Lecture 17

max flows, min cuts, and Ford-Fulkerson

Announcements

- HW 7 due Friday
- **FINAL EXAM:**
  - Wednesday December 13
  - 3:30– 6:30
  - Room 320-105
- Final exam covers up through today.
  - But material post-HW7 will be covered more lightly
- Monday 12/4 in class:
  - What's next?
- Wednesday 12/6 in class:
  - Review session

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.

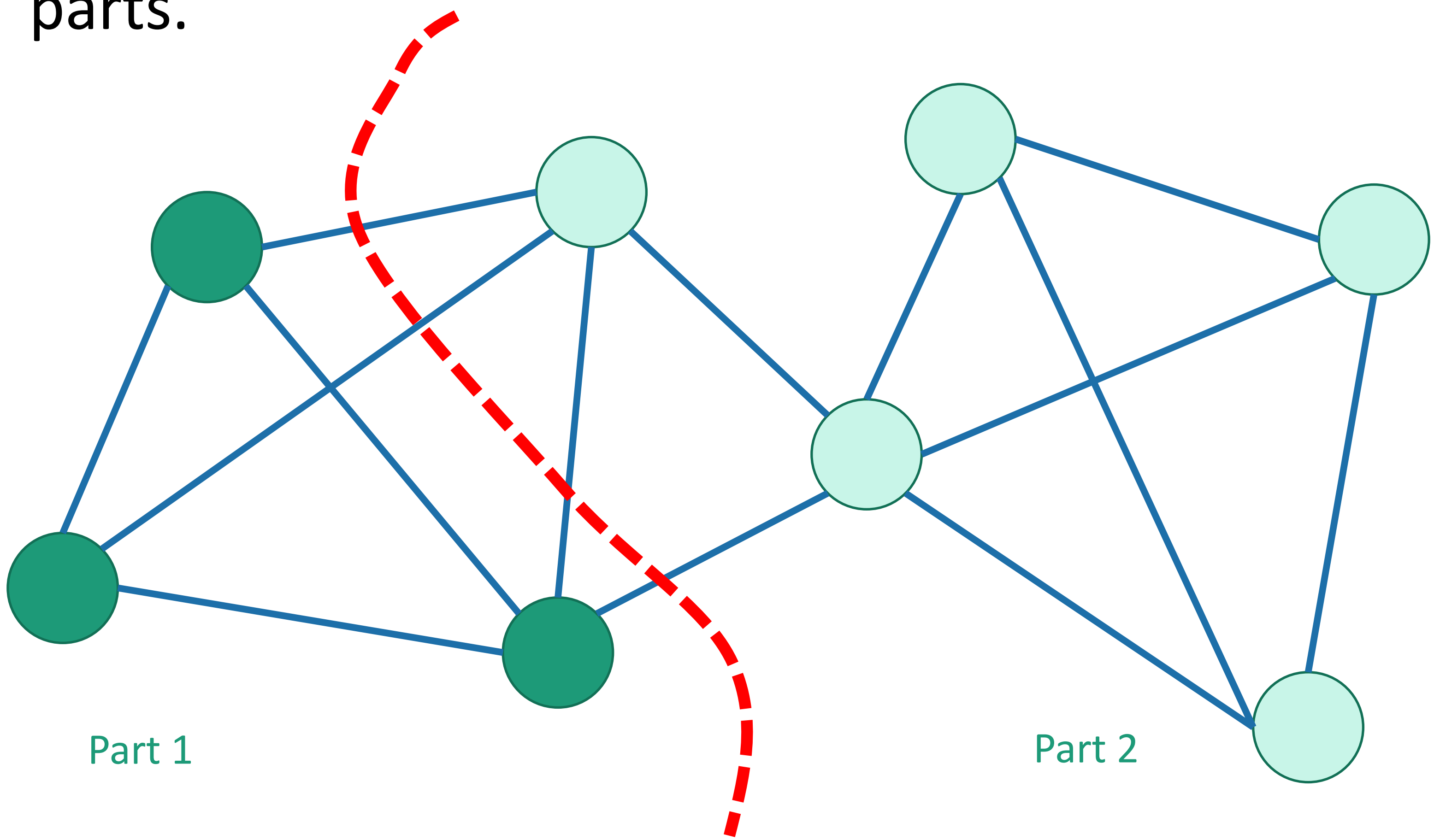
(However, the Ford-Fulkerson algorithm is fair game for the final).

Lucky the lackadaisical lemur

Last time

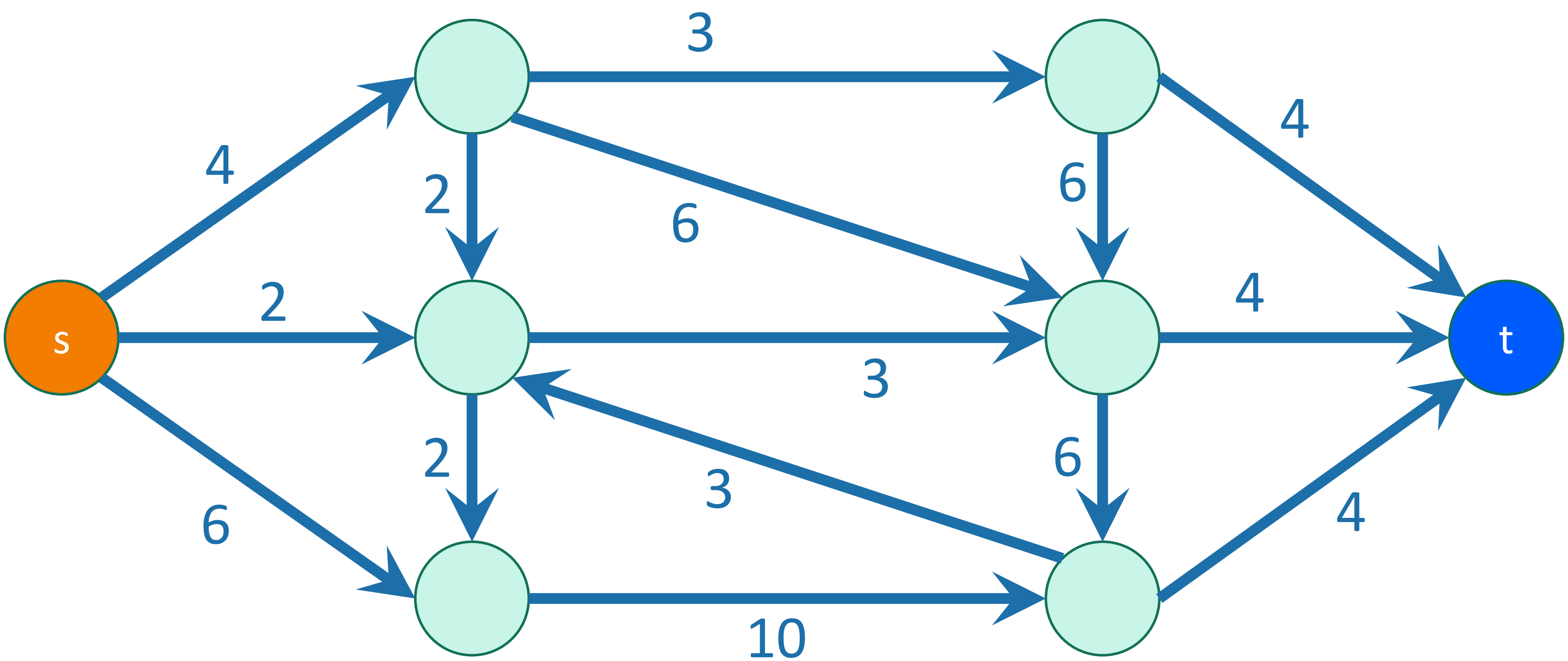
Last time graphs were undirected and unweighted.

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



Today

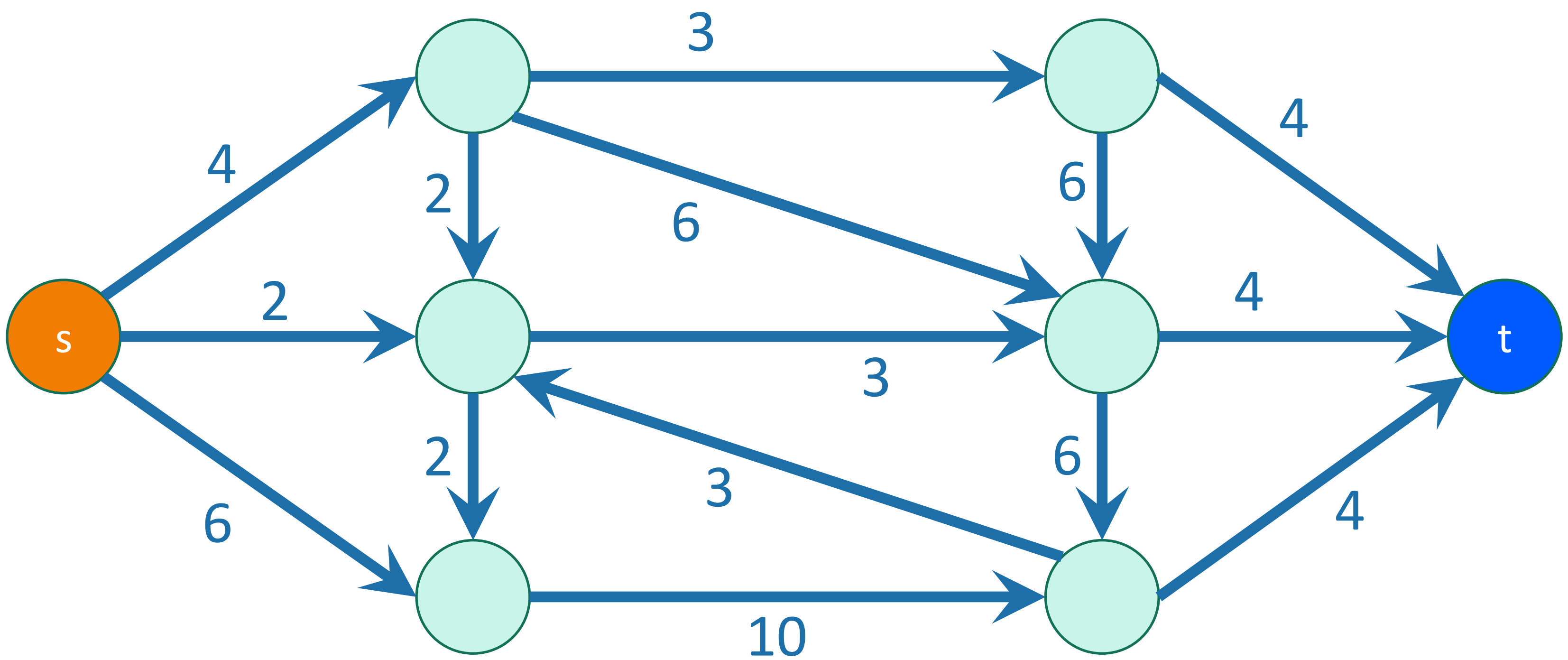
- Graphs are directed and edges have "capacities" (weights)
- We have a special "source" vertex s and "sink" vertex t.
  - s has only outgoing edges*
  - t has only incoming edges*



*at least for this class

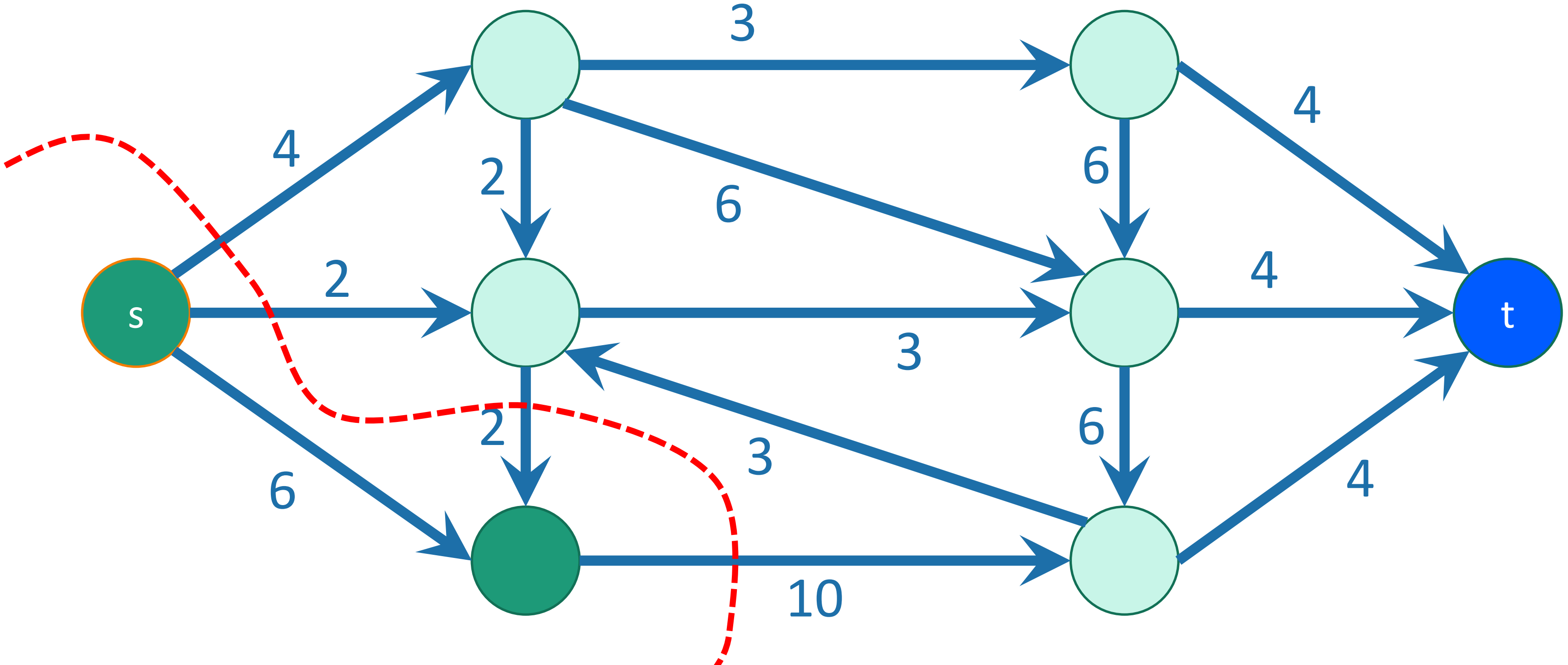
An **s-t cut**

is a cut which separates s from t



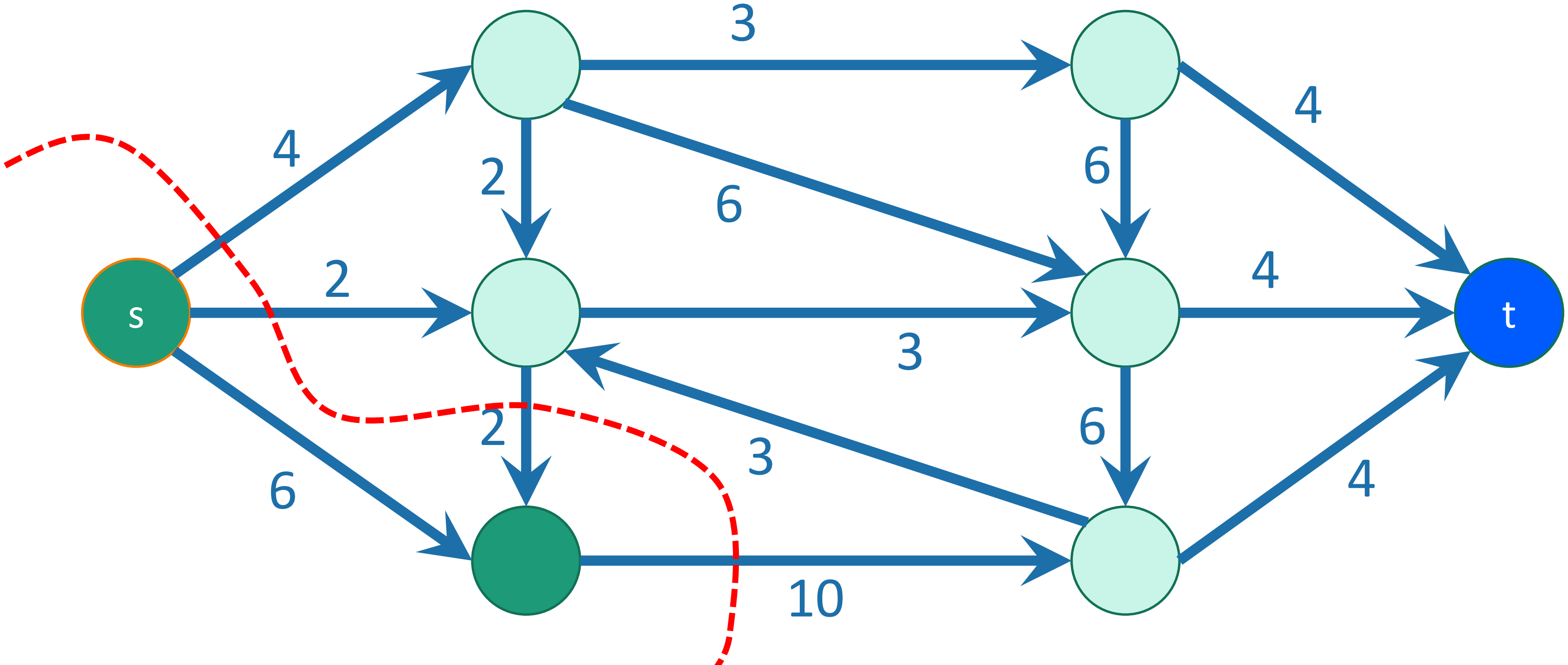
An **s-t cut**

is a cut which separates s from t



An **s-t cut**
is a cut which separates s from t

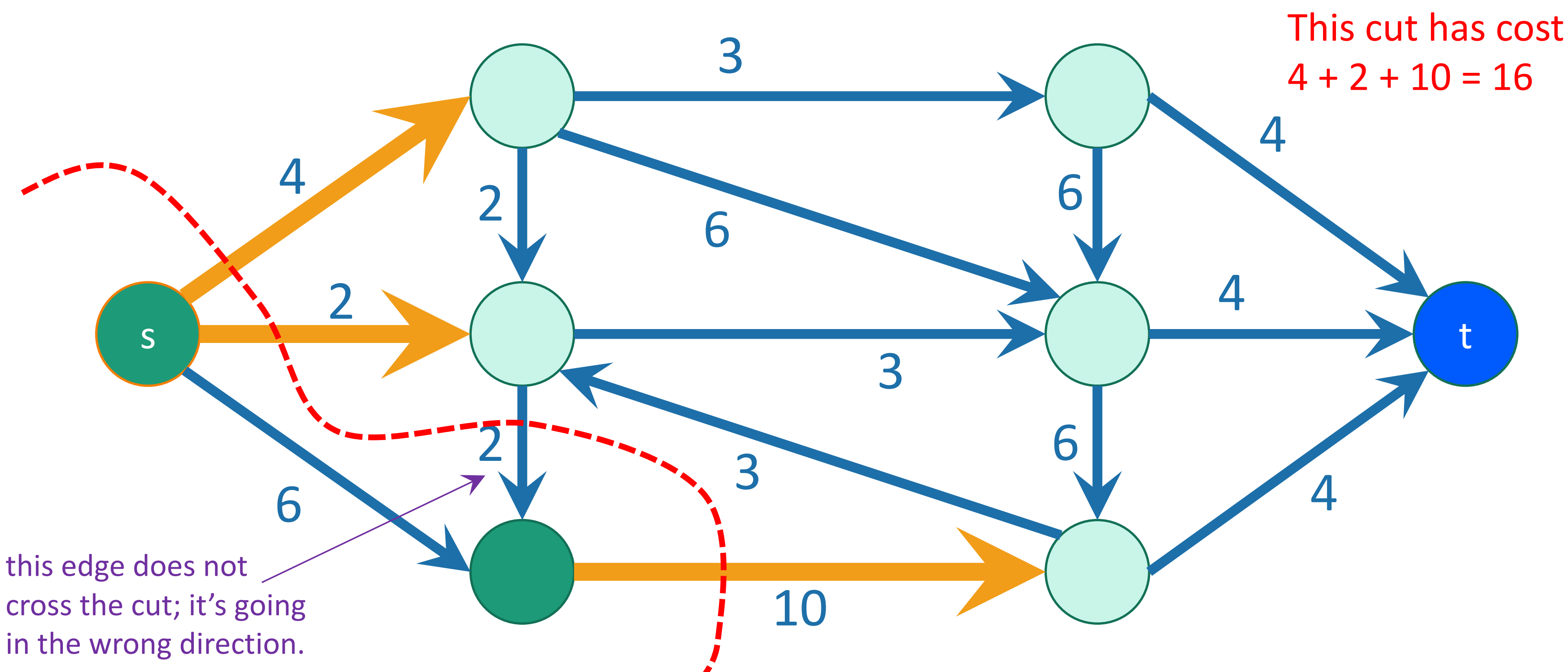
- An edge **crosses the cut** if it goes from s's side to t's side.



An **s-t cut**

is a cut which separates s from t

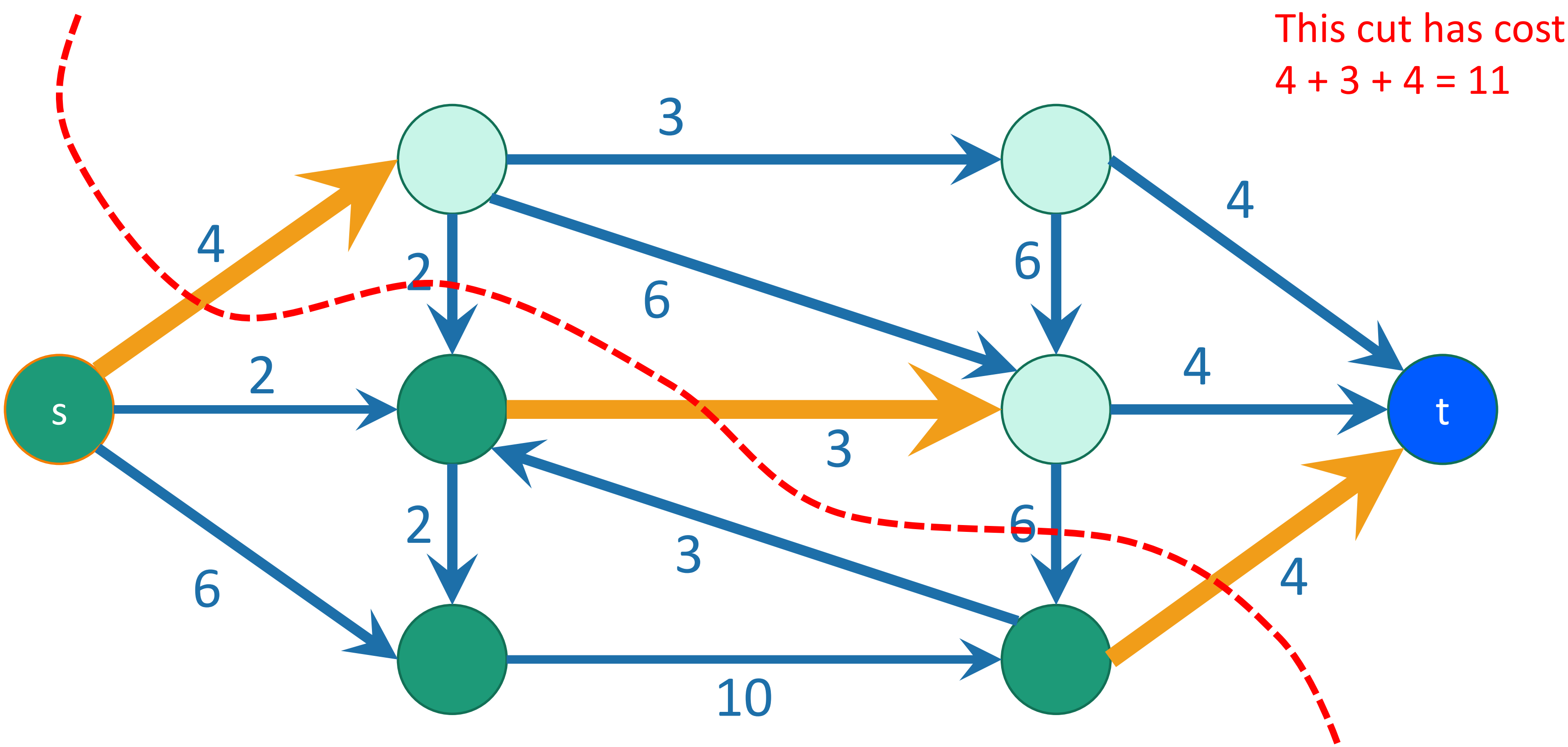
- An edge **crosses the cut** if it goes from s's side to t's side.
- The **cost** (or capacity) of a cut is the sum of the capacities of the edges that cross the cut.



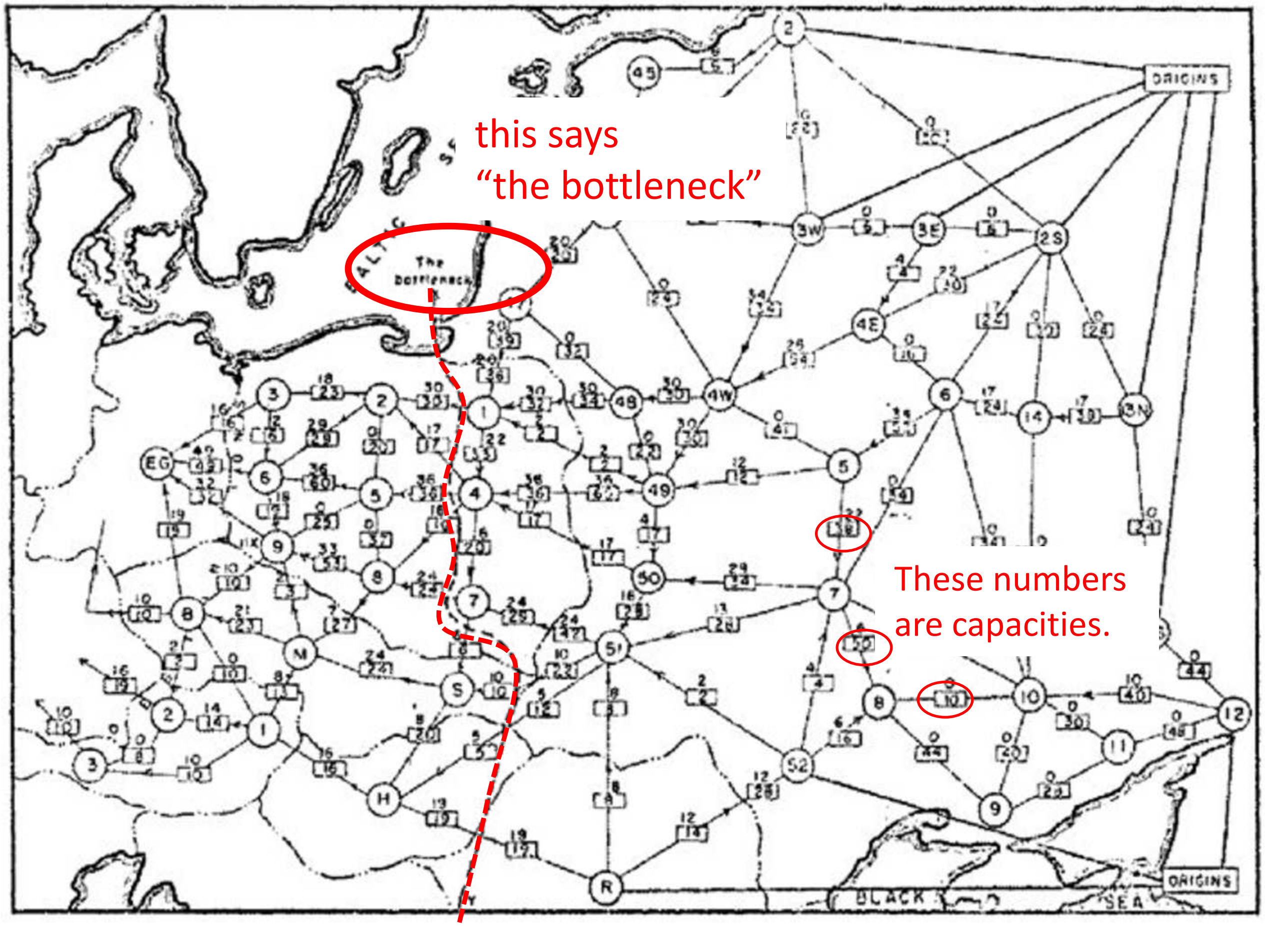
A minimum **s-t cut**

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

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



Example where this comes up

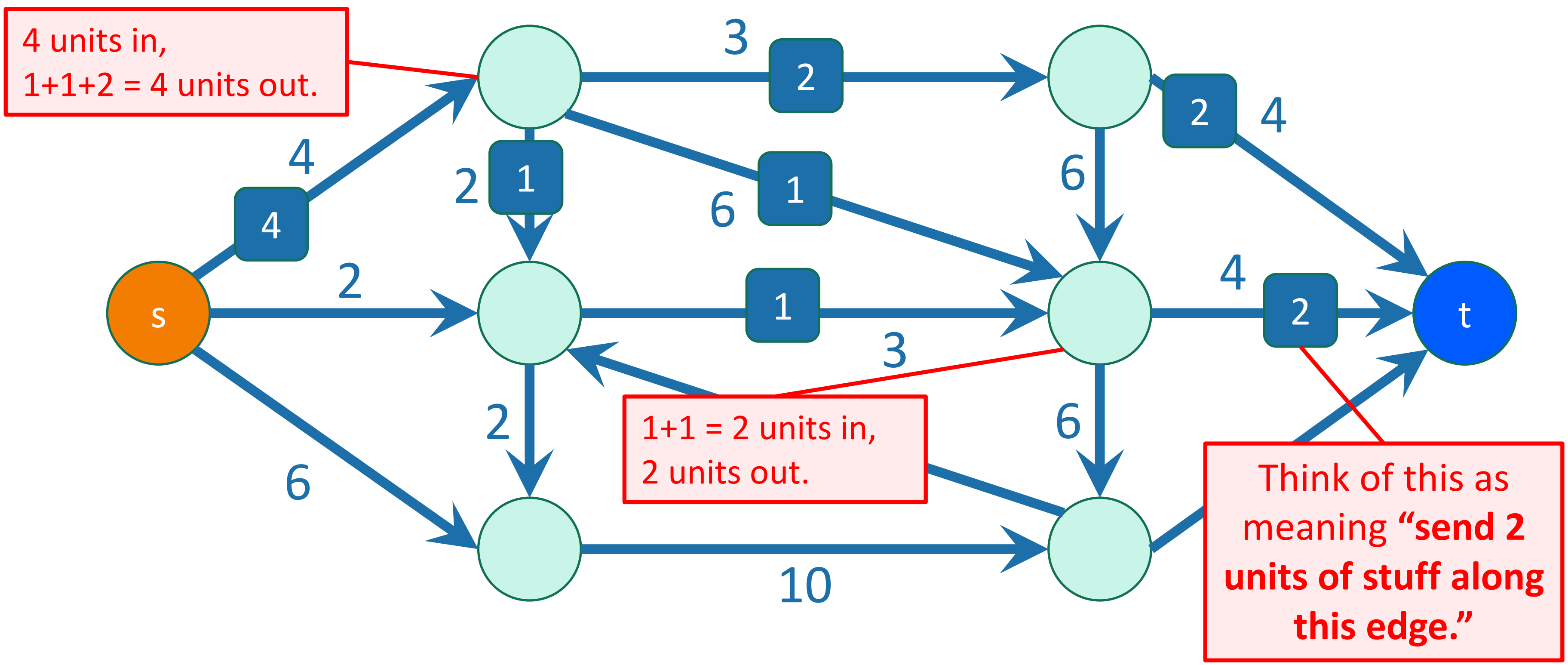


Schriver 2002

- 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 at the RAND corporation 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 that its capacity.
- At each vertex, the incoming flows must equal the outgoing flows.



