

Handouts on calendar  
(no physical copies)

# Max-Flow Min-Cut

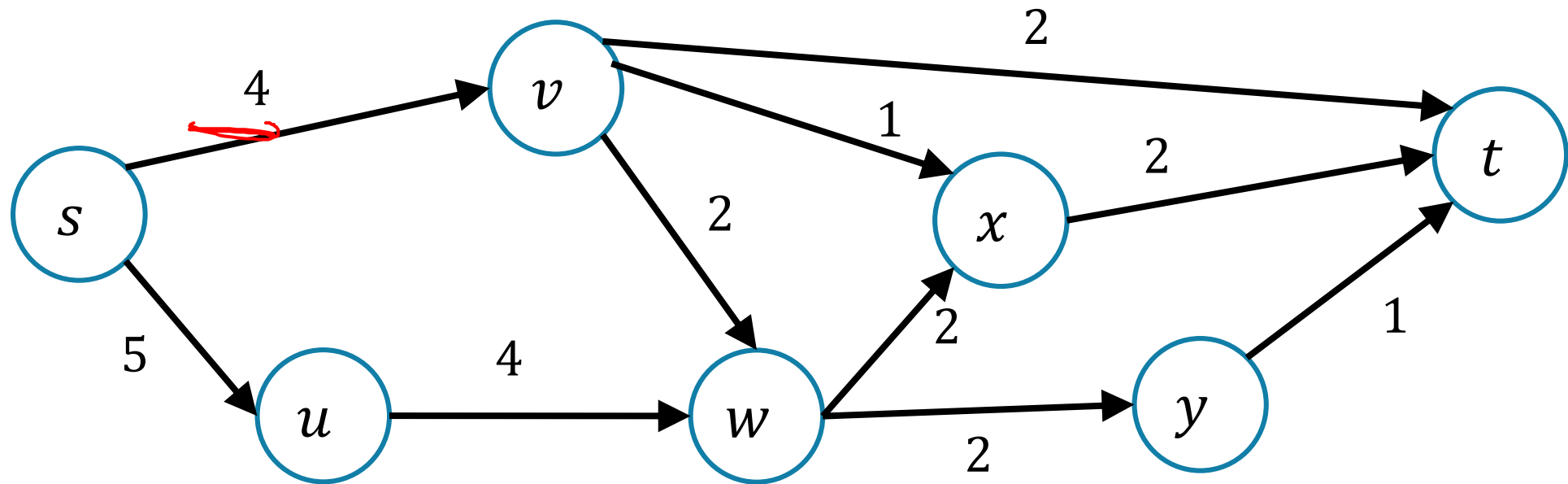
CSE 417 Fall 22  
Lecture 20

# Max Flow

We have a directed graph  $G$ , a source vertex  $s$  and a target vertex  $t$ .

We have some thing (water or data packets) we have to send from  $s$  to  $t$ .

Every edge has a capacity, it can only handle so many.

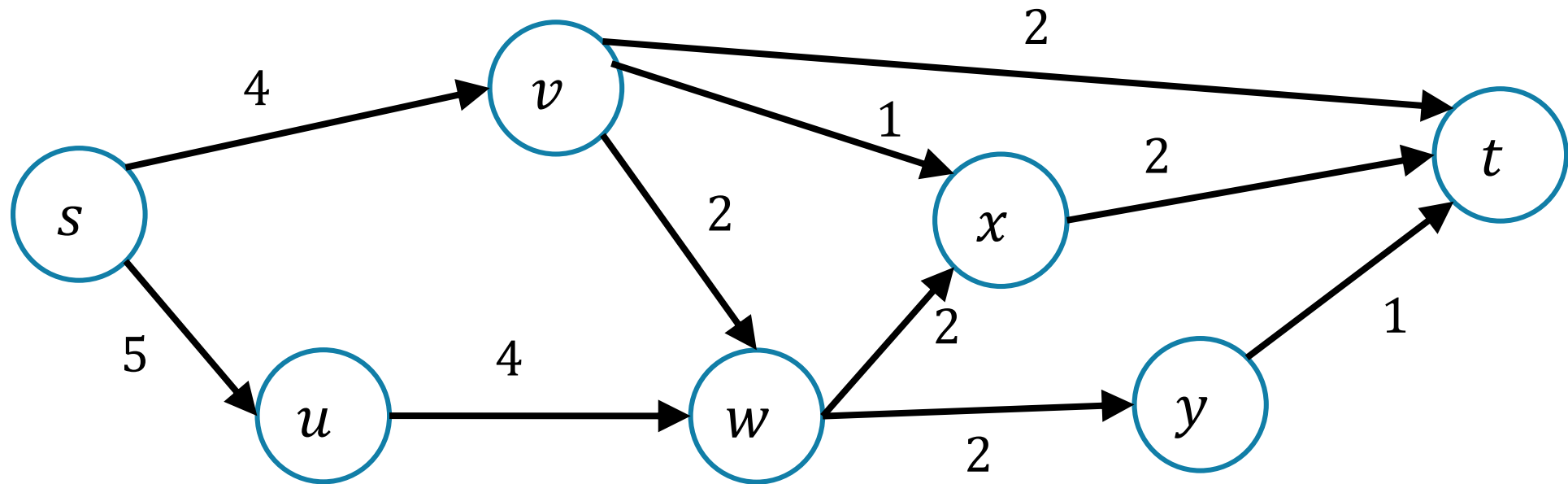


# Flows

A **flow** moves units of water from  $s$  to  $t$ .

Water can only be created at  $s$  and only disappear at  $t$ .

And you cannot move more water than the capacity on any edge.

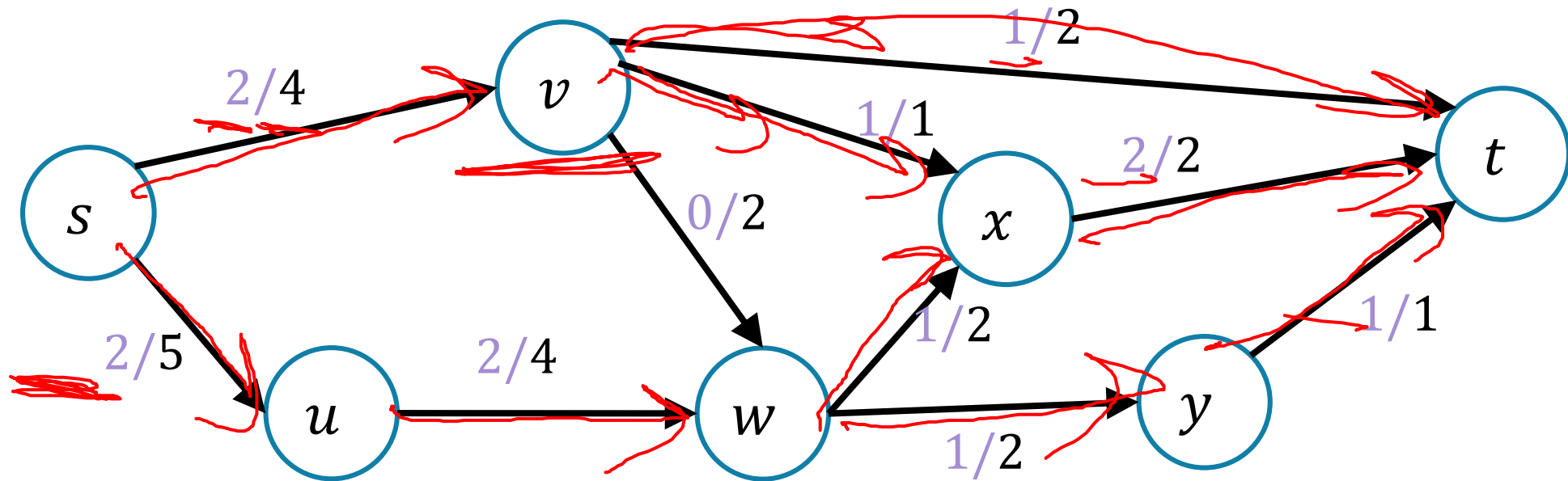


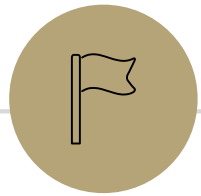
# Flows

A **flow** moves units of water from  $s$  to  $t$ .

Water can only be created at  $s$  and only disappear at  $t$ .

And you cannot move more water than the capacity on any edge.





# Finding Max Flows

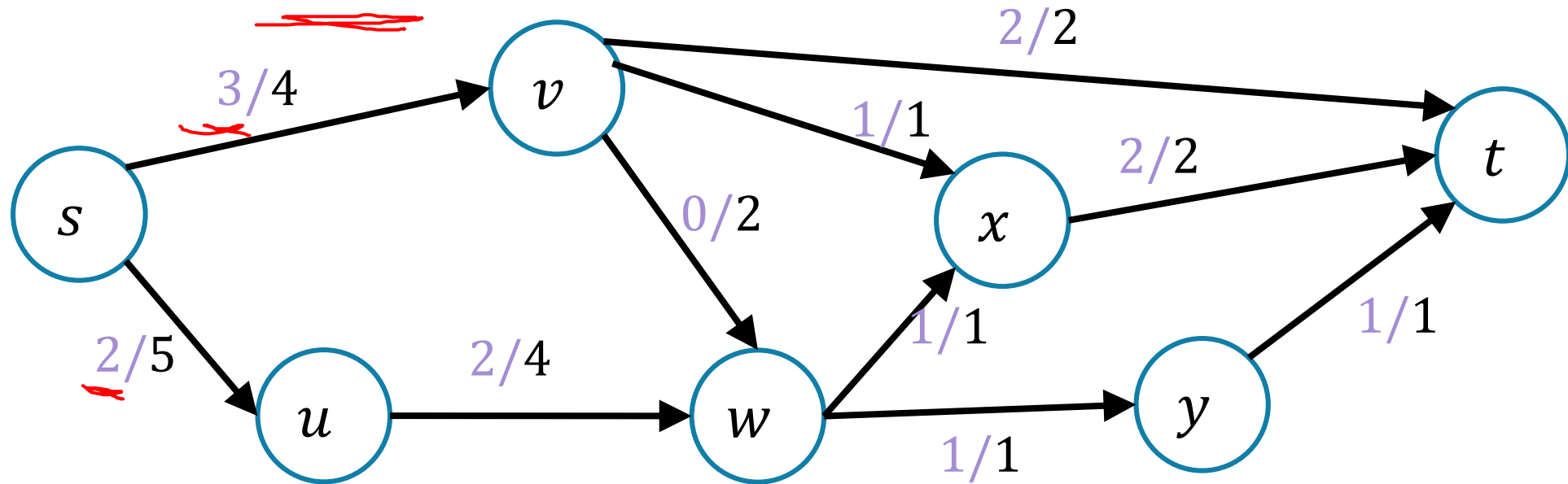
---

# Flows

The **value** or **size** of a flow is the net flow leaving  $s$

Or, equivalently the net flow entering  $t$ .

Value of this flow is 5.



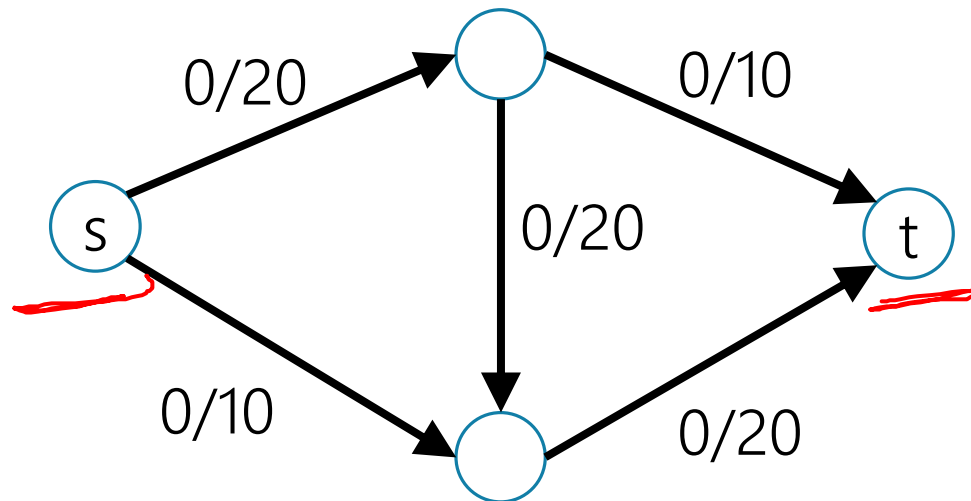
# Finding the Max Flow

Idea: find a path that we can push flow along.

