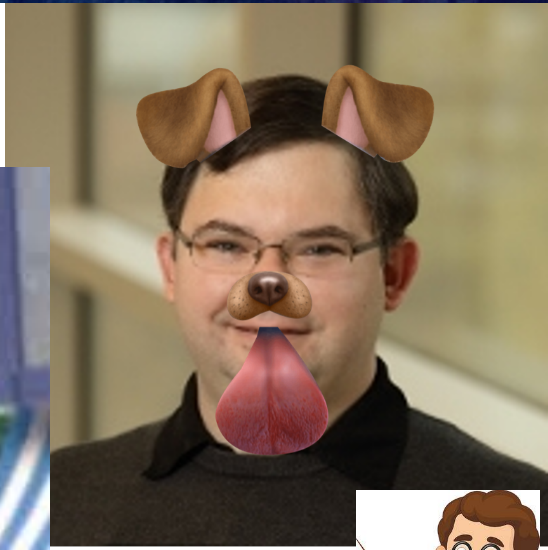
CS4102 Algorithms

Fall 2019



CS4102 Algorithms

Fall 2019

Today's Keywords

- Reductions
- Bipartite Matching
- Vertex Cover
- Independent Set

CLRS Readings

- Chapter 34

Homeworks

- HW8 due Saturday, 11/23, at 11pm
  - Python or Java
  - Marriage
- HW9 out tonight, due Thursday 12/5 at 11pm
  - Reductions, Graphs
  - Written (LaTeX)
- HW10C out tonight, due Thursday 12/5 at 11pm
  - Implement a problem from HW9
  - No late submissions

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

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 $\boldsymbol{B}$

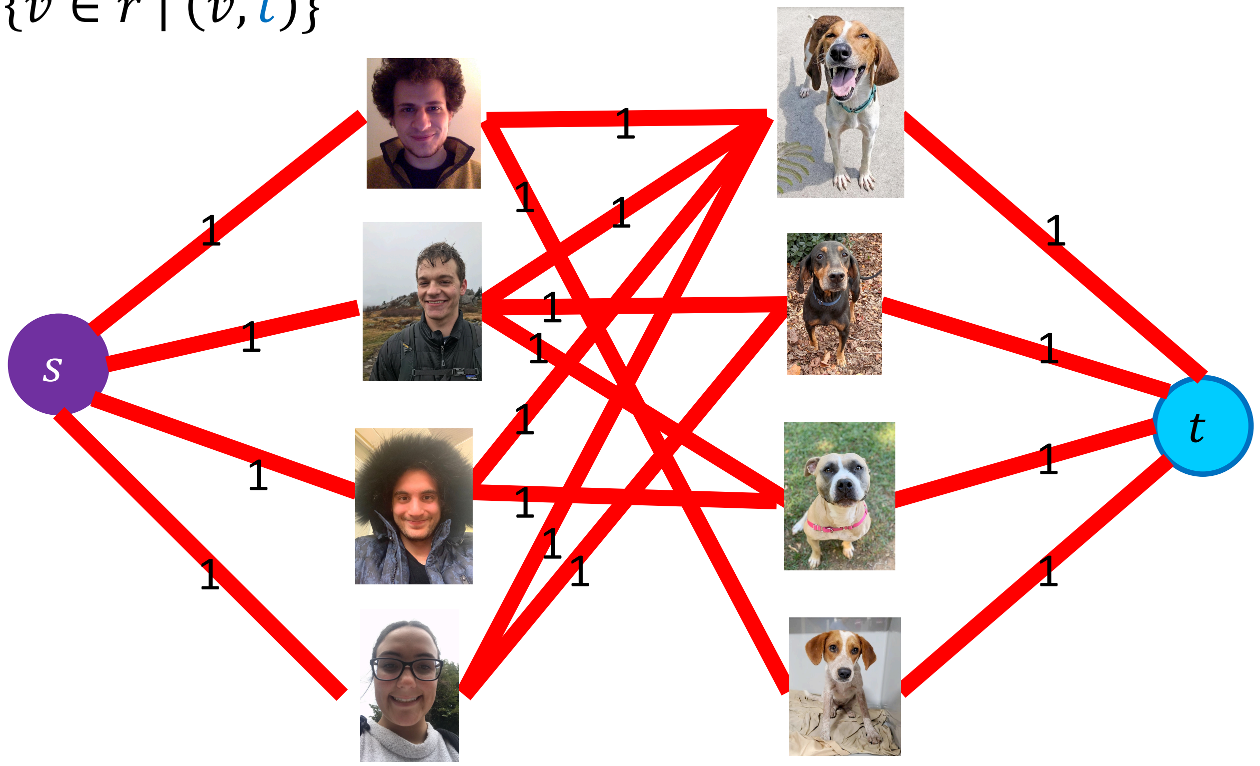
Alcohol, wood, matches

Maximum Bipartite Matching

Maximum Bipartite Matching Using Max Flow

Make $G = (L, R, E)$ a flow network $G' = (V', E')$ by:

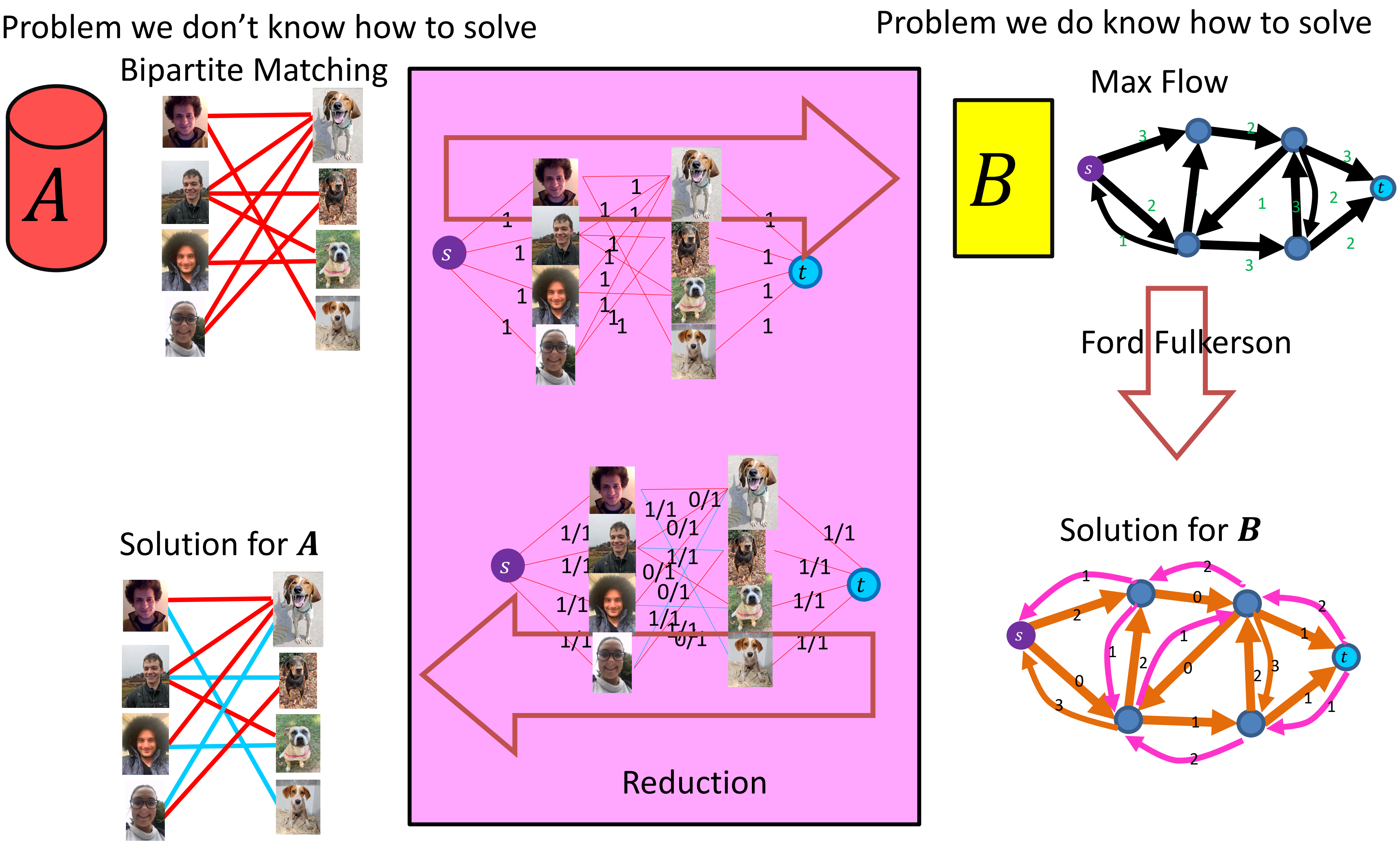
- Adding in a source and sink to the set of nodes:
  - $V' = L \cup R \cup \{s, t\}$
- Adding an edge from source to L and from R to sink:
  - $E' = E \cup \{u \in L \mid (s, u)\} \cup \{v \in r \mid (v, t)\}$
- Make each edge capacity 1:
  - $\forall e \in E', c(e) = 1$



Remember: need to show

1. How to map instance of MBM to MF (and back) - construction
2. A valid solution to MF instance is a valid solution to MBM instance

