Network Flows

Admin/Announcements

- CS Preregistration Info Session during Colloquium. Learn about:
  - courses offered next semester
  - major applications & forms
  - thesis applications & timelines
  - study abroad guidelines
- TAs and becoming a TA
  - Fill out TA feedback forms by Monday
  - Submit TA applications by April 21
- Williams Entrepreneurship Summit this Saturday!

Story So Far

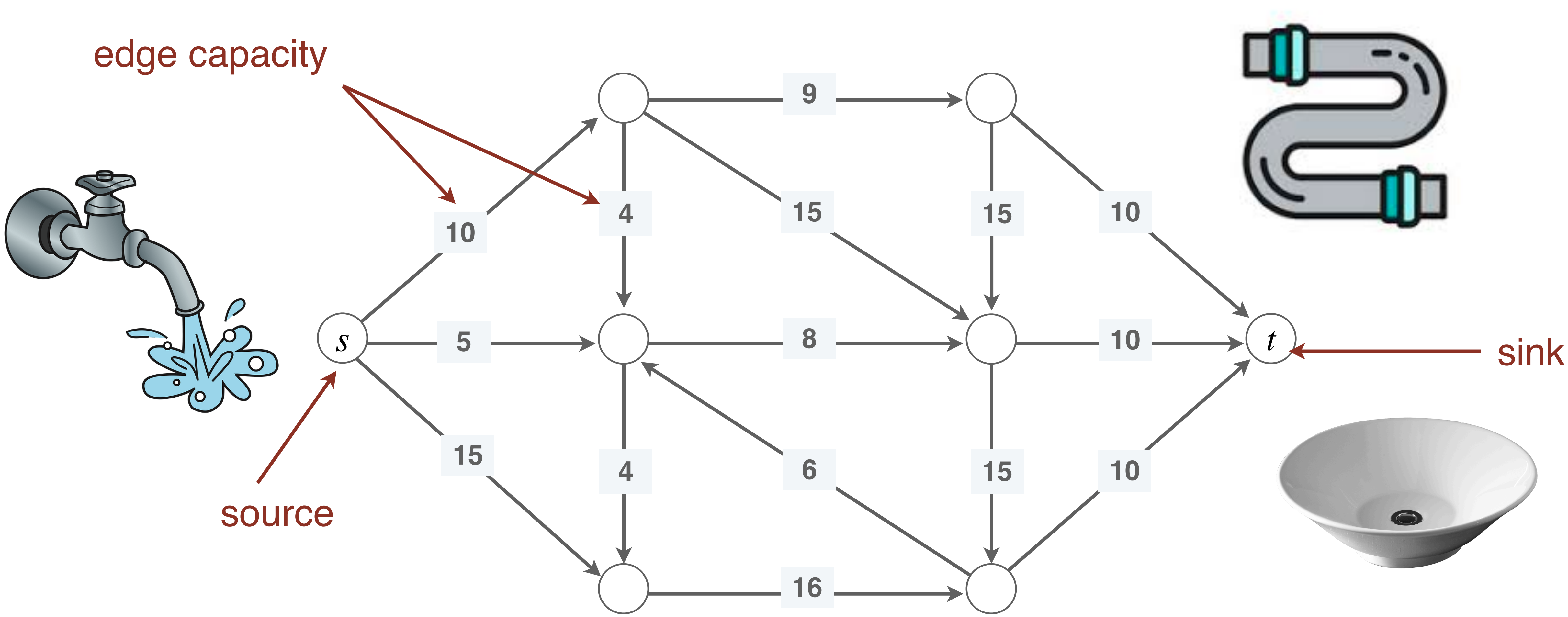
- Algorithmic design paradigms:
  - **Greedy**: often simplest algorithms to design, but only work for certain limited class of optimization problems
    - A good initial thought for most problems but rarely optimal
  - **Divide and Conquer**
    - Solving a problem by breaking it down into smaller subproblems and (often) combining results
  - **Dynamic programming**
    - Recursion with memoization: avoiding repeated work
    - Trade space (memoization structure representation) for time (reuse stored results of repeated computations)

New Algorithmic Paradigm

- **Network flows** model a variety of optimization problems
- These optimization problems look complicated with lots of constraints
  - At first they may seem to have nothing to do with networks or flows!
- Very powerful problem solving frameworks
- We'll focus on the concept of **problem reductions**
  - Problem **A** reduces to **B** if a solution to **B** leads to a solution to **A**
- We’ll learn how to prove that our reductions are correct

What's a Flow Network?

- A flow network is a directed graph $G = (V, E)$ with a
  - A **source** is a vertex s with in-degree 0
  - A **sink** is a vertex t with out-degree 0
  - Each edge $e \in E$ has **edge capacity** $c(e) > 0$



Assumptions

- Assume that each node v is on *some* s-t path, that is, $s \leadsto v \leadsto t$ exists, for any vertex $v \in V$
    - Implies G is connected and $m \geq n - 1$
- Assume **capacities are positive integers**
    - Will revisit this assumption & what happens otherwise
- Directed edge (u, v) written as $u \to v$
- For simplifying expositions, we will sometimes write $c(u \to v) = 0$ when $(u, v) \notin E$

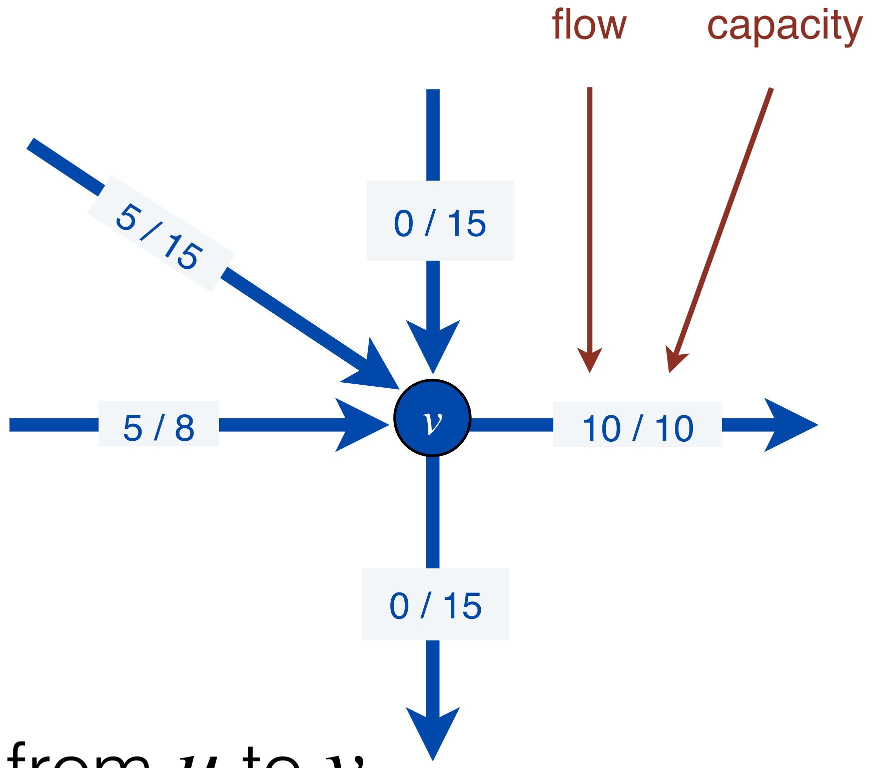
What's a Flow?

- Given a flow network, an (s,t)**-flow** or just **flow** (if source s and sink t are clear from context) $f : E \to \mathbb{Z}^+$ satisfies the following two constraints:

- **[Flow conservation]** $f_{in}(v) = f_{out}(v)$, for $v \neq s, t$ where

$$f_{in}(v) = \sum_u f(u \to v)$$

$$f_{out}(v) = \sum_w f(v \to w)$$

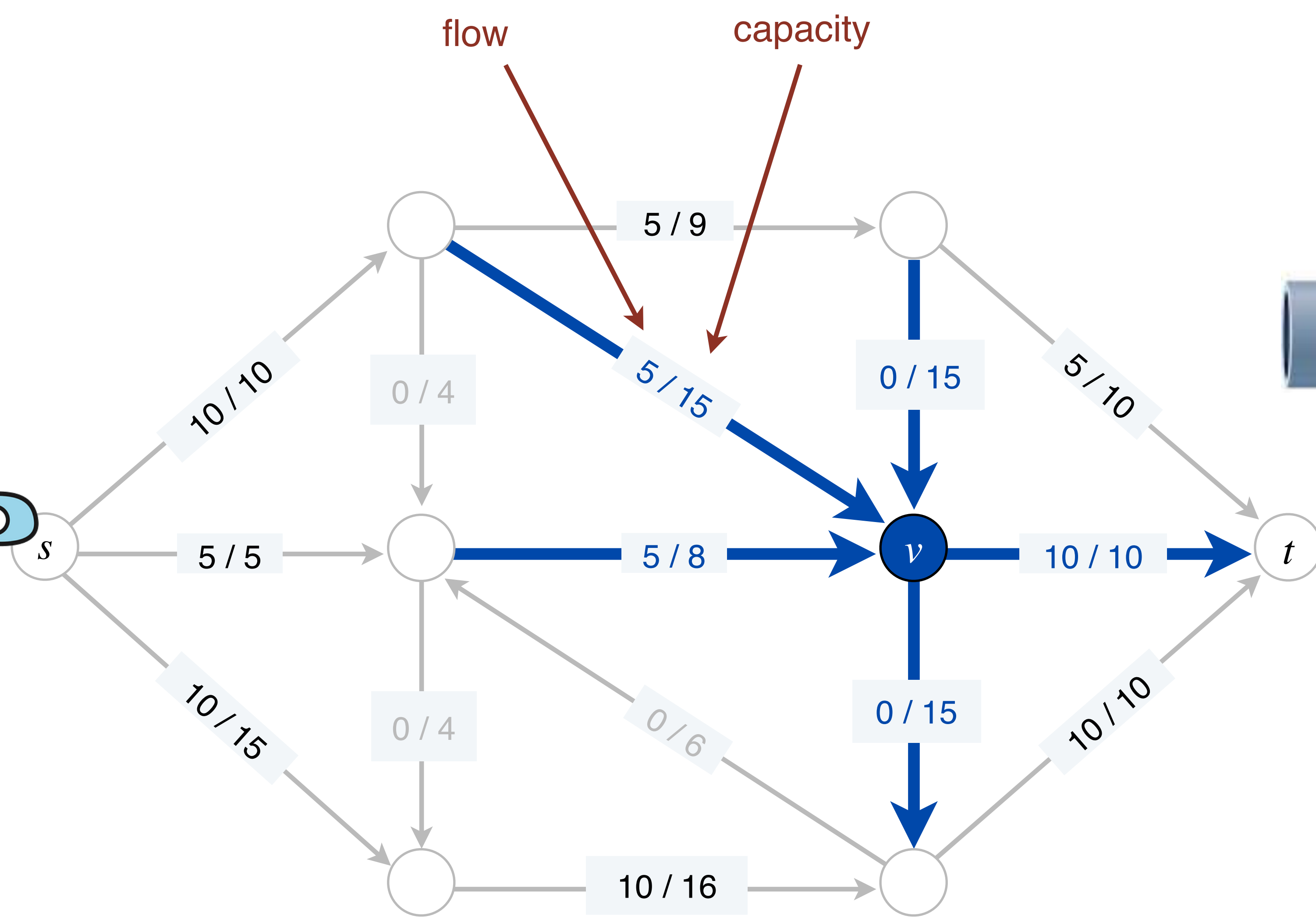


- To simplify, $f(u \to v) = 0$ if there is no edge from u to v

Feasible Flow

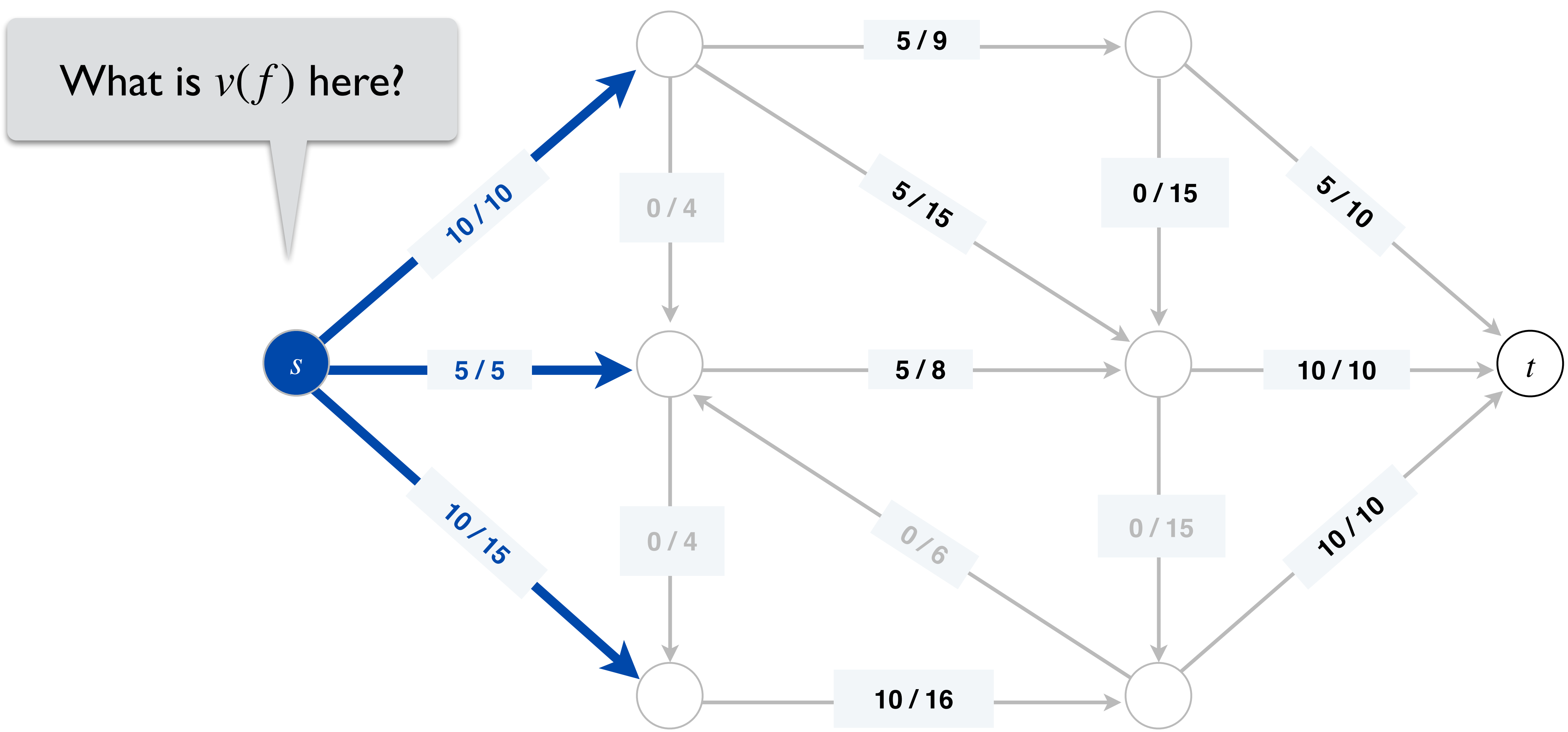
- And second, a feasible flow must satisfy the capacity constraints of the network, that is,

[Capacity constraint] for each $e \in E$, $0 \leq f(e) \leq c(e)$



Value of a Flow

- **Definition.** The **value** of a flow f, written $v(f)$, is $f_{out}(s)$.

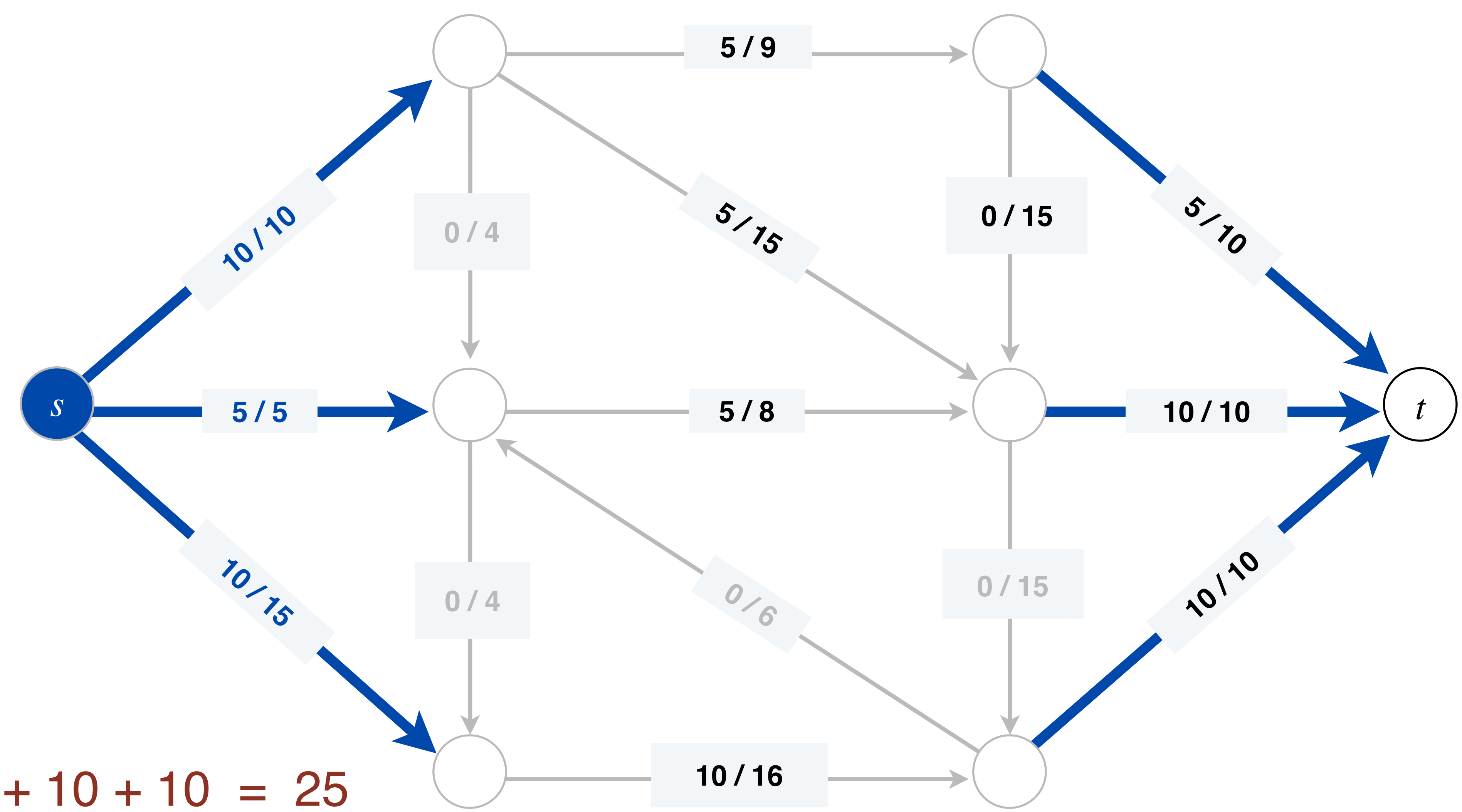


$v(f) = 5 + 10 + 10 = 25$

Value of a Flow

- **Definition.** The **value** of a flow f, written $v(f)$, is $f_{out}(s)$.
- **Lemma**. $f_{out}(s) = f_{in}(t)$

Intuitively, why do you think this is true?

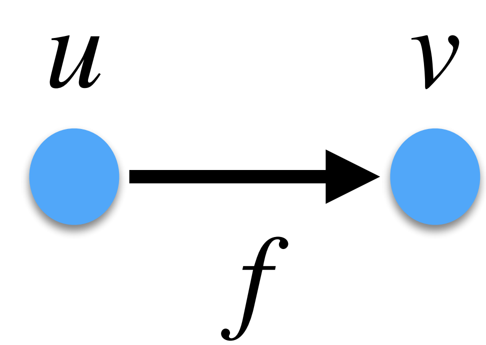


value = 5 + 10 + 10 = 25

Value of a Flow

Lemma. $f_{out}(s) = f_{in}(t)$

Proof. Let $f(E) = \sum_{e \in E} f(e)$

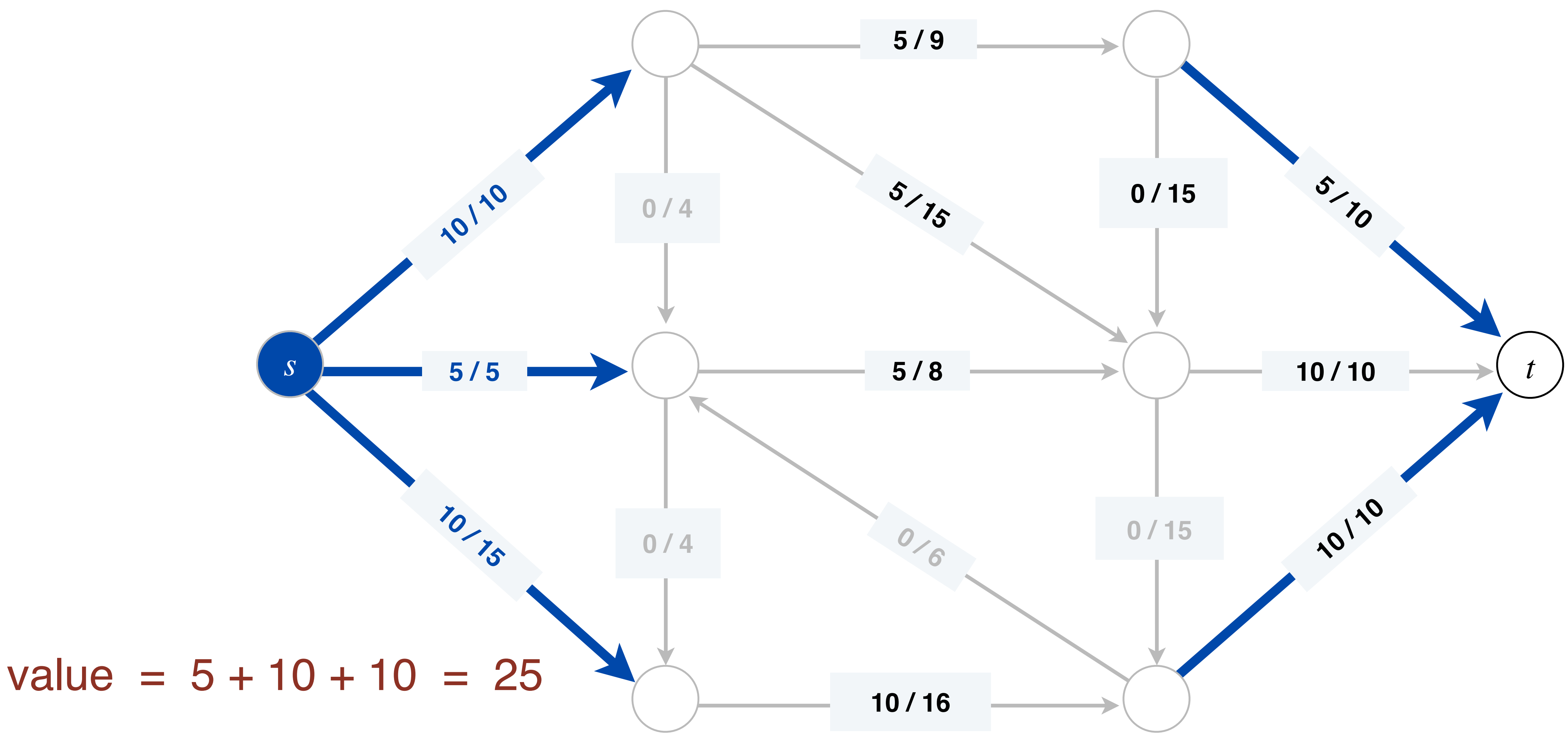


- Then, $\sum_{v \in V} f_{in}(v) = f(E) = \sum_{v \in V} f_{out}(v)$
- For every $v \neq s, t$ flow conservation implies $f_{in}(v) = f_{out}(v)$
- Thus all terms cancel out on both sides except

  $f_{in}(s) + f_{in}(t) = f_{out}(s) + f_{out}(t)$
- But $f_{in}(s) = f_{out}(t) = 0$ $\blacksquare$

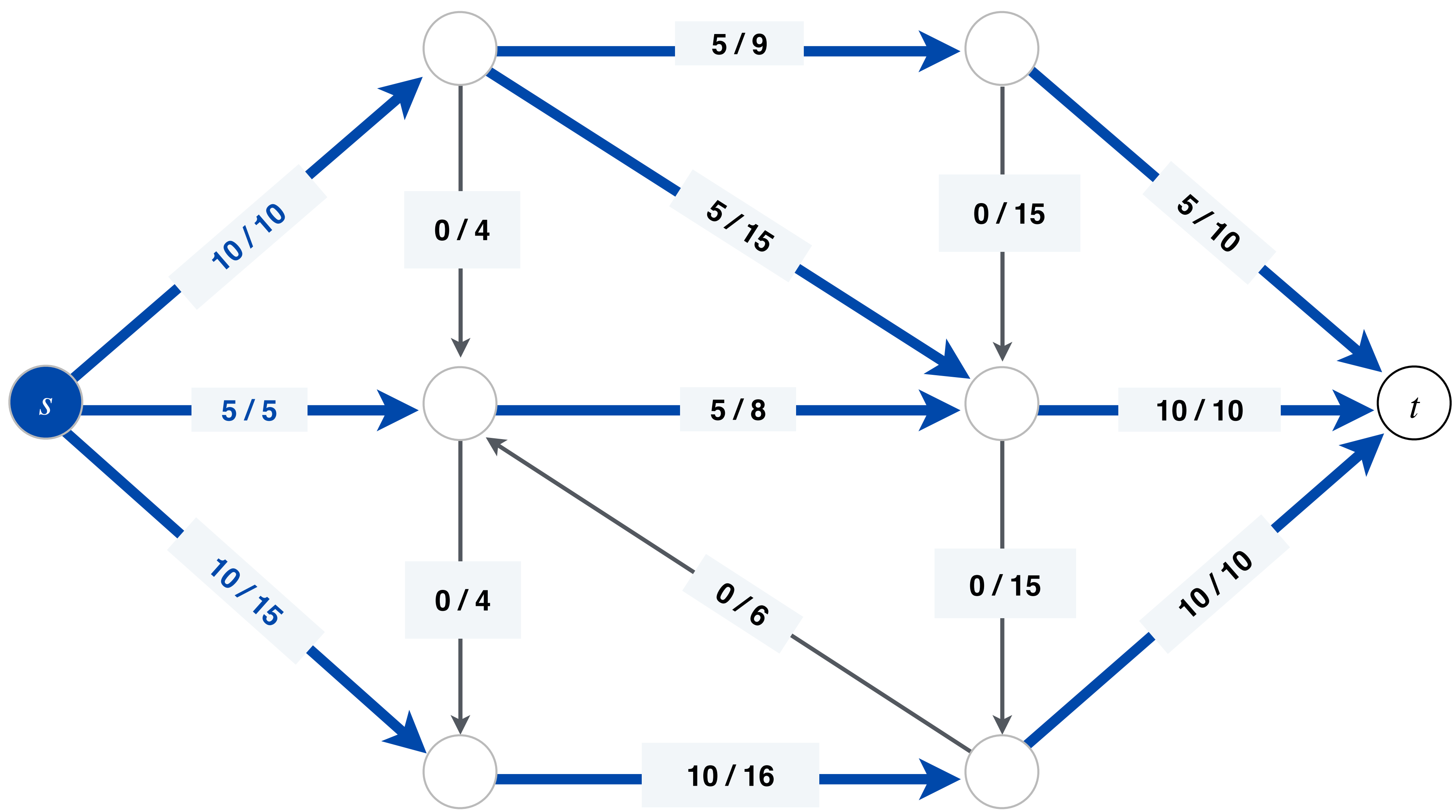
Value of a Flow

- **Lemma**. $f_{out}(s) = f_{in}(t)$
- **Corollary.** $v(f) = f_{in}(t)$.