Flows

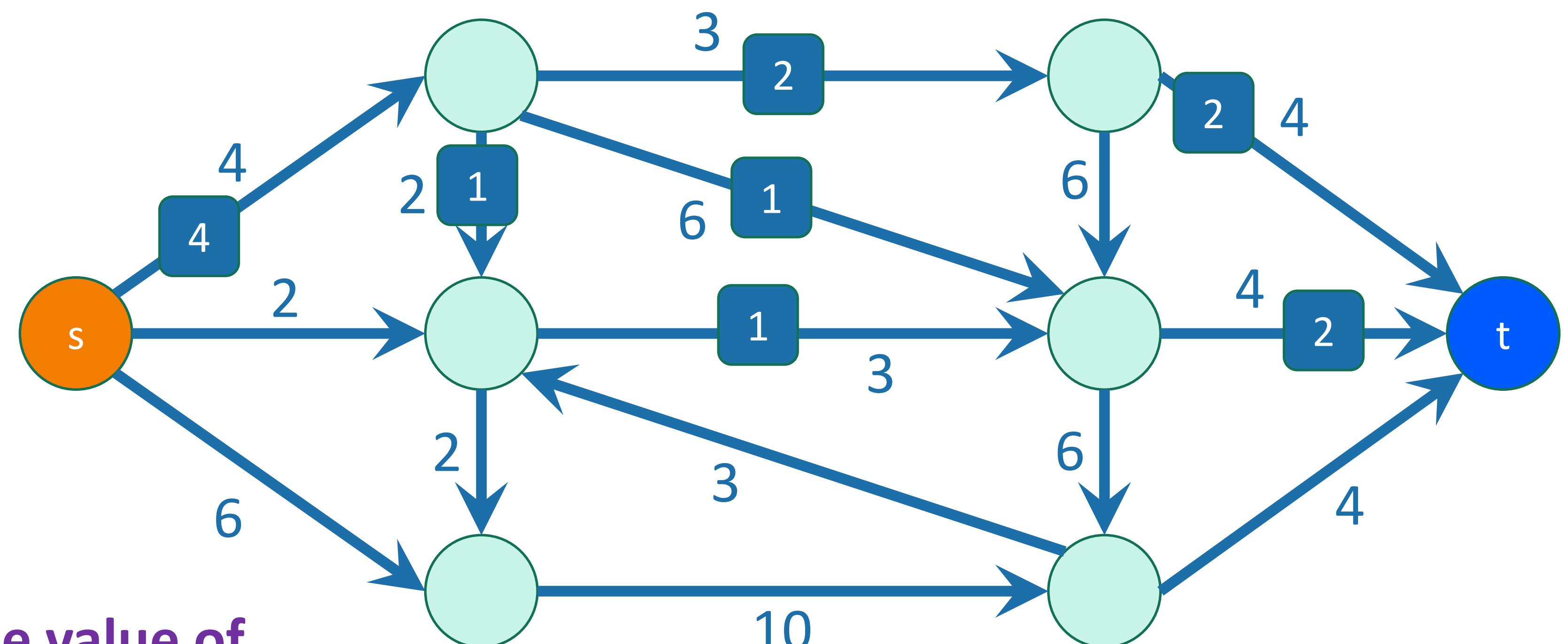
- The value of a flow is:
  - The amount of stuff coming out of s
  - The amount of stuff flowing into t
  - These are the same!

Because of conservation of flows at vertices,

stuff you put in

=

stuff you take out.

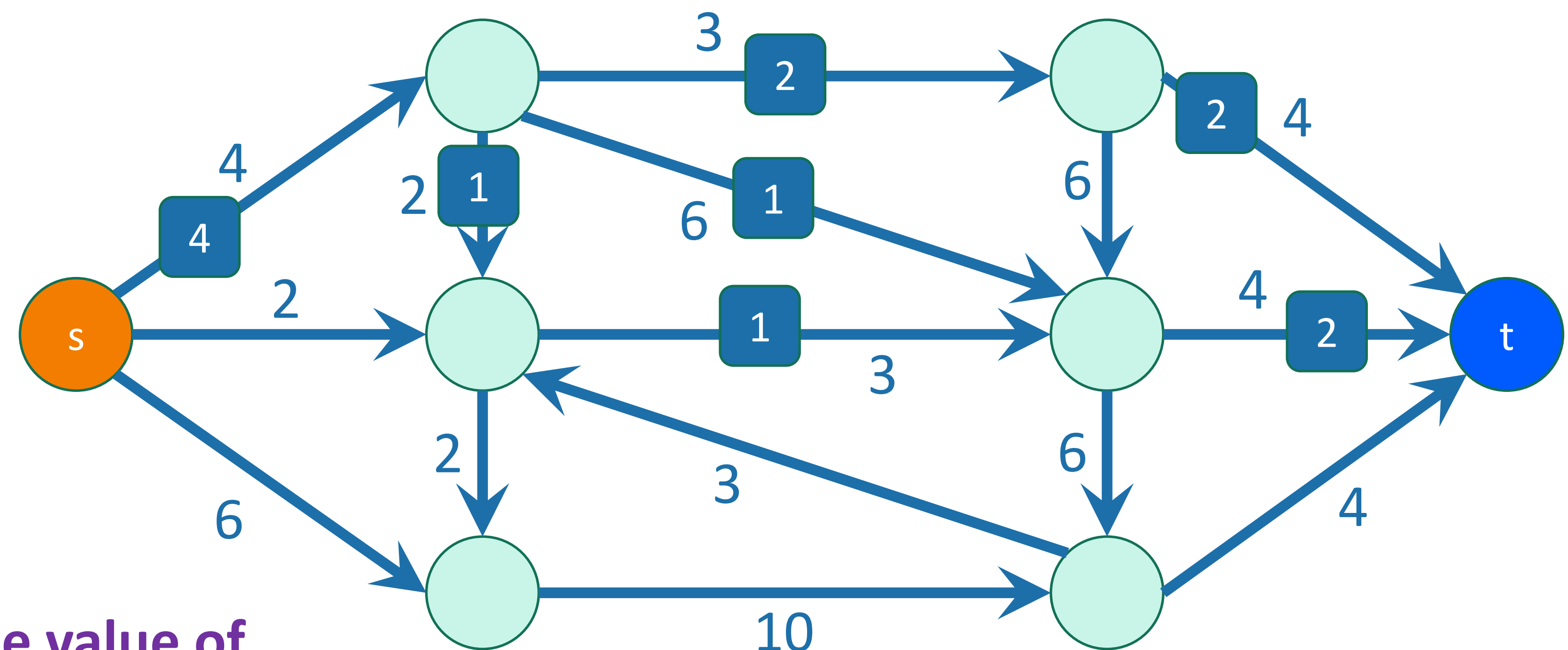


The value of this flow is 4.

A maximum flow

is a flow of maximum value.

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



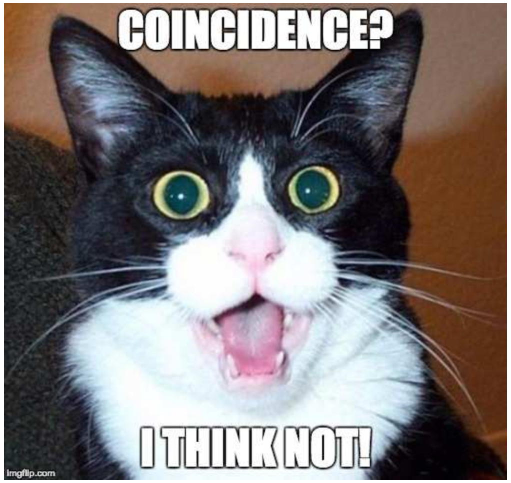
The value of this flow is 4.

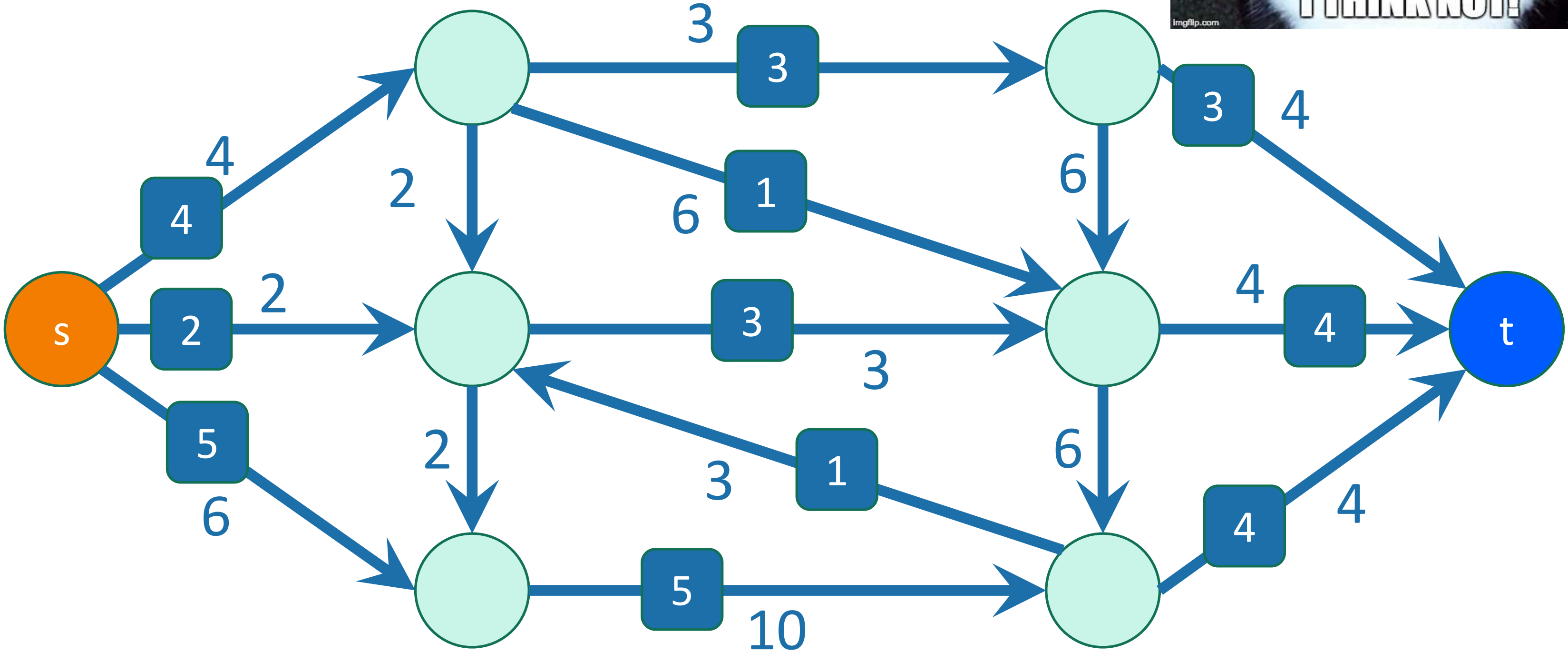
A maximum flow

is a flow of maximum value.

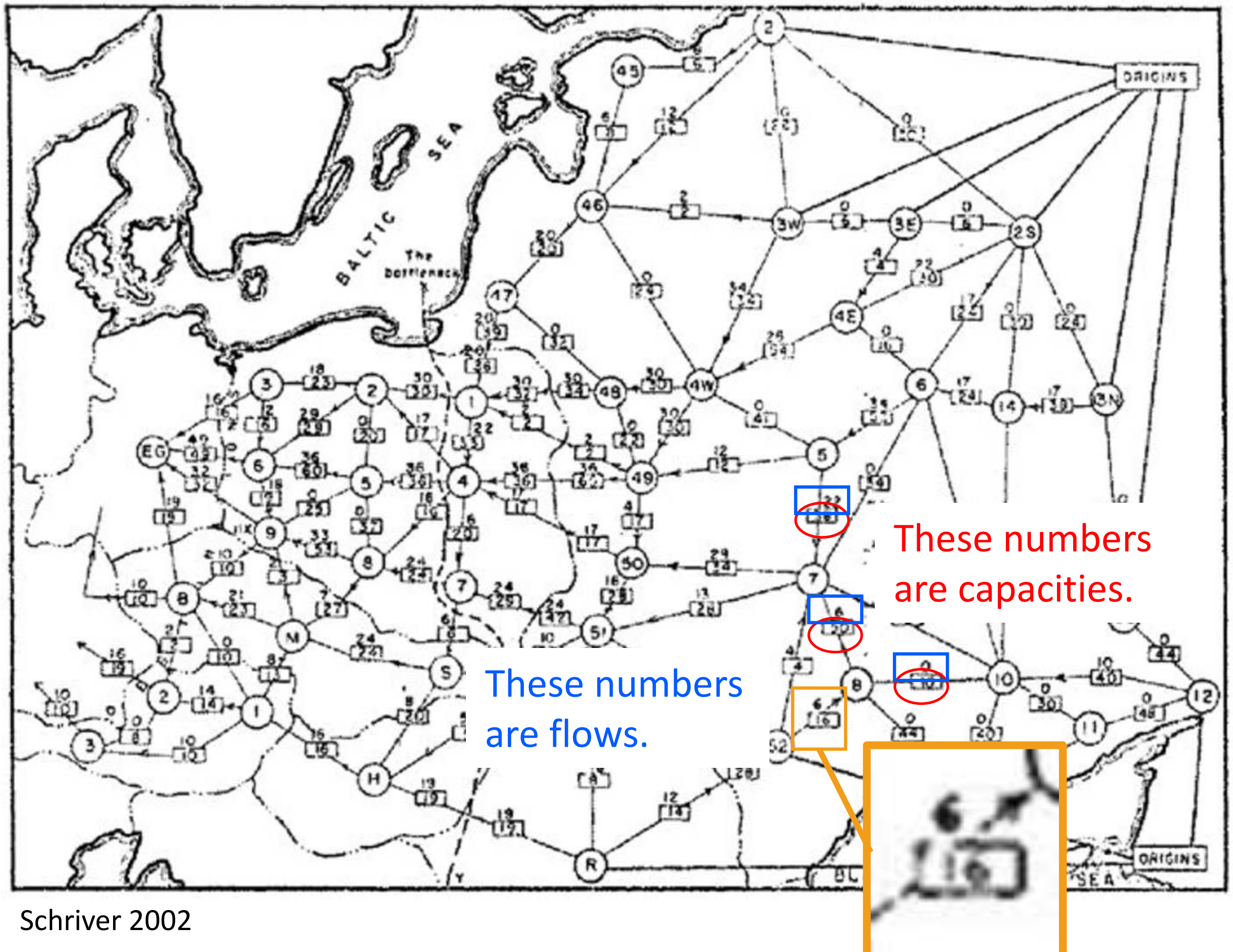
- This one is maximal; it has value 11.

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





Example

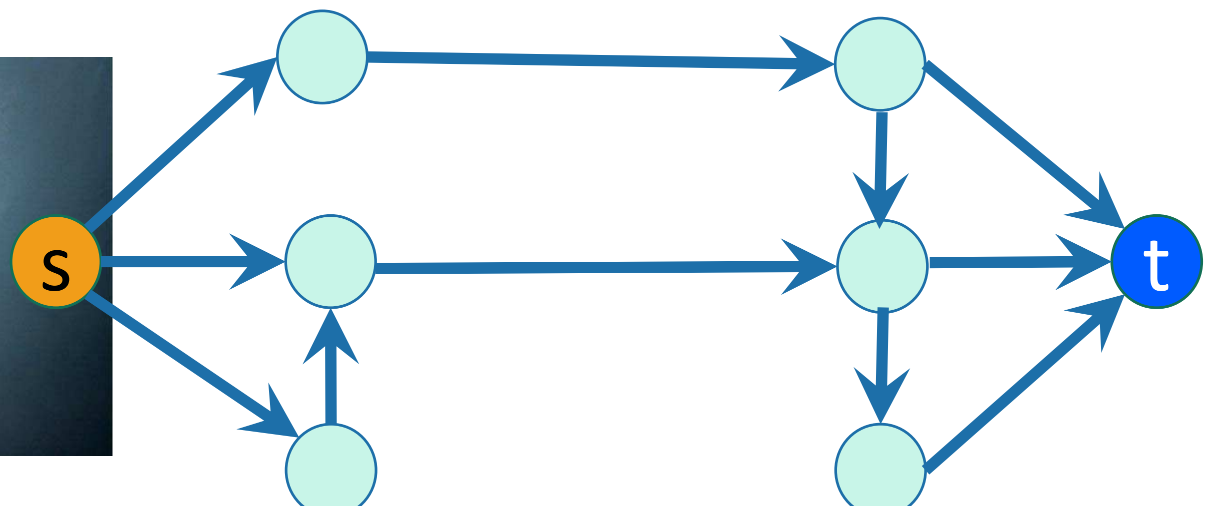


Schriver 2002

- 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.

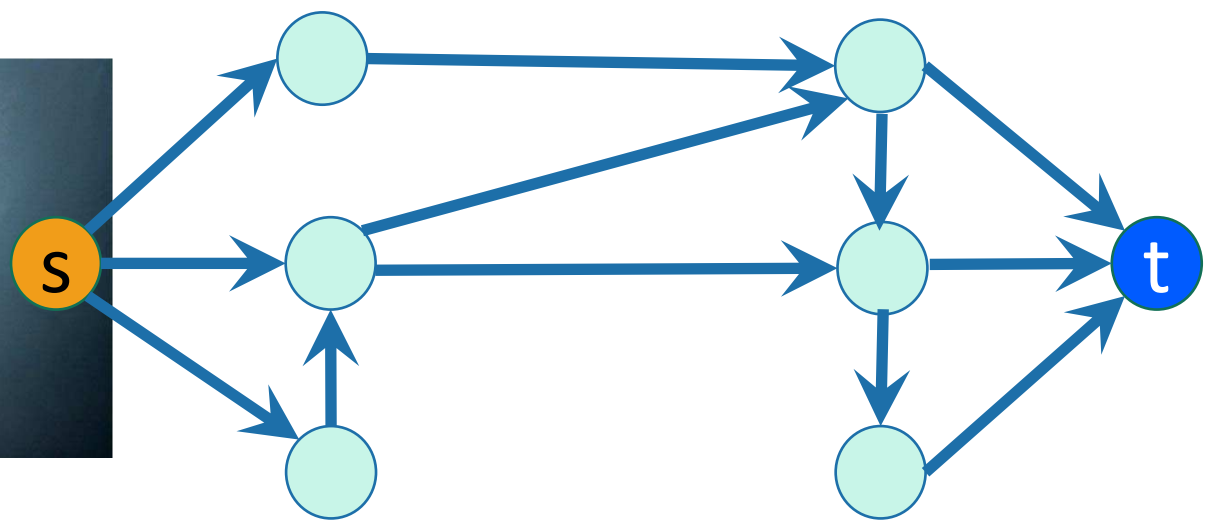
Pre-lecture exercise

- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t?
- 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?



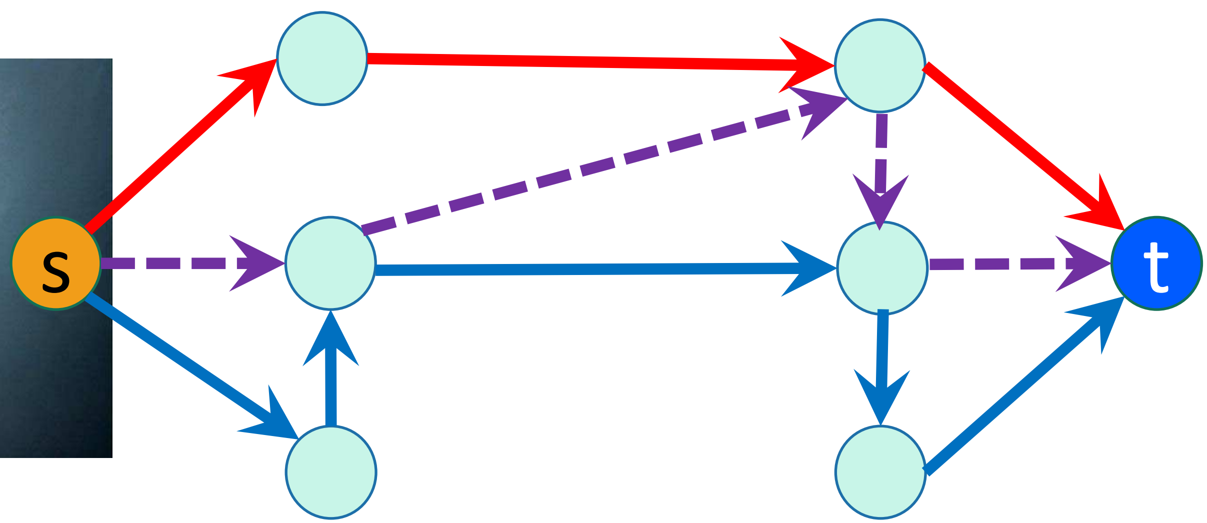
How about now?

- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t?
- 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?



How about now?

- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t?
- 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?



Pre-lecture exercise

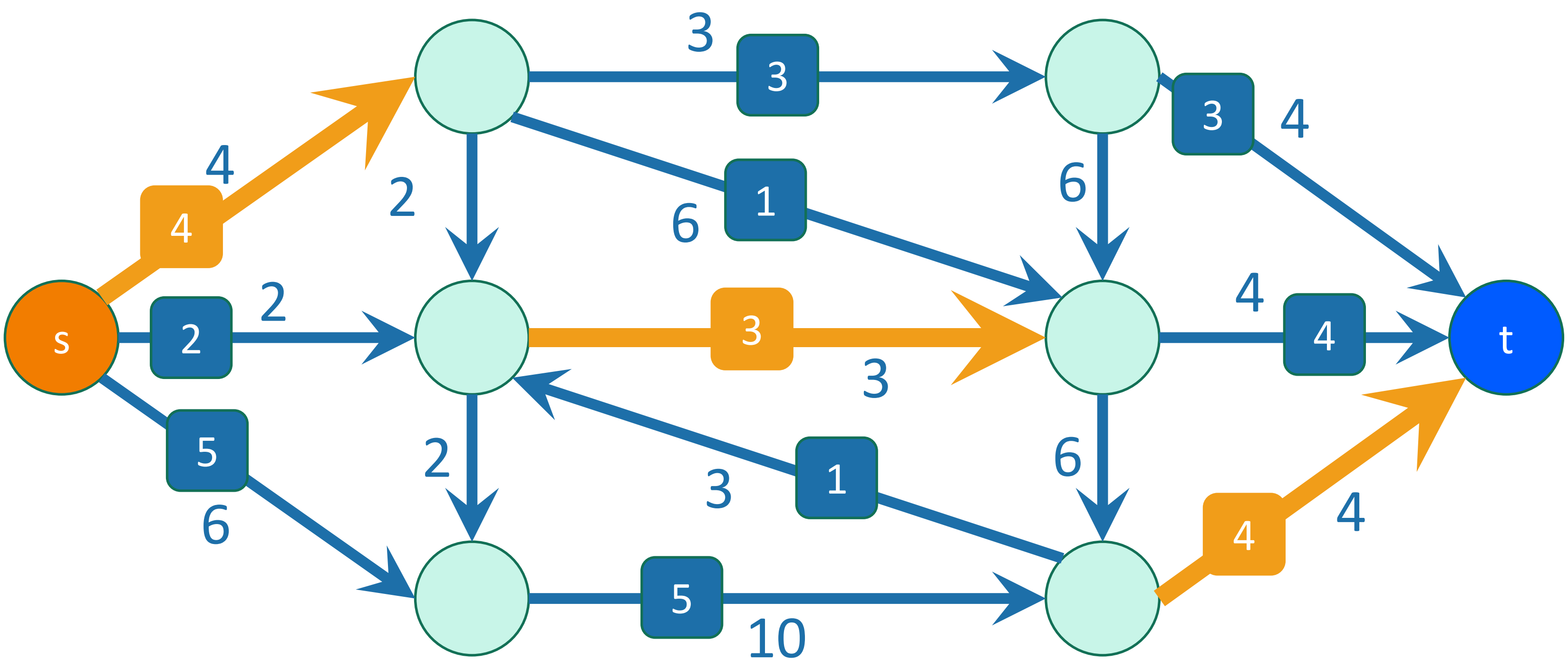
- Can you come up with a graph where the two numbers are different?

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

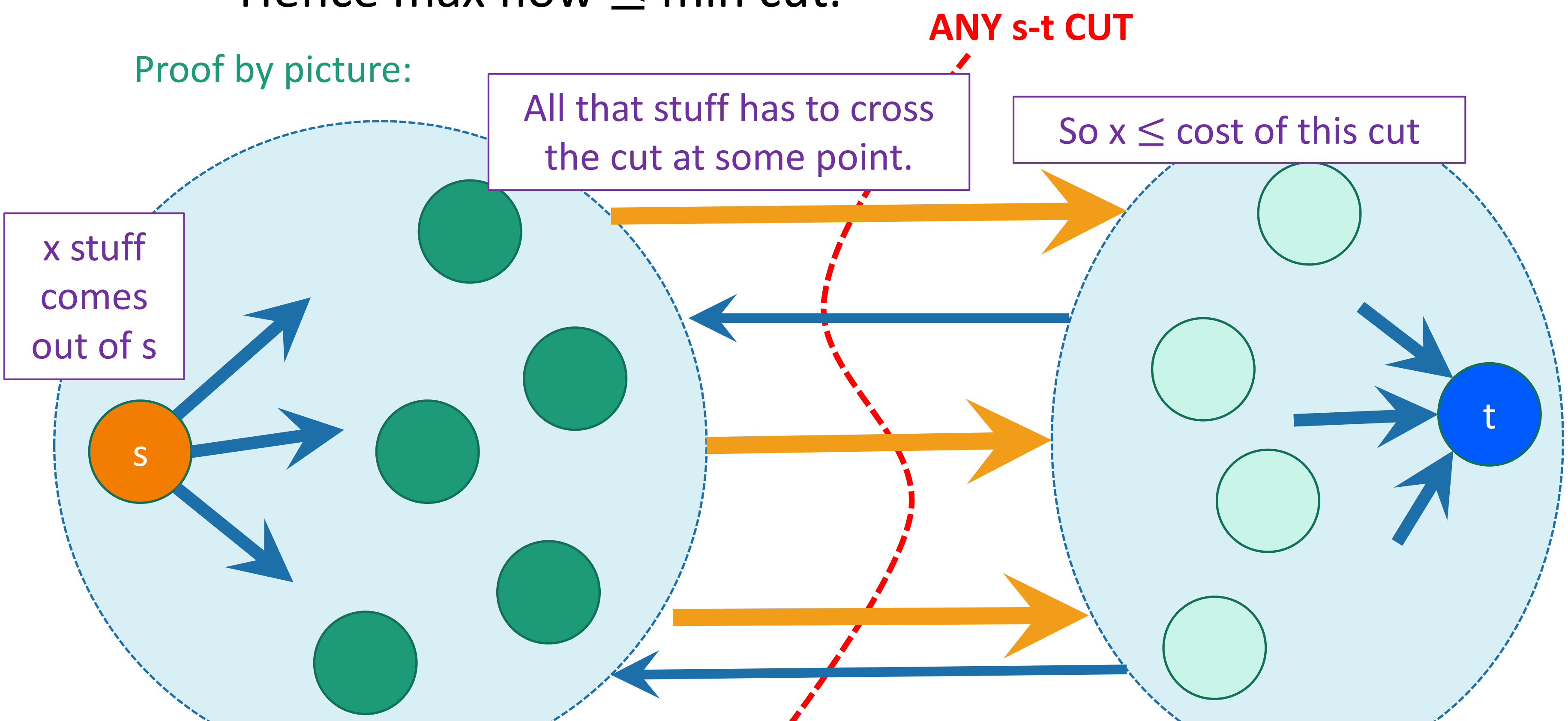
- Lemma 1: max flow $\leq$ min cut.
  - Proof-by-picture
- Lemma 2: max flow $\geq$ min cut.
  - Proof-by-algorithm, using a "Residual graph" G_f
  - Sub-Lemma: t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow.
    - $\Leftarrow$ first we do this direction:
    - Claim: If there is a path from s to t in G_f, then we can increase the flow in *G*.
    - Hence we couldn't have started with a max flow.
    - $\Rightarrow$ for this direction, proof-by-picture again.

This claim actually gives us an algorithm: Find paths from s to t in G_f and keep increasing the flow until you can't anymore.

Proof 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 max flow ≤ min cut.

Proof by picture:



Proof 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 max flow $\leq$ min cut.

- That was proof-by-picture.
- See the notes for proof-by-proof.
  - You are **not** responsible for proof-by-proof on the final.

Proof 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 max flow $\leq$ min cut.

- The theorem is stronger:
  - max flow $=$ min cut
  - Need to show max flow $\geq$ min cut.
  - Next: Proof by algorithm!

Ford-Fulkerson algorithm

- Usually we state the algorithm first and then prove that it works.
- Today we're going to just start with the proof, and this will inspire the 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.

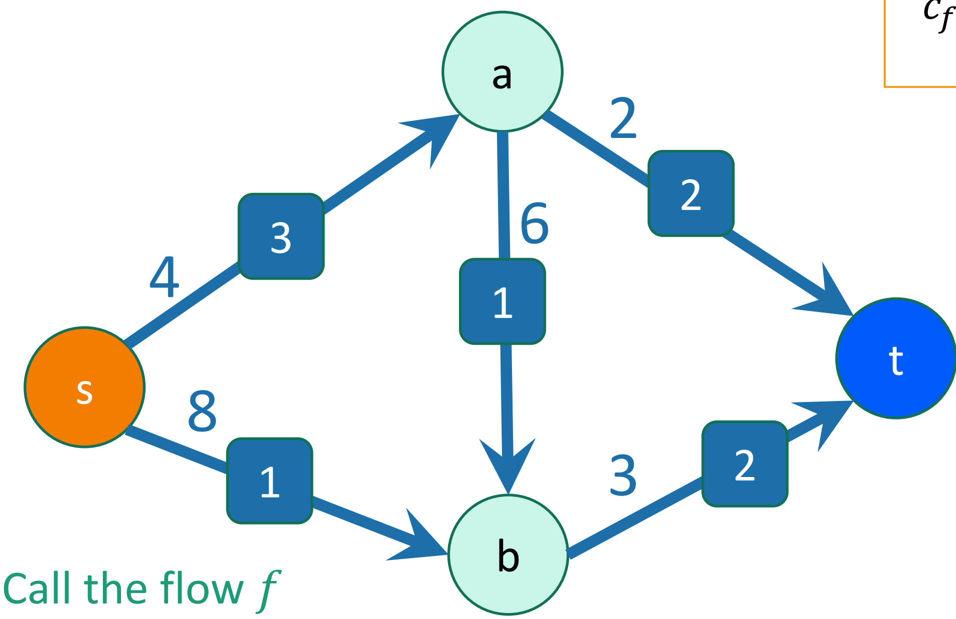
Assume for today that we don't have edges like this, although it's not necessary.

