CS 3100
Data Structures and Algorithms 2

Lecture 22: Reductions

Co-instructors: Robbie Hott and Ray Pettit

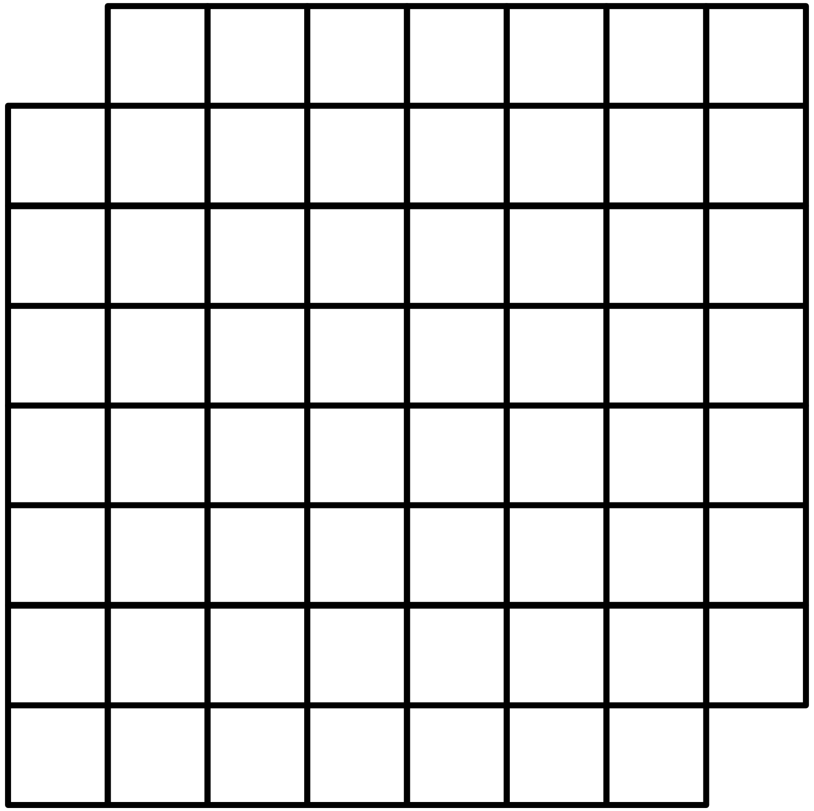
Spring 2024

Readings from CLRS 4th Ed: Network flow etc. in Chapter 24
(Reductions covered in CLRS but in a context we're not studying in CS3100)

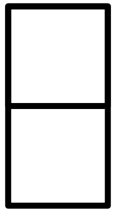
Warm-Up

Can you fill a 8×8 board with the corners missing using dominoes?

Can you tile this?



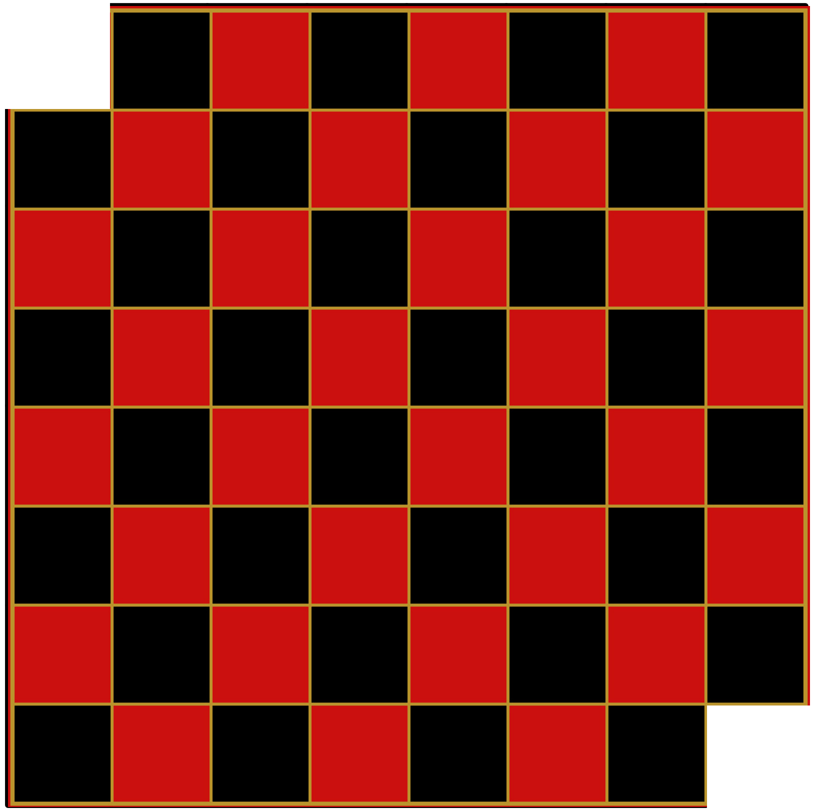
With these?



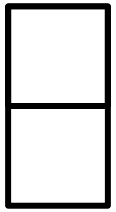
Warm-Up

Can you fill a 8×8 board with the corners missing using dominoes?

Can you tile this?



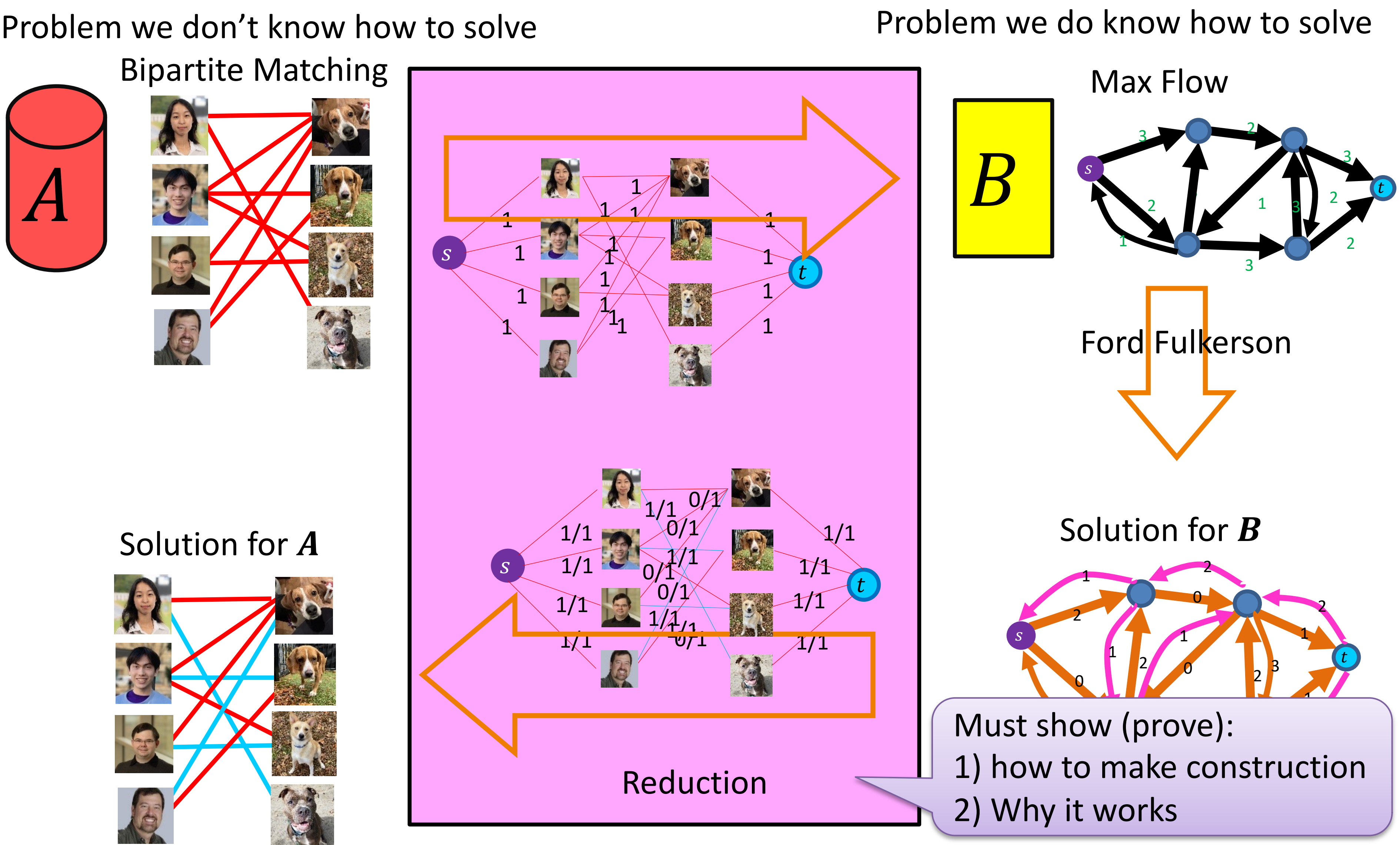
With these?



