CS4102 Algorithms

Fall 2018

Today's Keywords

- Reductions
- Bipartite Matching
- Vertex Cover
- Independent Set
- NP-Completeness

CLRS Readings

- Chapter 34

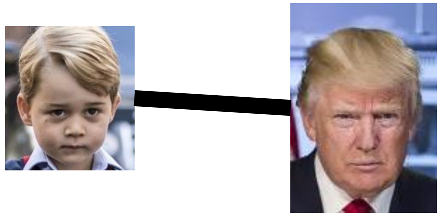
Homeworks

- HW8 due Friday 11/30 at 11pm
  - Written (use LaTeX)
  - Graphs

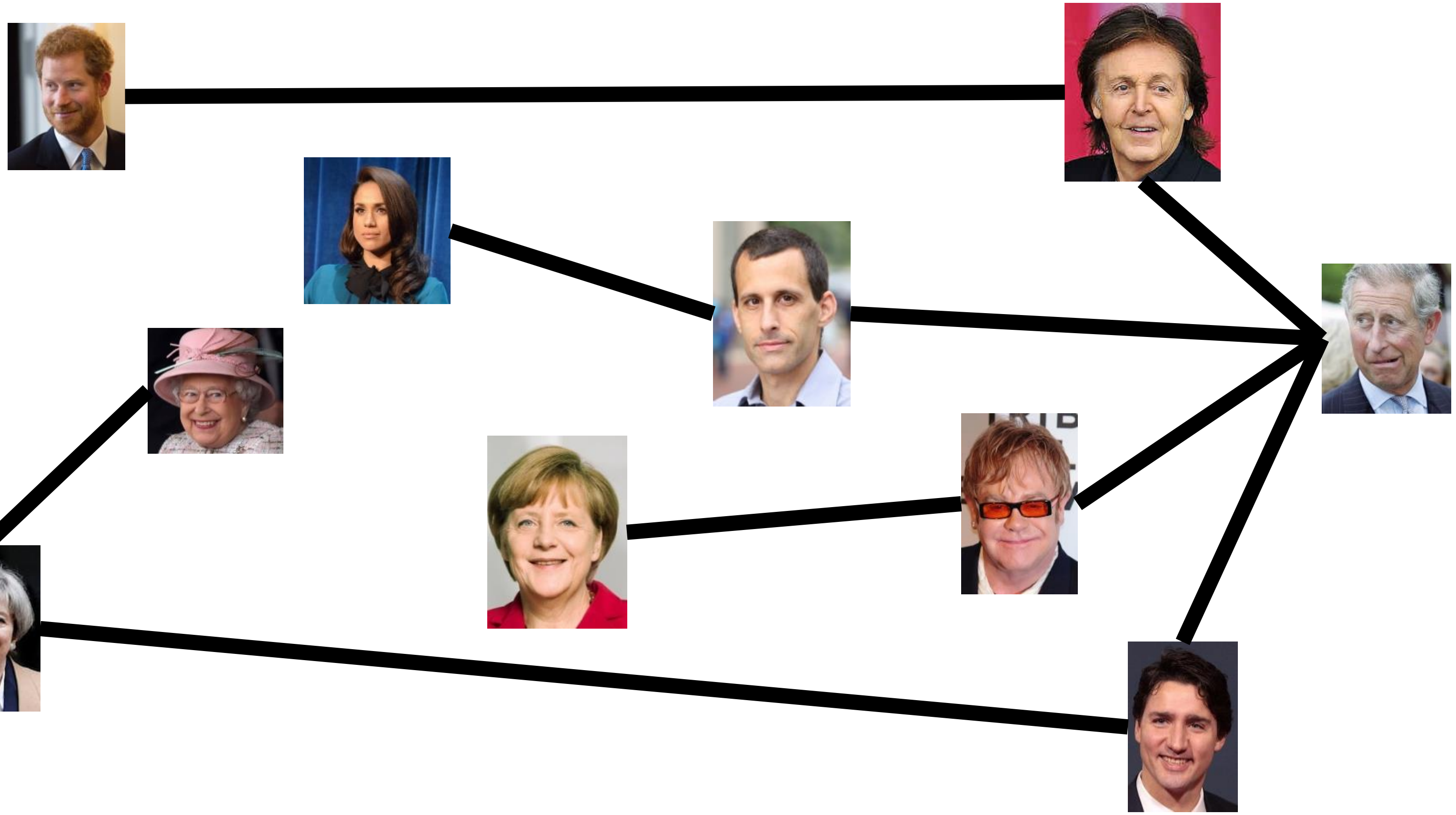
Reductions

- 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

Party Problem



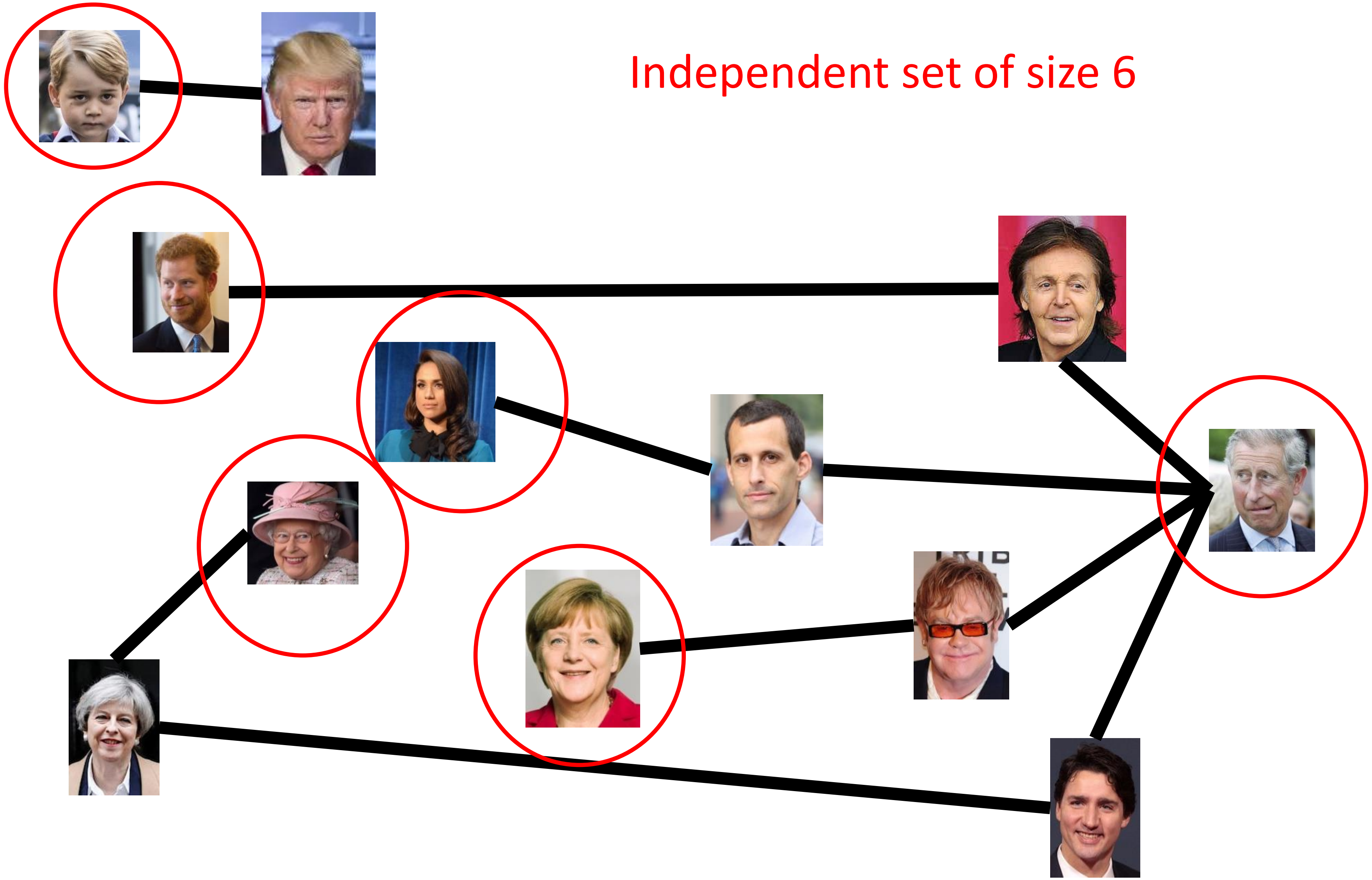
Draw Edges between people who don’t get along
Find the maximum number of people who get along



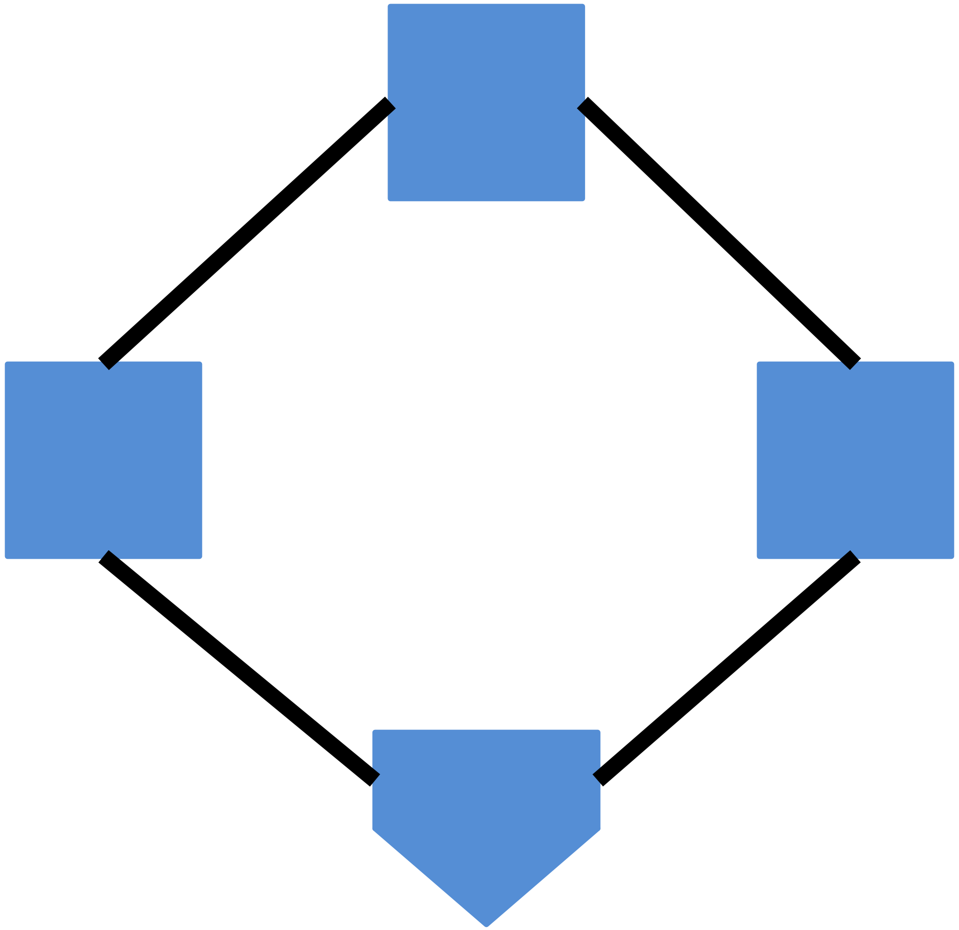
Maximum Independent Set

- 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

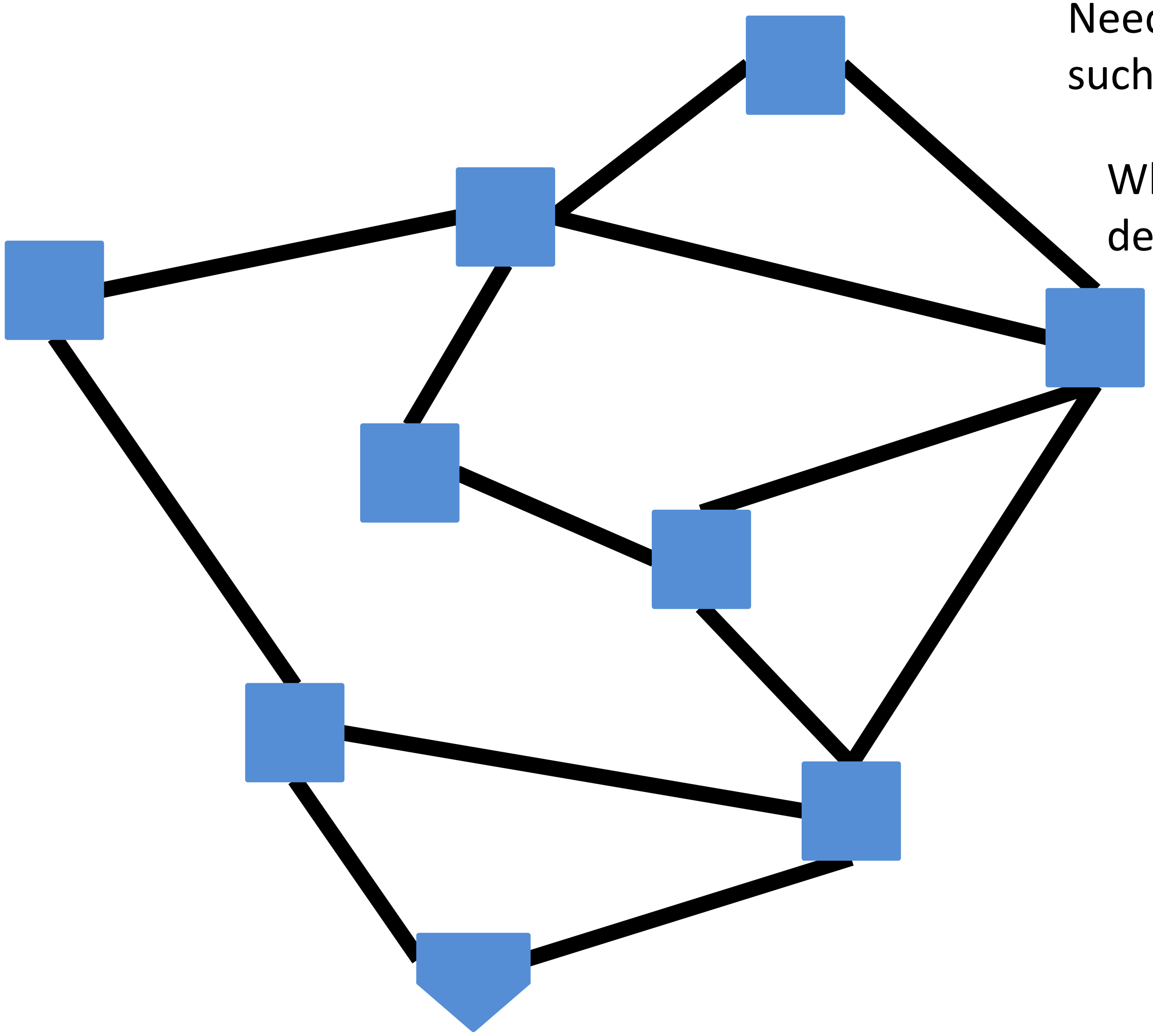
Example



Generalized Baseball



Generalized Baseball



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

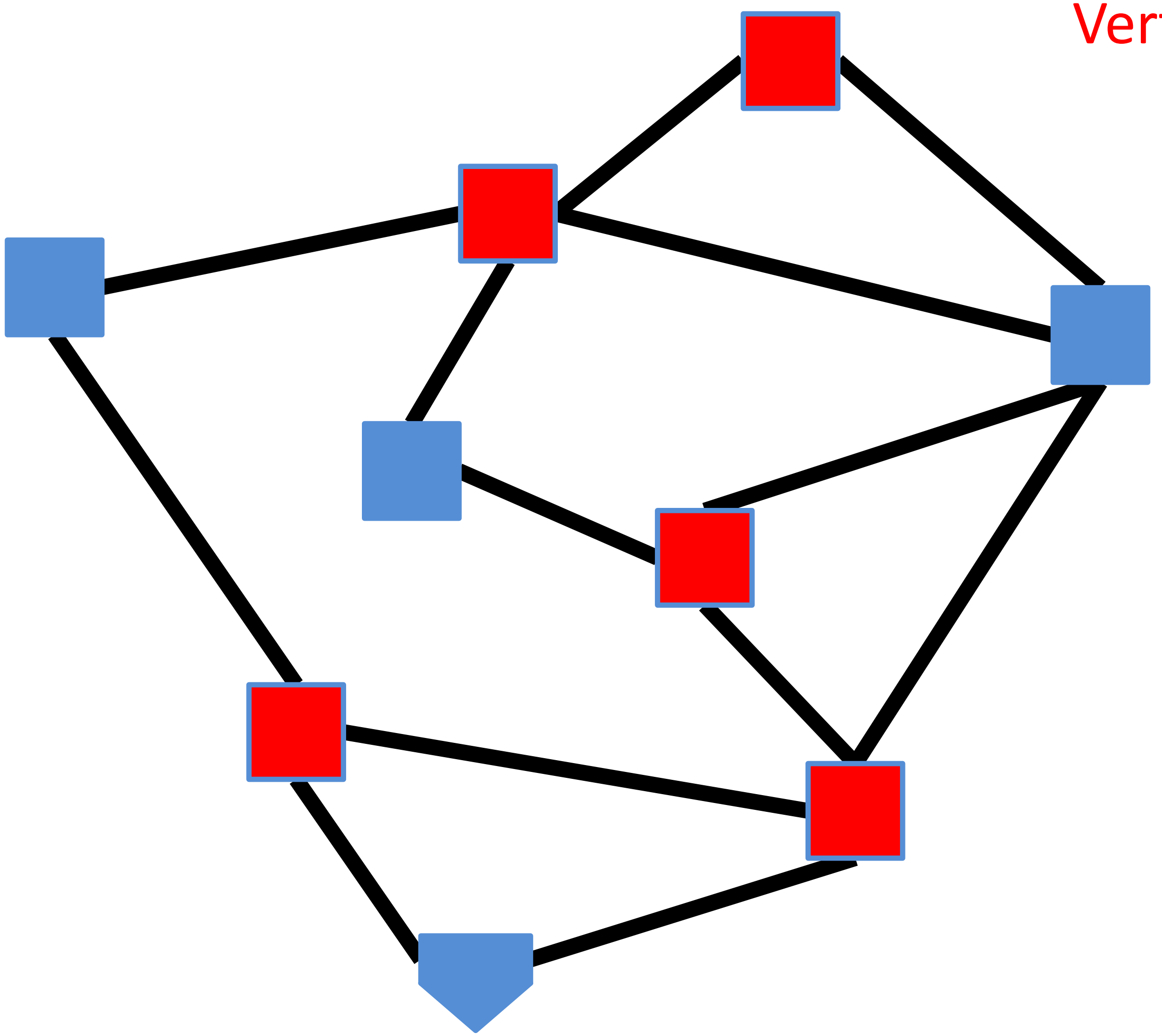
What's the fewest number of defenders needed?