Start from s, follow an edge with remaining capacity until you get to t

How much flow can you add? The minimum **remaining** capacity on any of the edges we used.

Does this work?



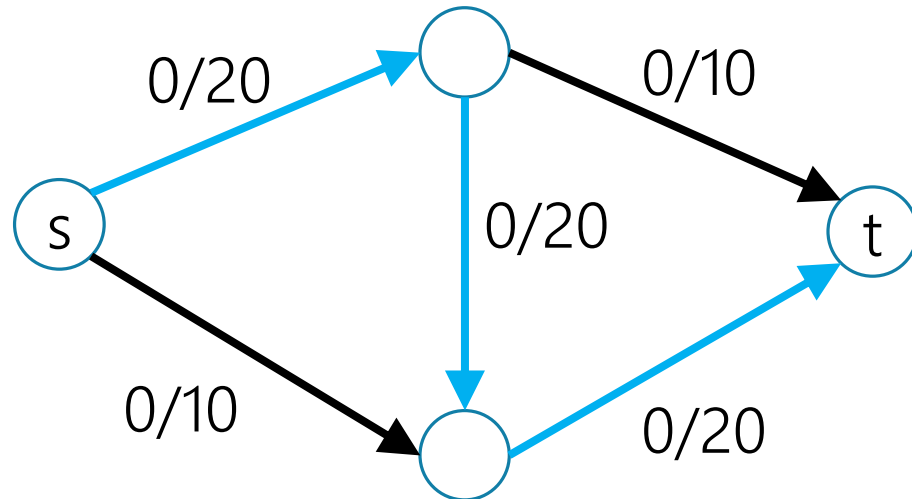
# Finding the Max Flow

Idea: find a path that we can push flow along.

Start from  $s$ , follow an edge with remaining capacity until you get to  $t$

How much flow can you add? The minimum **remaining** capacity on any of the edges we used.

Does this work?



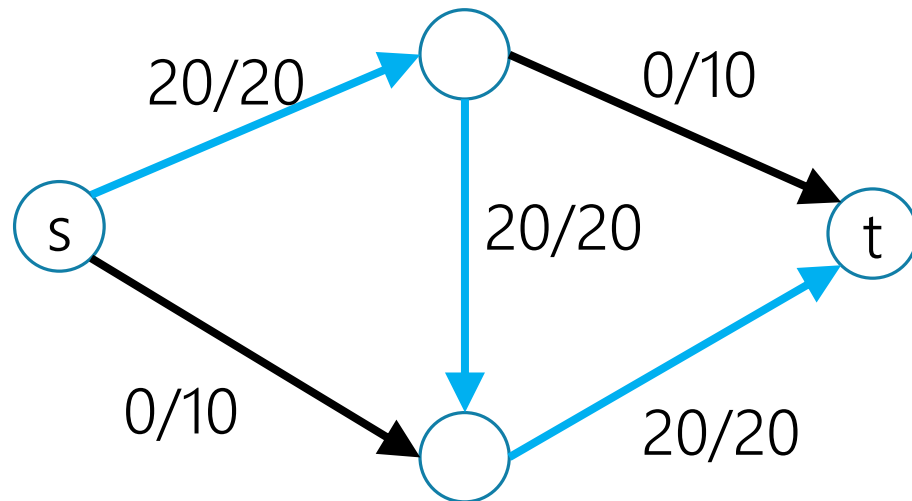
# Finding the Max Flow

Idea: find a path that we can push flow along.

Start from  $s$ , follow an edge with remaining capacity until you get to  $t$

How much flow can you add? The minimum **remaining** capacity on any of the edges we used.

Does this work?



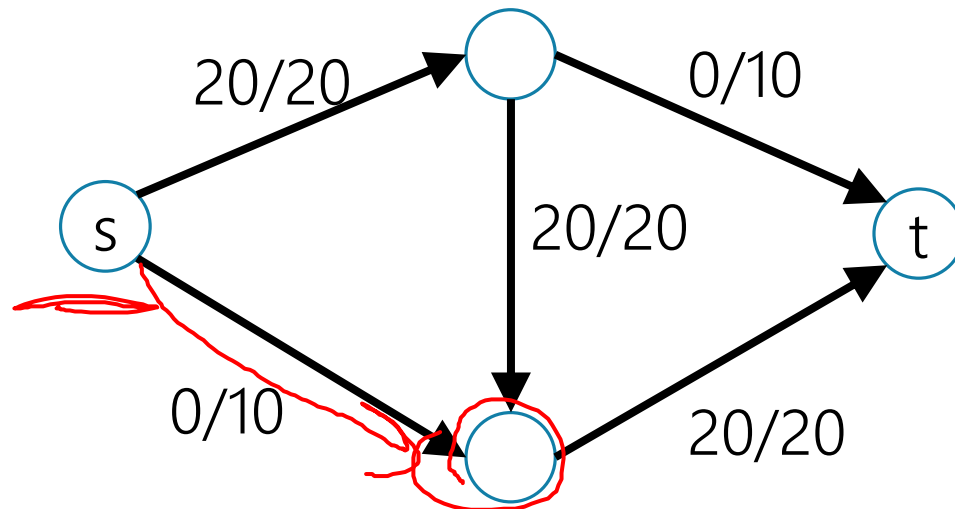
# Finding the Max Flow

Idea: find a path that we can push flow along.

Start from  $s$ , follow an edge with remaining capacity until you get to  $t$

How much flow can you add? The minimum **remaining** capacity on any of the edges we used.

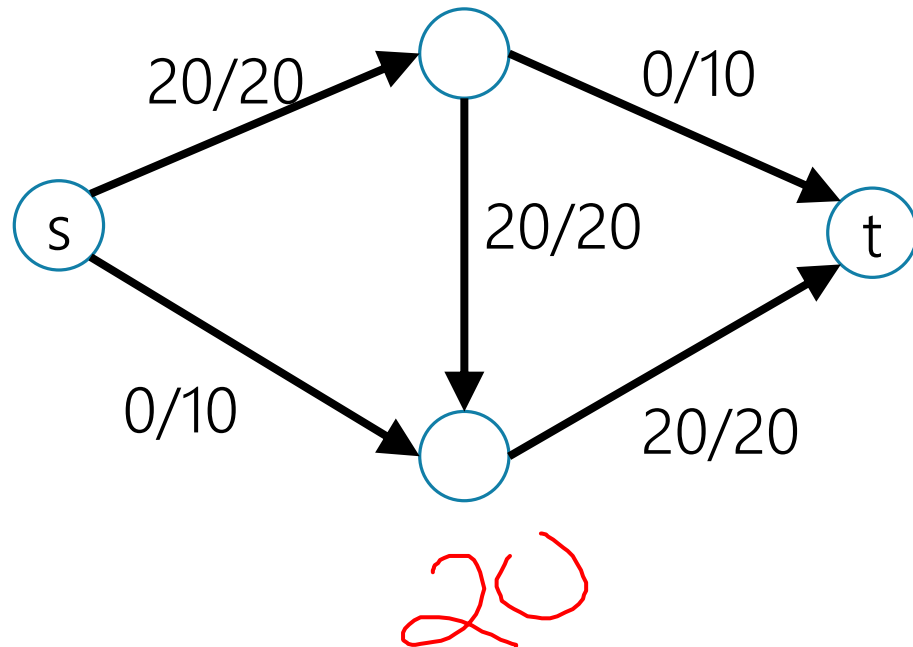
Does this work?



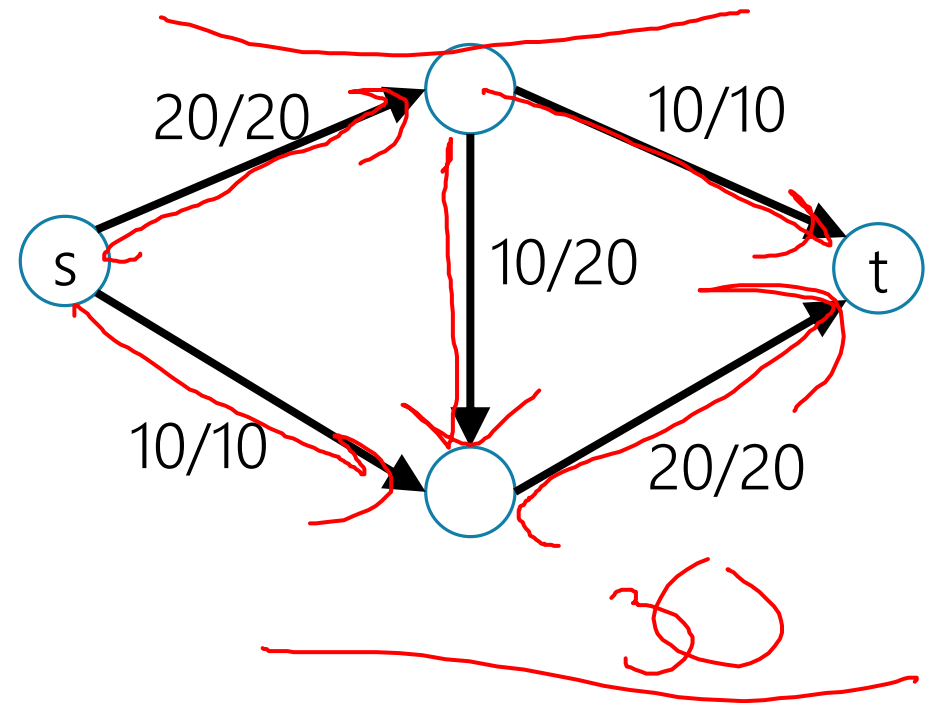
# Finding the Max Flow

We find a valid flow...but it might not be the maximum one.

What our first idea found



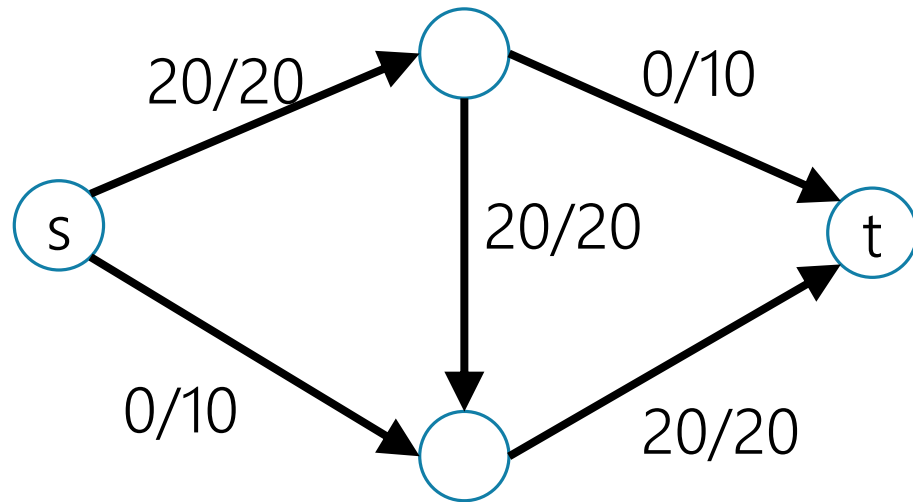
The true optimum



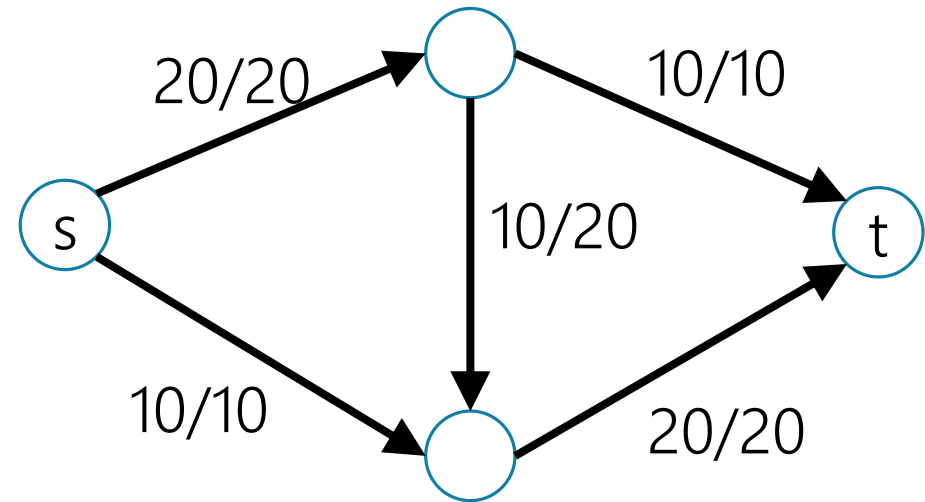
# Finding the Max Flow

How would we fix what our first idea found?

