Quantum algorithms: from basics to differential equations

School on Quantum Computation

November 14-25, 2022

ICTP-SAIFR, São Paulo, Brazil

Eduardo Inacio Duzzioni

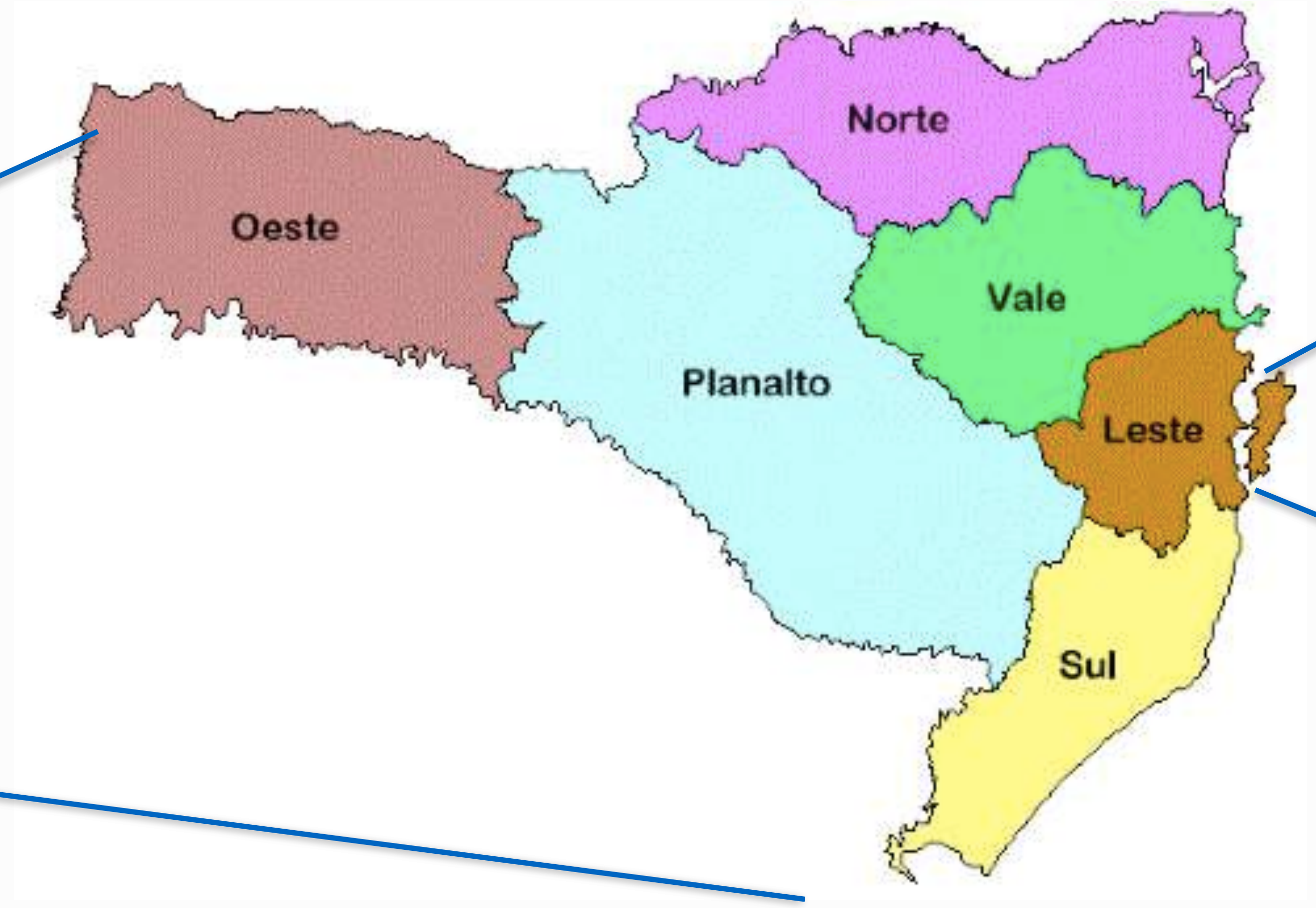
Physics Department

Federal University of Santa Catarina
& QuanBy Quantum Computing

Come visit us

Brazil

Santa Catarina



Florianópolis

Florianópolis city

Federal University of Santa Catarina

GIQSul - Grupo de Informação Quântica do Sul
(Southern Quantum Information Group)
https://giqsul.ufsc.br/

GCQ - Grupo de Computação Quântica
(Quantum Computing Group)
www.gcq.ufsc.br

QuanBy Quantum Computing
https://www.quanby.com.br

Outline

- **Day 1**
  - Preliminaries:
    - Turing machine
    - Quantifying computational resources
    - Complexity classes
    - Introduction to Quantum Computing: a historical perspective
    - Quantum computing models
    - Circuit notation and quantum gates
    - Universal sets of quantum gates
    - Solovay-Kitaev theorem
  - Quantum oracles
  - Oracular quantum algorithms:
    - Deutsch
    - Deutsch-Jozsa
    - Grover
    - Amplitude amplification

- **Day 2**
  - Quantum Fourier transform
  - Phase estimation
  - HHL
  - Modular exponentiation
  - Shor´s quantum algorithm
  - Hamiltonian simulation

- **Day 3**
  - Adiabatic quantum computing (AQC)
  - Quantum Annealing
  - Quantum approximate optimization algorithm - QAOA
  - Solving differential equations on quantum computers

Preliminaries

Algorithm

Algorithm is a finite sequence of instructions, typically used to solve a class of specific problems or to perform a computation.[1]

```
print("\n Normal Distribution")
print("-----------------")

circuit = QuantumCircuit(q,c)
normal = NormalDistribution(num_target_qubits = 5, mu=0, sigma=1,
low=- 1, high=1)
normal.build(circuit,q)
circuit.measure(q,c)

job = execute(circuit, backend, shots=8192)
job_monitor(job)
counts = job.result().get_counts()

print(counts)
sortedcounts = []
sortedkeys = sorted(counts)

for i in sortedkeys:
    for j in counts:
        if(i == j):
            sortedcounts.append(counts.get(j))
```

Python code: https://quantumcomputinguk.org/tutorials/tag/Python

PageRank algorithm - used by Google to search on internet.

Binary Search Algorithm - search on a structured database.

Euclid's algorithm - method for finding the greatest common divisor of two numbers.

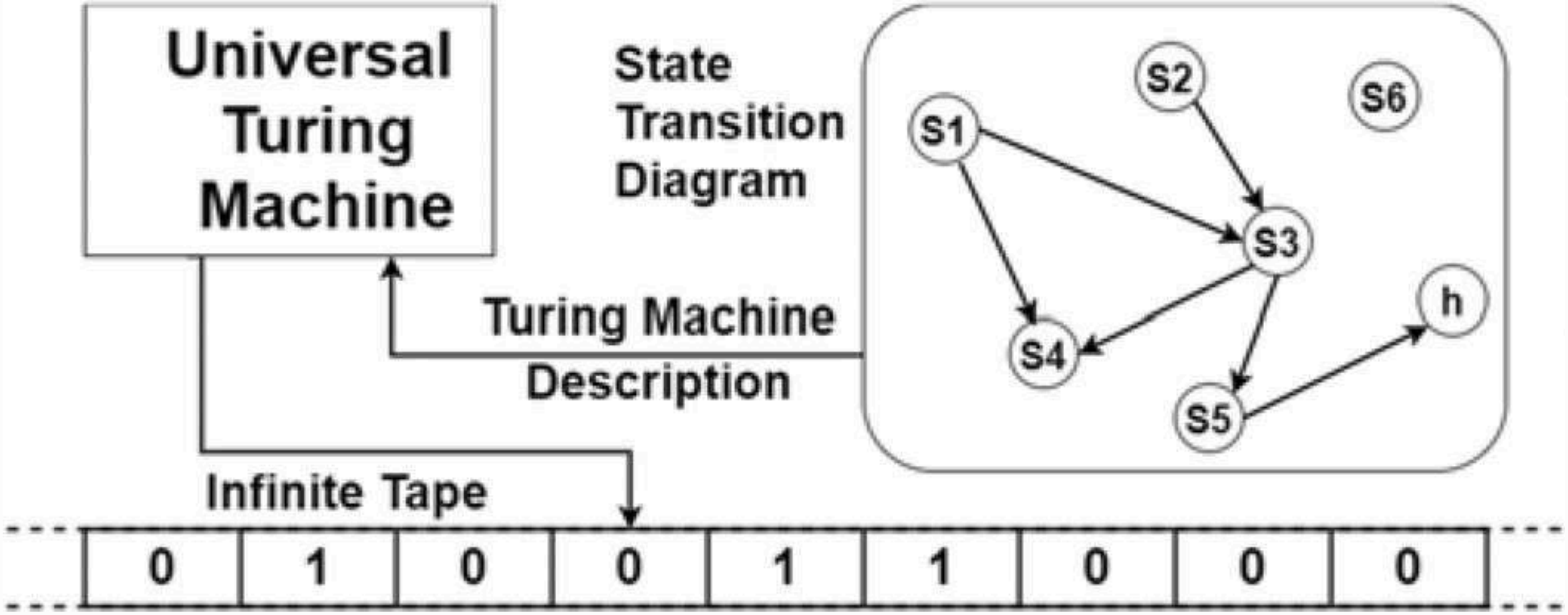
…

[1]Algorithm." *Merriam-Webster.com Dictionary*, Merriam-Webster, https://www.merriam-webster.com/dictionary/algorithm. Accessed 1 Nov. 2022.

Turing machine

(formal definition of algorithm ~ 1936)

A **Turing machine** is a mathematical model of computation describing an abstract machine that manipulates symbols on a strip of tape according to a table of rules. *https://en.wikipedia.org/wiki/Turing_machine#cite_note-2*



Alan M. Turing

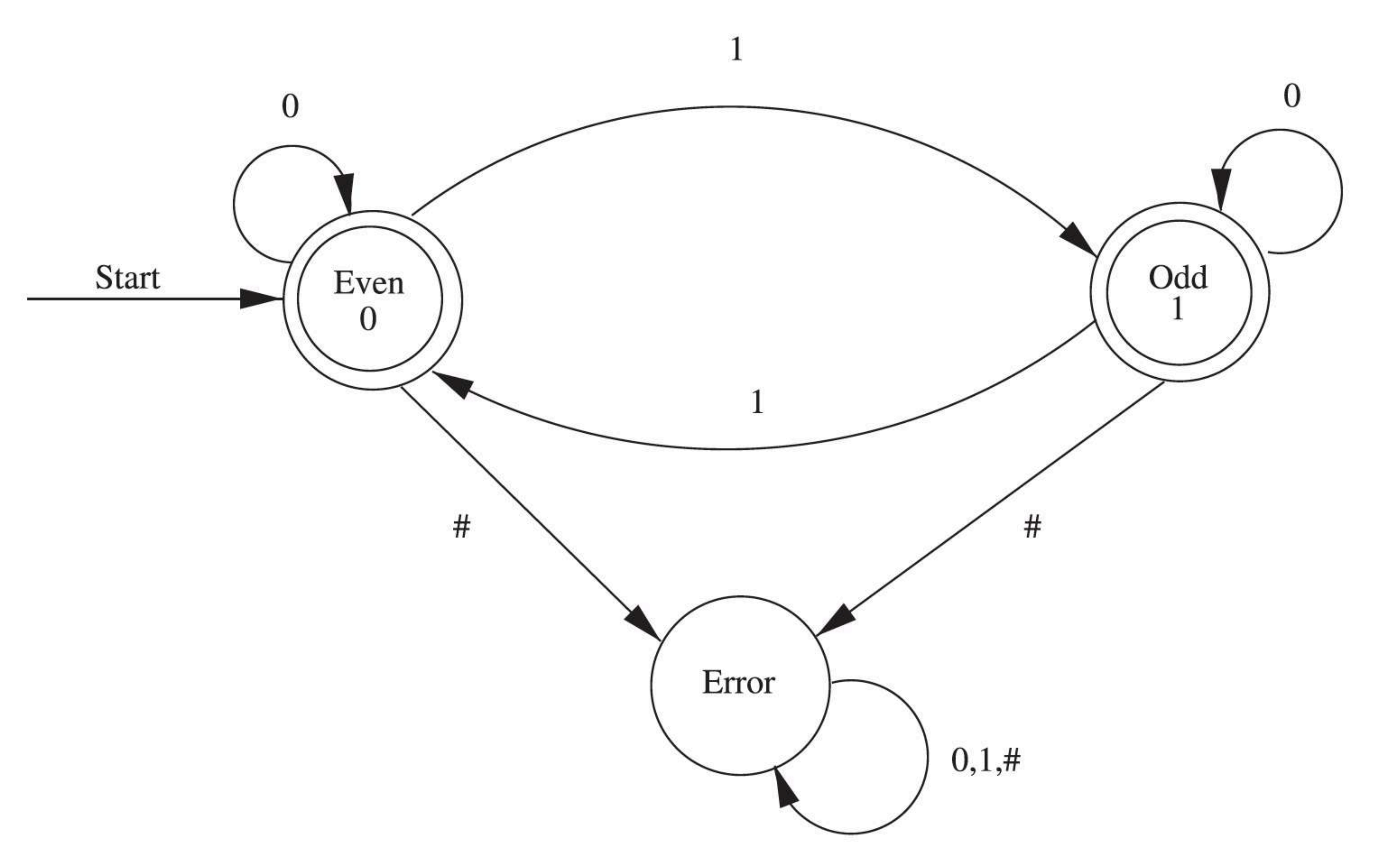
TM = {states, transition rules, a language, a control unit, memory tape}

