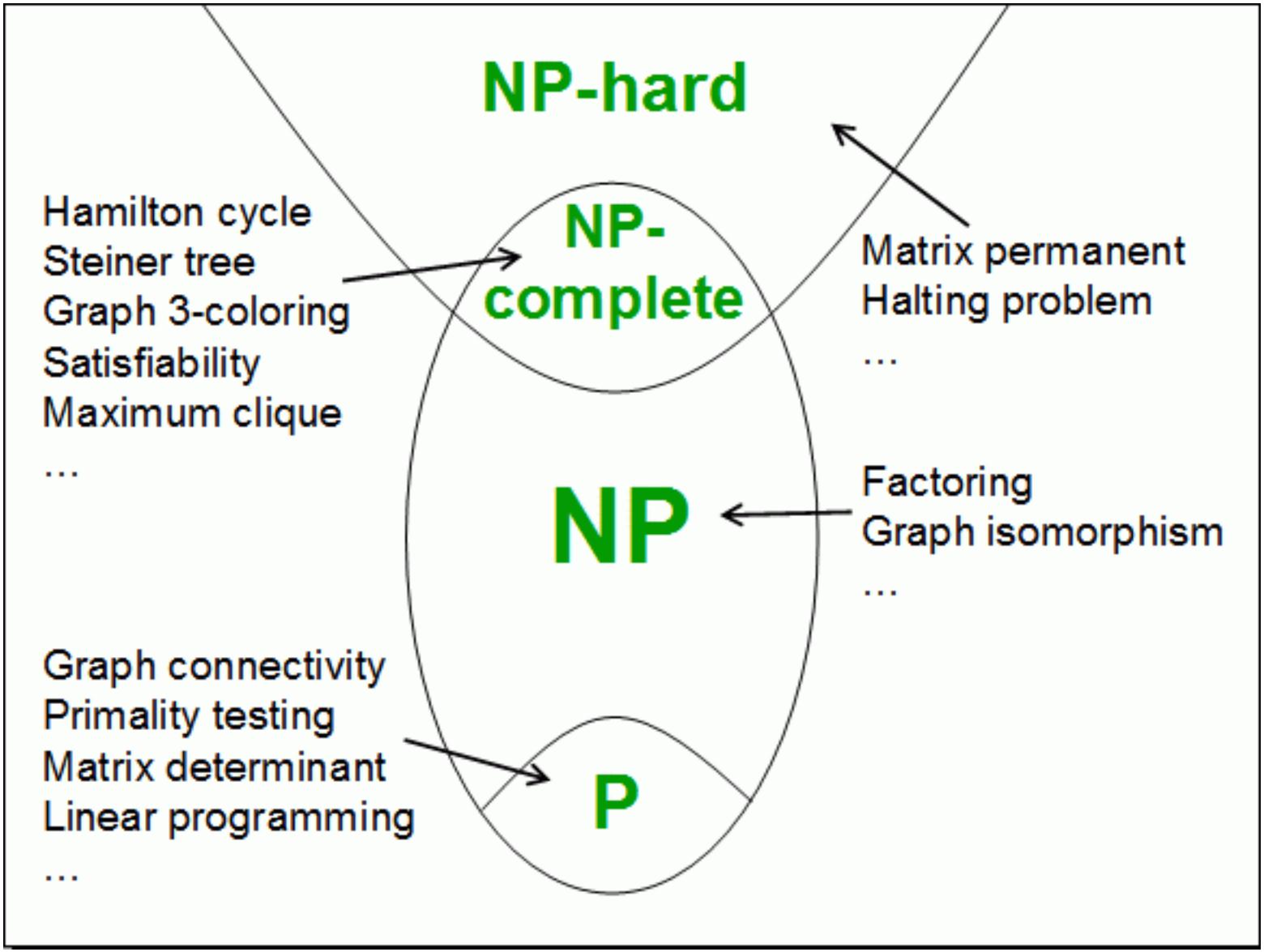
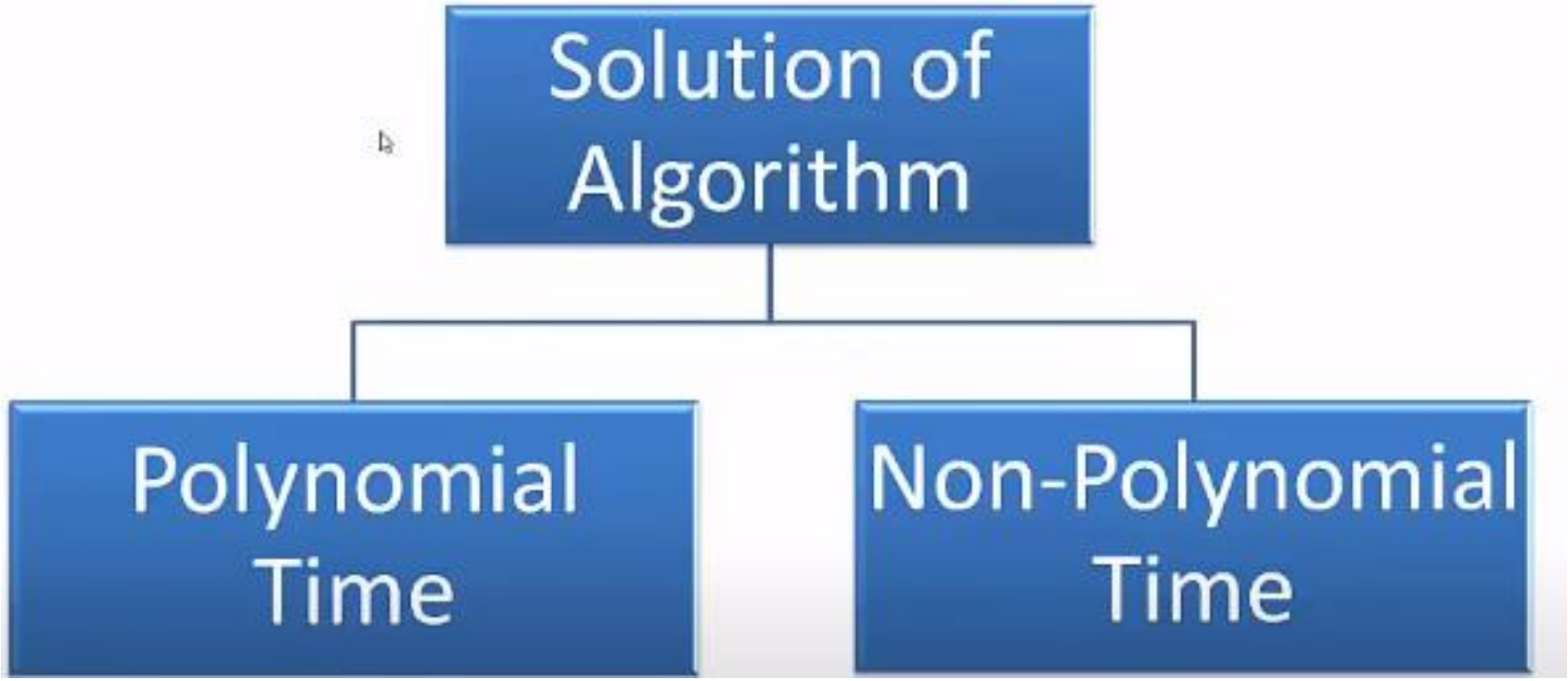
Design and Analysis of Algorithms

