Flow Networks:
Max Flow = Min Cut

Admin

- No pending problem set
    - I will hand out an activity that I encourage you use for practice
    - (Most) TA hours will still be held this weekend
        - Ask TAs any questions about the course
        - Ask TAs about the activity (Ford-Fulkerson Algorithm)
- Questions about pre-registration?
- Lab usage?

Relationship between Flows and Cuts

Recall: Cut Capacity

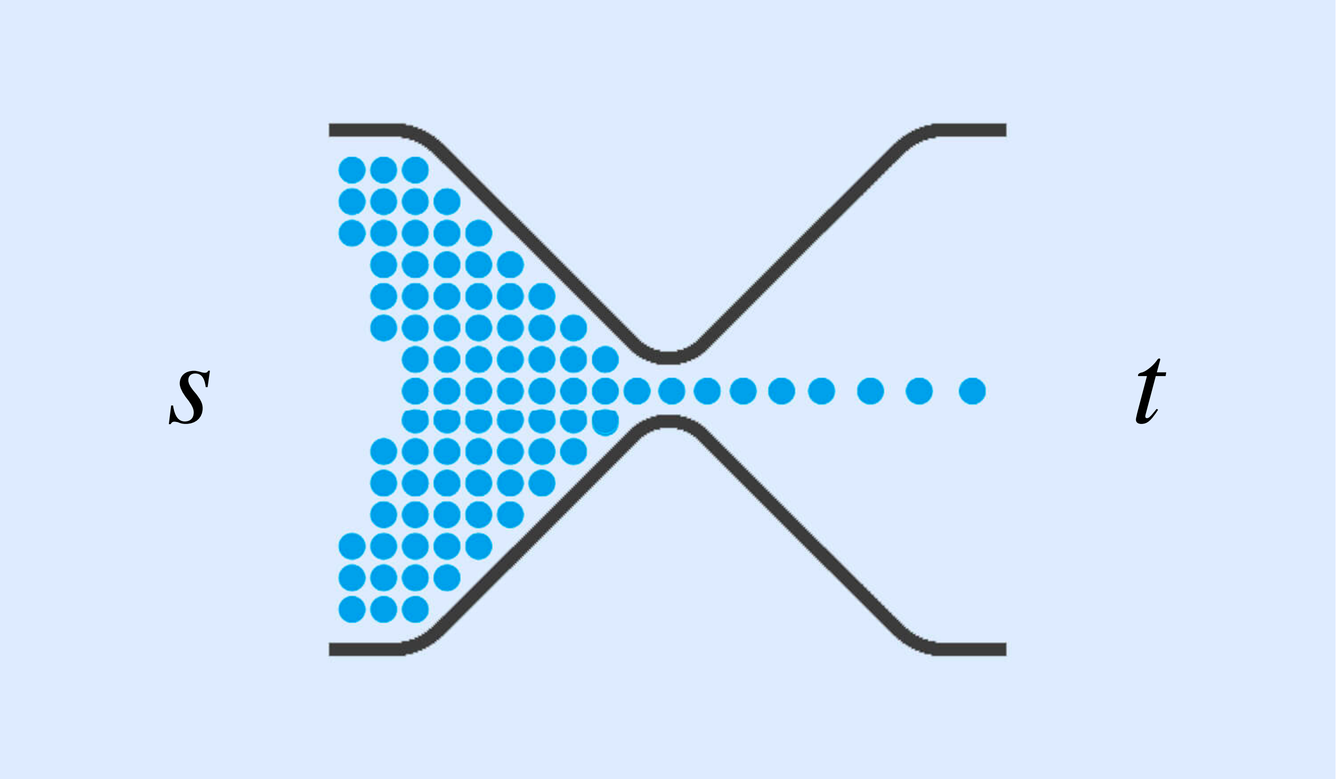
Recall. A cut (S, T) in a graph is a partition of vertices such that $S \cup T = V$, $S \cap T = \emptyset$ and S, T are non-empty.

- **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:

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

Recall: 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

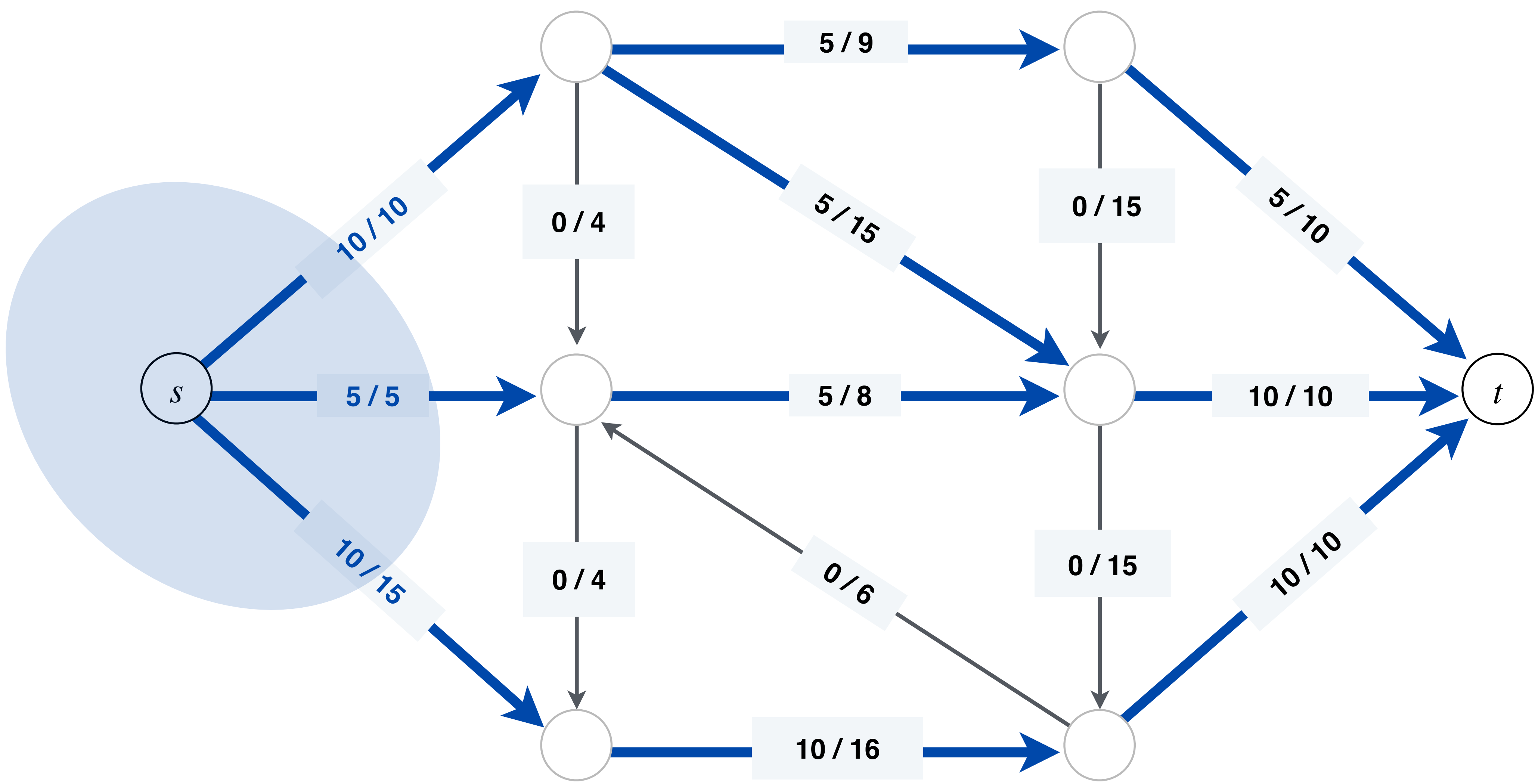


- We will now formalize this intuition

Flows and Cuts

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

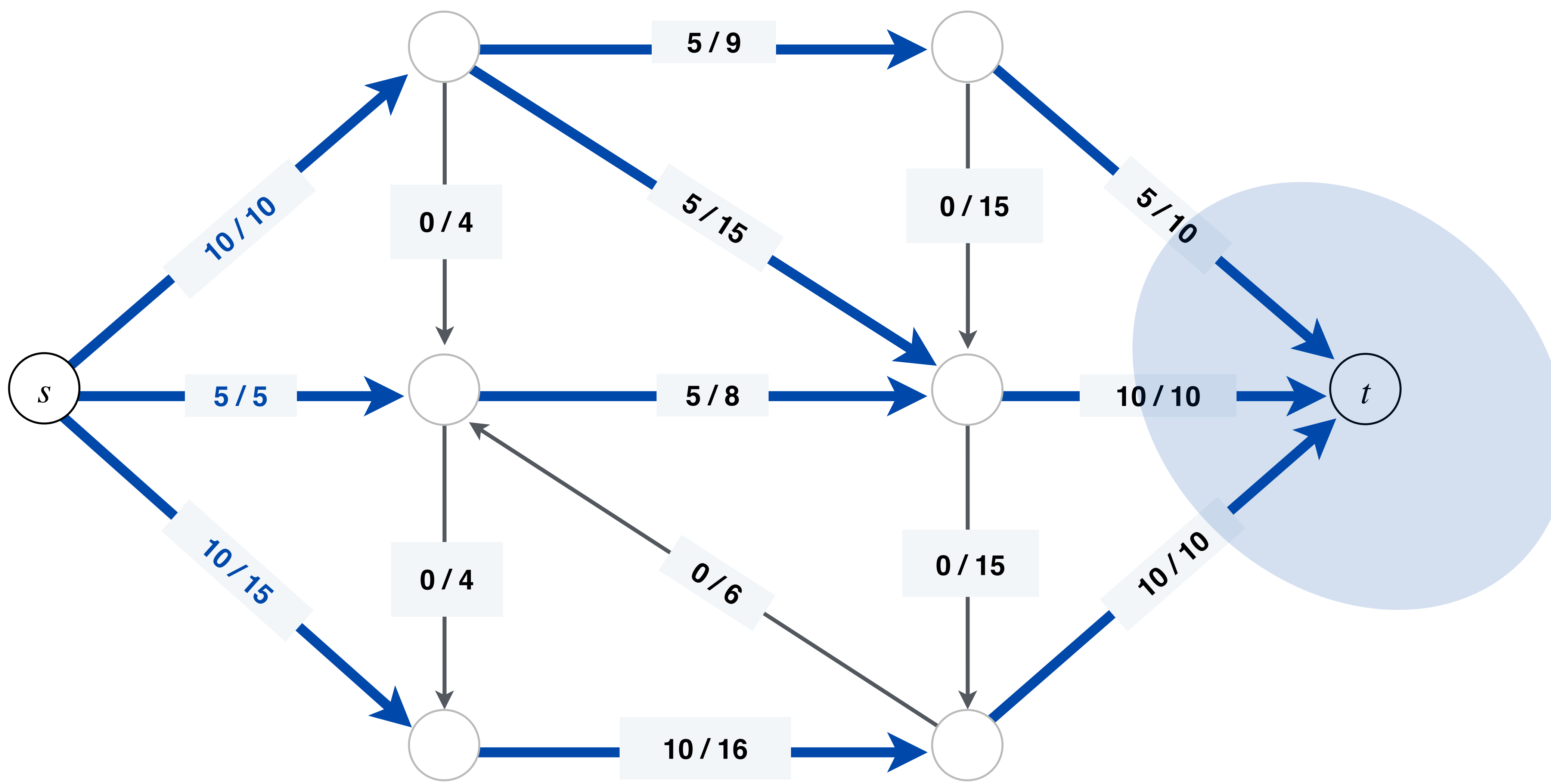
- There are two s-t cuts for which this is easy to see (which ones?)



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

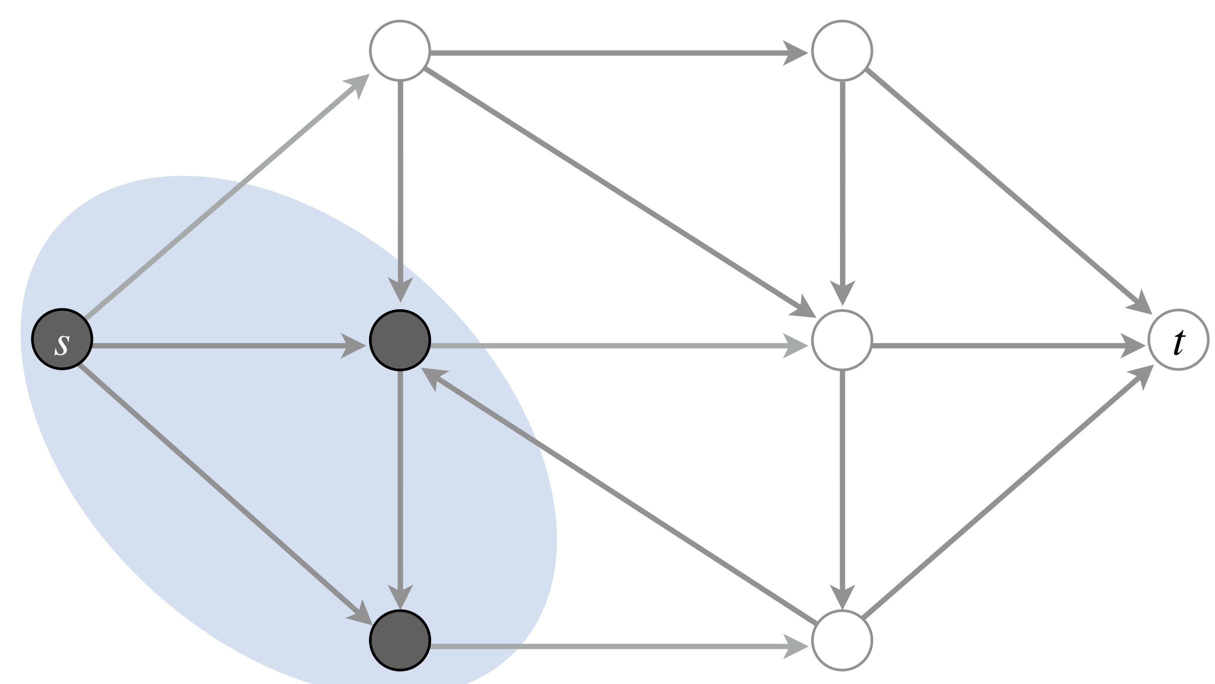
Proof. $f_{out}(S) - f_{in}(S)$

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

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

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

Rearranging terms

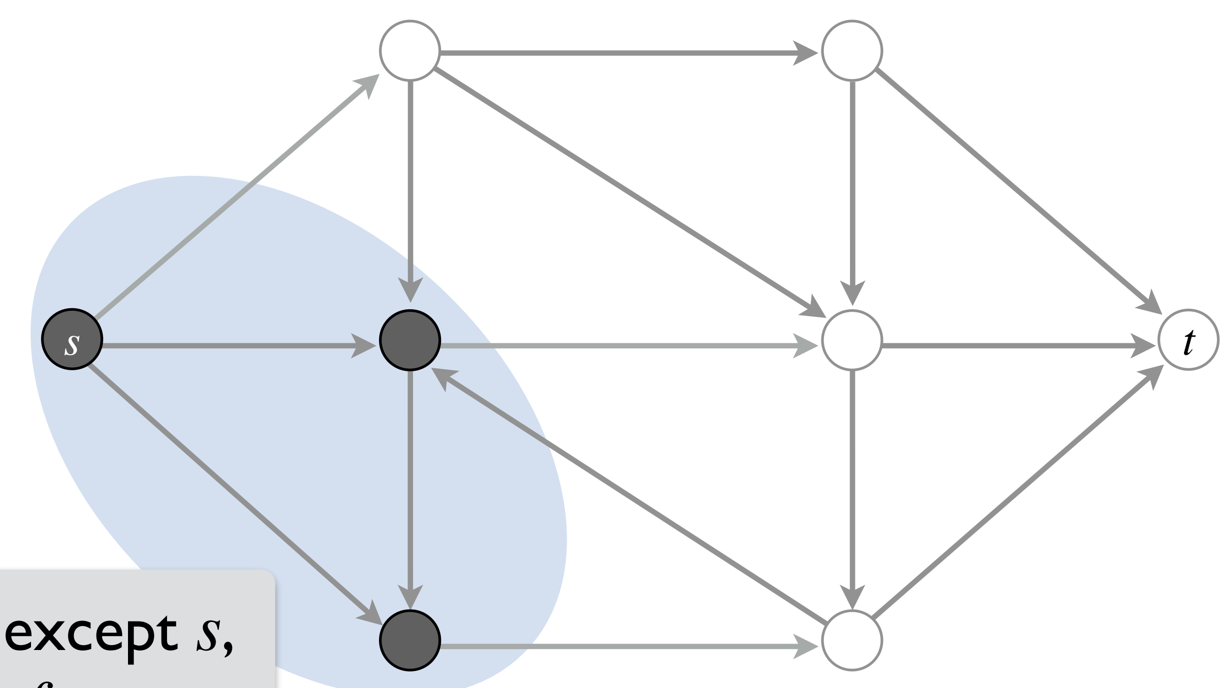
$$= \sum_{v,w \in S} f(v \to w) + \sum_{v \in S, w \in T} f(v \to w) - \sum_{v,u \in S} f(u \to v) - \sum_{v \in S, u \in T} f(u \to v)$$

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

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

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

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

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

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.)

