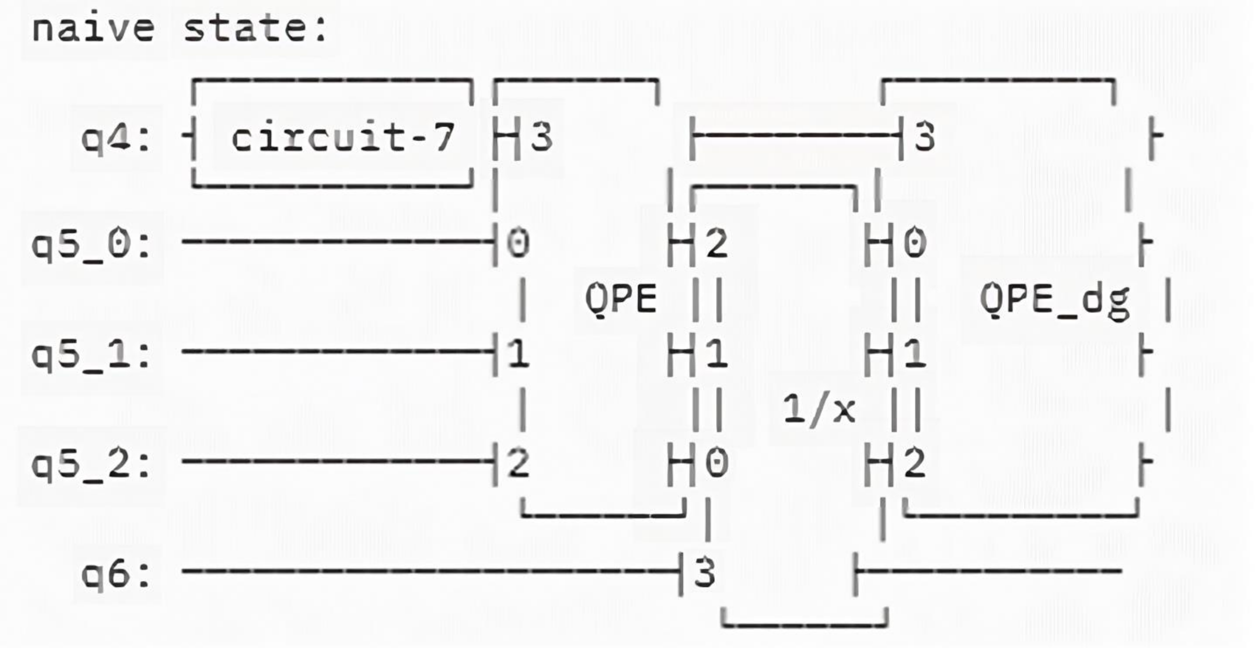
$$\sum_{i=0}^{2^{n_b}-1}\left|\lambda_i{}^{-1}b_i\right| = 1$$

As a result:

$$|\boldsymbol{\psi}_9\rangle = |x\rangle|0\rangle_0^{\otimes n}|1\rangle_a$$

Implementation of the HHL algorithm on Qiskit platform

```
print('naive state:')
print(naive_hhl_solution.state)
```



```
naive state:
      ┌───────────┐┌──────┐        ┌─────────┐
  q4: ┤ circuit-7 ├┤3     ├────────┤3        ├
      └───────────┘│      │┌──────┐│         │
q5_0: ─────────────┤0     ├┤2     ├┤0        ├
                   │  QPE ││      ││  QPE_dg │
q5_1: ─────────────┤1     ├┤1     ├┤1        ├
                   │      ││  1/x ││         │
q5_2: ─────────────┤2     ├┤0     ├┤2        ├
                   └──────┘│      │└─────────┘
  q6: ─────────────────────┤3     ├───────────
                           └──────┘
```

```
print('classical Euclidean norm:', classical_solution.euclidean_norm)
print('naive Euclidean norm:', naive_hhl_solution.euclidean_norm)
```

```
classical Euclidean norm: 1.1858541225631423
naive Euclidean norm: 1.18585412256138
```

Implementation of the HHL algorithm on Qiskit platform

The accuracy of HHl algorithm has been investigated by solving two sets of equations.

First example: Solving a 2*2 system of equations:

```
from qiskit.algorithms.linear_solvers.numpy_linear_solver
import NumPyLinearSolver


matrix = np.array([[1, -1/3], [-1/3, 1]])
vector = np.array([1, 0])


naive_hhl_solution = MY_HHL().solve(matrix, vector)


classical_solution = NumPyLinearSolver().solve(matrix, vector / np.linalg.norm(vector))

print('classical state:', classical_solution.state)
```

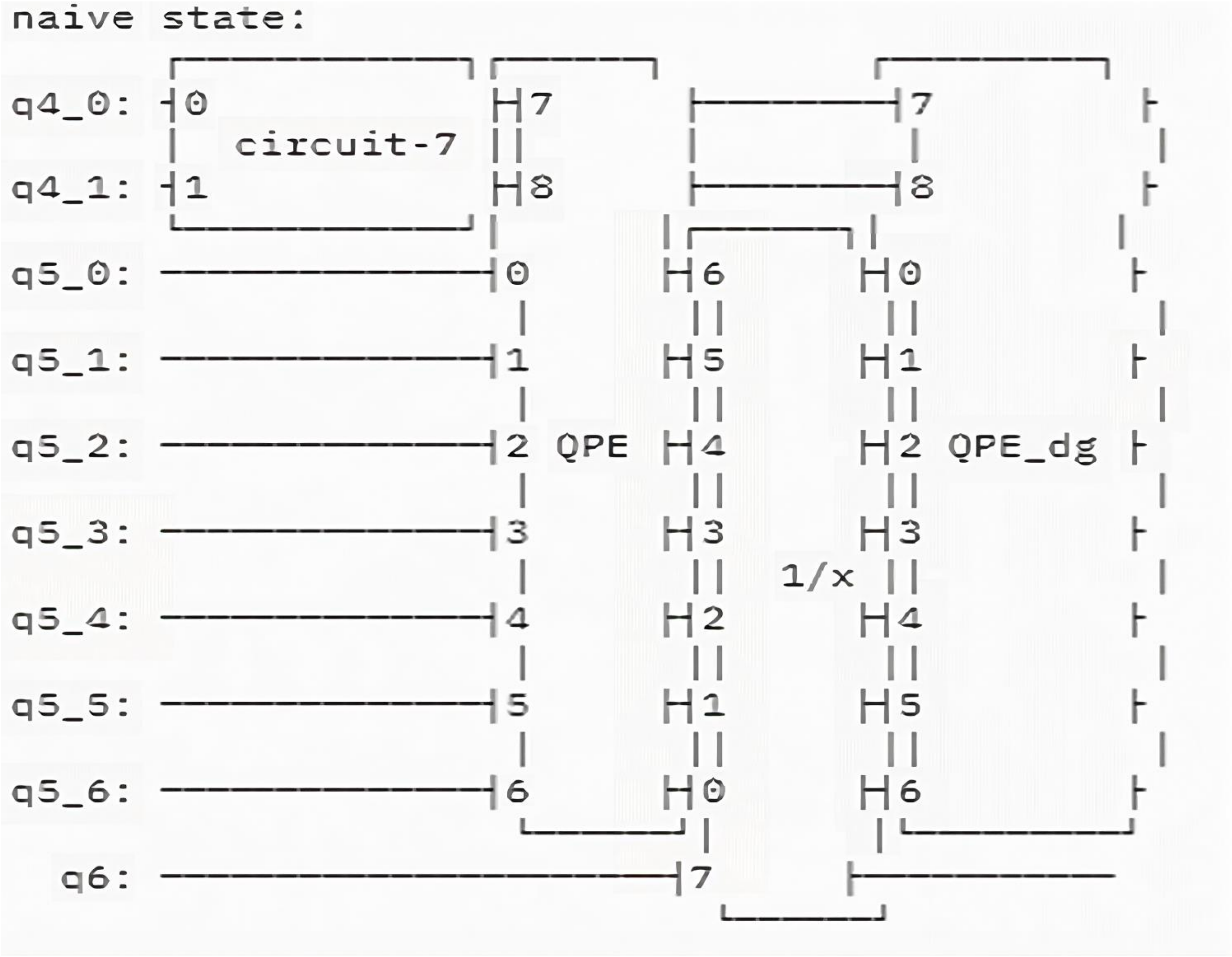
Implementation of the HHL algorithm on Qiskit platform