Max-Flow Problem

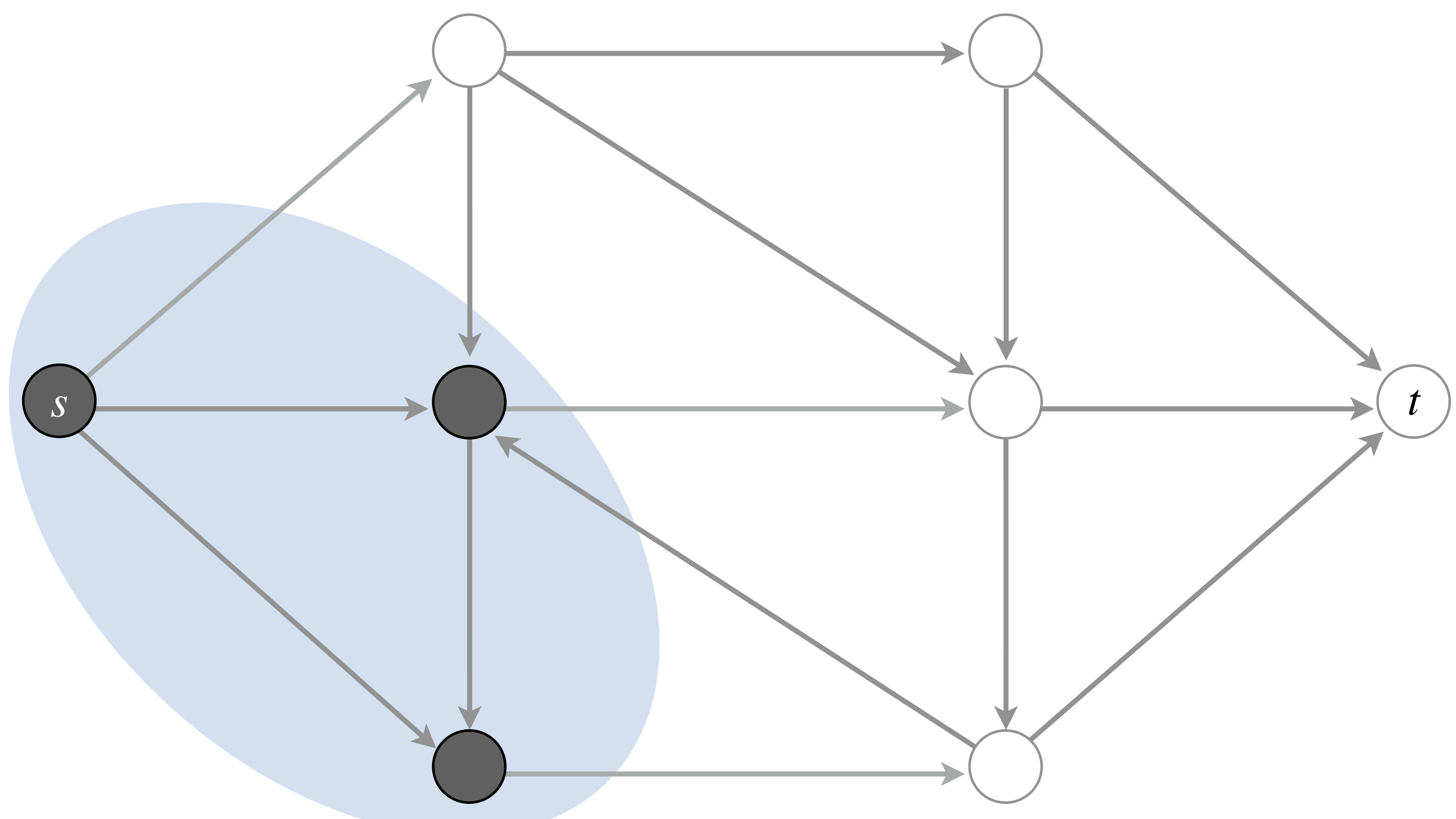
- **Problem**. Given an s-t flow network, find a **feasible** s-t flow of **maximum** value.



Minimum Cut Problem

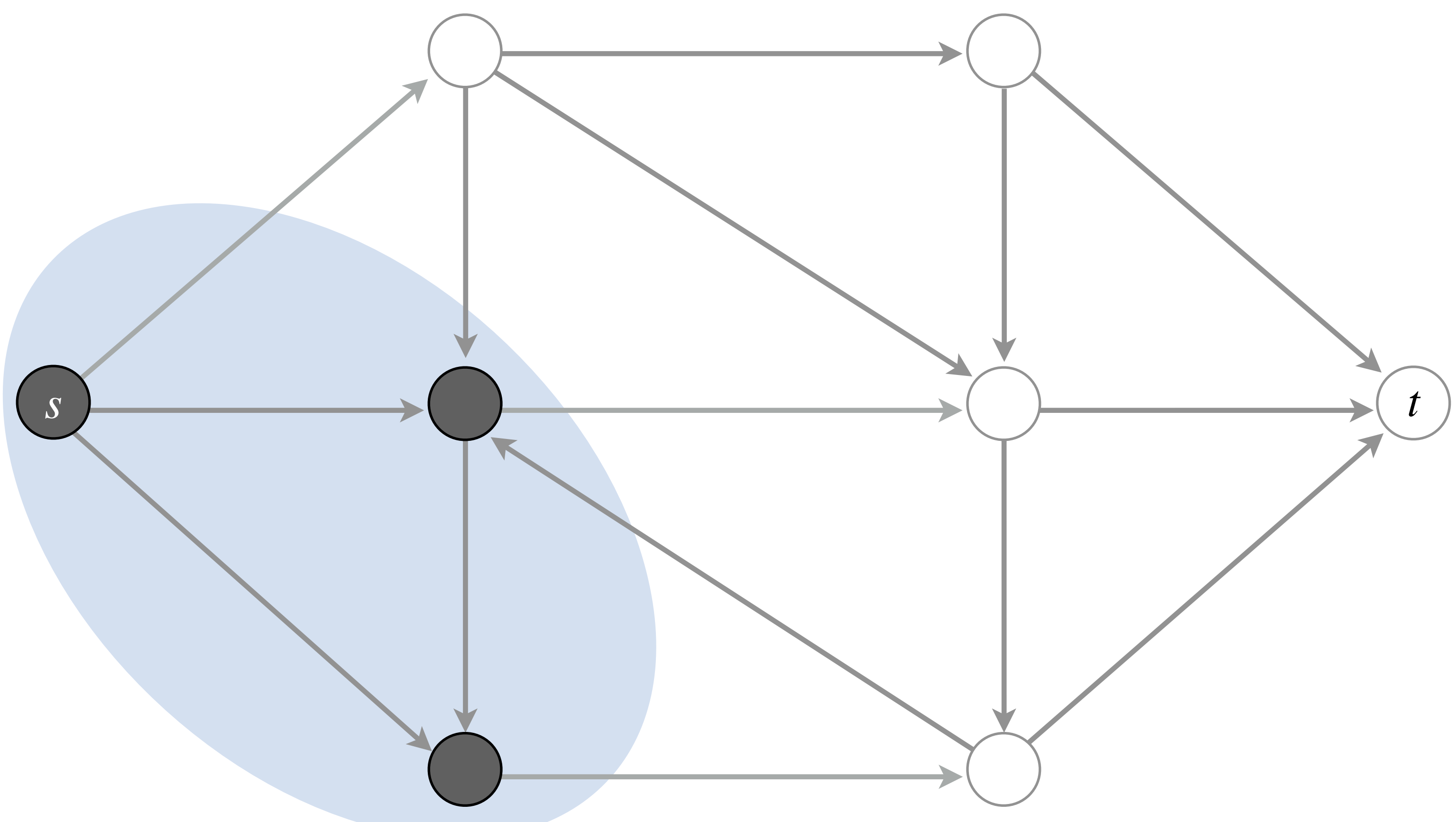
Cuts are Back!

- Cuts in graphs played a key role when we were designing algorithms for MSTs
- What is the definition of a cut?



Cuts in Flow Networks

- **Recall**. A cut (S, T) in a graph is a partition of vertices such that $S \cup T = V$, $S \cap T = \emptyset$ and S, T are non-empty.
- **Definition**. An (s, t)-*cut* is a cut (S, T) s.t. $s \in S$ and $t \in T$.



Cut Capacity

- **Recall**. A cut (S, T) in a graph is a partition of vertices such that $S \cup T = V$, $S \cap T = \emptyset$ and S, T are non-empty.
- **Definition**. An *(s, t)-cut* is a cut (S, T) s.t. $s \in S$ and $t \in T$.
- **Capacity** of a (s, t)-cut (S, T) is the sum of the capacities of edges leaving S:
  - $$c(S, T) = \sum_{v \in S, w \in T} c(v \to w)$$

Quick Quiz

Question. What is the capacity of the s-t cut given by grey and white nodes?

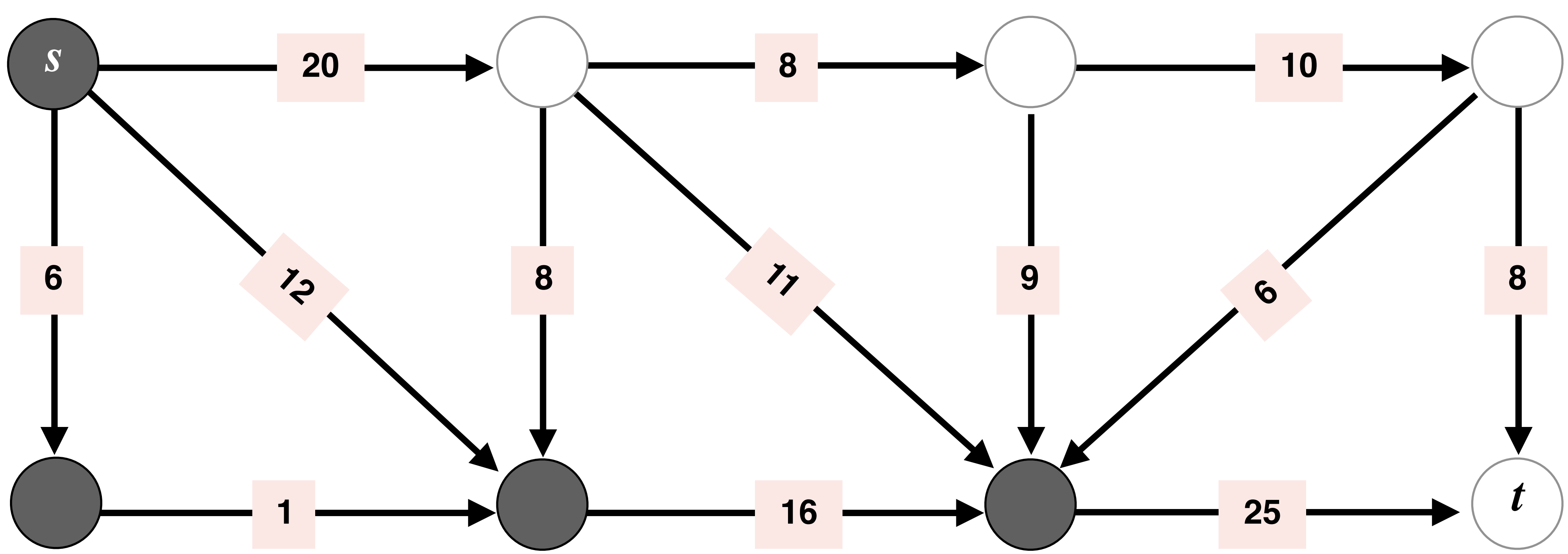
$$c(S, T) = \sum_{v \in S, w \in T} c(v \to w)$$

A. 11 (20 + 25 – 8 – 11 – 9 – 6)

B. 34 (8 + 11 + 9 + 6)

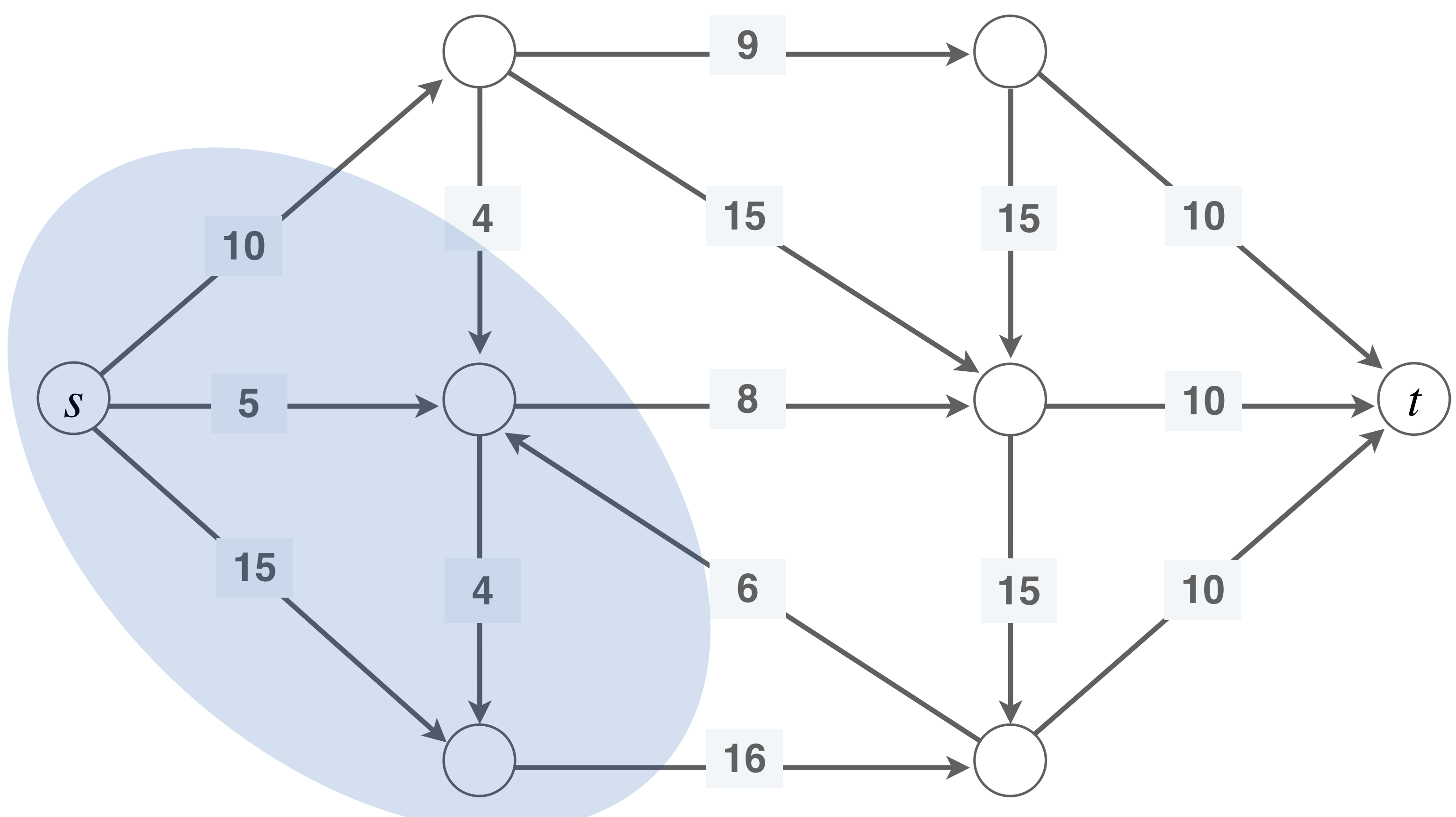
C. 45 (20 + 25)

D. 79 (20 + 25 + 8 + 11 + 9 + 6)



Min Cut Problem

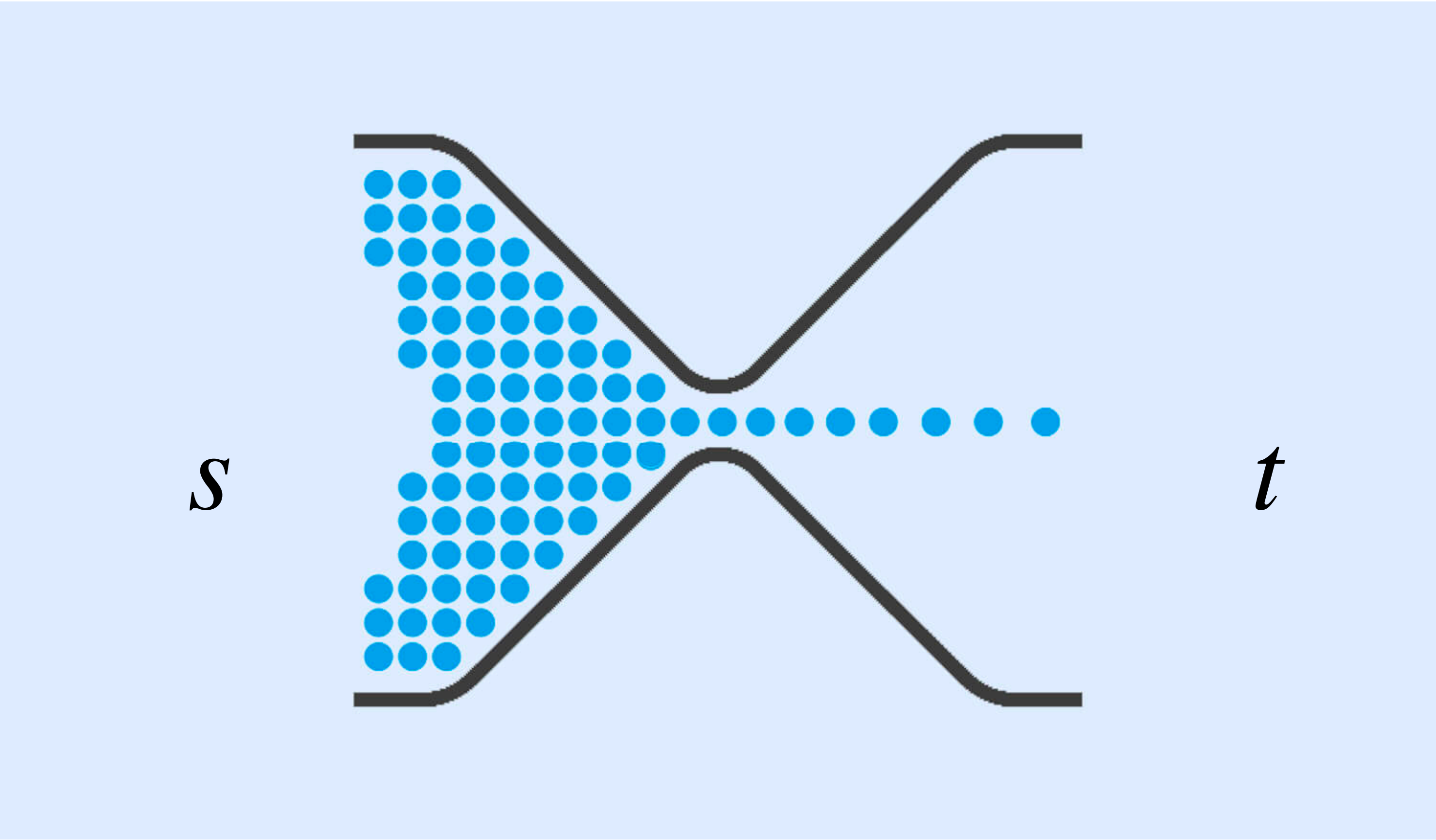
- **Problem**. Given an s-t flow network, find an s-t cut of **minimum** capacity.