Our modern computers UTM.

S. Malekmohamadi Faradonbe, F. Safi-Esfahani, and M. Karimian-kelishadrokhi, *SN Comput. Sci.* **1**, 333 (2020).

Example: finite automaton for parity computation

Given an input language (string) it compute the number of 1's.



- Double circled states are accept states
- Single-circled state is a reject state.
- The alphabet $\{0,1,\#\}$

Example: $L = \{00101101\}$ is even.

E. Salvador, et al., Contemporary Physics **59**, 174 (2018).

Quantifying computational resources

The execution of an algorithm requires the consumption of resources, such as *time*, *space*, and *energy*:

Time - the number of steps (elementary operations) needed to execute the algorithm;

Space - the amount of memory required to execute the algorithm;

Energy - the amount of energy required to execute the algorithm.

Depending on the problem, one or more resources may be minimized in favor of the others.

These resources are quantified through the *asymptotic analysis*.

Asymptotic analysis

The goal of asymptotic analysis is to capture the essential behavior of a given function according to the input size n for $n \to \infty$.

Big *O* notation - *O(n)*: it is used to give an upper bound on the asymptotic behavior of a function for large n .

Definition

$f(n) = O(g(n))$ means there exists constants $c > 0$ and integer $n_0 \geq 0$ such that $f(n) \leq cg(n)$ for $n \geq n_0$.

Example:

Function	Name
$324 = O(1)$	constant
$10 \log n + 50 = O(\log n)$	logarithmic
$a_k n^k + a_{k-a} n^{k-1} + \cdots + a_o = O(n^k)$	polynomial
$a^n + n^{35} = O(a^n); a > 1$	exponential

Informally: f grows as g or slower.

Big *Ω* notation - *Ω(n)*: it is used to give a lower bound on the asymptotic behavior of a function for large n.

Definition
$f(n) = \Omega(g(n))$ means there exists constants $c > 0$ and integer $n_0 \geq 0$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$. Or equivalently, $g(n) = O(f(n))$

Example:

Function
$5n - 2 = \Omega(\sqrt{n})$
$3n^2 + 10 = \Omega(n)$
$a_k n^k + a_{k-a} n^{k-1} + \cdots + a_o = \Omega(1)$
$a^n = \Omega(n^4); a > 1$

Informally: f grows as g or faster.

Big Θ notation - Θ(*n*): it combines the notations $O(n)$ and $\Omega(n)$, such that $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Definition

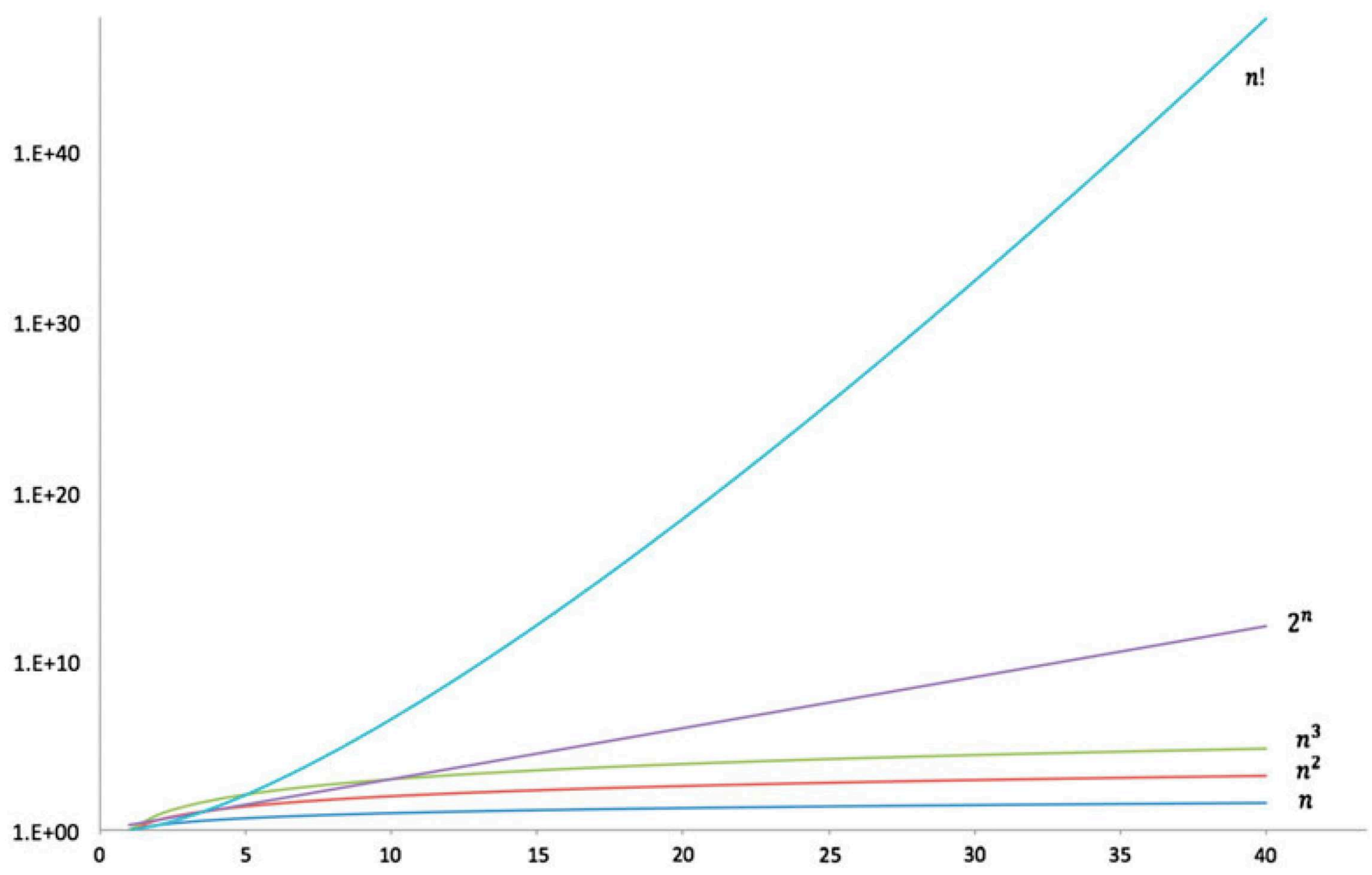
$f(n) = \Theta(g(n))$ means there exists constants $c_1, c_2 > 0$ and integer $n_0 \geq 0$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

Example:

Function
$5n - 2 = \Theta(n)$
$a_k n^k + a_{k-a} n^{k-1} + \cdots + a_o = \Theta(n^k)$
$7n^2 + \sqrt{n} \log n = \Theta(n^2)$

Informally: f grows in the same way as g.

Example



But, when a quantum algorithm is considered efficient?

An algorithm is called *efficient* if it takes a *polynomial time* or less to be solved.

Ex.: matrix multiplication, sum of integers, …

An algorithm is *inefficient* if it takes more than polynomial time, called *superpolynomial*.

This includes algorithms that take *subexponential time, exponential and factorial.*

Ex.: Factoring integer numbers classically (Number Field Sieve)

$$O(2^{n^{1/3}})$$

which are less than *exponential time*

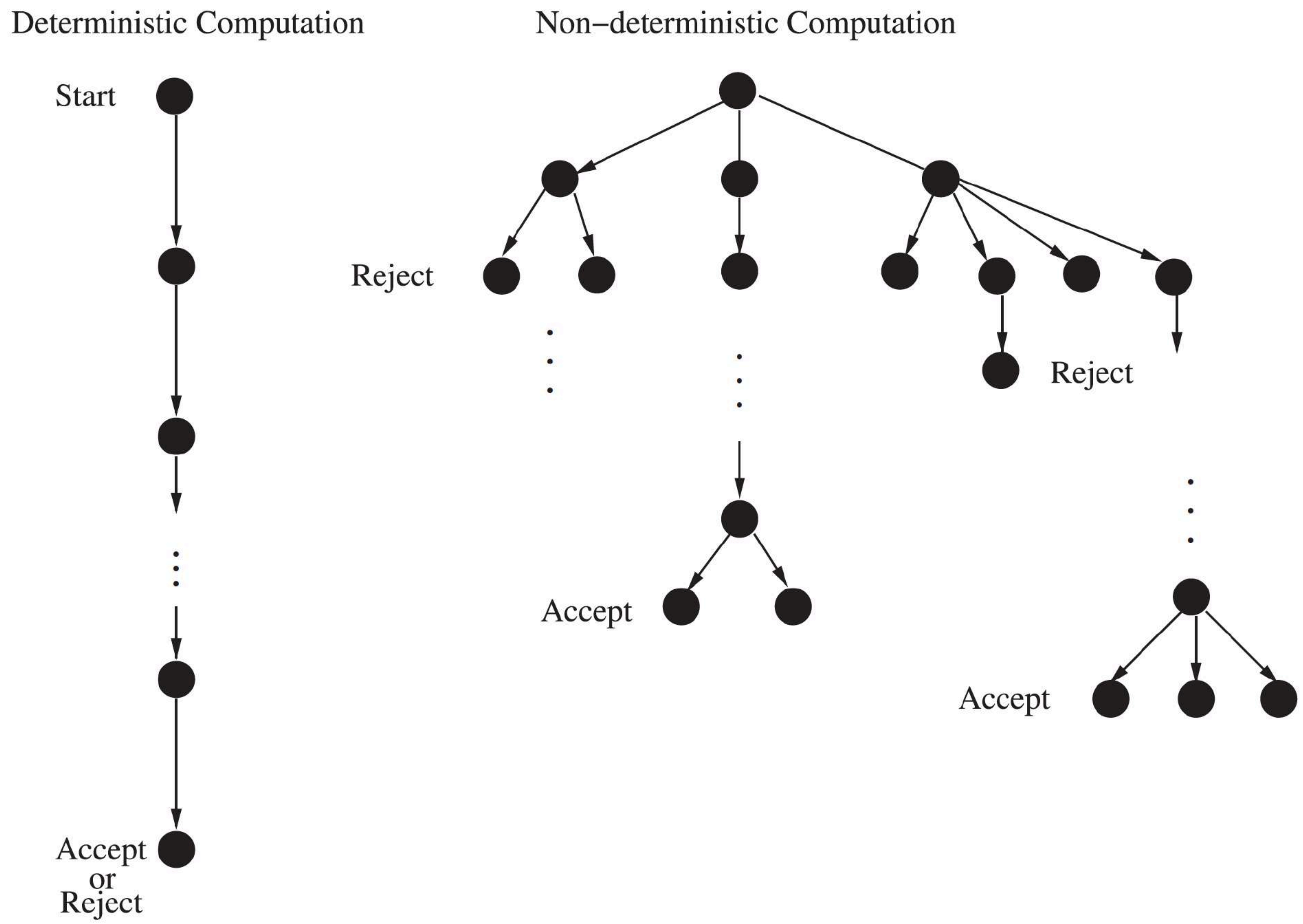
$$O(2^{poly(n)})$$

Deterministic and nondeterministic computation

Deterministic Turing machine - DTM

Nondeterministic Turing machine - NTM

For every step of a computation on a DTM, a state is followed by only one state.



