Introduction to quantum computing and simulability

Introduction to computational complexity theory I

Daniel J. Brod

Leandro Aolita

Ernesto Galvão

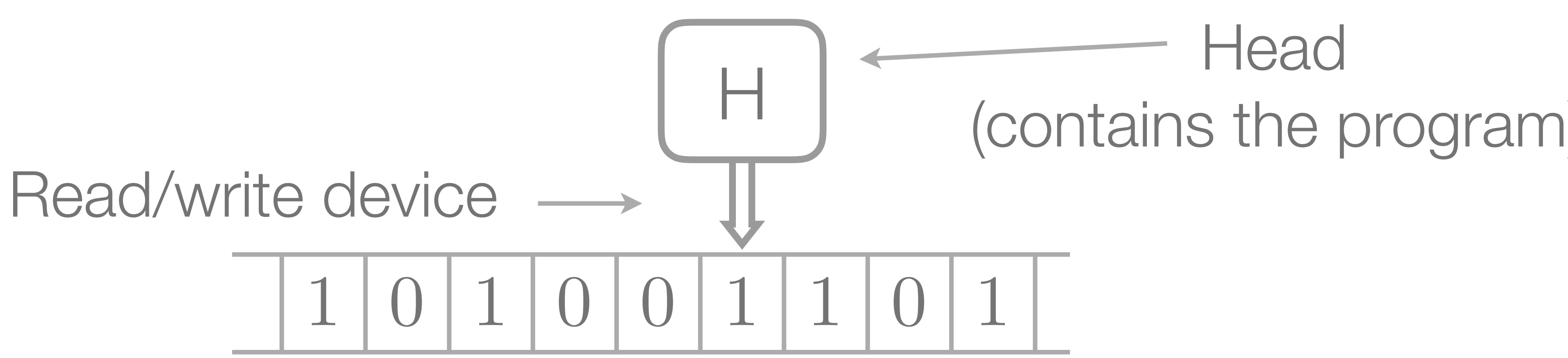
Instituto de Física
Universidade Federal Fluminense

Outline: Computational complexity theory I

- Classical computing 101
- Complexity Classes:
  - **P**;
  - **NP**;
  - Reductions and **NP**-completeness;
  - **BPP** and **BQP**;

Classical computing

- Information encoded in bits (0s and 1s);
- Bits manipulated by Turing machines:



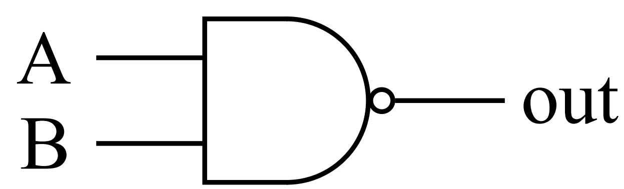
Classical computing

Church-Turing Thesis (physical version)

All computational problems solvable by a realistic physical system can be solved by a Turing machine.

Classical computing

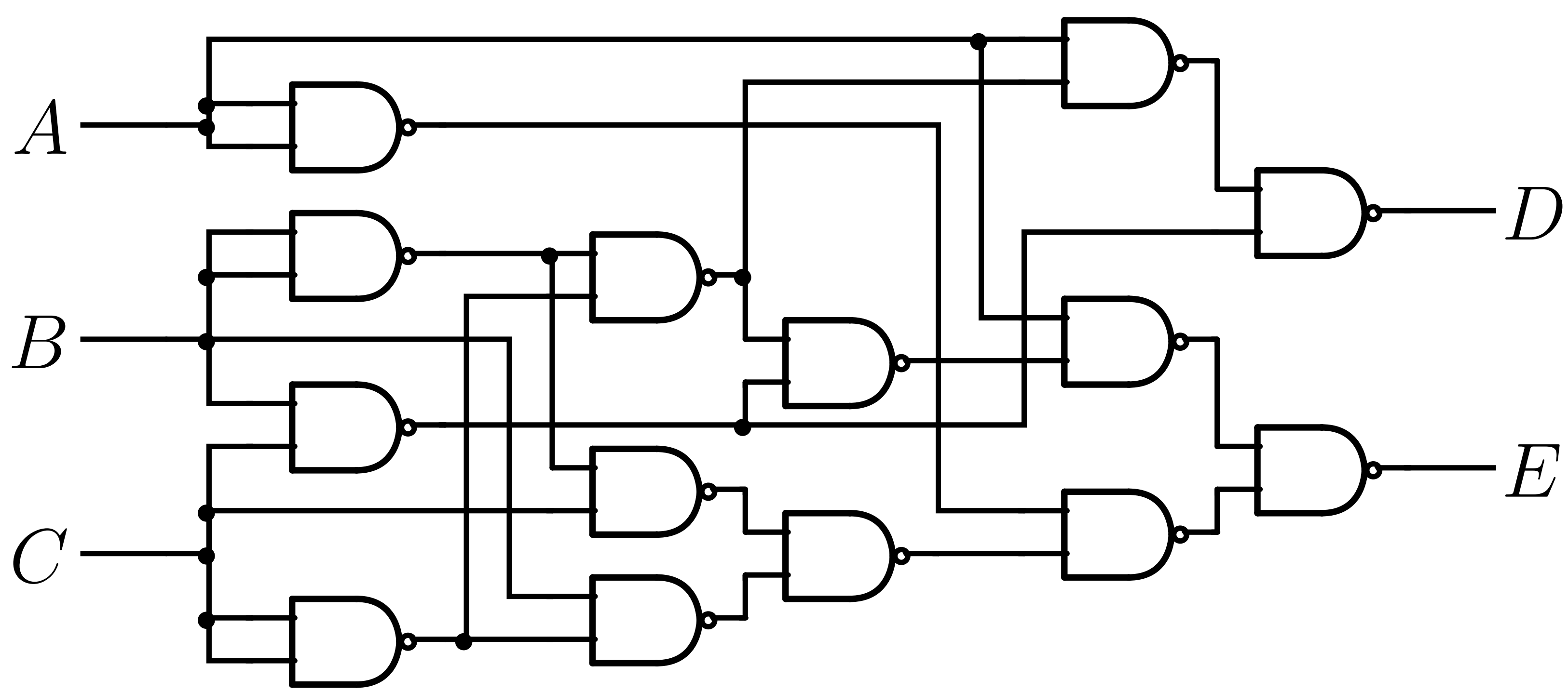
- Information encoded in bits (0s and 1s);
- Bits manipulated in Boolean circuits:



A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

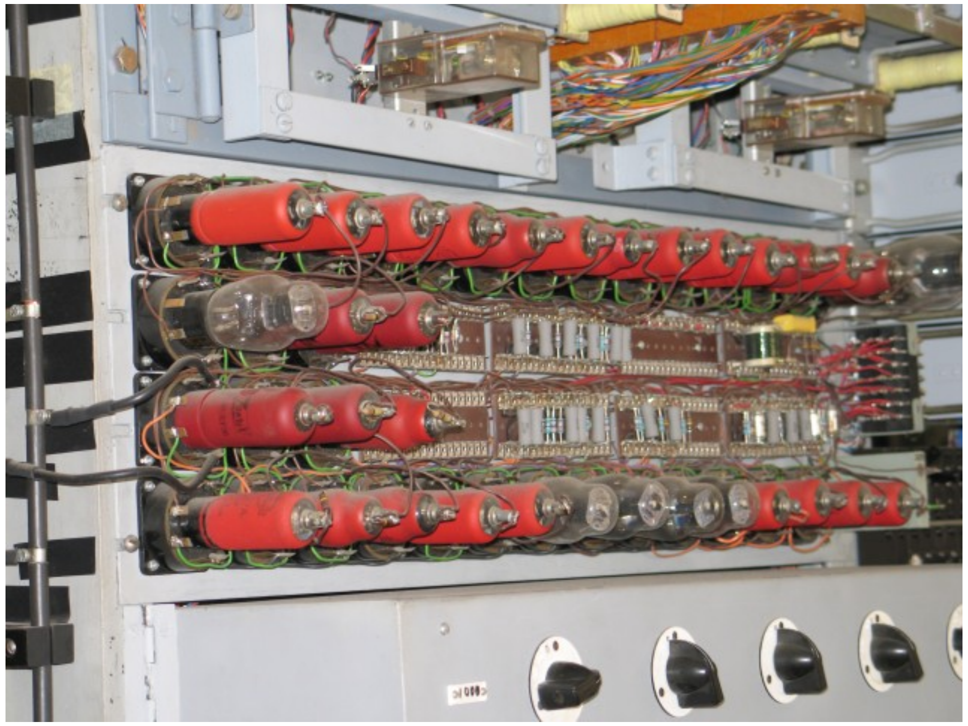
Classical computing

- Information encoded in bits (0s and 1s);
- Bits manipulated in Boolean circuits:



Classical computing

- Physical system is not too important for computability.
  - Vacuum tubes
    - Colossus (1943)



Replica of Colossus, Bletchley Park

Photo: Ian Petticrew

Classical computing

- Physical system is not too important for computability.
  - Transistors
    - Intel® 4004 (1971) - 2.300 transistors.
    - Intel® Core™ (2010) - 560.000.000 transistors.

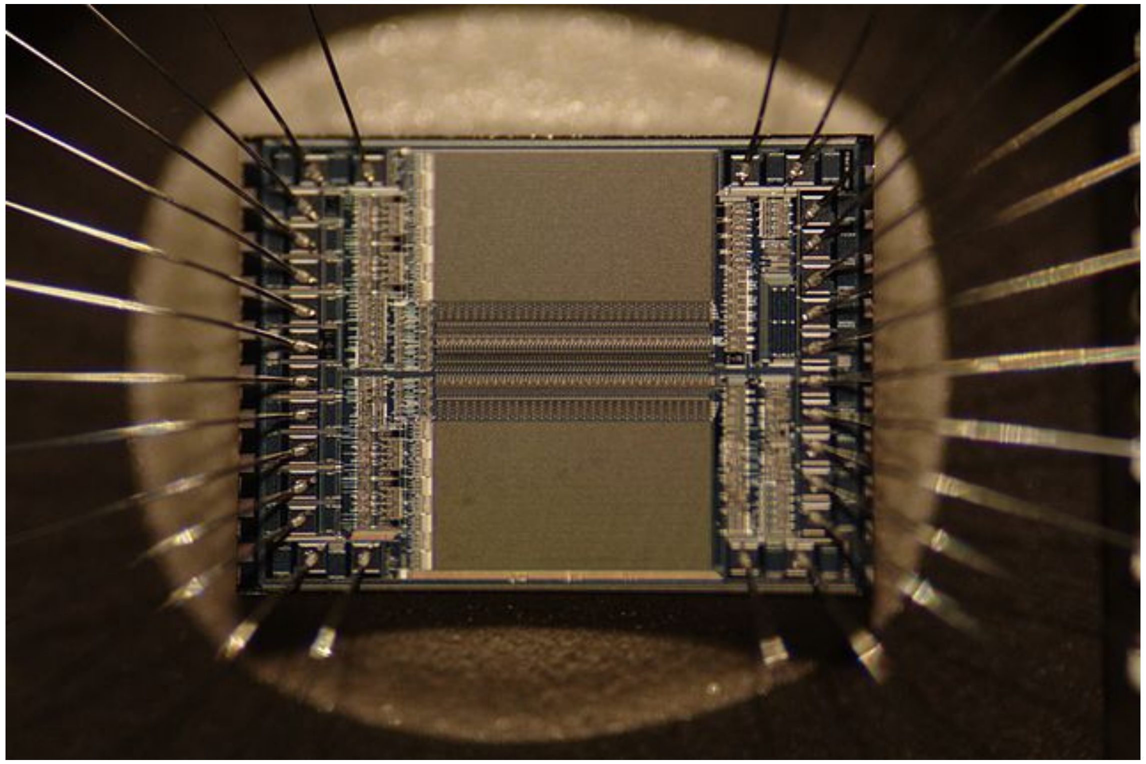


Photo: Wikipedia (Richard Wheeler)

