Advanced Algorithmic Techniques (COMP523)

Network Flows

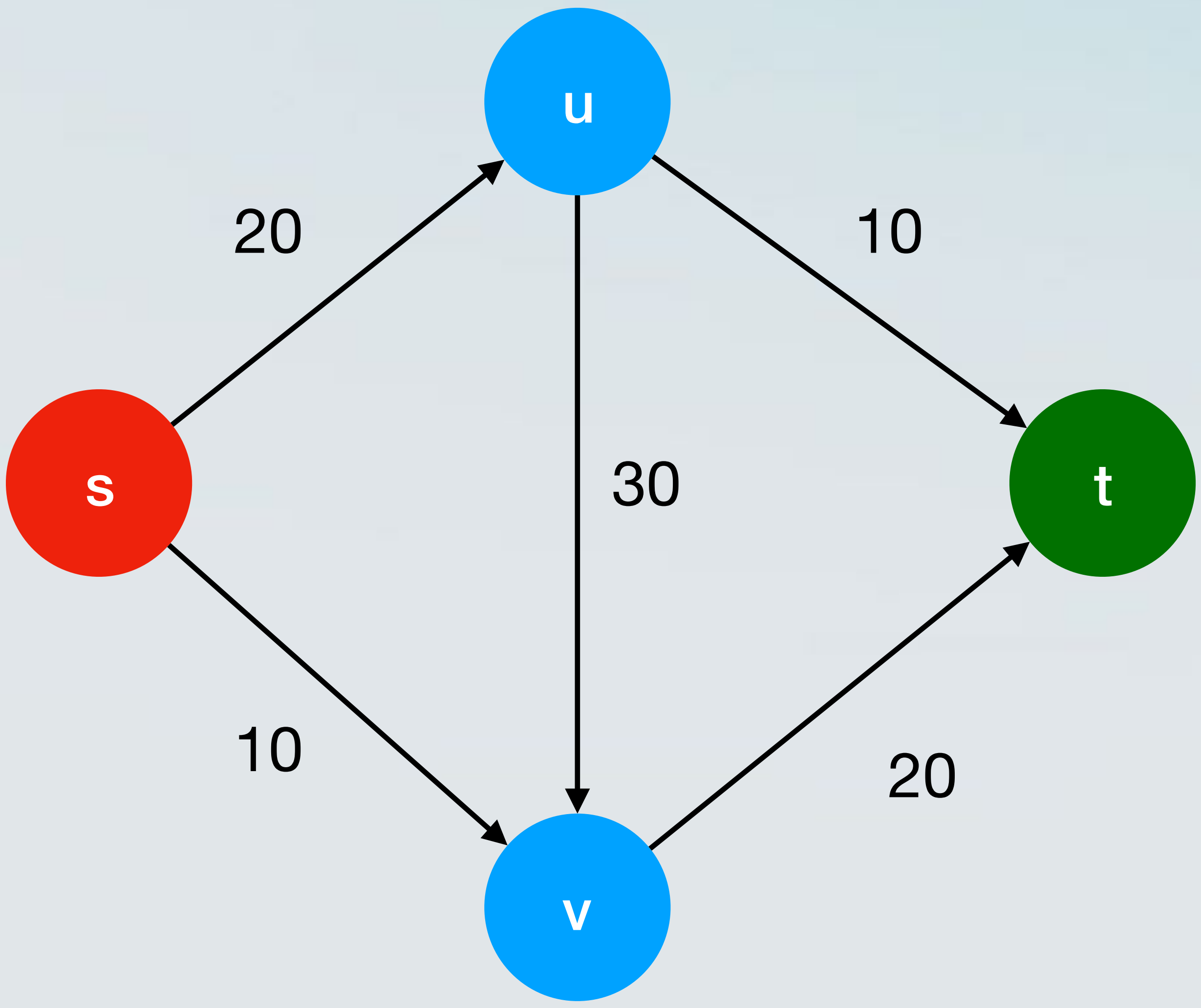
Recap and plan

- **Last 2 lectures:**
  - Dynamic Programming
  - Weighted Interval Scheduling, Subset Sum, Knapsack
- **This lecture:**
  - Network Flows
  - Maximum Flow
  - The Ford-Fulkerson Algorithm
  - Max-Flow - Min-Cut

Flow Networks

- A flow network is a directed graph $G=(V, E)$ with the following properties:
  - Each edge e in E has a nonnegative *capacity* c_e.
  - There is a single *source* node s in V.
  - There is a single *sink* node t in V.
  - All other nodes in $V - \{s, t\}$ are called *internal* nodes.

Example



Flow Networks

- Further assumptions:
  - The *source* s does not have any incoming edges.
  - The *sink* t does not have any outgoing edges.
  - There is at least one edge incident to each node.
  - All the capacities are integer numbers.

Flow

- An (s-t) flow is a function $f: E \to \mathbf{R}^+$, mapping each edge e to a nonnegative real number $f(e)$.

- A (feasible) flow must satisfy the following two properties:

  - (Capacity) For each e in E, we have $0 \leq f(e) \leq c_e$

  - (Flow Conservation) For each node v in V - {s, t}, we have that

$$\sum_{e \textbf{ into } v} f(e) = \sum_{e \textbf{ out of } v} f(e)$$

Flow

- The source s generates flow.
- The source t absorbs flow.
- Value of flow f, denoted val(f):
  - Total flow out of s.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

- Generally, define $f^{out}(v)$ and $f^{in}(v)$ for the flow going out of (resp. going into) node v.
- Similarly, define $f^{out}(S)$ and $f^{in}(S)$ for sets of nodes S.

The maximum flow problem

The maximum flow problem

Given a flow network G, find a flow of maximum possible value.

The maximum flow problem

Given a flow network G, find a flow of maximum possible value.

The maximum flow problem

Given a flow network G, find a flow of maximum possible value.

The maximum flow problem

Given a flow network G, find a flow of maximum possible value.

Let's try to design an algorithm for that.

An algorithm for max-flow

An algorithm for max-flow

- Let’s start with a feasible solution.

An algorithm for max-flow

- Let’s start with a feasible solution.
  - Any suggestions?

An algorithm for max-flow

- Let's start with a feasible solution.
  - Any suggestions?
  - $f(e) = 0$ for all e in E.

An algorithm for max-flow

- Let's start with a feasible solution.
  - Any suggestions?
  - $f(e) = 0$ for all e in E.
  - This is a feasible flow. But not very good.

An algorithm for max-flow

- Let’s start with a feasible solution.
  - Any suggestions?
  - $f(e) = 0$ for all e in E.
  - This is a feasible flow. But not very good.
  - Let’s try to increase it.

Increasing the flow

Increasing the flow

- The flow originates from s and goes to t.

Increasing the flow

- The flow originates from s and goes to t.
- So we have to find an (s-t) path, and route it via this path.

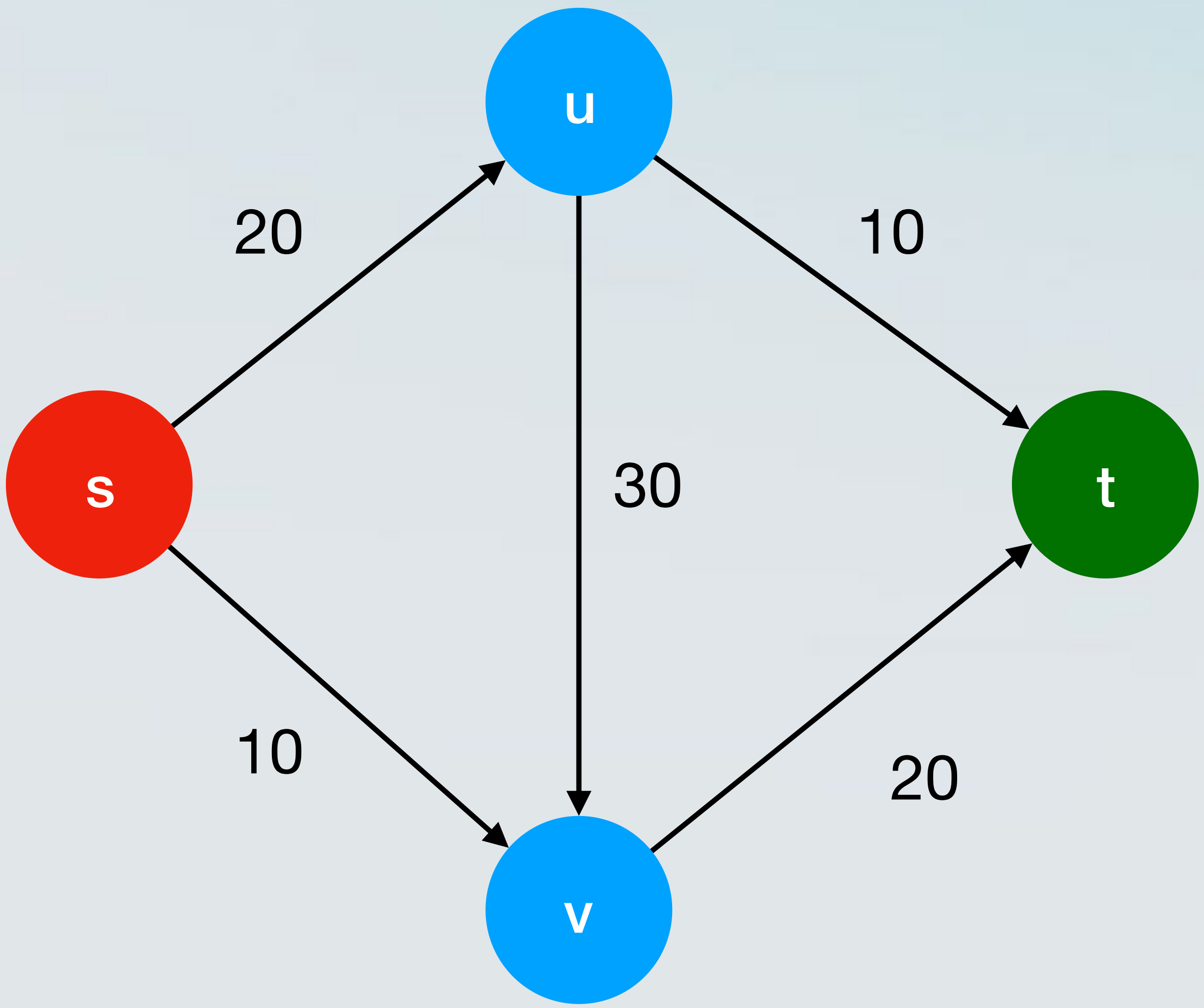
Increasing the flow

- The flow originates from s and goes to t.
- So we have to find an (s-t) path, and route it via this path.
- How much flow are we allowed to route through this path?

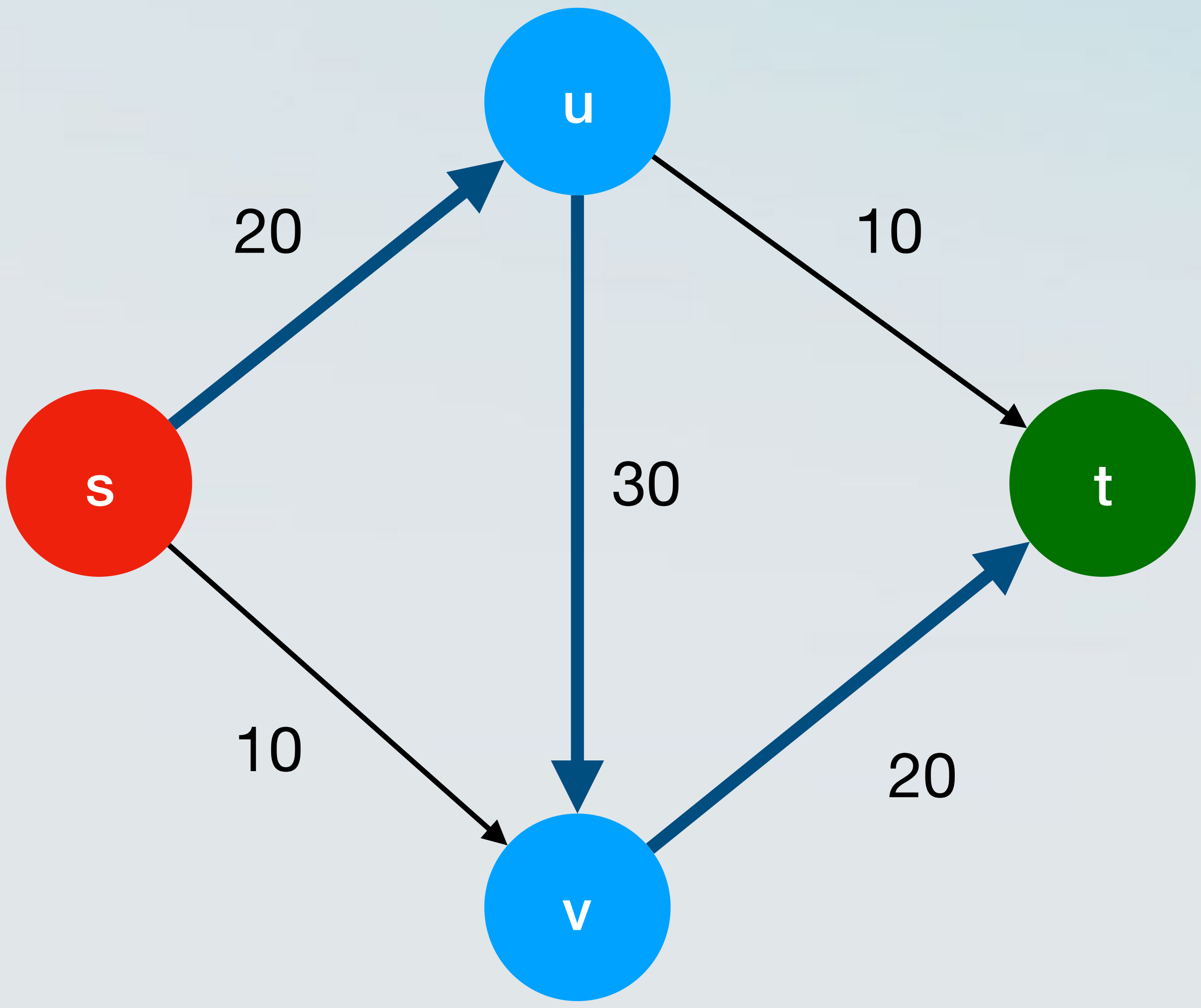
Increasing the flow

- The flow originates from s and goes to t.
- So we have to find an (s-t) path, and route it via this path.
- How much flow are we allowed to route through this path?
    - As much as the smallest capacity c_e of any edge e on the path.

