CSE 332 Spring 2026
Lecture 27: P and NP 2

Nathan Brunelle

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

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"

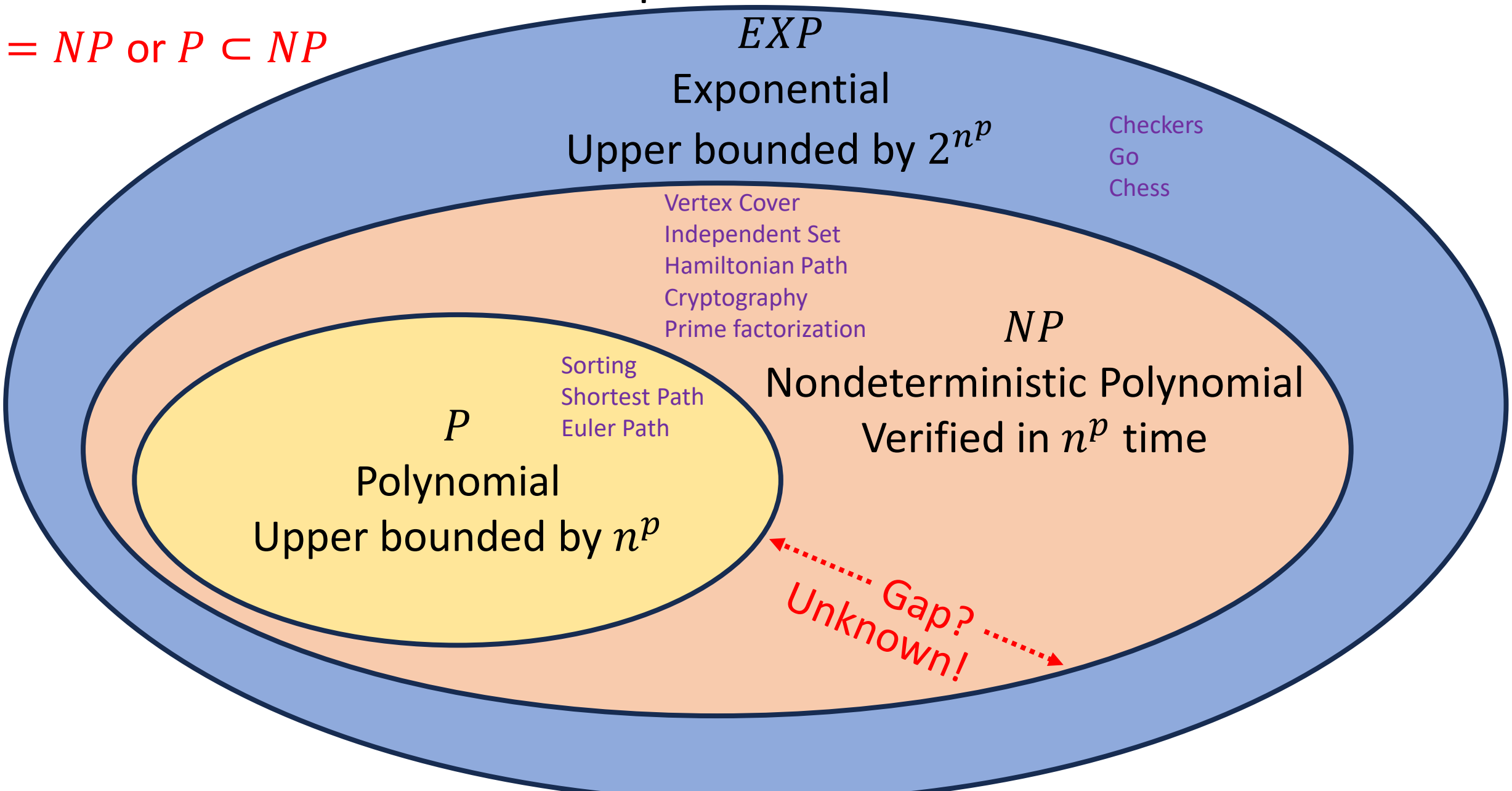
- There are some problems that we do not have polynomial time algorithms to solve, but provided answers are easy to check
- Hamiltonian Path:
  - It's "hard" to look at a graph and determine whether it has a Hamiltonian Path
  - It's "easy" to look at a graph and a candidate path together and determine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

Class NP

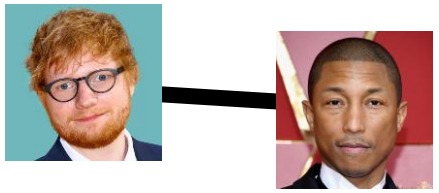
- NP
  - The set of problems for which a candidate solution can be verified in polynomial time
  - Stands for “Non-deterministic Polynomial”
    - Corresponds to algorithms that can guess a solution (if it exists), that solution is then verified to be correct in polynomial time
    - Can also think of 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?

