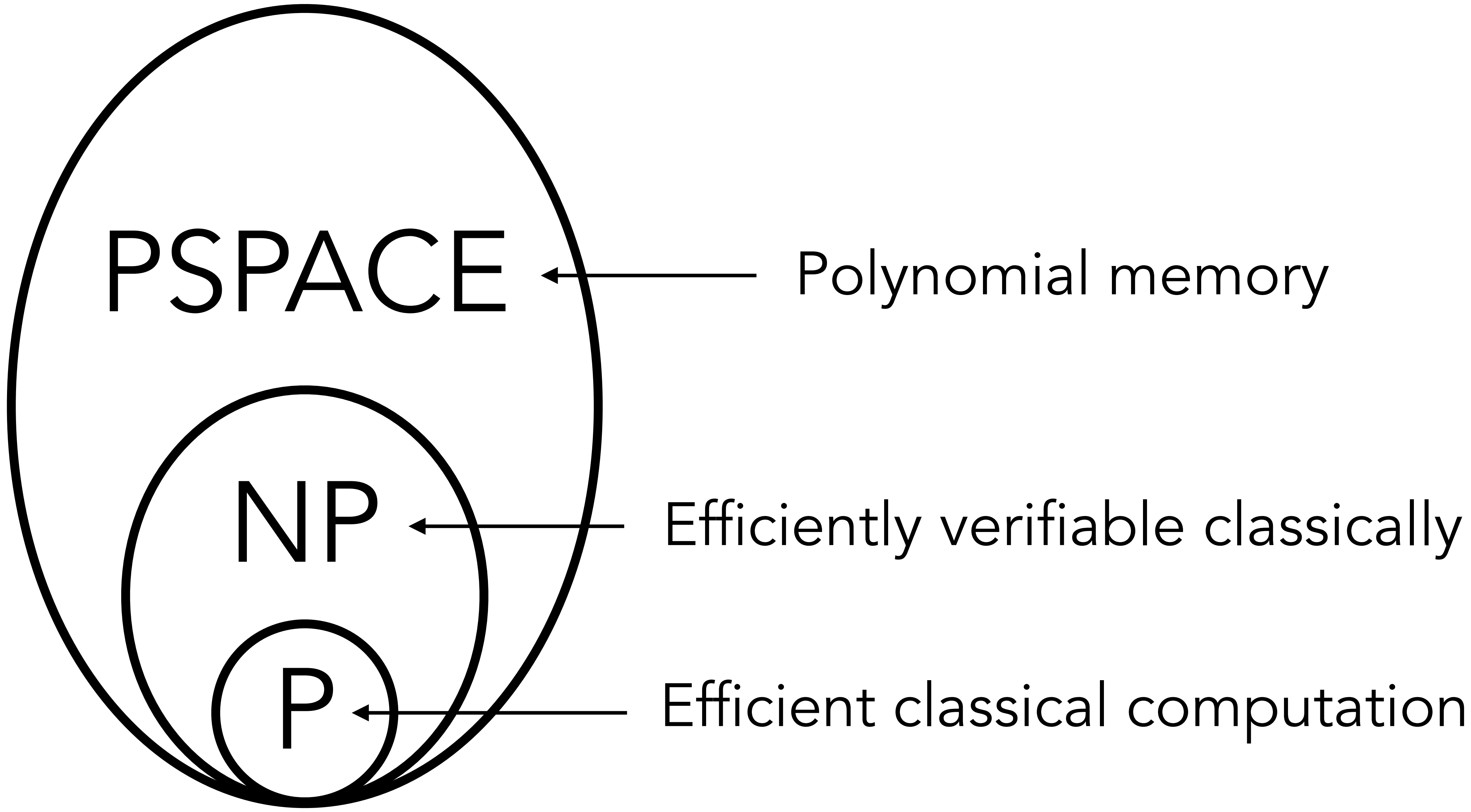
Computational Complexity Basics

A very short introduction

Complexity Theory

Study of computational resources needed
to solve different problems

Time? Memory?
Randomness?
Proofs? Interaction?



Turing machine and Decision Problems

Turing Machine mathematically formalizes what an algorithm is

You can think of it as a piece of code in some programming language that takes an input and gives an output

Turing Machine may take some auxiliary inputs such as random bits, advice, etc.

Turing machine and Decision Problems

Turing Machine mathematically formalizes what an algorithm is

You can think of it as a piece of code in some programming language that takes an input and gives an output

Turing Machine may take some auxiliary inputs such as random bits, advice, etc.