Second example: The solution of a 3x3 equation device has been investigated.

```
#define matrix and vector for solve

matrix = np.array([[0.0457968, -0.0309112, 0.0122894,0], [-0.0309112, 0.0520833, -0.0126651,0],[0.0122894, -0.0126651, 0.457968,0],[0,0,0,1]])
vector = np.array([0.000049, 0.0019531 , 0.00049,0])

naive_hhl_solution = MyHHL().solve(matrix, vector)

classical_solution = NumPyLinearSolver().solve(matrix, vector / np.linalg.norm(vector))
print('classical state:', classical_solution.state)
```

Implementation of the HHL algorithm on Qiskit platform

```
print('naive state:')
print(naive_hhl_solution.state)
```



```
naive state:
      ┌────────────┐┌──────┐        ┌─────────┐
q4_0: ┤0           ├┤7     ├────────┤7        ├
      │  circuit-7 ││      │        │         │
q4_1: ┤1           ├┤8     ├────────┤8        ├
      └────────────┘│      │┌──────┐│         │
q5_0: ──────────────┤0     ├┤6     ├┤0        ├
                    │      ││      ││         │
q5_1: ──────────────┤1     ├┤5     ├┤1        ├
                    │      ││      ││         │
q5_2: ──────────────┤2 QPE ├┤4     ├┤2 QPE_dg ├
                    │      ││      ││         │
q5_3: ──────────────┤3     ├┤3     ├┤3        ├
                    │      ││  1/x ││         │
q5_4: ──────────────┤4     ├┤2     ├┤4        ├
                    │      ││      ││         │
q5_5: ──────────────┤5     ├┤1     ├┤5        ├
                    │      ││      ││         │
q5_6: ──────────────┤6     ├┤0     ├┤6        ├
                    └──────┘│      │└─────────┘
  q6: ──────────────────────┤7     ├───────────
                            └──────┘
```

Implementation of the HHL algorithm on Qiskit platform

```
print('classical Euclidean norm:', classical_solution.euclidean_norm)
print('naive Euclidean norm:', naive_hhl_solution.euclidean_norm)
```

```
classical Euclidean norm: 38.42922665790783
naive Euclidean norm: 38.43989823369082
```

```
from qiskit.quantum_info import Statevector
naive_sv = Statevector(naive_hhl_solution.state).data
naive_full_vector = np.array([naive_sv[512], naive_sv[513],naive_sv[514], naive_sv[515]])
print('naive raw solution vector:', naive_full_vector)
```

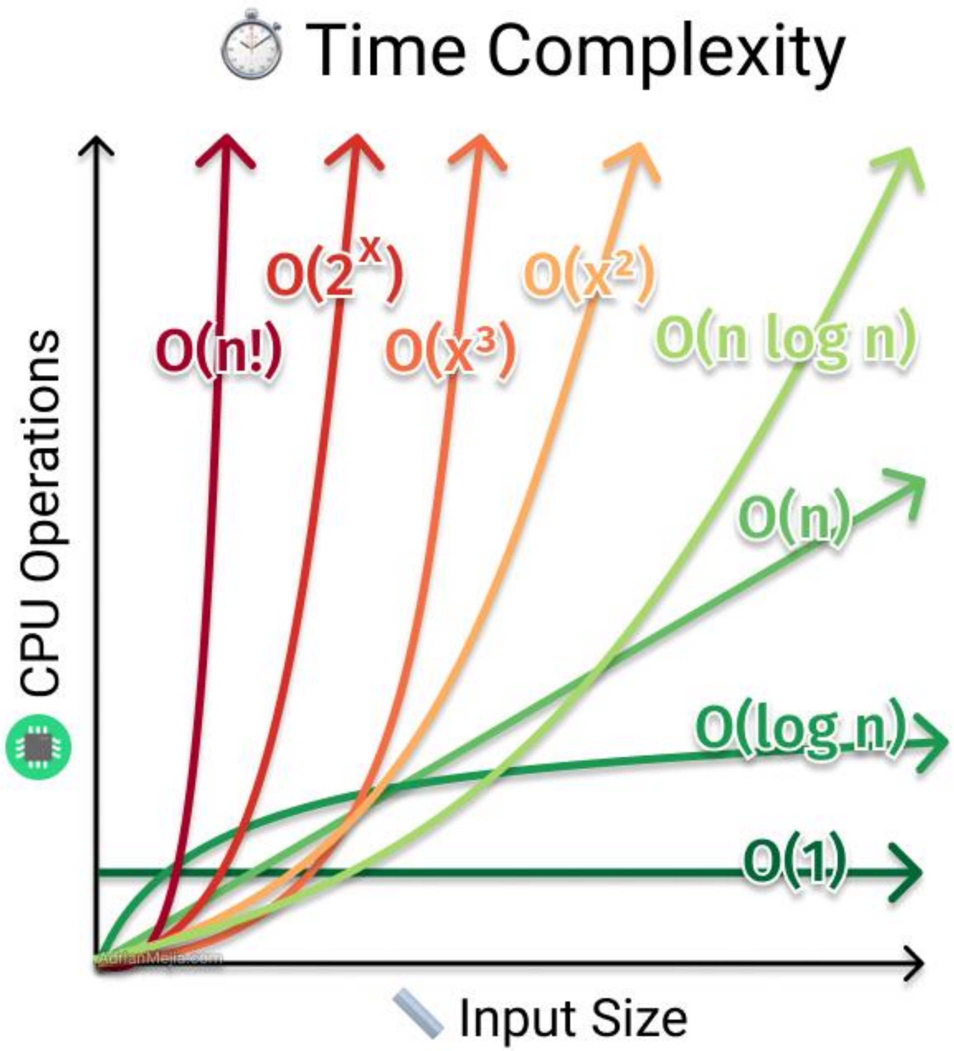
```
naive raw solution vector: [-2.73876640e-01-2.73876640e-01j -4.00967262e-01-4.00967262e-01j
                            -1.04173614e-02-1.04173614e-02j 3.60496340e-16-4.84428988e-17j  ]
```

```
print('full naive solution vector:', -
naive_hhl_solution.euclidean_norm*naive_full_vector/np.linalg.norm(naive_full_vector))
print('classical state:', classical_solution.state)
```

```
full naive solution vector: [ 2.16760881e+01 3.17347318e+01 8.24486687e-01 -2.85316427e-14]
classical state: [21.70665637 31.7008716 0.82539112 0. ]
```