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

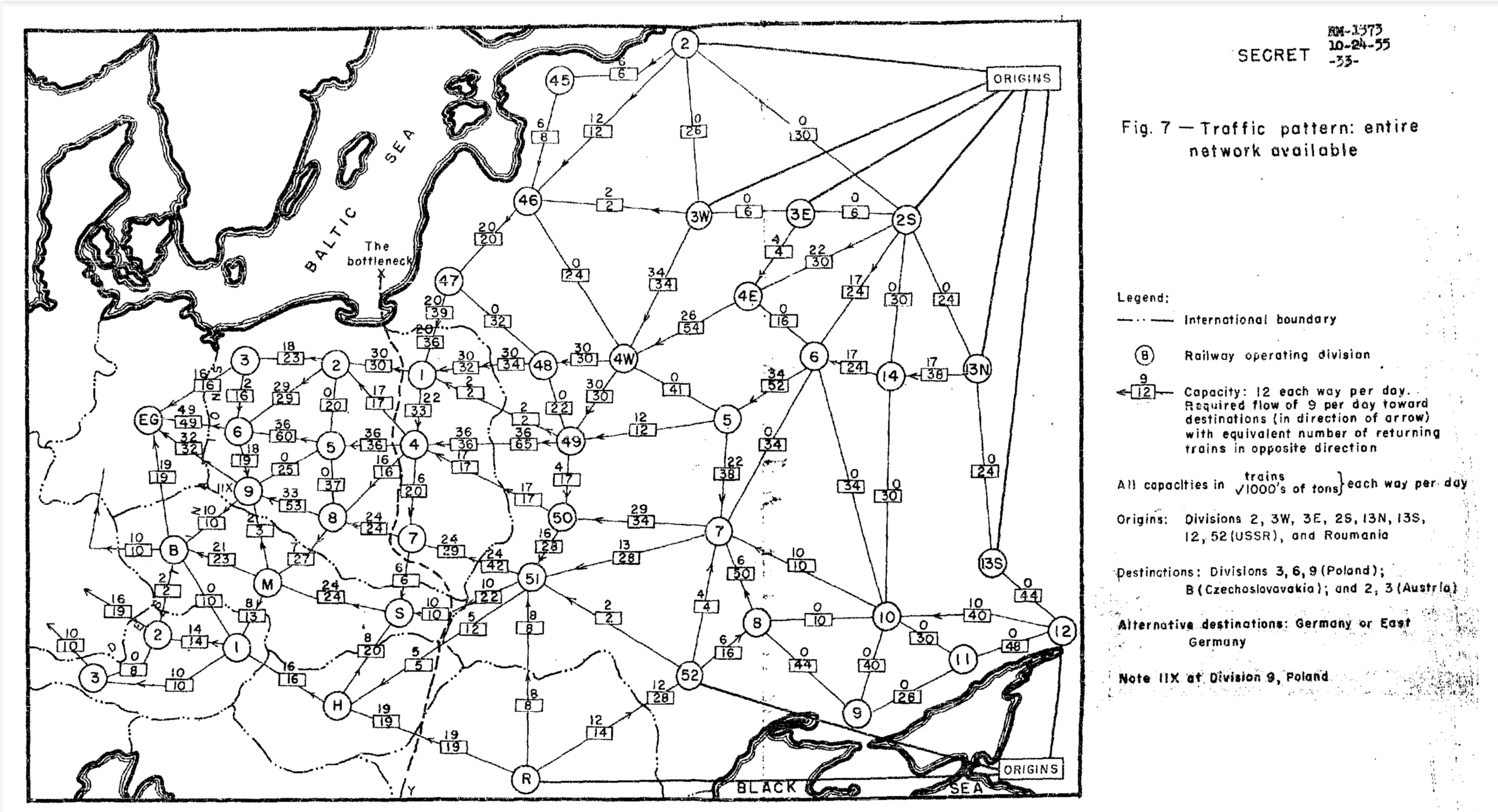


Image Credits: — Jeff Erickson's book and T[homas] E. Harris and F[rank] S. Ross. Fundamentals of a method for evaluating rail net capacities. The RAND Corporation, Research Memorandum RM-1517, October 24, 1955. United States Government work in the public domain. http://www.dtic.mil/dtic/tr/fulltext/u2/093458.pdf

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

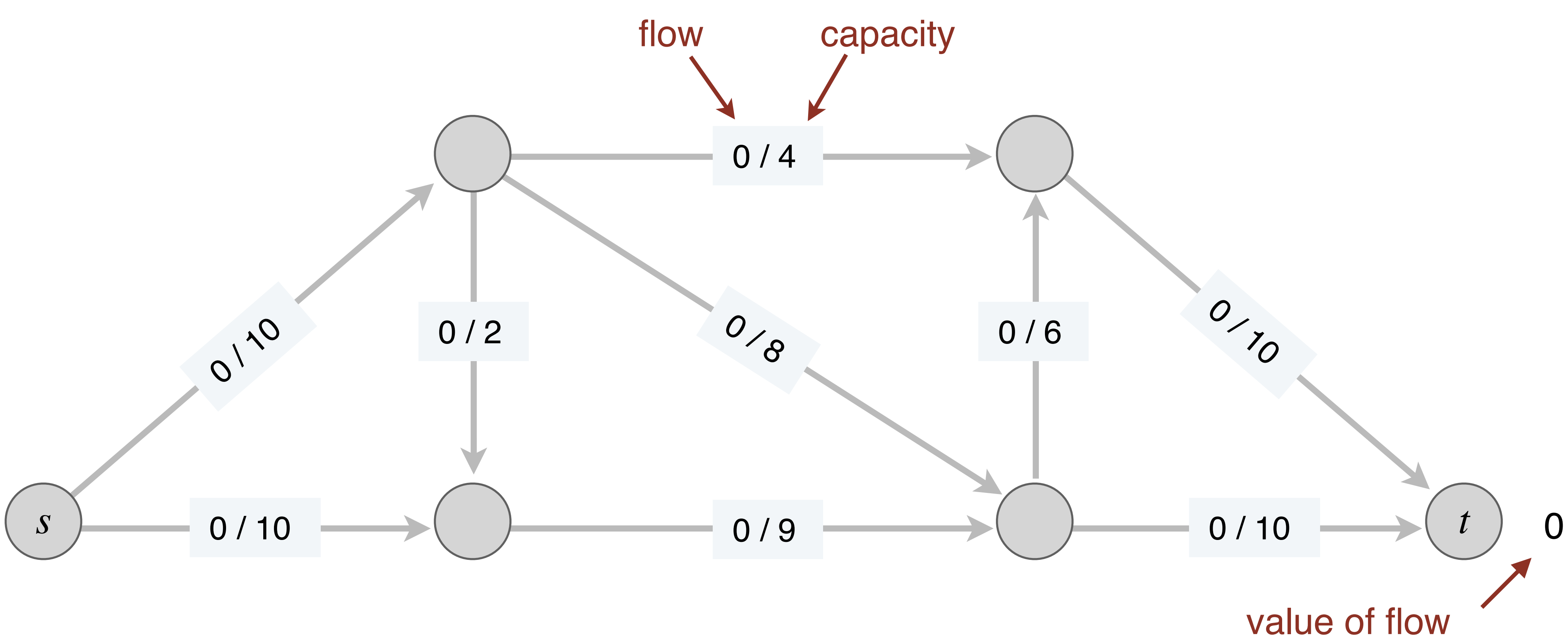
Greedy strategy:

- 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

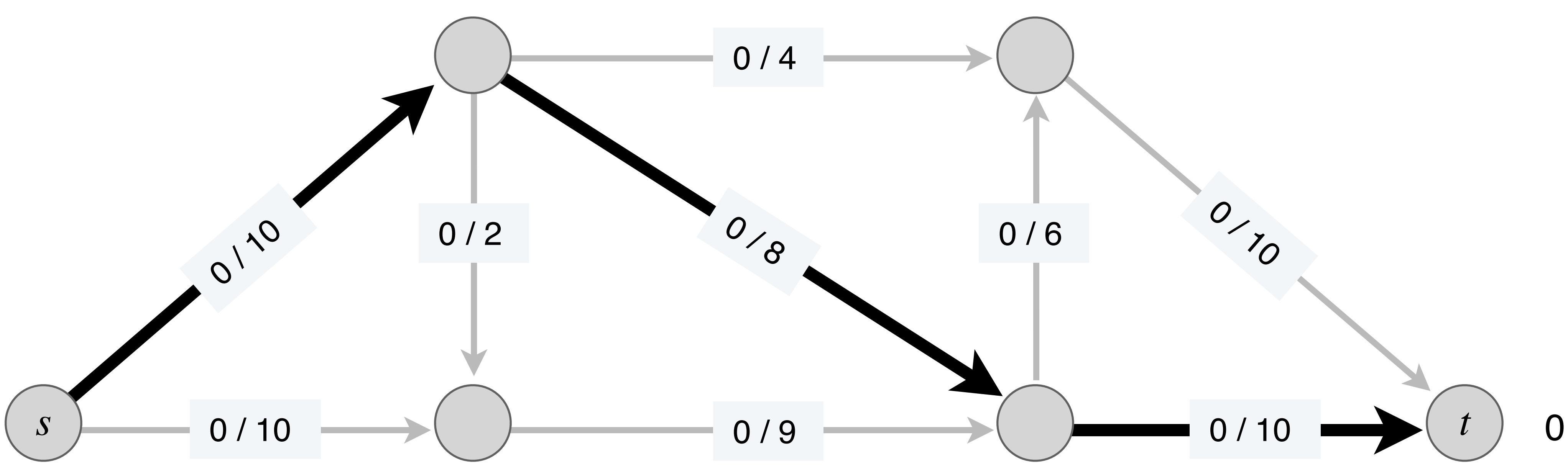
Towards a Max-Flow Algorithm

- 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



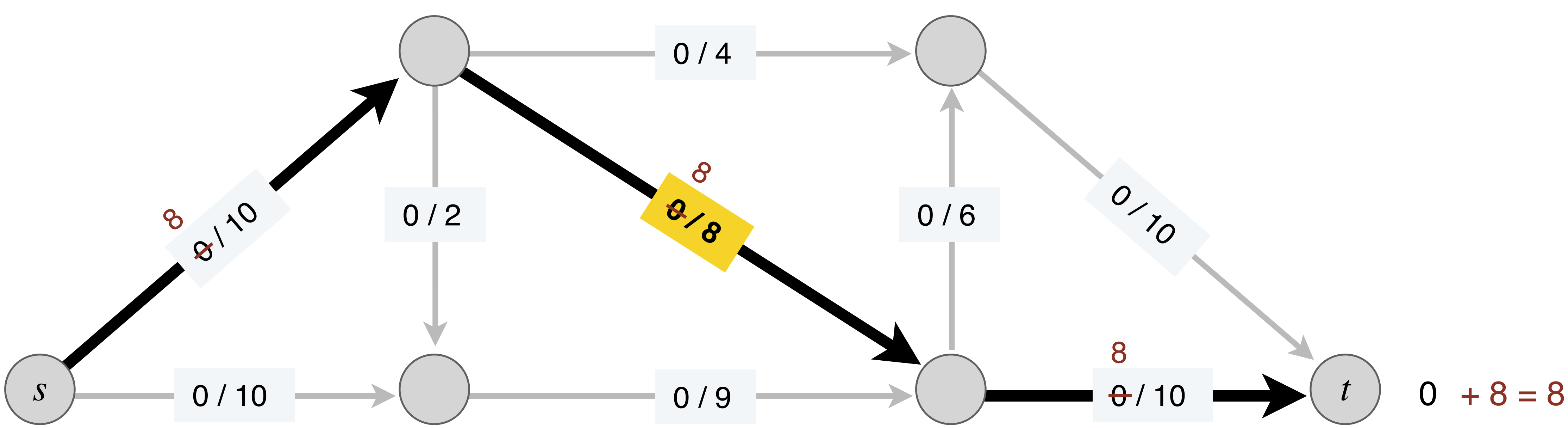
Towards a Max-Flow Algorithm

- 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



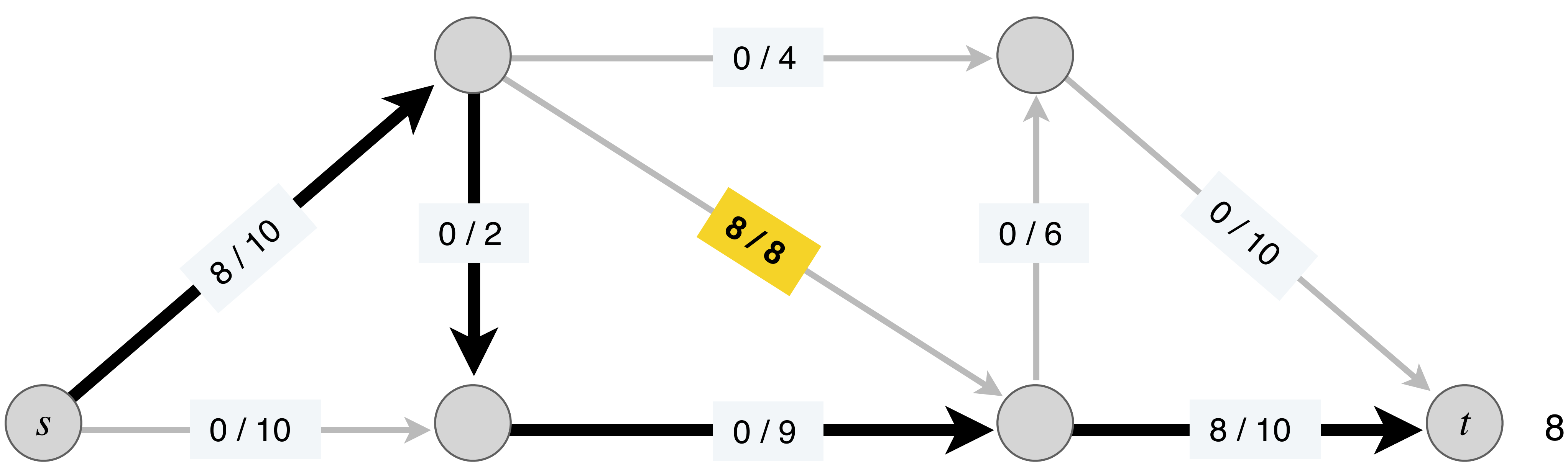
Towards a Max-Flow Algorithm

- 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



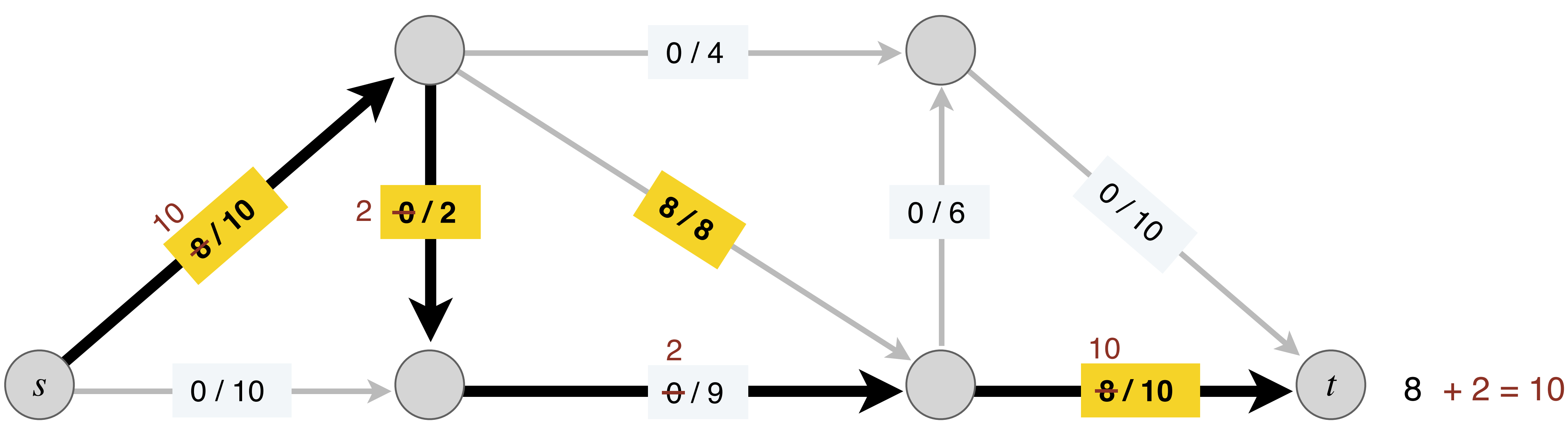
Towards a Max-Flow Algorithm

- 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



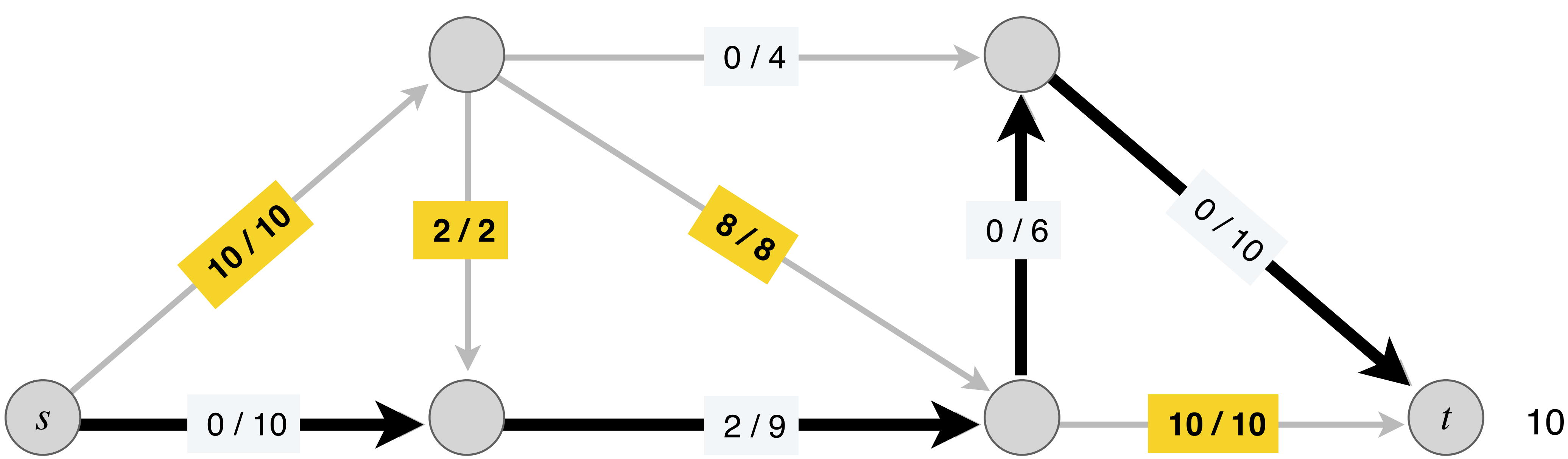
Towards a Max-Flow Algorithm

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

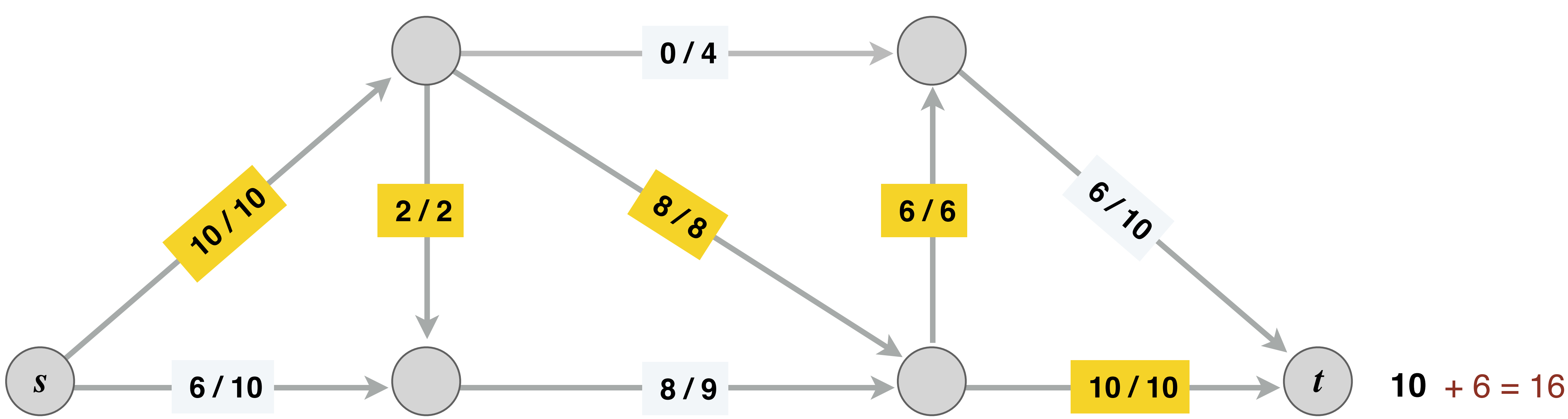


Towards a Max-Flow Algorithm

- 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

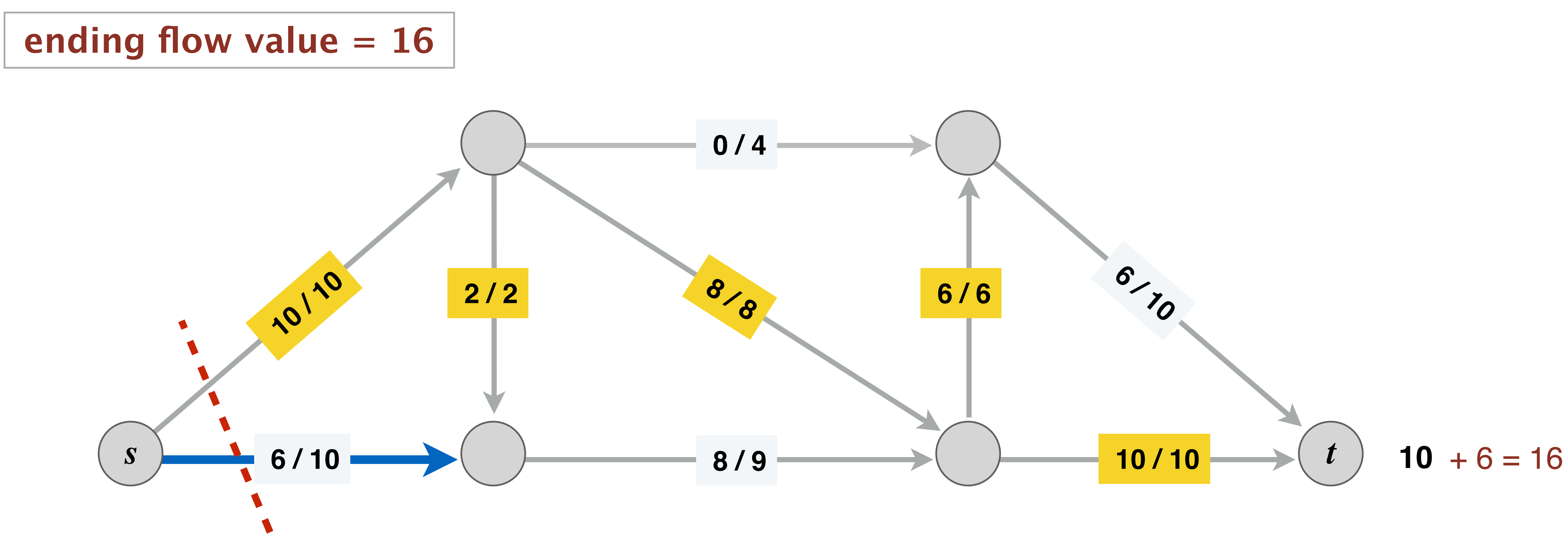
Is this the best we can do?

ending flow value = 16



Towards a Max-Flow Algorithm

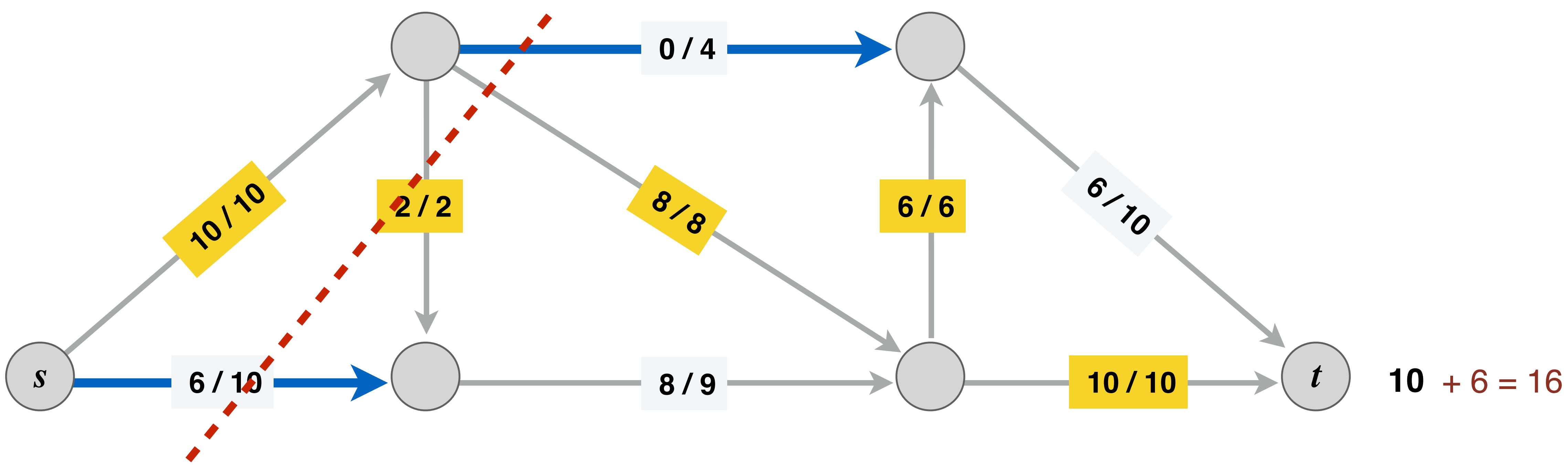
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Towards a Max-Flow Algorithm

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

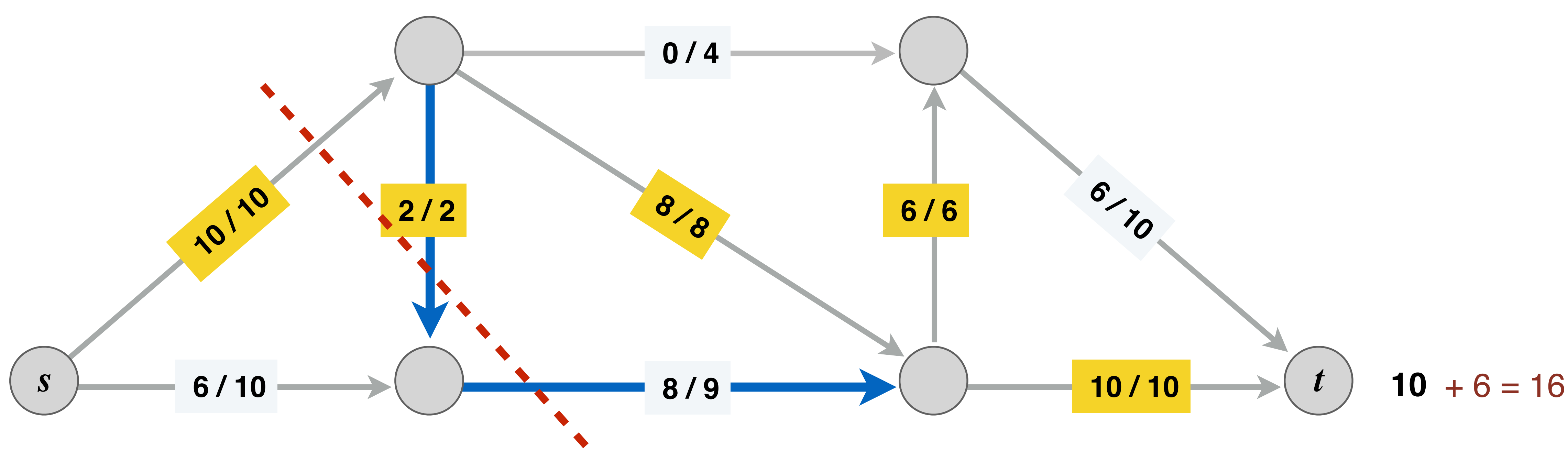
ending flow value = 16



Towards a Max-Flow Algorithm

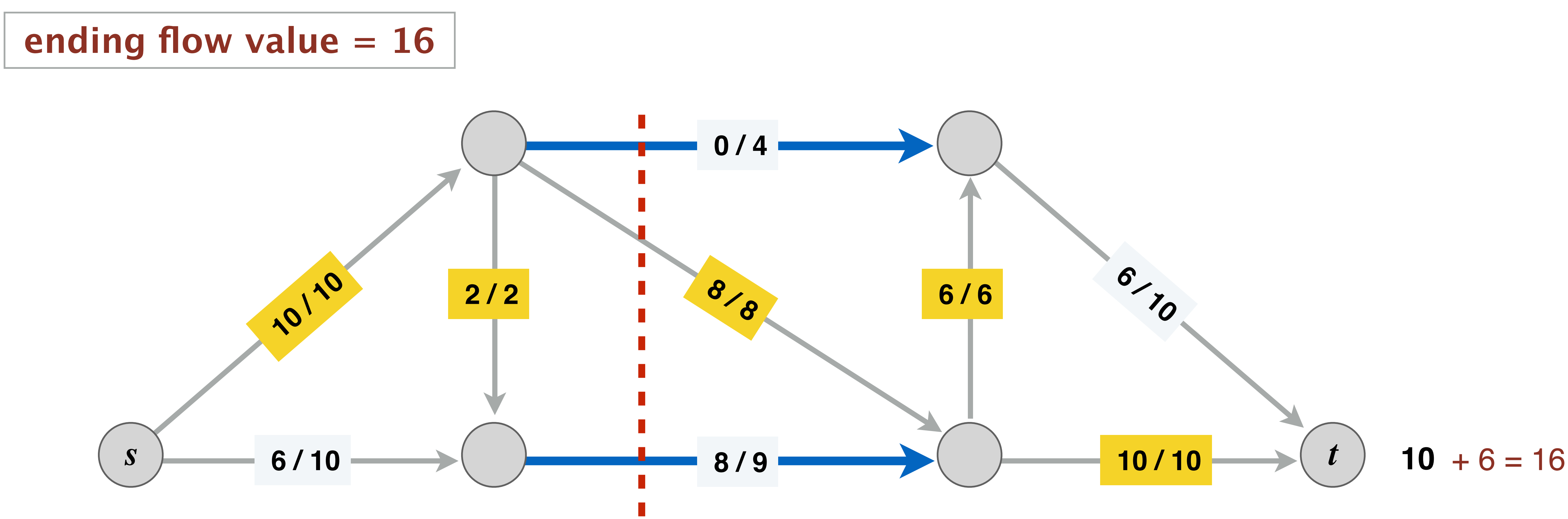
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

