

CS4102 Algorithms

Spring 2022

Warm up:

Show that $P = NP$

Today's Keywords

- Reductions
- P vs NP
- NP Hard, NP Completeness
- k-Independent Set
- k-Vertex Cover
- 3SAT
- k-Clique

- CLRS: Ch 34

Homeworks

- Unit C and D Programming Due 5/3
- Unit D Advanced Due 5/3
 - NP Completeness and Reductions
- Unit D Basic Due 5/3 (but no penalty submission through 5/6)

Final Exam

- Tuesday, May 10, 7pm in MEC 205 (our section)

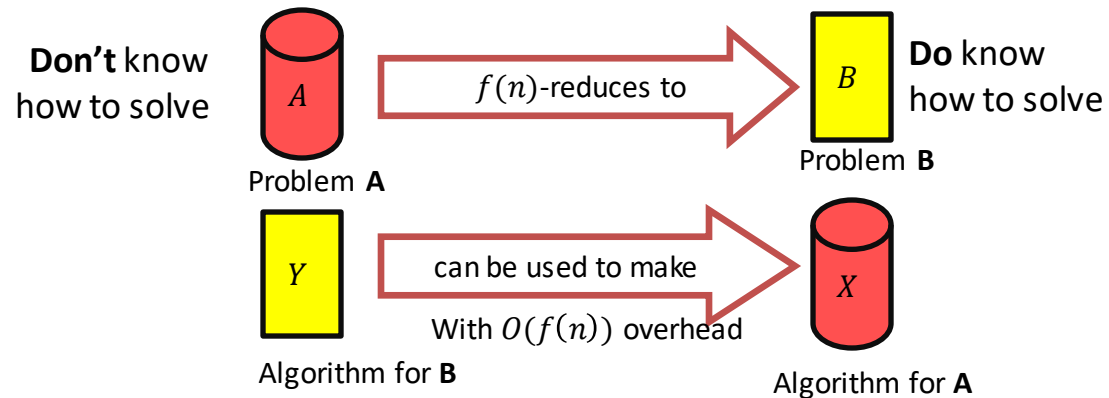
Reductions

- Algorithm technique of supreme ultimate power
- Convert instance of problem A to an instance of Problem B
- Convert solution of problem B back to a solution of problem A

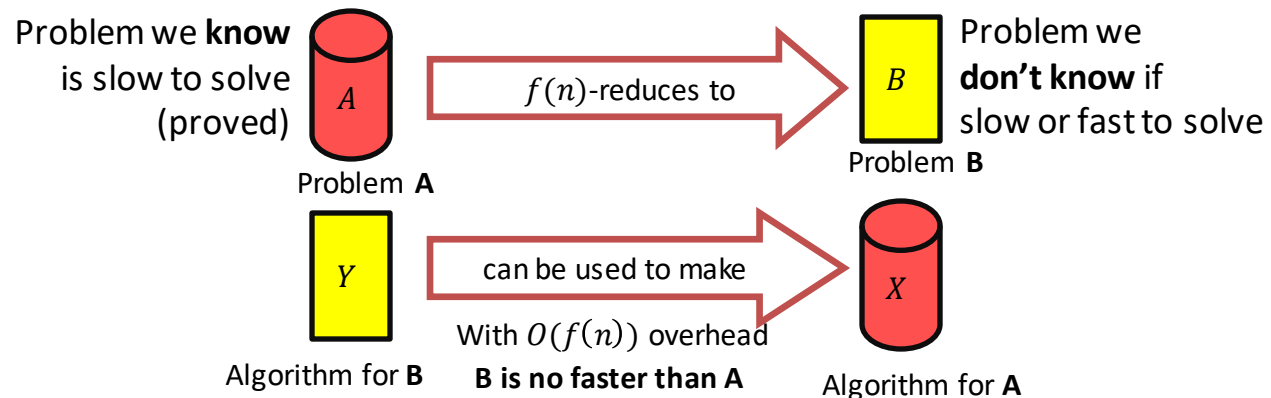
Reductions

Possible uses

- Use solver for B to solve A

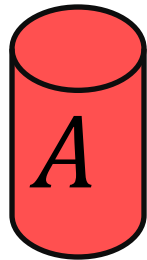


- Prove lower bound for B by showing it's as hard as A



MacGyver's Reduction

Problem we don't know how to solve



Opening a door



Solution for *A*

Keg cannon
battering ram



Problem we do know how to solve



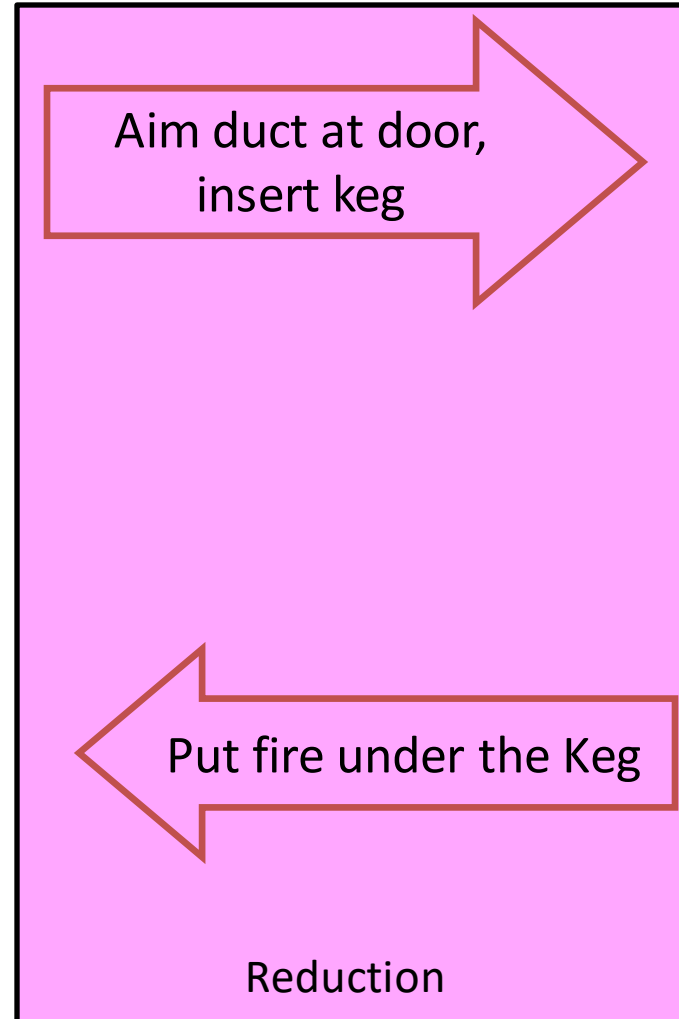
Lighting a fire



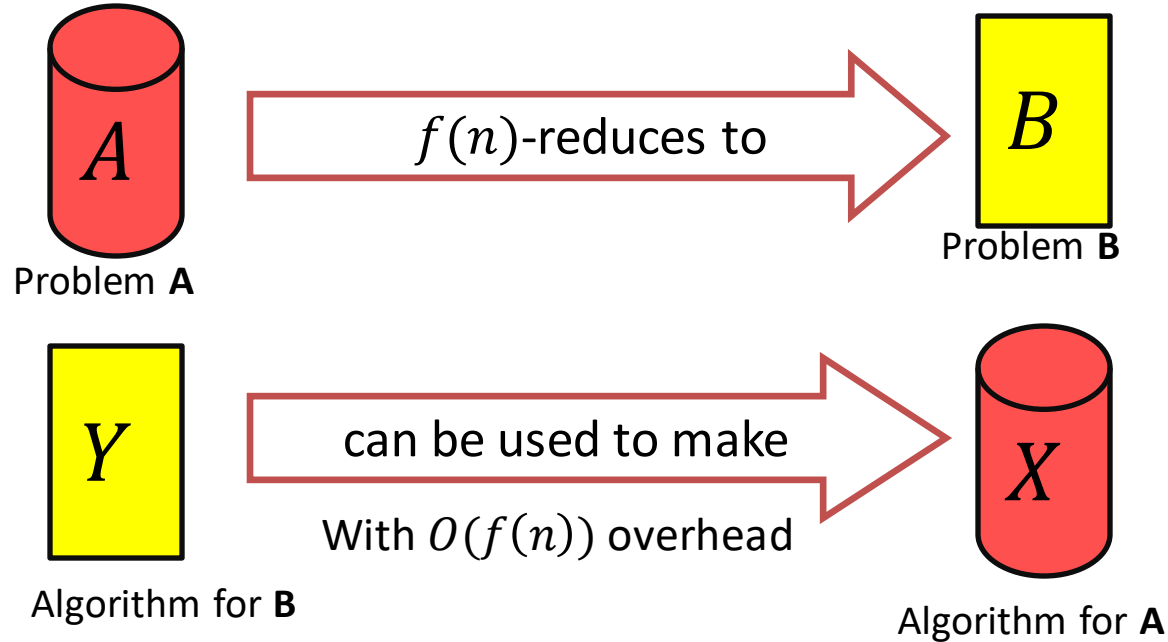
HOW?

Solution for *B*

Alcohol, wood,
matches



Reduction Proof Notation



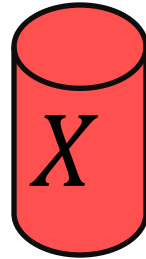
A is not a harder problem than B
 $A \leq B$

If A requires time $\Omega(f(n))$ time then B also requires $\Omega(f(n))$ time
 $A \leq_{f(n)} B$

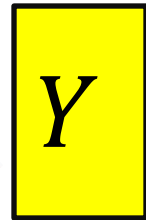
Or we could have solved A faster using B 's solver!

Proof of Lower Bound by Reduction

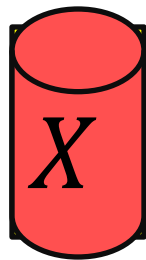
To Show: Y is slow



1. We know X is slow (by a proof)
(e.g., X = some way to open the door)



2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)



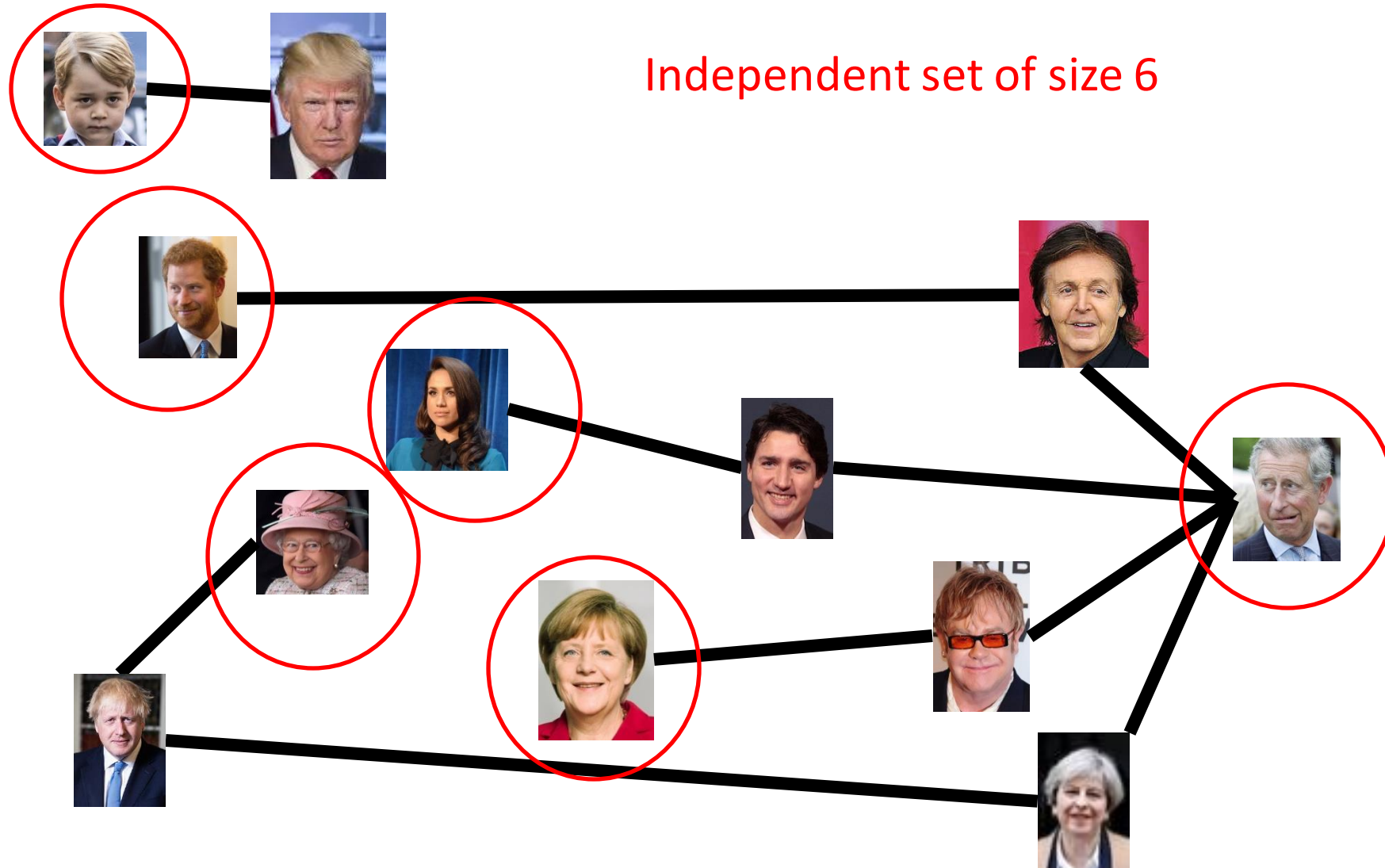
3. Show how to use Y to perform X quickly