Relationship between Flows and Cuts

Flows and Cuts

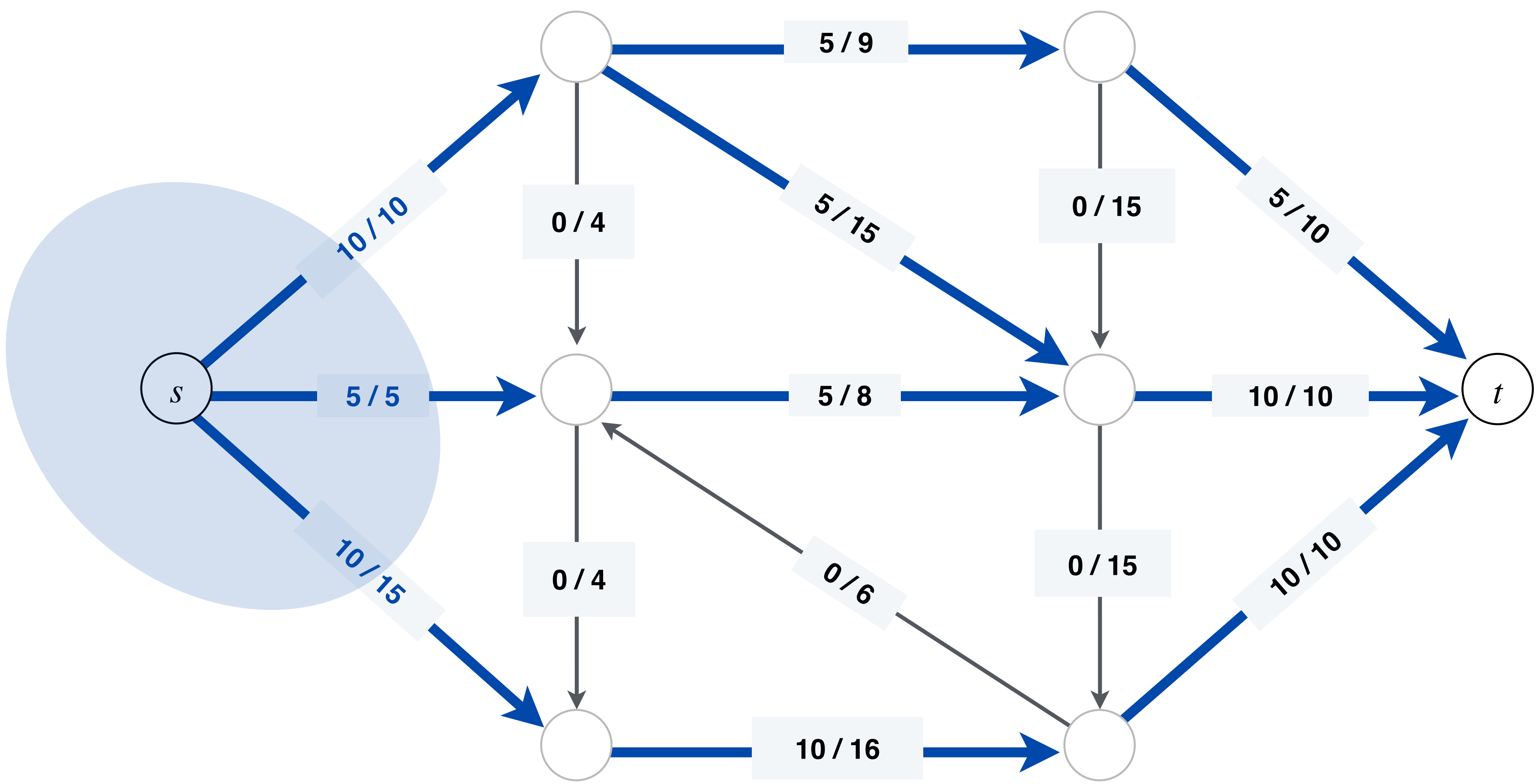
- Cuts represent "**bottlenecks**" in a flow network
- For any (s, t)-cut, all flow needs to “exit” S to get to t
- We will formalize this intuition



Flows and Cuts

Claim. Let f be **any** s-t flow and (S, T) be **any** s-t cut then
$v(f) \leq c(S, T)$

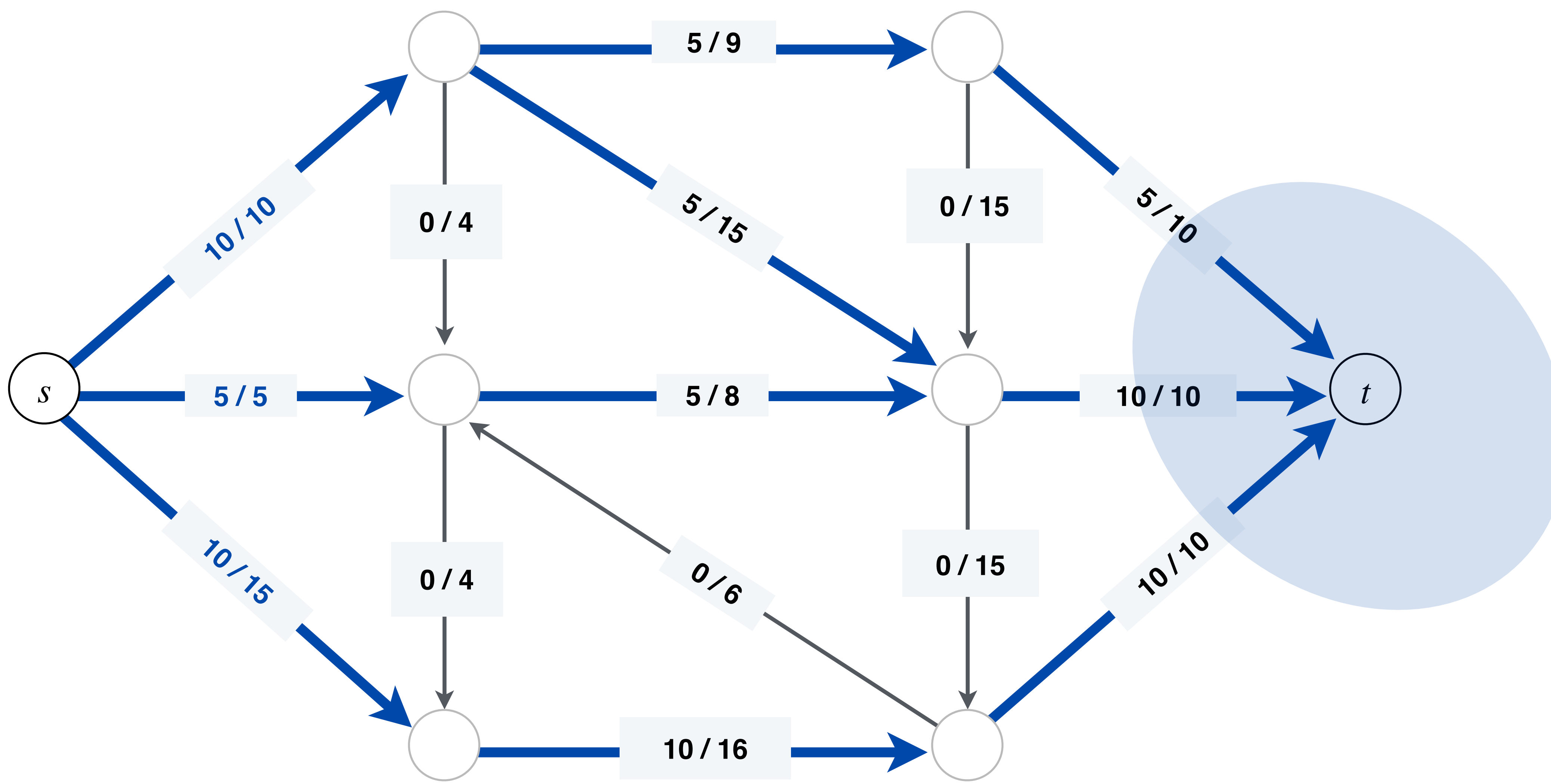
- There are two s-t cuts for which this is easy to see (which ones?)



Flows and Cuts

Claim. Let f be **any** s-t flow and (S, T) be **any** s-t cut then $v(f) \leq c(S, T)$

- There are two s-t cuts for which this is easy to see (which ones?)



Flows and Cuts

To prove this for any cut, we first relate the flow value in a network to the **net flow** leaving a cut

- **Lemma**. For *any* feasible (s, t)-flow f on $G = (V, E)$ and *any* (s, t)-cut , $v(f) = f_{out}(S) - f_{in}(S)$, where
  - $f_{out}(S) = \sum_{v \in S, w \in T} f(v \to w)$ (sum of flow 'leaving' S)
  - $f_{in}(S) = \sum_{v \in S, w \in T} f(w \to v)$ (sum of flow 'entering' S)
  - Note: $f_{out}(S) = f_{in}(T)$ and $f_{in}(S) = f_{out}(T)$

Flows and Cuts

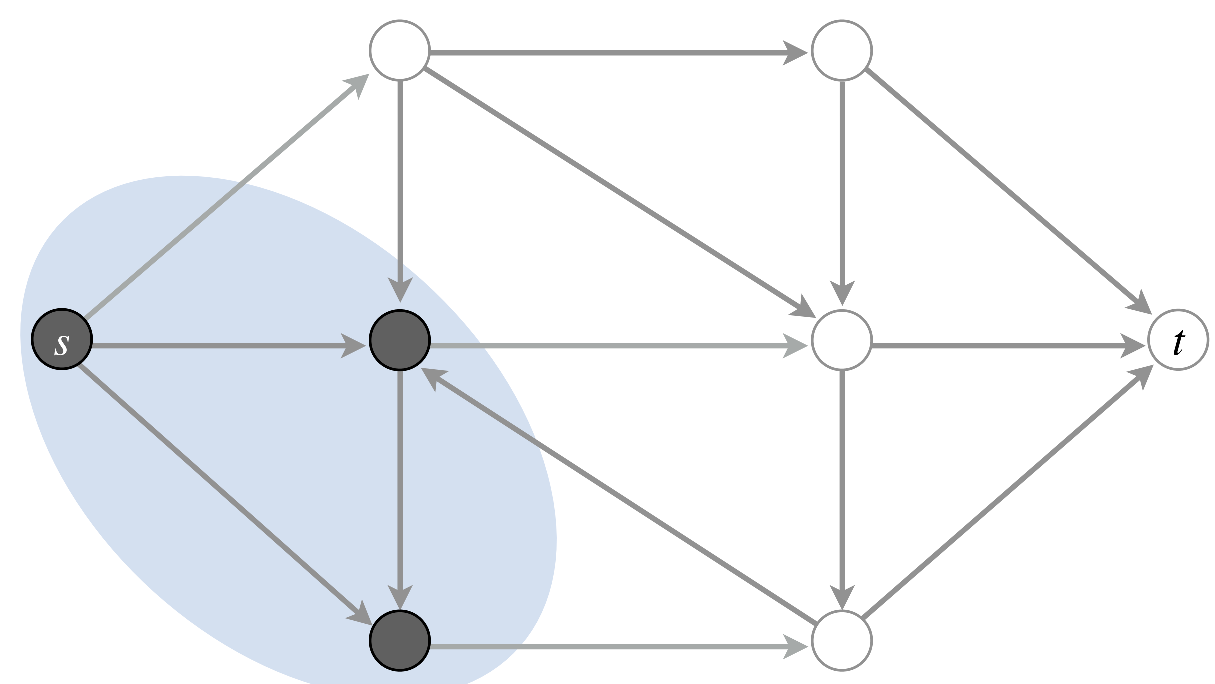
Proof. $f_{out}(S) - f_{in}(S)$

$$= \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v) \quad \text{[by definition]}$$

Adding zero terms

$$= \left[\sum_{v,w \in S} f(v \to w) - \sum_{v,u \in S} f(u \to v) \right] + \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)$$

These are the same sum: they sum the flow of all edges with both vertices in S



Flows and Cuts

Rearranging terms

Proof. $f_{out}(S) - f_{in}(S)$

$$= \left[\sum_{v,w\in S} f(v \to w) - \sum_{v,u\in S} f(u \to v) \right] + \sum_{v\in S, w\in T} f(v \to w) - \sum_{v\in S, u\in T} f(u \to v)$$

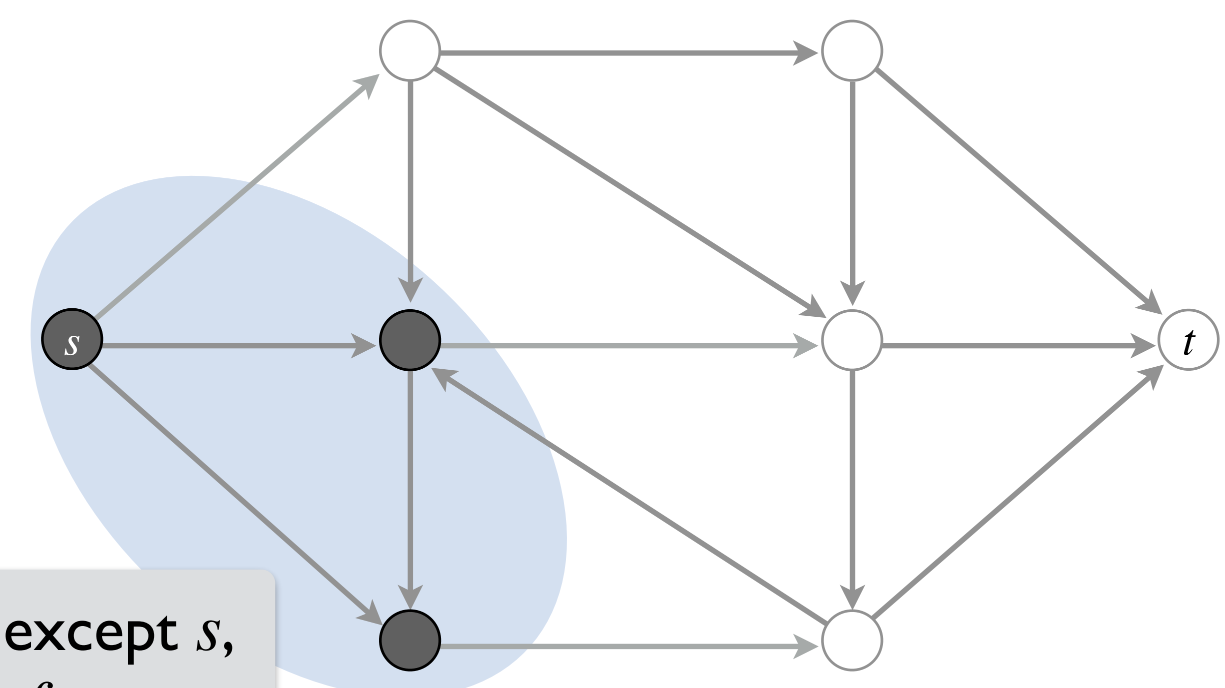
$$= \sum_{v,w\in S} f(v \to w) + \sum_{v\in S, w\in T} f(v \to w) - \sum_{v,u\in S} f(u \to v) - \sum_{v\in S, u\in T} f(u \to v)$$

$$= \sum_{v\in S} \left(\sum_{w} f(v \to w) - \sum_{u} f(u \to v) \right)$$

$$= \sum_{v\in S} f_{out}(v) - f_{in}(v)$$

$$= f_{out}(s) = v(f) \quad \blacksquare$$

Cancels out for all except s, which has no f_{in}



Flows and Cuts

- We use this result to prove that the value of a flow cannot exceed the capacity of **<u>any</u>** cut in the network.

- **Claim**. Let f be any s-t flow and (S, T) be any s-t cut then $v(f) \leq c(S, T)$

Sum of capacities leaving S

- **Proof.** $v(f) = f_{out}(S) - f_{in}(S)$

$$\leq f_{out}(S) = \sum_{v \in S, w \in T} f(v \to w)$$

$$\leq \sum_{v \in S, w \in T} c(v, w) = c(S, T)$$

When is $v(f) = c(S, T)$?

$f_{in}(S) = 0, \ f_{out}(S) = c(S, T)$

Max-Flow & Min-Cut

- Suppose the $c_{\min}$ is the capacity of the **minimum cut** in a network
- What can we say about the feasible flow we can send through it
  - cannot be more than $c_{\min}$
- In fact, whenever we find any s-t flow f and any s-t cut (S, T) such that, $v(f) = c(S, T)$ we can conclude that:
  - f is the maximum flow, and,
  - (S, T) is the minimum cut
- The question now is, given any flow network with min cut $c_{\min}$, is it always possible to route a feasible s-t flow f with $v(f) = c_{\min}$?

Max-Flow Min-Cut Theorem

There is a beautiful, powerful relationship between these two problems in given by the following theorem.

- **Theorem**. Given any flow network G, there exists a feasible (s, t)-flow f and an (s, t)-cut (S, T) such that,

$$v(f) = c(S, T)$$

- Informally, in a flow network, the **max-flow = min-cut**
- This will guide our algorithm design for finding max flow
- (Will prove this theorem by construction in a bit.)

Aside: Network Flow History

- In 1950s, US military researchers Harris and Ross wrote a classified report about the rail network linking Soviet Union and Eastern Europe
  - Vertices were the geographic regions
  - Edges were railway links between the regions
  - Edge weights were the rate at which material could be shipped from one region to next
- Ross and Harris determined:
  - Maximum amount of stuff that could be moved from Russia to Europe **(max flow)**
  - Cheapest way to disrupt the network by removing rail links **(min cut)**

Network Flow History

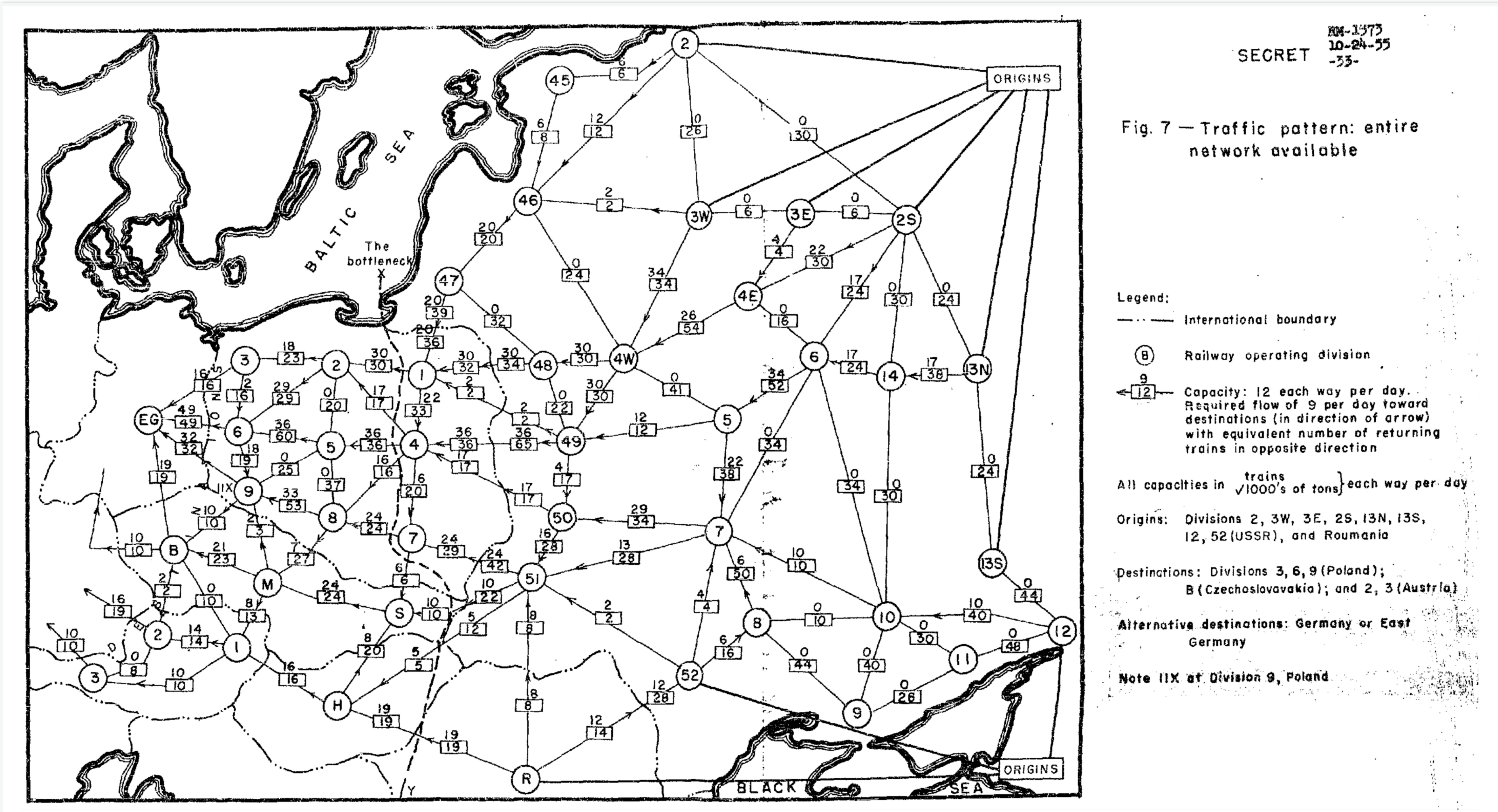


Image Credits: — Jeff Erickson's book and T[homas] E. Harris and F[rank] S. Ross. Fundamentals of a method for evaluating rail net capacities. The RAND Corporation, Research Memorandum RM-1517, October 24, 1955. United States Government work in the public domain. http://www.dtic.mil/dtic/tr/fulltext/u2/093458.pdf

Ford-Fulkerson Algorithm

Towards a Max-Flow Algorithm

We will design a max-flow algorithm and show that there is a s-t cut s.t. value of flow computed by algorithm $=$ capacity of cut

- Let's start with a greedy approach:
  - Pick an s-t path and push as much flow as possible down it
  - Repeat until you get stuck

Note: This won't actually work, but it gives us a sense of what we need to keep track of to improve it

Towards a Max-Flow Algorithm

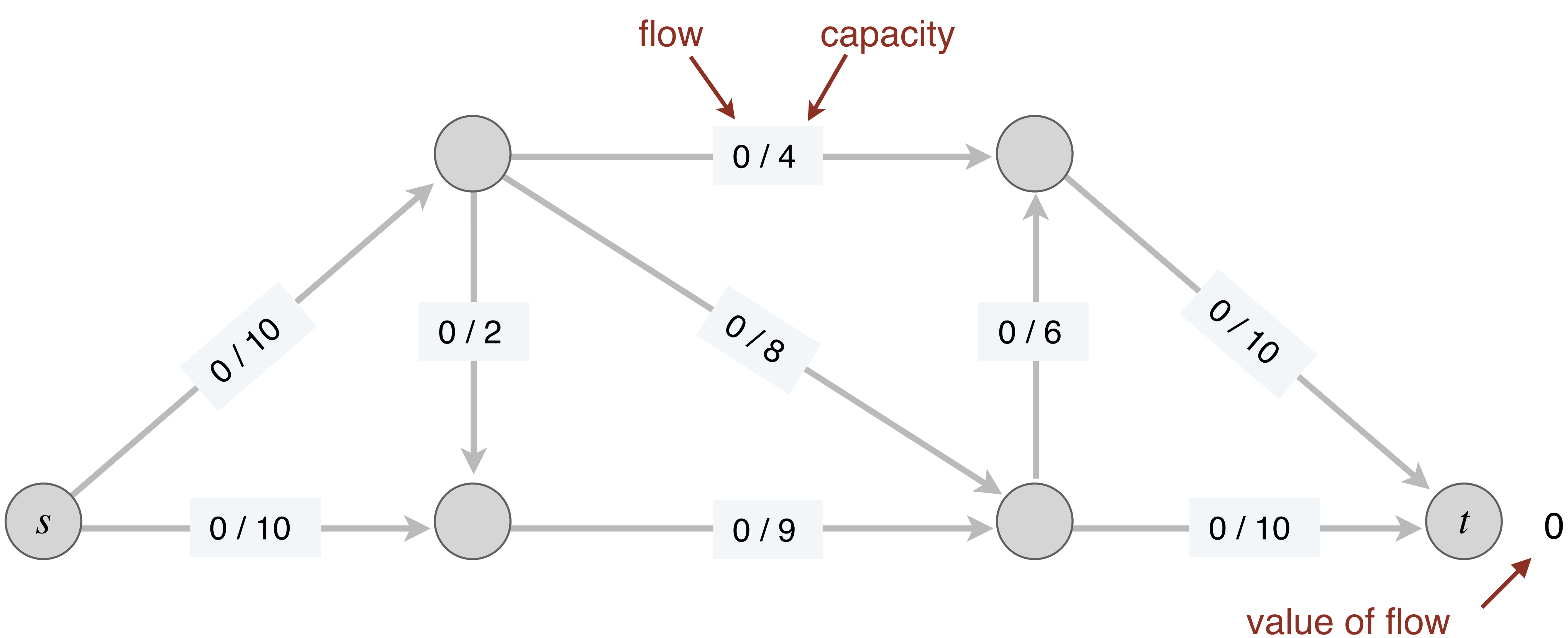
Greedy strategy:

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

- Let's explore an example

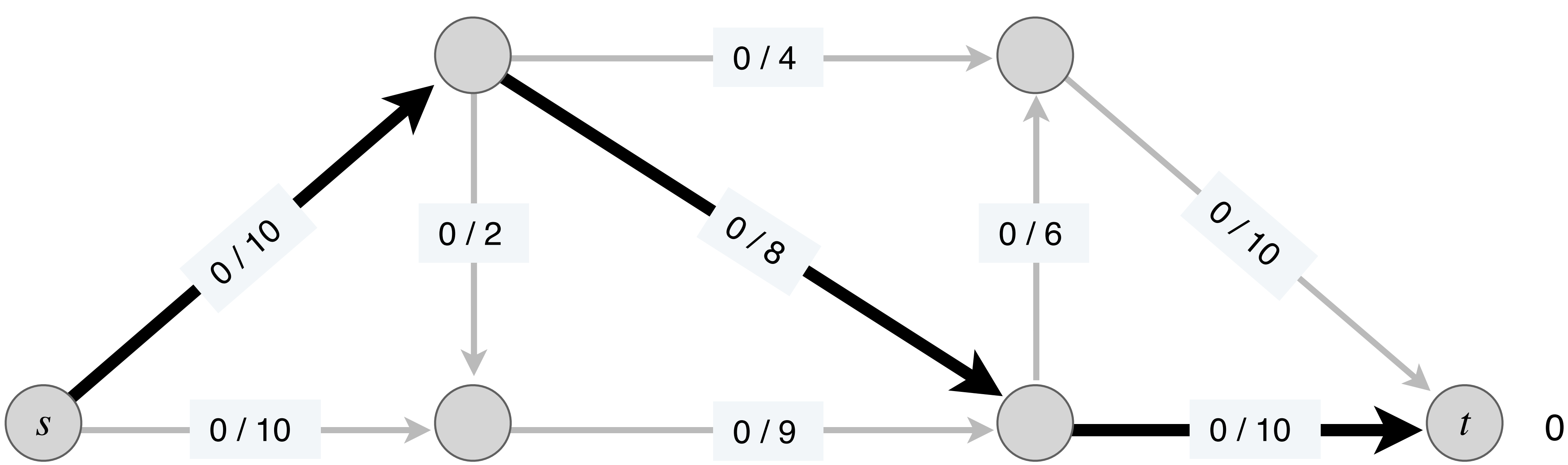
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