ending flow value = 16



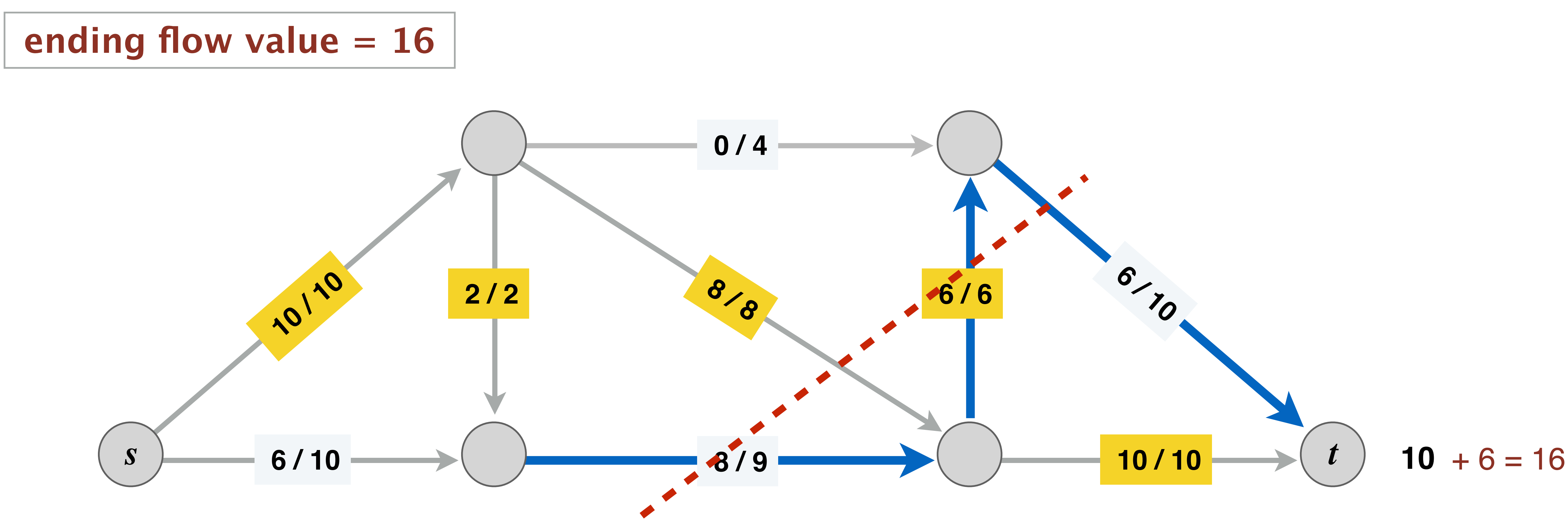
Towards a Max-Flow Algorithm

- 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



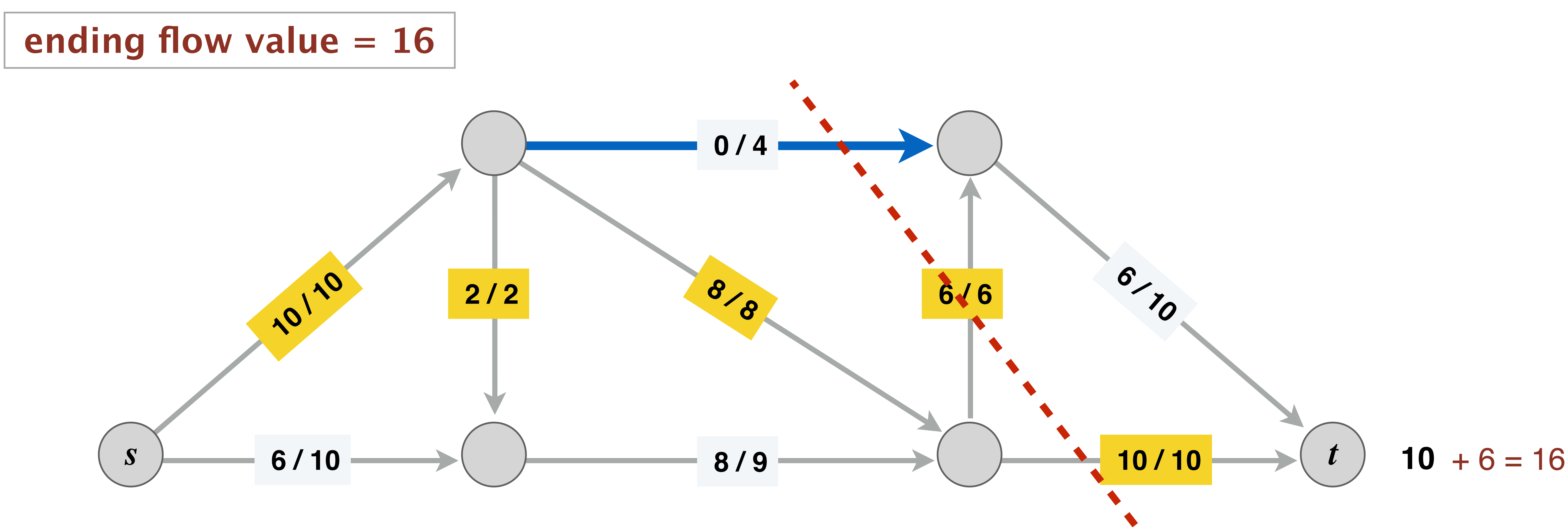
Towards a Max-Flow Algorithm

- 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



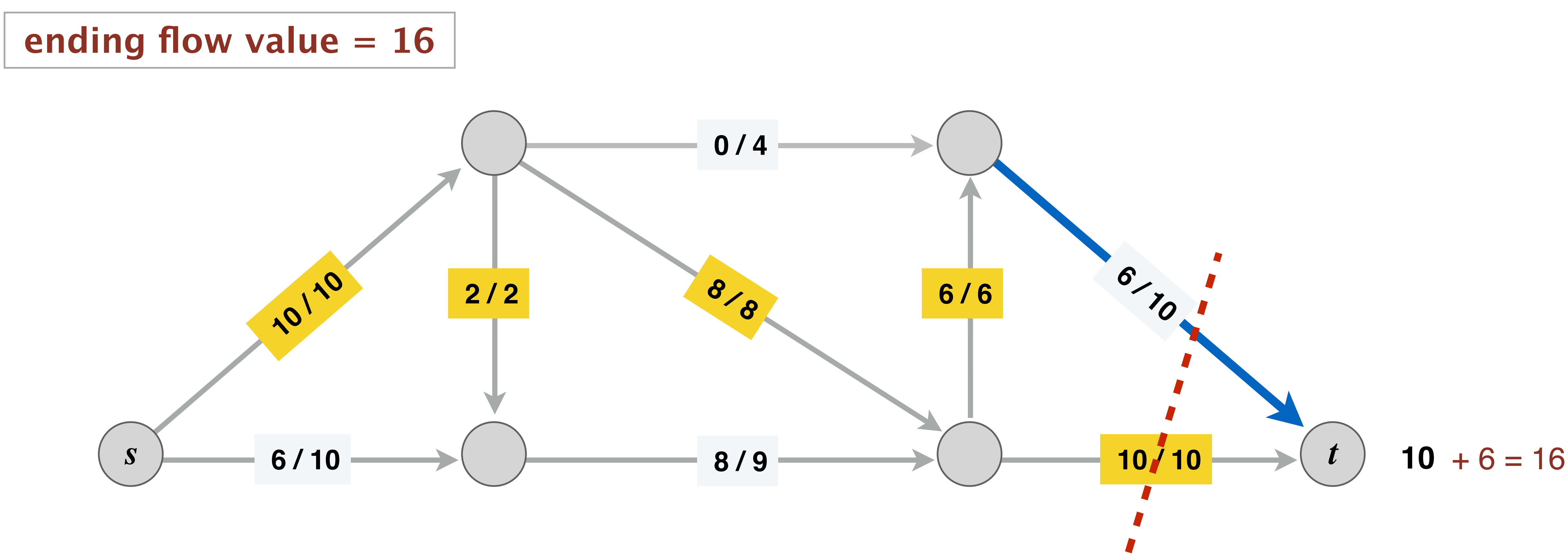
Towards a Max-Flow Algorithm

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



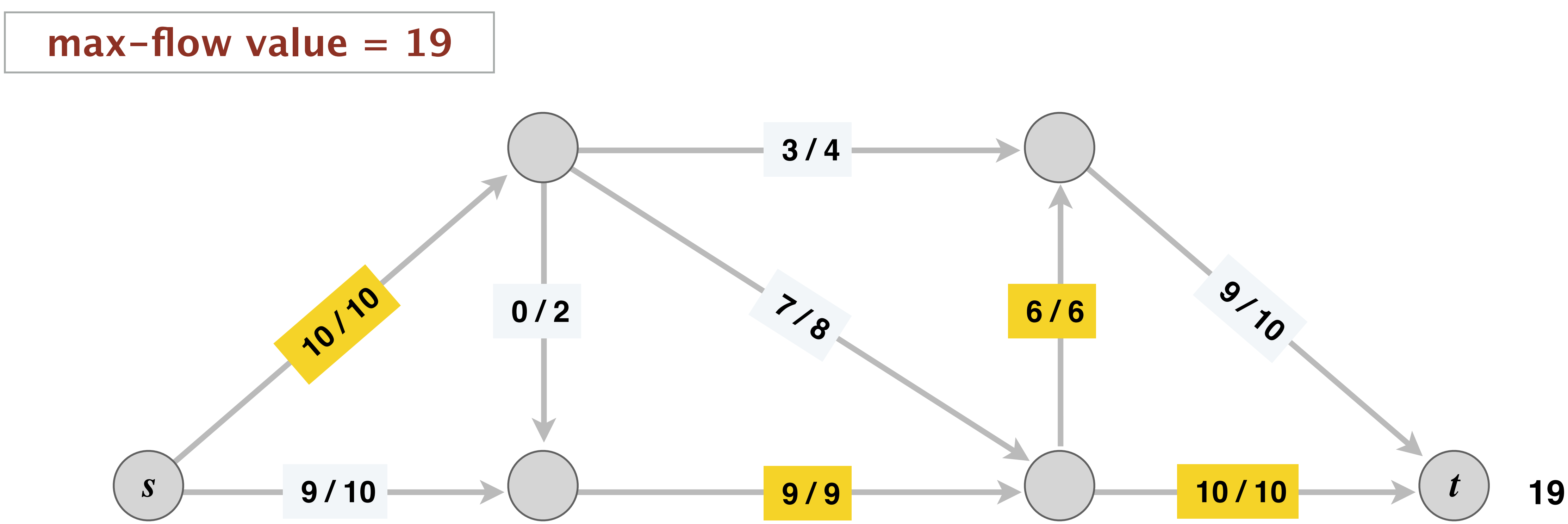
Towards a Max-Flow Algorithm

- 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



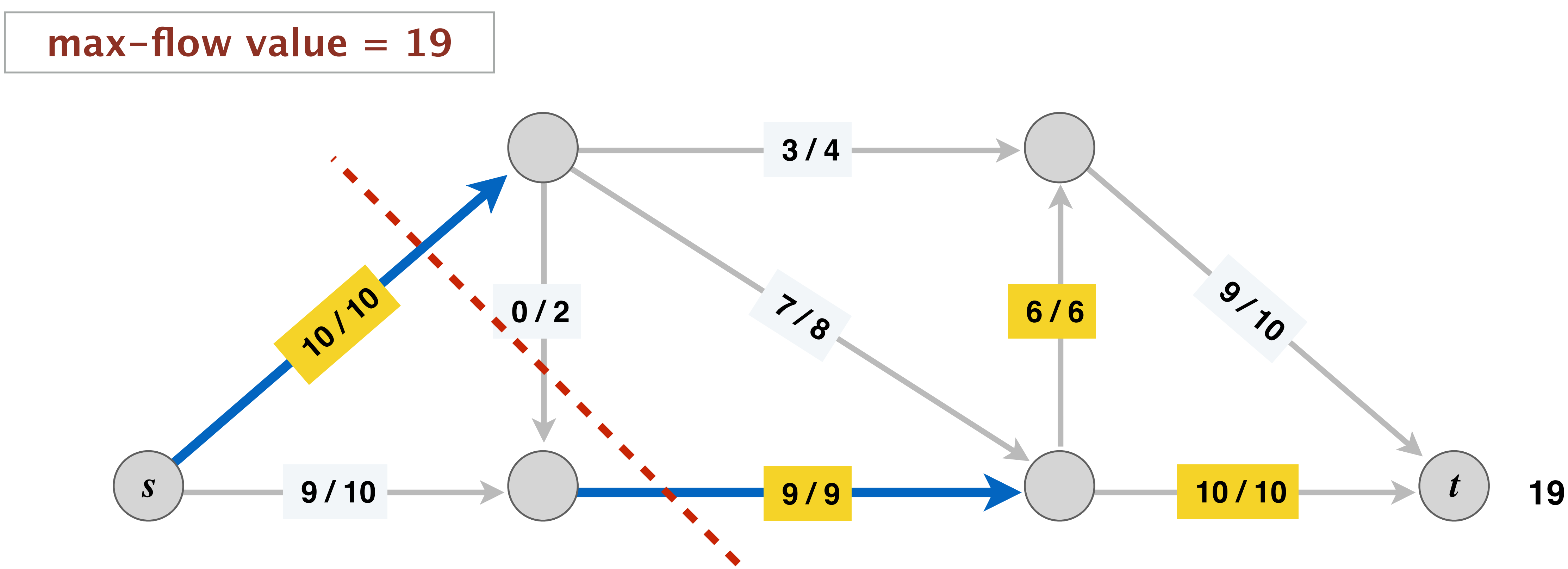
Towards a Max-Flow Algorithm

- 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

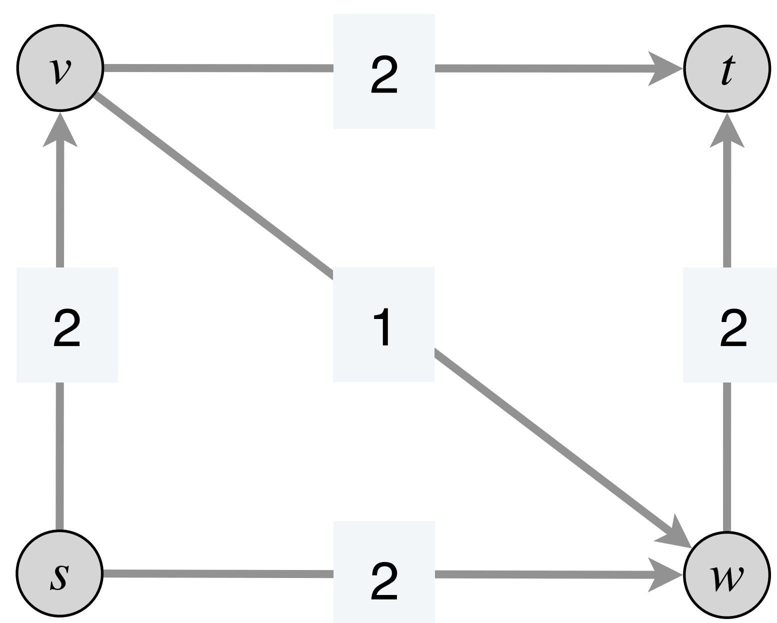
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Why Greedy Fails

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

- Consider the following flow network



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

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

Ford-Fulkerson Algorithm

Ford Fulkerson: Idea

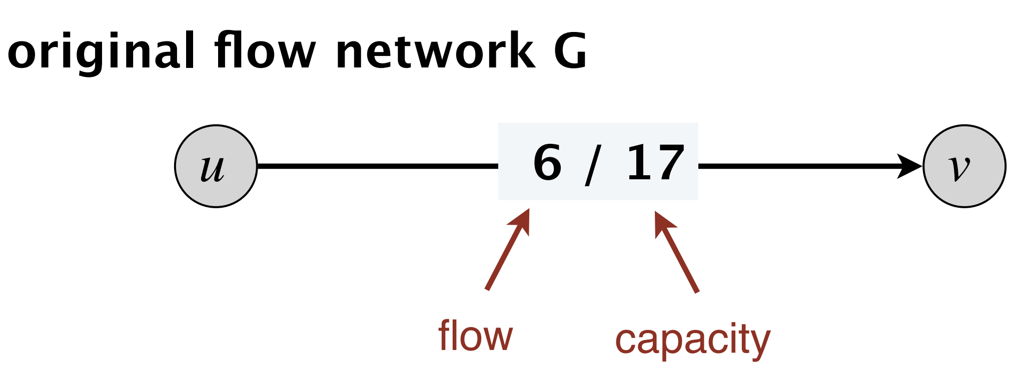
Goal: Want to make "forward progress" while letting ourselves undo previous decisions if they're getting in our way

