

8/3 Lecture Agenda

- Announcements
- Part 6-3: Max Flow
- 10 minute break!
- Part 6-4: Bipartite Matching

Announcements

- **Pre-HW6** will be out tonight, due next Wednesday.
- **HW6** will be out Friday, due next **Thursday**.
 - We are going to make it as automated as we can. (Or, for some problems, like "find a counterexample", you will know when you have it right)
- Templates / autograders for **HW5** Problem 6 coming very soon (tests & solutions are written)

Announcements

- This lecture is the last one in scope for the final!
 - Monday's has some interesting material but is optional.
- **Unit 6** will be covered in a more BFS than DFS way on the final, since there is not as much time to digest it via HW and repeated practice.
- We know it is important to have graded work back before the final. I will – at long last – be out of my "always busy with new content creation" loop soon (the final is mostly written)

WORLD 6-8

Flow Problems

Divide and Conquer

Sorting & Randomization

Data Structures

Graph Search

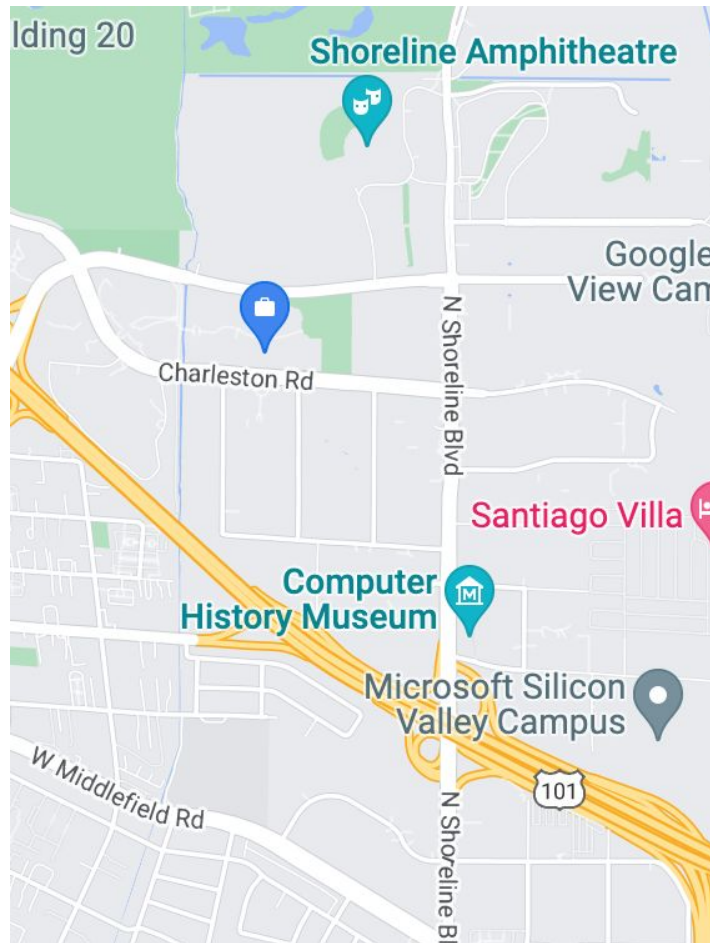
Dynamic Programming

Greed & Flow

Special Topics

A new thing to do with directed graphs

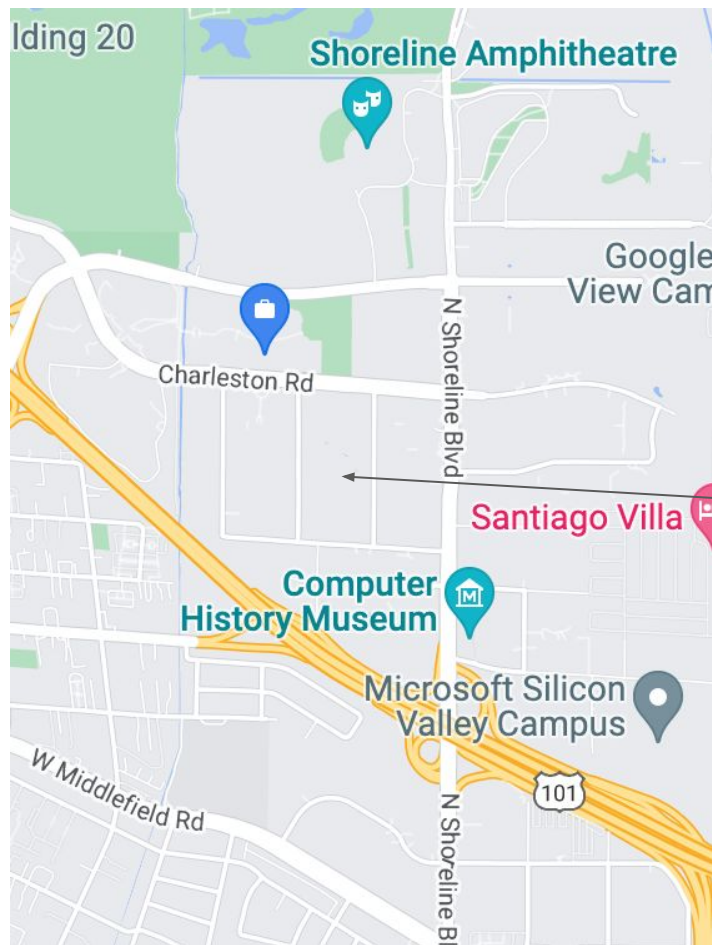
A concert at Shoreline just got out, and everyone is trying to get on 101. Shoreline Blvd. itself is *crawling*.



A new thing to do with directed graphs

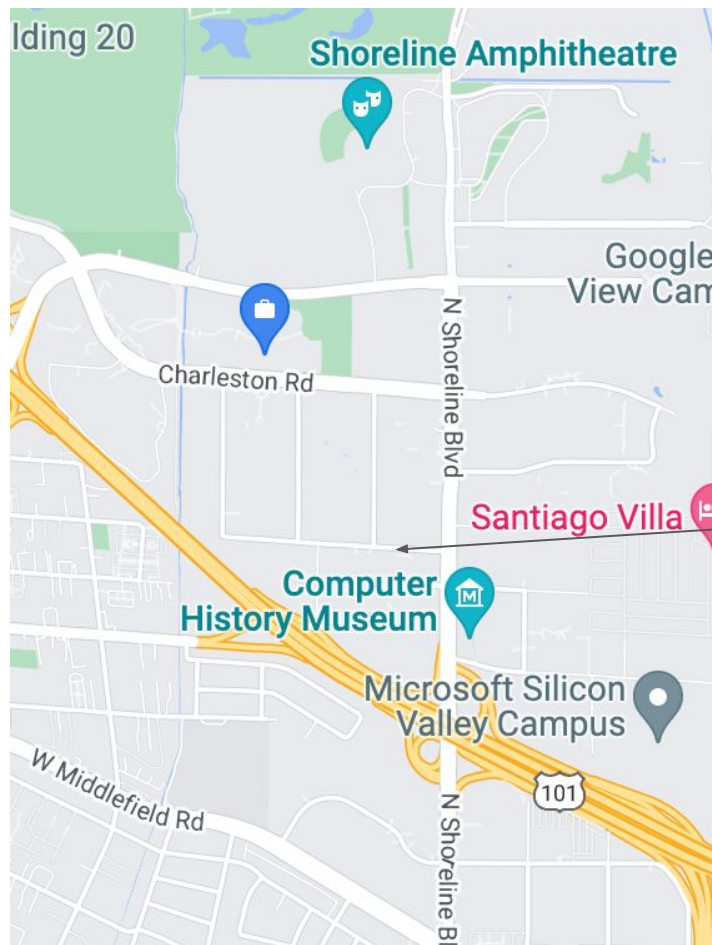
A concert at Shoreline just got out, and everyone is trying to get on 101. Shoreline Blvd. itself is *crawling*.

There are all these parallel streets in this part of the Google campus. But they are a sucker move!



A new thing to do with directed graphs

A concert at Shoreline just got out, and everyone is trying to get on 101. Shoreline Blvd. itself is *crawling*.



This one street (and the movie theater) just end up being the choke points. Any savings from the "shortcut" is illusory.

Aside: importing slides is hard

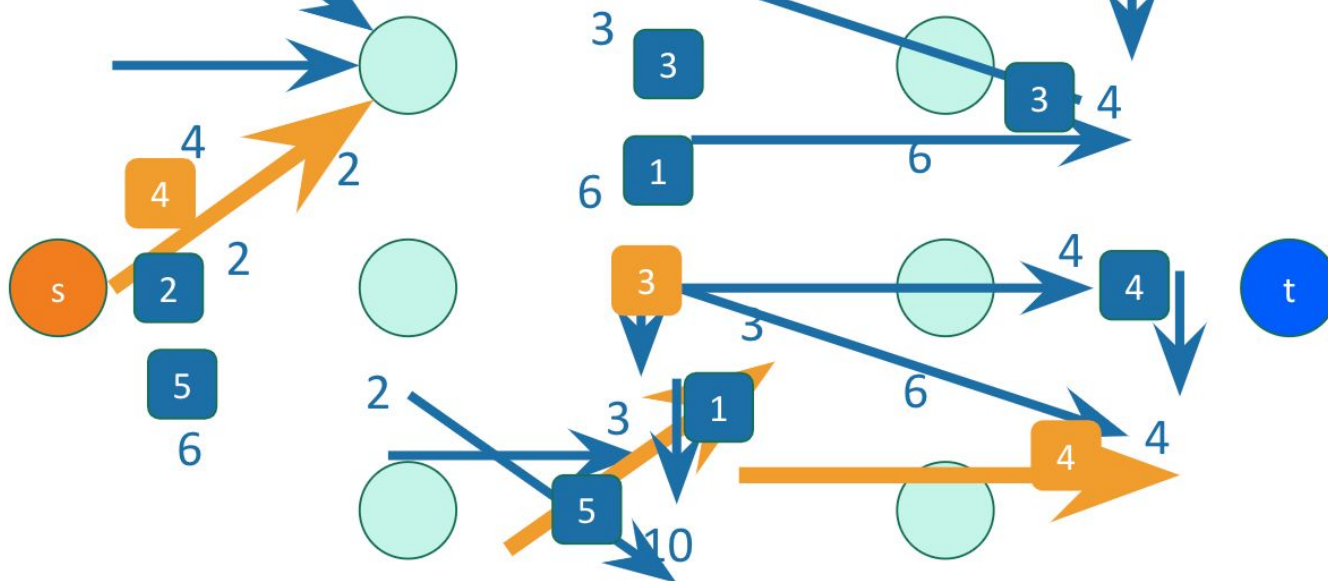
Theorem

Max-flow min-cut theorem

I ended up screenshotting a PDF. Apologies for artifacts

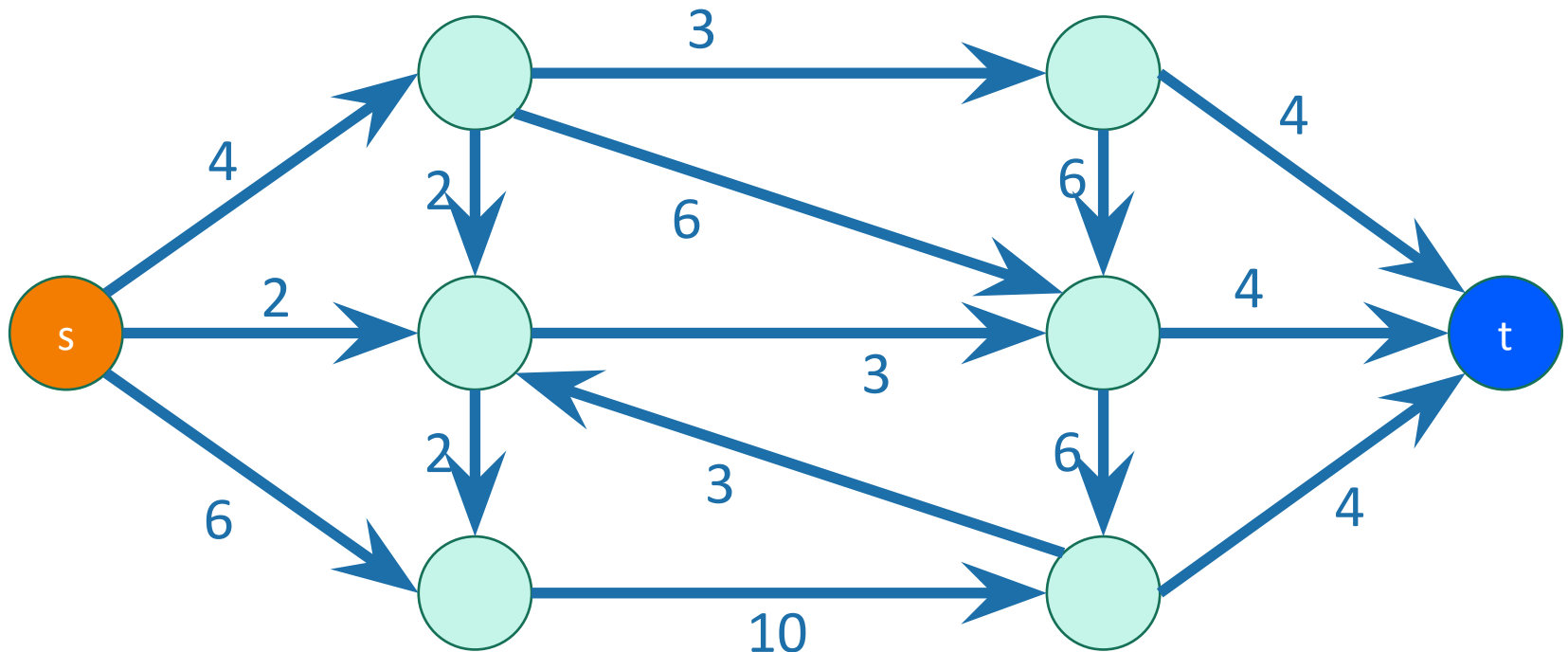
The value of a max flow from s to t is equal to the cost of a min s - t cut.

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



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*



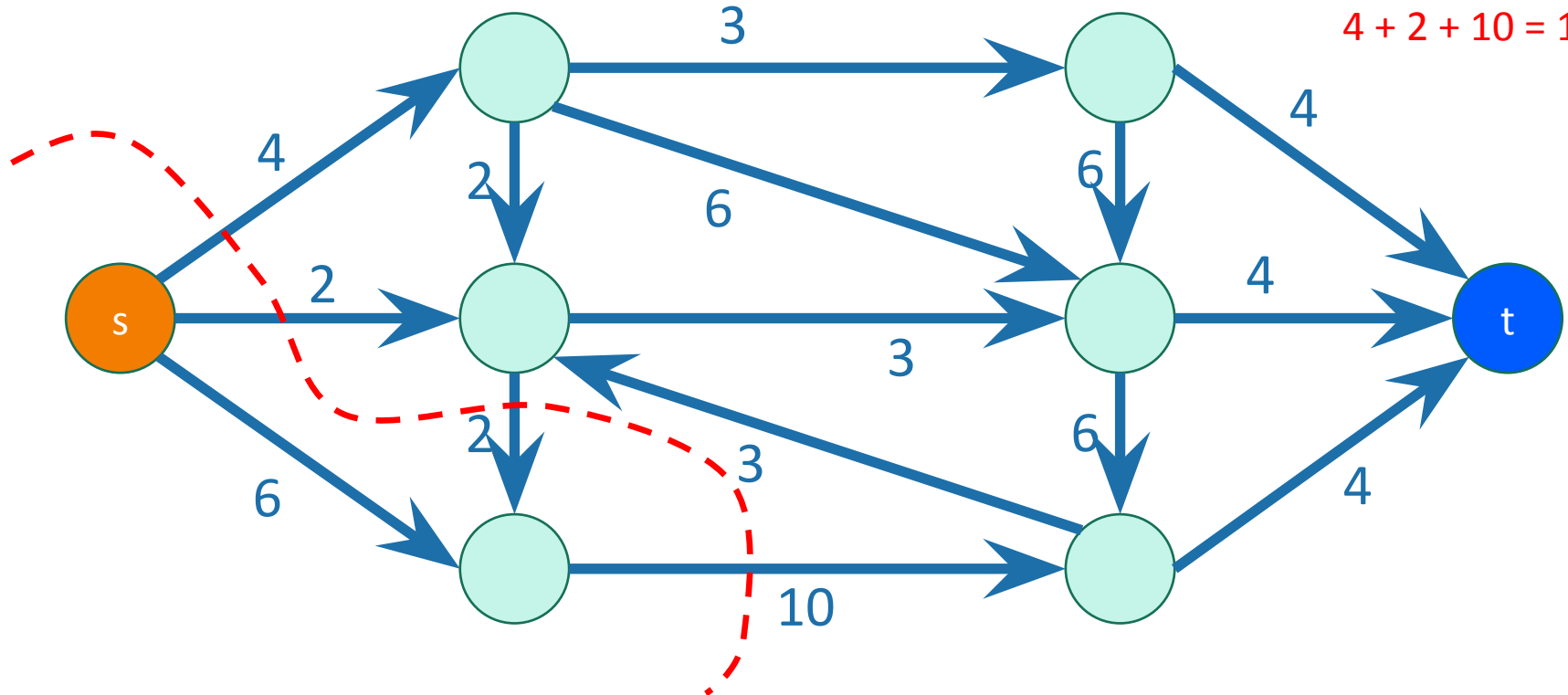
*at least for this class

An s-t cut

is a cut which separates s from t

- An 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.

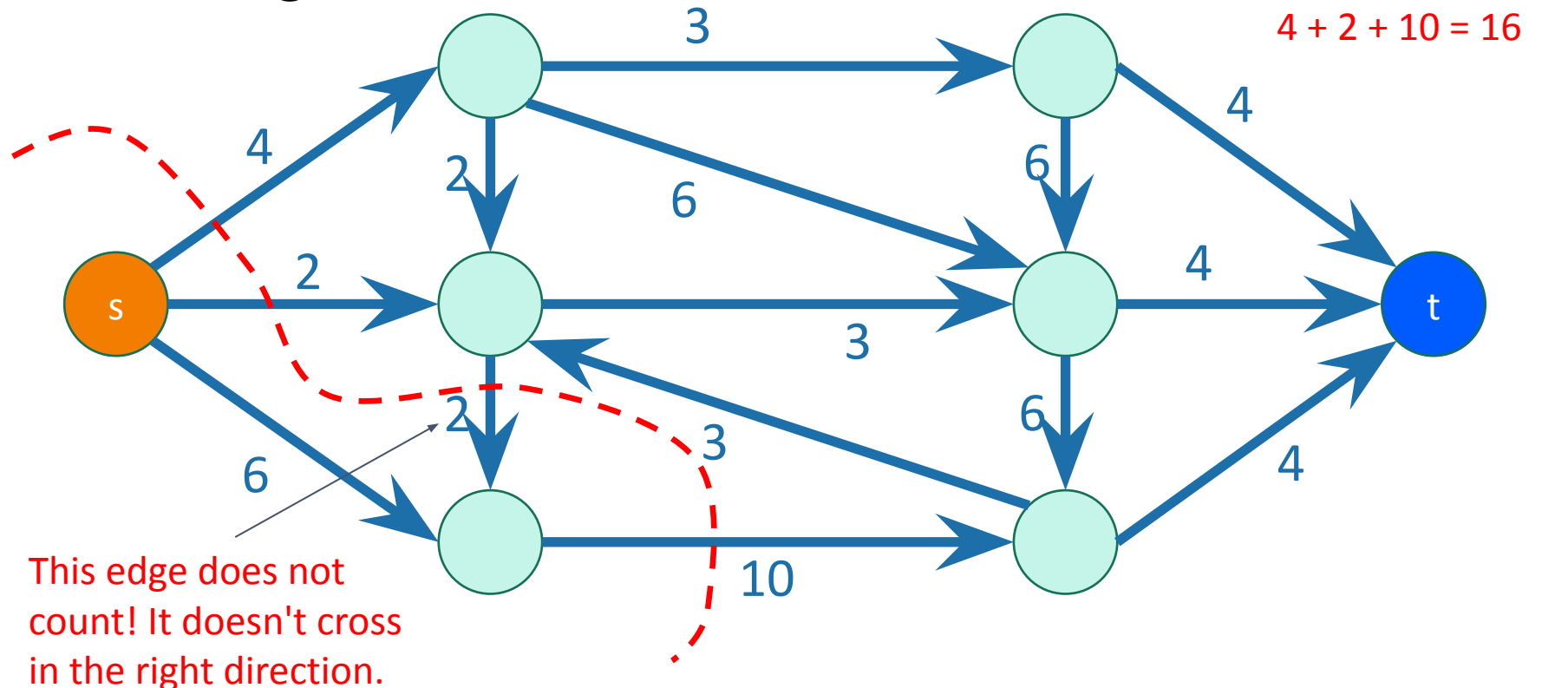
This cut has cost
 $4 + 2 + 10 = 16$



An s-t cut

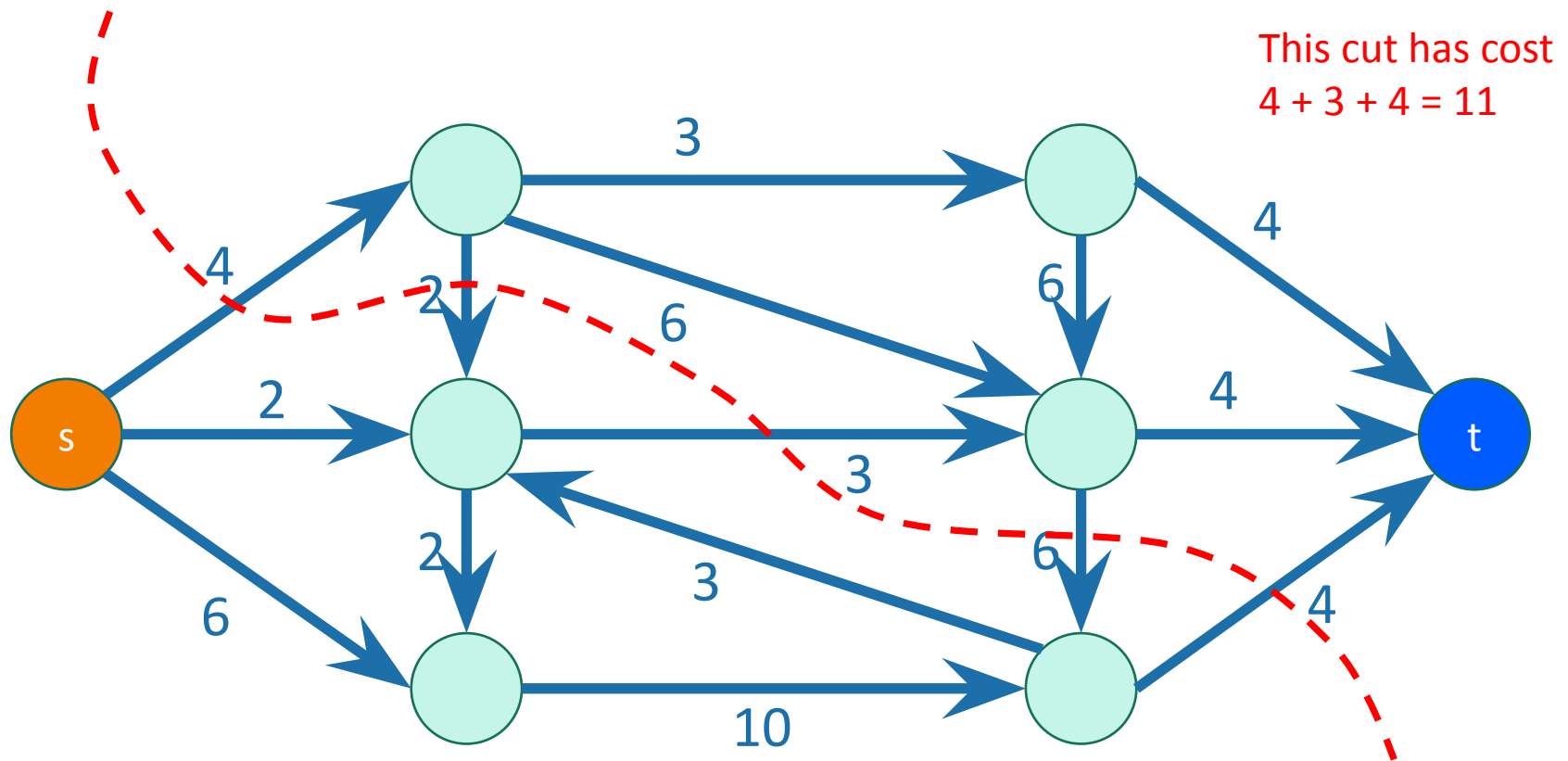
is a cut which separates s from t

- 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.

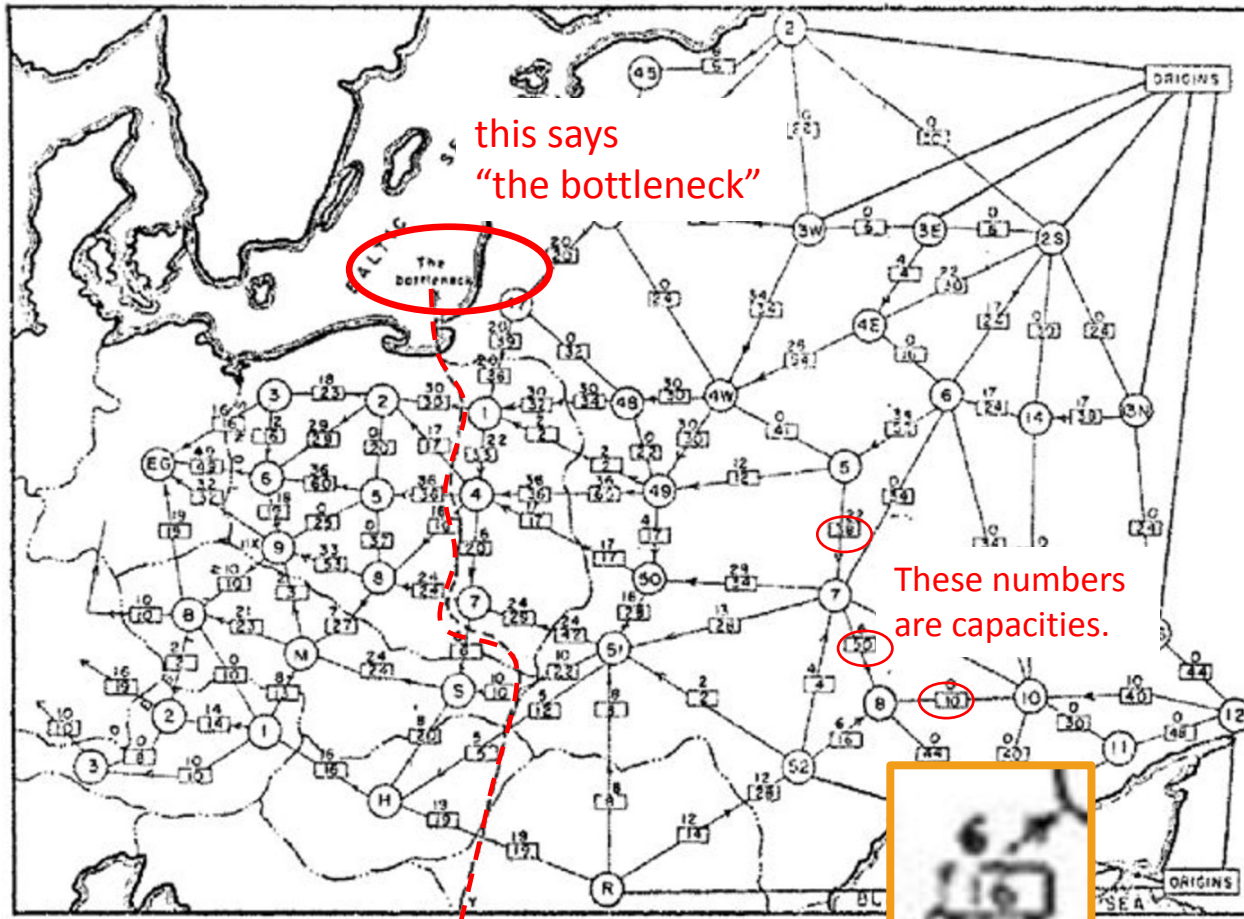


A minimum **s-t** cut

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



Example where this comes up

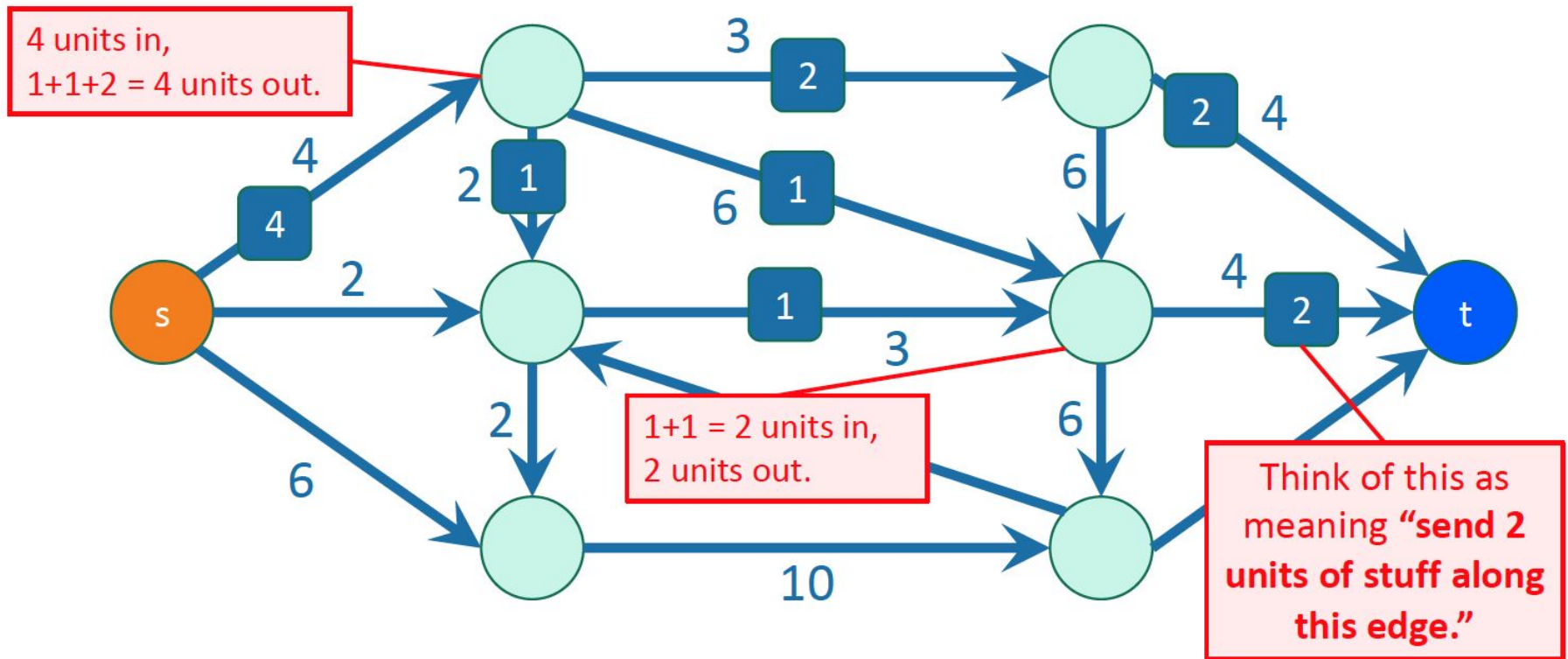


Schrivver 2002

- 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

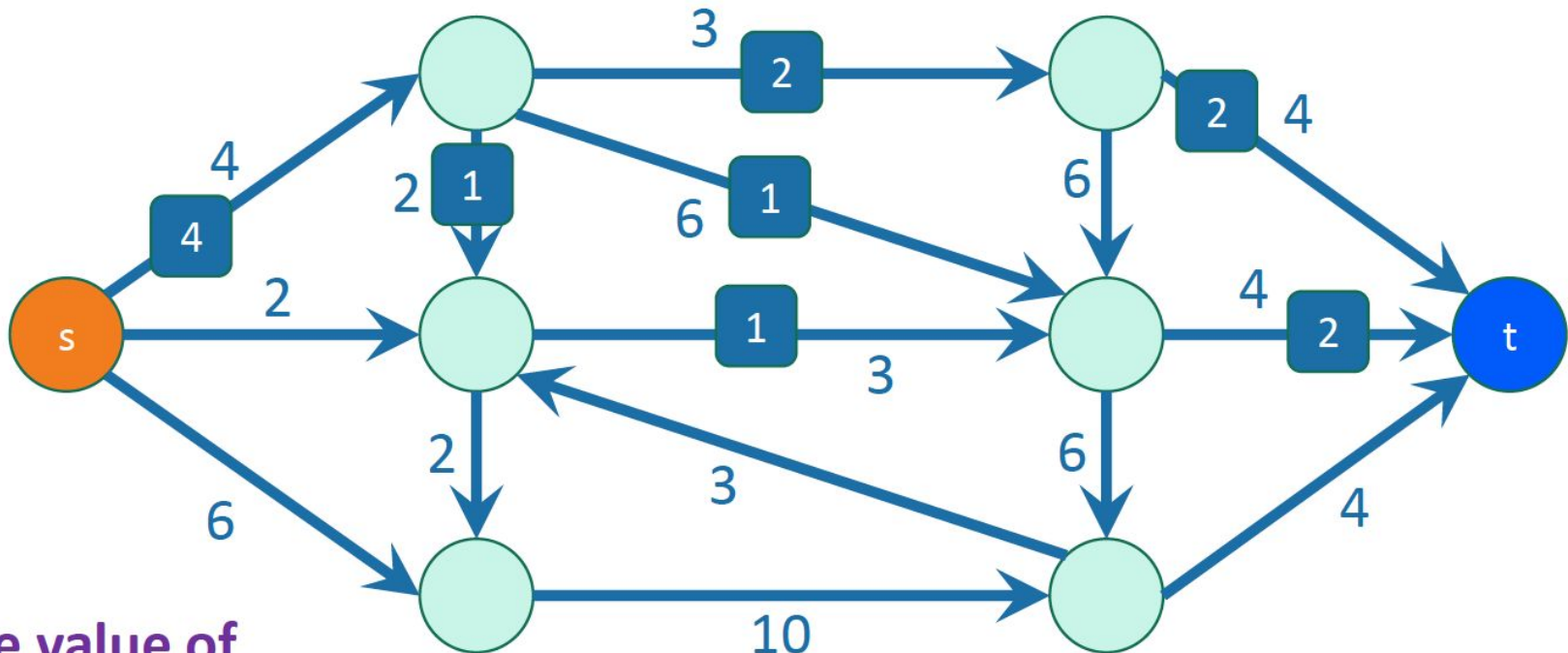
- In addition to a capacity, each edge has a **flow**
 - (unmarked edges in the picture have flow 0)
- The flow on an edge must be less than its capacity.
- At each vertex, the incoming flows must equal the outgoing flows.