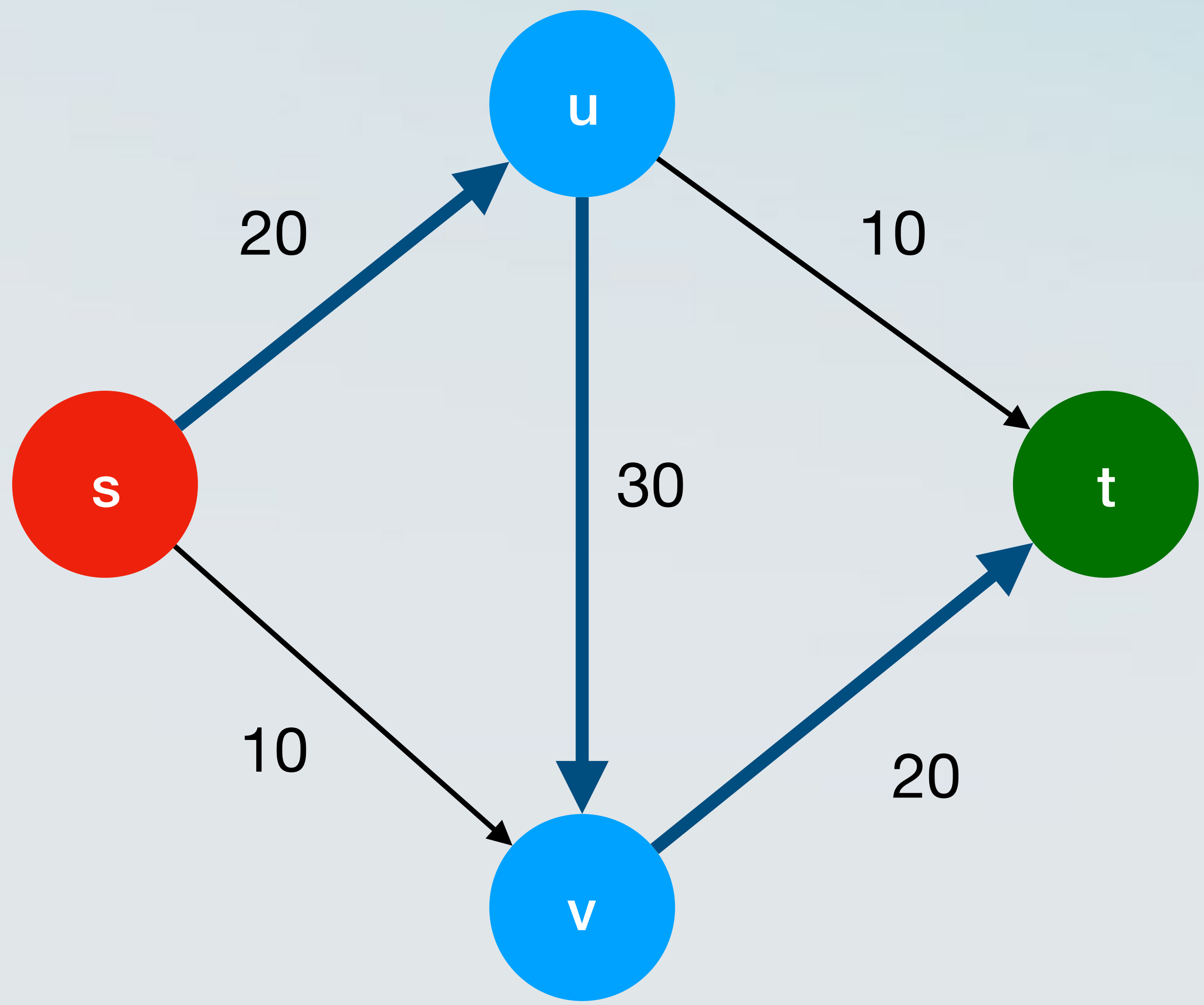
Example



Example

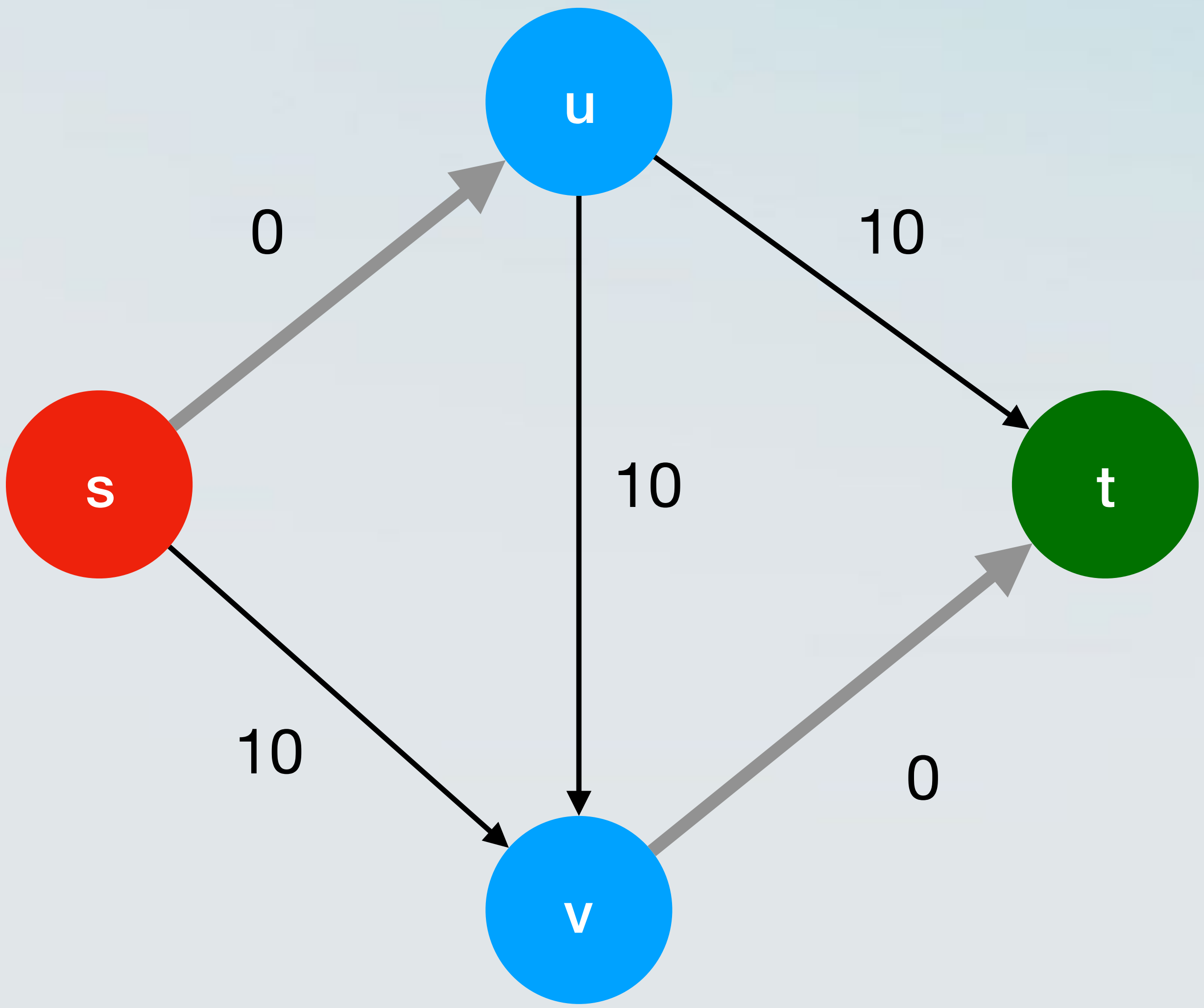


Example

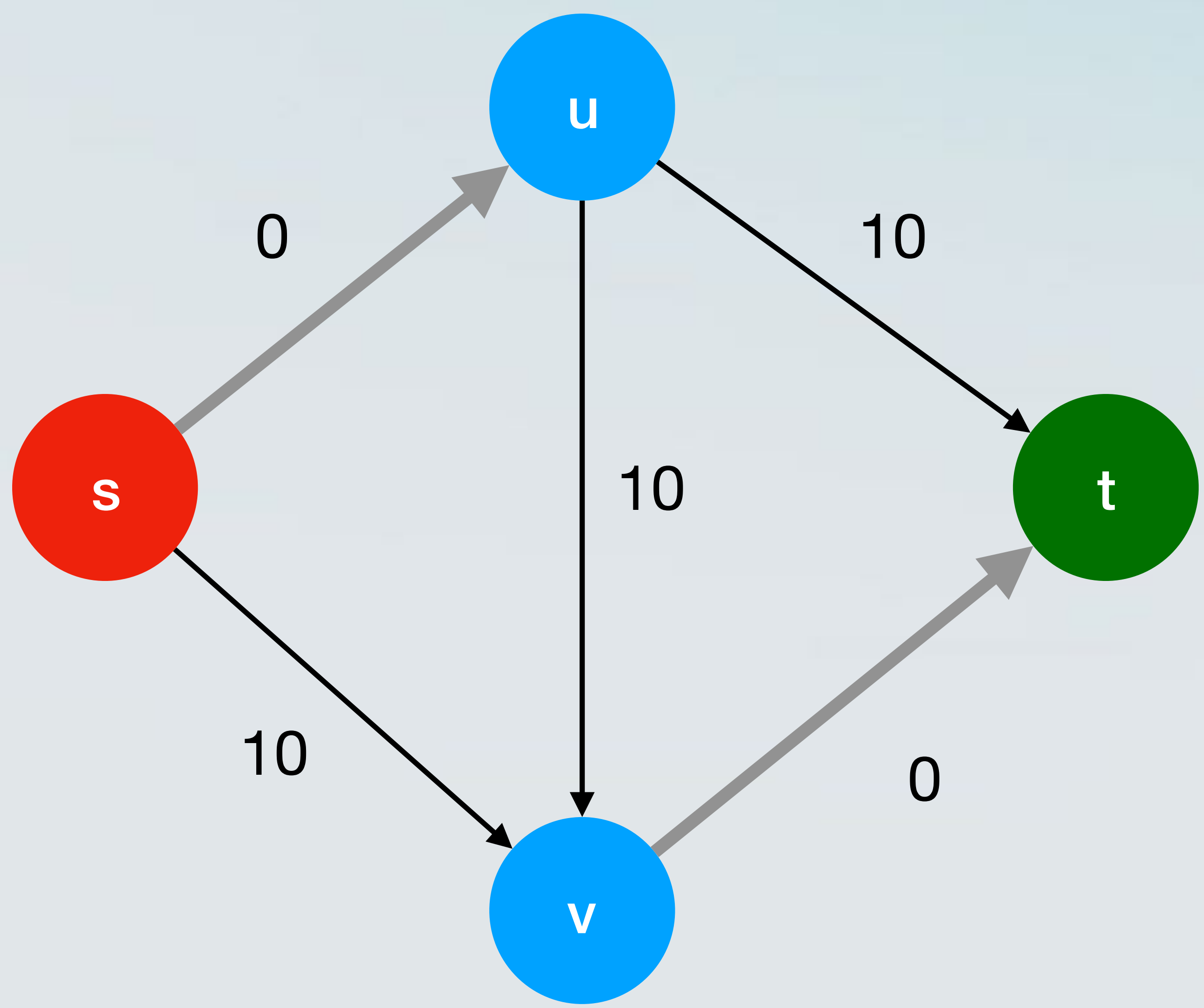


Is 20 the maximum flow?