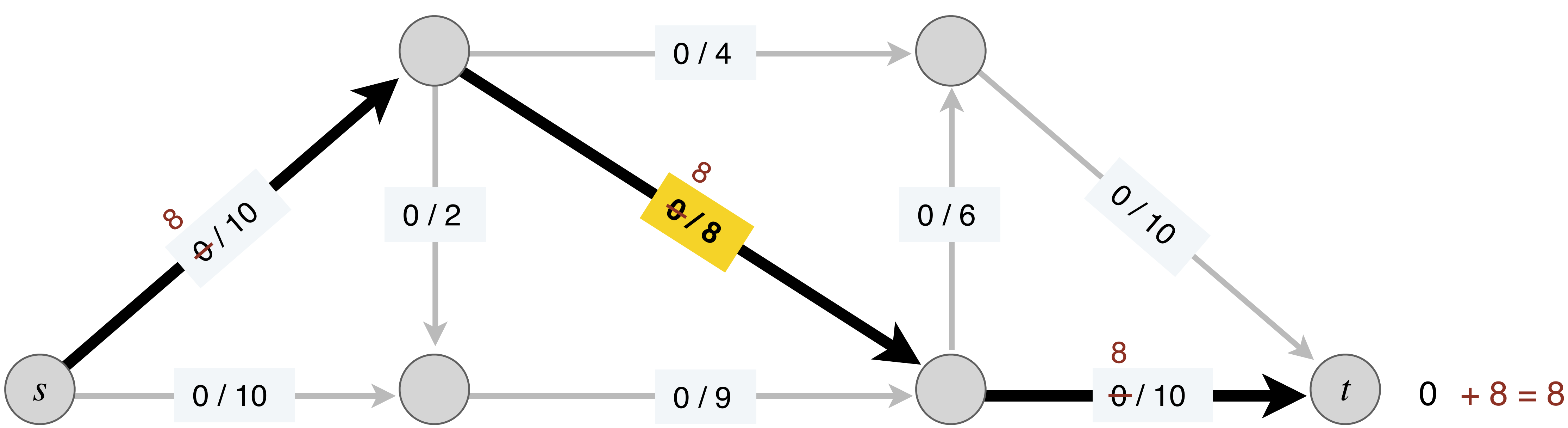
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



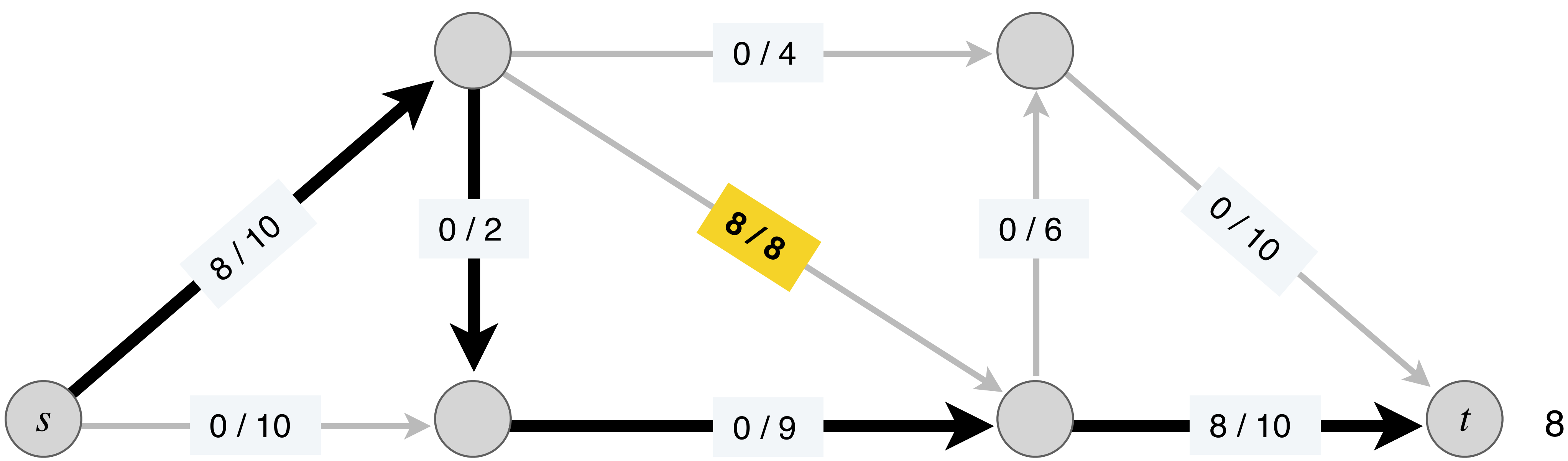
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



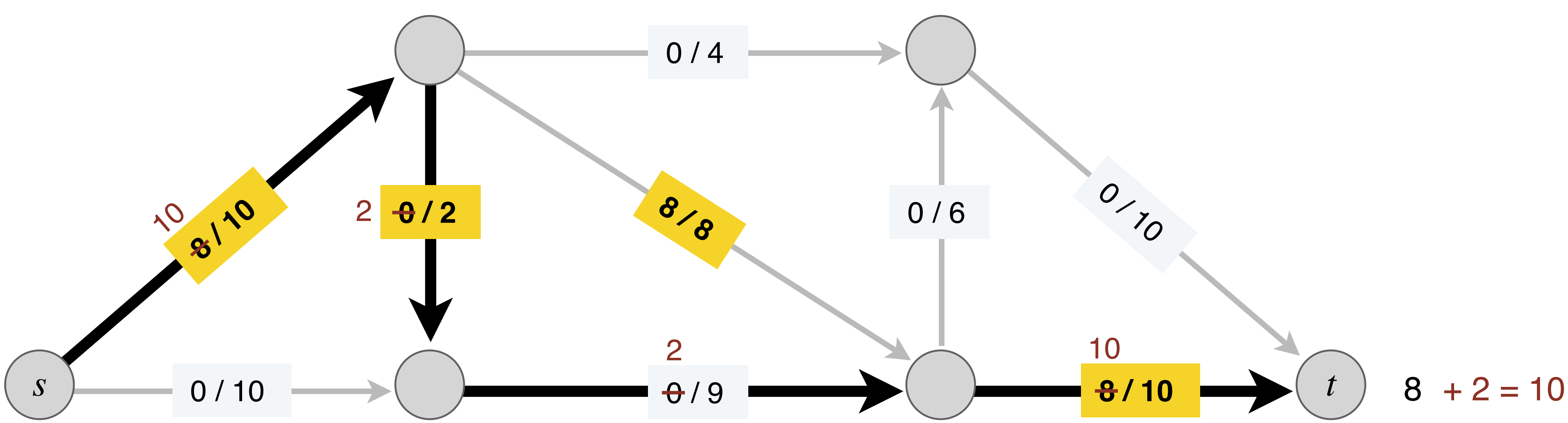
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



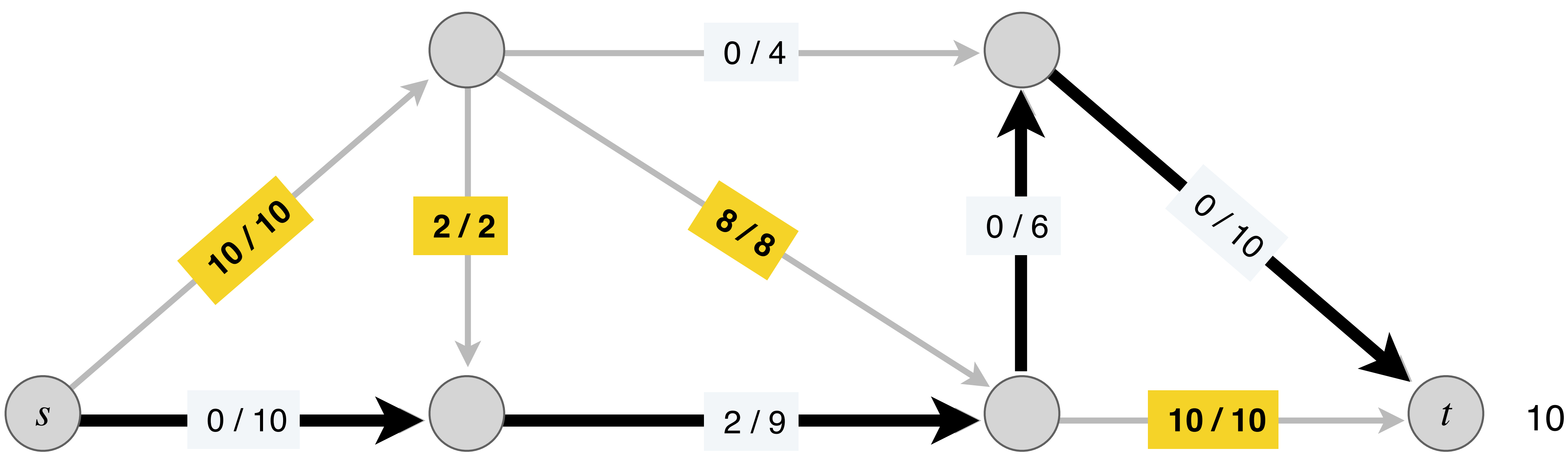
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

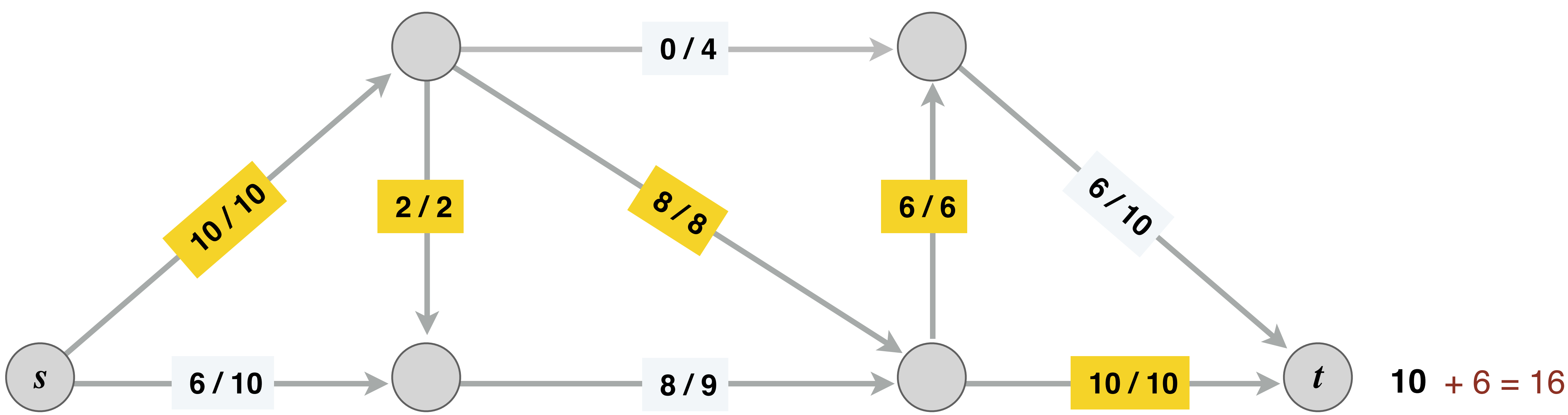


Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

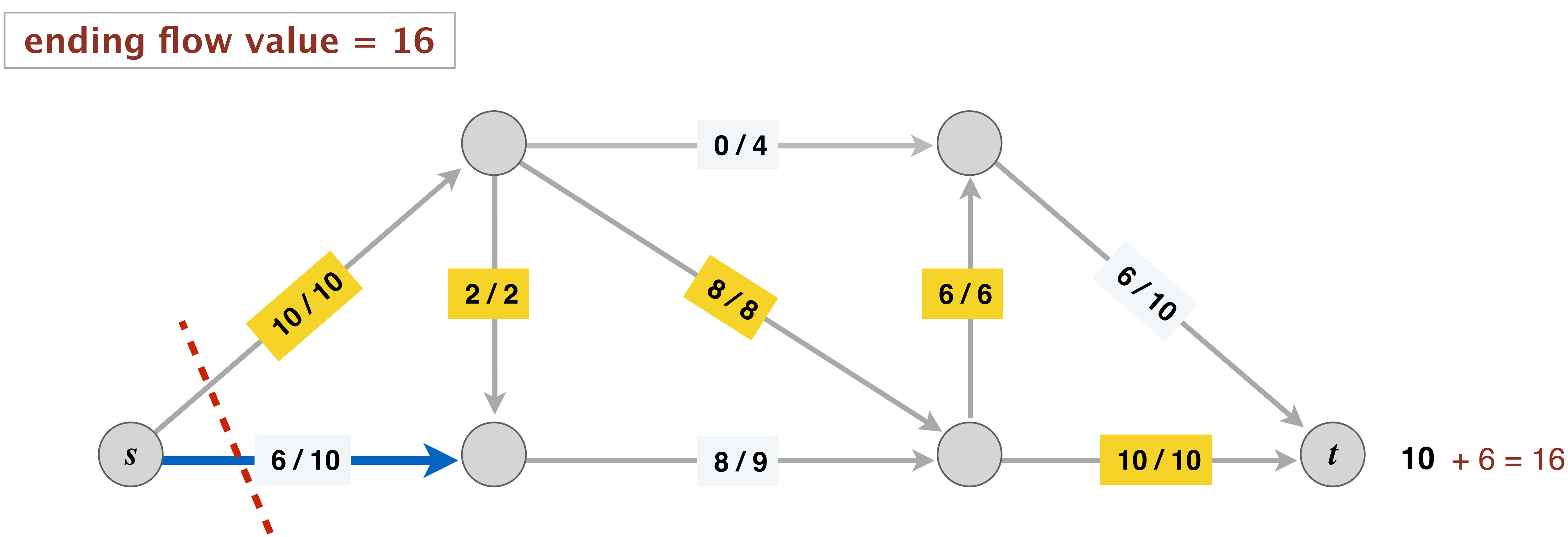
Is this the best we can do?

ending flow value = 16



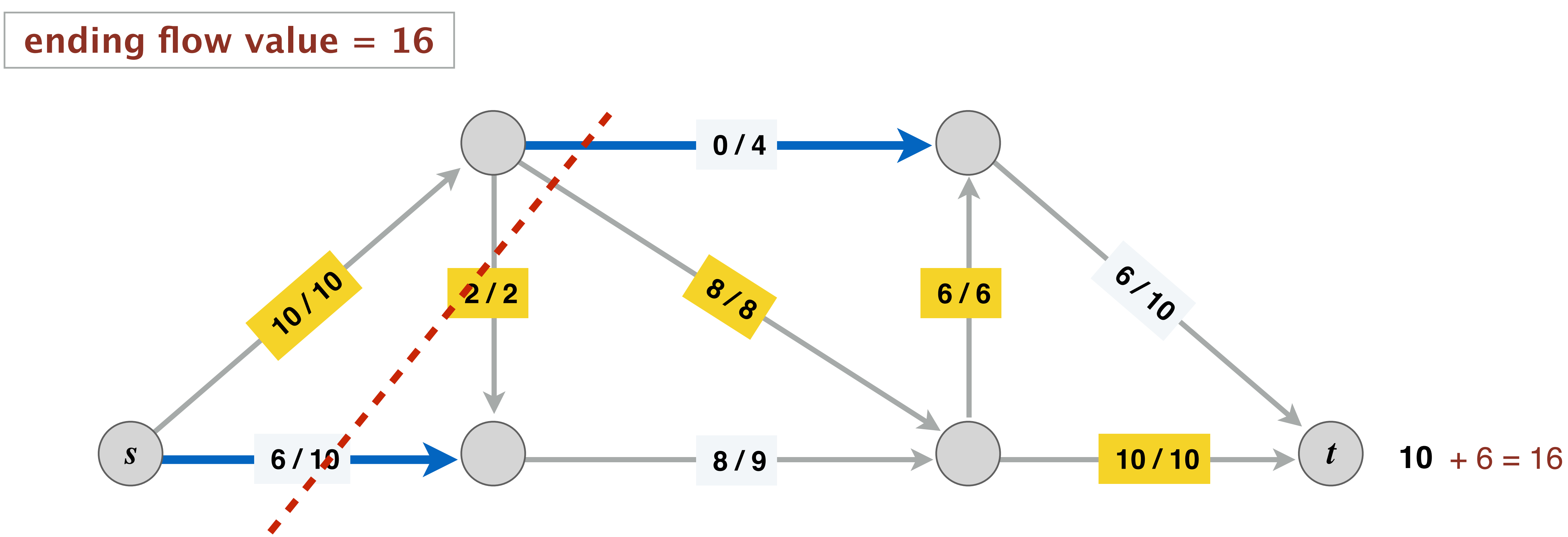
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Towards a Max-Flow Algorithm

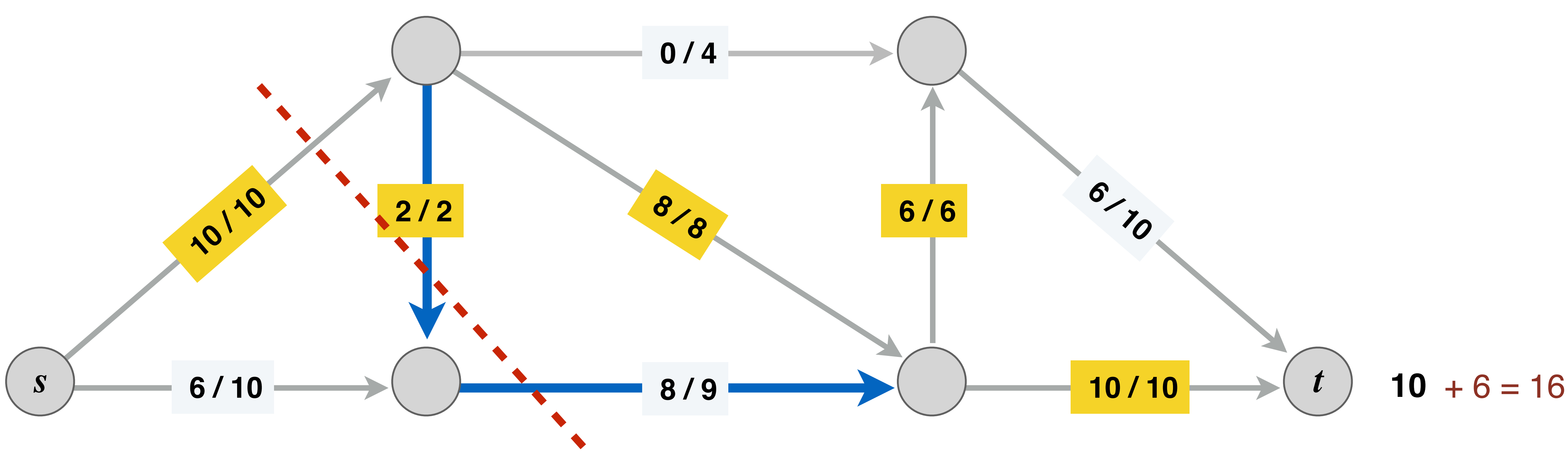
- Start with $f(e) = 0$ for each edge
- Find an $s \leadsto t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

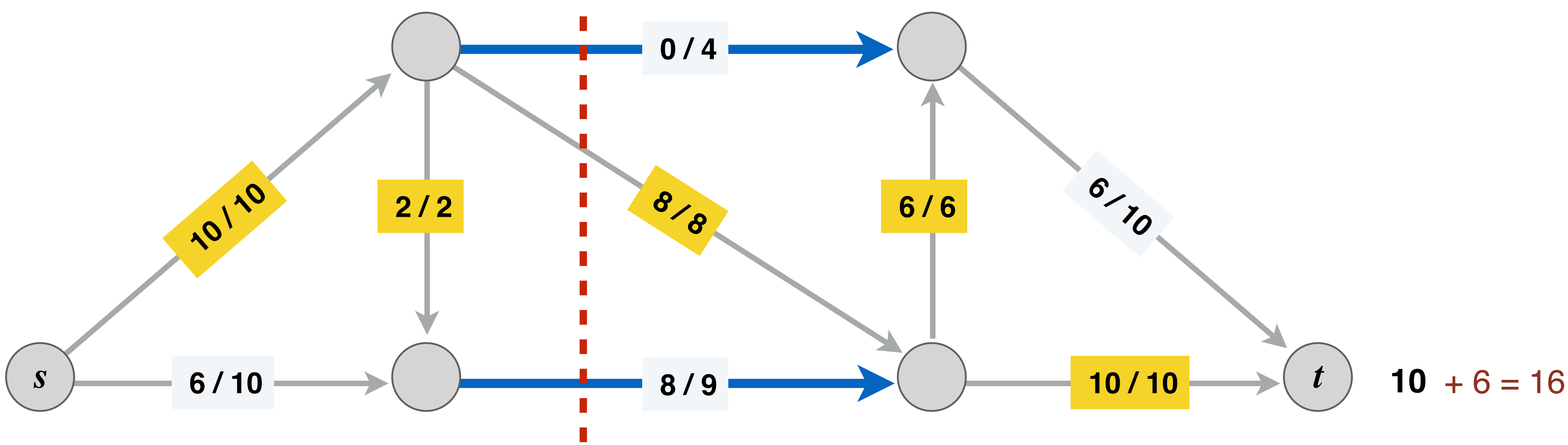
ending flow value = 16



Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

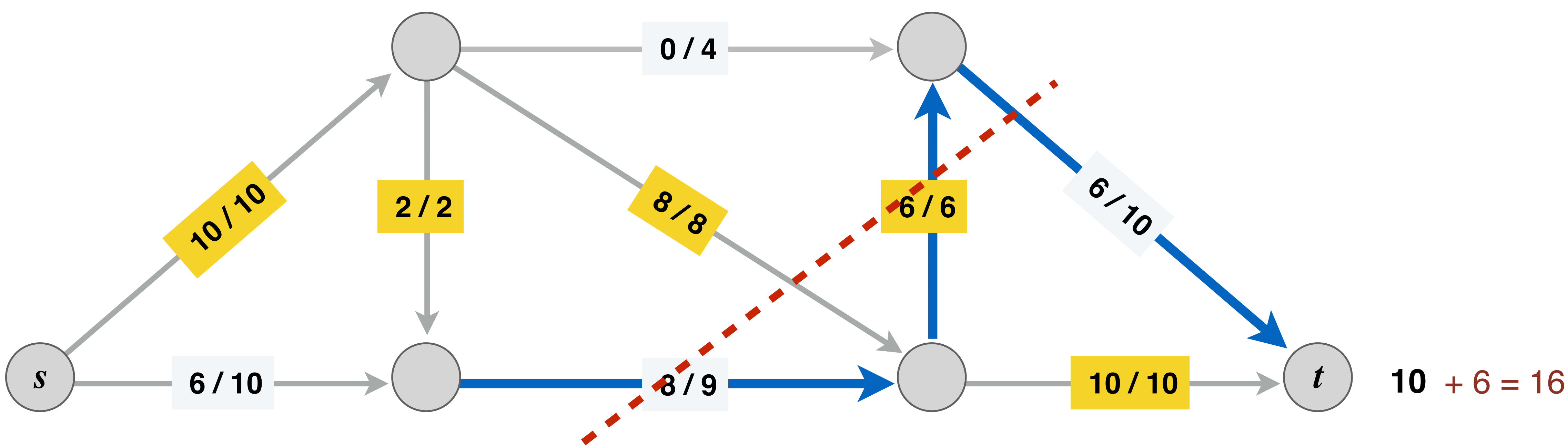
ending flow value = 16



Towards a Max-Flow Algorithm

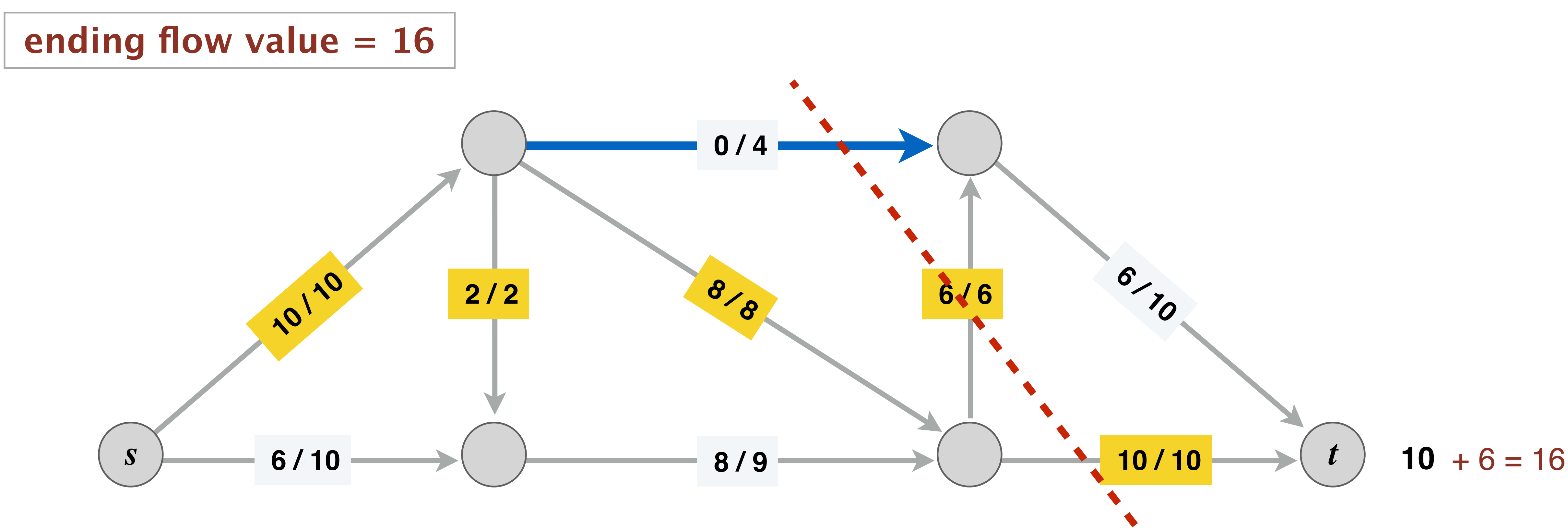
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

