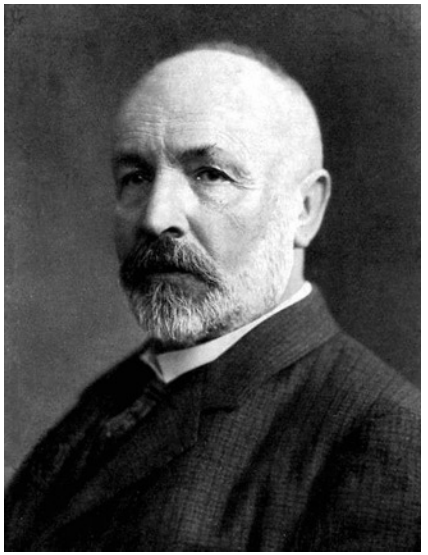
15-112
Fundamentals of Programming

Week 13, Lecture 1:
Uncomputability + P vs NP



April 12, 2016

Motivational Quote of the Day

"Computer Science is no more about computers than astronomy is about telescopes."

~~*Edsger Dijkstra*~~

- Michael Fellows

PART 1:
How computer science was born.

PART 2:
Uncomputable problems

PART 3:
Computational complexity and the “P vs NP” problem

PART 1:
How computer science was born.

What is computer science?

Is it:

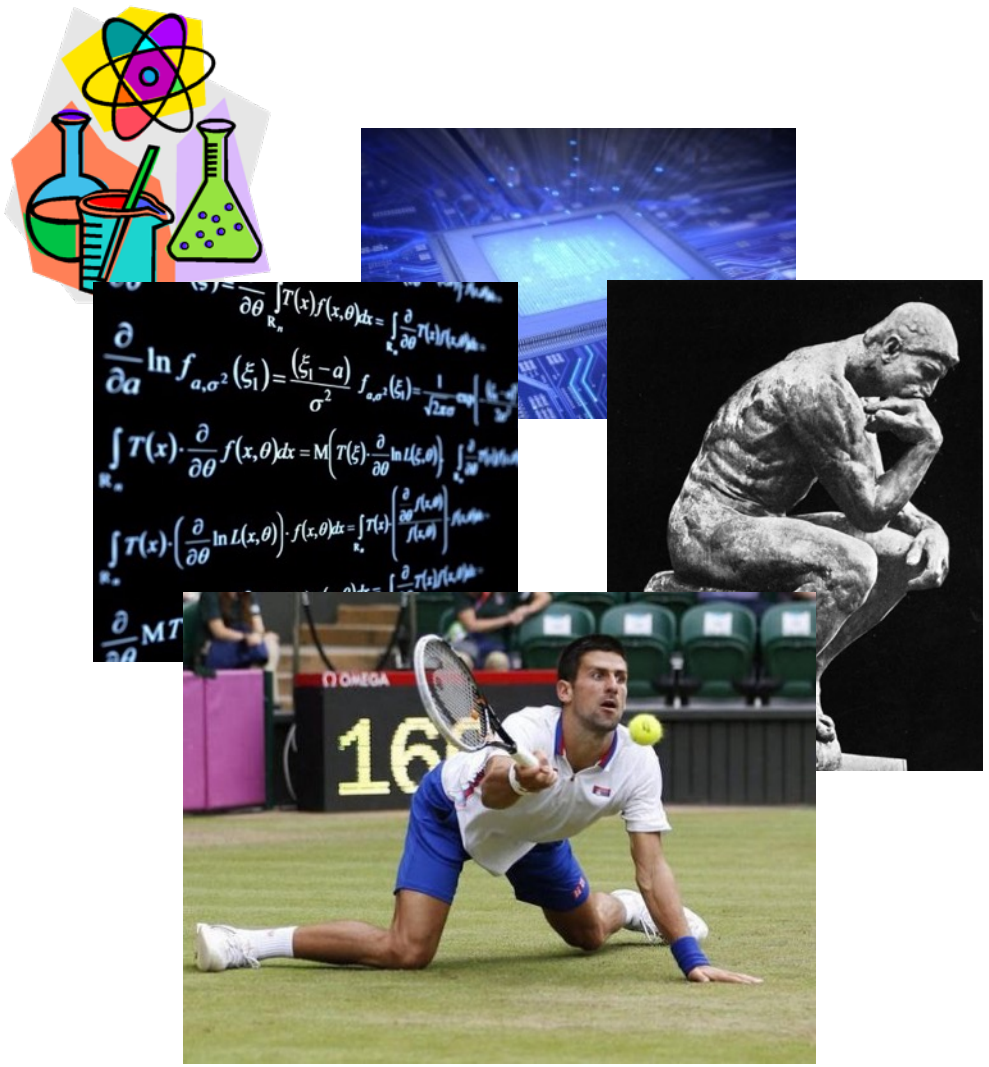
"Writing (Python) programs
that do certain tasks."

What is theoretical computer science?

What is computer science?

Is it branch of:

- science?
- engineering?
- math?
- philosophy?
- sports?



