Flow Networks:
Ford-Fulkerson Algorithm

Towards a Max-Flow Algorithm

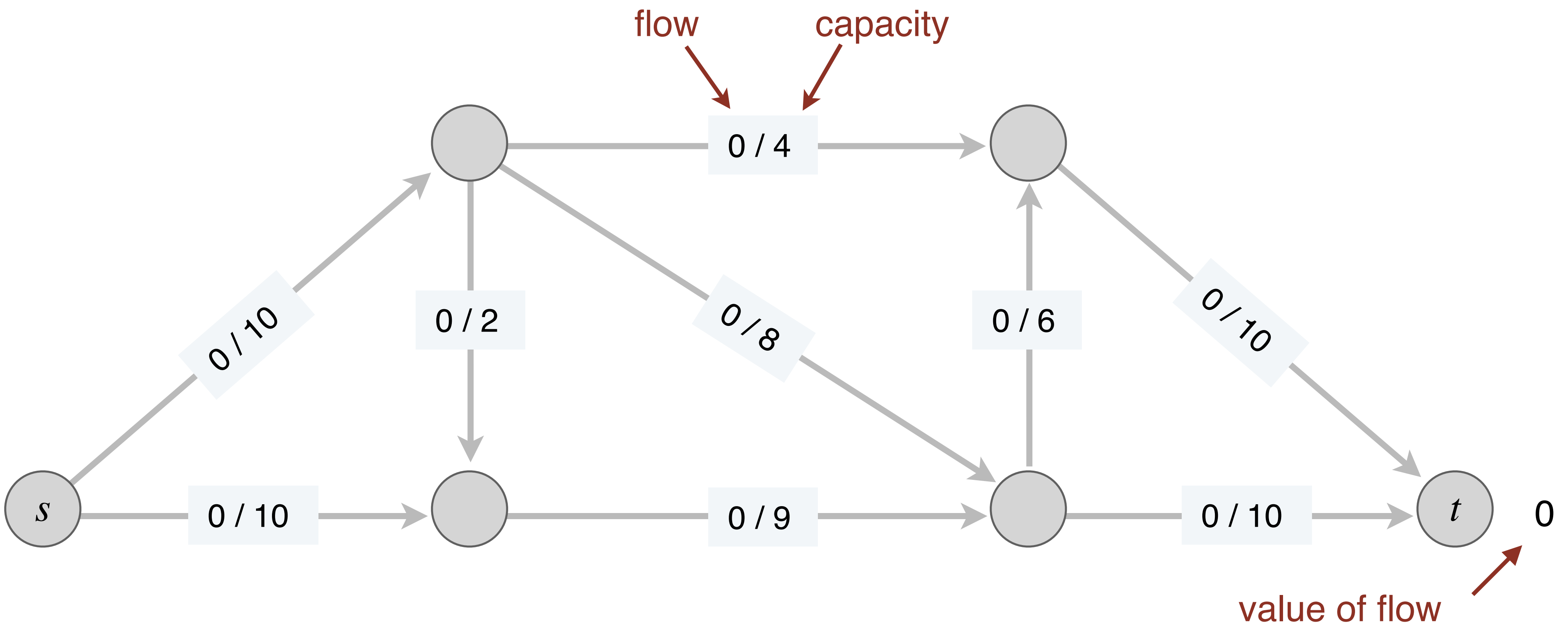
Greedy strategy:

- 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

- Let's explore an example

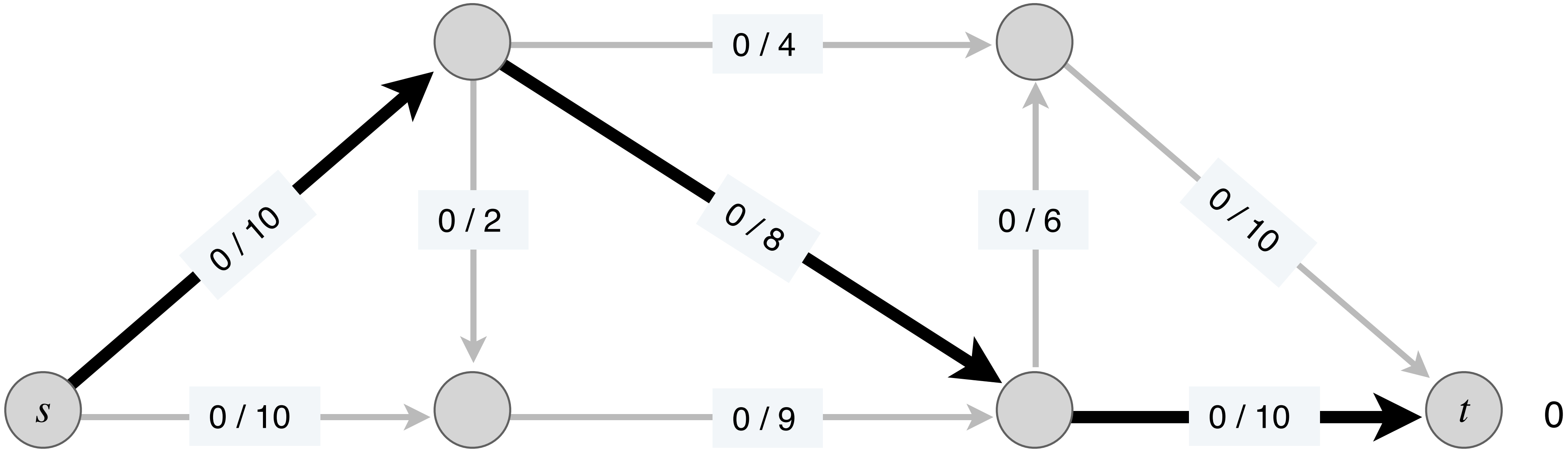
Towards a Max-Flow Algorithm

- 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



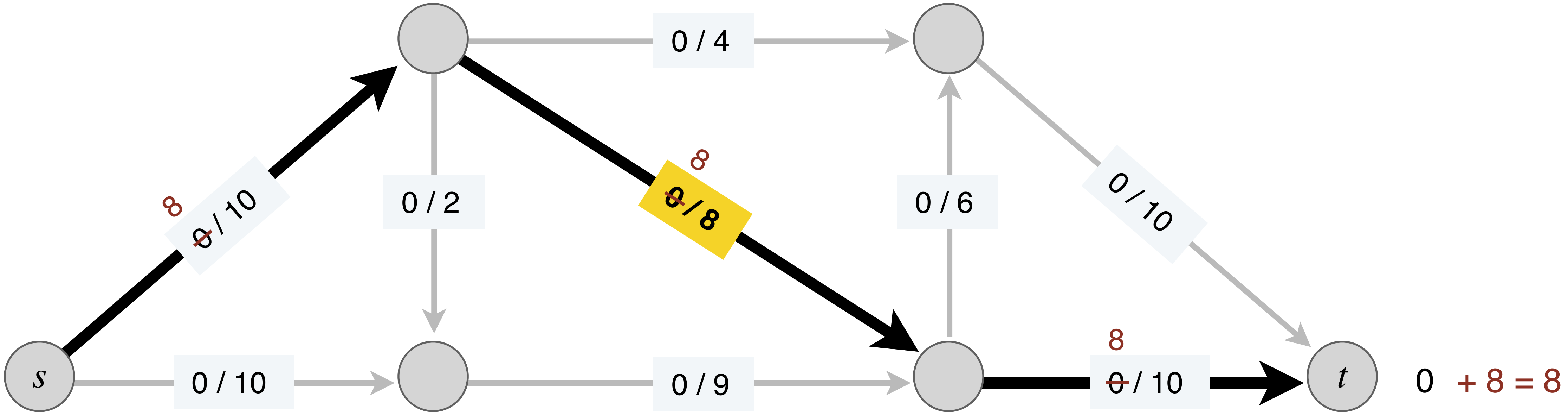
Towards a Max-Flow Algorithm

- 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



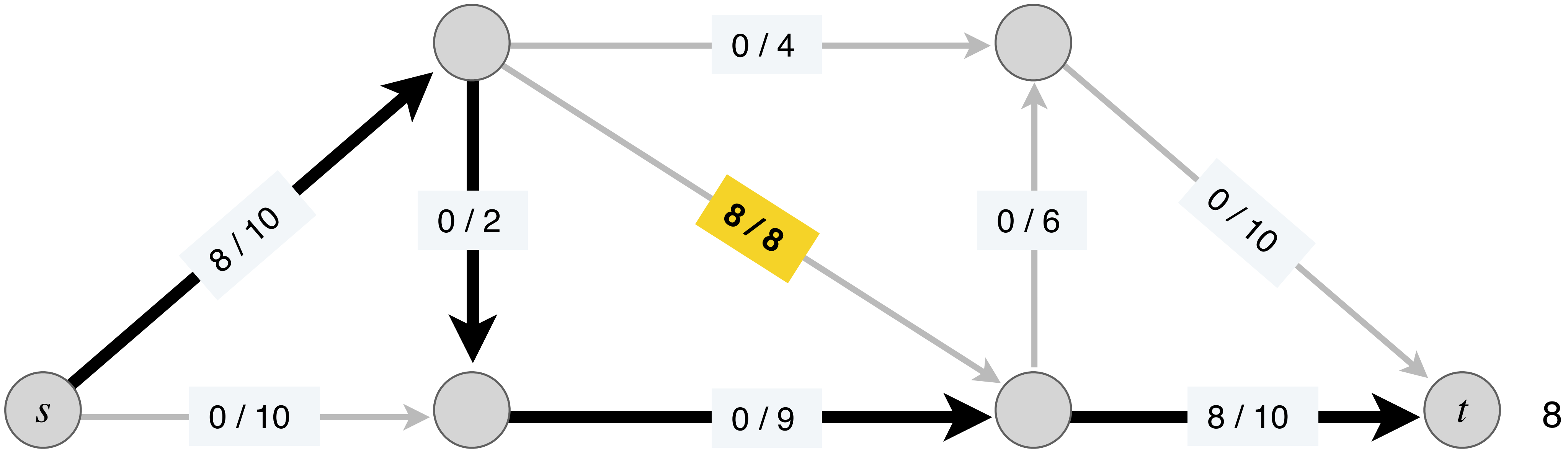
Towards a Max-Flow Algorithm

- 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



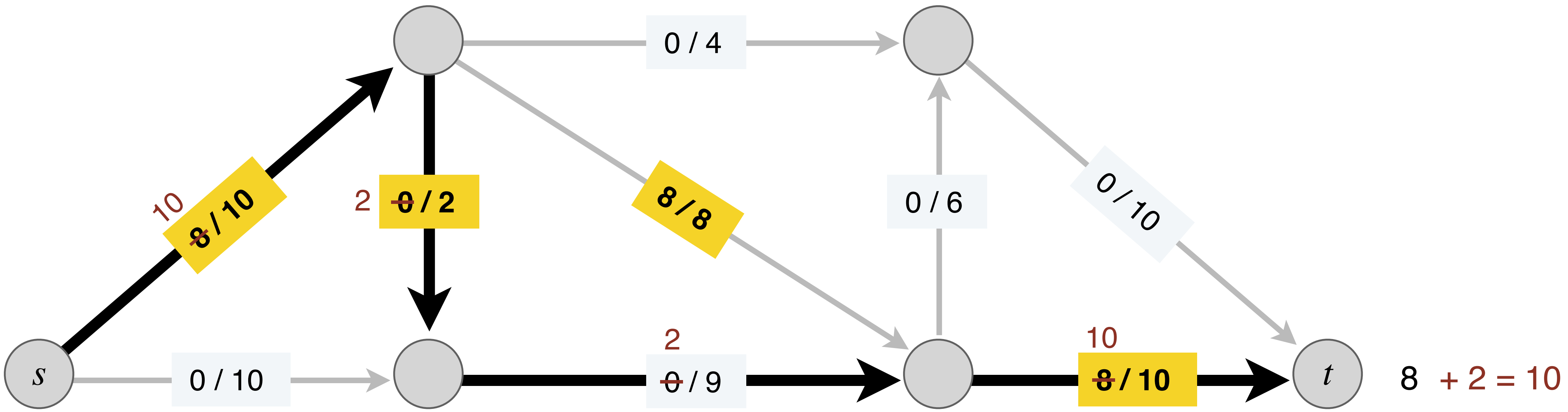
Towards a Max-Flow Algorithm

- 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



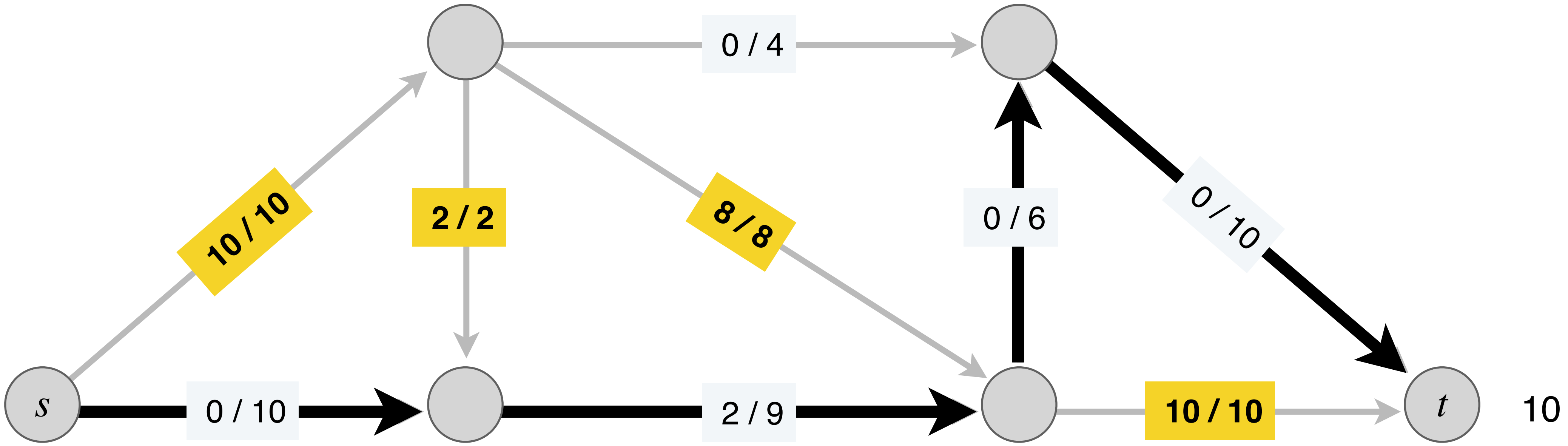
Towards a Max-Flow Algorithm

- 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

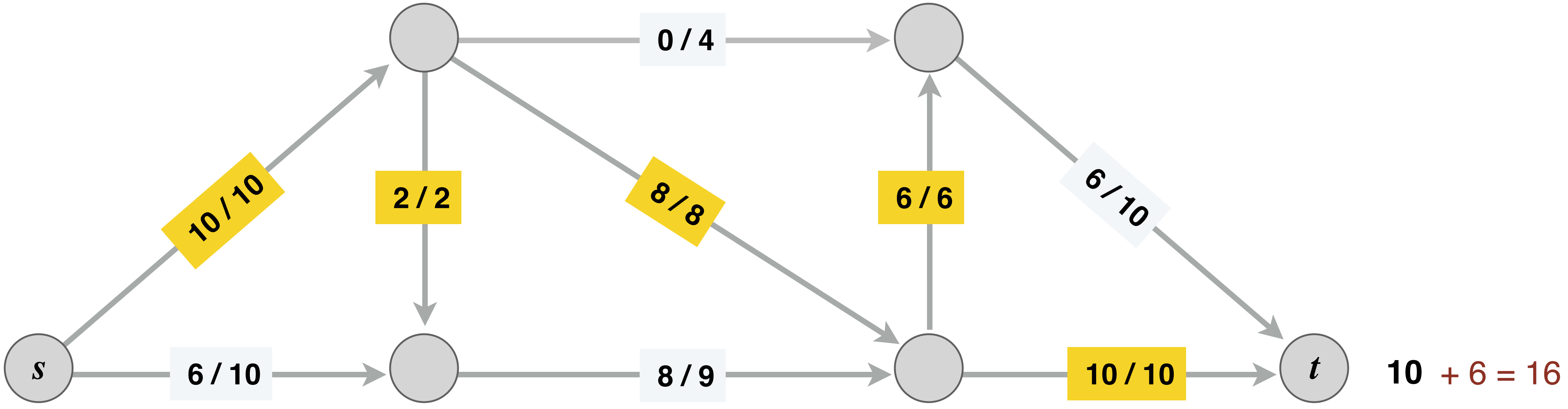


Towards a Max-Flow Algorithm

- 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?

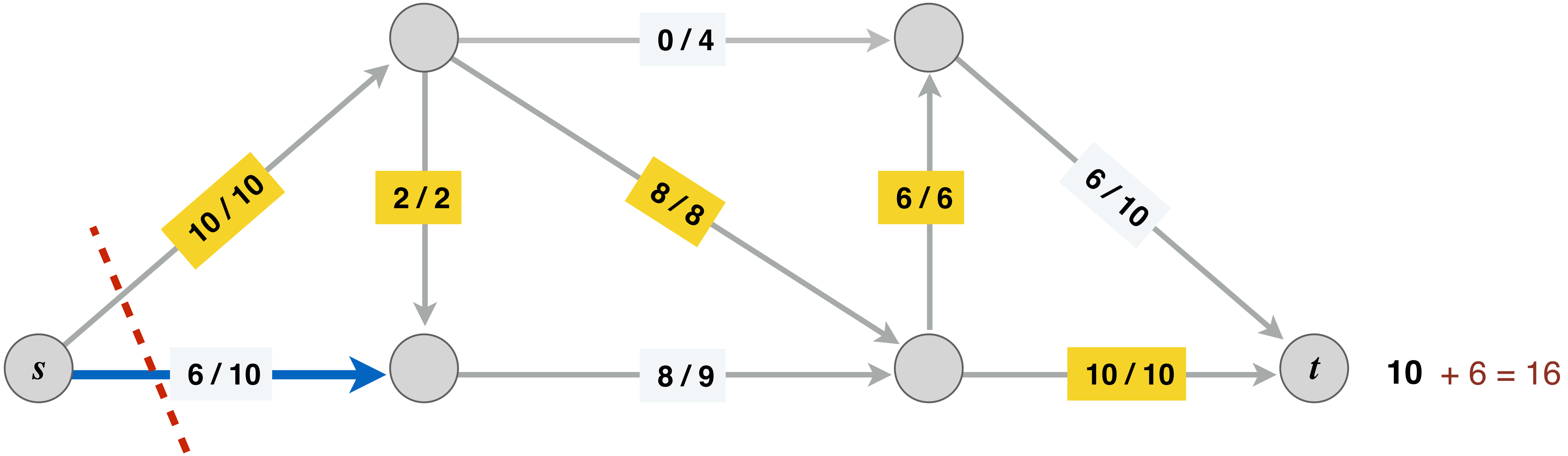
ending flow value = 16



Towards a Max-Flow Algorithm

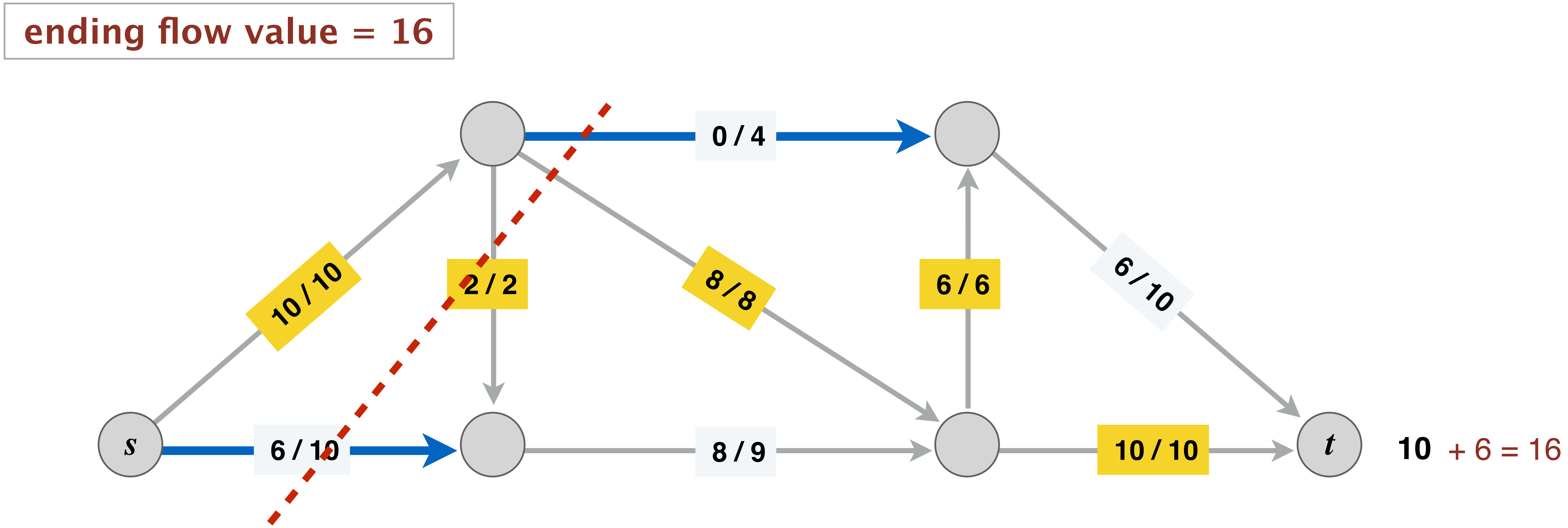
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

ending flow value = 16



Towards a Max-Flow Algorithm

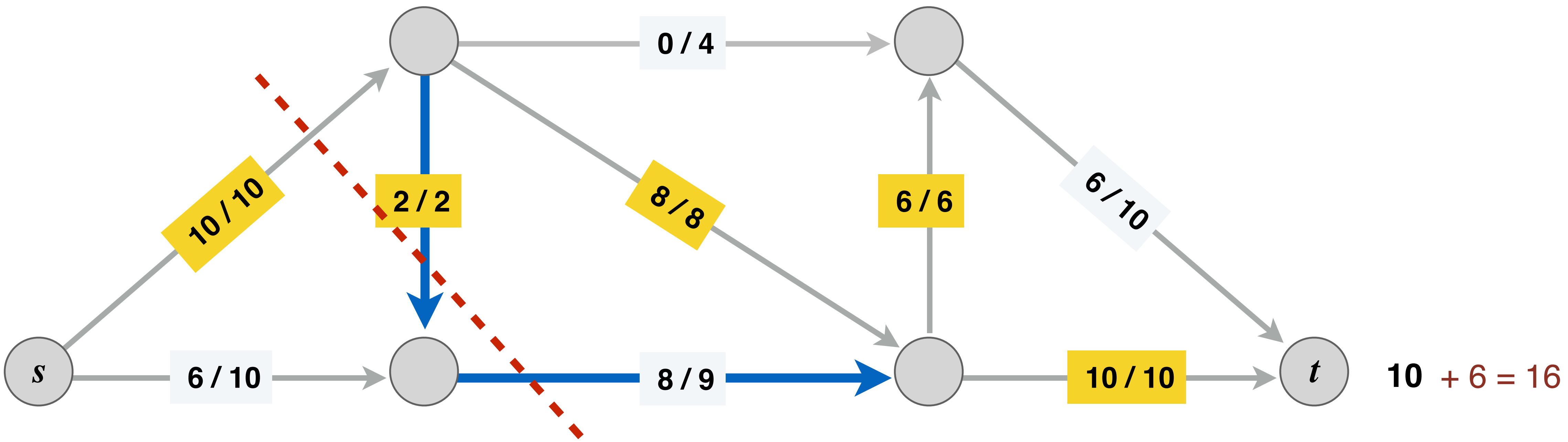
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



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

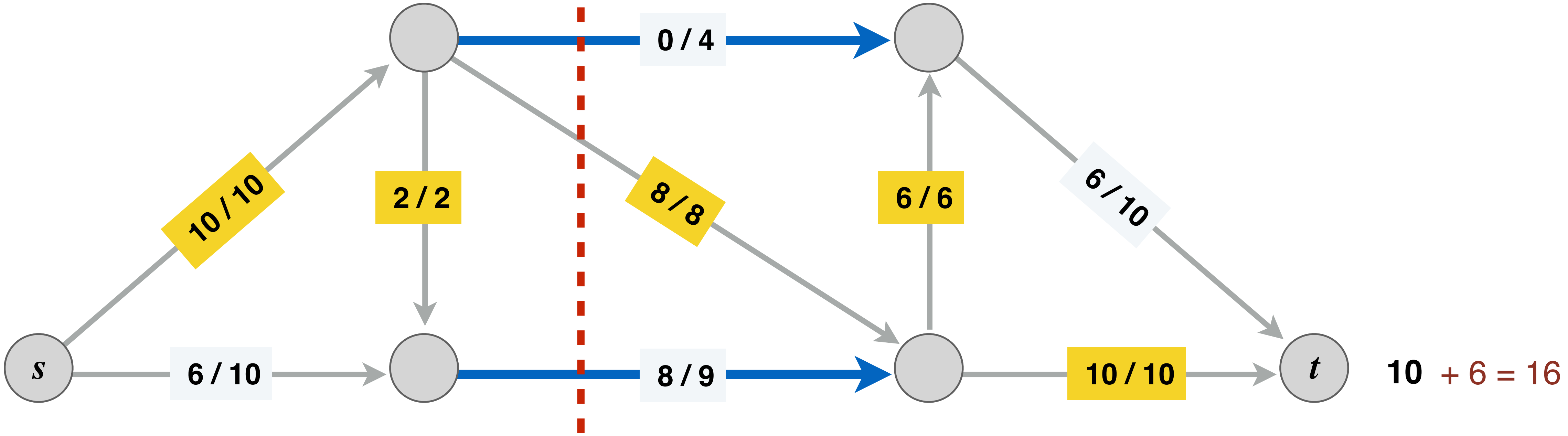
ending flow value = 16



Towards a Max-Flow Algorithm

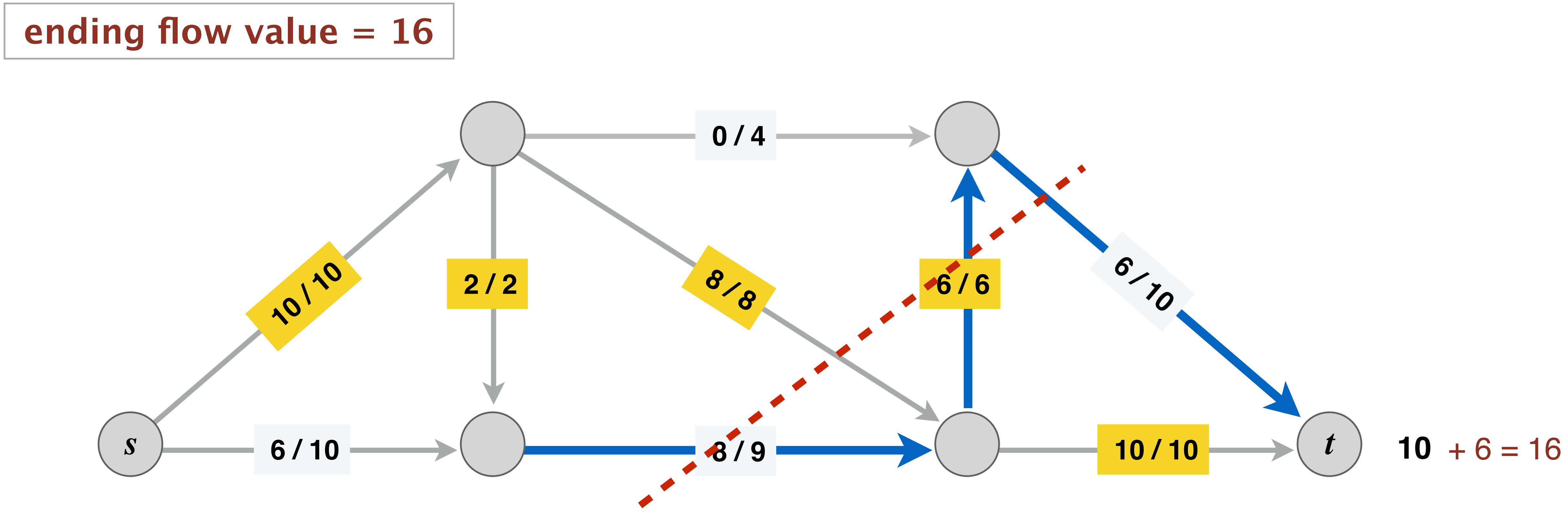
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

