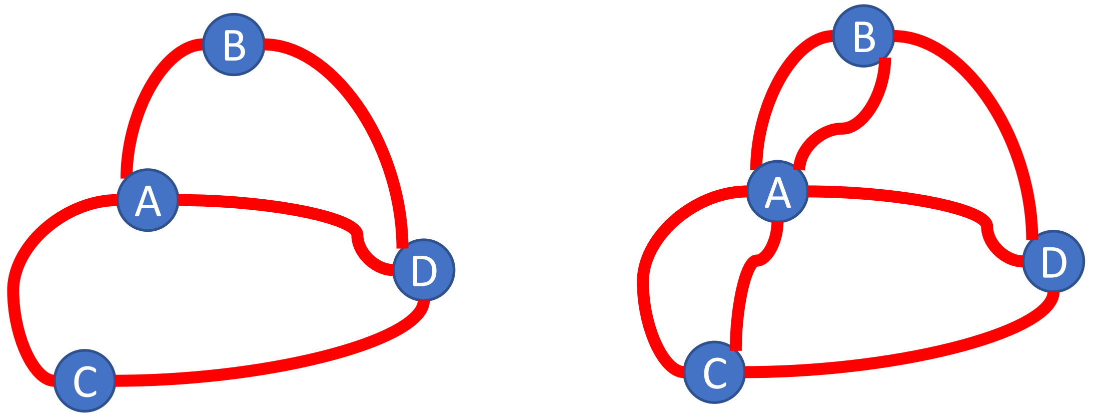
CSE 332 Autumn 2023
Lecture 29: P and NP

Nathan Brunelle

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

Euler Path Problem



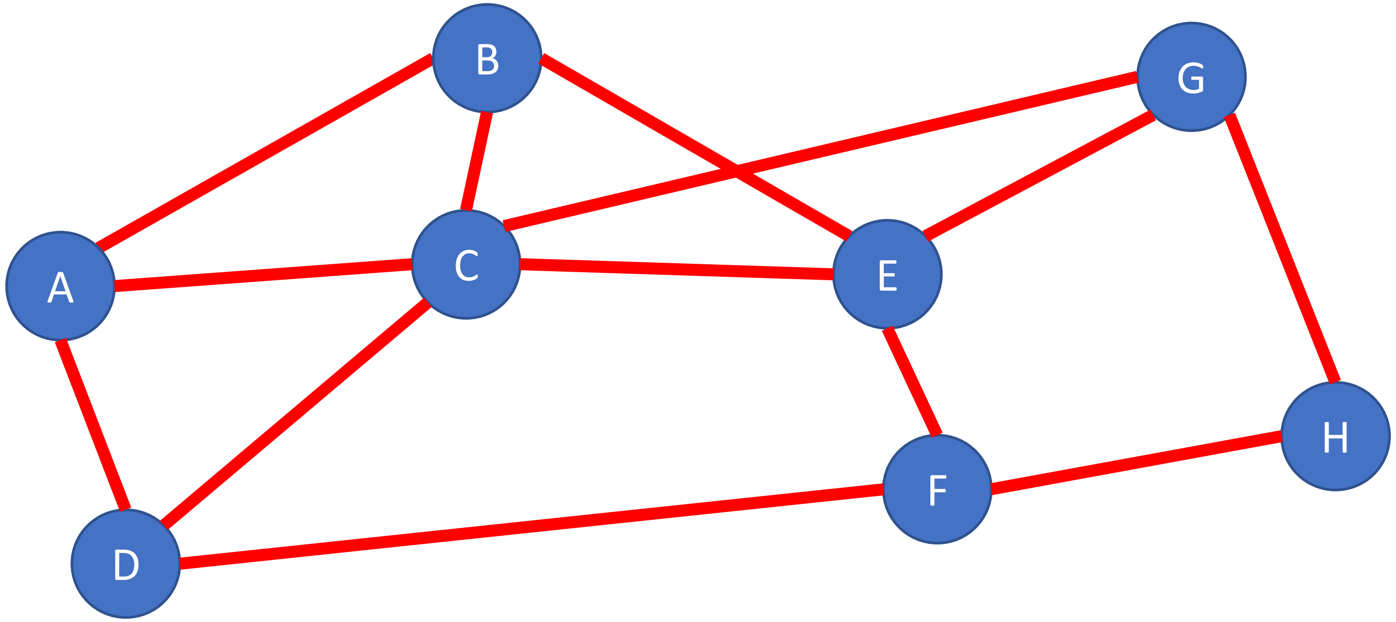
- Path:
  - A sequence of nodes $v_1, v_2, \ldots$ such that for every consecutive pair are connected by an edge (i.e. (v_i, v_{i+1}) is an edge for each i in the path)
- Euler Path:
  - A path such that every edge in the graph appears exactly once
    - If the graph is not simple then some pairs need to appear multiple times!
- Euler path problem:
  - Given an undirected graph $G = (V, E)$, does there exist an Euler path for G?

Algorithm for the Euler Path Problem

- Given an undirected graph $G = (V, E)$, does there exist an Euler path for G?
- Algorithm:
  - Check if the graph is connected
  - Check the degree of each node
  - If the number of nodes with odd degree is 0 or 2, return true
  - Otherwise return false
- Running time?
  - $O(V + E)$

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

Algorithms for the Hamiltonian Path Problem

- Option 1:
  - Explore all possible simple paths through the graph
  - Check to see if any of those are length V
  - Running time: $O(V!)$
- Option 2:
  - Write down every sequence of nodes
  - Check to see if any of those are a path
  - $O(V!)$
- Both options are examples of an **Exhaustive Search ("Brute Force") algorithm**

