Lec19: Advanced Algorithms

Algorithm I
COMP319-003
Spring 2023

Instructor: Jiyeon Lee
School of Computer Science and Engineering
Kyungpook National University **(KNU)**

Last time

- Two algorithms for Minimum Spanning Tree
  - Prim's algorithm
  - Kruskal's algorithm

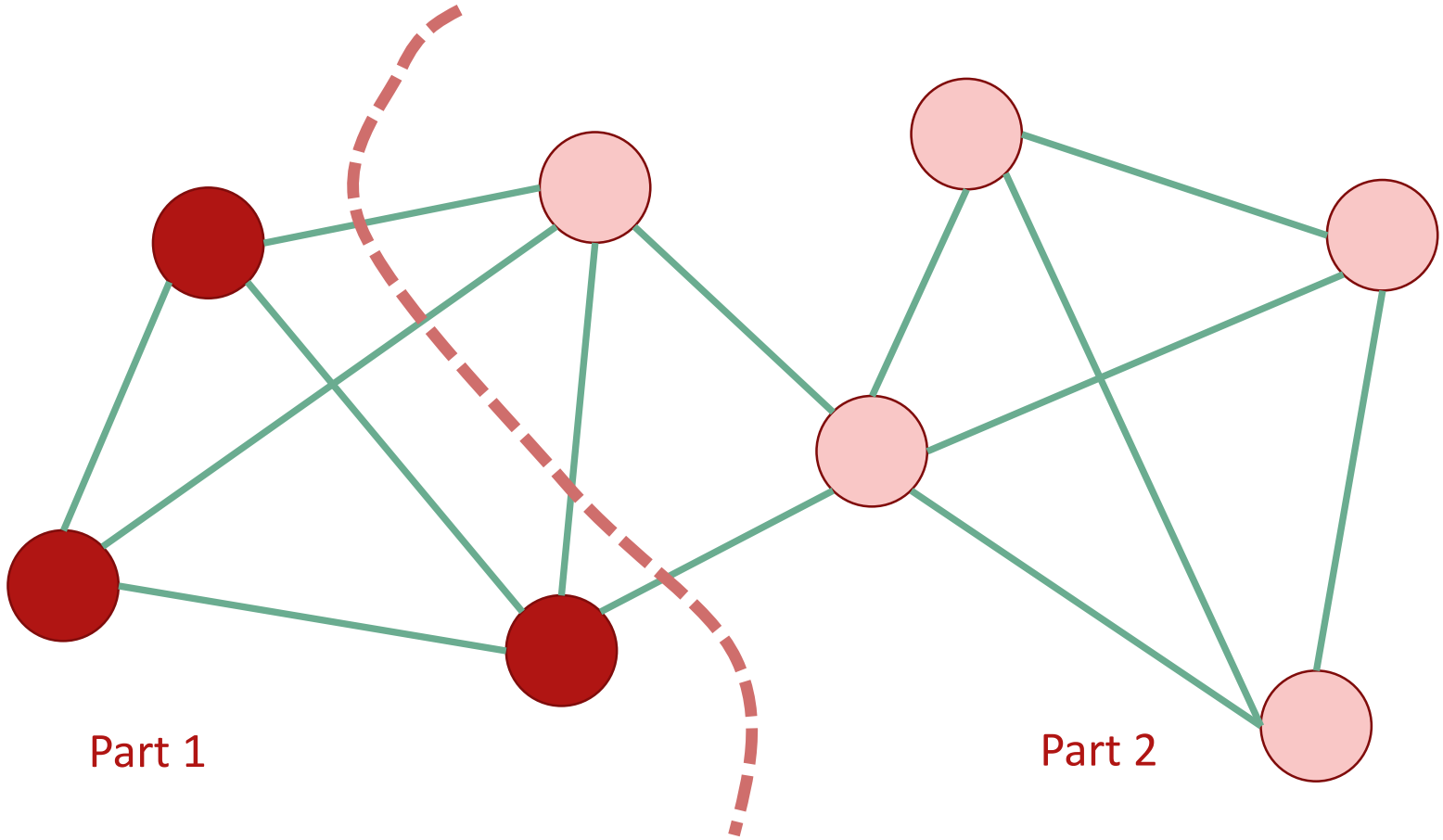
- Both are examples of greedy algorithms
  - Similar reasoning:
    - Optimal substructure: subgraphs generated by cuts.
    - The way to make safe choices is to choose light edges crossing the cut.

Today

- Minimum s-t cuts
- Maximum s-t flows
- The Ford-Fulkerson Algorithm
  - Finds min cuts and max flows!

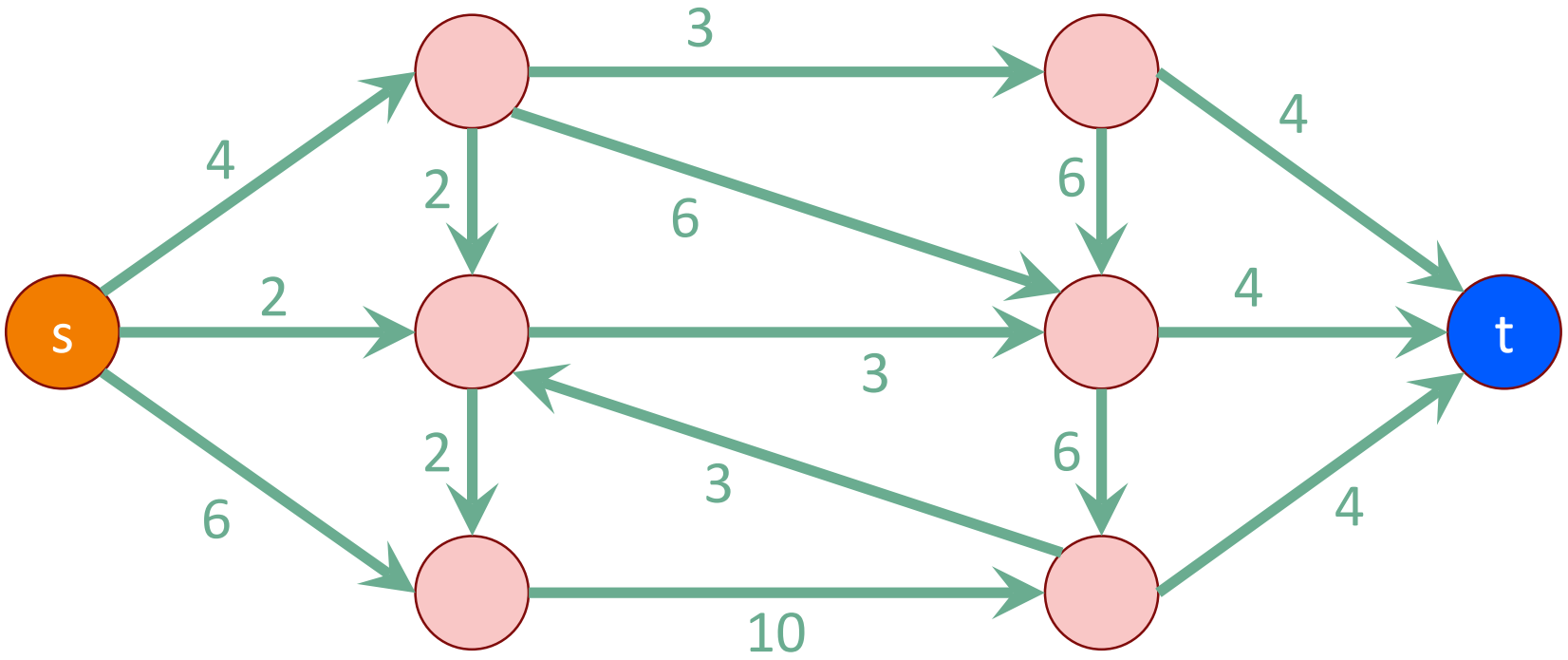
Cuts in graphs

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



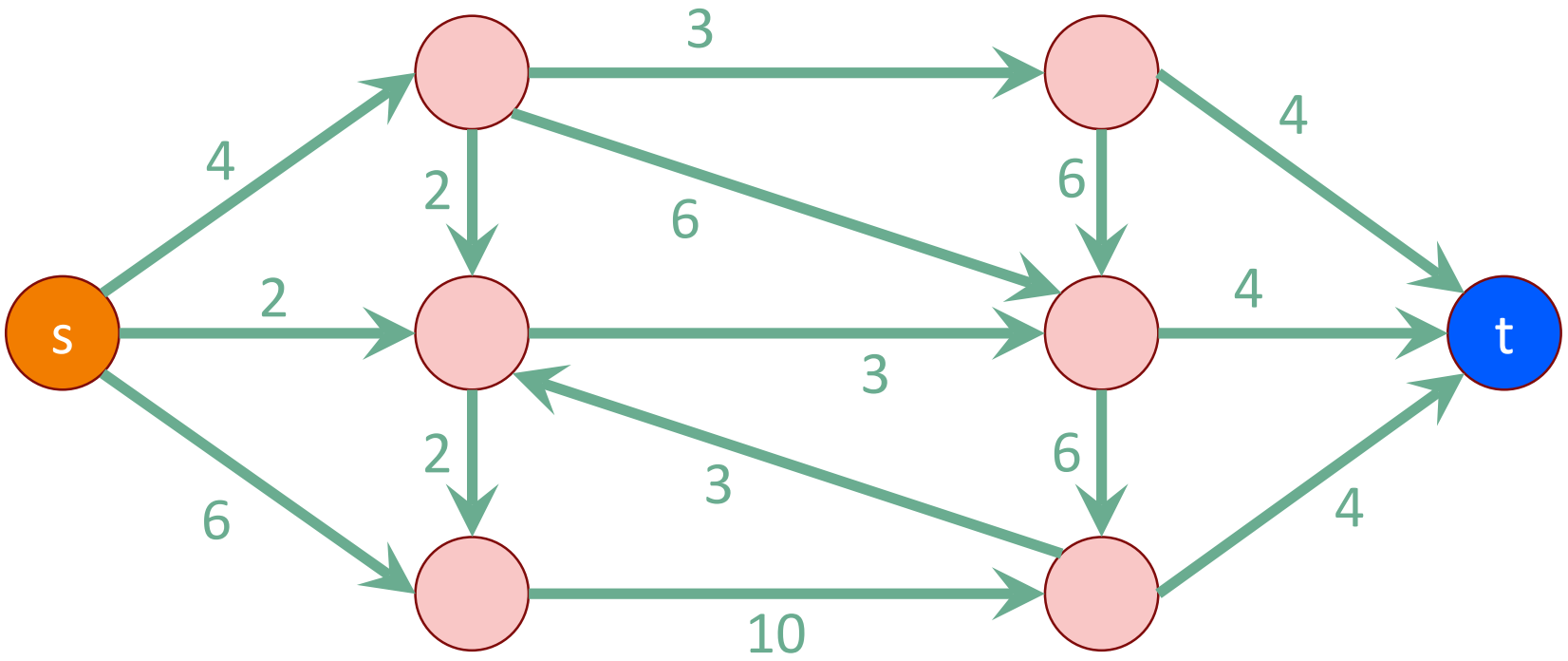
Today, we consider..

- Graphs are directed and edges have “capacities” (weights)
- We have a special “source” vertex s and “sink” vertex t.
  - * Simplifying for today: *s has only outgoing edges and t has only incoming edges*. Note everything can be generalized to arbitrarily directed graphs



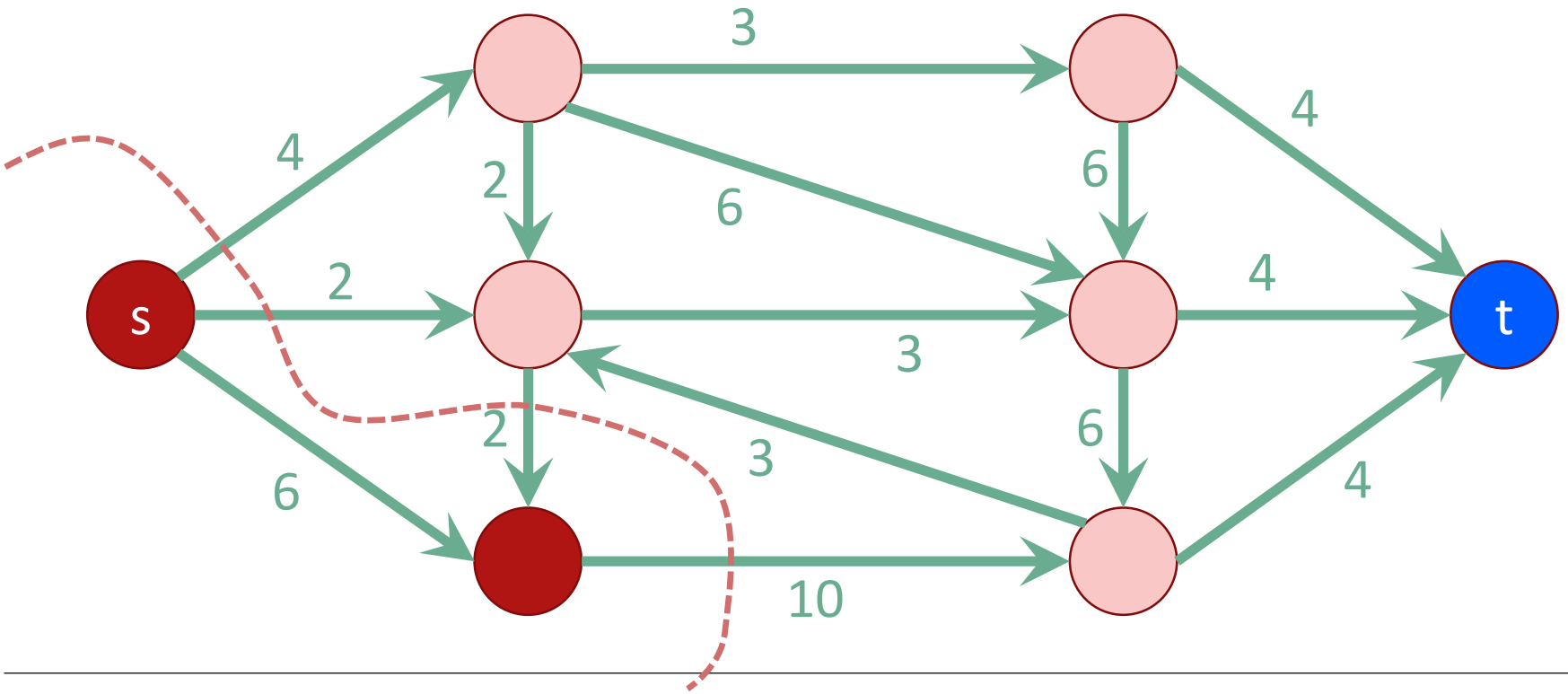
An s-t cut

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



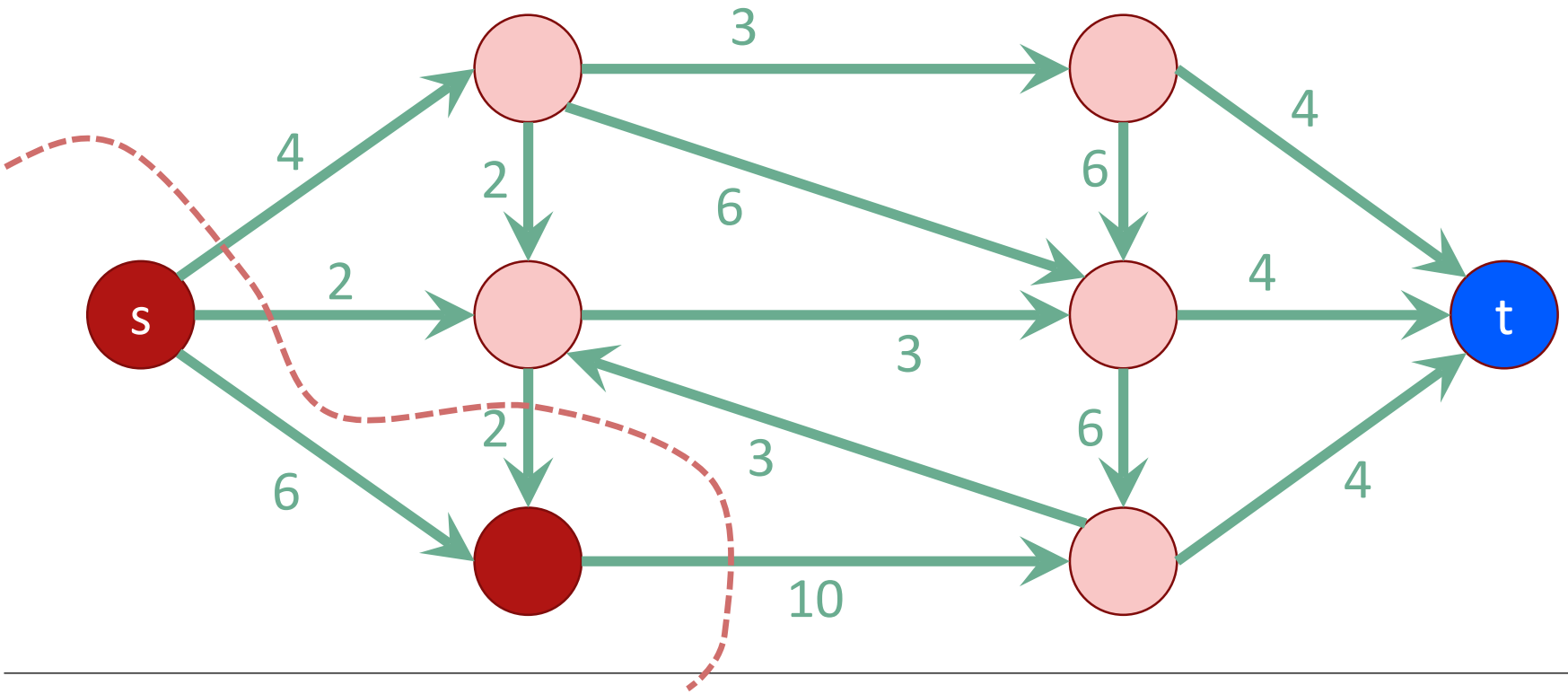
An s-t cut

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



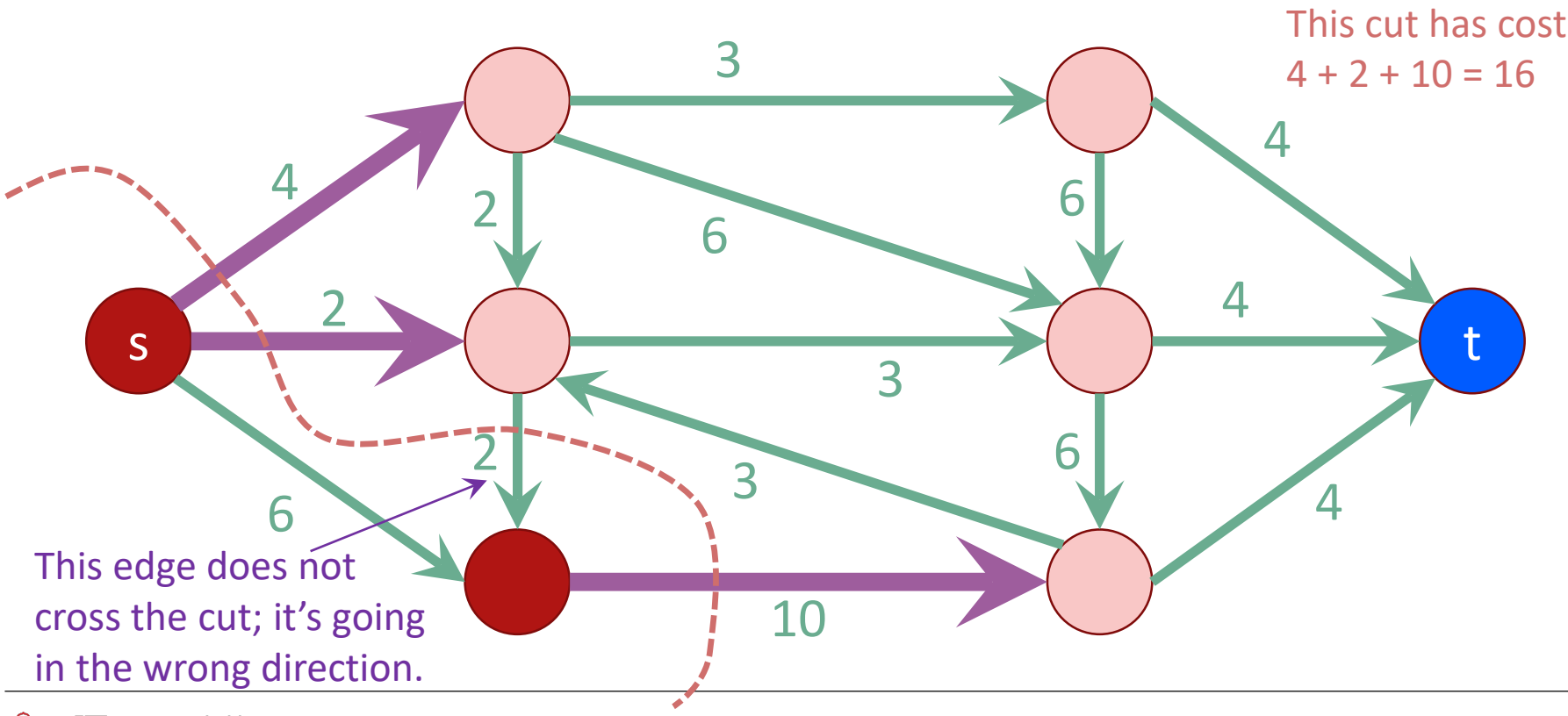
An s-t cut

- 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

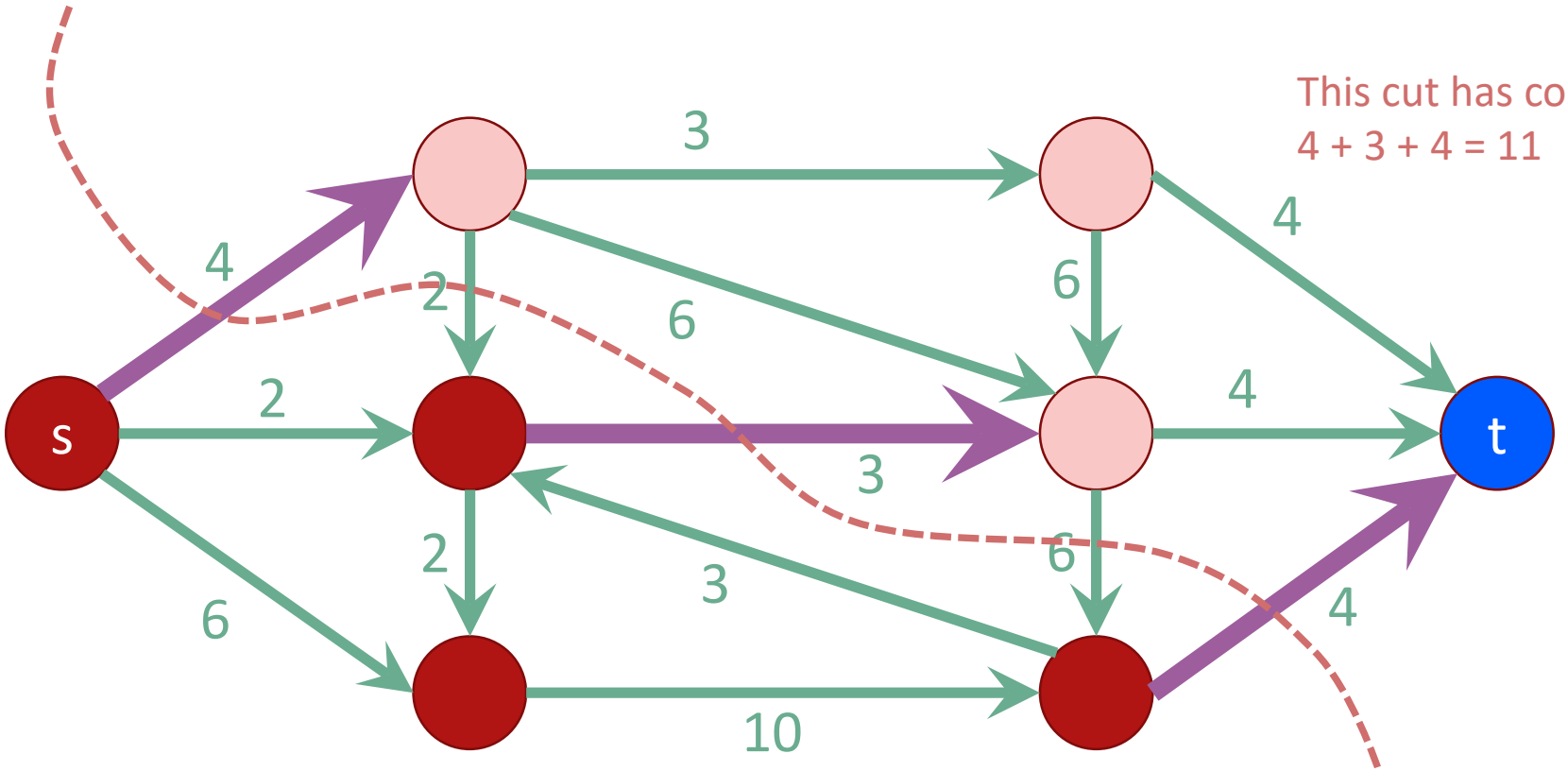
- An s-t cut is a cut which separates s from t.
- An edge **crosses the cut** if it goes from s's side to t's side.
- The **cost** (or capacity) of a cut is the sum of the capacities of the edges that cross the cut.