Physics

Theoretical physics

- come up with mathematical models

  Nature's language is mathematics

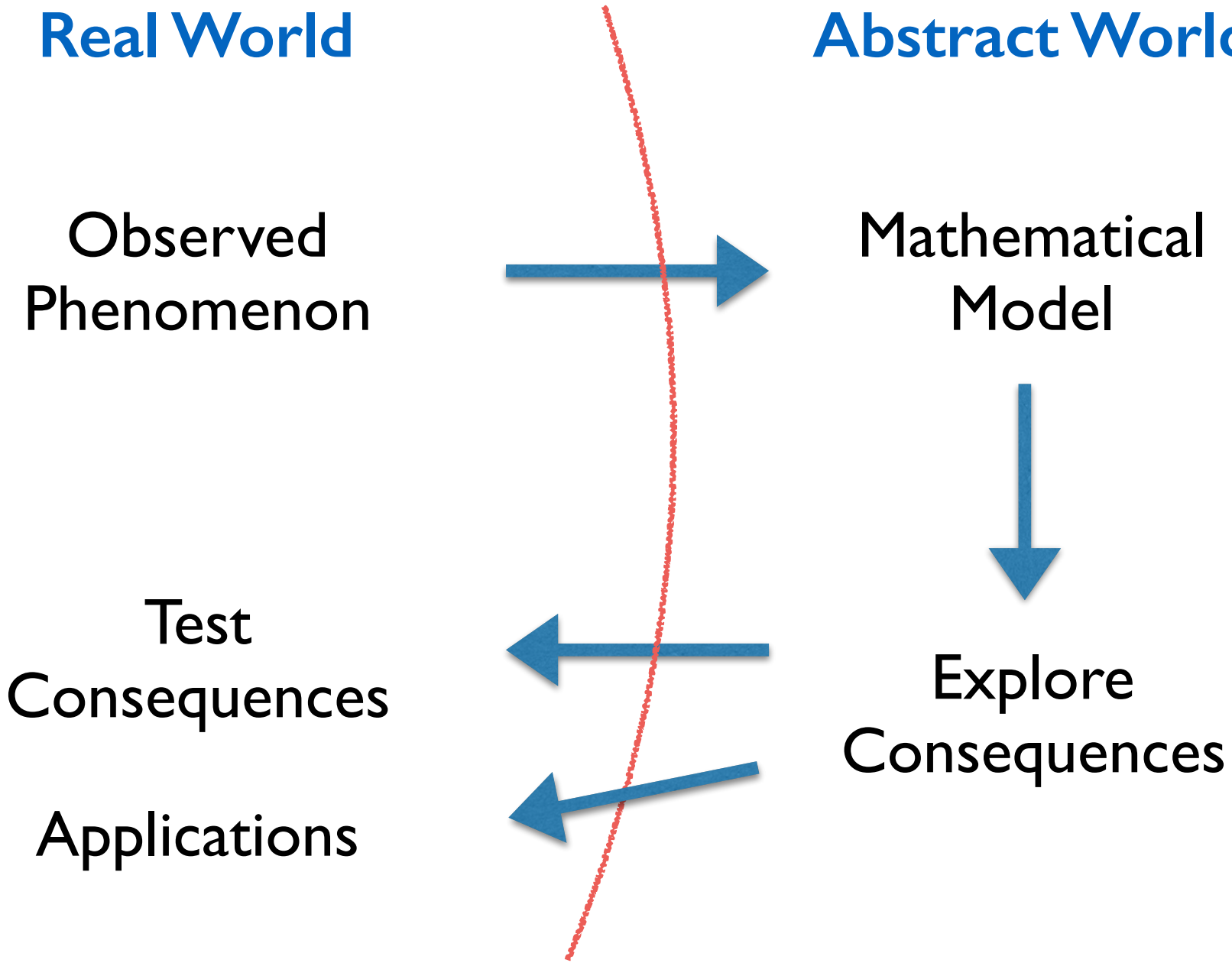
- derive the logical consequences

Experimental physics

- make observations about the universe
- test the model with experiments

Applications/Engineering

The role of theoretical physics



Theoretical Physics

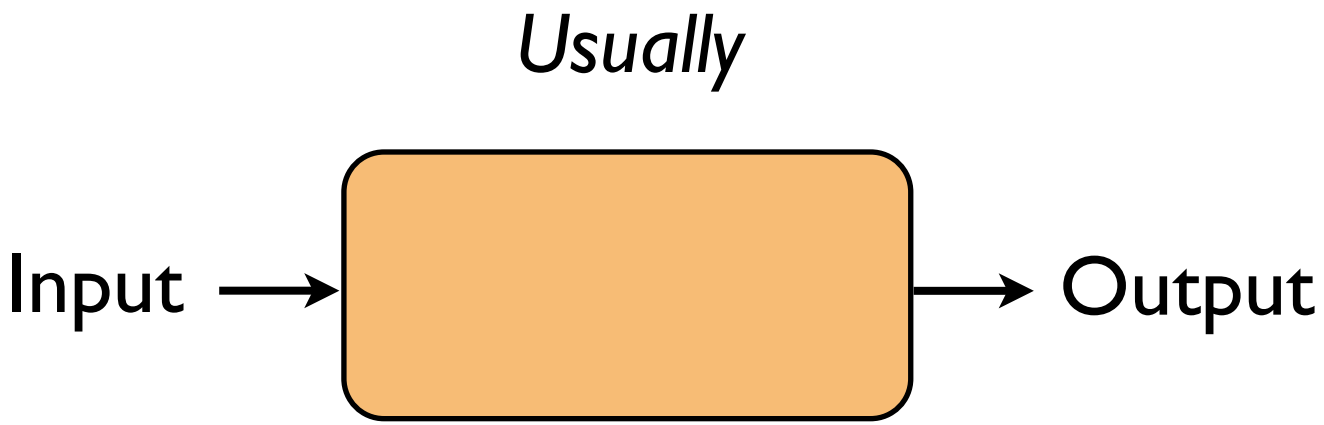
- science?
- engineering?
- math?
- philosophy?
- sports?

Computer Science

The science that studies computation.

Computation: manipulation of information/data.

Algorithm: rigorous description of how the data is manipulated.

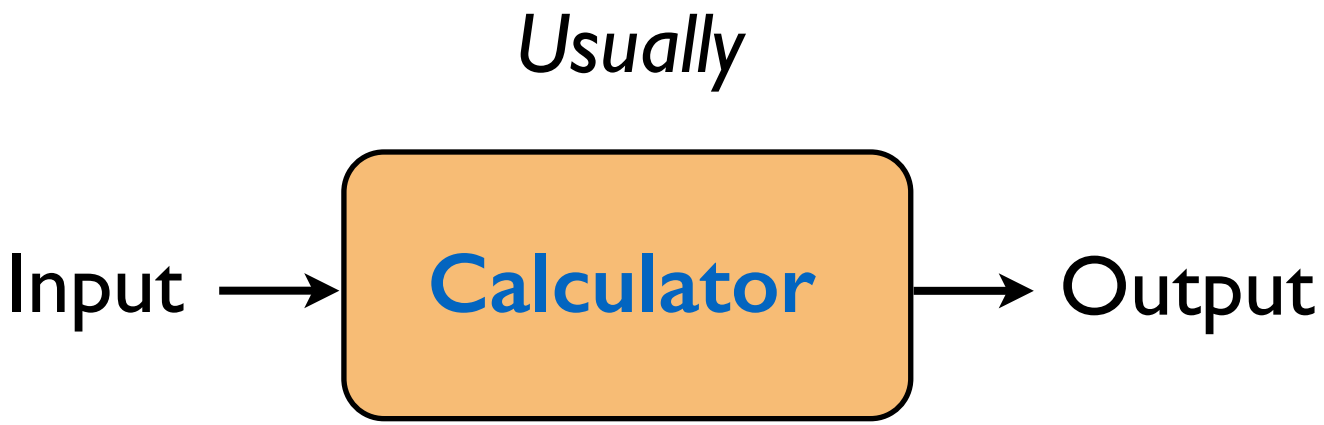


Computer Science

The science that studies computation.

Computation: manipulation of information/data.

Algorithm: rigorous description of how the data is manipulated.

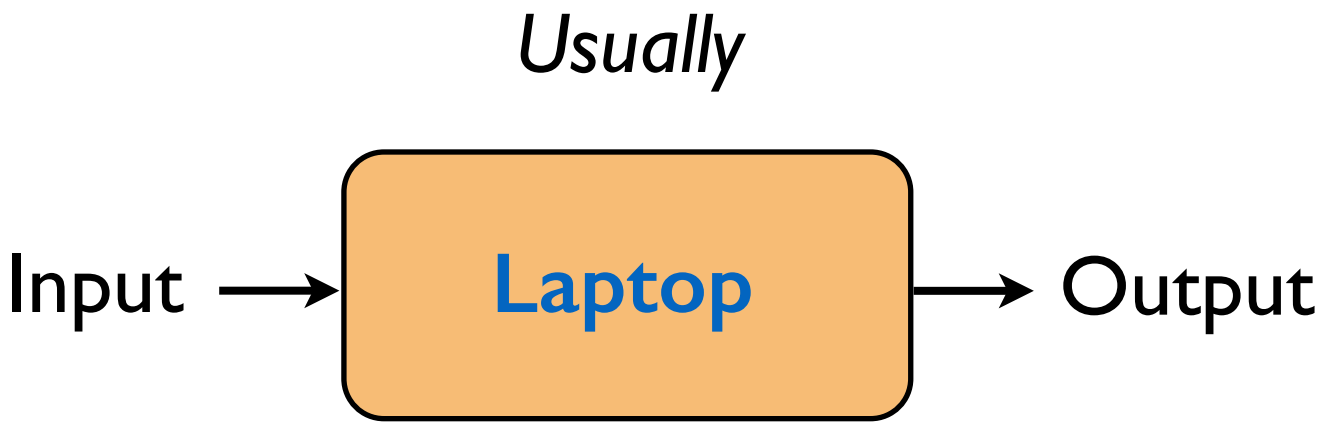


Computer Science

The science that studies computation.

Computation: manipulation of information/data.

Algorithm: rigorous description of how the data is manipulated.

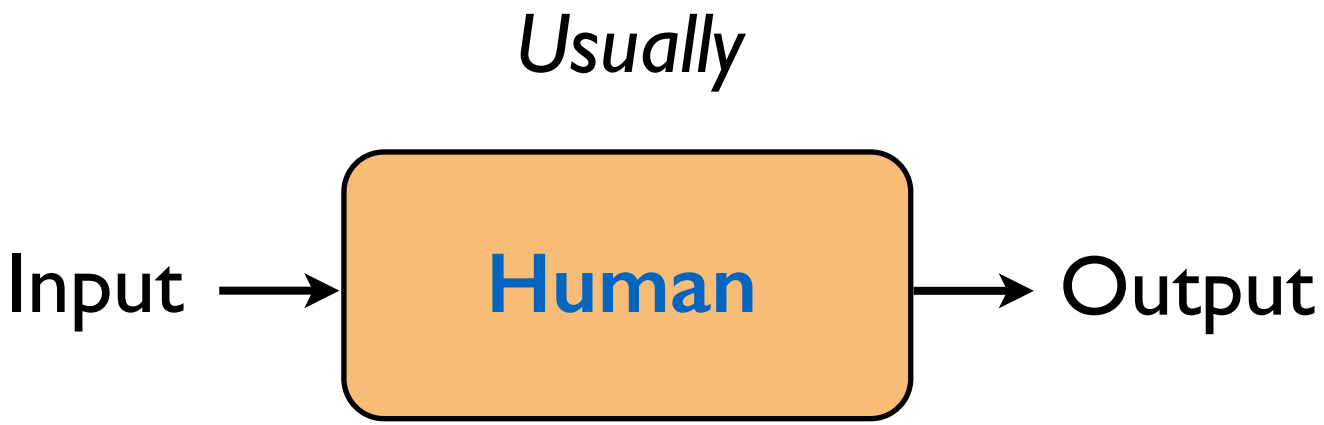


Computer Science

The science that studies computation.

Computation: manipulation of information/data.

Algorithm: rigorous description of how the data is manipulated.



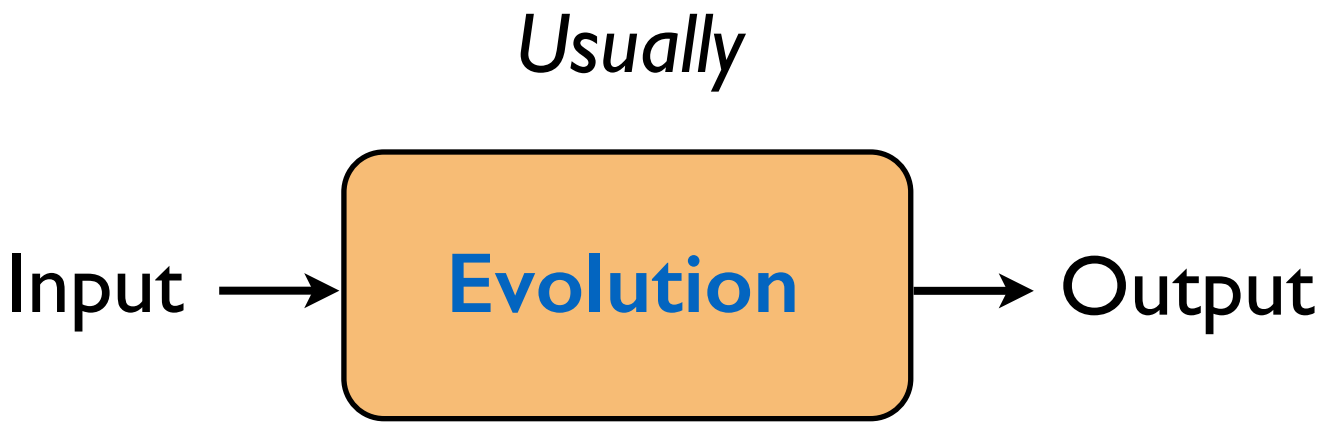
“Computers” in early 20th century

Computer Science

The science that studies computation.