What our first idea found



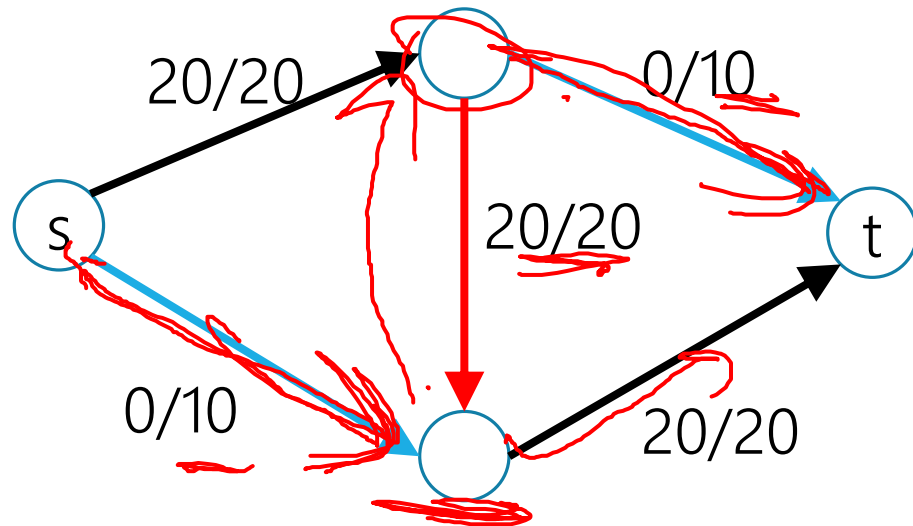
The true optimum



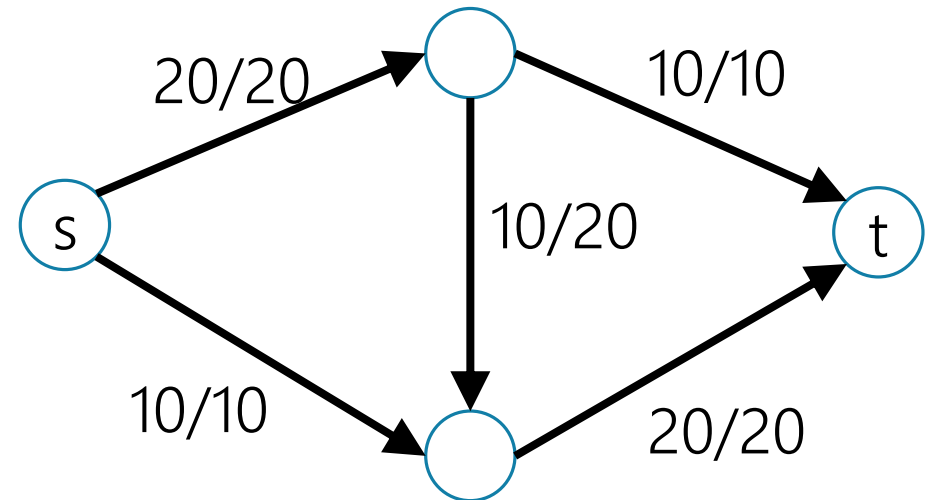
# Finding the Max Flow

How would we fix what our idea found?

What our first idea found



The true optimum

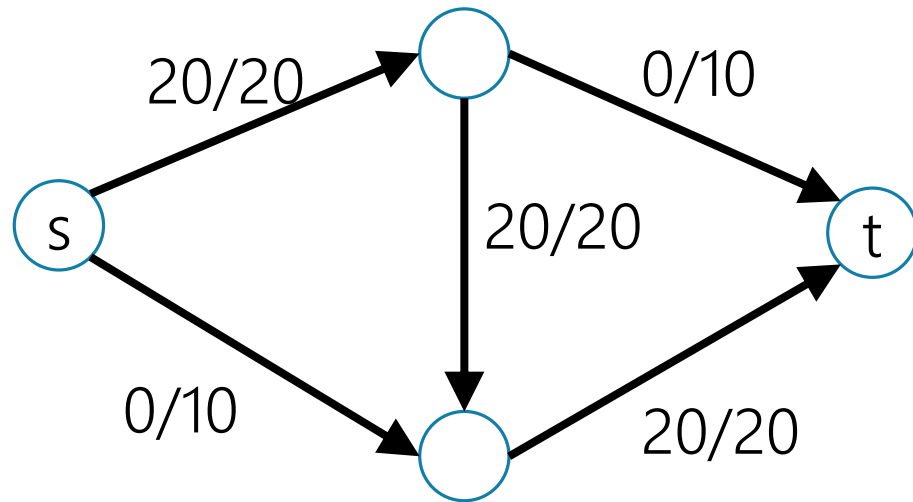


We can send extra flow from  $s$  and decrease the flow along another edge with existing flow  
And we'll still get a valid flow!

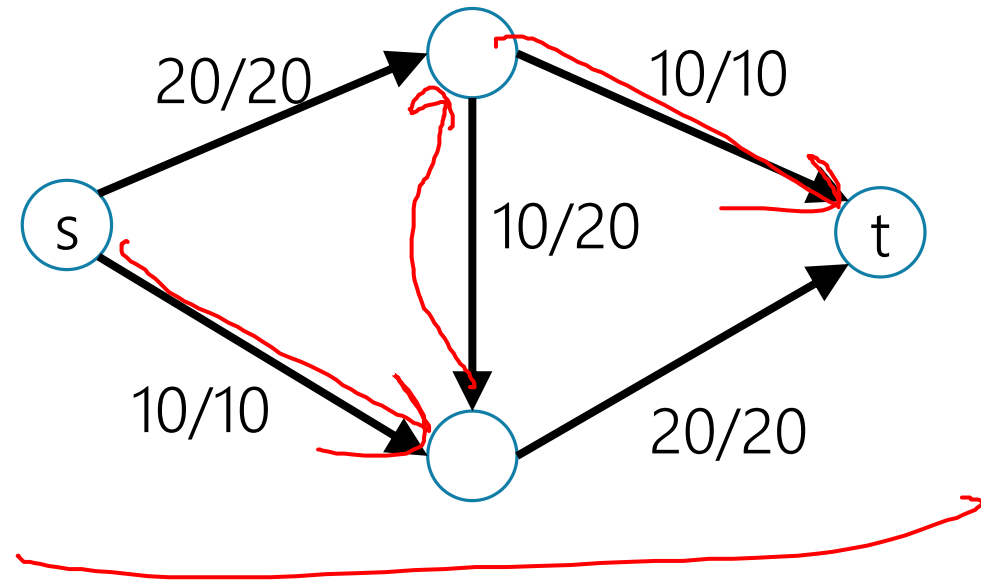
# Finding the Max Flow

How would we fix what our first idea found?

What our first idea found



The true optimum



# Fixing Our First Idea

When can we send flow in a direction?

When there is unused capacity OR

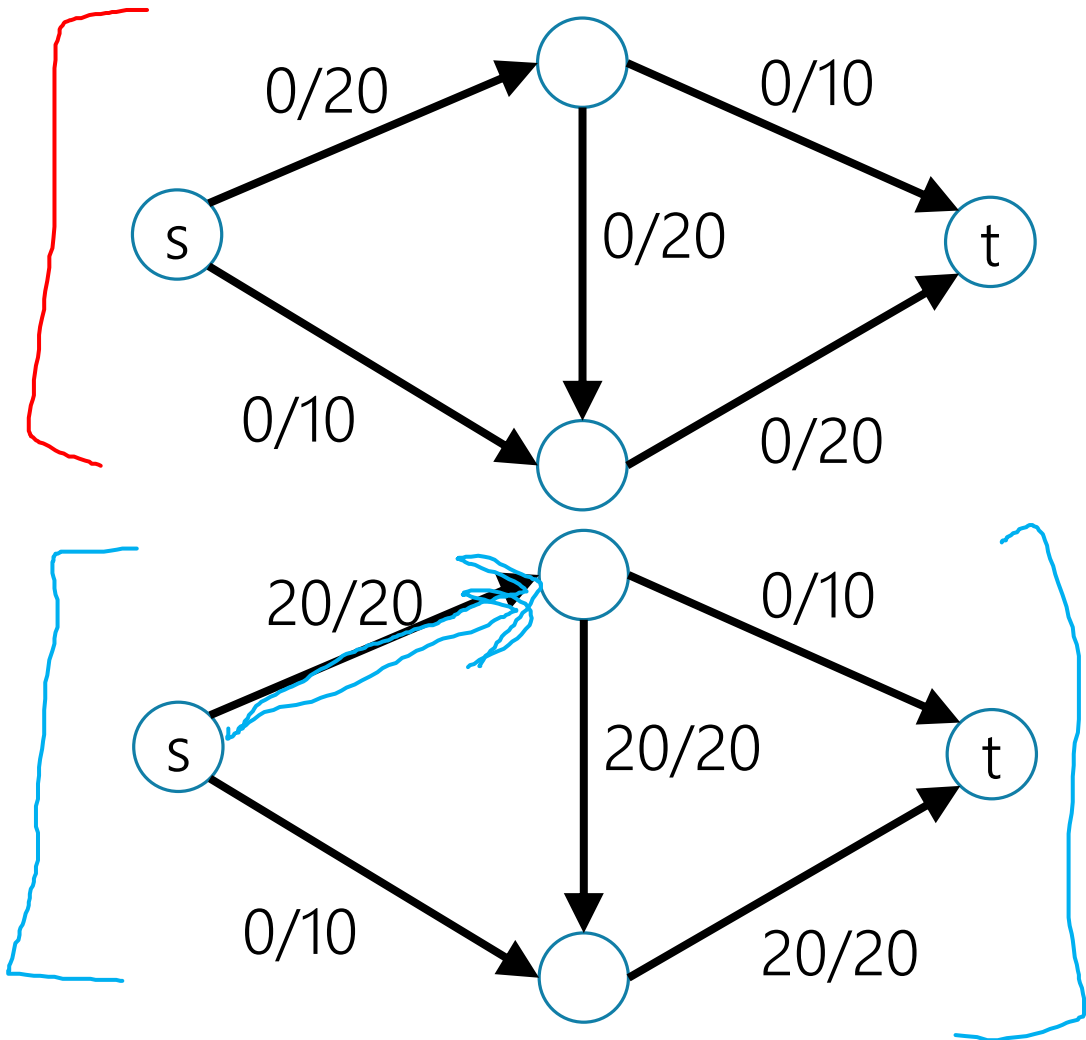
When there is flow going the other direction we can delete it.

Let's update the graph to take that into account.