For every step of a computation on a NTM, a state is followed by one or more states.

E. Salvador, et al., Contemporary Physics **59**, 174 (2018).

Complexity classes
(non-formal definitions)

The class P is the set of decision problems solvable in polynomial time on DTMs.

Ex: Determining if a number is prime; Linear programming, Binary search, …

Complexity classes
(non-formal definitions)

The class NP is the set of decision problems solvable in polynomial time on NTMs.

Equivalent definition: **NP is the set of decision problems for which it is possible to check, in polynomial time, if a proposed solution is indeed a solution.**
Observe that $P \subseteq NP$. The opposite is unknown.

Ex: Decision version of traveling salesman problem, integer factorization, boolean satisfiability, …

Complexity classes
(non-formal definitions)

The class NP-Complete is the set of decision problems in NP that can be mapped to other NP problems using at most polynomial resources.

Ex: Boolean satisfiability, Knapsack problem, Traveling salesman problem (decision) …

Complexity classes
(non-formal definitions)

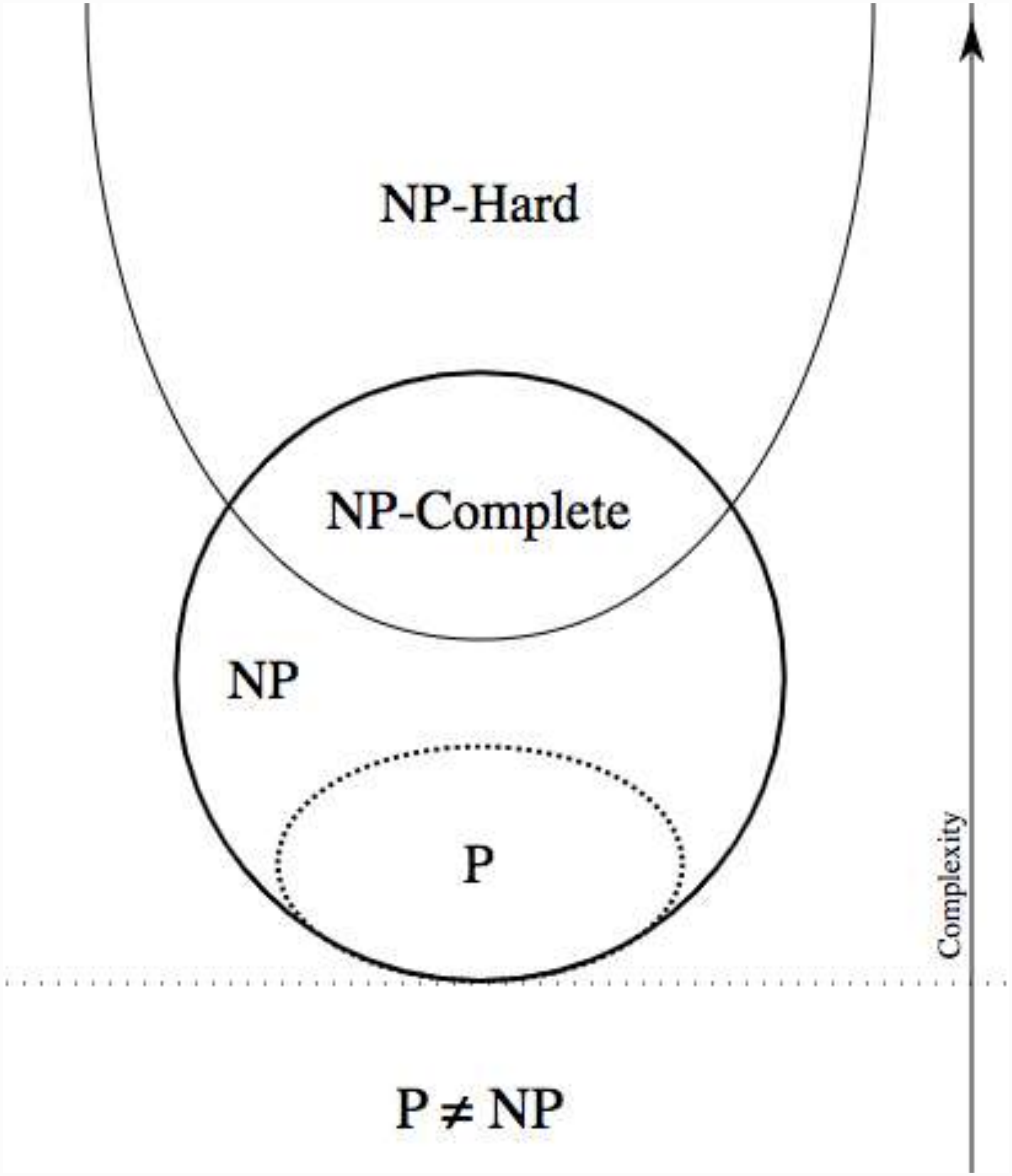
The class NP-Complete is the set of decision problems in NP that can be mapped to other NP problems using at most polynomial resources.

Ex: Boolean satisfiability, Knapsack problem, Traveling salesman problem (decision) …

NP-hard means as hard as the most difficult problem in NP.

Ex: Optimization problems. The *original* traveling salesman problem.

Euler diagram



https://en.wikipedia.org/wiki/NP-completeness

Introduction to quantum computing:
A historical perspective

Computability

Church-Turing Thesis: Any algorithmic process can be efficiently simulated using an Turing machine.

Paul Benioff

Turing machine
vs.
Reversible computation

A *Turing machine* can be simulated by the unitary (reversible) evolution of a quantum process.

Paul Benioff, *J. Statist. Phys.* **22**, 563 (1980).

1st Physics of Computation Conference

MIT Endicott House - 1981

ICTP SAIFR International Centre for Theoretical Physics South American Institute for Fundamental Research

Physics of Computation Conference Endicott House MIT May 6-8, 1981

1 Freeman Dyson
2 Gregory Chaitin
3 James Crutchfield
4 Norman Packard
5 Panos Ligomenides
6 Jerome Rothstein
7 Carl Hewitt
8 Norman Hardy
9 Edward Fredkin
10 Tom Toffoli
11 Rolf Landauer
12 John Wheeler
13 Frederick Kantor
14 David Leinweber
15 Konrad Zuse
16 Bernard Zeigler
17 Carl Adam Petri
18 Anatol Holt
19 Roland Vollmar
20 Hans Bremerman
21 Donald Greenspan
22 Markus Buettiker
23 Otto Floberth
24 Robert Lewis
25 Robert Suaya
26 Stan Kugell
27 Bill Gosper
28 Lutz Priese
39 Madhu Gupta
30 Paul Benioff
31 Hans Moravec
32 Ian Richards
33 Marian Pour-El
34 Danny Hillis
35 Arthur Burks
36 John Cocke
37 George Michaels
38 Richard Feynman
39 Laurie Lingham
40 Thiagarajan
41 ?
42 Gerard Vichniac
43 Leonid Levin
44 Lev Levitin
45 Peter Gacs
46 Dan Greenberger

Charles H. Bennett took the picture

Feynman´s proposal:
Quantum computers as universal quantum simulators

ICTP SAIFR
International Centre for Theoretical Physics
South American Institute for Fundamental Research

International Journal of Theoretical Physics, Vol. 21, Nos. 6/7, 1982

Simulating Physics with Computers

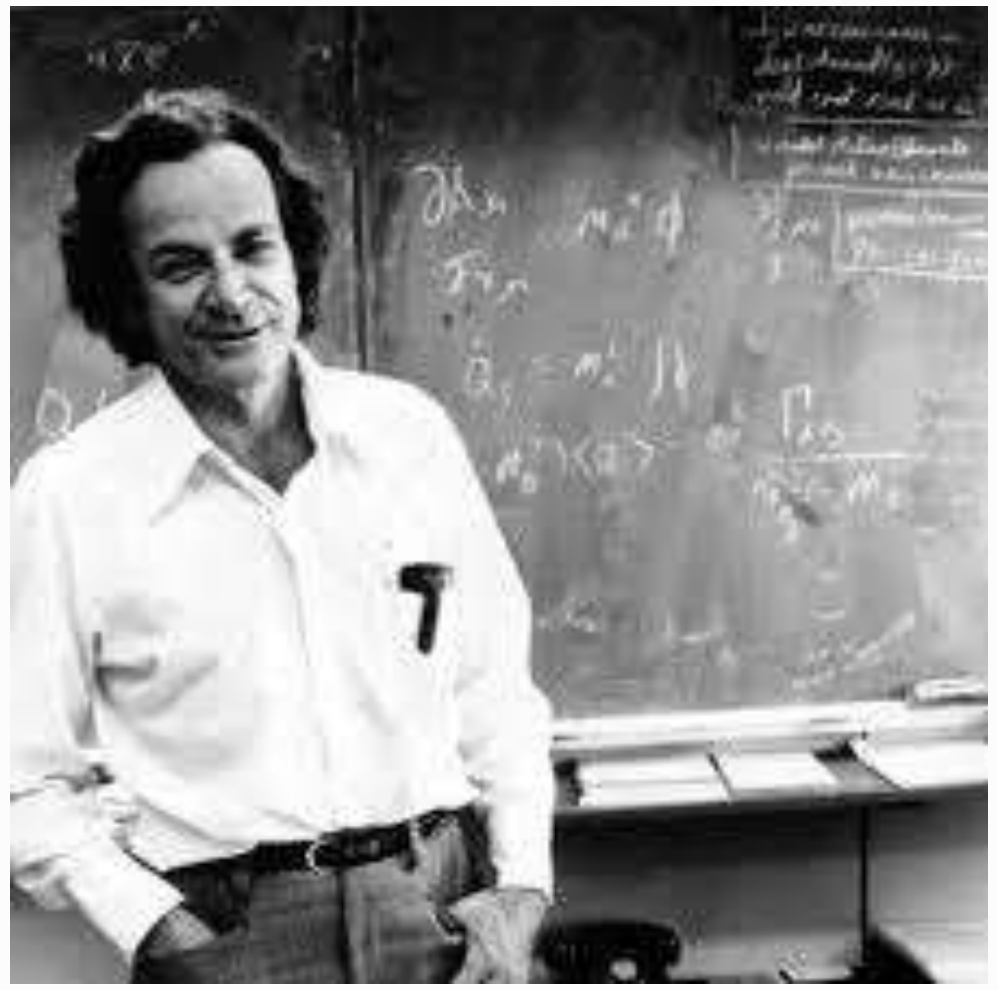
Richard P. Feynman

Department of Physics, California Institute of Technology, Pasadena, California 91107

Received May 7, 1981

4. QUANTUM COMPUTERS—UNIVERSAL QUANTUM SIMULATORS

The first branch, one you might call a side-remark, is, Can you do it with a new kind of computer—a quantum computer? (I'll come back to the other branch in a moment.) Now it turns out, as far as I can tell, that you can simulate this with a quantum system, with quantum computer elements. It's not a Turing machine, but a machine of a different kind. If we disregard the continuity of space and make it discrete, and so on, as an approximation (the same way as we allowed ourselves in the classical case), it does seem to



"Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy."