Computation: manipulation of information/data.

Algorithm: rigorous description of how the data is manipulated.



The computational lens

Computational physics

Computational biology

Computational chemistry

Computational neuroscience

Computational finance

…

Defining computer science

" Computer Science deals with the theoretical foundations of information and computation, together with practical techniques for the implementation and application of the foundations. "

- *Wikipedia*

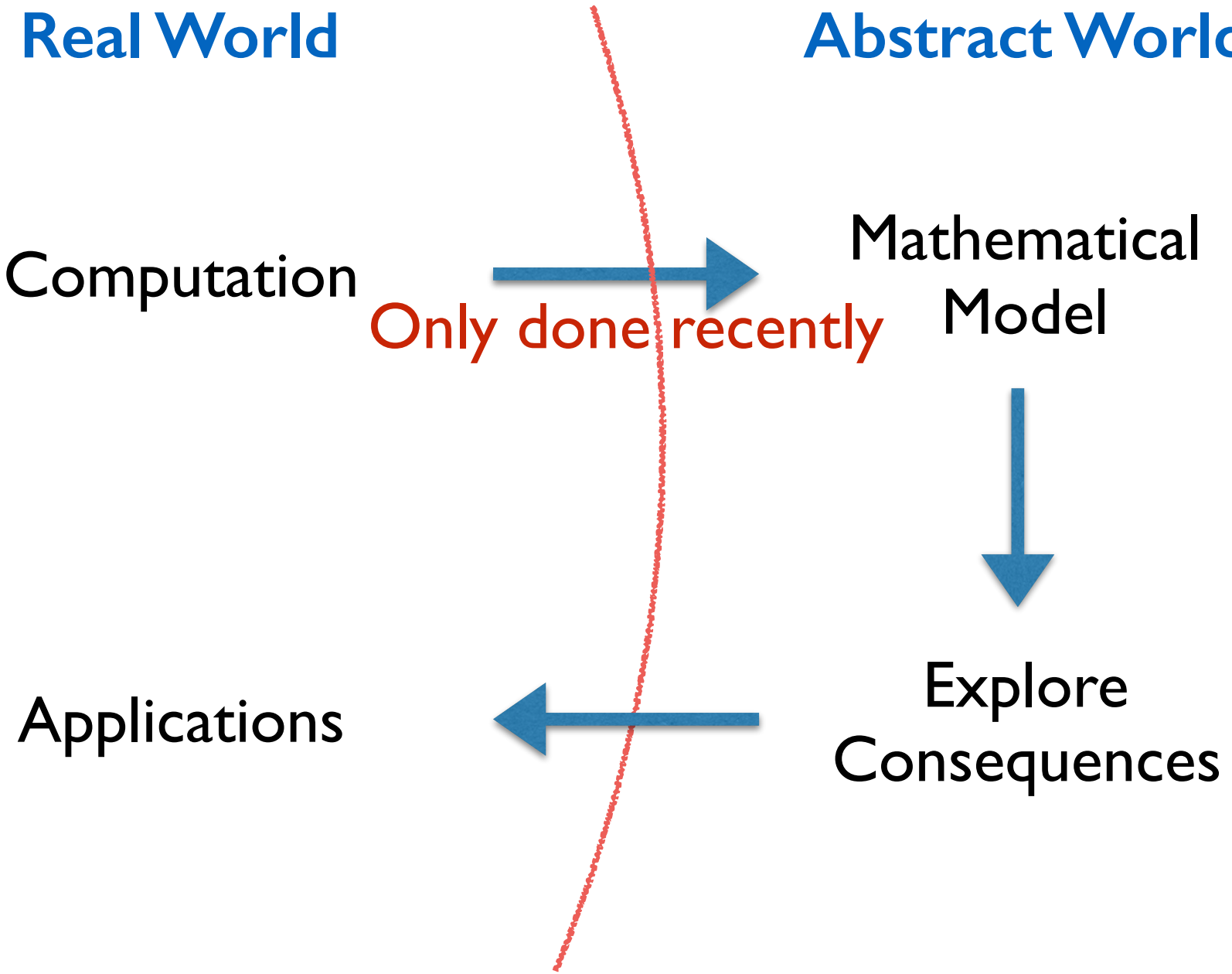
The role of theoretical computer science

Build a mathematical model for computation.

Explore the logical consequences.
Gain insight about computation.

Look for interesting applications.

The role of theoretical computer science



Simple examples of computation

$$
\begin{array}{r}
5127 \\
\times\ 4265 \\
\hline
25635 \\
307620 \\
1025400 \\
20508000 \\
\hline
21866655
\end{array}
$$

Doing computation by following a simple algorithm.

Simple examples of computation

Euclid's algorithm (~ 300BC):

```
def gcd(a, b):
    while (b != 0):
        t = b
        b = a % b
        a = t
    return a
```

We have been using algorithms for thousands of years.

Formalizing computation

We have been using algorithms for thousands of years.

Algorithm/Computation was only formalized in the 20th century!

Someone had to ask the right **question**.

David Hilbert, 1900

The Problems of Mathematics

"Who among us would not be happy to lift the veil behind which is hidden the future; to gaze at the coming developments of our science and at the secrets of its development in the centuries to come? What will be the ends toward which the spirit of future generations of mathematicians will tend? What methods, what new facts will the new century reveal in the vast and rich field of mathematical thought?"

2 of Hilbert's Problems

Hilbert's 10th problem (1900)

Is there a finitary procedure to determine if a given multivariate polynomial with integral coefficients has an integral solution?

e.g. $5x^2yz^3 + 2xy + y - 99xyz^4 = 0$

Entscheidungsproblem (1928)

Is there a finitary procedure to determine the validity of a given logical expression?

e.g. $\neg\exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$

(Mechanization of mathematics)

2 of Hilbert's Problems

Fortunately, the answer turned out to be NO.

2 of Hilbert's Problems

Gödel (1934):

Discusses some ideas for mathematical definitions of computation. But not confident what is a good definition.

Church (1936):

Invents lambda calculus.
Claims it should be the definition of an "algorithm".

Gödel, Post (1936):

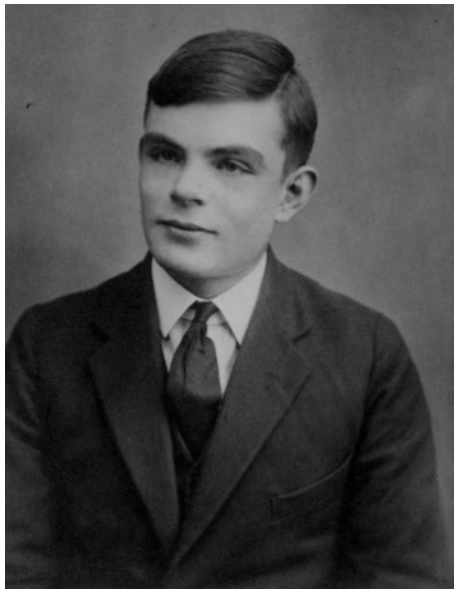
Arguments that Church's claim is not justified.

Meanwhile… in New Jersey… a certain British grad student, unaware of all these debates…

2 of Hilbert's Problems

Alan Turing (1936, age 22):

Describes a new model for computation, now known as the Turing Machine.™



Gödel, Kleene, and even Church:

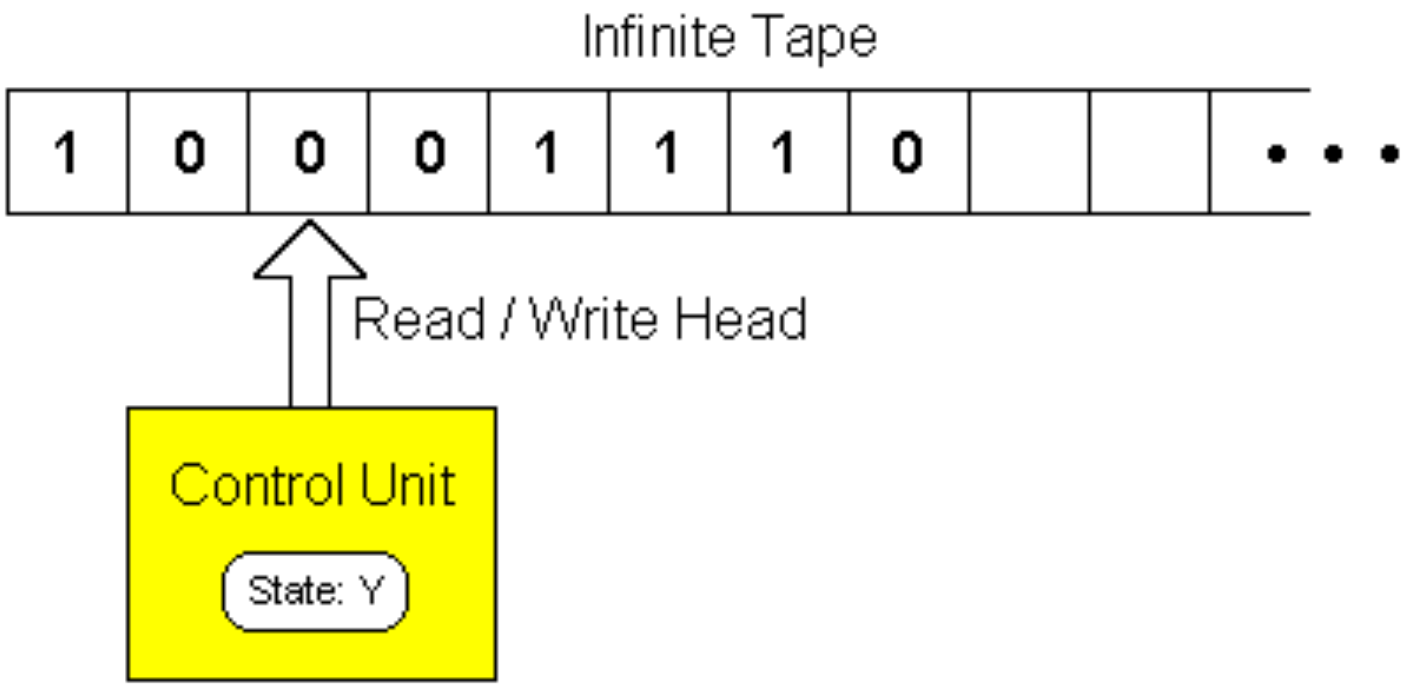
"Umm. Yeah. He nailed it. Game over. "Algorithm" defined."

