

# CSE 332 Spring 2026

## Lecture 26: P and NP

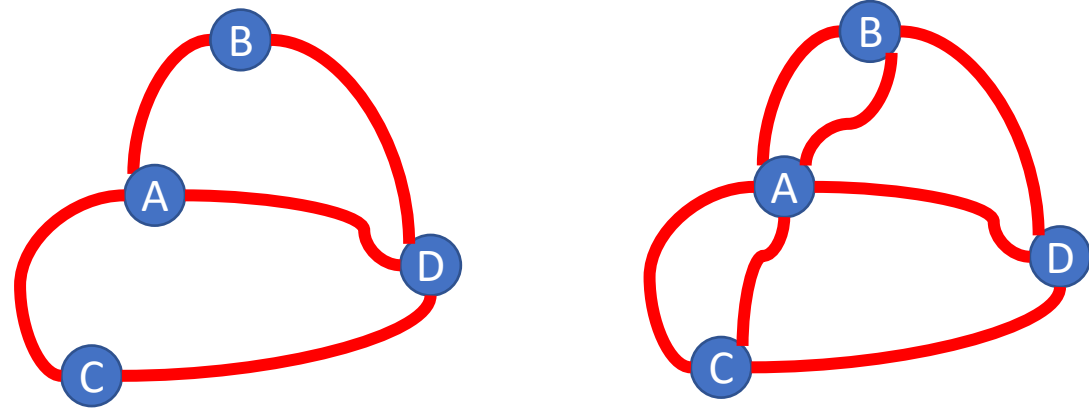
Nathan Brunelle

<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  $O(n)$  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

# Euler Path Problem



- Path:

- A sequence of nodes  $v_1, v_2, \dots$  such that for every consecutive pair are connected by an edge (i.e.  $(v_i, v_{i+1})$  is an edge for each  $i$  in the path)

- Euler Path:

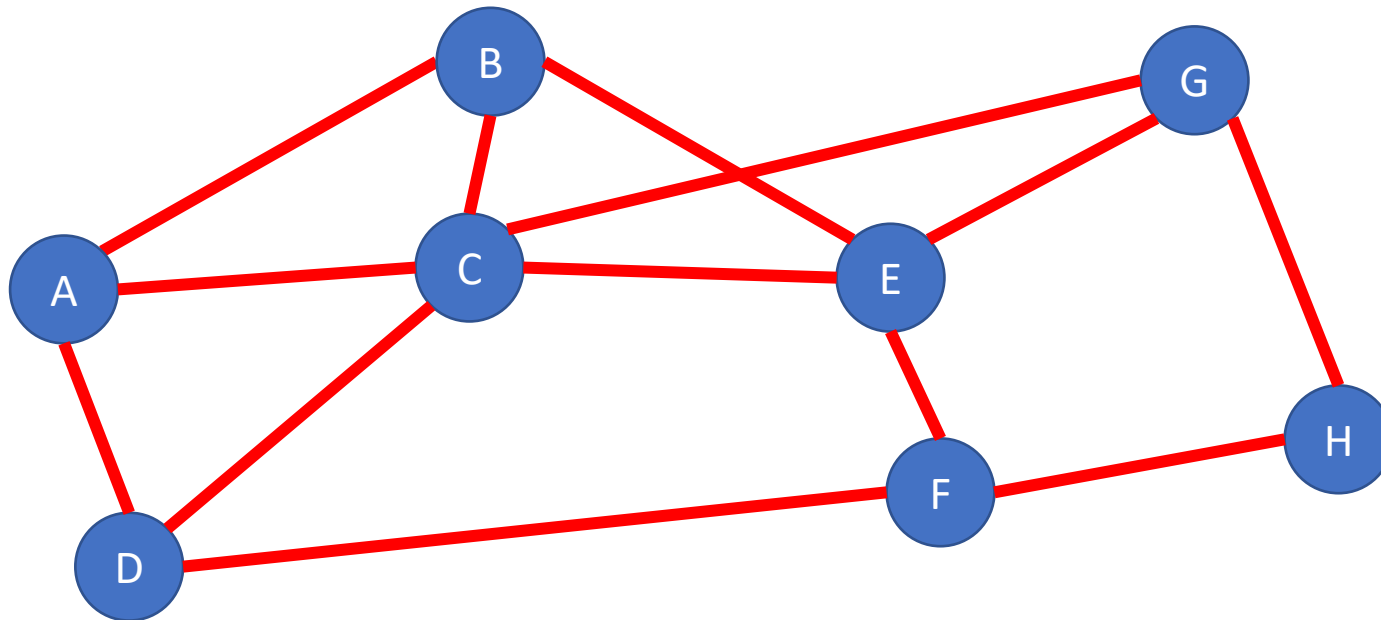
- A path such that every edge in the graph appears exactly once
  - If the graph is not simple then some pairs need to appear multiple times!

- Euler path problem:

- Given an undirected graph  $G = (V, E)$ , does there exist an Euler path for  $G$ ?

# A Seemingly Similar Problem

- Hamiltonian Path:
  - A path that includes every node in the graph exactly once
- Hamiltonian Path Problem:
  - Given a graph  $G = (V, E)$ , does that graph have a Hamiltonian Path?



True!  
*A, B, C, E, G, H, F, D*

# 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

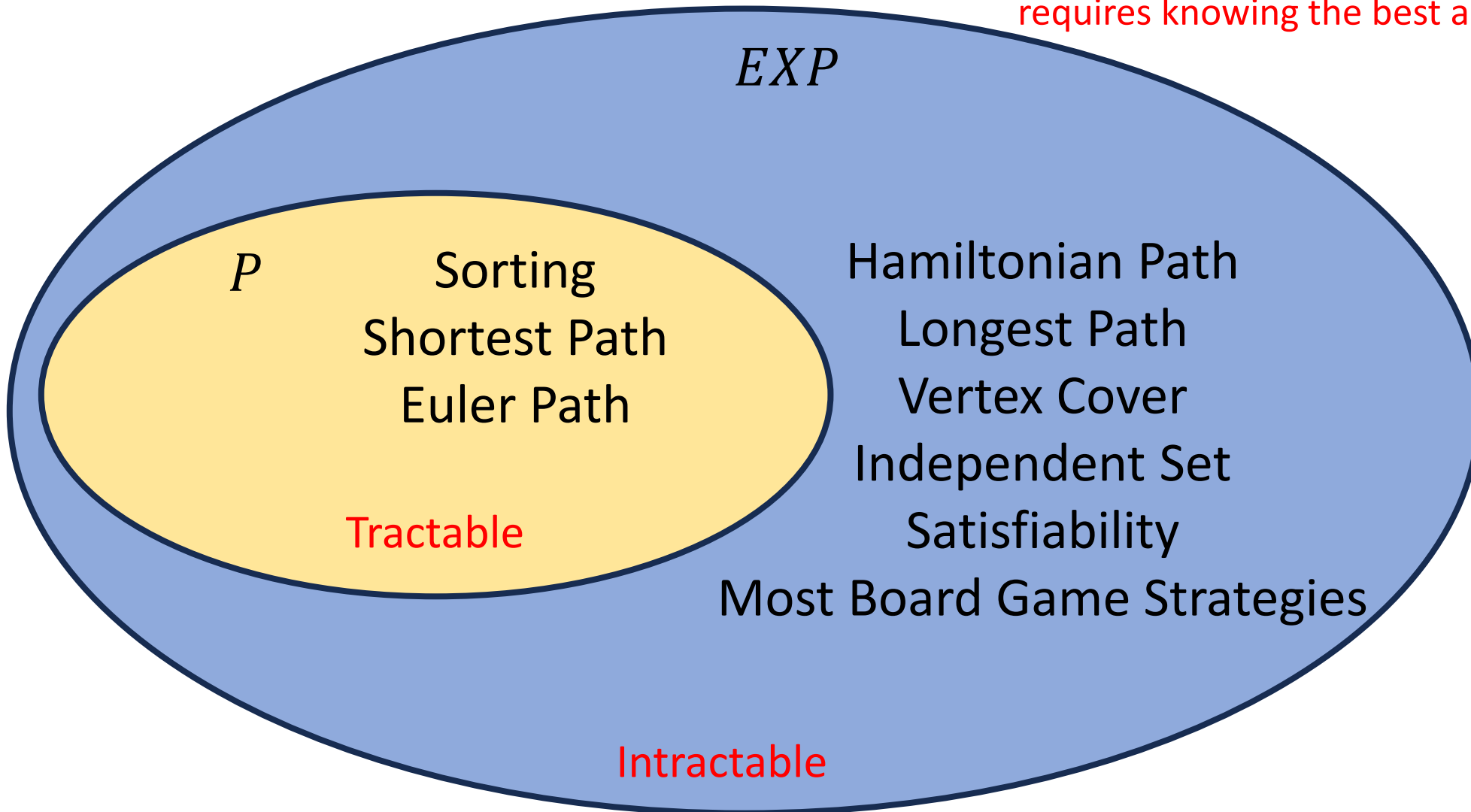
# Complexity Classes and Tractability

- 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  $O(2^{n^p})$  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

# $EXP$ and $P$

**Important!**

Some of the problems we've listed in  $EXP$  could also be members of  $P$ . Since membership is determined by a problem's *most* efficient algorithm, knowing if a problem belongs to  $P$  requires knowing the best algorithm possible!