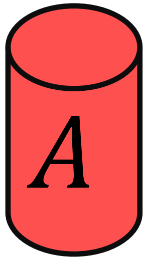
Bipartite Matching Reduction

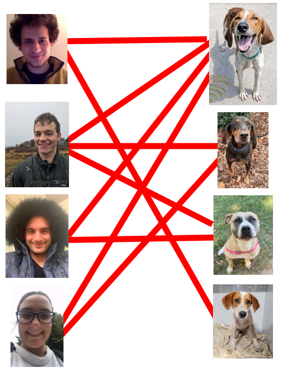


Bipartite Matching Reduction

Problem we don't know how to solve

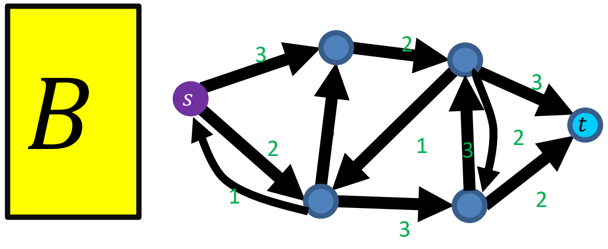
Bipartite Matching

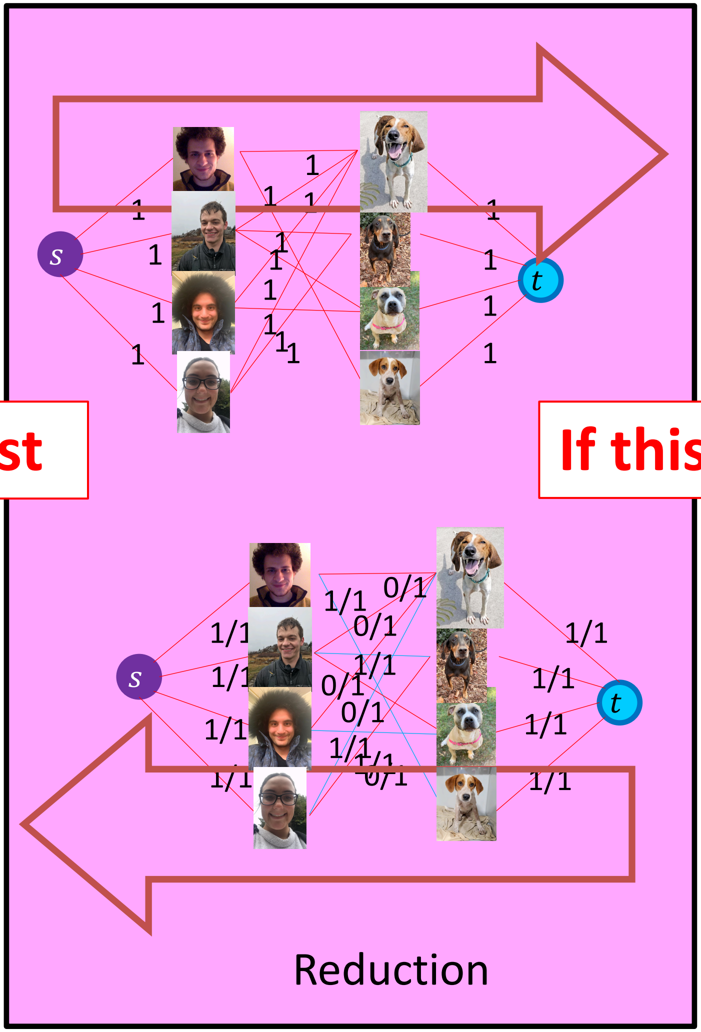




Problem we do know how to solve

Max Flow

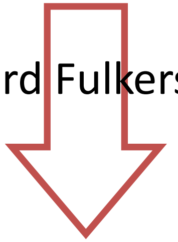




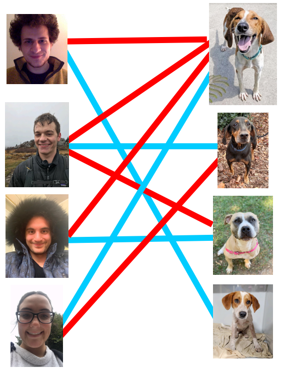
Then this is fast

If this is fast

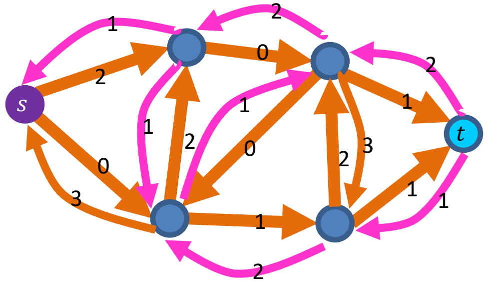
Ford Fulkerson



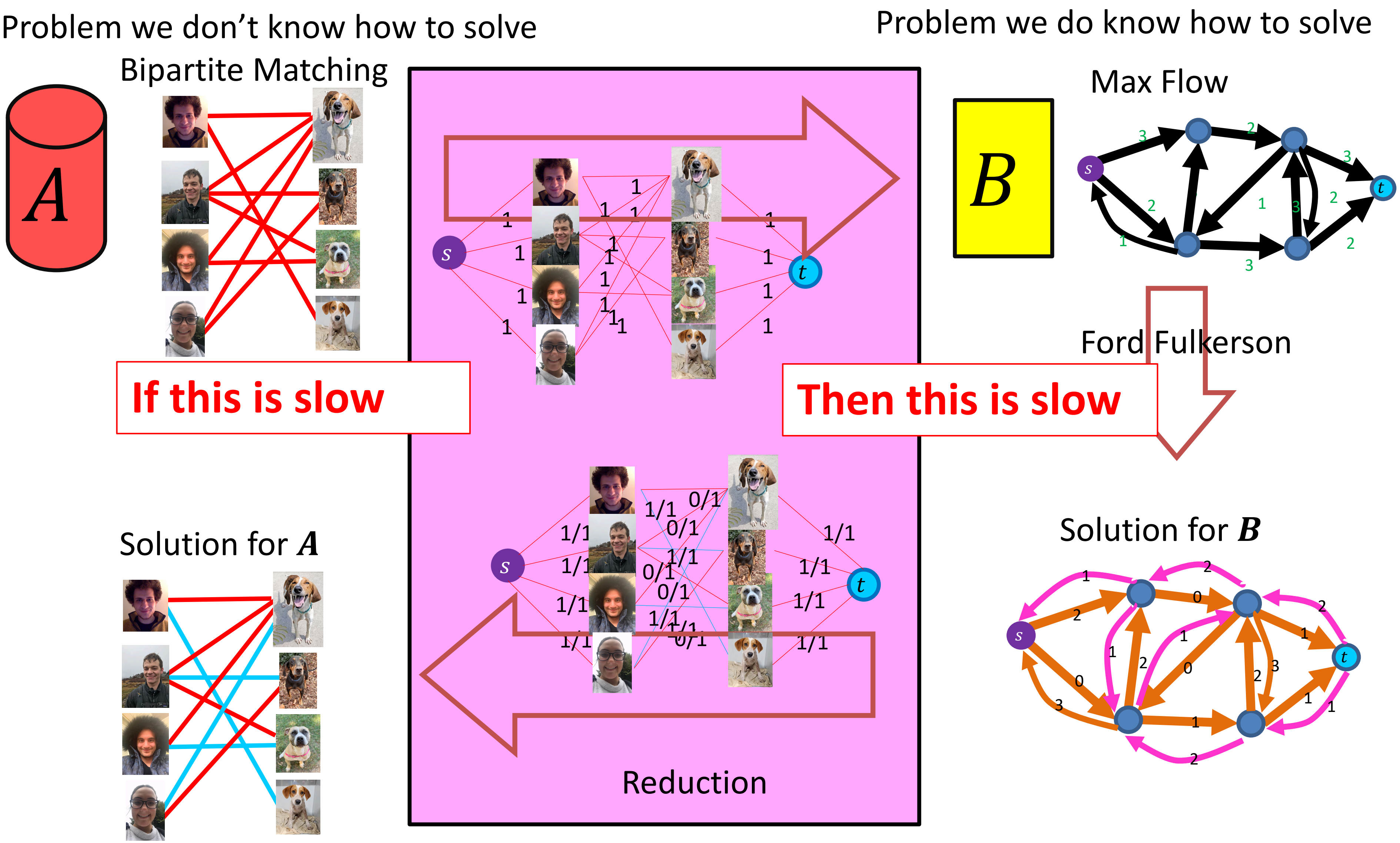
Solution for ***A***



Solution for ***B***

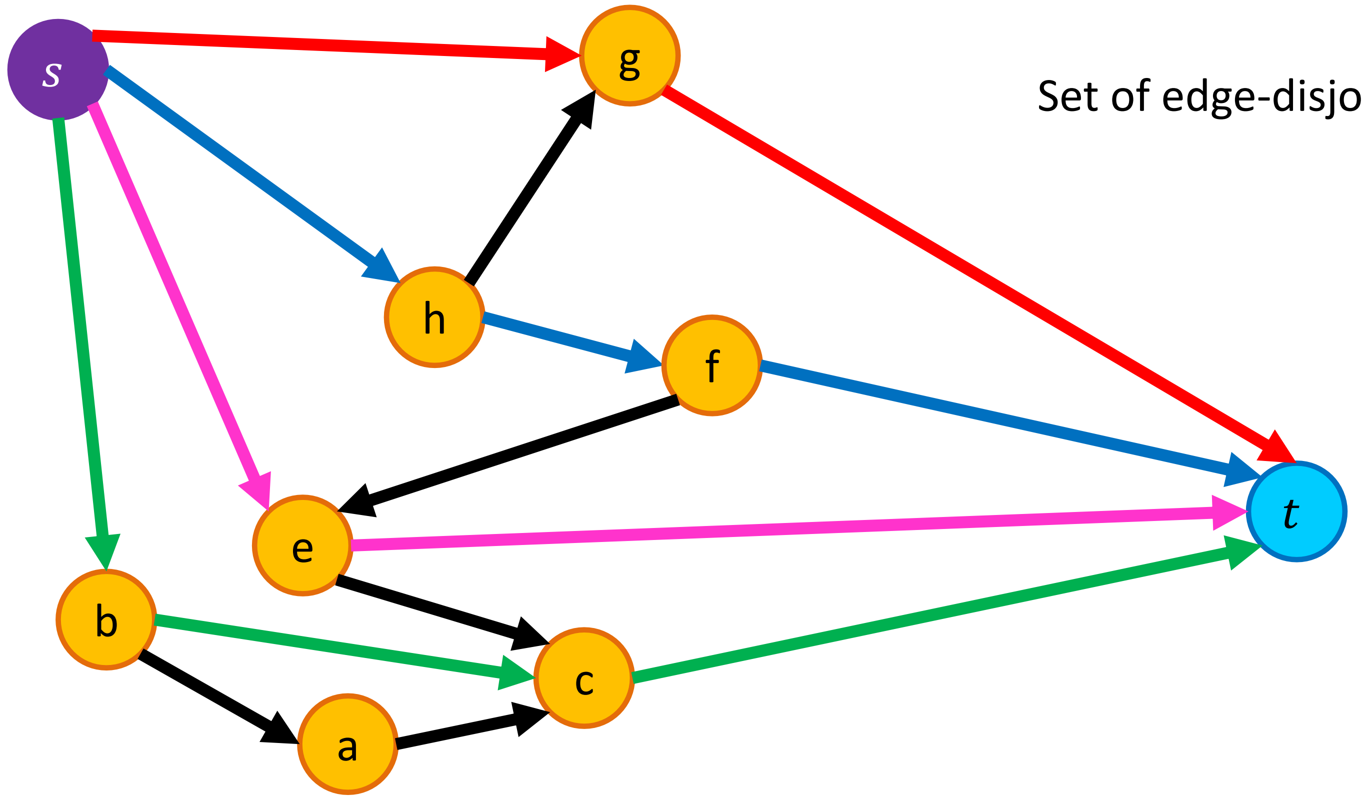


Bipartite Matching Reduction



Edge-Disjoint Paths

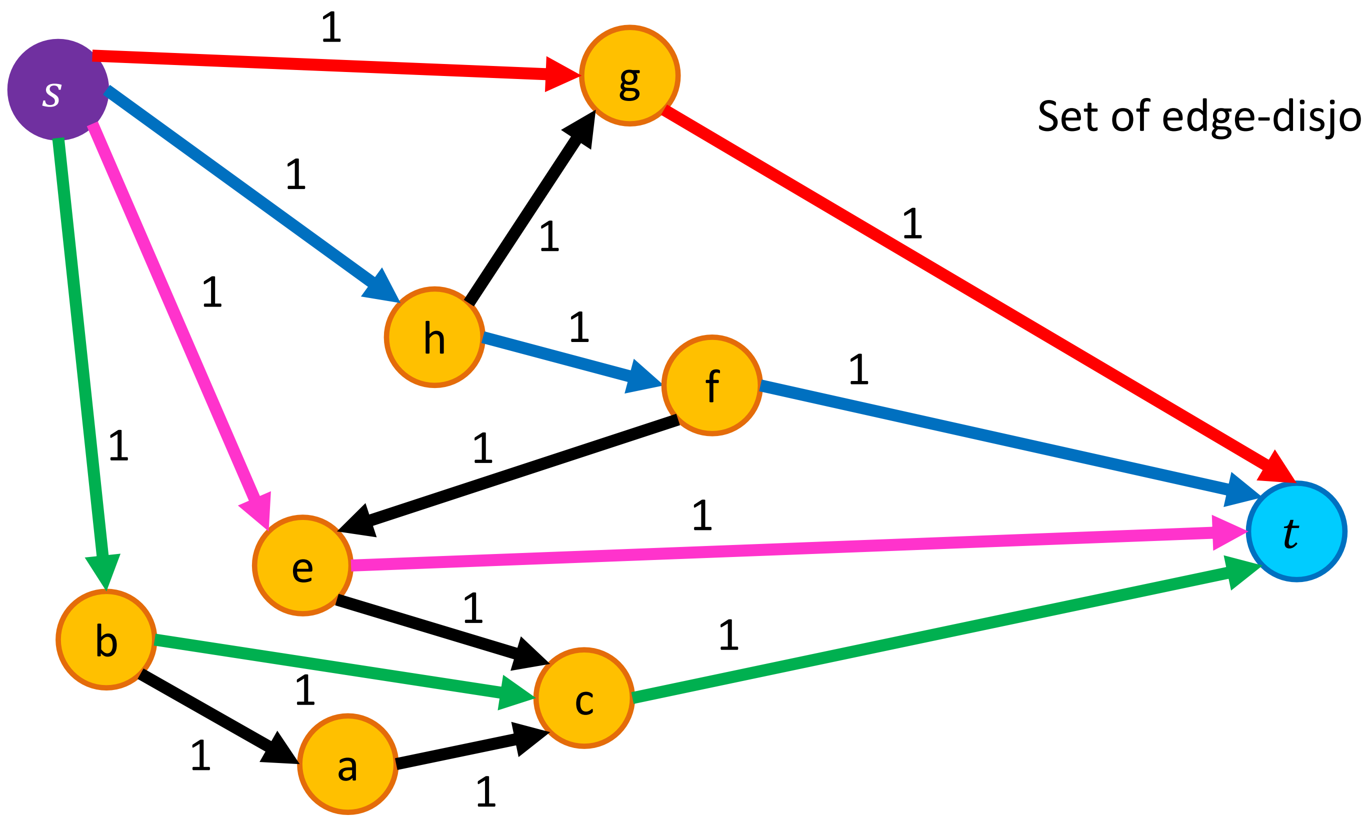
Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no edges



Set of edge-disjoint paths of size 4

Edge-Disjoint Paths Algorithm

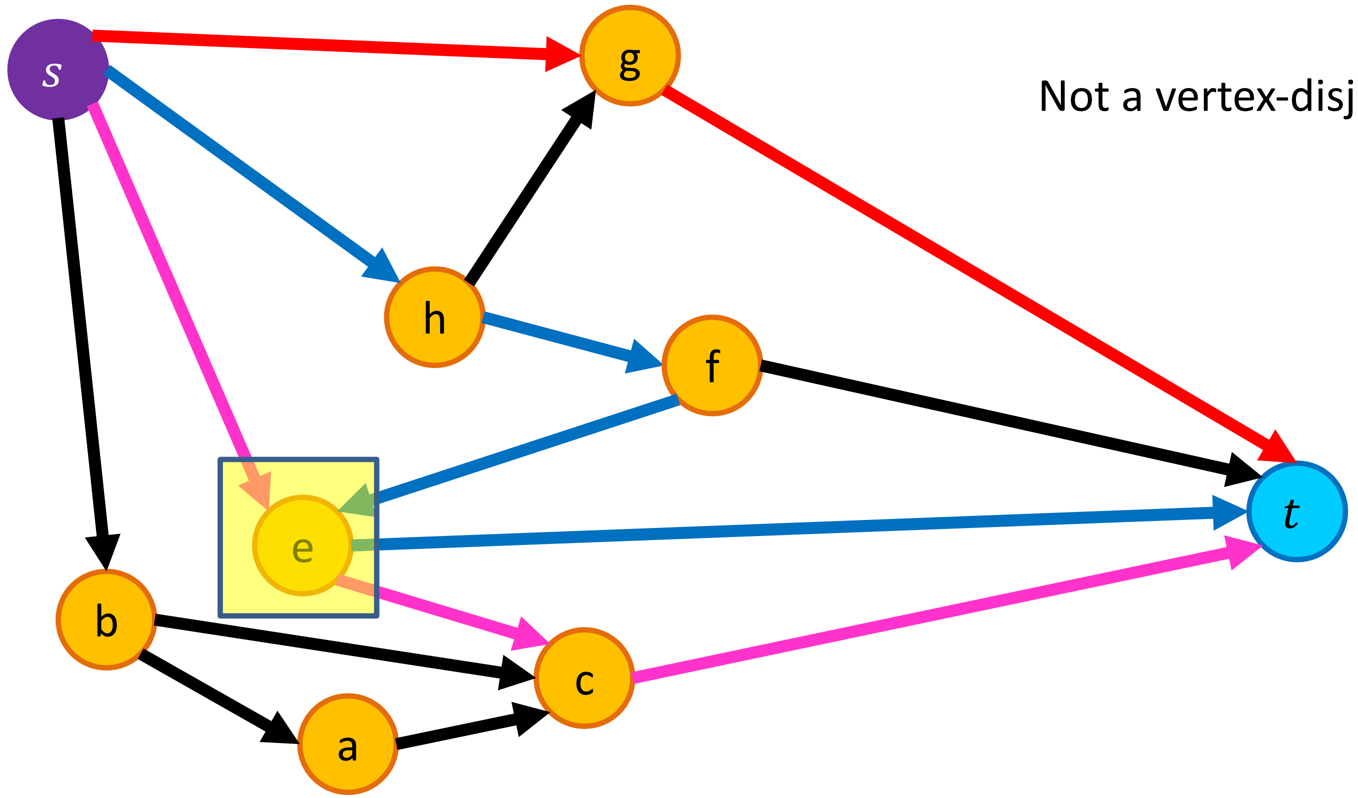
Make s and t the source and sink, give each edge capacity 1, find the max flow.



Set of edge-disjoint paths of size 4

Vertex-Disjoint Paths

Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no vertices



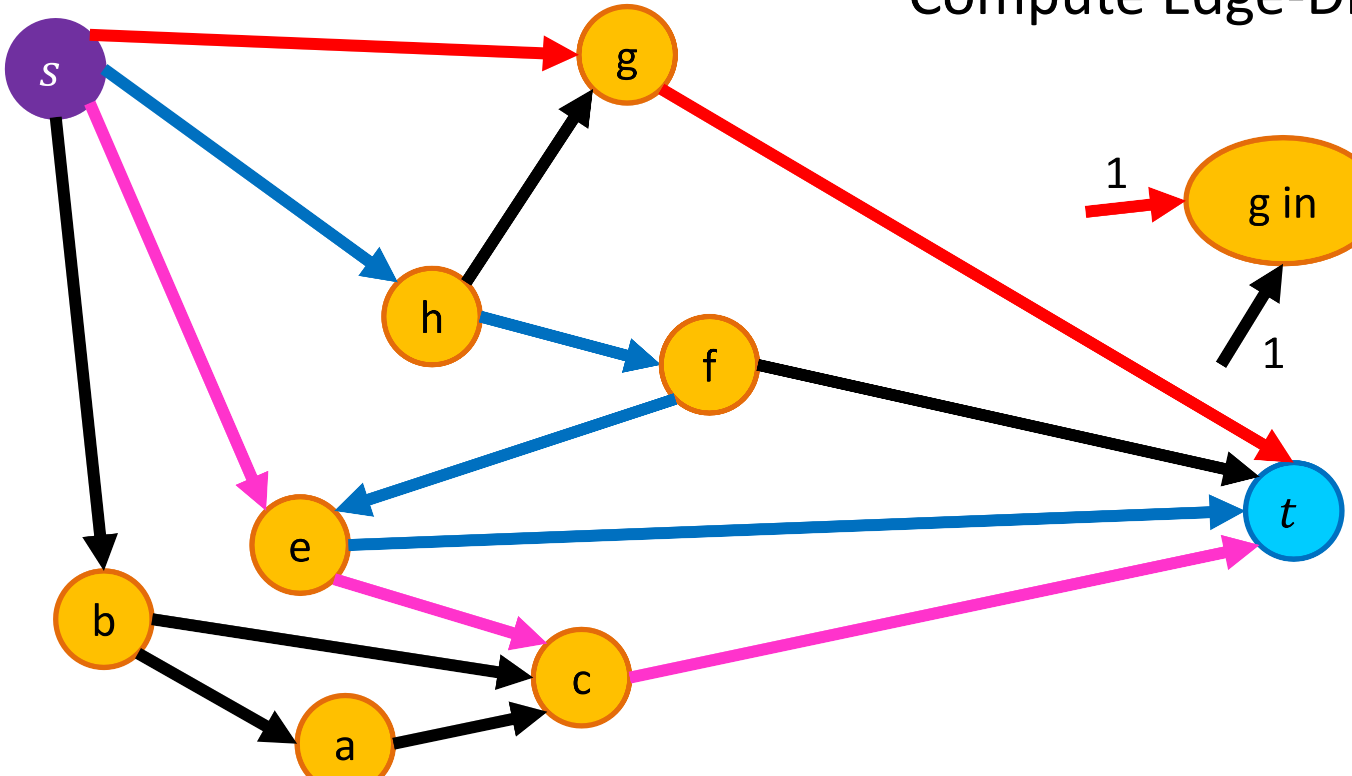
Not a vertex-disjoint path!

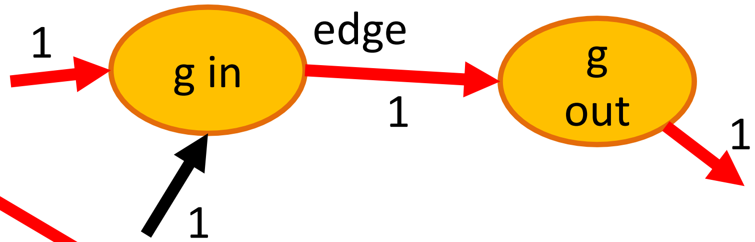
Vertex-Disjoint Paths Algorithm

Idea: Convert an instance of the vertex-disjoint paths problem into an instance of edge-disjoint paths

Make two copies of each node, one connected to incoming edges, the other to outgoing edges