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

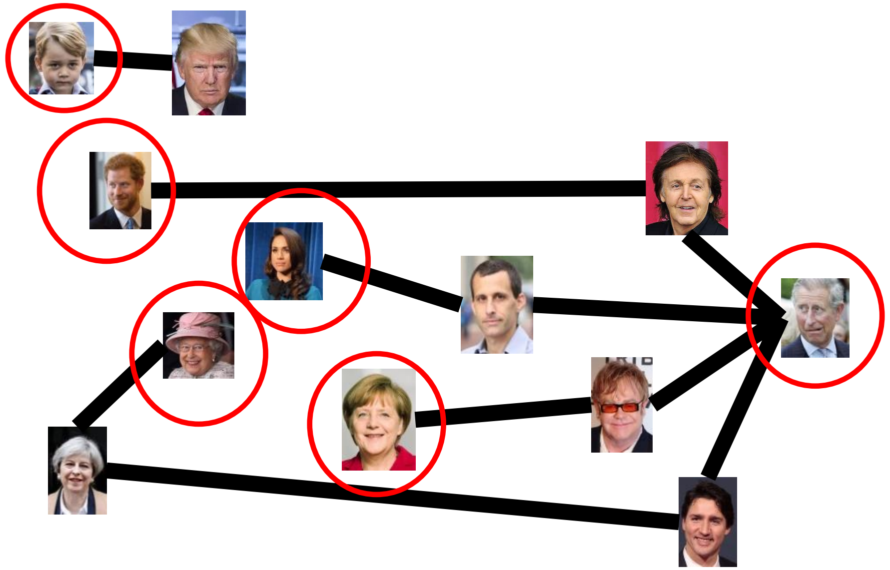
Vertex cover of size 5



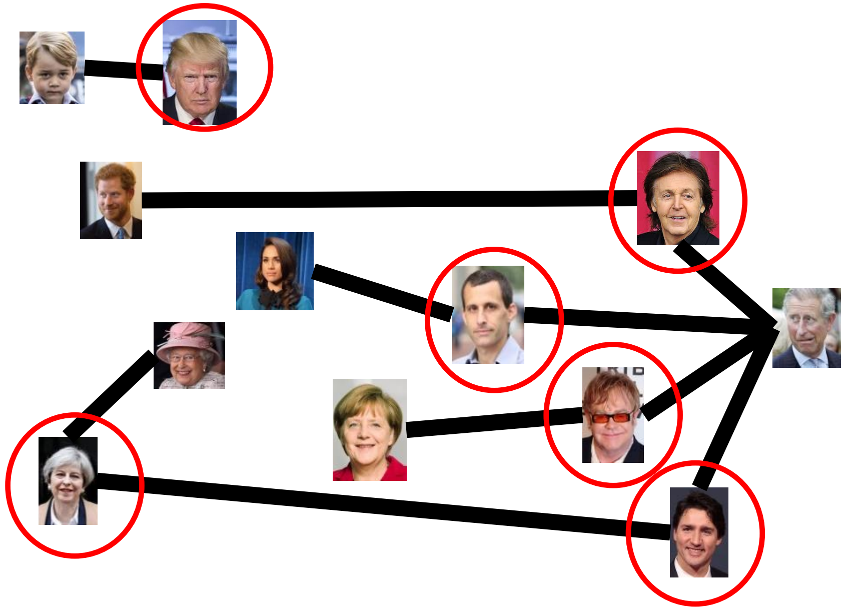
Reduction Idea

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

Independent Set



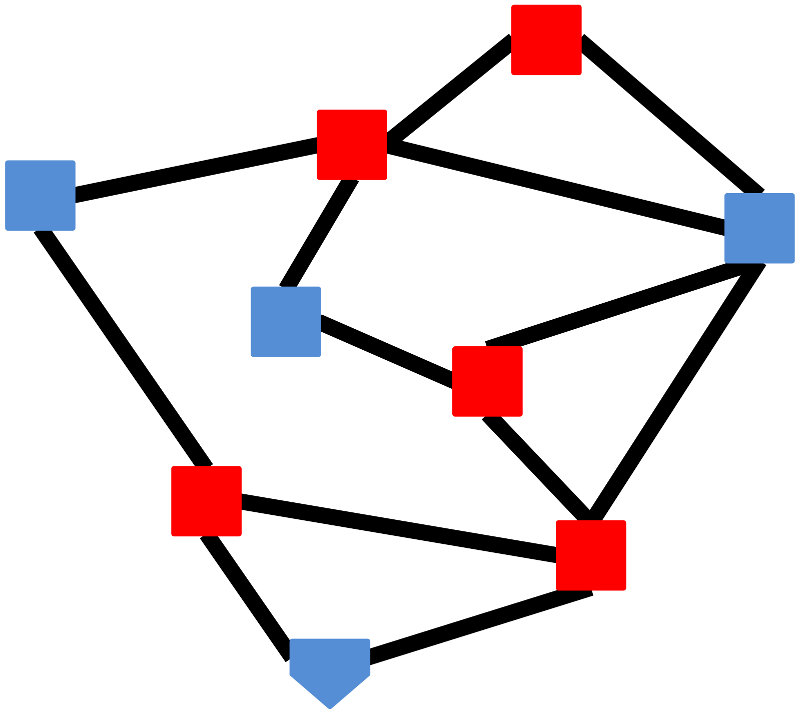
Vertex Cover



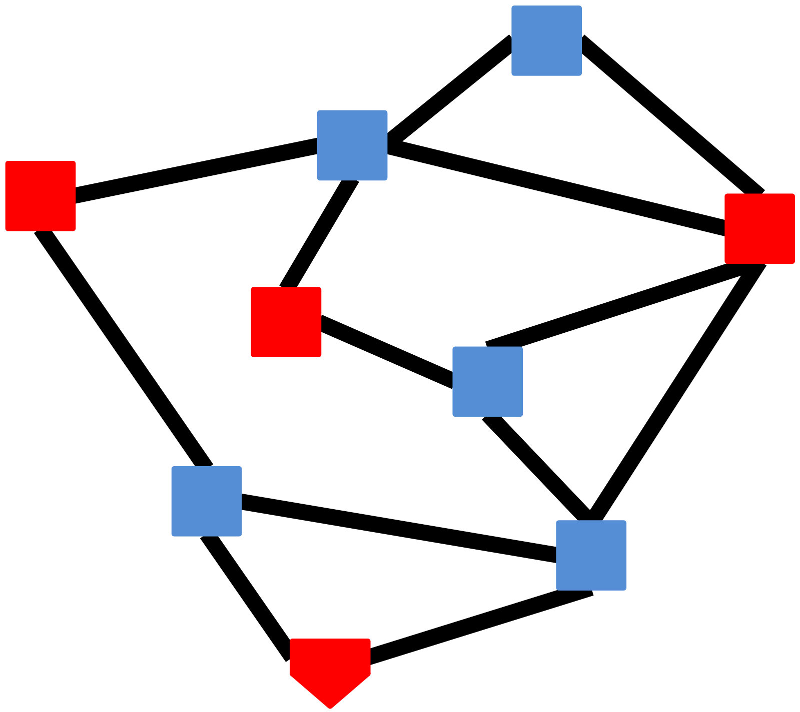
Reduction Idea

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

Vertex Cover



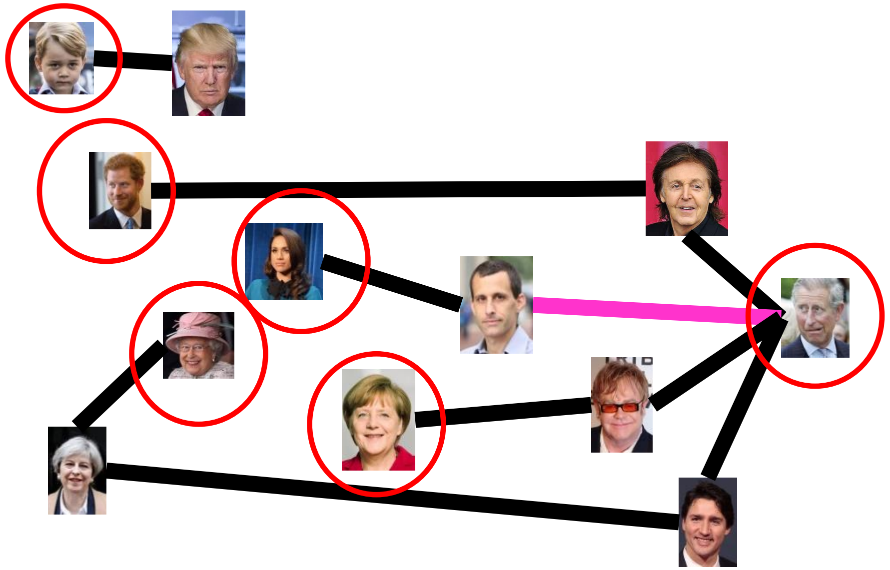
Independent Set



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 $(x, y) \in E$

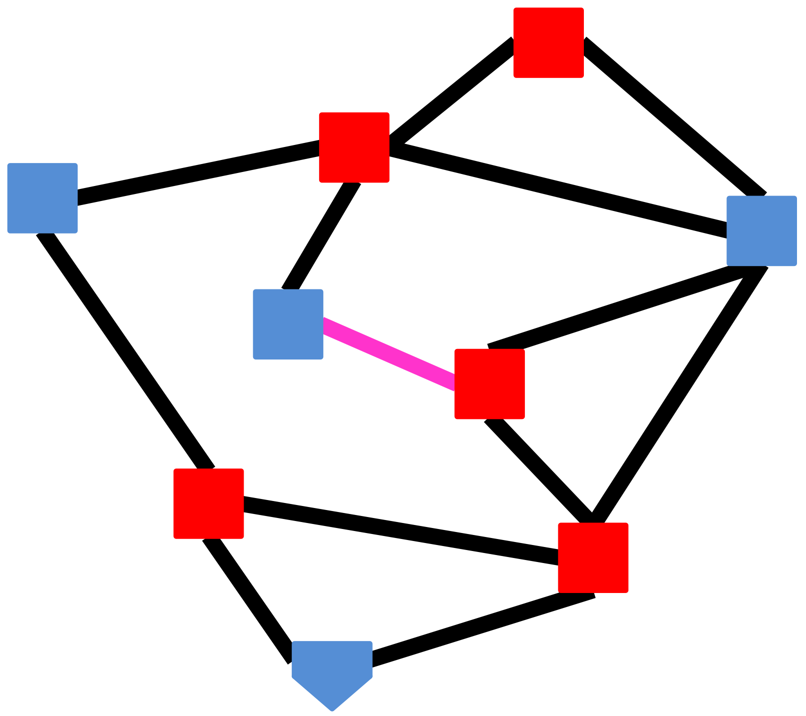
If $x \in S$ then $y \notin S$, because o.w. S would not be an independent set

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

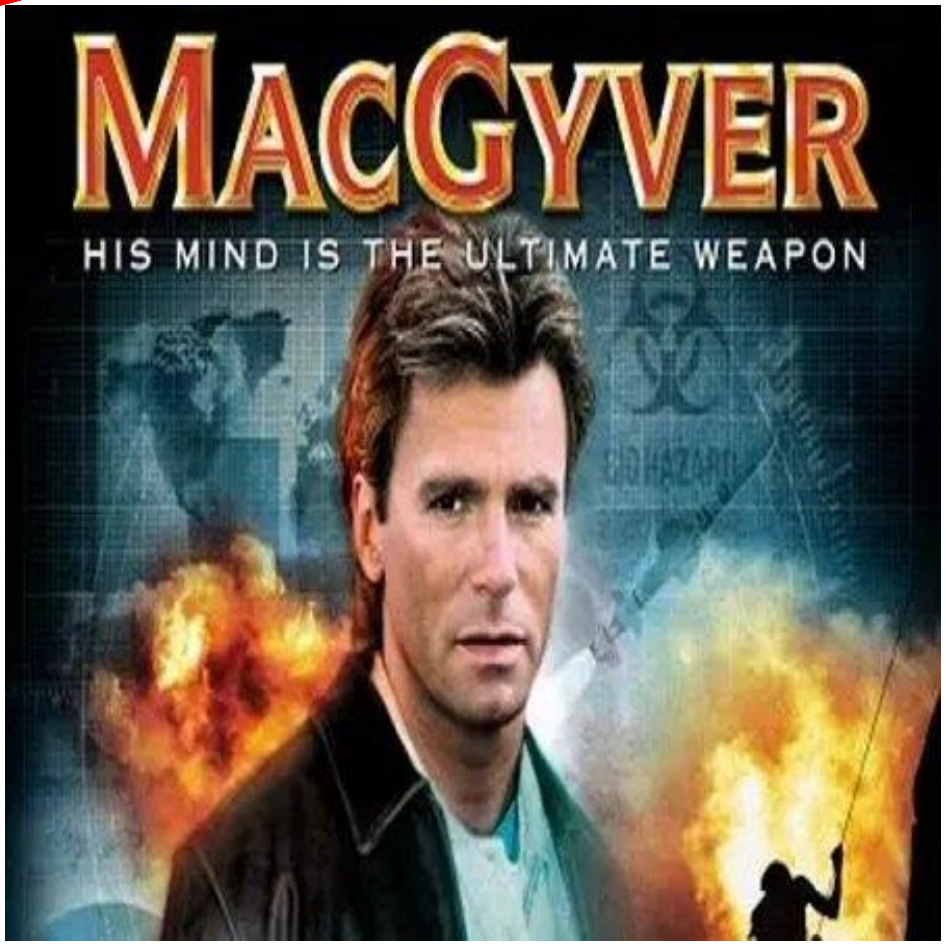
Therefore x and y are not both in S,

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

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?

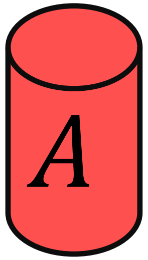
Solution for $\boldsymbol{B}$

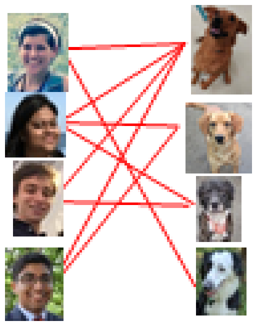
Alcohol, wood, matches

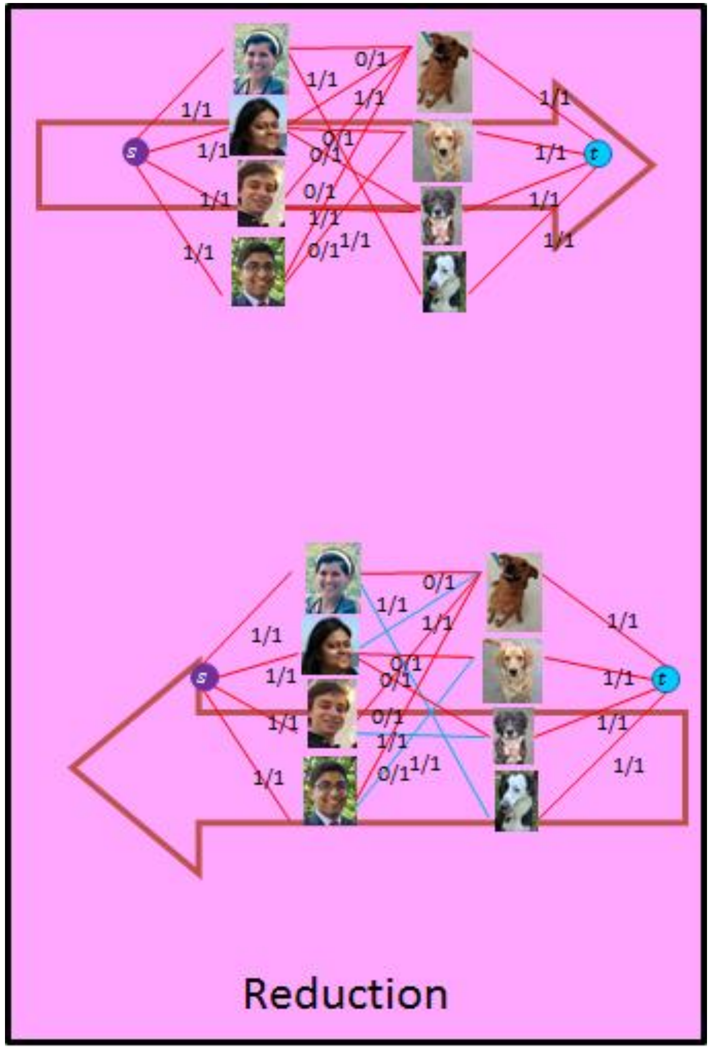
Bipartite Matching Reduction

Problem we don't know how to solve

Bipartite Matching

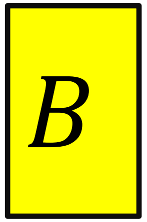


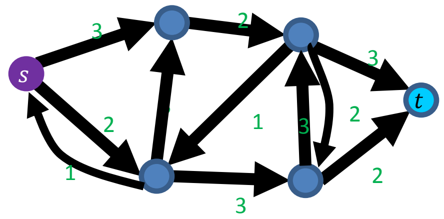


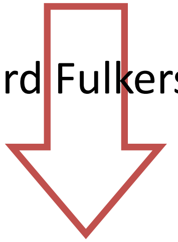


Problem we do know how to solve

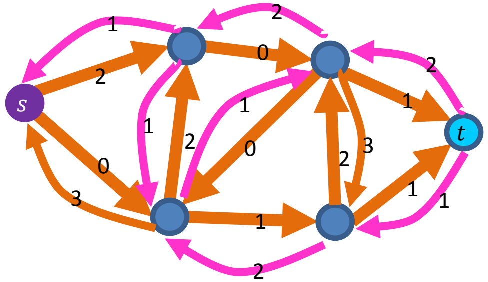
Max Flow



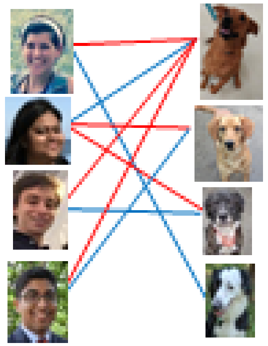




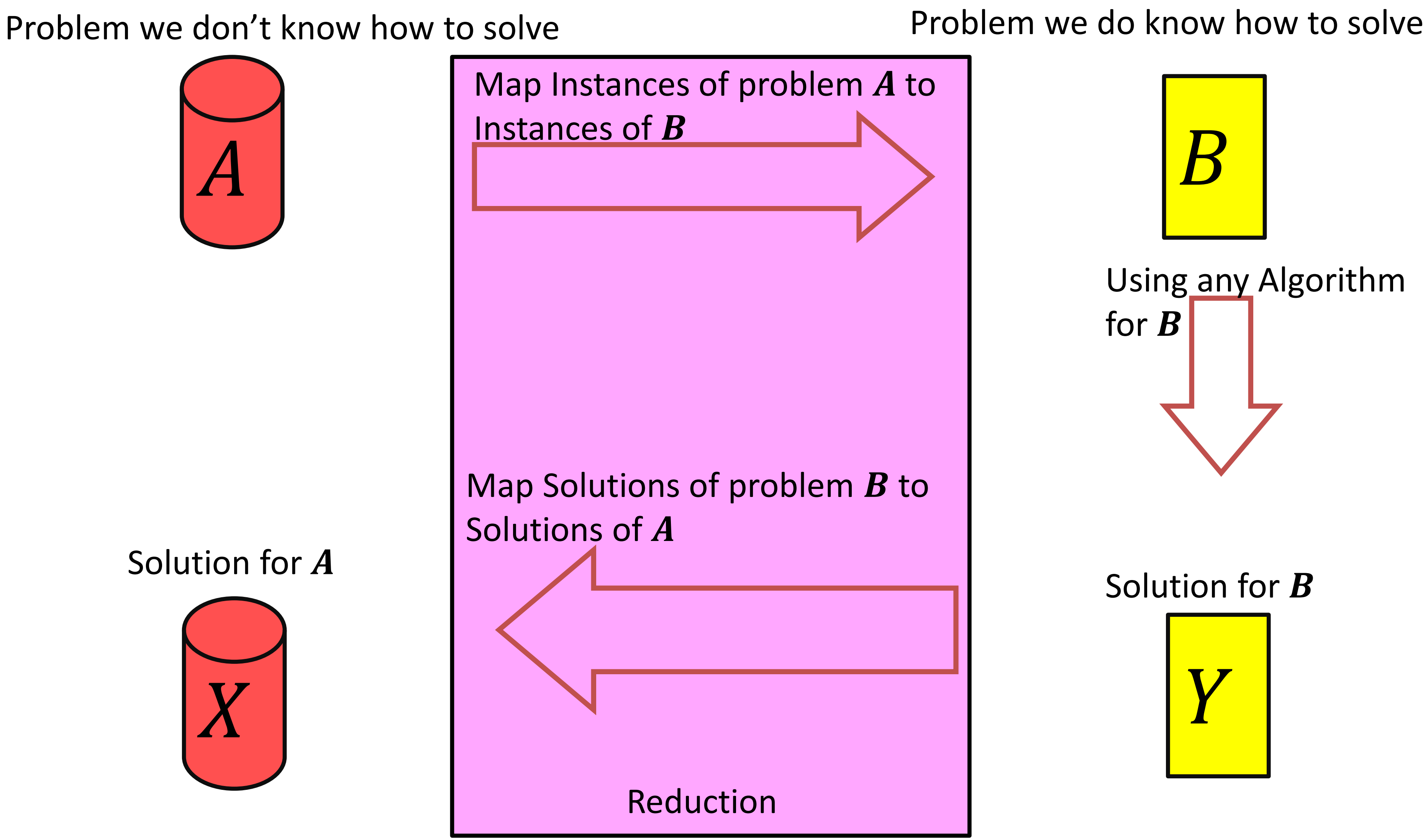
Solution for $\boldsymbol{B}$



Solution for $\boldsymbol{A}$



In General: Reduction



Worst-case lower-bound Proofs

Opening a door

A

Problem **A**

reduces to

B

Problem **B**

Lighting a fire

Alcohol, wood, matches

Y

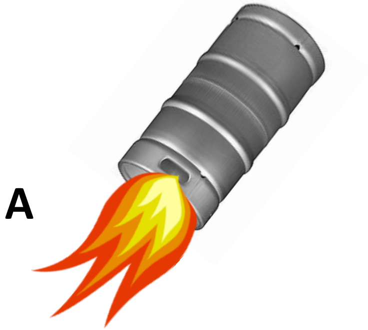
Algorithm for **B**

can be used to make

X

Algorithm for **A**

Keg cannon battering ram



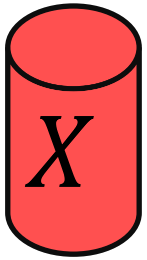
A is not a harder problem than B

$$\boldsymbol{A \leq B}$$

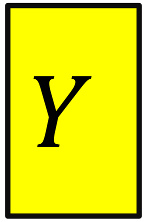
The name "reduces" is confusing: it is in the *opposite* direction of the making