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!

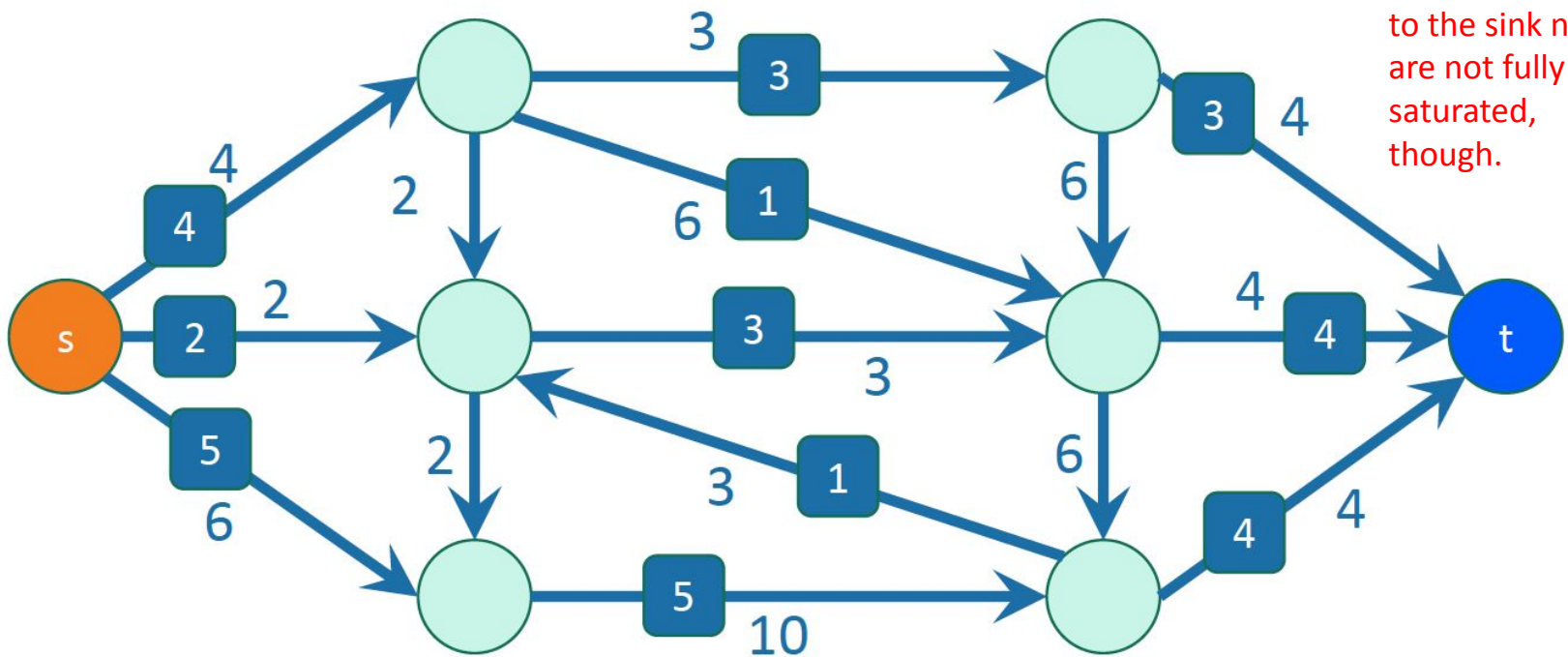
Because of conservation of flows at vertices,
stuff you put in
=
stuff you take out.



The value of this flow is 4.

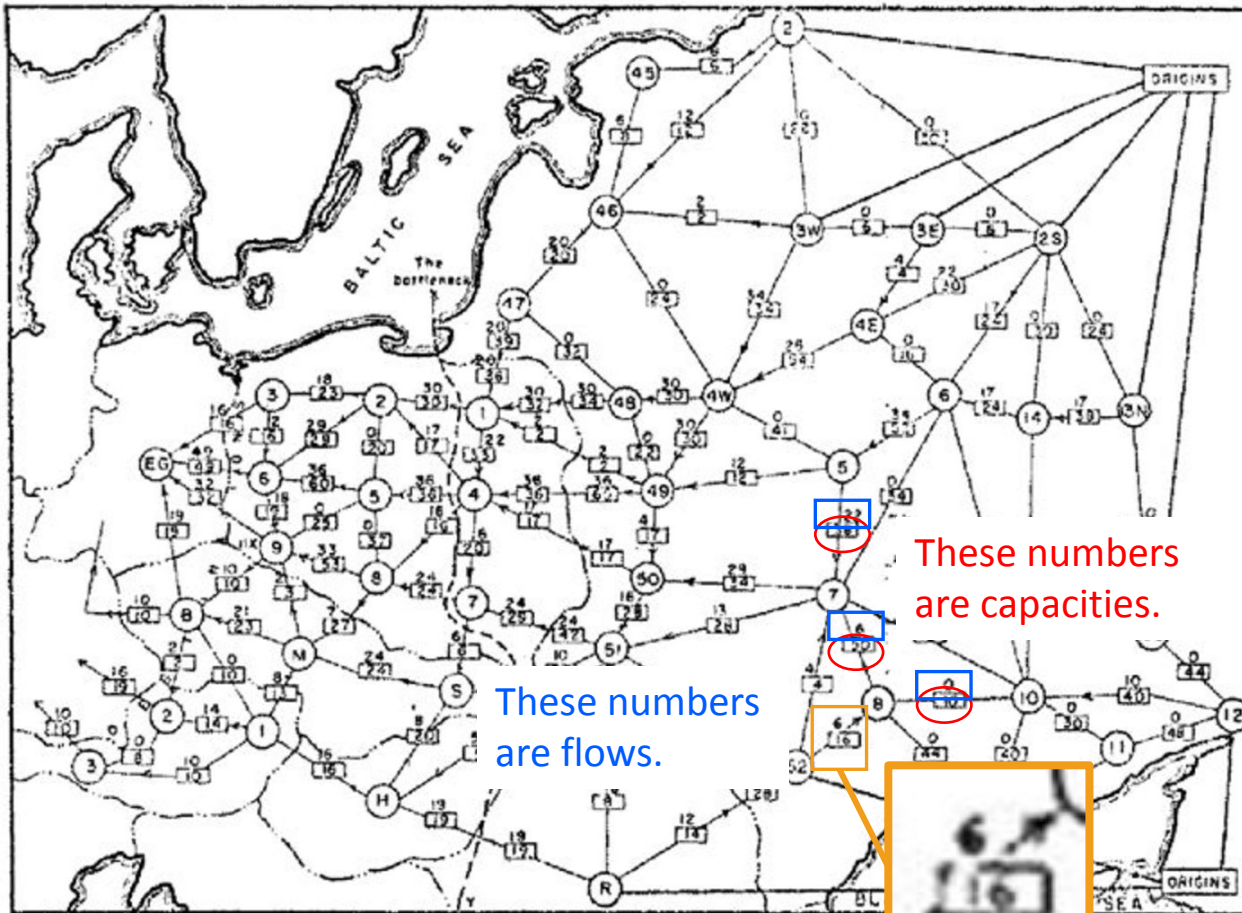
A maximum flow is a flow of maximum value.

- This one is maximum; it has value 11.



Notice the edges to the sink node are not fully saturated, though.

Example where this comes up

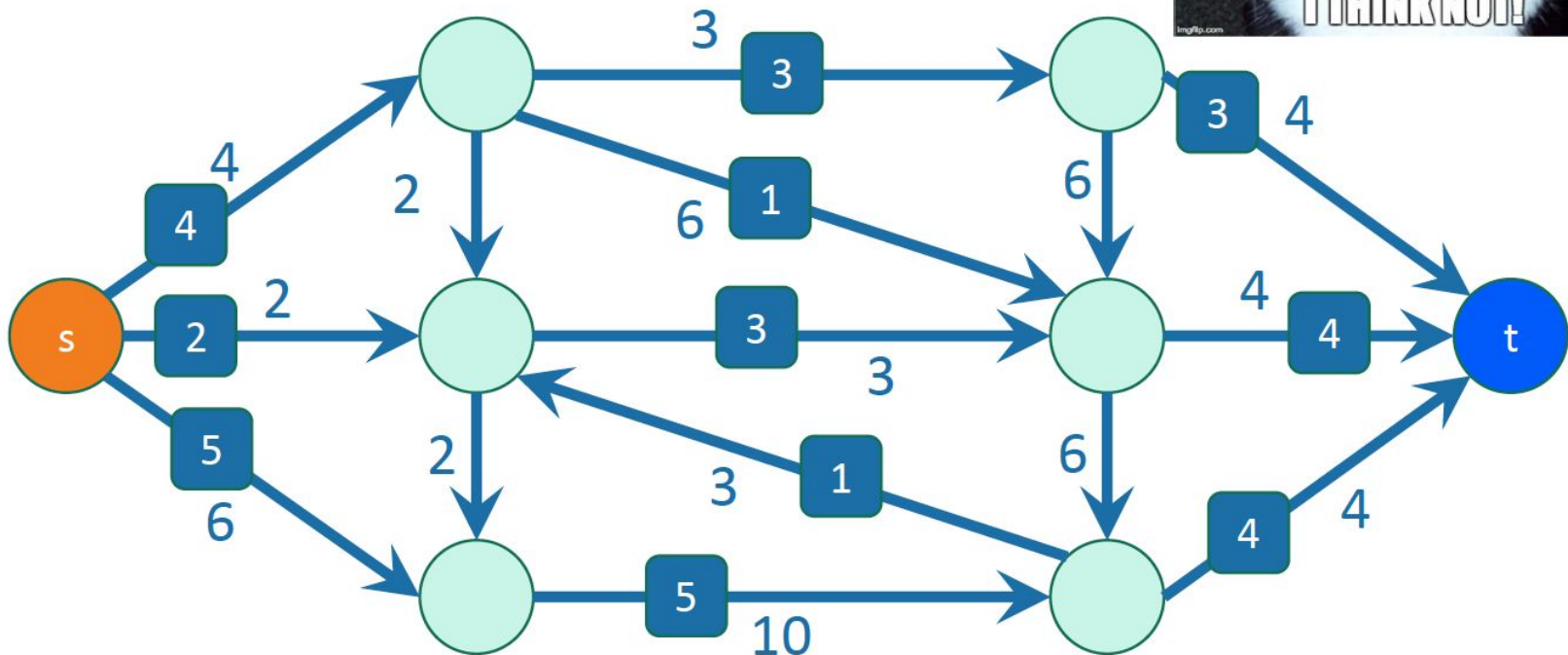
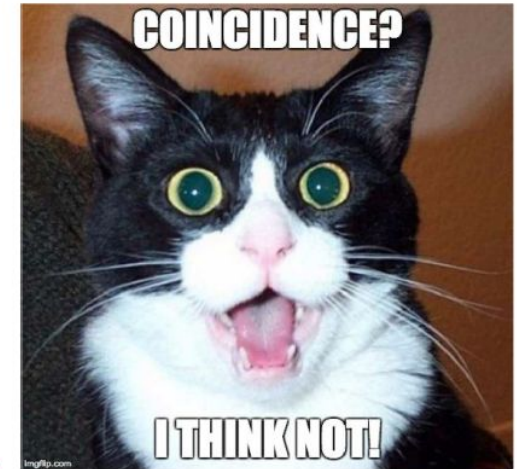


- 1955 map of rail networks from the Soviet Union to Eastern Europe.
 - Declassified in 1999.
 - 44 edges, 105 vertices
- The Soviet Union wants to route supplies from suppliers in Russia to Eastern Europe as efficiently as possible.

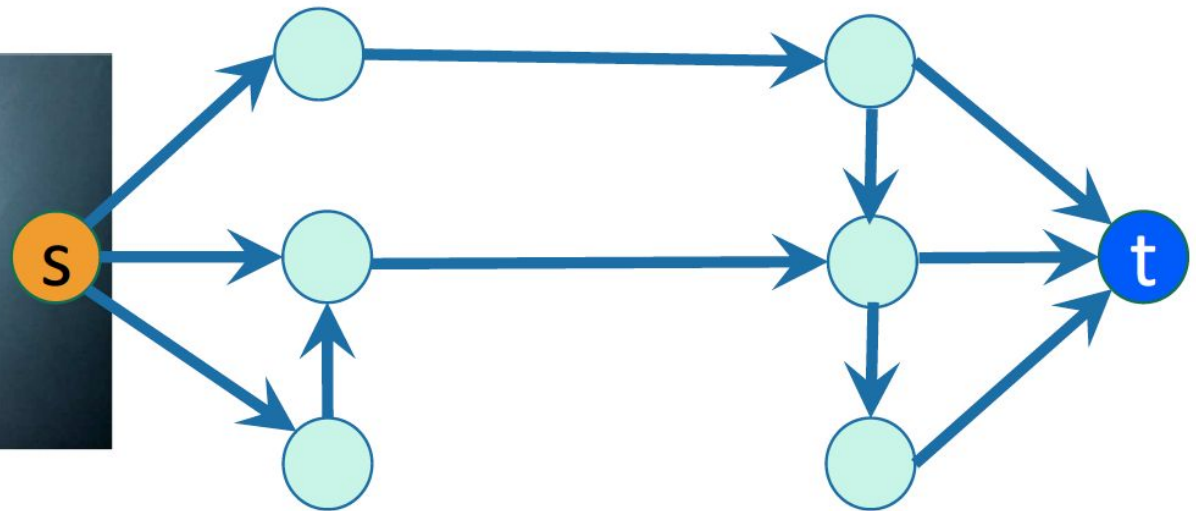
A maximum flow is a flow of maximum value.

- This one is maximum; it has value 11.

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

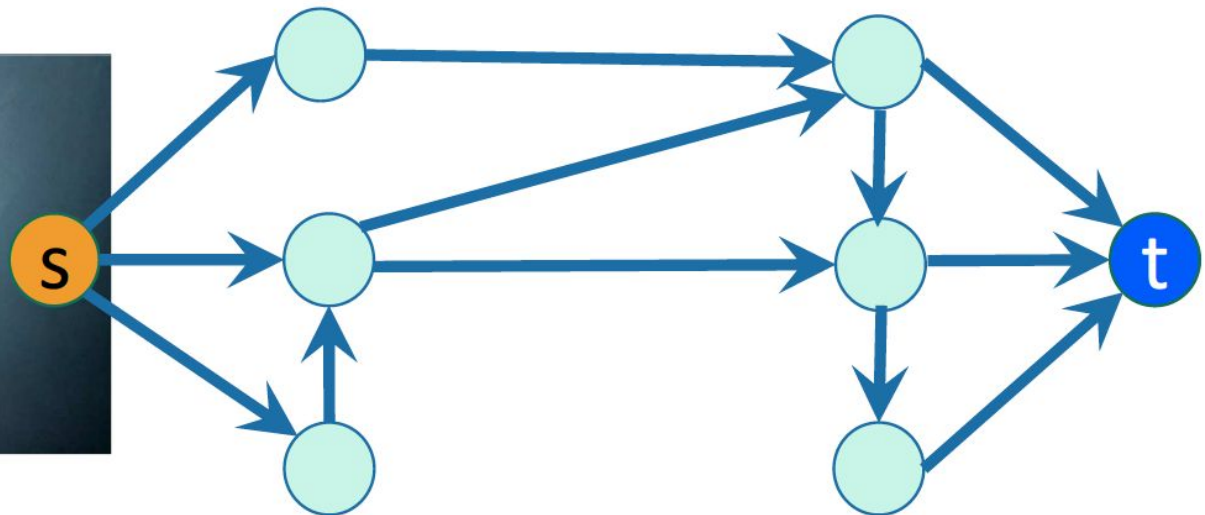


- Each edge is a (directed) rickety bridge.
- How many bridges need to fall down to disconnect s from t ? *For this graph, 2*
- If only one person can be on a bridge at a time, and you want to keep traffic moving (aka, no waiting at vertices allowed), how many people can get to t at a time? *Also 2!*



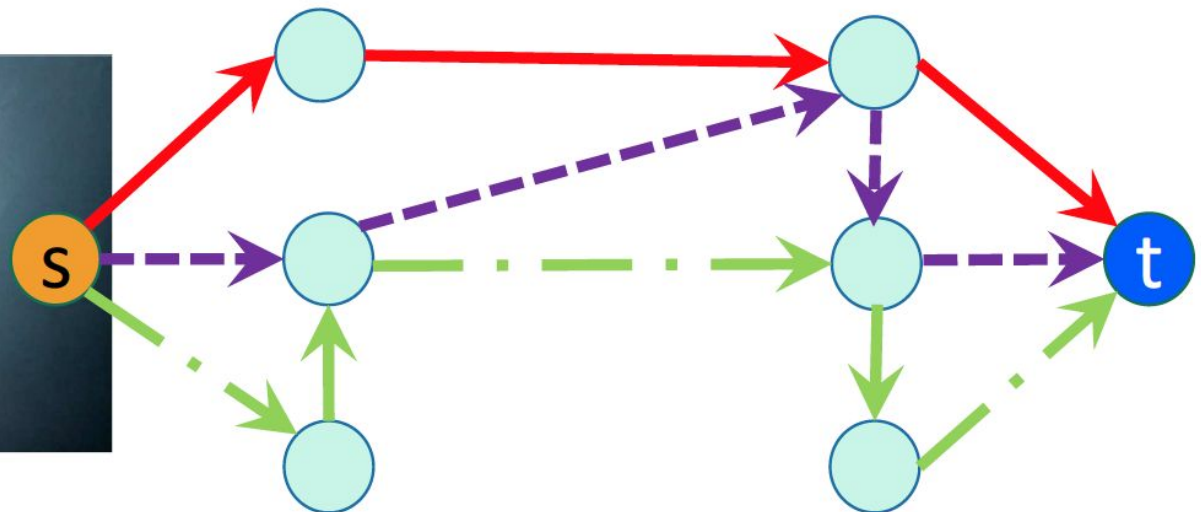
How about now?

- 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!*

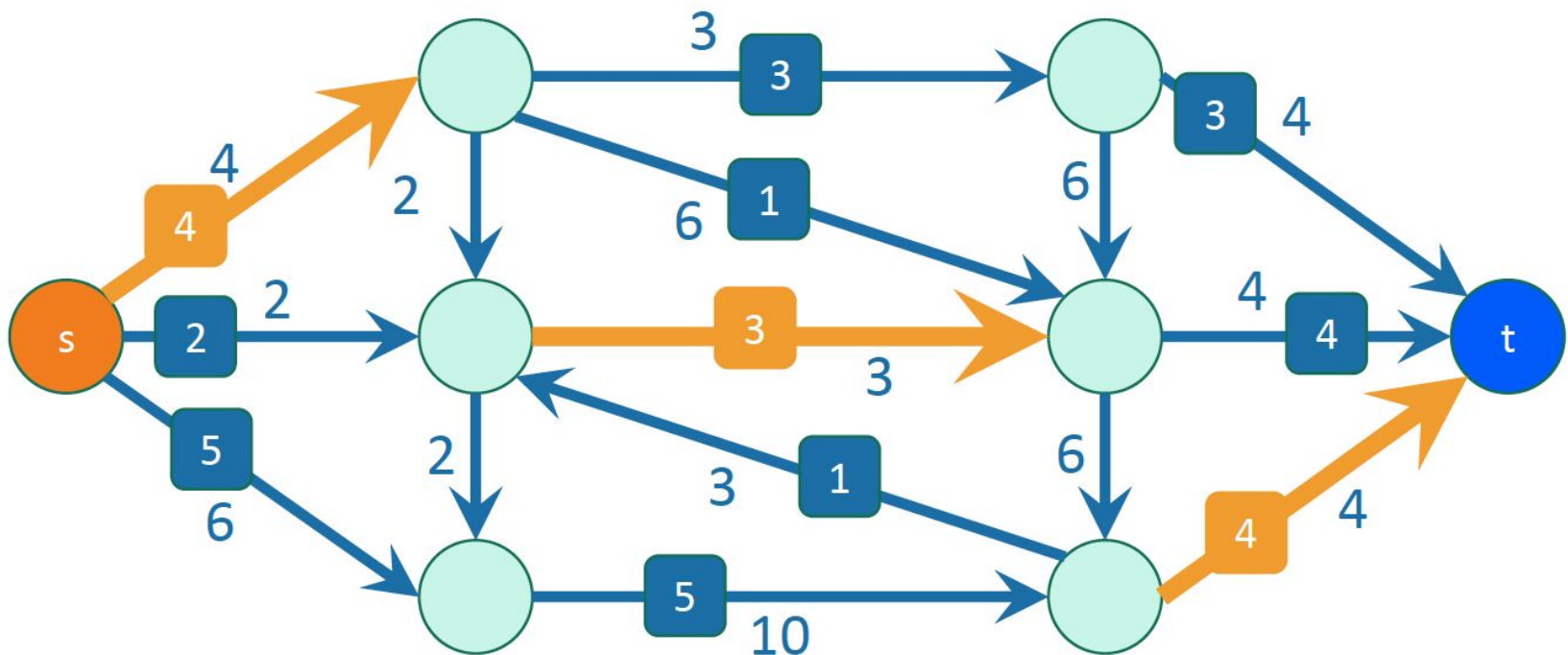


Theorem

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

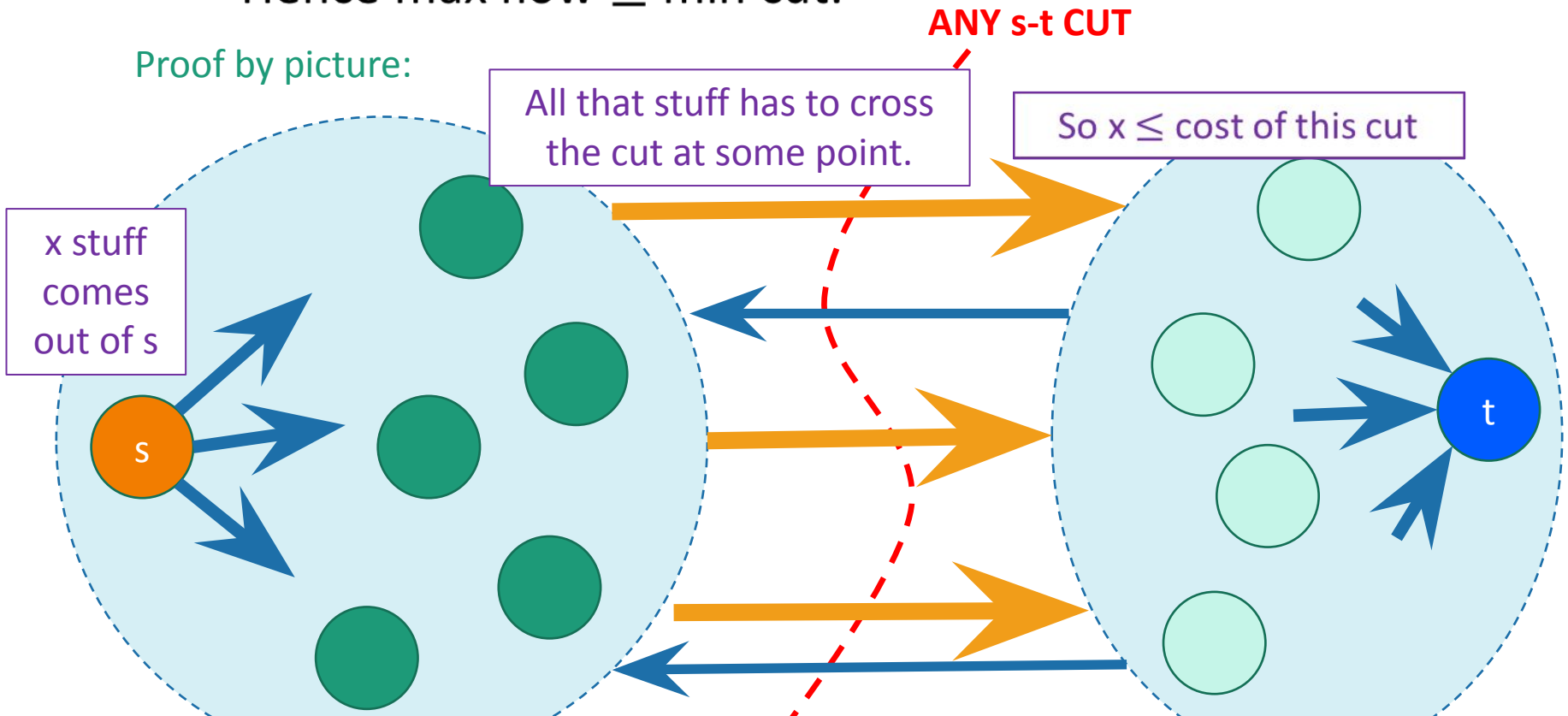
- Lemma 1: max flow \leq min cut.
 - Proof-by-picture
- What we actually want: max flow = min cut.
 - Proof-by-algorithm...the Ford-Fulkerson algorithm!
 - (Also using Lemma 1)

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 $\text{max flow} \leq \text{min cut}$.

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

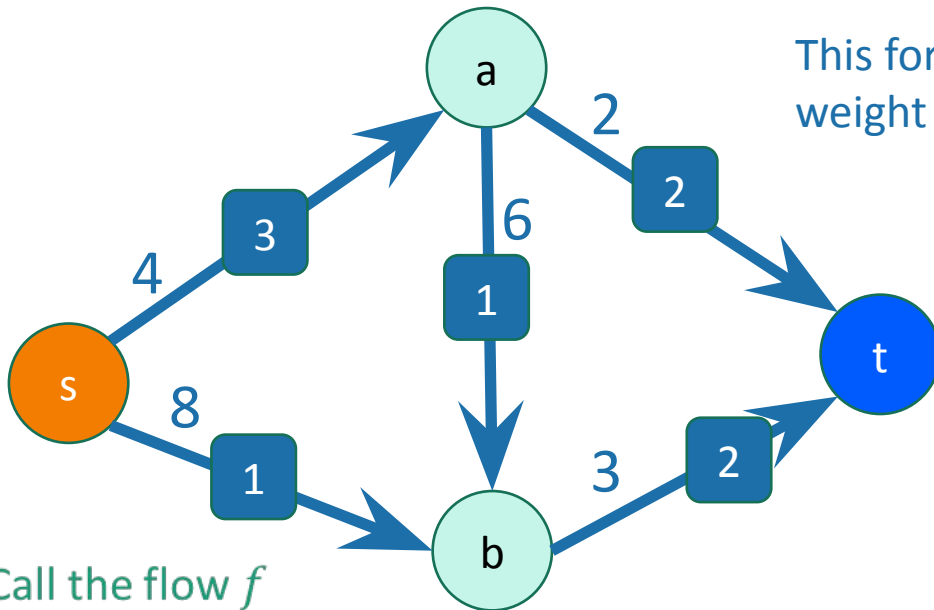
- Outline of algorithm:
 - Start with zero flow
 - We will maintain a “**residual graph**” G_f
 - A path from s to t in G_f will give us a way to improve our flow.
 - We will continue until there are no s - t paths left.

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



Tool: Residual networks

Say we have a flow

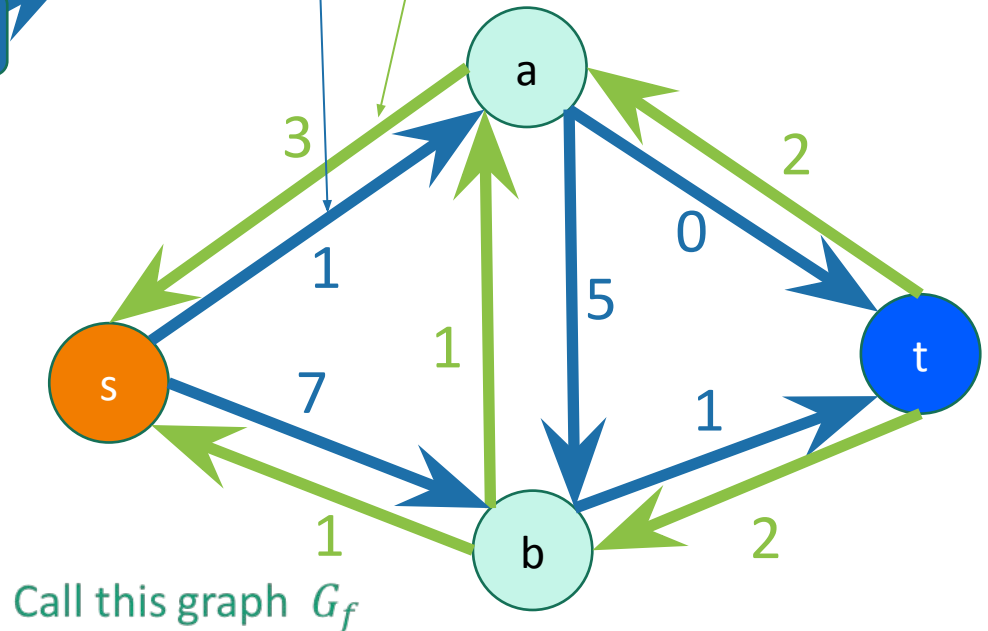


This forward edge has weight "capacity - flow".

This backward edge has weight "flow".

Call the flow f
Call the graph G

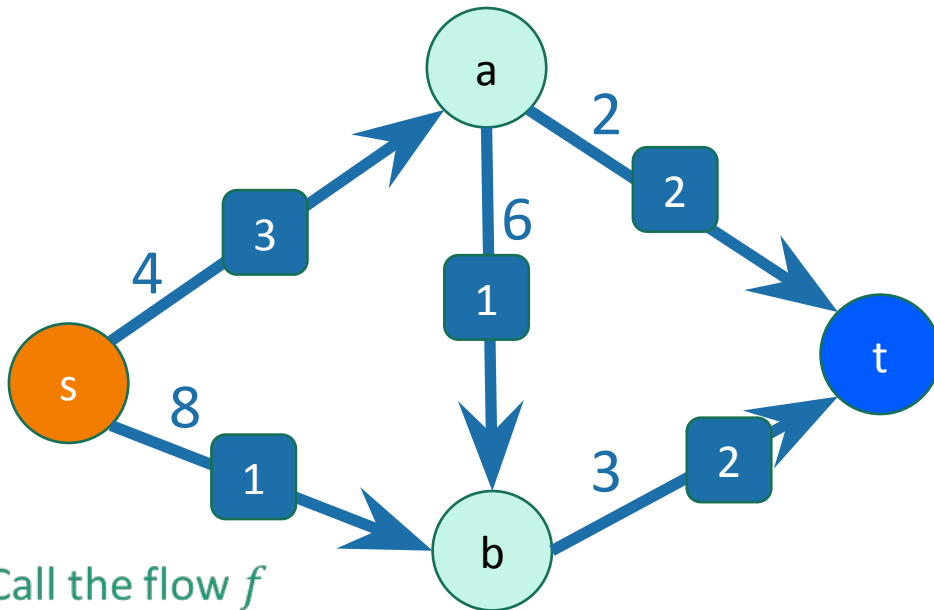
Create a new **residual**
network from this flow:



Call this graph G_f

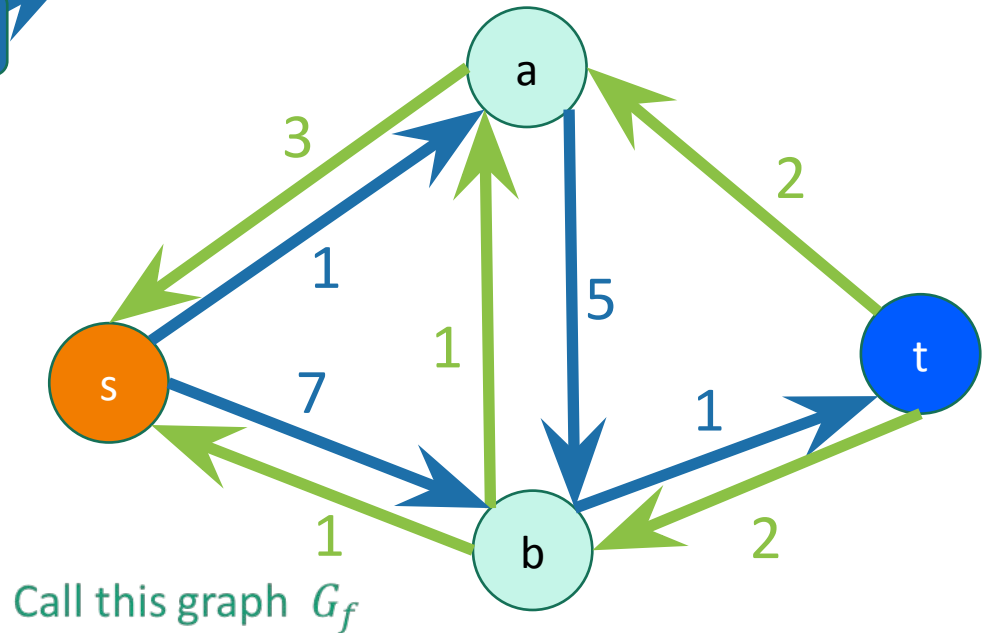
Tool: Residual networks

Say we have a flow



Call the flow f
Call the graph G

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



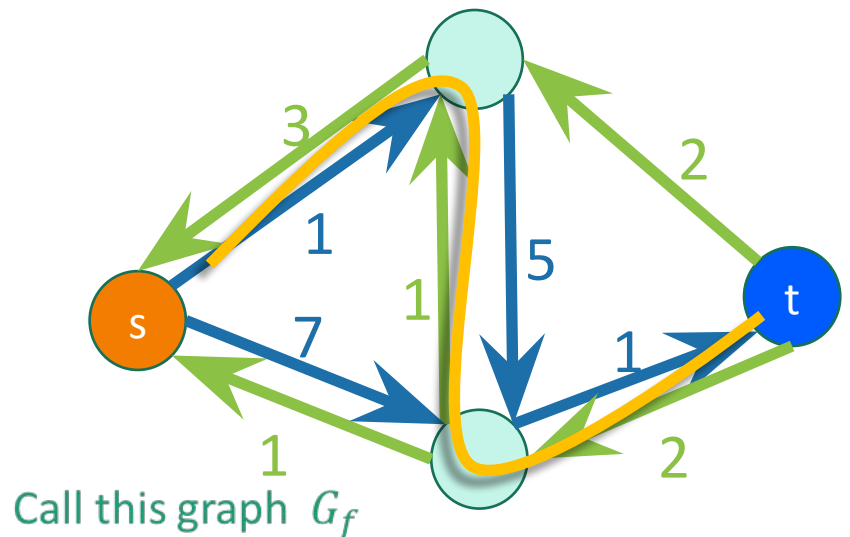
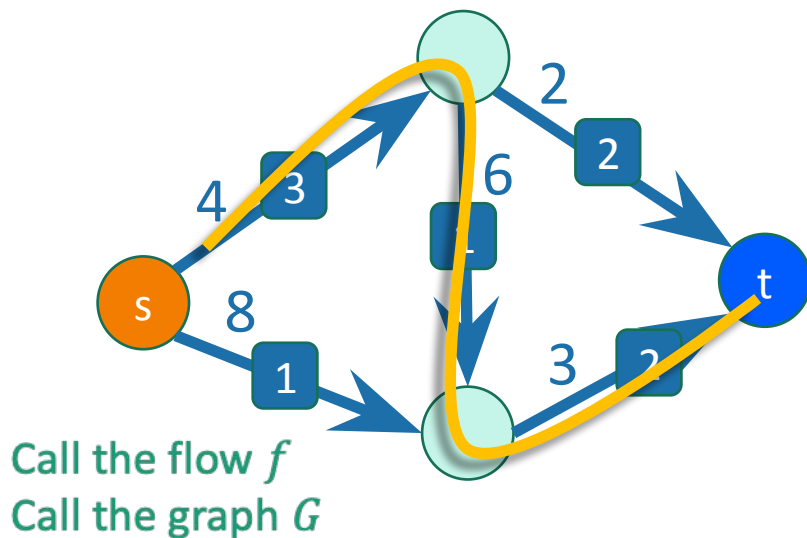
Call this graph G_f

Forward edges are the
amount that's left.