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

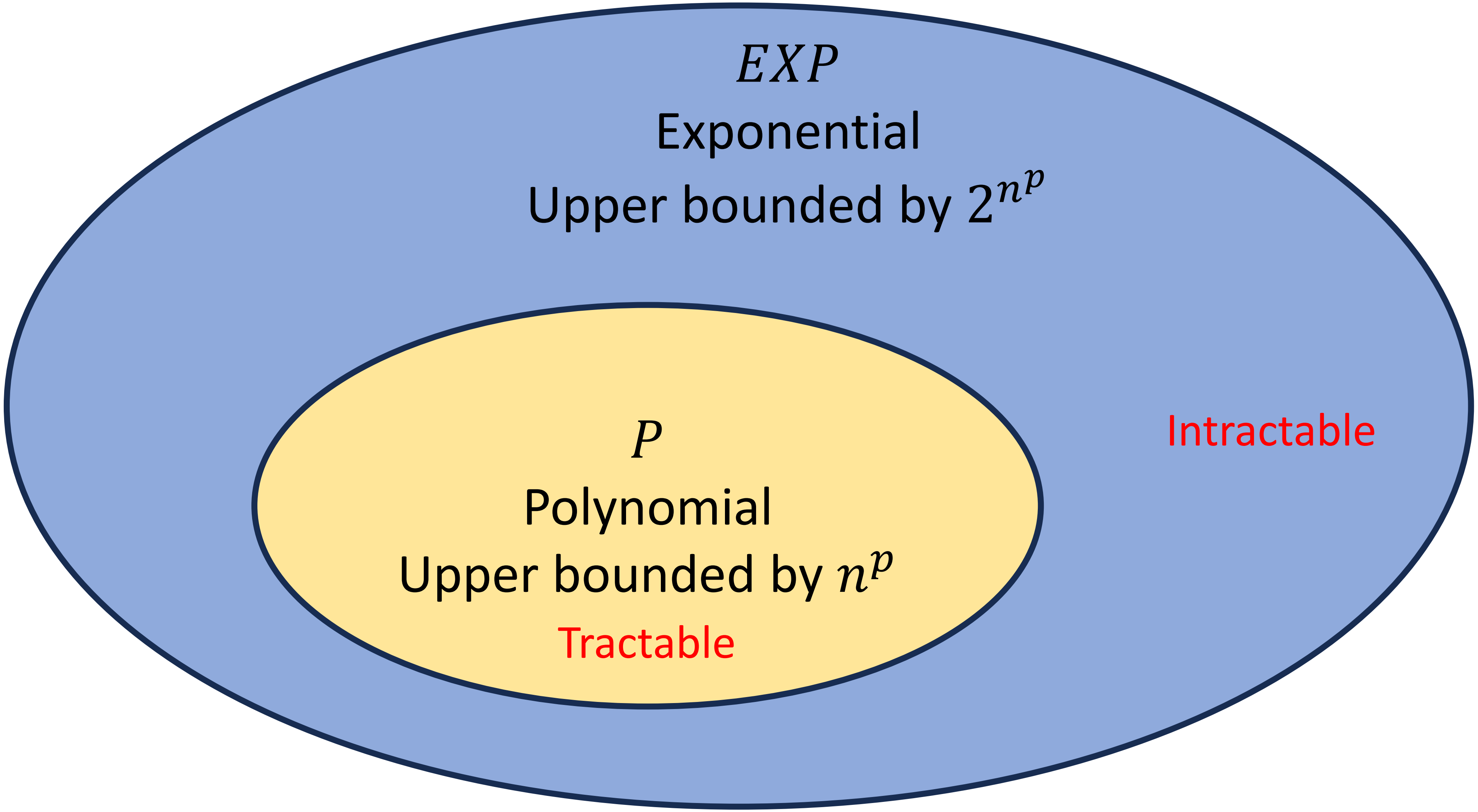
Running Times

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

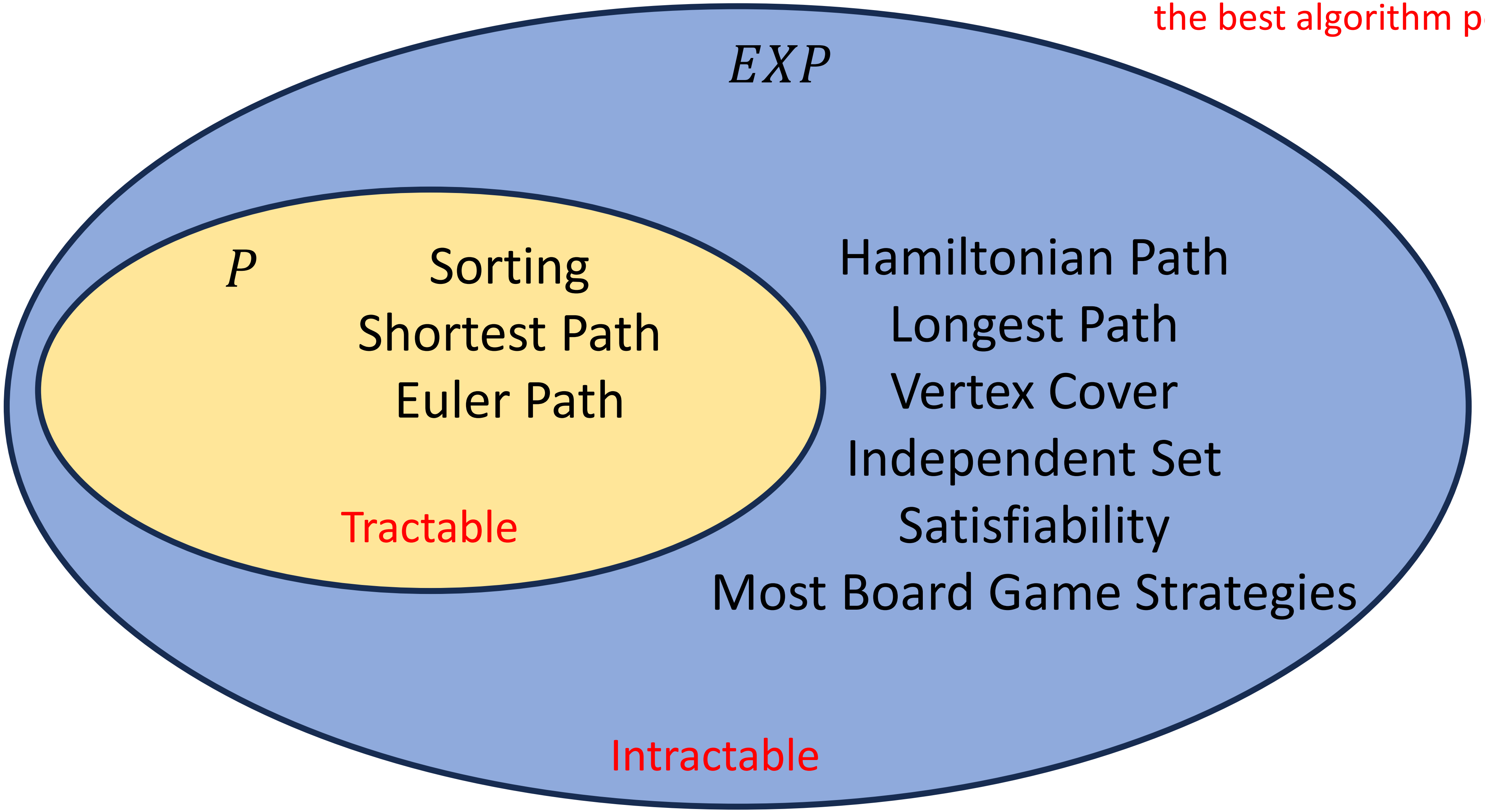
EXP and P

Important!
$P \subset EXP$
Every problem within P is also within EXP
The intractable ones are the problems within EXP but NOT P