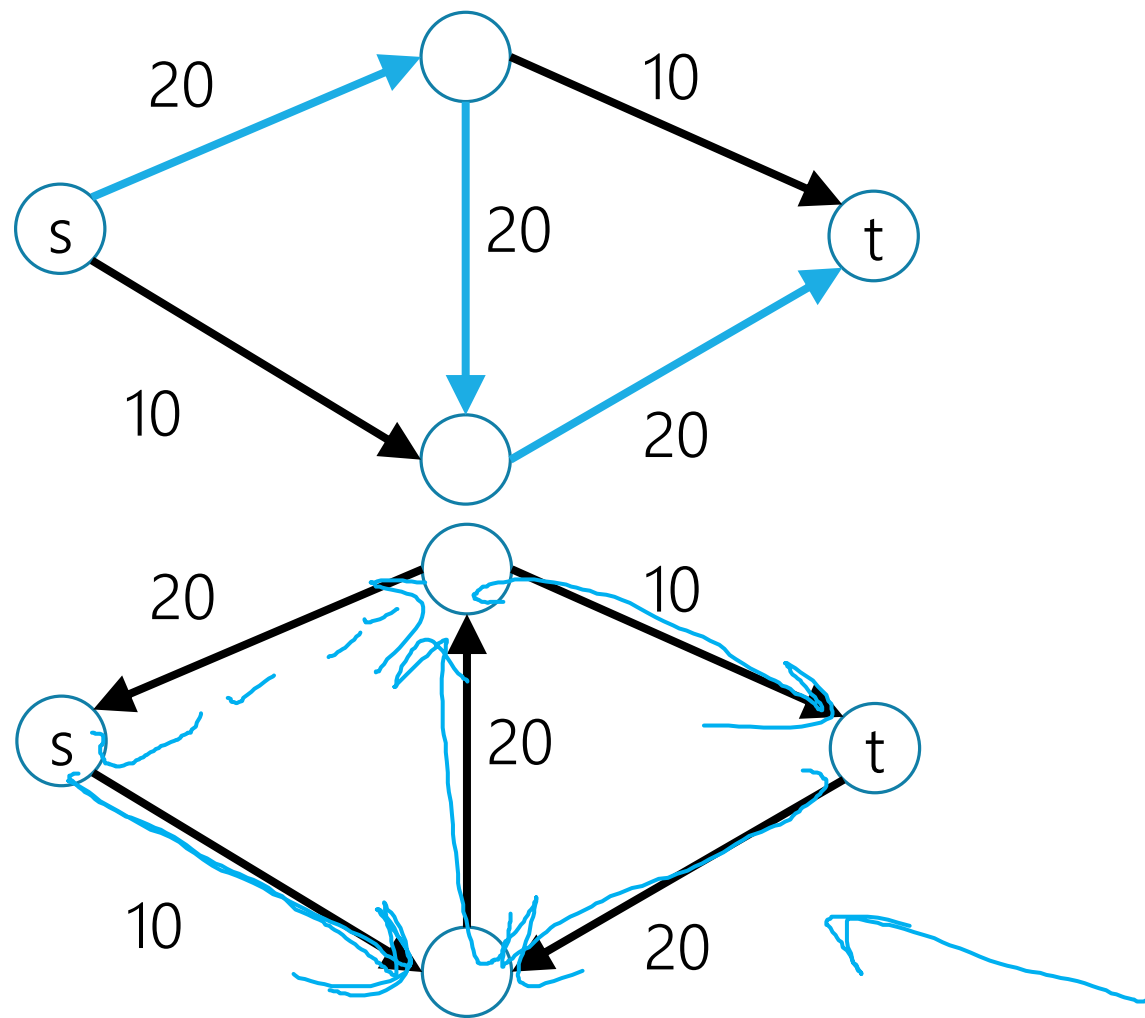
Have an edge weight demonstrate the changeable capacity (the "residual")

# An Example

Graph, with flow

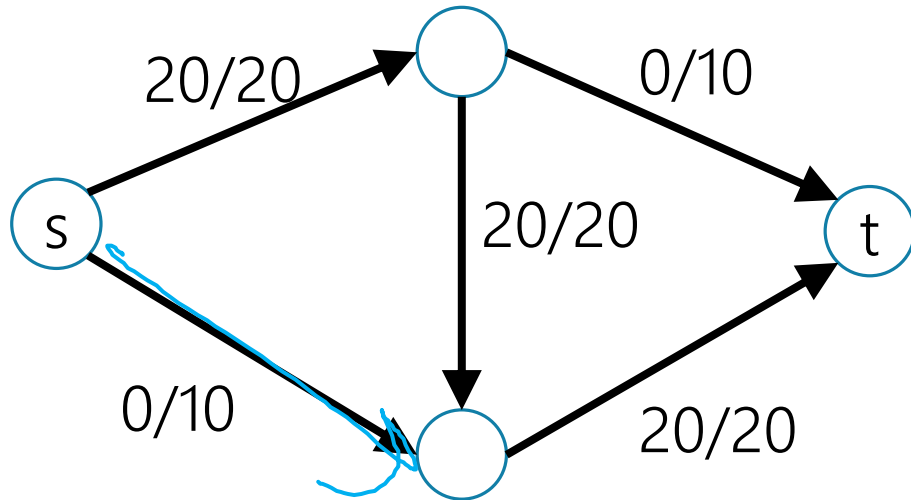


"Residual Graph"

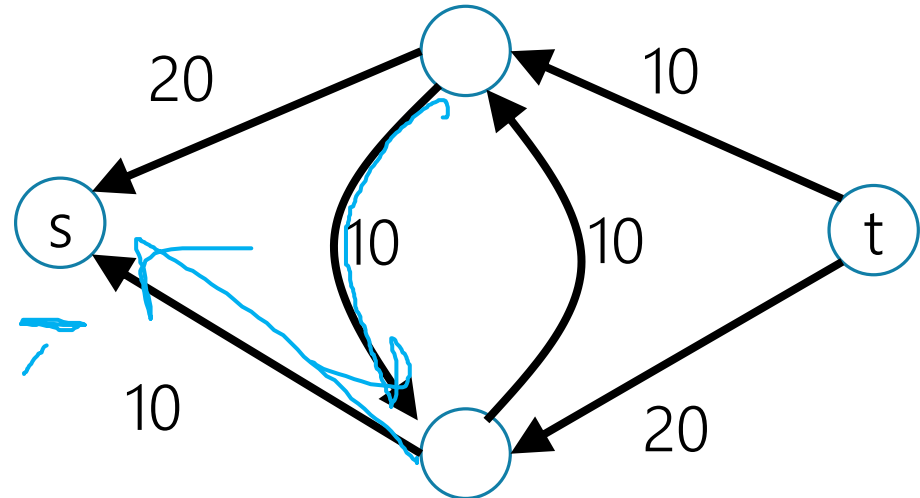
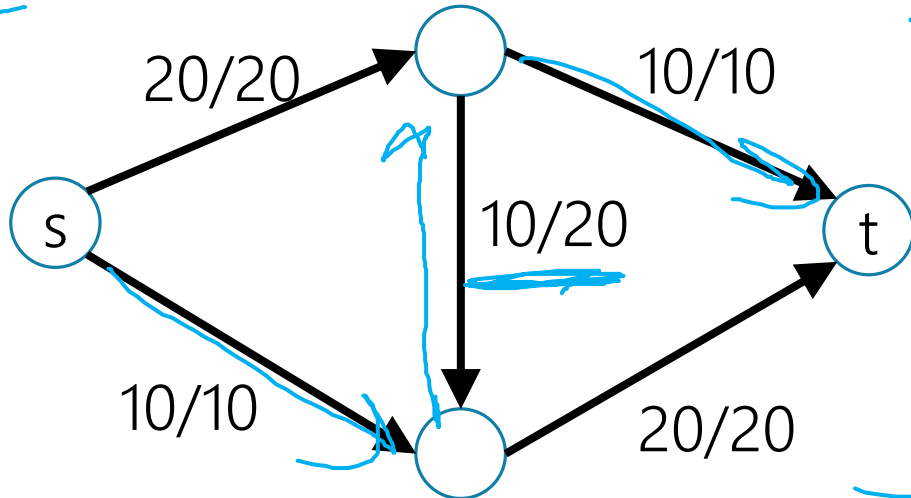
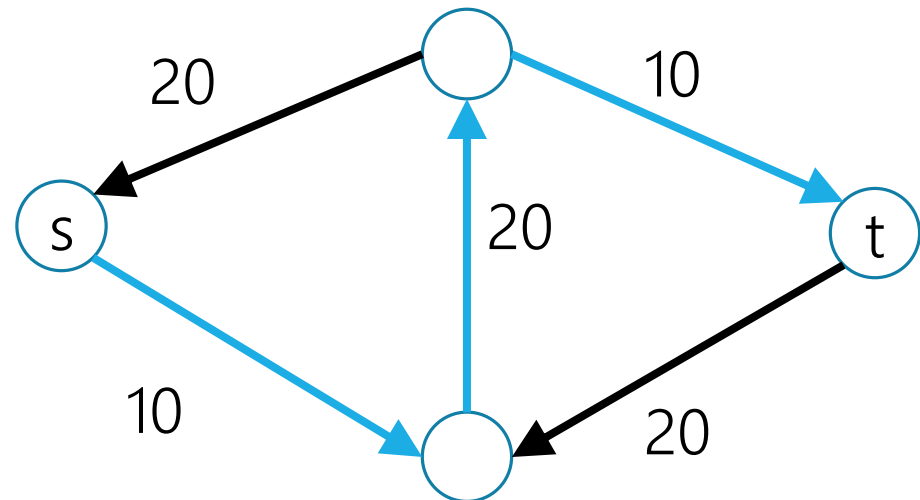


# An Example

Graph, with flow



"Residual Graph"



# Residual Graph

In general:

If the original graph has an edge  $(u, v)$  of capacity  $c$ , and the flow sends  $f_{u,v}$  along  $(u, v)$ :

Include  $(u, v)$  in the residual with capacity  $c - f_{u,v}$  as long as  $c - f_{u,v} > 0$  (if equal to zero, don't include the edge)

Include  $(v, u)$  [the edge going in the reverse direction] with capacity  $f_{u,v}$  as long as  $f_{u,v} > 0$

# Ford-Fulkerson Algorithm

While(flow is not maximum)

Run BFS in residual graph starting from  $s$ .

Record predecessors to find an  $s, t$ -path

Iterate through path, finding  $c$  minimum residual capacity on path.

Add  $c$  to every edge on path in flow

Update residual graph

# Ford-Fulkerson Algorithm

While(true)

Run BFS in residual graph starting from  $s$ .

Record predecessors to find an  $s, t$ -path

If you don't reach  $t$ , break. //otherwise you can still augment

Iterate through path, finding  $c$  minimum residual capacity on path.

Add  $c$  to every edge on path in flow

Update residual graph

Assuming  $E \geq V$  (isolated vertices won't affect the flow, so this is reasonable).

Running Time:  $O(E)$  per iteration. Number of iterations?

# Ford-Fulkerson Algorithm

If we have all integer capacities at the start...

The residual graph will always have integer capacities.

Why? The minimum capacity on the first path is an integer.

So we subtract or add integers to the residual graph.

And the result is more integers!

So in every iteration we add at least 1 unit of flow!

# Ford-Fulkerson Algorithm

While(true)

Run BFS in residual graph starting from  $s$ .

Record predecessors to find an  $s, t$ -path

If you don't reach  $t$ , break. //otherwise you can still augment

Iterate through path, finding  $c$  minimum residual capacity on path.

Add  $c$  to every edge on path in flow

Update residual graph

Running Time:  $O(E)$  per iteration. Number of iterations?  $O(f)$ , where  $f$  is the value of the maximum flow. Total  $O(Ef)$

$O(Ef)$  "final answer"

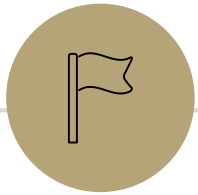
# Wait... $f$ ?

