# P versus NP
## A million-dollar question (and more)

Clément Dallard

---

Izleti v matematično vesolje

Univerza na Primorskem

November 24, 2021




# AHHH ... SWEET 60S ...


Source: `computerhistory.org`

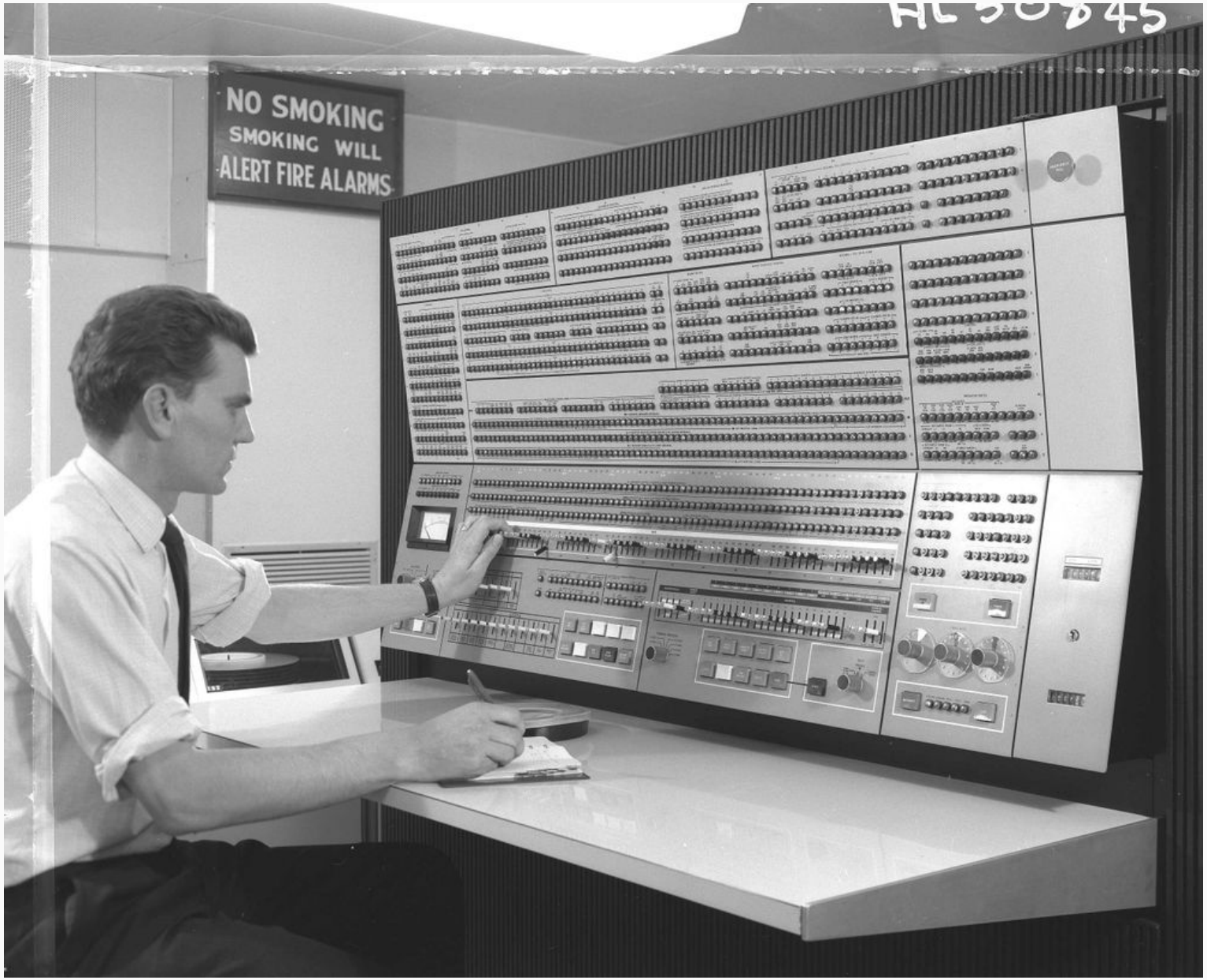
# AHHH ... SWEET 60S ...