$EXP \supset NP \supseteq P$ Example Members

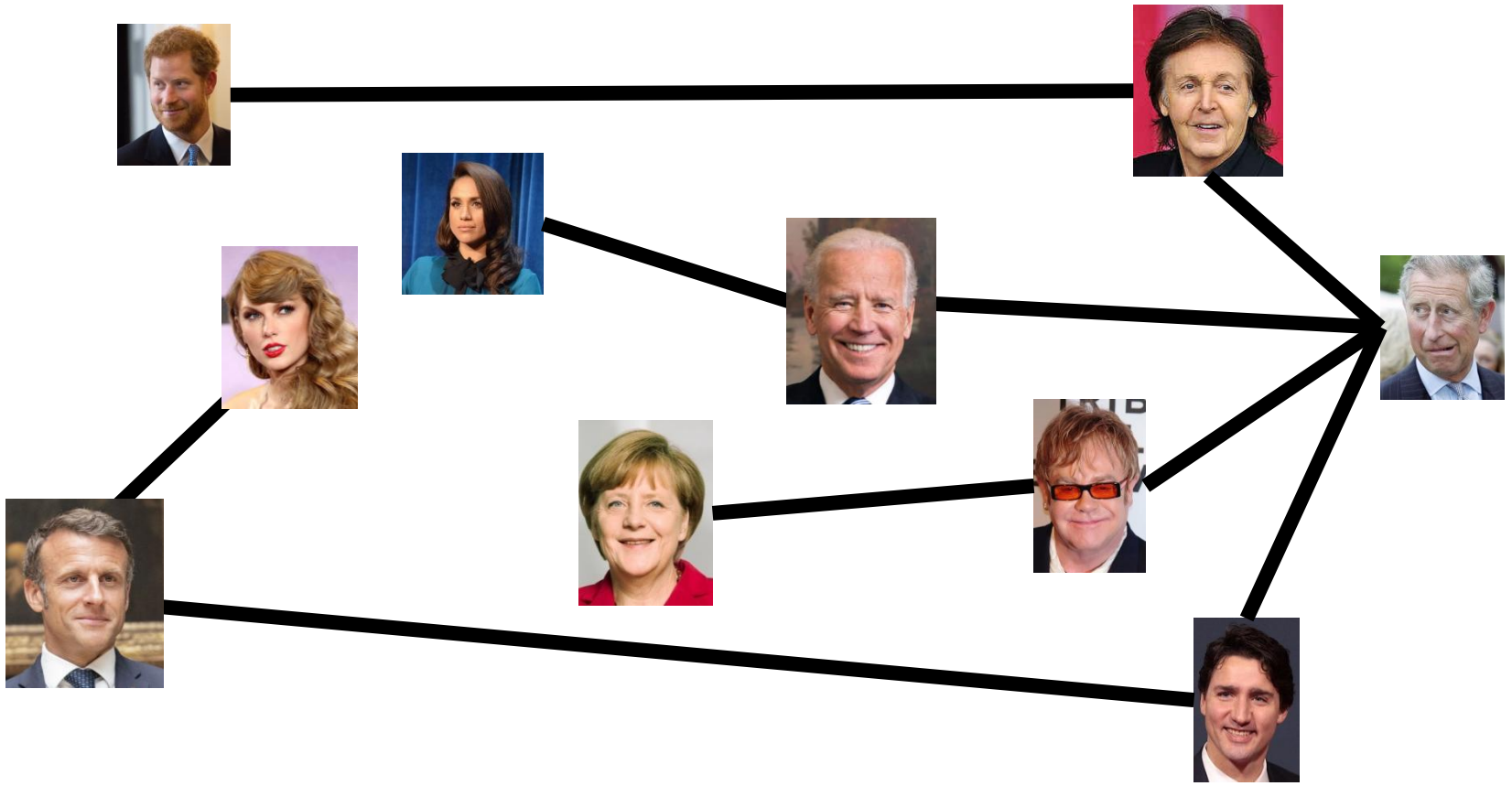
$P = NP$ or $P \subset NP$



Independent Set / Party Problem



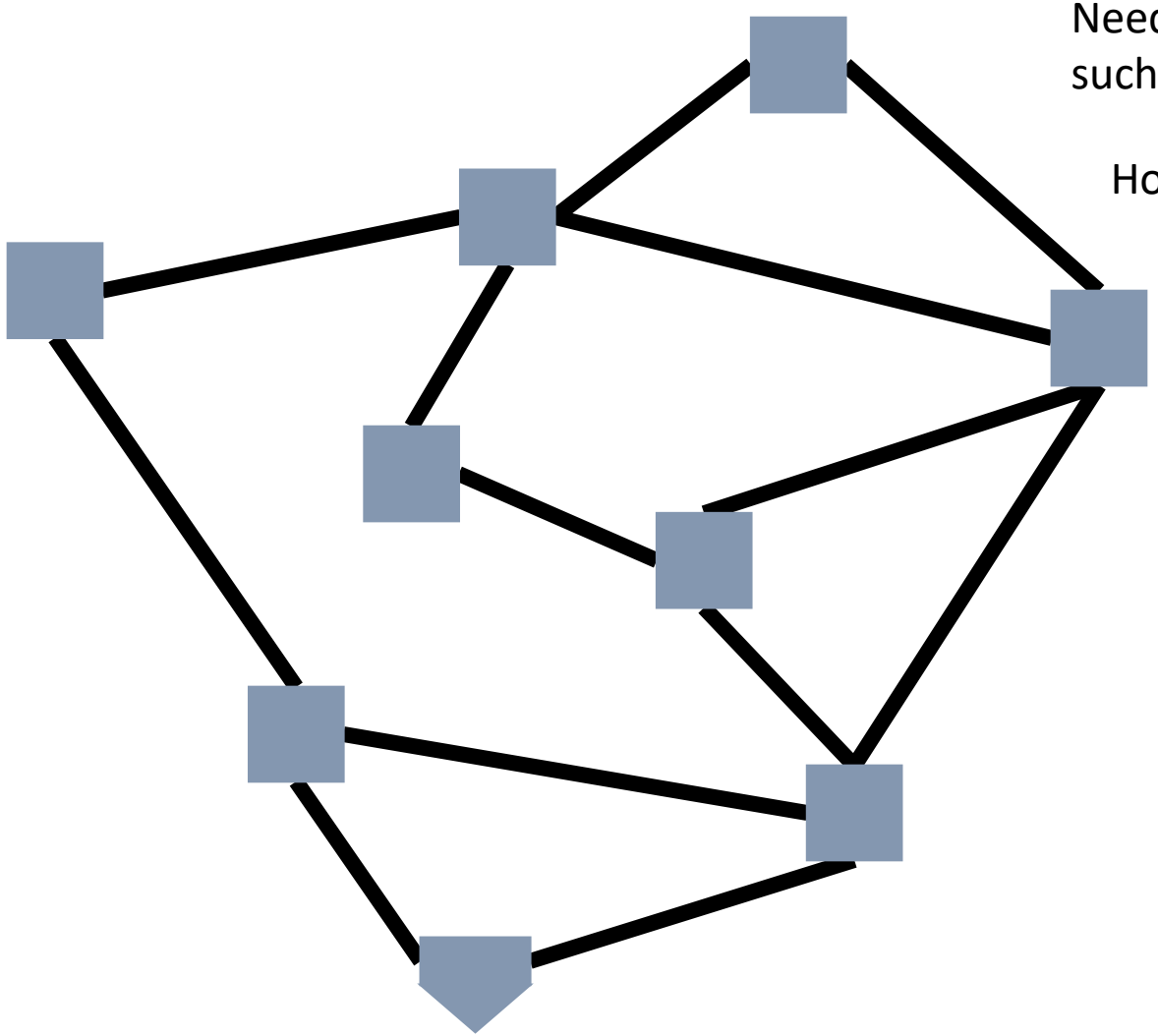
Draw Edges between people who don't get along
How many people can I invite to a party if everyone must get along?