Compute Edge-Disjoint paths on new graph

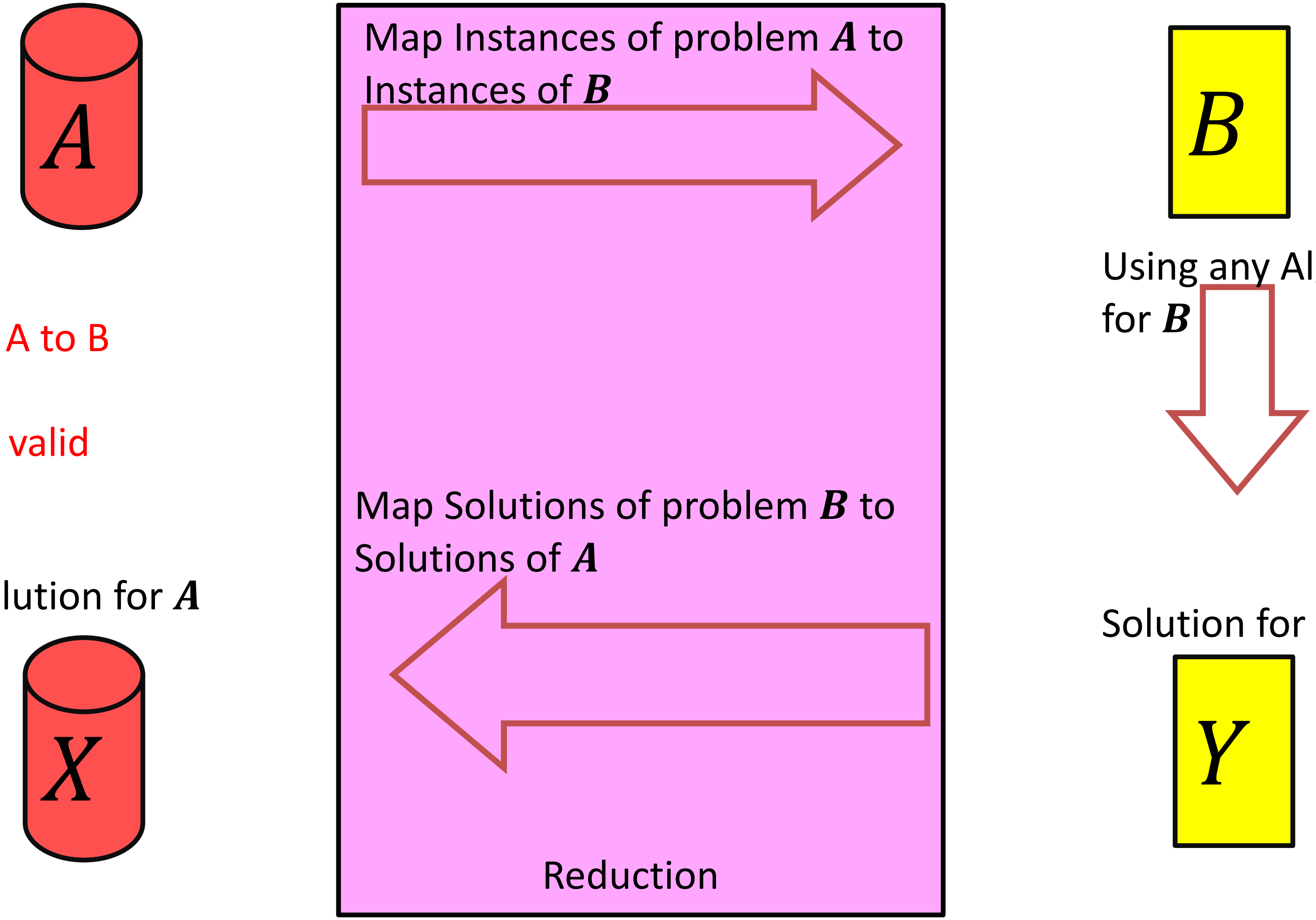




In General: Reduction

Problem we don't know how to solve

Problem we do know how to solve



Remember: need to show

1. How to map instance of A to B (and back)
2. Why solution to B was a valid solution to A

Worst-case lower-bound Proofs

Opening a door

A

Problem **A**

reduces to

B

Problem **B**

Lighting a fire

Alcohol, wood, matches

Y

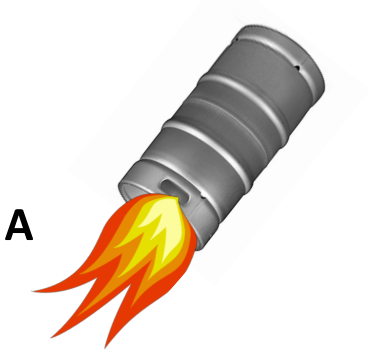
Algorithm for **B**

can be used to make

X

Algorithm for **A**

Keg cannon battering ram

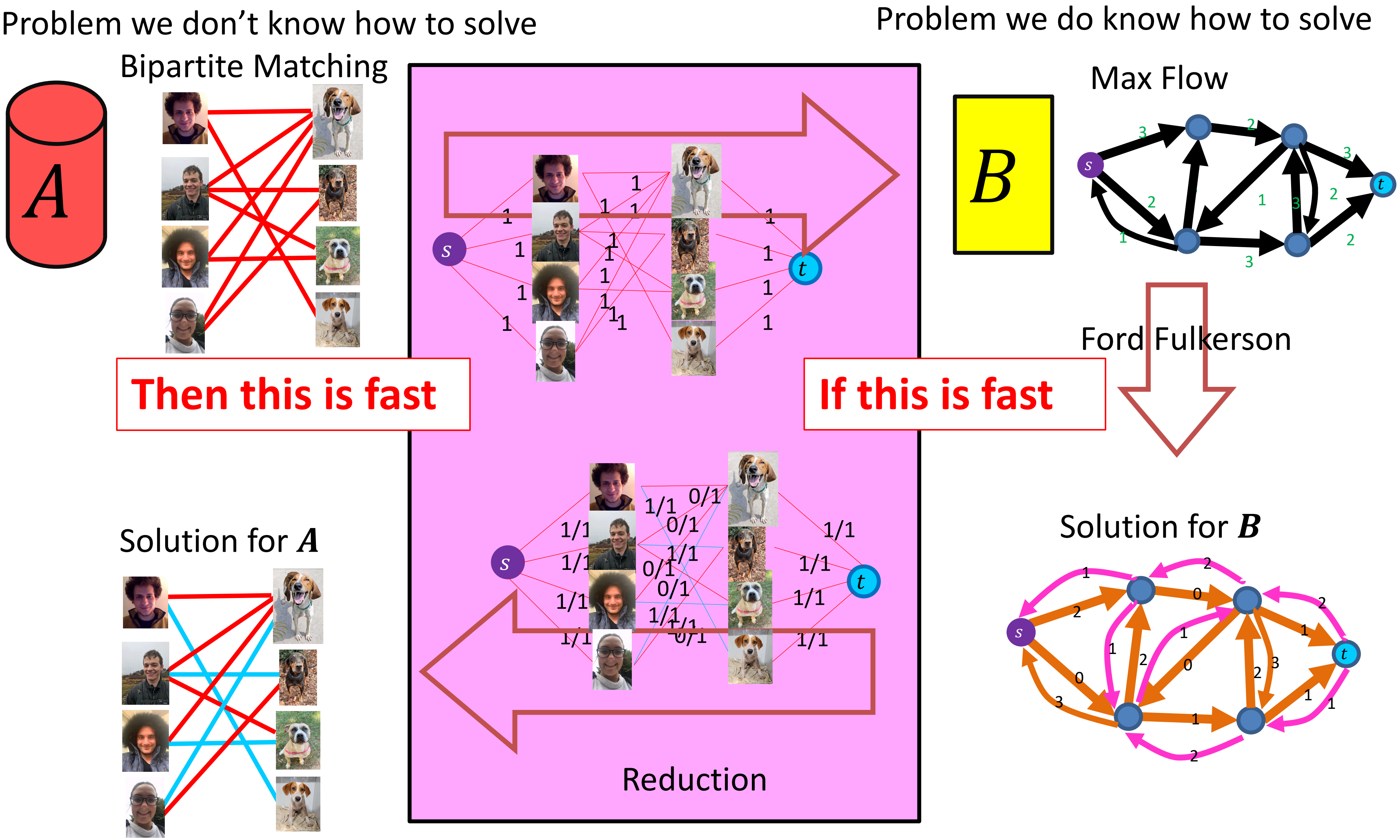


A is not a harder problem than B

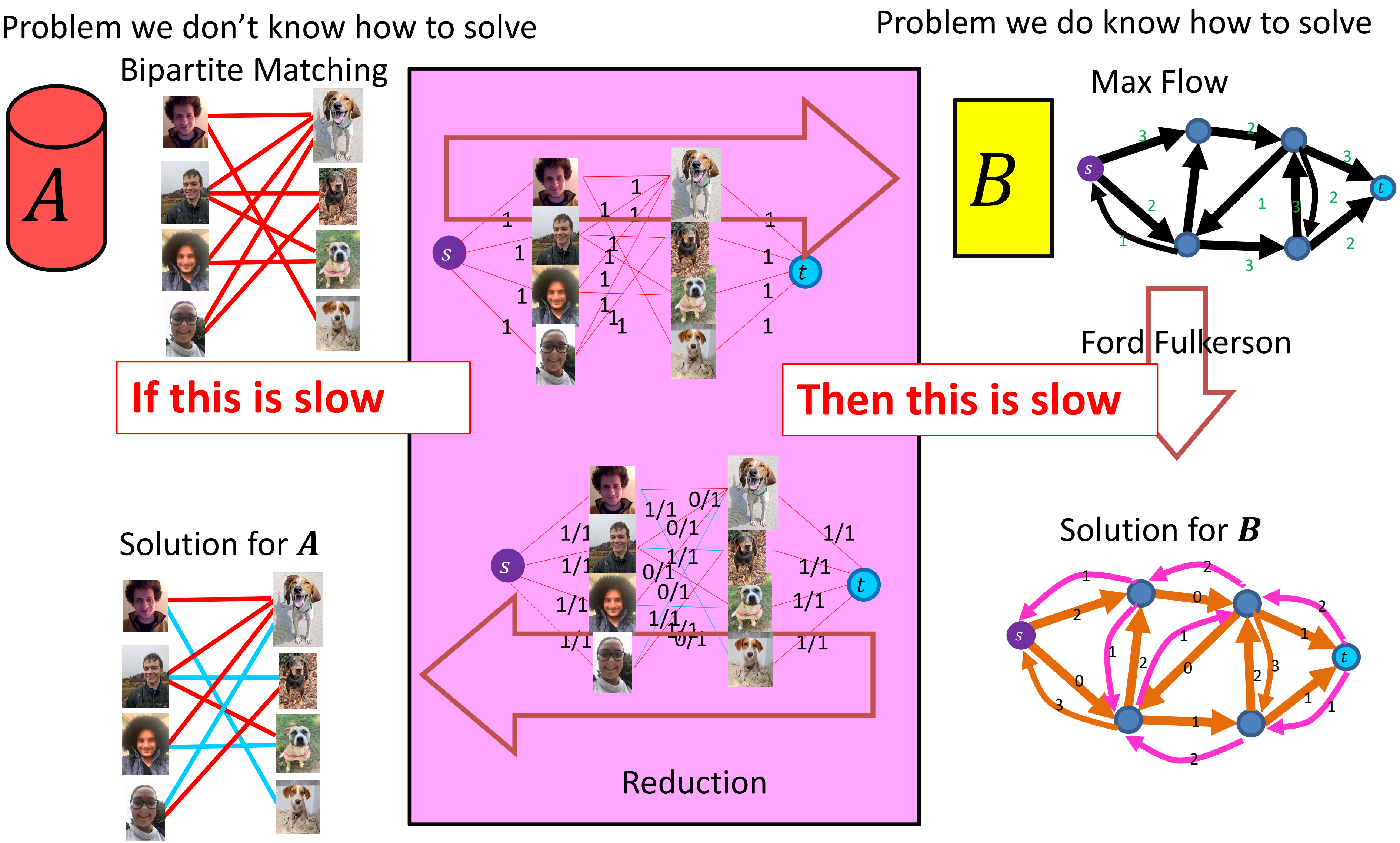
$$\boldsymbol{A \leq B}$$

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

Bipartite Matching Reduction

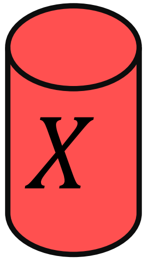


Bipartite Matching Reduction

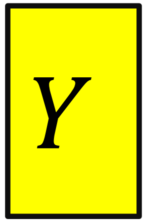


Proof of Lower Bound by Reduction

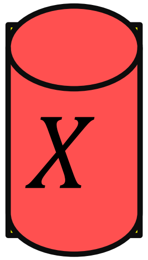
<u>To Show: Y is slow</u>

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

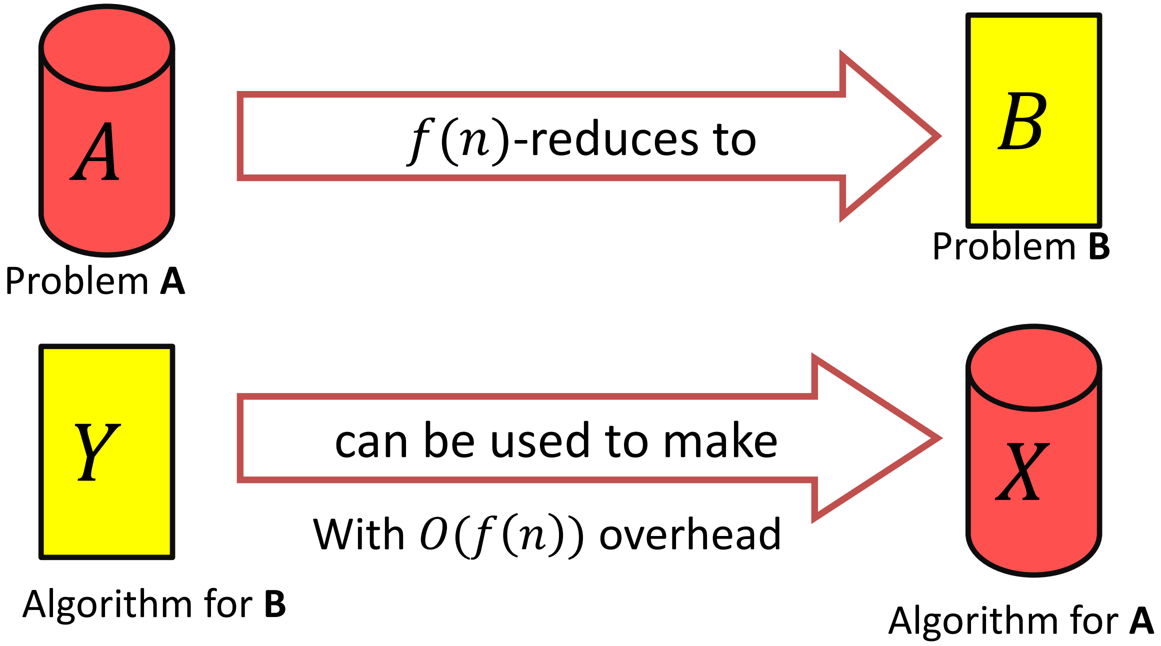
2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)

