Advanced Algorithmic Techniques (COMP523)

Network Flows 2

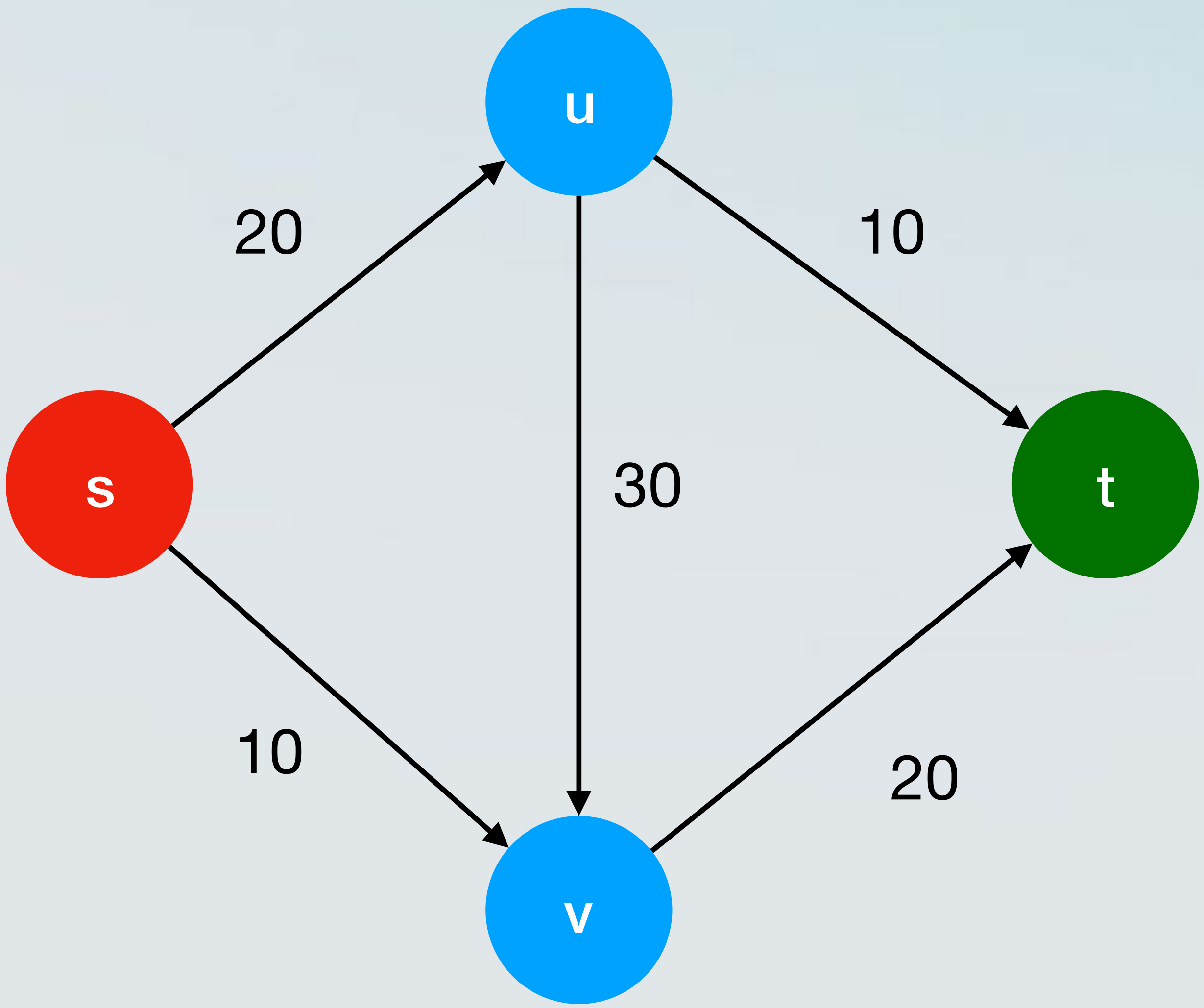
Recap and plan

- **Last lecture:**
  - Network Flows, Maximum Flow
  - Ford - Fulkerson
    - Feasibility, termination, running time
  - Max-Flow - Min-Cut
- **This lecture:**
  - Ford - Fulkerson
    - Optimality / Correctness
  - Better augmenting paths.
  - Maximum Bipartite Matching

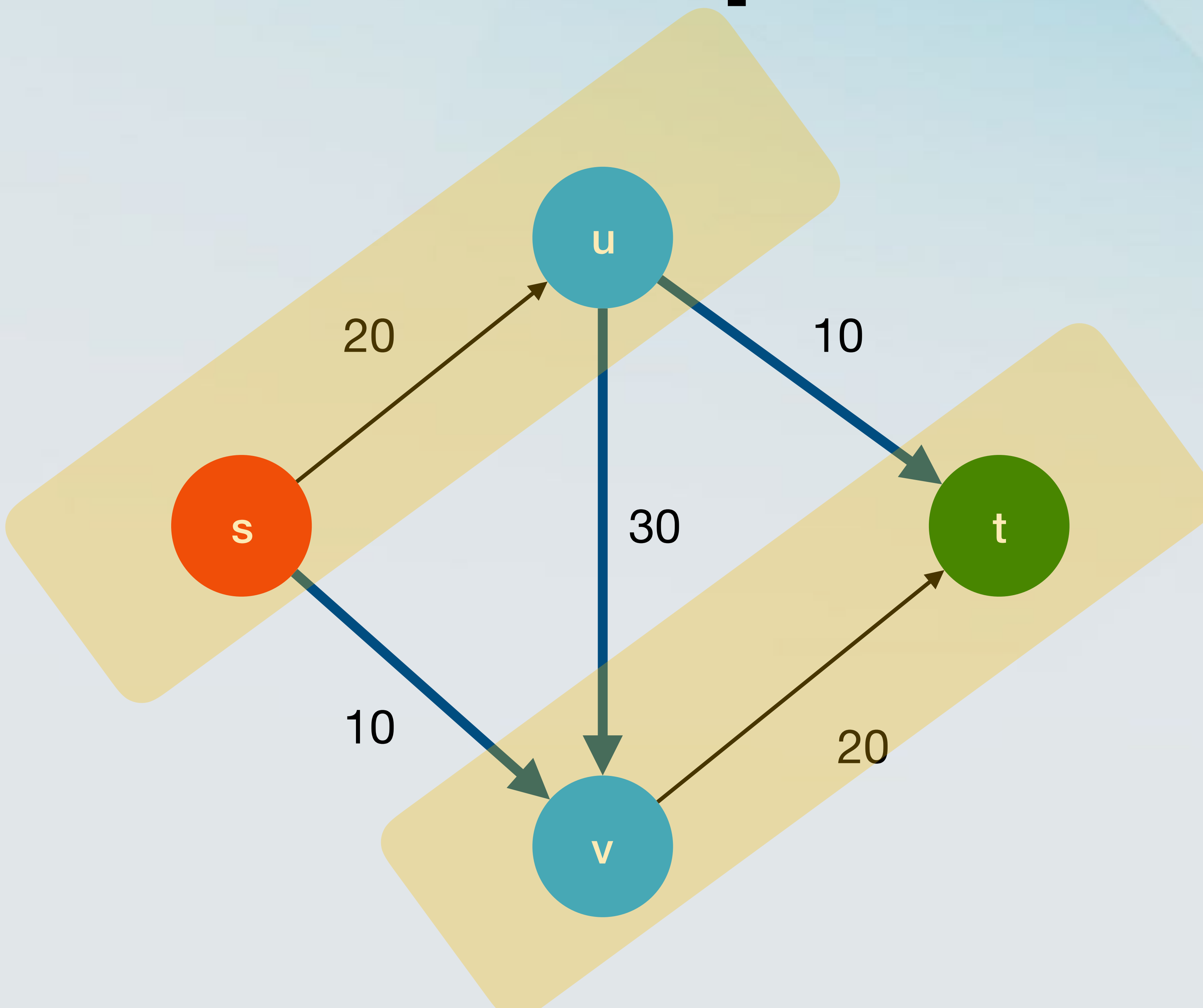
Minimum Cut

- 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.

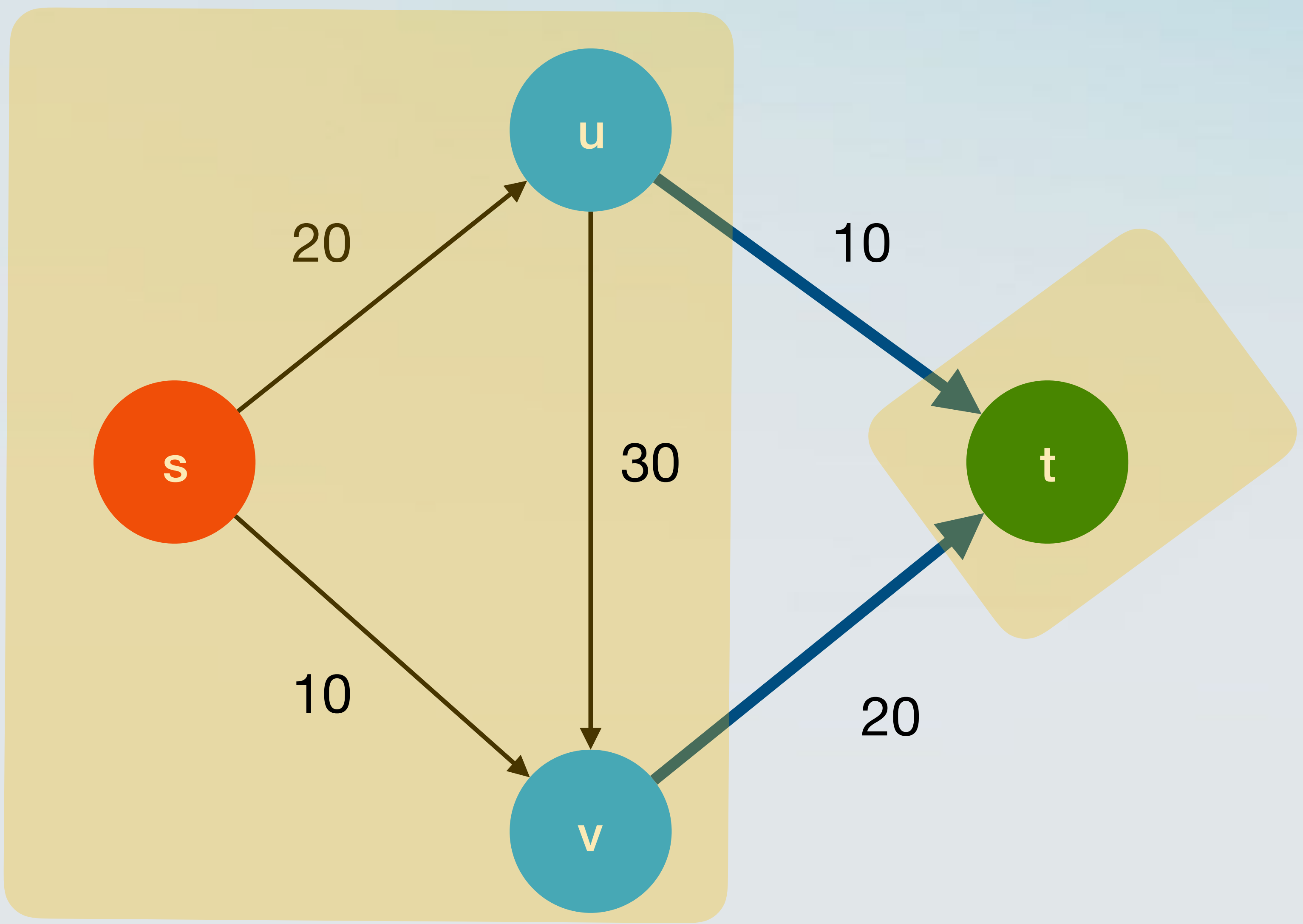
Example



Example



Example

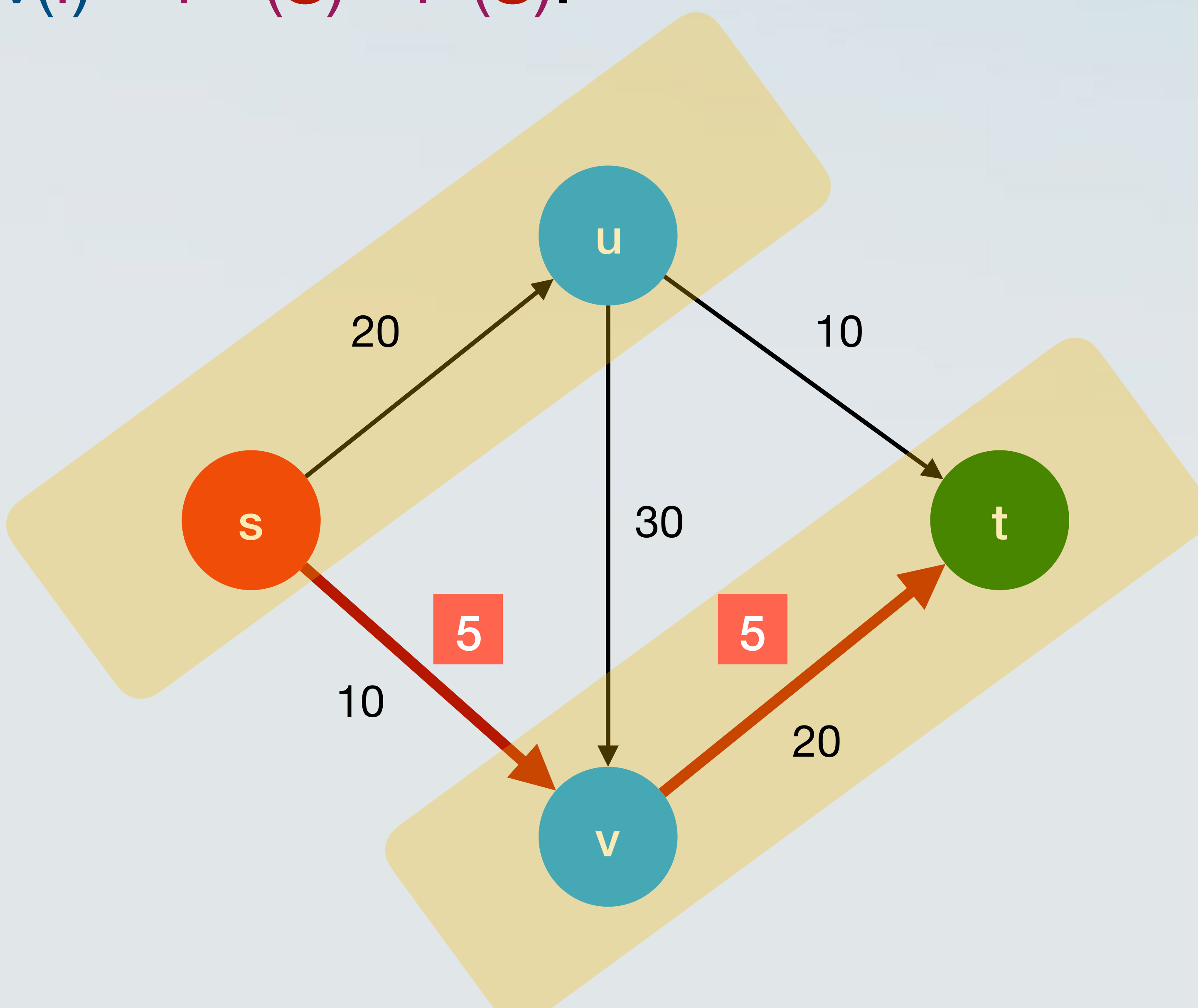


The Max-Flow Min-Cut Theorem

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

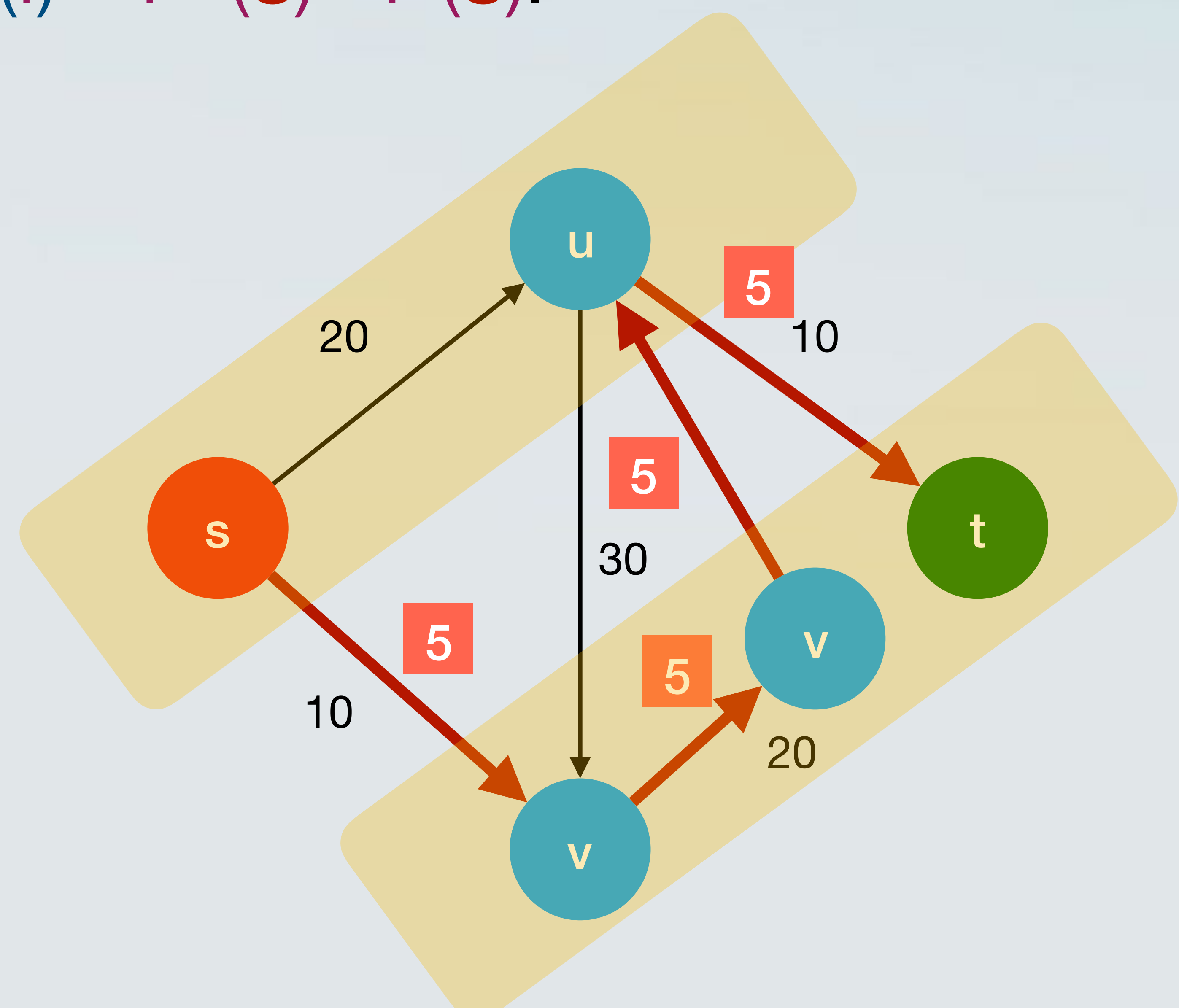
A series of facts

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.



