Introduction to Network Flows

Overview So Far & Going Forward

- So far, algorithmic paradigms:
  - Traversal-based graph algorithms
  - Greedy algorithms
  - Divide and conquer/ Recursion
  - Dynamic Programming/ Recursion without Repetition
- Next: “Flows” — model a variety of optimization problems
- After — Intractability (P vs NP, NP hard, NP complete, etc.)
- Finally — Approximation and Randomized Algorithms

Network Flow History

- In 1950s, US military researchers Harris and Ross wrote a classified report about the rail network linking Soviet Union and Easter 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

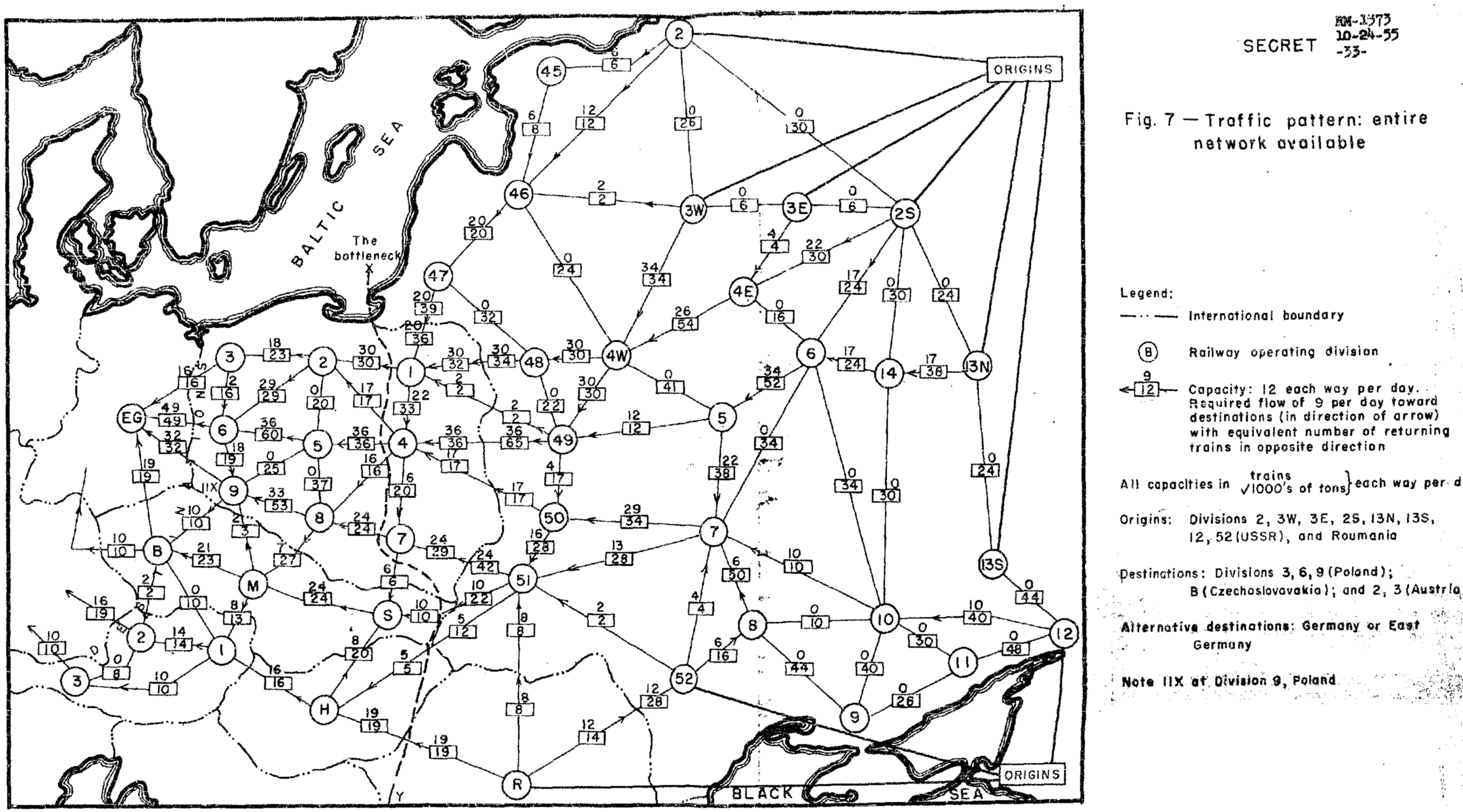
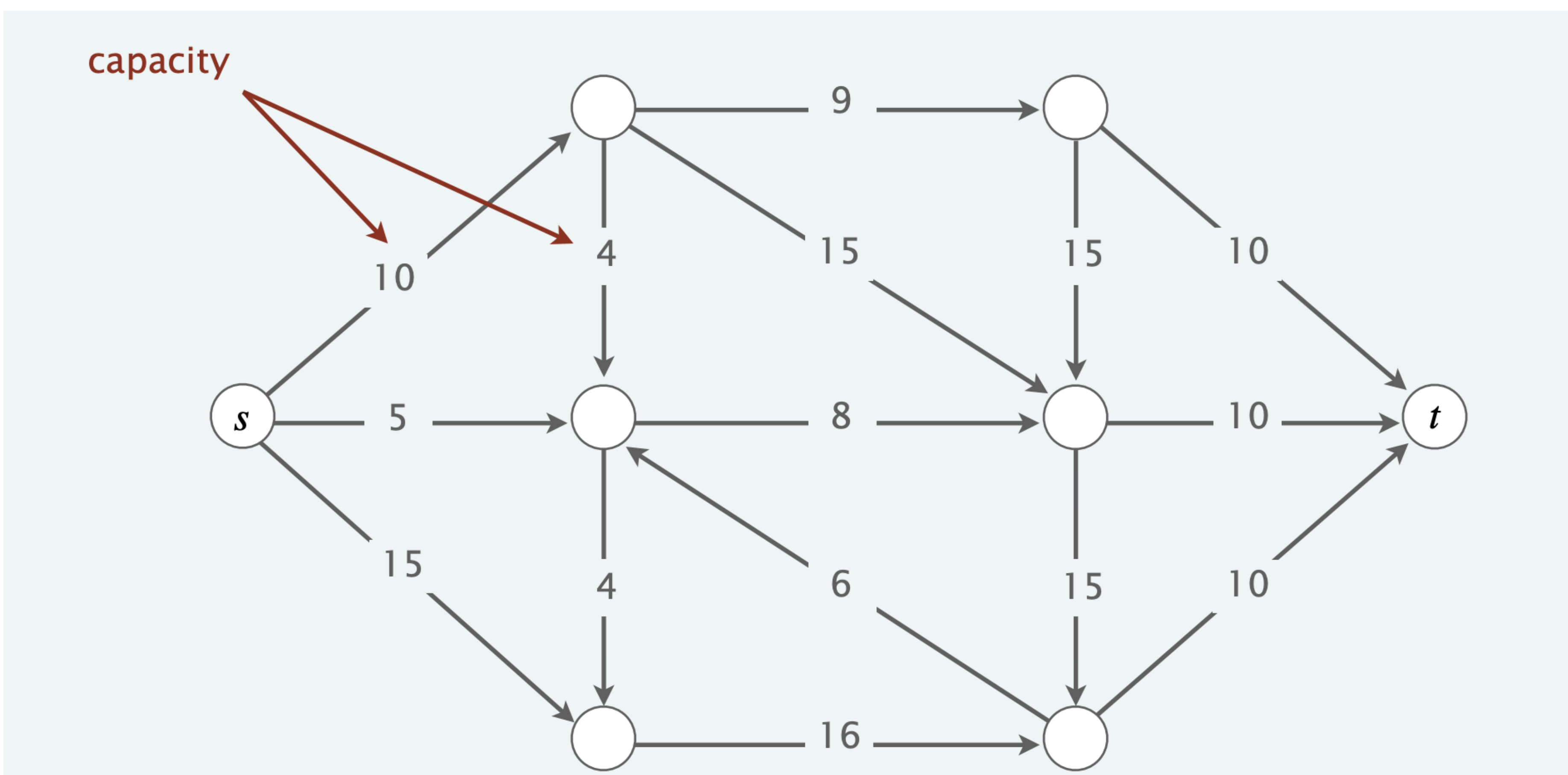