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

Reduction Proof Notation



A is not a harder problem than B

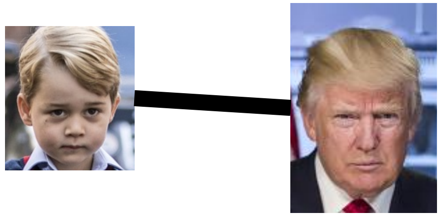
$$A \leq B$$

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

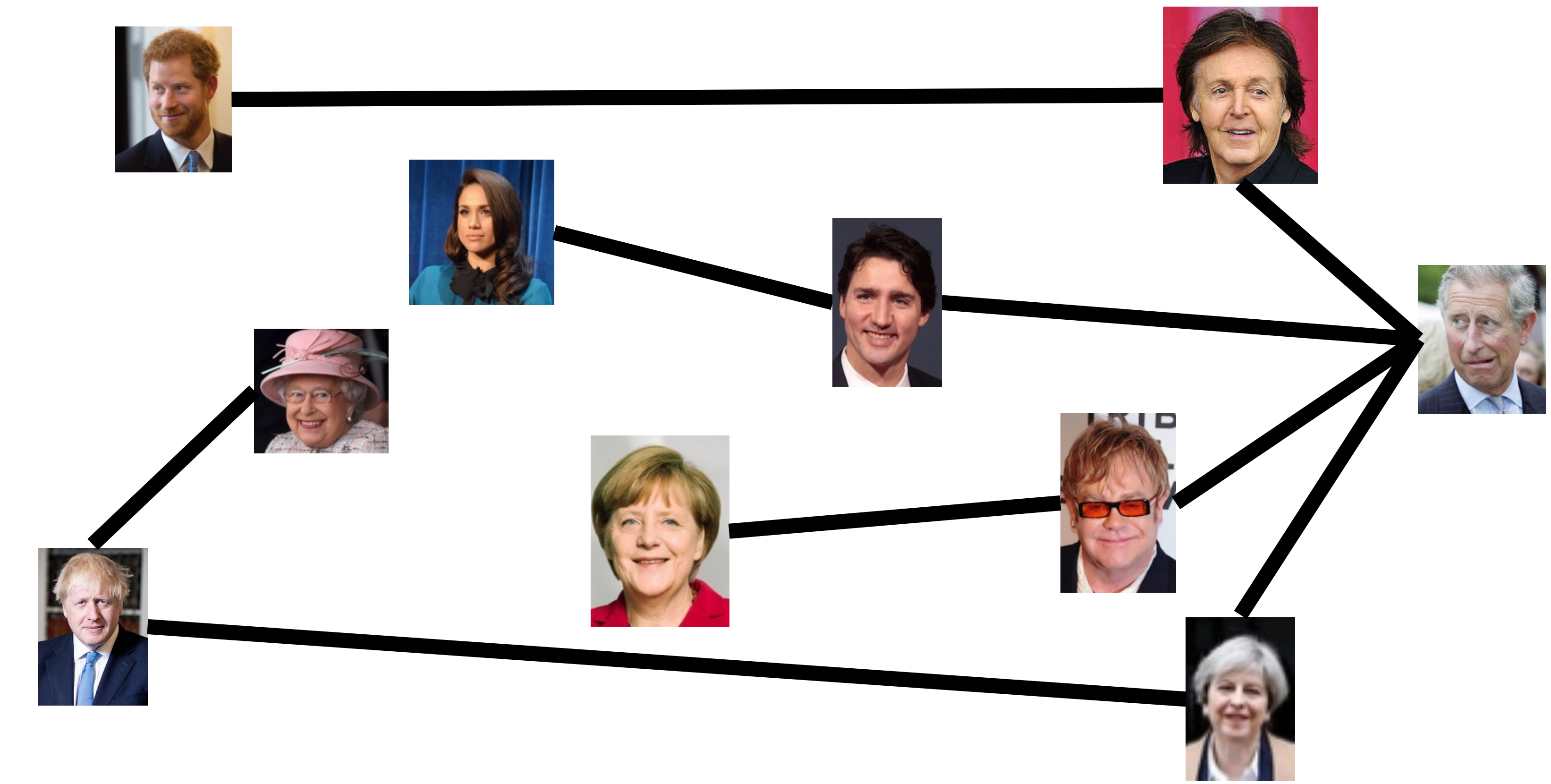
$$A \leq_{f(n)} B$$

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

Party Problem



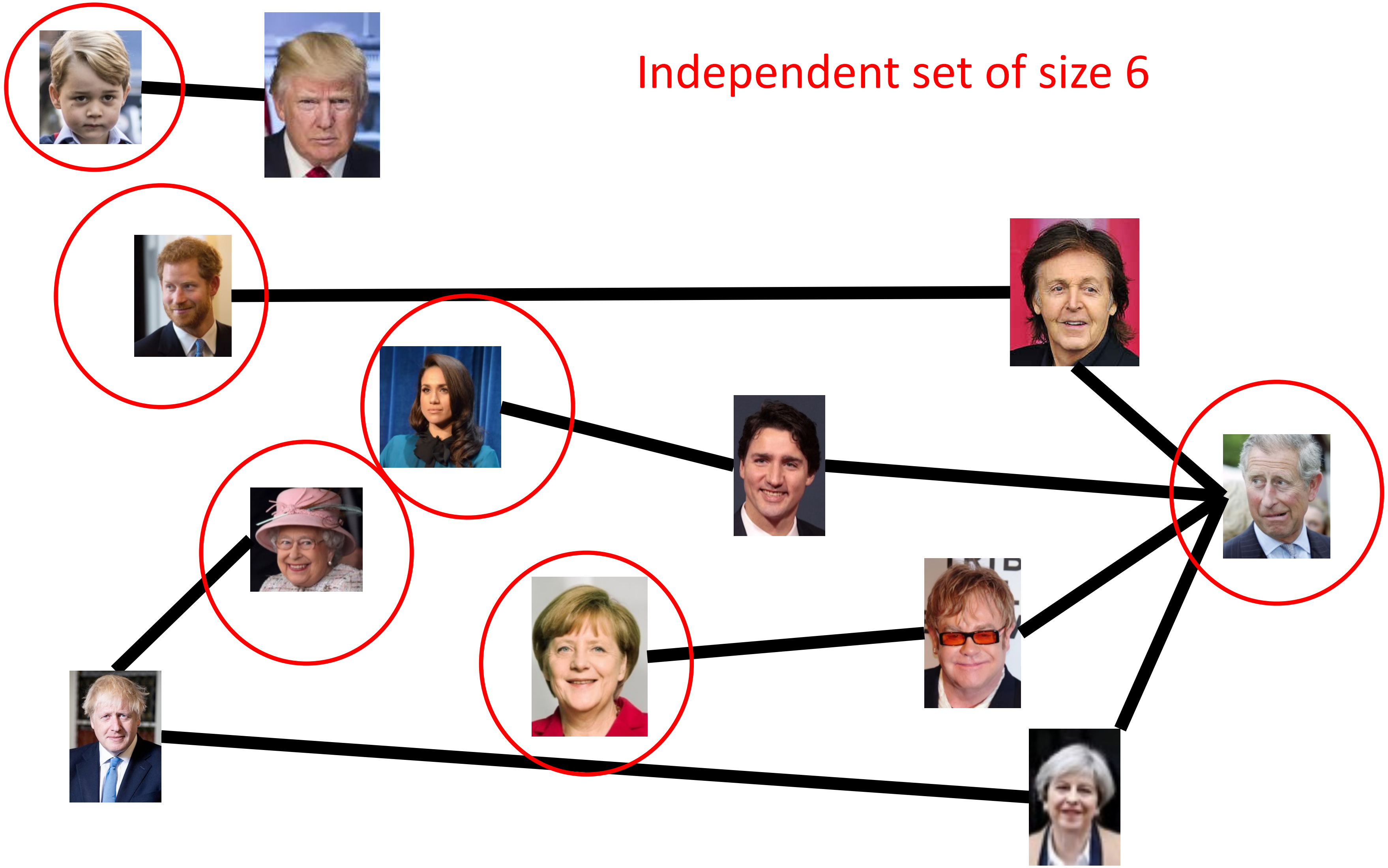
Draw Edges between people who don’t get along
Find the maximum number of people who get along



Maximum Independent Set

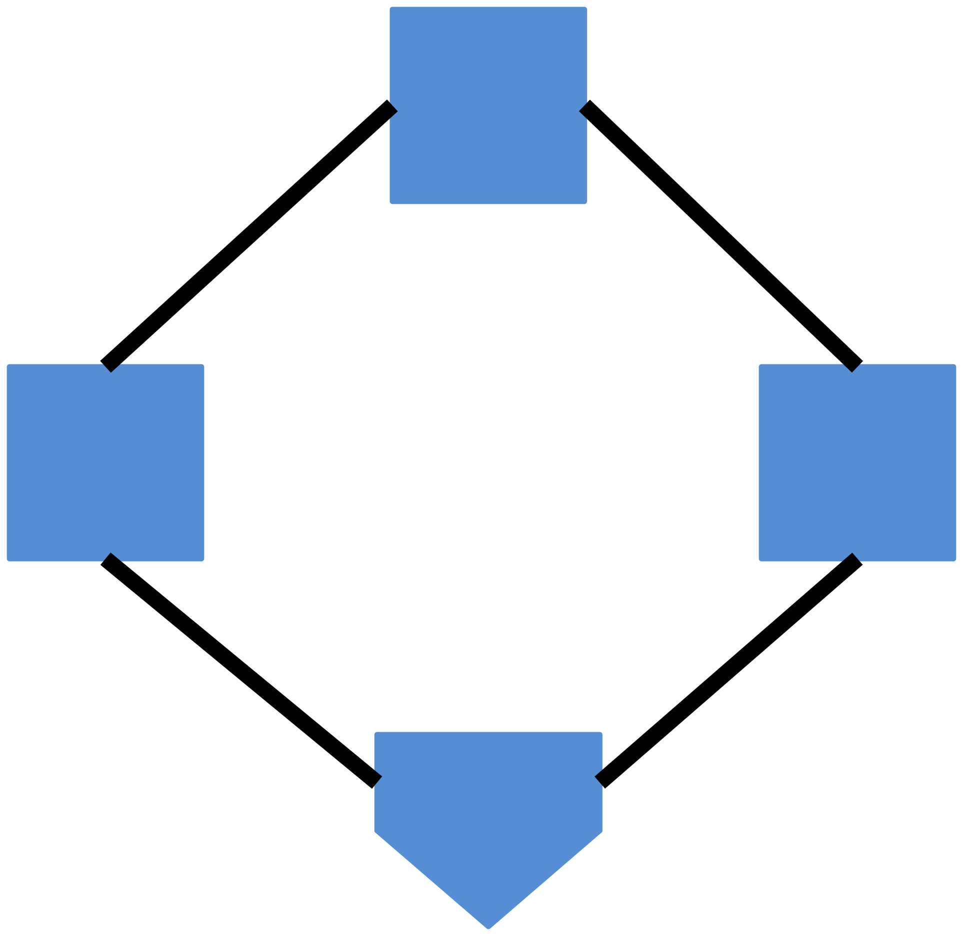
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

Example

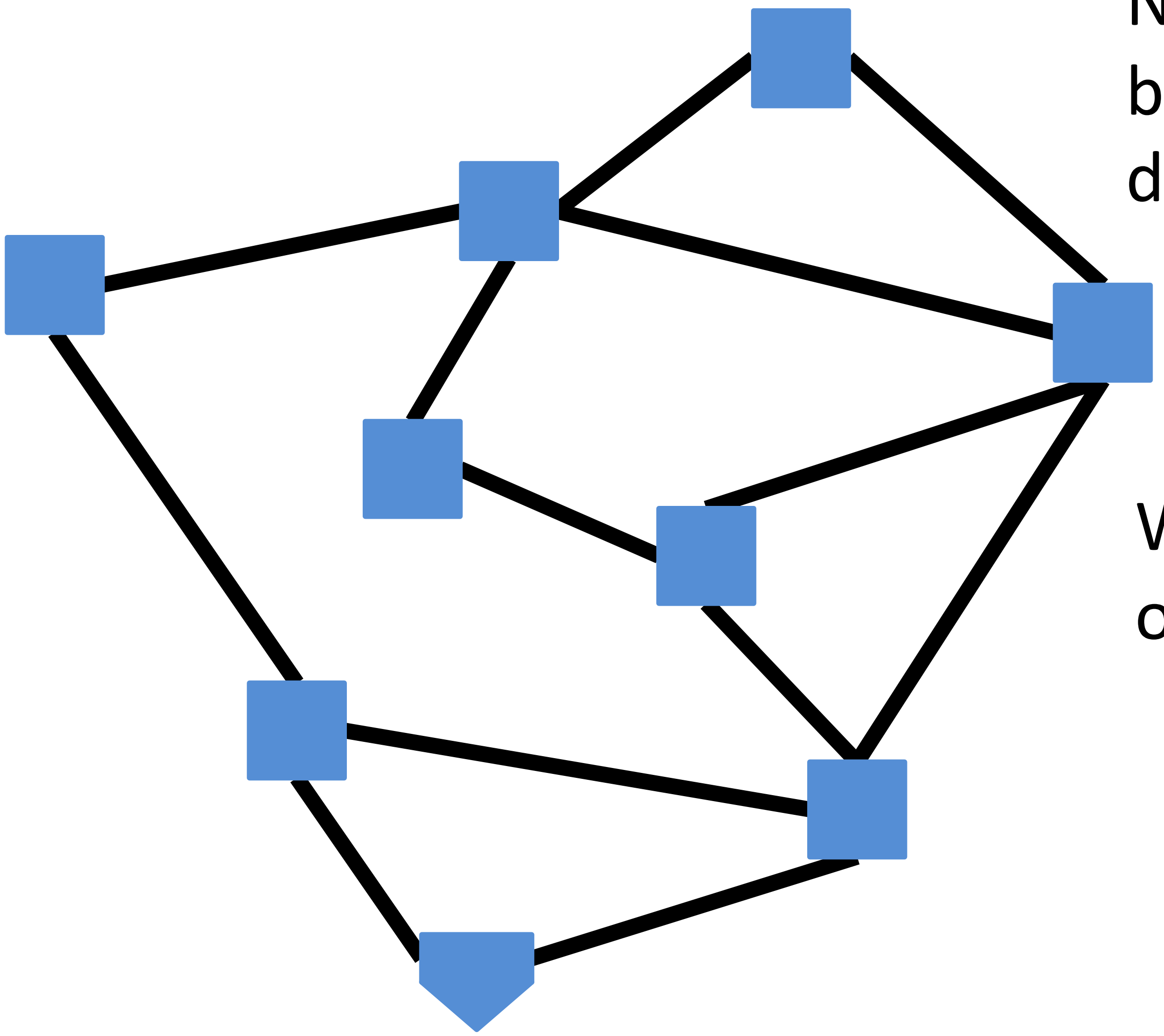


Independent set of size 6

Generalized Baseball



Generalized Baseball



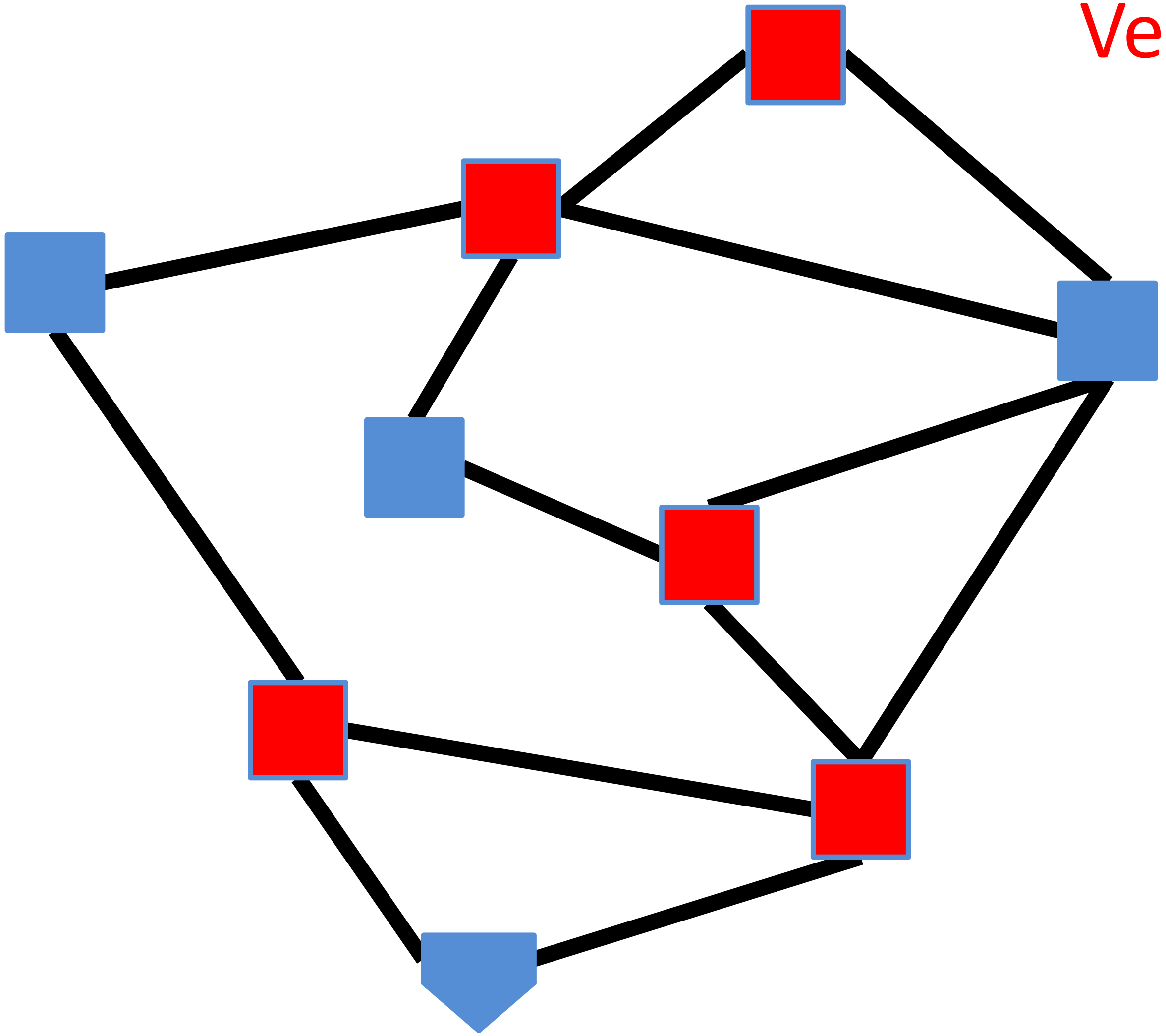
Need to place defenders on bases such that every edge is defended

What's the fewest number of defenders needed?

Minimum Vertex Cover

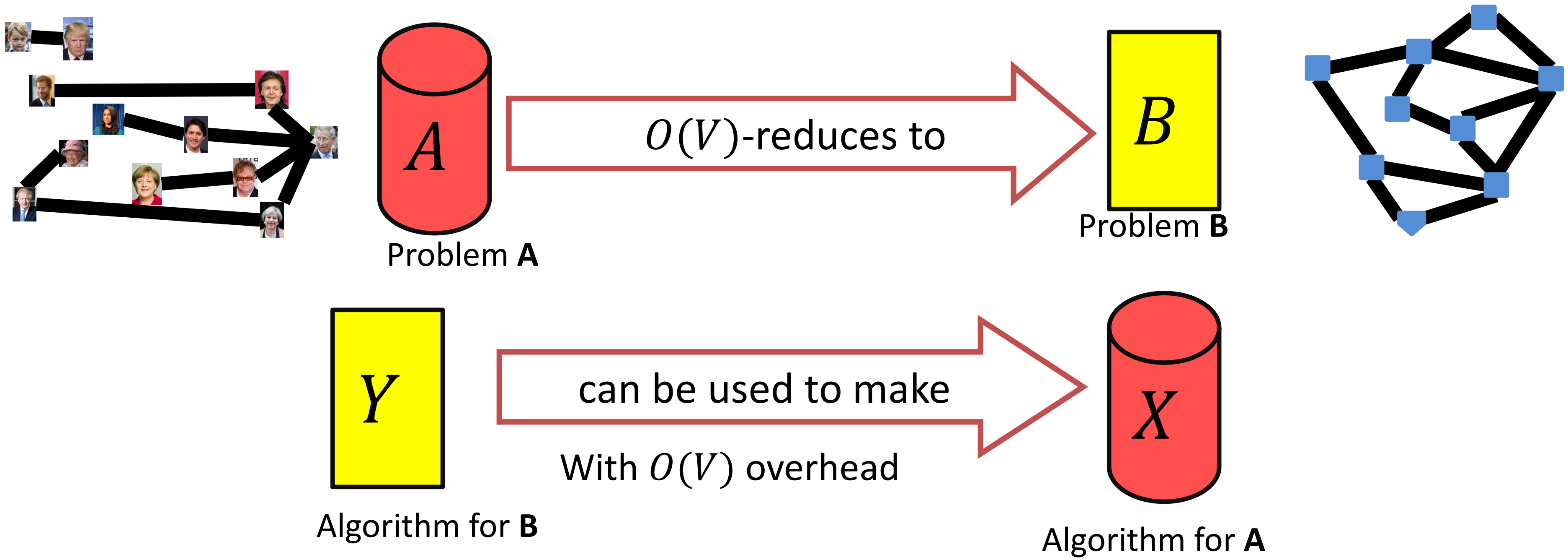
- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