Example

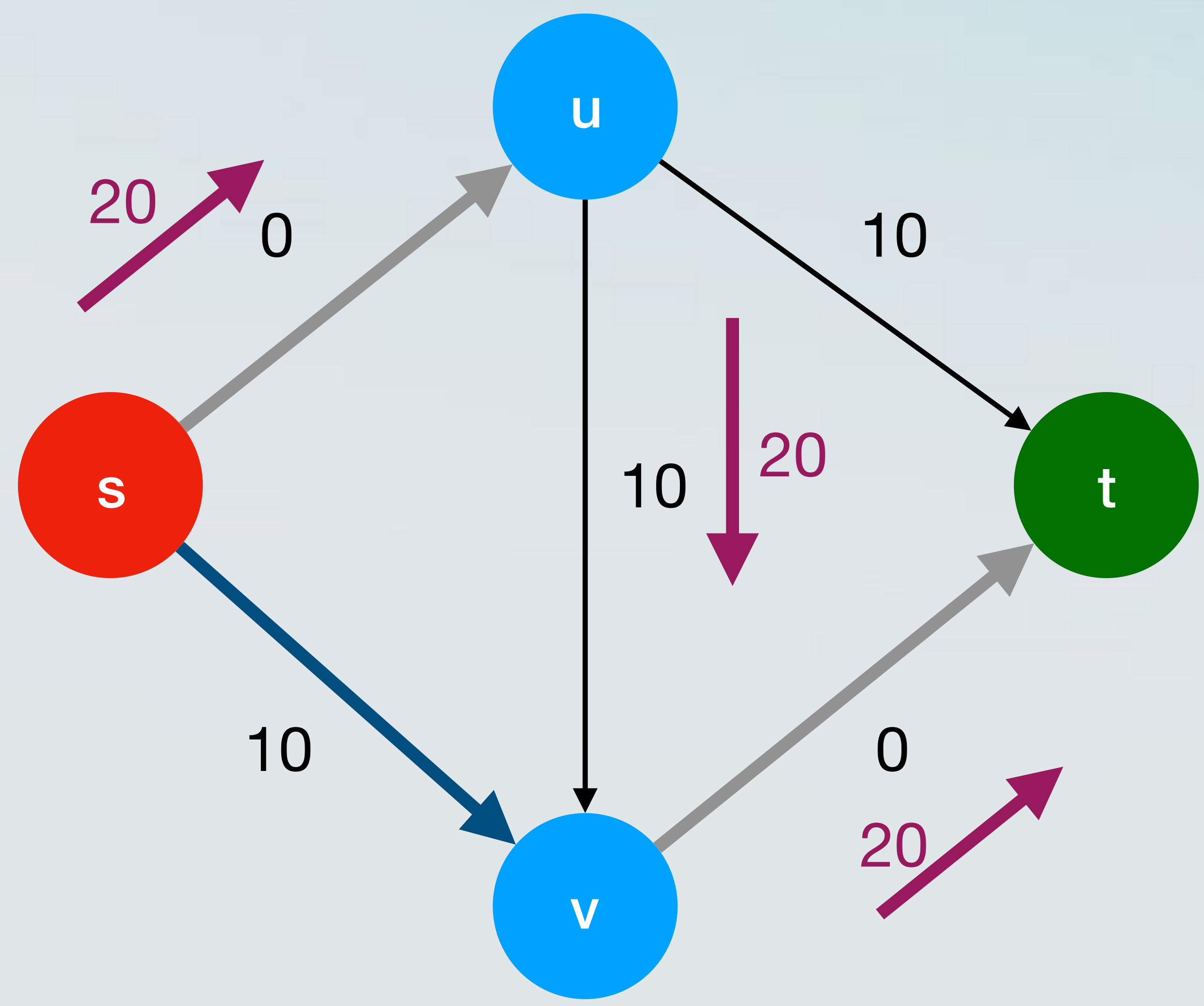


Example

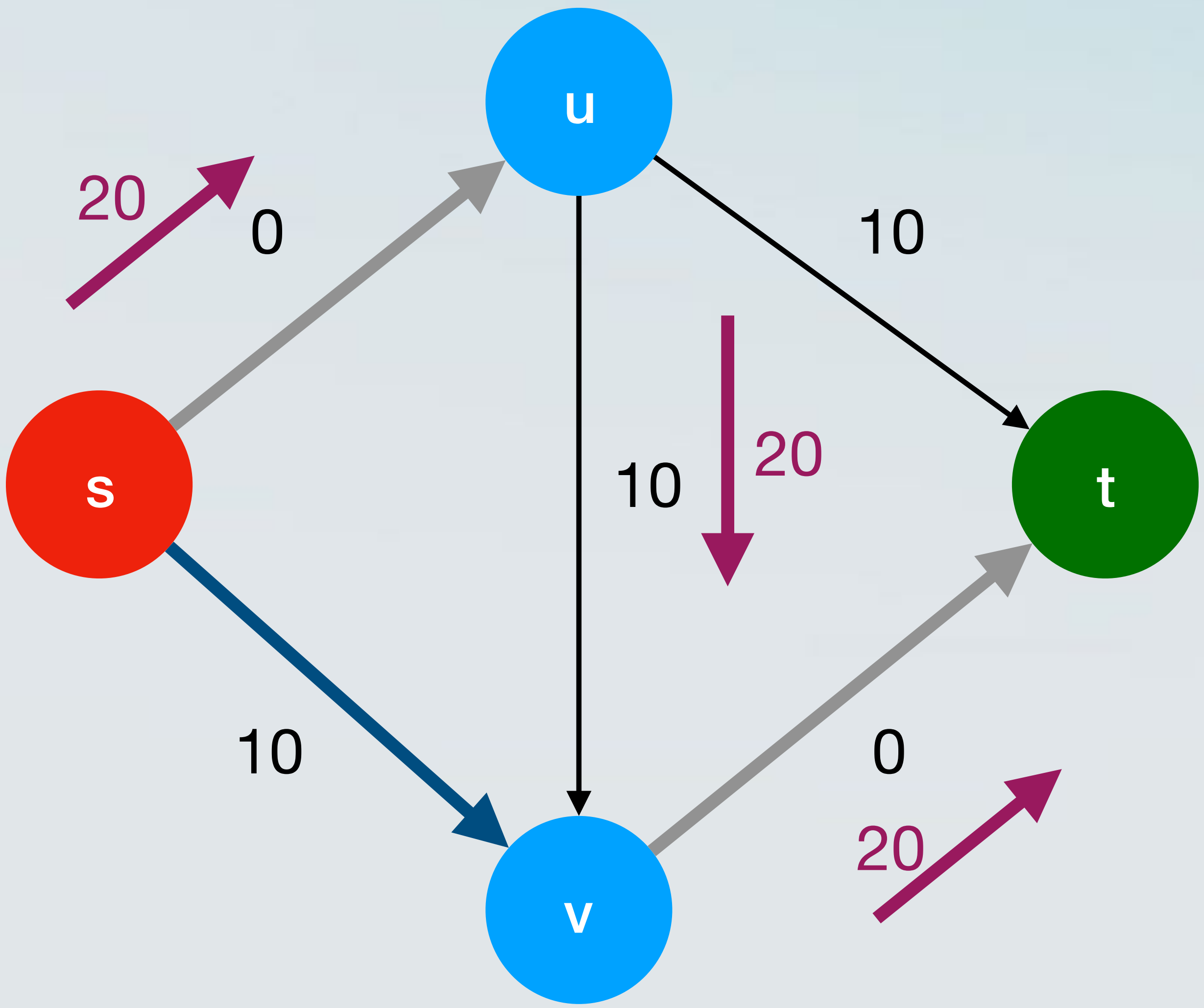


We are stuck!

What we would like to do

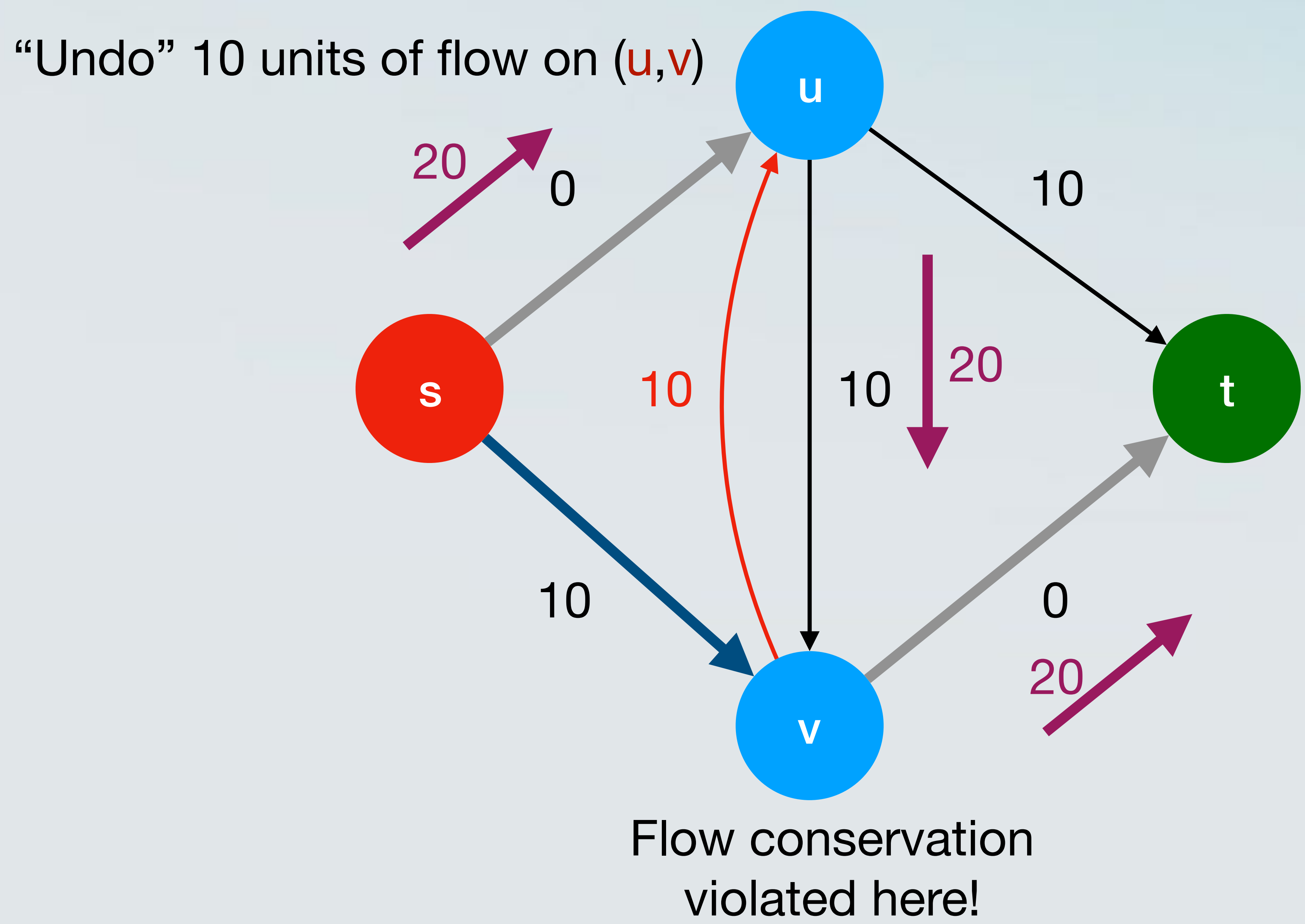


What we would like to do

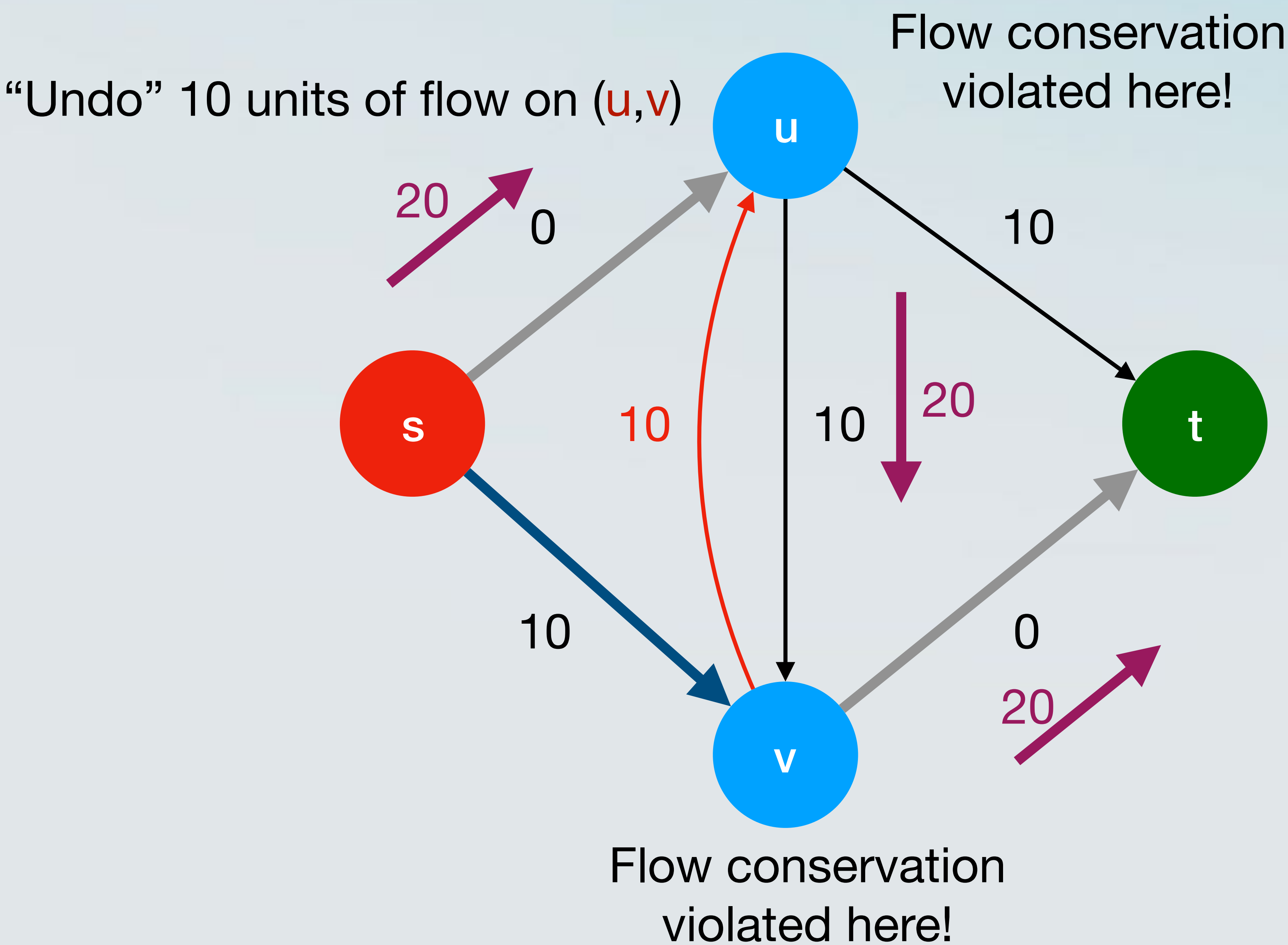


Flow conservation
violated here!

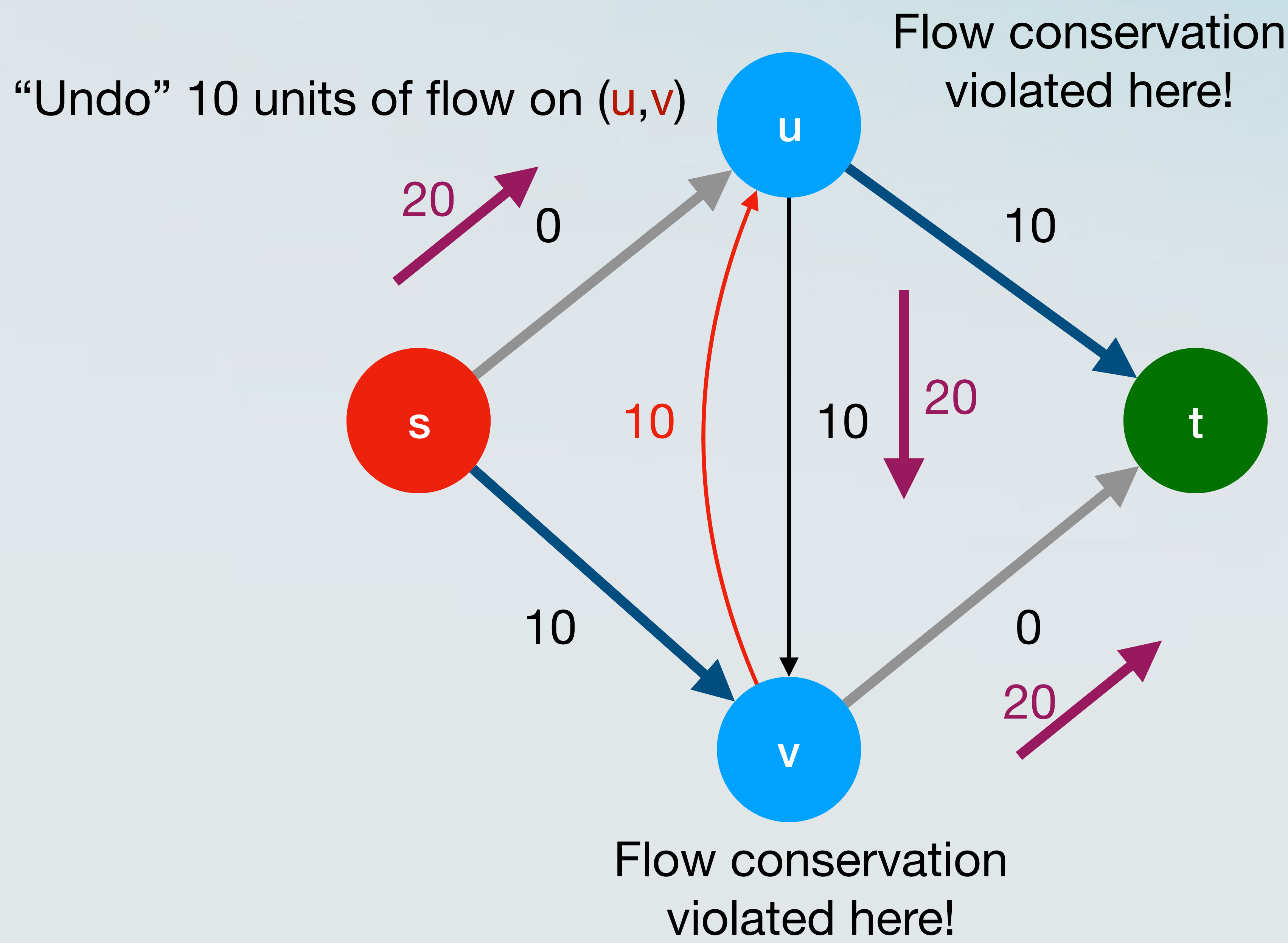
What we would like to do



What we would like to do



What we would like to do



Idea

- We can push flow *forward* on edges with leftover capacity.
- We can push flow *backward* on edges that are already carrying flow.
- How to we do that systematically?