Image Credits: — Jeff Erickson's book and T[homas] E. Harris and F[rank] S. Ross. Fundamentals of a method for evaluating rail net capacities. The RAND Corporation, Research Memorandum RM-1517, October 24, 1955. United States Government work in the public domain. http://www.dtic.mil/dtic/tr/fulltext/u2/093458.pdf

What's a Flow Network?

- A flow network is just 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
  - Edge capacities $c(e) > 0$ for each edge $e \in E$



Simplifying Assumptions/Notations

- Assume that each node v is on some s-t path, that is, $s \rightsquigarrow v \rightsquigarrow t$ exists, for any vertex $v \in V$
  - Implies G is connected, and $m \geq n - 1$
- Assume capacities are integers
- For simplifying expositions, assume $c(e) = 0$ if $e = (u, v)$ is not an edge, that is, for $u, v \in V$ and edge $(u, v) \notin E$
- Non-existent edges/capacities not shown in figures
- Directed edge (u, v) written as $u \rightarrow v$

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) $f : E \to \mathbb{Z}^+$ that satisfies:

  Flow conservation: $f_{in}(v) = f_{out}(v)$, for $v \neq s, t$ where

$$f_{in}(v) = \sum_u f(u \to v) \text{ and } f_{out}(v) = \sum_w f(v \to w)$$

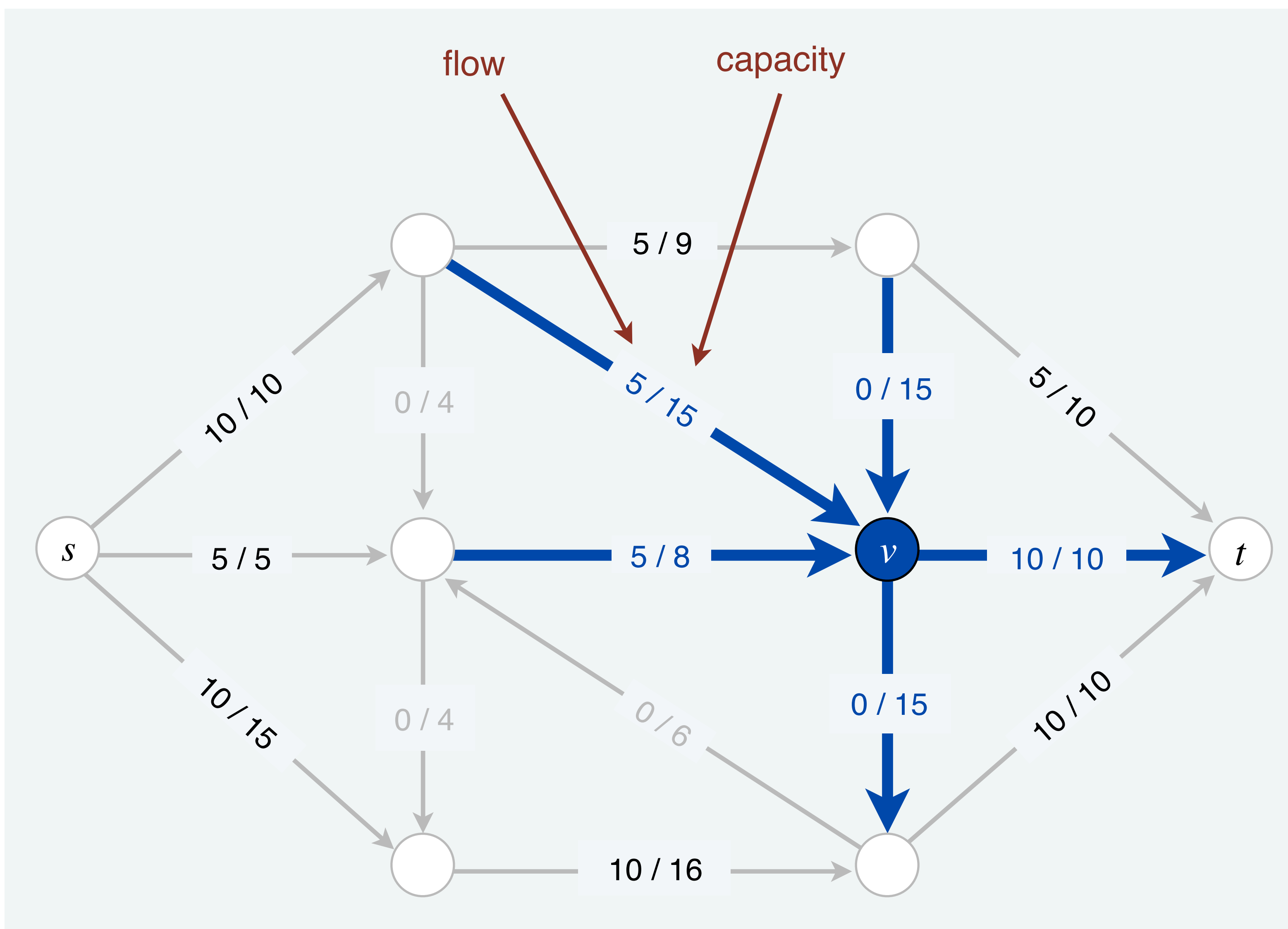
  That is, flow into v equals flow out of v

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

What is a Feasible Flow

- An (s, t)-flow is feasible if it satisfies the capacity constraints of the network, that is,:

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

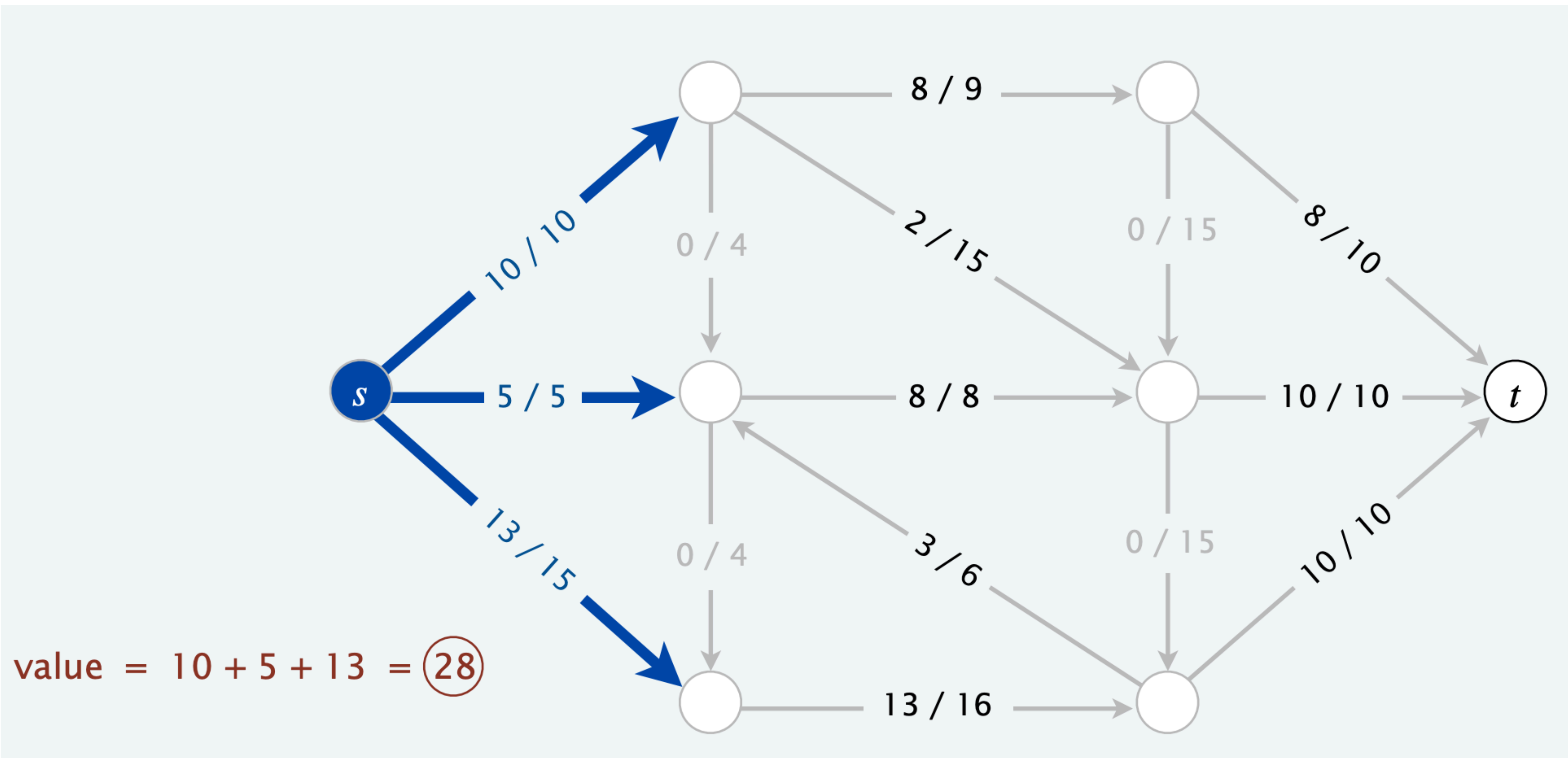


Value of a Flow

- **Definition.** The **value** of a flow f, written $v(f)$, is $f_{out}(s)$.
    - **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 $v \neq s, t : f_{in}(v) = f_{out}(v)$, leaving only $f_{in}(s) + f_{out}(s) = f_{in}(t) + f_{out}(t)$
        - But $f_{in}(s) = f_{out}(t) = 0$ ∎
- **Corollary.** $v(f) = f_{in}(t)$.

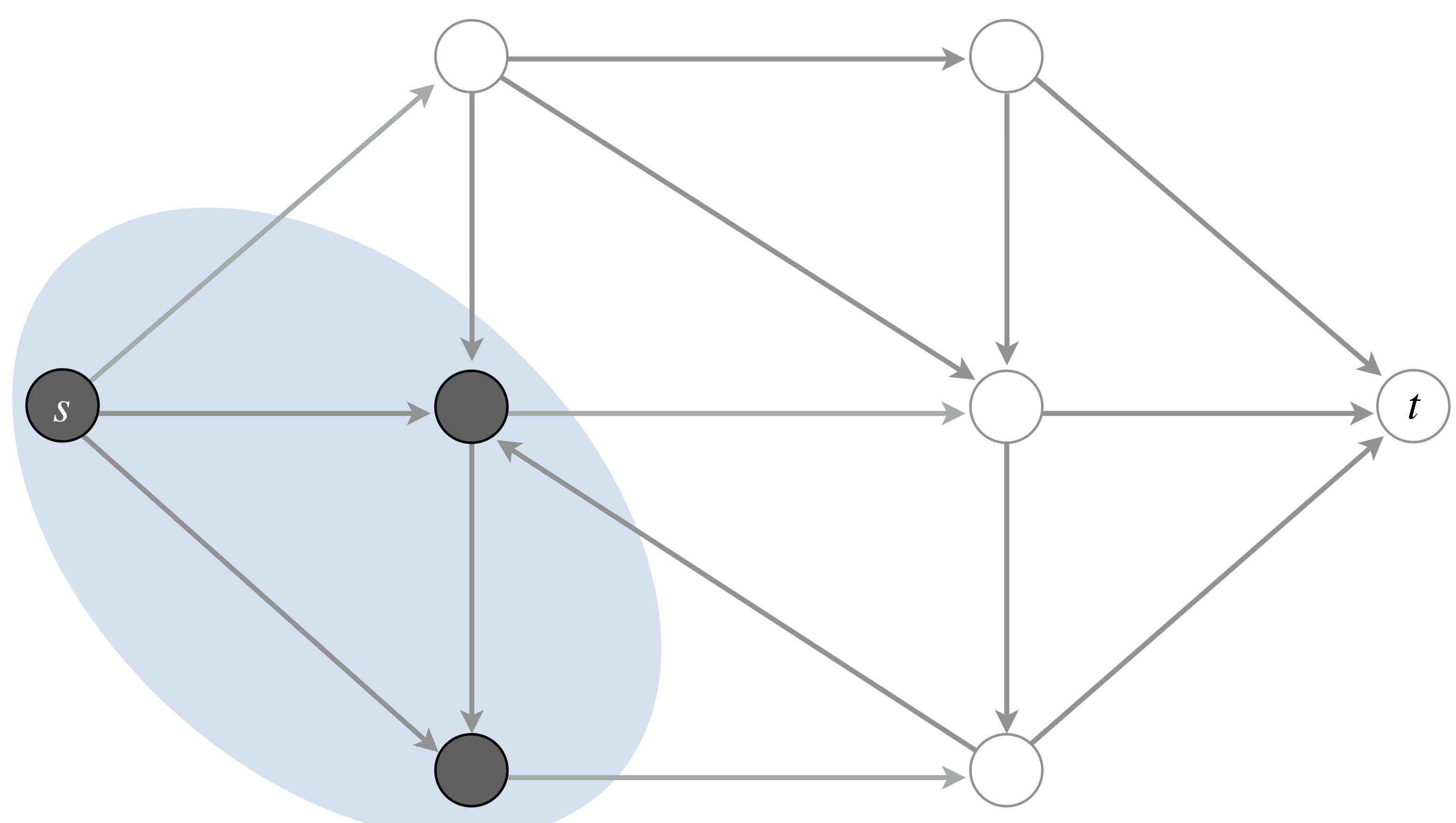
Max-Flow Problem

- Given a flow network, find a flow of maximum value.