Simulating physical systems with classical computers

How much memory does a computer need to simulate a material?

Pieces of iron

https://en.wikipedia.org/wiki/Iron

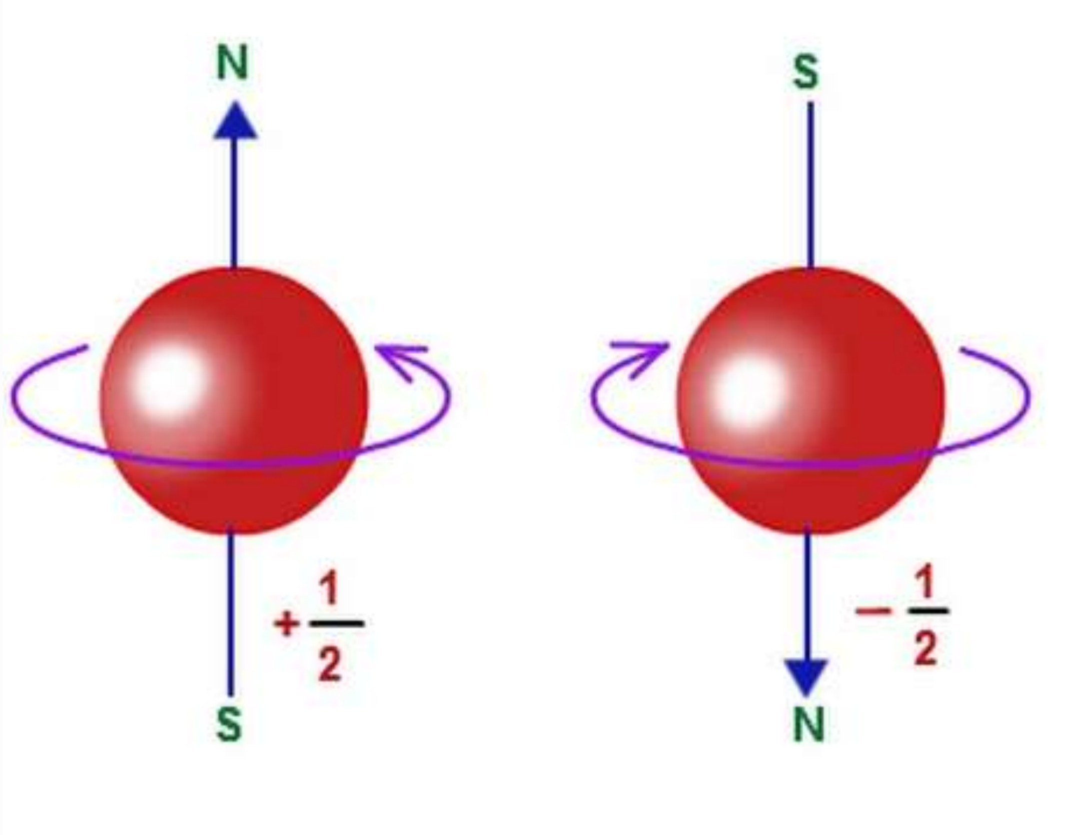
protons = 26
electrons = 26
neutrons = 30
 # particles = 82

Physical description includes spatial (position, linear momentum) and intrinsic properties (spin) of each particle

Let's simplify life...
Stick to what interests you!!

ICTP SAIFR International Centre for Theoretical Physics South American Institute for Fundamental Research

Let's consider spin ½ particles, such as electrons, protons and neutrons.



In the classical computer, each variable is represented by 64 bits (double precision). Therefore, **it takes 70 Tb of RAM to describe 43 spins**.

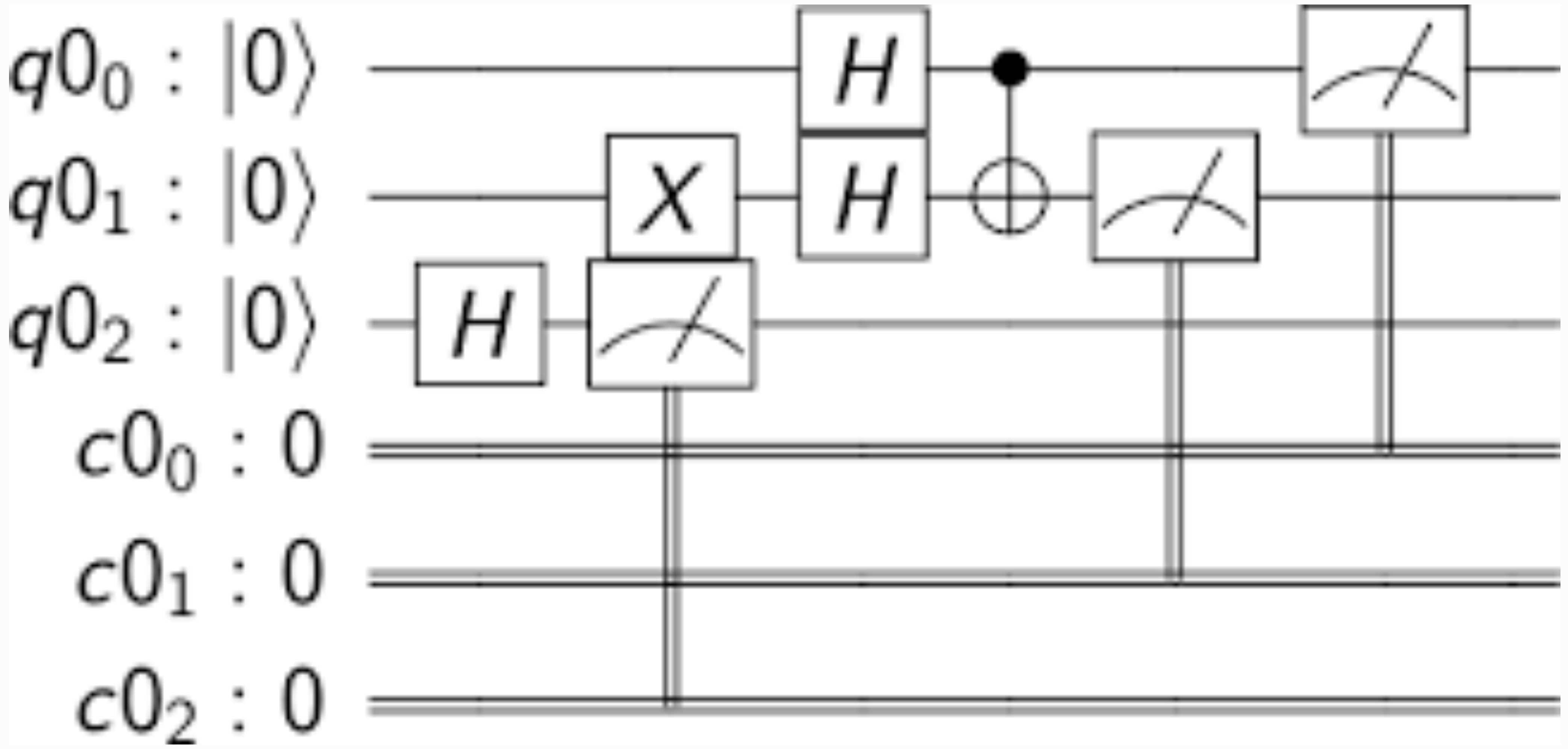
# spins 1/2	# bits
1	2
2	4
3	8 = 1byte
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
13	8.192 bits = 1 kilobyte
23	8.388.608 bits = 1 megabyte
33	8.589.934.592 bits = 1 gigabyte
43	8.796.093.022.208 bits = 1 terabyte

Circuital model of quantum computing

Non-formal definition

A Quantum Turing Machine (QTM) is a machine capable of effectively simulating an arbitrary physical system.

David Deutsch



Any quantum algorithm can be expressed formally as a particular quantum Turing machine.

D. Deutsch, *Proc. R. Soc. A*. **400**, 97 (1985).
A. Molina, J. Watrous, Proc Math Phys Eng Sci. **475**, 20180767 (2019).

Shor's factoring algorithm

Algorithms for Quantum Computation:
Discrete Log and Factoring

Extended Abstract

Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974 USA

email: shor@research.att.com

Abstract

This paper gives algorithms for the discrete log and the factoring problems that take random polynomial time on a quantum computer (thus giving the first examples of quantum cryptanalysis).

P. W. Shor, *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press: 124 (1994).

Peter Shor

RSA security foundation

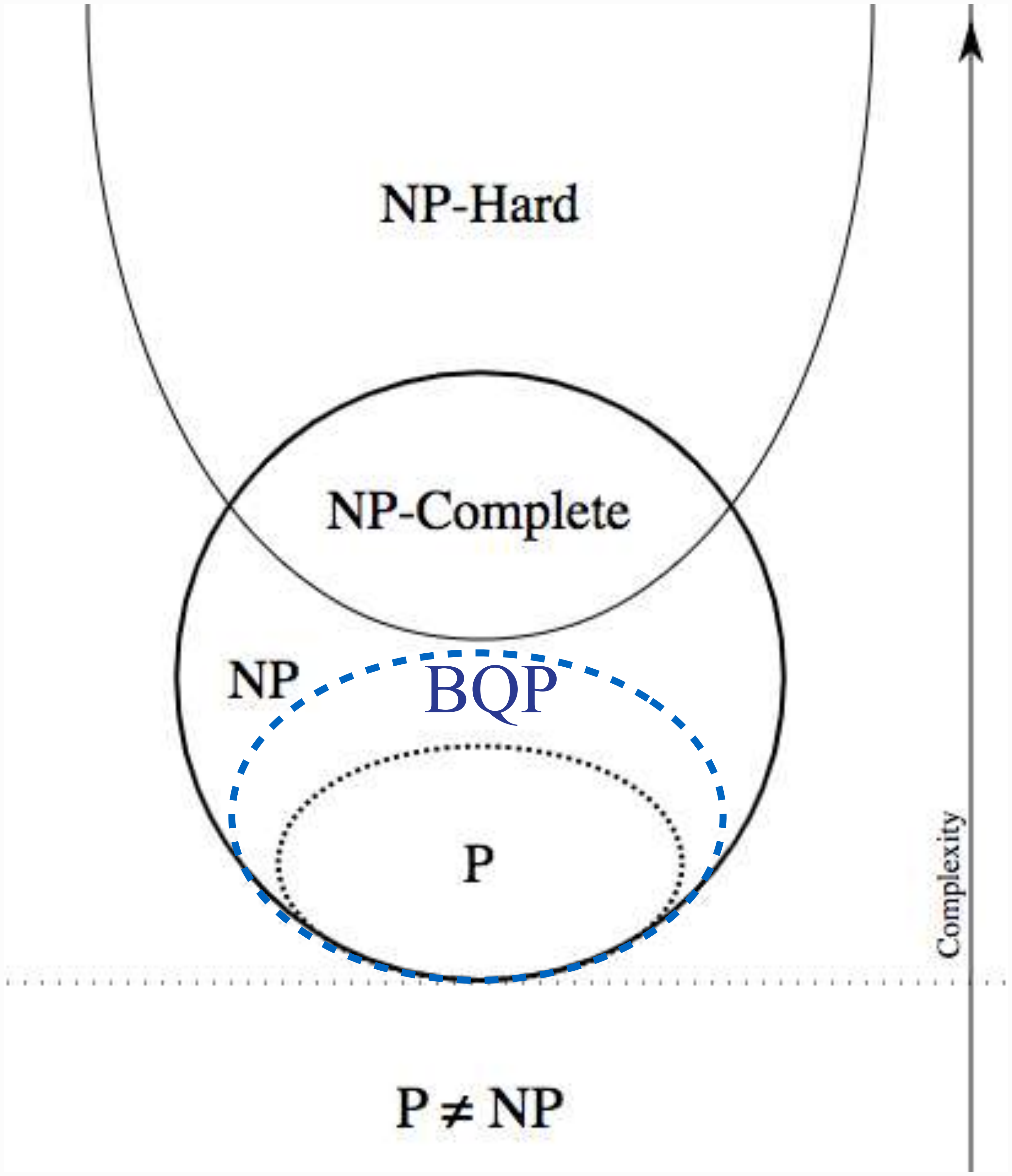
3 x 5 = 15 - **easy**

1917500294268803292859 9 x 1355877461004671178070 1 = 2.59989543×10^{44} - **hard**

After the advent of Shor’s factoring algorithm

Bounded-error quantum polynomial time (BQP) is the class of decision problems solvable by a quantum computer in polynomial time, with an error probability of at most 1/3 for all instances.