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

Vertex Cover / Generalized Baseball



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

How many defenders would suffice?

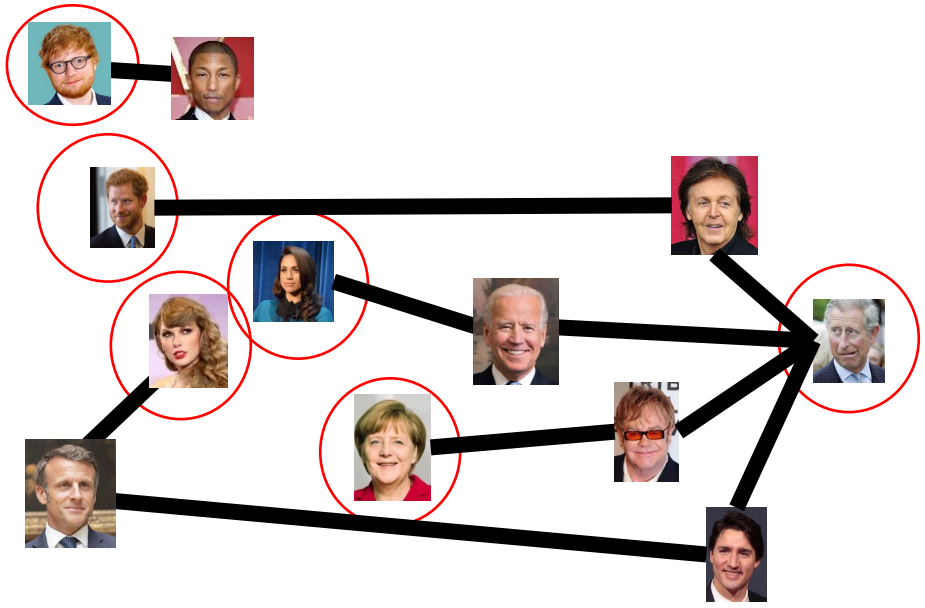
Vertex Cover

- 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

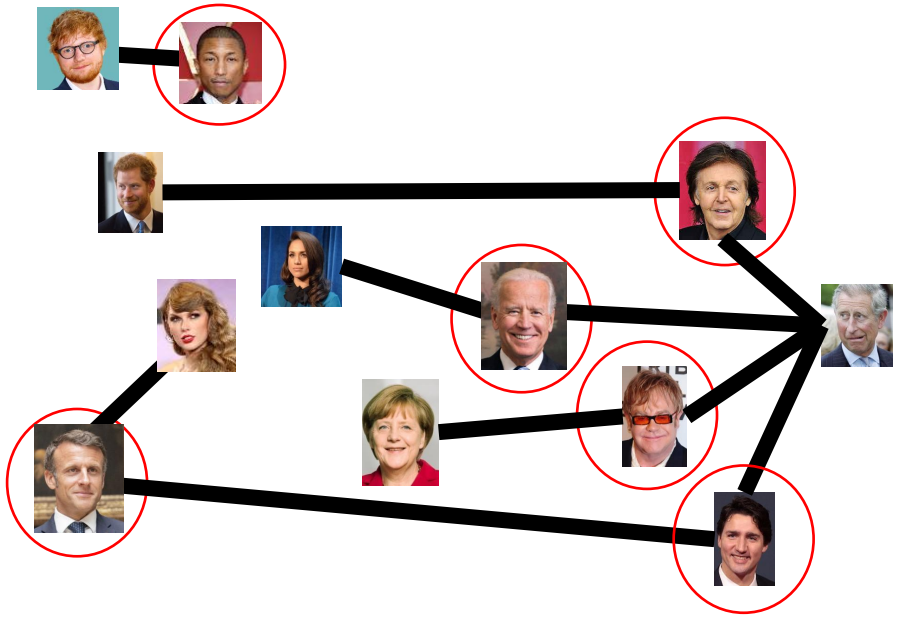
Transforming an IS into a VC

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

Independent Set



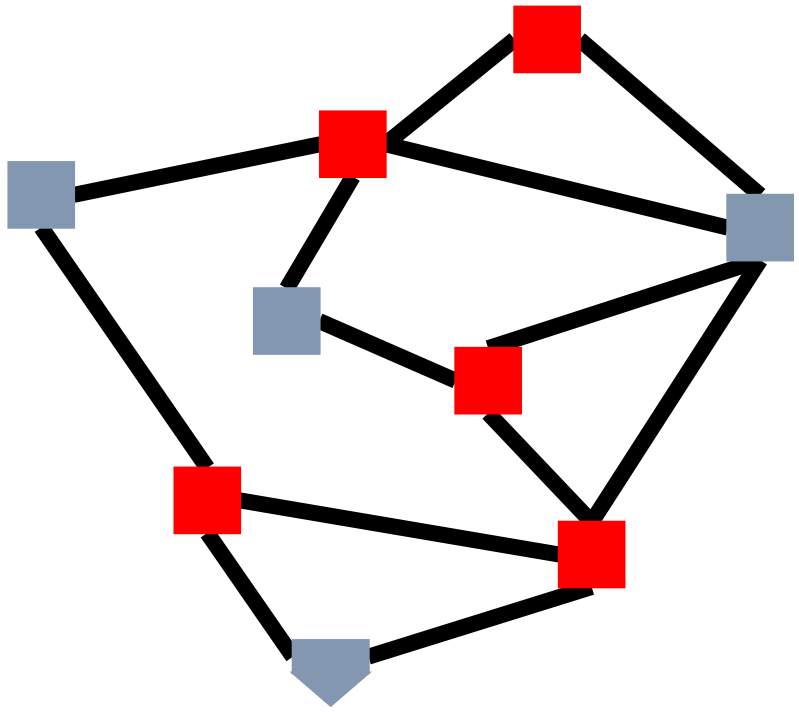
Vertex Cover



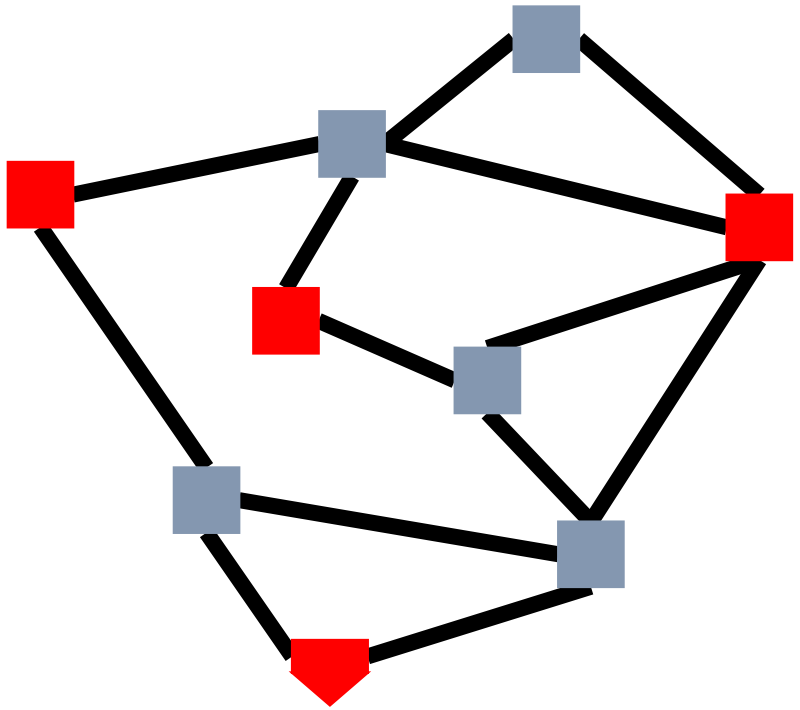
Transforming a VC into an IS

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!