# Studying Complexity and Tractability

- Organizing problems into complexity classes helps us to reason more carefully and flexibly about tractability
- The goal for each problem is to either
  - Find an efficient algorithm if it exists
    - i.e. show it belongs to  $P$
  - Prove that no efficient algorithm exists
    - i.e. show it does not belong to  $P$
- Complexity classes allow us to reason about sets of problems at a time, rather than each problem individually
  - If we can find more precise classes to organize problems into, we might be able to draw conclusions about the entire class
  - It may be easier to show a problem belongs to class  $C$  than to  $P$ , so it may help to show that  $C \subseteq P$

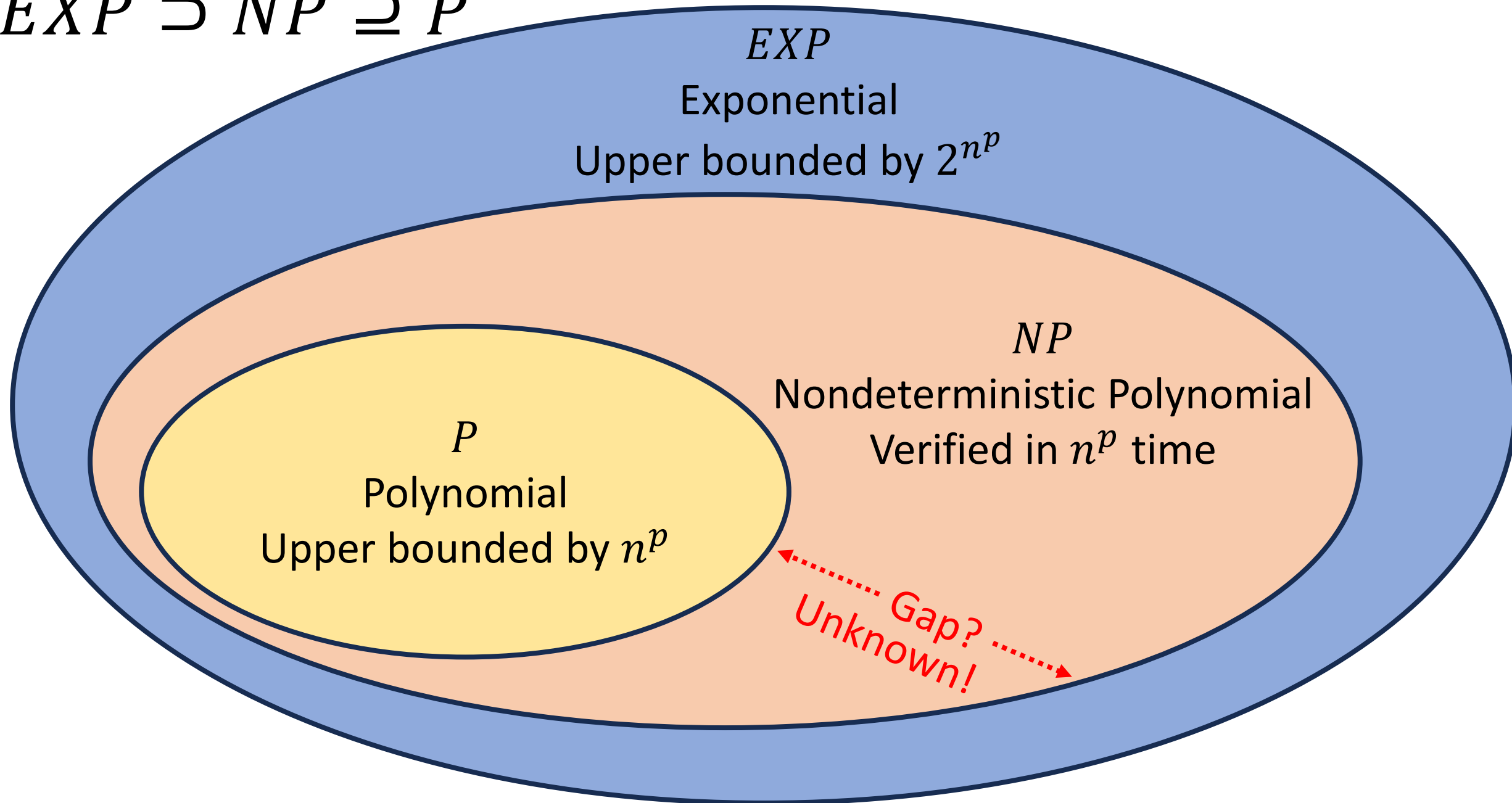
# Some problems in *EXP* seem “easier”

- 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

# Class $NP$

- $NP$ 
  - The set of problems for which a candidate solution can be verified in polynomial time
  - Stands for “Non-deterministic Polynomial”
    - Corresponds to algorithms that can guess a solution (if it exists), that solution is then verified to be correct in polynomial time
    - Can also think of as allowing a special operation that allows the algorithm to magically guess the right choice at each step of an exhaustive search
- $P \subseteq NP$ 
  - Why?

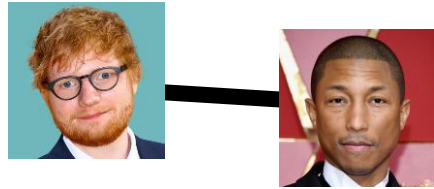
$$EXP \supset NP \supseteq P$$



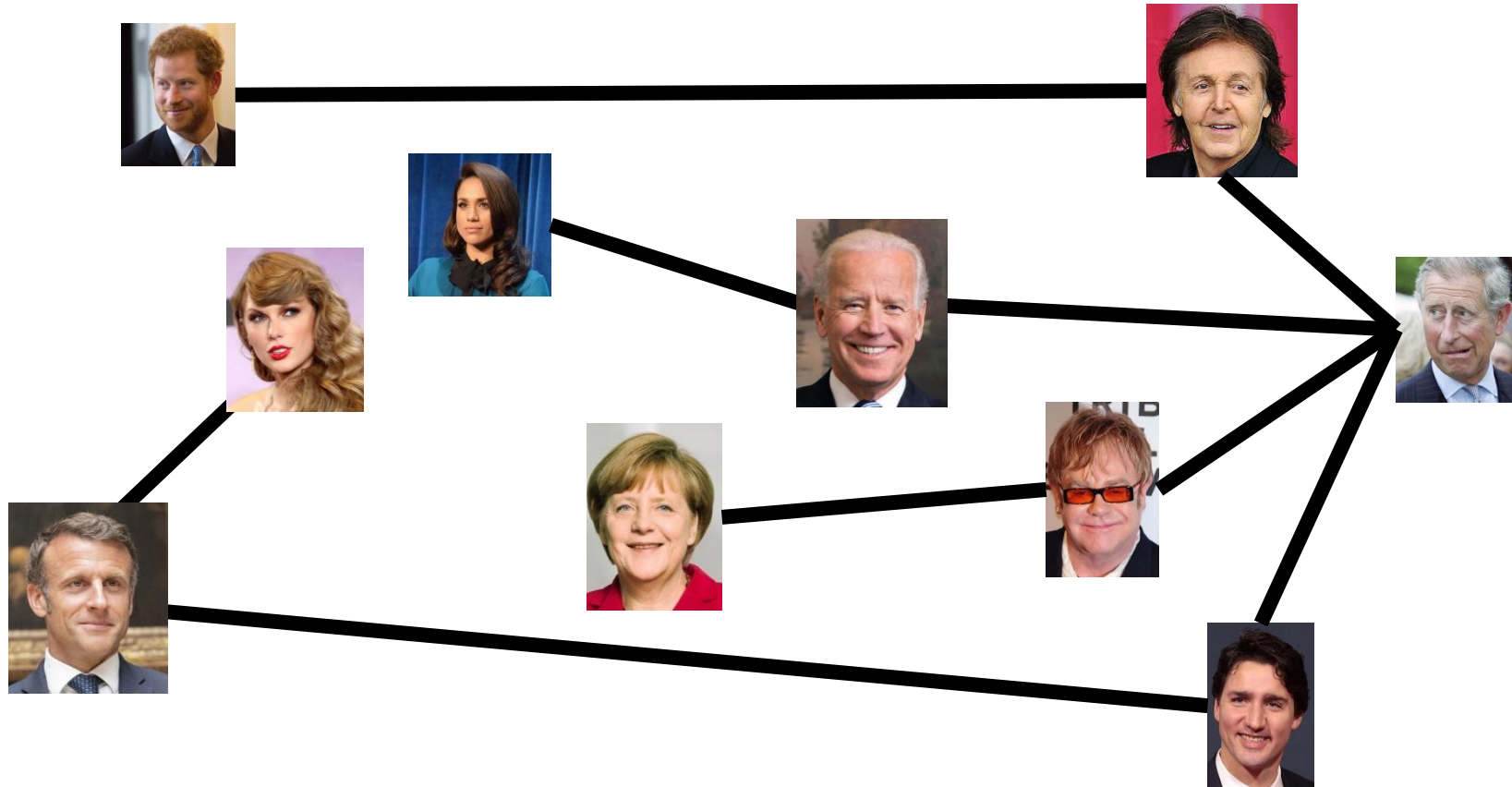
# Solving and Verifying Hamiltonian Path