Turing (1937):

TMs ≡ lambda calculus

Formalization of computation: Turing Machine

Turing Machine:



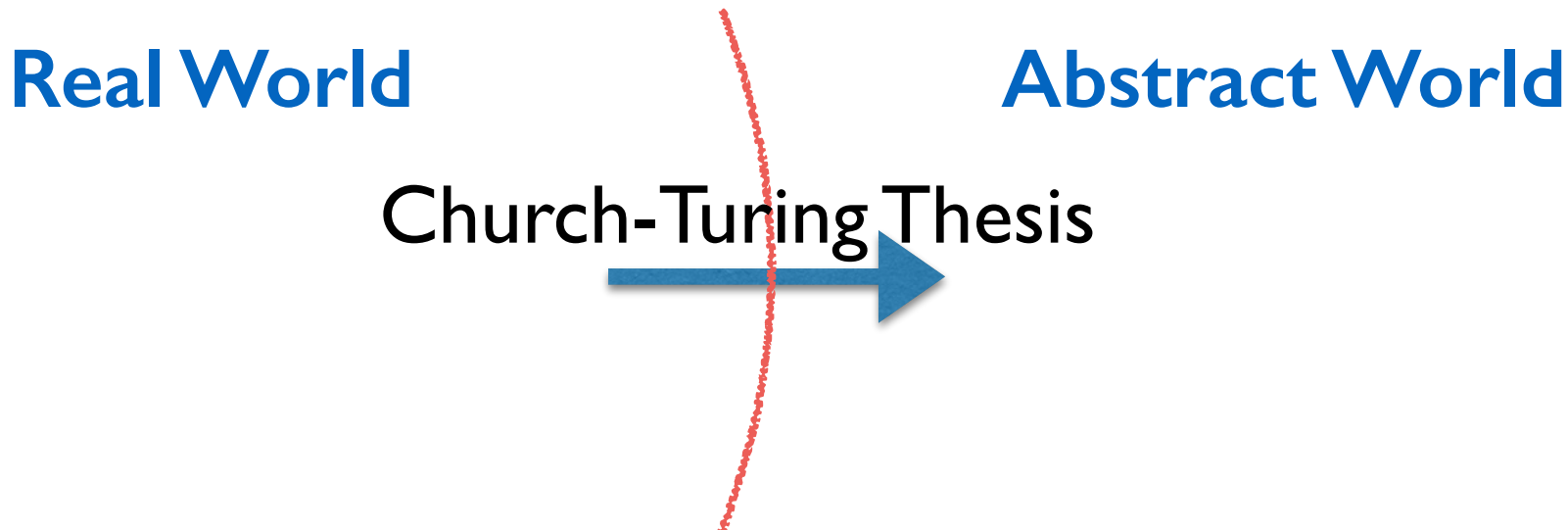
Church-Turing Thesis

Church-Turing Thesis:

The intuitive notion of "computable" is captured by functions computable by a Turing Machine.

(Physical) Church-Turing Thesis

Any computational problem that can be solved by a physical device, can be solved by a Turing Machine.



Back to 2 of Hilbert's Problems

Hilbert's 10th problem (1900)

Is there a finitary procedure to determine if a given multivariate polynomial with integral coefficients has an integral solution?

e.g. $5x^2yz^3 + 2xy + y - 99xyz^4 = 0$

Entscheidungsproblem (1928)

Is there a finitary procedure to determine the validity of a given logical expression?

e.g. $\neg\exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$

(Mechanization of mathematics)

Back to 2 of Hilbert's Problems

Hilbert's 10th problem (1900)

Is there an algorithm (a TM) to determine if a given multivariate polynomial with integral coefficients has an integral solution?

e.g. $5x^2yz^3 + 2xy + y - 99xyz^4 = 0$

Entscheidungsproblem (1928)

Is there an algorithm (a TM) to determine the validity of a given logical expression?

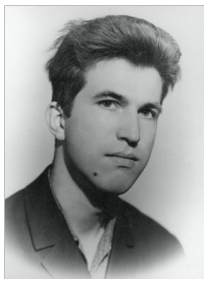
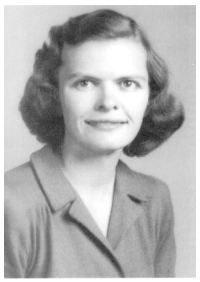
e.g. $\neg\exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$

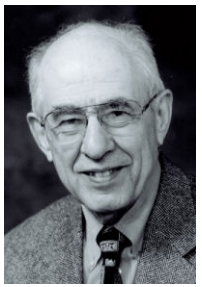
(Mechanization of mathematics)

Back to 2 of Hilbert's Problems

Hilbert's 10th problem (1900)

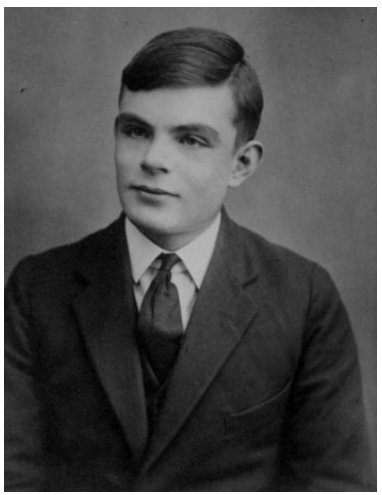
Matiyasevich-Robinson-Davis-Putnam (1970):



There is no algorithm to solve this problem.

Entscheidungsproblem (1928)

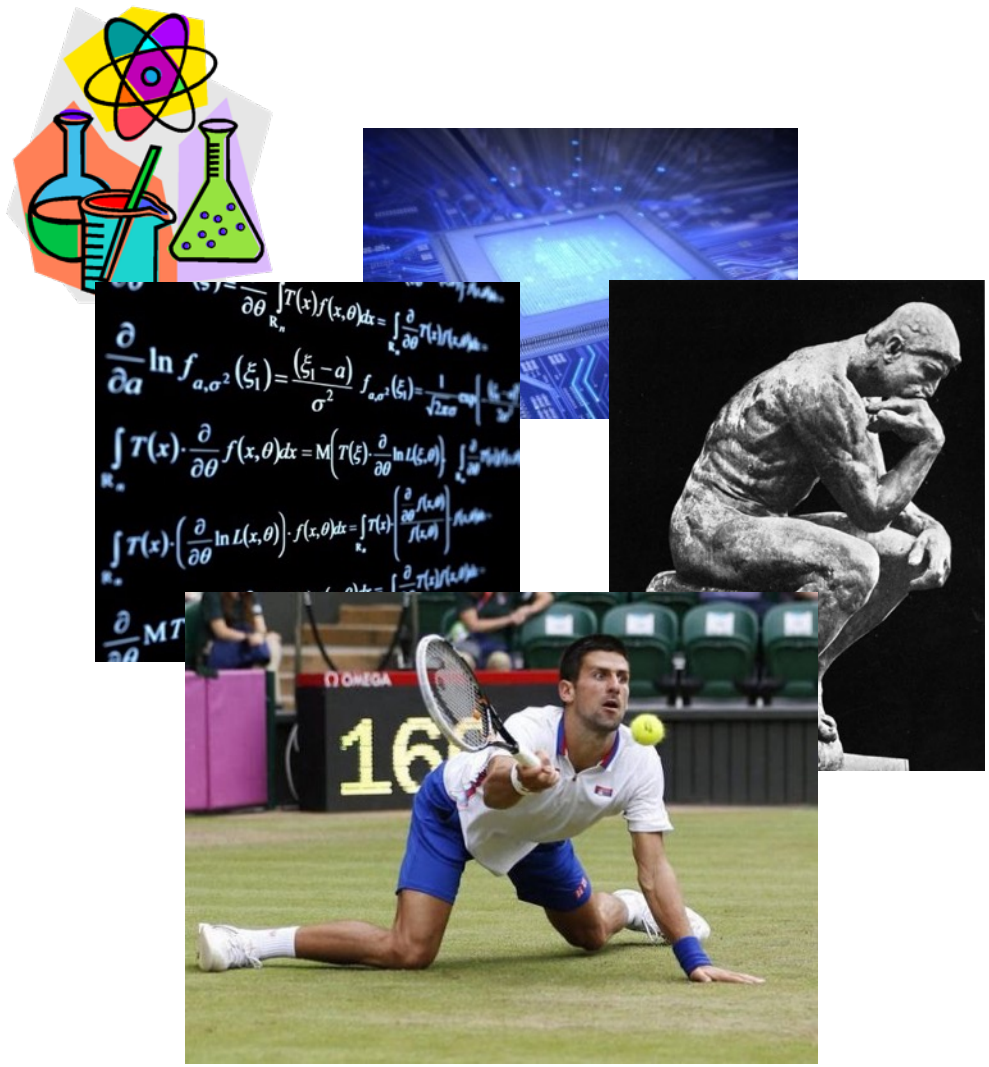


Turing (1936):

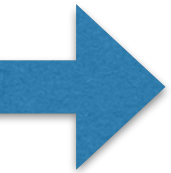
There is no algorithm to solve this problem.