Backwards edges are the
amount that's been used.

Residual networks tell us how to improve the flow.

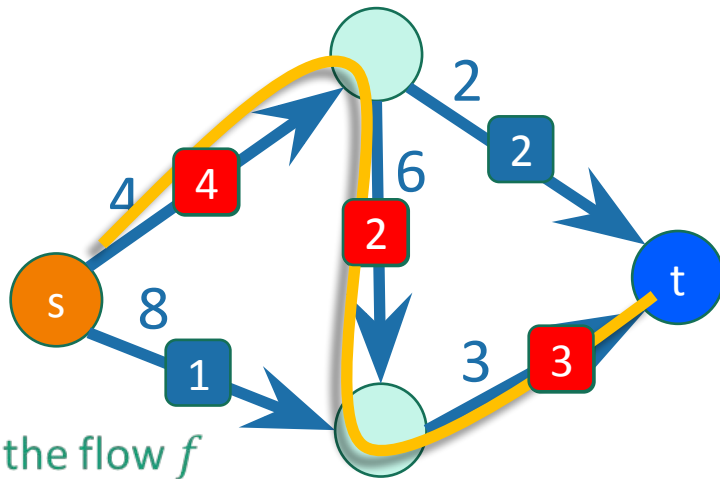
- **Definition:** A path from s to t in the residual network is called an **augmenting path**.
- **Claim:** If there is an augmenting path, we can increase the flow along that path.



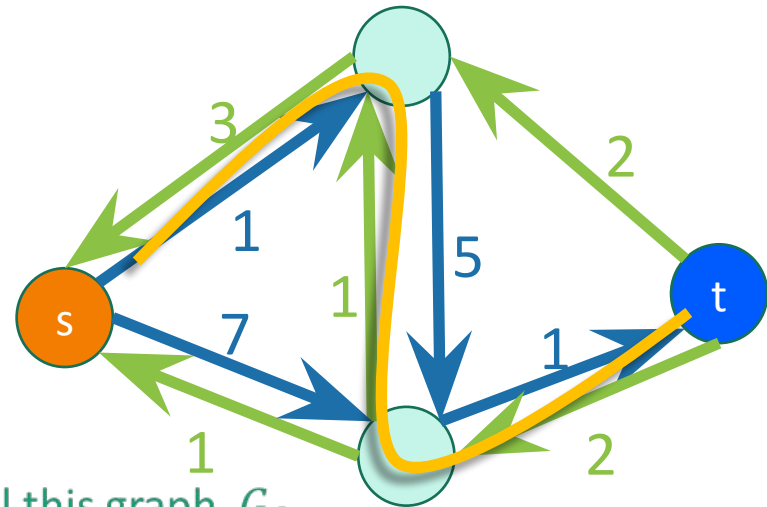
claim:

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



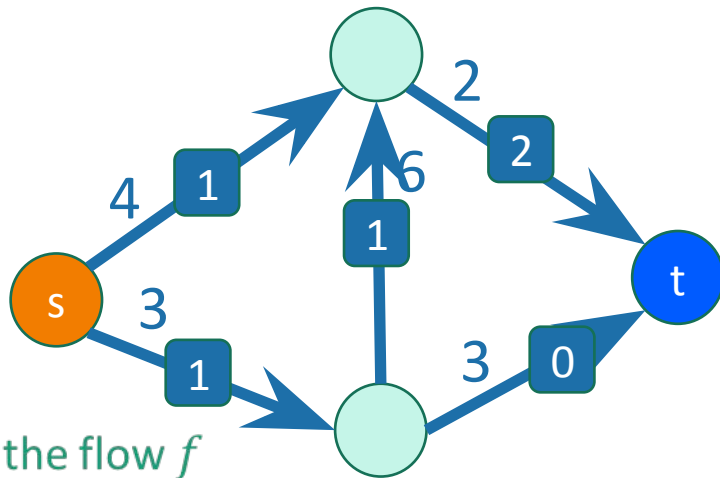
Call this graph G_f

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

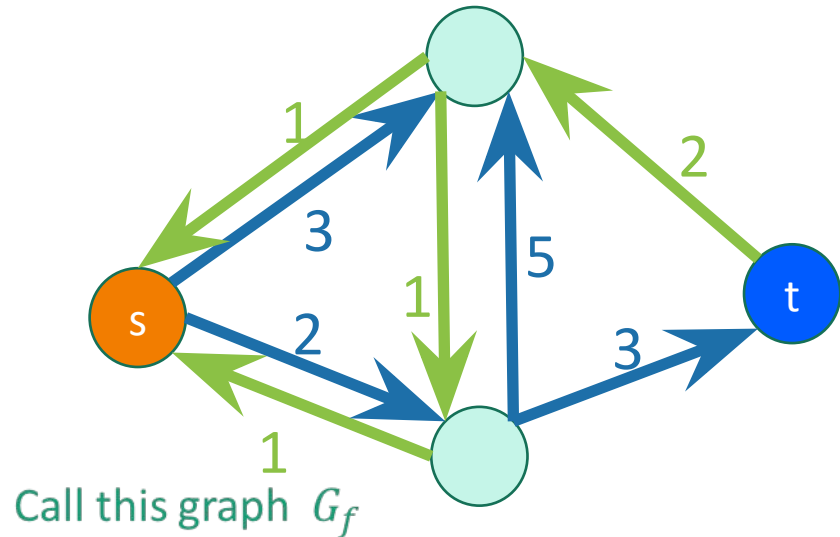
claim:

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



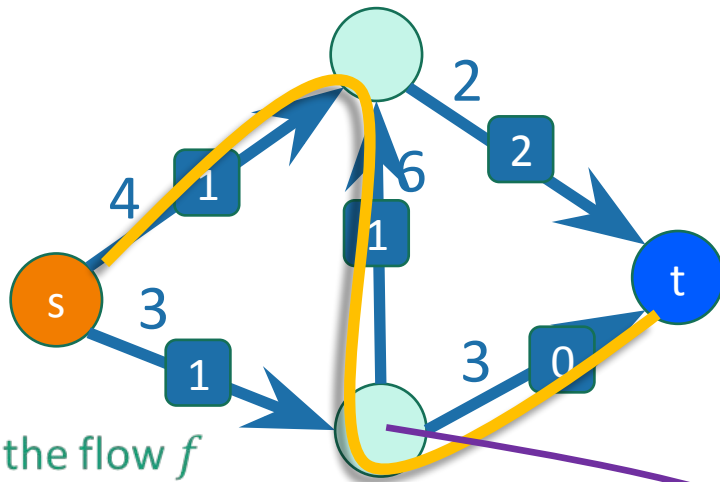
Call this graph G_f

I changed some of the weights and edge directions.

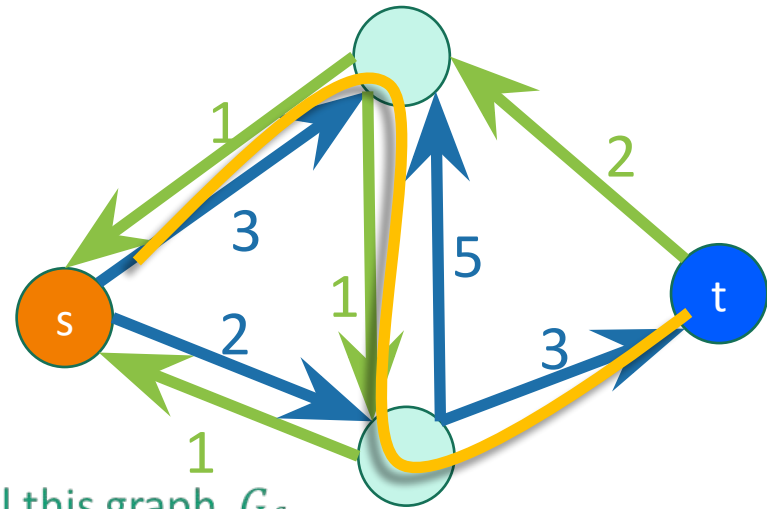
claim:

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!

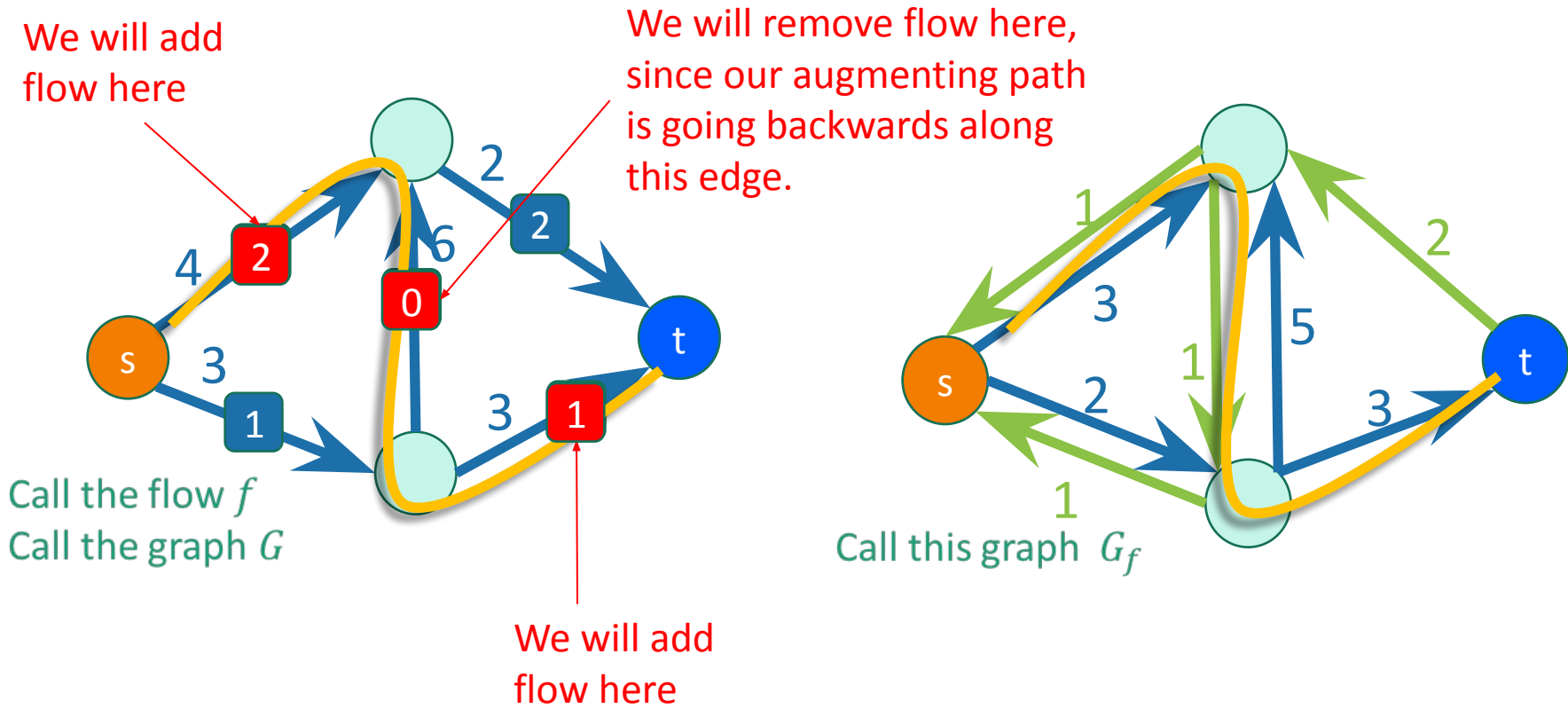
- For example, that will mess up the conservation of stuff at this vertex.

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:

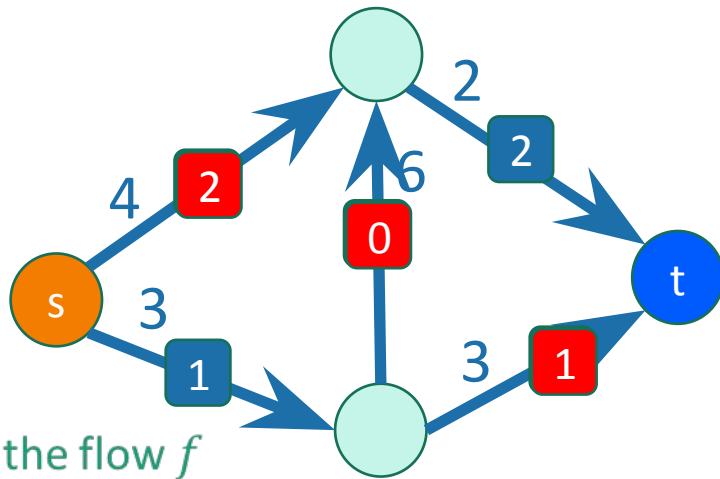


claim:

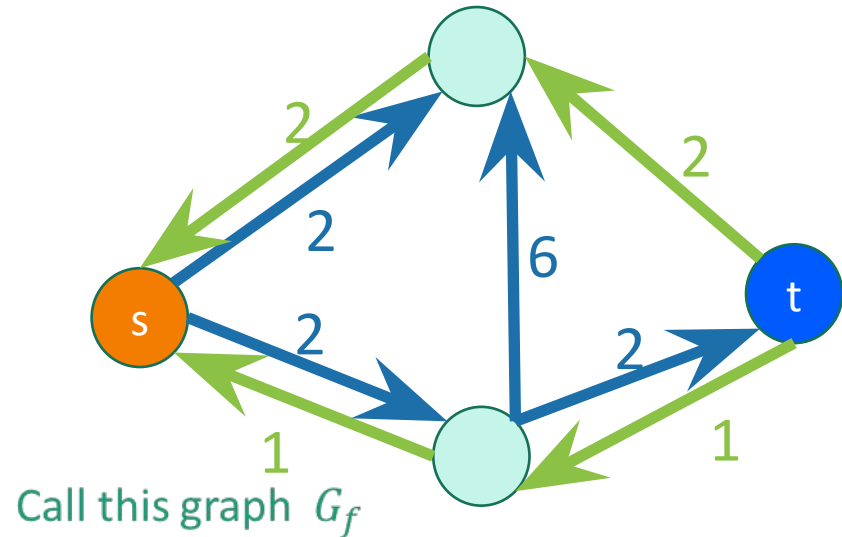
if there is an augmenting path, we can increase the flow along that path.

- In this case we do something a bit different:

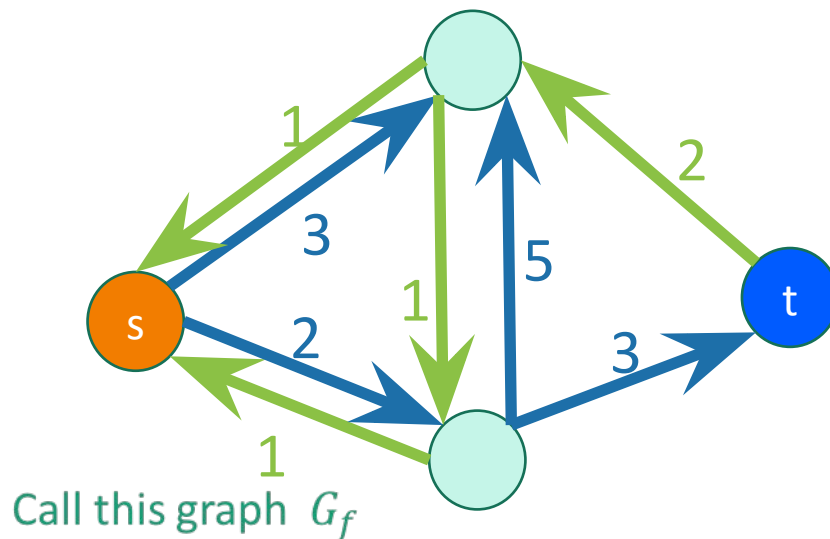
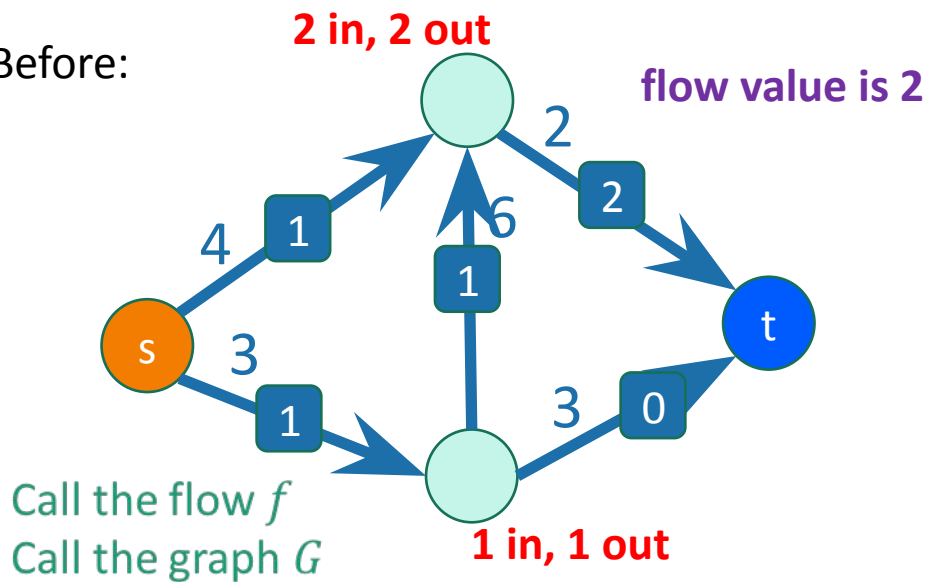
Then we'll update the residual graph:



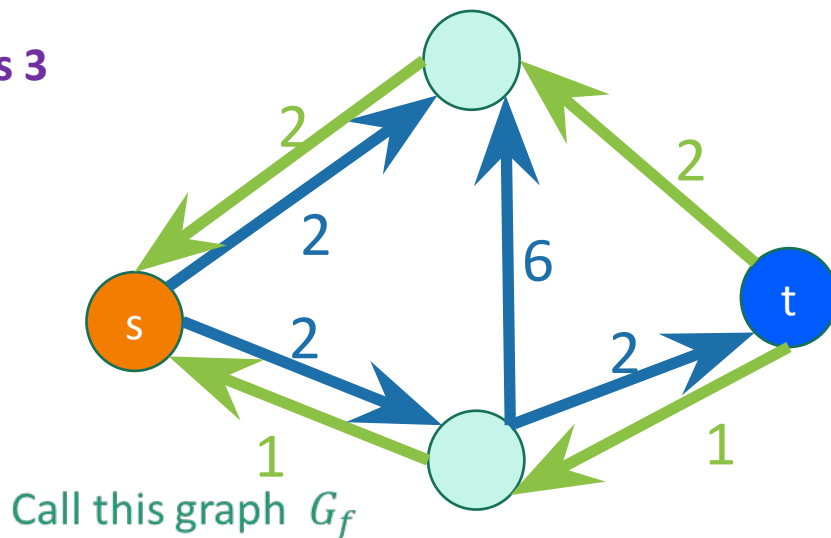
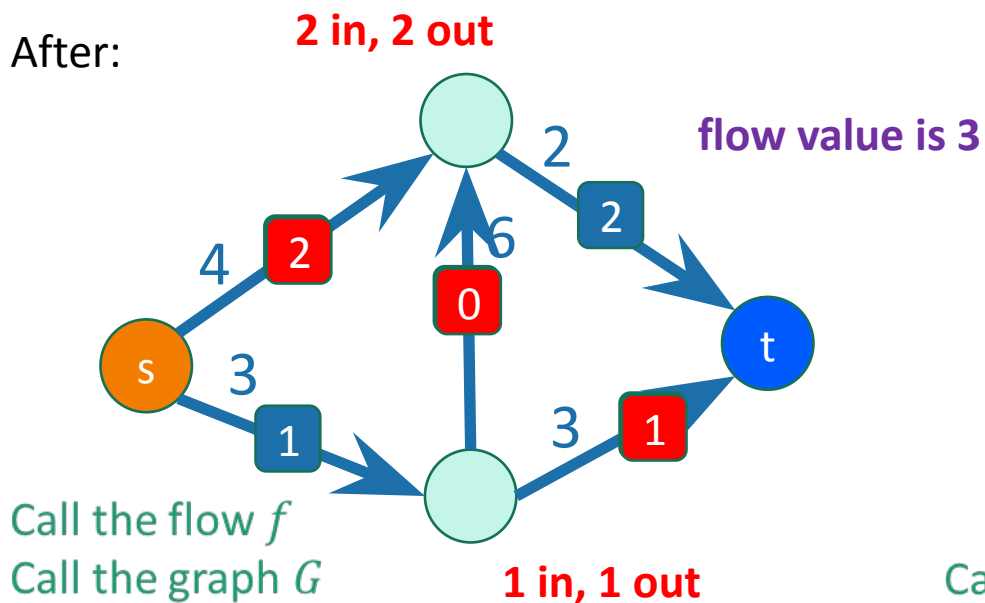
Call the flow f
Call the graph G



Before:



After:



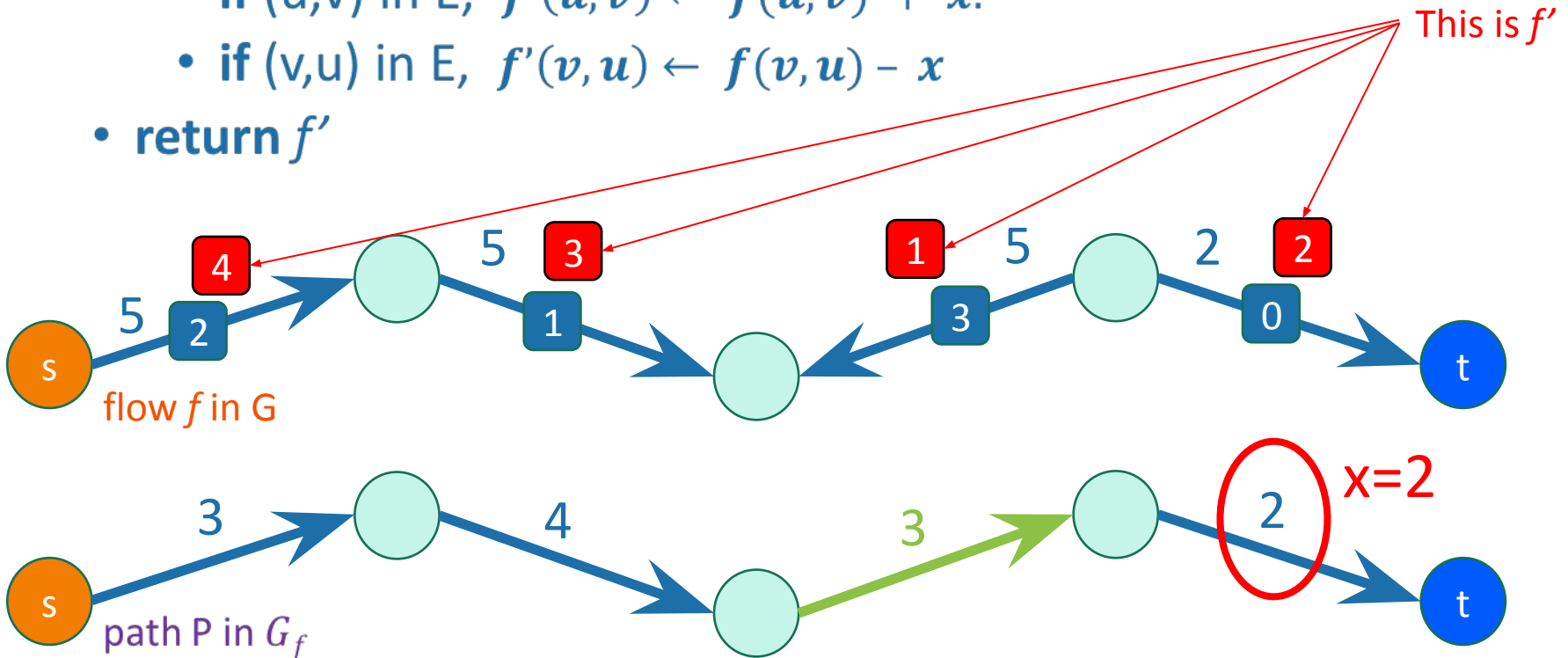
Still a legit flow, but with a bigger value!

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, $f'(u,v) \leftarrow f(u,v) + x$.
 - 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

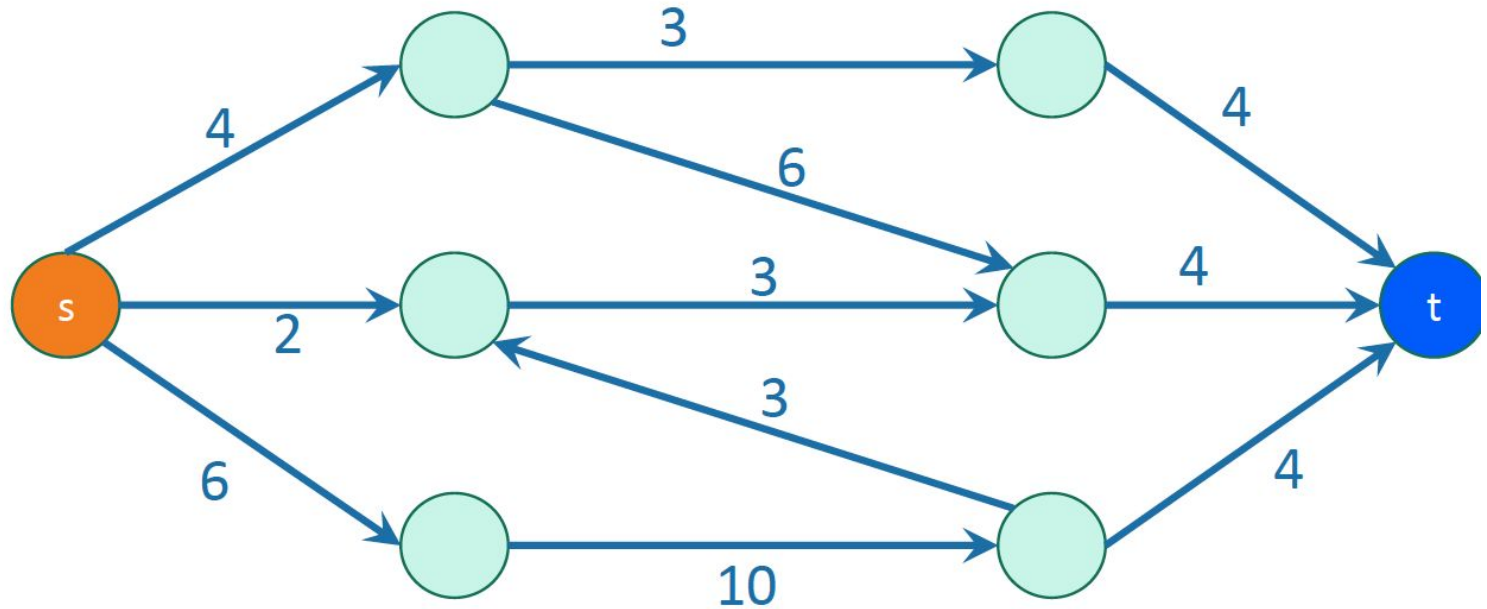
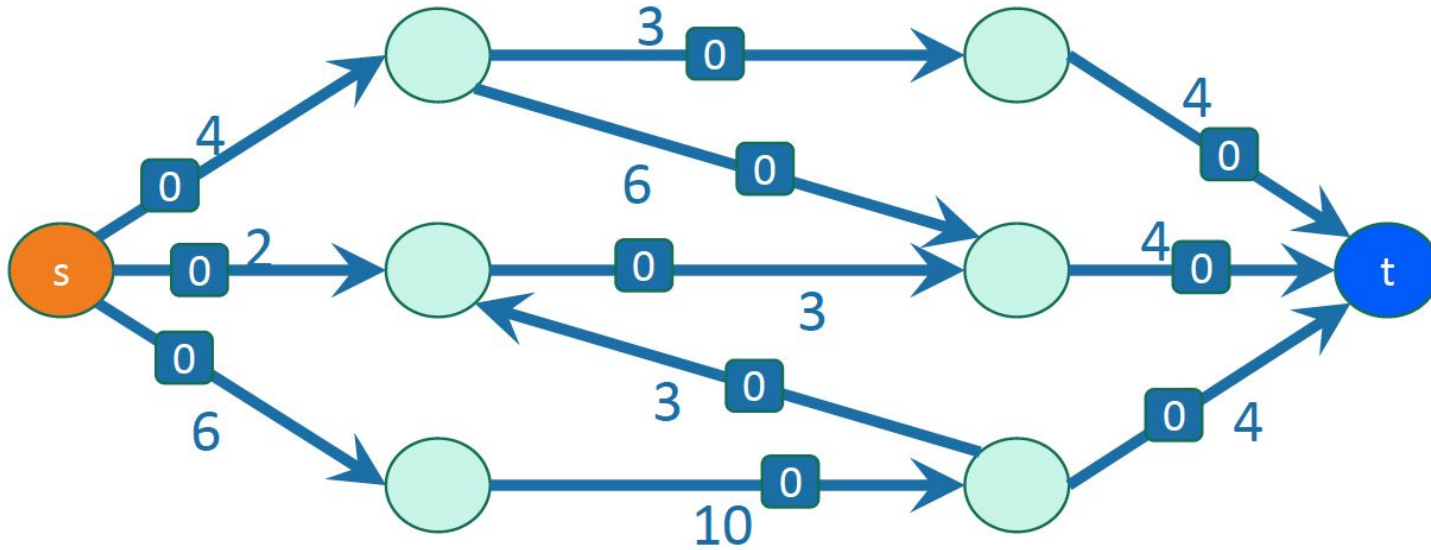
// eg, use BFS

- $f \leftarrow$ **increaseFlow**(P, f)

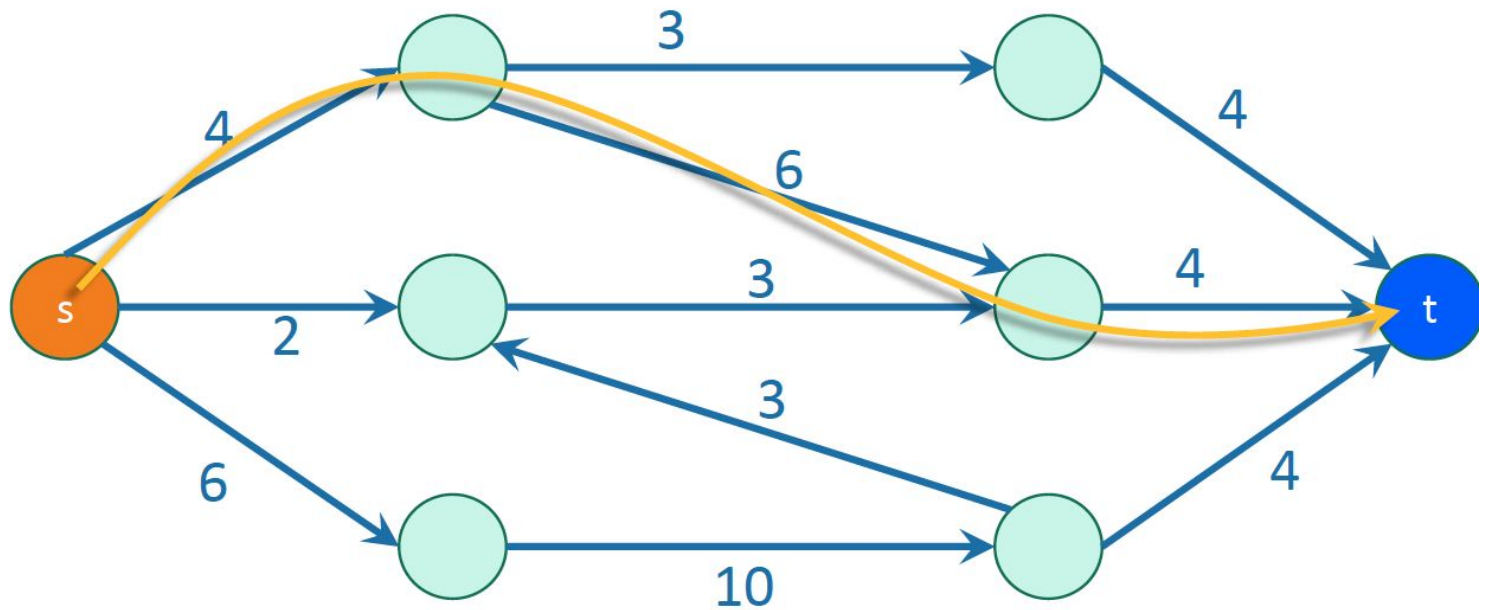
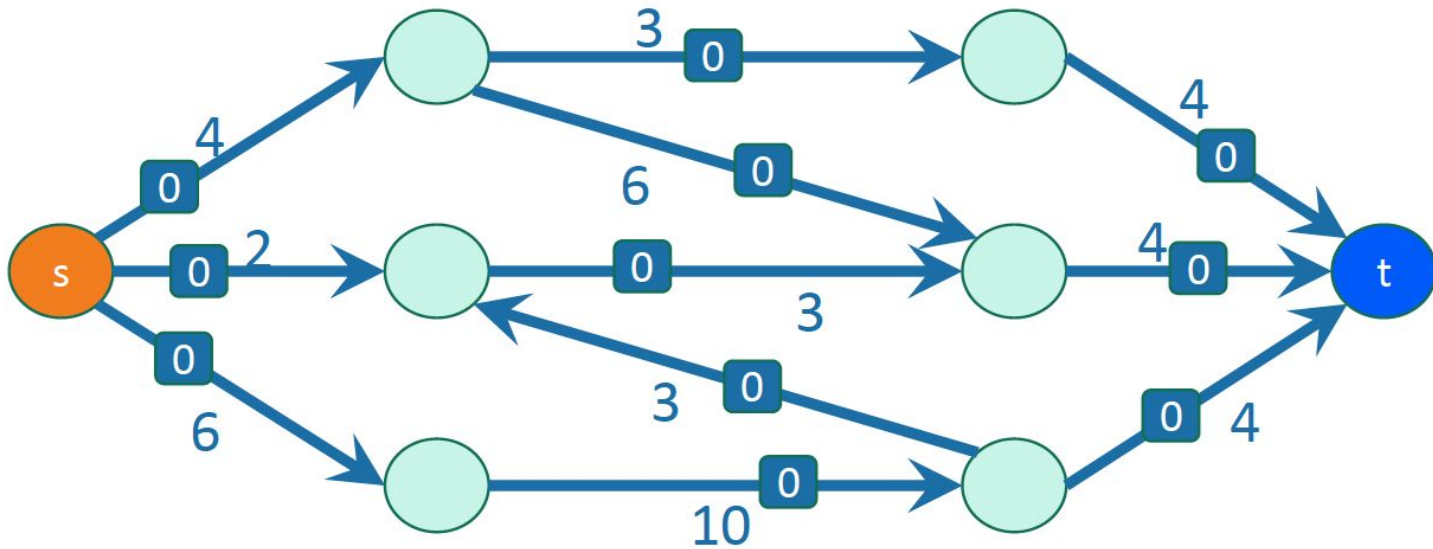
- update G_f