- Give an algorithm to solve Hamiltonian Path
  - Input:  $G = (V, E)$
  - Output: True if  $G$  has a Hamiltonian Path
  - Algorithm: Check whether each permutation of  $V$  is a path.
    - Running time:  $|V|!$ , so does not show whether it belongs to  $P$
- Give an algorithm to verify Hamiltonian Path
  - Input:  $G = (V, E)$  and a sequence of nodes
  - Output: True if that sequence of nodes is a Hamiltonian Path
  - Algorithm:
    - Check that each node appears in the sequence exactly once
    - Check that the sequence is a path
    - Running time:  $O(V \cdot E)$ , so it belongs to  $NP$

# Party Problem



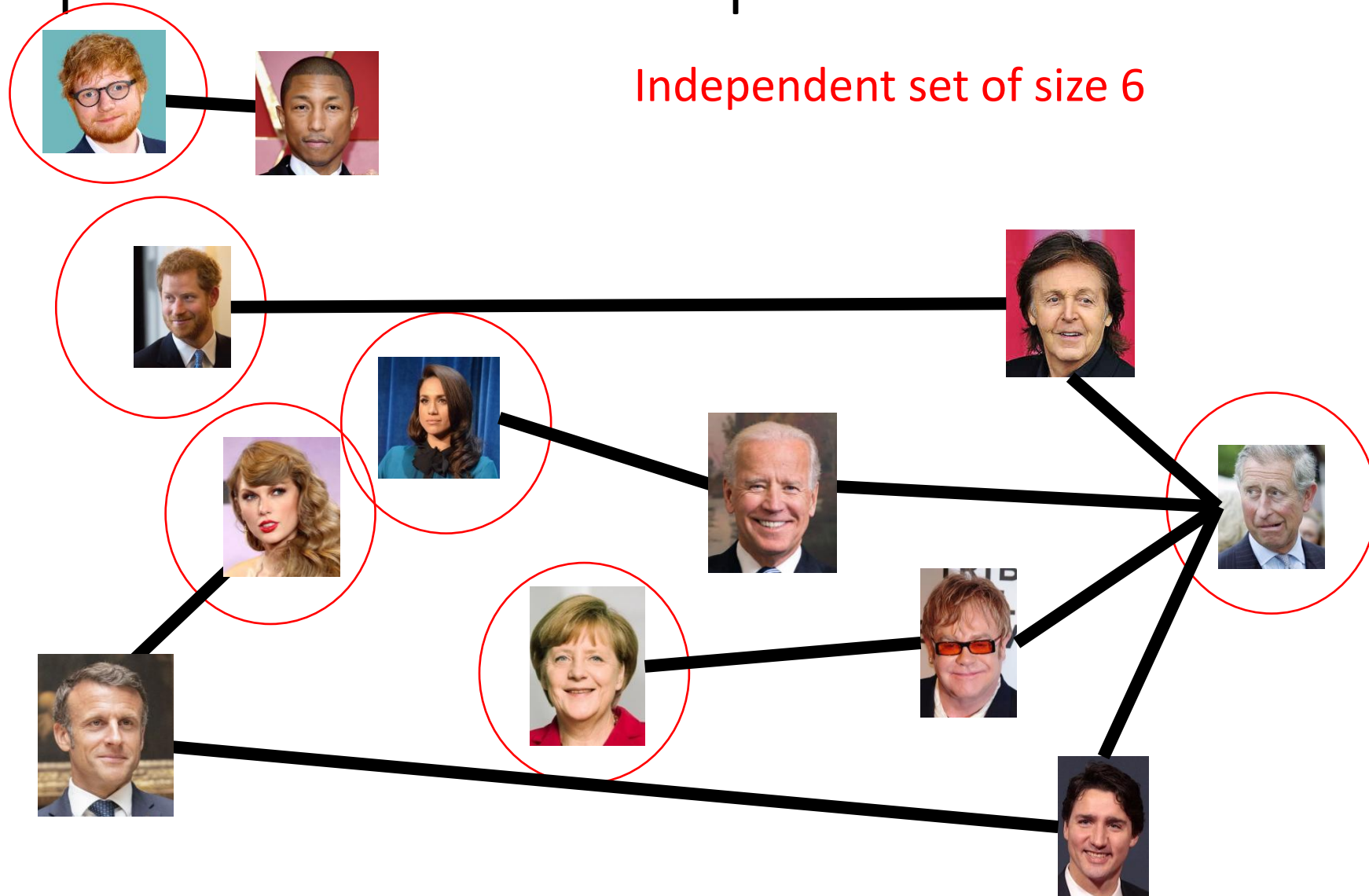
Draw Edges between people who don't get along  
How many people can I invite to a party if everyone must get along?



