NP-Completeness

CS 4102: Algorithms

Spring 2022

Robbie Hott and Tom Horton

Today's Keywords

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

Reductions

(We're about to get interested in problems that seem to require exponential time…)

Max-flow vs. Edge-disjoint paths

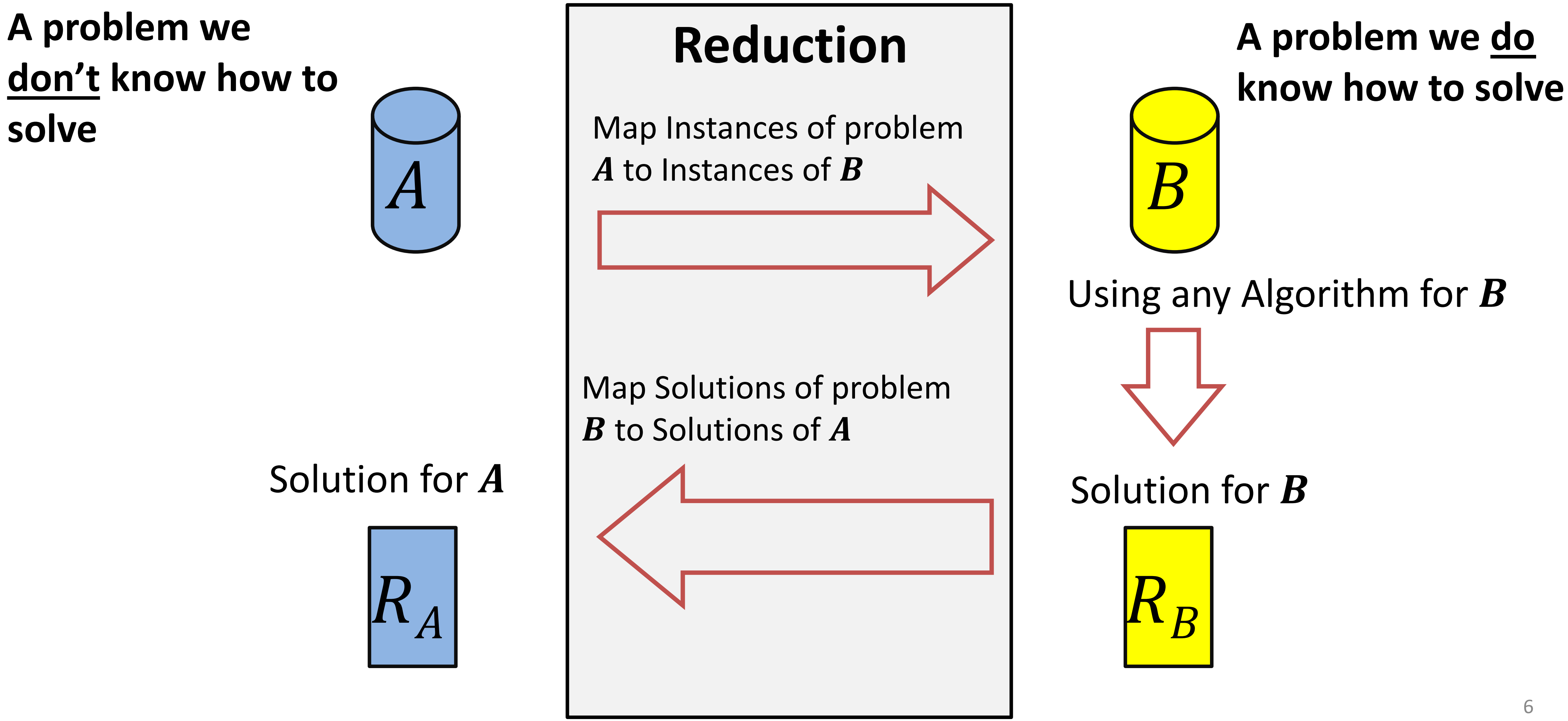
- These two problems are "related"
  - Here we're saying: if you can solve ***Max-flow***, you can solve ***Edge-disjoint path***
- Alternatively, we can say that one problem ***reduces*** to the other
  - The problem of finding Edge-disjoint paths ***reduces to*** the problem of finding max-flow
  - Maybe this ***reduction*** requires some work to "convert"
    - Could be nothing or minimal
  - For these problems, the cost of the conversion is ***hopefully small (more on this in a moment).***

Reduction

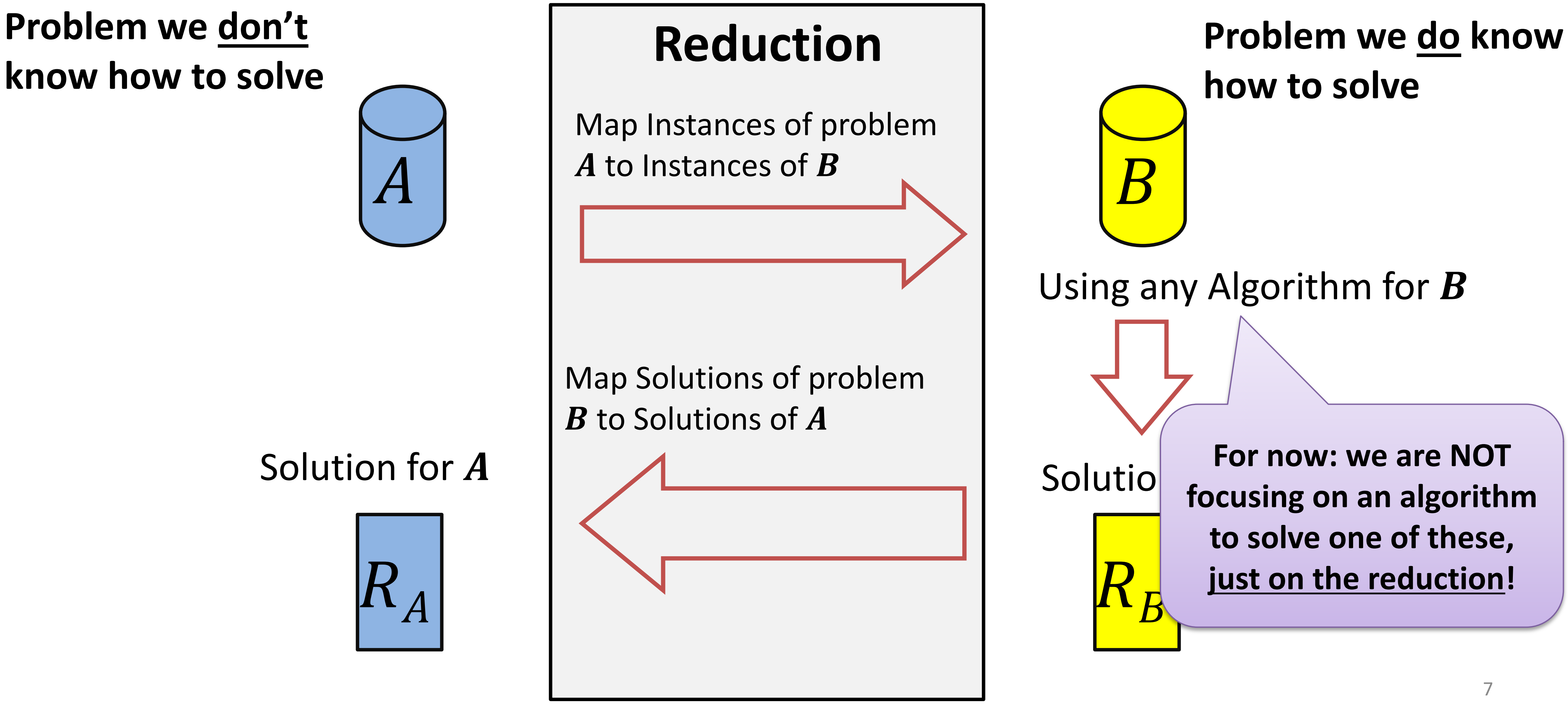
- A ***reduction*** is a transformation of one problem into another problem
  - Edge-disjoint paths is reducible to max-flow because we can use max-flow to solve it
  - Formally, problem A is ***reducible*** to problem B if we can use a solution to B to solve A
- We're particularly interested in reductions that happen fast!
  - *Meaning the work to do the conversion is fast (how fast?)*
- If A is ***polynomial-time reducible*** to B, we denote this as:

  $$A \le_p B$$
- **If the conversion takes linear time, we might say $A \le_n B$**

In General: A Reduction



In General: Reduction



Total Runtime

- The total runtime to solve A is:
  - ***Convert(A->B)***: Time it takes to convert problem A into problem B
  - ***Execute(B):*** Time it takes to execute algorithm to solve B
  - ***Solve(B->A)***: Time to convert solution of B back to solution of A

- ***Total Runtime: Convert(A->B) + Execute(B) + Solve(B->A)***

- Do you see why we want convert() and solve() to be FAST. We want the slowest part to be the actual algorithm that solves B if possible.