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 $t \in T$.



Cuts in Flow Networks

- For any flow f on $G = (V, E)$ and any (s, t)-*cut* (S, T), let
  - $f_{out}(S) = \sum_{v \in S, w \in T} f(v \to w)$ (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 $f_{in}(S) = f_{out}(T)$
- **Lemma.** Value of a flow, $v(f) = f_{out}(S) - f_{in}(S)$ is the net-flow out of S, for any (s, t)-cut (S, T).

Cuts in Flow Networks

- **Lemma.** Value of a flow, $v(f) = f_{out}(S) - f_{in}(S)$ is the net-flow out of S, for any (s, t)-cut (S, T).
- **Proof.**
- $v(f) = f_{out}(s)$
- $v(f) = f_{out}(s) - f_{in}(t) = \sum_{v \in S} (f_{out}(v) - f_{in}(v))$ (Adding some zeros)

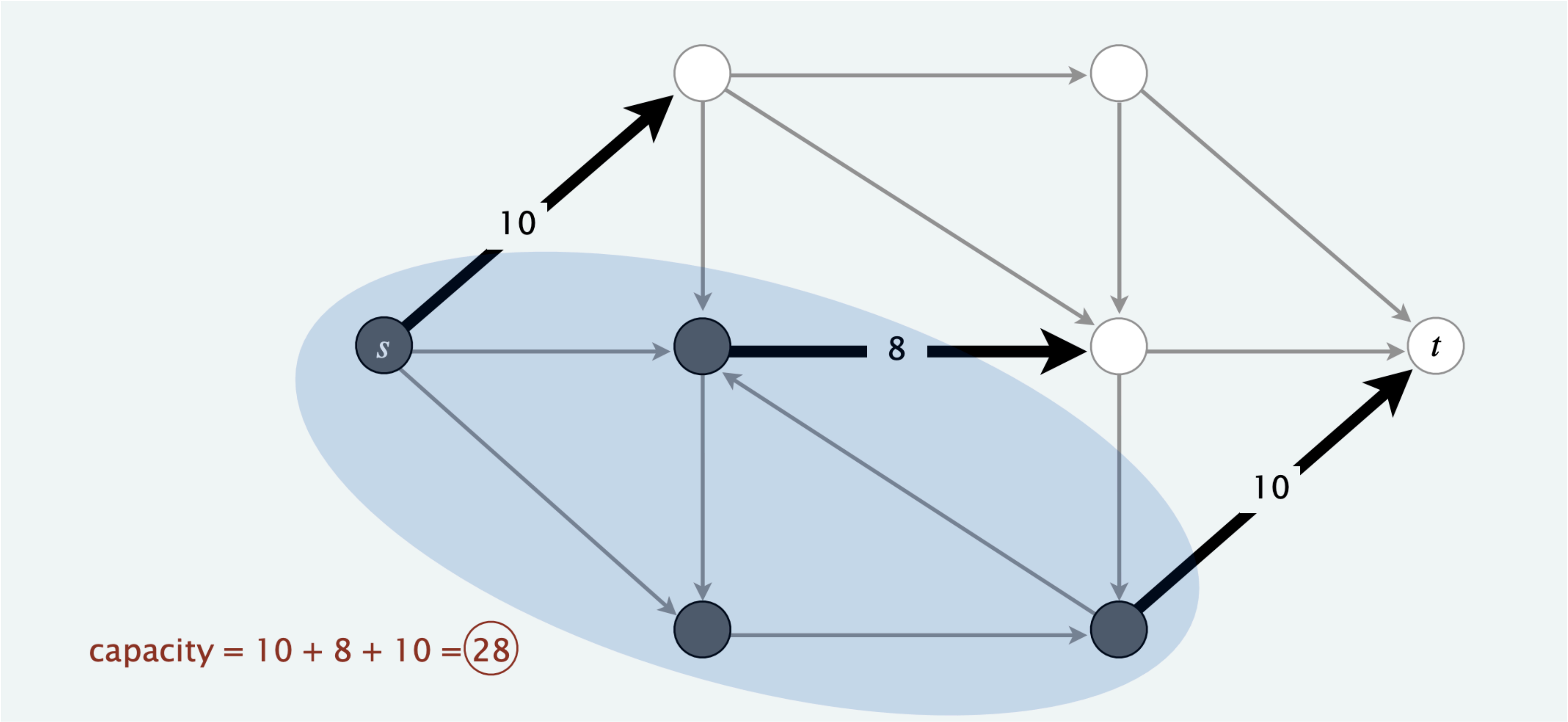
$= \sum_{v \in S} \Big(\sum_{w} f(v \to w) - \sum_{u} f(u \to v) \Big)$ (By definition)

$= \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)$ (all other edges cancel in pairs)

Capacities of Cuts

- Capacity of a (s, t)-cut (S, T) is the sum of the capacities of edges leaving S:

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



Network flow: quiz 1

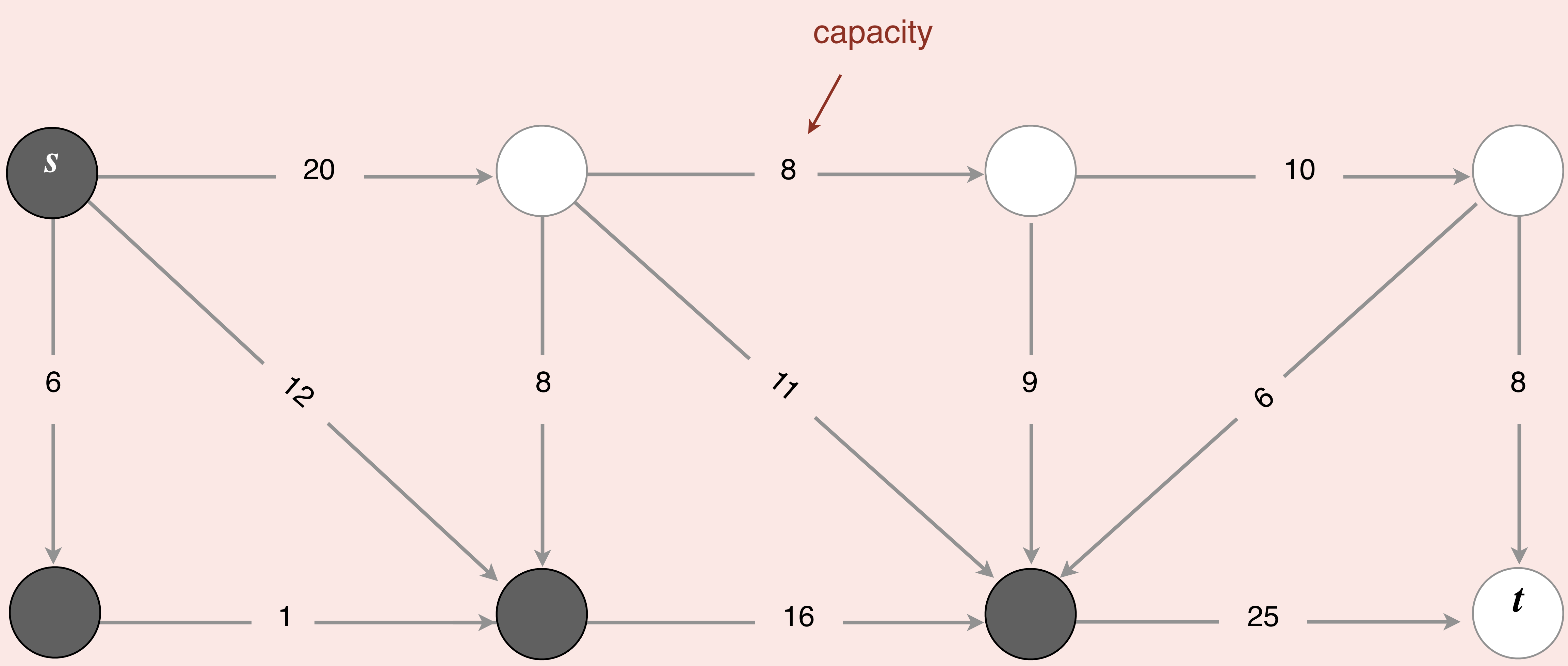
Which is the capacity of the given *st*-cut?

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)



Capacities of Cuts

- Capacity of a (s, t)-cut (S, T) is the sum of the capacities of edges leaving S:
  - $$c(S, T) = \sum_{v \in S, w \in T} f(v \rightarrow w)$$
- A **dual problem to max-flow**:
  - Find an (s, t)-cut of minimum capacity
- **Claim.** Let f be any s-t flow and (S, T) be any s-t cut then $v(f) \leq c(S, T)$

Relationship: Flows and Cuts

- **Claim.** Let f be any s-t flow and (S, T) be any s-t cut then $v(f) \leq c(S, T)$
- **Proof.**
  - $v(f) = f_{out}(S) - f_{in}(S)$

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

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

Max-Flow Min-Cut Theorem

- A beautiful, powerful relationship between these two problems in given by the following theorem
- **Theorem.** Given a flow network G, let f be an (s, t)-flow and let (S, T) be any (s, t)-cut of G then,

  $v(f) = c(S, T)$ if and only if

  f is a flow of maximum value and (S, T) is a cut of minimum capacity.
- Informally, in a flow network the max-flow = min-cut.

Max-Flow Min-Cut Theorem

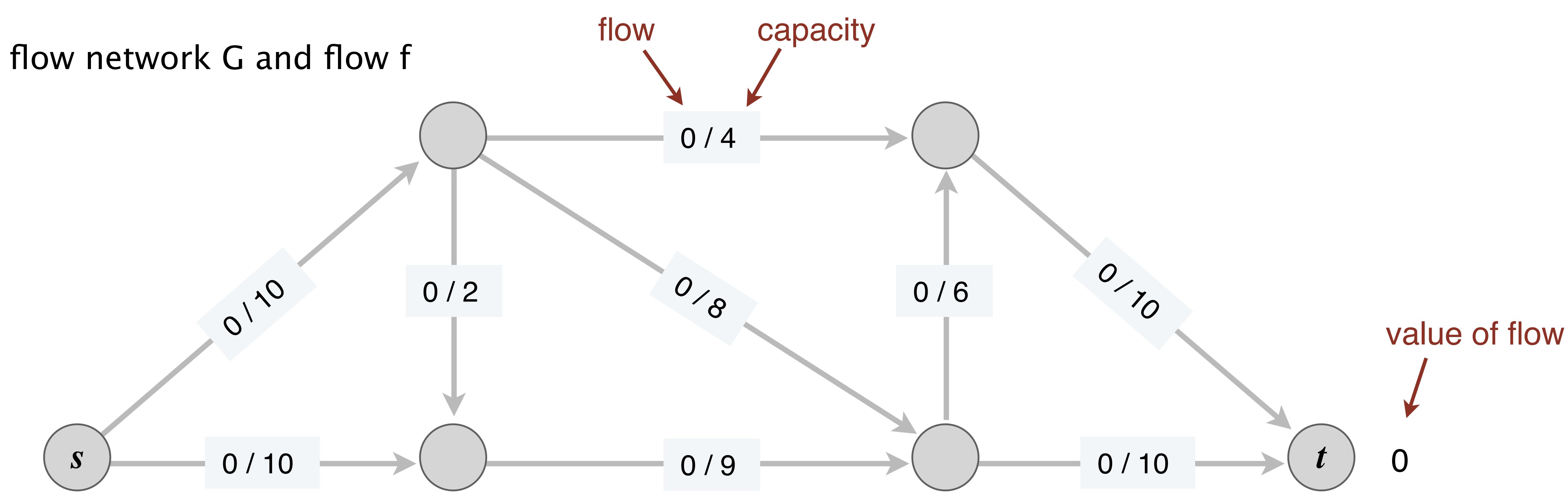
- We will prove the max-flow min-cut their by construction
  - Designing a max-flow algorithm, proving its optimality and showing the max-flow min-cut theorem holds
- Called the Ford-Fulkerson Algorithm
- First, we start with a greedy approach