Fact 1

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.



Fact 1

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.

Fact 1

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.

  - By definition, $v(f) = f^{out}(s)$.

Fact 1

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.

  - By definition, $v(f) = f^{out}(s)$.

  - By definition $f^{in}(s) = 0$.

Fact 1

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - By definition, $v(f) = f^{out}(s)$.
  - By definition $f^{in}(s) = 0$.
  - Hence, by definition $v(f) = f^{out}(s) - f^{in}(s)$.

Fact 1

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - By definition, $v(f) = f^{out}(s)$.
  - By definition $f^{in}(s) = 0$.
  - Hence, by definition $v(f) = f^{out}(s) - f^{in}(s)$.
  - For every other node v, we have that $f^{out}(v) - f^{in}(v) = 0$ (why?)

Fact 1

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
    - By definition, $v(f) = f^{out}(s)$.
    - By definition $f^{in}(s) = 0$.
    - Hence, by definition $v(f) = f^{out}(s) - f^{in}(s)$.
    - For every other node v, we have that $f^{out}(v) - f^{in}(v) = 0$ (why?)
    - Therefore we have:

$$v(f) = \sum_{v \in S} \left(f^{\mathbf{out}}(v) - f^{\mathbf{in}}(v) \right)$$

Fact 1 - Rewriting the sums

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have

$$v(f) = \sum_{v \in S} \left(f^{\mathbf{out}}(v) - f^{\mathbf{in}}(v) \right)$$

Fact 1 - Rewriting the sums

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have $$v(f) = \sum_{v \in S} \left(f^{\textbf{out}}(v) - f^{\textbf{in}}(v) \right)$$
  - Let's recount, using the edges and the flow f(e).

Fact 1 - Rewriting the sums

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut.
  Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have $$v(f) = \sum_{v \in S} \left(f^{\textbf{out}}(v) - f^{\textbf{in}}(v) \right)$$
  - Let's recount, using the edges and the flow f(e).
    - If an edge *has both endpoints in S*, it is counted once for "out" and once for "in", so it contributes 0.

Fact 1 - Rewriting the sums

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have $$v(f) = \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v) \right)$$
  - Let's recount, using the edges and the flow f(e).
    - If an edge *has both endpoints in S*, it is counted once for "out" and once for "in", so it contributes 0.
    - If an edge *has its "tail" in S*, it is only counted for "out" and contributes 1.

Fact 1 - Rewriting the sums

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut.
  Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have
    $$v(f) = \sum_{v \in S} \left(f^{\mathbf{out}}(v) - f^{\mathbf{in}}(v) \right)$$
  - Let's recount, using the edges and the flow f(e).
    - If an edge *has both endpoints in S*, it is counted once for "out" and once for "in", so it contributes 0.
    - If an edge *has its "tail" in S*, it is only counted for "out" and contributes 1.
    - If an edge *has its "head" in S*, it is only counted for "in" and contributes -1.

Fact 1 - Rewriting the sums

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have $$v(f) = \sum_{v \in S} \left(f^{\textbf{out}}(v) - f^{\textbf{in}}(v) \right)$$
  - Let's recount, using the edges and the flow f(e).
    - If an edge *has both endpoints in S*, it is counted once for "out" and once for "in", so it contributes 0.
    - If an edge *has its "tail" in S*, it is only counted for "out" and contributes 1.
    - If an edge *has its "head" in S*, it is only counted for "in" and contributes -1.
    - Otherwise the edge does not appear in the sum.

Fact 1 - Rewriting the sums

- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have $$v(f) = \sum_{v \in S} \left(f^{\mathbf{out}}(v) - f^{\mathbf{in}}(v) \right)$$

Fact 1 - Rewriting the sums

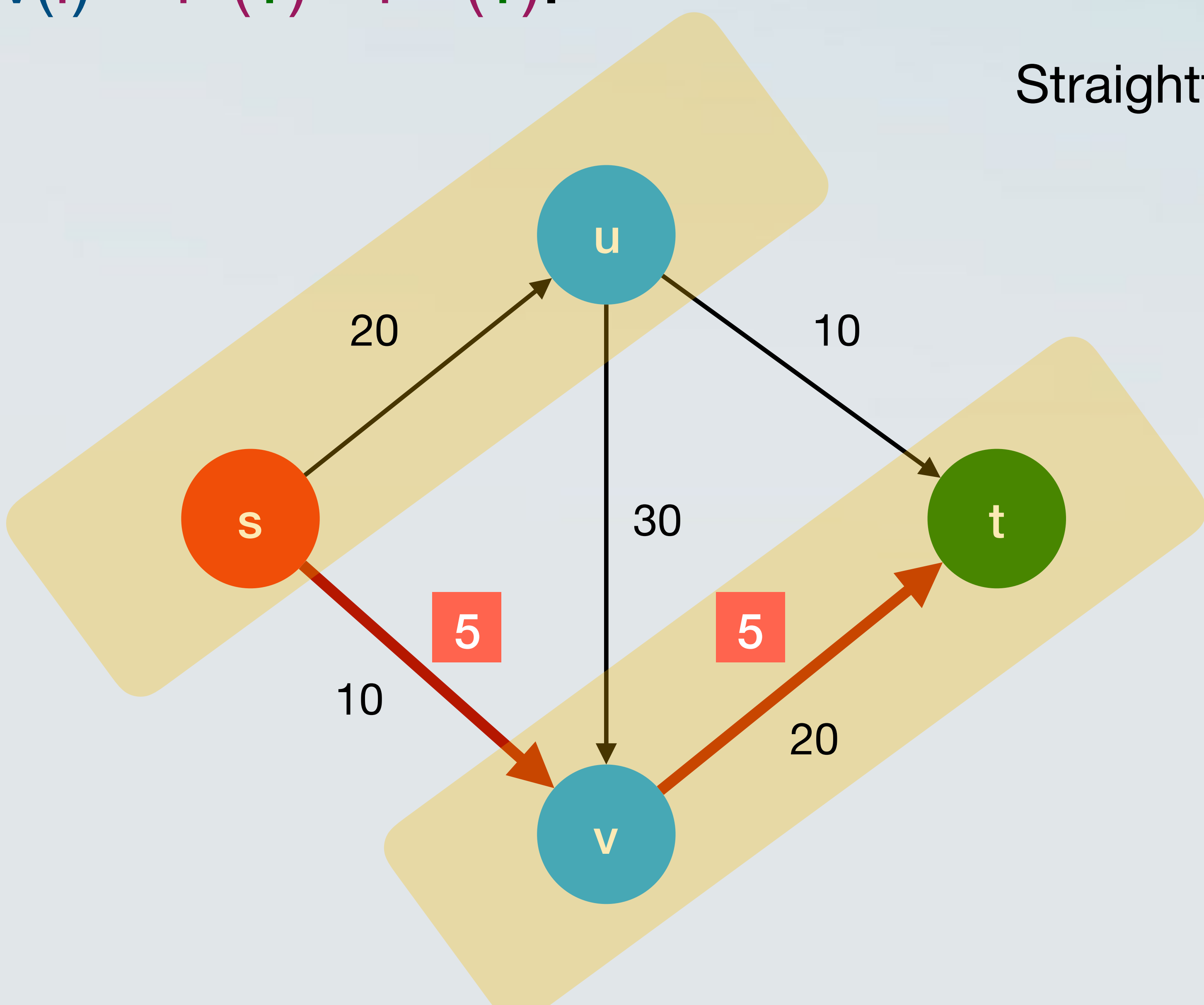
- Fact 1: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{out}(S) - f^{in}(S)$.
  - Therefore we have $$v(f) = \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v) \right)$$
  - We can write

$$v(f) = \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v) \right) = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ into } S} f(e) = f^{\text{out}}(S) - f^{\text{in}}(S)$$

A series of facts

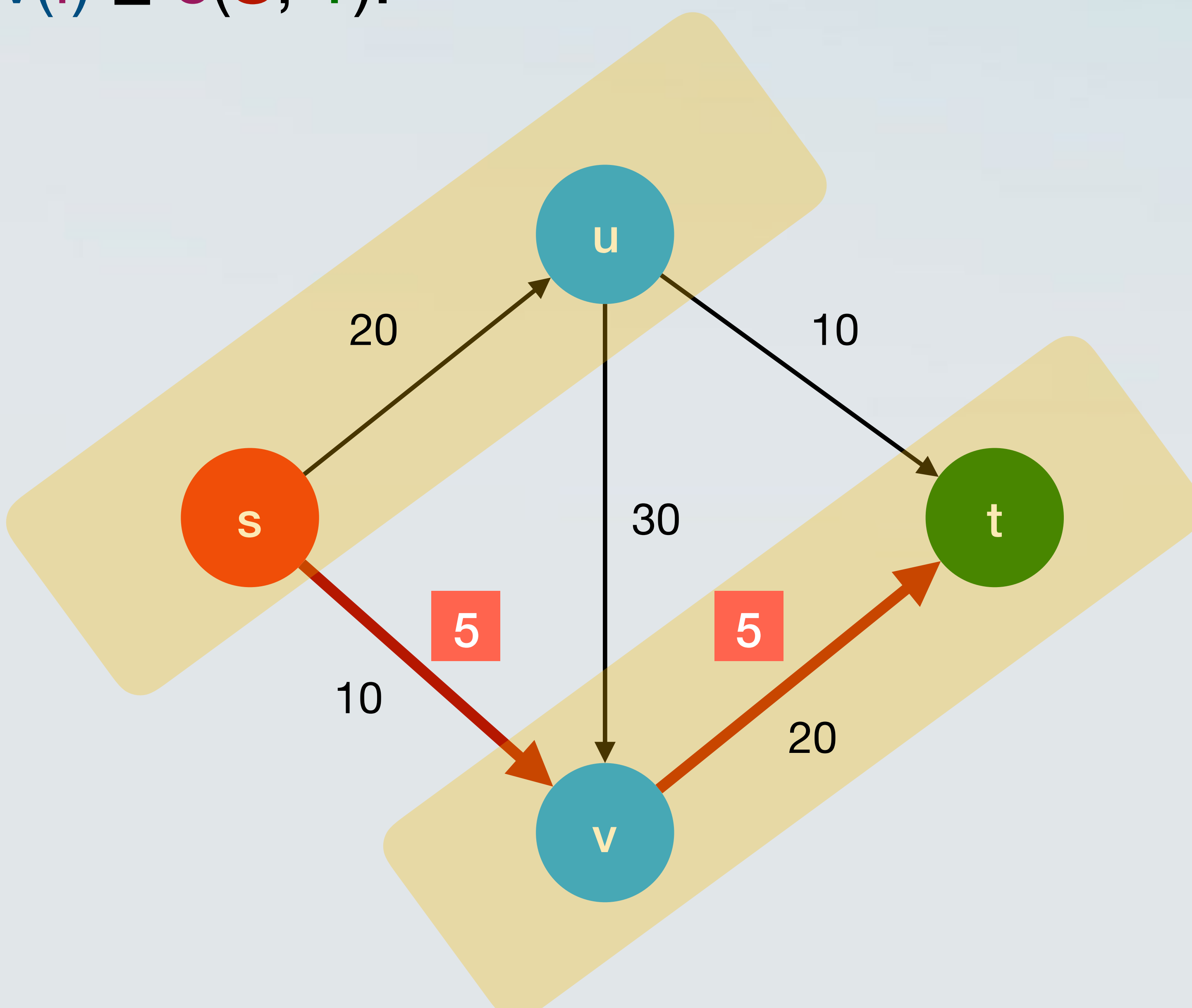
- Fact 2: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) = f^{in}(T) - f^{out}(T)$.

Straightforward by Fact 1.

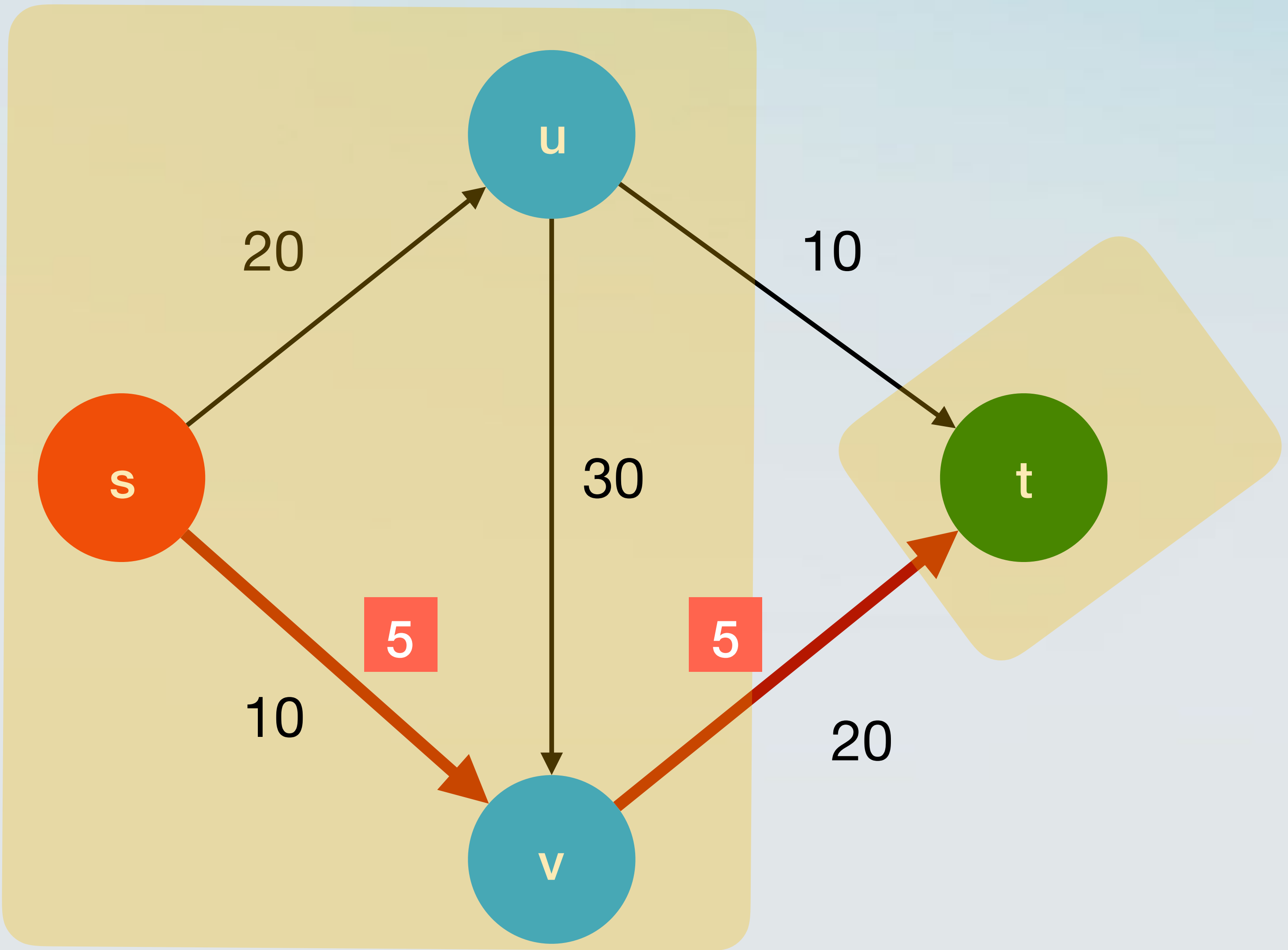


A series of facts

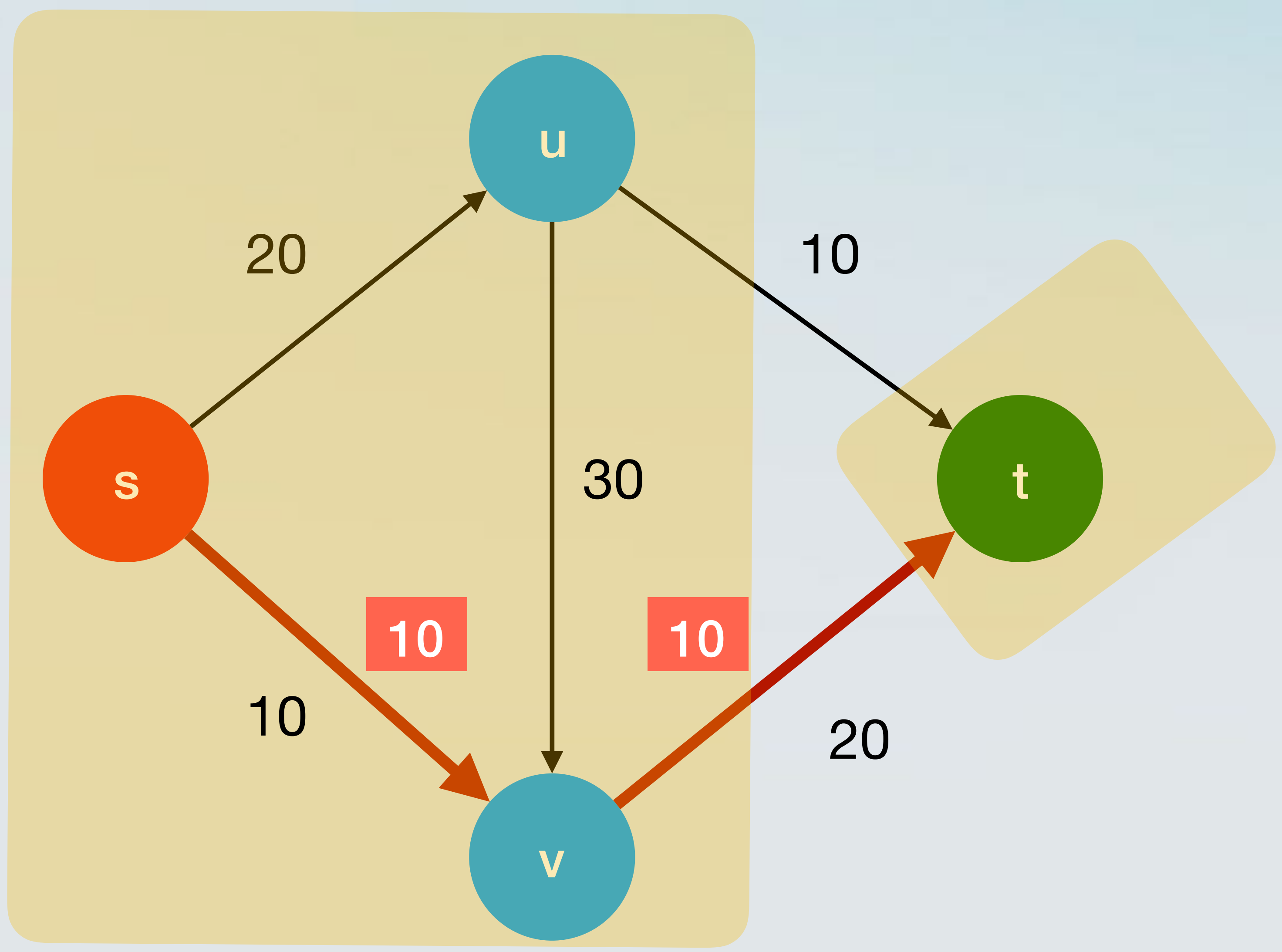
- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then v(f) ≤ c(S, T).