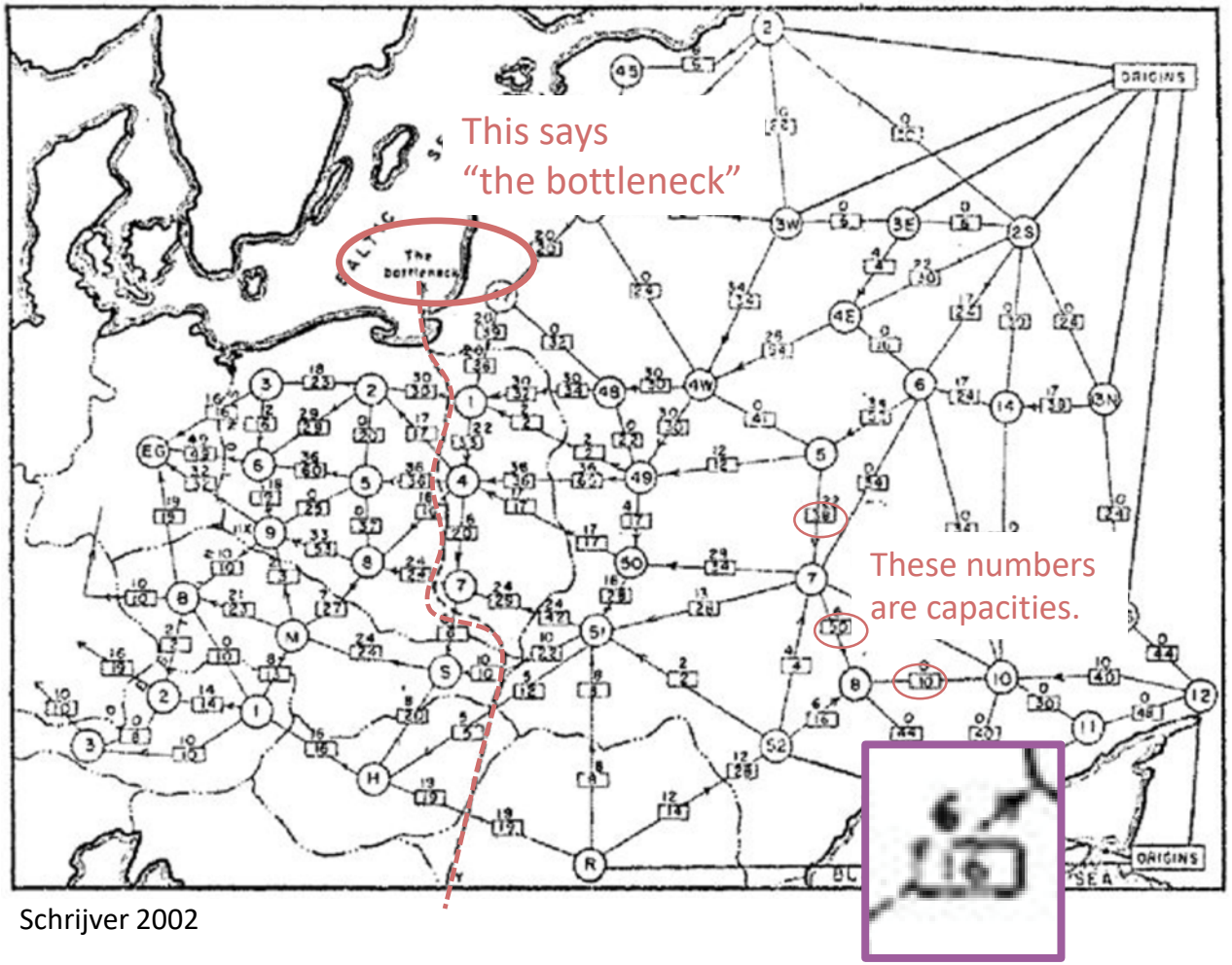
A minimum s-t cut

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

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



Where this comes up

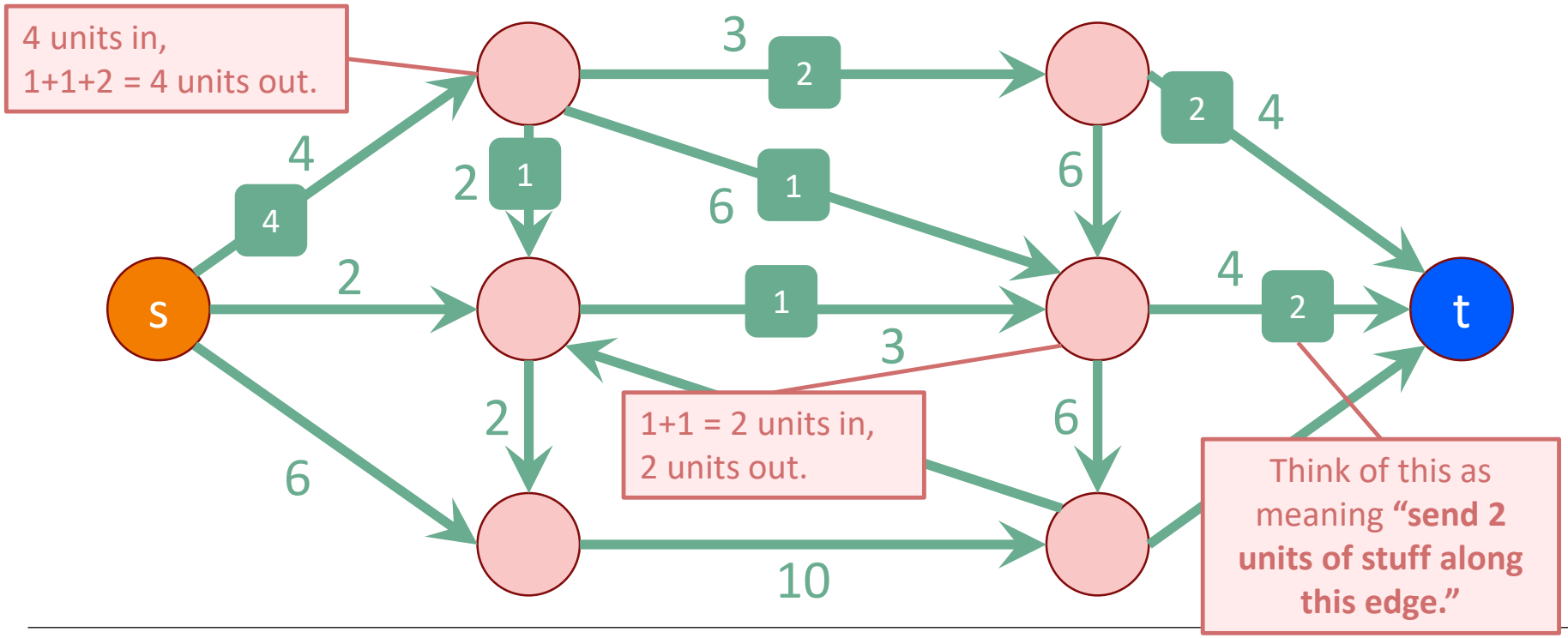


Schrijver 2002

- 1955 map of rail networks from the Soviet Union to Eastern Europe.
  - 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.

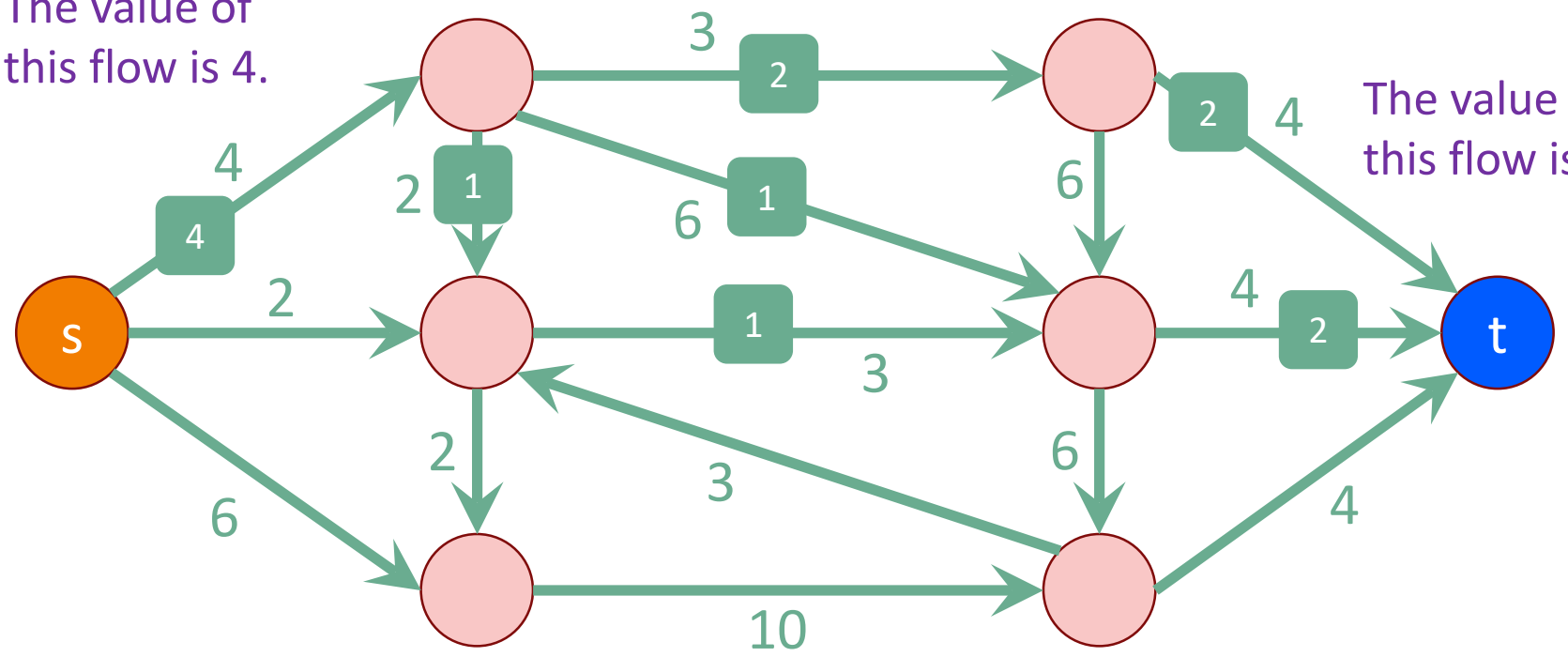


Flows

- 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**.

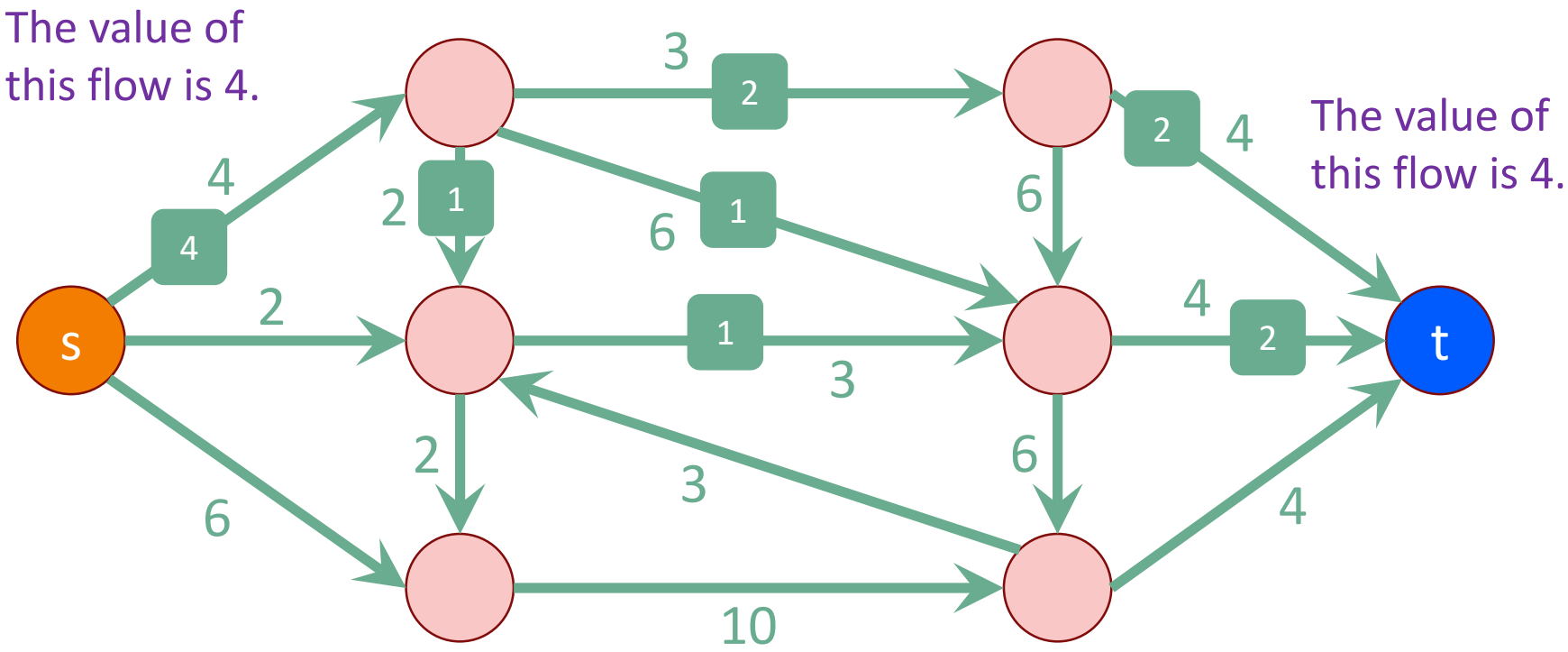
The value of this flow is 4.



The value of this flow is 4.

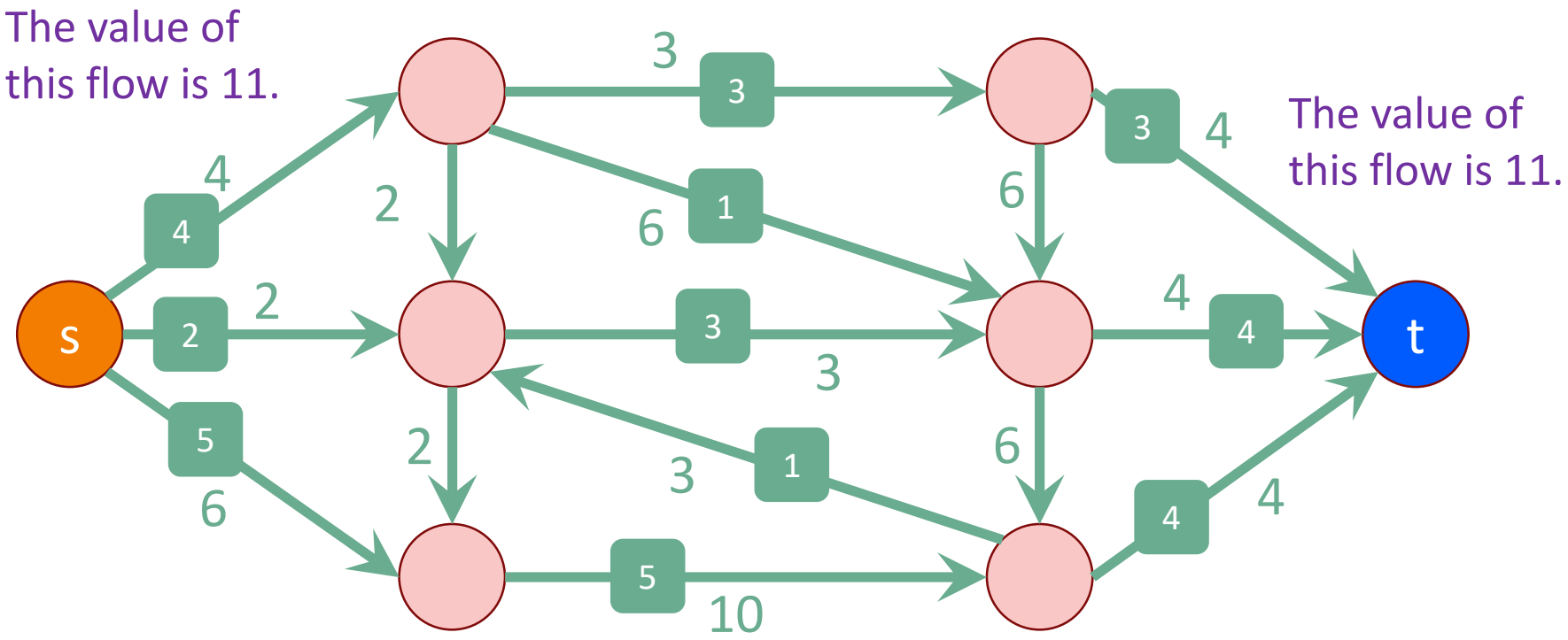
A maximum flow

- A maximum flow is a flow of maximum value.
- This example flow is pretty **wasteful**, I'm not utilizing the capacities very well.

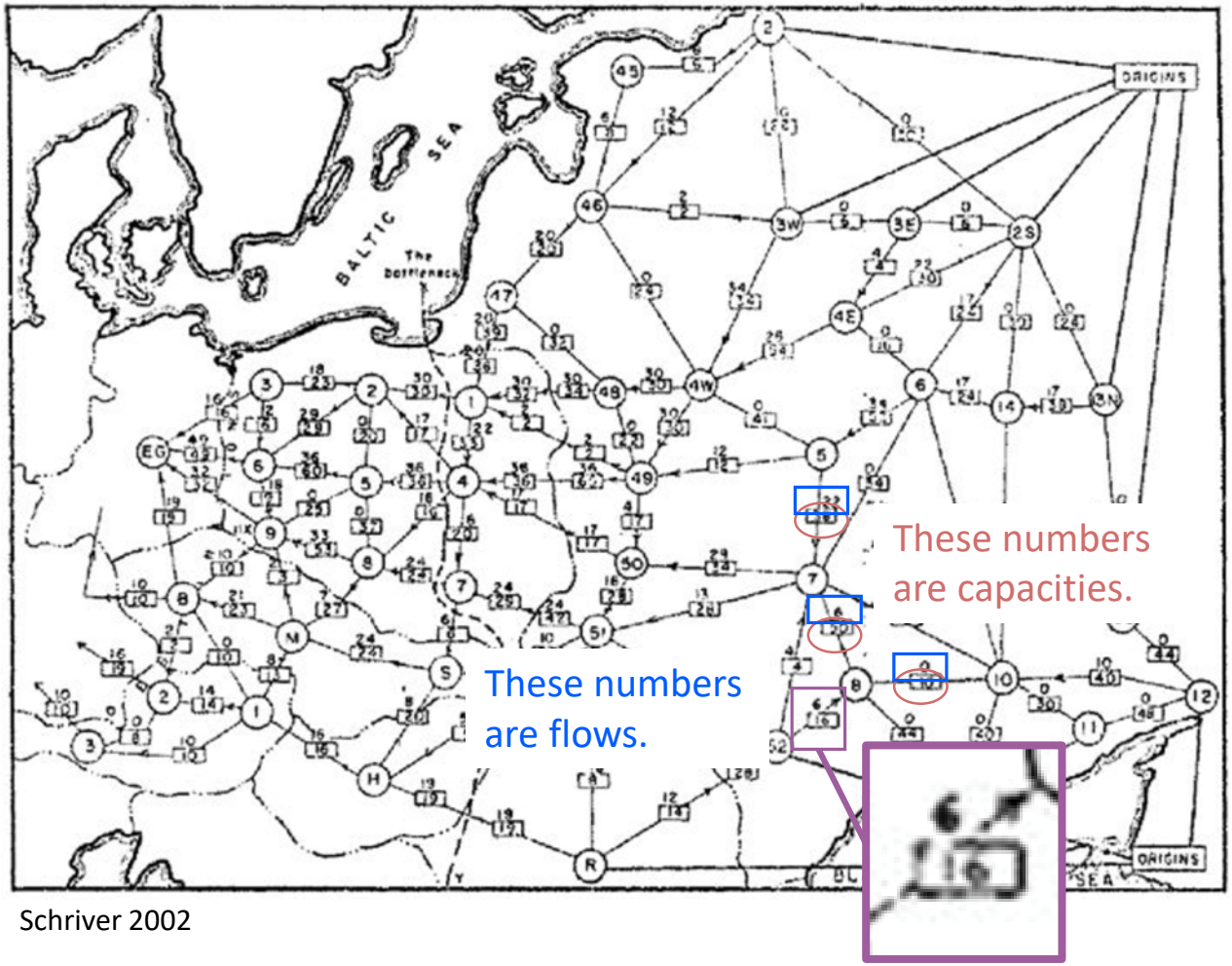


A maximum flow

- A maximum flow is a flow of maximum value.
- This one is **maximum**; it has value 11.