Members

Important!
Some of the problems listed in EXP could also be members of P
Since membership is determined by a problems 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"

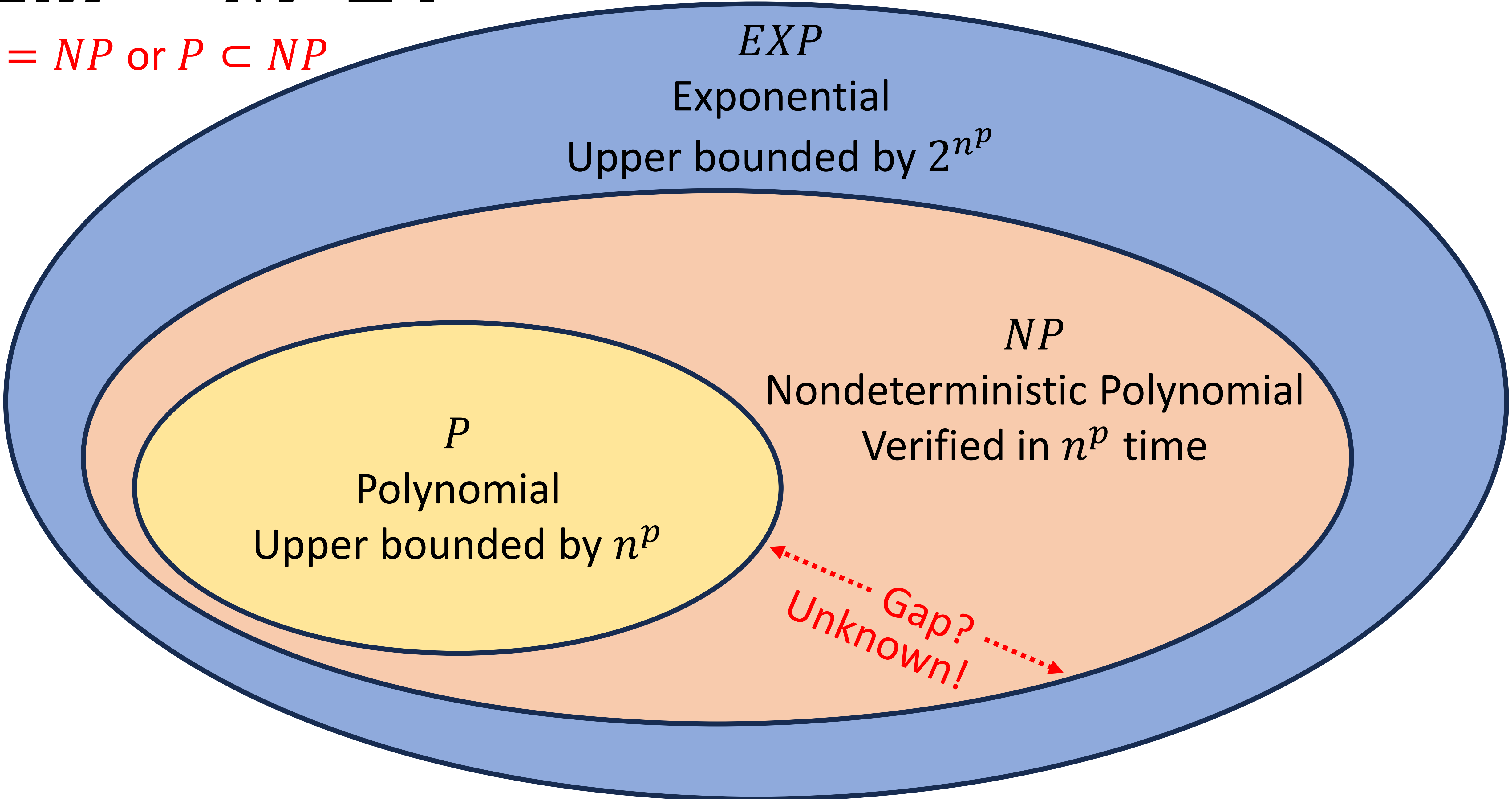
- There are some problems that we do not have polynomial time algorithms to solve, but provided answers are easy to check
- Hamiltonian Path:
  - It's "hard" to look at a graph and determine whether it has a Hamiltonian Path
  - It's "easy" to look at a graph and a candidate path together and determine 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 (or other algorithm)
- $P \subseteq NP$
  - Why?