Proof of Lower Bound by Reduction

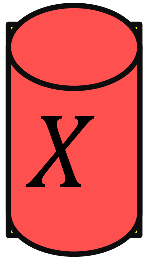
<u>To Show: Y is slow</u>



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



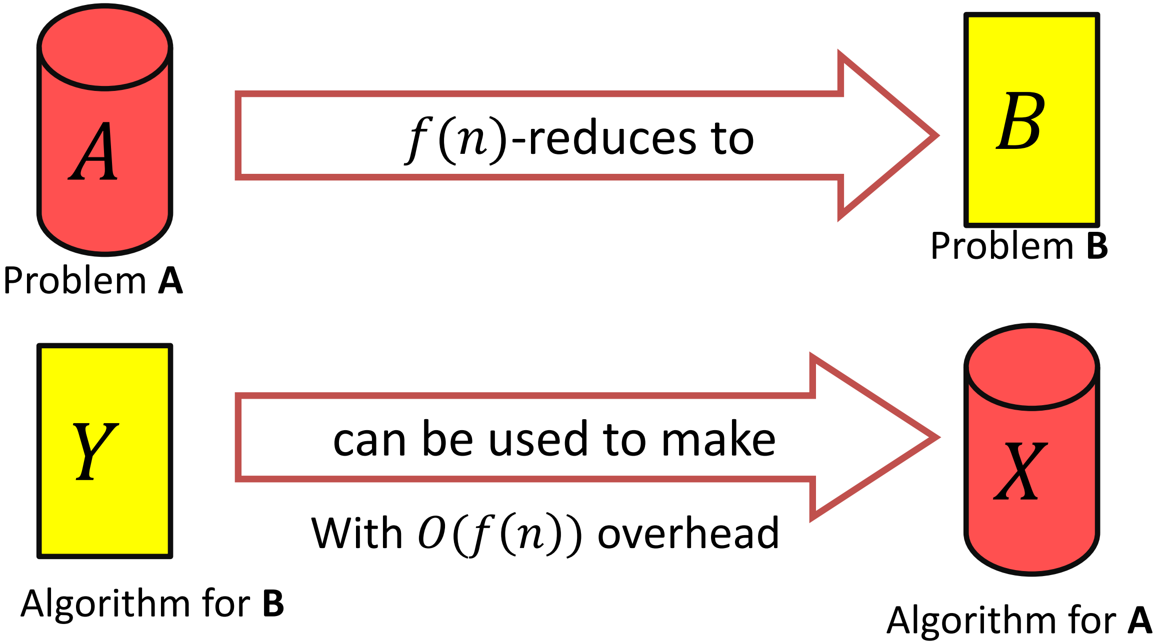
2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)



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

Reduction Proof Notation



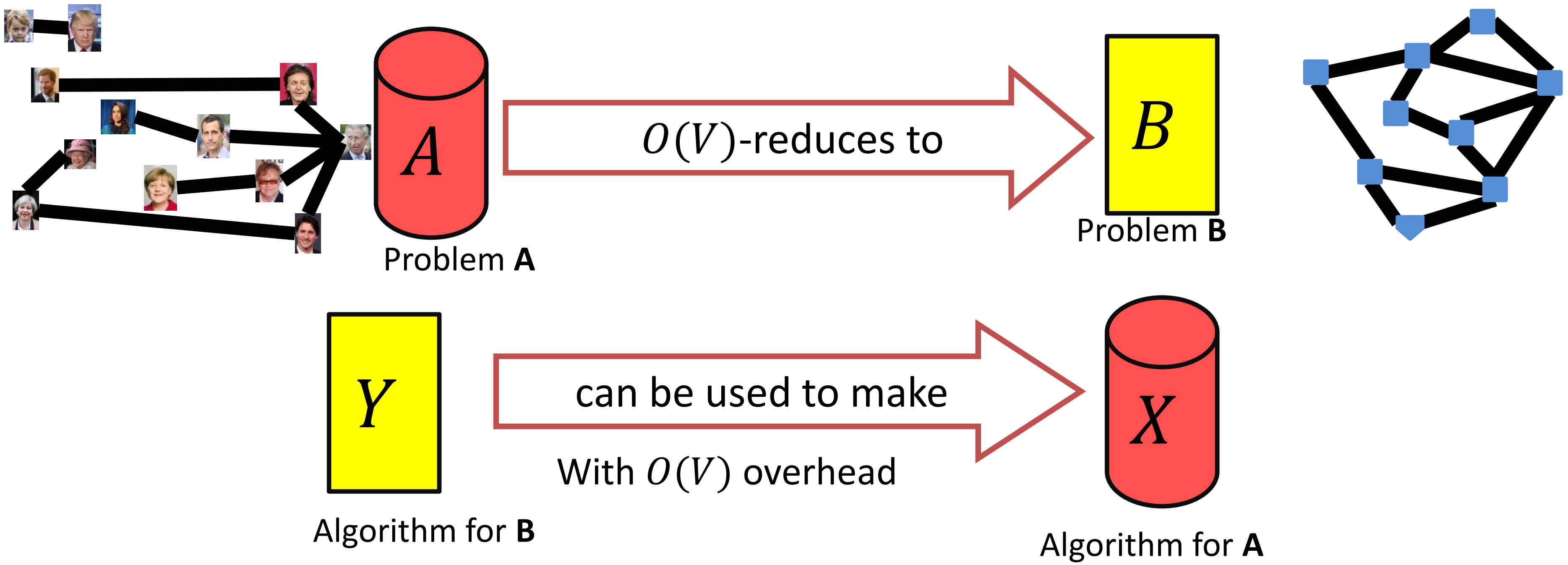
A is not a harder problem than B

$$A \le B$$

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

$$A \le_{f(n)} B$$

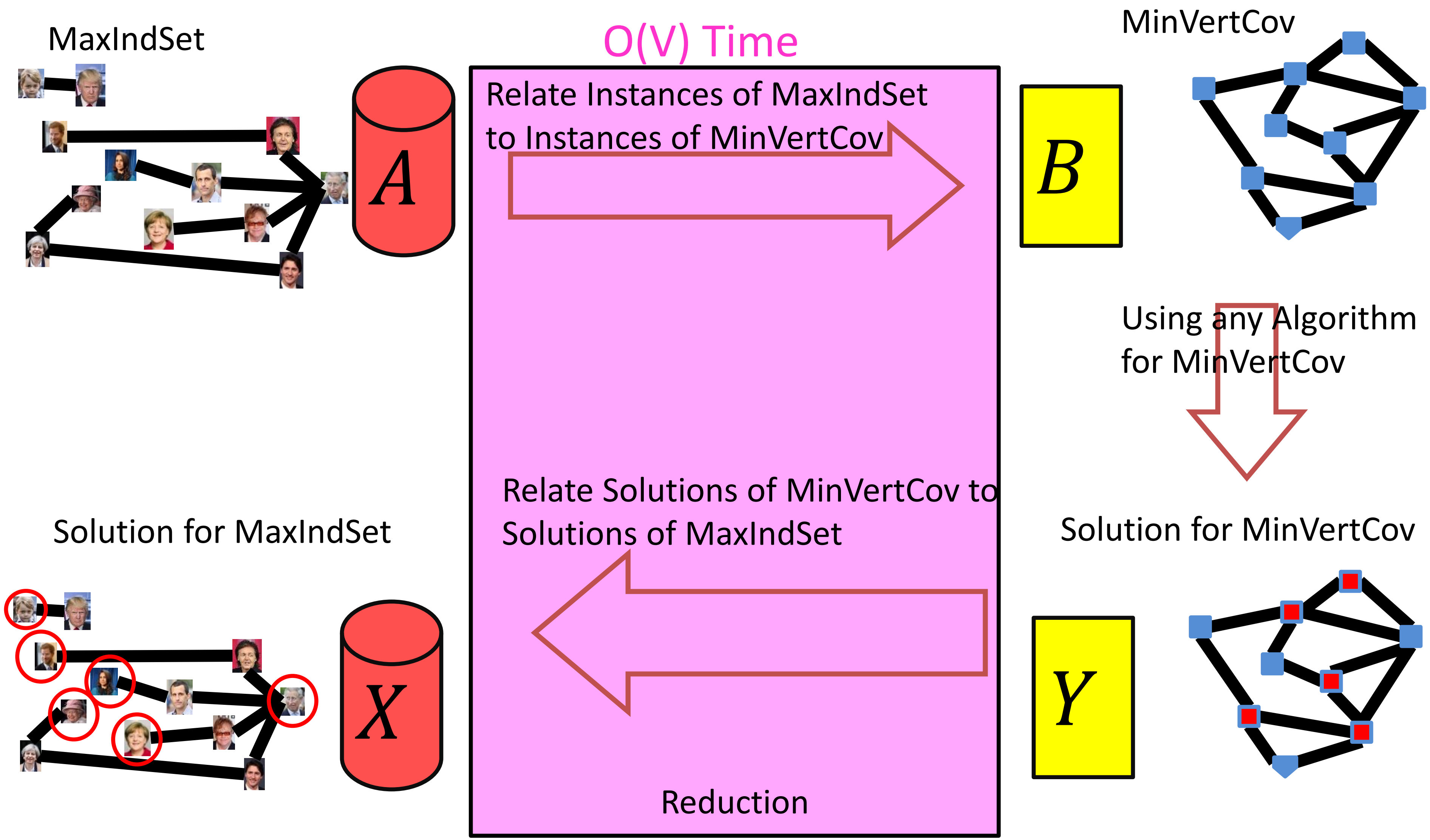
MaxIndSet$\leq_V$MinVertCov



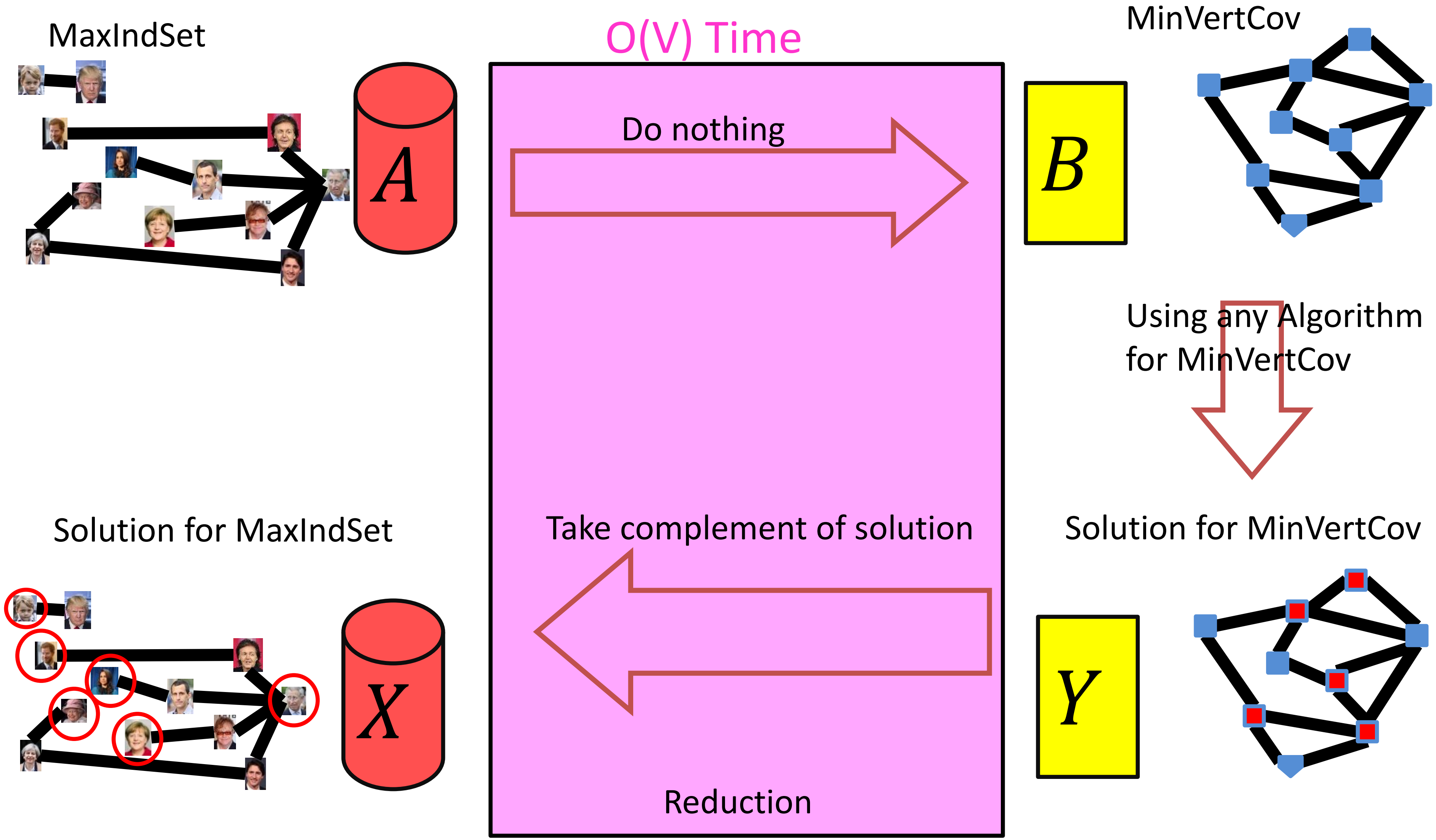
If A requires time $\Omega(f(n))$ time then B also requires $\Omega(f(n))$ time