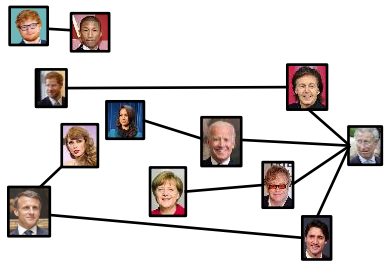
Reduction

- 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

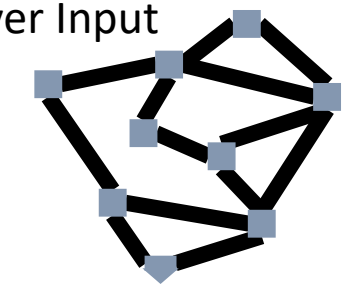


O(V) Time

Relate Independent Set input to Vertex Cover output

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

Vertex Cover Input



k

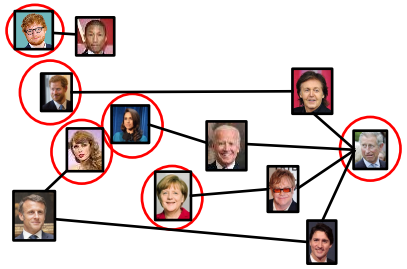
Any Algorithm for Vertex Cover

Relate Independent Set output to Vertex Cover output

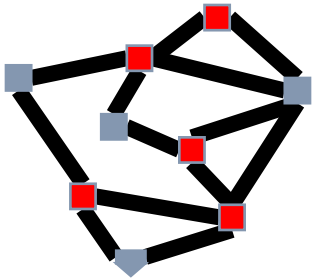
Give same output

Reduction

Independent Set Output



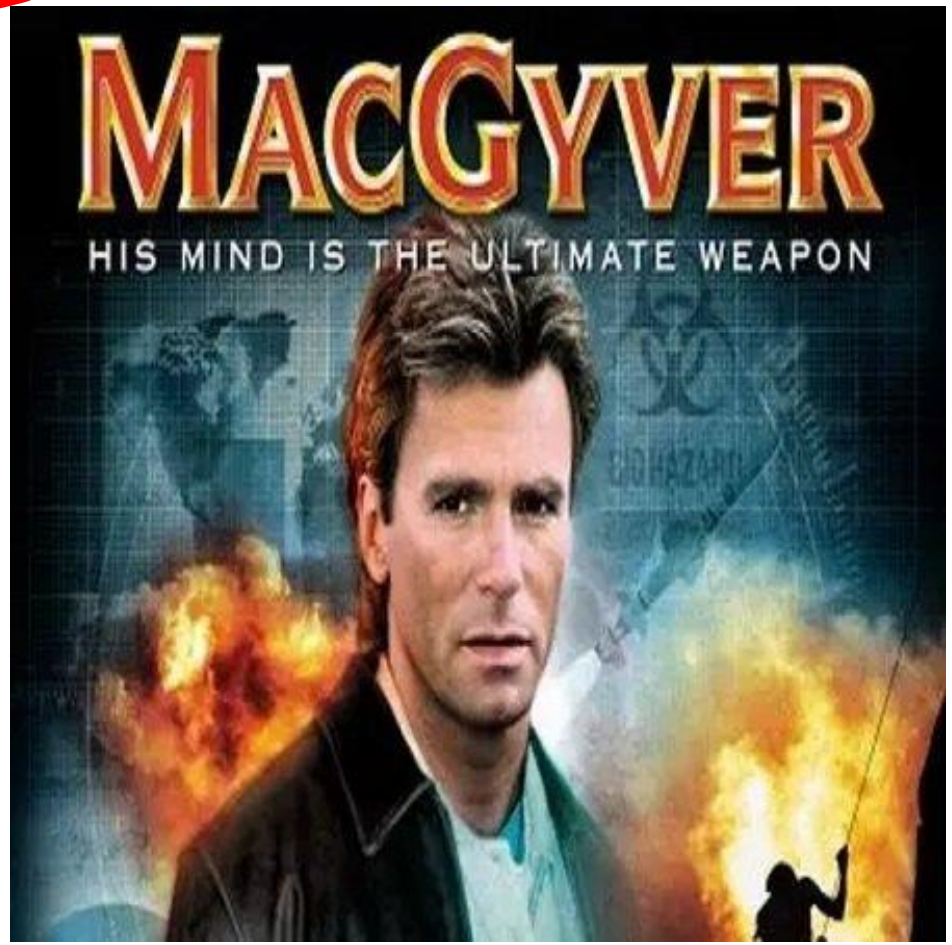
Vertex Cover Output



Reduction Analogy

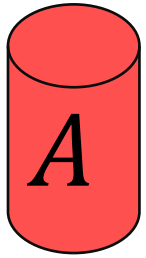
Shows how two different problems relate to each other

MOVIE TIME!



MacGyver's Reduction

Problem we don't know how to solve



Opening a door

Solution for A

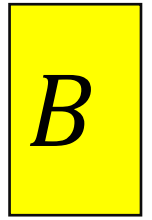
Keg cannon
battering ram

Aim duct at door, insert keg

Put fire under the Keg

Reduction

Problem we will solve instead



Lighting a fire

How?