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

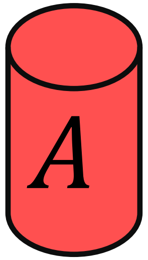
Bipartite Matching Reduction

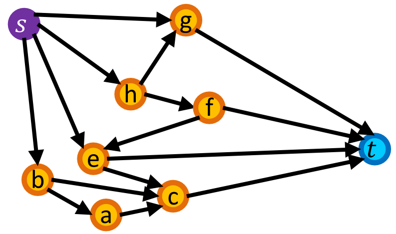


Edge Disjoint Paths Reduction

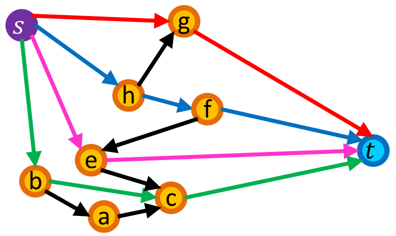
Problem we don't know how to solve

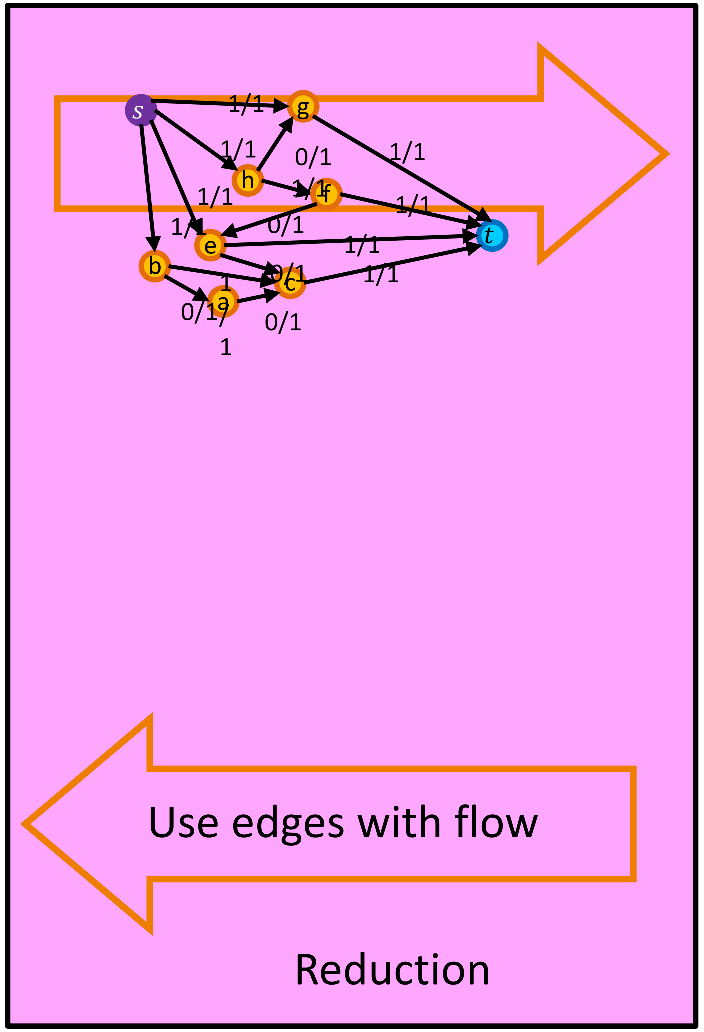
Edge Disjoint Paths





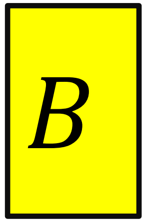
Solution for $\boldsymbol{A}$

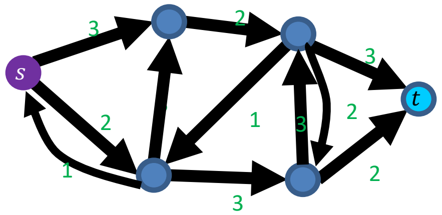




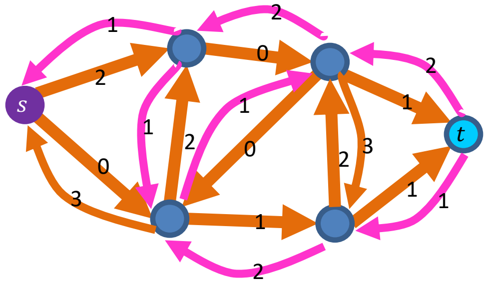
Problem we do know how to solve

Max Flow





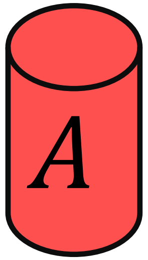
Solution for $\boldsymbol{B}$

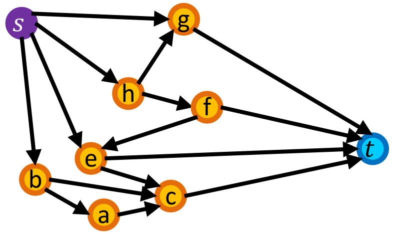


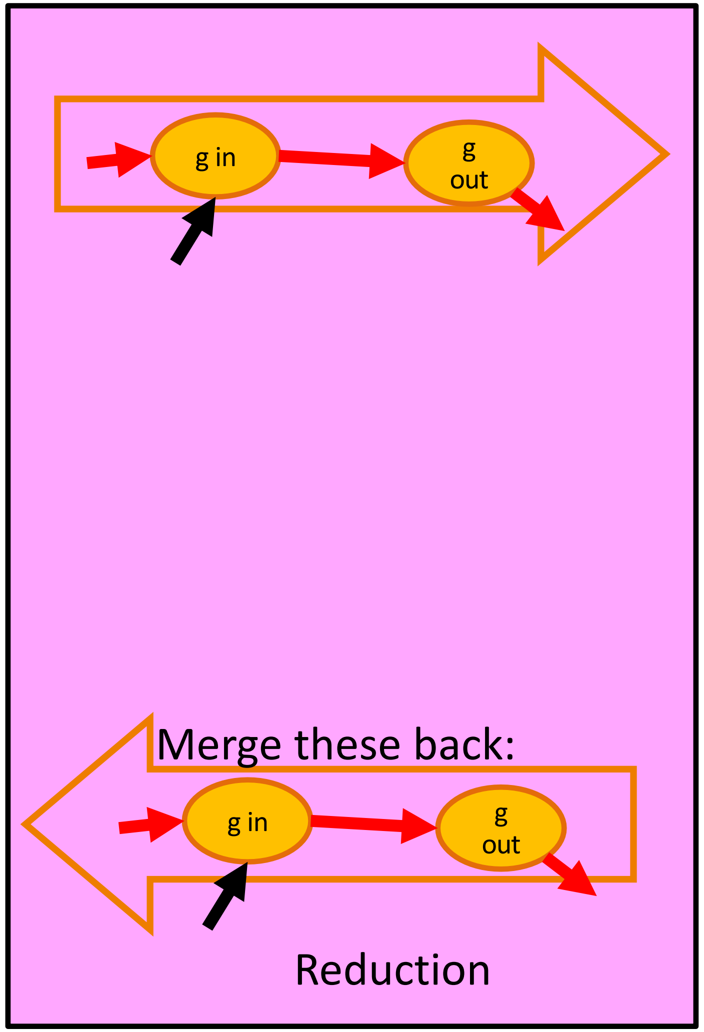
Vertex Disjoint Paths Reduction

Problem we don't know how to solve

Vertex Disjoint Paths

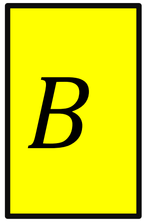


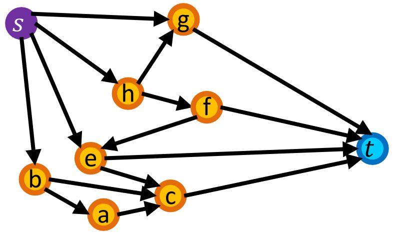




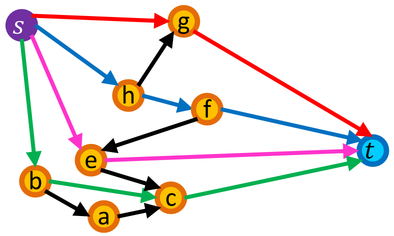
Problem we do know how to solve

Edge Disjoint Paths

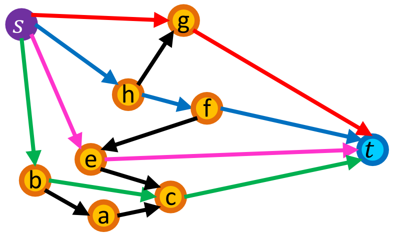




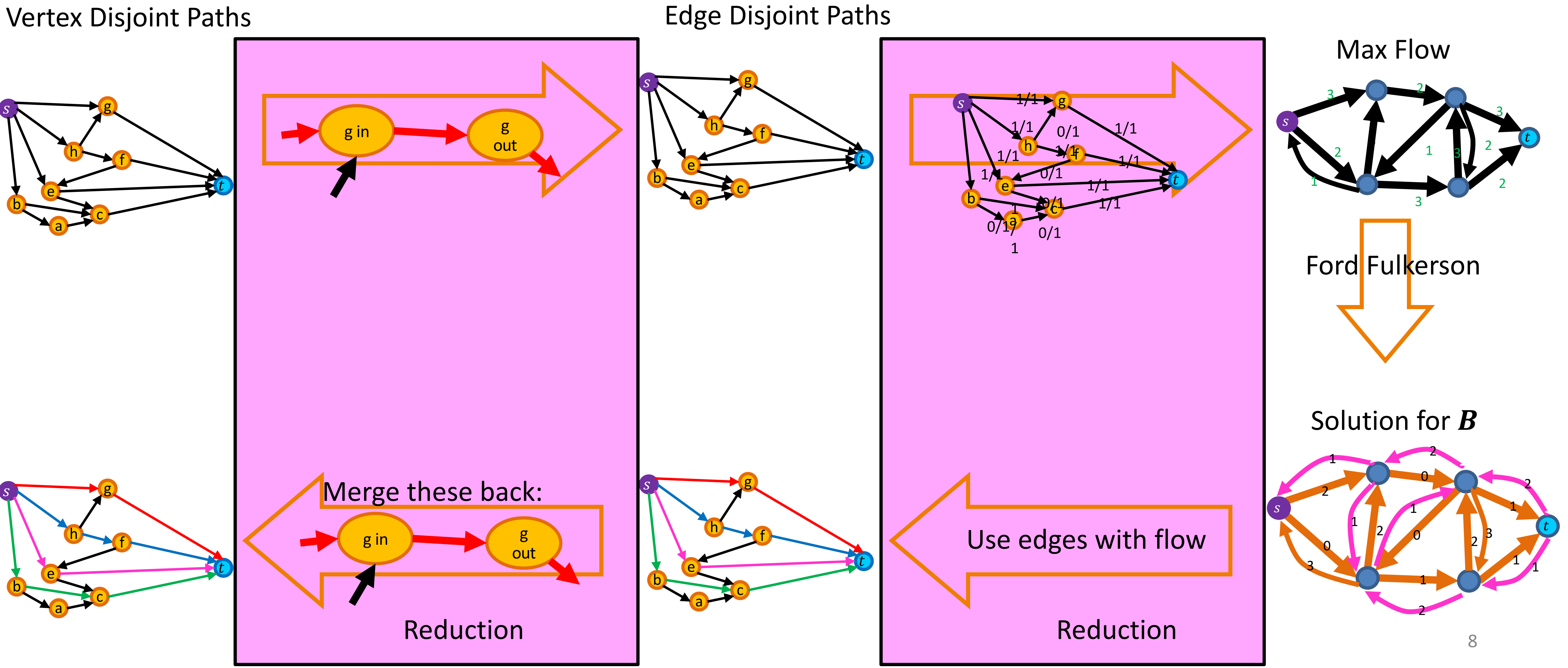
Solution for $\boldsymbol{A}$



Solution for $\boldsymbol{B}$



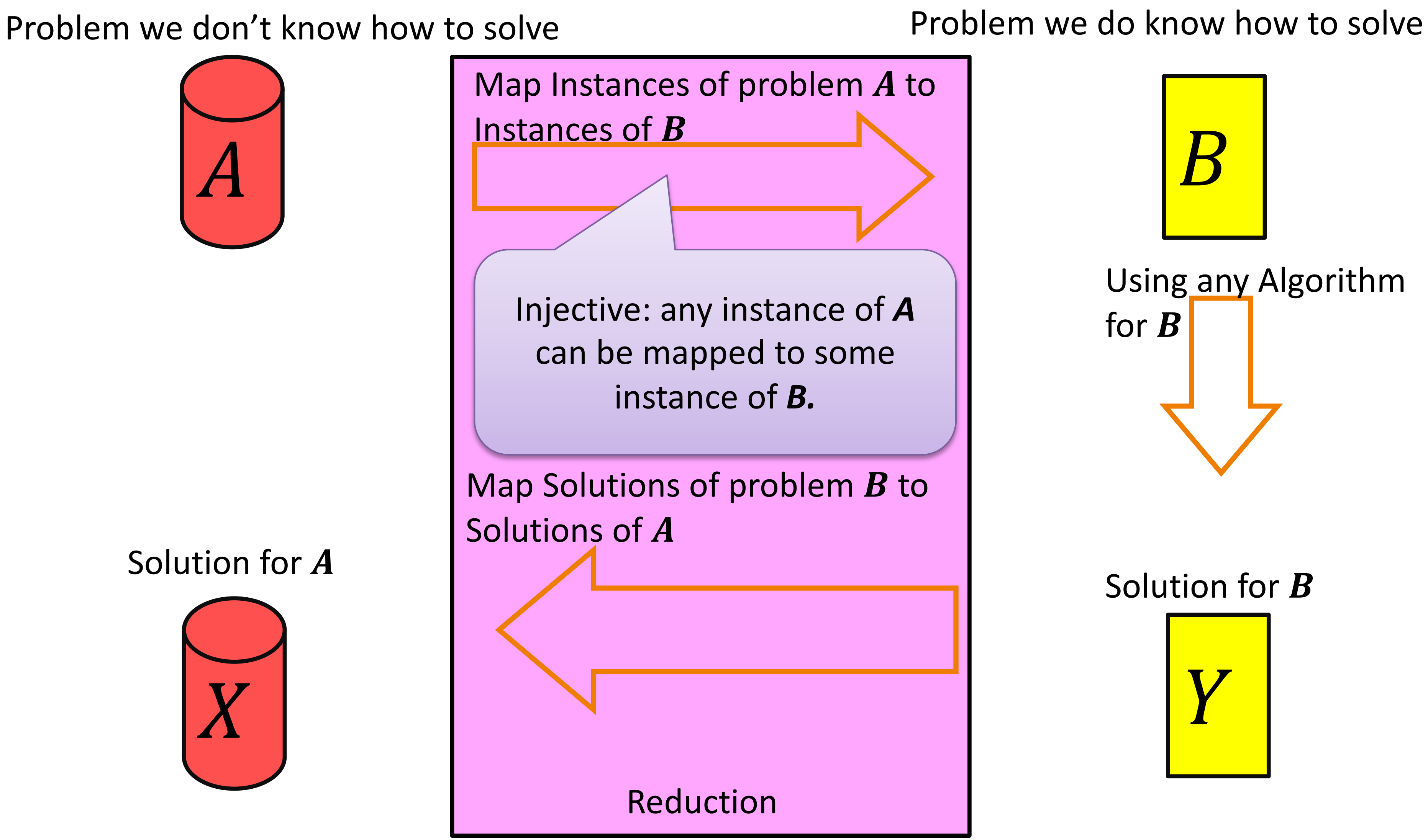
Vertex Disjoint Paths Big Picture



Reductions for New Algorithms

- Create an algorithm for a new problem by using one you already know!
- More algorithms = More opportunities!
- The problem you reduced to could itself be solved using a reduction!