Classical computing

- Physical system is not too important for computability.
  - Billiard balls (Newtonian mechanics)
    - Fredkin e Toffoli (1982 - proposta teórica)

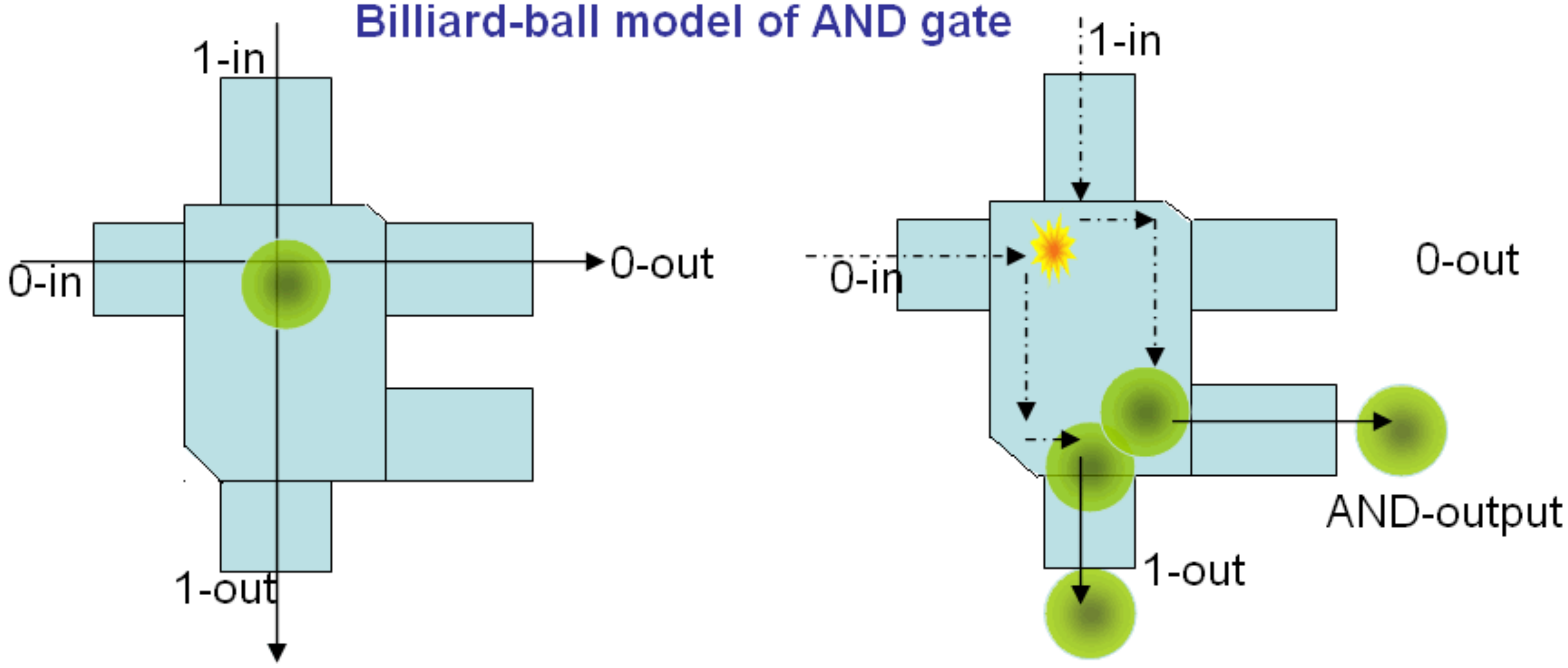


Imagem: Wikipedia

Church-Turing Thesis

Church-Turing Thesis (**Strong** version)

(Informal statement): all realistic physical systems are computationally equivalent.

- What do we mean by "equivalent"?
    - We are interested in **asymptotic** behaviour!

Polynomial vs exponential

- Definition: Efficient computation;
- Consider
  - Some problem $\mathcal{P}$ parameterized by "size" n; and
  - a model of computation $\mathcal{M}$.
- $\mathcal{M}$ solves $\mathcal{P}$ efficiently if there is an algorithm in $\mathcal{M}$ to solve $\mathcal{P}$ in time that grows as a polynomial (in n).
- Otherwise, $\mathcal{M}$ does not solve $\mathcal{P}$ efficiently.
  - e.g. if best possible algorithm for $\mathcal{P}$ in $\mathcal{M}$ takes **exponentially** long.

Polynomial vs exponential

- Why polynomial vs. exponential?
  - Asymptotically, exponentials grow faster than polynomials.

$n =$	10	100
$100n$	1000	10000
2^n	1024	1267650600228229401496703205376

- What about n^{100} and 1.001^n ?
  - Asymptotically efficient not always the same as efficient in practice.
  - Extreme polynomials not very common, tend to improve with time.

Polynomial vs exponential

Definition: big-O notation.

A function is $\mathrm{O}(f(n))$ if its leading term grows as $f(n)$ or slower.

e.g.: all functions below are $\mathrm{O}(n^2)$

$$n^2$$

$$n^2 + n$$

$$n$$

$$n^2 + \log n$$

$$n^2 + 10000n$$

Church-Turing Thesis

Church-Turing Thesis (physical version)

All computational problems solvable by a realistic physical system can be solved by a Turing machine.

Church-Turing Thesis (**Strong** version)

Any problem that can be solved **efficiently** by a realistic computational device can be solved **efficiently** by a Turing machine.

Outline: Computational complexity theory I

- Classical computing 101
- Complexity Classes:
    - **P**;
    - **NP**
    - Reductions and **NP**-completeness;
    - **BPP** and **BQP**;

Decision problems

Definition: Decision problem.

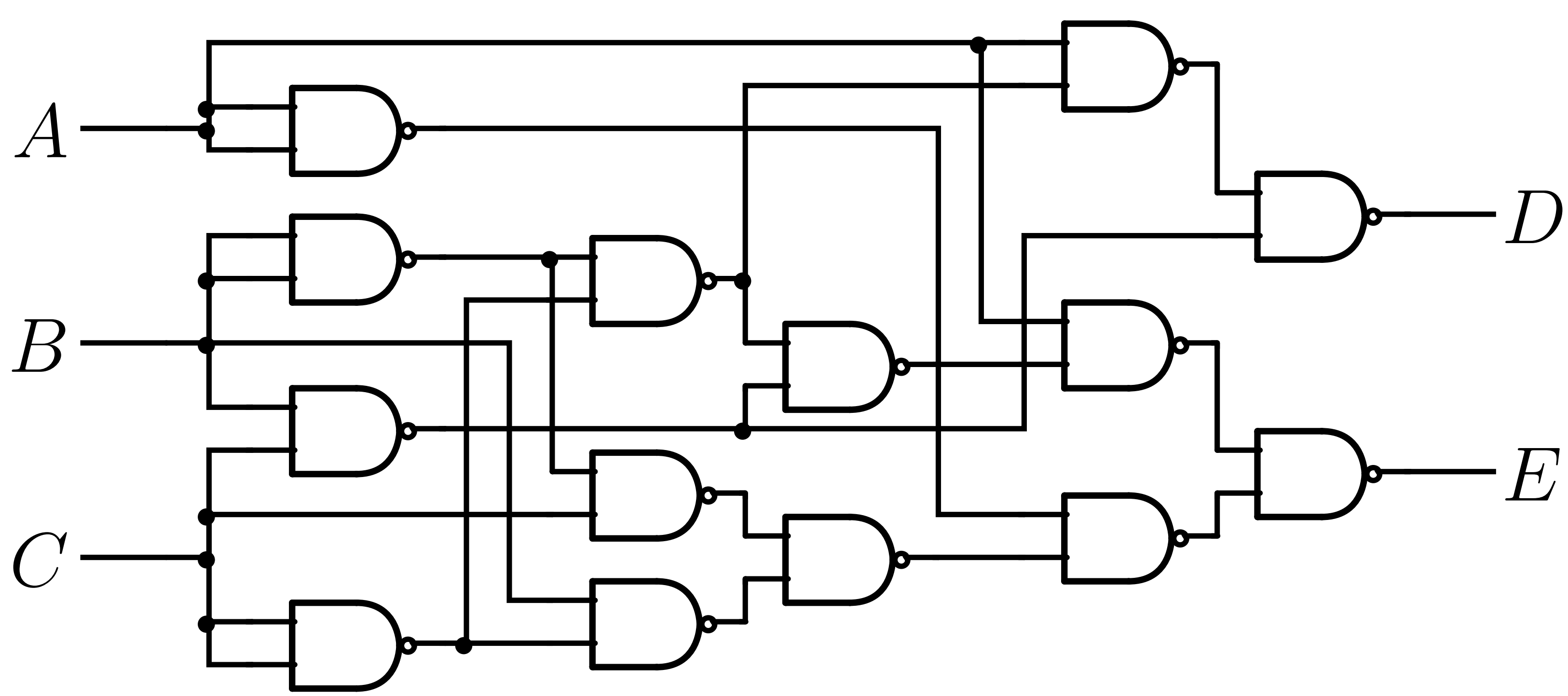
(informal) A decision problem is a YES/NO question!

- Ex (Primality testing): "Is x prime?"

Complexity classes: **P**

Definition: P (complexity class)

(informal) Decision problems that can be solved efficiently by classical computers.



Complexity classes: **P**

Definition: **P** (complexity class)

(formal) A problem is in **P** if and only if there is a uniform family of efficient classical circuits* such that, for all n-bit inputs x,

- In a YES instance the circuit outputs 1;
- In a NO instance the circuit outputs 0;

* Uniform family of efficient classical circuits:

- depend **only** on size n of input;
- have at most poly(n) gates;
- can be described in poly(n) time

Complexity classes: **P** - Examples

- Multiplying n x n matrices;
- Computing the determinant of n x n matrices;
- Finding the greatest common divisor of two n-digit numbers;
- Deciding if an n-digit number is prime;
- Many others!

