

Lecture 17

max flows, min cuts, and Ford-Fulkerson

Announcements:

COVID-19-related Course Changes

- Everything (including this lecture) will be remote from now on.
 - Office Hours, Review Session, Lectures, Final
- Stay tuned to Piazza for logistical details.
- If you need to leave campus, feel free.
 - (However, maybe self-quarantine when you get to where you are going...)

Lectures in Week 10

- I will record videos ahead of time
 - Like the one you are watching now!
- I will be available during our normal class time (M/W 10:30-11:50) on zoom to answer any questions that you have.
 - See Piazza for details!

Final Exam Info

- The exam will still be 3:30-6:30pm on Monday 3/16.
- It will be **remote, timed, and open-book.**
 - Current plan:
 - You will get access to the exam at 3:30pm.
 - By 6:30pm, you must upload it to Gradescope, like you would a normal HW assignment.
 - You do not need to print out the exam, you may write on blank paper or type while you look at the exam on your computer.
 - If you are taking the exam at an alternate time, you will do this procedure at that alternate time.
- **SEE (impending) PIAZZA POST FOR UPDATES and DETAILS!**

What to expect on the final exam

- Similar format to the midterm and practice exams.
 - A bit different in flavor because of the last-minute change to an open-book exam, but not that different.
- Covers Lectures 1-17.
 - Light coverage of Lecture 17 (today) since we don't have any HW on it.
- The exam **will be difficult**.
 - But don't worry! This means it will be difficult for everyone.
 - You don't have to get all of it correct to do well!
- While it will be difficult, we are being mindful about the length.
 - In particular, we are aware that you might want 15 minutes or so to scan and upload.

How to study for the final exam

- Do practice problems!
 - Homework, Section, Algorithms Illuminated, CLRS, practice exams.
- (Remotely) attend review session!
 - That's "section" this week.
- (Remotely) attend the "Review Party!"
 - That's the "HW party" this week.
 - CAs will be available remotely (probably on Zoom).
- (Remotely) attend office hours, and ask questions on Piazza!
 - We're here to help!

End of announcements!

The plan for today

- Minimum s-t cuts
- Maximum s-t flows
- The Ford-Fulkerson Algorithm
 - Finds min cuts and max flows!
- Applications
 - Why do we want to find these things?

This lecture will skip a few proofs, and you are **not** responsible for the proofs for the final exam.

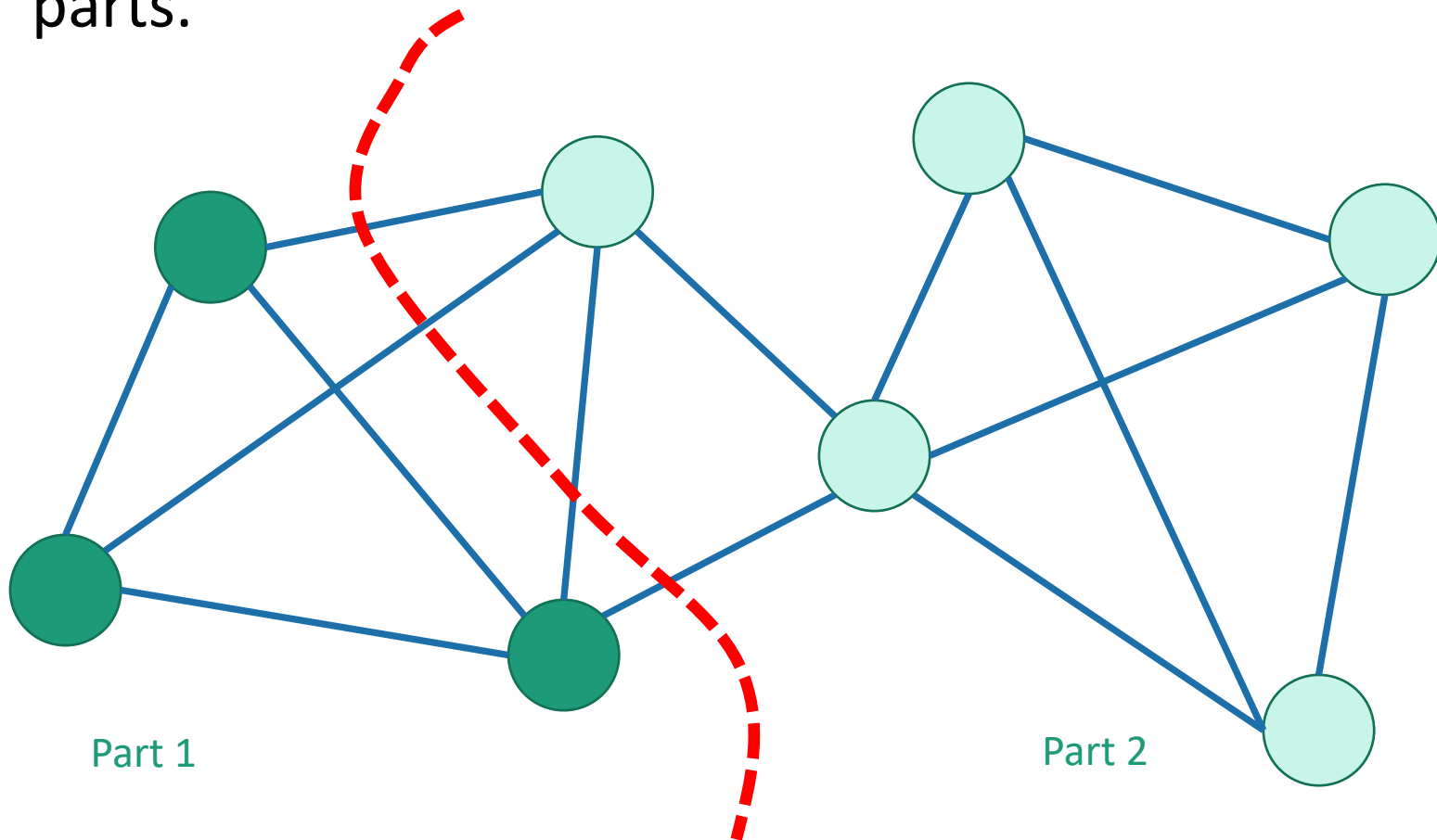


Lucky the lackadaisical lemur

Last time

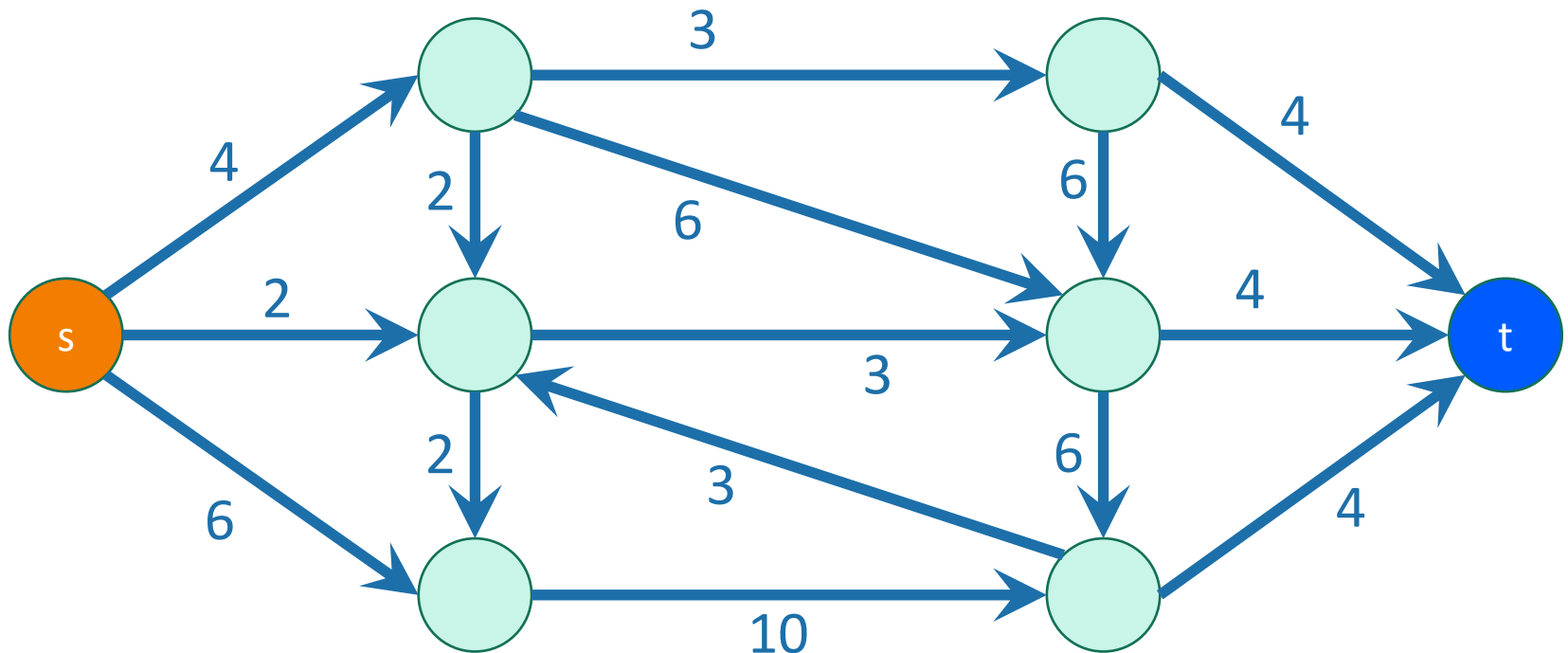
Last time graphs were undirected and unweighted.

- We talked about **global min-cuts**
- A cut is a partition of the vertices into two nonempty parts.



Today

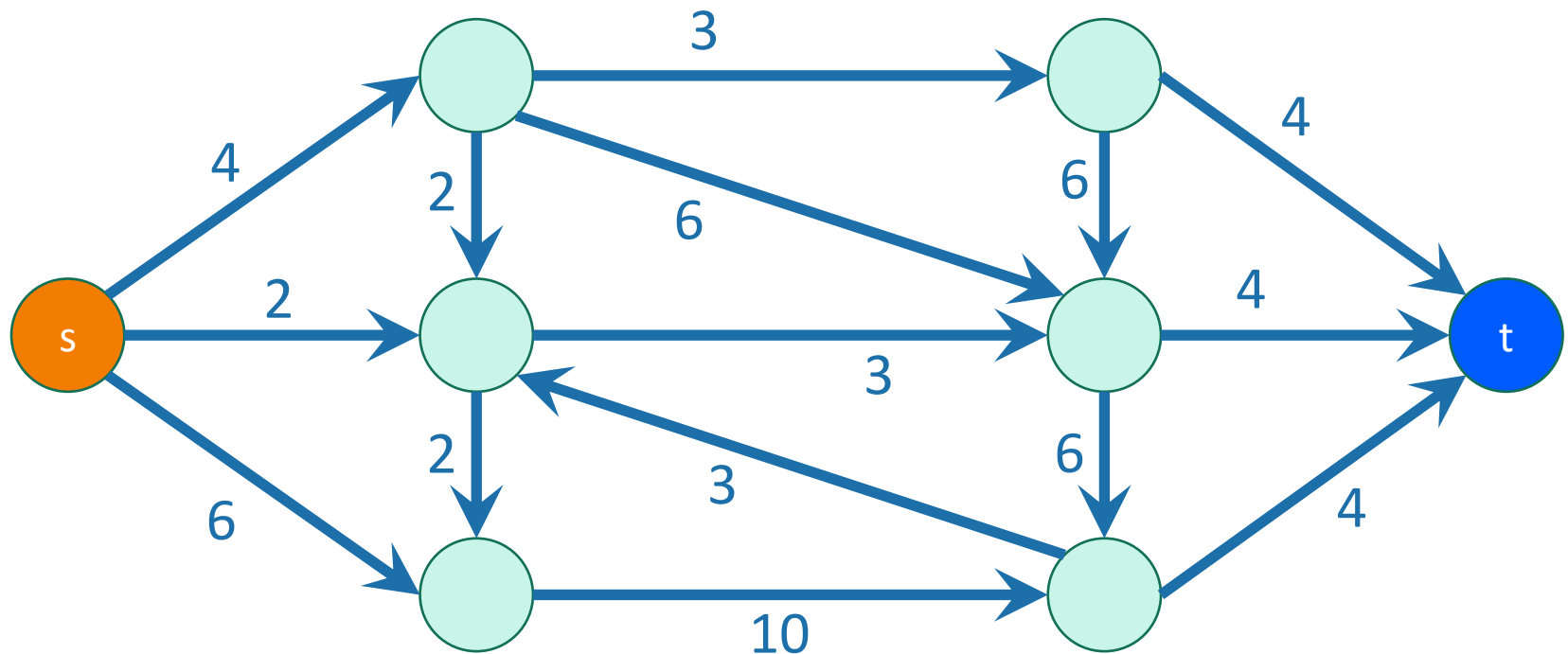
- Graphs are directed and edges have “capacities” (weights)
- We have a special “source” vertex s and “sink” vertex t .
 - s has only outgoing edges*
 - t has only incoming edges*



*at least for this class

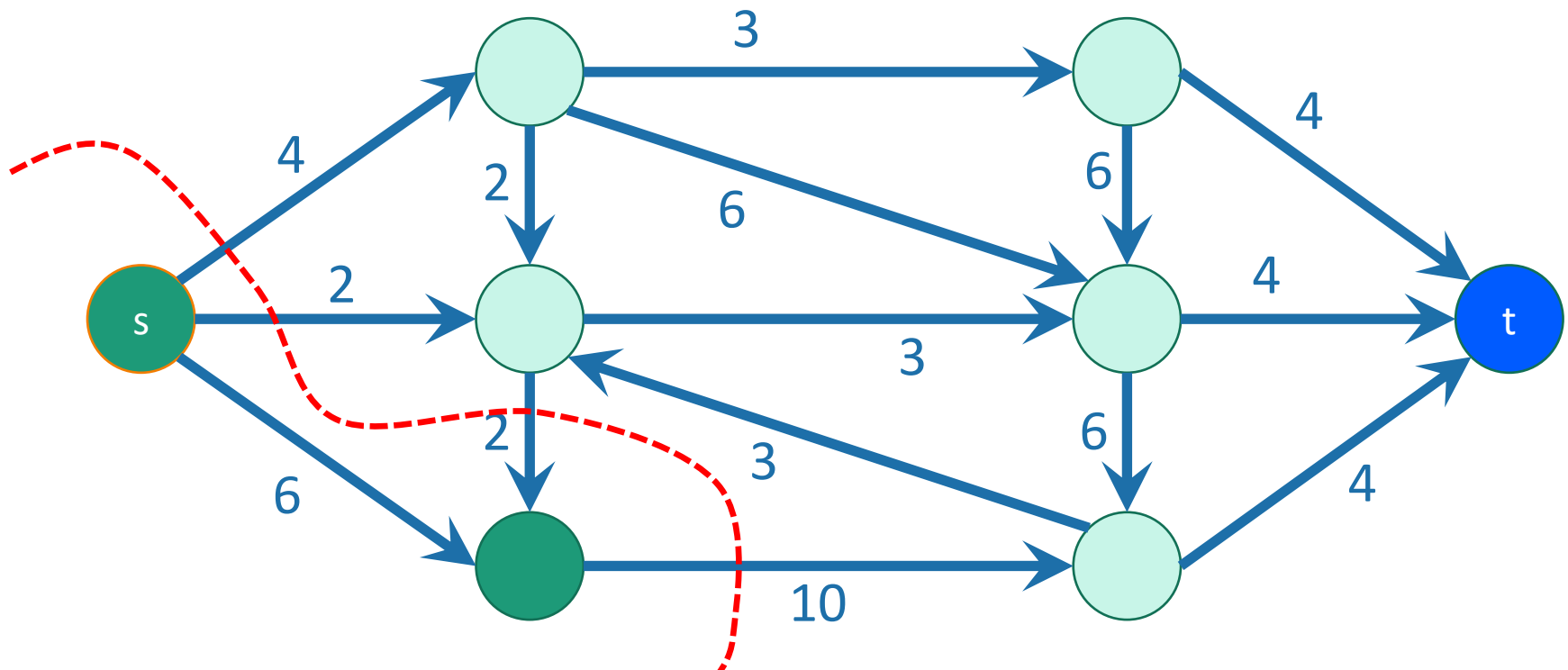
An s-t cut

is a cut which separates s from t



An s-t cut

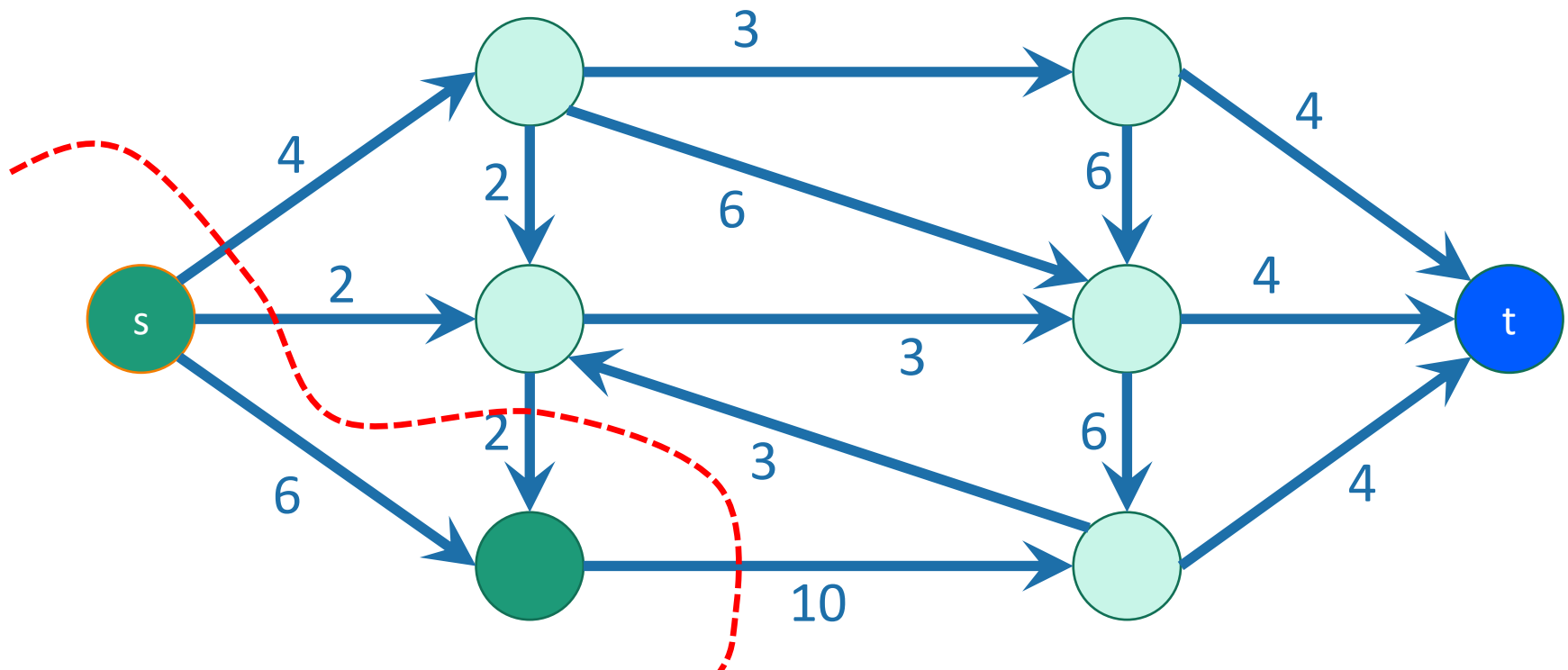
is a cut which separates s from t



An s-t cut

is a cut which separates s from t

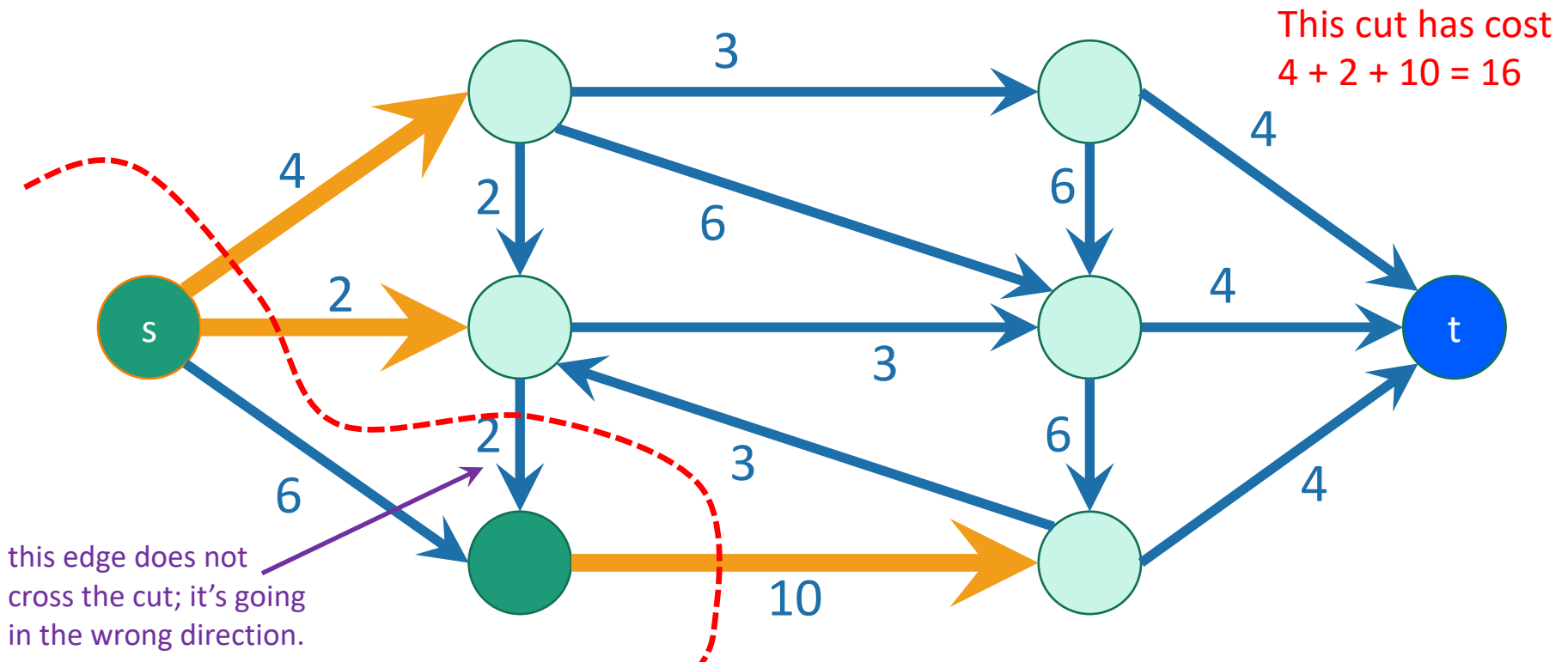
- An edge **crosses the cut** if it goes from s's side to t's side.



An s-t cut

is a cut which separates s from t

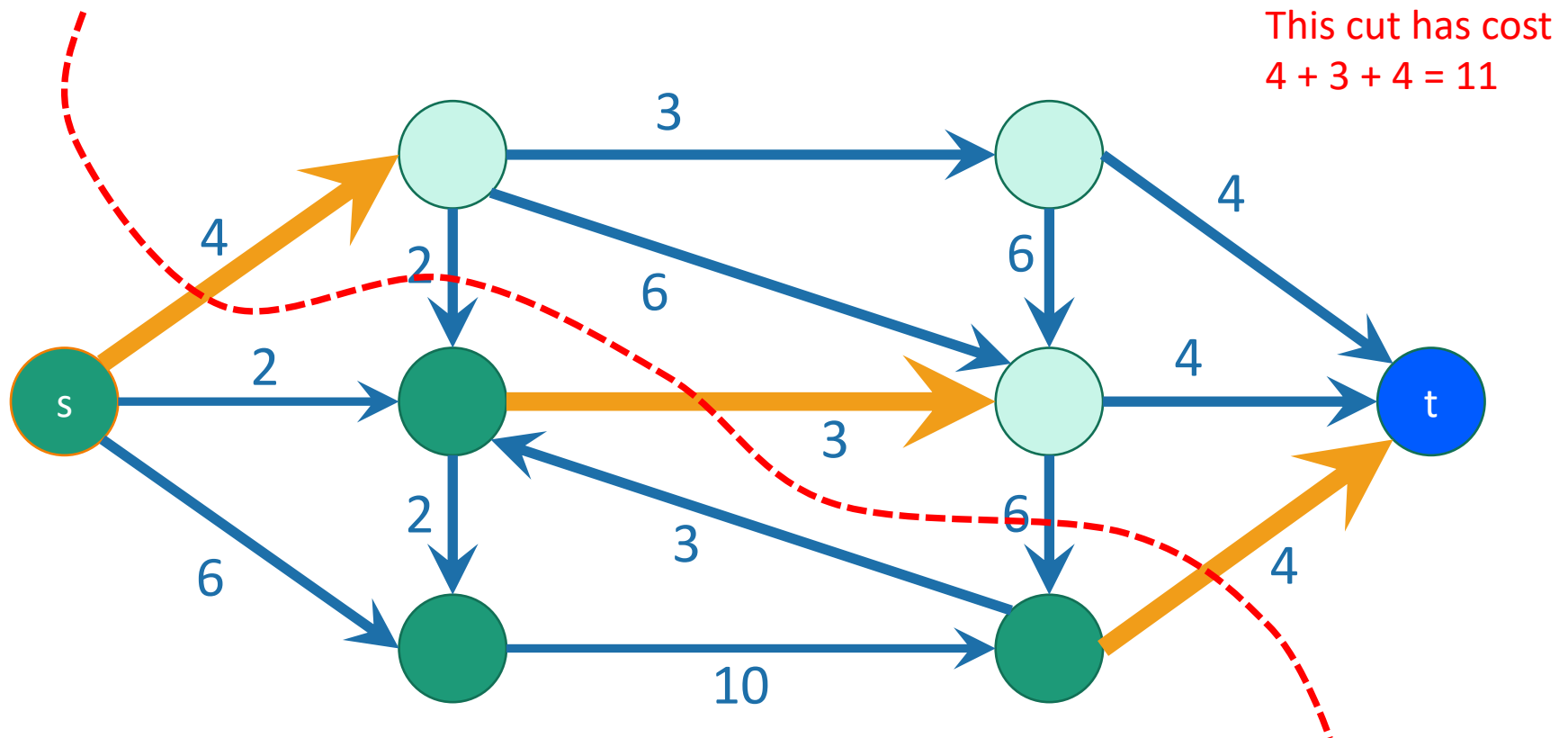
- An edge **crosses the cut** if it goes from s's side to t's side.
- The **cost** (or capacity) of a cut is the sum of the capacities of the edges that cross the cut.



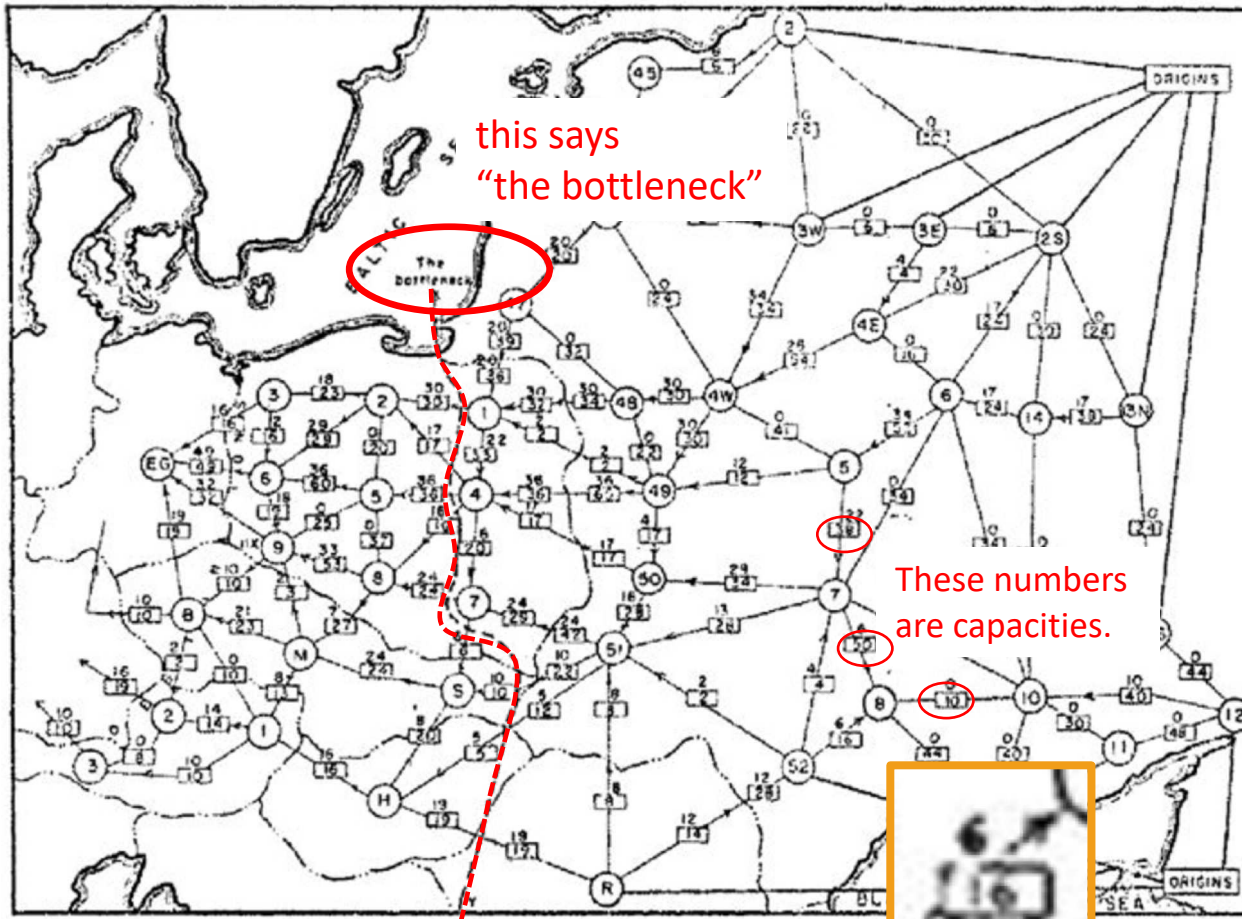
A minimum s-t cut

is a cut which separates s from t with minimum cost.

- Question: how do we find a minimum s-t cut?



