Lecture 16

max flows, min cuts, and Ford-Fulkerson

Announcements:

- Part II (2 videos) of EthiCS lectures on Algorithms in the Real World will be added to website.

- HW7 due today.
- HW8 (last homework!) out today

- This week's lectures (including today) are included in the course final.

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?

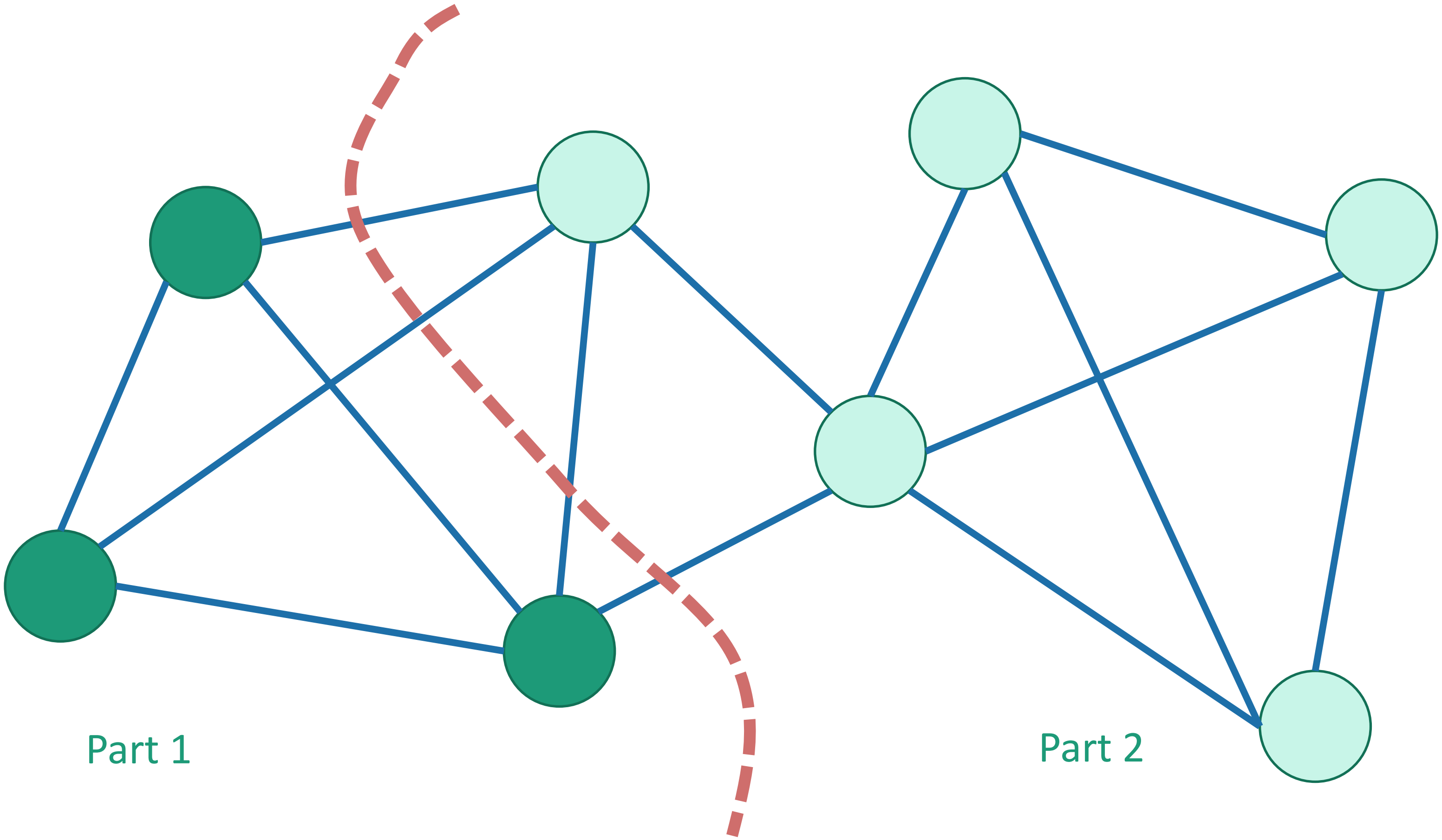
This lecture will skip a few proofs, but you can find them in the lecture notes.

Lucky the lackadaisical lemur

Cuts in graphs

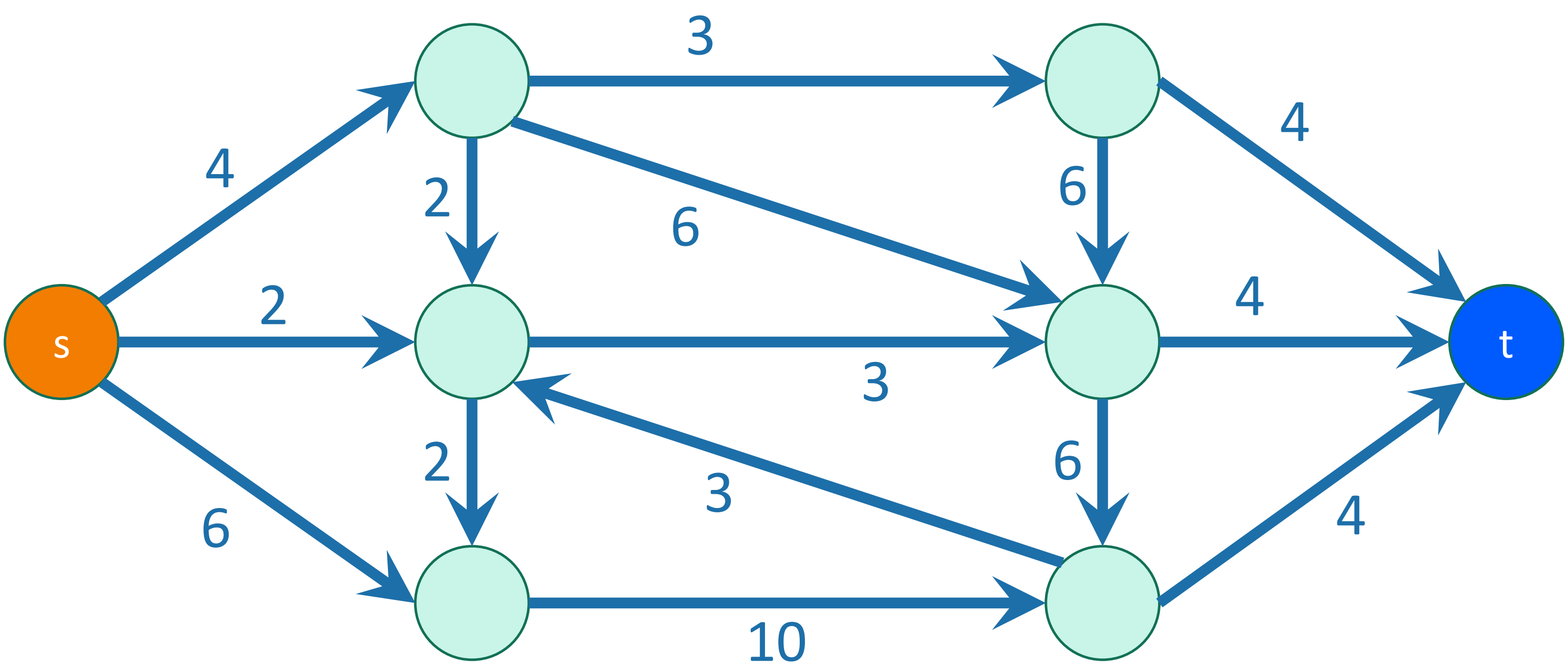
Last time, graphs were undirected and weighted.

- A cut is a partition of the vertices into two nonempty parts.



Today

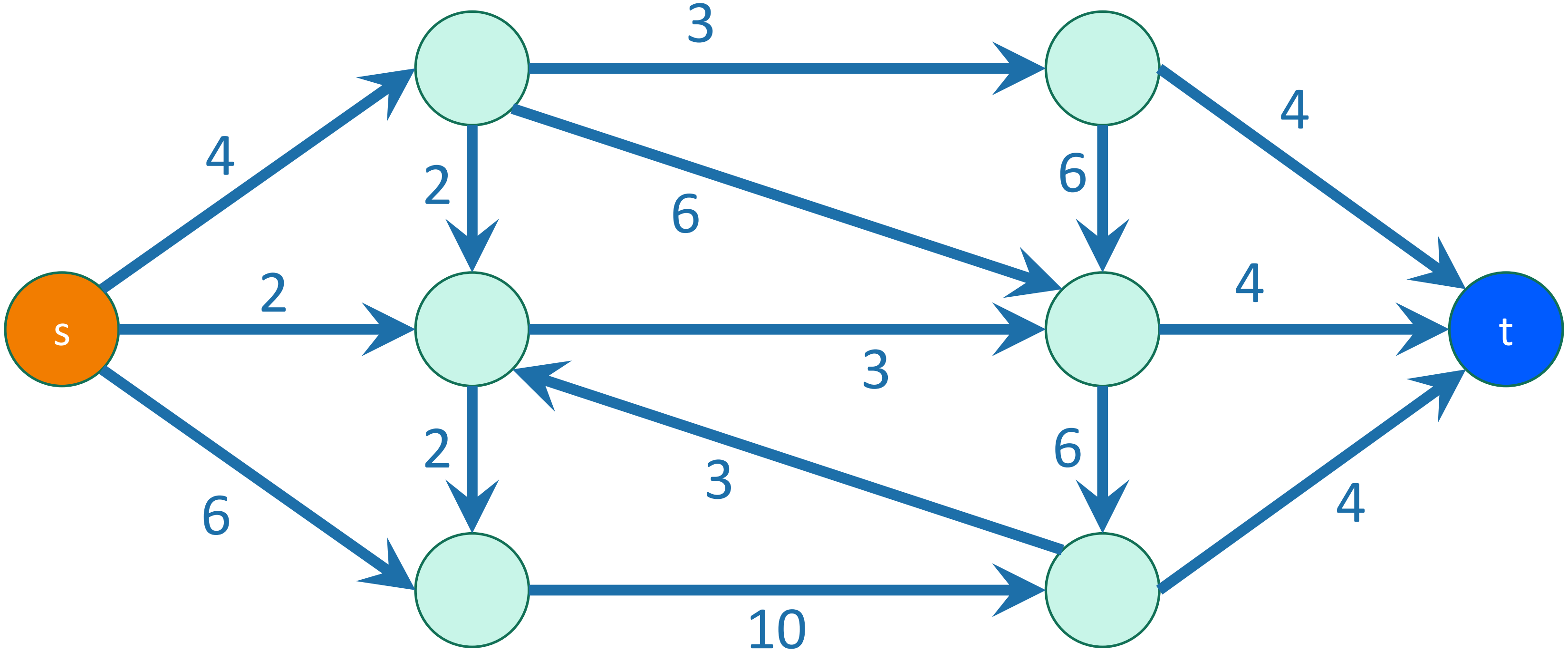
- Graphs are directed and edges have “capacities” (weights)
- We have a special “source” vertex s and “sink” vertex t.
    - s has only outgoing edges*
    - t has only incoming edges*



*simplifying assumptions, but everything can be generalized to arbitrary directed graphs

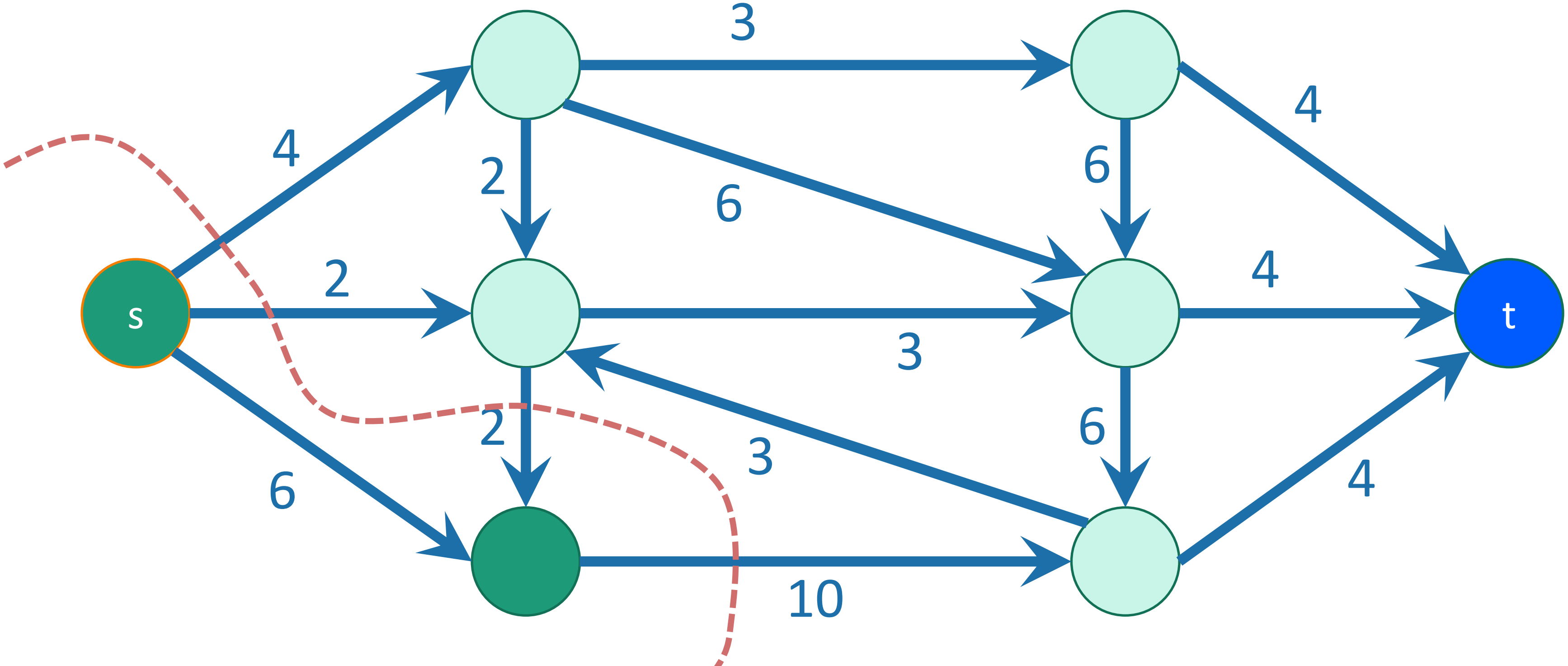
An **s-t cut**

is a cut which separates s from t



An **s-t cut**

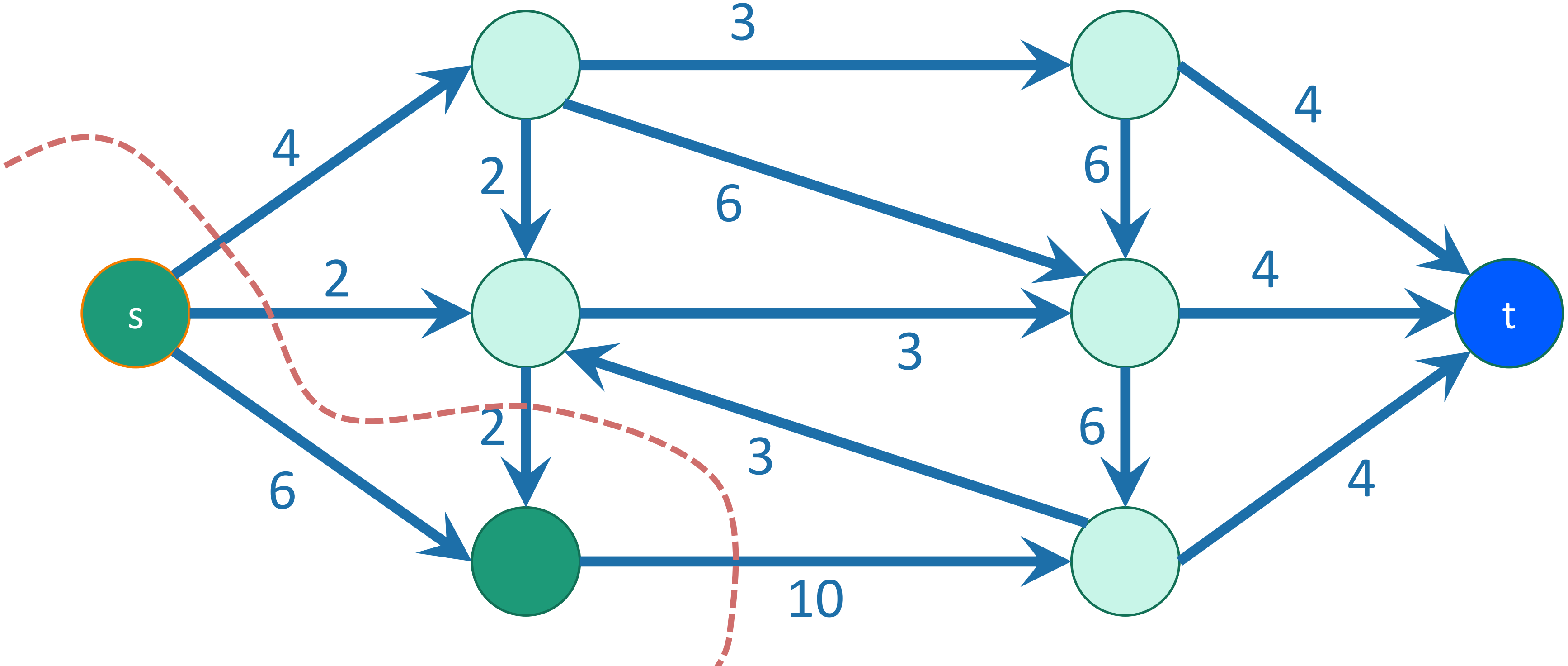
is a cut which separates s from t



An **s-t cut**

is a cut which separates s from t

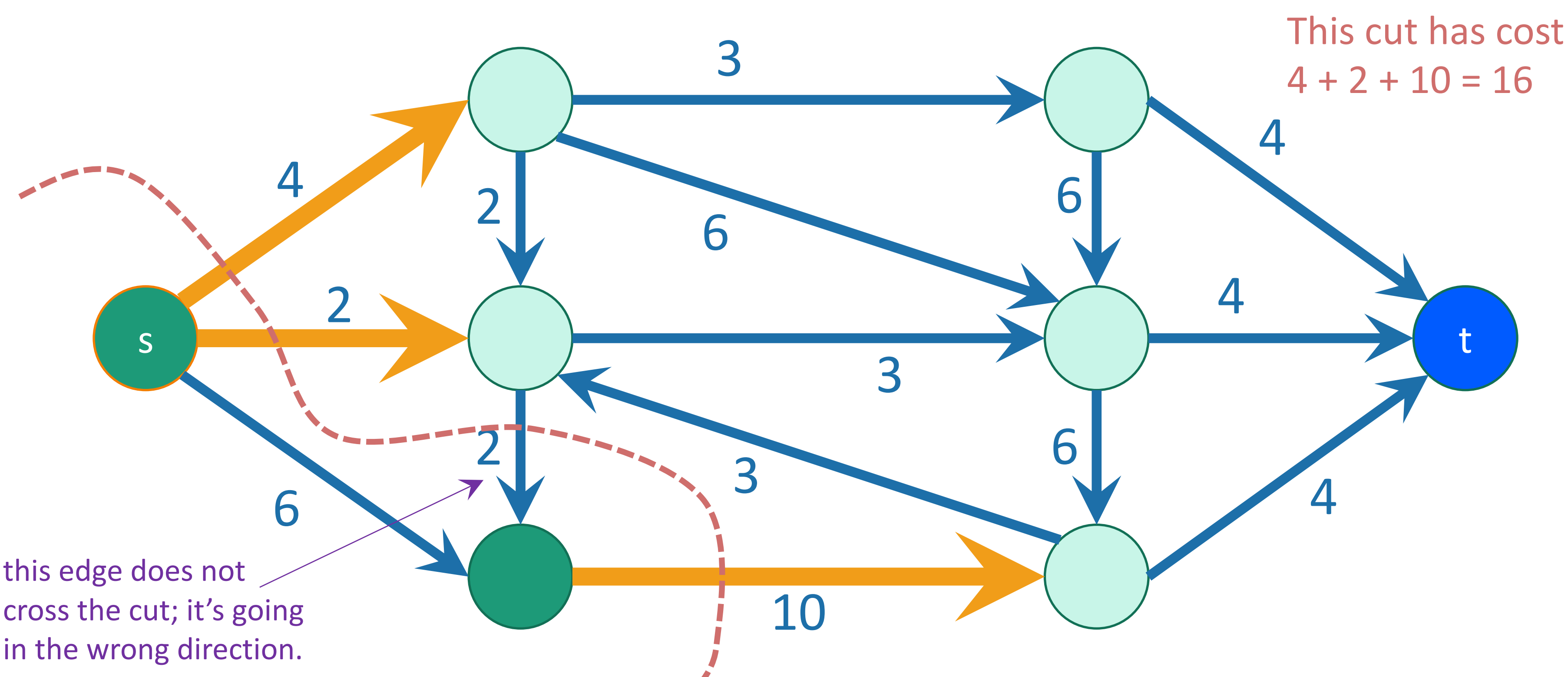
- An edge **crosses the cut** if it goes from s's side to t's side.



An **s-t cut**

is a cut which separates s from t

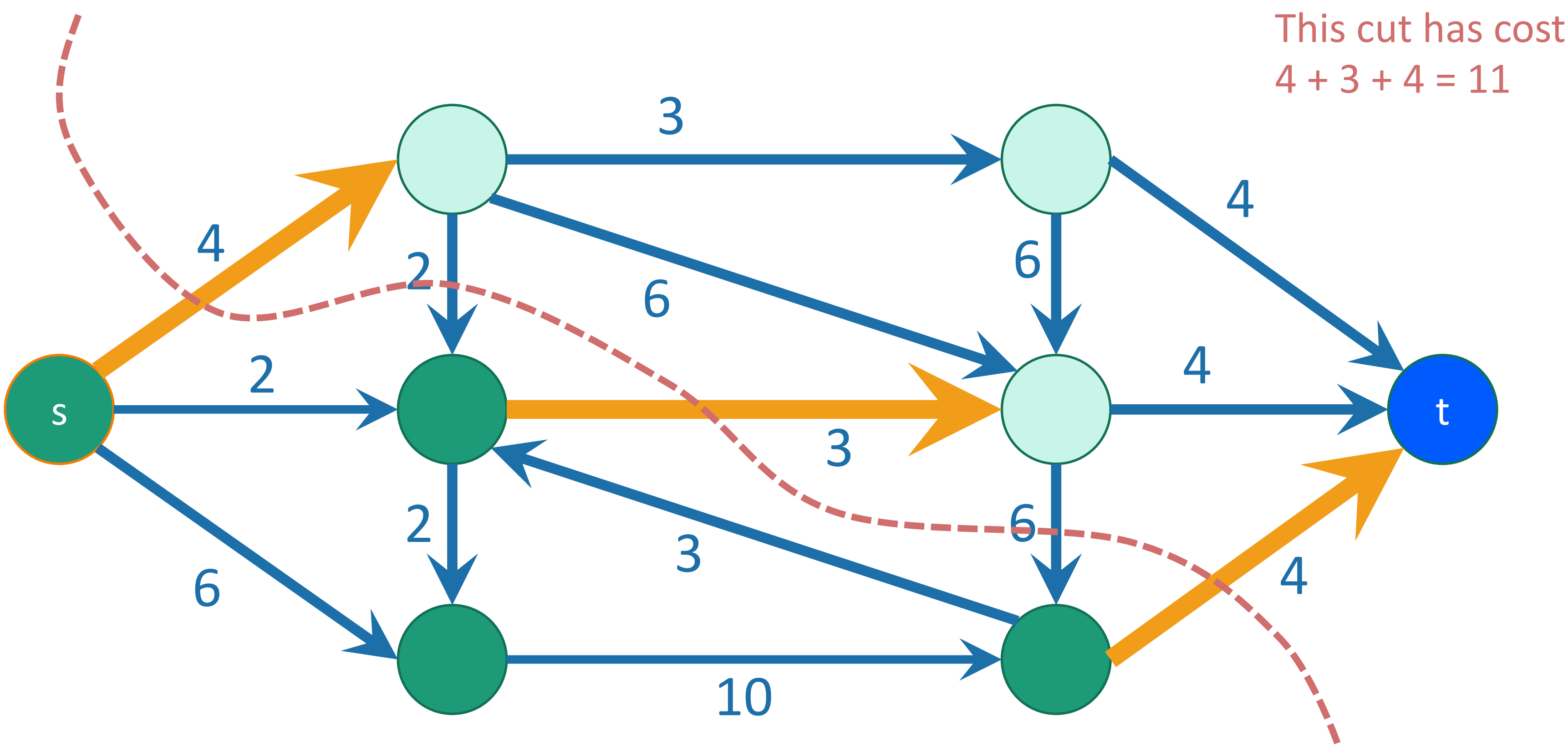
- An edge **crosses the cut** if it goes from s's side to t's side.
- The **cost** (or capacity) of a cut is the sum of the capacities of the edges that cross the cut.



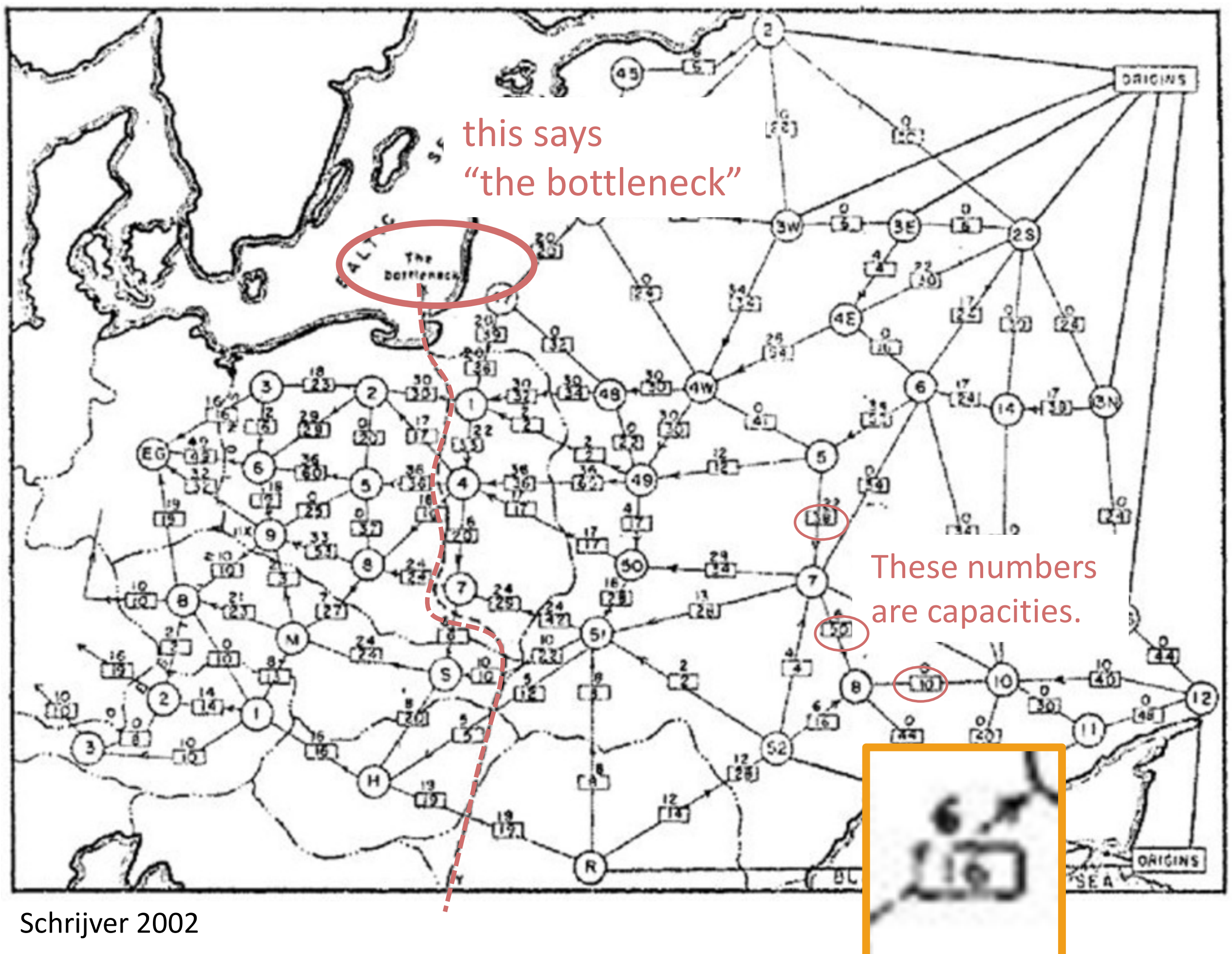
A minimum **s-t cut**

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

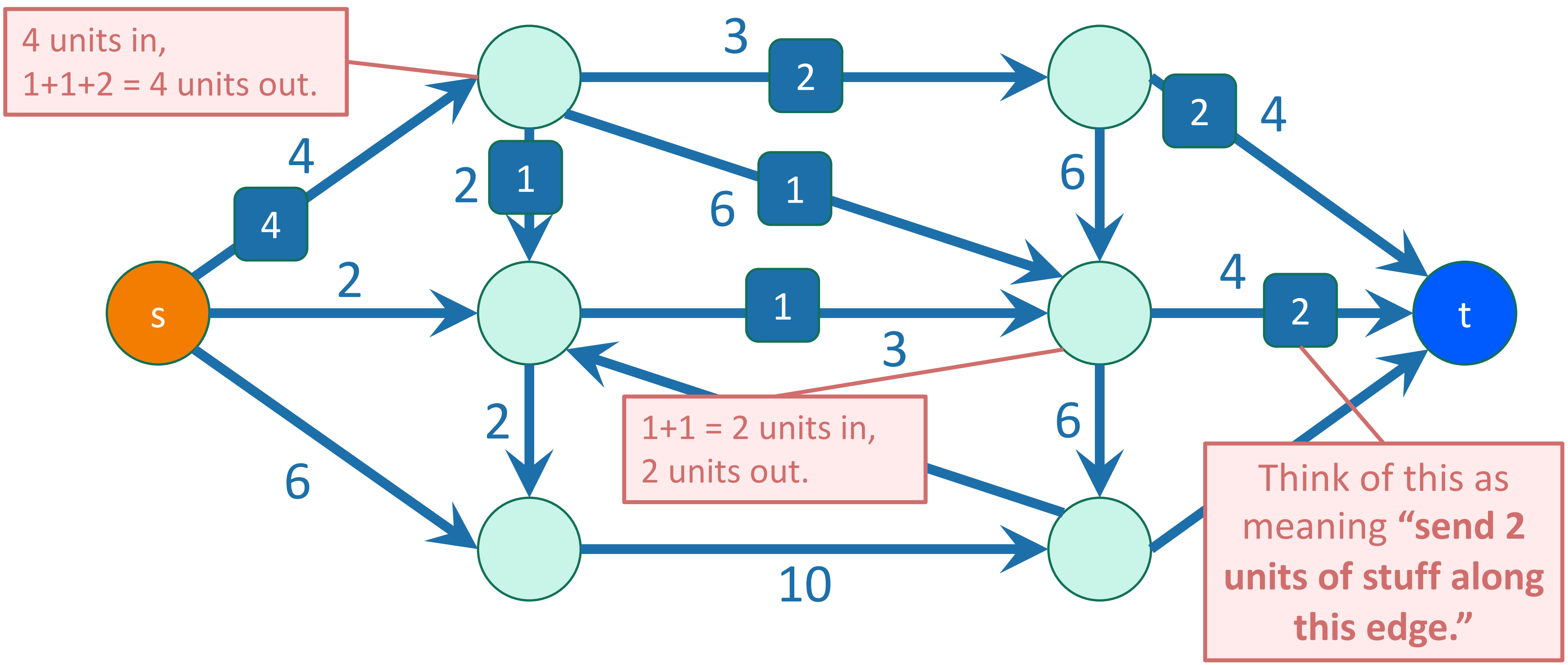


Schrijver 2002

- 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.