$EXP \supset NP \supseteq P$

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

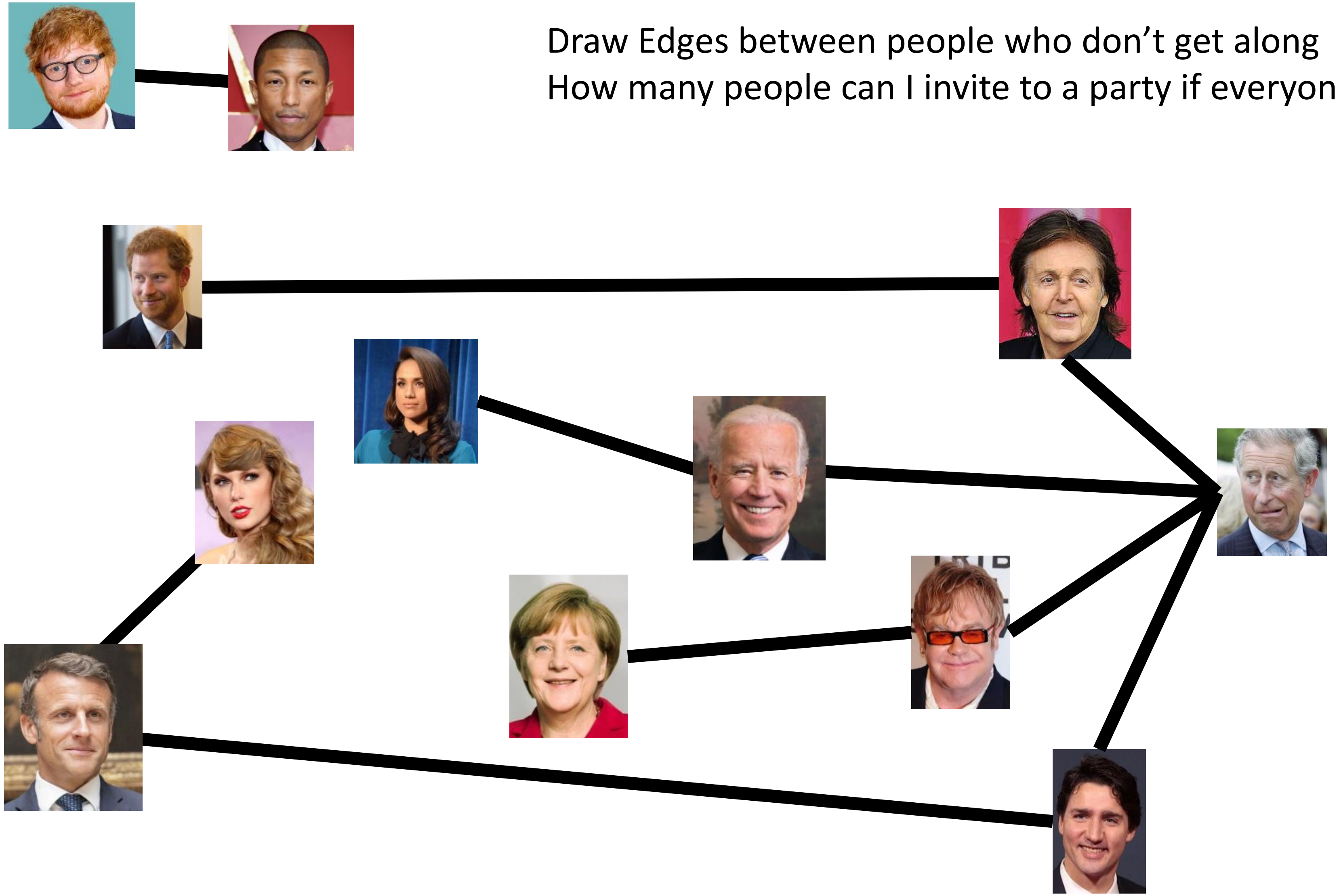


Solving and Verifying Hamiltonian Path

- 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
- 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

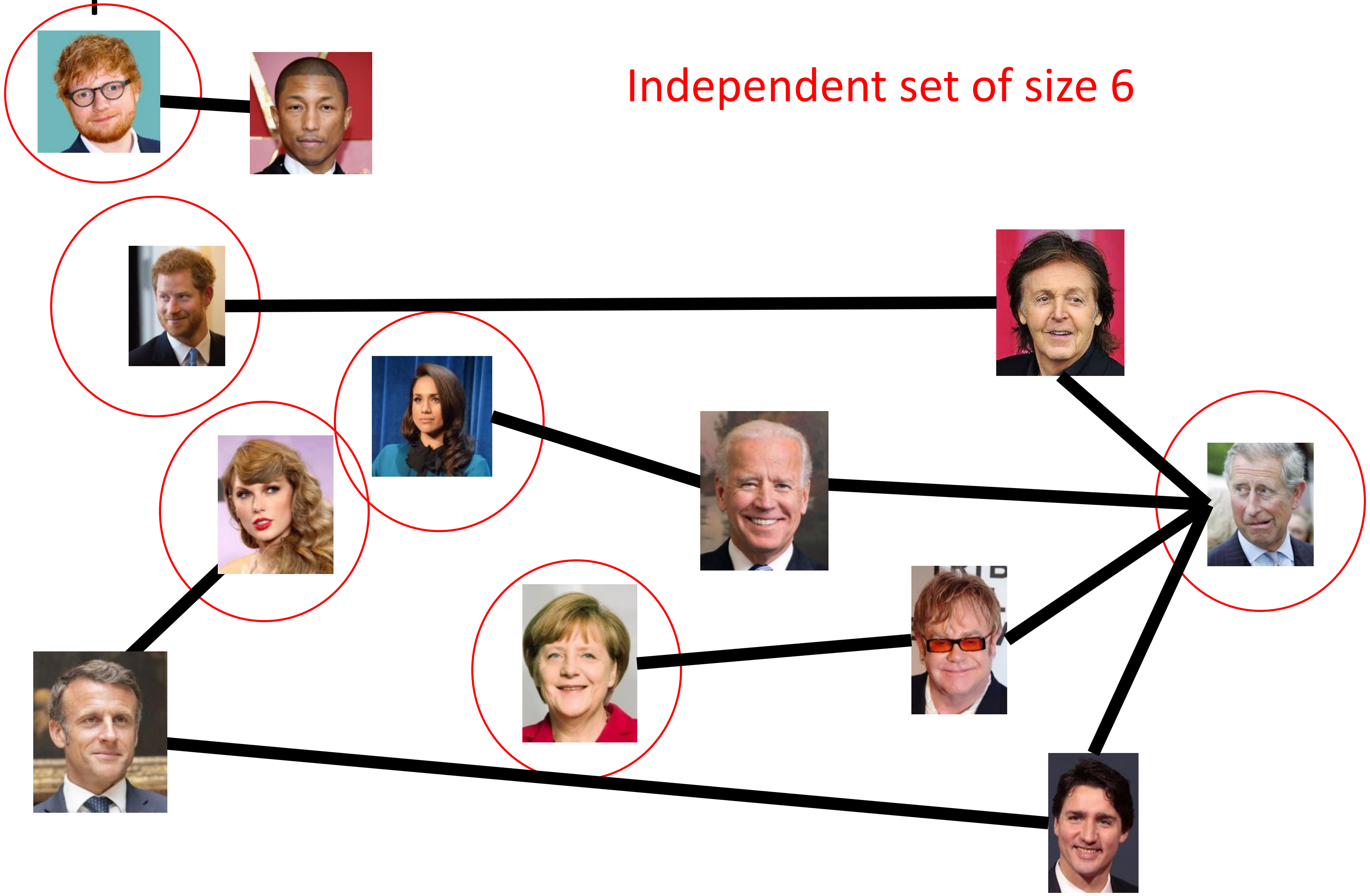
Draw Edges between people who don't get along
How many people can I invite to a party if everyone must get along?