Decision Problems

Typically, we are interested in resources (e.g. time, space) required for computing functions

$$f : \{0,1\}^* \to \{0,1\}$$

Sometimes promise functions or promised languages or search problems are considered

Languages, problems, functions are often used synonymously

Or equivalently, deciding whether a bitstring is in a language

A language is a subset of $\{0,1\}^*$
e.g. $L = \{x \mid f(x) = 1\}$

The Landscape of Complexity

The Landscape of Complexity

called **undecidable problems**

There are problems that cannot be solved by a Turing machine in any finite time

e.g. HALT(c, x)=1 iff the Turing machine whose code is c halts in finite time on input x

The Landscape of Complexity

called **undecidable problems**

There are problems that cannot be solved by a Turing machine in any finite time

e.g. HALT(c, x)=1 iff the Turing machine whose code is c halts in finite time on input x

P is the class of languages which can be decided with a Turing Machine that runs in polynomial time in the input length

Typically, we call polynomial run time as efficient

e.g. Linear Programming = Decide if a system of linear inequalities $a_i^\top x \leq b_i$ has a solution

e.g. 2SAT = Given a set of boolean 2-clauses (e.g. $x_i \vee \overline{x_j}$) decide if there is an assignment satisfying all clauses

The Landscape of Complexity

called **undecidable problems**

There are problems that cannot be solved by a Turing machine in any finite time

e.g. HALT(c, x)=1 iff the Turing machine whose code is c halts in finite time on input x

P is the class of languages which can be decided with a Turing Machine that runs in polynomial time in the input length

Typically, we call polynomial run time as efficient

e.g. Linear Programming = Decide if a system of linear inequalities $a_i^\top x \leq b_i$ has a solution

e.g. 2SAT = Given a set of boolean 2-clauses (e.g. $x_i \vee \overline{x}_j$) decide if there is an assignment satisfying all clauses

NP is the class of languages which can be efficiently decided when the Turing machine is given a polynomial-sized witness

e.g. 3SAT = Given a set of boolean 3-clauses (e.g. $x_i \vee \overline{x}_j \vee x_k$) decide if there is a satisfying assignment

The Landscape of Complexity

called **undecidable problems**

There are problems that cannot be solved by a Turing machine in any finite time

e.g. HALT(c, x)=1 iff the Turing machine whose code is c halts in finite time on input x

P is the class of languages which can be decided with a Turing Machine that runs in polynomial time in the input length

Typically, we call polynomial run time as efficient

e.g. Linear Programming = Decide if a system of linear inequalities $a_i^\top x \leq b_i$ has a solution

e.g. 2SAT = Given a set of boolean 2-clauses (e.g. $x_i \vee \overline{x}_j$) decide if there is an assignment satisfying all clauses

NP is the class of languages which can be efficiently decided when the Turing machine is given a polynomial-sized witness

e.g. 3SAT = Given a set of boolean 3-clauses (e.g. $x_i \vee \overline{x}_j \vee x_k$) decide if there is a satisfying assignment

P $\subseteq$ NP but whether it is a strict subset is a Millennium Prize Problem

Why?

The Landscape of Complexity

The Landscape of Complexity

PSPACE is the class of languages which can be decided with a Turing Machine that uses polynomial space

The Landscape of Complexity

PSPACE is the class of languages which can be decided with a Turing Machine that uses polynomial space

BPP is the class of languages which can be efficiently decided with a **randomized** Turing machine

If $x \in L$ then $\mathbb{P}[\text{TM says "}x \in L\text{"}] \geq 2/3$

If $x \notin L$ then $\mathbb{P}[\text{TM says "}x \in L\text{"}] \leq 1/3$

The probabilities $(2/3,1/3)$ can be made $(1-\epsilon,\epsilon)$: run multiple independent instances and take the majority outcome

The Landscape of Complexity

PSPACE is the class of languages which can be decided with a Turing Machine that uses polynomial space

BPP is the class of languages which can be efficiently decided with a **randomized** Turing machine