Tool: Residual networks

Say we have a flow

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & if \ \ (u,v) \in E \\ f(v,u) & if \ (v,u) \in E \\ 0 & else \end{cases}$$

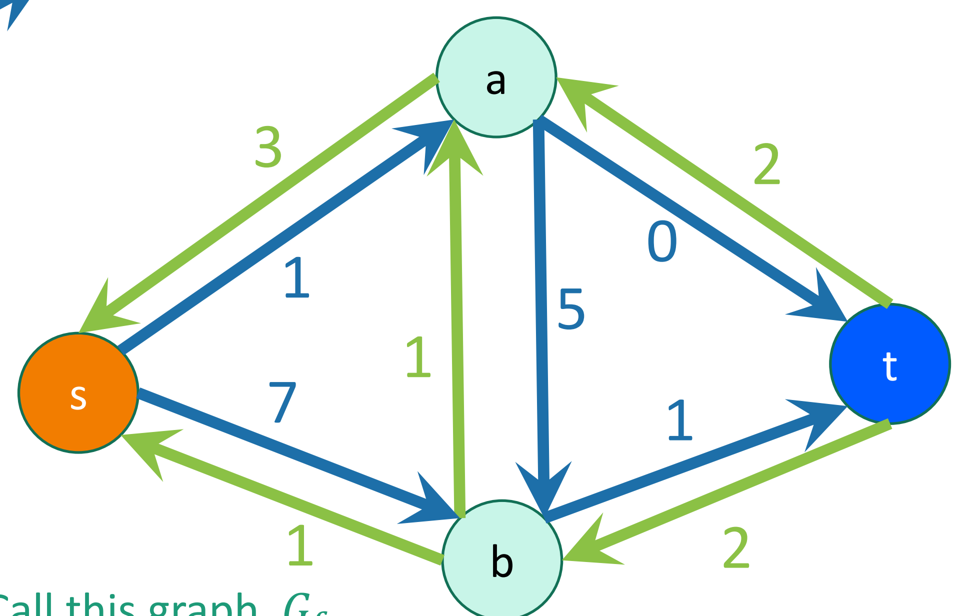
- $f(u,v)$ is the flow on edge (u,v).
- c(u,v) is the capacity on edge (u,v)



Call the flow f

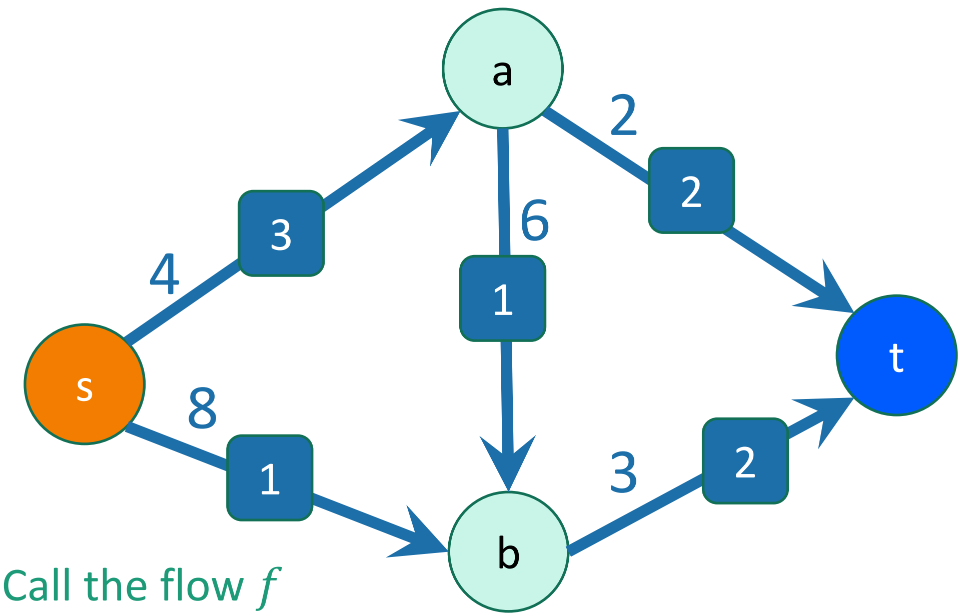
Call the graph G

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



Call this graph G_f

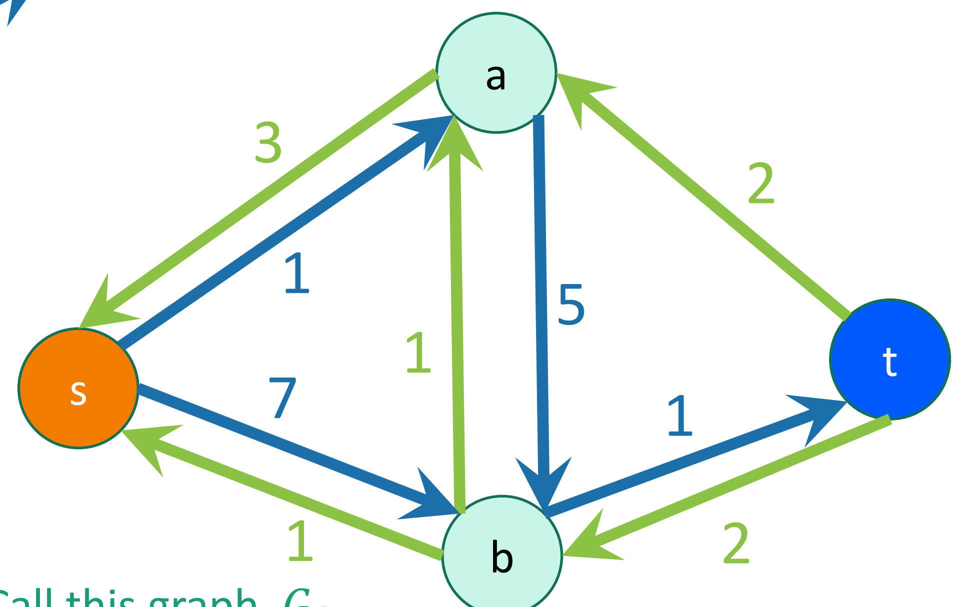
Tool: Residual networks
Say we have a flow



Call the flow f
Call the graph G

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

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



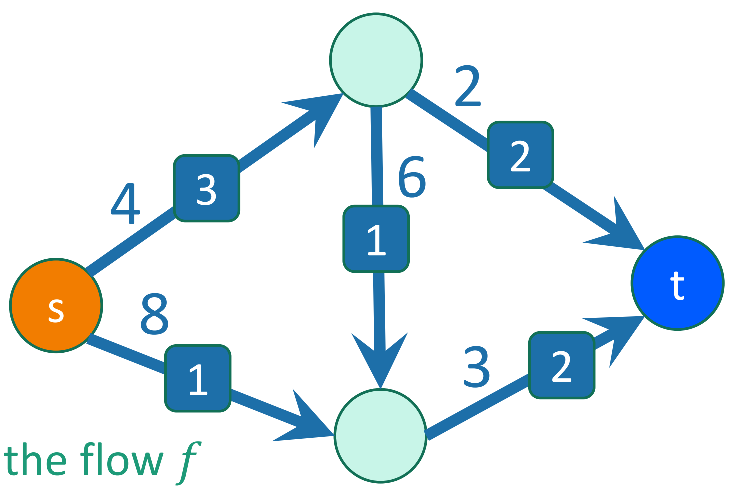
Call this graph G_f

Why look at residual networks?

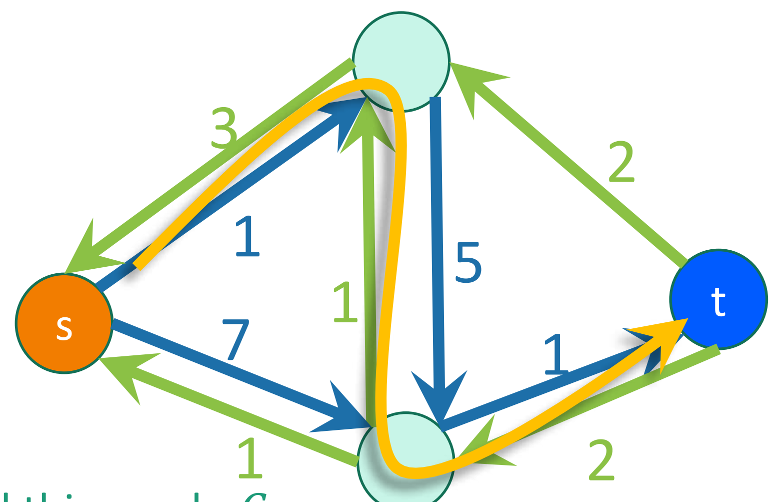
Lemma:

- t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow.

Example: **s is reachable from t in this example, so not a max flow.**



Call the flow f
Call the graph G



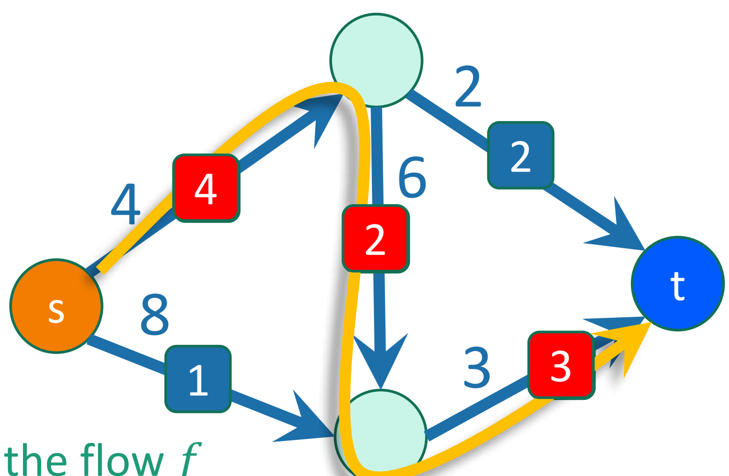
Call this graph G_f

Why look at residual networks?

Lemma:

- t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow.

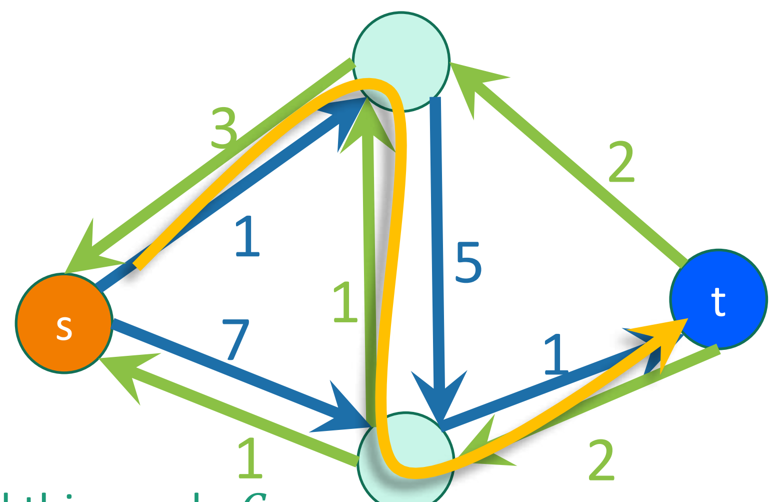
To see that this flow is not maximal, notice that we can improve it by sending one more unit more stuff along this path:



Call the flow f
Call the graph G

Example: **s is reachable from t in this example, so not a max flow.**

Now update the residual graph…



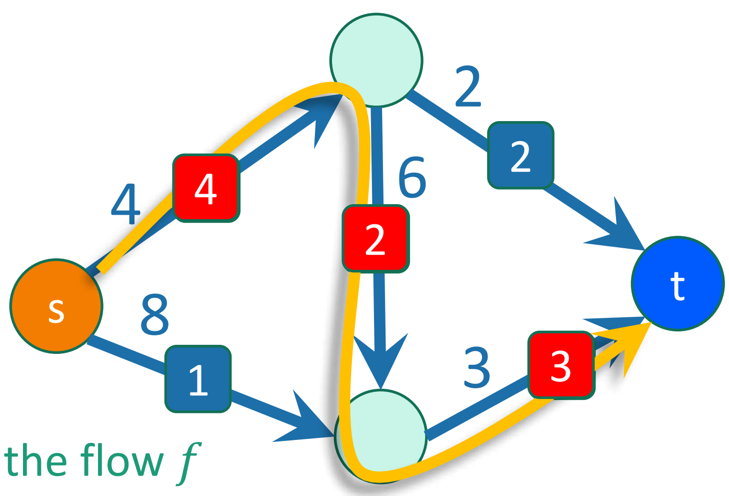
Call this graph G_f

Why look at residual networks?

Lemma:

- t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow.

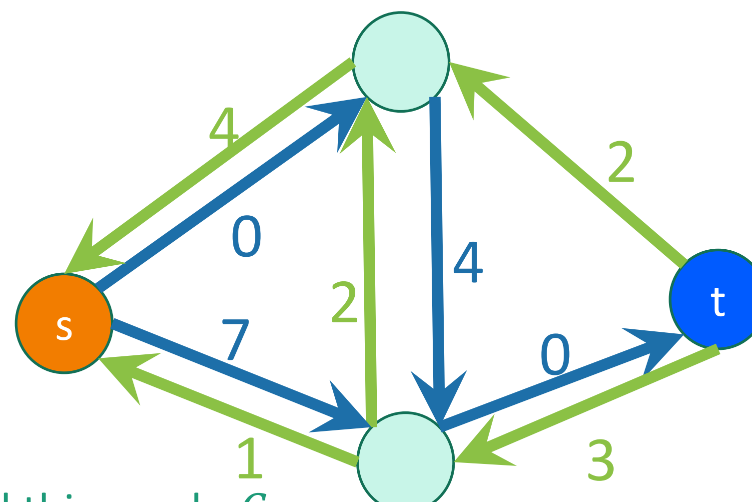
To see that this flow is not maximal, notice that we can improve it by sending one more unit more stuff along this path:



Call the flow f
Call the graph G

Example:

Now we get this residual graph:



Call this graph G_f

Why look at residual networks?

Lemma:

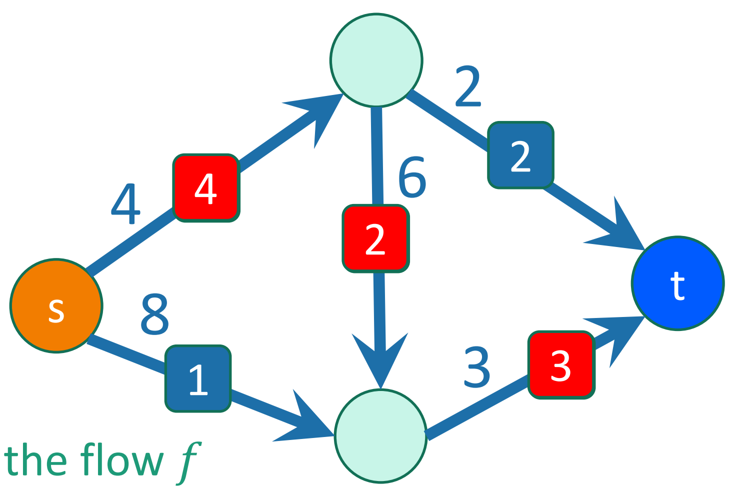
- t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow.

Example:

Now we get this residual graph:

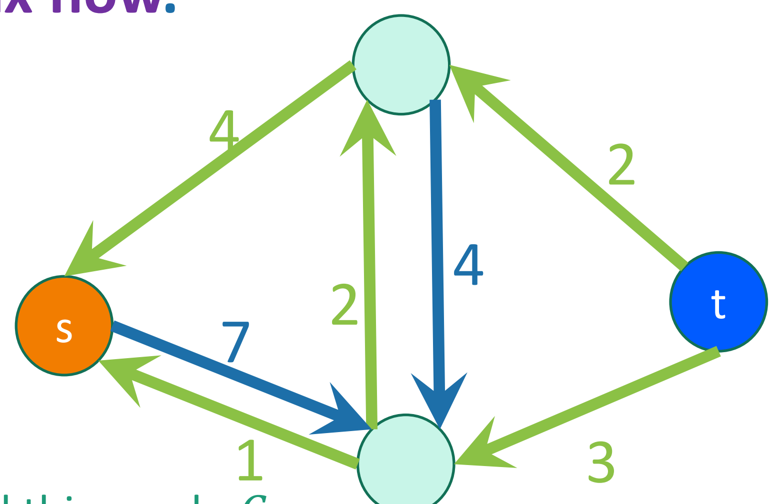
Now we can't reach t from s.

So the lemma says that f is a max flow.



Call the flow f

Call the graph G



Call this graph G_f

Let’s prove the Lemma

- t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow.

Lemma:

⇐ first this direction ⇐

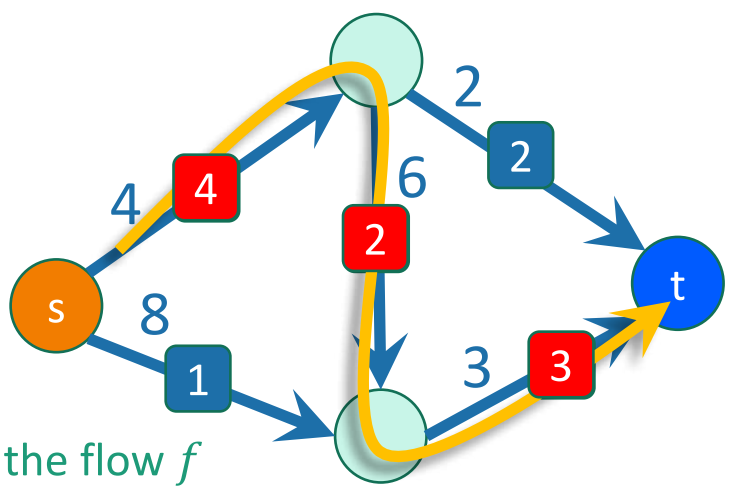
We will prove the contrapositive

t is not reachable from s in G_f ⇔ f is a max flow.

- Suppose there is a path from s to t in G_f.
  - This is called an augmenting path.
- **Claim:** if there is an augmenting path, we can increase the flow along that path.

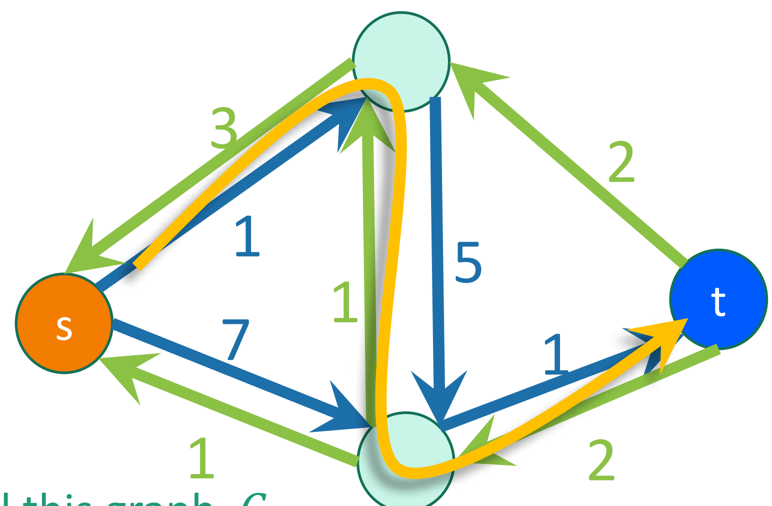
we will come back to this in a second.

- So do that and update the flow.
- This results in a bigger flow
  - so we can't have started with a max flow.



Call the flow f

Call the graph G

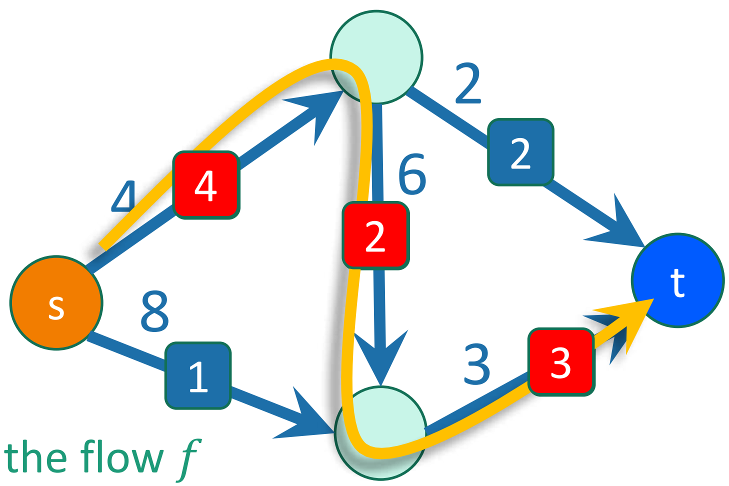


Call this graph G_f

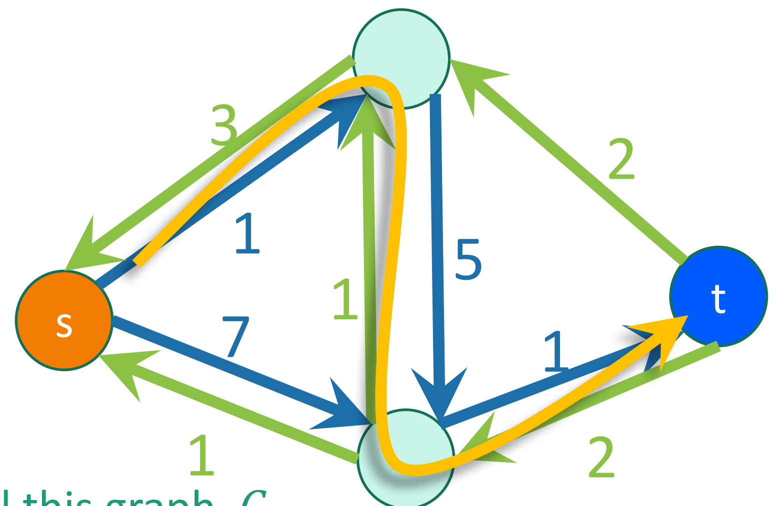
claim:

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

- In the situation we just saw, this is pretty obvious.



Call the flow f
Call the graph G



Call this graph G_f

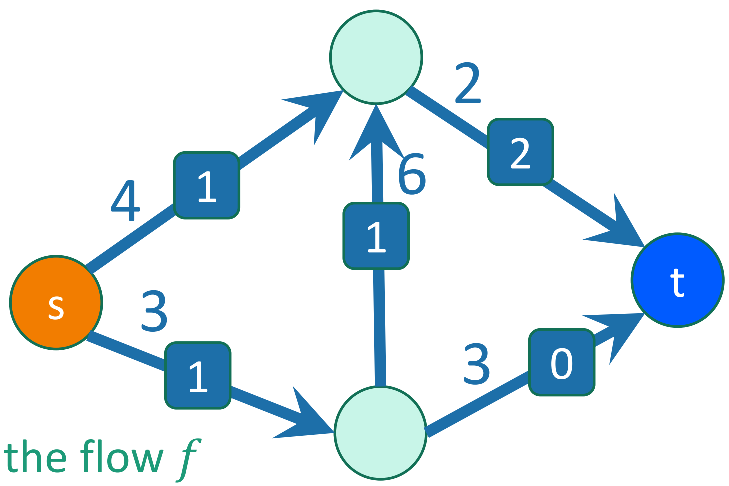
- Every edge on the path in G_f was a **forward edge**, so increase the flow on all the edges.

aka, an edge indicating how much stuff can still go through

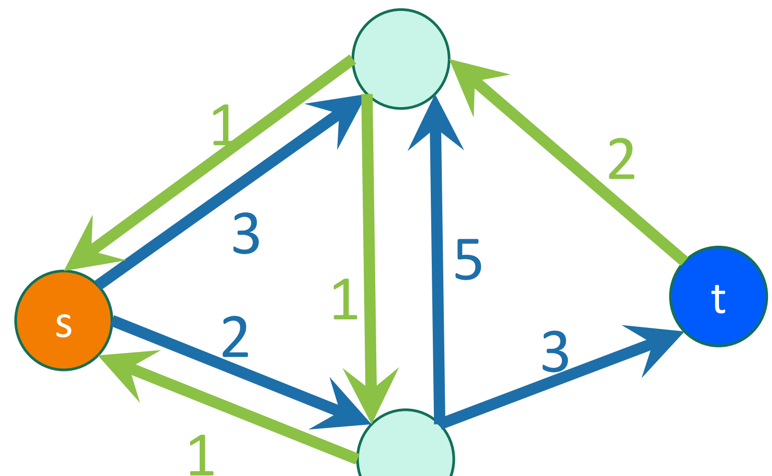
claim:

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

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



Call the flow f
Call the graph G



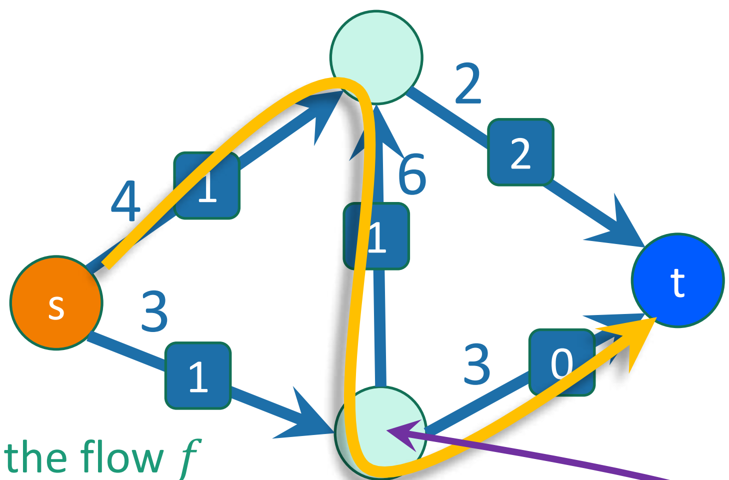
Call this graph G_f

I changed some of the weights and edge directions.

claim:

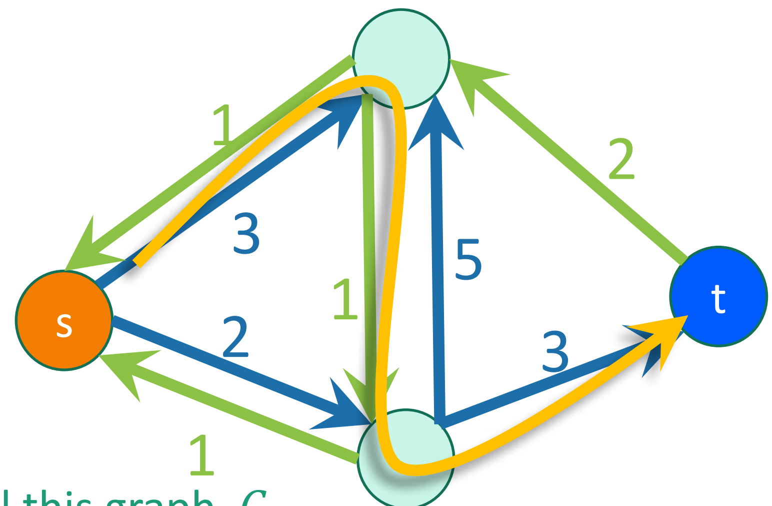
if there is an augmenting path, we can increase the flow along that path.

- But maybe 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!

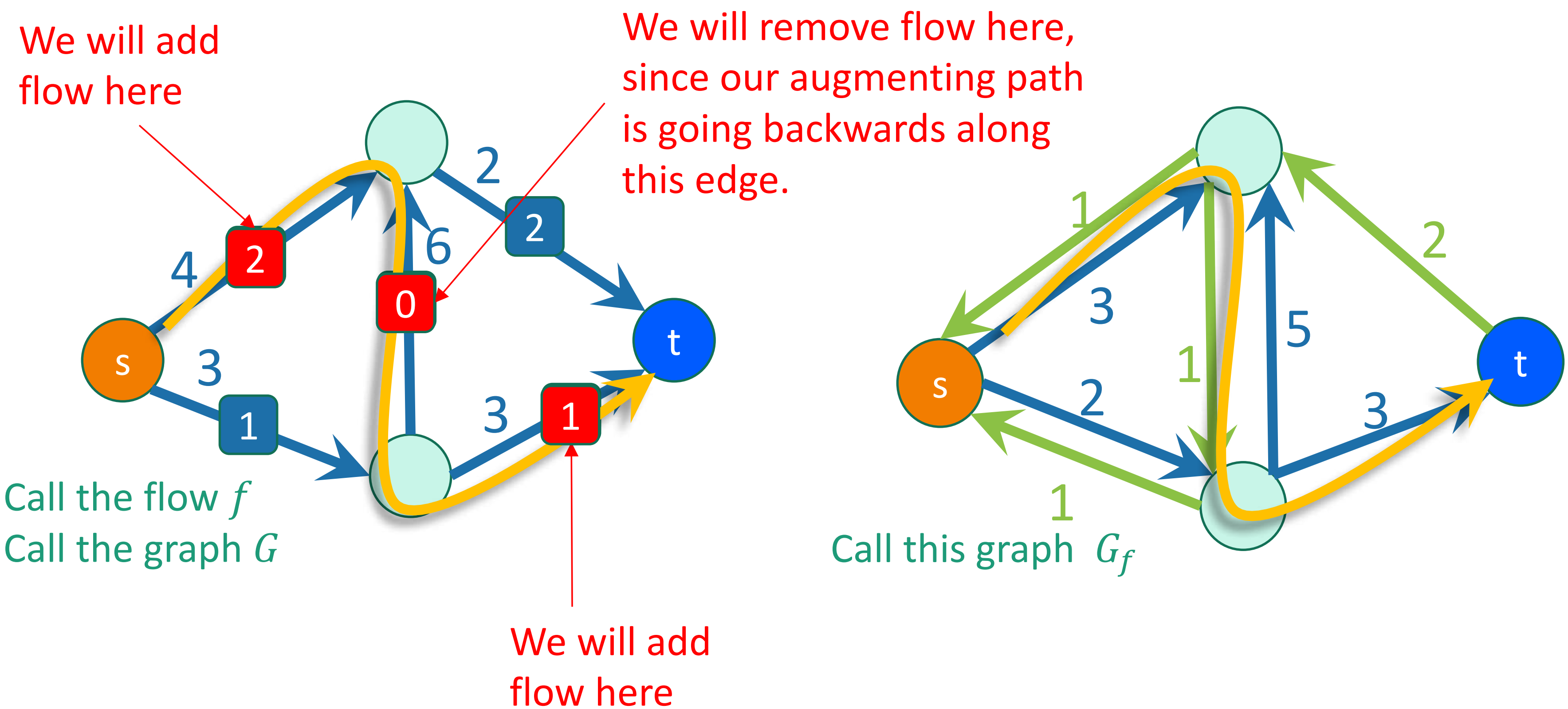
- For example, that will mess up the conservation of stuff at this vertex.