ending flow value = 16



Towards a Max-Flow Algorithm

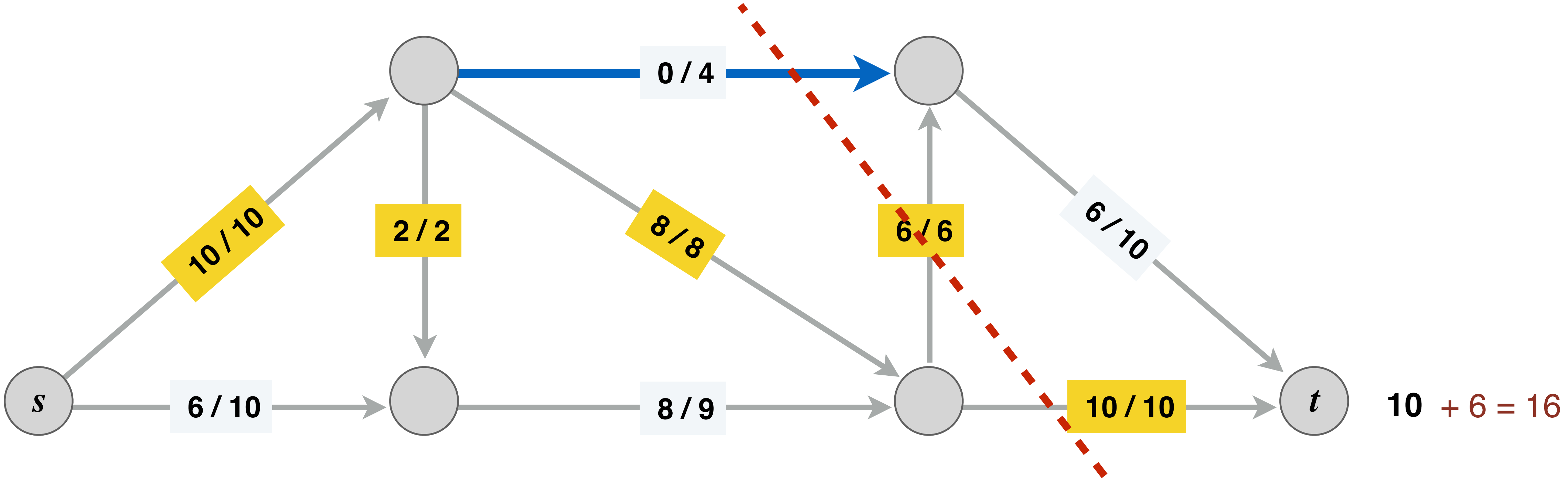
- Start with $f(e) = 0$ for each edge
- Find an $s \leadsto t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Towards a Max-Flow Algorithm

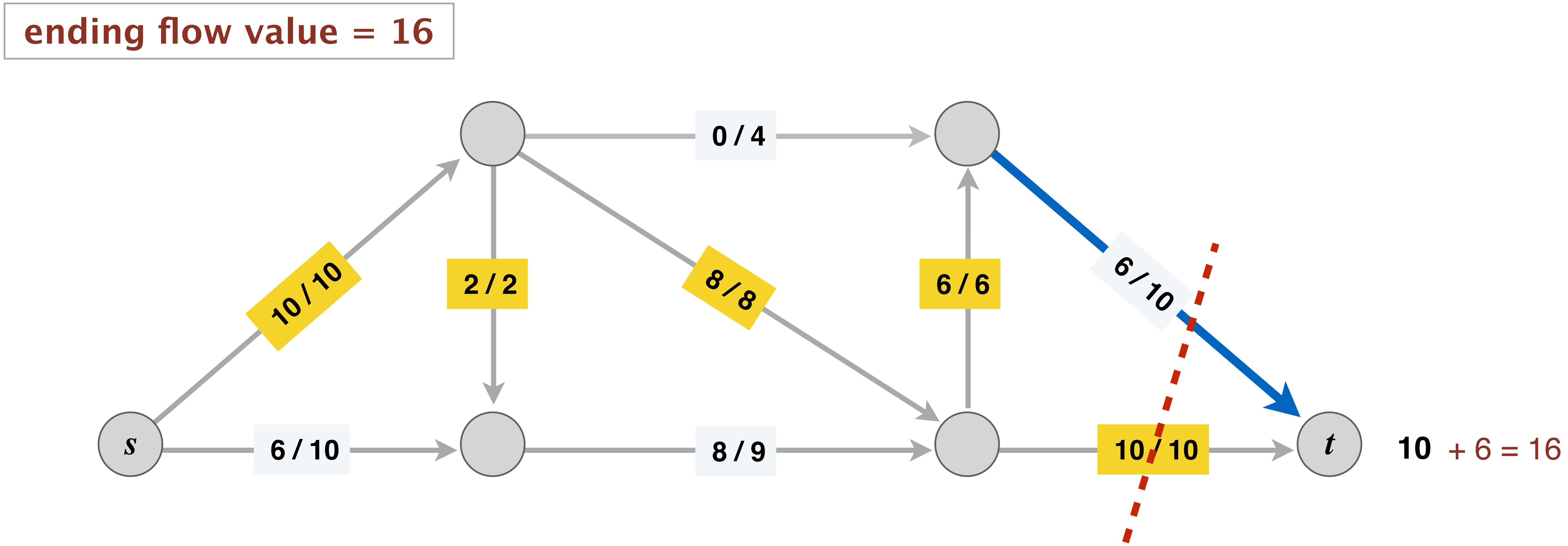
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

ending flow value = 16



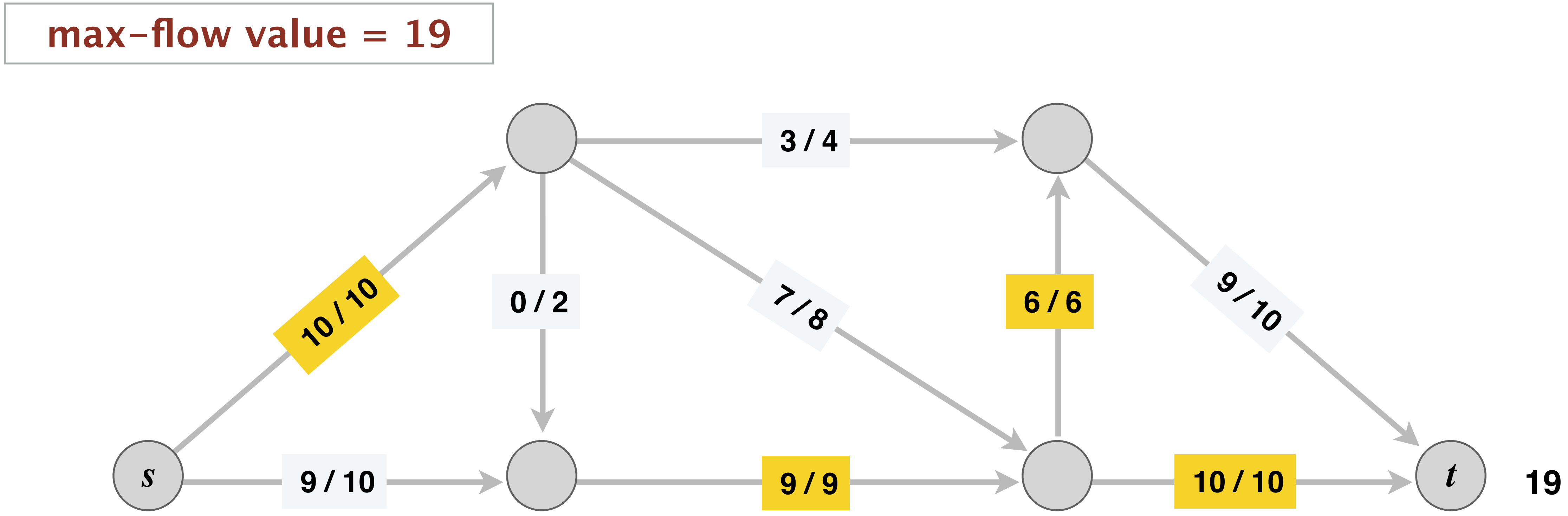
Towards a Max-Flow Algorithm

- 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



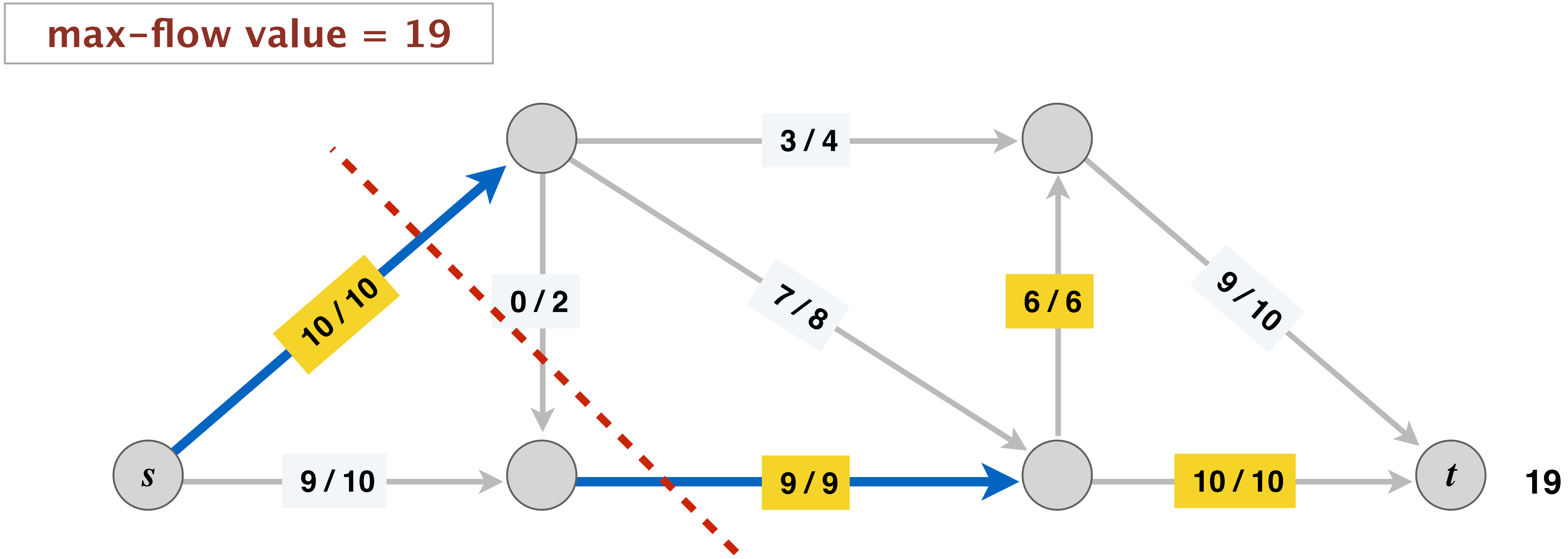
Towards a Max-Flow Algorithm

- 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

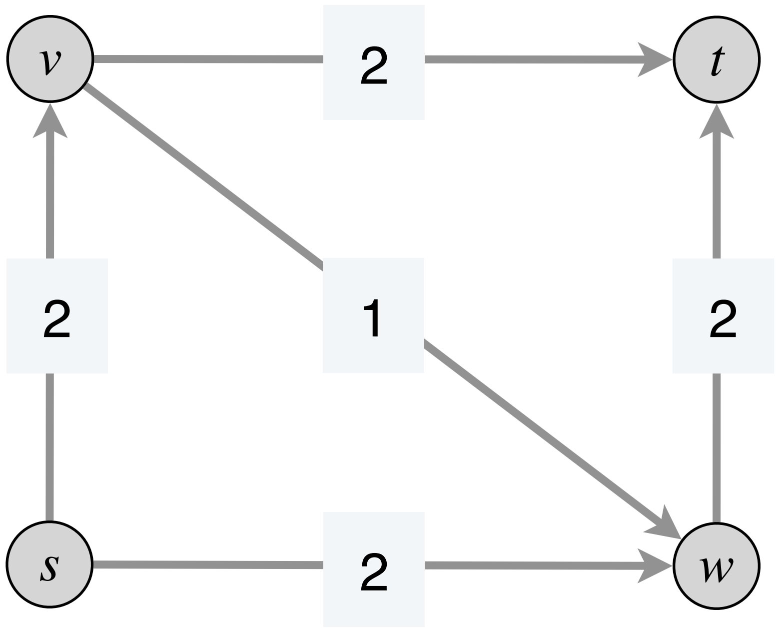
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



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

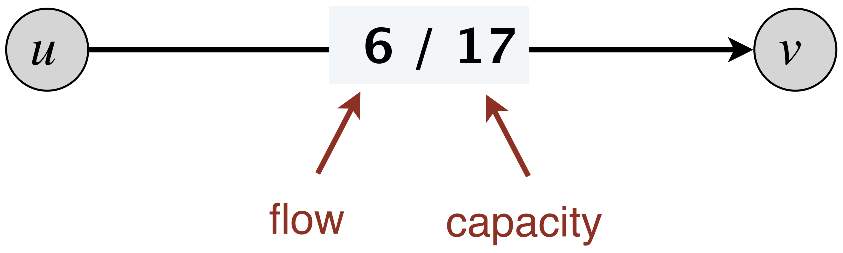
Given flow network $G = (V, E, c)$ and a feasible flow f on G, the **residual graph** $G_f = (V, E_f, c_f)$ is defined as follows:

- Vertices in G_f are the same as in 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 $f(e)$