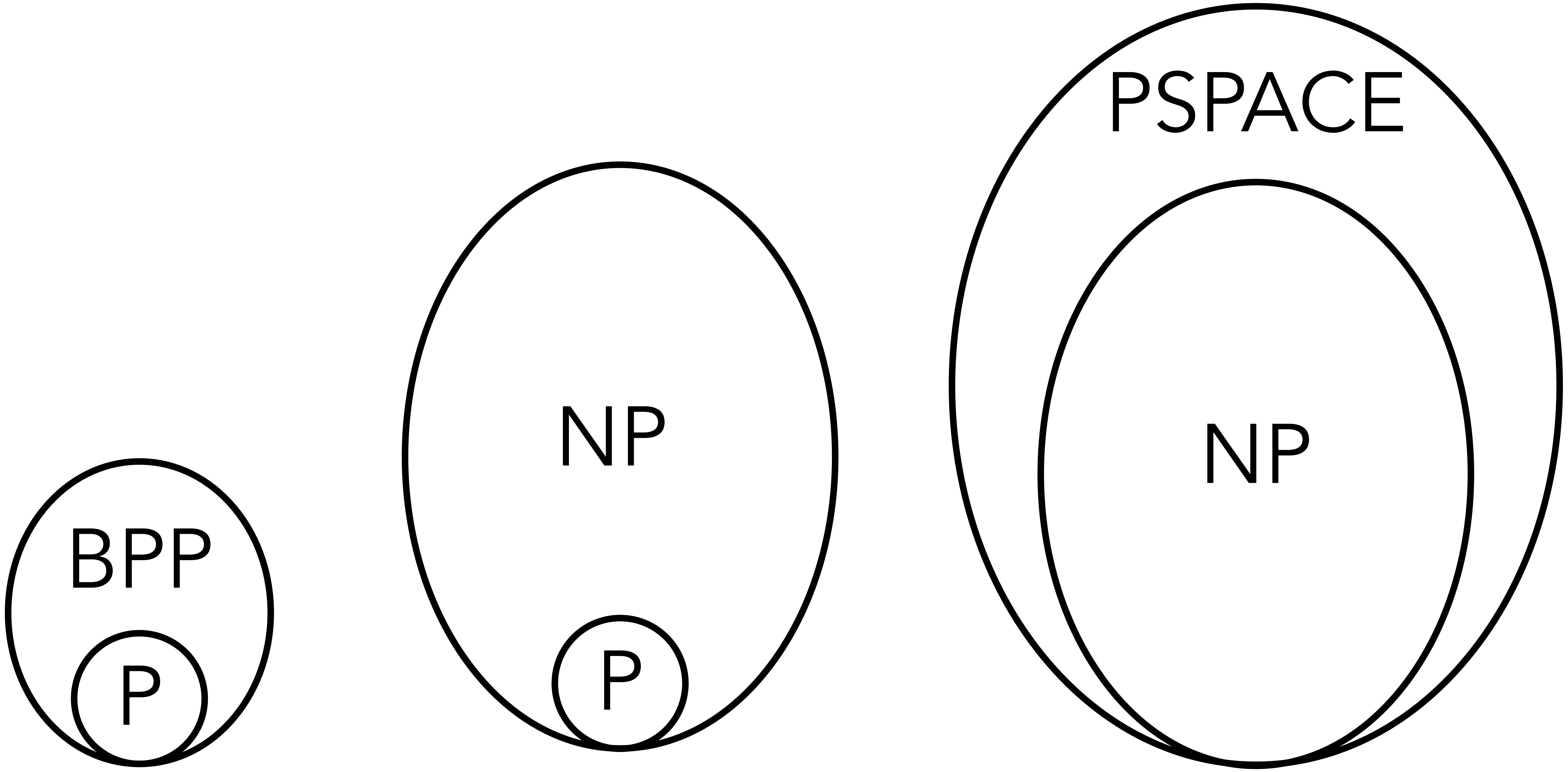
If $x \in L$ then $\mathbb{P}$[TM says "$x \in L$"] $\geq 2/3$