Relative Hardness

Total Runtime: Convert(A->B) + Execute(B) + Solve(B->A)

- Generally, we want execute() to be the slowest term, we then can say $A \leq_{\alpha} B$ where alpha is the runtime of convert() and solve()
- If Execute() IS the slowest part, then what does that mean about the relative difficulty of solving A versus B??
  - It means that B is as hard or harder than A. Why?
  - Because B provides an algorithm that solves A, so if algorithm to solve B gets faster, so does the algorithm that solves A.

Reductions and Hardness (again)

Problem we <u>don't</u> know how to solve

A

Solve A using B's solver.

Cost: Cost of B's solver plus O(f(n)) for reduction.

Solution for $\boldsymbol{A}$

R_A

Reduction

Map Instances of problem $\boldsymbol{A}$ to Instances of $\boldsymbol{B}$

Reduction cost: O(f(n))

Map Solutions of problem $\boldsymbol{B}$ to Solutions of $\boldsymbol{A}$

B

Problem we <u>do</u> know how to solve

Using any Algorithm for $\boldsymbol{B}$

R_B

Solution for $\boldsymbol{B}$

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

$\boldsymbol{A}$ is not a harder problem than $\boldsymbol{B}$

$$\boldsymbol{A \le B}$$

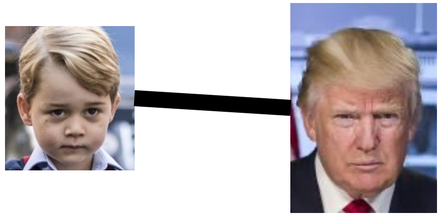
If $\boldsymbol{A}$ requires time $\boldsymbol{\Omega(f(n))}$ time, then $\boldsymbol{B}$ also requires $\boldsymbol{\Omega(f(n))}$ time

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

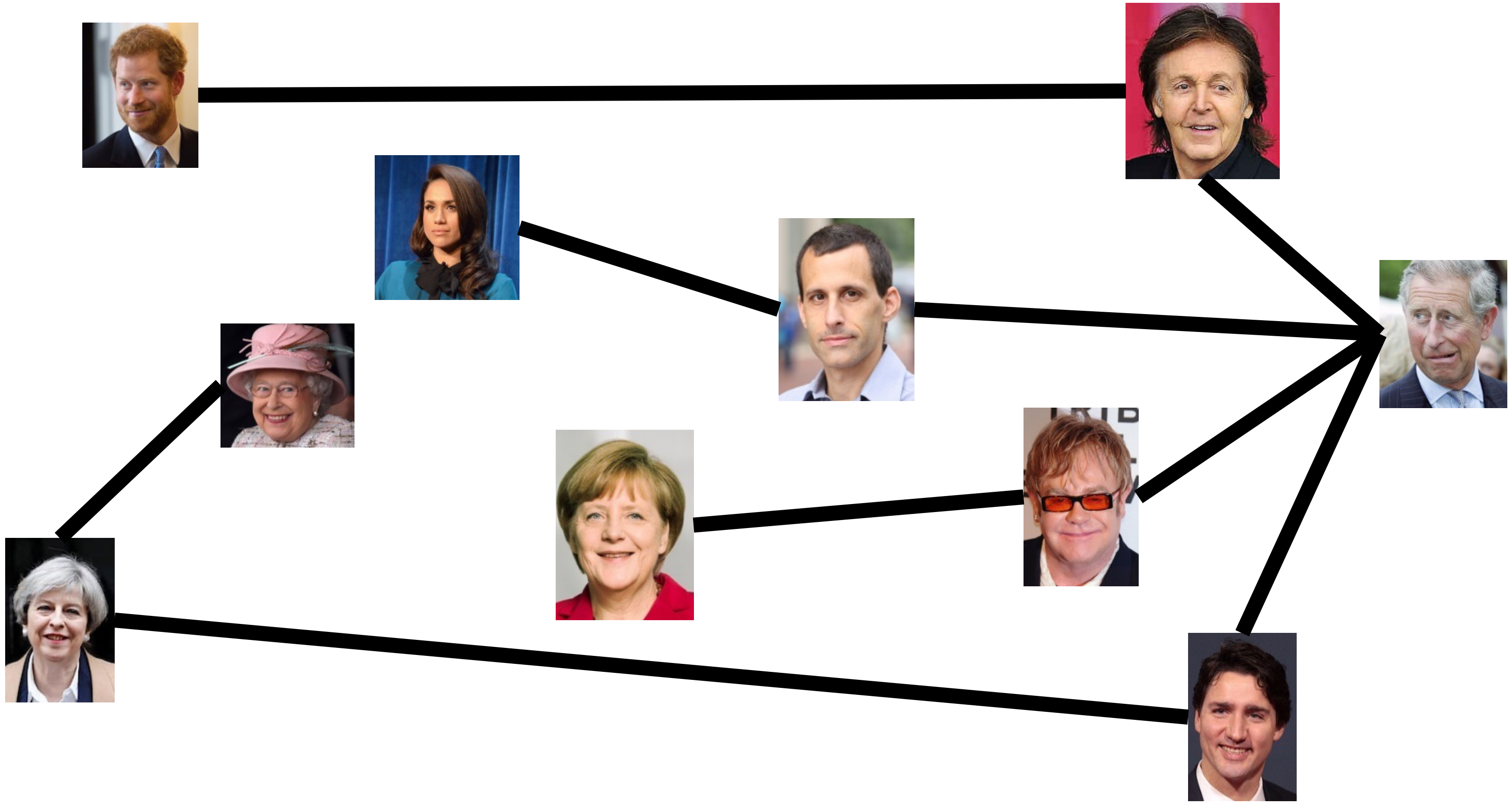
Reduction for Bipartite Matching

- We have transformed (in polynomial time) a **bipartite matching** problem into a **max flow** problem
- Specifically, bipartite-matching $\leq_p$ max-flow
  - Because we can transform bipartite matching to max-flow in polynomial time
- But is it the case that max-flow $\leq_p$ bipartite-matching?
  - Not so much: a solution to bipartite matching does not help us with a non-bipartite graph