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

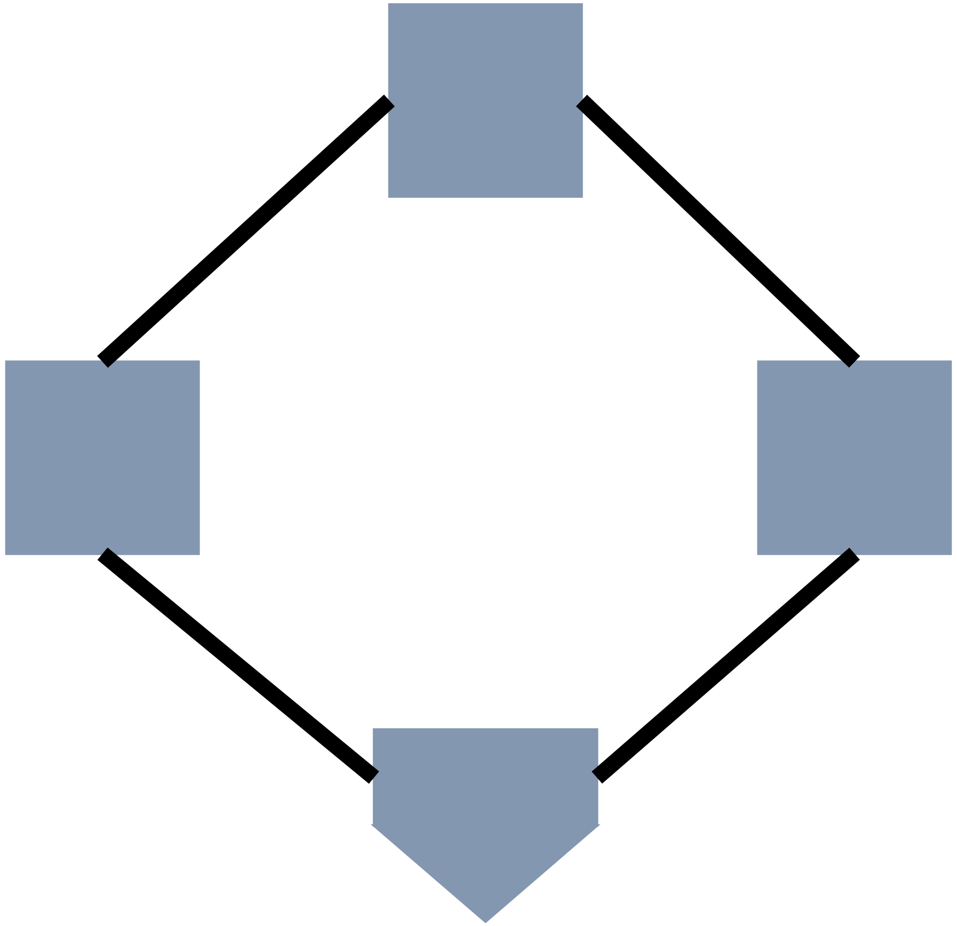
Example



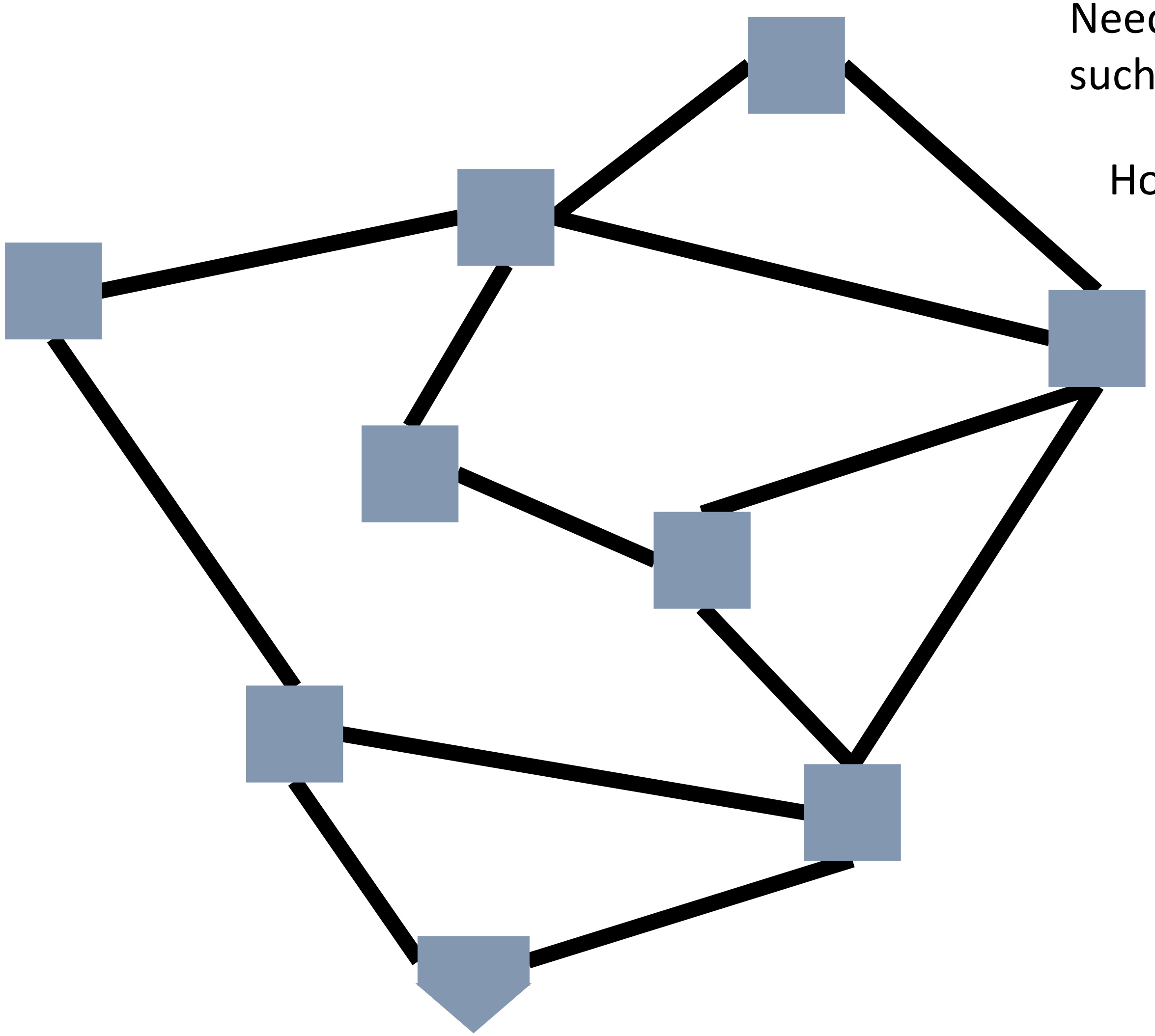
Solving and Verifying Independent Set

- Algorithm to solve independent set
  - Input: $G = (V, E)$ and a number k
  - Output: True if G has an independent set of size k
    - List every subset of V that has size k
      - $\approx |V|^{|V|-k}$
    - For each of the subsets, check whether any pair of nodes are adjacent
      - $k \cdot |E|$
- 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