If $x \notin L$ then $\mathbb{P}$[TM says "$x \in L$"] $\leq 1/3$

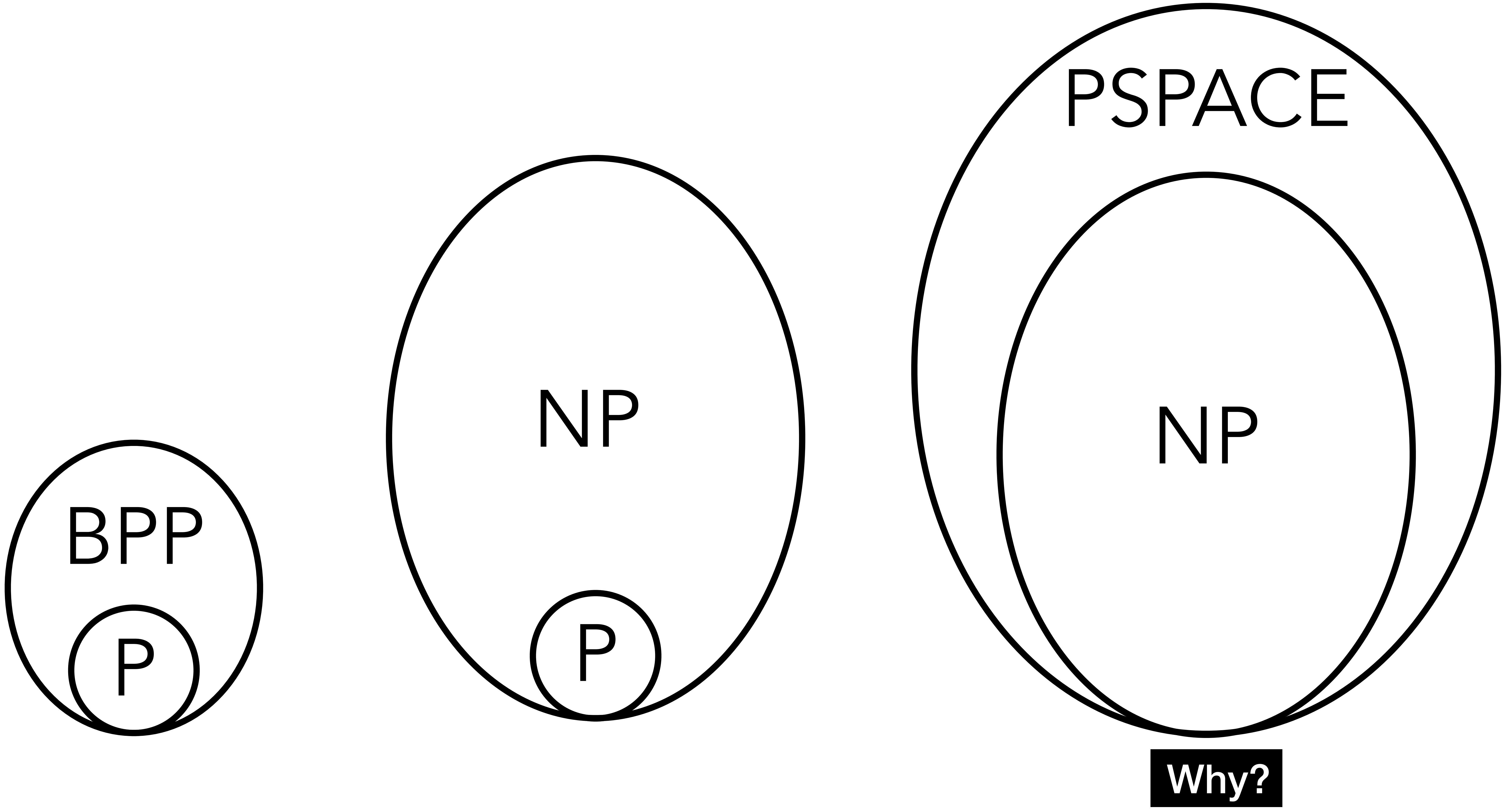
The probabilities $(2/3,1/3)$ can be made $(1-\epsilon, \epsilon)$: run multiple independent instances and take the majority outcome

It is believed that randomness does not help efficiency in computation, i.e. BPP=P