We haven't seen a running time before that depends on the answer

Normally, we want the running time directly in terms of the input.

There are tricks to speed up Ford-Fulkerson so you don't take too long if  $f$  is really big.

Some optional content about this at the end of this deck.



# Minimum Cut



# What's a Cut?

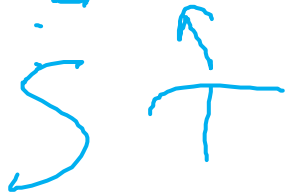
For directed graphs (like we have here)

An  $(s, t)$ -cut, is a split of the vertices into two sets  $(S, T)$

So that  $s$  is in  $S$ ,  $t$  is in  $T$ ,

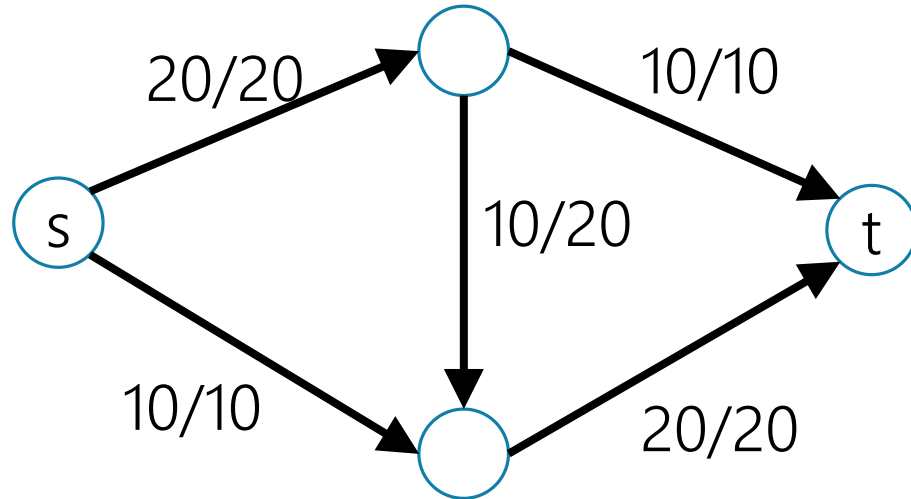
and every other vertex is in exactly one of  $S$  and  $T$ .

The capacity of a cut (or size of a cut) is the capacity of the edges going from  $s$  to  $t$  (don't count capacity from  $t$  to  $s$ ).

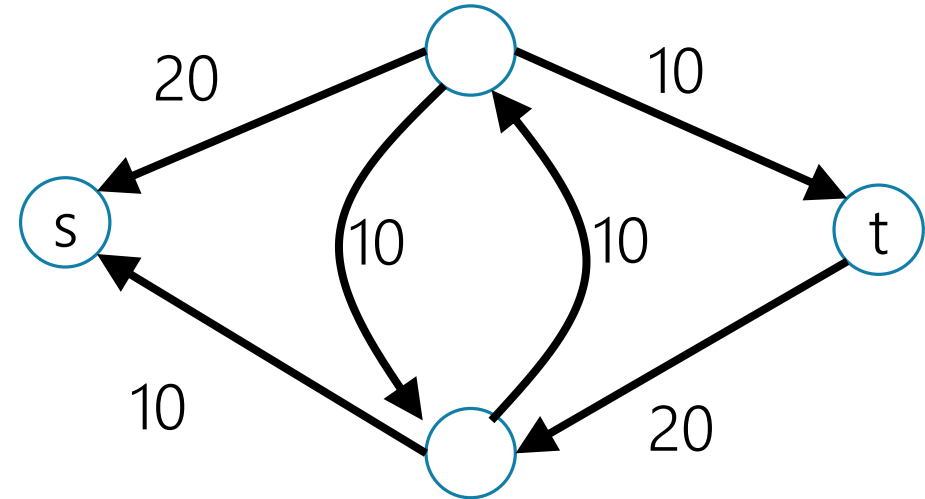


# An Example

Graph, with flow



"Residual Graph"

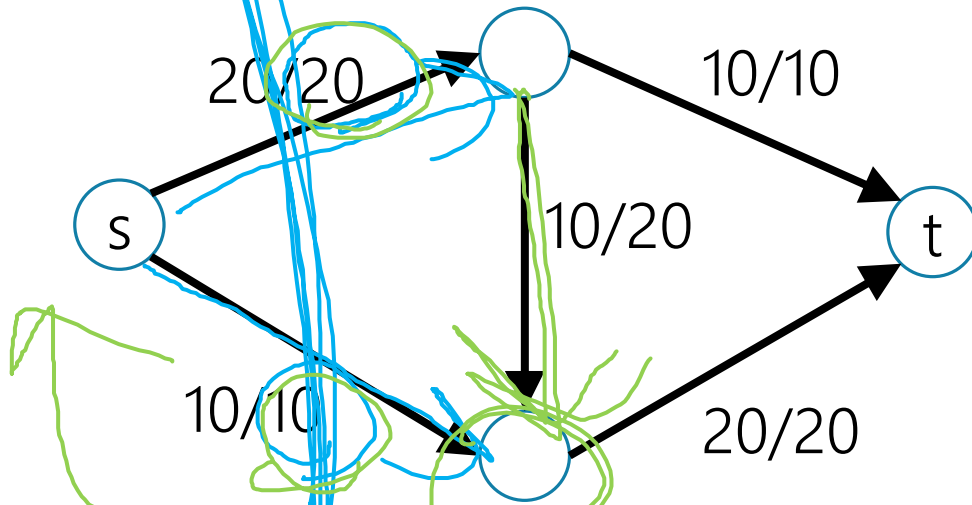


We can't get from  $s$  to  $t$  anymore in the residual graph. We're done! That's a maximum flow.

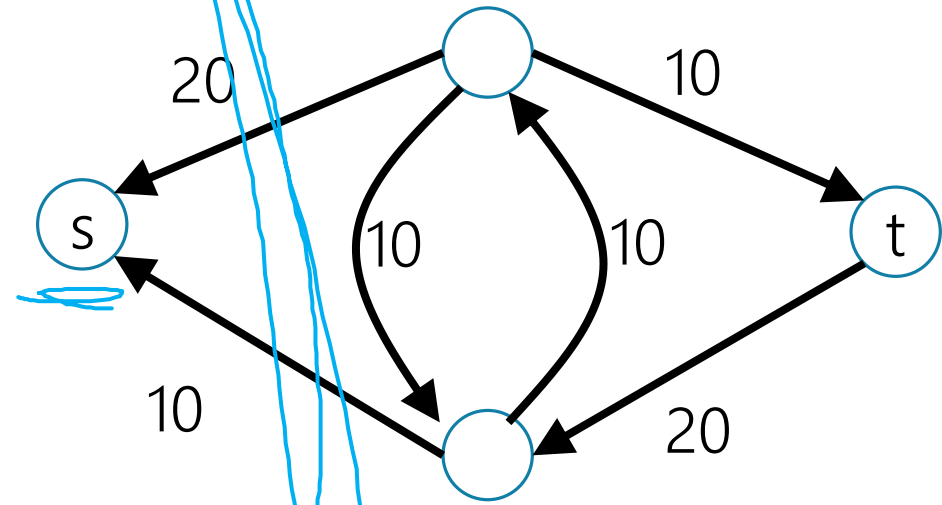
But...why?

# An Example

Graph, with flow



"Residual Graph"



Cut is  $(s, V - s)$  (i.e.  $s$  on one side, everything else on the other)

Edges from  $s$  to everything else? Capacity 30

We can't get more than 30 units of flow from  $s$  to  $t$ . Because it all (simultaneously) must cross from one side to the other.

# How Do We Know?

How do we know the algorithm is done?

When we can't we get from  $s$  to  $t$ ?

We've **cut** the (residual) graph.

$s$  and all the vertices you can reach from it on one side  
and  $t$  and all the vertices  $s$  can't reach on the other.

Take a look at the edges spanning the cut in the original graph.

In our first graph, that capacity was equal to the value of the max flow.

# Finding the min-cut

Maintain the residual graph.

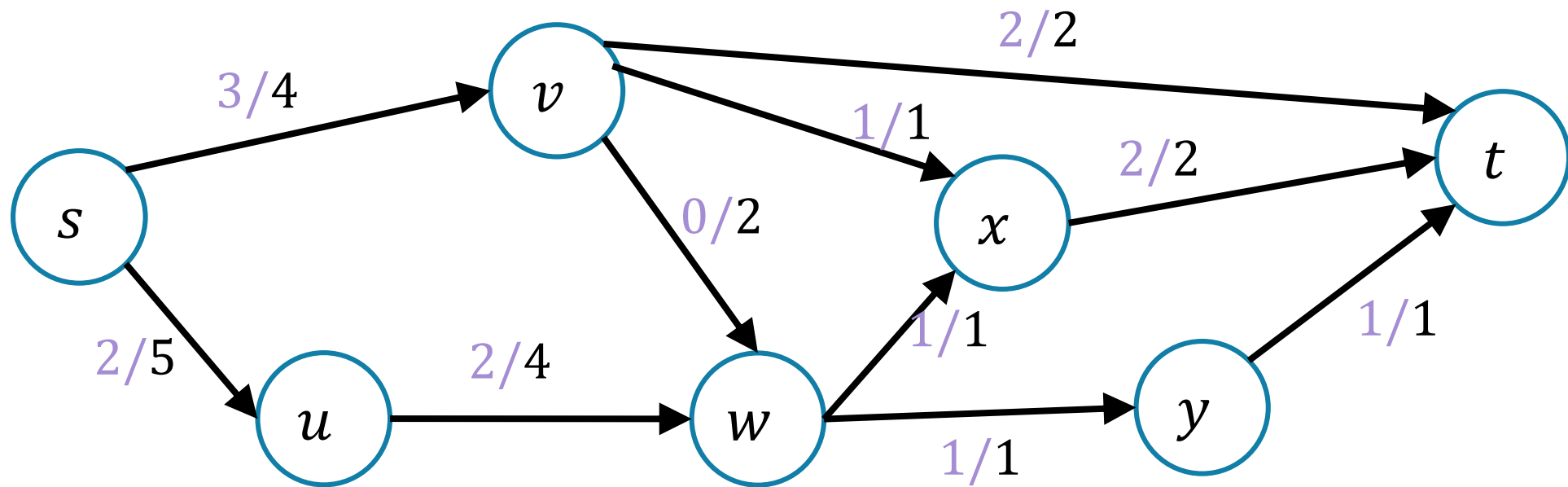
When you search from  $s$  and can't get to  $t$ :

$s$  and everything you can reach from  $s$  is on one side of the cut  
 $t$  and everything you can't reach from  $s$  is on the other side.

# Another Example

We started lecture with this flow.

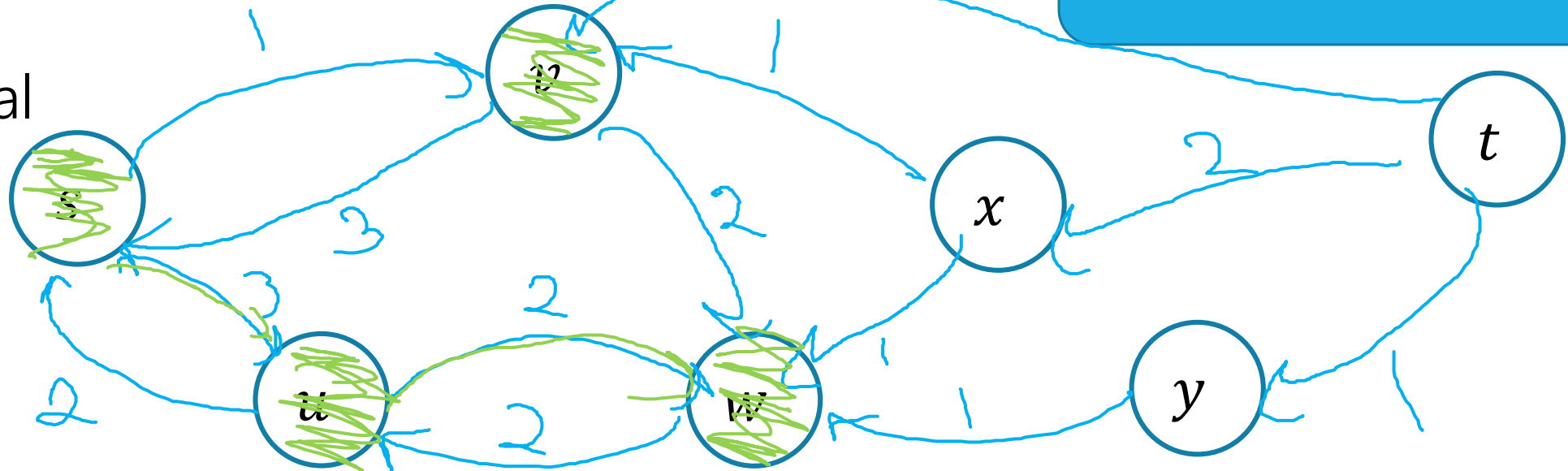
What's the residual graph? What's the cut?