Another (s-t) cut



Another (s-t) cut



A series of facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) \leq c(S, T)$.

A series of facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then v(f) ≤ c(S, T).

$$
\begin{aligned}
v(f) &= f^{\mathbf{out}}(S) - f^{\mathbf{in}}(S) \\
&\leq f^{\mathbf{out}}(S) \\
&= \sum_{e \text{ out of } S} f(e) \\
&\leq \sum_{e \text{ out of } S} c_e \\
&= c(S, T)
\end{aligned}
$$

A series of facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then v(f) ≤ c(S, T).

$$
\begin{aligned}
v(f) &= f^{\mathbf{out}}(S) - f^{\mathbf{in}}(S) && \text{by Fact 1} \\
&\leq f^{\mathbf{out}}(S) \\
&= \sum_{e \text{ out of } S} f(e) \\
&\leq \sum_{e \text{ out of } S} c_e \\
&= c(S, T)
\end{aligned}
$$

A series of facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then v(f) ≤ c(S, T).

$$\begin{aligned} v(f) &= f^{\mathbf{out}}(S) - f^{\mathbf{in}}(S) && \text{by Fact 1} \\ &\leq f^{\mathbf{out}}(S) && \text{straightforward} \\ &= \sum_{e \text{ out of } S} f(e) \\ &\leq \sum_{e \text{ out of } S} c_e \\ &= c(S, T) \end{aligned}$$

A series of facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then v(f) ≤ c(S, T).

$$
\begin{aligned}
v(f) &= f^{\mathbf{out}}(S) - f^{\mathbf{in}}(S) && \text{by Fact 1} \\
&\le f^{\mathbf{out}}(S) && \text{straightforward} \\
&= \sum_{e \text{ out of } S} f(e) && \text{by definition} \\
&\le \sum_{e \text{ out of } S} c_e \\
&= c(S, T)
\end{aligned}
$$

A series of facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then v(f) ≤ c(S, T).

$$
\begin{aligned}
v(f) &= f^{\mathbf{out}}(S) - f^{\mathbf{in}}(S) && \text{by Fact 1} \\
&\leq f^{\mathbf{out}}(S) && \text{straightforward} \\
&= \sum_{e \text{ out of } S} f(e) && \text{by definition} \\
&\leq \sum_{e \text{ out of } S} c_e && \text{by capacity constraint} \\
&= c(S, T)
\end{aligned}
$$

A series of facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then v(f) ≤ c(S, T).

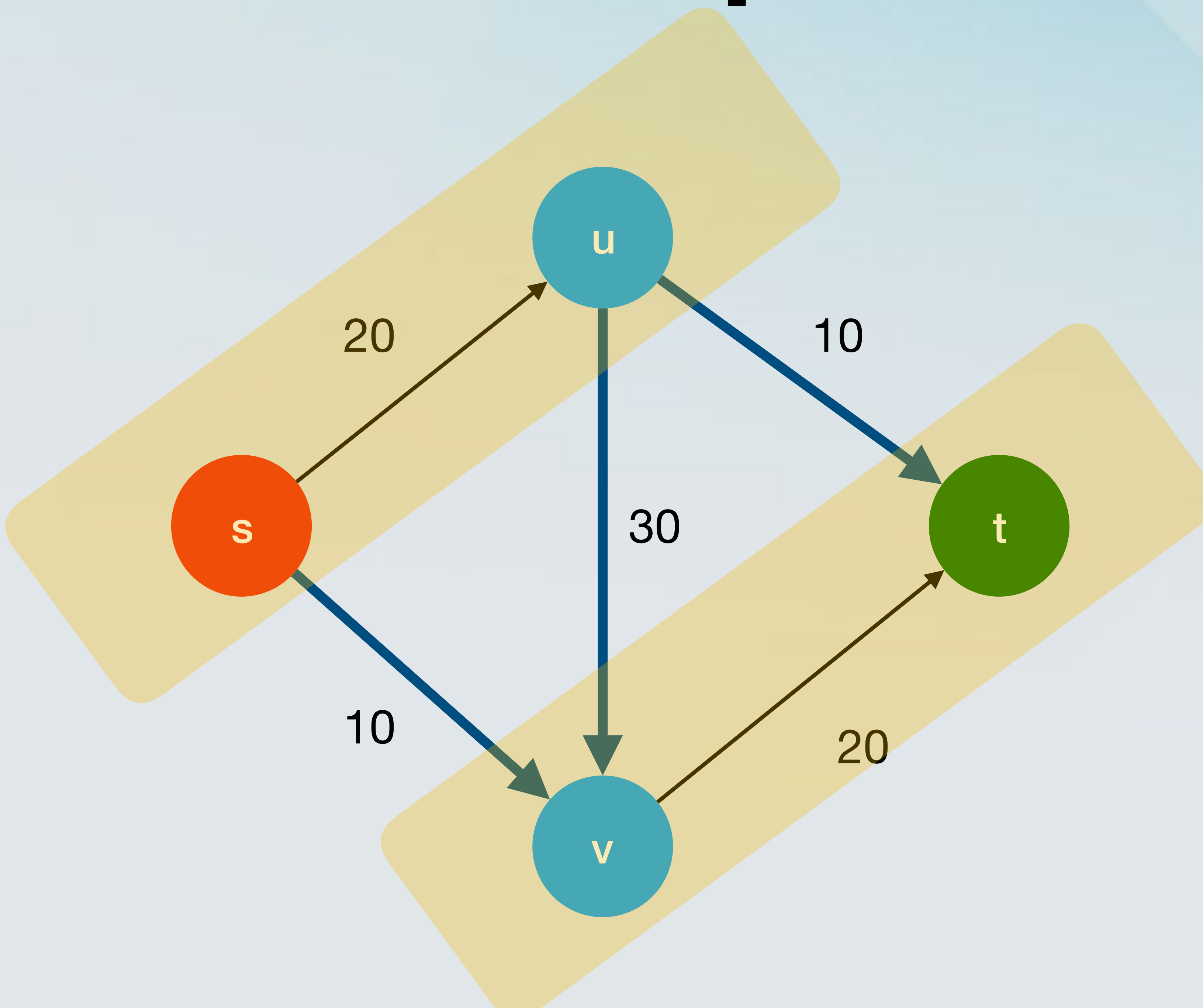
$$
\begin{aligned}
v(f) &= f^{\mathbf{out}}(S) - f^{\mathbf{in}}(S) && \text{by Fact 1} \\
&\leq f^{\mathbf{out}}(S) && \text{straightforward} \\
&= \sum_{e \text{ out of } S} f(e) && \text{by definition} \\
&\leq \sum_{e \text{ out of } S} c_e && \text{by capacity constraint} \\
&= c(S, T) && \text{by definition}
\end{aligned}
$$

Comparing facts

- Fact 3: Let f by any (s-t) flow and (S, T) be any (s-t) cut. Then $v(f) \leq c(S, T)$.

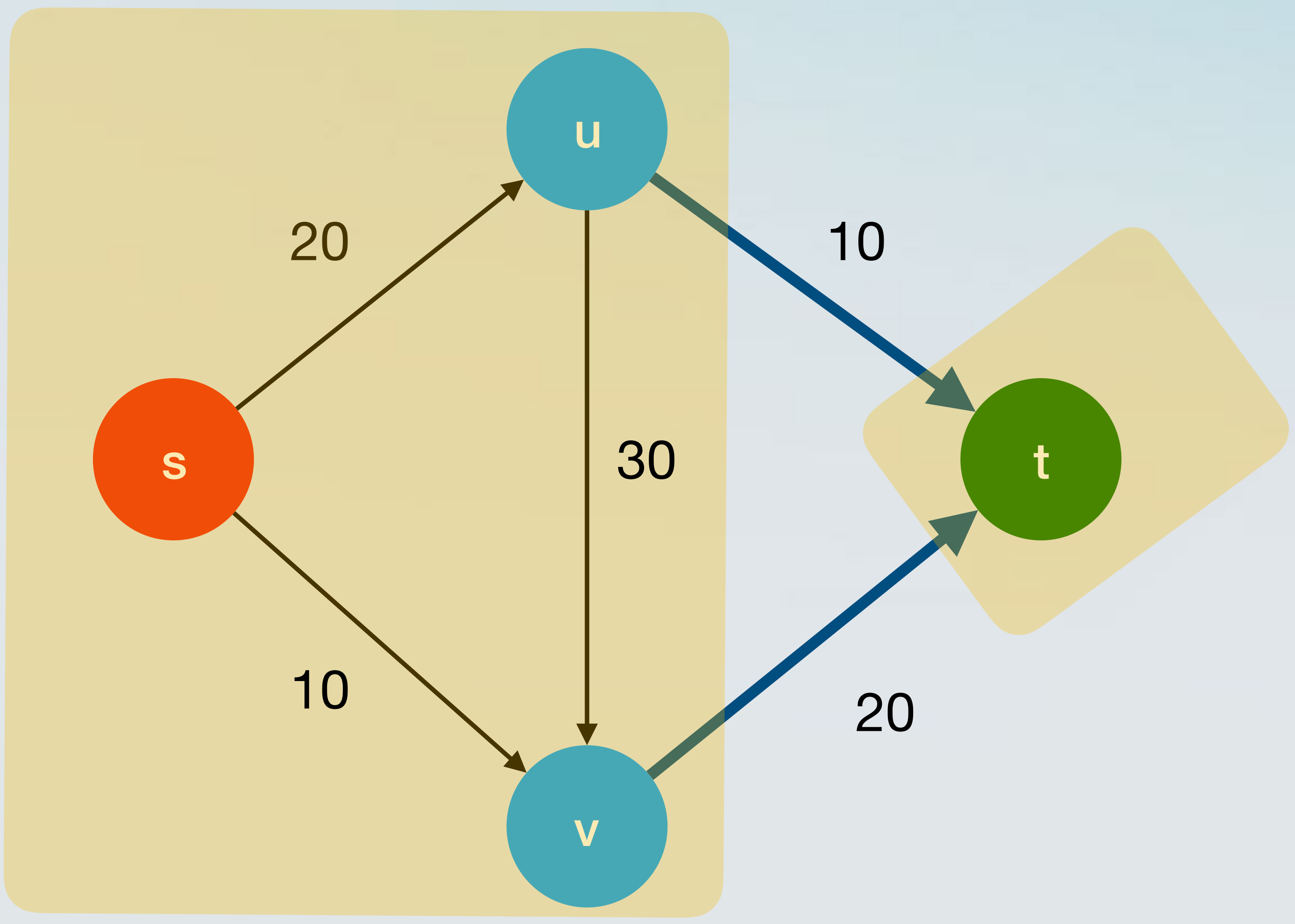
- Theorem: In every flow network, the value of the maximum flow is *equal* to the capacity of the minimum cut.

Example



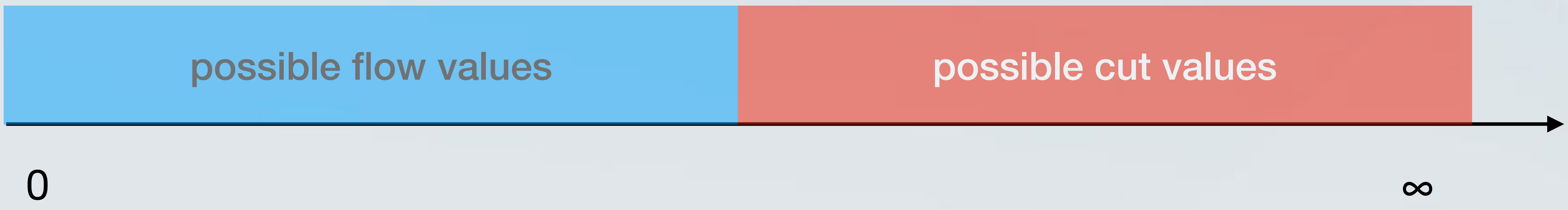
What can we safely say about the maximum flow?

Example

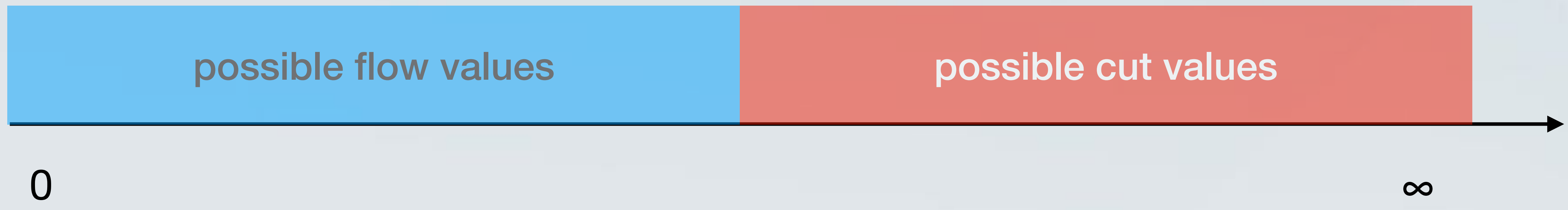


What can we safely say about the maximum flow?

Proof idea

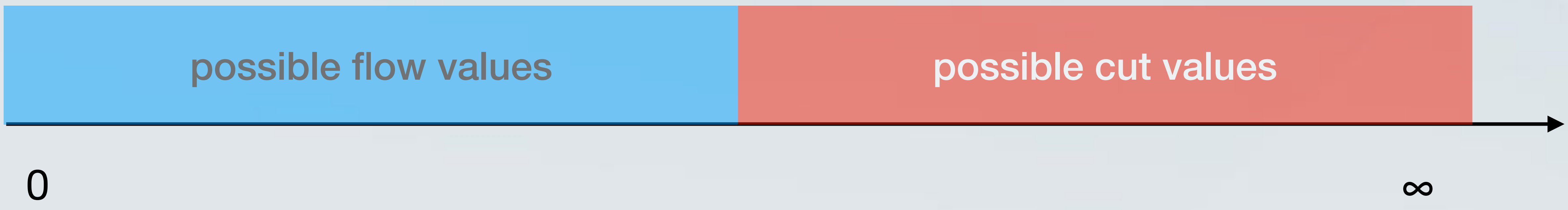


Proof idea



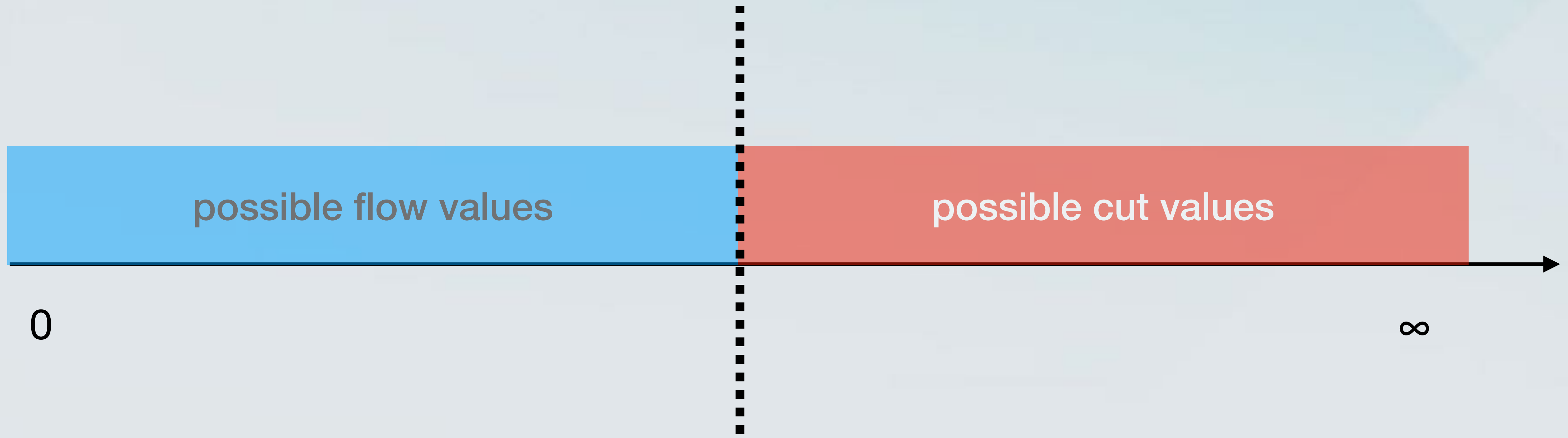
- How can we prove that a flow f* is maximum?

Proof idea



- How can we prove that a flow f^* is maximum?
- Find a cut with capacity $c = f^*$.

Proof idea



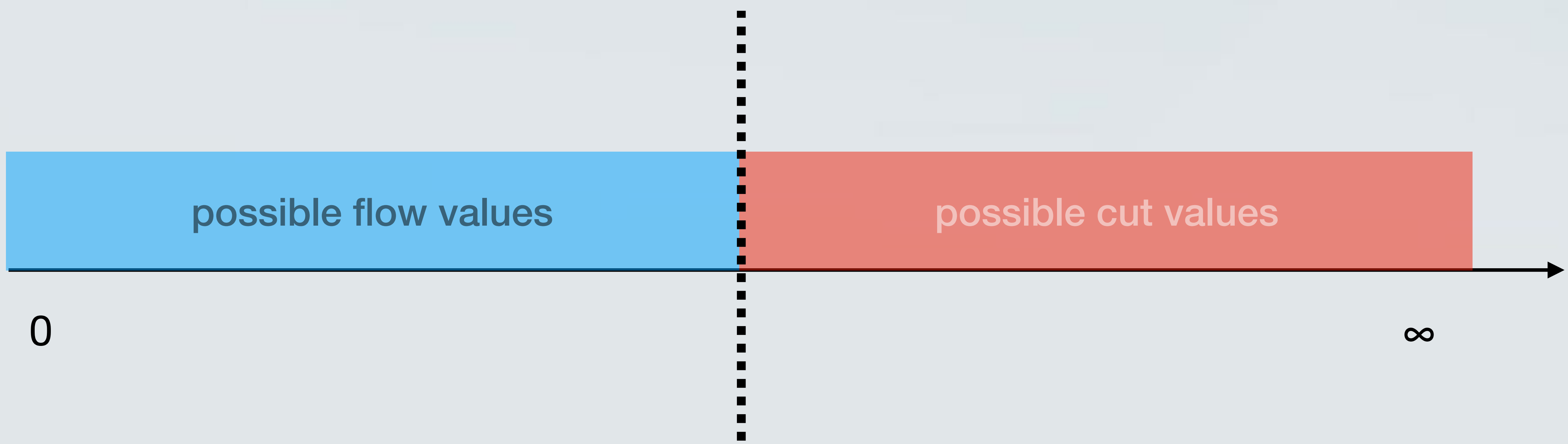
- How can we prove that a flow f* is maximum?
- Find a cut with capacity c = f*.