Where this comes up



Schriver 2002

- 1955 map of rail networks from the Soviet Union to Eastern Europe.
  - 44 edges, 105 vertices.
- The Soviet Union wants to route supplies from suppliers in Russia to Eastern Europe as efficiently as possible.
- In 1955, Ford and Fulkerson gave an algorithm which finds the optimal flow.

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

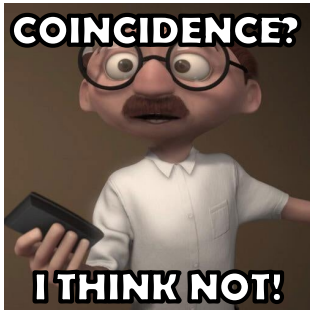
Air power is an effective means of interdicting an enemy's rail system, and such usage is a logical and important mission for this Arm.

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.

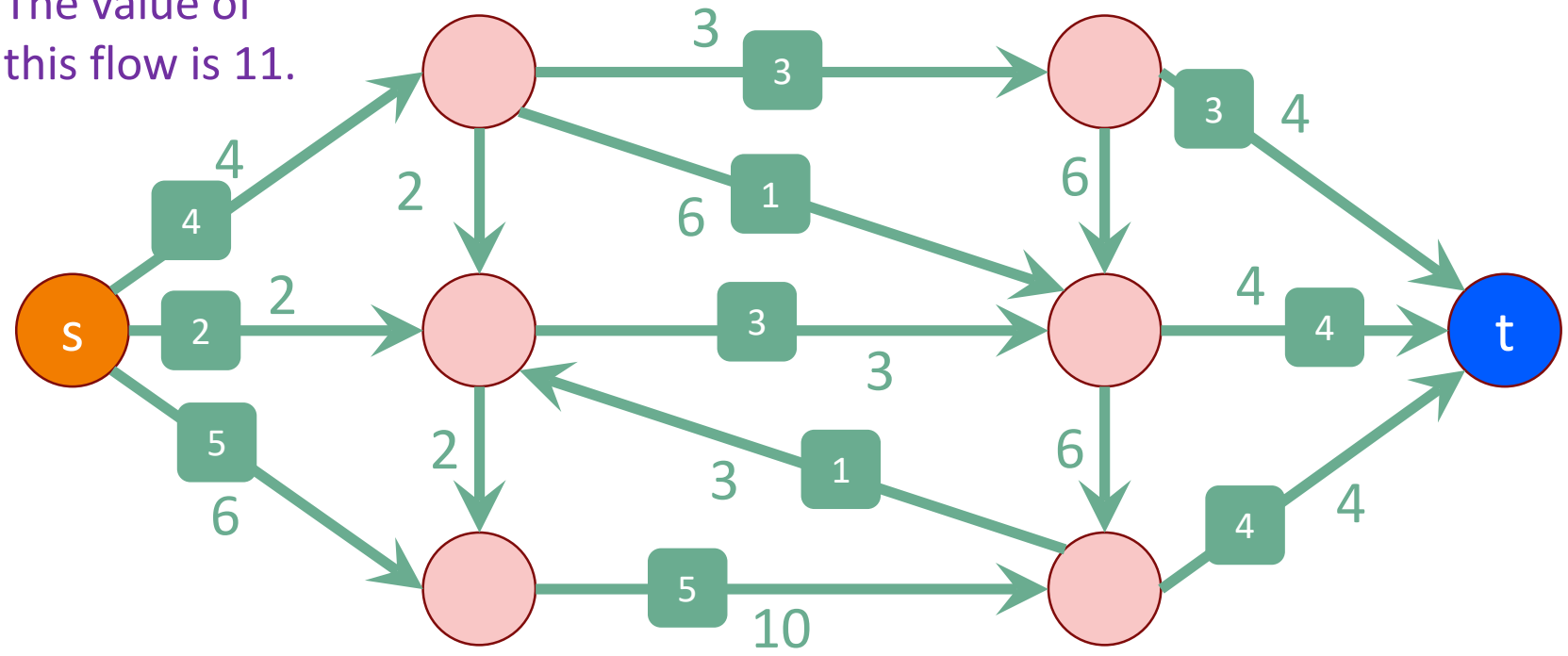
The present paper describes the fundamentals of a method intended to help the specialist who is engaged in estimating railway capacities, so that he might more readily accomplish this purpose and thus assist the commander and his staff with greater efficiency than is possible at present.

A maximum flow

- 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!**



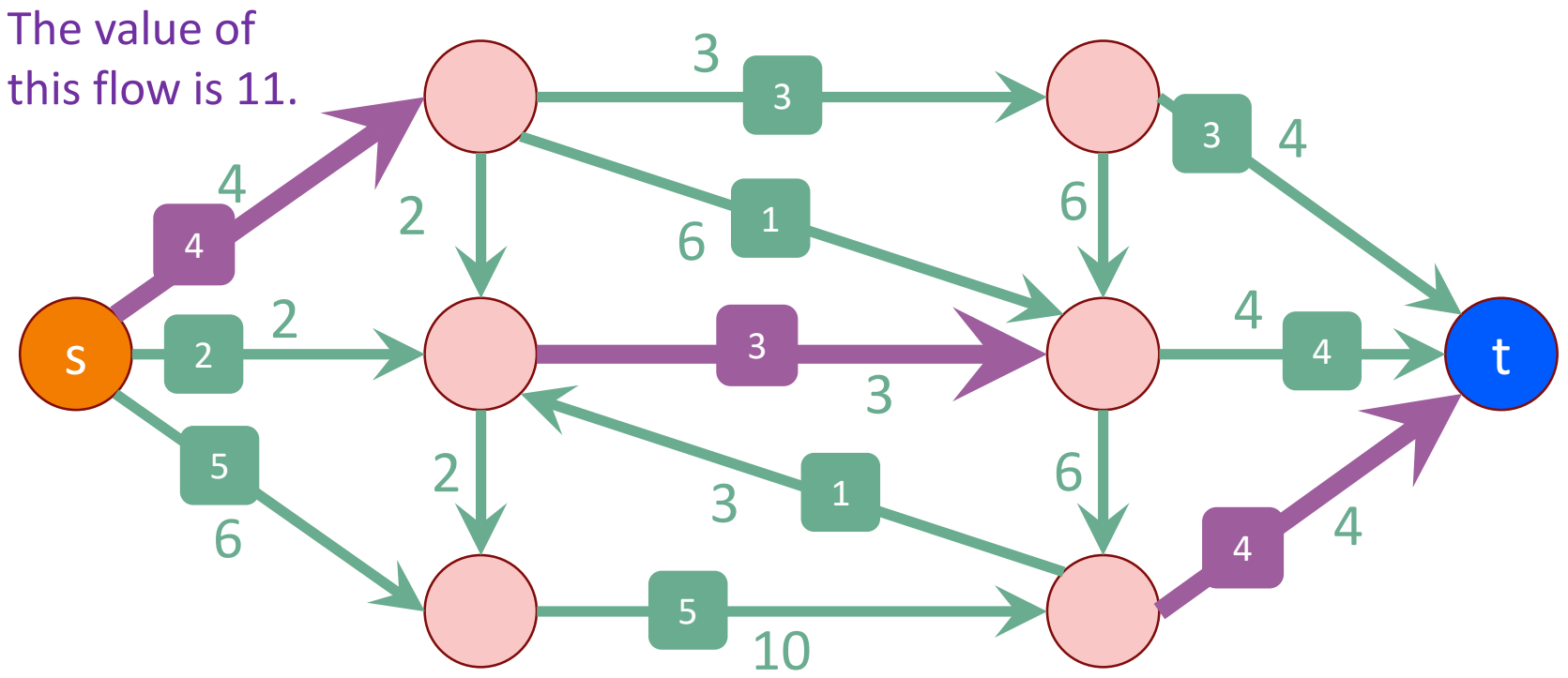
The value of this flow is 11.



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.



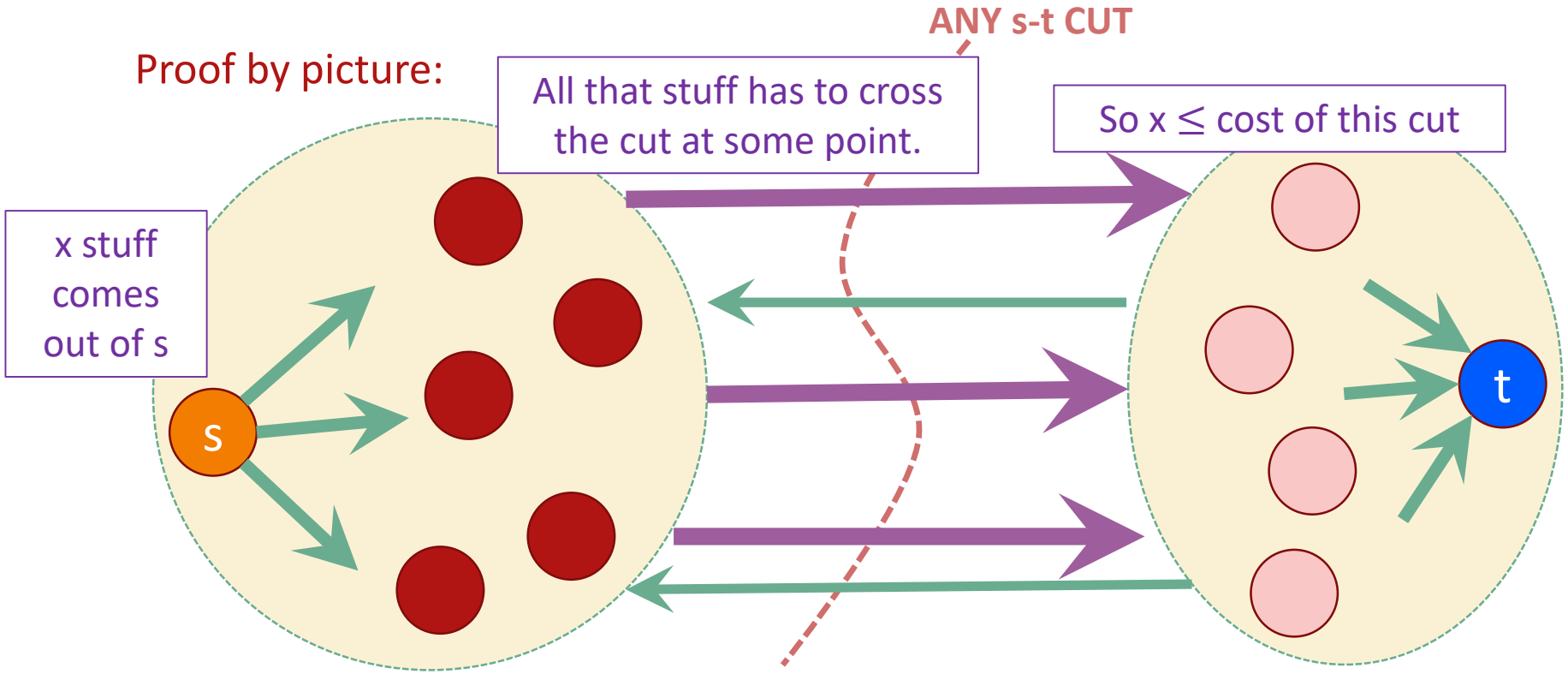
Proof outline

- **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.



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.

Ford-Fulkerson Algorithm

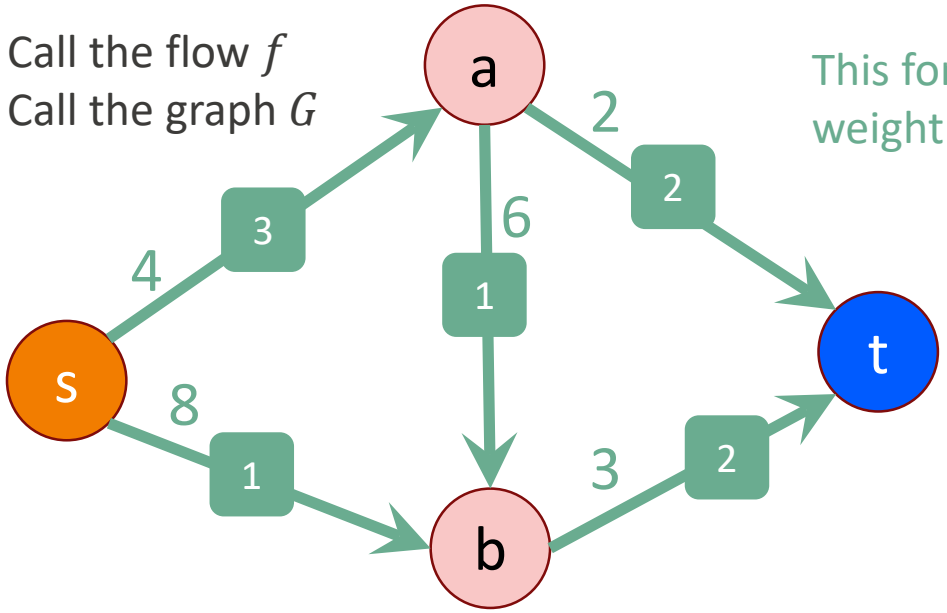
- **Outline of algorithm:**
  - Start with zero flow.
  - We will maintain a **"residual graph"** G_f .
  - A path from s to t in G_f will give us a way to improve our flow.
  - We will continue until there are no s-t paths left.

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

Residual networks

- Residual networks say we have a flow.

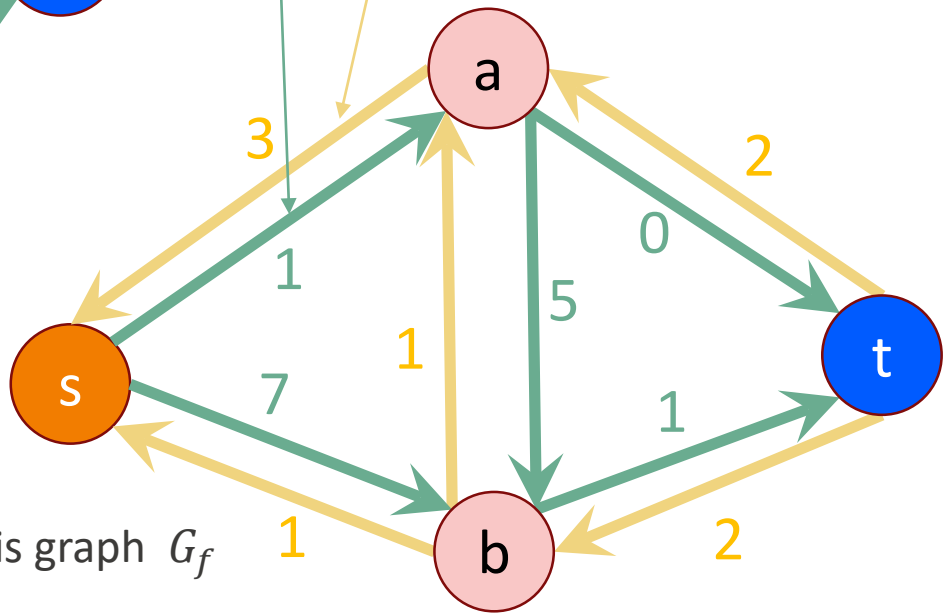
Call the flow f
Call the graph G



This forward edge has weight "capacity – flow".

This backward edge has weight "flow".

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

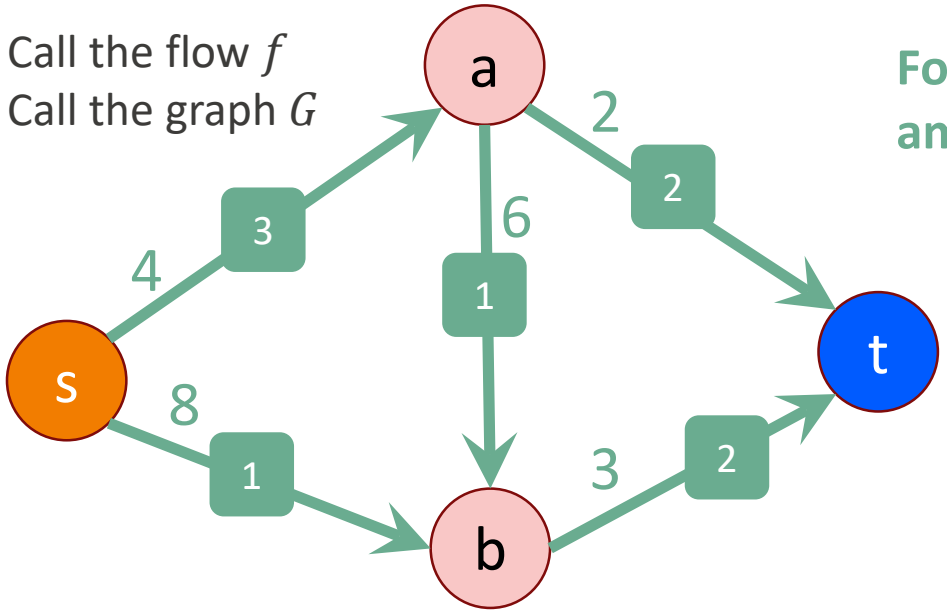


Call this graph G_f

Residual networks

- Residual networks say we have a flow.

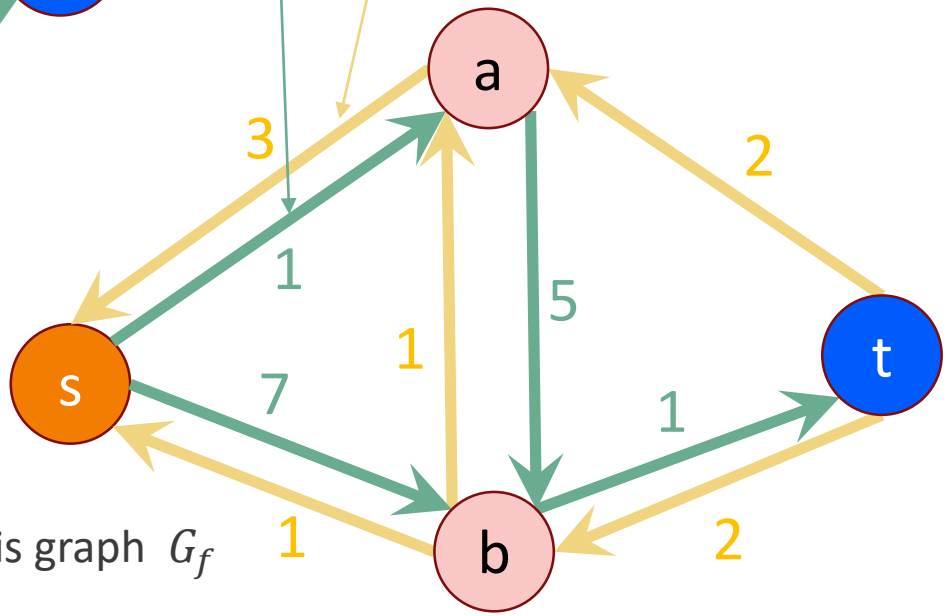
Call the flow f
Call the graph G



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:

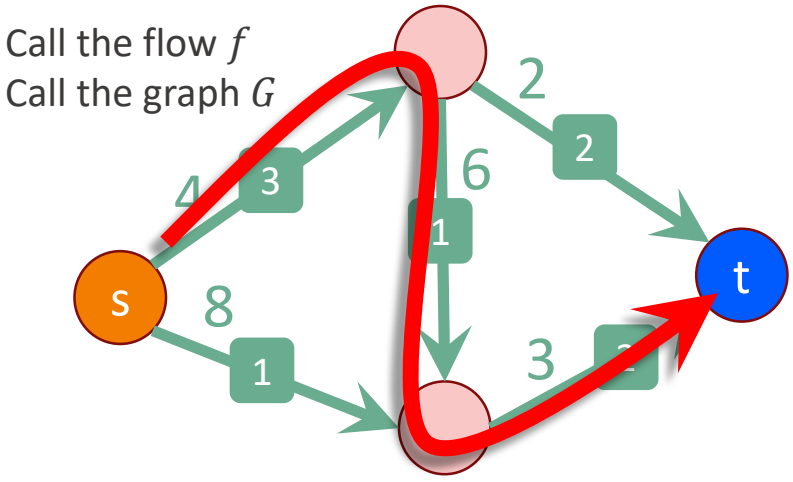


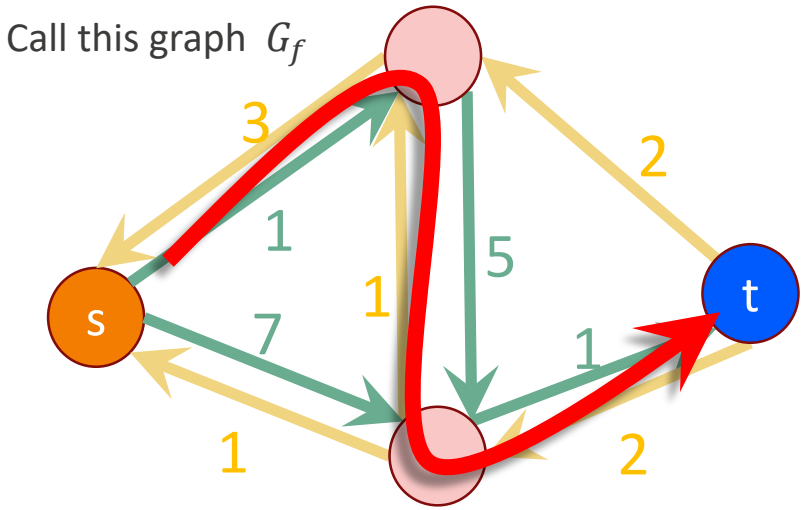
Call this graph G_f

Residual networks

- 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.**

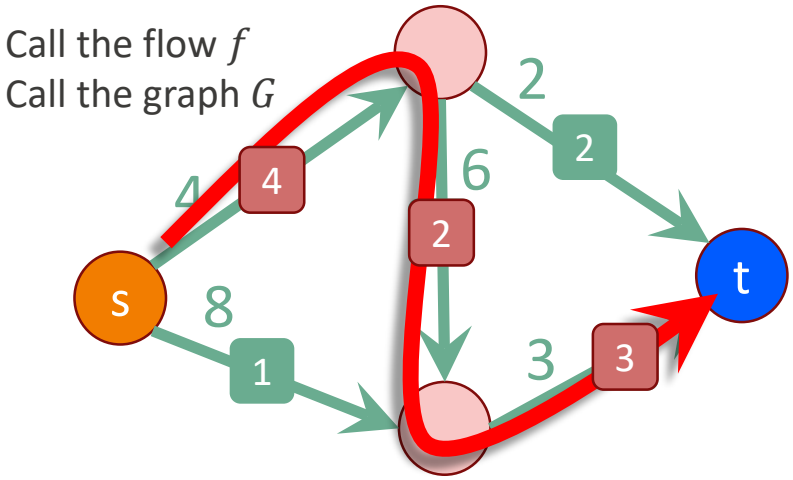
- **Claim**: If there is an augmenting path, we can increase the flow along that path.