Ex:
Integer factorization (Shor's algorithm),
Discrete logarithm,
Simulation of quantum systems,
Harrow-Hassidim-Lloyd (HHL) algorithm.



Quantum Computing Models

Universal models - implement universal quantum computation

- Quantum circuit model
- Adiabatic quantum computing
- Quantum Turing machine
- Measurement based quantum computation
- Quantum walks
- Topological quantum computing
- …

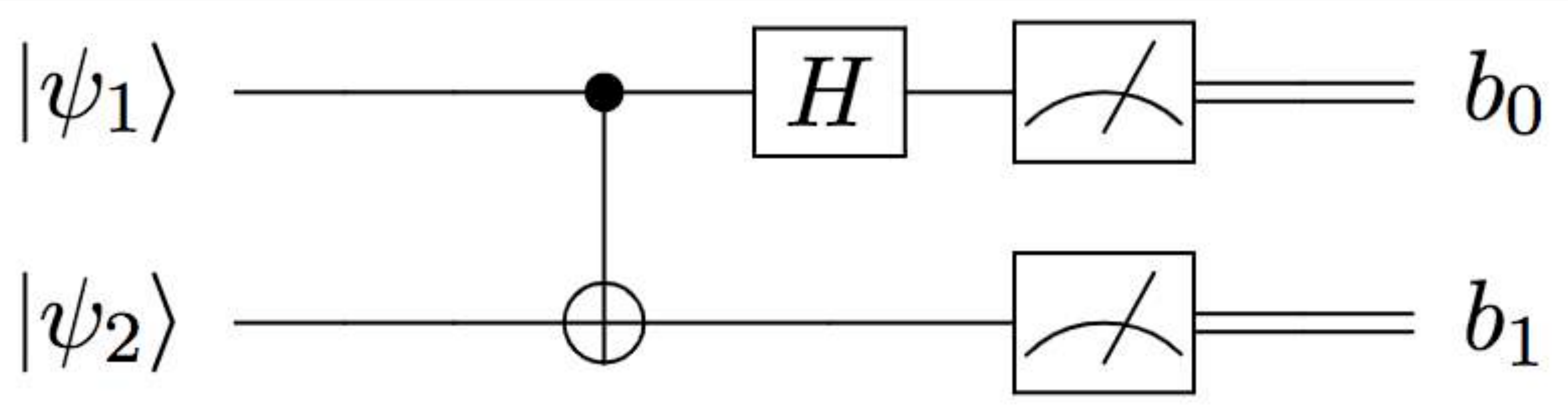
Restricted models - implement restricted functions

- *Quantum annealing* - calculates the ground state of the Ising model
- *Boson sampling* - calculates the permanent of matrices
- *Deterministic quantum computation with one quit (DQC1)* - calculates the trace of a unitary matrix
- …

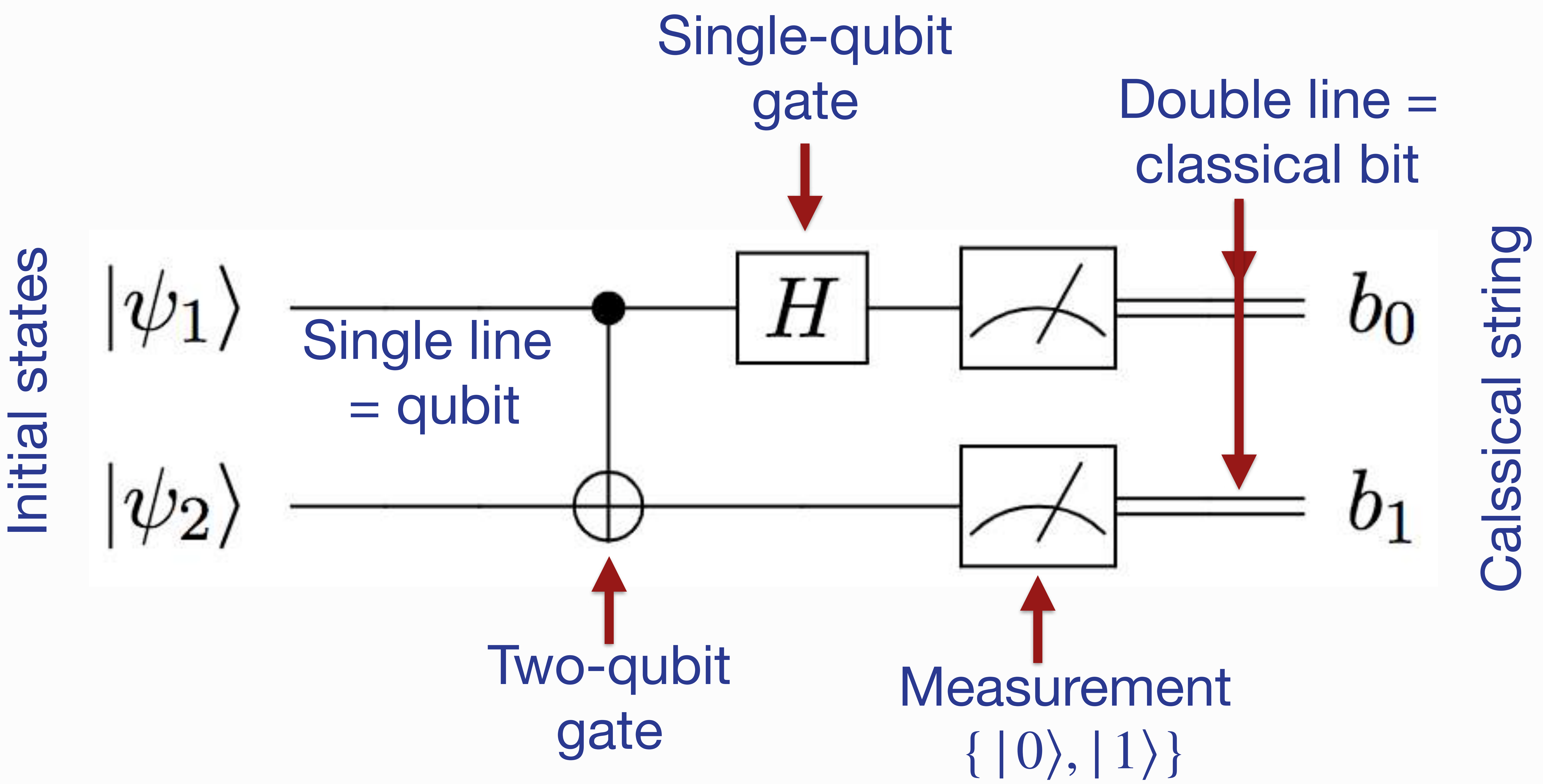
Quantum circuital model

Circuit notation

Simple example



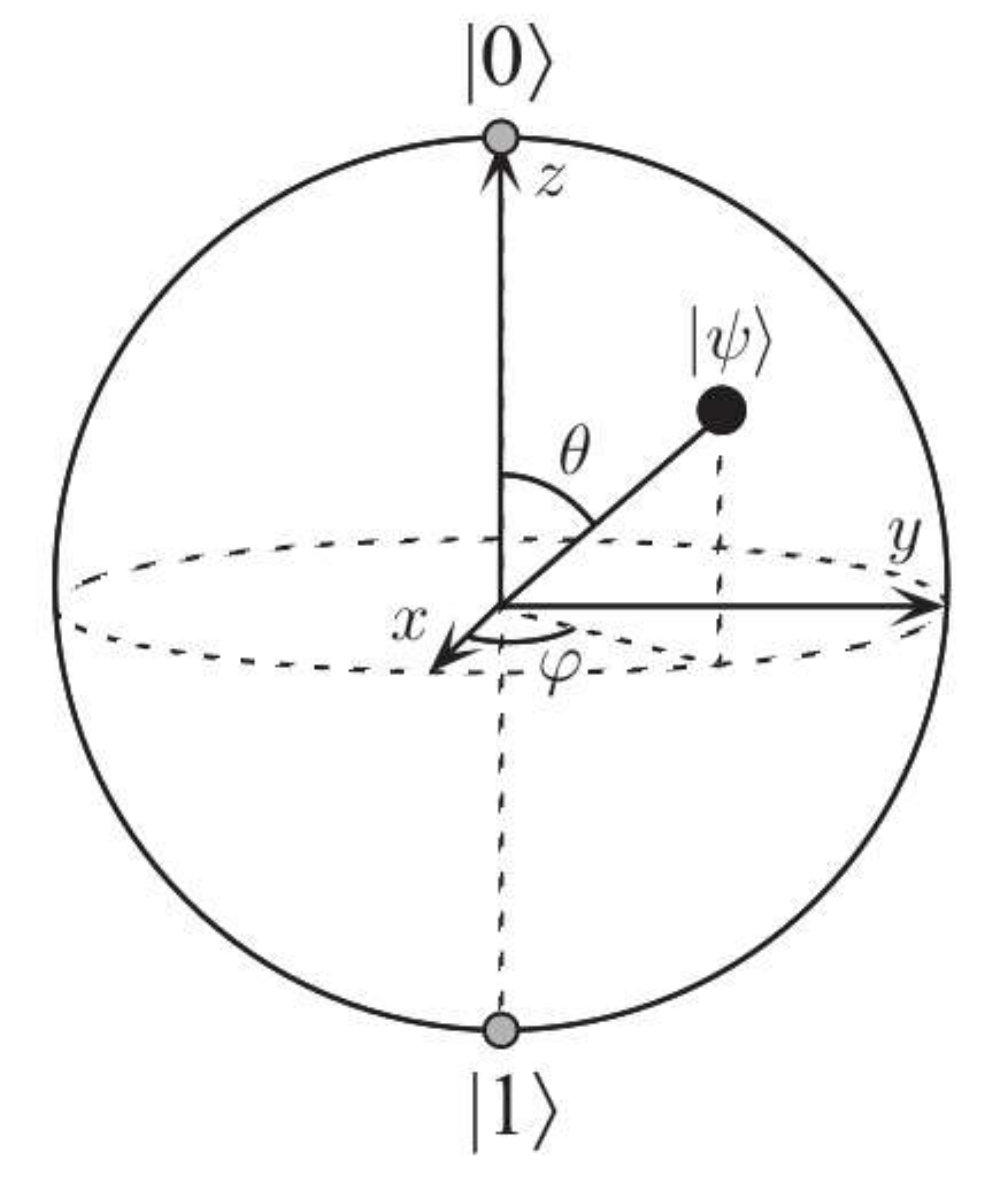
Circuit notation



(Discrete) time evolution - from left to right

Qubit - the building block

Bloch sphere



Qubit

$|\psi\rangle = a\,|0\rangle + b\,|1\rangle$ s.t. $a, b \in \mathbb{C}$

$|a|^2 + |b|^2 = 1$ -normalization condition

Qubit representation on the Bloch sphere

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

Polar angle $\theta \in [0, \pi]$

Azimuthal angle $\varphi \in [0, 2\pi)$

Quantum gates

Quantum gates are unitary operators

$$U^{-1} = U^{\dagger} \qquad \text{or} \qquad UU^{\dagger} = U^{\dagger}U = 1$$

The evolution of a given state in a quantum circuit is *reversible* and *preserve its norm.*

Single-qubit quantum gates

Single-qubit gates are rotations on the Bloch sphere.