A series of facts

- Fact 4: Let f by any (s-t) flow in G such that the residual graph G_f has no *augmenting paths*. Then there is an (s-t) cut C(S*, T*) in G such that c(S*, T*) = v(f).

A series of facts

- Fact 4: Let f by any (s-t) flow in G such that the residual graph G_f has no *augmenting paths*. Then there is an (s-t) cut C(S*, T*) in G such that c(S*, T*) = v(f).



Constructing the cut

Constructing the cut

- In the residual graph G_f, identify the nodes that are reachable from the source s.

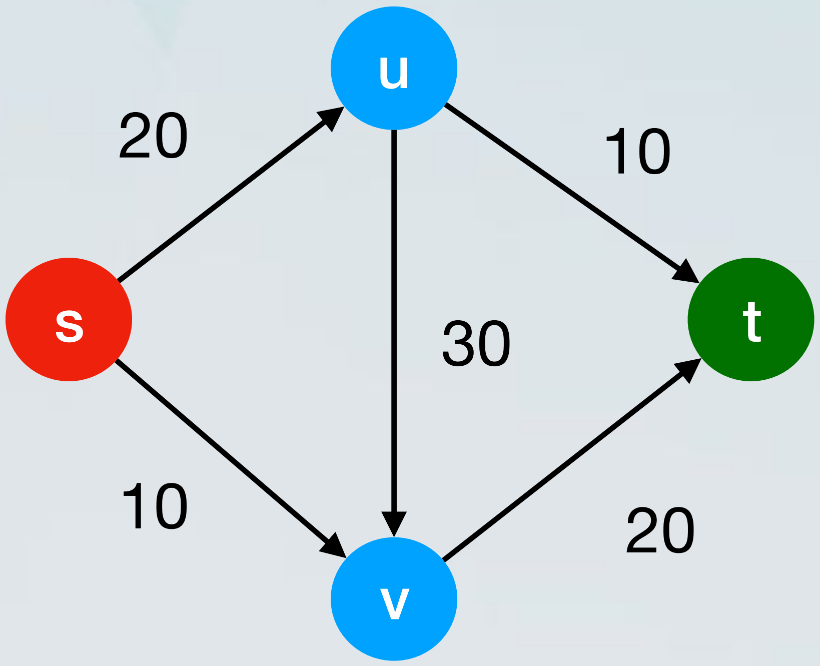
Constructing the cut

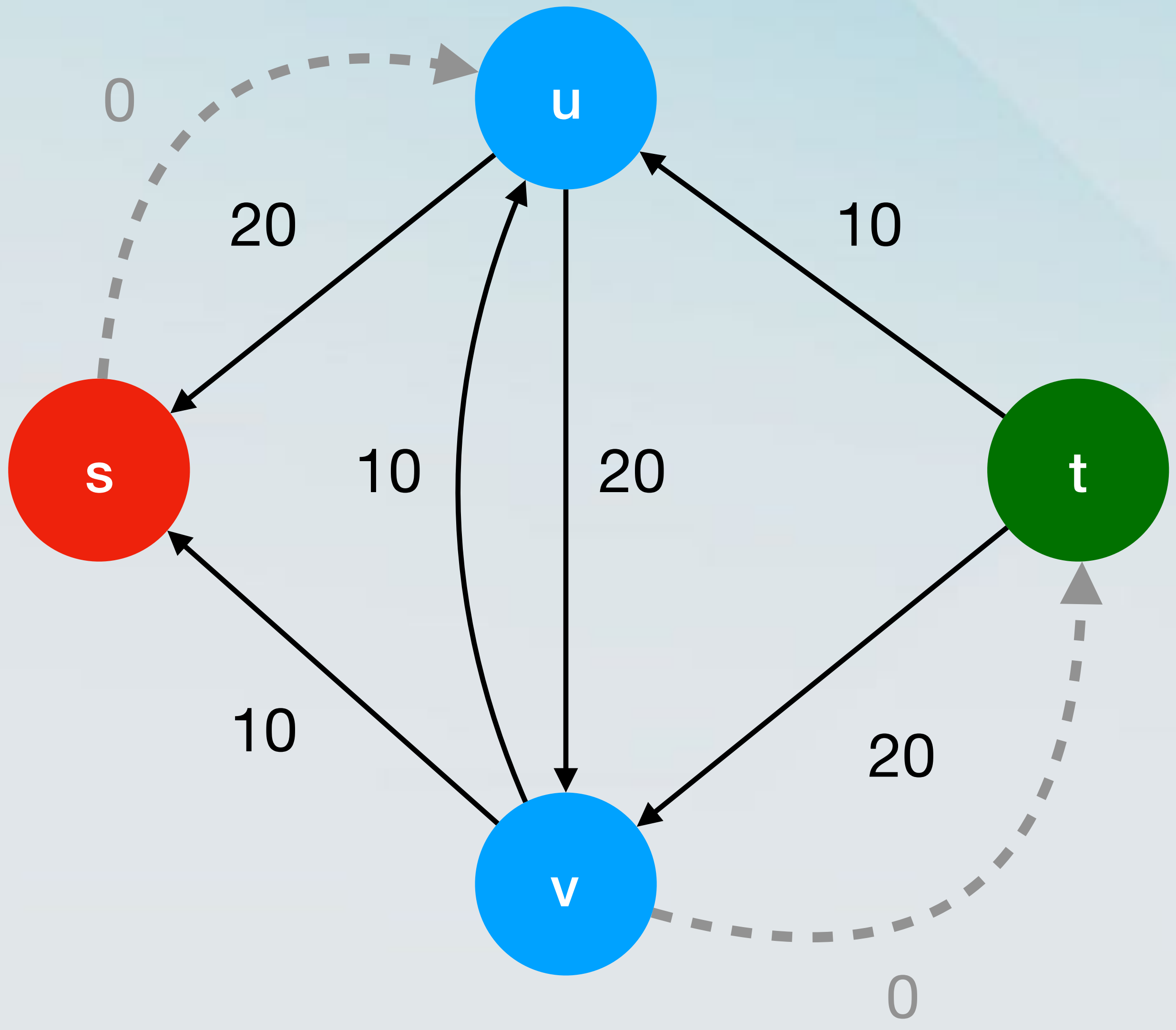
- In the residual graph G_f, identify the nodes that are reachable from the source s.
    - Put these in S^*.

Constructing the cut

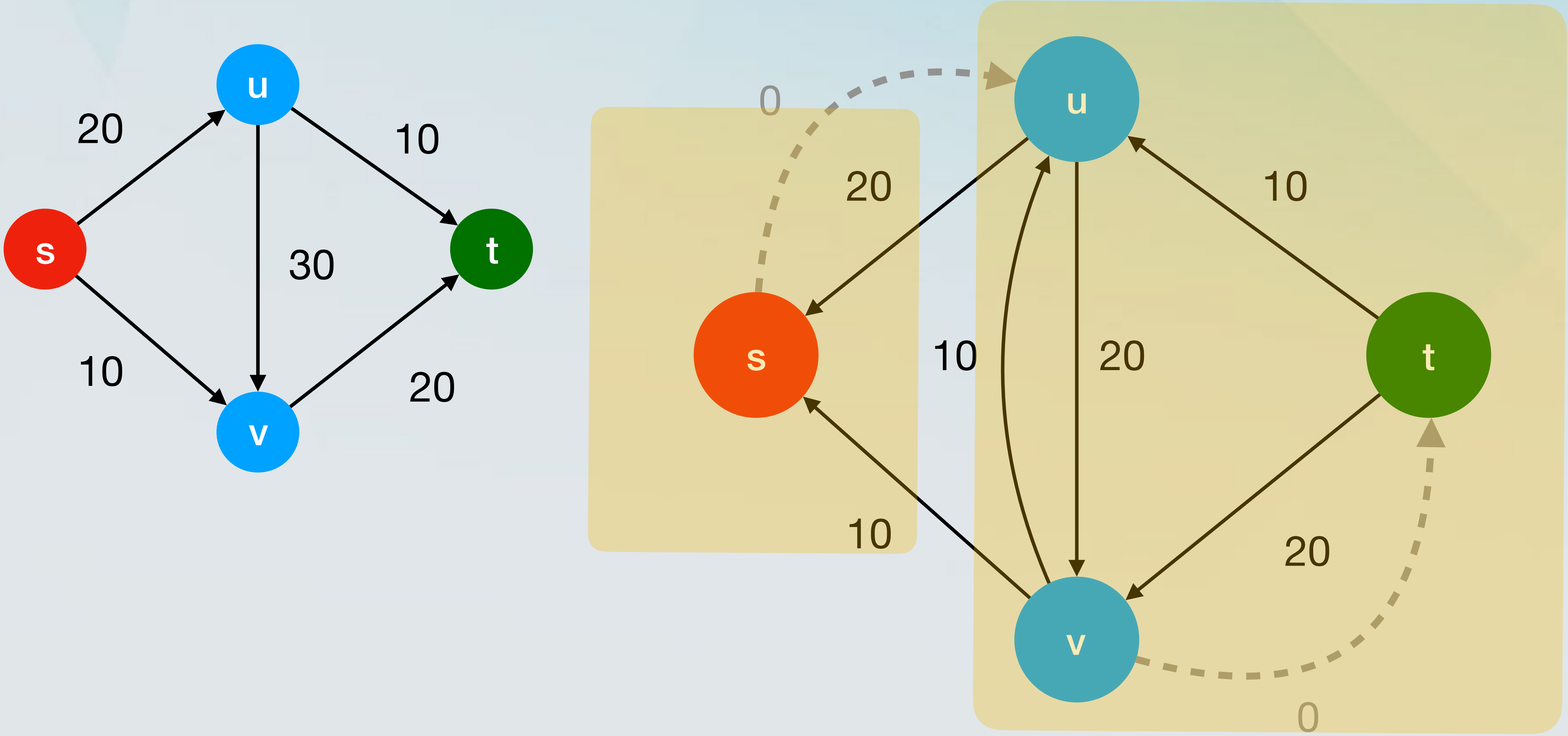
- In the residual graph G_f, identify the nodes that are reachable from the source s.
    - Put these in S^*.
    - Put the rest in T^*.

Example





Example



Proving Fact 4

- In the residual graph G_f, identify the nodes that are reachable from the source s.
  - Put these in S*.
  - Put the rest in T*.

Proving Fact 4

- In the residual graph G_f, identify the nodes that are reachable from the source s.
  - Put these in S*.
  - Put the rest in T*.
- Is this a cut?

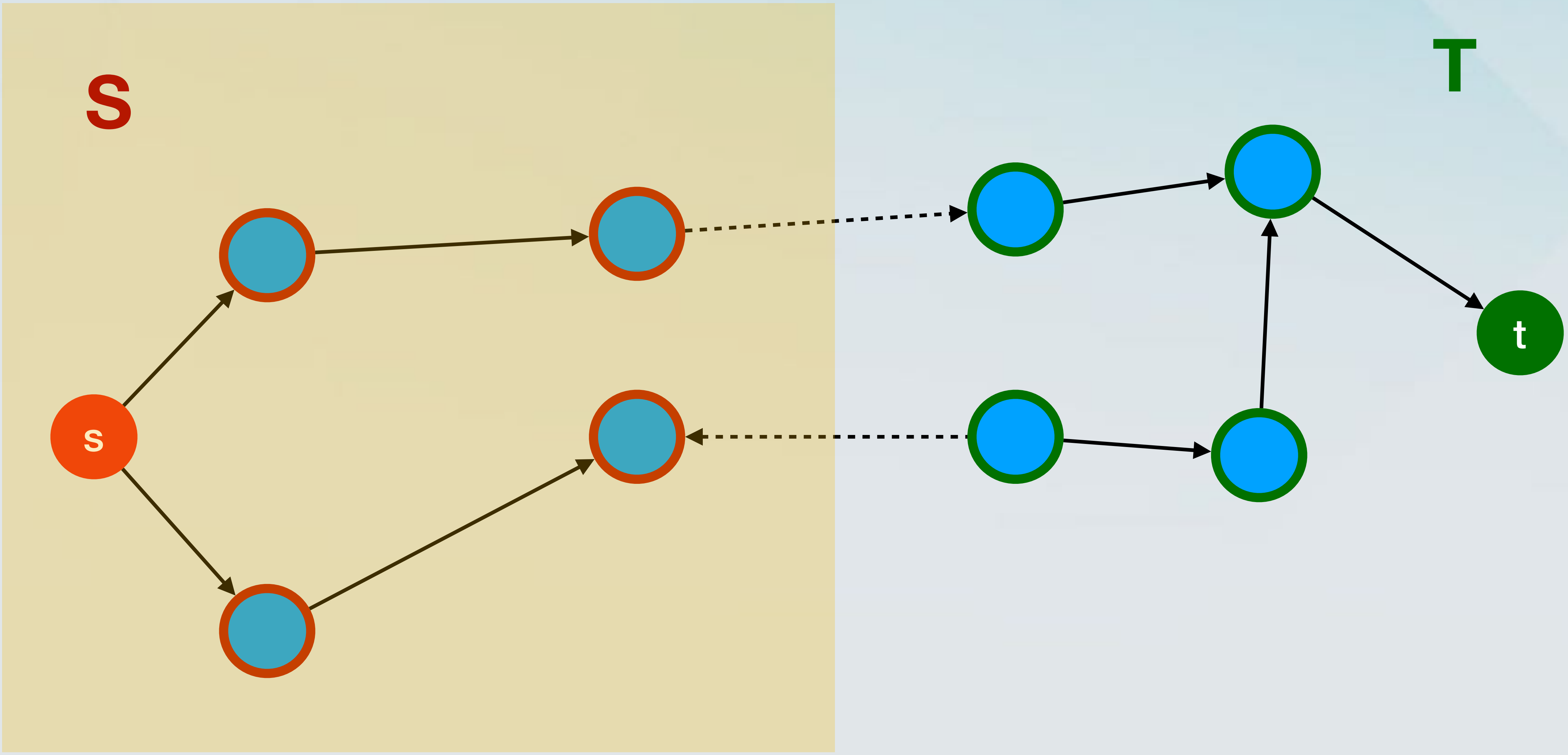
Proving Fact 4

- In the residual graph G_f, identify the nodes that are reachable from the source s.
  - Put these in S*.
  - Put the rest in T*.
- Is this a cut?
  - s is in S*.

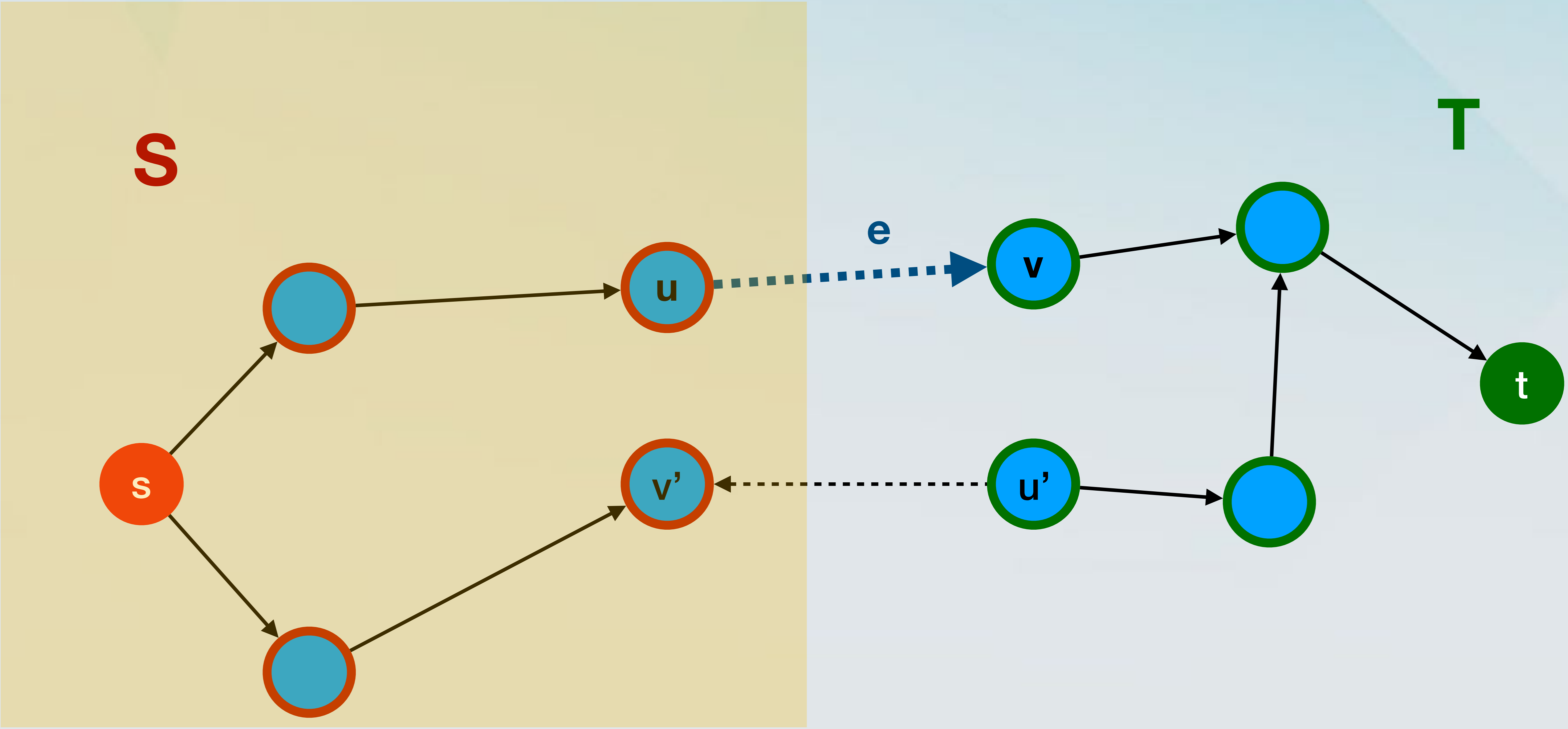
Proving Fact 4

- In the residual graph G_f, identify the nodes that are reachable from the source s.
  - Put these in S*.
  - Put the rest in T*.
- Is this a cut?
  - s is in S*.
  - t is in T* (why?).