Generalized Baseball



Generalized Baseball



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

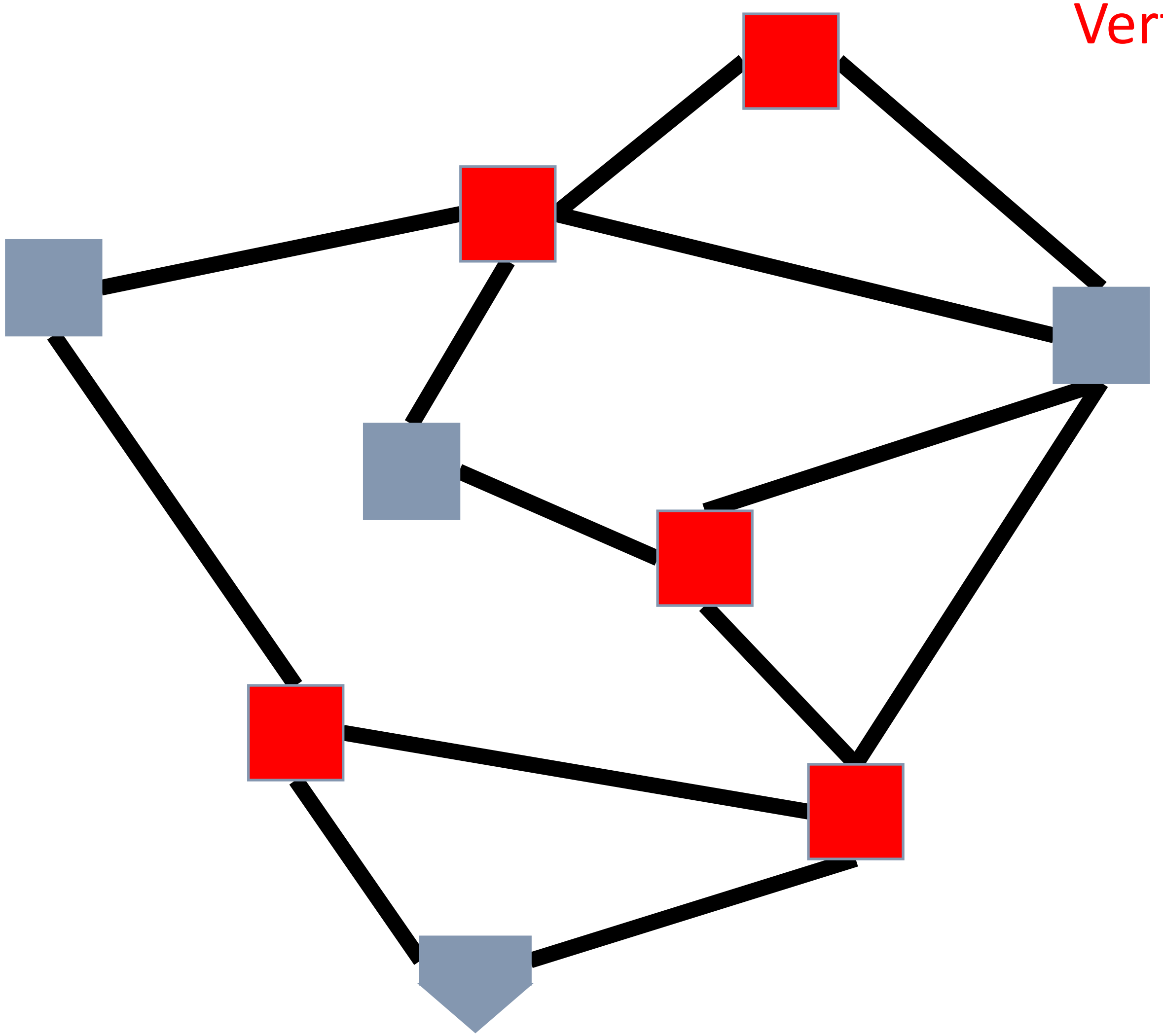
How many defenders would suffice?