Source: `flickr.com`, ArnoldReinhold

# AHHH ... SWEET 60S ...

Source: `computerhistory.org`

# AHHH … SWEET 60S …

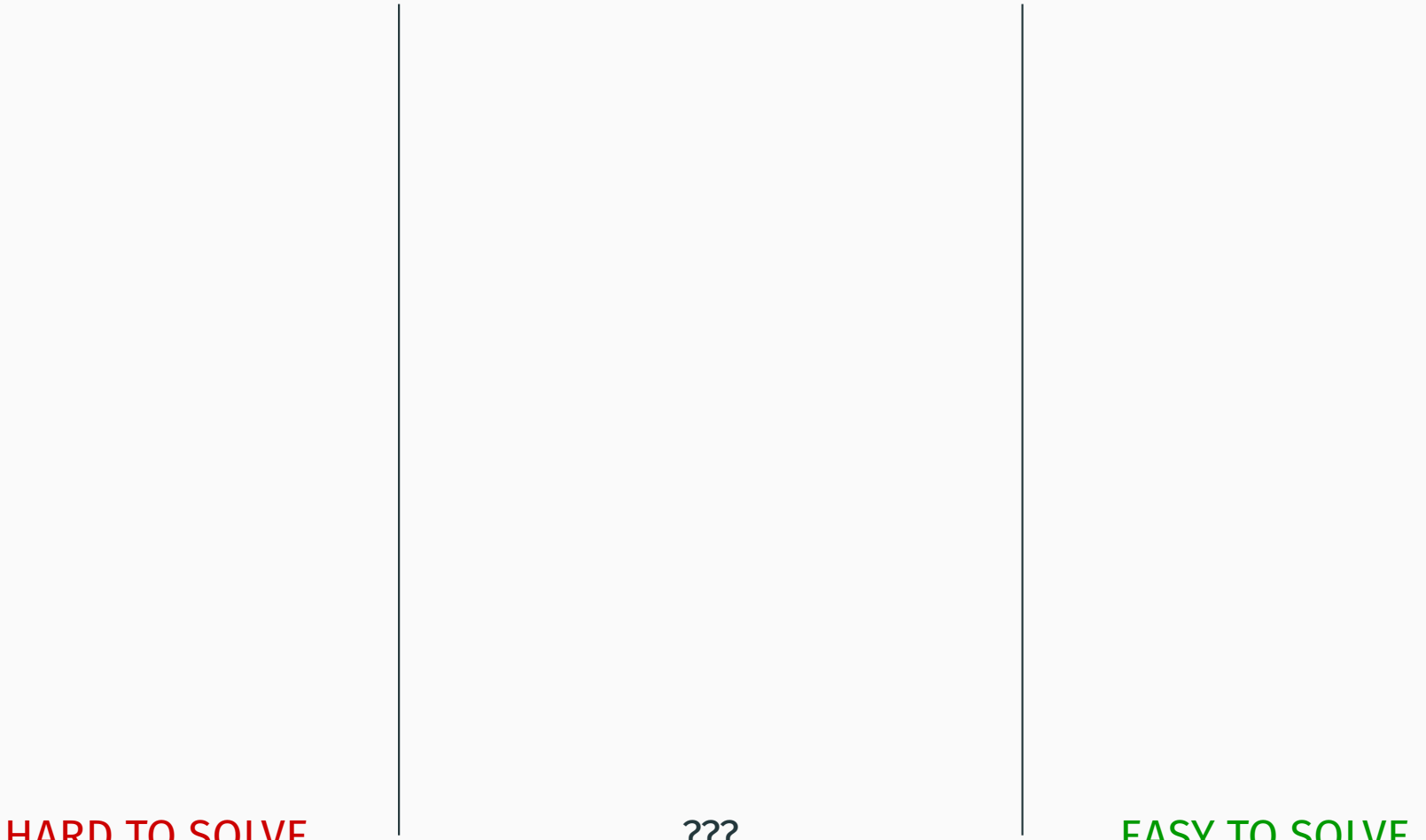



Source: `chilton-computing.org.uk`

# AHHH ... SWEET 60S ...

Source: `nersc.gov`

# HARD OR EASY, THAT IS THE QUESTION



# Hard or easy, that is the question

Multiplication

Shortest Path

Primality Testing

Sorting

HARD TO SOLVE

???

EASY TO SOLVE

# Hard or easy, that is the question

Perfect Strategy Chess

Multiplication

Shortest Path

Primality Testing

Sorting

HARD TO SOLVE

???

EASY TO SOLVE

# Hard or easy, that is the question

| HARD TO SOLVE | ??? | EASY TO SOLVE |
| --- | --- | --- |
| Perfect Strategy Chess | Traveling Salesman | Multiplication |
| | Satisfiability | Shortest Path |
| | Prime Factorization | Primality Testing |
| | Sudoku | Sorting |
| | Discrete Fourier Transform | |

# Hard or easy, that is the question

| HARD TO SOLVE | ??? | EASY TO SOLVE |
| --- | --- | --- |
| Perfect Strategy Chess | Traveling Salesman | Multiplication |
| | Satisfiability | Shortest Path |
| | Prime Factorization | Primality Testing |
| | Sudoku | Sorting |
| | | Discrete Fourier Transform |

# Hard or easy, that is the question

| | | |
|---|---|---|
| | Traveling Salesman | Multiplication |
| | Satisfiability | Shortest Path |
| Perfect Strategy Chess | Prime Factorization | Primality Testing |
| | Sudoku | Sorting |
| | | Discrete Fourier Transform |
| HARD TO SOLVE | ??? | EASY TO SOLVE |
| hard to check a solution | easy to check a solution | easy to check a solution |

P is the class of problems that can be solved efficiently.

(P stands for “deterministic Polynomial time.”)

# P AND NP

**P** is the class of problems that **can be solved efficiently**.
(**P** stands for "deterministic **P**olynomial time.")

**NP** is the class of problems whose **solutions can be checked efficiently**.
(**NP** stands for "**N**ondeterministic **P**olynomial time.")

# P AND NP

**P** is the class of problems that **can be solved efficiently**.
(**P** stands for "deterministic **P**olynomial time.")

**NP** is the class of problems whose **solutions can be checked efficiently**.
(**NP** stands for "**N**ondeterministic **P**olynomial time.")

**Efficiently** means in polynomial-time in the size of the input.

# P AND NP

**P** is the class of problems that **can be solved efficiently**.
(**P** stands for "deterministic **P**olynomial time.")

**NP** is the class of problems whose **solutions can be checked efficiently**.
(**NP** stands for "**N**ondeterministic **P**olynomial time.")

**Efficiently** means in polynomial-time in the size of the input.

Every problem in **P** is also in **NP**!

**P** is the class of problems that **can be solved efficiently**.
(**P** stands for "deterministic **P**olynomial time.")

**NP** is the class of problems whose **solutions can be checked efficiently**.
(**NP** stands for "**N**ondeterministic **P**olynomial time.")

**Efficiently** means in polynomial-time in the size of the input.

Every problem in **P** is also in **NP**!

The **P** versus **NP** question asks whether **finding** a solution is as easy as **checking** a solution.