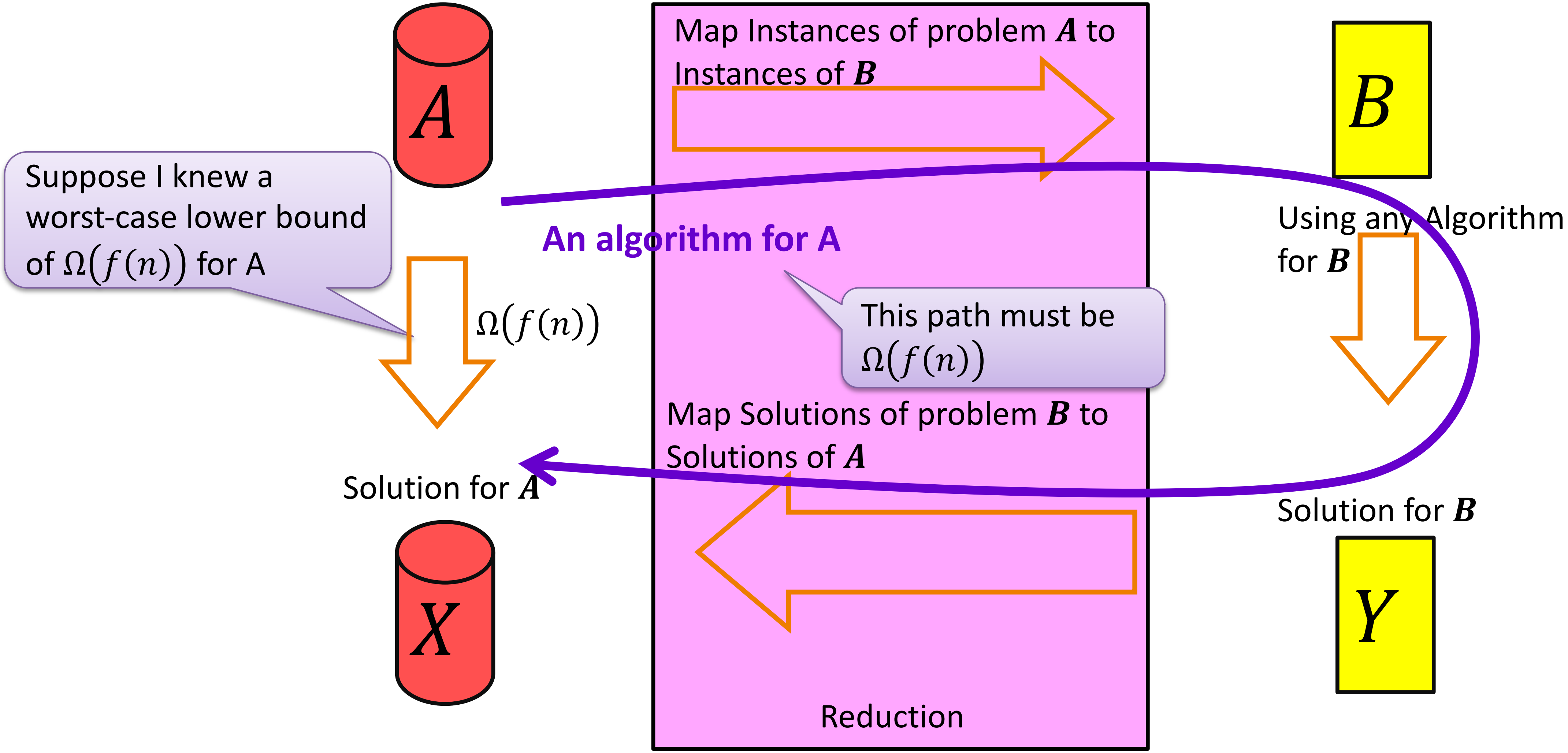
In General: Reduction



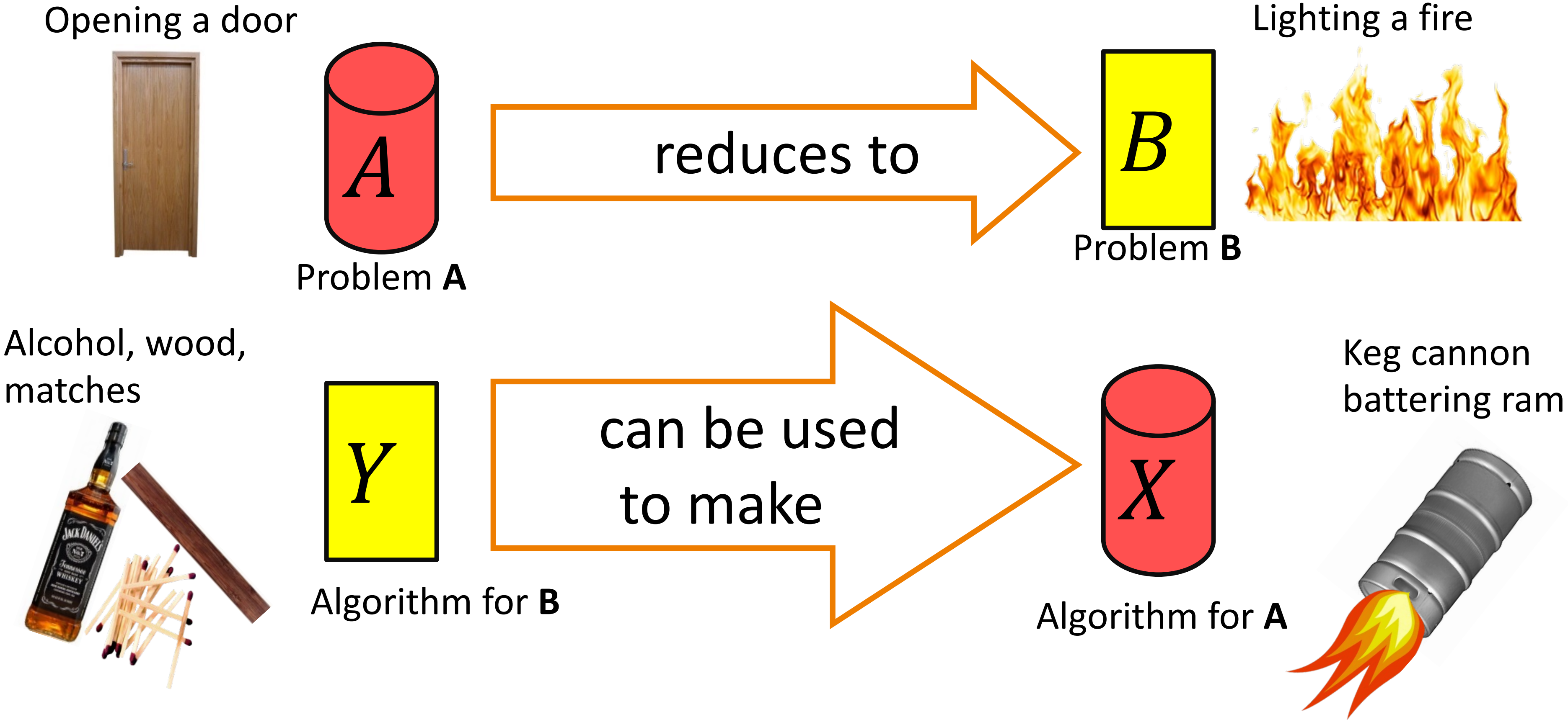
Worst Case Lower Bound

- Definition:
  - A **worst case lower bound** on a problem is an *asymptotic lower bound on the worst case running time* of any algorithm which solves it
  - If $f(n)$ is a worst case lower bound for problem A, then the worst-case running time of any algorithm which solves A must be $\Omega\big(f(n)\big)$
  - i.e. for sufficiently large values of n, for every algorithm which solves A, there is at least one input of size n which causes the algorithm to do $\Omega\big(f(n)\big)$ steps.
- Examples:
  - n is a worst-case lower bound on finding the minimum in a list
  - n^2 is a worst-case lower bound on matrix multiplication

Another use of Reductions



Worst-case lower-bound Proofs



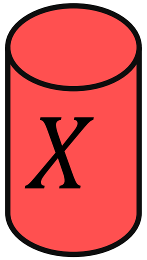
A is not a harder problem than B

$$A \le B$$

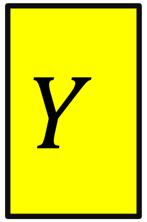
The name "reduces" is confusing: it is in the *opposite* direction of the making

Proof of Lower Bound by Reduction

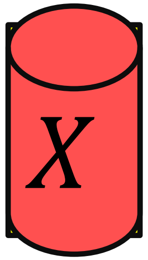
<u>To Show: Y is slow</u>



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



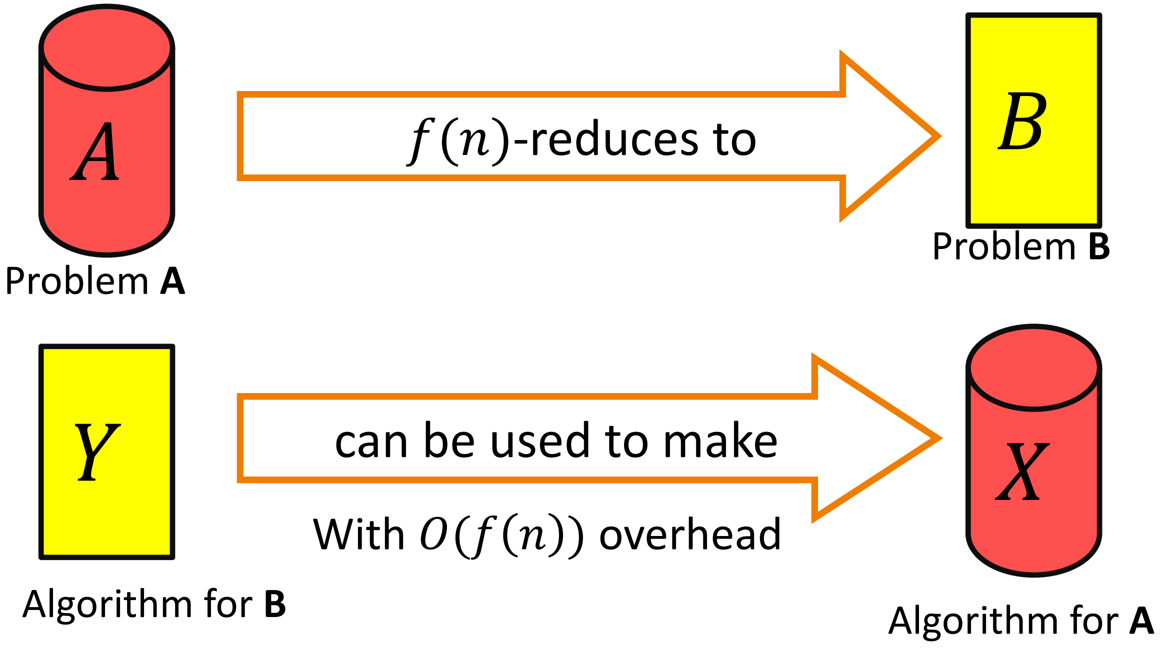
2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)



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

$$A \le B$$

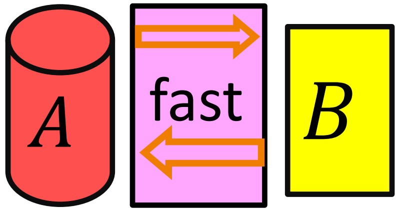
If A requires time $\Omega(f(n))$ time then B also requires $\Omega(f(n))$ time

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

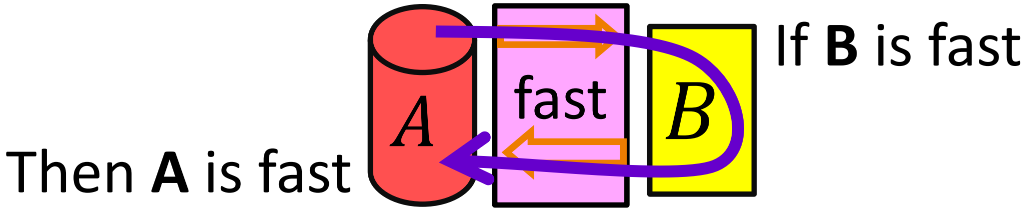
Or we could have solved A faster using B's solver!

Two Ways to use Reductions

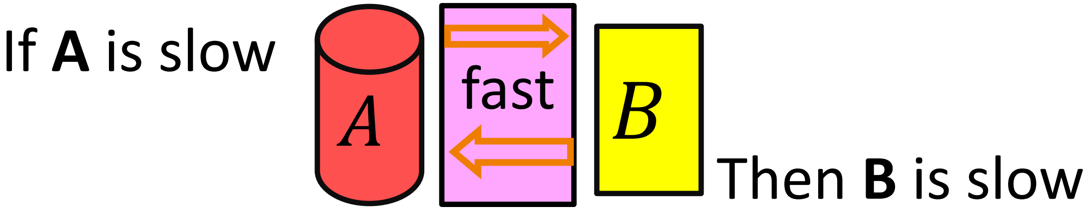
Suppose we have a "fast" reduction from **A** to **B**



1. A "fast" algorithm for **B** gives a fast algorithm for **A**



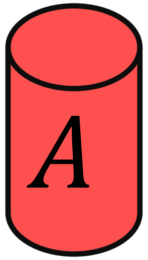
2. If we have a worst-case lower bound for **A**, we also have one for **B**