Time Complexity of HHL and classical algorithm

Comparing the solving speed of HHL algorithm with increasing problem size (n)



Matrix size	# classic operation	#Quantum operation
10*10	$10^3 = 1000$	$\log_{10} 10$ =1
20*20	$20^3 = 8000$	$\log_{10} 20$ =1.3
30*30	$30^3 = 1000$	$\log_{10} 30$ =1.47
40*40	$40^3 = 1000$	$\log_{10} 40$ =1.6
50*50	$50^3 = 1000$	$\log_{10} 50$ =1.69
100*100	$100^3 = 1M$	$\log_{10} 100$ =2
1000*1000	$1000^3 = 1G$	$\log_{10} 1000$ =3

Qiskit toolkit and quantum algorithm simulation

Quantum Computing Software Of 2024

1 Quantum Inspire

Best for diverse quantum backends compatibility

- Pricing upon request

2 Amazon Braket

Best for experimenting with quantum hardware

- From $0.075/user/month (billed based on usage).

3 Azure Quantum

Best for cloud quantum resources on Azure

- Pricing upon request

4 Rigetti Computing

Best for quantum-first hybrid systems

- Pricing upon request

5 IBM Quantum Cloud Software

Best for open-source quantum community collaboration

- Pricing upon request

6 Intel Quantum Simulator

Best for high-performance quantum simulations

- Pricing upon request

7 D-Wave Leap

Best for annealing-based quantum computing

- Pricing upon request