Example where this comes up

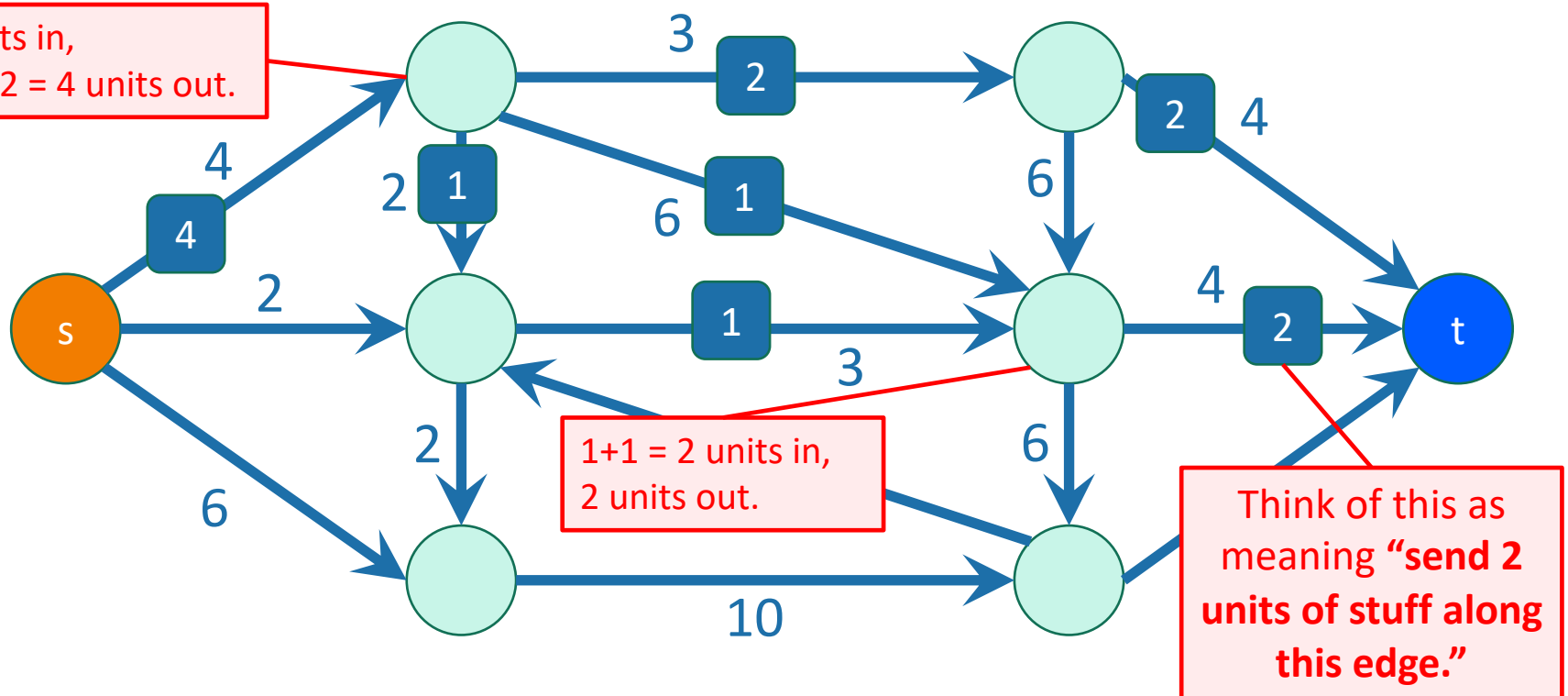


Schriver 2002

- 1955 map of rail networks from the Soviet Union to Eastern Europe.
 - Declassified in 1999.
 - 44 edges, 105 vertices
- The US wanted to cut off routes from **suppliers in Russia** to **Eastern Europe** as efficiently as possible.
- In 1955, Ford and Fulkerson gave an algorithm which finds the optimal s-t cut.

Flows

- In addition to a capacity, each edge has a **flow**
 - (unmarked edges in the picture have flow 0)
- The flow on an edge must be less than its capacity.
- At each vertex, the incoming flows must equal the outgoing flows.

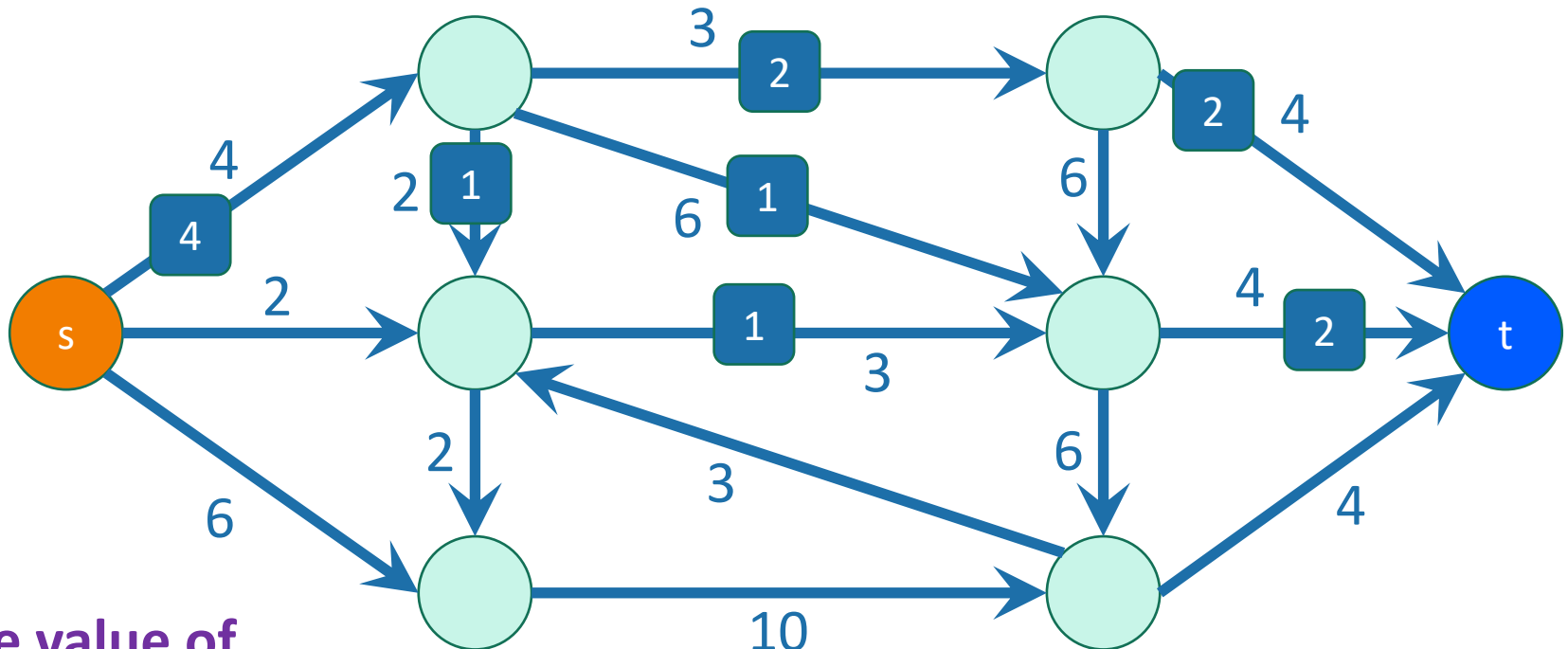


Flows

- The value of a flow is:
 - The amount of stuff coming out of s
 - The amount of stuff flowing into t
 - These are the same!

Because of conservation of flows at vertices,

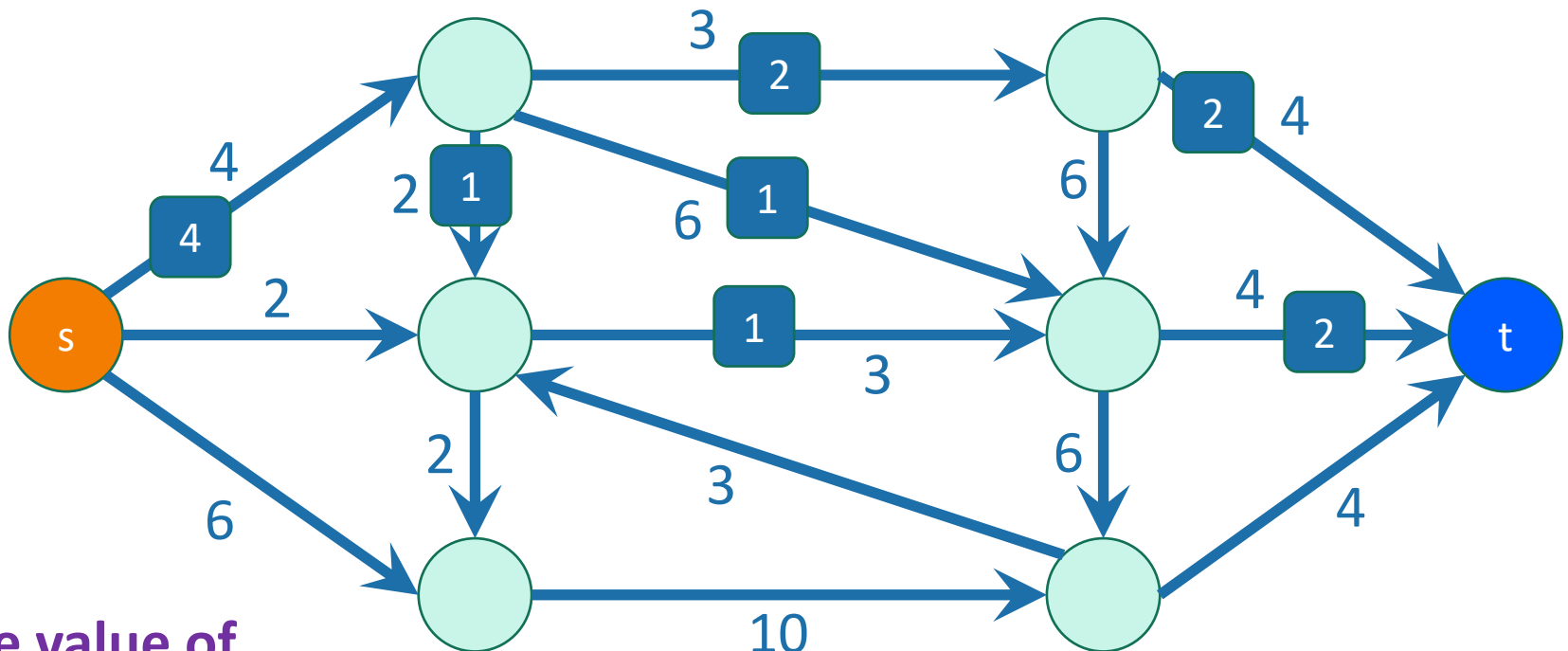
stuff you put in
=
stuff you take out.



The value of this flow is 4.

A maximum flow
is a flow of maximum value.

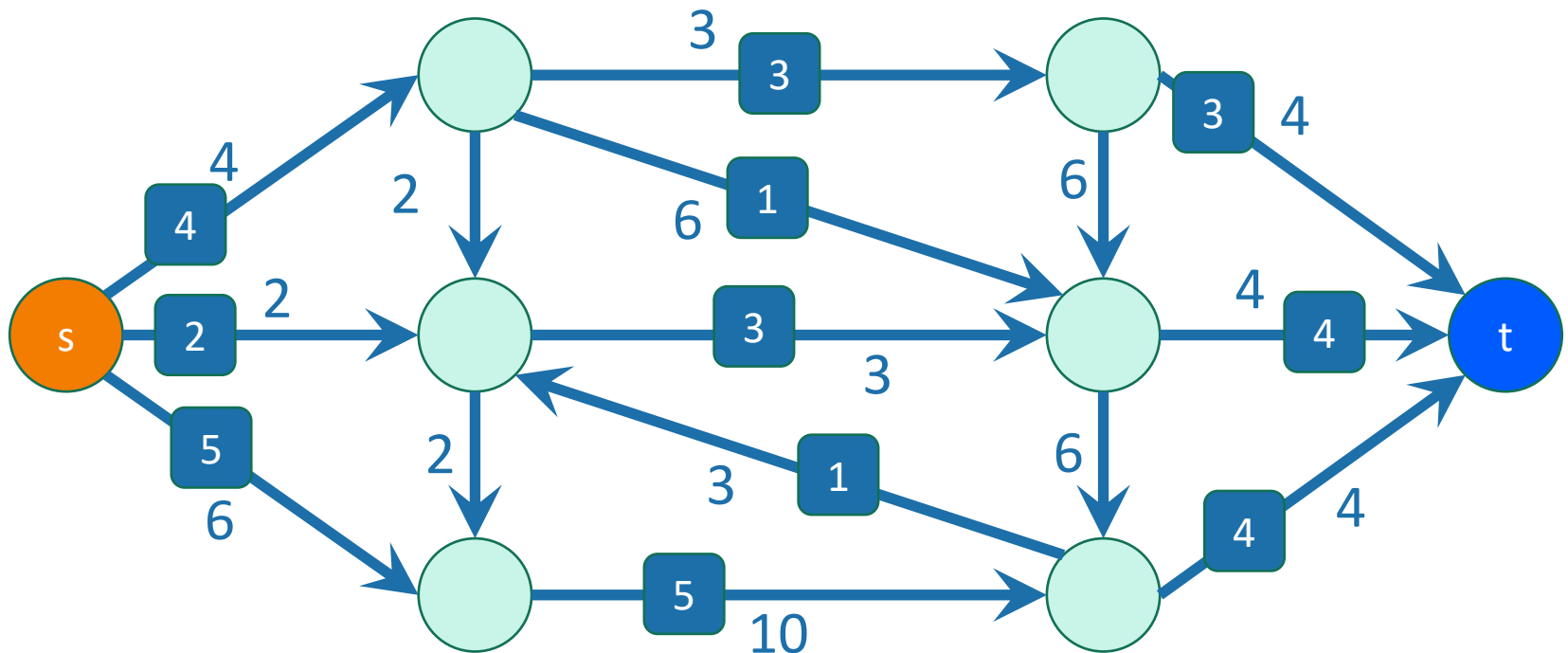
- This example flow is pretty wasteful, I'm not utilizing the capacities very well.



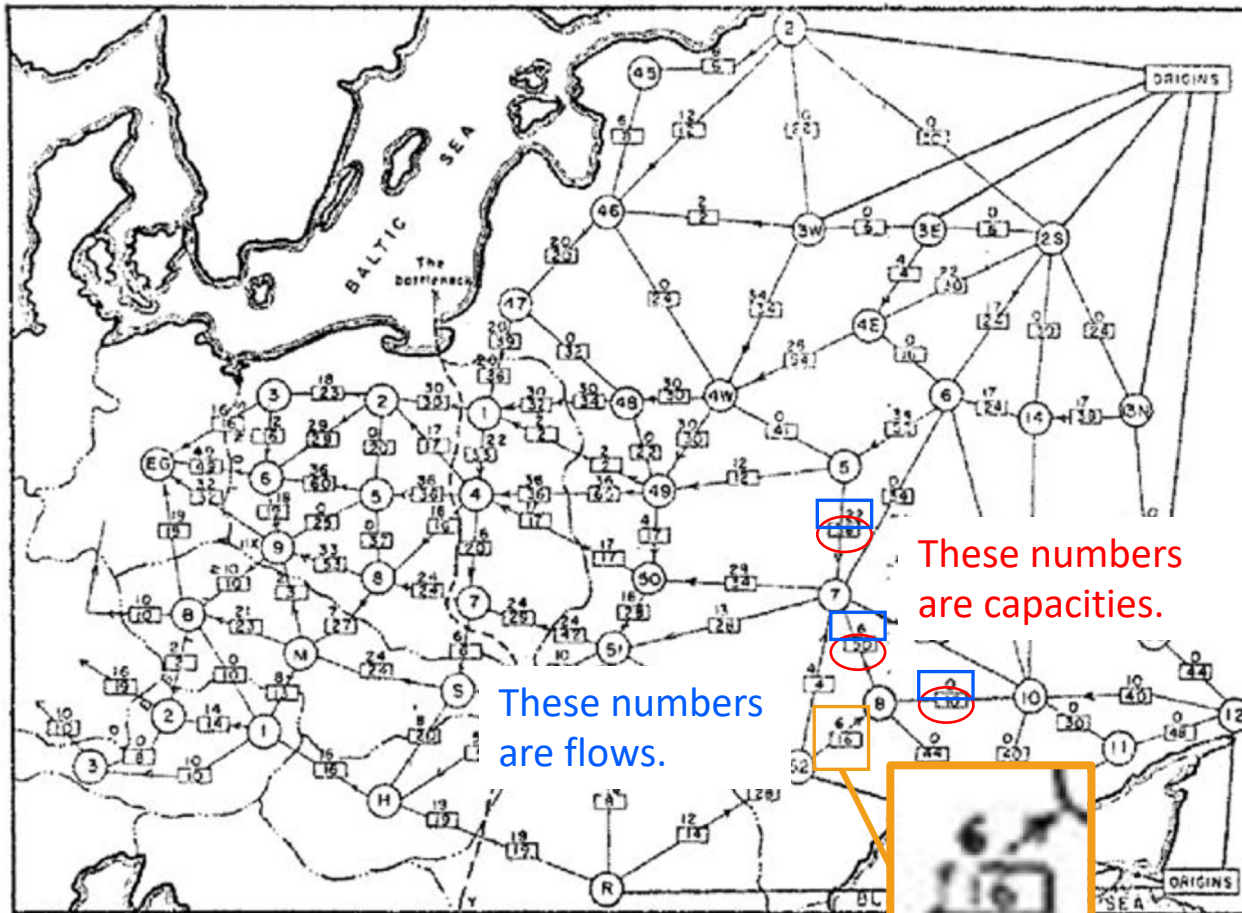
The value of
this flow is 4.

A maximum flow
is a flow of maximum value.

- This one is maximum; it has value 11.



Example where this comes up



These numbers are flows.

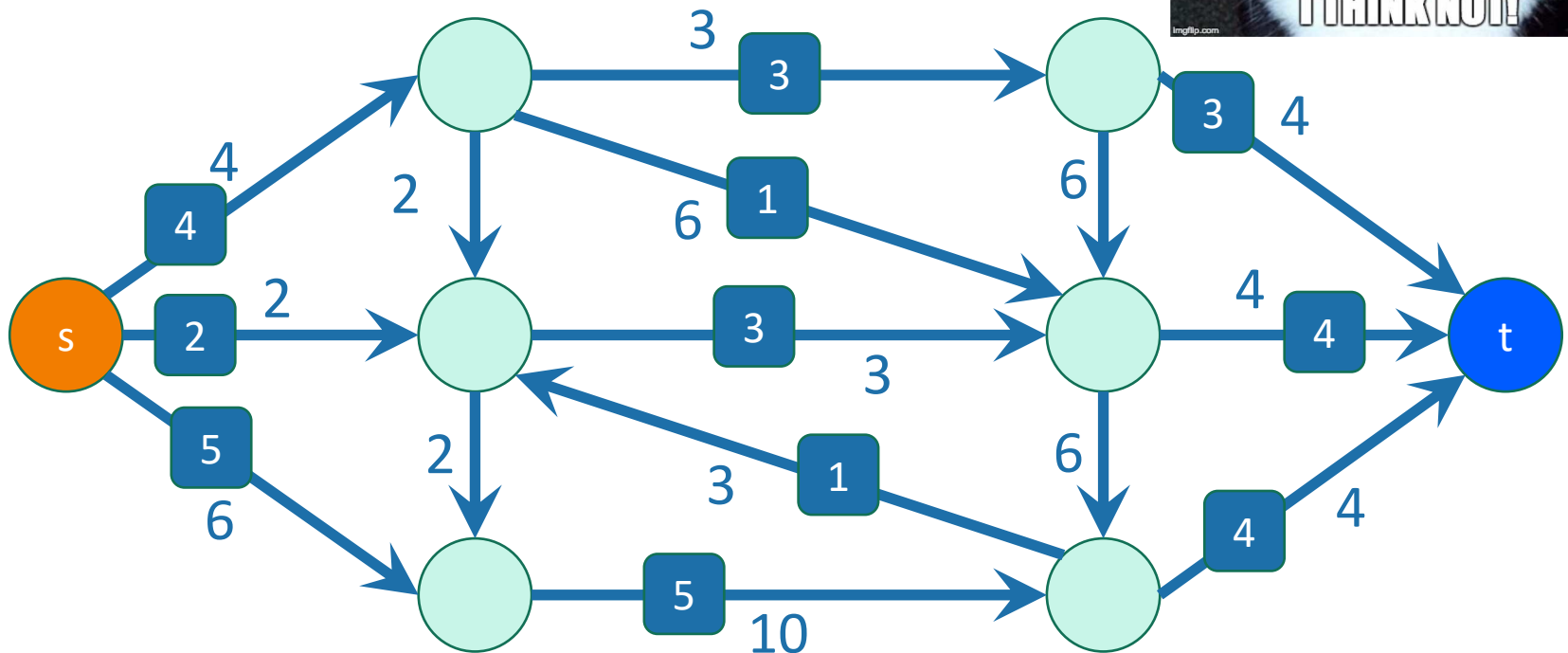
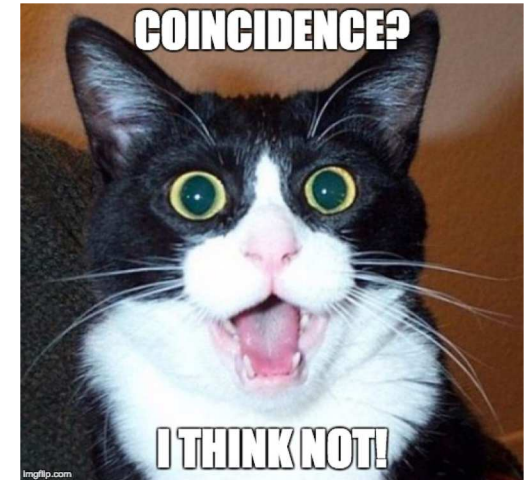
These numbers are capacities.

- 1955 map of rail networks from the Soviet Union to Eastern Europe.
 - Declassified in 1999.
 - 44 edges, 105 vertices
- The Soviet Union wants to route supplies from suppliers in Russia to Eastern Europe as efficiently as possible.

A maximum flow is a flow of maximum value.

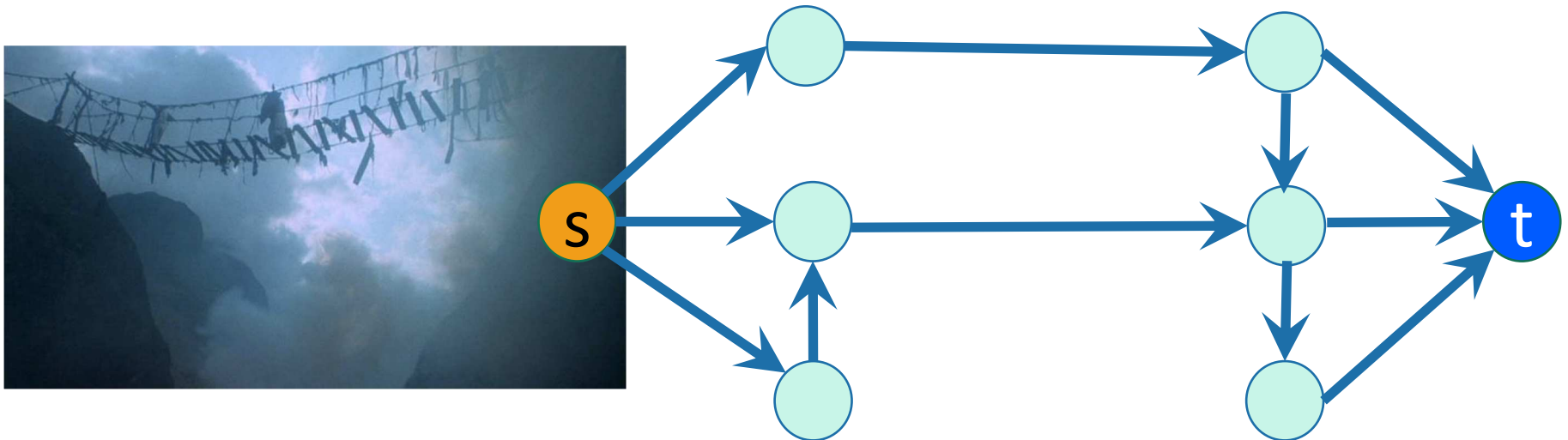
- This one is maximum; it has value 11.

That's the same as the minimum cut in this graph!



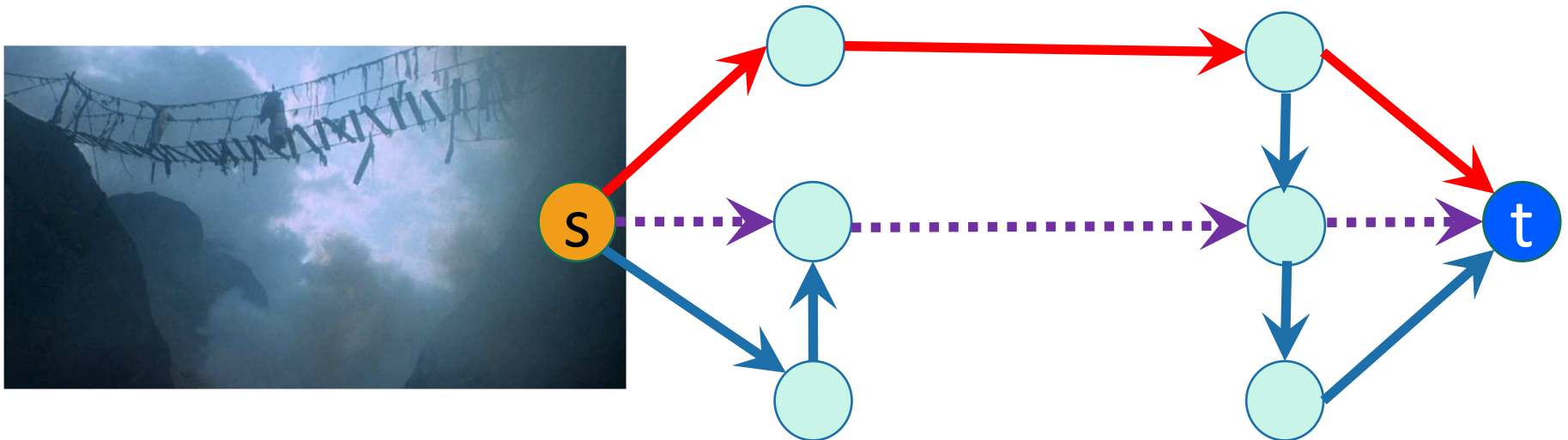
Pre-lecture exercise

- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t ? *For this graph, 2*
- If only one person can be on a bridge at a time, and you want to keep traffic moving (aka, no waiting at vertices allowed), how many people can get to t at a time? *Also 2!*



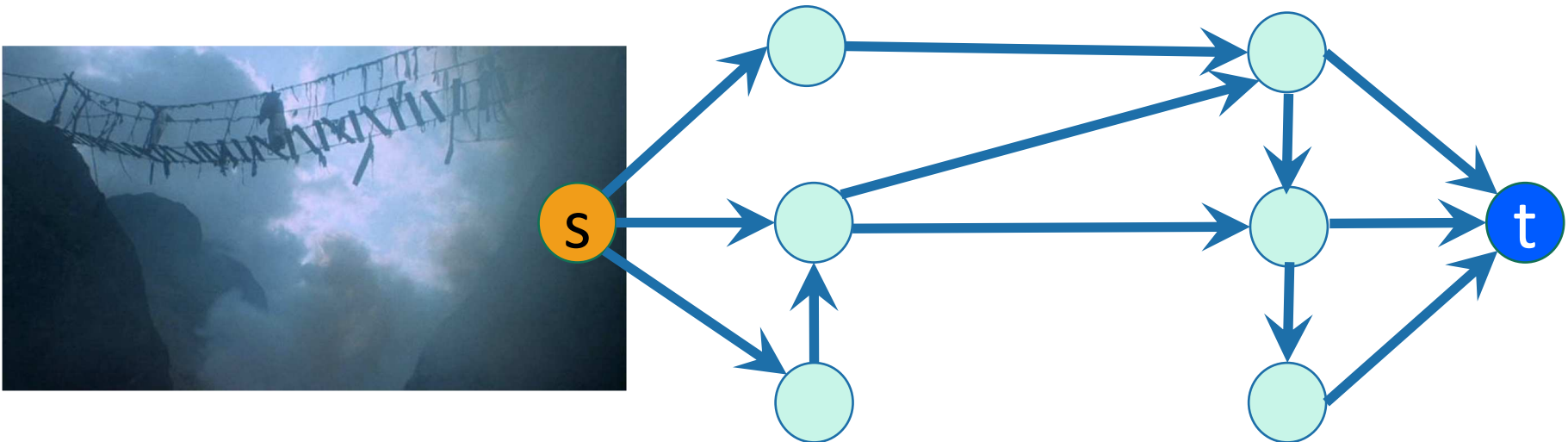
Pre-lecture exercise

- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t ? *For this graph, 2*
- If only one person can be on a bridge at a time, and you want to keep traffic moving (aka, no waiting at vertices allowed), how many people can get to t at a time? *Also 2!*



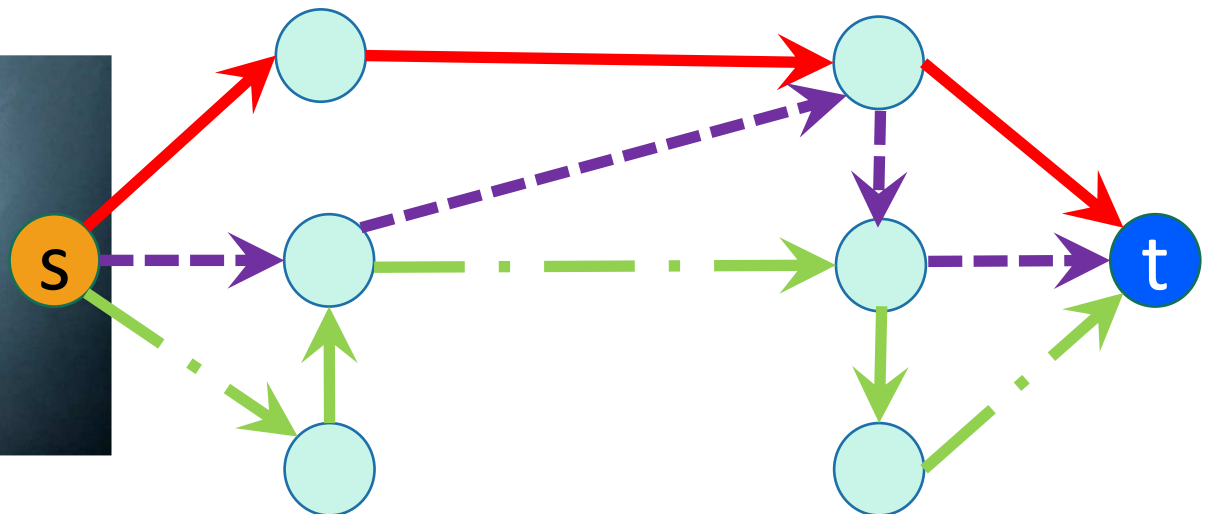
How about now?

- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t ? *For this graph, 3*
- If only one person can be on a bridge at a time, and you want to keep traffic moving (aka, no waiting at vertices allowed), how many people can get to t at a time? *Also 3!*



How about now?

- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t ? *For this graph, 3*
- If only one person can be on a bridge at a time, and you want to keep traffic moving (aka, no waiting at vertices allowed), how many people can get to t at a time? *Also 3!*



Pre-lecture exercise

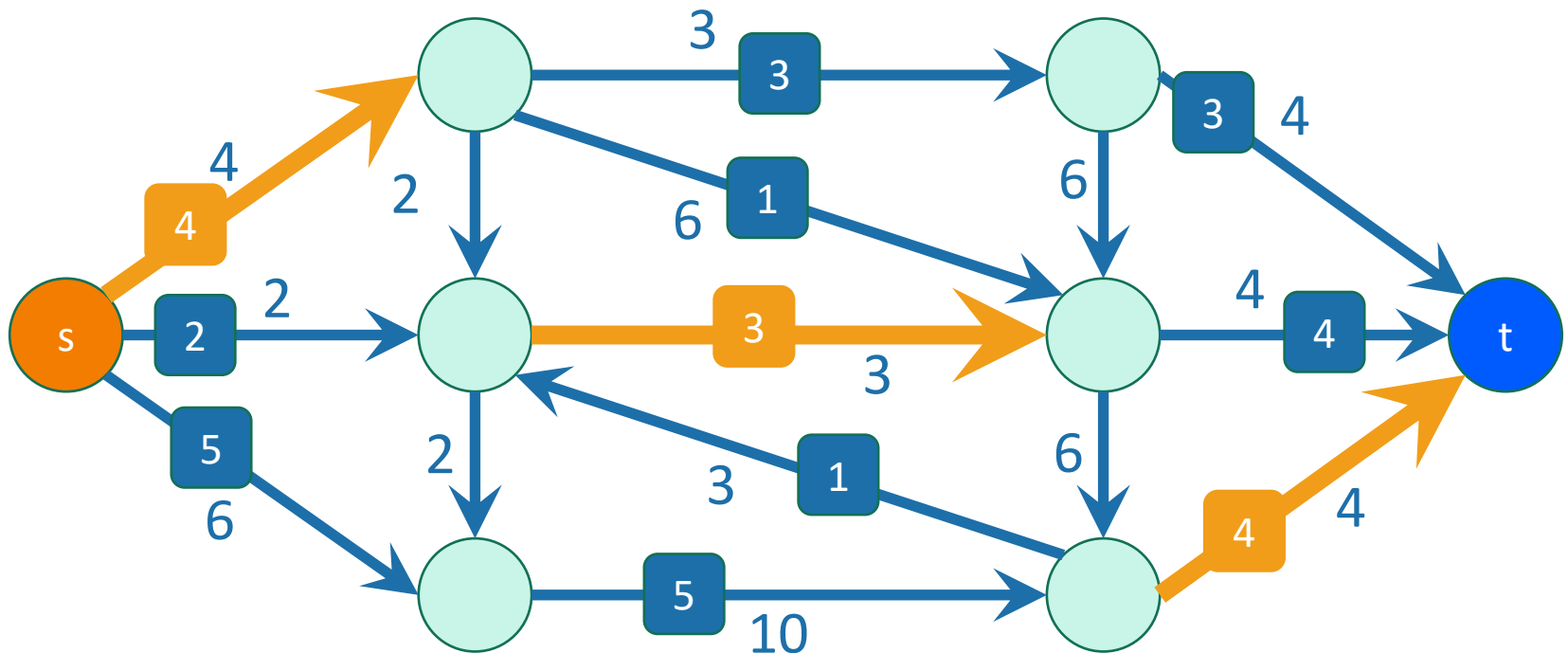
- Can you come up with a graph where the two numbers are different?

Theorem

Max-flow min-cut theorem

The value of a max flow from s to t is equal to the cost of a min s - t cut.

Intuition: in a max flow, the min cut better fill up, and this is the bottleneck.



Proof outline

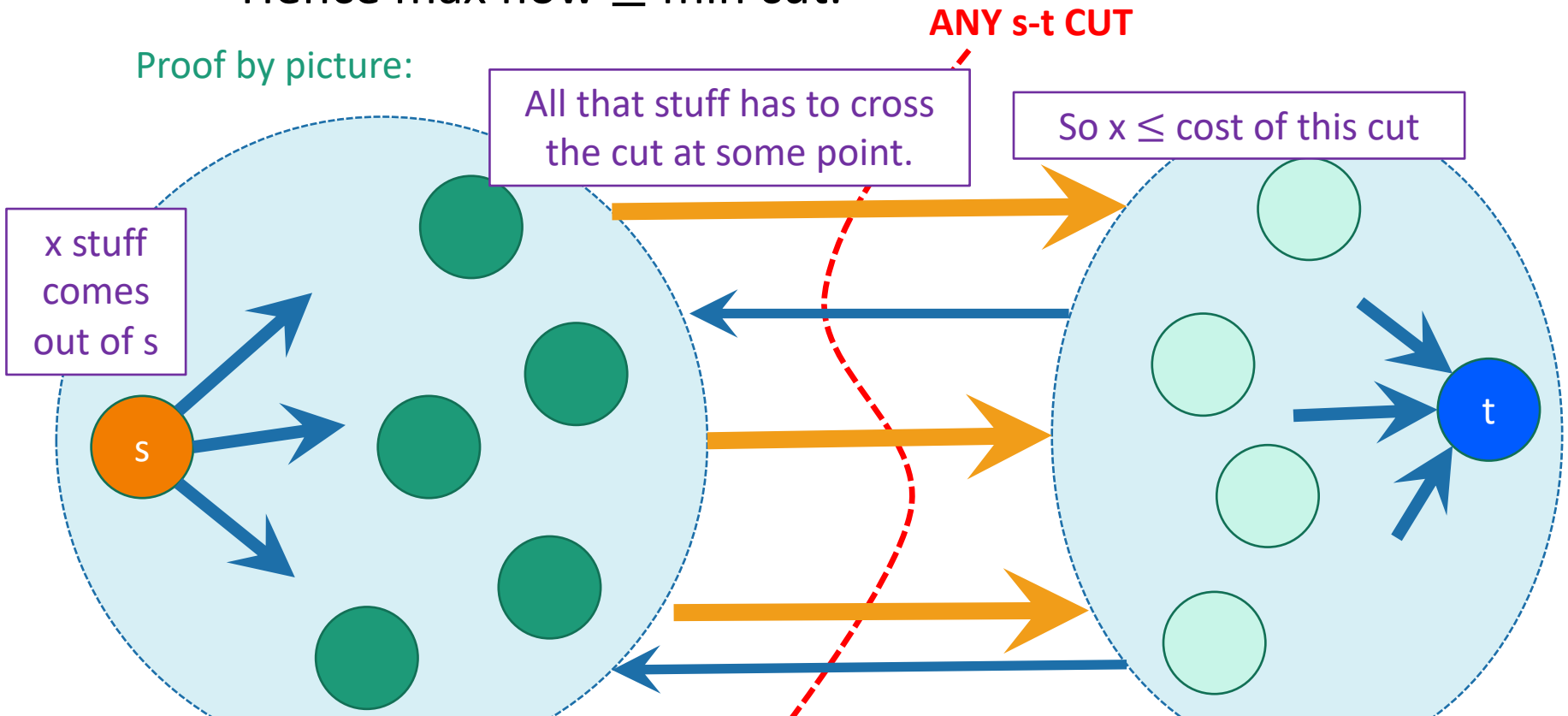
- Lemma 1: $\text{max flow} \leq \text{min cut}$.
 - Proof-by-picture
- What we actually want: $\text{max flow} = \text{min cut}$.
 - Proof-by-algorithm...the Ford-Fulkerson algorithm!
 - (Also using Lemma 1)

One half of Min-Cut Max-Flow Thm

- **Lemma 1:**

- For ANY s-t flow and ANY s-t cut, the value of the flow is at most the cost of the cut.
- Hence $\text{max flow} \leq \text{min cut}$.

Proof by picture:

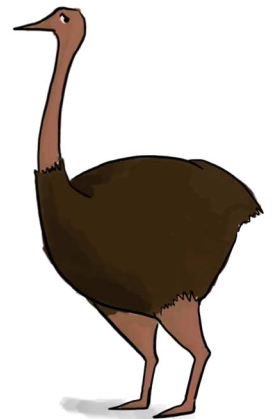


One half of Min-Cut Max-Flow Thm

- **Lemma 1:**

- For ANY s-t flow and ANY s-t cut, the value of the flow is at most the cost of the cut.
- Hence $\text{max flow} \leq \text{min cut}$.

- That was proof-by-picture.
- Good exercise to come up with a proof-by-proof!
 - You are **not** responsible for proof-by-proof for this class.



Min-Cut Max-Flow Thm

- **Lemma 1:**

- For ANY s-t flow and ANY s-t cut, the value of the flow is at most the cost of the cut.
- Hence $\text{max flow} \leq \text{min cut}$.

- The theorem is stronger:

- $\text{max flow} = \text{min cut}$
- This will be proof-by-algorithm!

Ford-Fulkerson algorithm

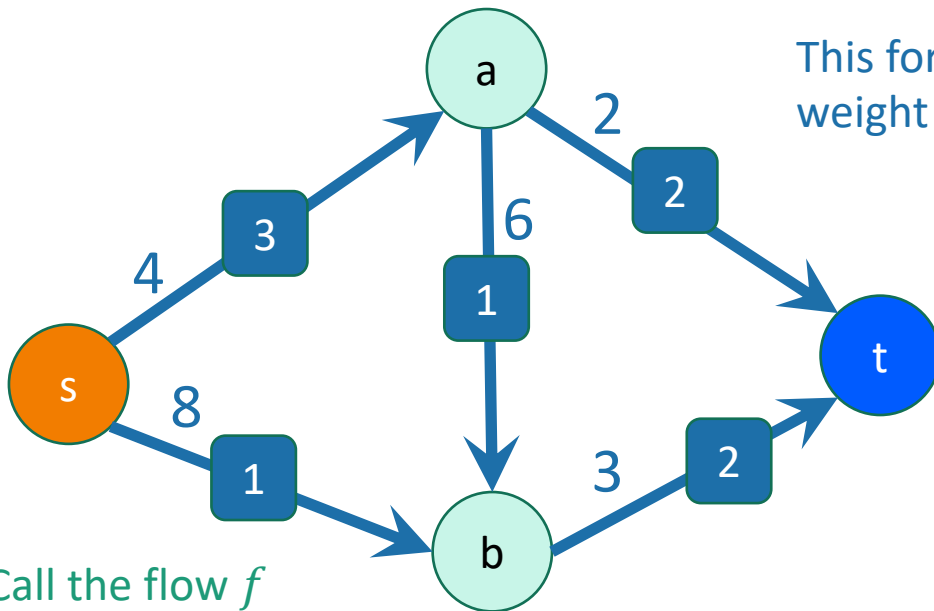
- Outline of algorithm:
 - Start with zero flow
 - We will maintain a “**residual graph**” G_f
 - A path from s to t in G_f will give us a way to improve our flow.
 - We will continue until there are no s - t paths left.

Assume for today that we don't have edges like this, although it's not necessary.



Tool: Residual networks

Say we have a flow

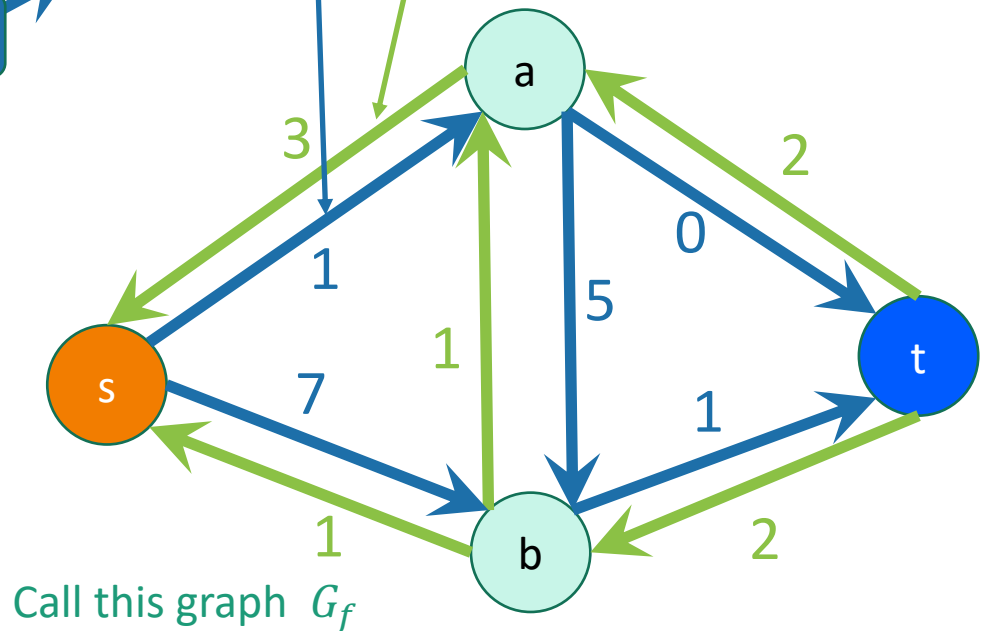


This forward edge has weight "capacity - flow".

This backward edge has weight "flow".

Call the flow f
Call the graph G

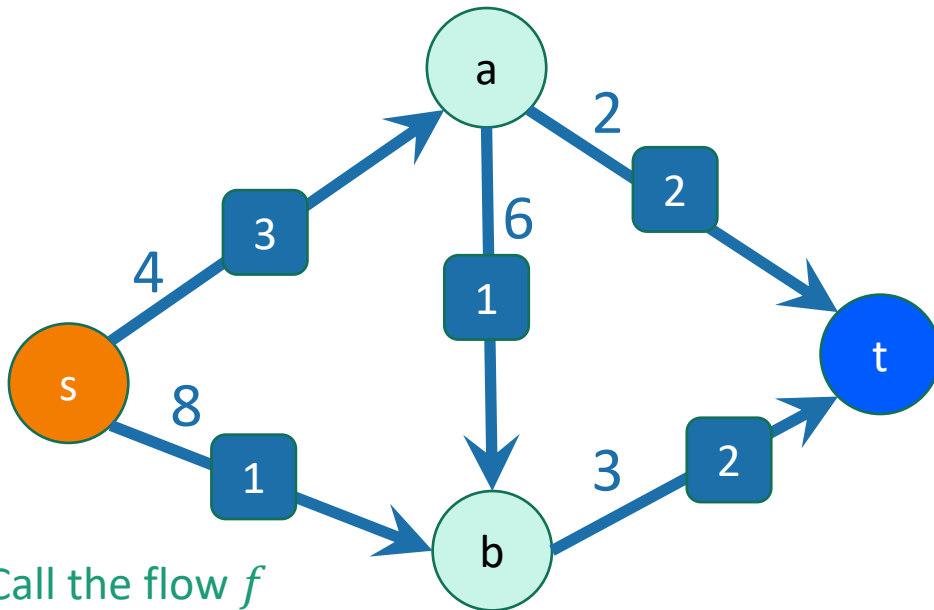
Create a new **residual**
network from this flow:



Call this graph G_f

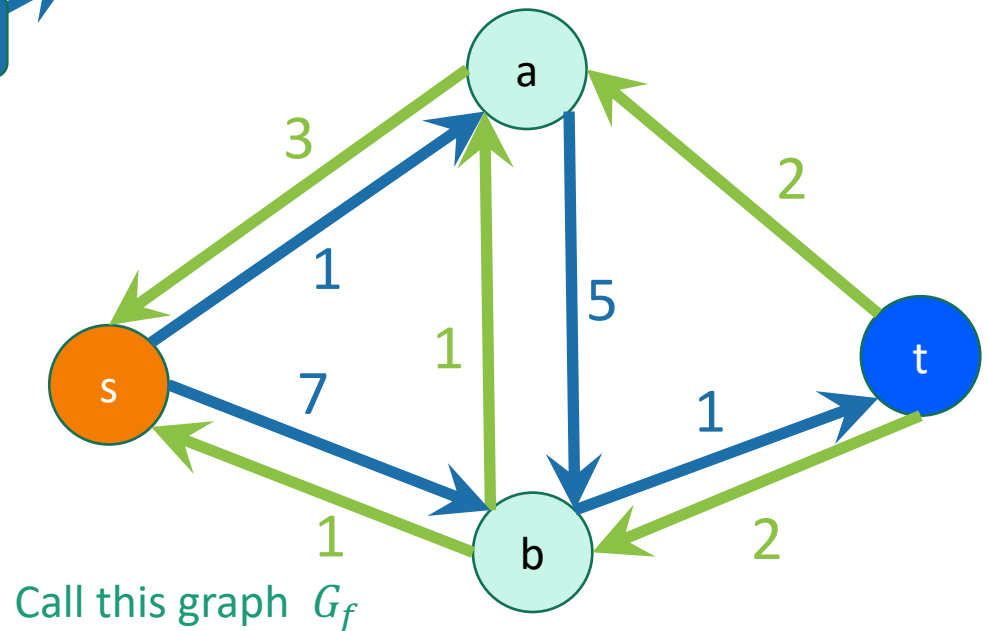
Tool: Residual networks

Say we have a flow



Call the flow f
Call the graph G

Create a new **residual network** from this flow:

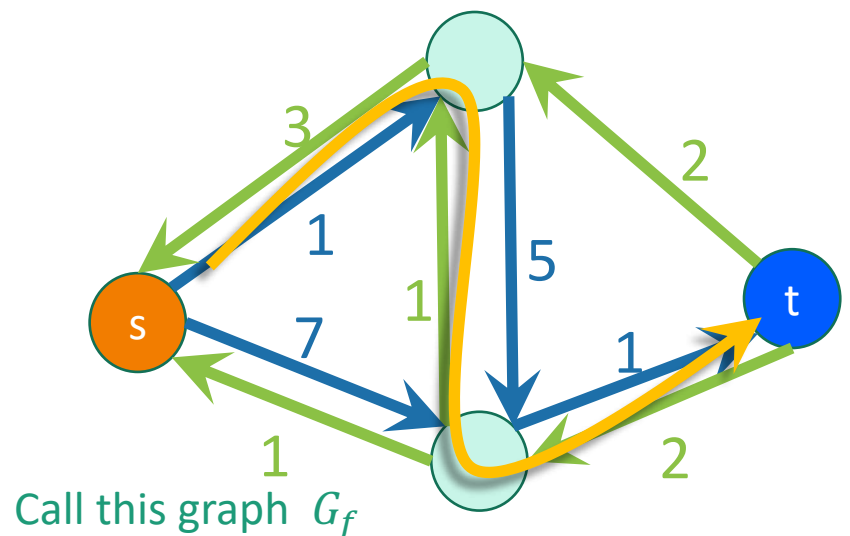
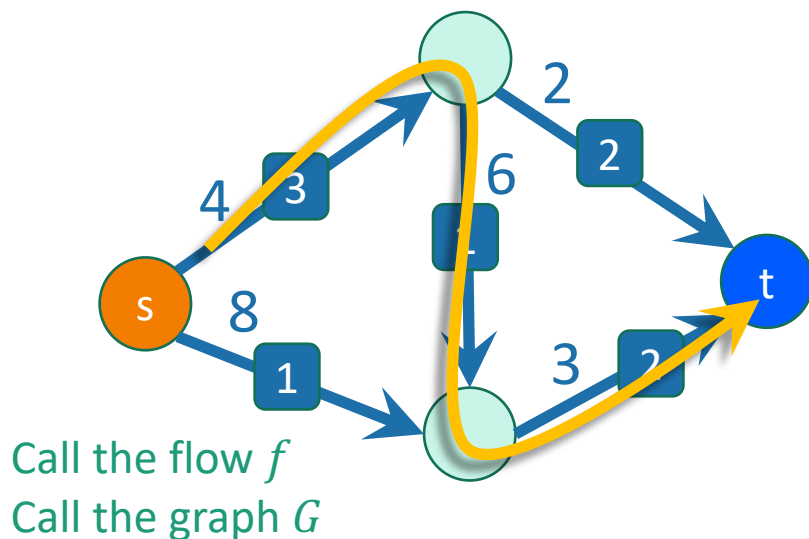


Forward edges are the amount that's left.
Backwards edges are the amount that's been used.

Call this graph G_f

Residual networks tell us how to improve the flow.

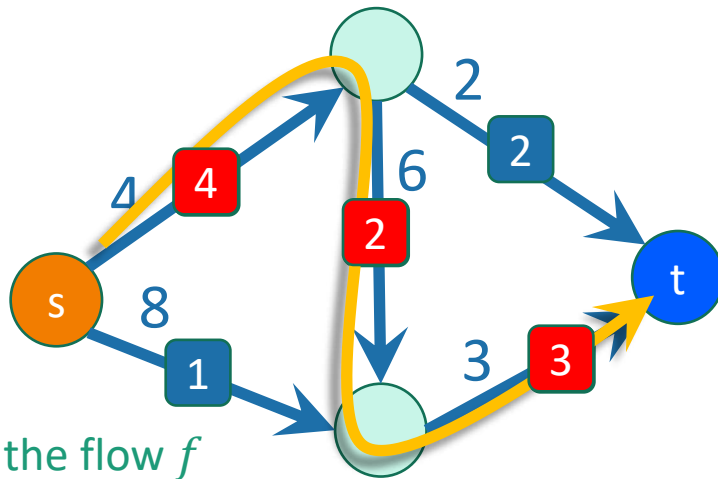
- **Definition:** A path from s to t in the residual network is called an **augmenting path**.
- **Claim:** If there is an augmenting path, we can increase the flow along that path.



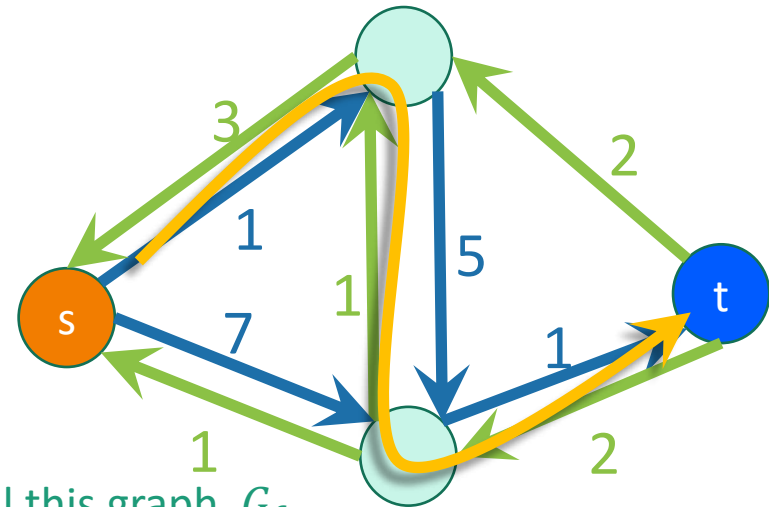
claim:

if there is an augmenting path, we can increase the flow along that path.

- Easy case: every edge on the path in G_f is a **forward edge**.



Call the flow f
Call the graph G



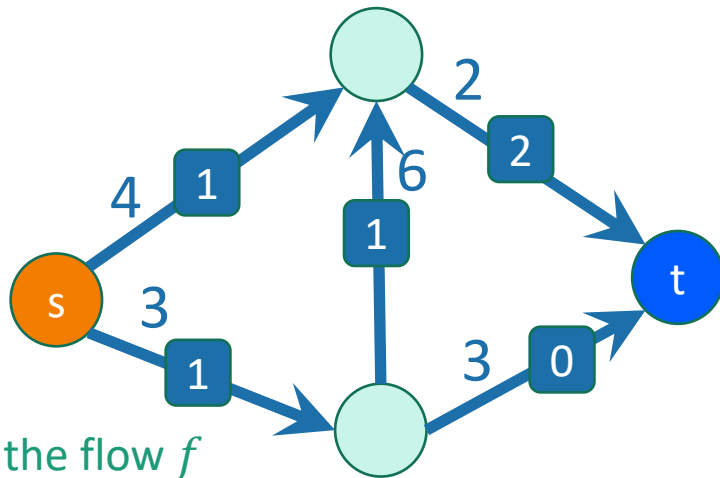
Call this graph G_f

- Forward edges indicate how much stuff can still go through.
- Just increase the flow on all the edges!

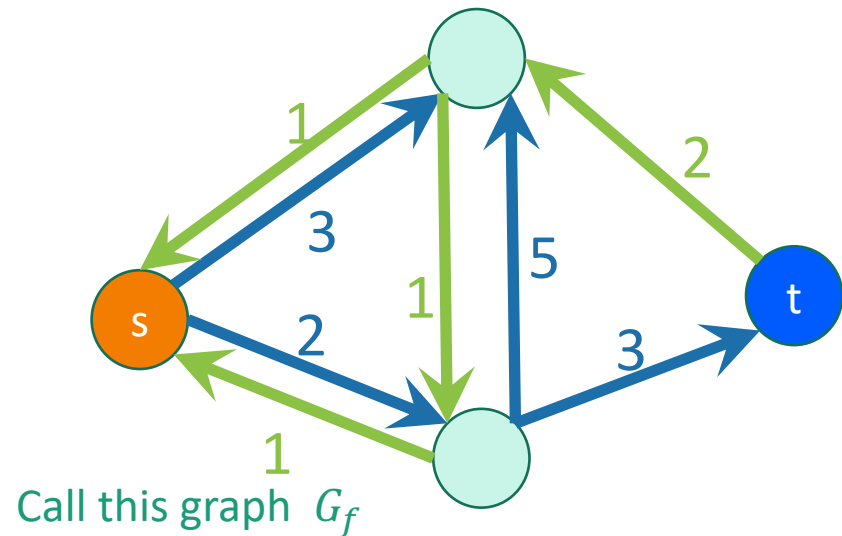
claim:

if there is an augmenting path, we can increase the flow along that path.

- Harder case: there are **backward edges** in the path.
 - Here's a slightly different example of a flow:



Call the flow f
Call the graph G



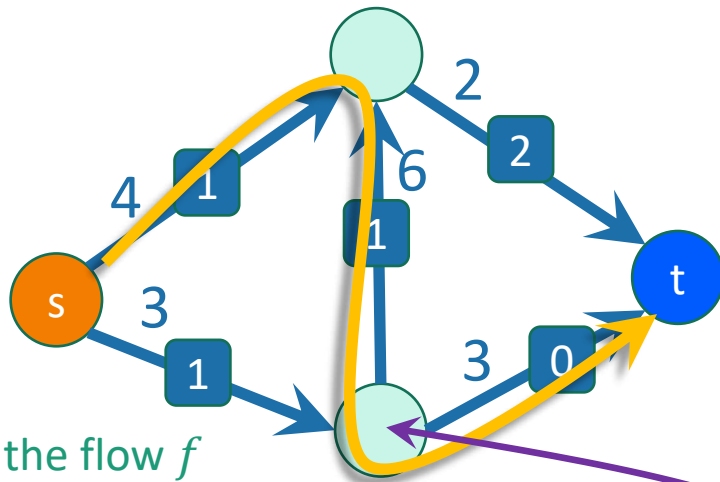
Call this graph G_f

I changed some of the weights and edge directions.

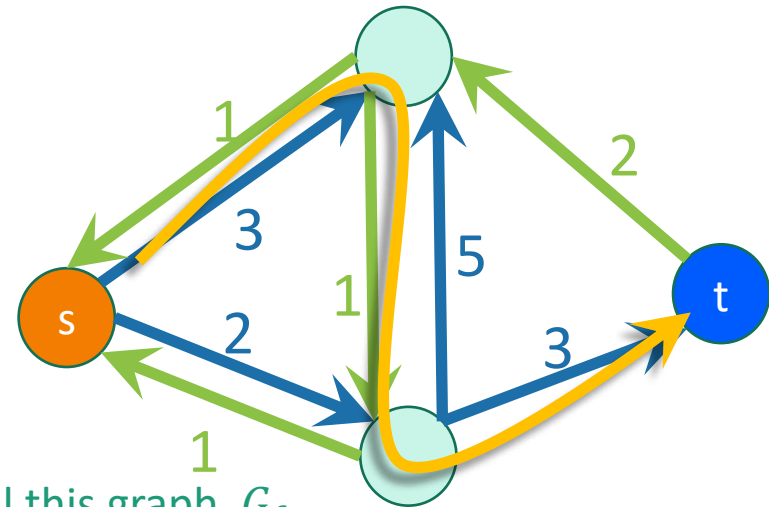
claim:

if there is an augmenting path, we can increase the flow along that path.

- Harder case: there are **backward edges** in the path.
 - Here's a slightly different example of a flow:



Call the flow f
Call the graph G



Call this graph G_f

Now we should NOT increase the flow at all the edges along the path!

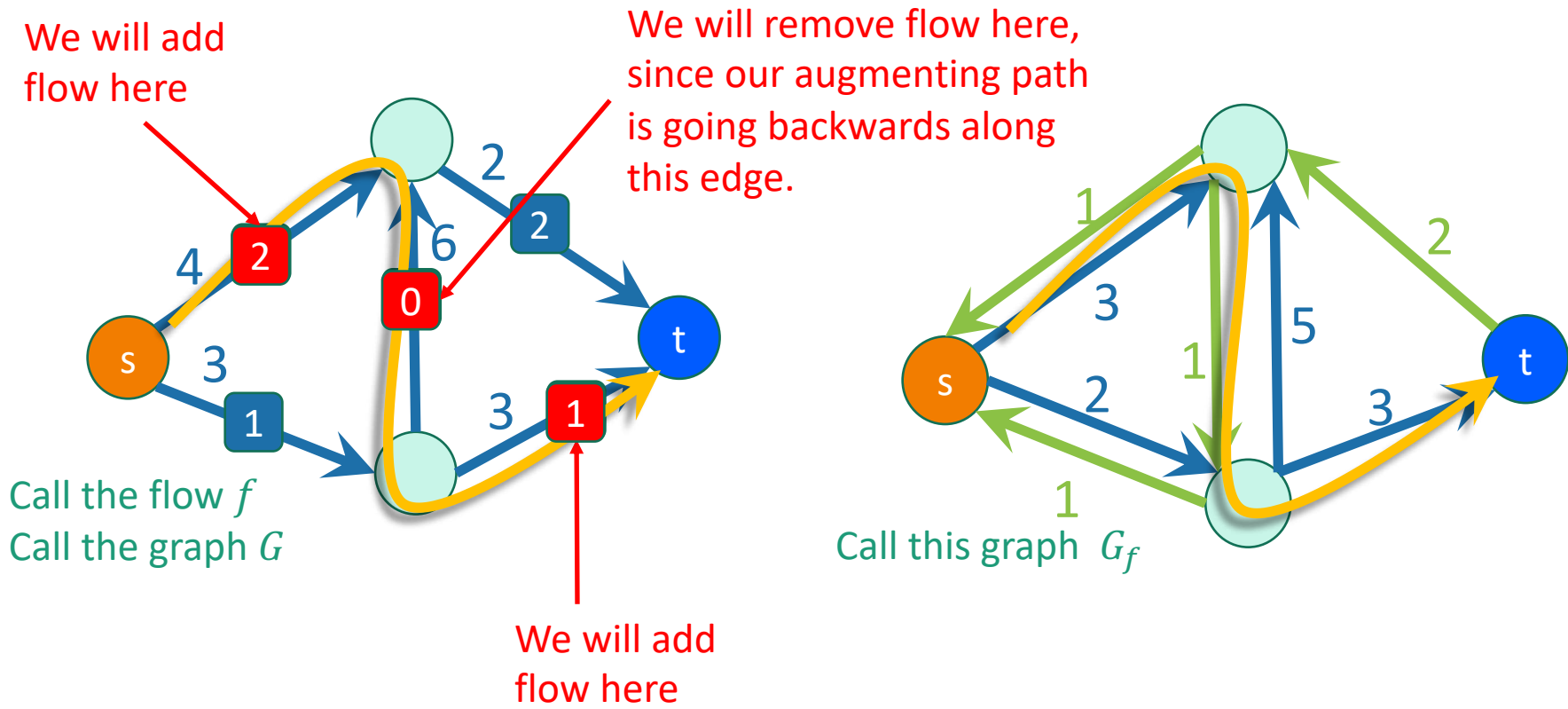
- For example, that will mess up the conservation of stuff at this vertex.