The residual graph G_f

The residual graph G_f

- The residual graph G_f of G (also called the residual network) is defined as follows:

The residual graph G_f

- The residual graph G_f of G (also called the residual network) is defined as follows:
    - The node set V_f of G_f is the same as V.

The residual graph G_f

- The residual graph G_f of G (also called the residual network) is defined as follows:
    - The node set V_f of G_f is the same as V.
    - For each edge $e=(u, v)$ of E on which $f(e) < c_e$, there are $c_e - f(e)$ “leftover” units of capacity.

The residual graph G_f

- The residual graph G_f of G (also called the residual network) is defined as follows:
  - The node set V_f of G_f is the same as V.
  - For each edge $e=(u, v)$ of E on which $f(e) < c_e$, there are $c_e - f(e)$ "leftover" units of capacity.
    - We will call this number the residual capacity of the edge e.

The residual graph G_f

- The residual graph G_f of G (also called the residual network) is defined as follows:
    - The node set V_f of G_f is the same as V.
    - For each edge $e=(u, v)$ of E on which $f(e) < c_e$, there are $c_e - f(e)$ "leftover" units of capacity.
        - We will call this number the residual capacity of the edge e.
        - We will call the edge e a *forward edge*.

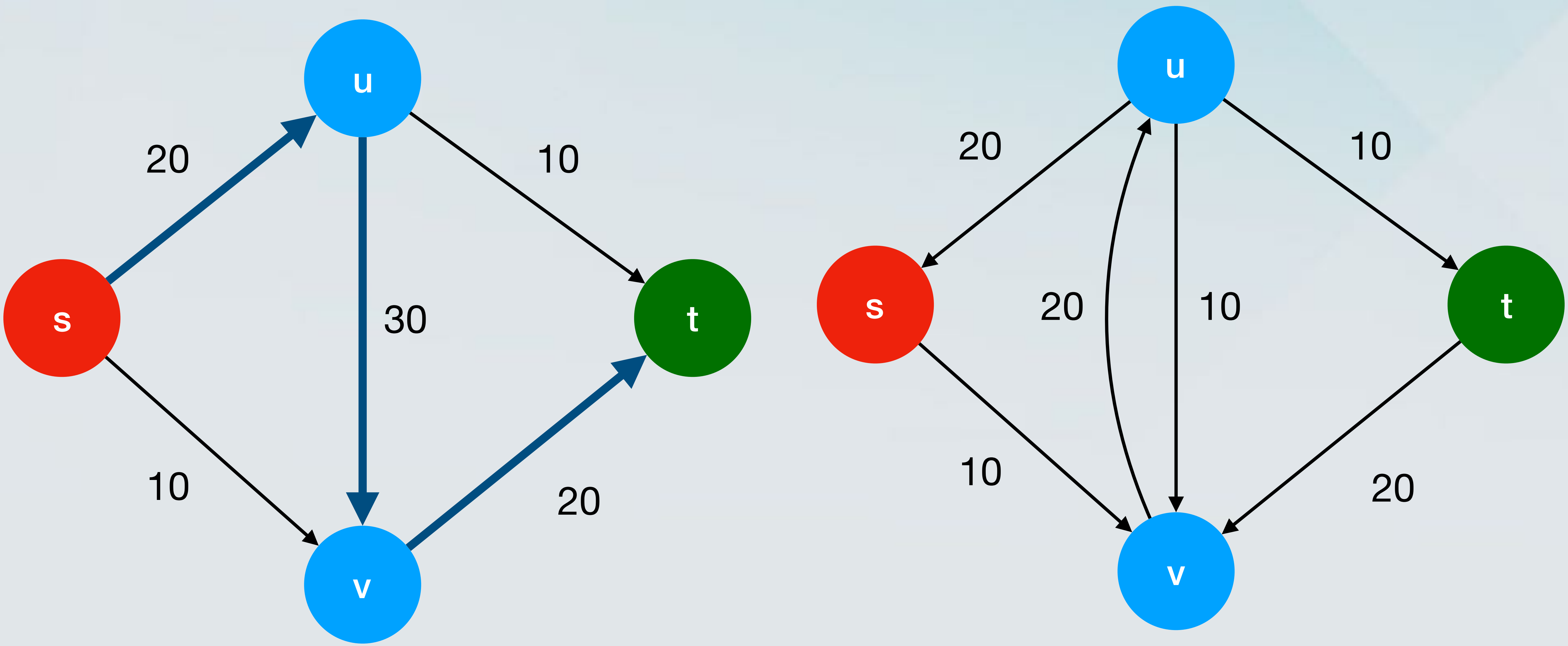
The residual graph G_f

- The residual graph G_f of G (also called the residual network) is defined as follows:
  - The node set V_f of G_f is the same as V.
  - For each edge $e=(u, v)$ of E on which $f(e) < c_e$, there are $c_e - f(e)$ “leftover” units of capacity.
    - We will call this number the residual capacity of the edge e.
    - We will call the edge e a *forward edge*.
  - For each edge $e=(u, v)$ of E on which $f(e) > 0$, there is an edge $e'=(u, v)$ in E_f with a capacity of $f(e)$.

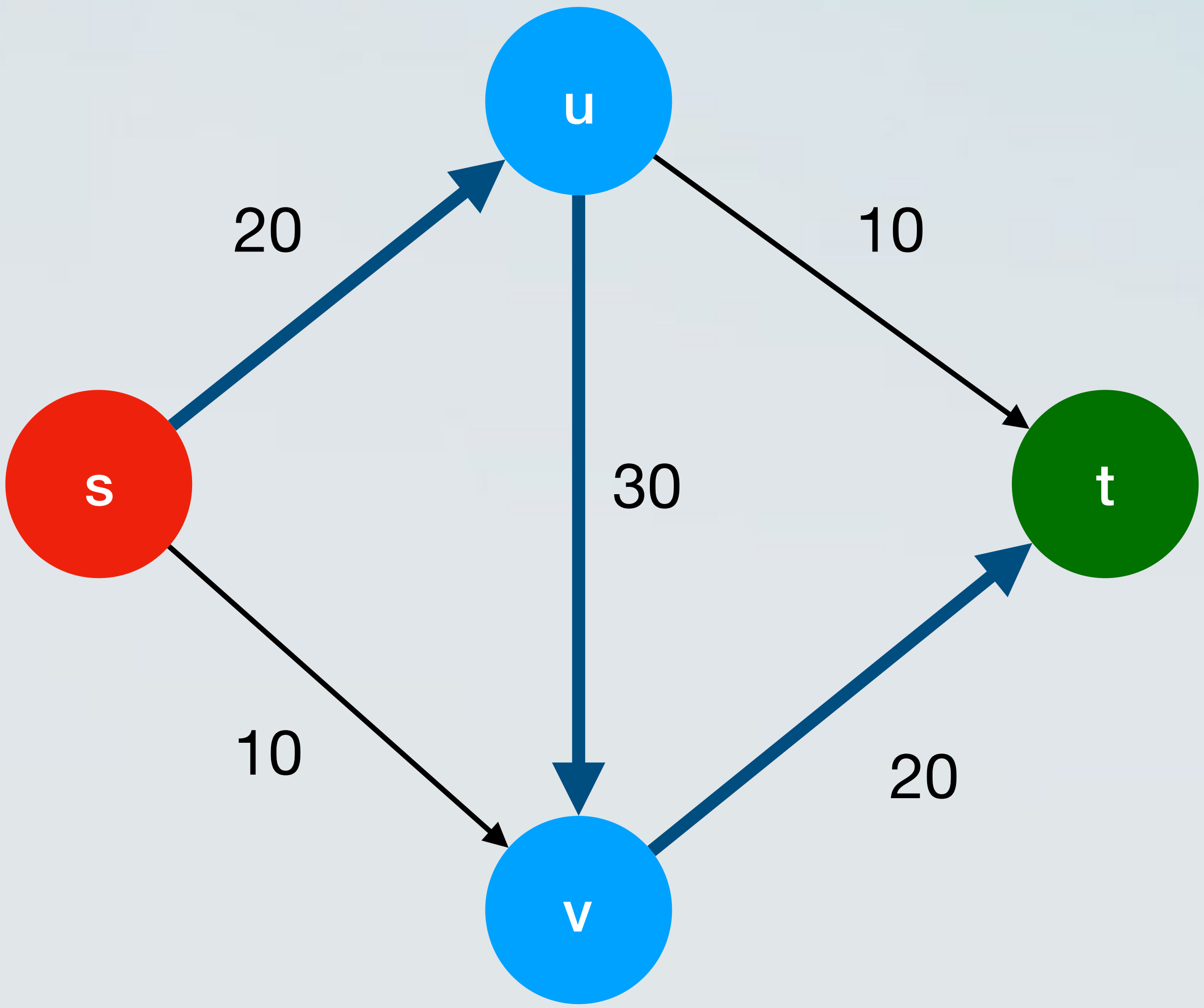
The residual graph G_f

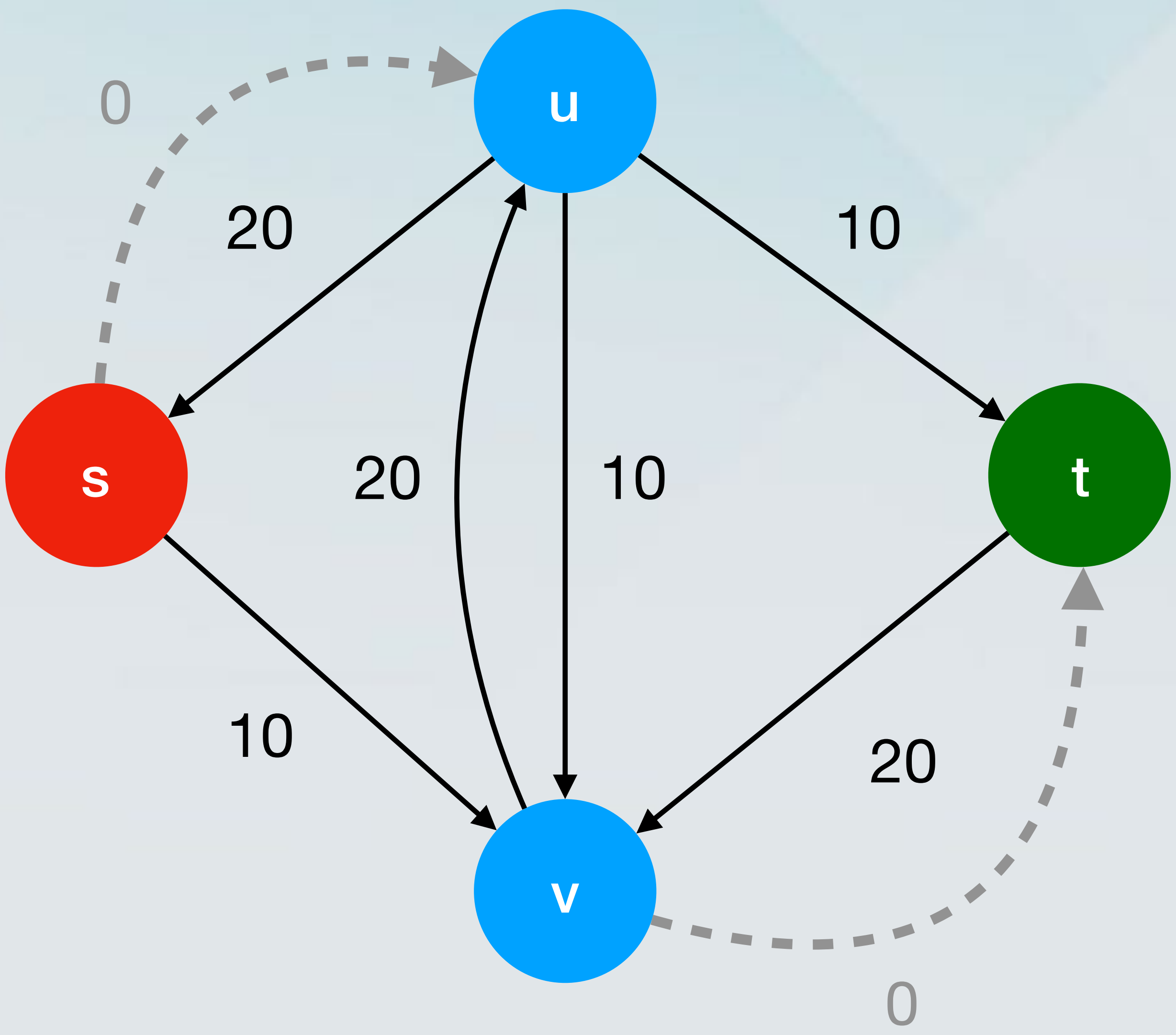
- The residual graph G_f of G (also called the residual network) is defined as follows:
  - The node set V_f of G_f is the same as V.
  - For each edge $e=(u, v)$ of E on which $f(e) < c_e$, there are $c_e - f(e)$ "leftover" units of capacity.
    - We will call this number the residual capacity of the edge e.
    - We will call the edge e a *forward edge*.
  - For each edge $e=(u, v)$ of E on which $f(e) > 0$, there is an edge $e'=(u, v)$ in E_f with a capacity of $f(e)$.
    - We will call the edge e' a *backward edge*.

Example



Example





Increasing the flow

- The flow originates from s and goes to t.
- So we have to find an (s-t) path, and route it via this path.

Increasing the flow

- The flow originates from s and goes to t.
- So we have to find an (s-t) path, and route it via this path.
- We will use paths in the residual graph G_f.