$$A \leq_V B$$

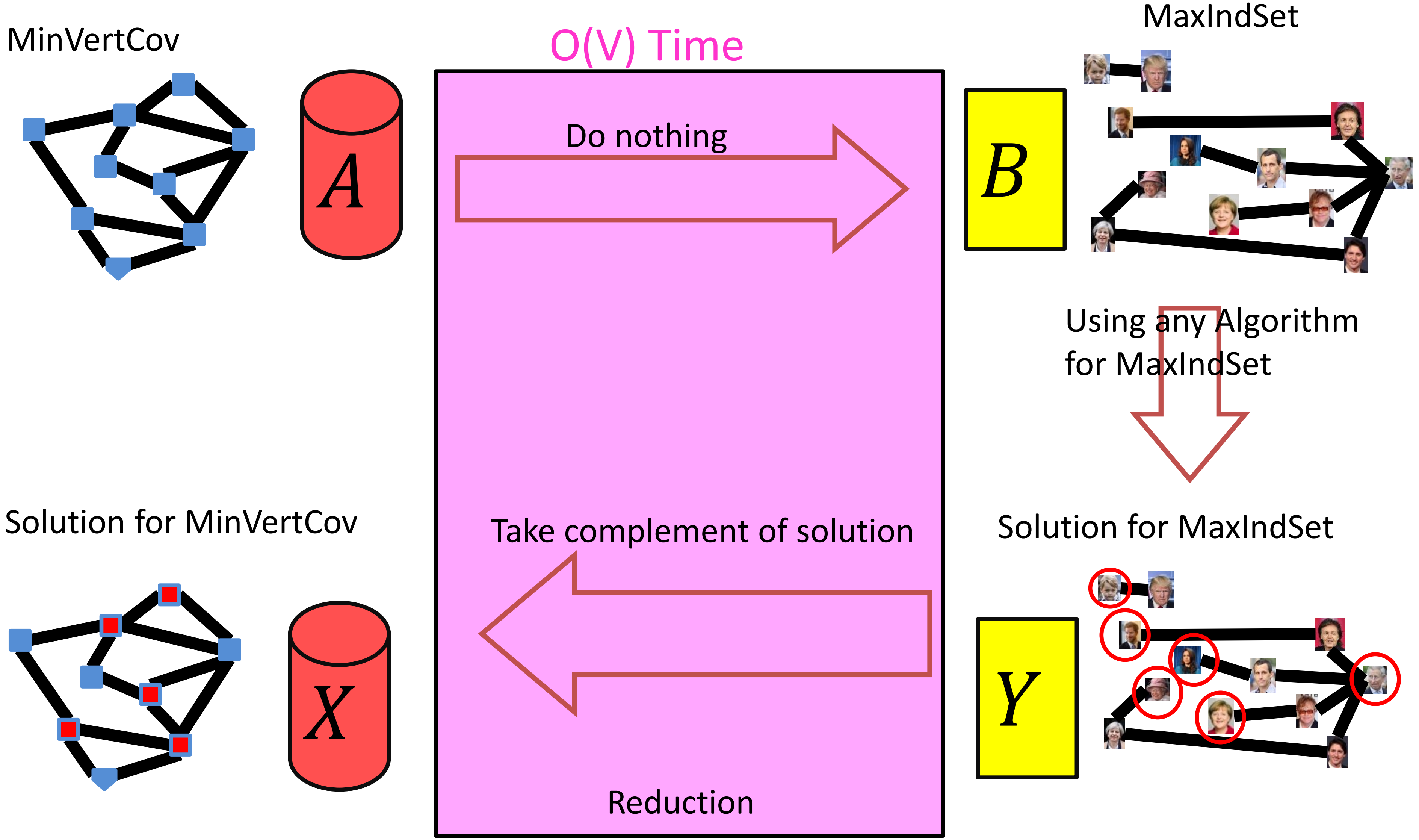
We need to build this Reduction



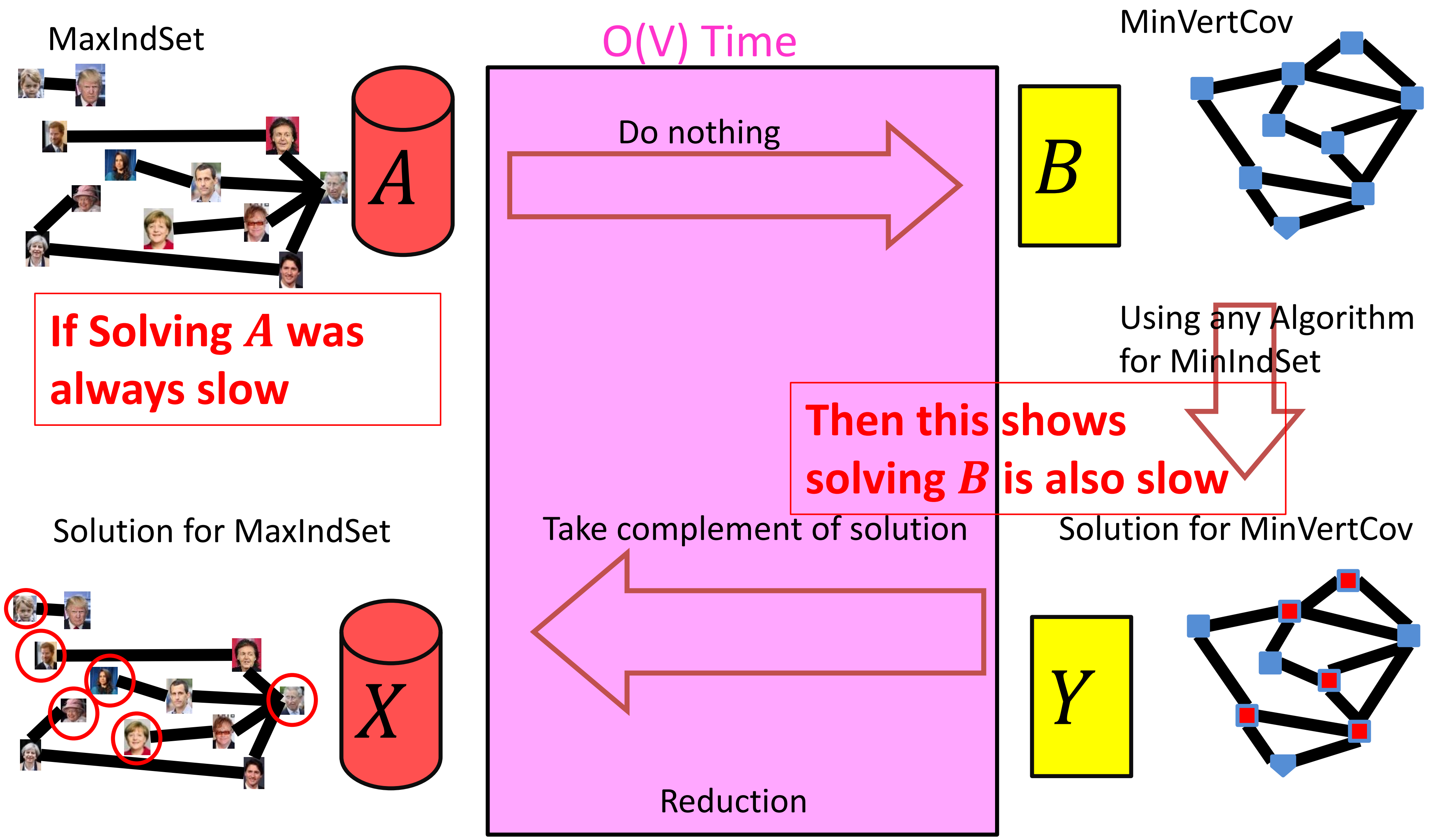
MaxVertCov V-Time Reducable to MinIndSet



MaxIndSet V-Time Reducable to MinVertCov

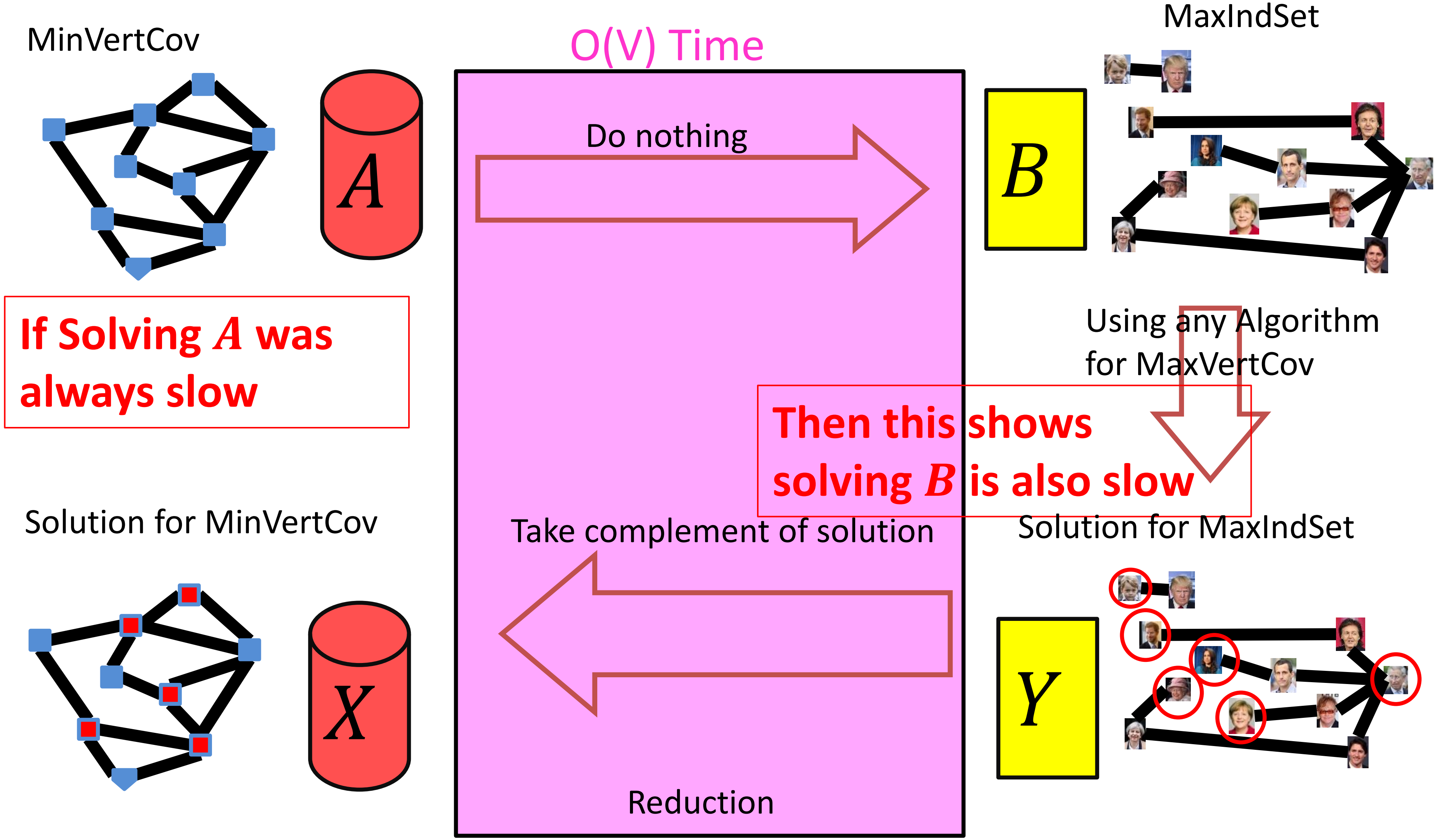


Corollary



MaxIndSet
O(V) Time
MinVertCov
A
Do nothing
B
If Solving A was always slow
Using any Algorithm for MinIndSet
Then this shows solving B is also slow
Solution for MaxIndSet
Take complement of solution
Solution for MinVertCov
X
Y
Reduction

Corollary



MinVertCov
O(V) Time
MaxIndSet
Do nothing
A
B
If Solving A was always slow
Using any Algorithm for MaxVertCov
Then this shows solving B is also slow
Solution for MinVertCov
Take complement of solution
Solution for MaxIndSet
X
Y
Reduction

Conclusion

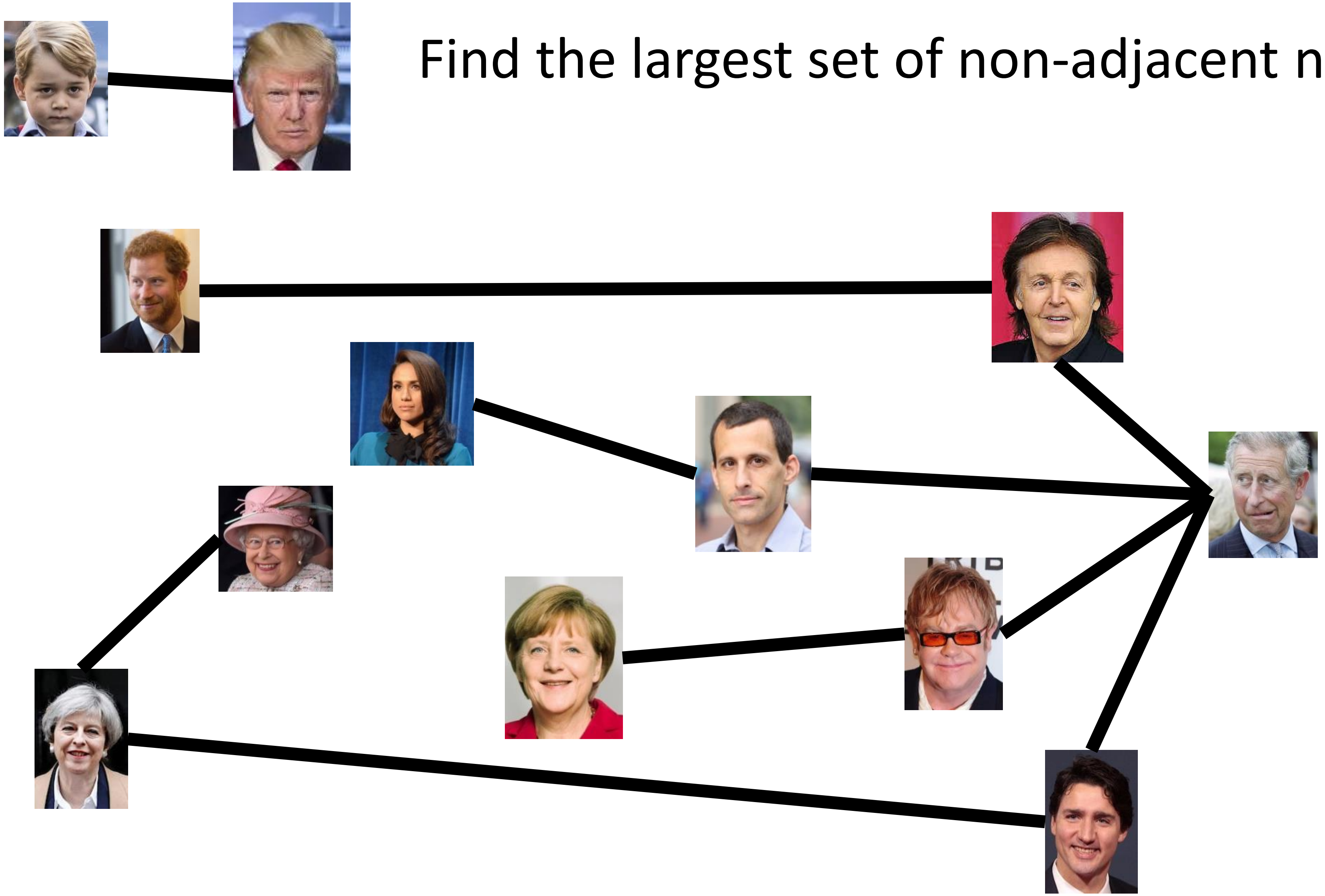
- MaxIndSet and MinVertCov are either both fast, or both slow
  - Spoiler alert: We don’t know which!
    - (But we think they’re both slow)
  - Both problems are NP-Complete

Mid-class warm up:

What is a Decision Problem?

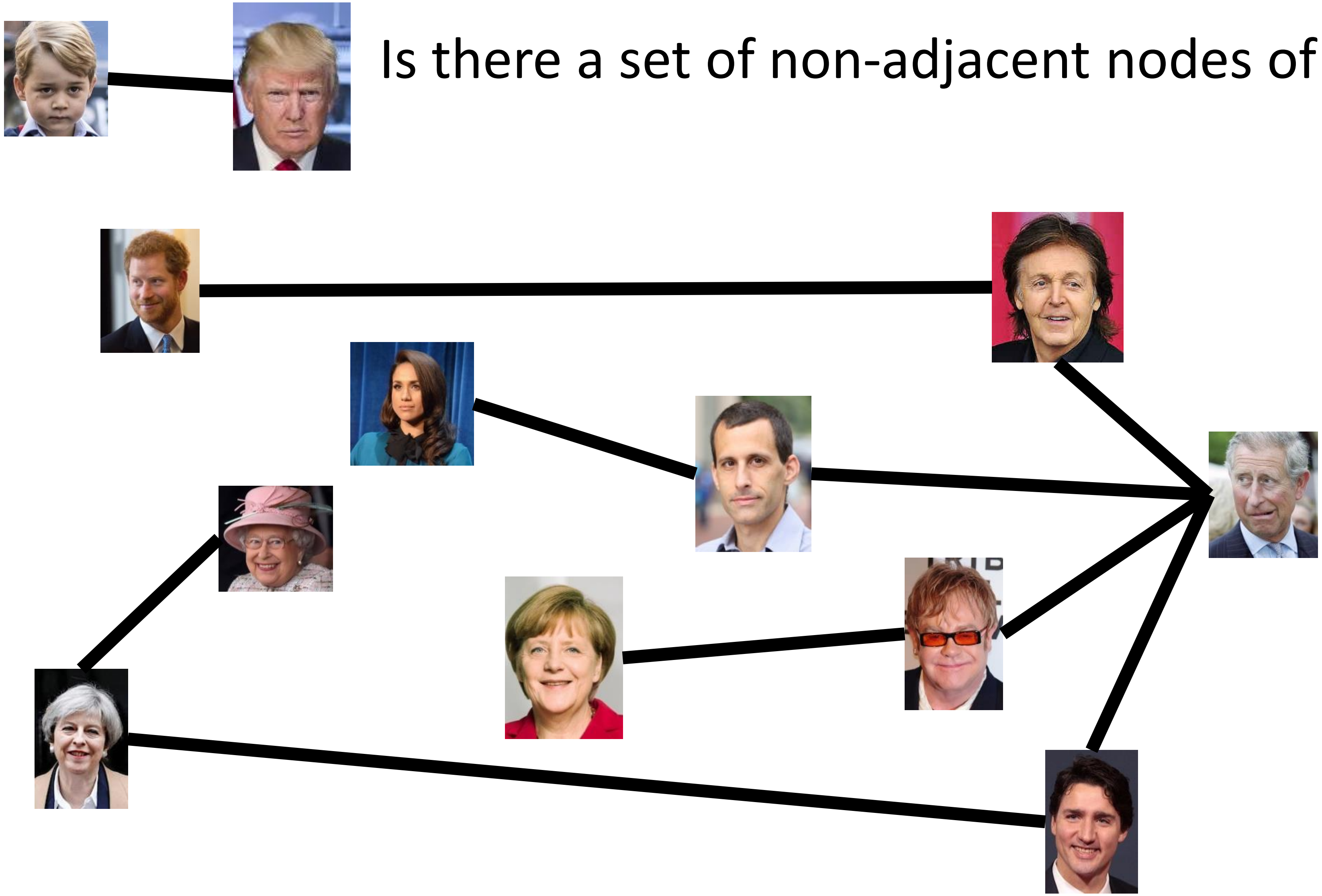
Max Independent Set

Find the largest set of non-adjacent nodes



k Independent Set

Is there a set of non-adjacent nodes of size k?



Maximum Independent Set

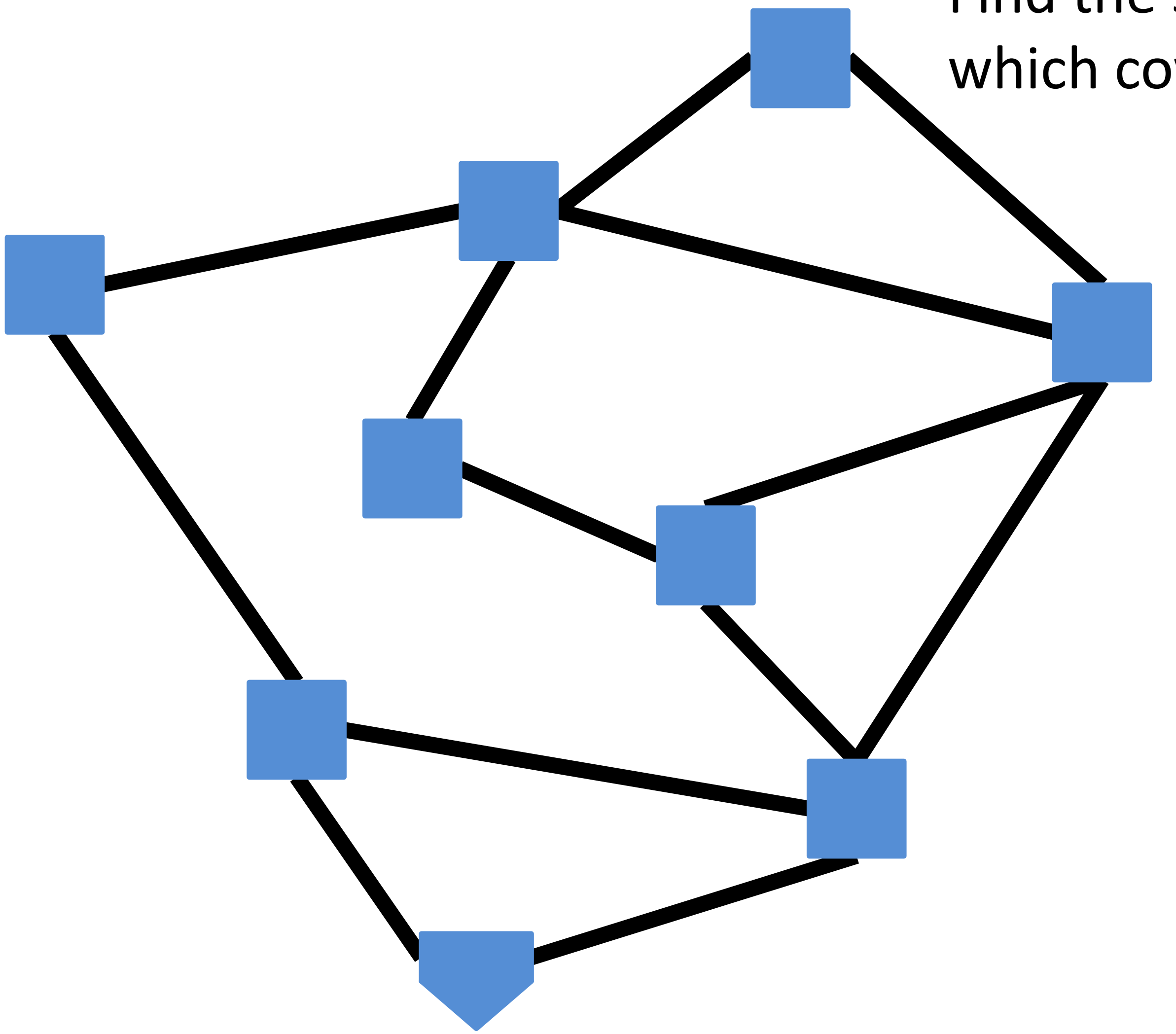
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