# 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  $k$

# Independent Set Example

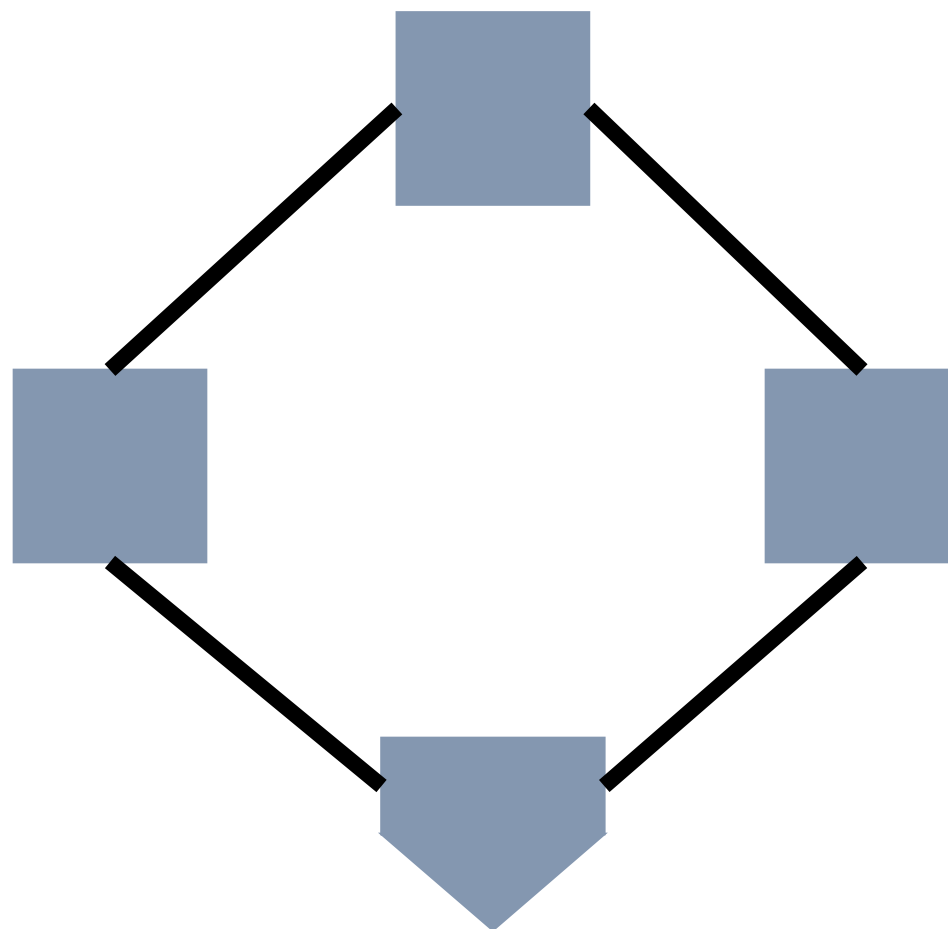


Independent set of size 6

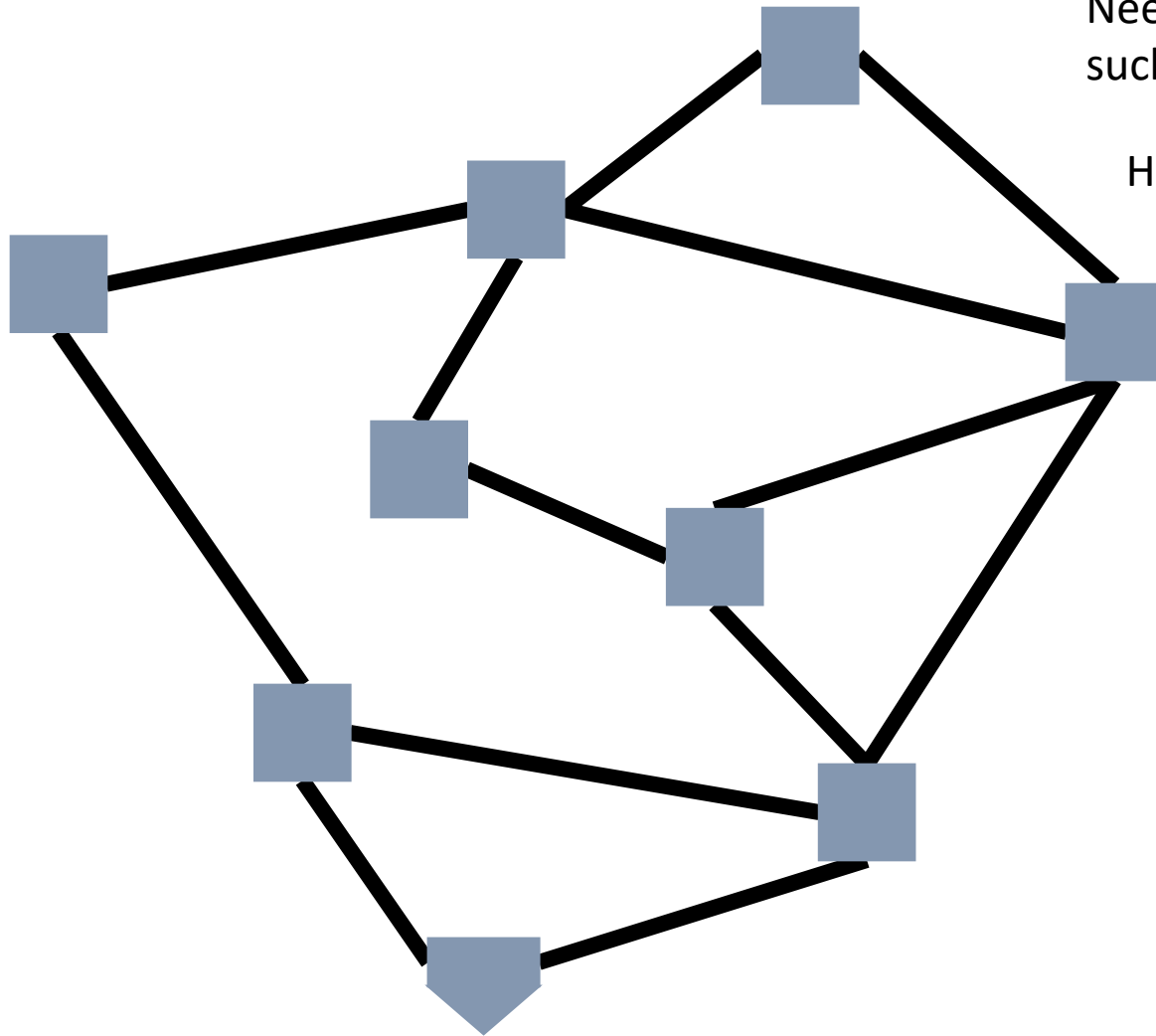
# Solving and Verifying Independent Set

- Give an algorithm to **solve** independent set
  - Input:  $G = (V, E)$  and a number  $k$
  - Output: True if  $G$  has an independent set of size  $k$
- Give an algorithm to **verify** independent set
  - Input:  $G = (V, E)$ , a number  $k$ , and a set  $S \subseteq V$
  - Output: True if  $S$  is an independent set of size  $k$

Baseball



# Generalized Baseball



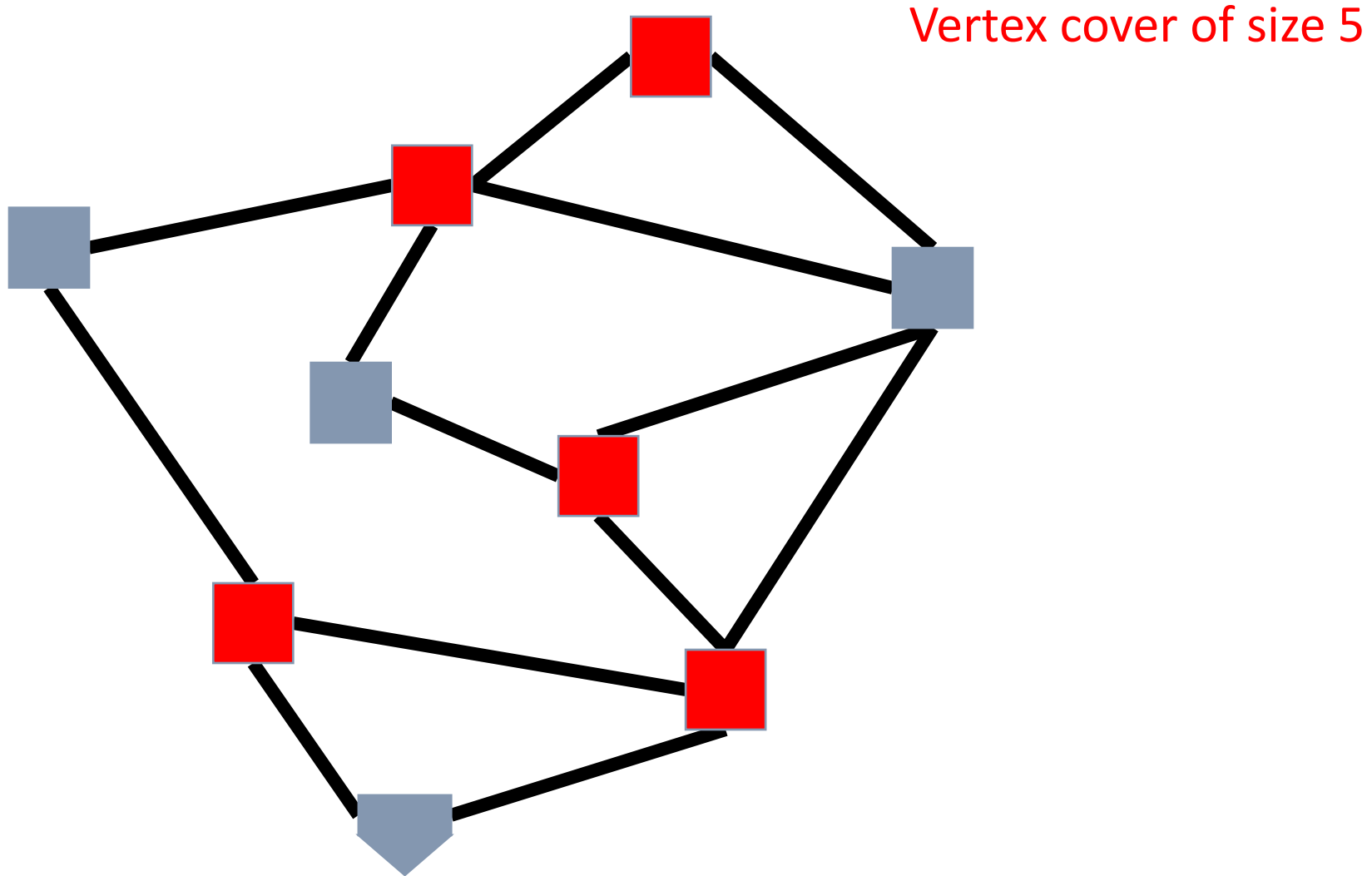
Need to place defenders on bases such that every edge is defended

How many defenders would suffice?

# 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  $k$

# Vertex Cover Example

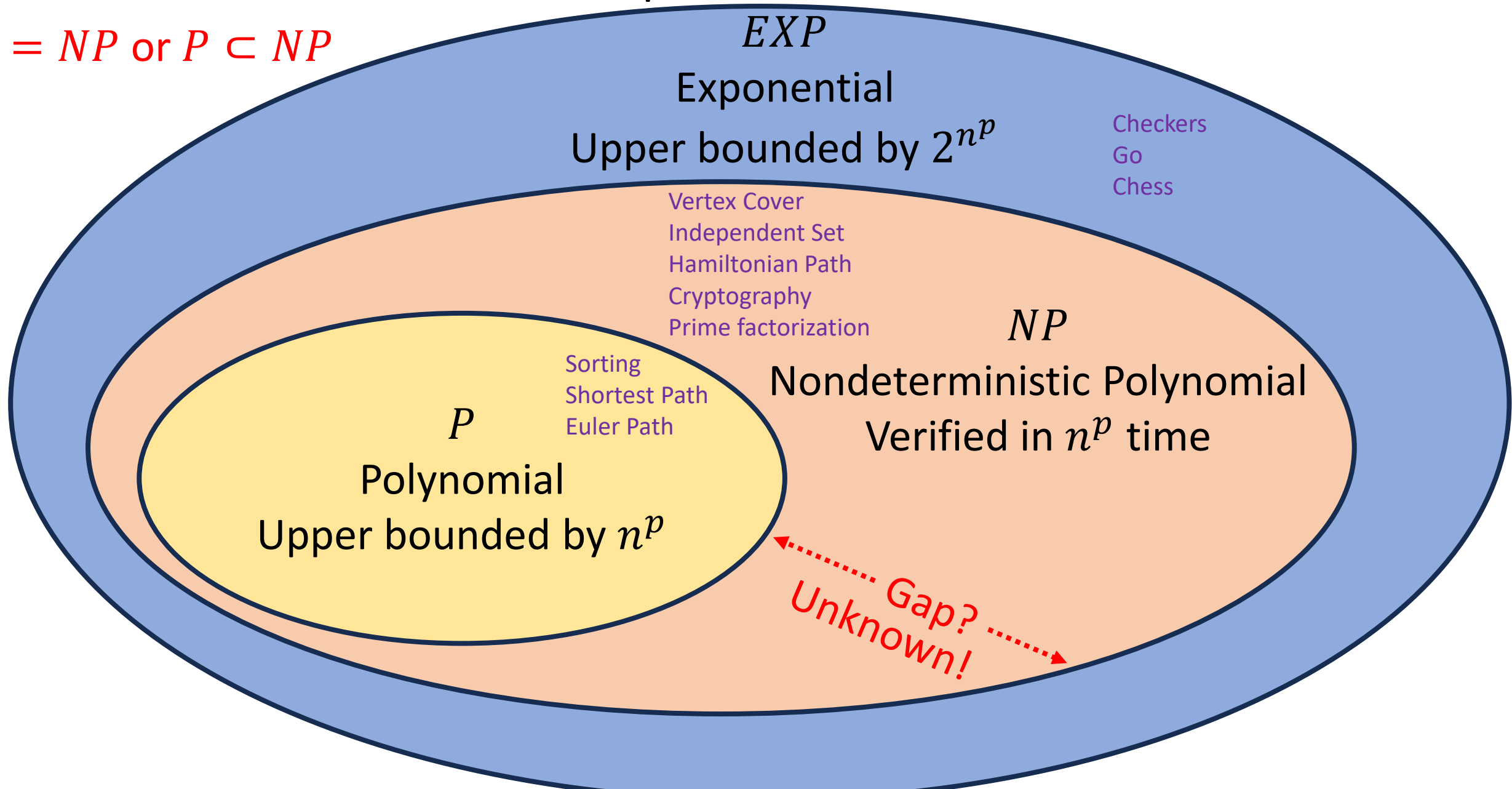


# Solving and Verifying Vertex Cover

- Give an algorithm to **solve** vertex cover
  - Input:  $G = (V, E)$  and a number  $k$
  - Output: True if  $G$  has a vertex cover of size  $k$
- Give an algorithm to **verify** vertex cover
  - Input:  $G = (V, E)$ , a number  $k$ , and a set  $S \subseteq V$
  - Output: True if  $S$  is a vertex cover of size  $k$

