CS 4102: Algorithms

Lecture 23: Bipartite Matching

David Wu

Fall 2019

Today’s Keywords

Edge-Disjoint Paths

Vertex-Disjoint Paths

Bipartite Matching

Reductions

CLRS Readings: Chapter 26, 34

Homework

HW8 out **today**, due **Thursday, November 21, 11pm**

- Programming assignment (Python or Java)
- Graph algorithms

HW9, **HW10C** out Thursday, November 21 (due Thursday, December 5)

- Graphs, Reductions
- Written (LaTeX)

Final Exam

Monday, December 9, 7pm in Olsson 120

- Practice exam coming next week
- Review session likely the weekend before

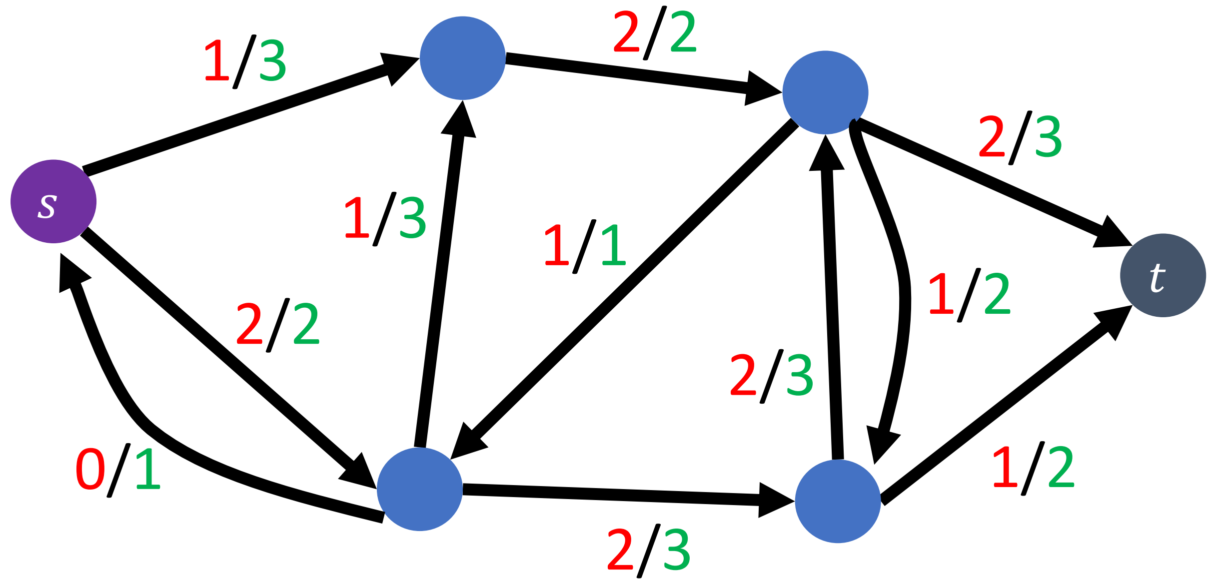
Exam conflicts: Will email out a sign-up form for alternative exam time

- Alternative exam only for student with an conflicting exam at the <u>same</u> time

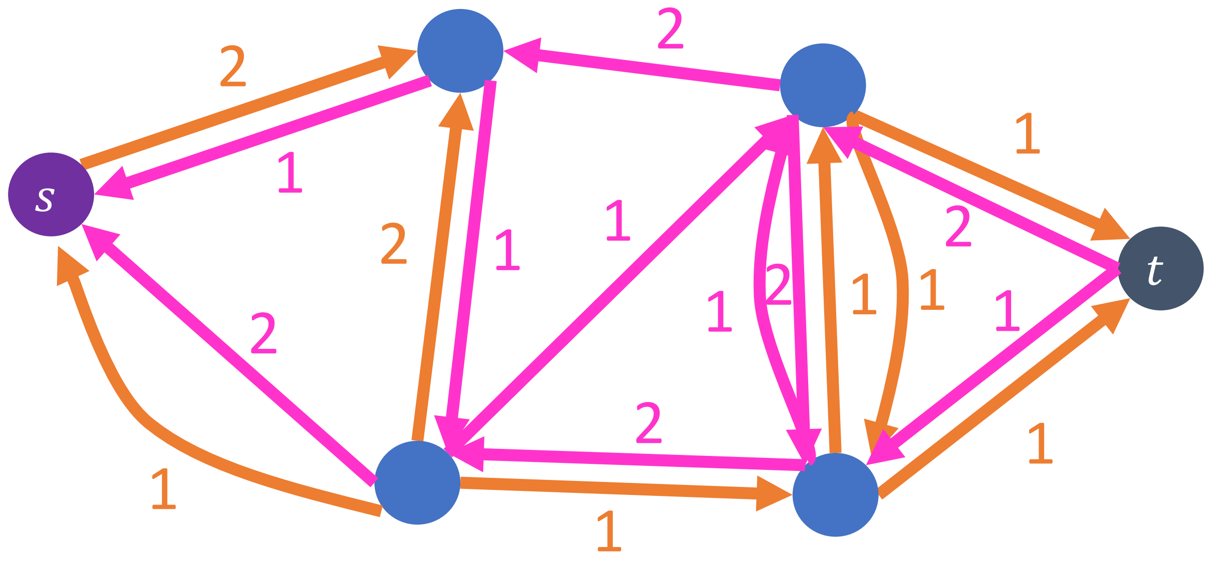
Review: Max Flow and Residual Graphs

Given a flow f in graph G, the residual graph G_f models additional flow that is possible

- Forward edge for each edge in G with weight set to remaining capacity $c(e) - f(e)$
  - Models additional flow that can be sent along the edge
- Backward edge by flipping each edge e in G with weight set to flow $f(e)$
  - Models amount of flow that can be removed from the edge



Flow f in G



Residual graph G_f

Max-Flow Min-Cut Theorem

Let f be a flow in a graph G

f is a maximum flow in G $\Rightarrow$ there are no augmenting paths in the residual graph G_f $\Rightarrow$ there exists a cut (S, T) of G where $|f| = \|S, T\|$

Statements are equivalent!

Implications:

- **Correctness of Ford-Fulkerson:** Ford-Fulkerson terminates when there are no more augmenting paths in the residual graph G_f, which means that f is a maximum flow
- **Max-flow min-cut duality:** the maximum flow in a network coincides with the minimum cut of the graph ($\max_f |f| = \min_{S,T} \|S, T\|$)

Warm-Up: Flow Integrality Theorem

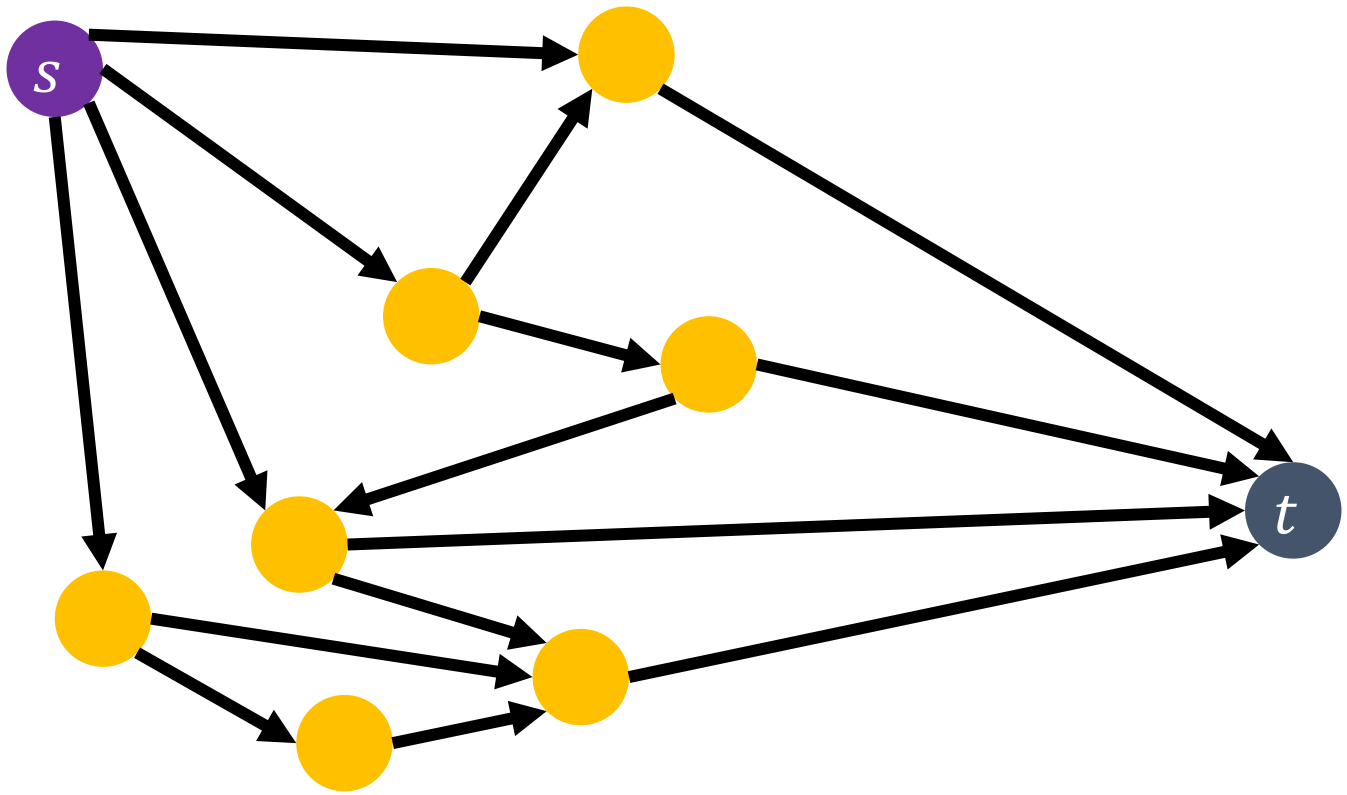
Theorem: If G is a flow graph with integer capacities, then there is a maximum flow that assigns integer flows to every edge

Proof: Follows by correctness of Ford-Fulkerson:

- If the graph G has integer capacities, then the initial residual graph will only have integer weights
- Each augmentation step in Ford-Fulkerson increases the flow along an edge by an integer amount (specifically, the least-weight edge in the augmenting path)
- The final flow output by Ford-Fulkerson uses integer flow along each edge, and by correctness of Ford-Fulkerson, this flow is maximal

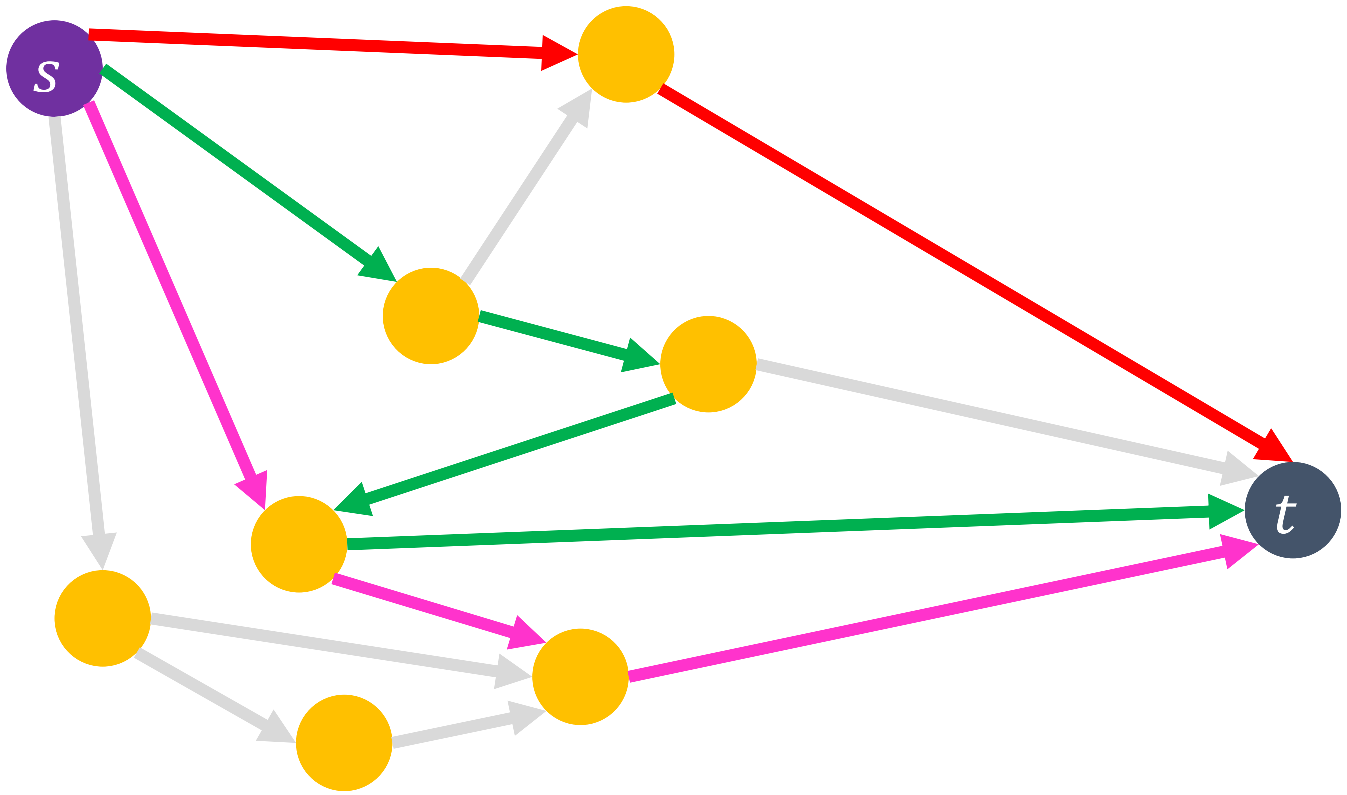
Edge-Disjoint Paths

Problem: 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



Edge-Disjoint Paths

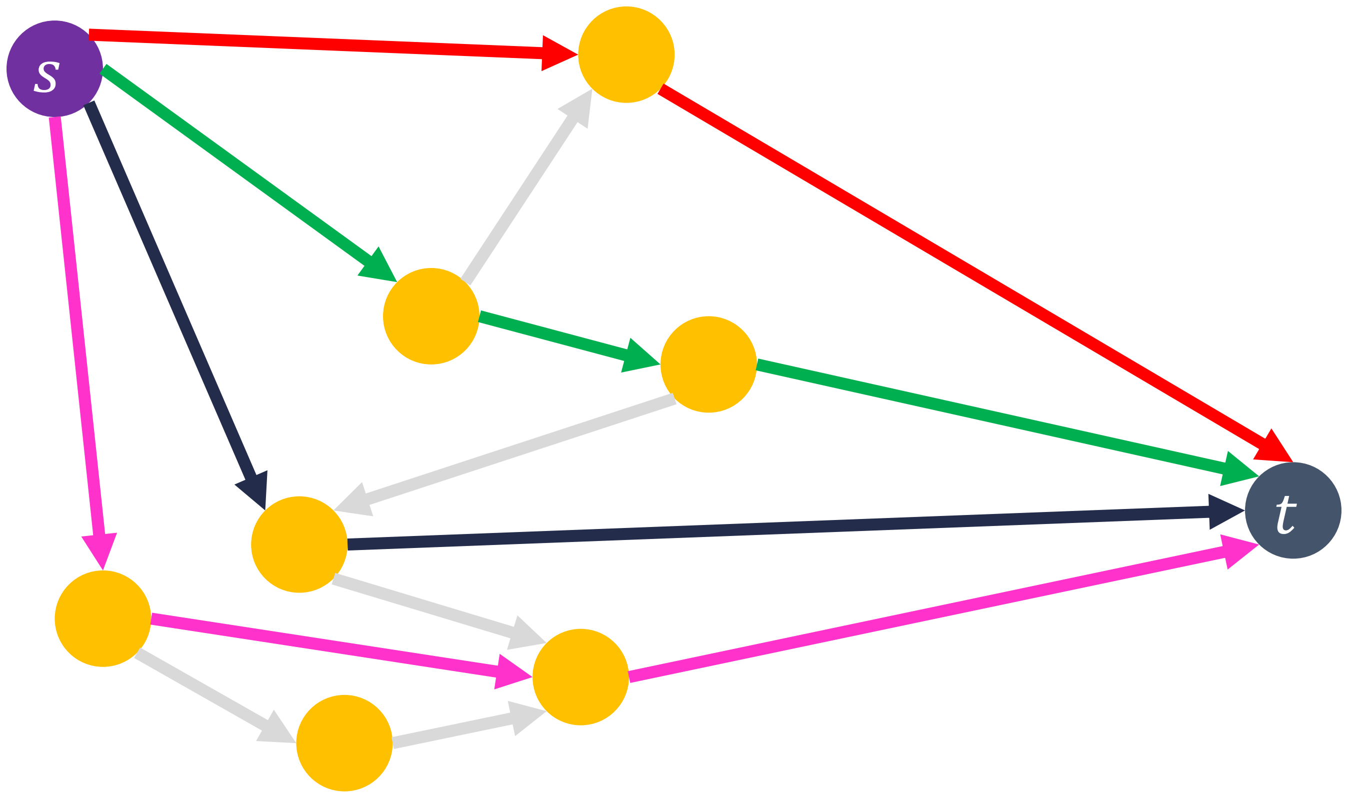
Problem: Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no edges