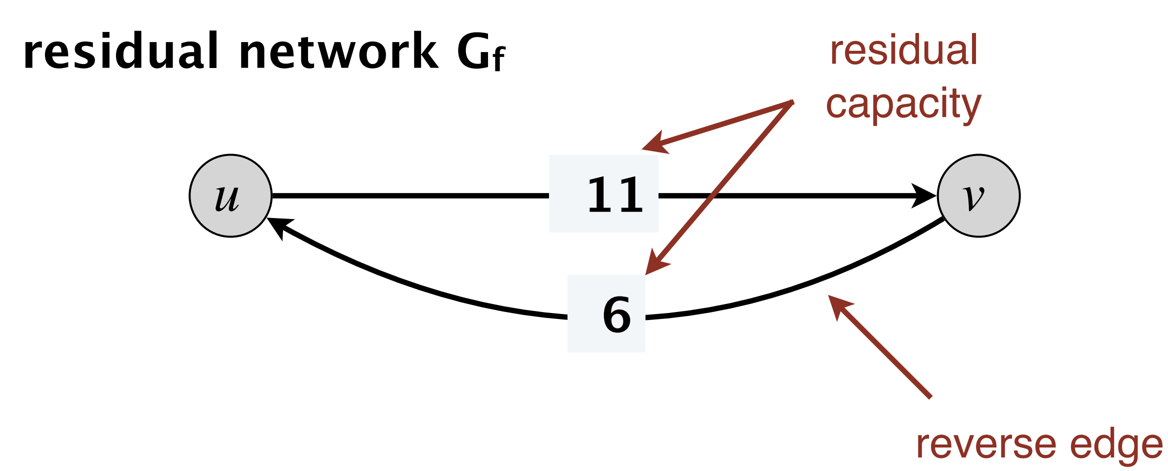
- **Idea**: keep track of where we can push flow
  - Can push more flow along an edge with remaining capacity
  - Can also push flow "back" along an edge that already has flow down it (undo a previous flow push)
- Need a way to systematically track these decisions

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$\rightsquigarrow$t path P in G_f)

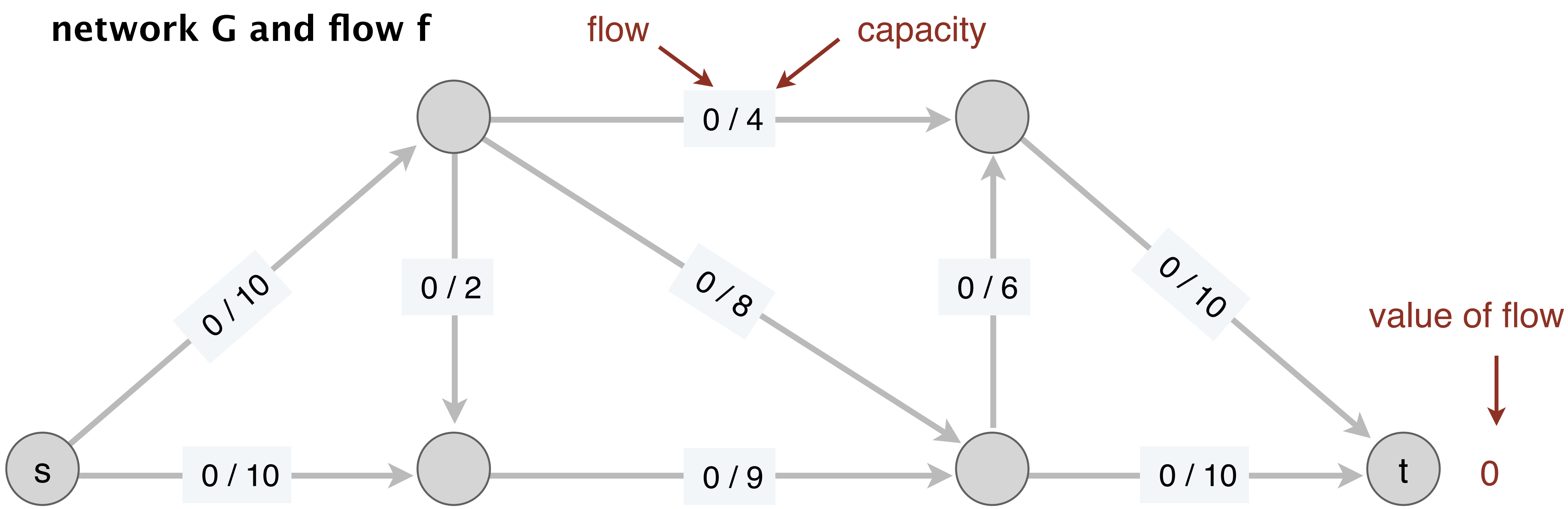
    $f \leftarrow$ AUGMENT(f, P).

    Update G_f.

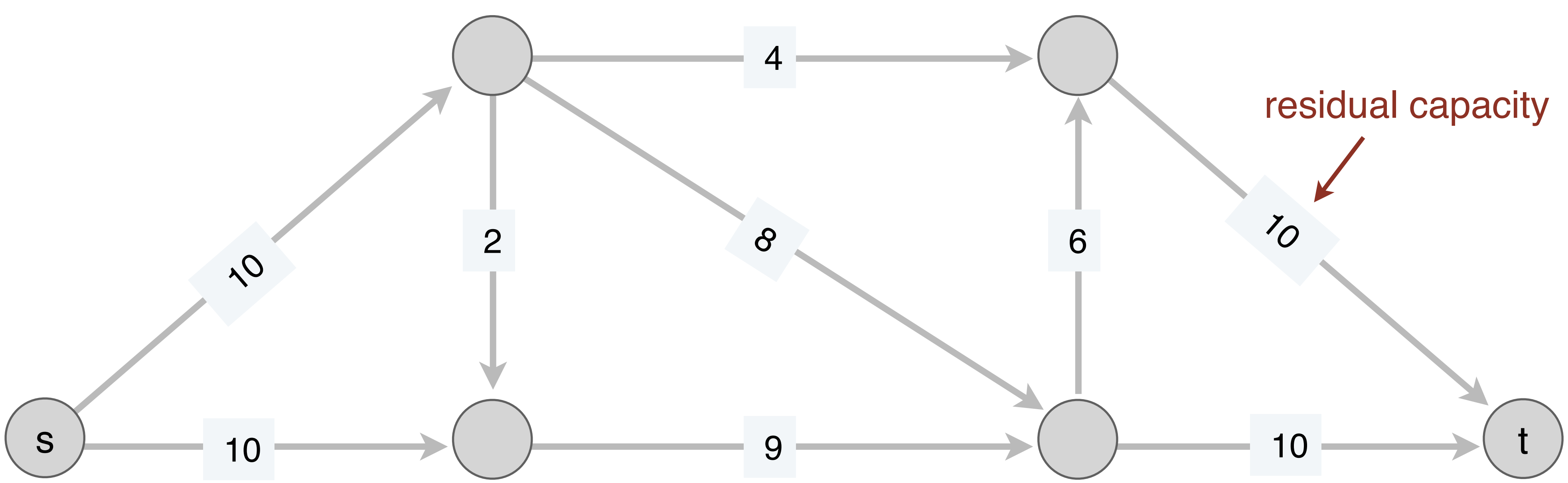
RETURN f.