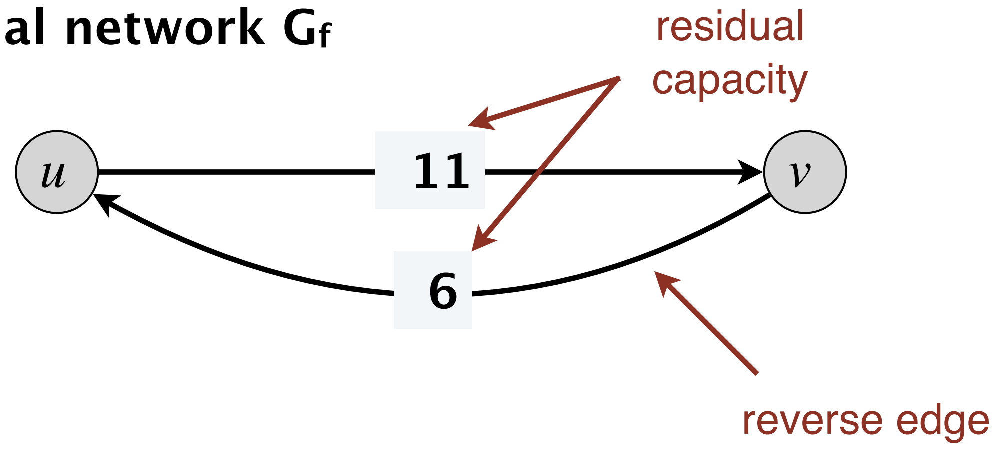
"unused" or "remaining" capacity

"used" capacity that we can undo

original flow network G



residual network Gf

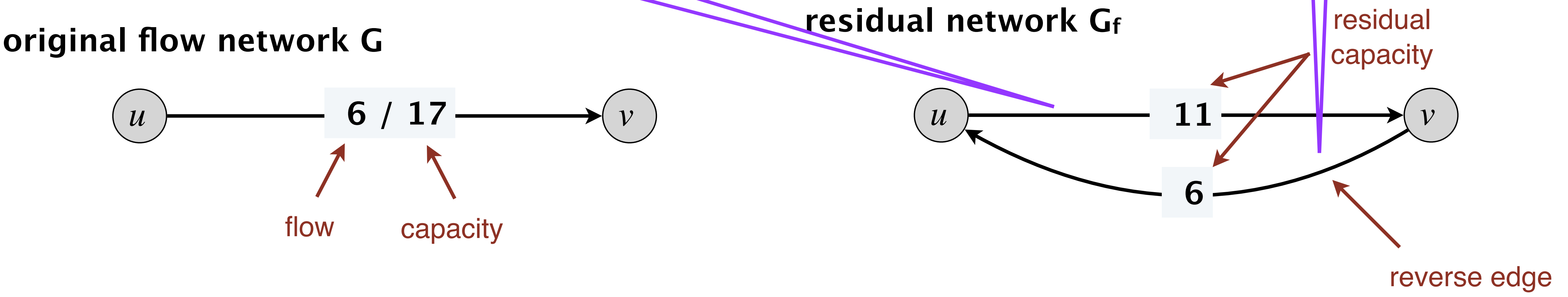


Residual Graph

- **(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 $f(e)$

What does it mean to push flow down a forward edge?

What does it mean to push flow down a reverse edge?



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
  - e.g., we find a bottleneck capacity edge on the path and push that much flow through it in G_f
- How do we translate this back to G?
  - 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
- The **bottleneck capacity** b of an augmenting path P is the minimum capacity of any edge in P.

Path that repeats no vertices

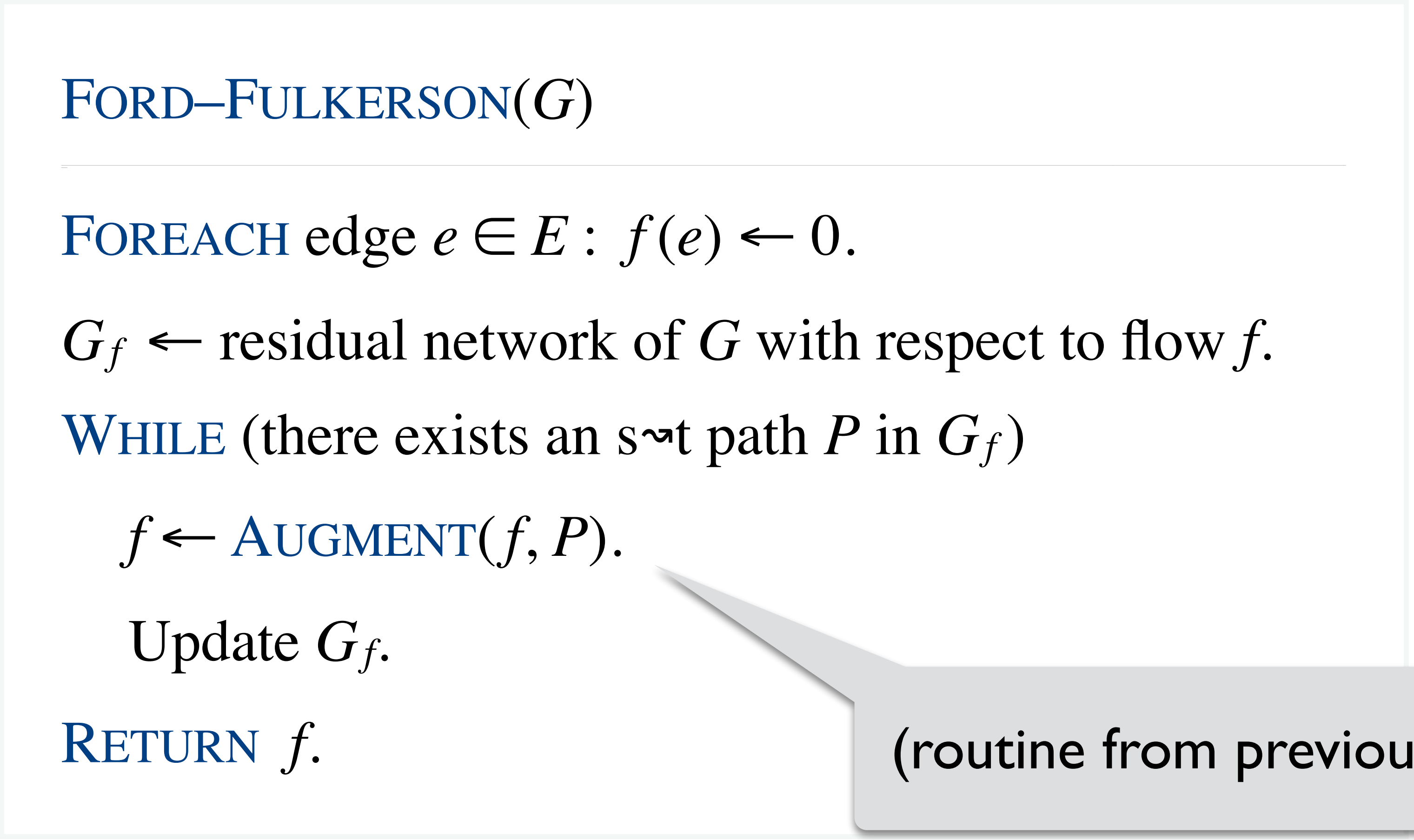
Some $s \rightsquigarrow t$ path P in G_f

```
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 updates 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



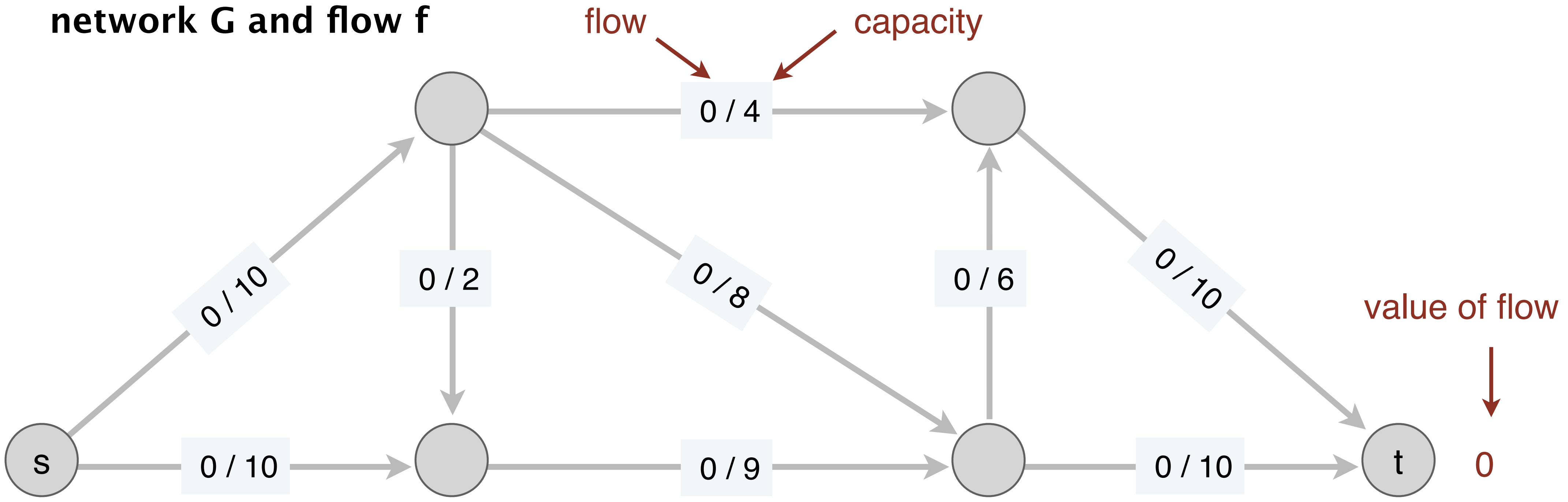
```
FORD–FULKERSON($G$)