Set of size 3

Edge-Disjoint Paths

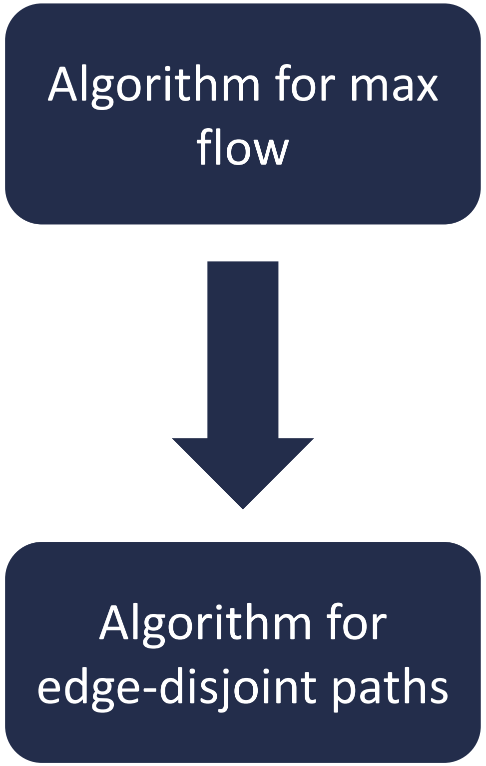
Problem: Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no edges

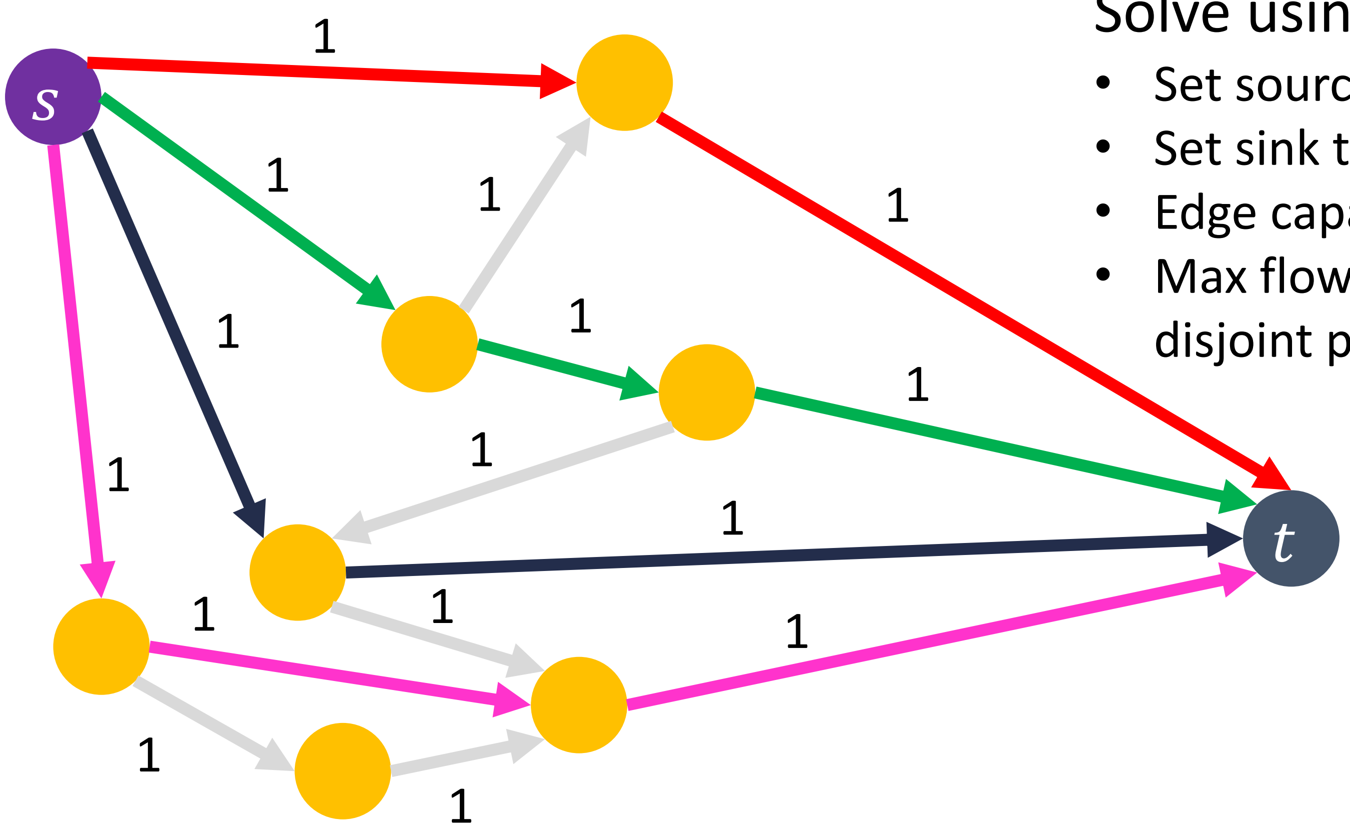


Set of size 4

Edge-Disjoint Paths

Problem: 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





Solve using max flow:

- Set source to be s
- Set sink to be t
- Edge capacity 1 for each edge
- Max flow = max number of edge-disjoint paths

Correctness of Edge-Disjoint Paths Algorithm

Theorem. The maximum flow equals the maximum number of edge-disjoint paths

Proof. Need to show two properties:

- If there is a flow with value k, then there are k edge-disjoint paths in the graph
- If there are k edge-disjoint paths from s to t in the graph, then there is a flow with value k

Without the first claim…

- Maximum flow could be much larger than the number of edge-disjoint paths

Without the second claim…

- Maximum flow could be much smaller than the number of edge-disjoint paths

Correctness of Edge-Disjoint Paths Algorithm

Theorem. The maximum flow equals the maximum number of edge-disjoint paths

Proof. Need to show two properties:

- If there is a flow with value k, then there are k edge-disjoint paths in the graph
- If there are k edge-disjoint paths from s to t in the graph, then there is a flow with value k

Claim 1. If there is a flow f with value k, then there are k edge-disjoint paths in the graph

- Take any edge (s, u_1) where $f(s, u_1) = 1$
- Since flow is conserved, there must be a sequence of nodes $u_2, \ldots, u_d = t$ where $f(u_i, u_{i+1}) = 1$ for all $i = 2, \ldots, d - 1$
- This gives a path from s to t (which delivers exactly 1 unit of flow)
- Set $f(s, u_1) = f(u_1, u_2) = \cdots = f(u_{d-1}, t) = 0$ and repeat this step to obtain the full set of k edge-disjoint paths

Correctness of Edge-Disjoint Paths Algorithm

Theorem. The maximum flow equals the maximum number of edge-disjoint paths

Proof. Need to show two properties:

- If there is a flow with value k, then there are k edge-disjoint paths in the graph
- If there are k edge-disjoint paths from s to t in the graph, then there is a flow with value k

Claim 2. If there are k edge-disjoint paths in the graph, then there is a flow with value k

- Since paths are edge disjoint, we can send 1 unit of flow along each of those paths
- Thus, there is a flow with value k

Correctness of Edge-Disjoint Paths Algorithm

Theorem. The maximum flow equals the maximum number of edge-disjoint paths

Proof. Need to show two properties:

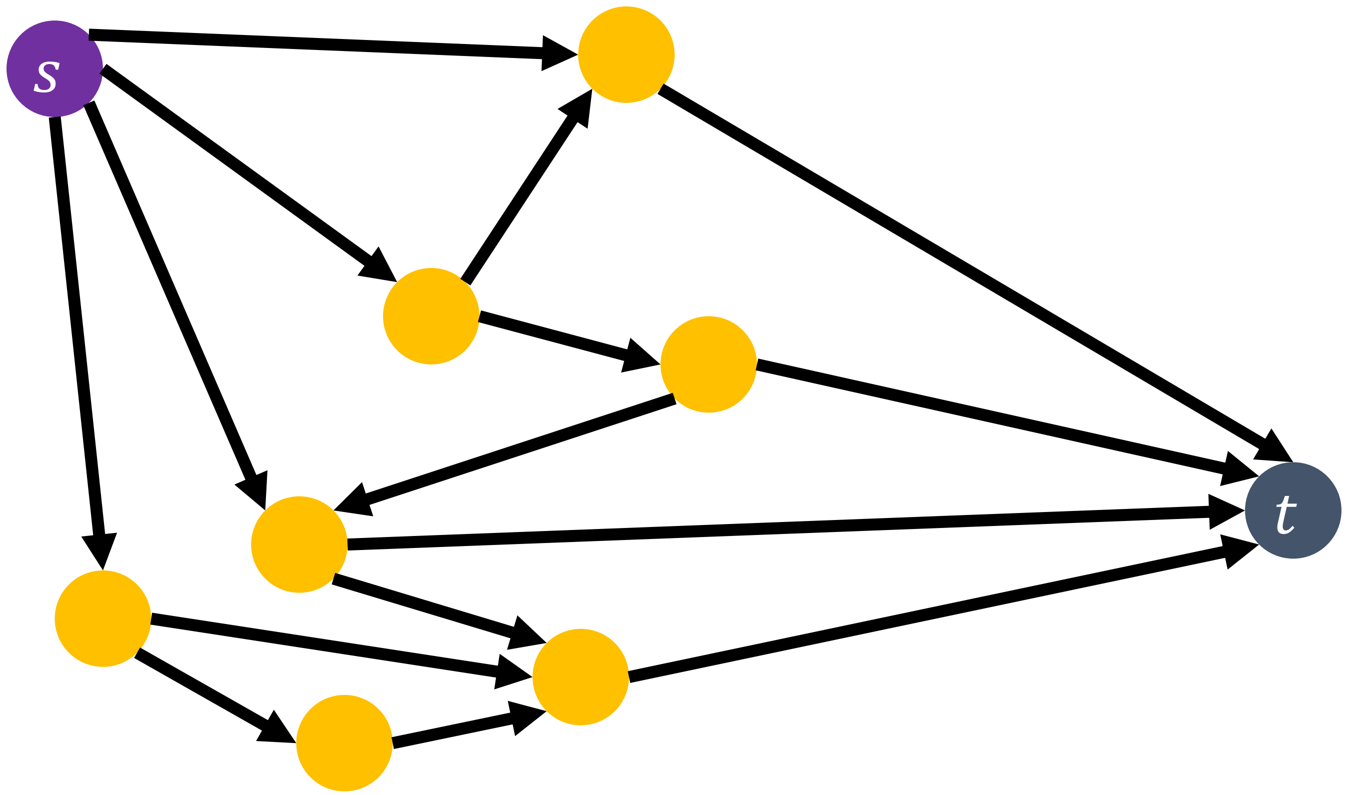
- If there is a flow with value k, then there are k edge-disjoint paths in the graph
- If there are k edge-disjoint paths from s to t in the graph, then there is a flow with value k

Conclusion: Finding the maximum flow in the graph G yields a maximal set of edge-disjoint paths in G

This is an example of a **reduction**: showing that solution to one problem (max flow) gives solution to another problem (edge-disjoint paths)