4. X is slow, but Y could be used to perform X quickly
conclusion: Y must not actually be quick

Maximum Independent Set

- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

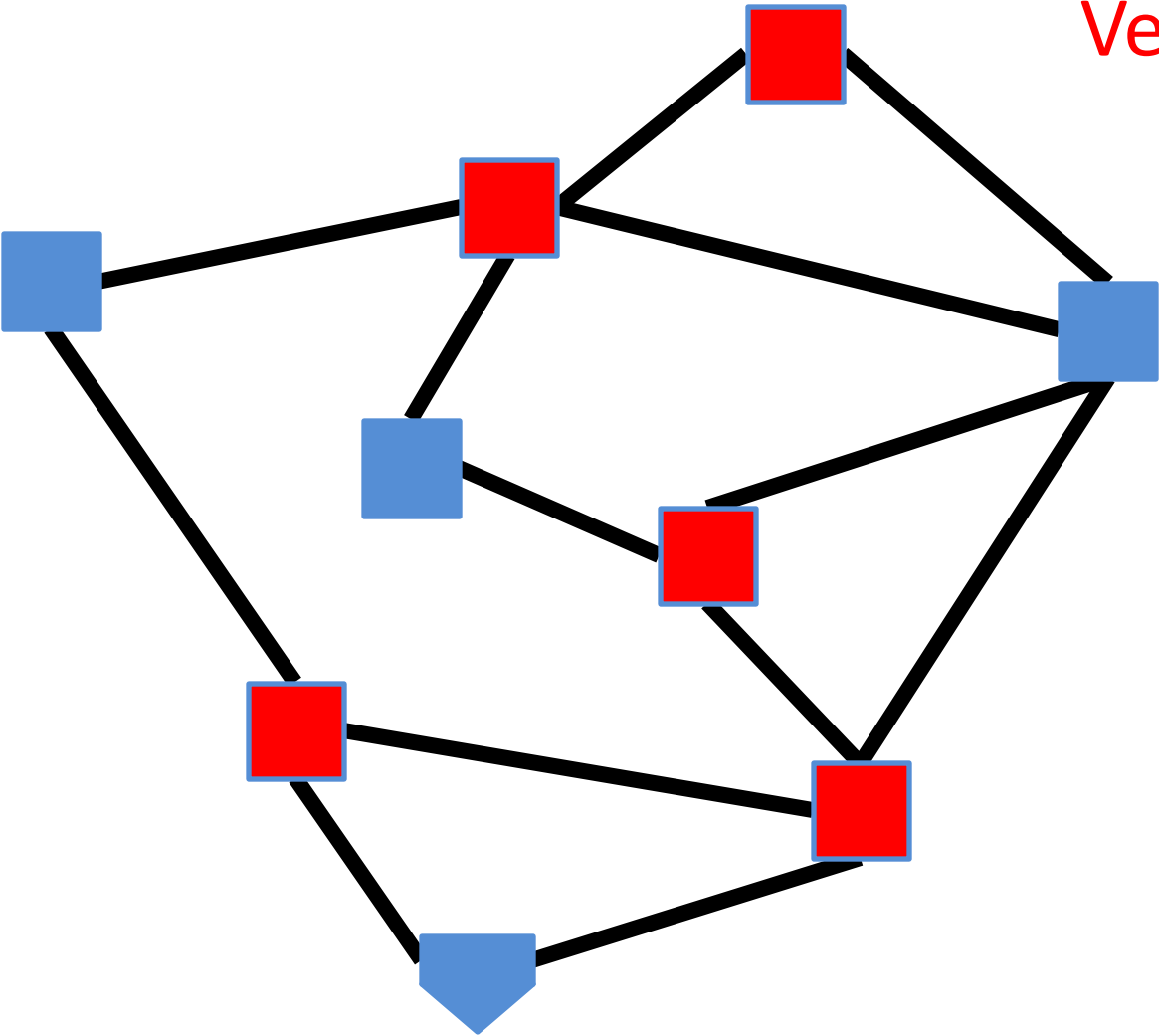
Example



Minimum Vertex Cover

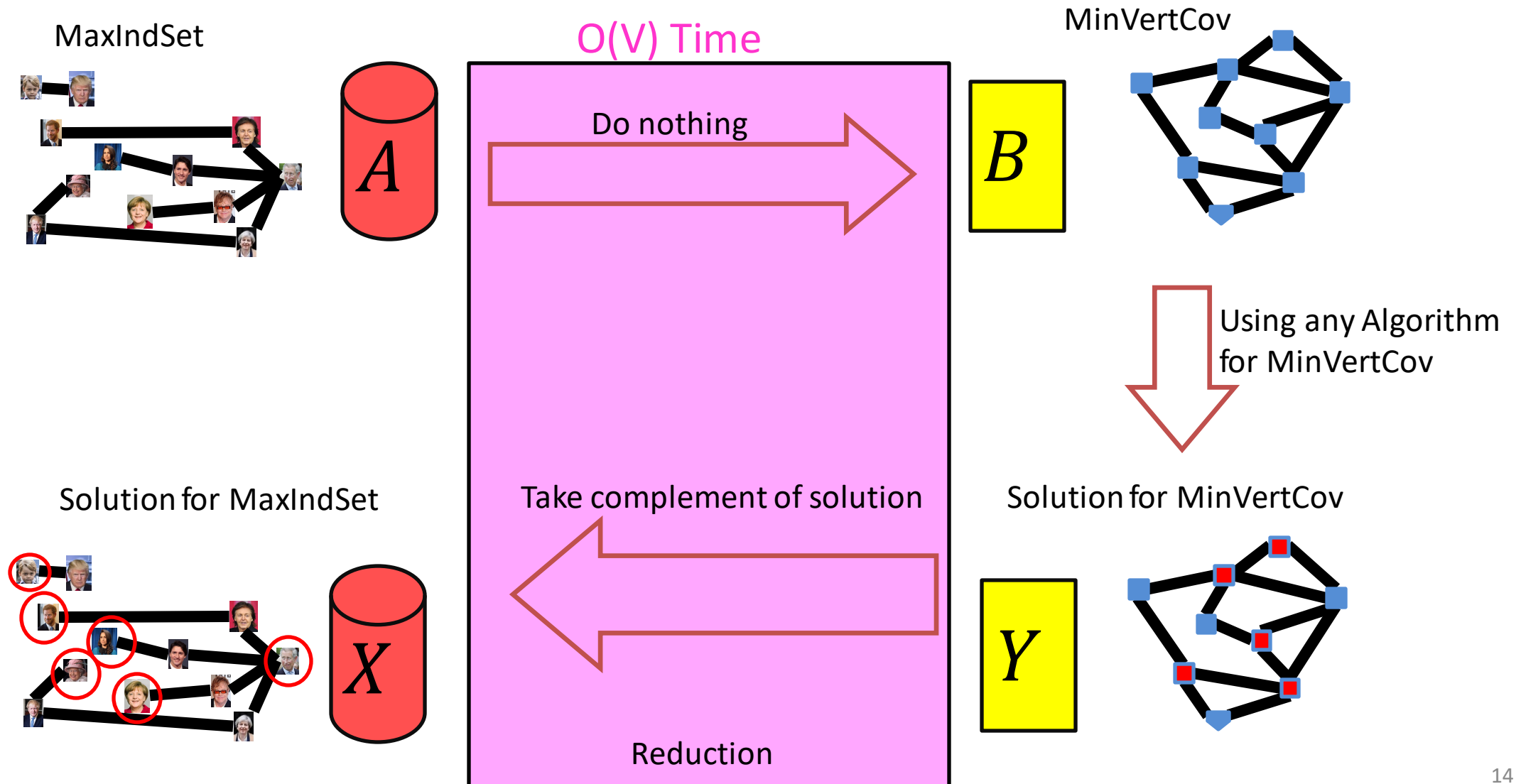
- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

Example

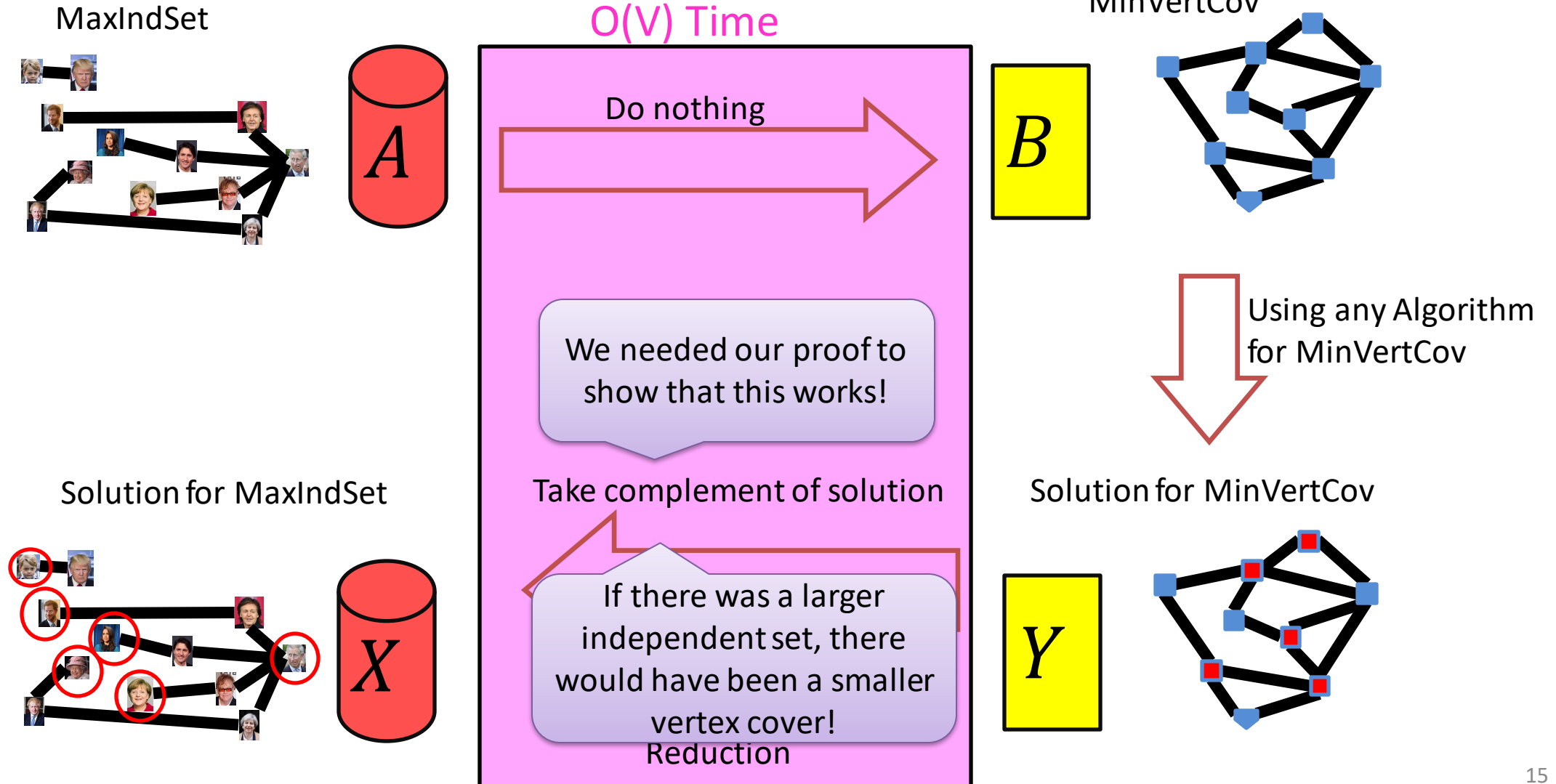


Vertex cover of size 5

MaxIndSet V -Time Reducible to MinVertCover

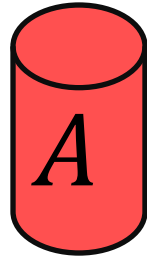
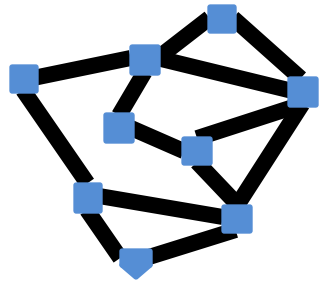


MaxIndSet V -Time Reducible to MinVertCov

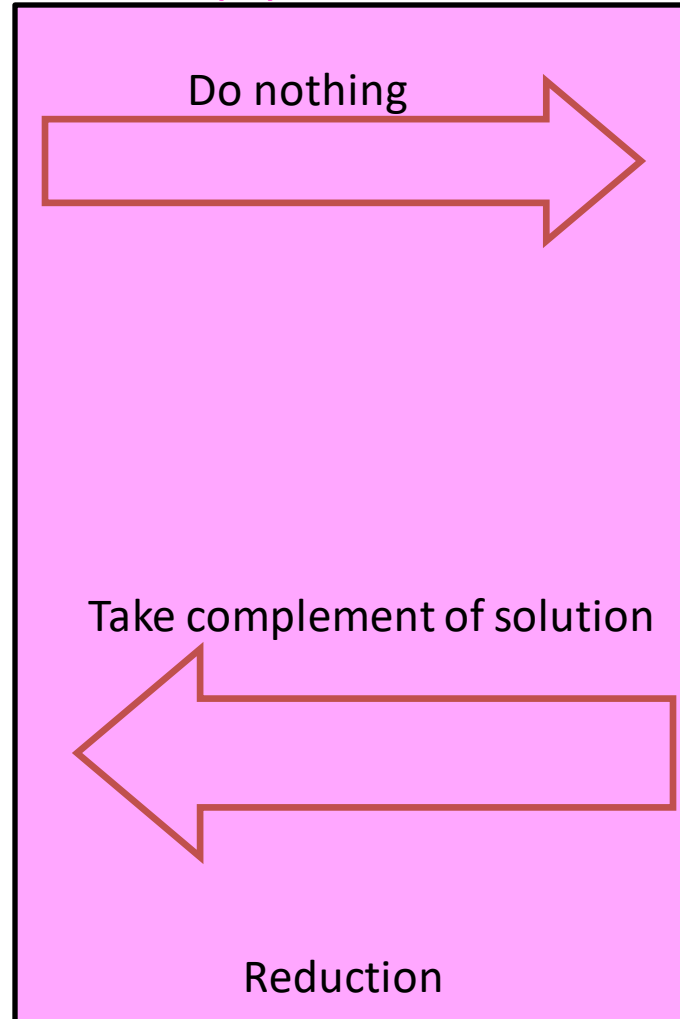


MinVertCover V -Time Reducible to MaxIndSet

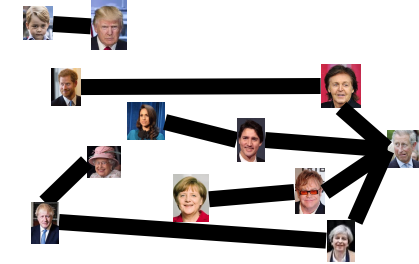
MinVertCov



$O(V)$ Time



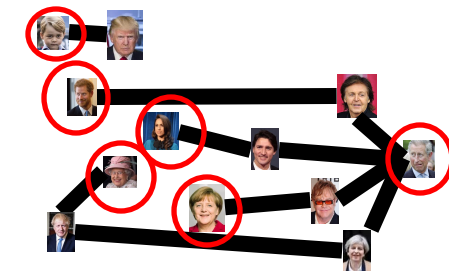
MaxIndSet



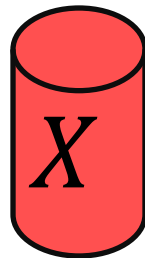
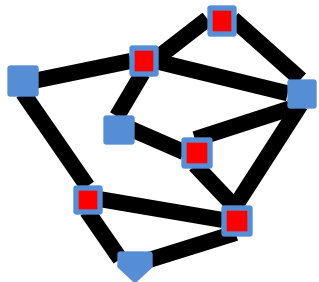
Using any Algorithm
for MaxIndSet