Computational Complexity
NP-Completeness

Slides from: Haidong Xue


NP-hard
NP-complete
NP
P
Hamilton cycle
Steiner tree
Graph 3-coloring
Satisfiability
Maximum clique
…
Matrix permanent
Halting problem
…
Factoring
Graph isomorphism
…
Graph connectivity
Primality testing
Matrix determinant
Linear programming
…


Solution of Algorithm
Polynomial Time
Non-Polynomial Time

What is polynomial-time?

- Polynomial-time: running time is $O(n^k)$, where k is a constant.
- Are they polynomial-time running time?
  - $T(n) = 3$
    - yes
  - $T(n) = n$
    - yes
  - $T(n) = nlg(n)$
    - yes
  - $T(n) = n^3$
    - yes

What is polynomial-time?

- Are the polynomial-time?
  - $T(n) = 5^n$
    - No
  - $T(n) = n!$
    - No
- Problems with polynomial-time algorithms are considered as tractable
- With polynomial-time, we can define **P** problems, and **NP** problems

What are P and NP?

- P problems
  - (The original definition) Problems that can be solved by **deterministic Turing machine** in polynomial-time.
  - (A equivalent definition) Problems that are solvable in polynomial time.
- NP problems
  - (The original definition) Problems that can be solved by **non-deterministic Turing machine** in polynomial-time.
  - (A equivalent definition) Problems that are **verifiable** in polynomial time.
    - Given a solution, there is a polynomial-time algorithm to tell if this solution is correct.

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

What are P and NP?

- Polynomial-time verification can be used to easily tell if a problem is a NP problem
- E.g.:
  - Sorting, n-integers
    - A candidate: an array
    - Verification: scan it once
  - Max-heapify, n-nodes:
    - A candidate: a complete binary search tree
    - Verification: scan all the nodes once
  - Find all the sub sets of a given set A, |A|=n
    - A candidate: a set of set
    - Verification: check each set

What are P and NP?

- Based on the definition of P and NP, which statements are correct?
  - "NP means non-polynomial"
    - No!
  - $\mathrm{P} \cap \mathrm{NP} = \emptyset$
    - No. $\mathrm{P} \subseteq \mathrm{NP}$
  - A P problem is also a NP problem
    - Yes. $\mathrm{P} \subseteq \mathrm{NP}$

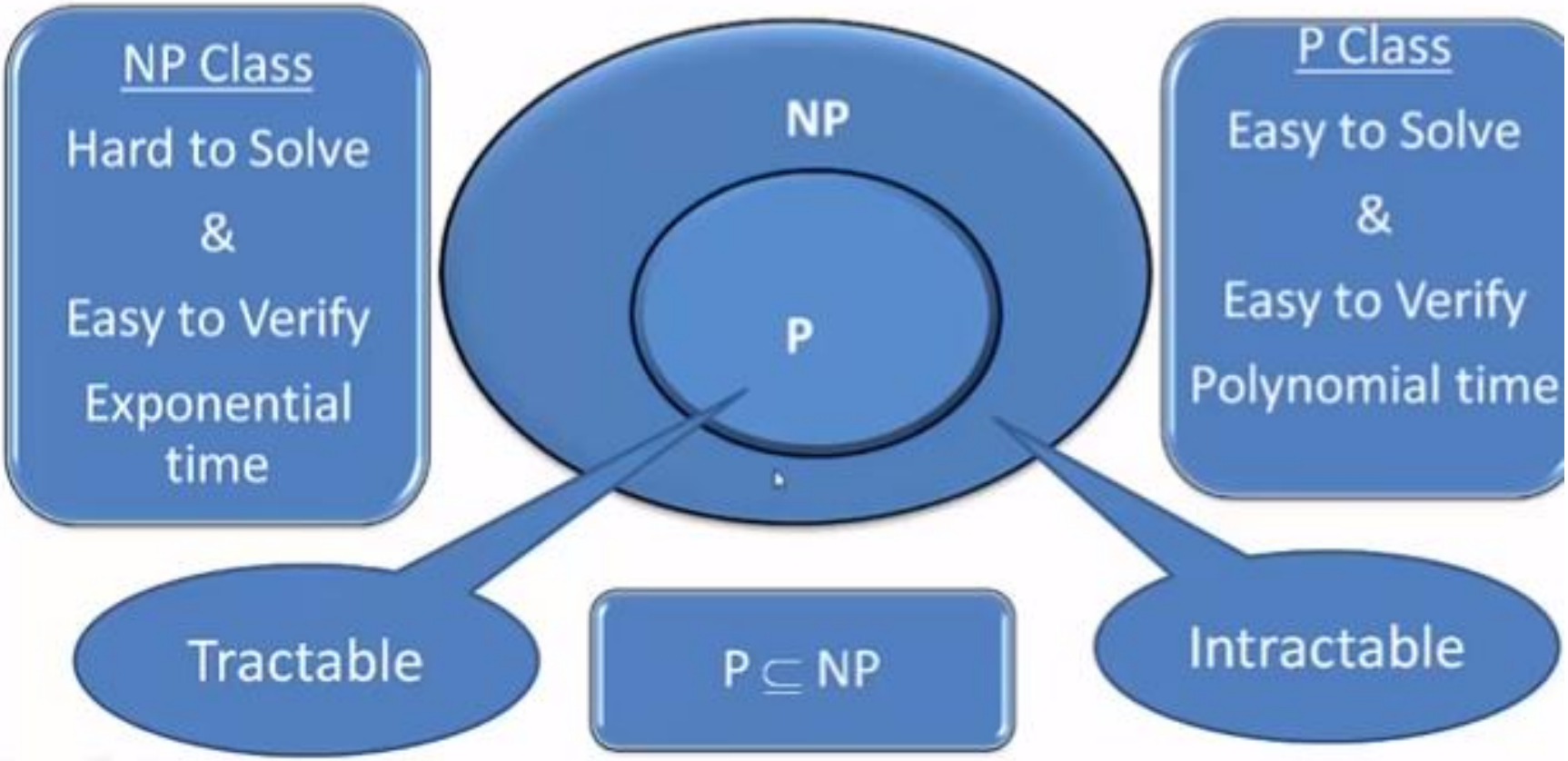
What are P and NP?

