CSE 332 Autumn 2024
Lecture 28: NP-Completeness

Nathan Brunelle, Chandni Rajasekaran

http://www.cs.uw.edu/332

Tractability

- Tractable:
  - Feasible to solve in the "real world"
- Intractable:
  - Infeasible to solve in the "real world"
- Whether a problem is considered "tractable" or "intractable" depends on the use case
  - For machine learning, big data, etc. tractable might mean $\mathrm{O}(n)$ or even $O(\log n)$
  - For most applications it's more like $O(n^3)$ or $O(n^2)$
- A strange pattern:
  - Most "natural" problems are either done in small-degree polynomial (e.g. n^2) or else exponential time (e.g. 2^n)
  - It's rare to have problems which require a running time of n^5, for example

Complexity Classes

- A Complexity Class is a set of problems (e.g. sorting, Euler path, Hamiltonian path)
  - The problems included in a complexity class are those whose most efficient algorithm has a specific upper bound on its running time (or memory use, or…)
- Examples:
  - The set of all problems that can be solved by an algorithm with running time $O(n)$
    - Contains: Finding the minimum of a list, finding the maximum of a list, buildheap, summing a list, etc.
  - The set of all problems that can be solved by an algorithm with running time $O(n^2)$
    - Contains: everything above as well as sorting, Euler path
  - The set of all problems that can be solved by an algorithm with running time $O(n!)$
    - Contains: everything we've seen in this class so far

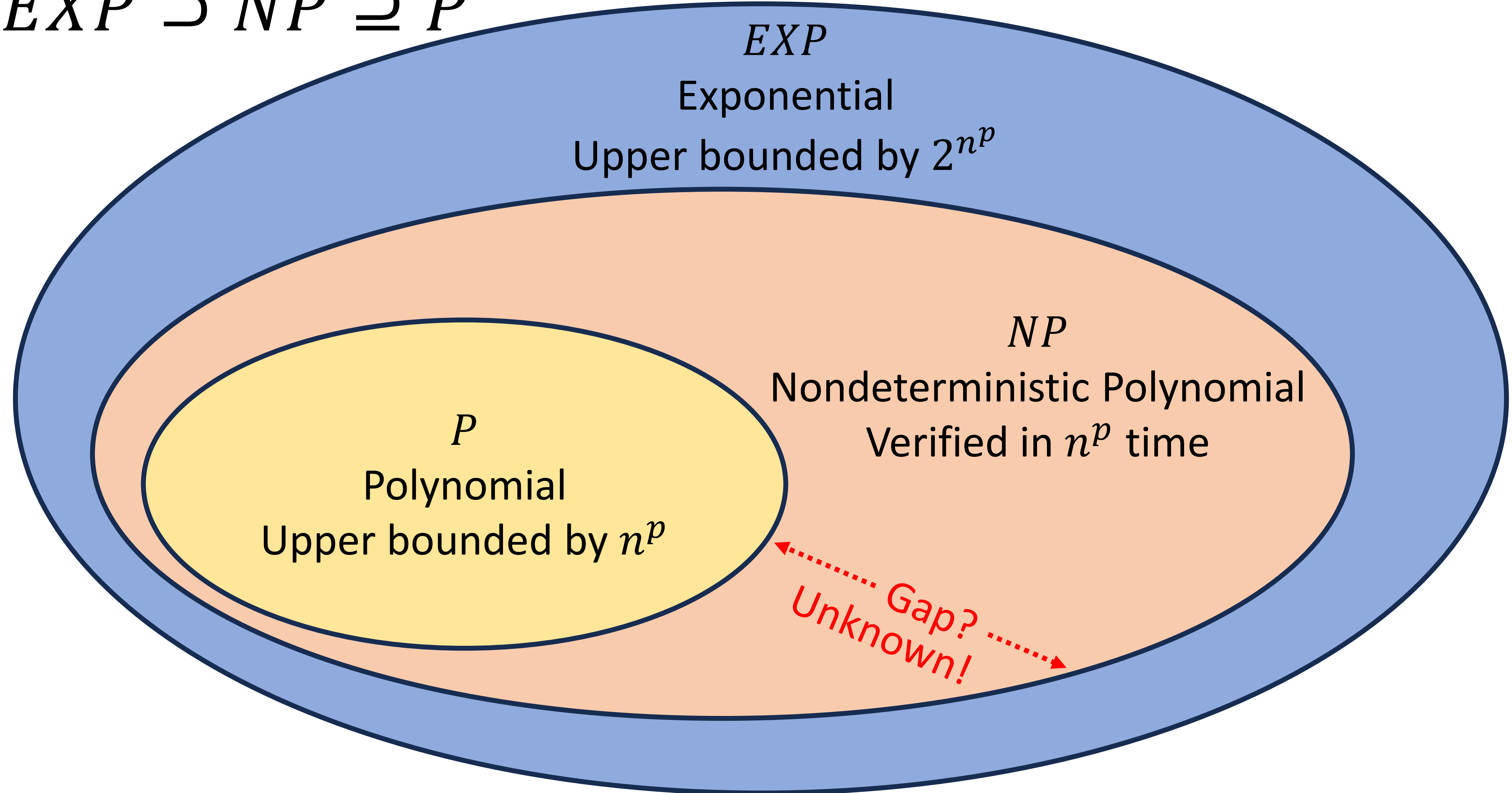
Complexity Classes and Tractability

- To explore what problems are and are not tractable, we give some complexity classes special names:
- Complexity Class P:
  - Stands for "Polynomial"
  - The set of problems which have an algorithm whose running time is $O(n^p)$ for some choice of $p \in \mathbb{R}$.
  - We say all problems belonging to P are "Tractable"
- Complexity Class EXP:
  - Stands for "Exponential"
  - The set of problems which have an algorithm whose running time is $O\left(2^{n^p}\right)$ for some choice of $p \in \mathbb{R}$
  - We say all problems belonging to $EXP - P$ are "Intractable"
    - Disclaimer: Really it's all problems outside of P, and there are problems which do not belong to EXP, but we're not going to worry about those in this class

Some problems in EXP seem "easier"

- There are some problems that we do not have polynomial time algorithms to solve, but provided answers are easy to check
- Hamiltonian Path:
  - It's "hard" to look at a graph and determine whether it has a Hamiltonian Path
  - It's "easy" to look at a graph and a candidate path together and determine whether THAT path is a Hamiltonian Path
    - It's easy to **verify** whether a given path is a Hamiltonian path