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:

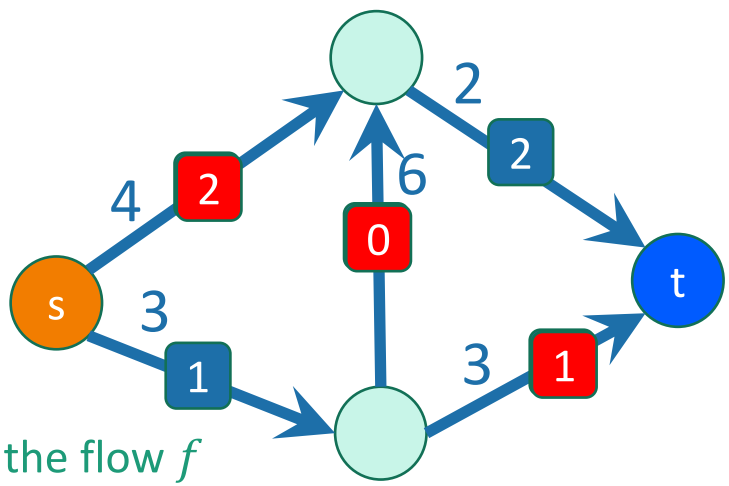


claim:

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

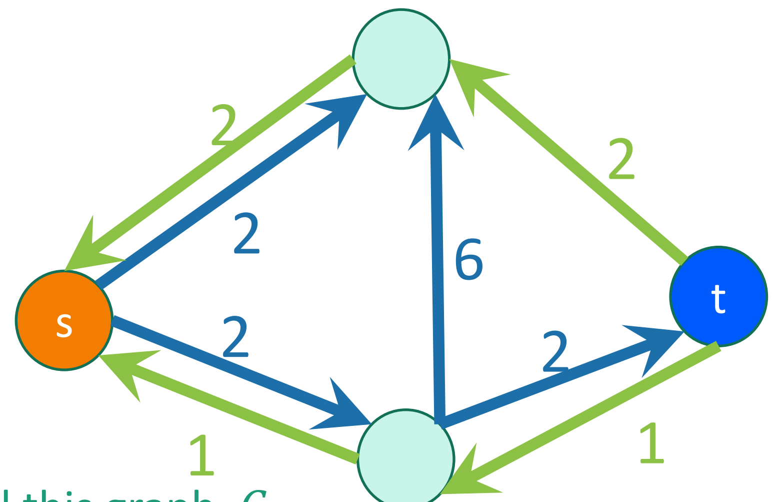
- In this case we do something a bit different:

Then we'll update the residual graph:



Call the flow f

Call the graph G

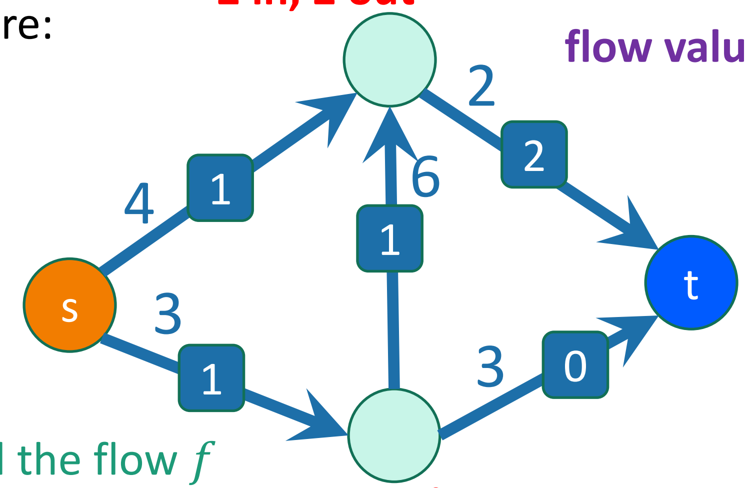


Call this graph G_f

Before:

2 in, 2 out

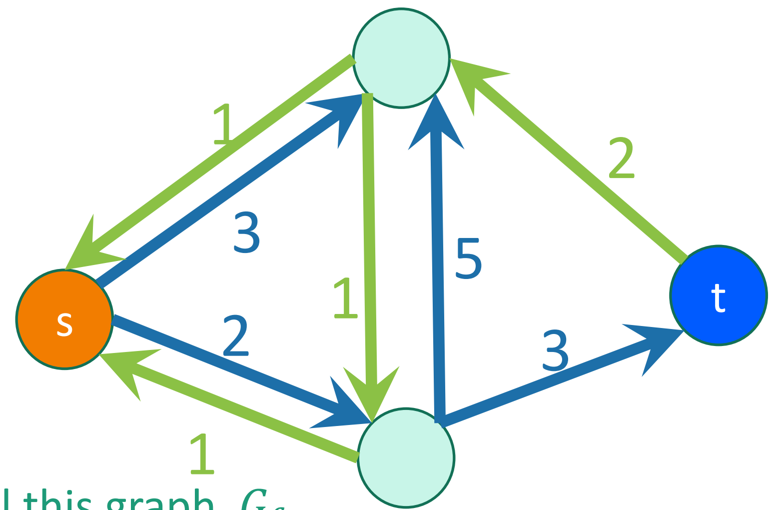
flow value is 2



Call the flow f

Call the graph G

1 in, 1 out

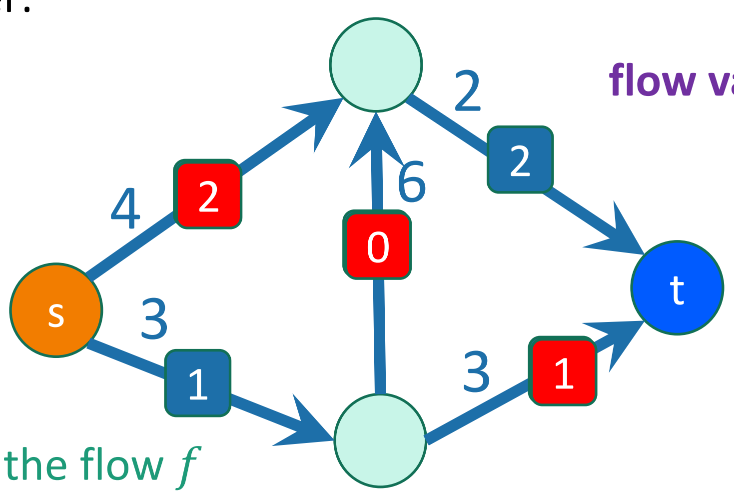


Call this graph G_f

After:

2 in, 2 out

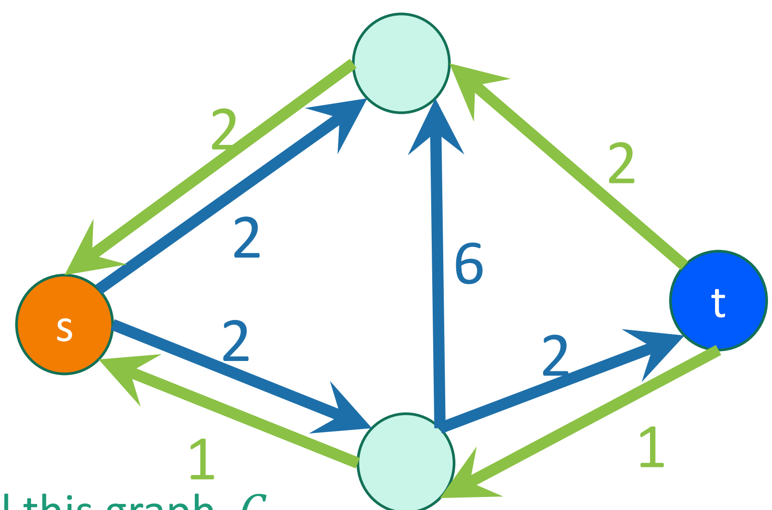
flow value is 3



Call the flow f

Call the graph G

1 in, 1 out



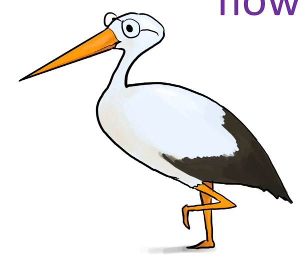
Call this graph G_f

Still a legit flow, but with a bigger value!

claim:

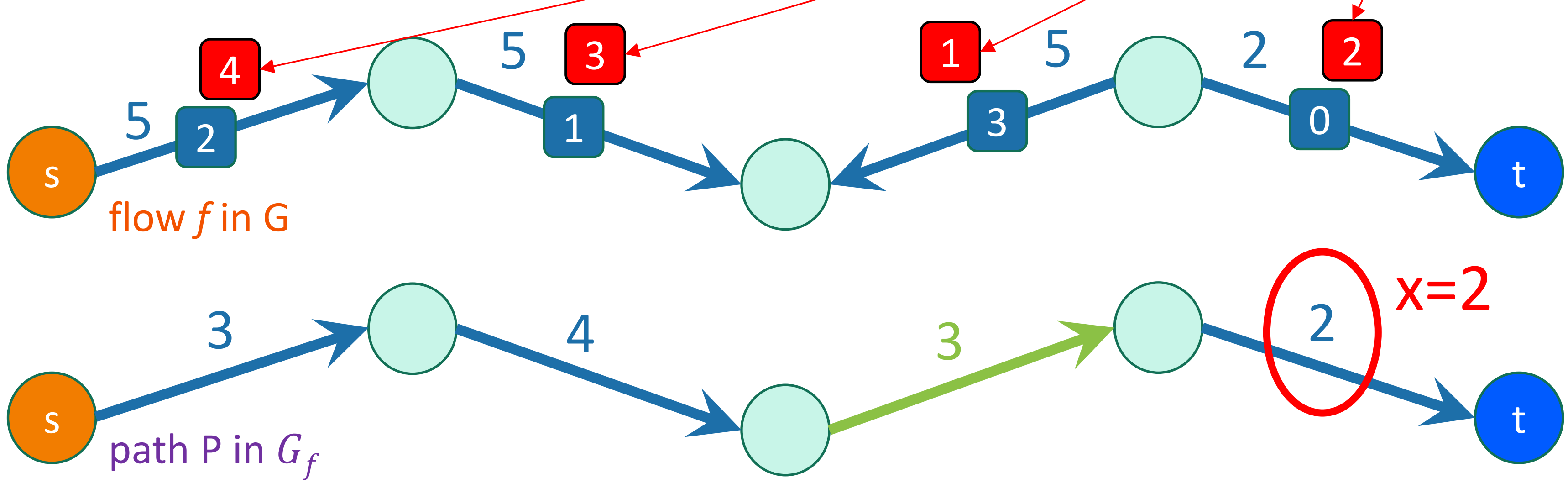
if there is an augmenting path, we can increase the flow along that path.

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



- 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'

This is f'



That proves the **claim**

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

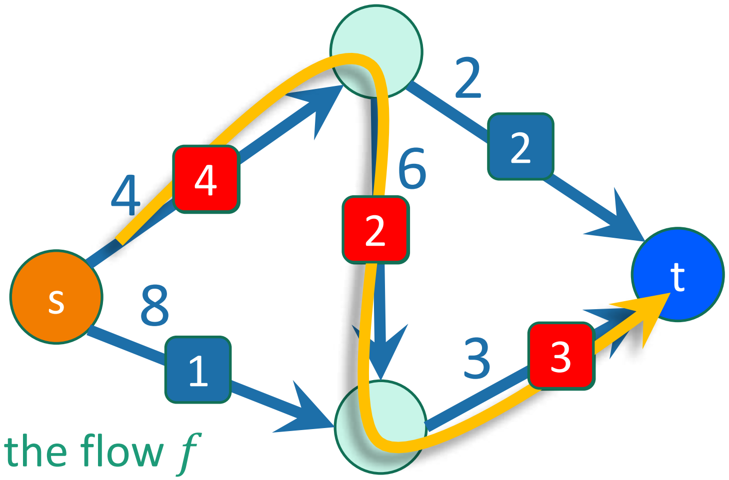
Lemma:

⇐ first this direction ⇐

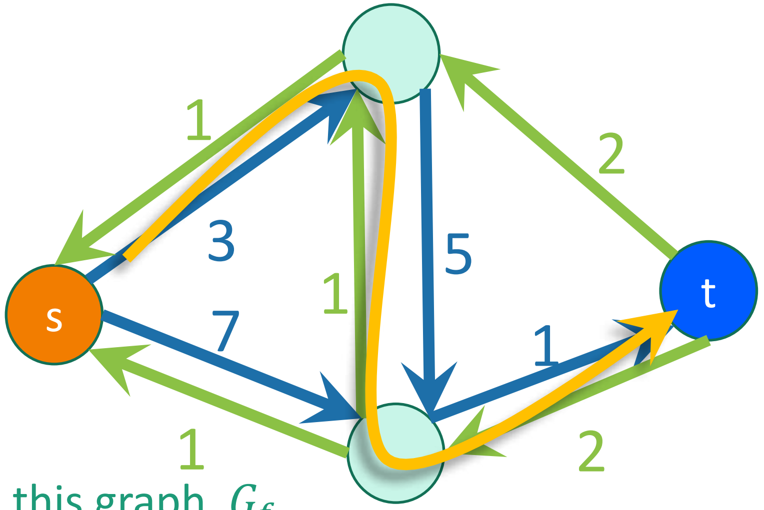
We will prove the contrapositive

t is not reachable from s in G_f ⇔ f is a max flow.

- Suppose there is a path from s to t in G_f.
  - This is called an augmenting path.
- Claim: if there is an augmenting path, we can increase the flow along that path. ✓
- So do that and update the flow.
- This results in a bigger flow
  - so we can't have started with a max flow. ✓



Call the flow f
Call the graph G



Call this graph G_f

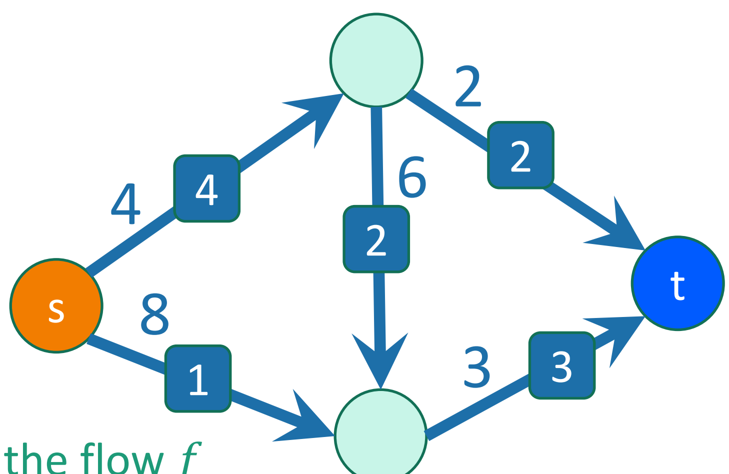
Lemma:

$\Rightarrow$ now this direction $\Rightarrow$

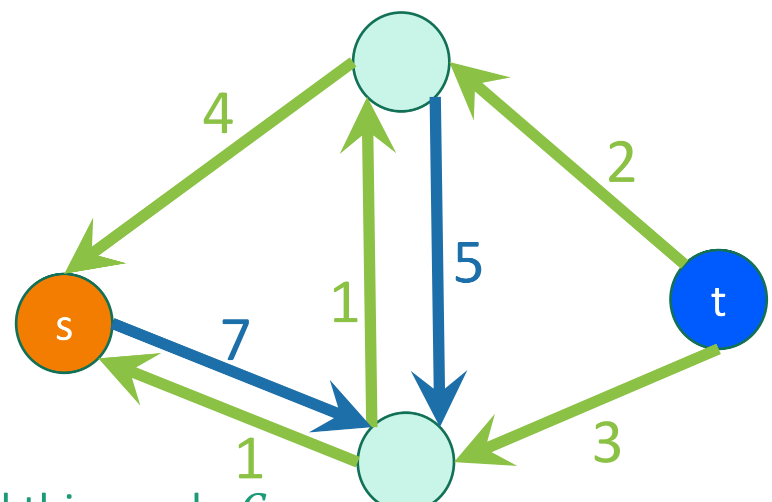
t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow.

- 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

Lemma:

⇒ now this direction ⇒

t is not reachable from s in G_f ⇔ f is a max flow.

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

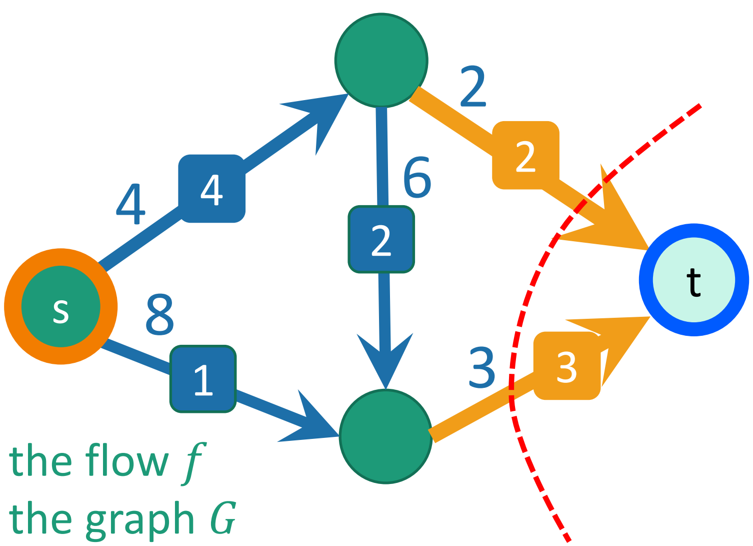
t lives here

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

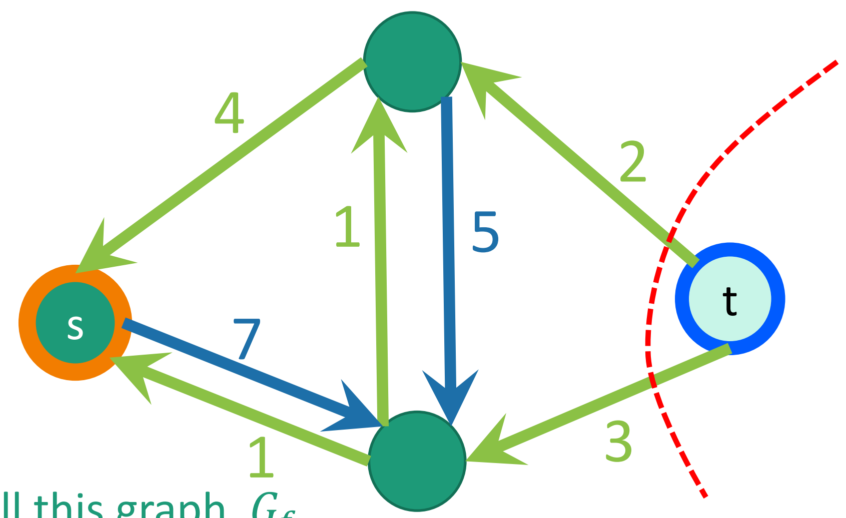
- The flow from s to t is **equal** to the cost of this cut.
  - Similar to proof-by-picture we saw before:
    - All of the stuff has to **cross the cut**.
    - The edges in the cut are **full** because they don't exist in G_f

Lemma 1

- **thus:** $\text{this flow value} = \text{cost of this cut} \geq \text{cost of min cut} \geq \text{max flow}$



Call the flow f
Call the graph G



Call this graph G_f

Lemma:

⇒ now this direction ⇒

t is not reachable from s in G_f ⇔ f is a max flow.

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

t lives here

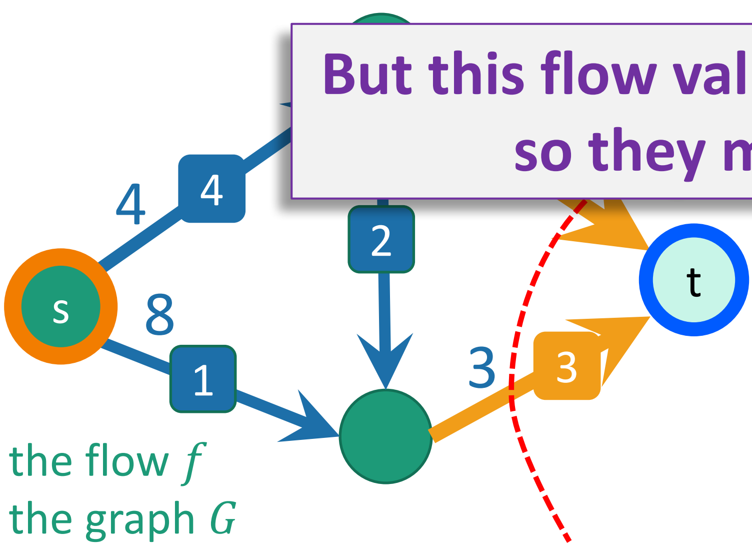
{things reachable from s} , {things not reachable from s}

- The flow 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

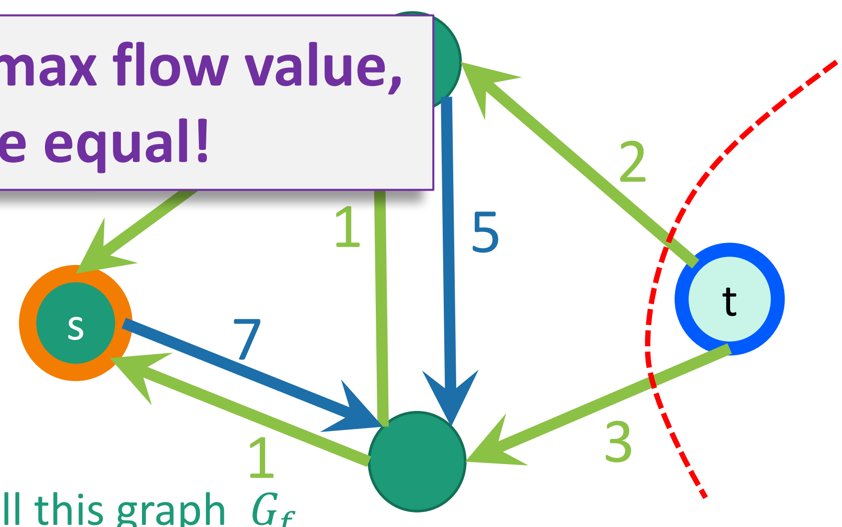
Lemma 1

- **thus:** this flow value $=$ cost of this cut $\geq$ cost of min cut $\geq$ max flow

But this flow value $\leq$ max flow value, so they must be equal!



Call the flow f
Call the graph G



Call this graph G_f

We've proved:

- t is not reachable from s in G_f $\Leftrightarrow$ f is a max flow

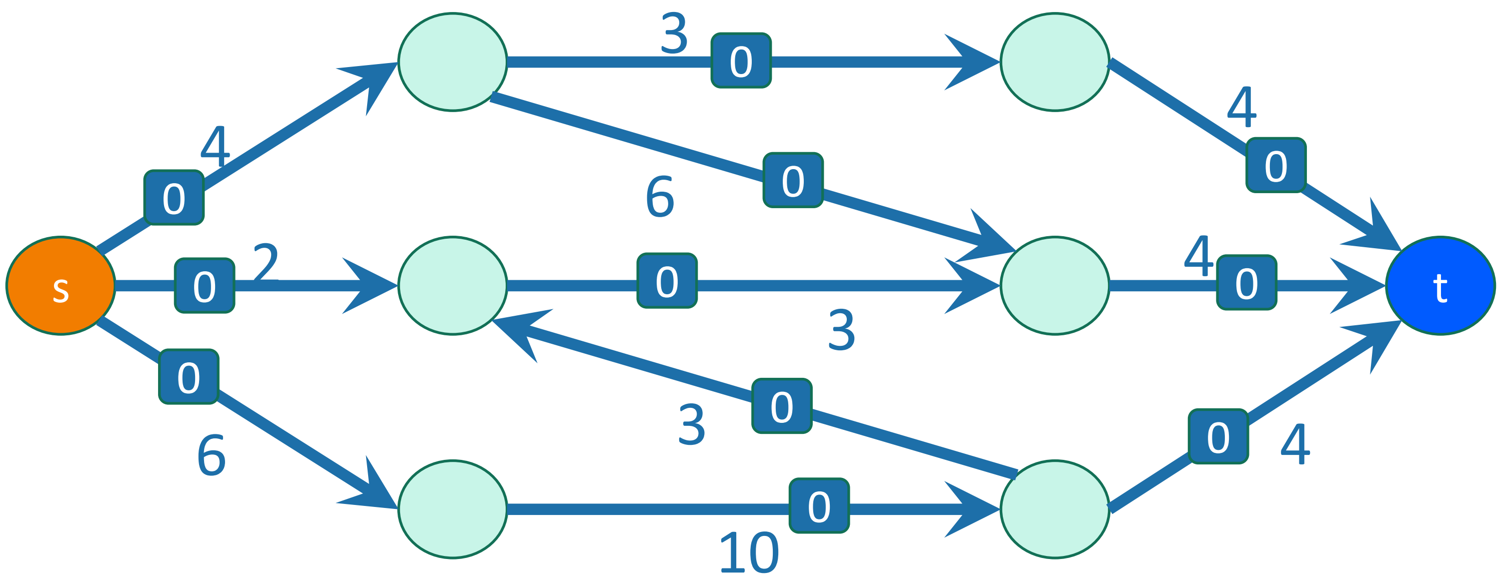
- This inspires an **algorithm**:
- **Ford-Fulkerson**(G):
  - f ← all zero flow.
  - $G_f \leftarrow G$
  - **while** t is reachable from s in G_f
    - Find a path P from s to t in G_f // eg, use BFS
    - f ← **increaseFlow**(*P*,f)
    - update G_f
  - **return** f

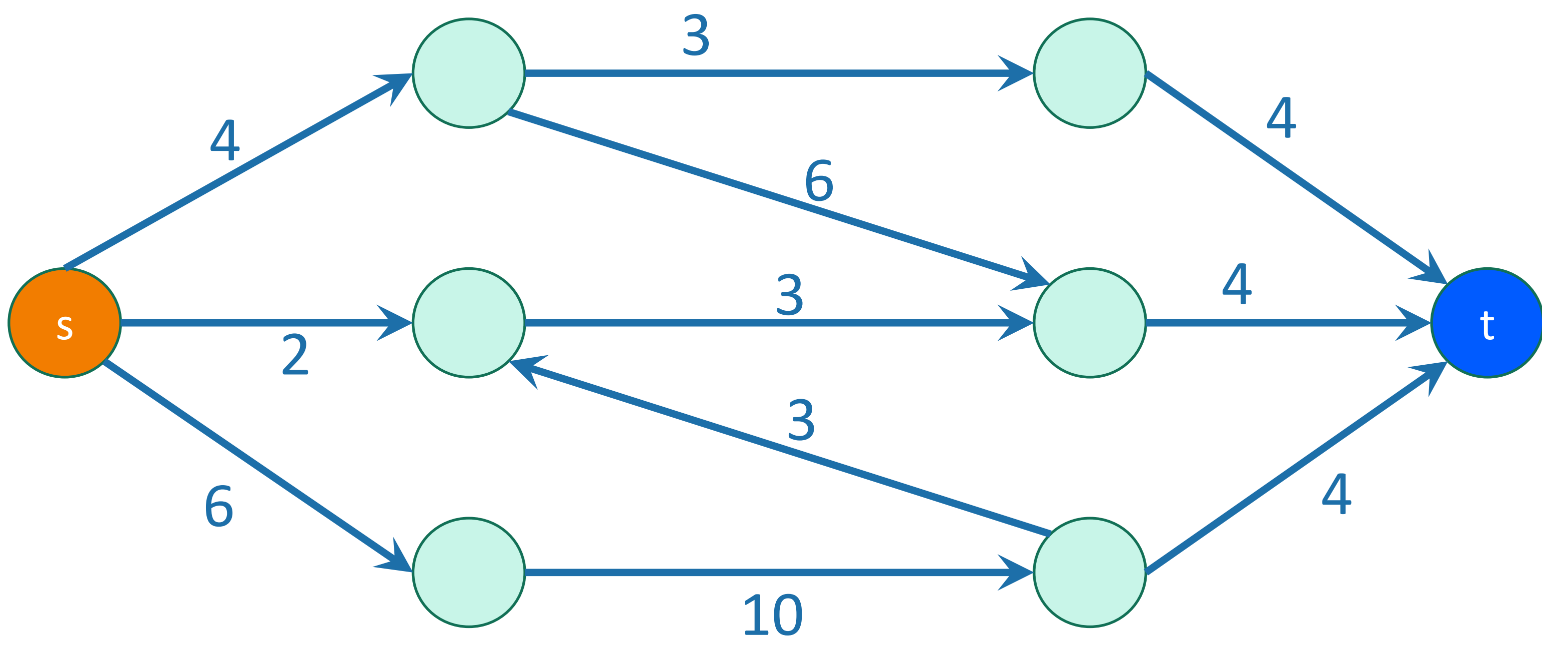
How do we choose which paths to use?

- 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?)