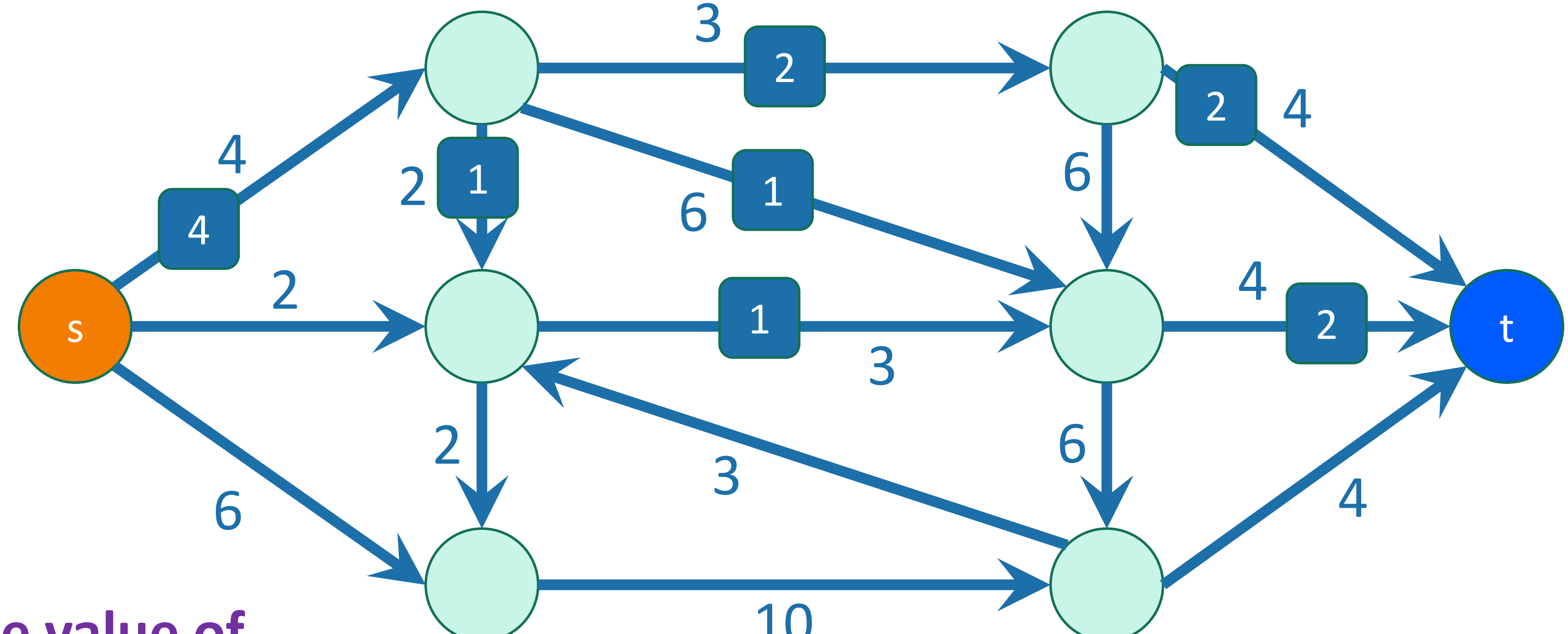


Flows

- 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.

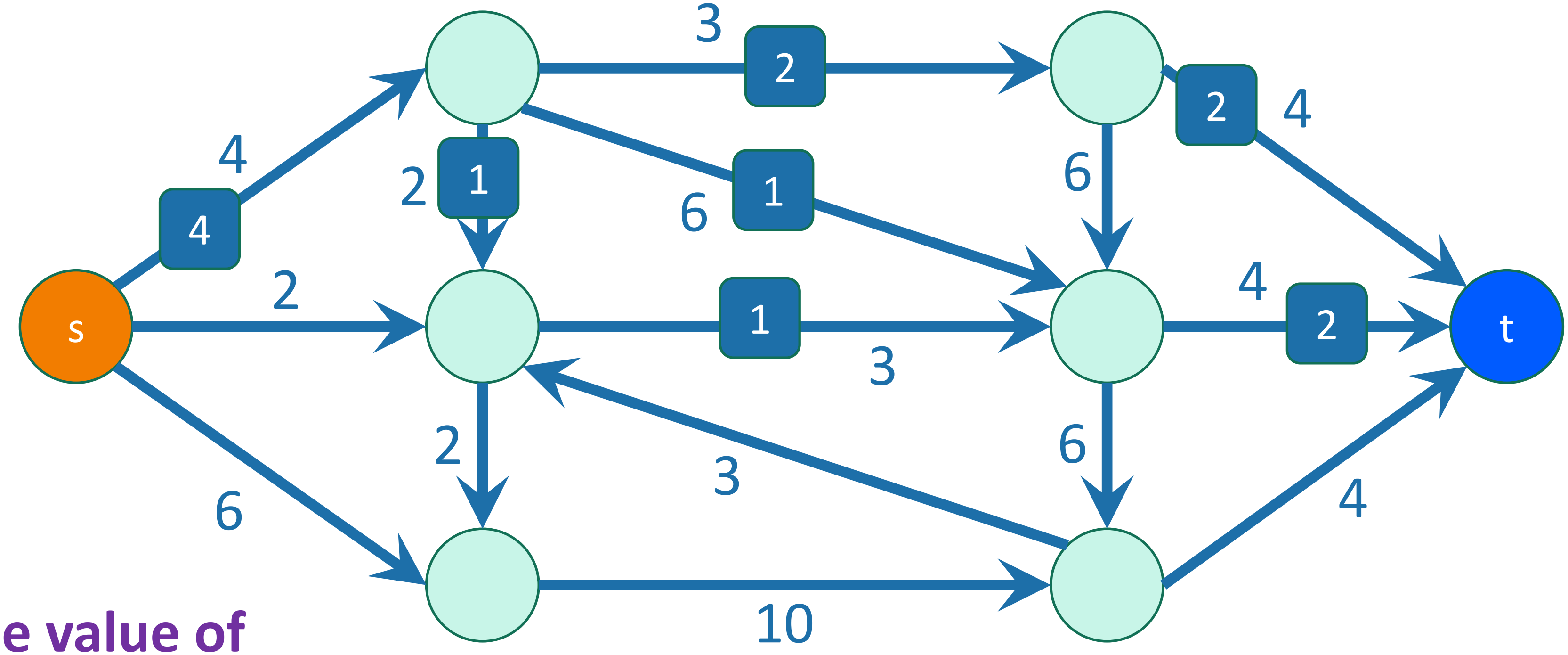


The value of this flow is 4.

A maximum flow

is a flow of maximum value.

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

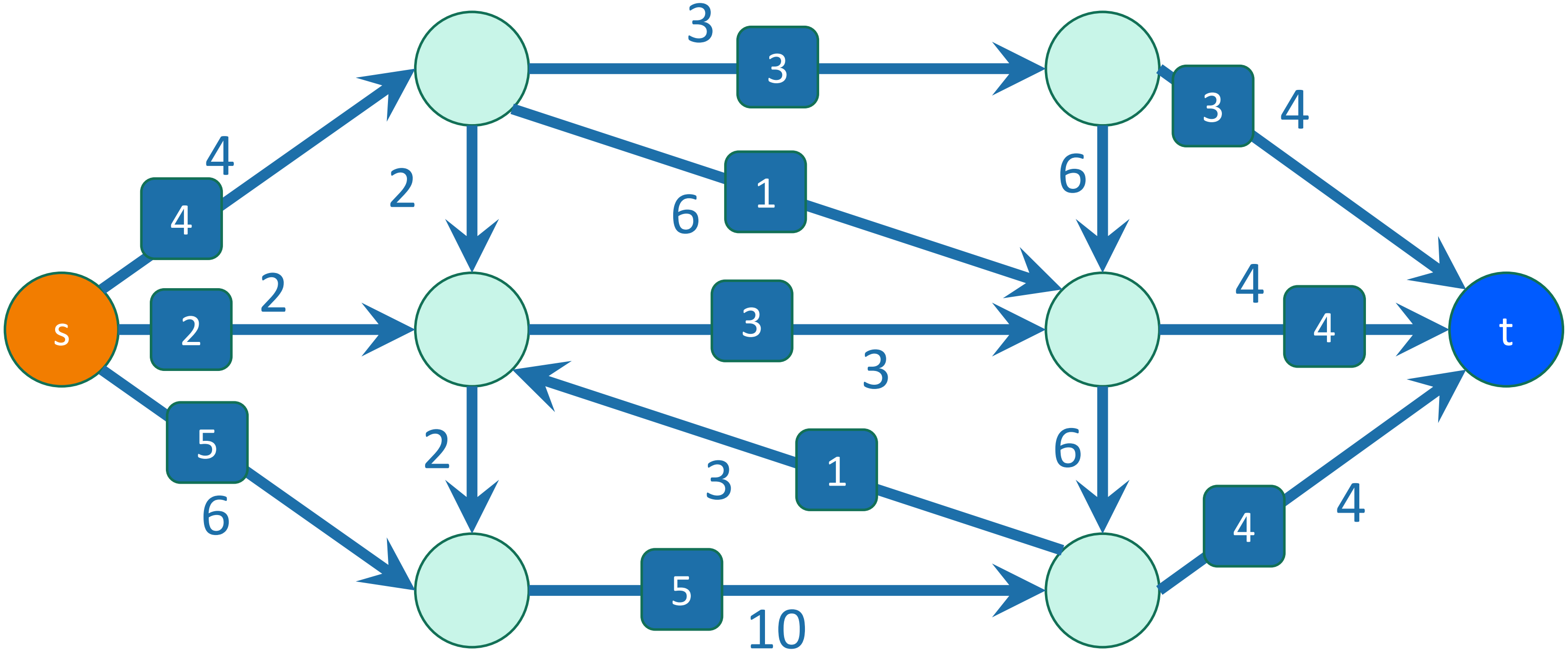


The value of this flow is 4.

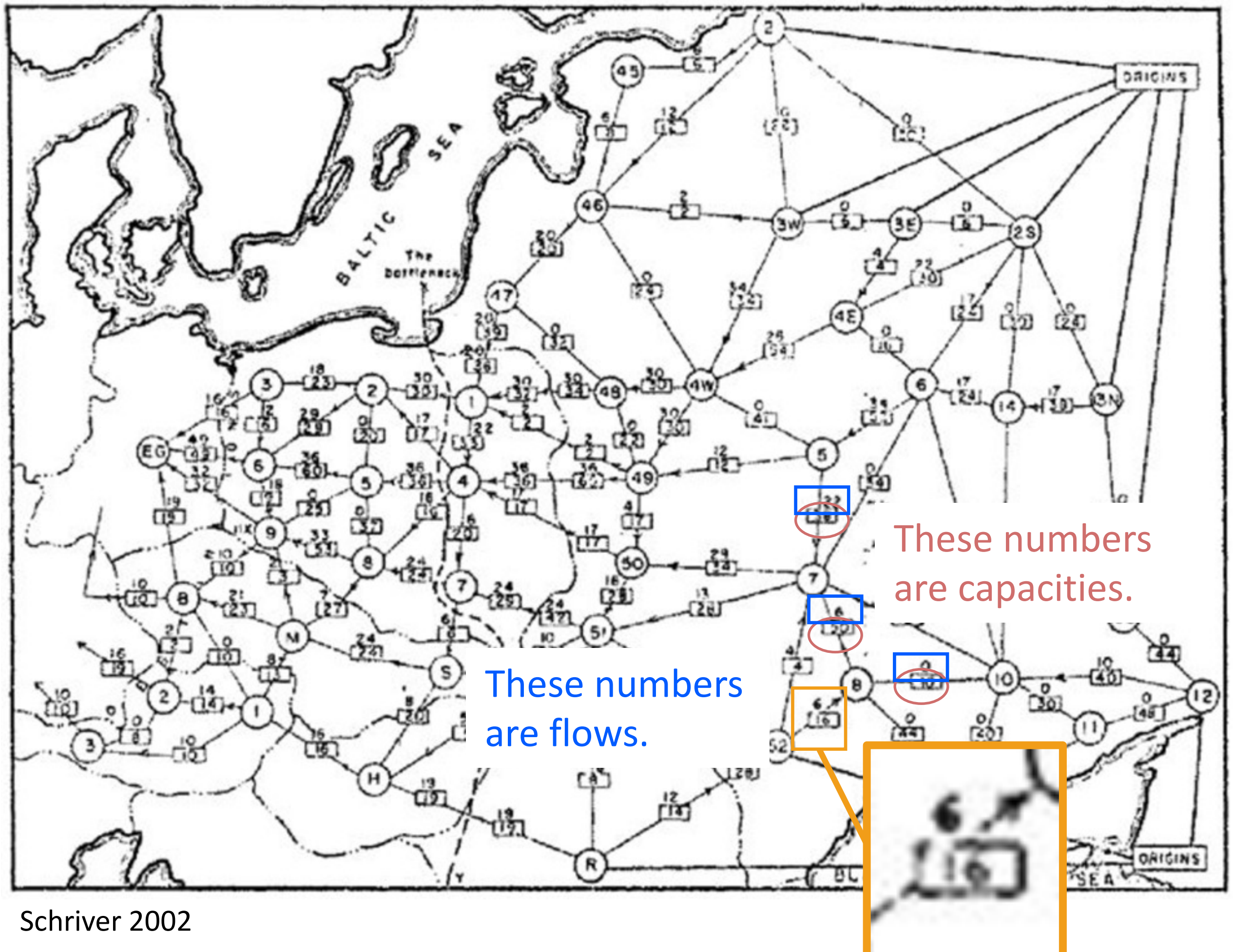
A maximum flow

is a flow of maximum value.

- This one is maximum; it has value 11.



Example where this comes up



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.

2. Max-Flow Min-Cut

The Soviet rail system also roused the interest of the Americans, and again it inspired fundamental research in optimization.

In their basic paper *Maximal Flow through a Network* (published first as a RAND Report of November 19, 1954), Ford and Fulkerson [5] mention that the maximum flow problem was formulated by T.E. Harris as follows:

> Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other.

In their 1962 book *Flows in Networks*, Ford and Fulkerson [7] give a more precise reference to the origin of the problem[5]:

> It was posed to the authors in the spring of 1955 by T.E. Harris, who, in conjunction with General F.S. Ross (Ret.), had formulated a simplified model of railway traffic flow, and pinpointed this particular problem as the central one suggested by the model [11].

Ford-Fulkerson's reference 11 is a secret report by Harris and Ross [11] entitled *Fundamentals of a Method for Evaluating Rail Net Capacities*, dated October 24, 1955[6] and written for the US Air Force. At our request, the Pentagon downgraded it to "unclassified" on May 21, 1999.

SECRET

U. S. AIR FORCE

PROJECT RAND

RESEARCH MEMORANDUM

FUNDAMENTALS OF A METHOD FOR EVALUATING RAIL NET CAPACITIES (U)

T. E. Harris
F. S. Ross

RM-1573

October 24, 1955

Copy No. 37

SECRET

SUMMARY

Air power is an effective means of interdicting an enemy's rail system, and such usage is a logical and important mission for this Arm.

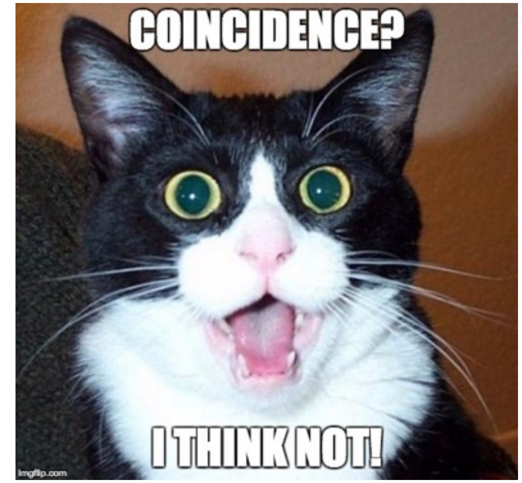
As in many military operations, however, the success of interdiction depends largely on how complete, accurate, and timely is the commander's information, particularly concerning the effect of his interdiction-program efforts on the enemy's capability to move men and supplies. This information should be available at the time the results are being achieved.

The present paper describes the fundamentals of a method intended to help the specialist who is engaged in estimating railway capacities, so that he might more readily accomplish this purpose and thus assist the commander and his staff with greater efficiency than is possible at present.

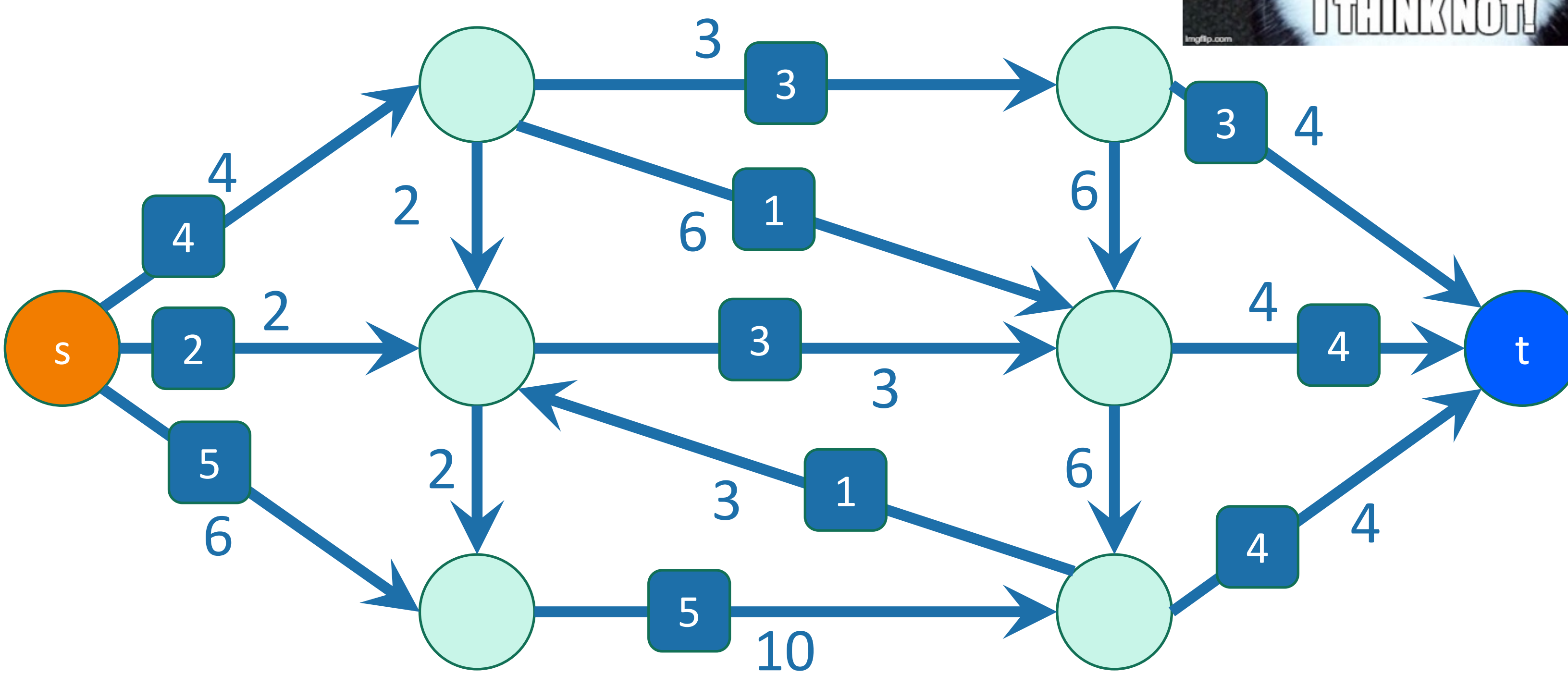
A maximum flow

is a flow of maximum value.

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

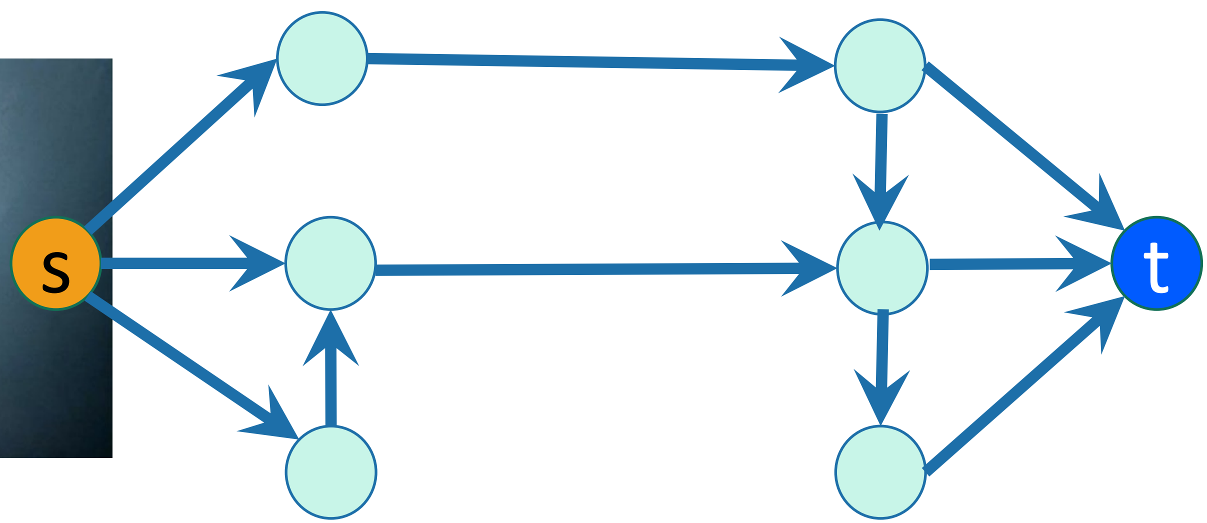


- This one is maximum; it has value 11.



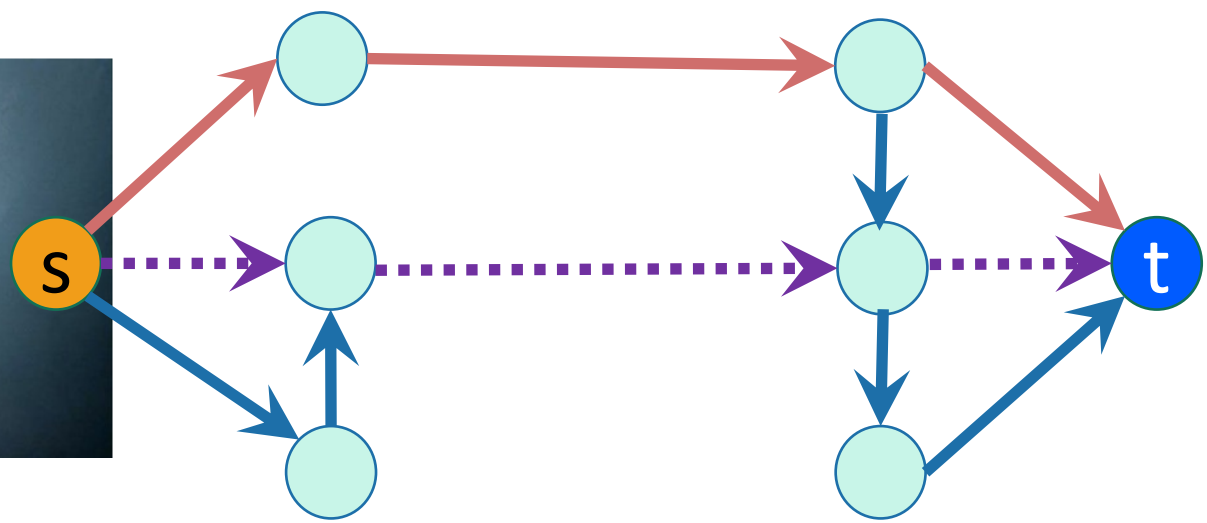
Pre-lecture exercise

- 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!



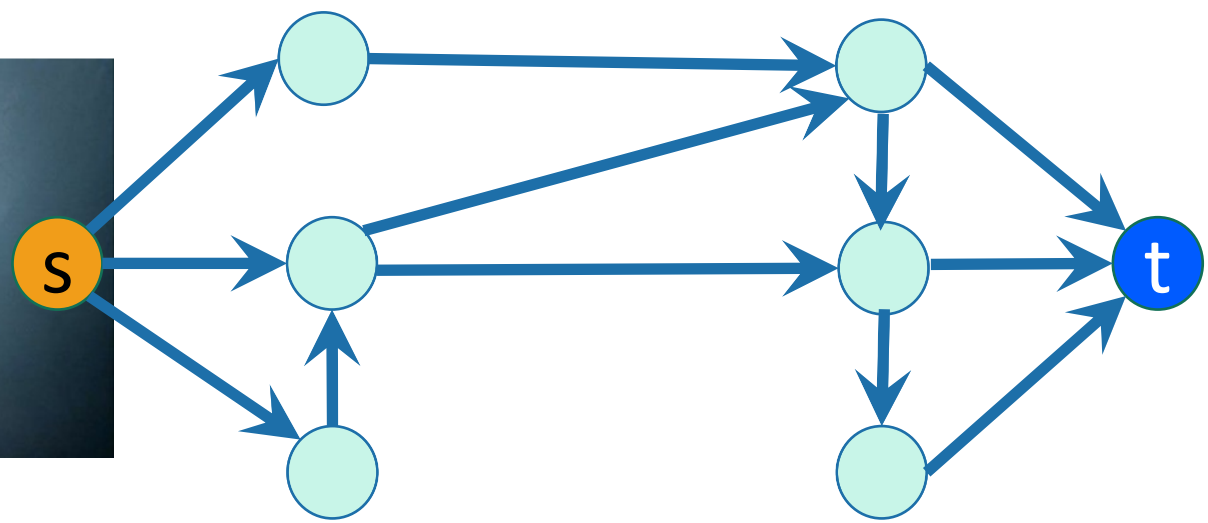
Pre-lecture exercise

- 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!



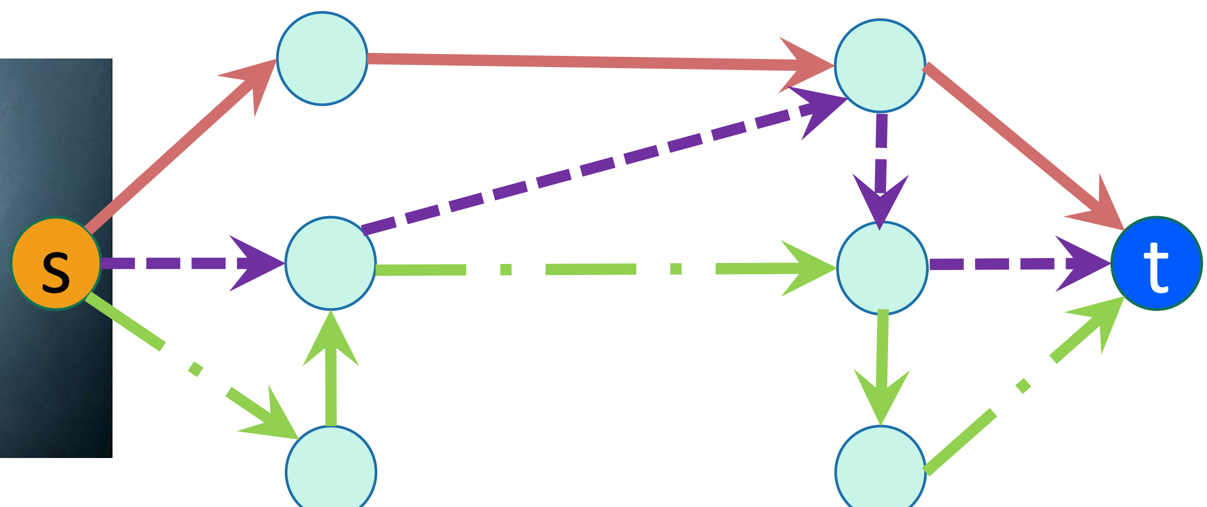
How about now?