Ford-Fulkerson Example

network G and flow f

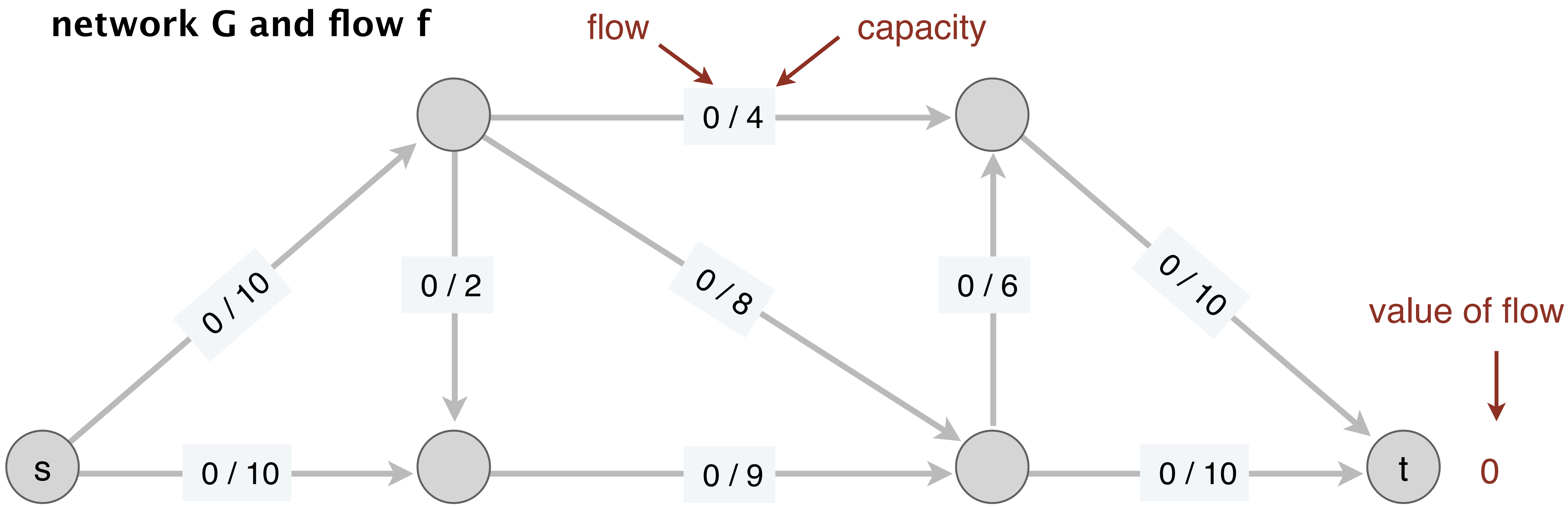


residual network G_f

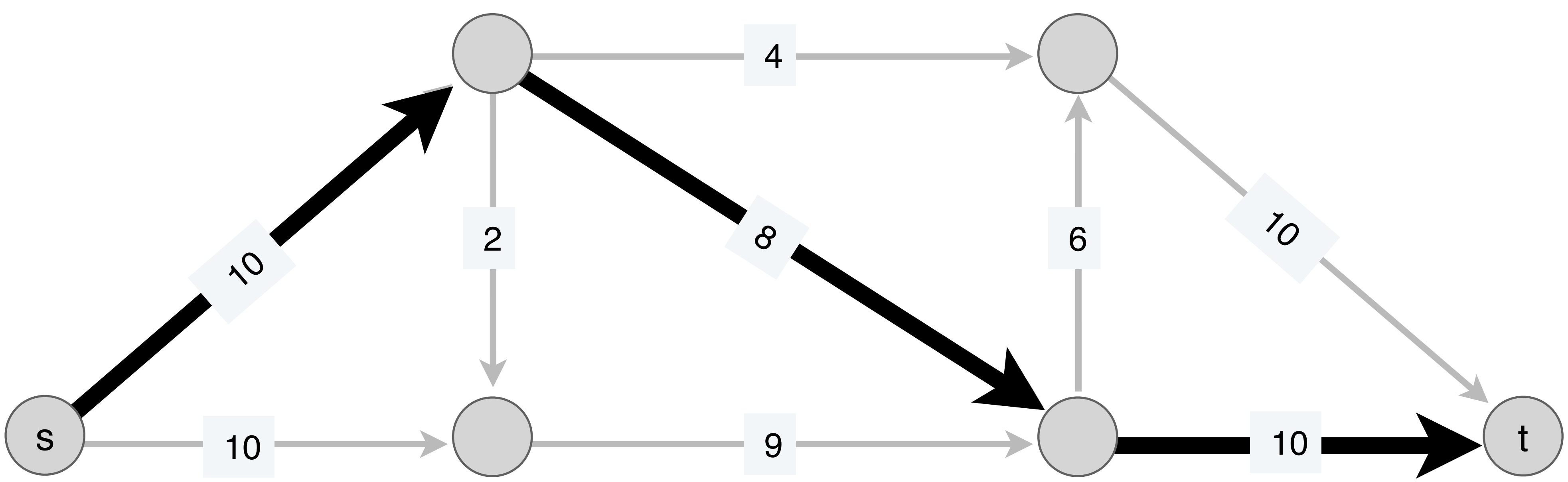


Ford-Fulkerson Example

network G and flow f

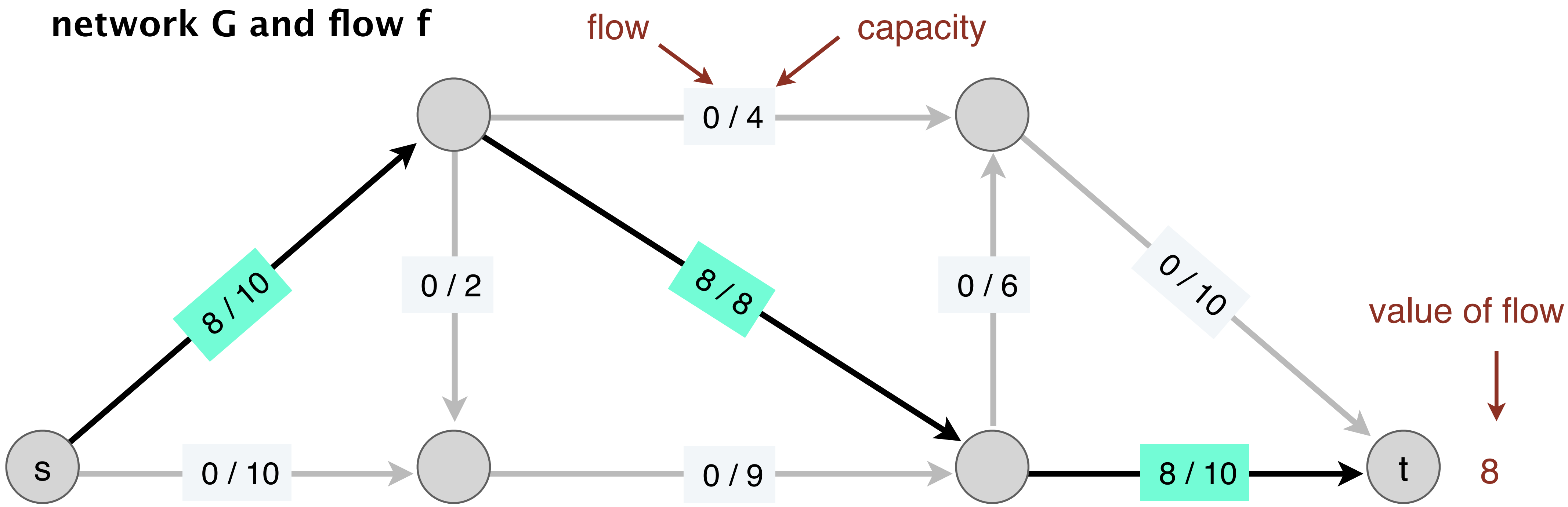


P in residual network G_f

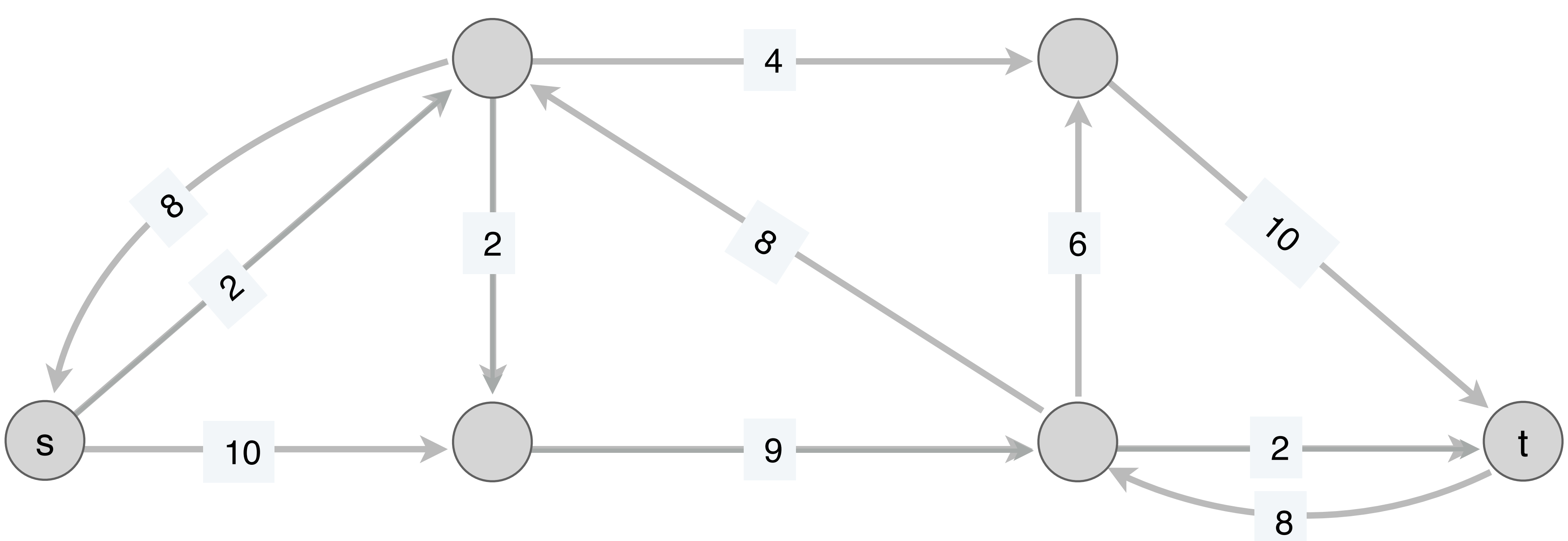


Ford-Fulkerson Example

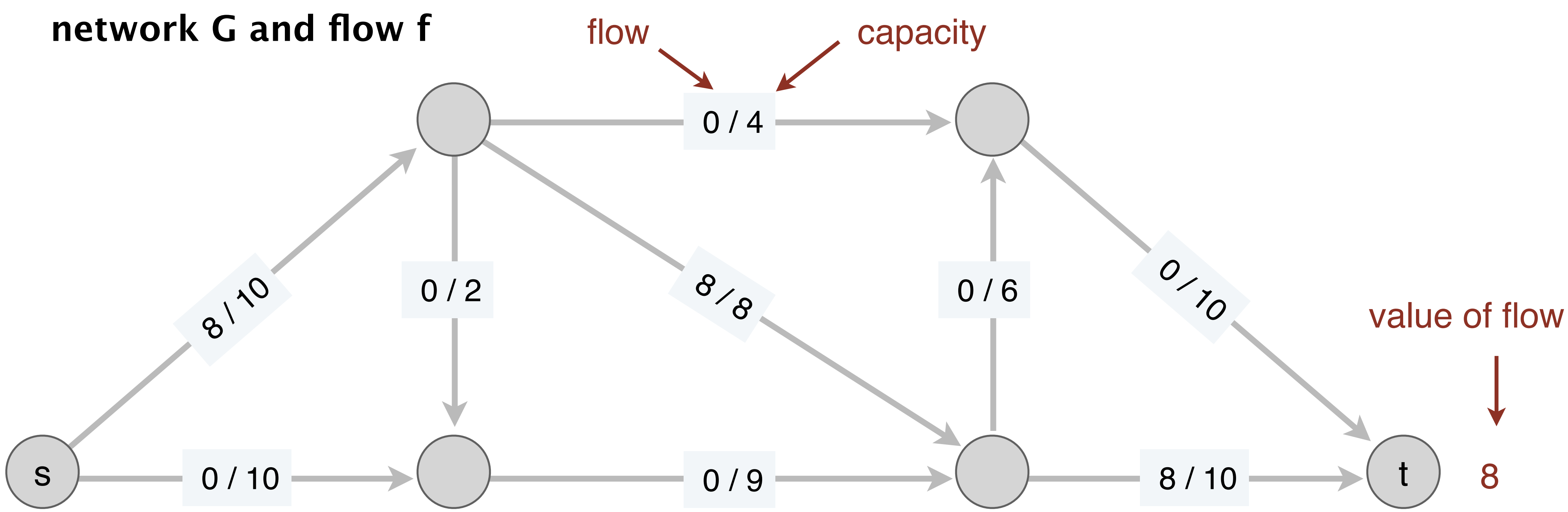
network G and flow f

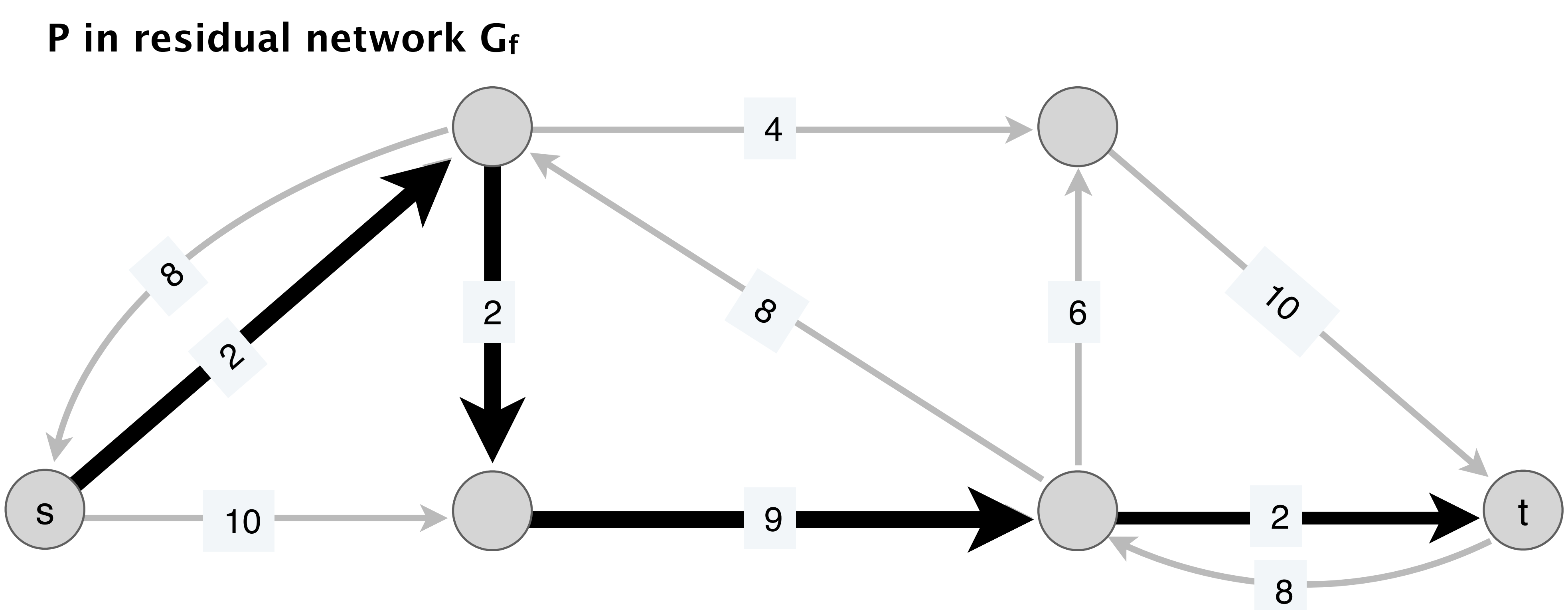


residual network G_f

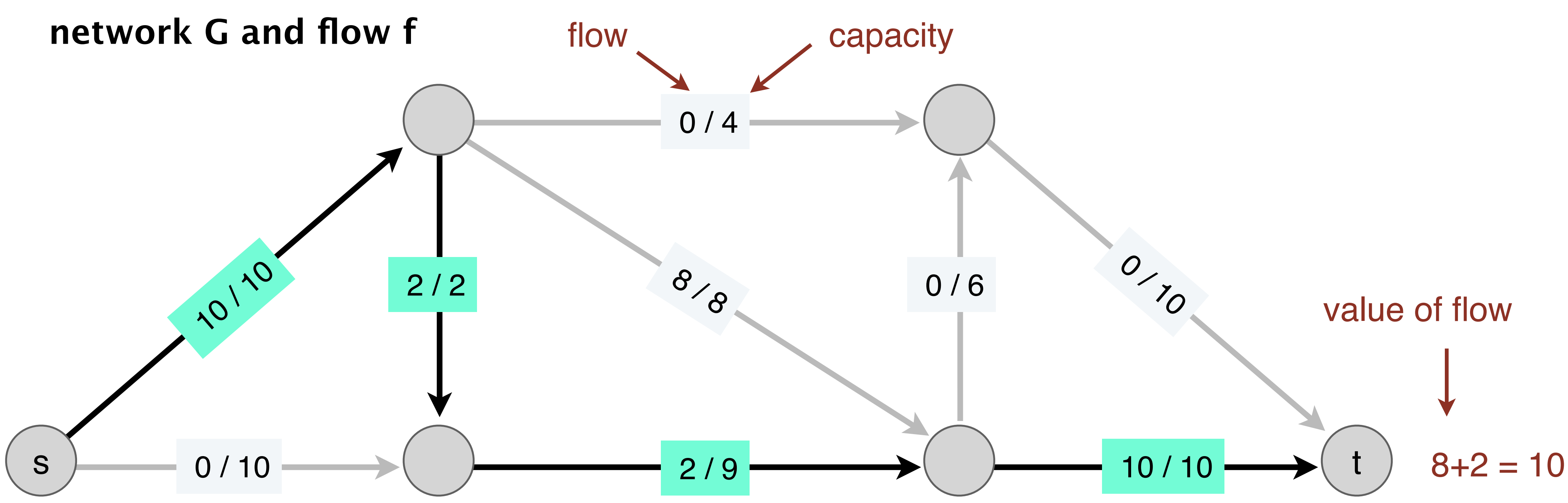