# $EXP \supset NP \supseteq P$ Example Members

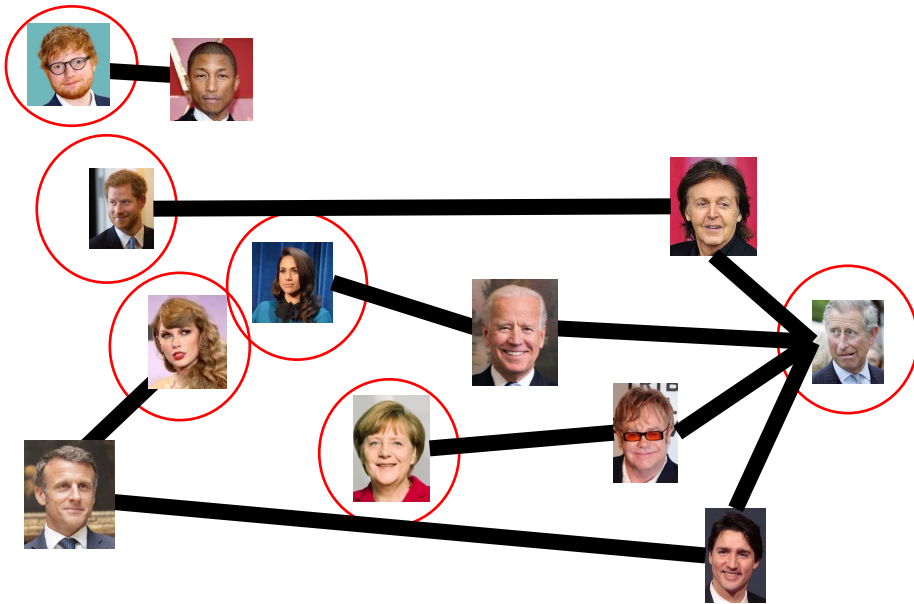
$P = NP$  or  $P \subset NP$



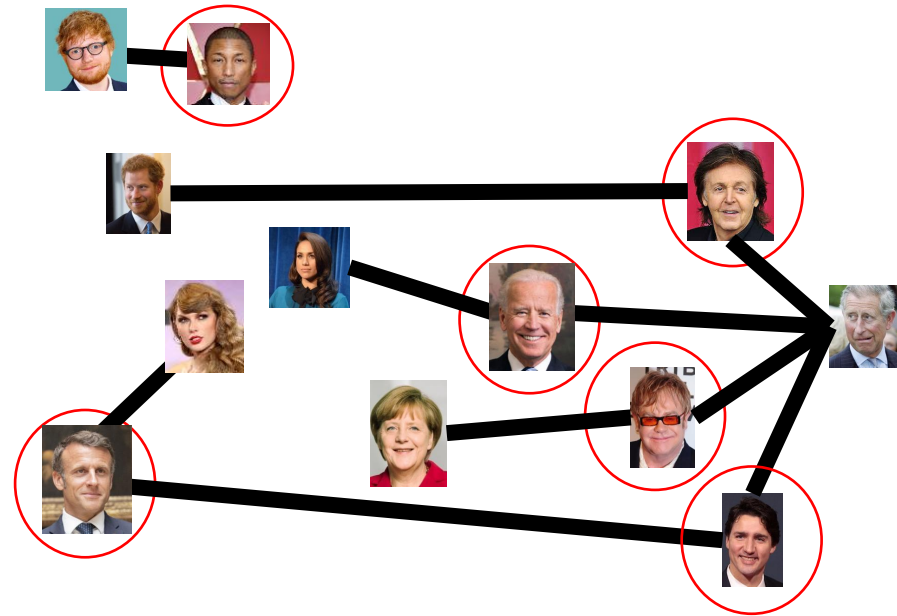
# Transforming an IS into a VC

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

Independent Set



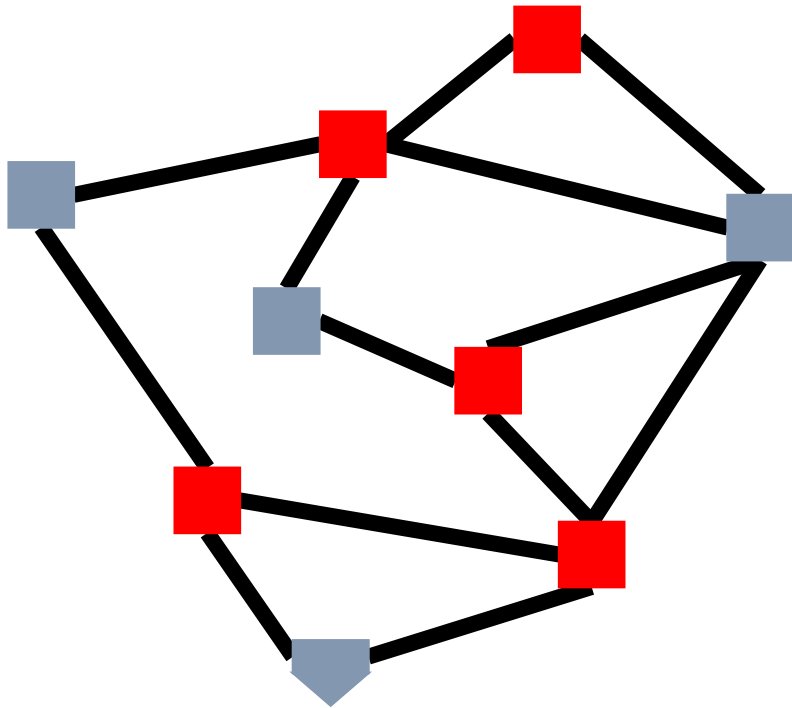
Vertex Cover



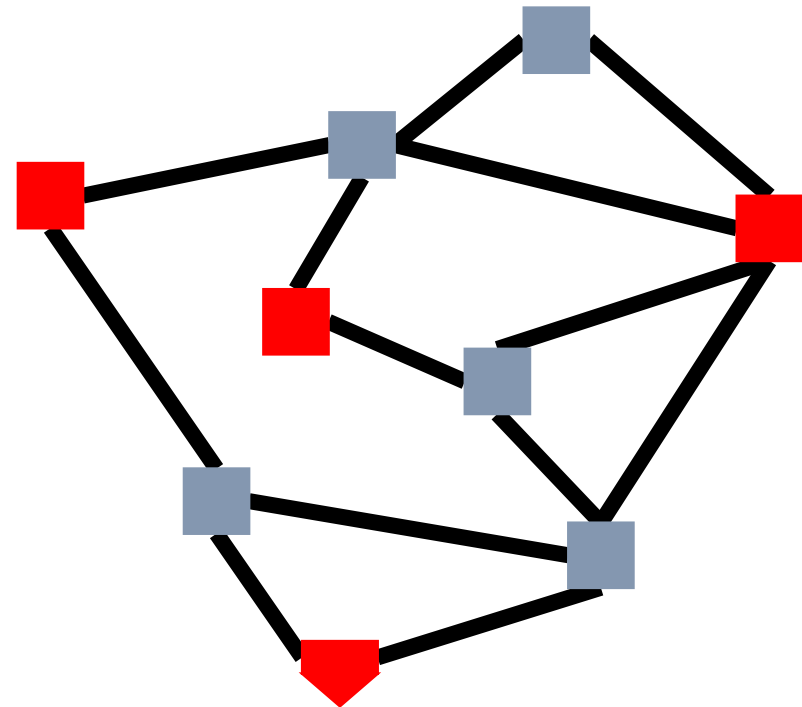
# Transforming a VC into an IS

$S$  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!