Party Problem



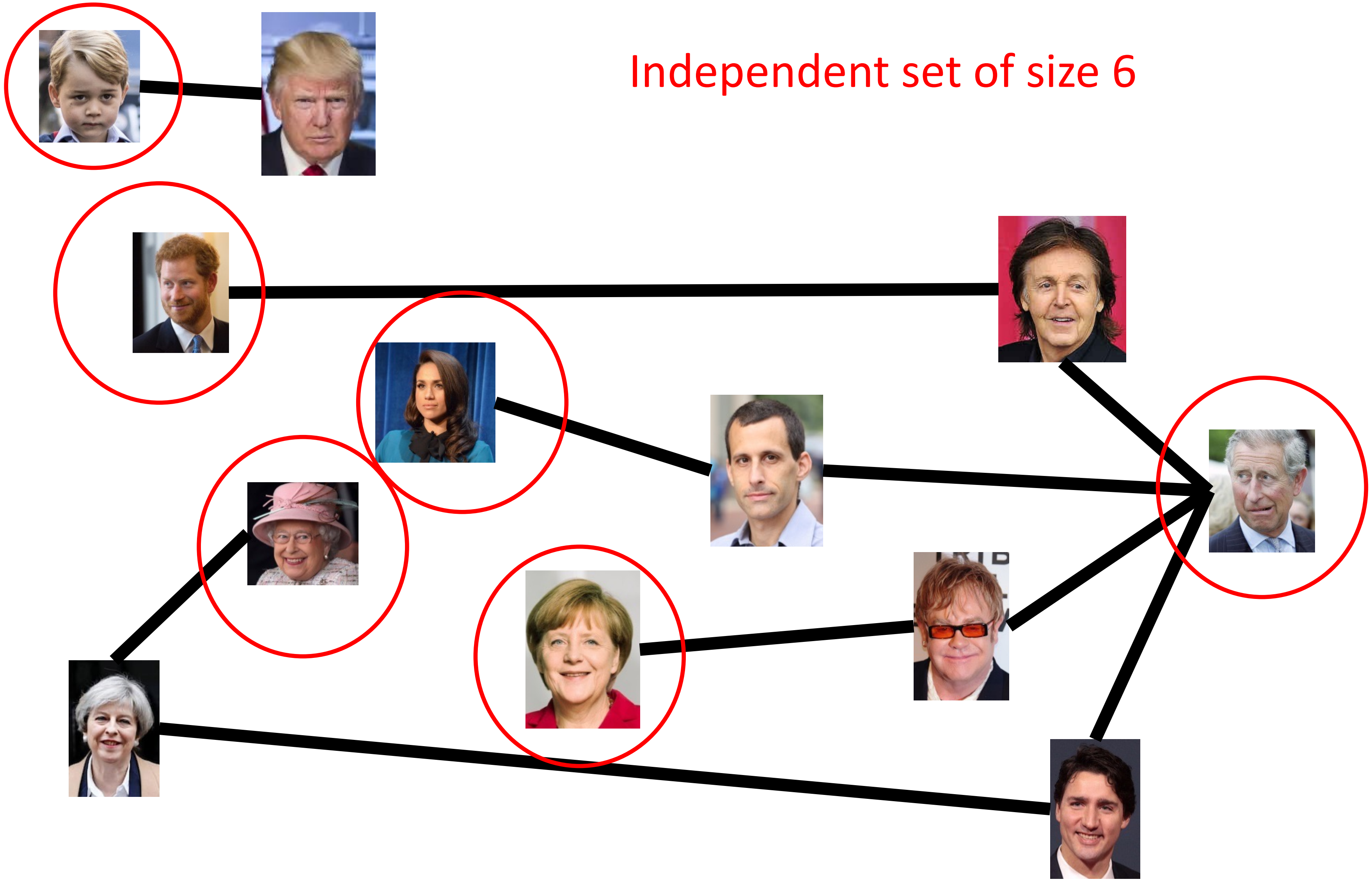
Draw edges between people who don’t get along.
Find the maximum number of people who get along.