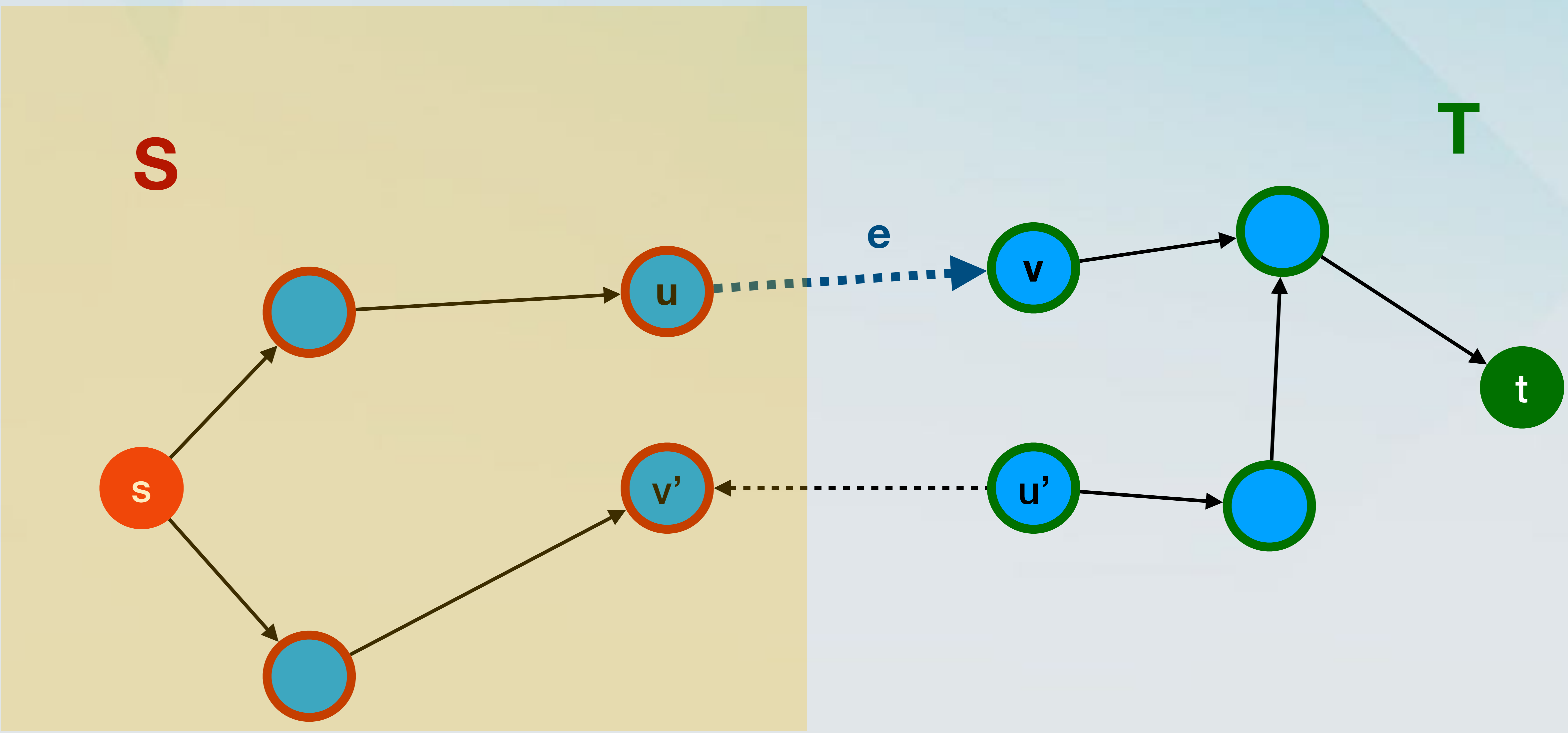
Proving Fact 4



Proving Fact 4

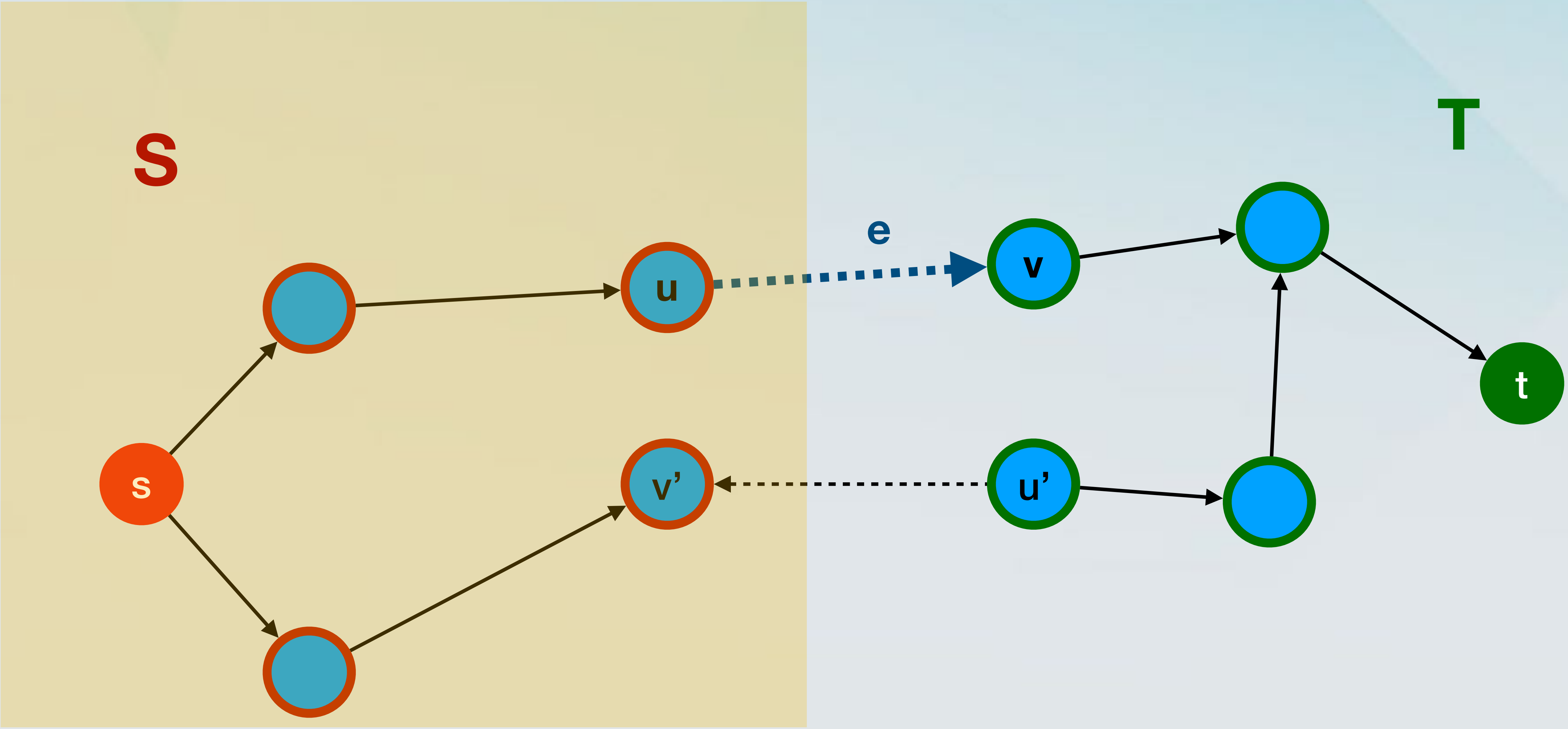


Proving Fact 4



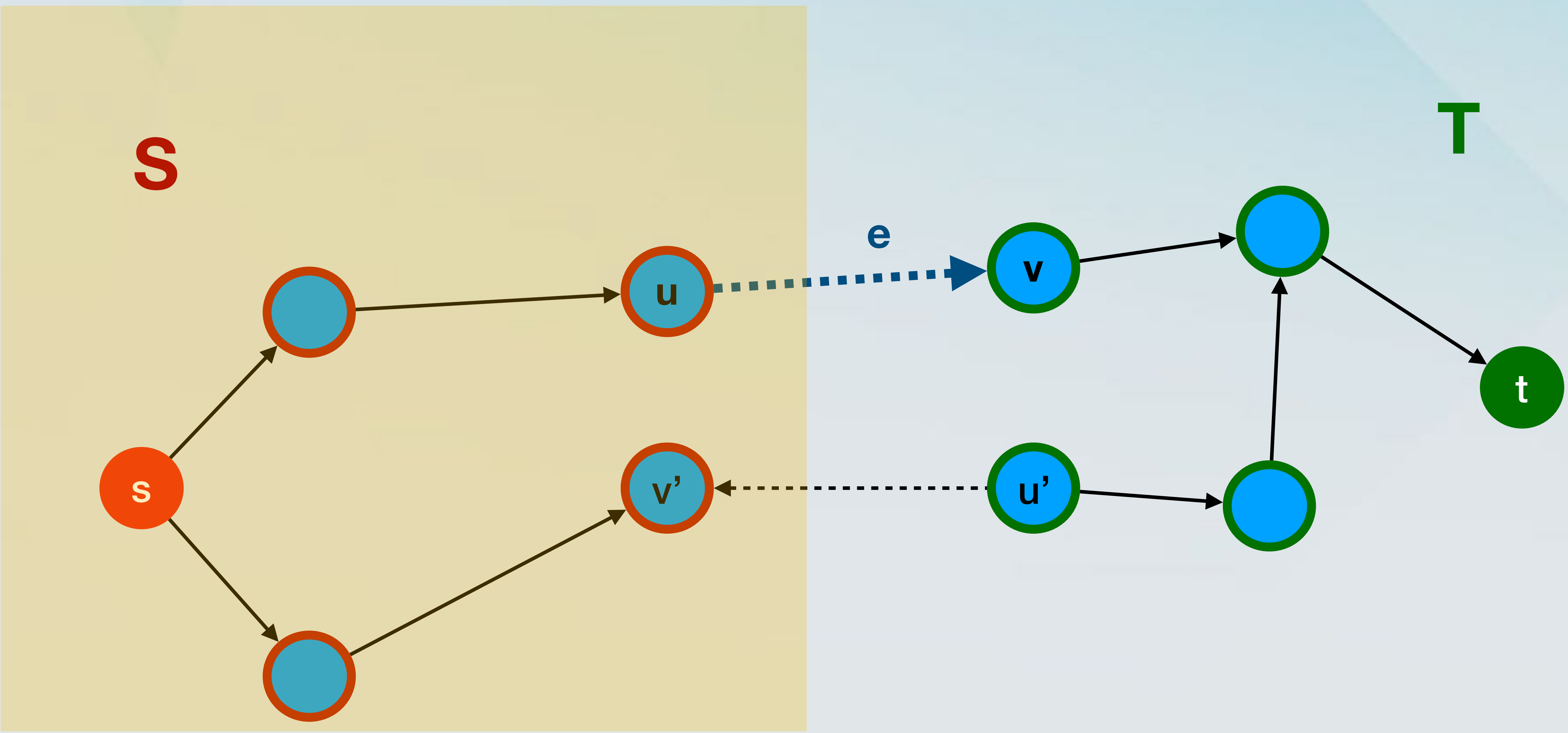
- Claim: **in G , f(e)** = c_e (i.e., e **in G** is *saturated* by the flow f).

Proving Fact 4



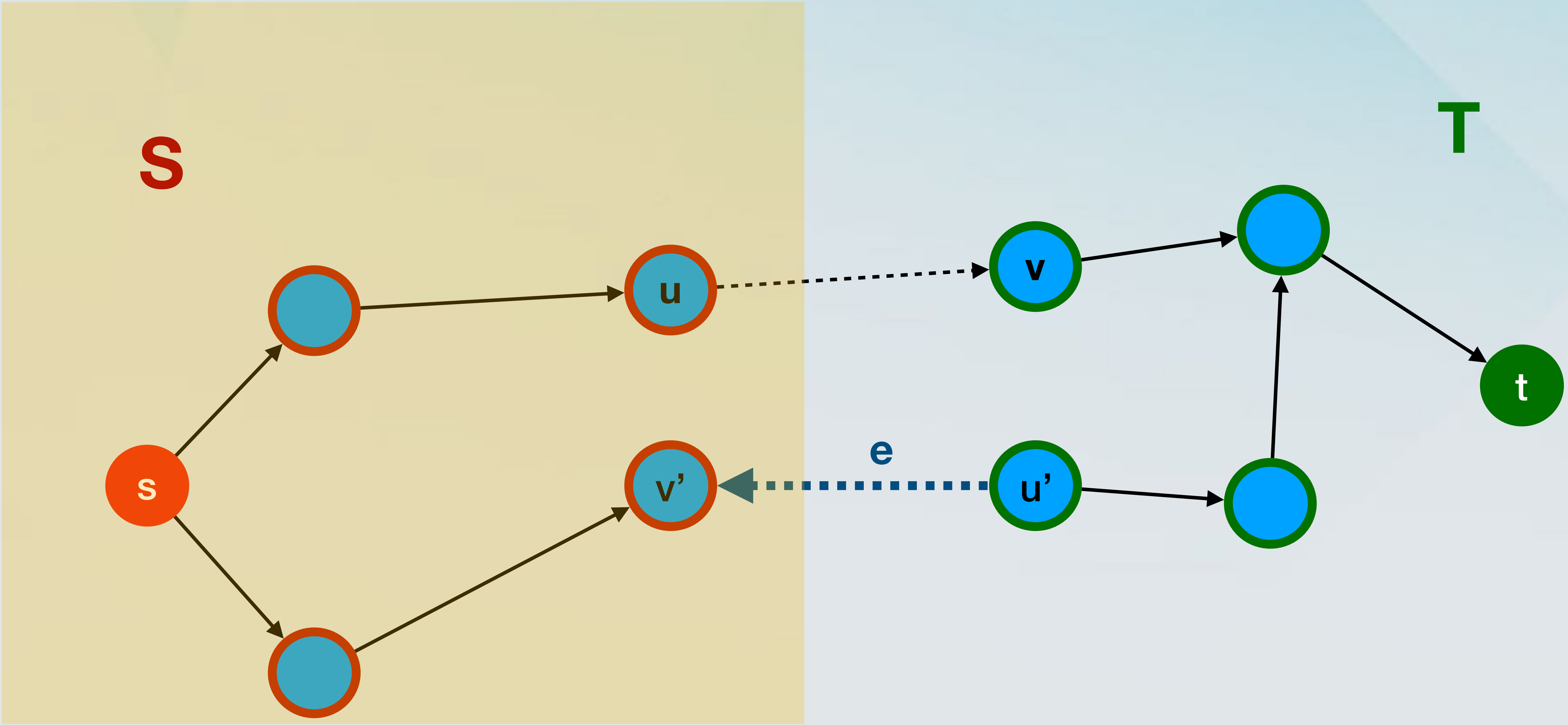
- Claim: **in G ,** $f(e) = c_e$ (i.e., e **in G** is *saturated* by the flow f).
  - If not, e would be a *forward edge* in G_f.

Proving Fact 4

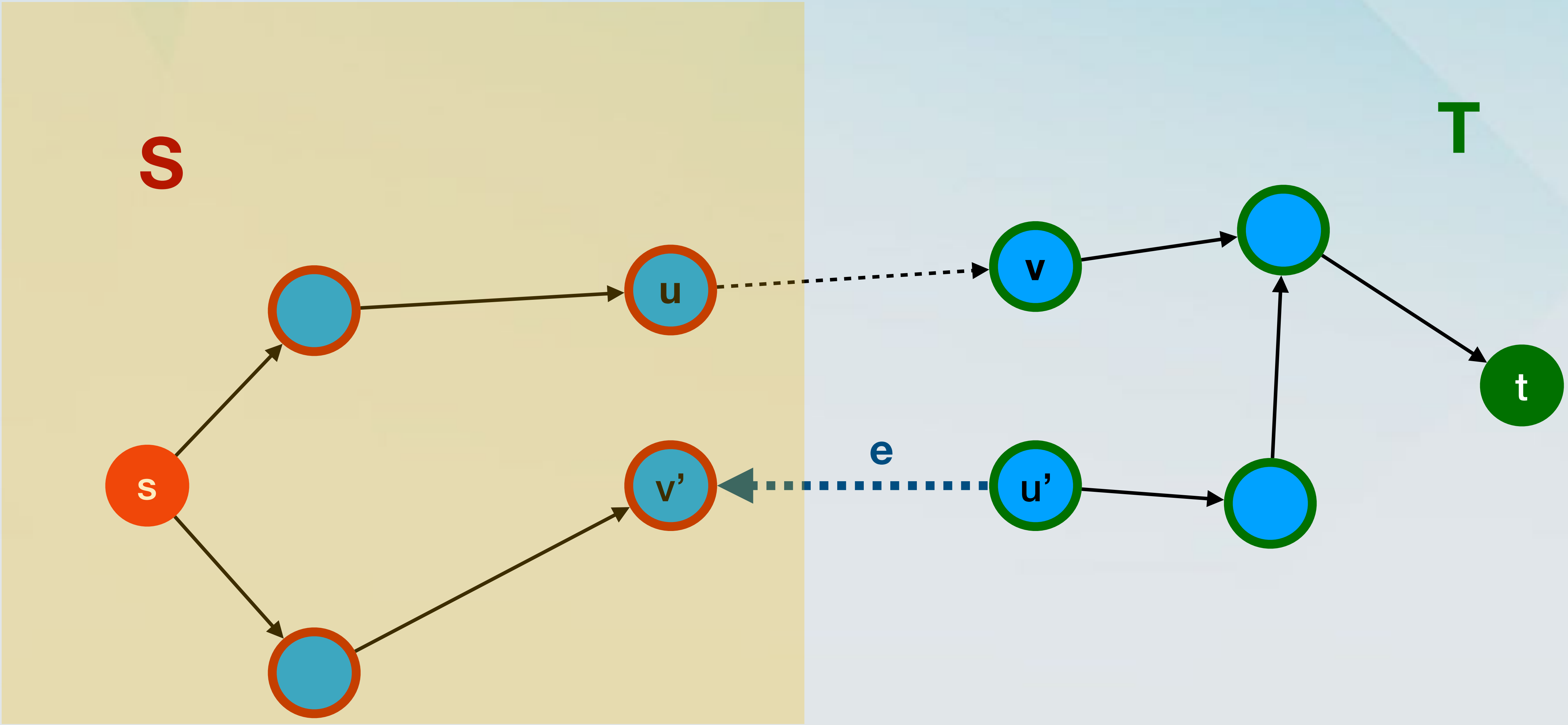


- Claim: **in G ,** f(e) = c_e (i.e., e **in G** is *saturated* by the flow f).
  - If not, e would be a *forward edge* in G_f.
  - There would exist a path (s, v).