Example



Vertex cover of size 5

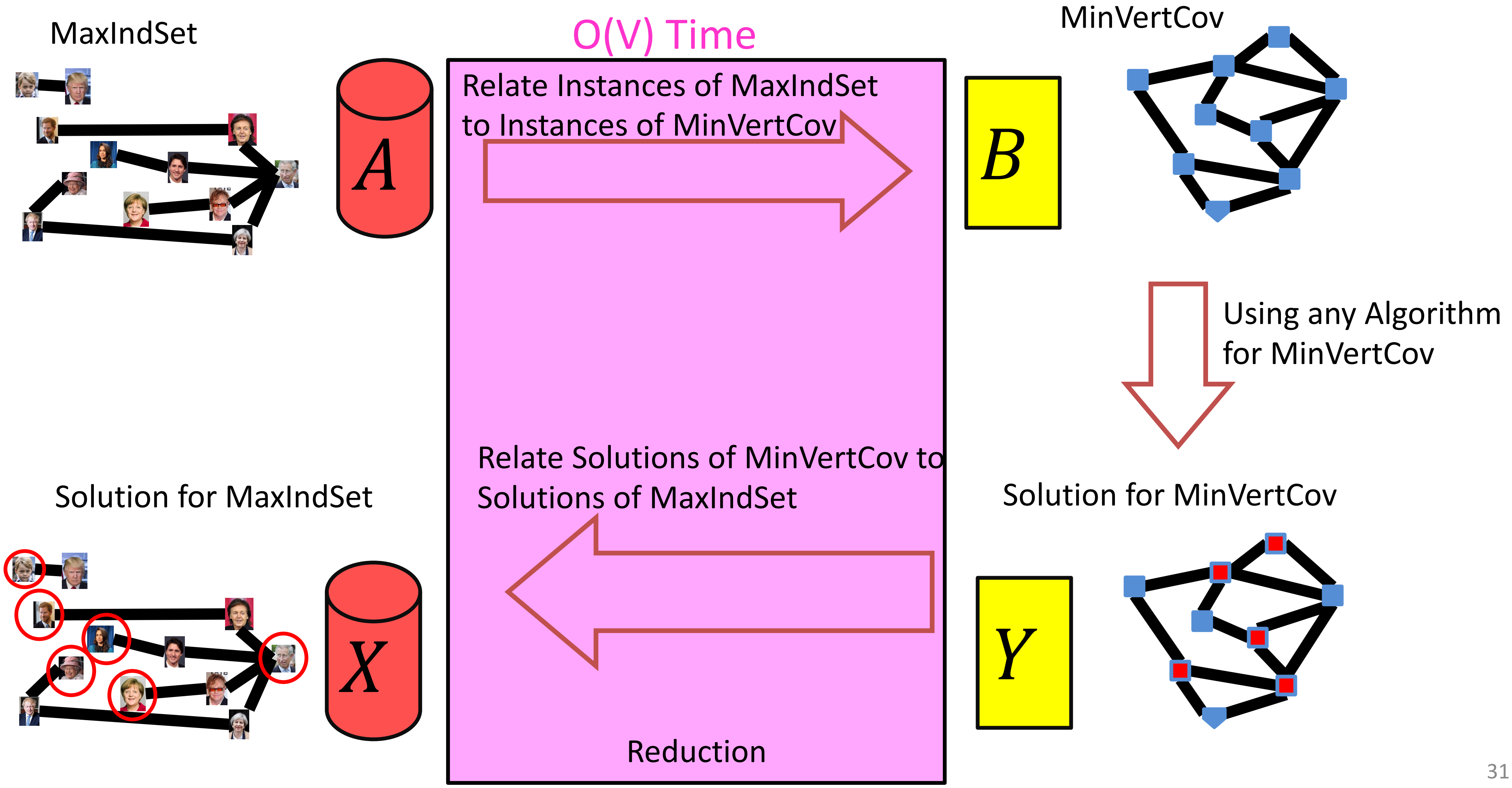
MaxIndSet$\leq_V$MinVertCov



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

$$A \leq_V B$$

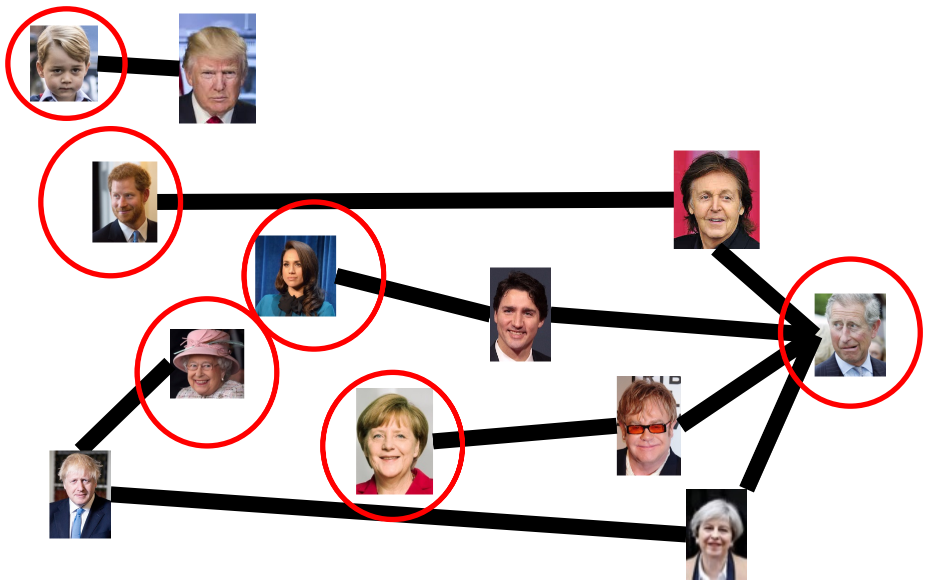
We need to build this Reduction



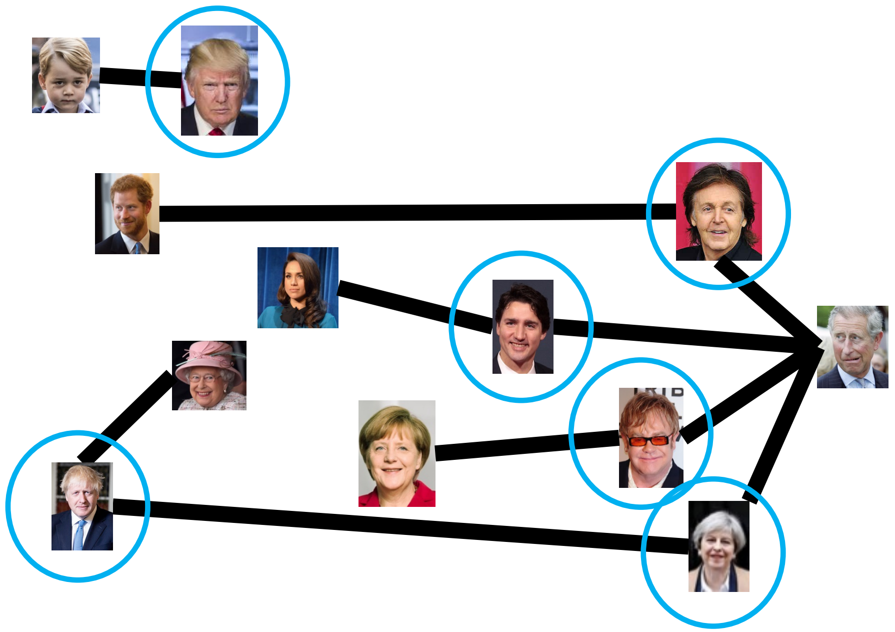
Reduction Idea

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

Independent Set



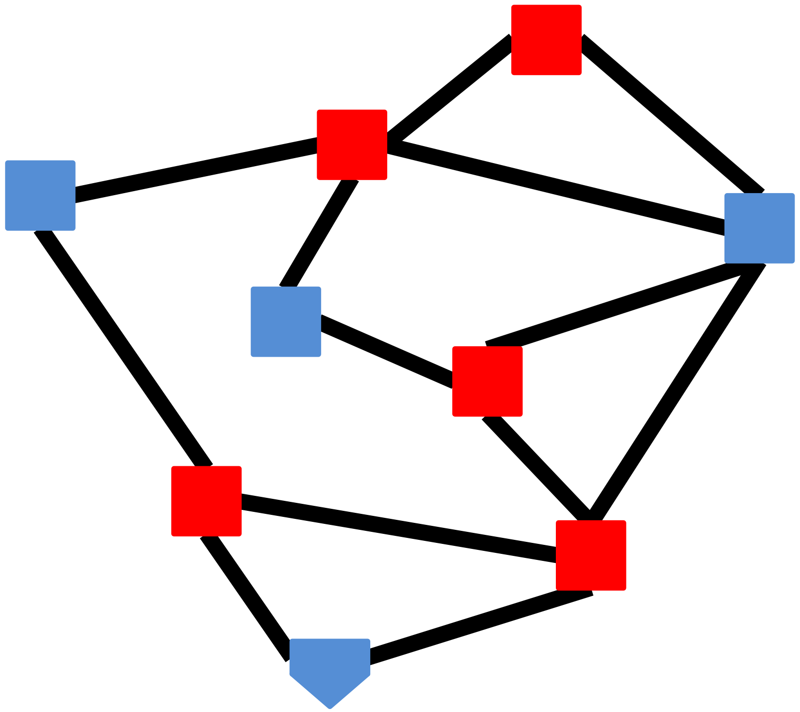
Vertex Cover



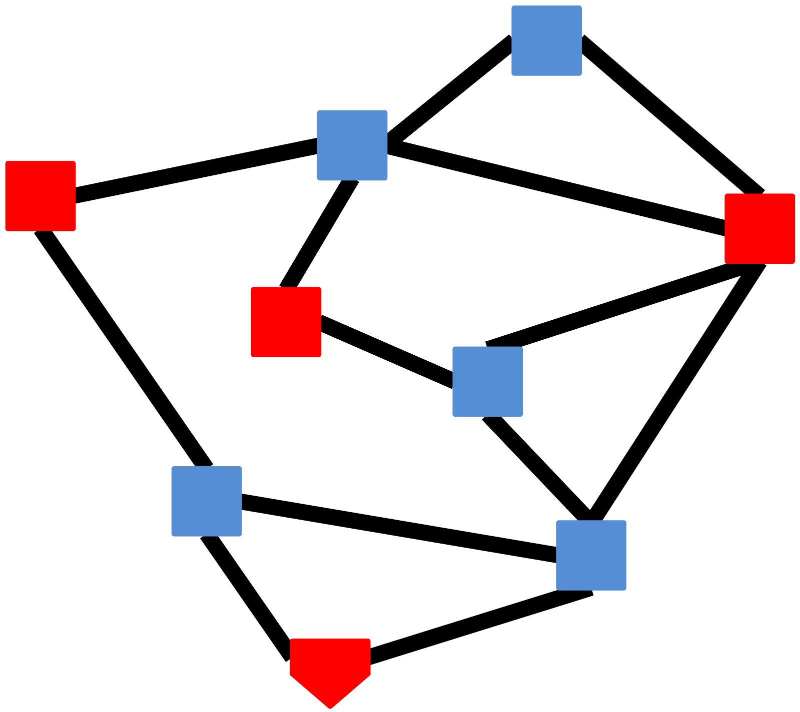
Reduction Idea

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

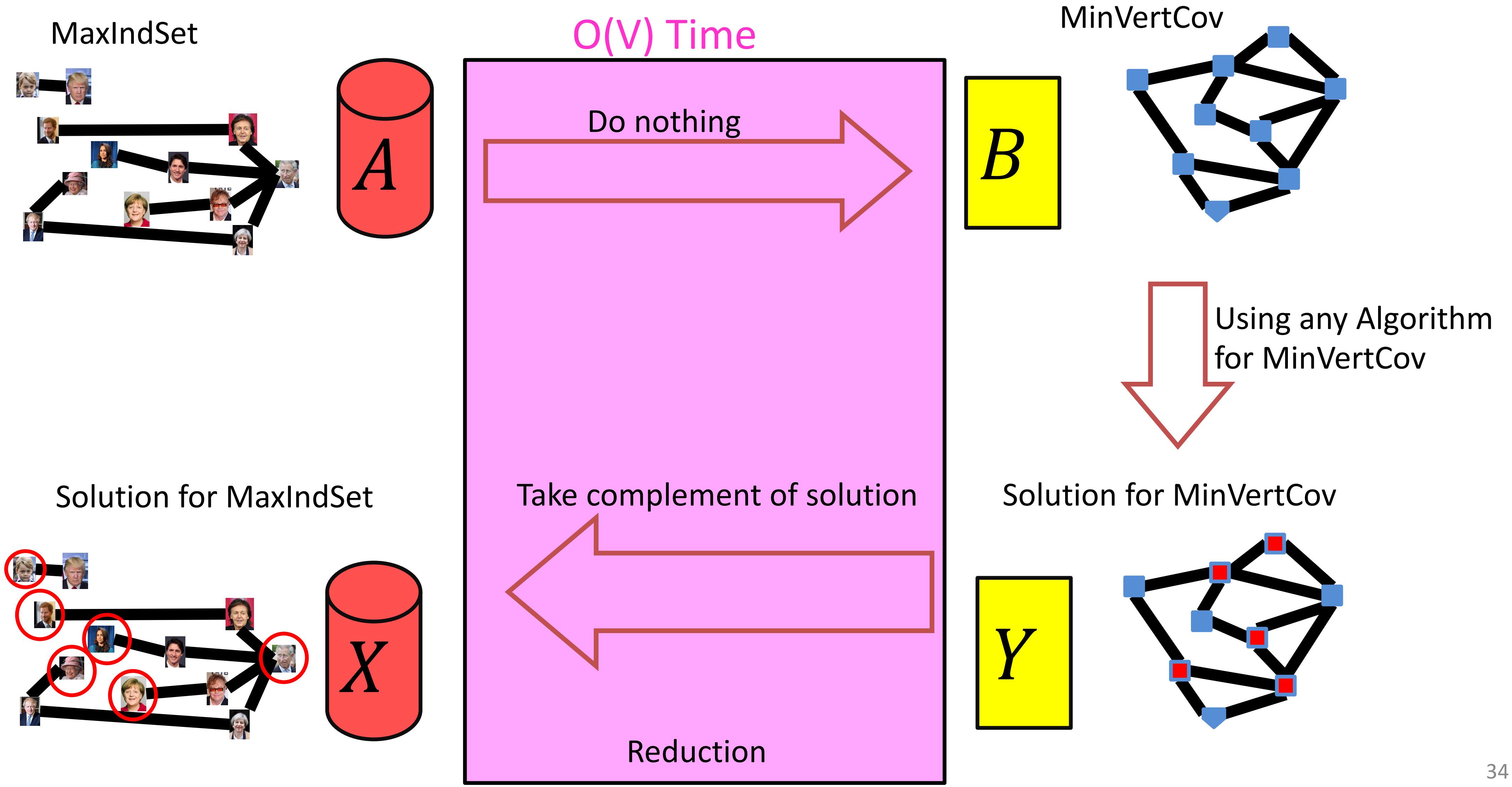
Vertex Cover



Independent Set



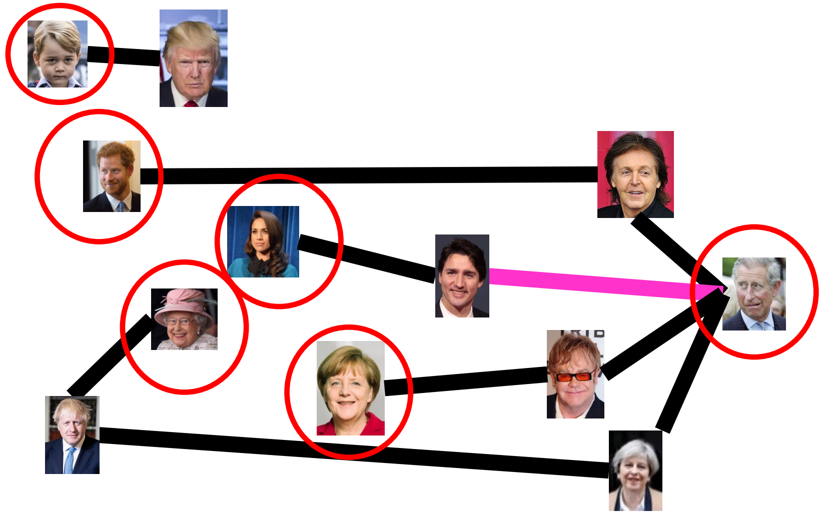
MaxVertCov V-Time Reducible to MinIndSet



Proof: ⇒

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 $(x, y) \in E$

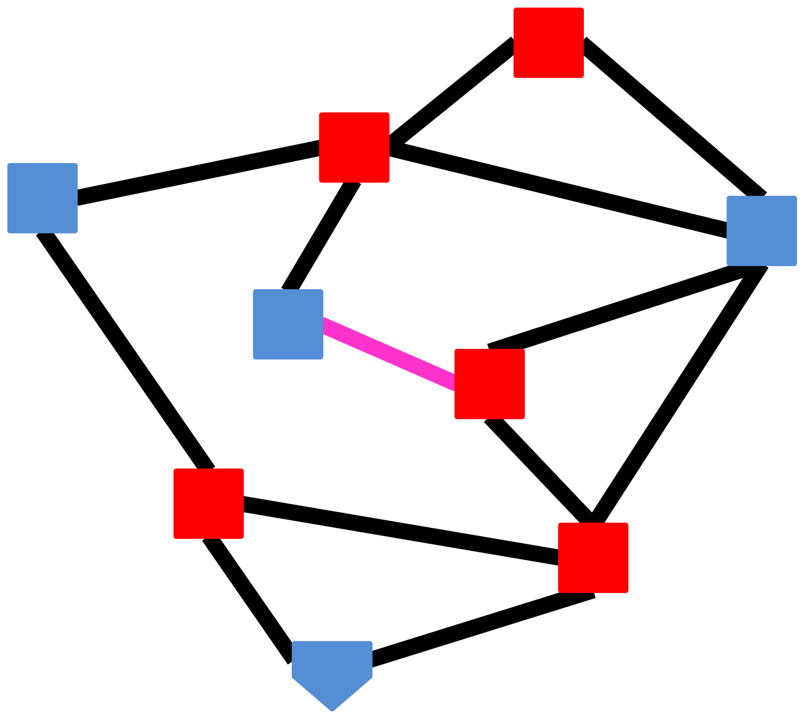
If $x \in S$ then $y \notin S$, because otherwise S would not be an independent set