The Landscape of Complexity

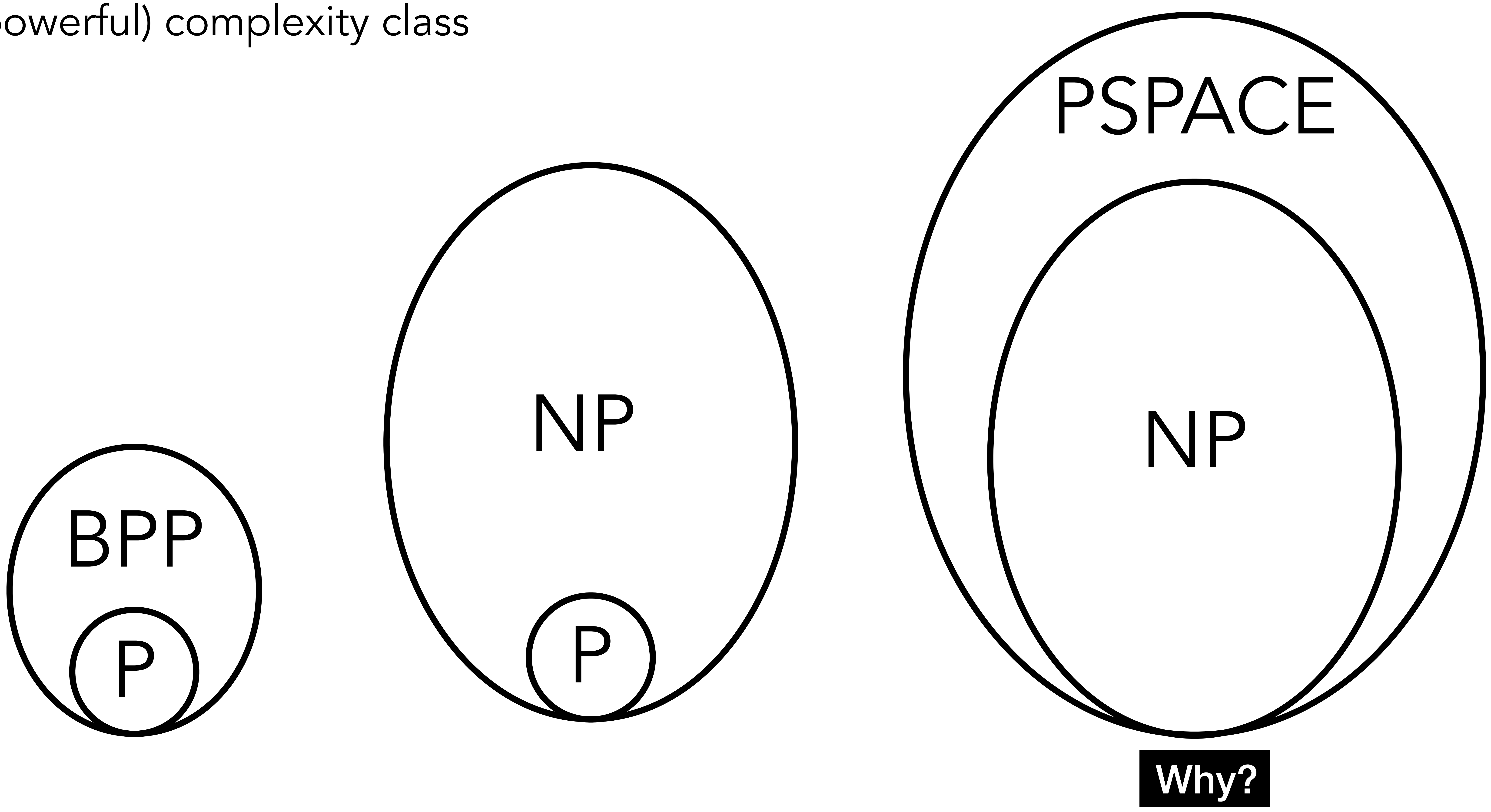


The Landscape of Complexity



The Landscape of Complexity

In general, we do not have any techniques to show that a problem does not lie in (a reasonably powerful) complexity class