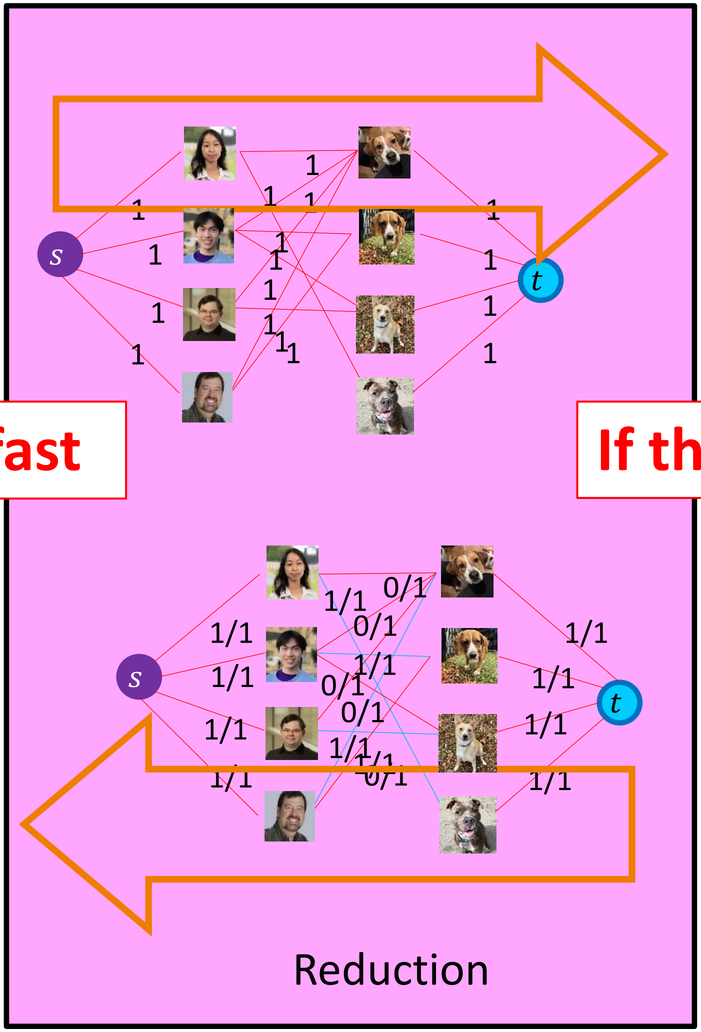
Bipartite Matching Reduction

Problem we don't know how to solve

Bipartite Matching

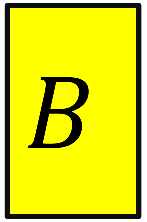


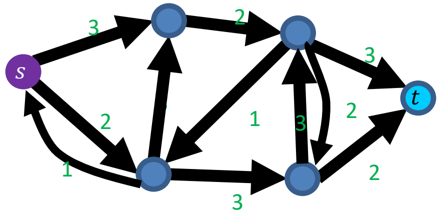
Then this is fast



Problem we do know how to solve

Max Flow

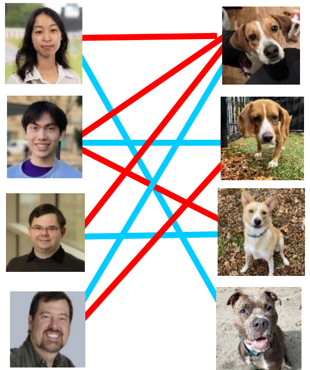




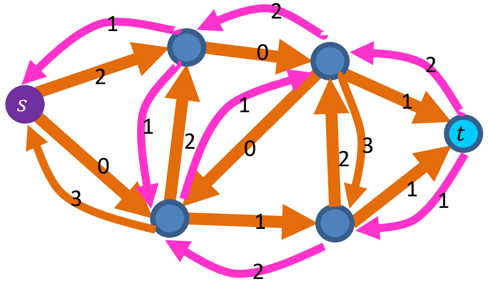
Ford Fulkerson

If this is fast

Solution for $\boldsymbol{A}$



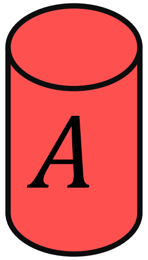
Solution for $\boldsymbol{B}$



Bipartite Matching Reduction

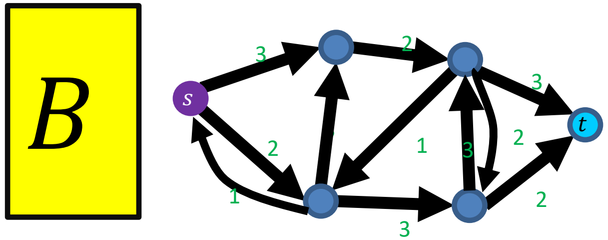
Problem we don't know how to solve

Bipartite Matching



Problem we do know how to solve

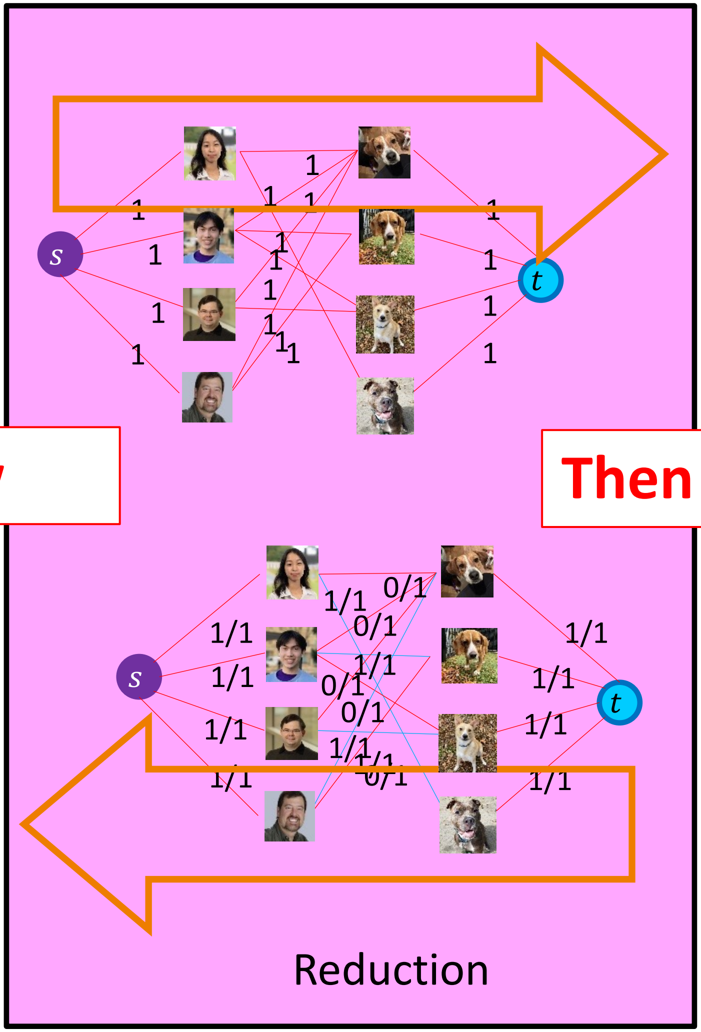
Max Flow



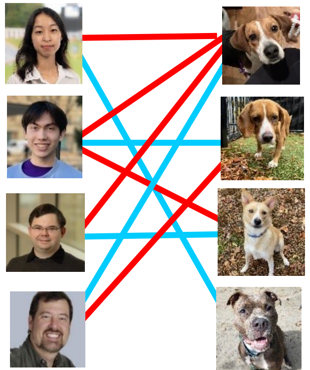
Ford Fulkerson

If this is slow

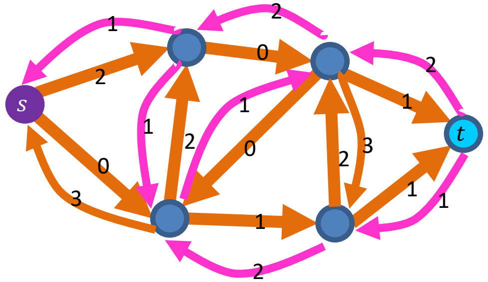
Then this is slow



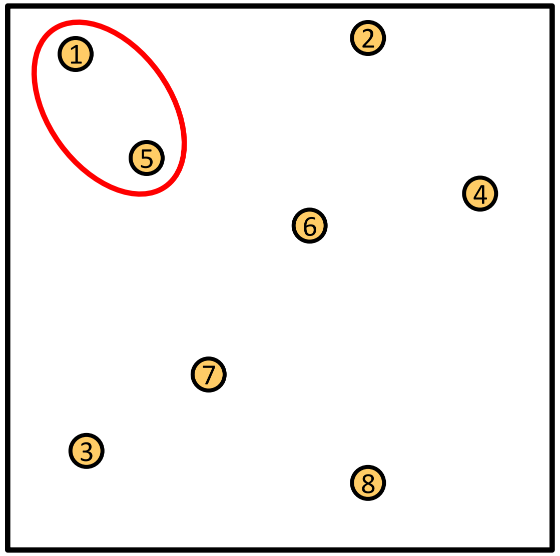
Solution for $\boldsymbol{A}$



Solution for $\boldsymbol{B}$



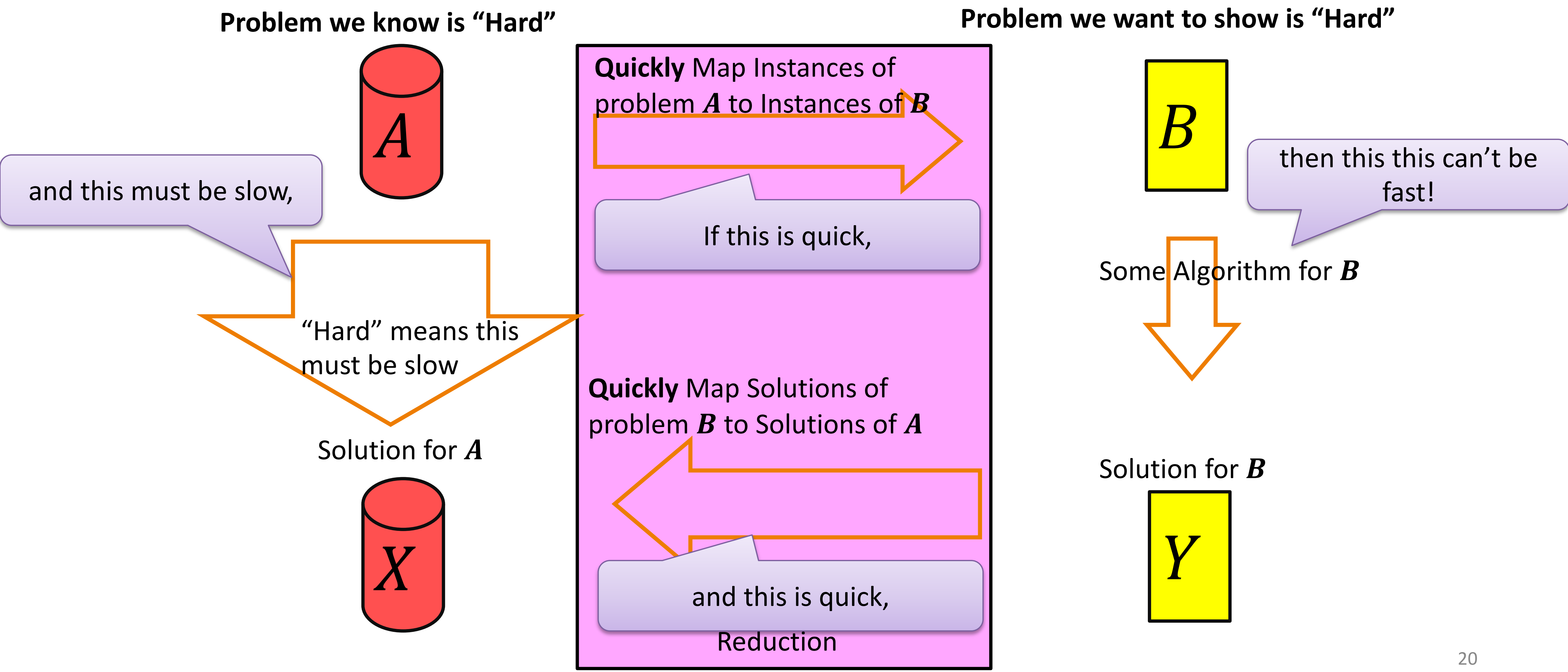
Worst-case Lower-Bound Using Reductions



- Closest Pair of points
  - D&C algorithm: $\Theta(n \log n)$
  - Can we do better?

- Idea: Show that doing closest pair in $o(n \log n)$ enables an impossibly fast algorithm for another problem

Reductions for Lower-Bounds



Reductions for Lower-Bound on CPP

Problem we know is $\Omega(n \log n)$

Then this can be done in $o(n \log n)$

A

$\boldsymbol{\Omega(n \log n)}$

Solution for $\boldsymbol{A}$

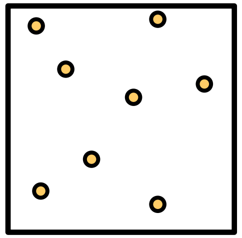
X

Map Instances of problem $\boldsymbol{A}$ to Instances of CPP in $\boldsymbol{o(n \log n)}$

Map Solutions of CPP to Solutions of $\boldsymbol{A}$ in $\boldsymbol{o(n \log n)}$

Reduction

Problem we want to show is $\Omega(n \log n)$

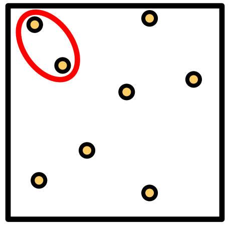


Closest Pair of Points

Some Algorithm for CPP

If this could be done in $o(n \log n)$

Solution for $\boldsymbol{B}$

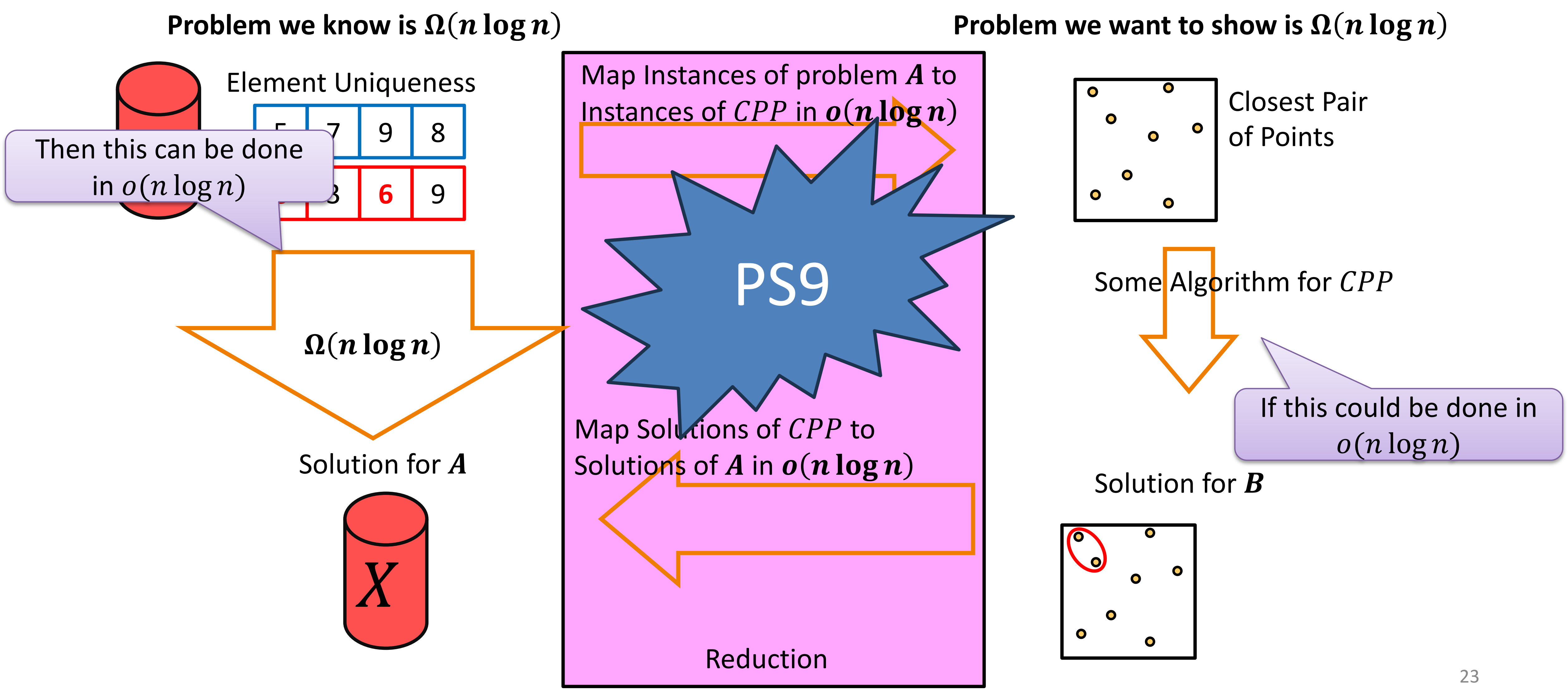


A "Hard" Problem: Element Uniqueness

113	901	555	512	245	800	018	121	**True**
103	801	401	**323**	255	**323**	999	101	**False**

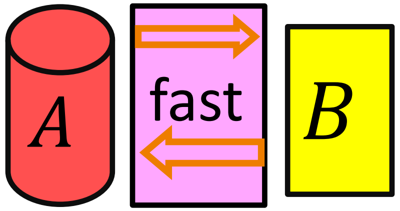
- Input:
  - A list of integers
- Output:
  - **True** if all values are unique, **False** otherwise
- Can this be solved in $O(n \log n)$ time?
  - Yes! Sort, then check if any adjacent elements match
- Can this be solved in $o(n \log n)$ time?
  - No! (we're going to skip this Proof)

