CS4102 Algorithms

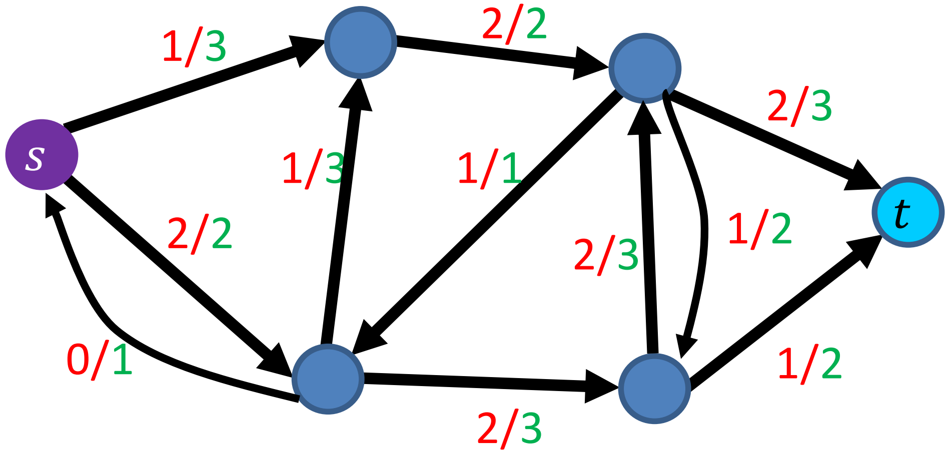
Fall 2018

<u>Warm up:</u>

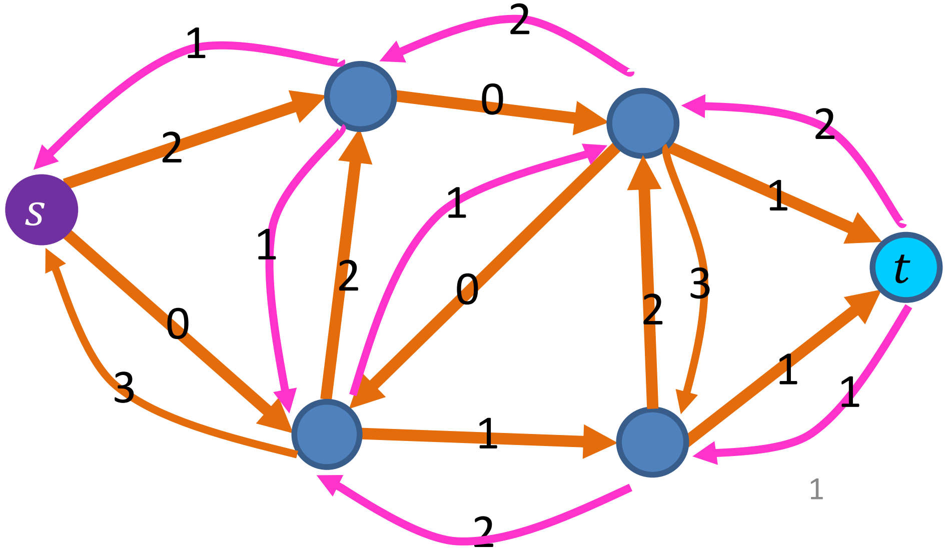
Remember Residual Graphs

- Keep track of net available flow along each edge
- "Forward edges": weight is equal to available flow along that edge in the flow graph
  - $w(e) = c(e) - f(e)$
- "Back edges": weight is equal to flow along that edge in the flow graph
  - $w(e) = f(e)$

Flow Graph G



Residual Graph G_f



Today's Keywords

- Reductions
- Bipartite Matching
- Vertex Cover
- Independent Set

CLRS Readings

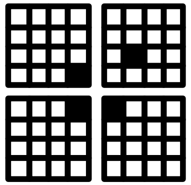
- Chapter 34

Homeworks

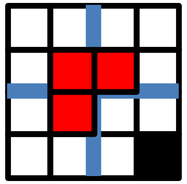
- HW8 due Friday 11/30 at 11pm
  - Written (use LaTeX)
  - Graphs

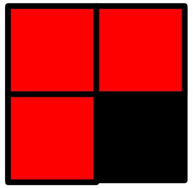
Divide and Conquer*

When is this a good strategy?

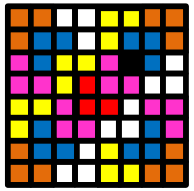


- **Divide**:
  - Break the problem into multiple subproblems, each smaller instances of the original
- **Conquer**:
  - If the suproblems are "large":
    - Solve each subproblem recursively
  - If the subproblems are "small":
    - Solve them directly (base case)





- **Combine**:
  - Merge together solutions to subproblems



*CLRS Chapter 4

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
  2. Select a good order for solving subproblems
     - Usually smallest problem first

Greedy Algorithms

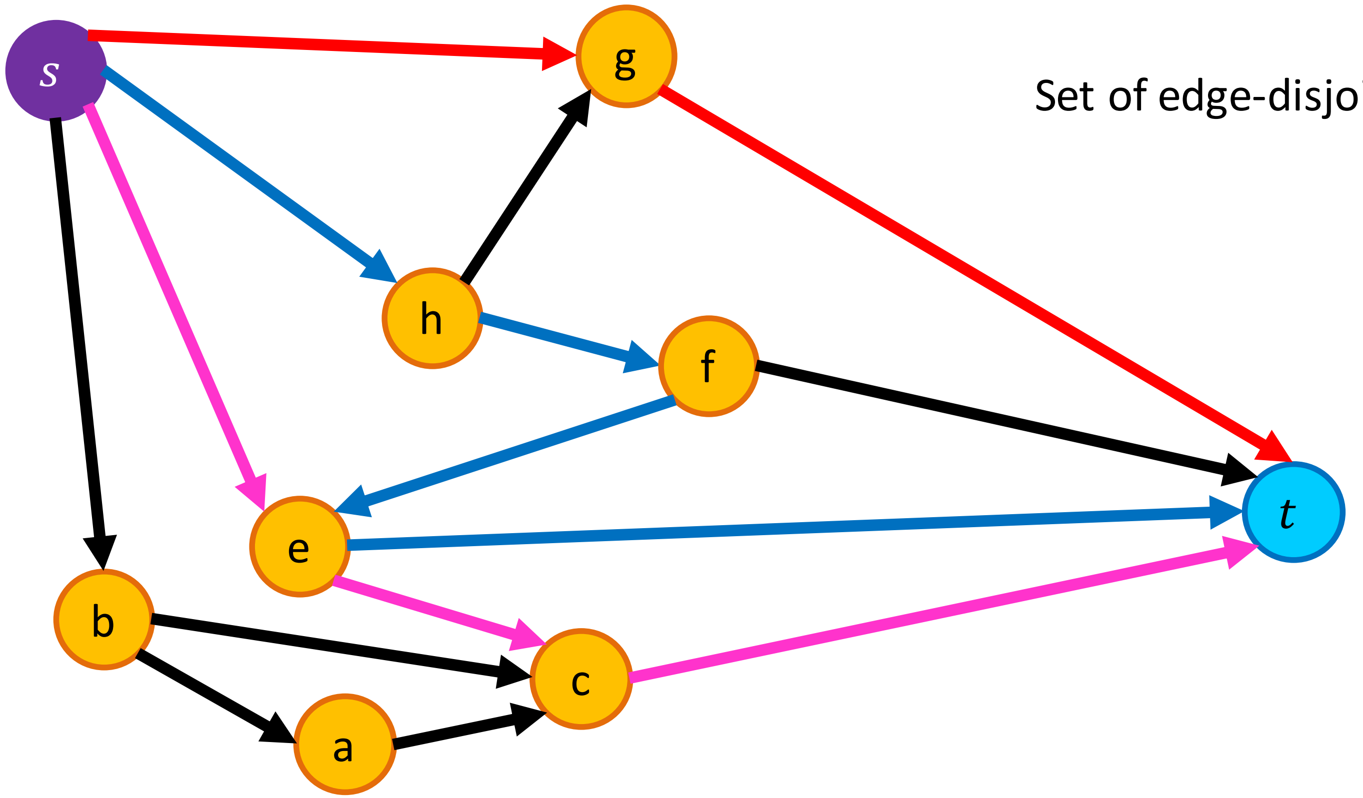
- Require Optimal Substructure
  - Solution to larger problem contains the solution to a smaller one
  - Only one subproblem to consider!
- Idea:
  1. Identify a greedy choice property
     - How to make a choice guaranteed to be included in some optimal solution
  2. Repeatedly apply the choice property until no subproblems remain

So far

- Divide and Conquer, Dynamic Programming, Greedy
  - Take an instance of Problem A, relate it to smaller instances of Problem A
- Next:
  - Take an instance of Problem A, relate it to an instance of Problem B

Edge-Disjoint Paths

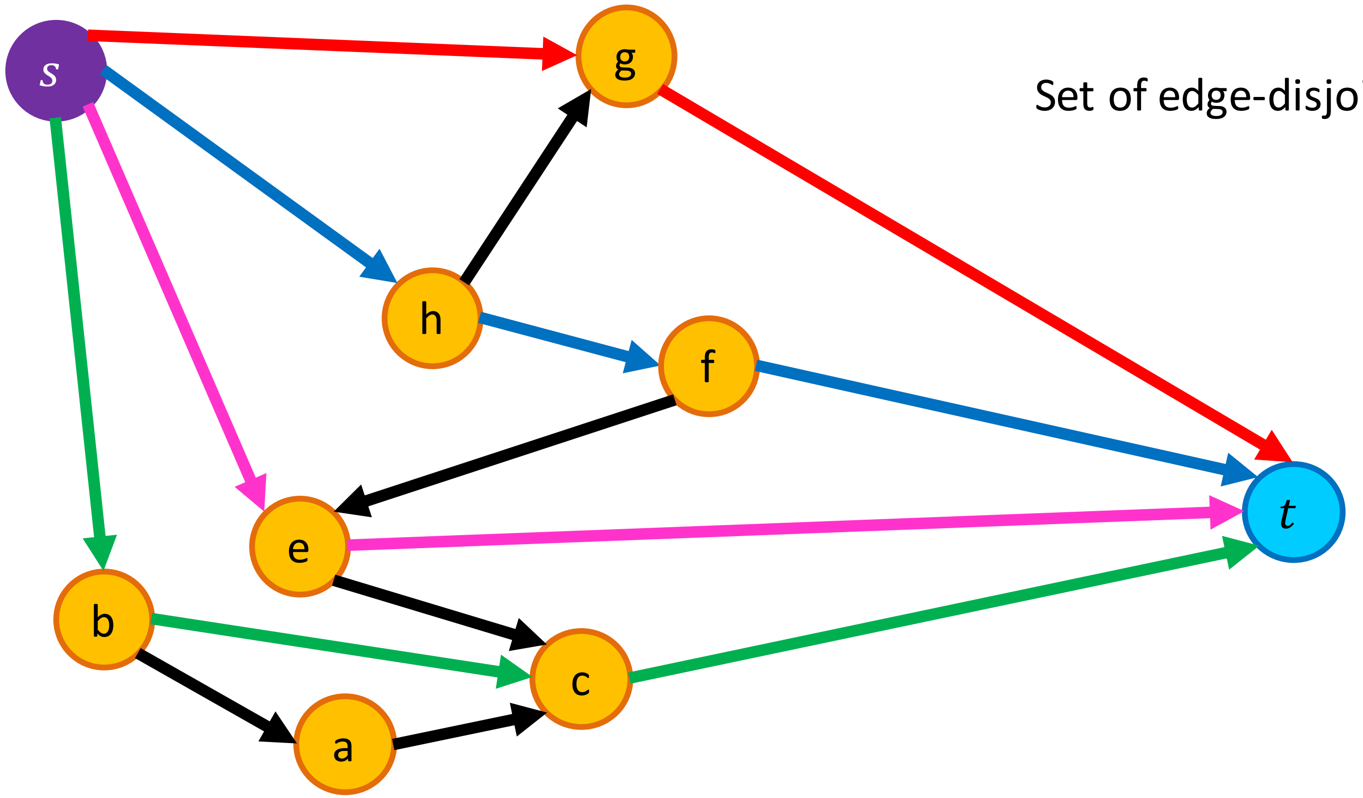
Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no edges