- **return** f

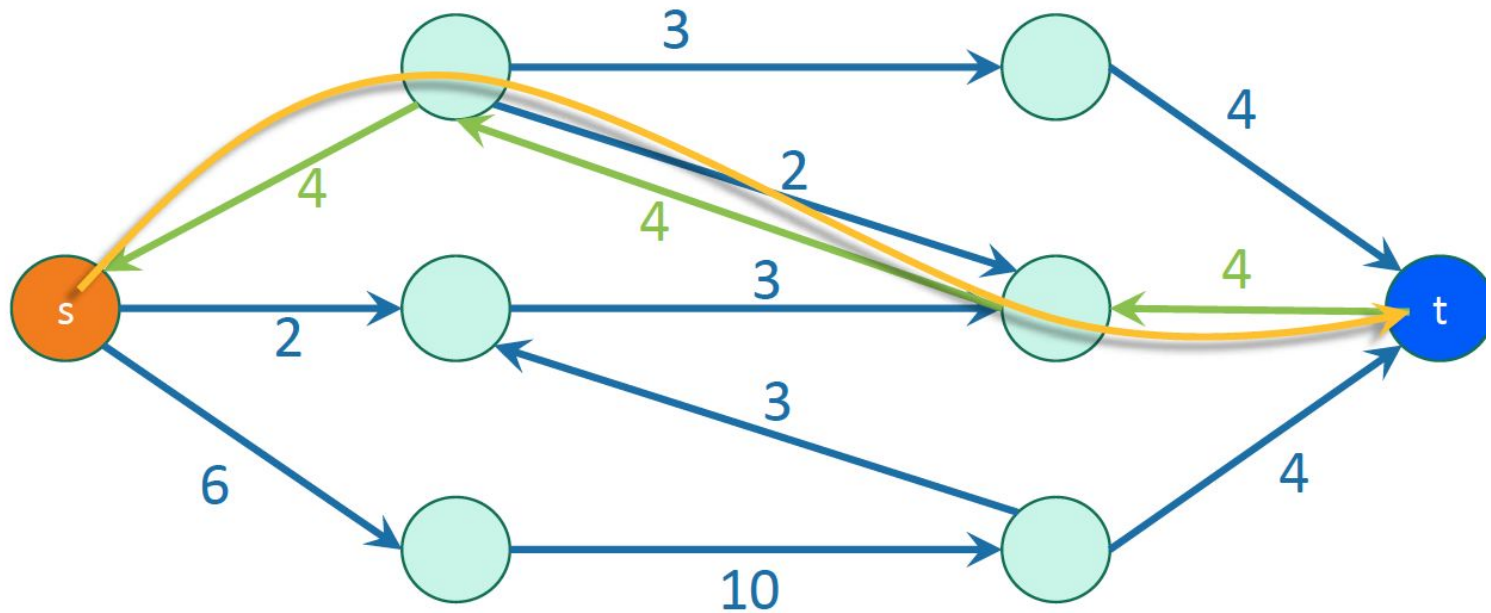
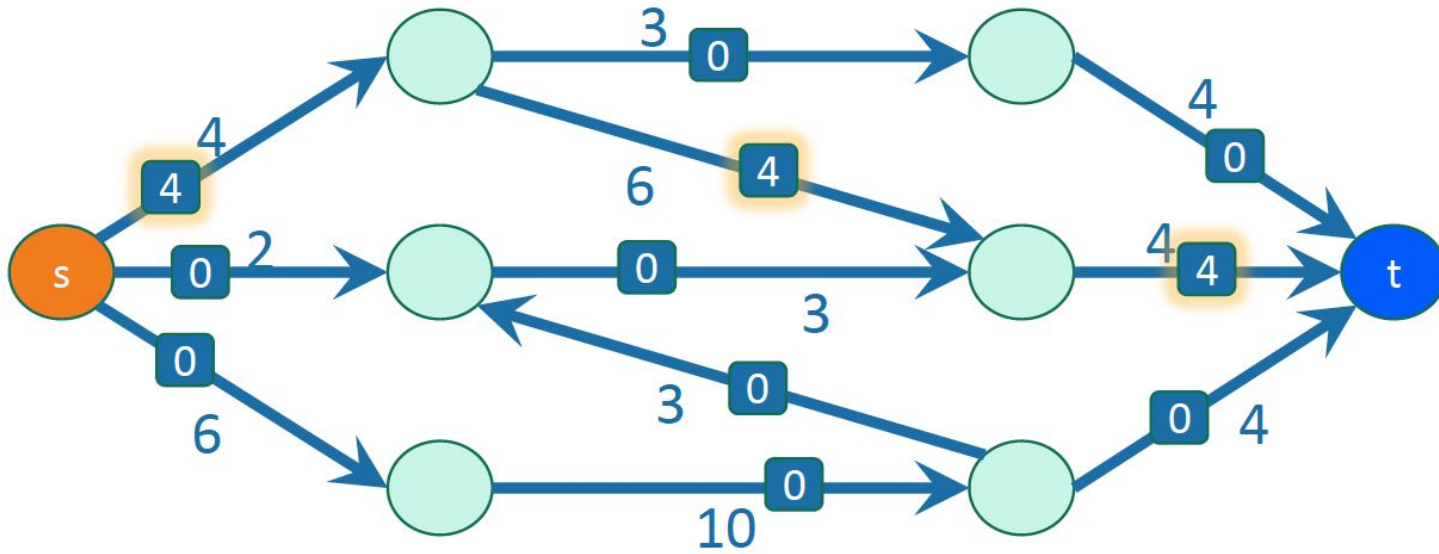
Example of Ford-Fulkerson



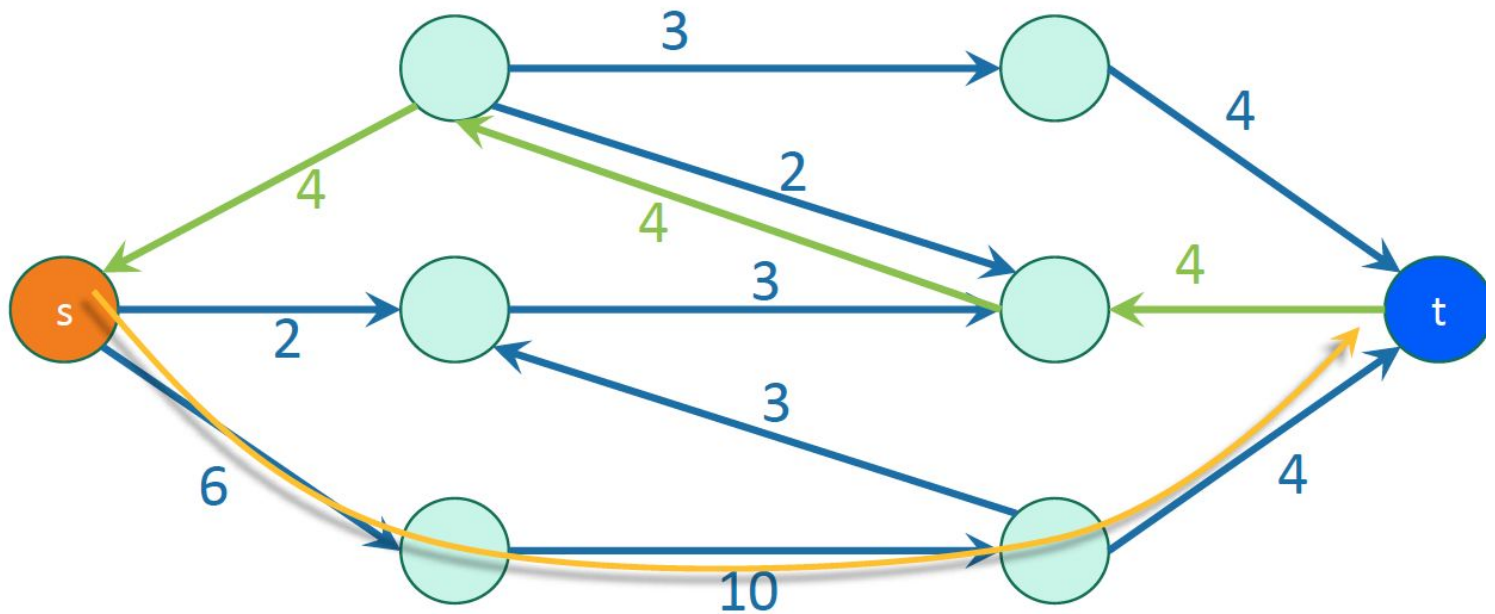
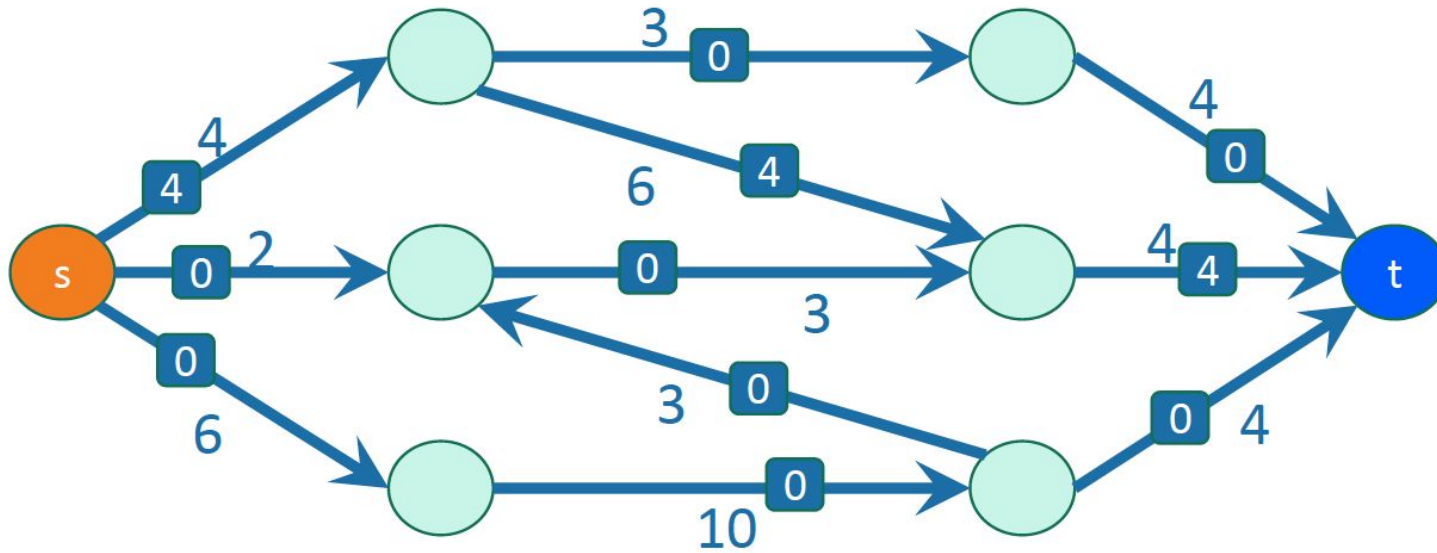
Example of Ford-Fulkerson



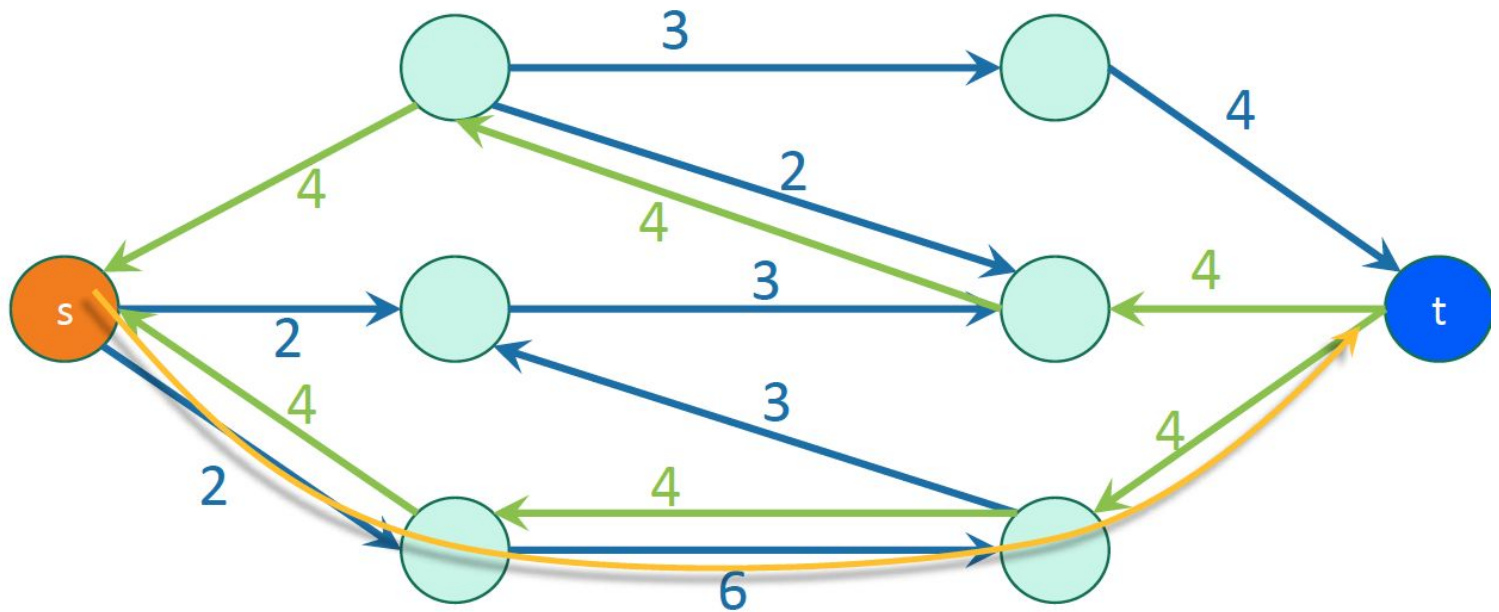
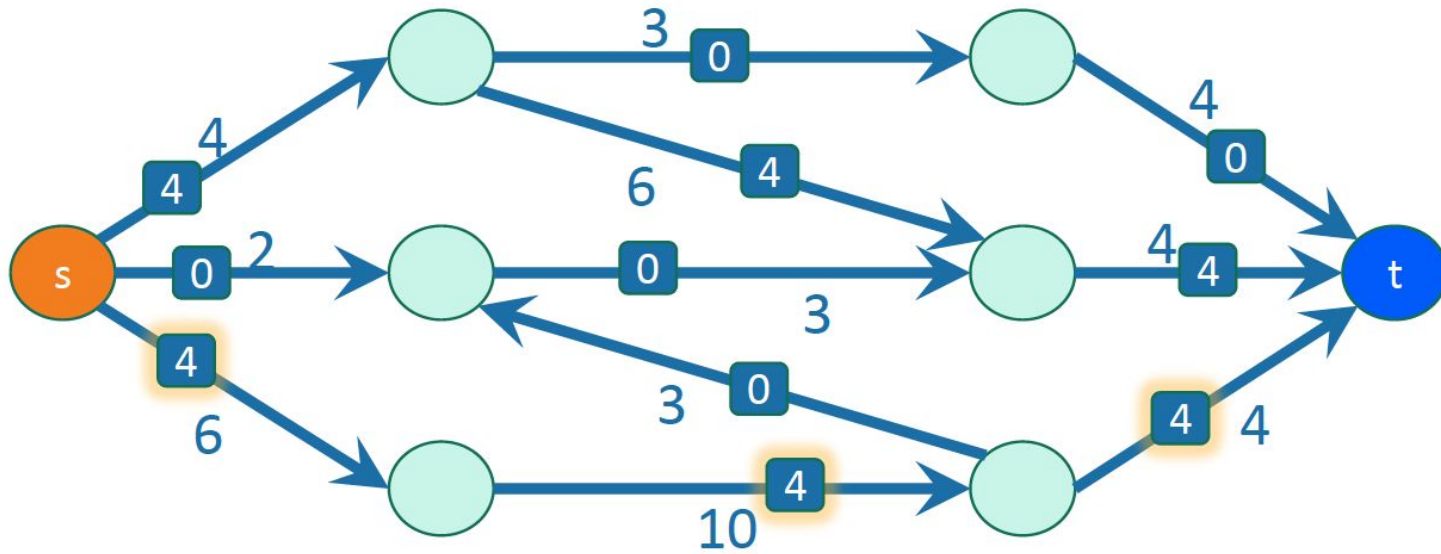
Example of Ford-Fulkerson



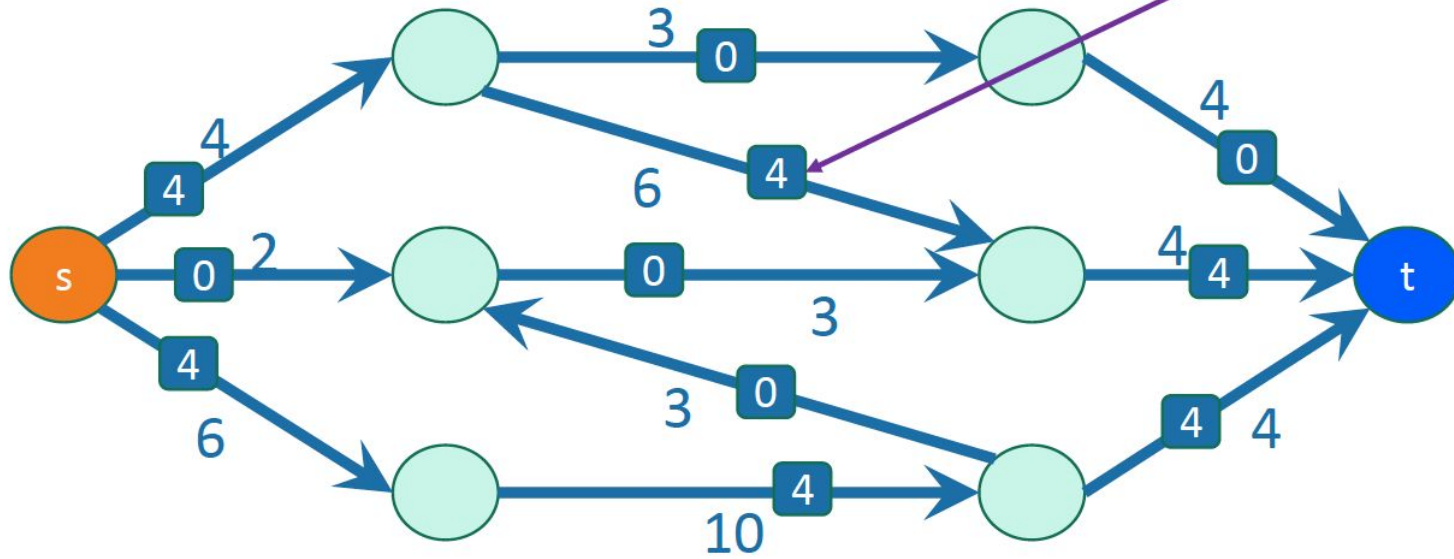
Example of Ford-Fulkerson



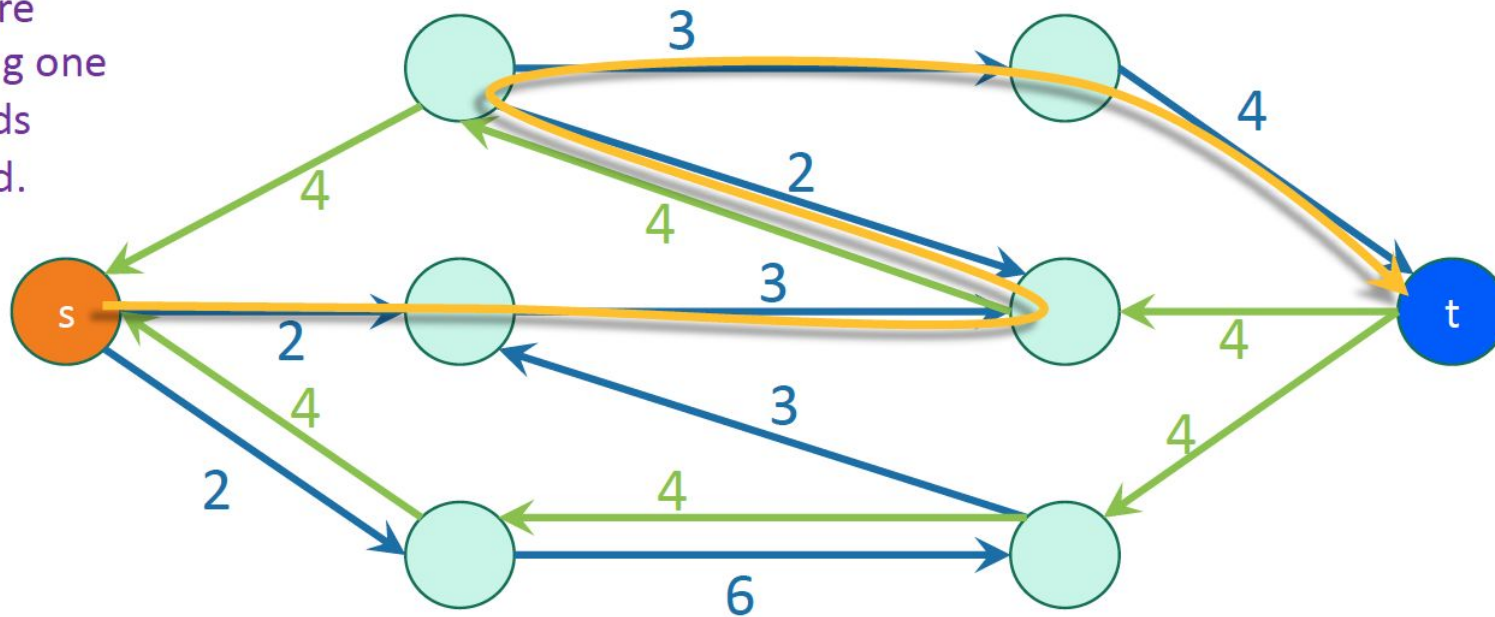
Example of Ford-Fulkerson



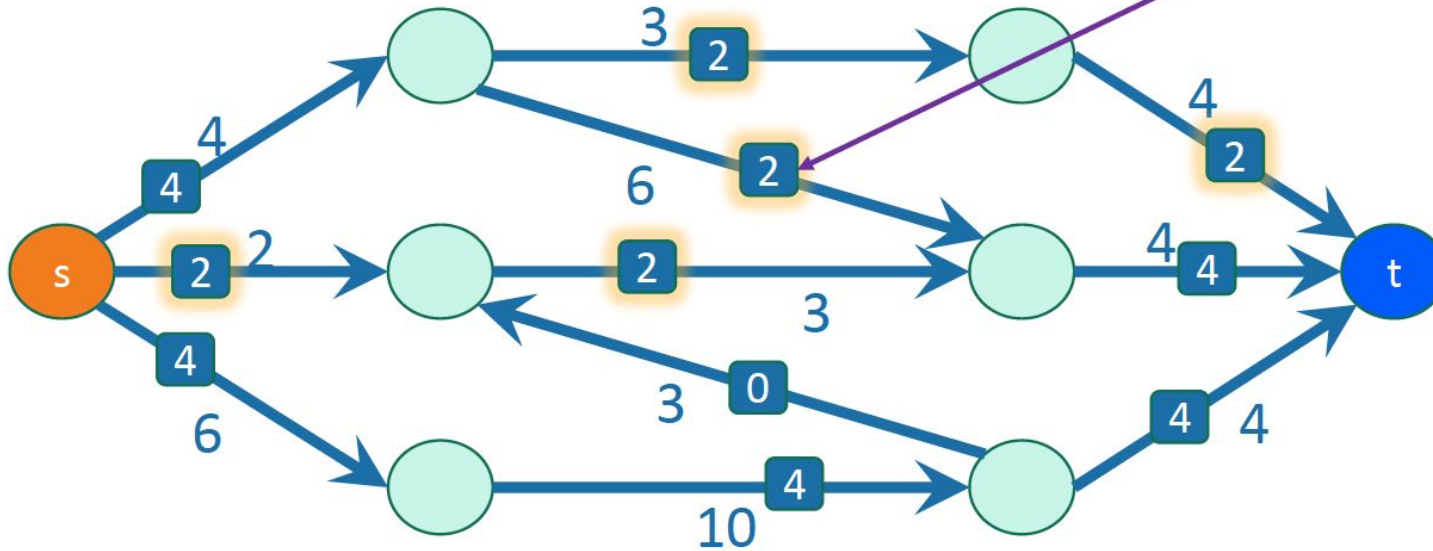
Example of Ford-Fulkerson



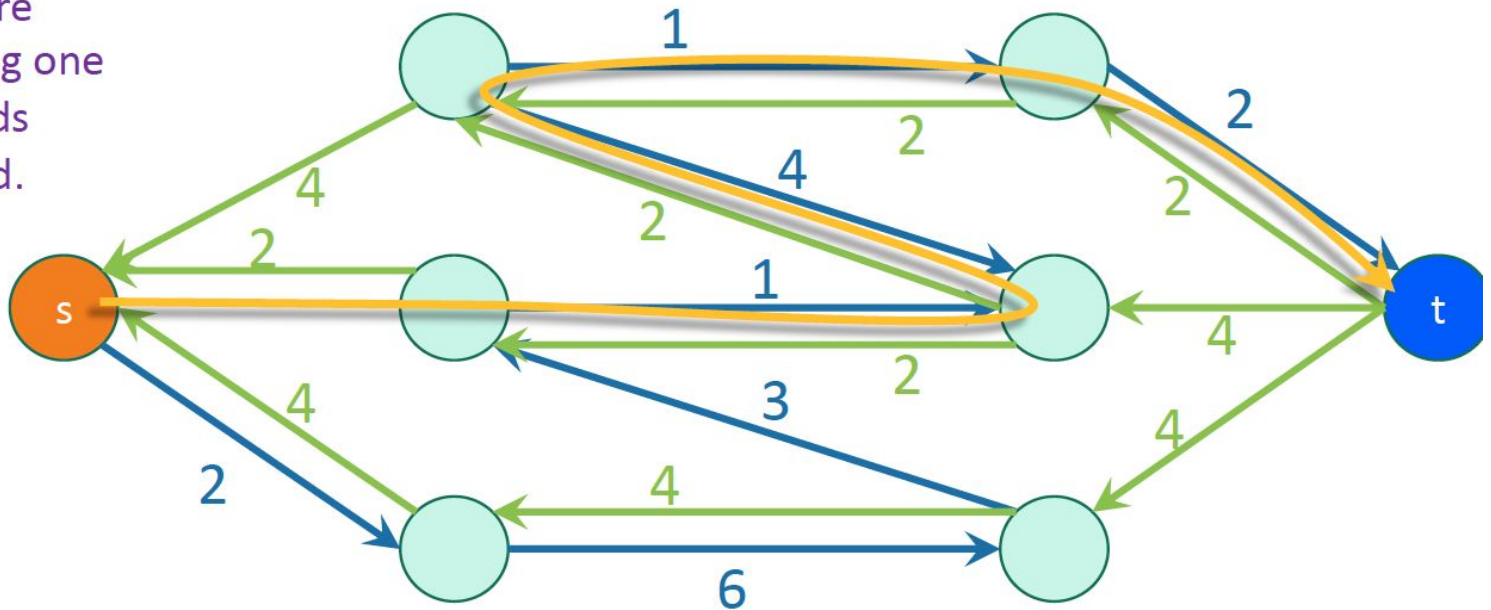
Notice that we're going back along one of the backwards edges we added.



Example of Ford-Fulkerson

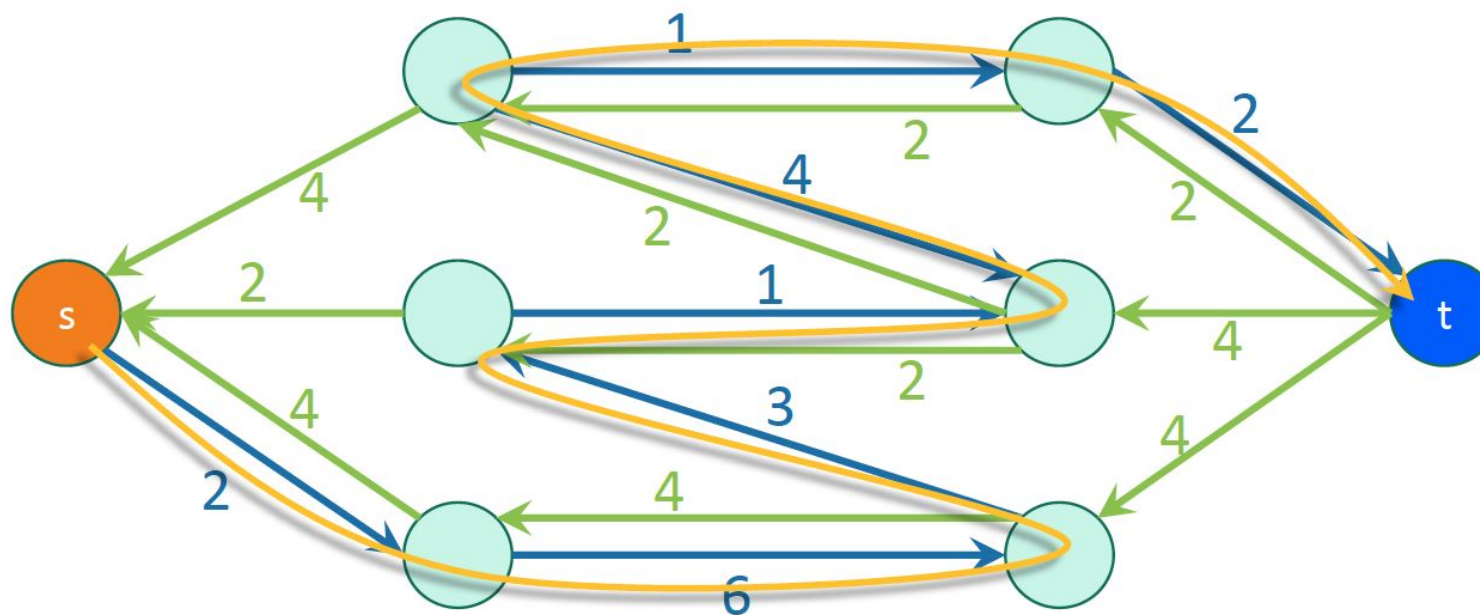
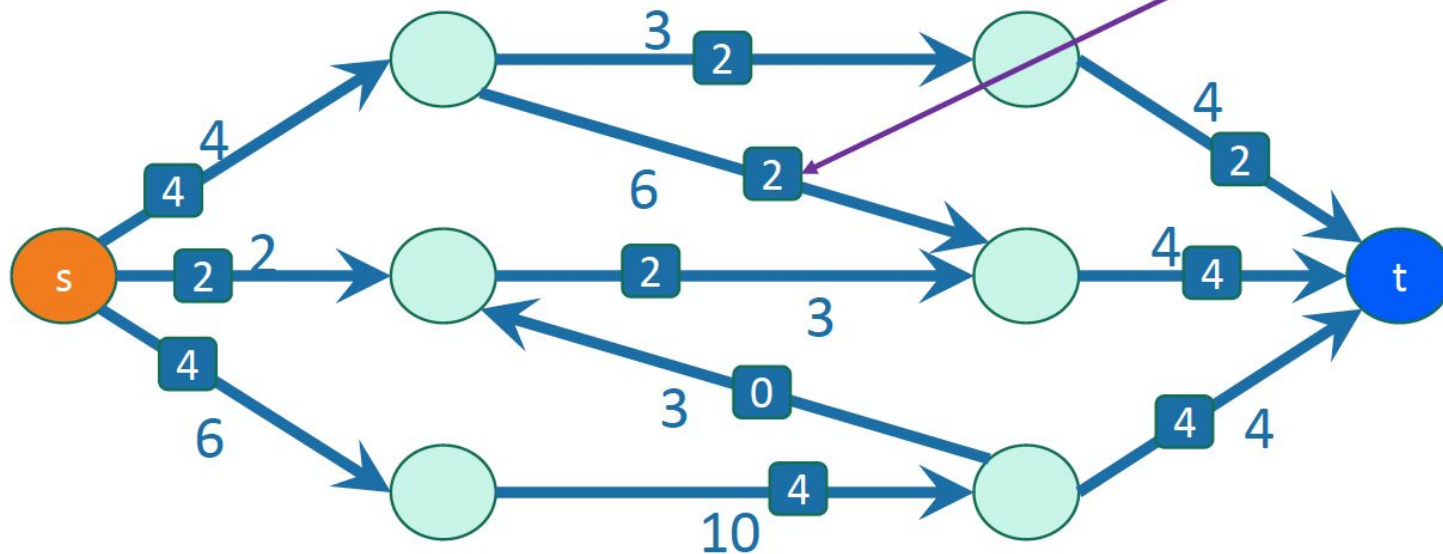


Notice that we're going back along one of the backwards edges we added.