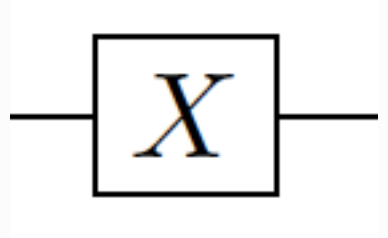
$$U = e^{-i\alpha} R_{\hat{n}}(\theta)$$

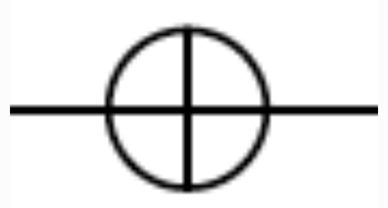
Exercise: Show that

$$R_{\hat{n}}(\theta) = \cos\left(\frac{\theta}{2}\right)\mathbb{I} - i\sin\left(\frac{\theta}{2}\right)\hat{n}\cdot\vec{\sigma} \qquad \vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$$

NOT gate or Pauli X gate

Symbol

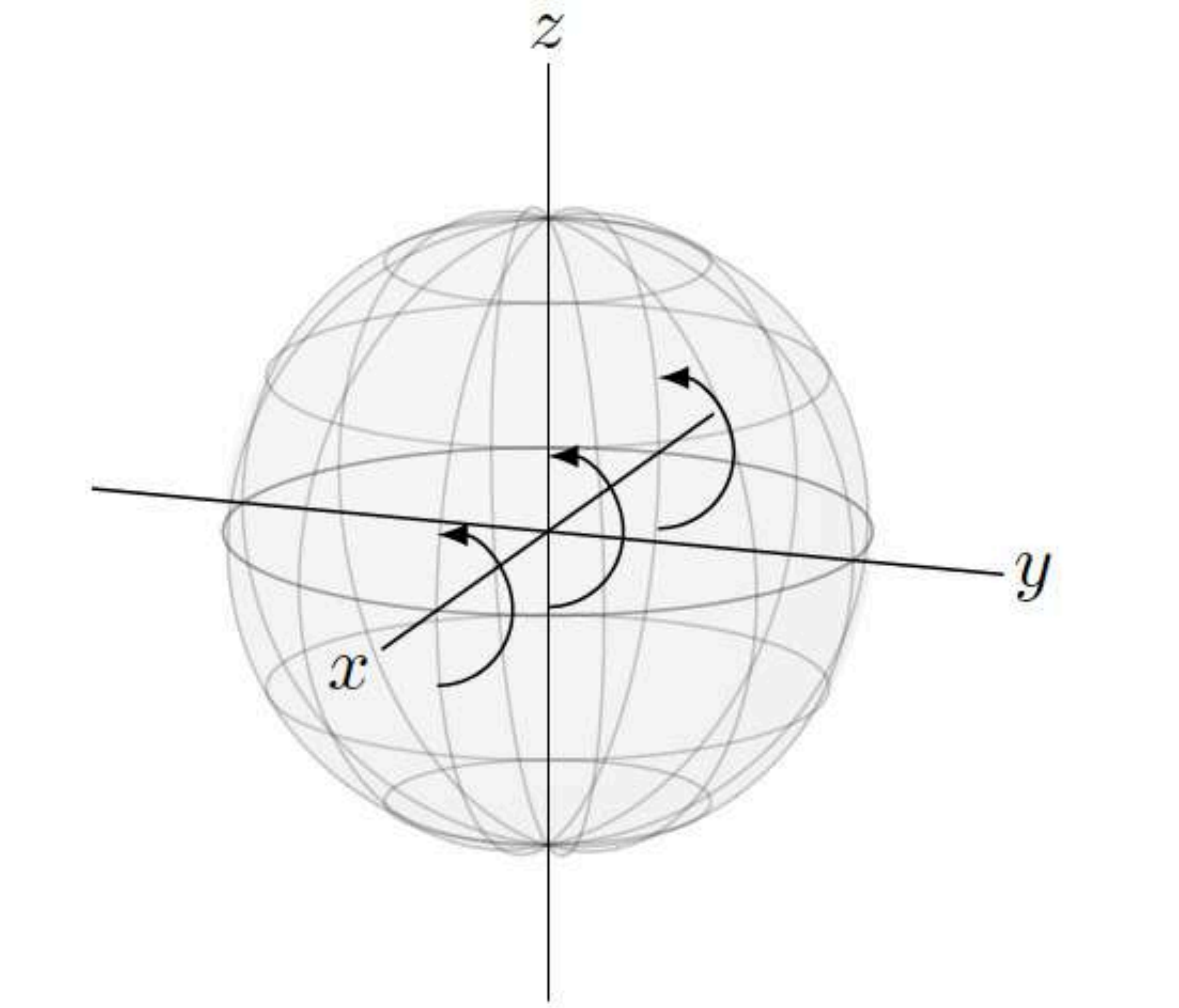




XOR

Matrix representation

$$\sigma_x = X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



$$R_{\hat{x}}(\pi) = X$$

Truth table

$$X|0\rangle = |1\rangle$$
$$X|1\rangle = |0\rangle$$

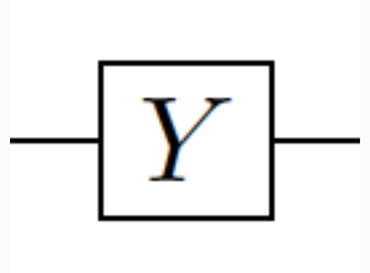
or

$$X|b\rangle = |\bar{b}\rangle$$

b=0,1 ; $\bar{b}$=NOT(b)

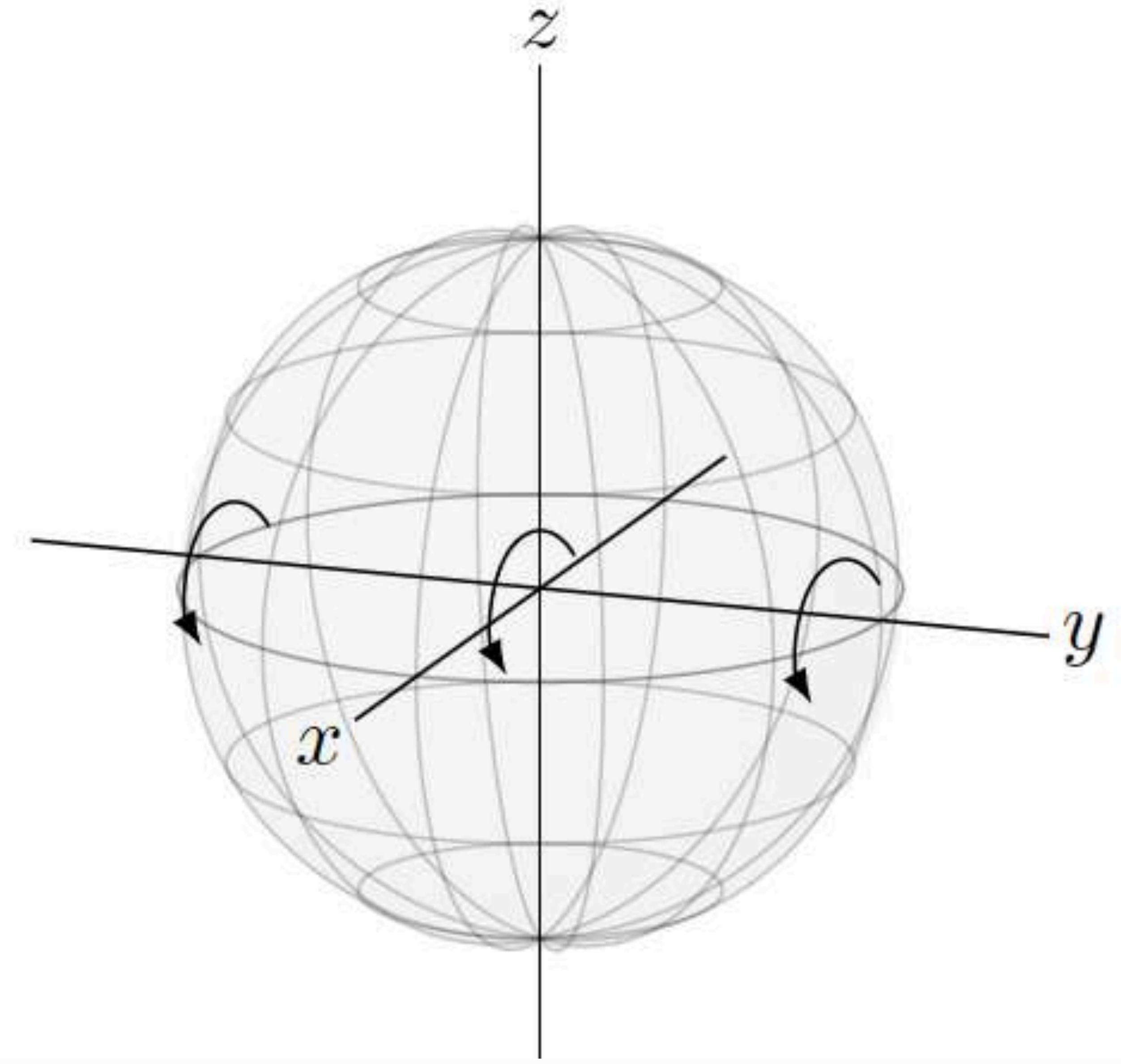
Pauli Y gate

Symbol



Matrix representation

$$\sigma_y = Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$



$$R_{\hat{y}}(\pi) = Y$$

Truth table

$$Y|0\rangle = i|1\rangle$$
$$Y|1\rangle = -i|0\rangle$$

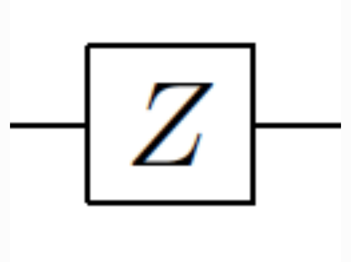
or

$$Y|b\rangle = (-1)^b i|\bar{b}\rangle$$

b=0,1 ; $\bar{b}$=NOT(b)