A problem is **NP-hard** if it is **as hard as any other problem in NP**, in the sense that every problem in **NP** can be “efficiently translated into” an **NP**-hard problem.

A problem is **NP-hard** if it is **as hard as any other problem in NP**, in the sense that every problem in **NP** can be “efficiently translated into” an **NP**-hard problem.

A problem is **NP-complete** if it is **NP-hard** and it **belongs to NP**.

A problem is **NP-hard** if it is **as hard as any other problem in NP**, in the sense that every problem in **NP** can be "efficiently translated into" an **NP**-hard problem.

A problem is **NP-complete** if it is **NP-hard** and it **belongs to NP**.

An efficient algorithm for an **NP**-complete problem would provide an efficient algorithm for **every** problem in **NP**, therefore showing that $\mathbf{P} = \mathbf{NP}$.

# SATISFIABILITY

Stephen Cook (1968)

Leonid Levin (1980s)

# SATISFIABILITY

Stephen Cook (1968)

Leonid Levin (1980s)

The **Cook-Levin theorem** (1971 / 1973) states that the SATISFIABLITY problem is **NP**-complete.

# SATISFIABILITY

Stephen Cook (1968)

Leonid Levin (1980s)

The **Cook-Levin theorem** (1971 / 1973) states that the SATISFIABLITY problem is **NP**-complete.

| SATISFIABILITY | |
|---|---|
| *Input:* | A boolean expression $\phi$. |
| *Question:* | Is $\phi$ satisfiable? |

Stephen Cook (1968)

Leonid Levin (1980s)

The **Cook-Levin theorem** (1971 / 1973) states that the SATISFIABLITY problem is **NP**-complete.

| SATISFIABILITY | |
|---|---|
| *Input:* | A boolean expression $\phi$. |
| *Question:* | Is $\phi$ satisfiable? |

Let $\phi = (x_1 \text{ OR } x_2) \text{ AND } (\bar{x_1} \text{ OR } x_2 \text{ OR } x_3) \text{ AND } (x_1 \text{ OR } \bar{x_3}) \text{ AND } (\bar{x_2} \text{ OR } \bar{x_3})$.
Then $\phi$ can be satisfied by setting $x_1 = \text{true}$, $x_2 = \text{true}$, and $x_3 = \text{false}$.

# SATISFIABILITY IS NOT ALONE

Richard Karp (1980)

In 1972, Karp showed that 21 other problems in **NP** are **NP**-complete by providing a polynomial-time reduction from SATISFIABILITY to each of these 21 problems.

# SATISFIABILITY IS NOT ALONE

Richard Karp (1980)

In 1972, Karp showed that 21 other problems in **NP** are **NP**-complete by providing a polynomial-time reduction from SATISFIABILITY to each of these 21 problems.

Nowadays, thousands of problems are known to be **NP**-complete.

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that $\mathbf{P} = \mathbf{NP}$.

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that $\mathbf{P} = \mathbf{NP}$.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that $\mathbf{P} = \mathbf{NP}$.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.
Then you have $2^n$ possible combinations to check (in the worst case).

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that **P** = **NP**.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.
Then you have $2^n$ possible combinations to check (in the worst case).

| Input size $n$ | Number of steps | Time to solve |
|---|---|---|

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that **P** = **NP**.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.
Then you have $2^n$ possible combinations to check (in the worst case).

| **Input size** $n$ | **Number of steps** | **Time to solve** |
|---|---|---|
| $n = 10$ | $2^{10} = 1024$ | $0,01$ s |

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that **P** = **NP**.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.
Then you have $2^n$ possible combinations to check (in the worst case).

| **Input size** $n$ | **Number of steps** | **Time to solve** |
|---|---|---|
| $n = 10$ | $2^{10} = 1024$ | $0,01$ s |
| $n = 20$ | $2^{20} > 10^6$ | $10,24$ s |

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that **P** = **NP**.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.
Then you have $2^n$ possible combinations to check (in the worst case).

| Input size $n$ | Number of steps | Time to solve |
|---|---|---|
| $n = 10$ | $2^{10} = 1024$ | $0,01$ s |
| $n = 20$ | $2^{20} > 10^6$ | $10,24$ s |
| $n = 30$ | $2^{30} > 10^9$ | $> 2.9$ h |

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that $\mathbf{P} = \mathbf{NP}$.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.
Then you have $2^n$ possible combinations to check (in the worst case).

| Input size $n$ | Number of steps | Time to solve |
|---|---|---|
| $n = 10$ | $2^{10} = 1024$ | $0,01$ s |
| $n = 20$ | $2^{20} > 10^6$ | $10,24$ s |
| $n = 30$ | $2^{30} > 10^9$ | $> 2.9$ h |
| $n = 40$ | $2^{40} > 10^{12}$ | $>$ 4 months |

We only know **exponential** algorithms for **NP**-complete problems: if we knew a **polynomial** (that is, **efficient**) algorithm, then we would know that **P** = **NP**.