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

Example

Vertex cover of size 5



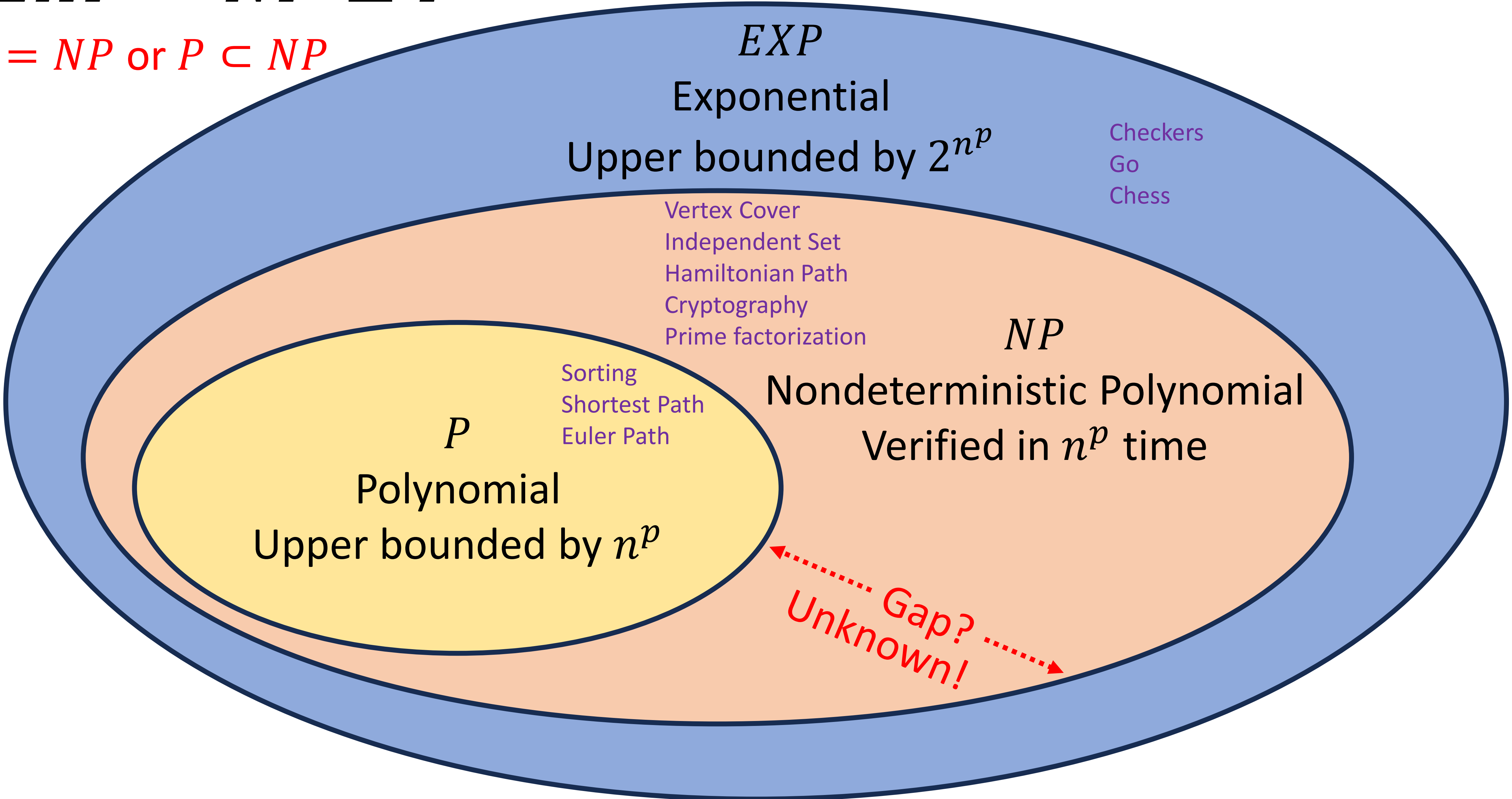
Solving and Verifying Vertex Cover

- Algorithm to solve vertex cover
  - Input: $G = (V, E)$ and a number k
  - Output: True if G has a vertex cover of size k

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

$EXP \supset NP \supseteq P$

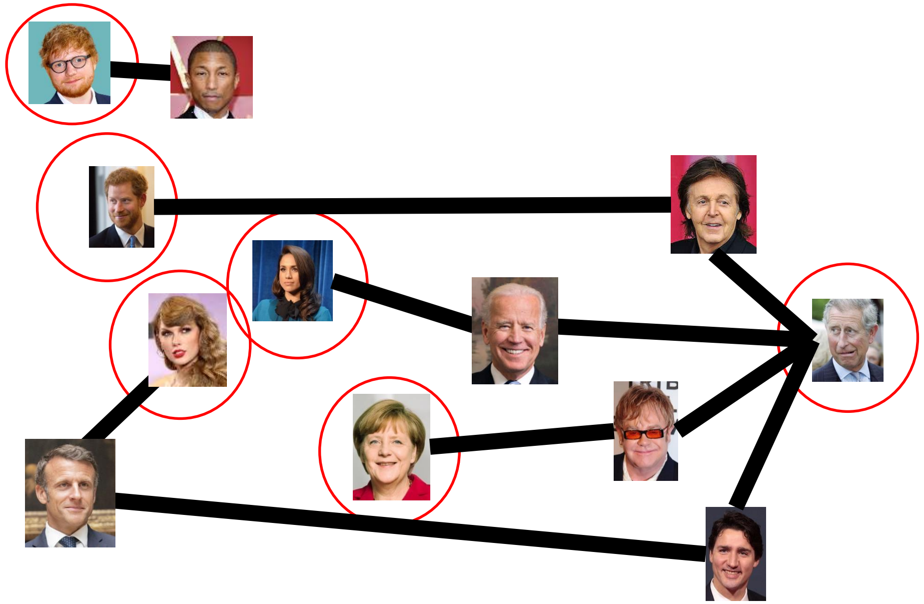
$P = NP$ or $P \subset NP$



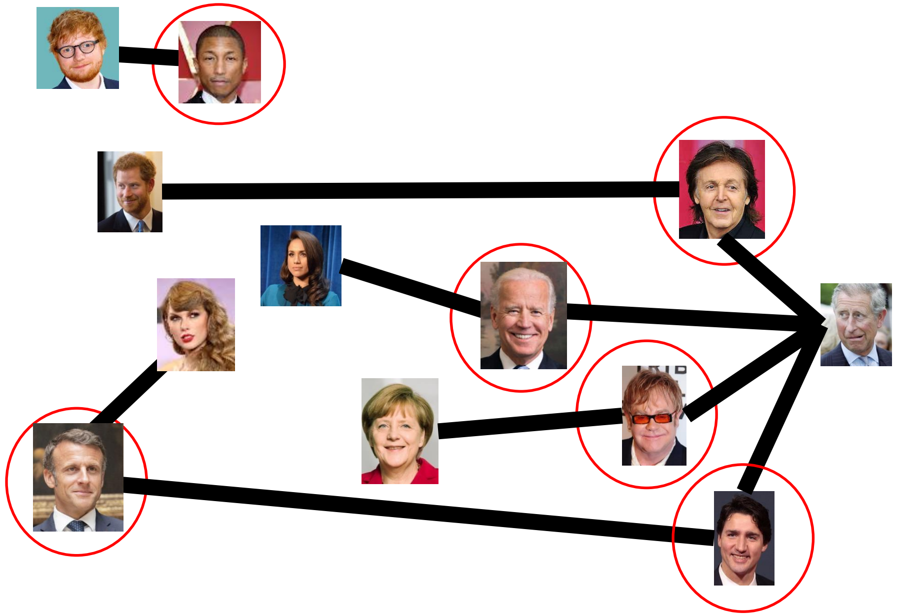
Way Cool!