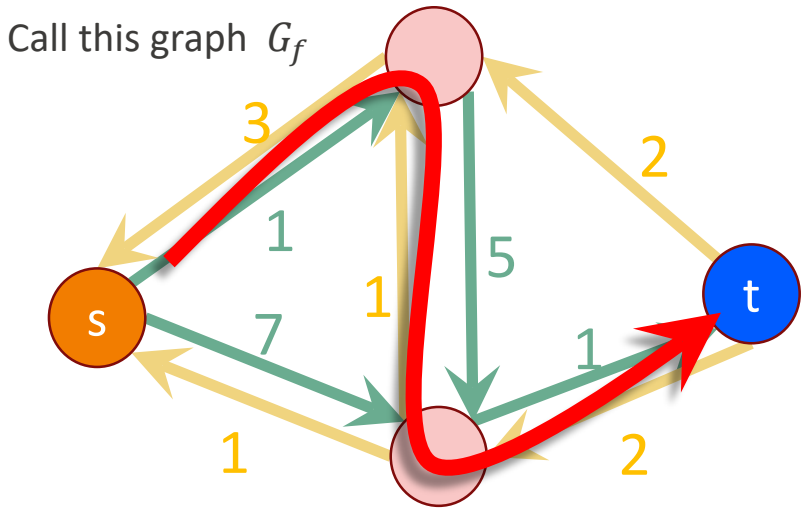




Residual networks

- **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**.





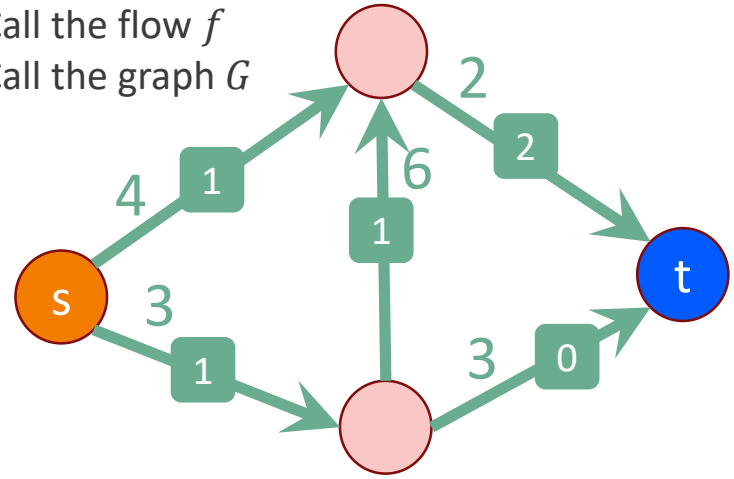
- Forward edges indicate how much stuff can still go through.
- Just increase the flow on all the edges!

Residual networks

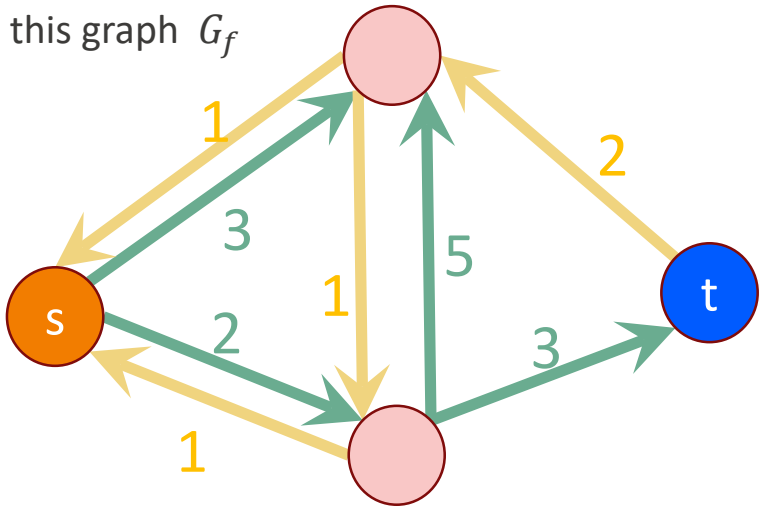
- **Claim:** if there is an augmenting path, we can increase the flow along that path.
- **Harder case:** there are **backward edges** in the path.
  - Here's a slightly different example of a flow:

/ I changed some of the weights and edge directions. */*

Call the flow f
Call the graph G



Call this graph G_f

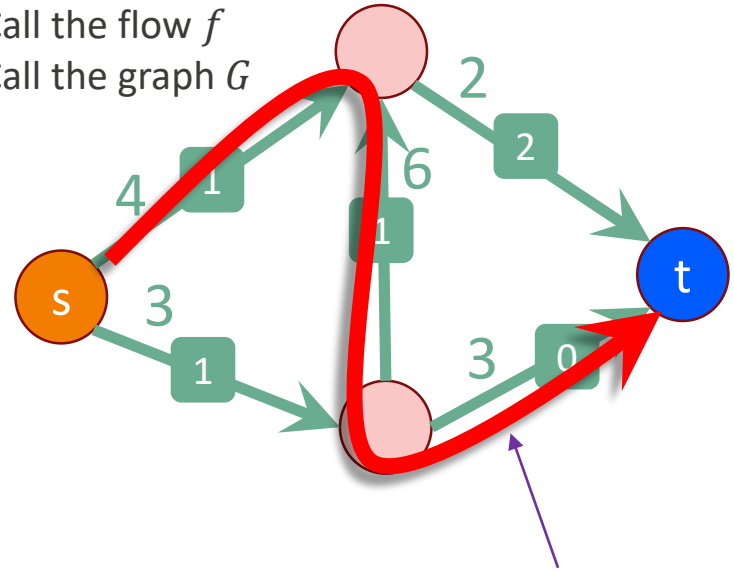


Residual networks

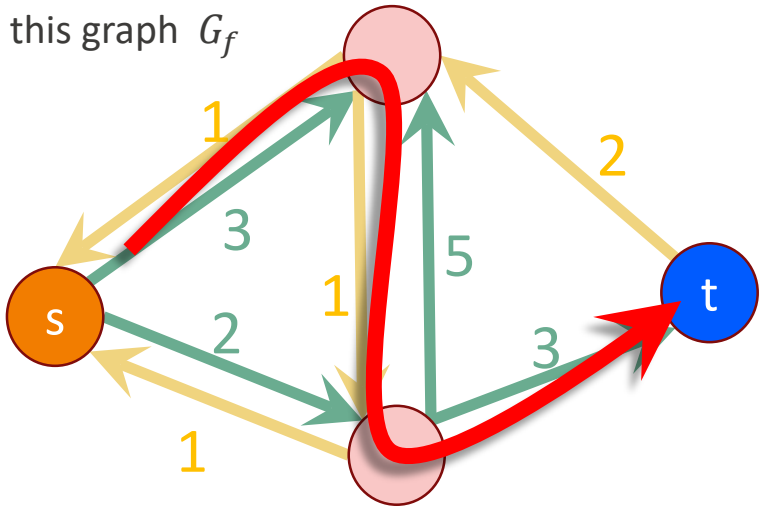
- **Claim:** if there is an augmenting path, we can increase the flow along that path.
- **Harder case:** there are **backward edges** in the path.
  - Here's a slightly different example of a flow:

/ I changed some of the weights and edge directions. */*

Call the flow f
Call the graph G



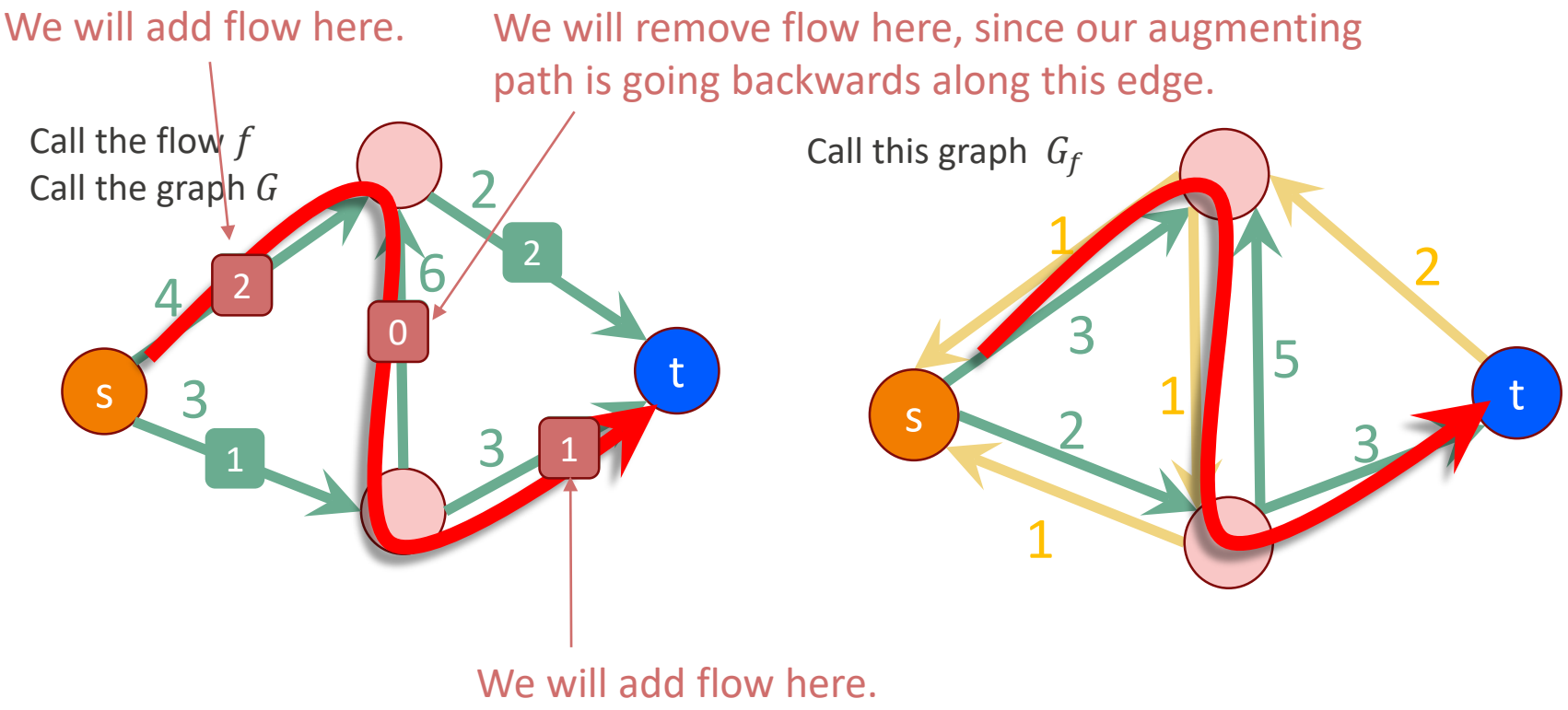
Call this graph G_f



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.

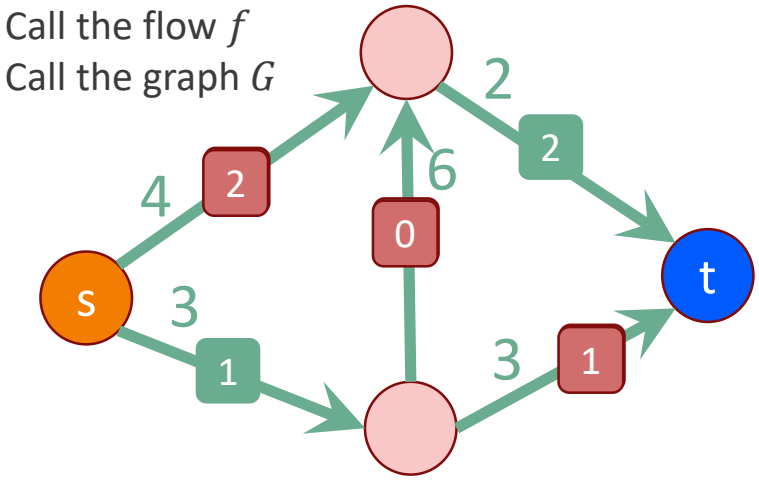
Residual networks

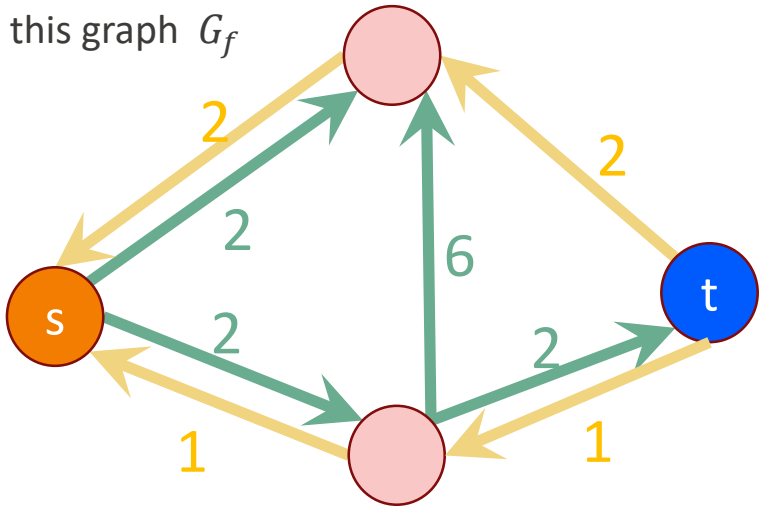
- **Claim:** if there is an augmenting path, we can increase the flow along that path.
- In this case we do something a bit different:



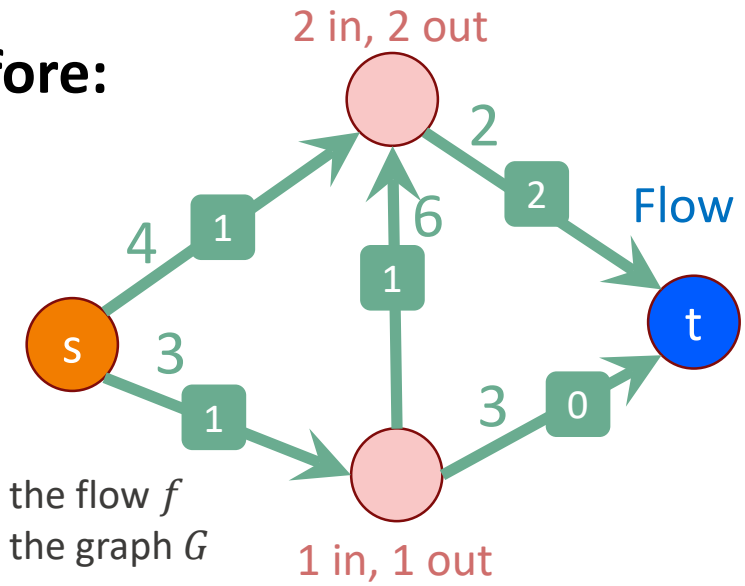
Residual networks

- **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:



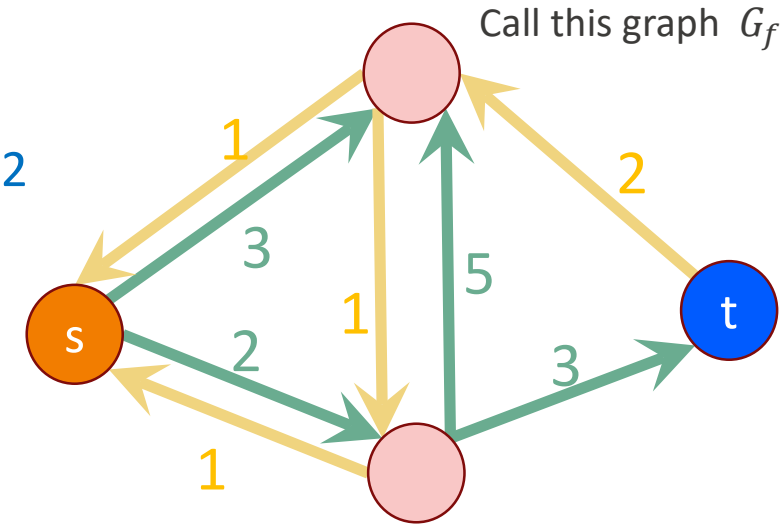


Before:

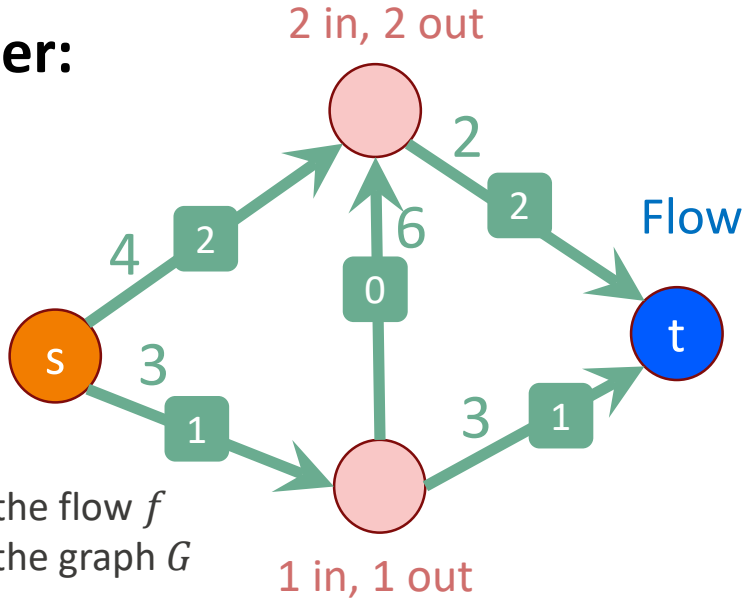


Flow value is 2

Call the flow f
Call the graph G

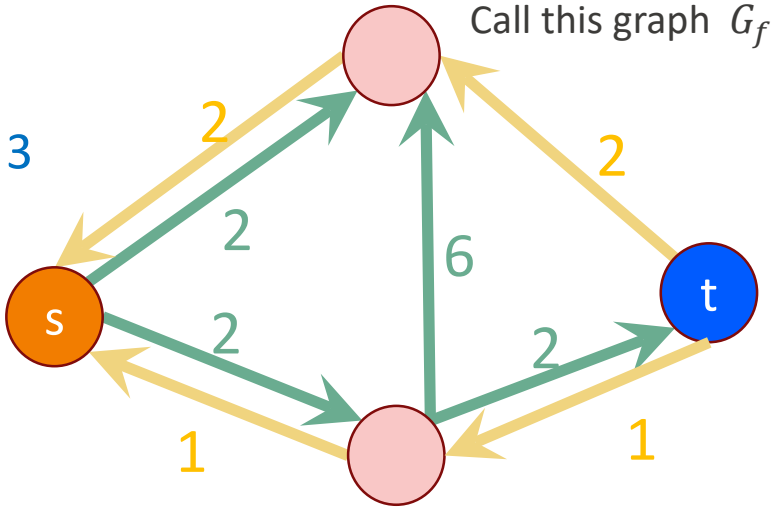


After:



Flow value is 3

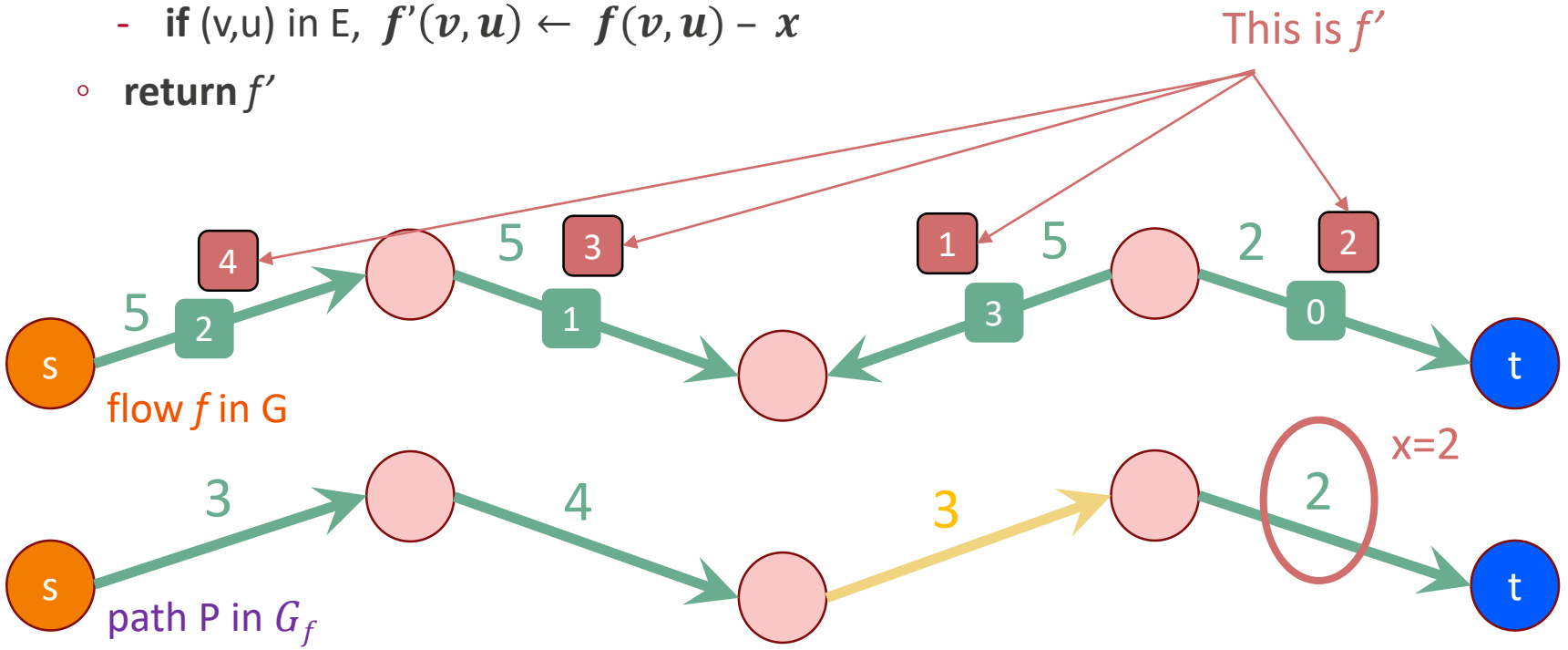
Call the flow f
Call the graph G



Still a legit flow, but with a bigger value!