- any problem solvable by a deterministic Turing machine in polynomial time is also solvable by a nondeterministic Turing machine in polynomial time. Thus, **P** $\subseteq$ **NP**

P = NP means whether an NP problem can belong to class P problem. In other words whether every problem whose solution can be verified by a computer in polynomial time can also be solved by a computer in polynomial time

The NP Class Problems, it is verified in polynomial time.

The P Class Problems, not only it is solved on polynomial time but it is verified also in polynomial time.



Is P = NP ?

If you can prove P = NP Then

Information security or online security is vulnerable to attack,

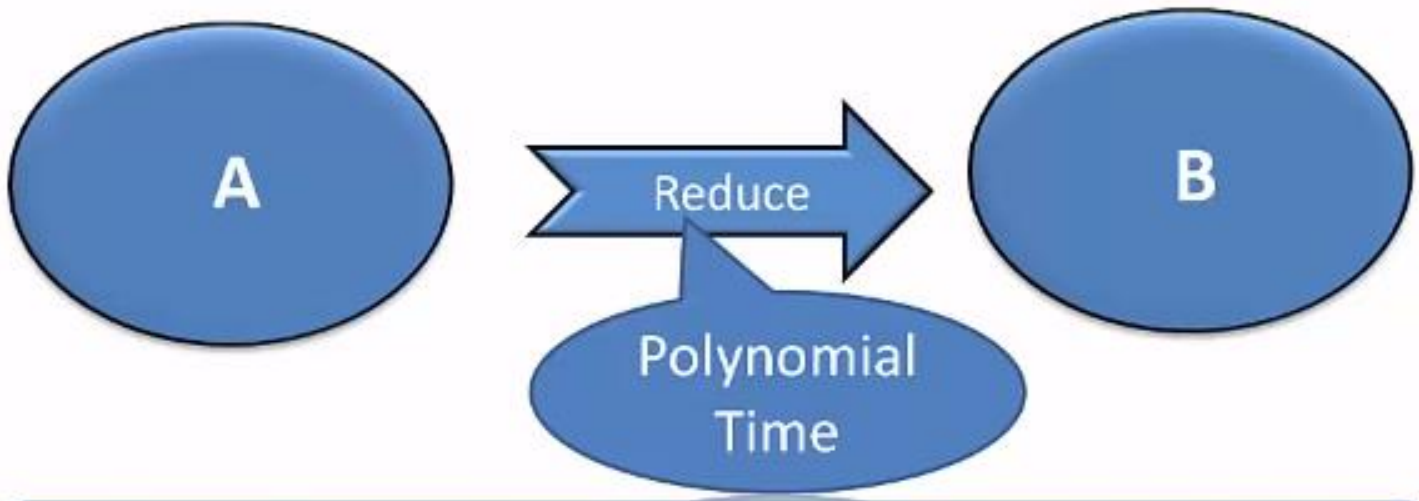
Everything become more efficient such as

Transportation, Scheduling, understanding DNA etc.

If you can prove P $\neq$ NP Then

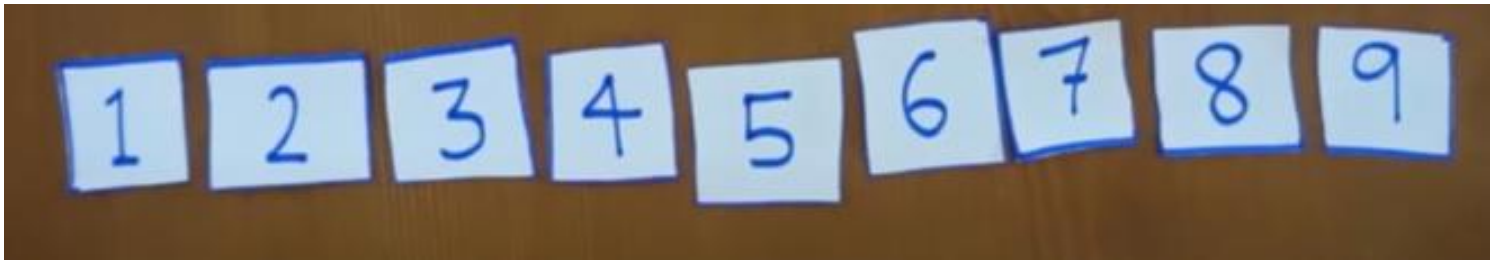
You can prove that there are some problems that can never be solved.