Alcohol, wood, matches

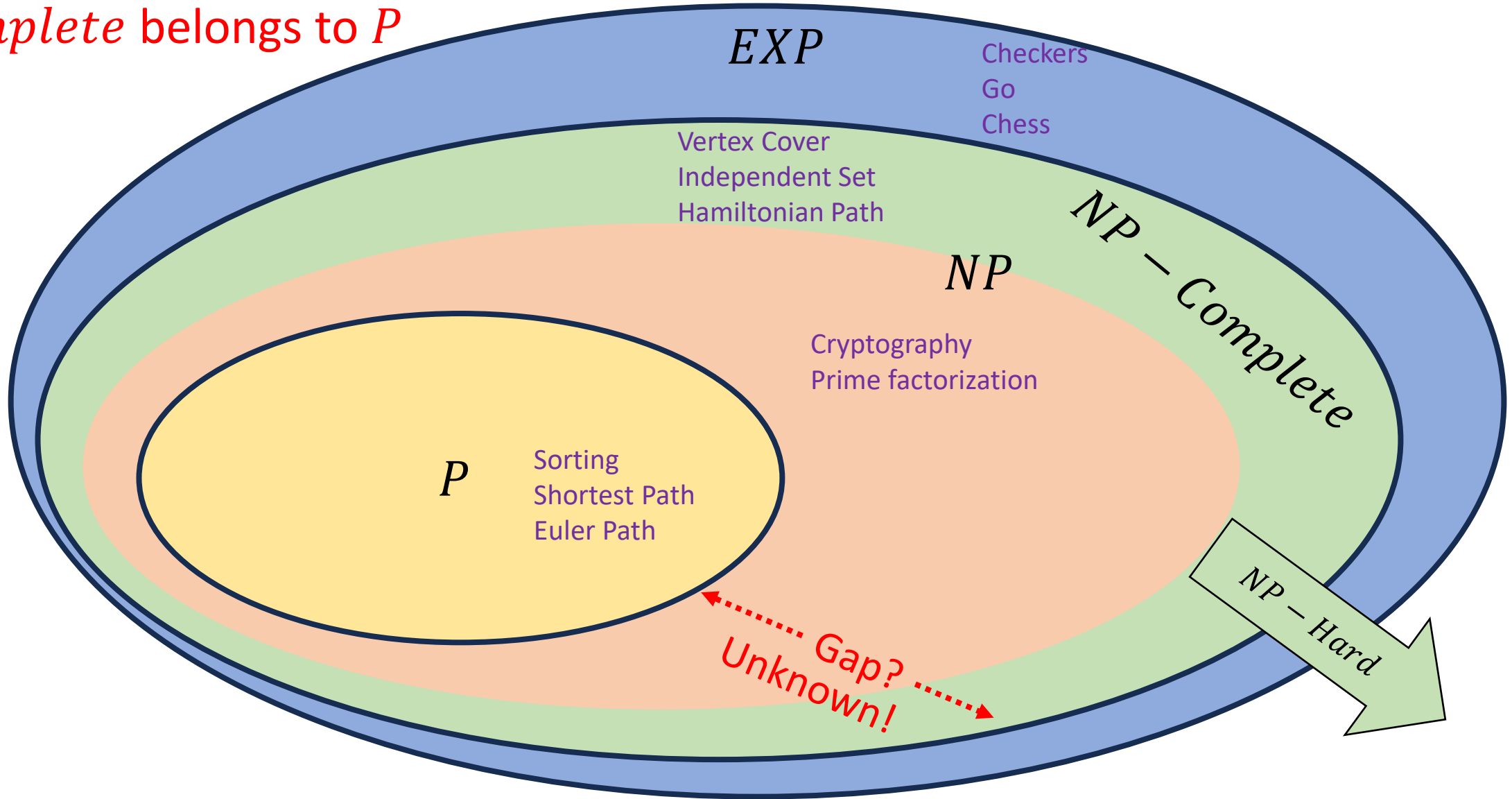
Solution for B

NP-Complete – High Level

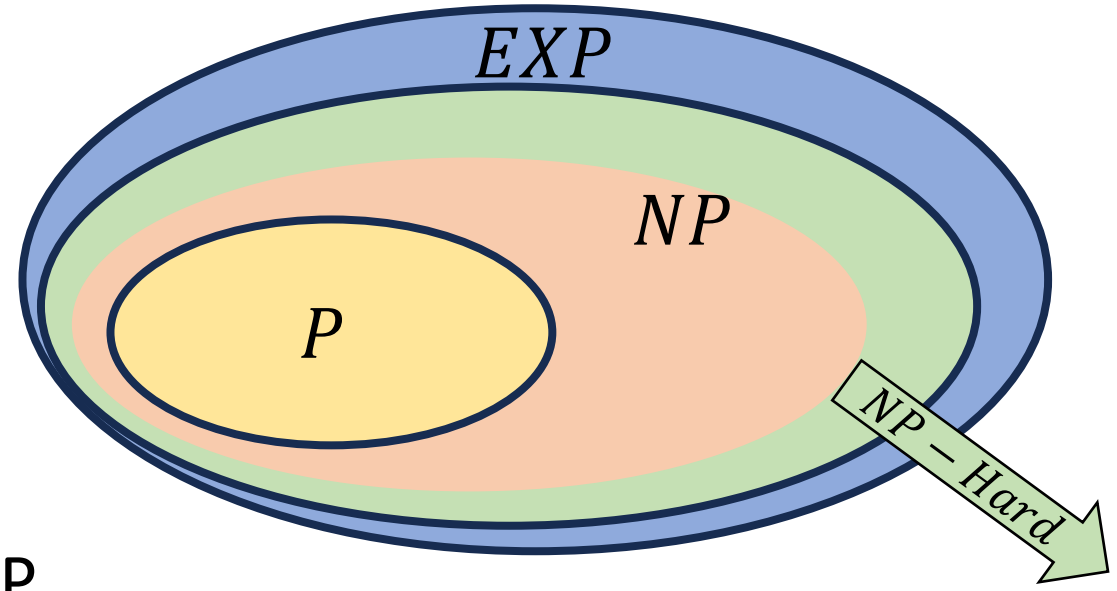
- A set of “together they stand, together they fall” problems
- The problems in this set either all belong to P, or none of them do
- Intuitively, the “hardest” problems in NP
- Collection of problems from NP that can all be “transformed” into each other in polynomial time
  - Like we could transform 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

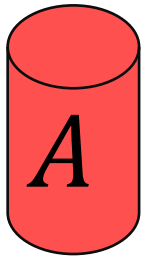


- How can we try to figure out if P=NP?
- Identify problems at least as “hard” as NP
  - If any of these “hard” problems can be solved in polynomial time, then all NP problems can be solved in polynomial time.
- Definition: NP-Hard:
  - B is NP-Hard provided EVERY problem within NP reduces to B in polynomial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