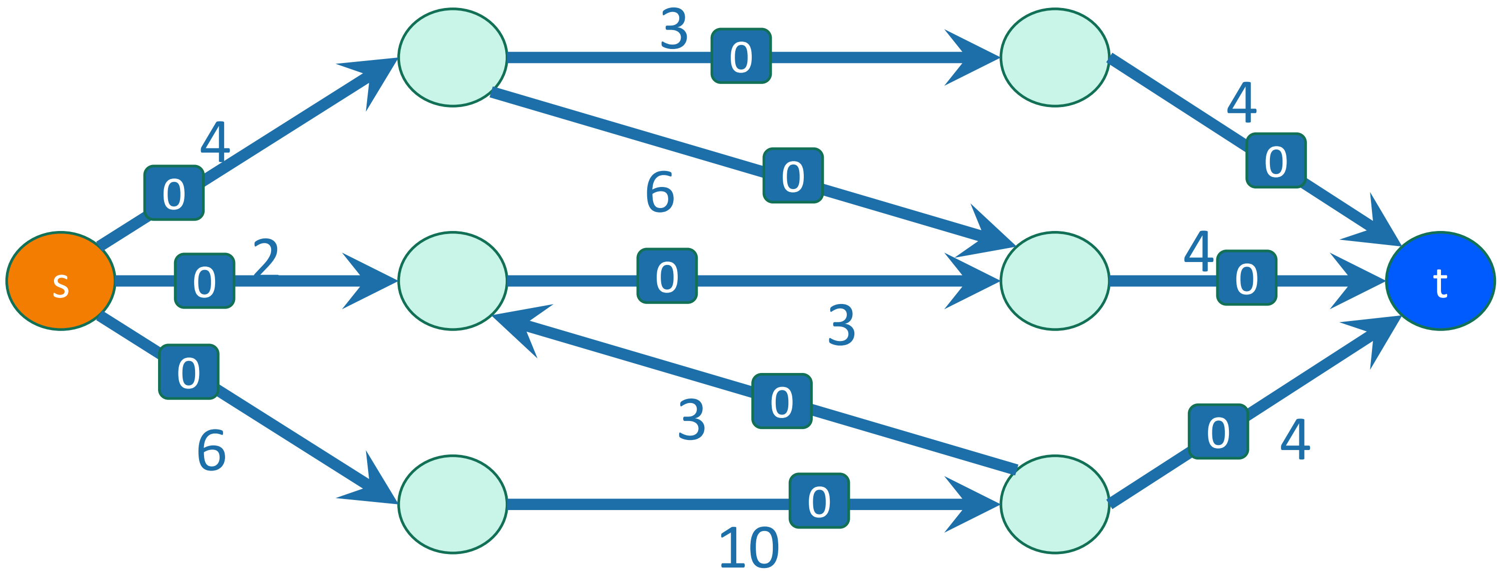
- We need to be careful with our path selection to make sure the algorithm terminates quickly.
  - Using BFS leads to the **Edmonds-Karp algorithm.** *(today)*
  - There's another way in the notes. *(optional)*

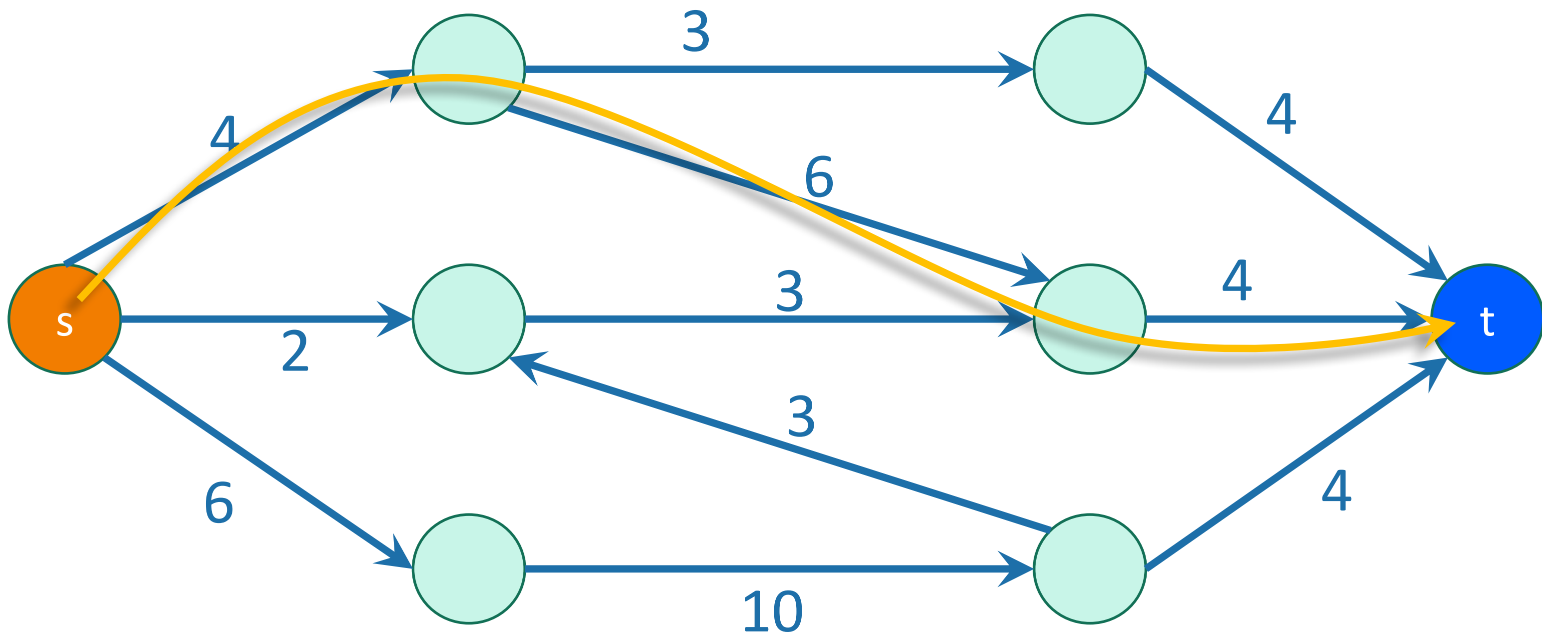
Example of Ford-Fulkerson



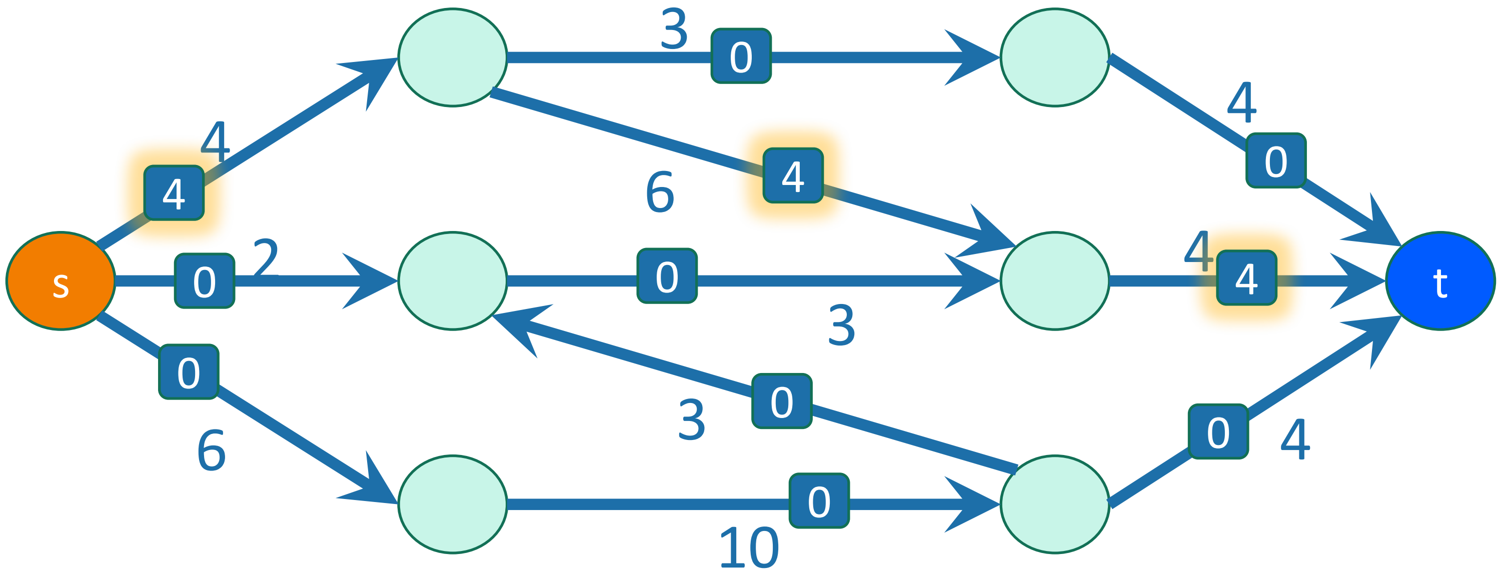


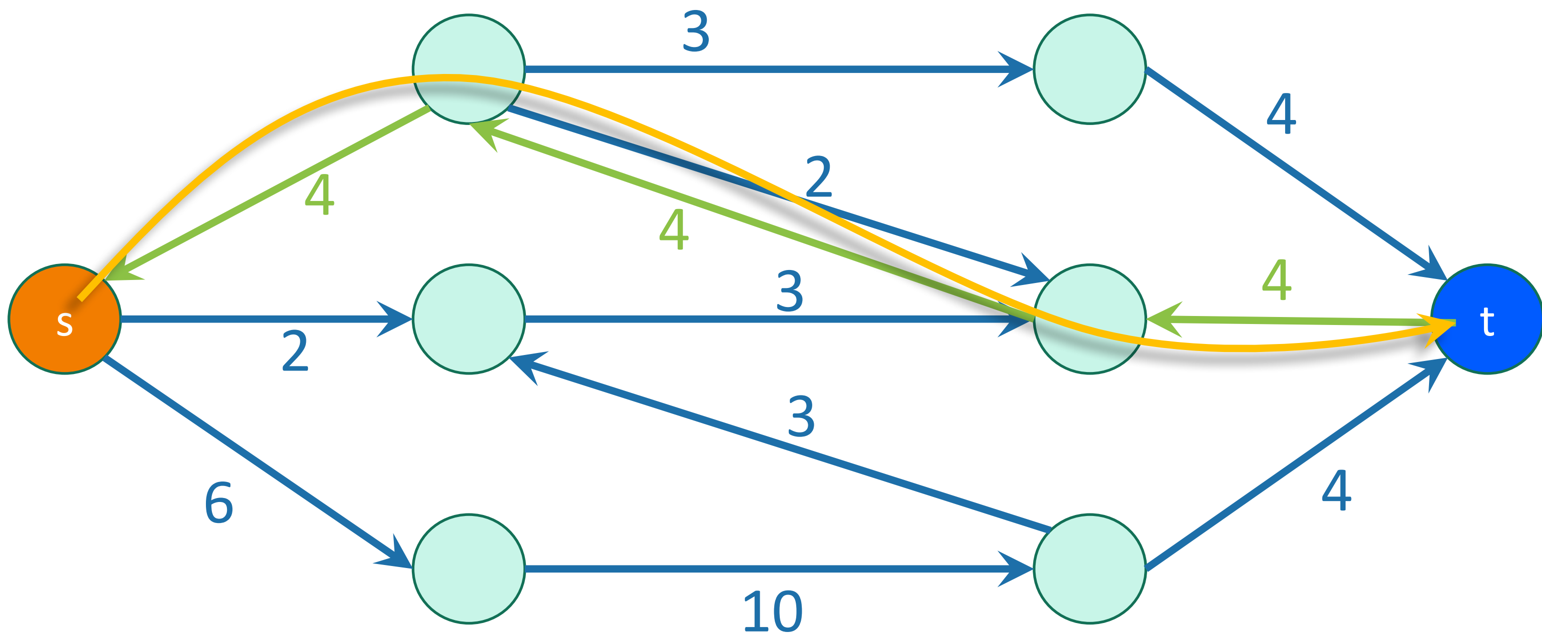
Example of Ford-Fulkerson



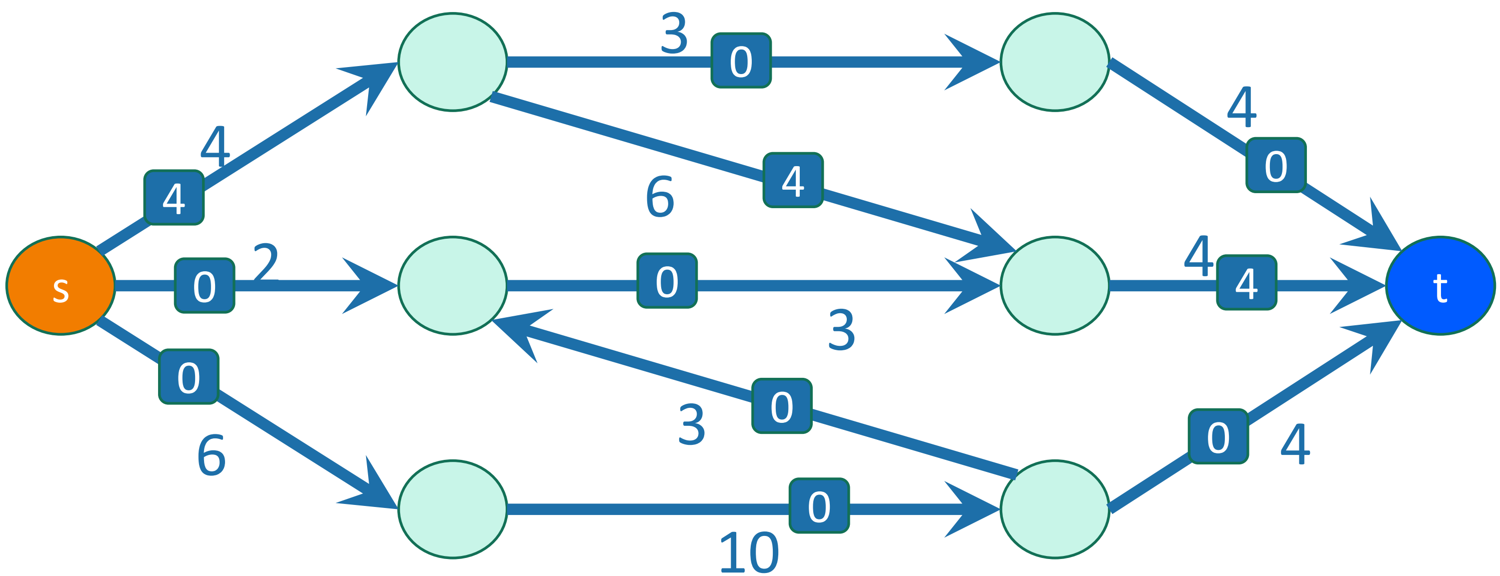


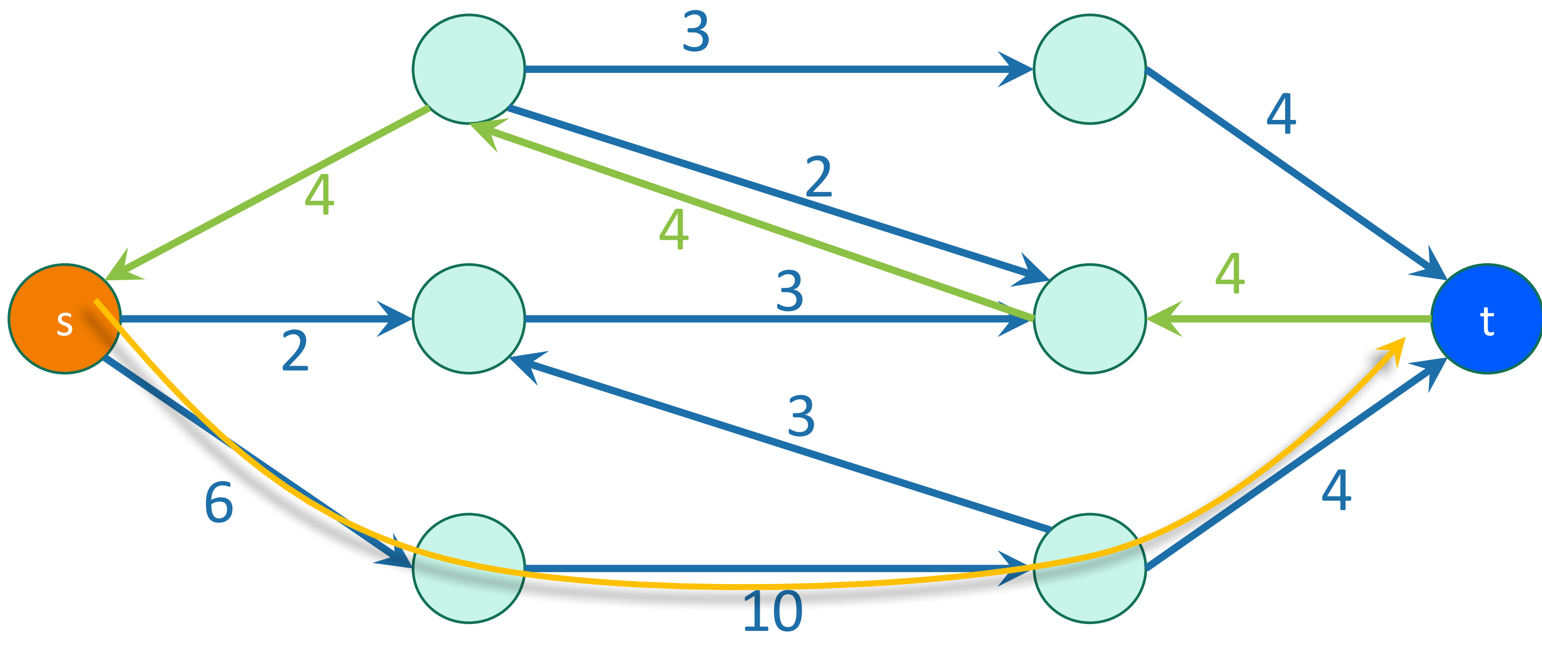
Example of Ford-Fulkerson



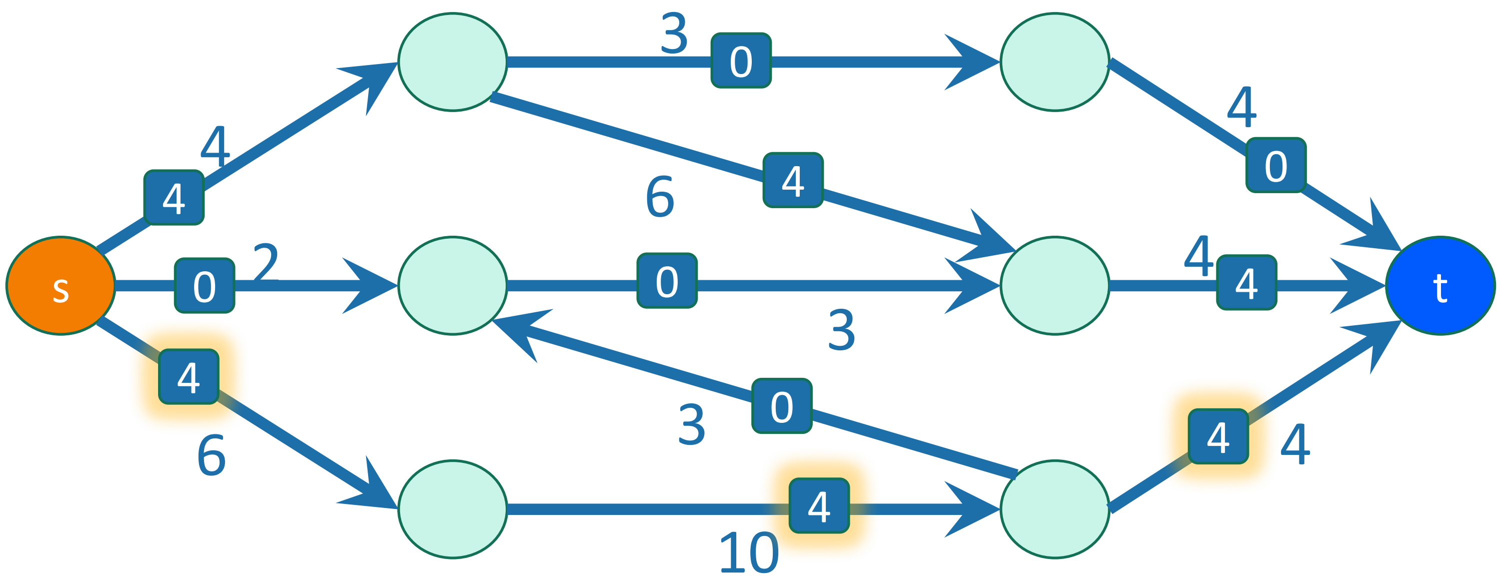


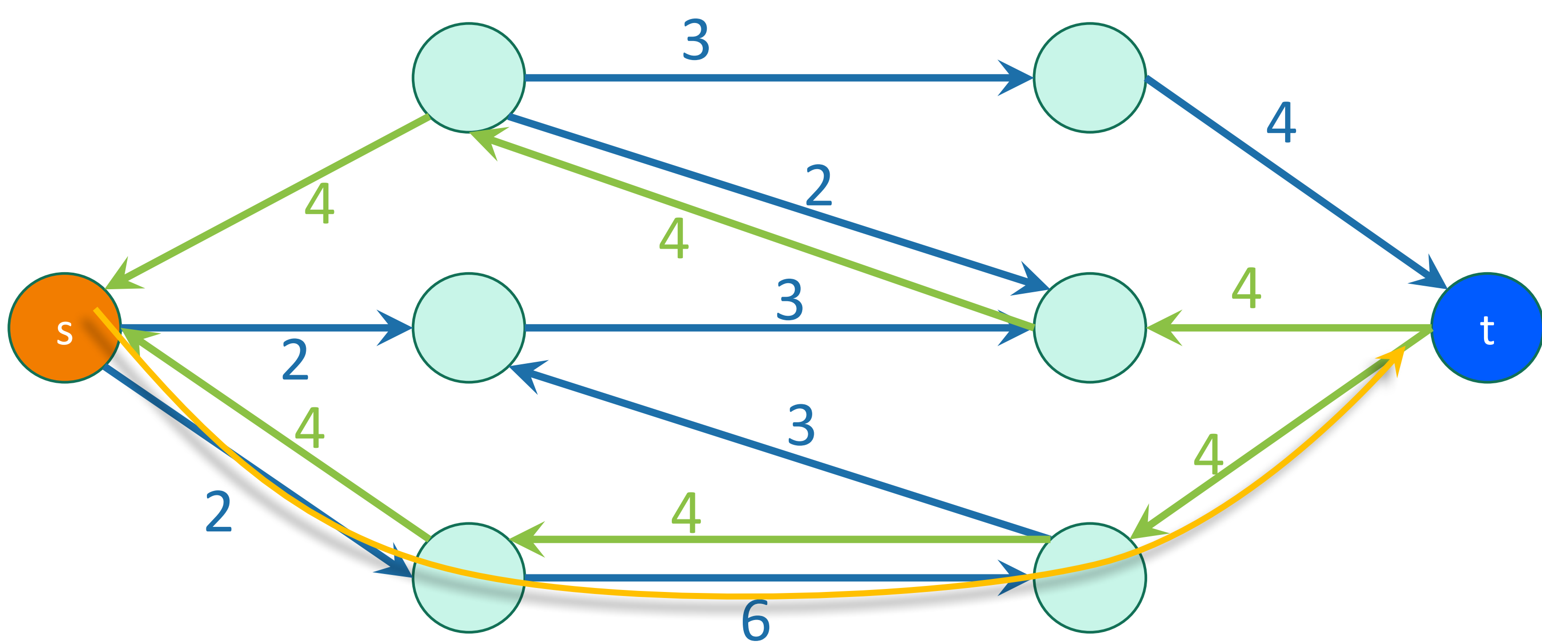
Example of Ford-Fulkerson





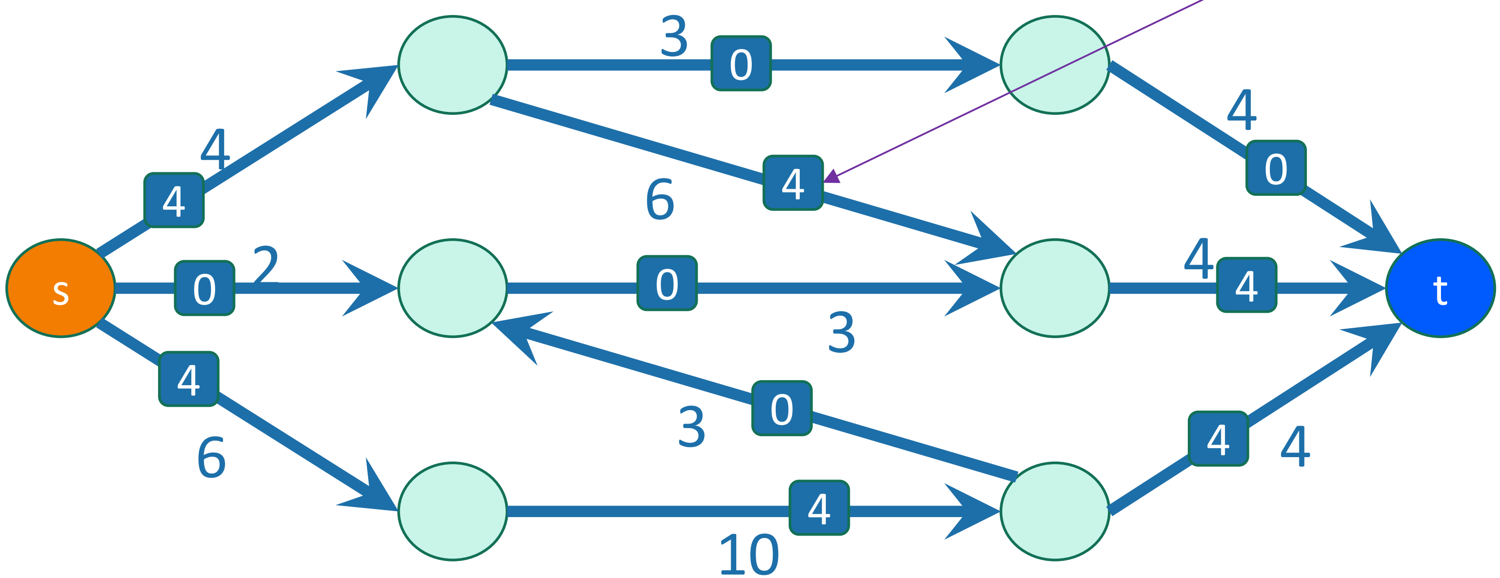
Example of Ford-Fulkerson



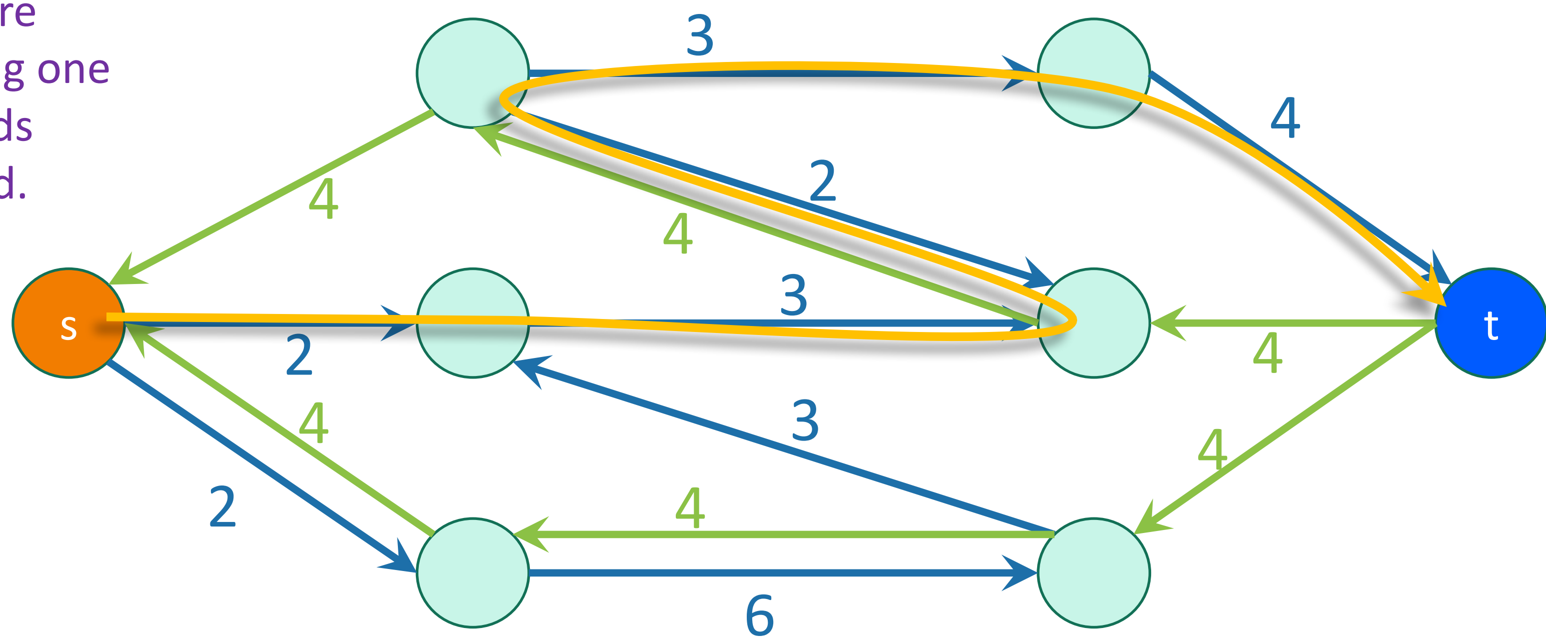


Example of Ford-Fulkerson

We will **remove** flow from this edge.

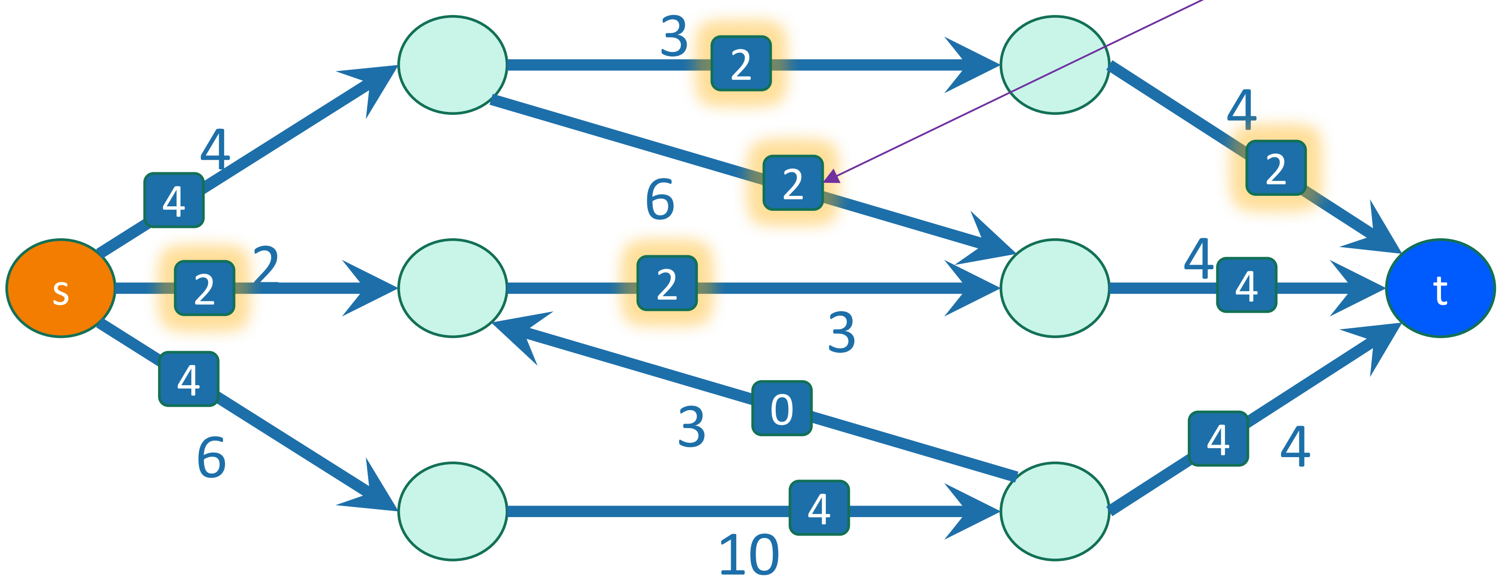


Notice that we're going back along one of the backwards edges we added.