Solution for MaxIndSet



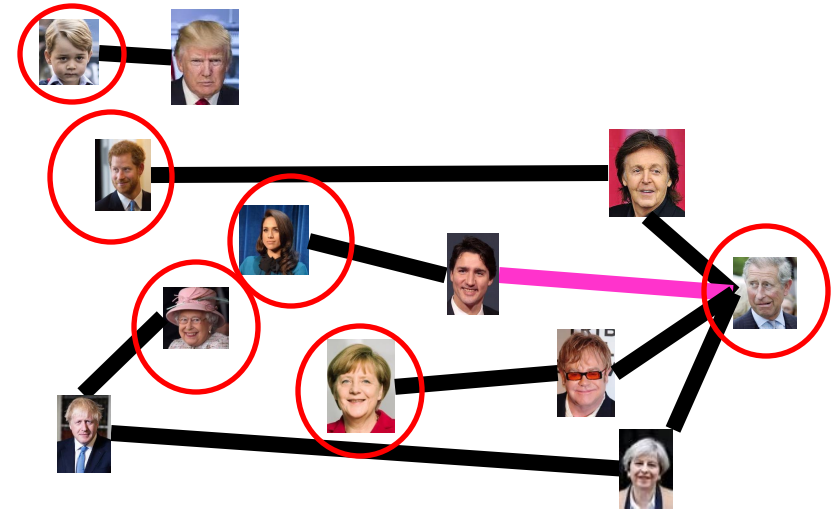
Solution for MinVertCov



Proof: \Rightarrow

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

Let S be an independent set



Consider any edge $(x, y) \in E$

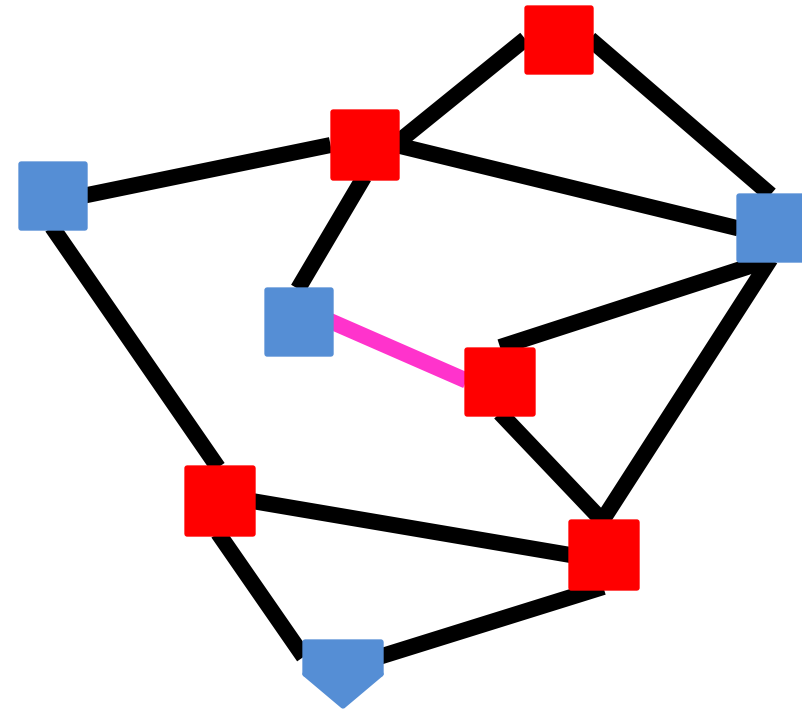
If $x \in S$ then $y \notin S$, because otherwise S would not be an independent set

Therefore $y \in V - S$, so edge (x, y) is covered by $V - S$

Proof: \Leftarrow

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

Let $V - S$ be a vertex cover



Consider any edge $(x, y) \in E$

At least one of x and y belong to $V - S$, because $V - S$ is a vertex cover

Therefore x and y are not both in S ,

No edge has both end-nodes in S , thus S is an independent set

Conclusion

- MaxIndSet and MinVertCov are either both fast, or both slow
 - Spoiler alert: We don't know which!
 - (But we think they're both slow)
 - Both problems are NP-Complete

Why Study NP-Completeness

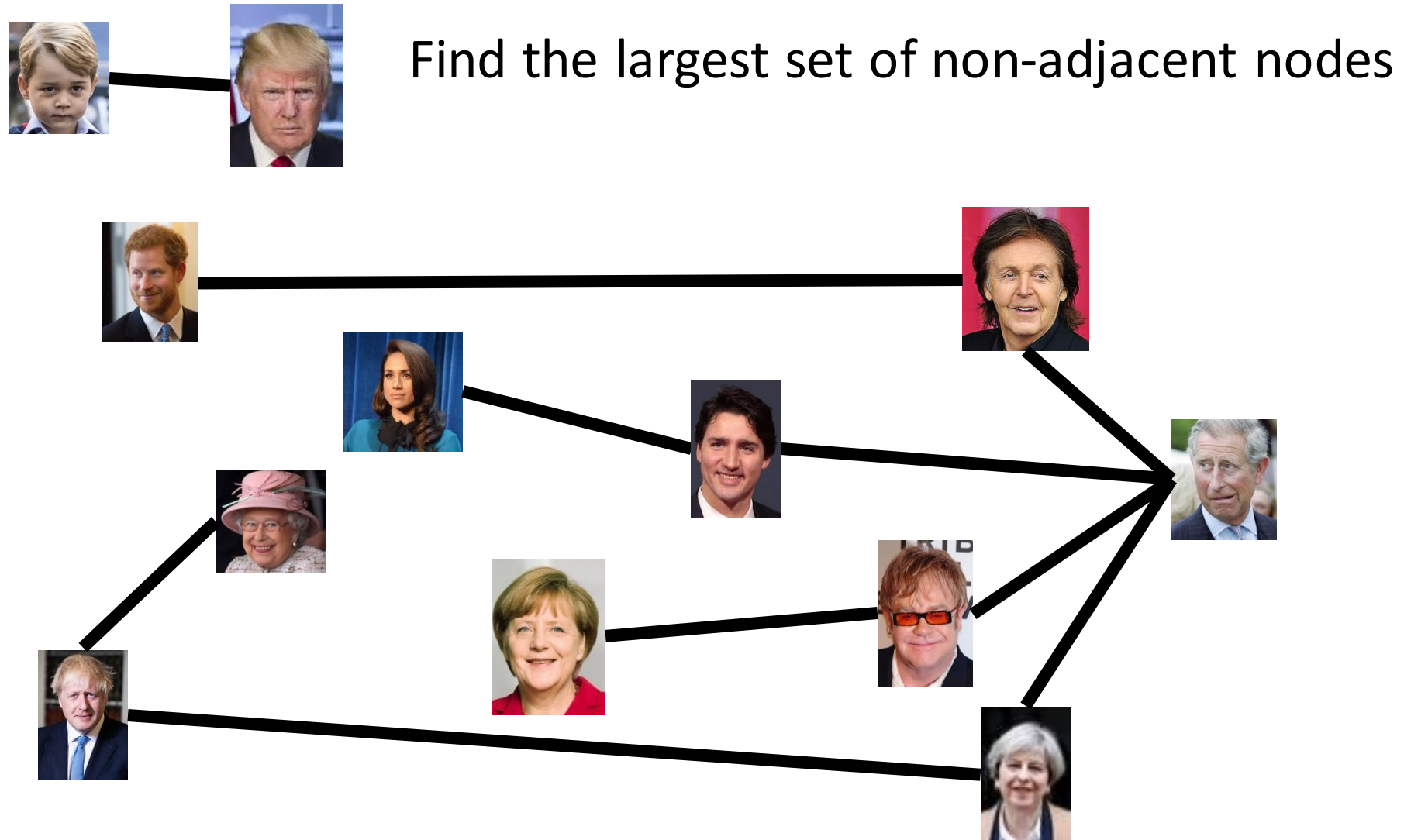
- All semester, we've studied *finding algorithms* to solve problems using various tools.
- Sometimes we instead need to prove that a problem is *extremely hard*, so as not to waste time on it!
 - NP-Complete Problems are hard
 - Let's go over a few of them quickly
 - Let's show how to prove a new problem is NP-Complete

Some Preliminaries

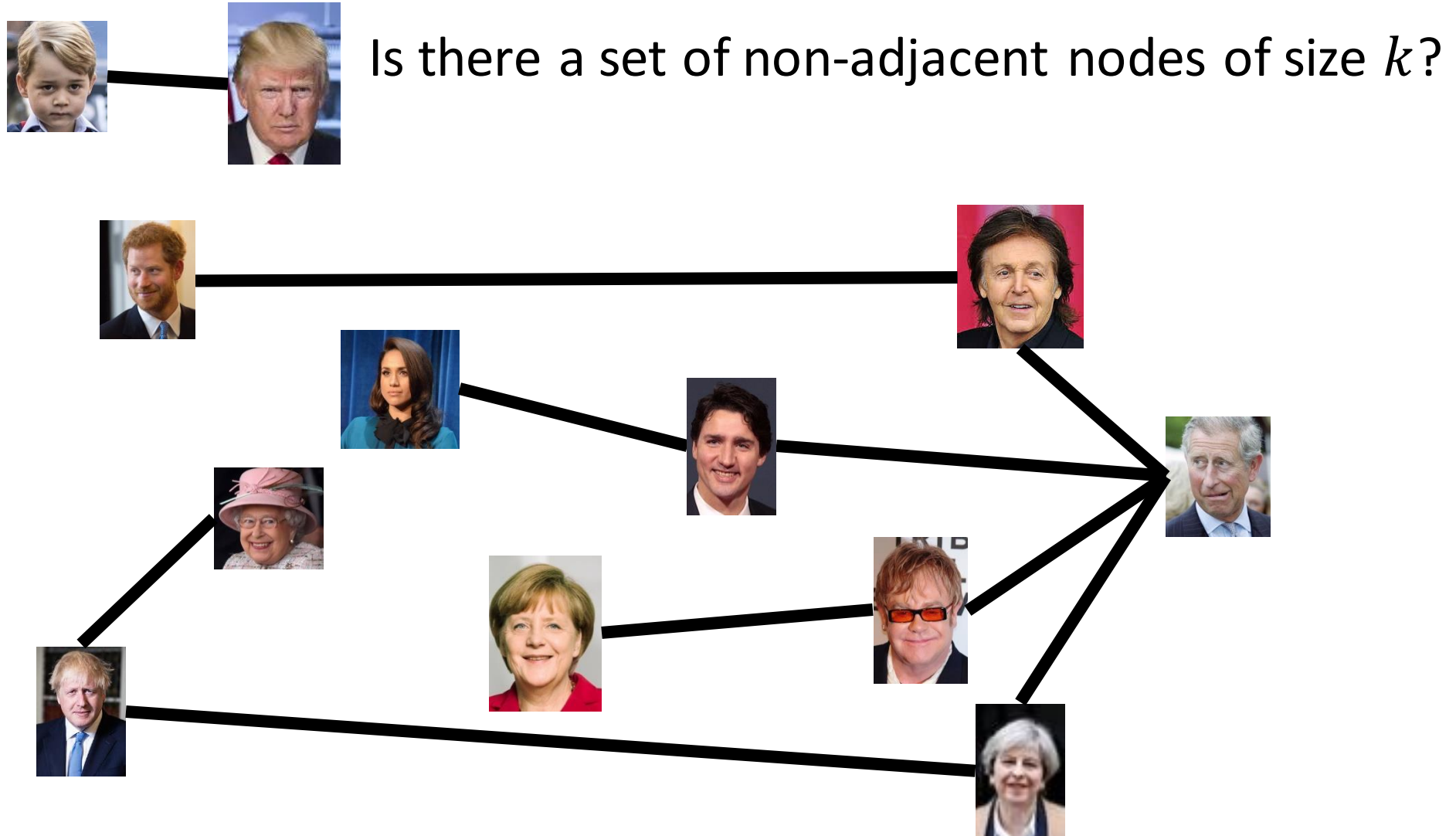
Before we go further on this topic....

- This is a complex (and interesting!) topic in CS theory
- In our few lectures, we may approach things from a simpler viewpoint than you'd get in a CS theory course
- The math and theory related to NP-complete problems starts with ***decision problems***
 - What's that? Let's use independent set and vertex cover as examples
 - What's described next applies to any optimization problems we've seen

Max Independent Set



k Independent Set



Maximum Independent Set

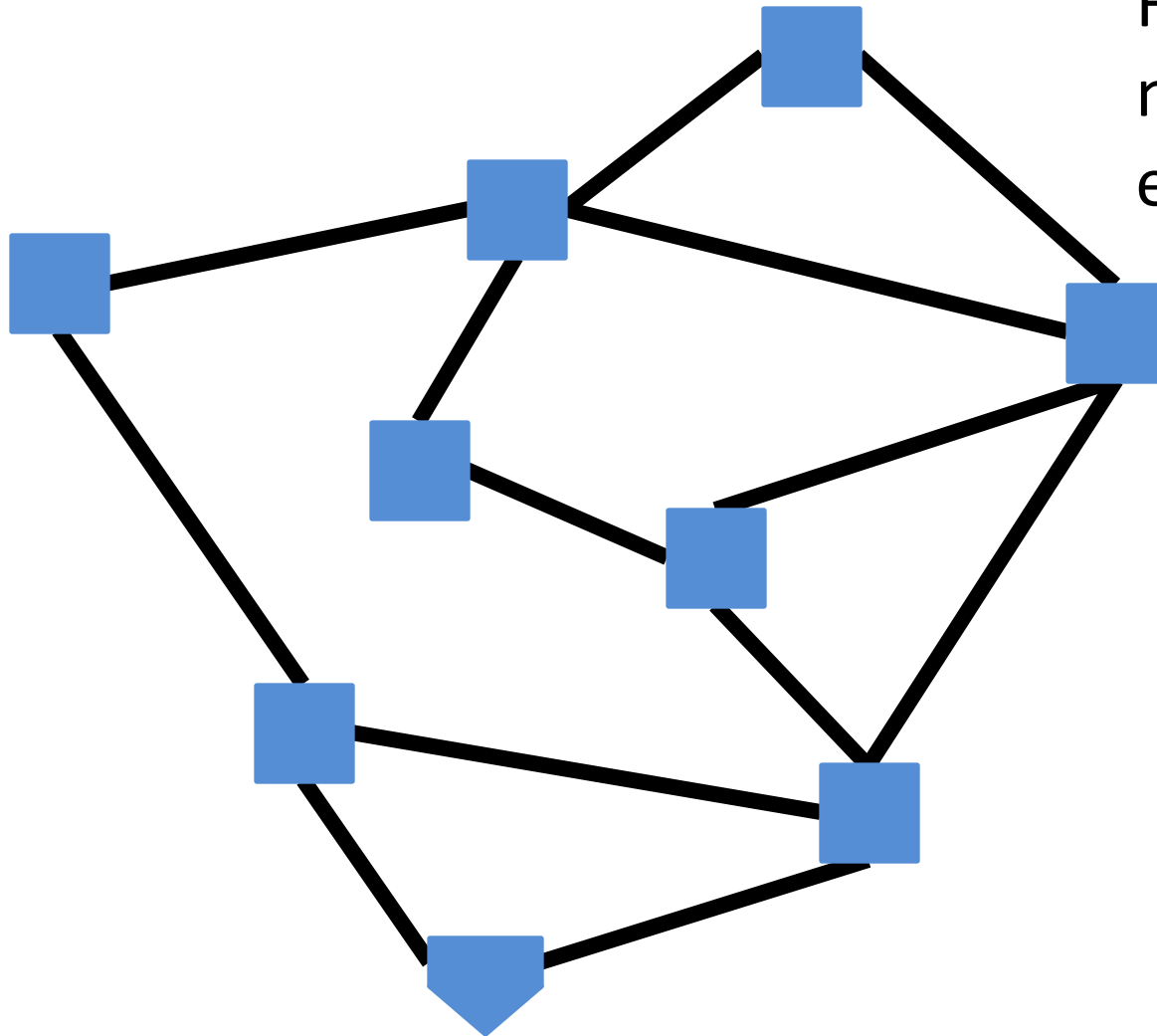
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