Working with the residual graph

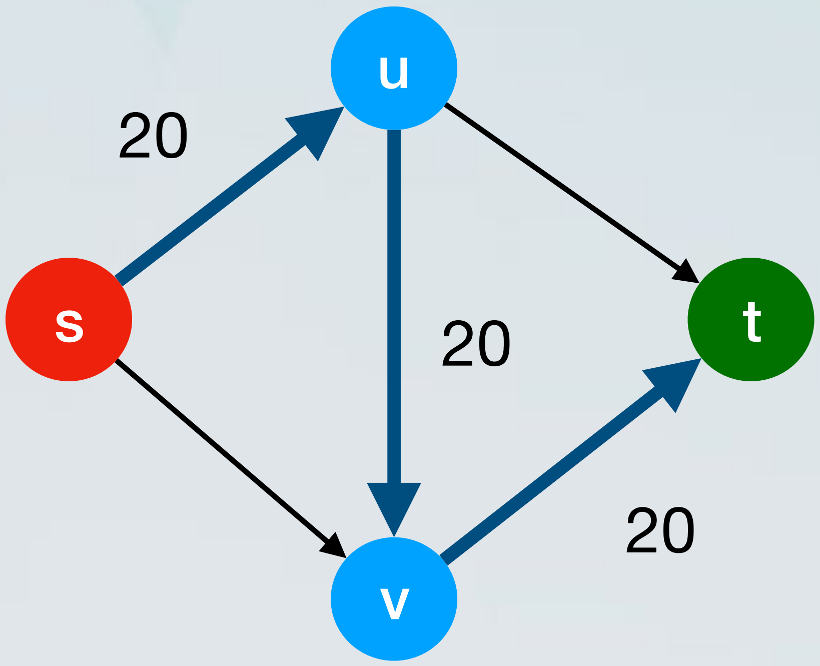
- Find an (s-t) path P in the residual graph.
  - We will call this an ***augmenting path***.
- Define the *bottleneck* of P,
  - denoted **bottleneck**(P, f)
  - to be the minimum residual capacity on any edge on P.
- Define the augmentation of flow f into flow f'
  - denoted **augment**(f, P)

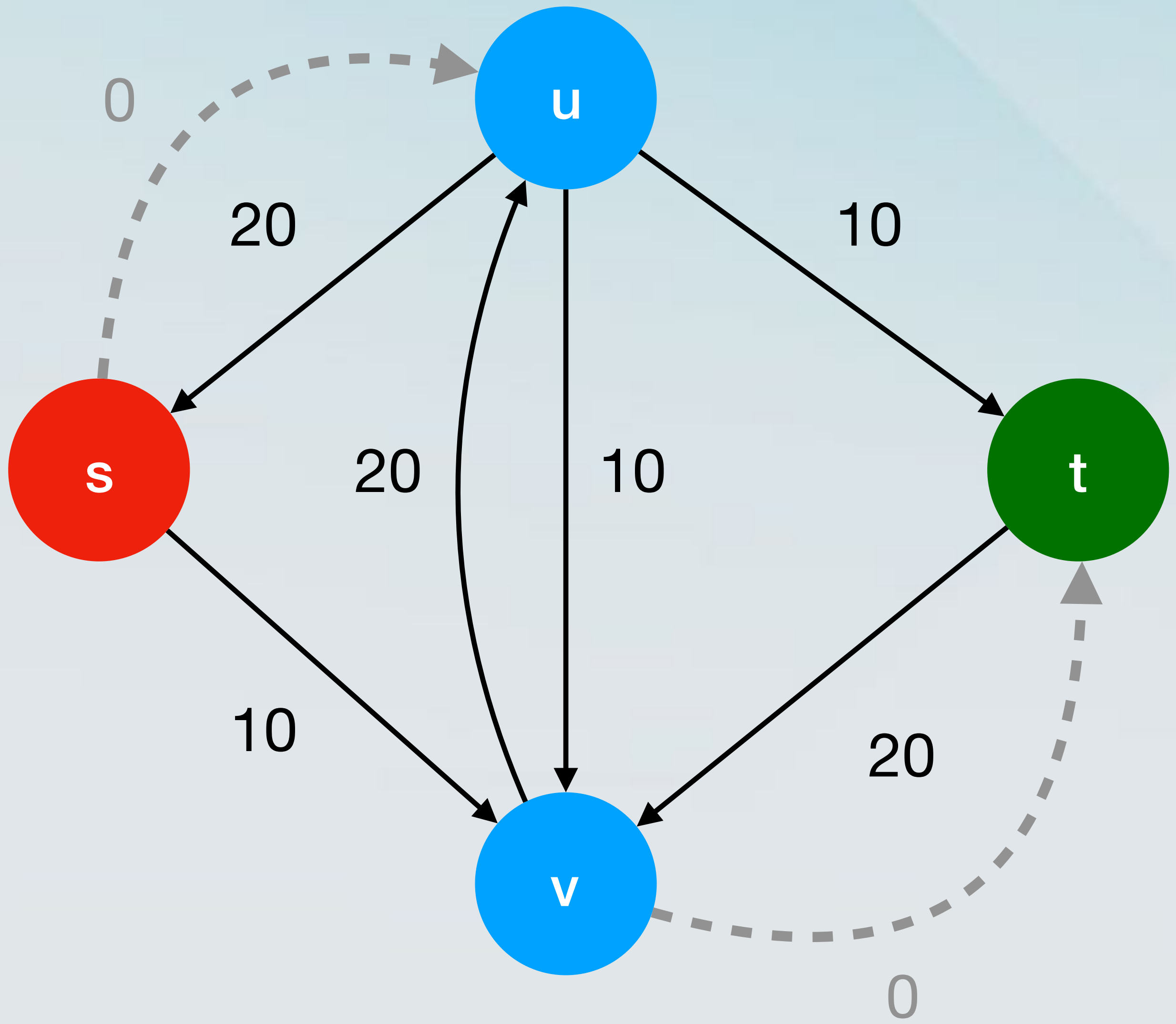
Augmenting the flow

augment(f, P)

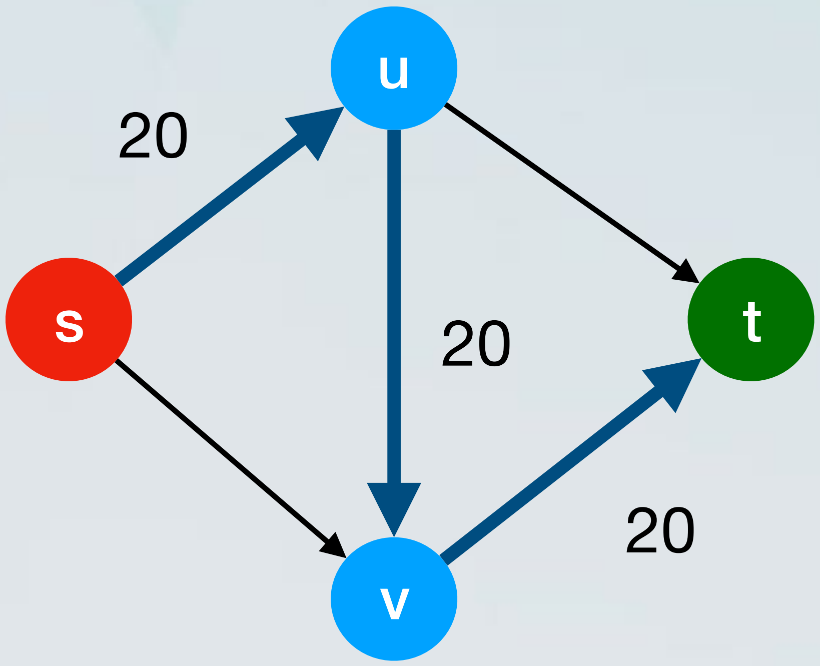
```
Let b = bottleneck(P, f)
For each edge e=(u, v) in P
      If e is a forward edge then
          Increase f(e) in G by b
      Else (e is a backward edge, and let e' = (v, u))
          Decrease f(e') in G by b
      EndIf
 EndFor

Return( f );
```