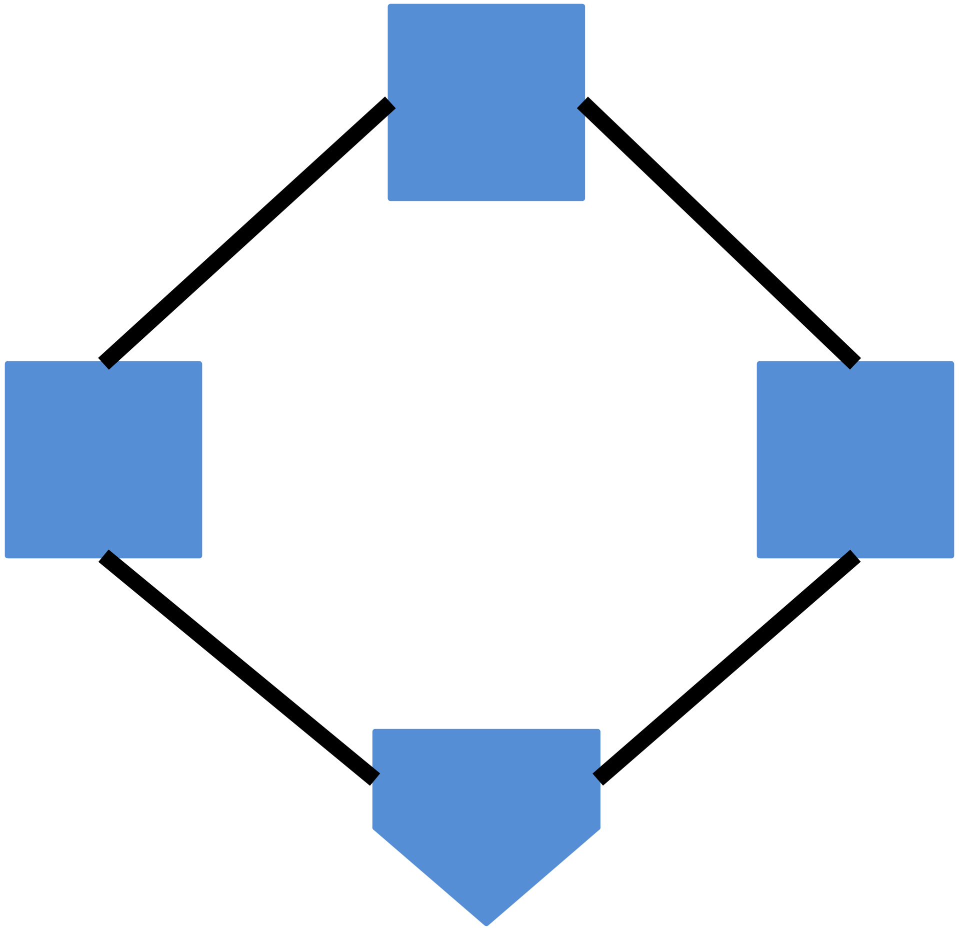
Maximum Independent Set

- 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

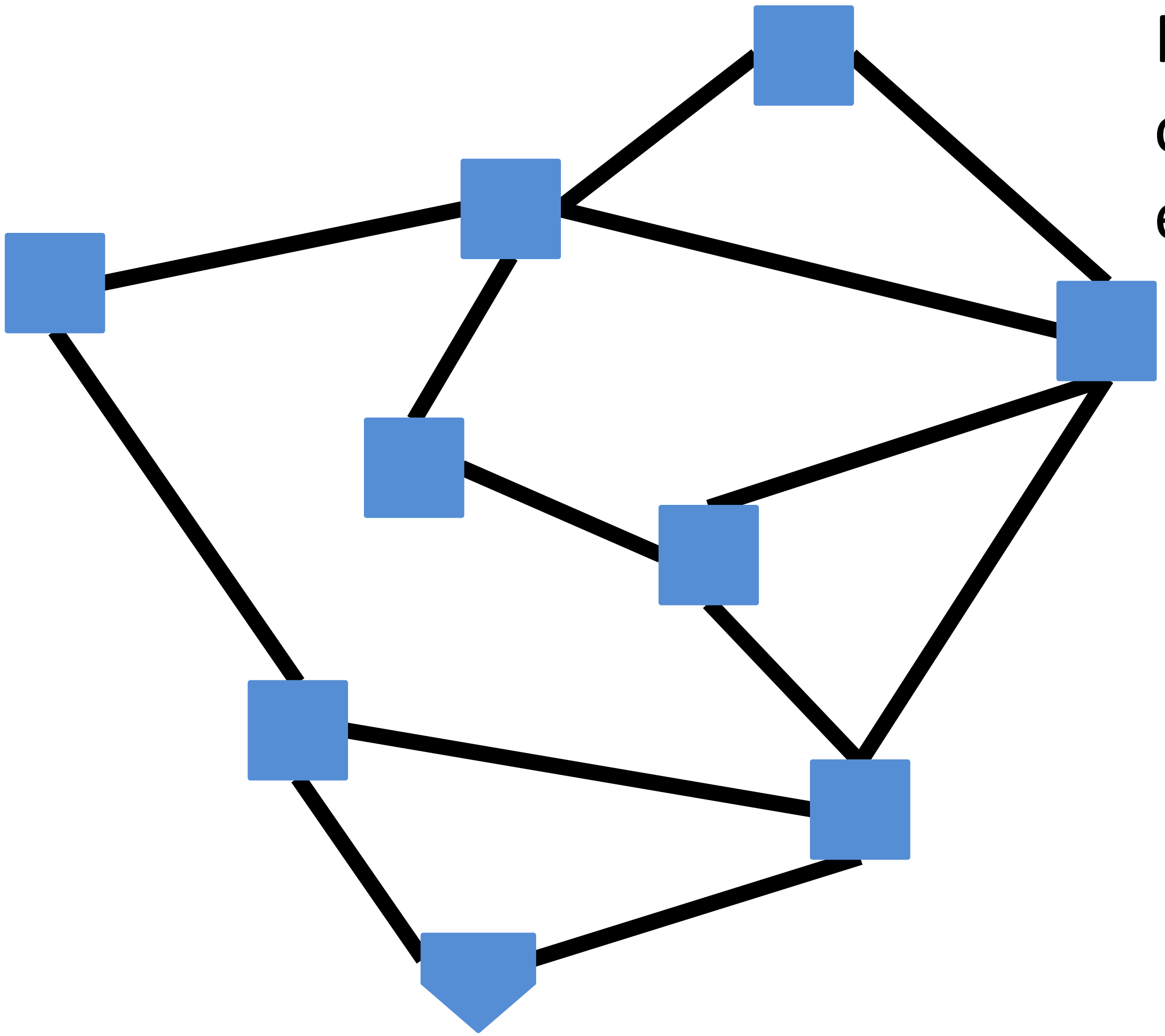
Example



Generalized Baseball



Generalized Baseball



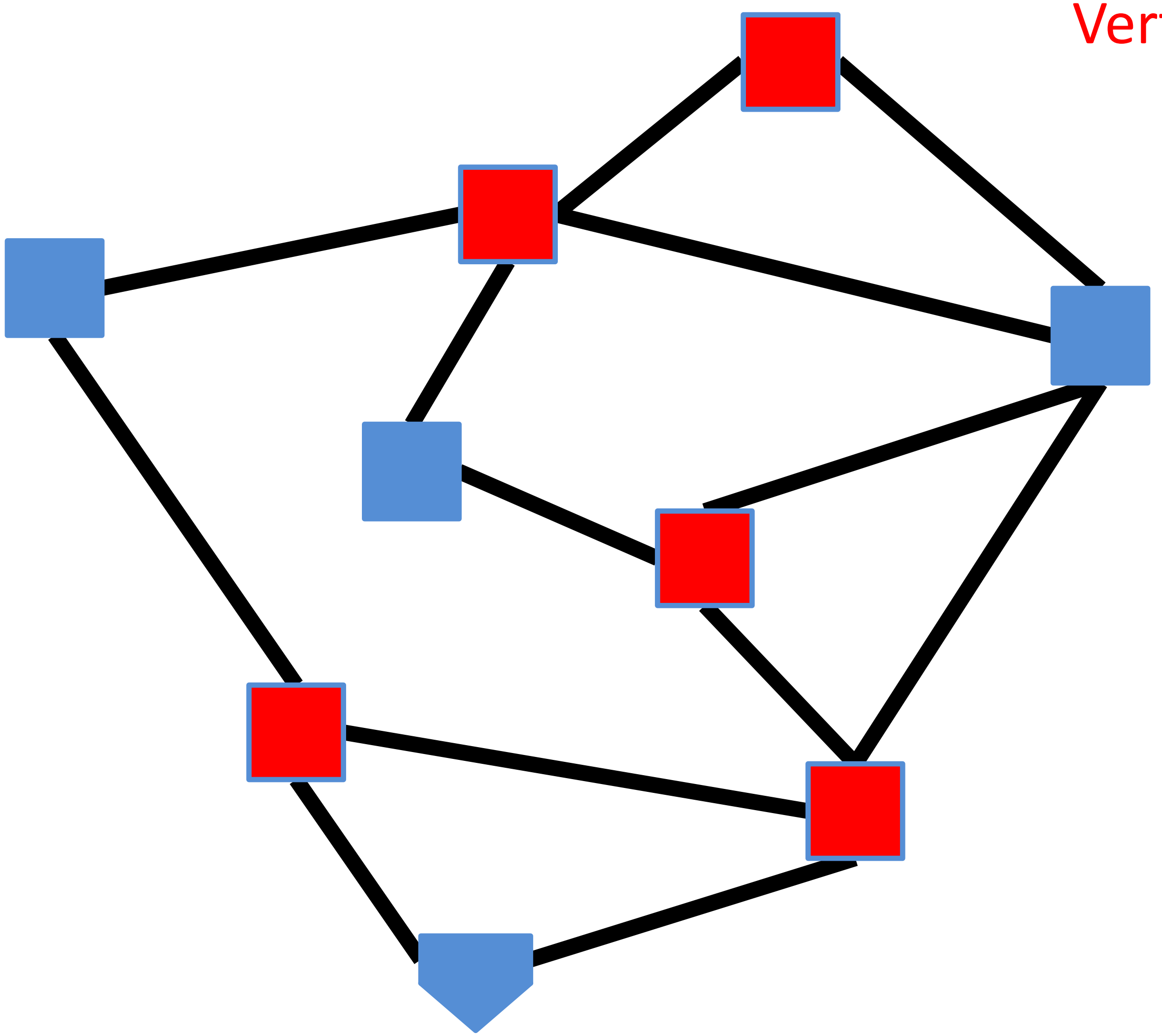
Need to place defenders on bases such that every edge is defended

What's the fewest number of defenders needed?

Minimum Vertex Cover

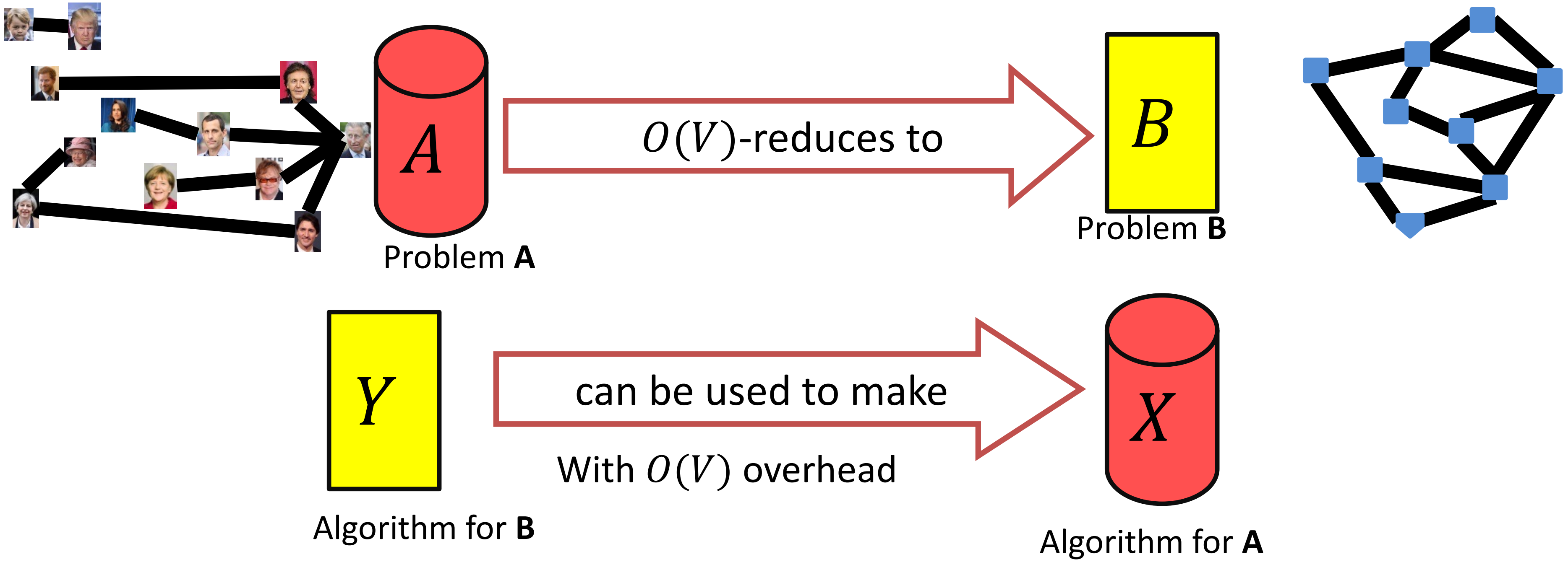
- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