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 $\boldsymbol{S}$ of size $\boldsymbol{k}$**

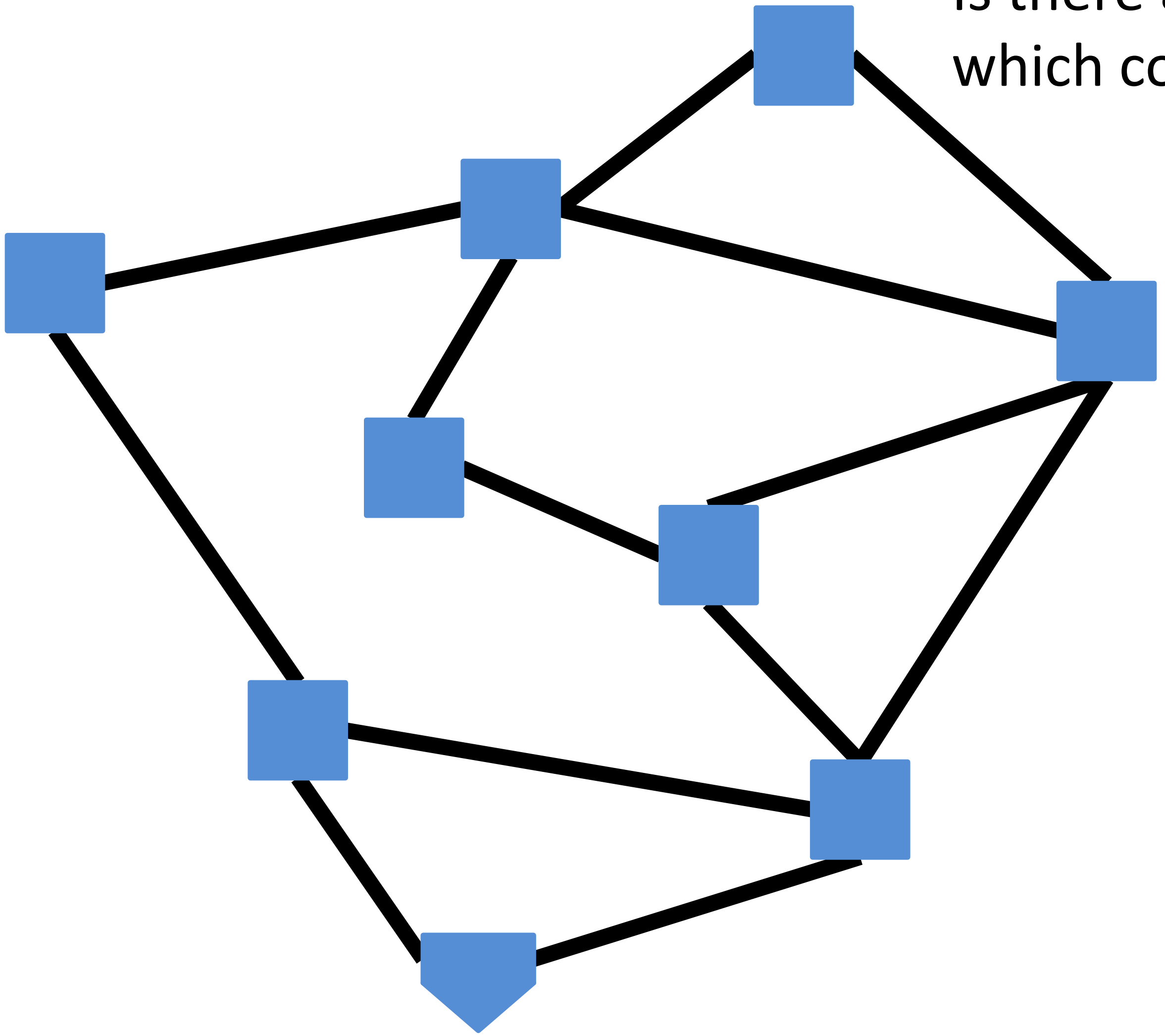
Min Vertex Cover

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 whether there is a vertex cover $\boldsymbol{C}$ of size $\boldsymbol{k}$**

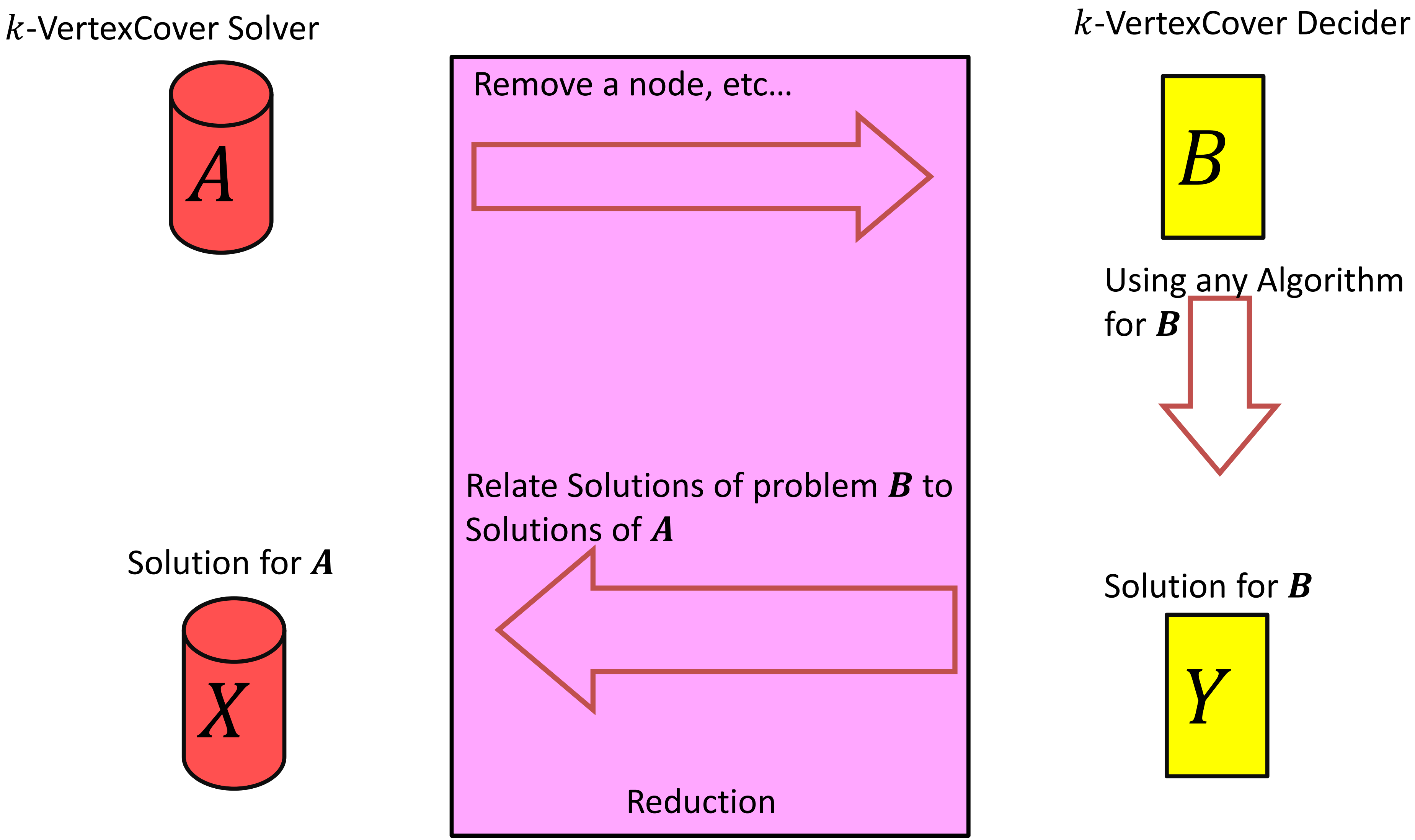
Problem Types

- Decision Problems:
  – Is there a solution?
    - Output is True/False
  – Is there a vertex cover of size k?
- Search Problems:
  – Find a solution
    - Output is complex
  – Give a vertex cover of size k
- Verification Problems:
  – Given a potential solution, is it valid?
    - Output is True/False
  – Is **this** a vertex cover of size k**?**

If we can solve this

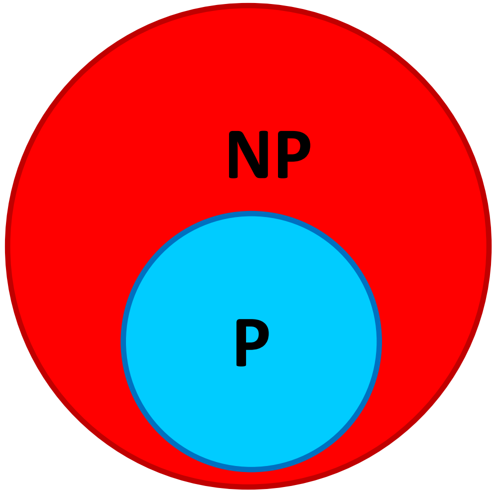
Then we can solve this

Reduction


k-VertexCover Solver
A
Solution for $\boldsymbol{A}$
X
Remove a node, etc…
Relate Solutions of problem $\boldsymbol{B}$ to Solutions of $\boldsymbol{A}$
Reduction
k-VertexCover Decider
B
Using any Algorithm for $\boldsymbol{B}$
Solution for $\boldsymbol{B}$
Y

P vs NP

- P
  - Deterministic Polynomial Time
  - Problems solvable in polynomial time
    - $O(n^p)$ for some number p
- NP
  - Non-Deterministic Polynomial Time
  - Problems verifiable in polynomial time
    - $O(n^p)$ for some number p
- 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

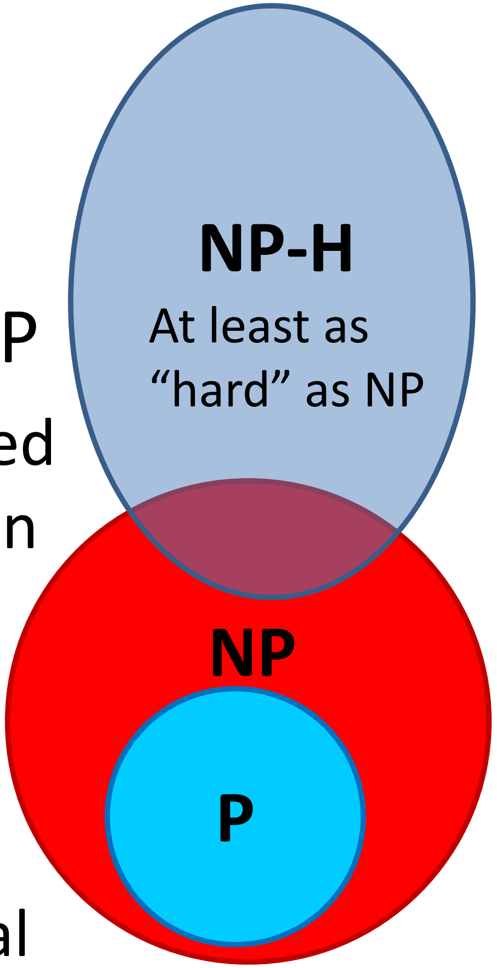
- To show: Given a potential solution, can we verify it in $O(n^p)$? $[n = V + E]$

How can we verify it?

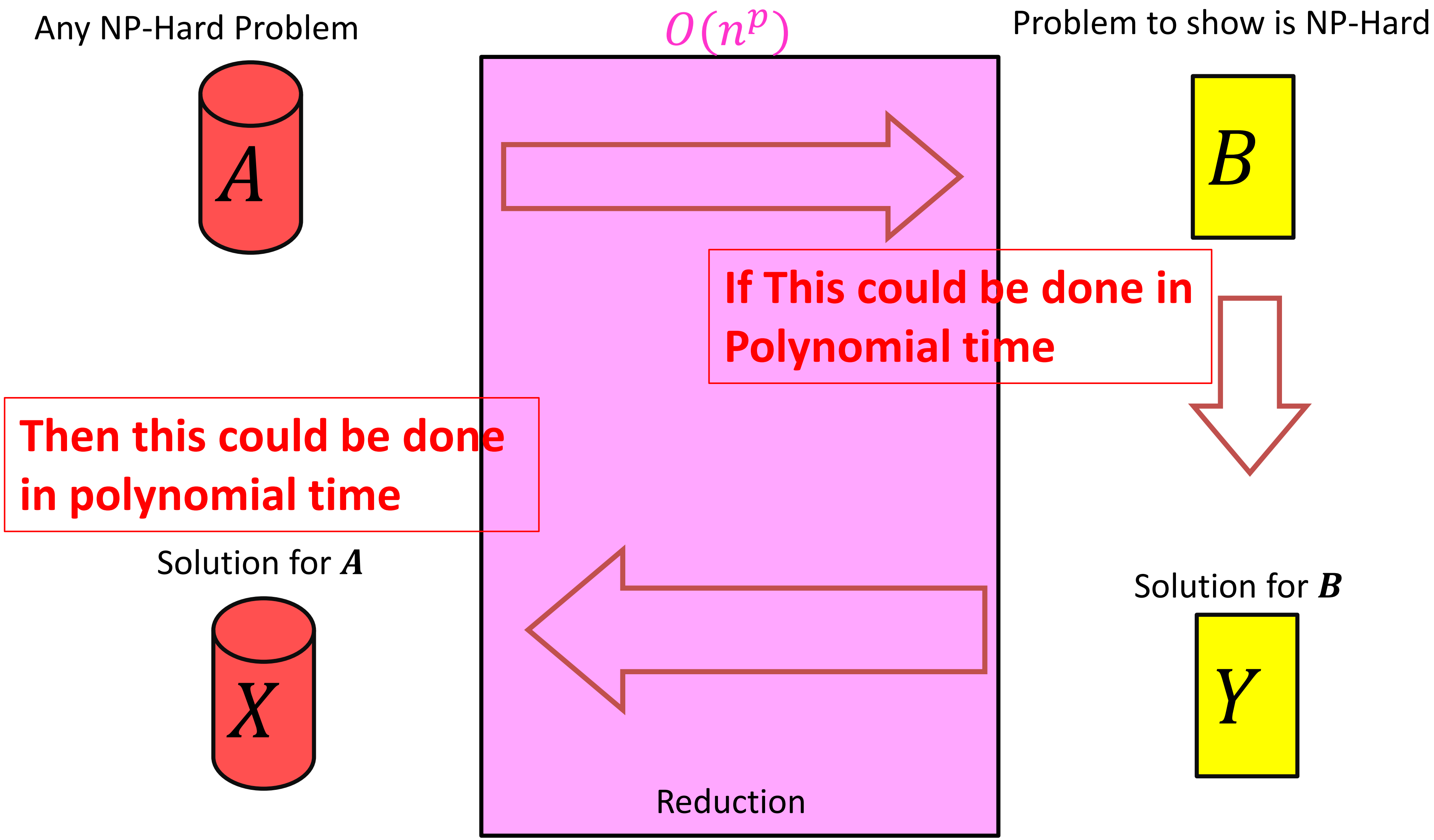
1. Check that it's of size k $O(V)$
2. Check that it's a Vertex Cover $O(E)$