Reduction:

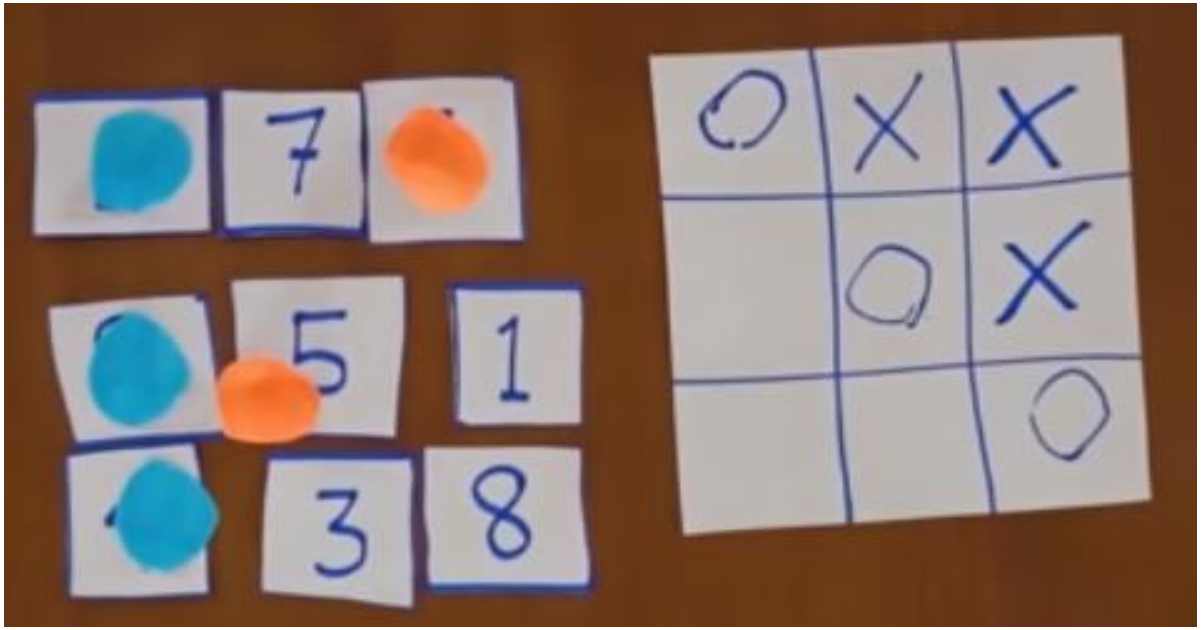


Let A and B are two problems then problem A reduces to problem B iff there is a way to solve A by deterministic algorithm that solve B in polynomial time.

If A is reducible to B we denote it by $A \propto B$


1 2 3 4 5 6 7 8 9


2 7 6
9 5 1
4 3 8


7
5 1
3 8

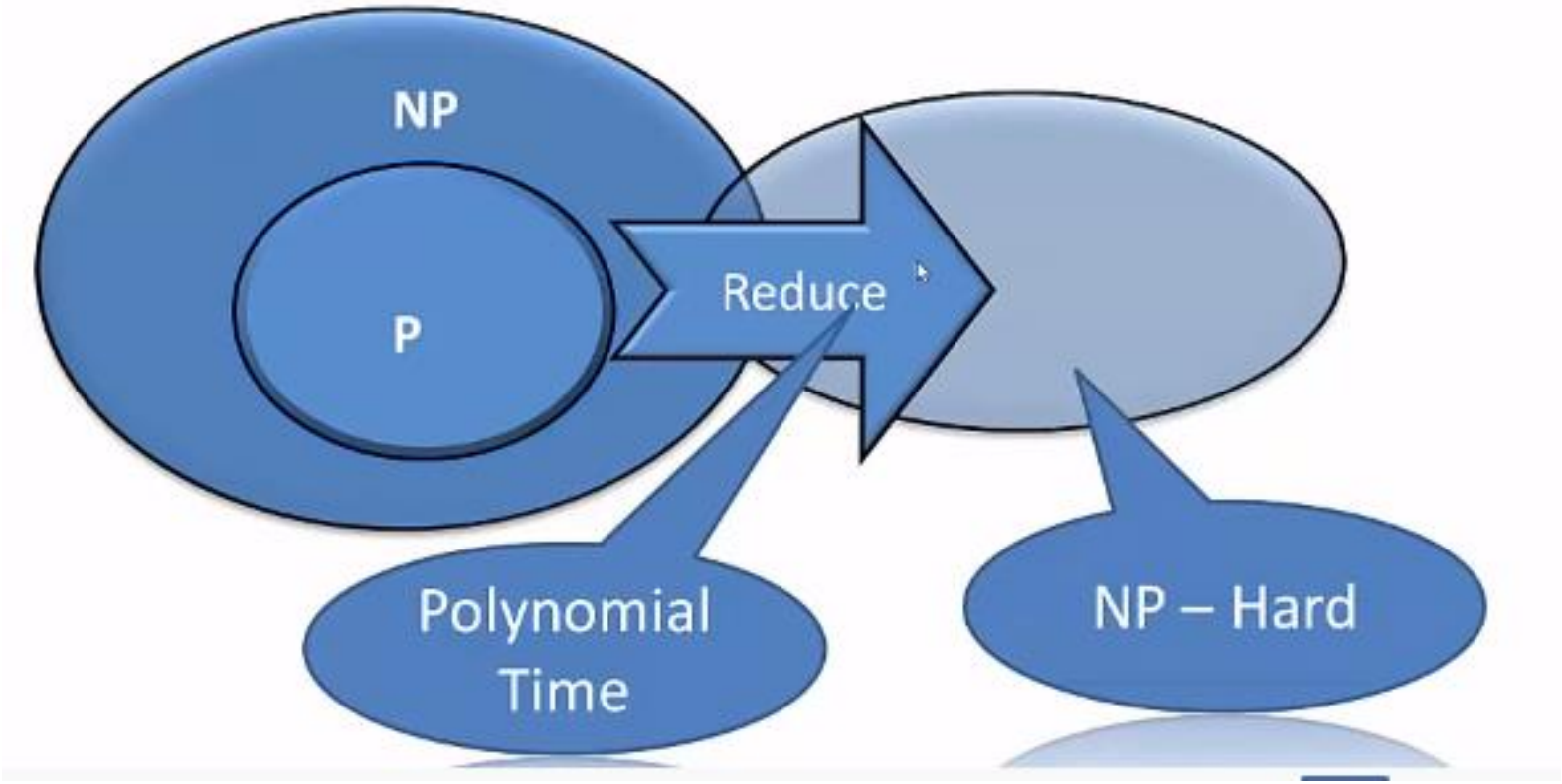
Reduction:

Properties:

1. if A is reducible to B and B in P then A in P.
2. A is not in P implies B is not in P

NP Hard Problem:

A Problem is NP-Hard if every problem in NP can be polynomial reduced to it.

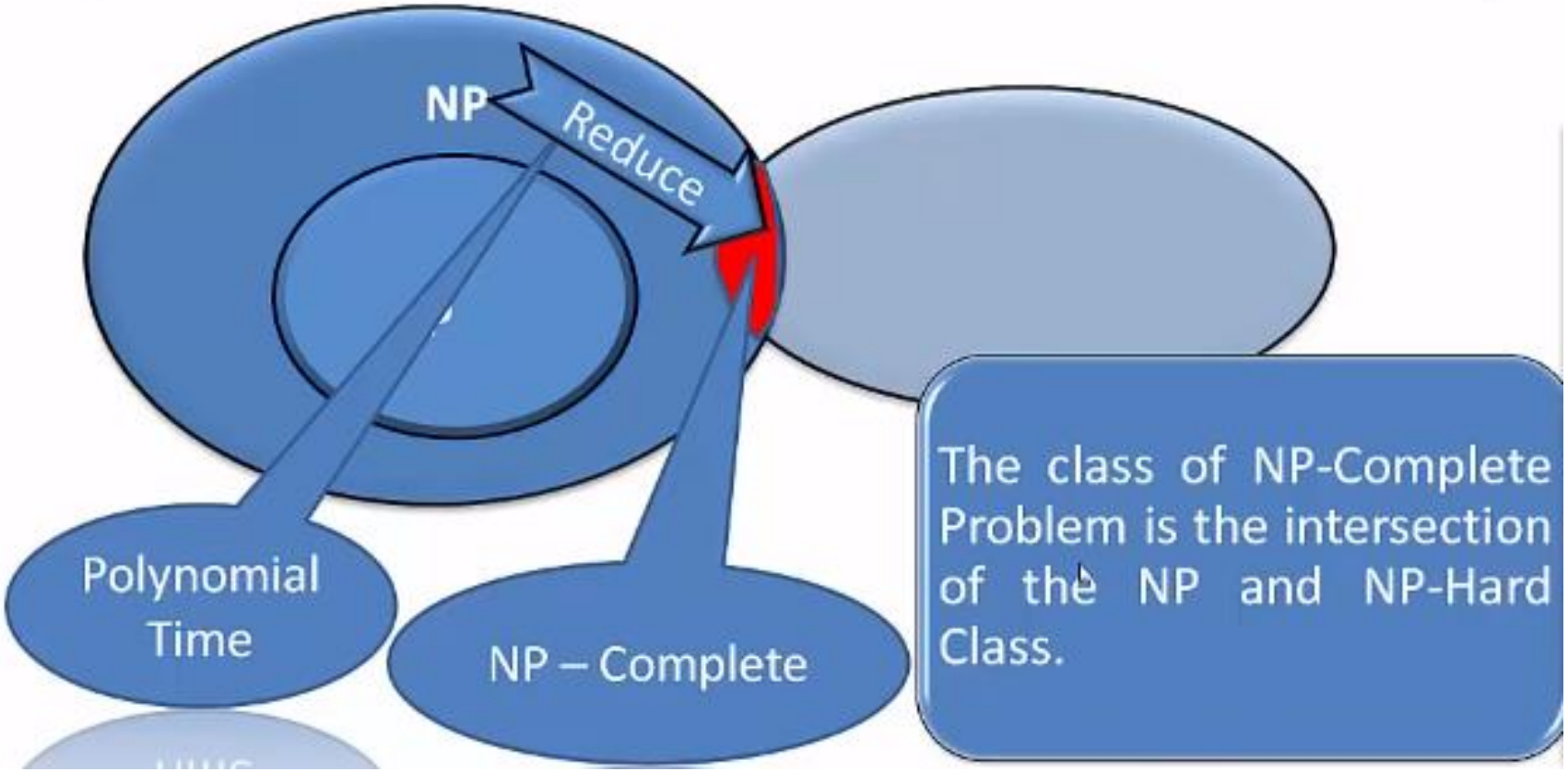


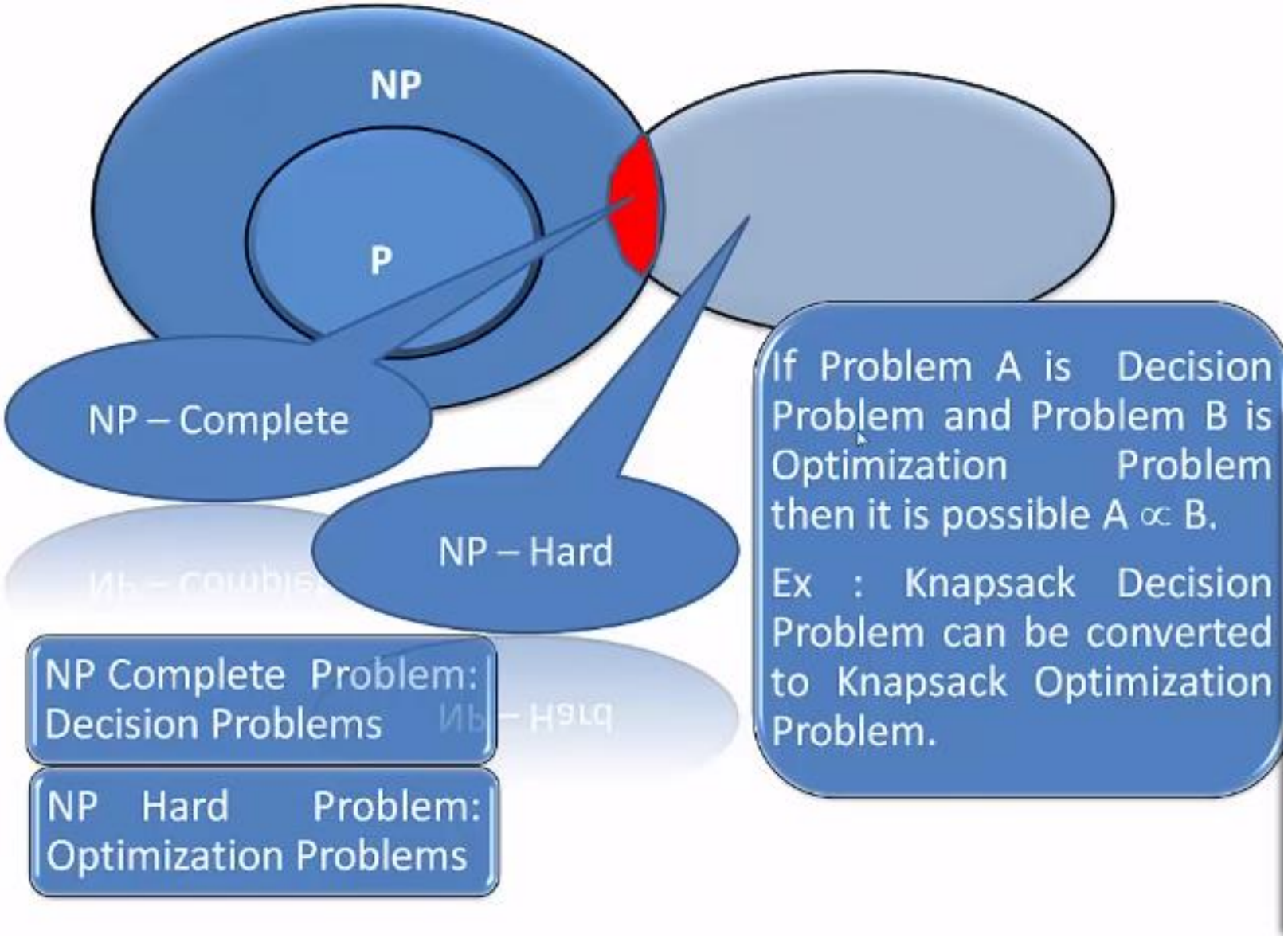