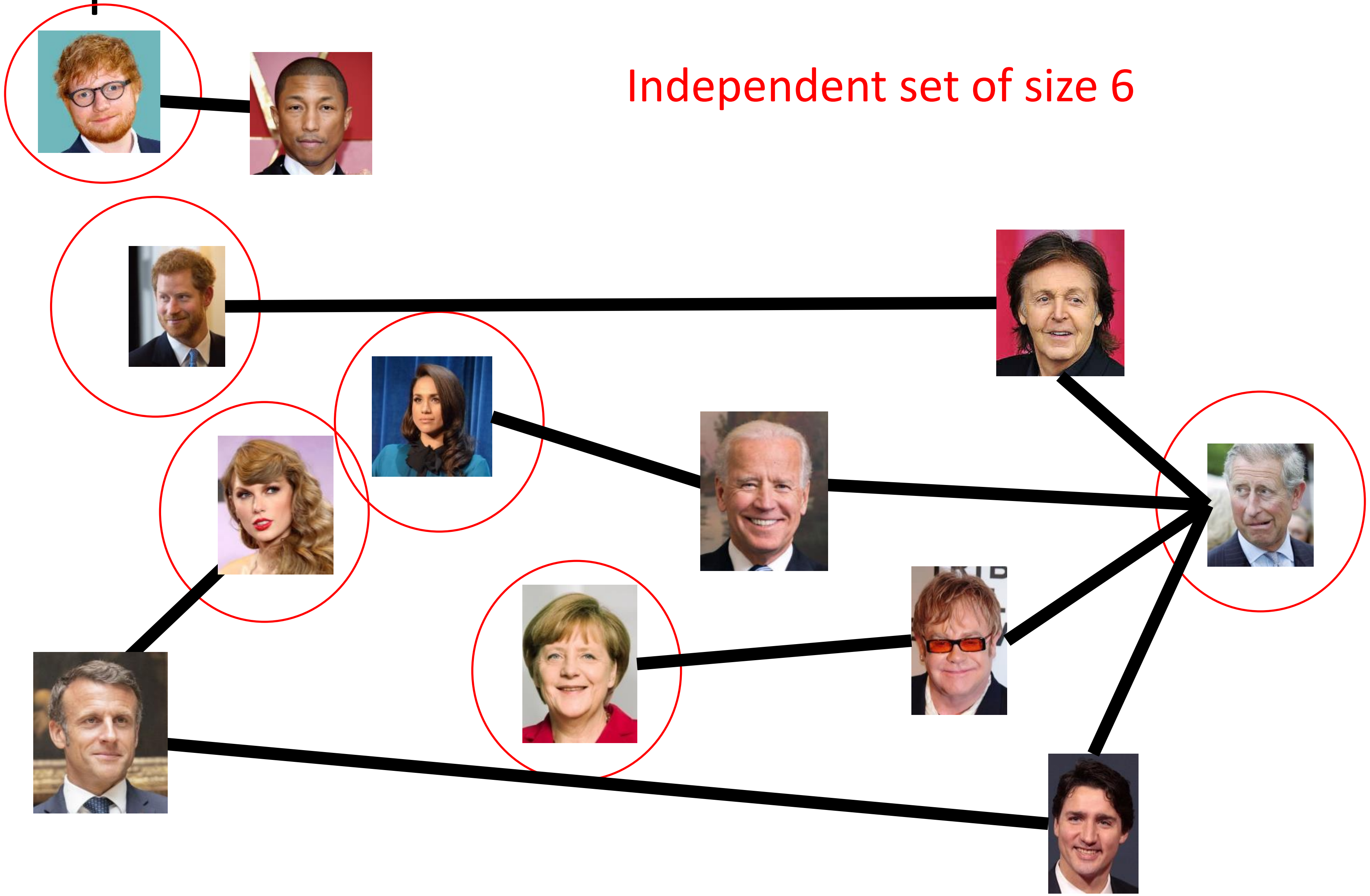
$EXP \supset NP \supseteq P$



Independent Set

- Independent set:
  - $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Independent Set Problem:
  - Given a graph $G = (V, E)$ and a number k, determine whether there is an independent set S of size k

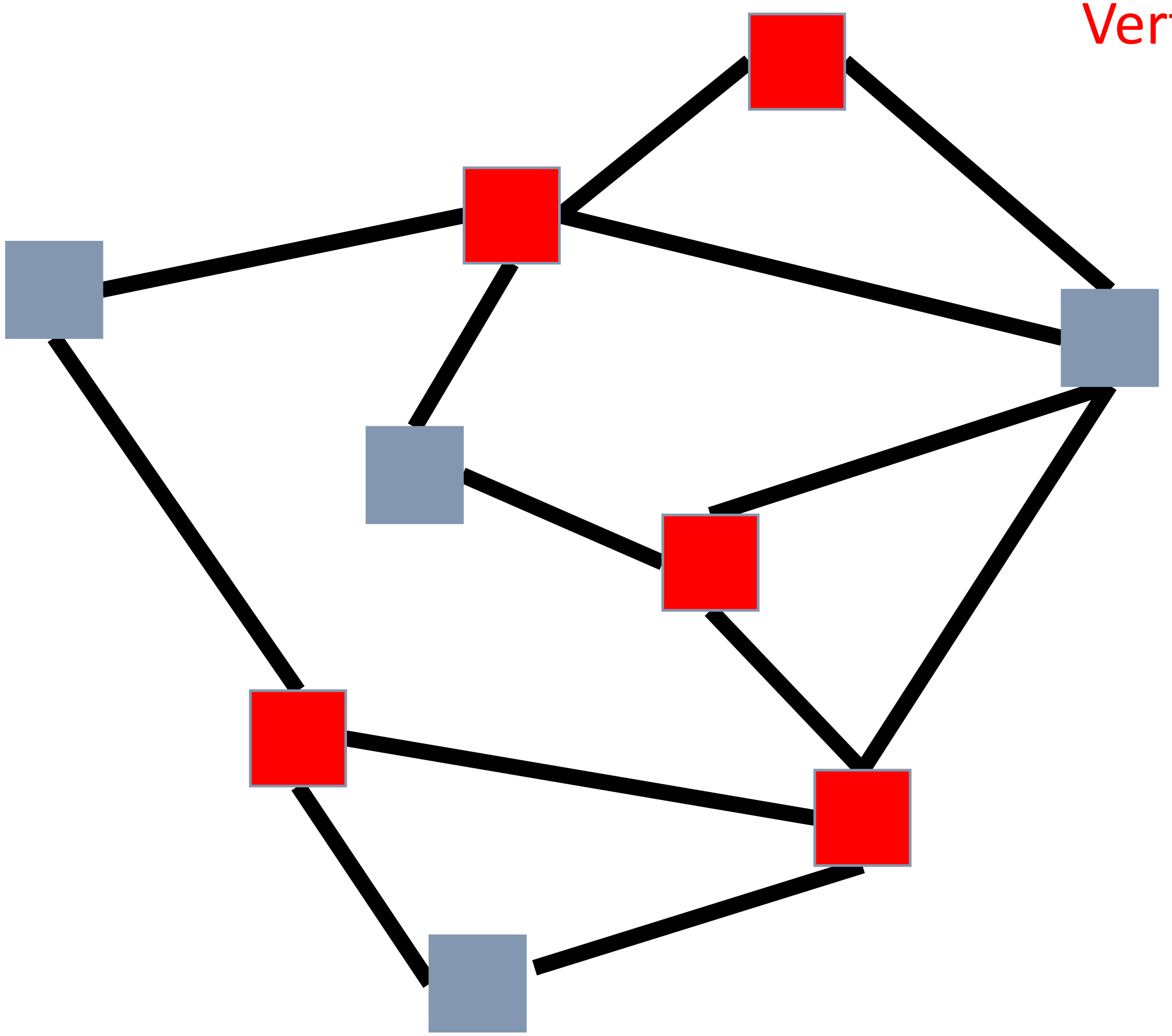
Example



Vertex Cover

- Vertex Cover:
  - $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Vertex Cover Problem:
  - Given a graph $G = (V, E)$ and a number k, determine if there is a vertex cover C of size k