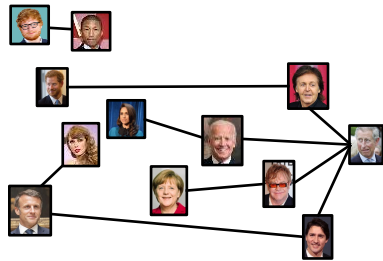
# Reduction

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

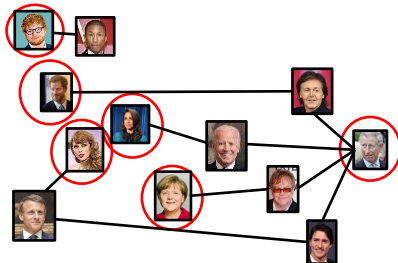
# Independent Set Reduces To Vertex Cover

Independent Set Input

$k$



Independent Set Output



$O(V)$  Time

Relate Independent Set input to Vertex Cover output

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

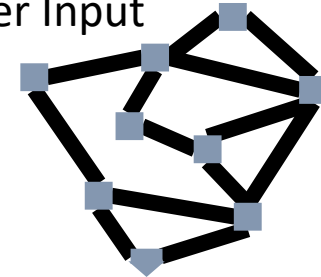
Relate Independent Set output to Vertex Cover output

Give same output

Reduction

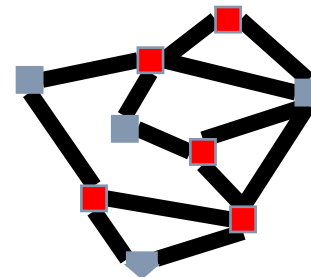
Vertex Cover Input

$k$



Any Algorithm for Vertex Cover

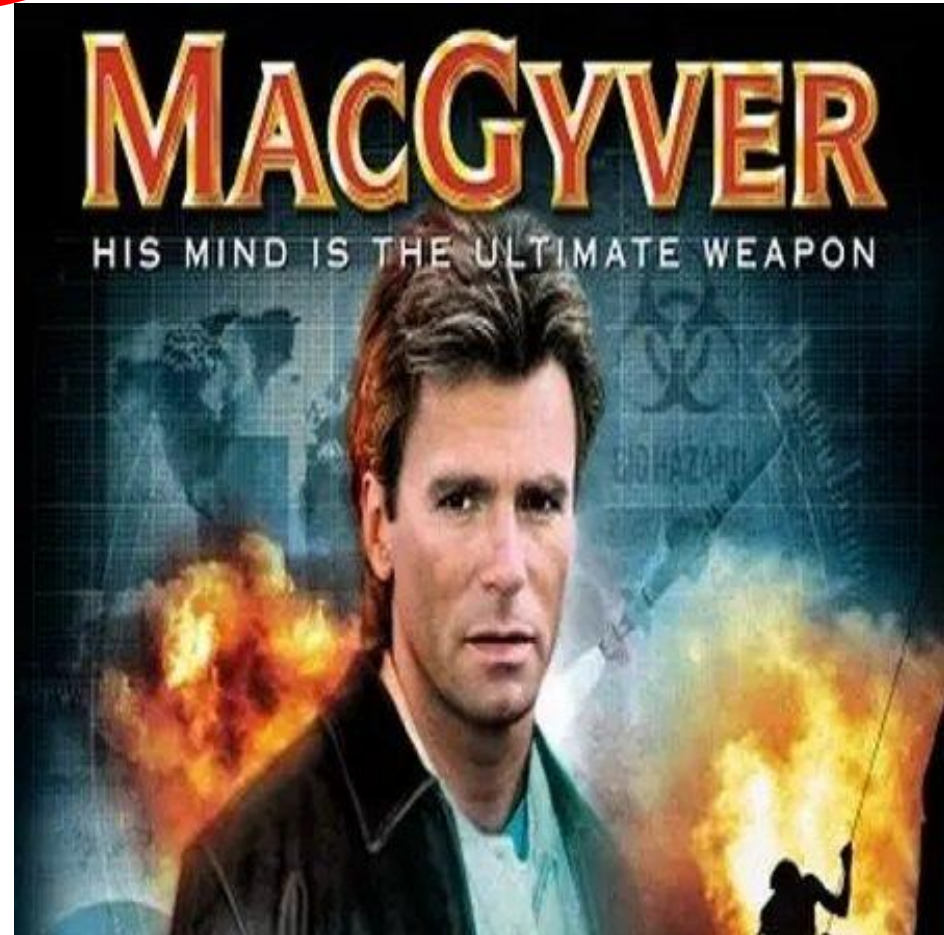
Vertex Cover Output



# Reduction Analogy

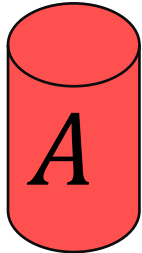
Shows how two different problems relate to each other

**MOVIE TIME!**



# MacGyver's Reduction

Problem we don't know how to solve



Opening a door

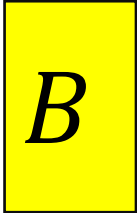


Solution for *A*

Keg cannon  
battering ram



Problem we will solve instead



Lighting a fire

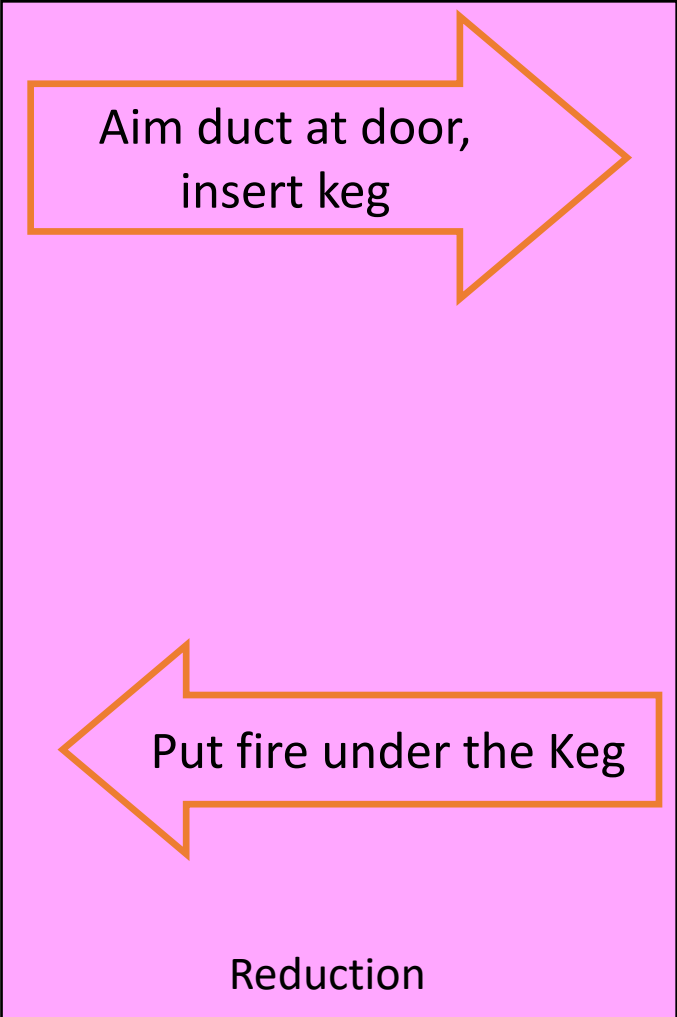


Alcohol, wood,  
matches



HOW?