Example



Vertex cover of size 5

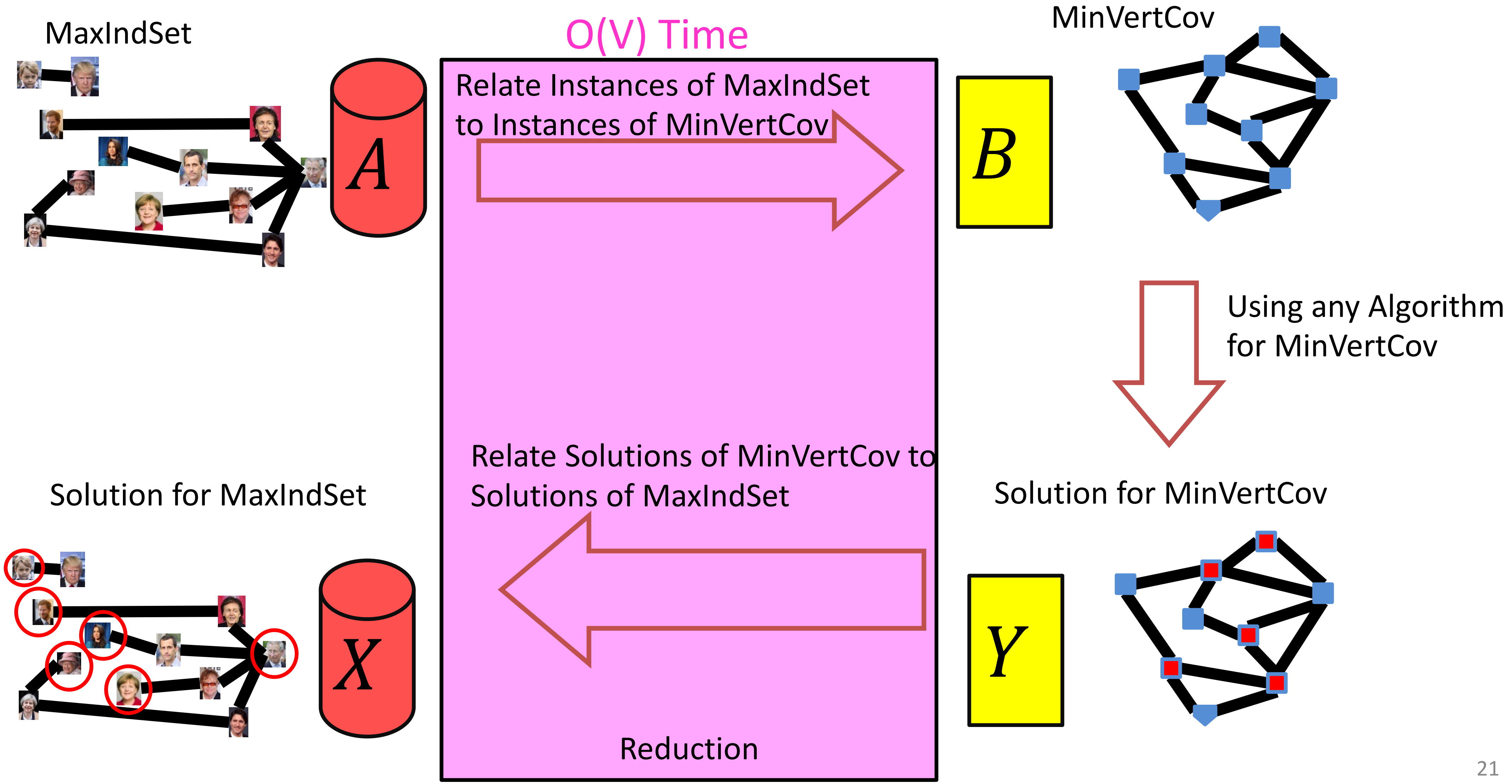
MaxIndSet$\leq_V$MinVertCov



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

$$A \leq_V B$$

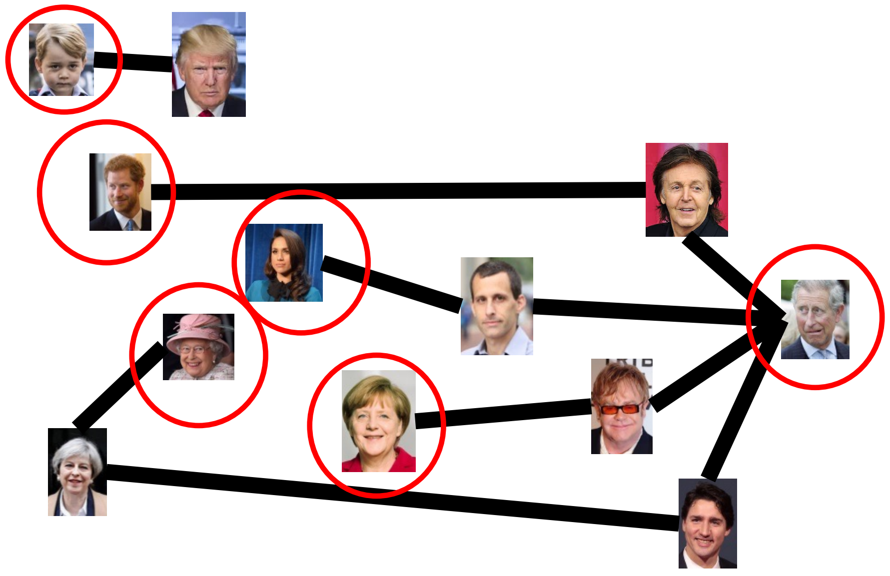
We need to build this Reduction



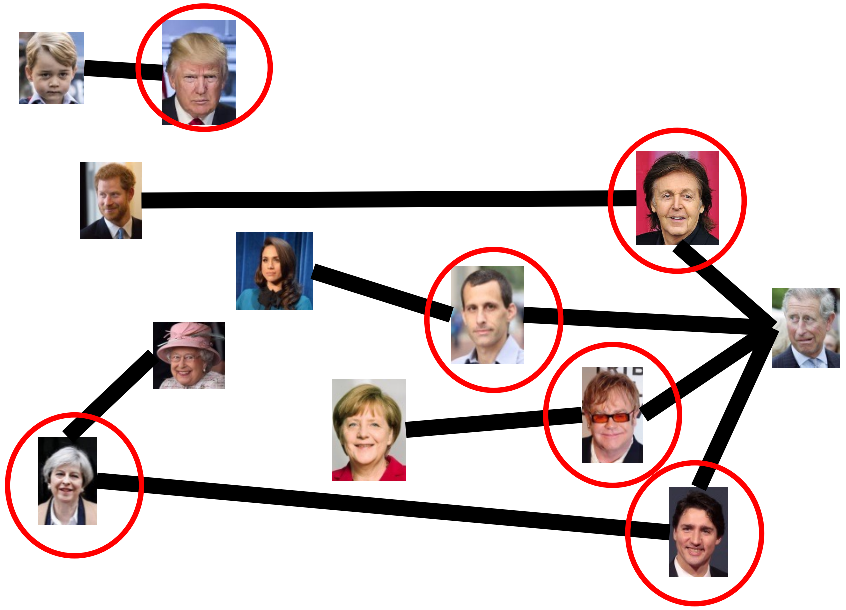
Reduction Idea

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

Independent Set



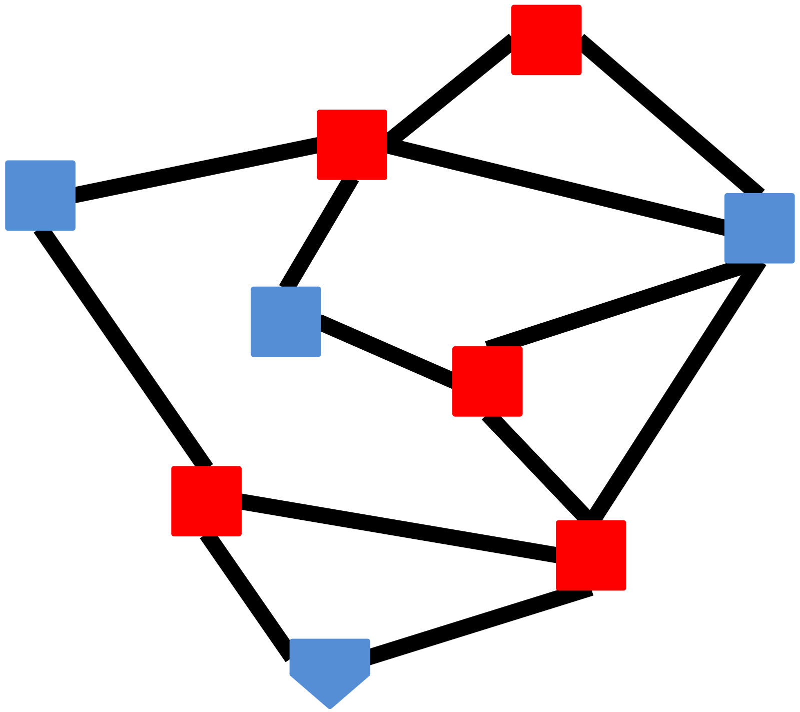