8 Xanadu PennyLane

Best for quantum neural networks

- Pricing upon request

9 Quantum AI

Best for Google's quantum research ecosystem

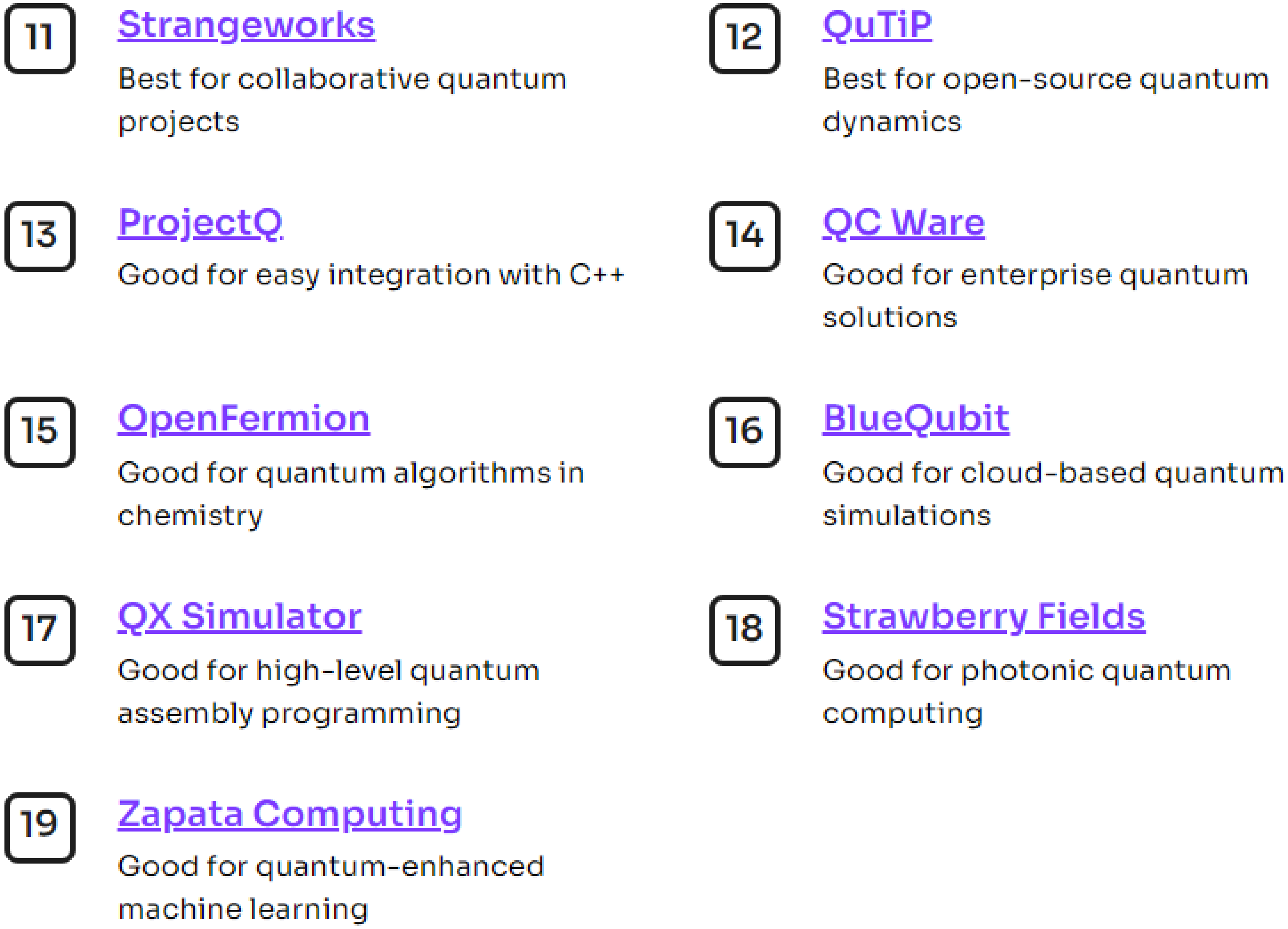
- Pricing upon request

10 qBraid

Best for quantum learning environments

- Pricing upon request

Quantum Computing Software Of 2024



11 **Strangeworks**
Best for collaborative quantum projects

12 **QuTiP**
Best for open-source quantum dynamics

13 **ProjectQ**
Good for easy integration with C++

14 **QC Ware**
Good for enterprise quantum solutions

15 **OpenFermion**
Good for quantum algorithms in chemistry

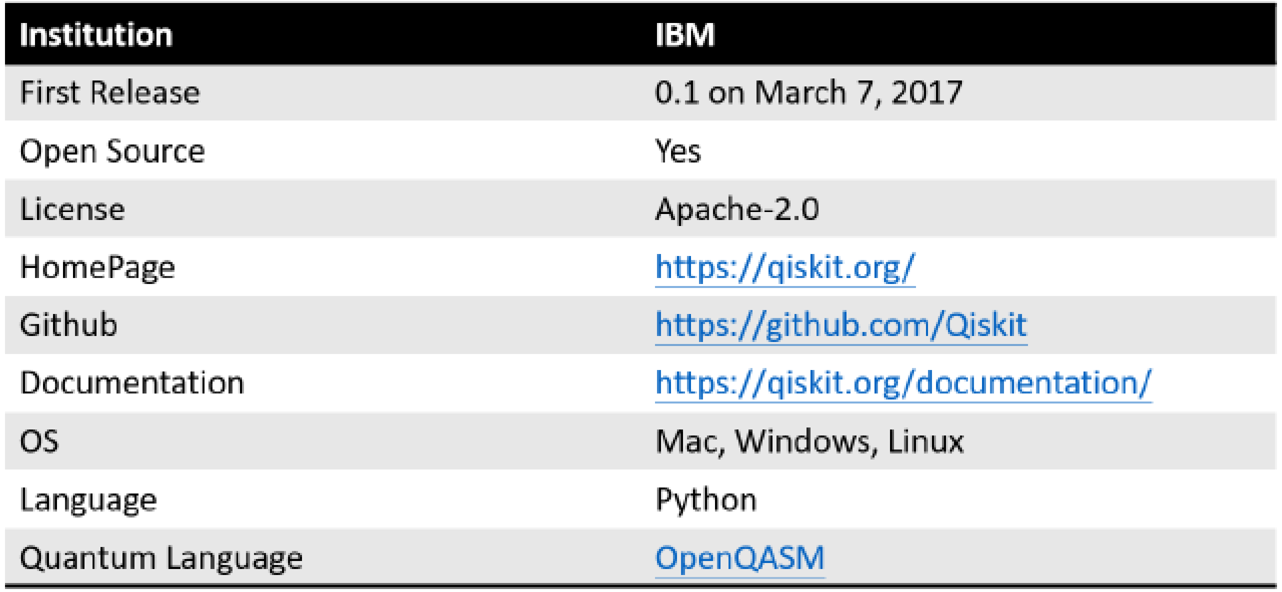
16 **BlueQubit**
Good for cloud-based quantum simulations

17 **QX Simulator**
Good for high-level quantum assembly programming

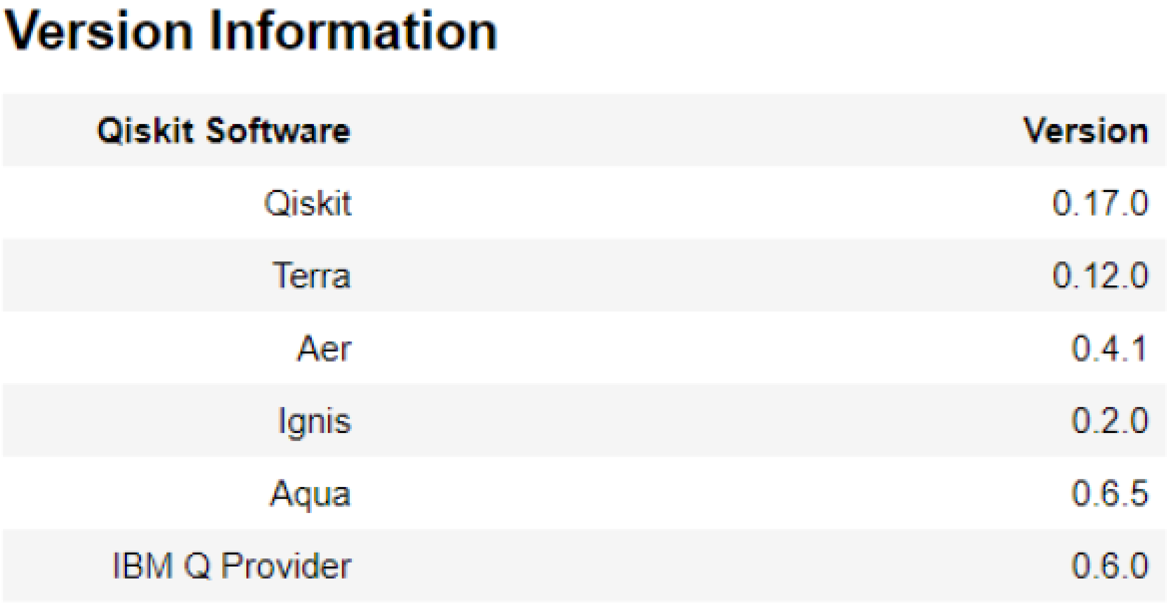
18 **Strawberry Fields**
Good for photonic quantum computing

19 **Zapata Computing**
Good for quantum-enhanced machine learning

Qiskit Overview



Institution	IBM
First Release	0.1 on March 7, 2017
Open Source	Yes
License	Apache-2.0
HomePage	https://qiskit.org/
Github	https://github.com/Qiskit
Documentation	https://qiskit.org/documentation/
OS	Mac, Windows, Linux
Language	Python
Quantum Language	OpenQASM