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

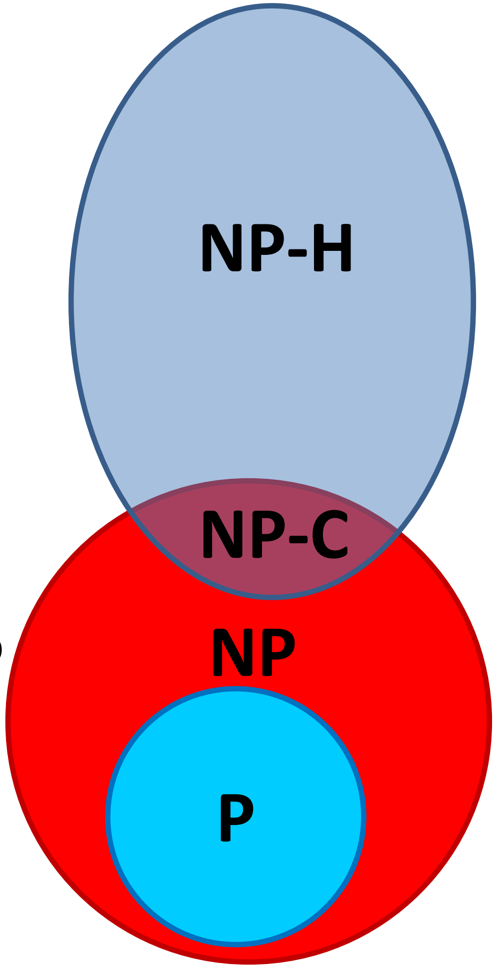


NP-Hardness Reduction



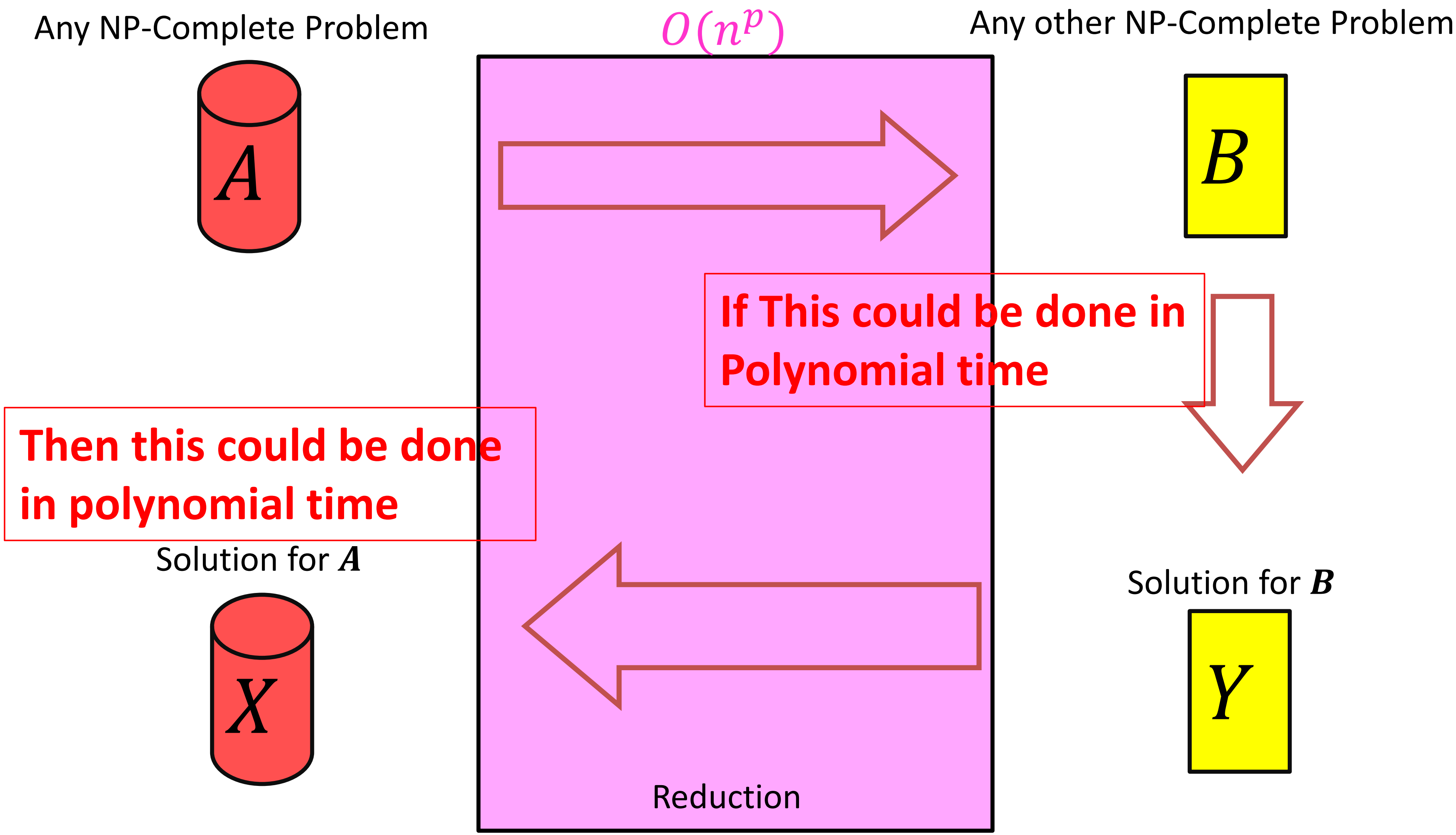
NP-Complete

- "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

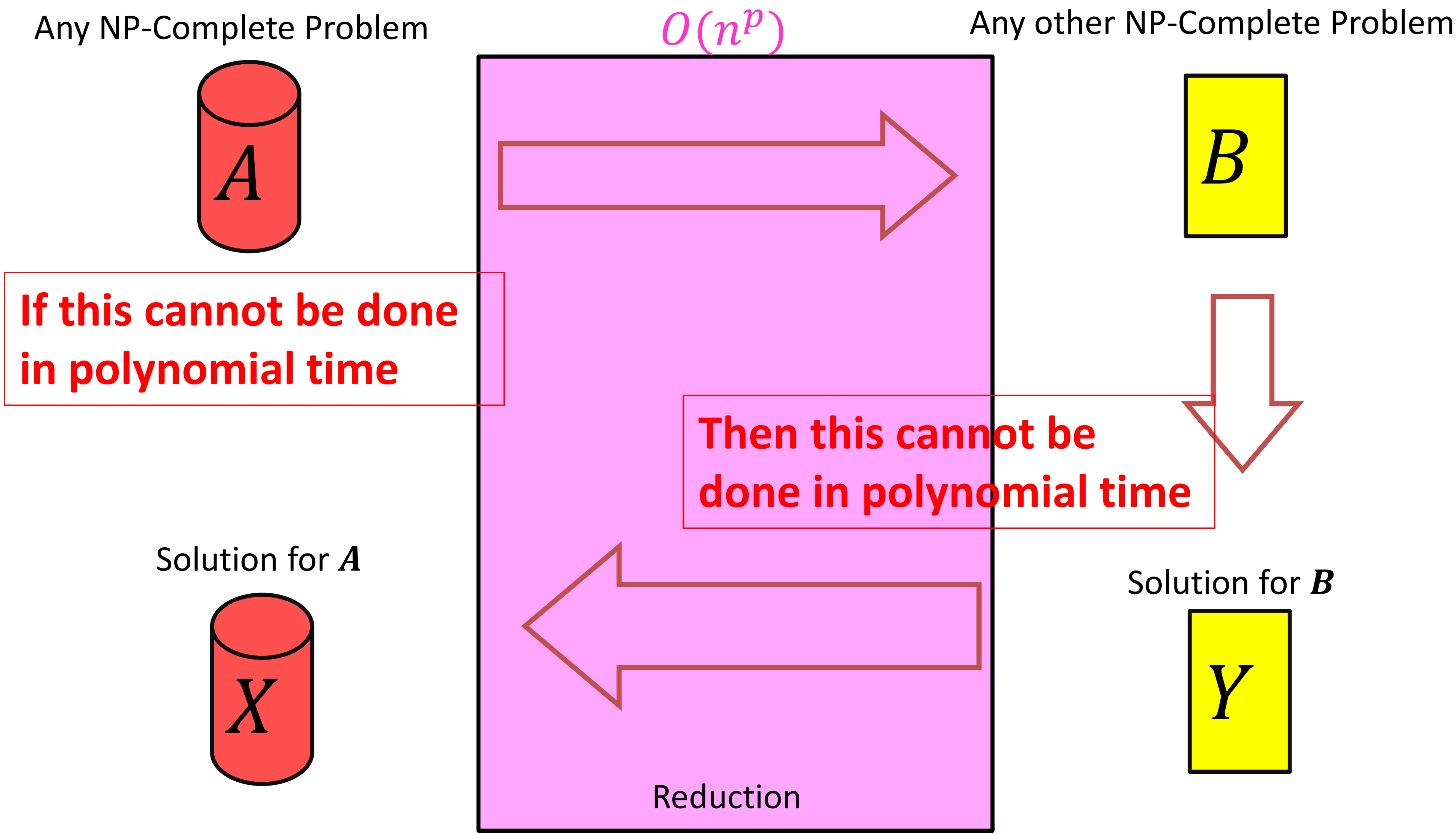


We now just need a FIRST NP-Hard problem

NP-Completeness



NP-Completeness



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?

$$\underbrace{(x \vee y \vee z)}_{\text{Clause}} \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

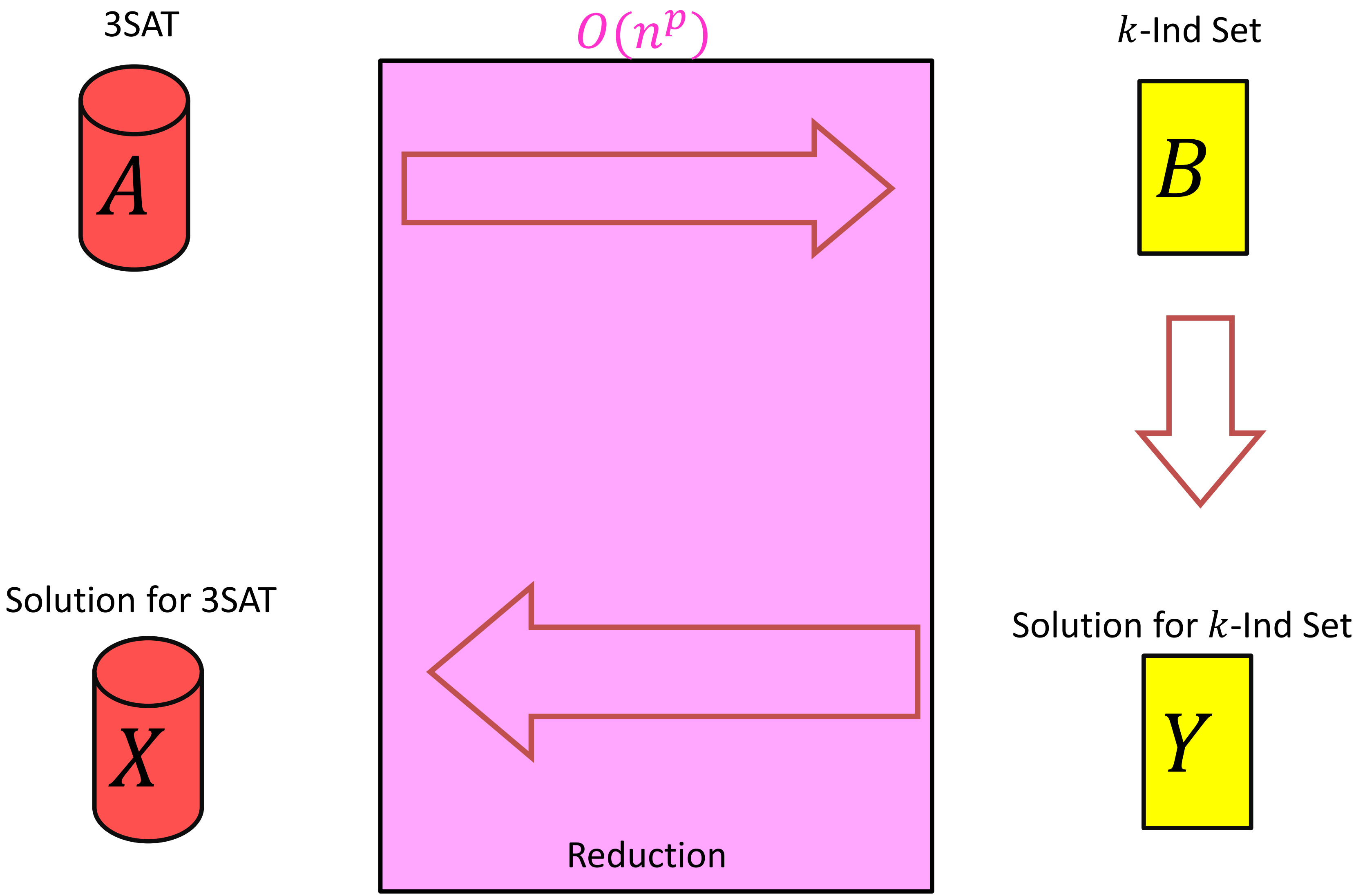
Variables

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

k-Independent Set is NP-Complete

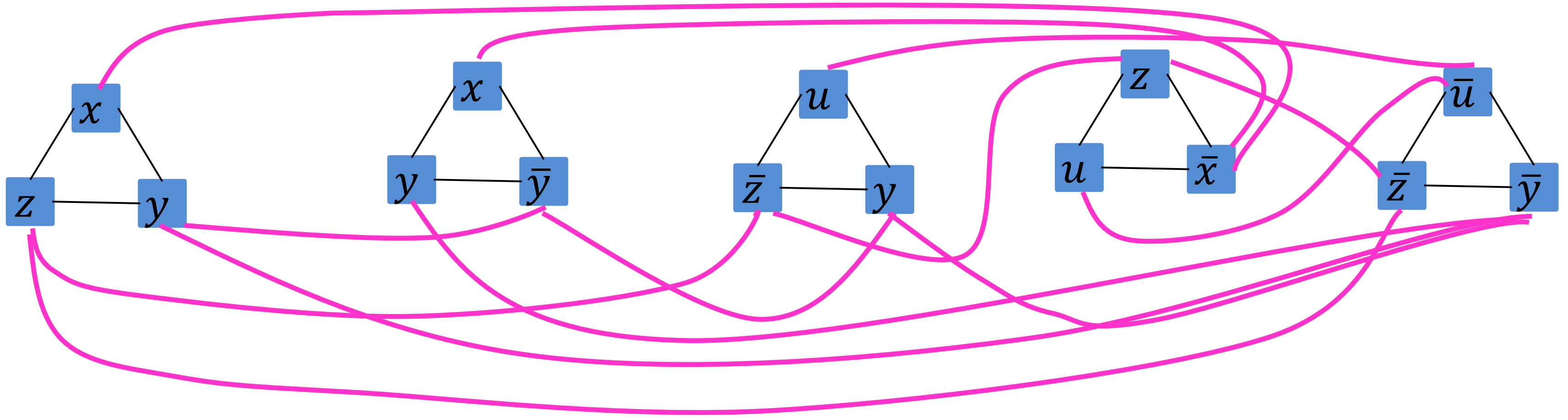
1. Show that it belongs to NP
   – Give a polynomial time verifier (slide 21)
2. Show it is NP-Hard
   – Give a reduction from a known NP-Hard problem
   – Show $3SAT \leq_p kIndSet$

$3SAT \leq_p kIndSet$



Instance of 3SAT to Instance of kIndSet

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



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

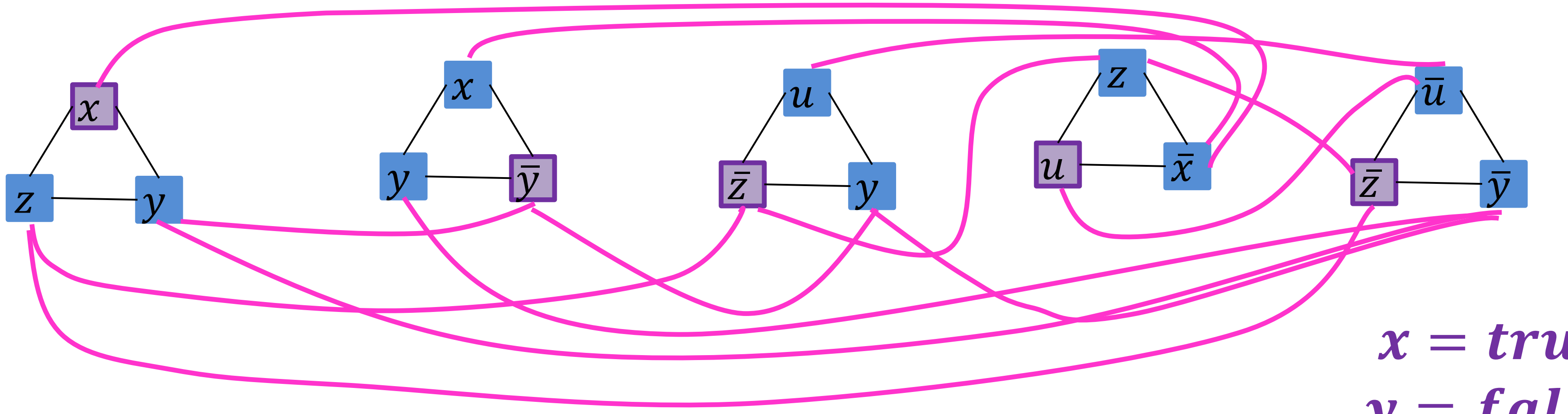
Connect each node to all of its opposites

Let k = number of clauses

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

kIndSet ⇒ Satisfying Assignment

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



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

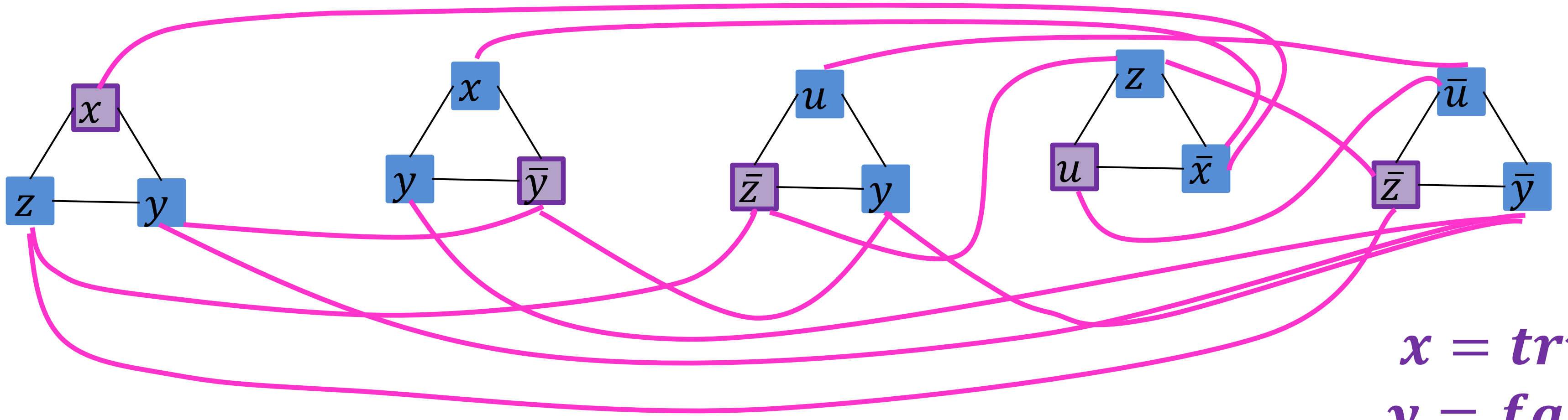
One node per triangle is in the Independent set: because we can have exactly k total in the set, and 2 in a triangle would be adjacent

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 ⇒ kIndSet

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



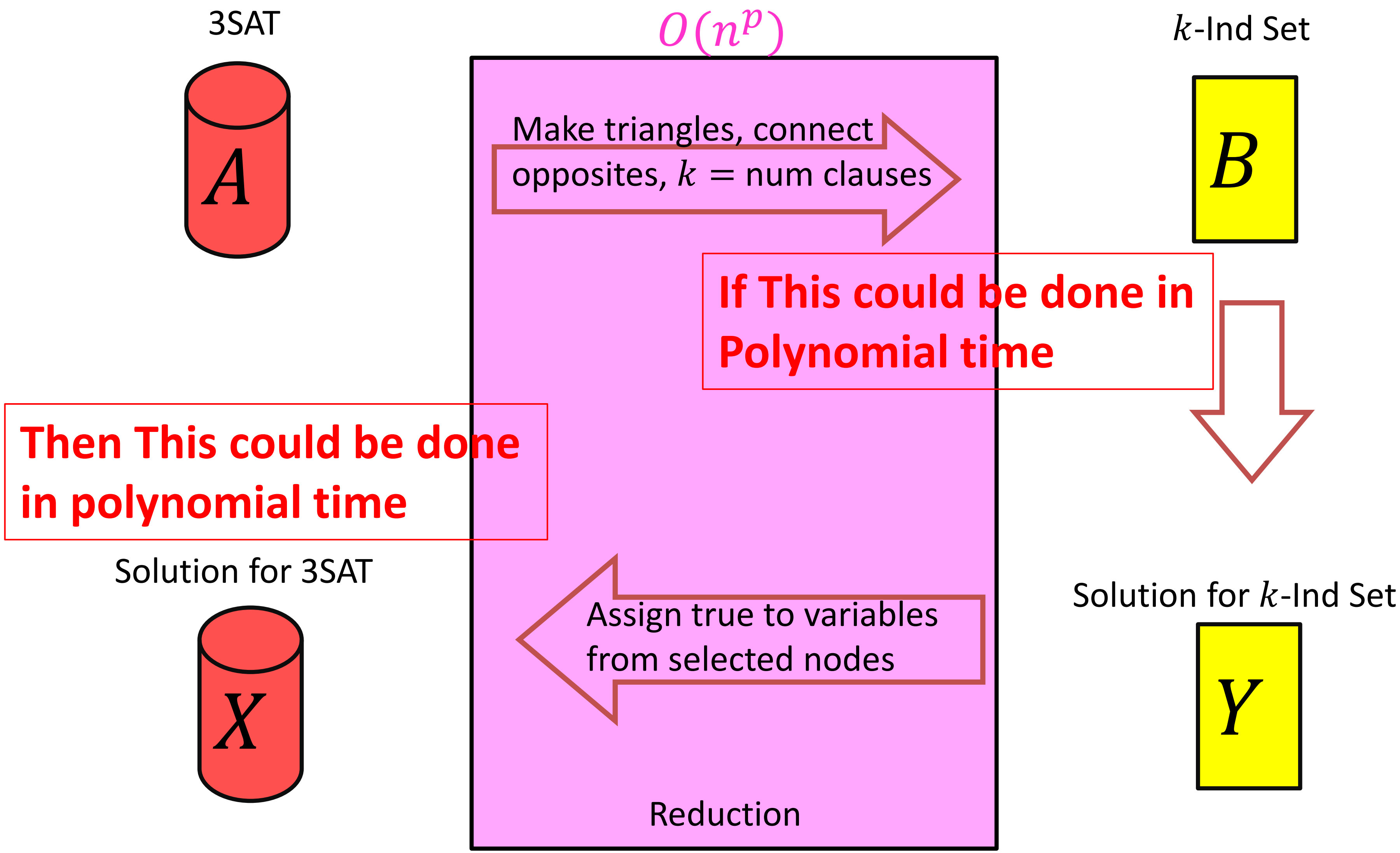
$\boldsymbol{x = true}$
$\boldsymbol{y = false}$
$\boldsymbol{z = false}$
$\boldsymbol{u = true}$