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 take an example

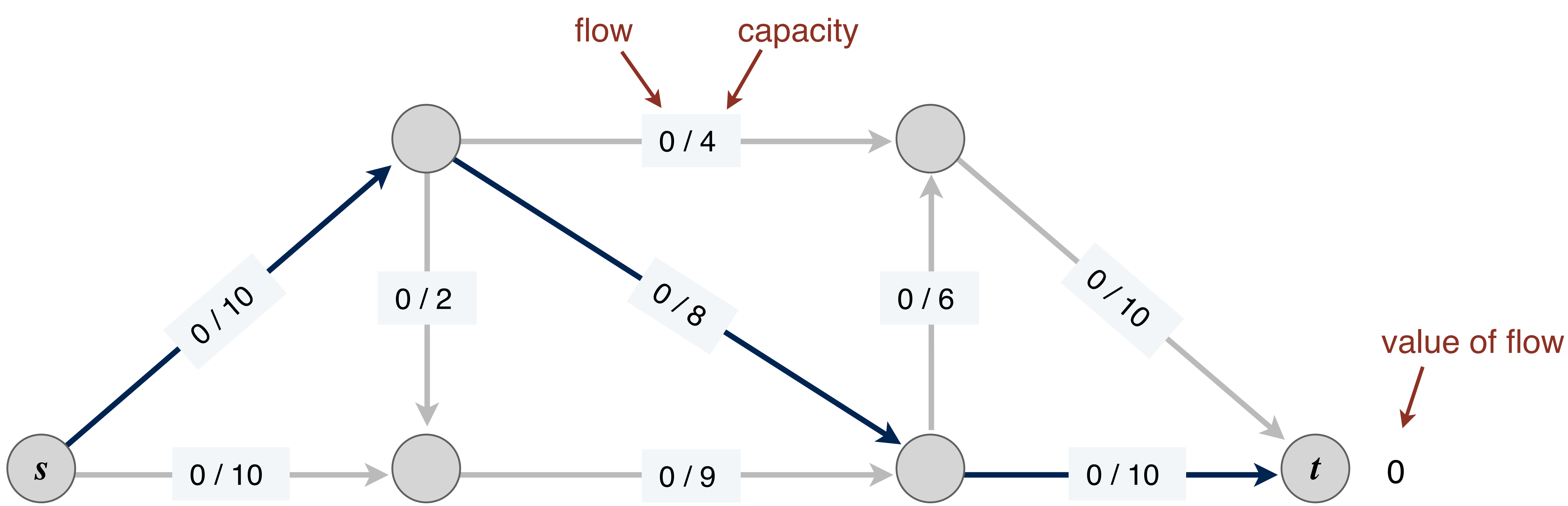
Towards a Max-Flow Algorithm

- 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



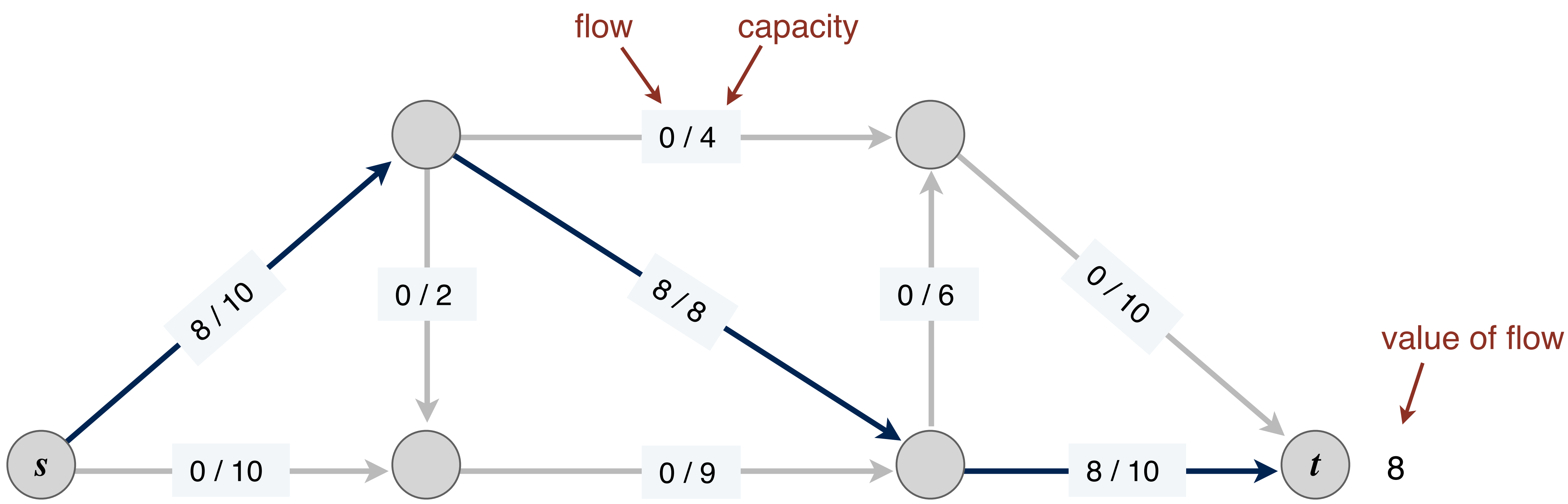
Towards a Max-Flow Algorithm

- 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



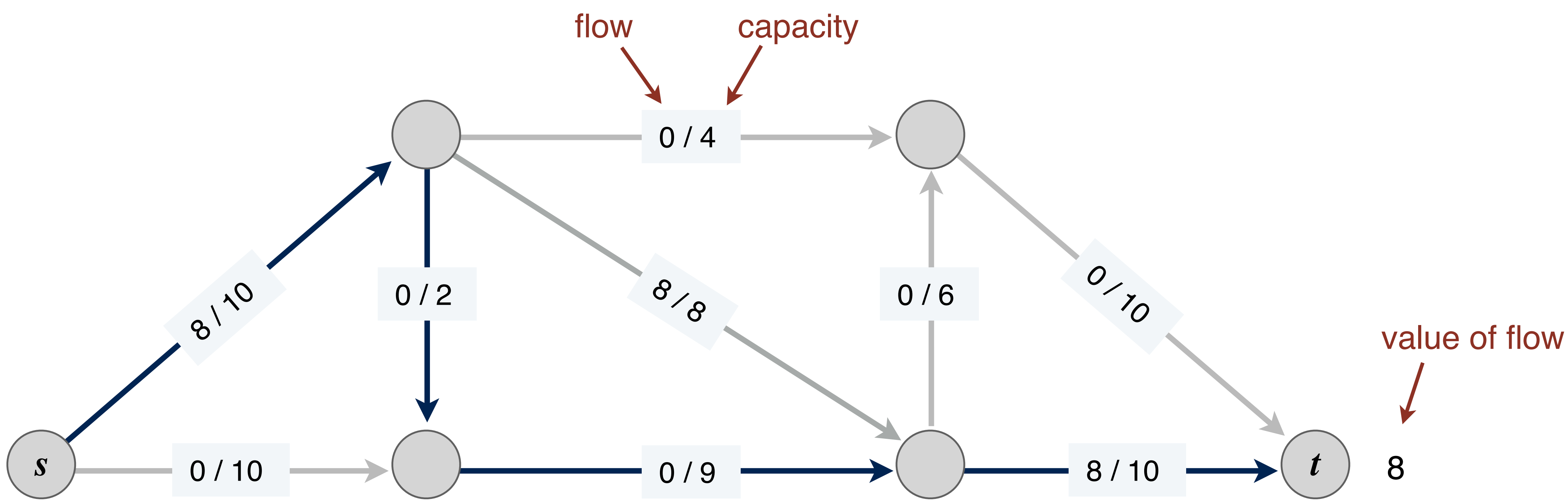
Towards a Max-Flow Algorithm

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



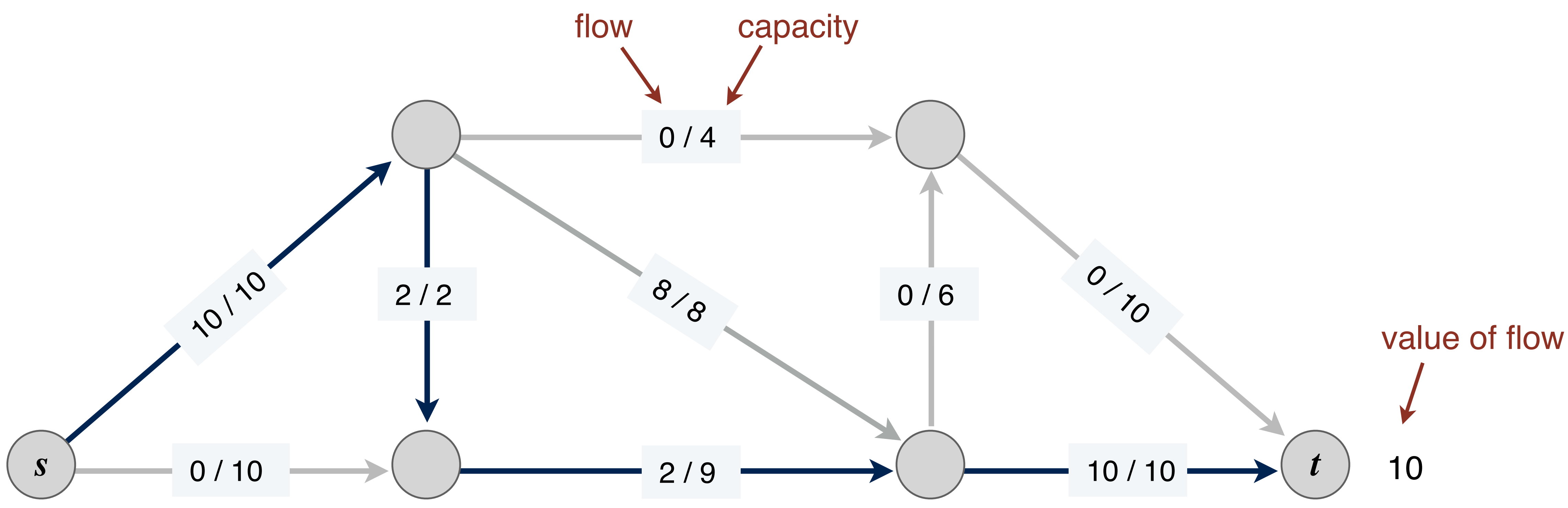