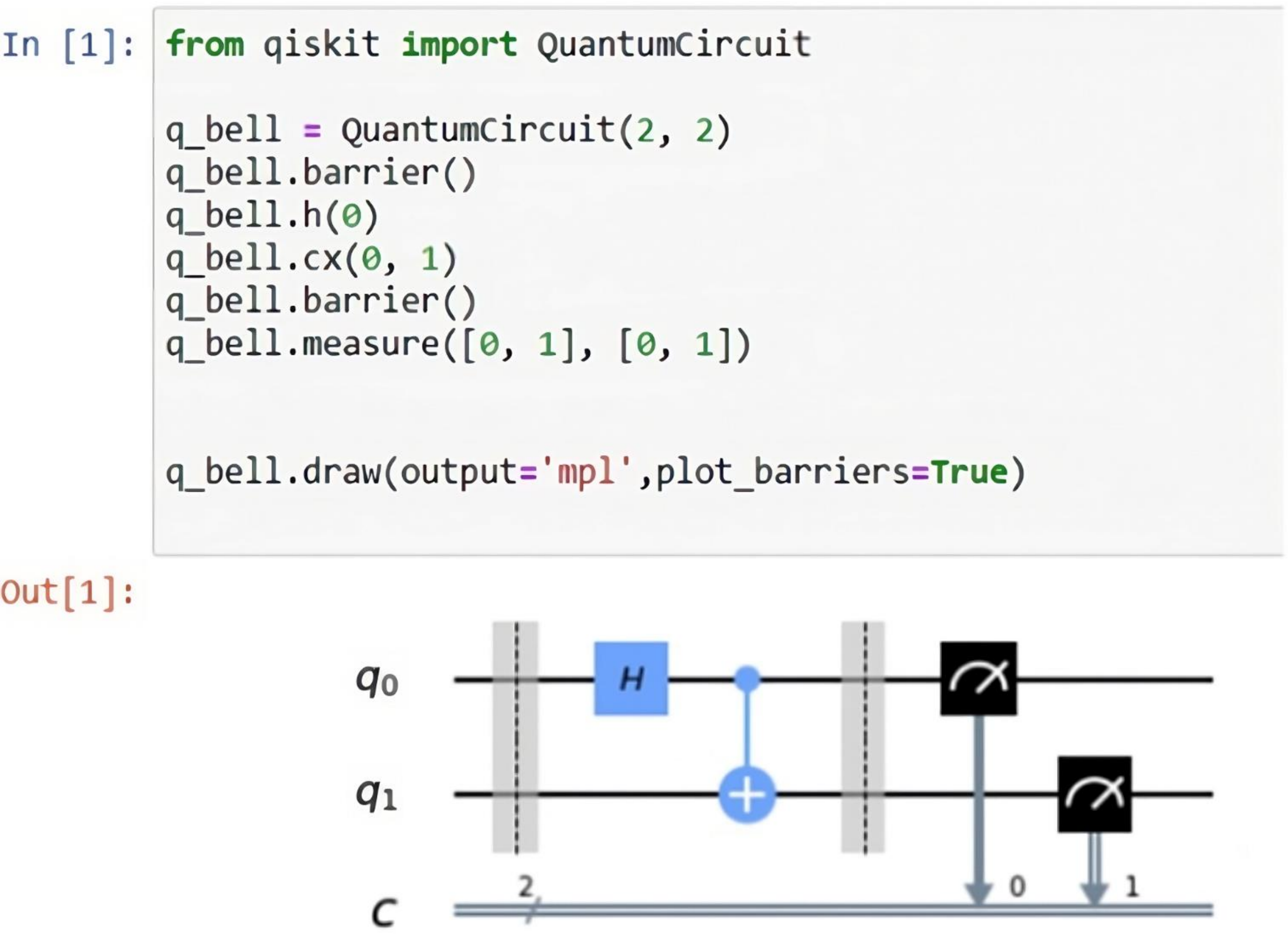
Version Information

Qiskit Software	Version
Qiskit	0.17.0
Terra	0.12.0
Aer	0.4.1
Ignis	0.2.0
Aqua	0.6.5
IBM Q Provider	0.6.0

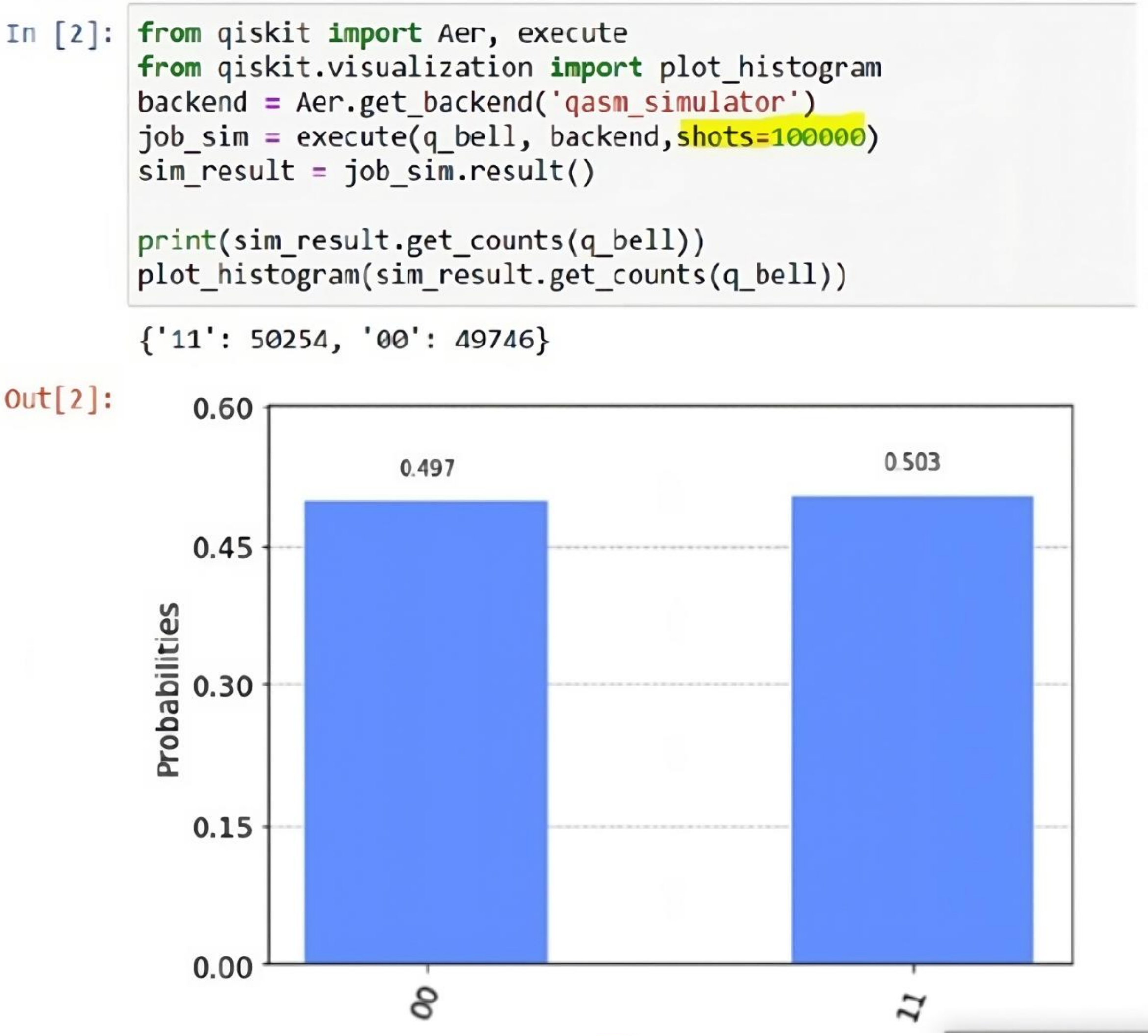
Installation

- Python / Anaconda (highly recommended for learning)
- `pip install qiskit`
- `pip install numpy`
- `pip install mathplot`
- `pip install histogram`