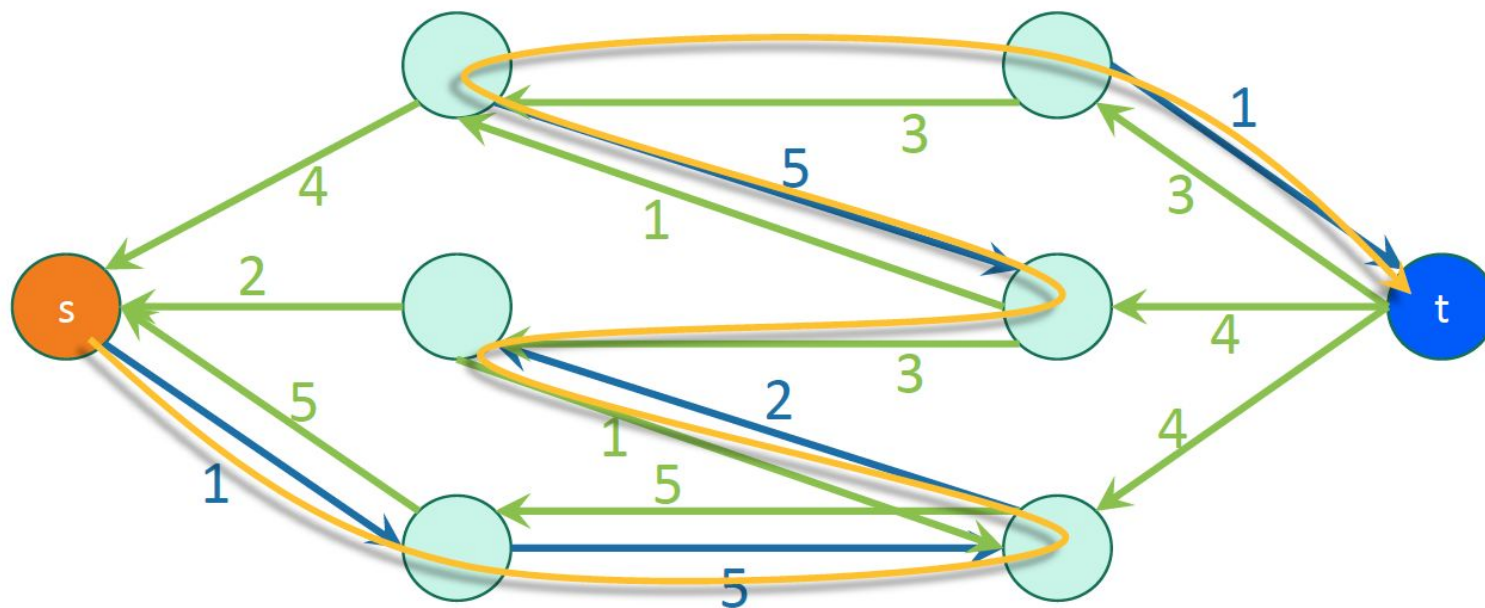
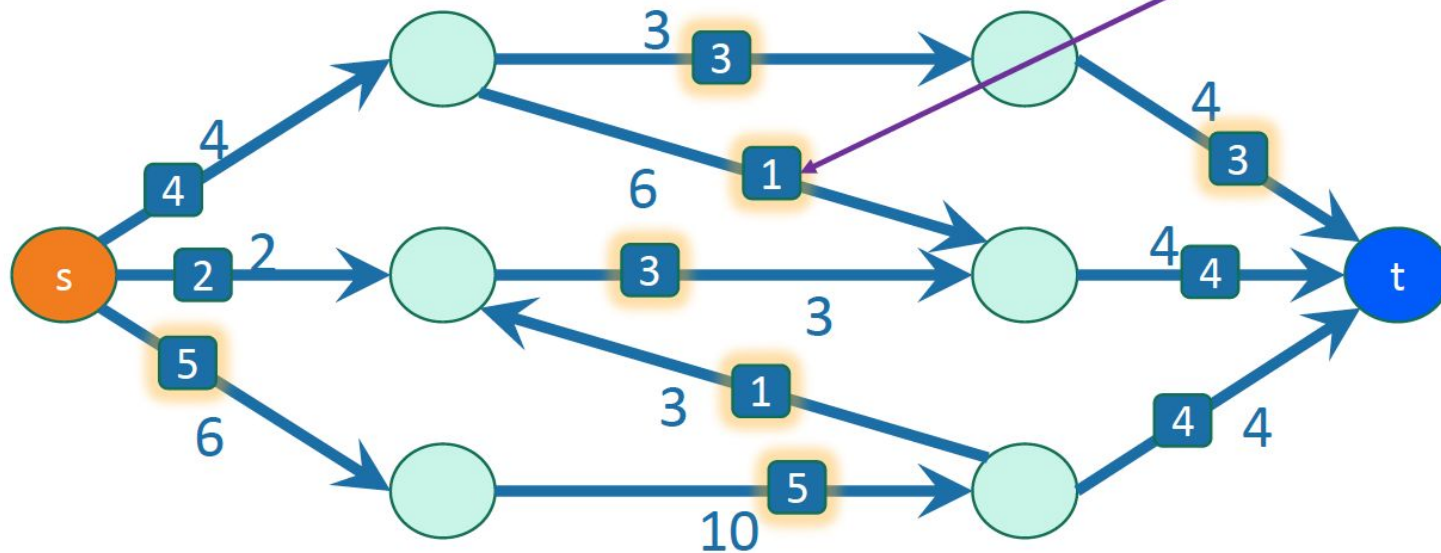
Example of Ford-Fulkerson

We will remove flow from this edge AGAIN.

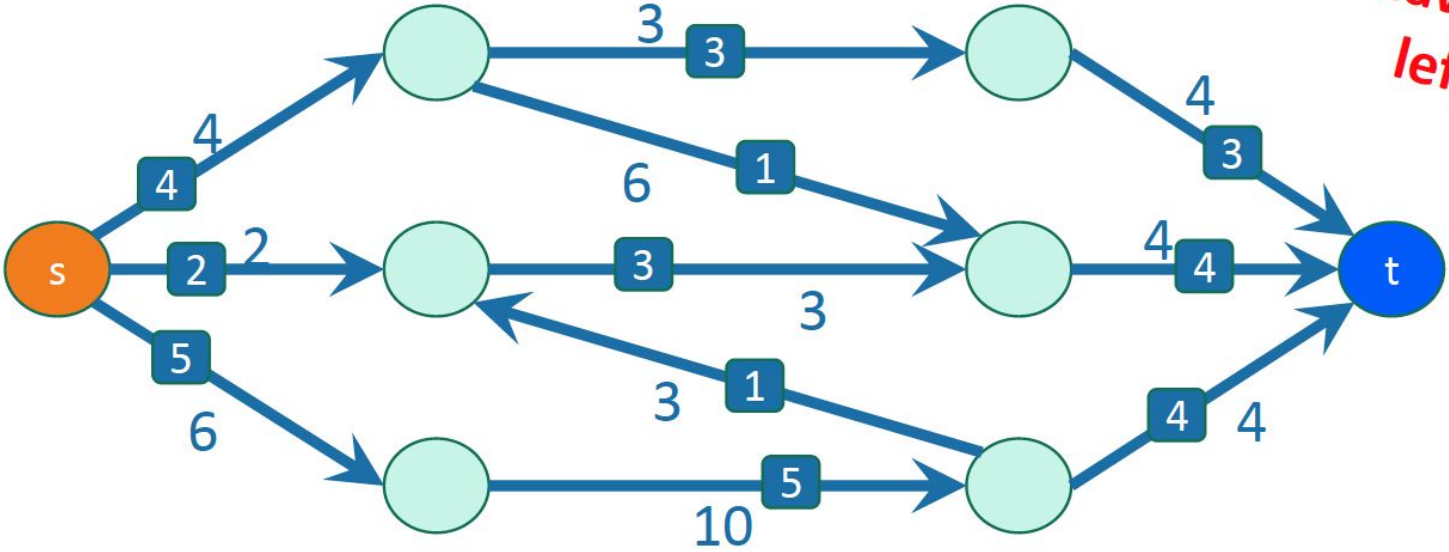


Example of Ford-Fulkerson

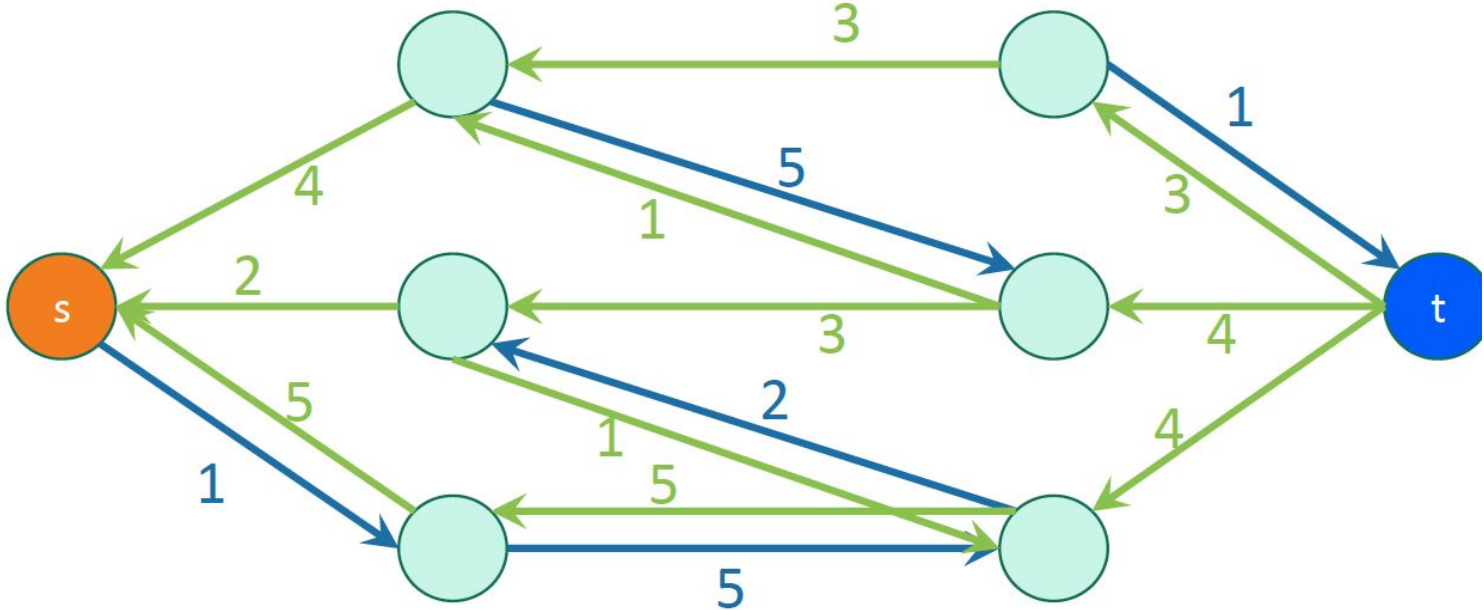
We will remove flow from this edge AGAIN.



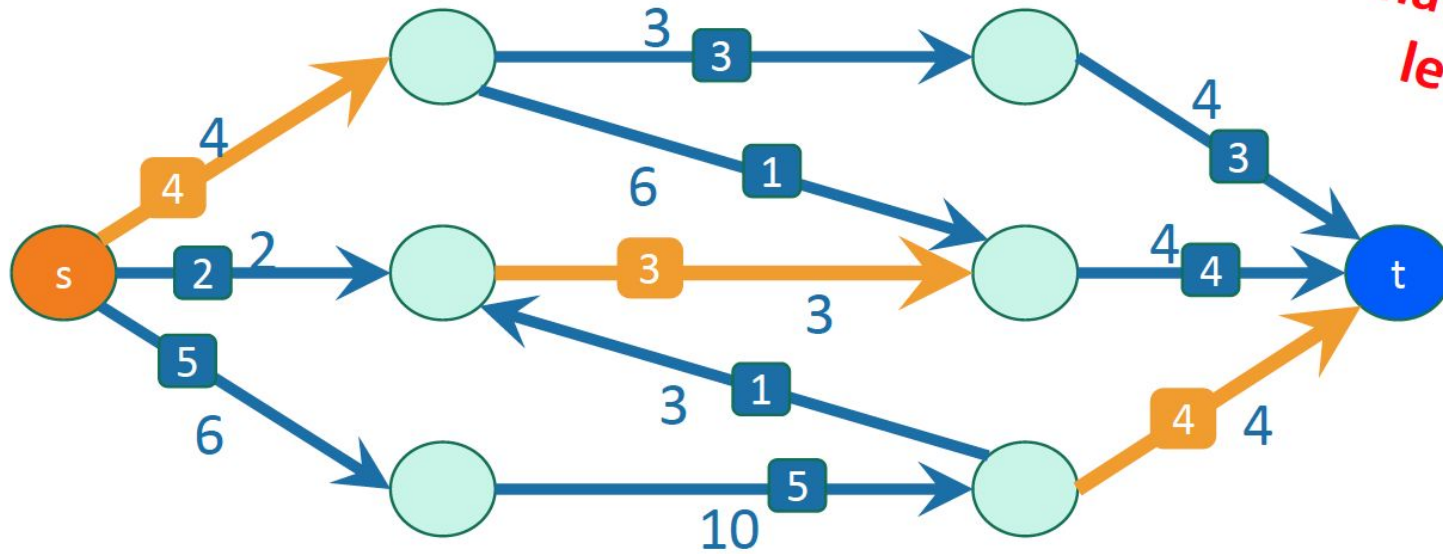
Example of Ford-Fulkerson



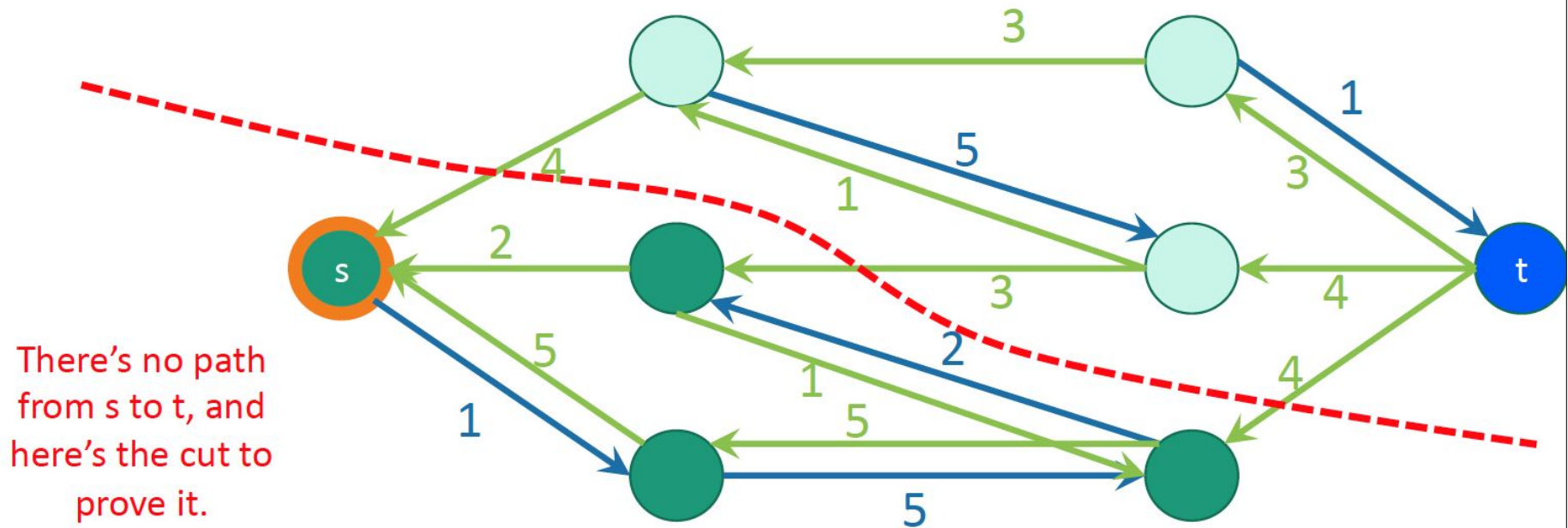
Now we have nothing left to do!



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.

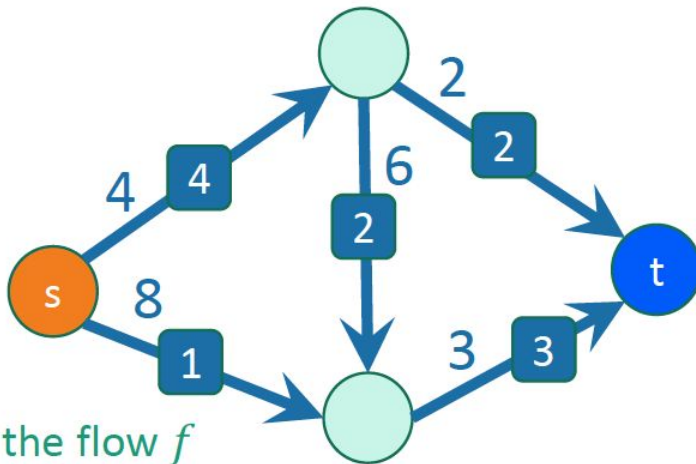
Why does Ford-Fulkerson work?

- 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.

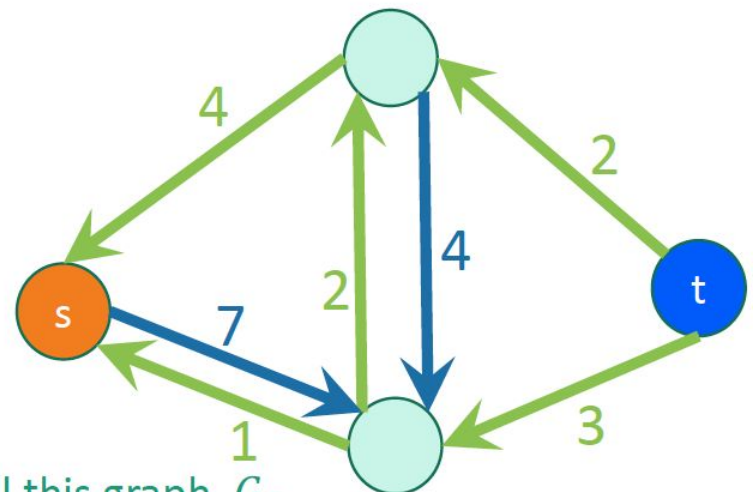
No augmenting path \Rightarrow max flow.

- 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

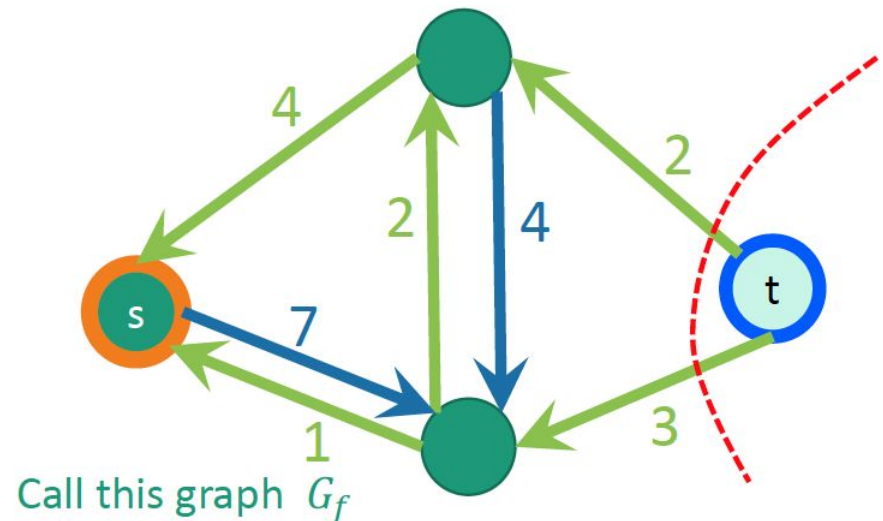
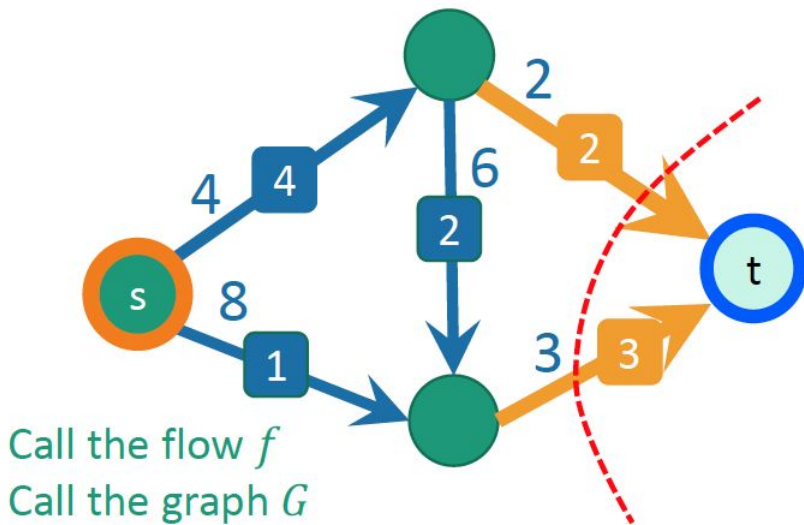
No augmenting path \Rightarrow max flow.

- 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



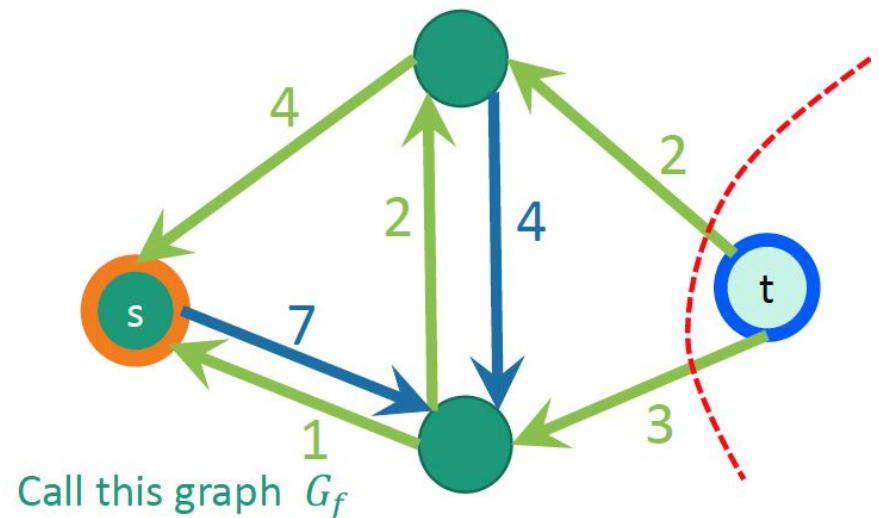
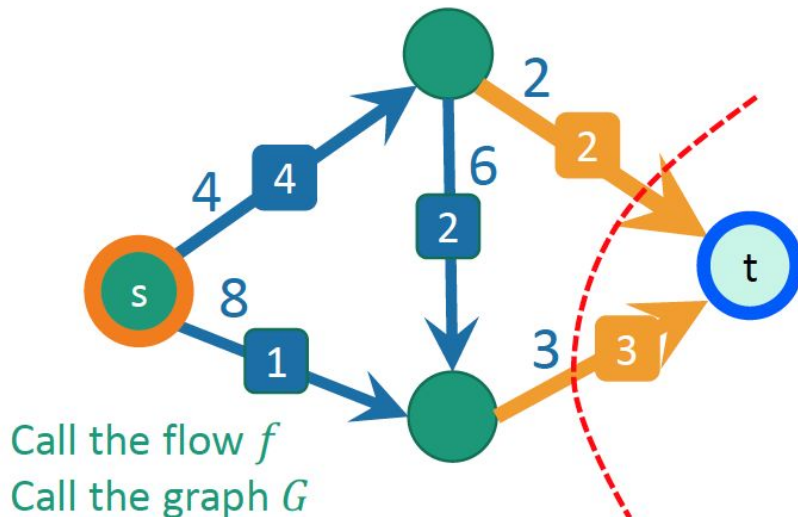
No augmenting path \Rightarrow max flow.

- 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:

t lives here

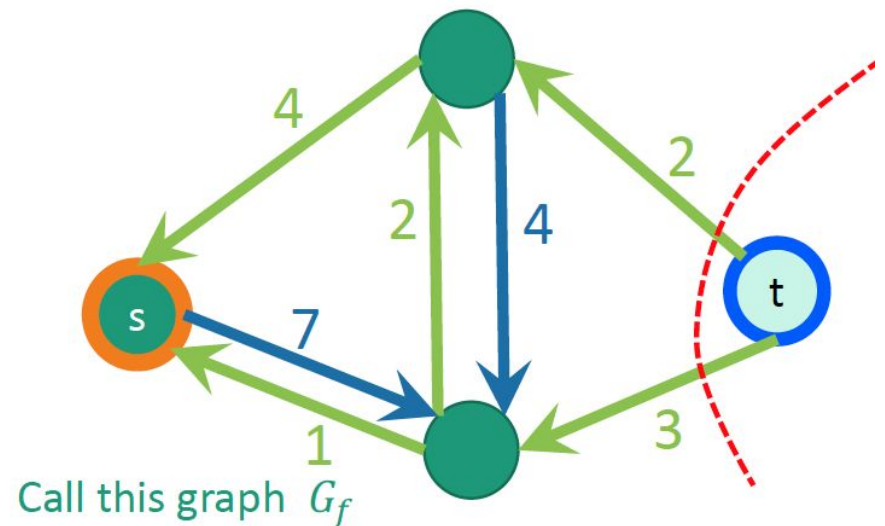
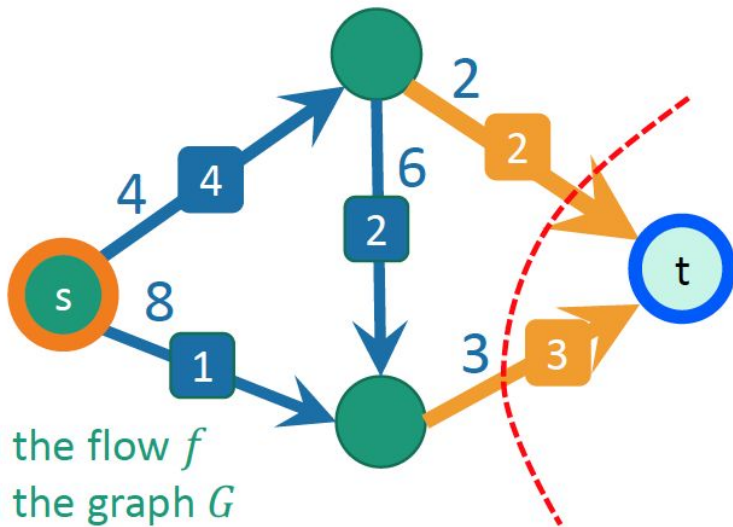
{things reachable from s }, {things not reachable from s }

- 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

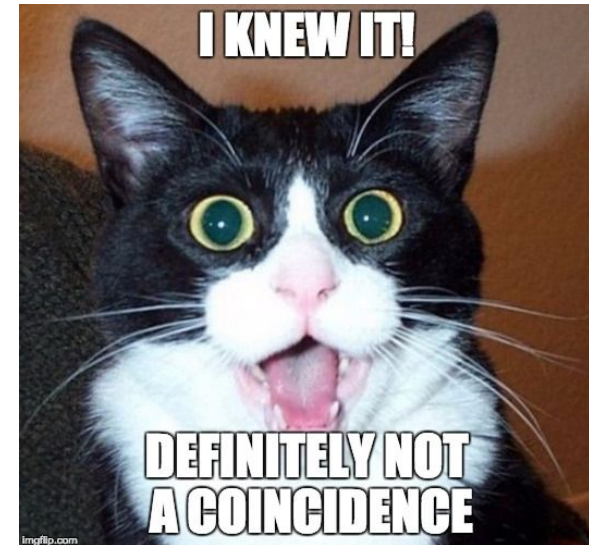
- Therefore f is a max flow!
- Thus, when Ford-Fulkerson stops, it's found the maximum flow.



Min-Cut Max-Flow Theorem

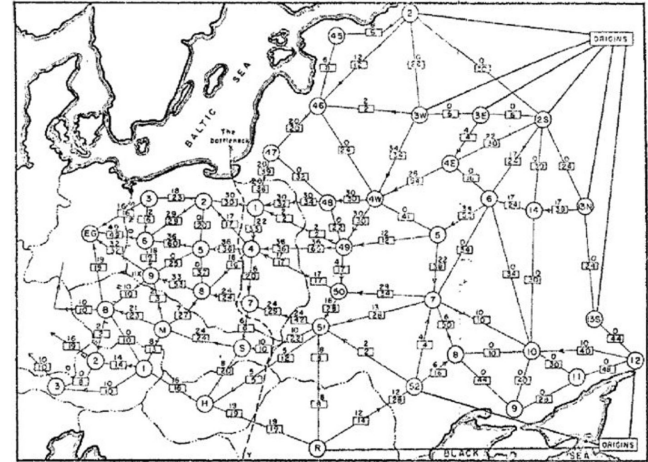
$\text{max flow} \geq \text{Value of } f = \text{cost of this cut} \geq \text{min cut} \geq \text{max flow}$

So everything is equal and min cut = max flow!



What have we learned?

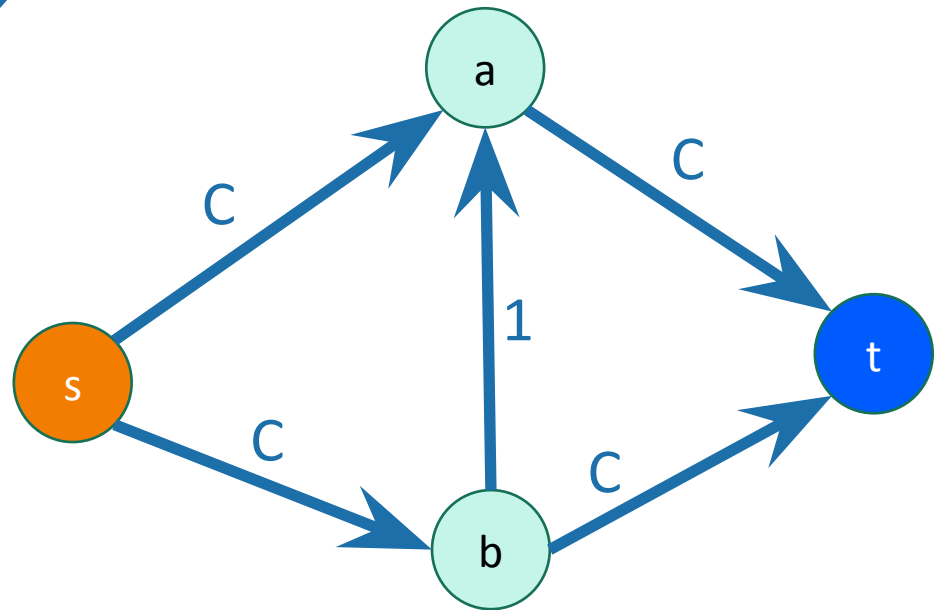
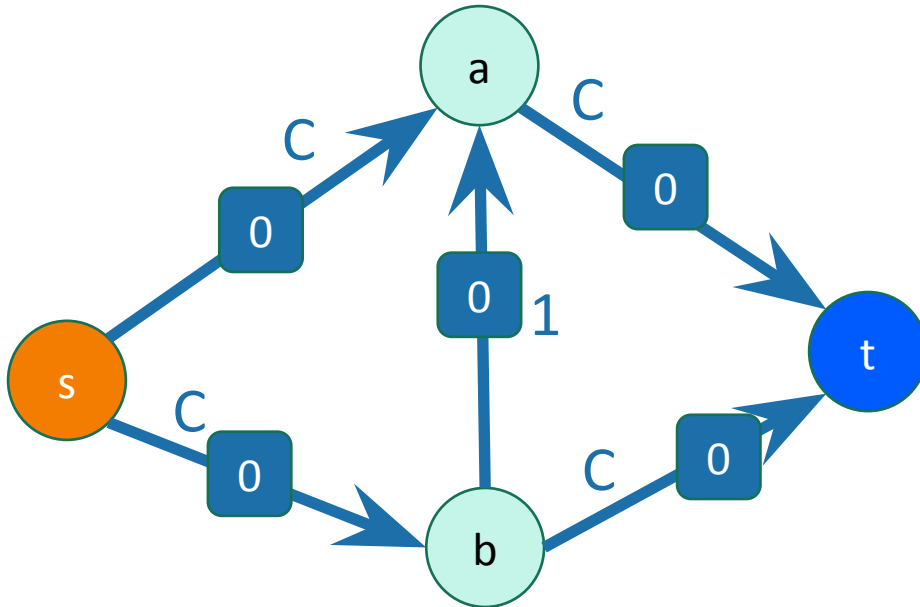
- Max s-t flow is equal to min s-t cut!
 - The USSR and the USA were trying to solve the same problem...
- The Ford-Fulkerson algorithm can find the min-cut/max-flow.
 - Repeatedly improve your flow along an augmenting path.
- How long does this take???



Why should we be concerned?

Suppose we just picked paths arbitrarily.