Reductions for Lower-Bound on CPP



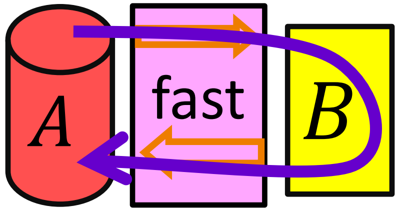
Two Ways to use Reductions

Suppose we have a "fast" reduction from **A** to **B**



1. A "fast" algorithm for **B** gives a fast algorithm for **A**

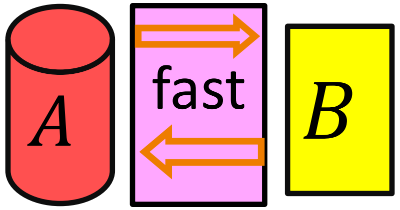
Then **A** is fast



If **B** is fast

2. If we have a worst-case lower bound for **A**, we also have one for **B**

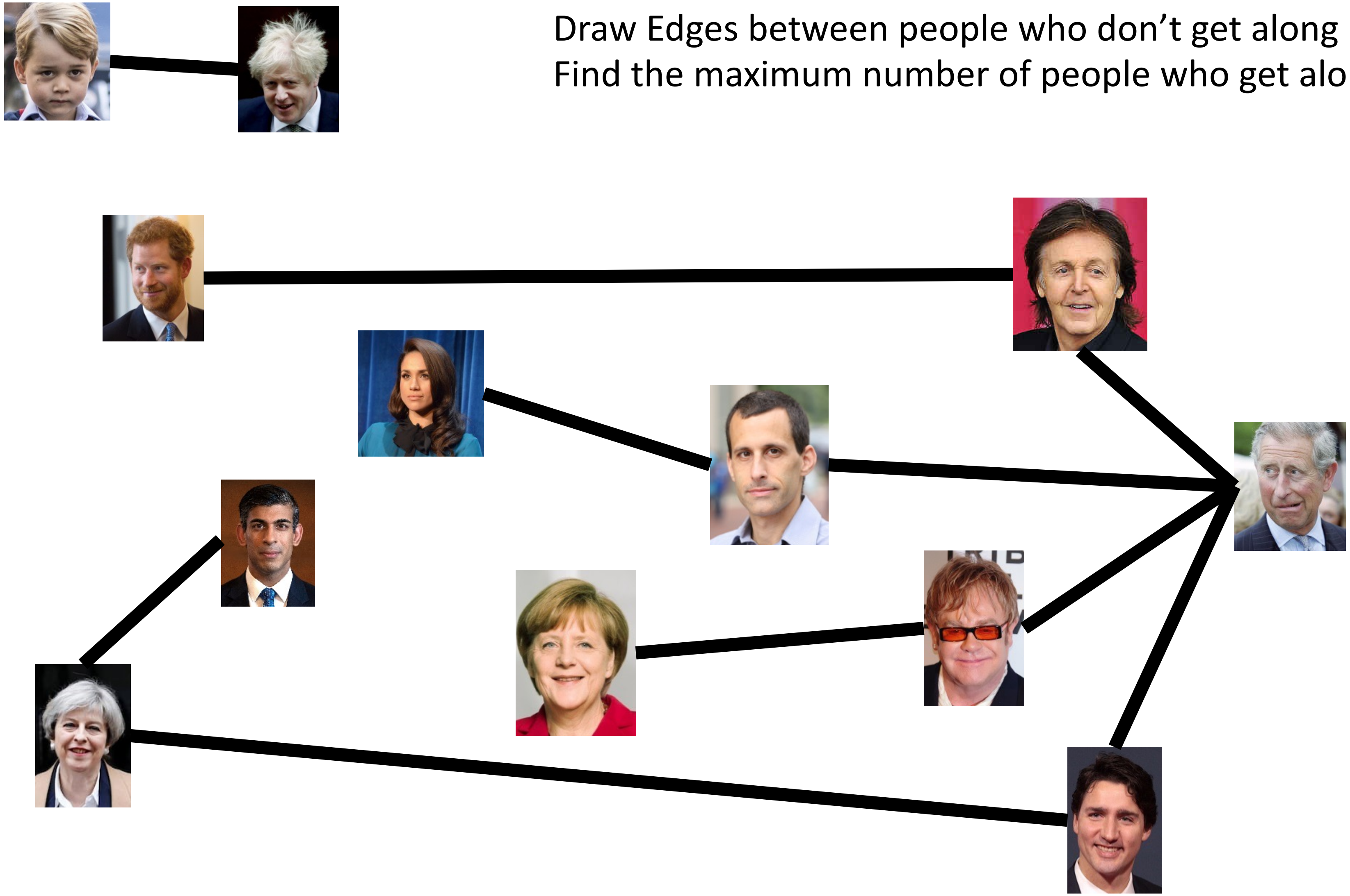
If **A** is slow
by proof



Then **B** is slow

Party Problem

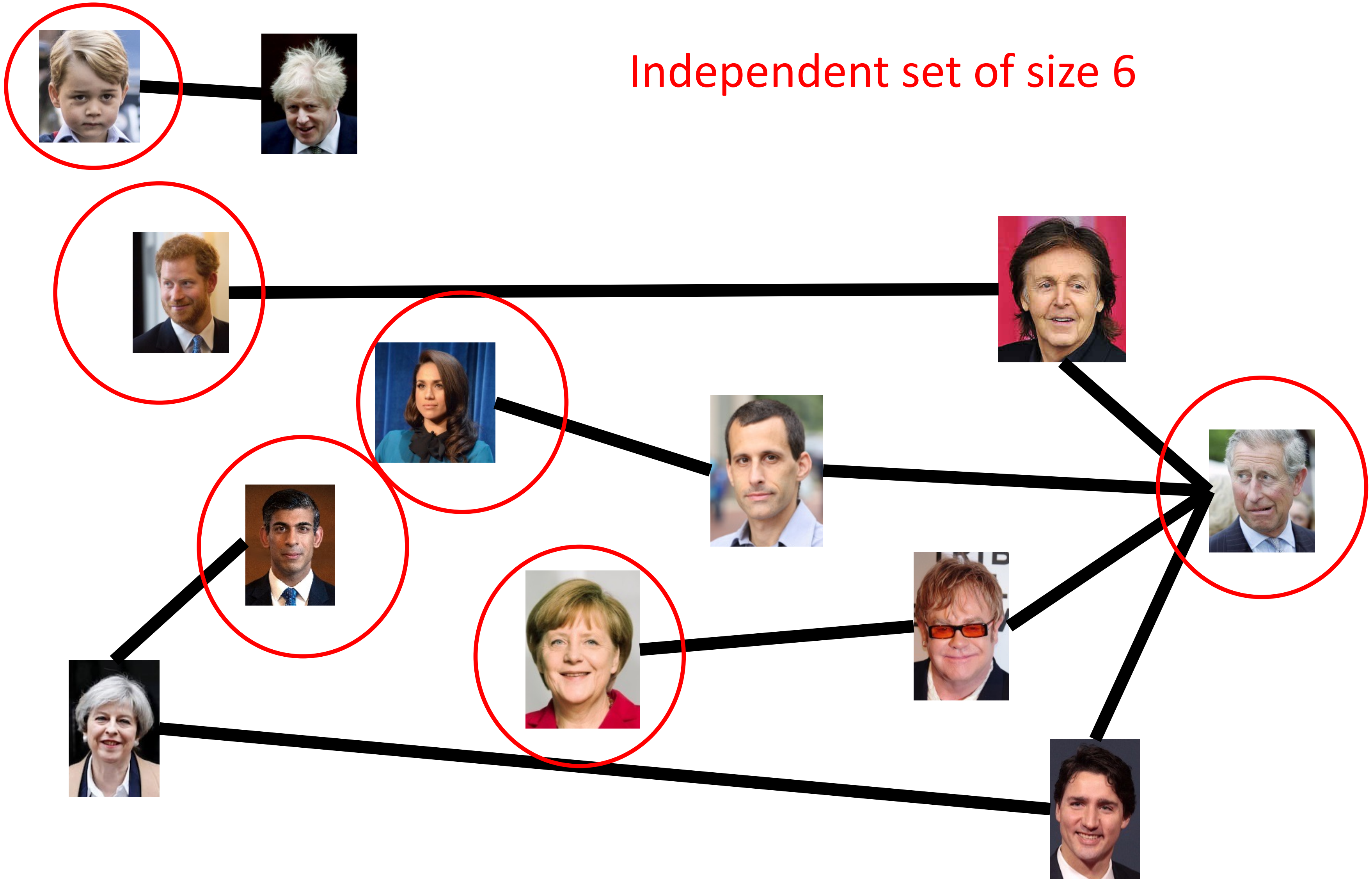
Draw Edges between people who don't get along
Find the maximum number of people who get along



Maximum Independent Set

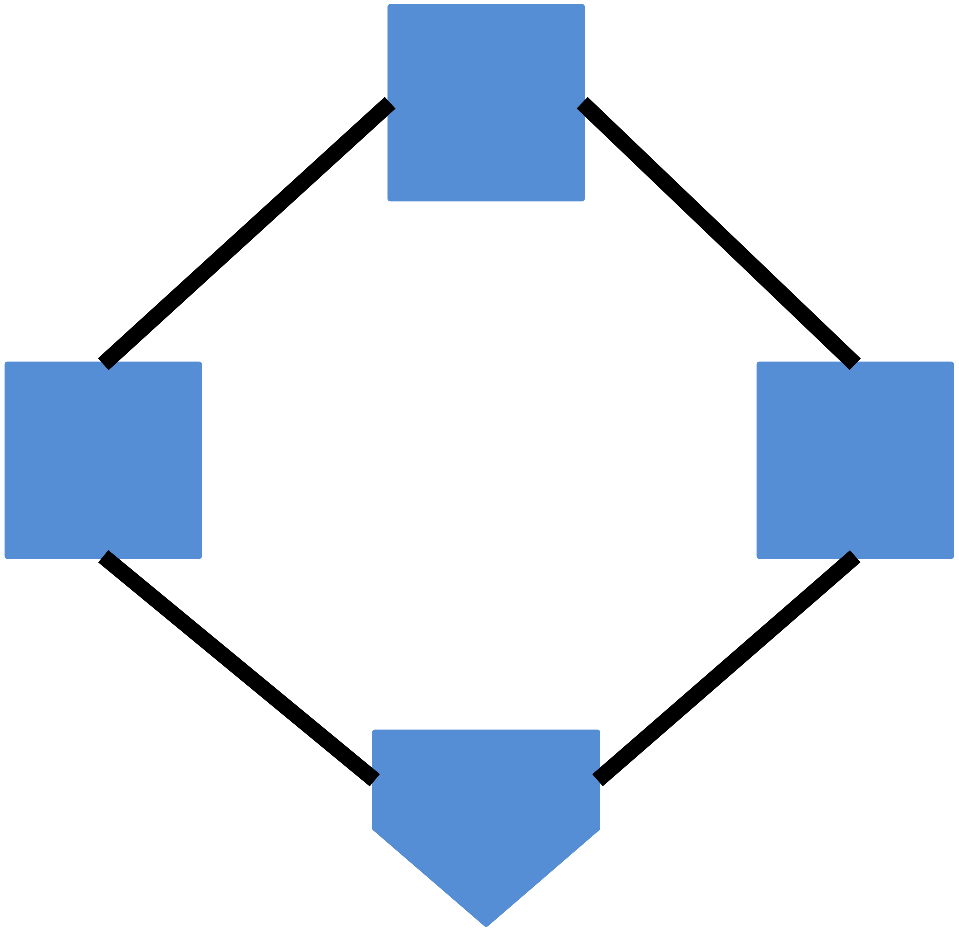
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