Example

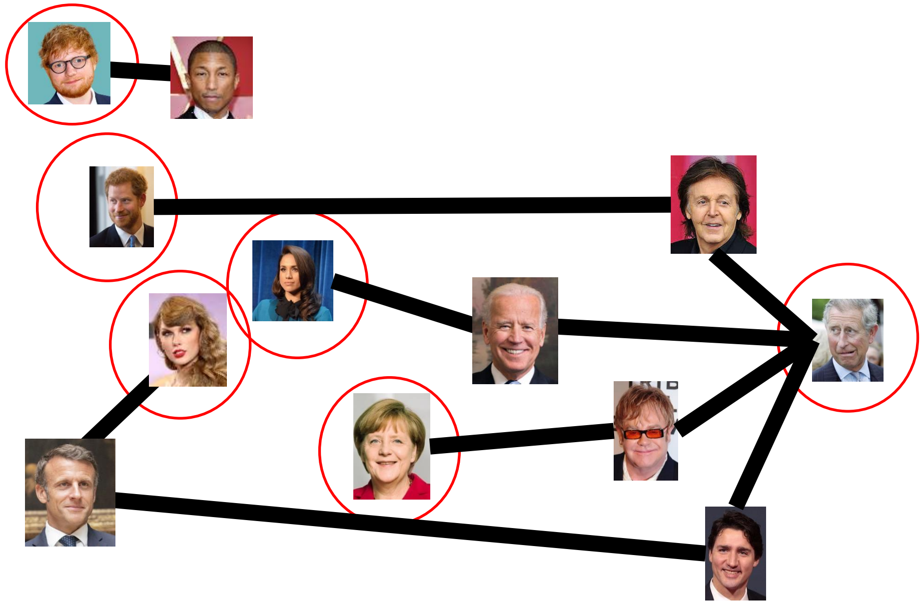


Vertex cover of size 5

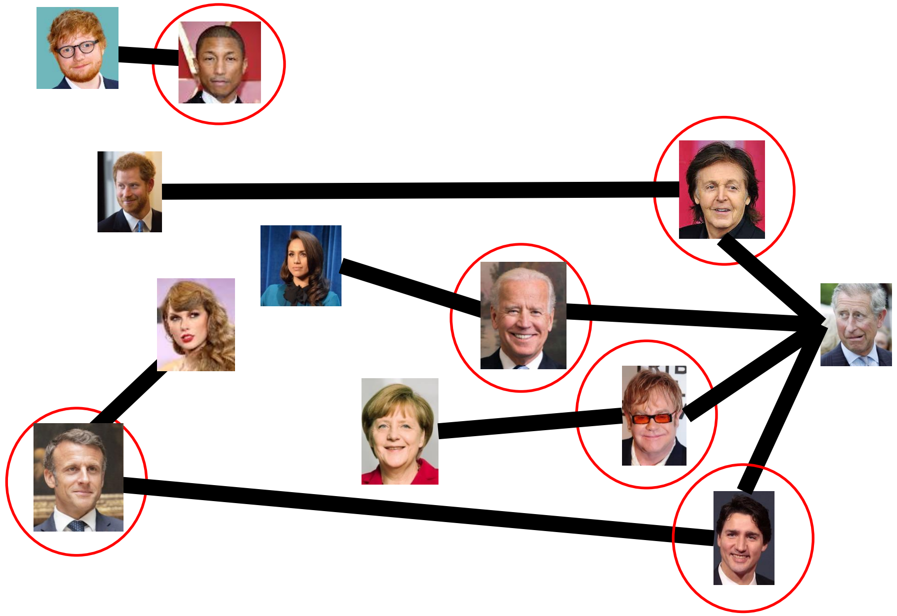
It’s easy to convert an Independent Set into a Vertex Cover!

S is an independent set of G iff $V - S$ is a vertex cover of G

Independent Set



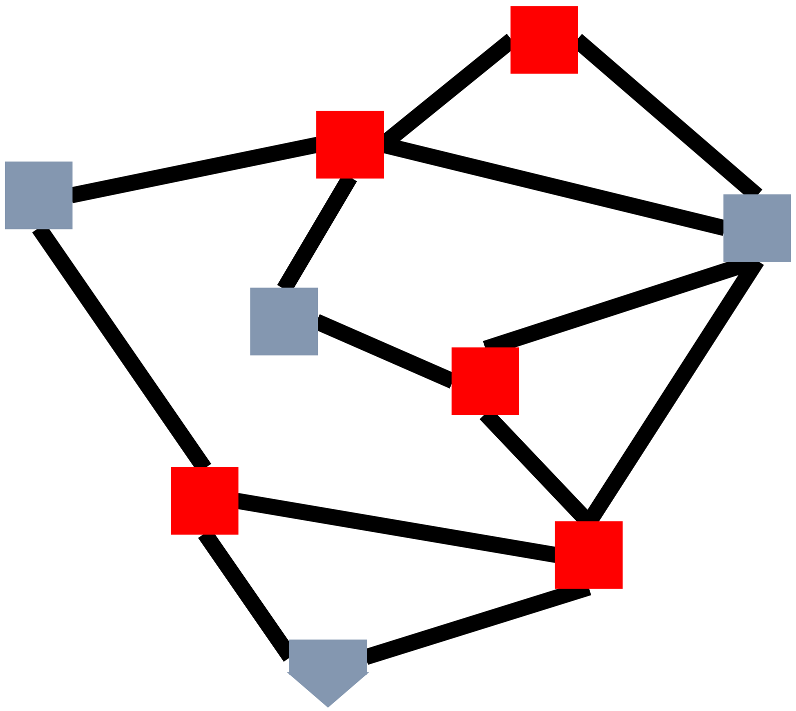
Vertex Cover



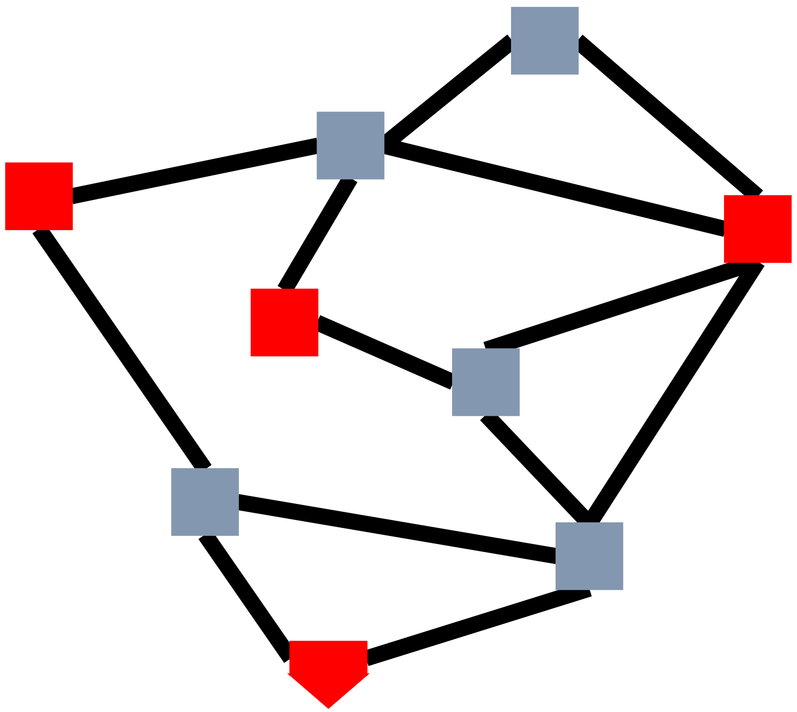
It’s easy to convert a Vertex Cover into an Independent Set!