Residual networks

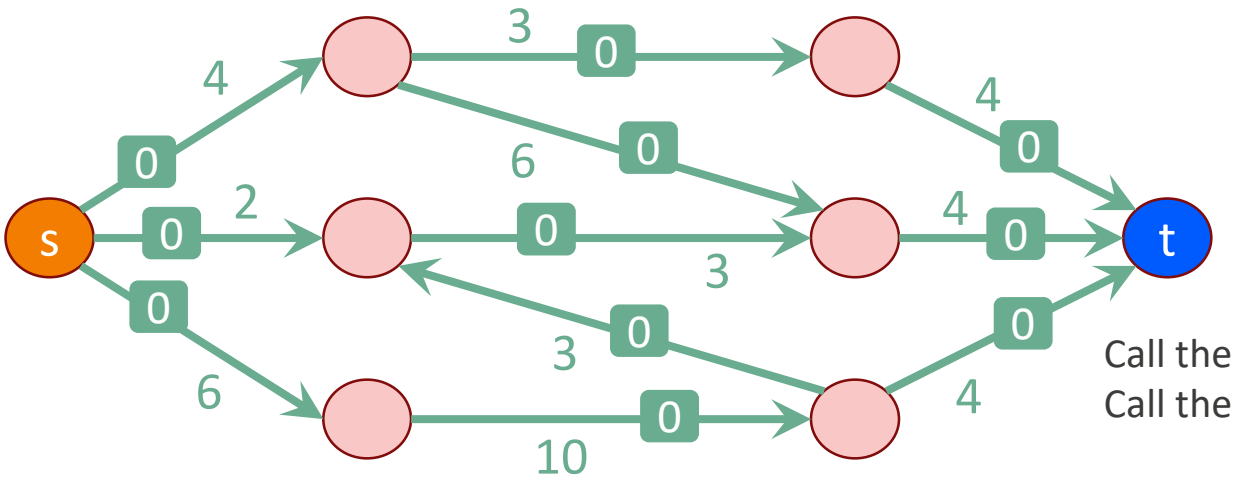
- **increaseFlow(path P in G_f , flow f):**
  - x = min weight on any edge in P
  - **for** (u,v) in P:
    - **if** (u,v) in E, $f'(u,v) \leftarrow f(u,v) + x$.
    - **if** (v,u) in E, $f'(v,u) \leftarrow f(v,u) - x$
  - **return** f'



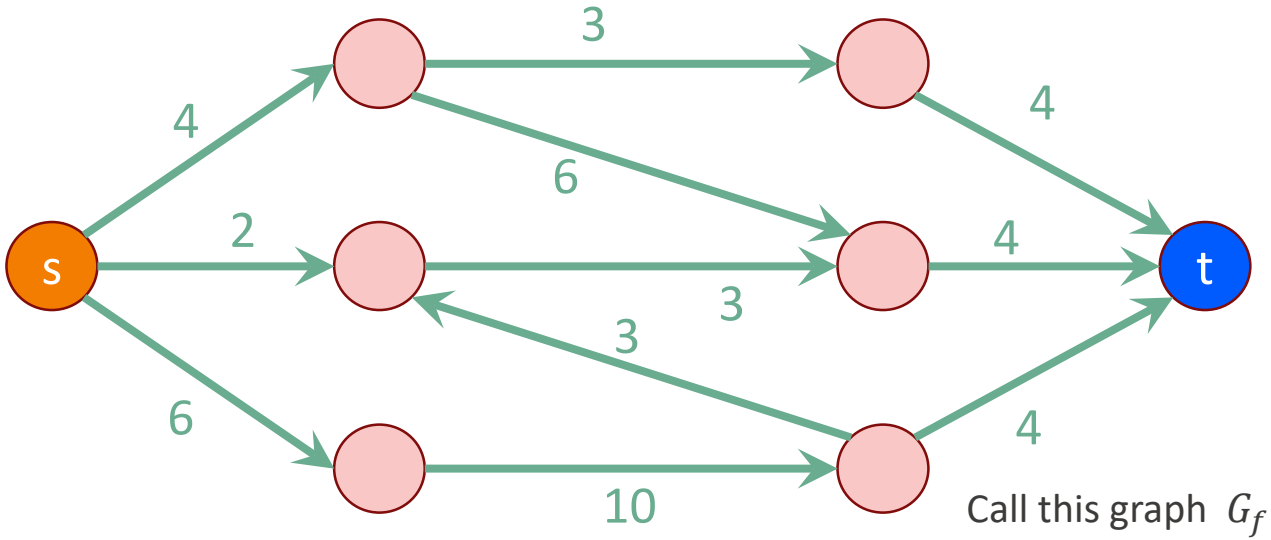
Ford-Fulkerson Algorithm

- **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 // e.g., use DFS or BFS
    - $f \leftarrow$ **increaseFlow**(*P, f*)
    - update G_f
  - **return** f