k Independent Set

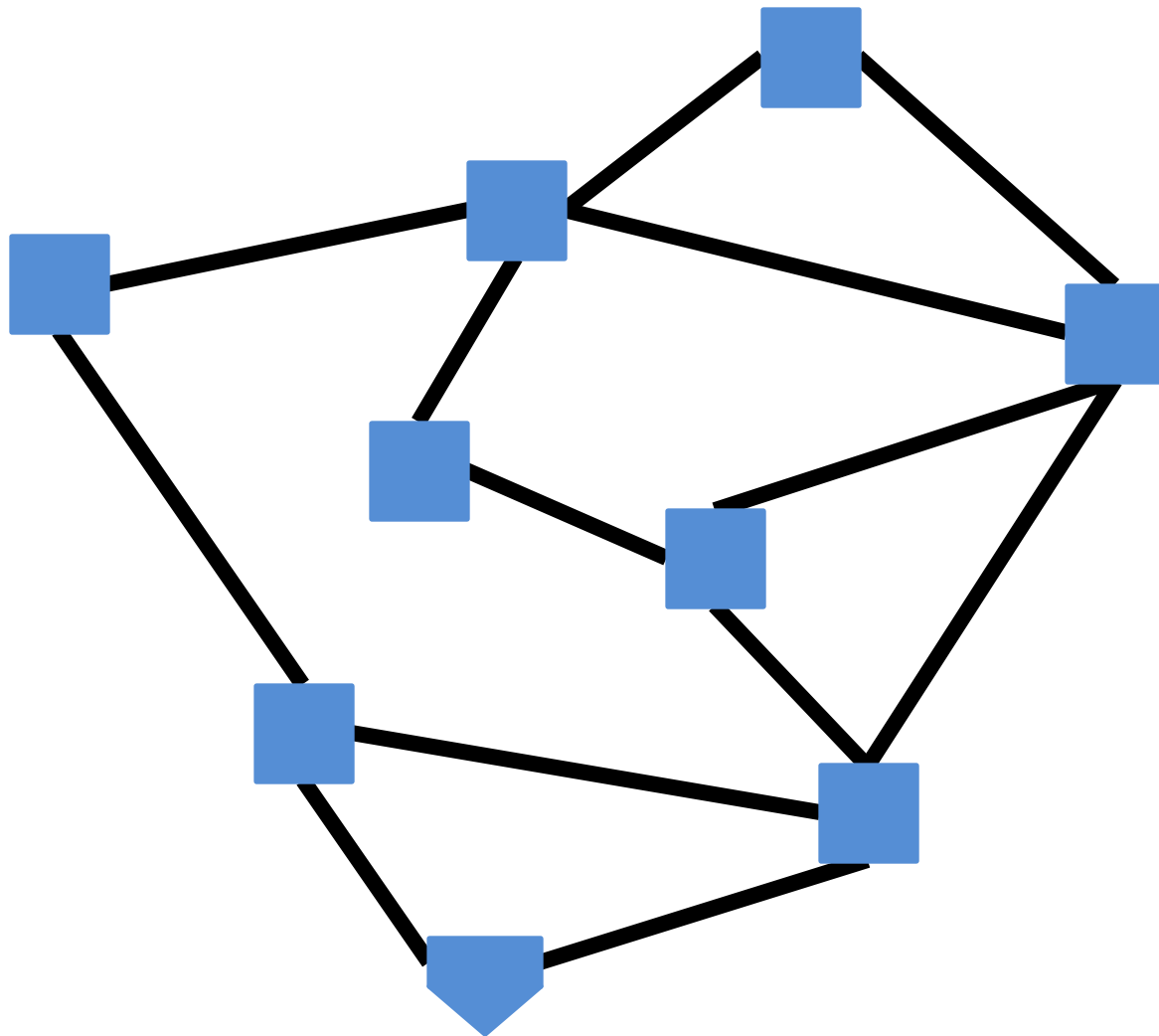
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- k Independent Set Problem: Given a graph $G = (V, E)$ and a number k , **determine whether there is an independent set S of size k**

Min Vertex Cover

Find the smallest set of nodes which covers every edge



k Vertex Cover



Is there a set of nodes of size k which covers every edge?

Minimum Vertex Cover

- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

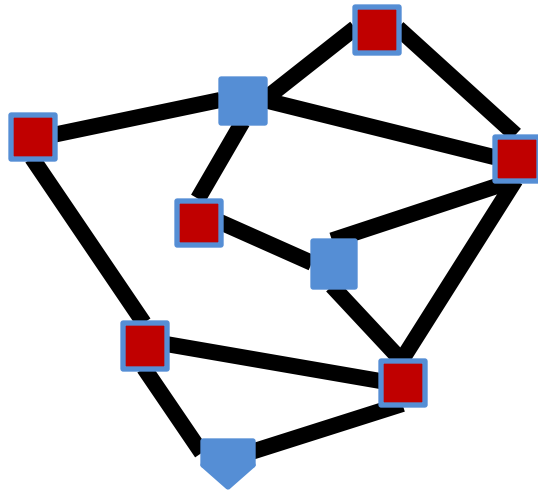
k Vertex Cover

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

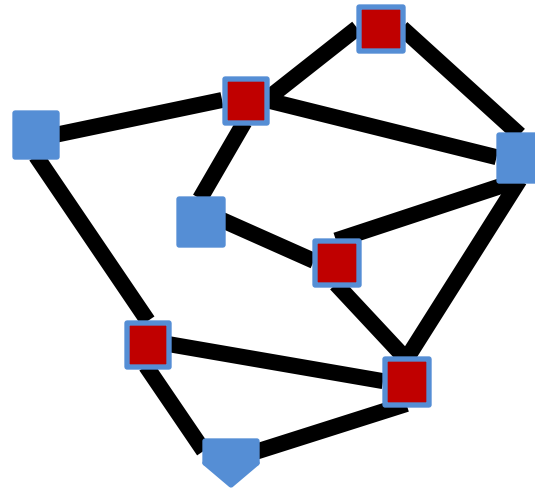
k Vertex Cover

- **k Vertex Cover Problem:** Given a graph $G = (V, E)$ and an integer k , determine if there is a vertex cover C of size k

True for $k=6$



True for $k=5$



Is 5 the smallest?
True for $k=4$?

Problem Types

- **Decision Problems:**

- Is there a solution?

- Result is True/False

- E.g. Is there a vertex cover of size k ?

If we can solve this...

- **Optimal Value Problems:**

- E.g. What's the min k for k -vertex cover decision problem?

...Then we can solve this,...

- **Search Problems:**

- Find a solution

- Result more complex than T/F or a k

- E.g. Find a vertex cover of size k

...and also this

- **Verification Problems:**

- Given a potential solution for an input, is that input valid?

- Result is True/False
- For *decision problem*, check solution to its *search problem*

- E.g. Is **set of vertices** a vertex cover of size k ?

Looking ahead:
We'll use this to define a problem classes P and NP

Looking ahead:
We'll use this to define a problem class called NP

Using a k -VertexCover decider to build a searcher

Note this is a reduction!

$kVC\text{-search} \leq_p kVC\text{-decider}$

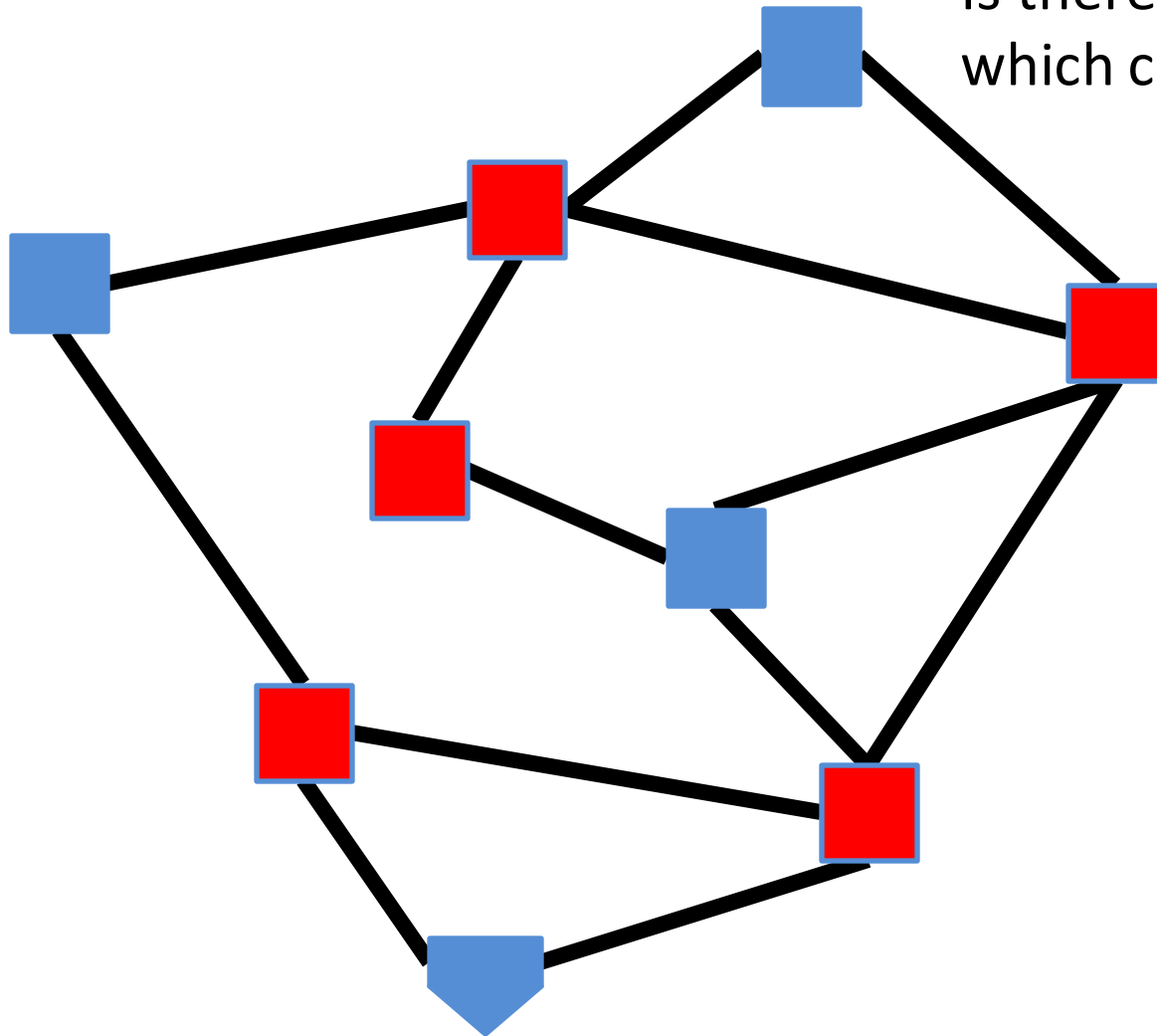
- Set $i = k - 1$
- Remove nodes (and incident edges) one at a time
- Check if there is a vertex cover of size i (i.e. use the “decider”)
 - If so, then that removed node was part of the k vertex cover, set $i = i - 1$
 - Else, it wasn't

Did I need this node to cover its edges to have a vertex cover of size k ?

5 Vertex Cover (Decision)

Is there a set of nodes of size 5 which covers every edge?

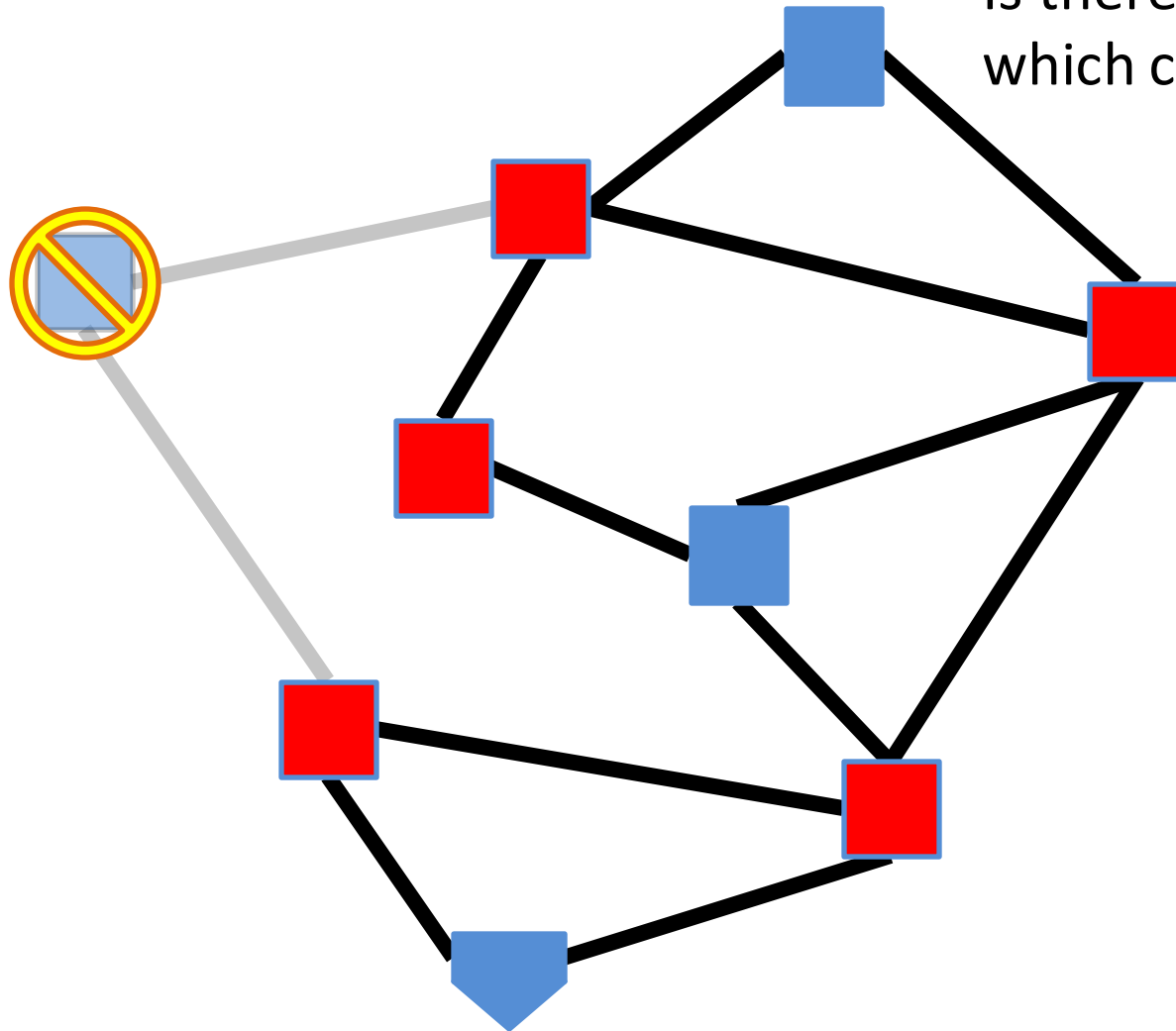
Yes!



4 Vertex Cover (Decision)

Is there a set of nodes of size 4 which covers every edge?