- 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

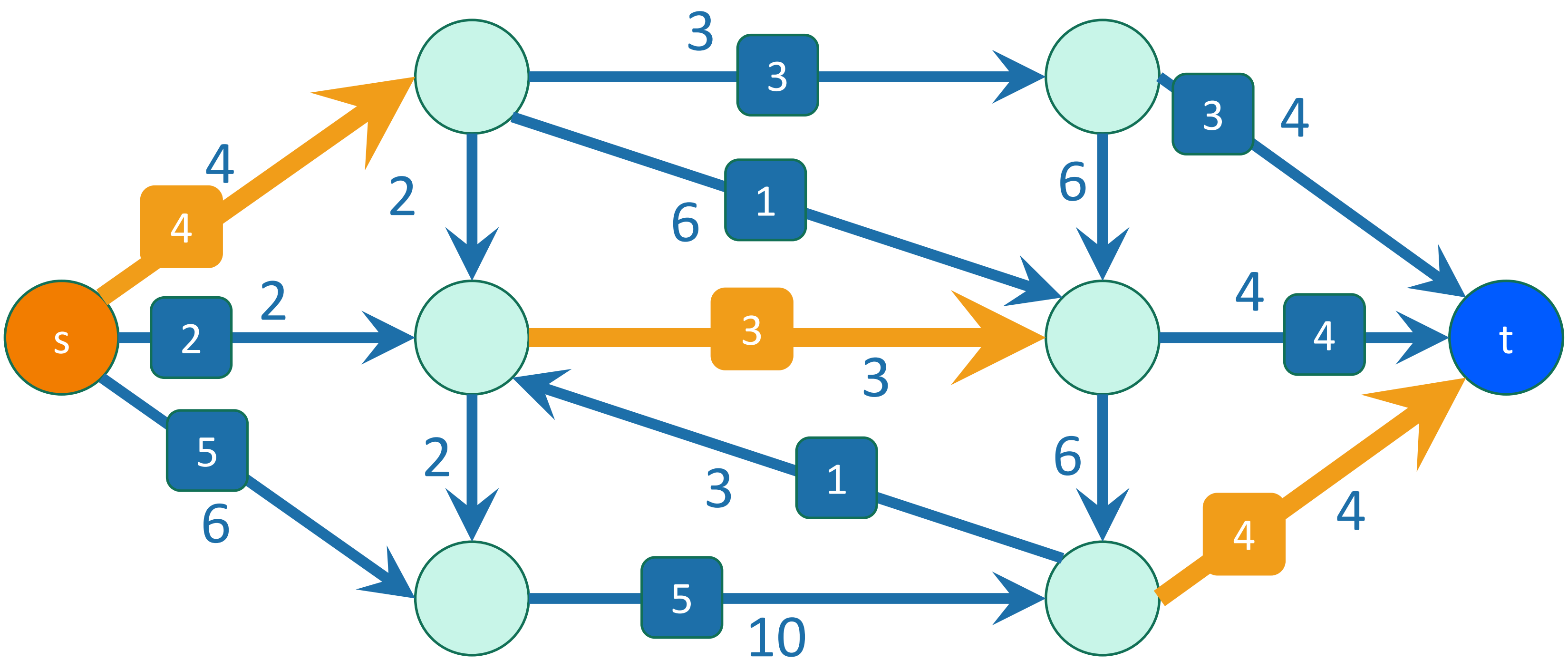
- Can you find a graph where the two numbers are different?

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.



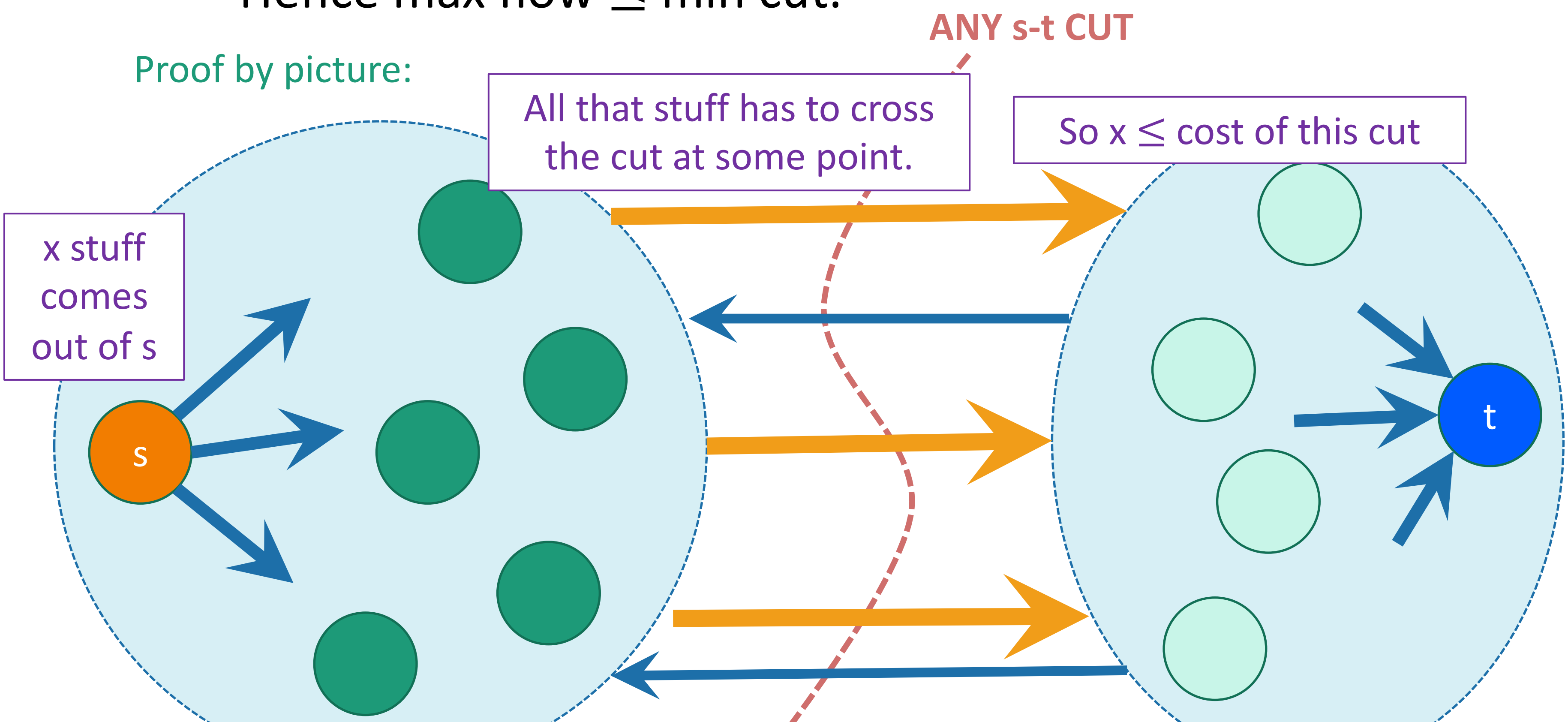
Proof outline

- Lemma 1: max flow $\leq$ min cut.
  - Proof-by-picture
- What we actually want: max flow $=$ 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 max flow ≤ 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 max flow $\leq$ min cut.

- That was proof-by-picture.
- Good exercise to convert this to a proof-by-proof!

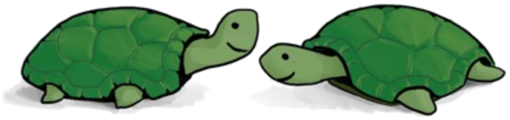
Min-Cut Max-Flow Thm

- **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.

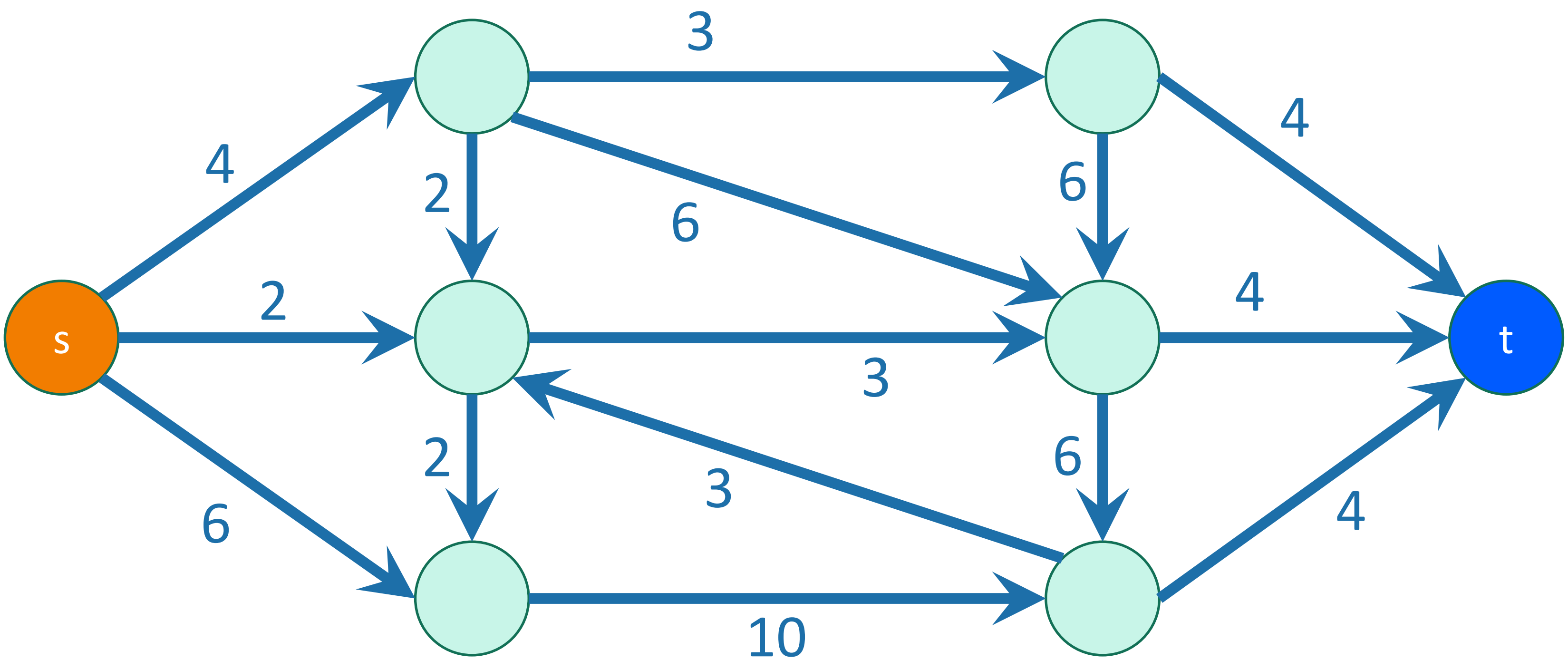
- The theorem is stronger:
  - max flow $=$ min cut
  - This will be proof-by-algorithm!

Maximum flow

- Let's brainstorm some algorithms for maximum flow.



Think-share!



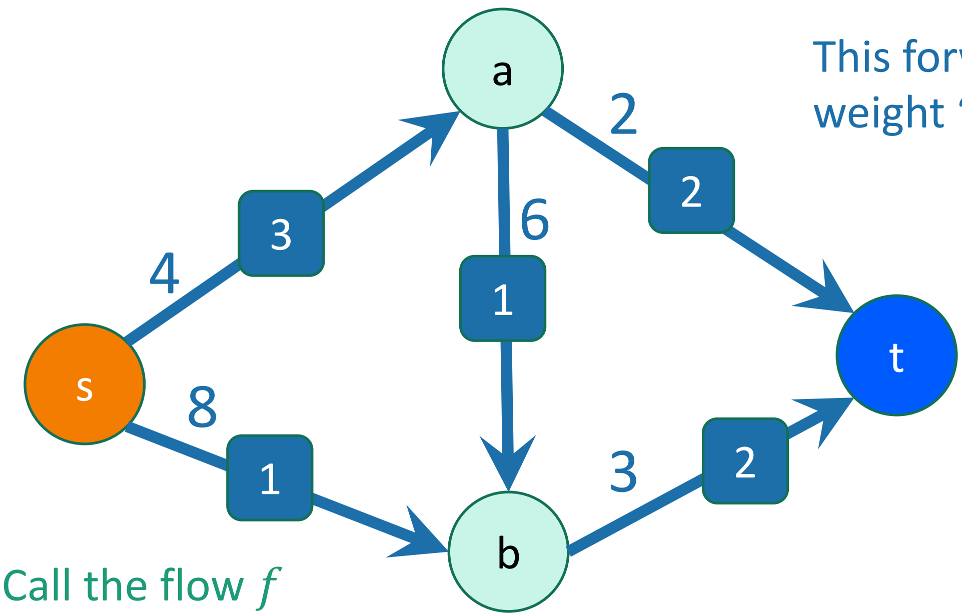
Ford-Fulkerson algorithm

- 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

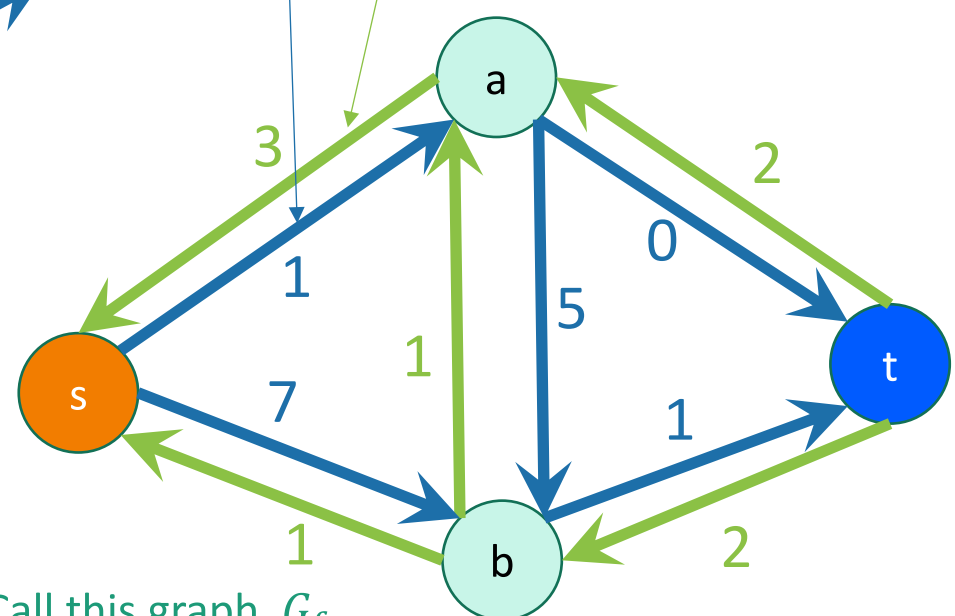


Call the flow f

Call the graph G

This forward edge has weight "capacity – flow".

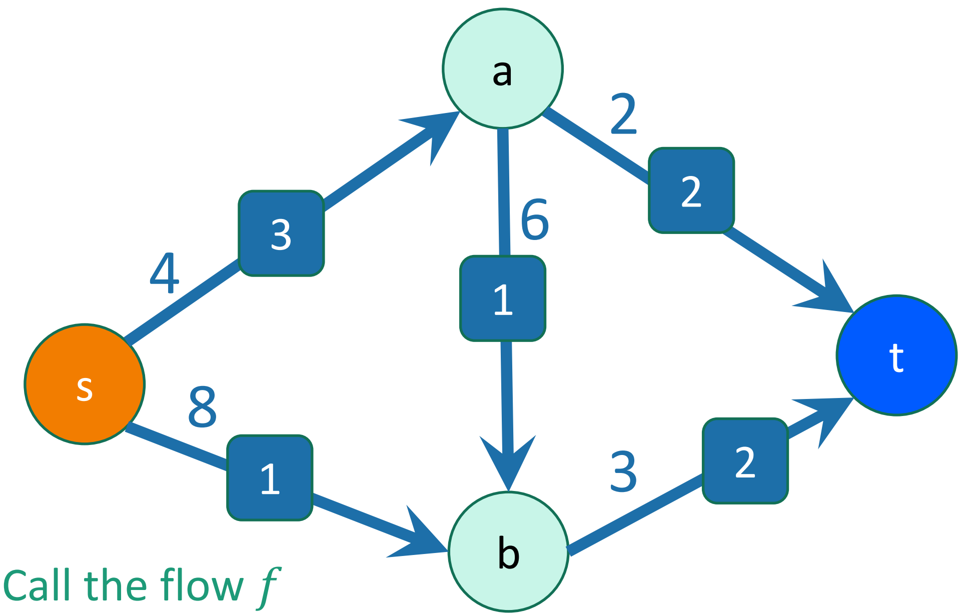
This backward edge has weight "flow".



Call this graph G_f

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

Tool: Residual networks
Say we have a flow

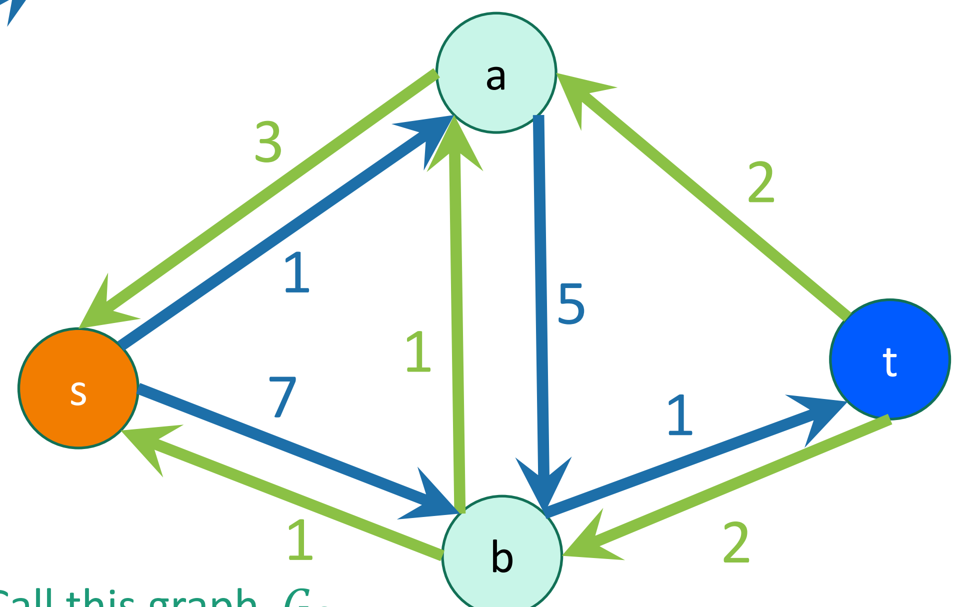


Call the flow f
Call the graph G

Forward edges are the amount that's left.

Backwards edges are the amount that's been used.

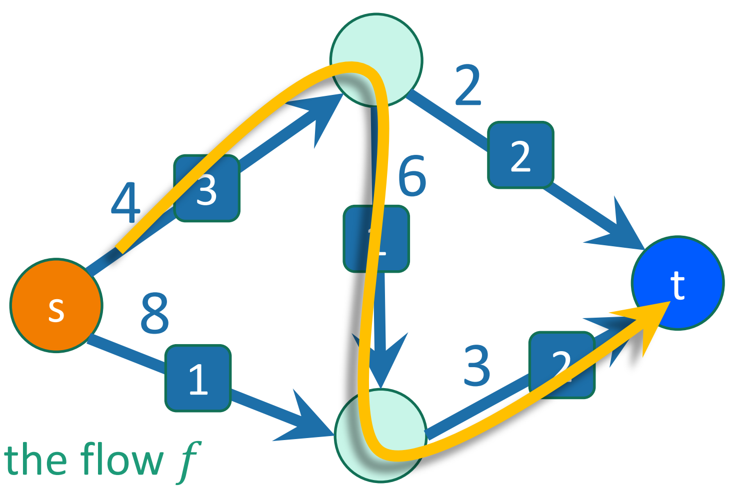
Create a new **residual network** from this flow:



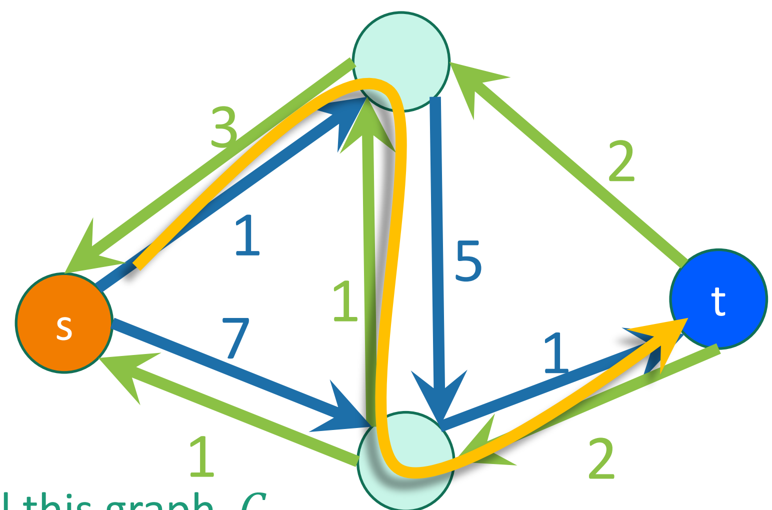
Call this graph G_f

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.



Call the flow f
Call the graph G

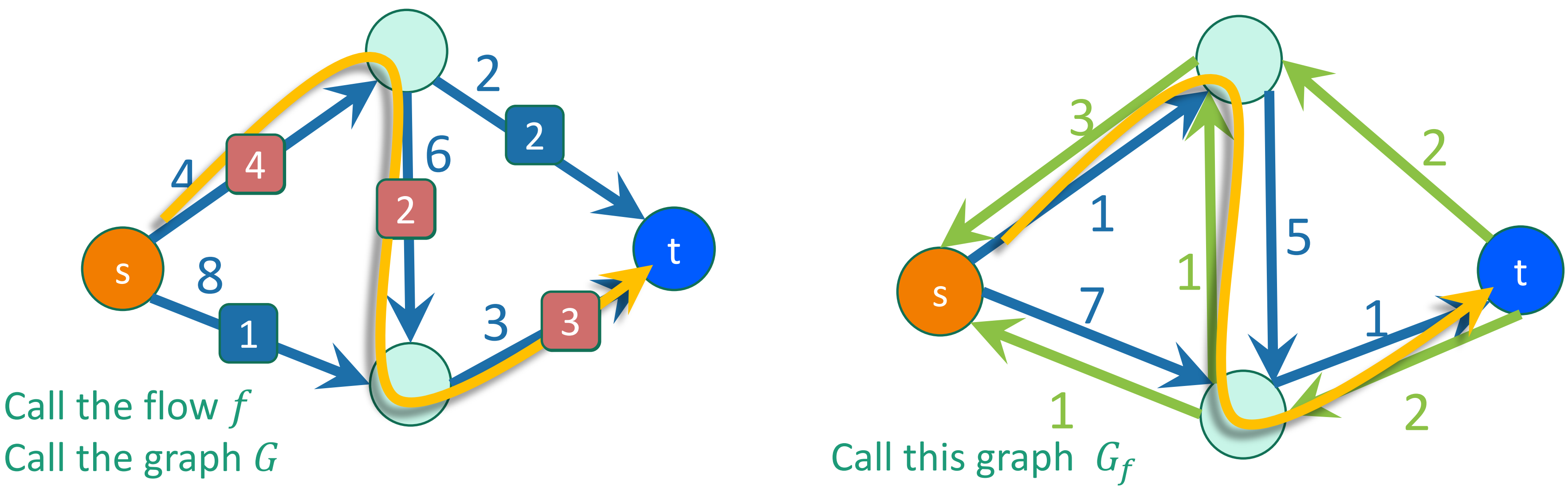


Call this graph G_f

Claim:

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

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

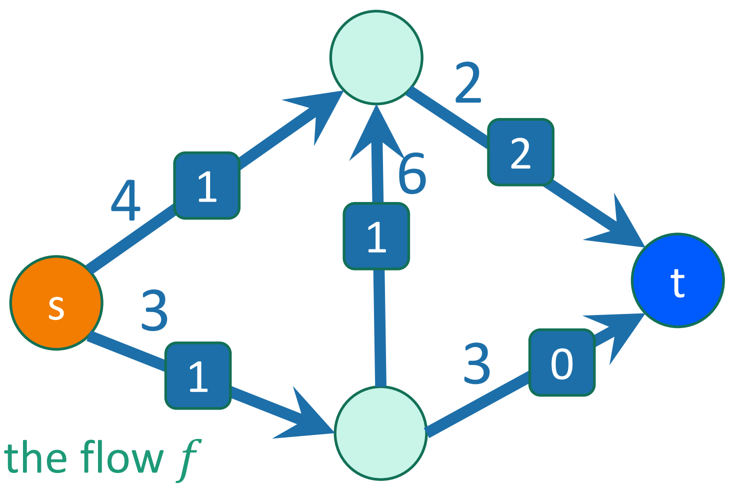


- Forward edges indicate how much stuff can still go through.
- Just increase the flow on all the edges!

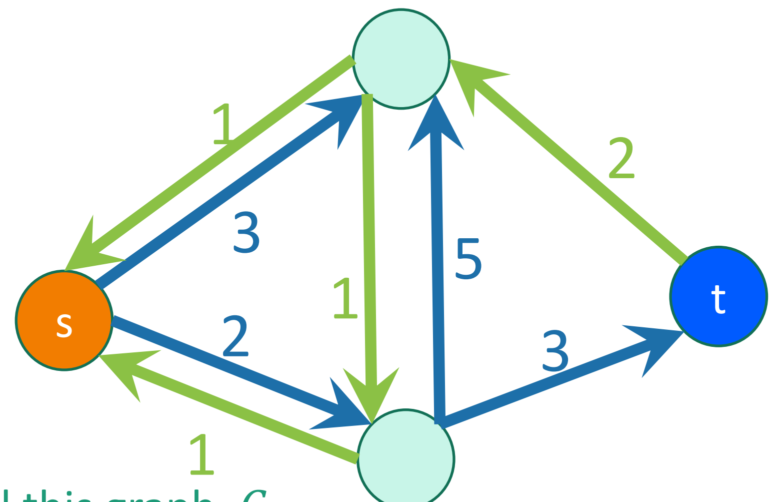
Claim:

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



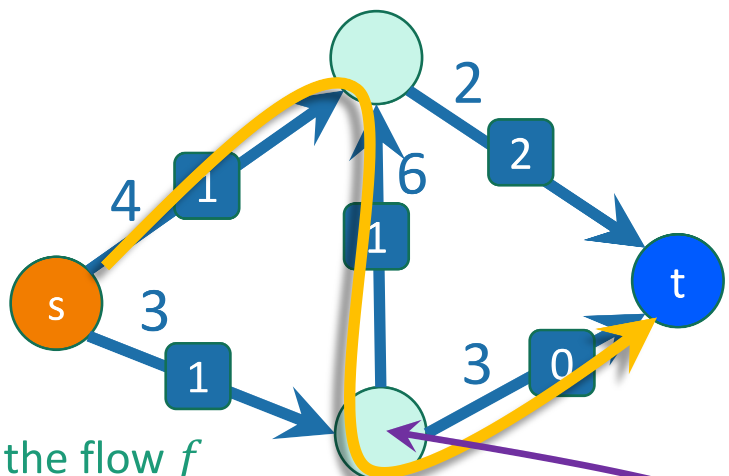
Call this graph G_f

I changed some of the weights and edge directions.

Claim:

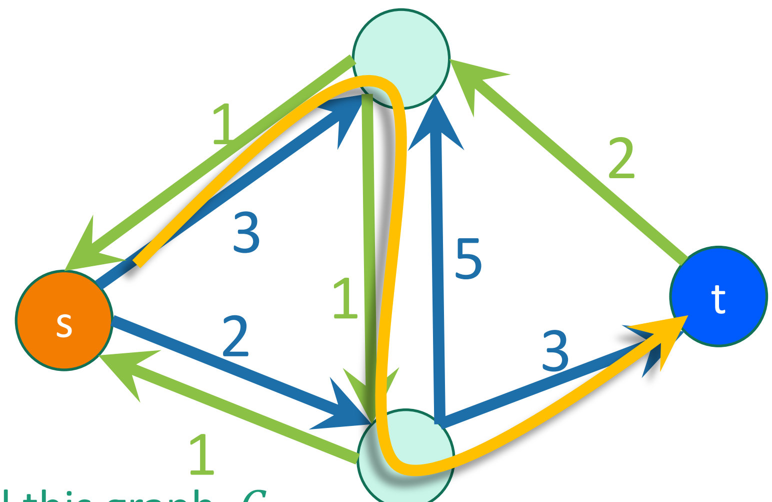
if there is an augmenting path, we can increase the flow along that path.

- 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!

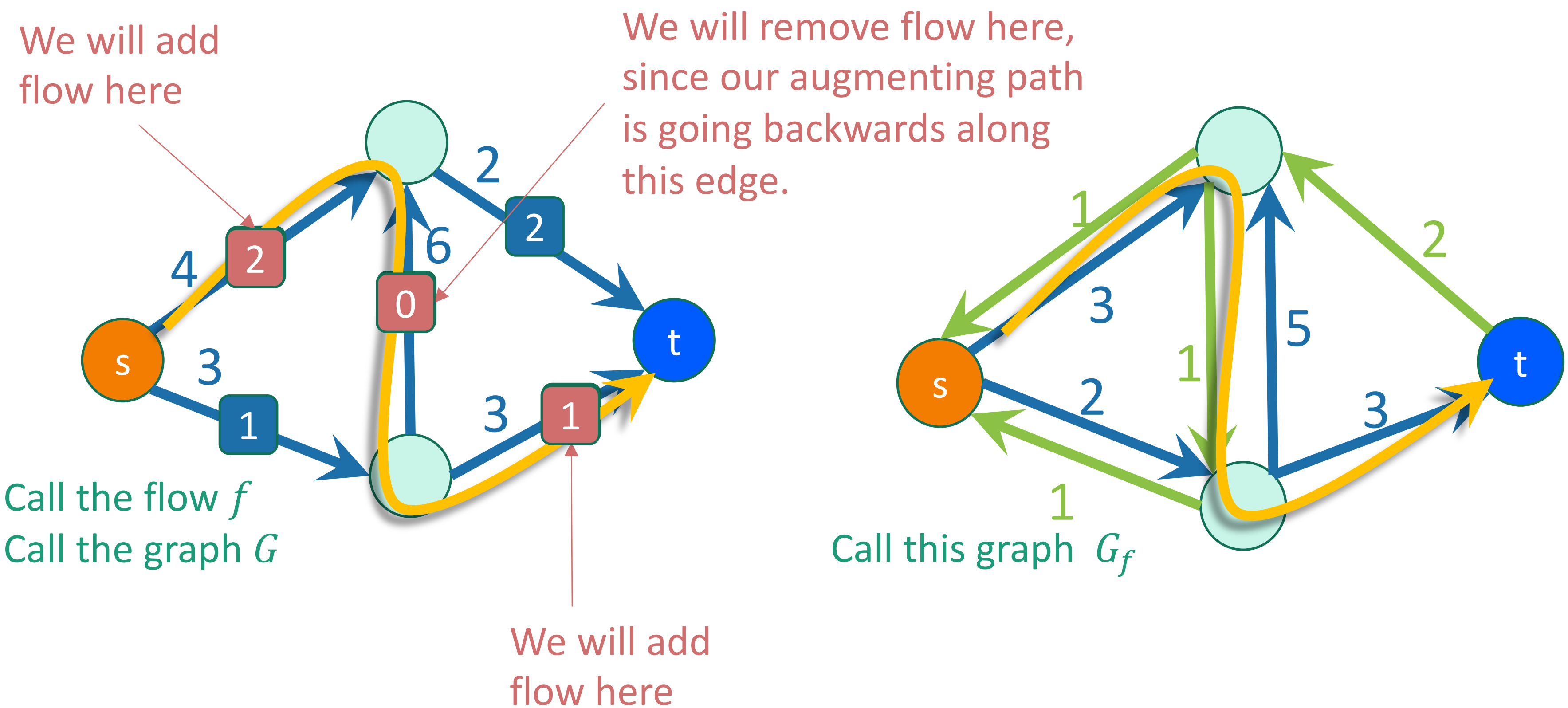
- For example, that will mess up the conservation of stuff at this vertex.