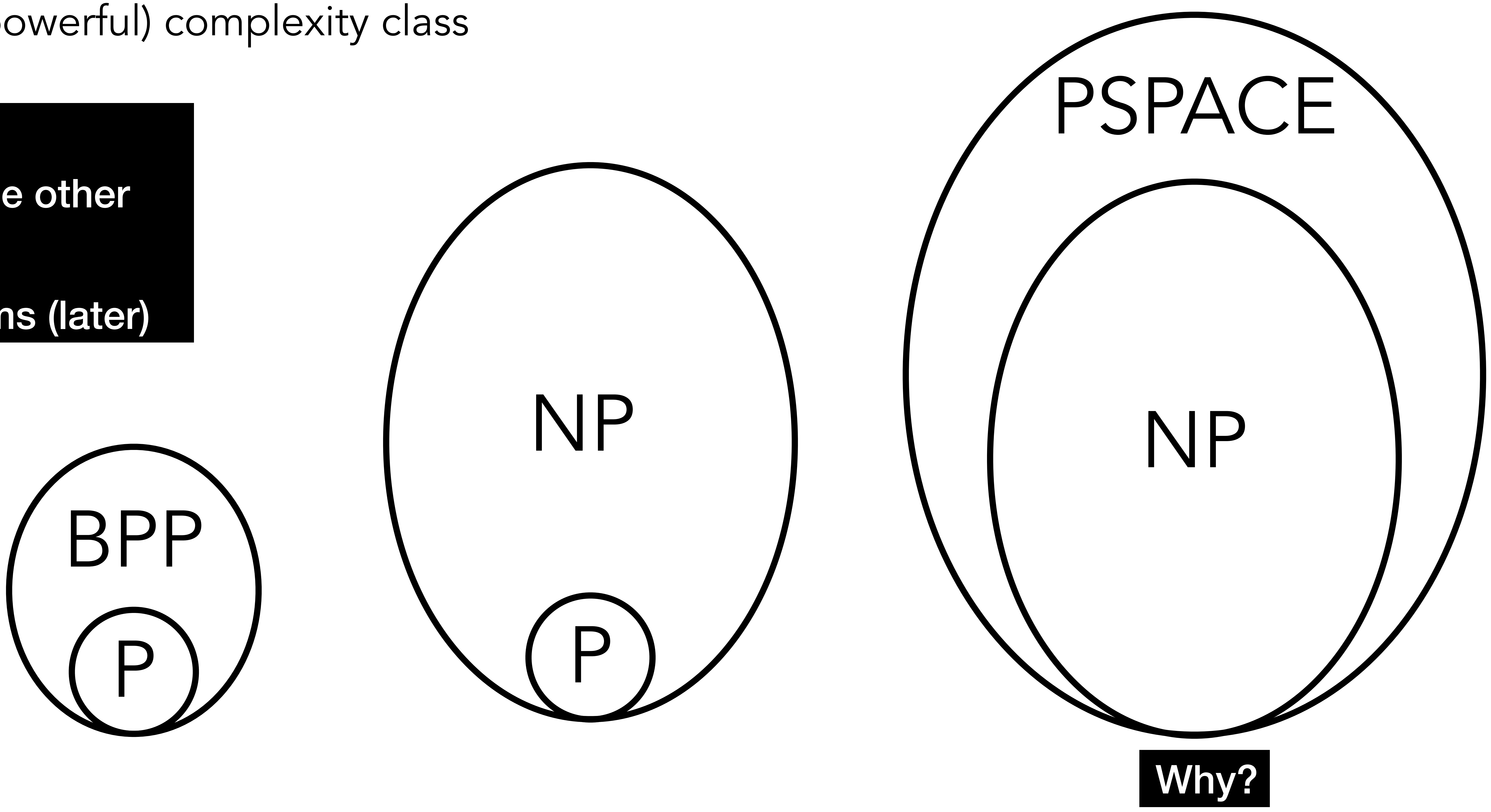


The Landscape of Complexity

In general, we do not have any techniques to show that a problem does not lie in (a reasonably powerful) complexity class

Alternatives?

- Say one problem is harder than the other
- Use Oracles (in a later lecture)
- Study restricted types of algorithms (later)



Reductions and Complete Problems

Reduction This formalizes that one problem is harder than another

$L_{easy} \leq L$ if an efficient algorithm for solving L gives an efficient algorithm for solving L_{easy}

The type of reductions depends on the complexity classes, e.g. reductions could be randomized, quantum, space-limited

Reductions and Complete Problems

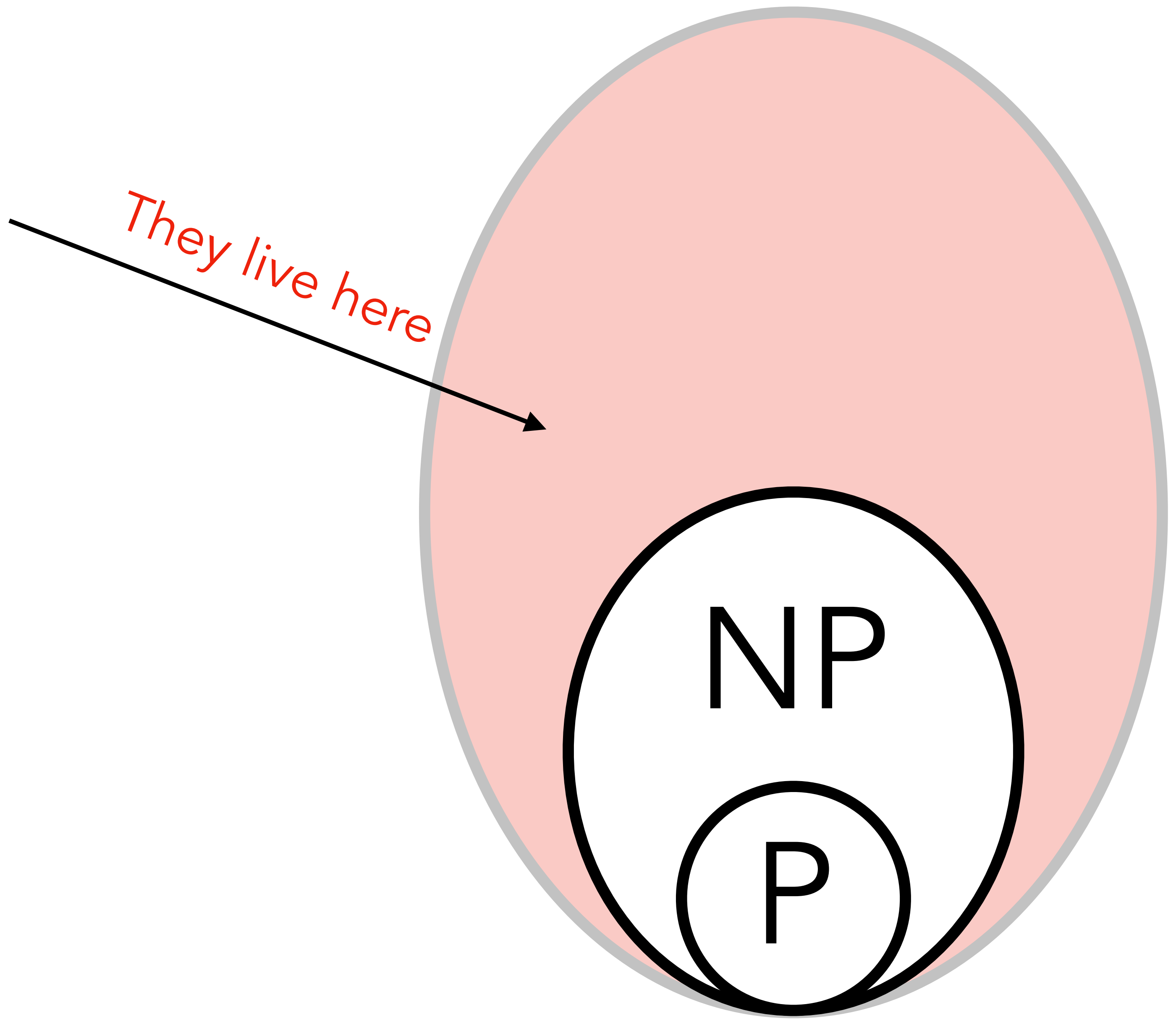
Reduction This formalizes that one problem is harder than another