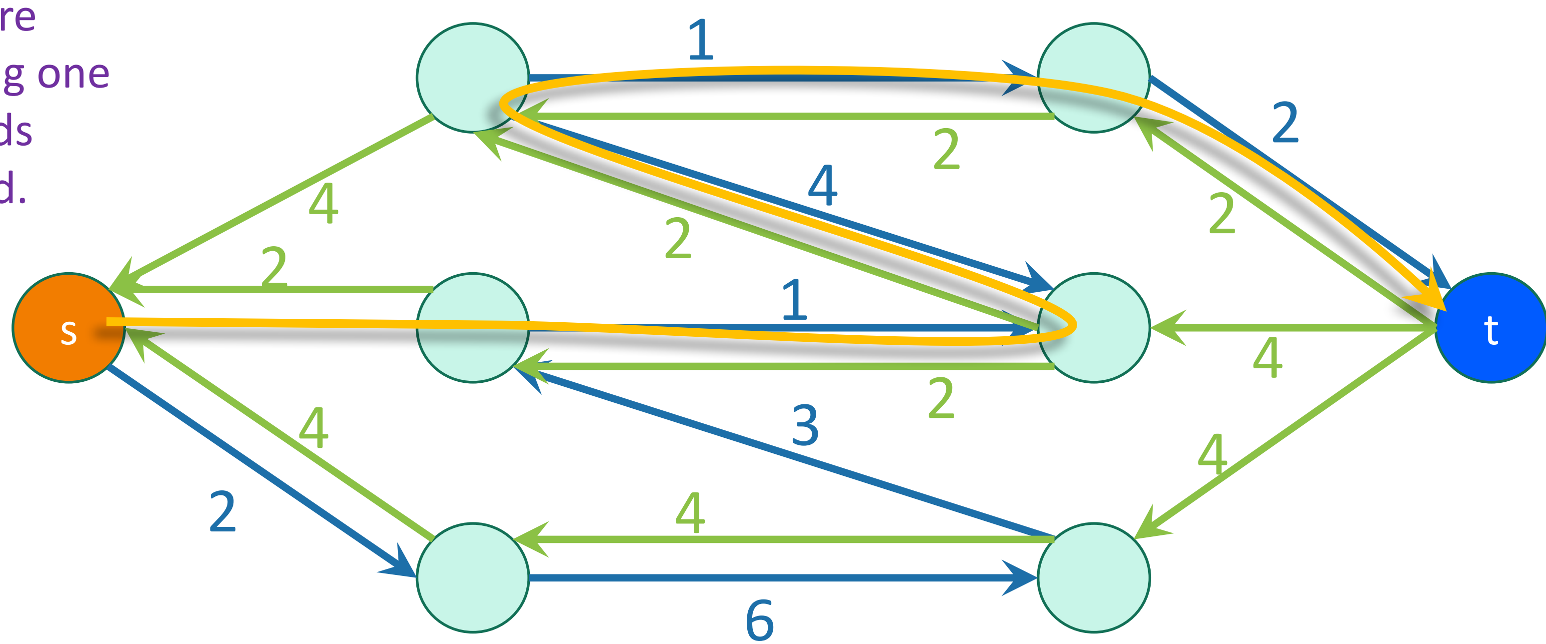


Example of Ford-Fulkerson

We will **remove** flow from this edge.

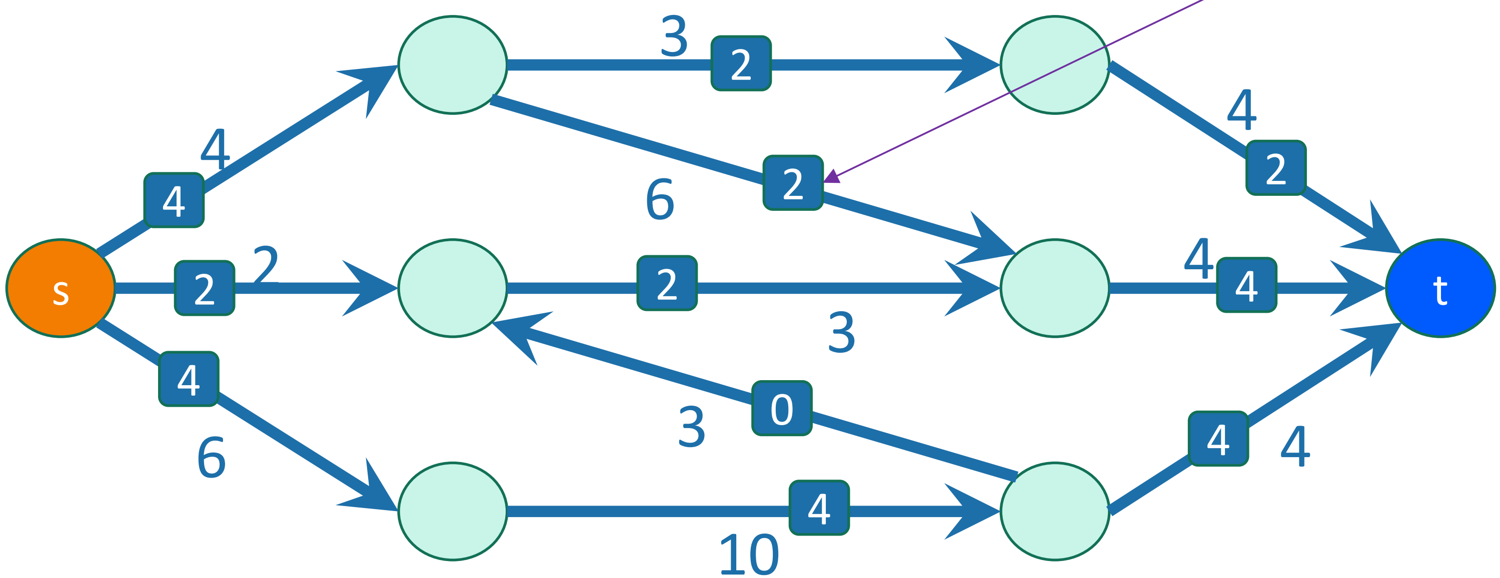


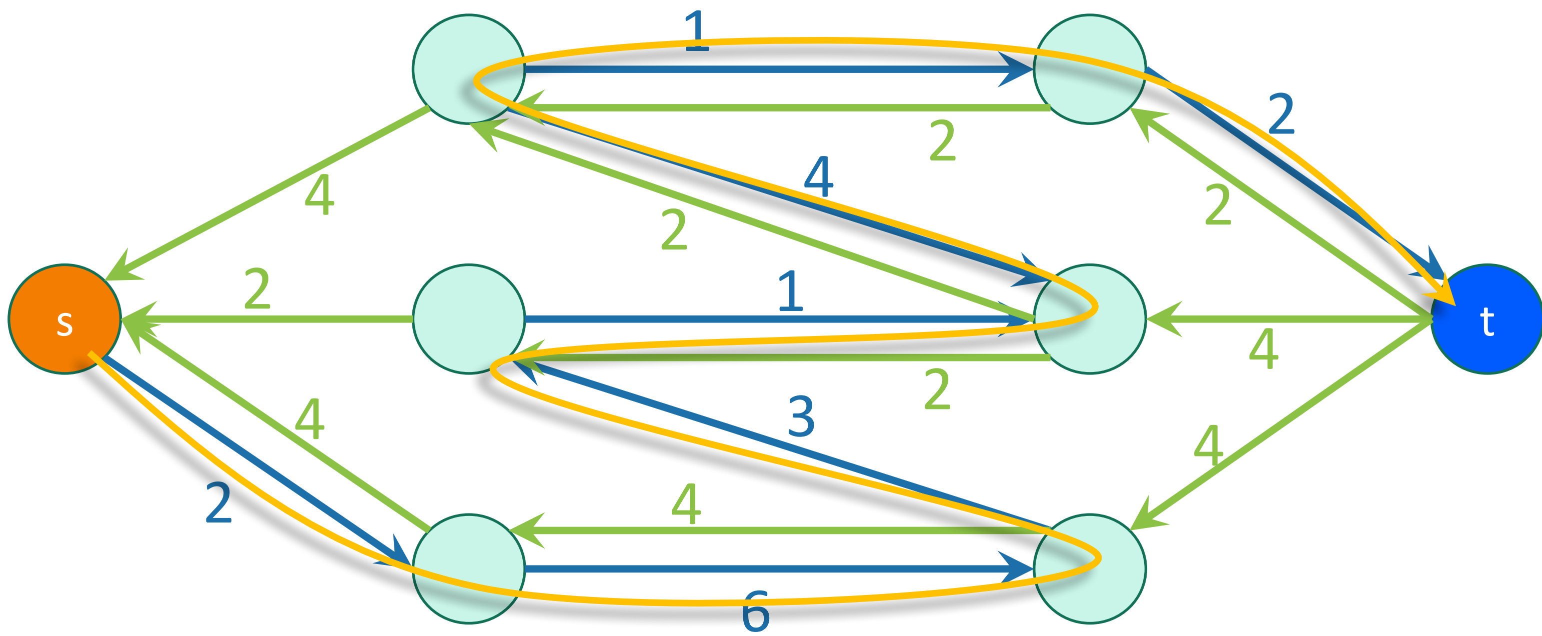
Notice that we're going back along one of the backwards edges we added.



Example of Ford-Fulkerson

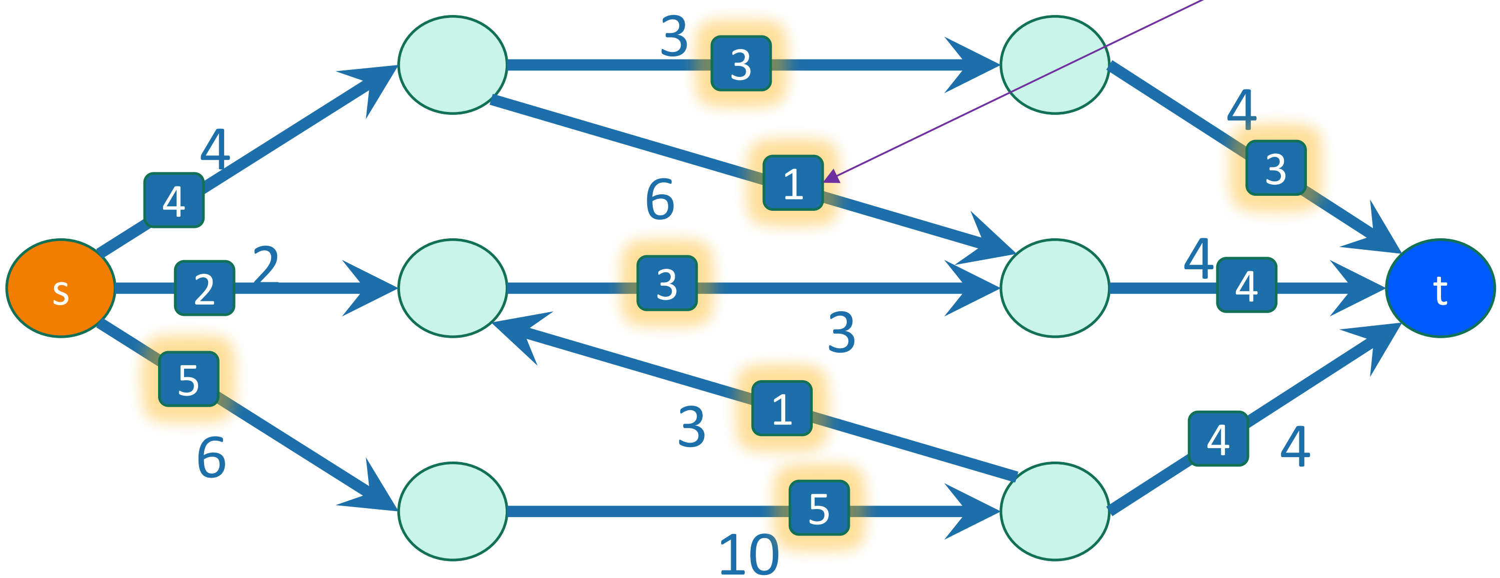
We will remove flow from this edge AGAIN.

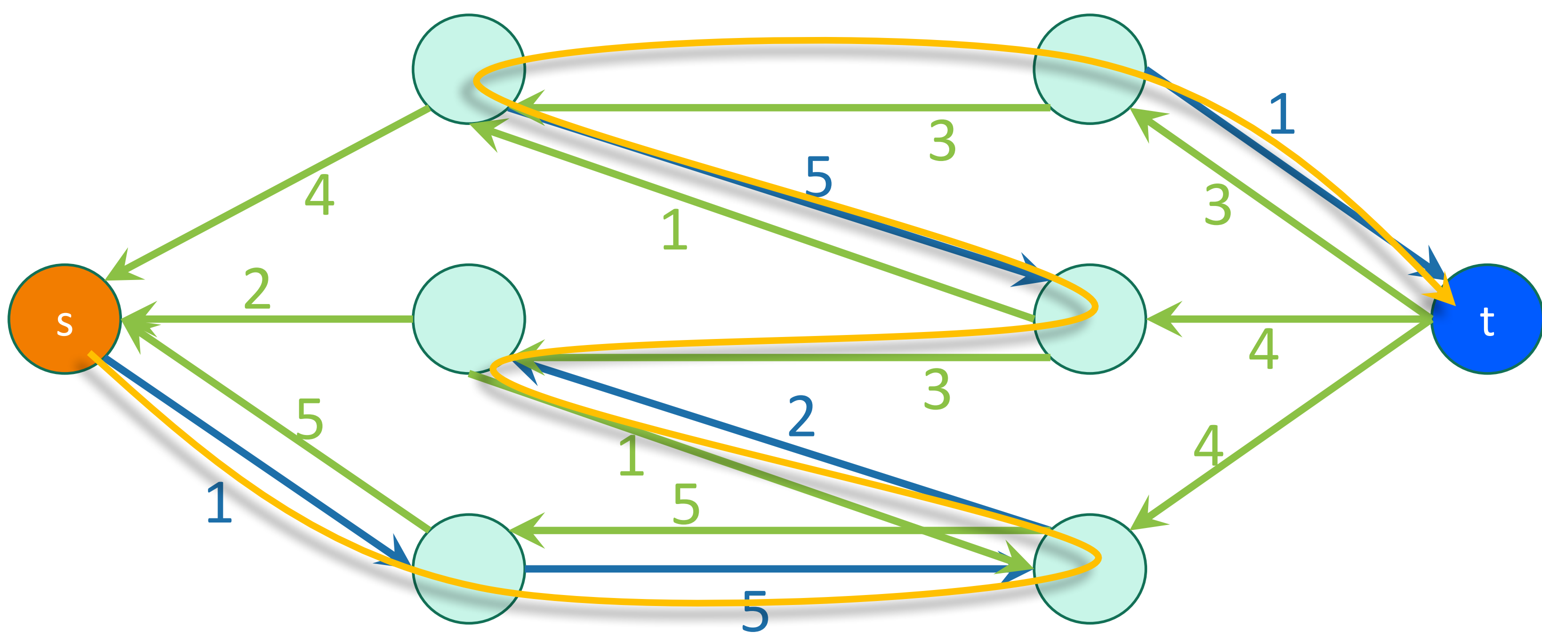




Example of Ford-Fulkerson

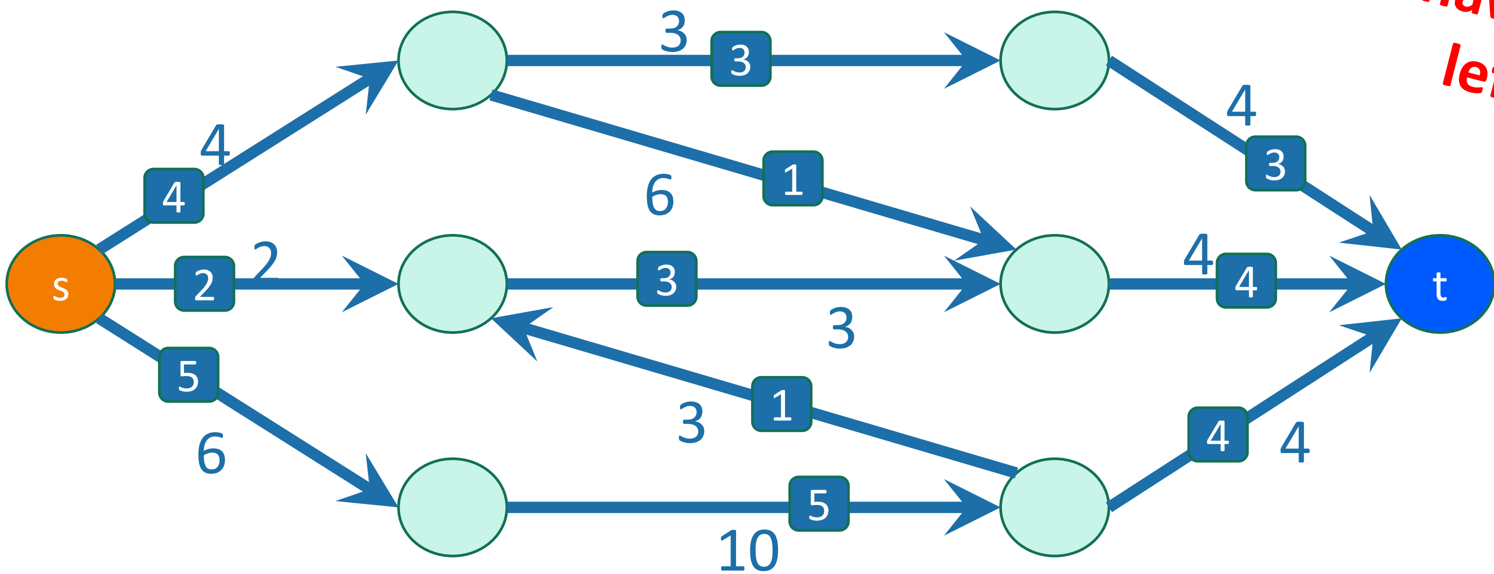
We will remove flow from this edge AGAIN.

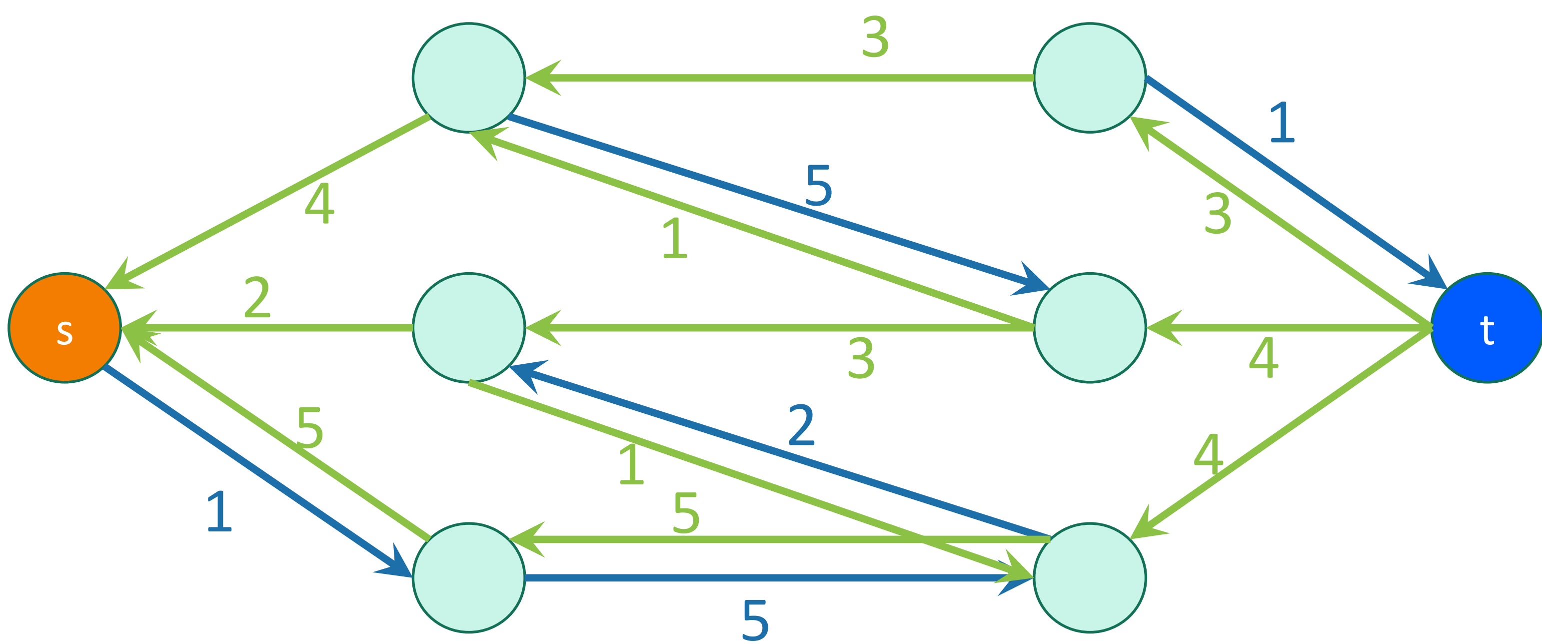




Example of Ford-Fulkerson

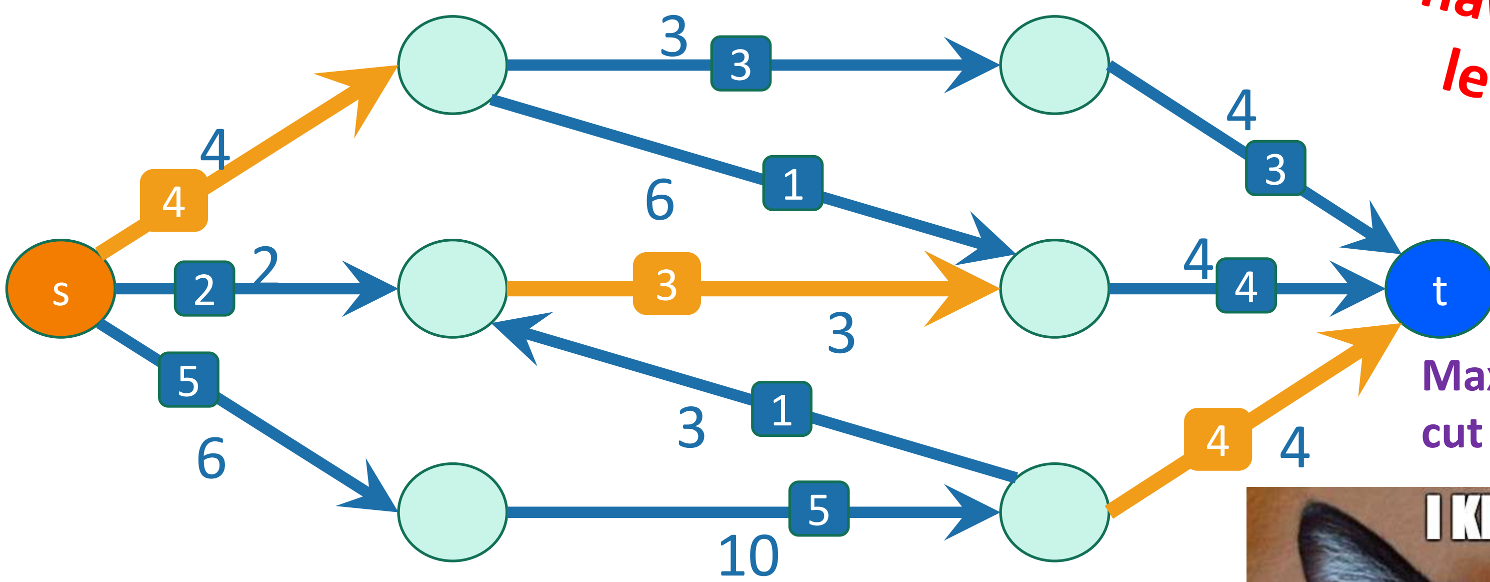
Now we have nothing left to do!



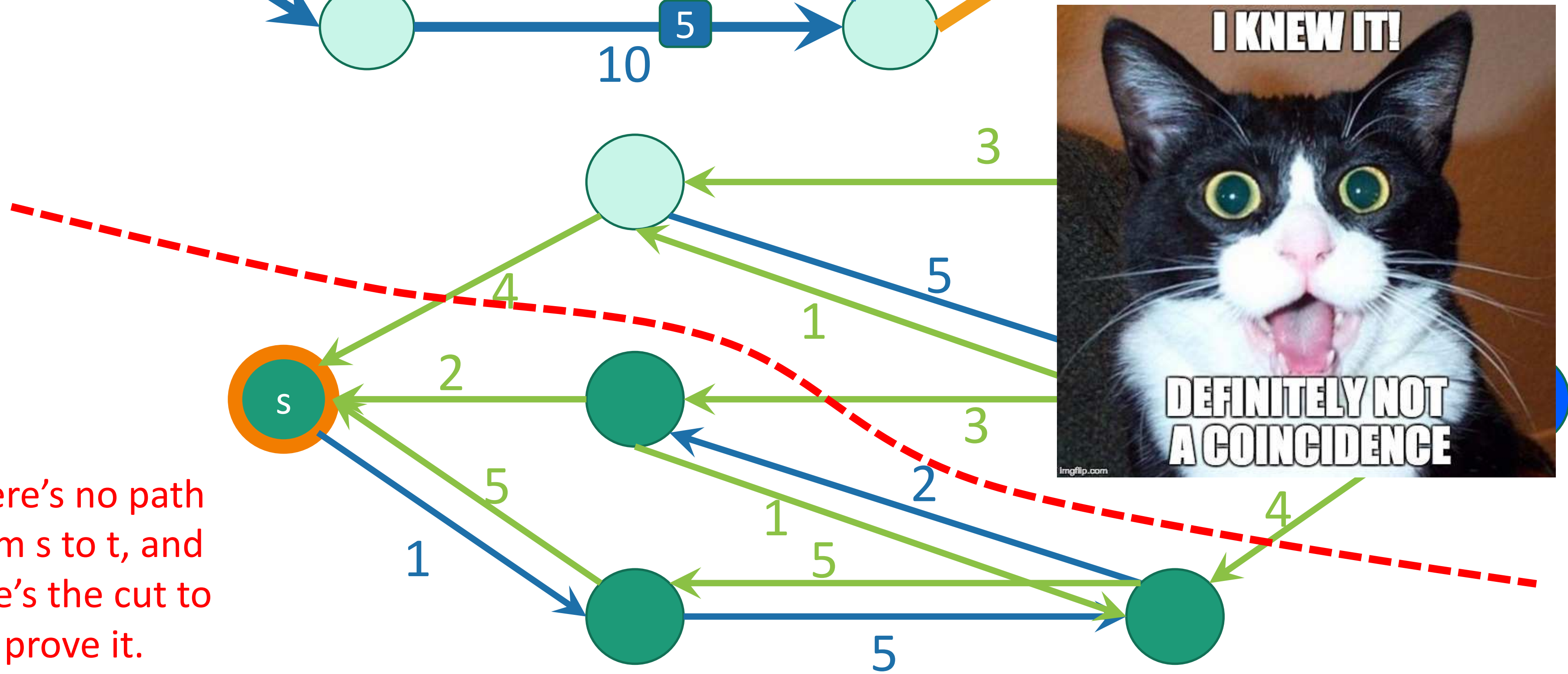


Example of Ford-Fulkerson

Now we have nothing left to do!



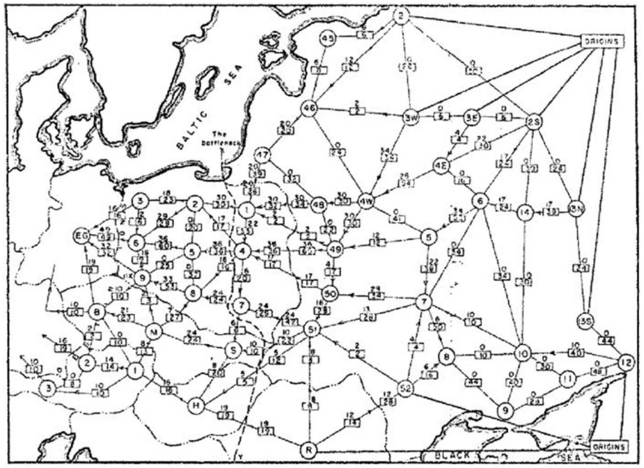
Max flow and min cut are both 11.



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

What have we learned?

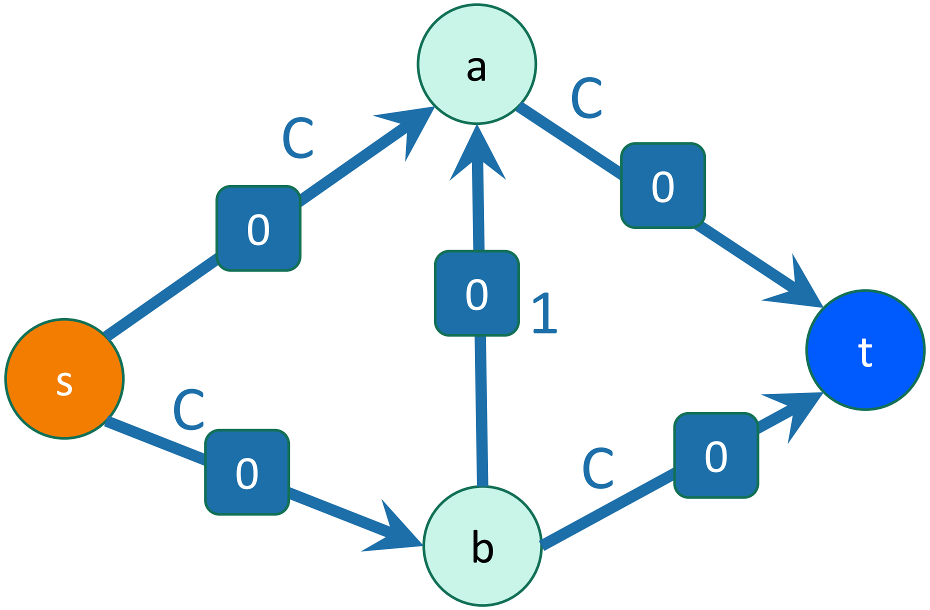
- Max s-t flow is equal to min s-t cut!
  - The USSR and the USA were trying to solve the same problem…
- The Ford-Fulkerson algorithm can find the min-cut/max-flow.
  - Repeatedly improve your flow along an augmenting path.
- **How long does this take???**

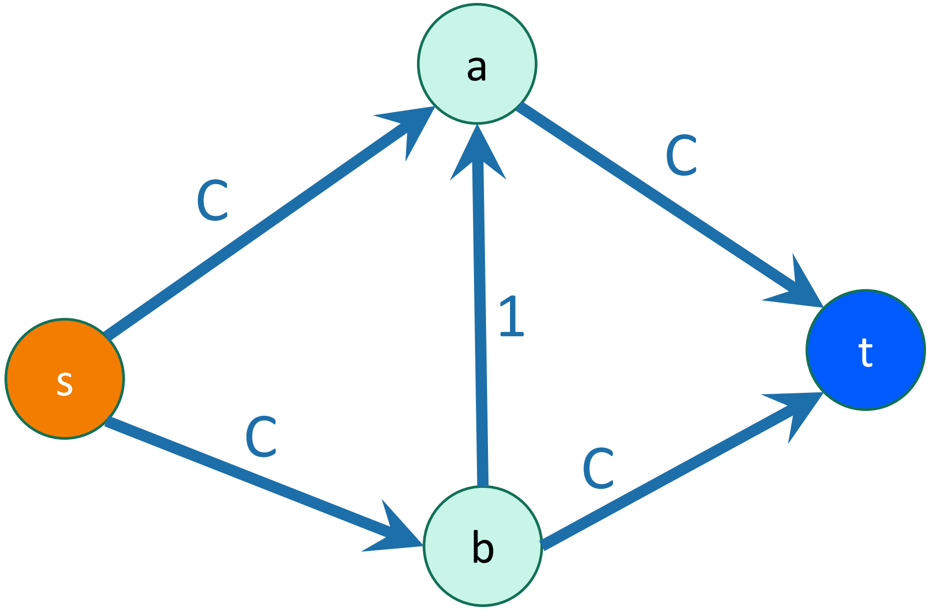


Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C.