ending flow value = 16



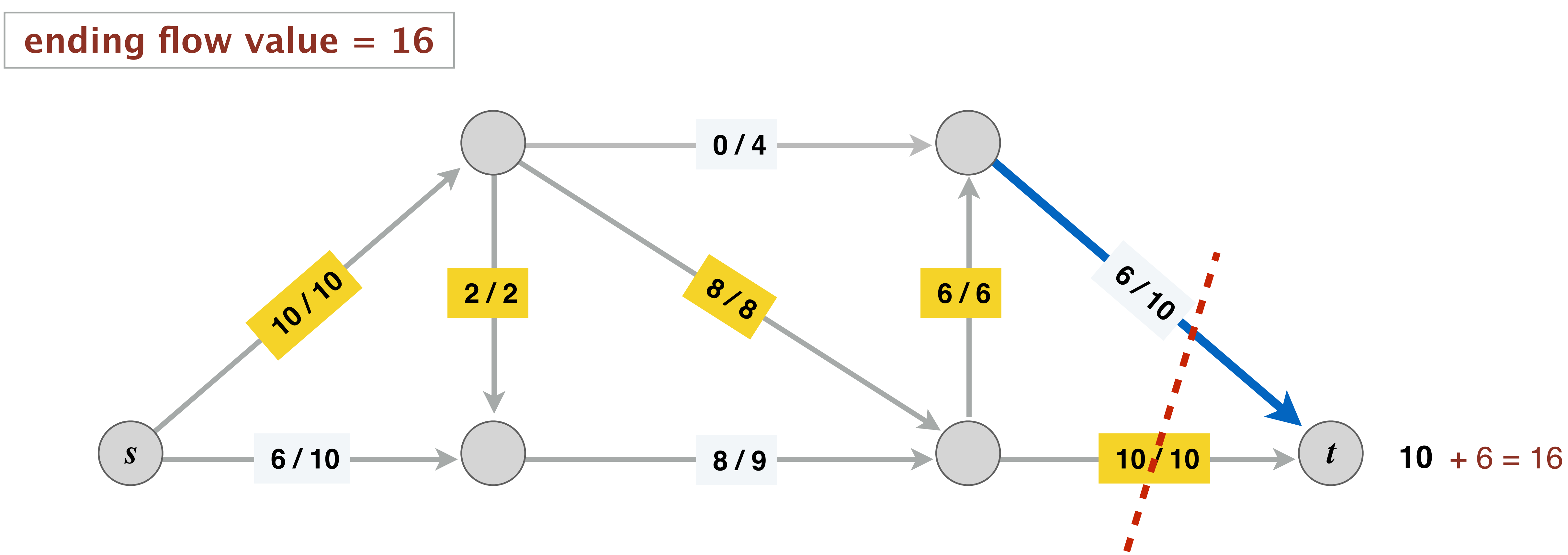
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



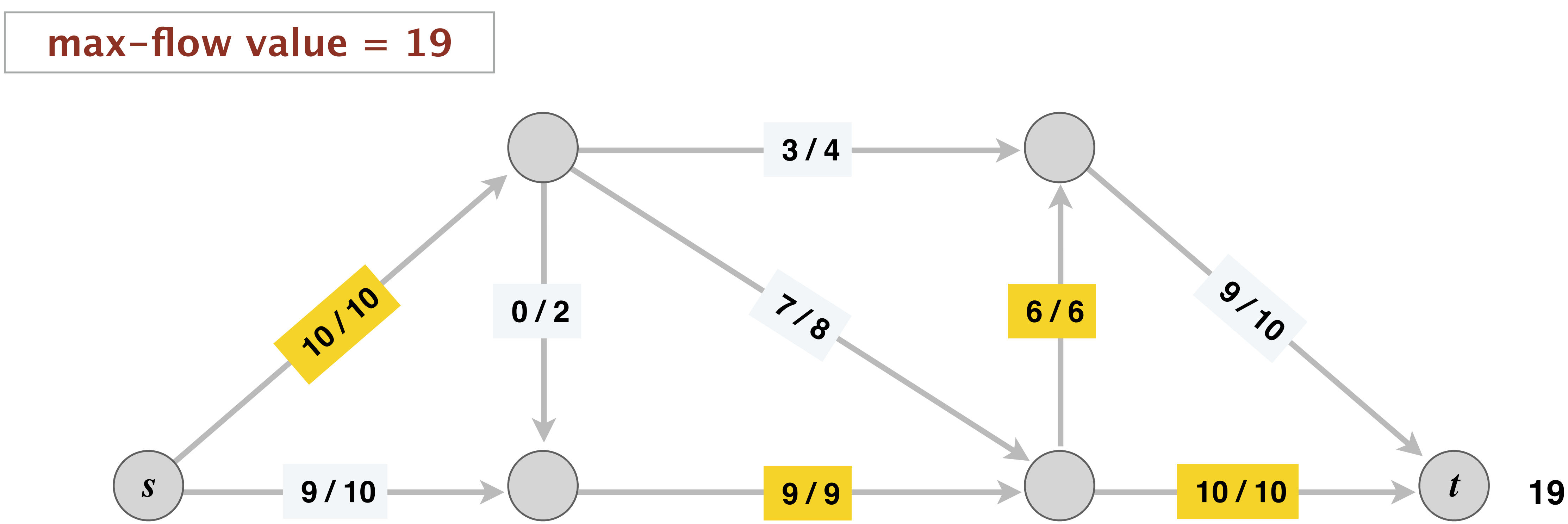
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



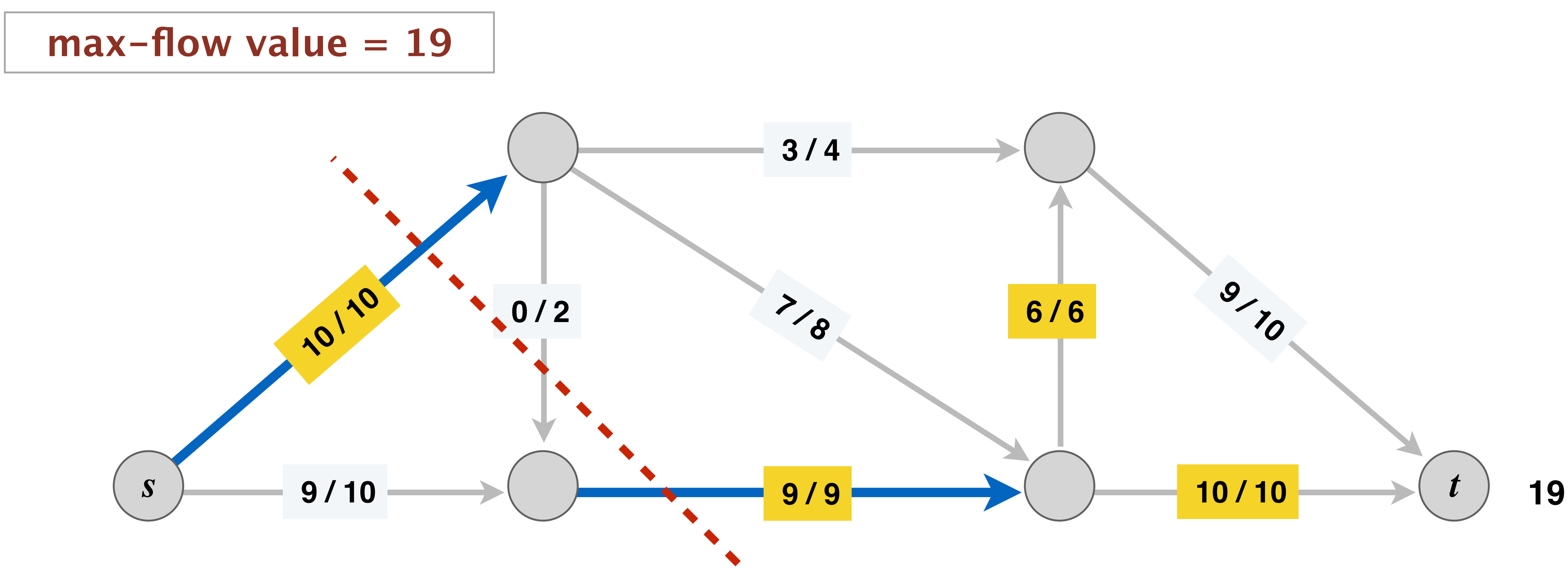
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Towards a Max-Flow Algorithm

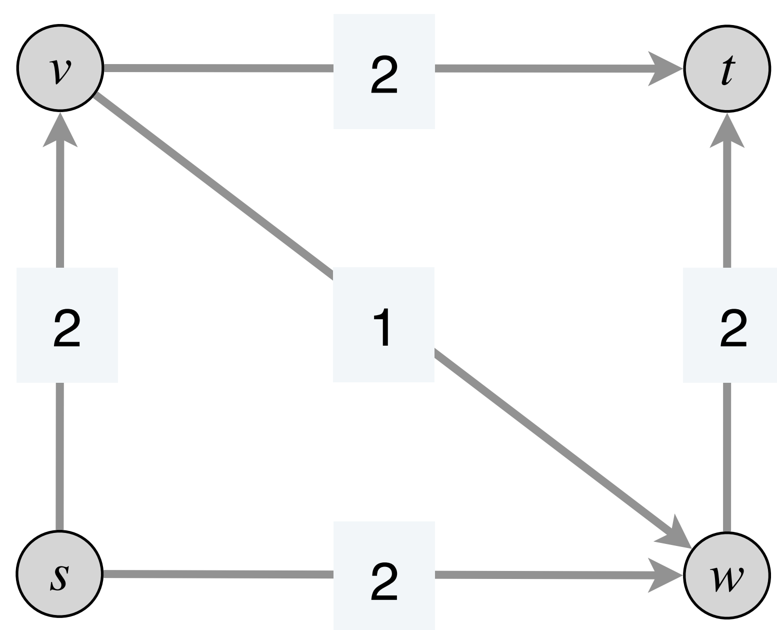
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Why Greedy Fails

Problem: greedy can never "undo" a bad flow decision

- Consider the following flow network



  - Greedy could choose $s \rightarrow v \rightarrow w \rightarrow t$ as first P

- **Takeaway**: Need a mechanism to "undo" bad flow decisions

Ford-Fulkerson Algorithm

Ford Fulkerson: Idea

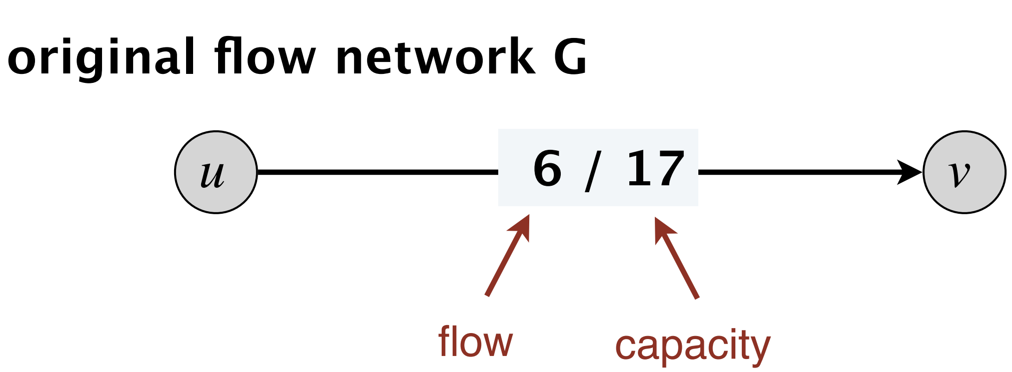
Goal: Want to make "forward progress" while letting ourselves undo previous decisions if they're getting in our way

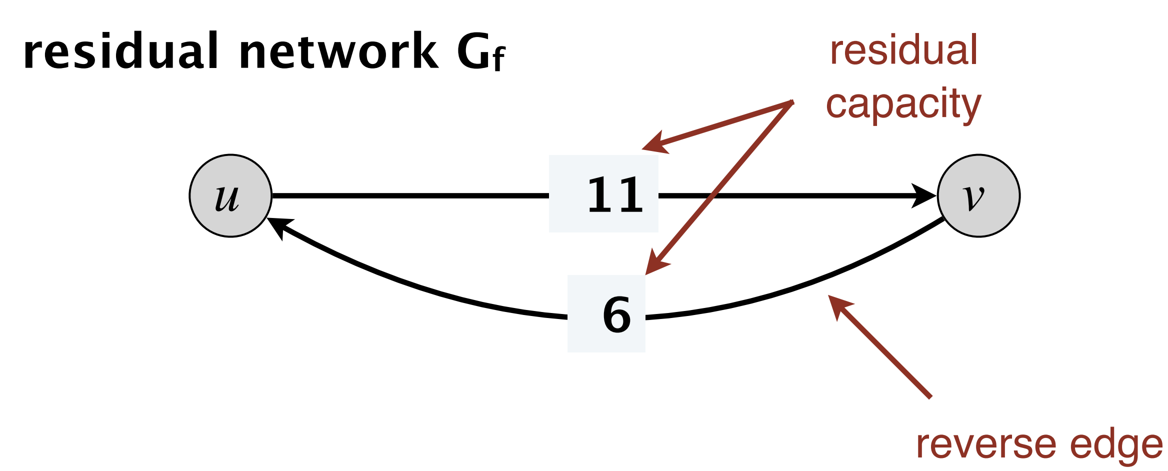
- **Idea**: keep track of where we can push flow
    - Can push more flow along any edge with remaining capacity
    - Can also push flow "back" along any edge that already has flow down it (undo a previous flow push)
- We need a way to systematically track these decisions

Residual Graph

Given flow network $G = (V, E, c)$ and a feasible flow f on G, **the residual graph** $G_f = (V, E_f, c_f)$ is defined as follows:

- Vertices in G_f same as G
- **(Forward edge)** For $e \in E$ with residual capacity $c(e) - f(e) > 0$, create $e \in E_f$ with capacity $c(e) - f(e)$
- **(Backward edge)** For $e \in E$ with $f(e) > 0$, create $e_{\text{reverse}} \in E_f$ with capacity $f(e)$





Flow Algorithm Idea

- Now we have a residual graph that lets us make forward progress or push back existing flow
- We will look for $s \rightsquigarrow t$ paths in G_f rather than G
- Once we have a path, we will "augment" flow along it similar to greedy
  - find bottleneck capacity edge on the path and push that much flow through it in G_f
- When we translate this back to G, this means:
  - We increment existing flow on a forward edge
  - Or we decrement flow on a backward edge

Augmenting Path & Flow

- An **augmenting path** P is a **simple** $s \rightsquigarrow t$ path in the residual graph G_f

Path that repeats no vertices

- The **bottleneck capacity** b of an augmenting path P is the minimum capacity of any edge in P.

Some $s \rightsquigarrow t$ path P in G_f

AUGMENT(f, P)

$b \leftarrow$ bottleneck capacity of augmenting path P.

FOREACH edge $e \in P$:

  IF ($e \in E$, *that is*, e *is forward edge*)

    Increase $f(e)$ in G by b

  ELSE

    Decrease $f(e)$ in G by b

RETURN f.

If/else update flow in G, not G_f

Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for each edge $e \in E$
- Find a simple $s \rightsquigarrow t$ path P in the residual network G_f
- Augment flow along path P by bottleneck capacity b
- Repeat until you get stuck

FORD–FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

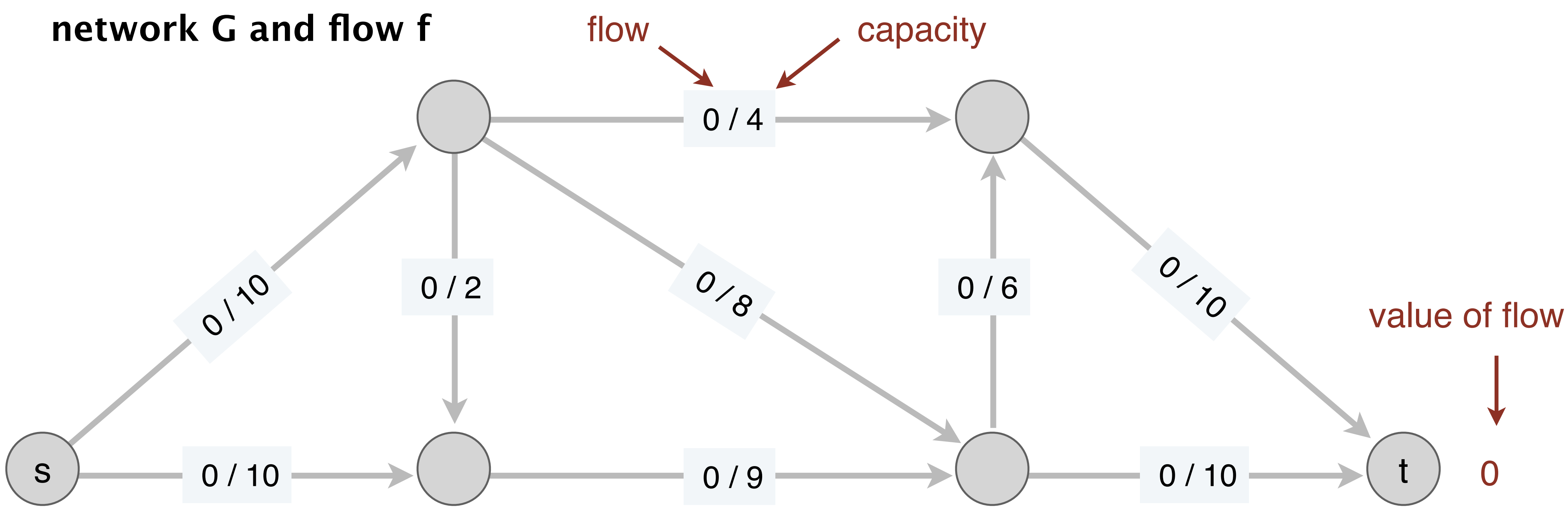
WHILE (there exists an s⤳t path P in G_f)

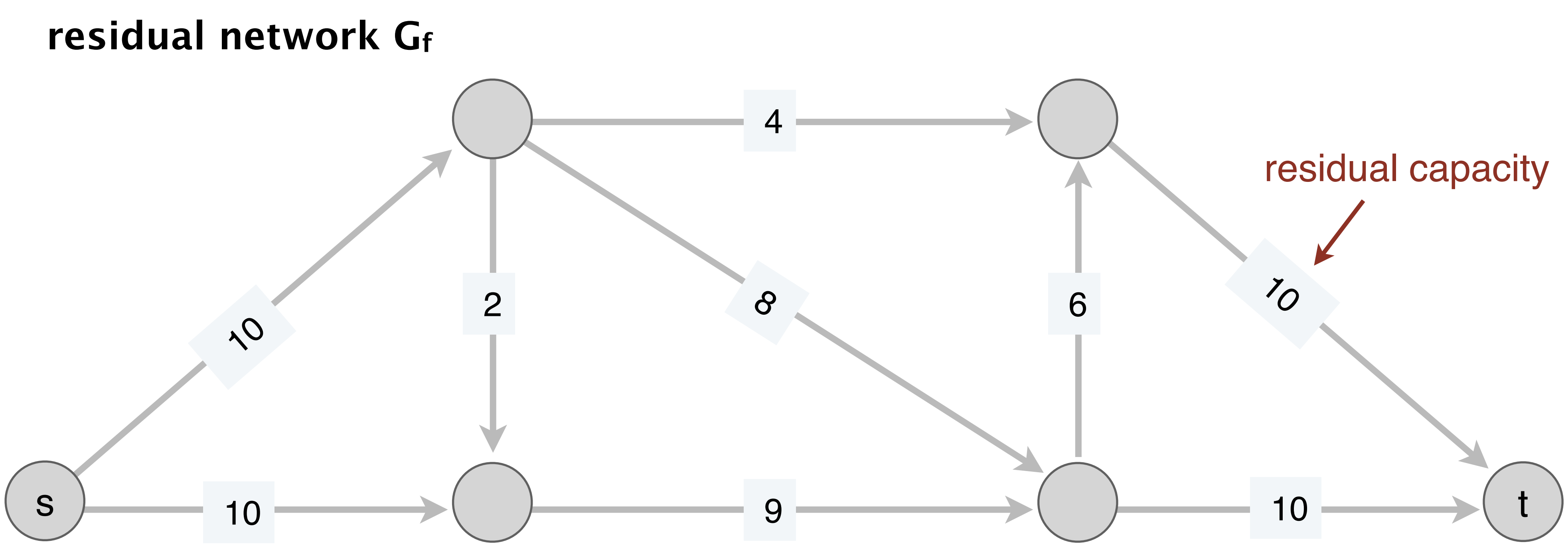
    $f \leftarrow$ AUGMENT(f, P).

    Update G_f.

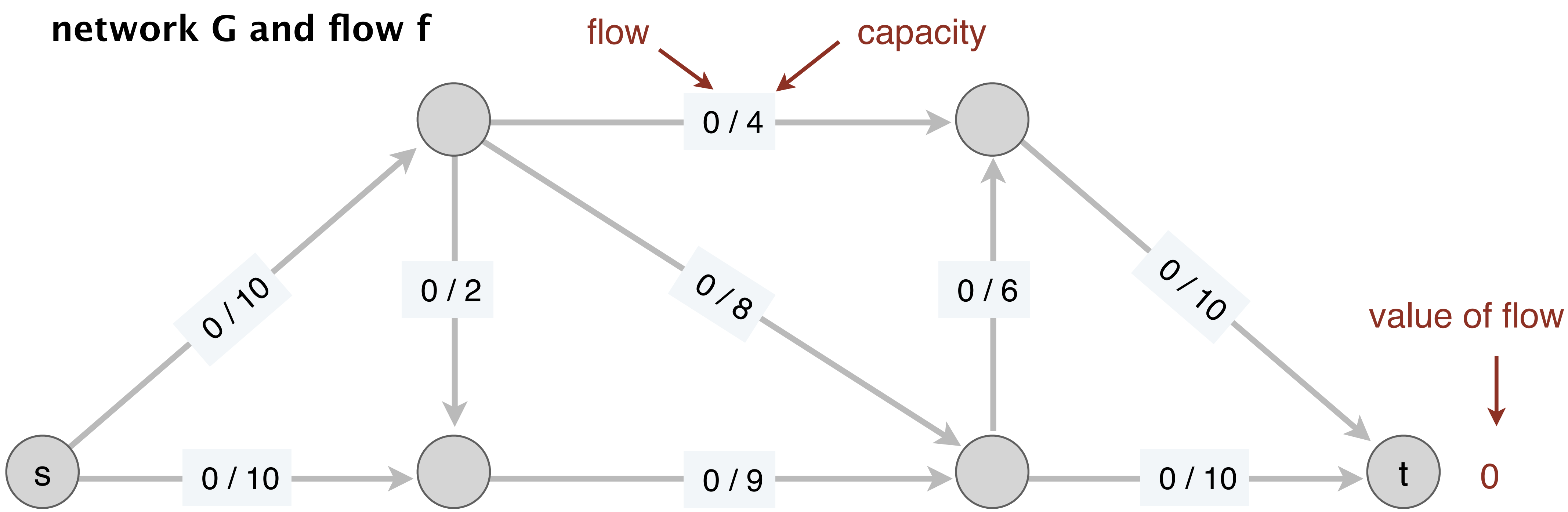
RETURN f.