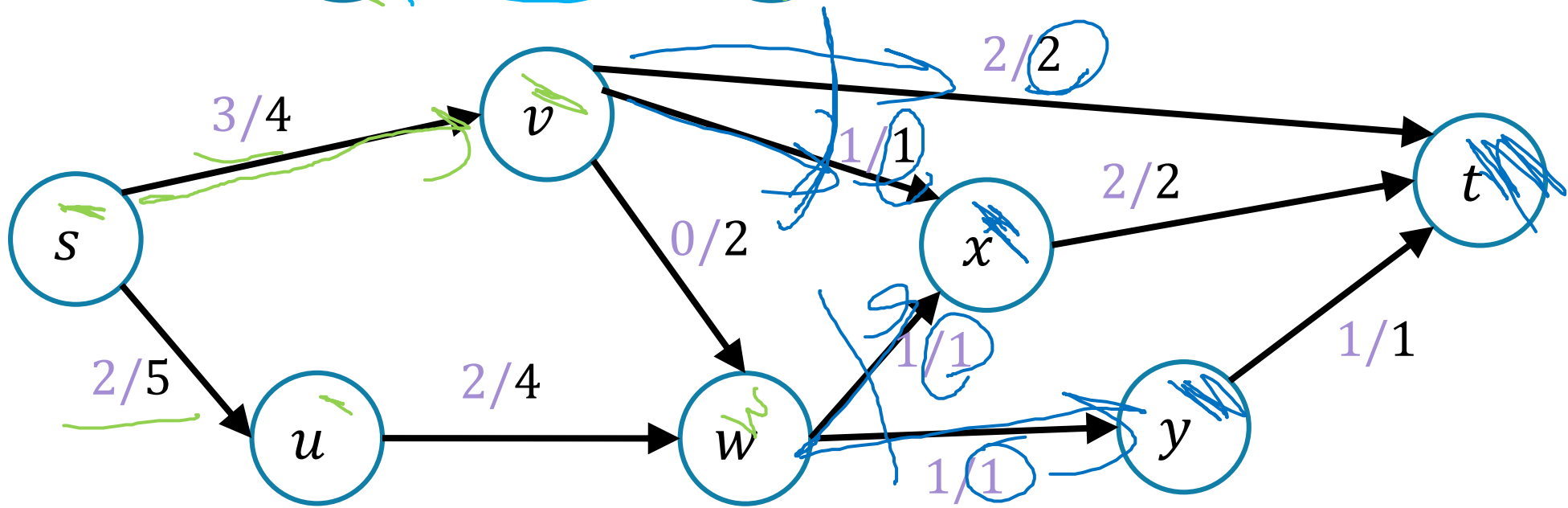
# Another Example

Pollev.com/robbie

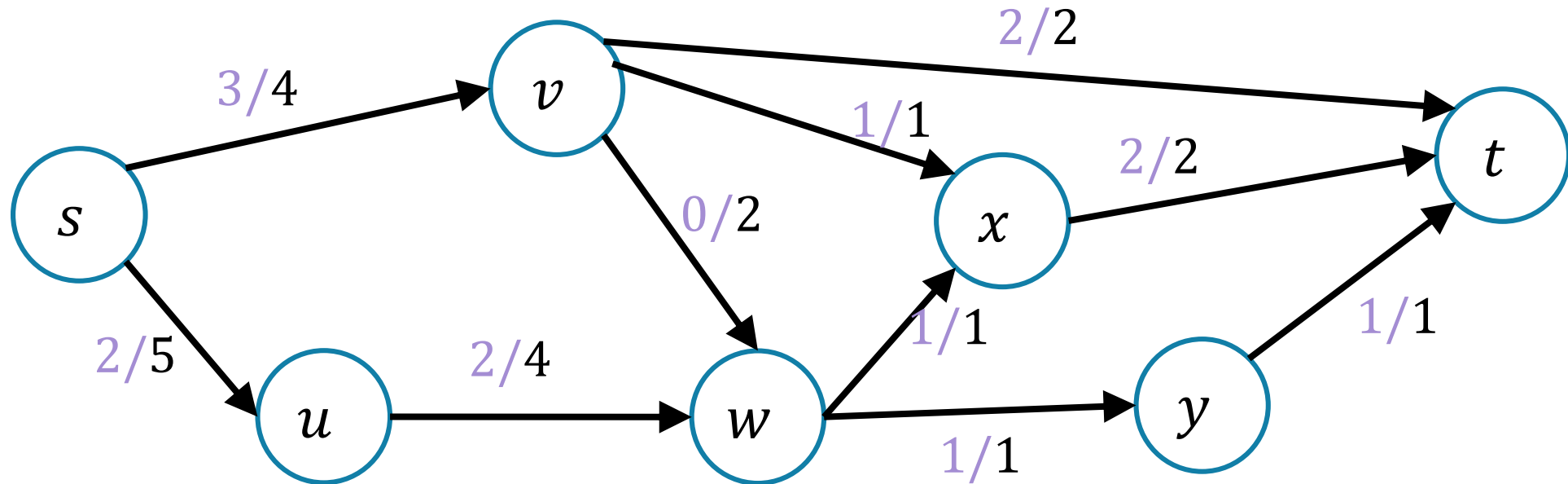
Residual



Flow



# Another Example



$(\{s, u, v, w\}, \{x, y, t\})$  is the cut in the residual graph.

What edges span?

$(v, x), (v, t), (w, x), (w, y)$

Total capacity? 5. What's the value of the flow? 5

# Max Flow-Min Cut Theorem

## Max-Flow-Min-Cut Theorem

The value of the maximum flow from  $s$  to  $t$  is equal to the value of the minimum cut separating  $s$  and  $t$ .

No proof, but here's some intuition:

Every cut is an upper-bound on the value of the flow (you can't have the total flow value higher than the cut).

Cut will be saturated (i.e. full flow going through all the edges) otherwise residual graph will have another edge. So flow and cut are equal values.

# So What?

Max-flow and min-cut are each interesting algorithmic problems.

They were first studied in the 1950s

The U.S. military wanted to know how much the Soviets could ship on their rail network.

And also which rail lines they would target to *disrupt* the network.

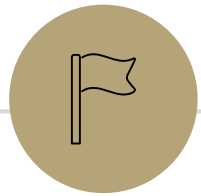
# So What?

Great quick check for if you've found the maximum flow (or min-cut).  
Check the other and see if the value is the same!

We'll see examples next time of max-flow used for modeling. In those cases the min-cut can be interpreted as a "barrier" to a good assignment.

It's also a nice example of **duality**

If you know what a "dual linear program" is – flows and cuts are dual LPs.



# Applications



# Applications of Max-Flow-Min-Cut

Max-Flow and Min-Cut are useful if you work for the water company...  
But they're also useful if you don't.

The most common application is assignment problems.

You have jobs and people who can do jobs – who is going to do which?

Big idea:

Let one unit of flow mean “assigning” one job to a person.

# Hey Wait...

Isn't this what stable matching is for?

Stable matching is very versatile, and it lets you encode preferences.

Max-flow assignment is even more versatile on the types of assignments.

But there's not an easy way to encode preferences.

# Example Problem

You and your housemates need to decide who is going to do each of the chores this week.

Some of your housemates are unable to do some chores.

Housemates: 1,2,3

Chores:

**A**rrange furniture, clean the **B**athroom, **C**ook dinner, do the **D**ishes

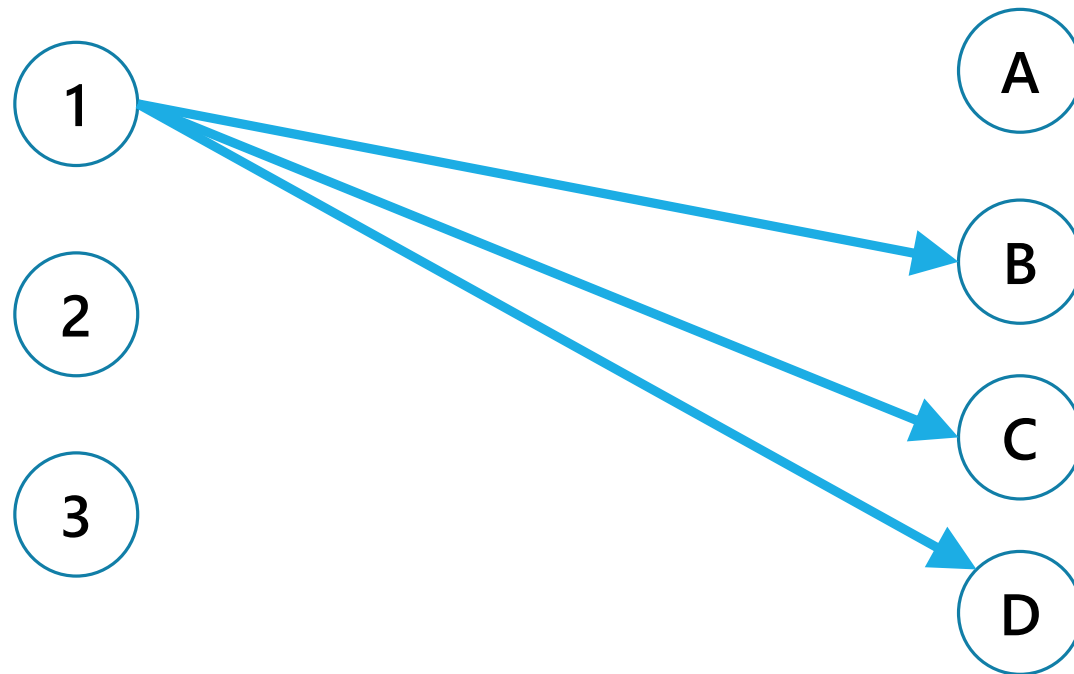
Housemate 1 is unable to arrange furniture, 2 is unable to cook.

# Example Problem

Housemate 1 is unable to arrange furniture, 2 is unable to cook.

Vertex for each housemate and chore.