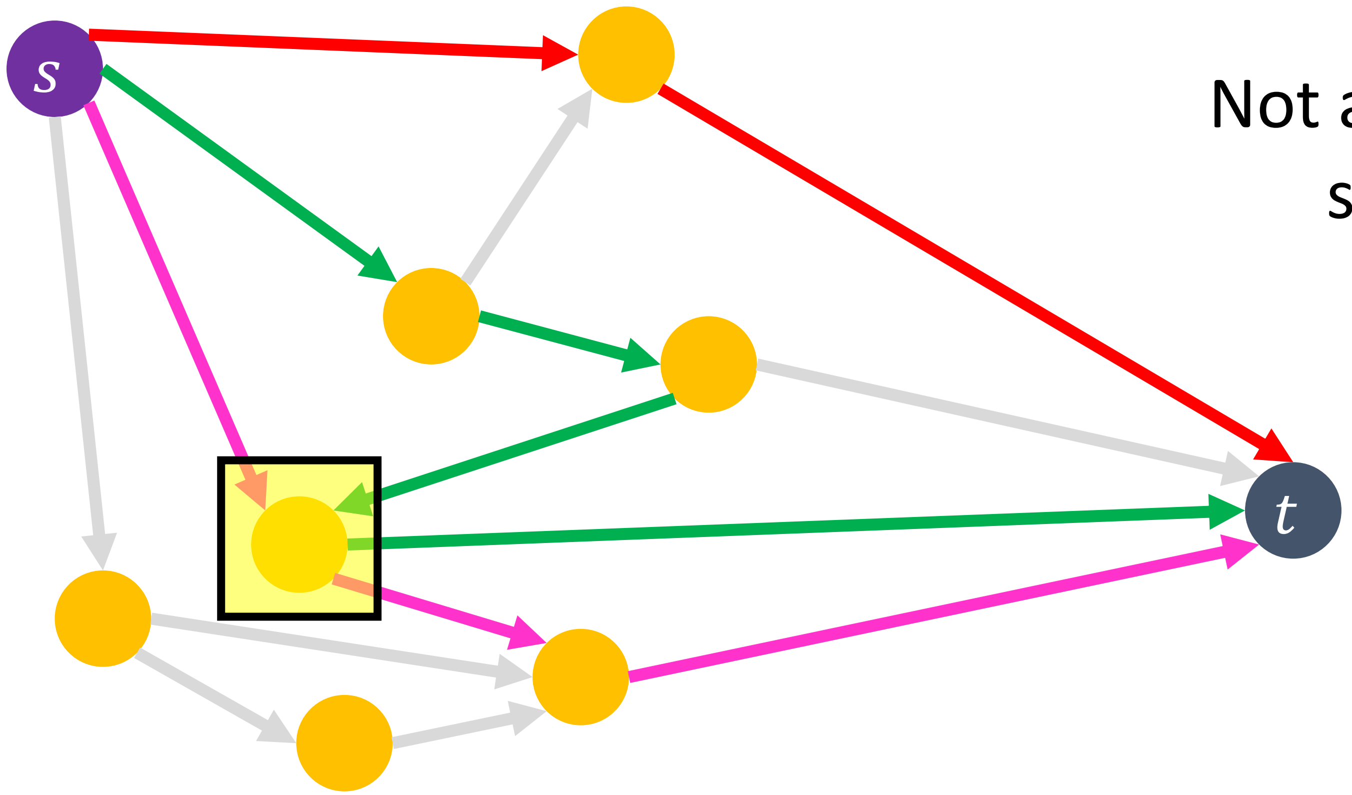
Vertex-Disjoint Paths

Problem: 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



Vertex-Disjoint Paths

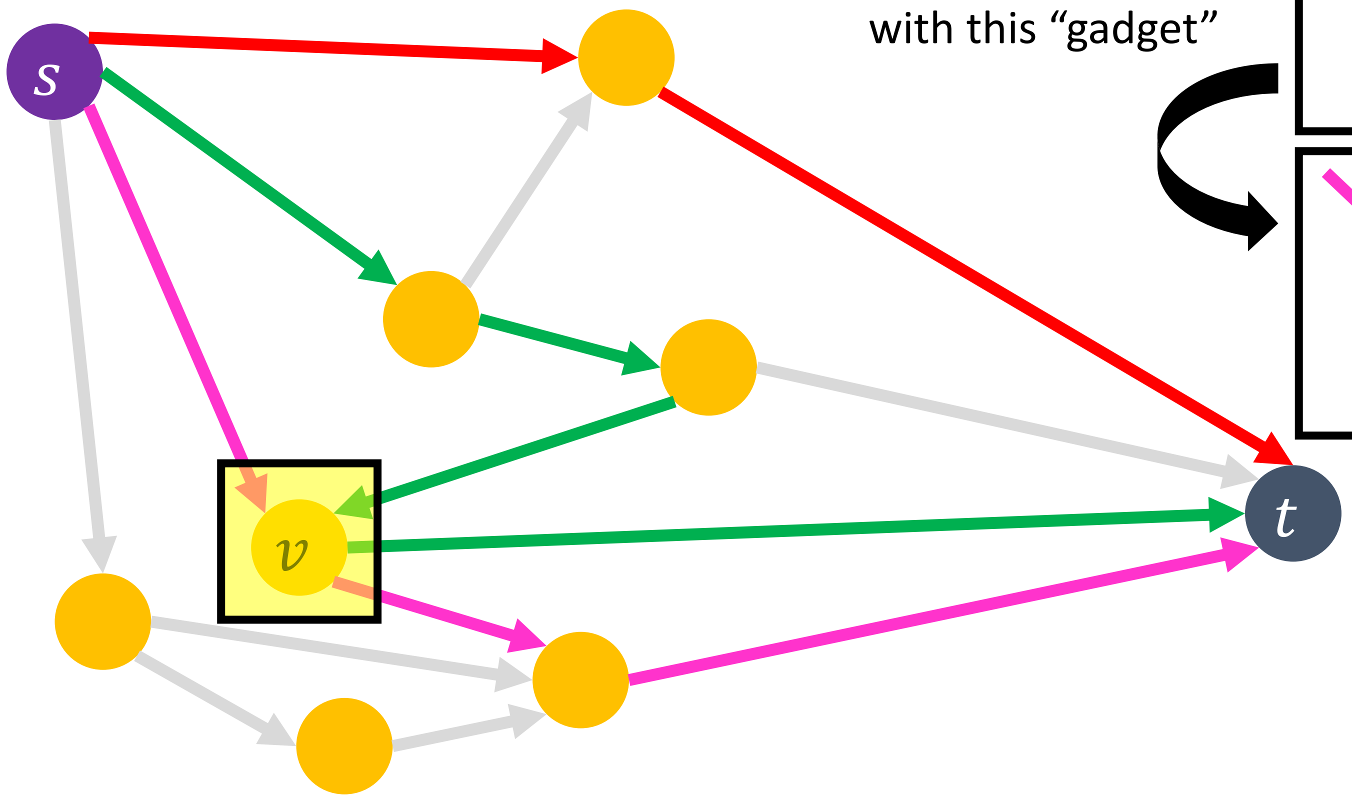
Problem: Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no <u>vertices</u>



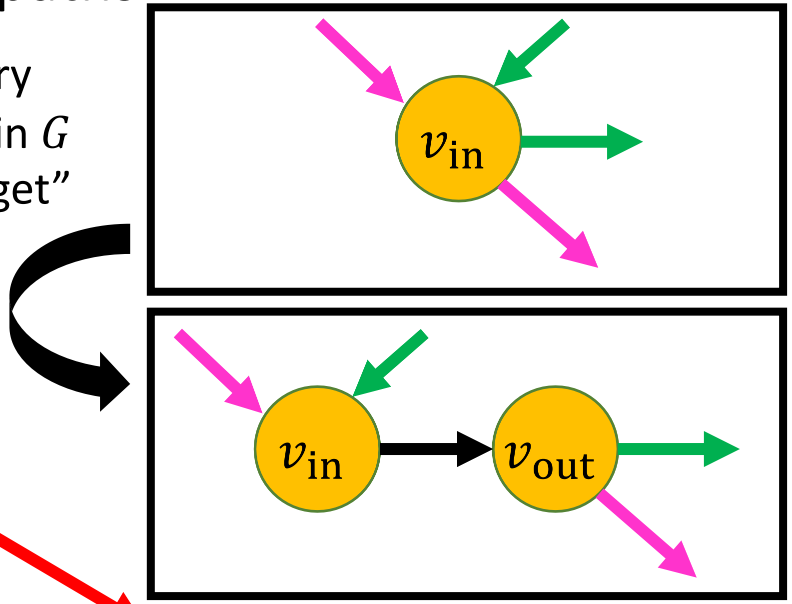
Not a vertex-disjoint set of paths!

Vertex-Disjoint Paths

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

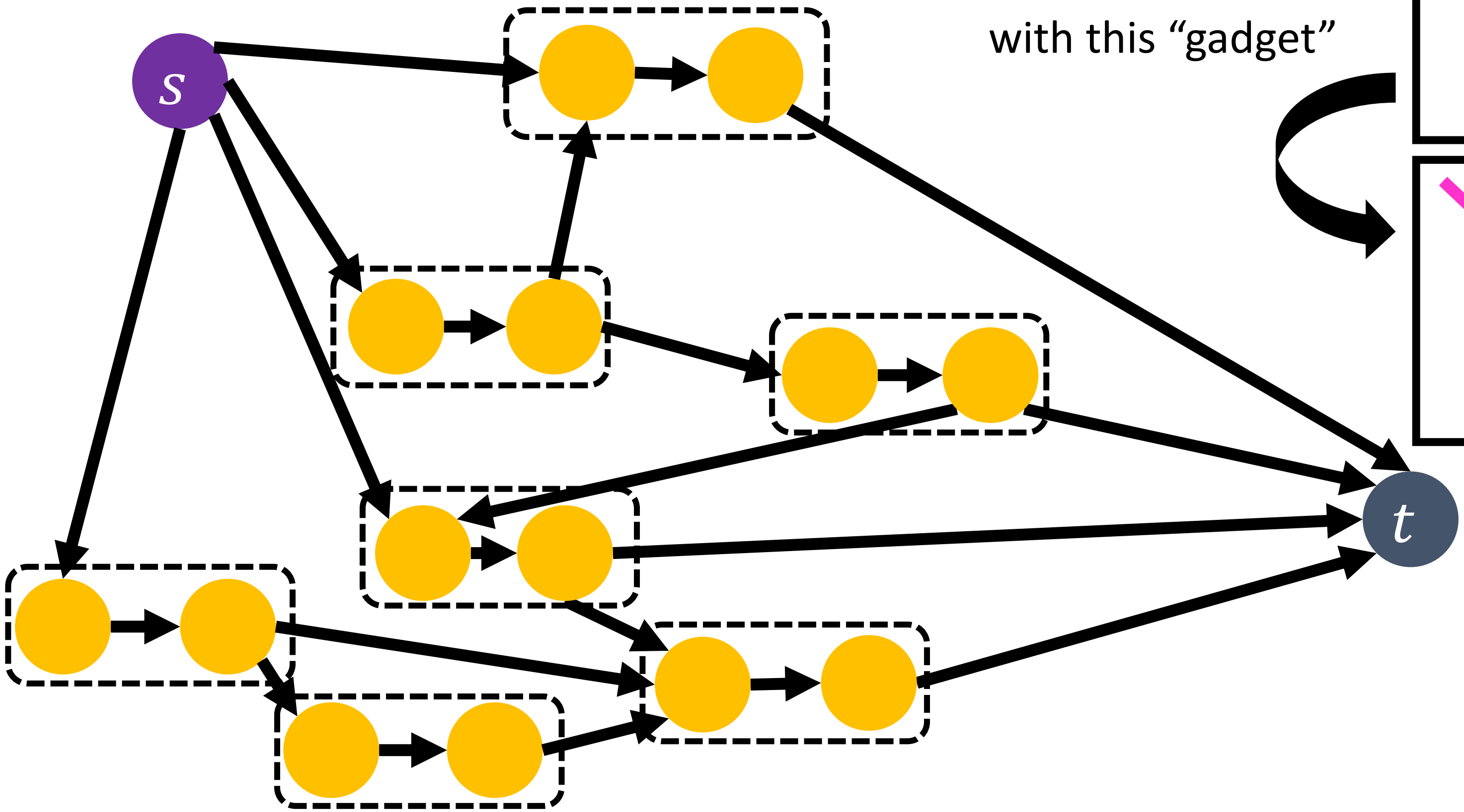


Replace every internal node in G with this "gadget"

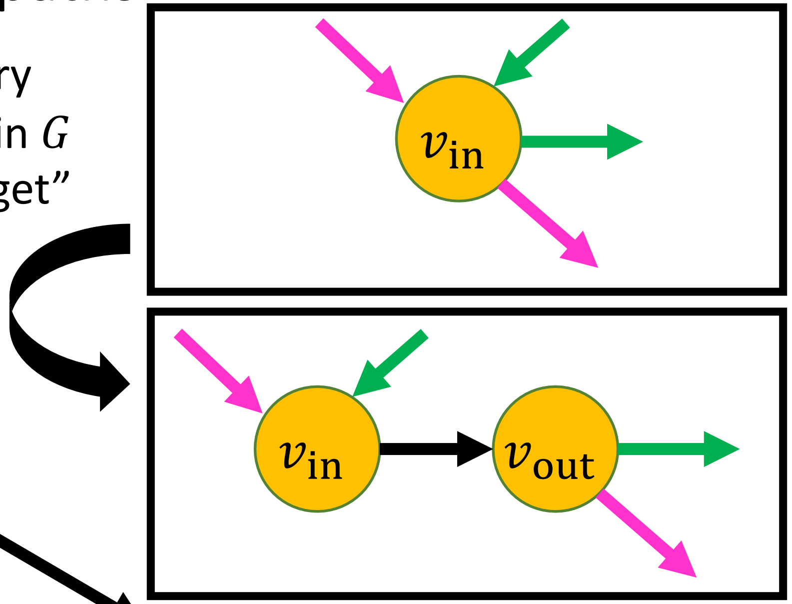


Vertex-Disjoint Paths

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



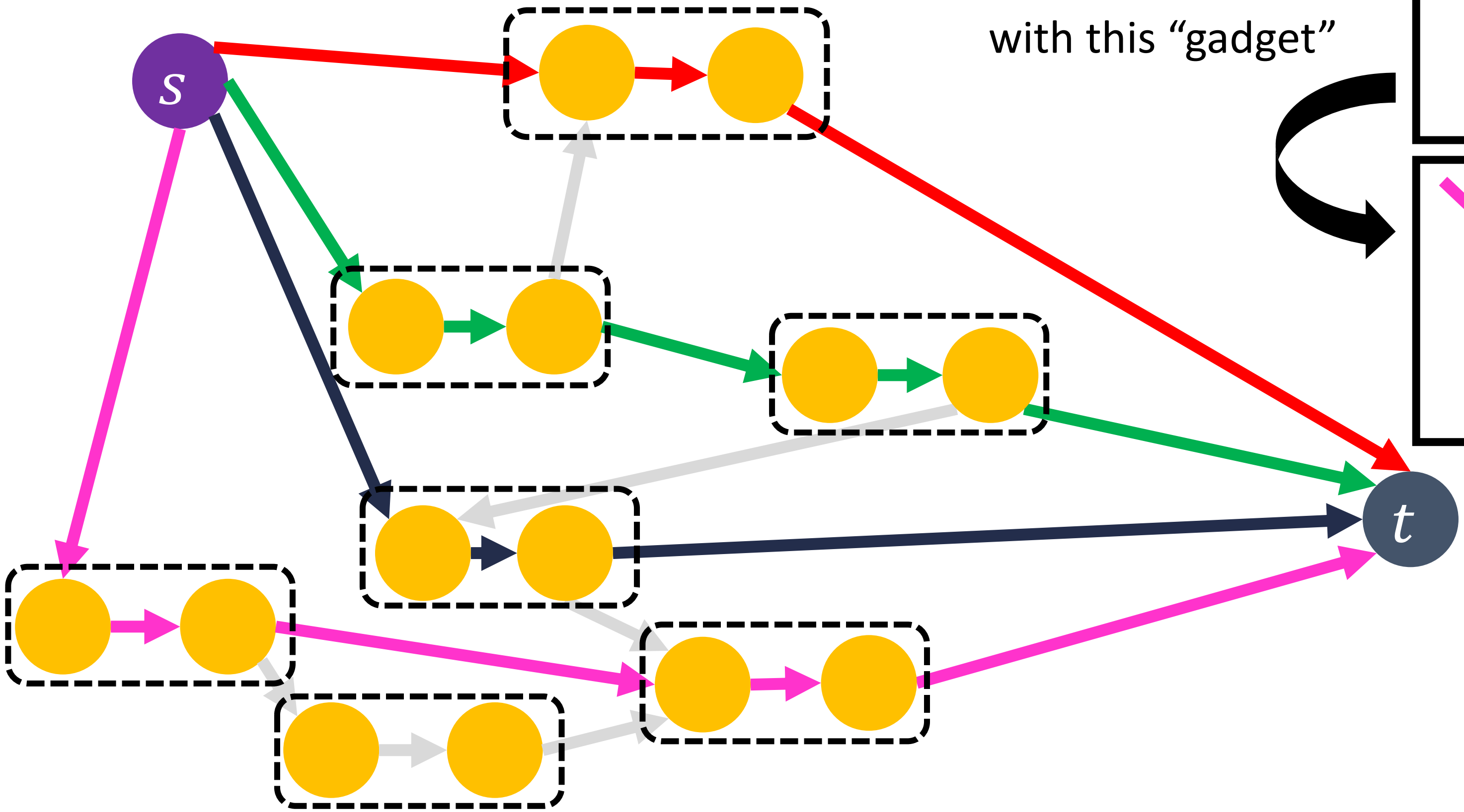
Replace every internal node in G with this "gadget"



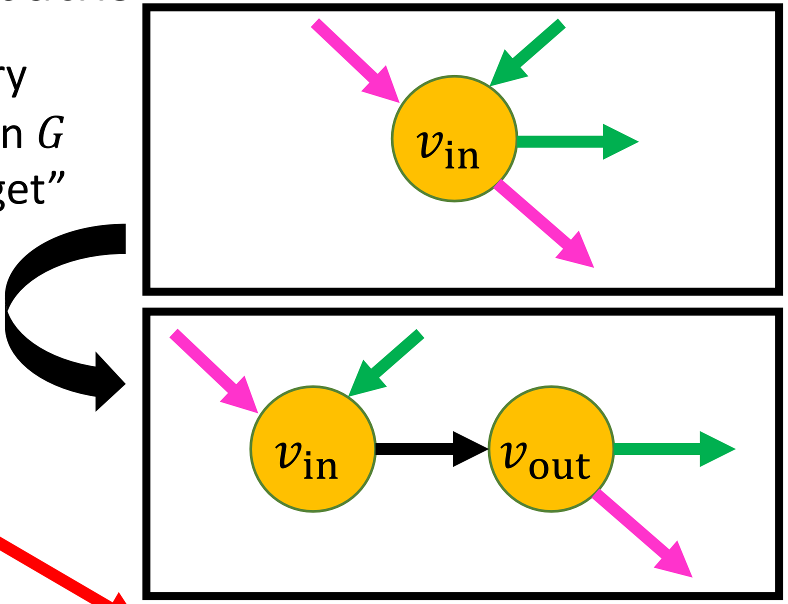
Solve edge-disjoint paths on new graph G'