S is an independent set of G iff $V - S$ is a vertex cover of G

Vertex Cover



Independent Set



Solving Vertex Cover and Independent Set

- Algorithm to solve vertex cover
  - Input: $G = (V, E)$ and a number k
  - Output: True if G has a vertex cover of size k
    - Check if there is an Independent Set of G of size $|V| - k$
- Algorithm to solve independent set
  - Input: $G = (V, E)$ and a number k
  - Output: True if G has an independent set of size k
    - Check if there is a Vertex Cover of G of size $|V| - k$

Either both problems belong to P, or else neither does!

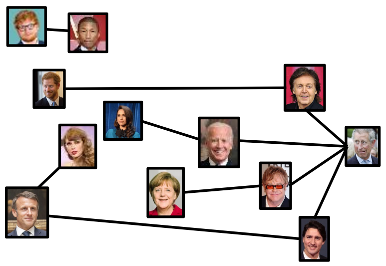
Reduction

- A strategy for creating algorithms
- Solve one problem by converting it into a different problem, then using an algorithm for that other problem.

Independent Set Reduces To Vertex Cover

Independent Set Input

k

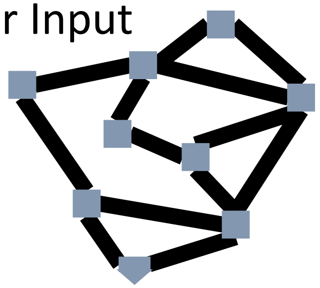


O(V) Time

Relate Independent Set input to Vertex Cover output

Use same graph
Set $k = |V| - k$

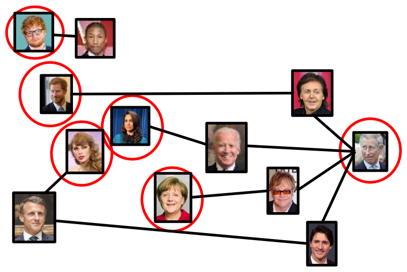
Vertex Cover Input



k

Any Algorithm for Vertex Cover

Independent Set Output

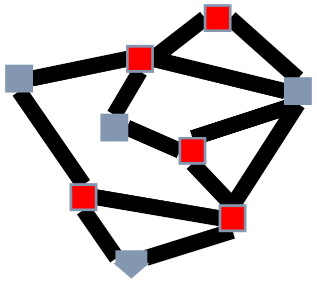


Relate Independent Set output to Vertex Cover output

Give same output

Reduction

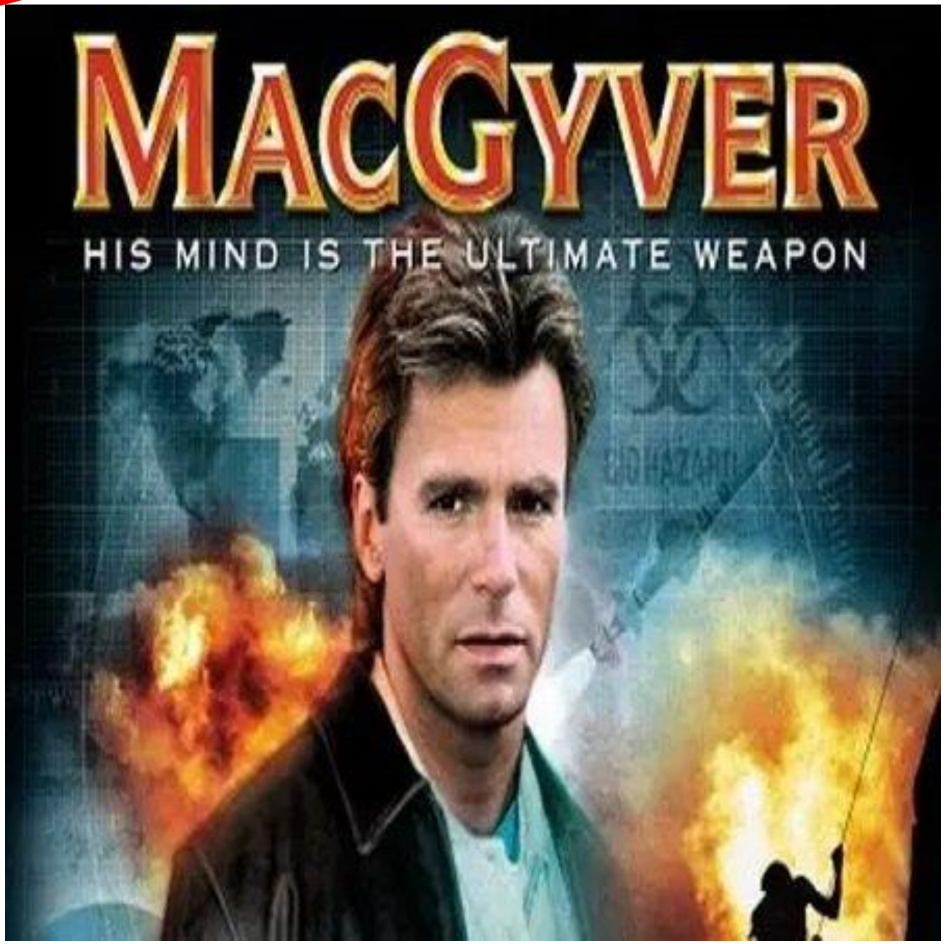
Vertex Cover Ouput



Reductions

Shows how two different problems relate to each other

MOVIE TIME!