Example of Ford-Fulkerson

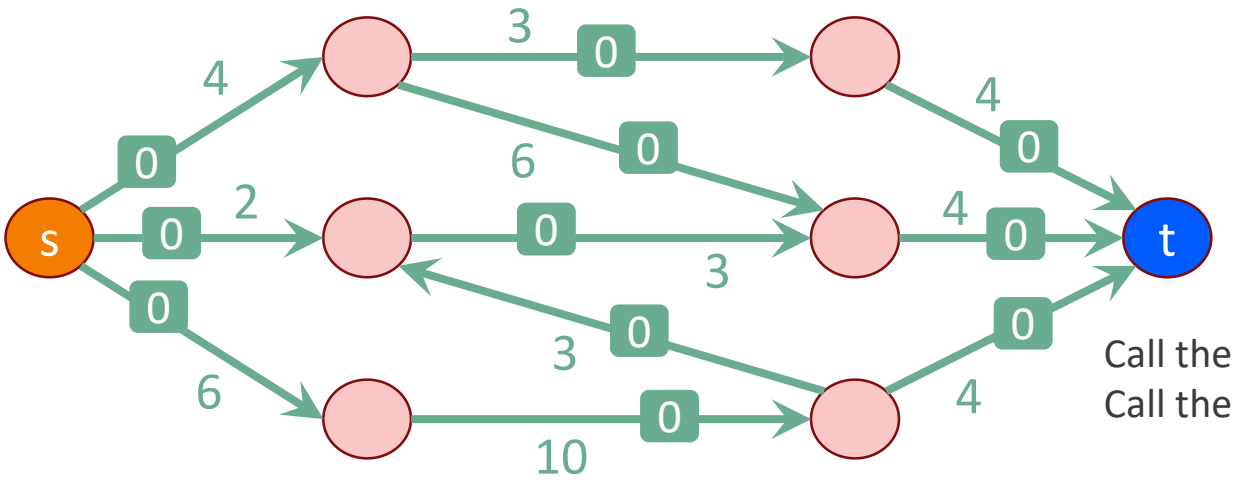


Call the flow f
Call the graph G

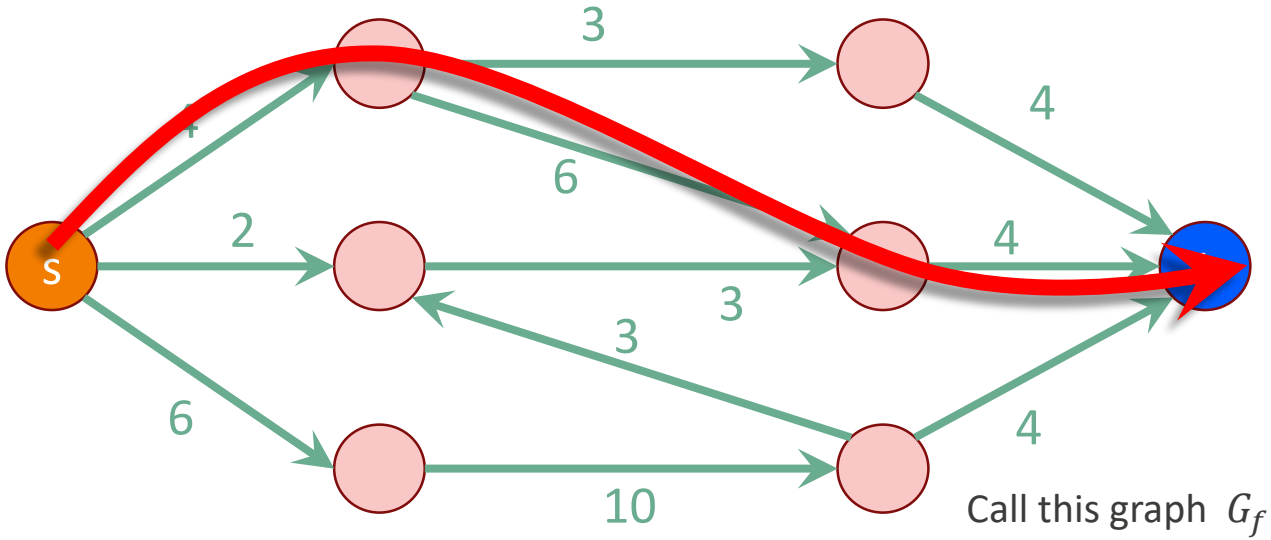


Call this graph G_f

Example of Ford-Fulkerson

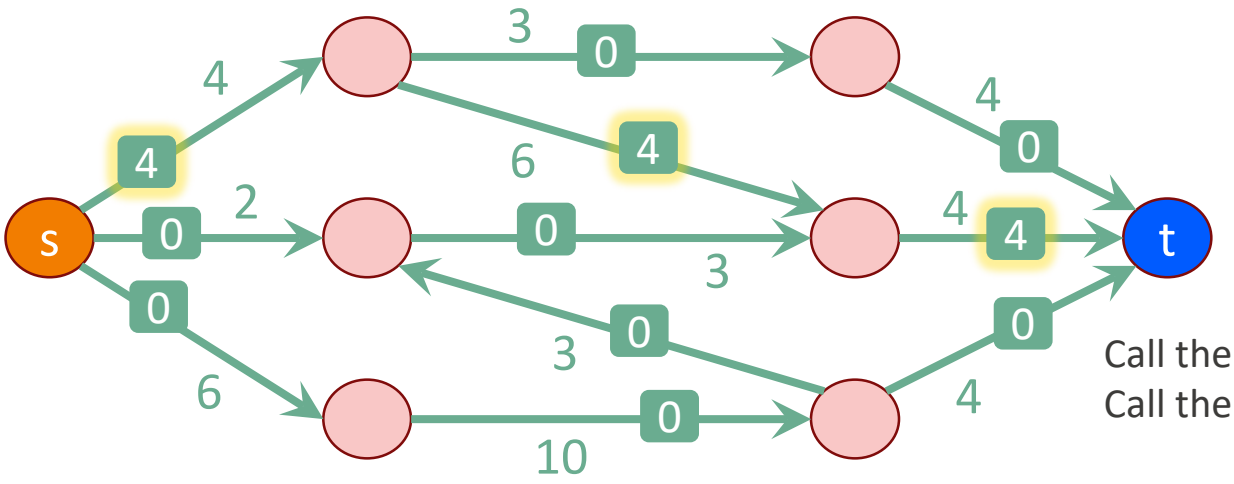


Call the flow f
Call the graph G

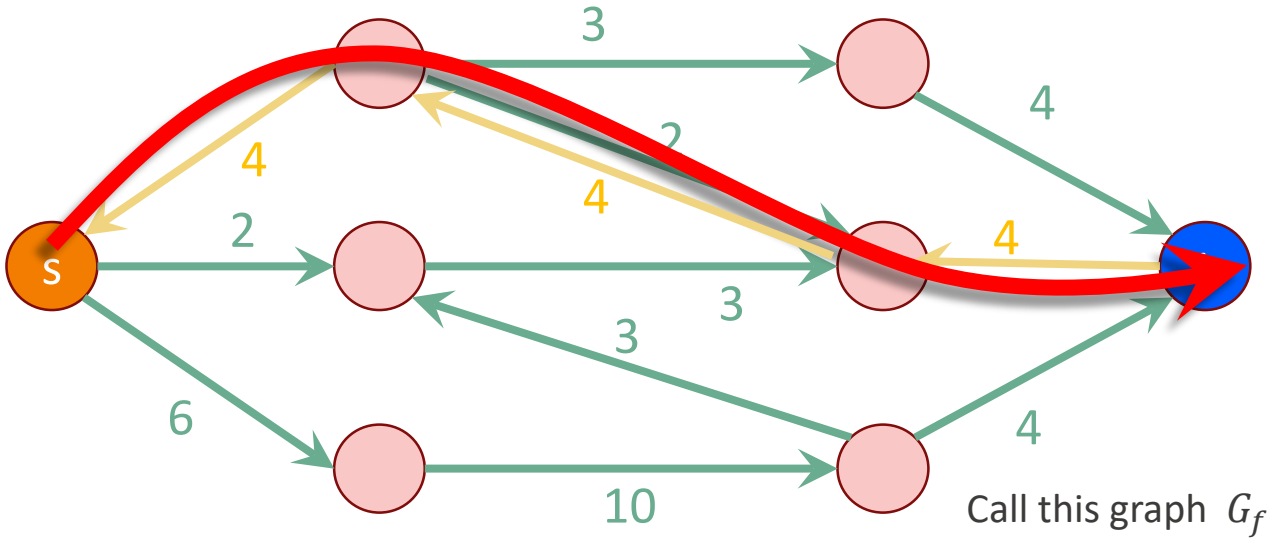


Call this graph G_f

Example of Ford-Fulkerson

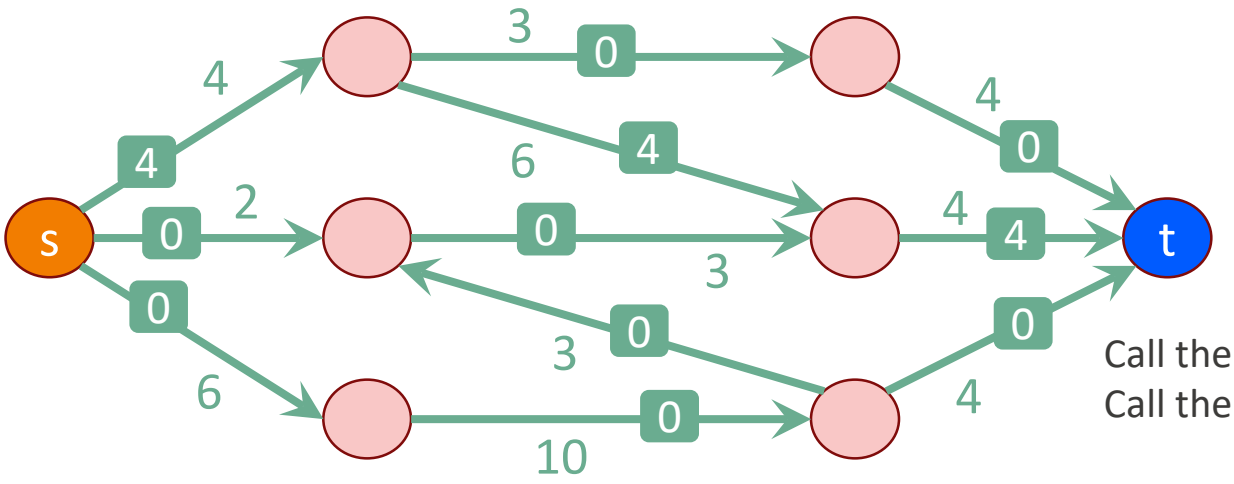


Call the flow f
Call the graph G

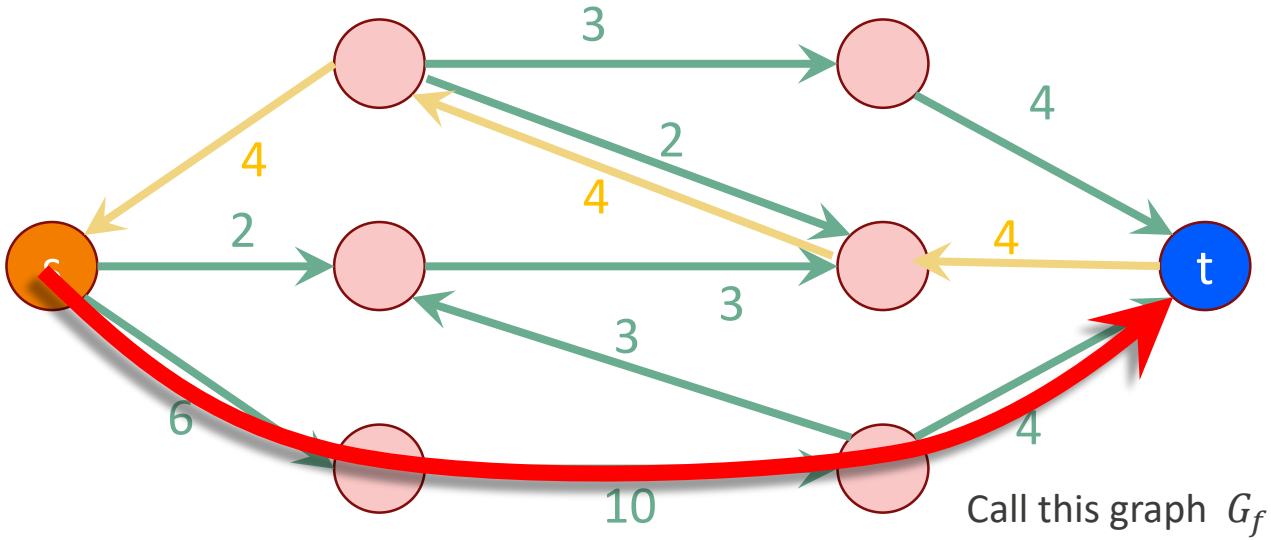


Call this graph G_f

Example of Ford-Fulkerson

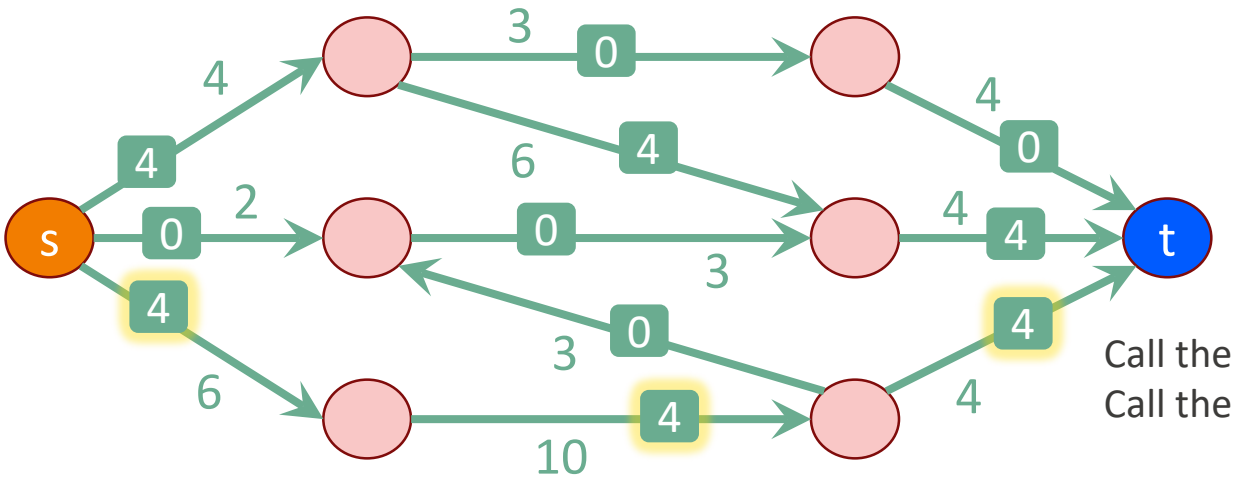


Call the flow f
Call the graph G

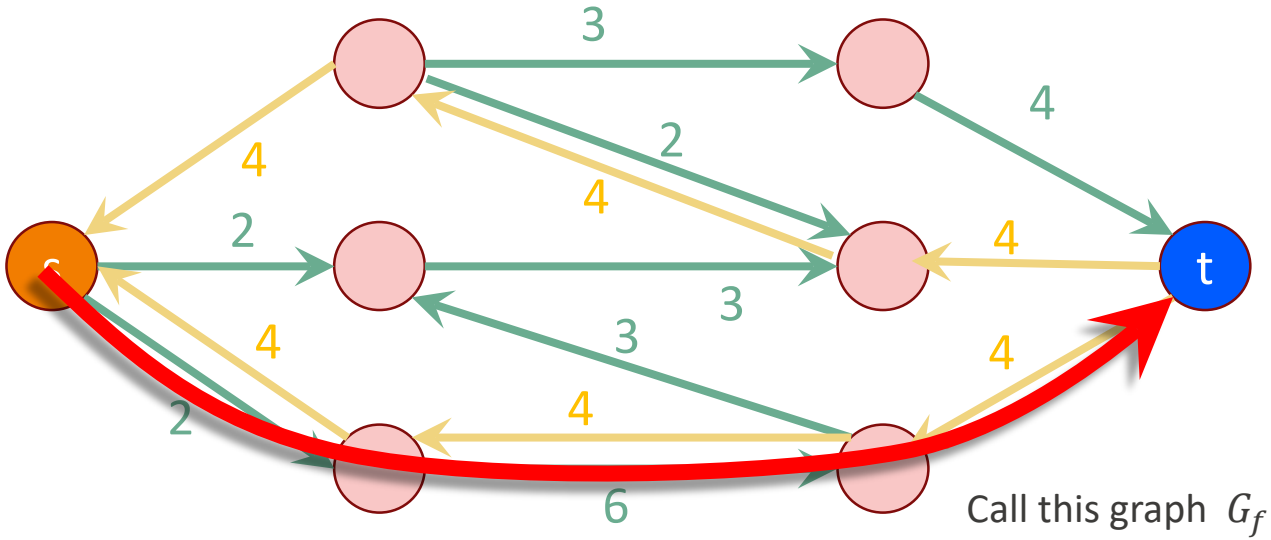


Call this graph G_f

Example of Ford-Fulkerson

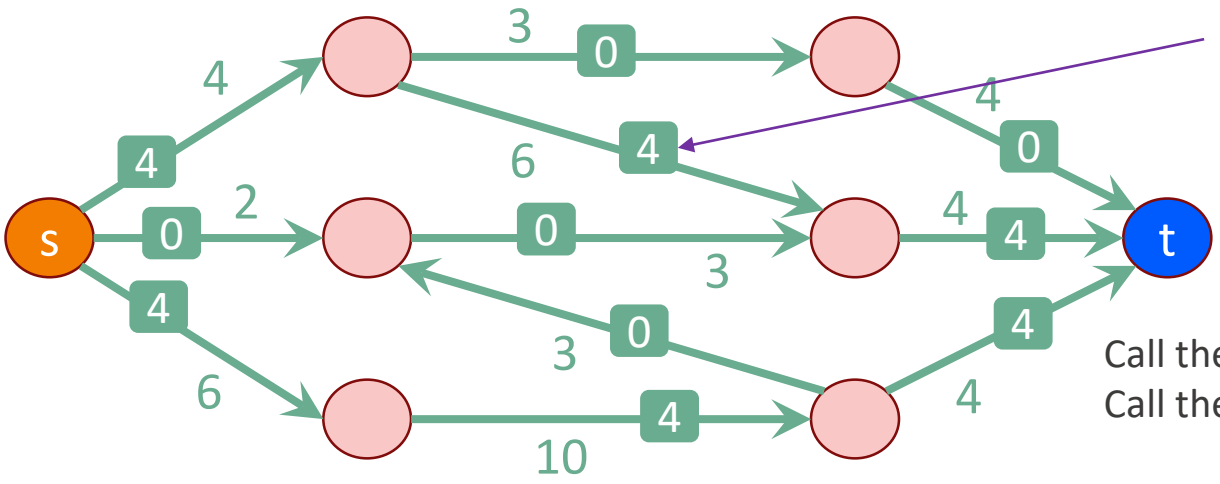


Call the flow f
Call the graph G



Call this graph G_f

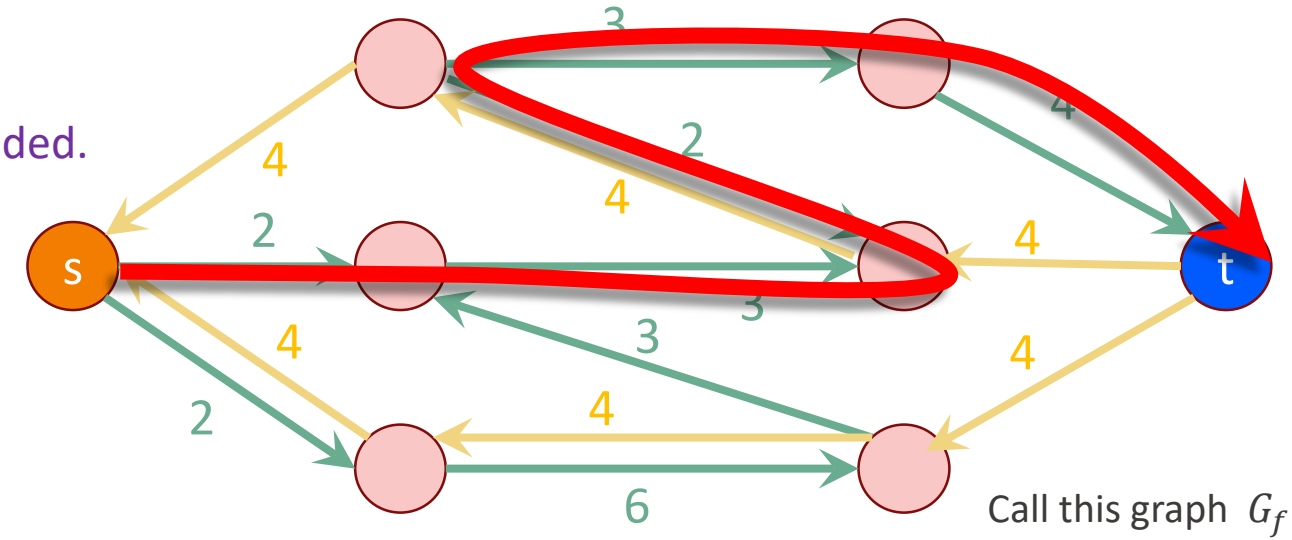
Example of Ford-Fulkerson



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

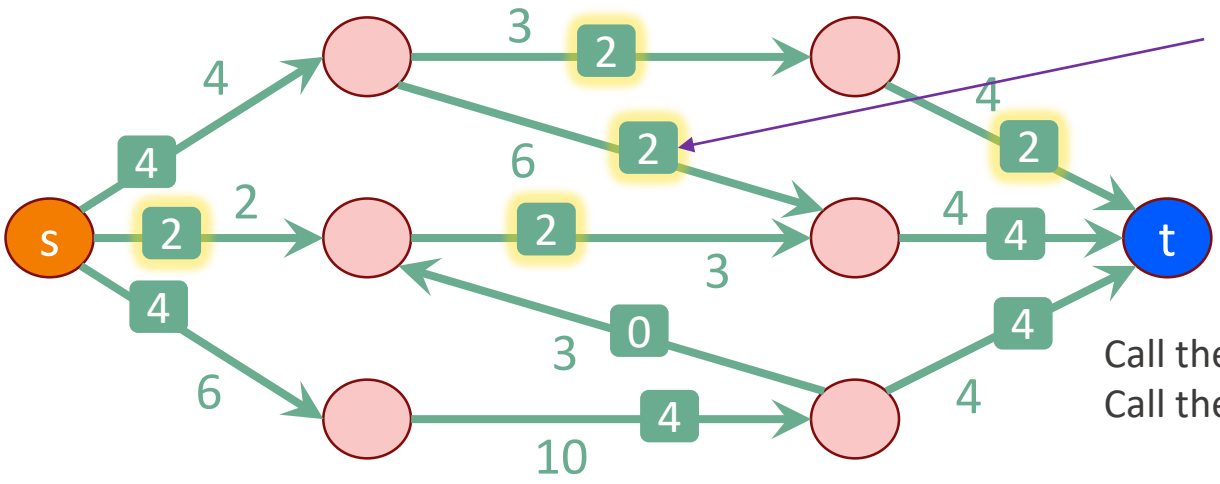
Call the flow f
Call the graph G

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



Call this graph G_f

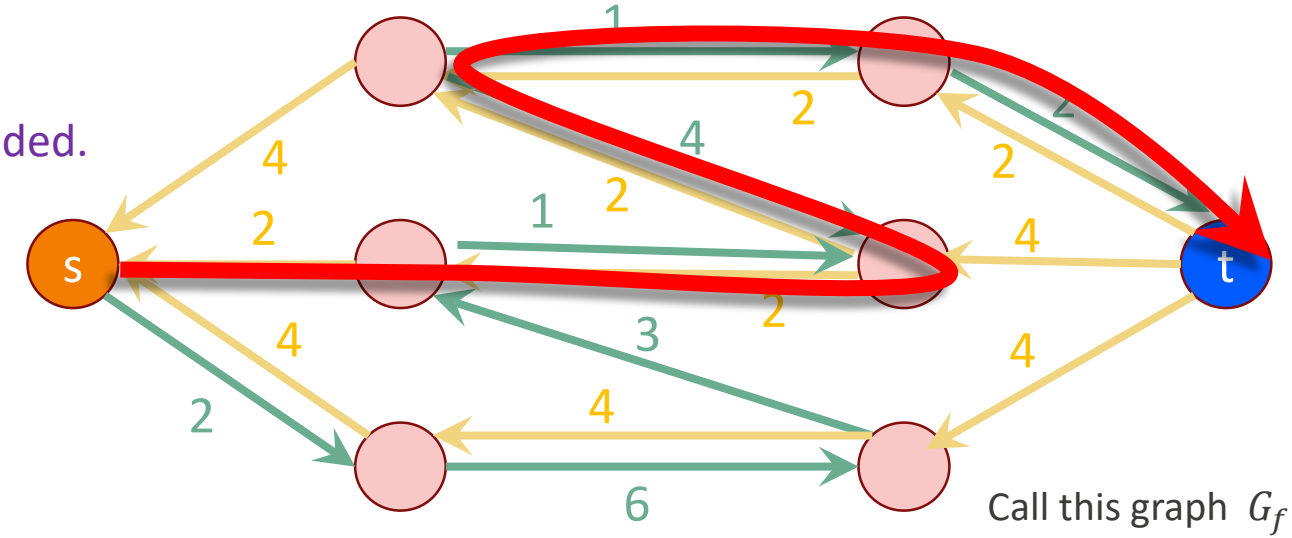
Example of Ford-Fulkerson



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

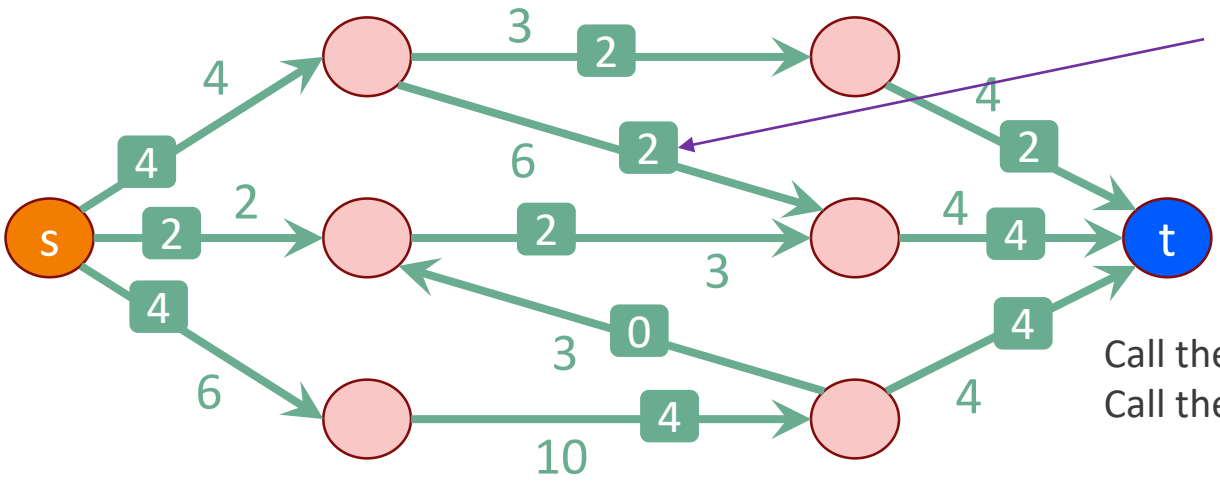
Call the flow f
Call the graph G

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



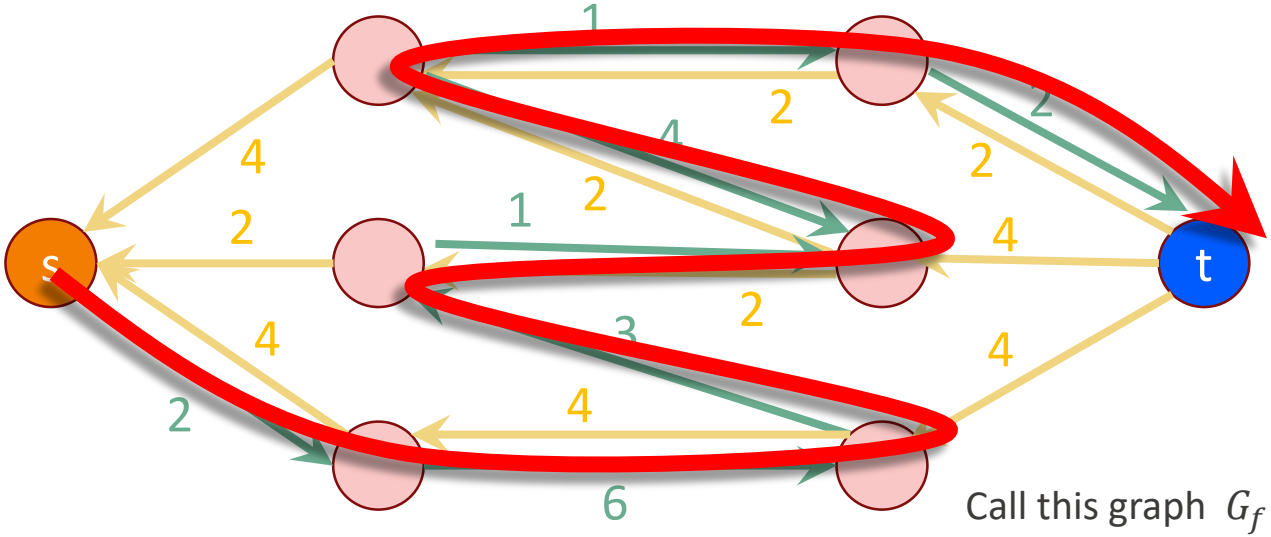
Call this graph G_f

Example of Ford-Fulkerson



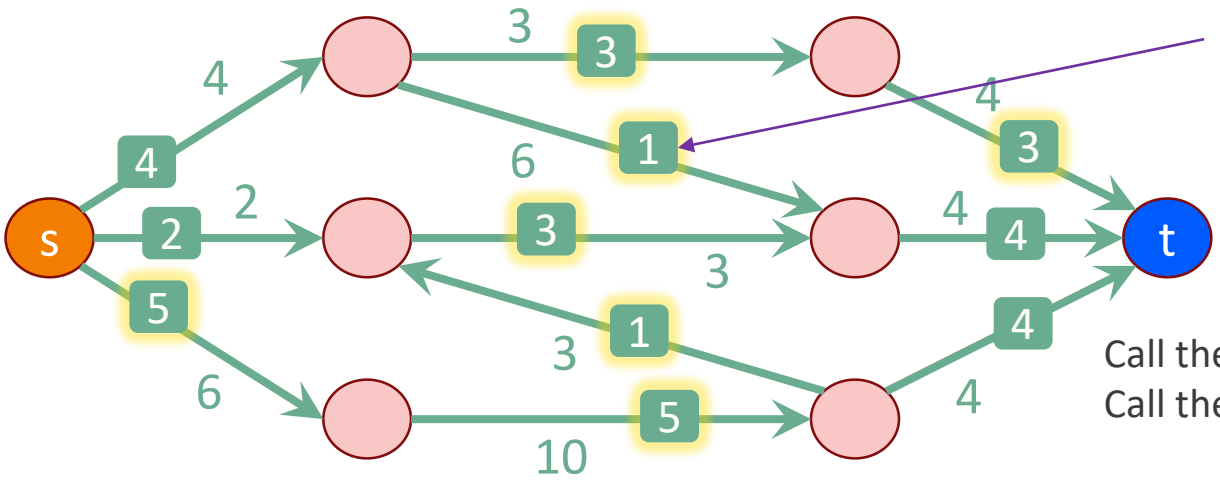
We will **remove** flow from this edge **AGAIN.**

Call the flow f
Call the graph G



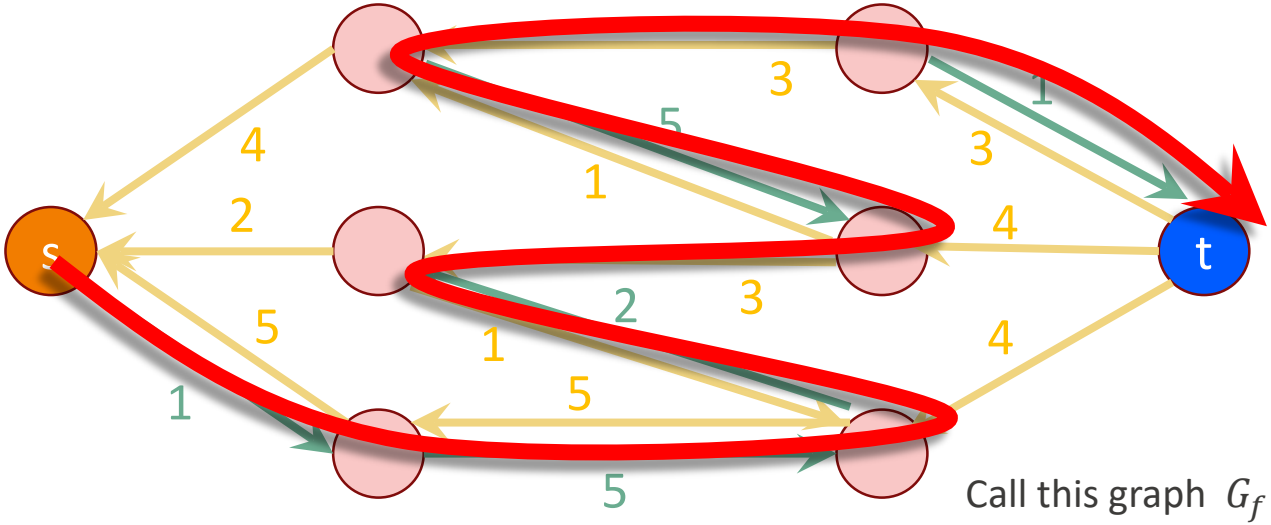
Call this graph G_f

Example of Ford-Fulkerson



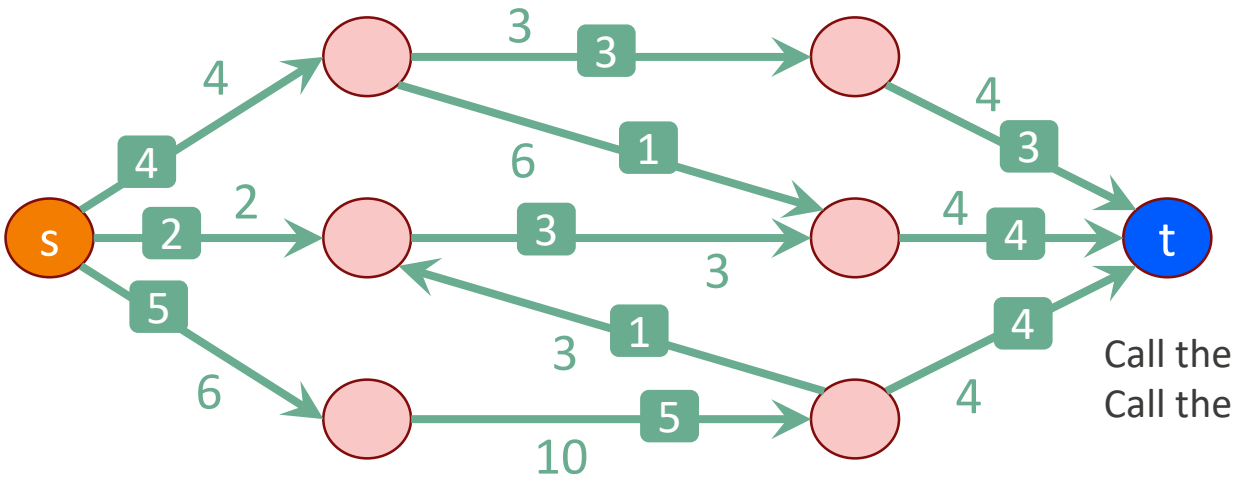
We will **remove** flow from this edge **AGAIN.**

Call the flow f
Call the graph G



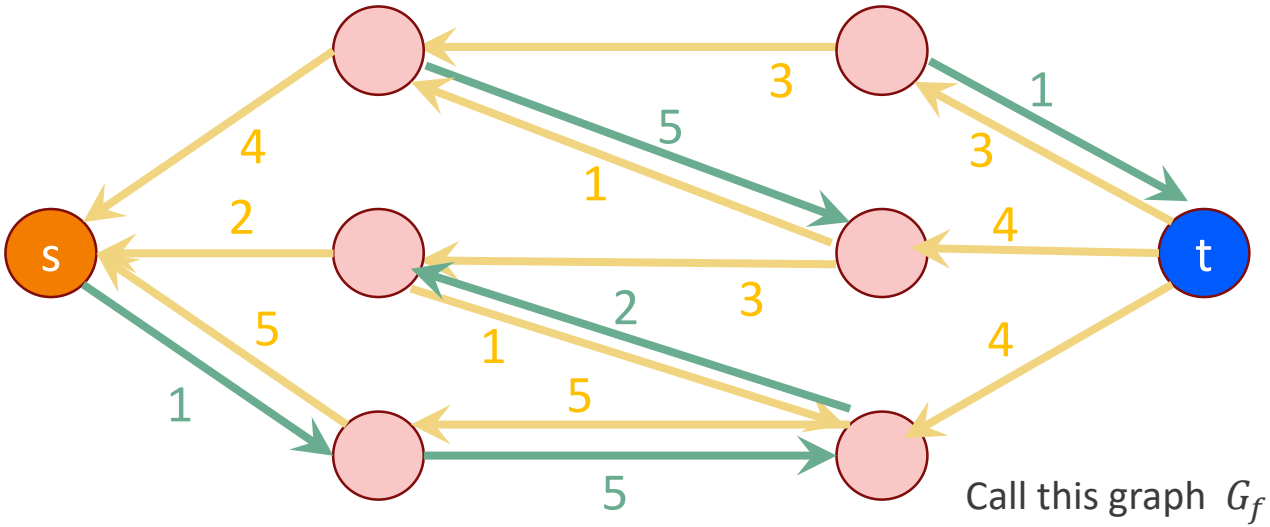
Call this graph G_f

Example of Ford-Fulkerson



Now we have nothing left to do!

Call the flow f
Call the graph G



Call this graph G_f

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?

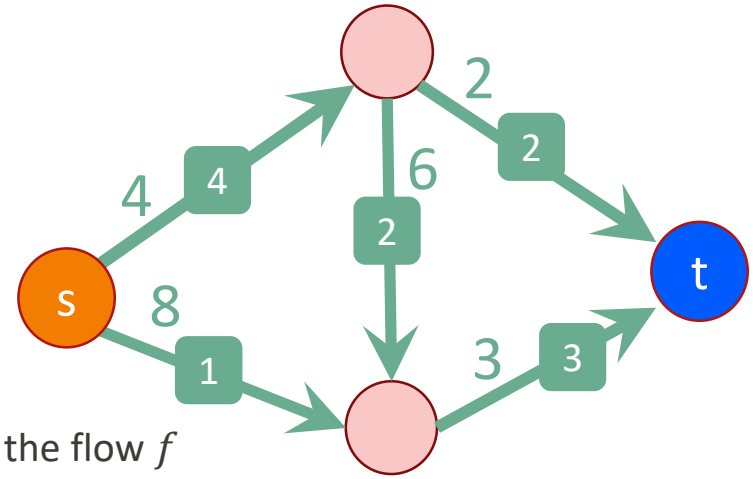
- **Lemma 2:** If there is no augmenting path in G_f then f is a maximum flow.