What are NP-complete problems?

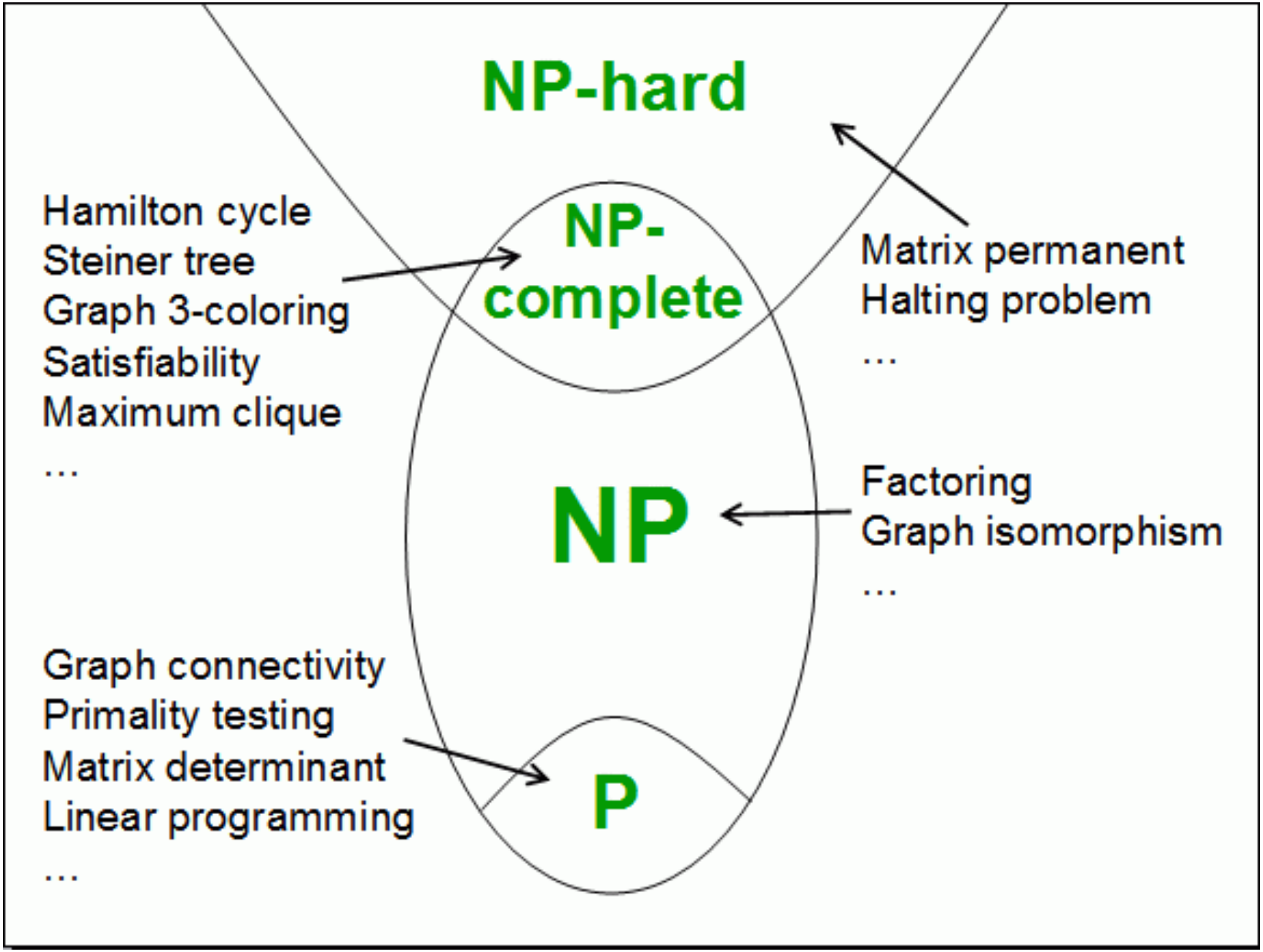
- A NP-complete problem(NPC) is
  - a NP problem
  - harder than all equal to all NP problems
- In other words, NPC problems are the hardest NP problems
- **So far**, no polynomial time algorithms are found for any of NPC problems