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

Independent Set



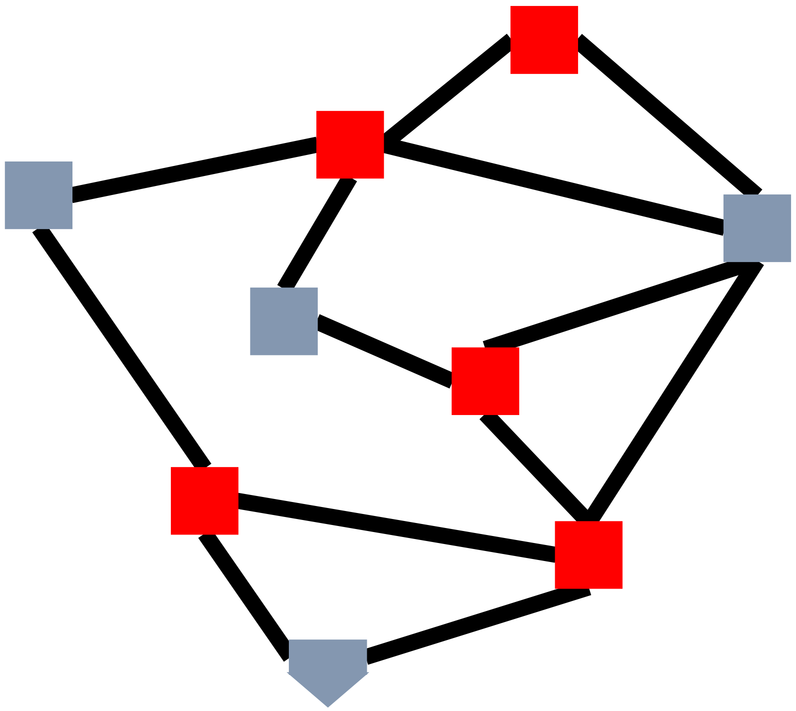
Vertex Cover



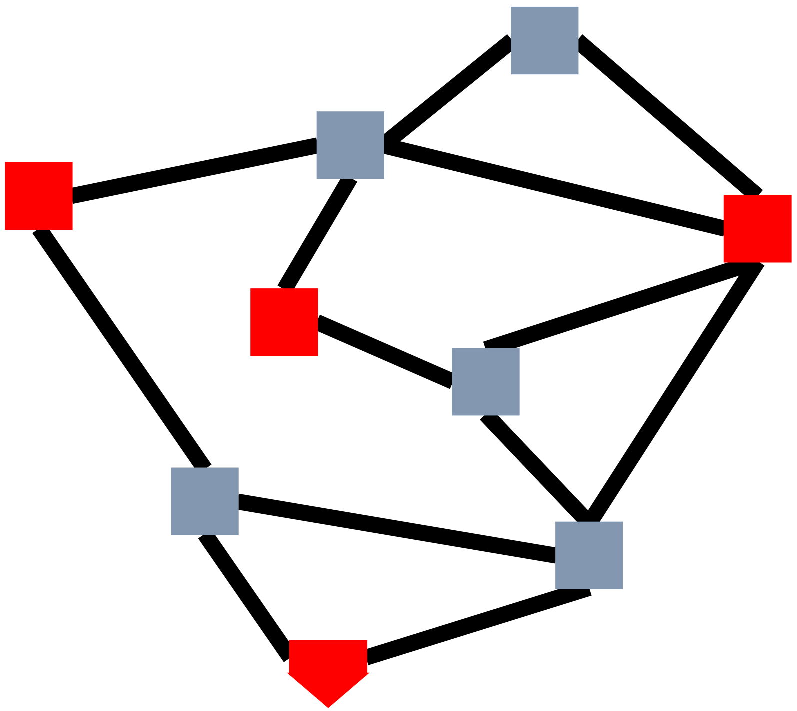
Way Cool!

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 $|V| - k$

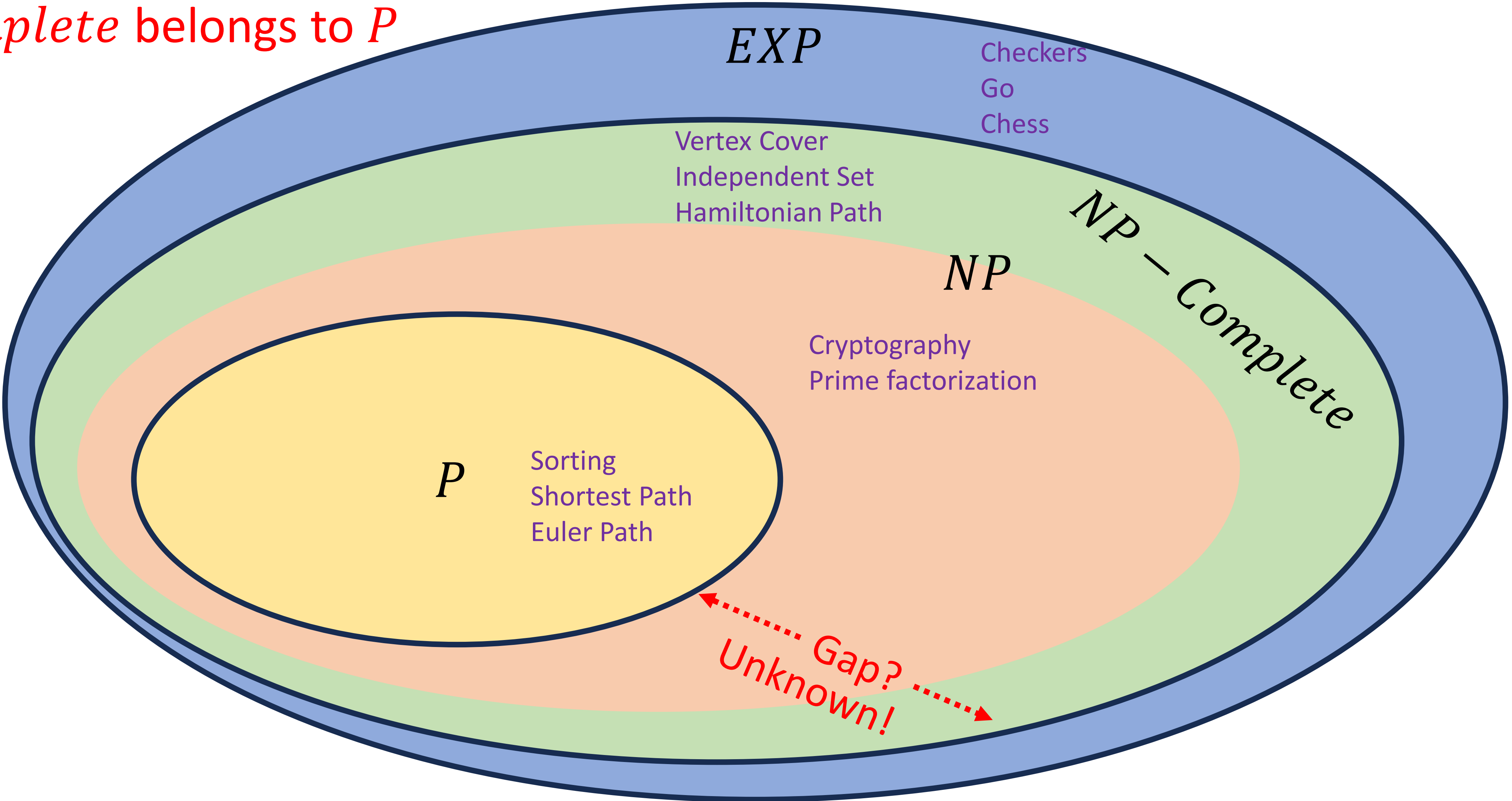
Either both problems belong to P, or else neither does!

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 …

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

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



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 $NP - Complete$
  - Therefore if any one member of $NP - Complete$ belongs to P, then $P = NP$

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