Vertex Cover



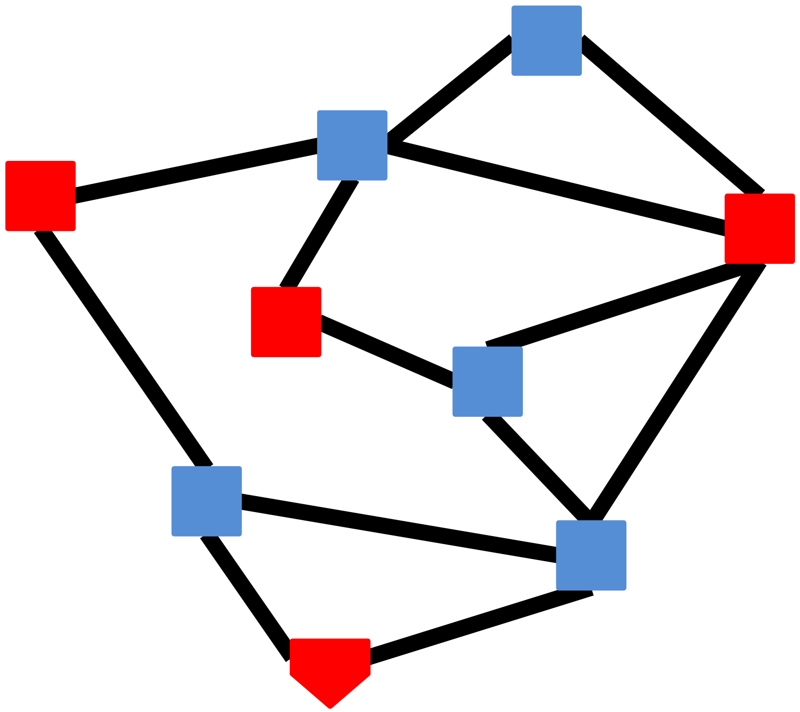
Reduction Idea

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

Vertex Cover



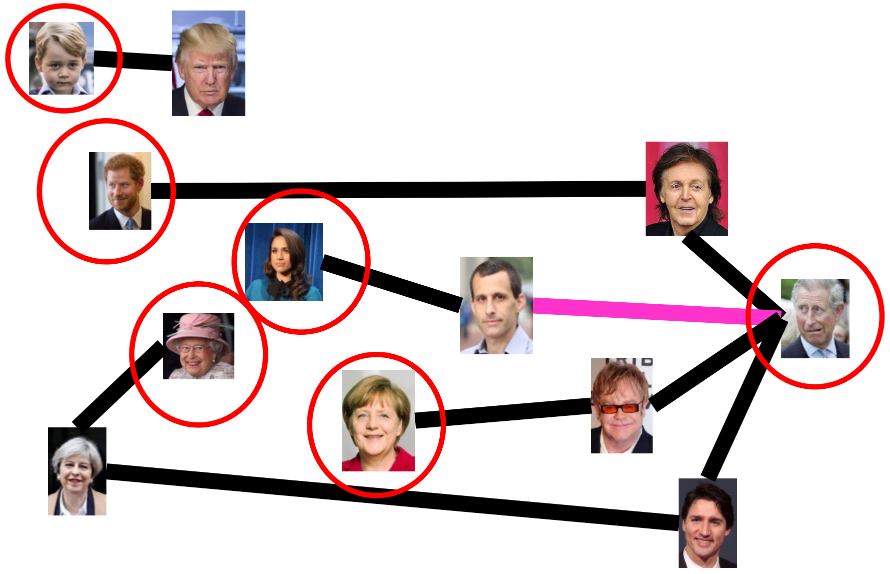
Independent Set



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 $(x, y) \in E$

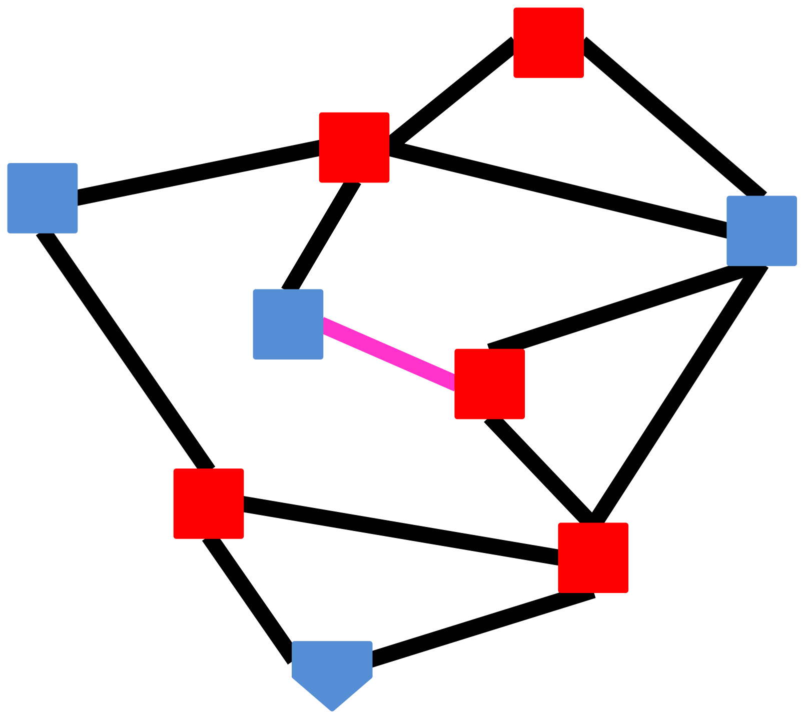
If $x \in S$ then $y \notin S$, because o.w. S would not be an independent set