Qiskit Code Example



```
In [1]: from qiskit import QuantumCircuit

        q_bell = QuantumCircuit(2, 2)
        q_bell.barrier()
        q_bell.h(0)
        q_bell.cx(0, 1)
        q_bell.barrier()
        q_bell.measure([0, 1], [0, 1])


        q_bell.draw(output='mpl',plot_barriers=True)

Out[1]:
```



```
In [2]: from qiskit import Aer, execute
        from qiskit.visualization import plot_histogram
        backend = Aer.get_backend('qasm_simulator')
        job_sim = execute(q_bell, backend,shots=100000)
        sim_result = job_sim.result()

        print(sim_result.get_counts(q_bell))
        plot_histogram(sim_result.get_counts(q_bell))

        {'11': 50254, '00': 49746}
Out[2]:
```

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*: Cambridge university press, 2010.

[2] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65, 386 (1958).

[3] Wittek, P. Quantum Machine Learning: What Quantum Computing Means to Data Mining (Academic Press, New York, NY, USA, 2014).

[4] Bordley, R. F. 1998. Quantum Mechanical and Human Violations of Compound Probability Principles: Toward a Generalized Heisenberg Uncertainty Principle. *Operations Research* 46: 923-926.

[5] Busemeyer, J.R., Bruza, P.D.: Quantum Models of Cognition and Decision. Cambridge University Press, Cambridge (2012)

[6] H. J. Morrell Jr and H. Y. Wong, "Step-by-Step HHL Algorithm Walkthrough to Enhance the Understanding of Critical Quantum Computing Concepts," *arXiv preprint arXiv:2108.09004*, 2021.

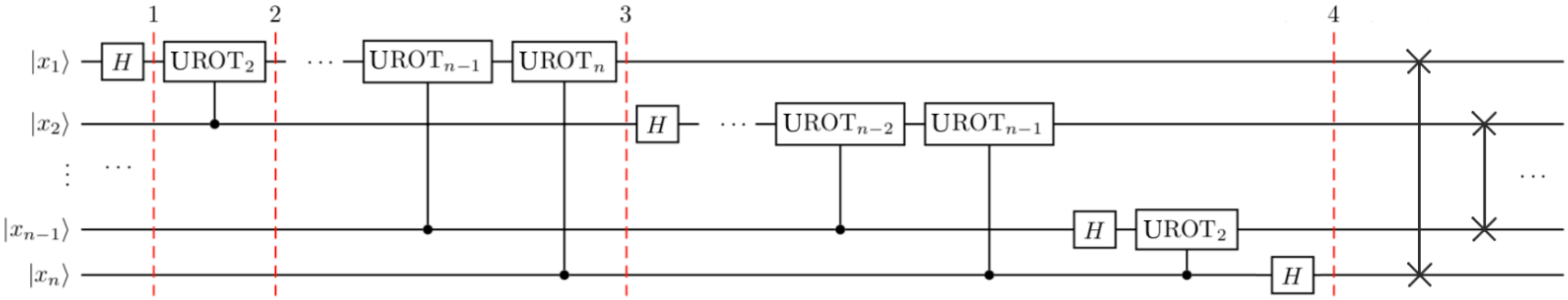
پایان

سپاس از توجه شما

مرکز تحقیقات
فناوری‌های
کوانتومی ایران

Quantum Fourier transform

- Given these two gates, a circuit that implements an n-qubit QFT is shown below.



- The circuit operates as follows. We start with an n-qubit input state $|x1x2\ldots xn\rangle$.

1. After the first Hadamard gate on qubit 1, the state is transformed from the input state to

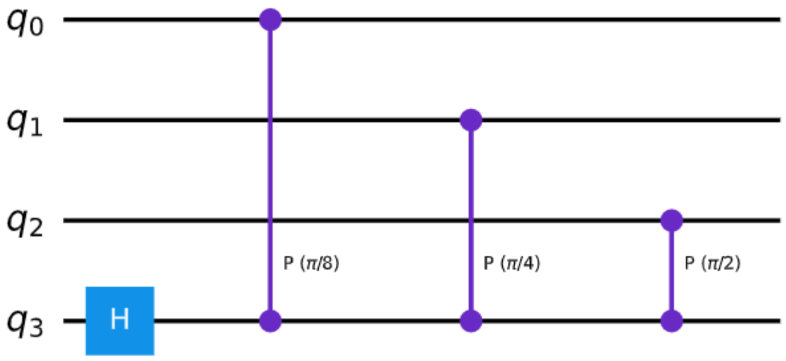
$$H_1|x_1x_2\ldots x_n\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3\ldots x_n\rangle$$

Quantum Fourier transform

```
def qft_rotations(circuit, n):
    if n == 0: # Exit function if circuit is empty
        return circuit
    n -= 1 # Indexes start from 0
    circuit.h(n) # Apply the H-gate to the most significant qubit
    for qubit in range(n):
        # For each less significant qubit, we need to do a
        # smaller-angled controlled rotation:
        circuit.cp(pi/2**(n-qubit), qubit, n)
```