Vertex-Disjoint Paths

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



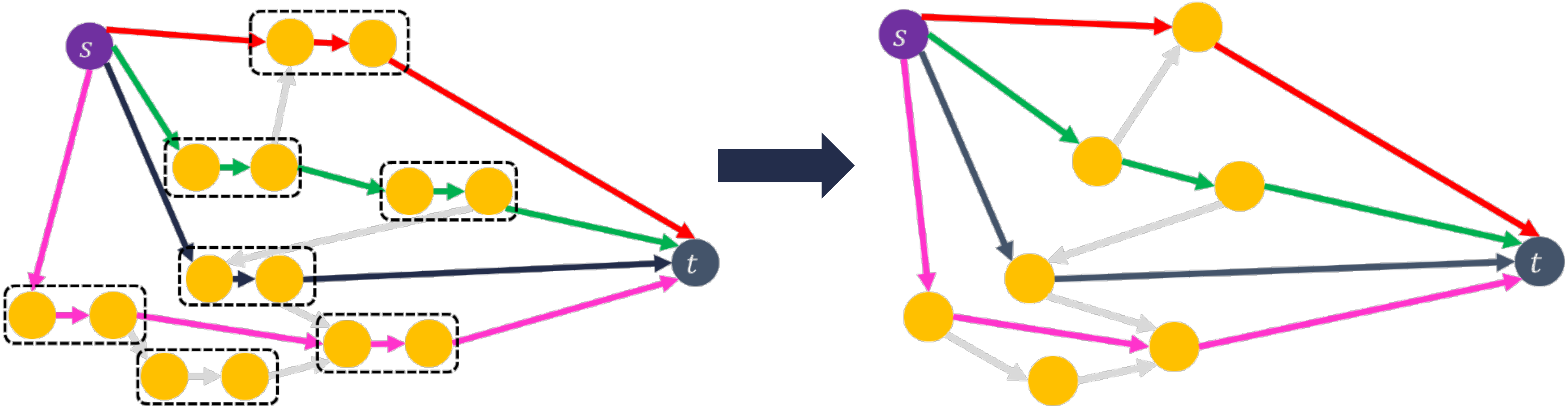
Replace every internal node in G with this "gadget"



Solve edge-disjoint paths on new graph G'

Vertex-Disjoint Paths

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



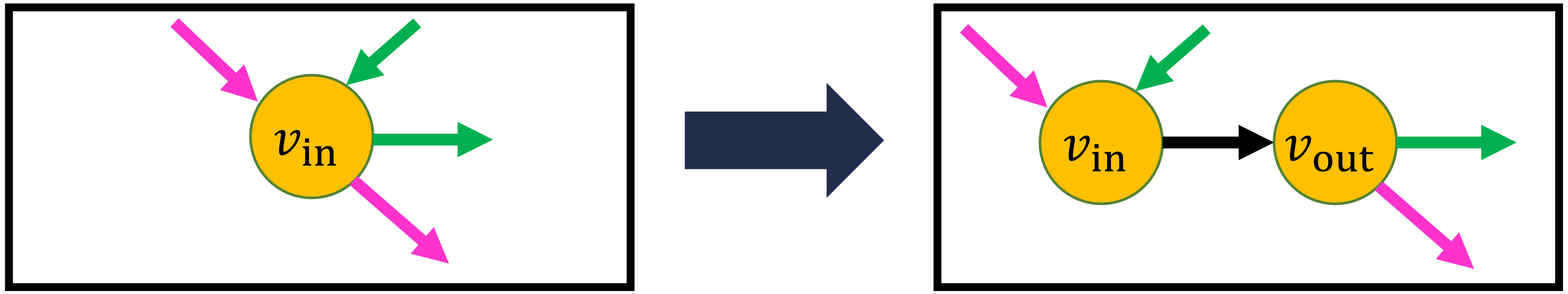
Set of edge-disjoint paths in G' corresponds to set of vertex-disjoint paths in G

Correctness of Vertex-Disjoint Paths Algorithm

Theorem. A set of paths is vertex-disjoint paths in G if and only if it is edge-disjoint in G'

Proof. Follows essentially by construction of the gadget

- A path $(s, v_1, \dots, v_n, t)$ in G maps to a path $\left(s, v_{1,\text{in}}, v_{1,\text{out}}, \dots, v_{n,\text{in}}, v_{n,\text{out}}, t\right)$ in G'
- A set of vertex-disjoint paths in G is also vertex-disjoint in G', and thus, must also be edge-disjoint
- A set of edge-disjoint paths in G' must also be vertex disjoint
  - There is only a single outgoing edge in $v_{i,\text{in}}$ and a single incoming edge in $v_{i,\text{out}}$
  - If two paths in G' use $v_{i,\text{in}}$ or $v_{i,\text{out}}$, they must use the edge $\left(v_{i,\text{in}}, v_{i,\text{out}}\right)$, which contradicts edge-disjointness



Graph G Graph G'

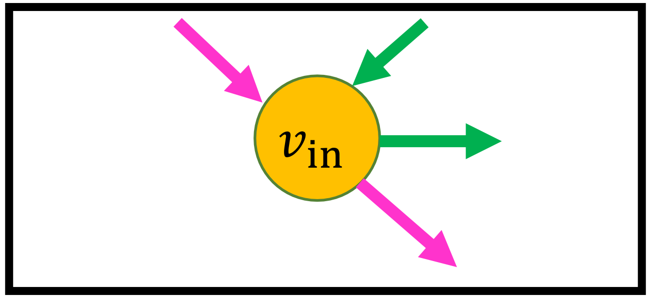
Correctness of Vertex-Disjoint Paths Algorithm

Theorem. A set of paths is vertex-disjoint paths in G if and only if it is edge-disjoint in G'

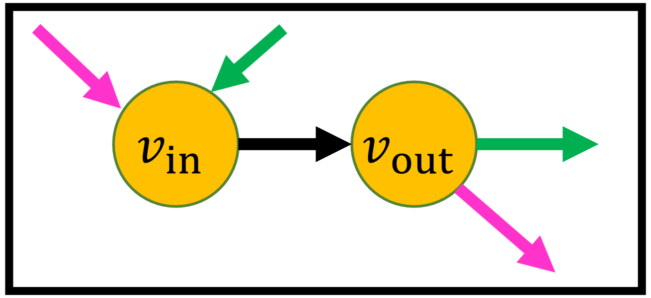
Conclusion. Solving edge-disjoint paths in G' gives a solution to vertex-disjoint paths in G

Why do we need to show <u>both</u> directions in the theorem?

- Vertex-disjoint in $G \Rightarrow$ edge-disjoint in G' needed to argue that optimal solution in G corresponds to some solution in G'
- Edge-disjoint in $G' \Rightarrow$ vertex-disjoint in G needed to argue that any feasible solution (in G') corresponds to a solution to the original problem (in G)



Graph G



Graph G'

Correctness of Vertex-Disjoint Paths Algorithm

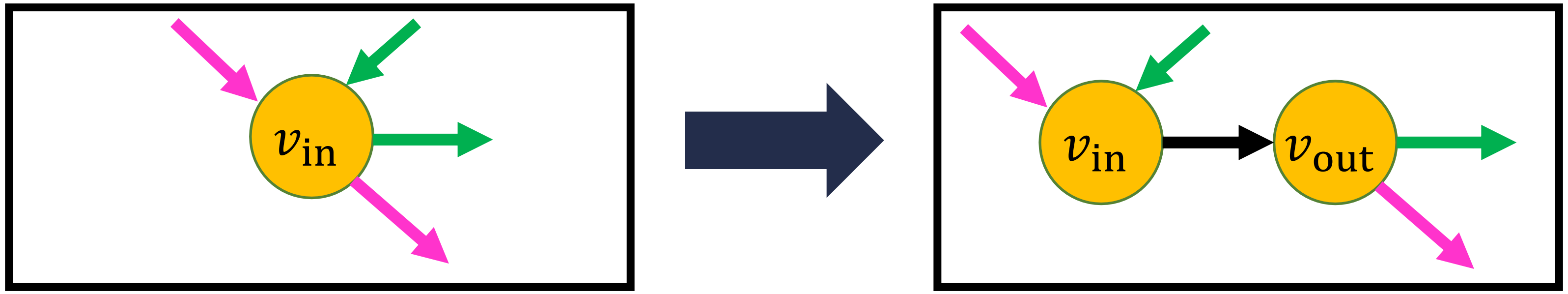
Theorem. A set of paths is vertex-disjoint paths in G if and only if it is edge-disjoint in G'

If this was not true, then the maximal set of vertex disjoint paths in G might not correspond to a set of edge-disjoint paths in G' (so finding the maximal set of edge-disjoint paths in G' may not give a solution to the original problem)

Conclusion. Solv … in G

Why do we need …

- Vertex-disjoint in $G \Rightarrow$ edge-disjoint in G' needed to argue that optimal solution in G corresponds to some solution in G'
- Edge-disjoint in $G' \Rightarrow$ vertex-disjoint in G needed to argue that any feasible solution (in G') corresponds to a solution to the original problem (in G)



Graph G Graph G'

Correctness of Vertex-Disjoint Paths Algorithm

Theorem. A set of paths is vertex-disjoint paths in G if and only if it is edge-disjoint in G'

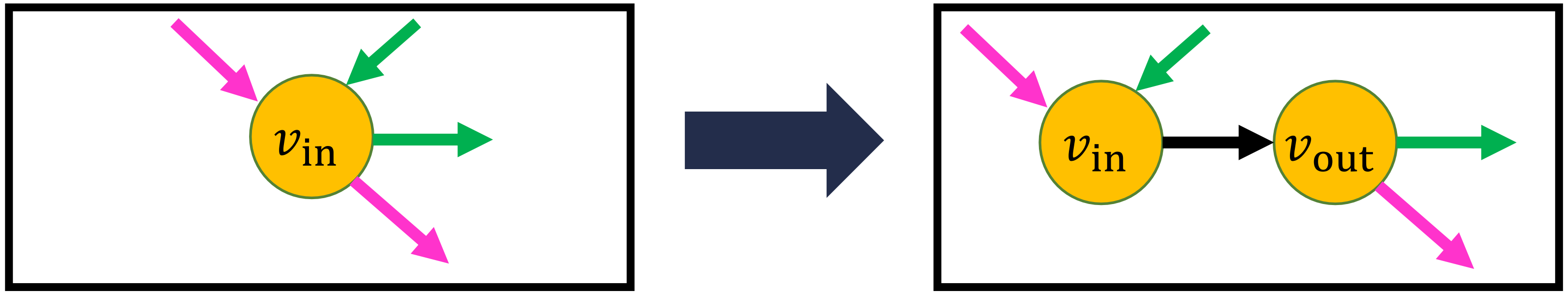
Conclusion. Solving edge-disjoint paths in G' gives a solution to vertex-disjoint paths in G

Why do we need

- Vertex-disj
  G correspon

If this was not true, then the maximal set of edge-disjoint paths in G' might not correspond to any collection of vertex-disjoint paths in G (so the solution to our new problem may not give a solution to the original problem)

- Edge-disjoint in G' $\Rightarrow$ vertex-disjoint in G needed to argue that any feasible solution (in G') corresponds to a solution to the original problem (in G)



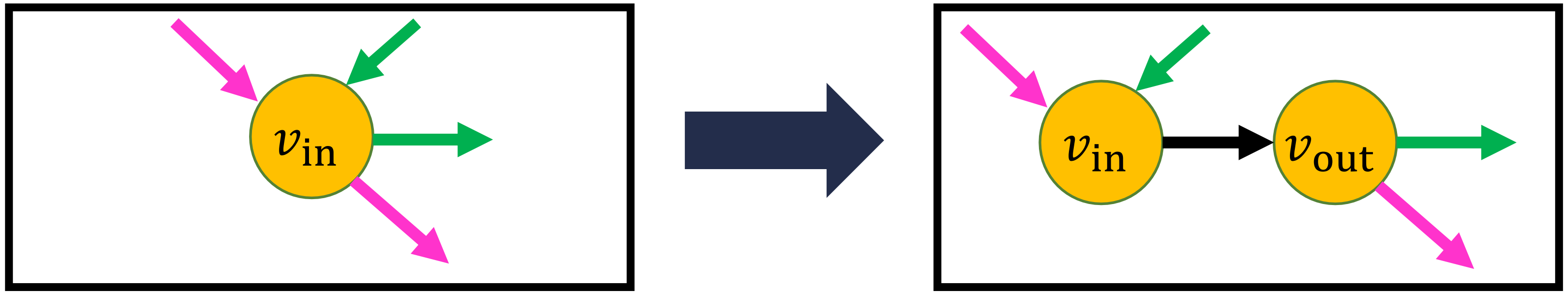
Graph G Graph G'

Correctness of Vertex-Disjoint Paths Algorithm

Theorem. A set of paths is vertex-disjoint paths in G if and only if it is edge-disjoint in G'

Conclusion. Solving edge-disjoint paths in G' gives a solution to vertex-disjoint paths in G

This is another example of a **reduction**: showing that solution to one problem (edge-disjoint paths in G') gives solution to another problem (vertex-disjoint paths in G)

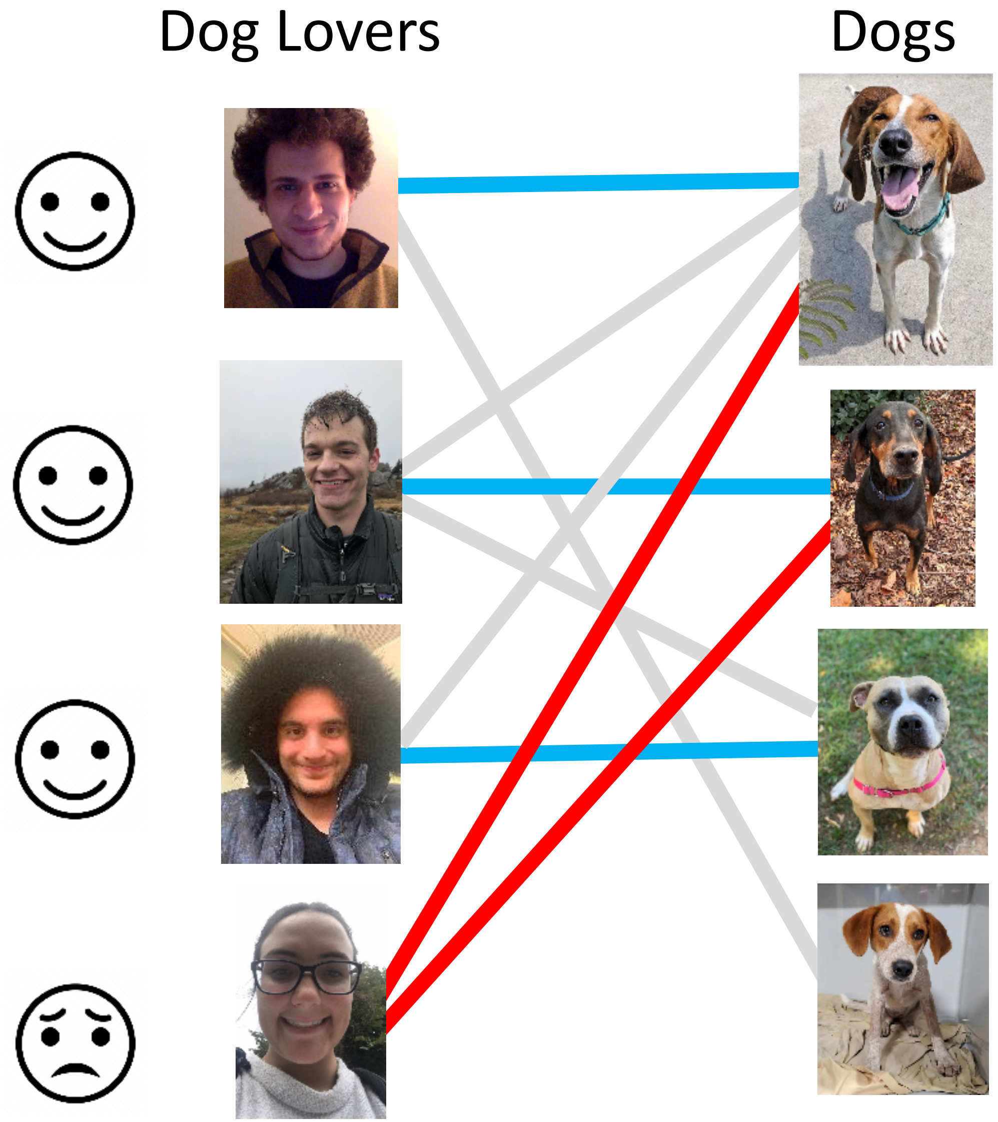


Graph G Graph G'