Proving Fact 4

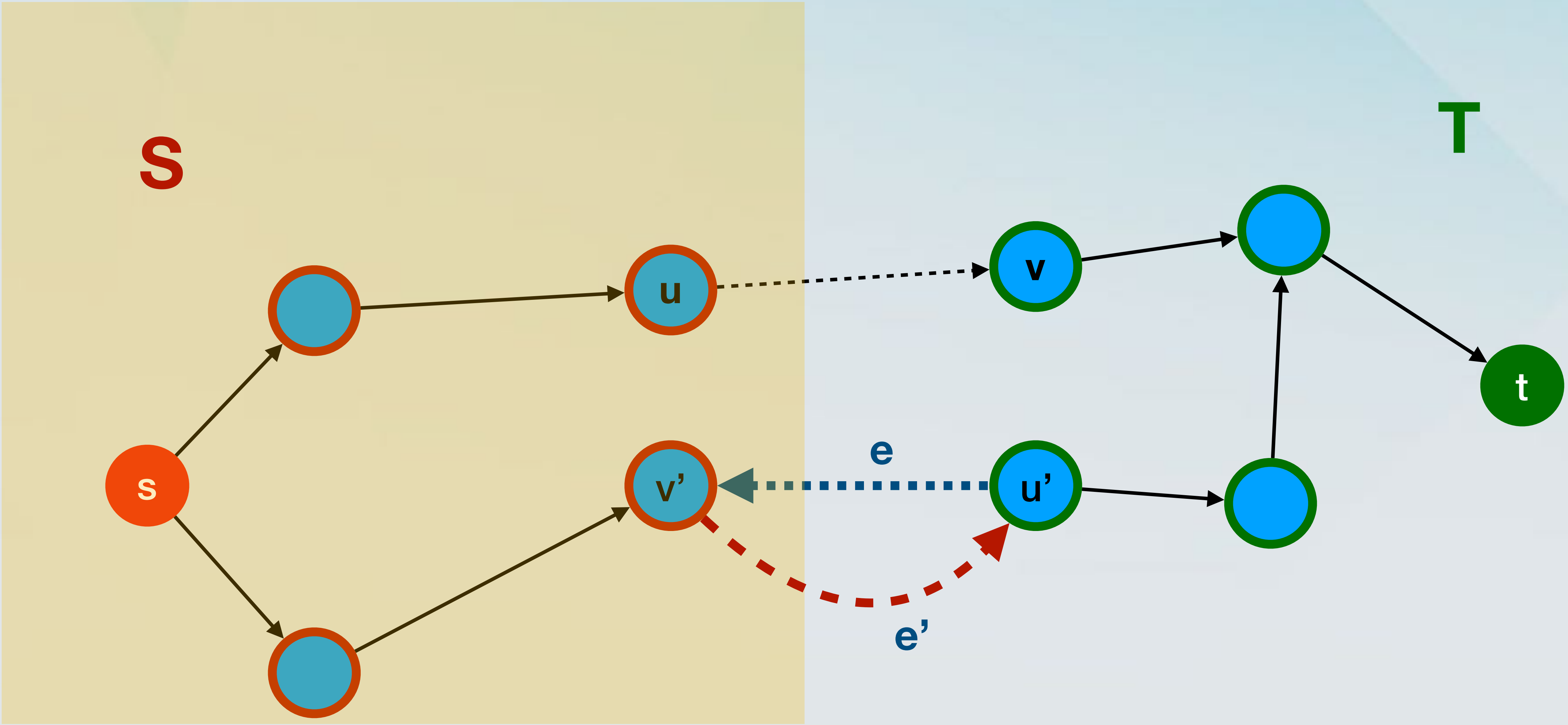


Proving Fact 4



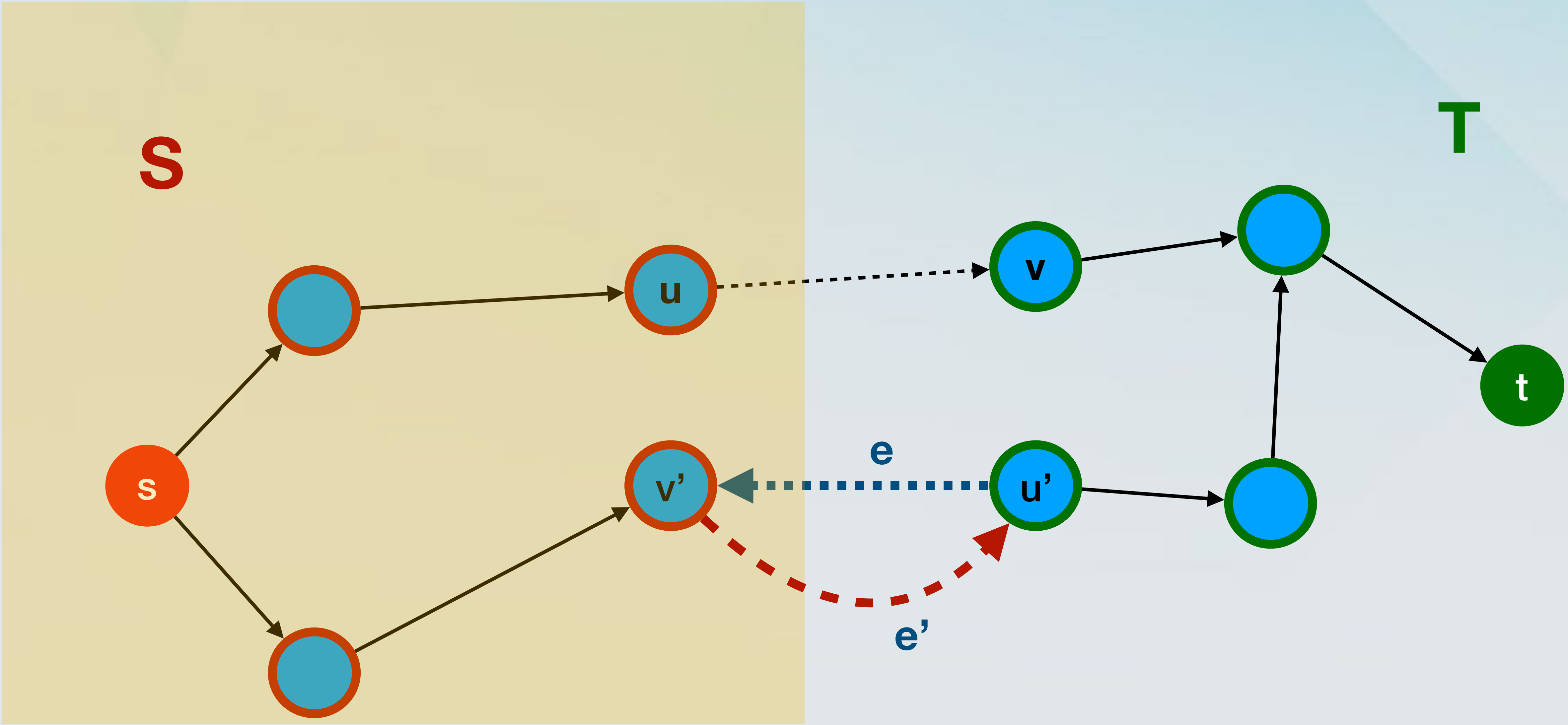
- Claim: **in G ,** f(e) = 0.

Proving Fact 4



- Claim: **in G ,** f(e) = 0.
  - If not, e would generate a *backward edge e'* in G_f.

Proving Fact 4



- Claim: **in G ,** f(e) = 0.
  - If not, e would generate a *backward edge e'* in G_f.
  - There would exist a path (s, u').

Proving Fact 4

- What do we get from this?
  - All edges *out of* S* are *saturated* by f.
  - All edges *into* S* *have 0 flow* in f.

Proving Fact 4

- What do we get from this?
  - All edges *out of* S* are *saturated* by f.
  - All edges *into* S* *have 0 flow* in f.

$$
\begin{aligned}
v(f) &= f^{\mathbf{out}}(S^*) - f^{\mathbf{in}}(S^*) \\
&= \sum_{e \text{ out of } S^*} f(e) - \sum_{e \text{ into } S^*} f(e) \\
&= \sum_{e \text{ out of } S} c_e - 0 \\
&= c(S^*, T^*)
\end{aligned}
$$

Putting everything together

- Fact 4: Let f by any (s-t) flow in G such that the residual graph G_f has no *augmenting paths*. Then there is an (s-t) cut C(S*, T*) in G such that c(S*, T*) = v(f).

Putting everything together

- Fact 4: Let f by any (s-t) flow in G such that the residual graph G_f has no *augmenting paths*. Then there is an (s-t) cut C(S*, T*) in G such that c(S*, T*) = v(f).

- Ford-Fulkerson stops when there are no augmenting paths in the residual network.

Putting everything together

- Fact 4: Let f by any (s-t) flow in G such that the residual graph G_f has no *augmenting paths*. Then there is an (s-t) cut C(S*, T*) in G such that c(S*, T*) = v(f).
- Ford-Fulkerson stops when there are no augmenting paths in the residual network.
- The value of the flow is equal to the capacity of *some* cut.

Putting everything together

- Fact 4: Let f by any (s-t) flow in G such that the residual graph G_f has no *augmenting paths*. Then there is an (s-t) cut $C(S^*, T^*)$ in G such that $c(S^*, T^*) = v(f)$.

- Ford-Fulkerson stops when there are no augmenting paths in the residual network.

- The value of the flow is equal to the capacity of *some* cut.

- This means that the value of the flow is maximum.

Related question

Related question

- How do we find *the value of the minimum cut* in a flow network?

Related question

- How do we find *the value of the minimum cut* in a flow network?

  - Run Ford-Fulkerson and output the value of the computed flow.

Related question

- How do we find *the value of the minimum cut* in a flow network?

    - Run Ford-Fulkerson and output the value of the computed flow.

- How do we find *a minimum cut* in a flow network?

Related question

- How do we find *the value of the minimum cut* in a flow network?
    - Run Ford-Fulkerson and output the value of the computed flow.
- How do we find *a minimum cut* in a flow network?
    - Run Ford-Fulkerson and look at the final residual graph.

Related question

- How do we find *the value of the minimum cut* in a flow network?
  - Run Ford-Fulkerson and output the value of the computed flow.
- How do we find *a minimum cut* in a flow network?
  - Run Ford-Fulkerson and look at the final residual graph.
  - Put the nodes reachable from s to S and the remaining nodes to T.

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.
  - The proof of the theorem follows from the proof of optimality for Ford-Fulkerson!

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?

Integer-Valued Flows

- Fact 5: If all the capacities in the flow network are integers, there is maximum flow for which every flow value f(e) is an integer.

Integer-Valued Flows

- Fact 5: If all the capacities in the flow network are integers, there is maximum flow for which every flow value f(e) is an integer.
  - This follows from the properties of the Ford-Fulkerson algorithm.

Integer-Valued Flows

- Fact 5: If all the capacities in the flow network are integers, there is maximum flow for which every flow value f(e) is an integer.
  - This follows from the properties of the Ford-Fulkerson algorithm.
    - It produces a maximum flow.

Integer-Valued Flows

- Fact 5: If all the capacities in the flow network are integers, there is maximum flow for which every flow value f(e) is an integer.
  - This follows from the properties of the Ford-Fulkerson algorithm.
    - It produces a maximum flow.
    - The capacities and flows are integers in every step of the execution.

Back to the 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?

Back to the running time

- The running time of *FF* is **O(**mF**)**, where F is the value of the maximum flow.

- Is this a polynomial time algorithm?

  - It runs in *pseudo-polynomial* time.

Back to the 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?
    - It runs in *pseudo-polynomial* time.
    - Should we be happy about this?

Back to the 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?
  - It runs in *pseudo-polynomial* time.
  - Should we be happy about this?
  - Is this problem NP-hard?

The Ford-Fulkerson Algorithm

Max-Flow

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

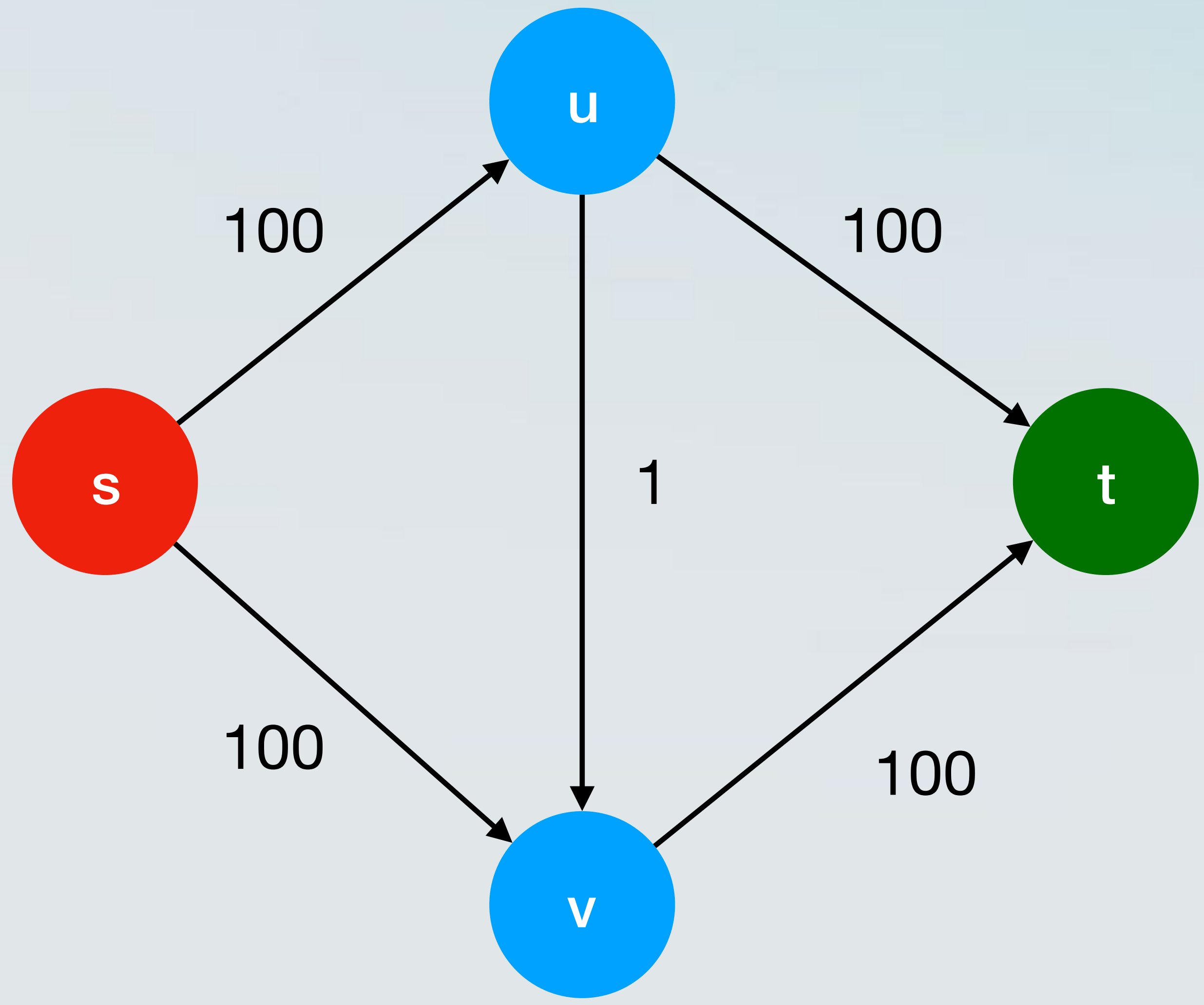
While there exists an s-t path in the residual graph Gf

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

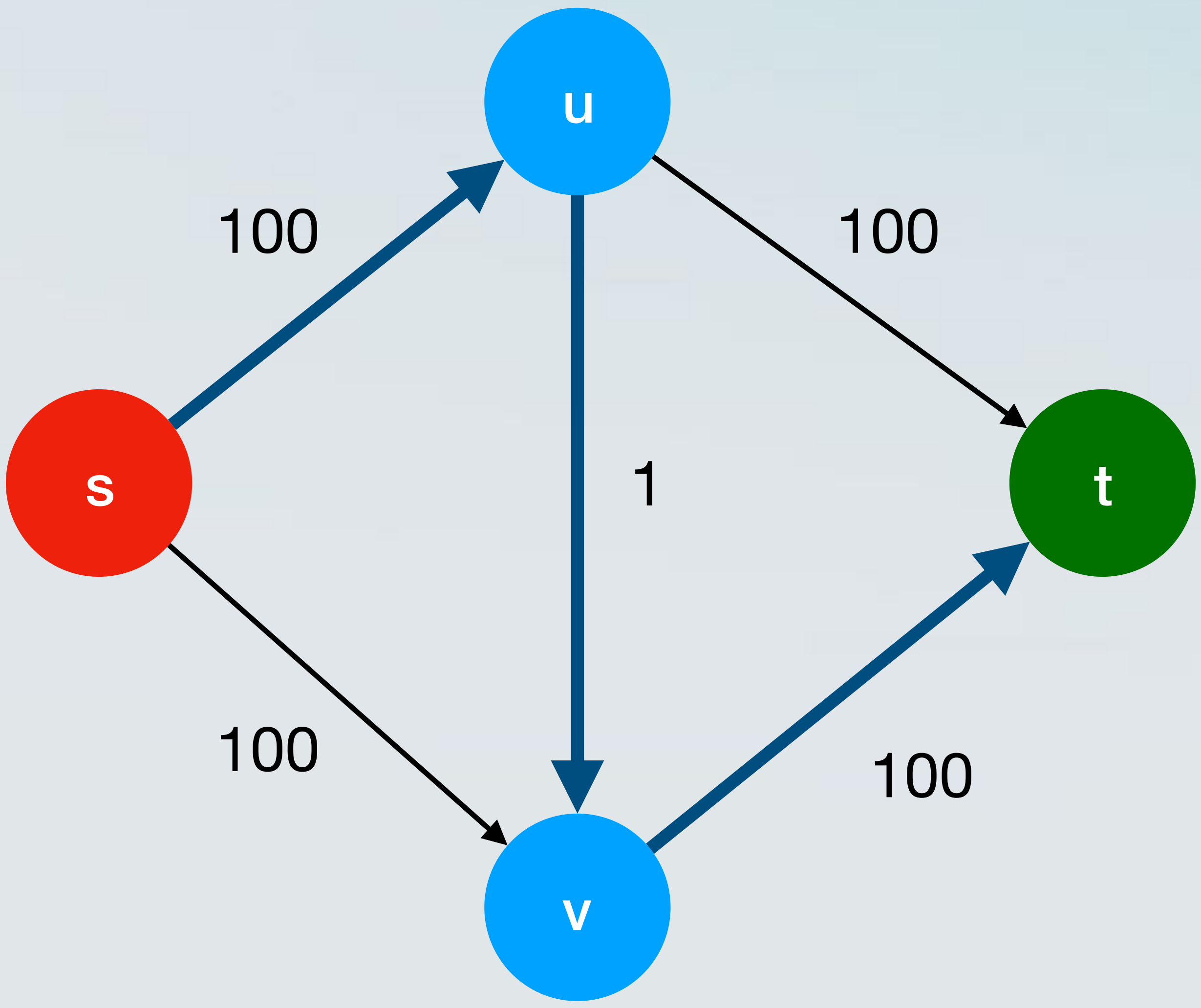
Endwhile

Return (f)