I changed some of the weights and edge directions.

claim:

if there is an augmenting path, we can increase the flow along that path.

- In this case we do something a bit different:

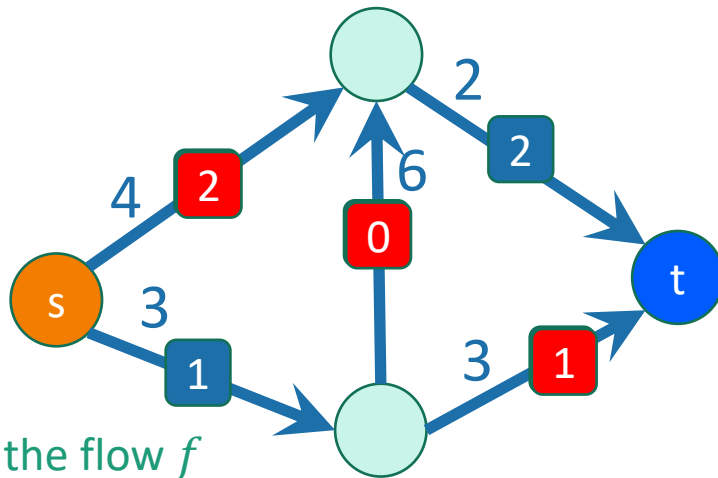


claim:

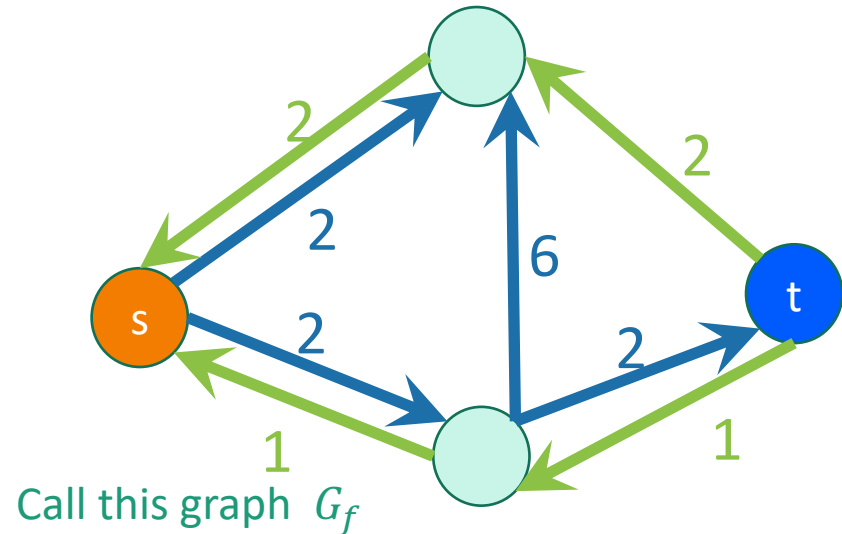
if there is an augmenting path, we can increase the flow along that path.

- In this case we do something a bit different:

Then we'll update the residual graph:

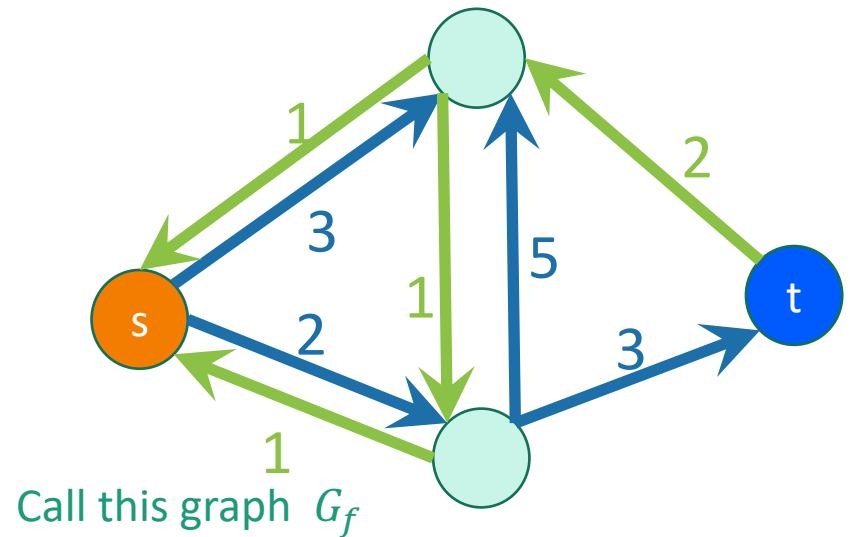
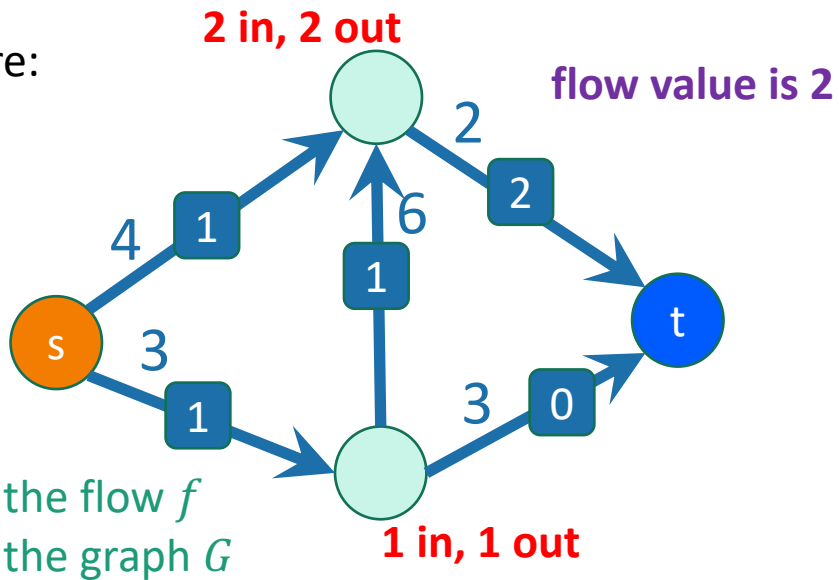


Call the flow f
Call the graph G

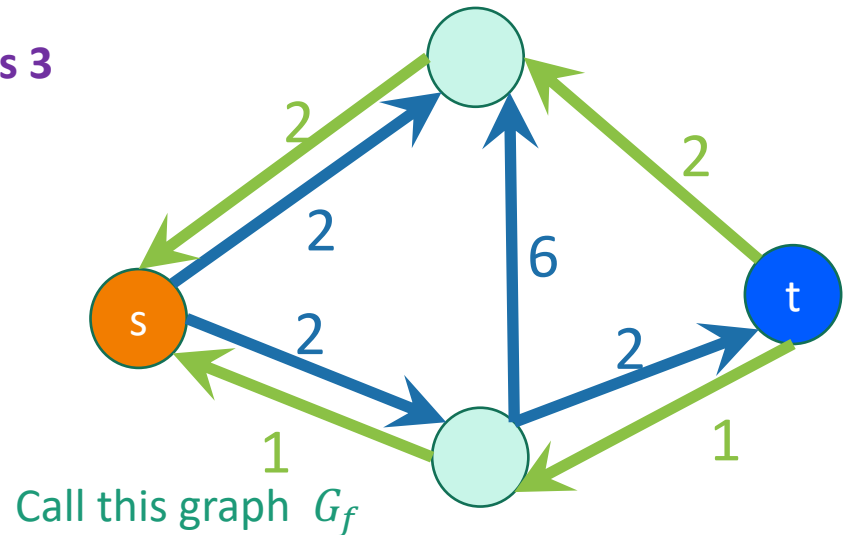
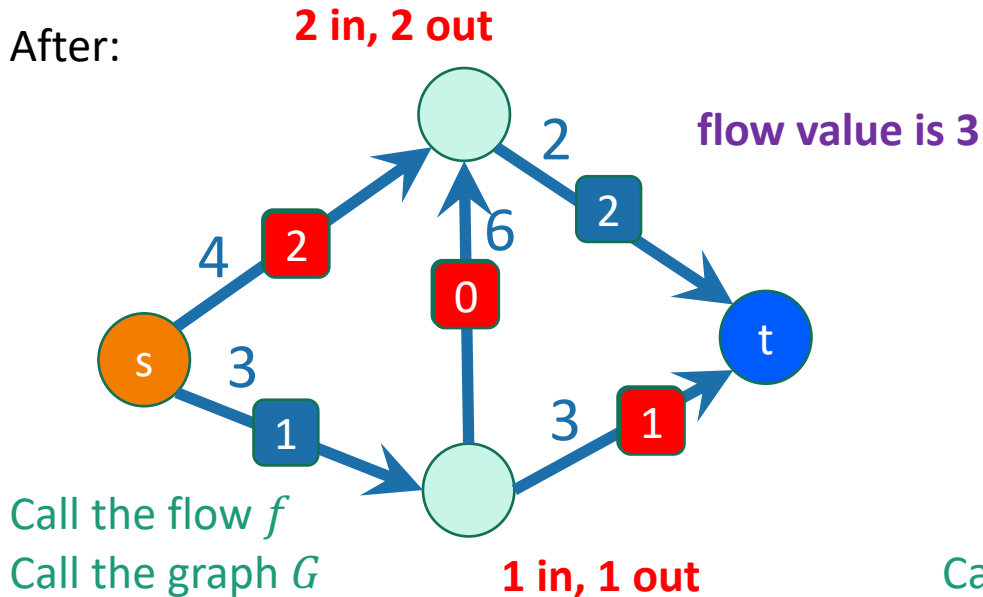


Call this graph G_f

Before:



After:

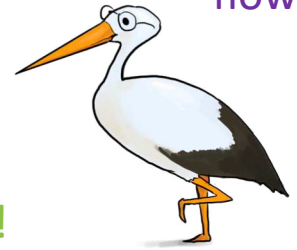


Still a legit flow, but with a bigger value!

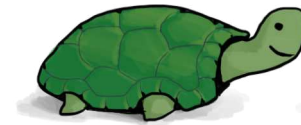
claim:

if there is an augmenting path, we can increase the flow along that path.

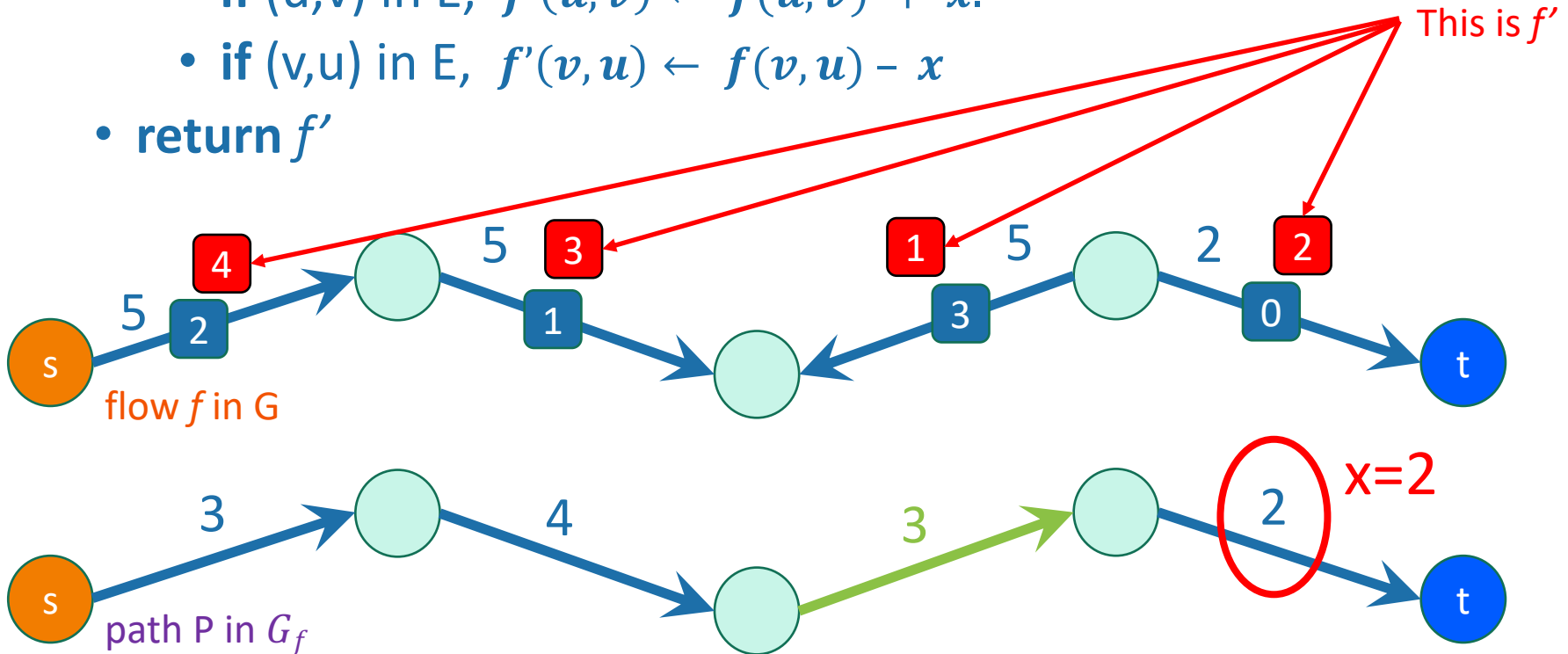
Check that this always makes a bigger (and legit) flow!



Seriously!
Pause the video now and check!!



- `increaseFlow(path P in G_f , flow f):`
 - $x = \min$ weight on any edge in P
 - **for** (u,v) in P:
 - **if** (u,v) in E, $f'(u,v) \leftarrow f(u,v) + x$.
 - **if** (v,u) in E, $f'(v,u) \leftarrow f(v,u) - x$
 - **return** f'



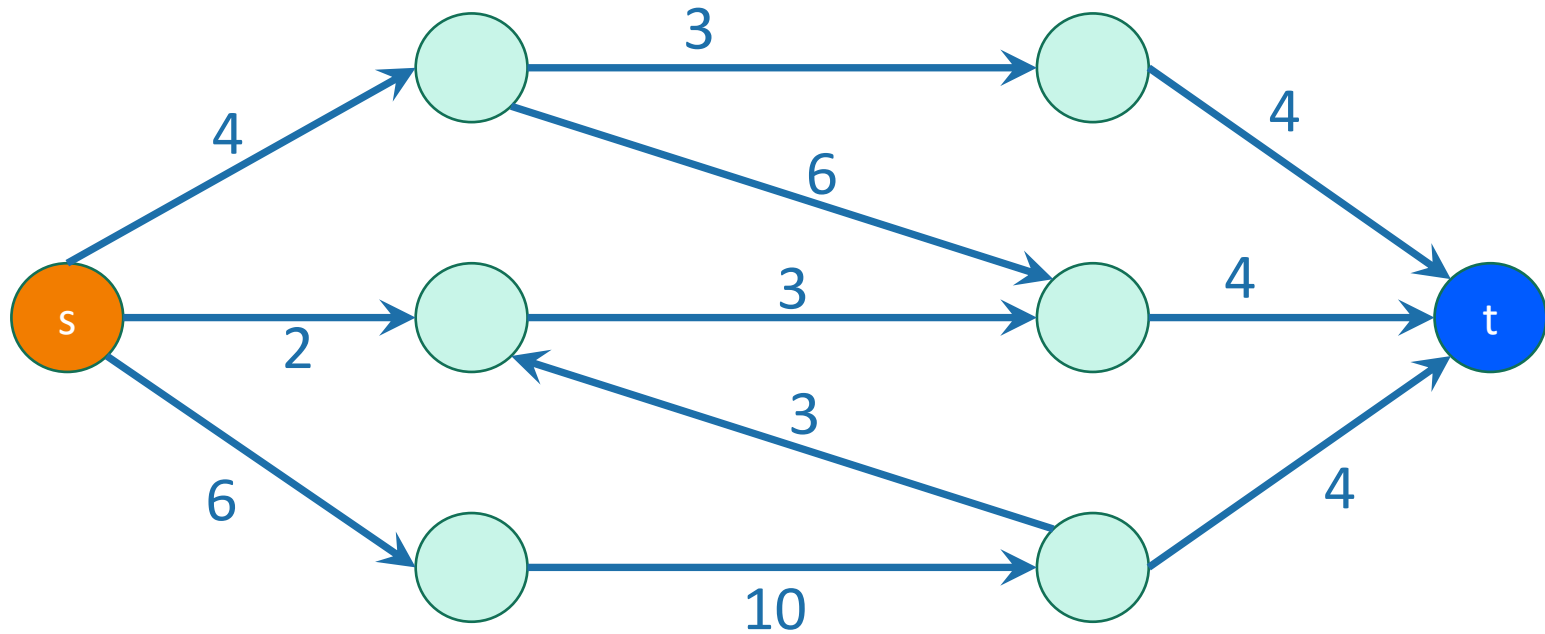
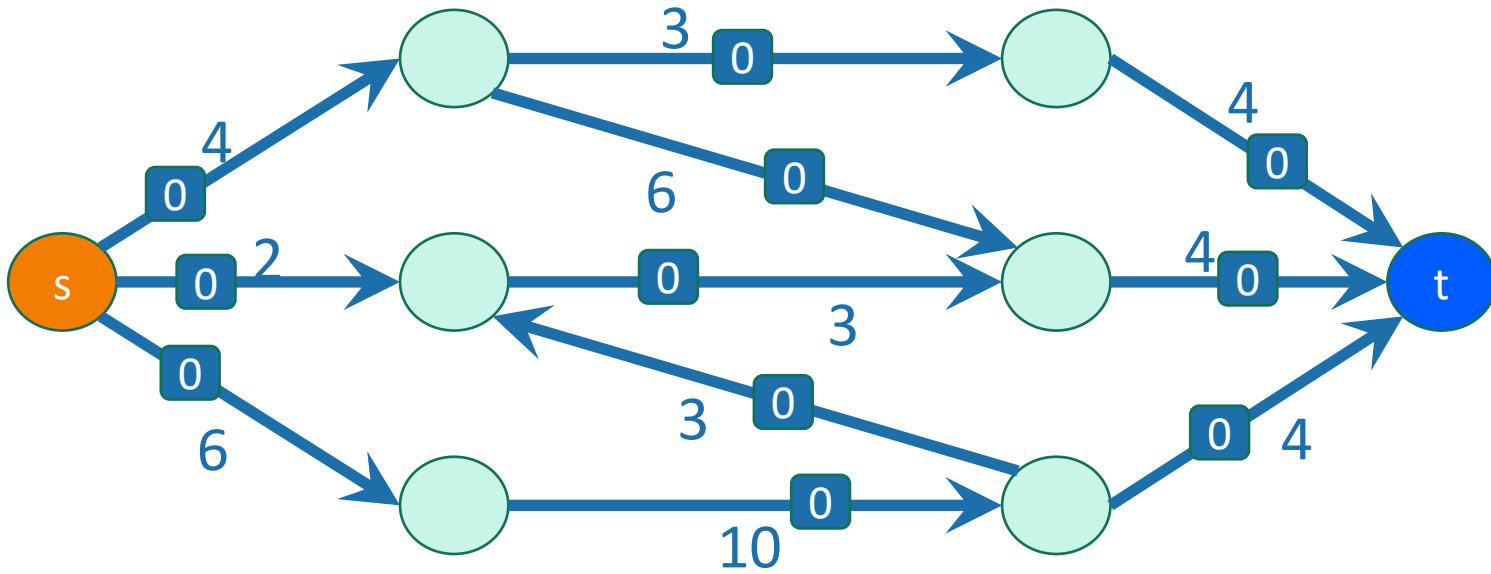
Ford-Fulkerson Algorithm

- **Ford-Fulkerson(G):**

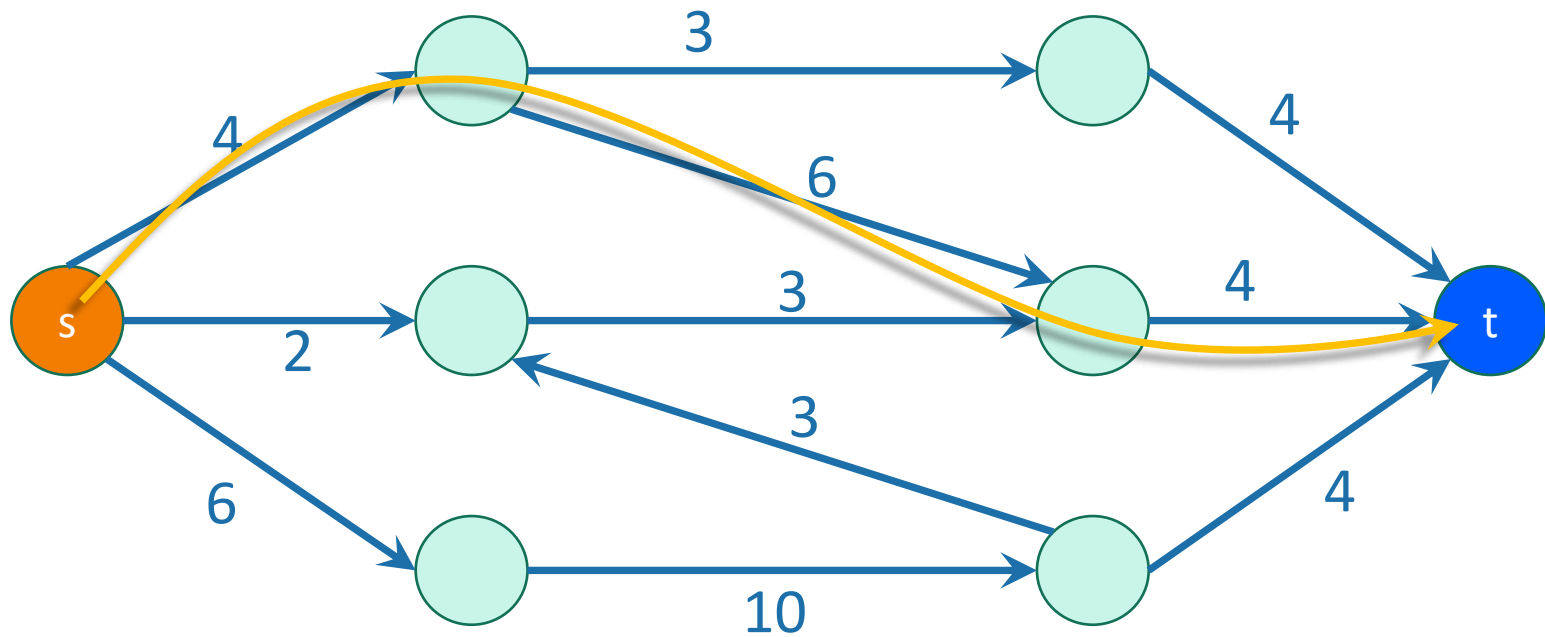
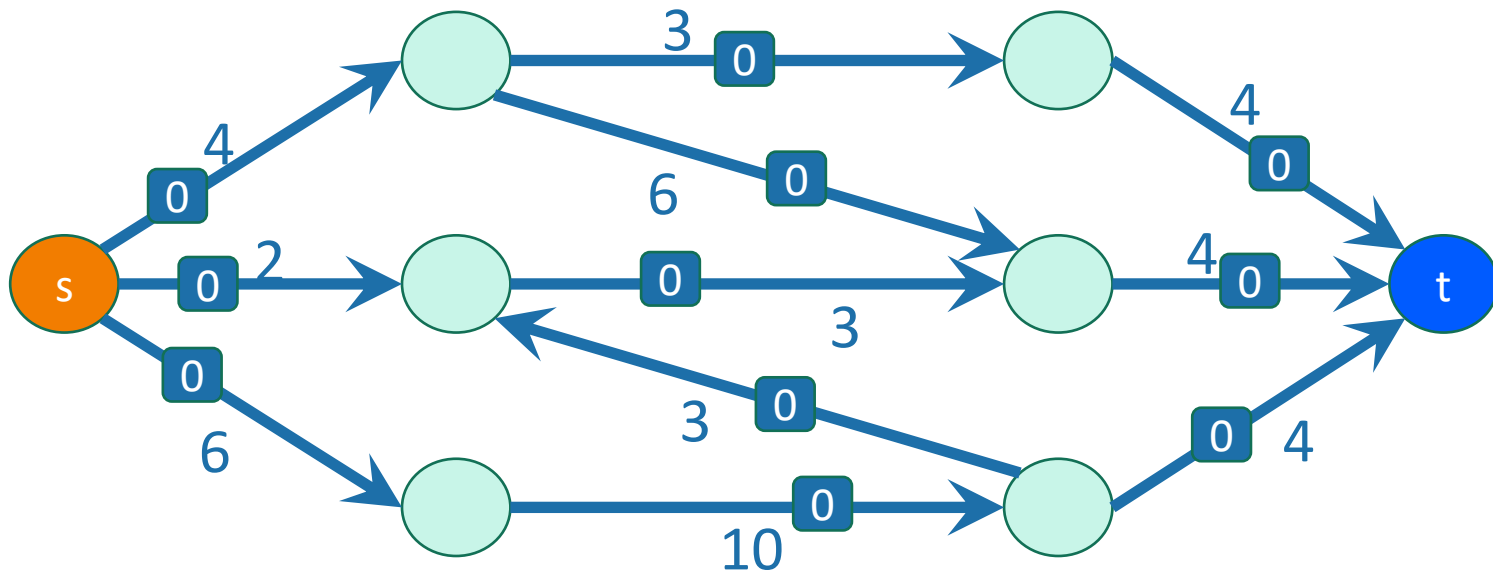
- $f \leftarrow$ all zero flow.
- $G_f \leftarrow G$
- **while** t is reachable from s in G_f
 - Find a path P from s to t in G_f
 - $f \leftarrow$ **increaseFlow**(P, f)
 - update G_f
- **return** f

// eg, use BFS

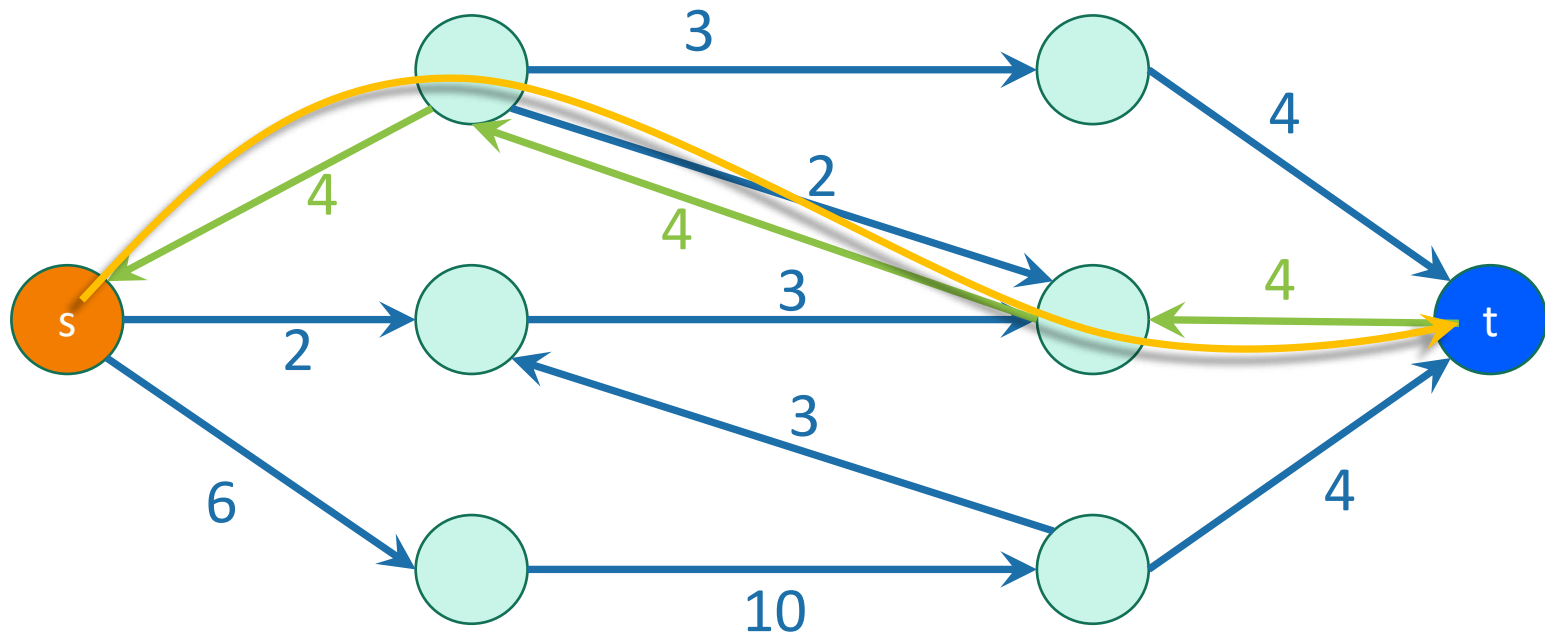
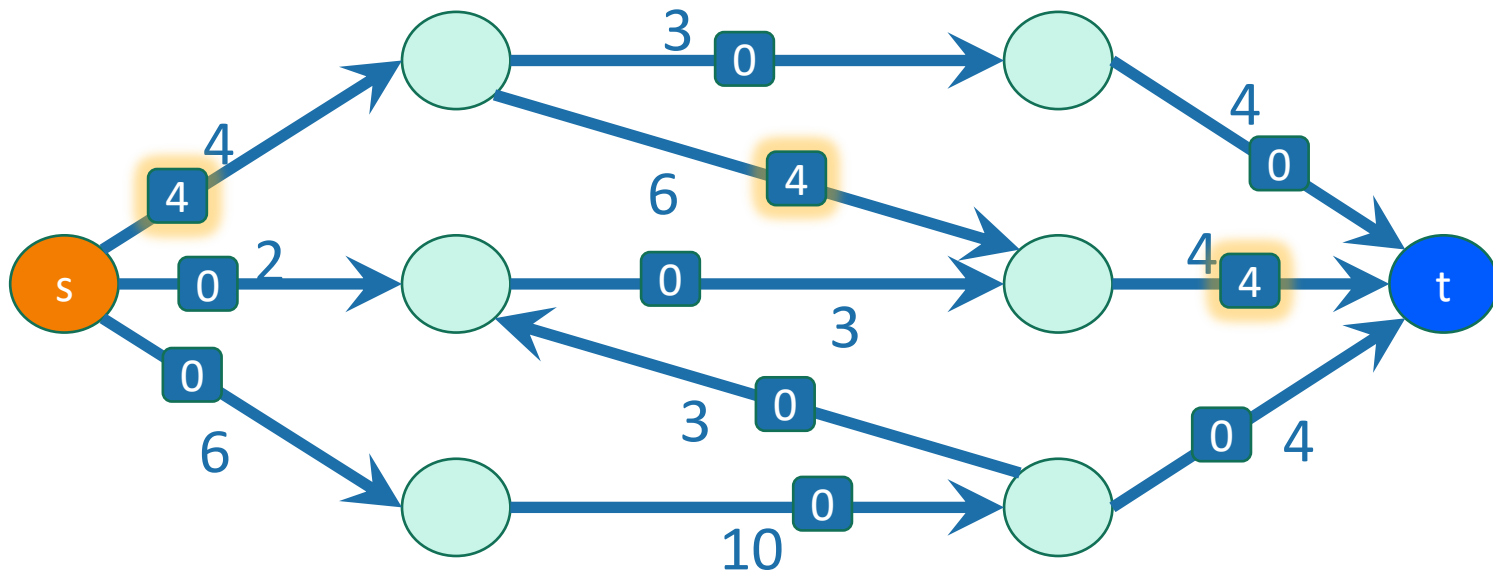
Example of Ford-Fulkerson