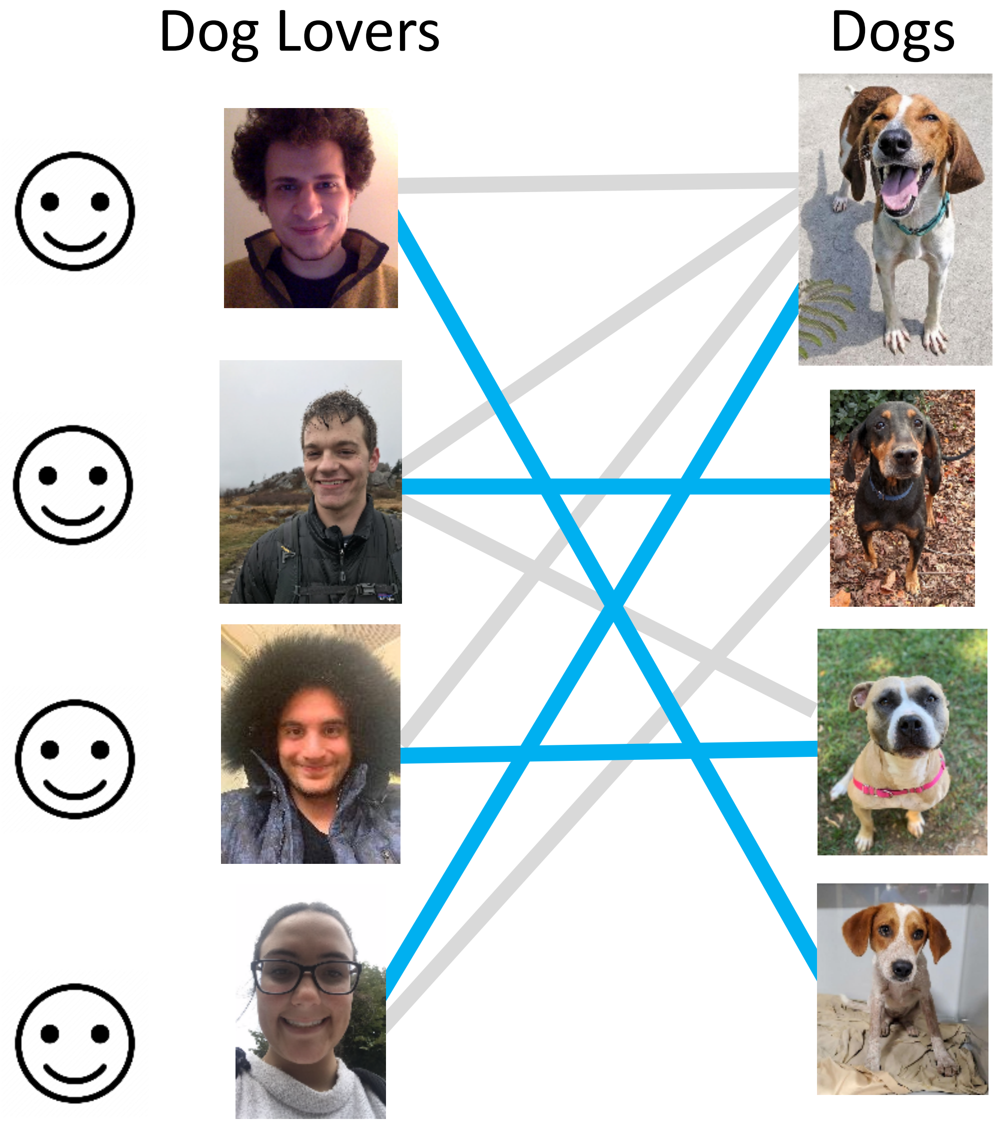
Maximum Bipartite Matching

Dog Lovers | Dogs

Maximum Bipartite Matching



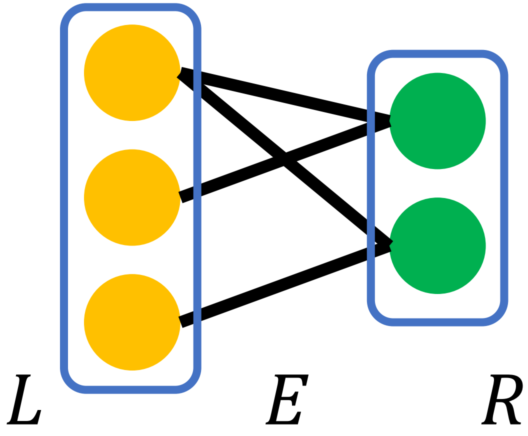
Maximum Bipartite Matching



Maximum Bipartite Matching

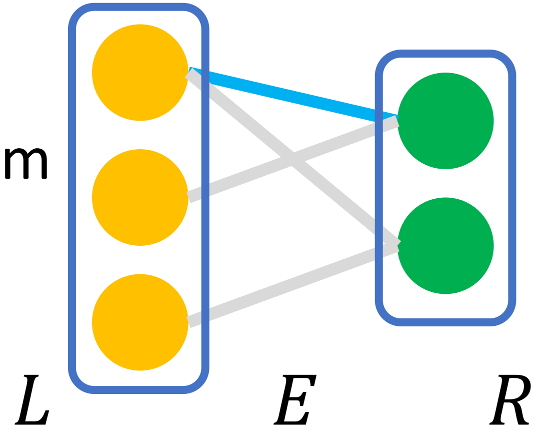
Let $G = (L, R, E)$ be a bipartite graph

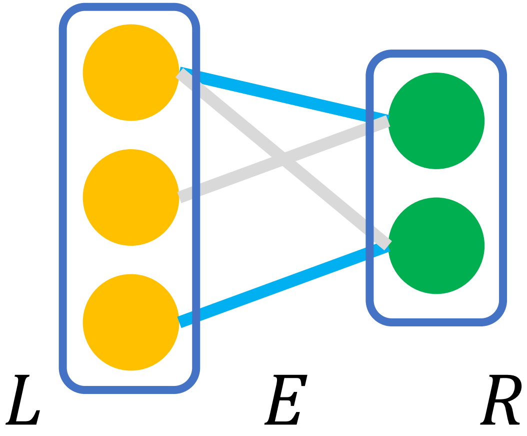
- L: a set of "left" nodes
- R: a set of "right" nodes
- E: a set of edges between L and R



Problem: find the largest set of edges $M \subseteq E$ (i.e., a matching) such that each node $u \in L$ or $v \in R$ is incident on at most one edge

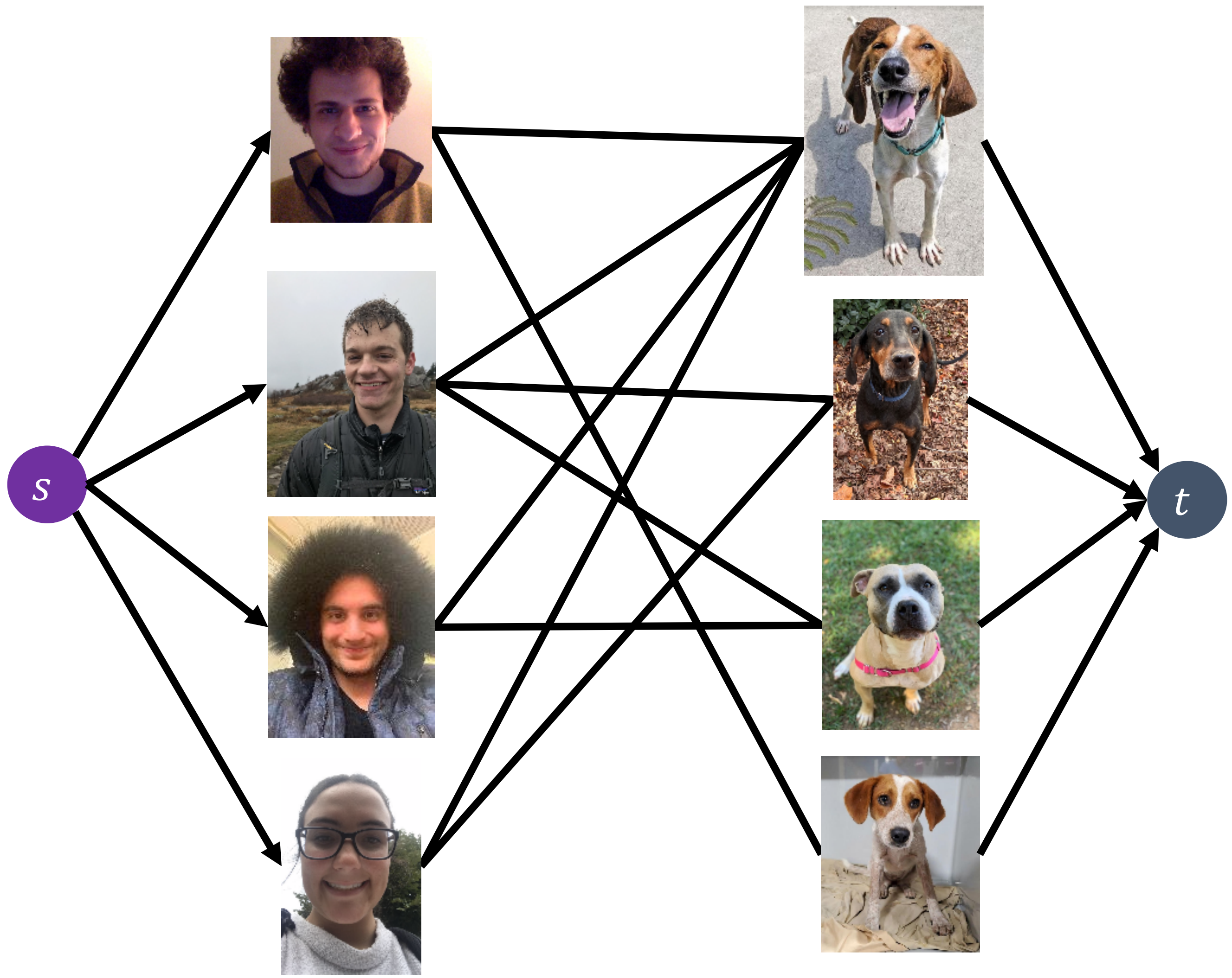
non-maximum matching





maximum matching

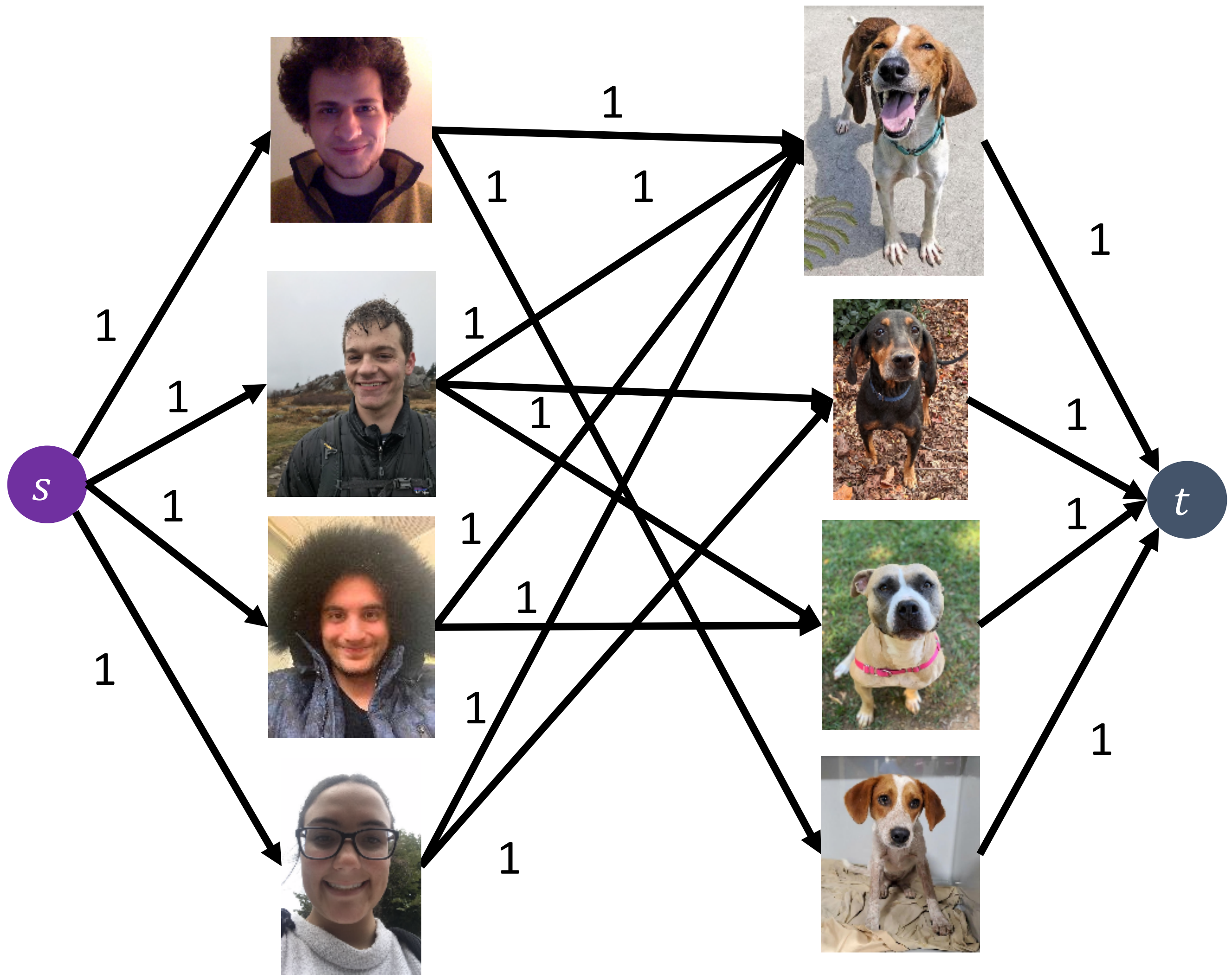
Maximum Bipartite Matching Using Max Flow



Idea: Convert (L, R, E) into a flow network $G' = (V', E')$ by introducing a source s and a sink t

- Connect the source to each left node
- Connect each right node to the sink

Maximum Bipartite Matching Using Max Flow



Idea: Convert (L, R, E) into a <u>flow network</u> $G' = (V', E')$ by introducing a source s and a sink t