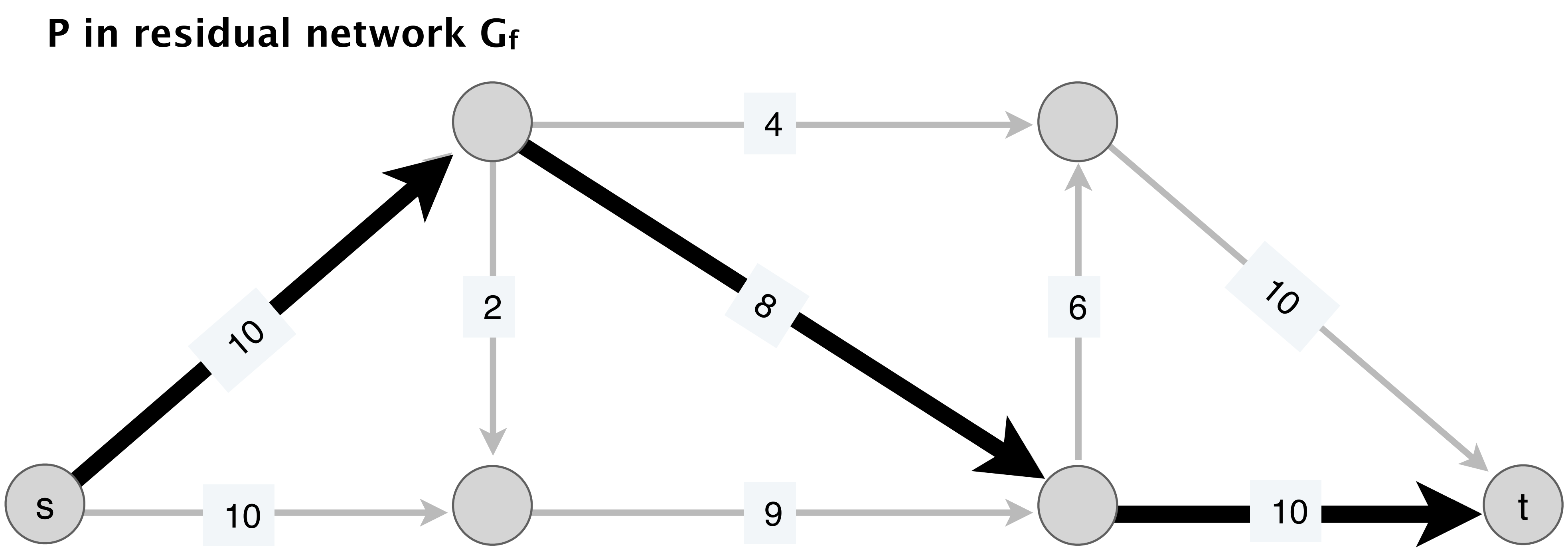
Ford-Fulkerson Example



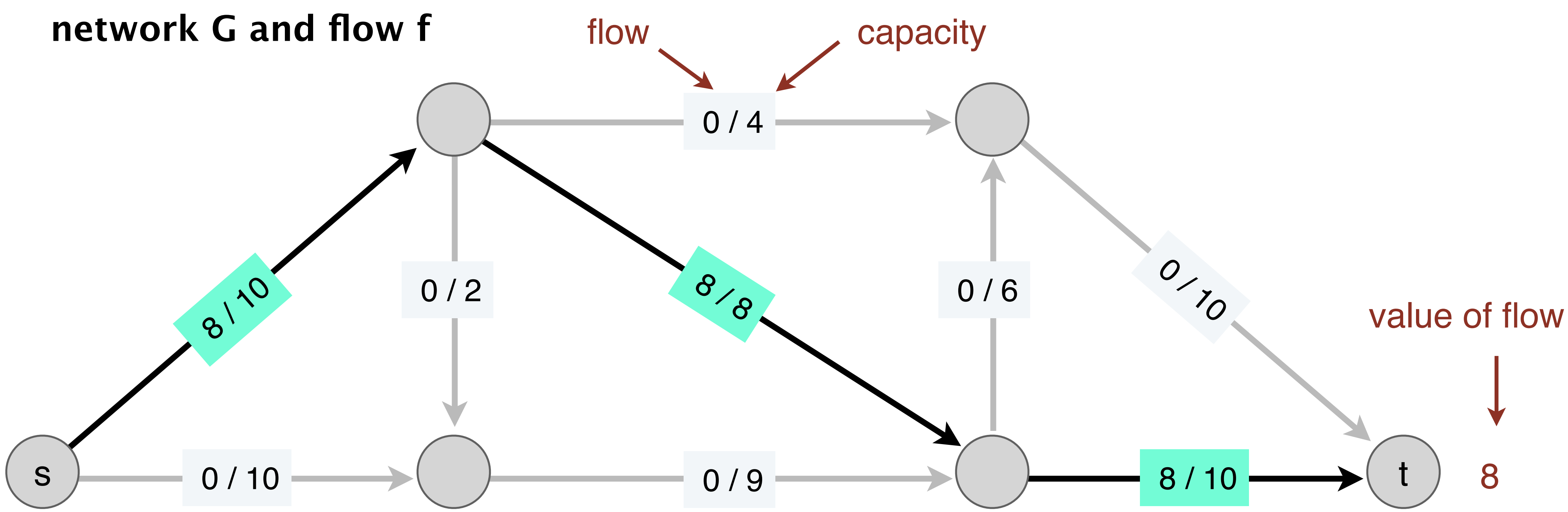


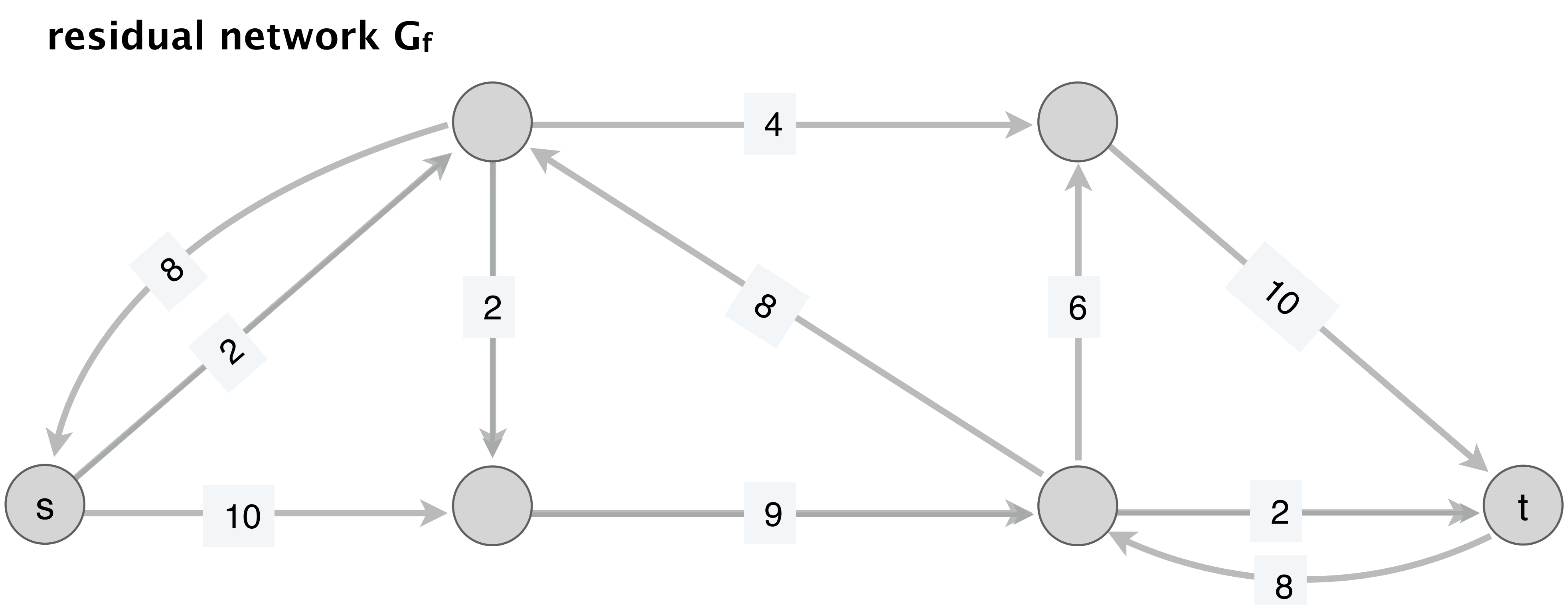
Ford-Fulkerson Example





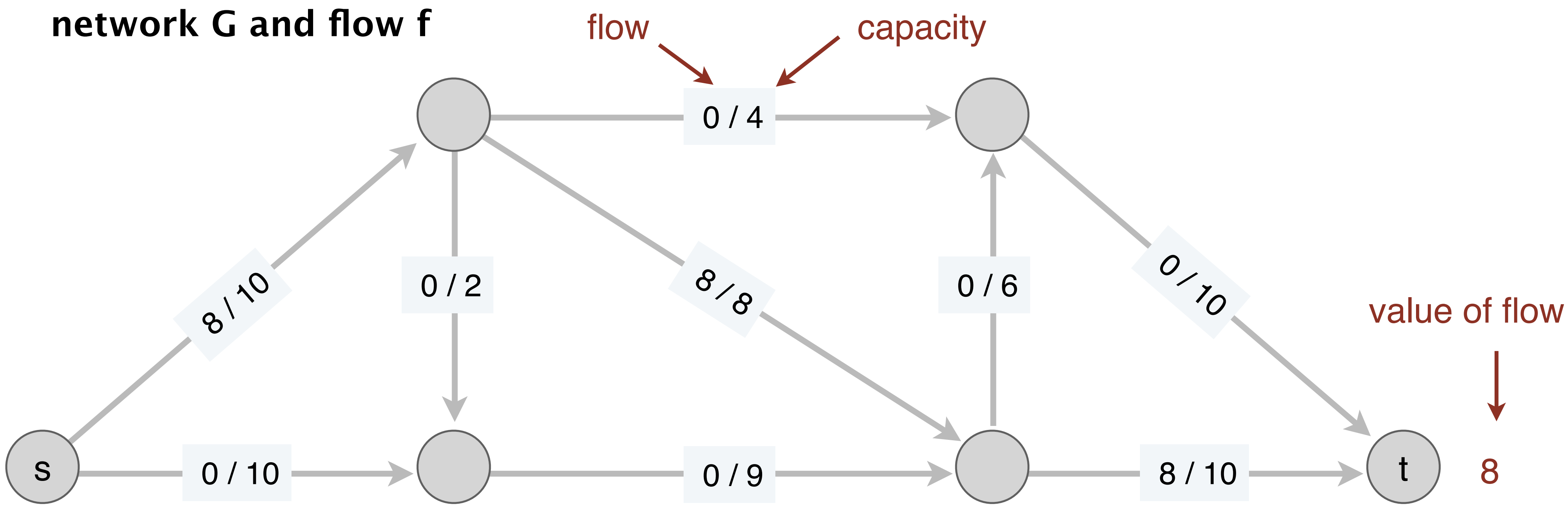
Ford-Fulkerson Example



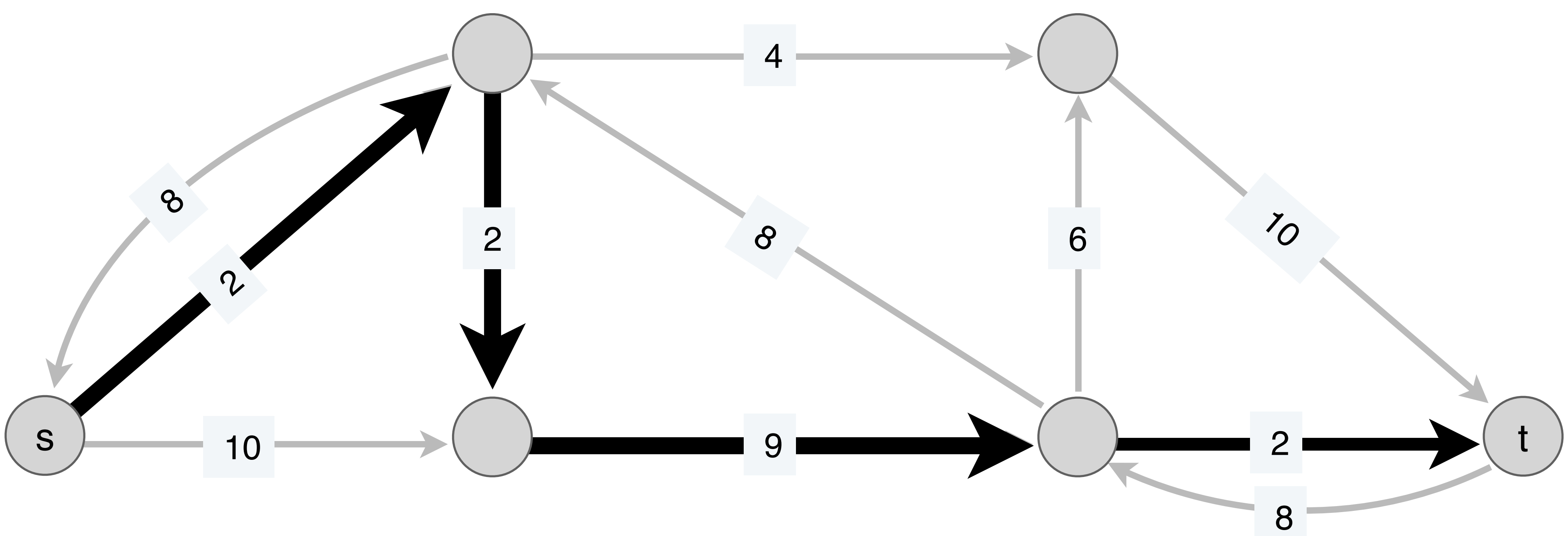


Ford-Fulkerson Example

network G and flow f

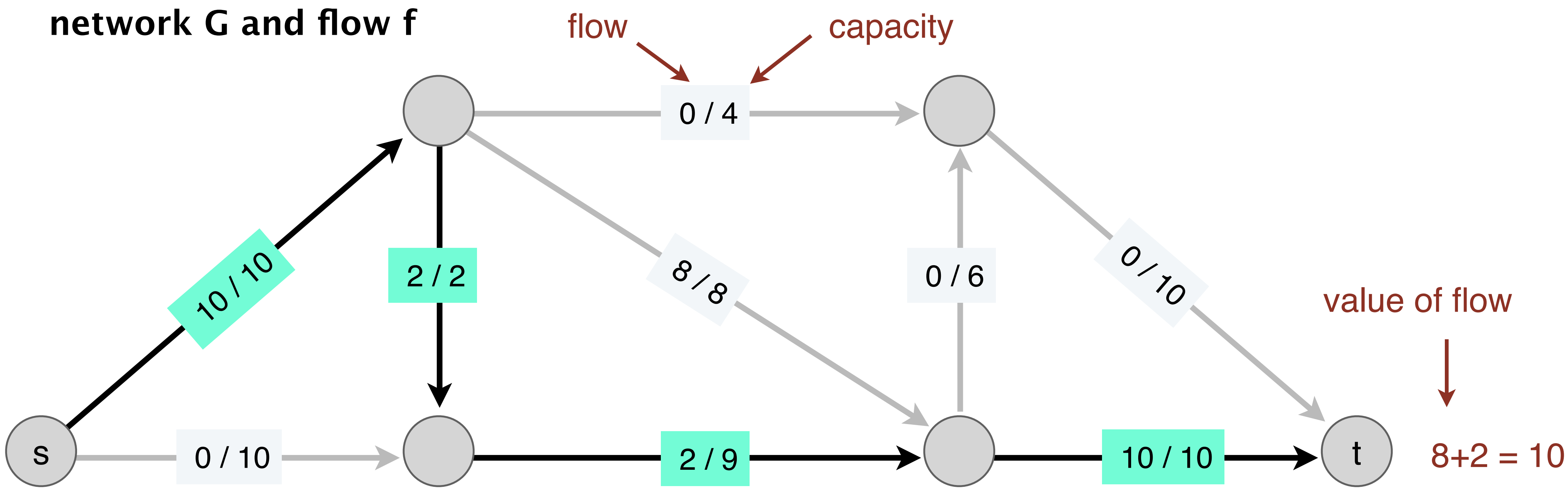


P in residual network G_f

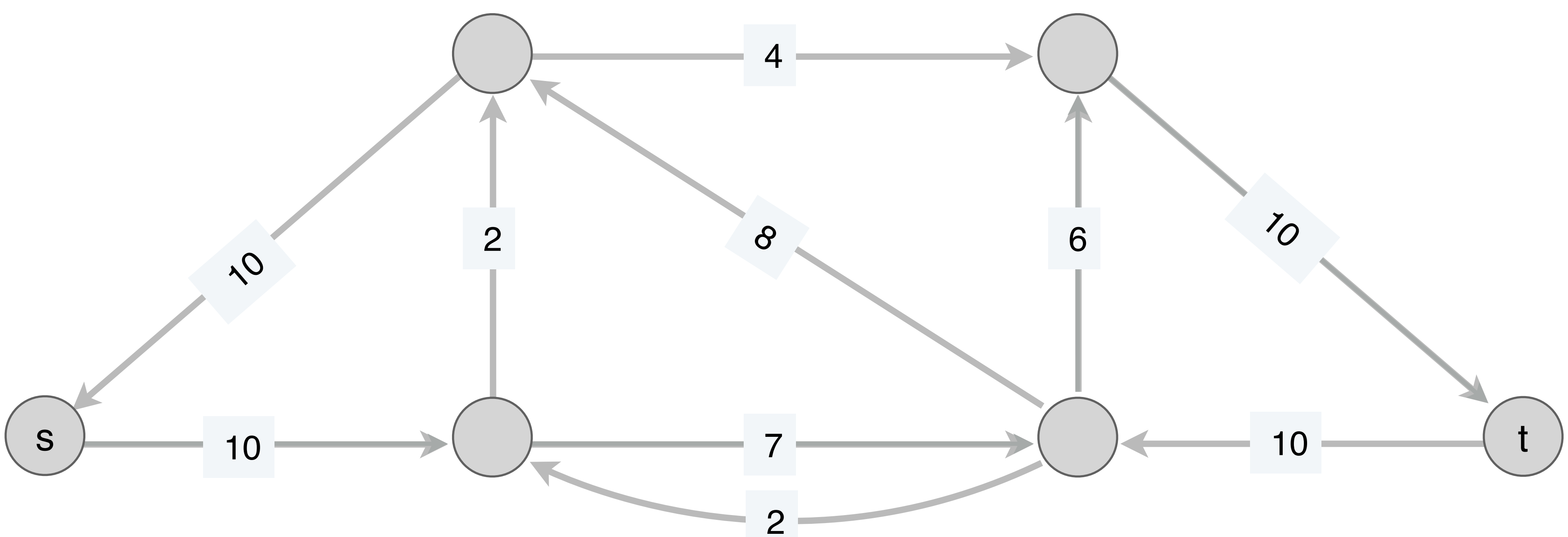


Ford-Fulkerson Example

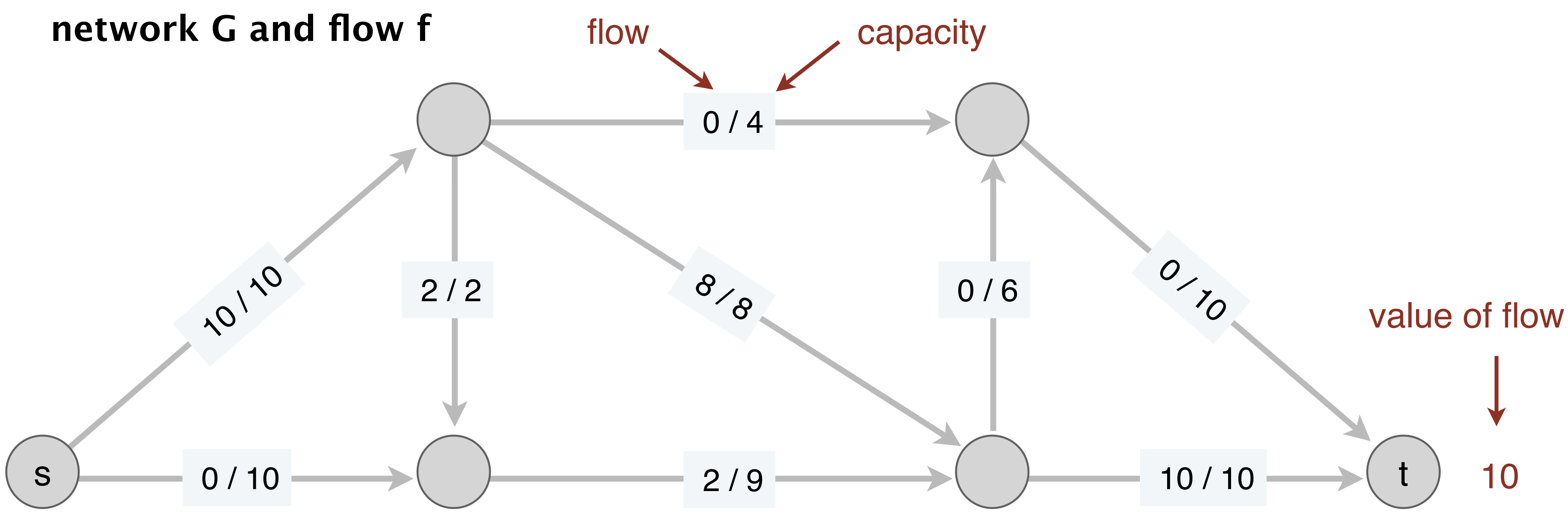
network G and flow f

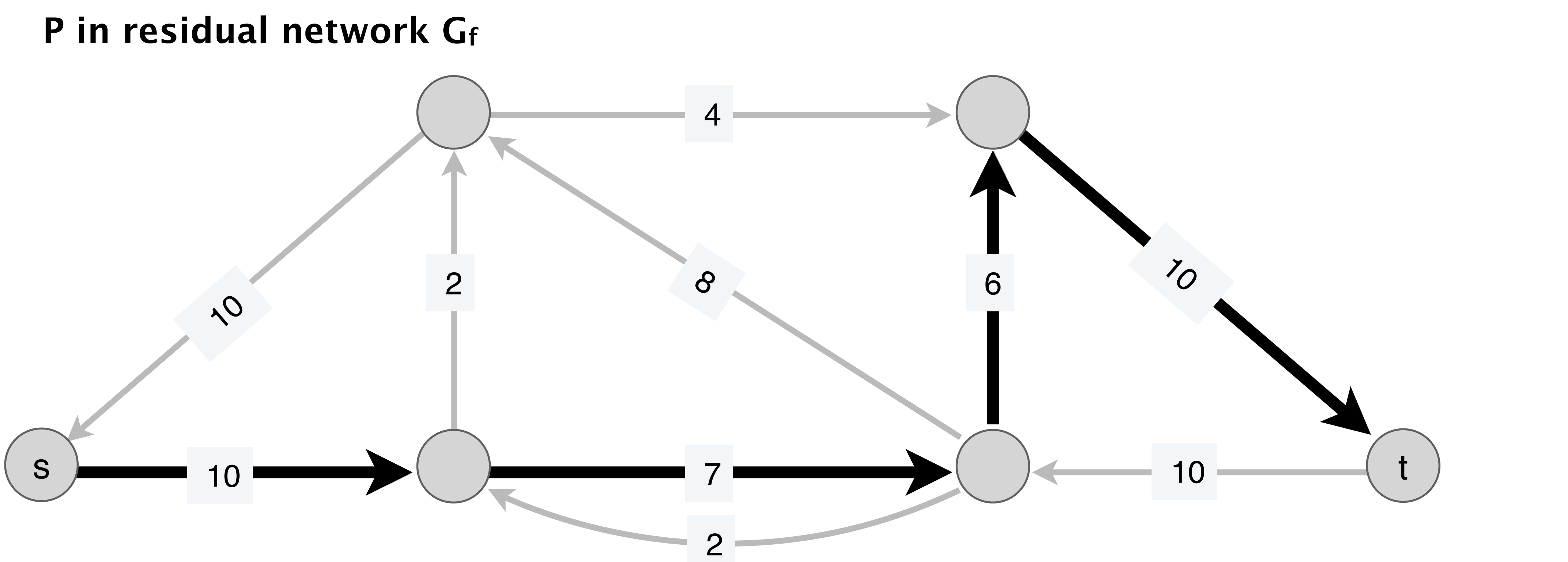


residual network G_f



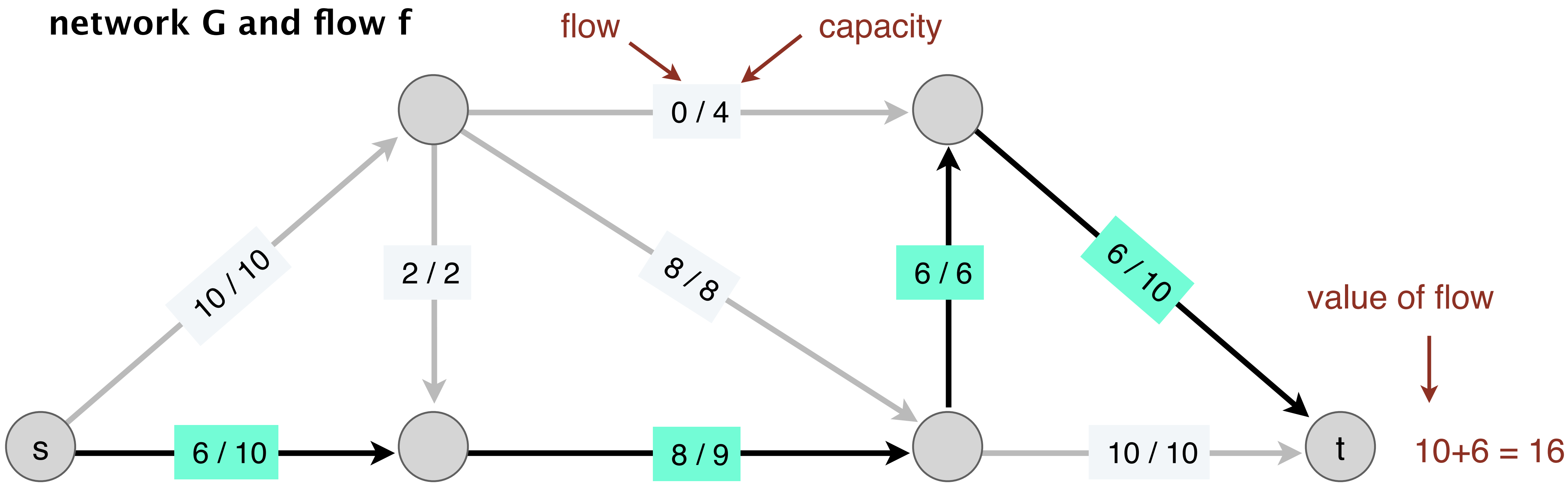
Ford-Fulkerson Example



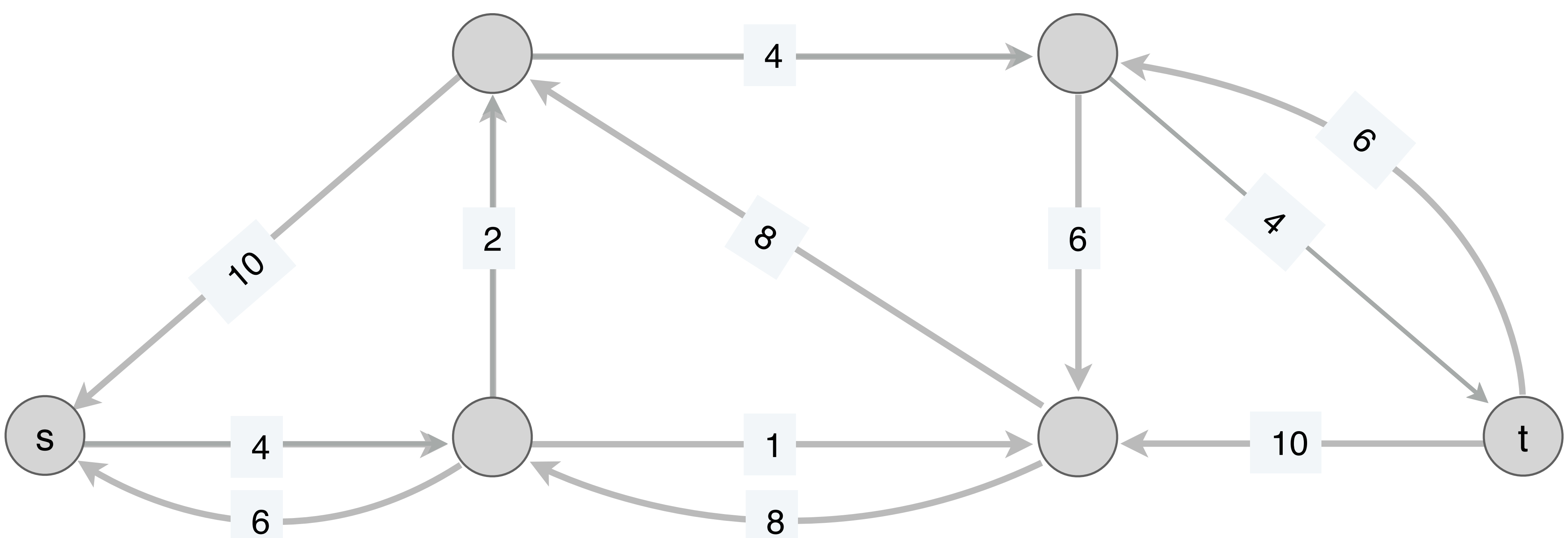


Ford-Fulkerson Example

network G and flow f

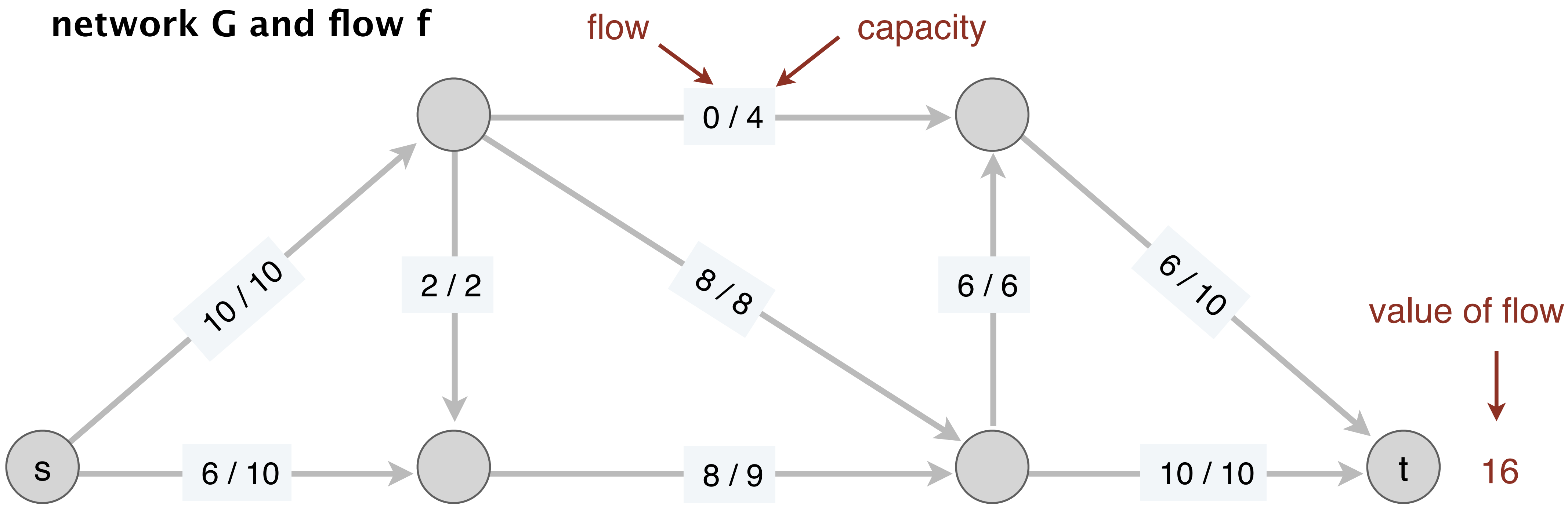


residual network G_f

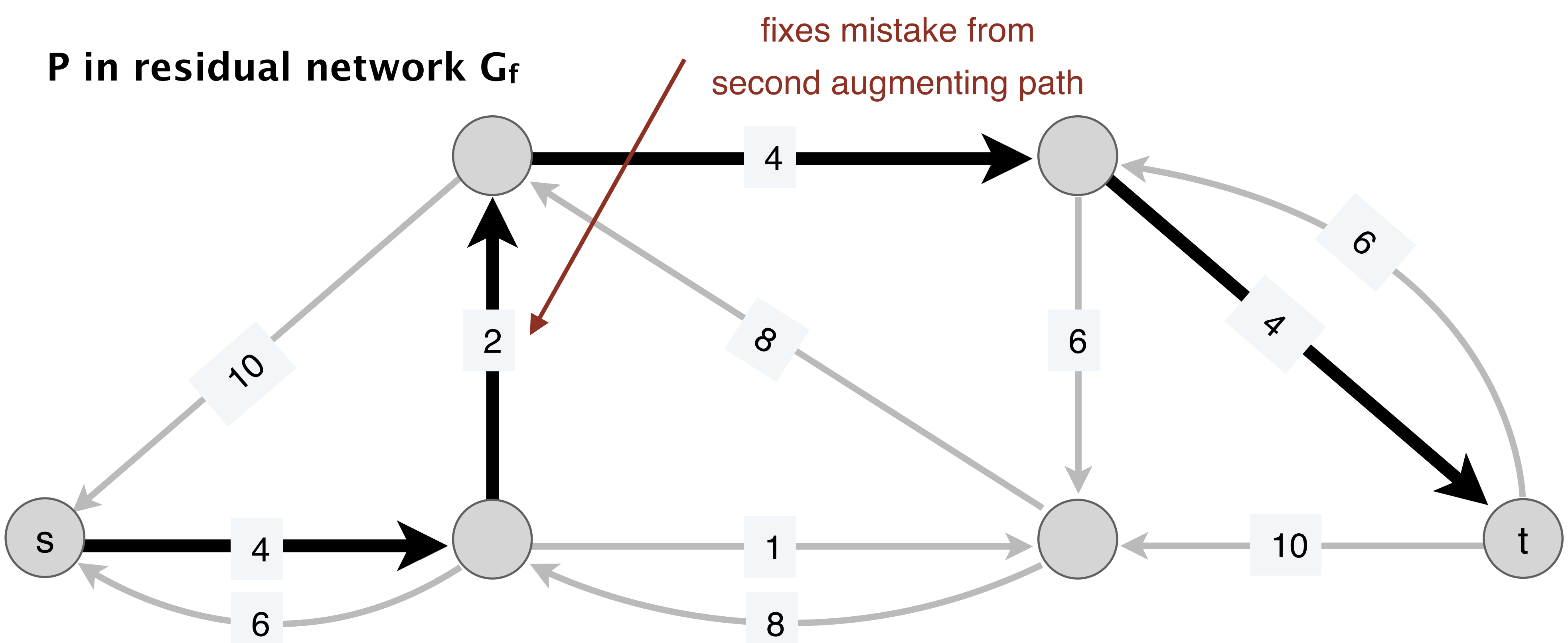


Ford-Fulkerson Example

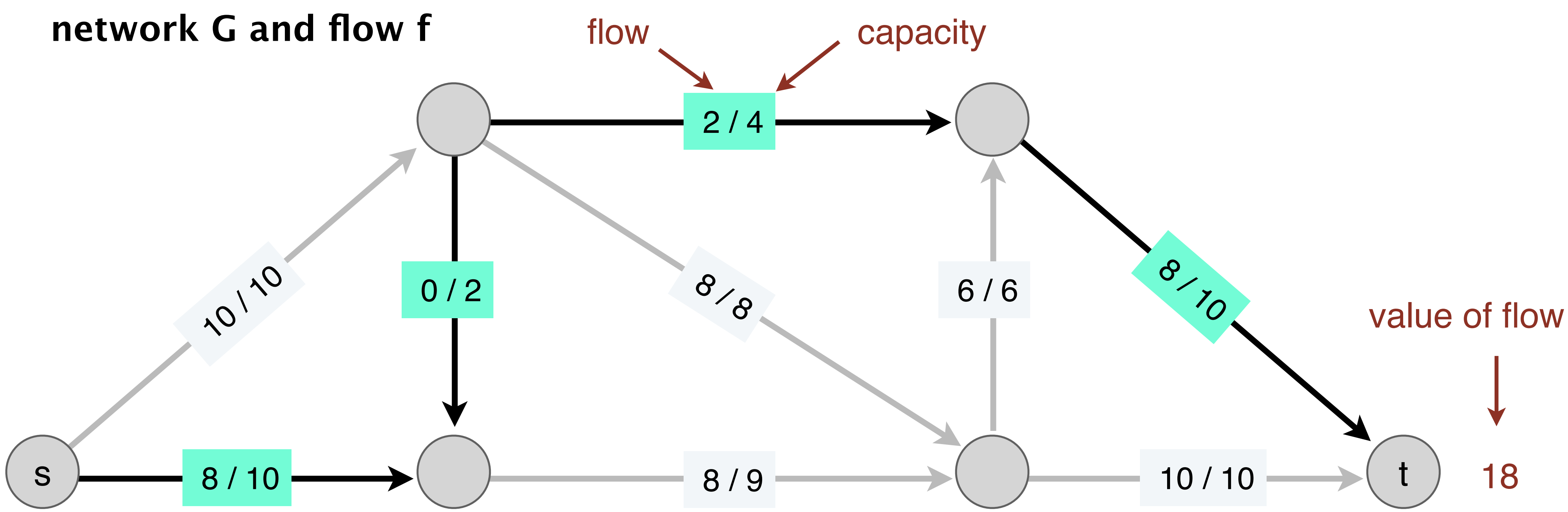
network G and flow f

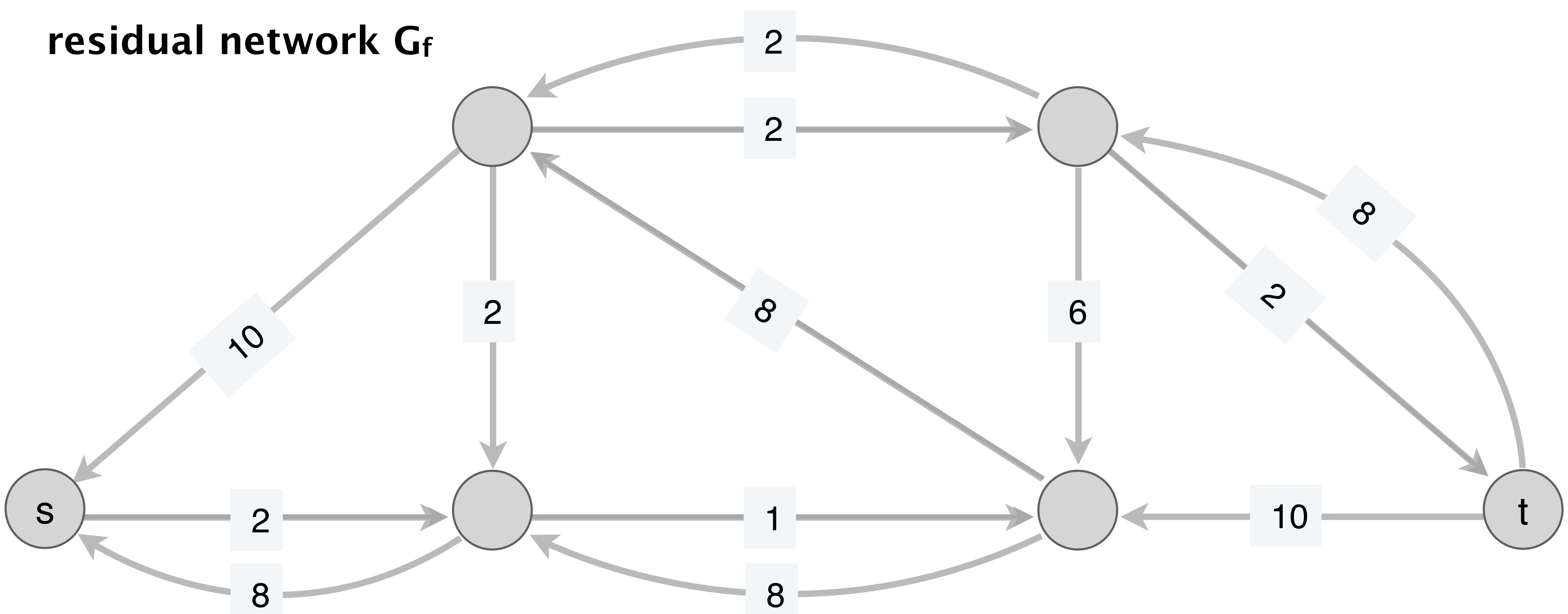


P in residual network G_f

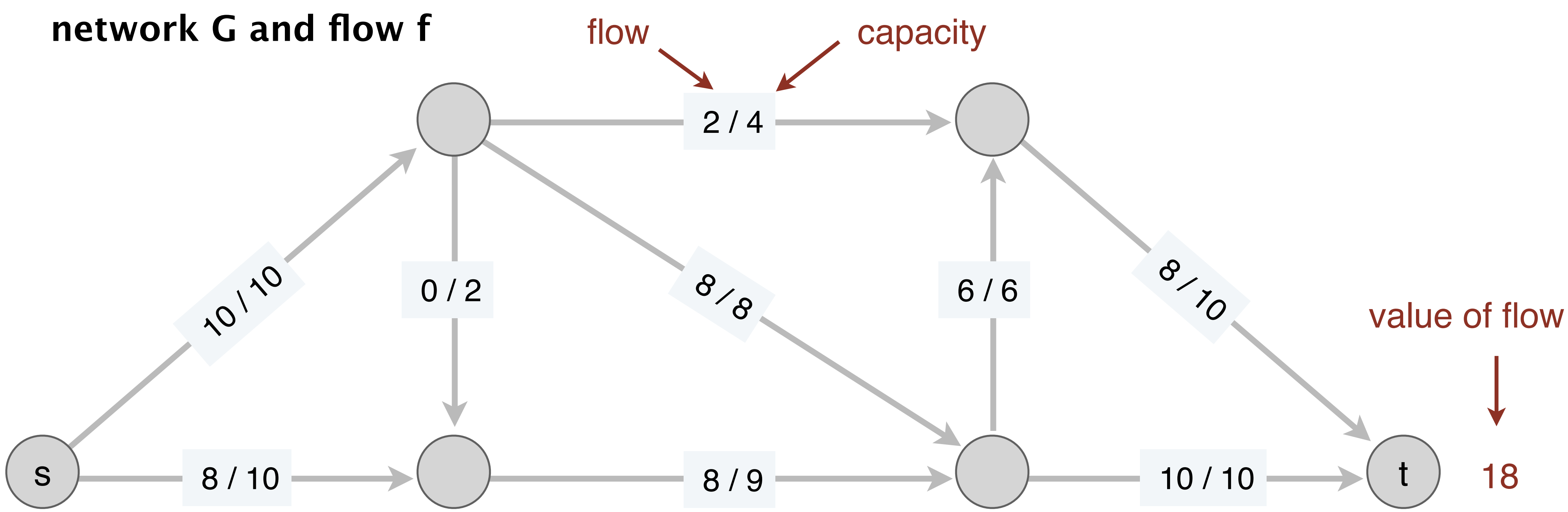


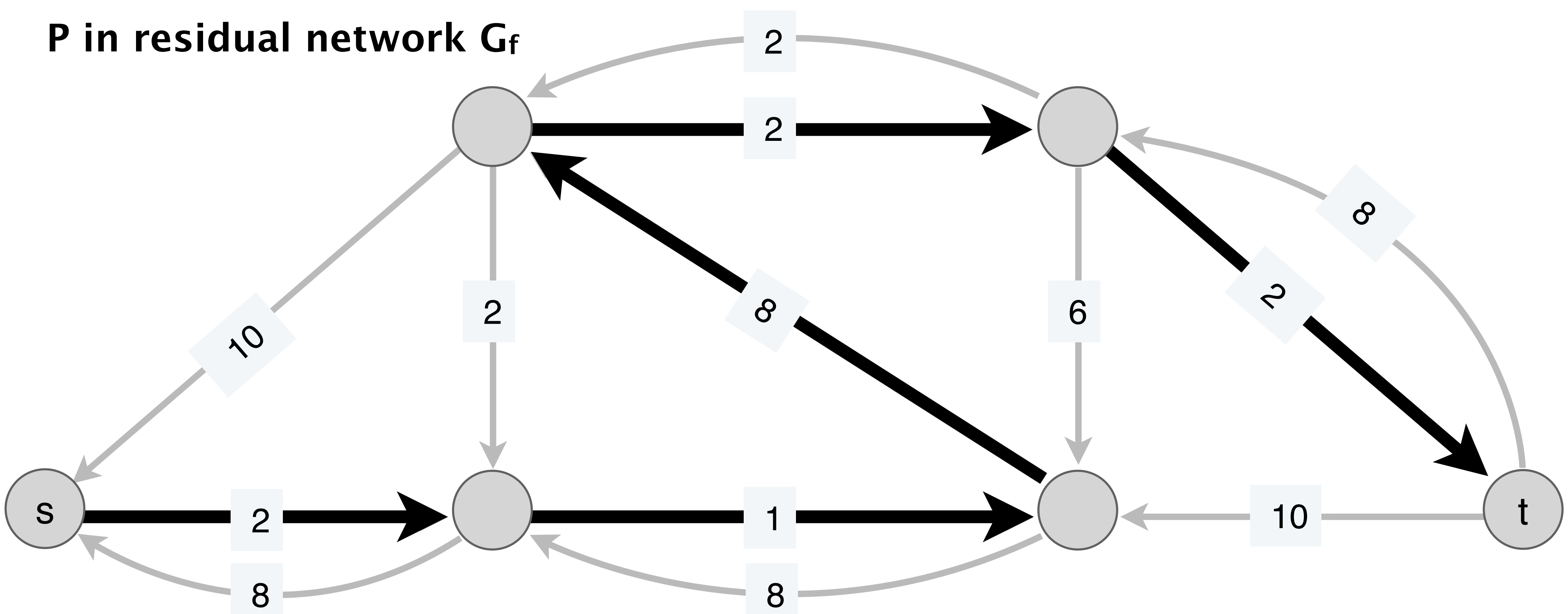
Ford-Fulkerson Example



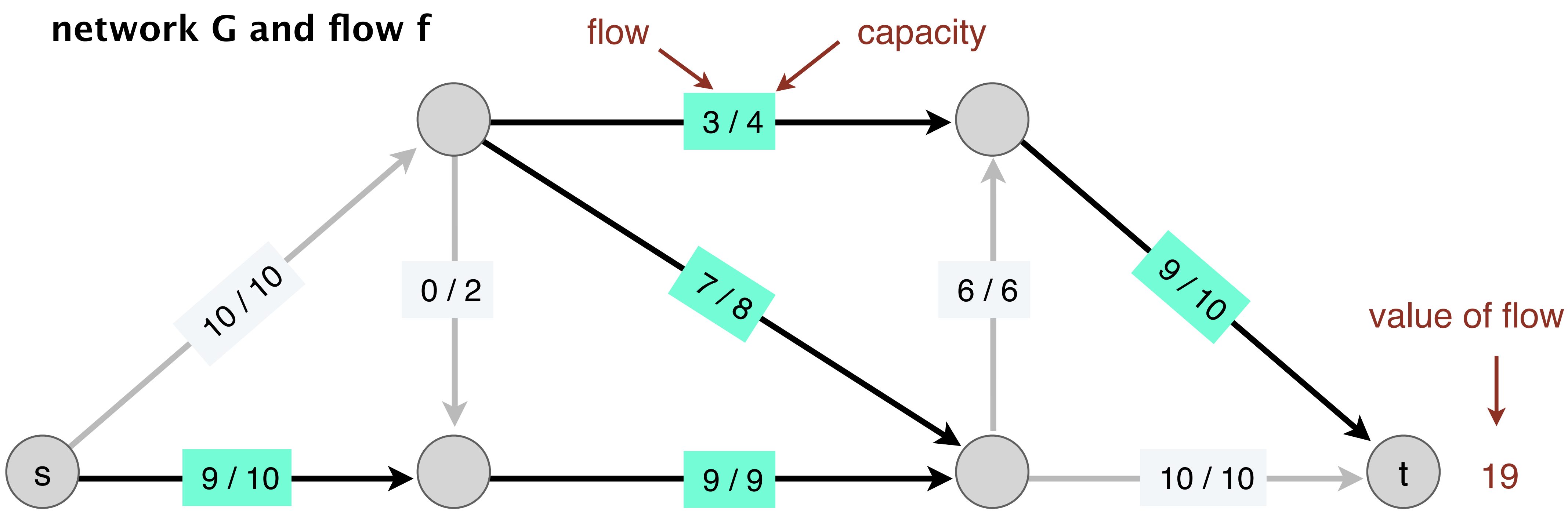


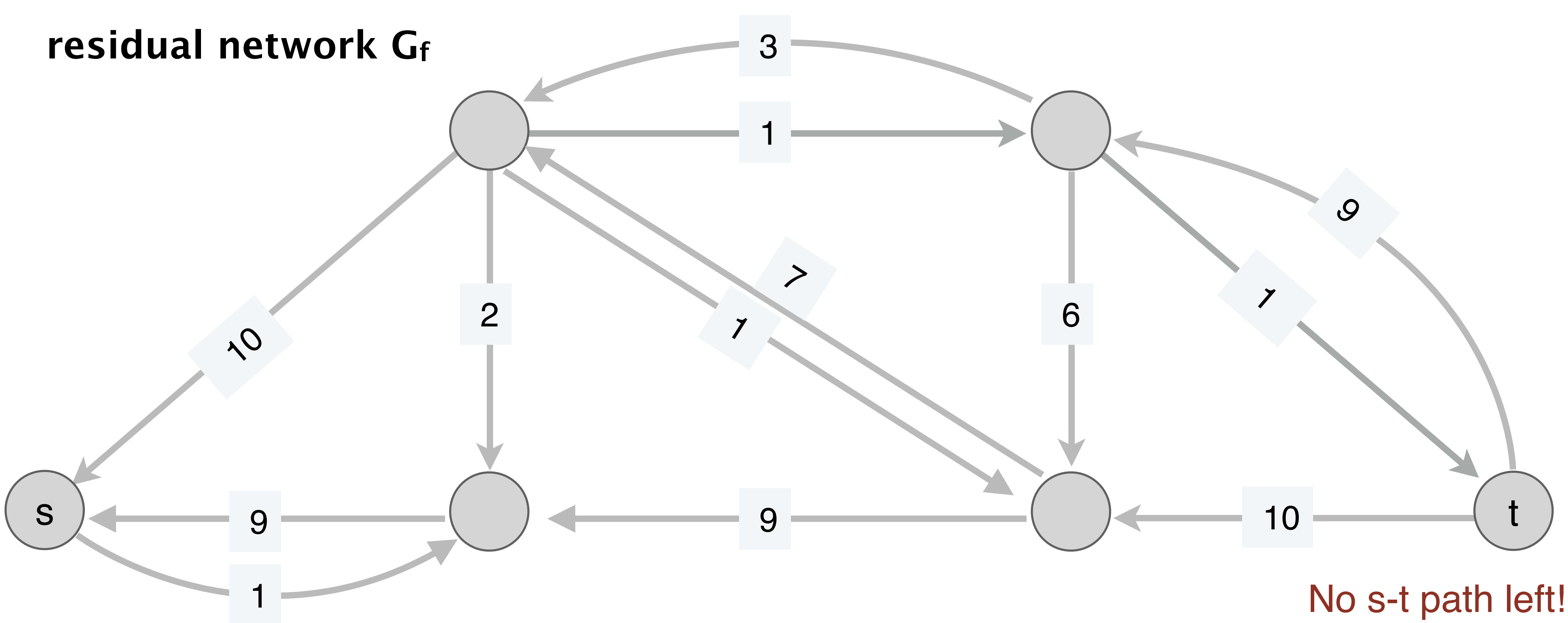
Ford-Fulkerson Example



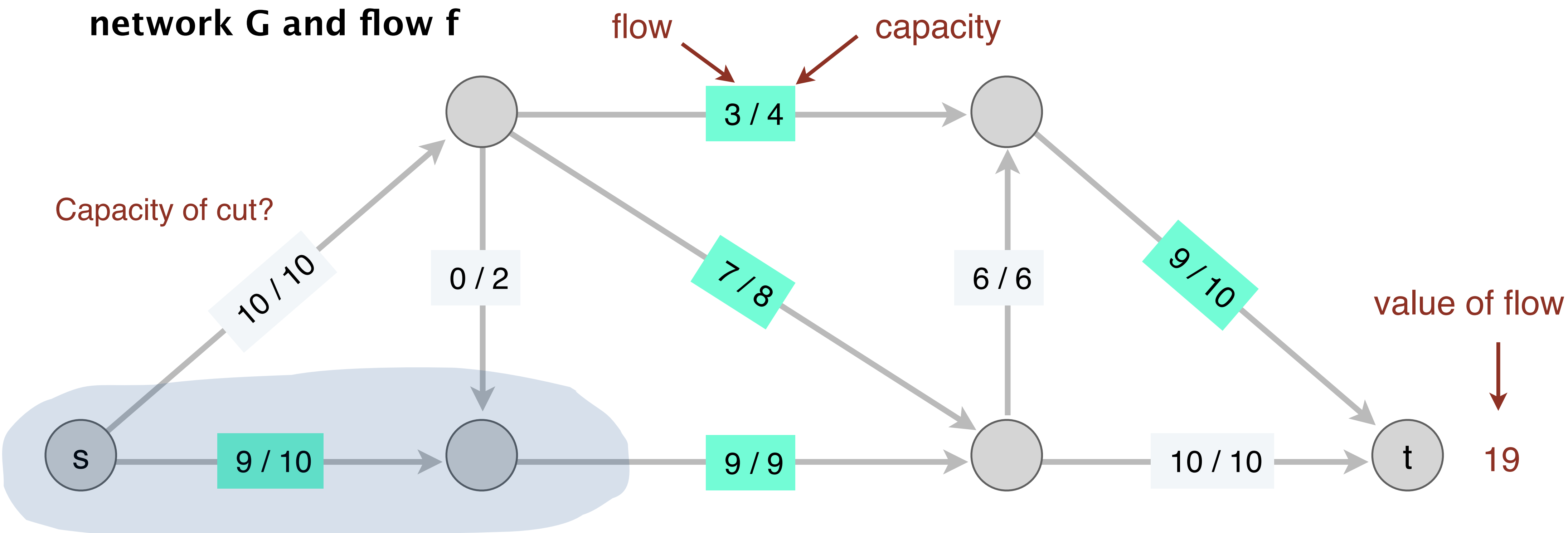


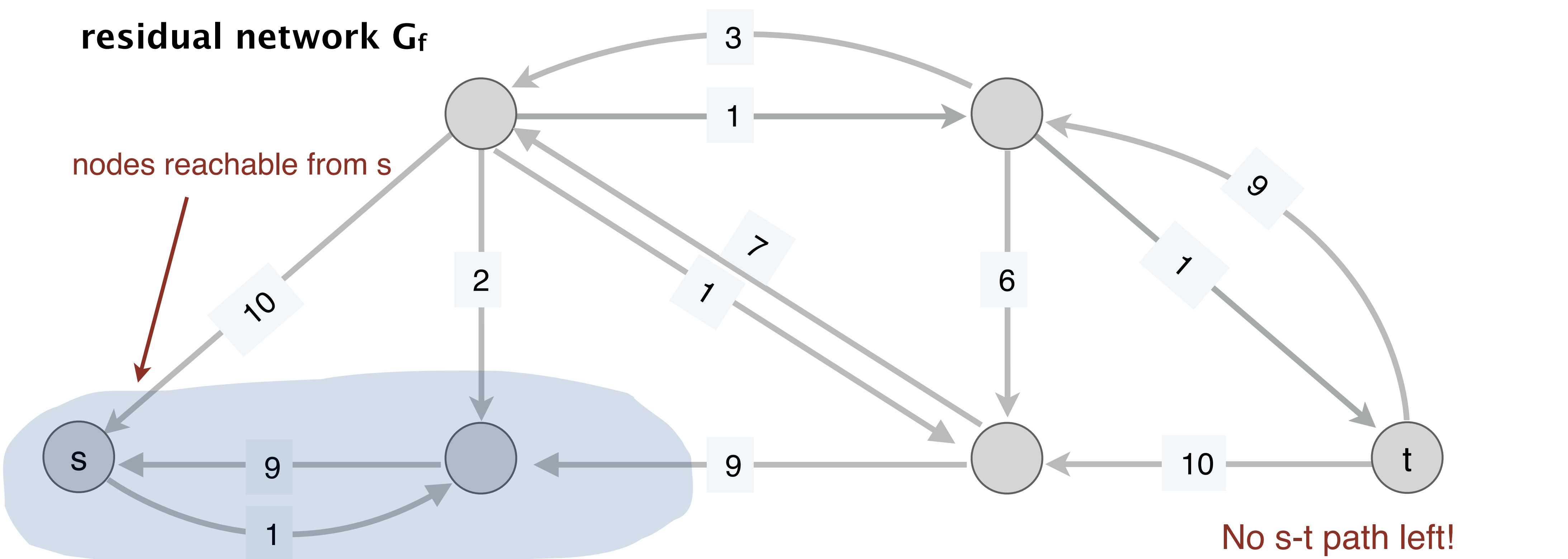
Ford-Fulkerson Example





Ford-Fulkerson Example





Analysis: Ford-Fulkerson

Analysis Outline

- Feasibility and value of flow:
    - Show that each time we update the flow, we are routing a feasible s-t flow through the network
    - And that value of this flow increases each time by that amount
- Optimality:
    - Final value of flow is the maximum possible
- Running time:
    - How long does it take for the algorithm to terminate?
- Space:
    - How much total space are we using?

Feasibility of Flow

- **Claim**. Let f be a feasible flow in G and let P be an augmenting path in G_f with bottleneck capacity b. Let $f' \leftarrow$ AUGMENT(f, P), then f' is **a feasible flow**.
- **Proof**. Only need to verify constraints on the edges of P (since $f' = f$ for other edges). Let $e = (u, v) \in P$
  - If e is a forward edge: $f'(e) = f(e) + b$

    $$\leq f(e) \ + \ (c(e) - f(e)) \ = \ c(e)$$

  - If e is a backward edge: $f'(e) = f(e) - b$

    $$\geq f(e) \ - \ f(e) \ = \ 0$$

- Conservation constraint hold on any node in $u \in P$:
  - $f_{in}(u) = f_{out}(u)$, therefore $f'_{in}(u) = f'_{out}(u)$ for both cases

Value of Flow: Making Progress

- **Claim**. Let f be a feasible flow in G and let P be an augmenting path in G_f with bottleneck capacity b. Let $f' \leftarrow$ AUGMENT(f, P), then $v(f') = v(f) + b$.
- **Proof**.
  - First edge $e \in P$ must be out of s in G_f
  - (P is simple so never visits s again)
  - e must be a forward edge (P is a path from s to t)
  - Thus $f(e)$ increases by b, increasing $v(f)$ by b ∎
- Note. Means the algorithm makes forward progress each time!

Optimality

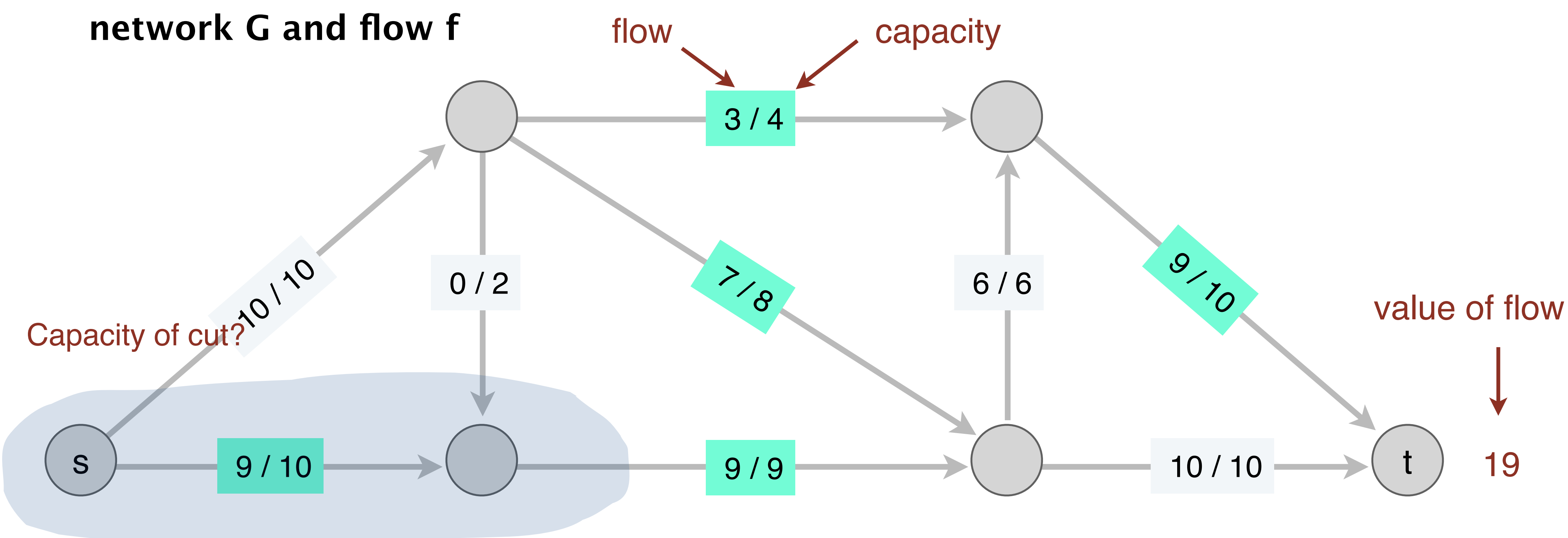
Ford-Fulkerson Optimality

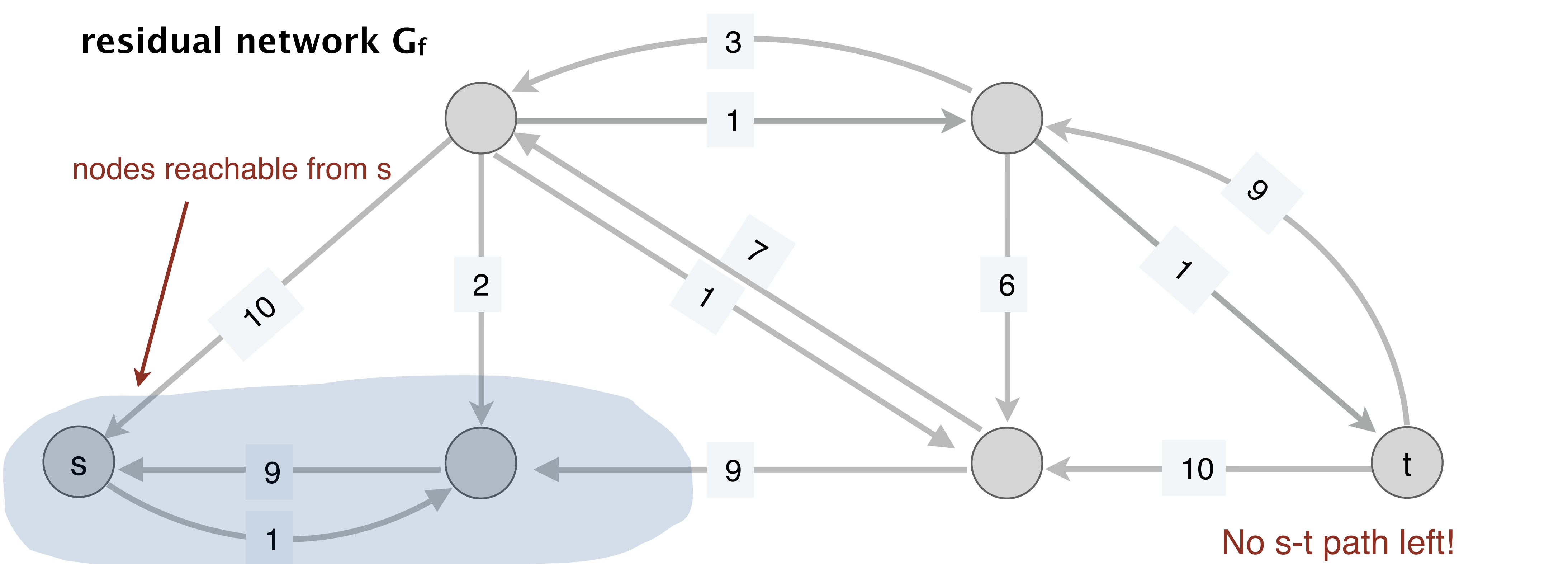
- **Recall**: If f is any feasible s-t flow and (S, T) is any s-t cut then $v(f) \leq c(S, T)$.
- We will show that the Ford-Fulkerson algorithm terminates in a flow that achieves optimality, that is,
- Ford-Fulkerson finds a flow f^* and there exists a cut (S^*, T^*) such that, $v(f^*) = c(S^*, T^*)$
- Proving this shows that it finds the maximum flow (and the min cut)
- This also **proves the max-flow min-cut theorem**

Ford-Fulkerson Optimality

- **Lemma**. Let f be an s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**.
- Let $S^* = \{v \mid v \text{ is reachable from } s \text{ in } G_f\}$, $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = u \to v$ with $u \in S^*, v \in T^*$, then what can we say about $f(e)$?

Recall: Ford-Fulkerson Example





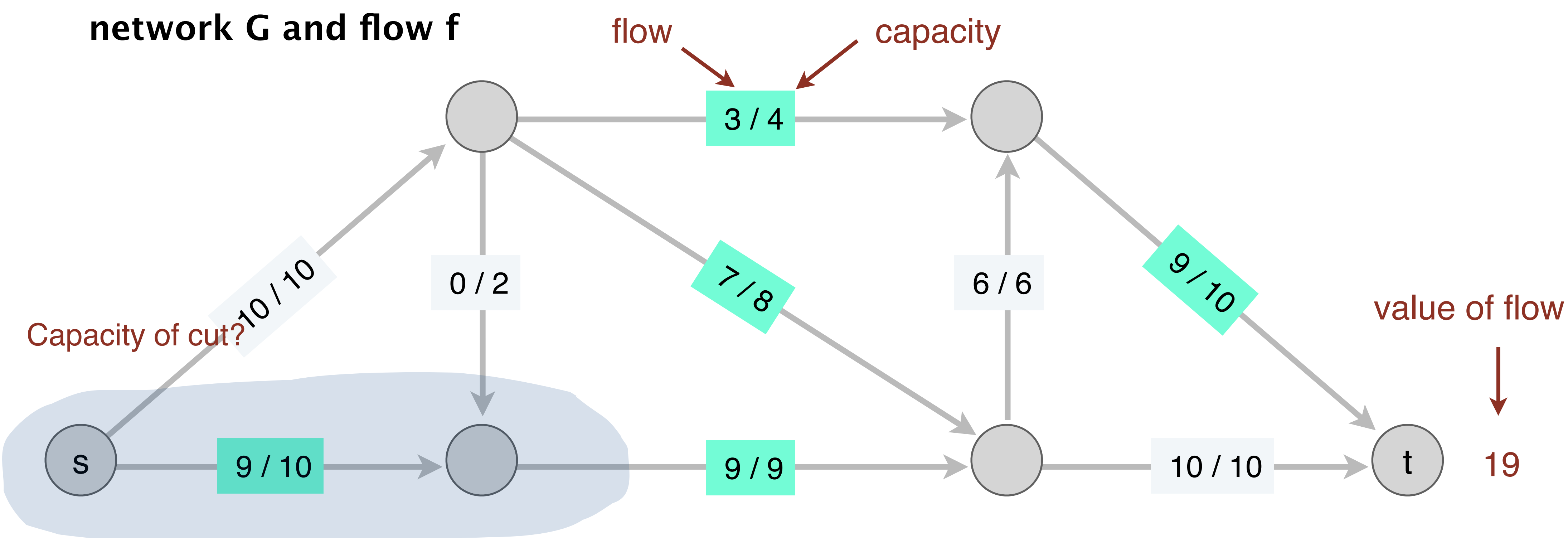
Ford-Fulkerson Optimality

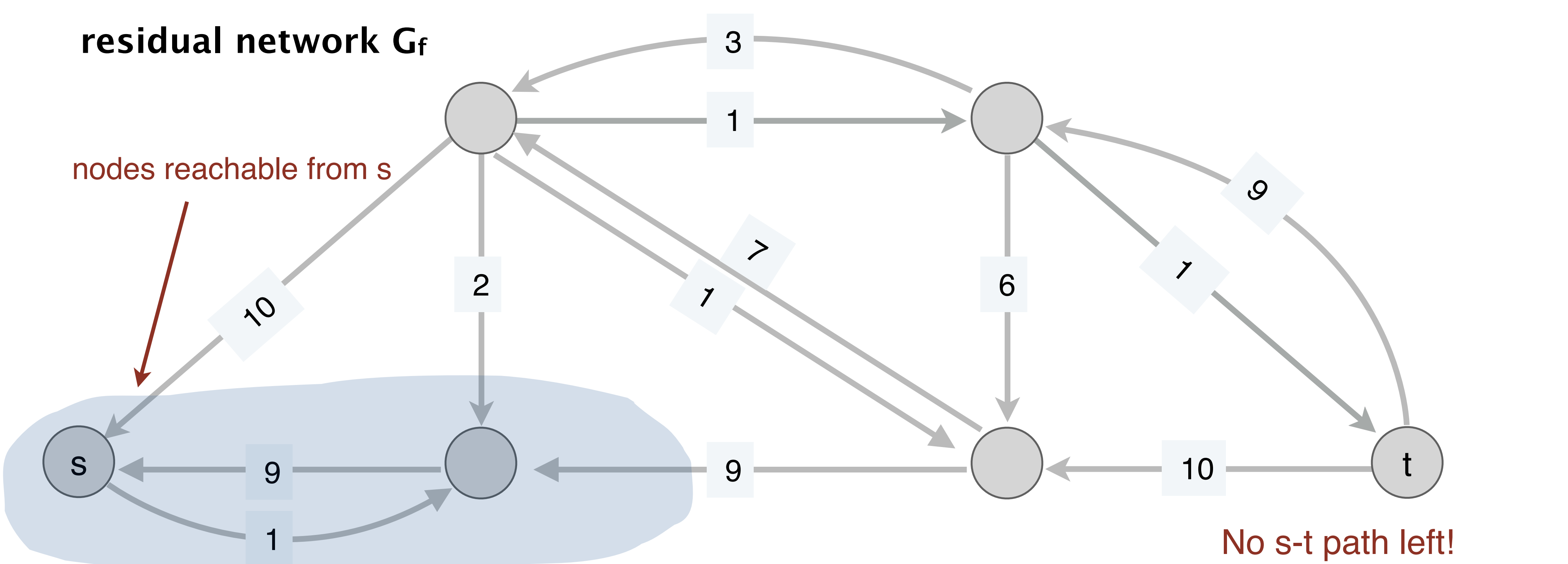
- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**.
- Let $S^* = \{v \mid v \text{ is reachable from } s \text{ in } G_f\}$, $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = u \to v$ with $u \in S^*, v \in T^*$, then what can we say about $f(e)$?
  - $f(e) = c(e)$

Ford-Fulkerson Optimality

- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**. **(Cont.)**
- Let $S^* = \{v \mid v$ is reachable from s in $G_f\}$, $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = w \rightarrow v$ with $v \in S^*, w \in T^*$, then what can we say about $f(e)$?

Recall: Ford-Fulkerson Example





Ford-Fulkerson Optimality

- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**. **(Cont.)**
- Let $S^* = \{v \mid v$ is reachable from s in $G_f\}$, $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \varnothing$