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 $\mathrm{V} - S$ be a vertex cover



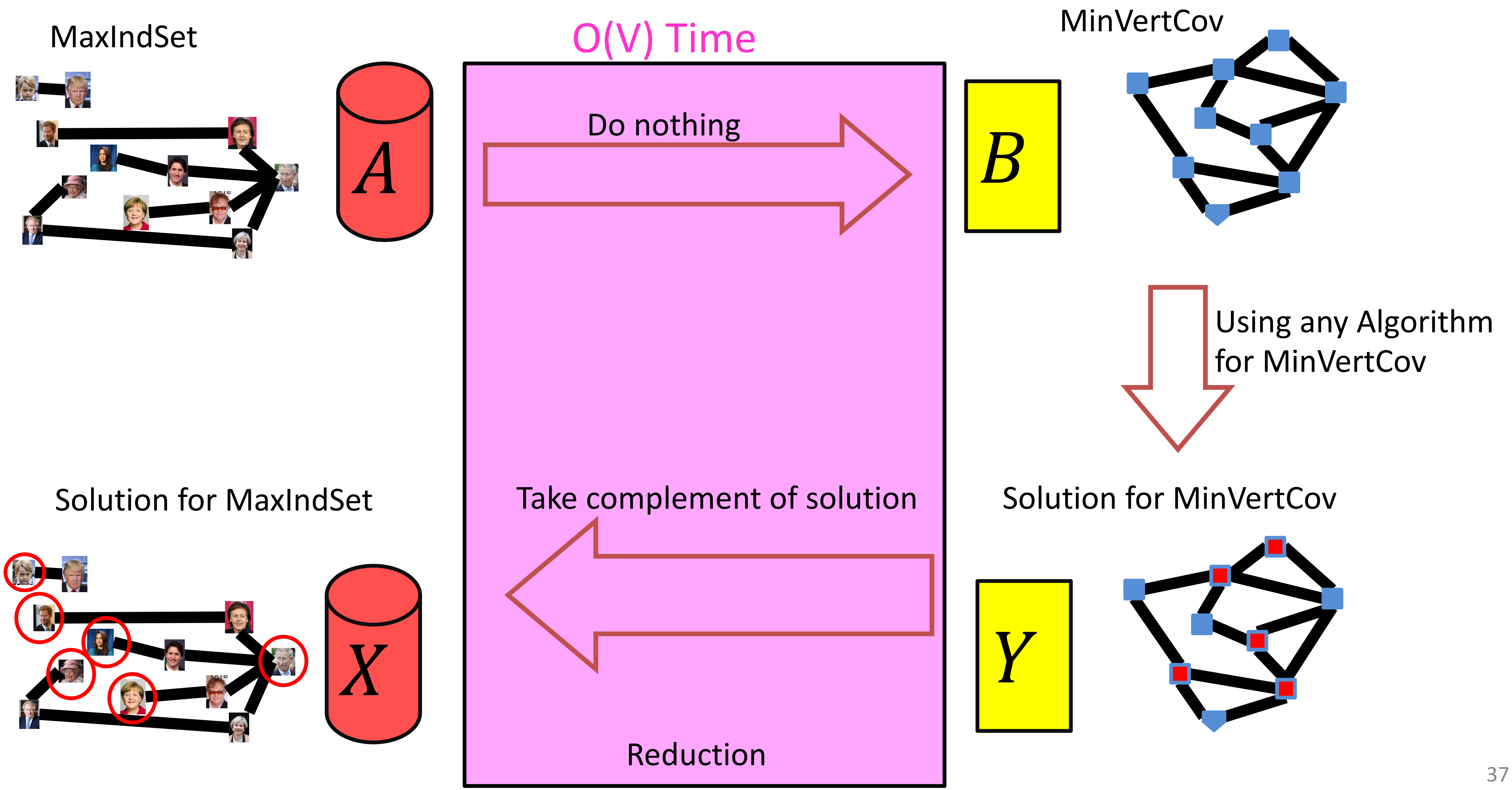
Consider any edge $(x, y) \in E$

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

Therefore x and y are not both in S,

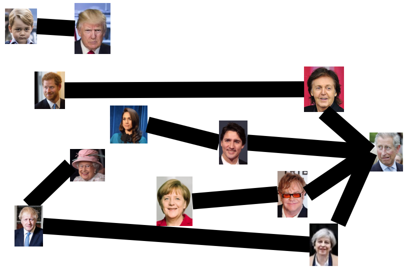
No edge has both end-nodes in S, thus S is an independent set

MaxVertCov V-Time Reducible to MinIndSet



MaxVertCov V-Time Reducible to MinIndSet

MaxIndSet



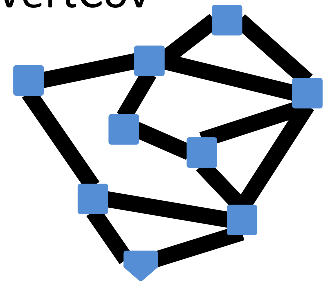
A

O(V) Time

Do nothing

B

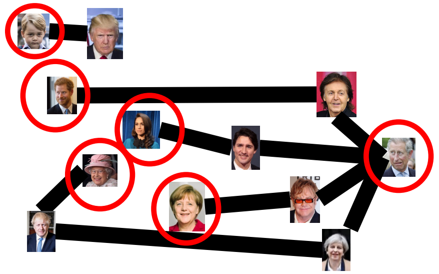
MinVertCov



Using any Algorithm for MinVertCov

We needed our proof to show that this works!

Solution for MaxIndSet



X

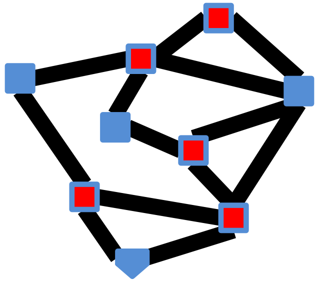
Take complement of solution

If there was a larger independent set, there would have been a smaller vertex cover!

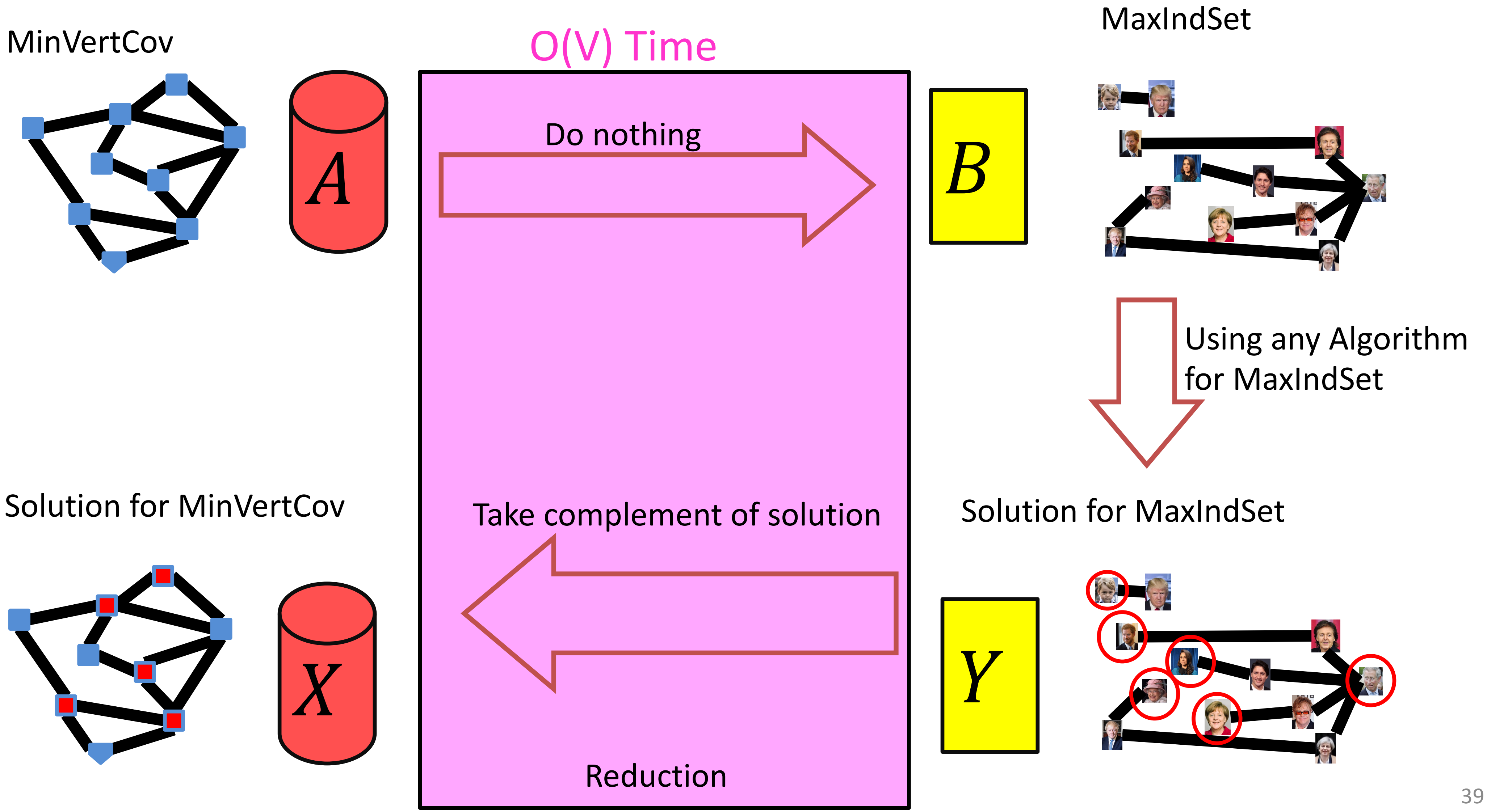
Reduction

Solution for MinVertCov

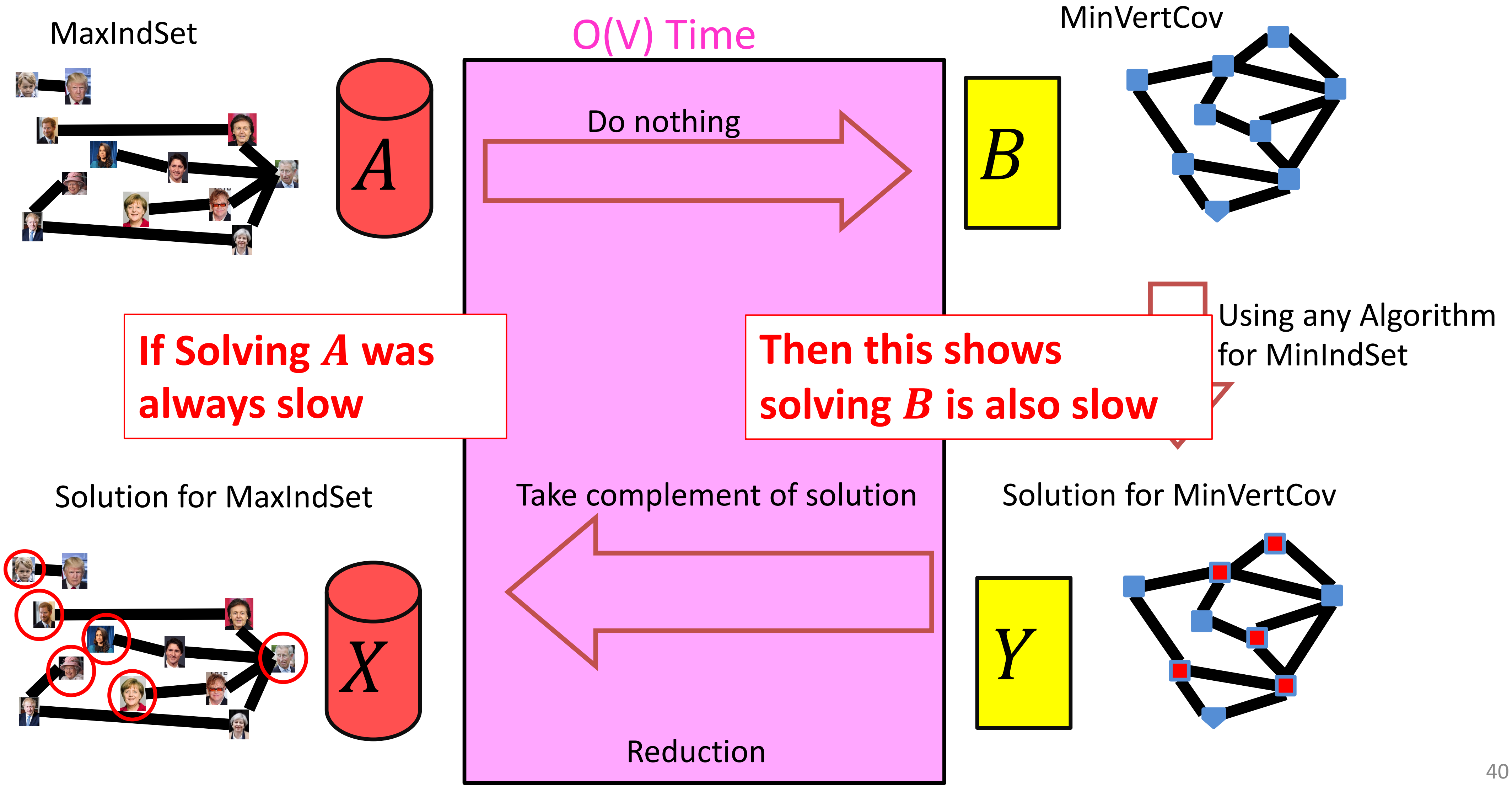
Y



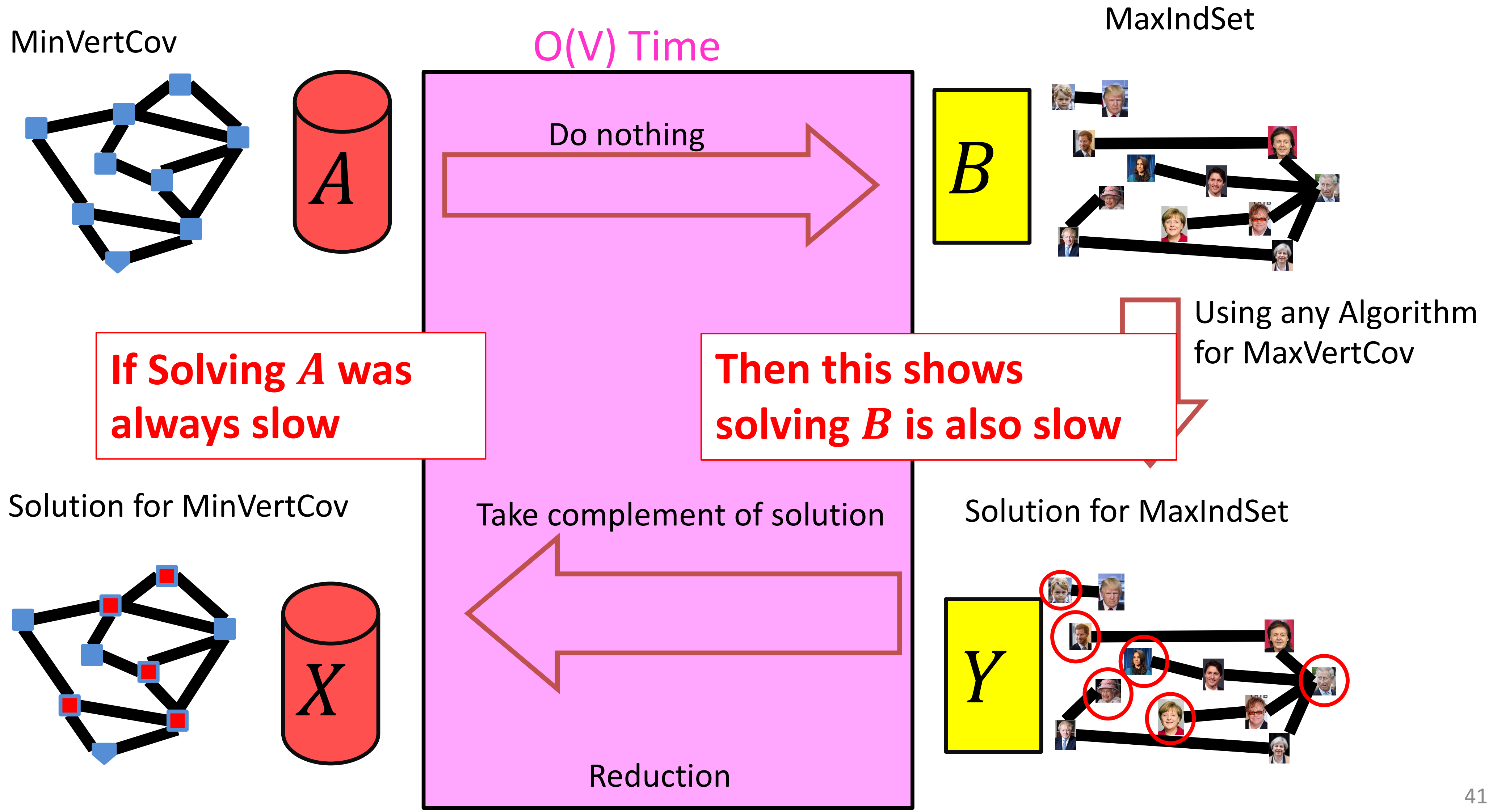
MaxIndSet V-Time Reducible to MinVertCov