Complexity classes: **P** - Examples

- Claim: Computing the determinant of an n x n matrix M is in **P**.
- Reasoning:
    1. Determinant is not a decision problem! ✔

Complexity classes: **P** - Examples

- Claim: Computing the determinant of an n x n matrix M is in **P**.
- Reasoning:
  1. Determinant is not a decision problem! ✔
  2. Compute from definition?

$$\det M = \underbrace{\sum_{\sigma \in S_n}}_{\text{This sum has } n! \text{ terms}} \left(\operatorname{sgn}(\sigma) \prod_{i=1}^{n} m_{i,\sigma_i} \right)$$

This sum has $n!$ terms ☹

Complexity classes: **P** - Examples

- Claim: Computing the determinant of an n x n matrix M is in **P**.
- Reasoning:
  1. Determinant is not a decision problem! ✔
  2. Computing from definition is no good. ✘
  3. Use a shortcut: $\det AB = \det A \det B$ ✔

$$M = \underbrace{\begin{pmatrix} a_1 & b_1 & 0 & & 0 \\ c_1 & d_1 & 0 & \dots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & & 1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & 0 & b_2 & & 0 \\ 0 & 1 & 0 & \dots & 0 \\ c_2 & 0 & d_2 & & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & & 1 \end{pmatrix} \dots}_{O(n^2) \text{ matrices}}$$

Complexity classes: **P** - Examples

- Claim: Computing the determinant of an n x n matrix M is in **P**.
- Reasoning:
  1. Determinant is not a decision problem! ✔
  2. Computing from definition is no good. ✘
  3. Use a shortcut: $\det AB = \det A \det B$ ✔
  4. Running time using shortcut + Gaussian elimination: $O\left(n^3\right)$

Outline: Computational complexity theory I

- Classical computing 101
- Complexity Classes:
  - **P**;
  - **NP**
  - Reductions and **NP**-completeness;
  - **BPP** and **BQP**;

Complexity classes: **NP**

Definition: **NP** (complexity class)

(informal) Decision problems whose solution can be **checked** efficiently by classical computers.

- Example: Factoring

Hard

$$67030883744037259 = 179424673 \times 373587883$$

Easy

Complexity classes: **NP**

Definition: **NP** (complexity class)

(formal) A problem is in **NP** if and only if there is a uniform family of efficient classical circuits that takes as inputs an n-bit string x and a witness y such that

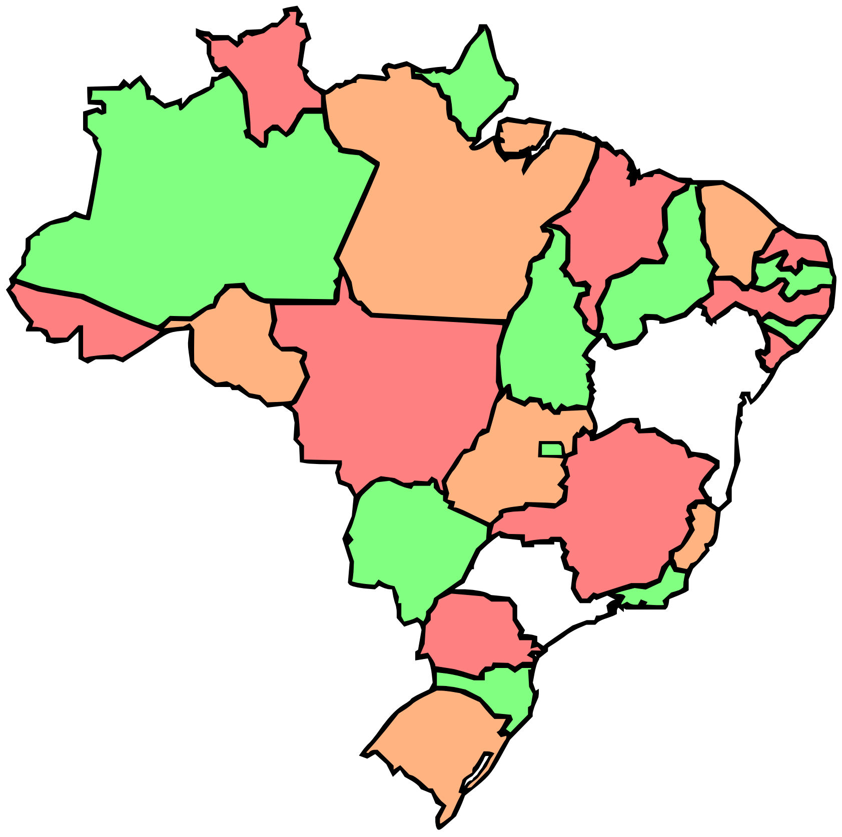
- In the YES instance, there is y of length poly(n) such that the circuit outputs 1;
- In the NO instance, for all y of length poly(n) the circuit outputs 0;

Complexity classes: **NP** - Examples

- Travelling salesman
  - Given a list of n cities, is there a path that visits all of them and is shorter than some length x?

Complexity classes: **NP** - Examples

- 3-Coloring
  - Can a map with n regions be painted with only 3 colors such that no neighbors have the same color?



(the answer is always yes for 4 colors!)

Complexity classes: Reductions

- Consider the following two **NP** problems:

Definition: **3-SAT**

Let $\{x_1, x_2 \ldots x_n\}$ be a set of n true/false variables. Let Φ be a formula of the type

$$\Phi = (x_1 \vee x_2 \vee x_5) \wedge (x_2 \vee \neg x_3 \vee x_6) \wedge \cdots$$

Can we set $x_1, x_2, \ldots, x_n$ to true/false such that Φ is true?

- Examples:

$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$

Complexity classes: Reductions

- Consider the following two **NP** problems:

Definition: 3-SAT

Let $\{x_1,\ x_2 \ldots\ x_n\}$ be a set of n true/false variables. Let Φ be a formula of the type

$$\Phi = (x_1 \vee x_2 \vee x_5) \wedge (x_2 \vee \neg x_3 \vee x_6) \wedge \cdots$$

Can we set $x_1,\ x_2, \ldots,\ x_n$ to true/false such that Φ is true?

- Examples:

$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$

$$x_2 = x_3 = x_4 = \top \quad ✔$$

Complexity classes: Reductions

- Consider the following two **NP** problems:

> Definition: **3-SAT**
>
> Let $\{x_1,\ x_2 \dots\ x_n\}$ be a set of n true/false variables. Let Φ be a formula of the type
>
> $$\Phi = (x_1 \vee x_2 \vee x_5) \wedge (x_2 \vee \neg x_3 \vee x_6) \wedge \cdots$$
>
> Can we set $x_1,\ x_2,\dots,\ x_n$ to true/false such that Φ is true?

- Examples:

$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge$$
$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

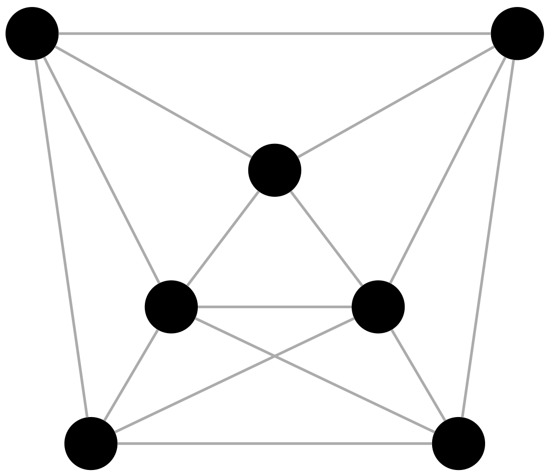
✘

Complexity classes: Reductions

- Consider the following two **NP** problems:

> Definition: k-**Clique**
>
> Does a graph of n vertices have a clique (i.e. a complete subgraph) of size k?

- Example: $k = 4$

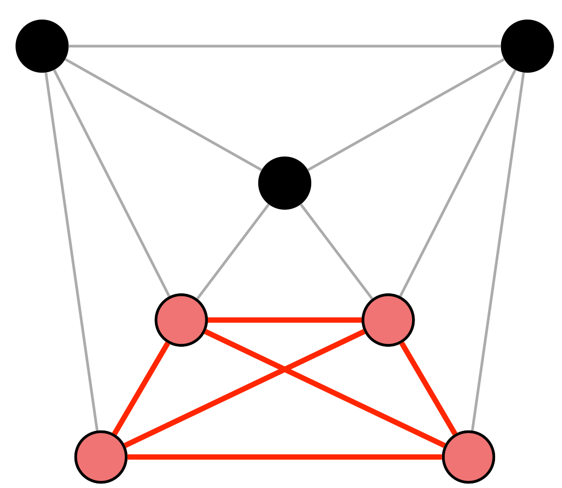


Complexity classes: Reductions

- Consider the following two **NP** problems:

> Definition: k-**Clique**
>
> Does a graph of n vertices have a clique (i.e. a complete subgraph) of size k?

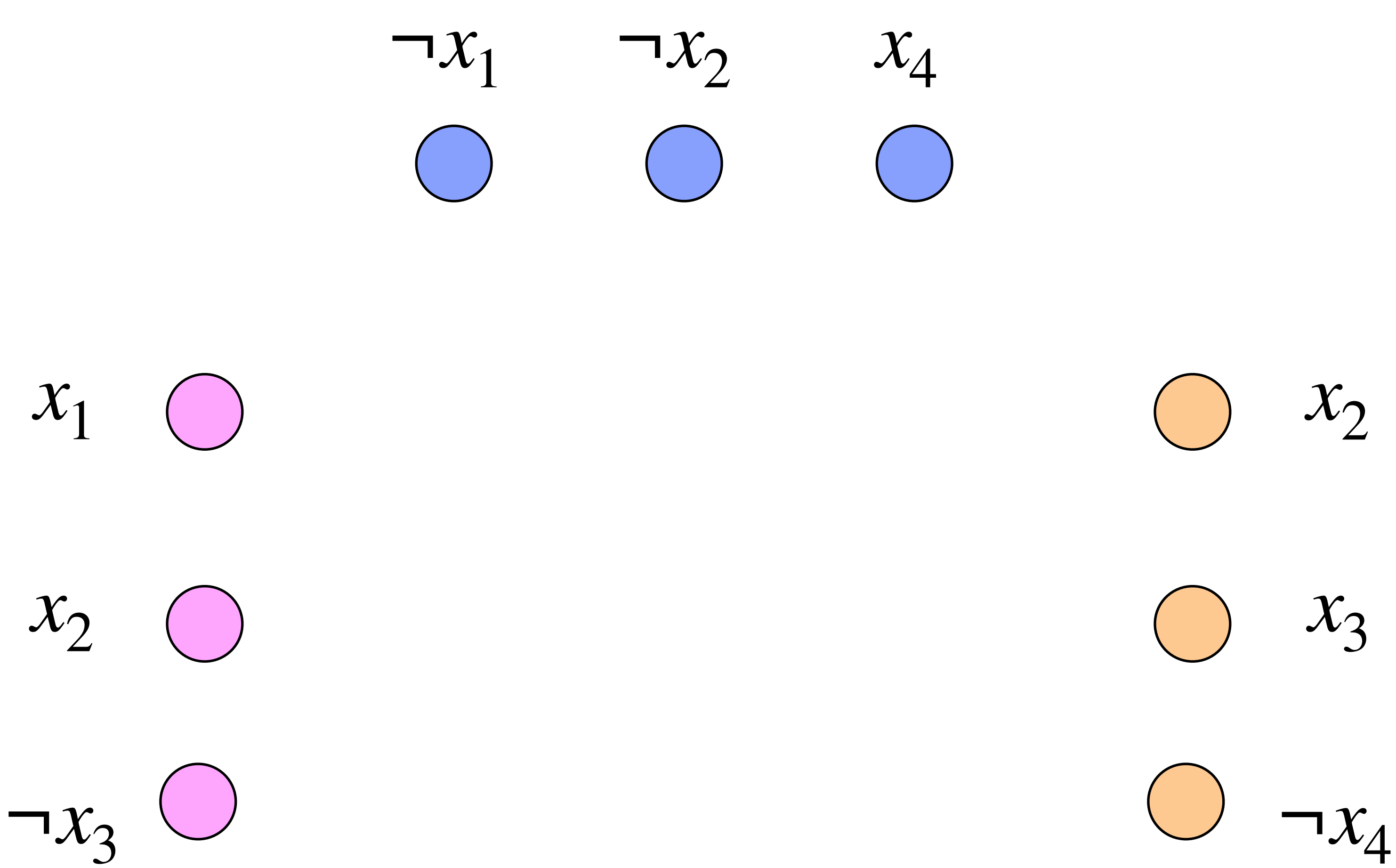
- Example: $k = 4$



Complexity classes: Reductions

- **3-SAT** vs. k-**Clique**: What do they have in common?
  - Consider the following **3-SAT** instance:

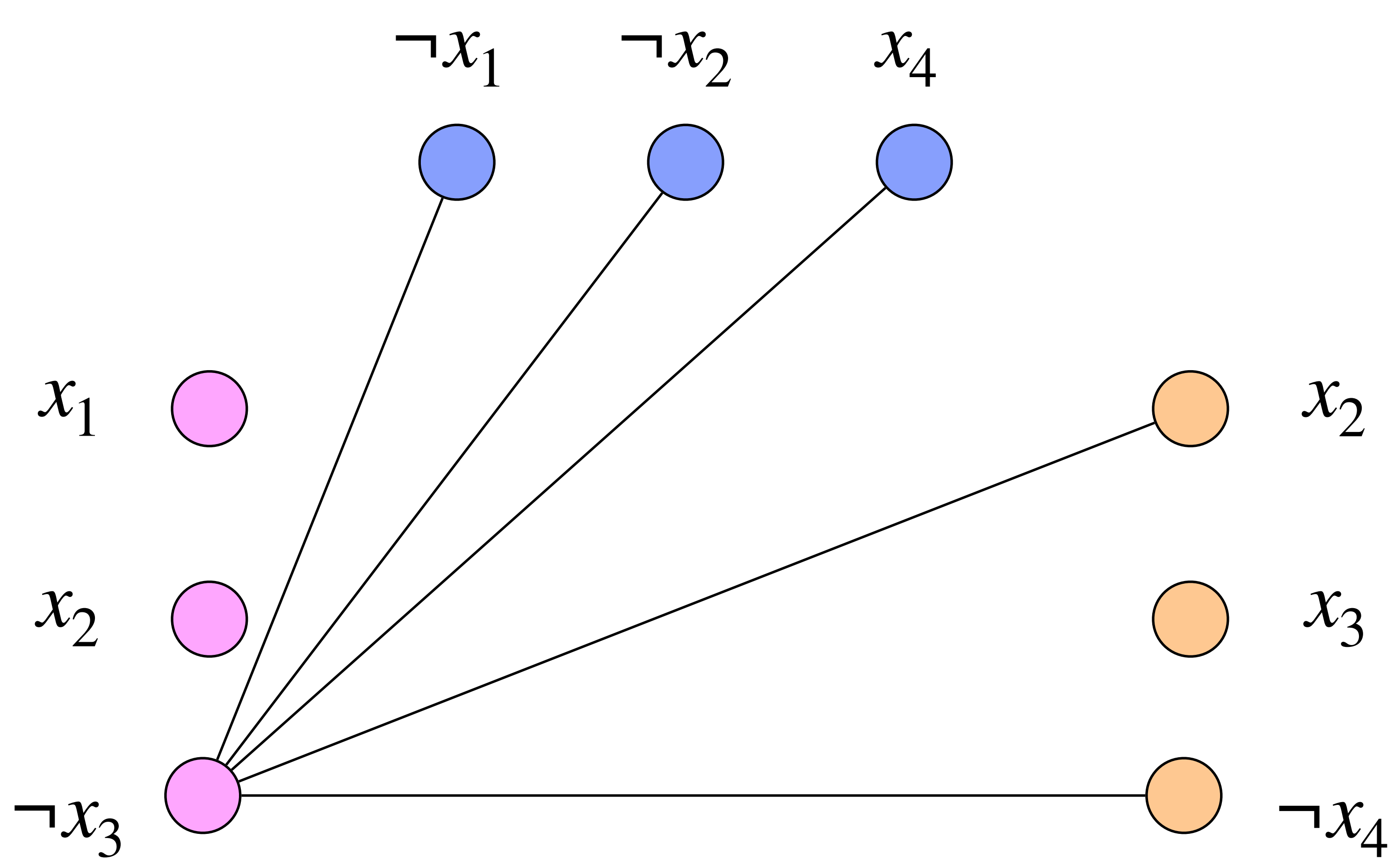
$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$



Complexity classes: Reductions

- **3-SAT** vs. k-**Clique**: What do they have in common?
  - Consider the following **3-SAT** instance:

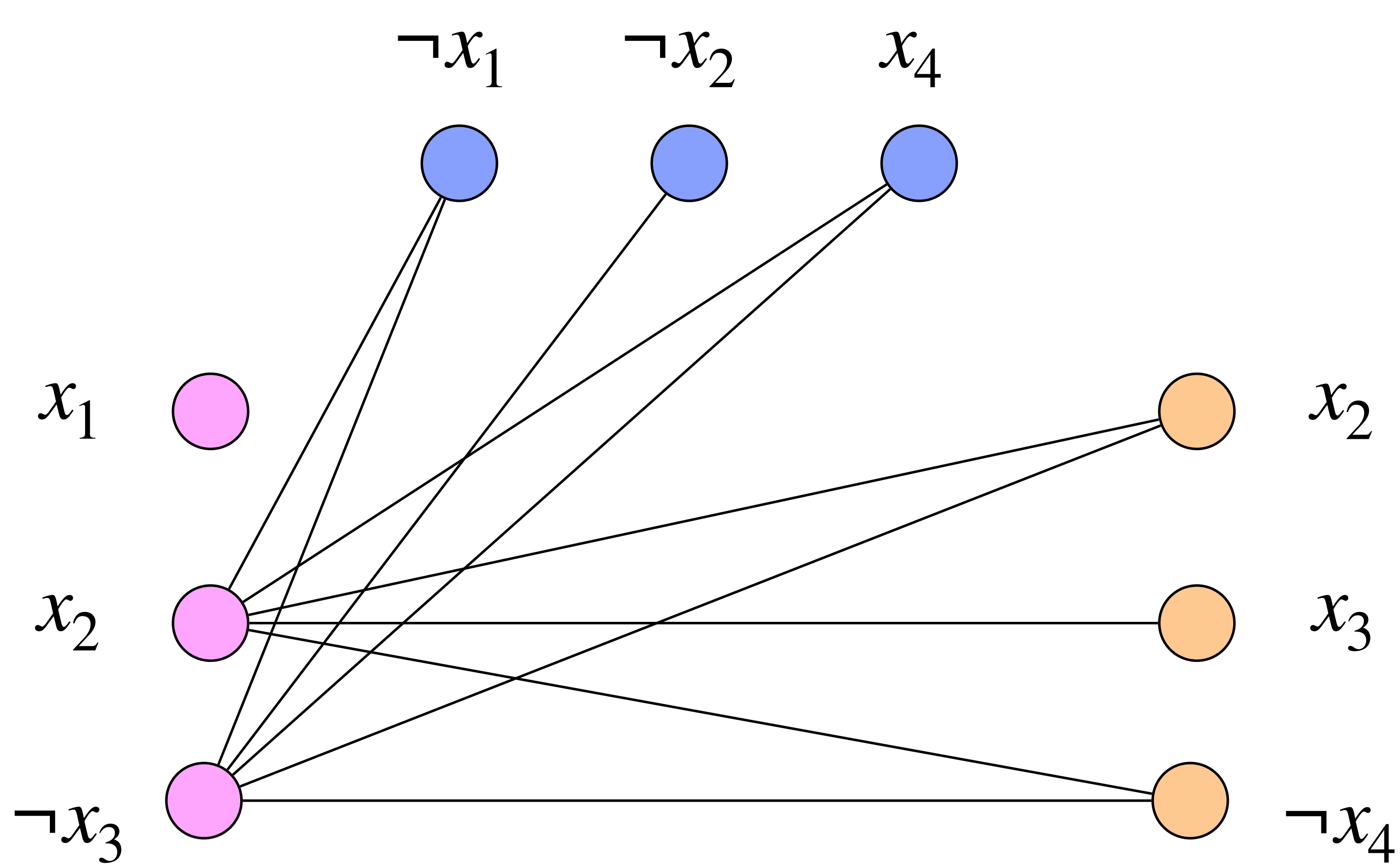
$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$



Complexity classes: Reductions

- **3-SAT** vs. k-**Clique**: What do they have in common?
  - Consider the following **3-SAT** instance:

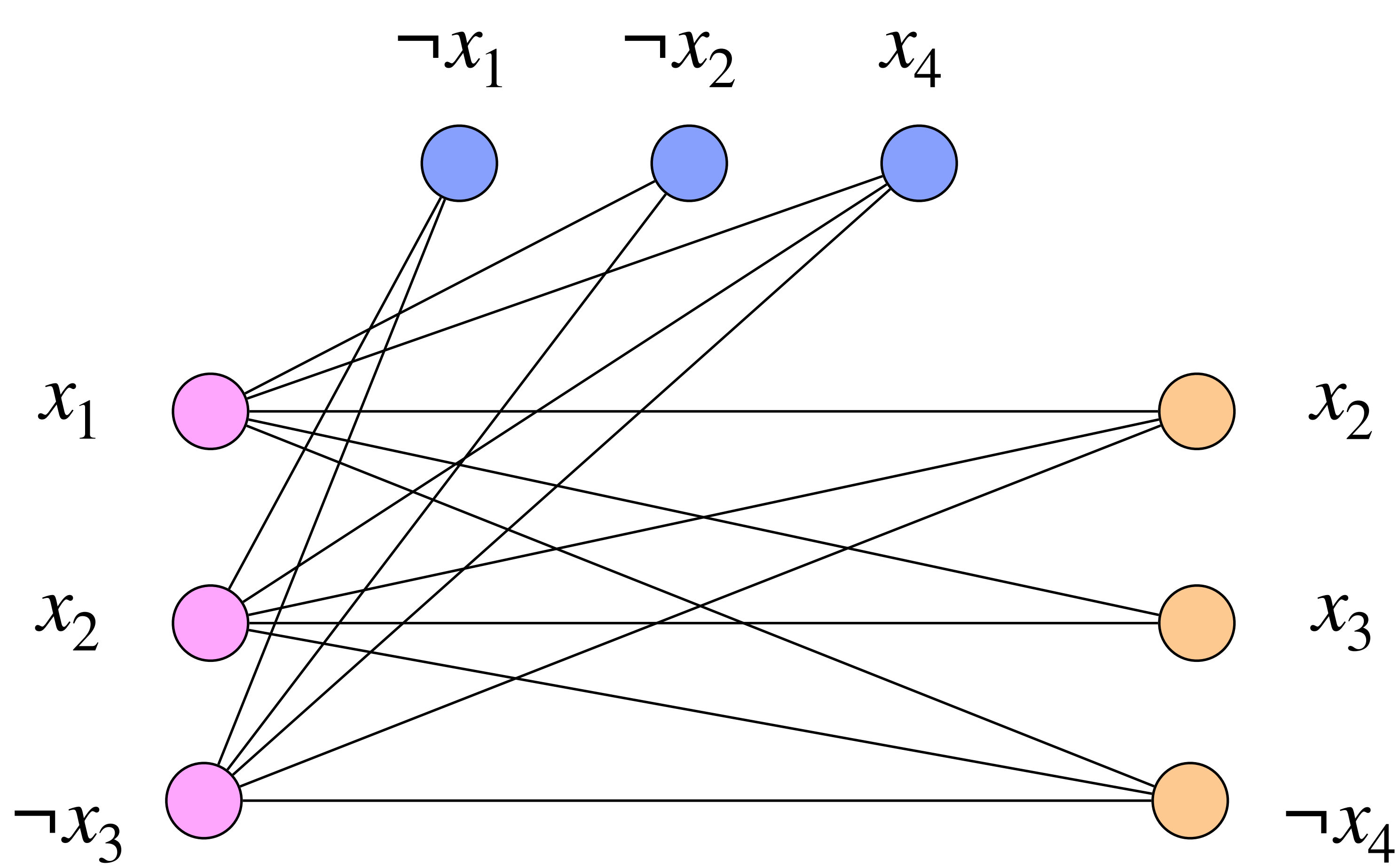
$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$



Complexity classes: Reductions

- **3-SAT** vs. k-**Clique**: What do they have in common?
  - Consider the following **3-SAT** instance:

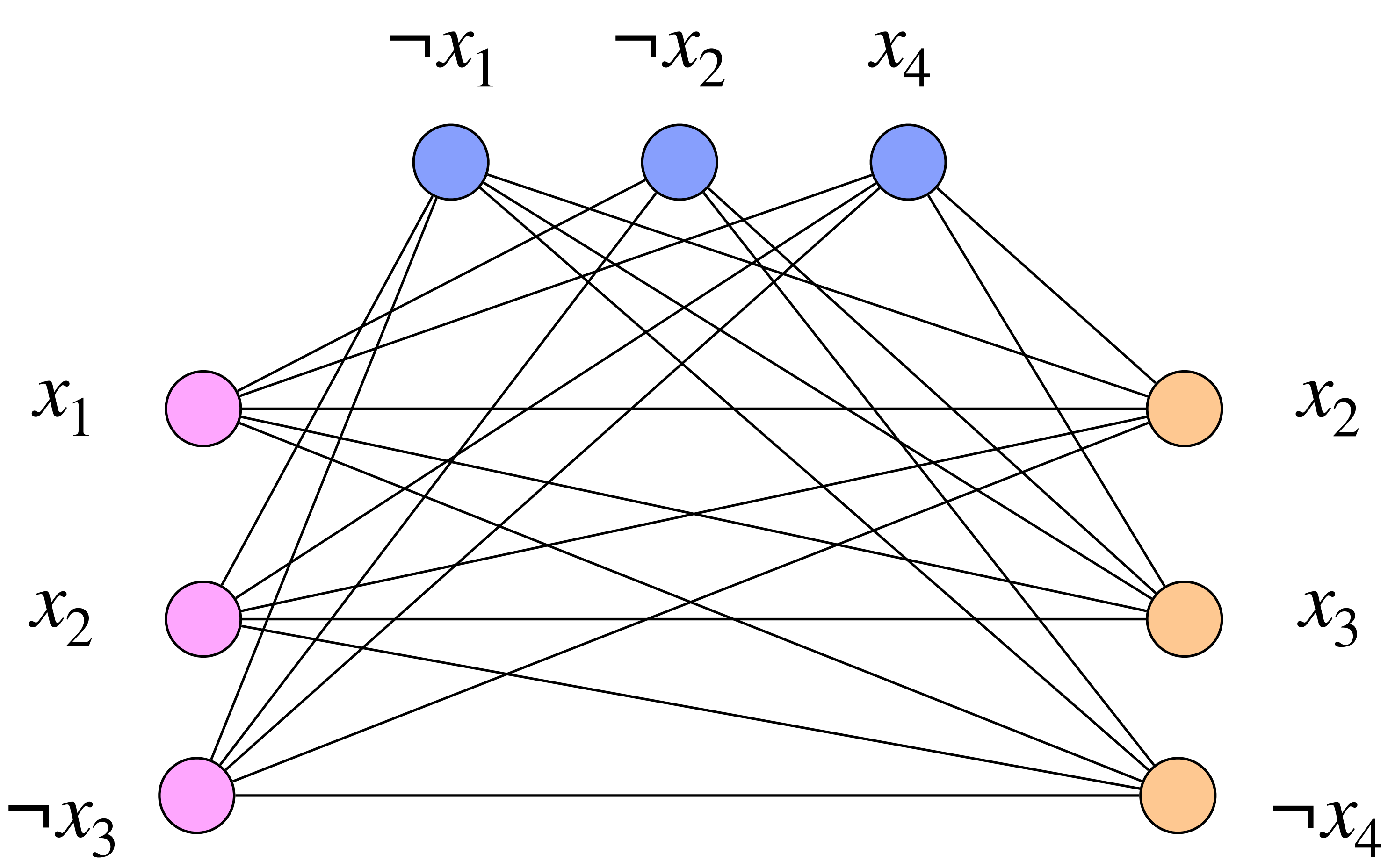
$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$



Complexity classes: Reductions

- **3-SAT** vs. k-**Clique**: What do they have in common?
  - Consider the following **3-SAT** instance:

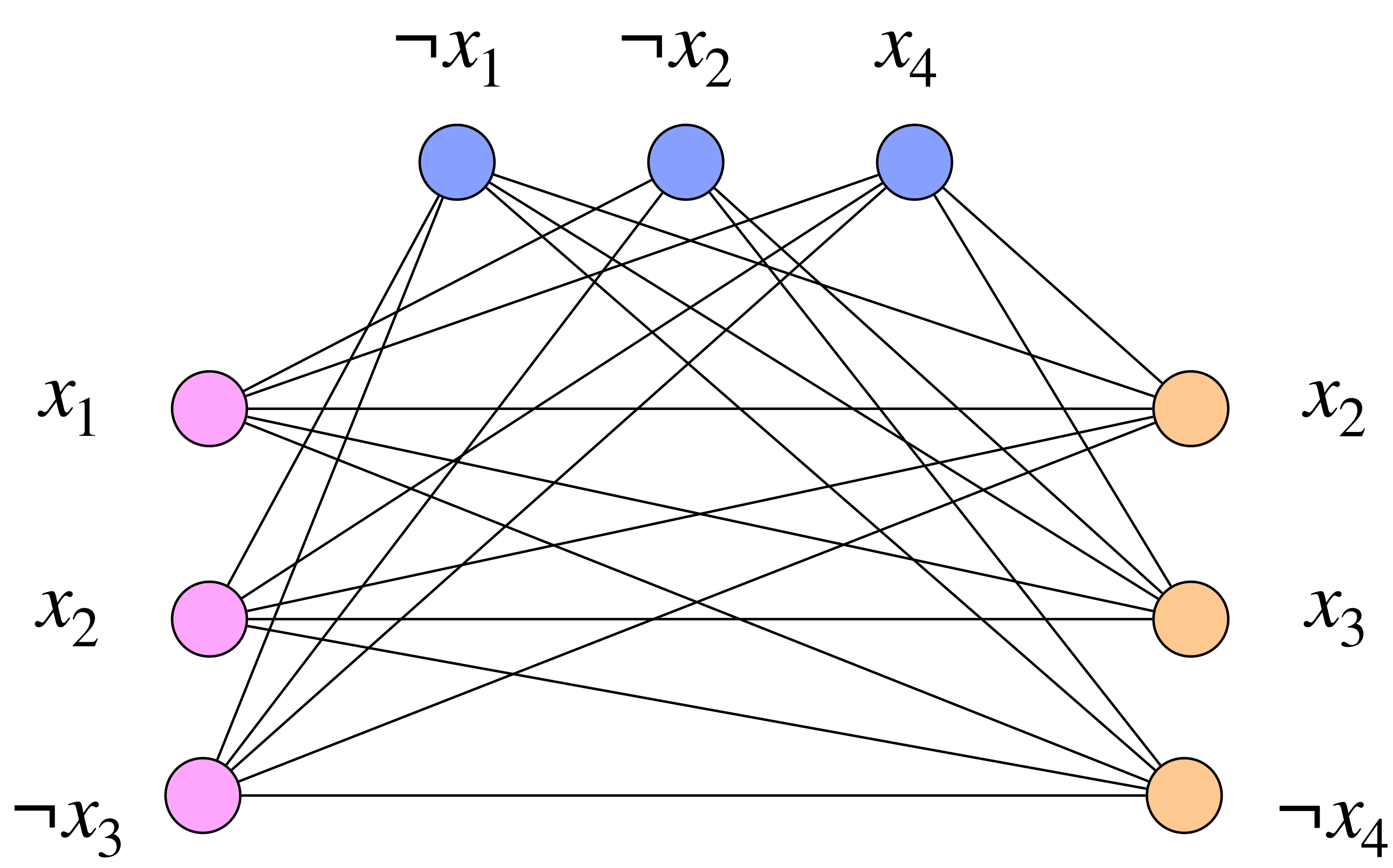
$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$



Complexity classes: Reductions

- **3-SAT** vs. k-**Clique**: What do they have in common?
    - Consider the following **3-SAT** instance:

$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$

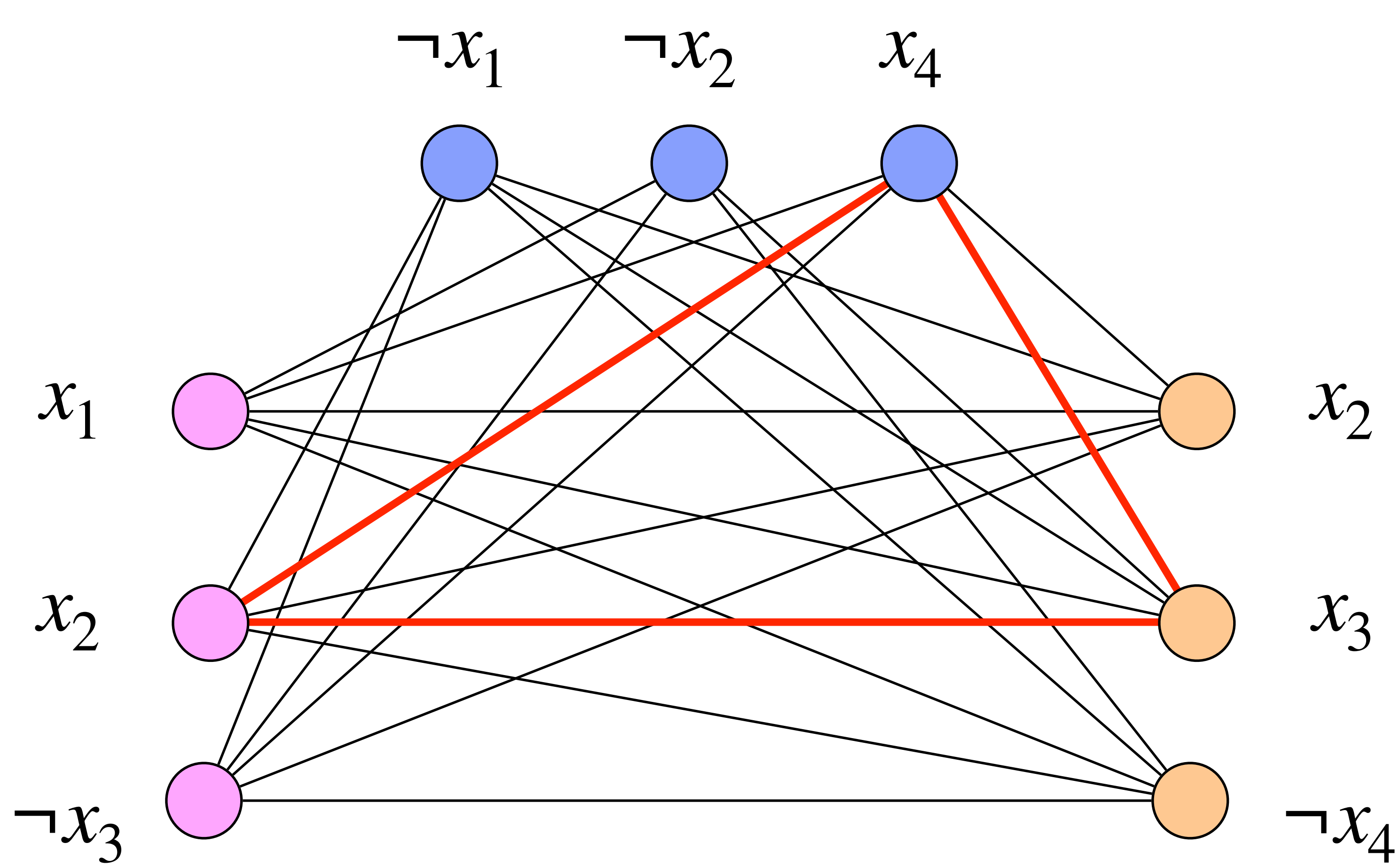


If this graph has a **3-clique** the formula can be satisfied!

Complexity classes: Reductions

- **3-SAT** vs. k-**Clique**: What do they have in common?
  - Consider the following **3-SAT** instance:

$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$



If this graph has a **3-clique** the formula can be satisfied!

$x_2 = x_3 = x_4 = \top$

Complexity classes: Reductions

Interesting conclusion

This **3-SAT** instance can be solved via a **3-Clique** instance.

Very interesting conclusion

Any n-clause **3-SAT** instance can be solved via a n-**Clique** instance. The mapping between the questions and corresponding answers can be done in poly(n) time.

Complexity classes: Reductions

Definition: Reduction

(Informal) Problem **A** reduces to problem **B** if an algorithm for **B** can be used to find a solution fo **A**, and the mapping between them can be done efficiently.

Intuitively, this says **B** is at least **as hard as A**.

Example: **3-SAT** reduces to k-**Clique**.

Complexity classes: Reductions

Mind-blowing conclusion

Every problem in **NP** reduces to **3-SAT!**

Complexity classes: Reductions

Definition: **NP-complete**

(Informal) A problem is **NP-hard** if any other **NP** problem reduces to it.

It is also **NP-complete** if it is in **NP** and is **NP-hard**.

Cook-Levin Theorem (1971/1973)

3-SAT is **NP-complete**.

Complexity classes: **NP** - more examples

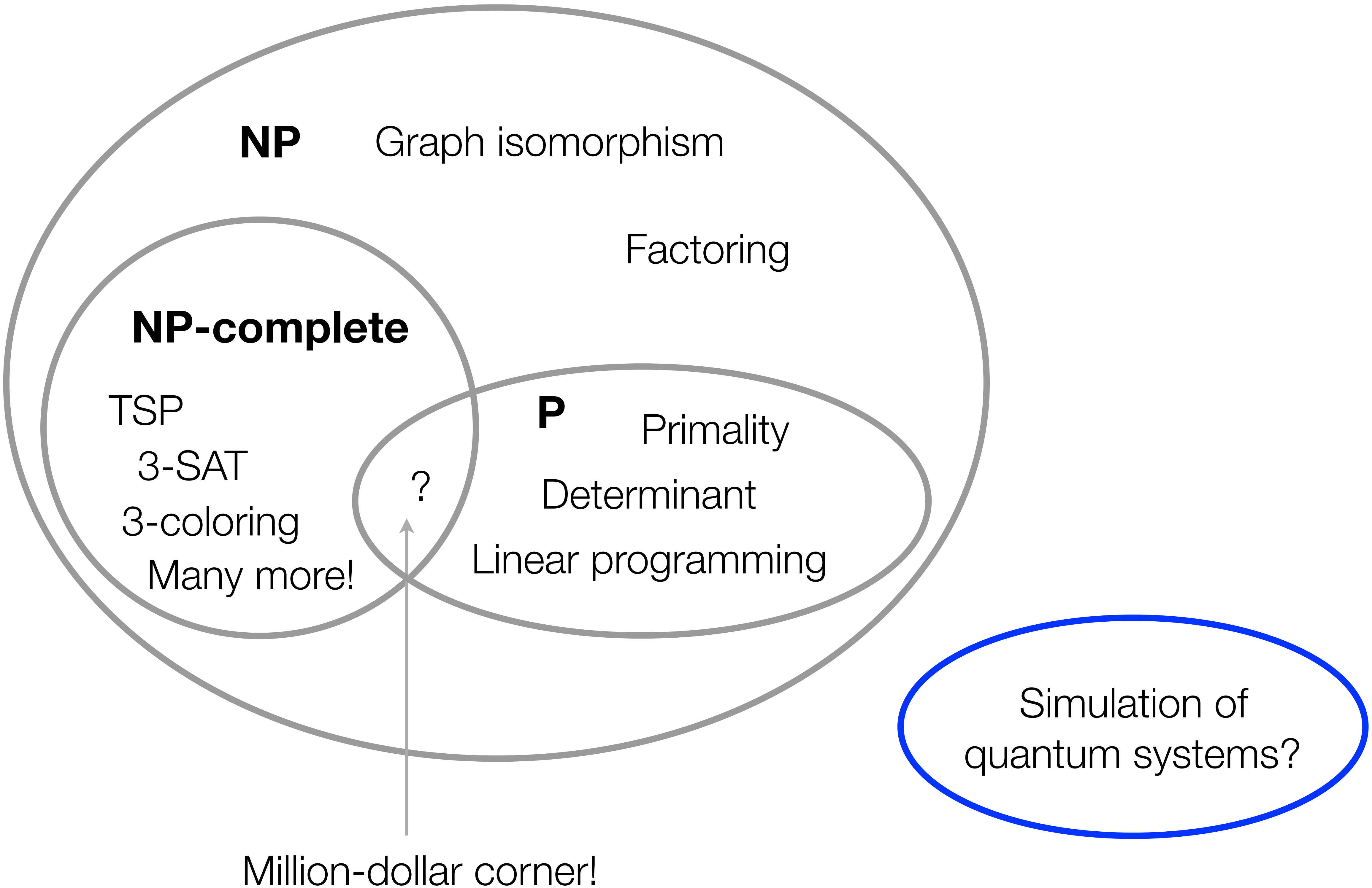
- **Hamiltonian cycle**: In a graph of n vertices, is there a cycle that visits each vertex exactly once?
- **Subset sum**: Given a collection of n integers, is there a subset of them that sums to 0?
- **Graph isomorphism**: Are two n-vertex graphs identical up to relabelling?
- Protein folding, vehicle routing, scheduling.
- Sudoku, tetris and Minesweeper
- A **huge** number of others!
    - Of the **NP** problems listed so far, only **Factoring** and **Graph isomorphism** are <u>not</u> **NP**-complete!

Complexity classes: **P** vs.**NP**

Is every problem in **NP** also in **P**?

- One of the main open questions in mathematics today!
  - Worth 1 million dollars! (really!)
- **Really** hard question!
  - It would take a single efficient algorithm for a single **NP**-complete problem to prove **P** = **NP**. No such algorithm has been found.
  - Most complexity theorists believe the answer is **no.**

Complexity classes



Outline: Computational complexity theory I

- Classical computing 101
- Complexity Classes:
  - **P**;
  - **NP**
  - Reductions and **NP**-completeness;
  - **BPP** and **BQP**;

Complexity classes: **BQP**

Recall…

Definition: **P** (complexity class)

(formal) A problem is in **P** if and only if there is a uniform family of efficient classical circuits such that, for all n-bit input x,

- In a YES instance the circuit outputs 1;
- In a NO instance the circuit outputs 0;

Complexity classes: **BQP**

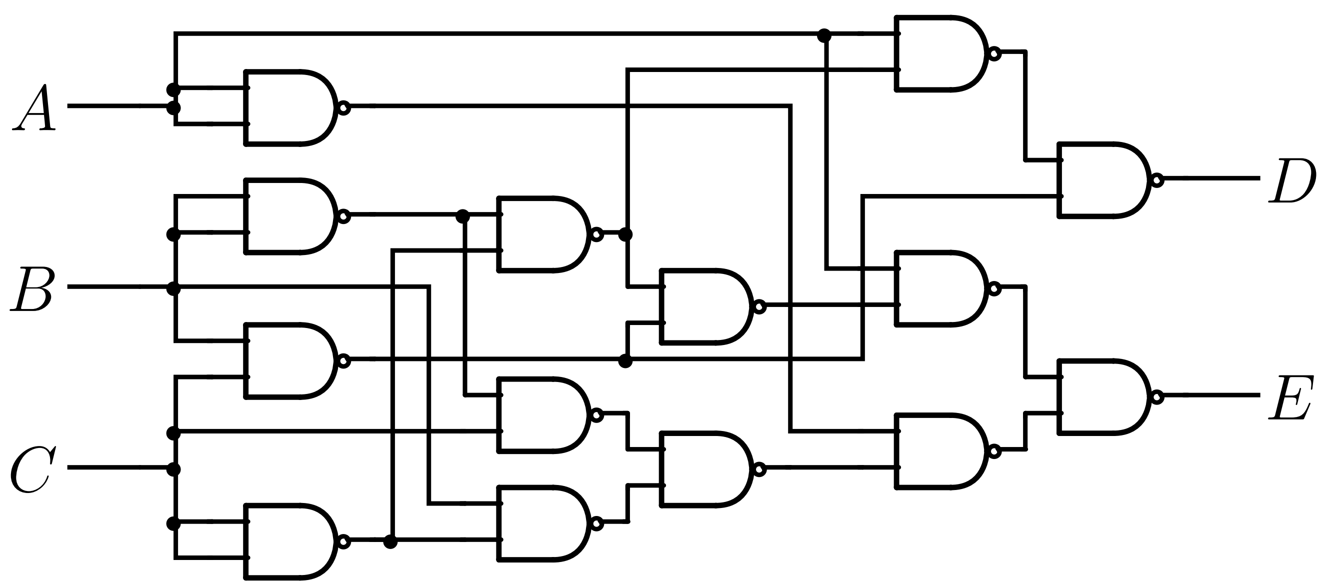
Definition: **BPP** (complexity class)