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 $f$.
```

(routine from previous slide)

Ford-Fulkerson Example

network G and flow f

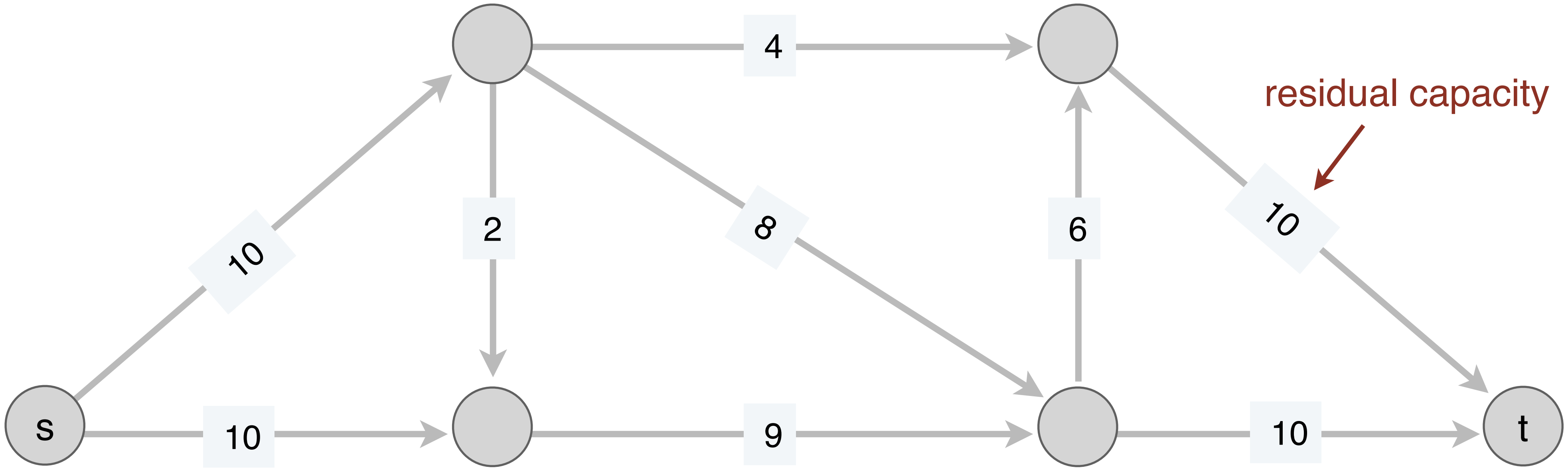

flow

capacity

value of flow

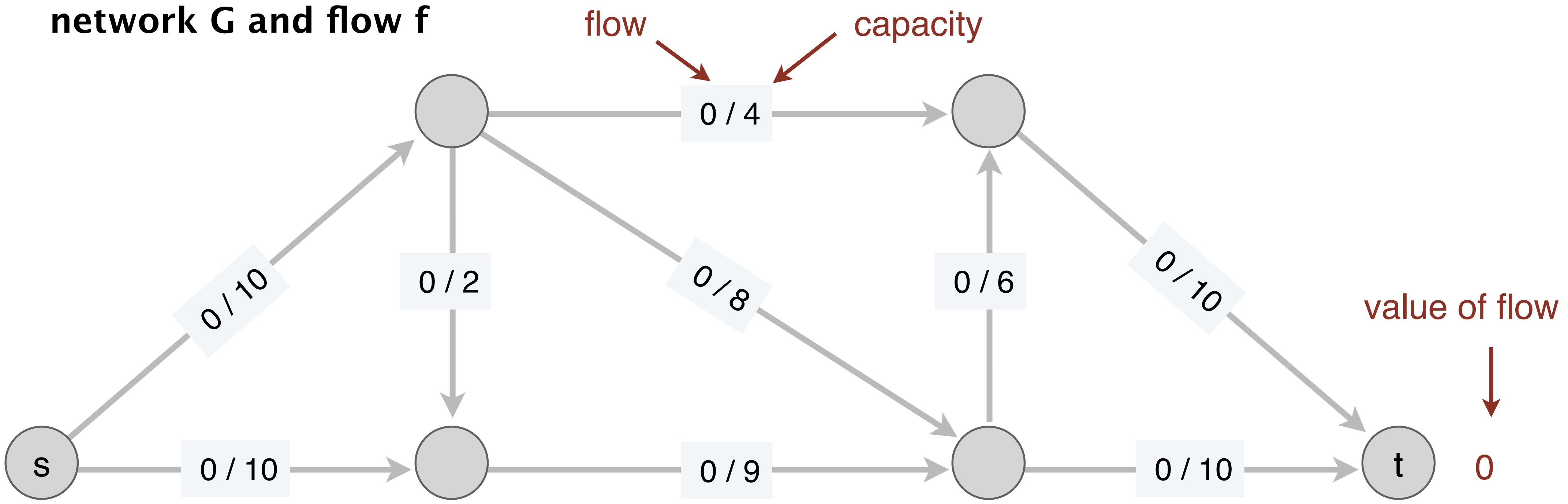
0

residual network G$_f$

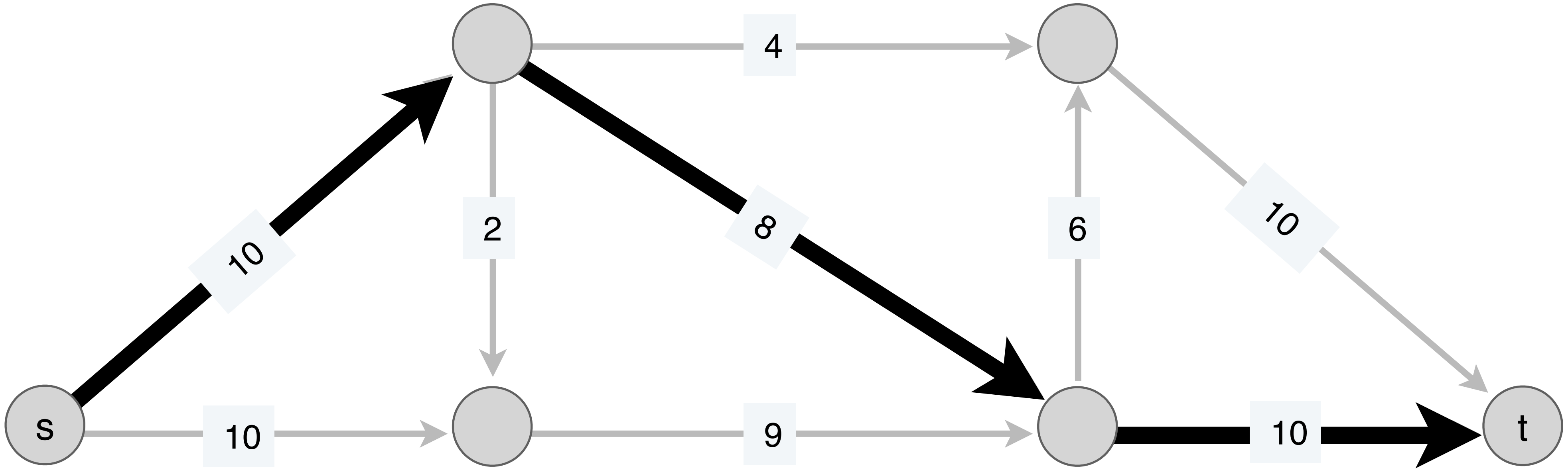

residual capacity

Ford-Fulkerson Example

network G and flow f

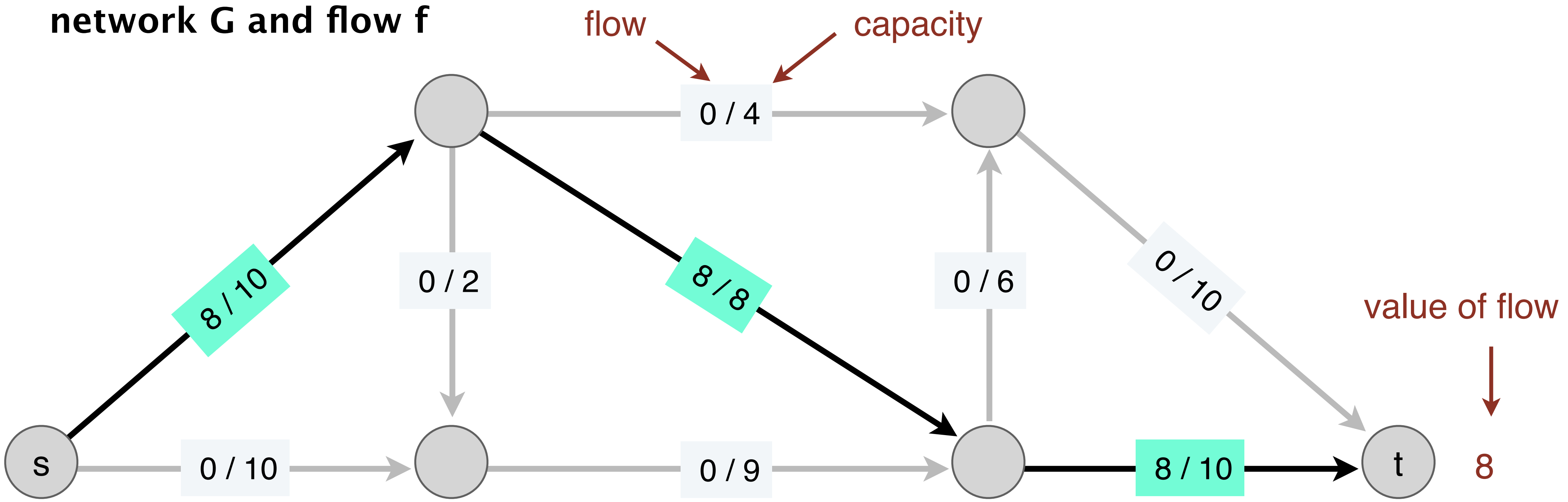


P in residual network G_f

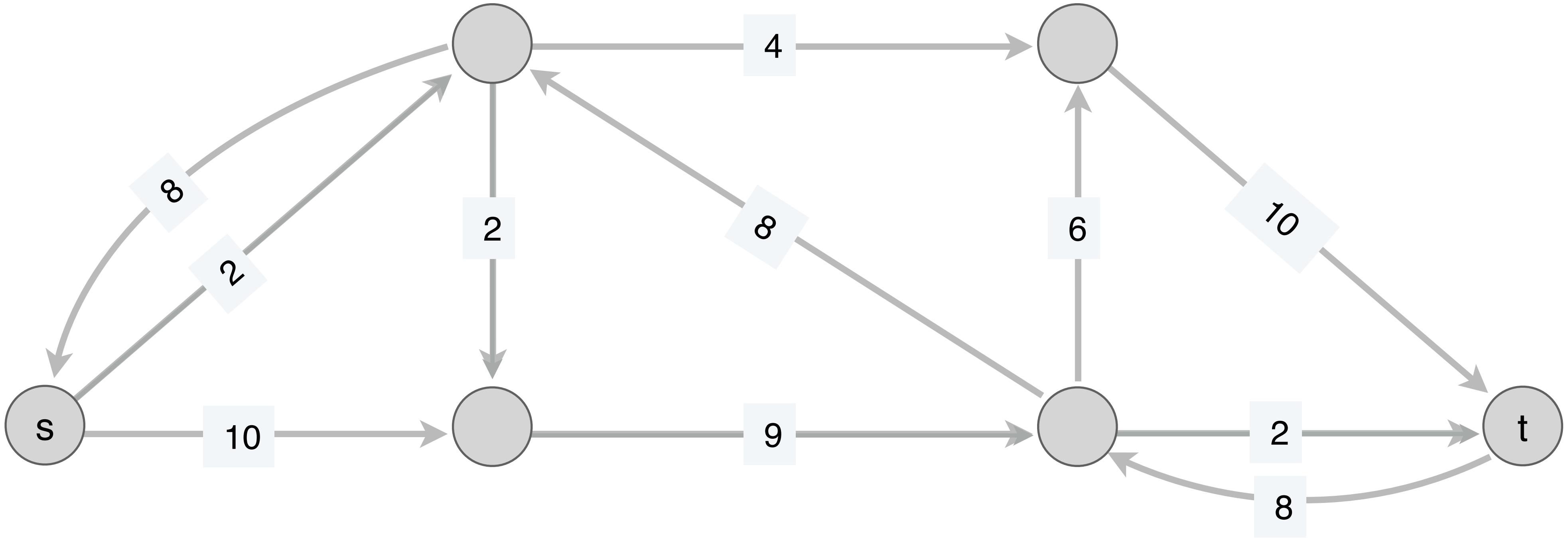


Ford-Fulkerson Example

network G and flow f