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



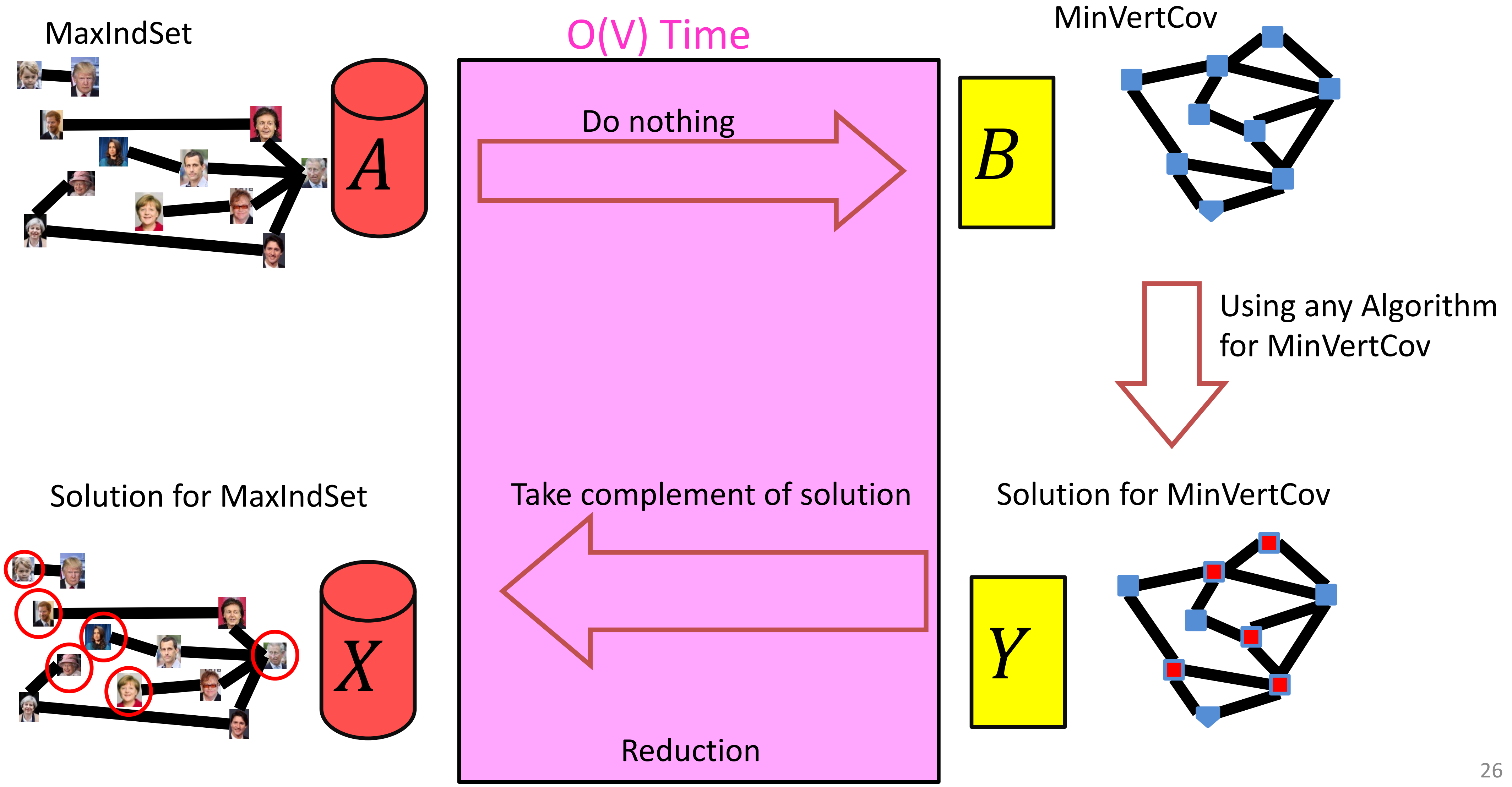
Consider any edge $(x, y) \in E$

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

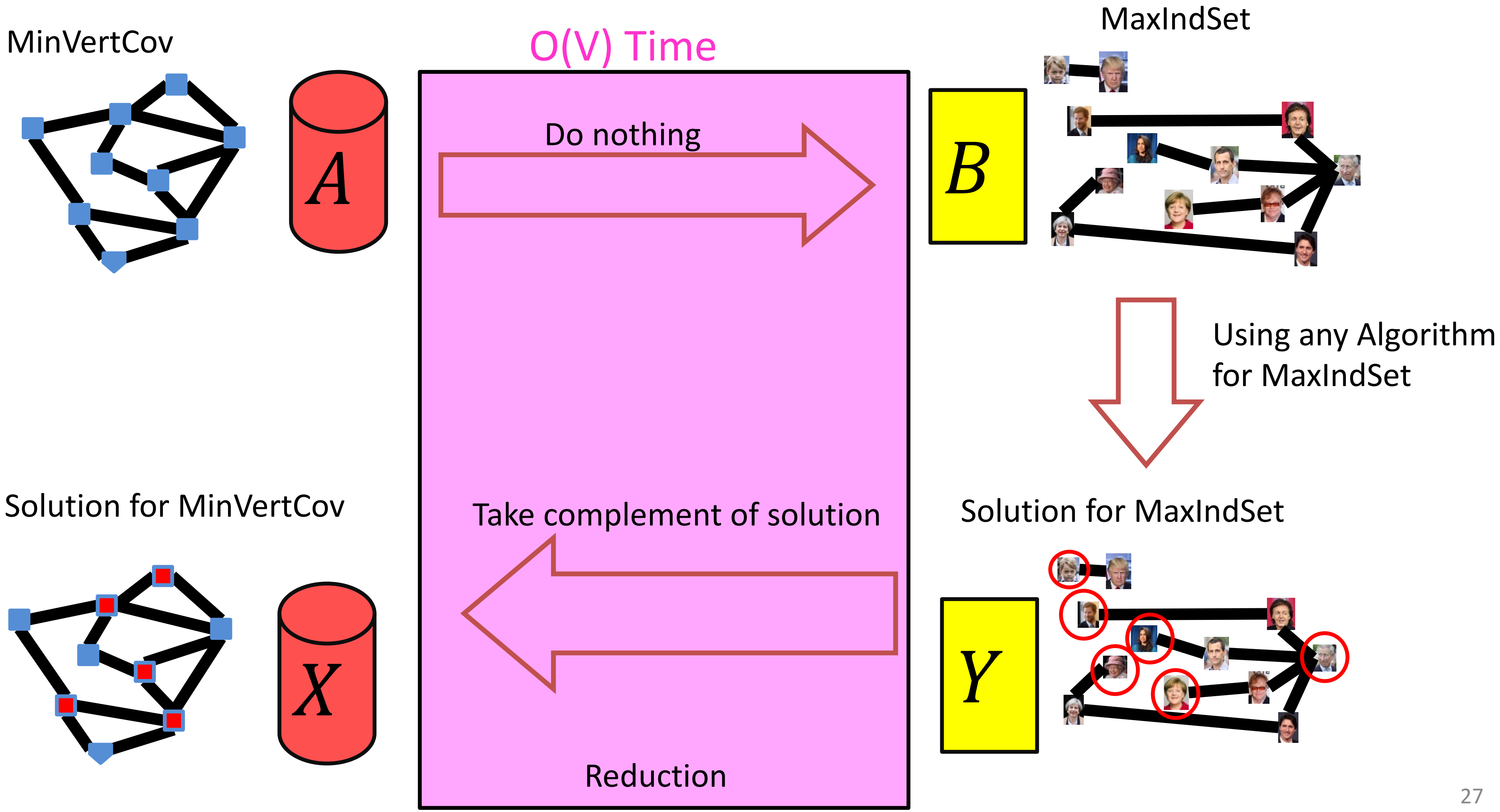
Therefore x and y are not both in S,

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