(formal) A problem is in **BPP** if and only if there is a uniform family of efficient classical circuits such that, for all n-bit input x,

- The circuits have access to a source of random bits;
- In a YES instance the circuit outputs 1 with probability > 2/3;
- In a NO instance, the circuit outputs 0 with probability > 2/3;

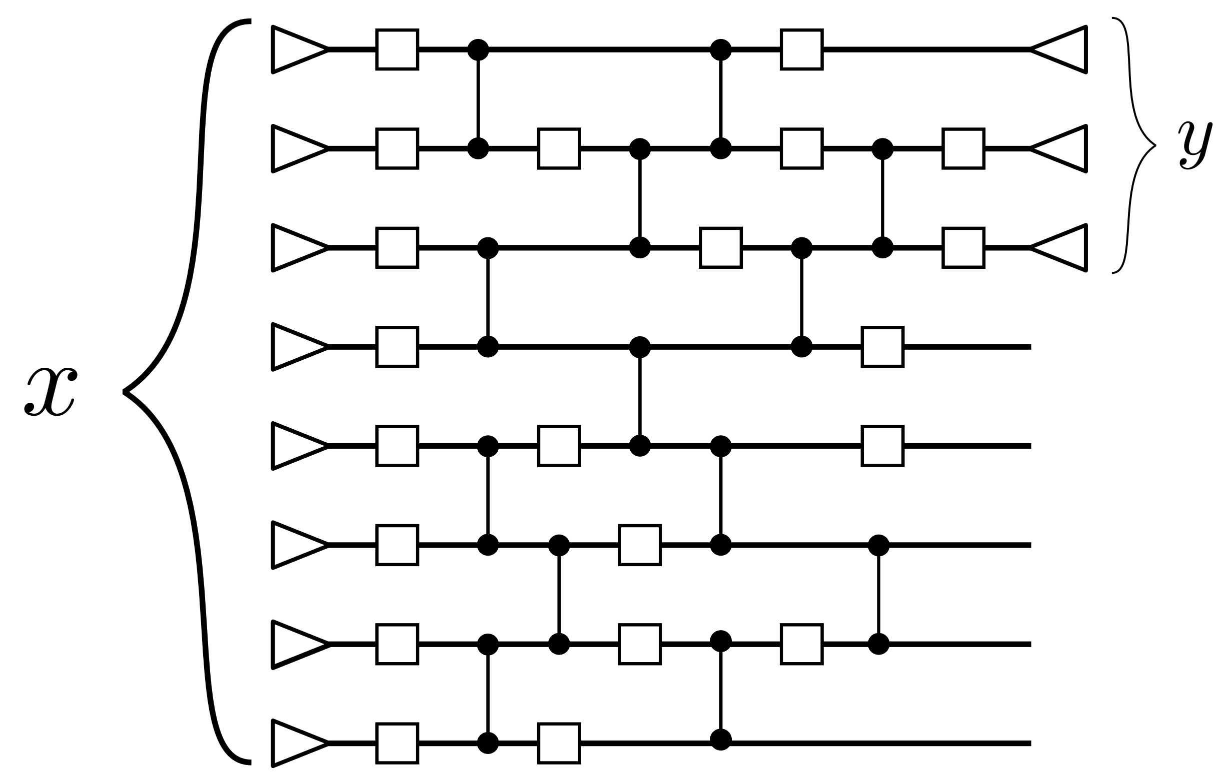
* Computer scientists believe **BPP** = **P**, although there are problems in **BPP** currently not known to be in **P.**

Complexity classes: **BQP**



P
(or **BPP** if we have random bits)

BQP



Complexity classes: **BQP**

Definition: **BQP** (complexity class)

(formal) A problem is in **BQP** if and only if is exists a uniform family of efficient quantum circuits such that, for all n-qubit input x,

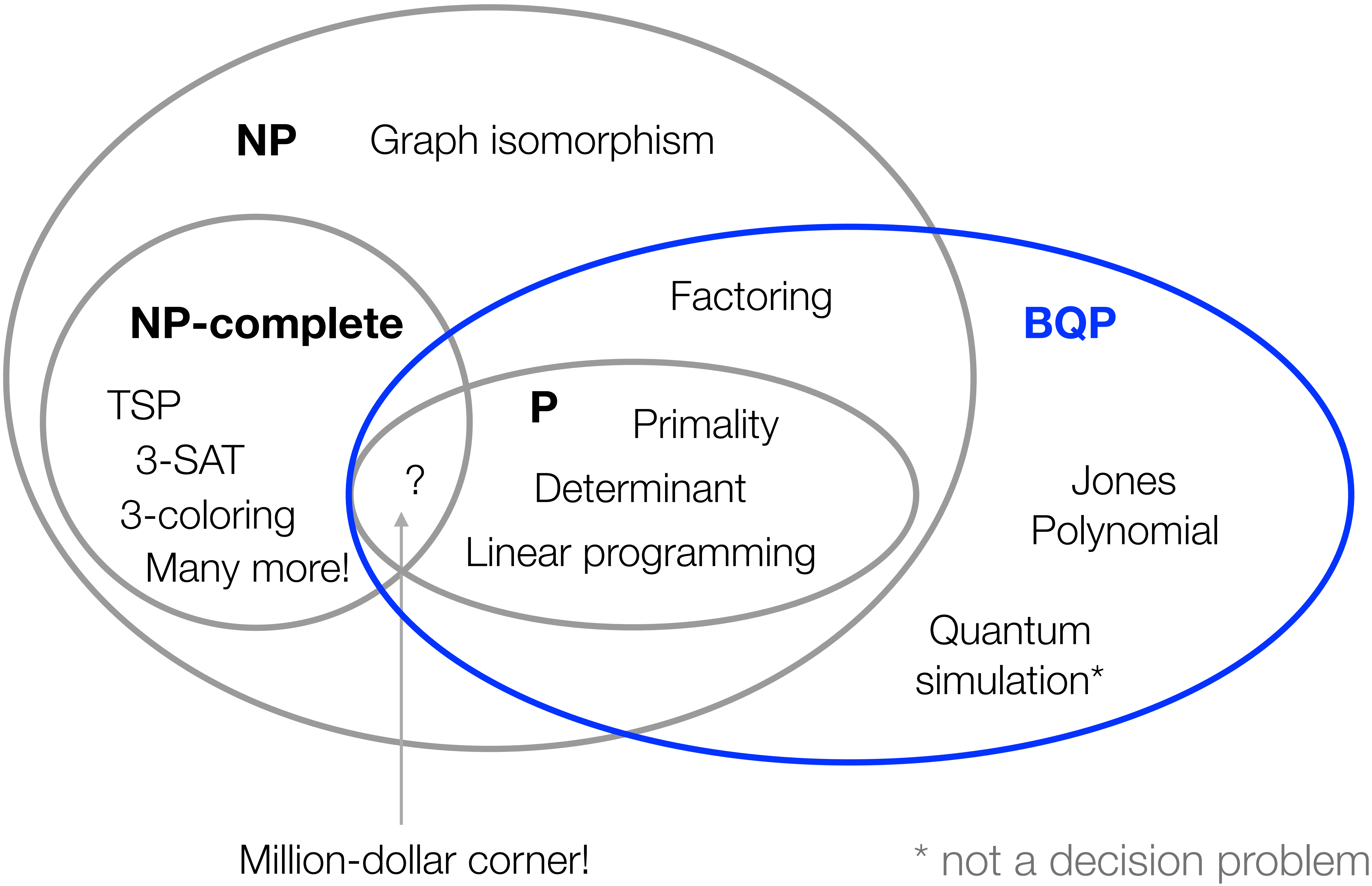
- In a YES instance the output qubit is 1 with probability > 2/3;
- In a NO instance, the output qubit is 0 with probability > 2/3;

* Randomness is built in!