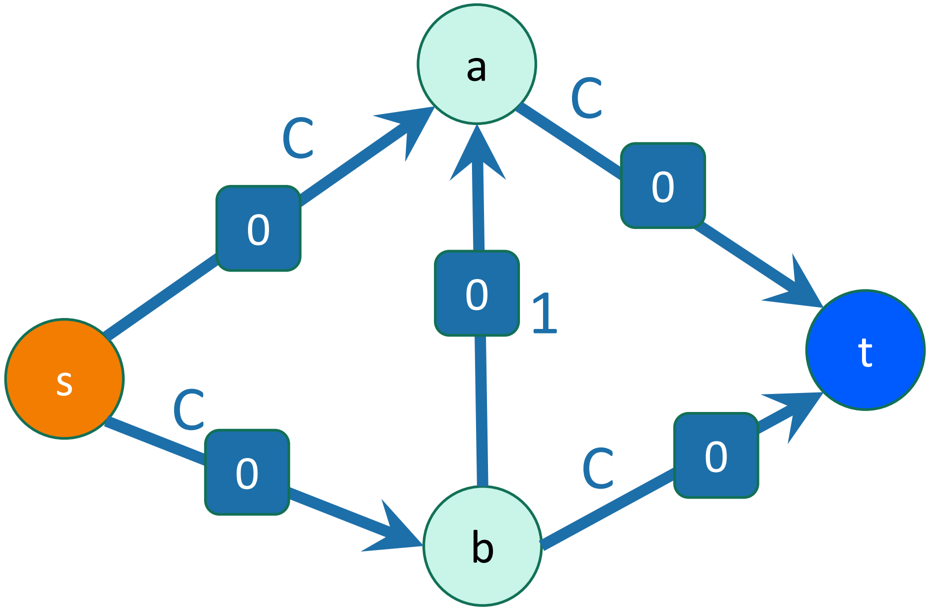


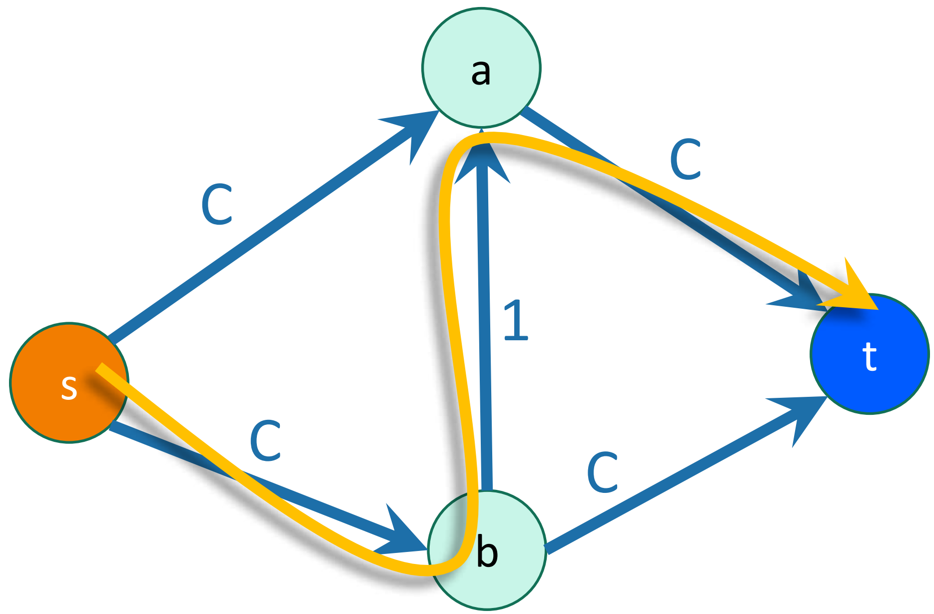


Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C.

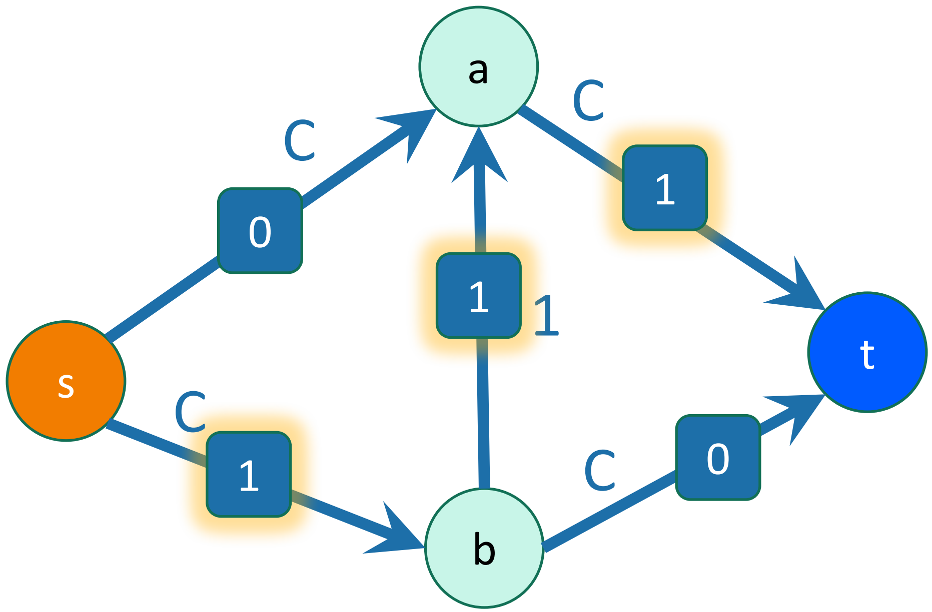




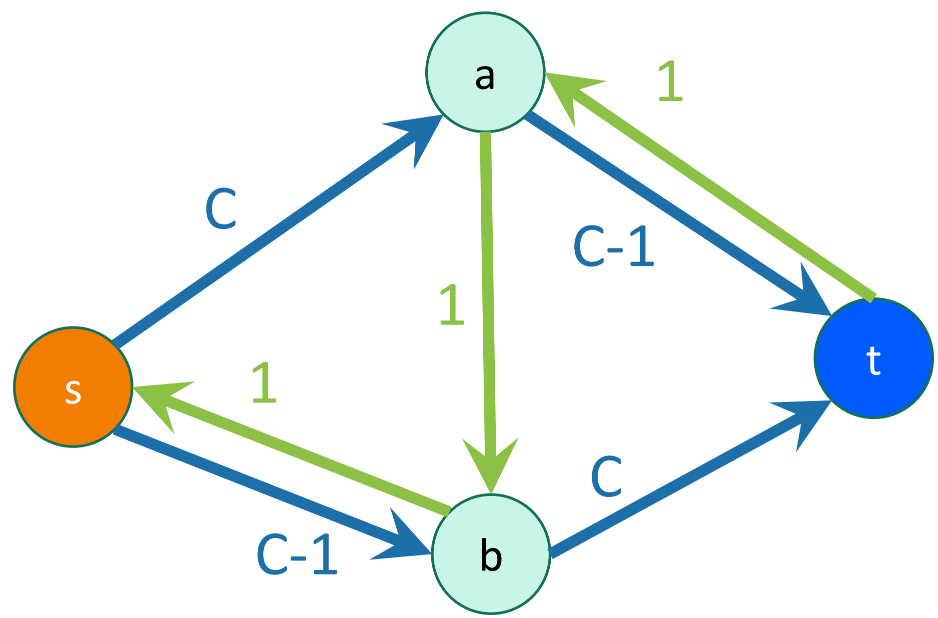
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really
big number C.



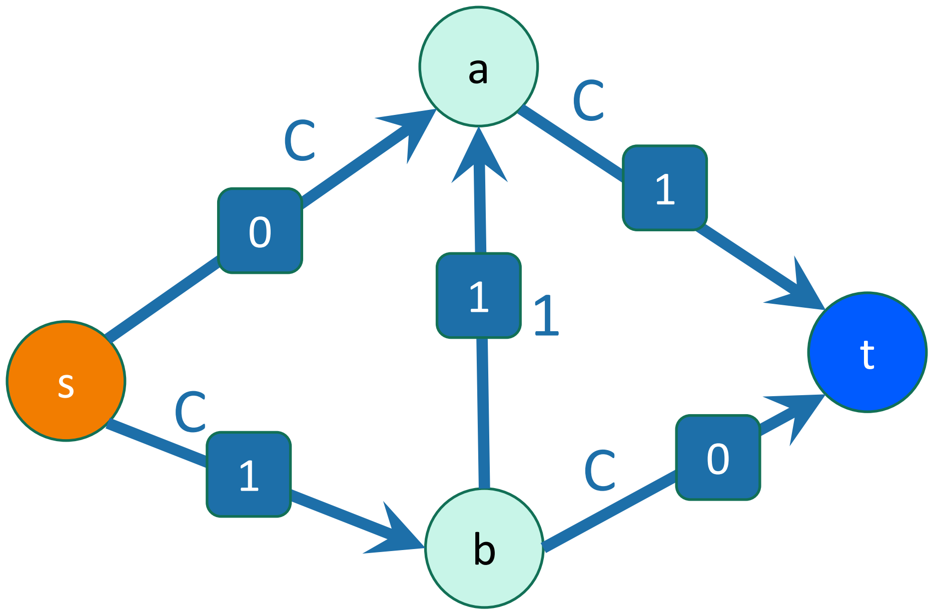
The edge (b,a) disappeared from the residual graph!

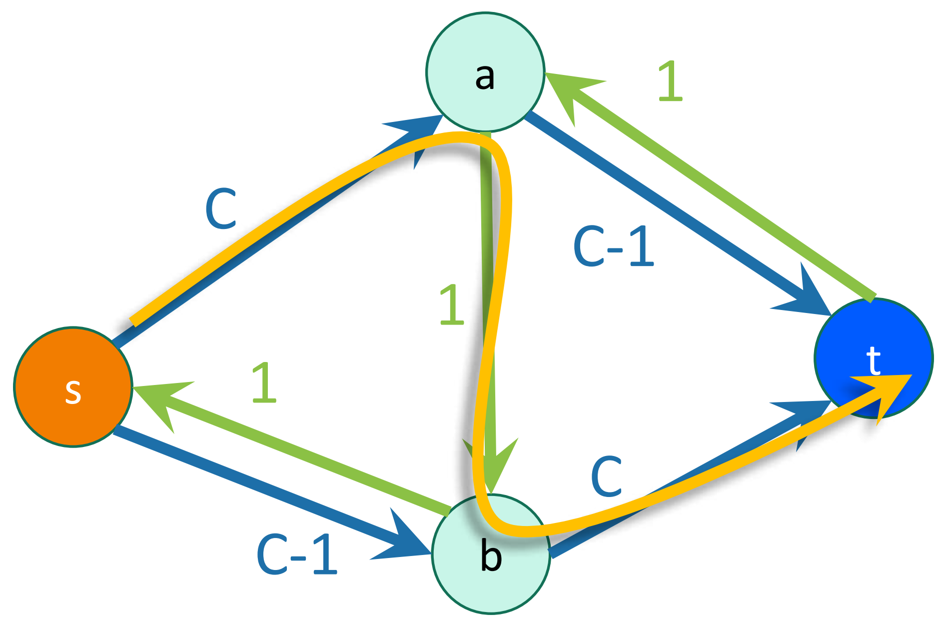


Why should we be concerned?

Suppose we just picked paths arbitrarily.

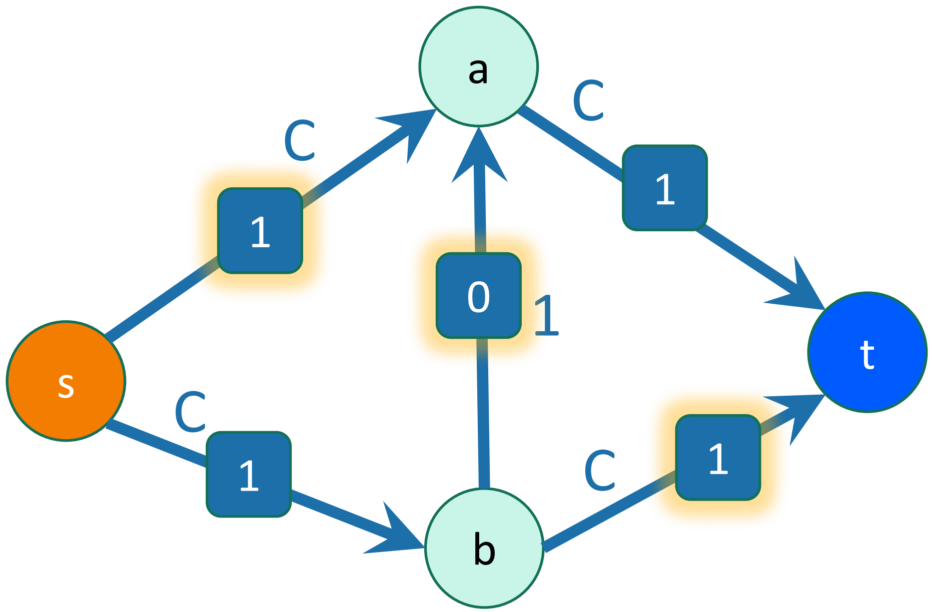
Choose a really big number C.



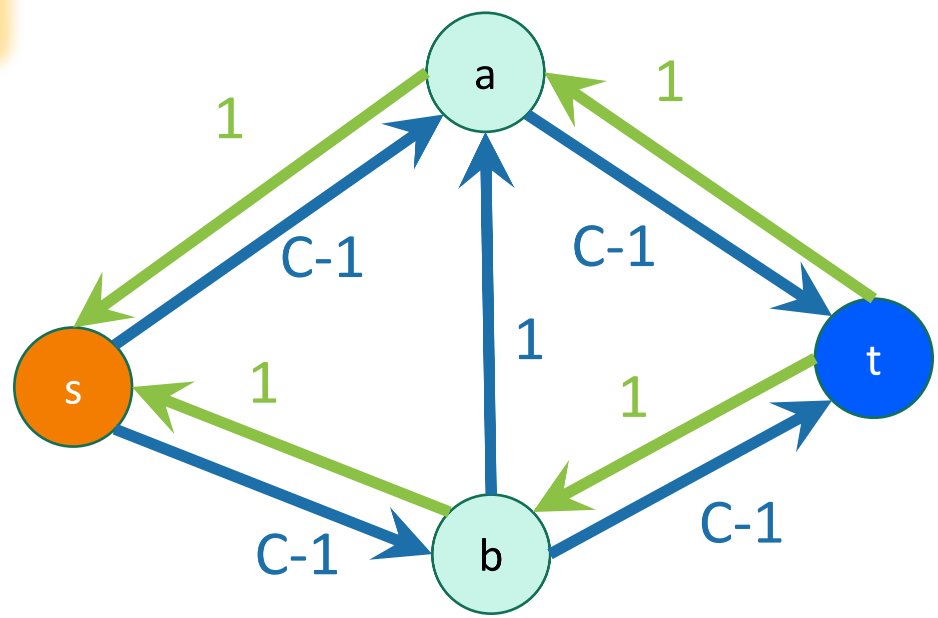


Why should we be concerned?

Suppose we just picked paths arbitrarily.



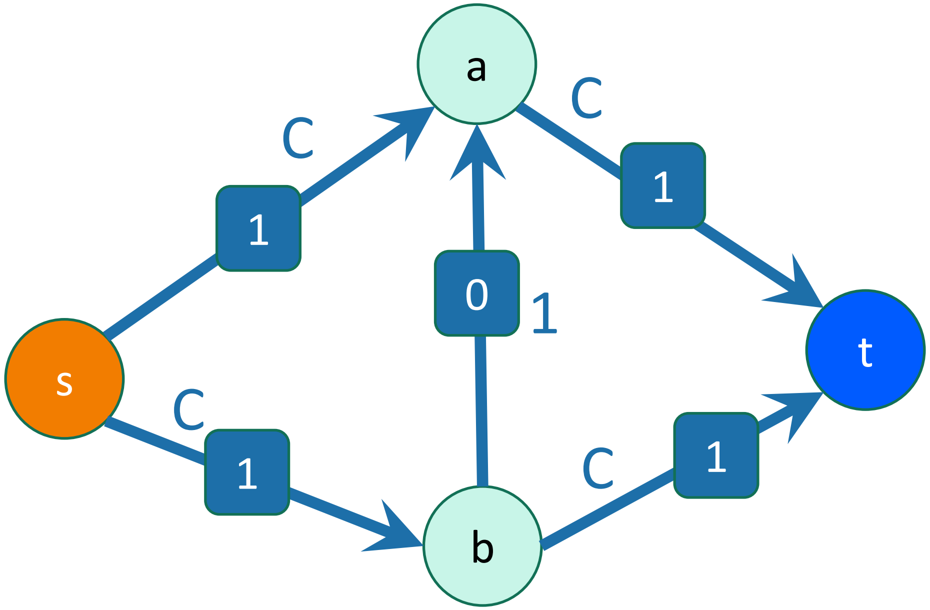
Choose a really big number C.



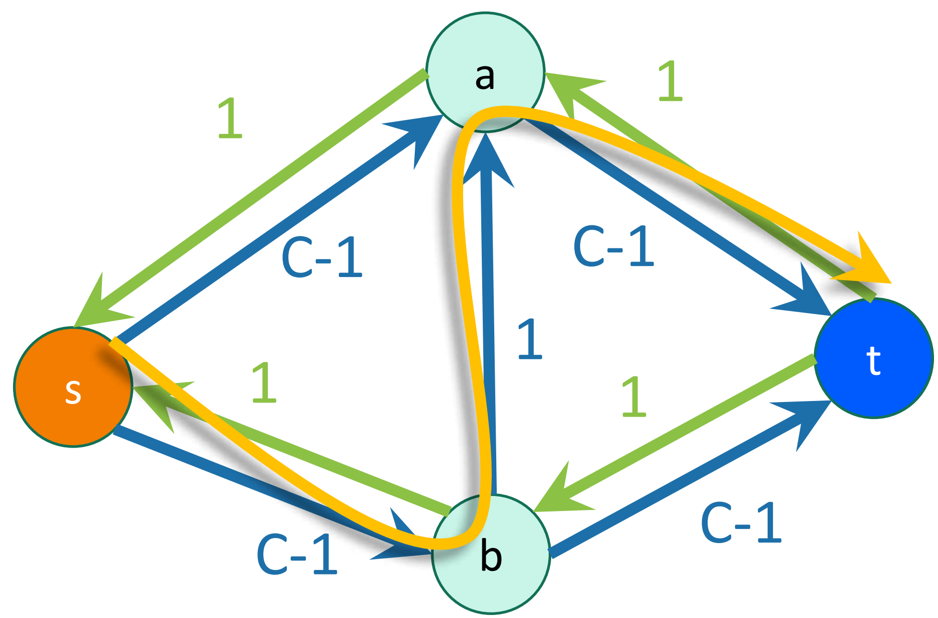
The edge (b,a) re-appeared in the residual graph!

Why should we be concerned?

Suppose we just picked paths arbitrarily.



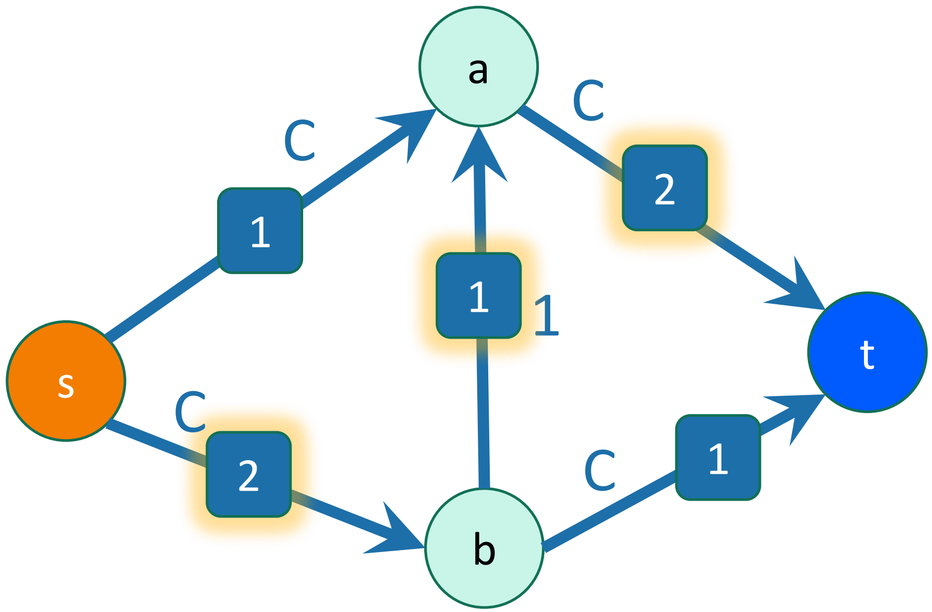
Choose a really big number C.

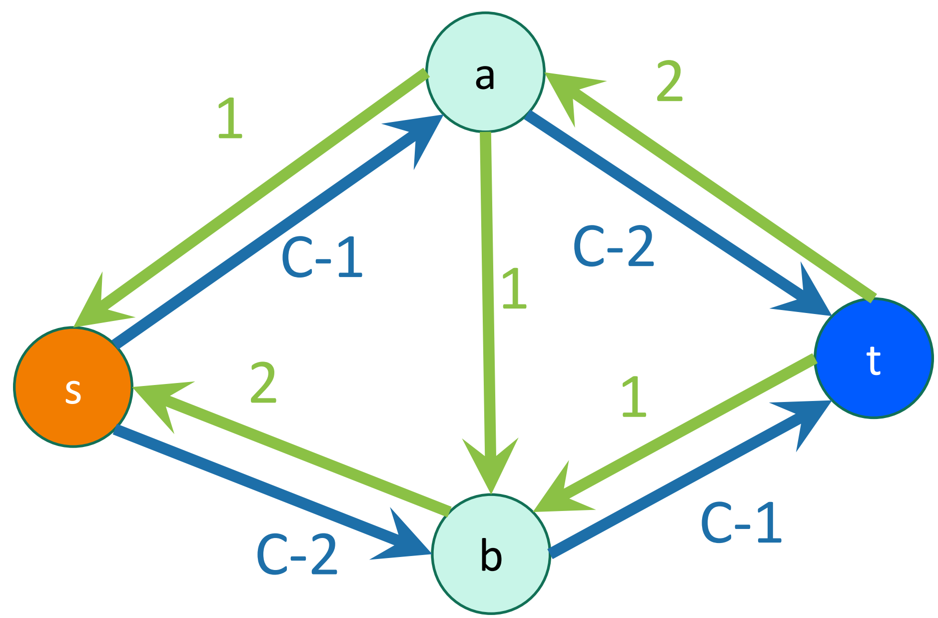


Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C.



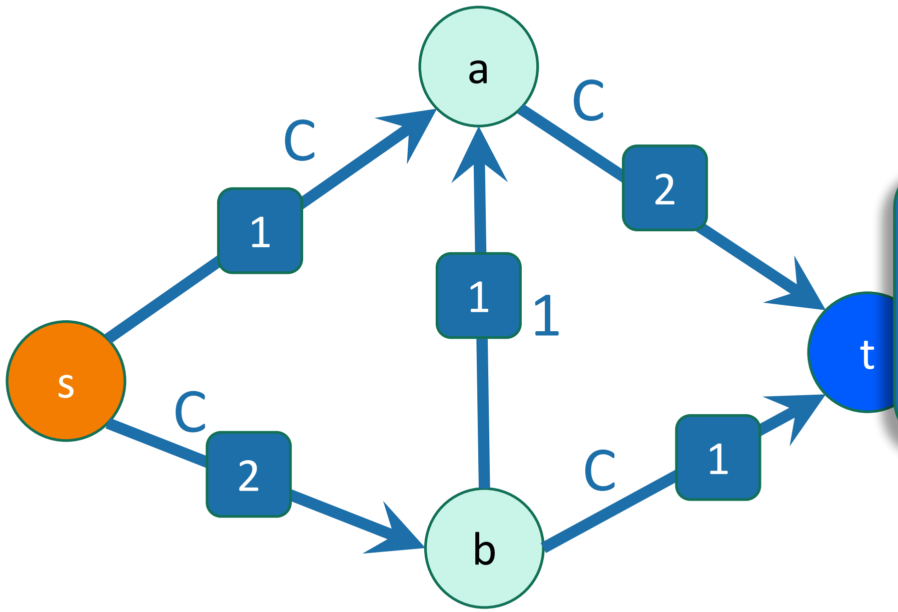


The edge (b,a) disappeared from the residual graph!

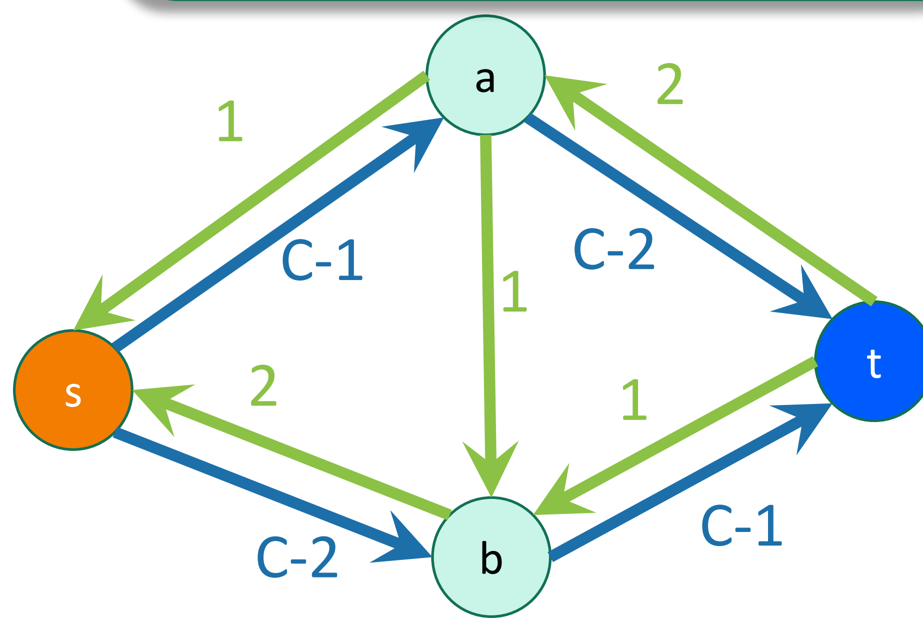
Why should we be concerned?

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!

Doing Ford-Fulkerson with BFS is called the **Edmonds-Karp algorithm**.

Theorem

- If you use BFS, the Ford-Fulkerson algorithm runs in time **$O(nm^2)$.**

Doesn't have anything to do with the edge weights!

- We will skip the proof in class.
  - You can check it out in the notes if you are interested.
  - It will **not** be on the exam.
- Basic idea:
  - The number of times you remove an edge from the residual graph is O(n).
    - This is the hard part
  - There are at most m edges.
  - Each time we remove an edge we run BFS, which takes time O(n+m).
    - Actually, O(m), since we don't need to explore the whole graph, just the stuff reachable from s.

One more useful thing

- 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 integral.

But wait, there's more!