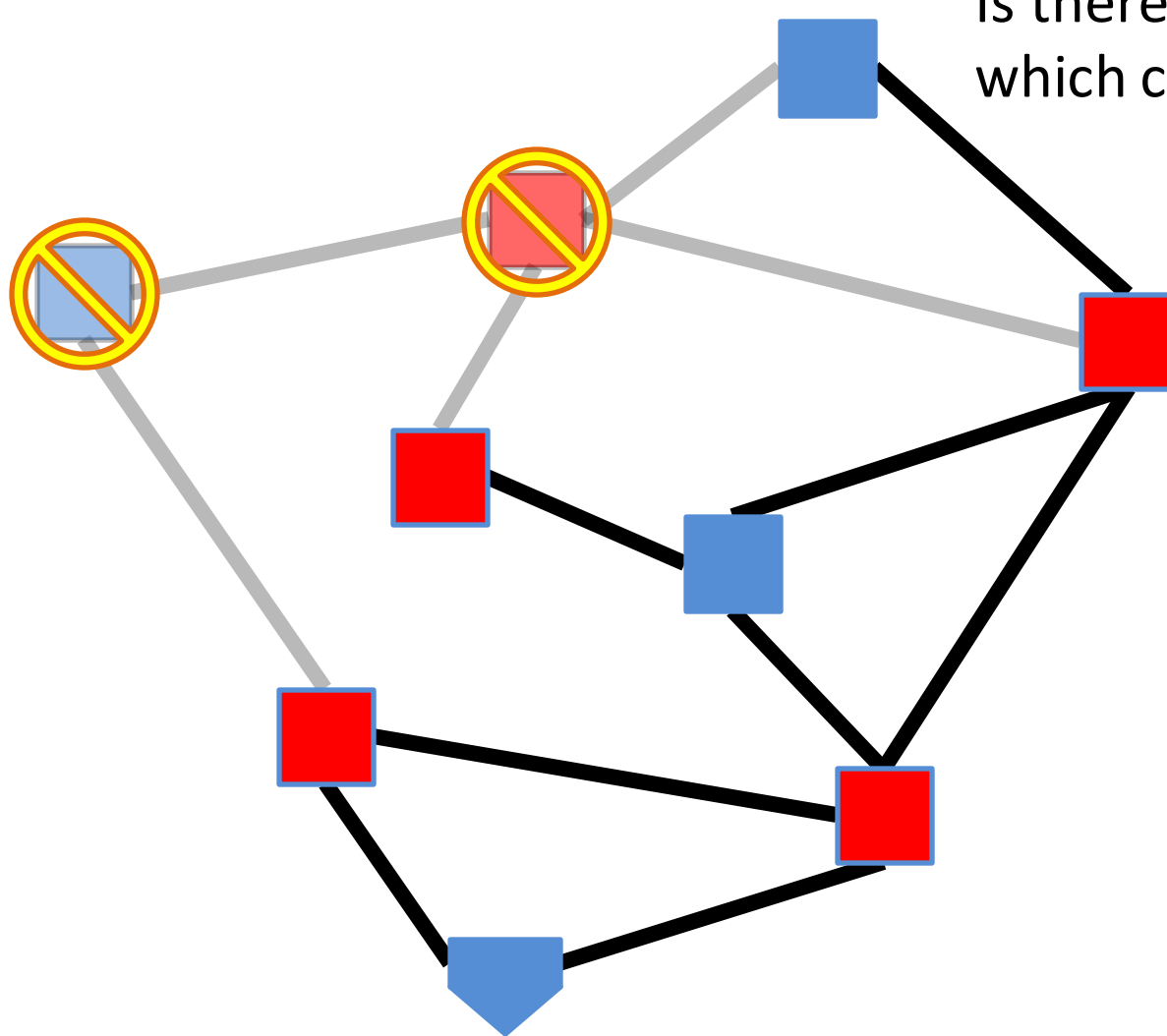
No!



4 Vertex Cover (Decision)

Is there a set of nodes of size 4 which covers every edge?

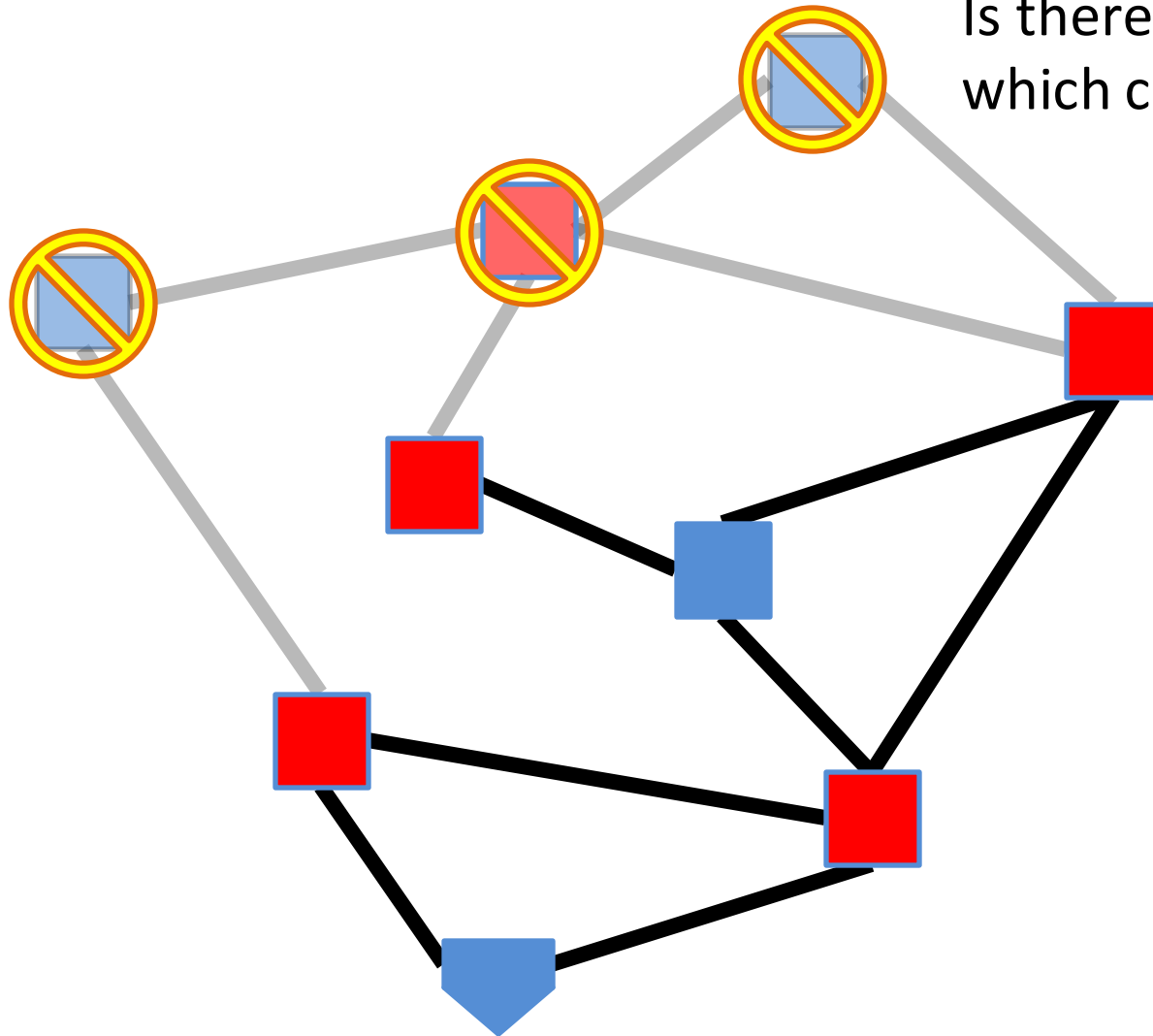
Yes!



3 Vertex Cover (Decision)

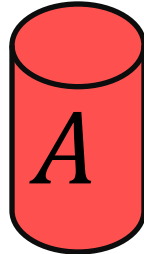
Is there a set of nodes of size 3 which covers every edge?

No!

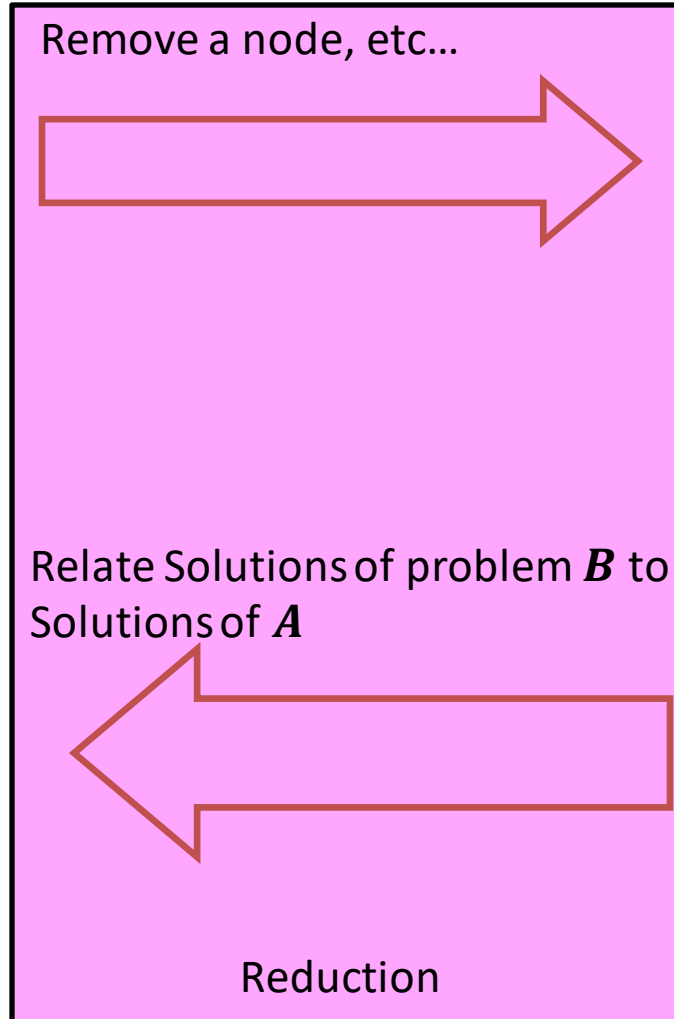
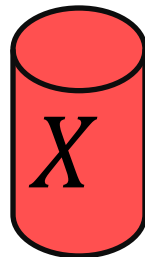


Reduction

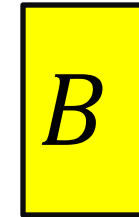
k -VertexCover Solver



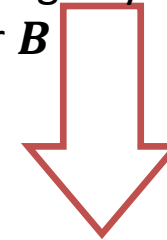
Solution for A



k -VertexCover Decider



Using any Algorithm for B

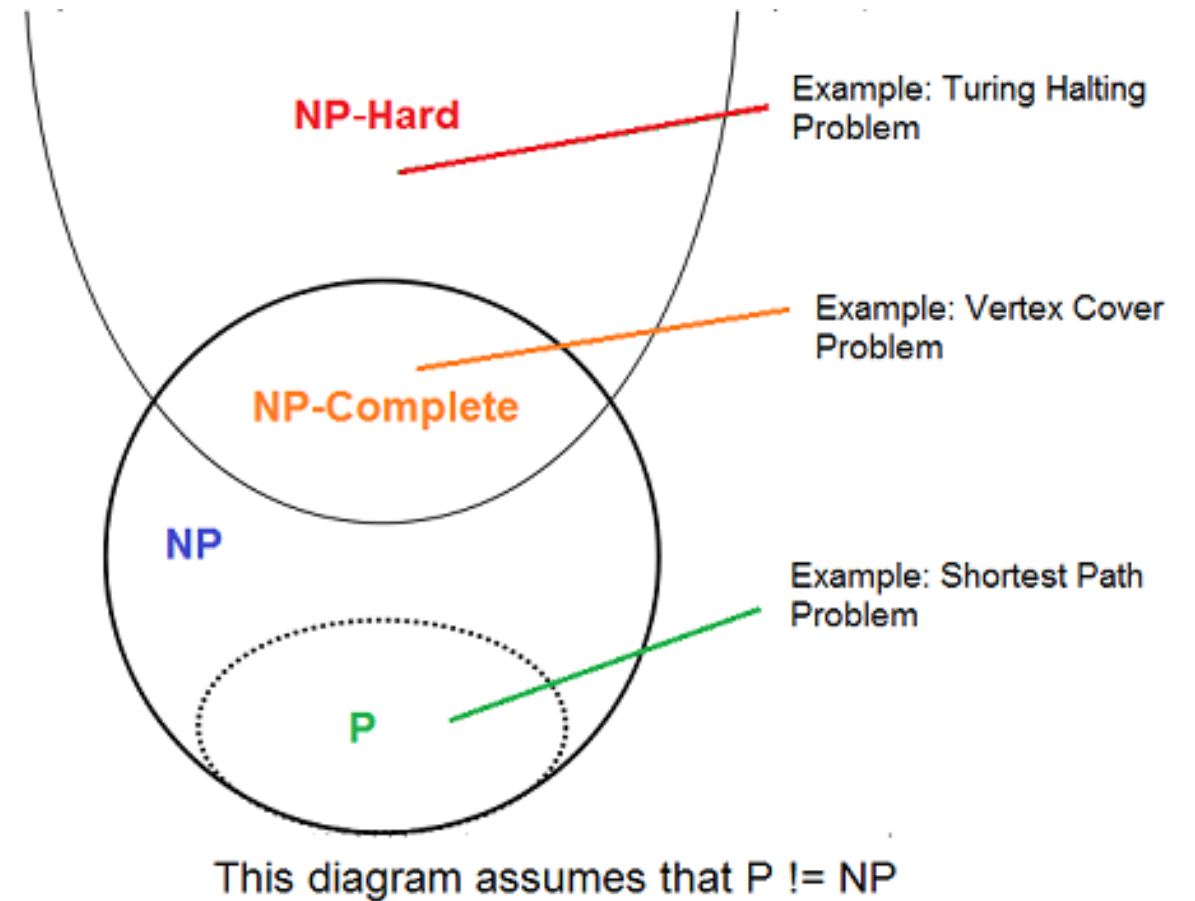


Solution for B



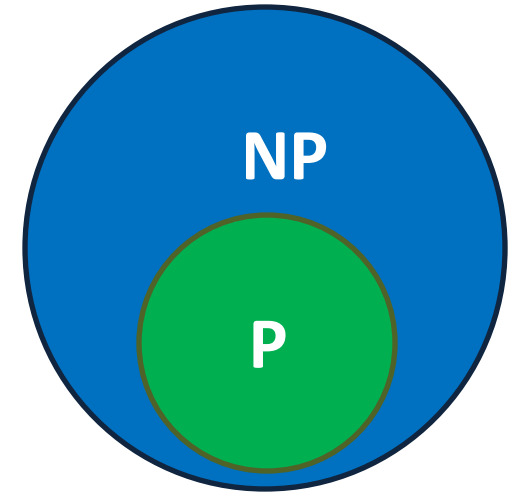
Quick Background!

- **P**: Set of problems solved in polynomial time (e.g., sorting a list)
- **NP**: Set of problems that can be:
 - 1) Solved in non-deterministic polynomial time
 - 2) A solution verified in polynomial time
- **NP-Hard**: Set of problems that are as hard as (or harder) than the hardest problems in NP
- **NP-Complete**: Set of problems that are both NP and NP-Hard (i.e., the equally hardest problems in NP)



Classes of Problems: P vs NP

- P
 - Deterministic Polynomial Time
 - P is the set of problems solvable in polynomial time
 - $O(n^c)$ for some number c
- NP
 - Non-Deterministic Polynomial Time
 - NP is the set of problems **verifiable** in polynomial time
 - Verify a proposed solution (not find one) in $O(n^c)$ for some number c
 - For decision problems, really verifying using some information we call a *certificate*
- Open Problem: Does $P=NP$?
 - Certainly $P \subseteq NP$



k -Independent Set is NP

- To show: Given a potential solution, can we **verify** it in $O(n^p)$?
[$n = V + E$]

How can we verify it?