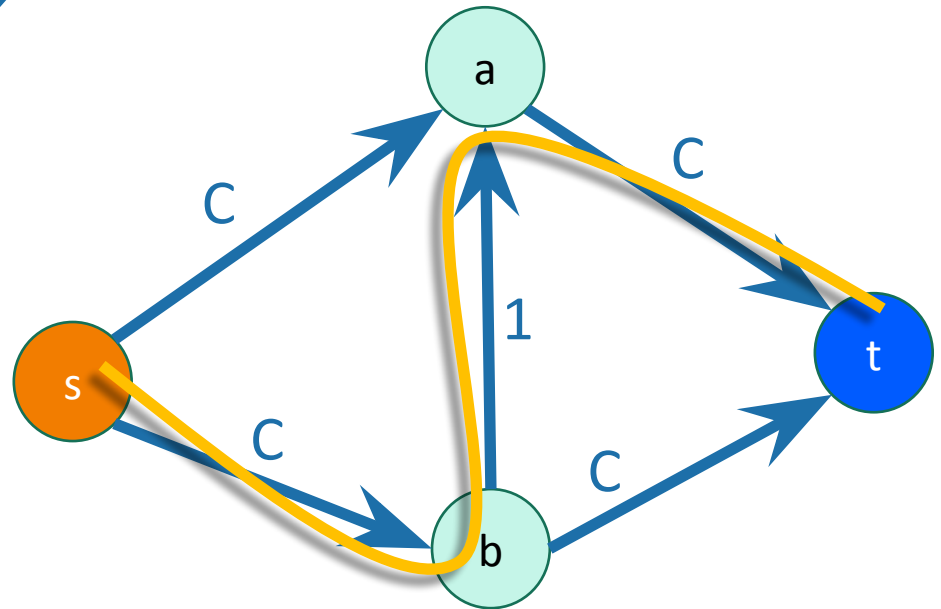
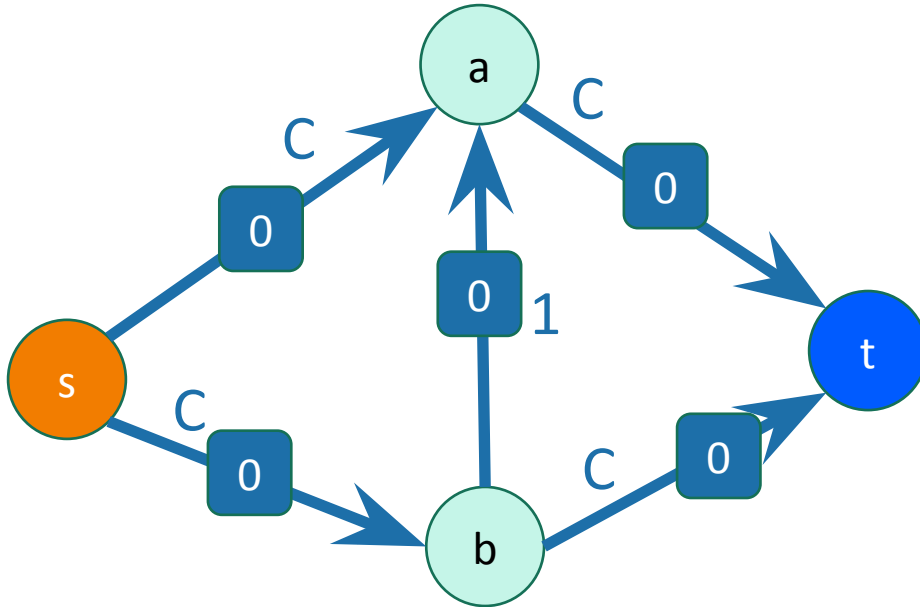
Choose a really big number C .



Why should we be concerned?

Suppose we just picked paths arbitrarily.

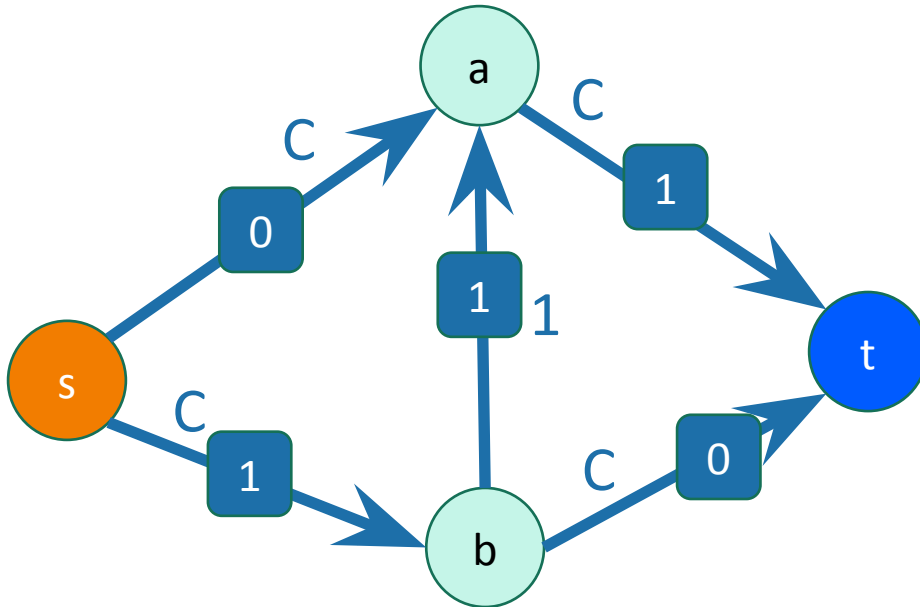
Choose a really big number C .



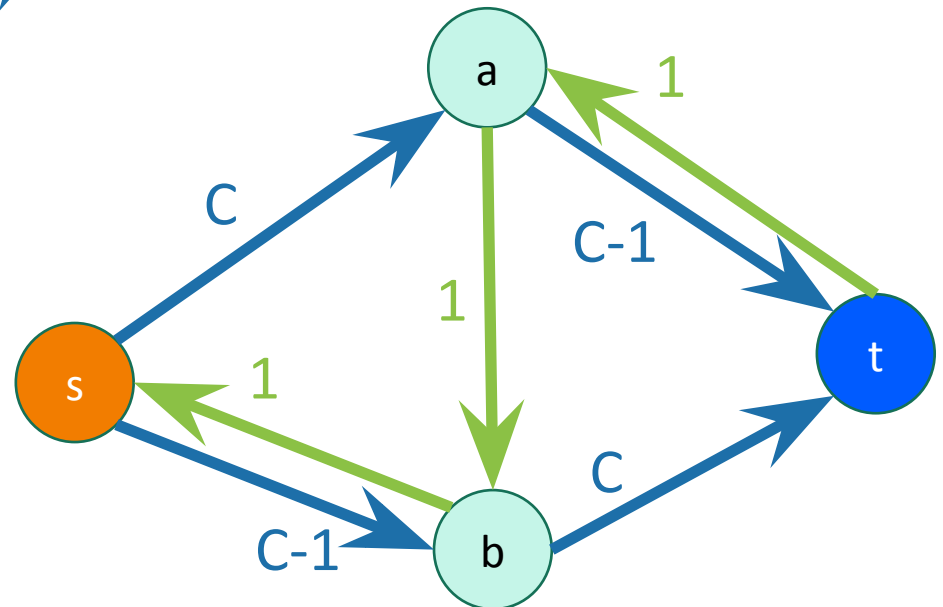
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C .



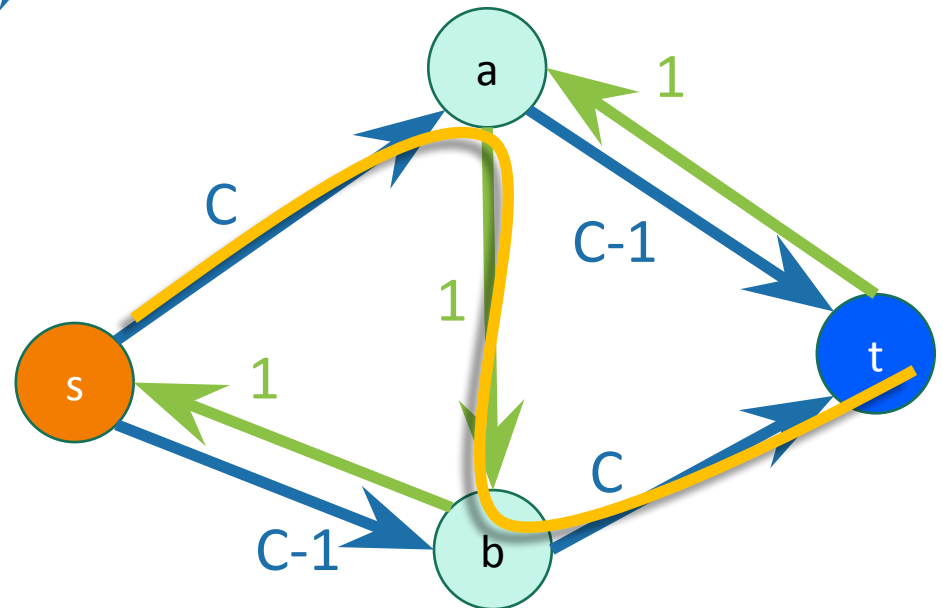
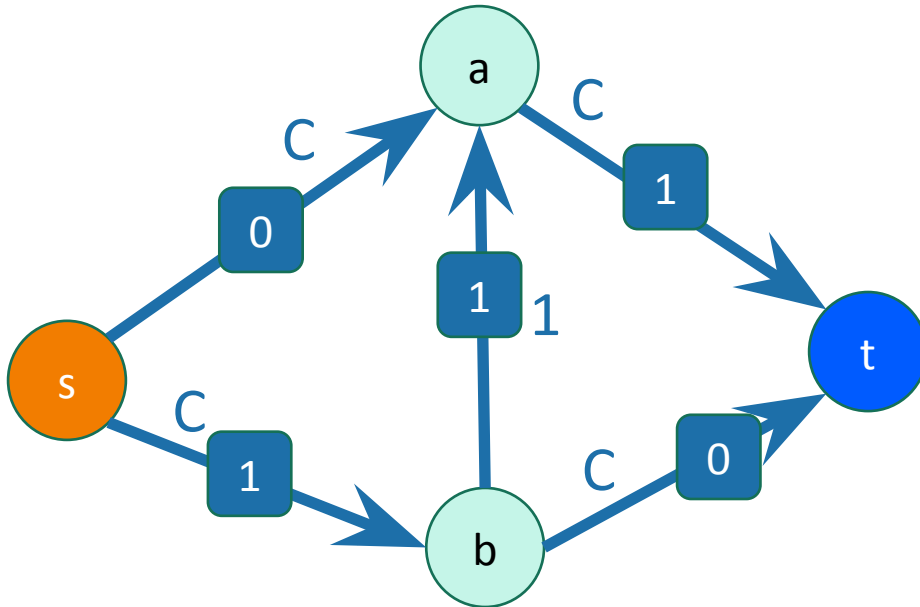
The edge (b,a) disappeared from the residual graph!



Why should we be concerned?

Suppose we just picked paths arbitrarily.

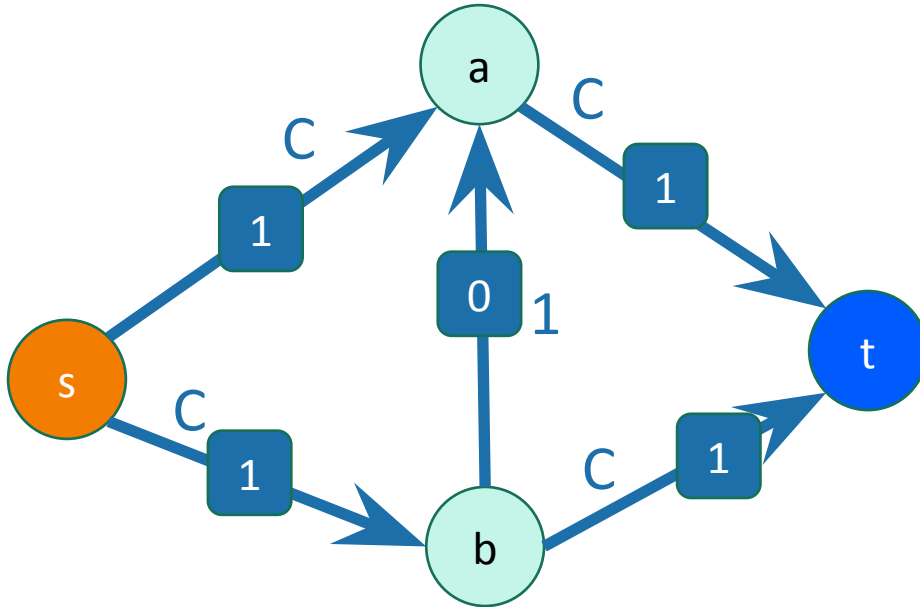
Choose a really big number C .



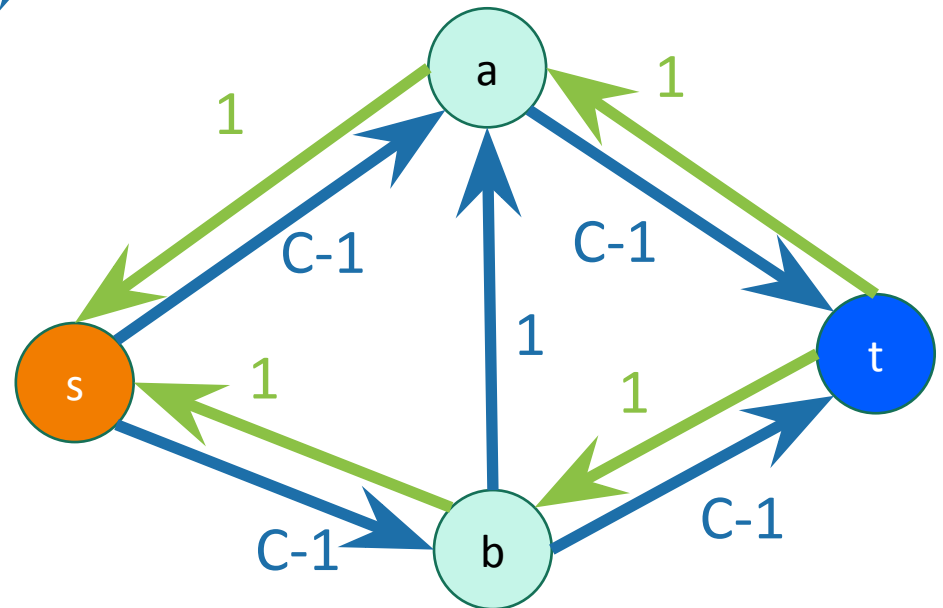
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C .



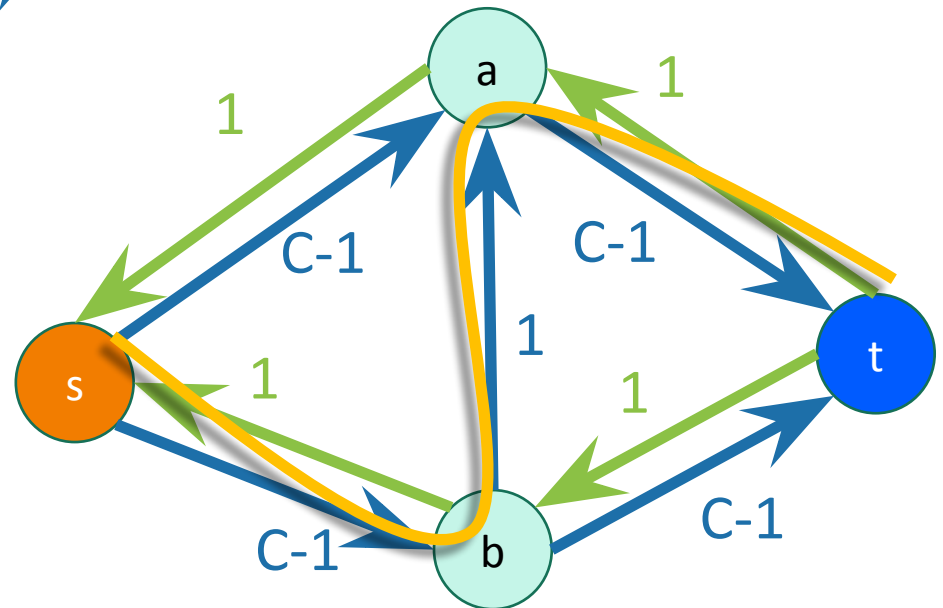
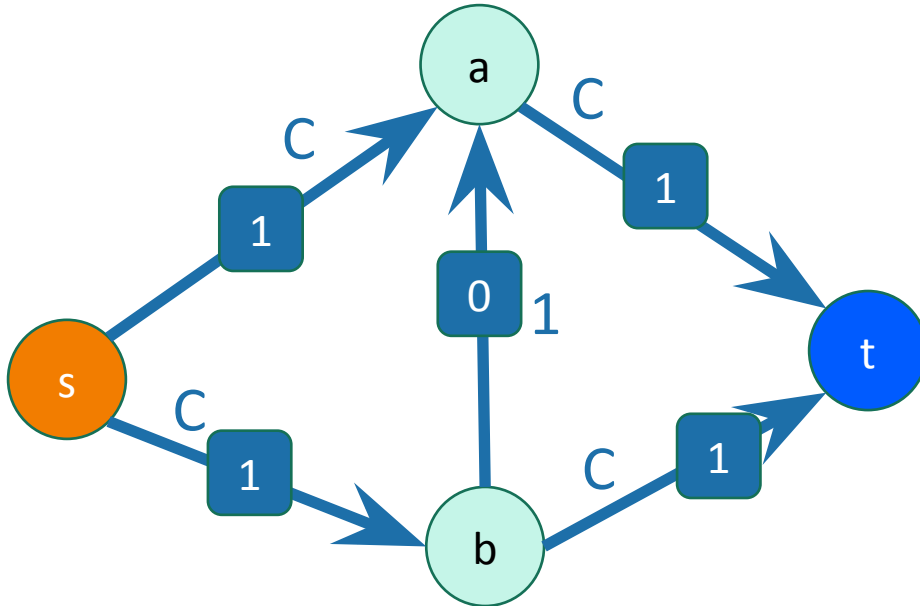
The edge (b,a) re-appeared in the residual graph!



Why should we be concerned?

Suppose we just picked paths arbitrarily.

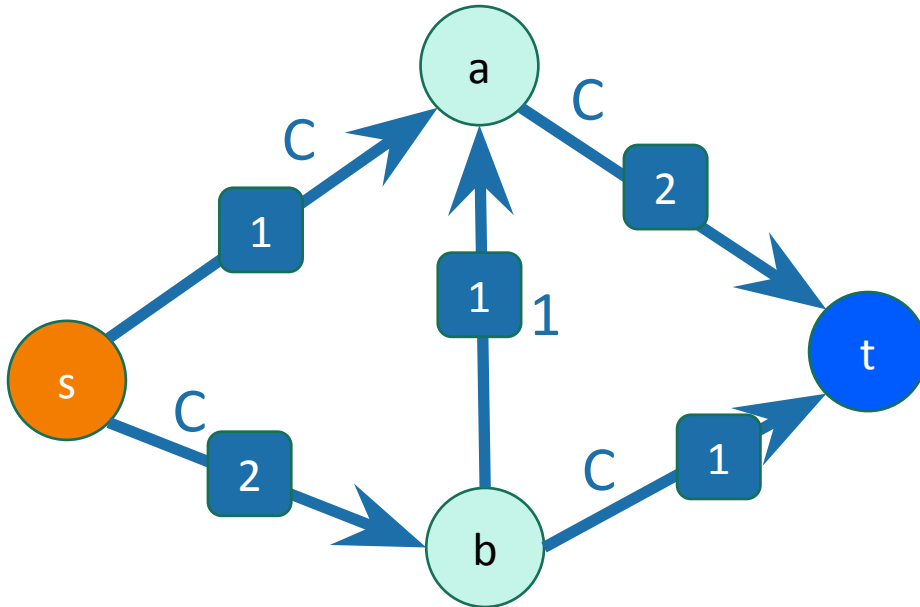
Choose a really big number C .



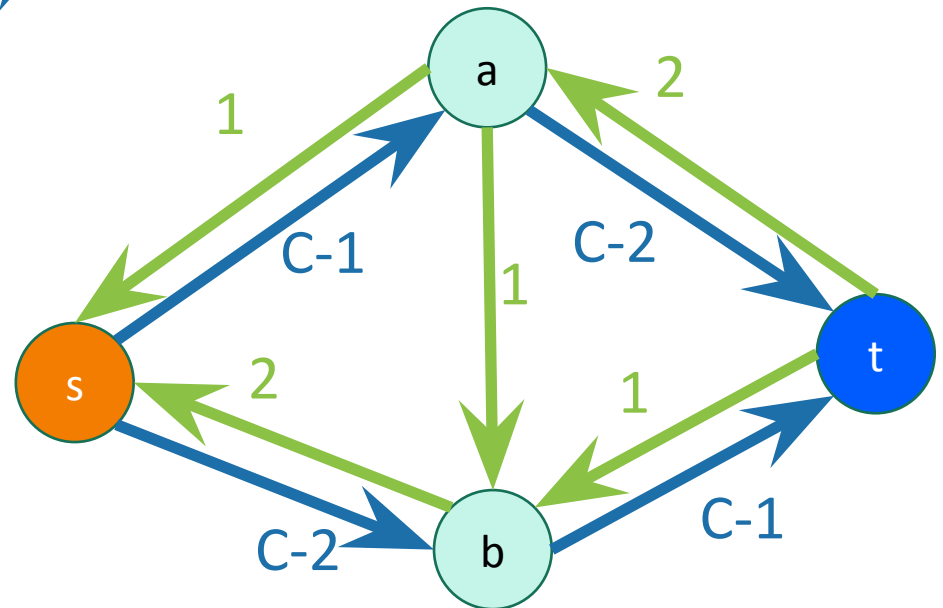
Why should we be concerned?

Suppose we just picked paths arbitrarily.

Choose a really big number C .



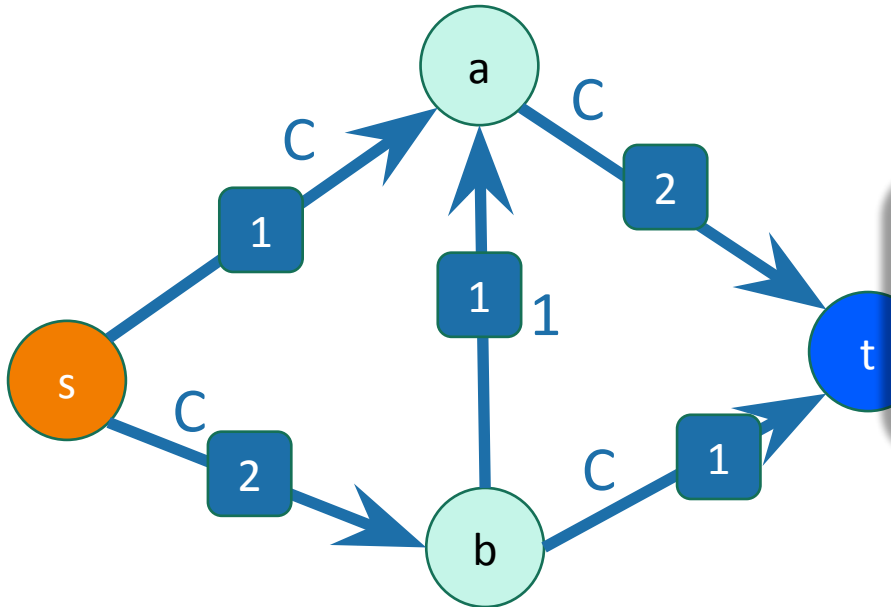
The edge (b,a) disappeared from the residual graph!



Why should we be concerned?

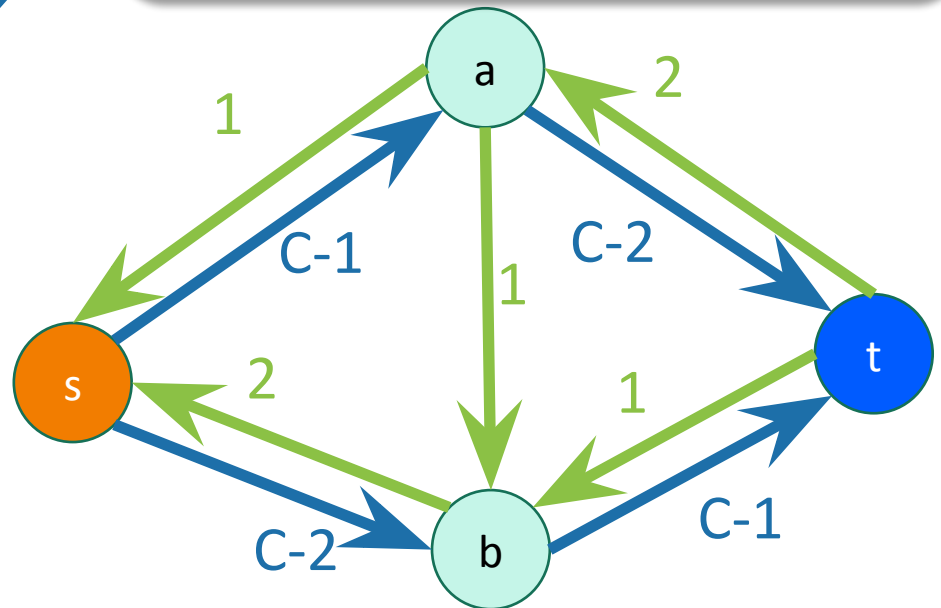
Suppose we just picked paths arbitrarily.

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.

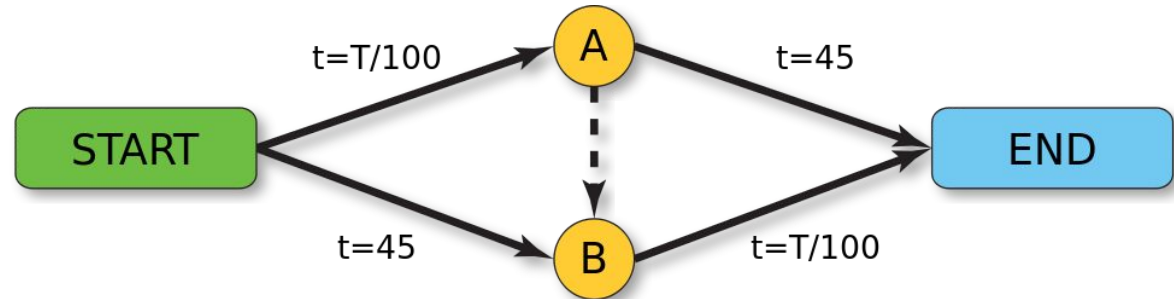
Network flow is complicated

Braess's paradox

From Wikipedia, the free encyclopedia

Braess's paradox is the observation that adding one or more roads to a road network can slow down overall traffic flow through it. The paradox was discovered by German mathematician [Dietrich Braess](#) in 1968.

(you're not responsible for this content, but it's good to know about)



Team sports strategy [\[edit\]](#)

It has been suggested that in basketball, a team can be seen as a network of possibilities for a route to scoring a basket, with a different efficiency for each pathway, and a star player could reduce the overall efficiency of the team, analogous to a shortcut that is overused increasing the overall times for a journey through a road network. A proposed solution for maximum efficiency in scoring is for a star player to shoot about the same number of shots as teammates. However, this approach is not supported by hard statistical evidence, as noted in the original paper.^[14]

In soccer [Helenio Herrera](#) is well known for his famous quote "with 10 [players] our team plays better than with 11".

Breaking news (thanks Ivan!)

NETWORKS

Researchers Achieve ‘Absurdly Fast’ Algorithm for Network Flow

 12 | 

Computer scientists can now solve a decades-old problem in practically the time it takes to write it down.

That’s where the inner workings of Spielman and Teng’s algorithm come in. Their algorithm provides a novel way to use a “low-stretch spanning tree” — a sort of internal backbone that captures many of the network’s most salient features. Given such a tree, there’s always at least one good cycle you can build by adding a single link from outside the tree. So having a low-stretch spanning tree drastically reduces the number of cycles you need to consider.

Even then, for the algorithm to run quickly, the team couldn’t afford to build a brand new spanning tree at every step. Instead, they had to ensure that each new cycle caused only minor ripple effects in the spanning trees, so they could reuse most of their previous computations. Achieving this level of control was “the core difficulty,” said [Yang Liu](#), a graduate student at Stanford University who is one of the paper’s authors.

Almost linear time!

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

8/3 Lecture Agenda

- Announcements
- Part 6-3: Max Flow
- 10 minute break!
- Part 6-4: Bipartite Matching

8/3 Lecture Agenda

- Announcements
- Part 6-3: Max Flow
- 10 minute break!
- Part 6-4: Bipartite Matching