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

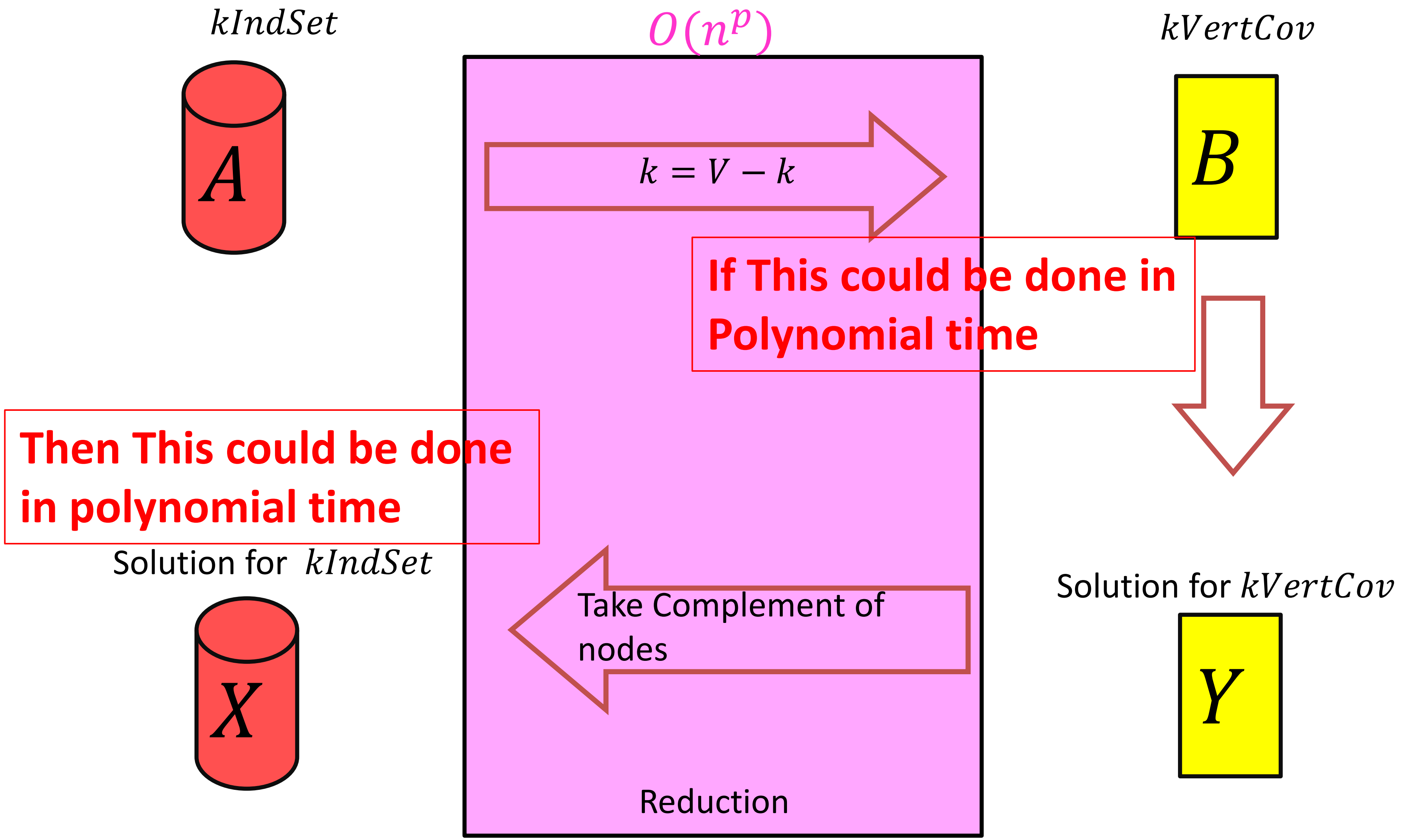
$3SAT \leq_p kIndSet$



k-Vertex Cover is NP-Complete

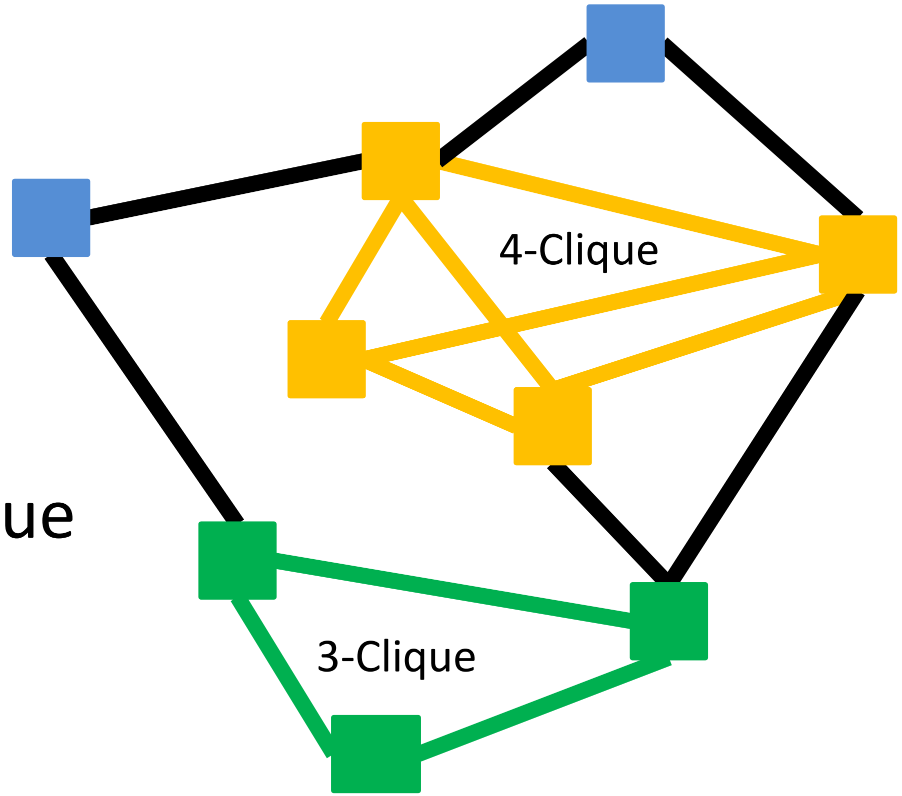
1. Show that it belongs to NP
   - Give a polynomial time verifier (slide 22)
2. Show it is NP-Hard
   - Give a reduction from a known NP-Hard problem
   - We showed $kIndSet \leq_p kVertCov$
     - (Last Class)

$kIndSet \leq_p kVertCov$



k-Clique Problem

- Clique: A complete subgraph
- k-Clique Problem:
  - Given a graph G and a number k, is there a clique of size k?

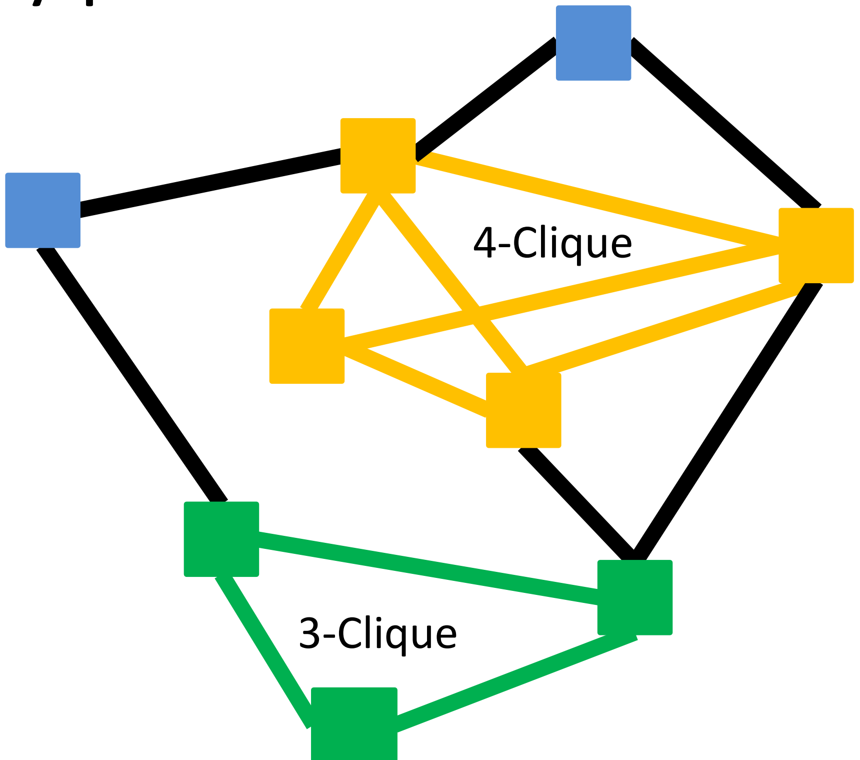


k-Clique is NP-Complete

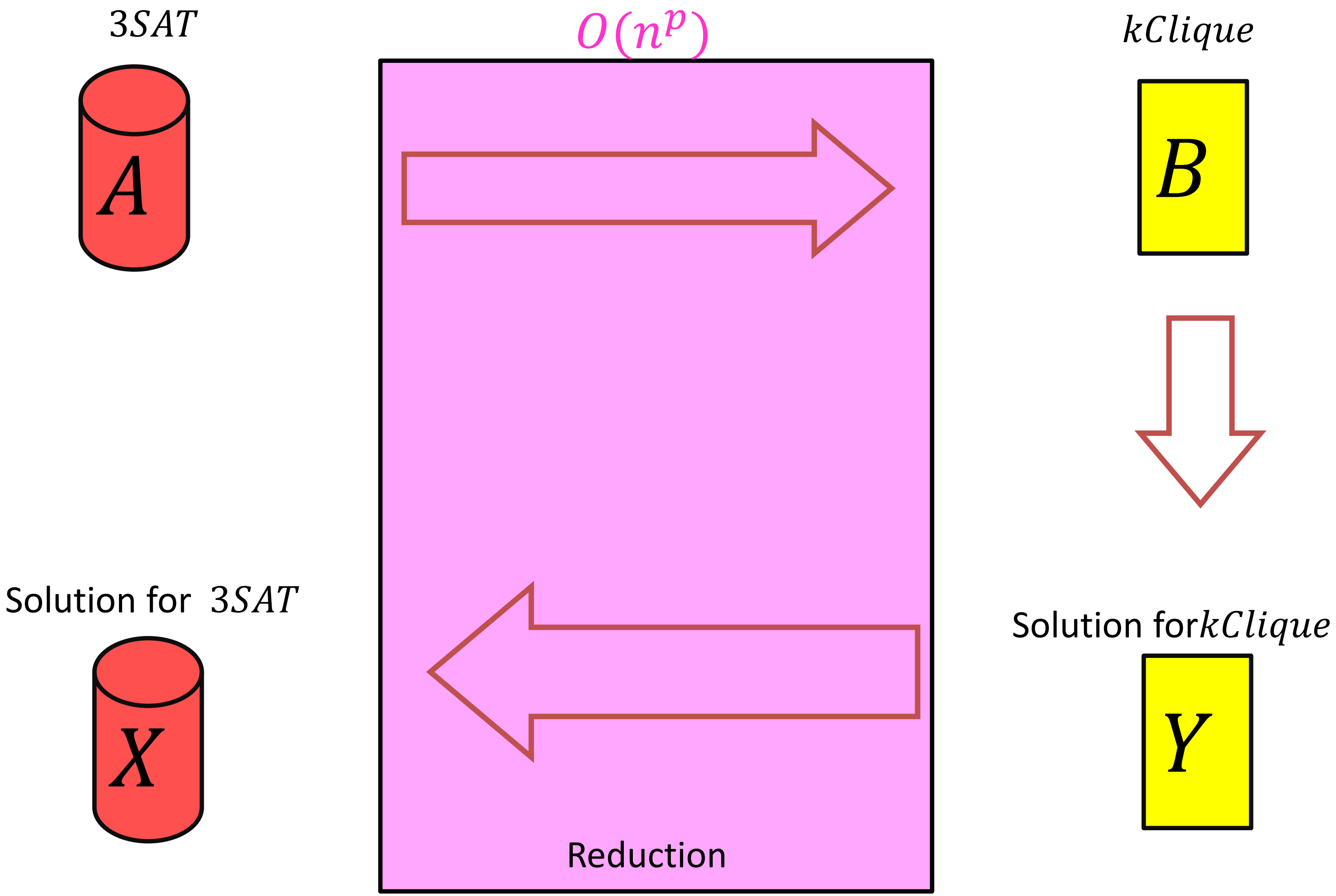
1. Show that it belongs to NP
   – Give a polynomial time verifier
2. Show it is NP-Hard
   – Give a reduction from a known NP-Hard problem
   – We will show $3SAT \leq_p kClique$

k-Clique is NP

1. Given a Graph and a potential solution
2. Check that the solution has k nodes
3. Check that every pair of nodes share an edge

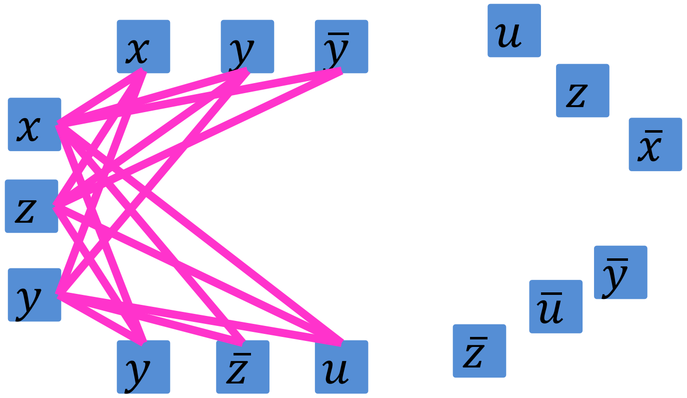


$3SAT \leq_p kClique$



Instance of 3SAT to Instance of kClique

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



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

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

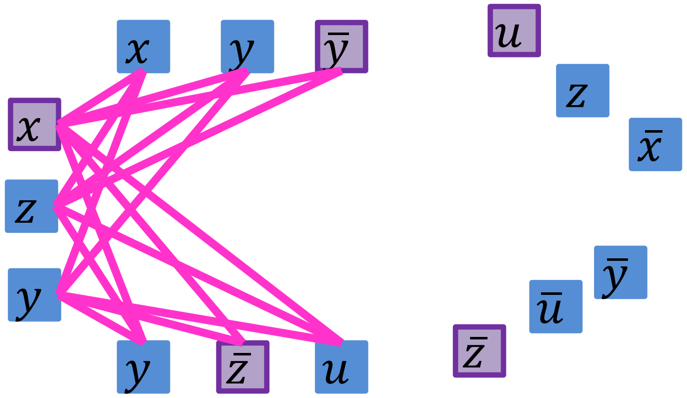
Connect each node to all non-contradictory nodes in the other clauses (i.e., anything that's not its negation)

Let k = number of clauses

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

kClique $\Rightarrow$ Satisfying Assignment

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



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

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

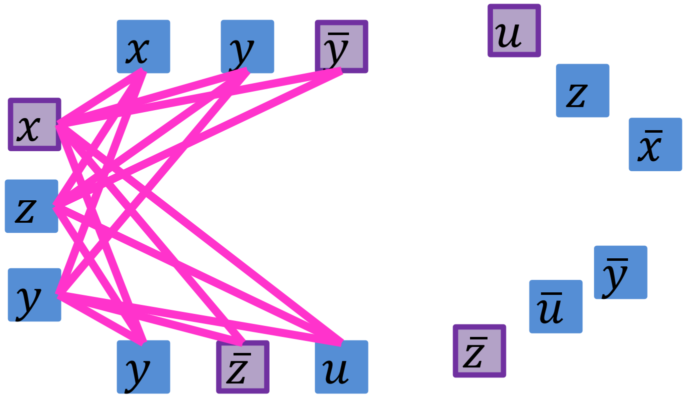
To have a k-Clique, must have one node from each triplet

Cannot select a node for both a variable and its negation

Therefore selection of nodes is a satisfying assignment

Satisfying Assignment $\Rightarrow k$Clique

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



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

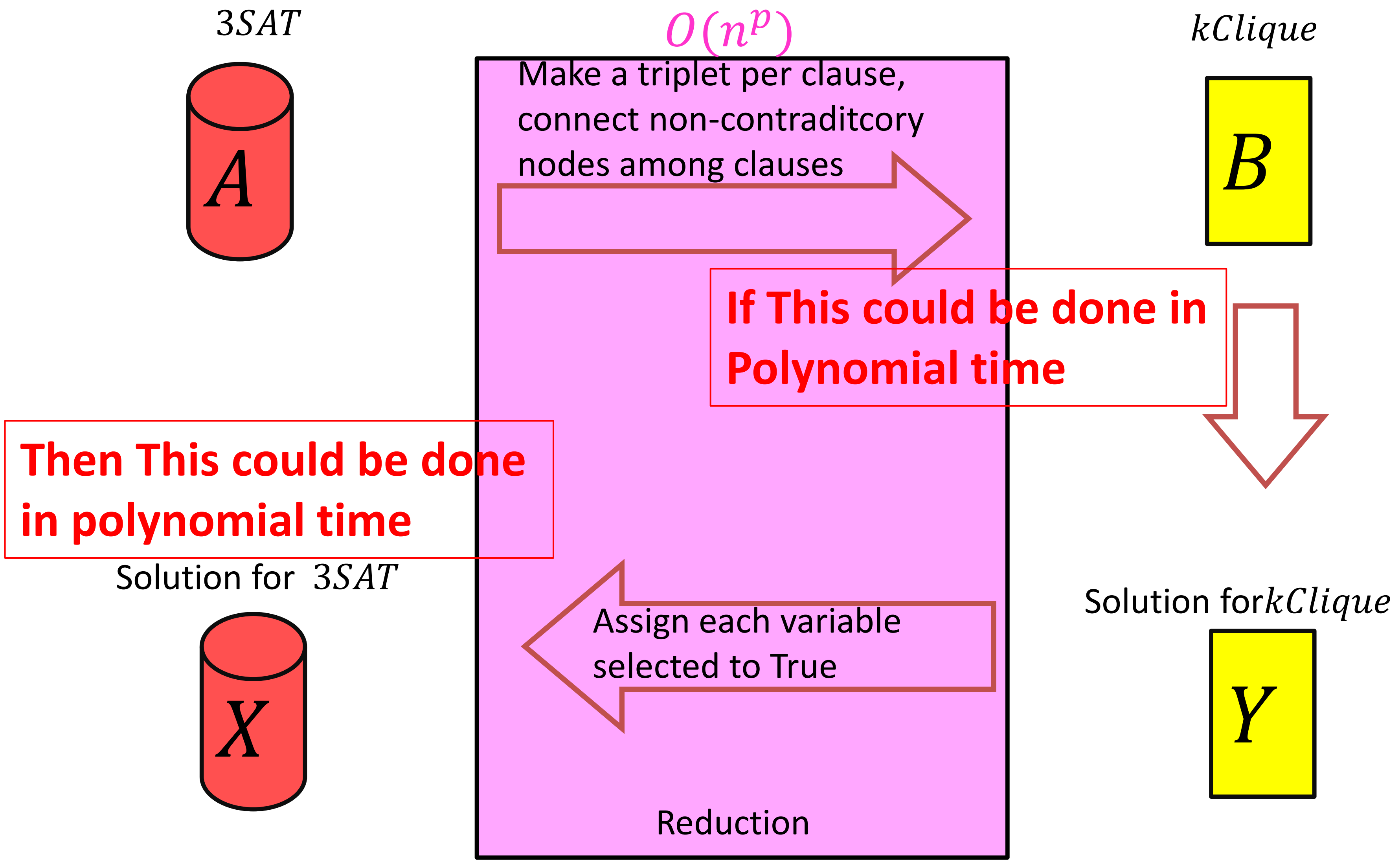
Select one node for a true variable from each clause