Suppose that you want to solve SATISFIABILITY when given a boolean expression with $n$ variables.
Then you have $2^n$ possible combinations to check (in the worst case).

| Input size $n$ | Number of steps | Time to solve |
|---|---|---|
| $n = 10$ | $2^{10} = 1024$ | $0,01$ s |
| $n = 20$ | $2^{20} > 10^6$ | $10,24$ s |
| $n = 30$ | $2^{30} > 10^9$ | $> 2.9$ h |
| $n = 40$ | $2^{40} > 10^{12}$ | $>$ 4 months |
| $n = 50$ | $2^{50} > 10^{15}$ | $>$ 348 years |

- Sudoku: NP-complete (Yato, 2003)

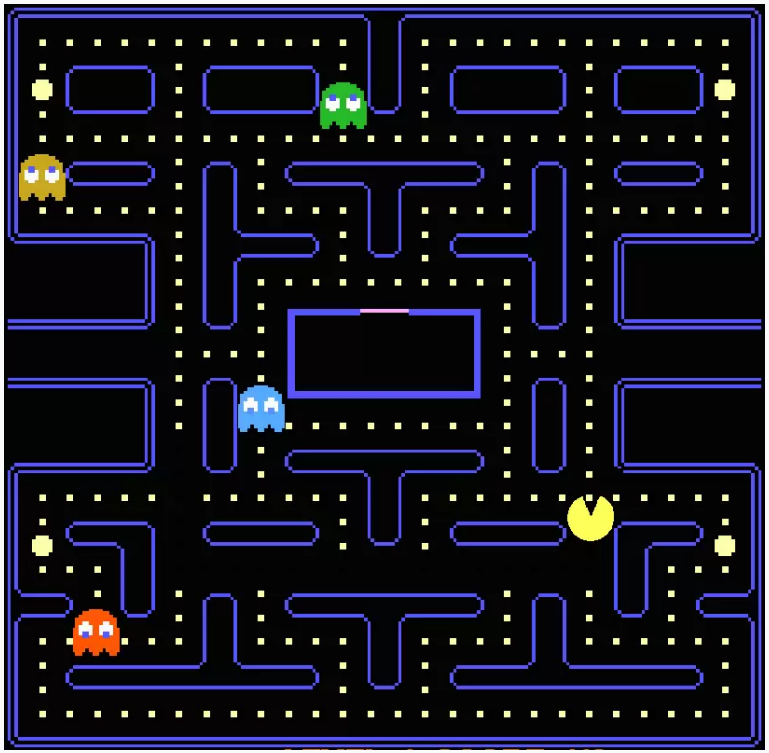
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

# Some (fun) NP-hard and NP-complete problems

- **Sudoku**: **NP**-complete (Yato, 2003)
- **Candy Crush**: **NP**-hard (Gualà, Leucci, and Natale, 2014)

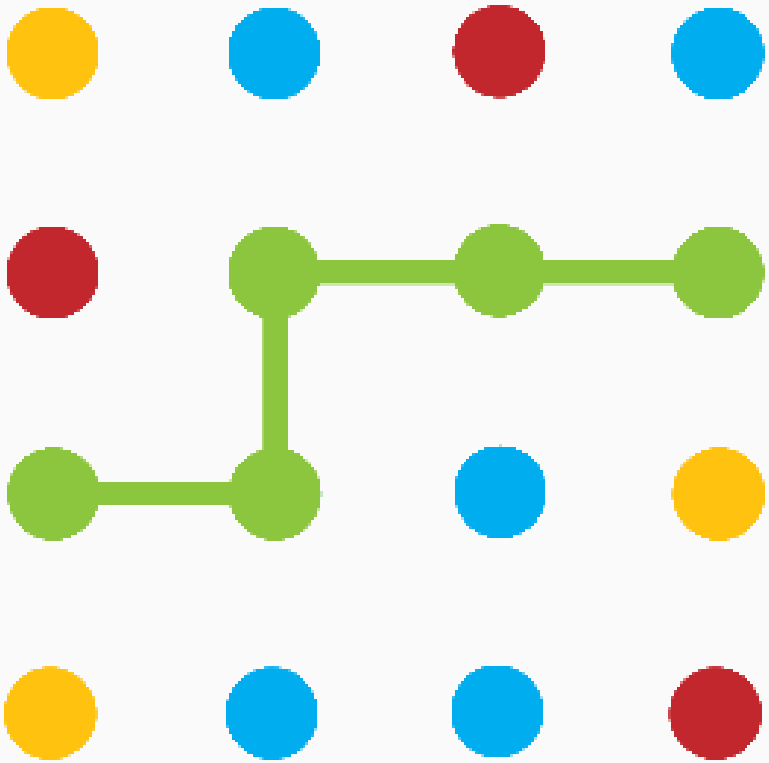
# Some (fun) NP-hard and NP-complete problems

- Sudoku: NP-complete
  (Yato, 2003)
- Candy Crush: NP-hard
  (Gualà, Leucci, and Natale, 2014)
- Pac-Man: NP-hard
  (Viglietta, 2014)