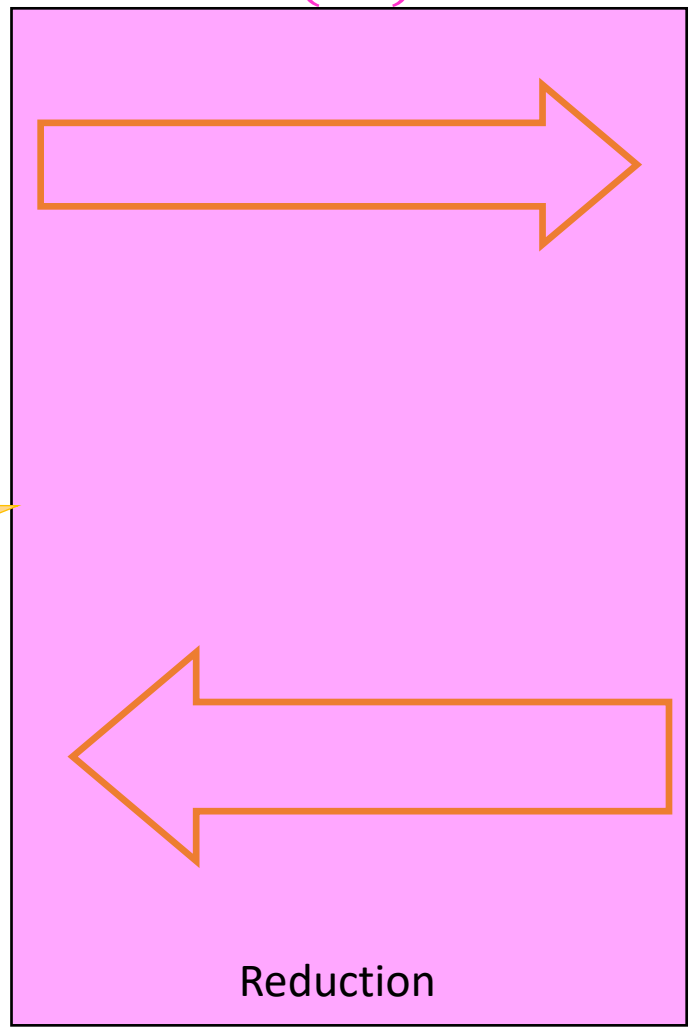
NP-Hard Idea

For every NP problem

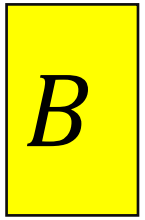
Any NP Problem



$O(n^p)$



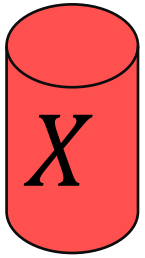
An NP-Hard Problem



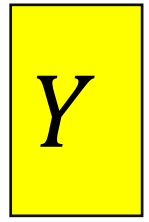
So if this was $O(n^p)$ we can solve any NP problem in polynomial time