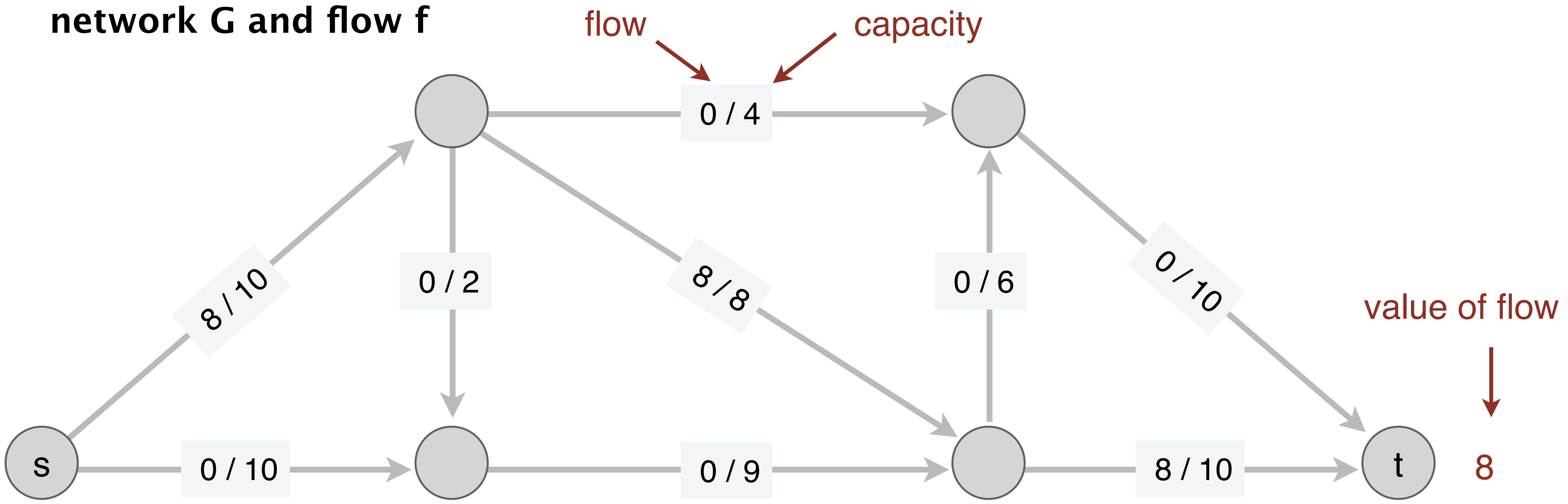


residual network G_f

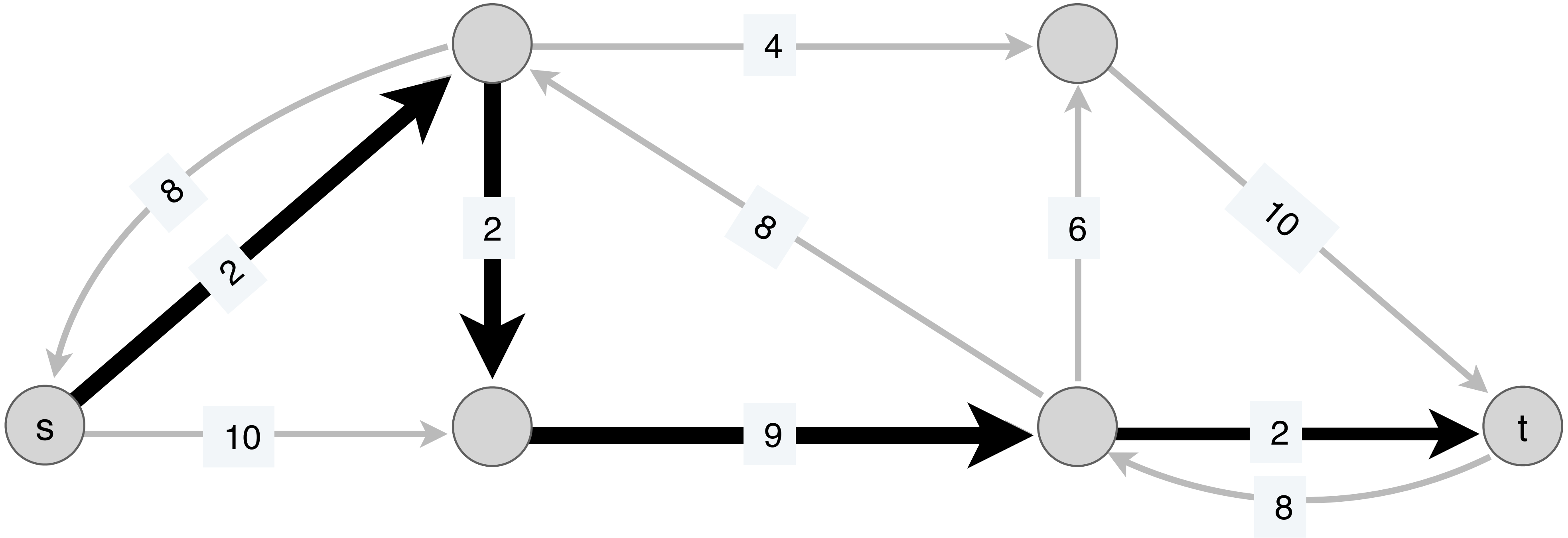


Ford-Fulkerson Example

network G and flow f

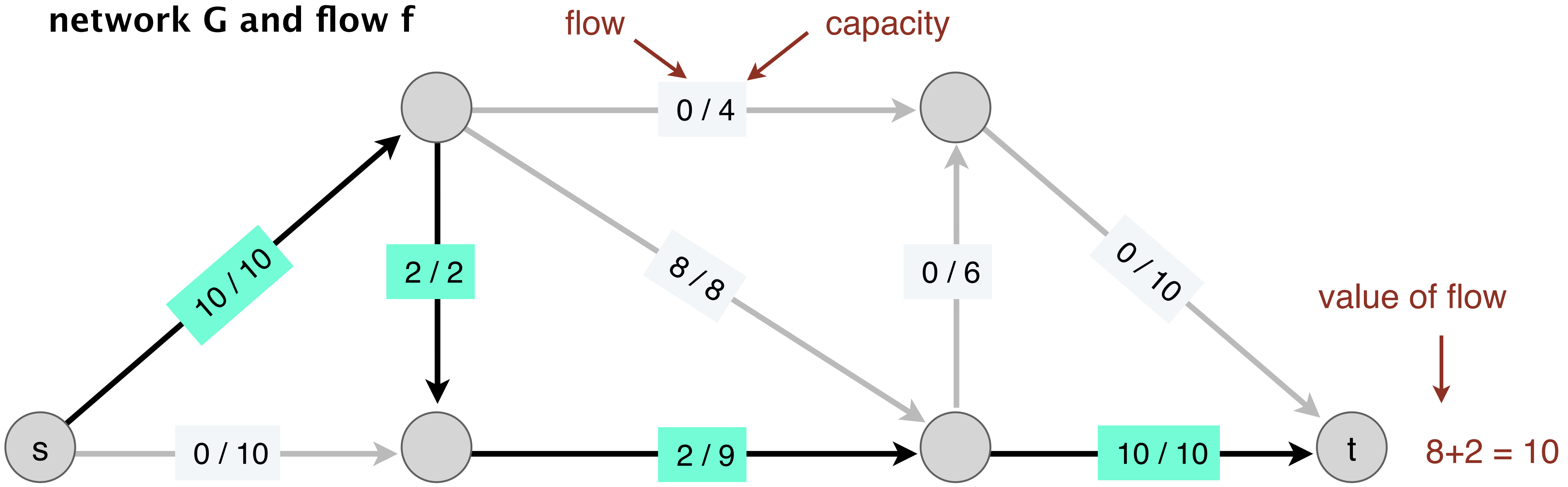


P in residual network G_f

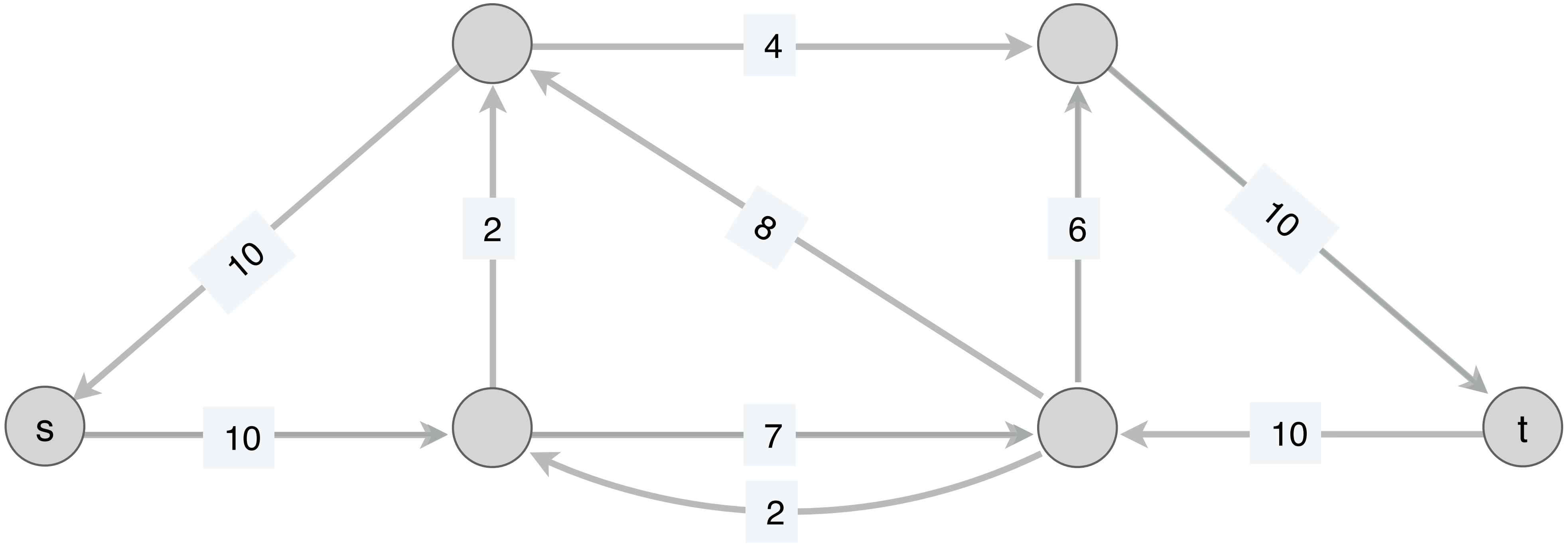


Ford-Fulkerson Example

network G and flow f

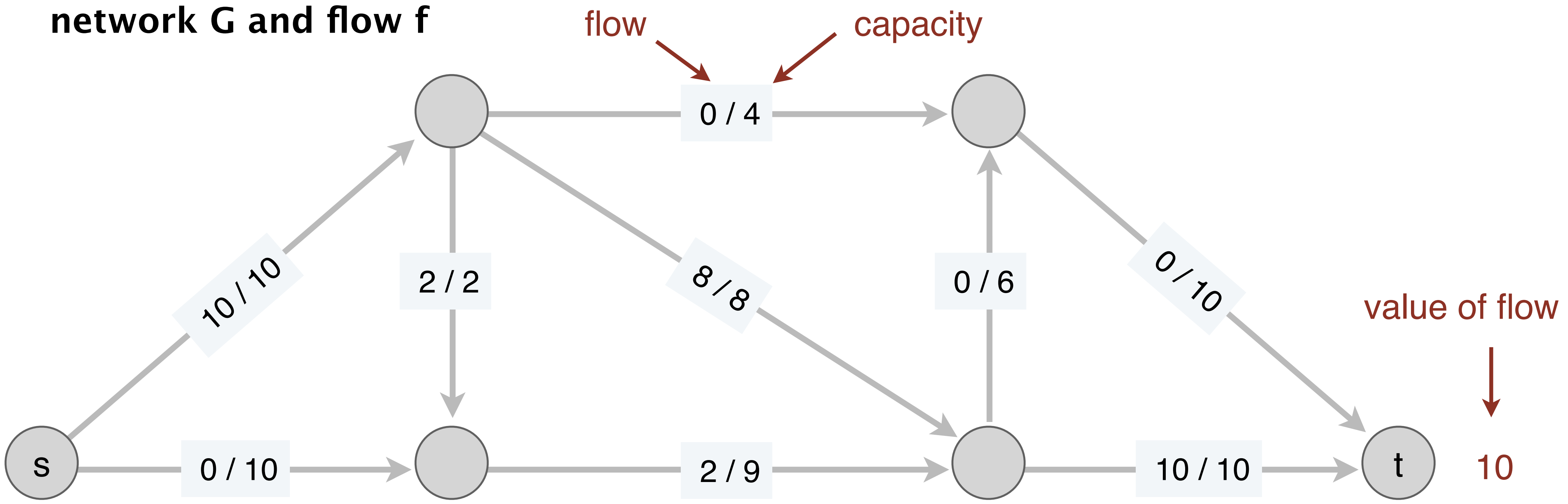


residual network Gf

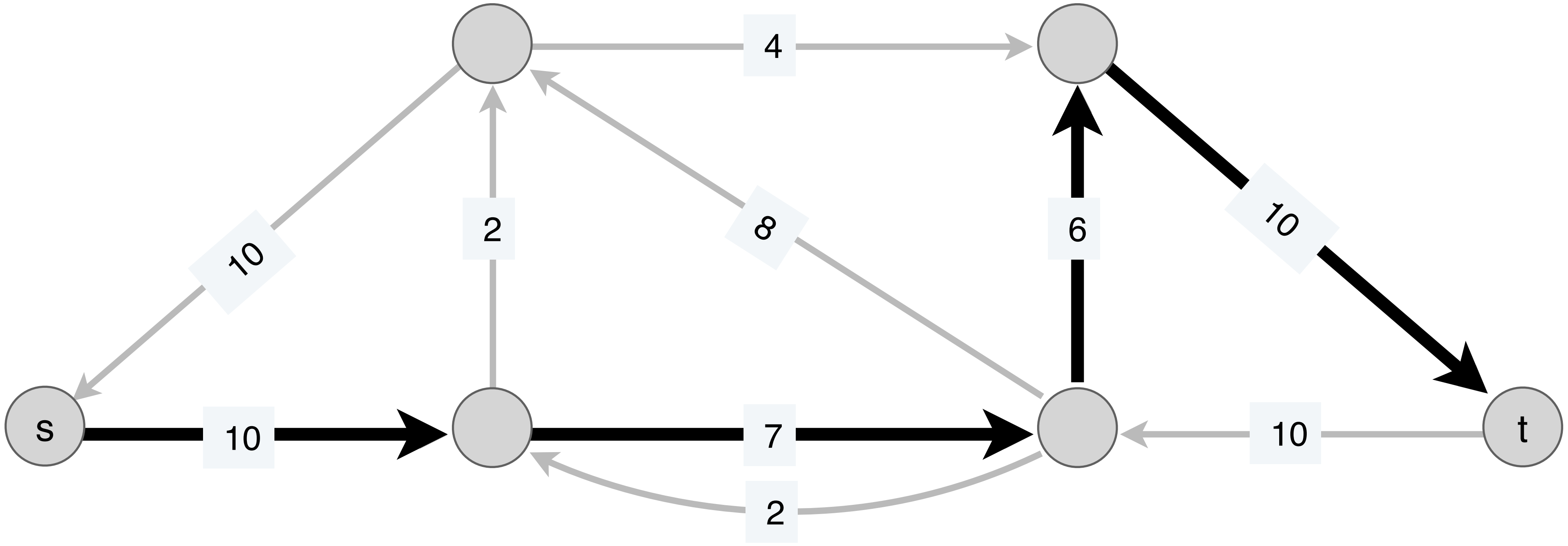


Ford-Fulkerson Example

network G and flow f

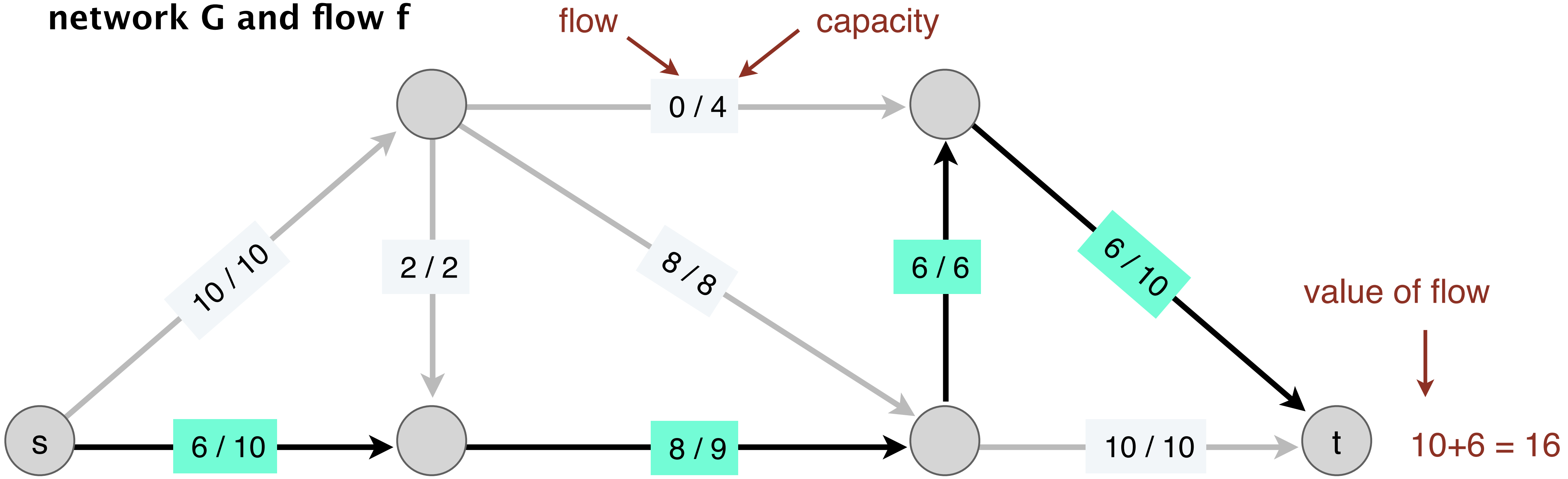


P in residual network G_f

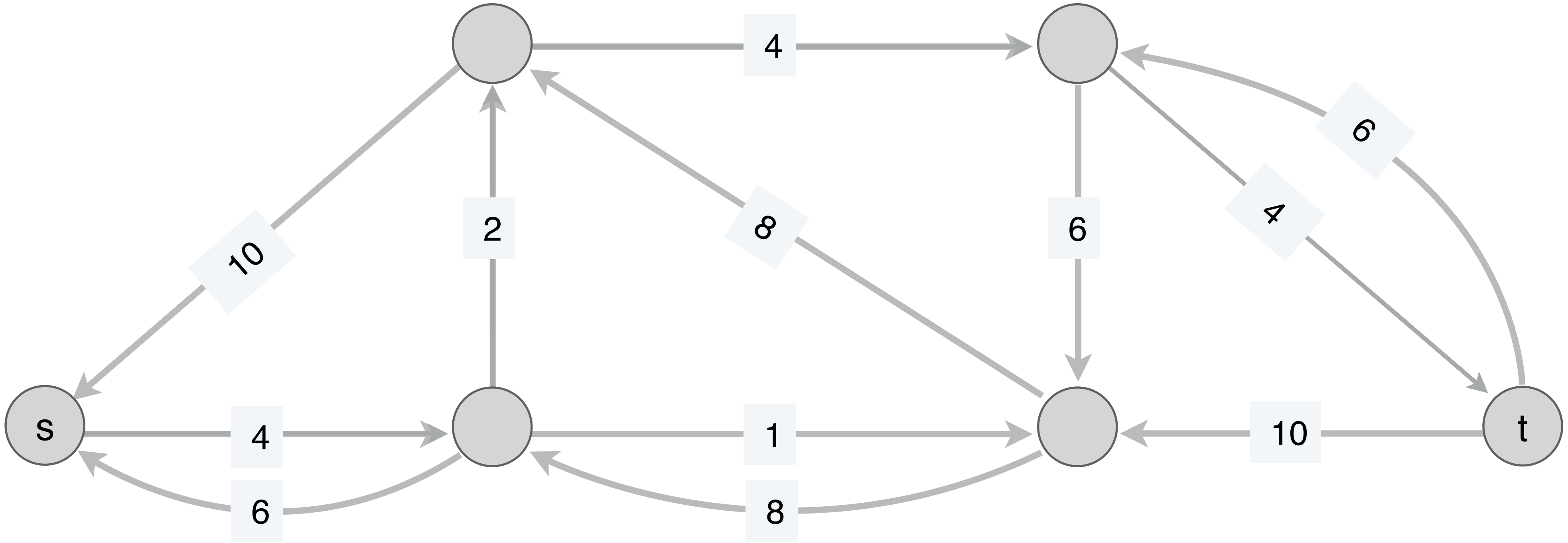