WORLD 6-4

Bipartite Matching

Divide and Conquer

Sorting & Randomization

Data Structures

Graph Search

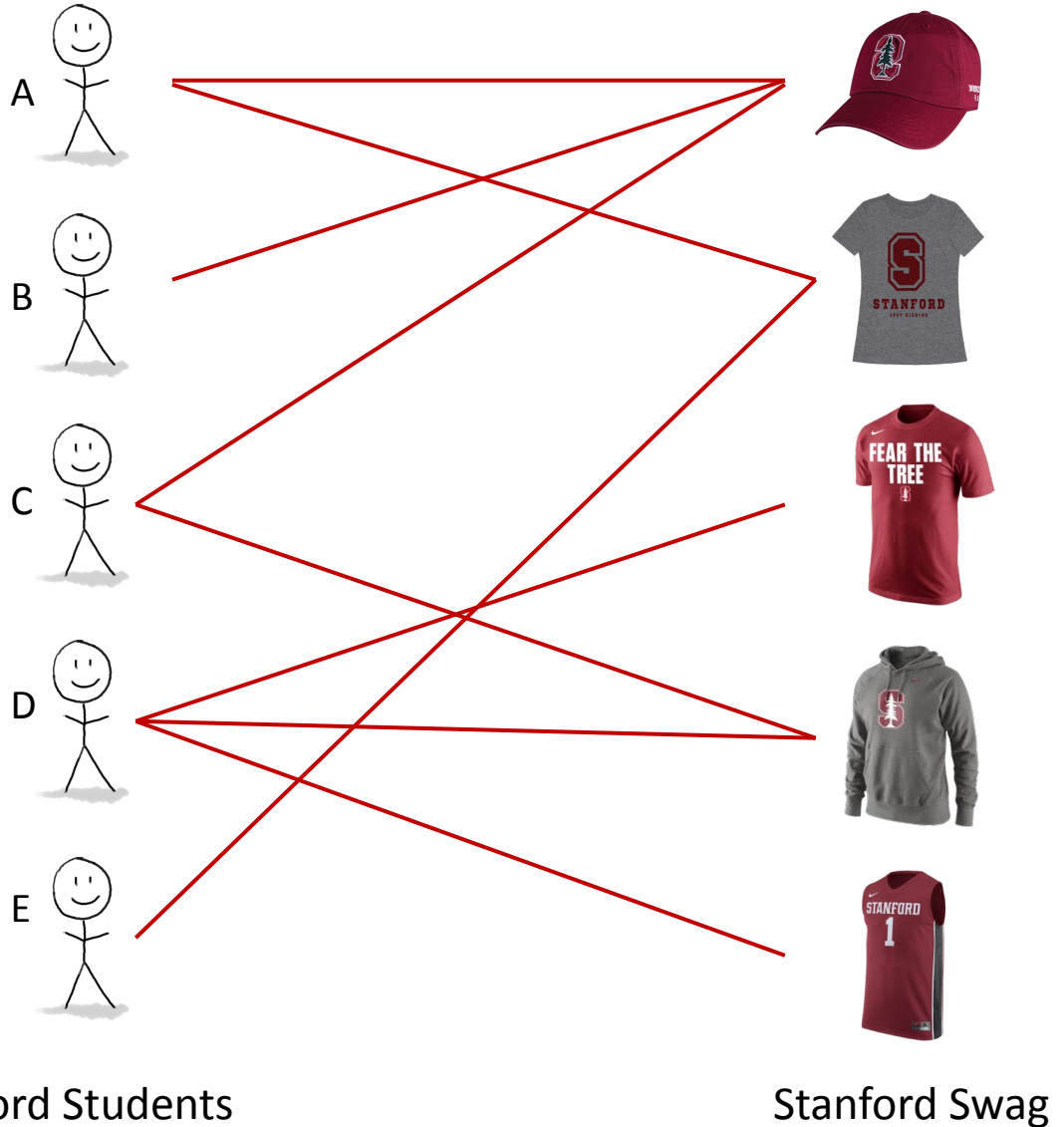
Dynamic Programming

Greed & Flow

Special Topics

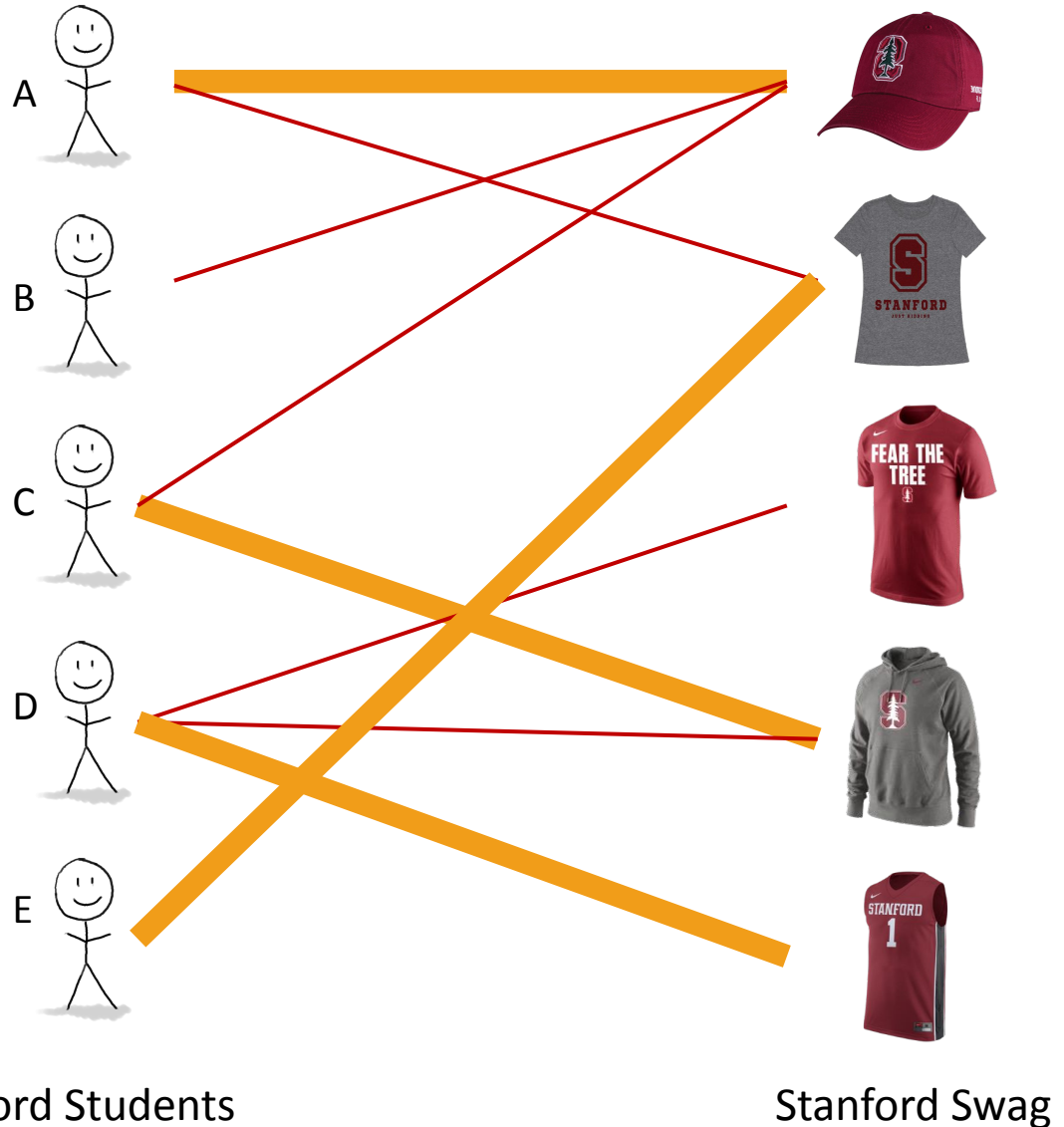
Maximum matching in bipartite graphs

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



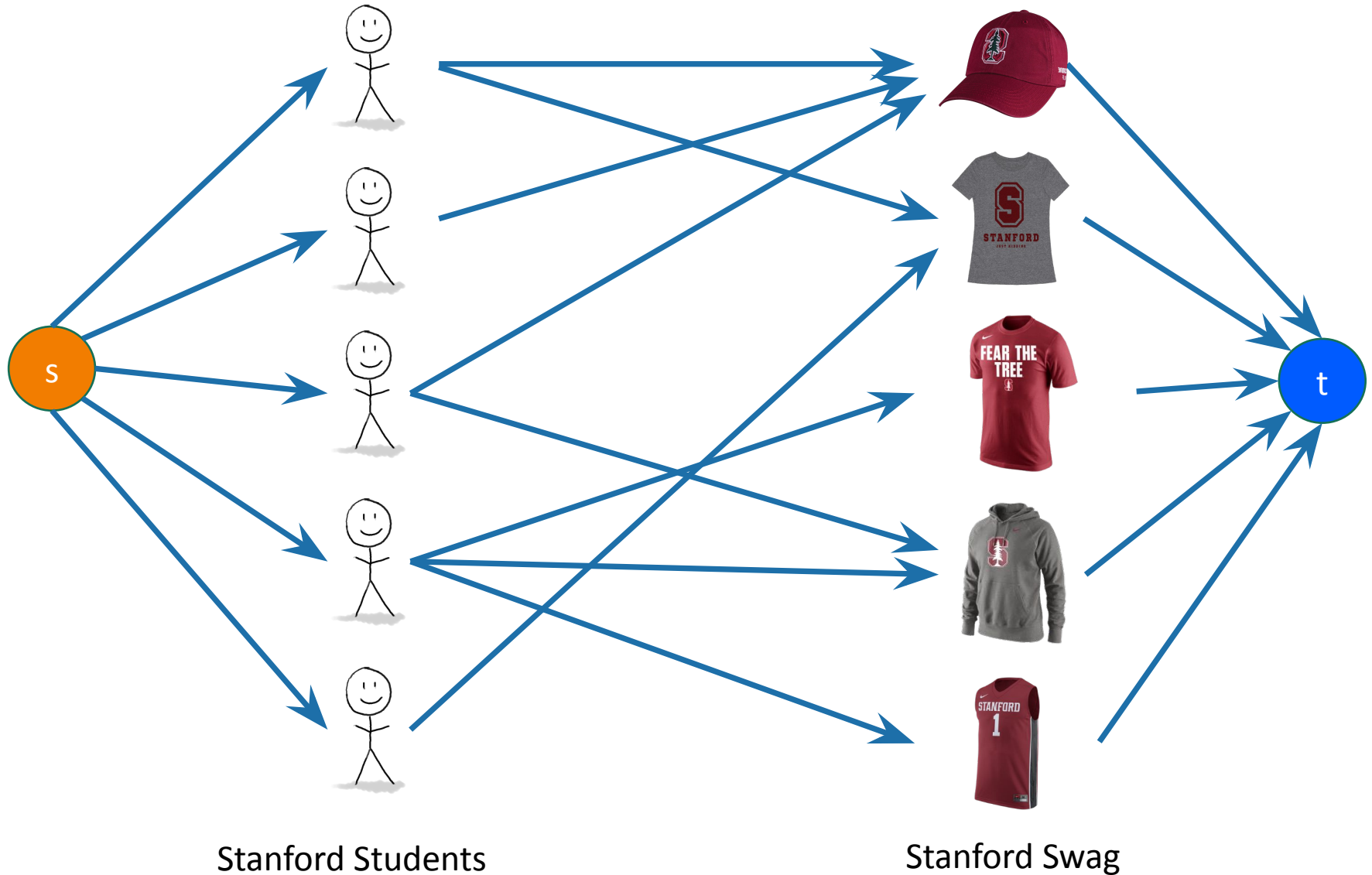
Maximum matching in bipartite graphs

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



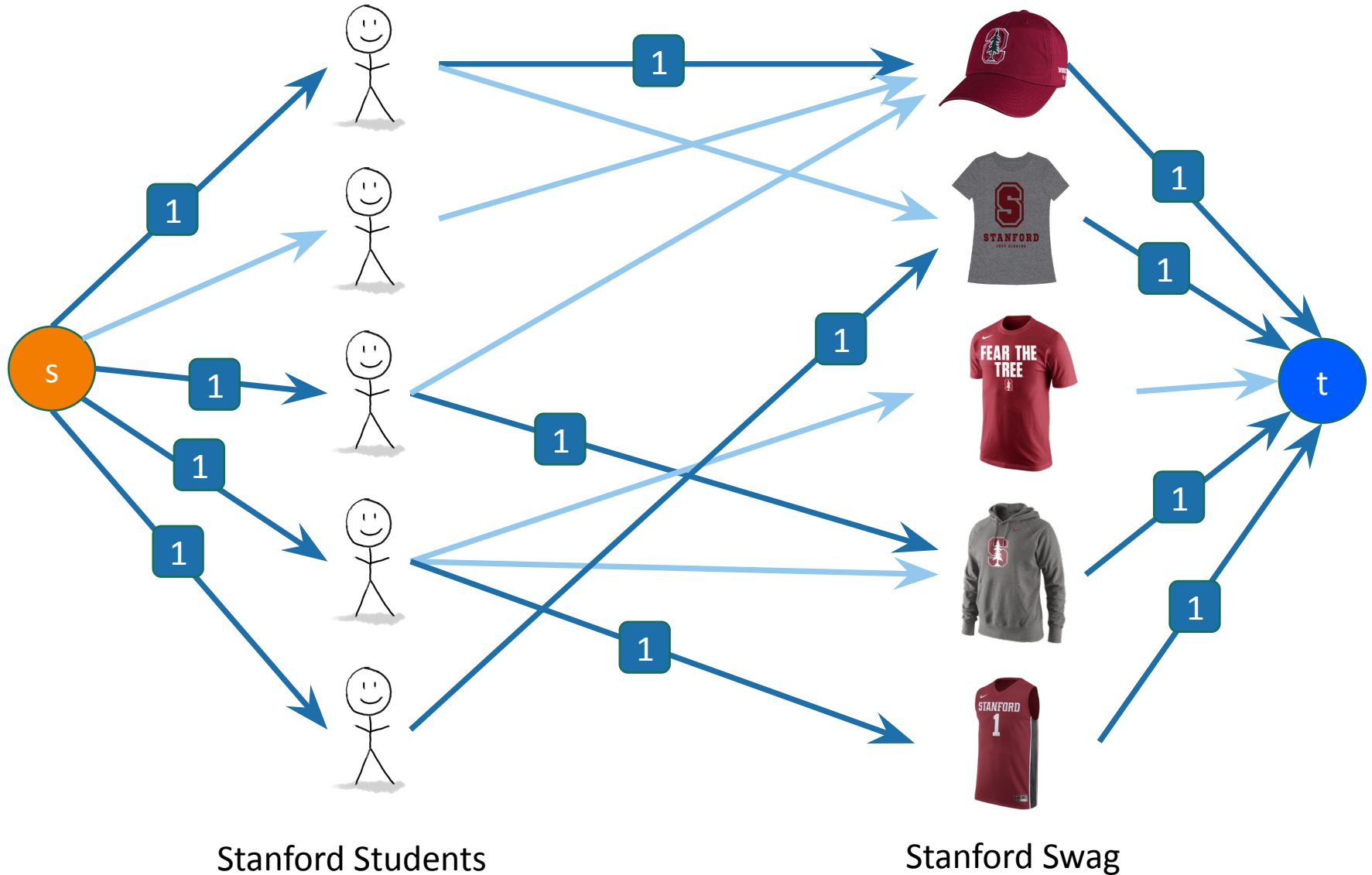
Solution via max flow

All edges have capacity 1.



Solution via max flow

All edges have capacity 1.

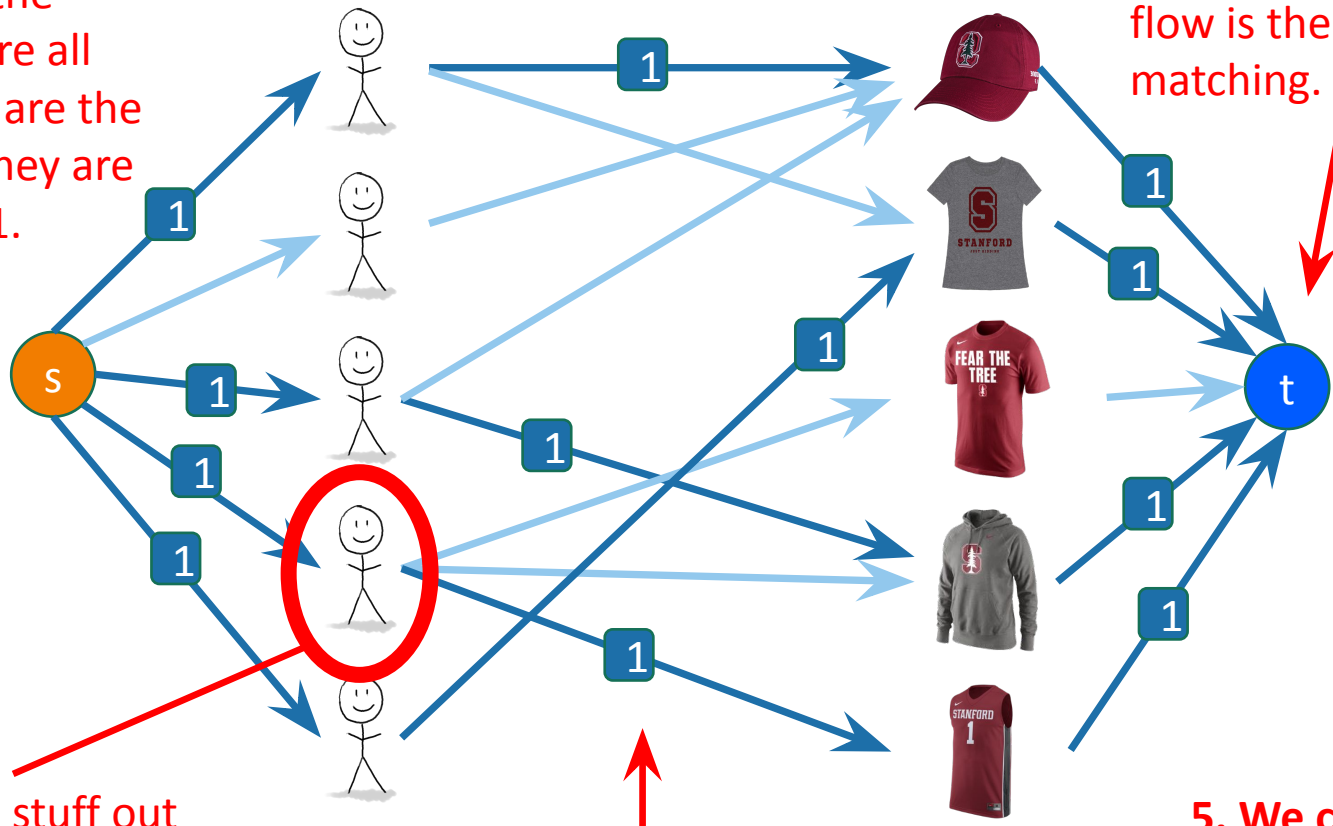


Solution via max flow

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.

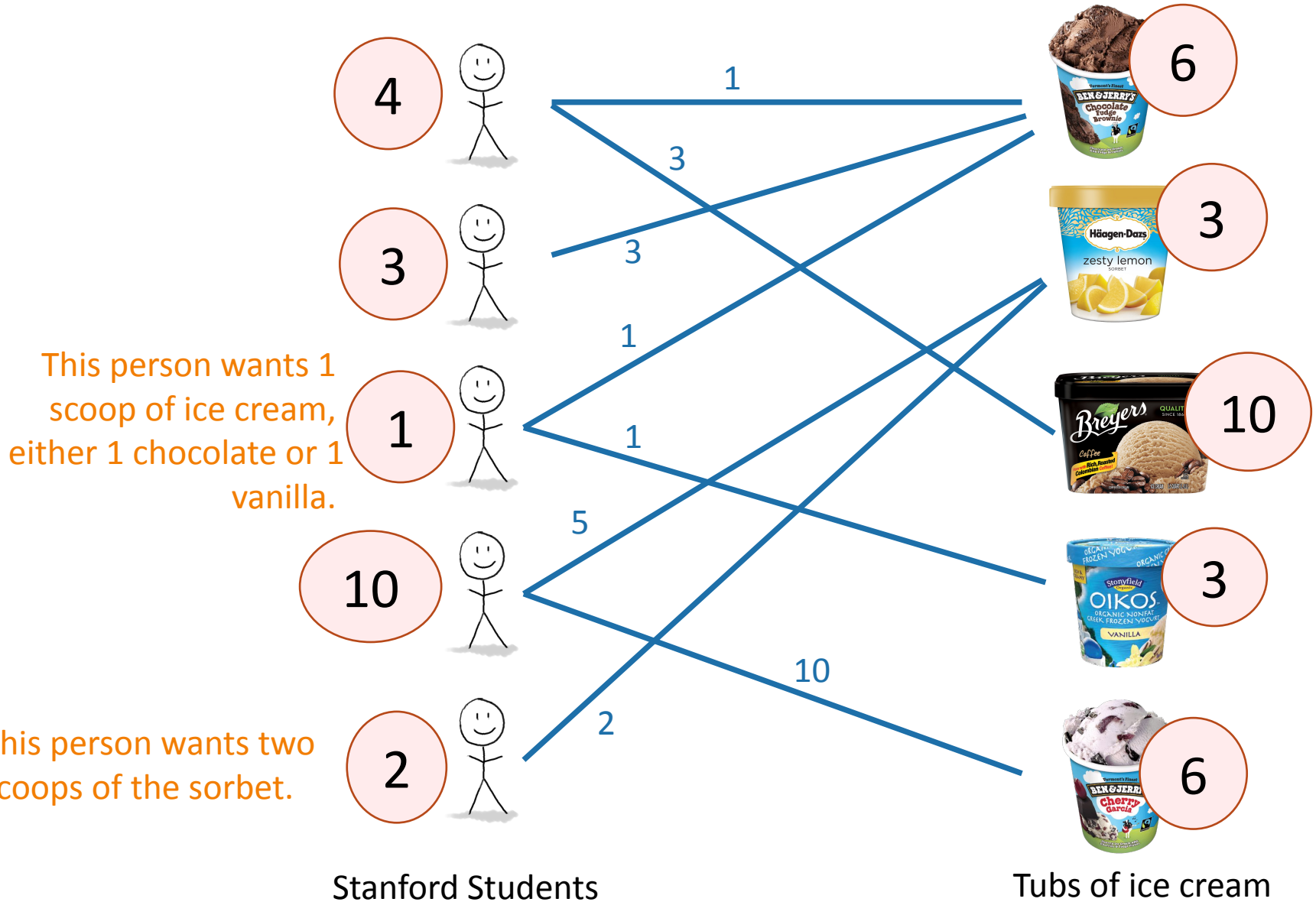
A slightly more complicated example: assignment problems



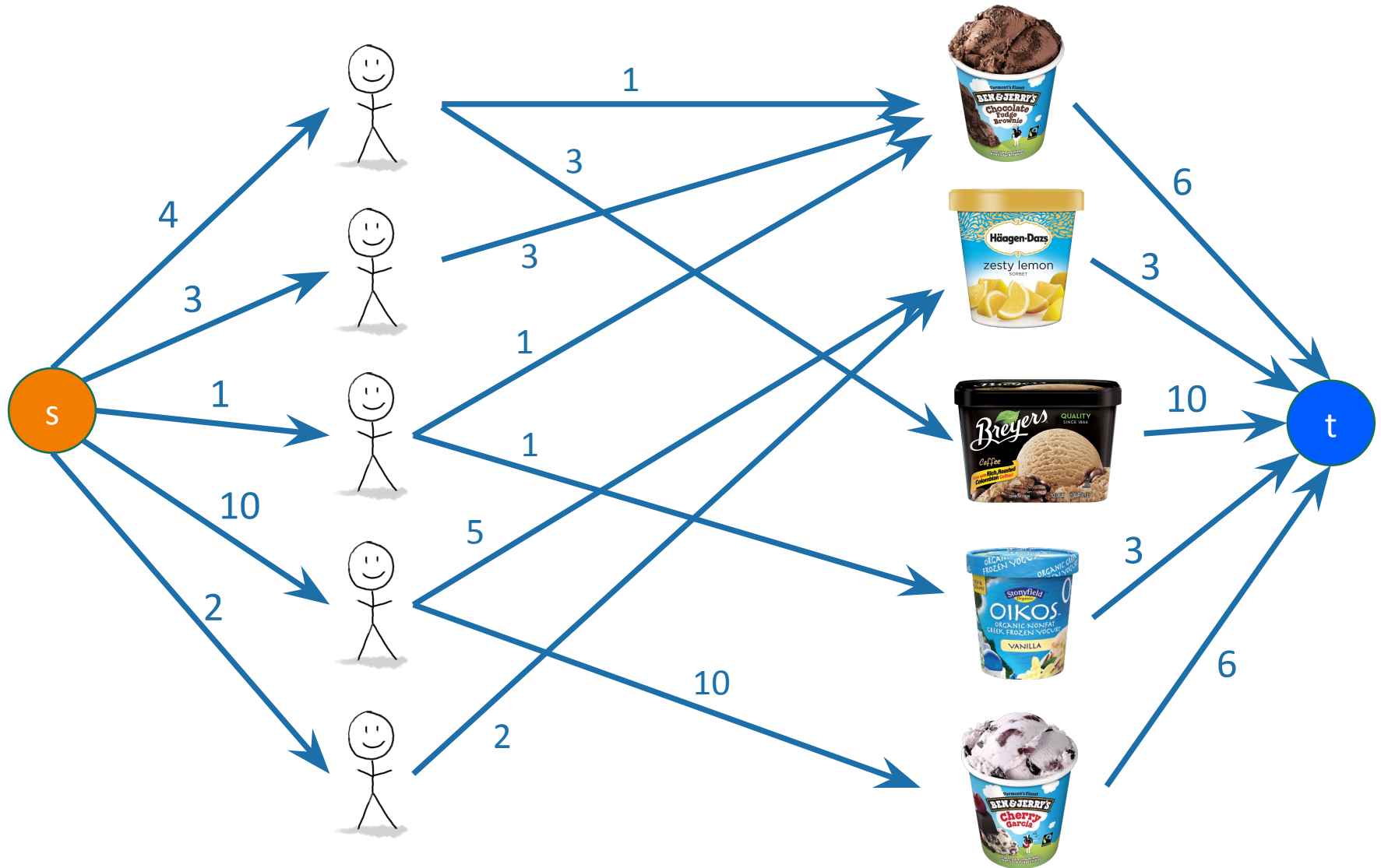
- One set X
 - Example: Stanford students
- Another set Y
 - Example: tubs of ice cream
- Each x in X can participate in $c(x)$ matches.
 - Student x can only eat 4 scoops of ice cream.
- Each y in Y can only participate in $c(y)$ matches.
 - Tub of ice cream y only has 10 scoops in it.
- Each pair (x,y)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

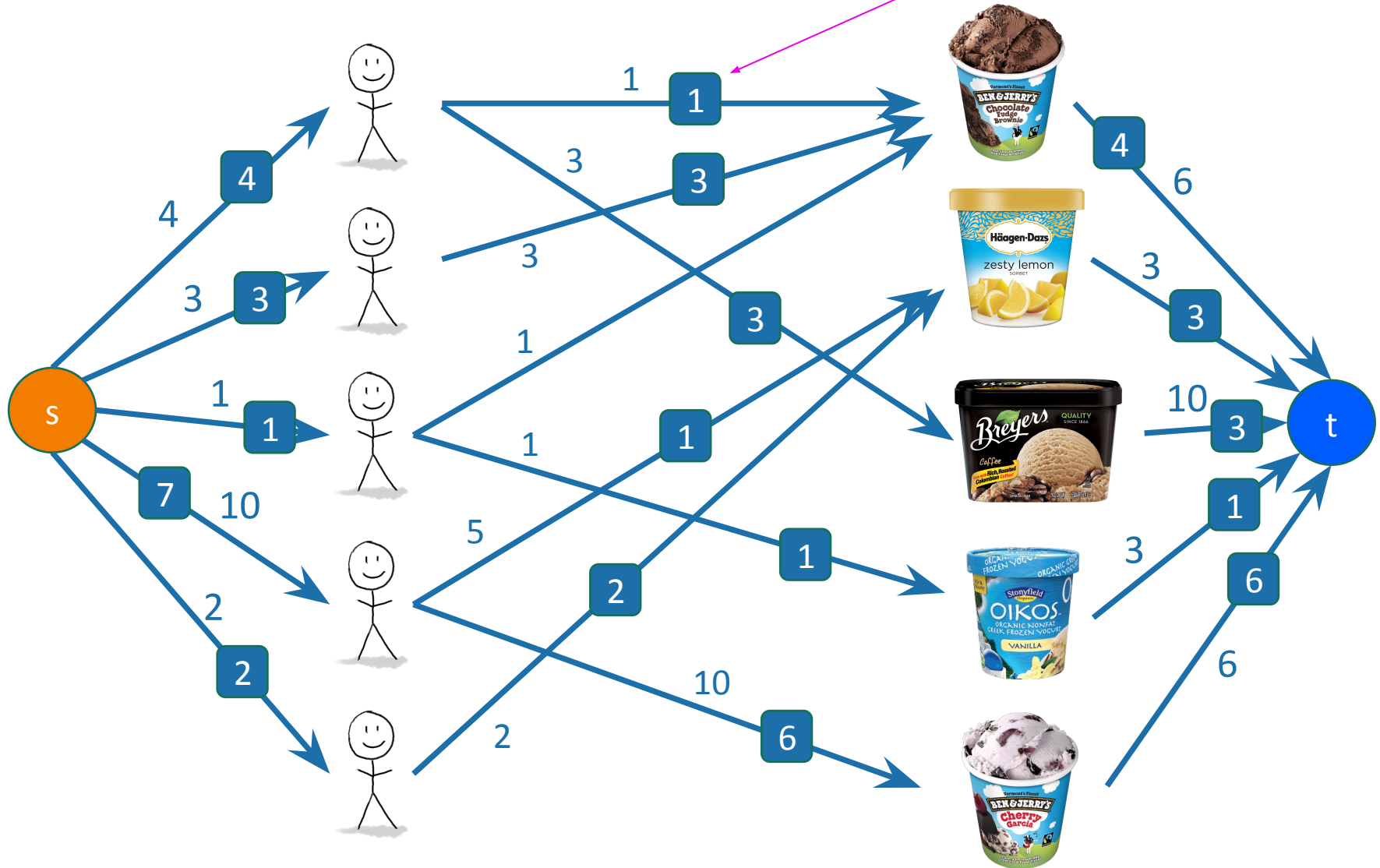


Stanford Students

Tubs of ice cream

Solution via max flow

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

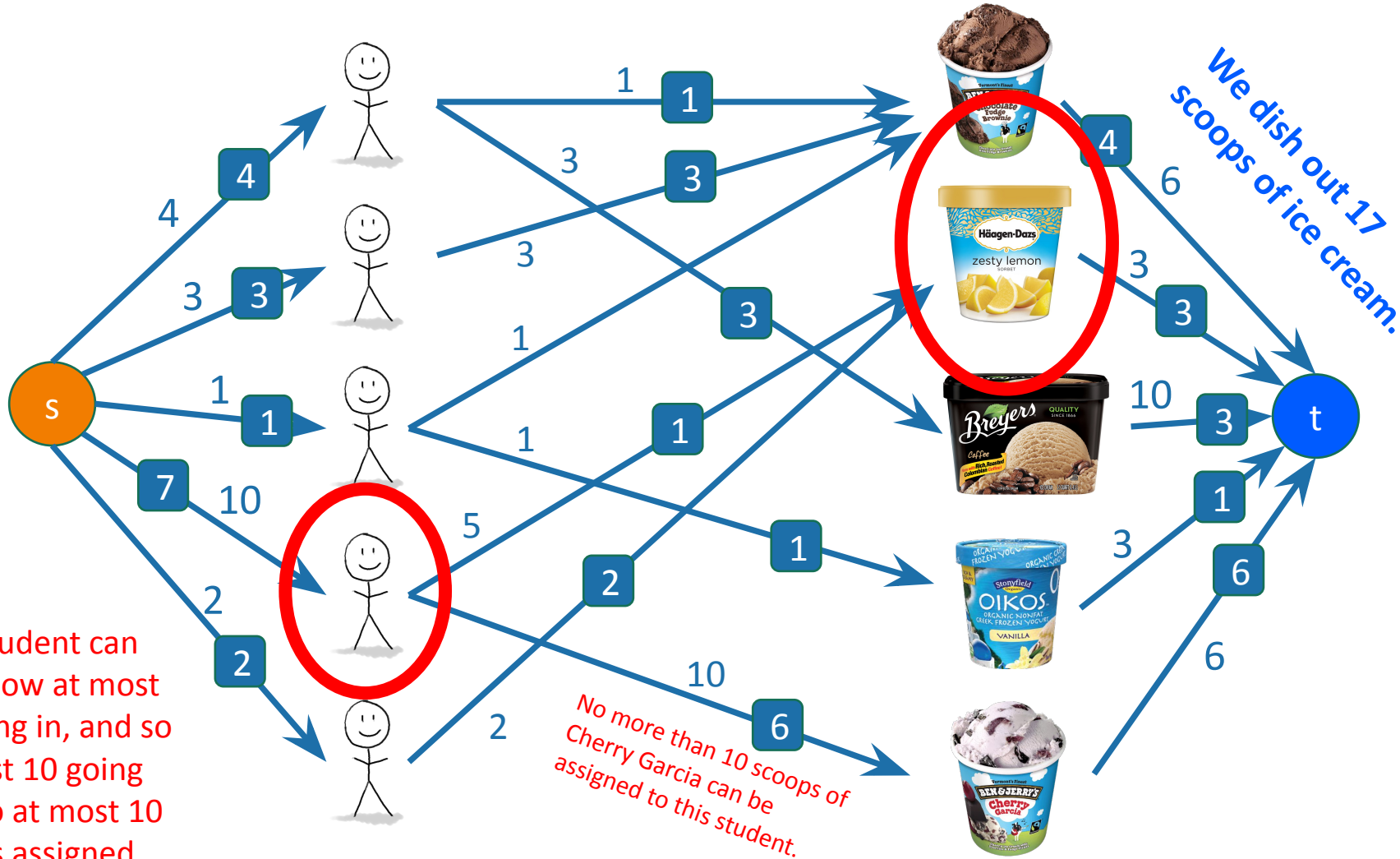


Stanford Students

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.

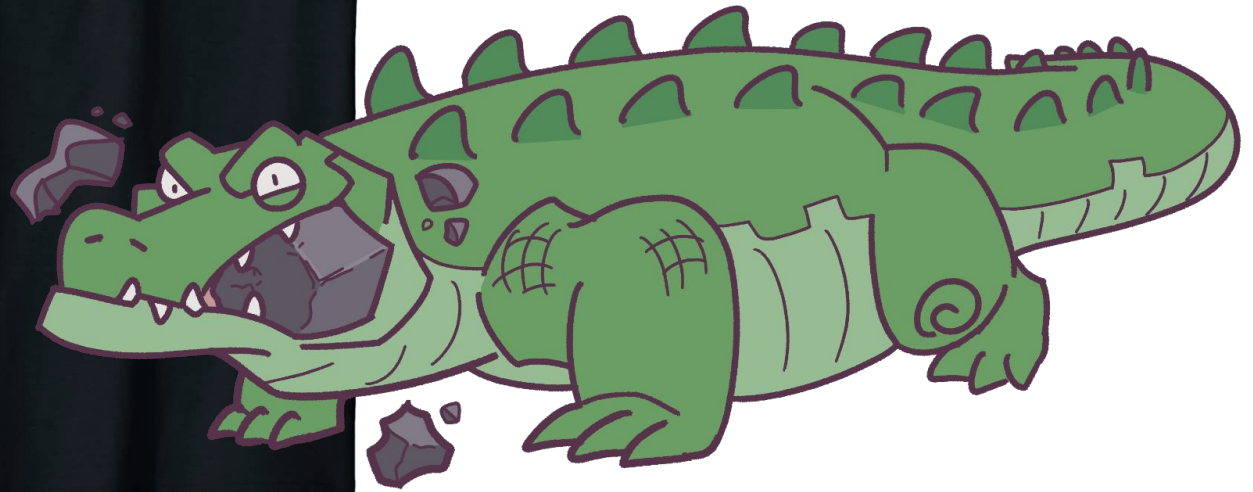
No more than 10 scoops of Cherry Garcia can be assigned to this student.

We dish out 17 scoops of ice cream.

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






**This technique can even solve problems
that aren't obviously flow/assignment
problems...**

WARNING








Can my team still top the standings?

NL West

Team	W	L	Pct	GB	Home	Away	L10
 Dodgers	70	33	.680	-	35-15	35-18	7-3
 Padres	60	46	.566	11.5	30-22	30-24	6-4
 Giants	51	53	.490	19.5	29-25	22-28	3-7
 Diamondbacks	46	57	.447	24.0	27-27	19-30	5-5
 Rockies	46	60	.434	25.5	30-27	16-33	3-7

Teams can "clinch" playoffs entry

Pacific			GP	W	L	OT	PTS	P%	R
1		Calgary	82	50	21	11	111	.677	4
2		Edmonton	82	49	27	6	104	.634	3
3		Los Angeles	82	44	27	11	99	.604	3
4		Vegas	82	43	31	8	94	.573	3
5		Vancouver	82	40	30	12	92	.561	3

Can my team still top the standings?

(We're assuming that the team with the most total wins is the overall winner of the sport, which is not how it usually works.)

This initially seems like a pretty easy greedy problem!

For the rest of the season, assume that:

- my team wins every game
- any team that is ahead of my team loses every game (at least until they are not ahead)

Can my team still top the standings?

(We're assuming that the team with the most total wins is the overall winner of the sport, which is not how it usually works.)

This initially seems like a pretty easy greedy problem!

For the rest of the season, assume that:

- my team wins every game
- any team that is ahead of my team loses every game (at least until they are not ahead)

What's wrong with this?

Can my team still make it?

This initially seems like a pretty easy greedy problem!

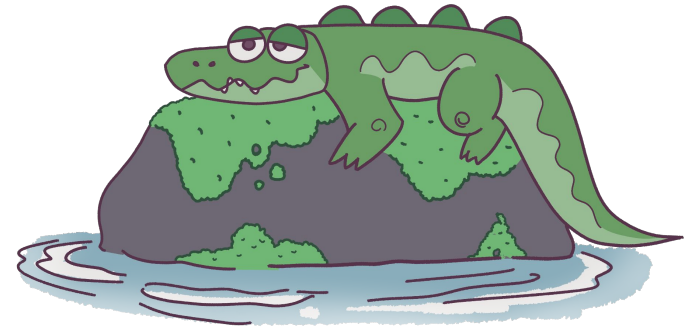
For the rest of the season, assume that:

- my team wins every game
- any team that is ahead of my team loses every game (at least until they are not ahead)

These teams might be playing each other! They can't both lose...

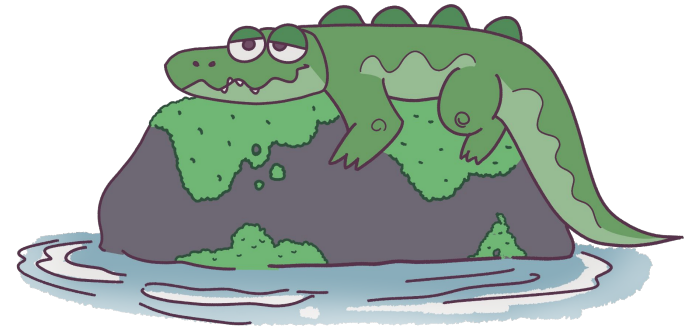
OK, fine

- when two teams that are ahead of my team play, the team that is *farthest* ahead of my team loses



OK, fine

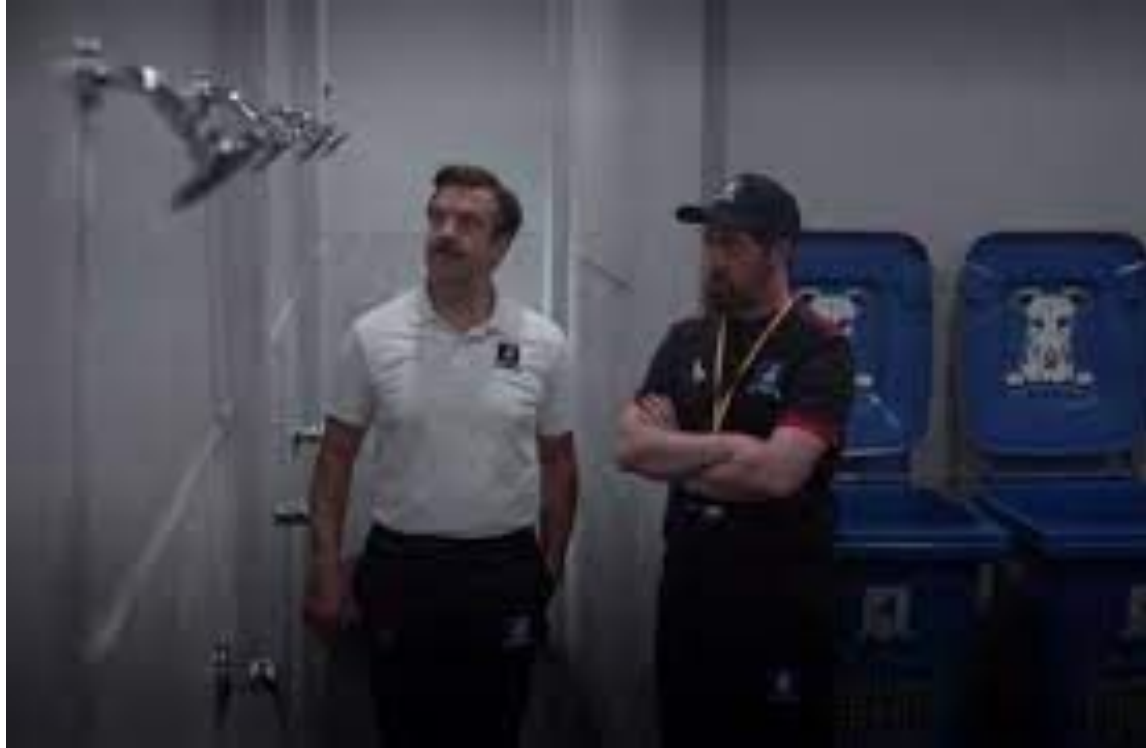
- when two teams that are ahead of my team play, the team that is *farthest* ahead of my team loses



What if they're tied and we make the wrong choice breaking the tie? and it bites us later?