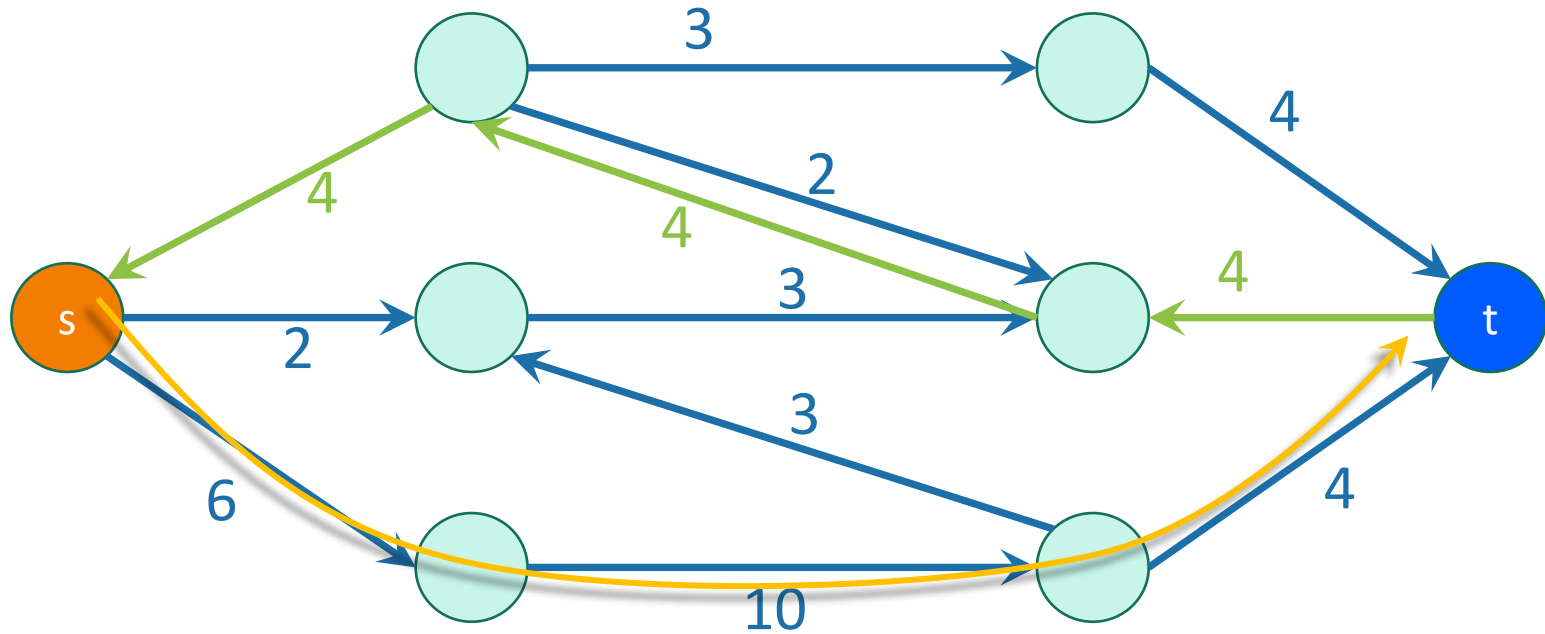
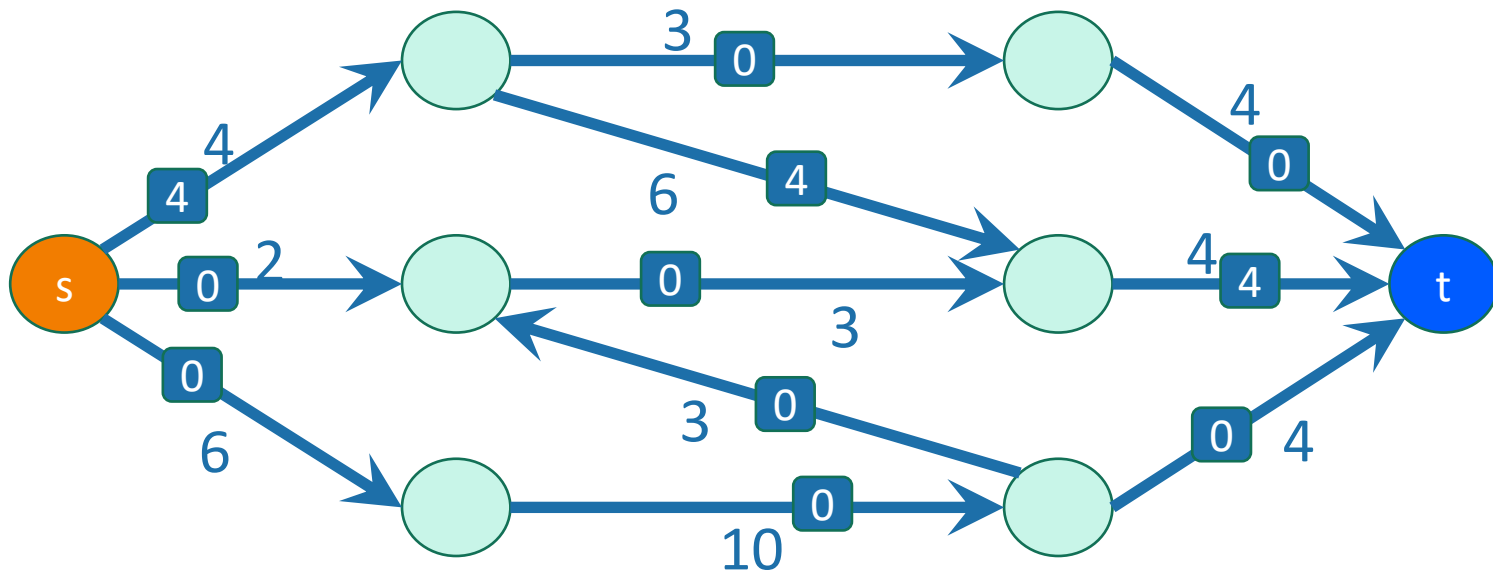
Example of Ford-Fulkerson



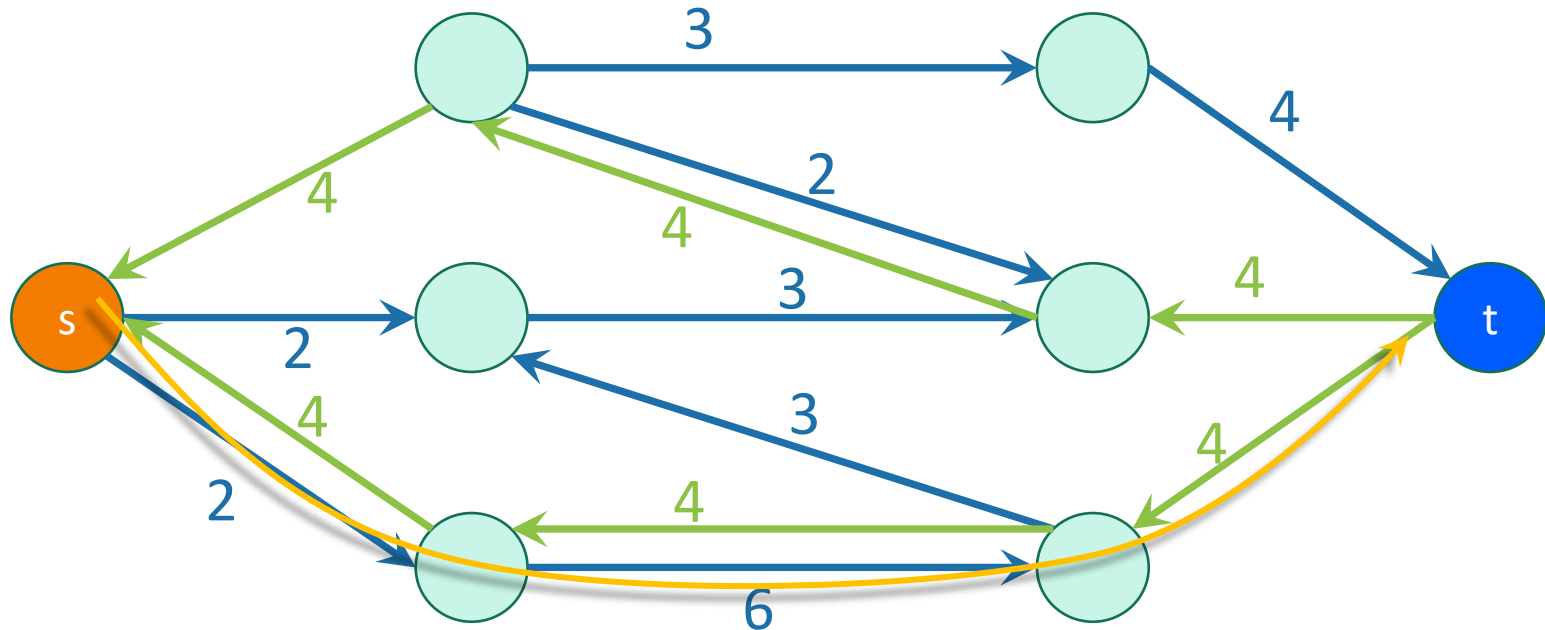
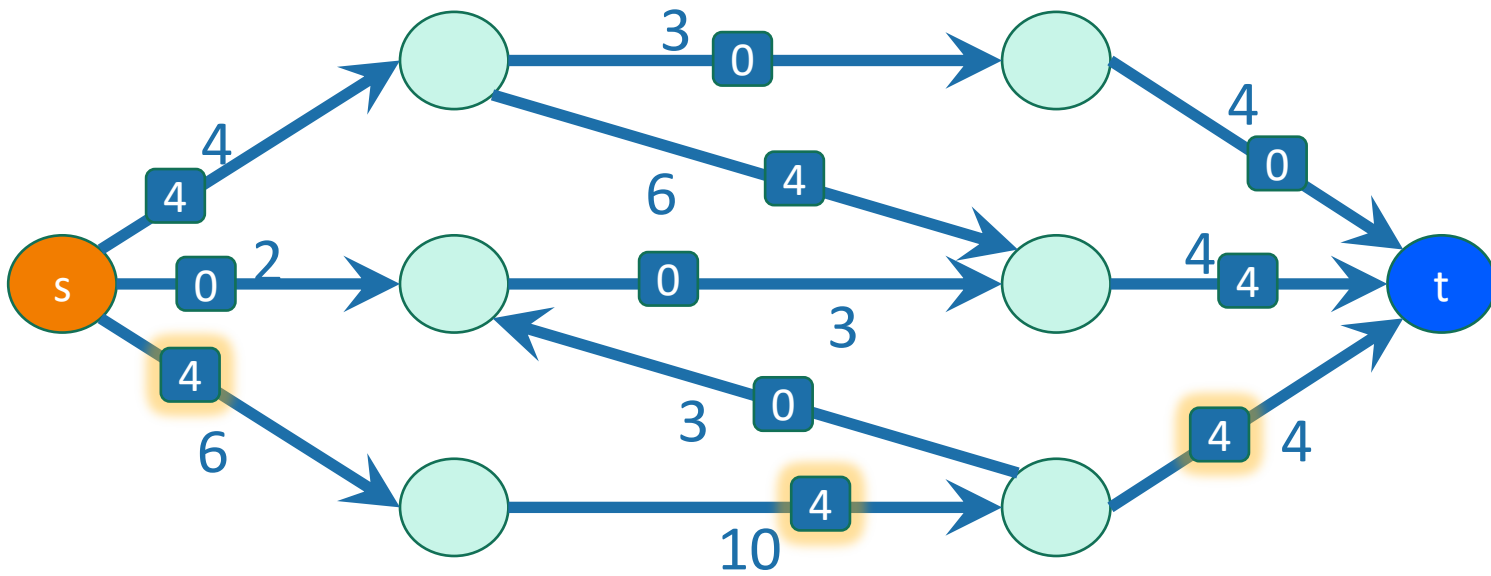
Example of Ford-Fulkerson



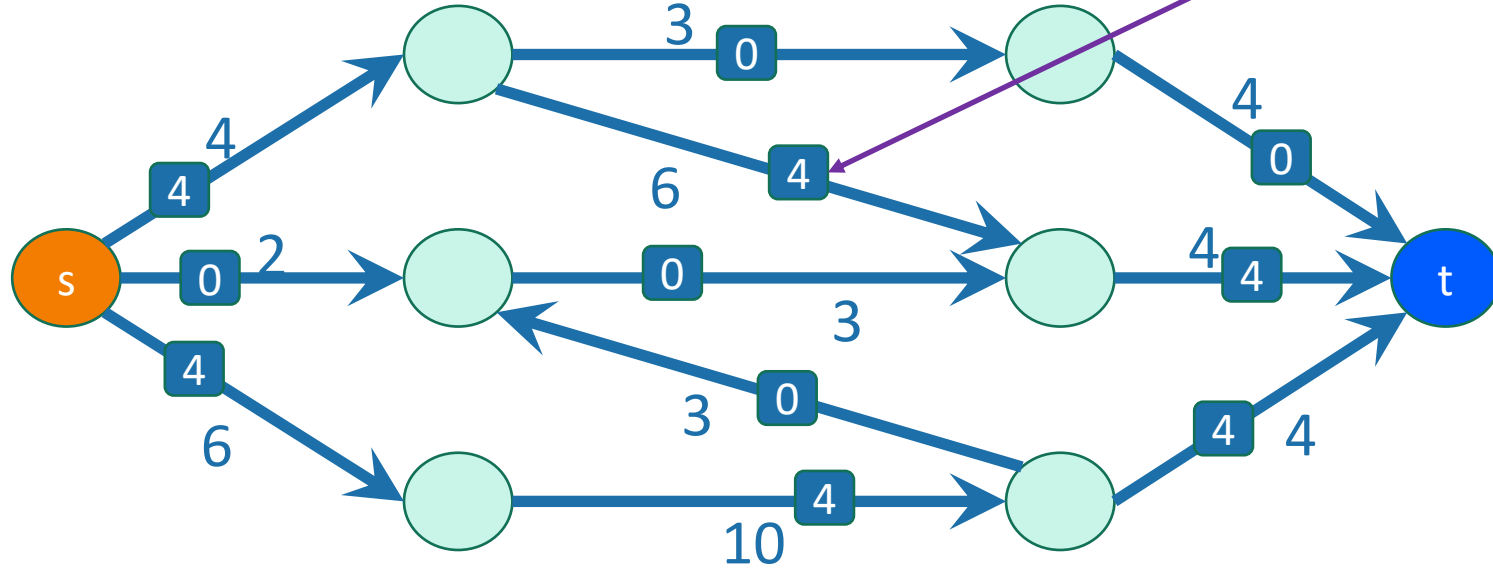
Example of Ford-Fulkerson



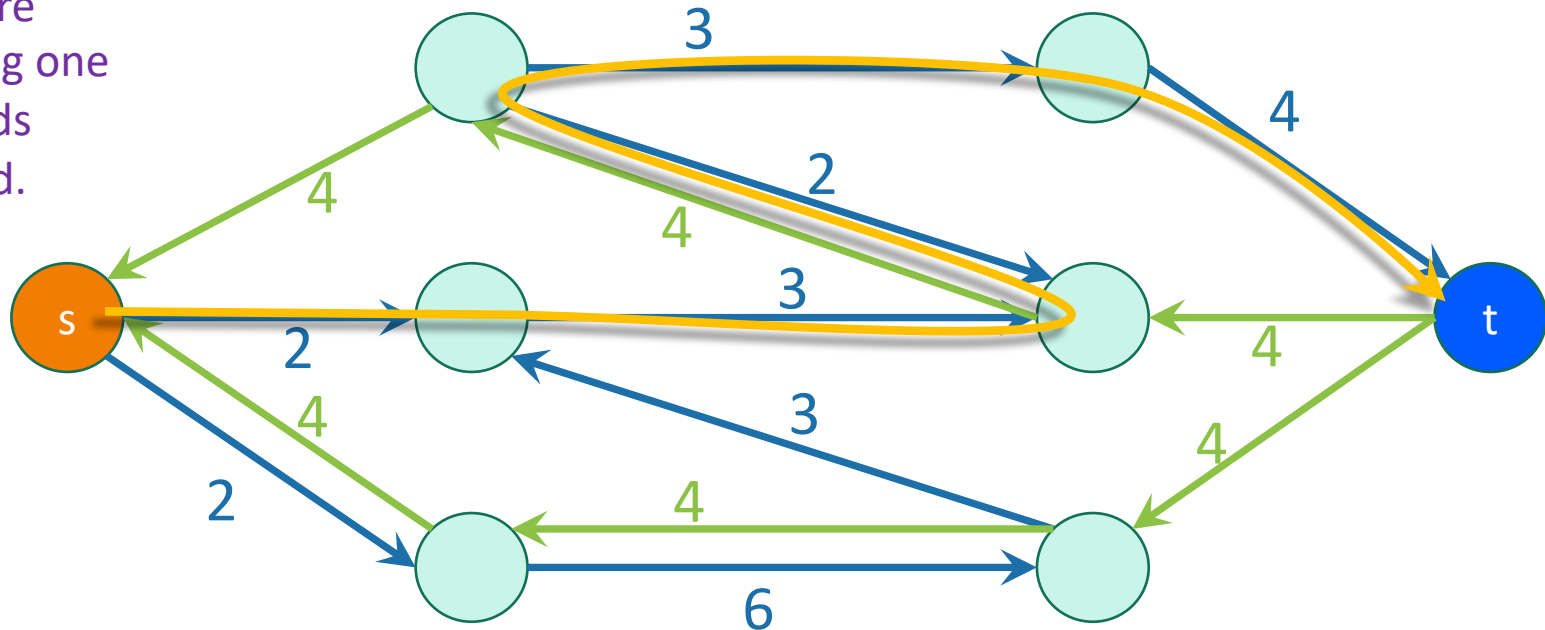
Example of Ford-Fulkerson



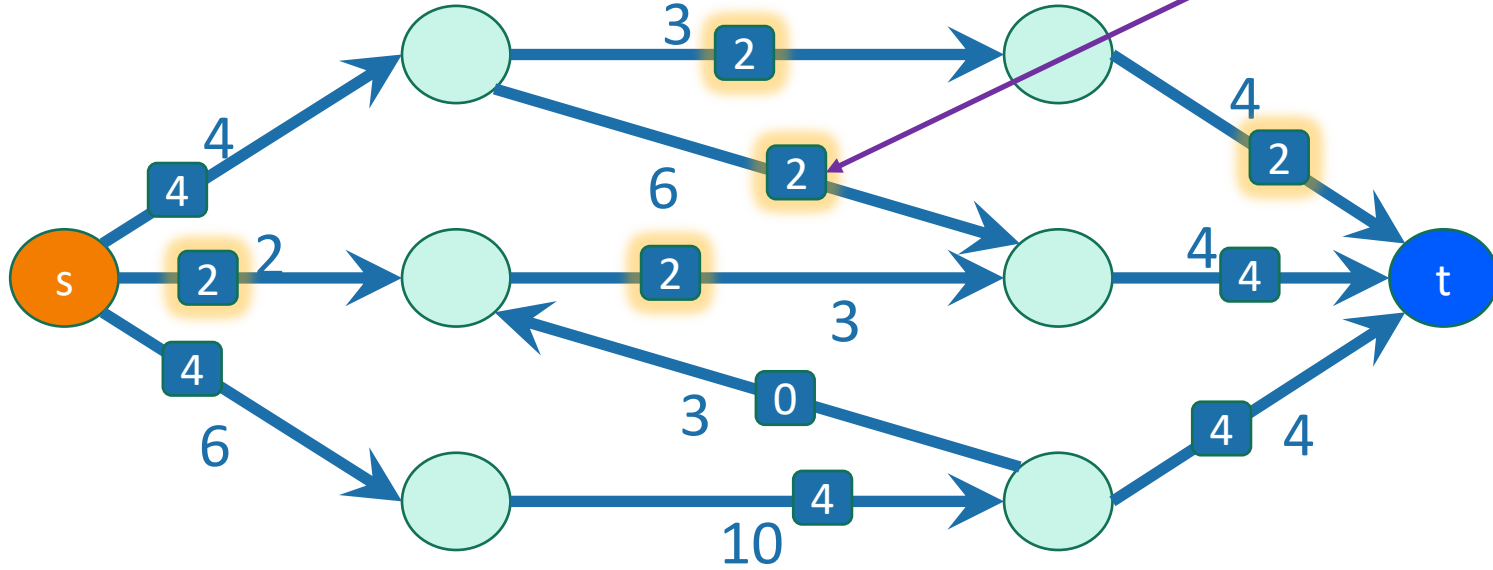
Example of Ford-Fulkerson



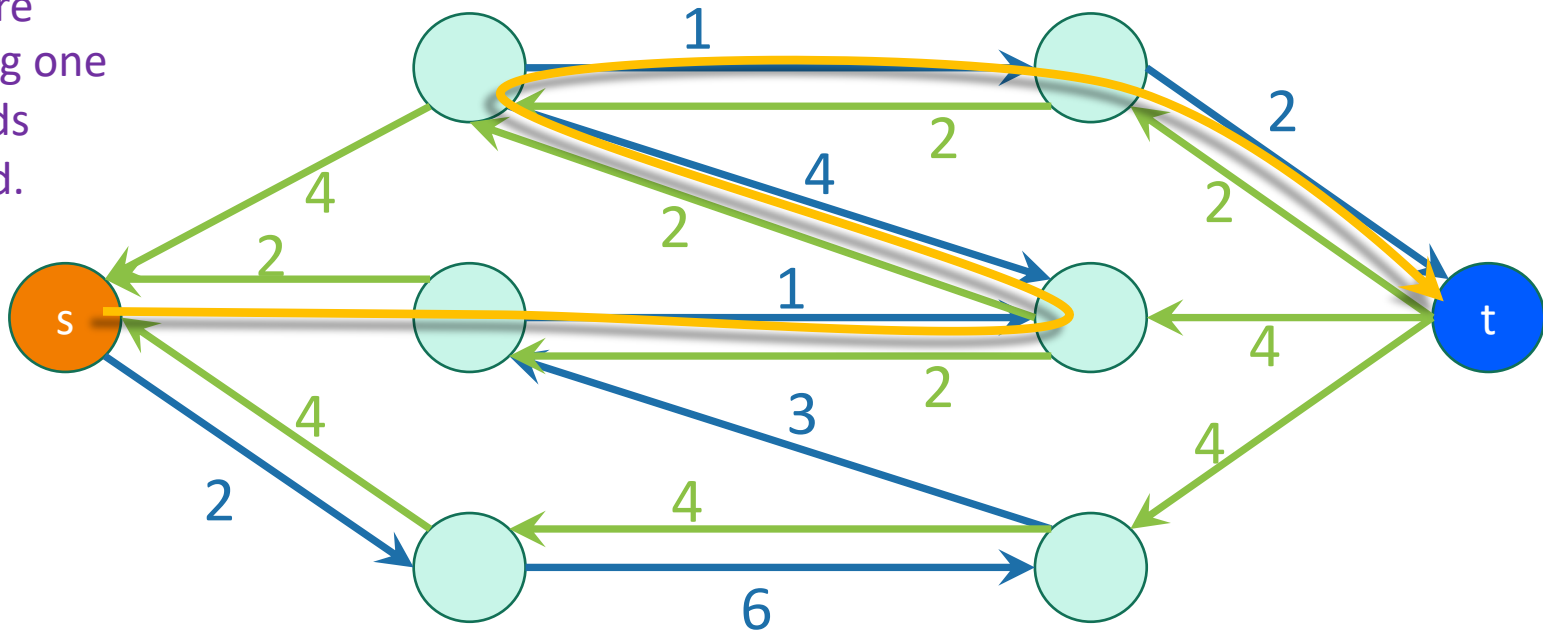
Notice that we're going back along one of the backwards edges we added.



Example of Ford-Fulkerson

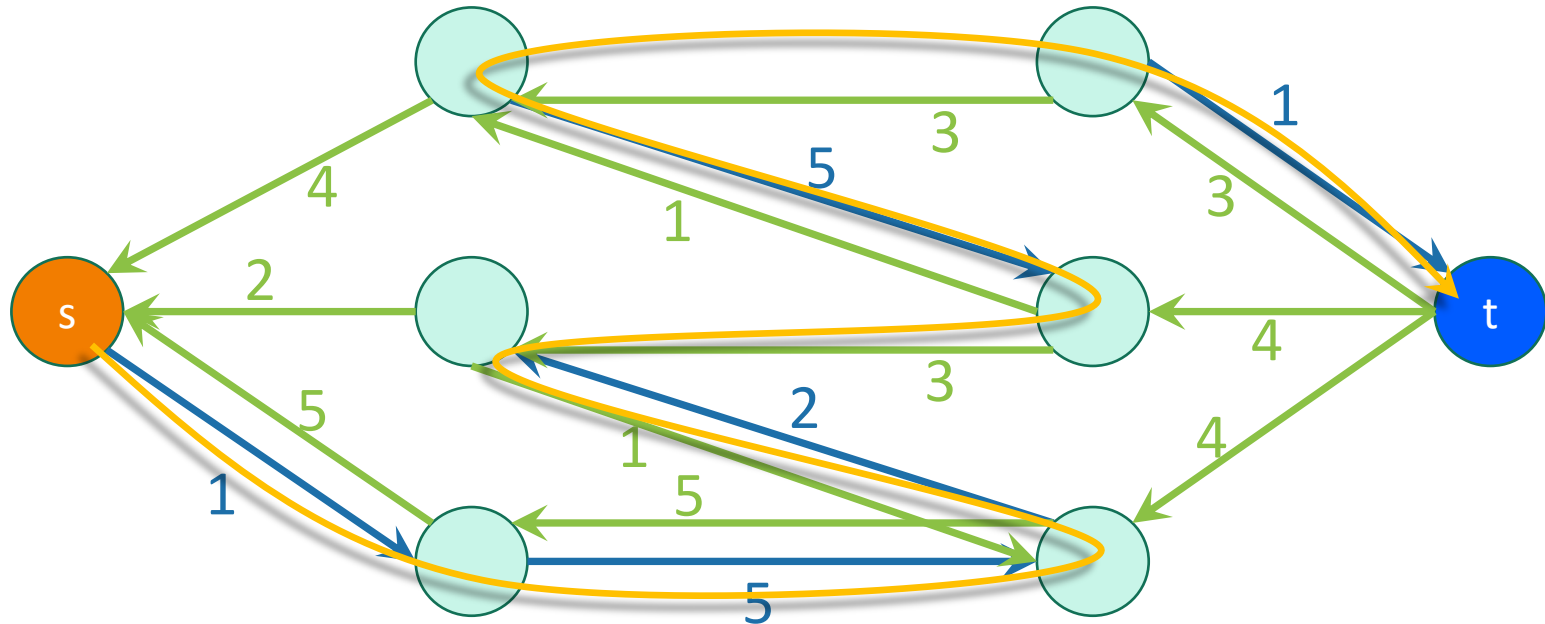
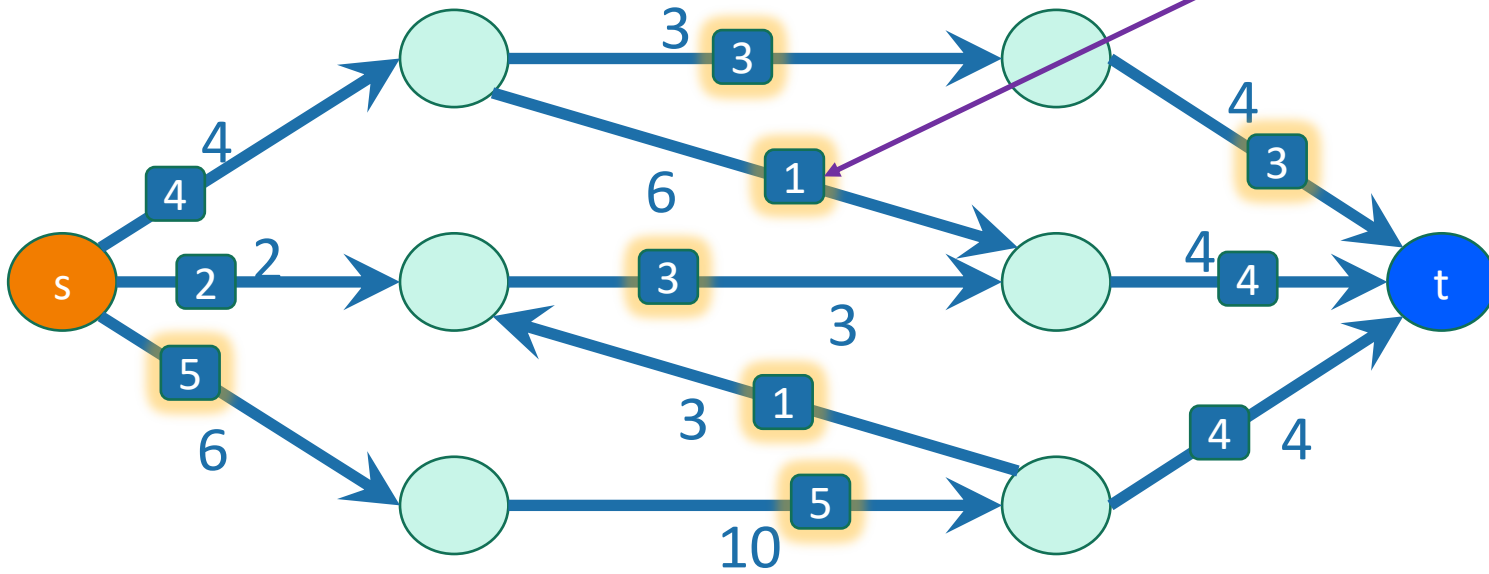


Notice that we're going back along one of the backwards edges we added.