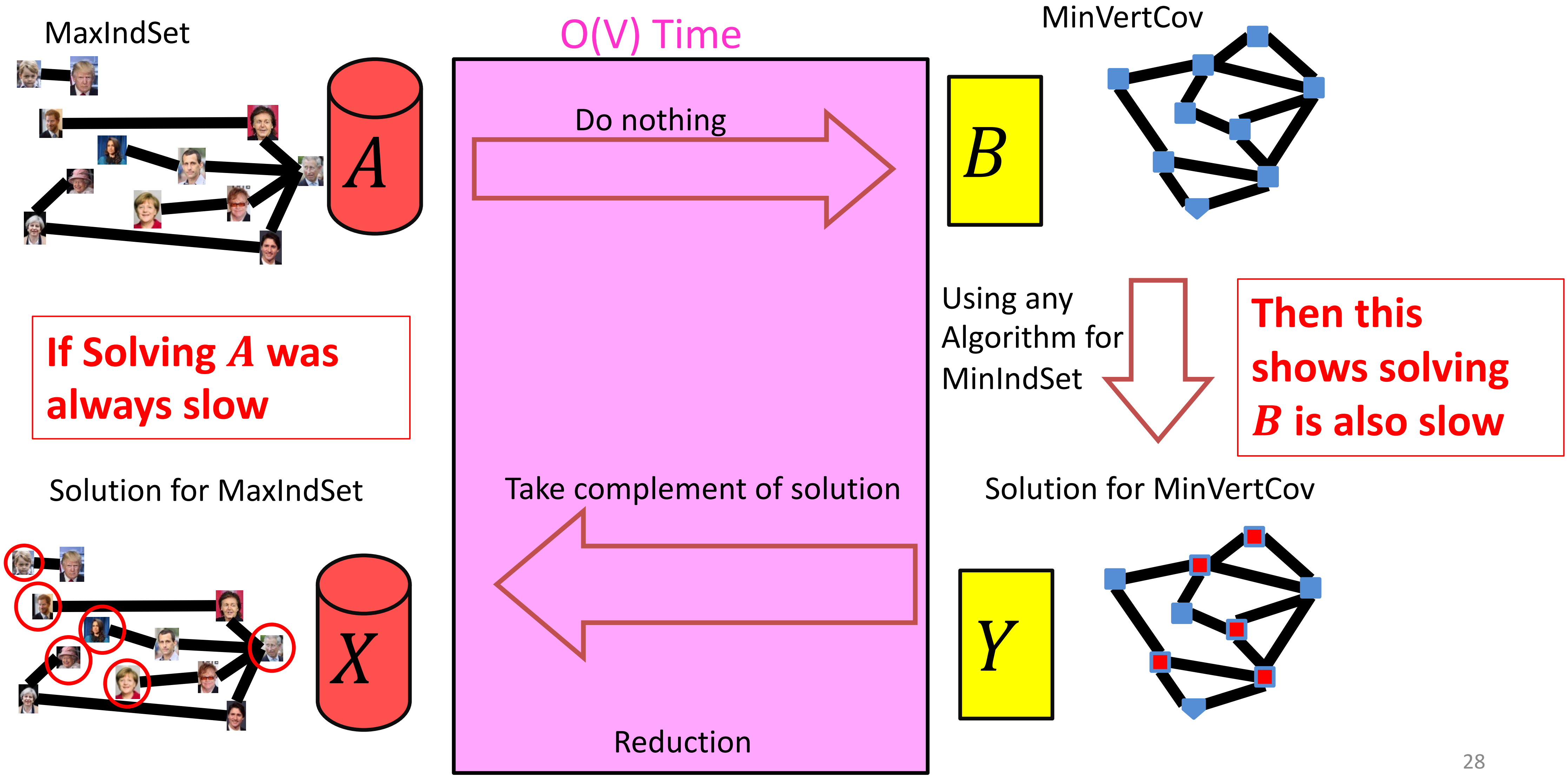
MaxVertCov V-Time Reducible to MinIndSet



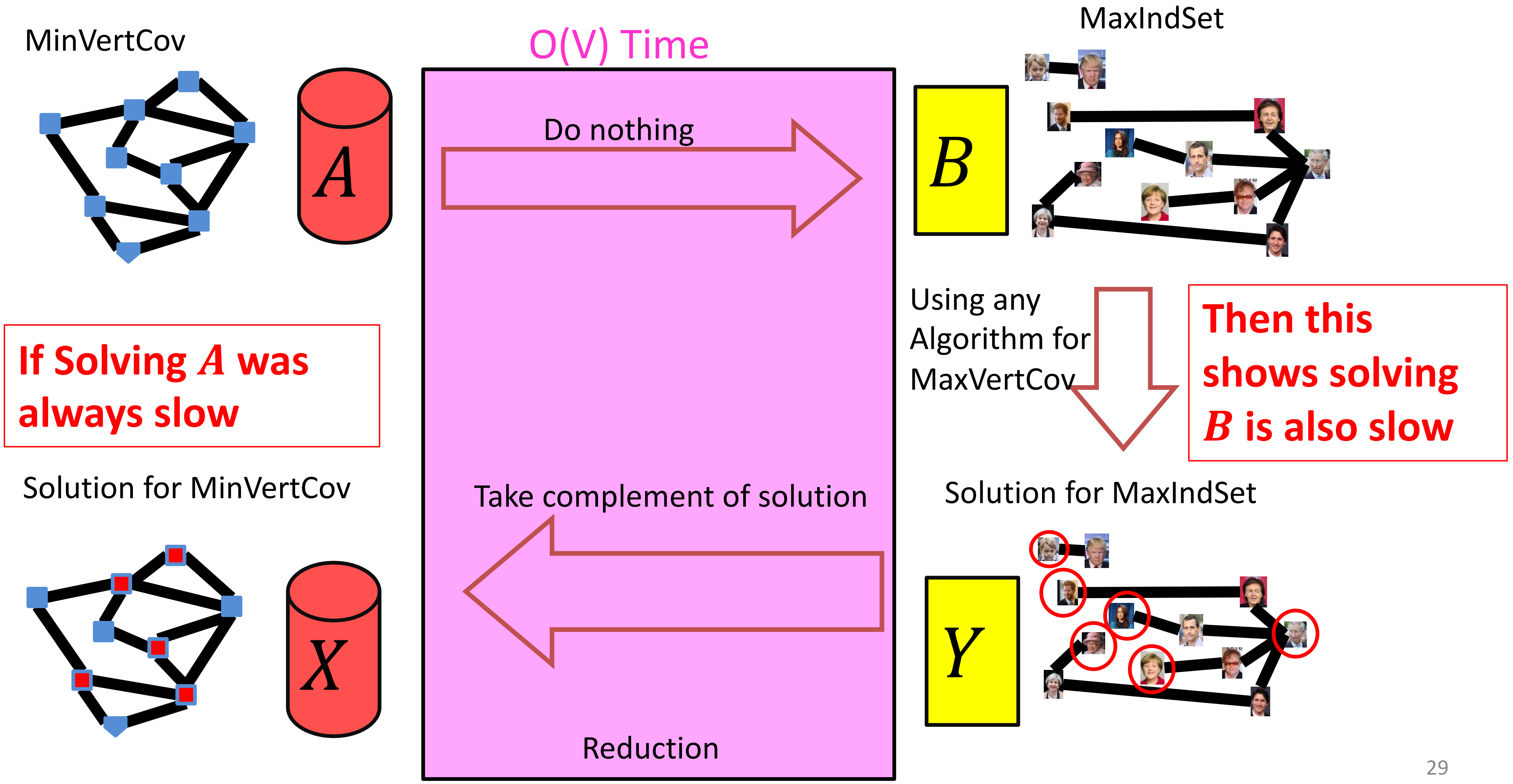
MaxVertCov V-Time Reducible to MinIndSet



Corollary


MaxIndSet
A
O(V) Time
Do nothing
MinVertCov
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
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
    - Next time!