Corollary


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

Corollary


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

Conclusion

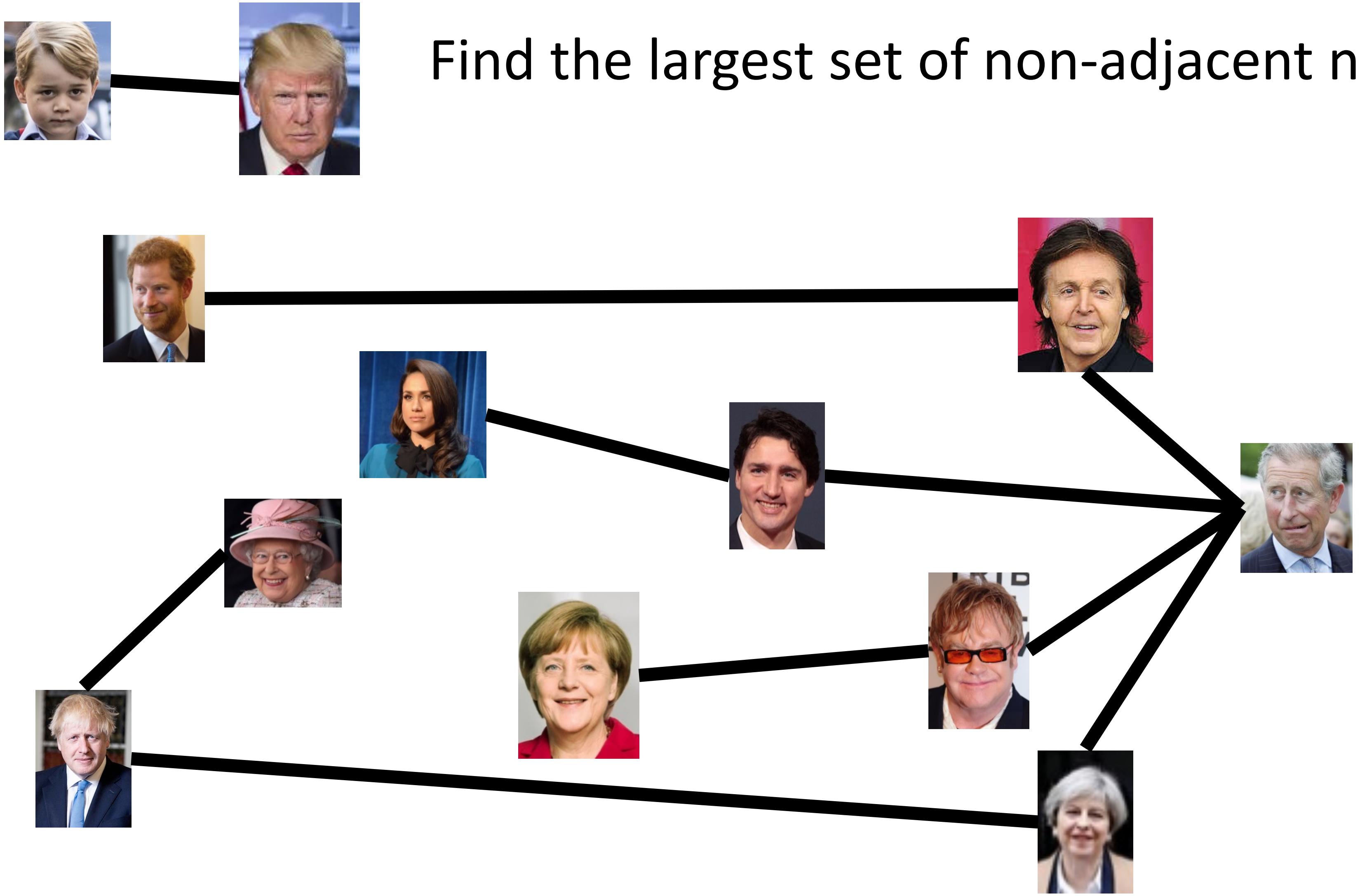
- MaxIndSet and MinVertCov are either both fast, or both slow
  - Spoiler alert: We don’t know which!
    - (But we think they’re both slow)
  - Both problems are NP-Complete

Mid-class warm up:

What is a Decision Problem?

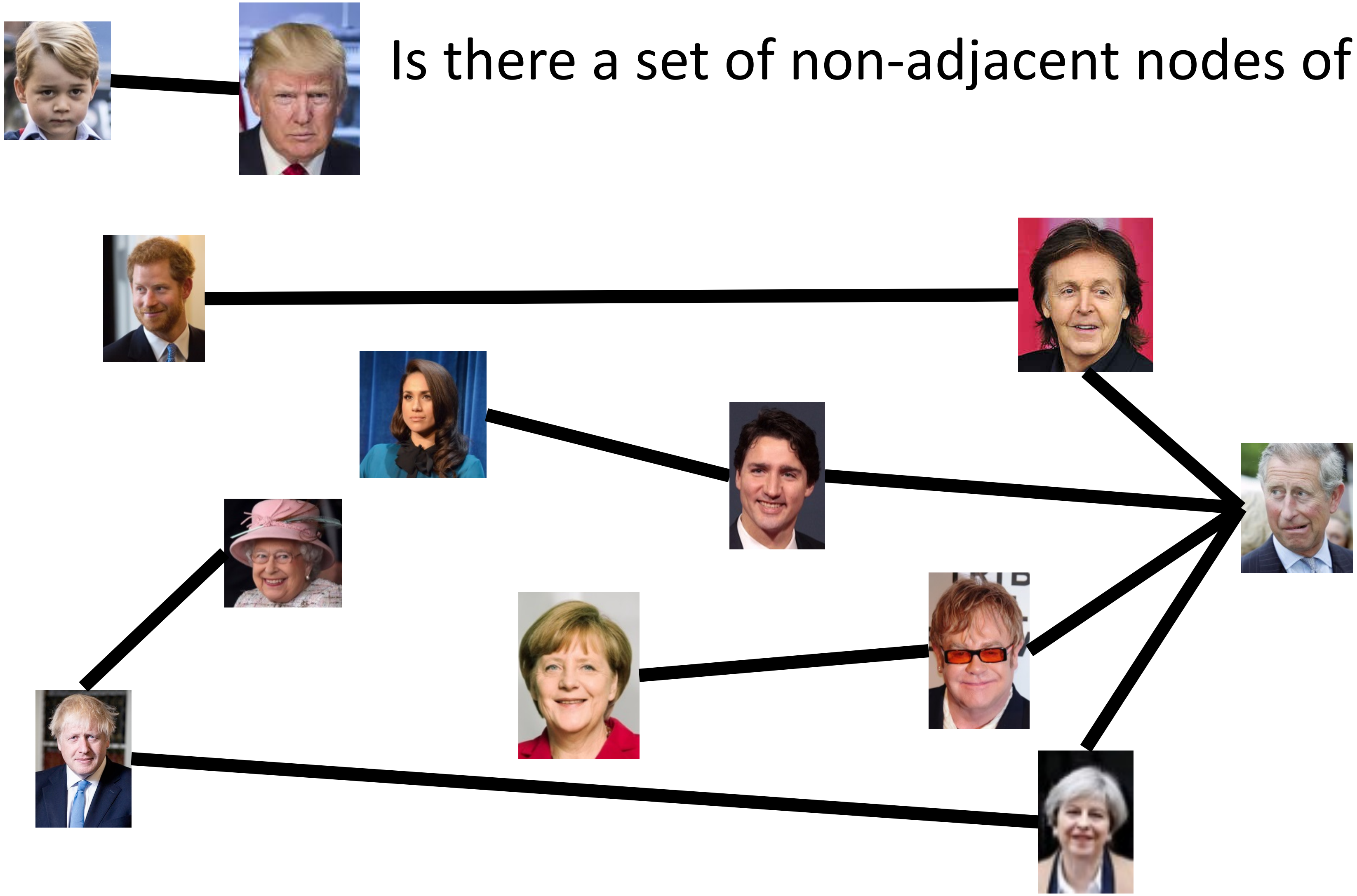
Max Independent Set

Find the largest set of non-adjacent nodes



k Independent Set

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



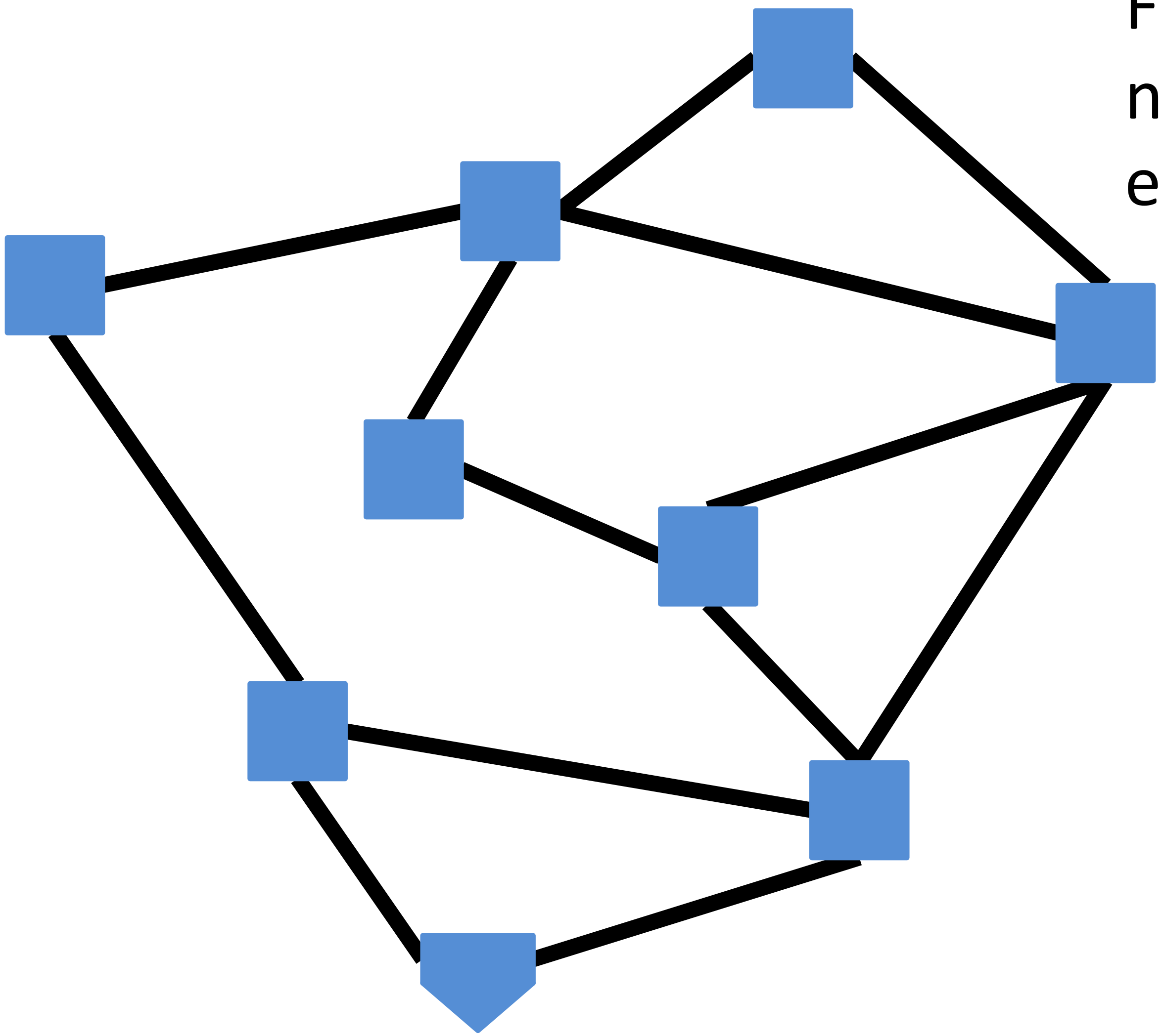
Maximum Independent Set

- 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

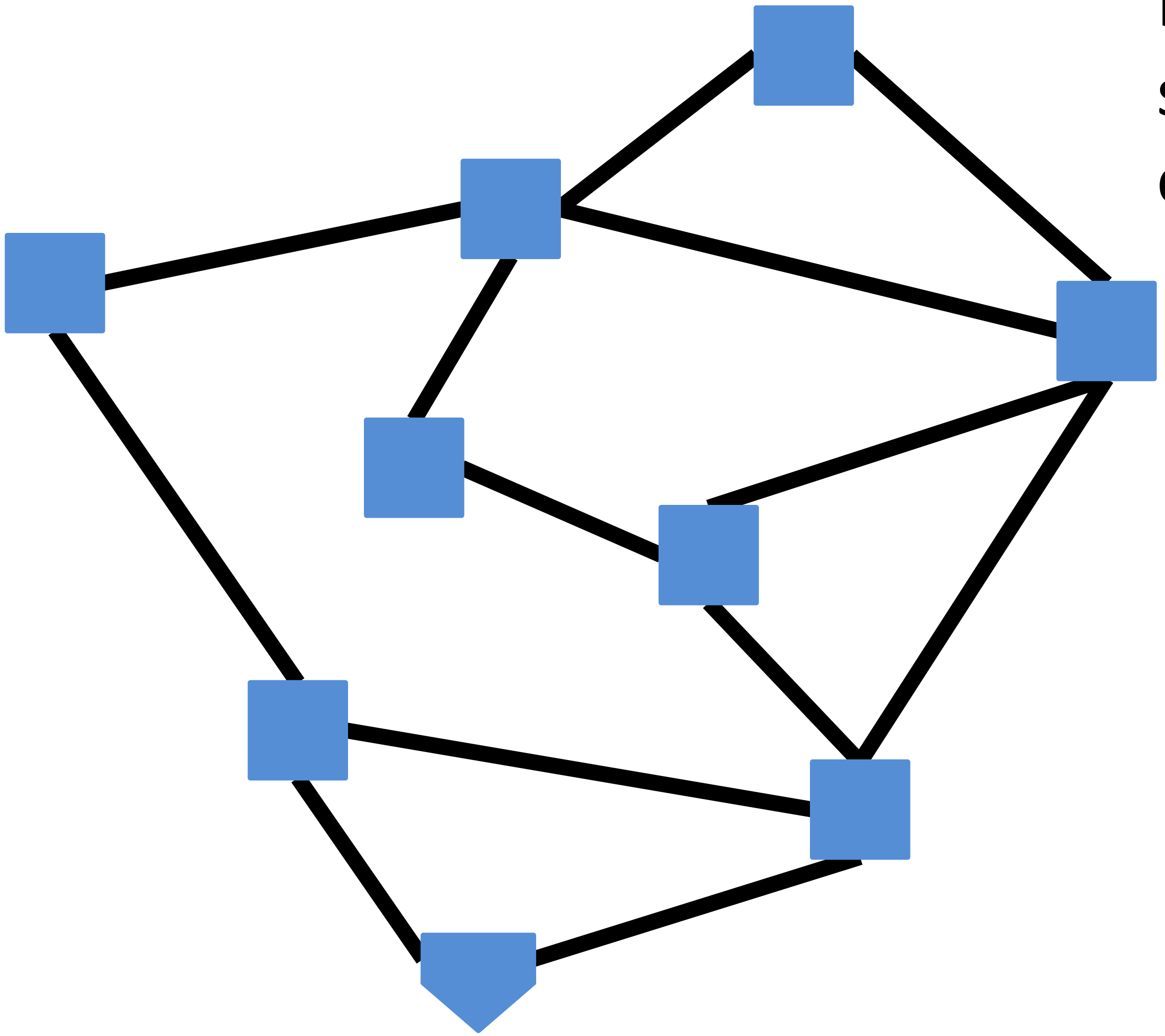
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- k Independent Set Problem: Given a graph $G = (V, E)$ and a number k, **determine whether there is an independent set $\boldsymbol{S}$ of size $\boldsymbol{k}$**

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?