1. Check that it's of size k $O(V)$
2. Check that it's an independent set $O(V^2)$

k -Vertex Cover is NP

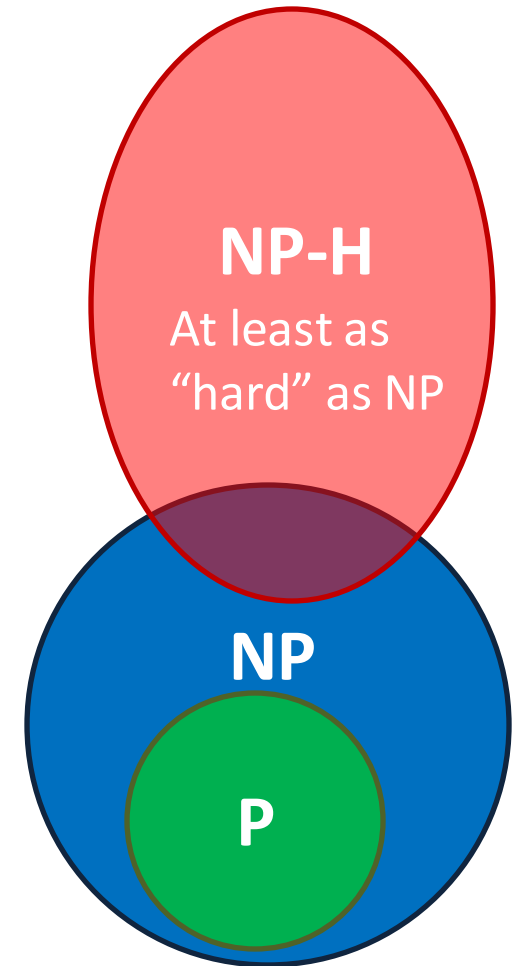
- To show: Given a potential solution, can we **verify** it in $O(n^p)$?
[$n = V + E$]

How can we verify it?

1. Check that it's of size k $O(V)$
2. Check that it's a Vertex Cover $O(E)$

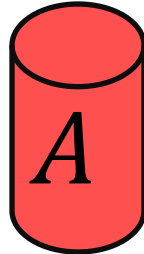
NP-Hard

- 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 if $\forall A \in NP, A \leq_p B$
 - $A \leq_p B$ means A reduces to B in polynomial time
 - Remember: $A \leq_p B$ implies A is not harder than B



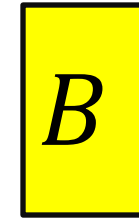
NP-Hardness Reduction

Any NP Problem



$O(n^p)$

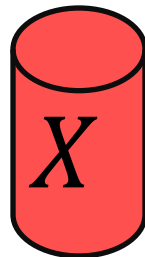
Problem to show is NP-Hard



Then this could be solved
in polynomial time

If This could be solved
in Polynomial time

Solution for A



Solution for B



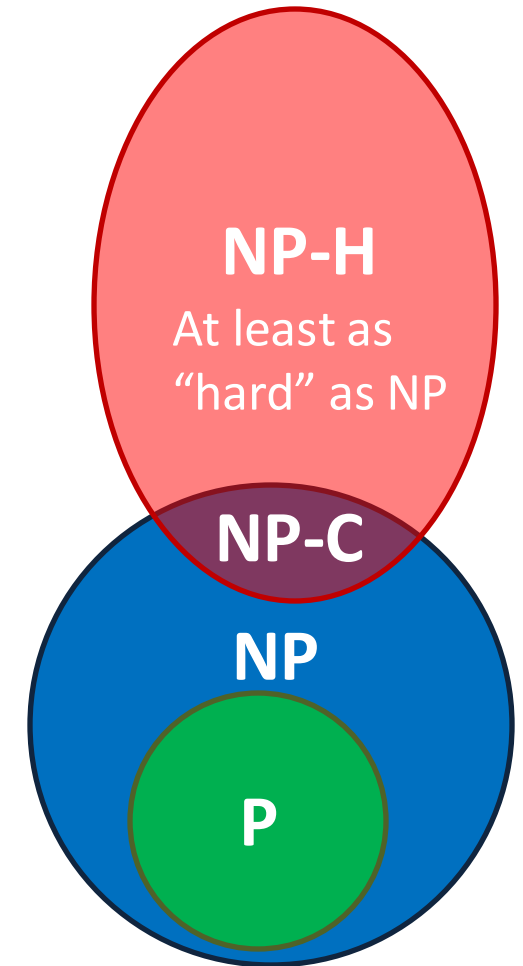
Reduction

$$A \leq_p B$$

NP-Complete

NP-Complete = $NP \cap NP\text{-Hard}$

- The “hardest” of all the problems in NP
- An NP-C problem is polynomial iff all NP problems are polynomial. I.e. $P=NP$
- If $P=NP$, then all NP-C problems are polynomial
- “Together they stand, together they fall”
- **How to show a problem C is NP-Complete?**
 - Show C belongs to NP
 - Show we can verify a solution in polynomial time
 - Show C is NP-Hard
 - $\forall A \in NP, A \leq_p C$ (That sounds really hard to do!)
 - Or, show a reduction from another NP-Hard problem. (Why? Details next.)



We now just need a FIRST NP-Hard problem

NP-Completeness

- So...a problem is NP-Complete if you can do the following:
- 1) Show how to verify it in polynomial time
 - Given a solution to the problem, verify it is correct
 - That algorithm's runtime needs to be a polynomial (usually easy)
- 2) Show the problem is NP-Hard (as hard or harder than a known NP-Hard Problem)
 - Take a currently known NP-Hard problem (let's call it A)
 - Show that $A \leq_p X$ (where X is your problem)
 - Why? If A is NP-Hard, then: *any NP problem $\leq_p A$*
 - Transitivity: *any NP problem $\leq_p A \leq_p X$*
 - So X satisfies definition of NP-Hard

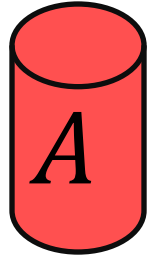
“Consequences” of NP-Completeness

- NP-Complete is the set of “hardest” problems in NP, with these important properties:
 - If any *one* NP-Complete problem can be solved in polynomial time...
 - ...then *every* NP-Complete problem can be solved in polynomial time...
 - ...and in fact *every* problem in **NP** can be solved in polynomial time (which would show **P = NP**)
 - Or, prove an exponential lower-bound for *any single NP-hard* problem, then *every NP-hard* problem (including **NP-C**) is exponential

Therefore: solve (say) traveling salesperson problem in $O(n^{100})$ time, you've proved that **P = NP**. Retire rich & famous!

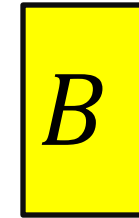
NP-Completeness: $A \leq_p B$ and $B \in P$

Any NP-Complete Problem



$O(n^p)$

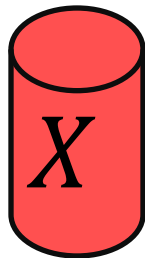
Any other NP-Complete Problem



Then this could be done
in polynomial time

If This could be done in
polynomial time

Solution for A



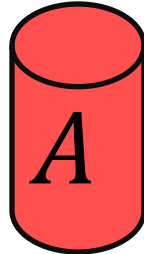
Solution for B



Reduction

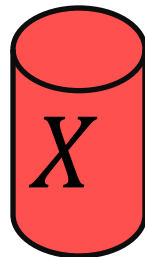
NP-C: $A \leq_p B$ and we prove A not in P

Any NP-Complete Problem

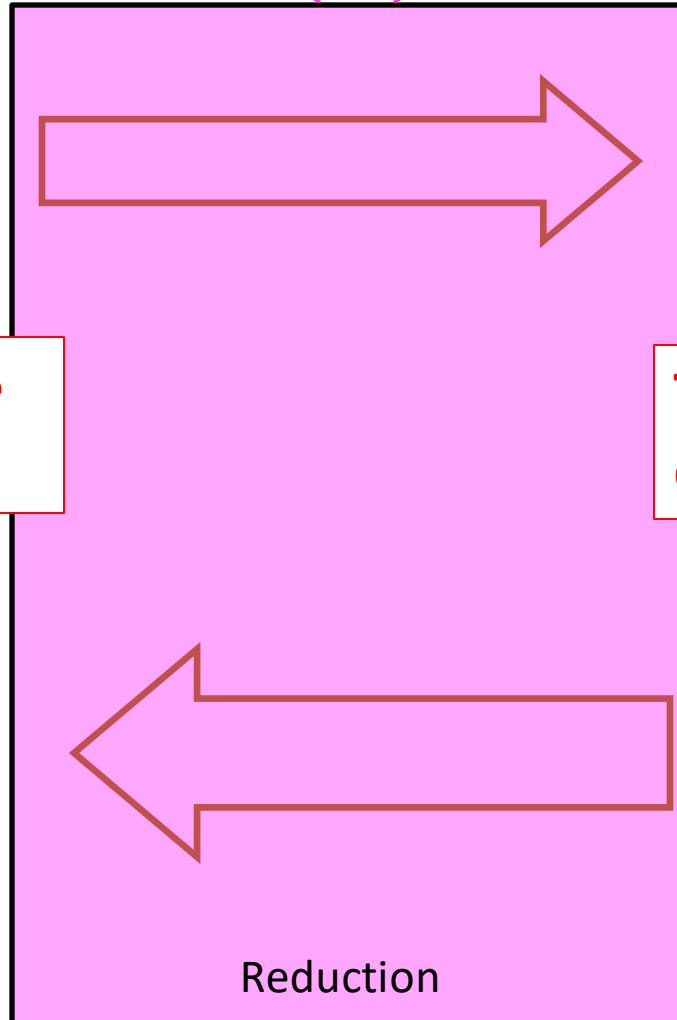


If this cannot be done in polynomial time

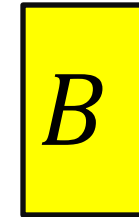
Solution for A



$O(n^p)$



Any other NP-Complete Problem



Then this cannot be done in polynomial time

Solution for B



Summary of Where We Are

- Focusing on “hard” problems, those that seem to be exponential
- Reductions used to show “hardness” relationships between problems
- Starting to define “classes” of problems based on complexity issues
 - **P** are problems that can be solved in polynomial time
 - **NP** are problems where a solution can be verified in polynomial time
 - **NP-hard** are problems that are at least as hard as anything in NP
 - **NP-complete** are NP-hard problems that “stand or fall together”

Review: P and NP Summary

- **P** = set of problems that can be solved in polynomial time
- **NP** = set of problems for which a solution can be verified in polynomial time
 - Note: this is a more “informal” definition, but it’s fine for CS4102
 - See later slide on “certificates” for more info.
- **$P \subseteq NP$**
- Open question: Does **$P = NP$** ?

More Reminders and Some Consequences

- **Definition of NP-Hard and NP-Complete:**
 - If all problems $A \in \mathbf{NP}$ are reducible to B , then B is *NP-Hard*
 - We say B is *NP-Complete* if:
 - B is NP-Hard
 - and $B \in \mathbf{NP}$
- Any NP-C must reduce to any other NP-C. Can you see why?
- If $B \leq_p C$ and B is NP-Complete, C is also NP-Complete
 - Don't see why? We'll show details in two more slides
 - As long as $C \in \mathbf{NP}$. Otherwise can only say $C \in \mathbf{NP-hard}$.

3-SAT

- Shown to be NP-Hard by Cook and Levin (independently)
- Given a 3-CNF formula (logical AND of **clauses**, each an OR of 3 **variables**), Is there an **assignment** of true/false to each variable to make the formula true?

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

Clause

Variables

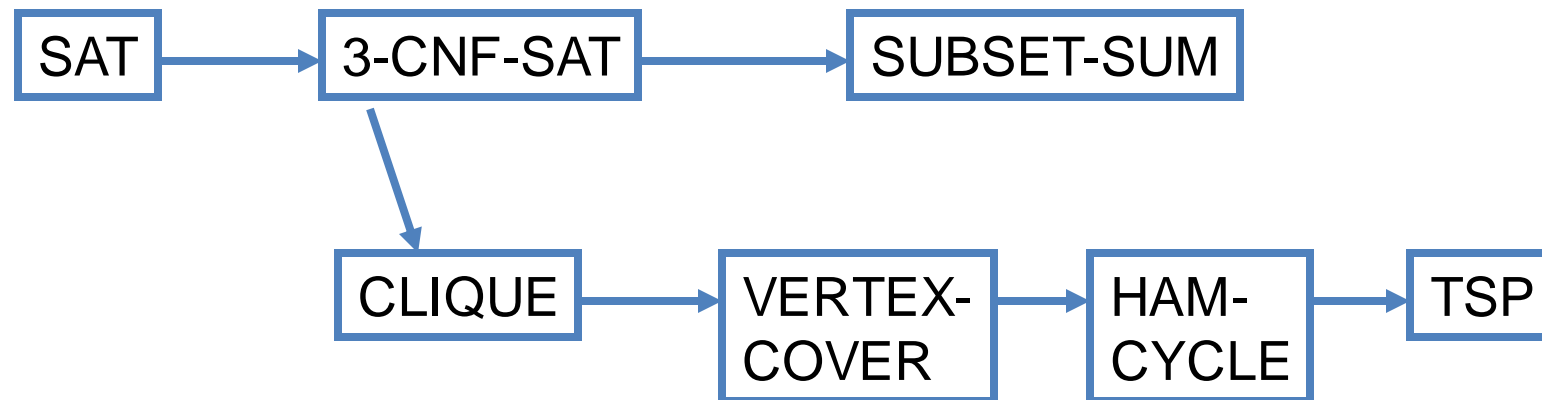
$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

Conjunctive Normal Form (CNF)

- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
 - *Literal*: an occurrence of a Boolean or its negation
 - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
 - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
 - *3-CNF*: each clause has exactly 3 distinct literals
 - Ex: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5 \vee x_3 \vee x_4)$
 - Notice: true if at least one literal in each clause is true
 - Note: Arbitrary SAT expressions can be translated into CNF forms by introducing intermediate variables etc.

Joining the Club

- Given one NP-Complete problem, others can join the club
 - Prove that SAT reduces to another problem, and so on...