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:

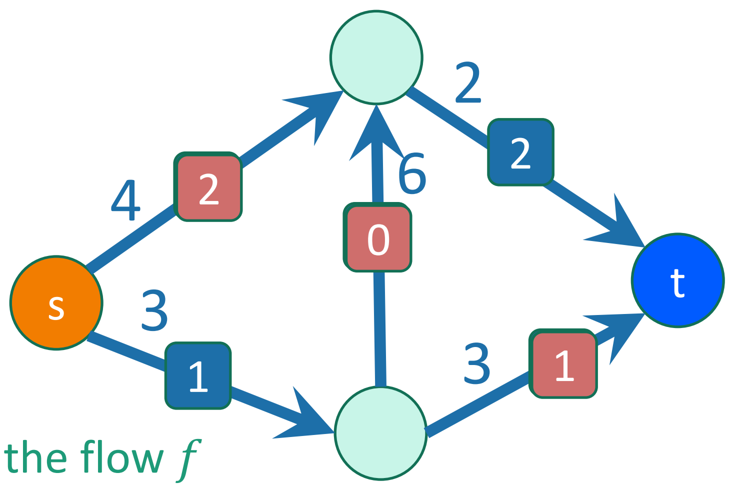


Claim:

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

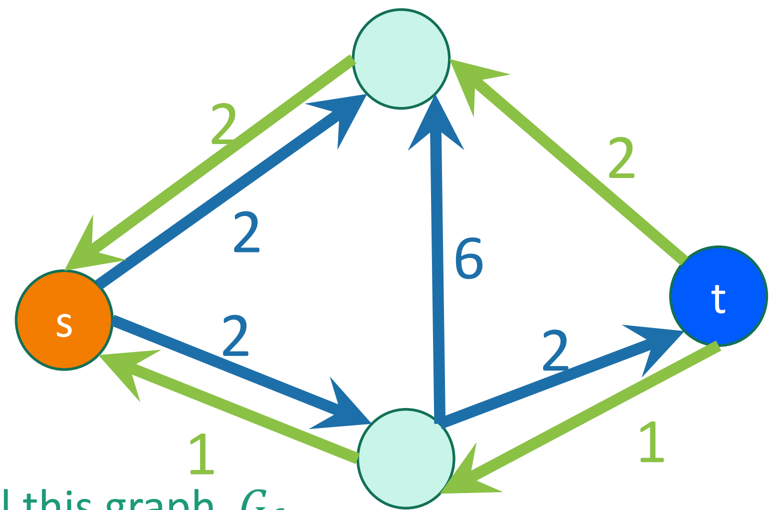
- In this case we do something a bit different:

Then we'll update the residual graph:



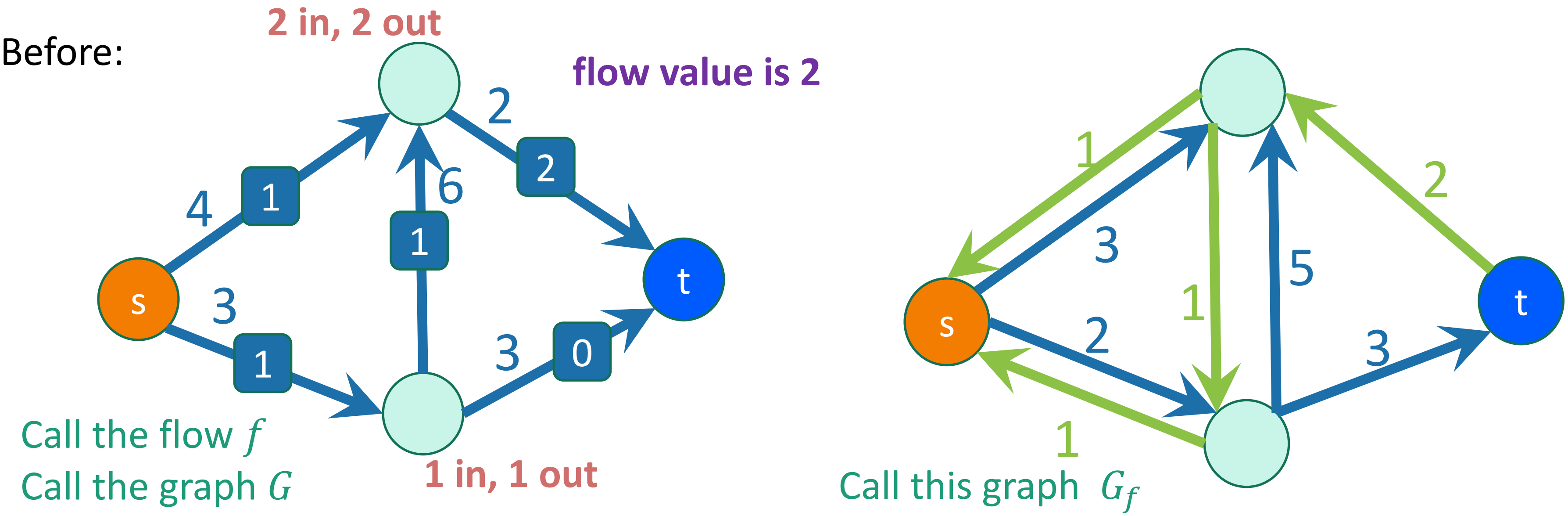
Call the flow f

Call the graph G

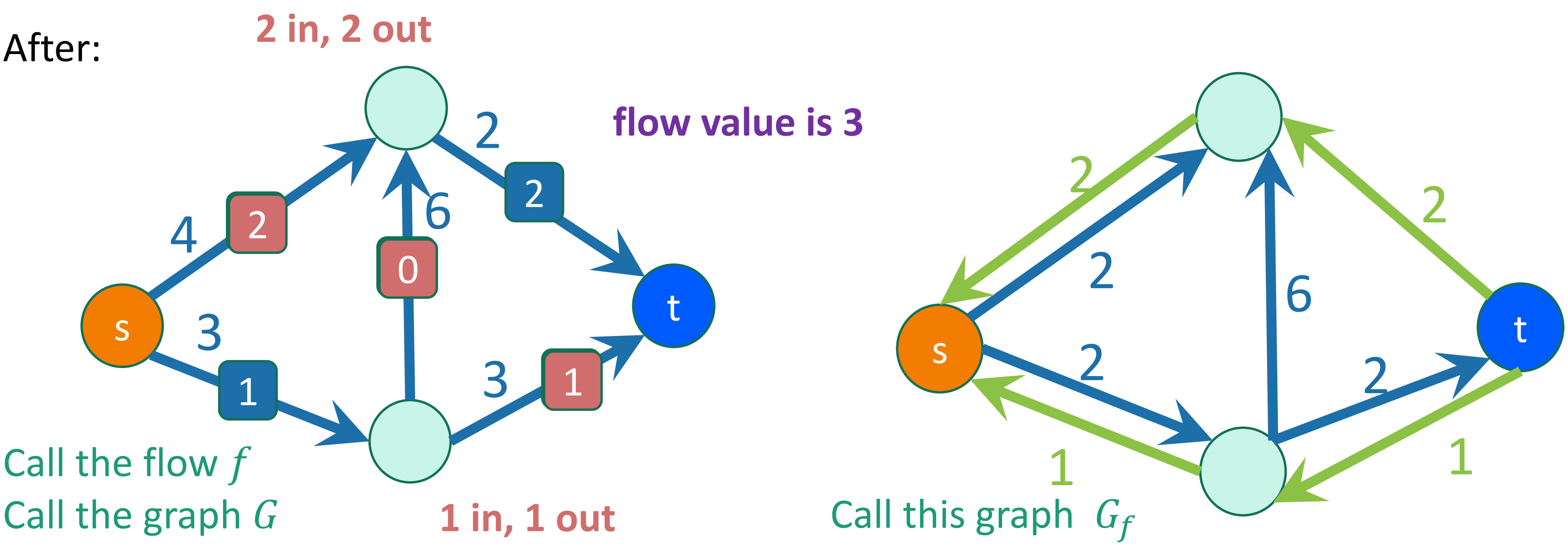


Call this graph G_f

Before:



After:



Still a legit flow, but with a bigger value!

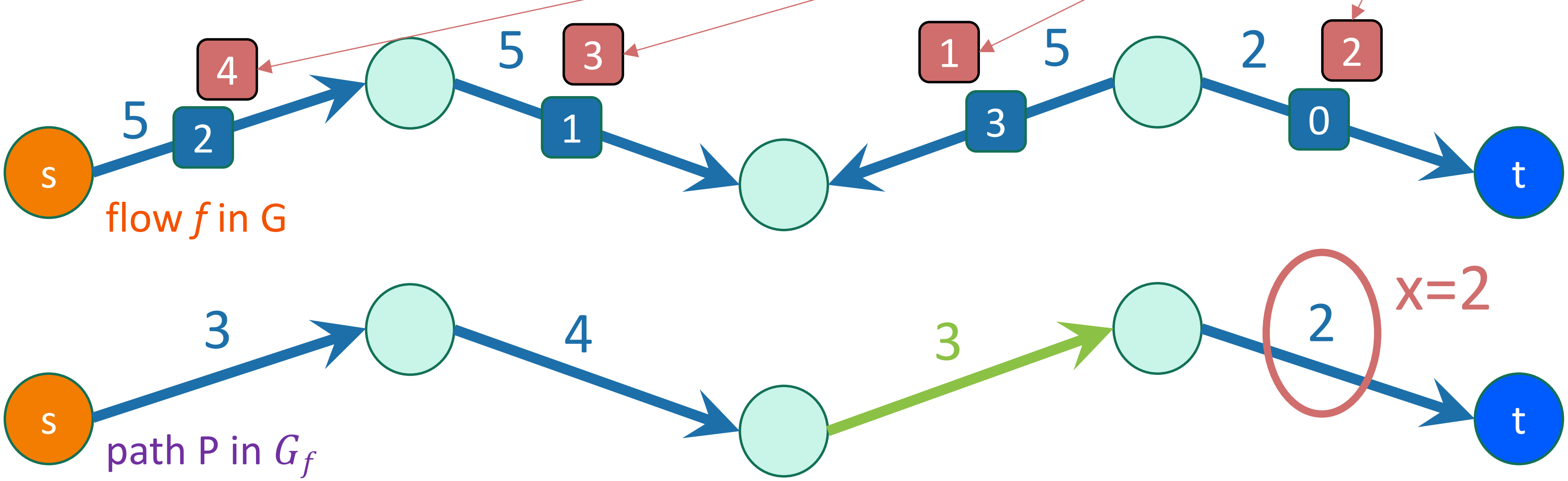
Claim:

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

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

- 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, $\boldsymbol{f'(u,v) \leftarrow f(u,v) + x}$.
    - **if** (v,u) in E, $\boldsymbol{f'(v,u) \leftarrow f(v,u) - x}$
  - **return** f'

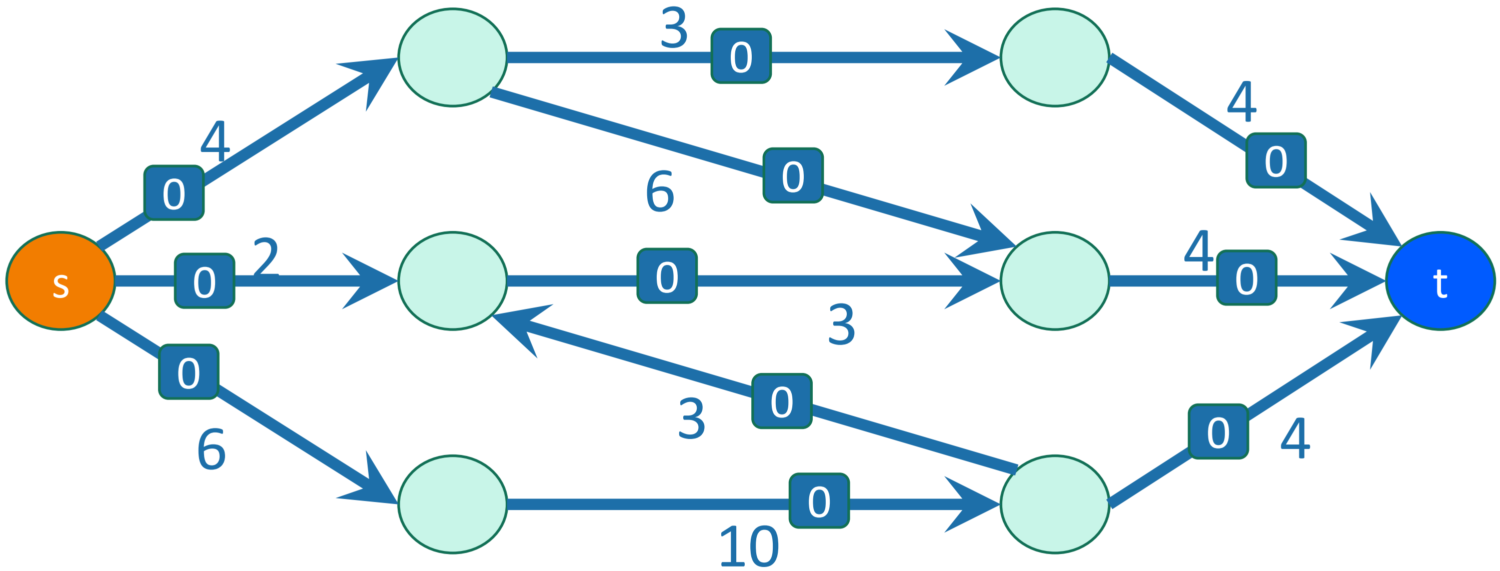
This is f'

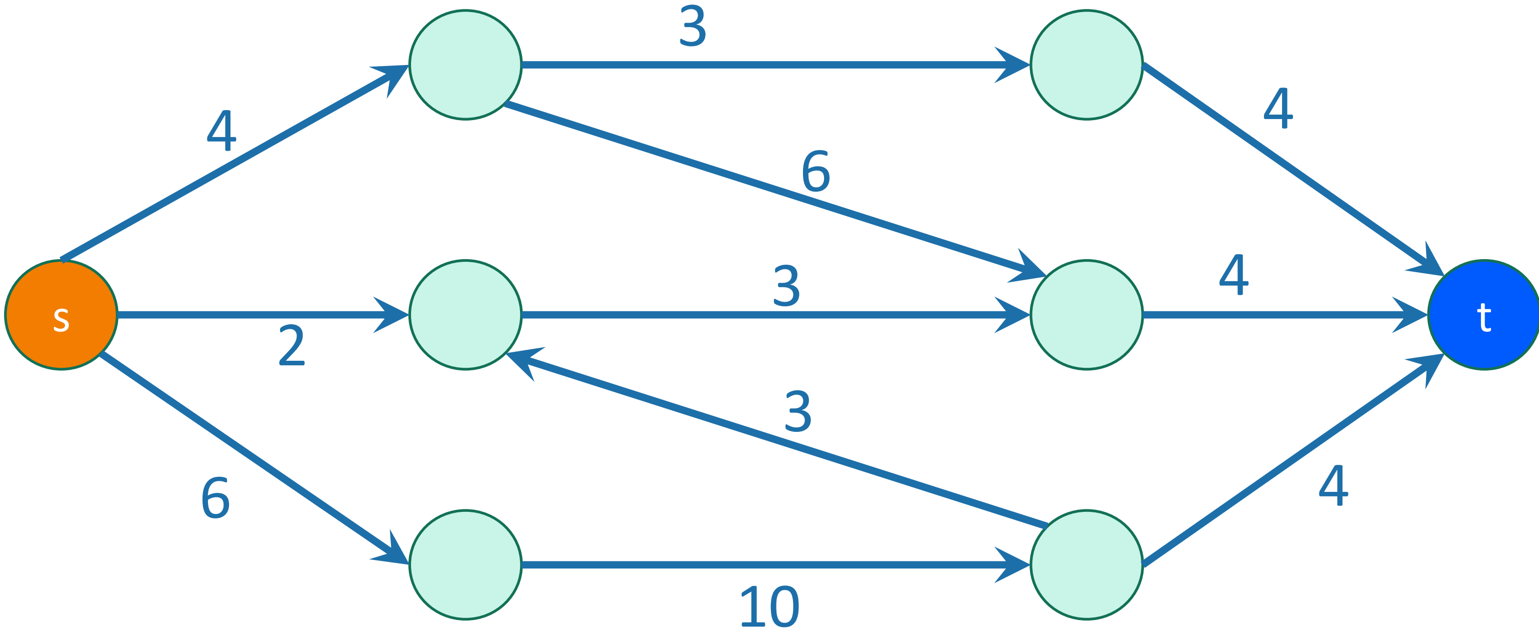


Ford-Fulkerson Algorithm

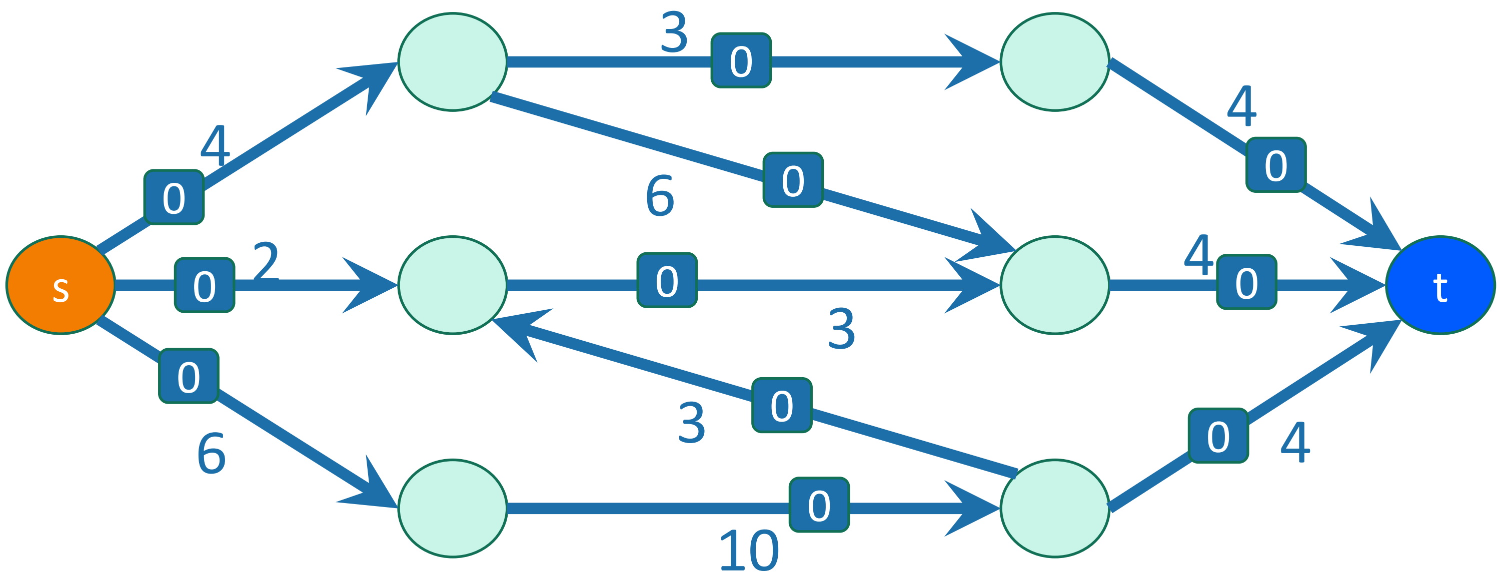
- **Ford-Fulkerson**(G):
  - $f \leftarrow$ all zero flow.
  - $G_f \leftarrow G$
  - **while** t is reachable from s in G_f
    - Find a path P from s to t in G_f // e.g., use DFS or BFS
    - $f \leftarrow$ **increaseFlow**(P,f)
    - update G_f
  - **return** f