Set of edge-disjoint paths of size 3

Edge-Disjoint Paths

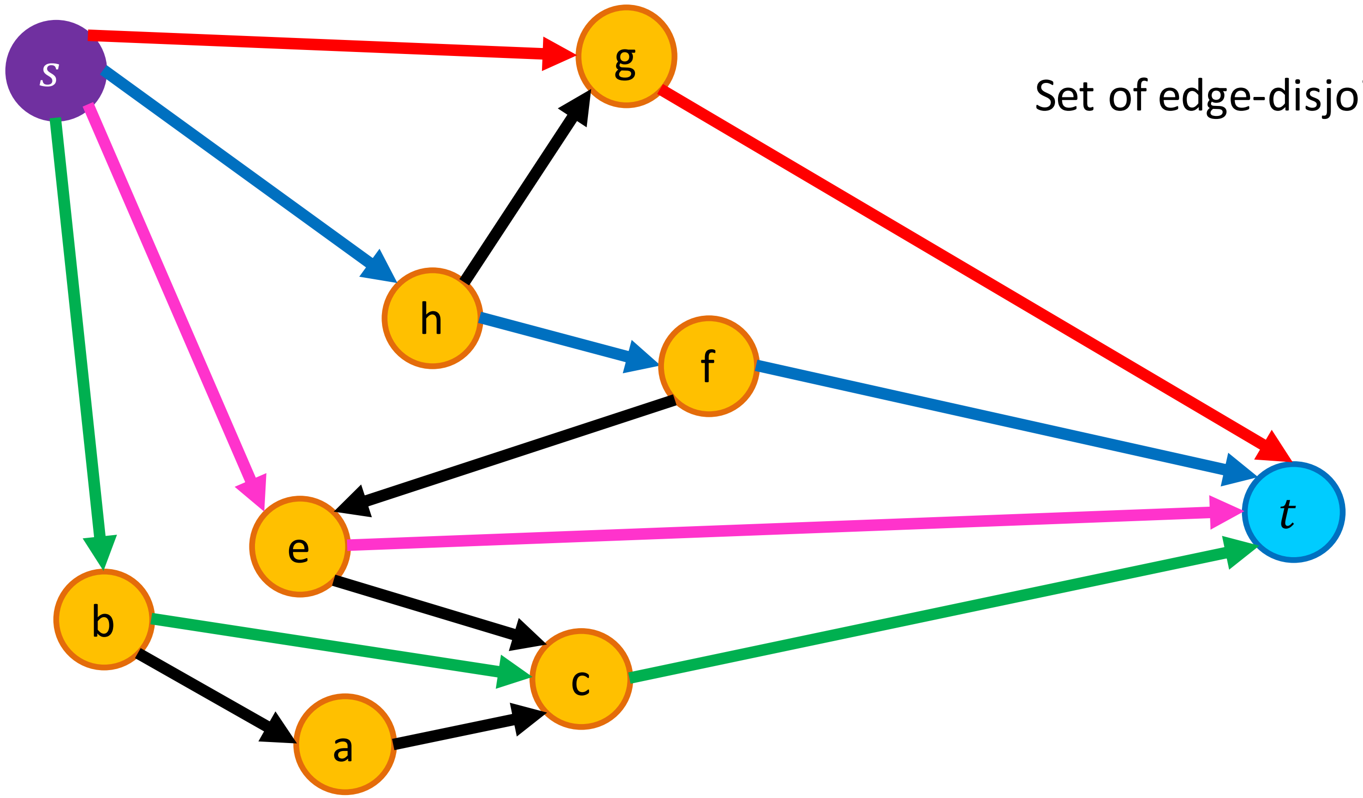
Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no edges



Set of edge-disjoint paths of size 4

Edge-Disjoint Paths Algorithm

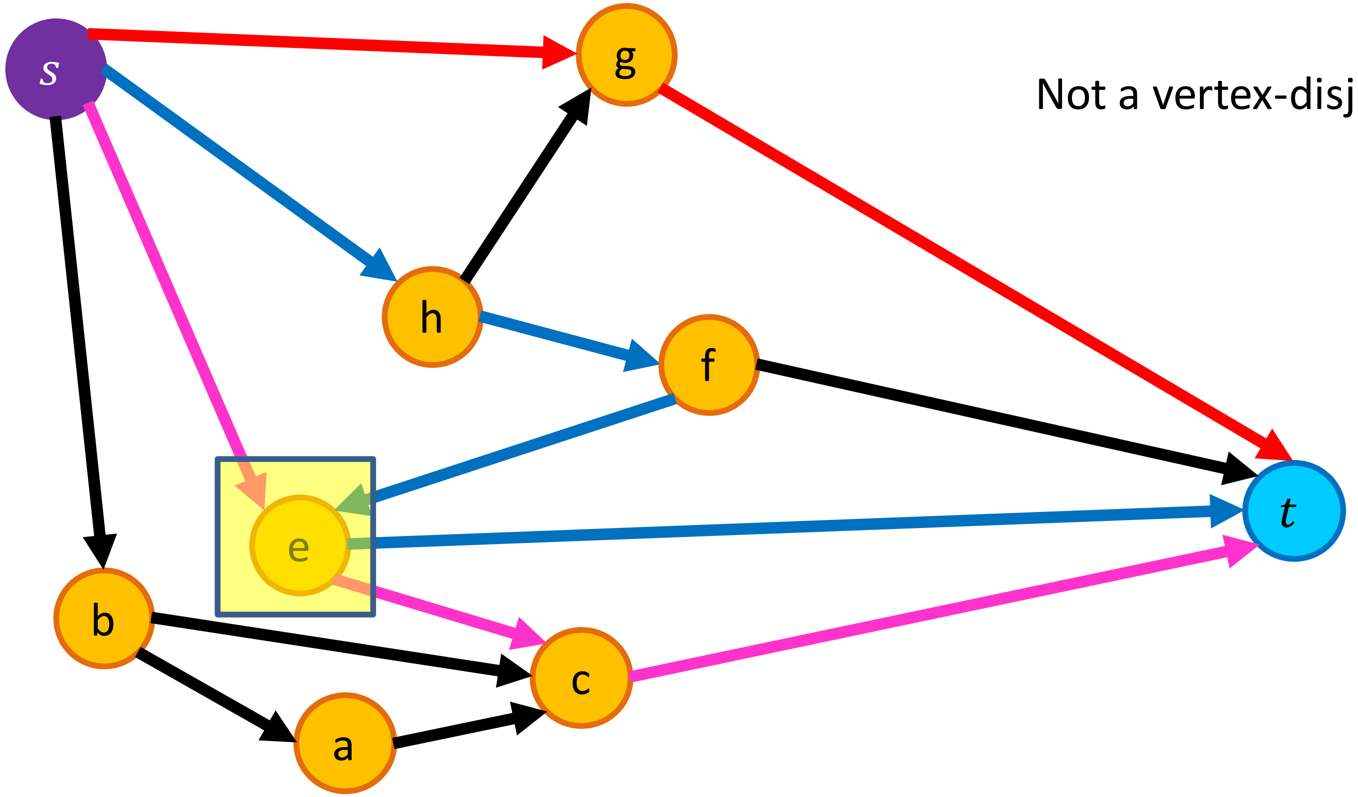
Make s and t the source and sink, give each edge capacity 1, find the max flow.



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



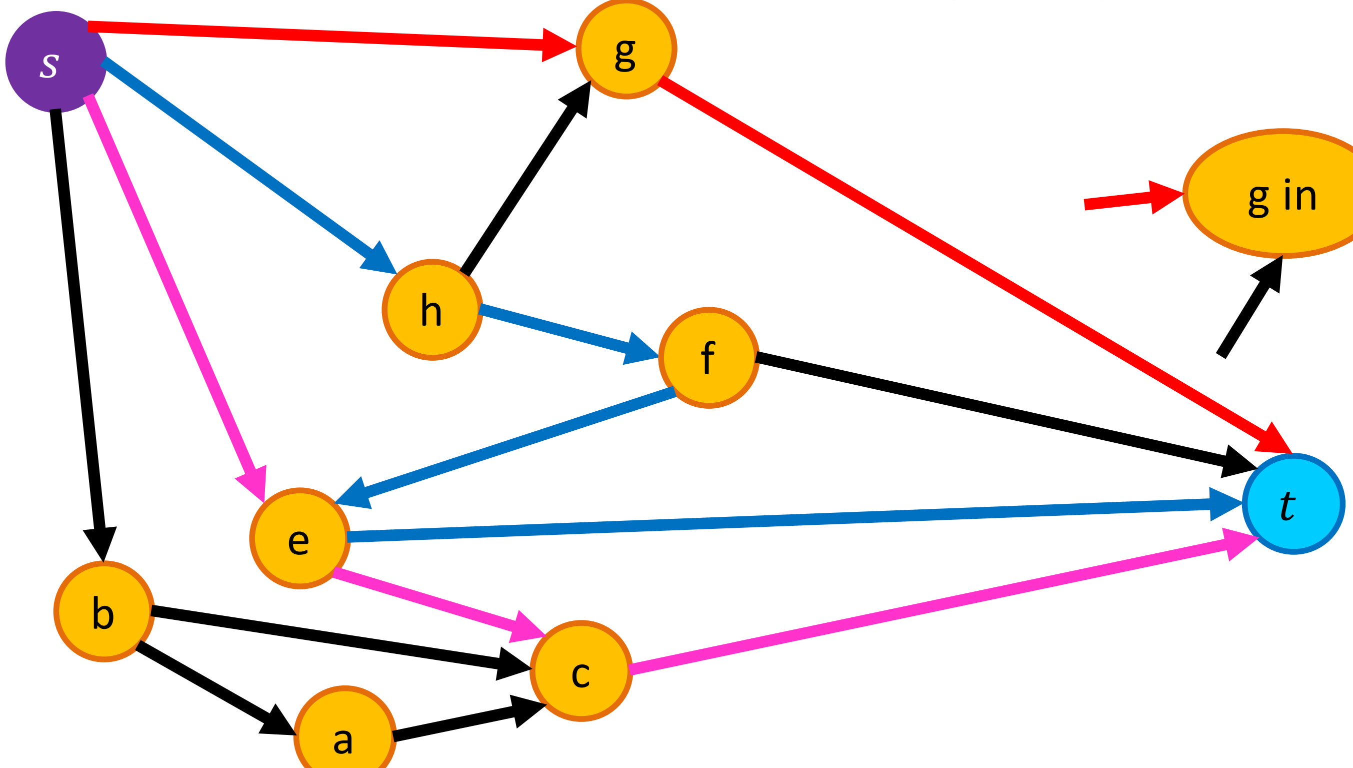
Not a vertex-disjoint path!

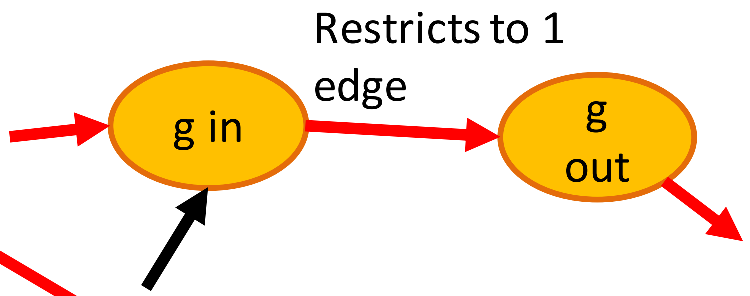
Vertex-Disjoint Paths Algorithm

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

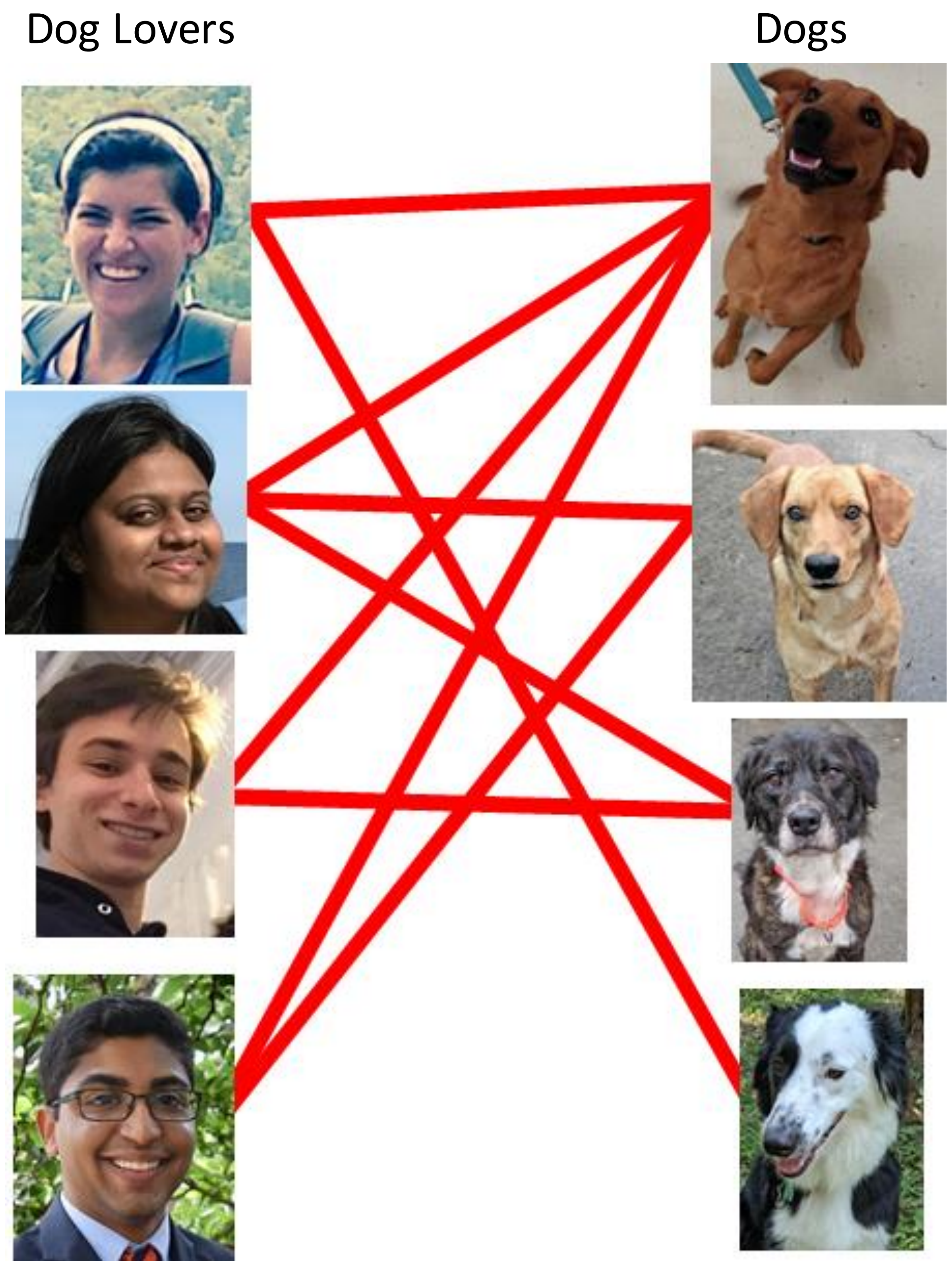
Make two copies of each node, one connected to incoming edges, the other to outgoing edges