Edge if the housemate **could** do the chore

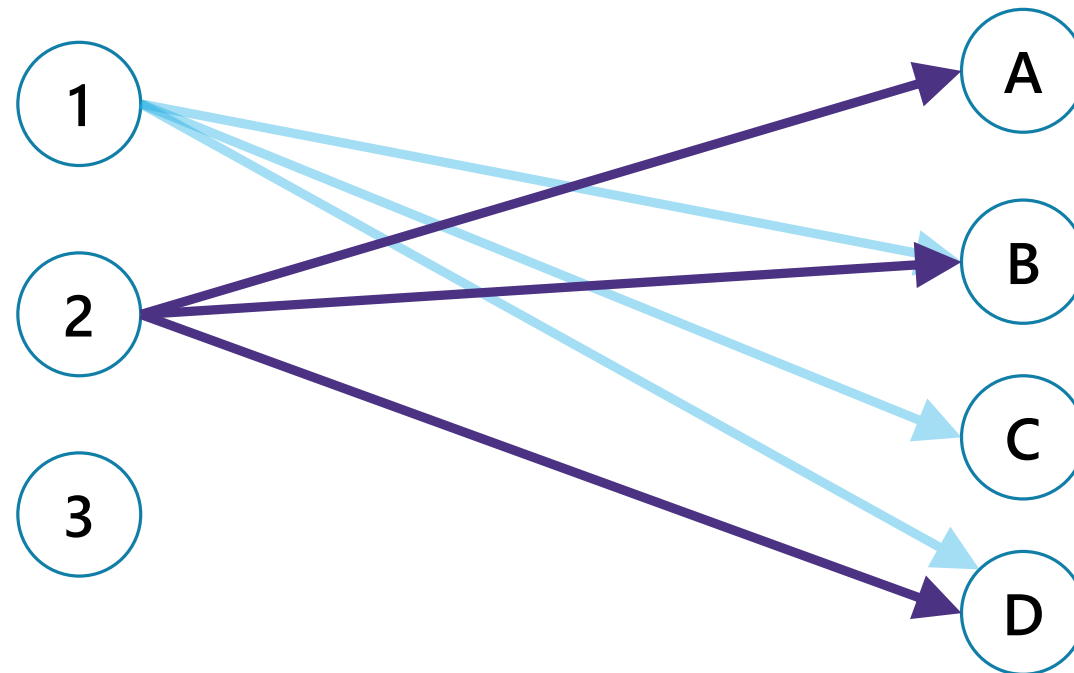


# Example Problem

Housemate 1 is unable to arrange furniture, 2 is unable to cook.

Vertex for each housemate and chore.

Edge if the housemate **could** do the chore

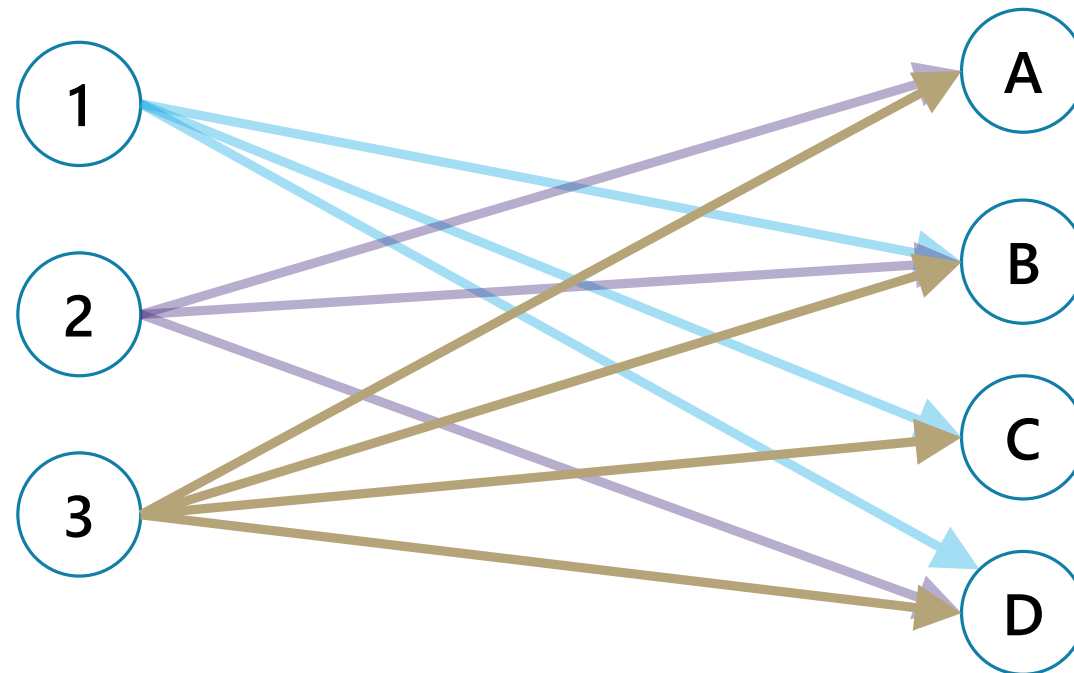


# Example Problem

Housemate 1 is unable to arrange furniture, 2 is unable to cook.

Vertex for each housemate and chore.

Edge if the housemate **could** do the chore

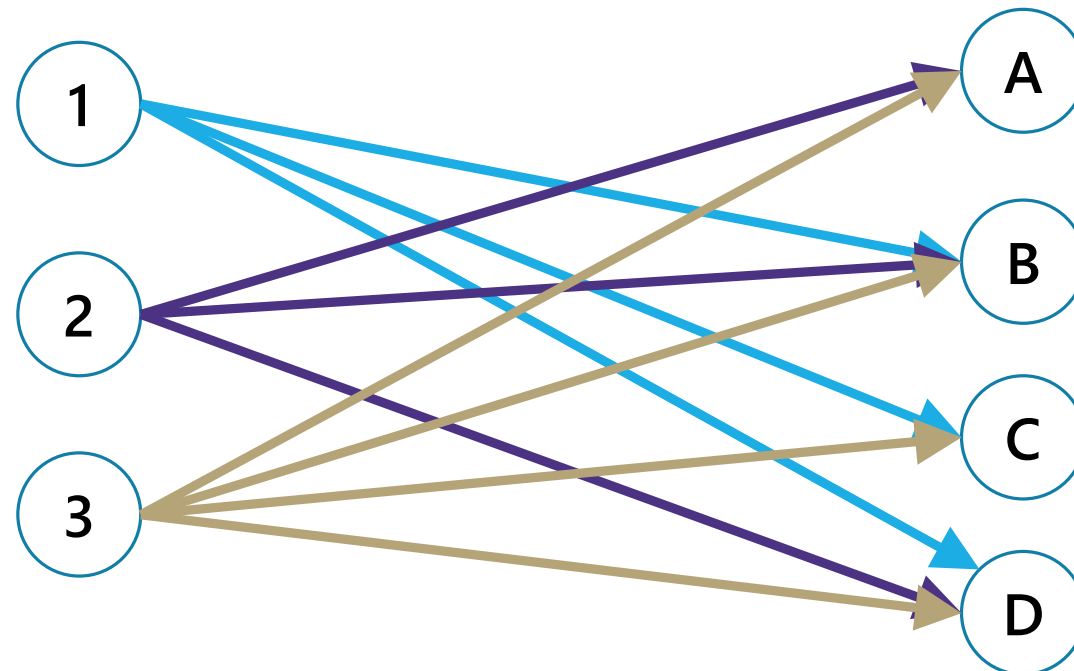


# Example Problem

Housemate 1 is unable to arrange furniture, 2 is unable to cook.

Vertex for each housemate and chore.

Edge if the housemate **could** do the chore

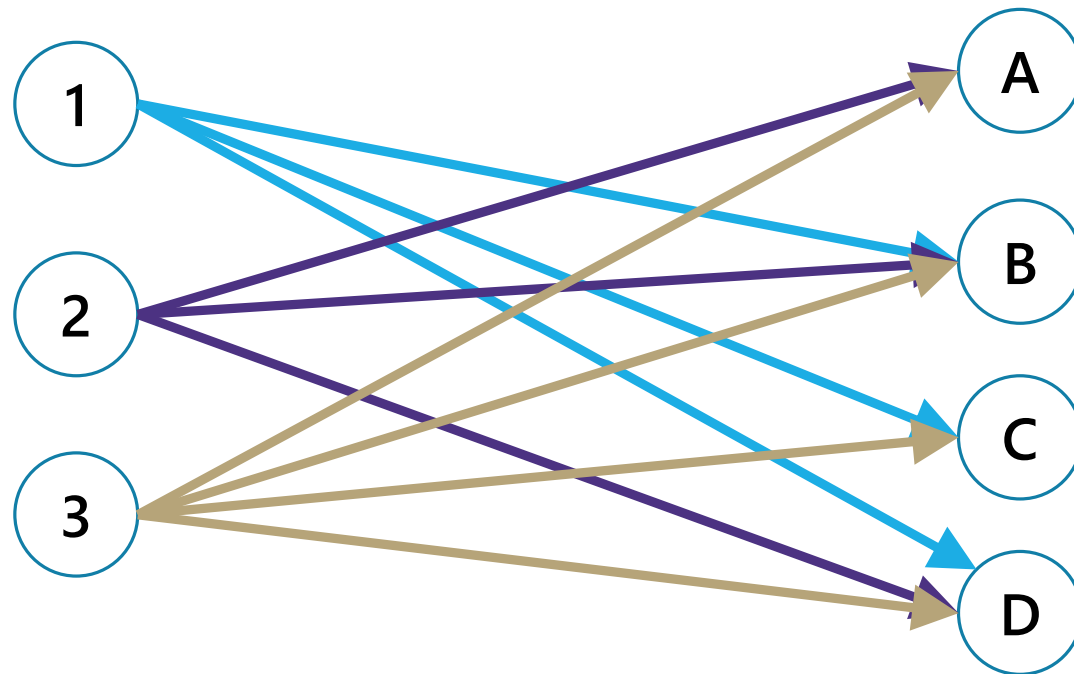


# Example Problem

Idea: Flow from 1 to  $B$  means "make housemate 1 do chore B."

Every chore needs to be done (by one person).

Every person needs to do at most two chores.

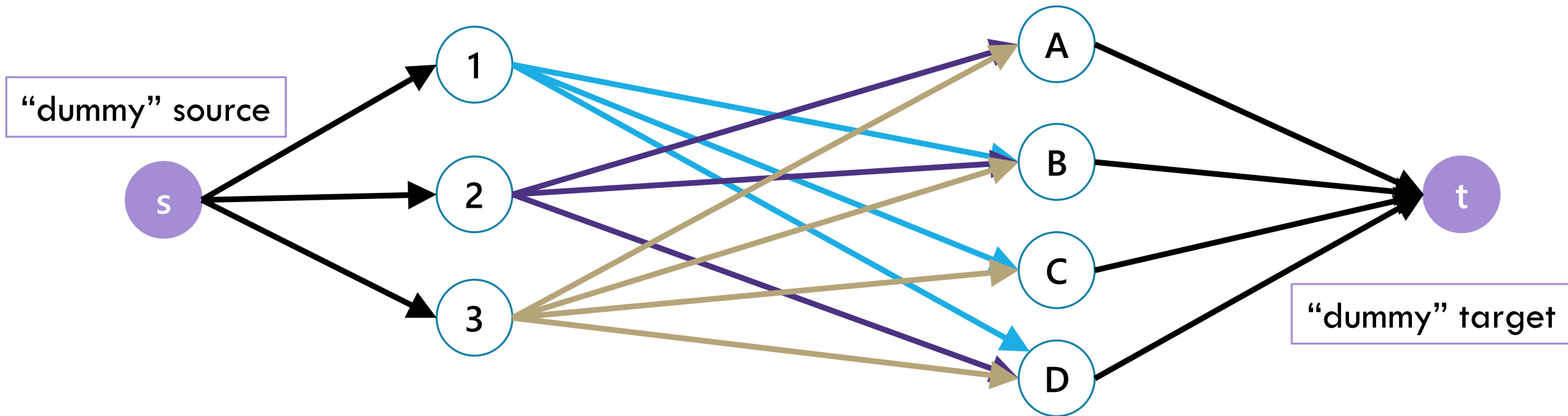


# Example Problem

Idea: Flow from 1 to  $B$  means "make housemate 1 do chore B."

Every chore needs to be done (by one person).

Every person needs to do at most two chores.