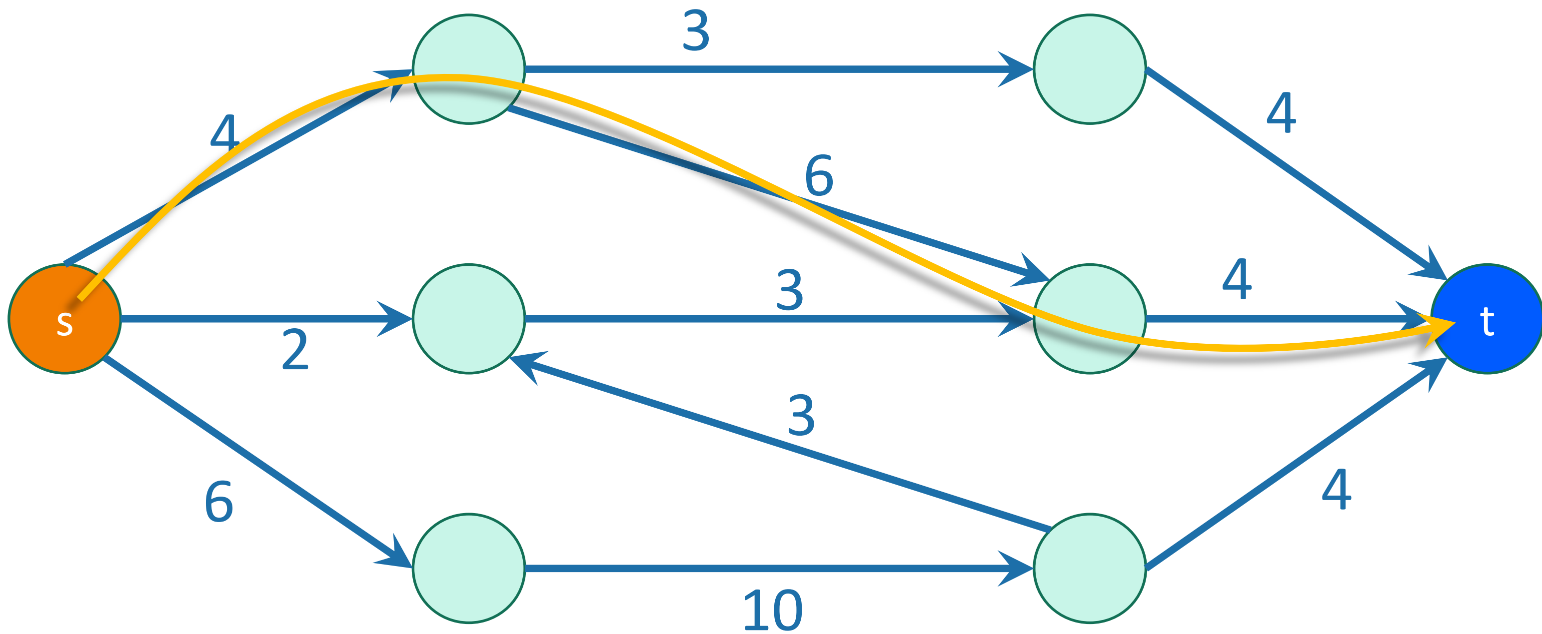
Example of Ford-Fulkerson



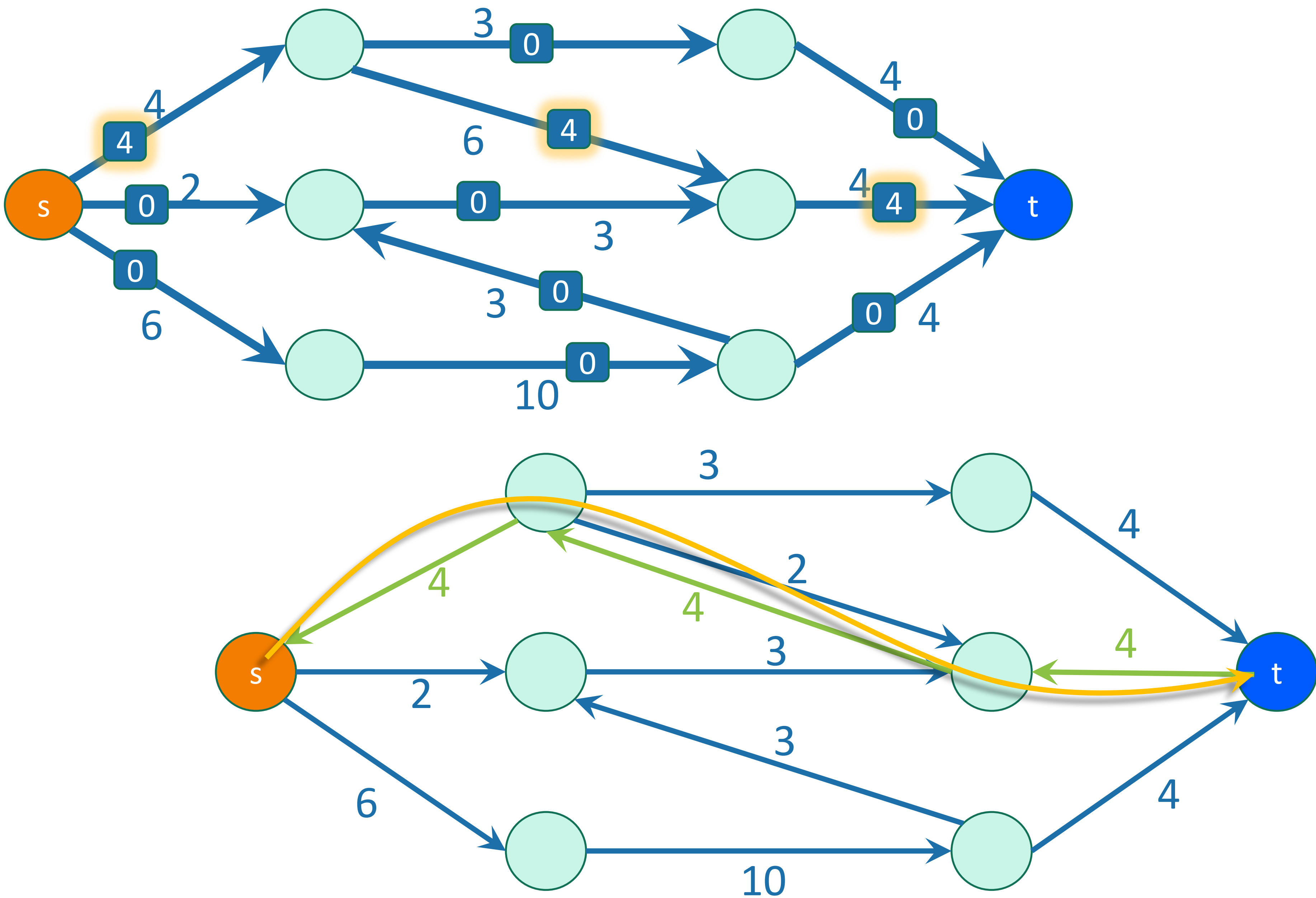


Example of Ford-Fulkerson

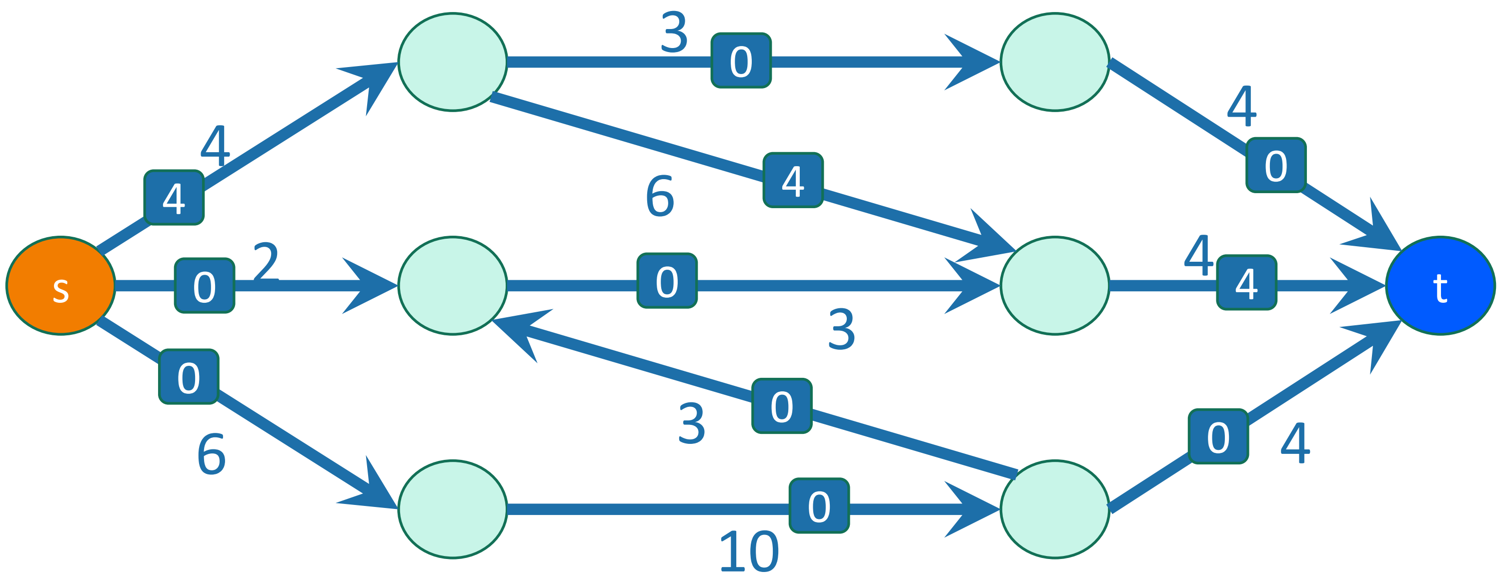


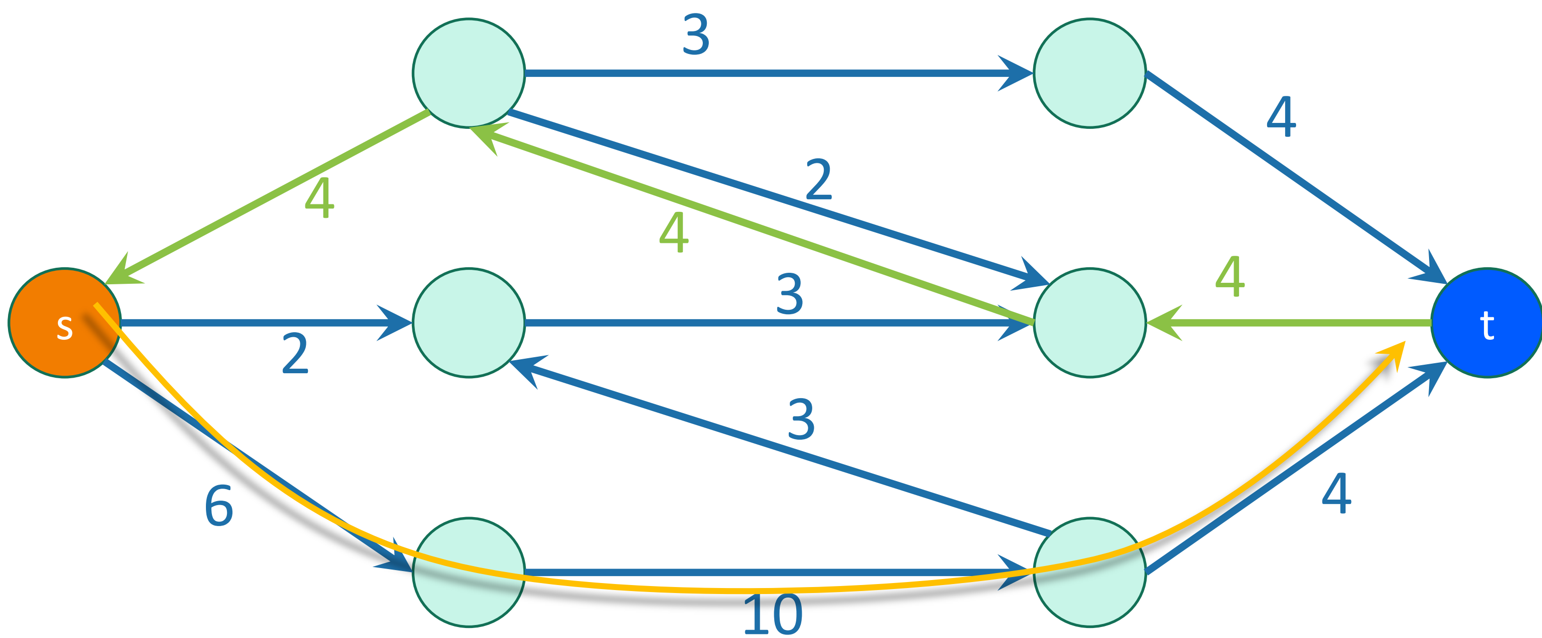


Example of Ford-Fulkerson

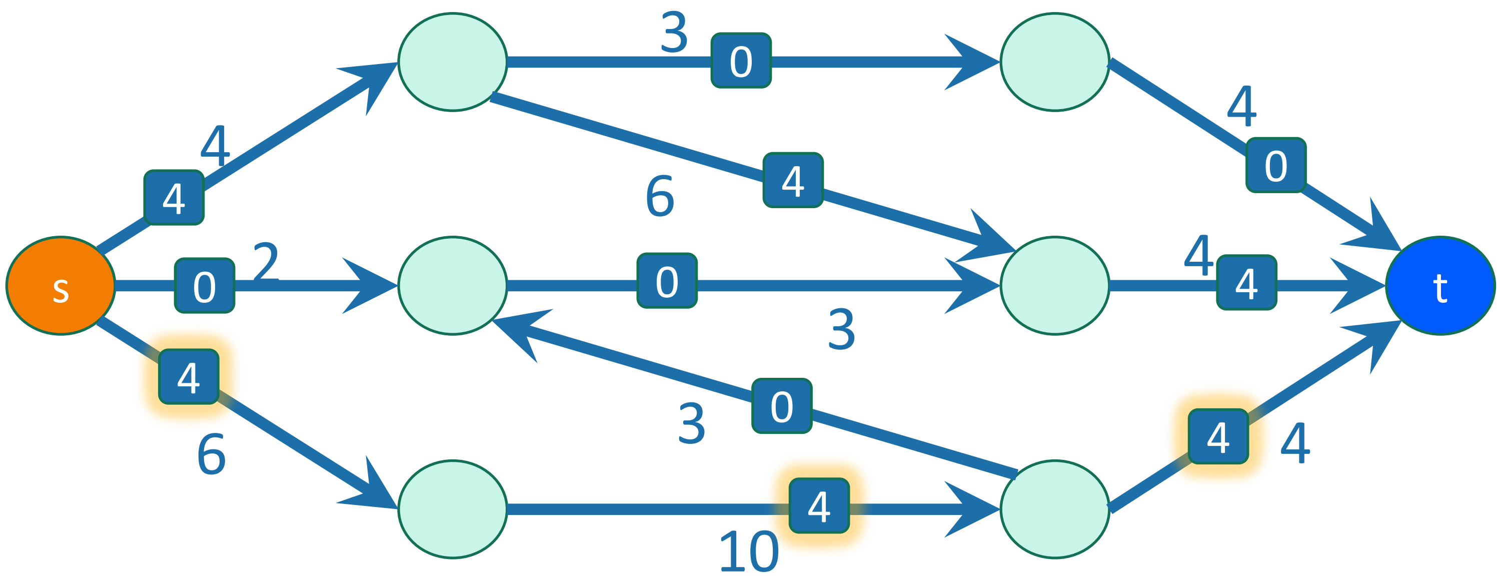


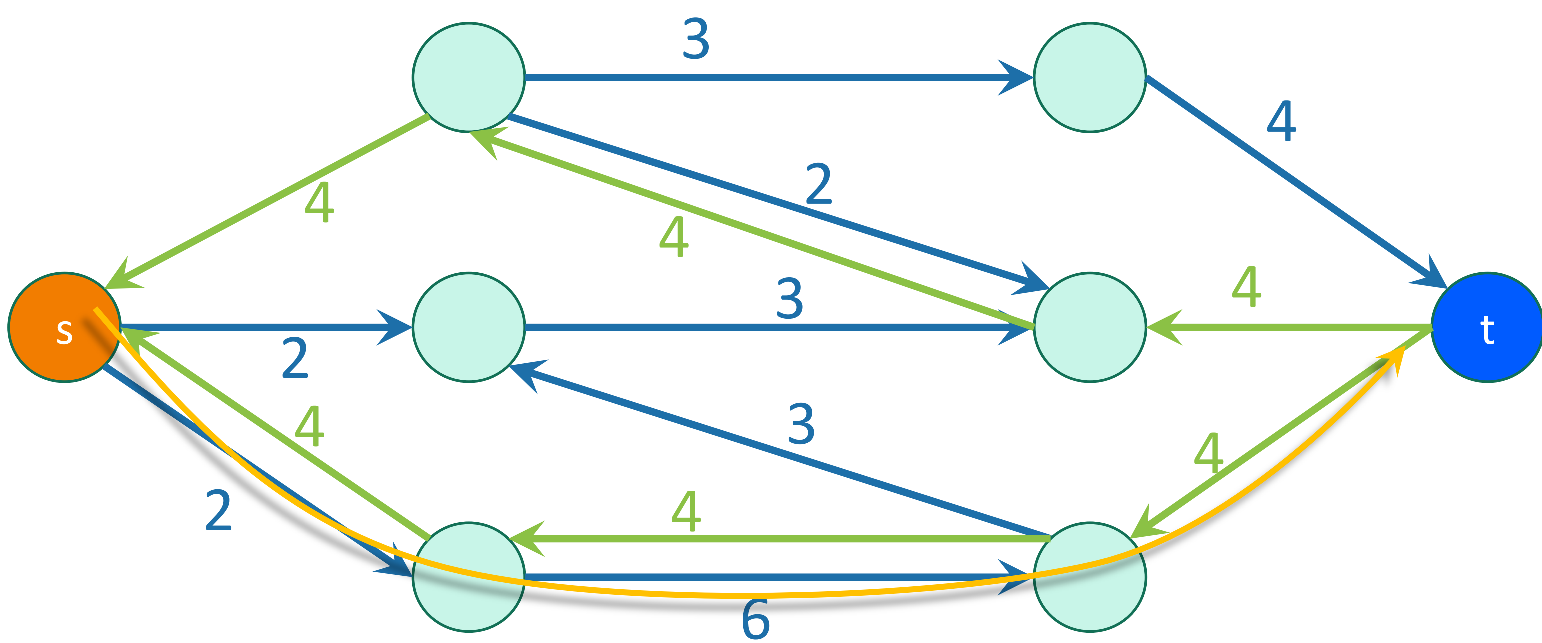
Example of Ford-Fulkerson





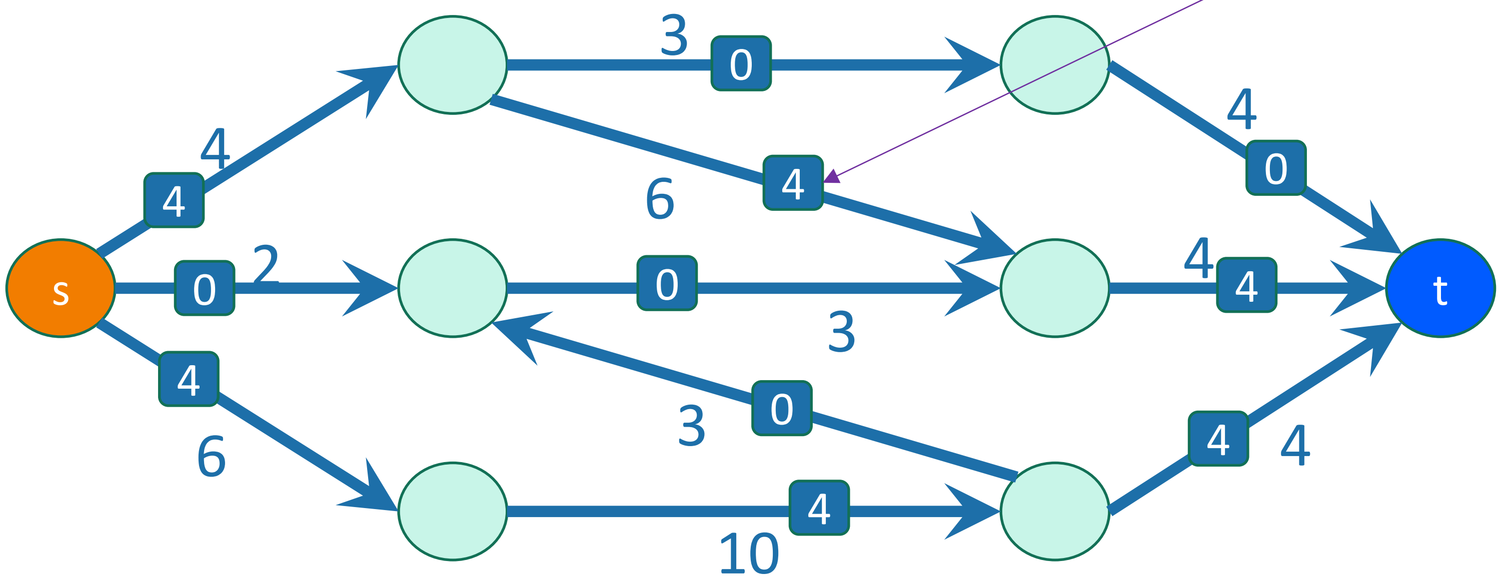
Example of Ford-Fulkerson



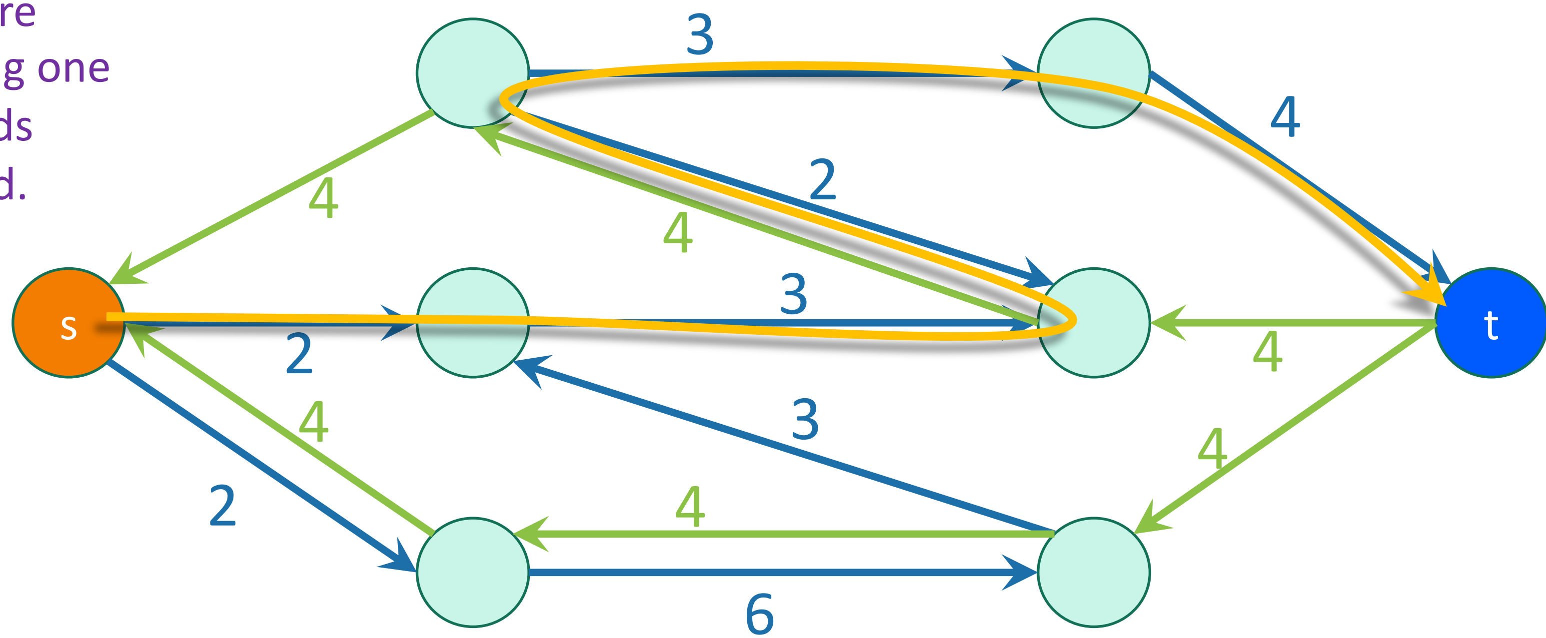


Example of Ford-Fulkerson

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

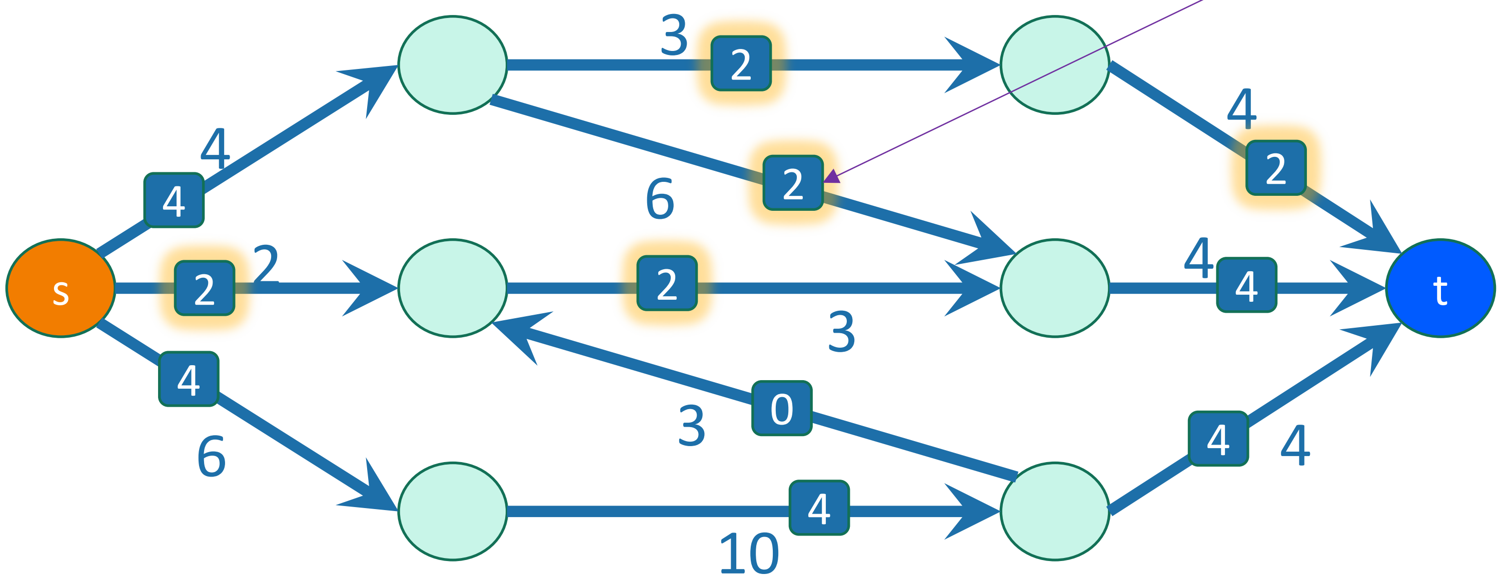


Notice that we're going back along one of the backwards edges we added.

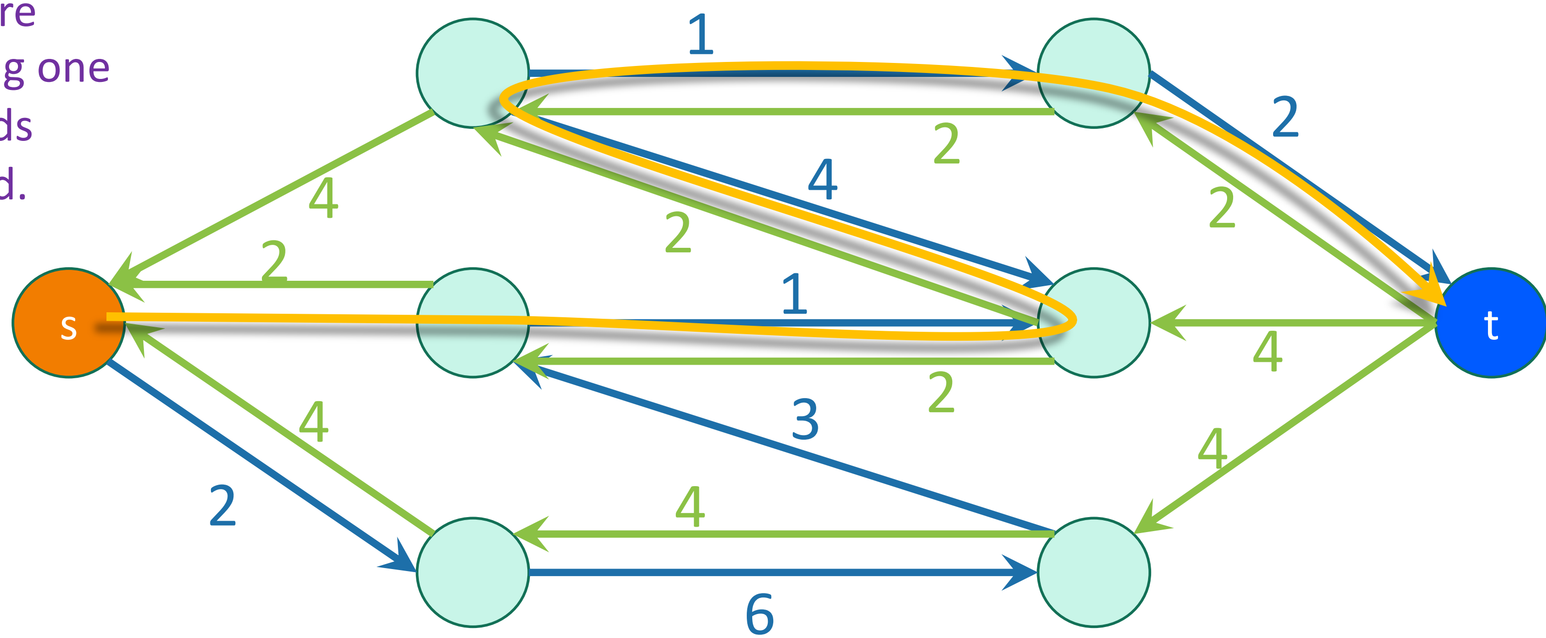


Example of Ford-Fulkerson

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

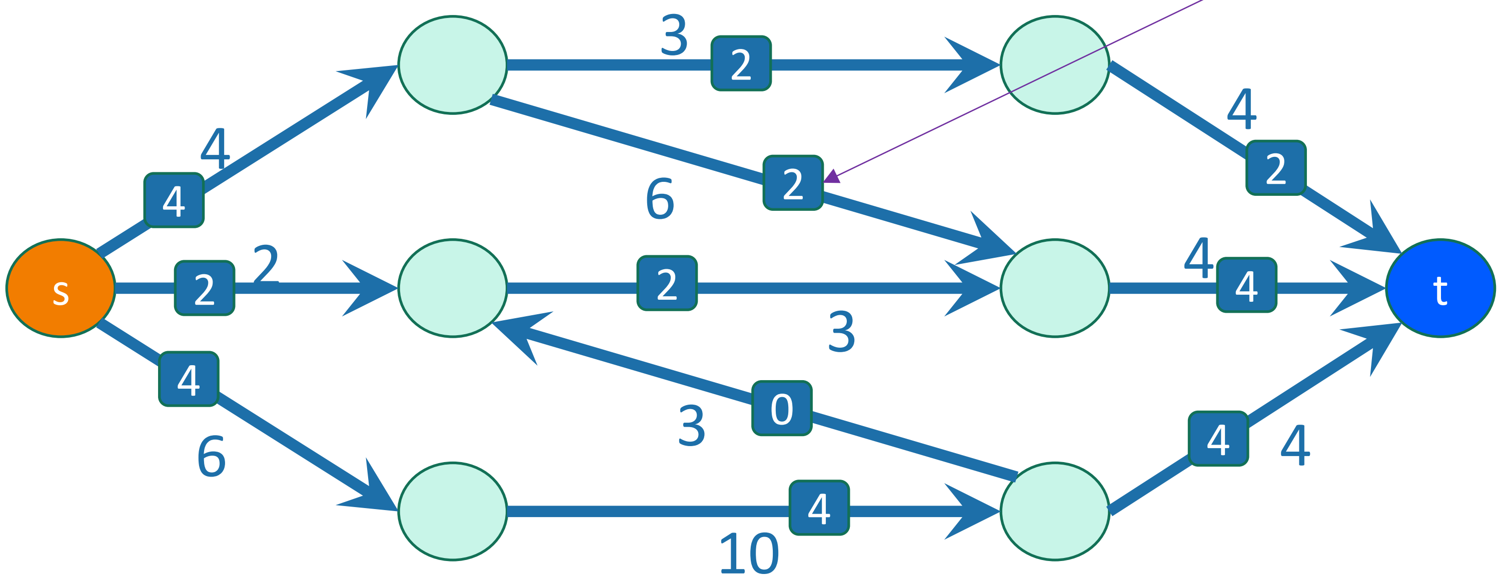


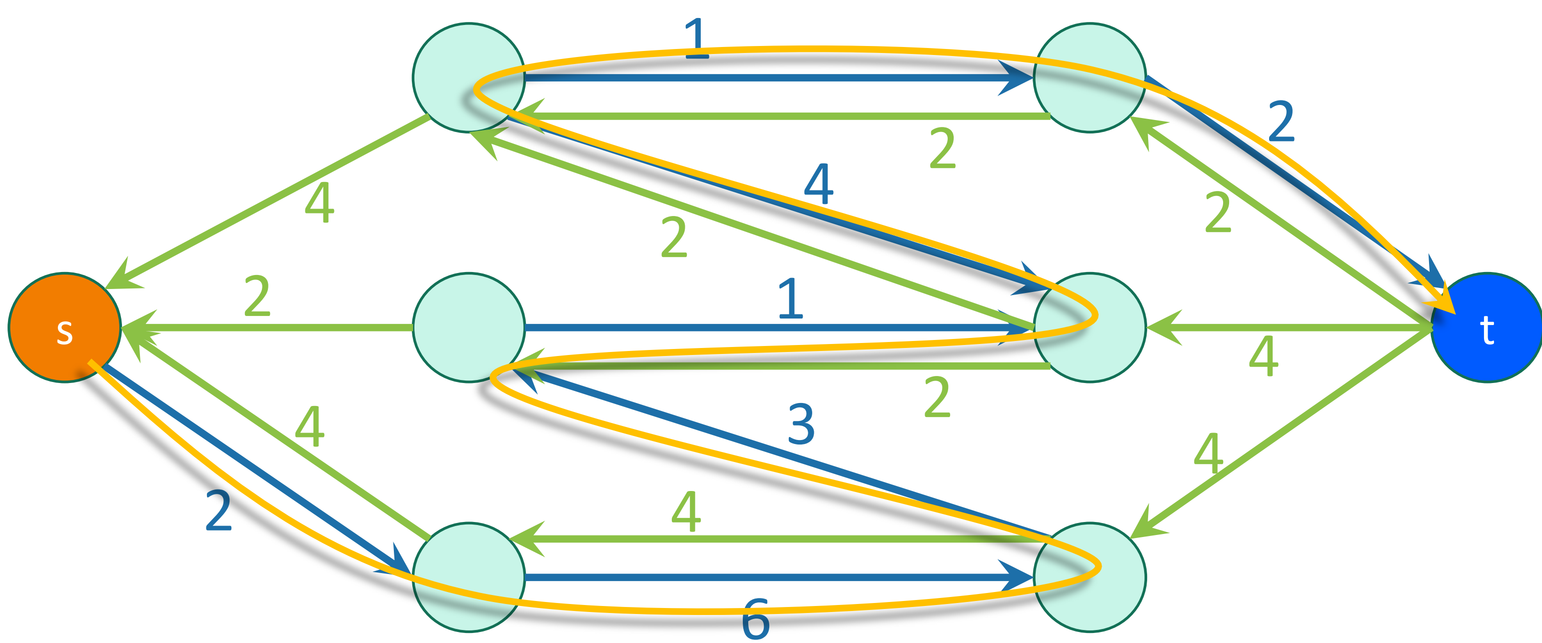
Notice that we're going back along one of the backwards edges we added.



Example of Ford-Fulkerson

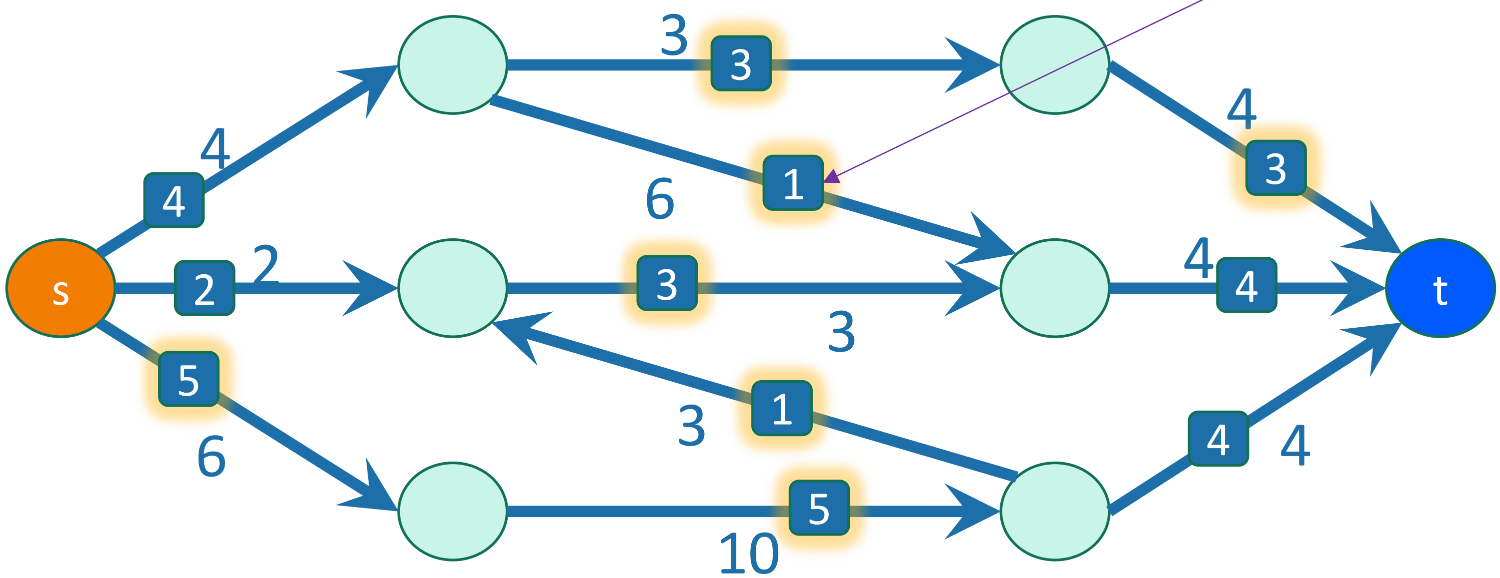
We will remove flow from this edge AGAIN.