Ford-Fulkerson Example

network G and flow f

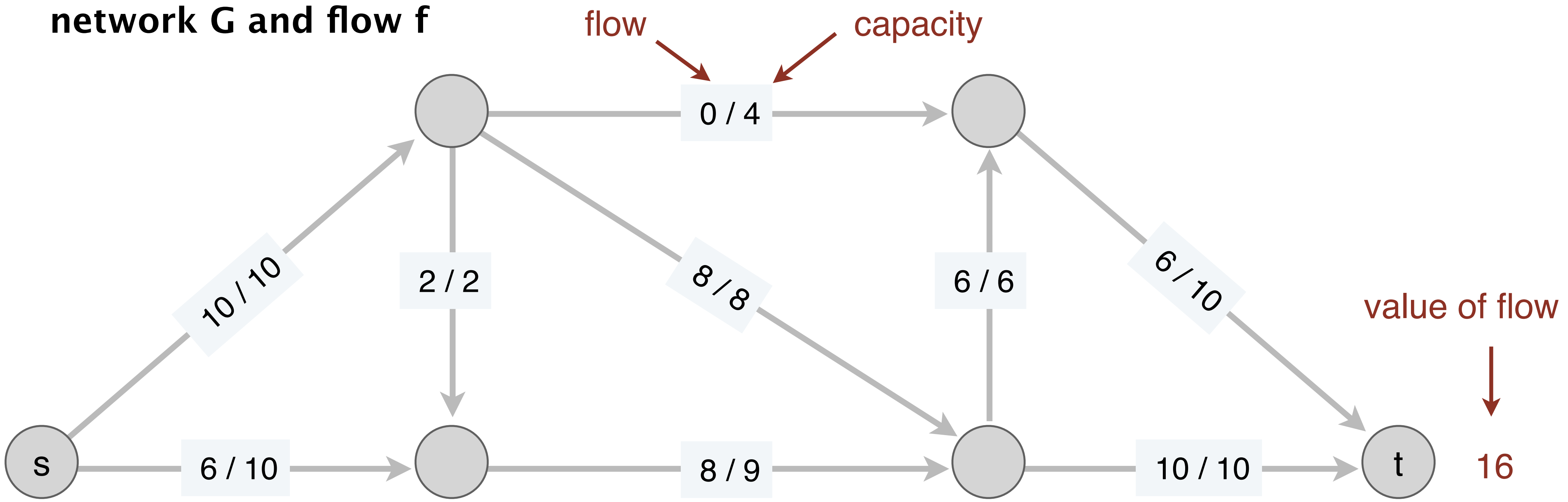


residual network G_f



Ford-Fulkerson Example

network G and flow f



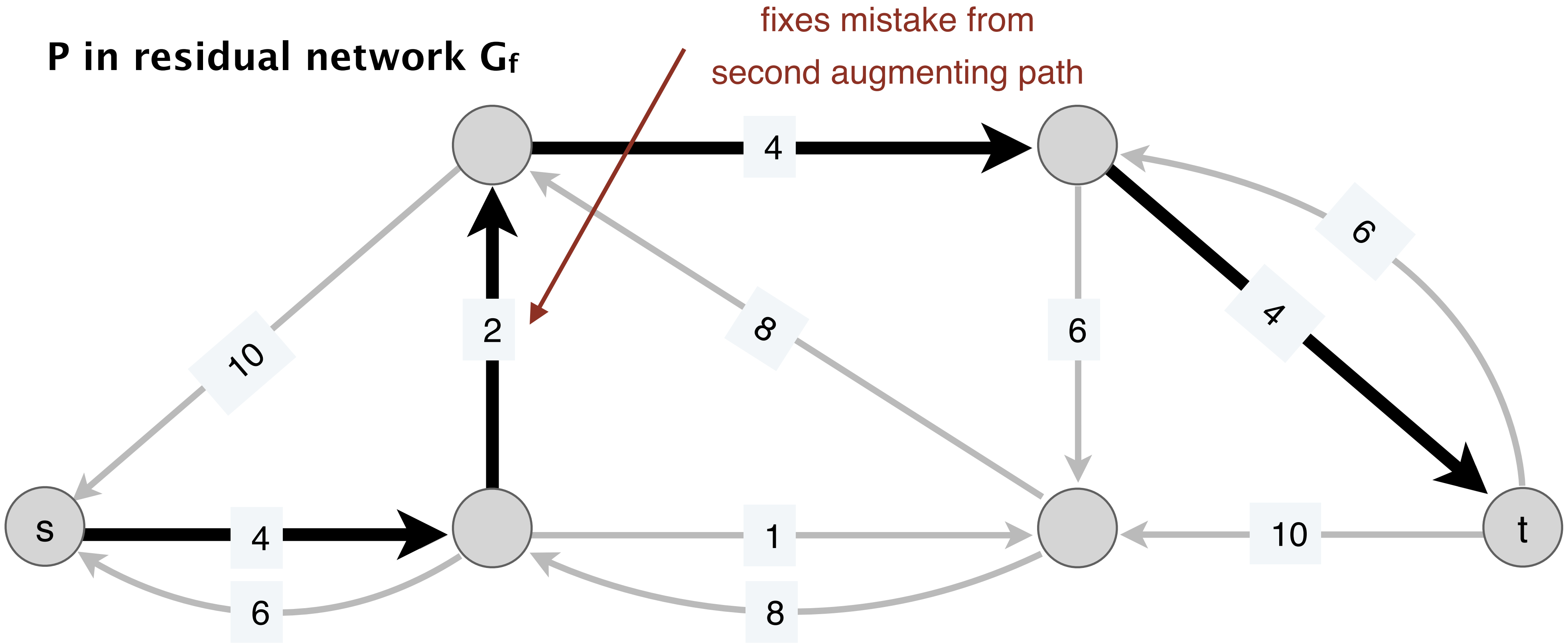
flow

capacity

value of flow

16

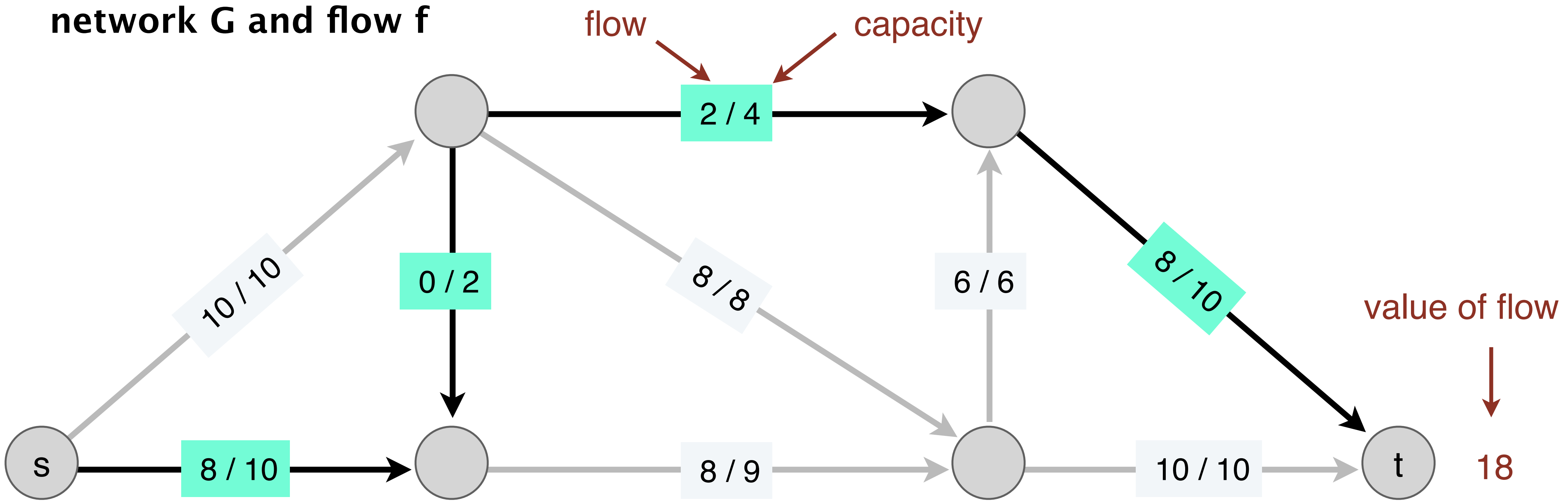
P in residual network G_f



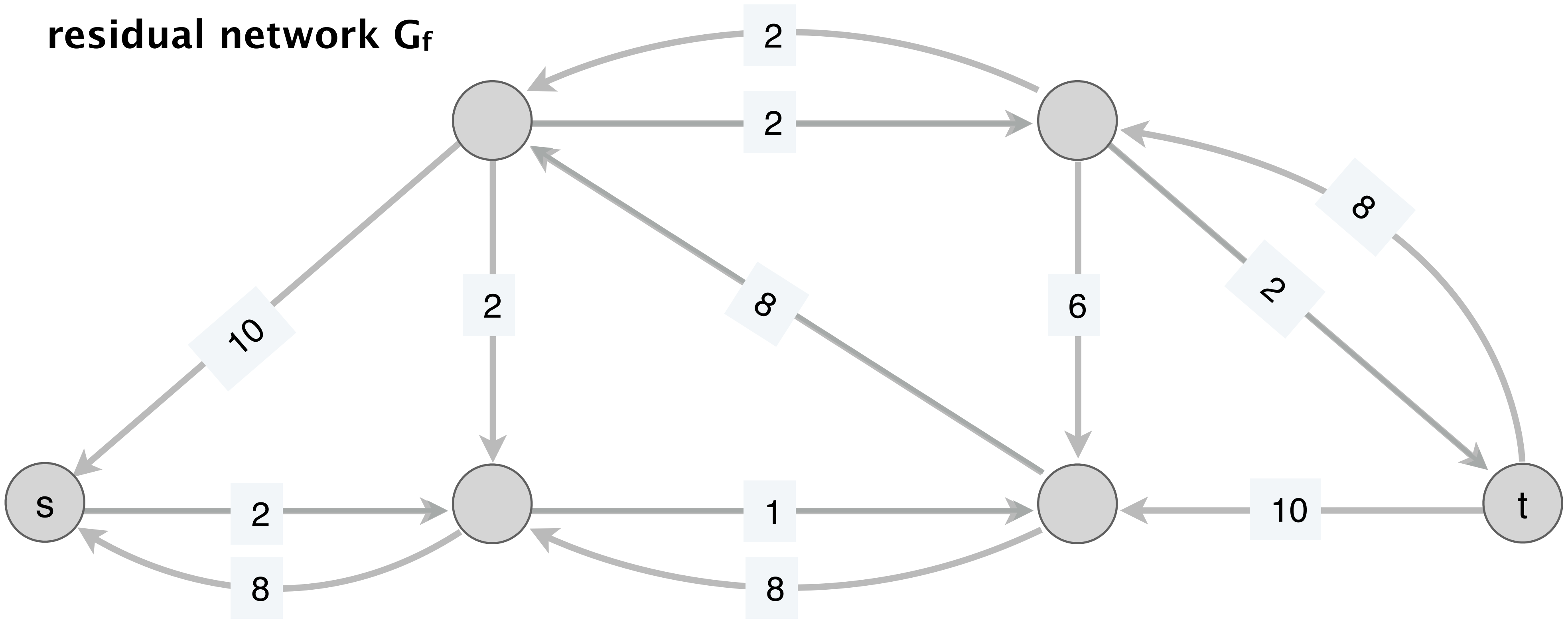
fixes mistake from second augmenting path

Ford-Fulkerson Example

network G and flow f

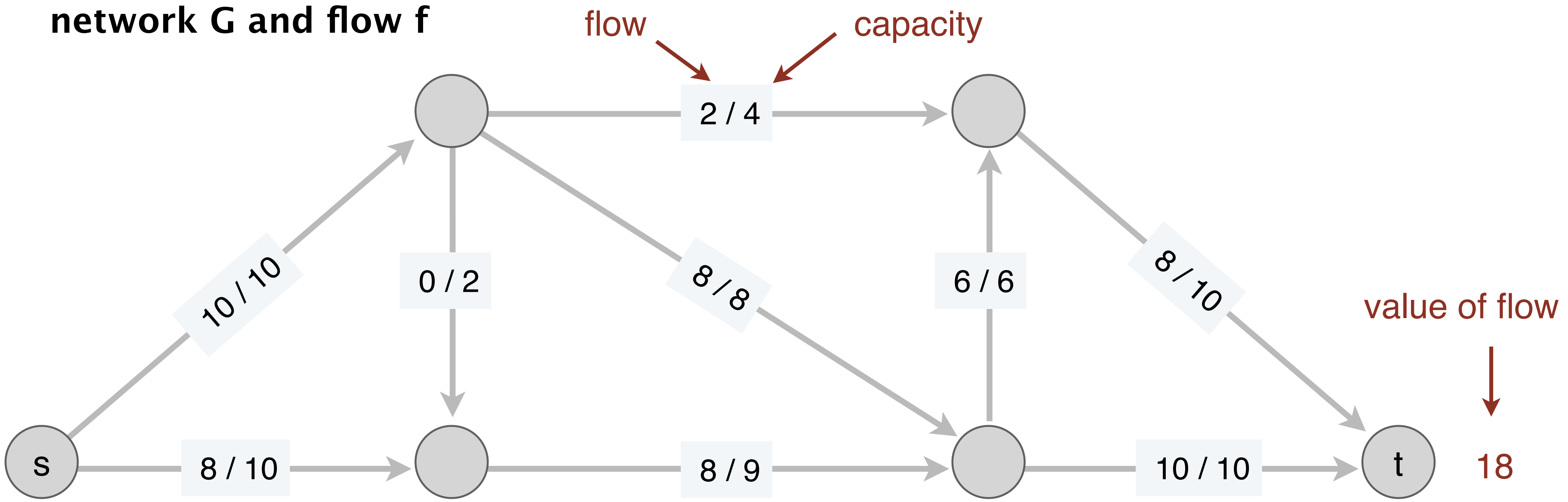


residual network G_f

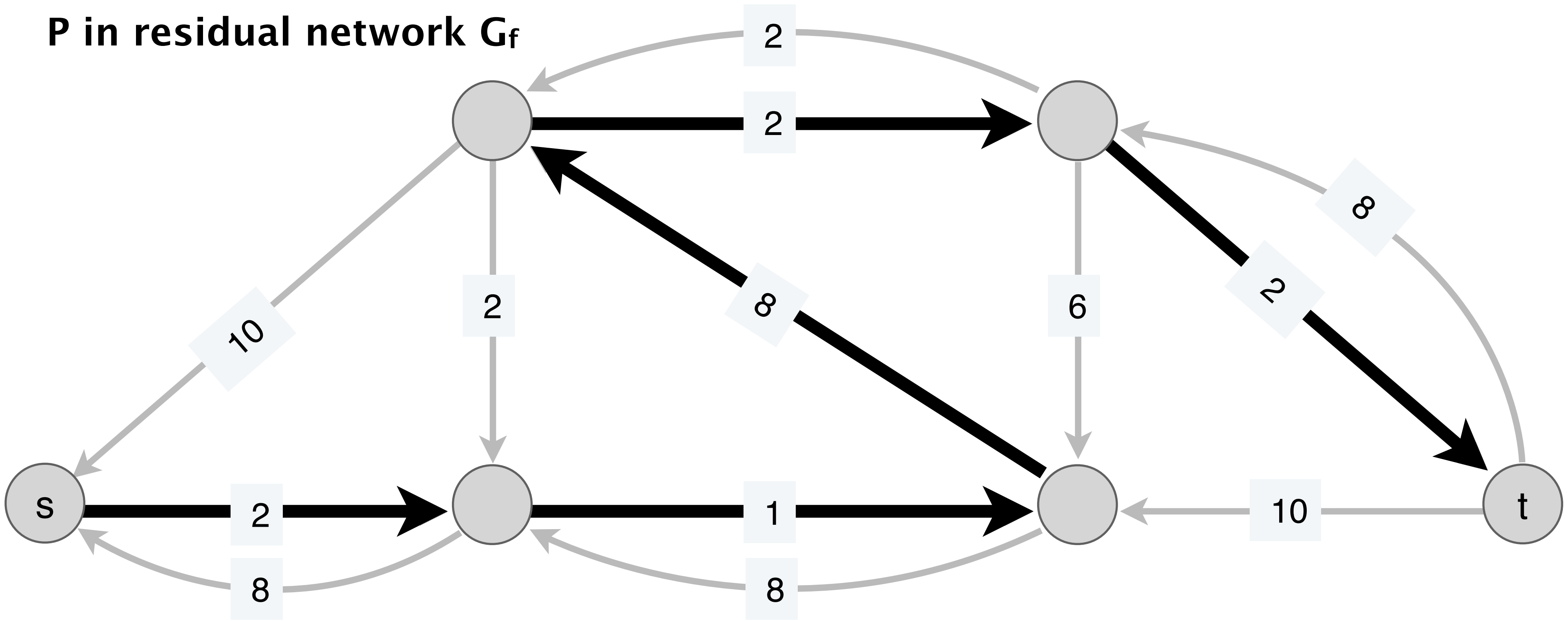


Ford-Fulkerson Example