# Some (fun) NP-hard and NP-complete problems

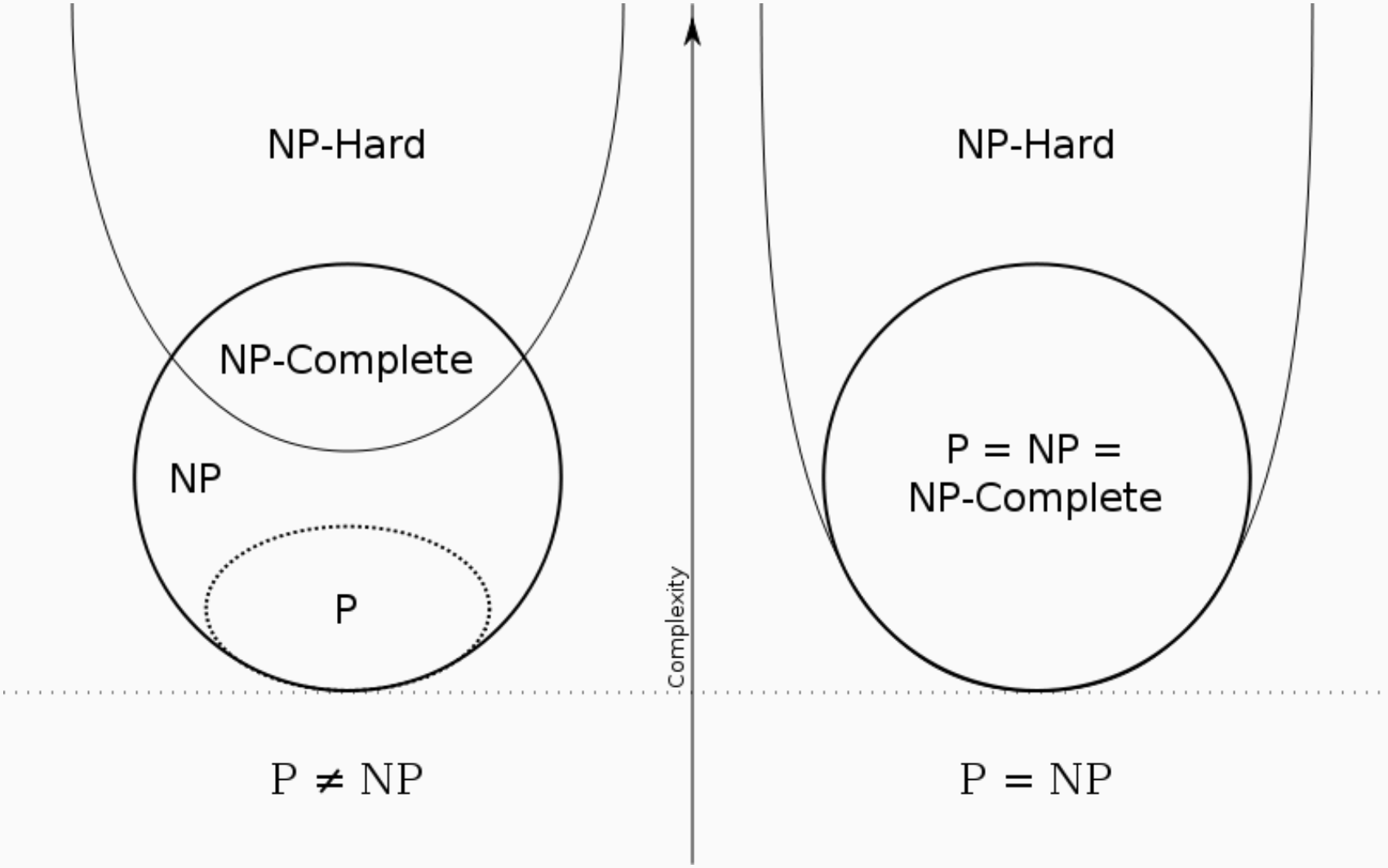
- **Sudoku**: NP-complete
  (Yato, 2003)
- **Candy Crush**: NP-hard
  (Gualà, Leucci, and Natale, 2014)
- **Pac-Man**: NP-hard
  (Viglietta, 2014)
- **(Two) Dots**: NP-complete
  (Misra, 2016)



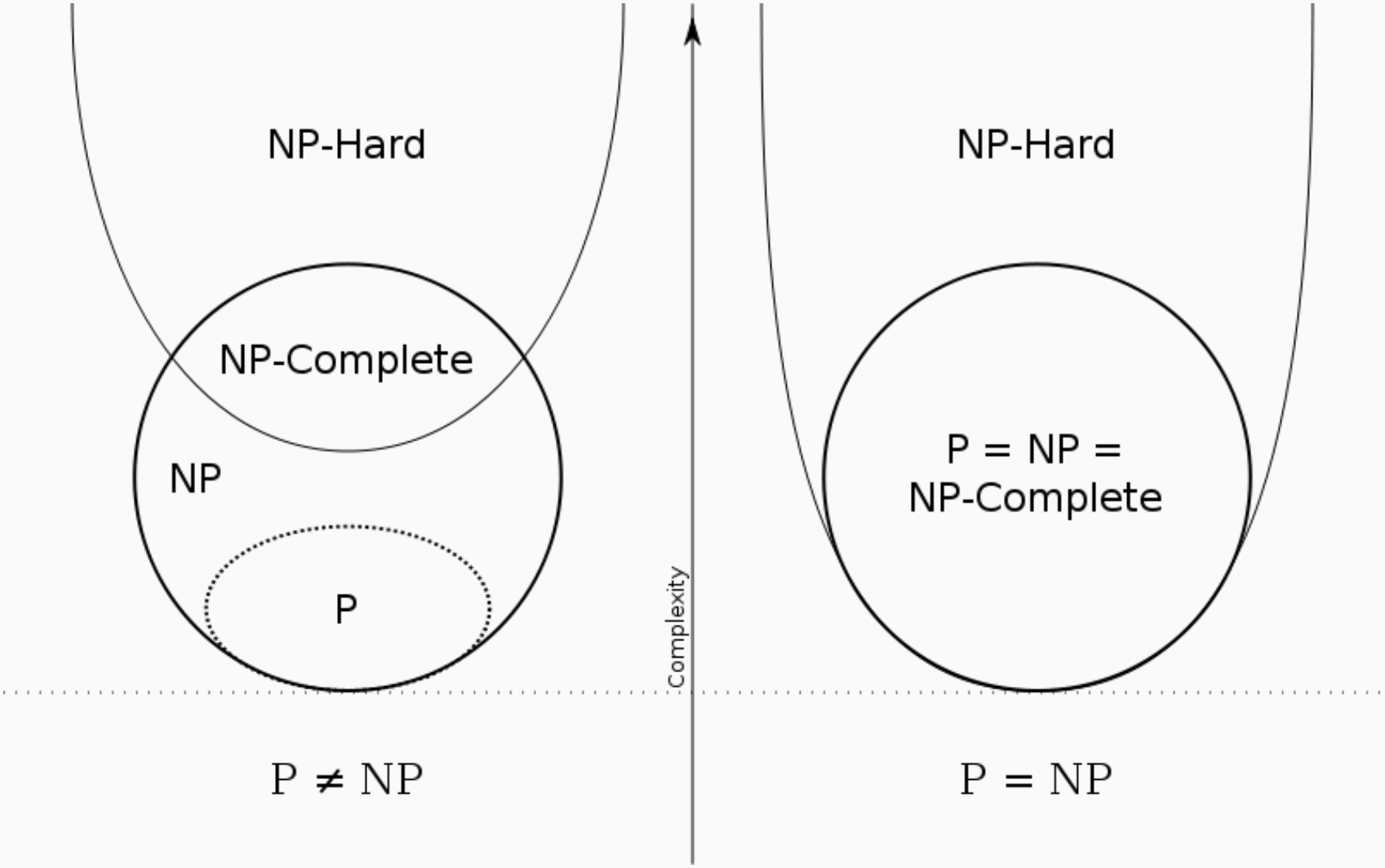
# Some (fun) NP-hard and NP-complete problems