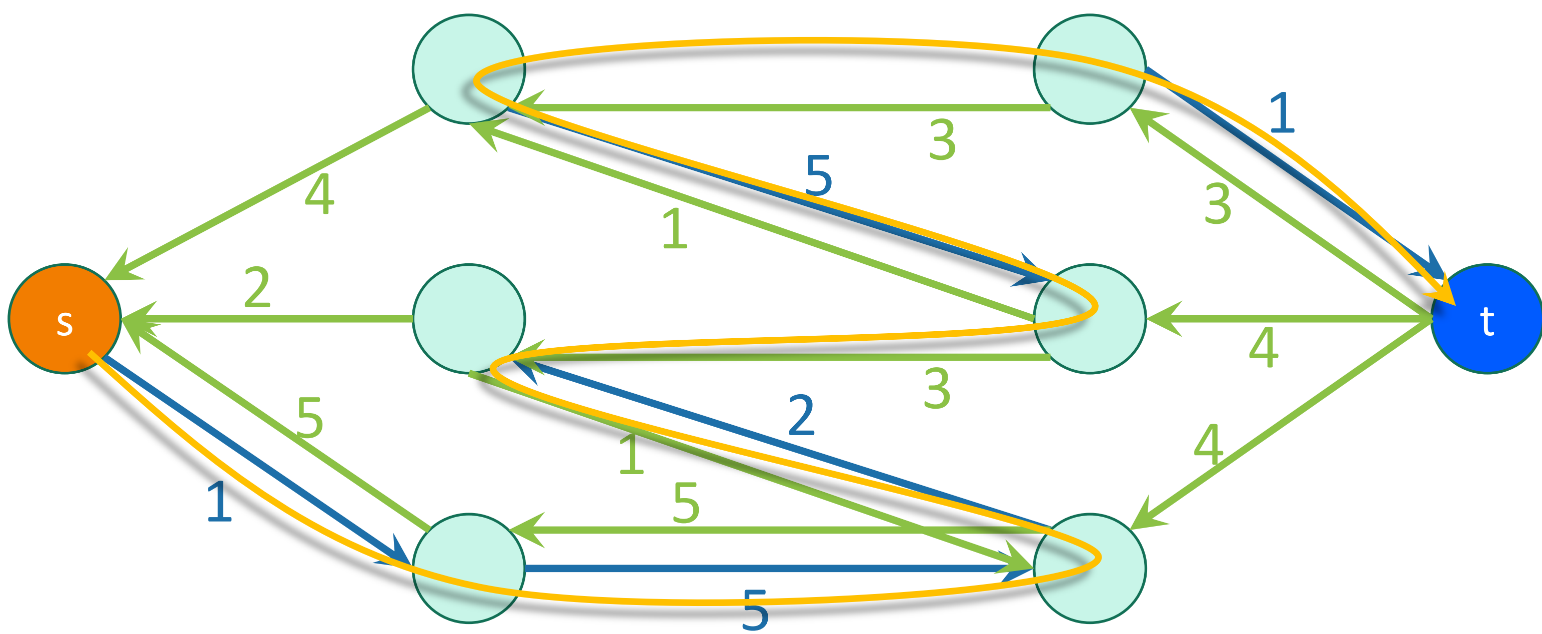




Example of Ford-Fulkerson

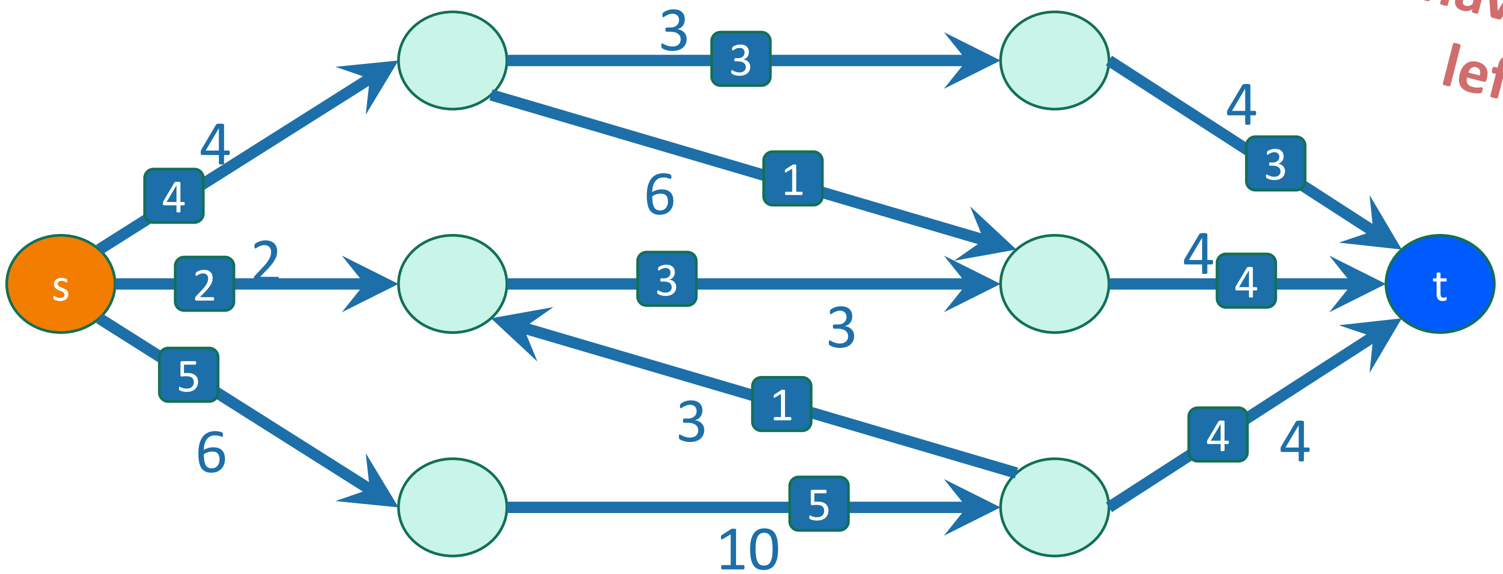
We will remove flow from this edge AGAIN.

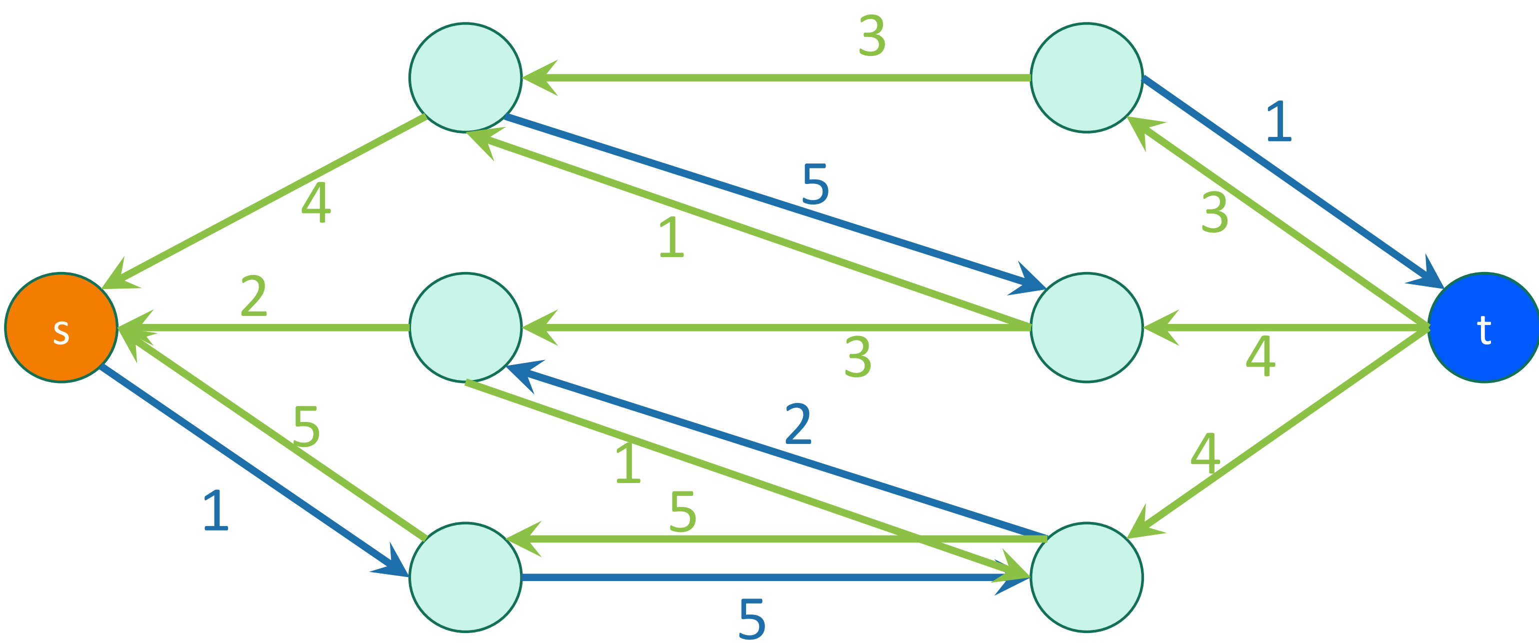




Example of Ford-Fulkerson

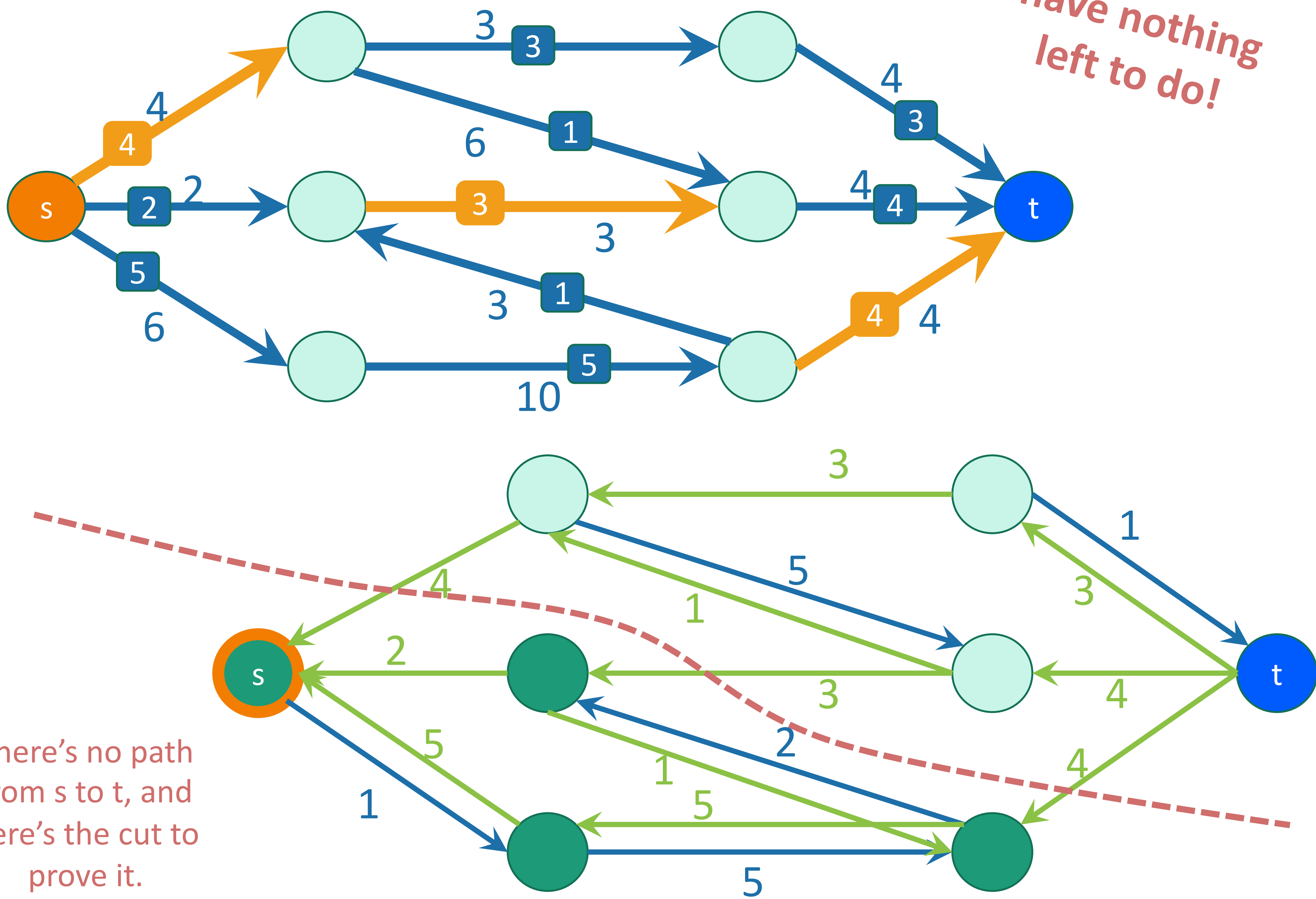
Now we have nothing left to do!





Example of Ford-Fulkerson

Now we have nothing left to do!



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

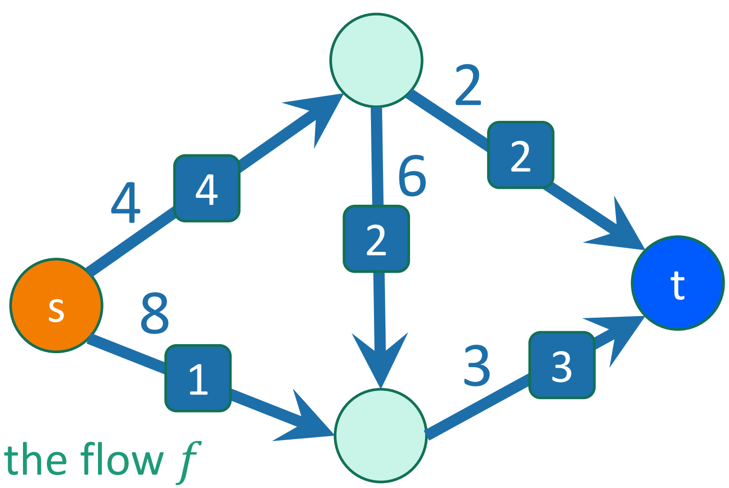
Why does Ford-Fulkerson work?

- 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.

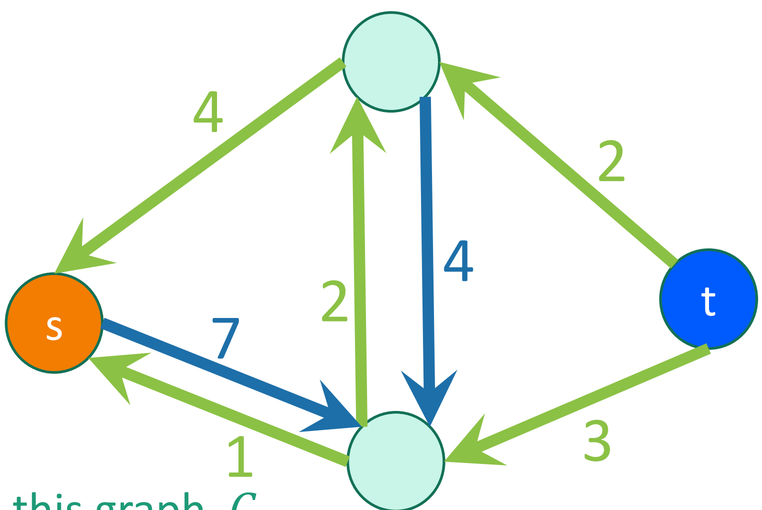
No augmenting path ⇒ max flow.

- 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

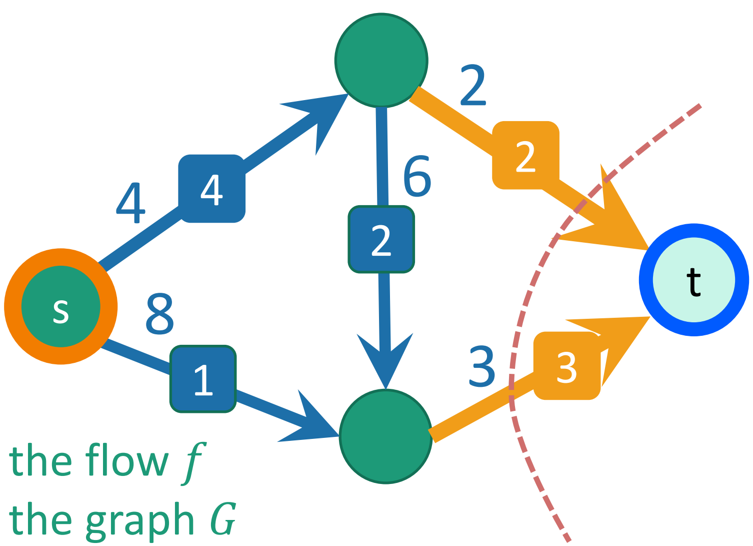
No augmenting path ⇒ max flow.

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

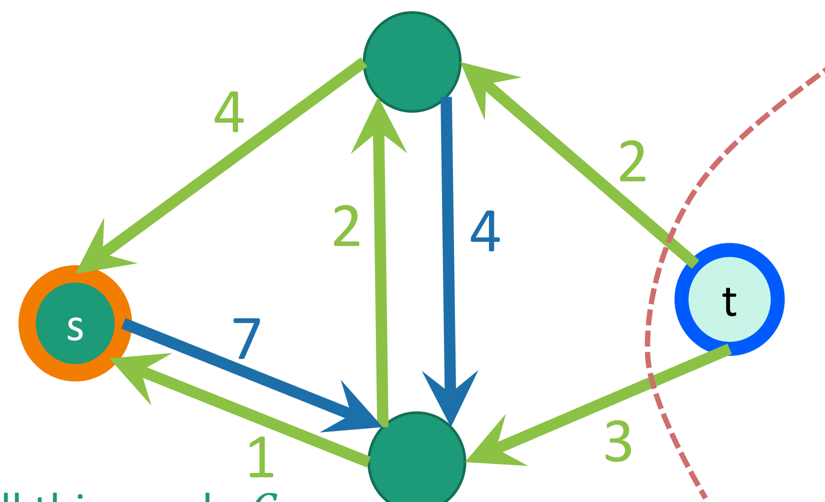
t lives here

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

- The 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



Call the flow f
Call the graph G



Call this graph G_f

No augmenting path ⇒ max flow.

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

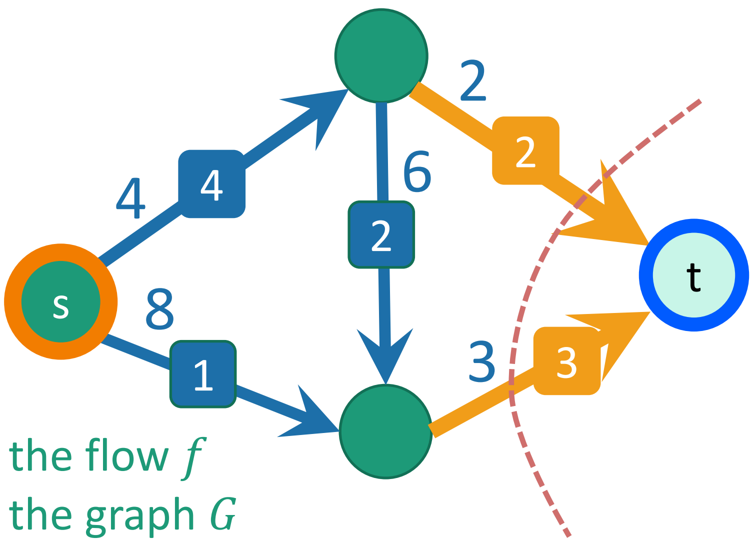
t lives here

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

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

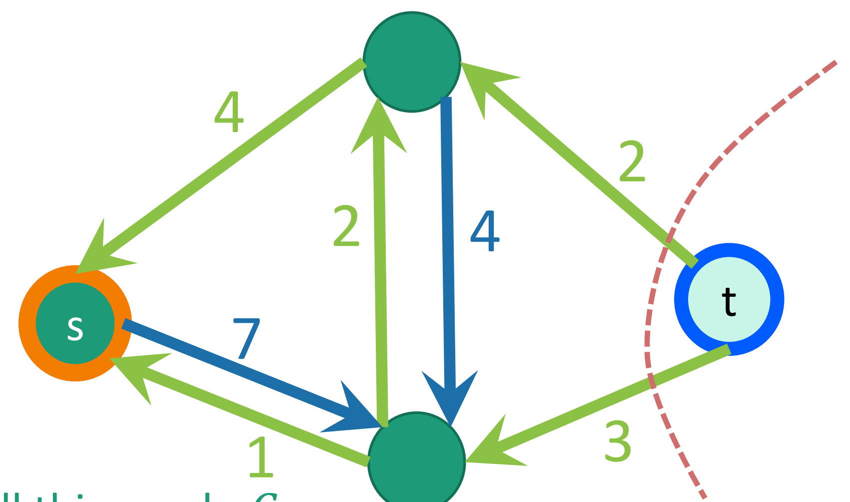
Lemma 1

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



Call the flow f

Call the graph G



Call this graph G_f

No augmenting path ⇒ max flow. ✓

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

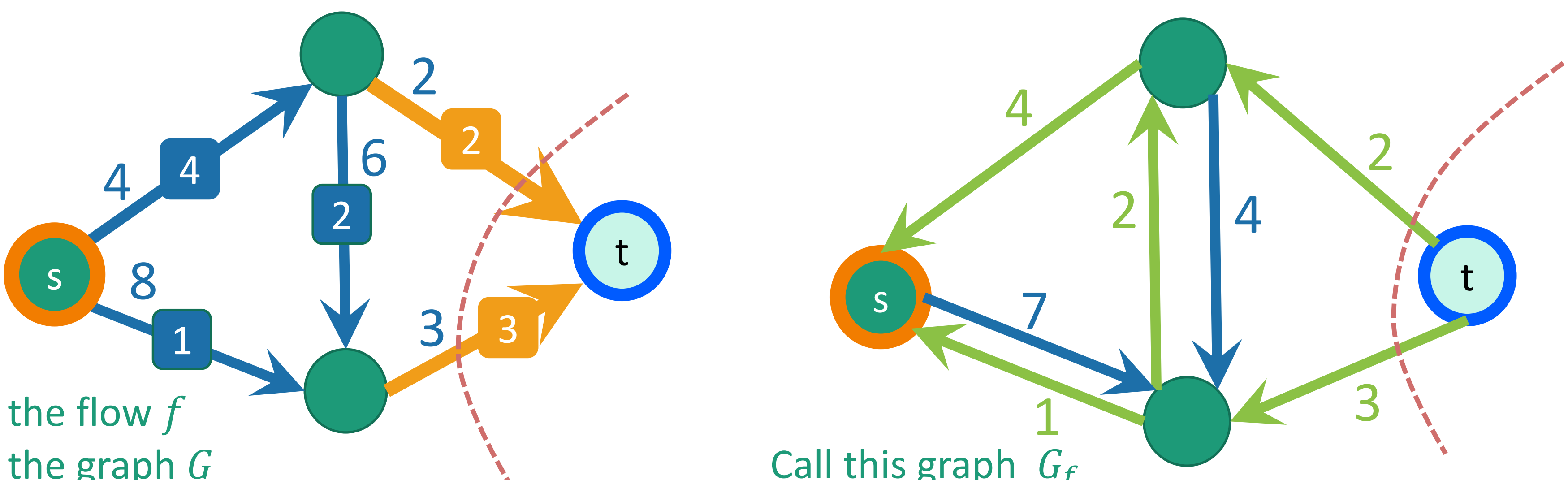
  t lives here

  {things reachable from s} , {things not reachable from s}
- The value of the flow f from s to t is **equal** to the cost of this cut.

Lemma 1

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

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



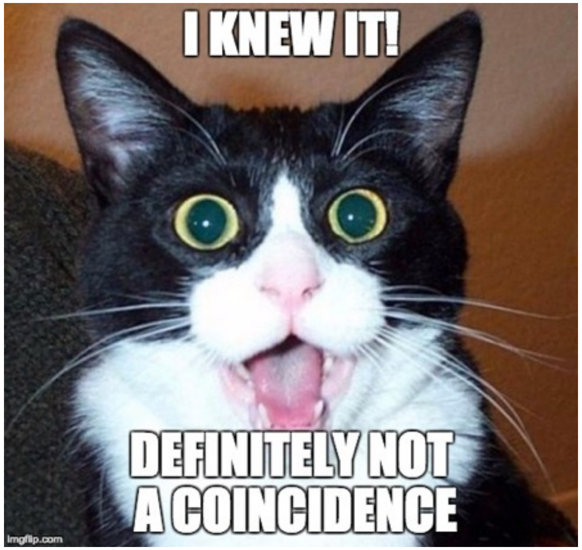
Call the flow f
Call the graph G

Call this graph G_f

Min-Cut Max-Flow Theorem

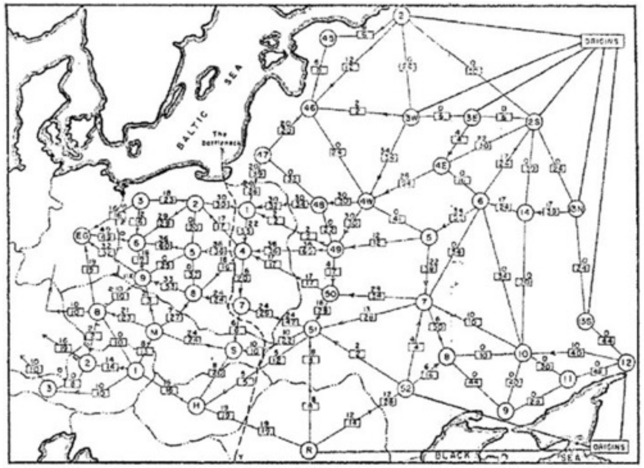
max flow $\geq$ Value of f = cost of this cut $\geq$ min cut $\geq$ max flow

So everything is equal and min cut = max flow!



What have we learned?

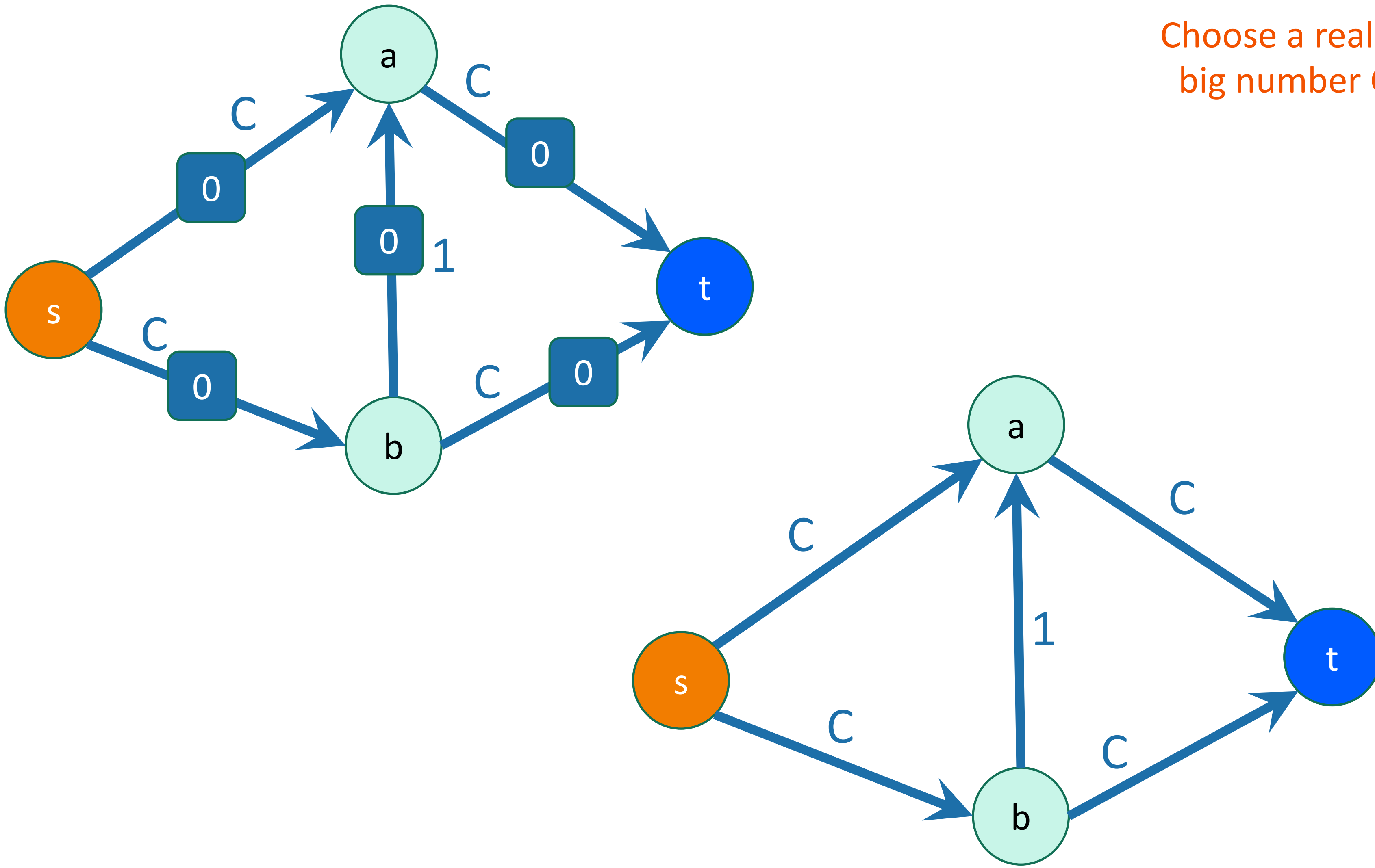
- Max s-t flow is equal to min s-t cut!
  - The USSR and the USA were trying to solve the same problem…
- The Ford-Fulkerson algorithm can find the min-cut/max-flow.
  - Repeatedly improve your flow along an augmenting path.
- **How long does this take???**



Why should we be concerned?