MacGyver's Reduction

Problem we don't know how to solve

A

Opening a door

Solution for $\boldsymbol{A}$

Keg cannon battering ram

Aim duct at door, insert keg

Put fire under the Keg

Reduction

Problem we do know how to solve

B

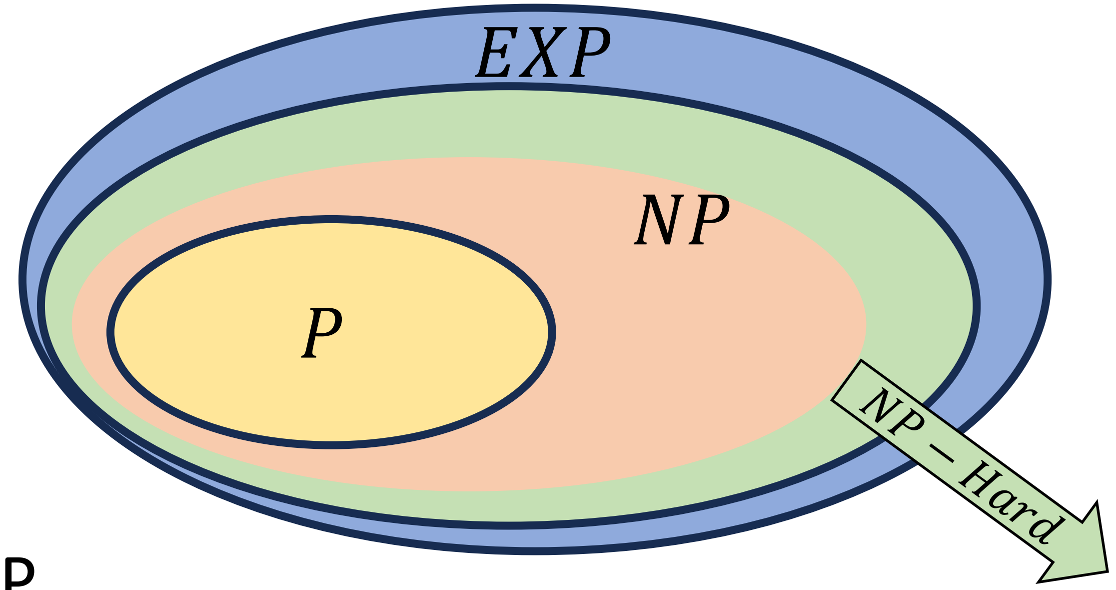
Lighting a fire

How?