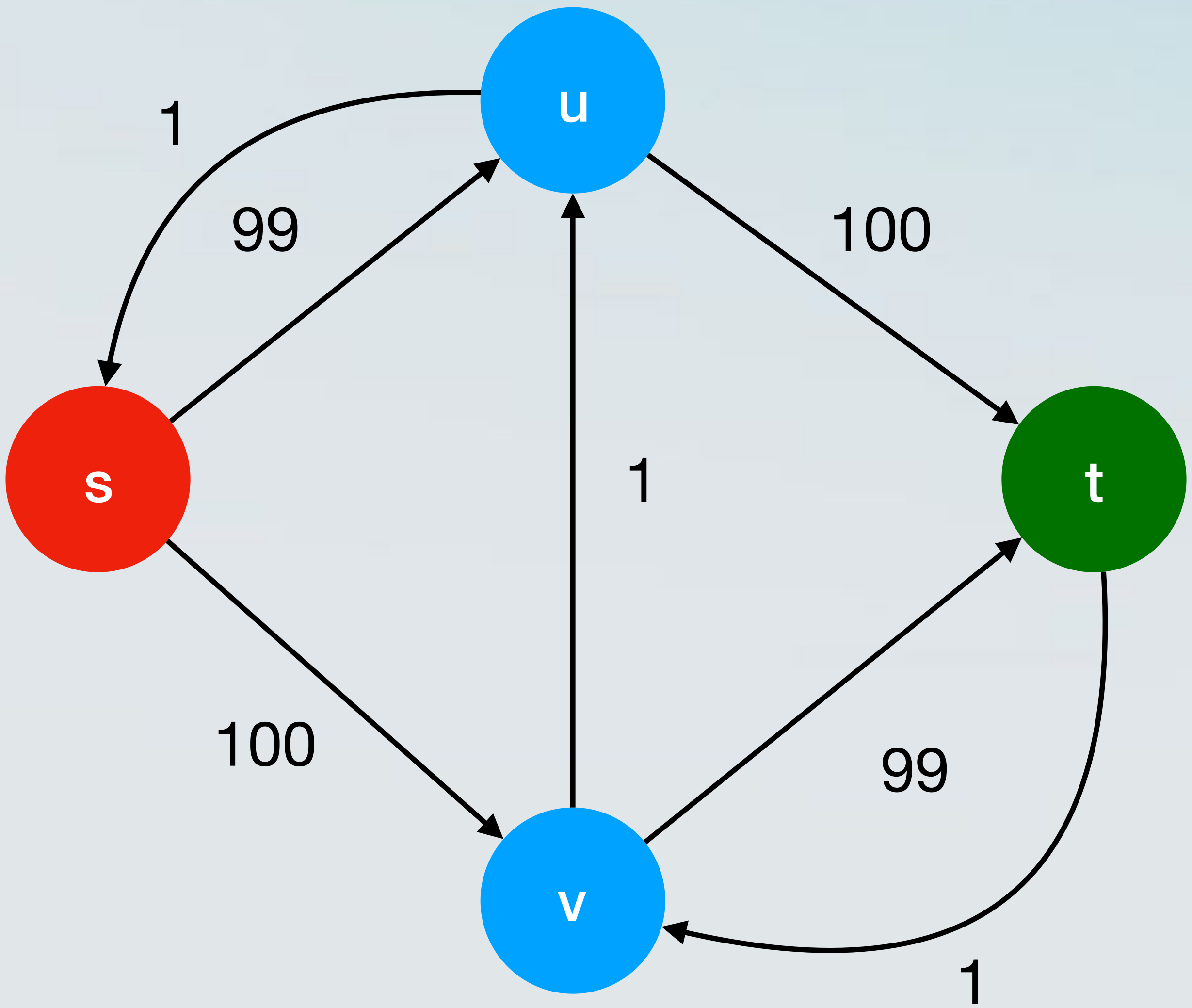
Example



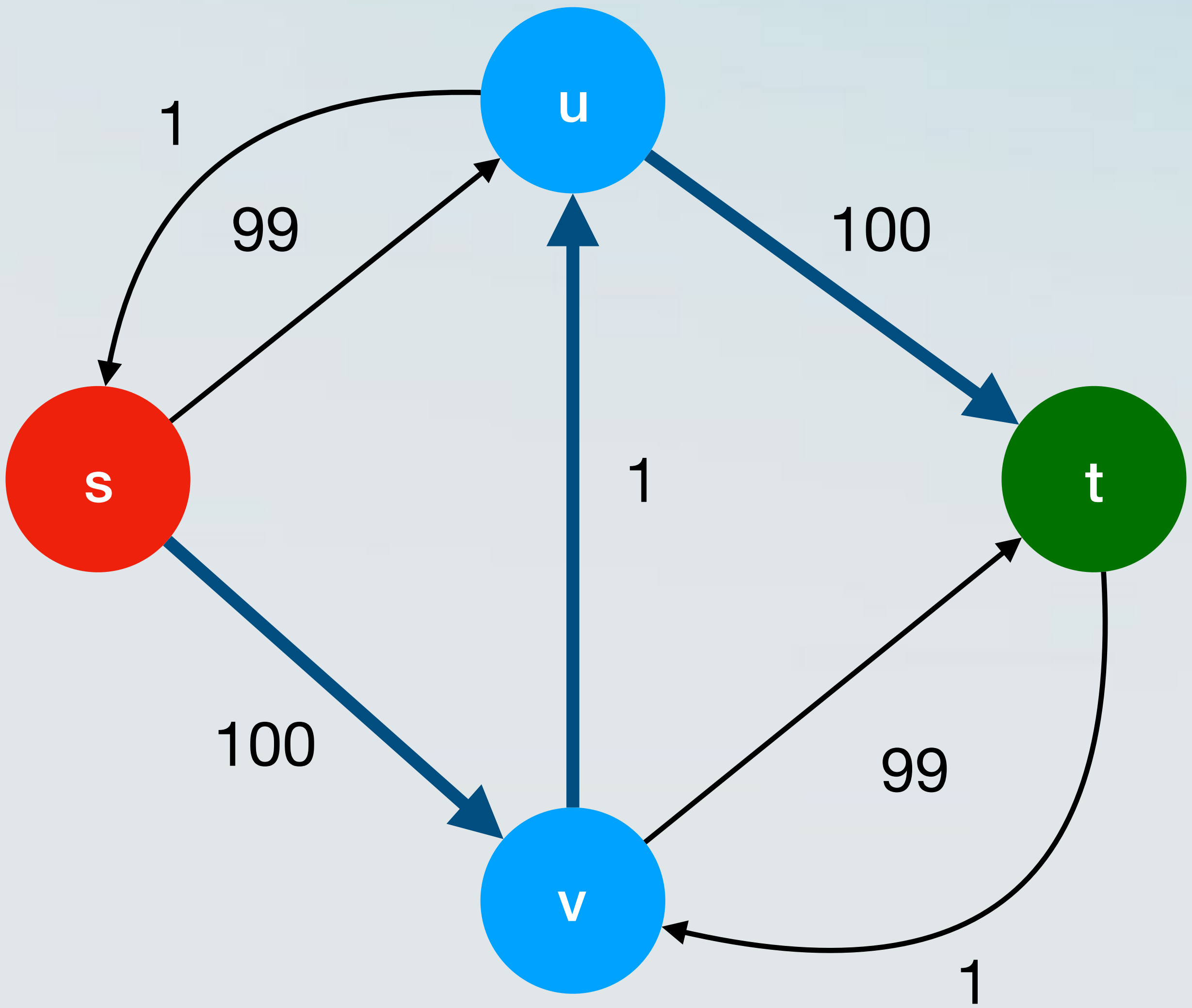
Example



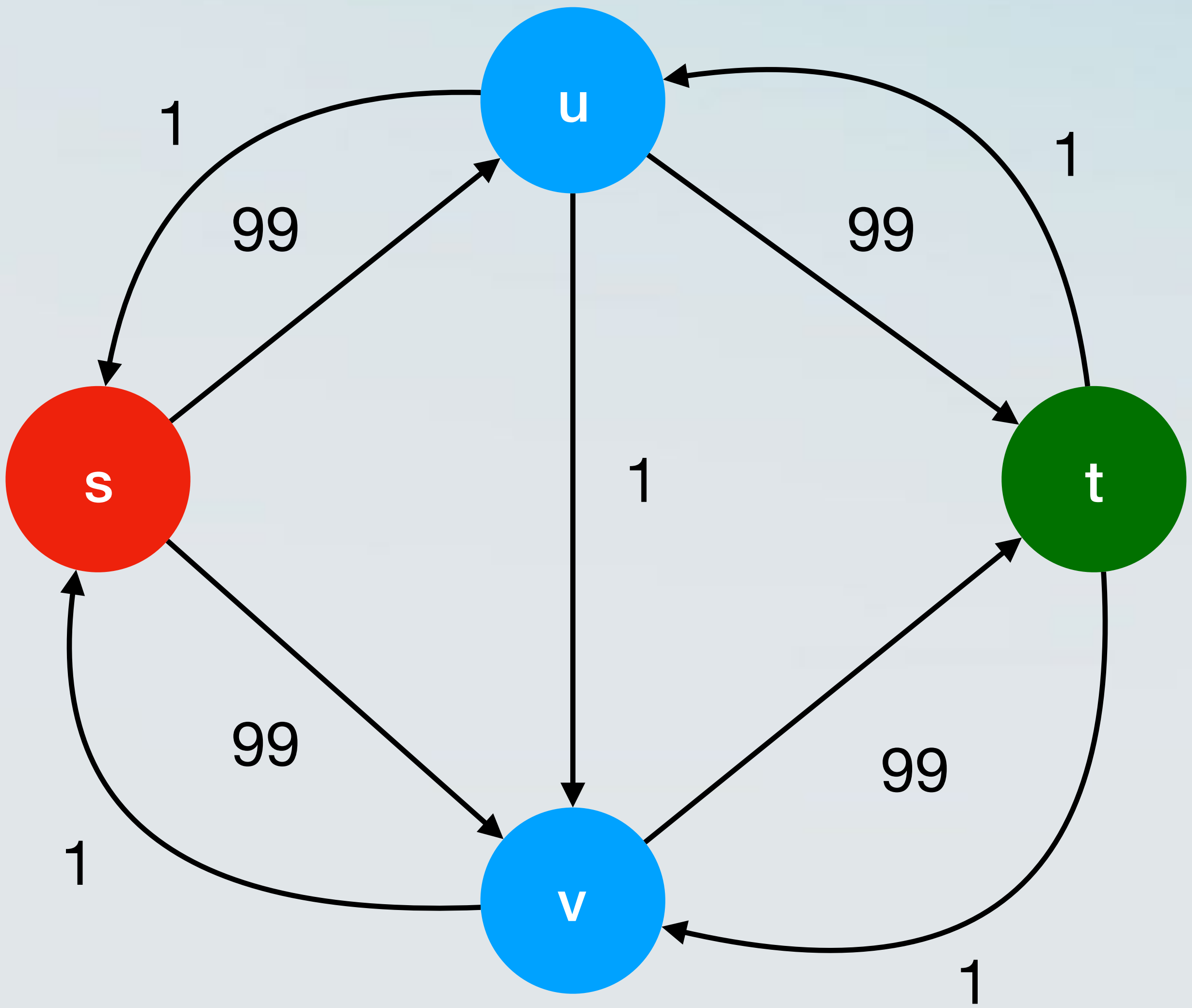
Example



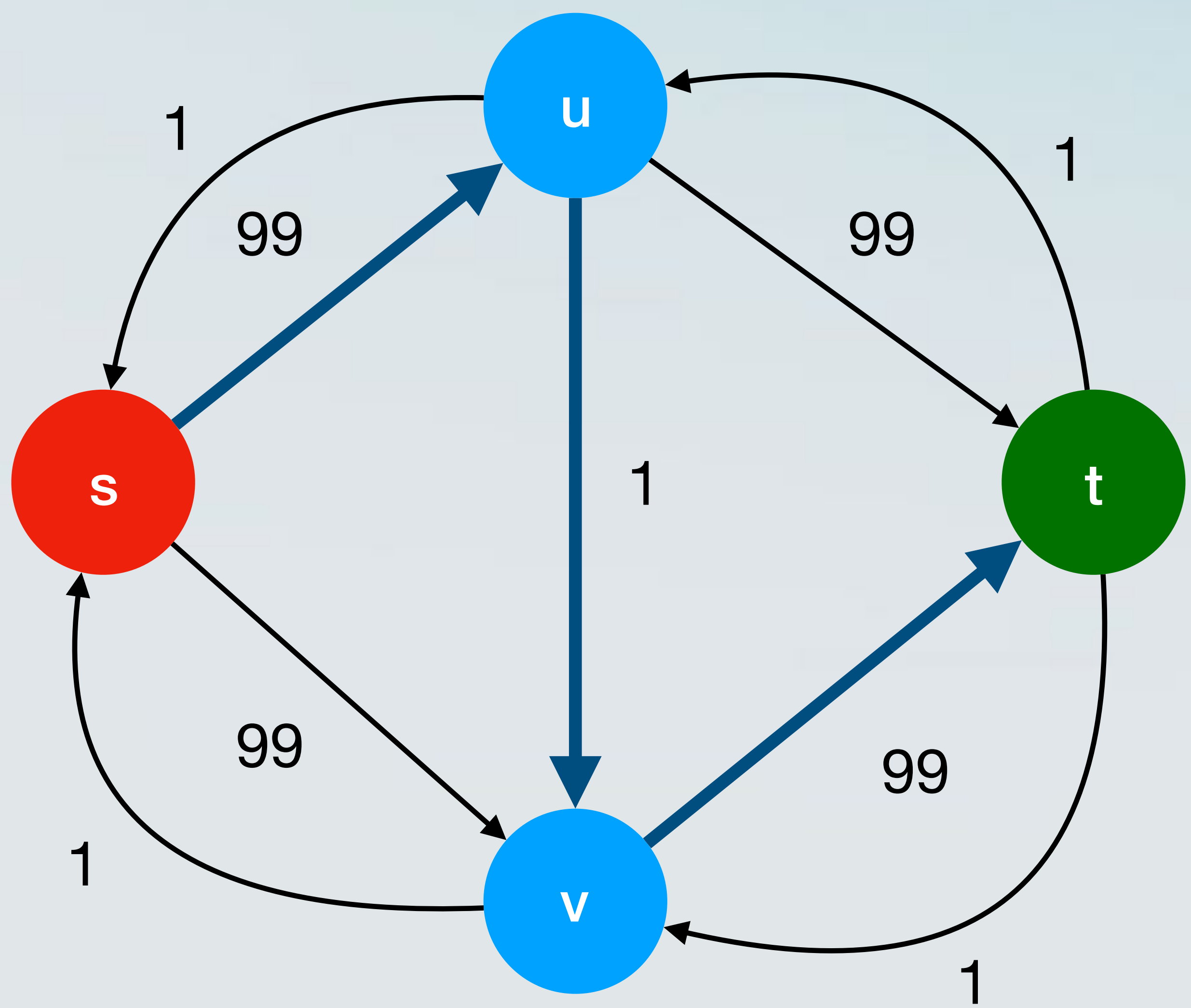
Example



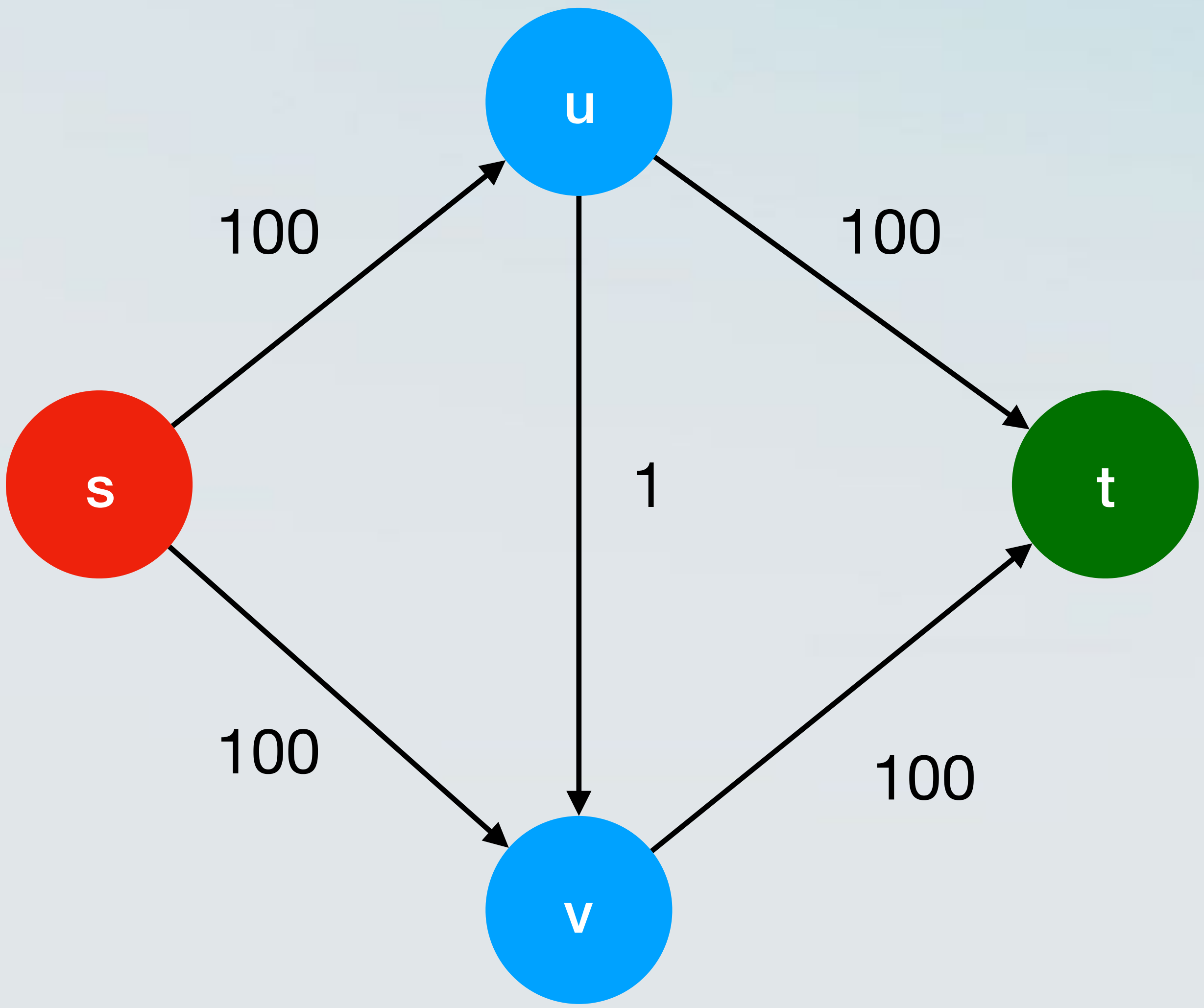
Example



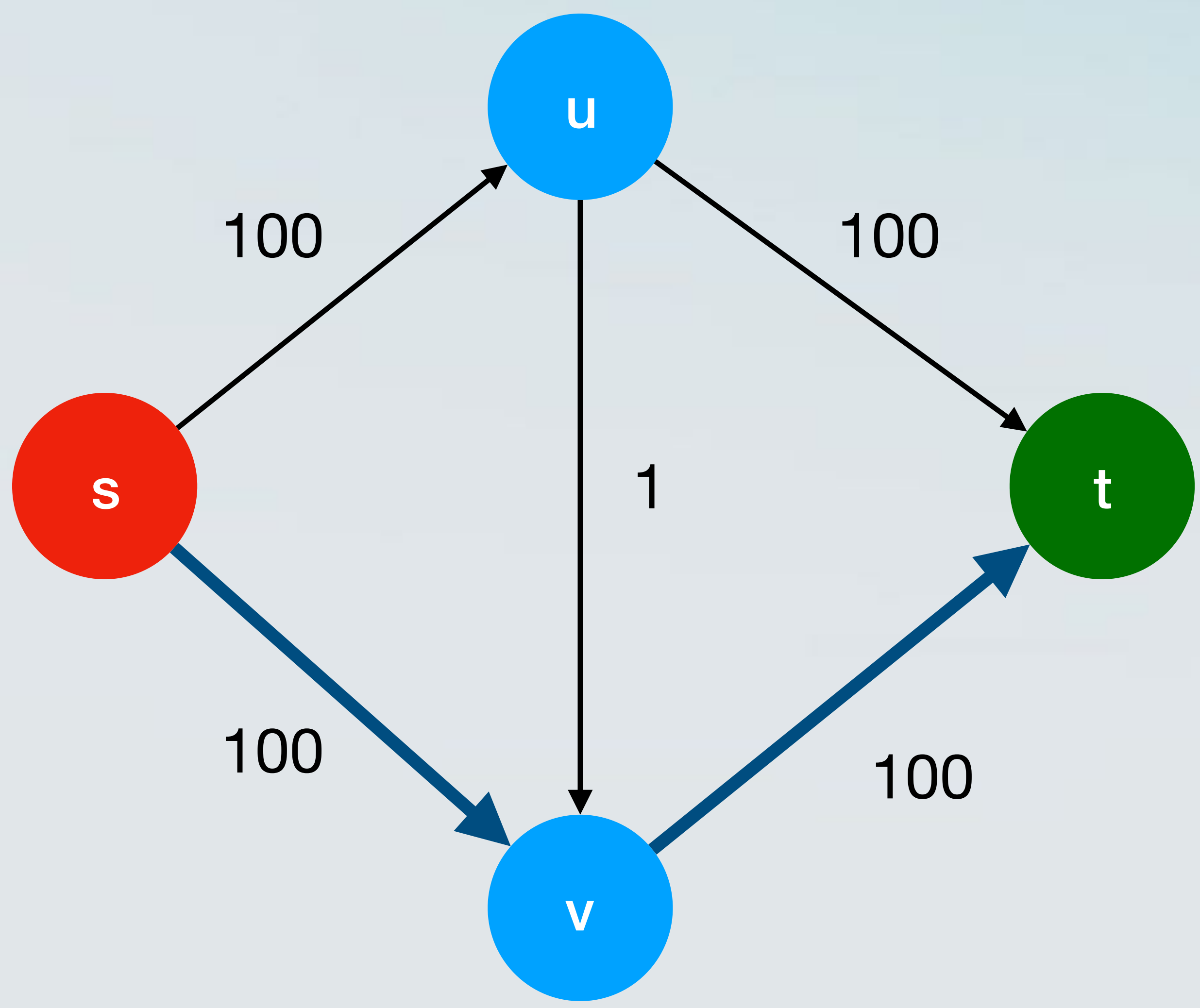
Example



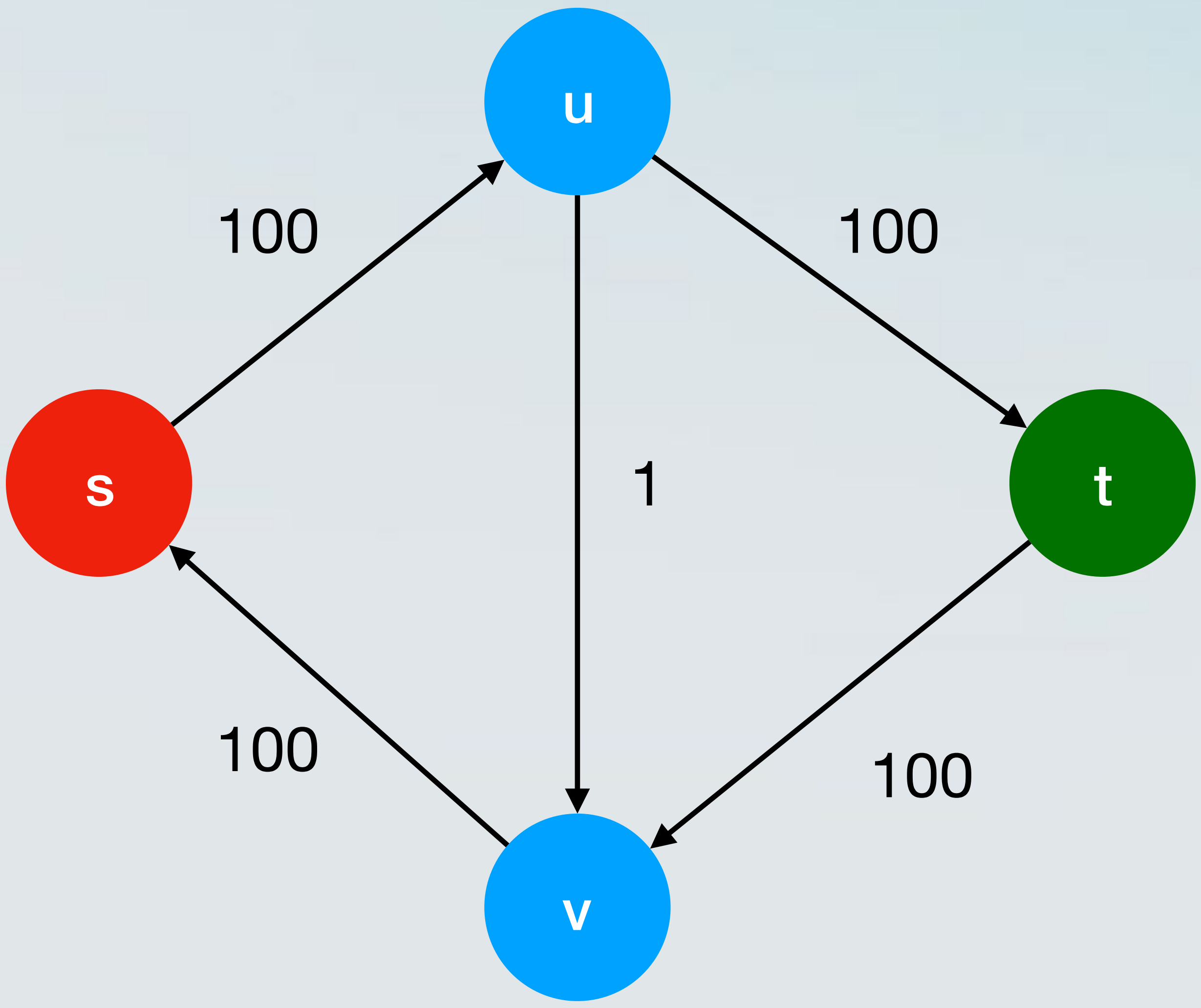
Example



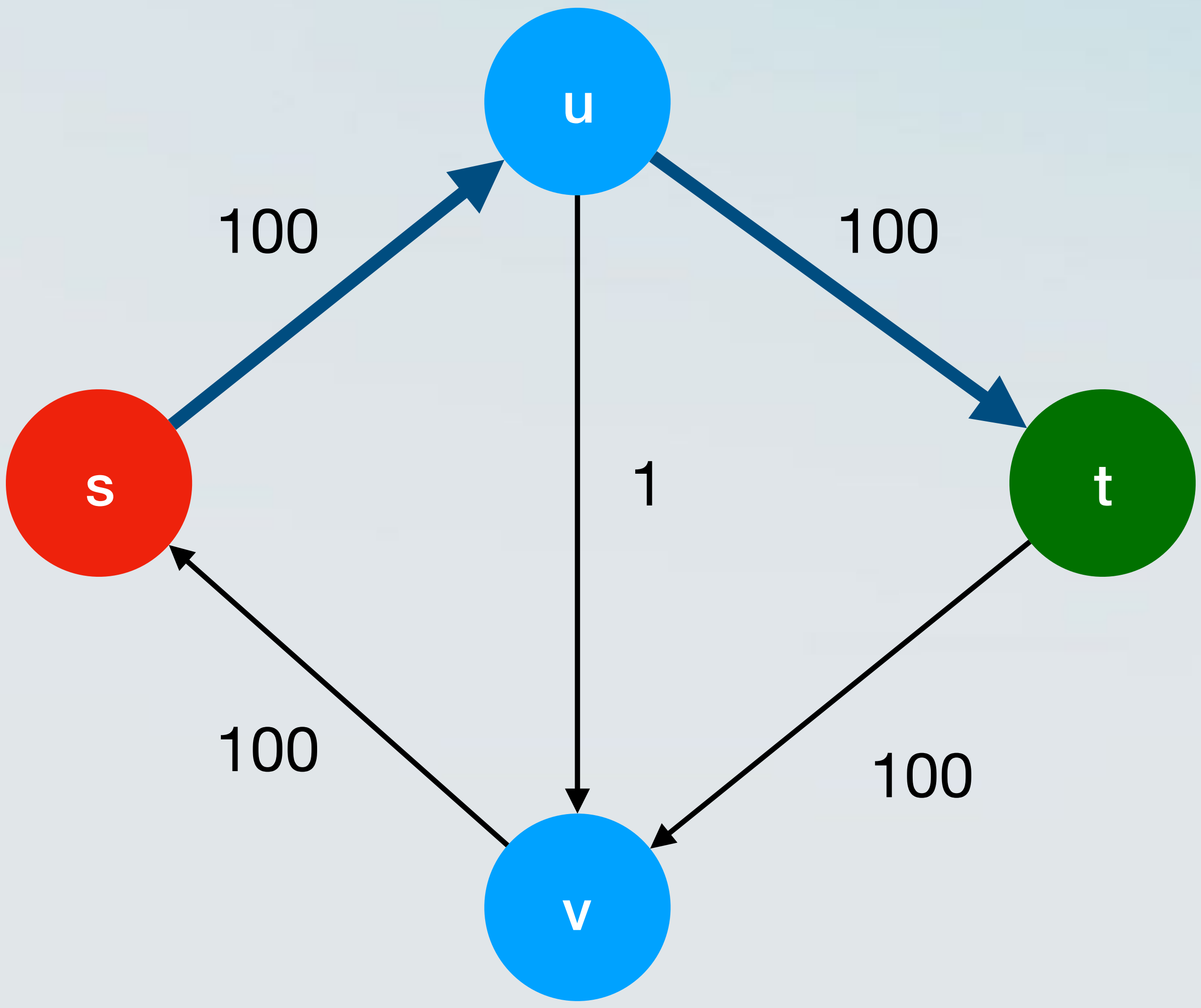
Example



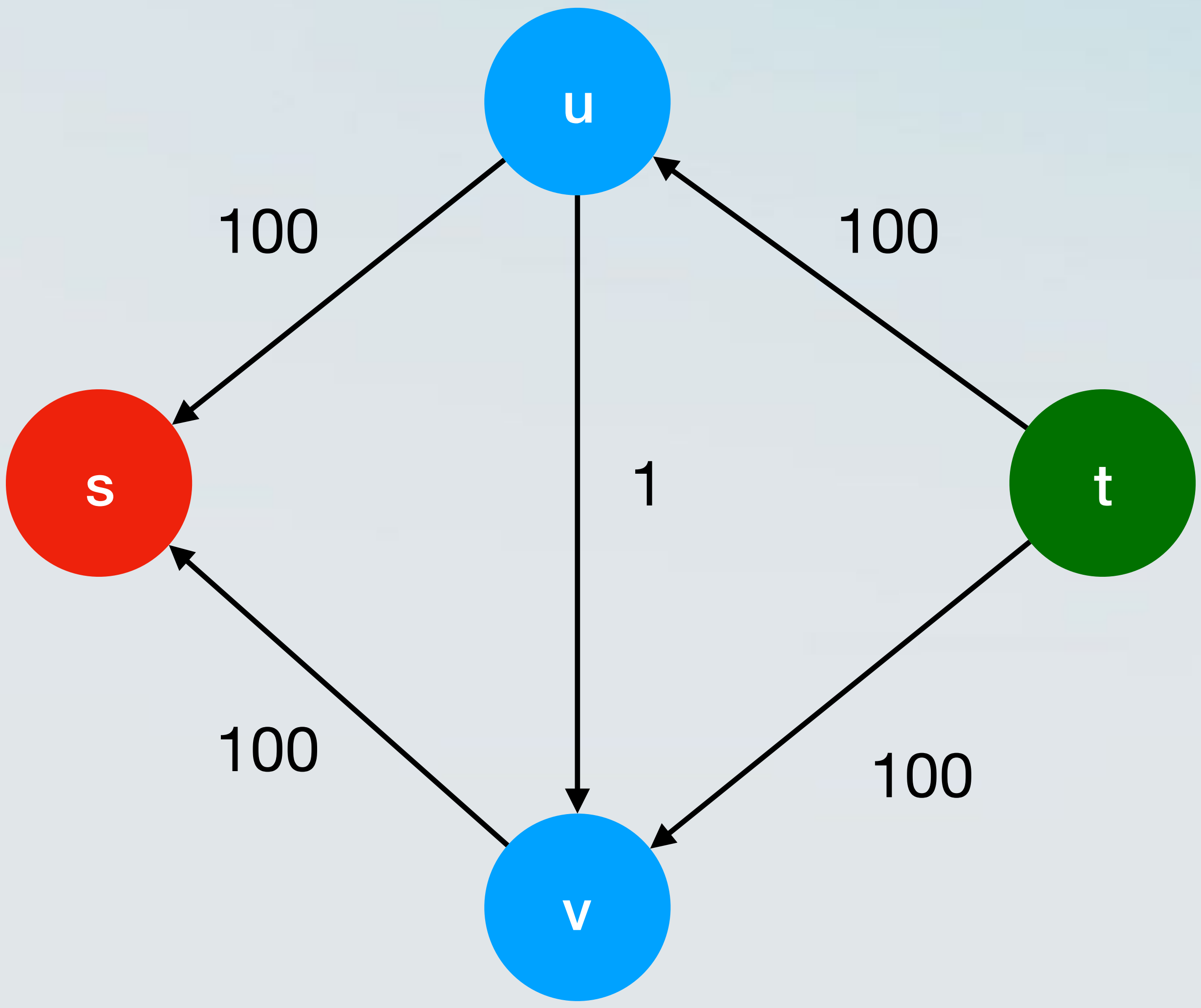
Example



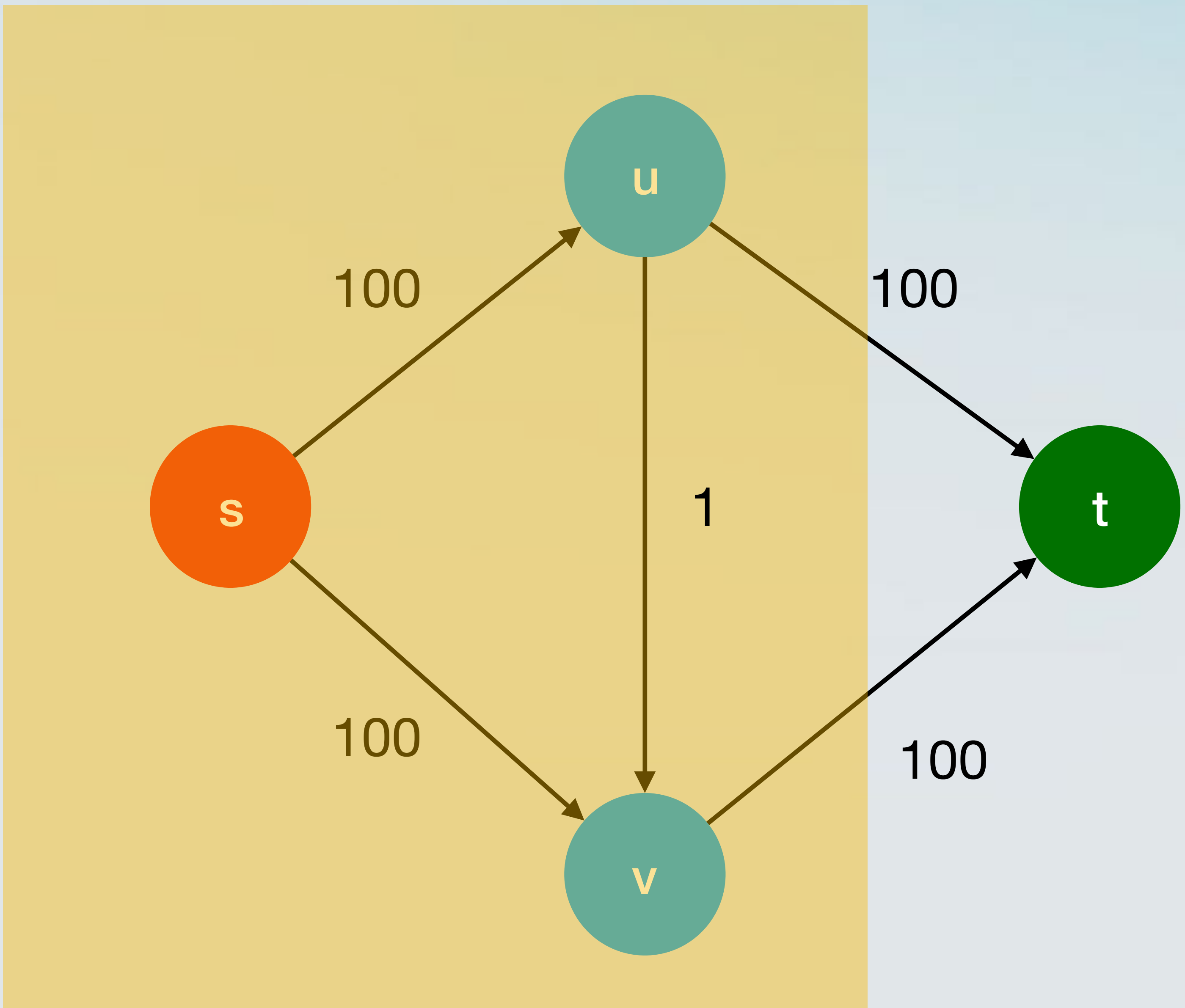
Example



Example



Example



Max-Flow in polynomial time

- We made the algorithm must faster by simply selecting the shortest path with available capacity.
- Can we always hope to do that?

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)

The Edmonds-Karp 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 the shortest such path P

f’ = **augment**(f, P)

Update f to be f’

Update the residual graph to be Gf’

Endwhile

Return (f)

The Edmonds-Karp Algorithm

- The Edmonds-Karp version of the Ford-Fulkerson algorithm runs in time **O(**nm^2**).**
- The shortest path can be found using a BFS search.