Compute Edge-Disjoint paths on new graph

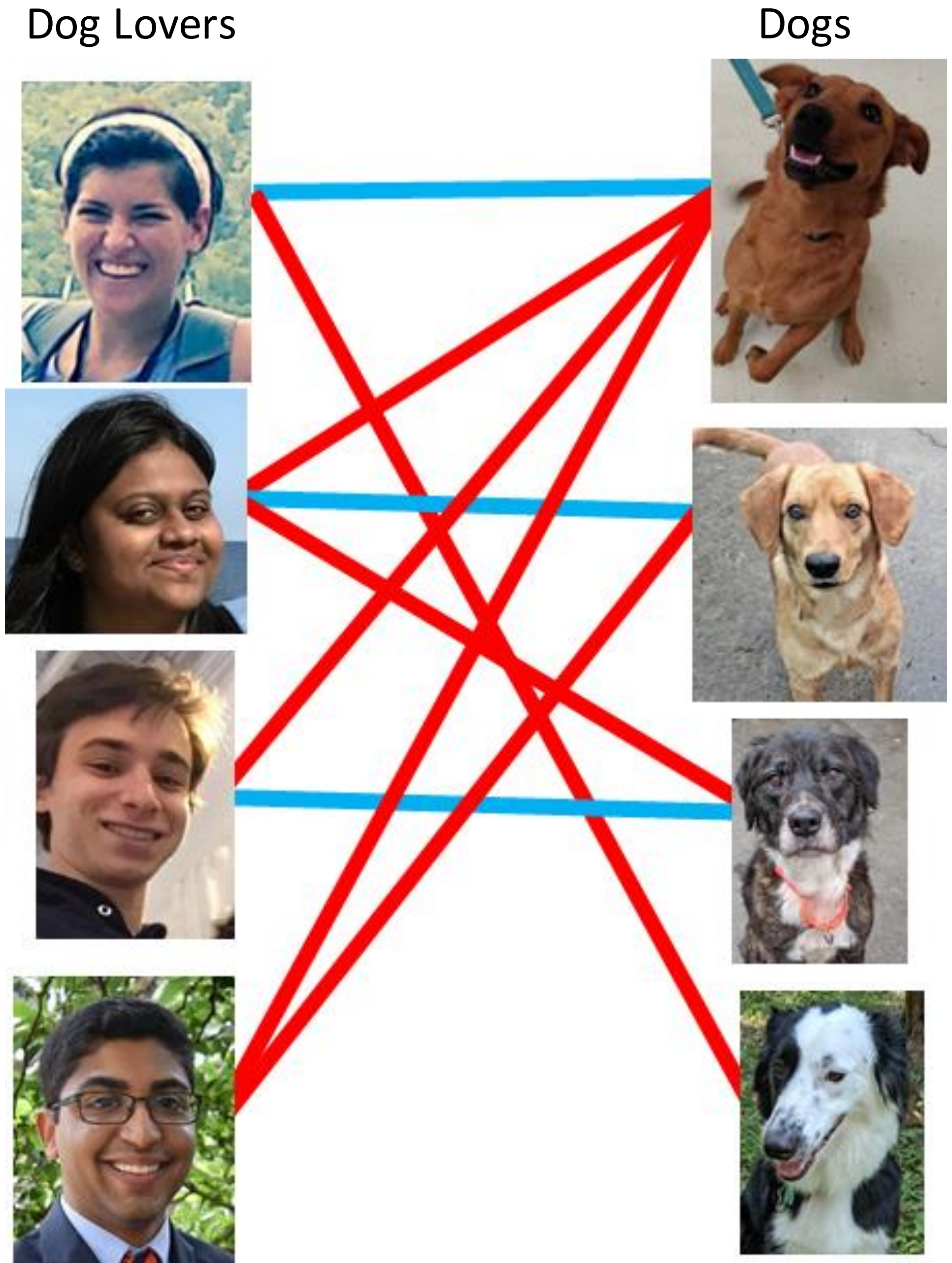




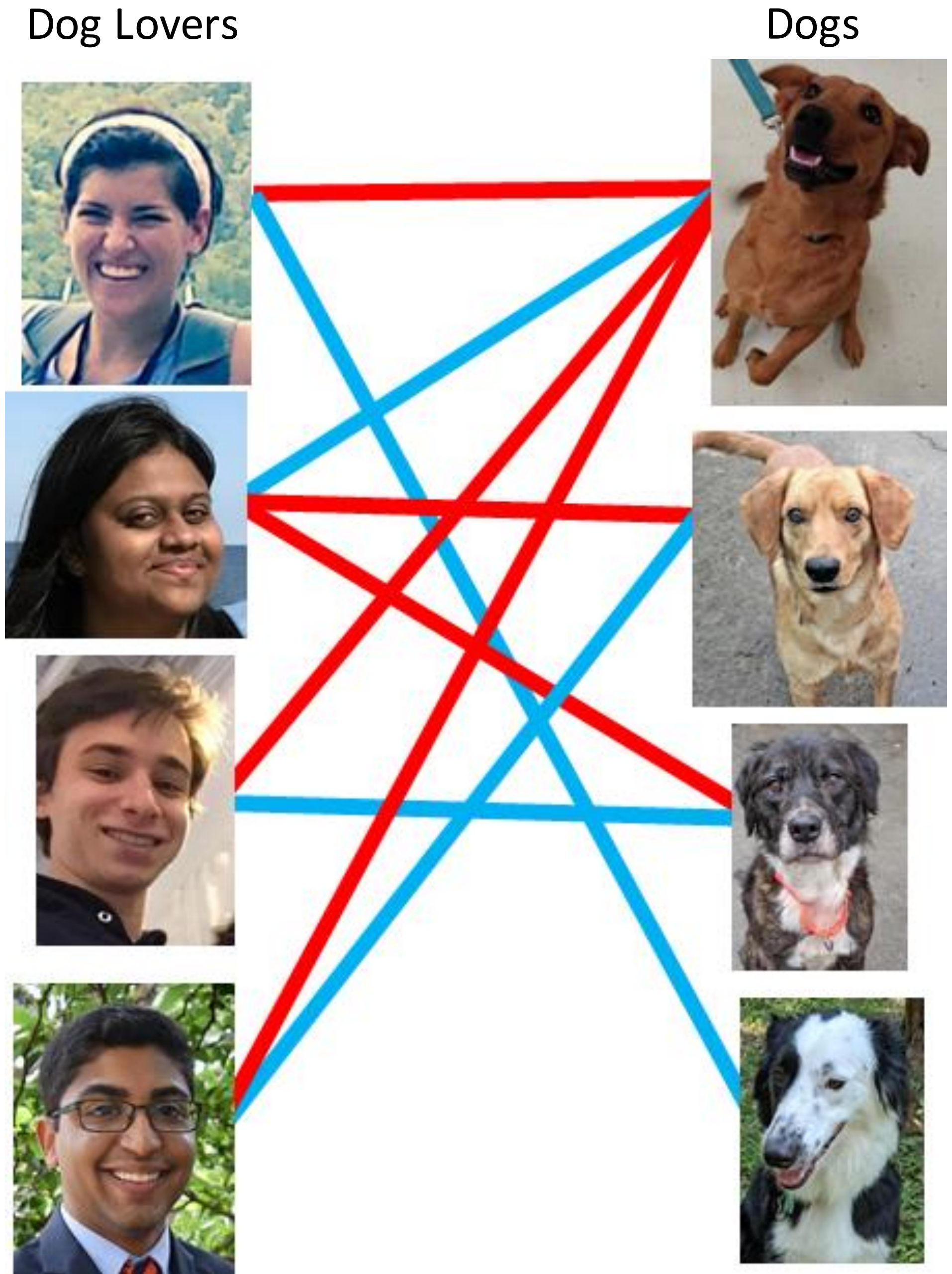
Maximum Bipartite Matching



Maximum Bipartite Matching



Maximum Bipartite Matching



Maximum Bipartite Matching

Given a graph $G = (L, R, E)$

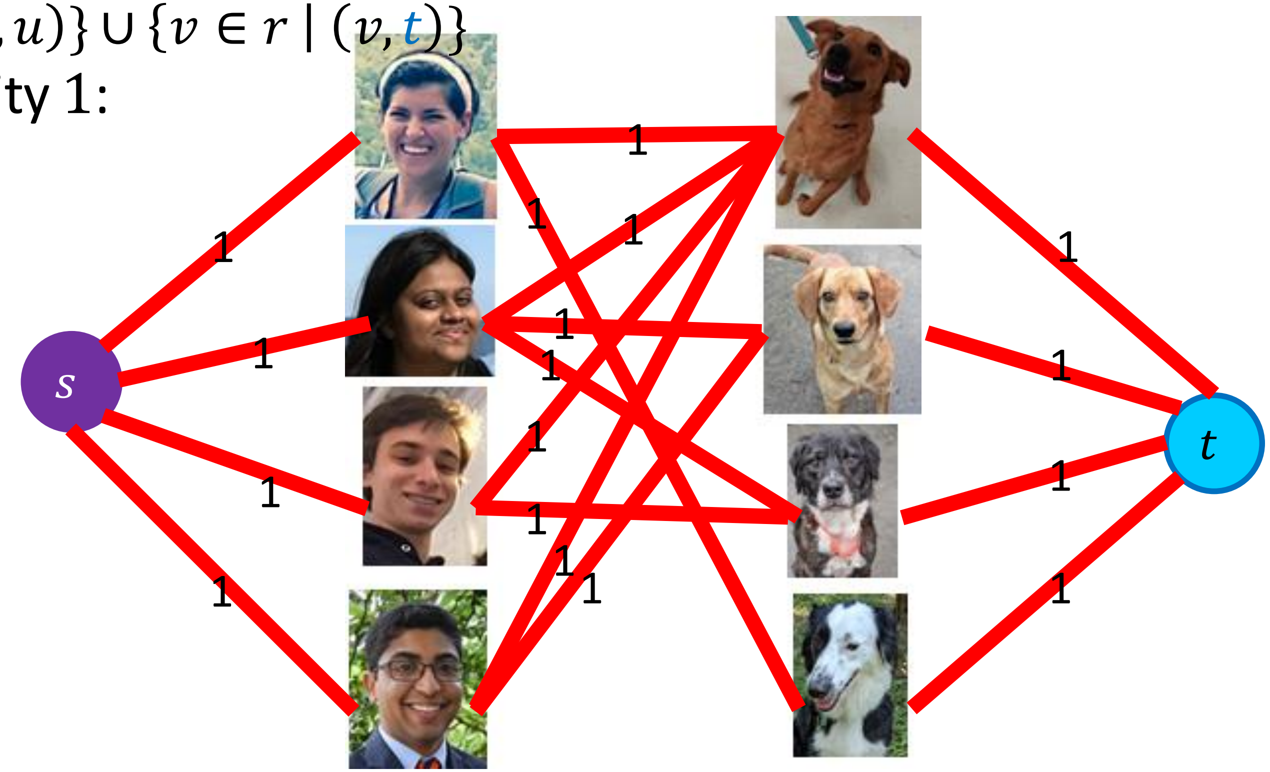
a set of left nodes, right nodes, and edges between left and right

Find the largest set of edges $M \subseteq E$ such that each node $u \in L$ or $v \in R$ is incident to at most one edge.

Maximum Bipartite Matching Using Max Flow

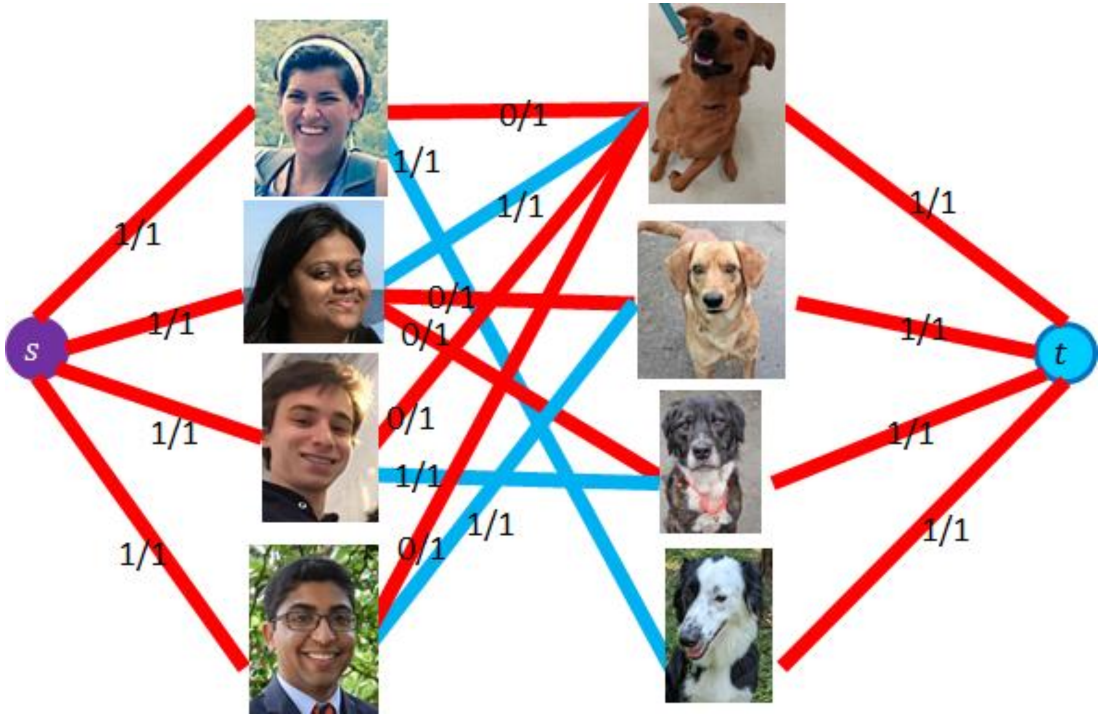
Make $G = (L, R, E)$ a flow network $G' = (V', E')$ by:

- 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:
  - $\forall e \in E', c(e) = 1$



Run Time $\Theta(E \cdot V)$

1. Make G into G' $\Theta(L + R)$
2. Compute Max Flow on G' $\Theta(E \cdot V)$ $|f| \leq L$
3. Return M as all "middle" edges with flow 1 $\Theta(L + R)$



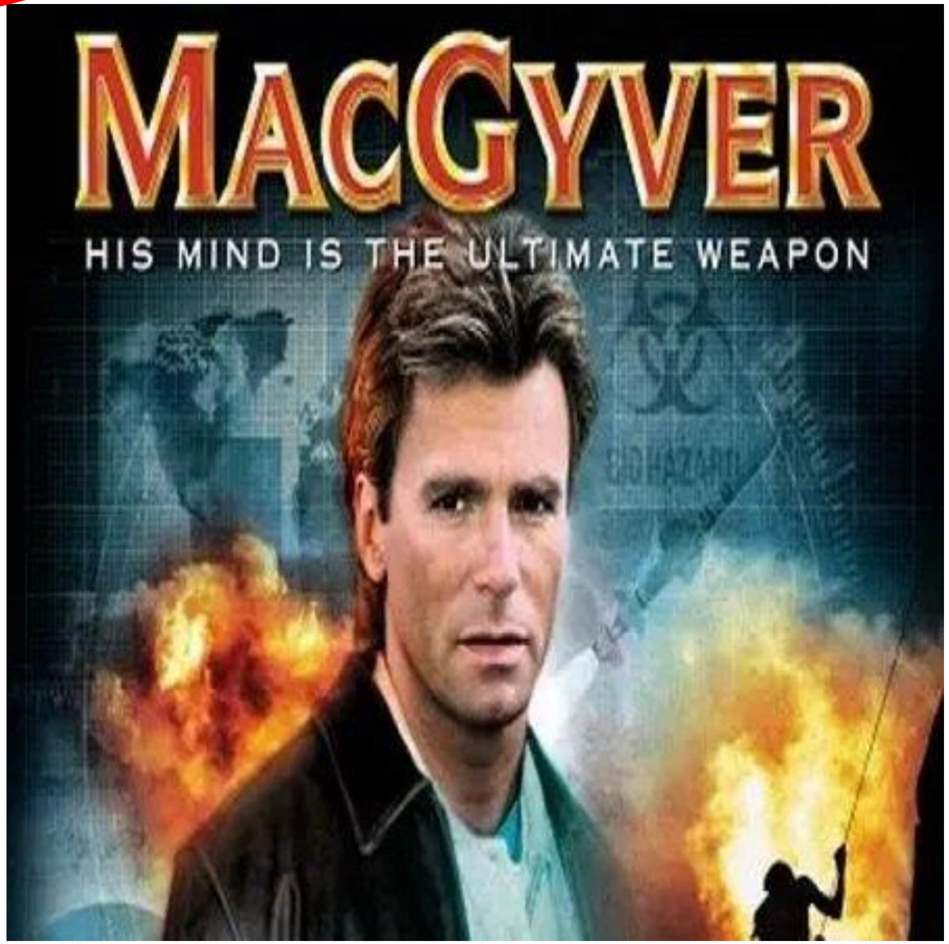
Reductions

- 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

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?