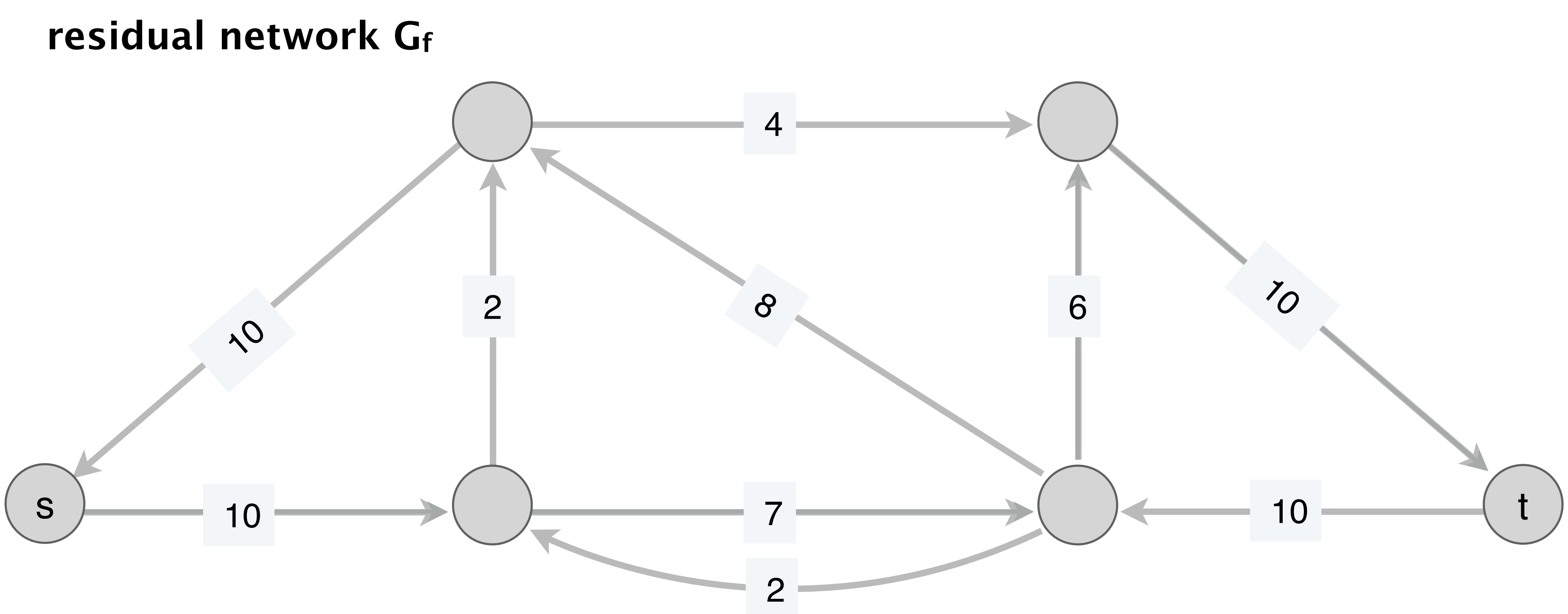
Ford-Fulkerson Example





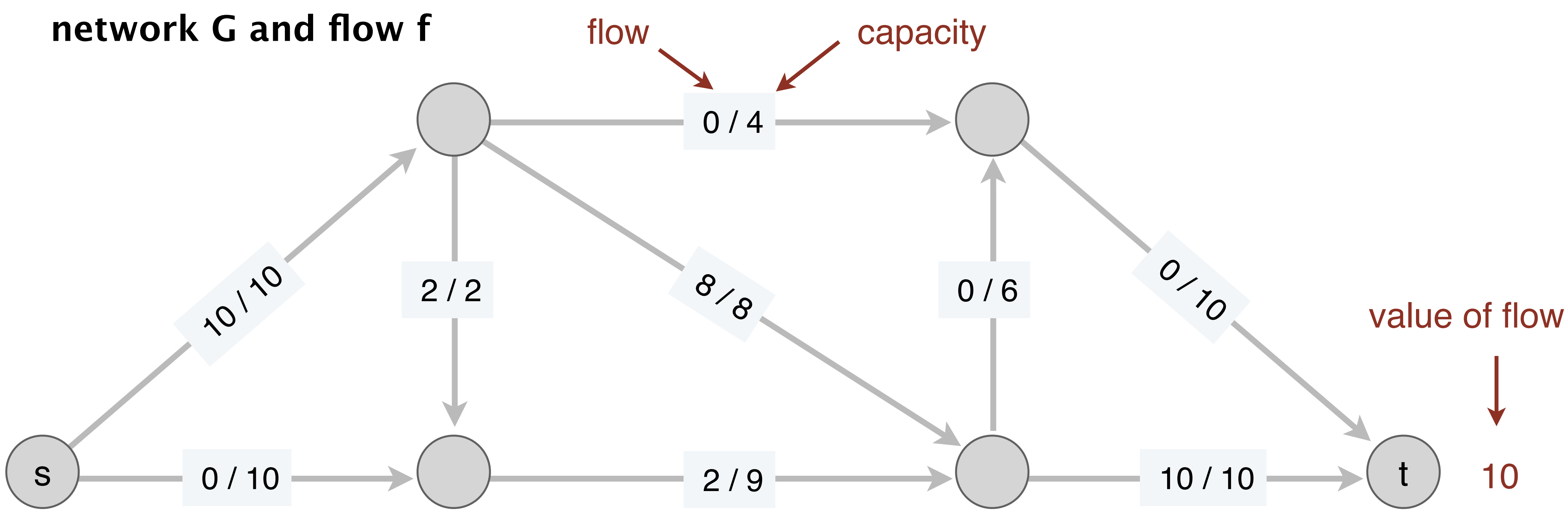
Ford-Fulkerson Example



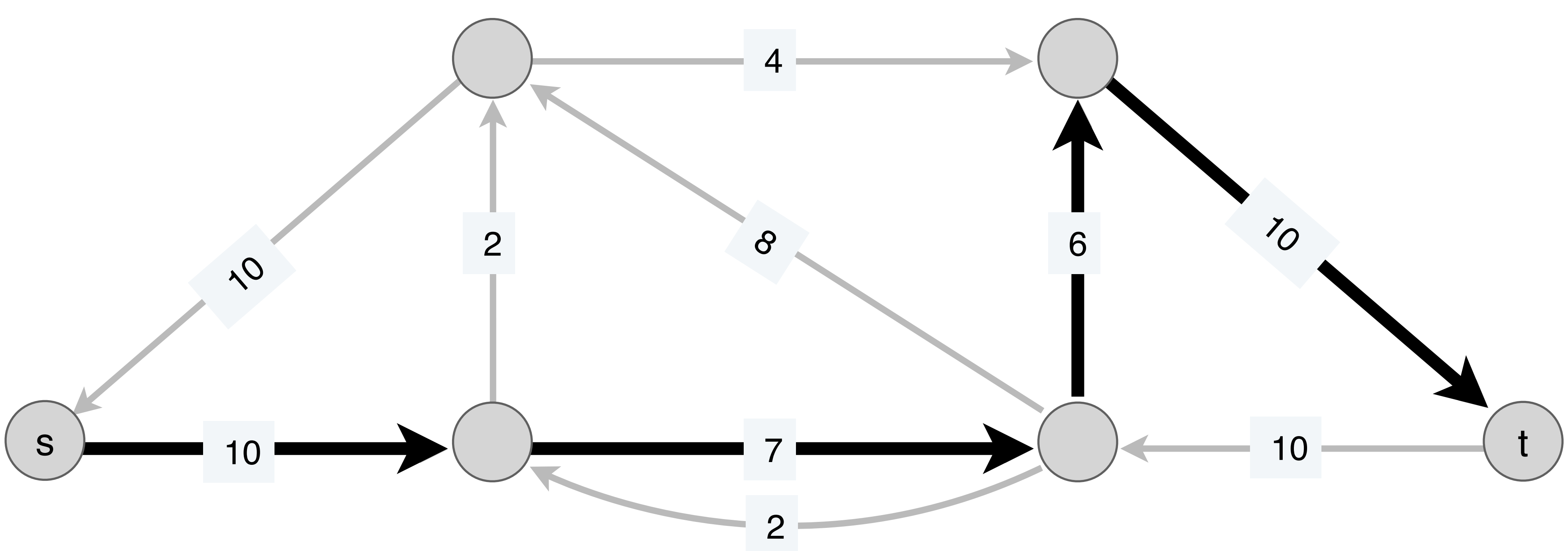


Ford-Fulkerson Example

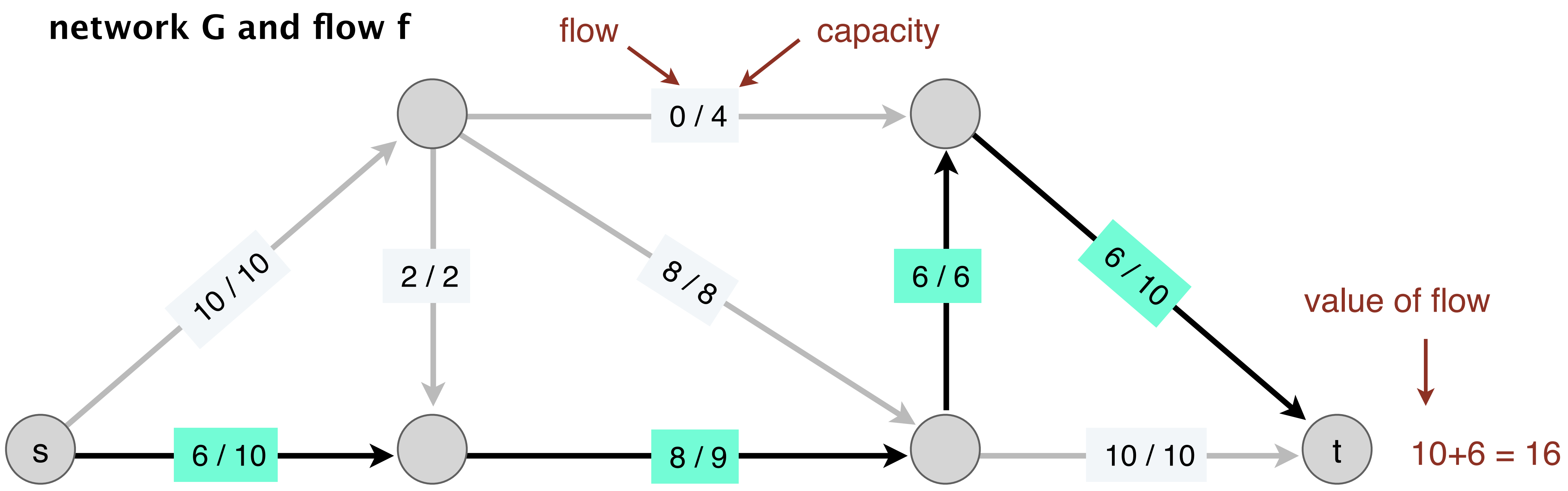
network G and flow f

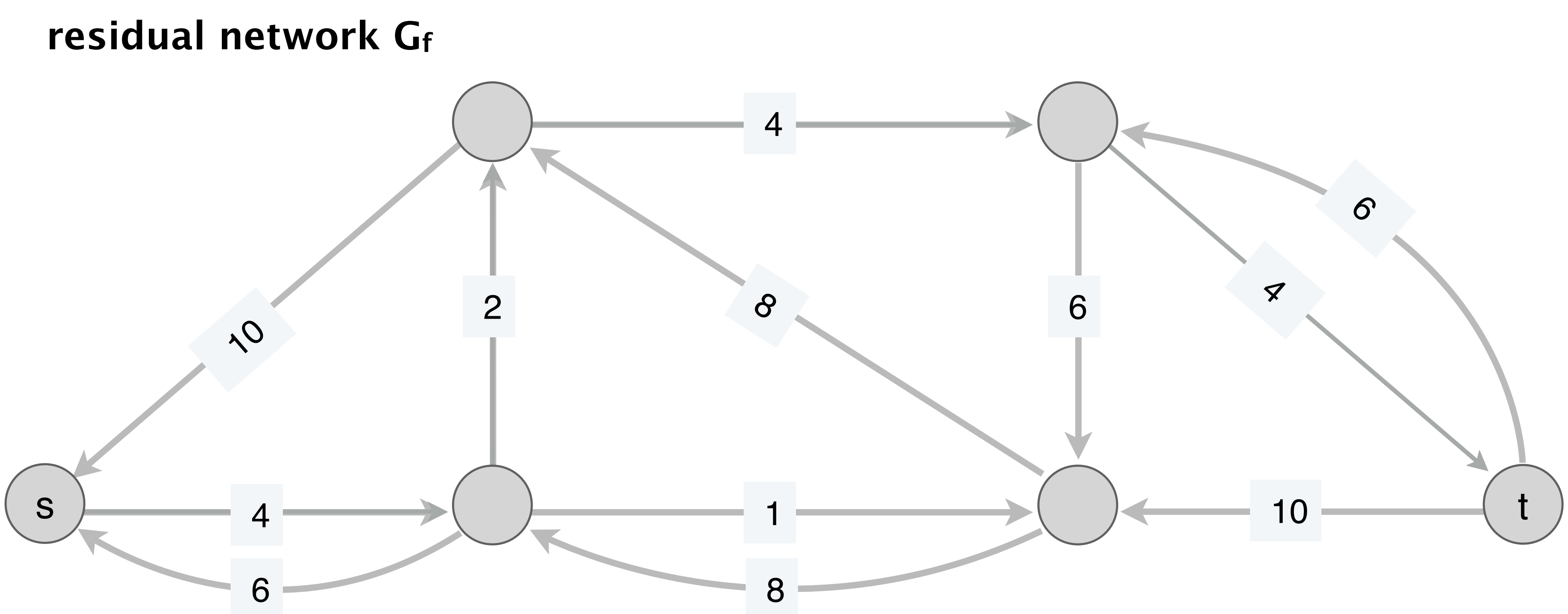


P in residual network G_f

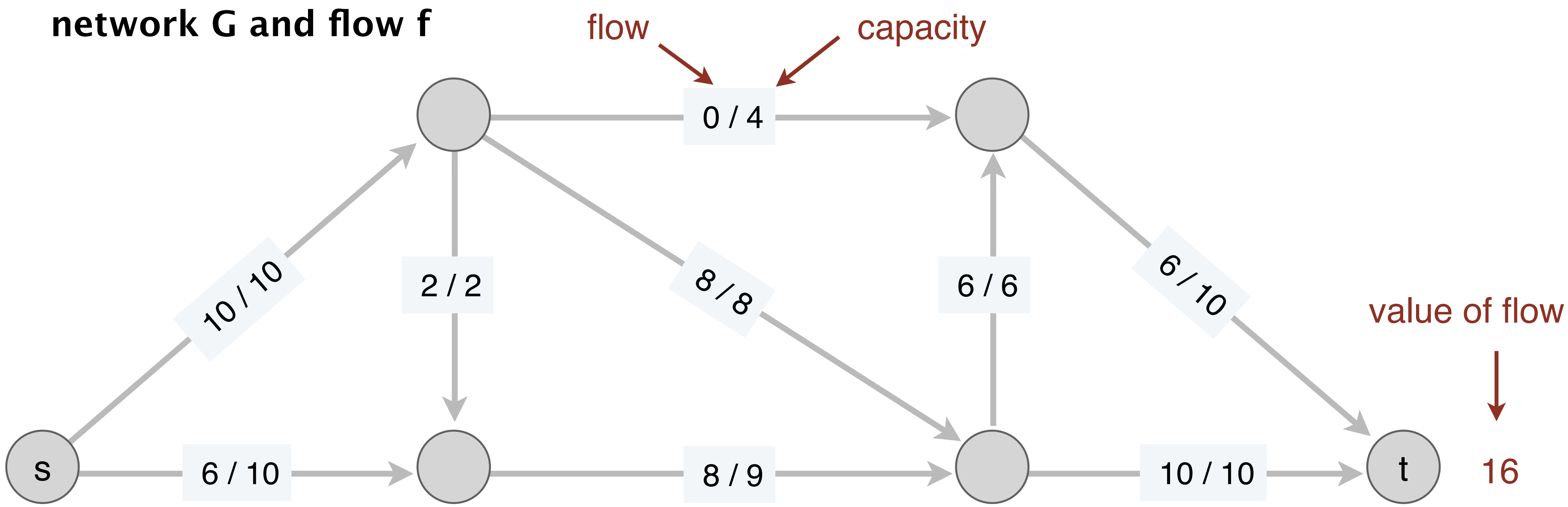


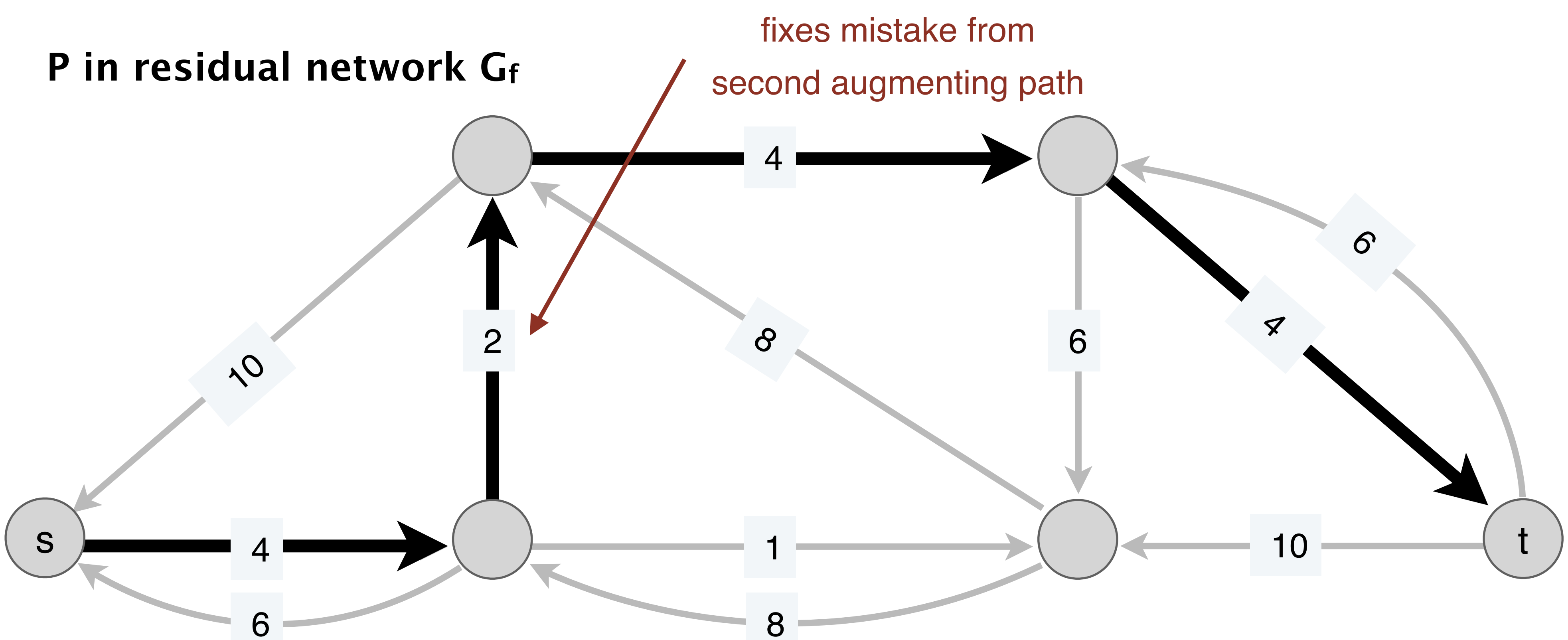
Ford-Fulkerson Example





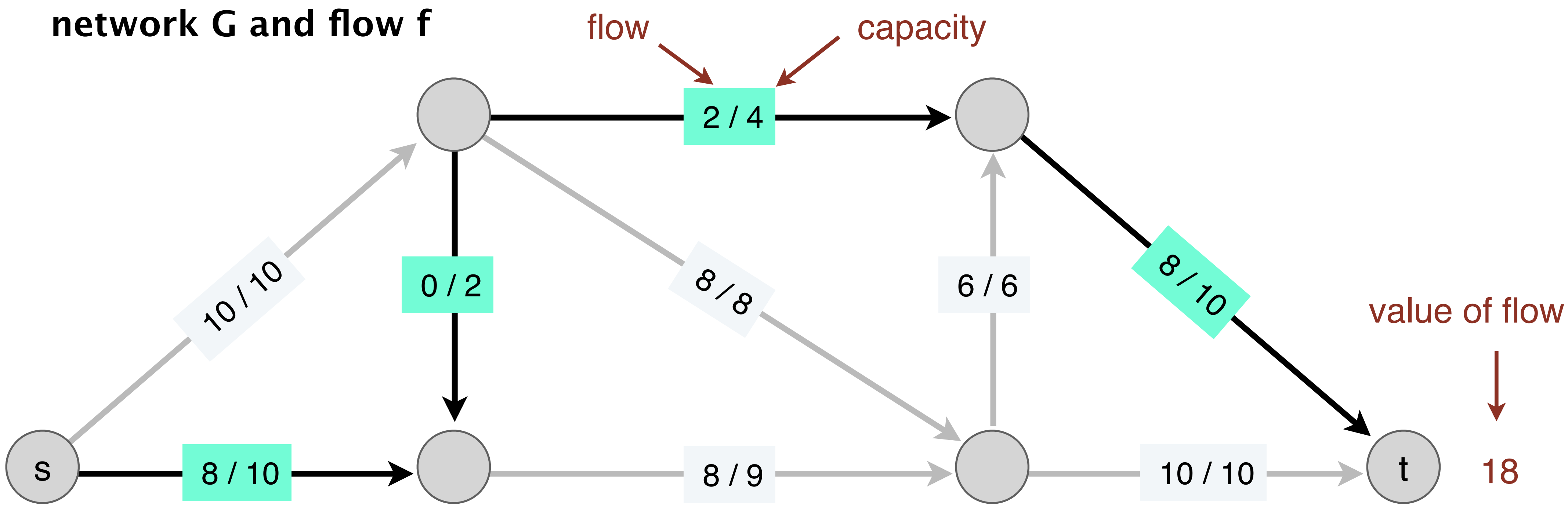