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

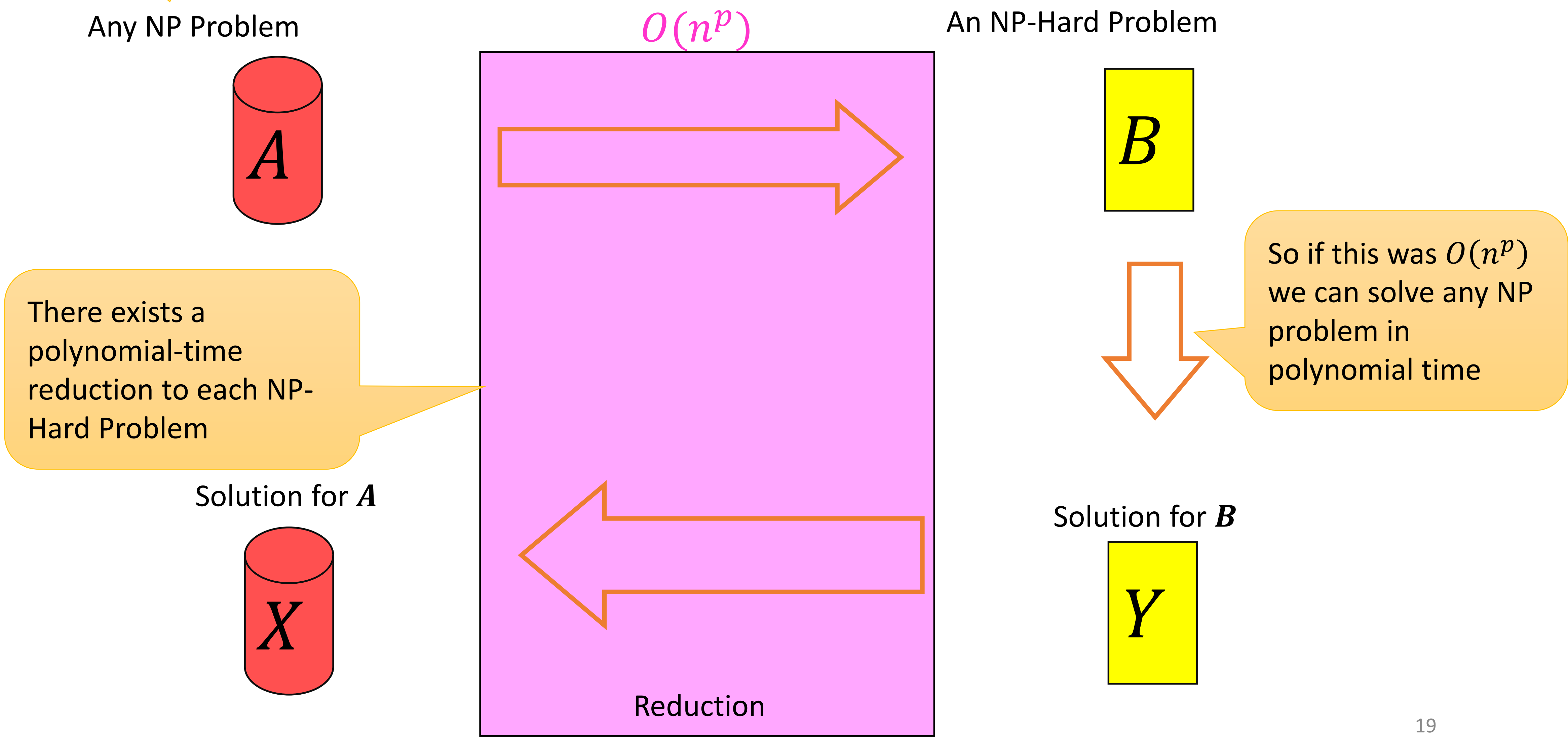
NP-Hard



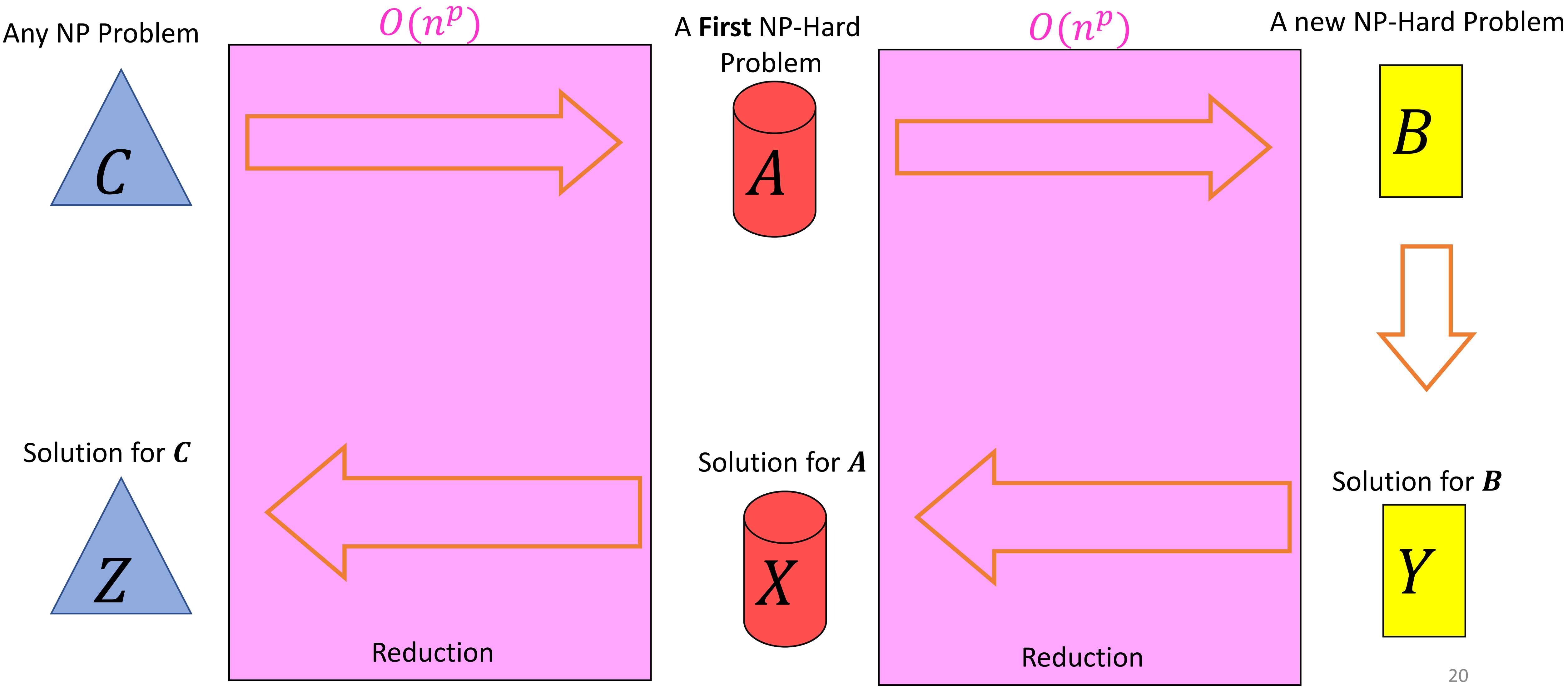
- How can we try to figure out if P=NP?
- Identify problems at least as “hard” as NP
  - If any of these “hard” problems can be solved in polynomial time, then all NP problems can be solved in polynomial time.
- Definition: NP-Hard:
  - B is NP-Hard provided EVERY problem within NP reduces to B in polynomial time