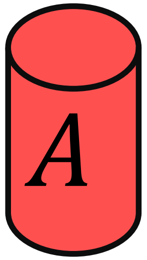
Solution for $\boldsymbol{B}$

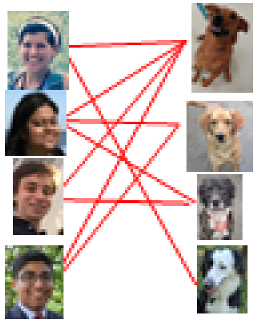
Alcohol, wood, matches

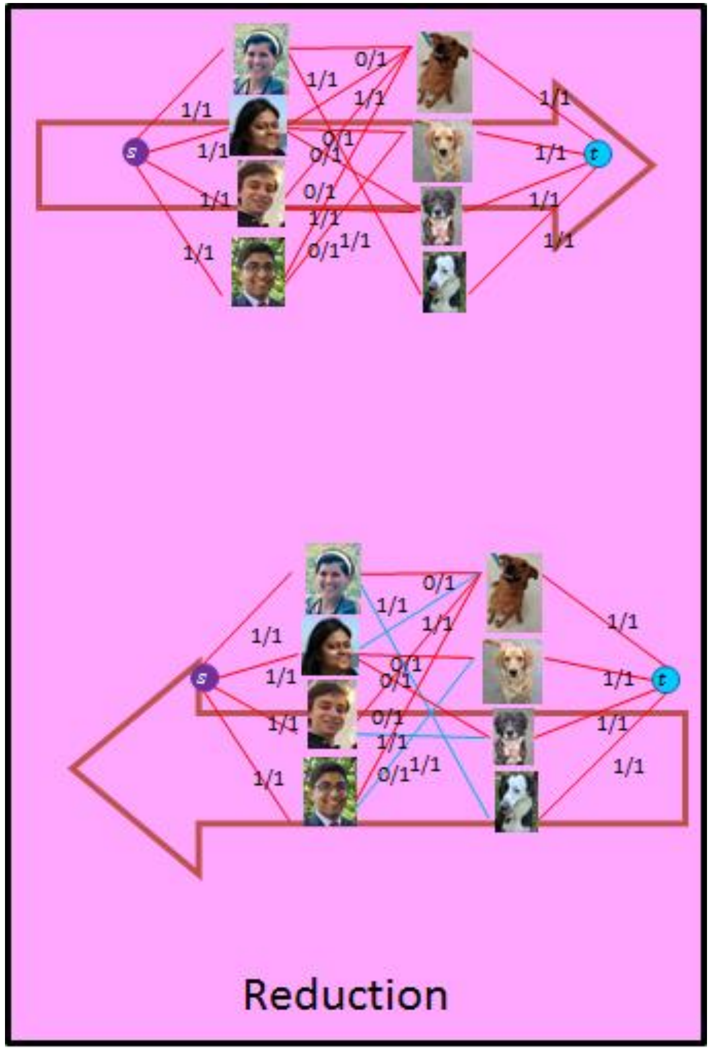
Bipartite Matching Reduction

Problem we don't know how to solve

Bipartite Matching

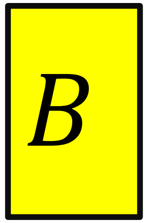


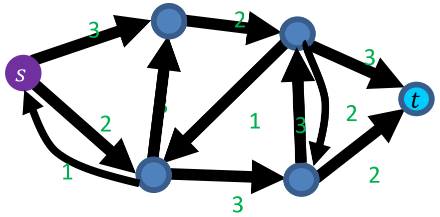




Problem we do know how to solve

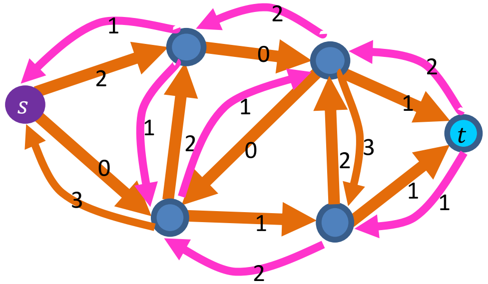
Max Flow



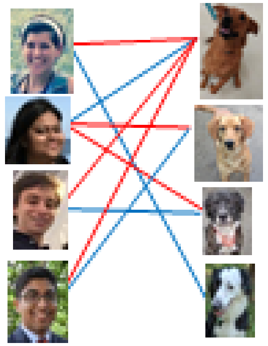


Ford Fulkerson

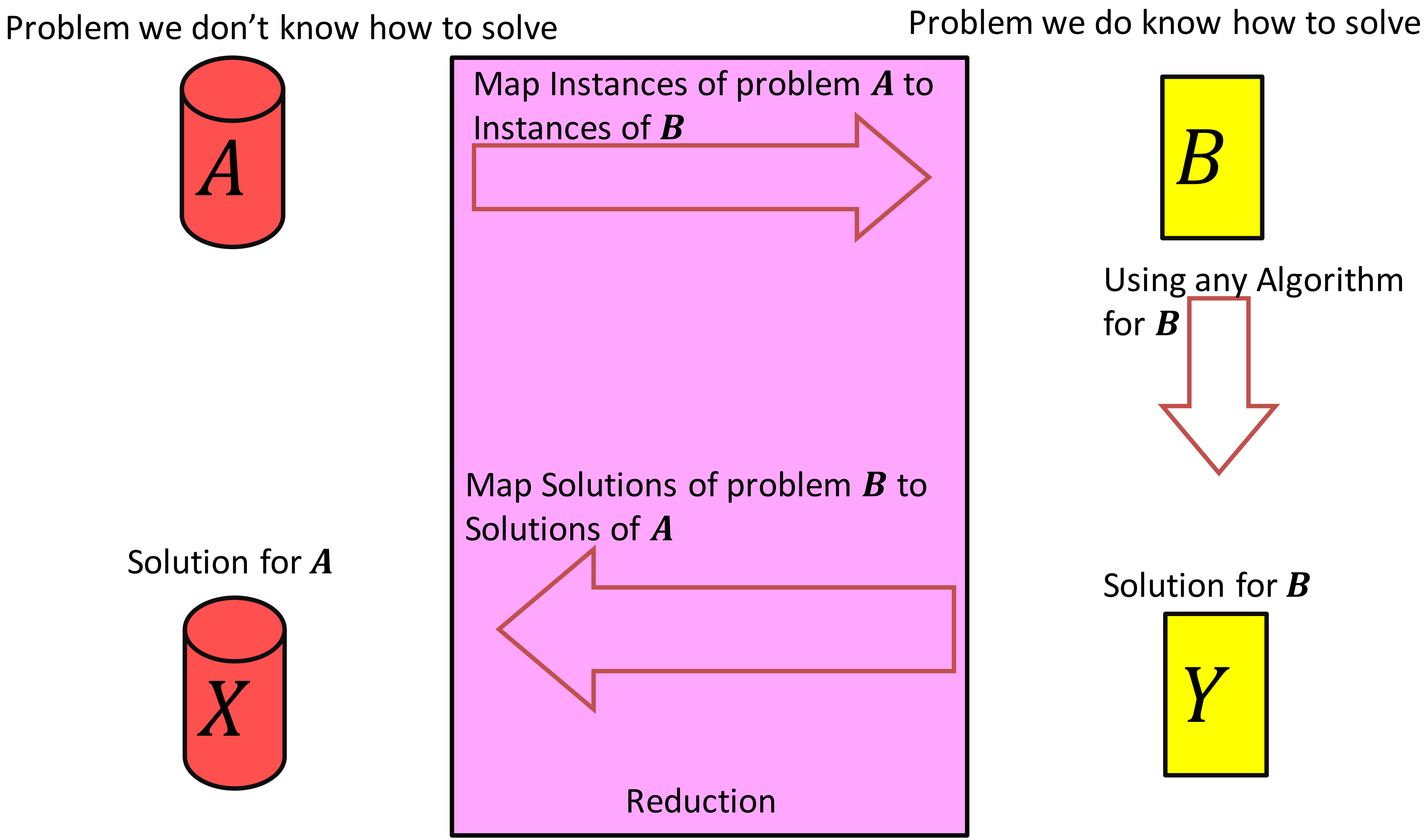
Solution for $\boldsymbol{B}$



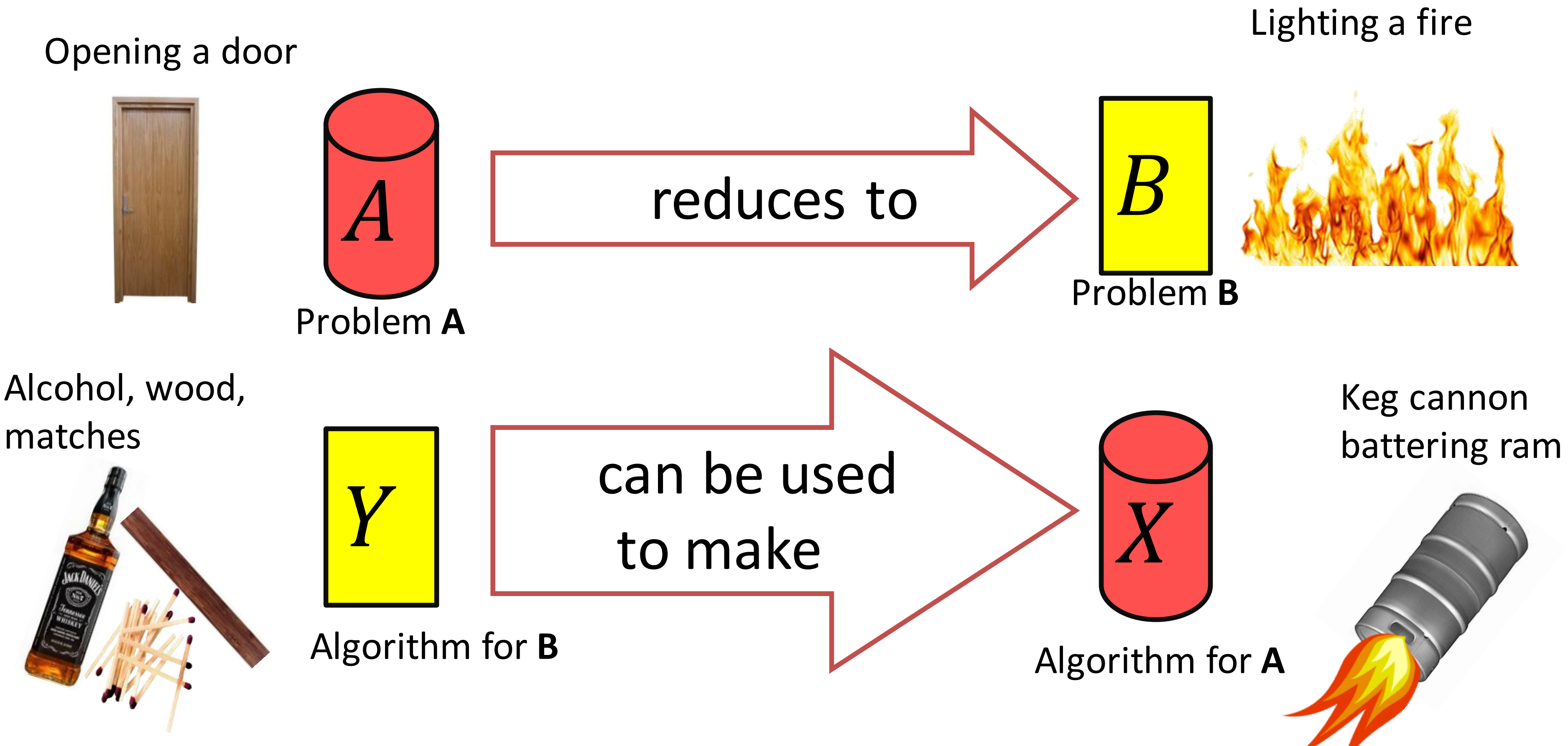
Solution for $\boldsymbol{A}$



In General: Reduction



Worst-case lower-bound Proofs



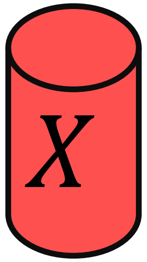
A is not a harder problem than B

$$A \le B$$

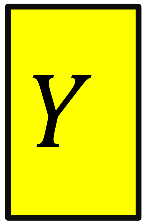
The name "reduces" is confusing: it is in the *opposite* direction of the making

Proof of Lower Bound by Reduction

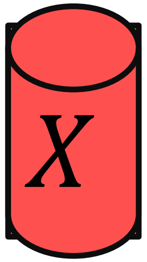
<u>To Show: Y is slow</u>



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



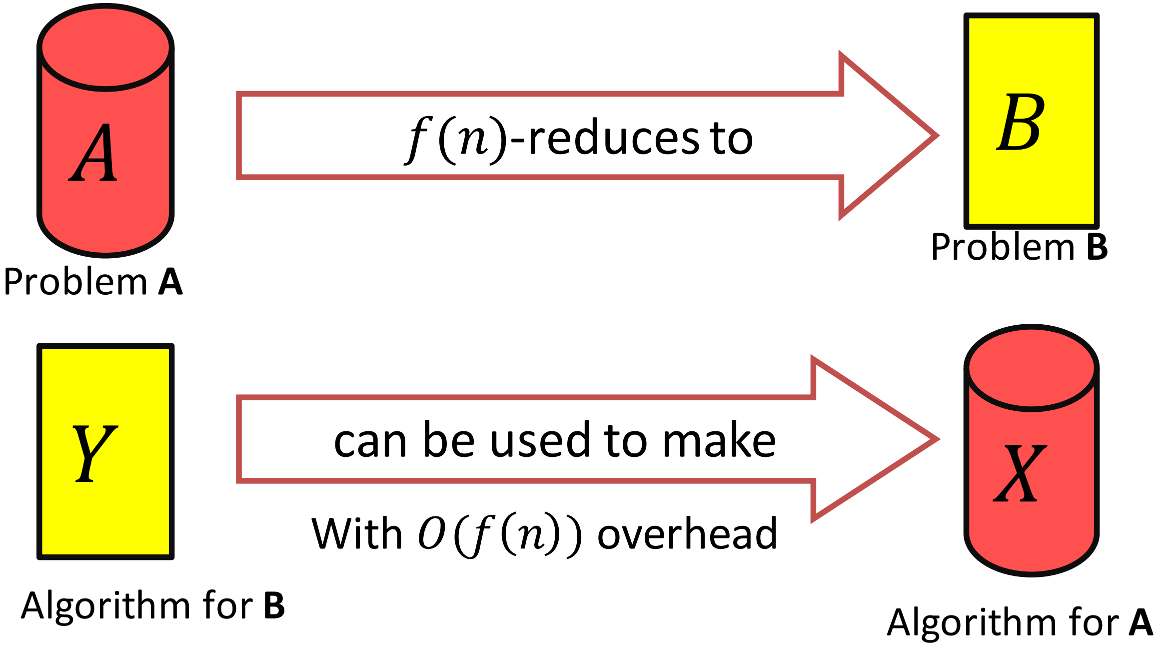
2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)



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

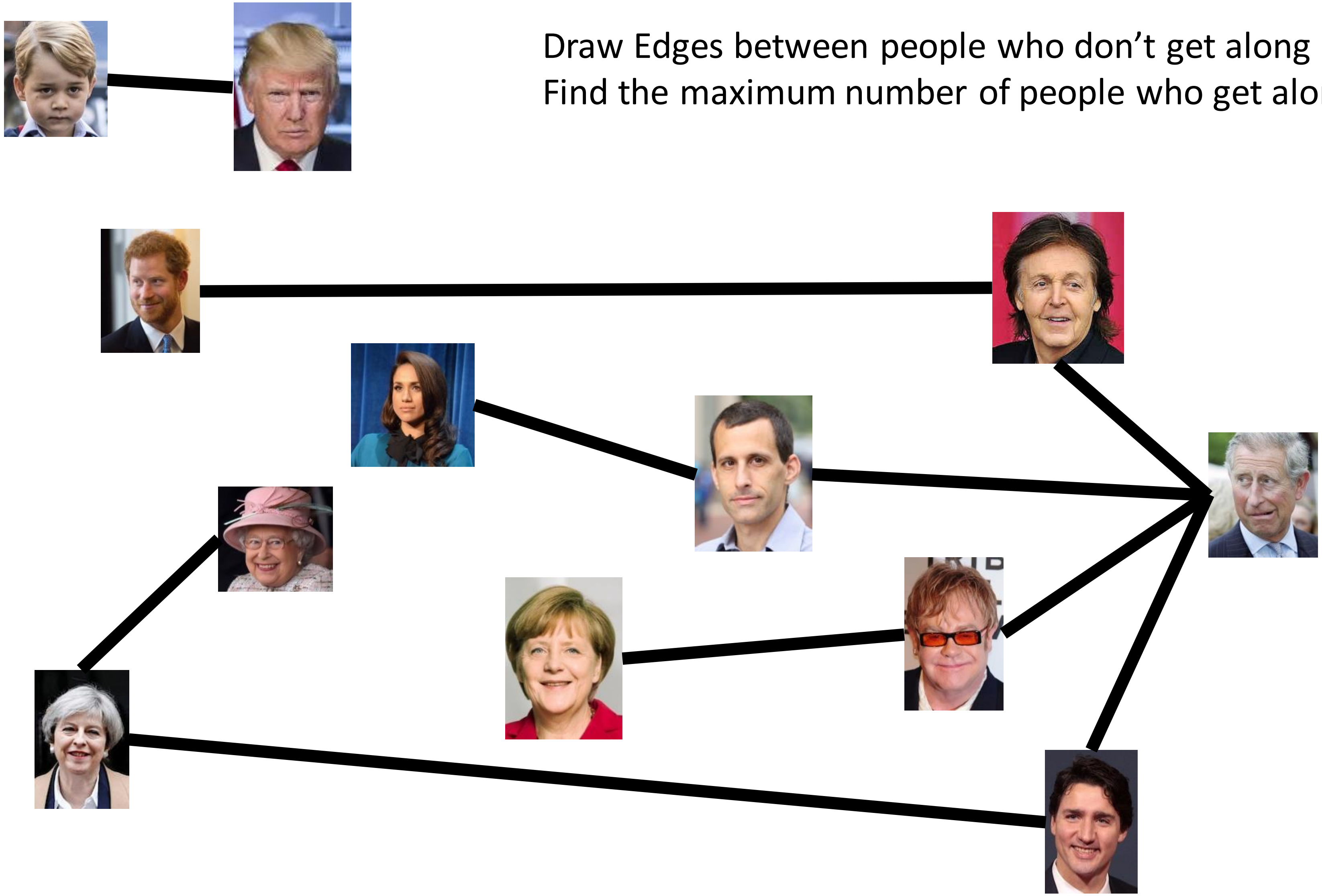
$$A \leq B$$

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

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

Party Problem

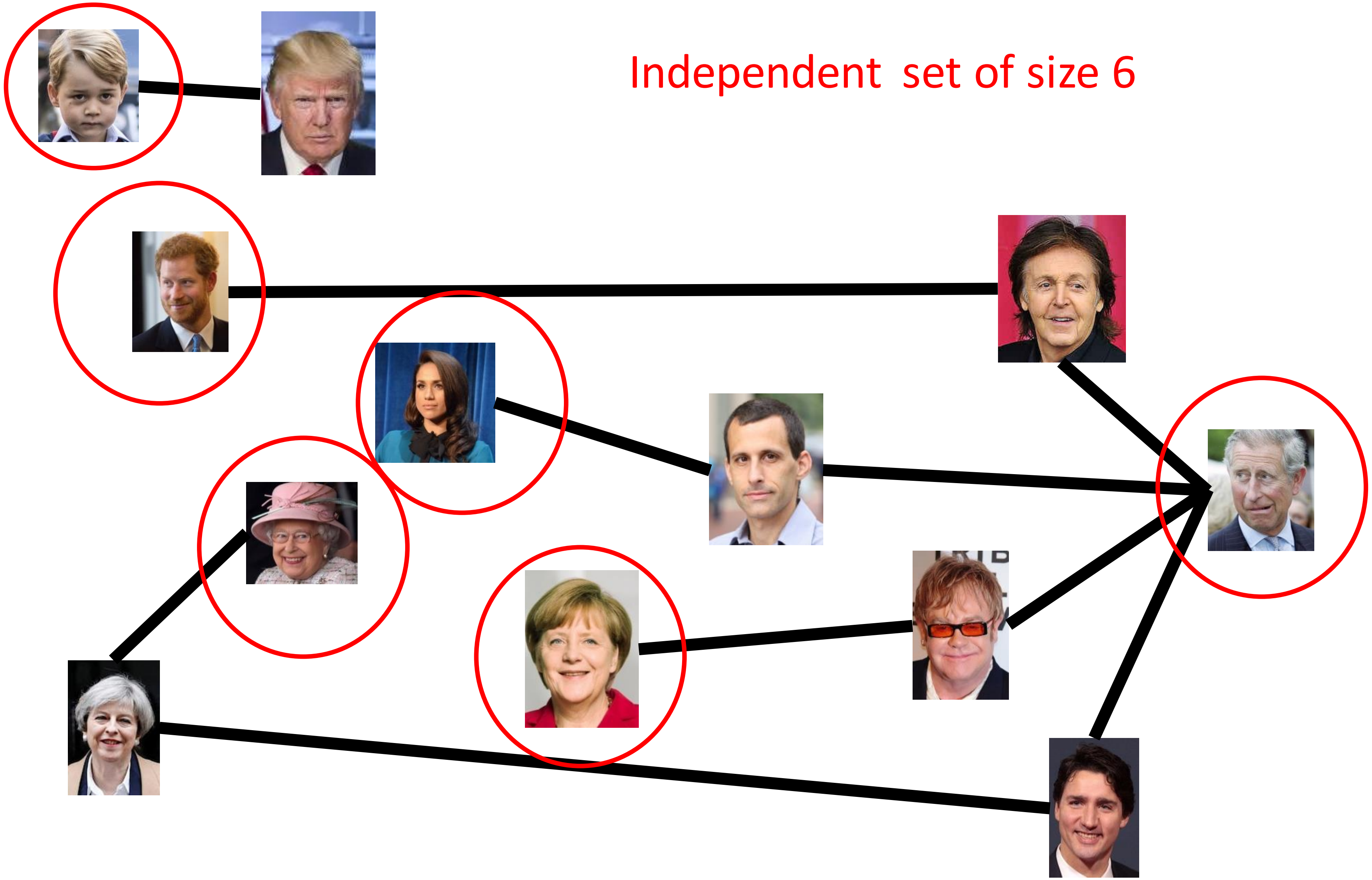
Draw Edges between people who don't get along
Find the maximum number of people who get along