Solution for *B*

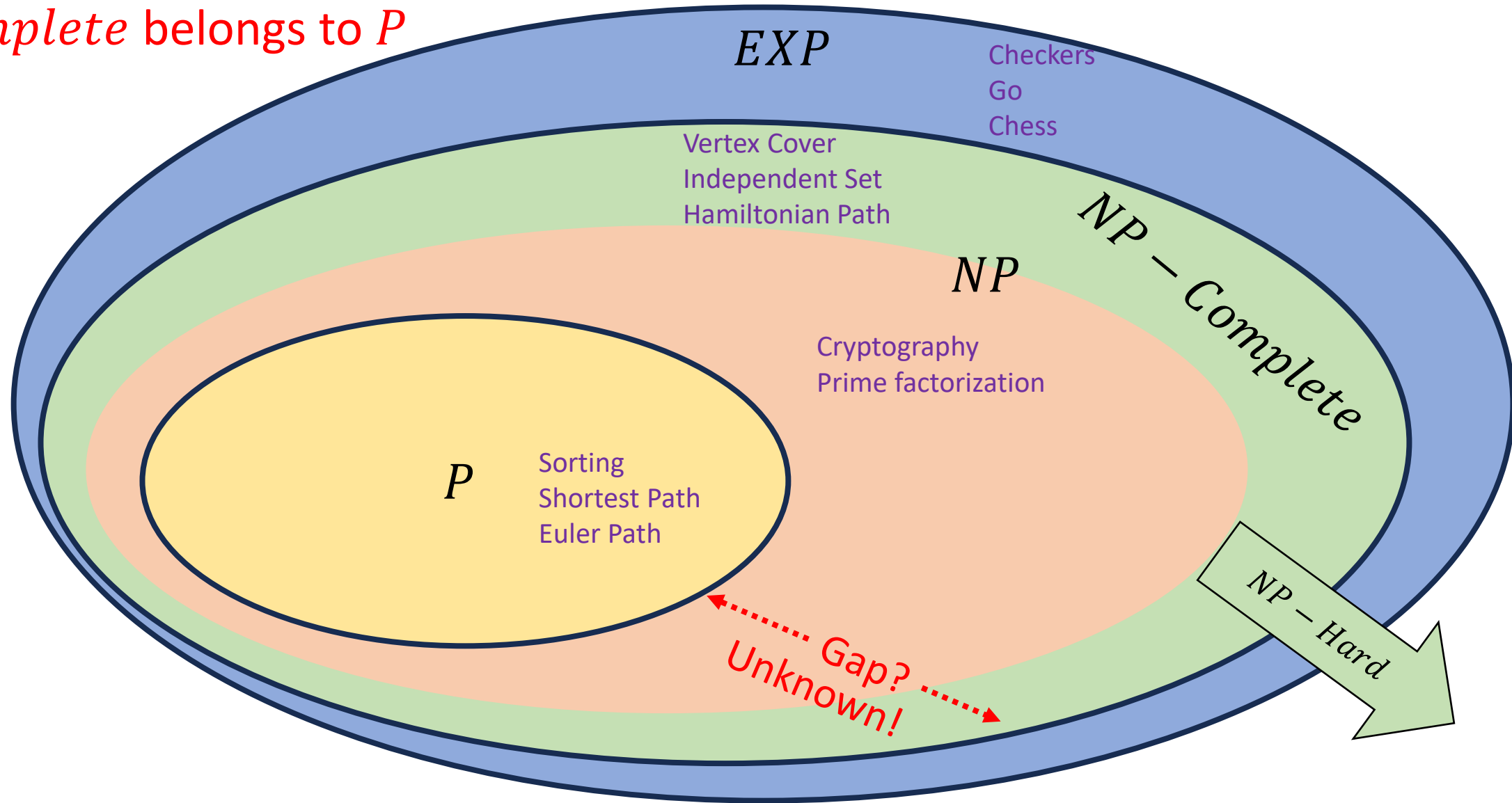


# NP-Complete – High Level

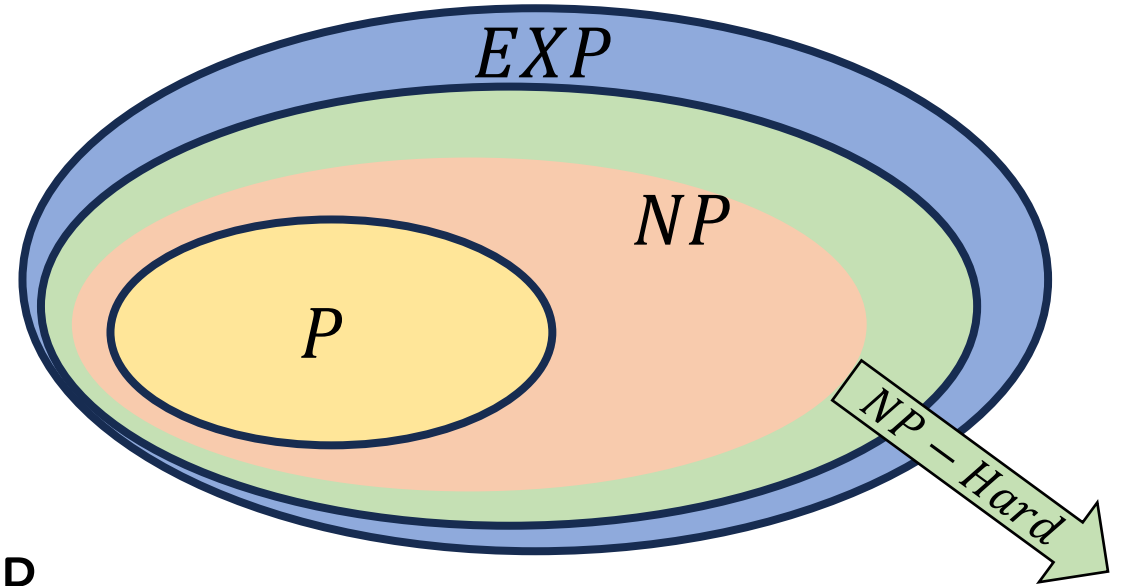
- A set of “together they stand, together they fall” problems
- The problems in this set either all belong to  $P$ , or none of them do
- Intuitively, the “hardest” problems in NP
- Collection of problems from  $NP$  that can all be “transformed” into each other in polynomial time
  - Like we could transform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

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

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



# NP-Hard

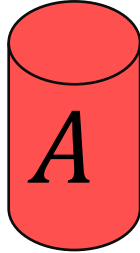


- How can we try to figure out if  $P=NP$ ?
- Identify problems at least as “hard” as  $NP$ 
  - If any of these “hard” problems can be solved in polynomial time, then all  $NP$  problems can be solved in polynomial time.
- Definition:  $NP-Hard$ :
  - $B$  is  $NP-Hard$  provided EVERY problem within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

For every NP problem

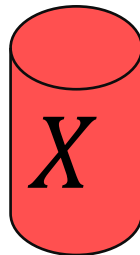
# NP-Hard Idea

Any NP Problem

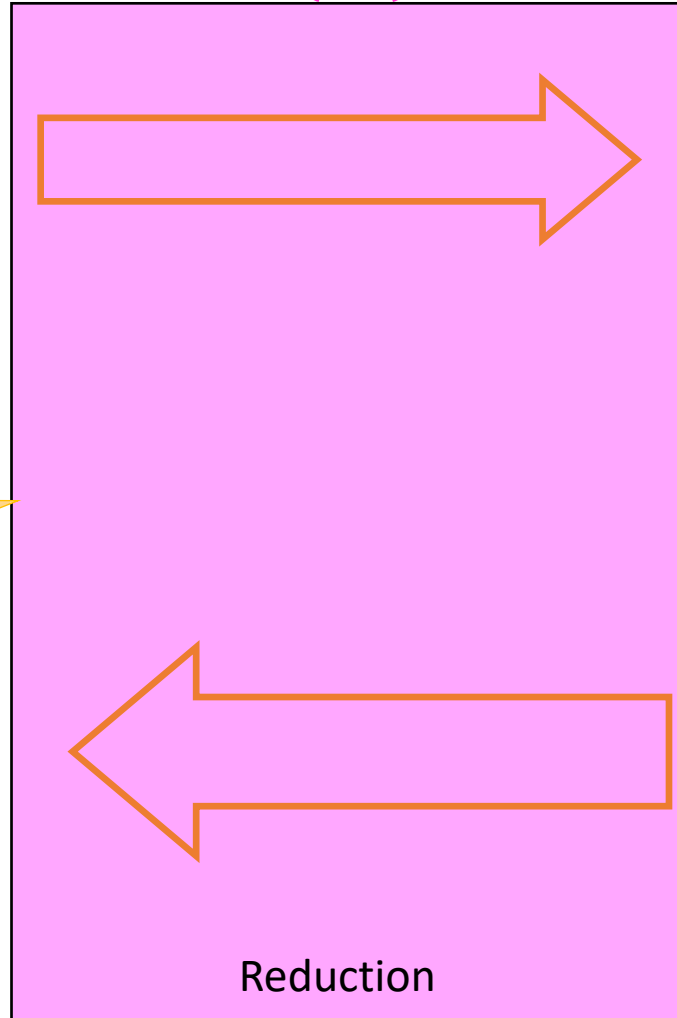


There exists a polynomial-time reduction to each NP-Hard Problem

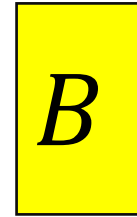
Solution for *A*



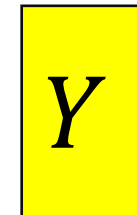
$O(n^p)$



An NP-Hard Problem



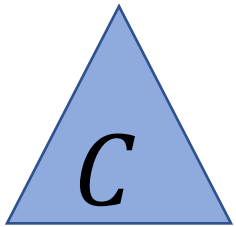
Solution for *B*



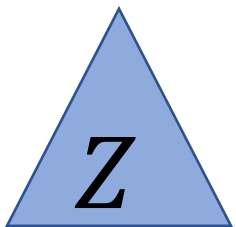
So if this was  $O(n^p)$  we can solve any NP problem in polynomial time