NP-Hard Idea

For every NP problem



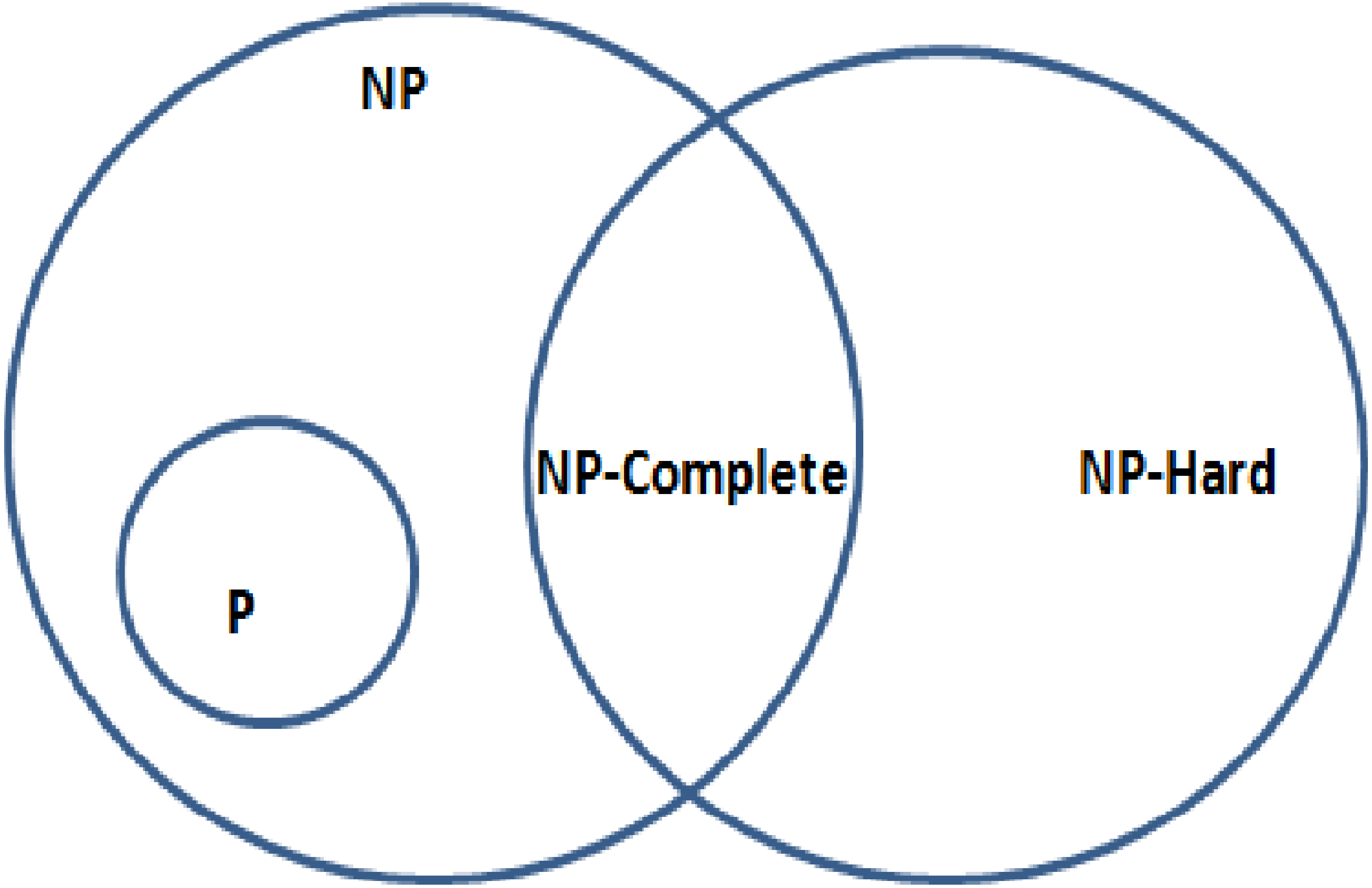
Showing NP-Hardness



NP-Complete

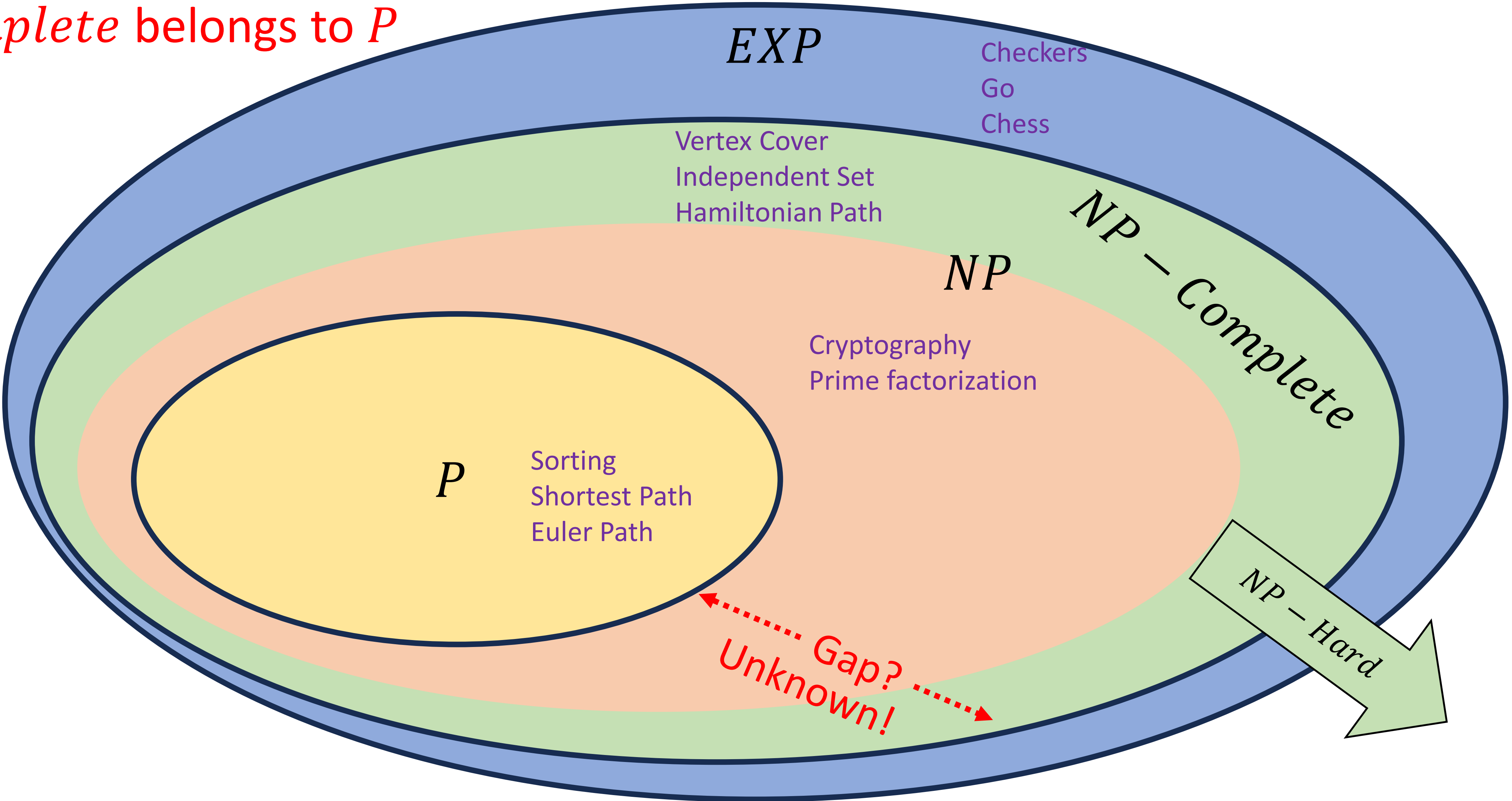
- A set of “together they stand, together they fall” problems
- The problems in this set either all belong to P, or none of them do
- Intuitively, the “hardest” problems in NP
- Collection of problems from NP that can all be “transformed” into each other in polynomial time
  - Like we could transform independent set to vertex cover, and vice-versa
  - We can also transform vertex cover into Hamiltonian path, and Hamiltonian path into independent set, and …