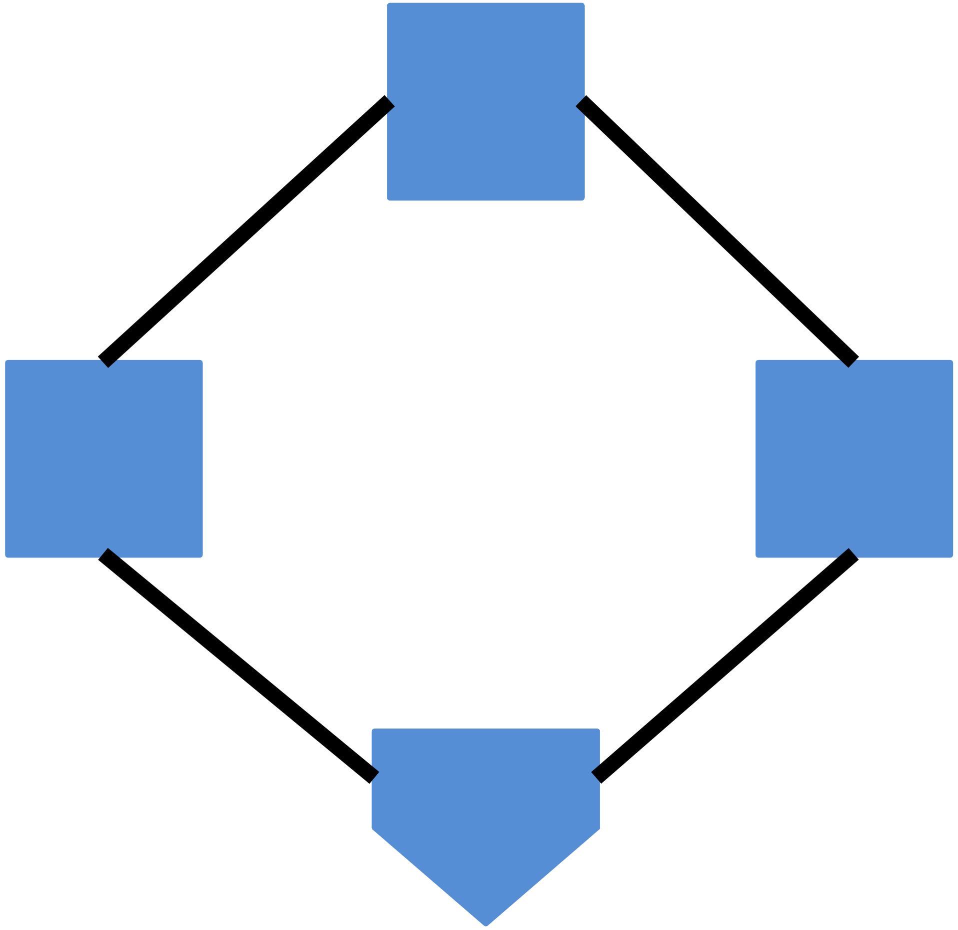
Maximum Independent Set

- 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

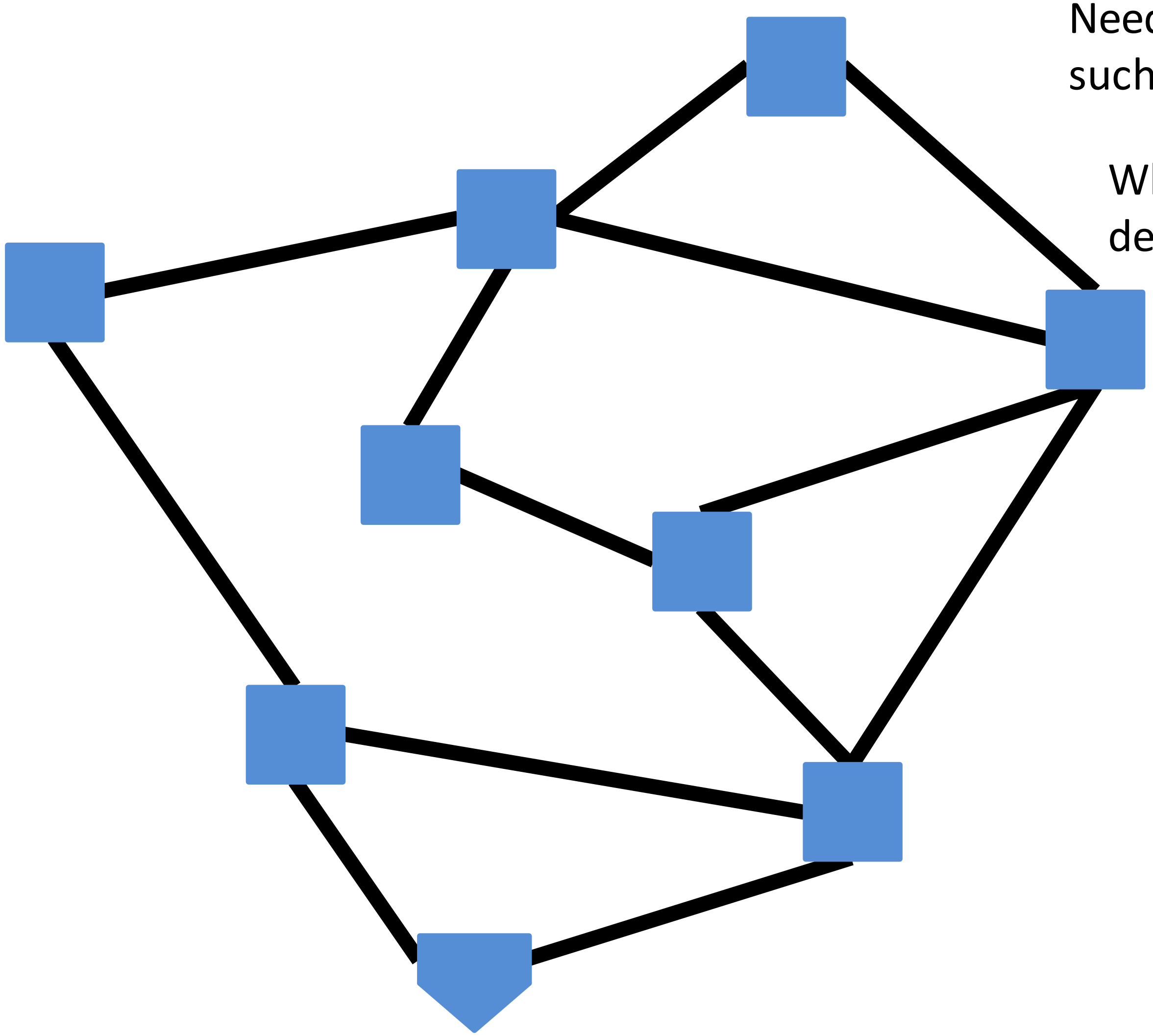
Example



Generalized Baseball



Generalized Baseball



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

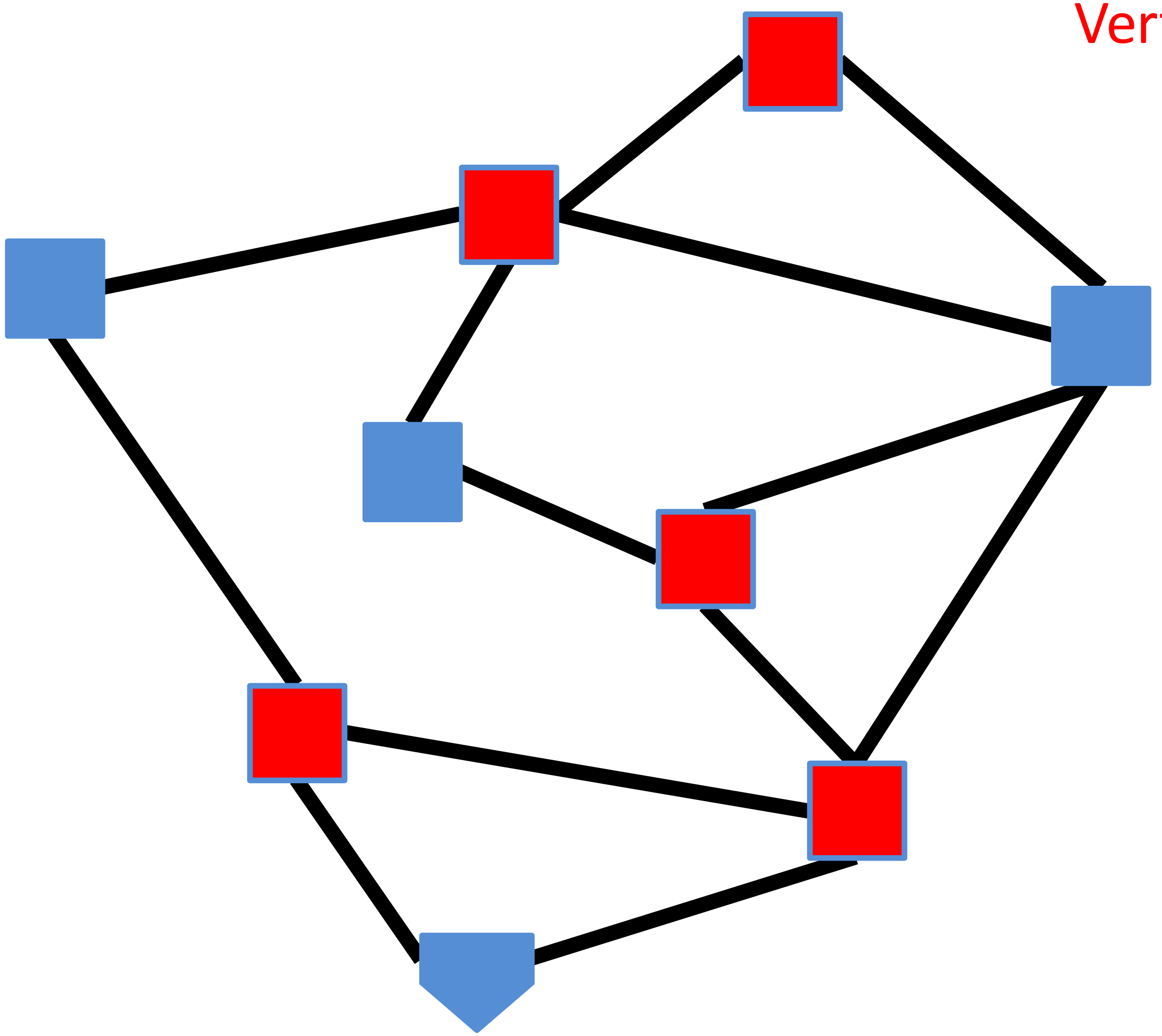
What's the fewest number of defenders needed?

Minimum Vertex Cover

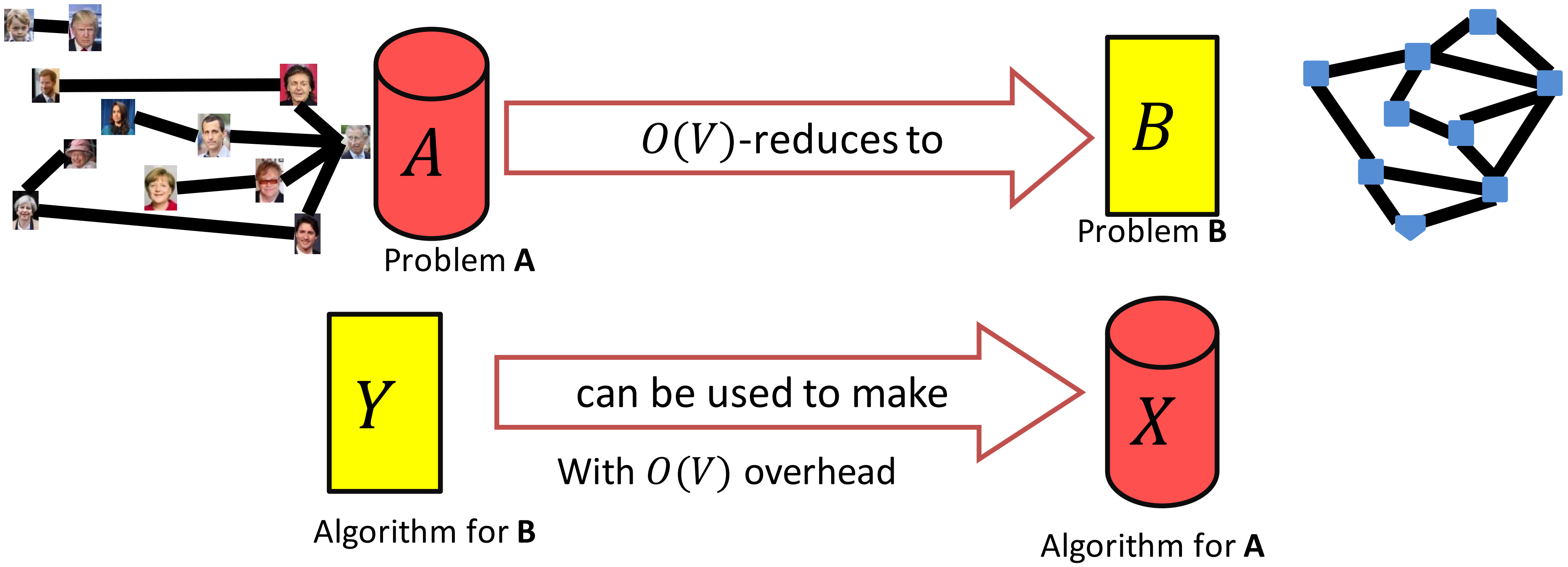
- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