Towards a Max-Flow Algorithm

- 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



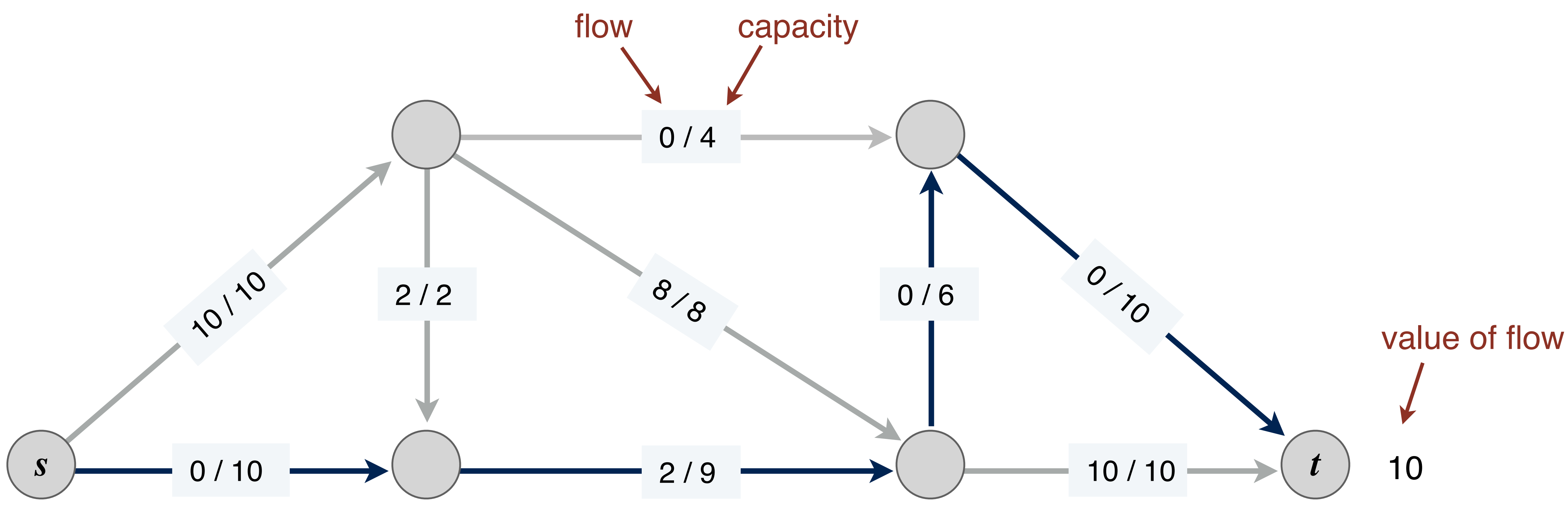
Towards a Max-Flow Algorithm

- 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



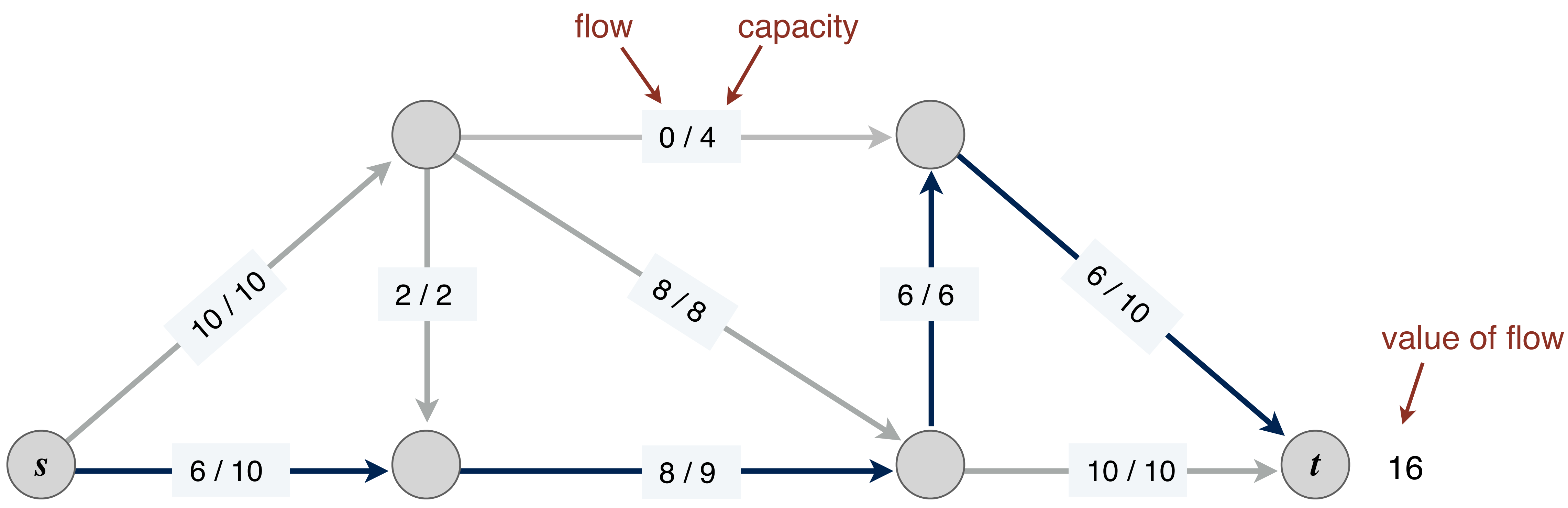
Towards a Max-Flow Algorithm

- 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



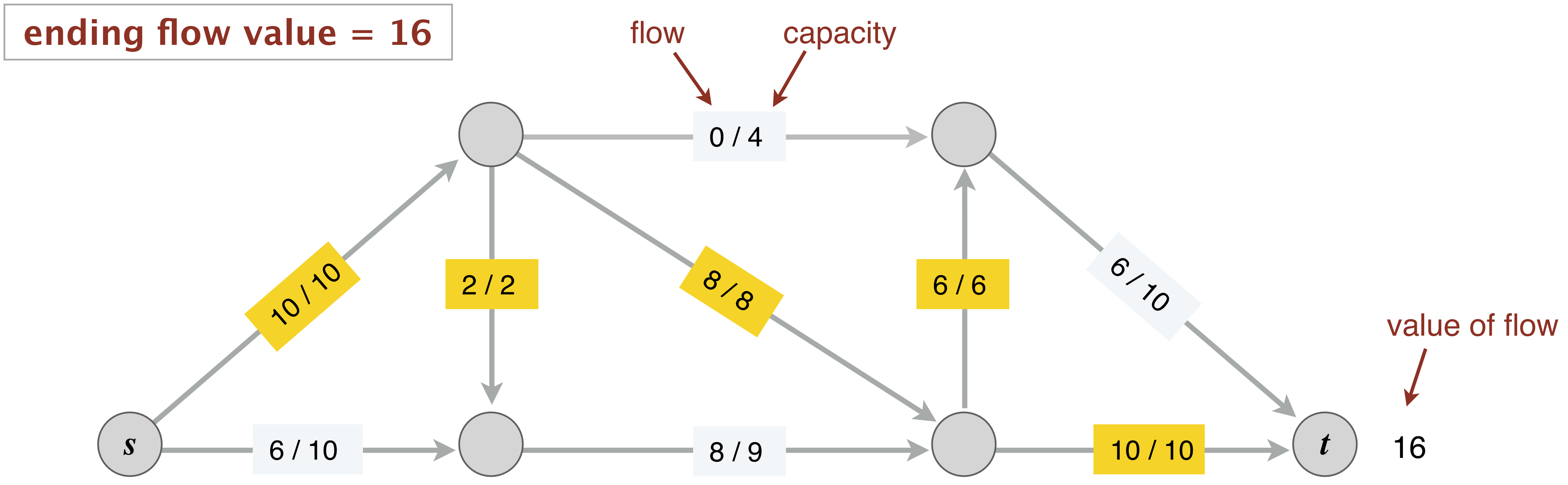
Towards a Max-Flow Algorithm

- 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