Another Representation

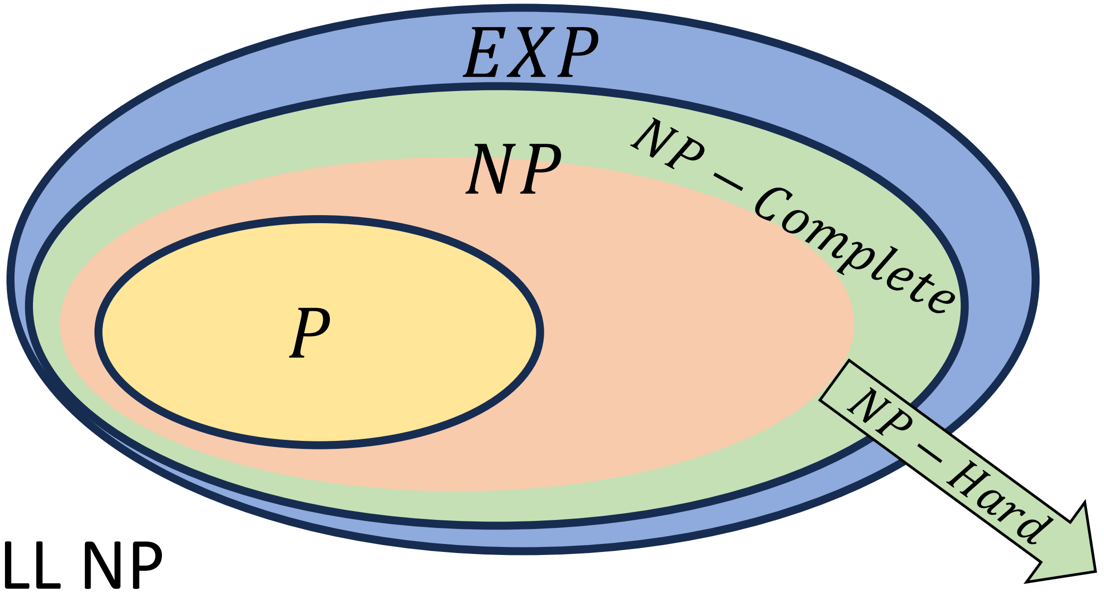


$EXP \supset NP-Complete \supseteq NP \supseteq P$

$P = NP$ iff some problem from $NP-Complete$ belongs to P

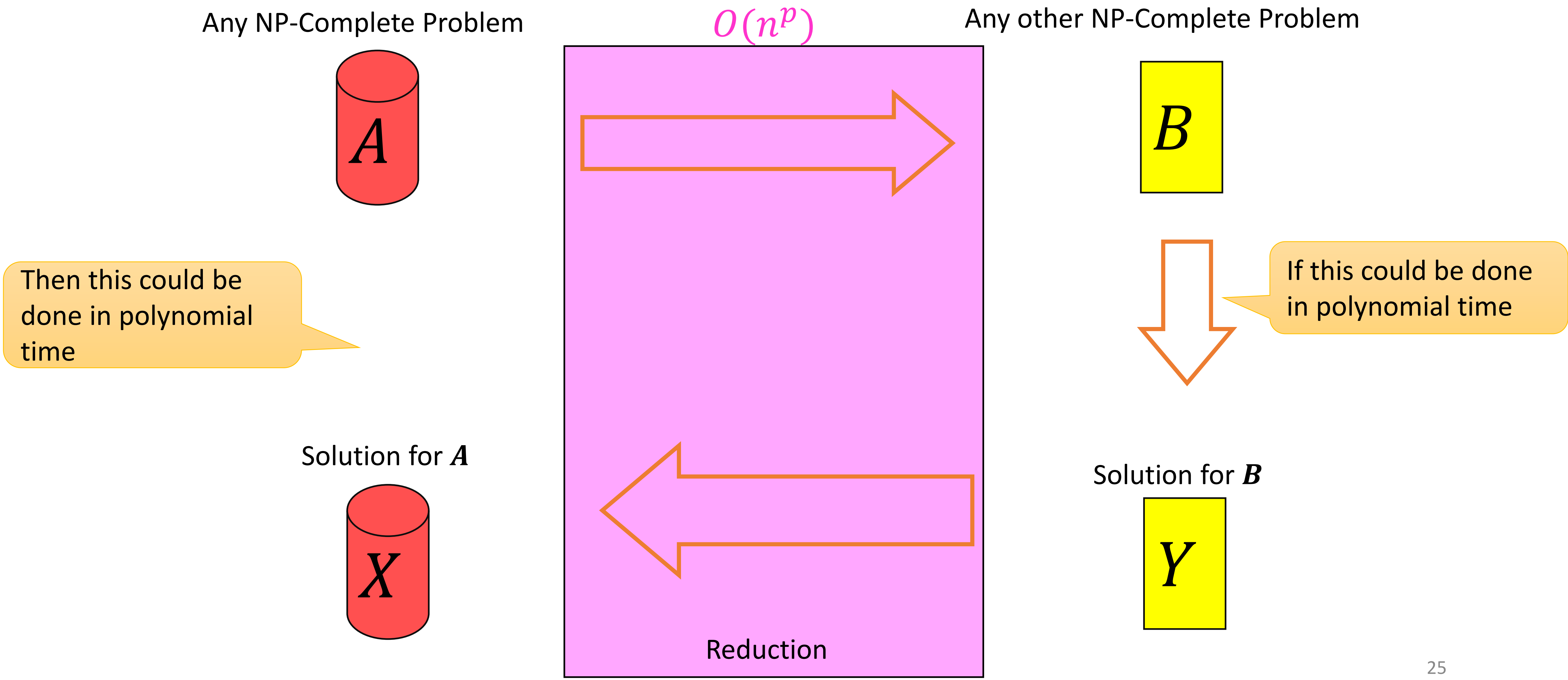


NP-Complete

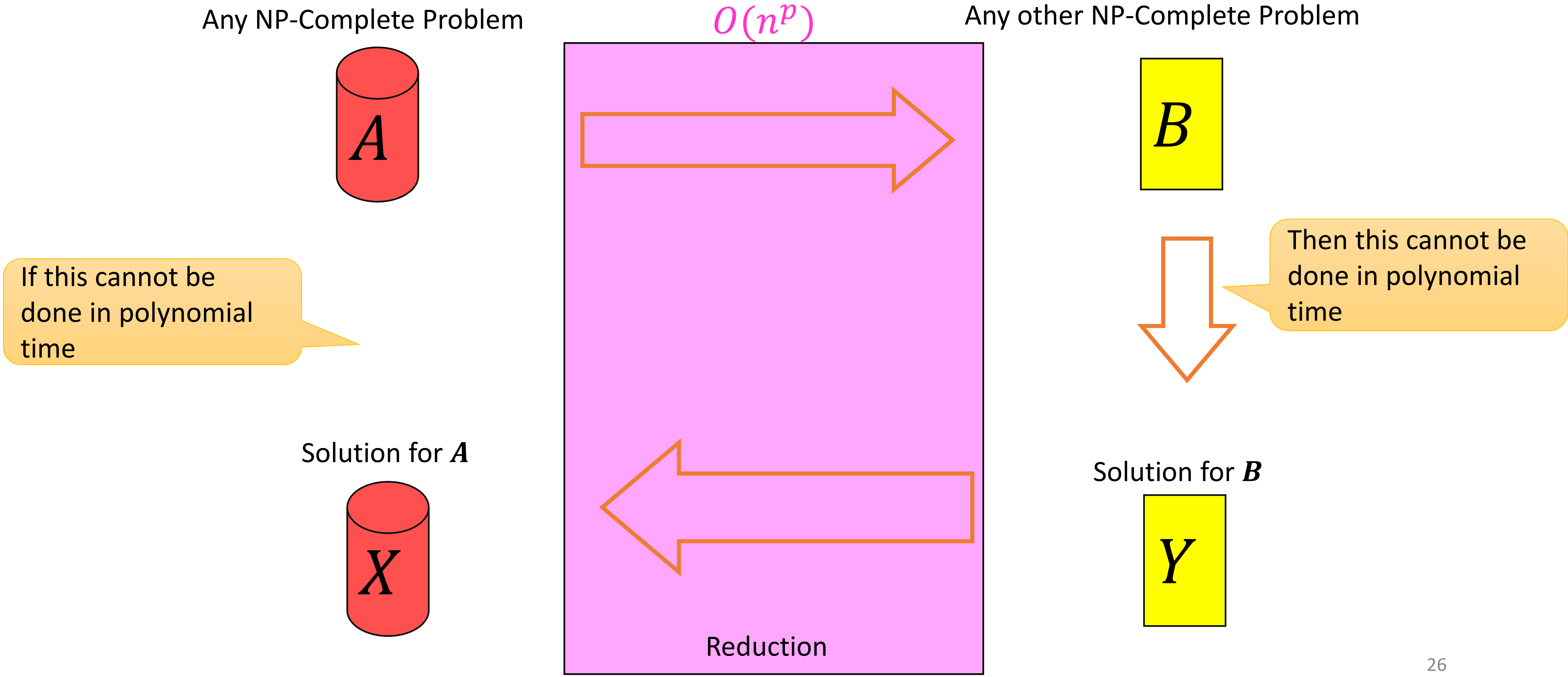


- "Together they stand, together they fall"
- Problems solvable in polynomial time iff ALL NP problems are
- NP-Complete = NP ∩ NP-Hard
- **How to show a problem is NP-Complete?**
  - Show it belongs to NP
    - Give a polynomial time verifier
  - Show it is NP-Hard
    - Give a reduction from another NP-H problem

NP-Completeness



NP-Completeness



Overview

- Problems not belonging to P are considered intractable
- The problems within NP have some properties that make them seem like they might be tractable, but we've been unsuccessful with finding polynomial time algorithms for many
- The class $NP - Complete$ contains problems with the properties:
  - All members are also members of NP
  - All members of NP can be transformed into every member of $NP - Complete$
    - Because they are both NP and $NP - Hard$
  - If any one member of $NP - Complete$ belongs to P, then $P = NP$
  - If any one member of $NP - Complete$ is outside of P, then $P \neq NP$

Why should YOU care?

- If you can find a polynomial time algorithm for any $NP - Complete$ problem then:
  - You will win $1million
  - You will win a Turing Award
  - You will be world famous
  - You will have done something that no one else on Earth has been able to do in spite of the above!
- If you are told to write an algorithm a problem that is $NP - Complete$
  - You can tell that person everything above to set expectations
  - Change the requirements!
  - **Approximate the solution**: Instead of finding a path that visits every node, find a path that visits at least 75% of the nodes
  - **Add Assumptions**: problem might be tractable if we can assume the graph is acyclic, a tree
  - **Use Heuristics**: Write an algorithm that’s “good enough” for small inputs, ignore edge cases

Why should YOU care?

- The entire field of cryptography relies on it (nearly at least)
  - Requires decrypting with a key is easier than decrypting without a key
    - This is strongly related to requiring a difference in difficulty between verifying a candidate solution and finding a solution in the first place
- If $P \neq NP$
  - Some problems remain intractable
  - Cryptography persists
- If $P = NP$
  - We may get efficient solutions for important problems
  - Cryptography is potentially doomed.