$L_{easy} \leq L$ if an efficient algorithm for solving L gives an efficient algorithm for solving L_{easy}

The type of reductions depends on the complexity classes, e.g. reductions could be randomized, quantum, space-limited

E.g. every problem in NP is easier than 3SAT

Such problems are called **NP-hard**



Reductions and Complete Problems

Reduction This formalizes that one problem is harder than another

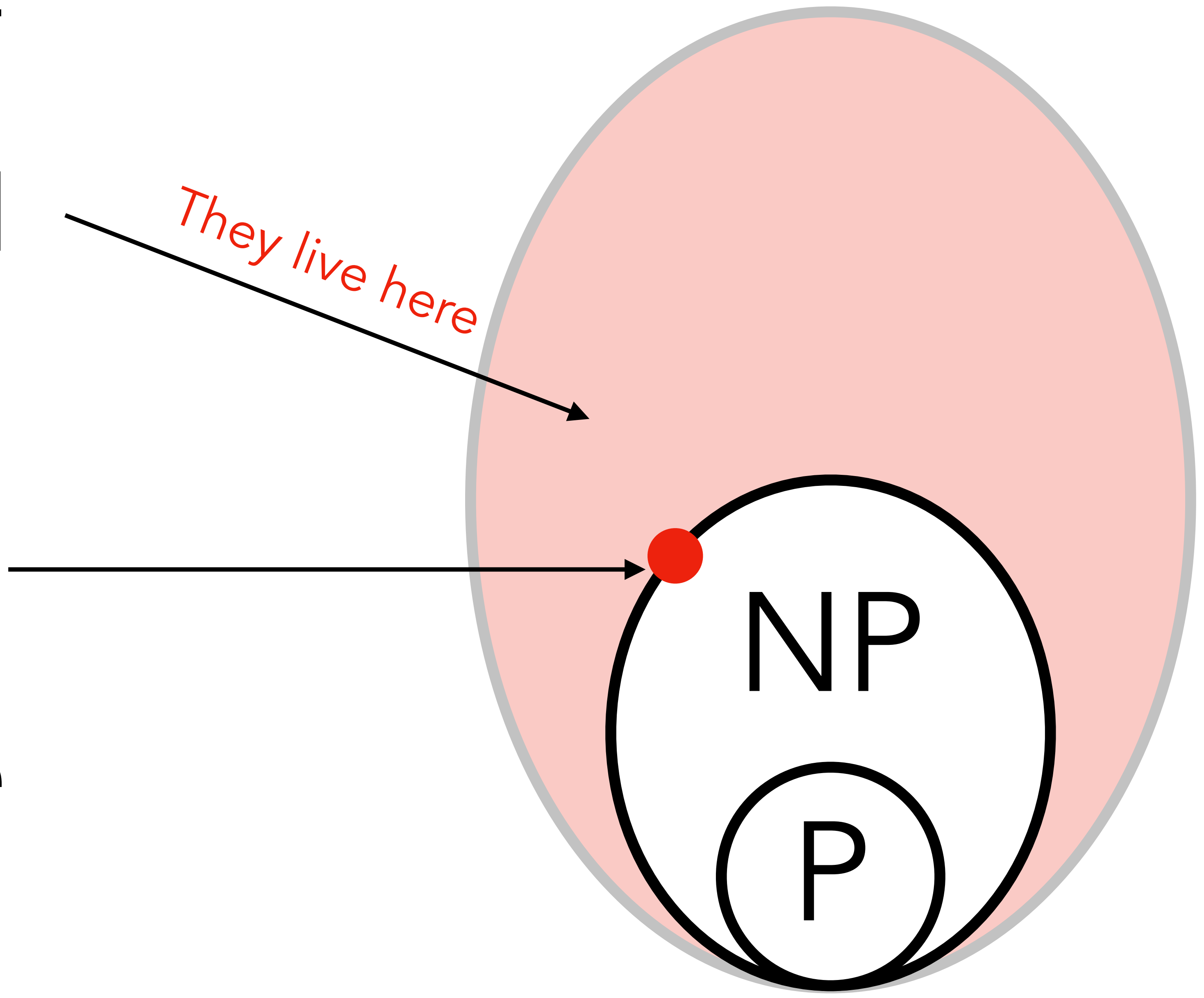
$L_{easy} \leq L$ if an efficient algorithm for solving L gives an efficient algorithm for solving L_{easy}