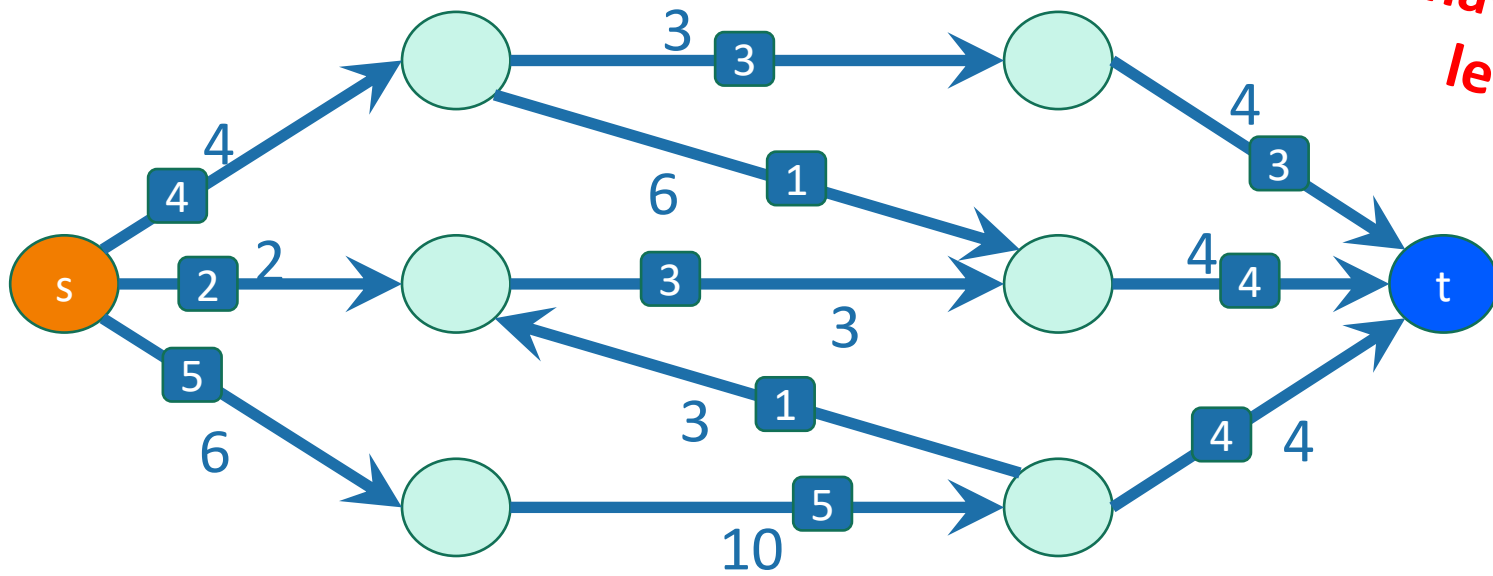


Example of Ford-Fulkerson

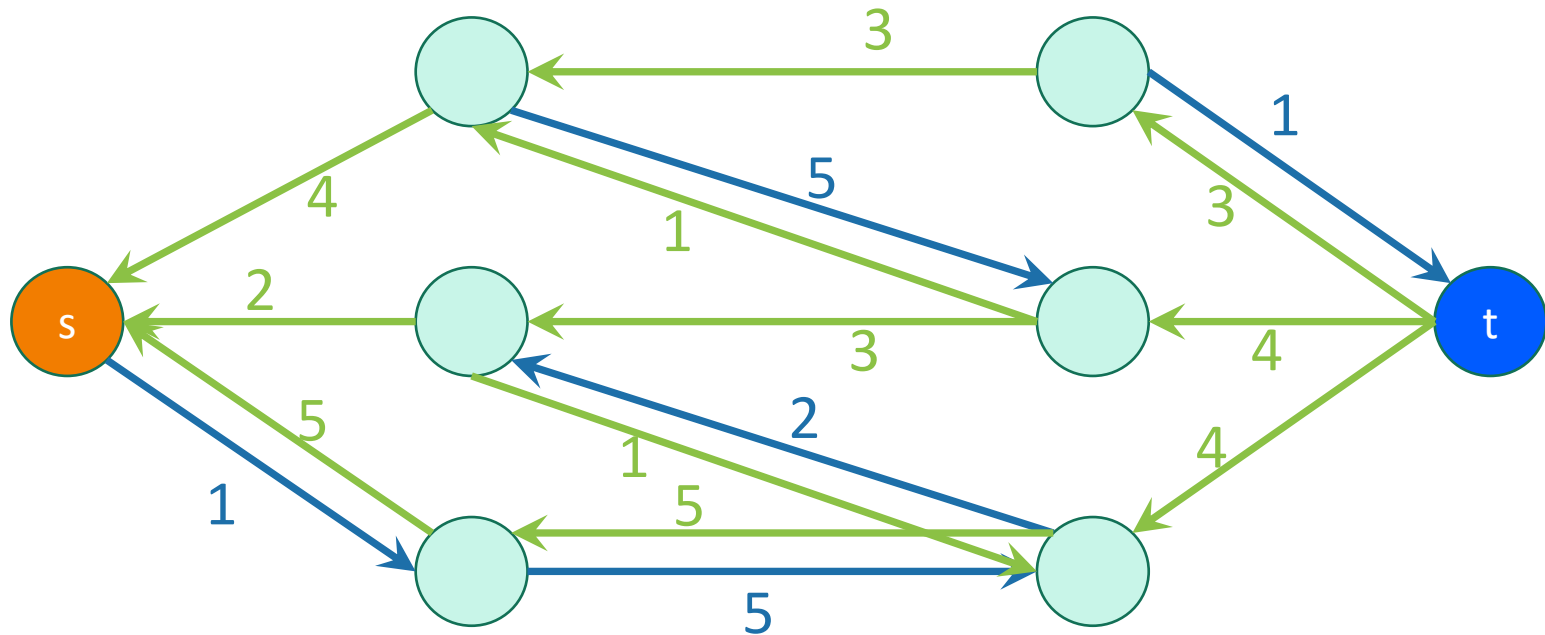
We will remove flow from this edge AGAIN.



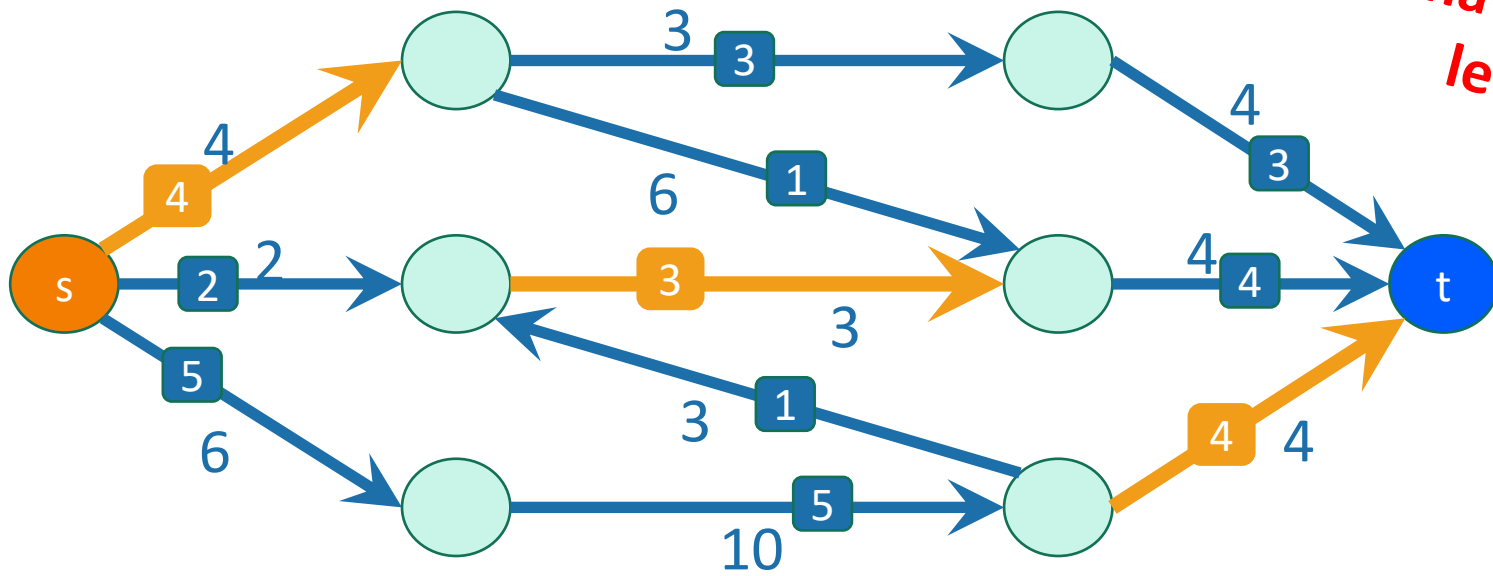
Example of Ford-Fulkerson



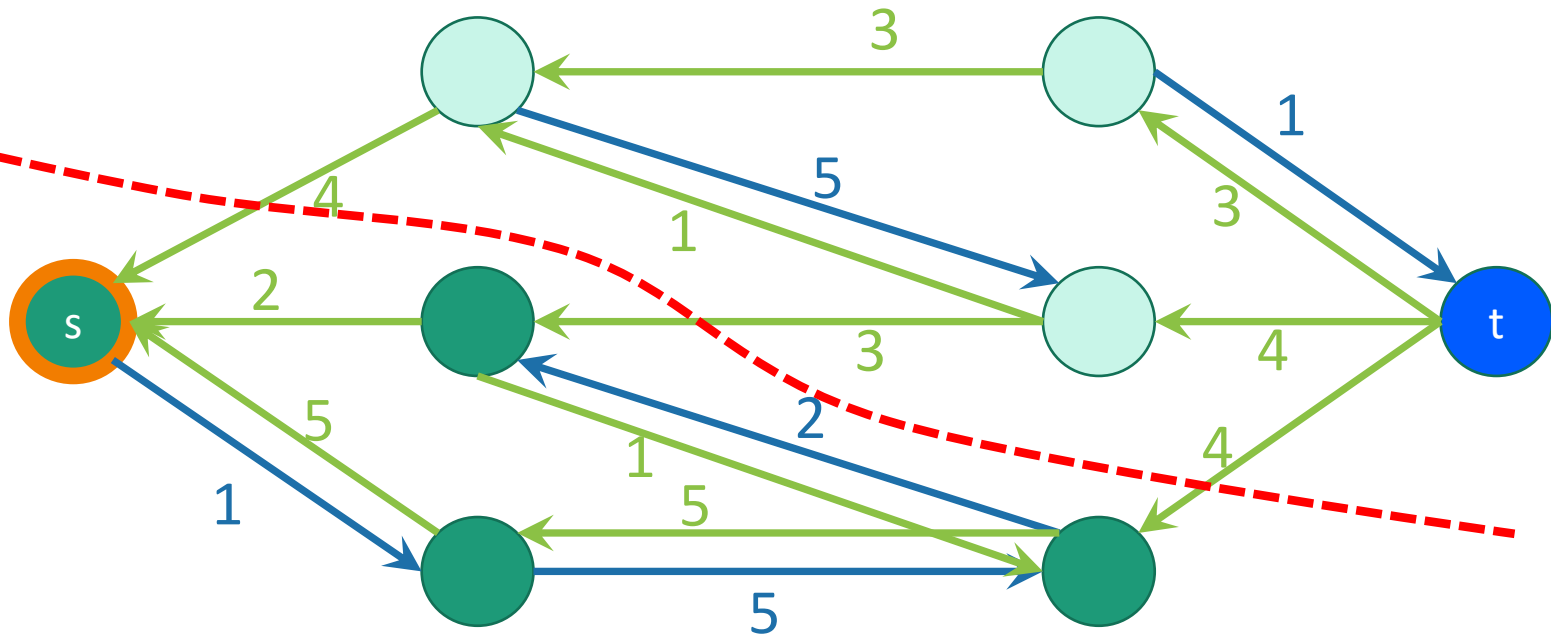
*Now we
have nothing
left to do!*



Example of Ford-Fulkerson



Now we have nothing left to do!



There's no path from s to t , and here's the cut to prove it.

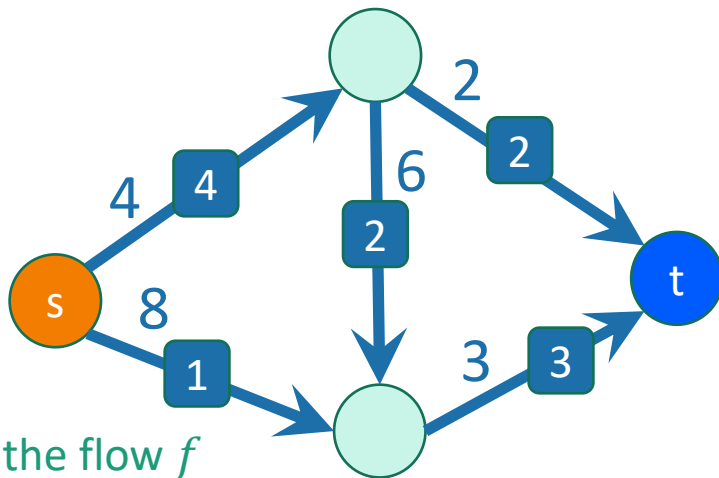
Why does Ford-Fulkerson work?

- Just because we can't improve the flow anymore using an augmenting path, does that mean there isn't a better flow?
- **Lemma 2:** If there is no augmenting path in G_f then f is a maximum flow.

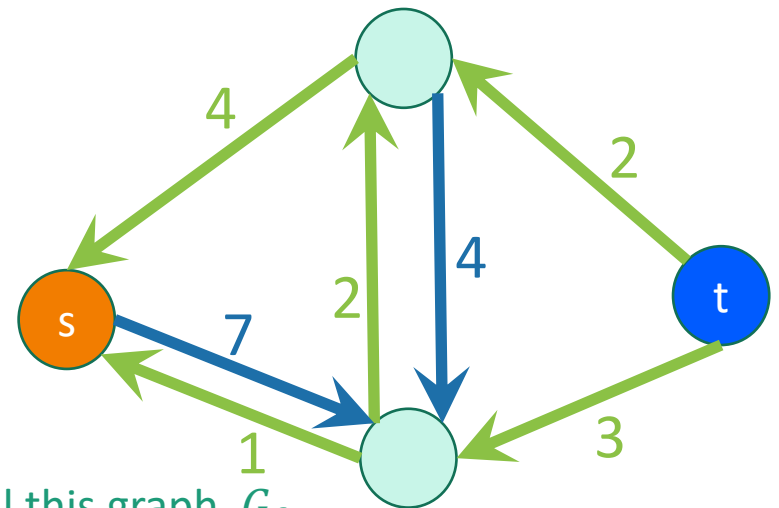
No augmenting path \Rightarrow max flow.

- Suppose there is not a path from s to t in G_f .
- Consider the cut given by:

{things reachable from s } , **{things not reachable from s }**



Call the flow f
Call the graph G



Call this graph G_f

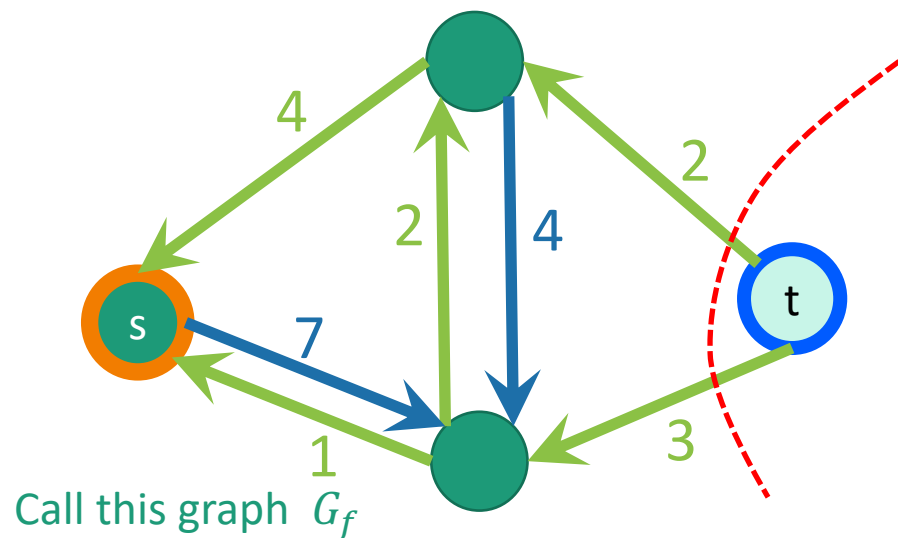
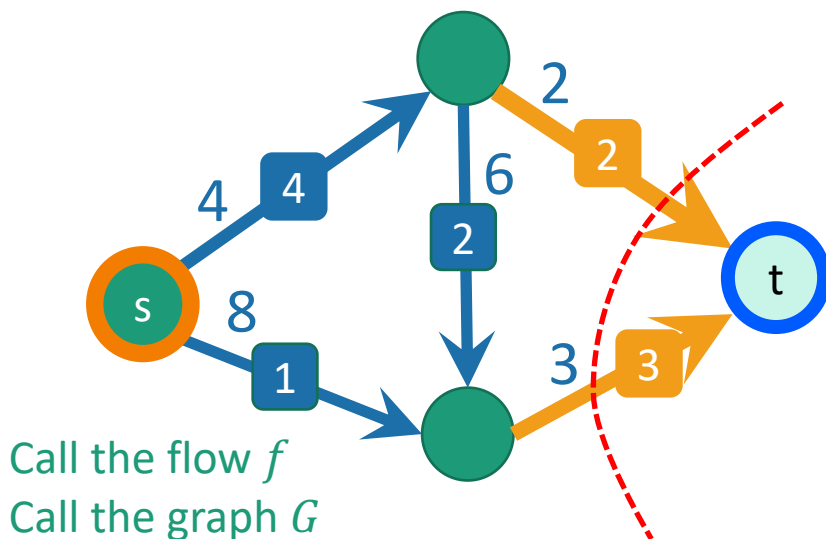
No augmenting path \Rightarrow max flow.

- Suppose there is not a path from s to t in G_f .
- Consider the cut given by:

{things reachable from s } , **{things not reachable from s }**

t lives here

- The value of the flow f from s to t is **equal** to the cost of this cut.
 - Similar to proof-by-picture we saw before:
 - All of the stuff has to **cross the cut**.
 - The edges in the cut are **full** because they don't exist in G_f



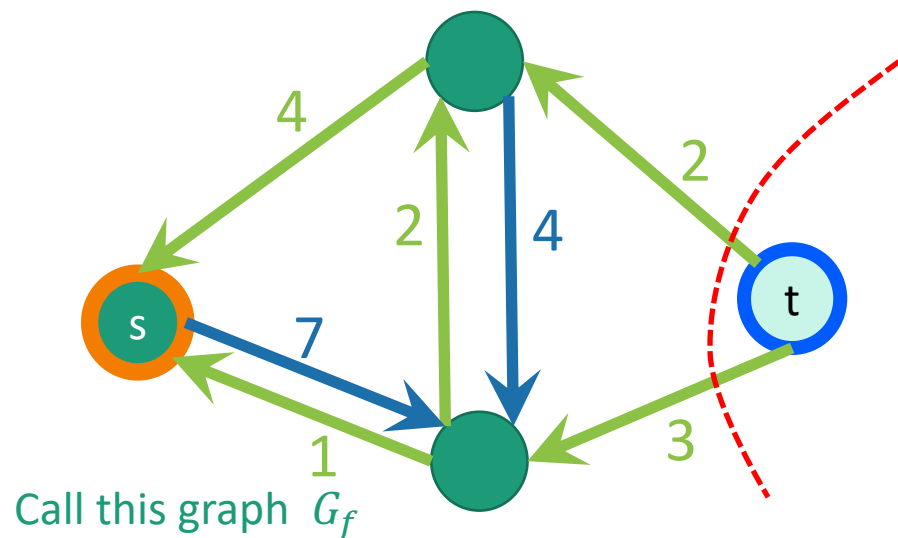
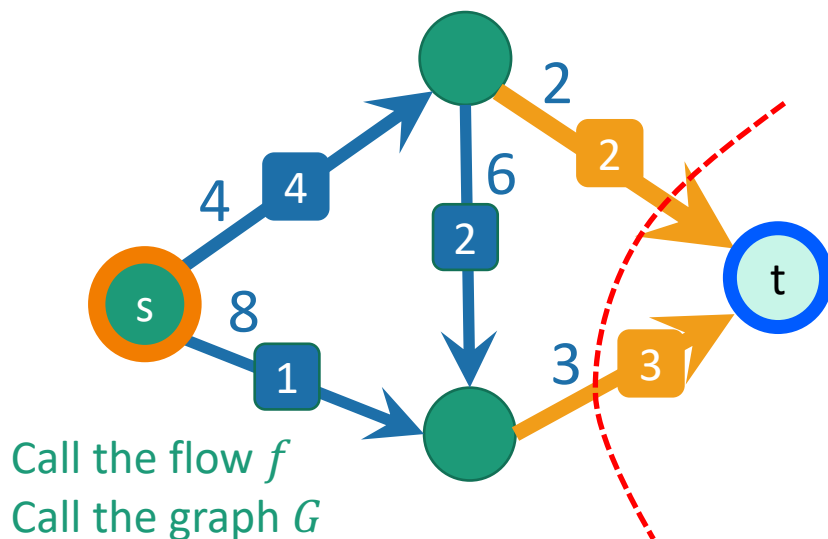
No augmenting path \Rightarrow max flow.

- Suppose there is not a path from s to t in G_f .
- Consider the cut given by:

{things reachable from s }, **{things not reachable from s }** t lives here

- The value of the flow f from s to t is **equal** to the cost of this cut.

Value of f = cost of this cut \geq min cut \geq max flow Lemma 1



No augmenting path \Rightarrow max flow. ✓

- Suppose there is not a path from s to t in G_f .
- Consider the cut given by:

{things reachable from s }, {things not reachable from s }

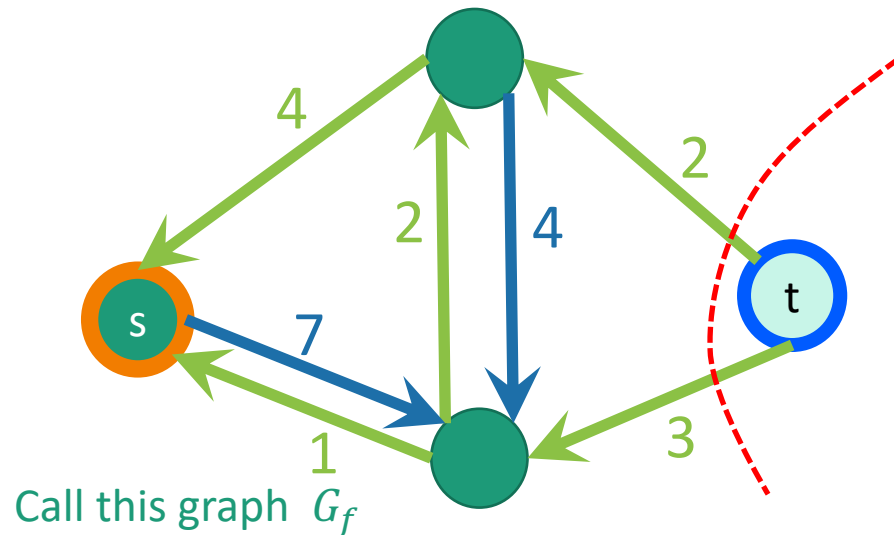
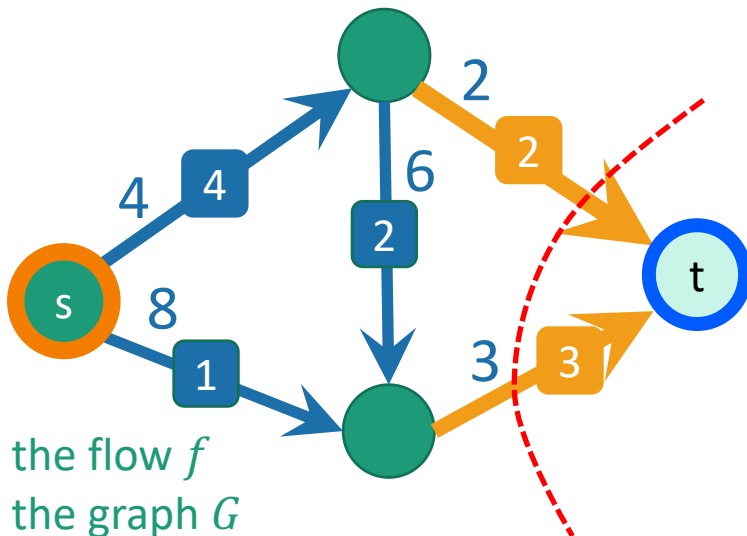
t lives here

- The value of the flow f from s to t is **equal** to the cost of this cut.

Value of f = cost of this cut \geq min cut \geq max flow

Lemma 1

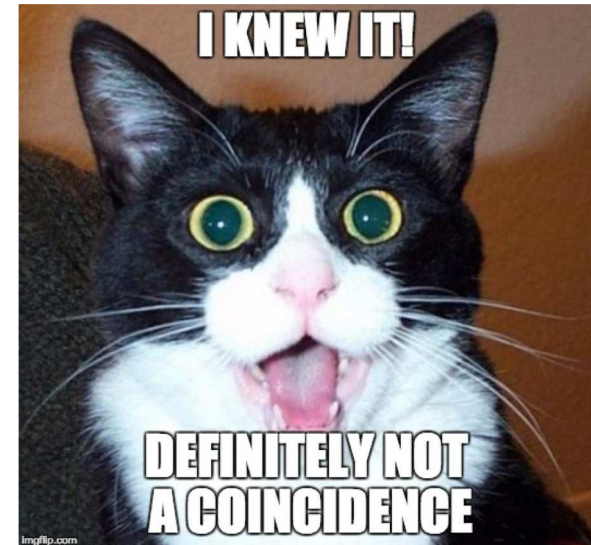
- Therefore f is a max flow!
- Thus, when Ford-Fulkerson stops, it's found the maximum flow.



Min-Cut Max-Flow Theorem

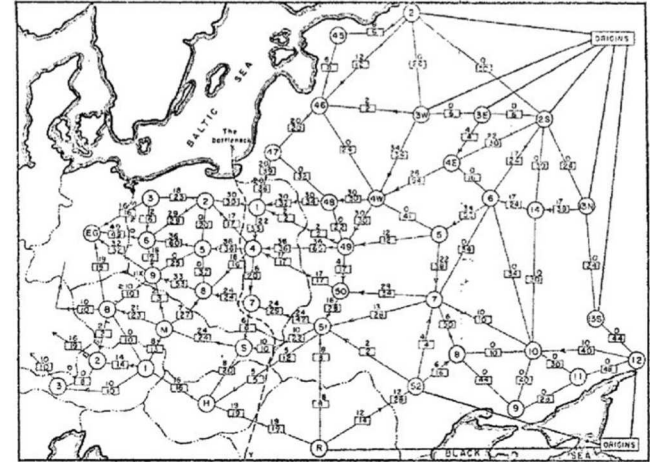
$\text{max flow} \geq \text{Value of } f = \text{cost of this cut} \geq \text{min cut} \geq \text{max flow}$

So everything is equal and min cut = max flow!



What have we learned?

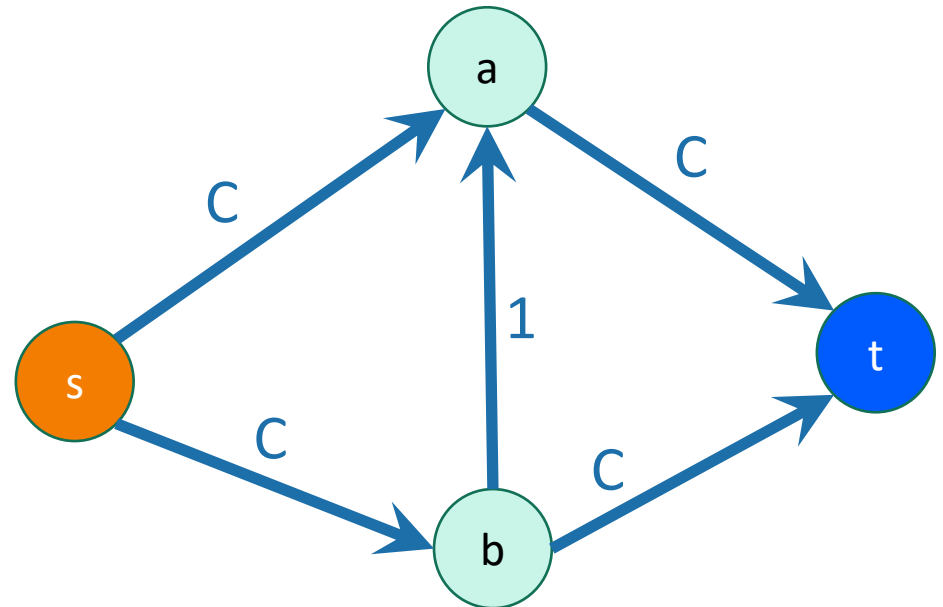
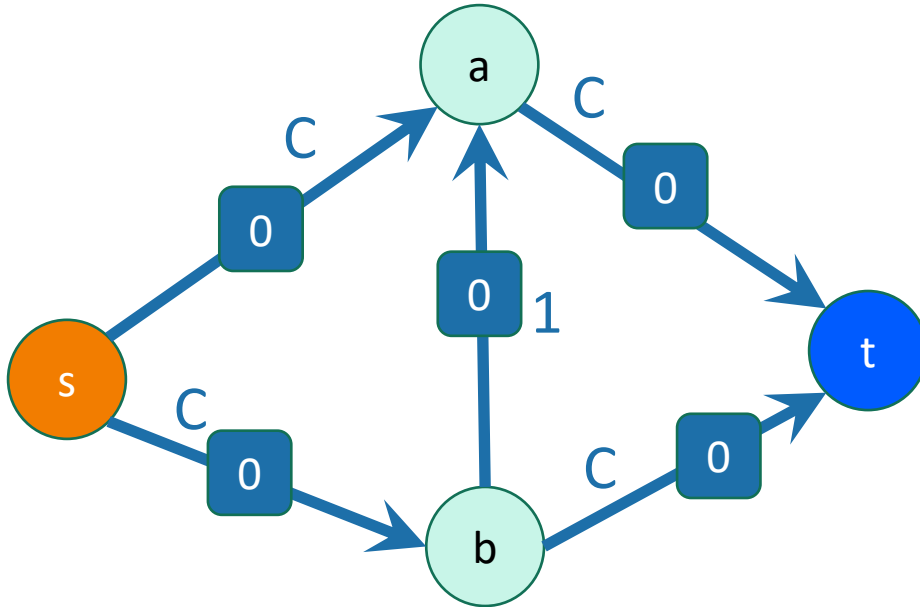
- Max s-t flow is equal to min s-t cut!
 - The USSR and the USA were trying to solve the same problem...
- The Ford-Fulkerson algorithm can find the min-cut/max-flow.
 - Repeatedly improve your flow along an augmenting path.
- **How long does this take???**



Why should we be concerned?

Suppose we just picked paths arbitrarily.