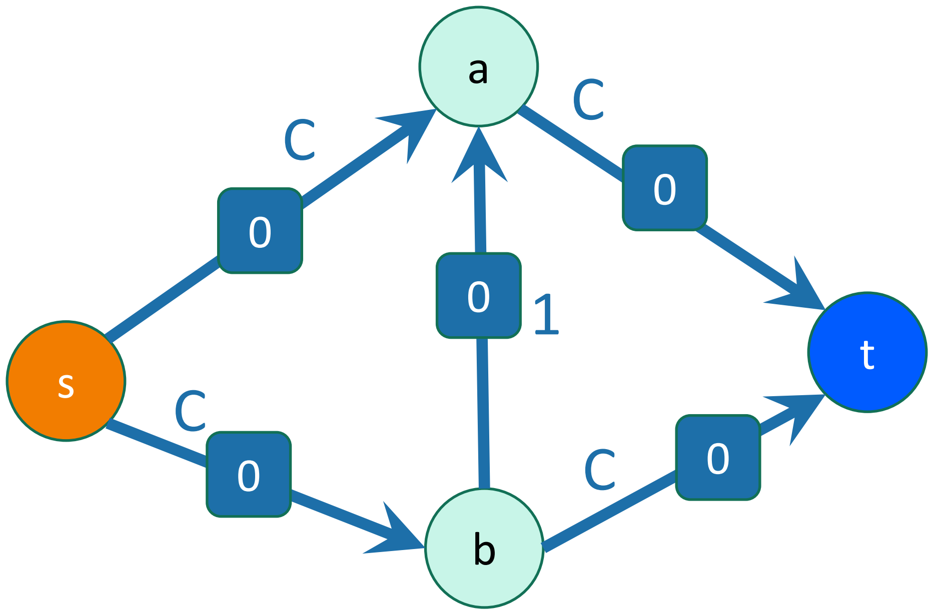
Suppose we just picked paths arbitrarily.

Choose a really big number C.

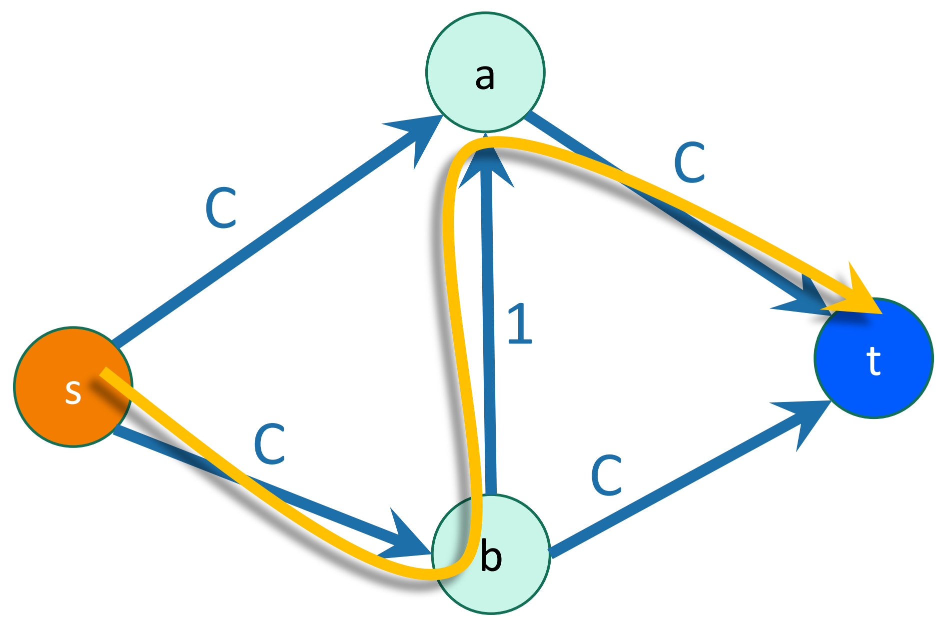


Why should we be concerned?

Suppose we just picked paths arbitrarily.



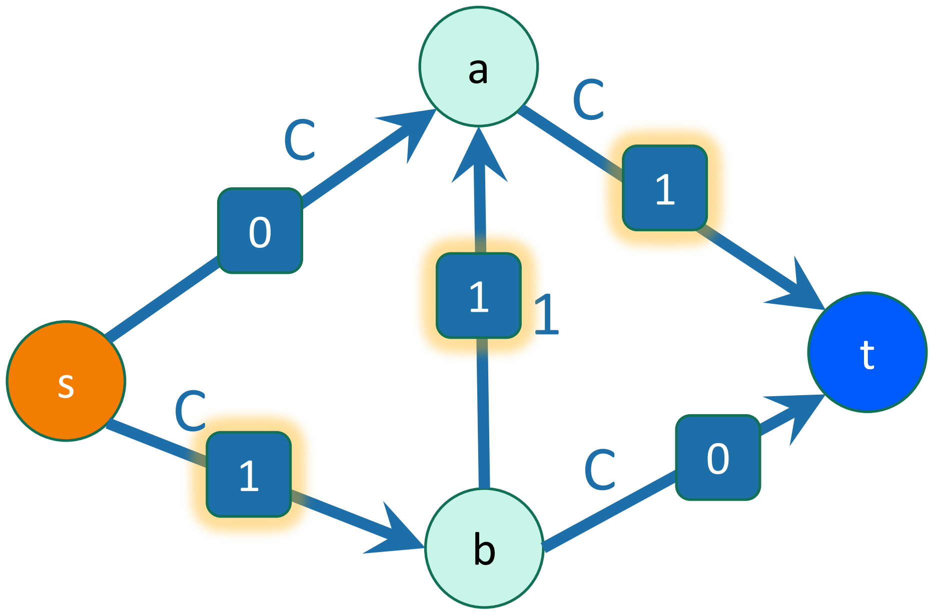
Choose a really big number C.



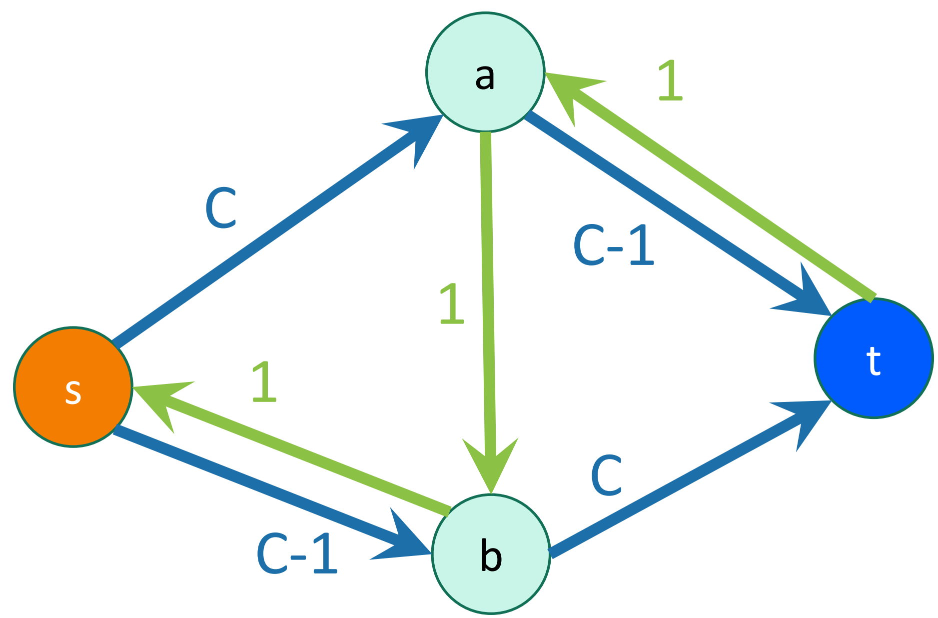
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C.



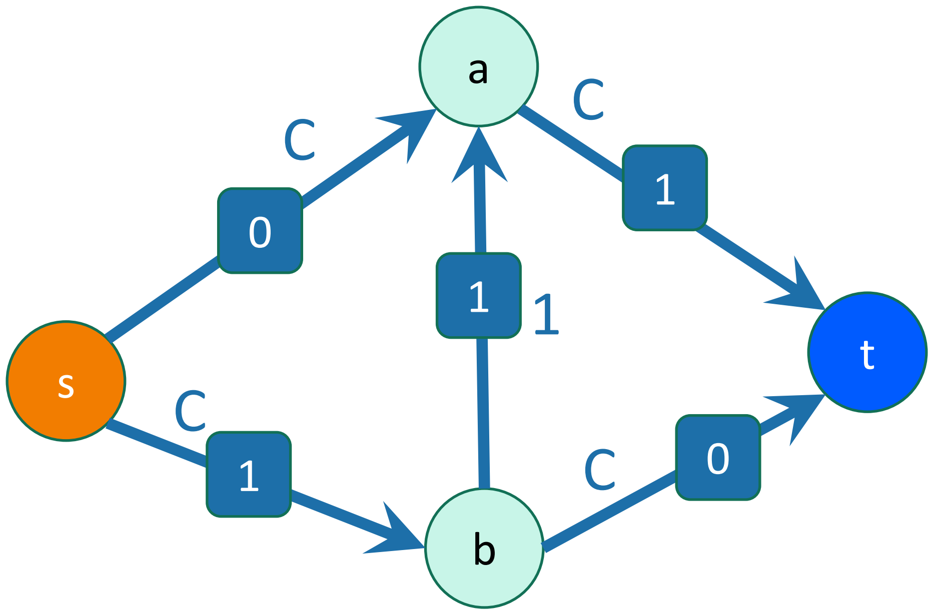
The edge (b,a) disappeared from the residual graph!

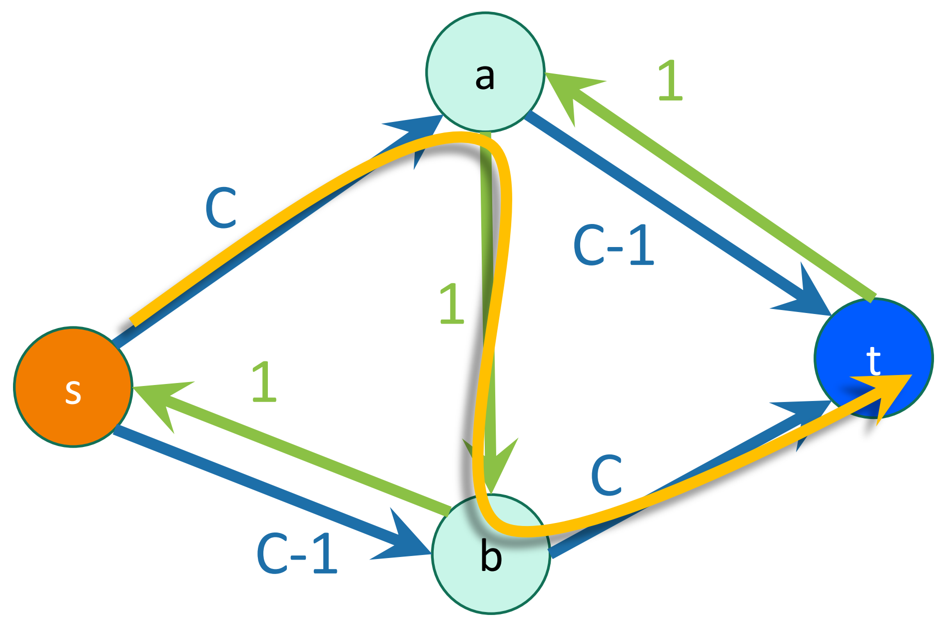


Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C.

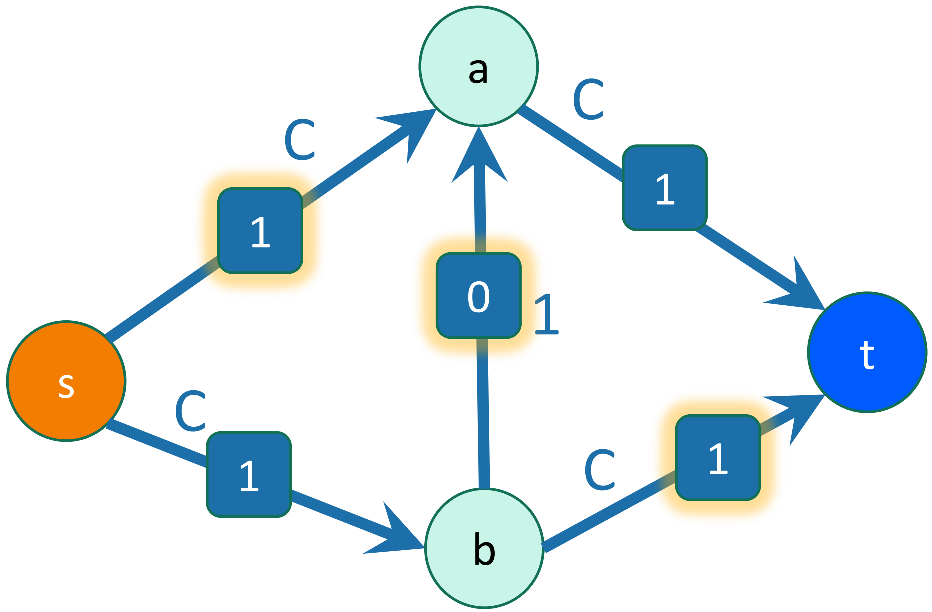


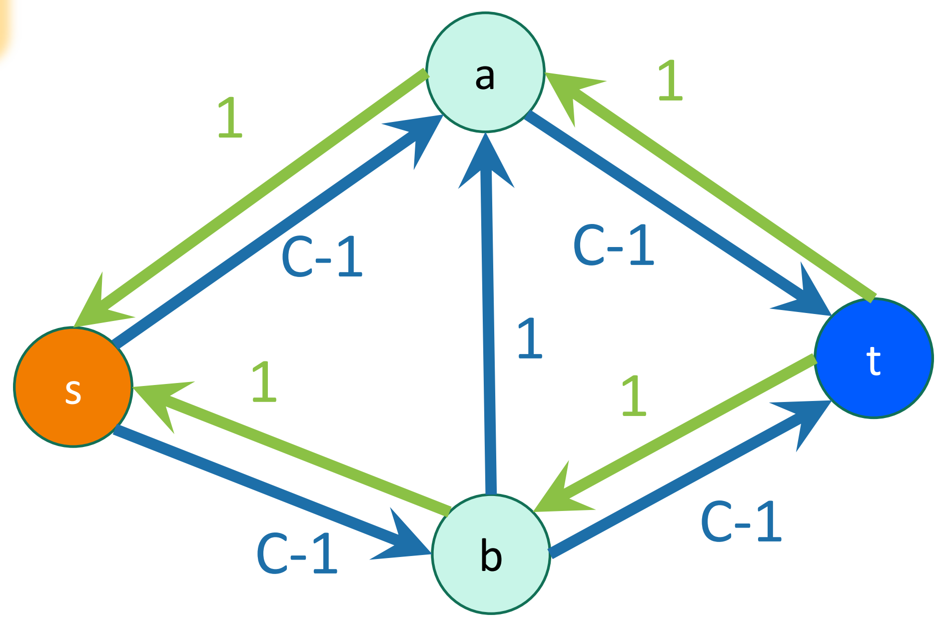


Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C.



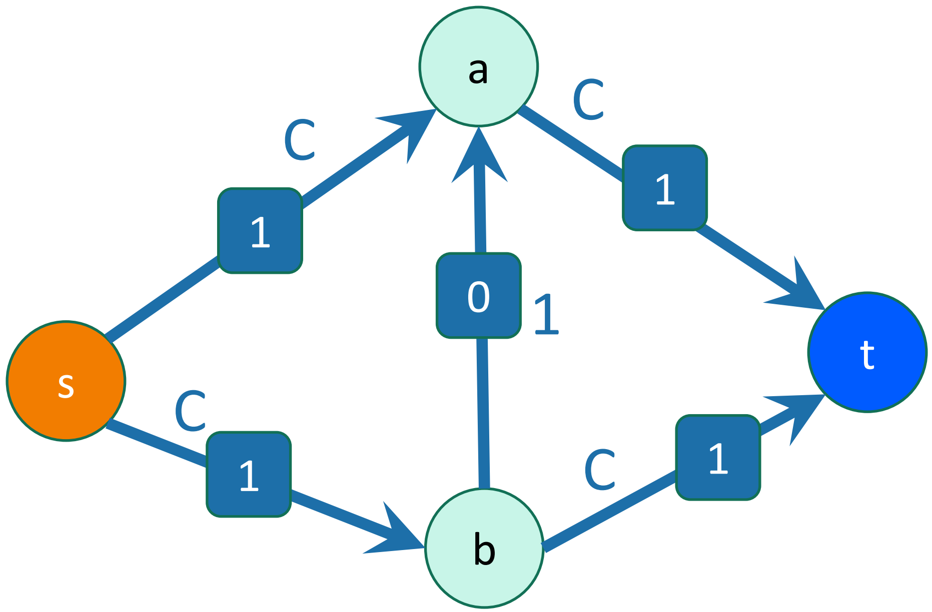


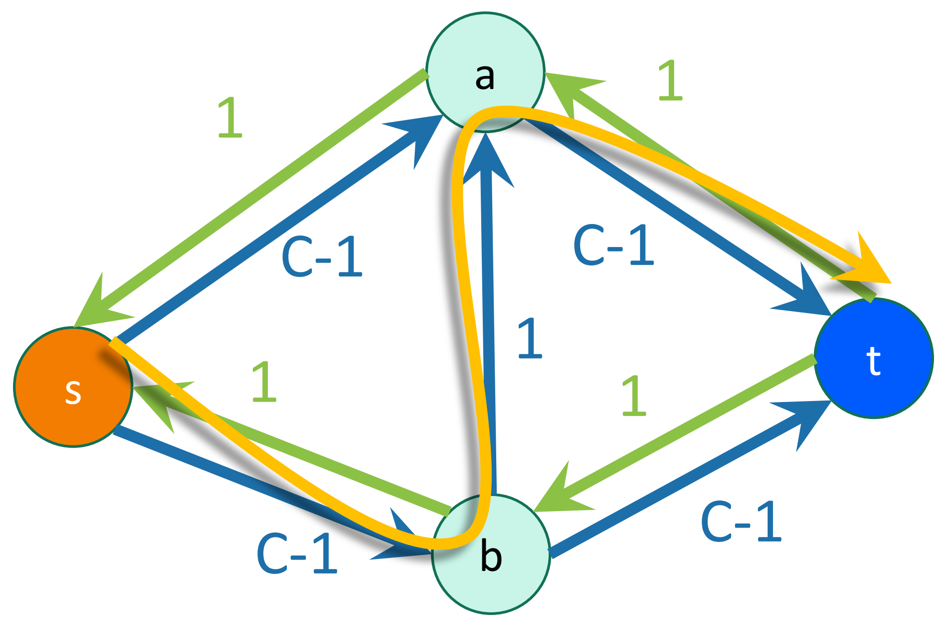
The edge (b,a) re-appeared in the residual graph!

Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C.

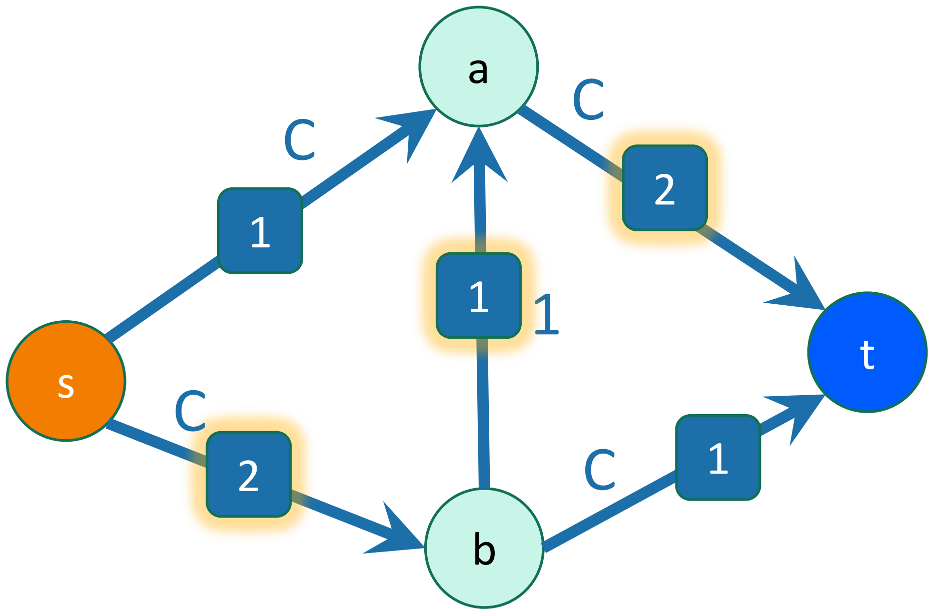




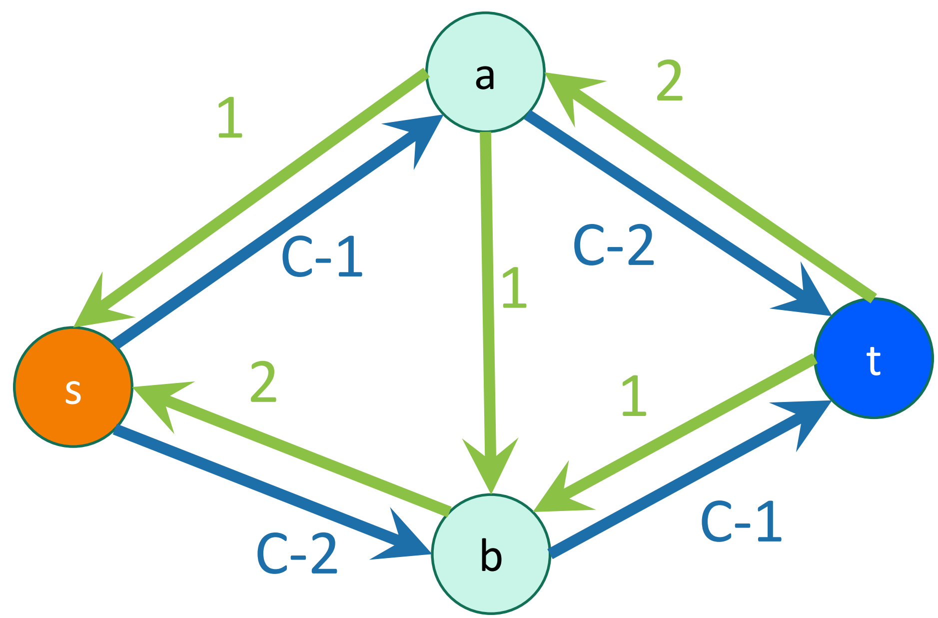
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really
big number C.



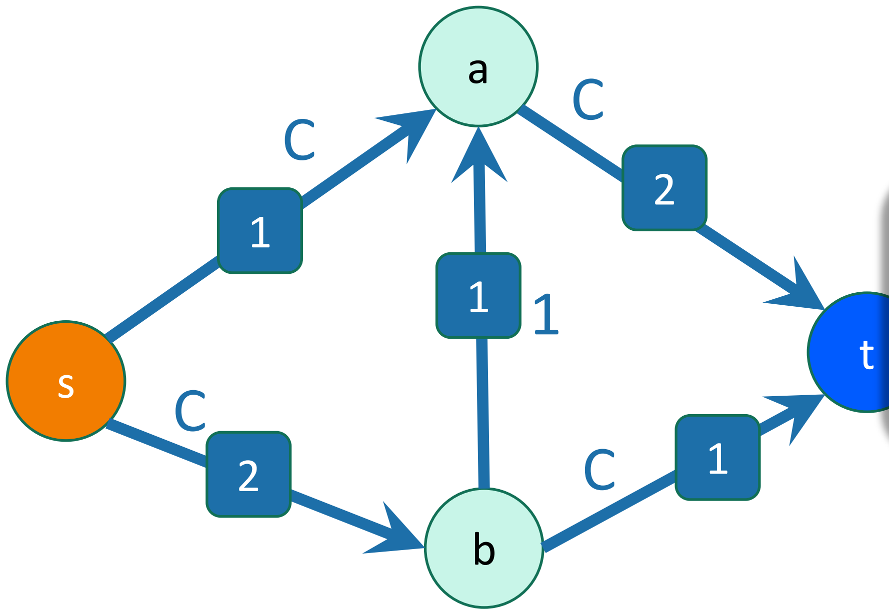
The edge (b,a) disappeared
from the residual graph!



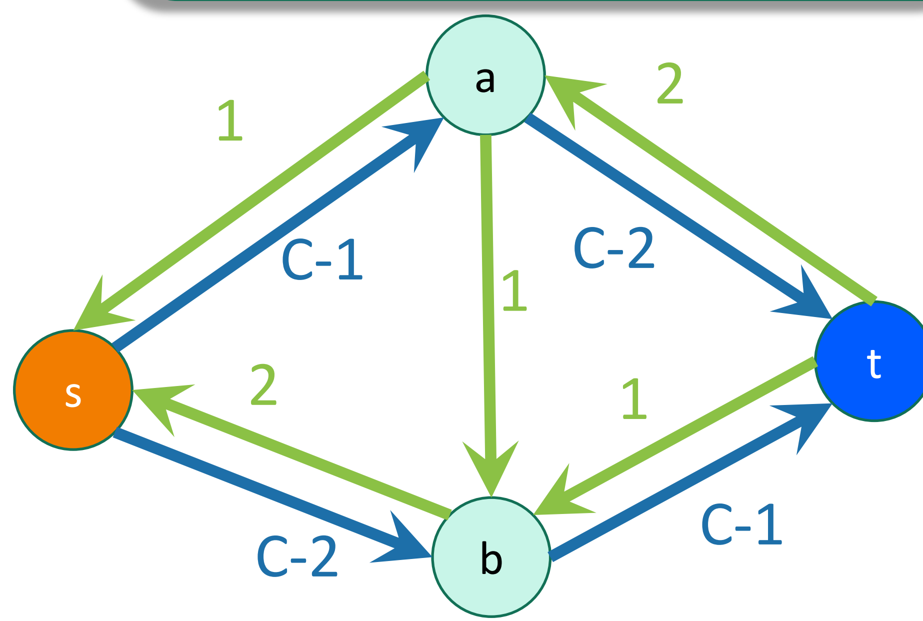
Why should we be concerned?

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.

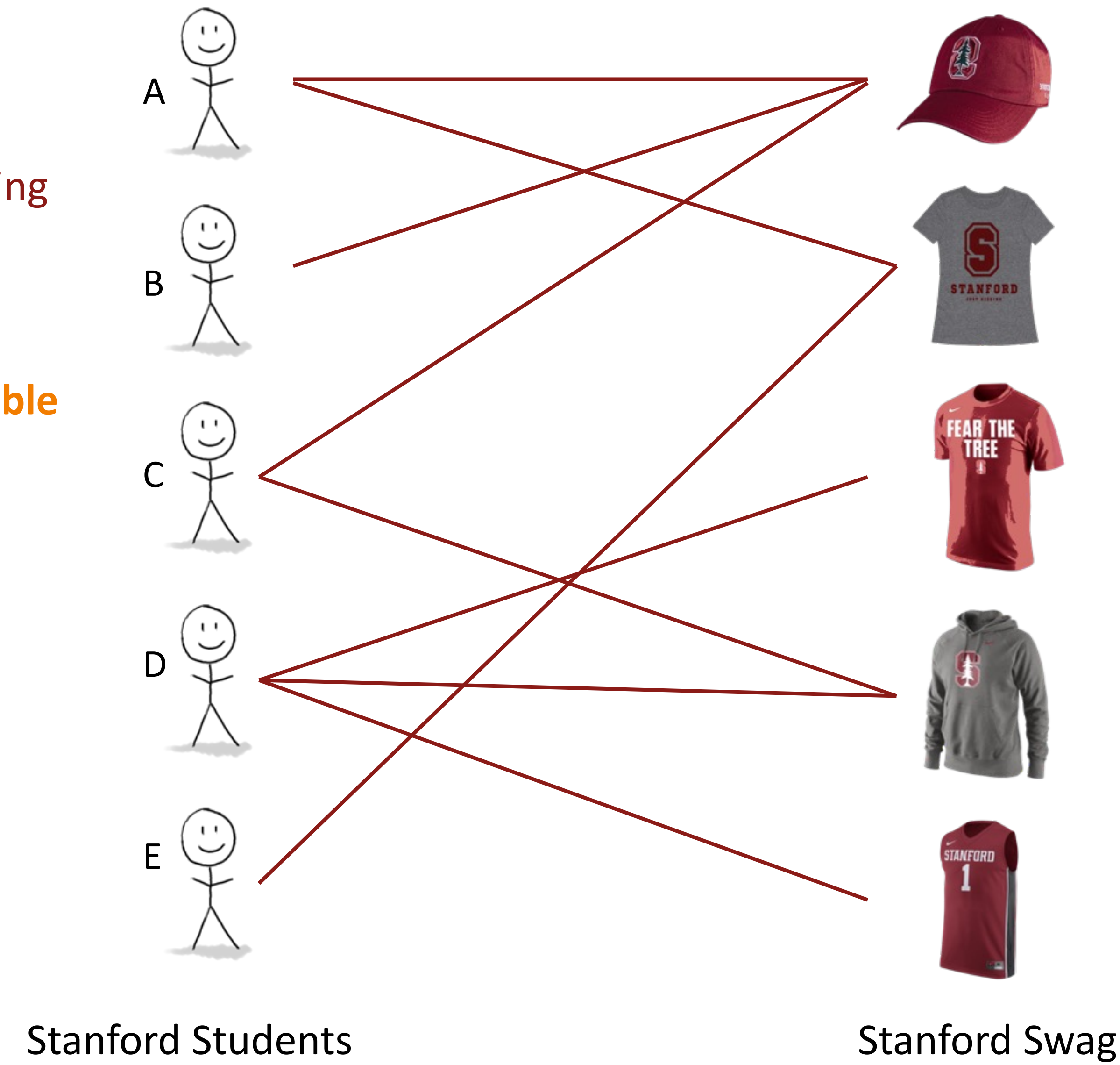
But wait, there's more!