Example

Vertex cover of size 5



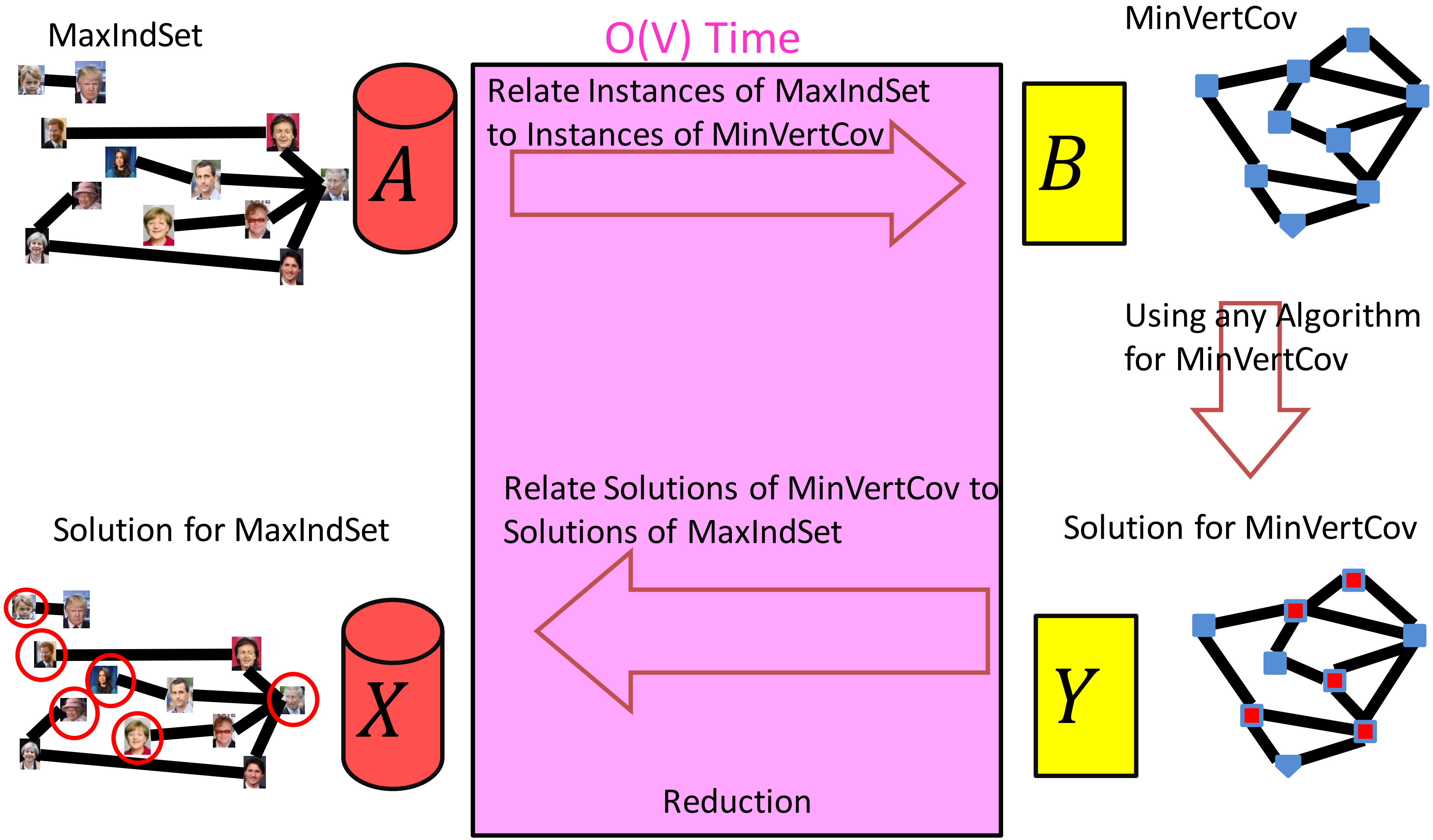
MaxIndSet$\leq_V$MinVertCov



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

$$A \leq_V B$$

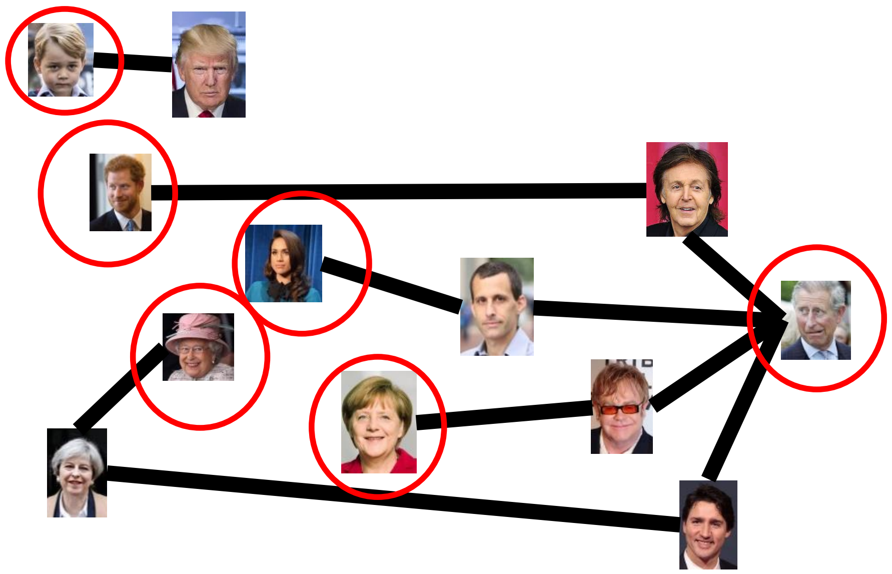
We need to build this Reduction



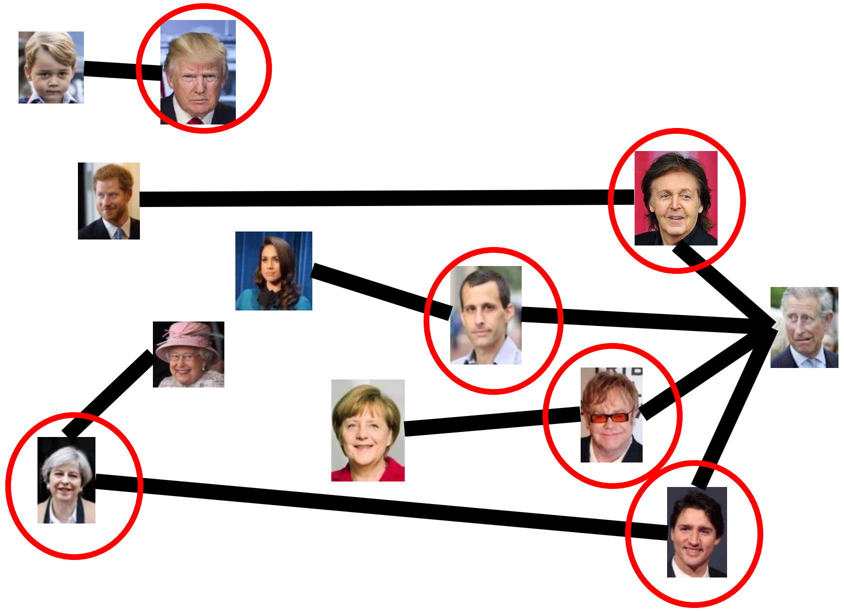
Reduction Idea

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

Independent Set

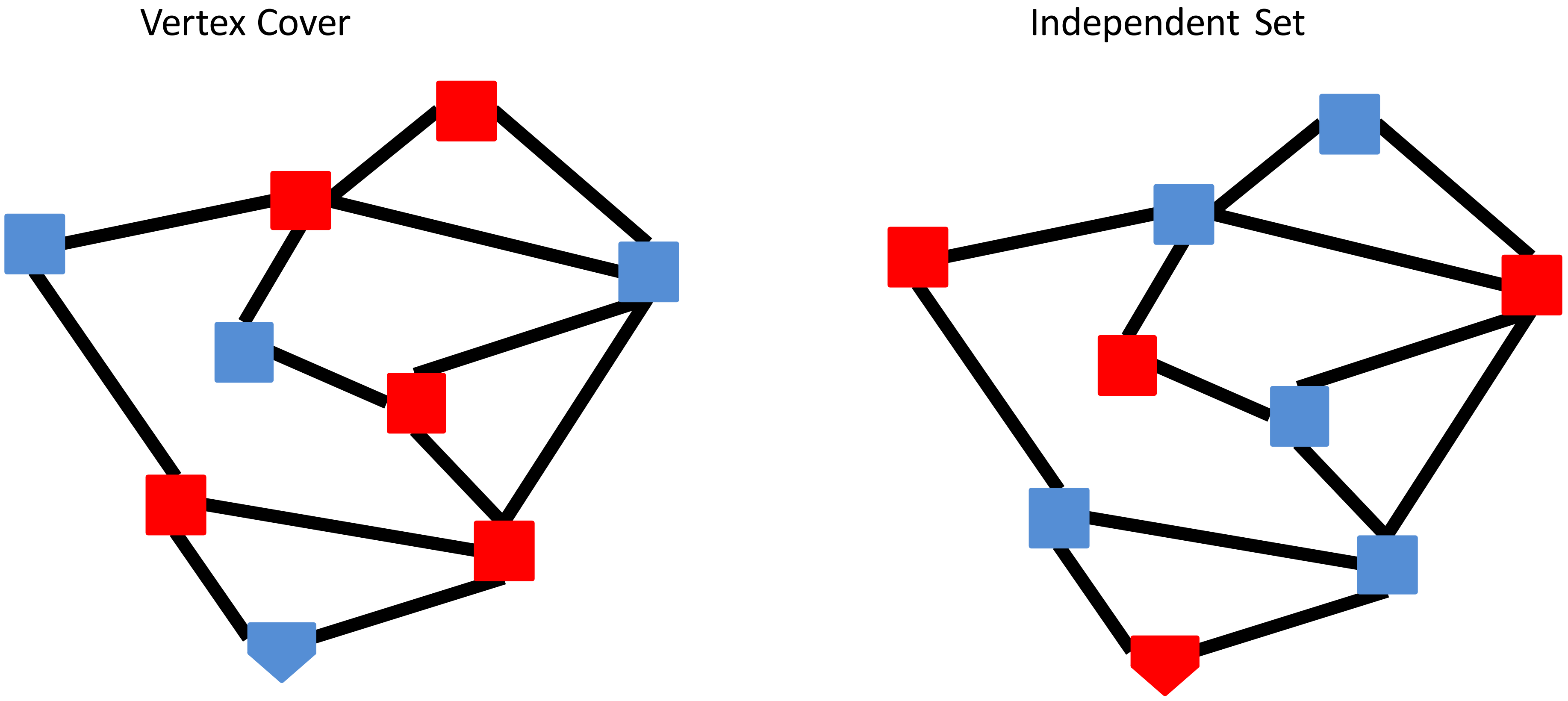


Vertex Cover



Reduction Idea

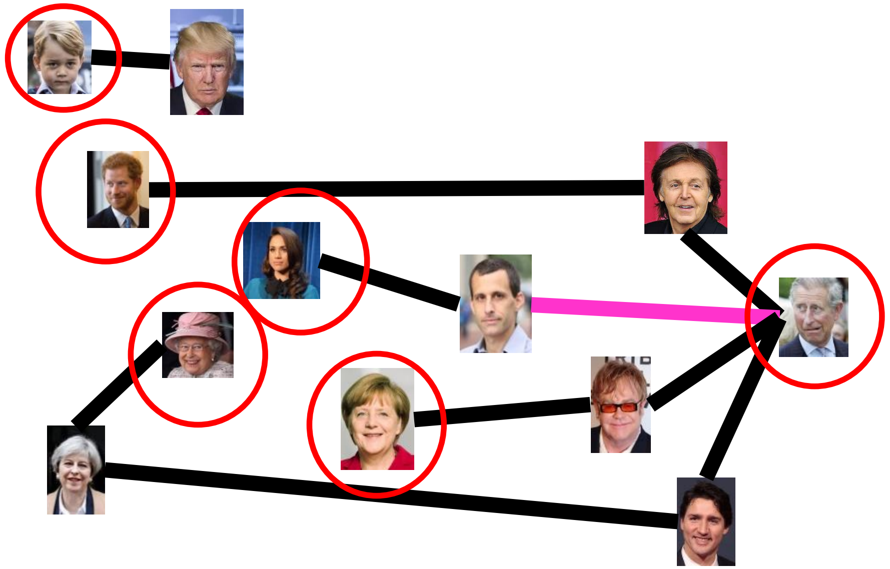
S is an independent set of G iff $V - S$ is a vertex cover of G



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 $(x, y) \in E$

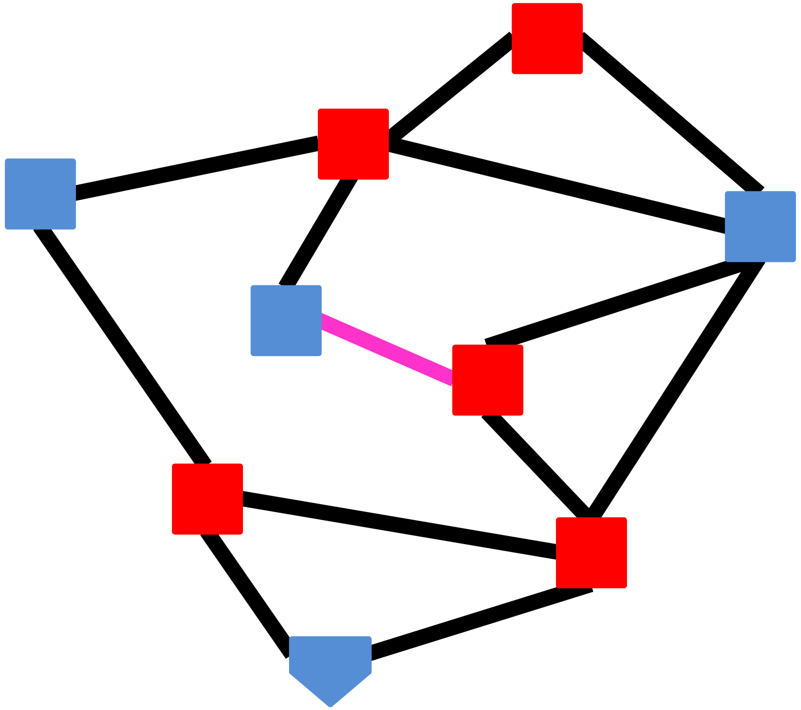
If $x \in S$ then $y \notin S$, because o.w. S would not be an independent set