- Sudoku: NP-complete
  (Yato, 2003)
- Candy Crush: NP-hard
  (Gualà, Leucci, and Natale, 2014)
- Pac-Man: NP-hard
  (Viglietta, 2014)
- (Two) Dots: NP-complete
  (Misra, 2016)
- Angry Birds: NP-complete
  (Stephenson, Renz, and Ge, 2017)

# A GLIMPSE AT THE COMPLEXITY ZOO



# A GLIMPSE AT THE COMPLEXITY ZOO




If **P** $\neq$ **NP**, then we know that there are problems in **NP** that are neither **NP**-complete nor in **P** (Ladner, 1975).

# You said one million dollars?

**Millenium prize problems** (Clay institute):

- Yang–Mills and Mass Gap
- Riemann Hypothesis
- P vs NP Problem
- Navier–Stokes Equation
- Hodge Conjecture
- Poincaré Conjecture
- Birch and Swinnerton-Dyer Conjecture

# You said one million dollars?

**Millenium prize problems** (Clay institute):

- Yang–Mills and Mass Gap
- Riemann Hypothesis
- P vs NP Problem
- Navier–Stokes Equation
- Hodge Conjecture
- Poincaré Conjecture (solved by Perelman, 2003)
- Birch and Swinnerton-Dyer Conjecture

# How to get a million dollars?

To show that $\mathsf{P} = \mathsf{NP}$:

- provide a proof (possibly non-constructive)
  or
- find an **efficient** algorithm for an **NP**-complete problem.

# How to get a million dollars?

To show that $\mathsf{P} = \mathsf{NP}$:

- provide a proof (possibly non-constructive)
  or
- find an efficient algorithm for an **NP**-complete problem.

There are at least 62 papers claiming that $\mathsf{P} = \mathsf{NP}$.

# How to get a million dollars?

To show that $\mathsf{P} = \mathsf{NP}$:

- provide a proof (possibly non-constructive)
  or
- find an **efficient** algorithm for an $\mathsf{NP}$-complete problem.

There are at least 62 papers claiming that $\mathsf{P} = \mathsf{NP}$.

To show that $\mathsf{P} \neq \mathsf{NP}$: no way around it, you'll need an actual proof.

# How to get a million dollars?

To show that $\mathsf{P} = \mathsf{NP}$:

- provide a proof (possibly non-constructive)
  or
- find an efficient algorithm for an NP-complete problem.

There are at least 62 papers claiming that $\mathsf{P} = \mathsf{NP}$.

To show that $\mathsf{P} \neq \mathsf{NP}$: no way around it, you'll need an actual proof.

There are at least 50 papers claiming that $\mathsf{P} \neq \mathsf{NP}$.

# How to get a million dollars?

To show that $\mathsf{P} = \mathsf{NP}$:

- provide a proof (possibly non-constructive)
  or
- find an efficient algorithm for an **NP**-complete problem.

There are at least 62 papers claiming that $\mathsf{P} = \mathsf{NP}$.

To show that $\mathsf{P} \neq \mathsf{NP}$: no way around it, you'll need an actual proof.

There are at least 50 papers claiming that $\mathsf{P} \neq \mathsf{NP}$.

If you can answer the **P** versus **NP** problem, don't hesitate to contact me.

# How to get a million dollars?

To show that $\mathsf{P} = \mathsf{NP}$:

- provide a proof (possibly non-constructive)
  or
- find an efficient algorithm for an NP-complete problem.

There are at least 62 papers claiming that $\mathsf{P} = \mathsf{NP}$.

To show that $\mathsf{P} \neq \mathsf{NP}$: no way around it, you'll need an actual proof.

There are at least 50 papers claiming that $\mathsf{P} \neq \mathsf{NP}$.

If you can answer the P versus NP problem, don't hesitate to contact me.

If you prove that $\mathsf{P} = \mathsf{NP}$ by providing an efficient algorithm for an NP-complete problem, don't say anything to anyone (except me).

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

- most of our current encryption algorithms should be changed;

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

- most of our current encryption algorithms should be changed;
- problems that are easy to check would be just as easy to solve;

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

- most of our current encryption algorithms should be changed;
- problems that are easy to check would be just as easy to solve;
- a great scientific and technologic leap forward for humanity.

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

- most of our current encryption algorithms should be changed;
- problems that are easy to check would be just as easy to solve;
- a great scientific and technologic leap forward for humanity.