Computer Science

- science?
- engineering?
- math?
- philosophy?
- sports?



2 Main Questions in TCS



Computability of a problem:

Is there an algorithm to solve it?

Complexity of a problem:

Is there an **efficient** algorithm to solve it?

- time
- space (memory)
- randomness
- quantum resources

PART 2:
Uncomputable problems

Working as a course assistant for 15-112

We need to write an autograder for

isAwesomeHappyCarolPrime

student submission

isAwesomeHappyCarolPrime

the correct program

isAwesomeHappyCarolPrime

Do they return True on exactly the same inputs?

Working as a course assistant for 15-112

We need to write an autograder for

isAwesomeHappyCarolPrime

Kosbie's version

Student submission

returns True on exactly same inputs?

True or False

Working as a course assistant for 15-112

A "simpler" problem

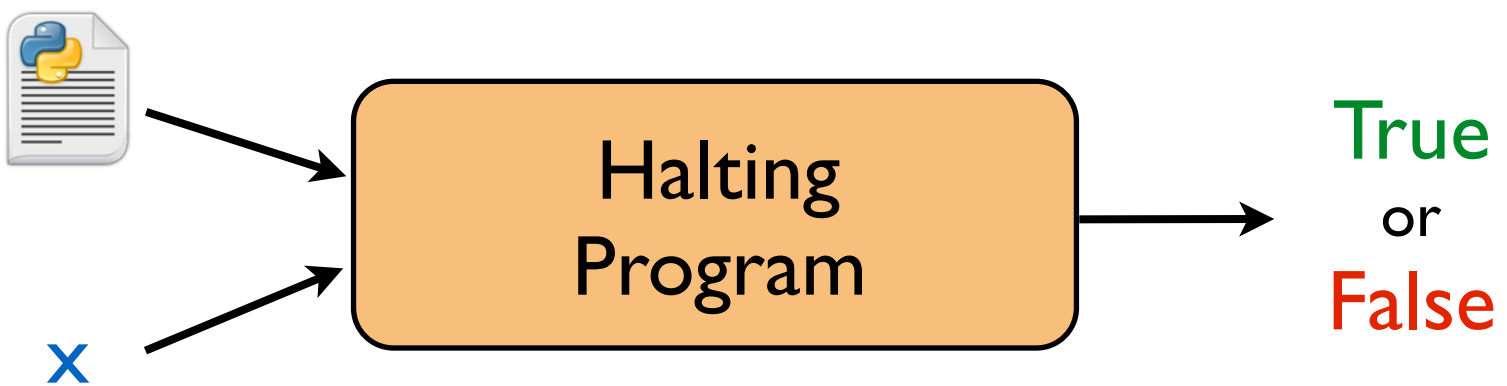
Write an autograder
that checks if a given program
goes into an infinite loop.

Working as a course assistant for 15-112

Halting Problem

Inputs: A Python program source code. 
An *input* to the program. x

Outputs: True if the program halts for the given *input*.
False otherwise.



Theorem: The **halting problem** is uncomputable.

Proof by Python

Assume such a program exists:

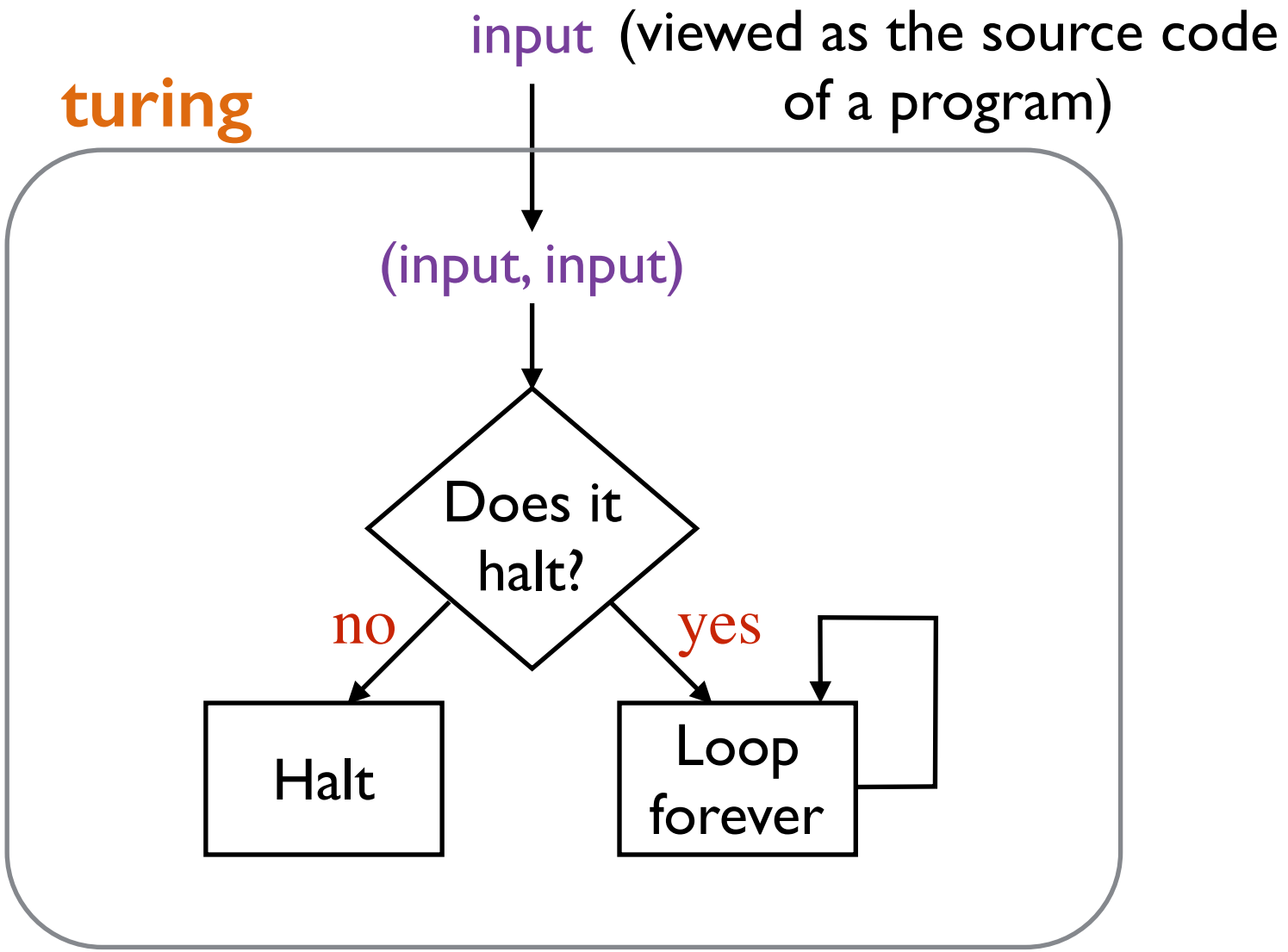
```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings
```