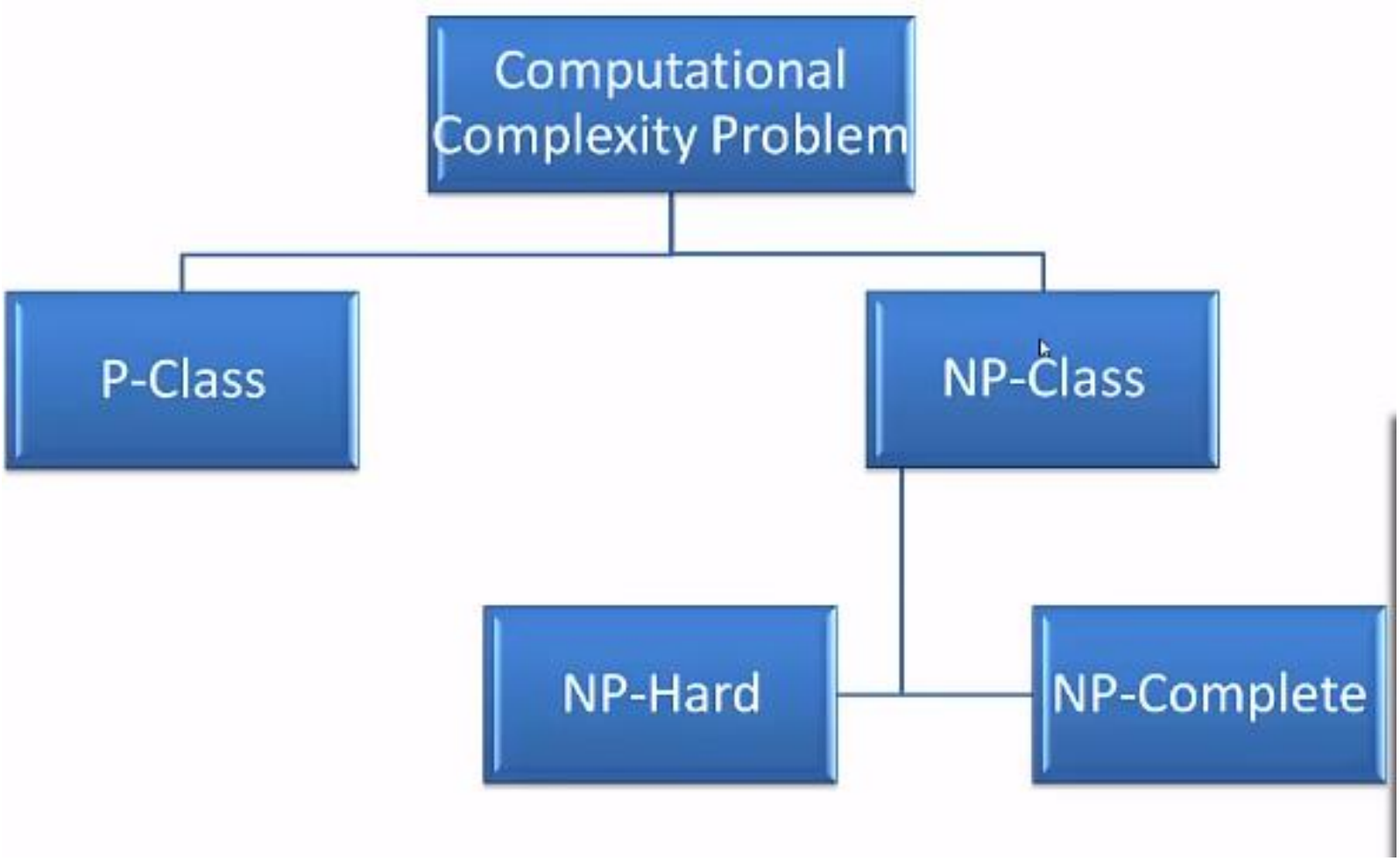
NP Complete Problem:

A Problem is NP-Complete if it is in NP and it is NP-Hard.