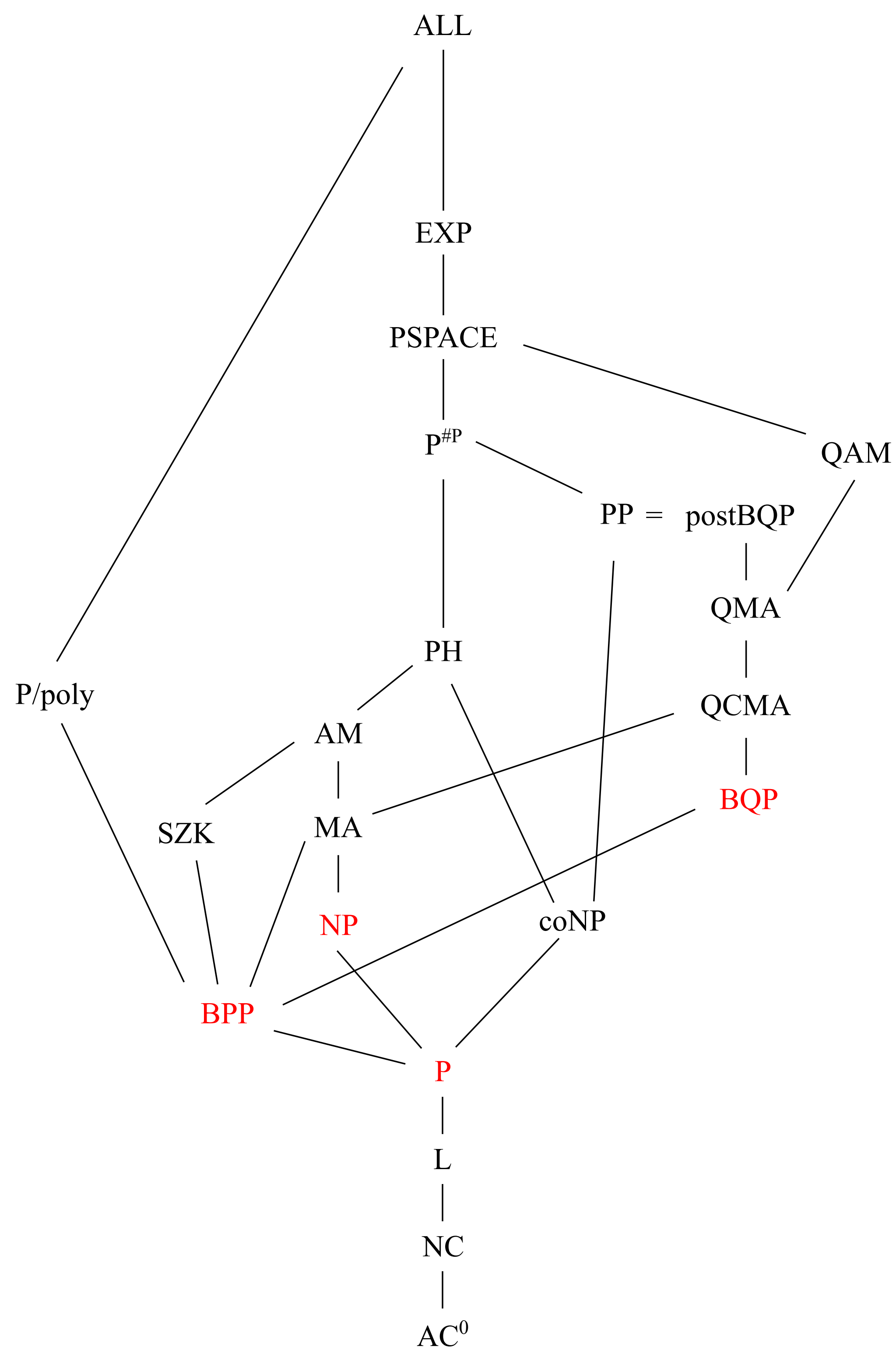
Complexity classes: **BQP**

- Factoring (Shor - 1994)
- Discrete Log (Shor - 1994)
- Quantum simulations (Feynman, Lloyd and others)
- Unstructured search (Grover - 1996)
- Element distinctness (Shi - 2002, Ambainis - 2007)
- Jones polynomials (Aharonov *et al* - 2006)
- And many others to come!

Complexity classes



The Complexity (Petting) Zoo



Scott Aaronson (Complexity Zoo)

The Complexity Zoo (includes Lions)

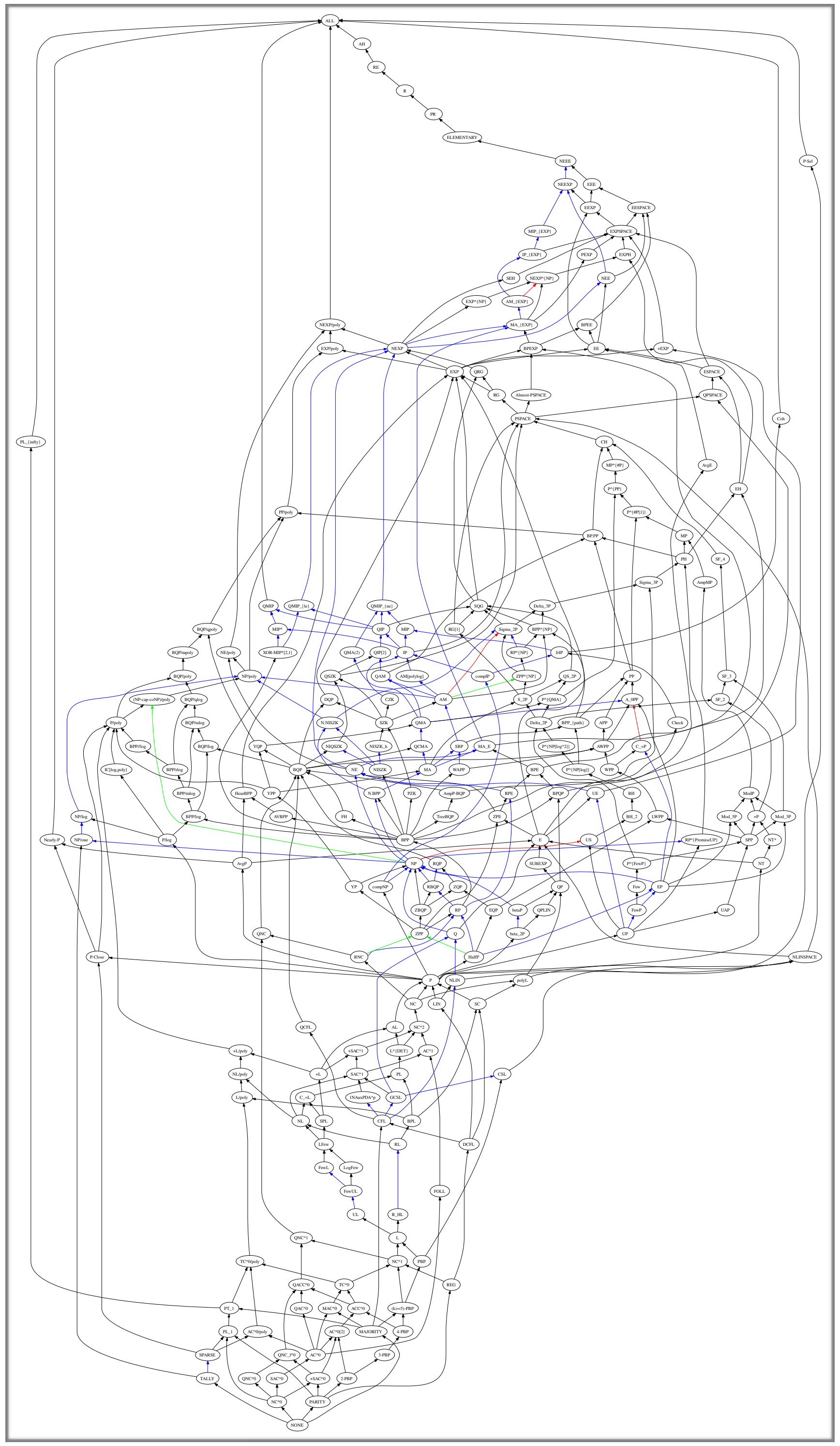
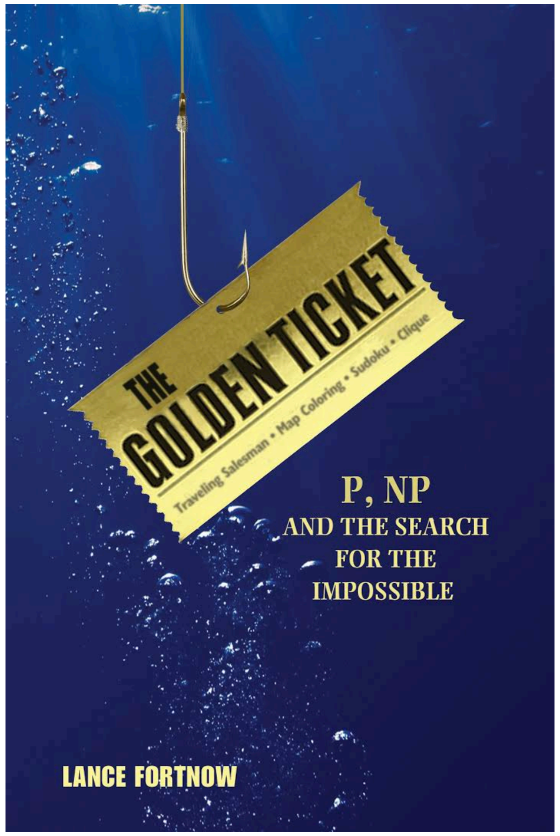


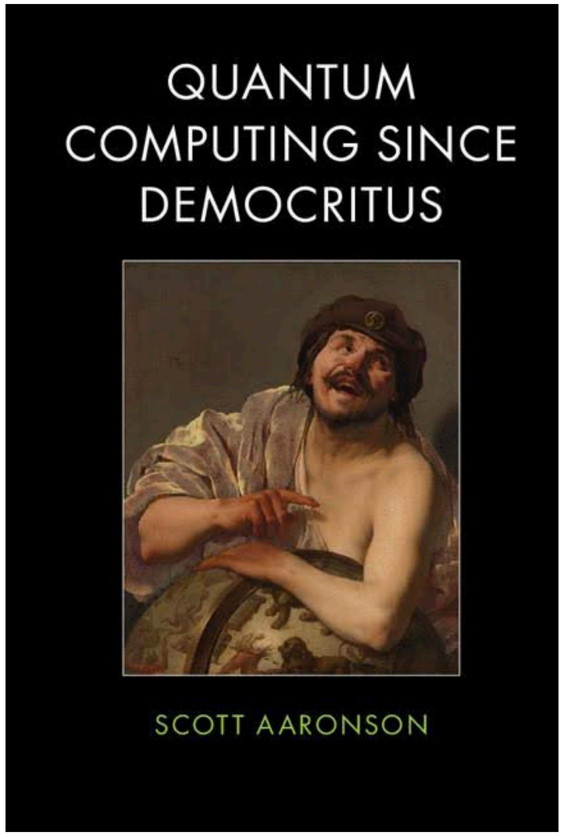
Imagem: Greg Kuperberg (Complexity Zoo)

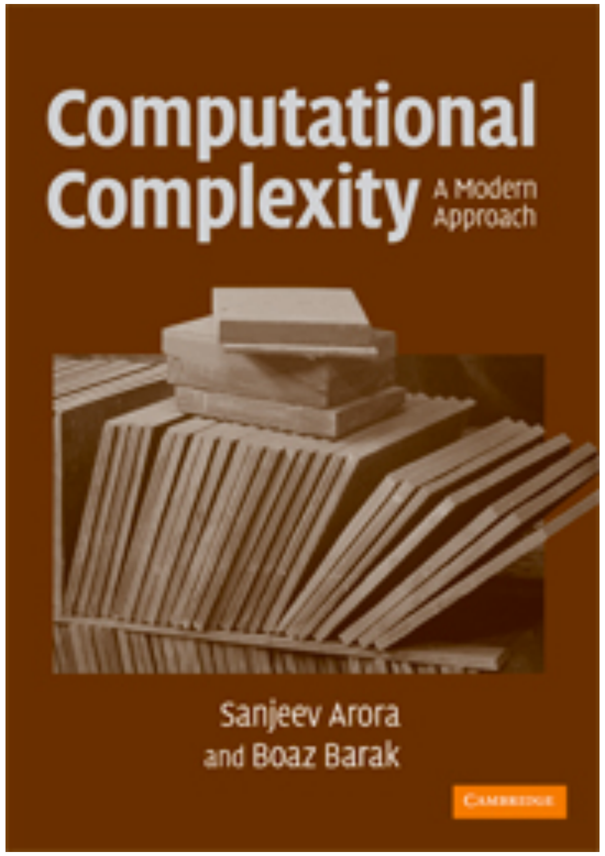
I want to know more!



Lance Fortnow,
"The Golden Ticket: **P**, **NP** and the search for the impossible"

Scott Aaronson,
"Quantum computing since Democritus"





S. Arora and B. Barak
"Computational complexity: a modern approach"