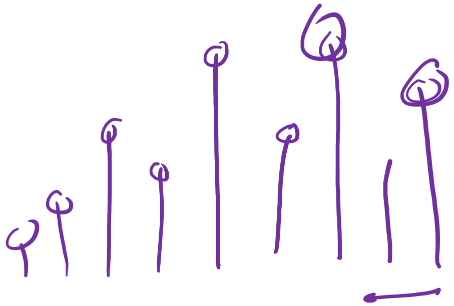
Example



Independent set of size 6

Generalized Baseball



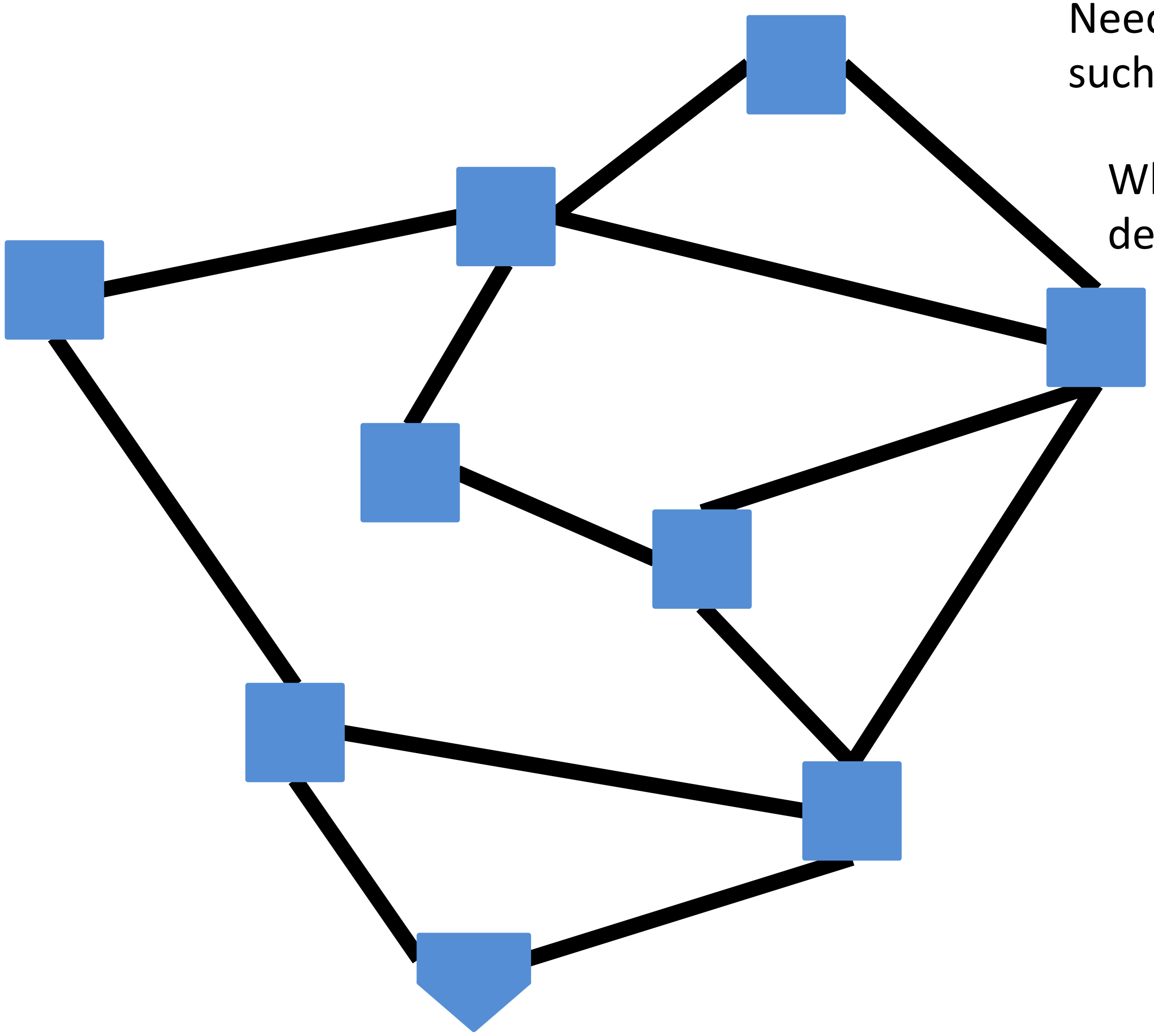


$n \in O(n^2)$

$n^2 \in \Theta(n^2)$

$n^3 \in \Omega(n^2)$

Generalized Baseball



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

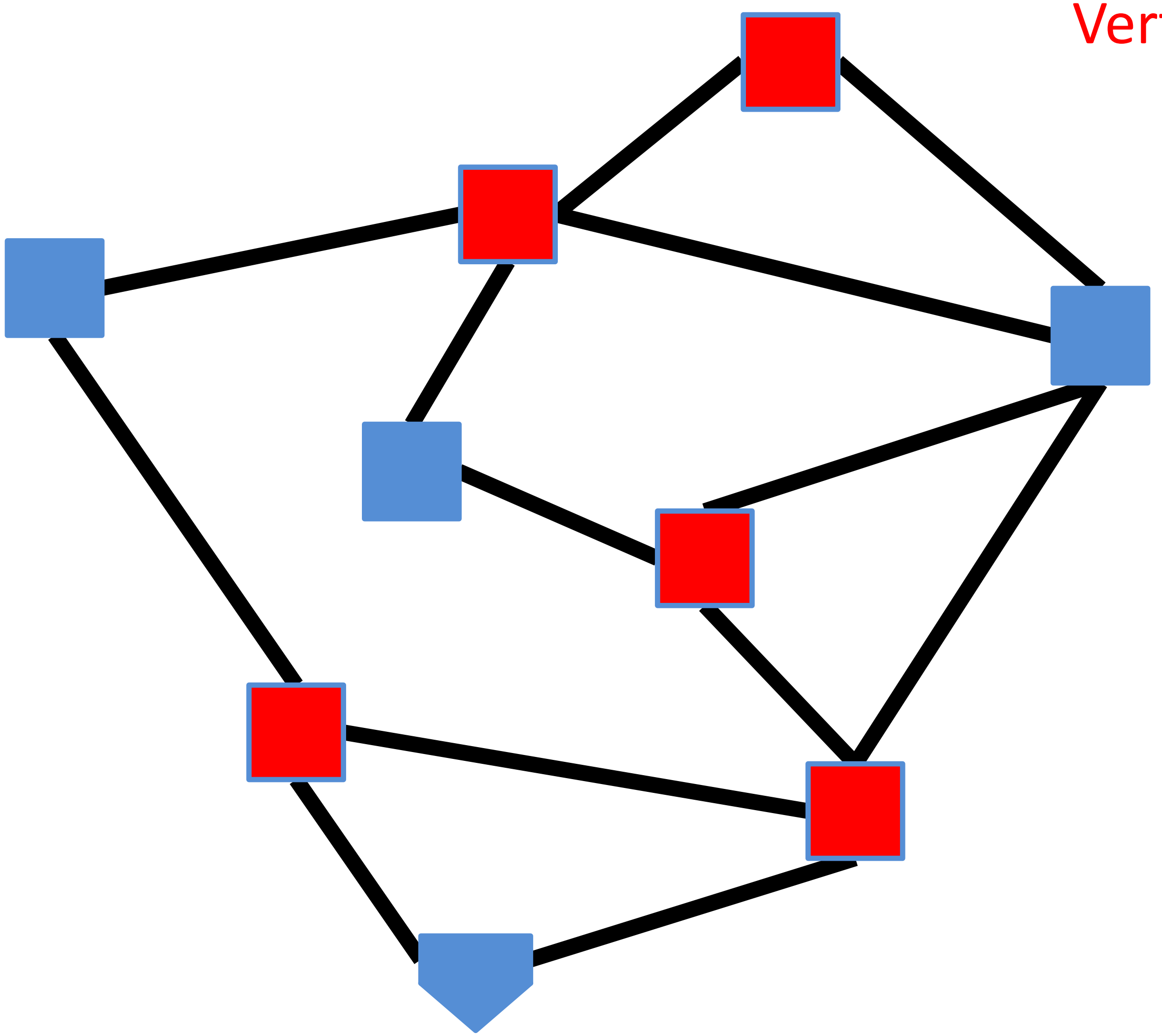
What's the fewest number of defenders needed?

Minimum Vertex Cover

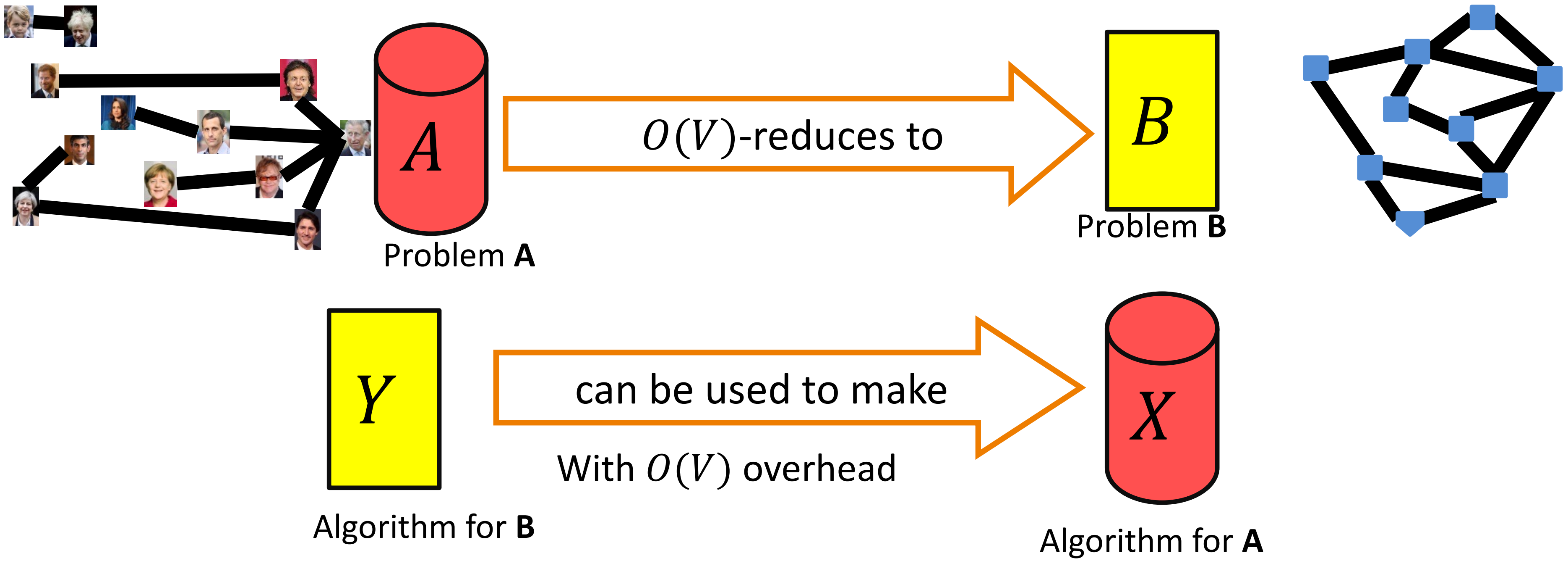
- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