Ford-Fulkerson Example



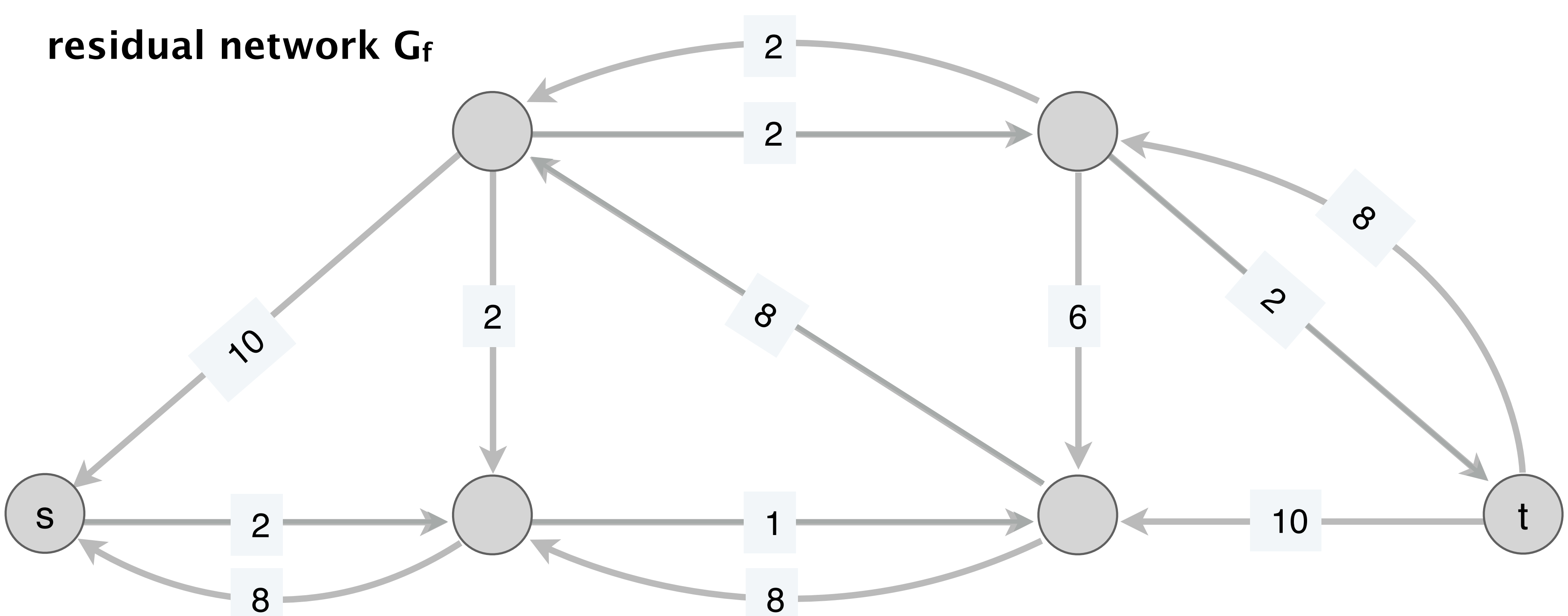


Ford-Fulkerson Example

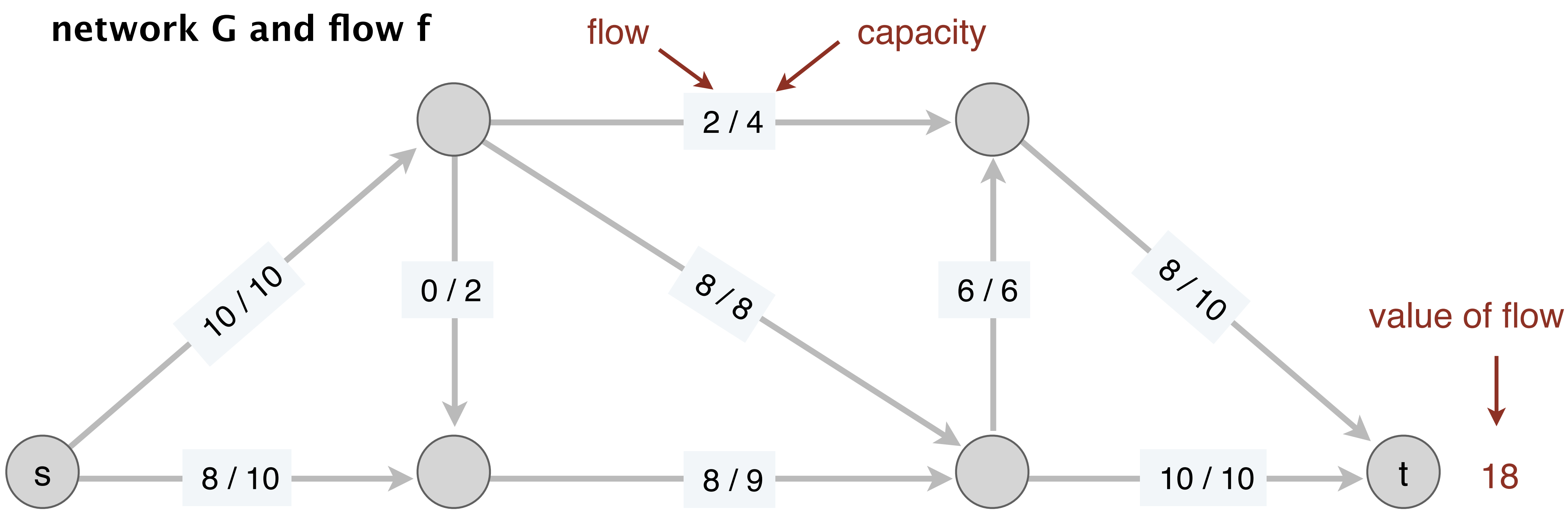
network G and flow f

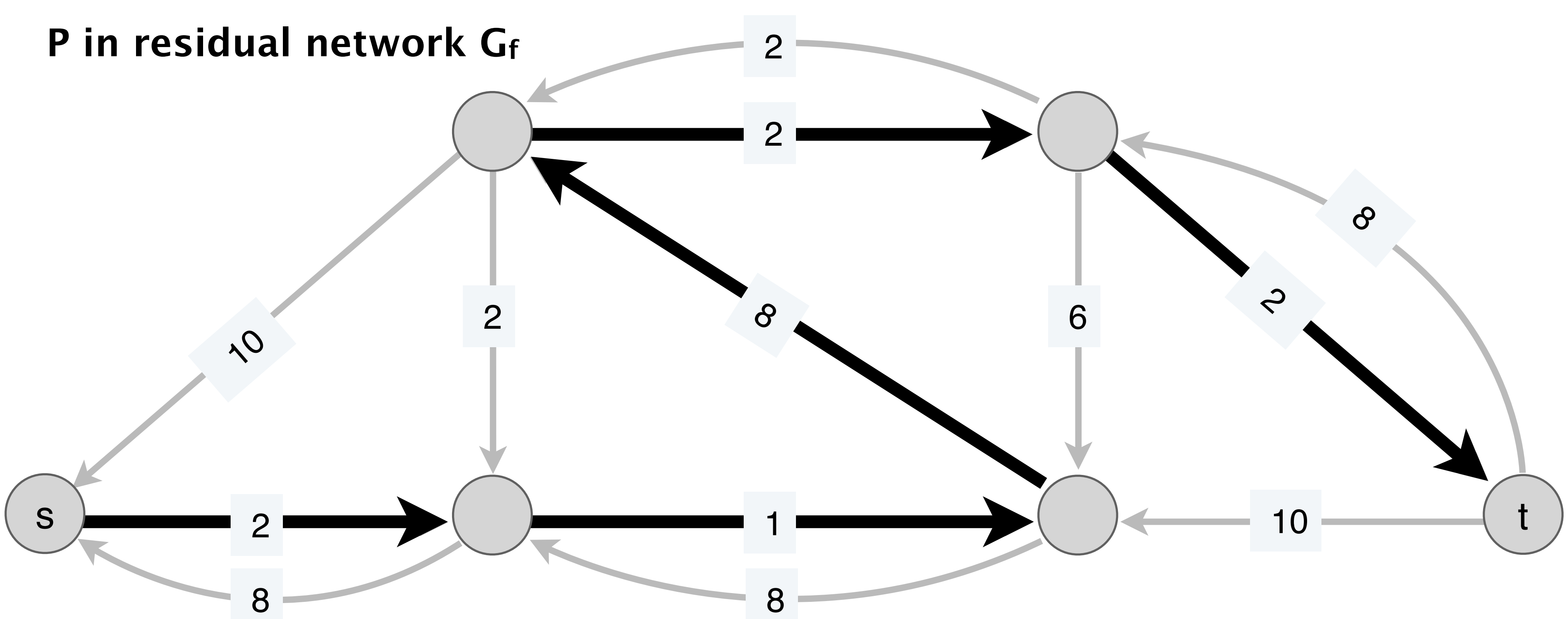


residual network G_f

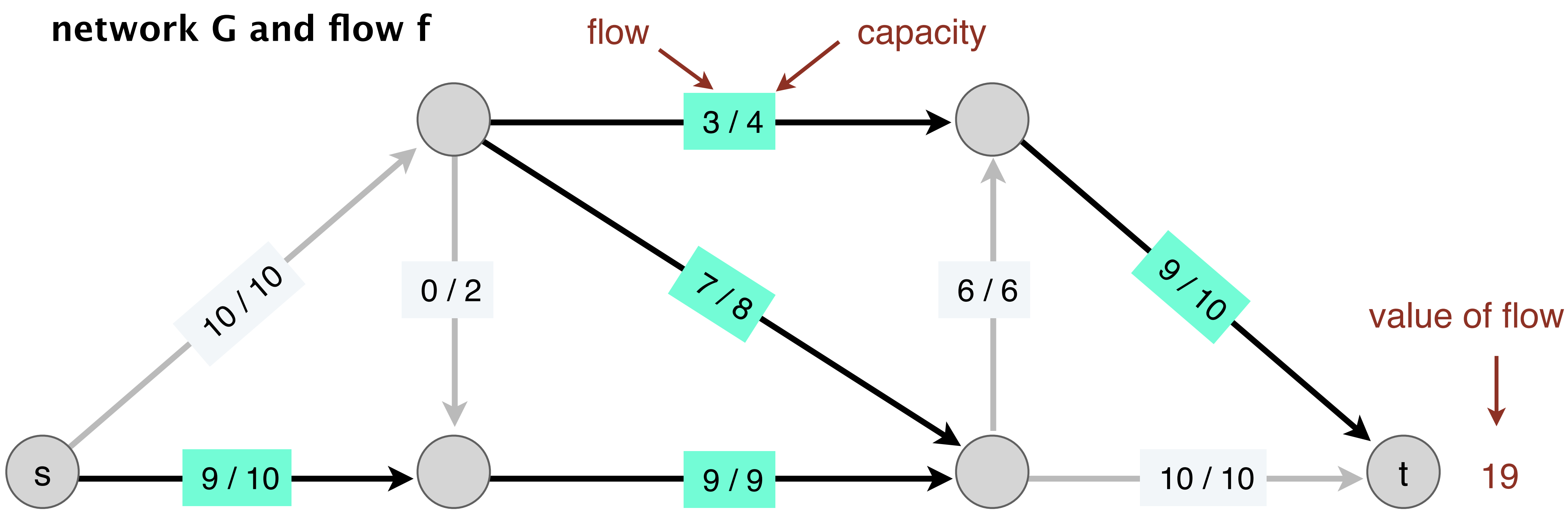


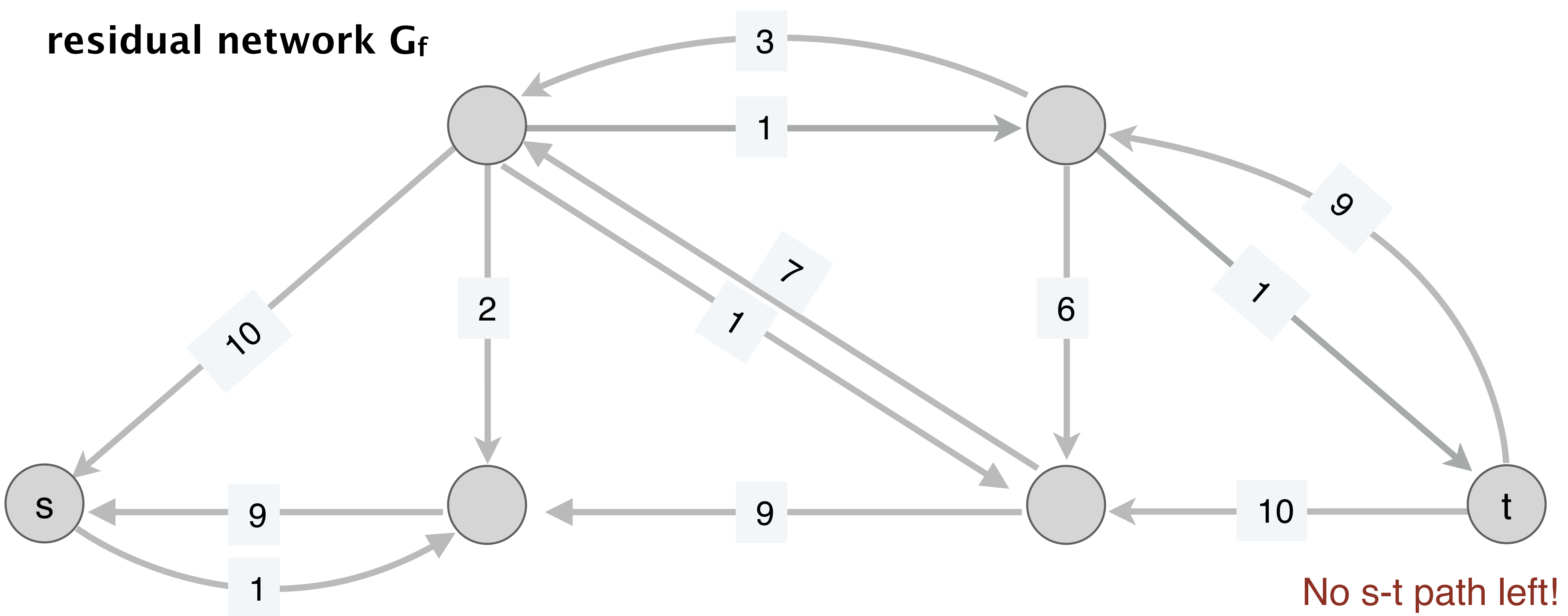
Ford-Fulkerson Example



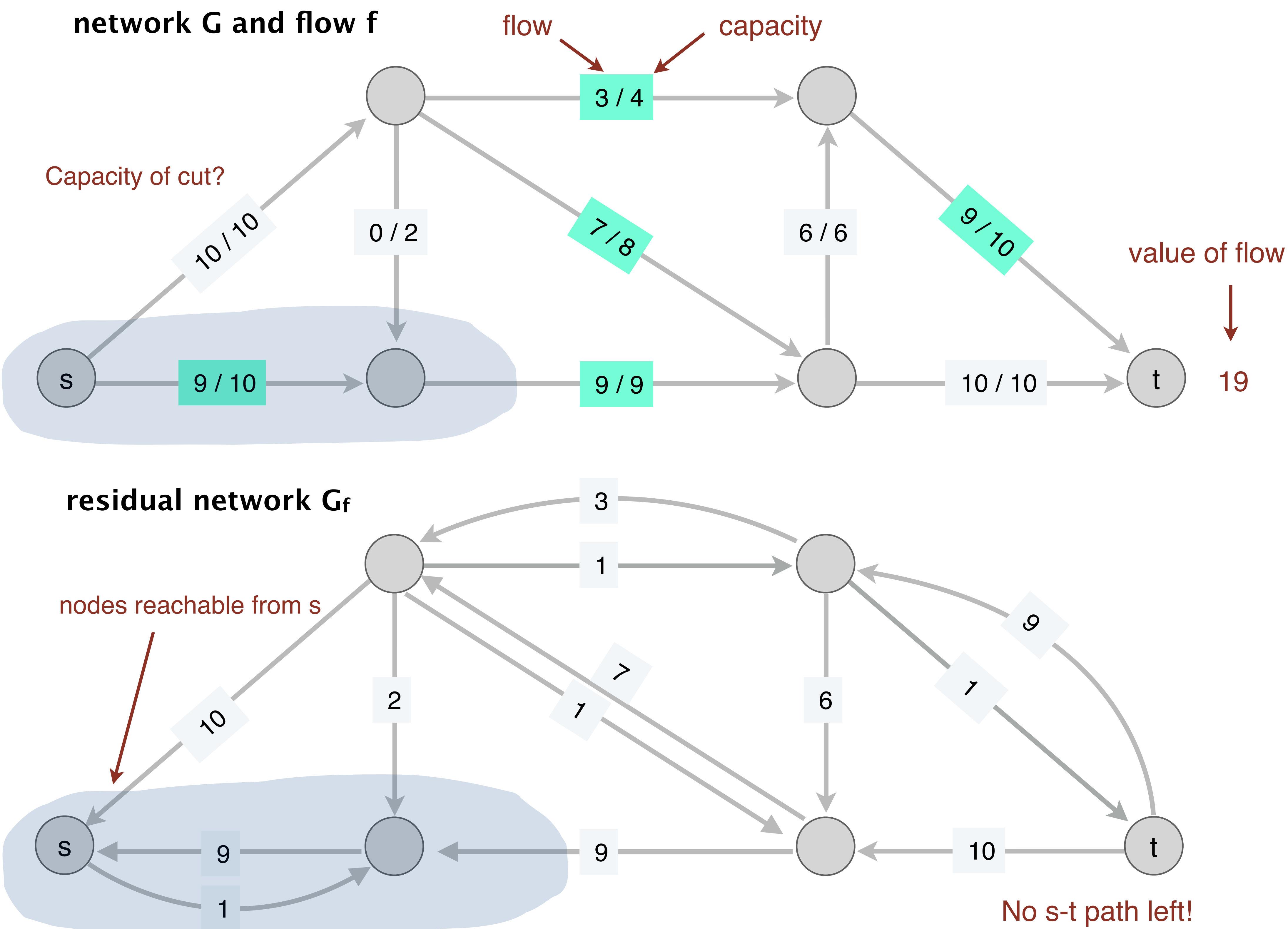


Ford-Fulkerson Example





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!

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)