Proof by Python

Assume such a program exists:

```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings


def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

Proof by Python



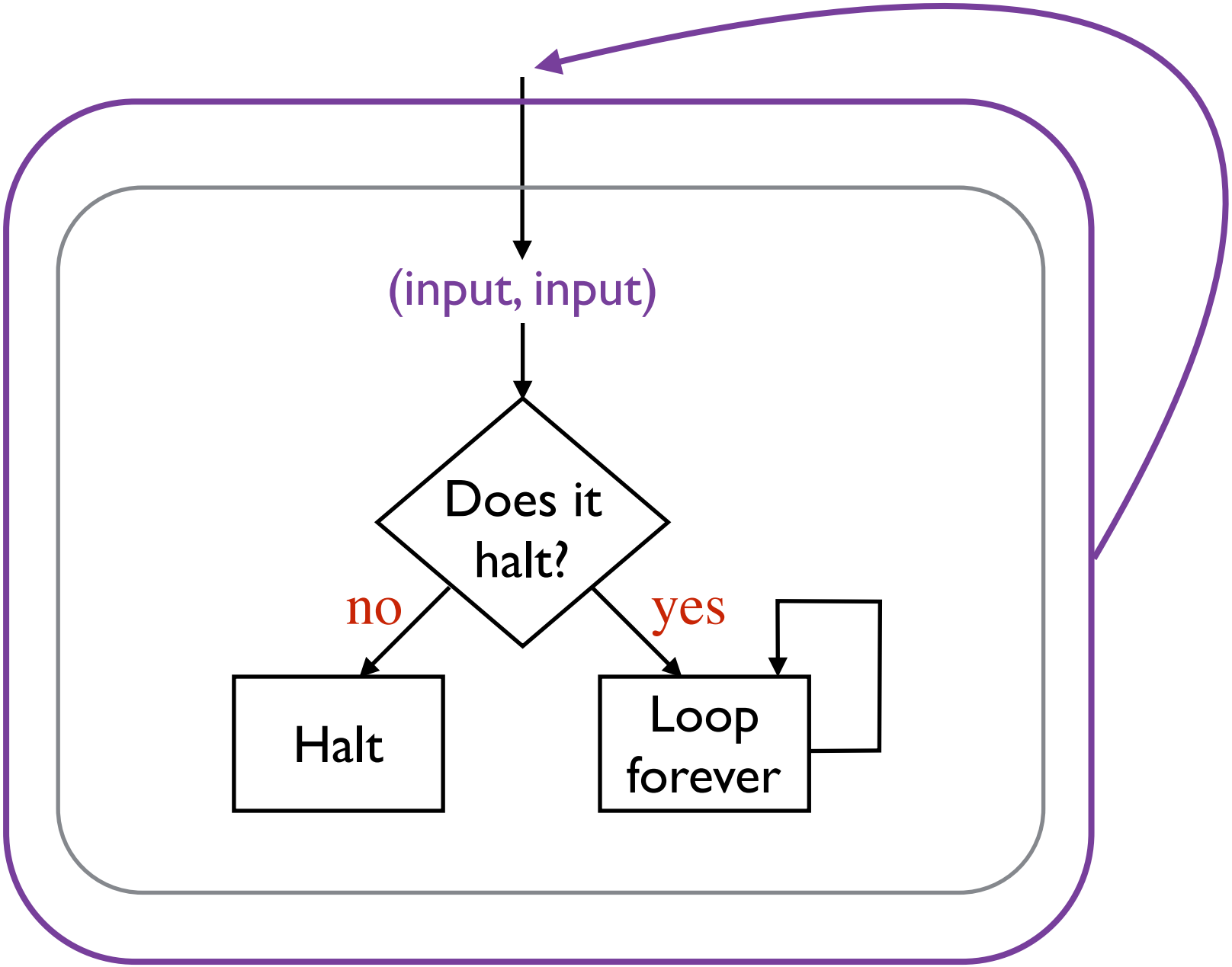
Proof by Python

Assume such a program exists:

```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings


def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

What happens when you call turing(*turing*) ?

Proof by Python



Proof by Python

Assume such a program exists:

```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings


def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

What happens when you call turing(*turing*) ?

if halt(*turing*, *turing*) is True ----> turing(*turing*) doesn't halt

if halt(*turing*, *turing*) is False ----> turing(*turing*) halts

Proof by Python

Assume such a program exists:

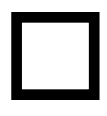
```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings


def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

What happens when you call turing(*turing*) ?

halt(*turing*, *turing*) is True ---> turing(*turing*) doesn't halt

halt(*turing*, *turing*) is False ---> turing(*turing*) halts

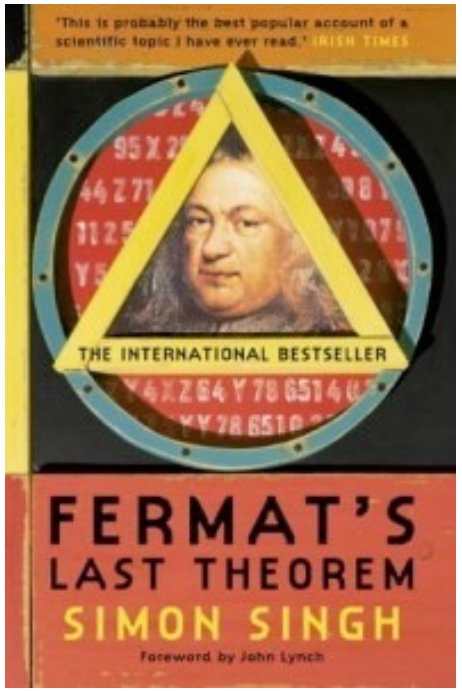
CONTRADICTION



So what?

- No guaranteed autograder program. ☹

- Consider the following program:




```
def fermat():
    t = 3
    while (True):
        for n in range(3, t+1):
            for x in range(1, t+1):
                for y in range(1, t+1):
                    for z in range(1, t+1):
                        if (x**n + y**n == z**n):  return (x, y, z, n)
        t += 1
```

<u>Question</u>: Does this program halt?

So what?

- Reductions to other problems
  imply that those problems are uncomputable as well.

Entscheidungsproblem

Is there a finitary procedure to determine the validity of a given logical expression?

e.g. $\neg\exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$

(Mechanization of mathematics)

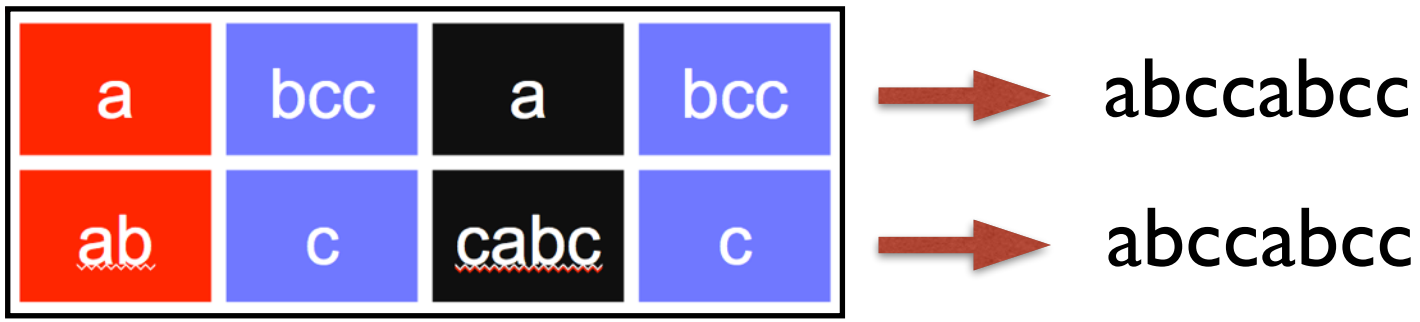
Hilbert's 10th Problem

Is there a program to determine if a given multivariate polynomial with integral coefficients has an integral solution?

So what?

Input: A finite collection of "dominoes" having strings written on each half.

Output: **True** if it is possible to match the strings.



Uncomputable!

Proved in 1946 by Post.

Mortal Matrices

Input: Two 21x21 matrices of integers A and B.

Output: **True** iff it is possible to multiply A and B together (multiple times in any order) to get to the 0 matrix.

Uncomputable!

Proved in 2007 by Halava, Harju, Hirvensalo.

So what?

Different laws of physics ----->

Different computational devices ----->

Every problem computable (???)

Can you come up with sensible laws of physics
such that the Halting Problem becomes computable?

PART 3:
Computational complexity and the “P vs NP” problem

2 Main Questions in TCS

Computability of a problem:

Is there an algorithm to solve it?

Complexity of a problem:

Is there an **efficient** algorithm to solve it?

- time
- space (memory)
- randomness
- quantum resources

Computational Complexity

2 camps:

- trying to come up with efficient algorithms (algorithm designers)

- trying to show **no** efficient algorithm exists (complexity theorists)

Computational Complexity

2 camps:

- trying to come up with efficient algorithms (algorithm designers)

- trying to show **no** efficient algorithm exists (complexity theorists)

What is efficient?

$T(N)$ = (worst-case) running time for inputs of length *N*

"Efficient"

$T(N) = N$

$T(N) = N^2$

$T(N) = N^3$

$T(N) = N^k$

(polynomial time)

Inefficient

Anything not poly-time

$T(N) = 2^N$

(exponential time)

practically uncomputable!

Exponential running time example 1

Subset Sum Problem

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exponential running time example I

Subset Sum Problem

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exhaustive Search (Brute Force Search):

> Try every possible subset and see if it sums to 0.

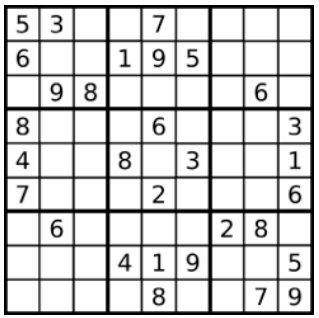
subsets is 2^N $\Longrightarrow$ running time at least 2^N

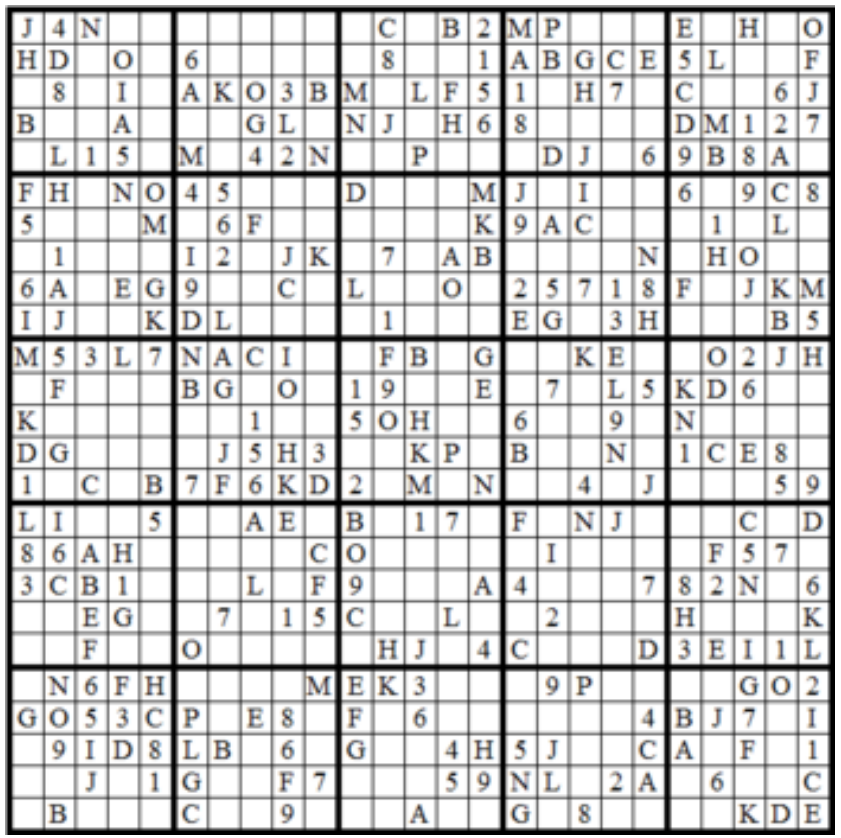
Note: checking if a given subset sums to 0 is **easy**.