Shouldn't it depend on the schedule of future games?

We've got ourselves a flow problem!



wait, how on earth would we use flow here?

This part is still true

For the rest of the season, assume that:

- our team wins every game

Why? Consider any solution in which our team loses a future game, but still wins overall. But then we could have our team win that game instead. It only moves us *up* in the standings and some other team *down*, so we still win overall.

More greedy stuff

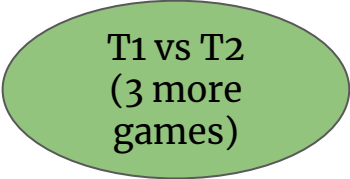
- Every other team either has fewer wins than us (a "nonthreat"), or doesn't (a "threat")
- Why call them nonthreats? They can never pass us if we keep winning.
 - this assumes all teams have the same number of remaining games...
- In future games:
 - **Threat-Nonthreat:** Make nonthreat win.
 - **Nonthreat-Nonthreat:** Doesn't matter.
 - **Threat-Threat:** Here we have to be careful...

Threat-threat games

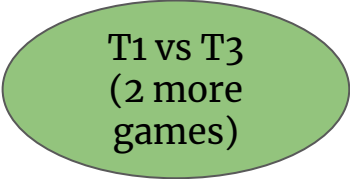
- For each threat team: suppose for now that they win all their threat-threat games, and that gets them a total of W_T wins.
- We already know our best-case number of wins W_U .
- We need to make sure that threat team loses at least $(W_T - W_U + 1)$ threat-threat games, or they will be ahead of (or tied with) us at the end...

Toward a flow solution...

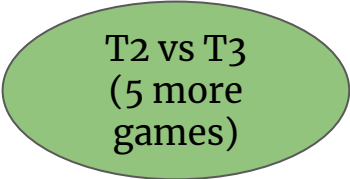
- For each *pair* of threat teams, we care about how many games they have remaining.
- Each such game can give one of those teams one loss.



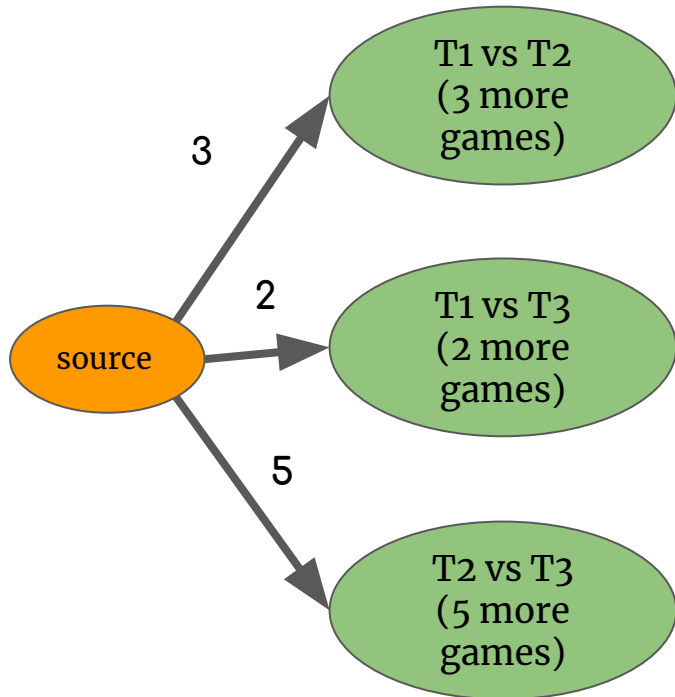
T1 vs T2
(3 more
games)



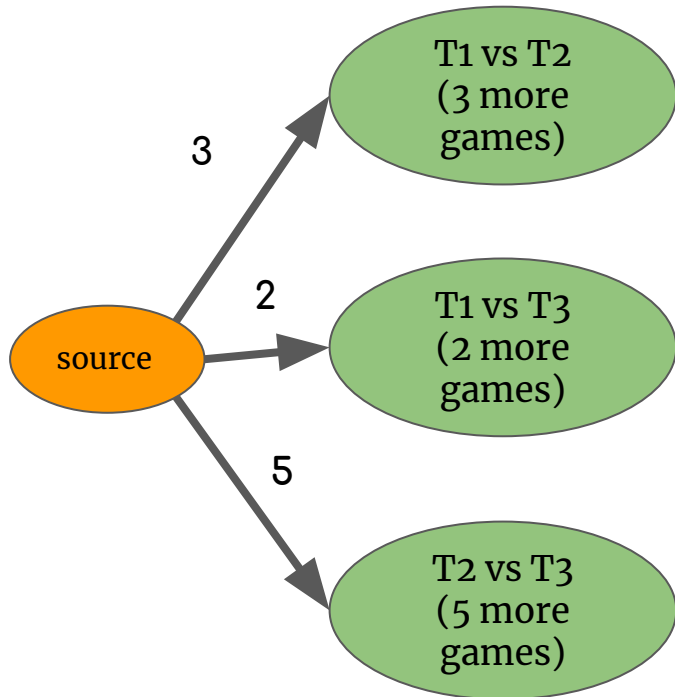
T1 vs T3
(2 more
games)



T2 vs T3
(5 more
games)

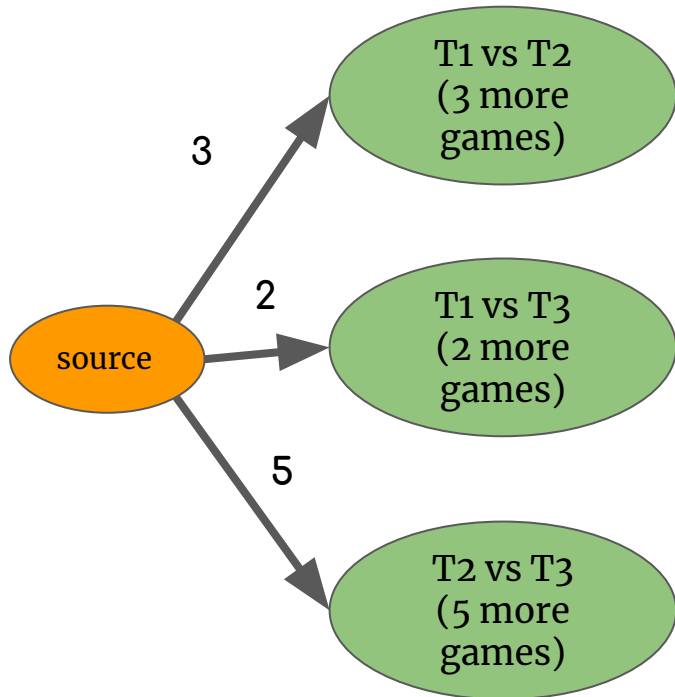


*Our graph will
push losses
around.*



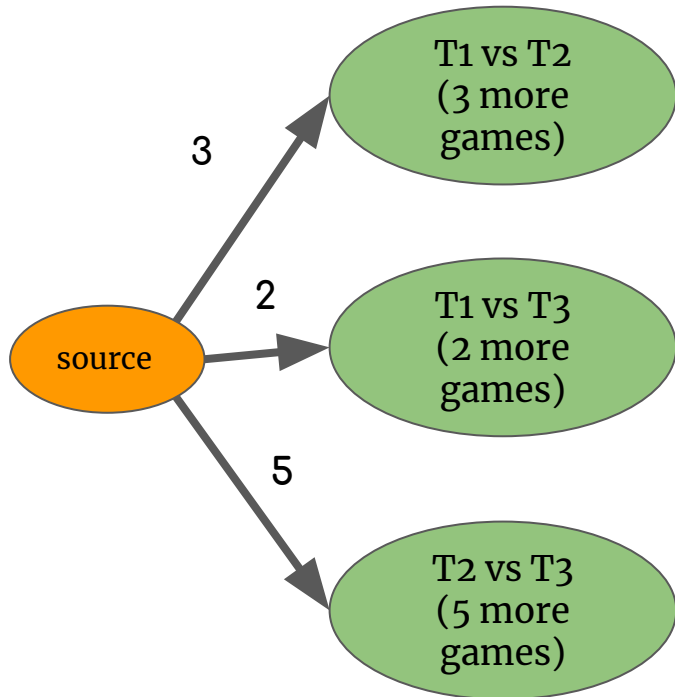
Our graph will push losses around.





Our graph will push losses around.

why these numbers? we can't give out more losses (from games of one type) than there are games of that type.

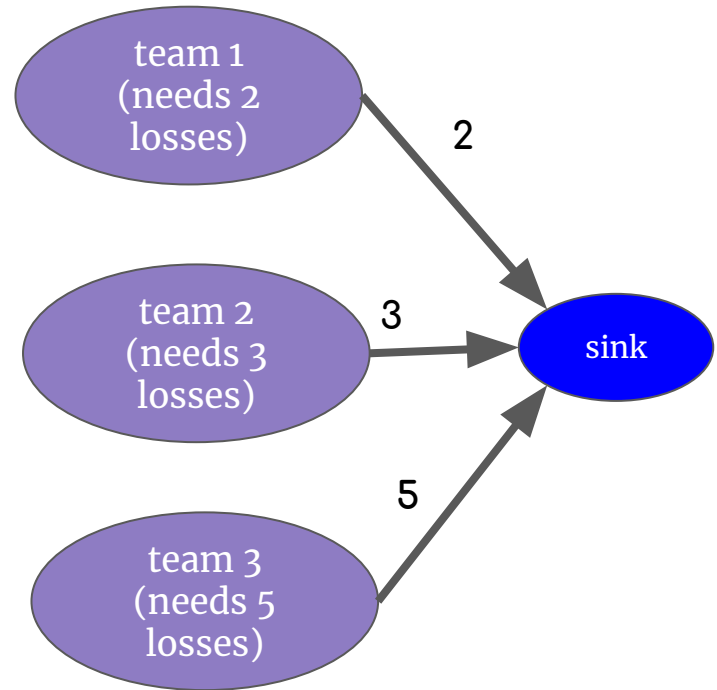
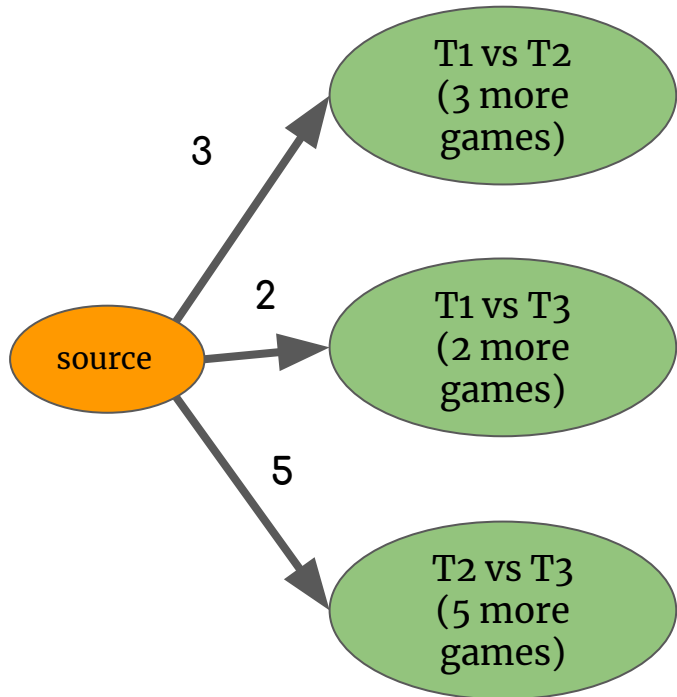


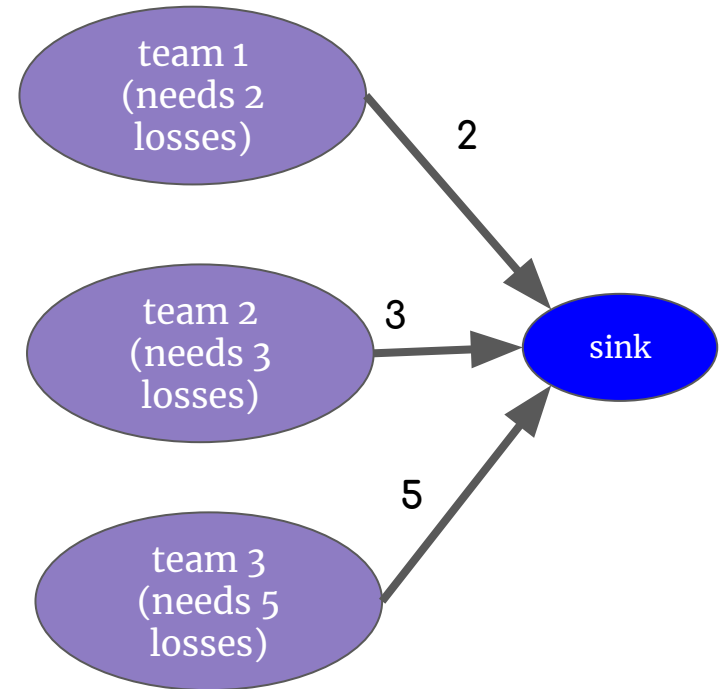
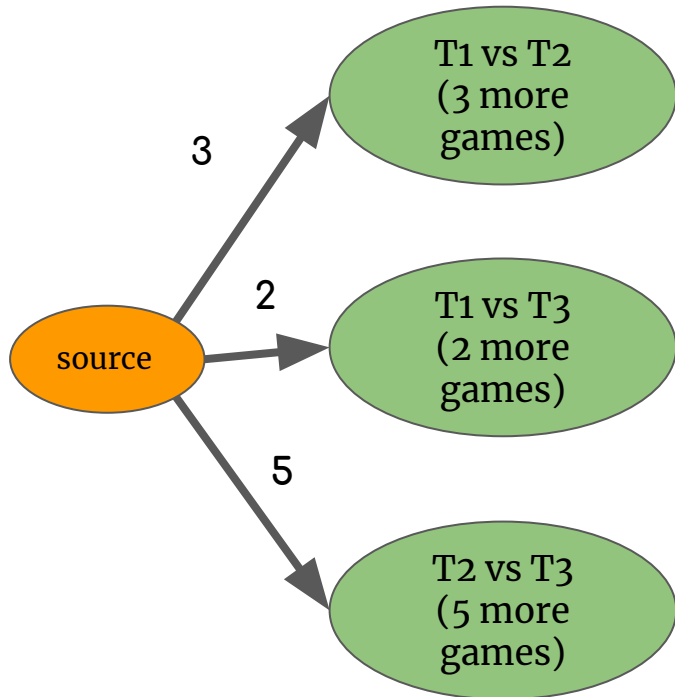
team 1
(needs 2
losses)

team 2
(needs 3
losses)

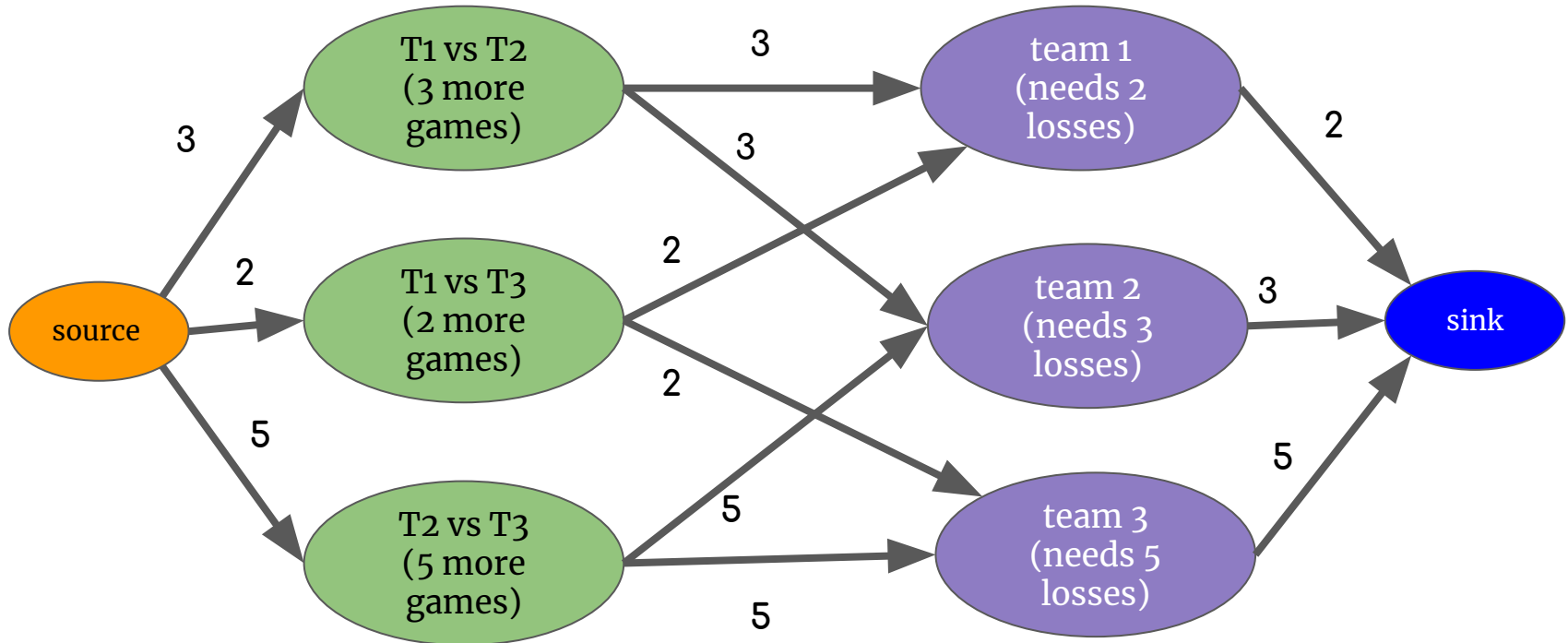
team 3
(needs 5
losses)

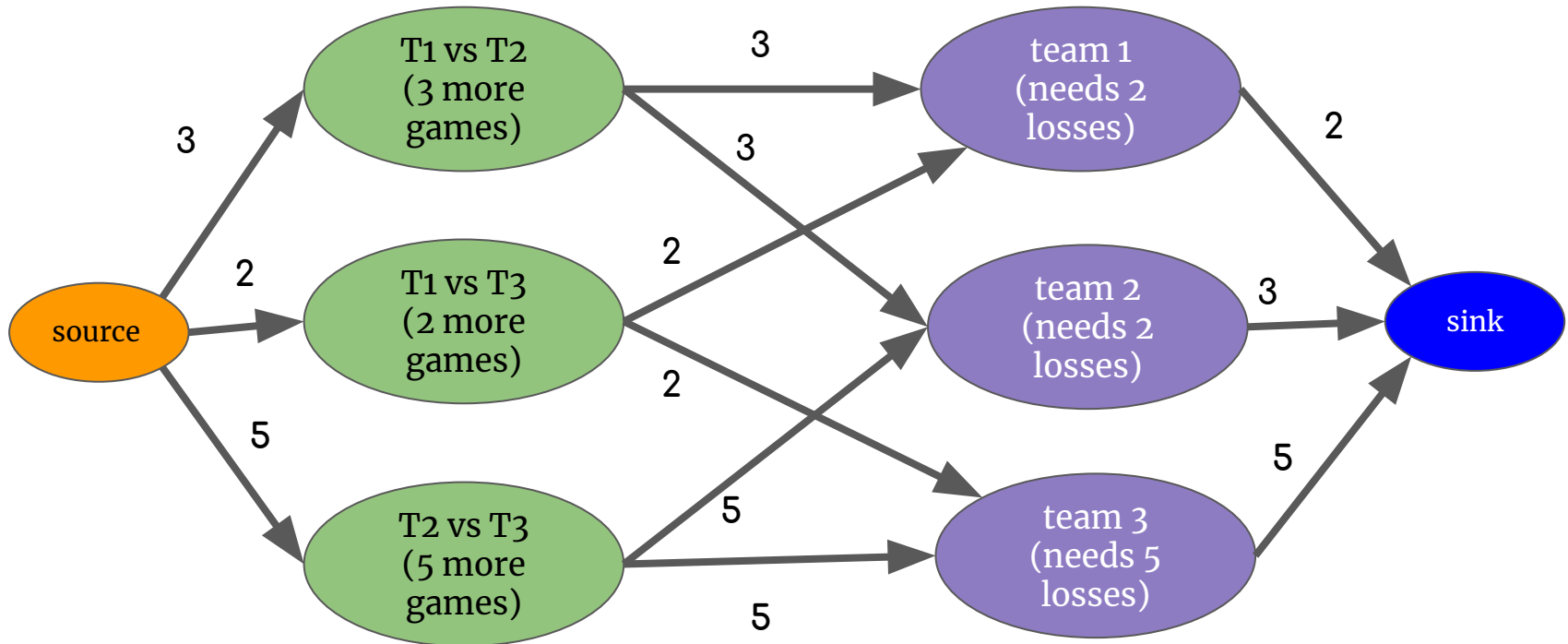
*Other side of the
bipartition:
teams!*





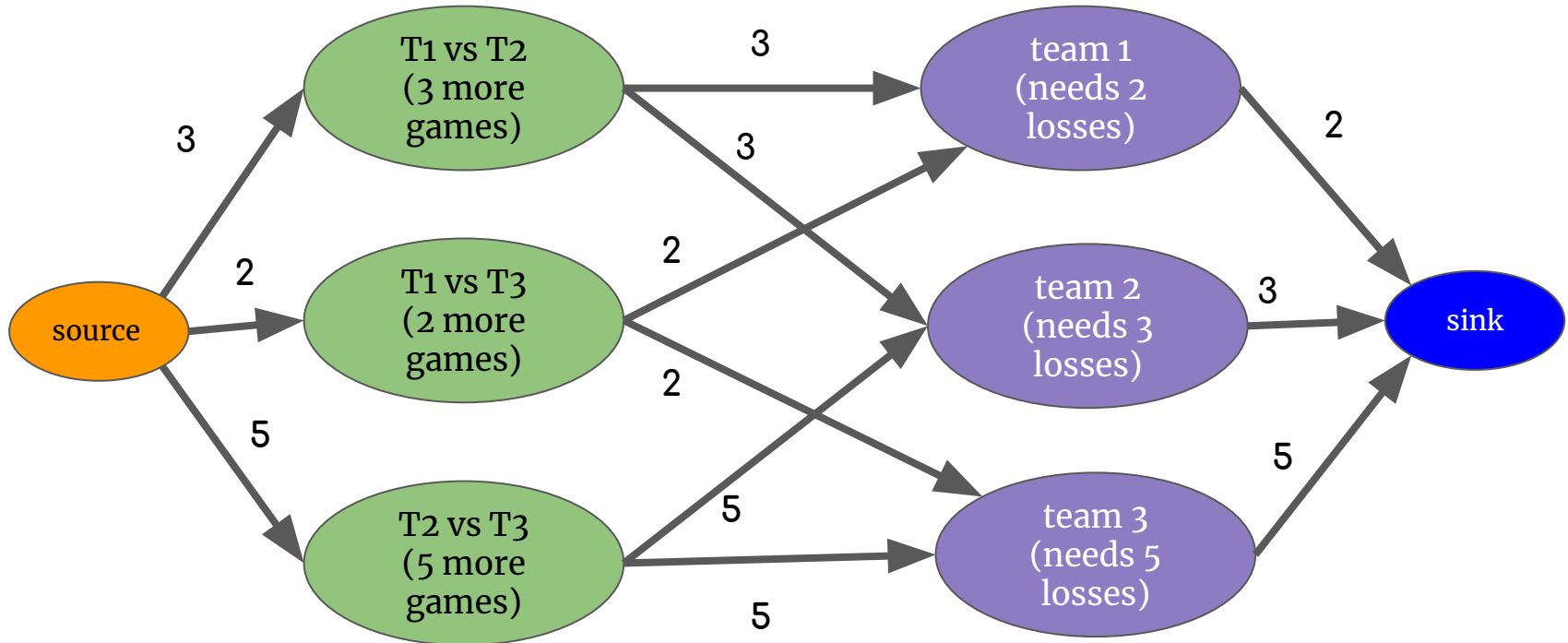
why these numbers? we get no credit for additional losses beyond what we need



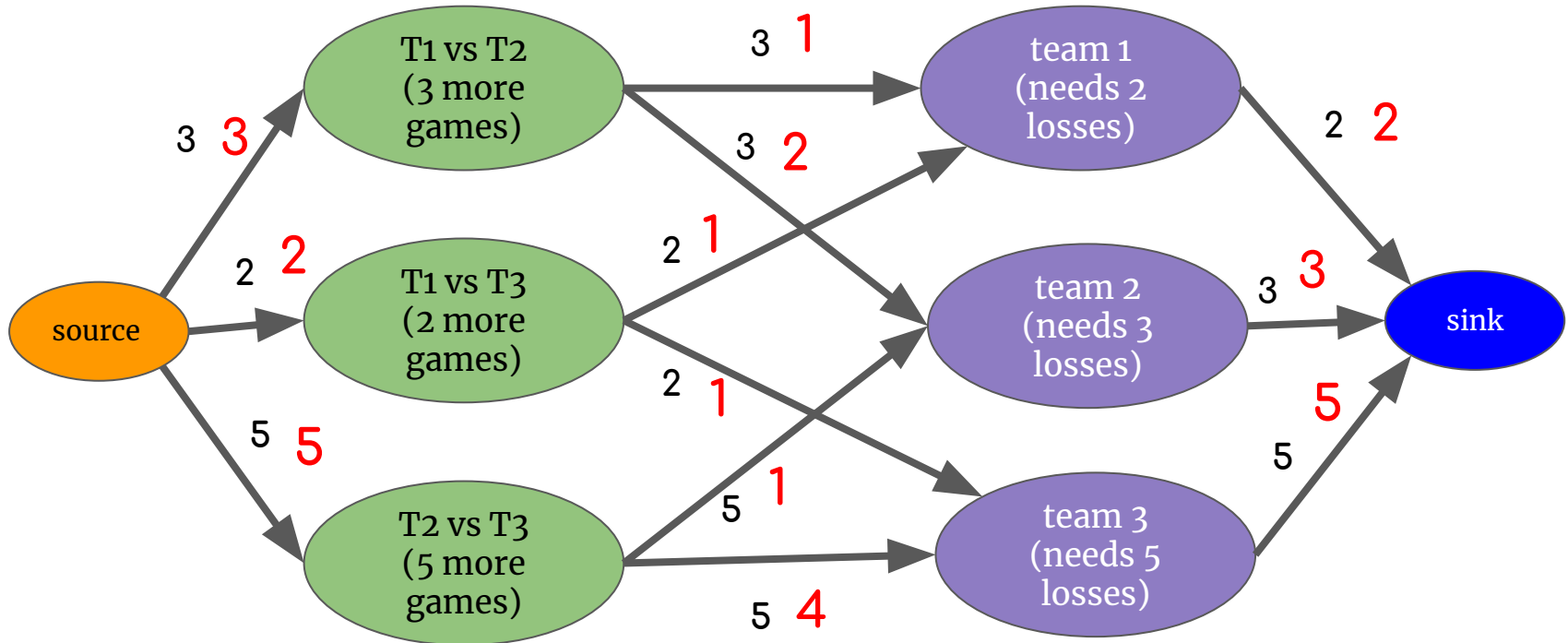


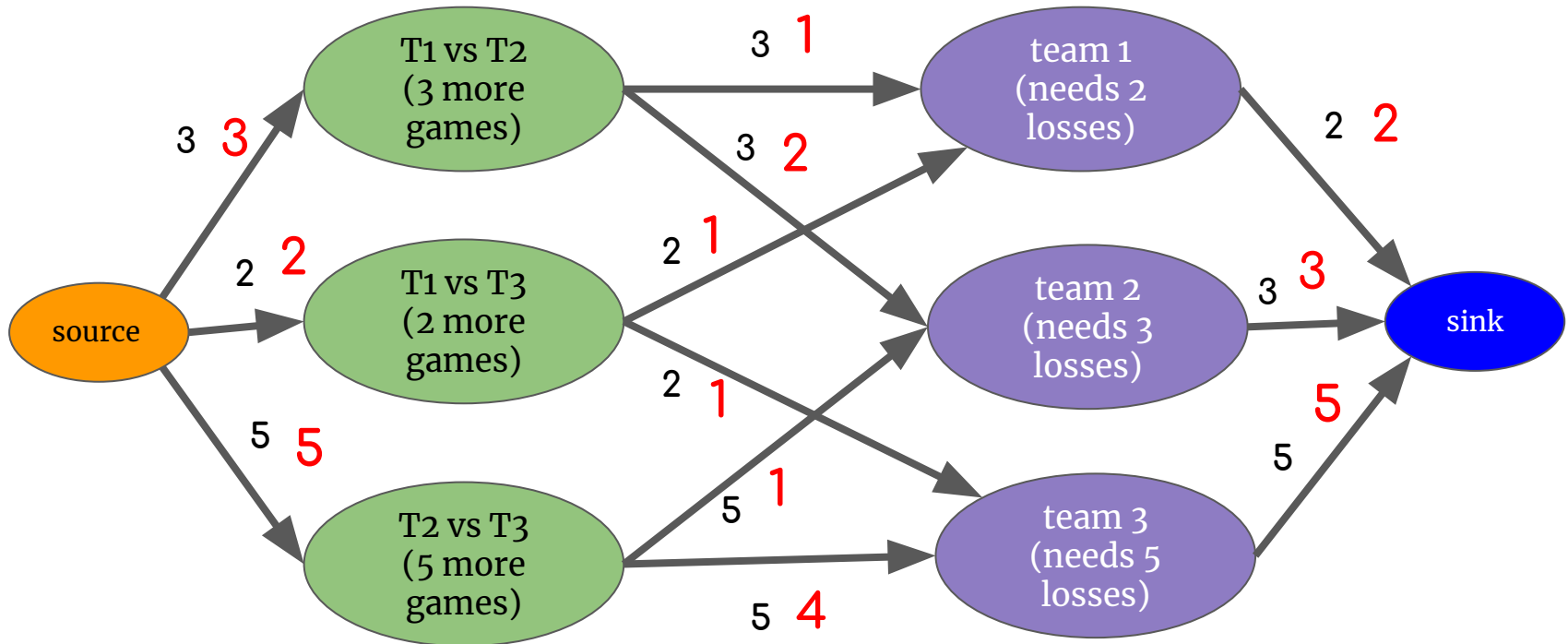
why these numbers? teams can't lose more games of a certain type than there are...

Then solve!

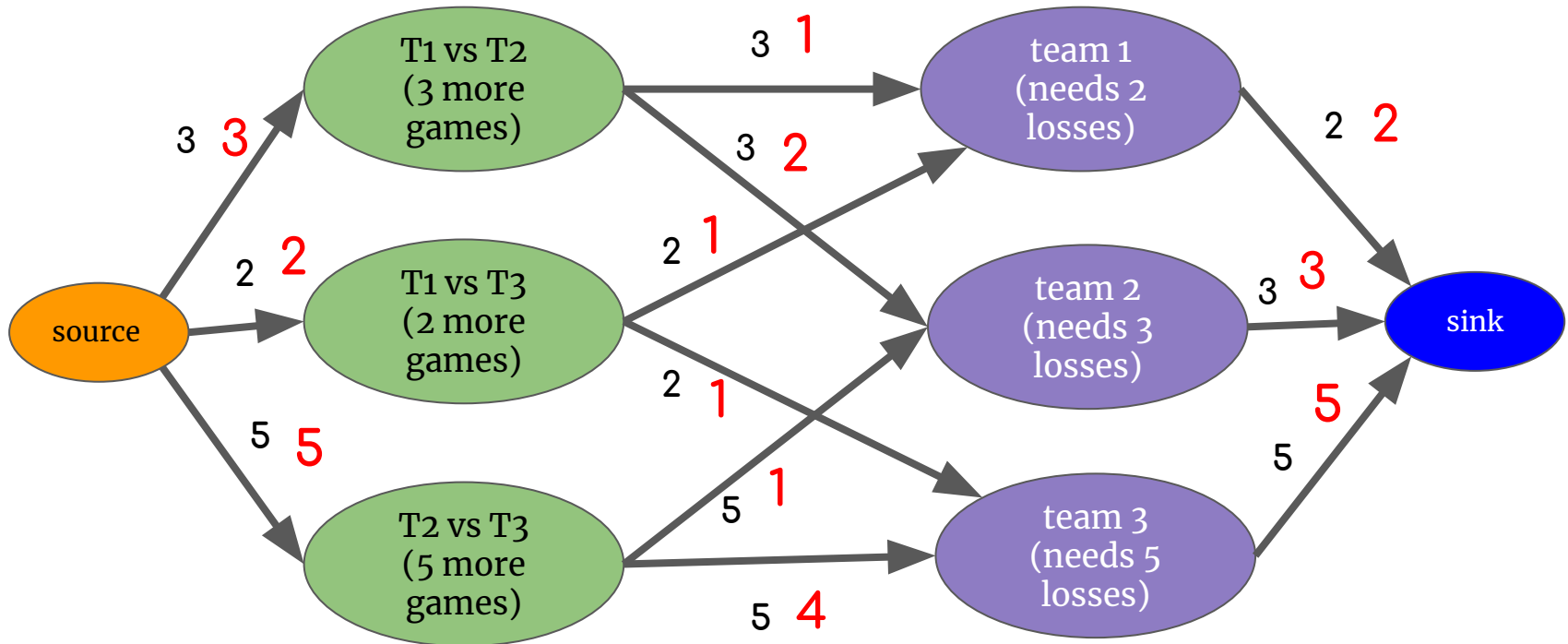


Then solve!





If we can fully saturate the sink, we win. Otherwise we don't.



Warning: this side of the graph actually has $O(\text{teams}^2)$ nodes!

But who cares? This is awesome