The type of reductions depends on the complexity classes, e.g. reductions could be randomized, quantum, space-limited

E.g. every problem in NP is easier than 3SAT

Such problems are called **NP-hard**

3SAT is also in **NP**



Such problems are called **NP-complete**

These are the hardest problems in NP

Reductions and Complete Problems

Reduction This formalizes that one problem is harder than another

$L_{easy} \leq L$ if an efficient algorithm for solving L gives an efficient algorithm for solving L_{easy}

The type of reductions depends on the complexity classes, e.g. reductions could be randomized, quantum, space-limited

E.g. every problem in NP is easier than 3SAT

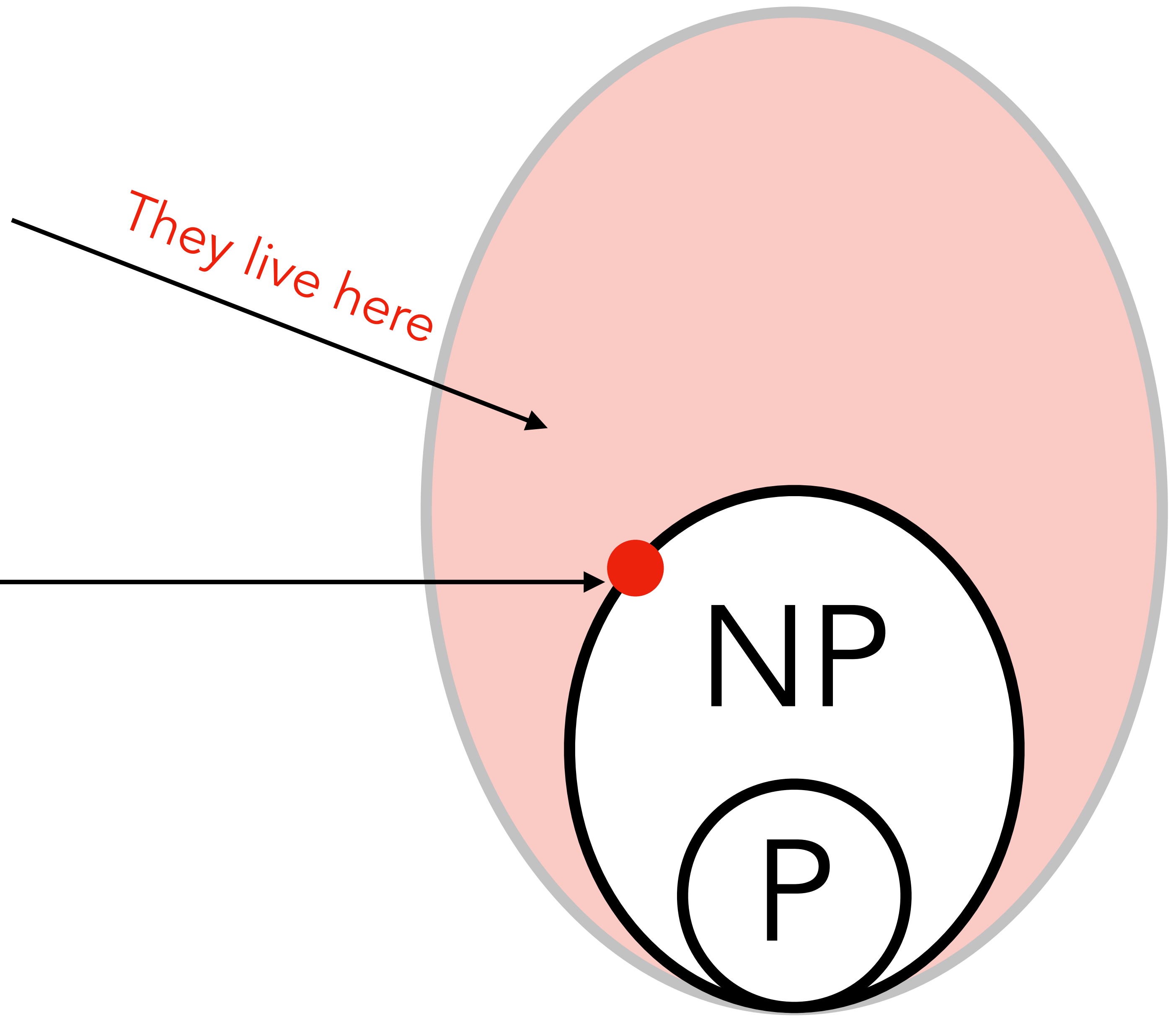
Such problems are called **NP-hard**

3SAT is also in **NP**

Such problems are called **NP-complete**

These are the hardest problems in NP

Some complexity classes are not known to have complete problems



More to come

- Oracles and Diagonalization
- Boolean Circuits
- Polynomial Hierarchy
- PCP Theorem
- Supplementary Homework: read Chapters 1 and 2