Example

Vertex cover of size 5



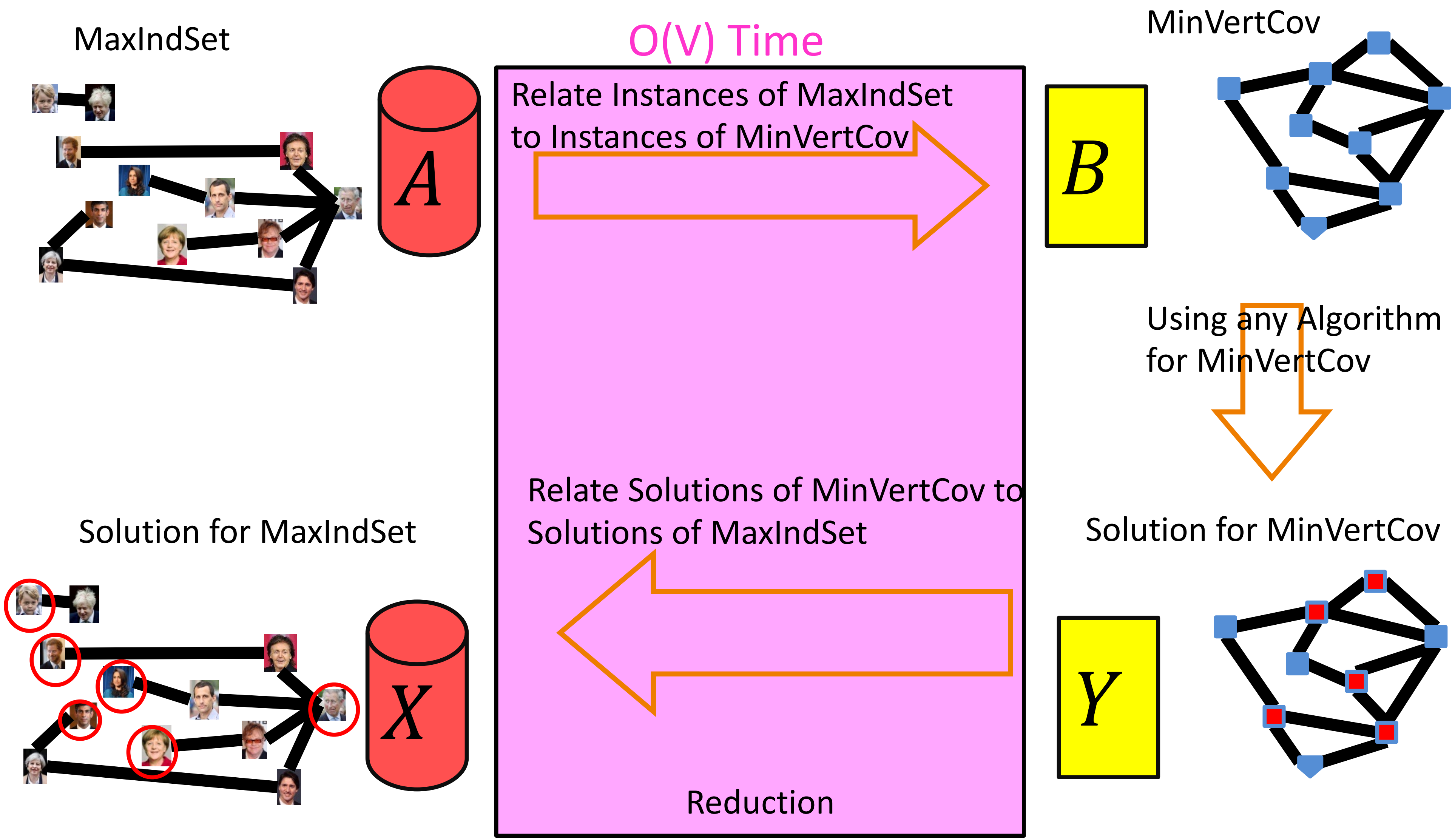
MaxIndSet$\leq_V$MinVertCov



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

$$A \leq_V B$$

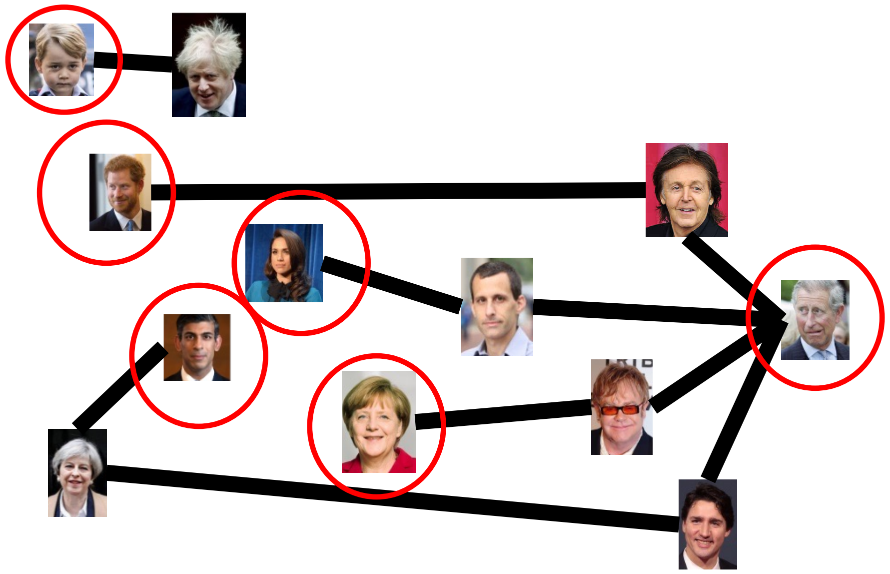
We need to build this Reduction



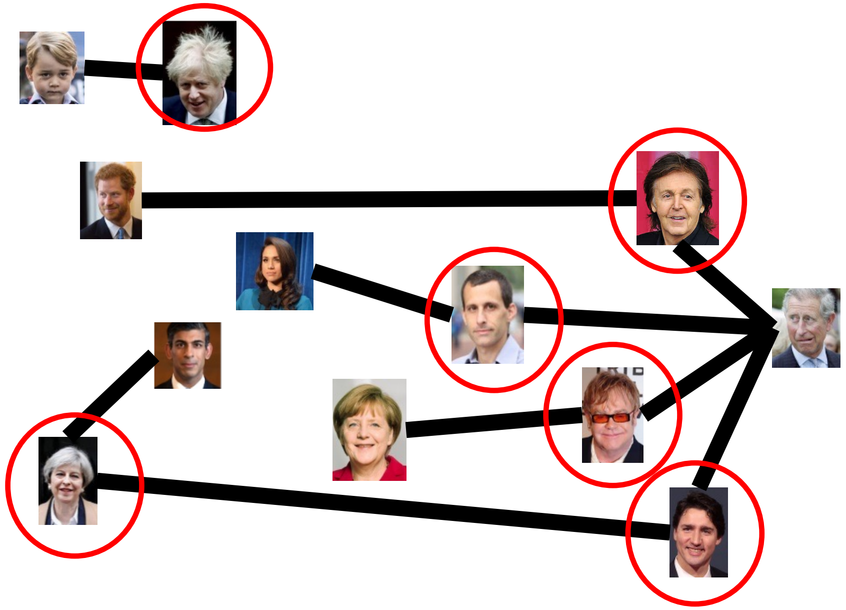
Reduction Idea

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

Independent Set



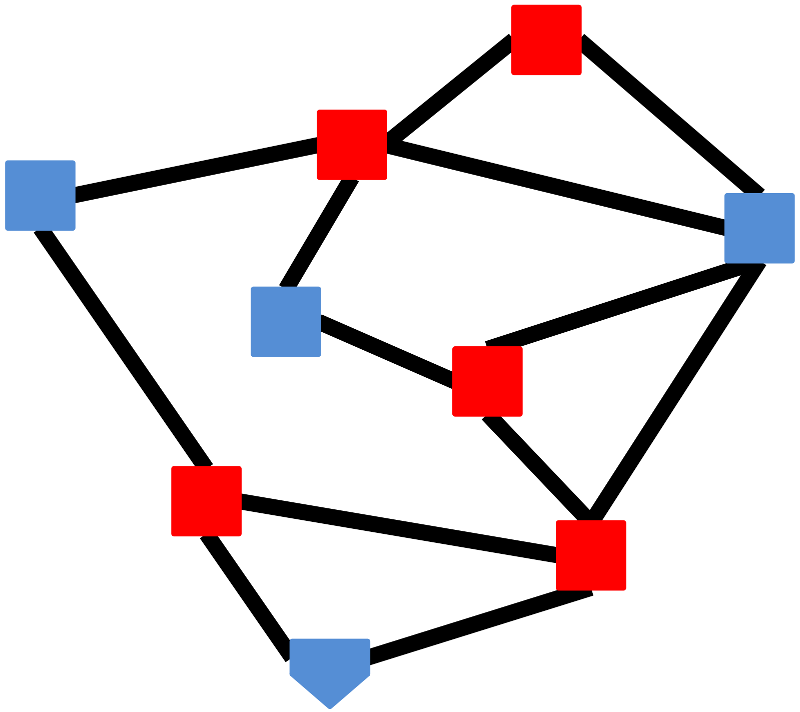
Vertex Cover



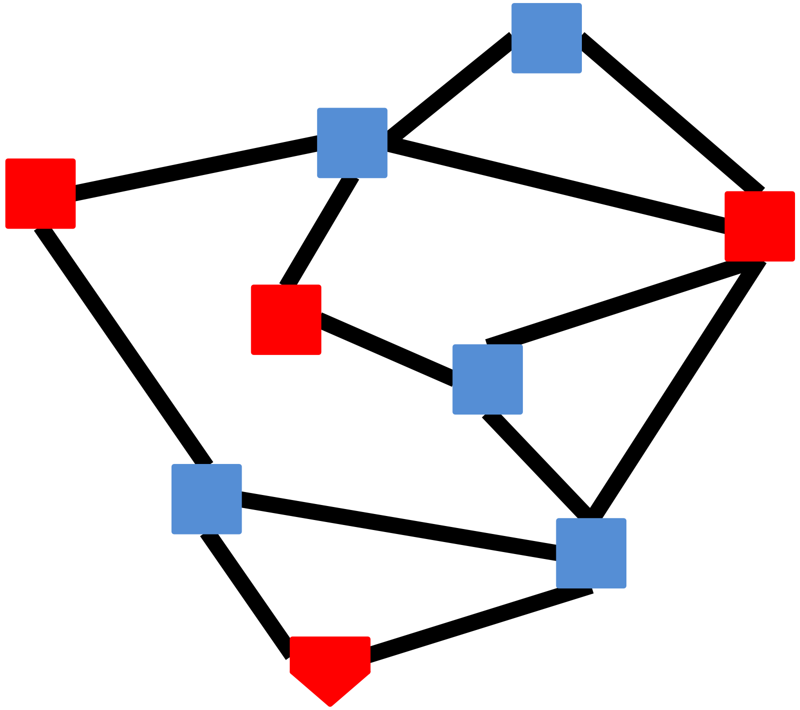
Reduction Idea

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

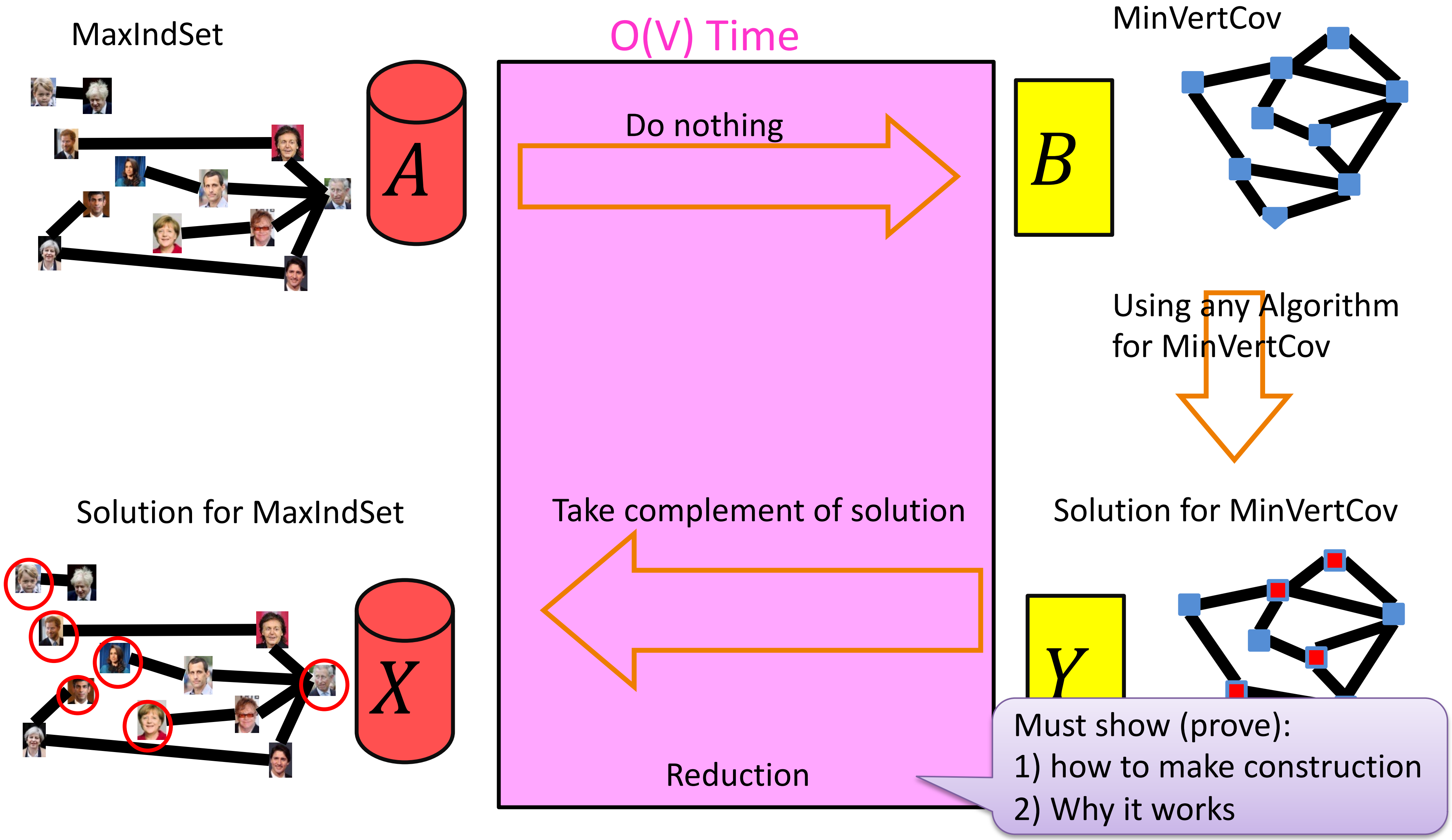
Vertex Cover



Independent Set



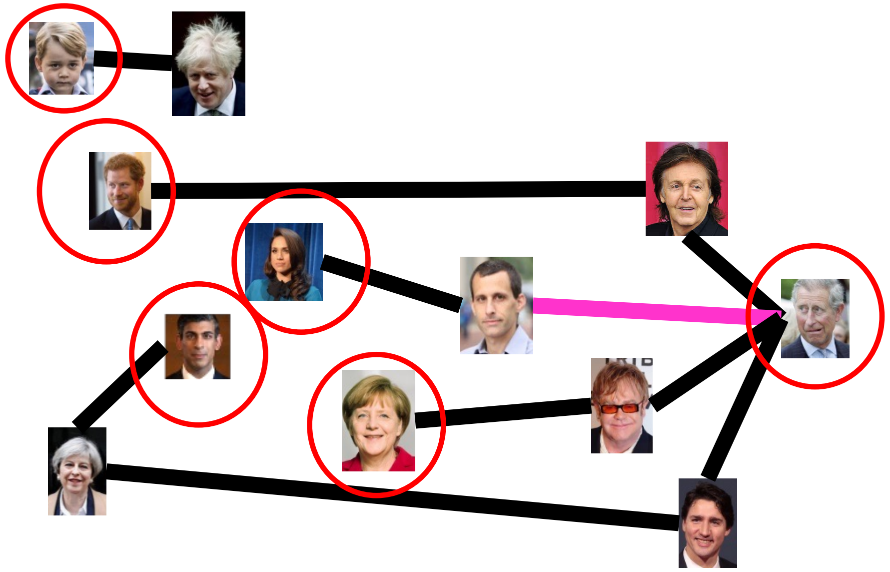
MaxVertCov V-Time Reducible to MinIndSet



Proof: $\Rightarrow$

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 $(x, y) \in E$

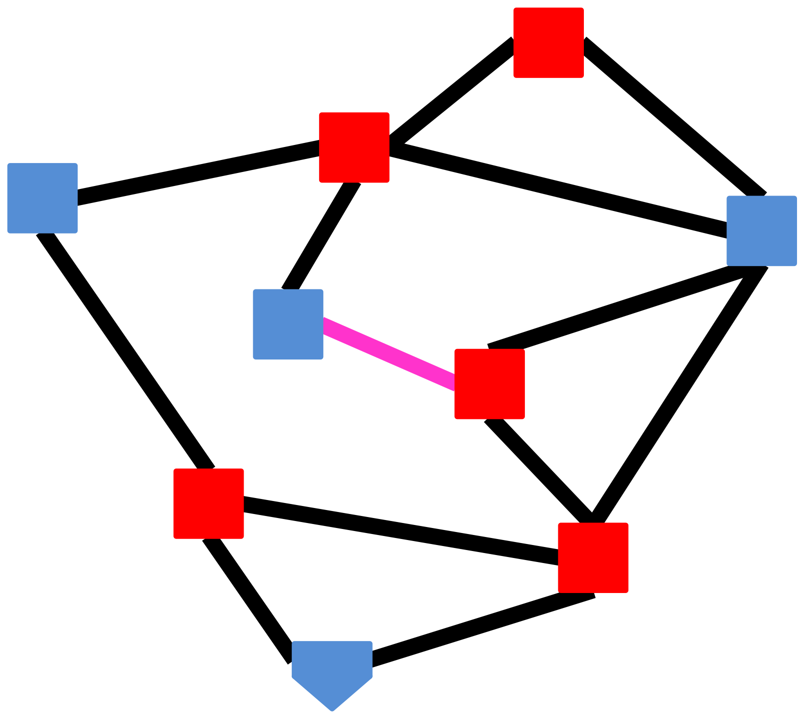
If $x \in S$ then $y \notin S$, because other wise S would not be an independent set