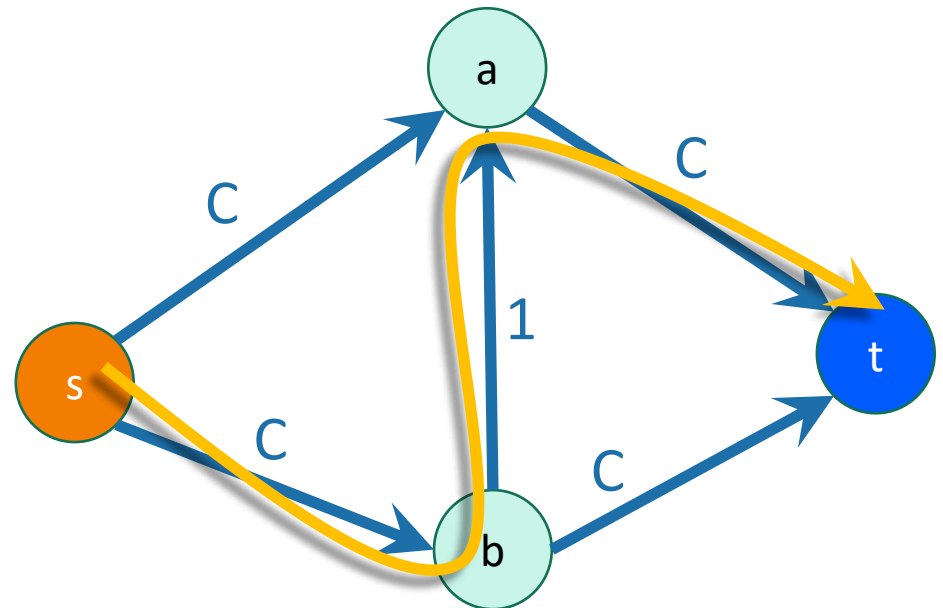
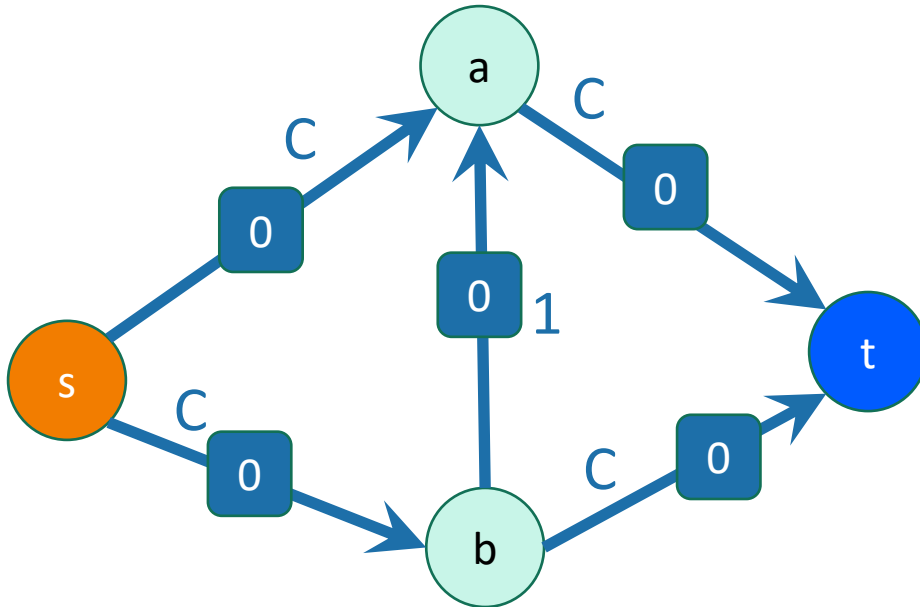
Choose a really big number C .



Why should we be concerned?

Suppose we just picked paths arbitrarily.

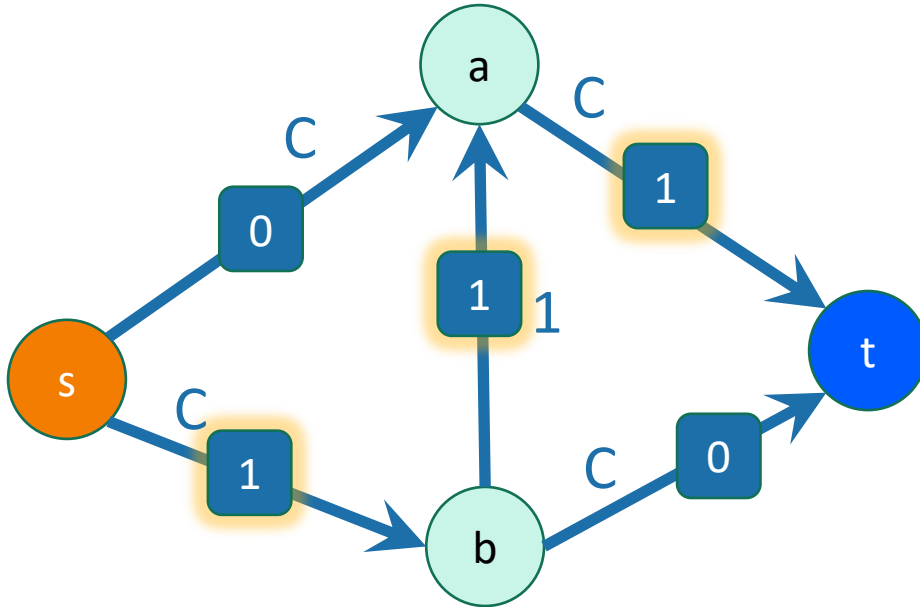
Choose a really big number C .



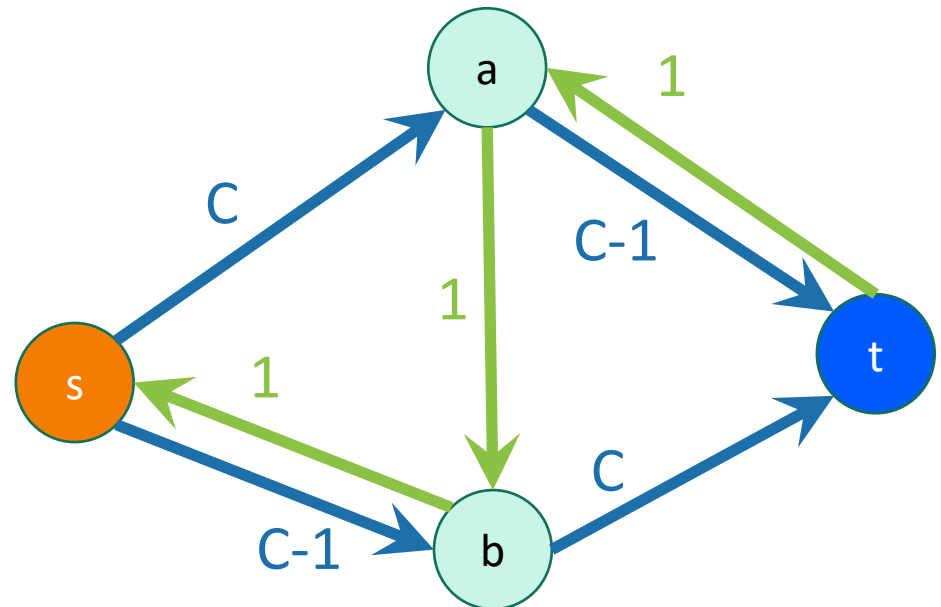
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C .



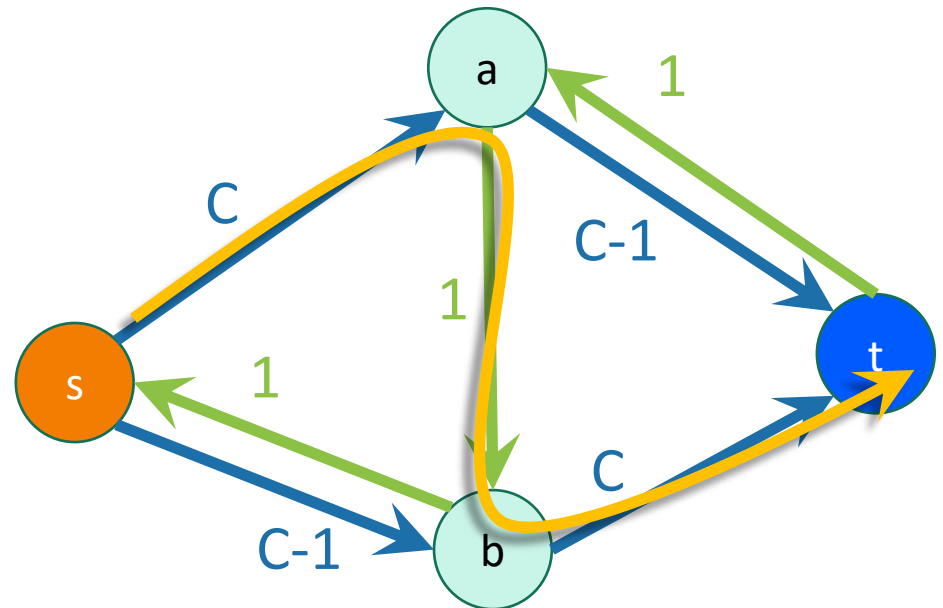
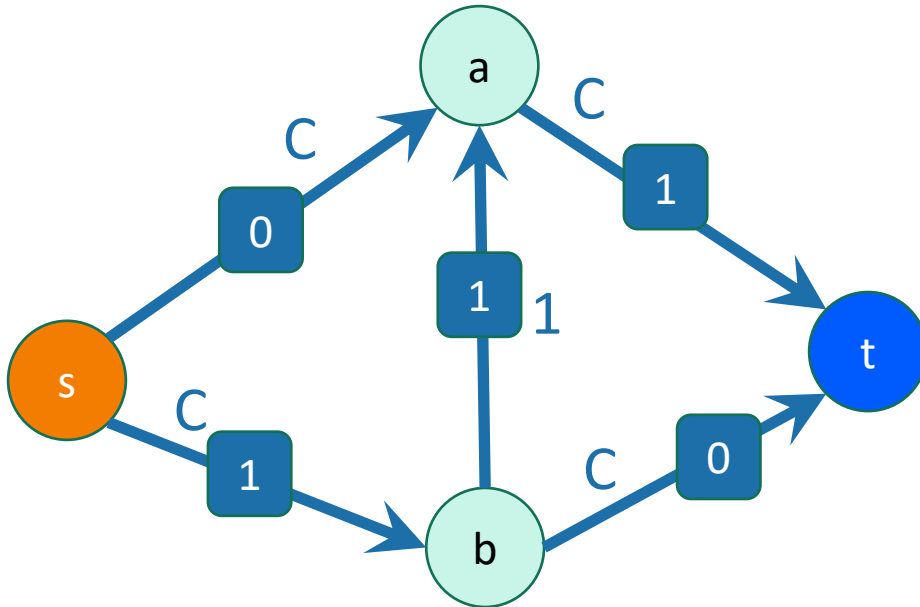
The edge (b,a) disappeared from the residual graph!



Why should we be concerned?

Suppose we just picked paths arbitrarily.

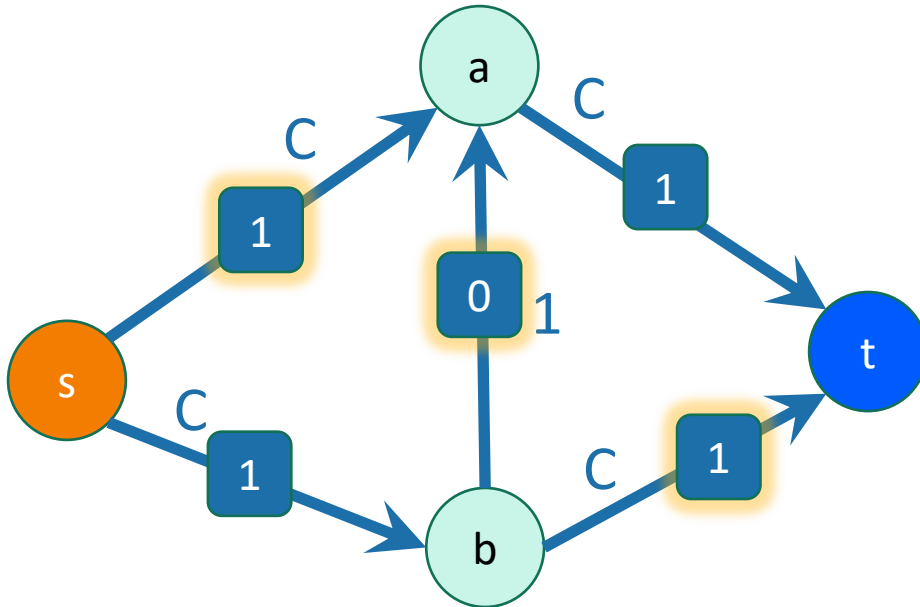
Choose a really big number C .



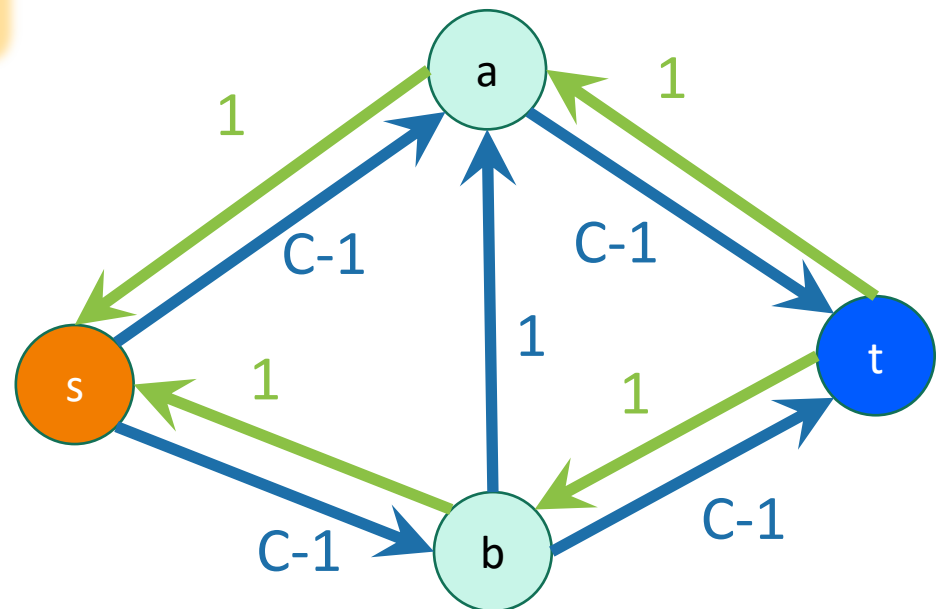
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C .



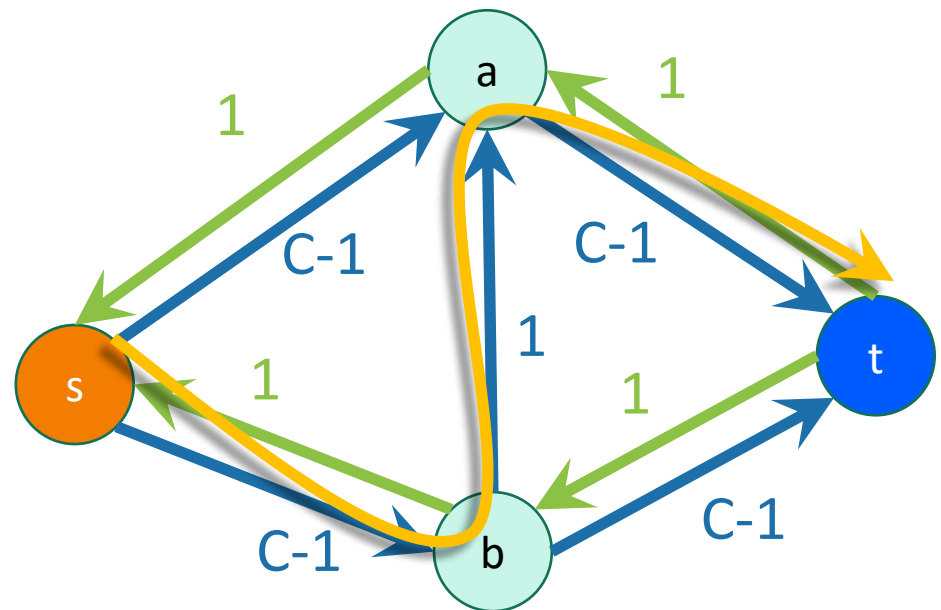
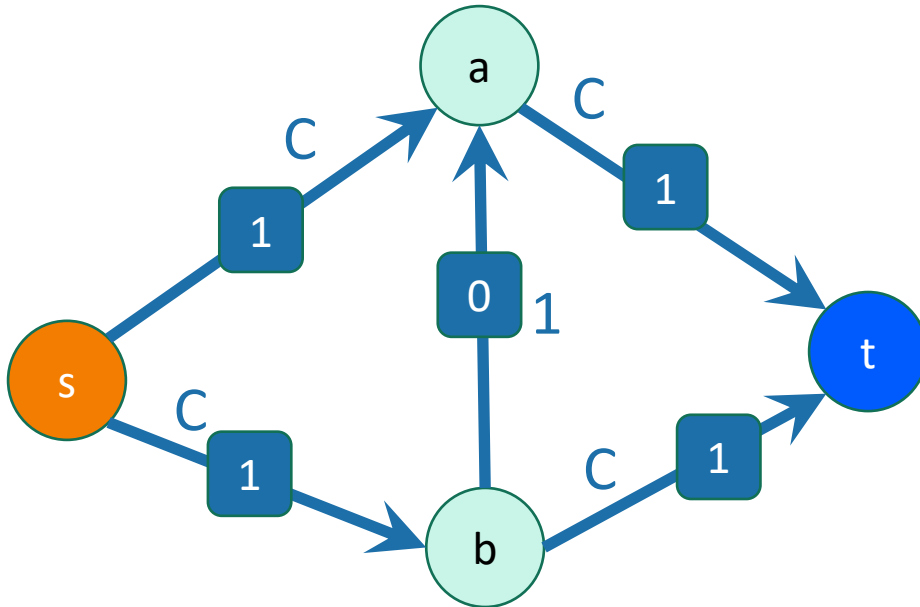
The edge (b,a) re-appeared in the residual graph!



Why should we be concerned?

Suppose we just picked paths arbitrarily.

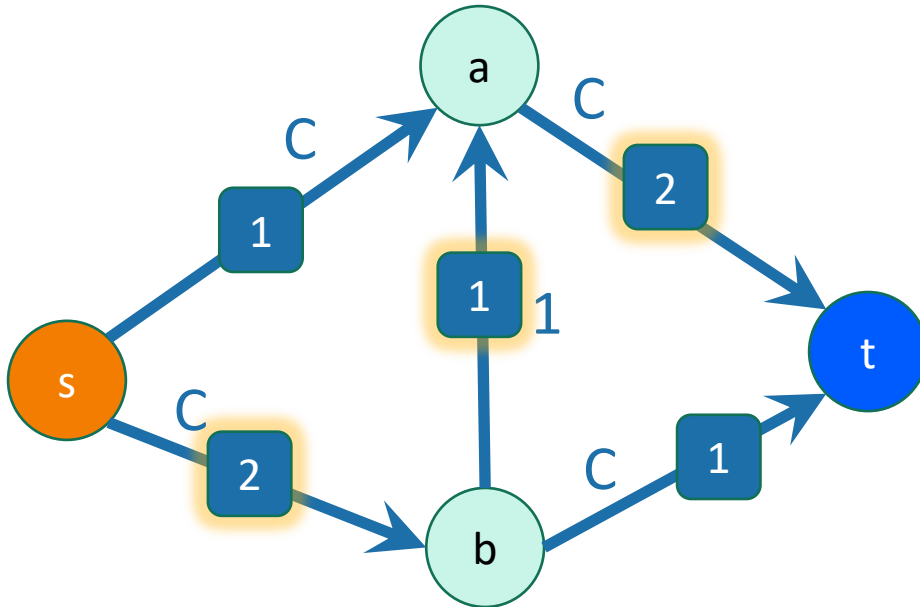
Choose a really big number C .



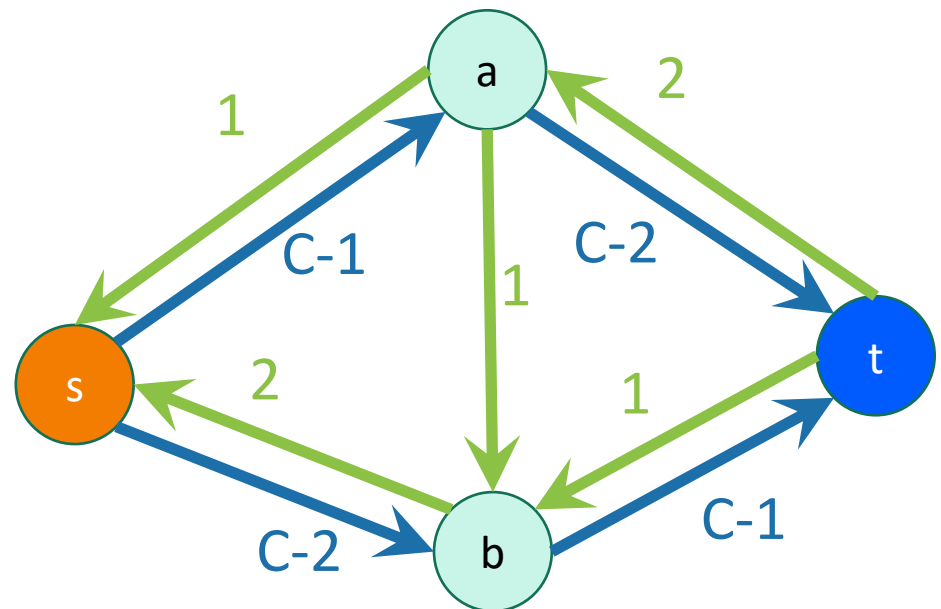
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C .



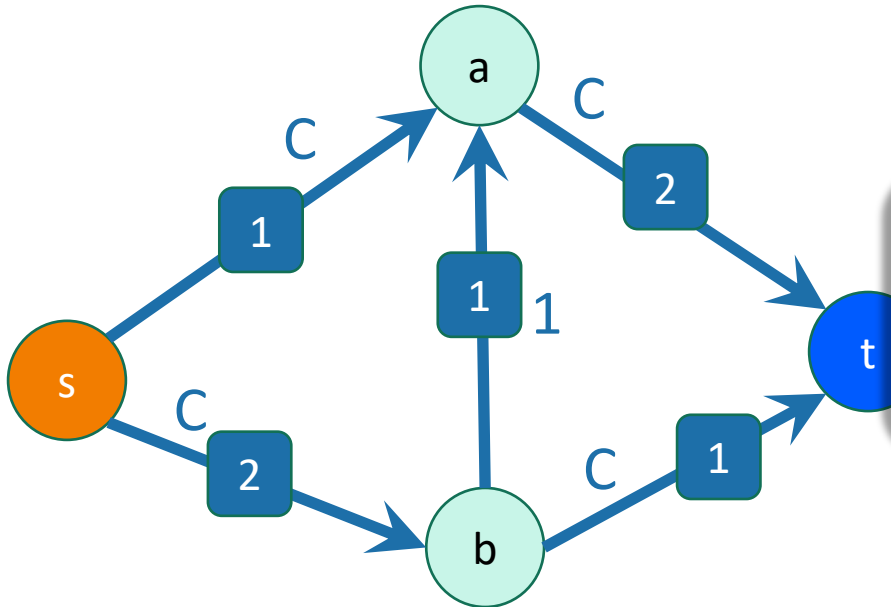
The edge (b, a) disappeared from the residual graph!



Why should we be concerned?

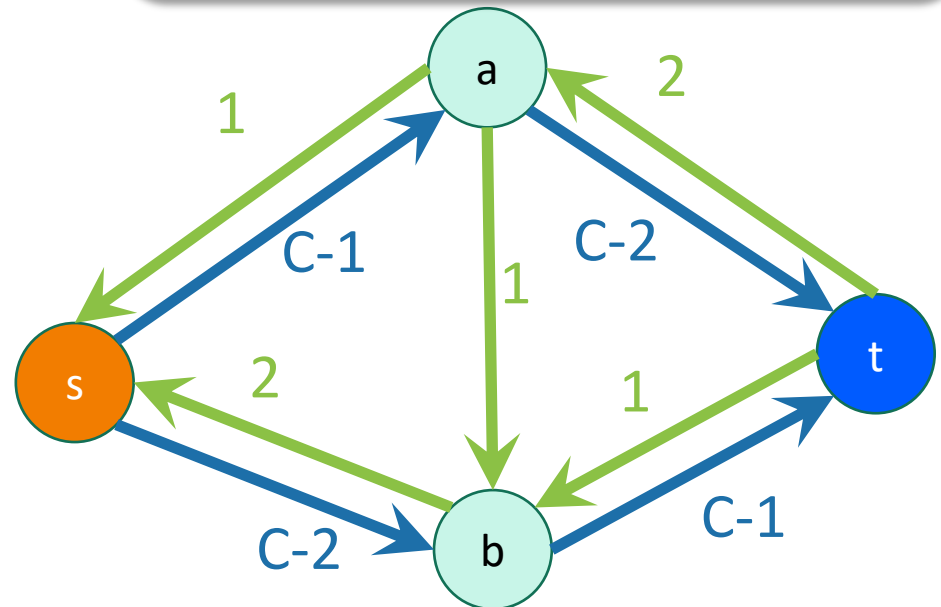
Suppose we just picked paths arbitrarily.

Choose a really big number C .



This will go on for C steps, adding flow along (b,a) and then subtracting it again.

The edge (b,a) disappeared from the residual graph!



How do we choose which paths to use?

- The analysis we did still works no matter how we choose the paths.
 - That is, the algorithm will be **correct** if it terminates.
- **However, the algorithm may not be efficient!!!**
 - May take a long time to terminate
 - (Or may actually never terminate?)
- We need to be careful with our path selection to make sure the algorithm terminates quickly.
 - Using BFS leads to the **Edmonds-Karp algorithm**.
 - It turns out this will work in time $O(nm^2)$ – proof skipped.
 - (That's not the only way to do it!)

One more useful observation

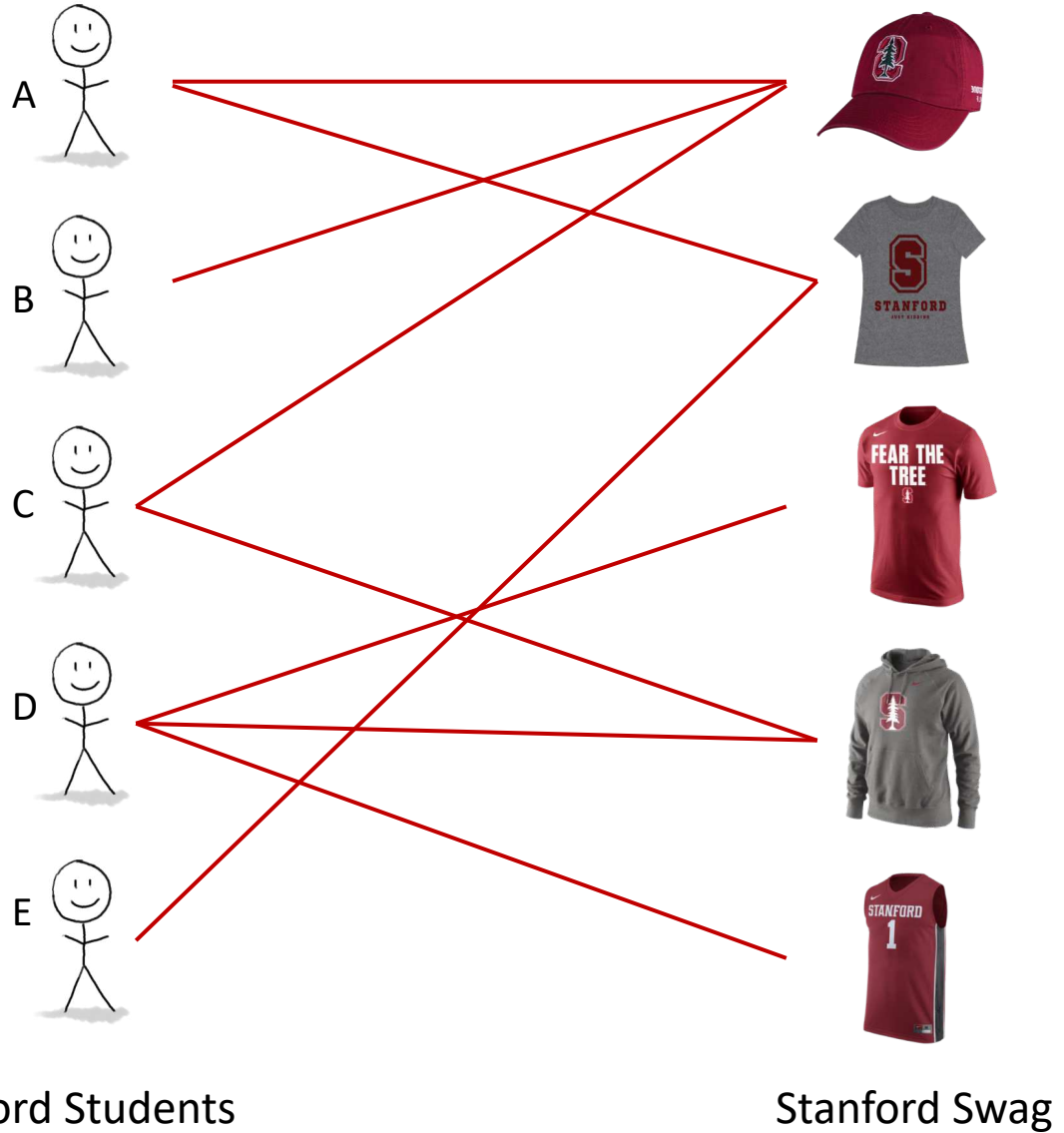
- If all the capacities are integers, then the flows in any max flow are also all integers.
 - When we update flows in Ford-Fulkerson, we're only ever adding or subtracting integers.
 - Since we started with 0 (an integer), everything stays an integer.

But wait, there's more!