Exponential running time example 2

Sudoku Problem

Given a partially filled *N* by *N* sudoku board, determine if there is a solution.





Exponential running time example 2

Sudoku Problem

Given a partially filled *N* by *N* sudoku board, determine if there is a solution.

Exhaustive Search (Brute Force Search):

> Try every possible way of filling the empty cells.

If the initial board is say half empty,
possibilities is $N^{N^2/2}$.

Note: checking if a given solution is correct is **easy**.

Exponential running time example 3

Theorem Proving Problem

(informal description)

Given a mathematical proposition P and an integer k, determine if P has a proof of length at most k.

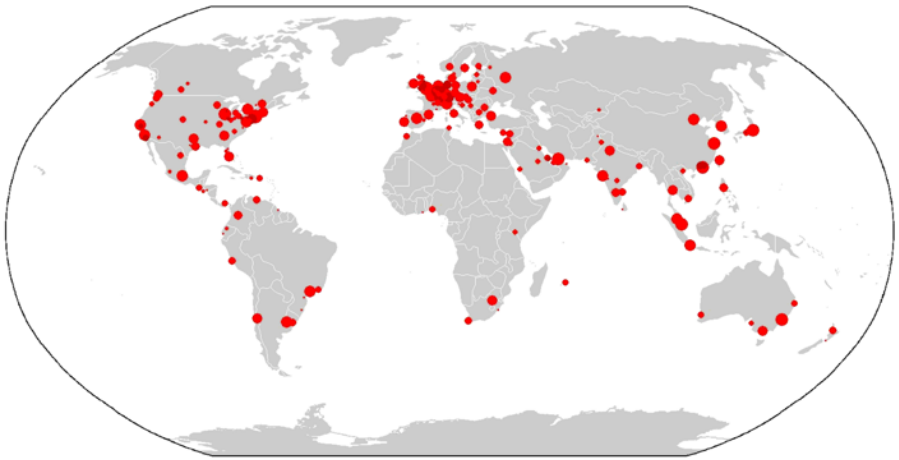
Exhaustive Search (Brute Force Search):

> Try every possible "proof" of length at most k, and check if it corresponds to a valid proof.

Note: checking if a given proof is correct is **easy**.

Exponential running time example 4

Traveling Salesperson Problem



In which order should you visit the cities so that ticket cost is < $10000?

Exhaustive Search (Brute Force Search):

> Try every possibility.

Note: checking if a given solution has the desired cost is **easy**.

Exponential running time example 5

Tetris Problem

Given a sequence of Tetris pieces, and a number k, can you clear more than k lines?

Exhaustive Search (Brute Force Search):

> Try every possibile way of putting the pieces.

Note: checking if a given solution clears more than k lines **easy**.

Other interesting computational problems

> multiplying two integers
> sorting a list
> factoring integers
> computing Nash Equilibria of games
> DNA sequence allignment
> simulation of quantum systems

…

Some we know how to solve efficiently.

- still would be awesome to improve the running time.

Some we don't know how to solve efficiently.

- some we hope can be solved efficiently.
- some we hope **can't** be solved efficiently.

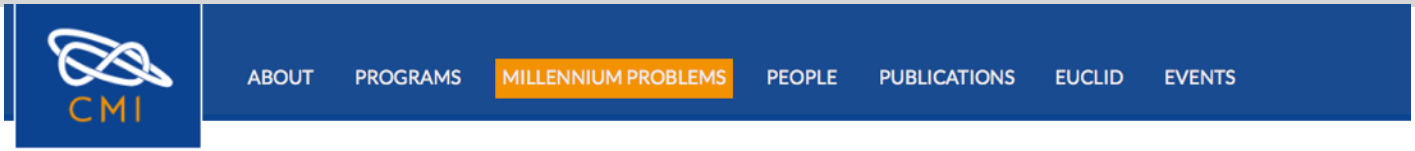
In our quest to understand efficient computation,
(and life, the universe, and everything)
we come across:

P vs NP problem

"Can creativity be automated?"

Biggest open problem in all of Computer Science.
One of the biggest open problems in all of Mathematics.

P vs NP



Millennium Problems

1 million dollar question

Yang–Mills and Mass Gap

Experiment and computer simulations suggest the existence of a "mass gap" in the solution to the quantum versions of the Yang-Mills equations. But no proof of this property is known.

Riemann Hypothesis

The prime number theorem determines the average distribution of the primes. The Riemann hypothesis tells us about the deviation from the average. Formulated in Riemann's 1859 paper, it asserts that all the 'non-obvious' zeros of the zeta function are complex numbers with real part 1/2.

P vs NP Problem

If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

Navier–Stokes Equation

This is the equation which governs the flow of fluids such as water and air. However, there is no proof for the most basic questions one can ask: do solutions exist, and are they unique? Why ask for a proof? Because a proof gives not only certitude, but also understanding.

Hodge Conjecture

The answer to this conjecture determines how much of the topology of the solution set of a system of algebraic equations can be defined in terms of further algebraic equations. The Hodge conjecture is known in certain special cases, e.g., when the solution set has dimension less than four. But in dimension four it is unknown.

Poincaré Conjecture

In 1904 the French mathematician Henri Poincaré asked if the three dimensional sphere is characterized as the unique simply connected three manifold. This question, the Poincaré conjecture, was a special case of Thurston's geometrization conjecture. Perelman's proof tells us that every three manifold is built from a set of standard pieces, each with one of eight well-understood geometries.

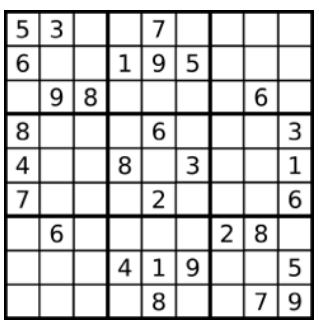
Birch and Swinnerton-Dyer Conjecture

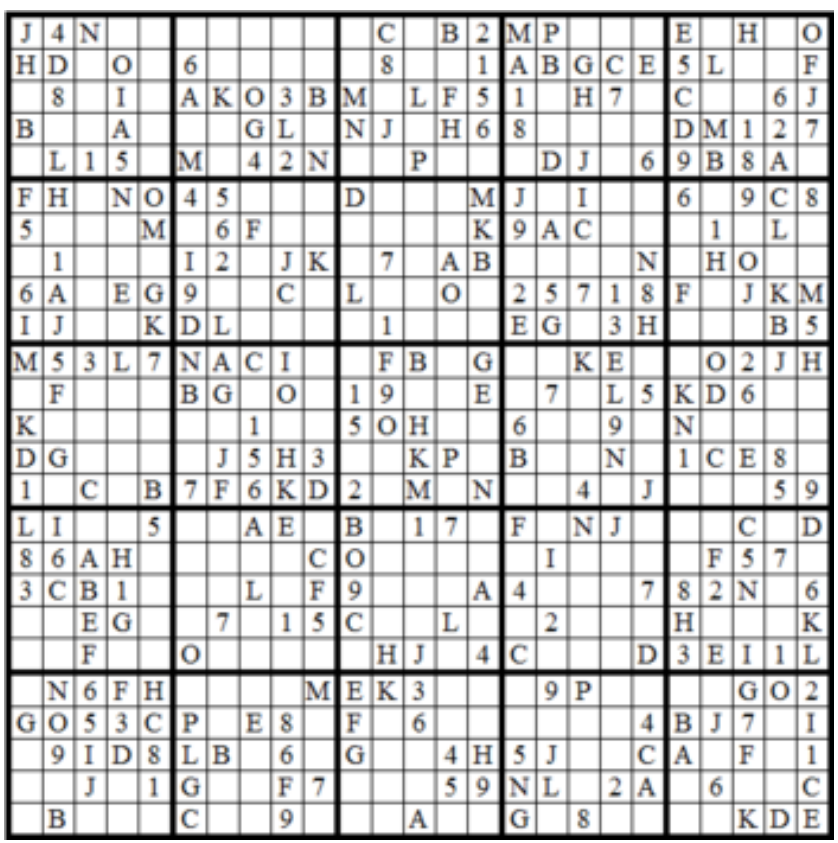
Supported by much experimental evidence, this conjecture relates the number of points on an elliptic curve mod p to the rank of the group of rational points. Elliptic curves, defined by cubic equations in two variables, are fundamental mathematical objects that arise in many areas: Wiles' proof of the Fermat Conjecture, factorization of numbers into primes, and cryptography, to name three.