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



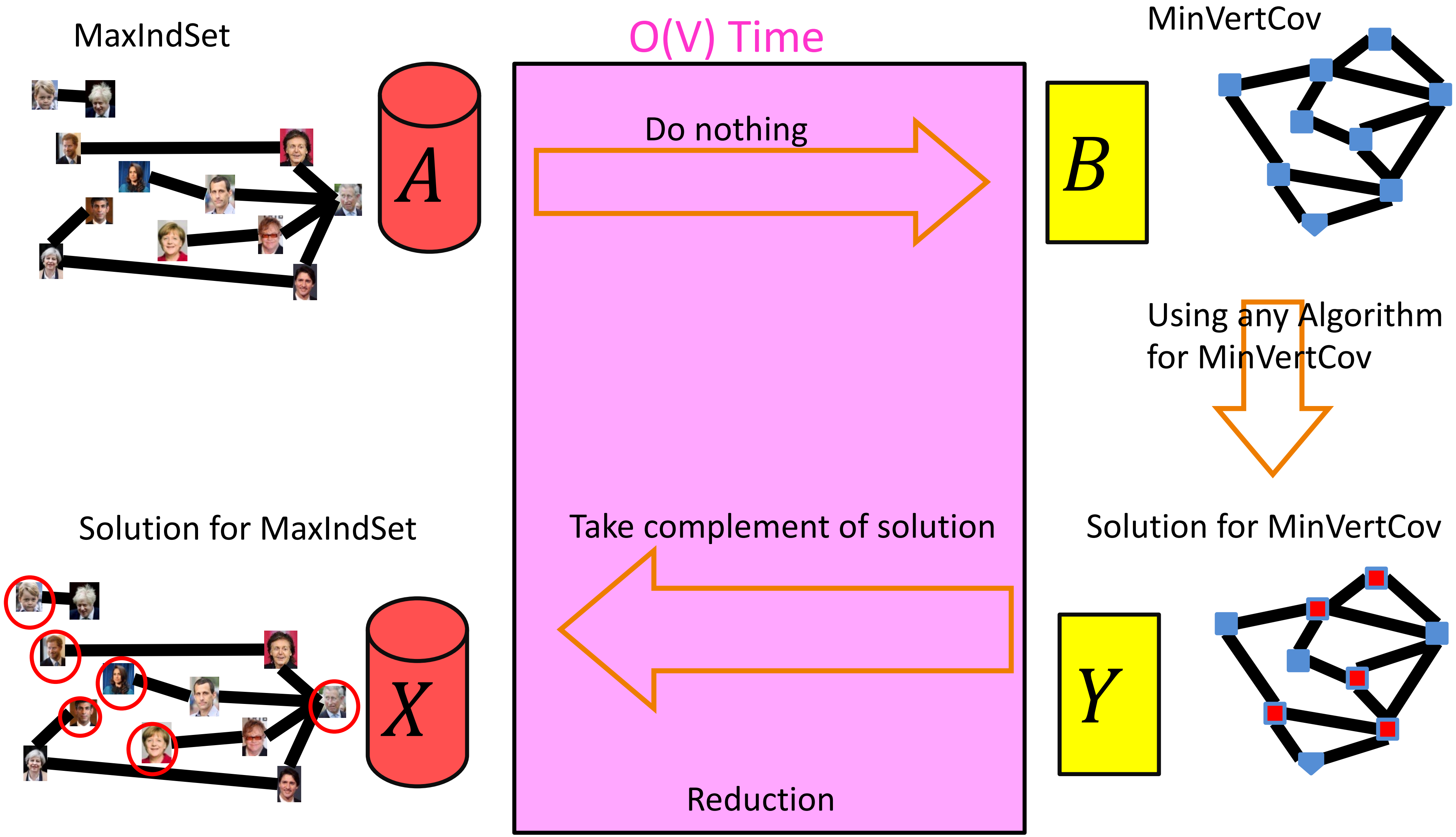
Consider any edge $(x, y) \in E$

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

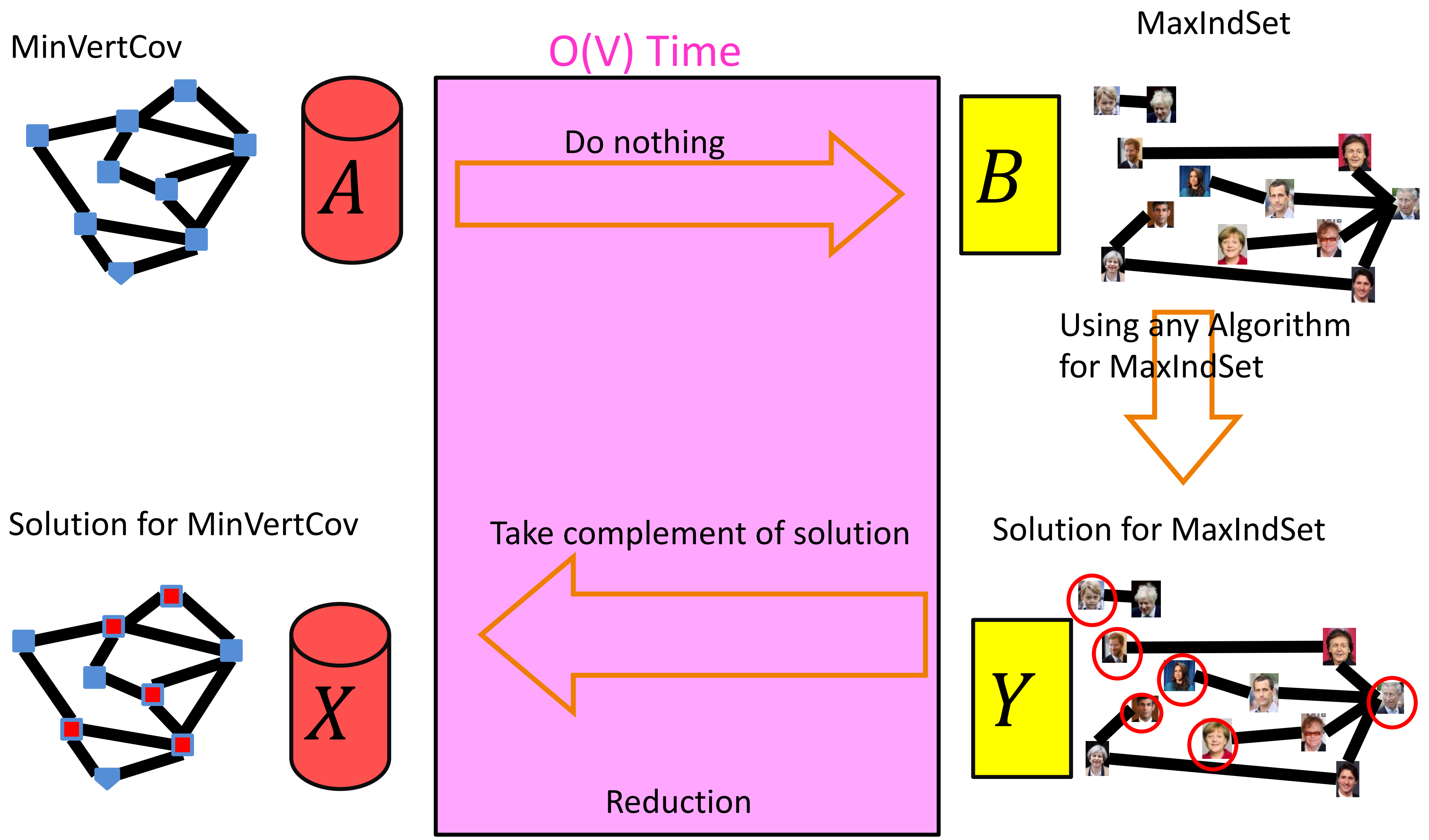
Therefore x and y are not both in S,

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

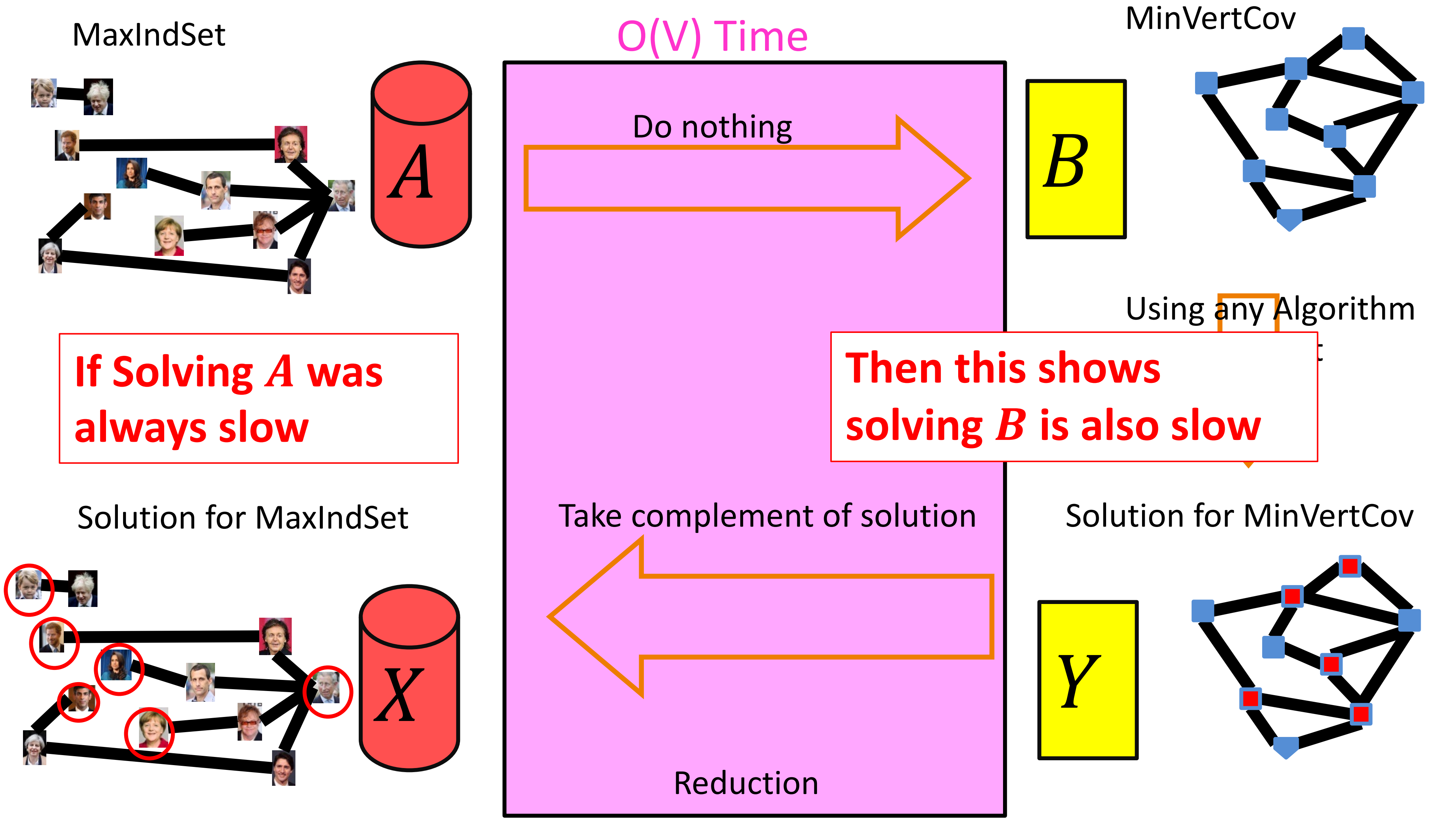
MaxVertCov V-Time Reducible to MinIndSet



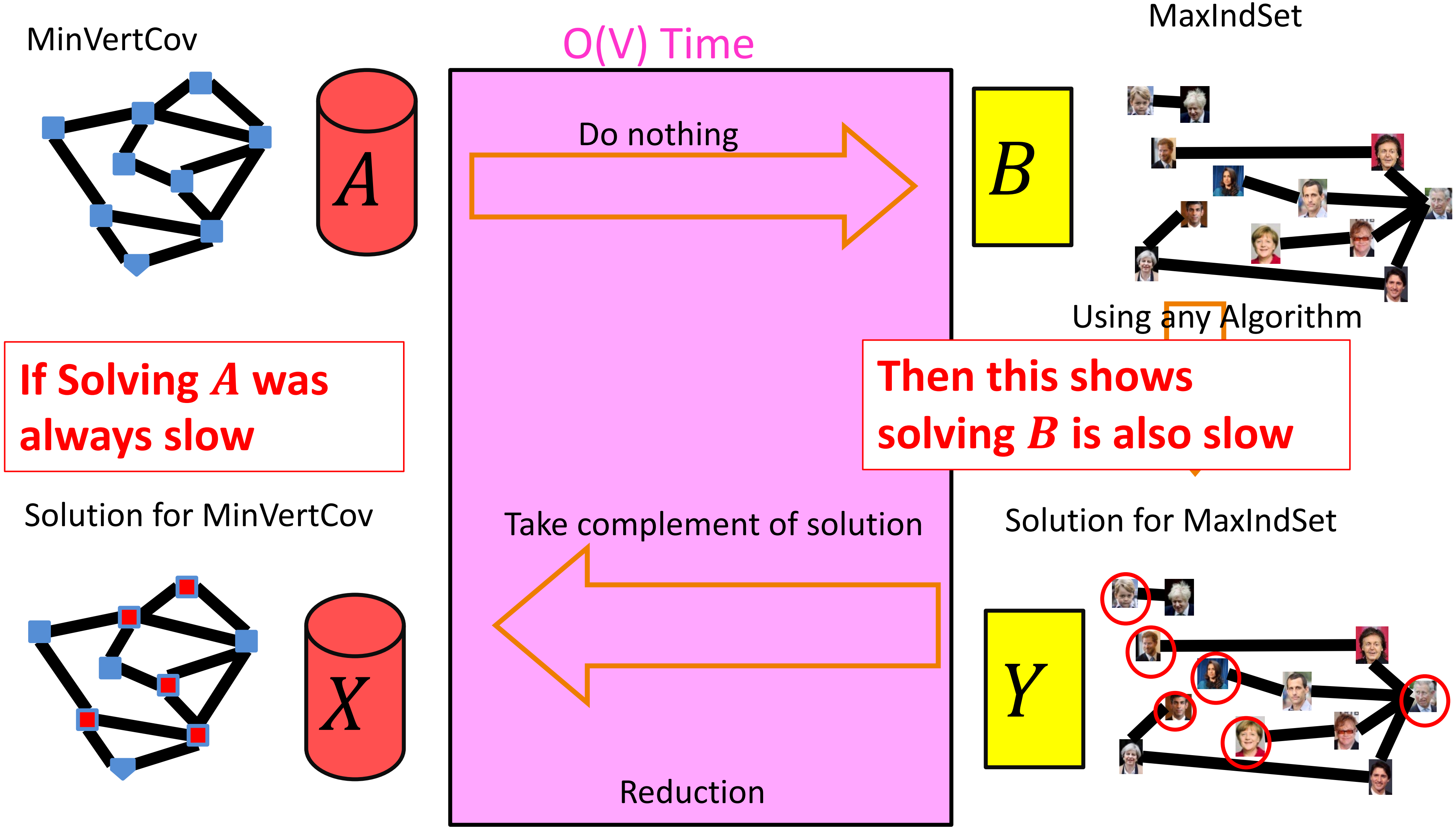
MaxIndSet V-Time Reducible to MinVertCov



Corollary


MaxIndSet
A
O(V) Time
Do nothing
B
MinVertCov
Using any Algorithm
If Solving A was always slow
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
A
Do nothing
B
Using any Algorithm
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
    - More in DMT2