- Consider an edge $e = w \to v$ with $v \in S^*, w \in T^*$, then what can we say about $f(e)$?
  - $f(e) = 0$

Otherwise, there would have been a backwards edge in the residual graph

Ford-Fulkerson Optimality

- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**. **(Cont.)**
- Let $S^* = \{v \mid v$ is reachable from s in $G_f\}$, $T^* = V - S^*$
- Thus, all edges leaving S^* are completely saturated and all edges entering S^* have zero flow
- $v(f) = f_{out}(S^*) - f_{in}(S^*) = f_{out}(S^*) = c(S^*, T^*)$ ∎
- **Corollary**. Ford-Fulkerson returns the maximum flow.

Ford-Fulkerson Algorithm
Running Time

Ford-Fulkerson Performance

FORD–FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

WHILE (there exists an s⇝t path P in G_f)

  $f \leftarrow$ AUGMENT(f, P).

  Update G_f.

RETURN f.

- Does the algorithm terminate?
- Can we bound the number of iterations it does?
- Running time?

Ford-Fulkerson Running Time

- Recall we proved that with each call to AUGMENT, we increase **value of flow** by $b =$ bottleneck(G_f, P)
- **Assumption**. Suppose all capacities $c(e)$ are integers.
- **Integrality invariant.** Throughout Ford–Fulkerson, every edge flow $f(e)$ and corresponding residual capacity is an integer. Thus $b \geq 1$.
- Let $C = \max_u c(s \to u)$ be the maximum capacity among edges leaving the source s.
- It must be that $v(f) \leq (n-1)C$
- Since, $v(f)$ increases by $b \geq 1$ in each iteration, it follows that FF algorithm terminates in at most $v(f) = O(nC)$ iterations.

Ford-Fulkerson Performance

FORD–FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

WHILE (there exists an s⇝t path P in G_f)

  $f \leftarrow$ AUGMENT(f, P).

  Update G_f.

RETURN f.

- Operations in each iteration?
  - Find an augmenting path in G_f
  - Augment flow on path
  - Update G_f

Ford-Fulkerson Running Time

- **Claim.** Ford-Fulkerson can be implemented to run in time $O(nmC)$, where $m = |E| \geq n - 1$ and $C = \max_u c(s \to u)$.
- **Proof**. Time taken by each iteration:
- Finding an augmenting path in G_f
  - G_f has at most $2m$ edges, using BFS/DFS takes $O(m + n) = O(m)$ time
- Augmenting flow in P takes $O(n)$ time
- Given new flow, we can build new residual graph in $O(m)$ time
- Overall, $O(m)$ time per iteration ∎

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - Shikha Singh