- 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

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

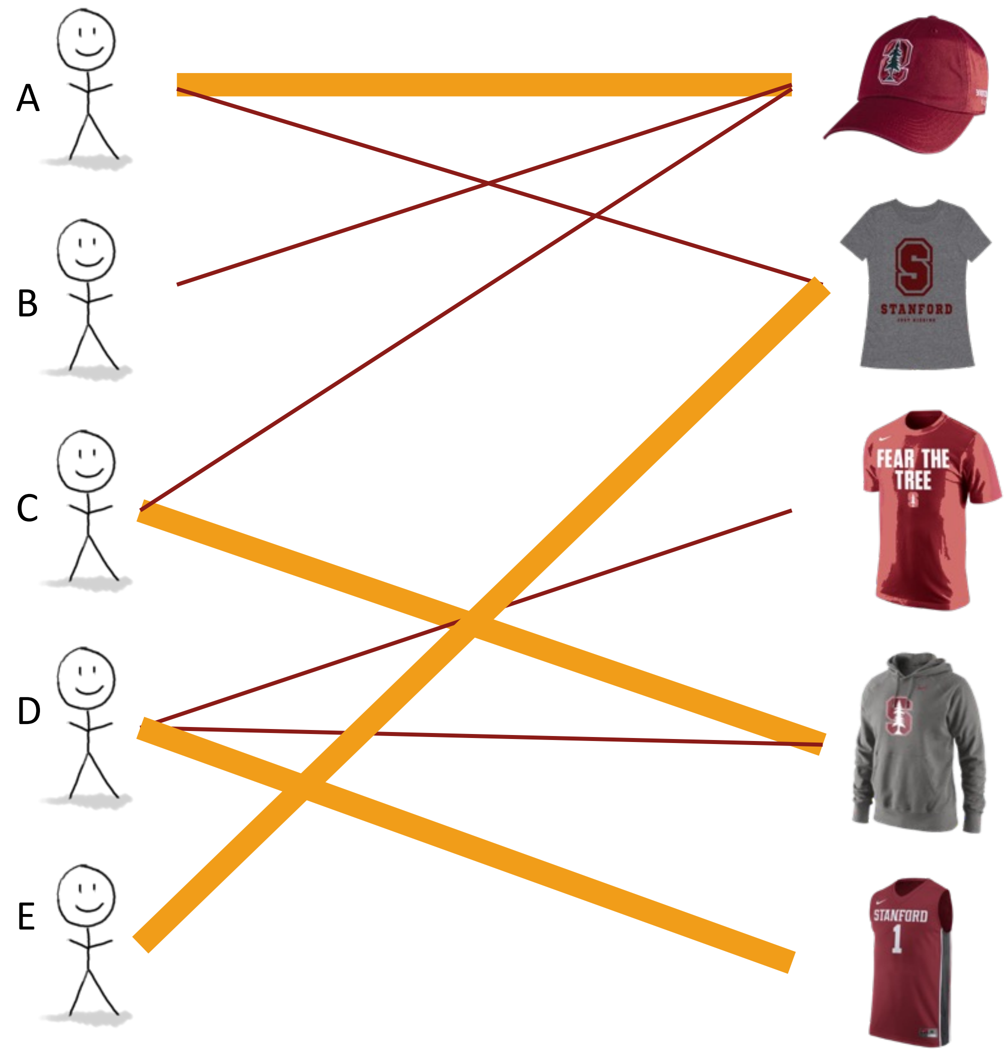
- Different students only want certain items of Stanford swag (depending on fit, style, etc.)
- **How can we make as many students as possible happy?**



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?**

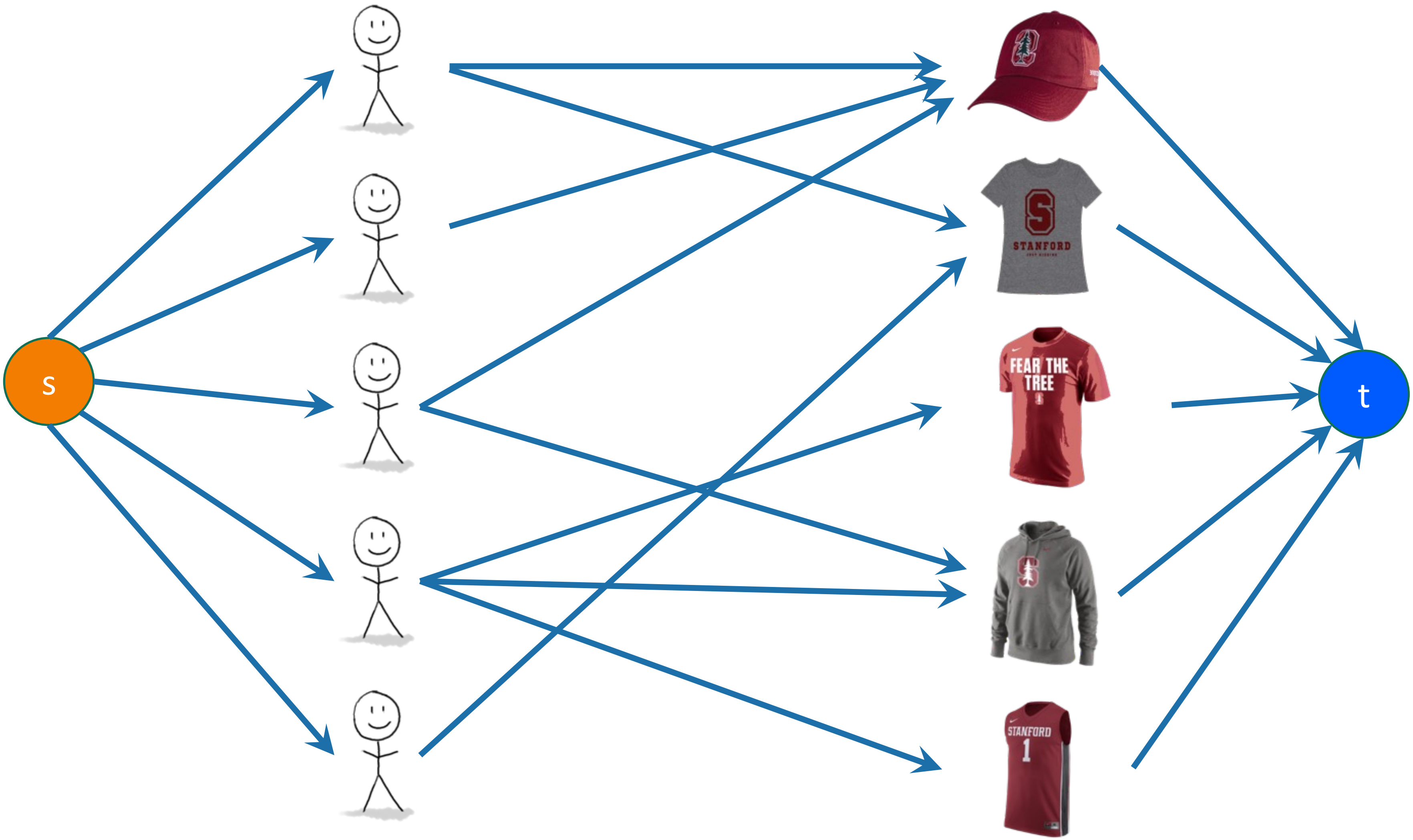


Stanford Students

Stanford Swag

Solution via max flow

All edges have capacity 1.

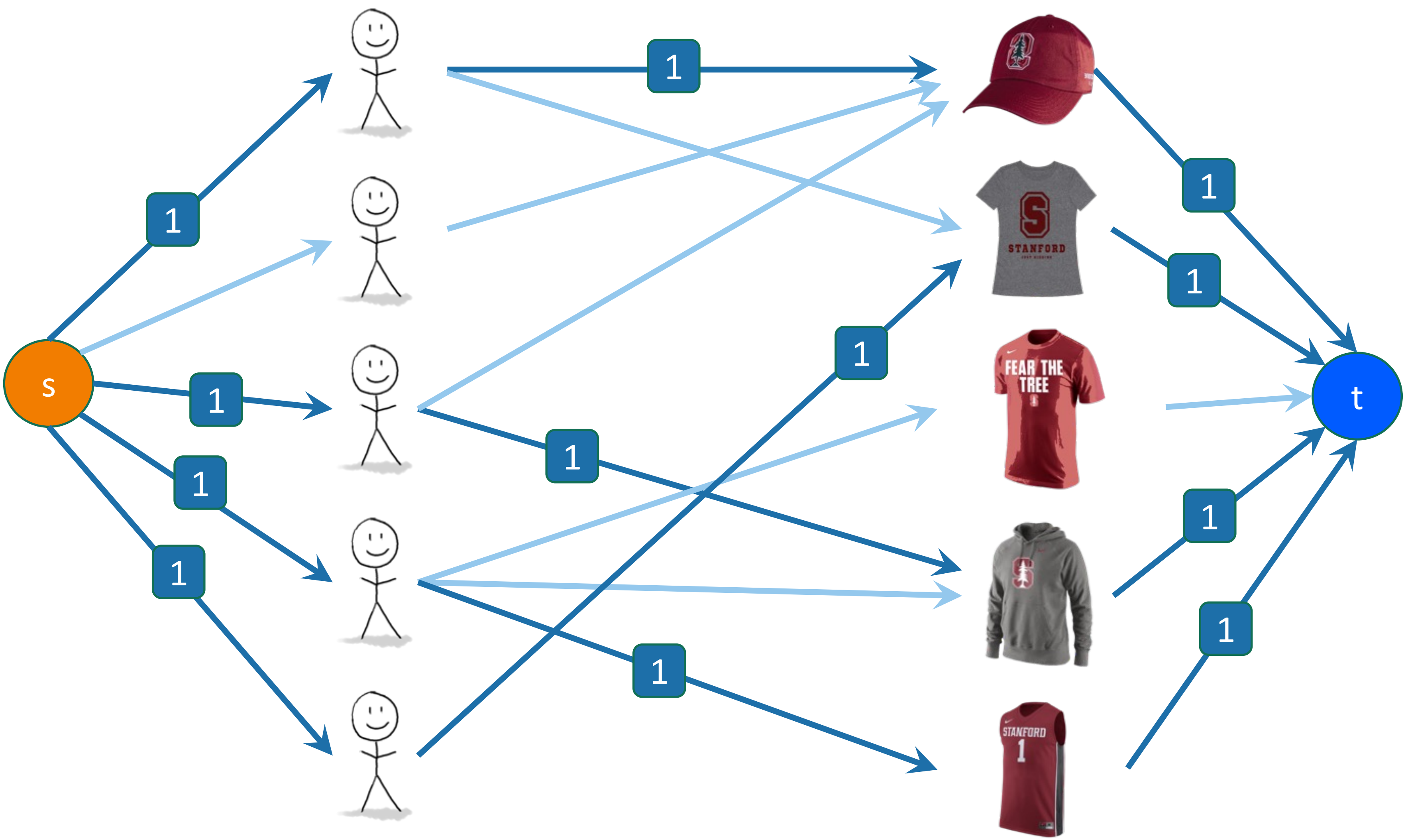


Stanford Students

Stanford Swag

Solution via max flow

All edges have capacity 1.



Stanford Students

Stanford Swag

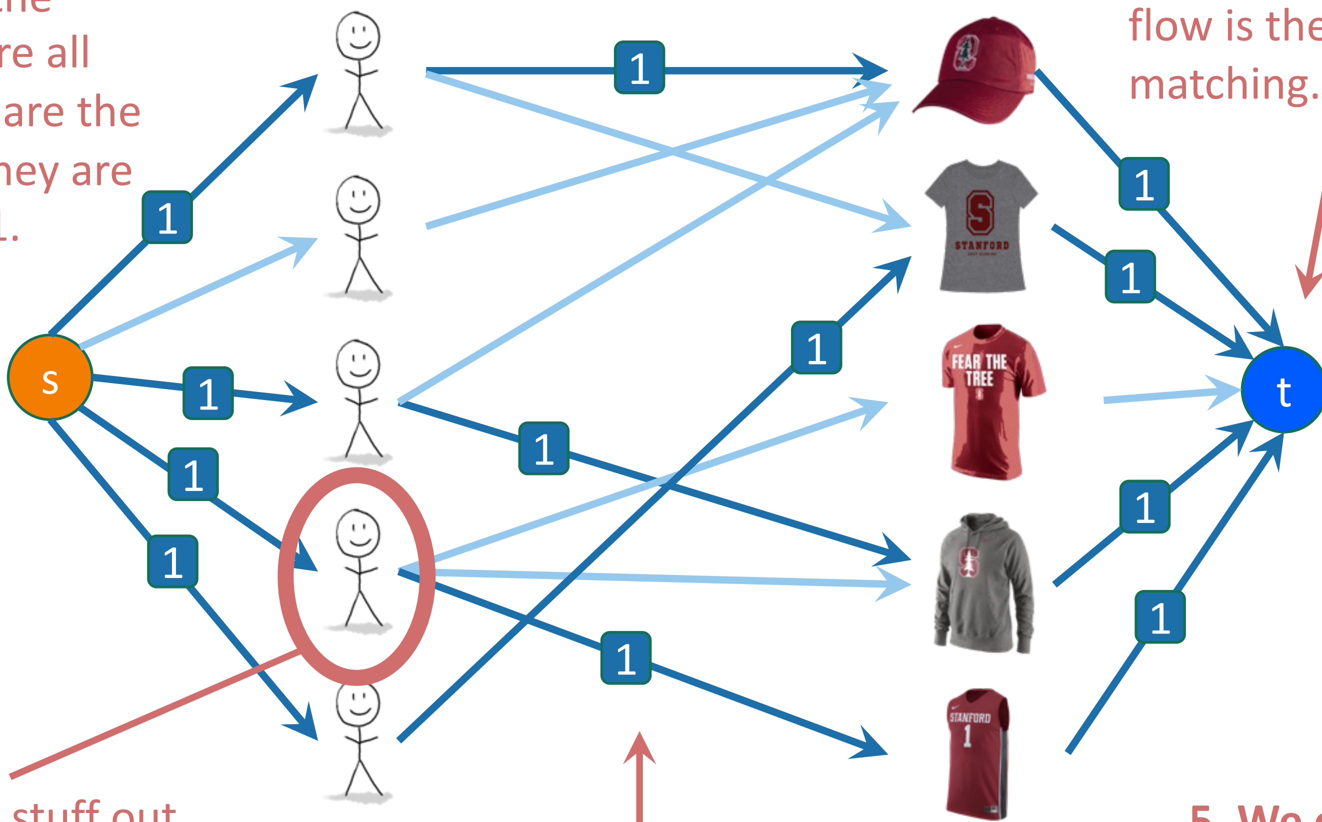
Solution via max flow

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

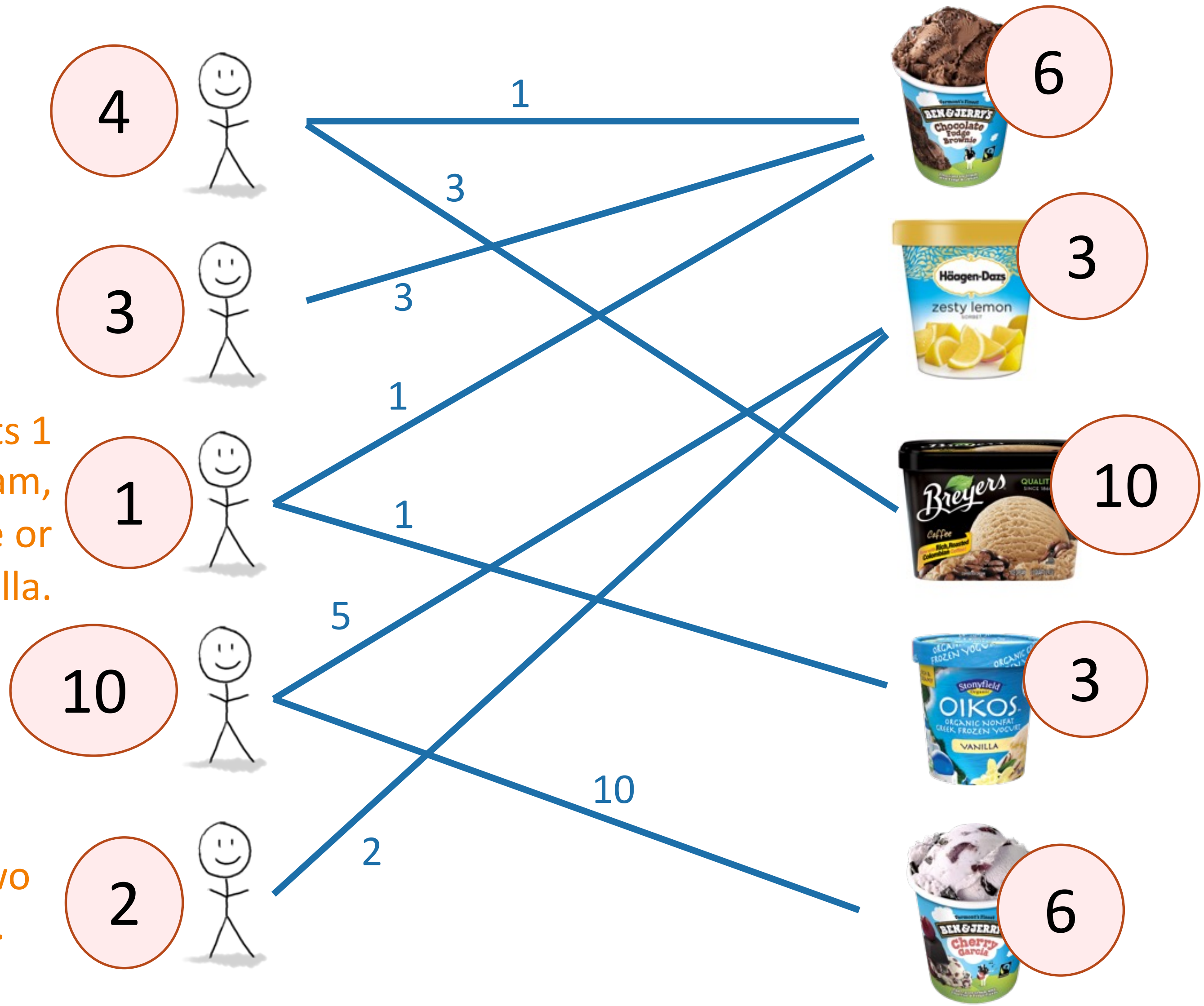
- One set X
  - Example: Stanford students
- Another set Y
  - Example: tubs of ice cream
- Each x in X can participate in c(x) matches.
  - Student x can only eat 4 scoops of ice cream.
- Each y in Y can only participate in c(y) matches.
  - Tub of ice cream y only has 10 scoops in it.
- Each pair (x,y)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

Example

How can we serve as much ice cream as possible?

This person wants 1 scoop of ice cream, either 1 chocolate or 1 vanilla.

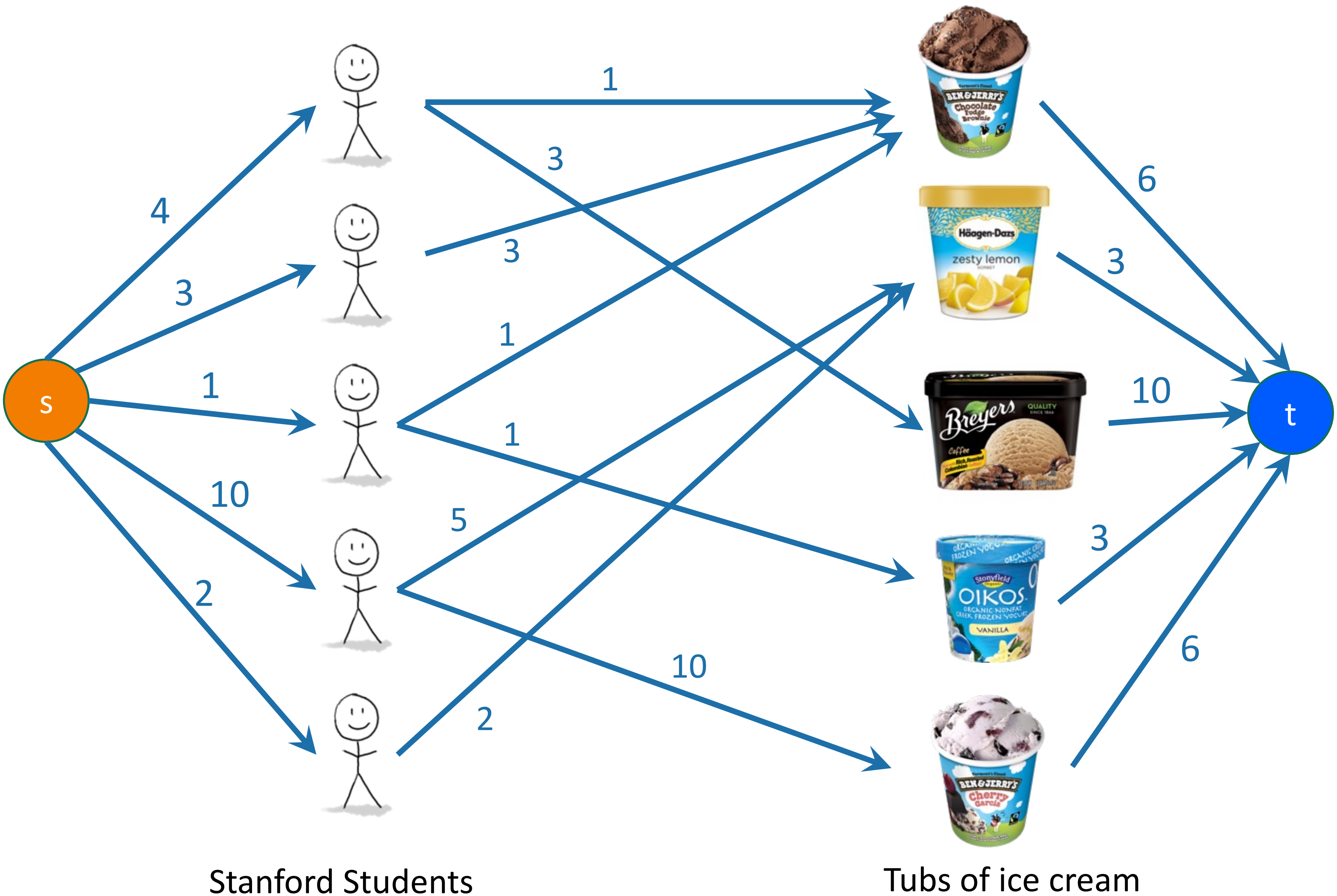
This person wants two scoops of the sorbet.



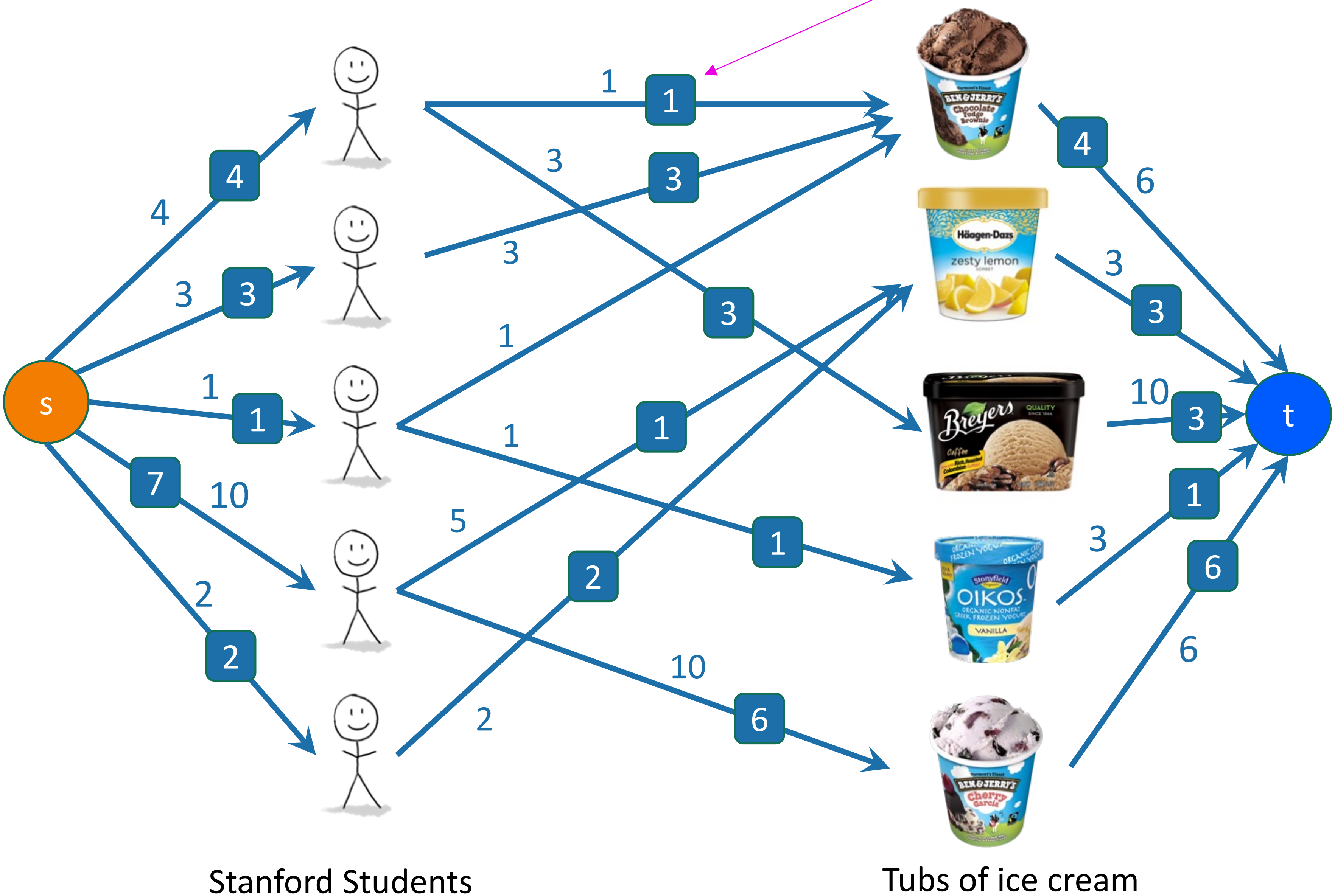
Stanford Students

Tubs of ice cream

Solution via max flow

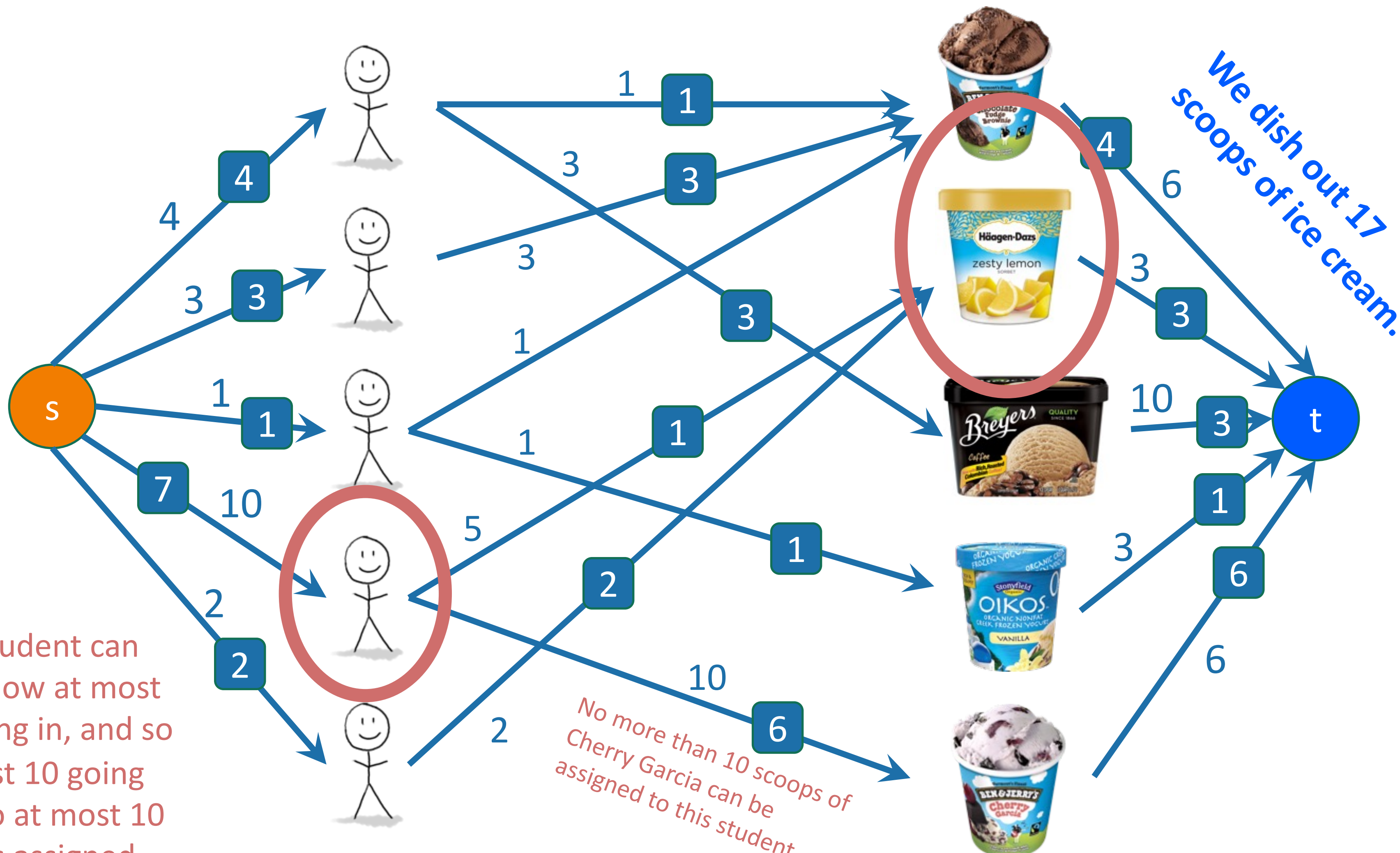


Solution via max flow



Solution via max flow

No more than 3 scoops of sorbet can be assigned.



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

Can we do better?

State-of-the-art max flow

Maximum Flow and Minimum-Cost Flow in Almost-Linear Time

(Preliminary Version)

Li Chen*
Georgia Tech
lichen@gatech.edu

Rasmus Kyng†
ETH Zurich
kyng@inf.ethz.ch

Yang P. Liu‡
Stanford University
yangpliu@stanford.edu

Richard Peng
University of Waterloo §
y5peng@uwaterloo.ca

Maximilian Probst Gutenberg†
ETH Zurich
maxprobst@ethz.ch

Sushant Sachdeva¶
University of Toronto
sachdeva@cs.toronto.edu

March 2, 2022

Abstract

We give an algorithm that computes exact maximum flows and minimum-cost flows on directed graphs with m edges and polynomially bounded integral demands, costs, and capacities in $m^{1+o(1)}$ time. Our algorithm builds the flow through a sequence of $m^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $m^{o(1)}$ time using a dynamic data structure.

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!
  - E.g., assignment problems.

Next time

- Stable Matchings!
  - Deferred Acceptance (Gale-Shapley) Algorithm

Source: https://www.nrmp.org/about/