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

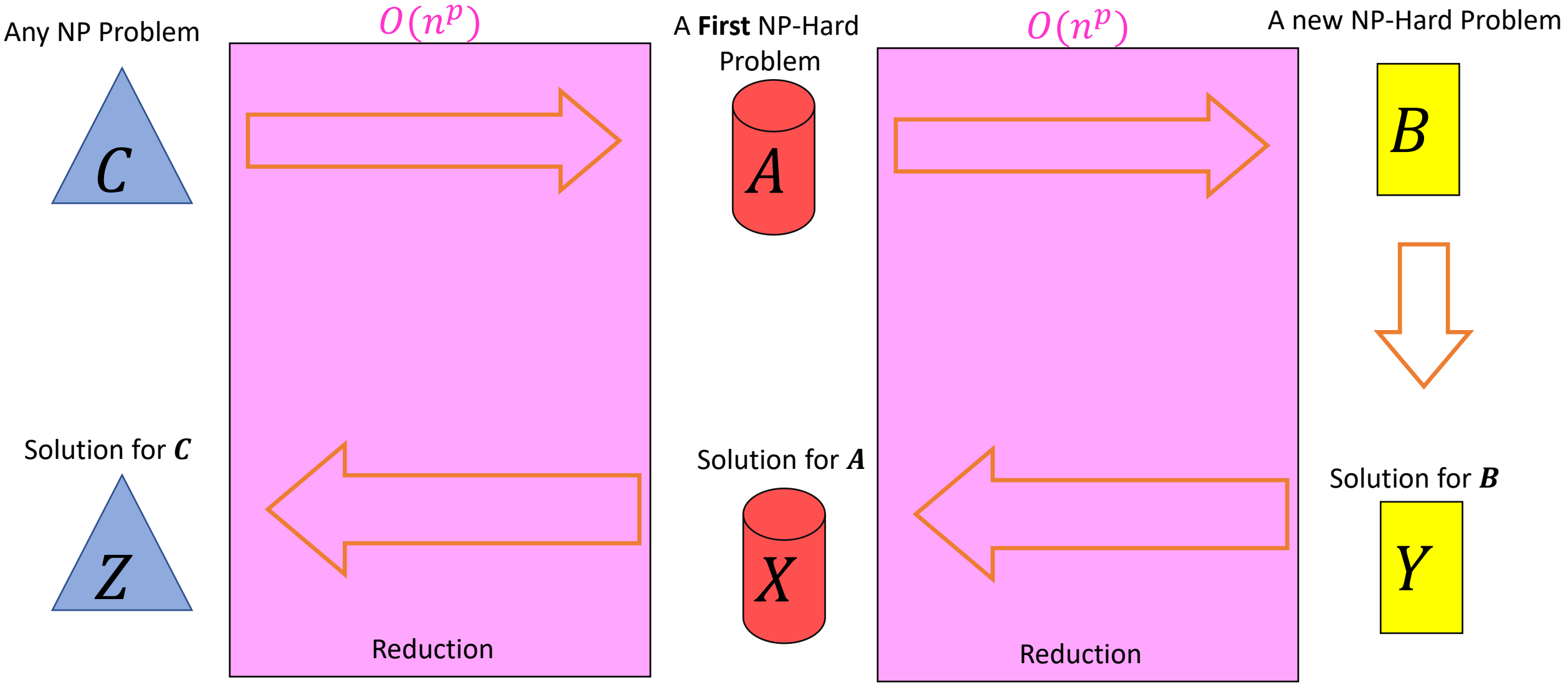
Solution for $\boldsymbol{A}$



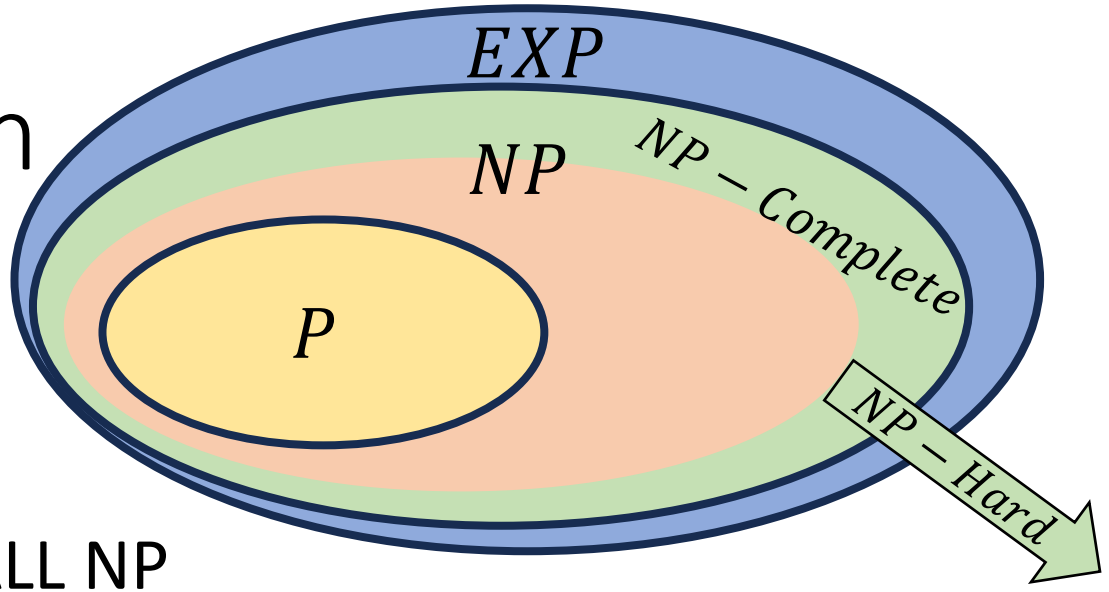
Solution for $\boldsymbol{B}$



Showing NP-Hardness

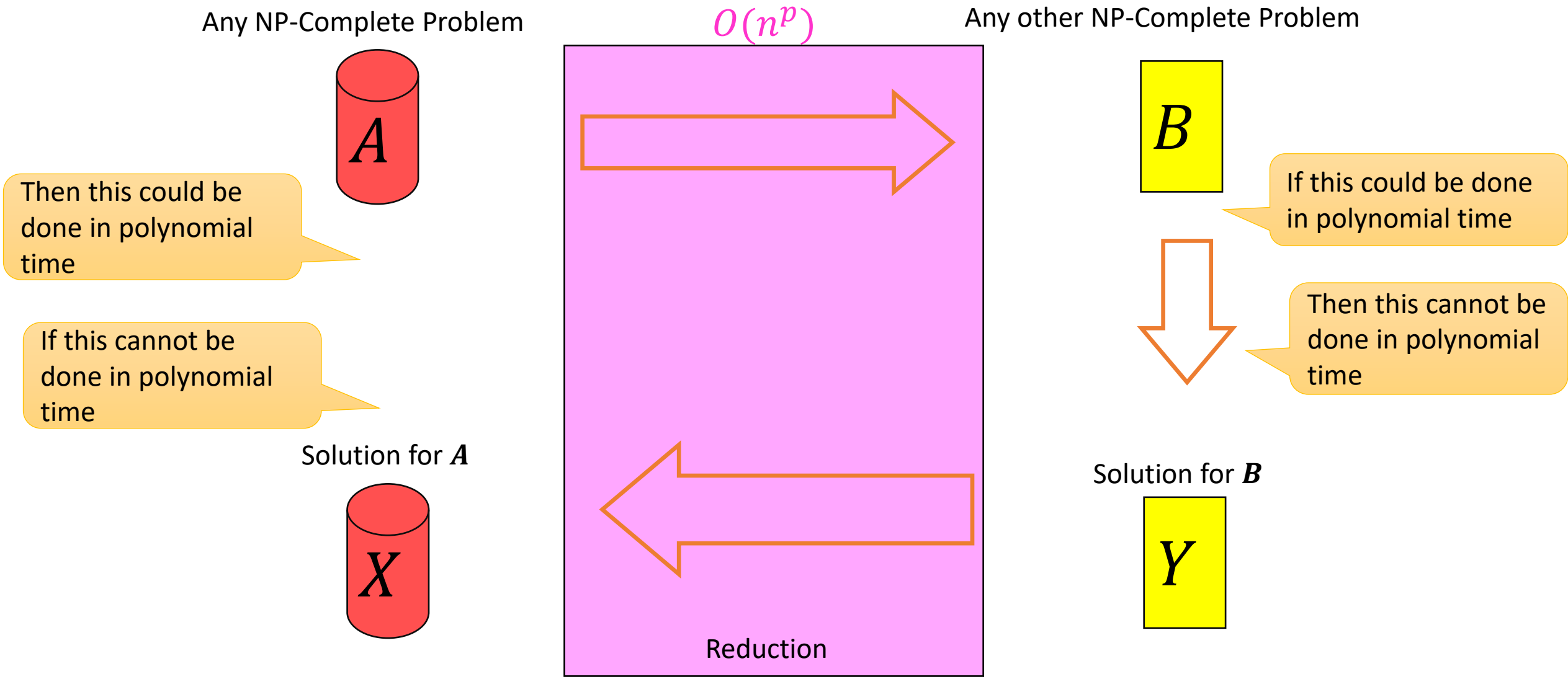


NP-Complete - Definition



- “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

Using 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

Public Key Crypto Requires $P \neq NP$

- Public key cryptography involves two keys:
  - A "public key" k_{pub} used for encryption
    - To encrypt m we compute $E(m, k_{pub}) = c$
  - A "private key" k_{priv} used for decryption
    - To decrypt c we compute $D(c, k_{priv}) = m$
- To be secure we need:
  - $E(m, k_{pub})$ to belong to P
  - $D(c, k_{pub})$ to not belong to P
  - However, $D(c, k_{pub})$ must belong to NP
    - Given c, k_{pub}, m we can verify if $D(c, k_{pub}) = m$ by checking if $E(m, k_{pub}) = c$

Does P=NP?

Poll Year	$P \neq NP$	$P = NP$	Unprovable	Don't Care	Don't Know	Don't Know & Don't Care	Other
2002	61%	9%	4%	1%	22%	0%	3%
2012	83%	9%	3%	3%	0.66%	0.66%	0.66%
2019	88%	12%	--	--	--	--	--

https://www.cs.umd.edu/users/gasarch/BLOGPAPERS/pollpaper3.pdf