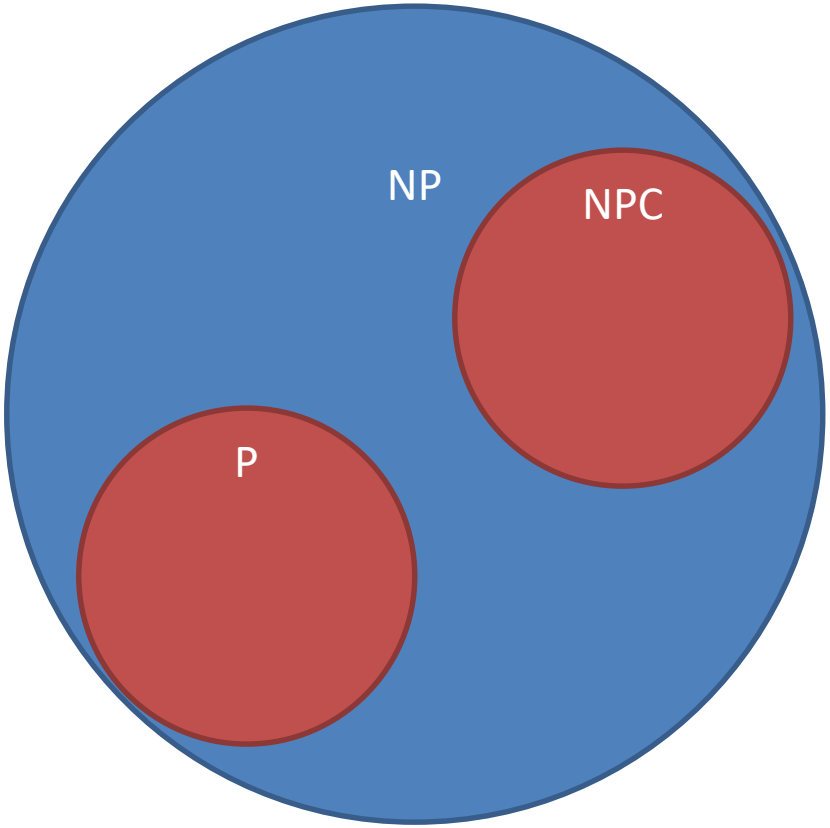

NP
P
NP – Complete
NP – Hard
If Problem A is Decision Problem and Problem B is Optimization Problem then it is possible A ∝ B.
Ex : Knapsack Decision Problem can be converted to Knapsack Optimization Problem.
NP Complete Problem: Decision Problems
NP Hard Problem: Optimization Problems


NP-hard
NP-complete
NP
P
Hamilton cycle
Steiner tree
Graph 3-coloring
Satisfiability
Maximum clique
…
Matrix permanent
Halting problem
…
Factoring
Graph isomorphism
…
Graph connectivity
Primality testing
Matrix determinant
Linear programming
…

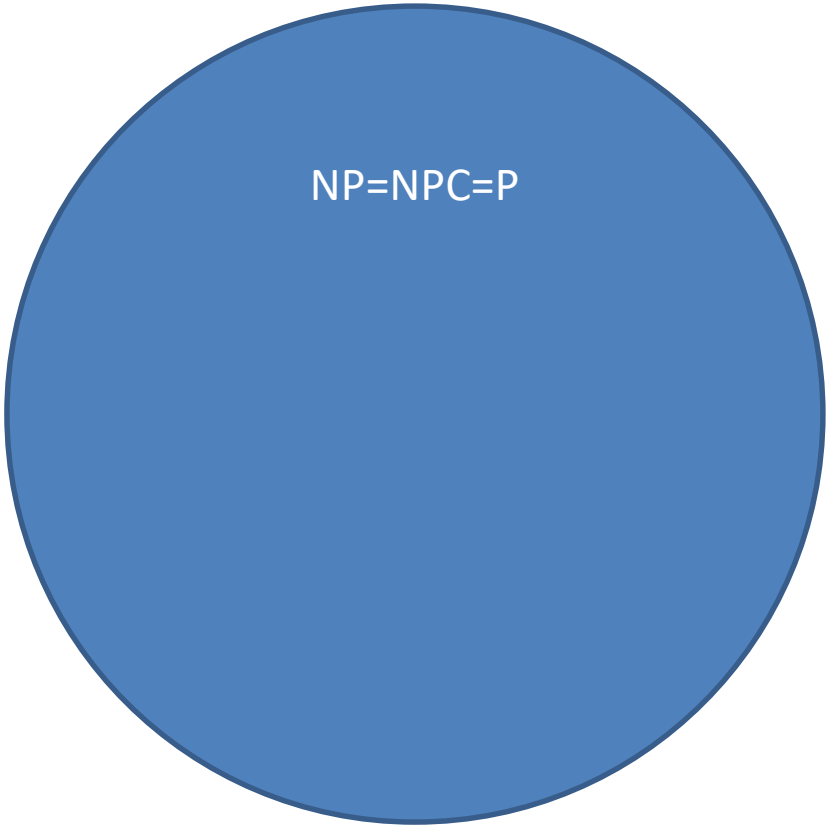

Computational Complexity Problem
P-Class
NP-Class
NP-Hard
NP-Complete

What are NP-complete problems?

Most scientists believe:



However, it is not ruled out that:



NP-Hard problems: problems harder than or equal to NPC problems

Why we study NPC?

- One of the most important reasons is:
  - If you see a problem is NPC, you can stop from spending time and energy to develop a fast polynomial-time algorithm to solve it.
- Just tell your boss it is a NPC problem
- How to prove a problem is a NPC problem?

How to prove a problem is a NPC problem?

- A common method is to prove that it is not easier than a known NPC problem.
- To prove problem A is a NPC problem
  - Choose a NPC problem B
  - Develop a **polynomial-time algorithm translate A to B**
- A **reduction algorithm**
- If A can be solved in polynomial time, then B can be solved in polynomial time. It is contradicted with B is a NPC problem.
- So, A cannot be solved in polynomial time, it is also a NPC problem.

What if a NPC problem needs to be solved?

- Buy a more expensive machine and wait
  - (could be 1000 years)
- Turn to approximation algorithms
  - Algorithms that produce near optimal solutions