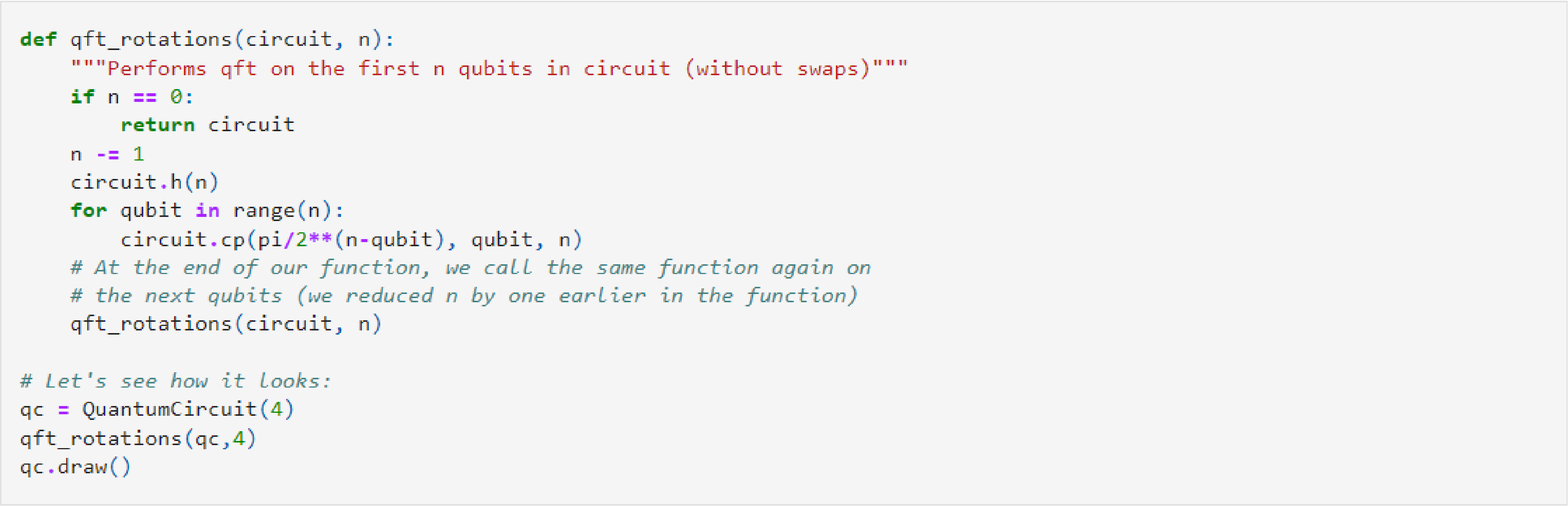
- Let’s see how this looks:

```
qc = QuantumCircuit(4)
qft_rotations(qc,4)
qc.draw()
```

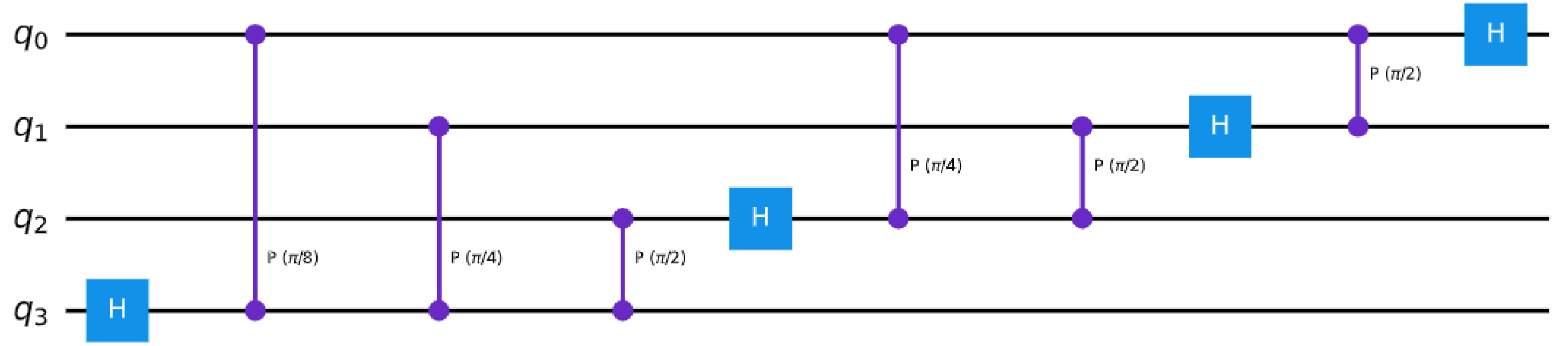


Quantum Fourier transform

- Great! This is the first part of our QFT. Now we have correctly rotated the most significant qubit, we need to correctly rotate the second most significant qubit.



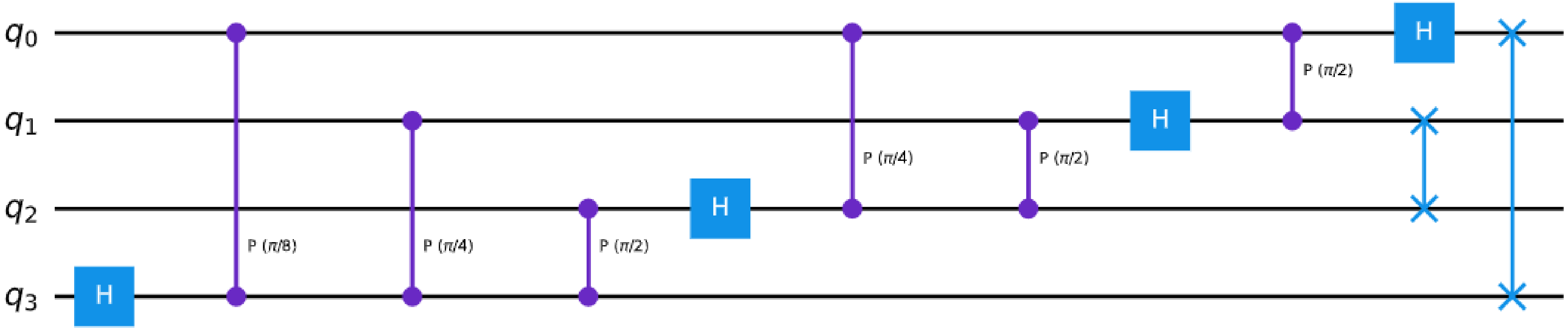
```
def qft_rotations(circuit, n):
    """Performs qft on the first n qubits in circuit (without swaps)"""
    if n == 0:
        return circuit
    n -= 1
    circuit.h(n)
    for qubit in range(n):
        circuit.cp(pi/2**(n-qubit), qubit, n)
    # At the end of our function, we call the same function again on
    # the next qubits (we reduced n by one earlier in the function)
    qft_rotations(circuit, n)

# Let's see how it looks:
qc = QuantumCircuit(4)
qft_rotations(qc,4)
qc.draw()
```



Quantum Fourier transform

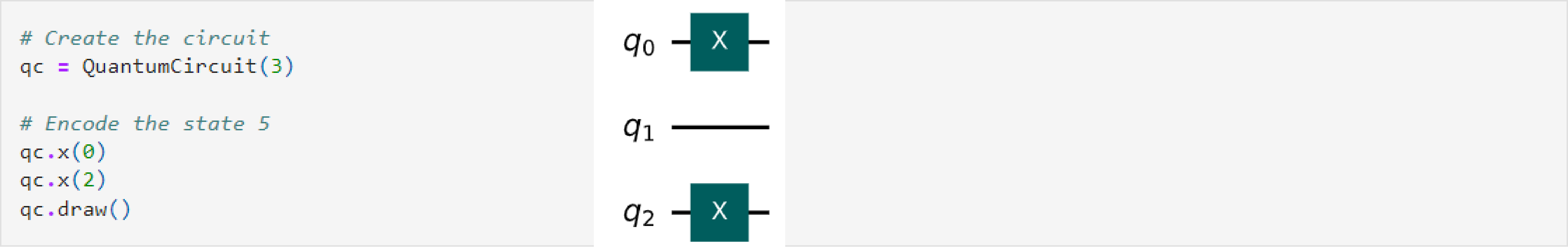
- Finally, we need to add the swaps at the end of the QFT function to match the definition of the QFT.

```
def swap_registers(circuit, n):
    for qubit in range(n//2):
        circuit.swap(qubit, n-qubit-1)
    return circuit

def qft(circuit, n):
    """QFT on the first n qubits in circuit"""
    qft_rotations(circuit, n)
    swap_registers(circuit, n)
    return circuit

# Let's see how it looks:
qc = QuantumCircuit(4)
qft(qc,4)
qc.draw()
```