- Membership in NP-Complete grows...
- Classic textbook: Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

k -Independent Set is NP-Complete

1. Show that it belongs to NP
 - Give a polynomial time verifier
2. Show it is NP-Hard
 - Give a reduction from a known NP-Hard problem
 - Show $3SAT \leq_p kIndSet$

Remember: k -Independent Set is NP

- To show: Given a certificate (“solution” for the search problem), can we **verify** it in $O(n^p)$? [$n = V + E$]

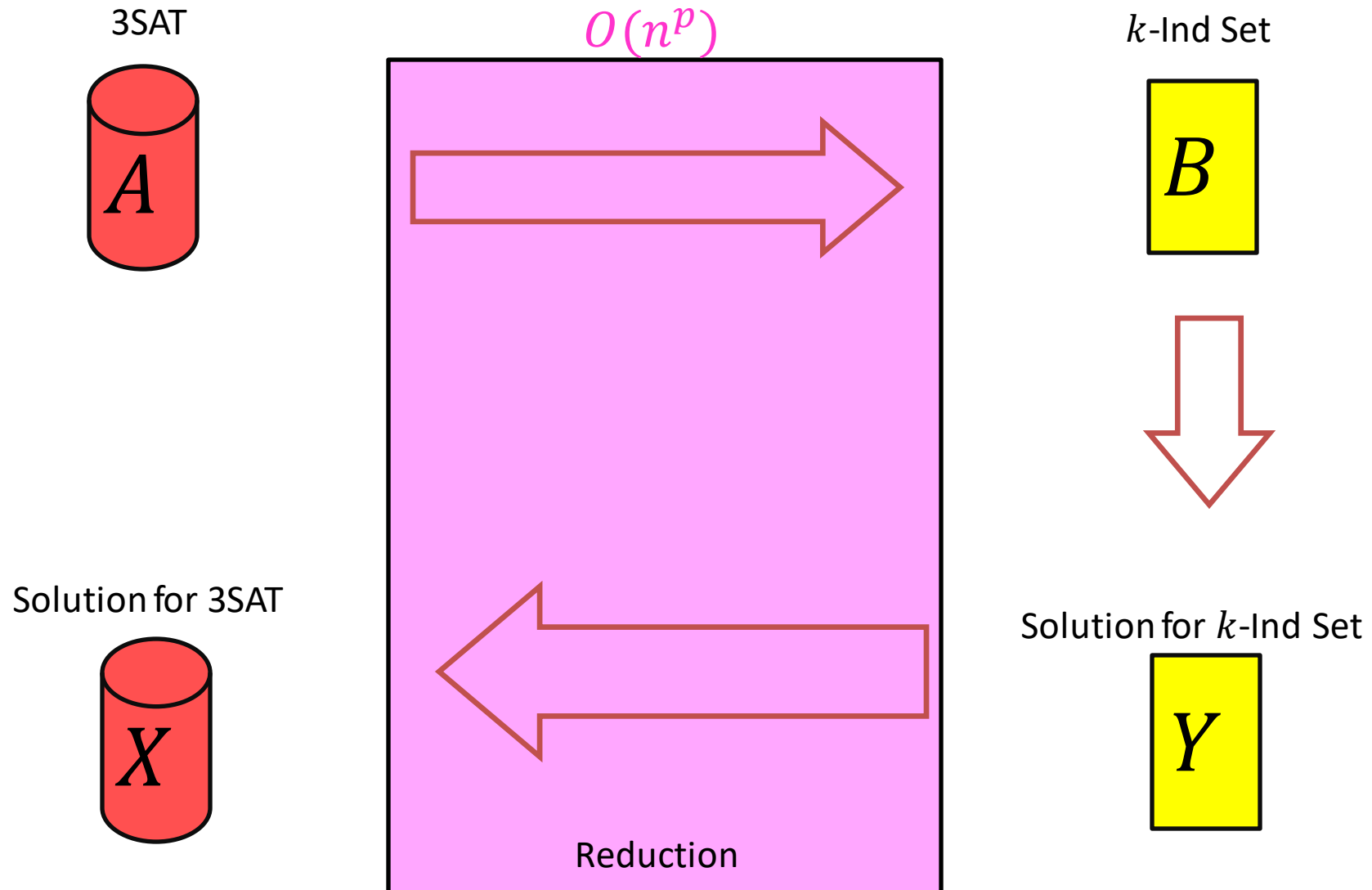
How can we verify it?

1. Check that it's of size k $O(V)$
2. Check that it's an independent set $O(V^2)$

k -Independent Set is NP-Complete

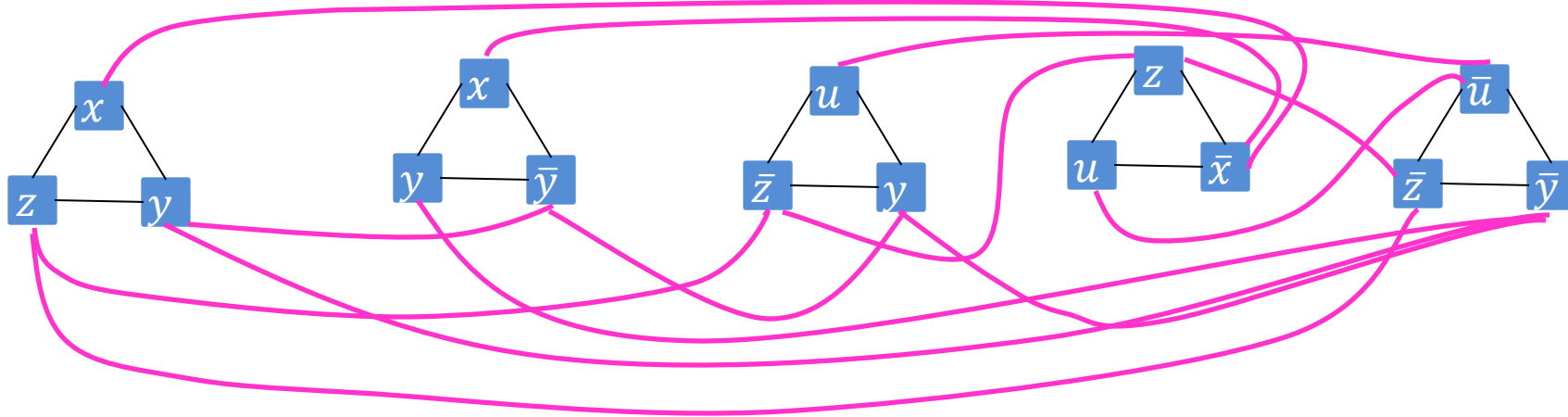
1. Show that it belongs to NP
 - Give a polynomial time verifier
2. Show it is NP-Hard
 - Give a reduction from a known NP-Hard problem
 - Show $3SAT \leq_p kIndSet$

$3SAT \leq_p kIndSet$



Instance of 3SAT to Instance of k IndSet

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



For each clause, produce a triangle graph with its three variables as nodes

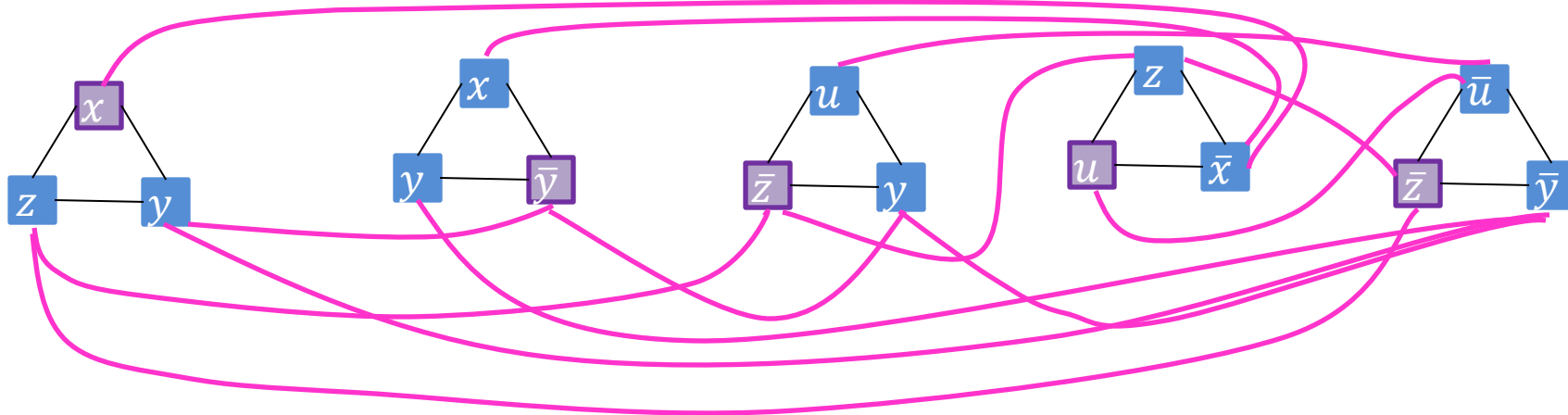
Connect each node to all of its opposites

Let k = number of clauses

There is a k -IndSet in this graph **iff** there is a satisfying assignment

k IndSet \Rightarrow Satisfying Assignment

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



$x = true$
 $y = false$
 $z = false$
 $u = true$

One node per triangle is in the Independent set:

because we can have exactly k total in the set, and 2 in a triangle would be adjacent

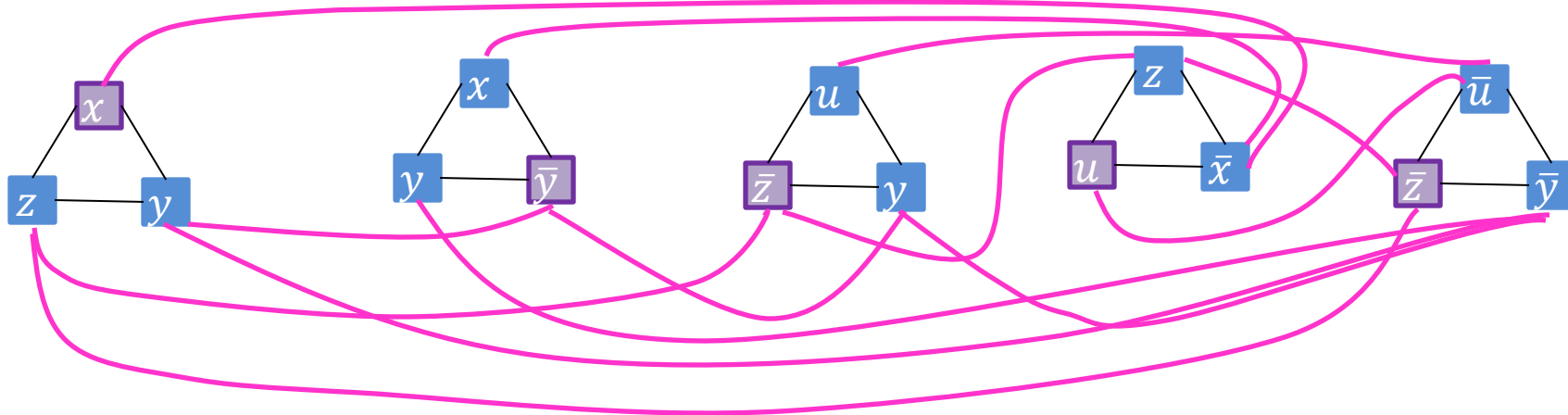
If x is selected in some triangle, \bar{x} is not selected in any triangle:

Because every x is adjacent to every \bar{x}

Set the variable which each included node represents to “true”

Satisfying Assignment $\Rightarrow k$ IndSet

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



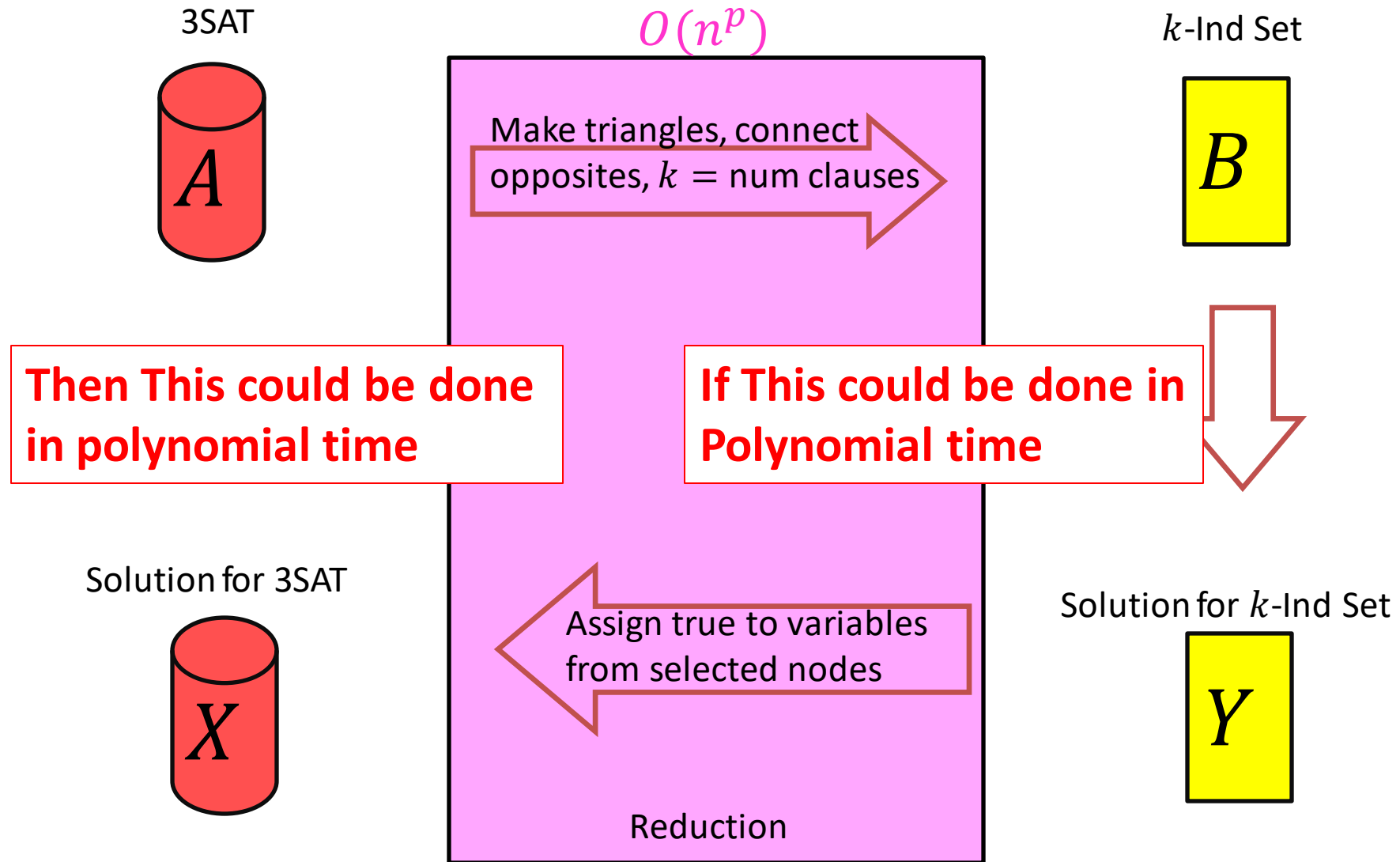
$x = true$
 $y = false$
 $z = false$
 $u = true$

Use one true variable from the assignment for each triangle

The independent set has k nodes, because there are k clauses

If any variable x is true then \bar{x} cannot be true

$3SAT \leq_p kIndSet$



k -Vertex Cover is NP-Complete

1. Show that it belongs to NP
 - Give a polynomial time verifier
2. Show it is NP-Hard
 - Give a reduction from a known NP-Hard problem
 - We showed $kIndSet \leq_p kVertCov$

Remember: k -Vertex Cover is NP

- To show: Given a certificate (“solution” for the search problem), can we verify it in $O(n^p)$? [$n = V + E$]