Maybe **P** = **NP** but the time complexity to solve **NP**-complete problems, although polynomial, is **HUMONGOUS**.

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

- most of our current encryption algorithms should be changed;
- problems that are easy to check would be just as easy to solve;
- a great scientific and technologic leap forward for humanity.

Maybe **P** = **NP** but the time complexity to solve **NP**-complete problems, although polynomial, is **HUMONGOUS**.

If we only have a **non-contructive proof** or a "**not efficient enough**" algorithm:

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

- most of our current encryption algorithms should be changed;
- problems that are easy to check would be just as easy to solve;
- a great scientific and technologic leap forward for humanity.

Maybe **P** = **NP** but the time complexity to solve **NP**-complete problems, although polynomial, is **HUMONGOUS**.

If we only have a **non-contructive proof** or a "**not efficient enough**" algorithm:

- a scientific accomplishment, although somewhat frustrating;

# IF WE KNOW THAT P = NP

If we have a **really efficient** algorithm for solving **NP**-complete problems:

- most of our current encryption algorithms should be changed;
- problems that are easy to check would be just as easy to solve;
- a great scientific and technologic leap forward for humanity.

Maybe **P** = **NP** but the time complexity to solve **NP**-complete problems, although polynomial, is **HUMONGOUS**.

If we only have a **non-contructive proof** or a "**not efficient enough**" algorithm:

- a scientific accomplishment, although somewhat frustrating;
- nothing else really changes, but we should be ready to update our encryption algorithms!

# IF WE KNOW THAT P $\neq$ NP

Well, that's (almost) life as we know it today ...

# IF WE KNOW THAT P $\neq$ NP

Well, that's (almost) life as we know it today ...

Still, that would be a great scientific accomplishment.

# IF WE KNOW THAT P $\neq$ NP

Well, that's (almost) life as we know it today ...

Still, that would be a great scientific accomplishment.

We would feel better when we don't finish a Sudoku grid!

The NUMBER SCRABBLE game:

- two players take turn to select numbers from 1 to 9 without repeating any numbers already selected;

1 2 3 4 5 6 7 8 9

Player 1:

Player 2:

The NUMBER SCRABBLE game:

- two players take turn to select numbers from 1 to 9 without repeating any numbers already selected;
- the first player with a sum of exactly 15 using any three of their number selections wins the game.

1 2 3 4 5 6 7 8 9

Player 1:

Player 2:

The NUMBER SCRABBLE game:

- two players take turn to select numbers from 1 to 9 without repeating any numbers already selected;
- the first player with a sum of exactly 15 using any three of their number selections wins the game.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Player 1: | | | 3 | | | | | | |
| Player 2: | | | | | | | | | |

The NUMBER SCRABBLE game:

- two players take turn to select numbers from 1 to 9 without repeating any numbers already selected;
- the first player with a sum of exactly 15 using any three of their number selections wins the game.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Player 1: | | | 3 | | | | | | |
| Player 2: | | | | | | 6 | | | |

The NUMBER SCRABBLE game:

- two players take turn to select numbers from 1 to 9 without repeating any numbers already selected;
- the first player with a sum of exactly 15 using any three of their number selections wins the game.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Player 1: | | | 3 | | 5 | | | | |
| Player 2: | | | | | | 6 | | | |

The NUMBER SCRABBLE game:

- two players take turn to select numbers from 1 to 9 without repeating any numbers already selected;
- the first player with a sum of exactly 15 using any three of their number selections wins the game.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Player 1: | | | 3 | | 5 | | | | |
| Player 2: | | | | | | 6 | 7 | | |

The NUMBER SCRABBLE game:

- two players take turn to select numbers from 1 to 9 without repeating any numbers already selected;
- the first player with a sum of exactly 15 using any three of their number selections wins the game.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Player 1: | | | 3 | | 5 | | | | |
| Player 2: | | | | | | 6 | 7 | | |

If displayed as a magic square:

| | | |
|---|---|---|
| 2 | 7 | 6 |
| 9 | 5 | 1 |
| 4 | 3 | 8 |

# SUPPORTING P = NP

Maybe we just haven't found **yet** an efficient algorithm to solve **NP**-complete problems!

# SUPPORTING P = NP

Maybe we just haven't found **yet** an efficient algorithm to solve **NP**-complete problems!

Besides, some problems have been proved to belong to **P** ony after several years of research ...

# SUPPORTING P $\neq$ NP

There are some clever people out there but no one could come up with an efficient algorithm for an **NP**-complete problem in more than 50 years.

# SUPPORTING P $\neq$ NP

There are some clever people out there but no one could come up with an efficient algorithm for an **NP**-complete problem in more than 50 years.

> *If **P** = **NP**, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett.*
>
> Scott Aaronson

# SUPPORTING P $\neq$ NP

There are some clever people out there but no one could come up with an efficient algorithm for an **NP**-complete problem in more than 50 years.

> *If $P = NP$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett.*
>
> Scott Aaronson

> *Given how many confusions and weird misinterpretations it's spawned, I, Scott Aaronson, hereby formally disown my statement about how "everyone who could appreciate a symphony would be Mozart" if $P = NP$.*
>
> Scott Aaronson

# Can we even prove it?

Maybe the **P** versus **NP** question simply not provable in any of the current logical theories.

Recall that we only have algorithms that are **exponential** in the size of the input for **NP**-complete problems.

Recall that we only have algorithms that are **exponential** in the size of the input for **NP**-complete problems.

- **Heuristics**: don't compute an optimal solution, just try to not be too bad.

Recall that we only have algorithms that are **exponential** in the size of the input for **NP**-complete problems.