No augmenting path ⇒ max flow.

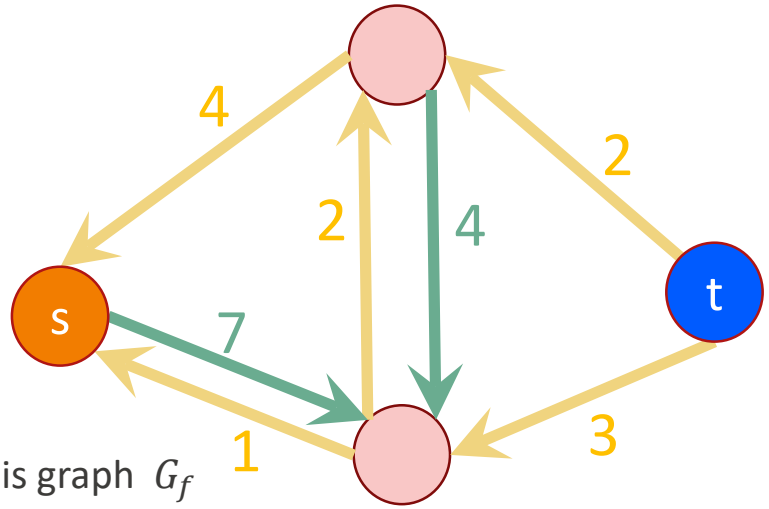
- Suppose there is not a path from s to t in G_f.
- Consider the cut given by:

t lives here

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



Call the flow f
Call the graph G



Call this graph G_f

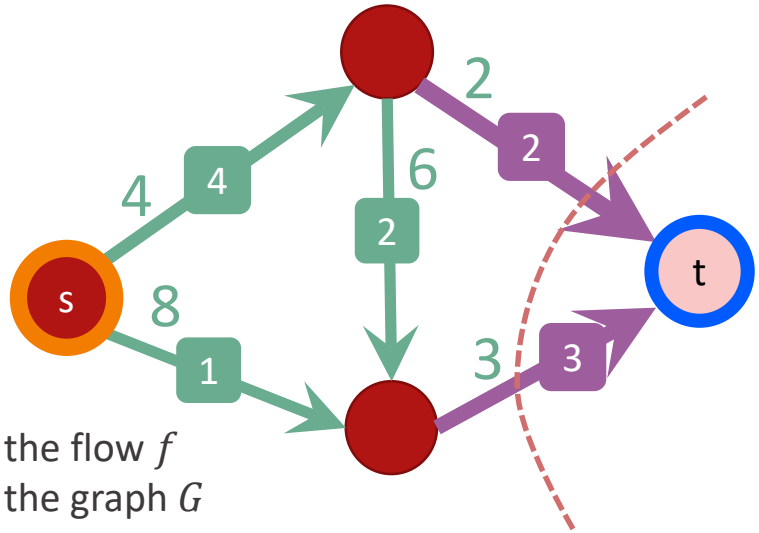
No augmenting path ⇒ max flow.

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

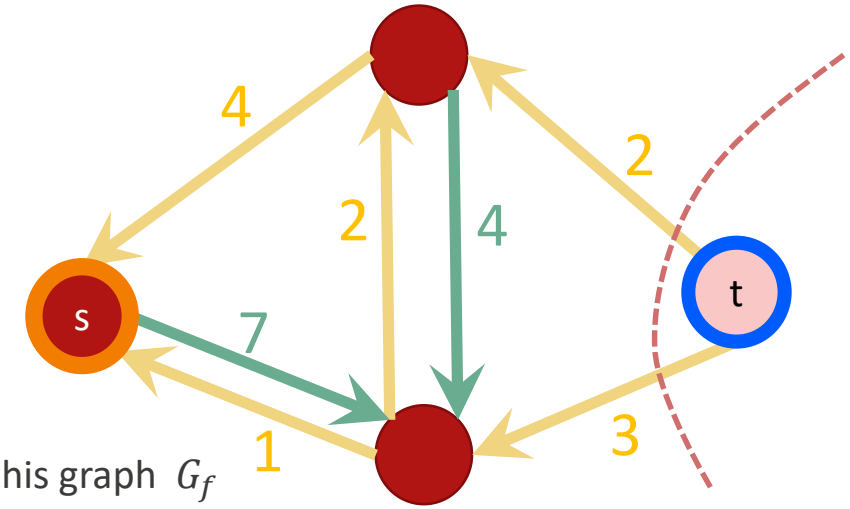
t lives here

{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:

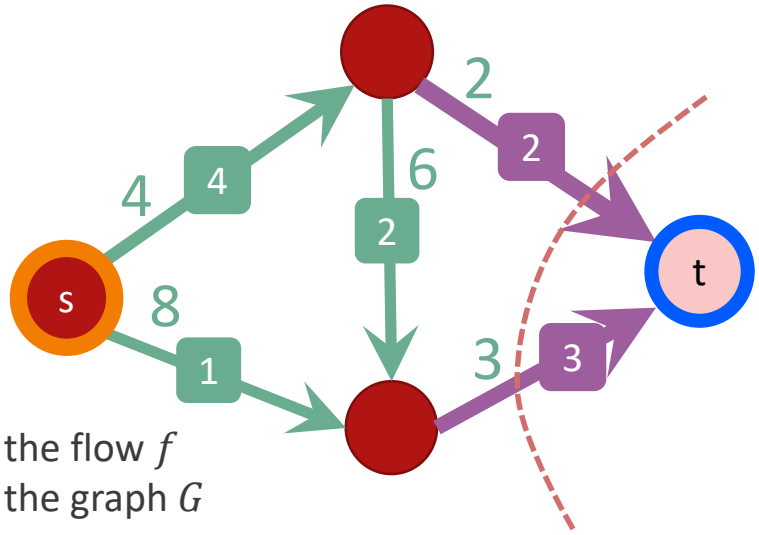
t lives here

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

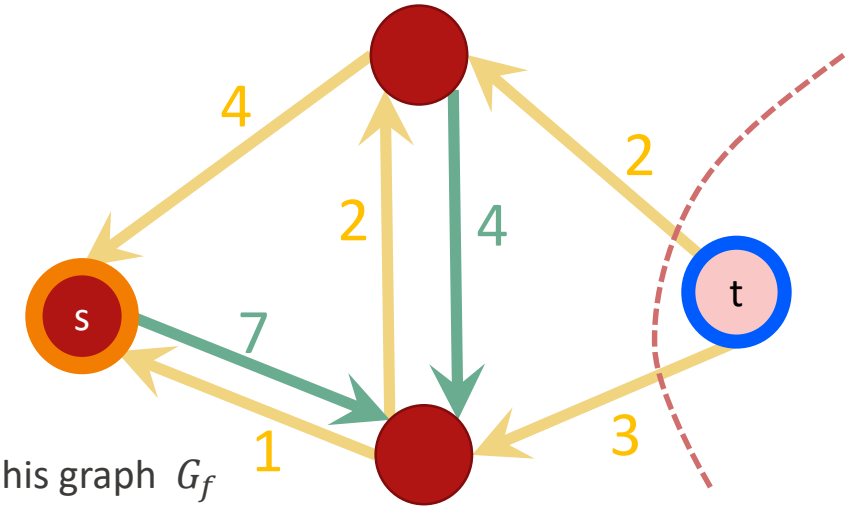
- The value of the flow f from s to t is **equal** to the cost of this cut.

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

Lemma 1



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

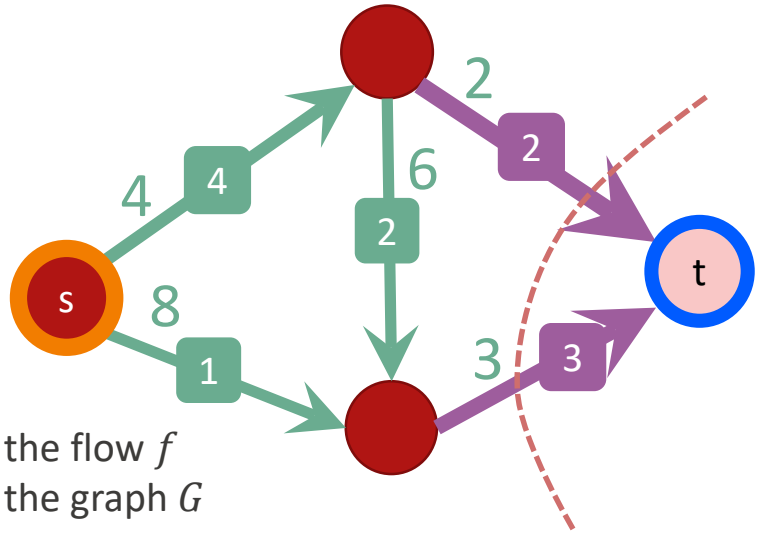
{things reachable from s} , {things not reachable from s}

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

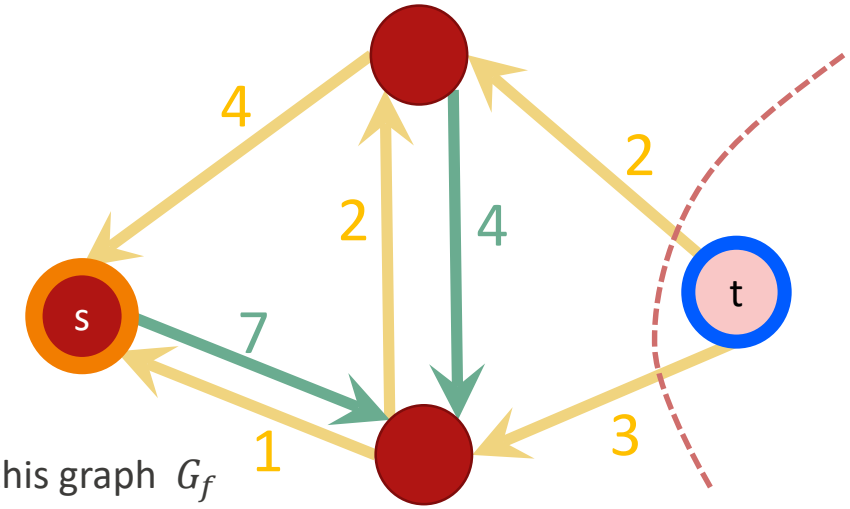
Value of f = cost of this cut $\geq$ min cut $\geq$ max flow

Lemma 1

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



Call the flow f
Call the graph G

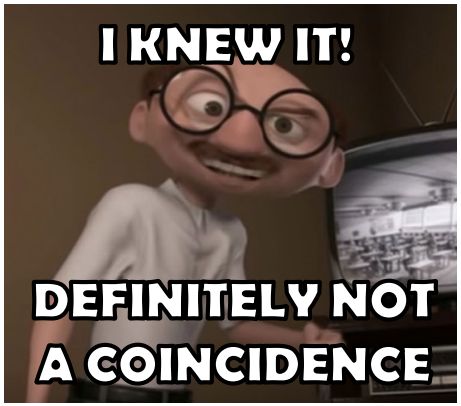


Call this graph G_f

Min-Cut Max-Flow Theorem

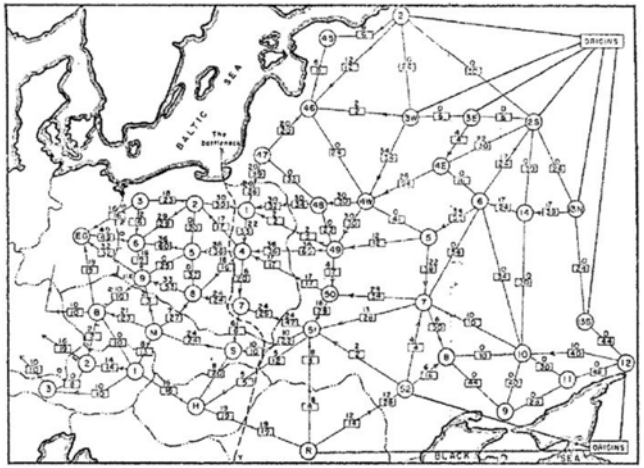
$$\textbf{max flow} \geq \textbf{Value of } f \textbf{ = cost of this cut} \geq \textbf{min cut} \geq \textbf{max flow}$$

So everything is equal and min cut = max flow!



What have we learned?

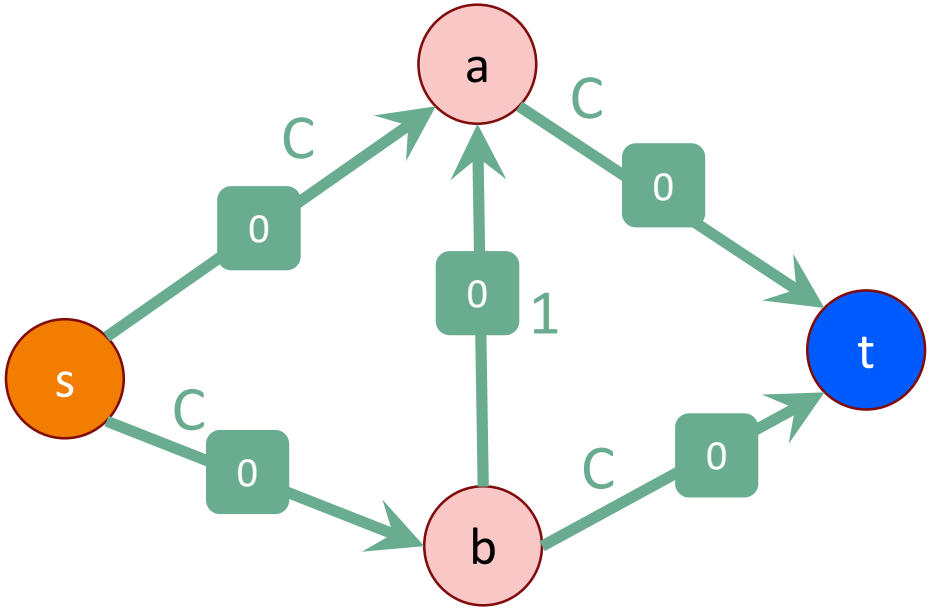
- Max s-t flow is equal to min s-t cut!
  - The Soviet Union and the USA were trying to solve the same problem…
- The Ford-Fulkerson algorithm can find the min-cut/max-flow.
  - Repeatedly improve your flow along an augmenting path.



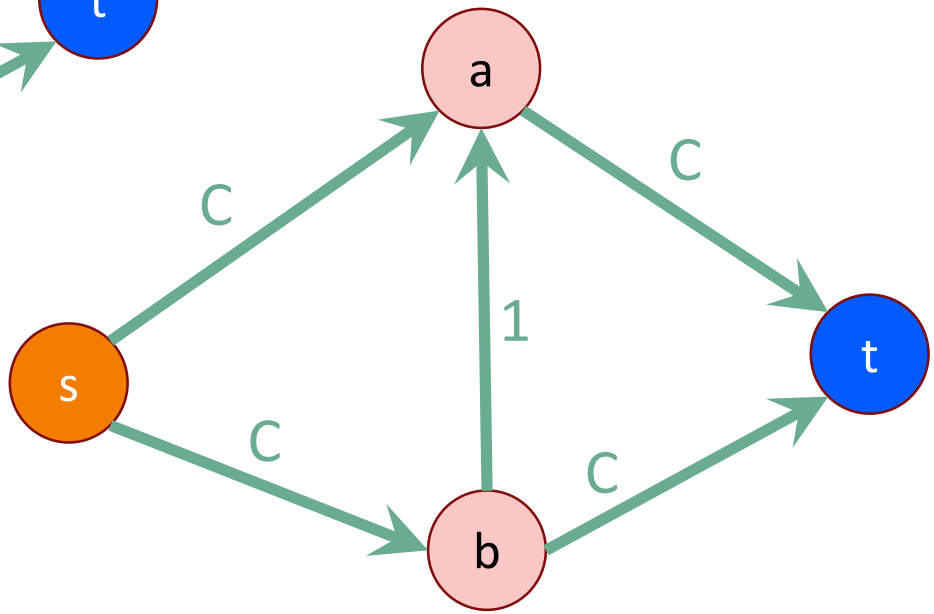
- **How long does this take?**

Why should we be concerned?

- Suppose we just picked paths arbitrarily.

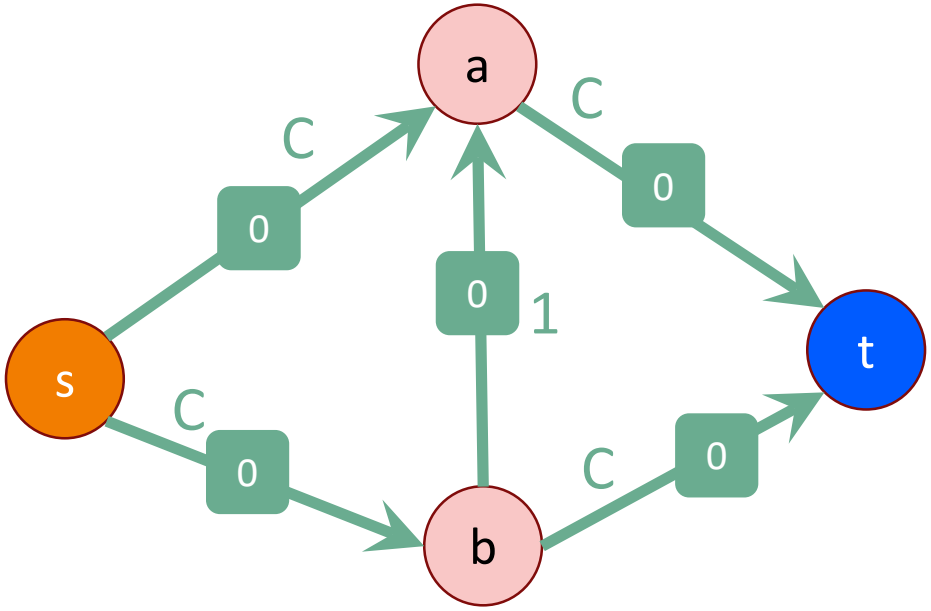


Choose a really big number C
like 1,000,000,000,000.

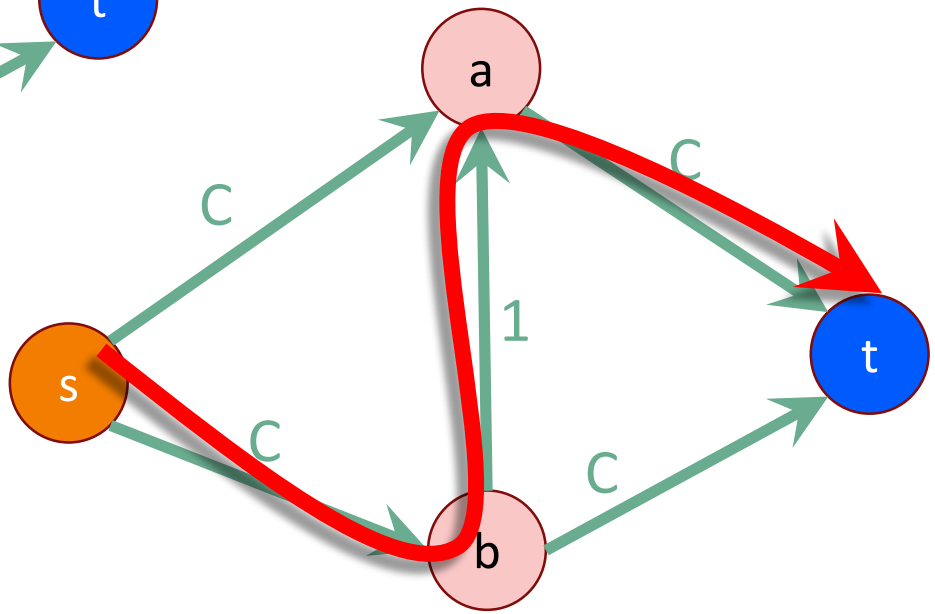


Why should we be concerned?

- Suppose we just picked paths arbitrarily.

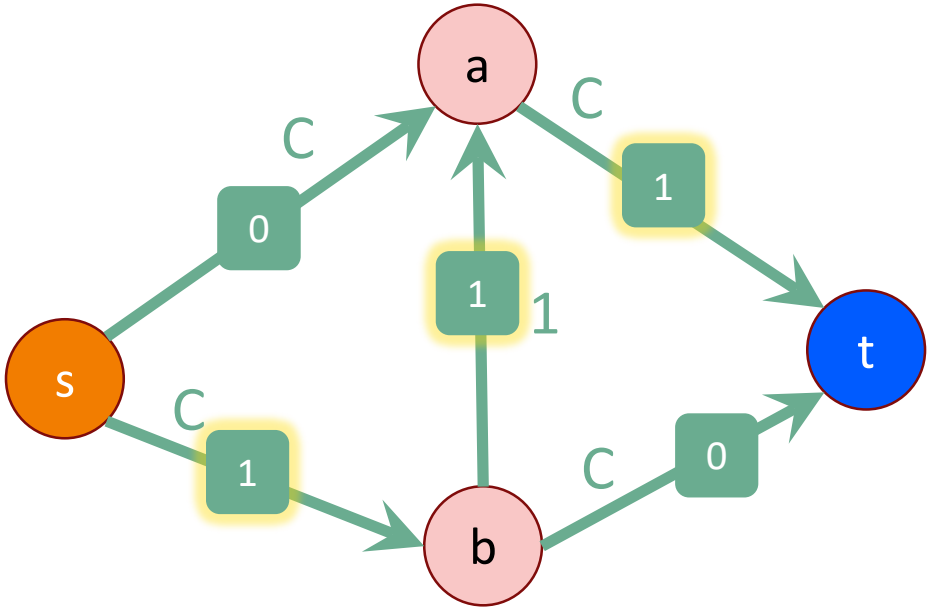


Choose a really big number C
like 1,000,000,000,000.