If we can solve this

- Search Problems:
  – Find a solution
    - Output is complex
  – Give a vertex cover of size k

Then we can solve this

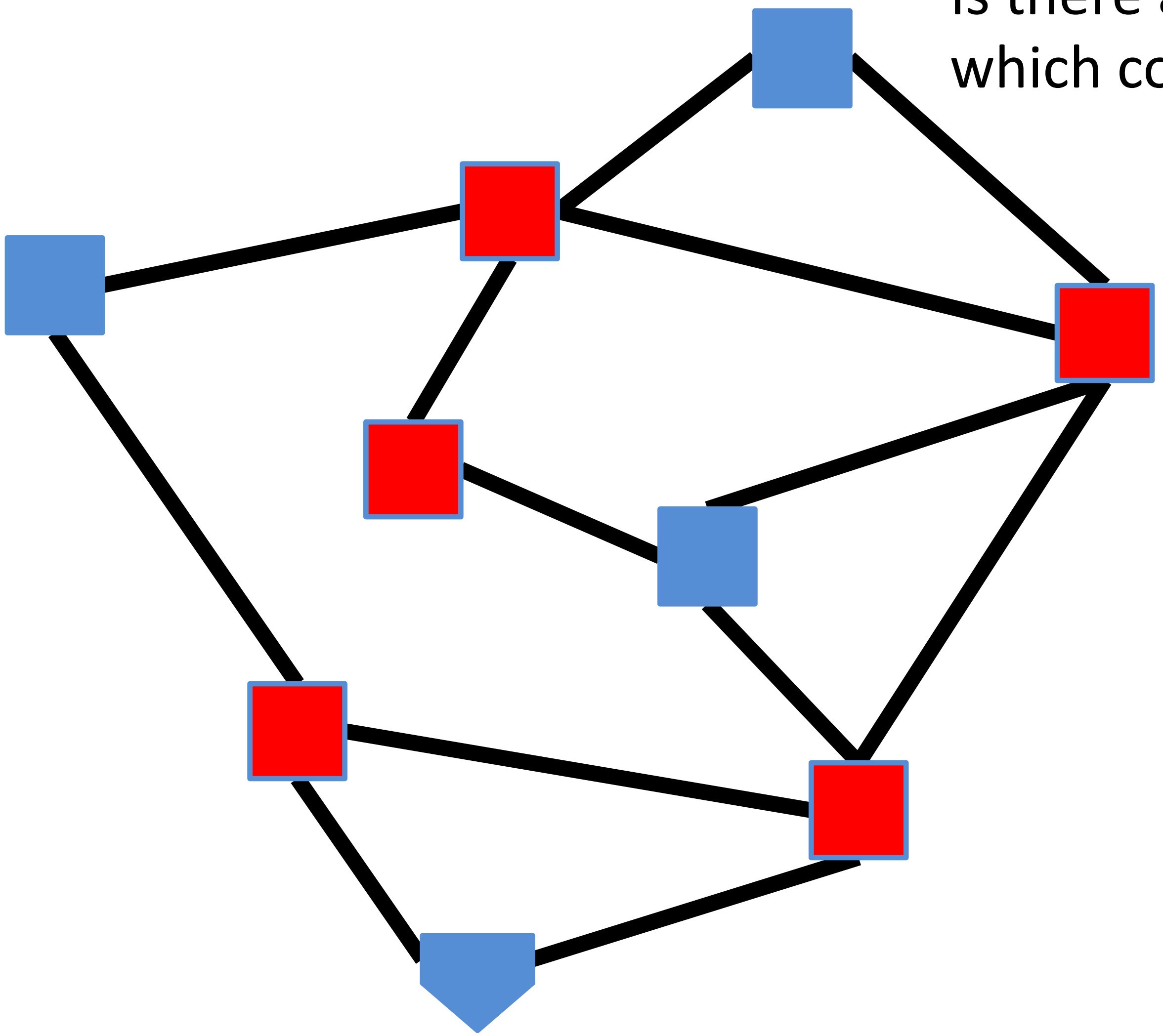
- Verification Problems:
  – Given a potential solution, is it valid?
    - Output is True/False
  – Is **this** a vertex cover of size k**?**

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

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

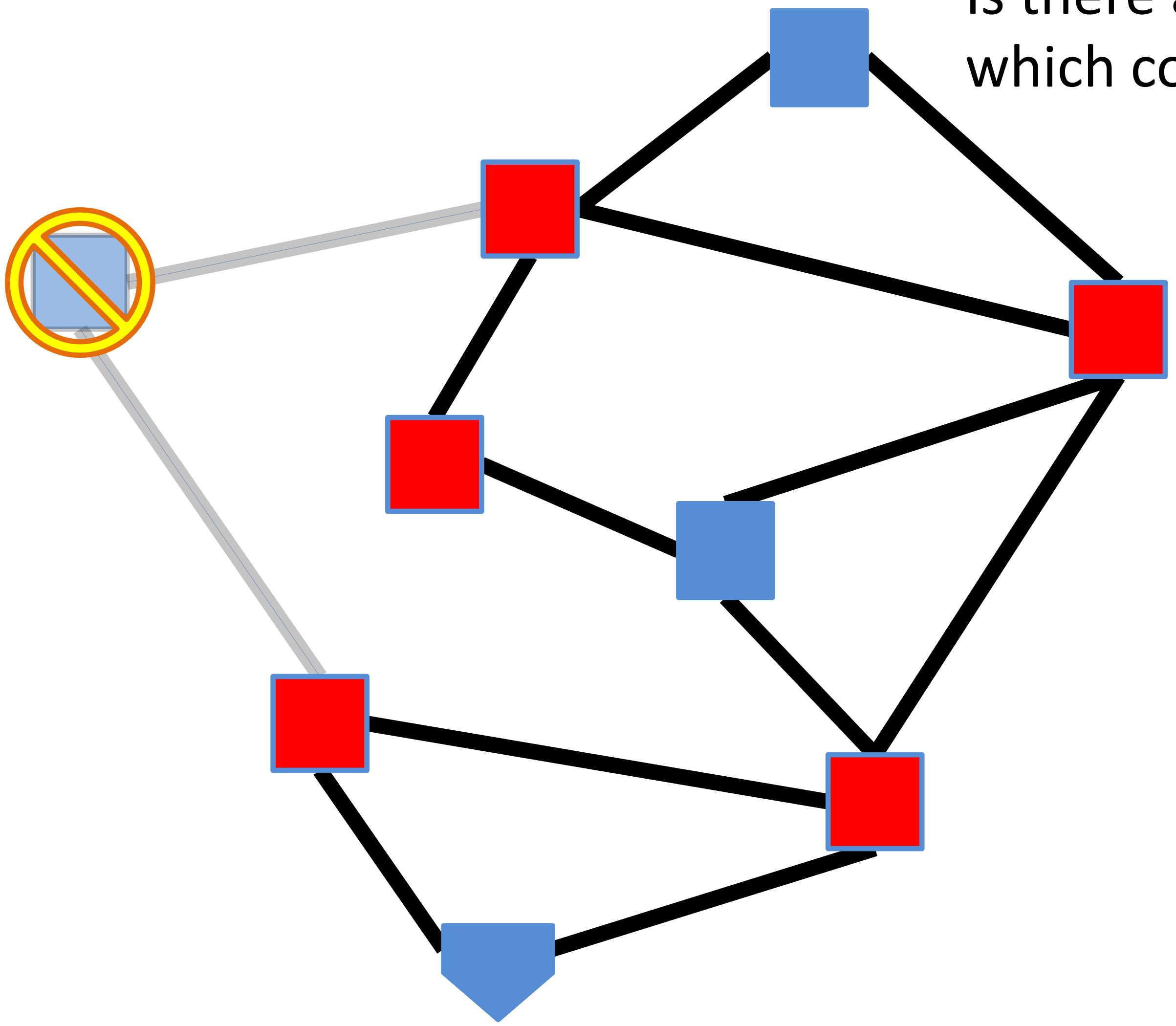
5 Vertex Cover (Decision)



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

Yes!

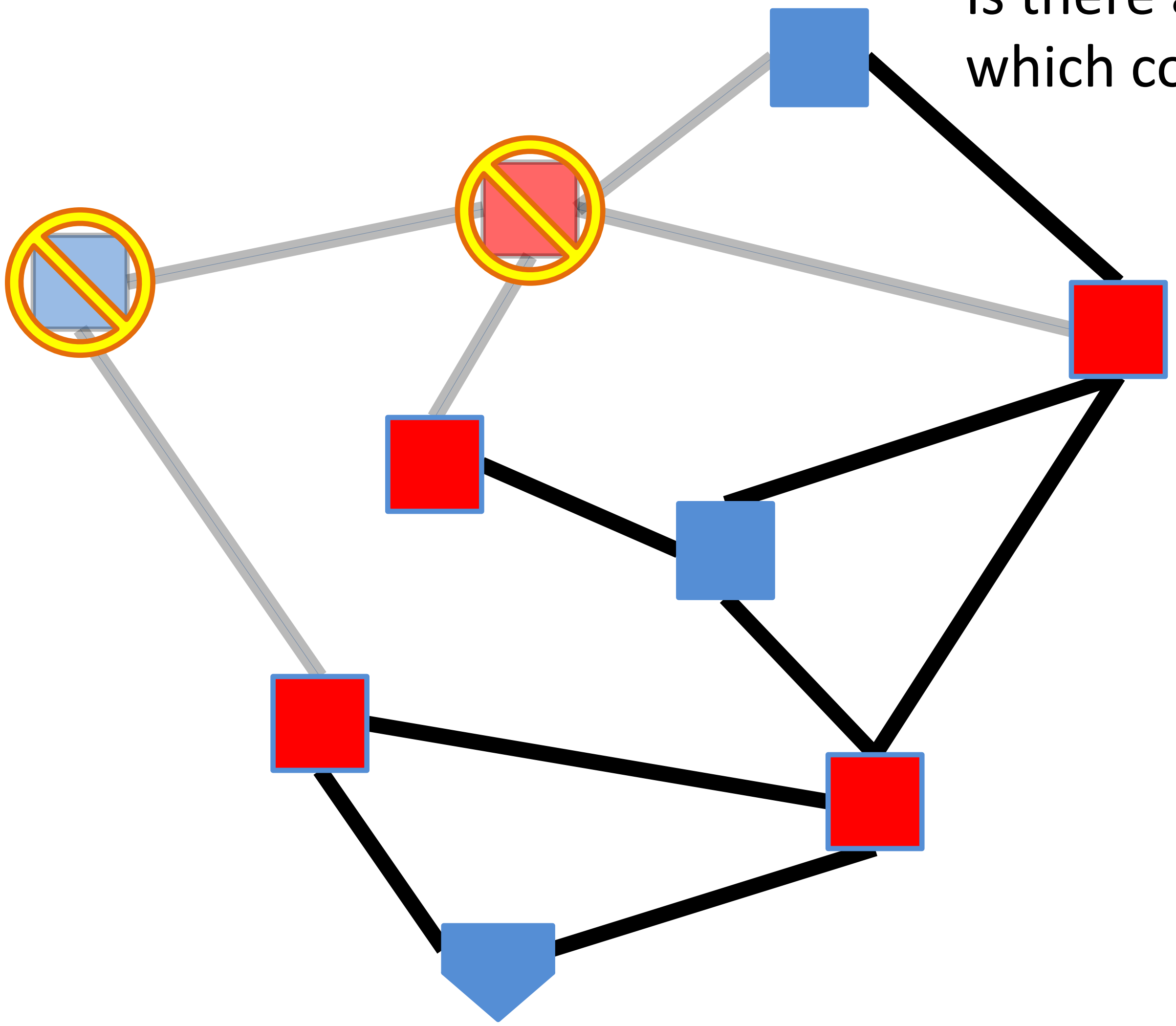
4 Vertex Cover (Decision)



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

No!

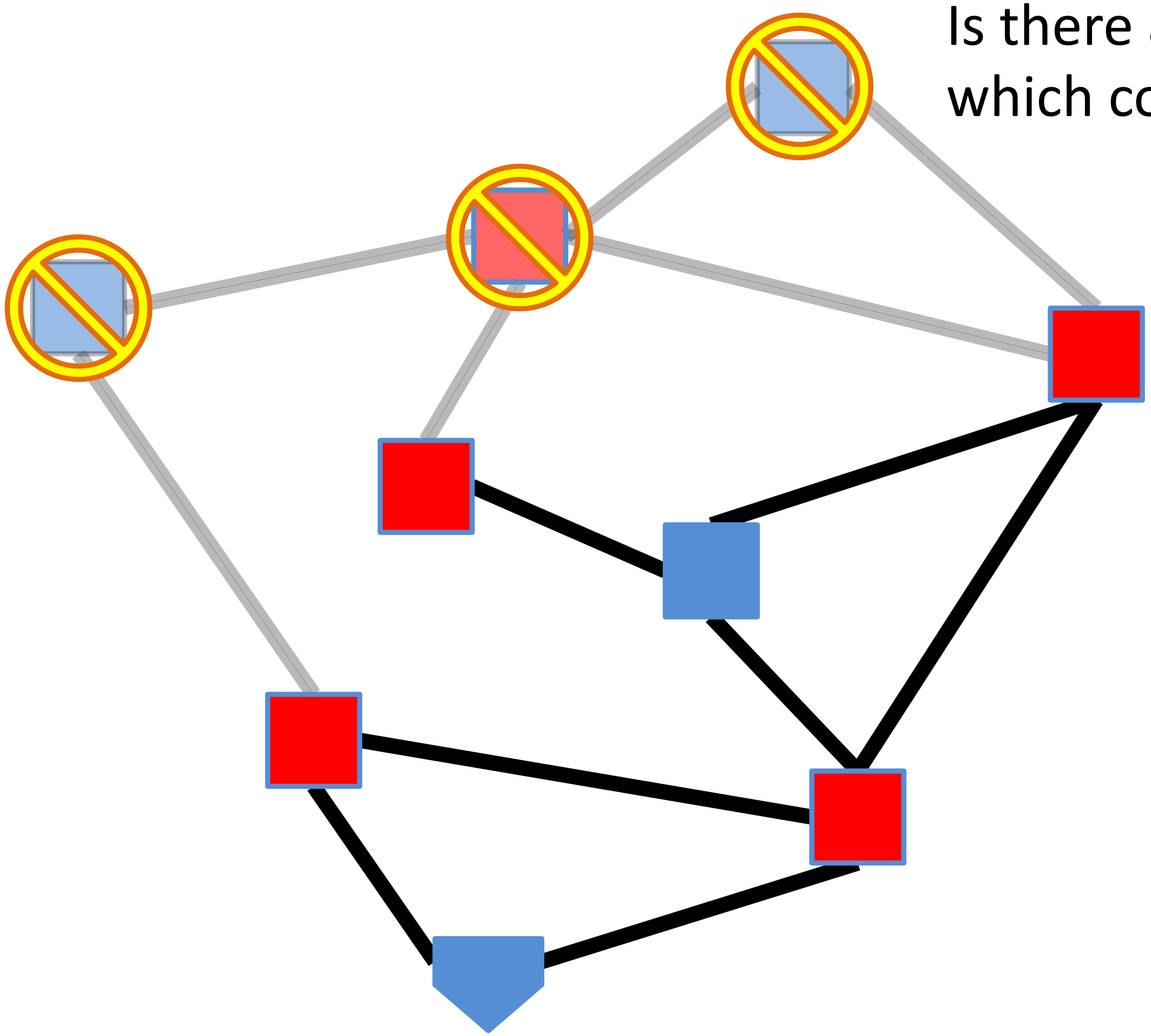
4 Vertex Cover (Decision)



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

Yes!

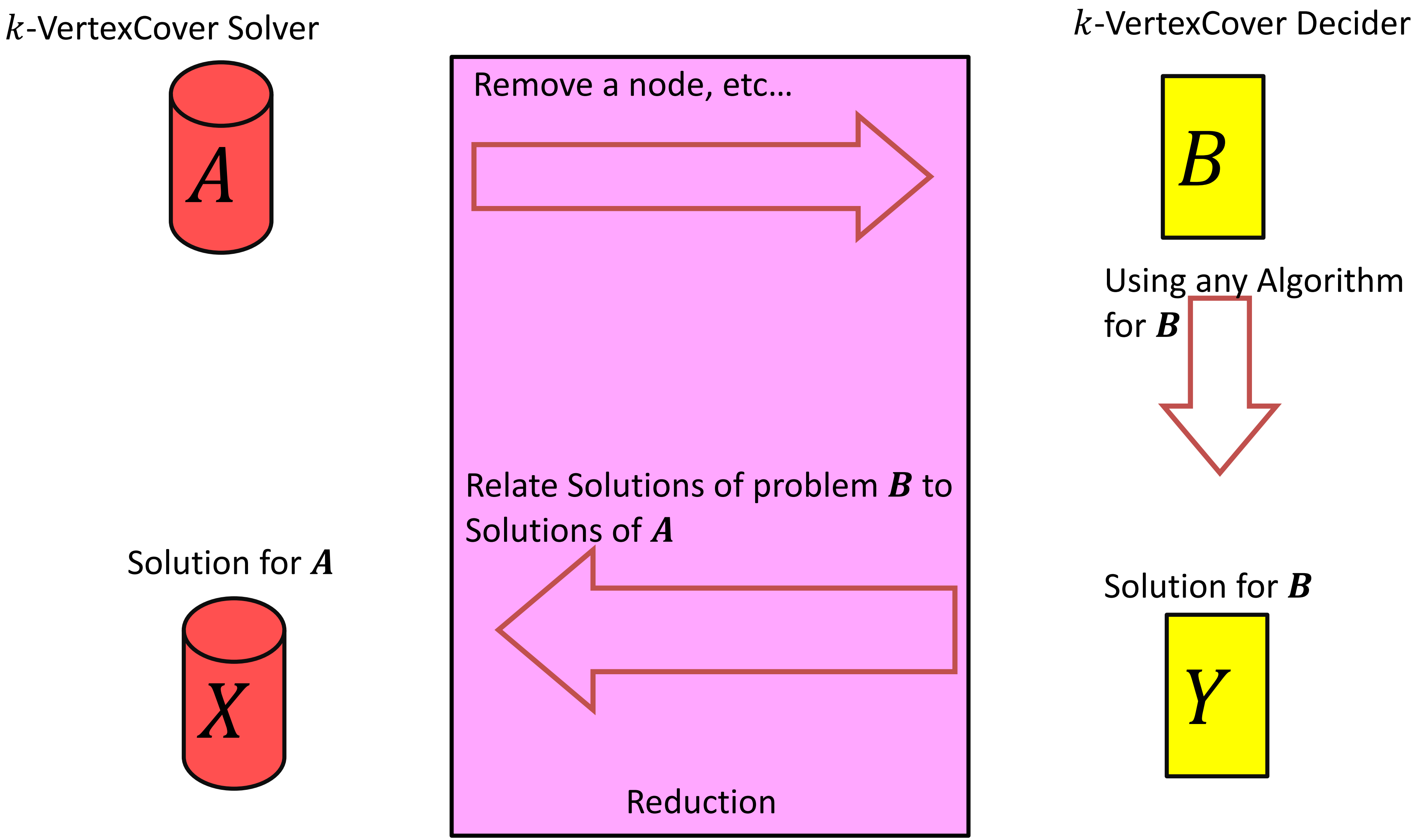
3 Vertex Cover (Decision)



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

No!

Reduction



k-VertexCover Solver
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