# Showing NP-Hardness

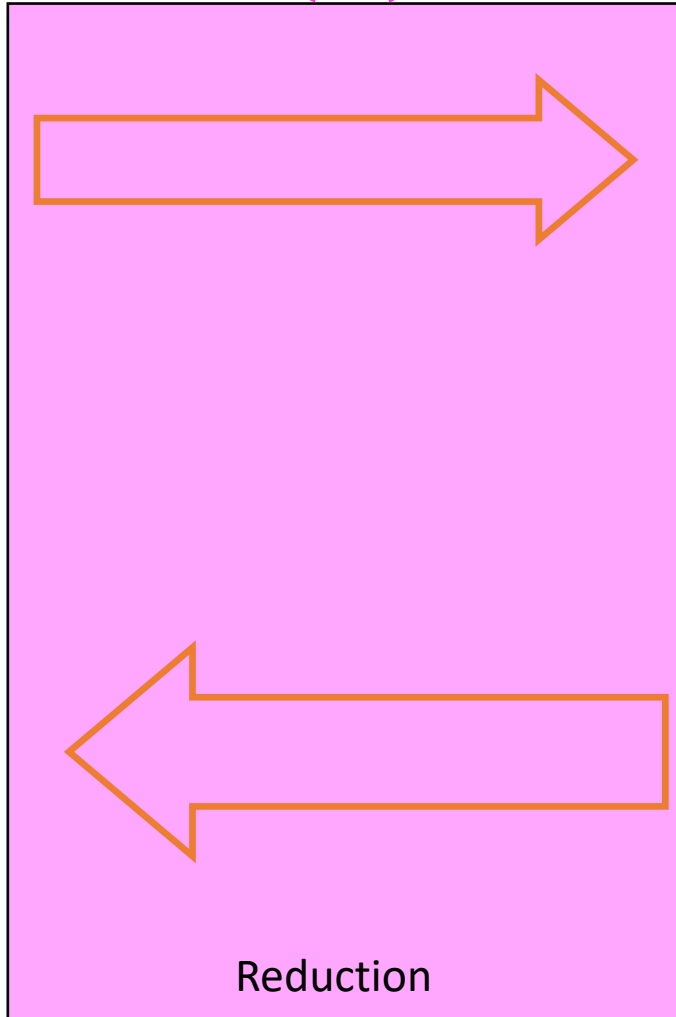
Any NP Problem



Solution for  $C$



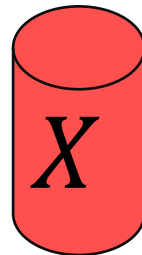
$O(n^p)$



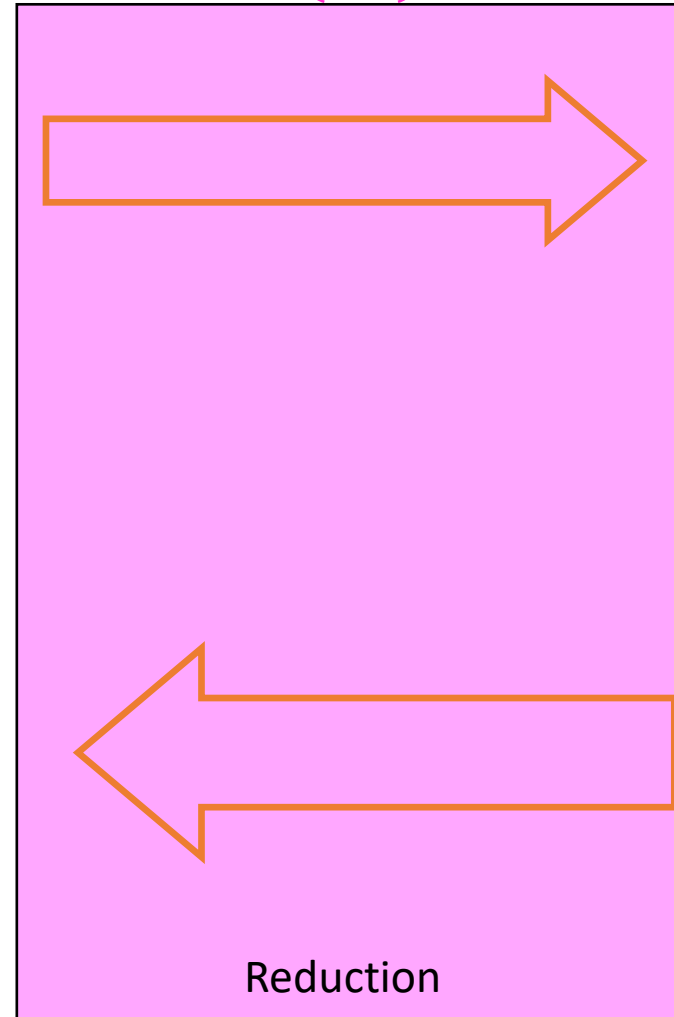
A First NP-Hard Problem



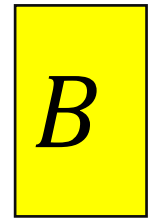
Solution for  $A$



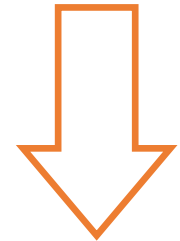
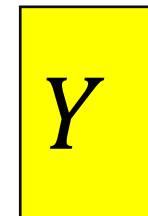
$O(n^p)$



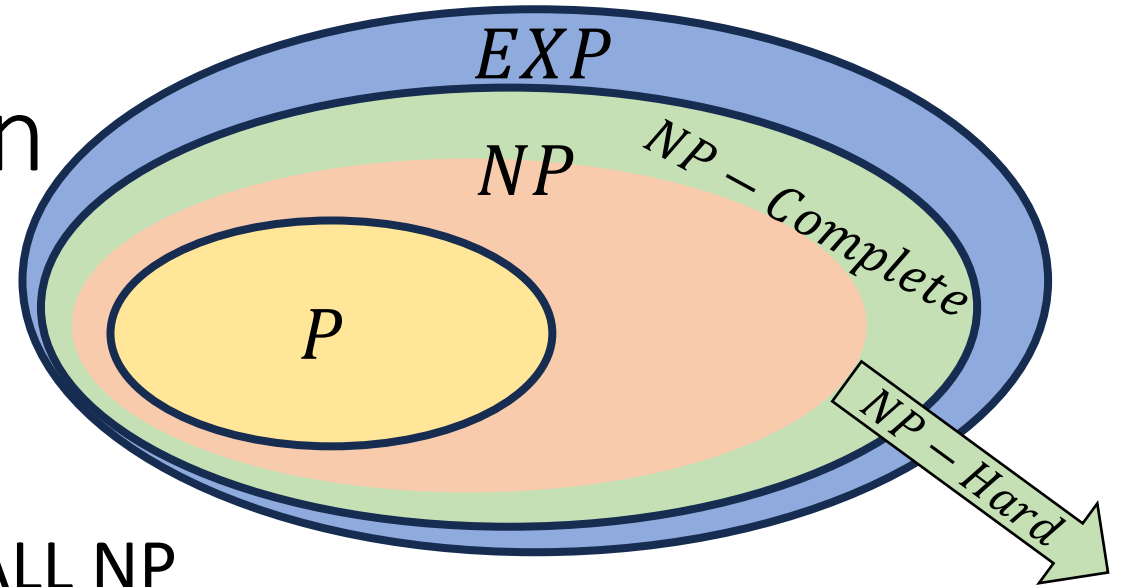
A new NP-Hard Problem



Solution for  $B$



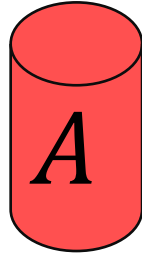
# NP-Complete - Definition



- “Together they stand, together they fall”
- Problems solvable in polynomial time iff ALL NP problems are
- $NP - Complete = NP \cap NP - Hard$
- **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

# Using NP-Completeness

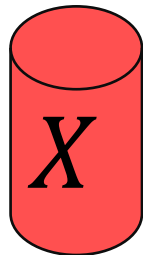
Any NP-Complete Problem



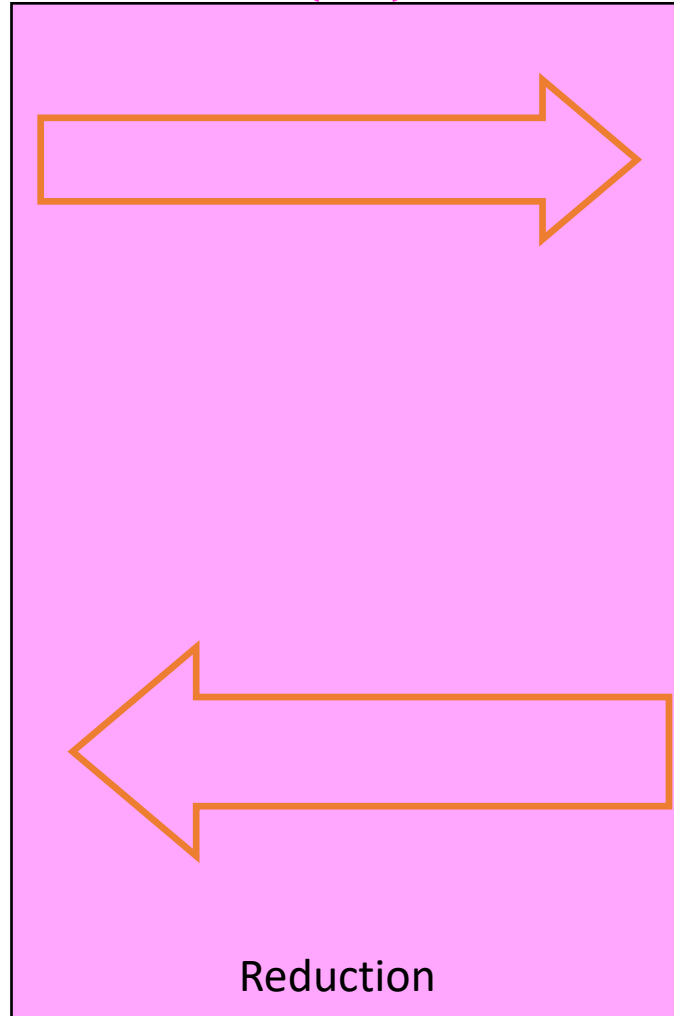
Then this could be done in polynomial time

If this cannot be done in polynomial time

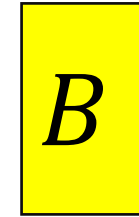
Solution for  $A$



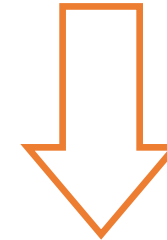
$O(n^p)$



Any other NP-Complete Problem

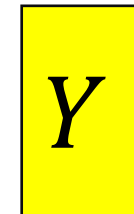


If this could be done in polynomial time



Then this cannot be done in polynomial time

Solution for  $B$



# 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? (1/2)

- 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? (2/2)

- 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.