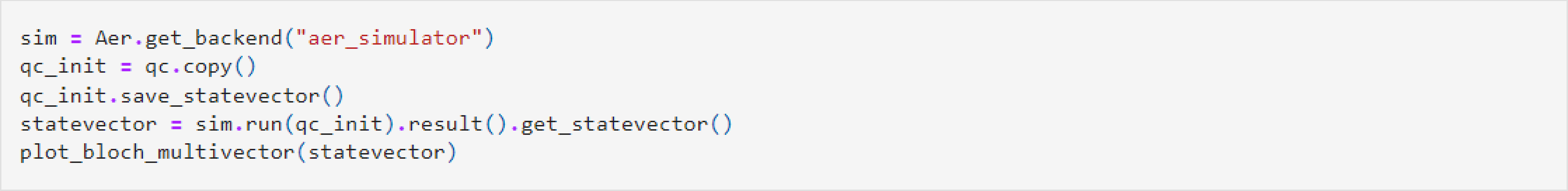


Quantum Fourier transform

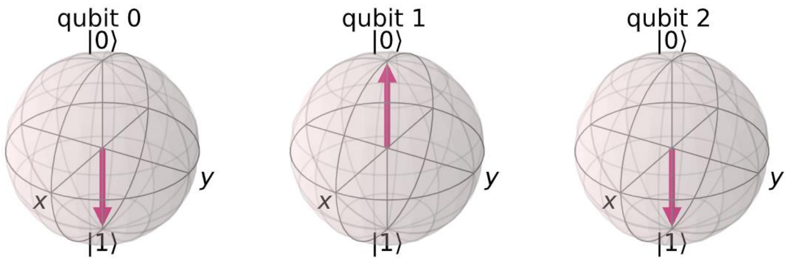
- We now want to demonstrate this circuit works correctly.
- To do this we must first encode a number in the computational basis.



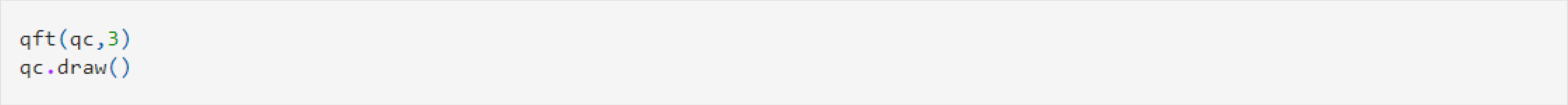
```
# Create the circuit
qc = QuantumCircuit(3)

# Encode the state 5
qc.x(0)
qc.x(2)
qc.draw()
```



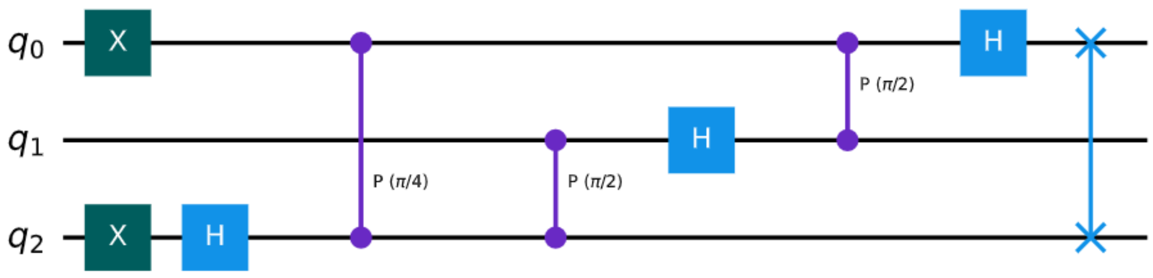
```
sim = Aer.get_backend("aer_simulator")
qc_init = qc.copy()
qc_init.save_statevector()
statevector = sim.run(qc_init).result().get_statevector()
plot_bloch_multivector(statevector)
```

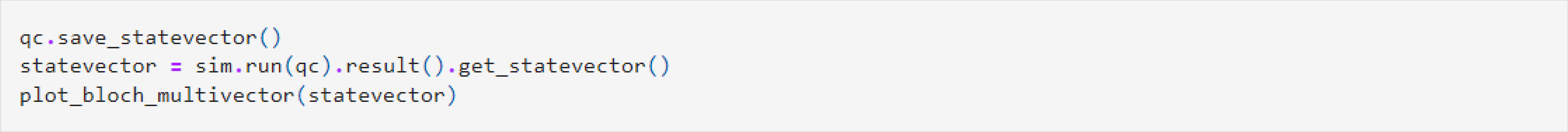


Quantum Fourier transform

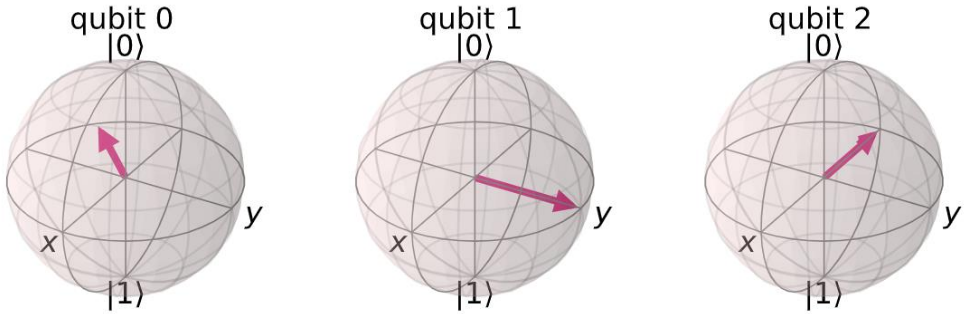


```
qft(qc,3)
qc.draw()
```





```
qc.save_statevector()
statevector = sim.run(qc).result().get_statevector()
plot_bloch_multivector(statevector)
```



پایان

سپاس از توجه شما

مرکز تحقیقات
فناوری‌های
کوانتومی ایران