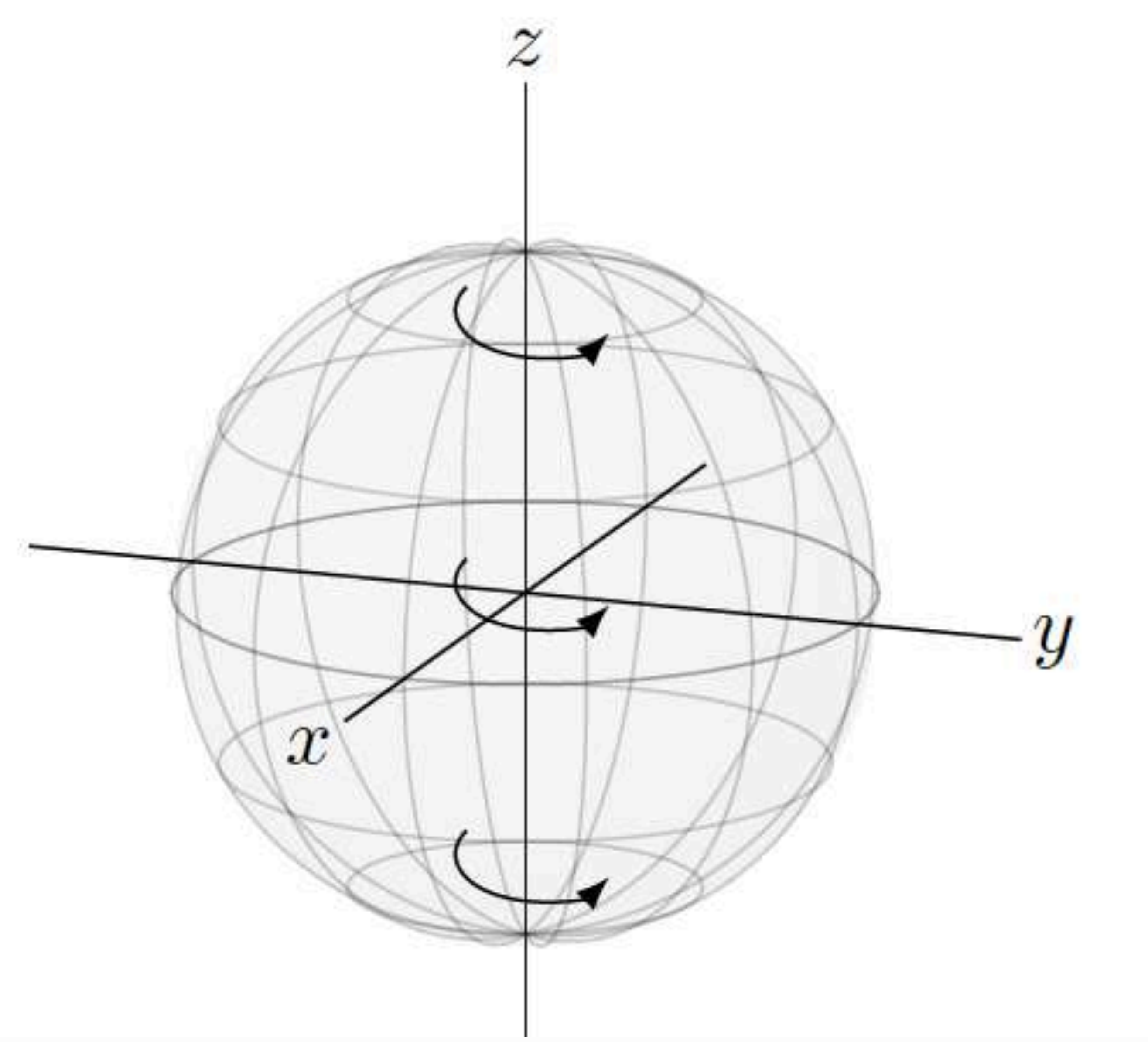
Pauli Z gate

Symbol



Matrix representation

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



$$R_{\hat{z}}(\pi) = Z$$

Truth table

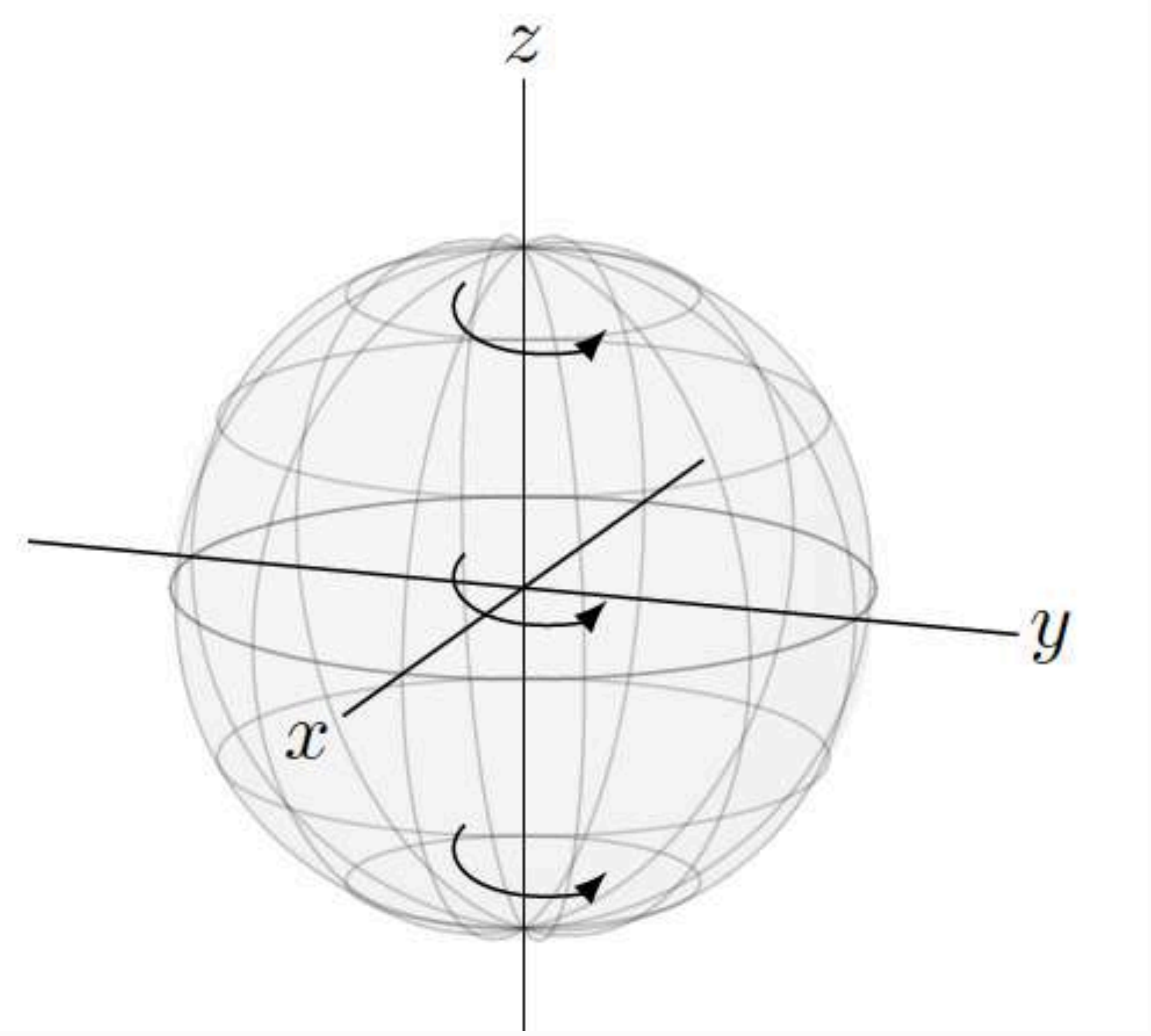
$$Z\left|0\right\rangle = \left|0\right\rangle$$
$$Z\left|1\right\rangle = -\left|1\right\rangle$$

or

$$Z\left|b\right\rangle = (-1)^b \left|b\right\rangle$$

$b{=}0,1$

Quantum phase gate



Matrix representation

$$P(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

Truth table

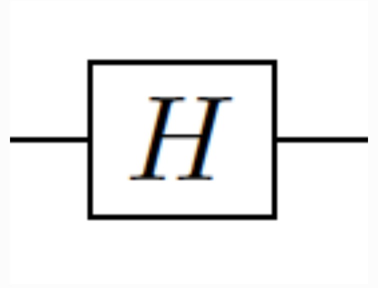
$$P(\theta)|0\rangle = |0\rangle$$
$$P(\theta)|1\rangle = e^{i\theta}|1\rangle$$

$$R_{\hat{z}}(\pi) = P(\pi) = Z$$
$$R_{\hat{z}}(\pi/2) = P(\pi/2) = S$$ → Phase gate
$$R_{\hat{z}}(\pi/4) = P(\pi/4) = T$$ → T gate

Exercise: What is the relation between the gates Z, S, and T?

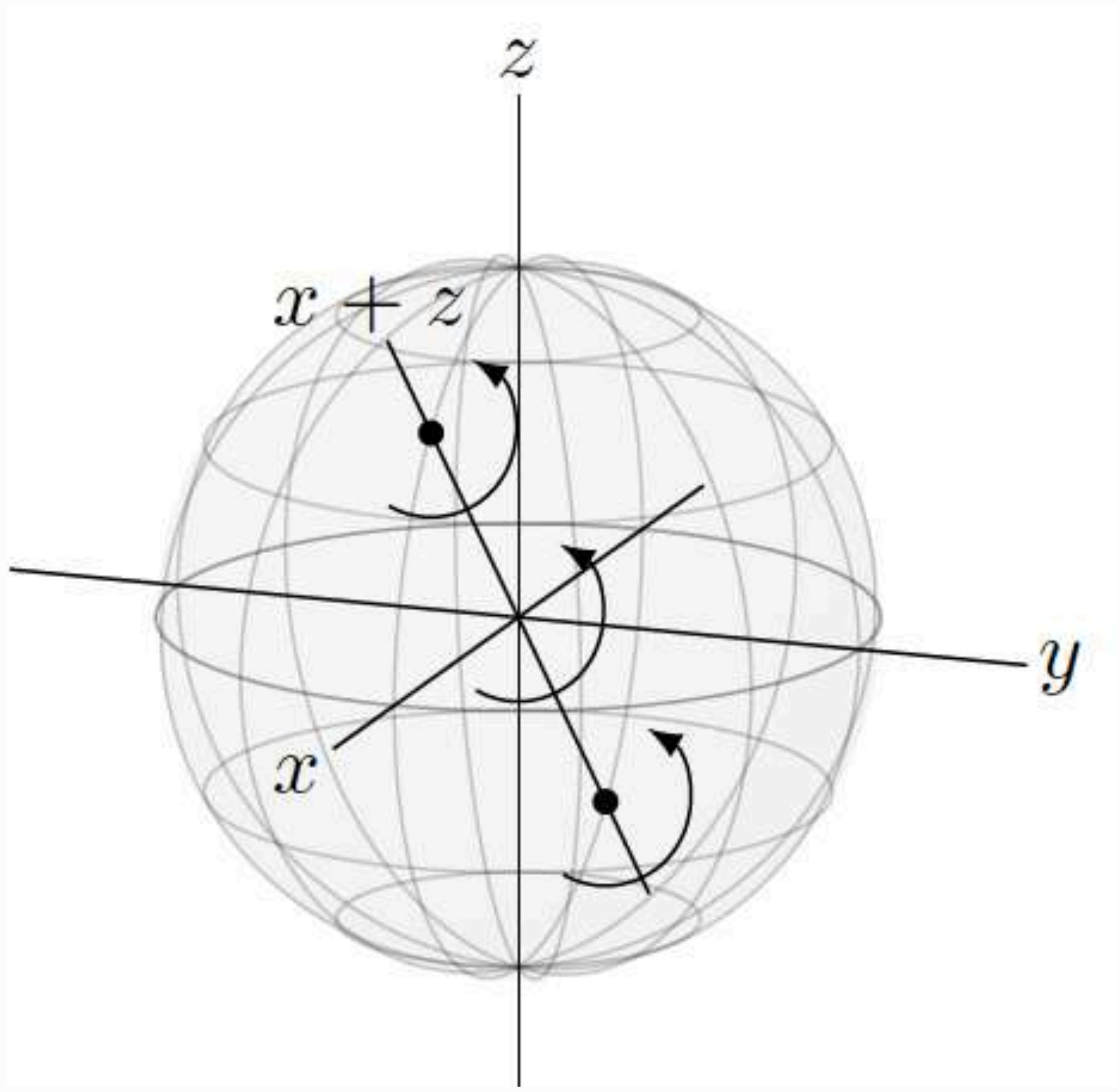
Hadamard gate

Symbol



Matrix representation

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



Truth table

$$H|0\rangle = |+\rangle$$
$$H|1\rangle = |-\rangle$$

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

It "creates" superposition!