- Min-cut and max-flow are not just useful for the USA and the USSR in 1955.
- The Ford-Fulkerson algorithm is the basis for many other graph algorithms.
- For the rest of today, we'll see a few:
 - Maximum bipartite matching
 - Integer assignment problems

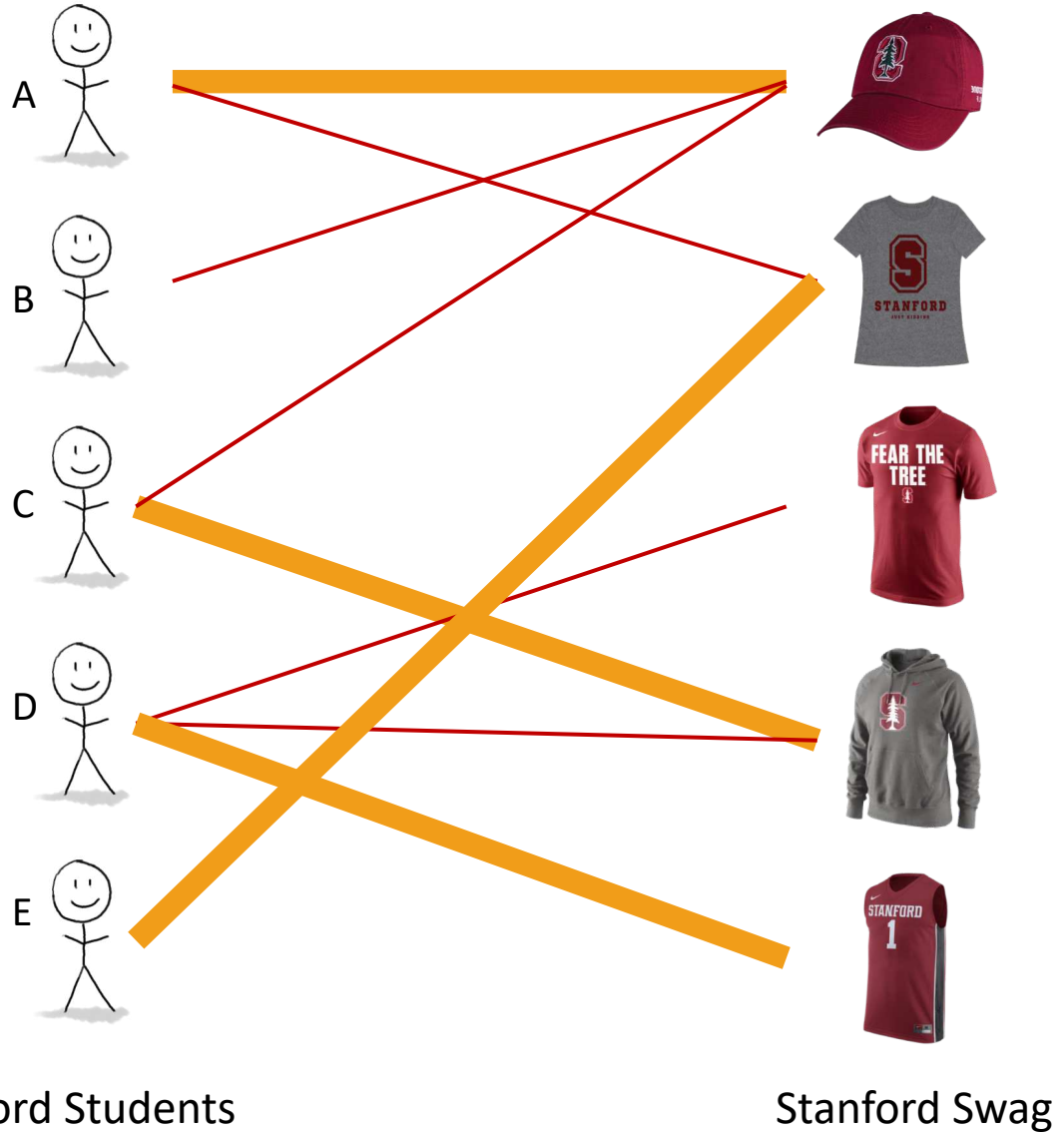
Maximum matching in bipartite graphs

- Different students only want certain items of Stanford swag (depending on fit, style, etc).
- How can we make as many students as possible happy?



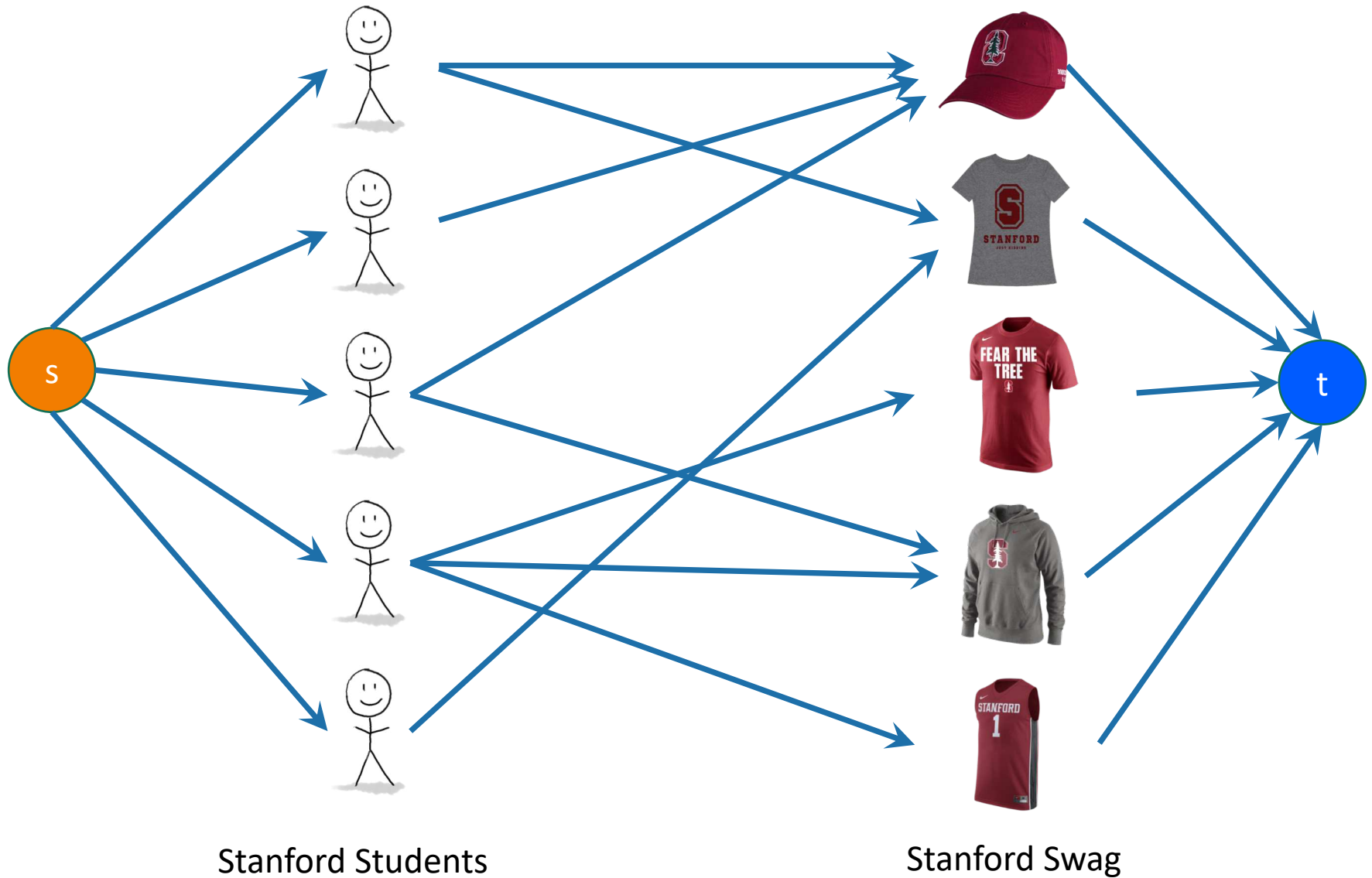
Maximum matching in bipartite graphs

- Different students only want certain items of Stanford swag (depending on fit, style, etc).
- How can we make as many students as possible happy?



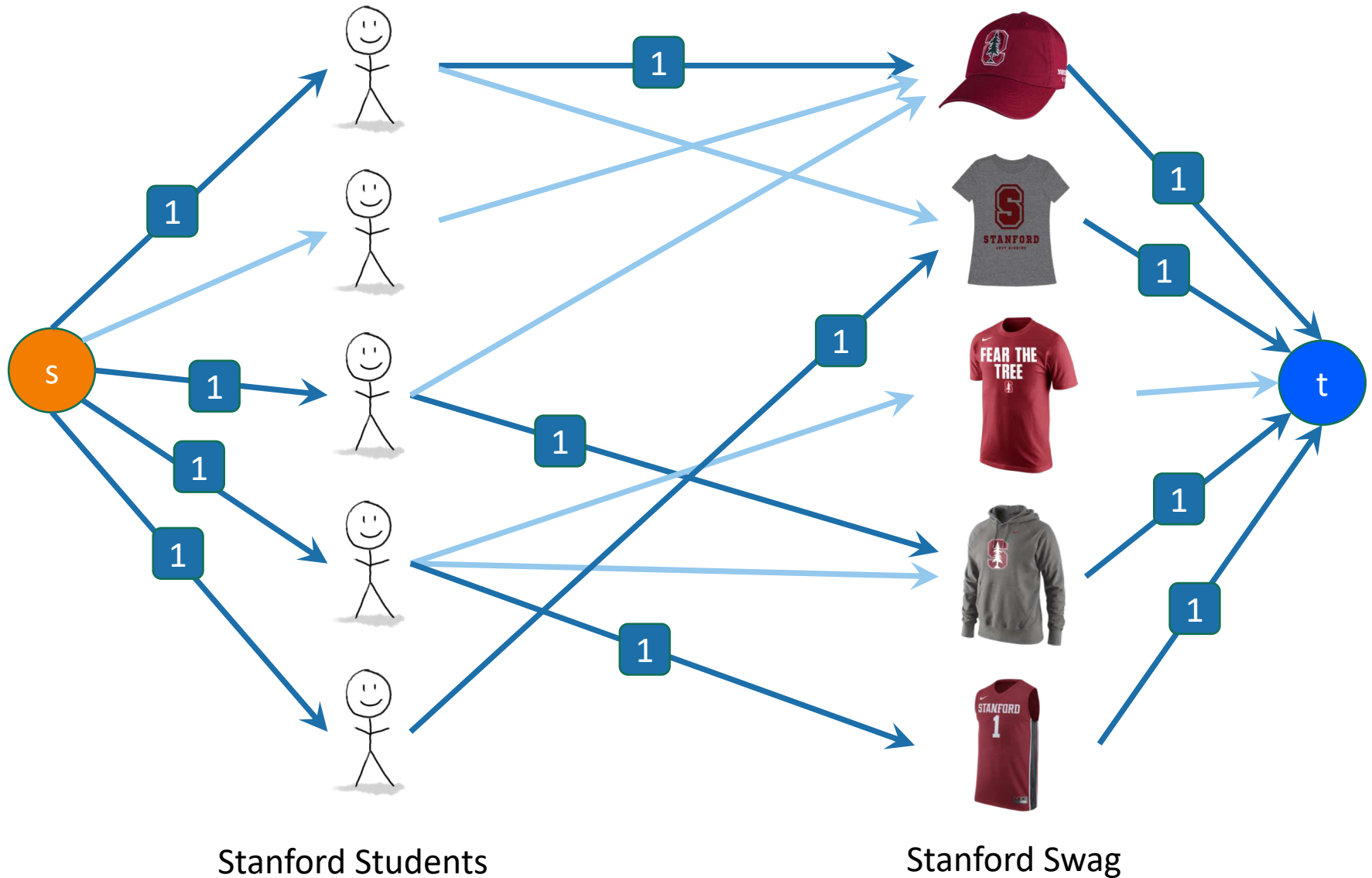
Solution via max flow

All edges have capacity 1.



Solution via max flow

All edges have capacity 1.

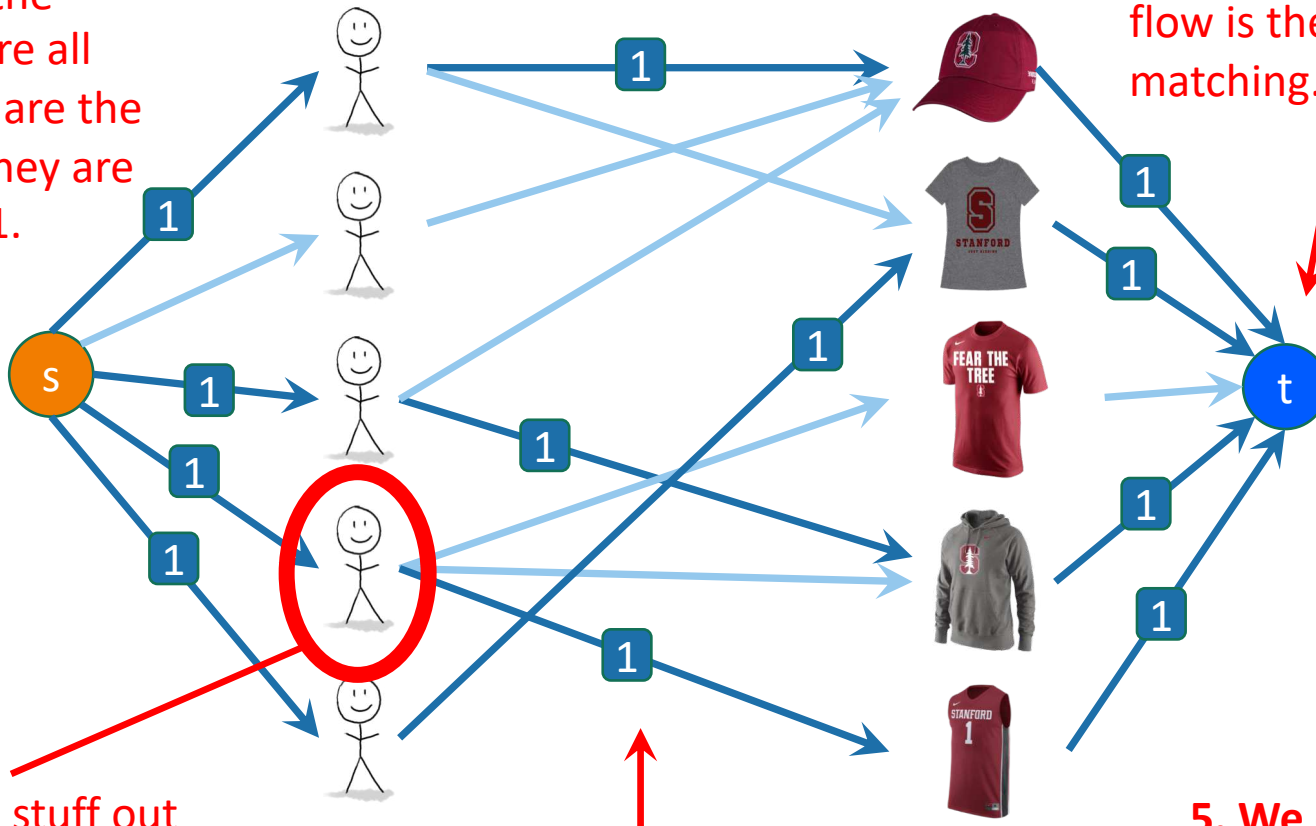


Solution via max flow

why does this work?

All edges have capacity 1.

1. Because the capacities are all integers, so are the flows – so they are either 0 or 1.



4. The value of the flow is the size of the matching.

Value of this flow is 4.

2. Stuff in = stuff out means that the number of items assigned to each student 0 or 1. (And vice versa).

3. Thus, the edges with flow on them form a matching. (And, any matching gives a flow).

5. We conclude that the max flow corresponds to a max matching.

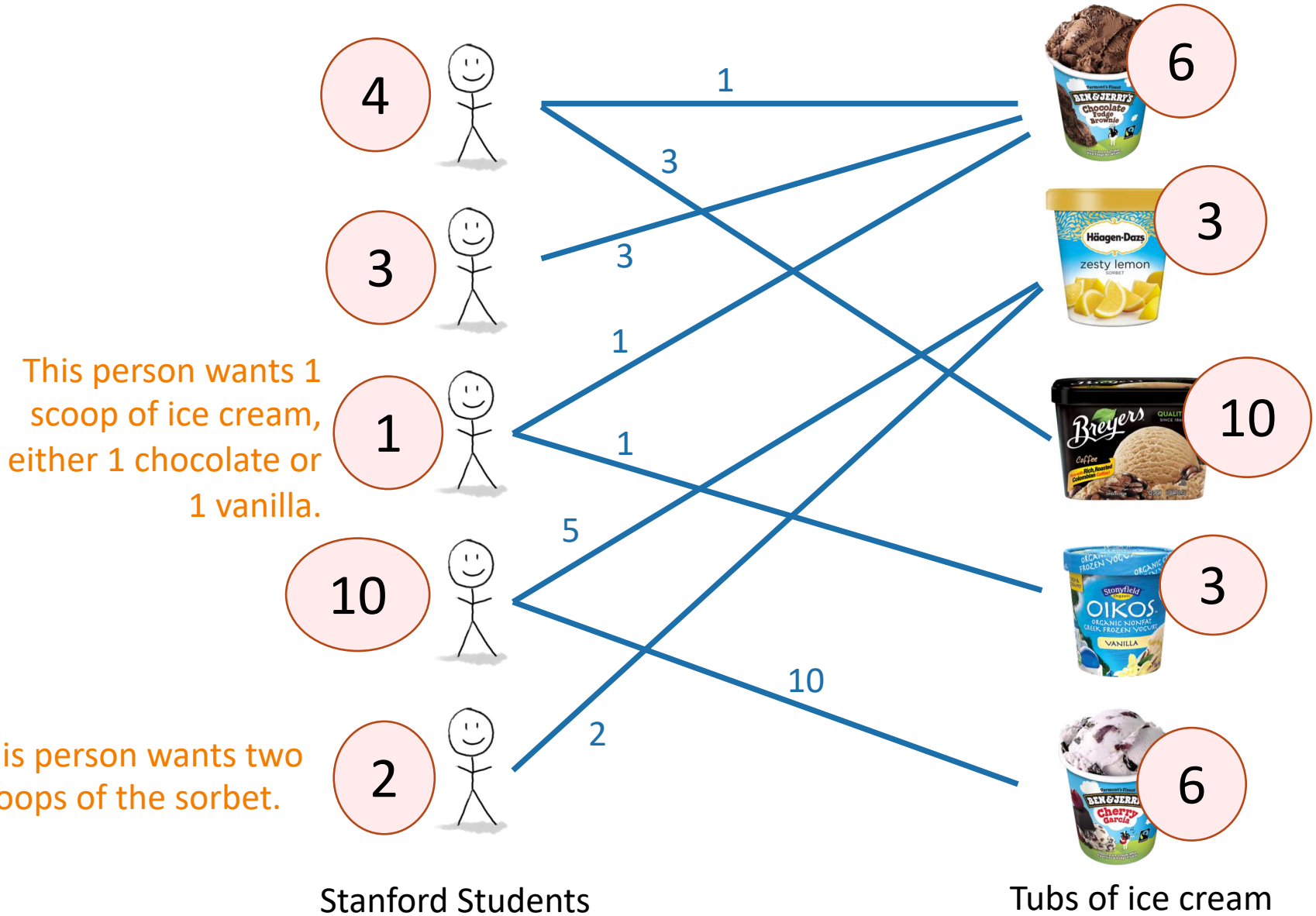
A slightly more complicated example: assignment problems

- One set X
 - Example: Stanford students
- Another set Y
 - Example: tubs of ice cream
- Each x in X can participate in $c(x)$ matches.
 - Student x can only eat 4 scoops of ice cream.
- Each y in Y can only participate in $c(y)$ matches.
 - Tub of ice cream y only has 10 scoops in it.
- Each pair (x,y) can only be matched $c(x,y)$ times.
 - Student x only wants 3 scoops of flavor y
 - Student x' doesn't want any scoops of flavor y'
- **Goal: assign as many matches as possible.**

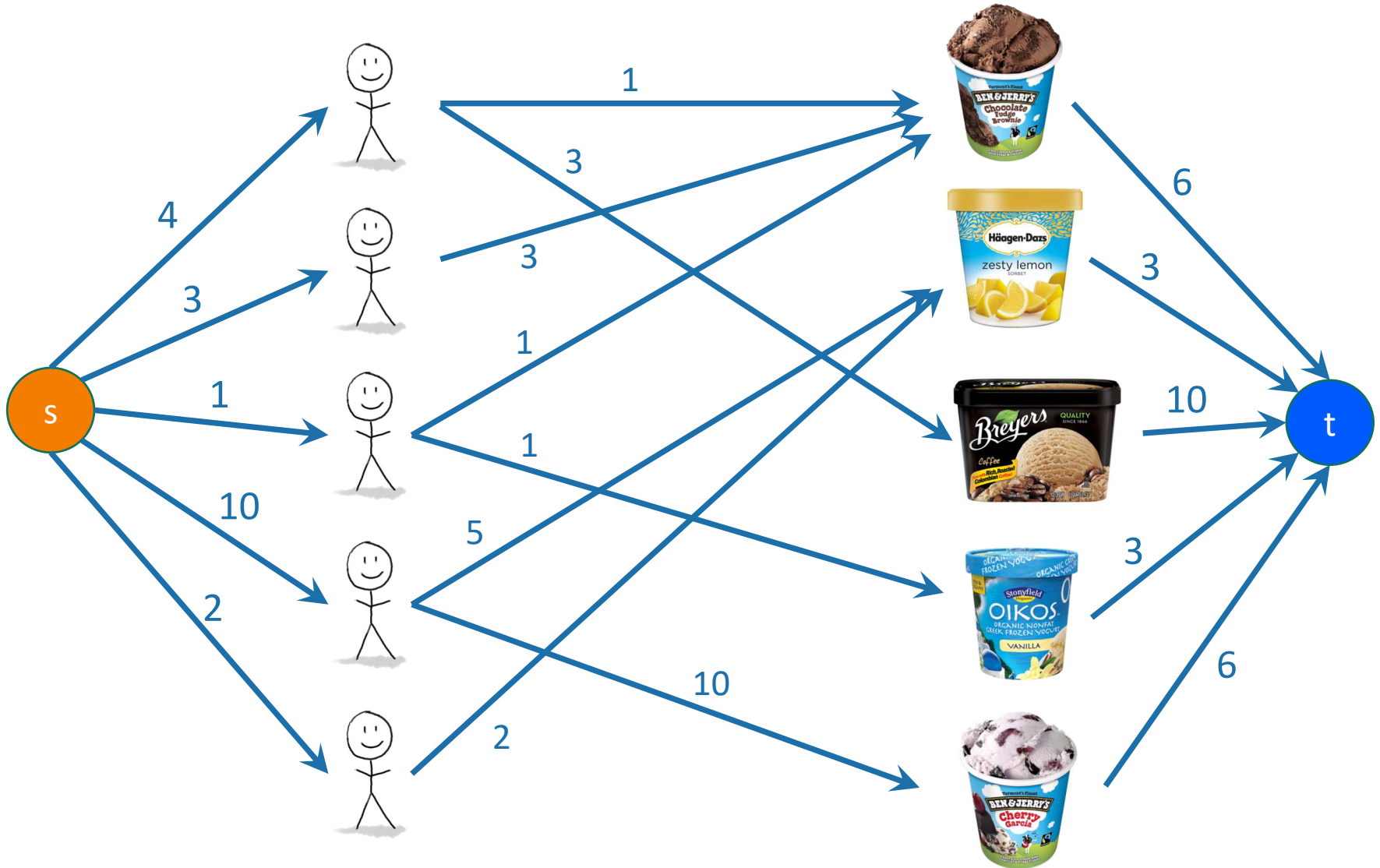


How can we serve as much ice cream as possible?

Example



Solution via max flow

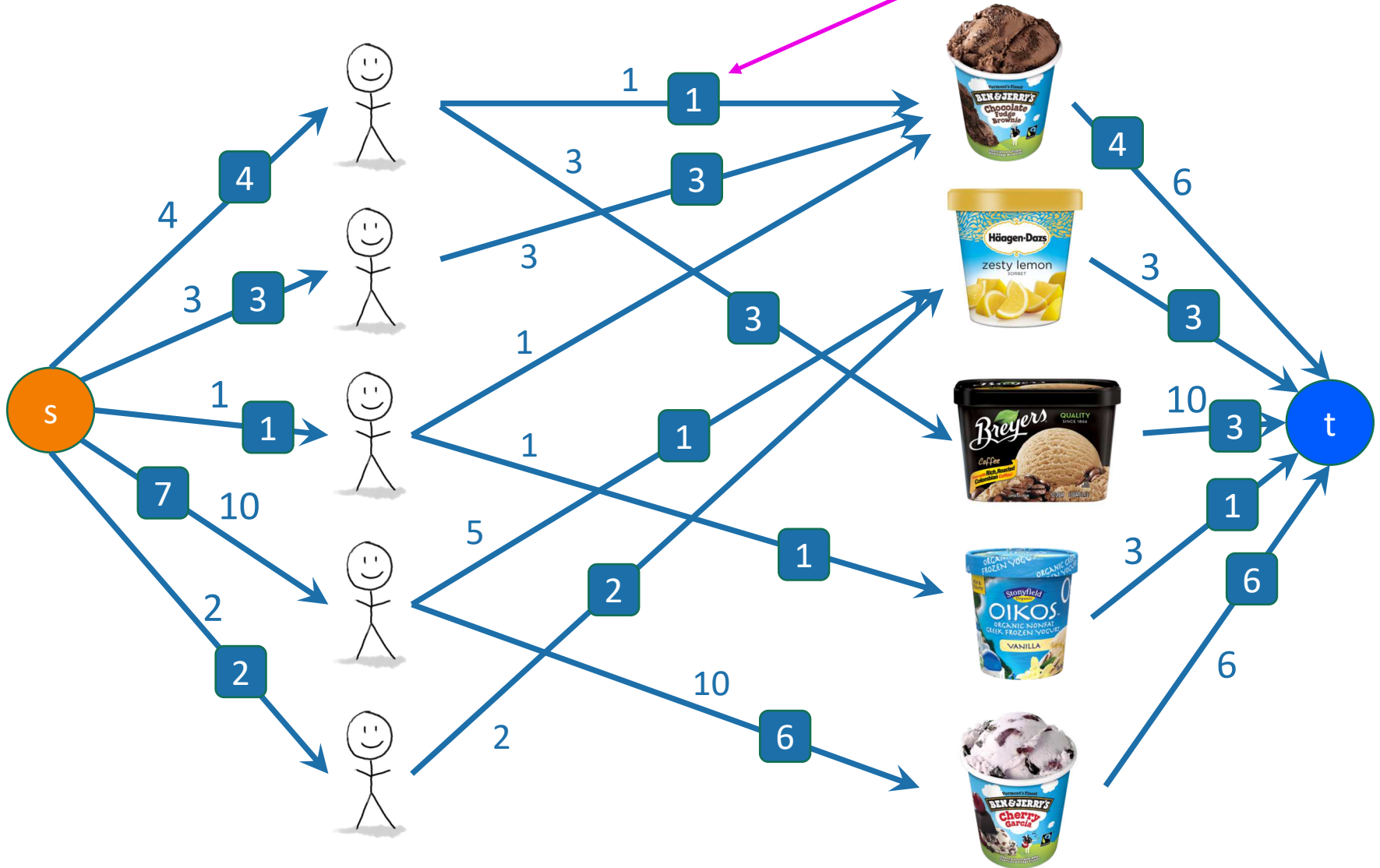


Stanford Students

Tubs of ice cream

Solution via max flow

Give this person **1** scoop of this ice cream.

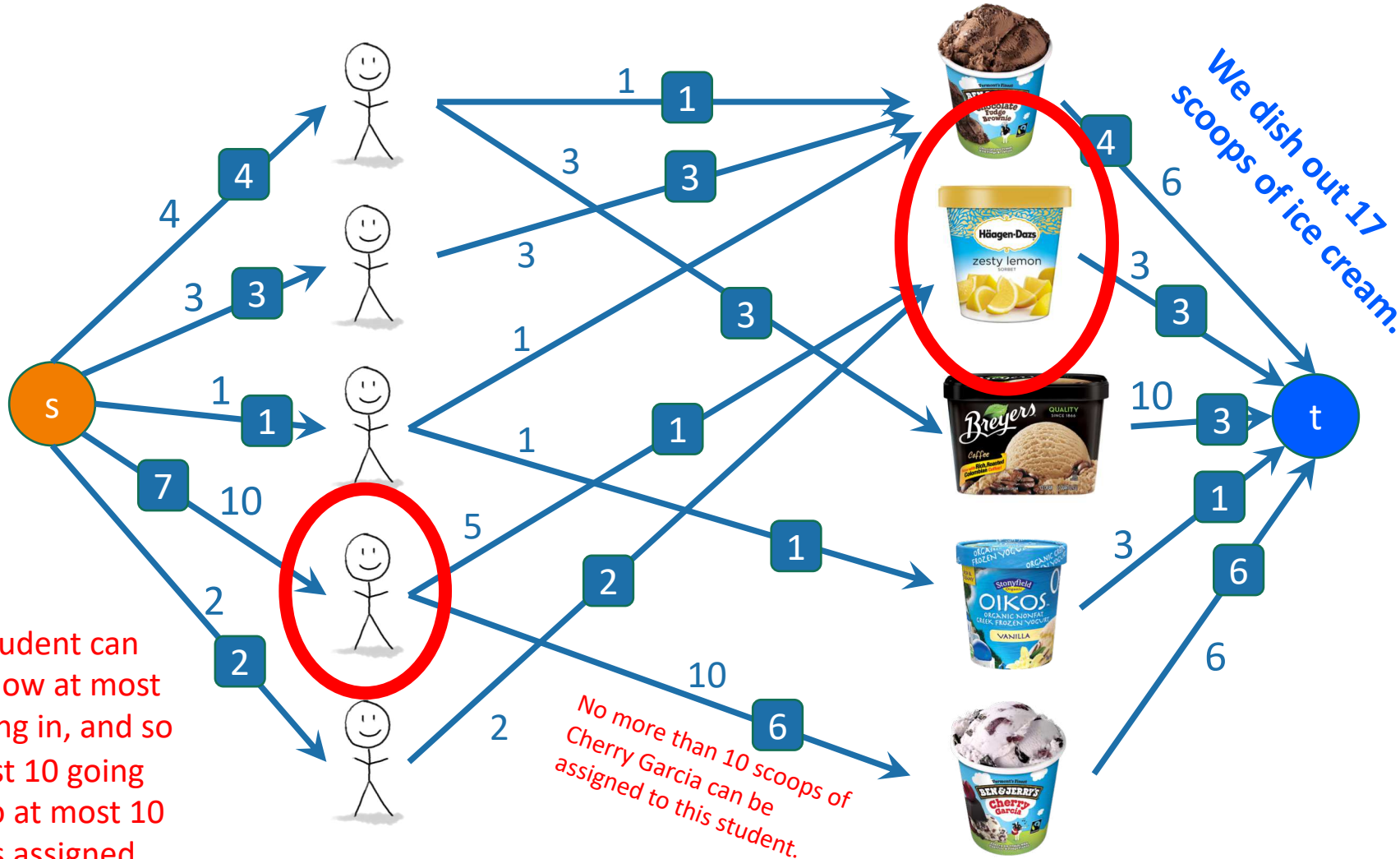


Stanford Students

Tubs of ice cream

Solution via max flow

No more than 3 scoops of sorbet can be assigned.



This student can have flow at most 10 going in, and so at most 10 going out, so at most 10 scoops assigned.

No more than 10 scoops of Cherry Garcia can be assigned to this student.

We dish out 17 scoops of ice cream.

As before, flows correspond to assignments, and max flows correspond to max assignments.

What have we learned?

- Max flows and min cuts aren't just for railway routing.
 - Immediately, they apply to other sorts of routing too!
 - But also they are useful for assigning items to Stanford students!

Recap

- Today we talked about s-t cuts and s-t flows.
- The **Min-Cut Max-Flow Theorem** says that minimizing the cost of cuts is the same as maximizing the value of flows.
- The **Ford-Fulkerson algorithm** does this!
 - Find an augmenting path
 - Increase the flow along that path
 - Repeat until you can't find any more paths and then you're done!
- An important algorithmic primitive!
 - eg, assignment problems.

Next time

- More recap
 - A look back at the class
- More cool stuff in algorithms!
 - A look forward at future classes