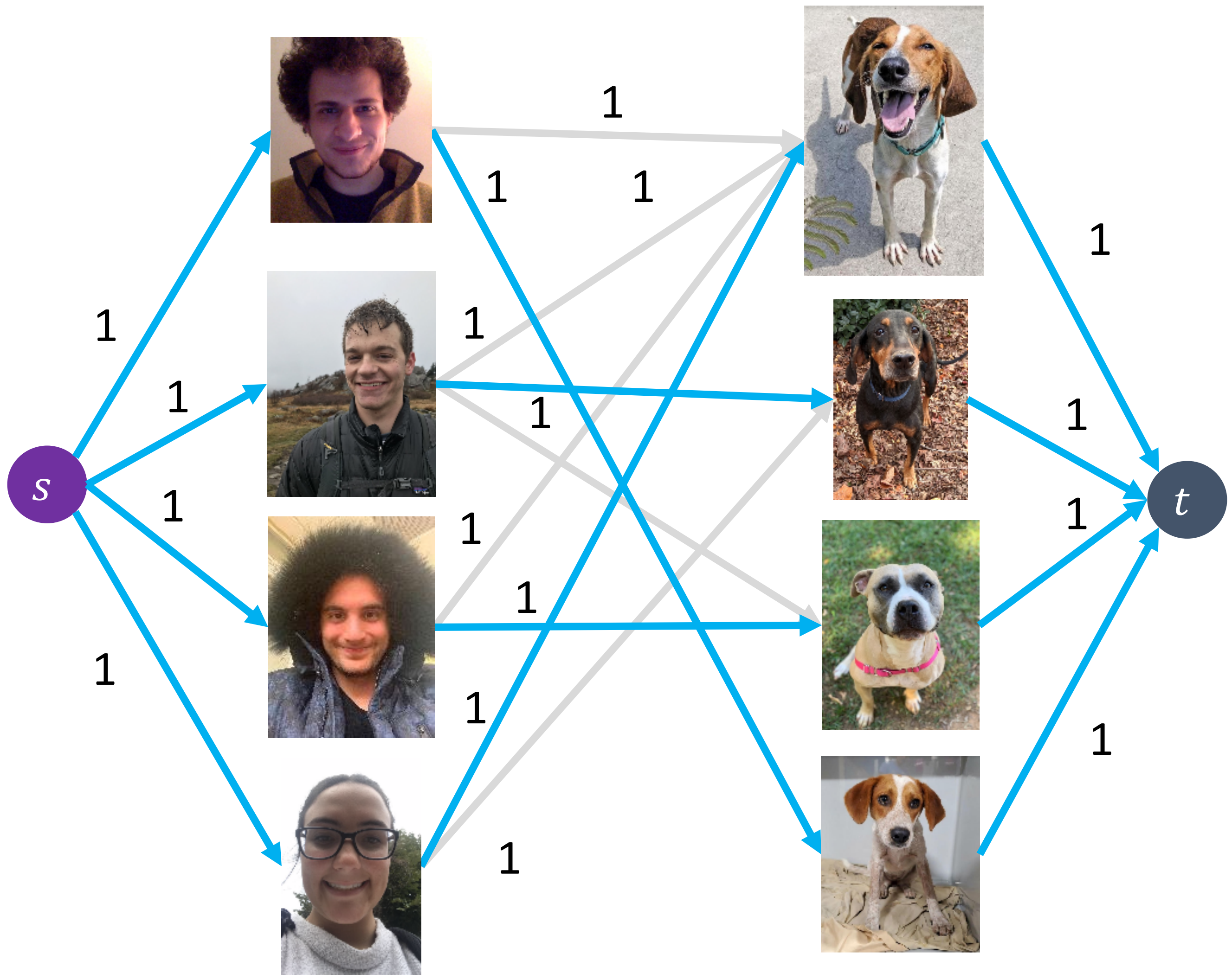
- Connect the source to each left node
- Connect each right node to the sink
- For each edge in E, introduce a directed edge from the left node to the right node
- Assign capacity 1 to each of the edges

In particular:

- $V' = L \cup R \cup \{s, t\}$
- $E' = \{ (s, u) \mid u \in L \} \cup \{ (v, t) \mid v \in R \} \cup E$

Compute a max (integer) flow in G'

Maximum Bipartite Matching Using Max Flow



Idea: Convert (L, R, E) into a flow network $G' = (V', E')$ by introducing a source s and a sink t

- Connect the source to each left node
- Connect each right node to the sink
- For each edge in E, introduce a directed edge from the left node to the right node
- Assign capacity 1 to each of the edges

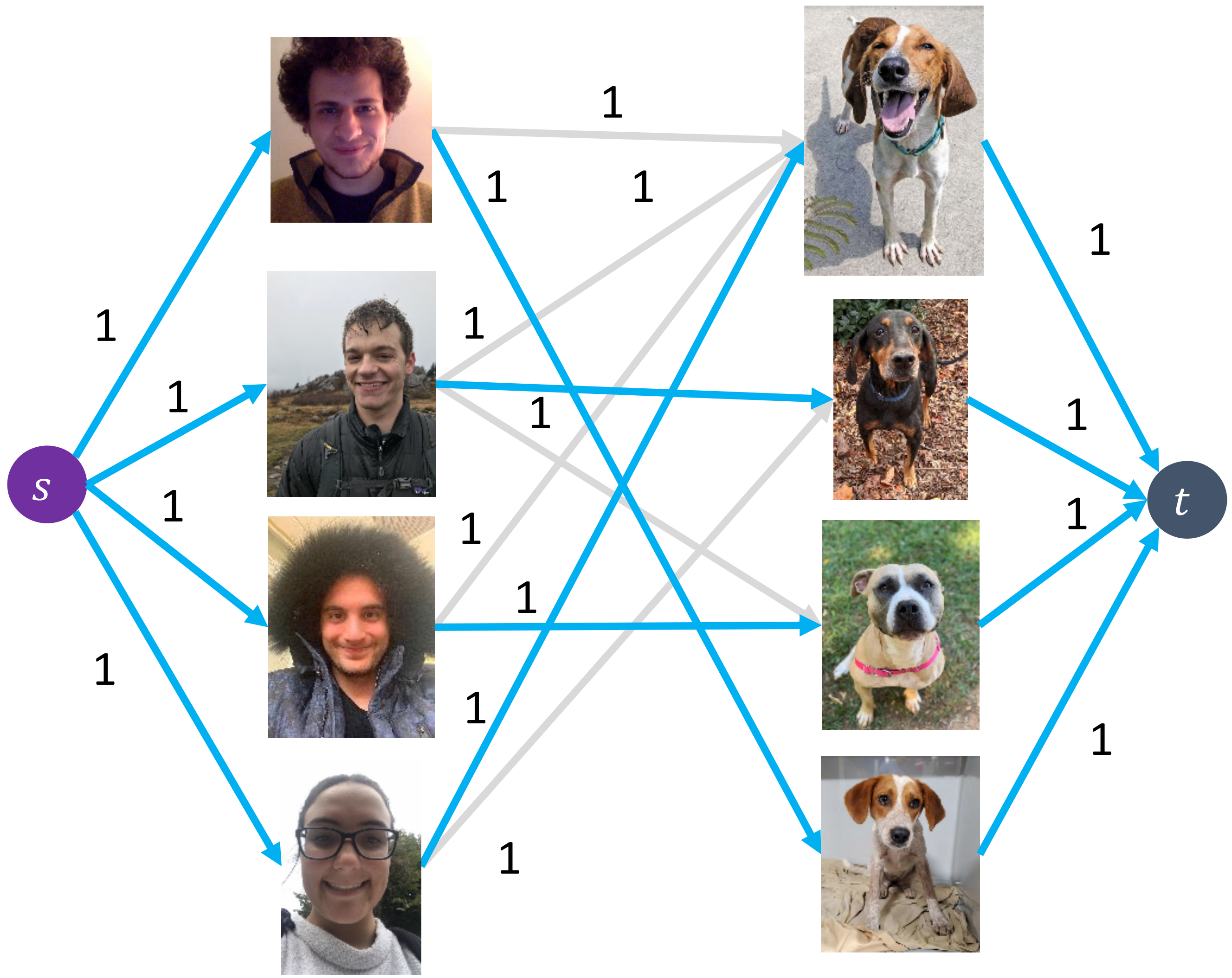
In particular:

- $V' = L \cup R \cup \{s, t\}$
- $E' = \{ (s, u) \mid u \in L \} \cup \{ (v, t) \mid v \in R \} \cup E$

Compute a max (integer) flow in G'

Claim: edges used in the max flow (between L and R) is precisely a maximum matching

Bipartite Matching Correctness



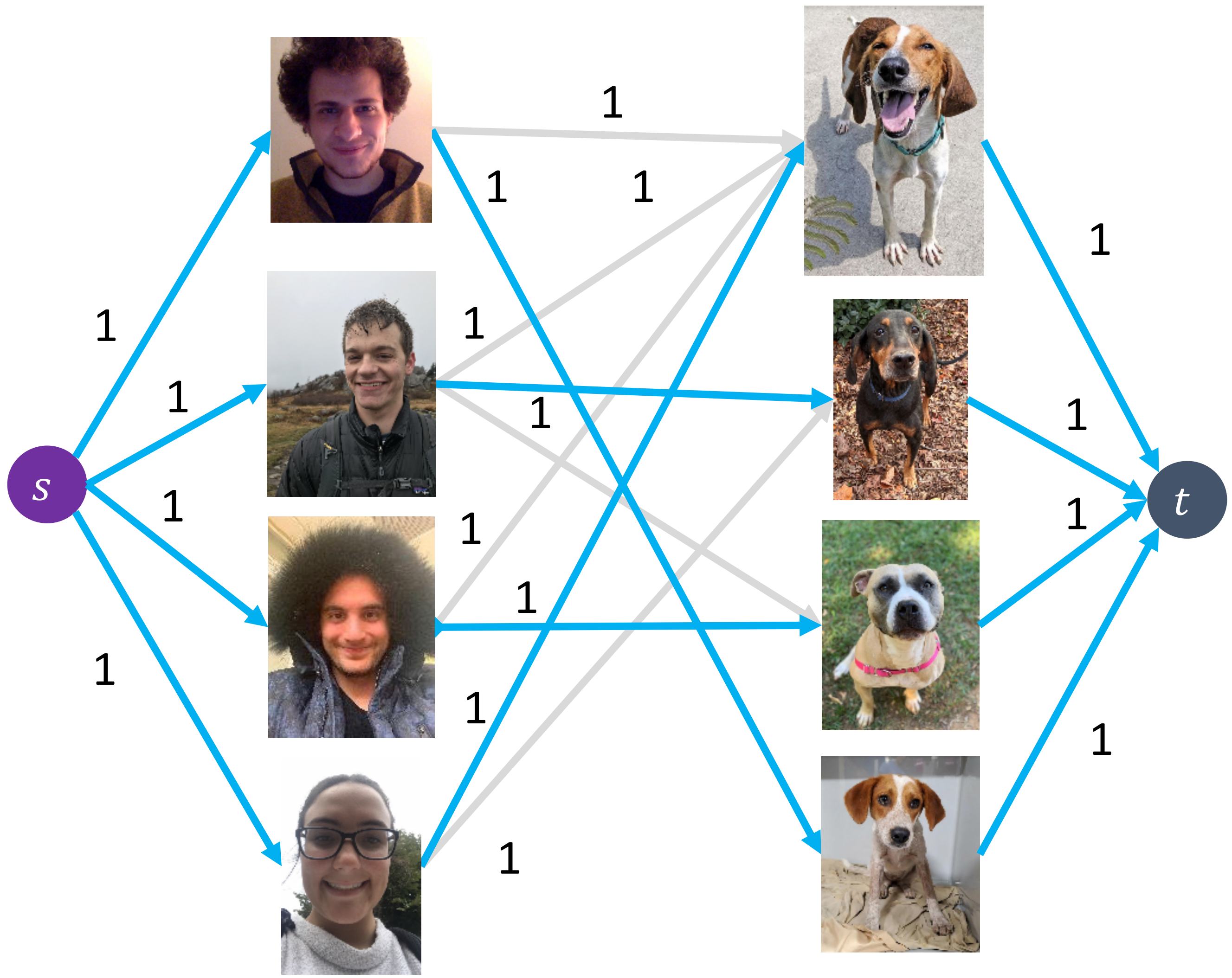
Let M be the set of edges used by the max flow in G'
Similar to before, we need to show the following:

- If there is an integer flow with value k, there is a matching with k edges
- If there is a matching with k edges, there is an integer flow with value k

Claim: Integer flow with value $k \Rightarrow$ matching of size k

- Since capacities are $0/1$, flow along each edge in f is also $0/1$
- At most 1 unit of flow can enter each node in L
- If there is 1 unit of flow entering $u \in L$, there must be exactly 1 unit of flow exiting L (along an edge in E) to some node $v \in R$
- There is only 1 outgoing edge from v to t so all other incoming edges to r have 0 flow

Bipartite Matching Correctness



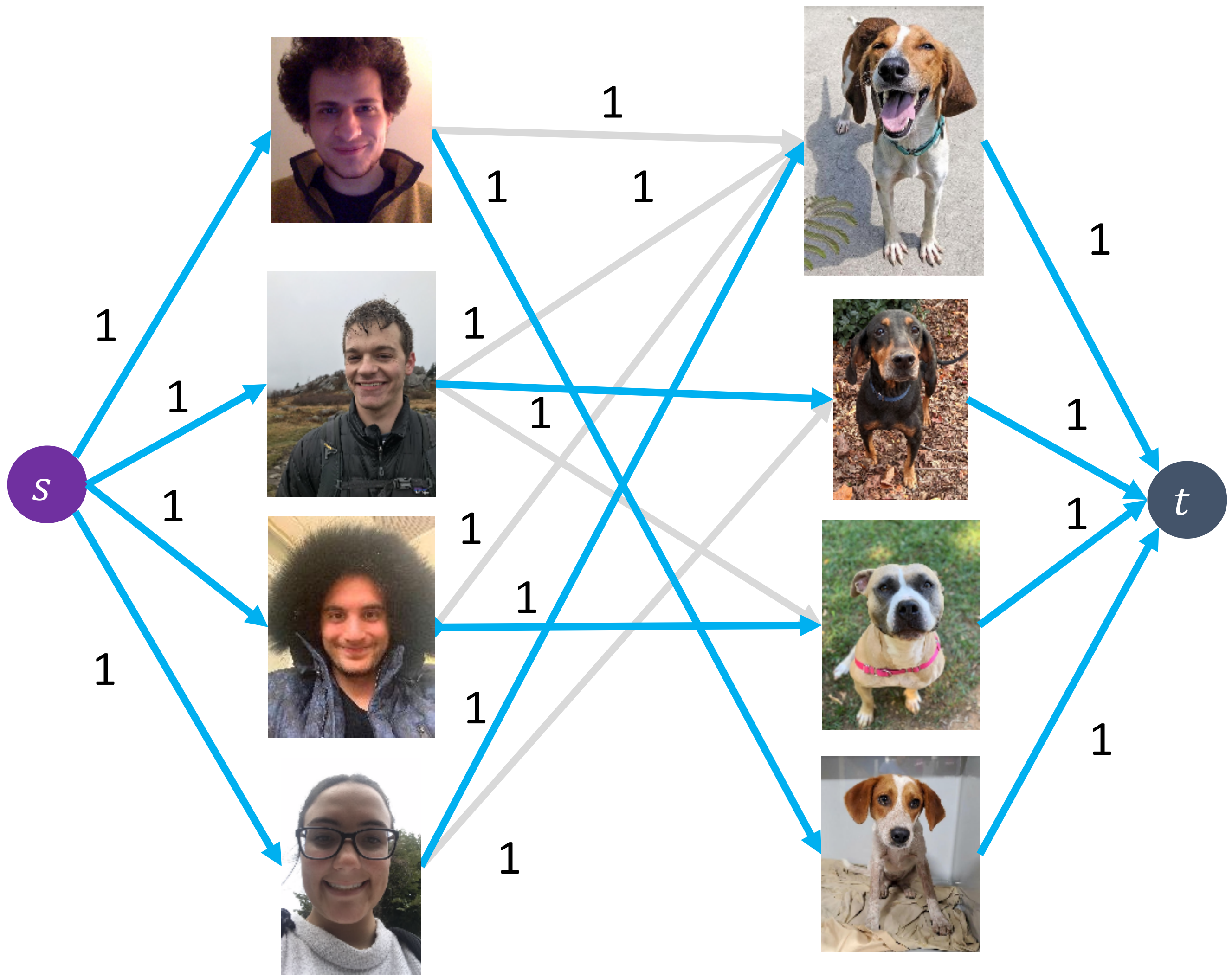
Let M be the set of edges used by the max flow in G'
Similar to before, we need to show the following:

- If there is an integer flow with value k, there is a matching with k edges
- If there is a matching with k edges, there is an integer flow with value k

Claim: Matching of size $k \Rightarrow$ integer flow with value k

- Send 1 unit of flow from s to each matched node in L, 1 unit of flow along the matched edges from L to R, and 1 unit of flow from each matched node in R to the sink t
- This yields an integer flow with value k in G'

Bipartite Matching Correctness



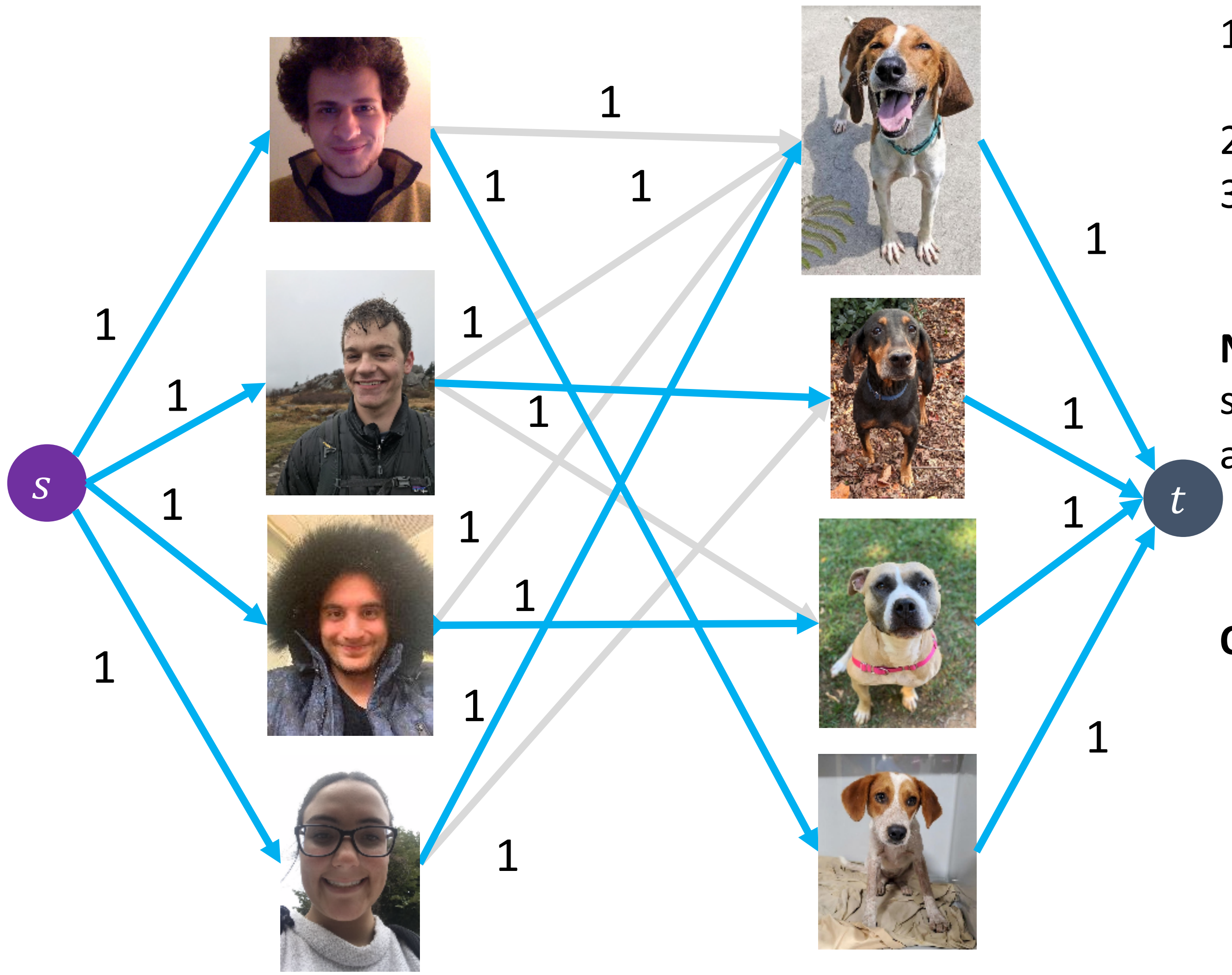
Let M be the set of edges used by the max flow in G'
Similar to before, we need to show the following:

- If there is an integer flow with value k, there is a matching with k edges
- If there is a matching with k edges, there is an integer flow with value k

Conclusion: Solving max (integer) flow in $G' = (V', E')$ yields a maximum bipartite matching in $G = (L, R, E)$

This is another example of a **reduction**: showing that solution to one problem (max flow in G') gives solution to another problem (max bipartite matching in G)

Running Time of Bipartite Matching



1. Construct flow graph $G' = (V', E')$ from $G = (L, R, E)$ — $O(|L| + |R|)$
2. Find a max (integer) flow in G' — $O\big(|E|(|L| + |R|)\big)$
3. Output the set of edges between L and R with flow 1 — $O(|L| + |R|)$

Note: Maximum flow $|f|$ in G' is bounded by $\min(|L|, |R|)$, so running time of Ford-Fulkerson (assuming linear-time augmenting path selection) is

$$O(|E| \cdot \min(|L|, |R|)) = O\big(|E|(|L| + |R|)\big)$$

Overall running time: $O\big(|E|(|L| + |R|)\big) = O(|E||V|)$

Reductions

Very <u>general</u> technique for designing algorithms

Idea: map the problem A (that we are trying to solve) to another problem B that we already know how to solve

So far in this course, we have reduced problems to smaller subproblems (i.e., divide and conquer, dynamic programming, greedy algorithms); reductions reduce one problem to a <u>different</u> problem

Reductions

Very general technique for designing algorithms

Idea: map the problem A (that we are trying to solve) to another problem B that we already know how to solve

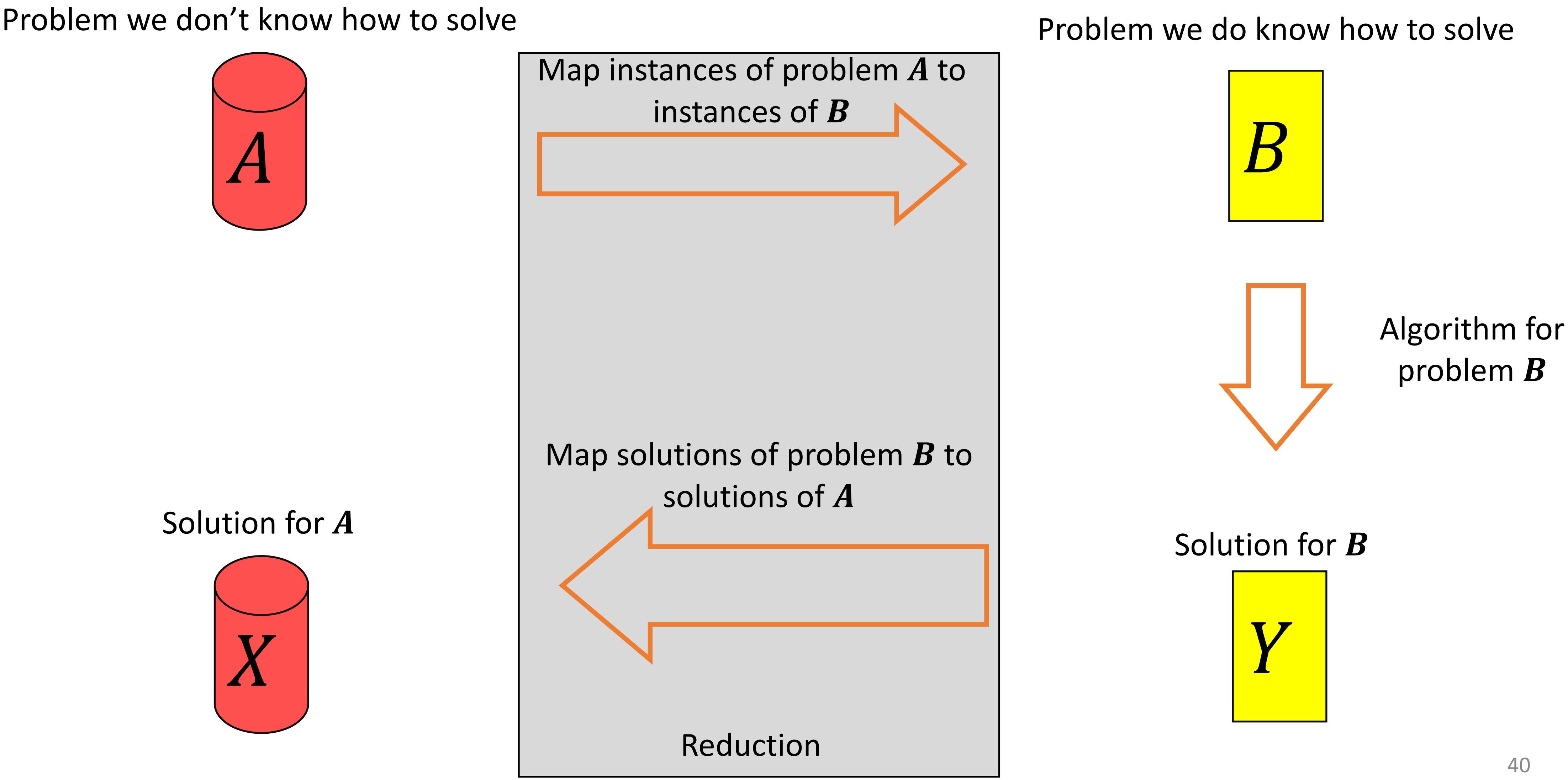
Blueprint:

1. Convert instance of Problem A into an instance of Problem B
2. Convert solution of Problem B back into a solution of Problem A

Both of these steps need to be efficient

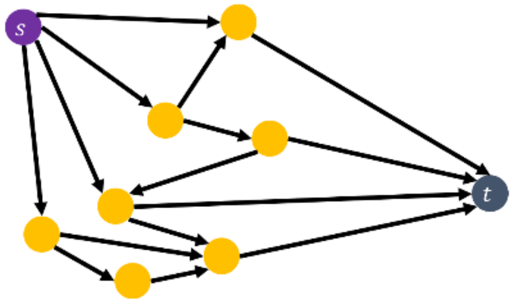
Analysis: Show that solution to Problem B can be used to obtain solution to Problem A

Reductions

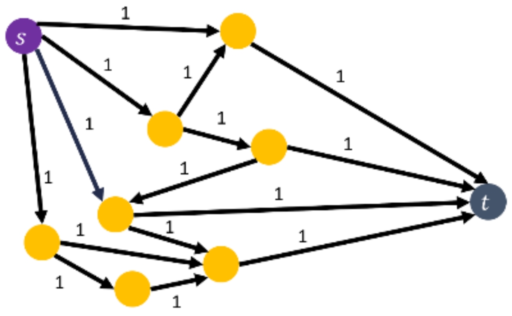

Problem we don't know how to solve
A
Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$
Problem we do know how to solve
B
Algorithm for problem $\boldsymbol{B}$
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$
Solution for $\boldsymbol{A}$
X
Solution for $\boldsymbol{B}$
Y
Reduction

Reduction Examples

Edge-disjoint paths



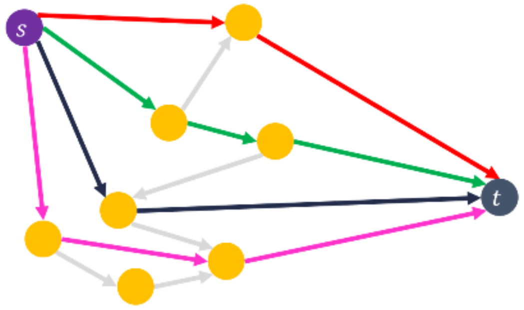
Max flow

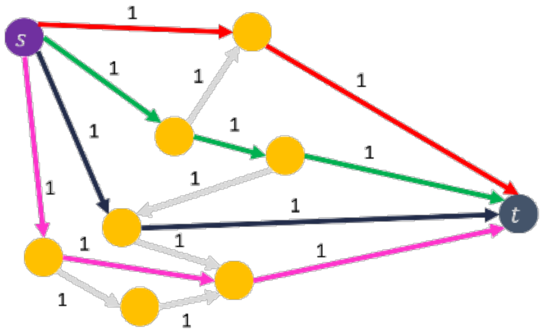


Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

Ford-Fulkerson

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

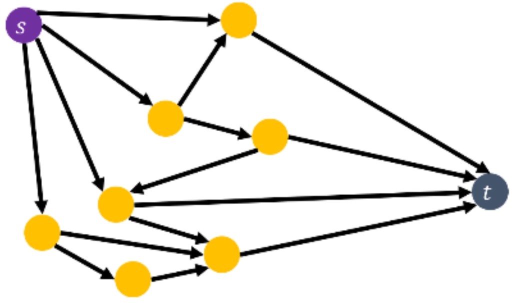




Reduction

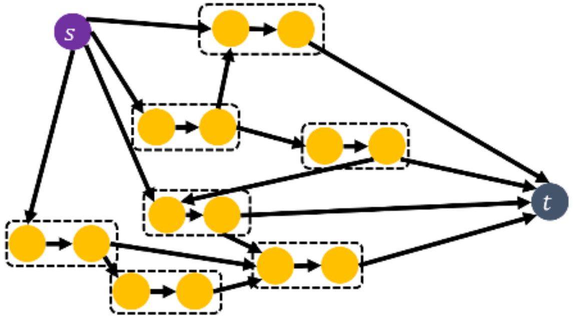
Reduction Examples

Vertex-disjoint paths

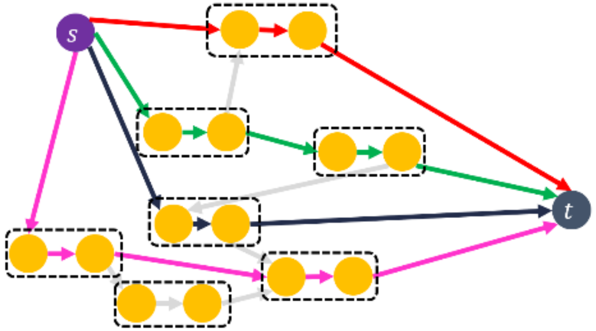


Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

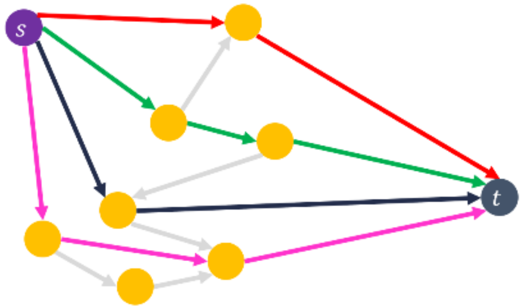
Edge-disjoint paths



Edge-disjoint paths algorithm



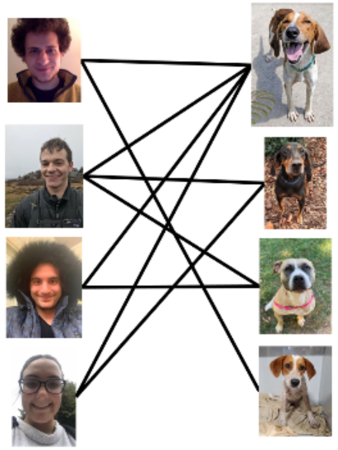
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