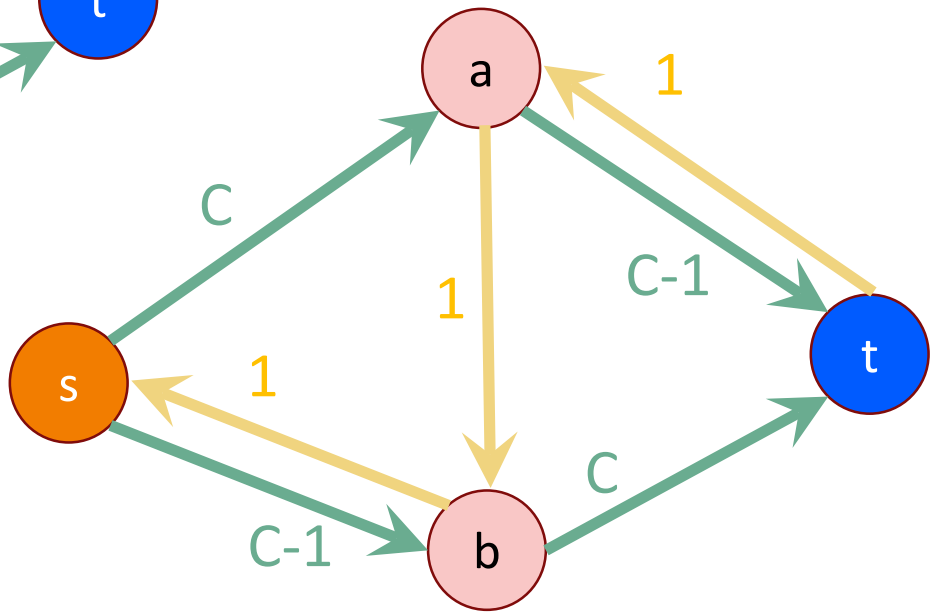
Why should we be concerned?

- Suppose we just picked paths arbitrarily.



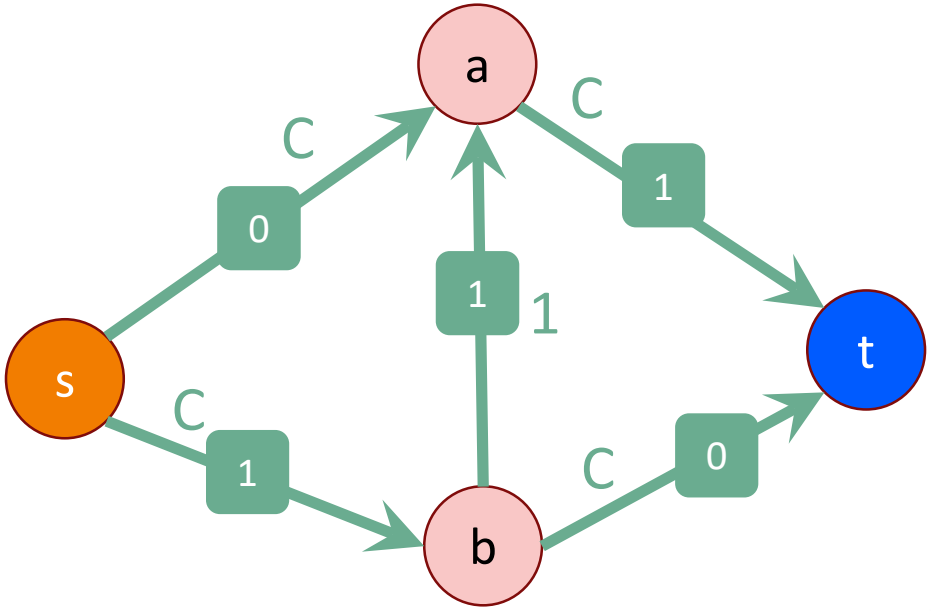
Choose a really big number C
like 1,000,000,000,000.

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

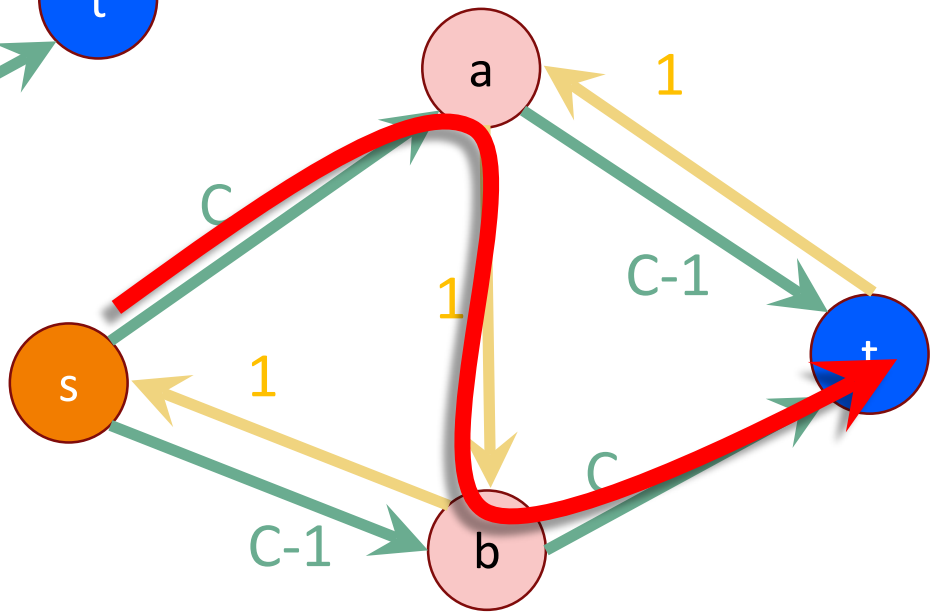


Why should we be concerned?

- Suppose we just picked paths arbitrarily.

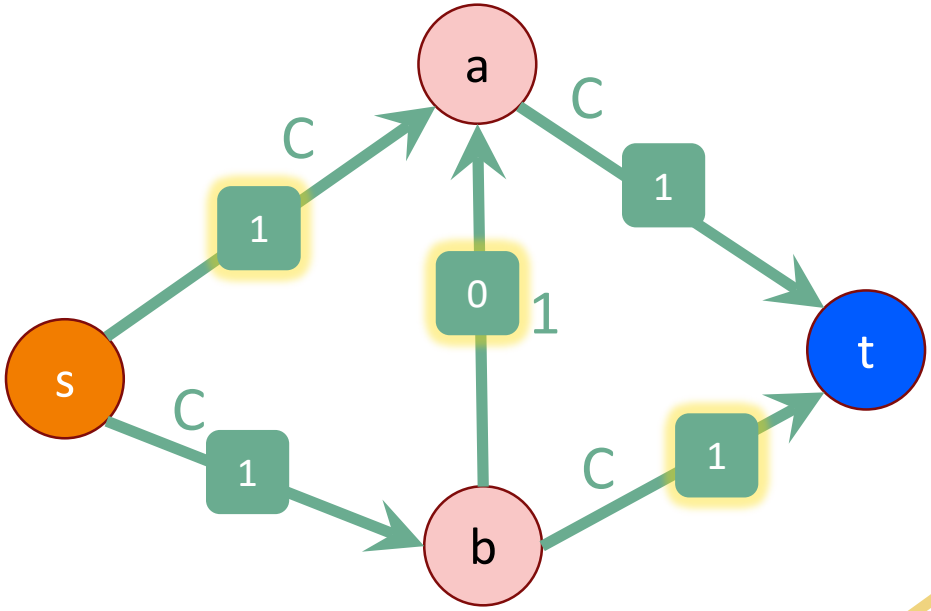


Choose a really big number C
like 1,000,000,000,000.



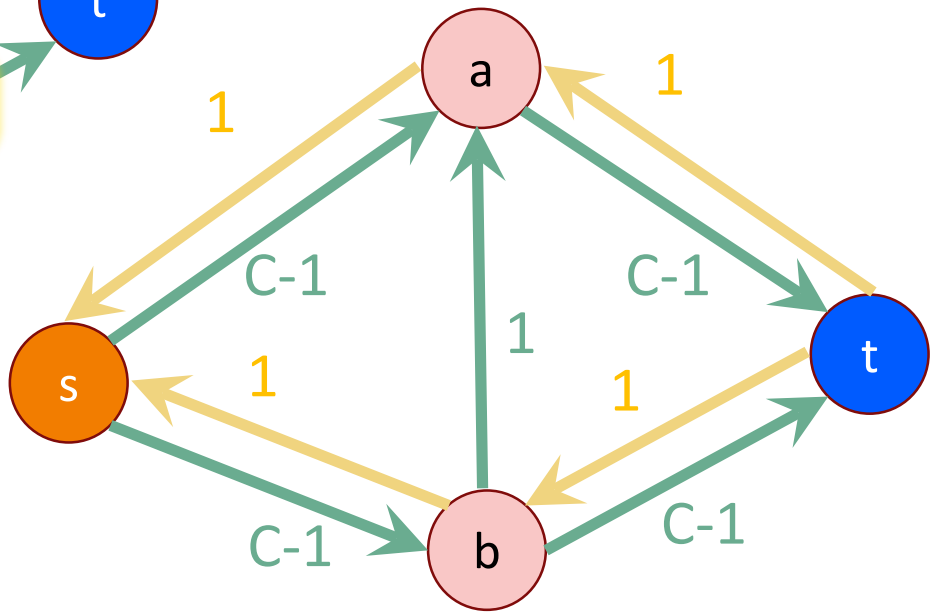
Why should we be concerned?

- Suppose we just picked paths arbitrarily.



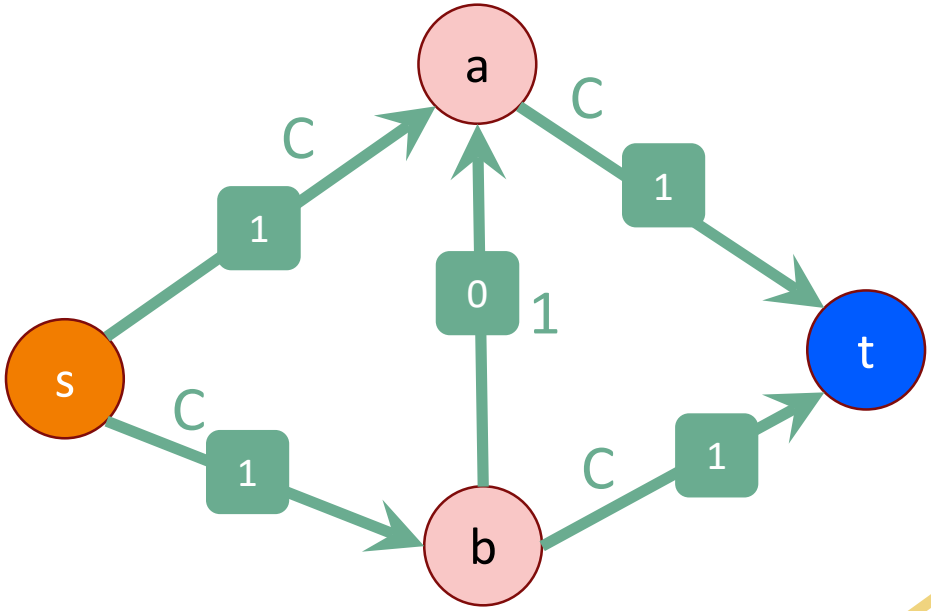
Choose a really big number C
like 1,000,000,000,000.

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

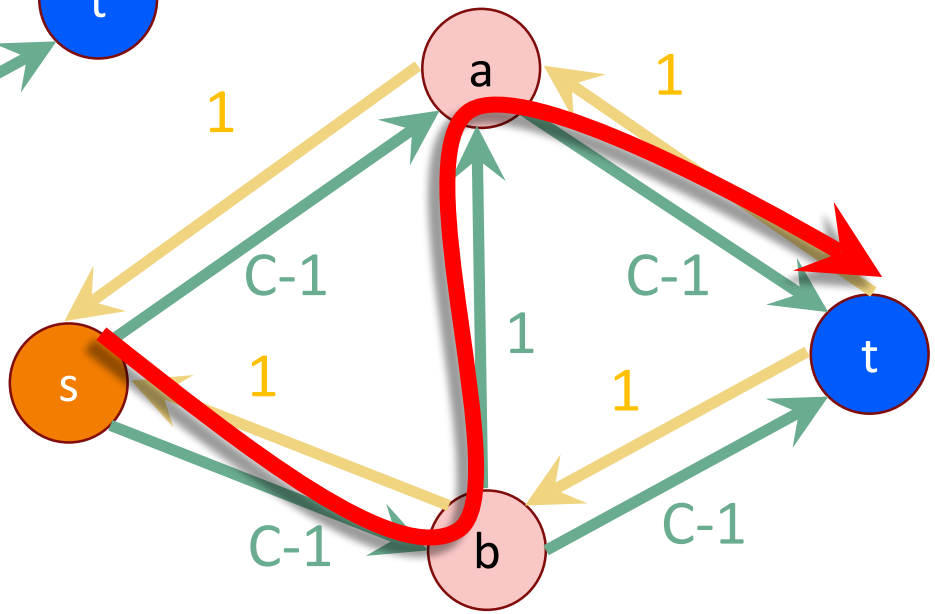


Why should we be concerned?

- Suppose we just picked paths arbitrarily.

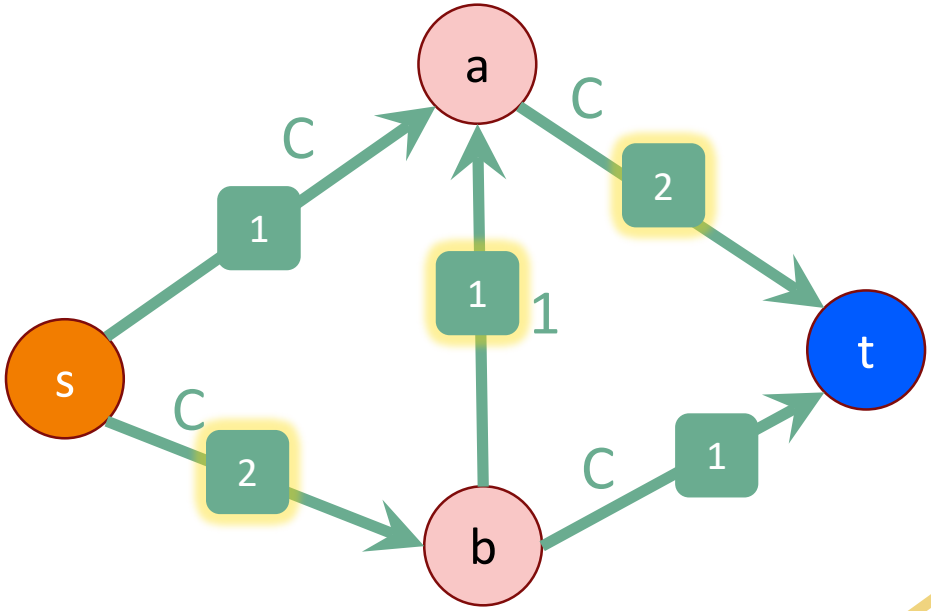


Choose a really big number C
like 1,000,000,000,000.



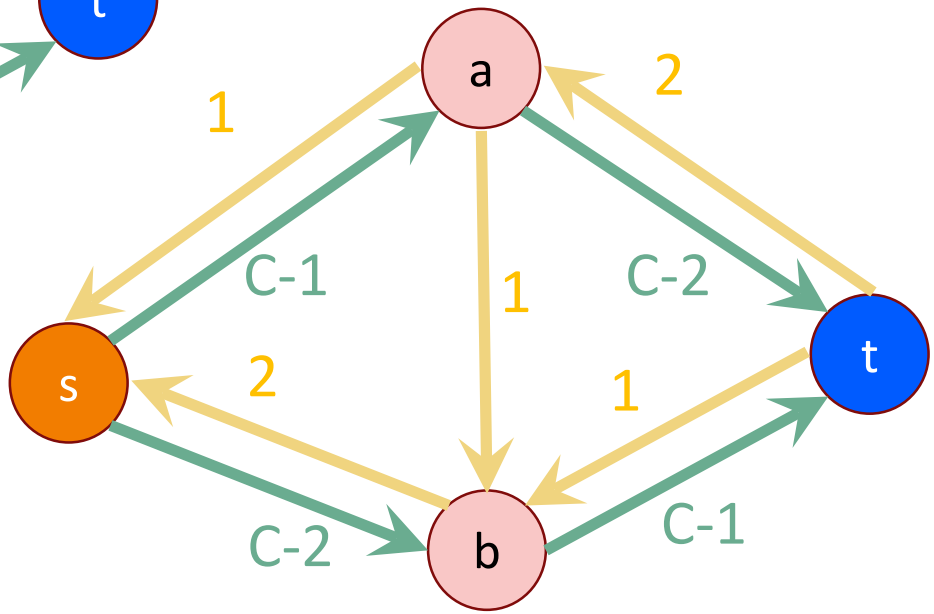
Why should we be concerned?

- Suppose we just picked paths arbitrarily.



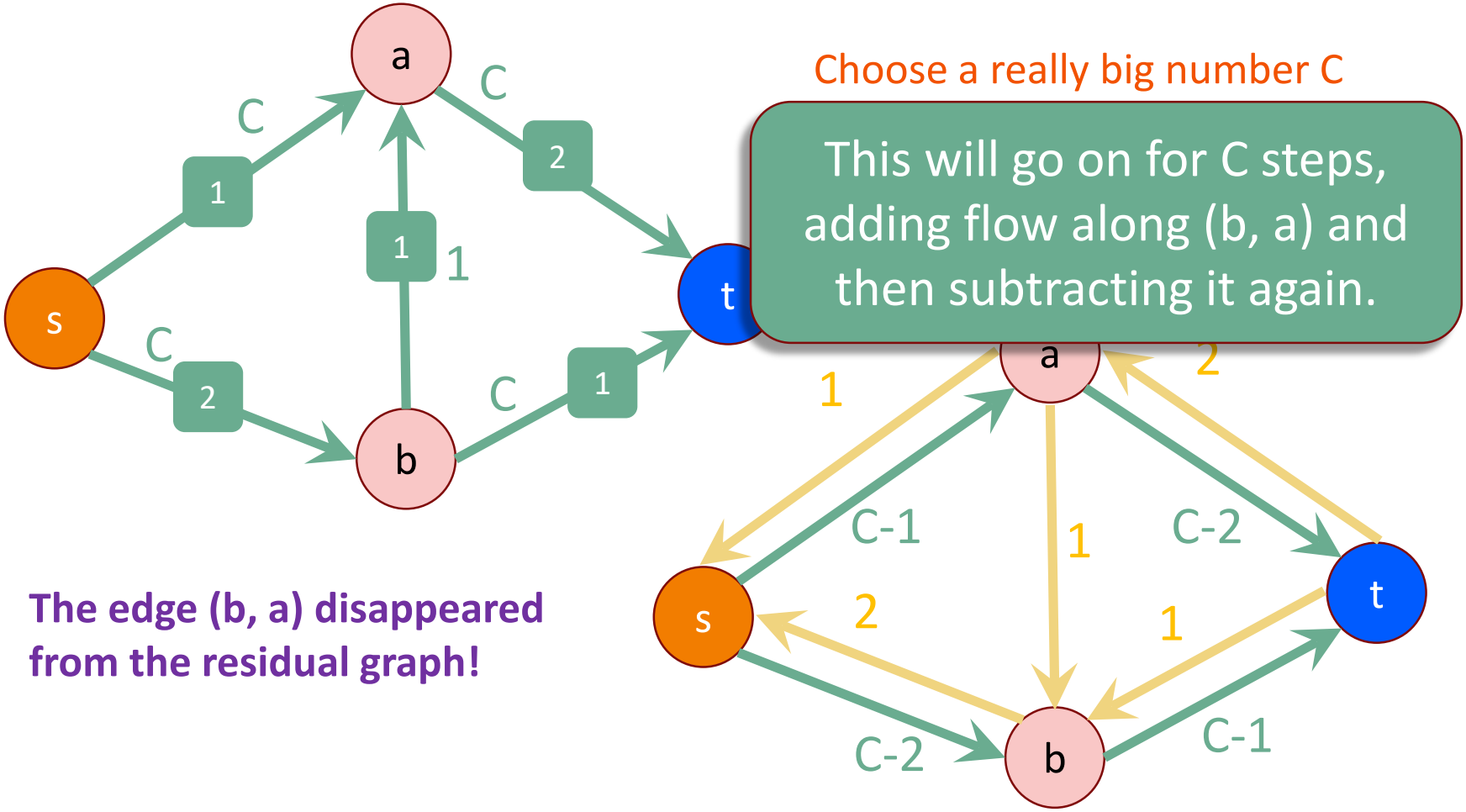
Choose a really big number C
like 1,000,000,000,000.

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



Why should we be concerned?

- Suppose we just picked paths arbitrarily.



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

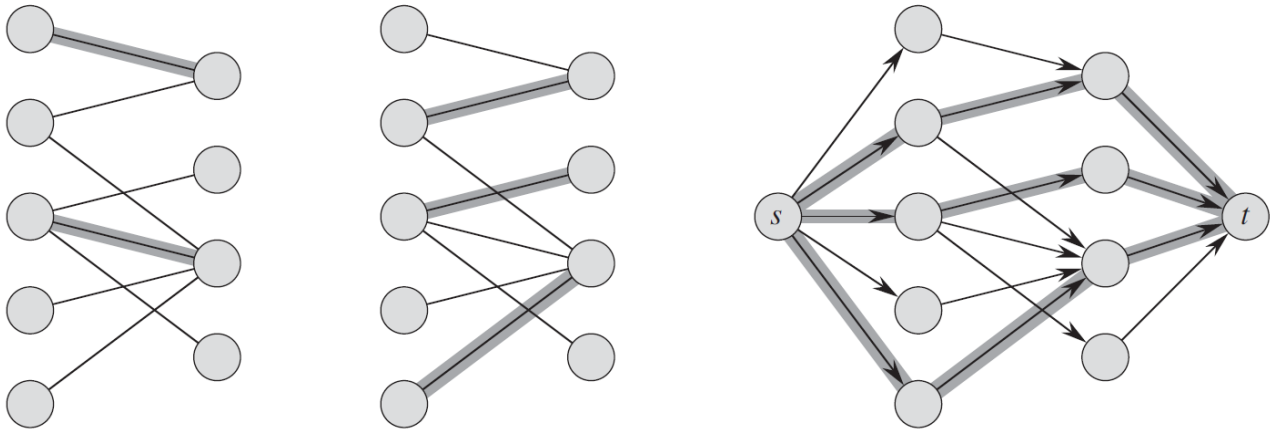
How do we choose which paths to use?

- 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 $O(nm^2)$
  - Read CLRS 26.4 for further study.

There's more!

- Min-cut and max-flow are not just useful for the USA and the Soviet Union in 1955.
- The Ford-Fulkerson algorithm is the basis for many other graph algorithms.
  - Maximum bipartite matching
  - Integer assignment problems



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., maximum bipartite matching, assignment problems.


Any Question?