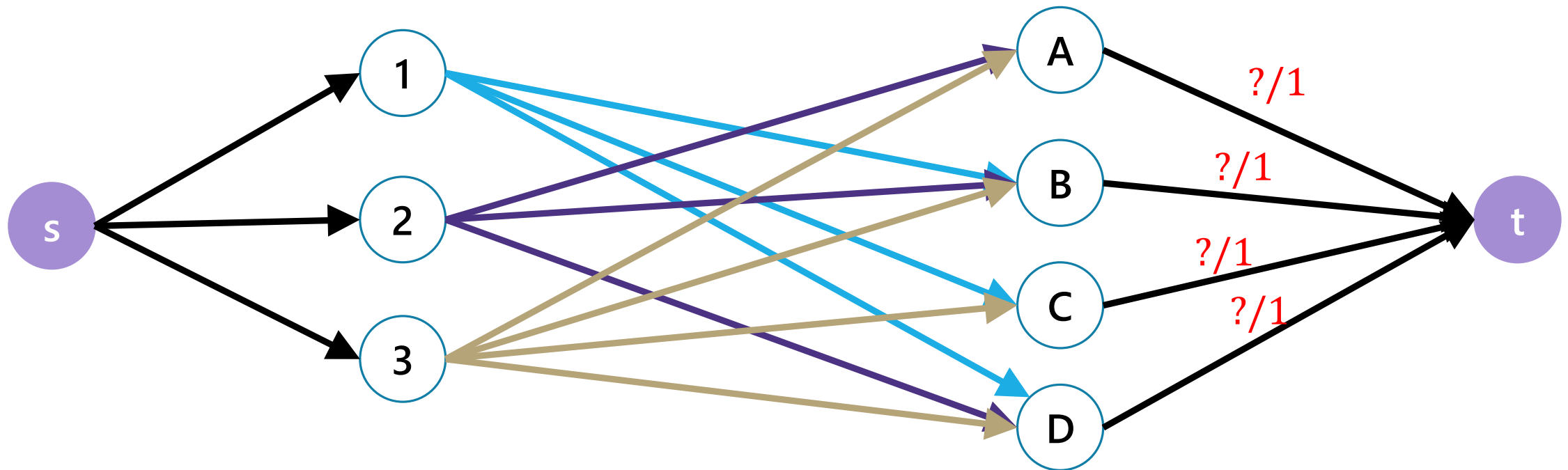


# Example Problem

Idea: Flow from 1 to  $B$  means "make housemate 1 do chore B."

Every chore needs to be done (by one person).

Every person needs to do at most two chores.

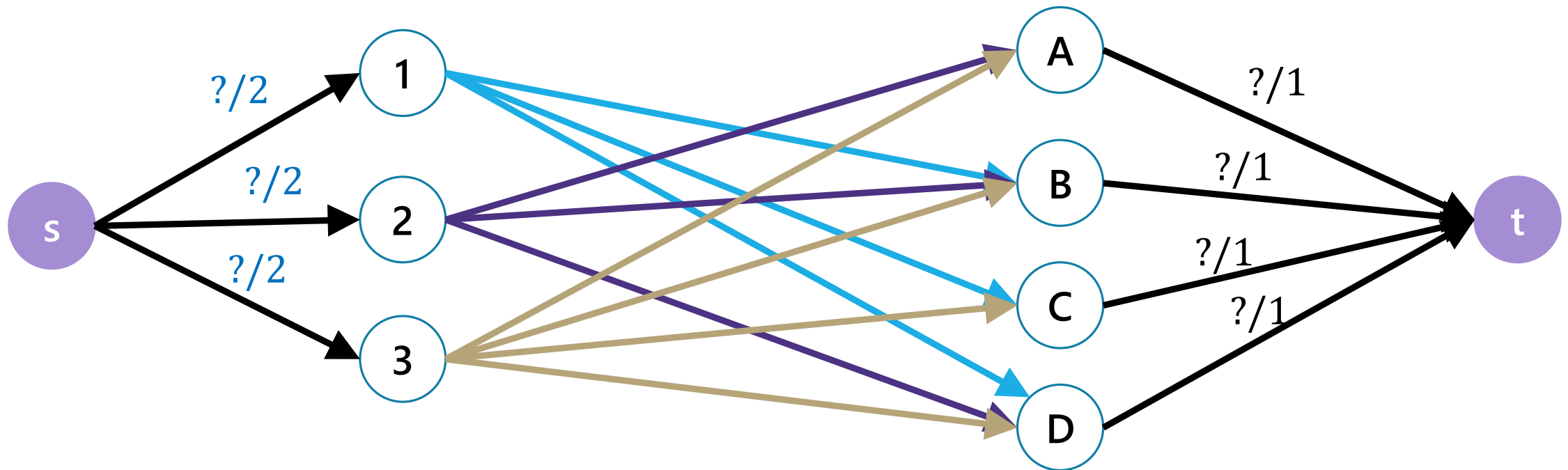


# Example Problem

Idea: Flow from 1 to  $B$  means "make housemate 1 do chore B."

Every chore needs to be done (by one person).

Every person needs to do at most two chores.

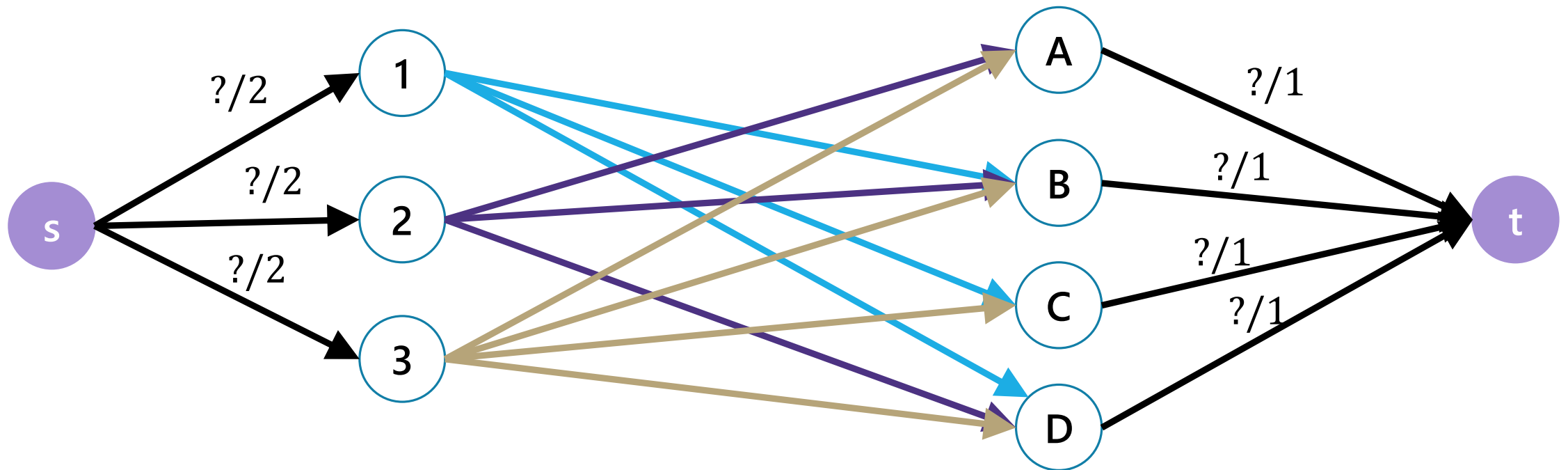


# Example Problem

What are the capacities for the middle edges?

Could make them 1 (make sure you don't get "two units of cooking")

All our requirements are already (implicitly) encoded. So could make them  $\infty$  instead.

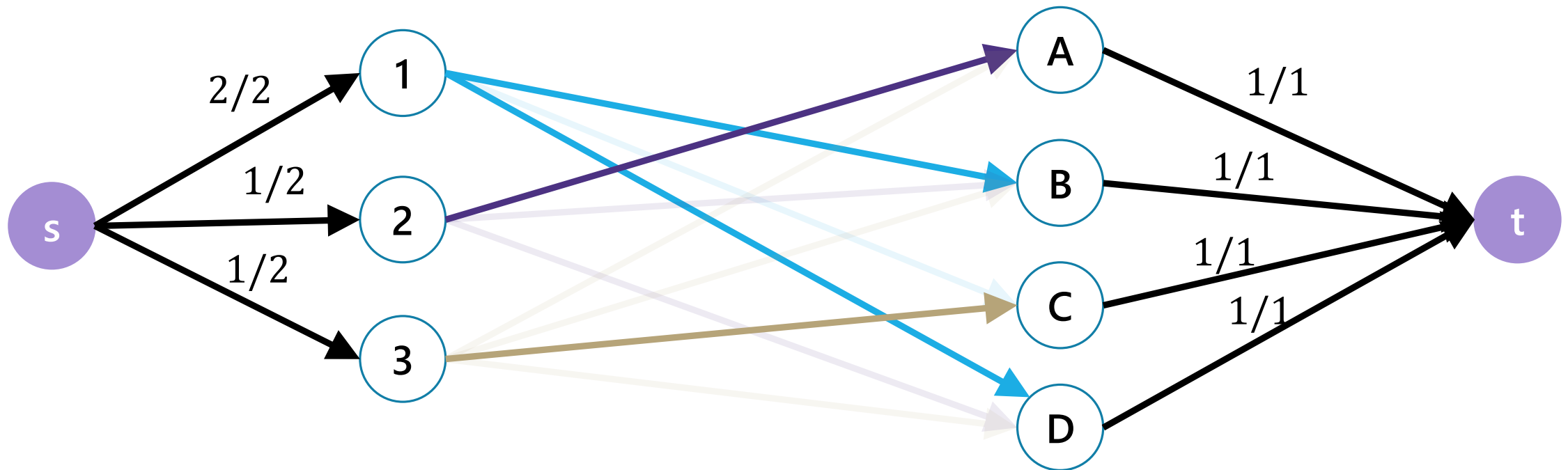


# Example Problem

Find a max flow...And read off the assignment!

Full color: 1 unit of flow, faded: 0 units of flow

1 cleans the bathroom and does the dishes, 2 arranges furniture, 3 cooks.



# Why are all of our constraints met?

Every chore gets done

No one does more than 2 chores

People only do chores they're capable of

# Why are all of our constraints met?

Every chore gets done

A flow of value 4 sends one unit of flow through each of A,B,C,D (because the edges to  $t$  are all capacity 1), so a max-flow ensures if possible we'll find an assignment.

No one does more than 2 chores

Only 2 units of flow can go through any person vertex (because edges from  $s$  to people are all capacity 2).

People only do chores they're capable of

There is only an edge from a person to a chore if they can do that chore.

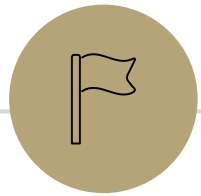
# One More Requirement...

There's another requirement we haven't mentioned:

People only get "whole units" of chores  
i.e. you don't have two people each doing half of the cooking.

The max-flow approach guarantees this! As long as our requirements are integers (or  $\infty$ ) as well.

Same logic as Friday's lecture – Ford-Fulkerson will only add integers to the current flow.



## **Some Practice**



# Write an LP for finding the biggest flow

Let  $c(e)$  be the capacity on edge  $e$

# Write an LP for finding the biggest flow

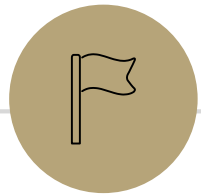
Let  $c(e)$  be the capacity on edge  $e$

Max  $\sum_{e:e \text{ enters } t} x_e$

Subject to:

$\sum_{e:e \text{ enters } u} x_e = \sum_{e:e \text{ leaves } u} x_e$  for every  $u$  other than  $s$  and  $t$

$0 \leq x_e \leq c(e)$  for every edge  $e$



# Speeding Up Ford-Fulkerson

# Speeding Up Ford-Fulkerson

Ford-Fulkerson is only slow if  $f$  is big and we keep doing very small updates.

If instead of finding **any**  $s, t$  path you find a particular one (the one with the fewest edges, or the one that will lead to the biggest increase) you can guarantee you'll do fewer iterations.

Each iteration will take a little longer to find the good path, but the tradeoff is often worth it.

In particular some algorithms end up with running times independent of  $f$ .

In practice, Ford-Fulkerson is usually fine!

The fastest-known (theoretically) is Orlin's algorithm: running time  $O(VE)$ .