So what is the P vs NP question?

The P vs NP question is the following:

Can the Sudoku problem be solved in polynomial time?





WTF?!

So what is the P vs NP question?

The P vs NP question is the following:

Can the Tetris problem be solved in poly-time?

So what is the P vs NP question?

The P vs NP question is the following:

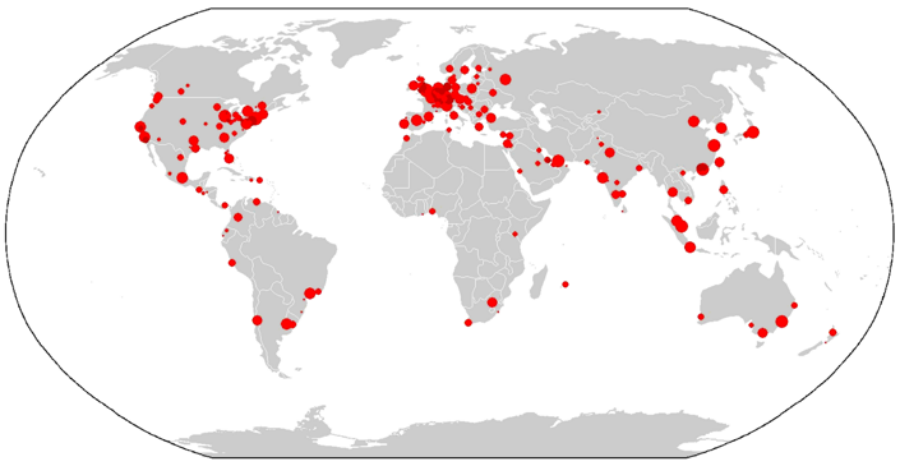
Can the Subset Sum problem be solved in poly-time?

4	-3	-2	7	99	5	1

So what is the P vs NP question?

The P vs NP question is the following:

Can the Traveling Salesperson problem be solved in poly-time?



So what is the P vs NP question?

The P vs NP question is the following:

Can the Theorem Proving problem be solved in poly-time?

What is going on?!?

Understanding P vs NP

P = {problems computable in polynomial time}

(think: "efficiently solvable problems")

NP = {problems whose solutions can be verified in polynomial time}

Problems we have seen in this section are all in NP.

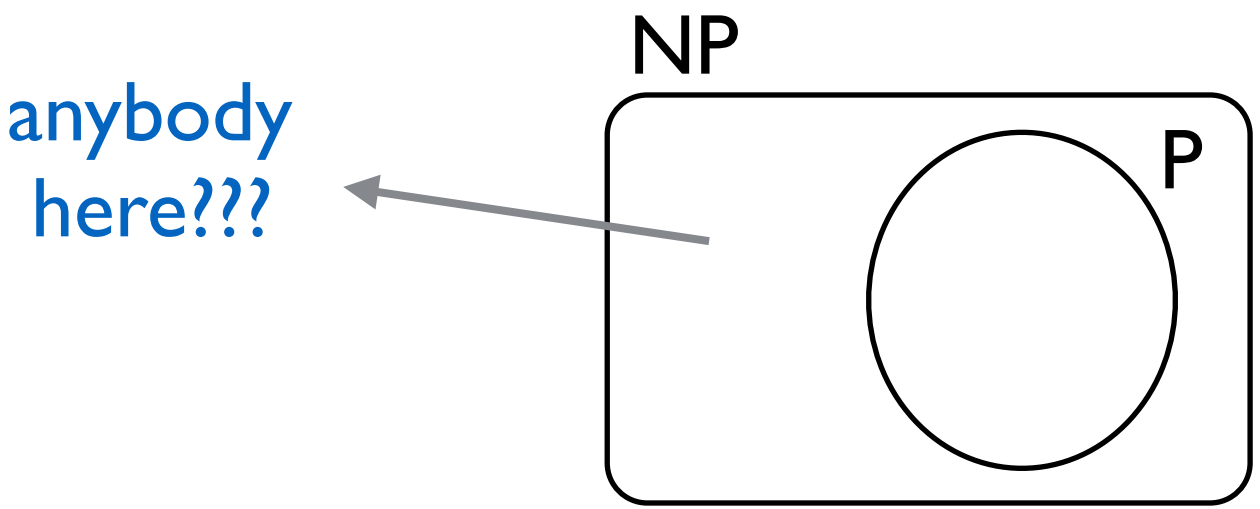
(think: "problems that can be solved using brute force search.")

Note: these are not formal definitions.

P vs NP problem

"Can creativity be automated?"

is the question $P \stackrel{?}{=} NP$



What does P and NP stand for?

P stands for Polynomial time.

How about **NP**

Not Polynomial?

None Polynomial?

No Polynomial?

Nurse Practitioner?

It stands for Nondeterministic Polynomial time.

What does P and NP stand for?

Other contenders for the name of the class:

Herculean

Formidable

Hard-boiled

PET "possibly exponential time"

"provably exponential time"

"previously exponential time"

NP-completeness

How is **P vs NP** equivalent to whether Sudoku can be solved efficiently???

Definition:

We say that a problem A is **NP-complete** if

1) A is in NP

2) A has the following property:

- **if** A has a poly-time algorithm,
  every problem in NP has a poly-time algorithm.

(think: "NP-complete problems are the hardest ones in NP")

NP-completeness

How is **P vs NP** equivalent to whether Sudoku can be solved efficiently???

Definition:

We say that a problem A is **NP-complete** if

1) A is in NP

2) A has the following property:

- **if** A has a poly-time algorithm,
  every problem in NP has a poly-time algorithm.

Why would any problem satisfy this definition?!

NP-completeness

Some NP-complete problems:

Sudoku problem

Tetris problem

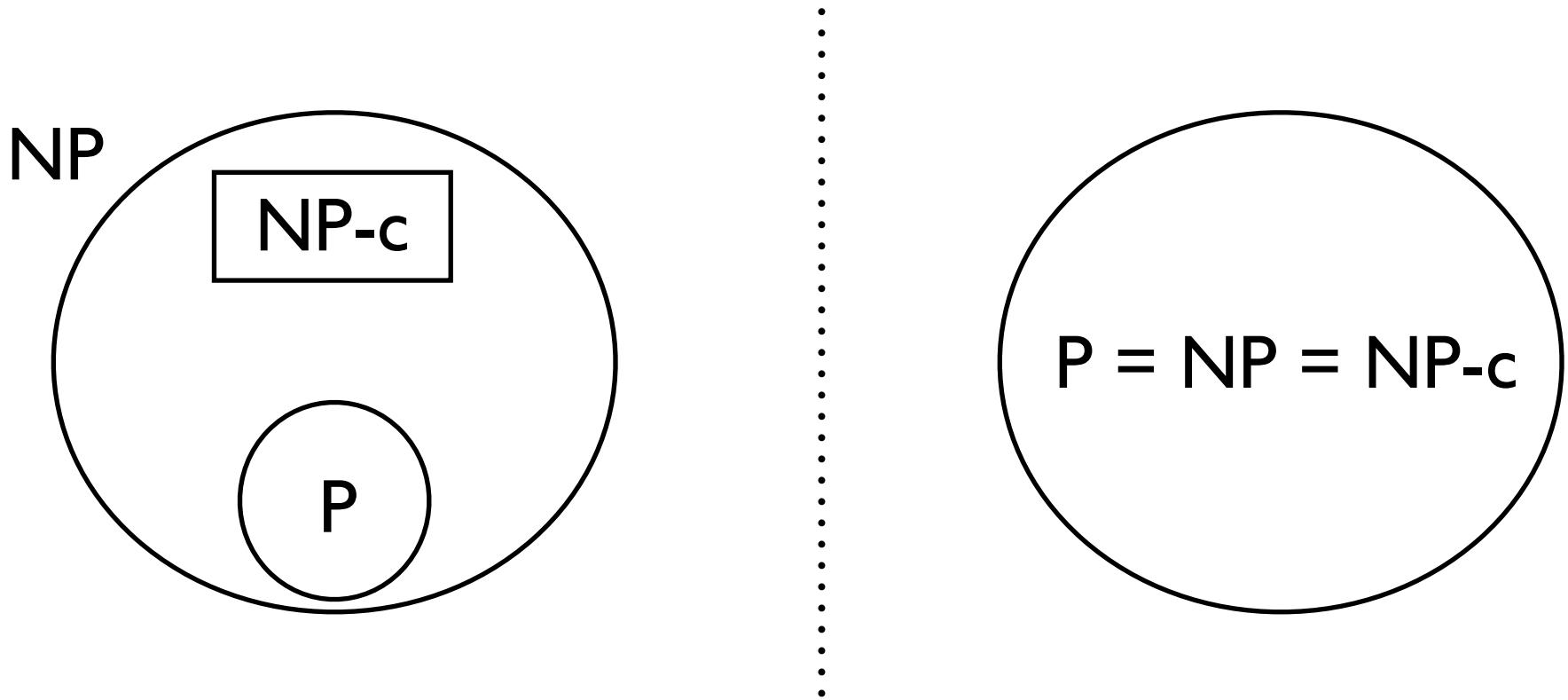
Traveling Salesperson problem

Theorem Proving problem

…

(and thousands of more)

Two possible worlds



If you show a problem is NP-complete,
that is considered good evidence that the problem is hard.

(most people believe **P ≠ NP**)

How do you show a problem is NP-complete?!?

Take 15-251 and I'll show you.