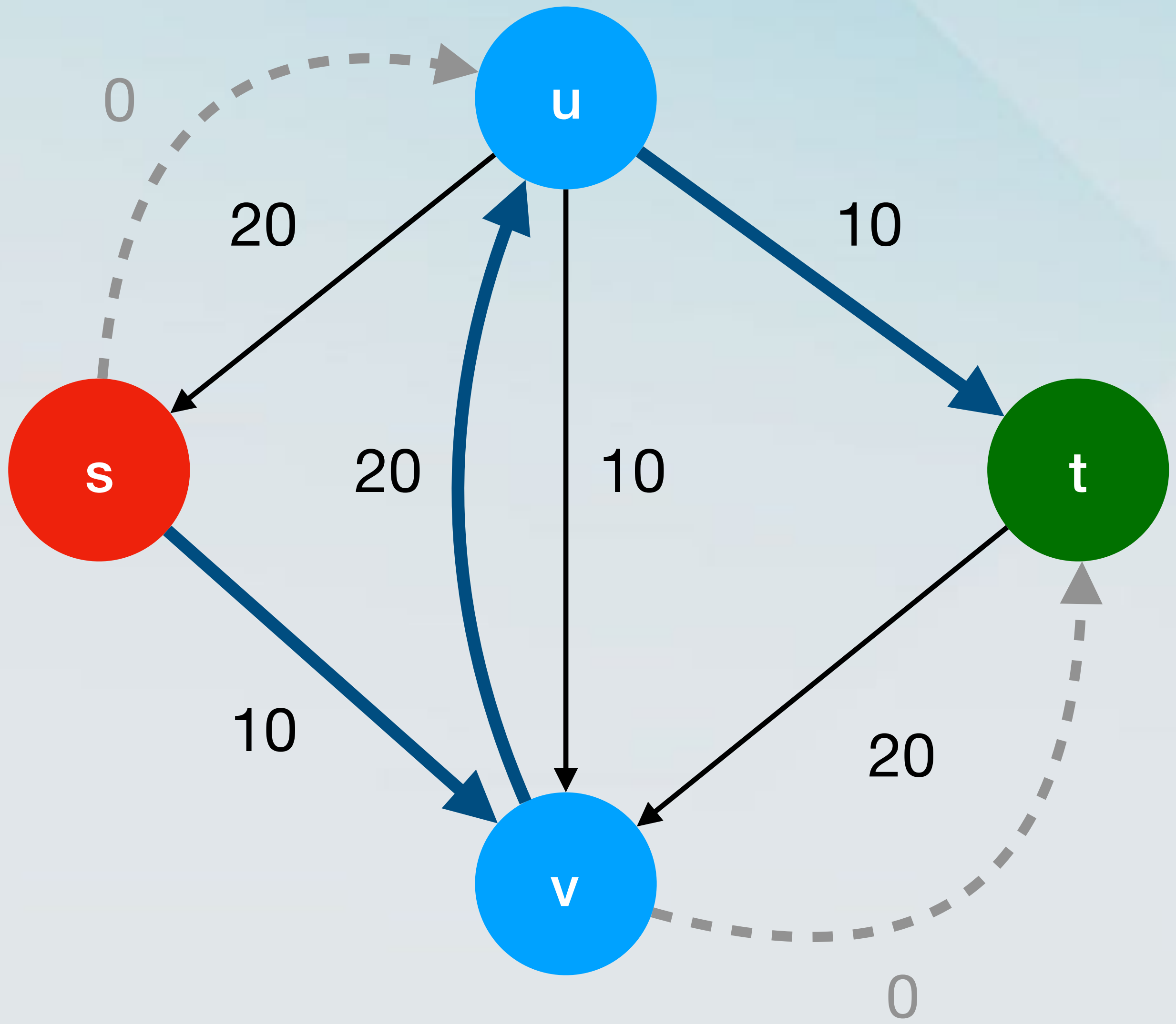
Example



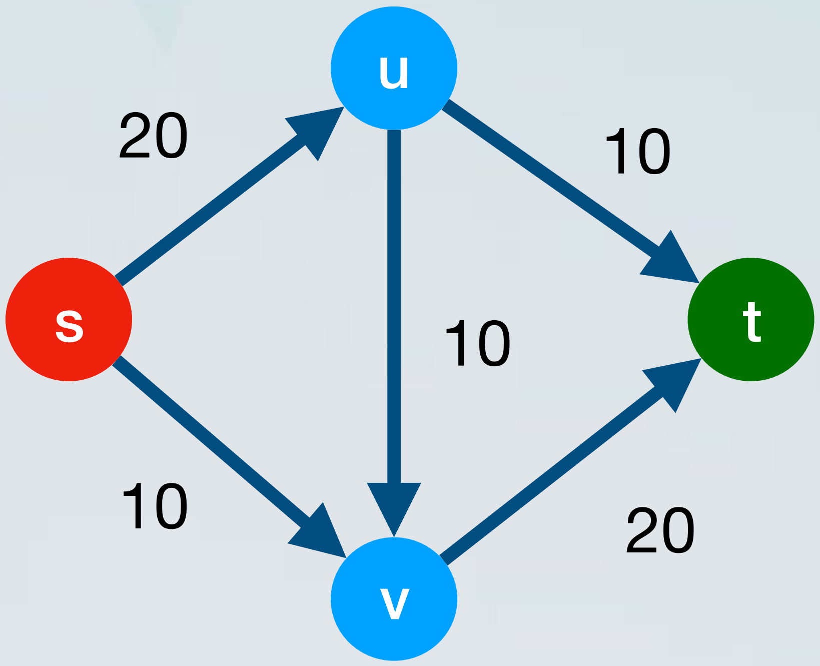


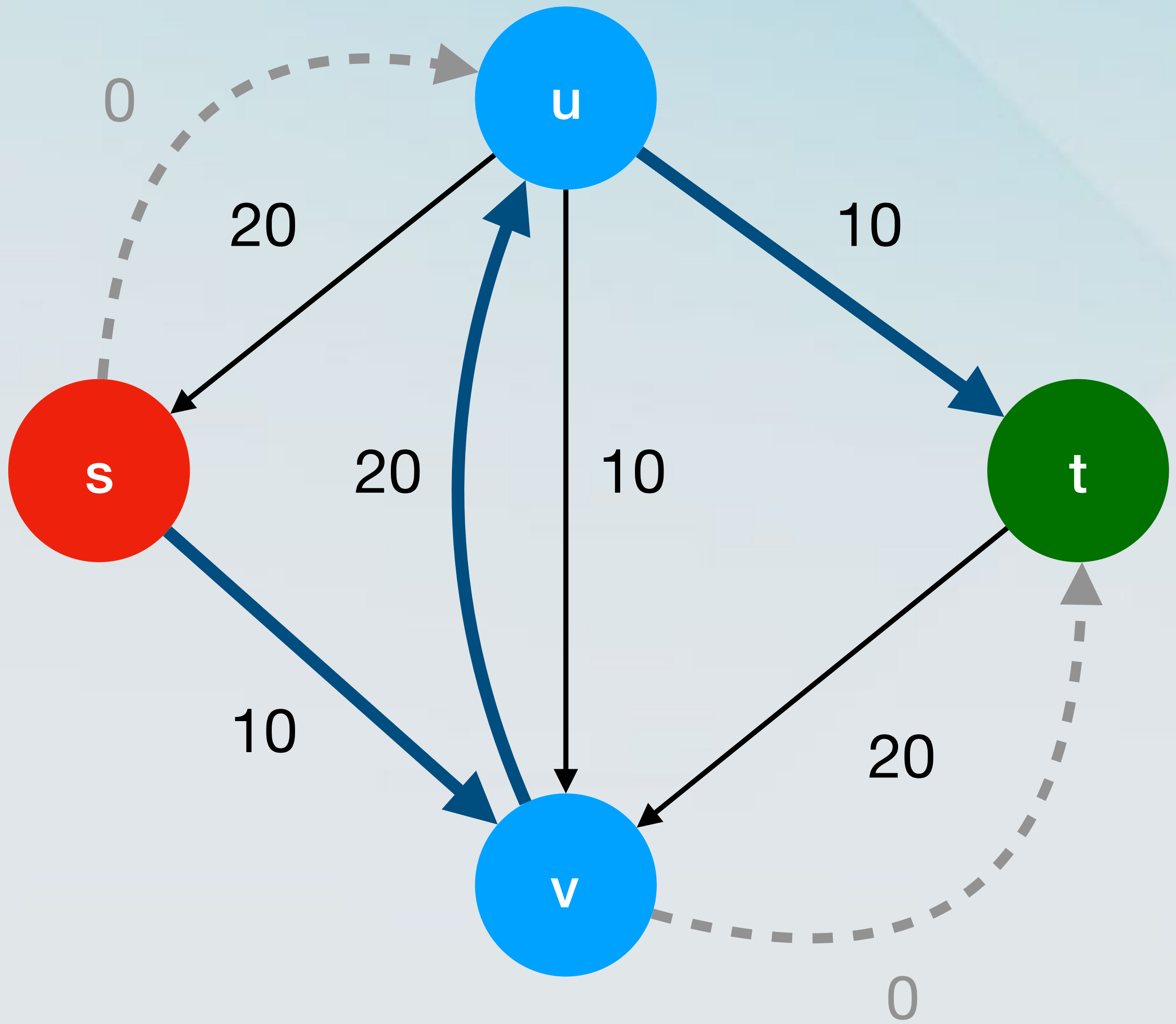
Example



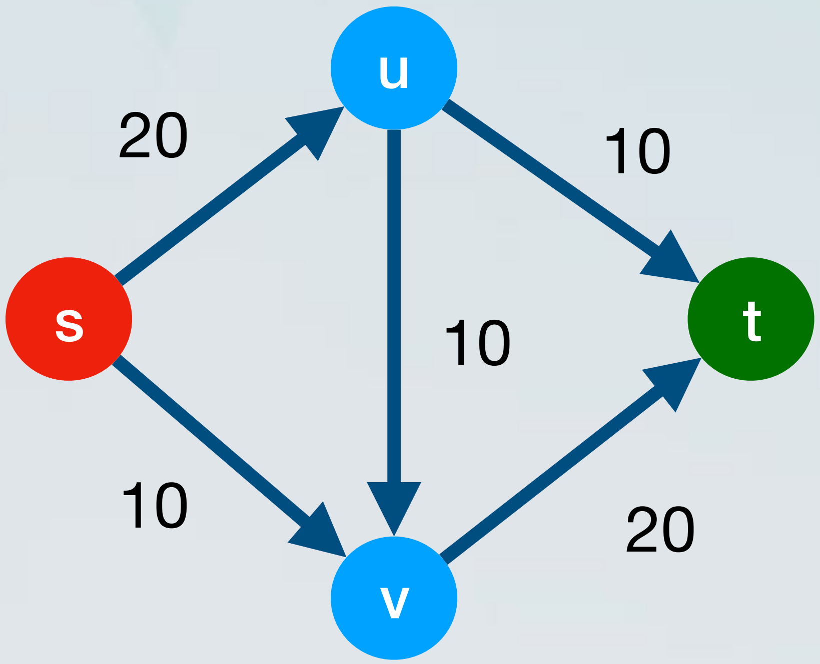


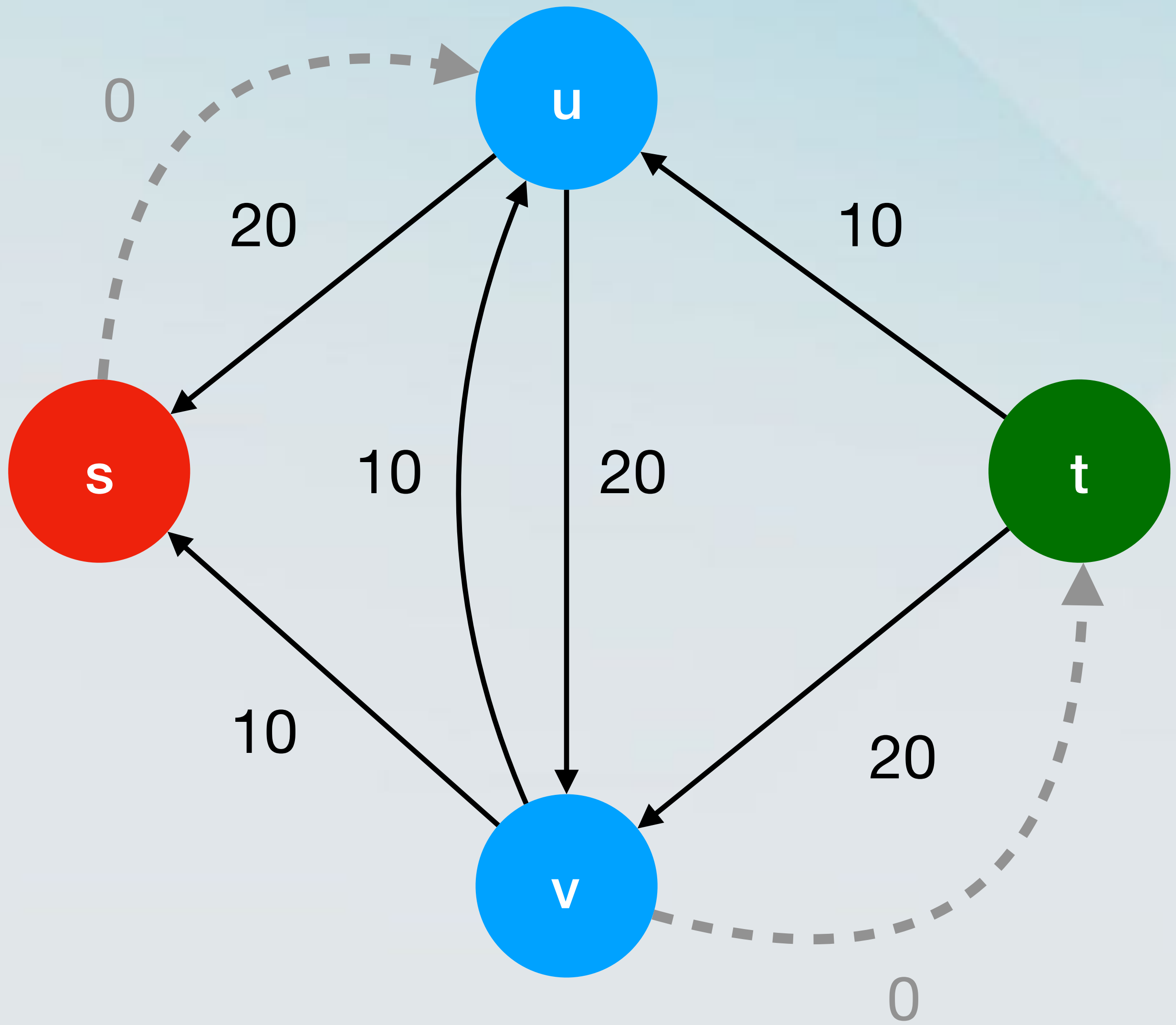
Example





Example





Feasibility

- Let f’ = **augment**(f, P)
- Is f’ a flow?
    - Suffices to only check edges e in P (why?)
    - Consider an arbitrary edge e = (u, v) of P.

Feasibility (capacity)

Feasibility (capacity)

- Consider an arbitrary edge $e = (u, v)$ of P.

Feasibility (capacity)

- Consider an arbitrary edge e = (u, v) of P.
  - Suppose that e is a *forward edge*.

Feasibility (capacity)

- Consider an arbitrary edge $e = (u, v)$ of P.
  - Suppose that e is a *forward edge*.
    - $0 \le f(e) \le f'(e) = f(e) + b \le f(e) + (c_e - f(e)) = c_e$

Feasibility (capacity)

- Consider an arbitrary edge $e = (u, v)$ of P.
  - Suppose that e is a *forward edge*.
    - $0 \leq f(e) \leq f'(e) = f(e) + b \leq f(e) + (c_e - f(e)) = c_e$
  - Suppose that e is a *backward edge*.

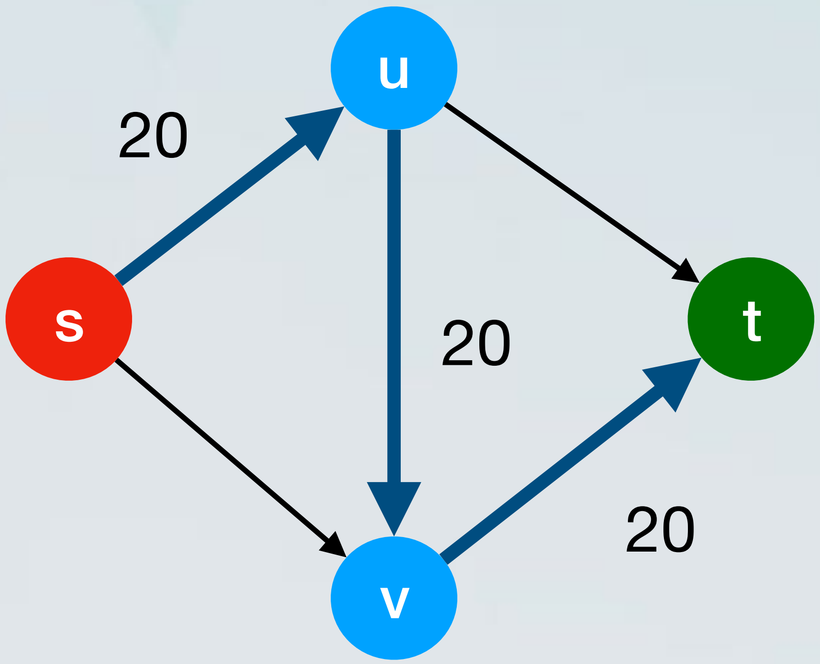
Feasibility (capacity)

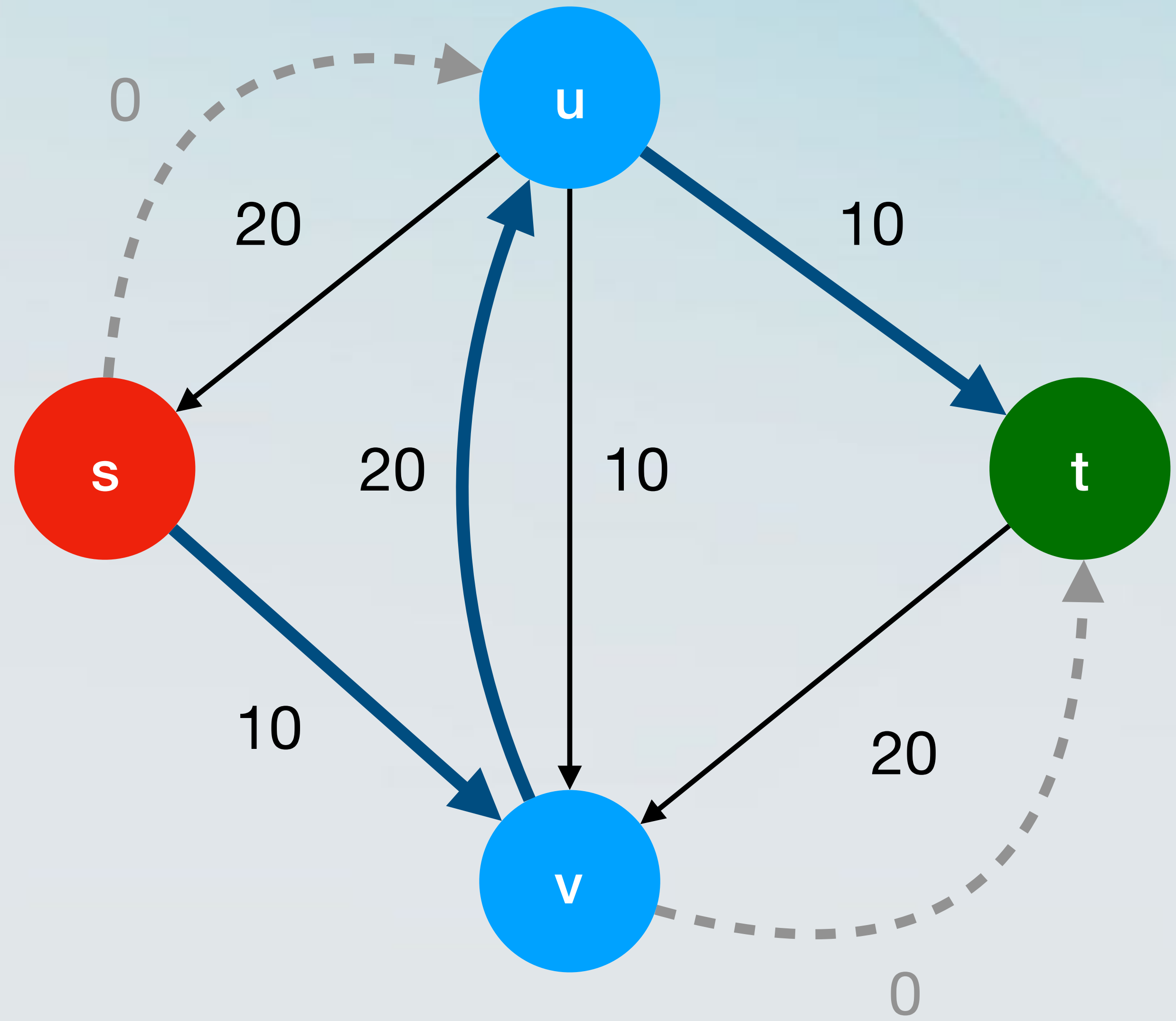
- Consider an arbitrary edge $e = (u, v)$ of P.
  - Suppose that e is a *forward edge*.
    - $0 \leq f(e) \leq f'(e) = f(e) + b \leq f(e) + (c_e - f(e)) = c_e$
  - Suppose that e is a *backward edge*.
    - $c_e \geq f(e) \geq f'(e) = f(e) - b \geq f(e) - f(e) = 0$