There will be k nodes selected

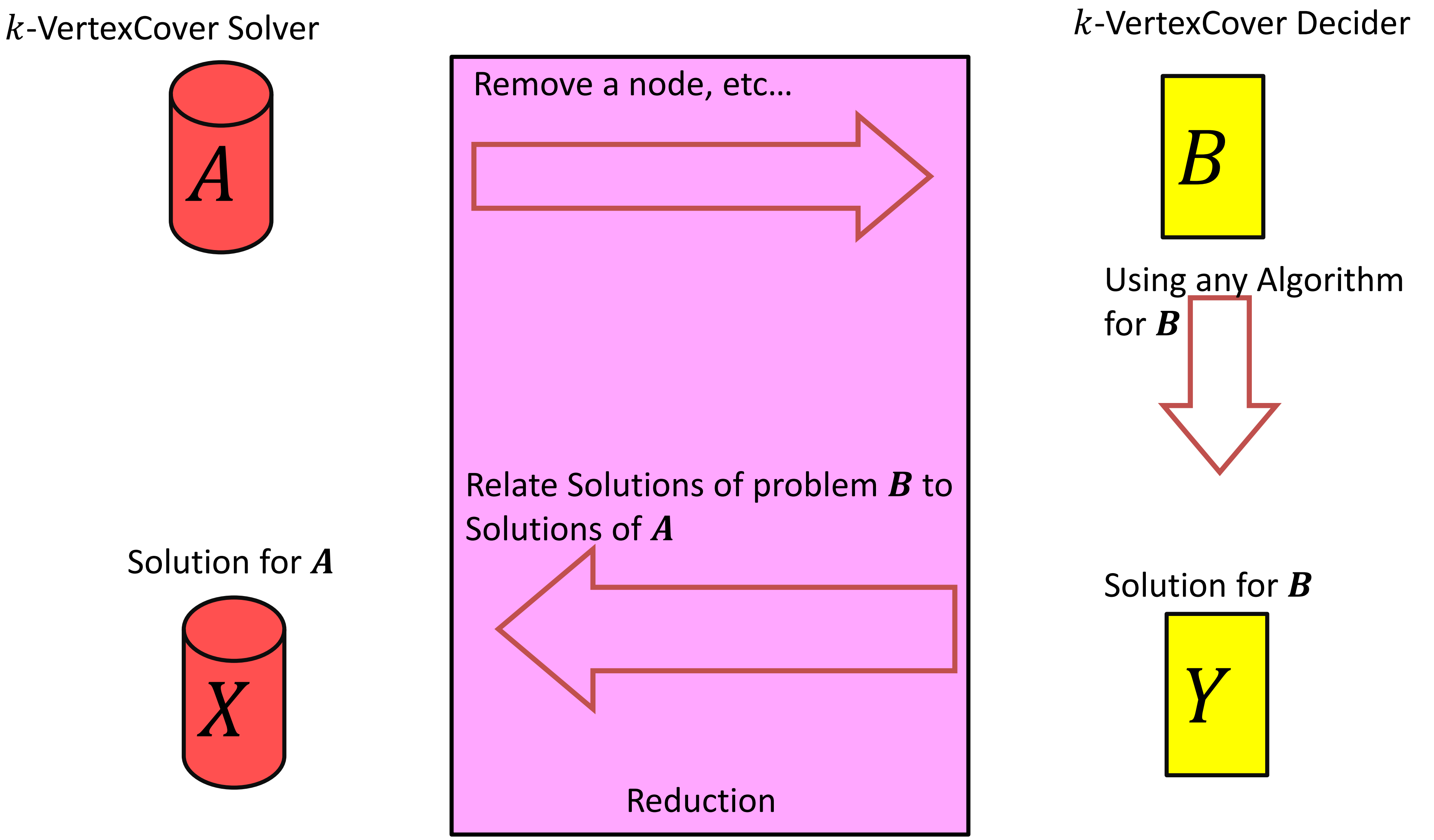
We can't select both a node and its negation

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

$3SAT \leq_p kClique$



Reduction



k-VertexCover Solver
A
Solution for $\boldsymbol{A}$
X
Remove a node, etc…
Relate Solutions of problem $\boldsymbol{B}$ to Solutions of $\boldsymbol{A}$
Reduction
k-VertexCover Decider
B
Using any Algorithm for $\boldsymbol{B}$
Solution for $\boldsymbol{B}$
Y

Problem Types

- Decision Problems:
  - Is there a solution?
    - Output is True/False
  - Is there a vertex cover of size k?
- Search Problems:
  - Find a solution
    - Output is complex
  - Give a vertex cover of size k
- Verification Problems:
  - Given a potential solution, is it valid?
    - Output is True/False
  - Is **this** a vertex cover of size k**?**

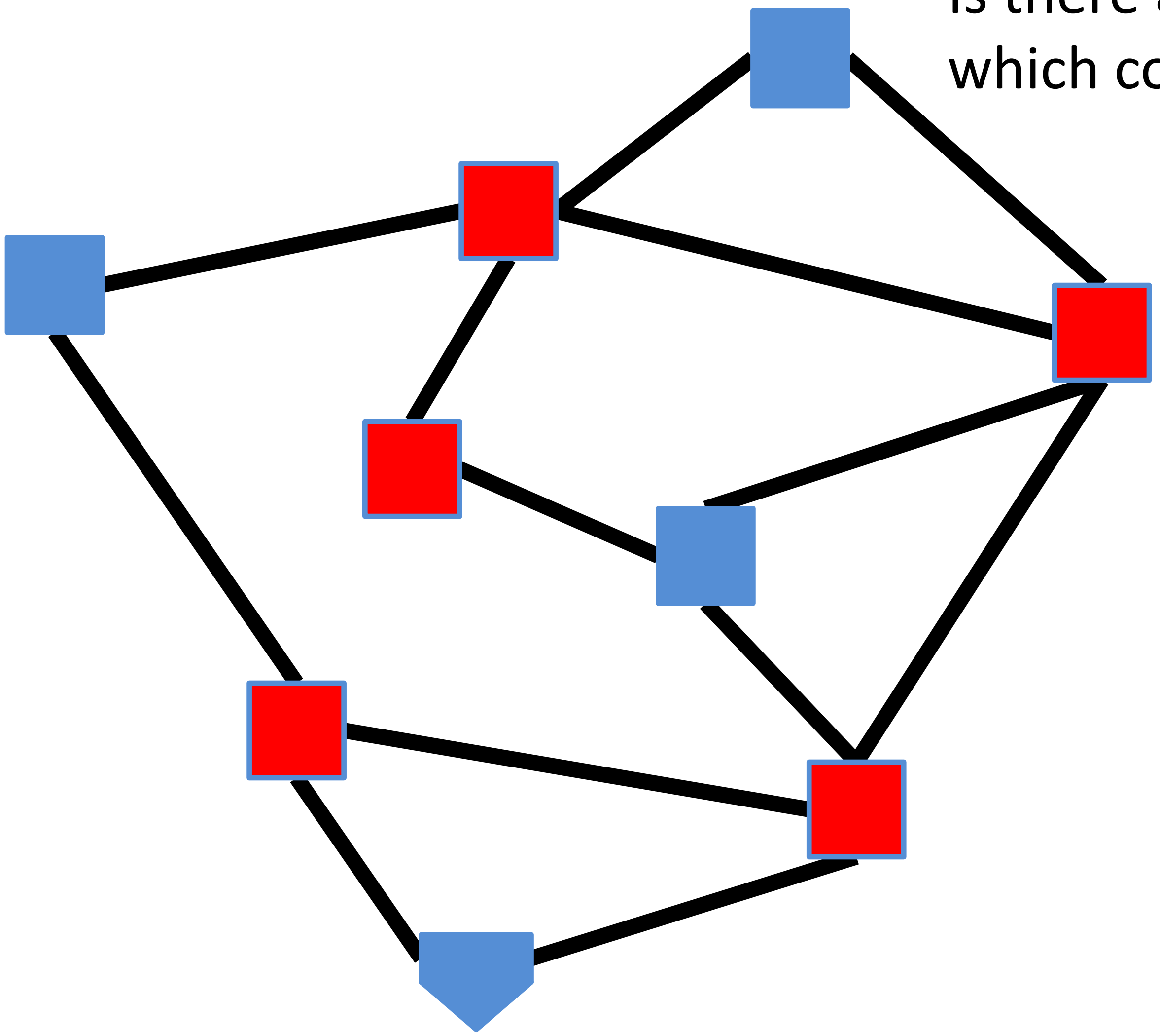
If we can solve this

Then we can solve this

Using a k-VertexCover decider to build a searcher

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

5 Vertex Cover (Decision)



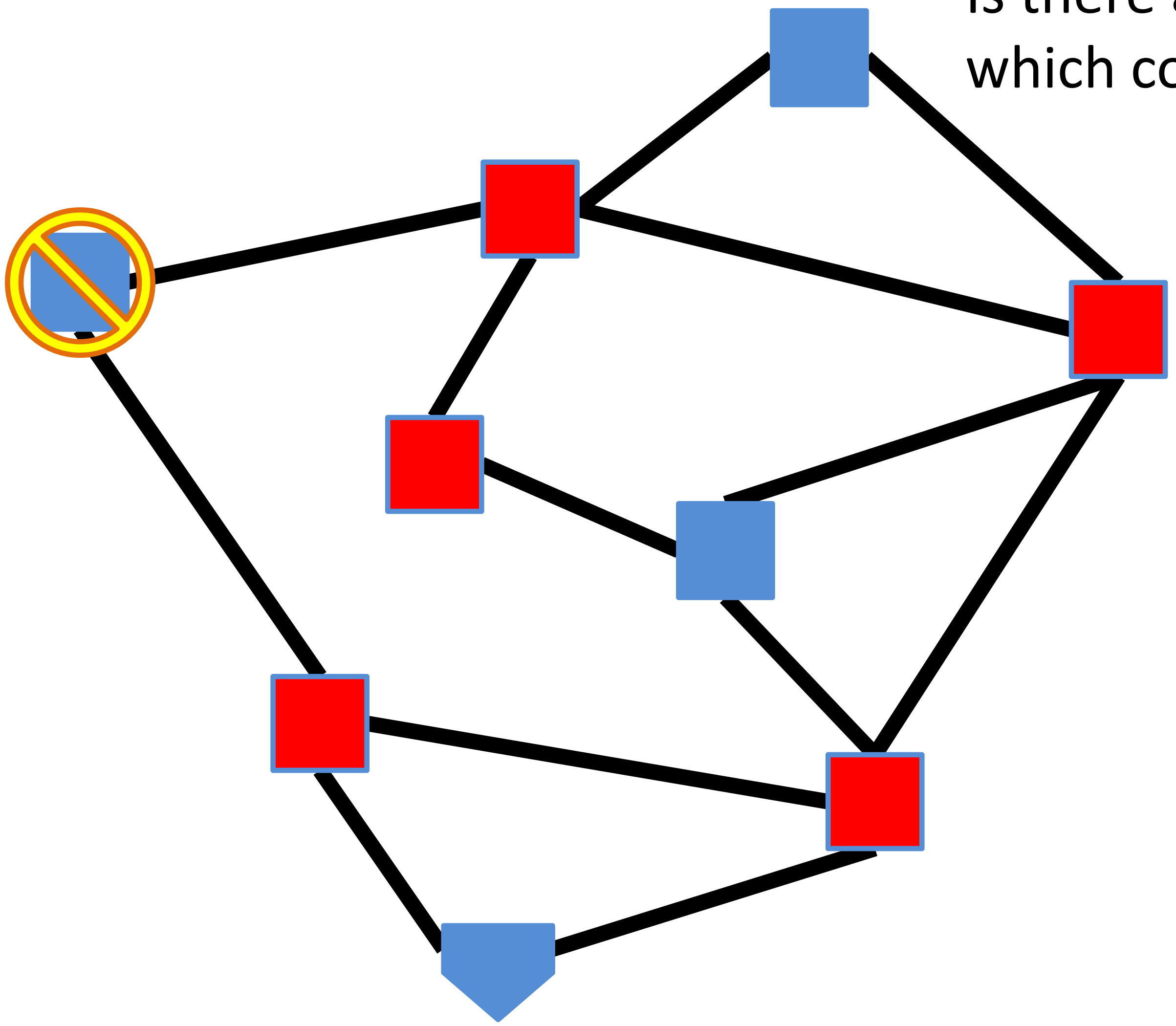
Is there a set of nodes of size 5 which covers every edge?

Yes!

4 Vertex Cover (Decision)

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

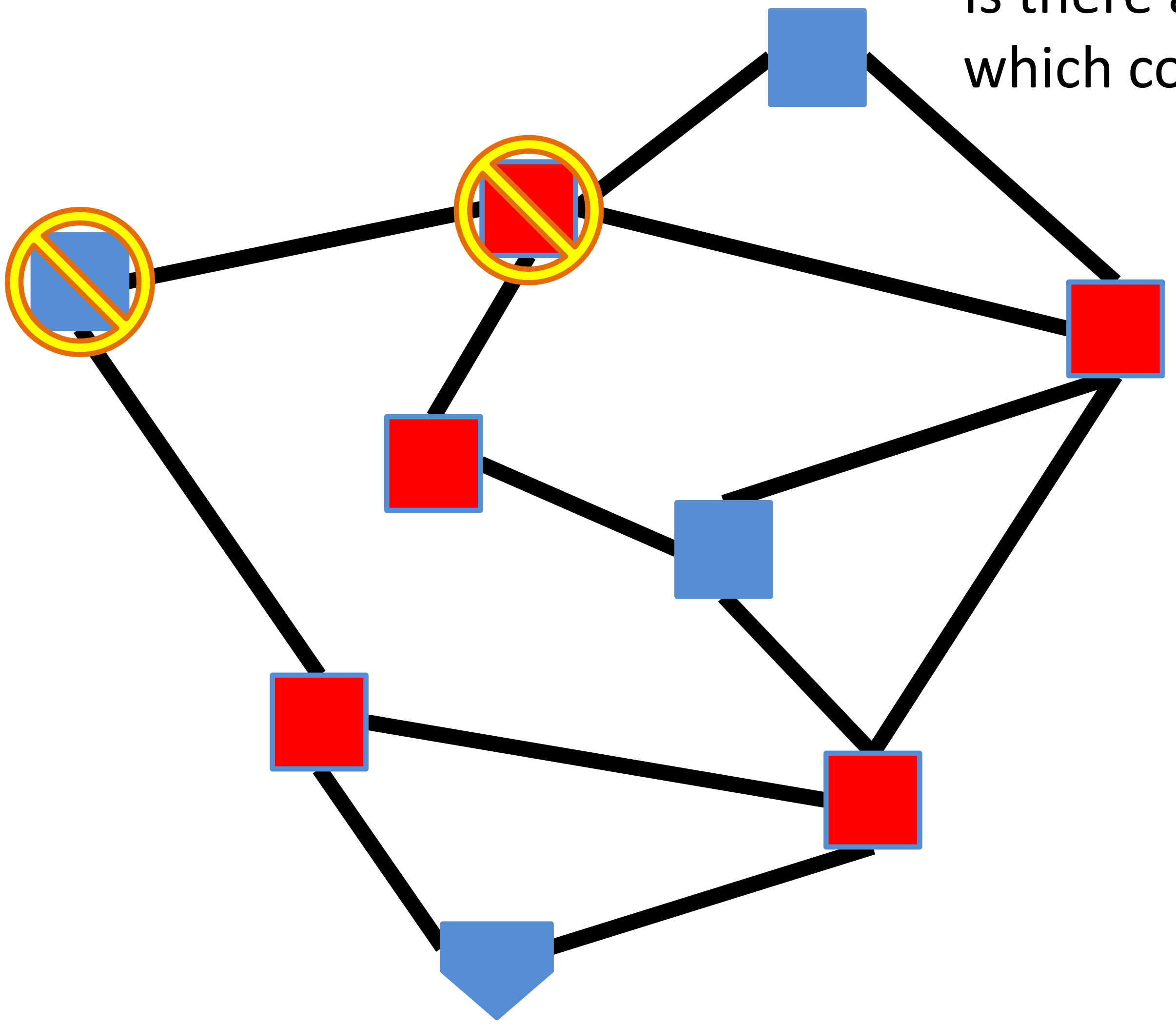
No!



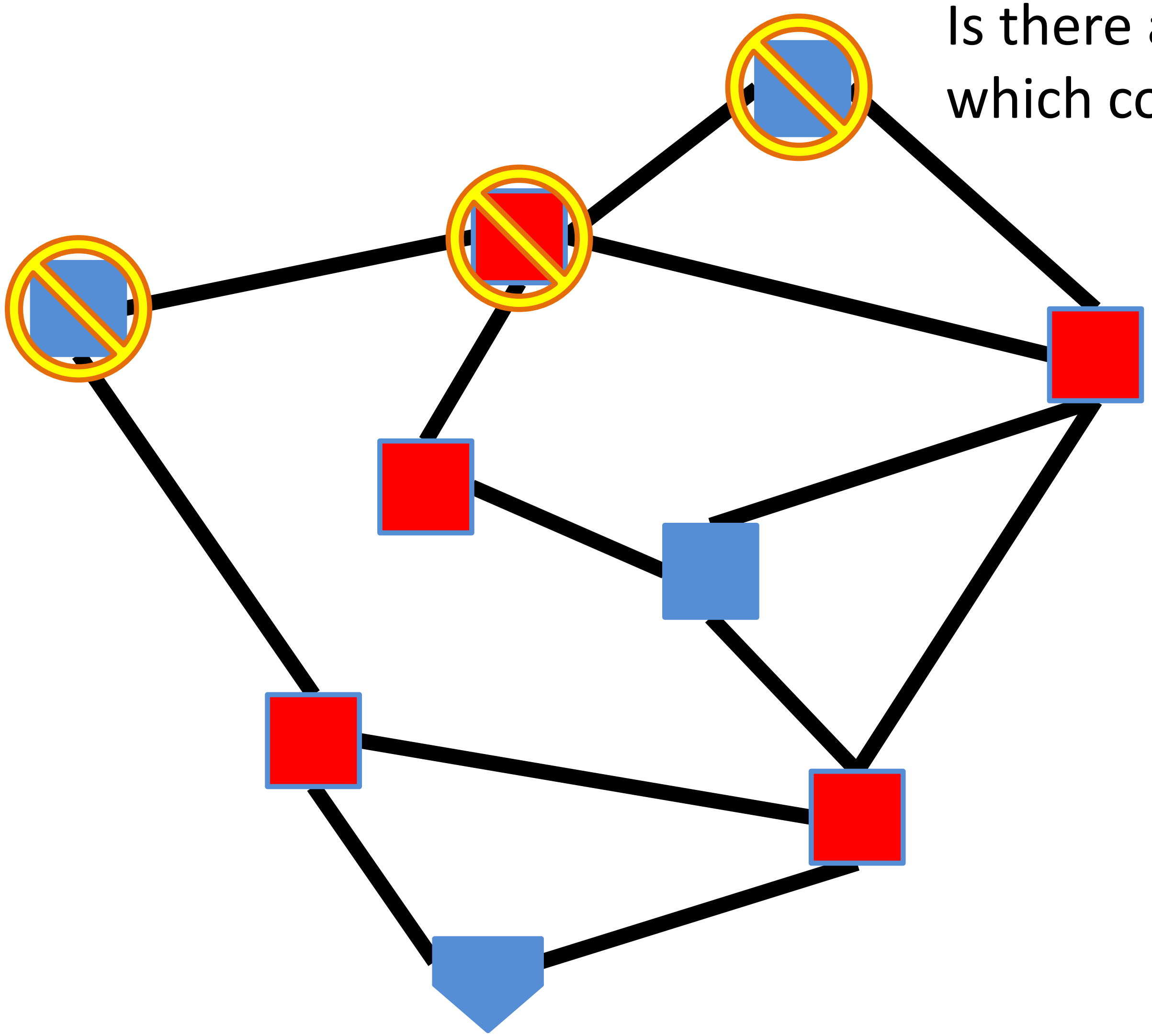
4 Vertex Cover (Decision)

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!