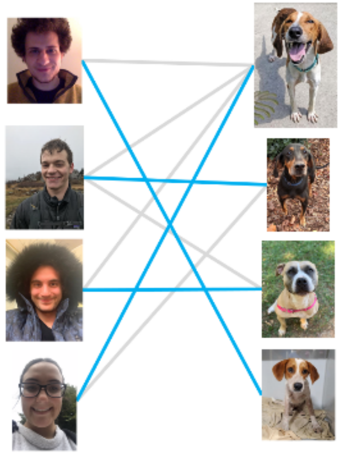


Reduction

Reduction Examples

Maximum bipartite matching



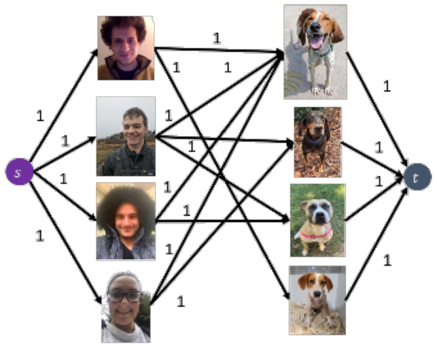


Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

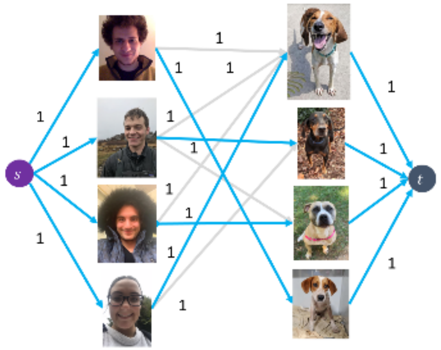
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

Reduction

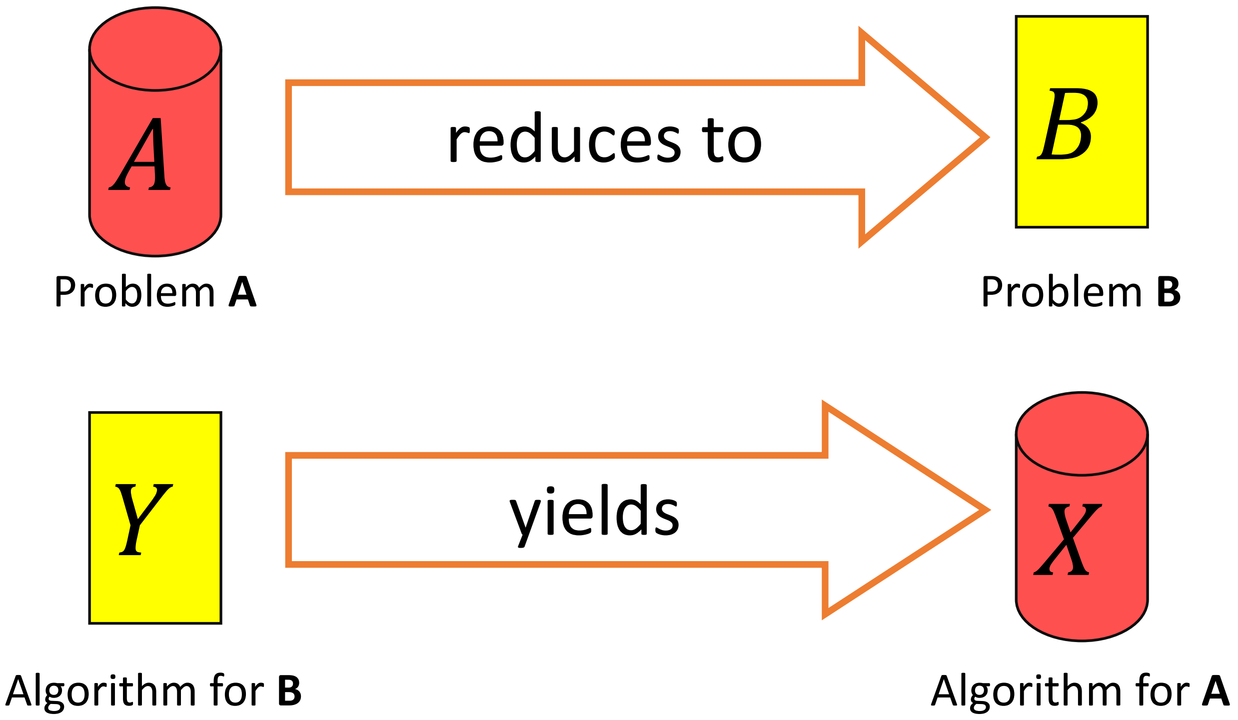
Max flow



Ford-Fulkerson

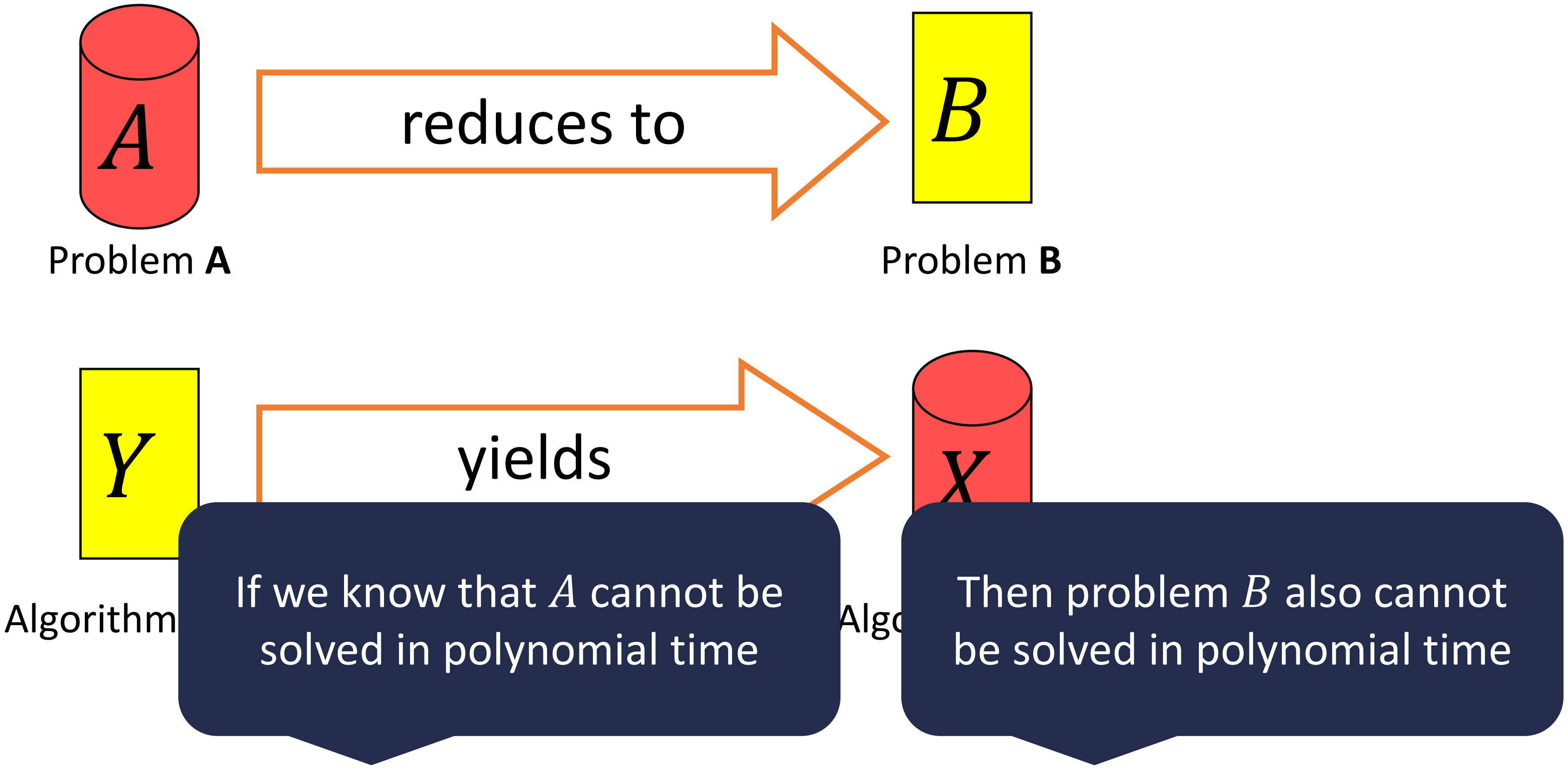


Understanding Reductions



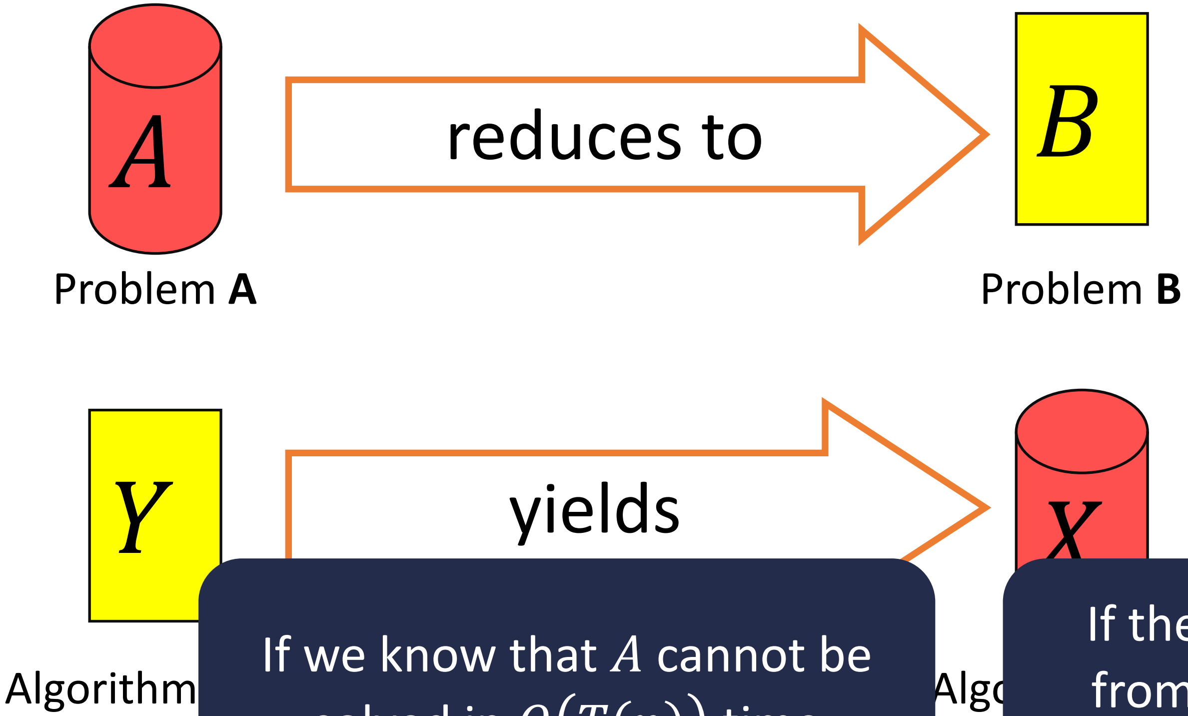
Implication: A is <u>no more difficult</u> than B
(denoted $A \leq B$)

Worst-Case Lower Bounds via Reductions



Implication: A is <u>no more difficult</u> than B
denoted $A \leq B$

Worst-Case Lower Bounds via Reductions

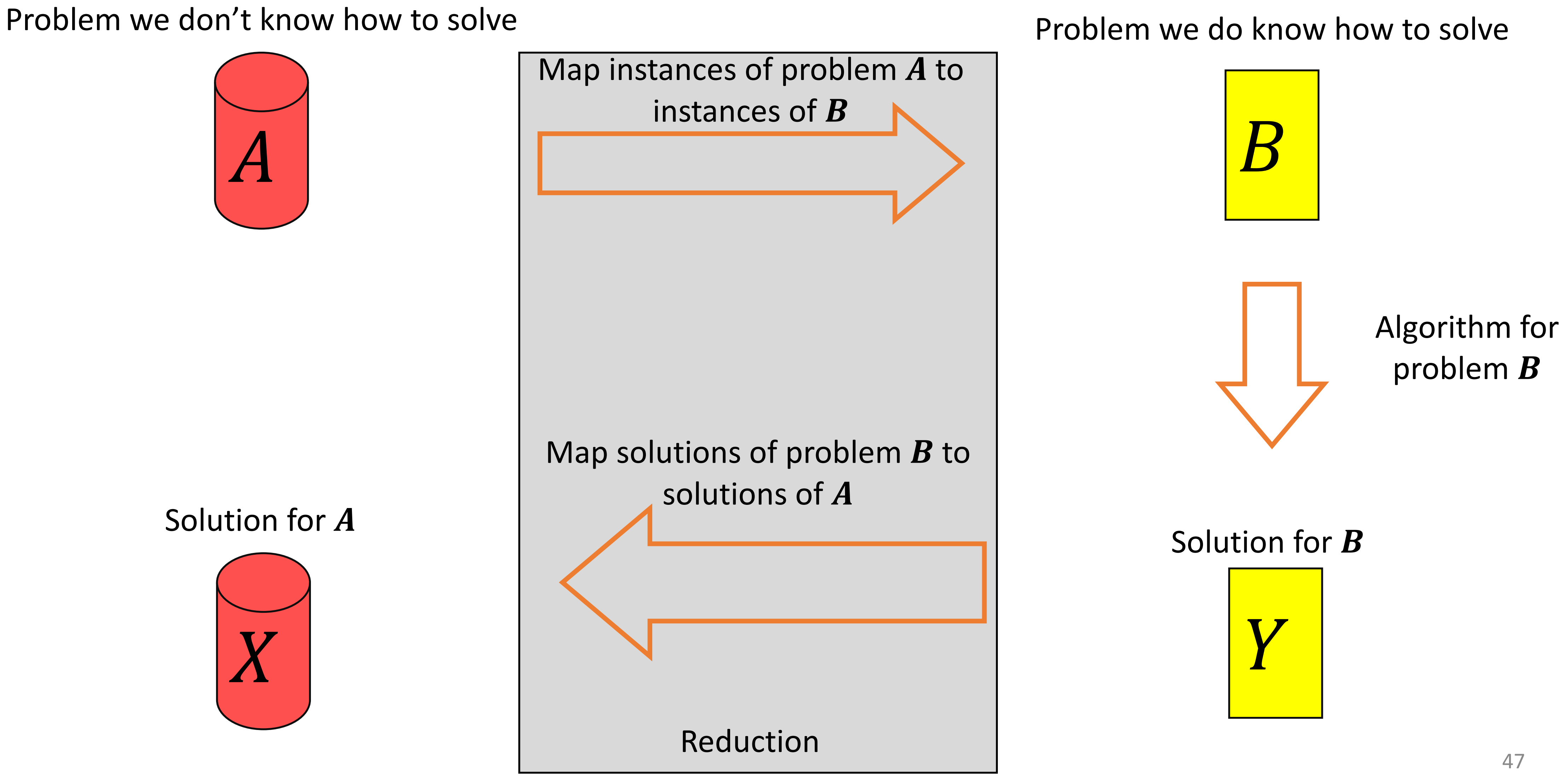


If we know that A cannot be solved in $O(T(n))$ time

If there is a $O(T(n))$ reduction from A to B, then B cannot be solved in O$(T(n))$ time

Implication: A is no more difficult than B
denoted $A \leq B$

Reductions


Problem we don’t know how to solve
A
Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$
Problem we do know how to solve
B
Algorithm for problem $\boldsymbol{B}$
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$
Solution for $\boldsymbol{A}$
X
Solution for $\boldsymbol{B}$
Y
Reduction