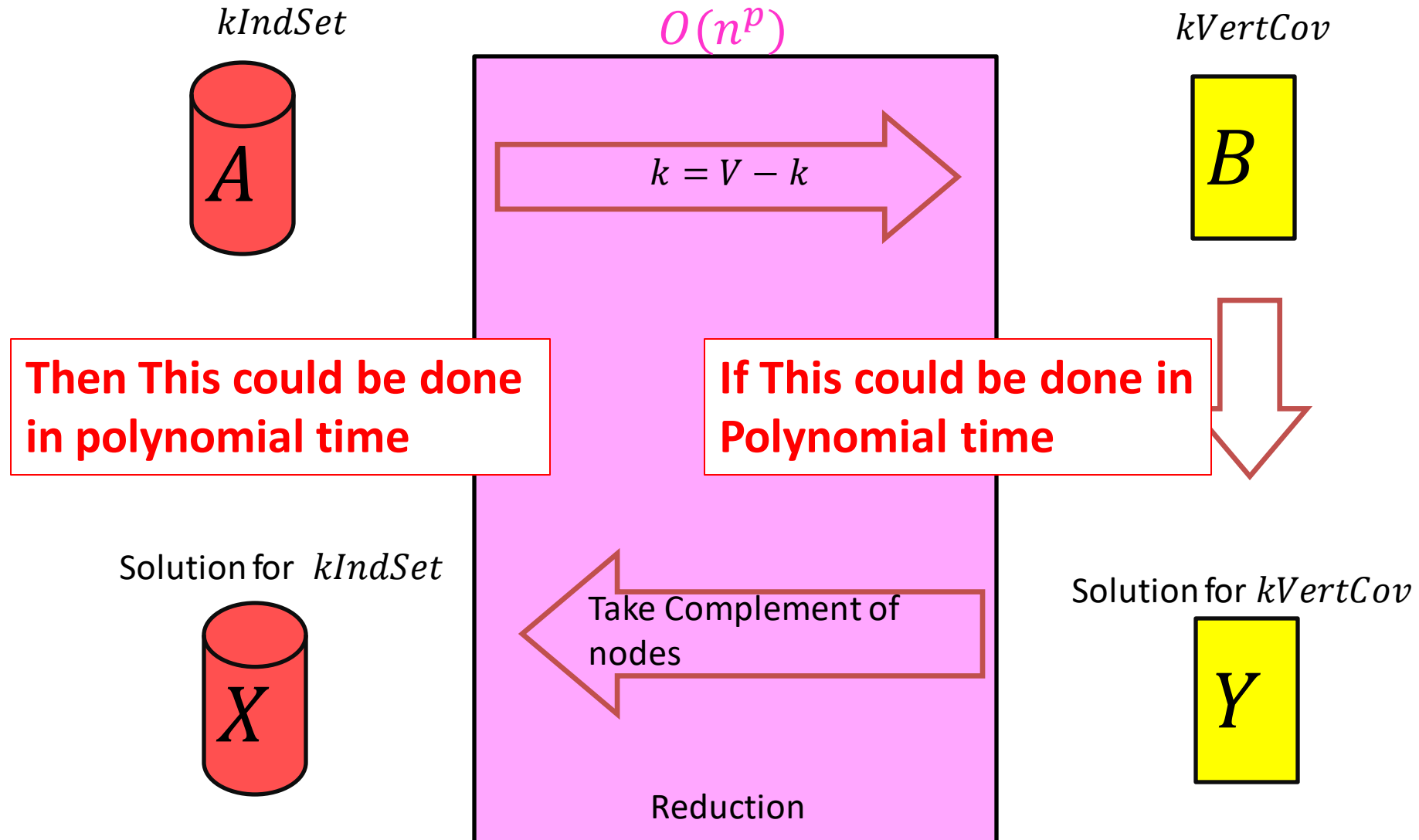
How can we verify it?

1. Check that it's of size k $O(V)$
2. Check that it's a Vertex Cover $O(E)$

k -Vertex Cover is NP-Complete

1. Show that it belongs to NP
 - Give a polynomial time verifier
2. Show it is NP-Hard
 - Give a reduction from a known NP-Hard problem
 - We showed $kIndSet \leq_p kVertCov$

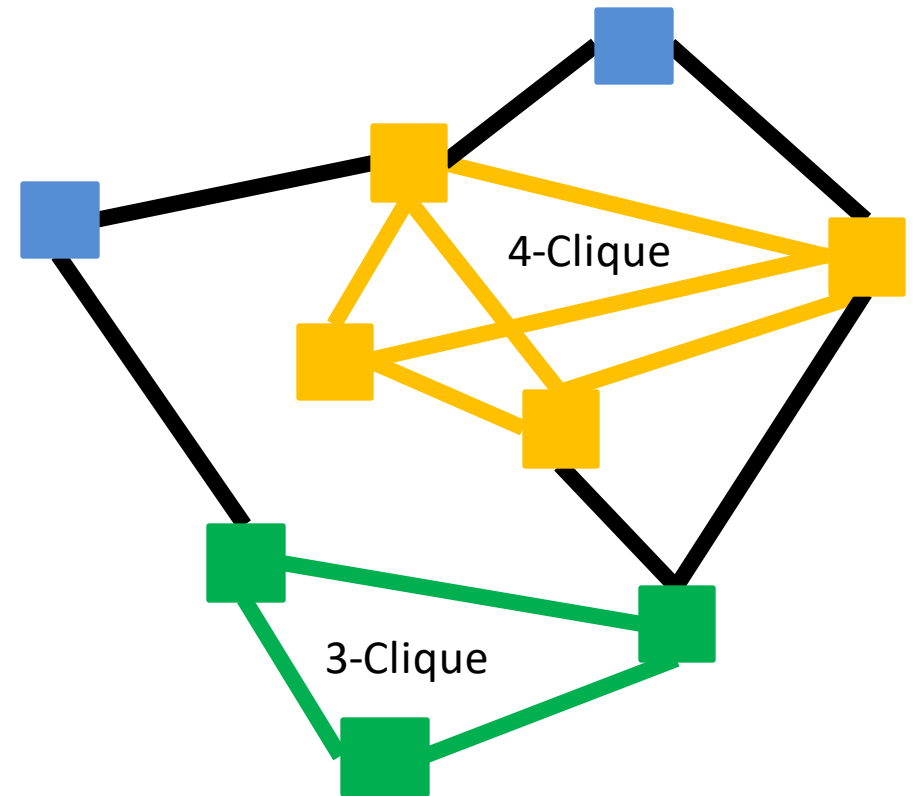
Remember: $kIndSet \leq_p kVertCov$



k -Clique Problem

Given a graph G and a number k , is there a *clique* of size k ?

- Clique: A complete subgraph

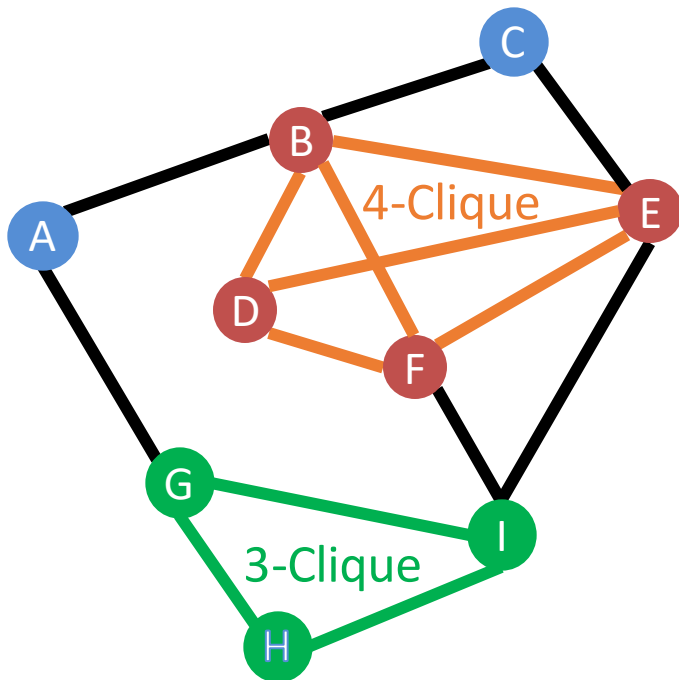


k -Clique is NP-Complete

1. Show that it belongs to NP
 - Give a polynomial time verifier
2. Show it is NP-Hard
 - Give a reduction from a known NP-Hard problem
 - We will show $3SAT \leq_p kClique$

k -Clique is in NP

- **Show:** For any graph G :
 - There is a short certificate (“solution”) that G has a k -clique
 - The certificate can be checked efficiently (in polynomial time)



Graph G

Suppose $k = 4$

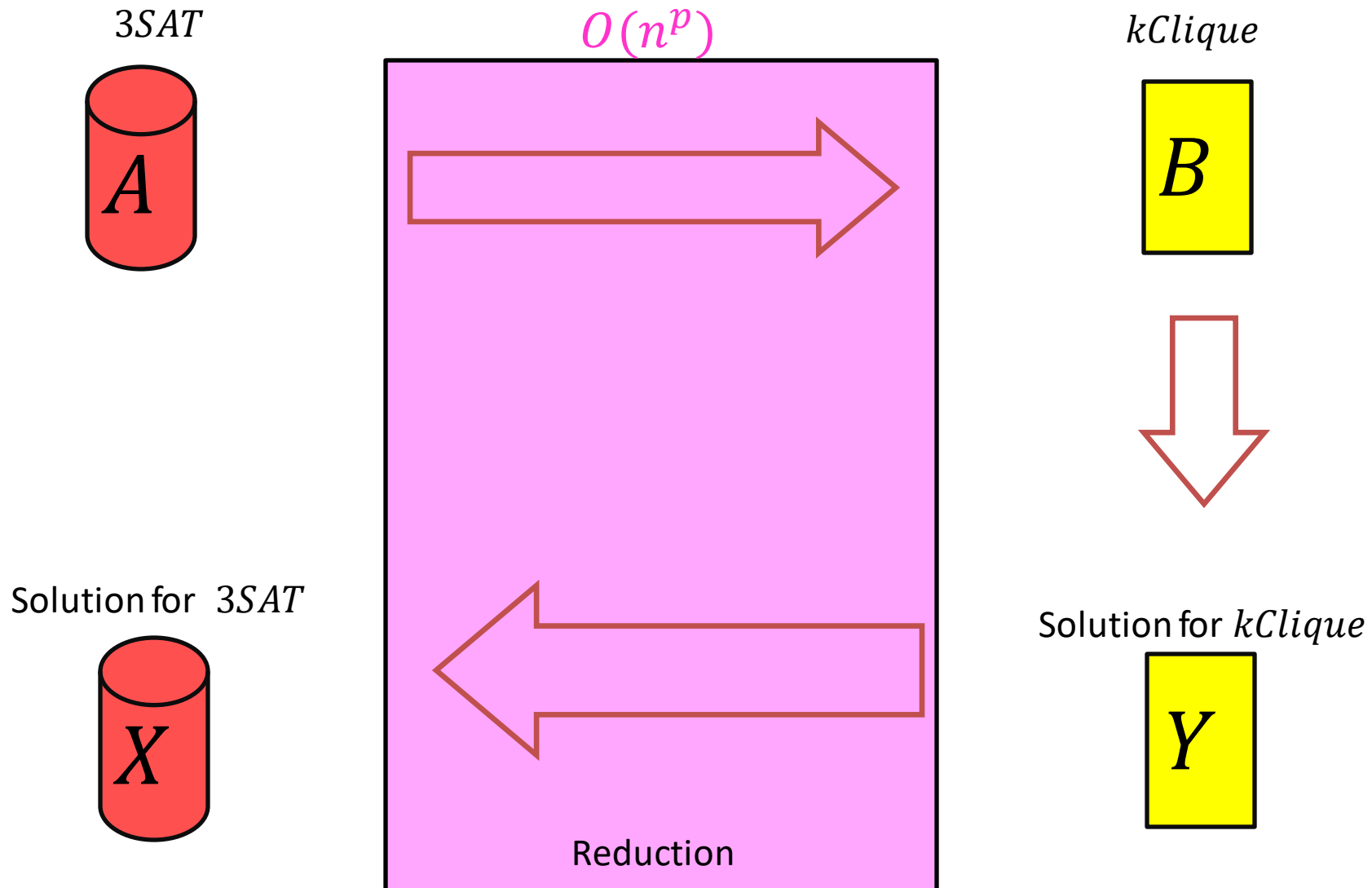
Certificate for G : $S = \{B, D, E, F\}$
(nodes in the k -clique)

Checking the certificate:

- Check that $|S| = k$ $O(k) = O(|V|)$
- Check that every pair of nodes in S share an edge $O(k^2) = O(|V|^2)$

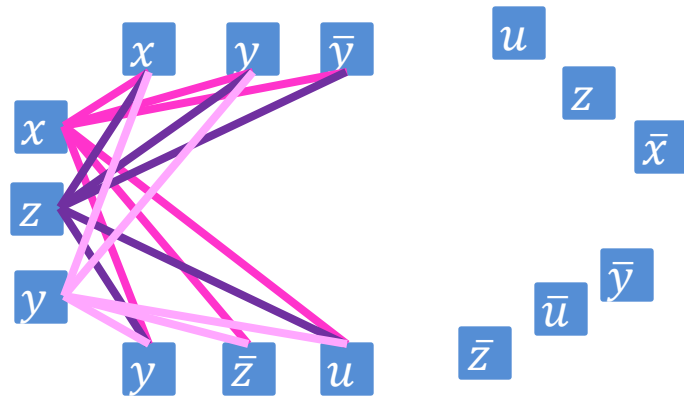
Total time: $O(|V|^2) = \text{poly}(|V| + |E|)$

$3SAT \leq_p kClique$



Instance of 3SAT to Instance of k Clique

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



(also do this for the other clauses, omitted due to clutter)

For each clause, produce a node for each of its three variables

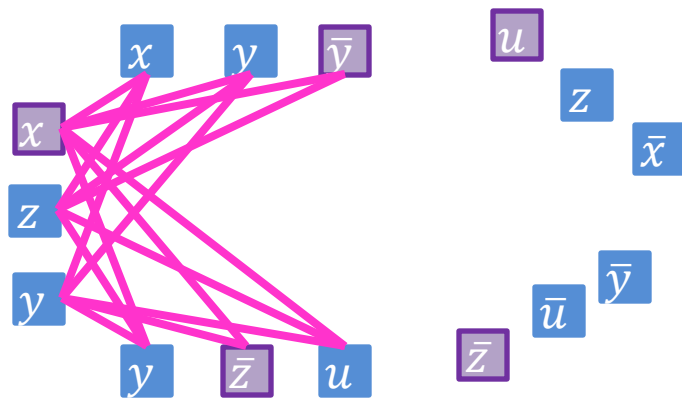
Connect each node to all non-contradictory nodes in the other clauses (i.e., anything that's not its negation)

Let k = number of clauses

There is a k -Clique in this graph **iff** there is a satisfying assignment

k Clique \Rightarrow Satisfying Assignment

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

There are k triplets in the graph, and no two nodes in the same triplet are adjacent

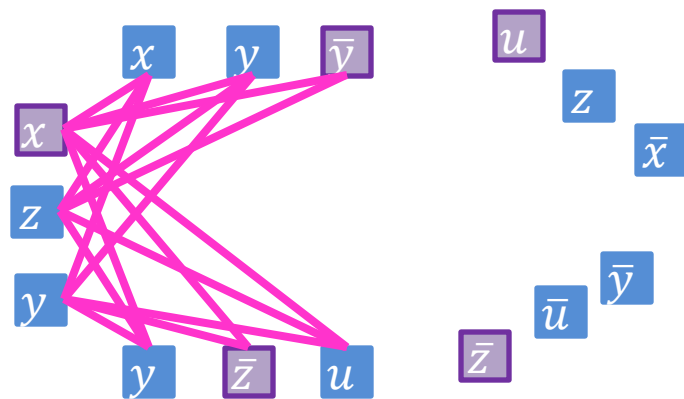
To have a k -Clique, must have one node from each triplet

Cannot select a node for both a variable and its negation

Therefore selection of nodes is a satisfying assignment

Satisfying Assignment $\Rightarrow k$ Clique

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



x = true
y = false
z = false
u = true

Select one node for a true variable from each clause

There will be k nodes selected

We can't select both a node and its negation

All nodes will be non-contradictory, so they will be pairwise adjacent

$3SAT \leq_p kClique$