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



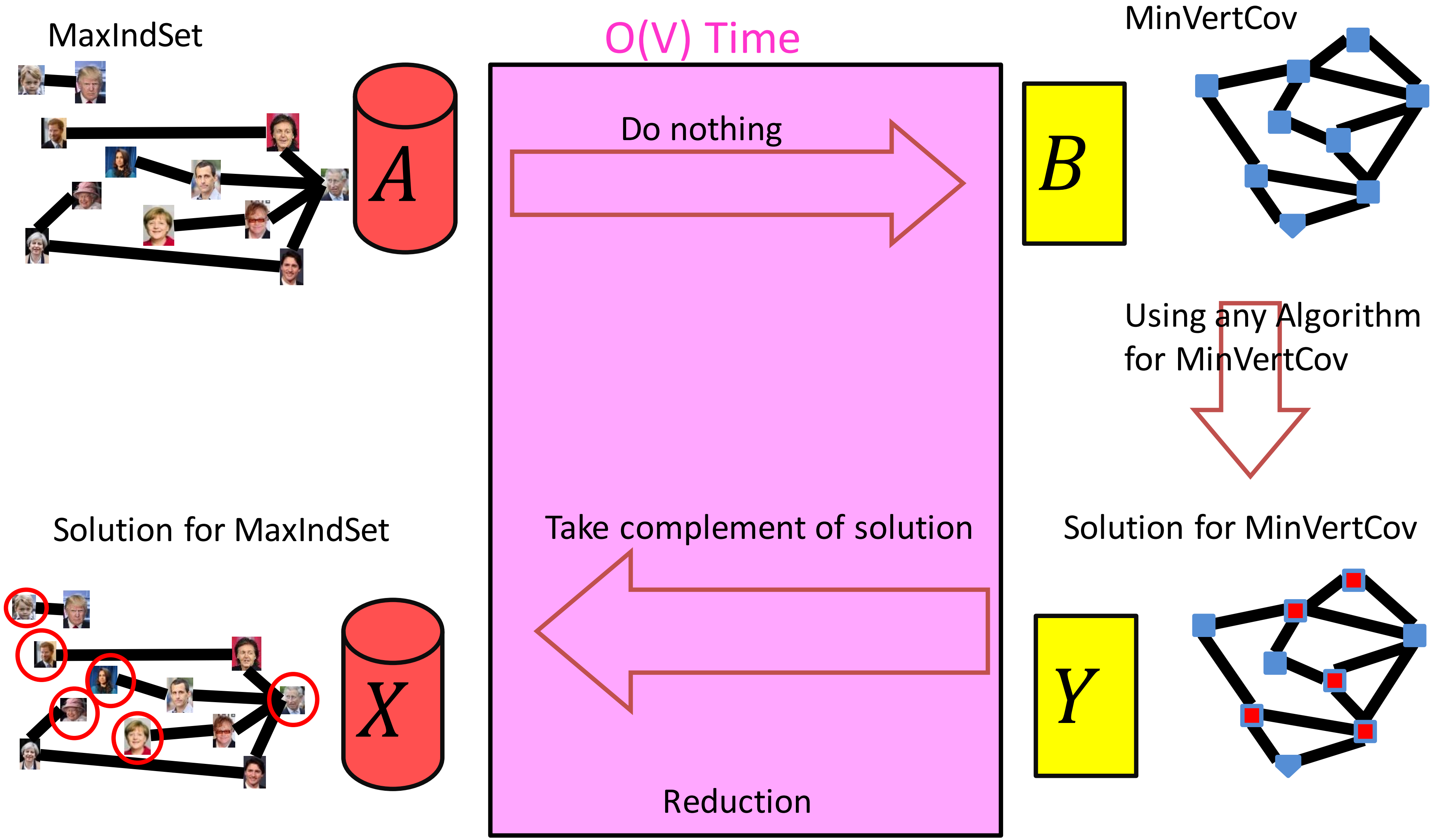
Consider any edge $(x, y) \in E$

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

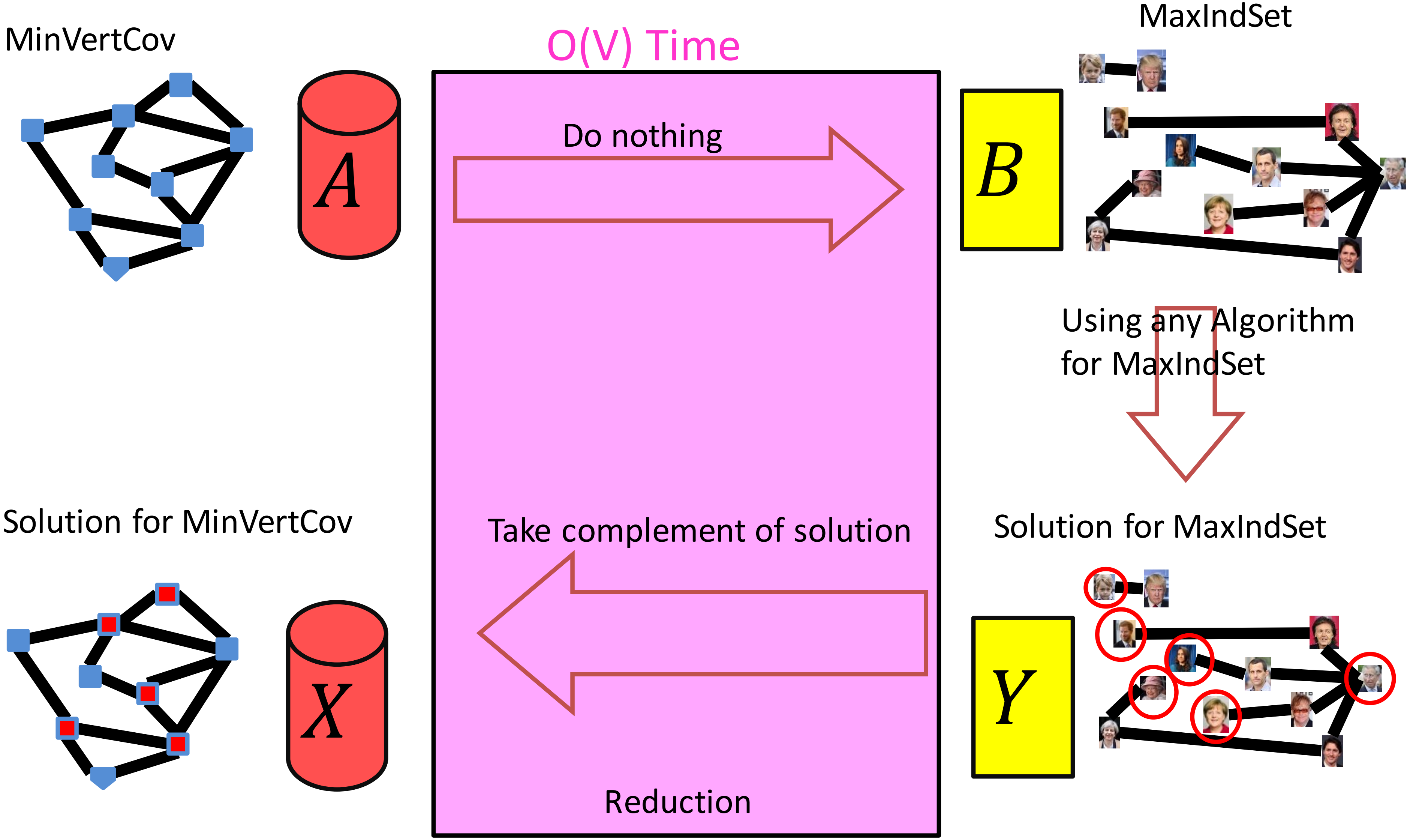
Therefore x and y are not both in S,

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

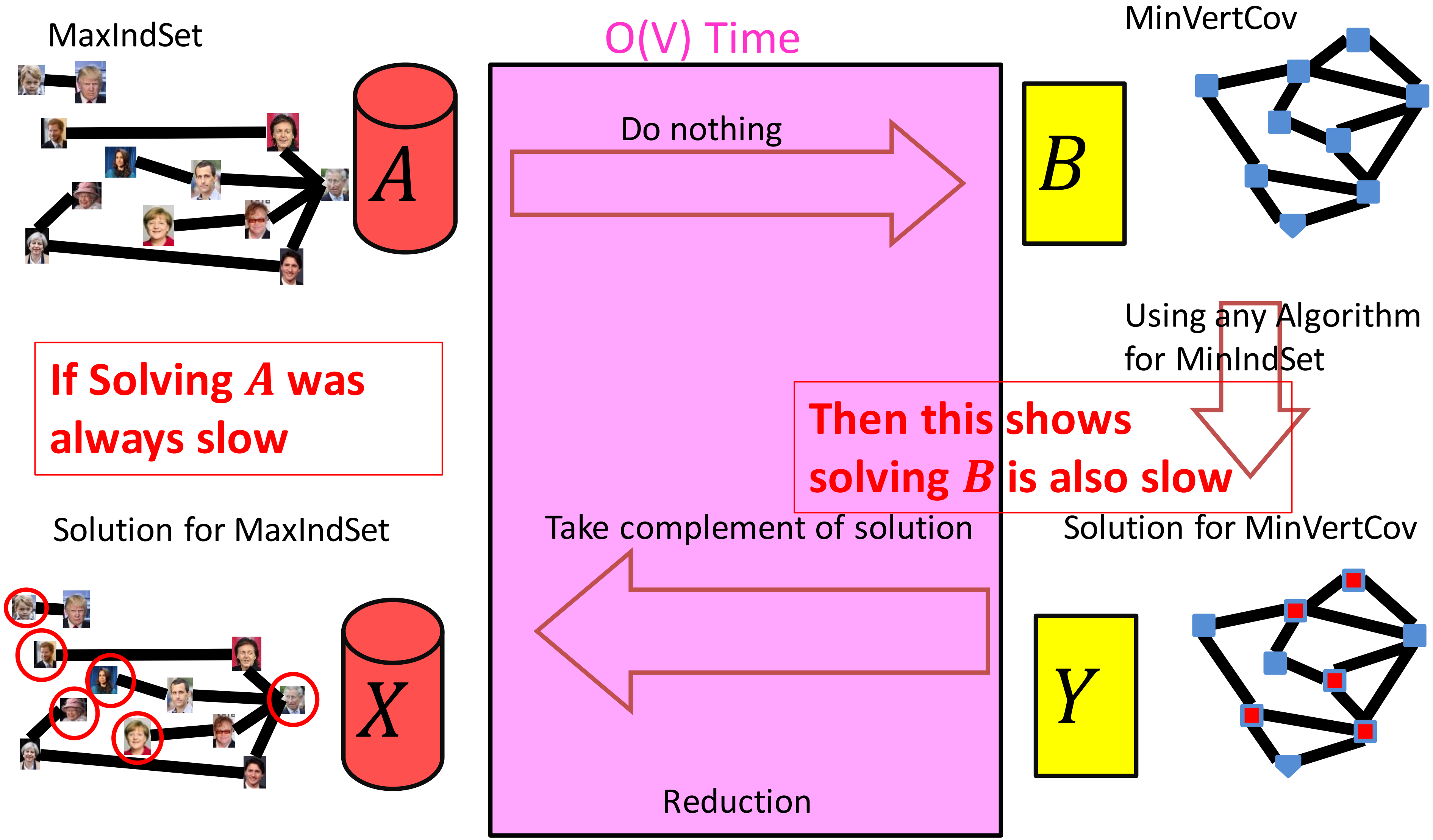
MaxVertCov V-Time Reducable to MinIndSet



MaxIndSet V-Time Reducable to MinVertCov



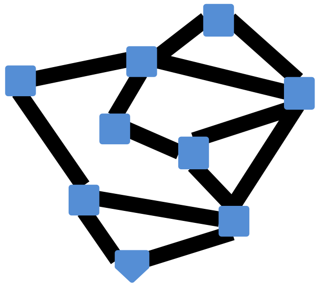
Corollary



MaxIndSet
O(V) Time
MinVertCov
A
Do nothing
B
Using any Algorithm
for MinIndSet
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



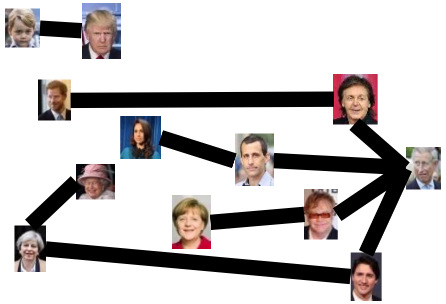
A

O(V) Time

Do nothing

B

MaxIndSet



Using any Algorithm
for MaxVertCov

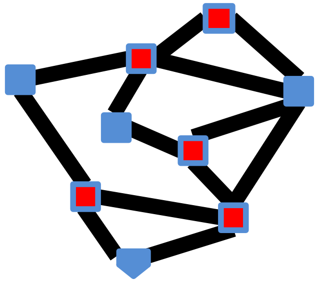
If Solving A was always slow

Then this shows solving B is also slow

Solution for MinVertCov

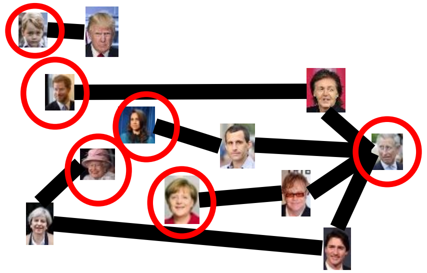
Take complement of solution

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!