Feasibility (capacity)

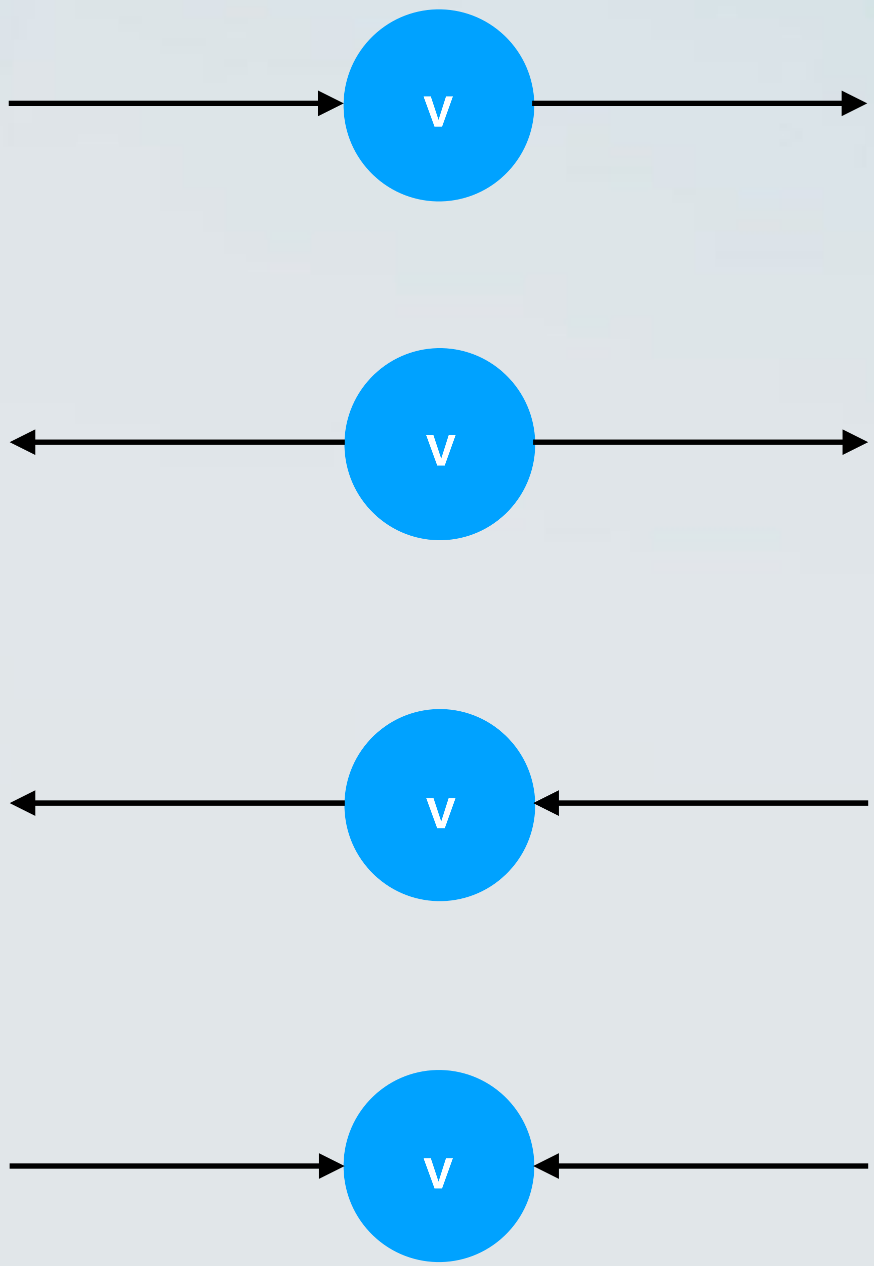
- Consider an arbitrary edge $e = (u, v)$ of P.
  - Suppose that e is a *forward edge*.
    - $0 \leq f(e) \leq f'(e) = f(e) + b \leq f(e) + (c_e - f(e)) = c_e$
  - Suppose that e is a *backward edge*.
    - $c_e \geq f(e) \geq f'(e) = f(e) - b \geq f(e) - f(e) = 0$
  - The capacity condition holds.

Example





Feasibility (flow conservation)



forward , forward

backward , forward

backward , backward

forward , backward

The Ford-Fulkerson Algorithm

The Ford-Fulkerson Algorithm

- Start with a *0 flow* f on all edges.

The Ford-Fulkerson Algorithm

- Start with a *0 flow* f on all edges.
- Find an (s-t) path in the residual graph G_f and consider it as the *augmenting path*.

The Ford-Fulkerson Algorithm

- Start with a *0 flow* f on all edges.
- Find an (s-t) path in the residual graph G_f and consider it as the *augmenting path*.
- Augment the flow f on this path to obtain new flow f'.

The Ford-Fulkerson Algorithm

- Start with a *0 flow* f on all edges.
- Find an (s-t) path in the residual graph G_f and consider it as the *augmenting path*.
- Augment the flow f on this path to obtain new flow f'.
- Update the residual graph to G_f'.

The Ford-Fulkerson Algorithm

- Start with a *0 flow* f on all edges.
- Find an (s-t) path in the residual graph G_f and consider it as the *augmenting path*.
- Augment the flow f on this path to obtain new flow f'.
- Update the residual graph to G_f'.
- Repeat the same process for flow f' and graph G_f'.

The Ford-Fulkerson Algorithm

- Start with a *0 flow* f on all edges.
- Find an (s-t) path in the residual graph G_f and consider it as the *augmenting path*.
- Augment the flow f on this path to obtain new flow f'.
- Update the residual graph to G_f'.
- Repeat the same process for flow f' and graph G_f'.
- Until the residual graph has no more (s-t) paths.

The Ford-Fulkerson Algorithm

Max-Flow

Initially set f(e) = 0 for all e in E.

While there exists an s-t path in the residual graph Gf

Choose such a path P
f' = **augment**(f, P)
Update f to be f'
Update the residual graph to be Gf'

Endwhile

Return (f)

Ford-Fulkerson analysis

Ford-Fulkerson analysis

- **Feasibility**

Ford-Fulkerson analysis

- **Feasibility**
  - Does the algorithm produce a flow if it terminates?

Ford-Fulkerson analysis

- **Feasibility**
  - Does the algorithm produce a flow if it terminates?
- **Termination**

Ford-Fulkerson analysis

- **Feasibility**
  - Does the algorithm produce a flow if it terminates?
- **Termination**
  - Does the algorithm always terminate?

Ford-Fulkerson analysis

- **Feasibility**
  - Does the algorithm produce a flow if it terminates?
- **Termination**
  - Does the algorithm always terminate?
- **Running Time**

Ford-Fulkerson analysis

- **Feasibility**
  - Does the algorithm produce a flow if it terminates?
- **Termination**
  - Does the algorithm always terminate?
- **Running Time**
  - What is the running time of the algorithm?

Ford-Fulkerson analysis

- **Feasibility**
  - Does the algorithm produce a flow if it terminates?
- **Termination**
  - Does the algorithm always terminate?
- **Running Time**
  - What is the running time of the algorithm?
- **Optimality / Correctness**

Ford-Fulkerson analysis

- **Feasibility**
  - Does the algorithm produce a flow if it terminates?
- **Termination**
  - Does the algorithm always terminate?
- **Running Time**
  - What is the running time of the algorithm?
- **Optimality / Correctness**
  - Does the algorithm produce a maximum flow?

Feasibility

Feasibility

- We start from the *0 flow*.

Feasibility

- We start from the *0 flow*.
  - Obviously feasible.

Feasibility

- We start from the *0 flow*.
  - Obviously feasible.
- In each call f' = **augment**(f, P) , where f is a feasible flow we get a feasible flow f'.

Feasibility

- We start from the *0 flow*.
  - Obviously feasible.
- In each call f’ = **augment**(f, P) , where f is a feasible flow we get a feasible flow f’.
  - This is what we established in the previous slides.

Feasibility

- We start from the *0 flow*.
    - Obviously feasible.
- In each call f' = **augment**(f, P) , where f is a feasible flow we get a feasible flow f'.
    - This is what we established in the previous slides.
- We never, at any step, produce an infeasible flow.

Termination

Termination

- Simple fact: At every step of the algorithm, all the residual capacities and flow values are integers.

Termination

- Simple fact: At every step of the algorithm, all the residual capacities and flow values are integers.

- Claim: The value of the flow strictly increases in each augmentation step.

Termination

- Simple fact: At every step of the algorithm, all the residual capacities and flow values are integers.
- Claim: The value of the flow strictly increases in each augmentation step.
    - Take the first edge e on the augmenting path P in the residual graph.

Termination

- Simple fact: At every step of the algorithm, all the residual capacities and flow values are integers.
- Claim: The value of the flow strictly increases in each augmentation step.
  - Take the first edge e on the augmenting path P in the residual graph.
  - The edge e must be incident to s. It must also be a *forward edge*.

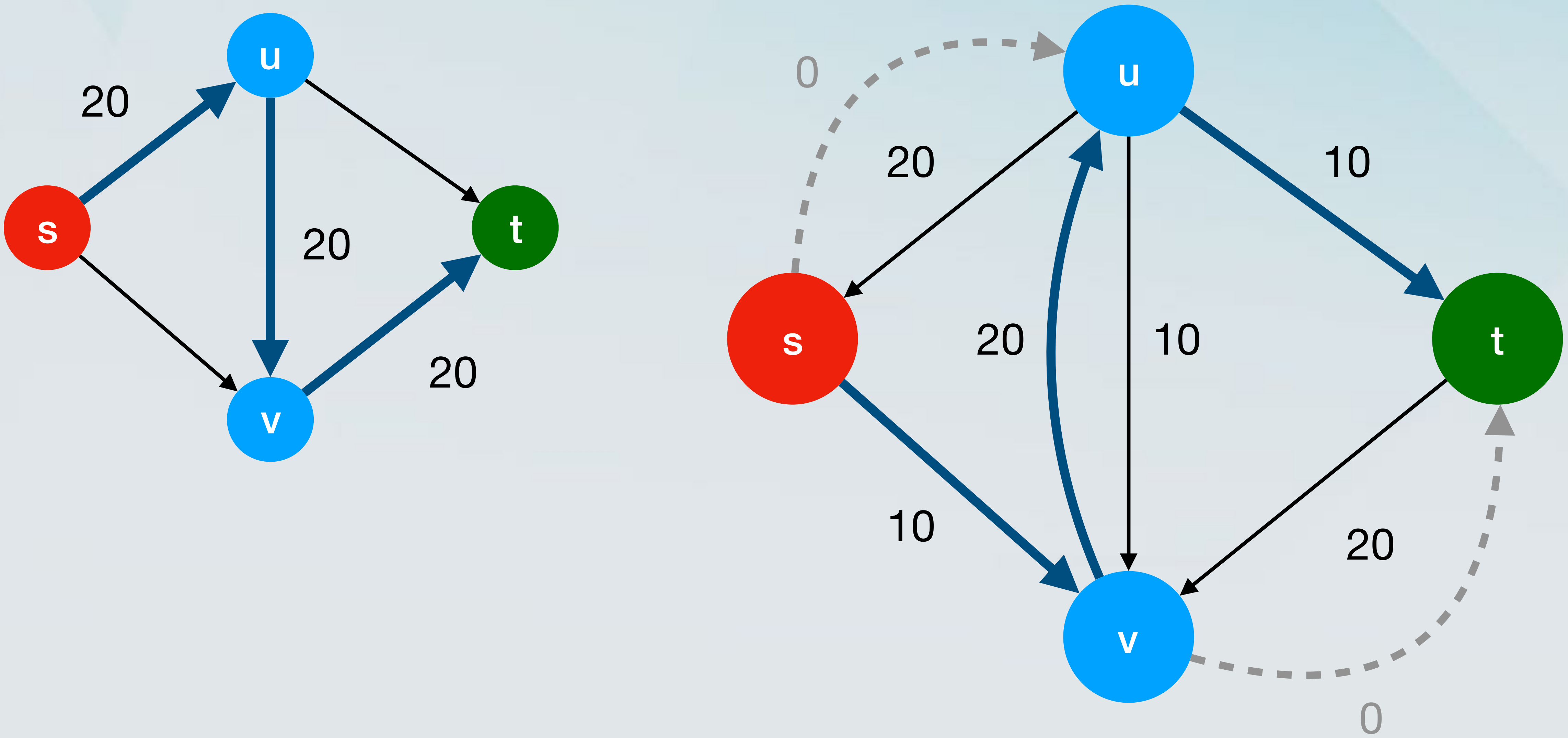
Termination

- Simple fact: At every step of the algorithm, all the residual capacities and flow values are integers.
- Claim: The value of the flow strictly increases in each augmentation step.
  - Take the first edge e on the augmenting path P in the residual graph.
  - The edge e must be incident to s. It must also be a *forward edge*.
  - We increase the previous flow by bottleneck(P, f) and we don’t change the flow on any other edge incident to s.

Termination

- Simple fact: At every step of the algorithm, all the residual capacities and flow values are integers.
- Claim: The value of the flow strictly increases in each augmentation step.
  - Take the first edge e on the augmenting path P in the residual graph.
  - The edge e must be incident to s. It must also be a *forward edge*.
  - We increase the previous flow by bottleneck(P, f) and we don't change the flow on any other edge incident to s.
  - Therefore f' is larger than f by bottleneck(P, f), which is strictly positive.

Example



Termination

Termination

- In every augmentation step of the algorithm, the value of the flow will strictly increase.

Termination

- In every augmentation step of the algorithm, the value of the flow will strictly increase.
- The algorithm will not “cycle” through the values.

Termination

- In every augmentation step of the algorithm, the value of the flow will strictly increase.
- The algorithm will not “cycle” through the values.
- So the algorithm will terminate.

Termination

- In every augmentation step of the algorithm, the value of the flow will strictly increase.
- The algorithm will not “cycle” through the values.
- So the algorithm will terminate.
    - Not necessarily!

Termination

- In every augmentation step of the algorithm, the value of the flow will strictly increase.
- The algorithm will not “cycle” through the values.
- So the algorithm will terminate.
  - Not necessarily!
  - The maximum flow has to be bounded!

Termination

Termination

- What can we use as a bound for the maximum flow?