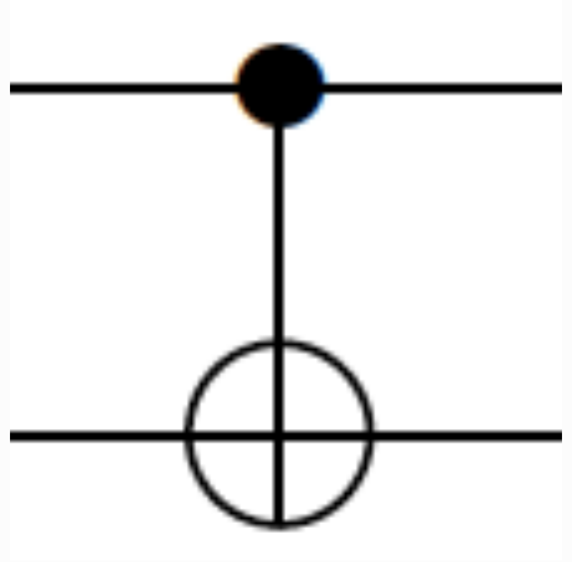
Exercise: Obtain the direction of the rotation axis for the Hadamard gate.

Two-qubit quantum gates

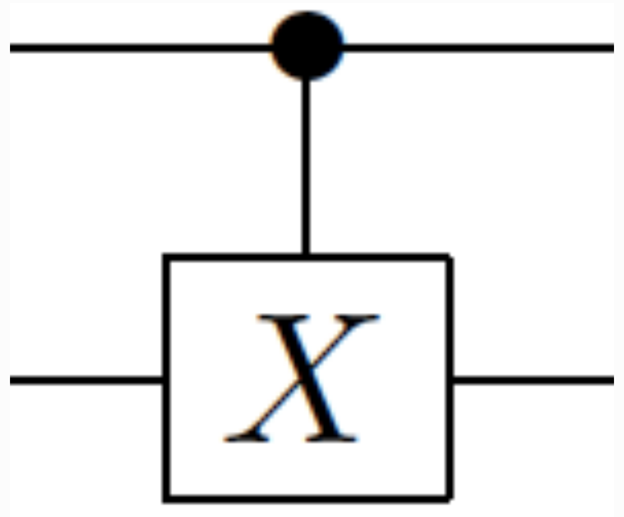
CNOT - Controlled NOT gate

Symbol

Control



Target



Matrix representation

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Truth table

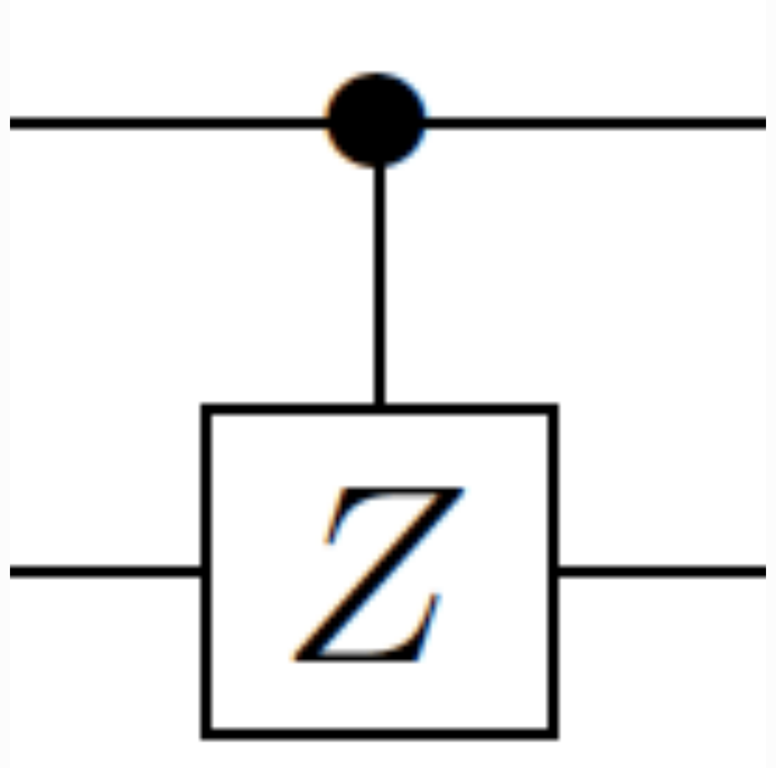
$$\text{CNOT}\,|00\rangle = |00\rangle$$
$$\text{CNOT}\,|01\rangle = |01\rangle$$
$$\text{CNOT}\,|10\rangle = |11\rangle$$
$$\text{CNOT}\,|11\rangle = |10\rangle$$

or

$$\text{CNOT}\,|a, b\rangle = |a, a \oplus b\rangle$$

CZ gate - Controlled Z gate

Symbol



Matrix representation

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Truth table

$$\begin{aligned} CZ\,|00\rangle &= |00\rangle \\ CZ\,|01\rangle &= |01\rangle \\ CZ\,|10\rangle &= |10\rangle \\ CZ\,|11\rangle &= -\,|11\rangle \end{aligned}$$

or

$$\begin{aligned} CZ\,|0b\rangle &= |0b\rangle \\ CZ\,|1b\rangle &= Z_2\,|1b\rangle = |1\rangle\,(Z\,|b\rangle) \end{aligned}$$

$b=0,1$

Exercise 1: Write the CNOT gate using the CZ gate and one-qubit gates.

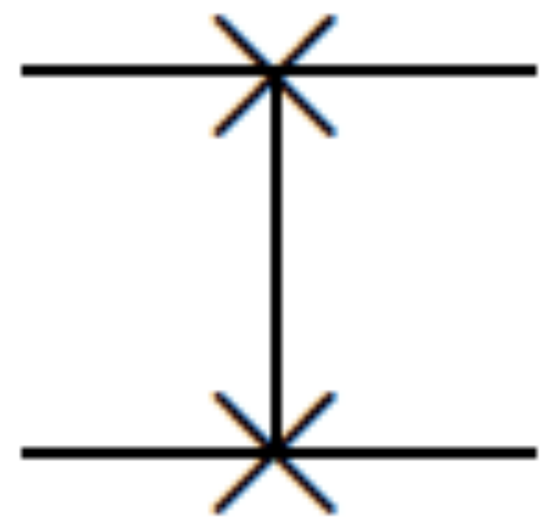
Exercise 2: Use the CNOT gate and one-qubit gates to create the four Bell basis states.

$$|\psi_+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \qquad |\phi_+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

$$|\psi_-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \qquad |\phi_-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

SWAP gate

Symbol



Matrix representation

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Truth table

$$\text{SWAP}\,|00\rangle = |00\rangle$$
$$\text{SWAP}\,|01\rangle = |10\rangle$$
$$\text{SWAP}\,|10\rangle = |01\rangle$$
$$\text{SWAP}\,|11\rangle = |11\rangle$$

or

$$\text{SWAP}\,|ab\rangle = |ba\rangle$$

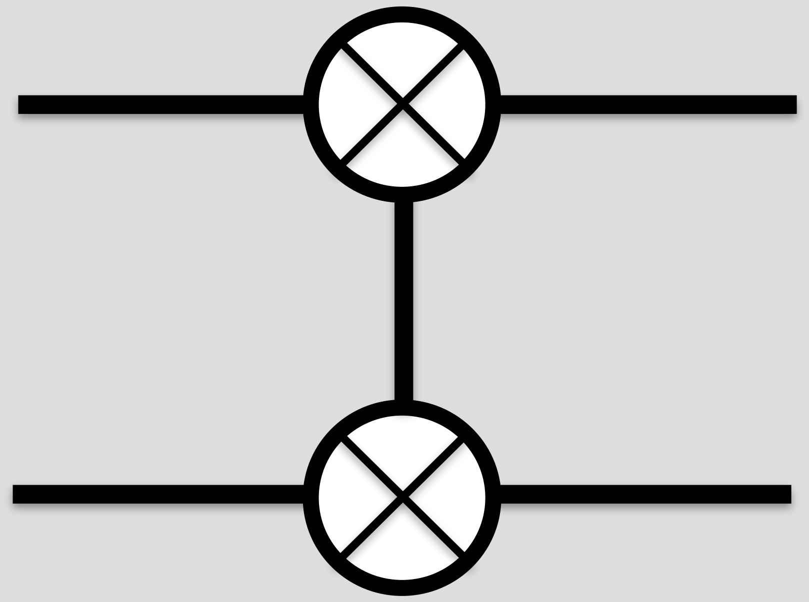
$a,b=0,1$

iSWAP gate

Exercise: The iSWAP gate is given by

$$i\text{SWAP} = R_{XX+YY}\left(\frac{-\pi}{2}\right) = \exp\left[i\frac{\pi}{4}(X \otimes X + Y \otimes Y)\right]$$

Symbol



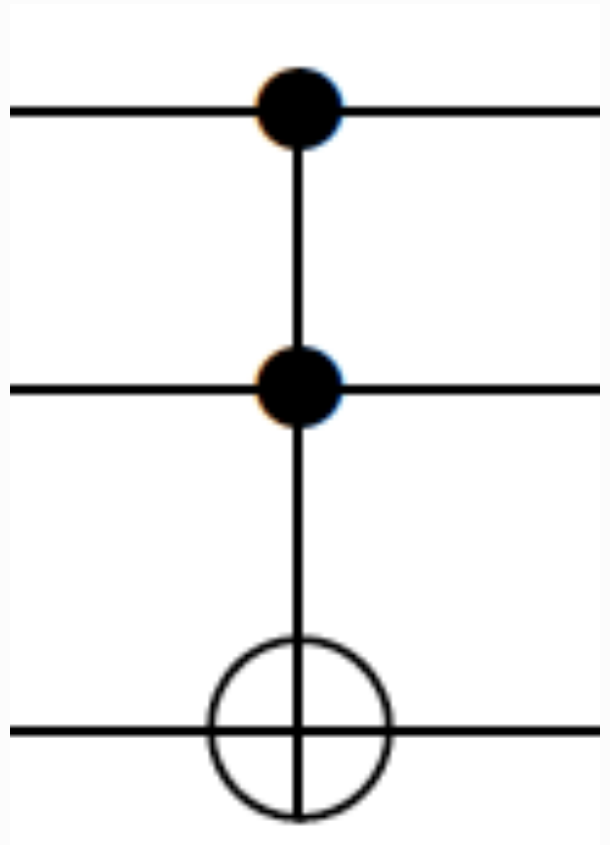
Show that its matrix representation is

$$i\text{SWAP} = = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Three-qubit quantum gates

CCNOT - Toffoli gate

Symbol



Matrix representation

$$\text{CCNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Truth table

$$\text{CCNOT}\,|00c\rangle = |00c\rangle$$
$$\text{CCNOT}\,|01c\rangle = |01c\rangle$$
$$\text{CCNOT}\,|10c\rangle = |10c\rangle$$
$$\text{CCNOT}\,|11c\rangle = |11\overline{c}\rangle$$

c=0,1 ; $\overline{c}$=NOT(c)

or

$$\text{CCNOT}\,|a,b,c\rangle = |a,b,(a\cdot b)\oplus c\rangle$$

a,b,c=0,1

The Toffoli gate is universal for classical computing.

Exercise: Show that the Toffoli gate can be constructed from one-qubit gates plus the CNOT gate. *Hint:* Just do it if you have a computer!