- **Heuristics**: don't compute an optimal solution, just try to not be too bad.
- **Specific inputs**: develop efficient and optimal algorithms for specific kind of inputs.

Recall that we only have algorithms that are **exponential** in the size of the input for **NP**-complete problems.

- **Heuristics**: don't compute an optimal solution, just try to not be too bad.
- **Specific inputs**: develop efficient and optimal algorithms for specific kind of inputs.
- **Improve the base of the exponent**: $1.234^n$ grows way slower than $2^n$.

# QUANTUM COMPUTING TO THE RESCUE?

**BQP** is the class of problems that **can be solved efficiently** with an error probability of at most 1/3 (for all inputs) **on a quantum computer**.

(**BQP** stands for "**B**ounded-error **Q**uantum **P**olynomial time.")

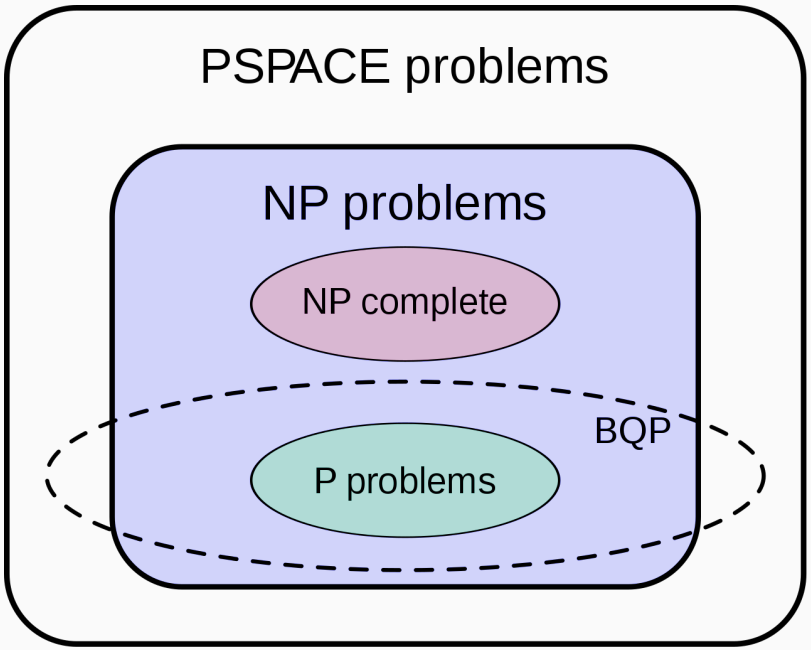
# QUANTUM COMPUTING TO THE RESCUE?

**BQP** is the class of problems that **can be solved efficiently** with an error probability of at most 1/3 (for all inputs) **on a quantum computer**.
(**BQP** stands for "**B**ounded-error **Q**uantum **P**olynomial time.")

- We know that $\mathsf{P} \subseteq \mathsf{BQP}$.

**BQP** is the class of problems that **can be solved efficiently** with an error probability of at most 1/3 (for all inputs) **on a quantum computer**.
(**BQP** stands for "**B**ounded-error **Q**uantum **P**olynomial time.")

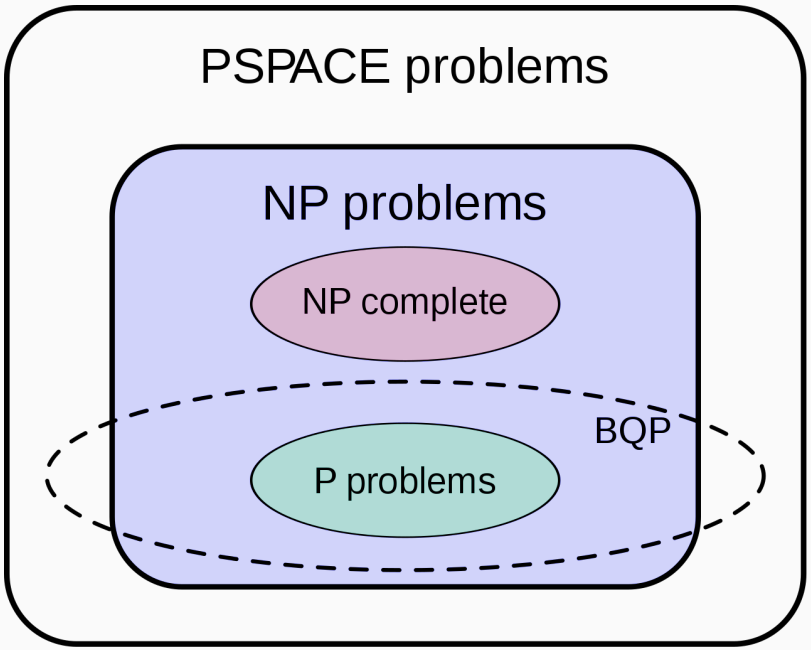
- We know that $\mathbf{P} \subseteq \mathbf{BQP}$.
- It is believed that $\mathbf{NP} \not\subseteq \mathbf{BQP}$.




Suspected relationship of **BQP** with **NP**

**BQP** is the class of problems that **can be solved efficiently** with an error probability of at most 1/3 (for all inputs) **on a quantum computer**.
(**BQP** stands for "**B**ounded-error **Q**uantum **P**olynomial time.")

- We know that $\mathbf{P} \subseteq \mathbf{BQP}$.
- It is believed that $\mathbf{NP} \not\subseteq \mathbf{BQP}$.
- We can solve some problems efficiently on quantum computers, including PRIME FACTORIZATION, but no **NP**-complete problem.




Suspected relationship of **BQP** with **NP**

# Thank you!

Questions?