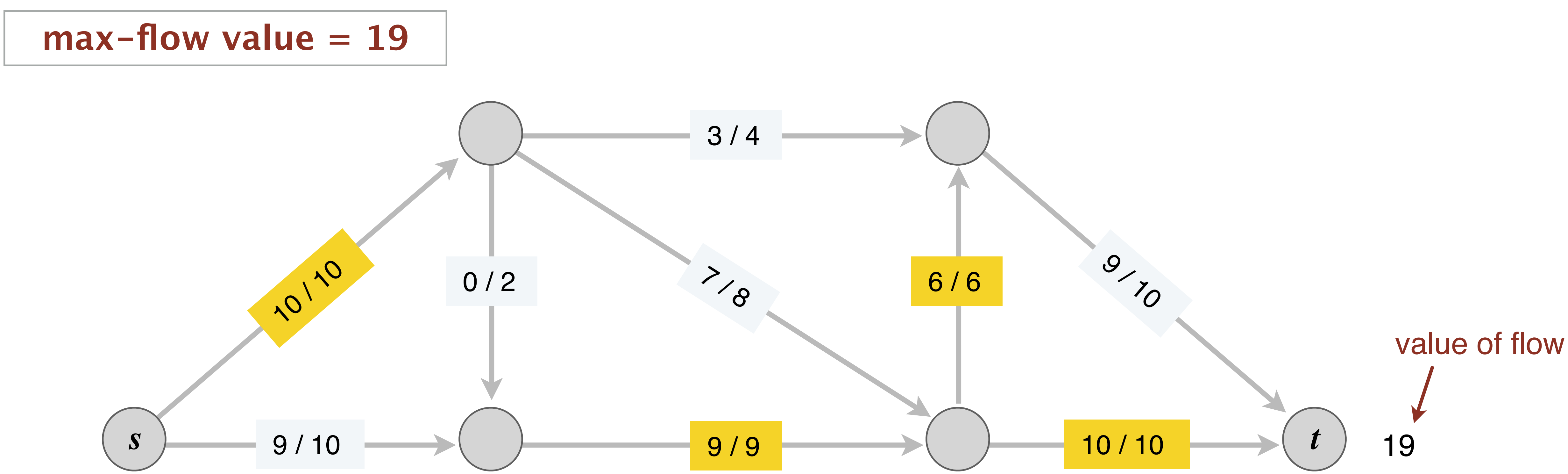
Towards a Max-Flow Algorithm

- 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



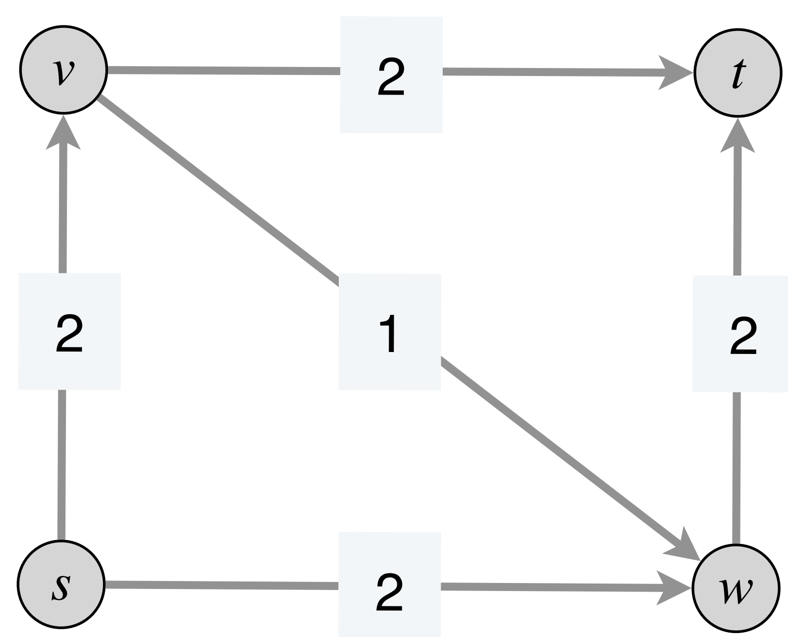
Towards a Max-Flow Algorithm

- 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
  - Unique max flow has $f(v \to w) = 0$
  - Greedy could choose $s \to v \to w \to t$ as first P



- Key: Need a mechanism to "undo" previous flow decisions

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)