Termination

- What can we use as a bound for the maximum flow?
- We can use the total capacity C out of s. $\sum_{e \textbf{ out of } s} c_e$

Termination

- What can we use as a bound for the maximum flow?
- We can use the total capacity C out of s. $\sum_{e \text{ out of } s} c_e$
- So the algorithm will terminate in at most C steps.

Termination

- What can we use as a bound for the maximum flow?
- We can use the total capacity C out of s. $\sum_{e \textbf{ out of } s} c_e$
- So the algorithm will terminate in at most C steps.
    - Is this true?

Integer Capacities

Integer Capacities

- Since the capacities are integers, the flow will increase *by at least 1* in each augmentation step.

Integer Capacities

- Since the capacities are integers, the flow will increase *by at least 1* in each augmentation step.

- So yes, it is true for integer capacities?

Integer Capacities

- Since the capacities are integers, the flow will increase *by at least 1* in each augmentation step.
- So yes, it is true for integer capacities?
- Is it true for capacities which are real numbers?

Integer Capacities

- Since the capacities are integers, the flow will increase *by at least 1* in each augmentation step.
- So yes, it is true for integer capacities?
- Is it true for capacities which are real numbers?
    - No - tutorial on Friday.

Running Time

Running Time

- We have assumed that every node is incident to at least on edge in the graph.

Running Time

- We have assumed that every node is incident to at least on edge in the graph.
    - This means that O(m+n) = O(m).

Running Time

- We have assumed that every node is incident to at least on edge in the graph.
  - This means that O(m+n) = O(m).
- The termination analysis already hinted at the running time.

Running Time

- We have assumed that every node is incident to at least on edge in the graph.
  - This means that O(m+n) = O(m).
- The termination analysis already hinted at the running time.
  - In each step, we need to find an augmenting path and compute the residual graph.

Running Time

- We have assumed that every node is incident to at least on edge in the graph.
  - This means that $O(m+n) = O(m)$.
- The termination analysis already hinted at the running time.
  - In each step, we need to find an augmenting path and compute the residual graph.
  - How do we do that?

Running Time

- We have assumed that every node is incident to at least on edge in the graph.
  - This means that $O(m+n) = O(m)$.
- The termination analysis already hinted at the running time.
  - In each step, we need to find an augmenting path and compute the residual graph.
  - How do we do that?
  - How many iterations do we have?

Running Time

- We have assumed that every node is incident to at least on edge in the graph.
  - This means that O(m+n) = O(m).
- The termination analysis already hinted at the running time.
  - In each step, we need to find an augmenting path and compute the residual graph.
  - How do we do that?
  - How many iterations do we have?
- The running time of *FF* is **O(*m*F)**, where F is the value of the maximum flow.

Running Time

Running Time

- The running time of *FF* is $\mathbf{O}(m\mathrm{F})$, where F is the value of the maximum flow.

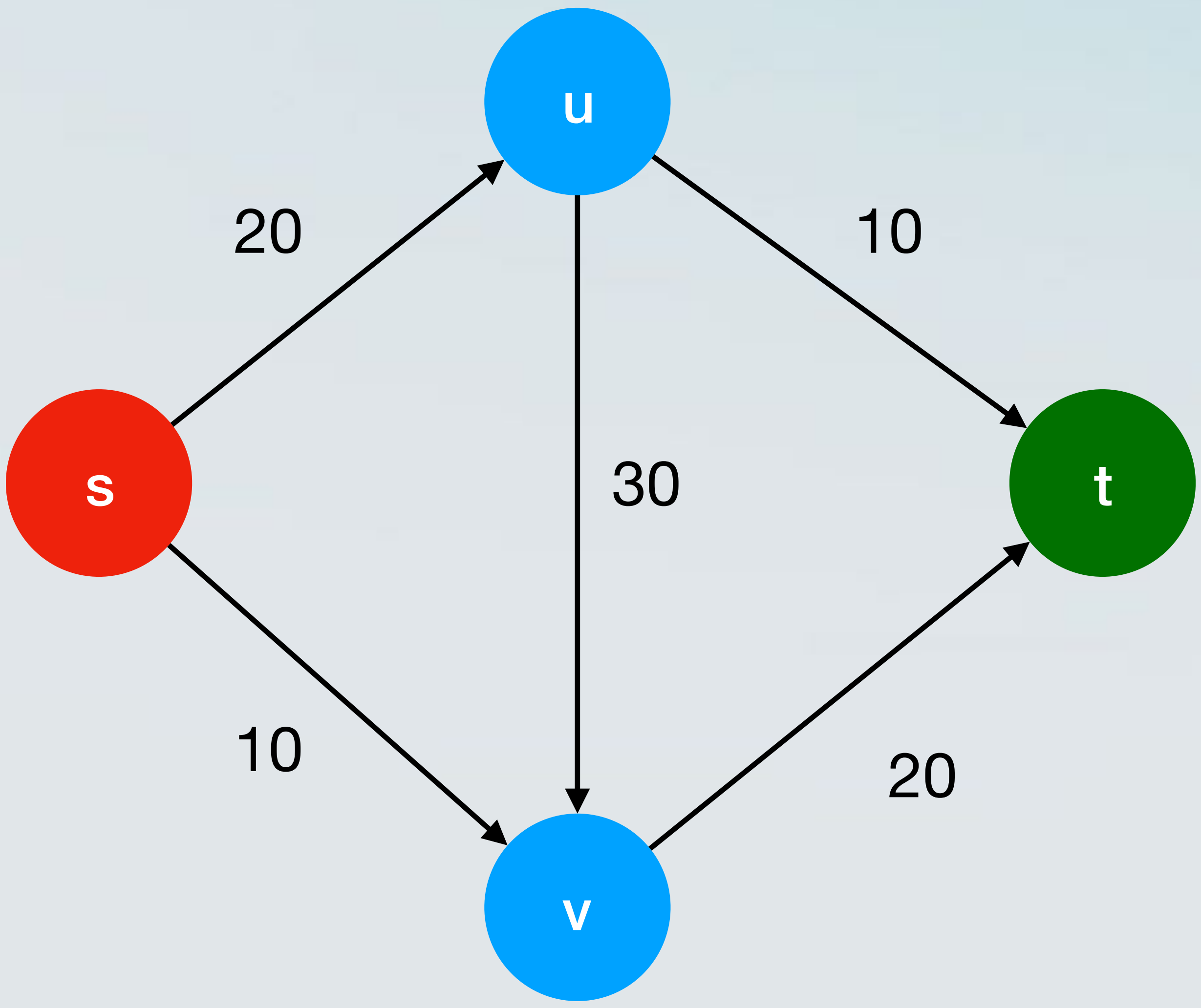
Running Time

- The running time of *FF* is **O(*m*F)**, where F is the value of the maximum flow.
- Is this a polynomial time algorithm?

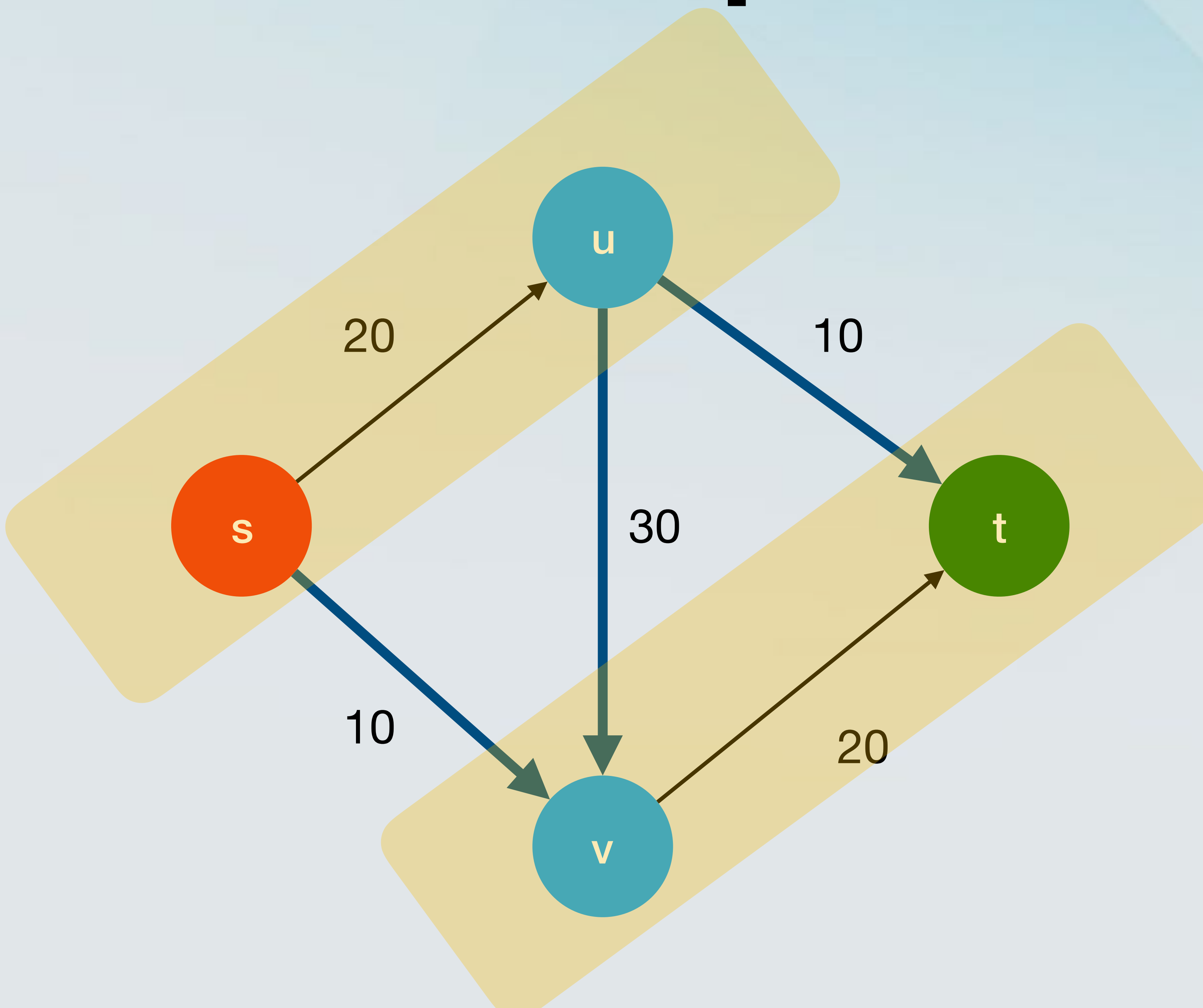
Minimum Cut

- A *cut* C is a partition of the nodes of G into two sets S and T, such that s is in S and t is in T.
- The capacity c(S,T) of a cut C is the sum of capacities of all edges "out of S"
    - these are edges (u, v) where u is in S and v is in T.

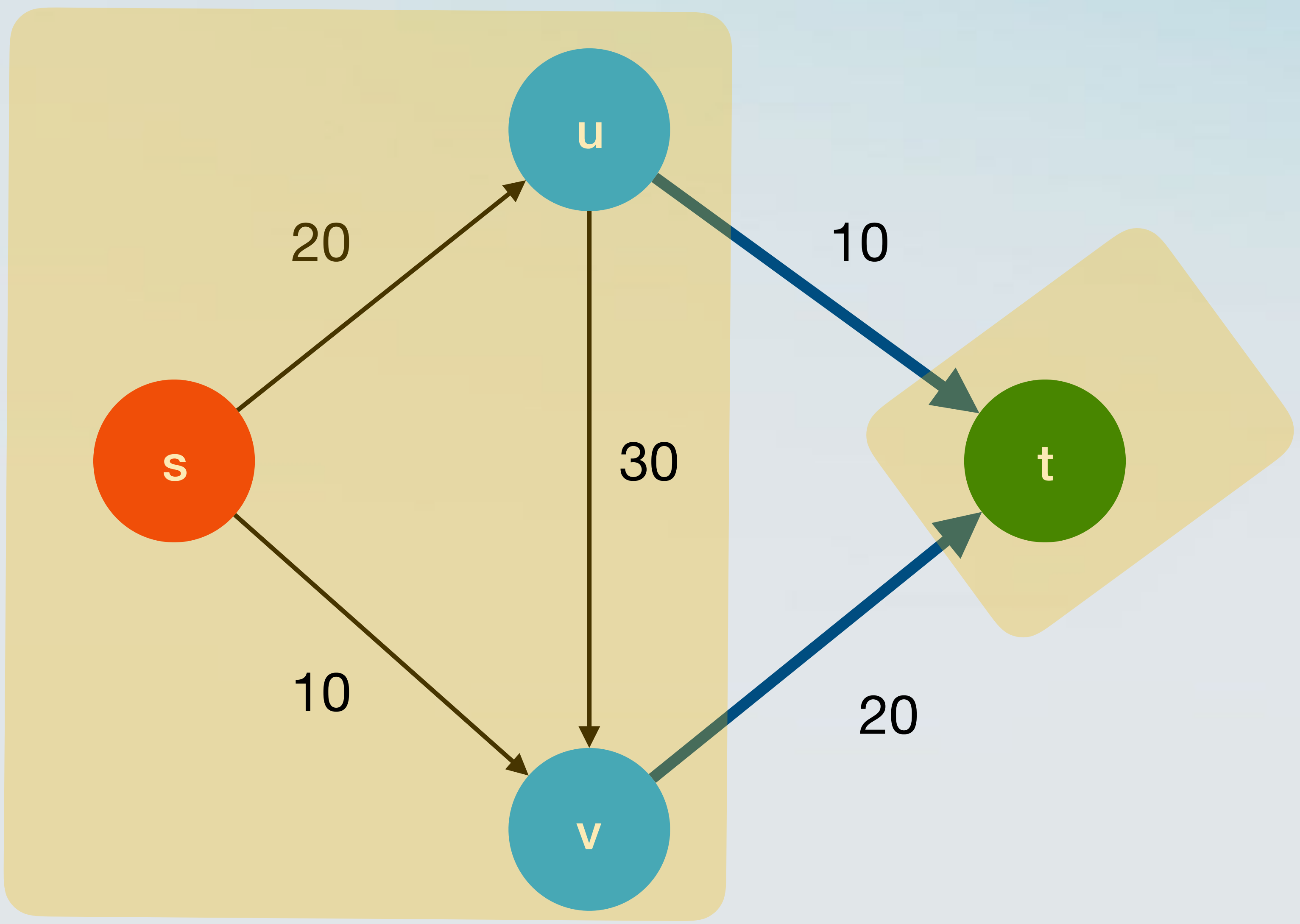
Example



Example



Example



The Max-Flow Min-Cut Theorem

- Theorem: In every flow network, the value of the maximum flow is *equal* to the capacity of the minimum cut.

Optimality / Correctness

- Next Lecture!