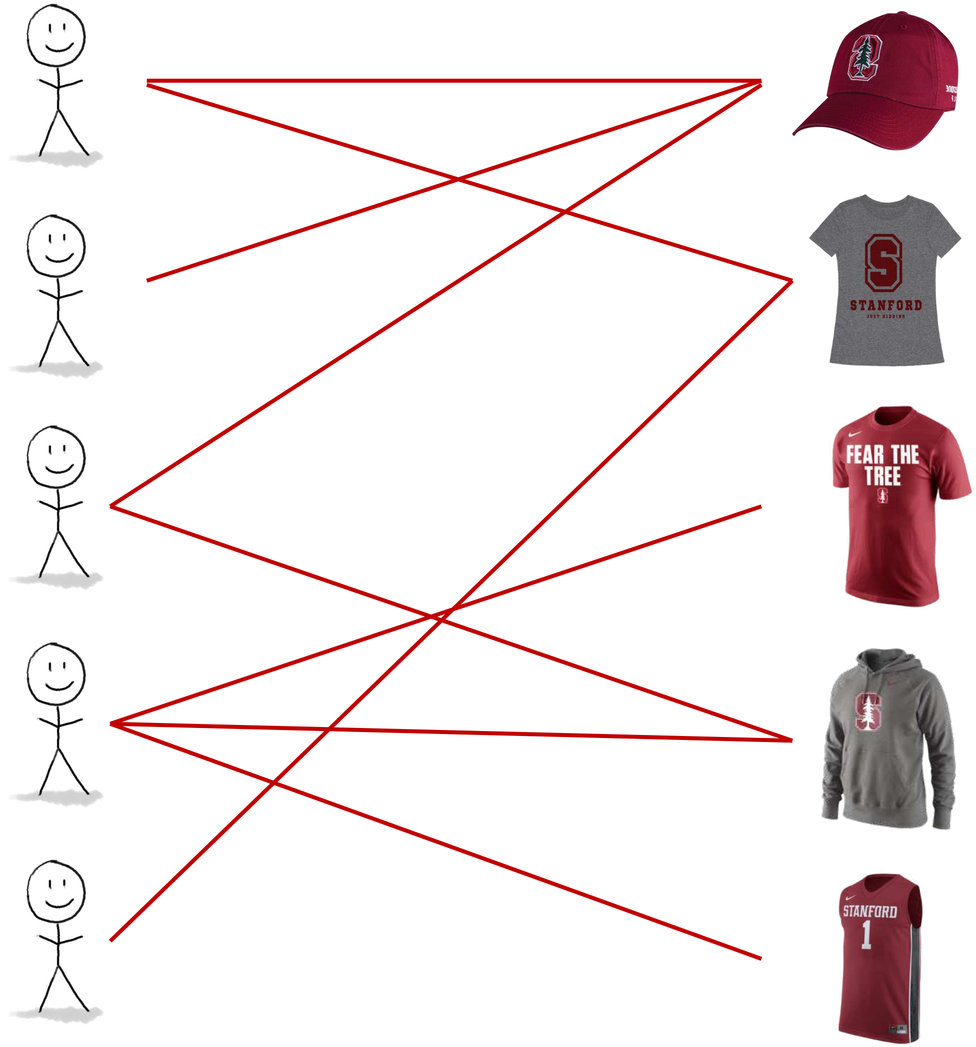
- Min-cut and max-flow are not just useful for the USA and the USSR in 1955.
  - An important algorithmic primitive!
- The Ford-Fulkerson algorithm is the basis for many other graph algorithms.
- For the rest of today, we'll see a few:
  - Maximum bipartite matching
  - Integer assignment problems

Some of the following material shamelessly stolen from Jeff Erickson's excellent lecture notes:
http://jeffe.cs.illinois.edu/teaching/algorithms/2009/notes/17-maxflowapps.pdf

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?**

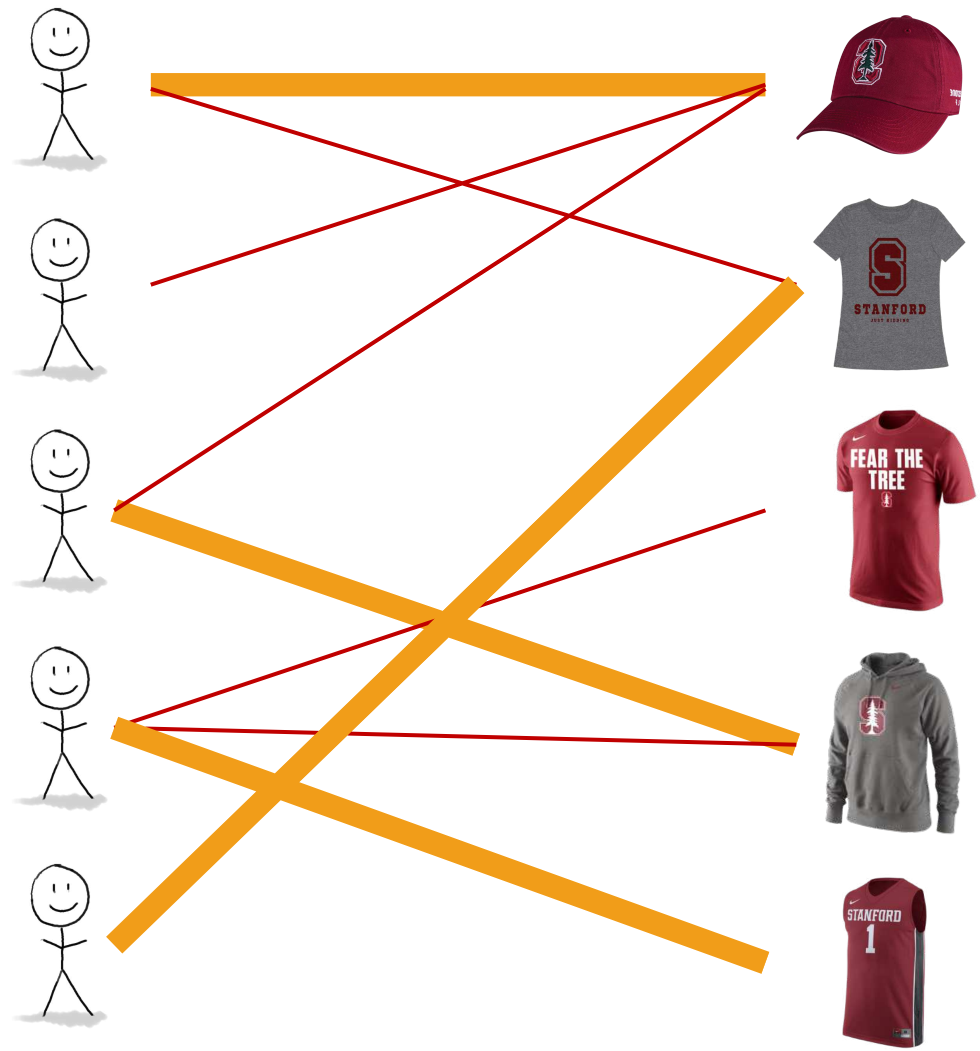


Stanford Students

Stanford Swag

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?**

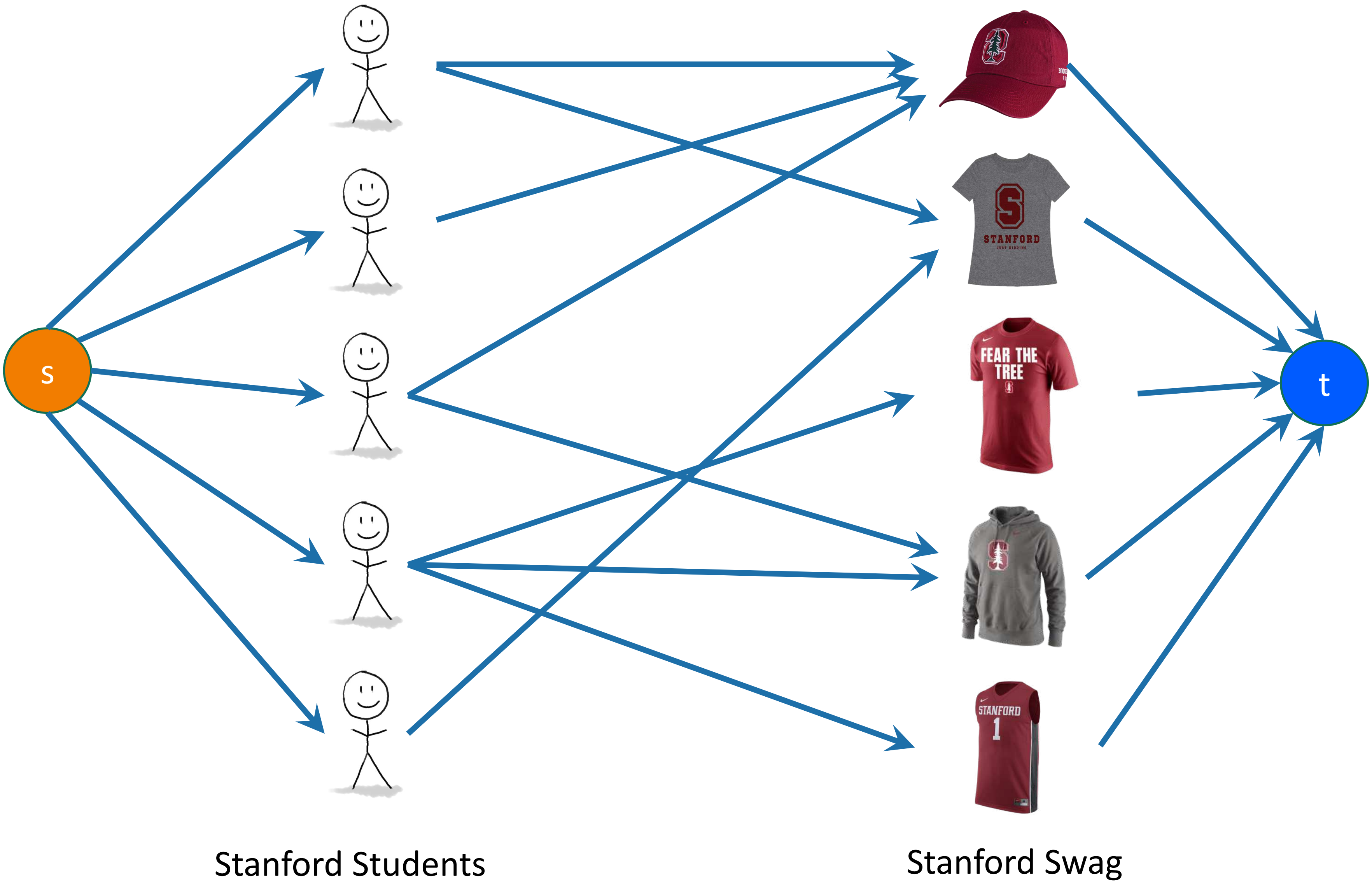


Stanford Students

Stanford Swag

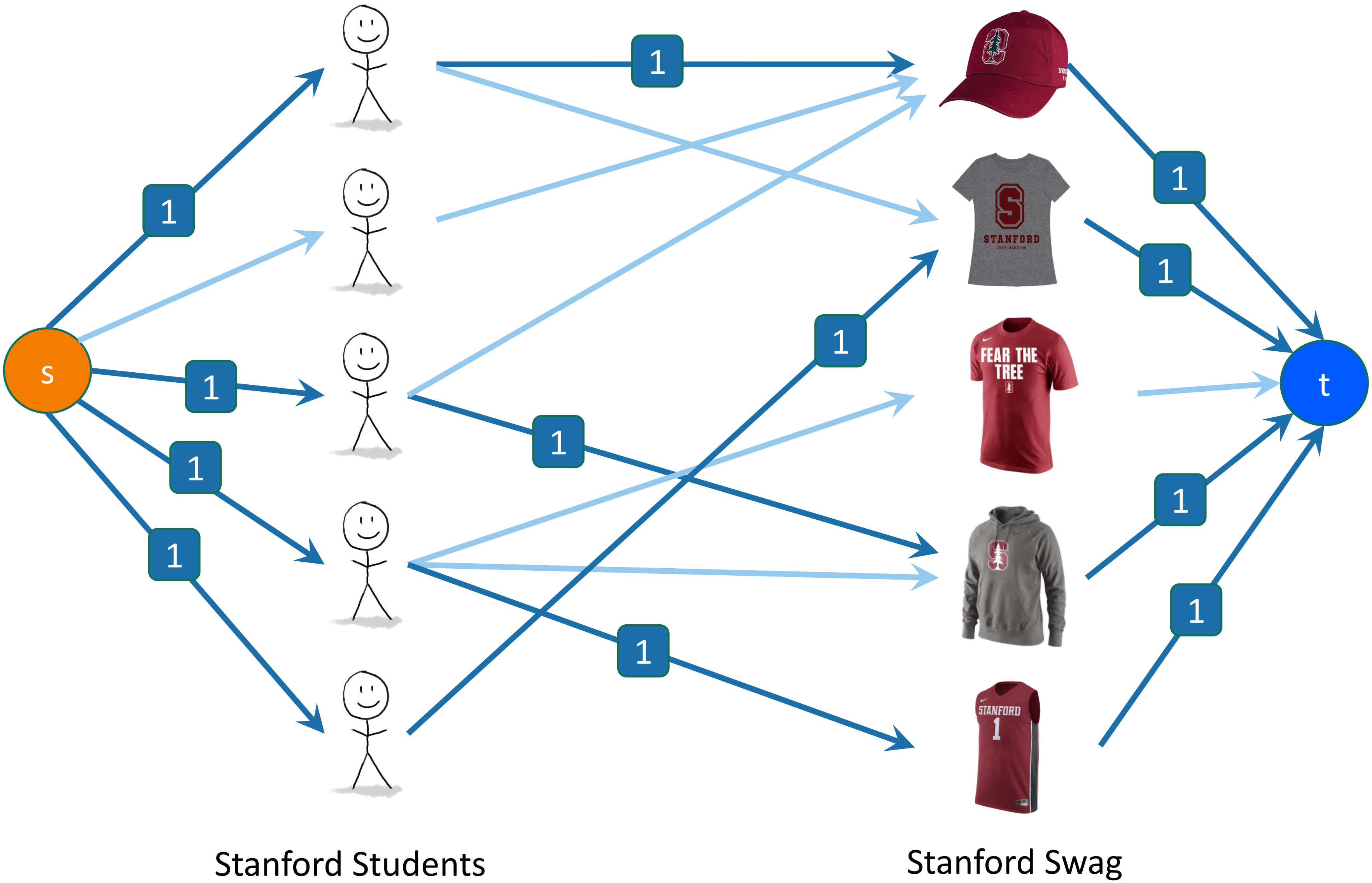
Solution via max flow

All edges have capacity 1.



Solution via max flow

All edges have capacity 1.



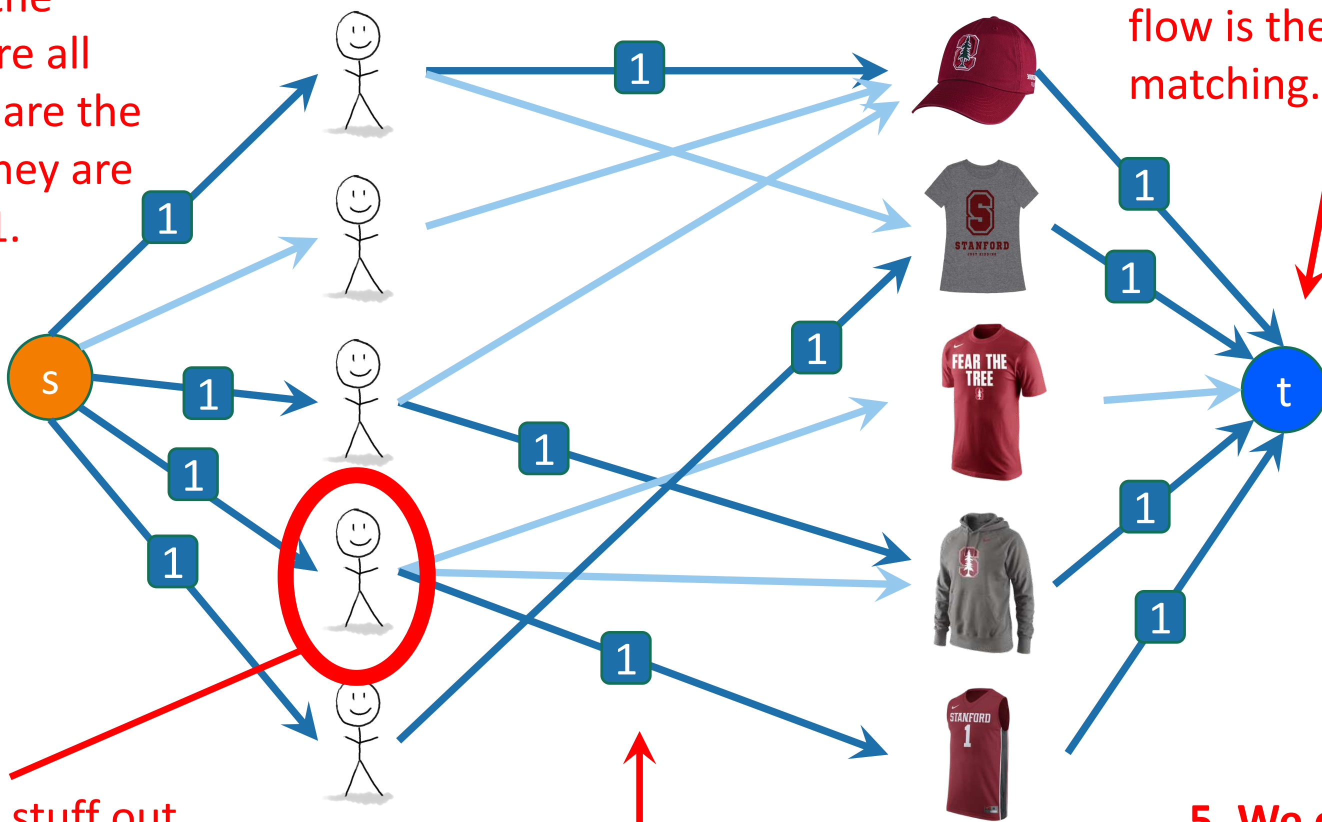
Solution via max flow

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.



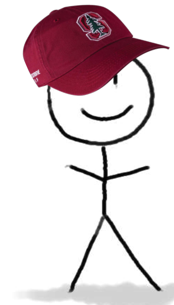
Value of this flow is 4.

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 maximal matching.

A slightly more complicated example:
assignment problems



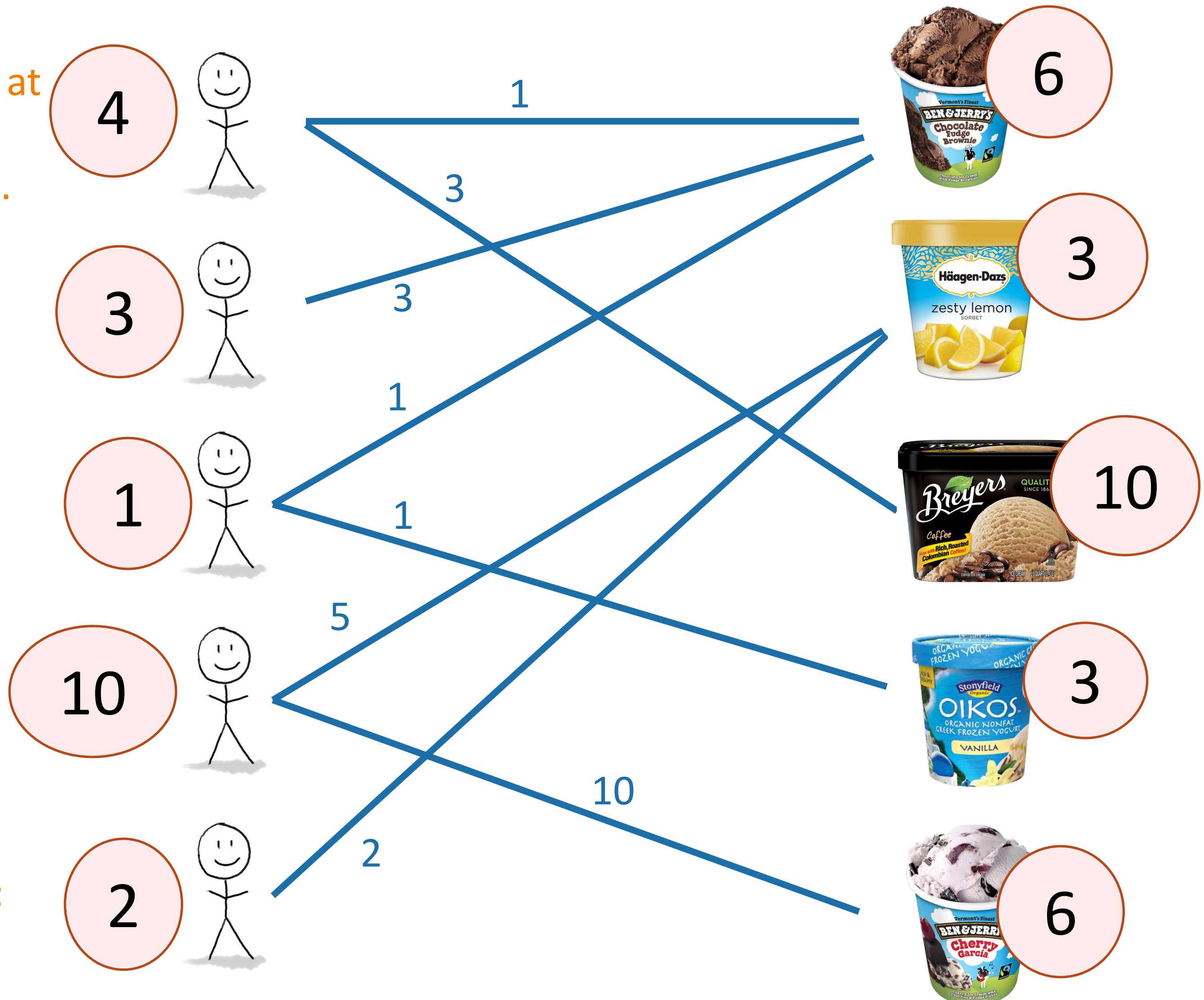
- One set X
  - Example: Stanford students
- Another set Y
  - Example: tubs of ice cream
- Each x in X can participate in c(x) matches.
  - Student x can only eat 4 scoops of ice cream.
- Each y in Y can only participate in c(y) matches.
  - Tub of ice cream y only has 10 scoops in it.
- Each pair (x,y) 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.**

Example

How can we serve as much ice cream as possible?

This person wants 4 scoops of ice cream, at most 1 of chocolate and at most 3 coffee.

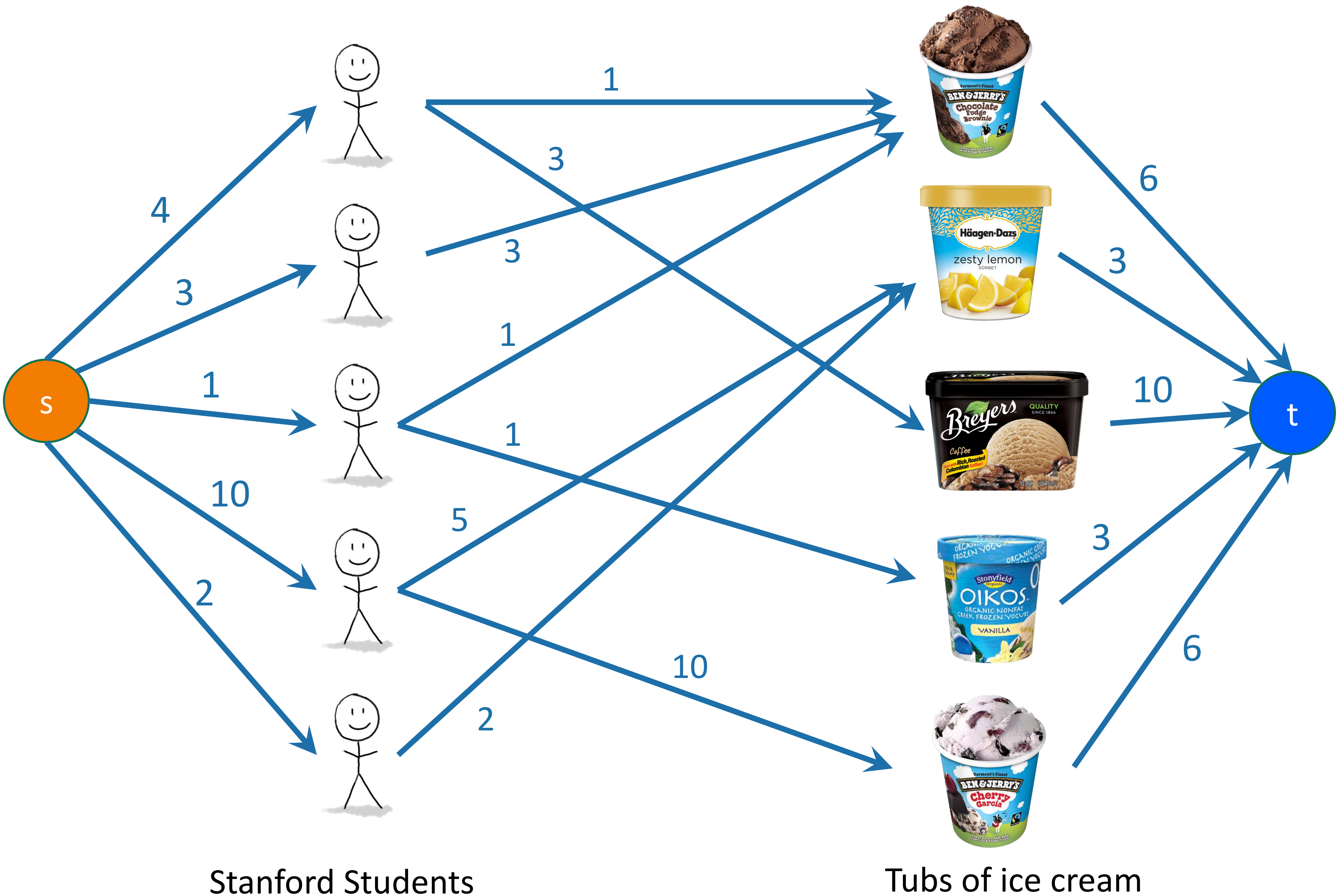
This person is vegan and not that hungry; they only want two scoops of the sorbet.



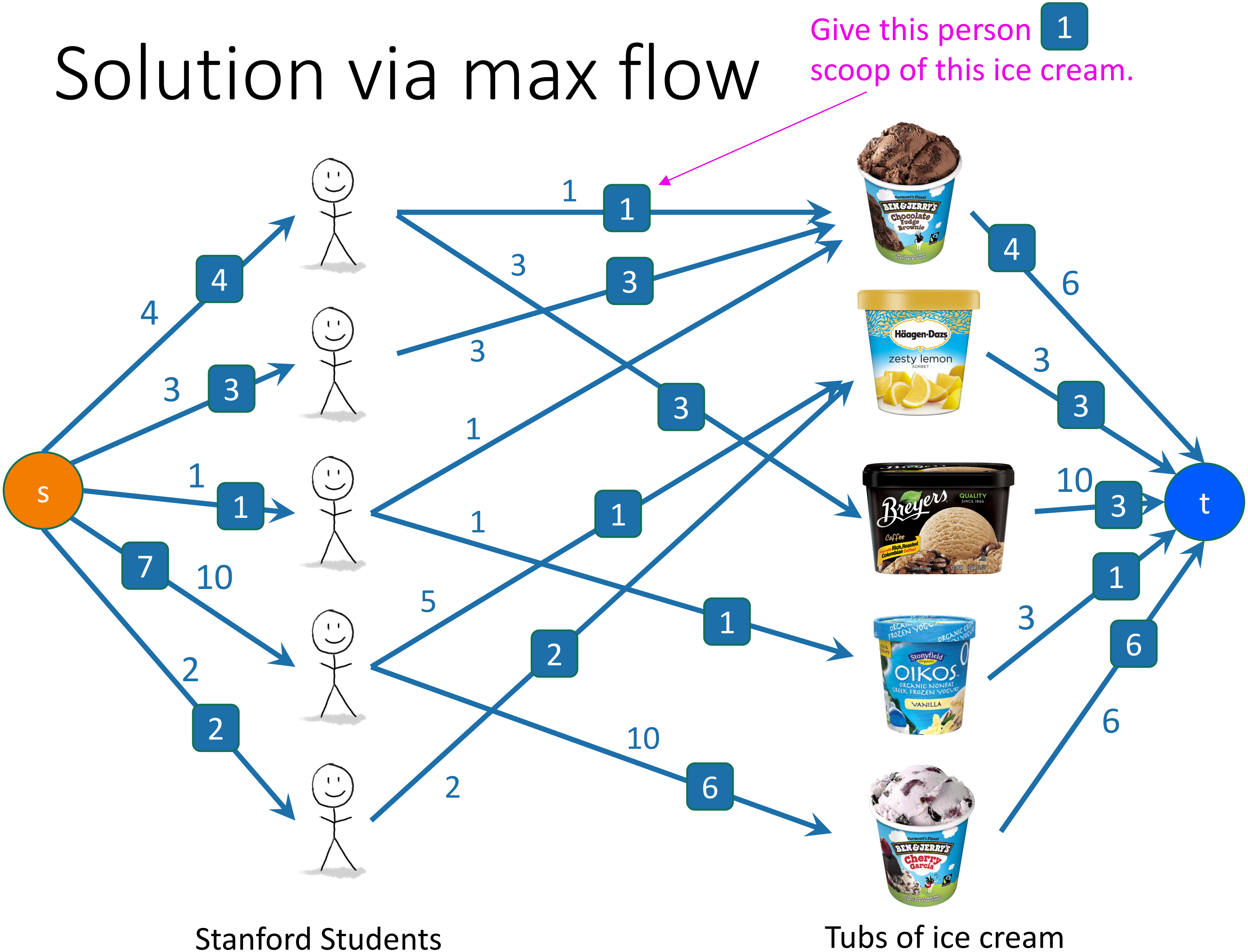
Stanford Students

Tubs of ice cream

Solution via max flow

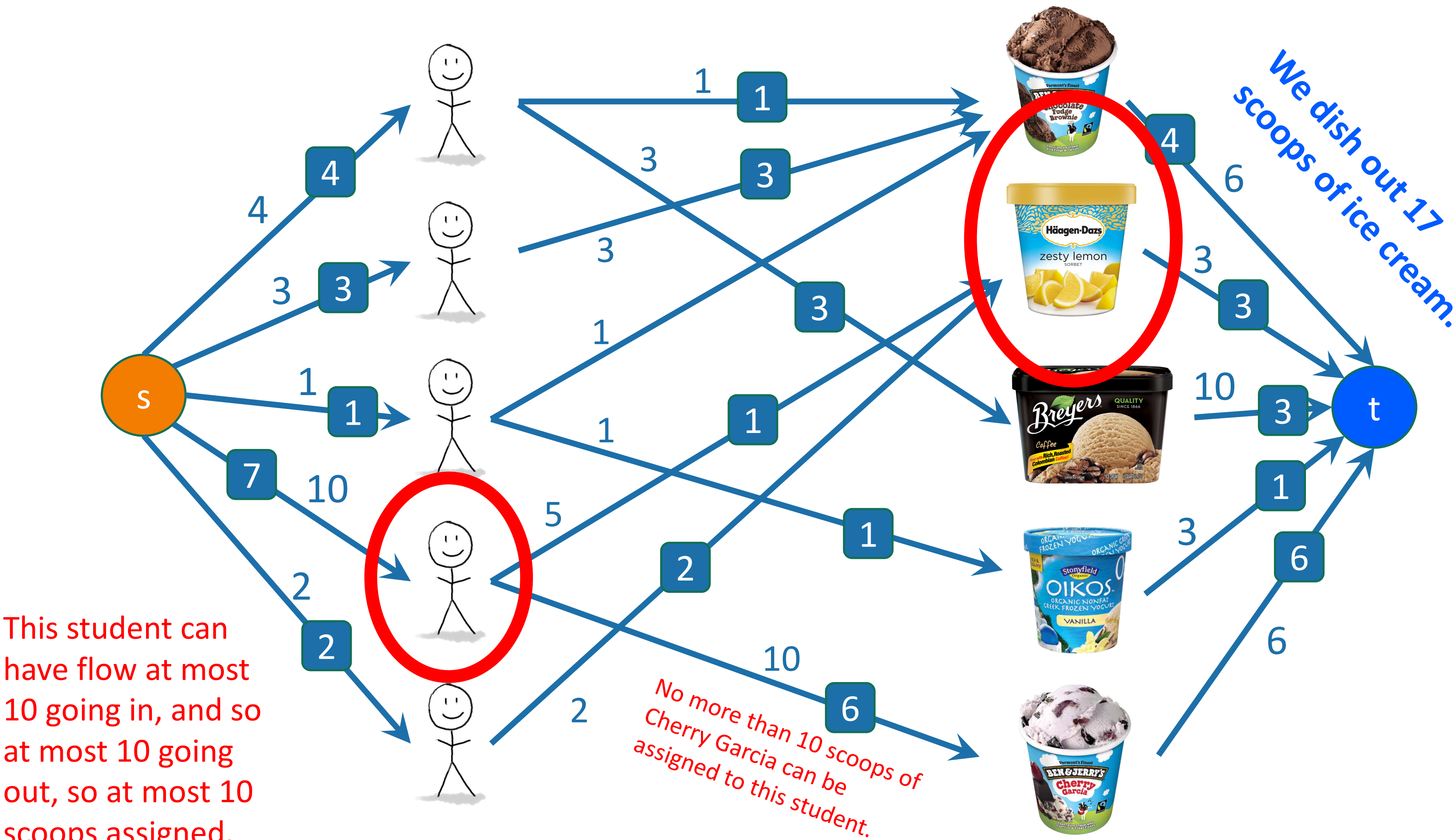


Solution via max flow



Solution via max flow

No more than 3 scoops of sorbet can be assigned.



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