network G and flow f

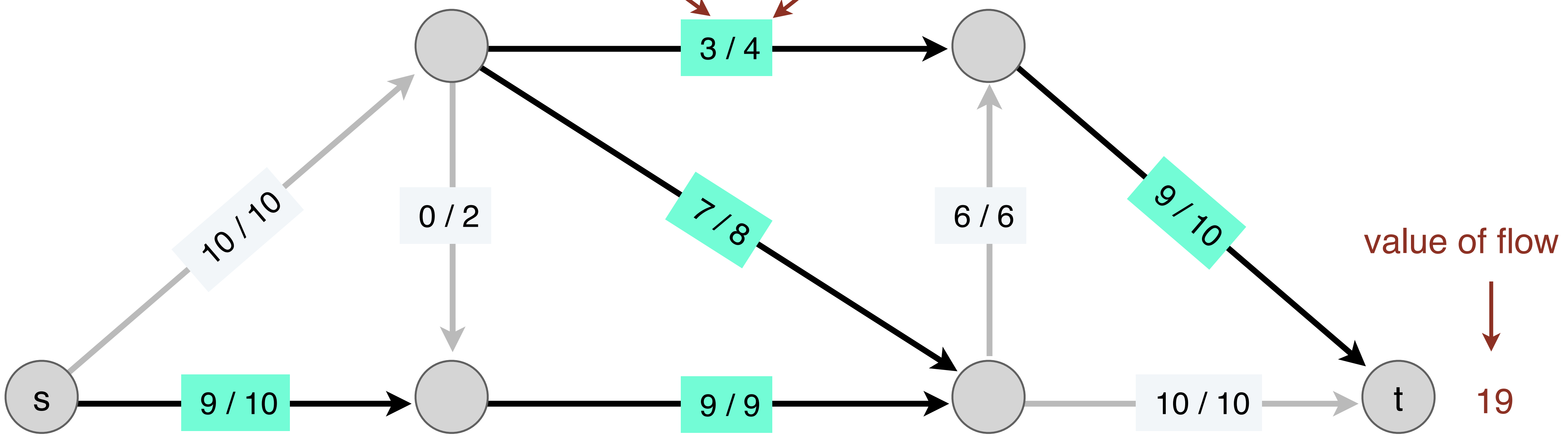


P in residual network G_f



Ford-Fulkerson Example

network G and flow f



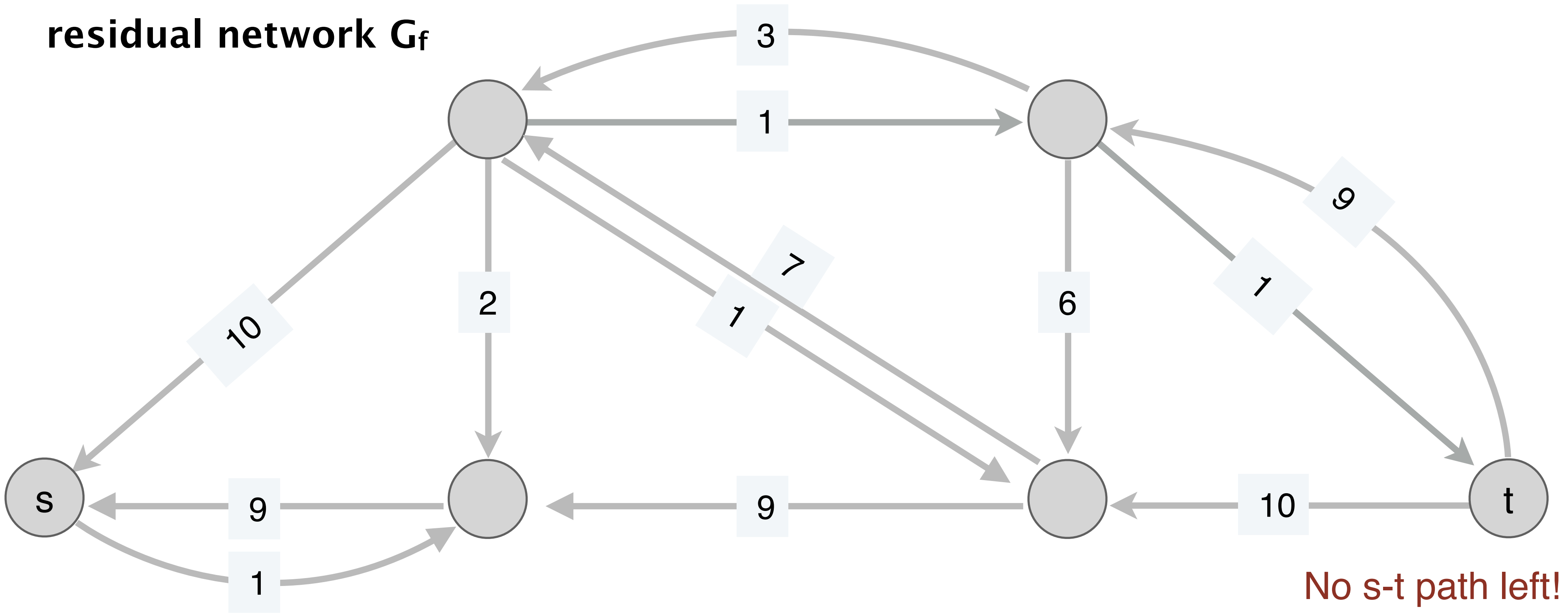
flow

capacity

value of flow

19

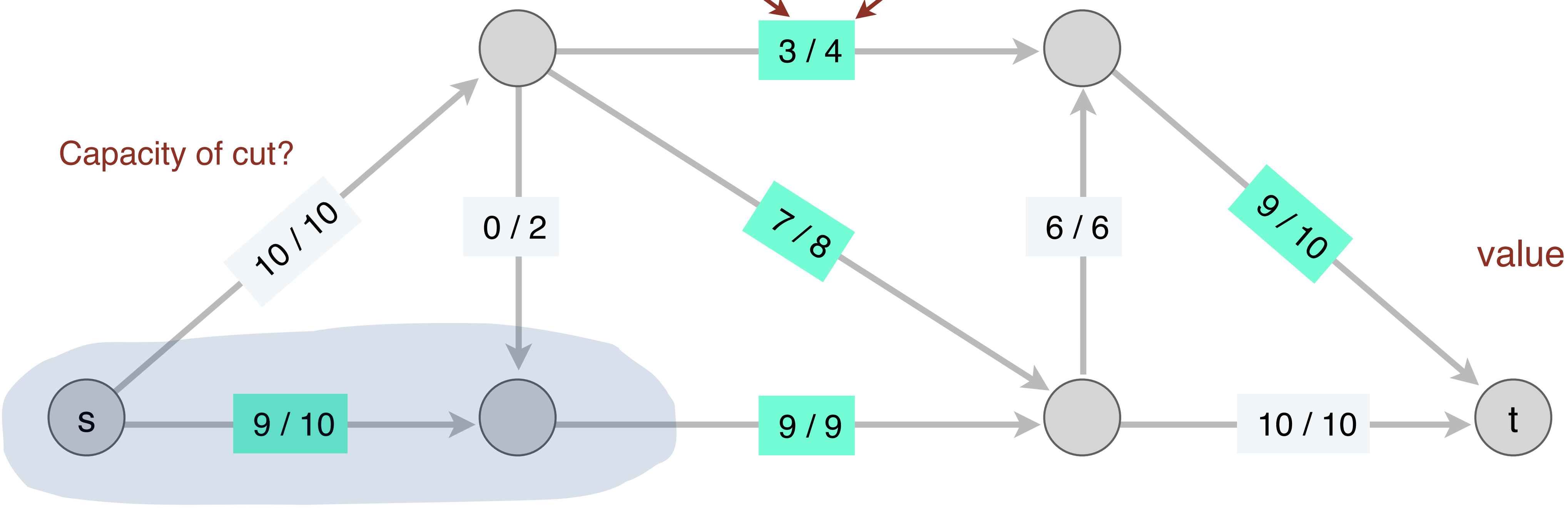
residual network G_f



No s-t path left!

Ford-Fulkerson Example

network G and flow f



flow

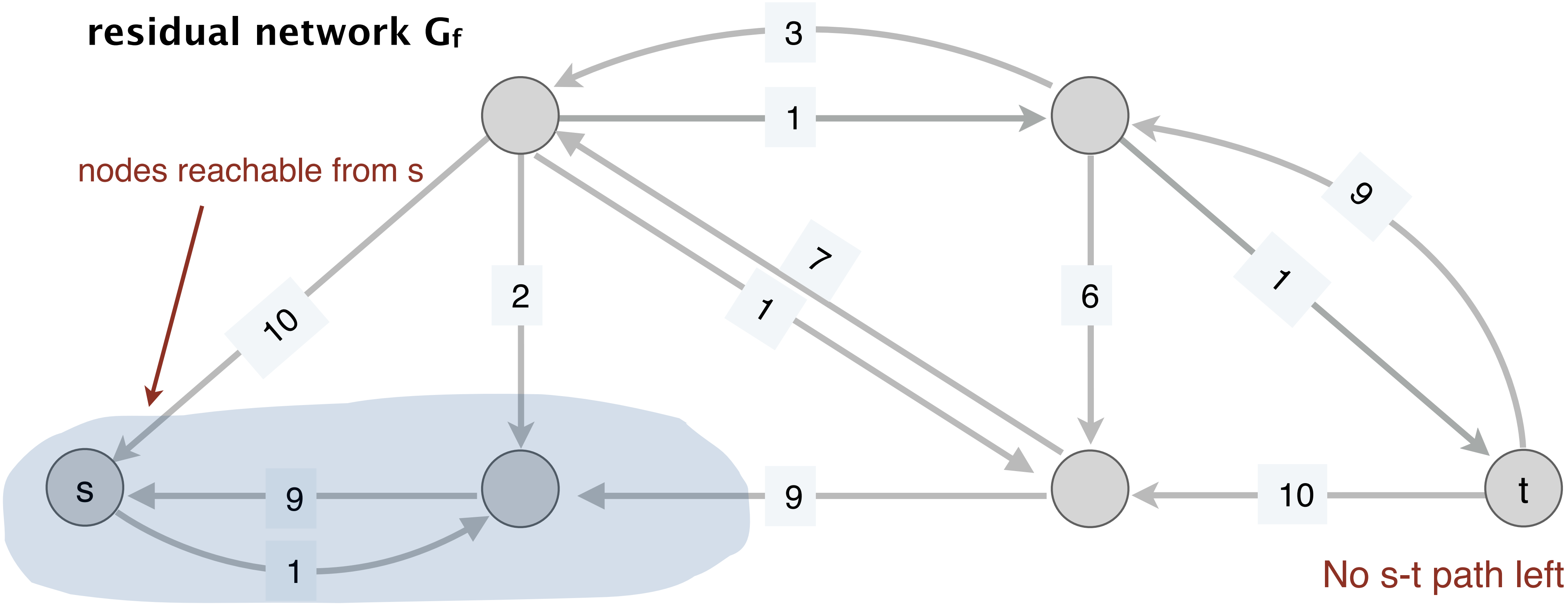
capacity

Capacity of cut?

value of flow

19

residual network G_f



nodes reachable from s

No s-t path left!

Analysis: Ford-Fulkerson

Analysis Outline (Things to Prove)

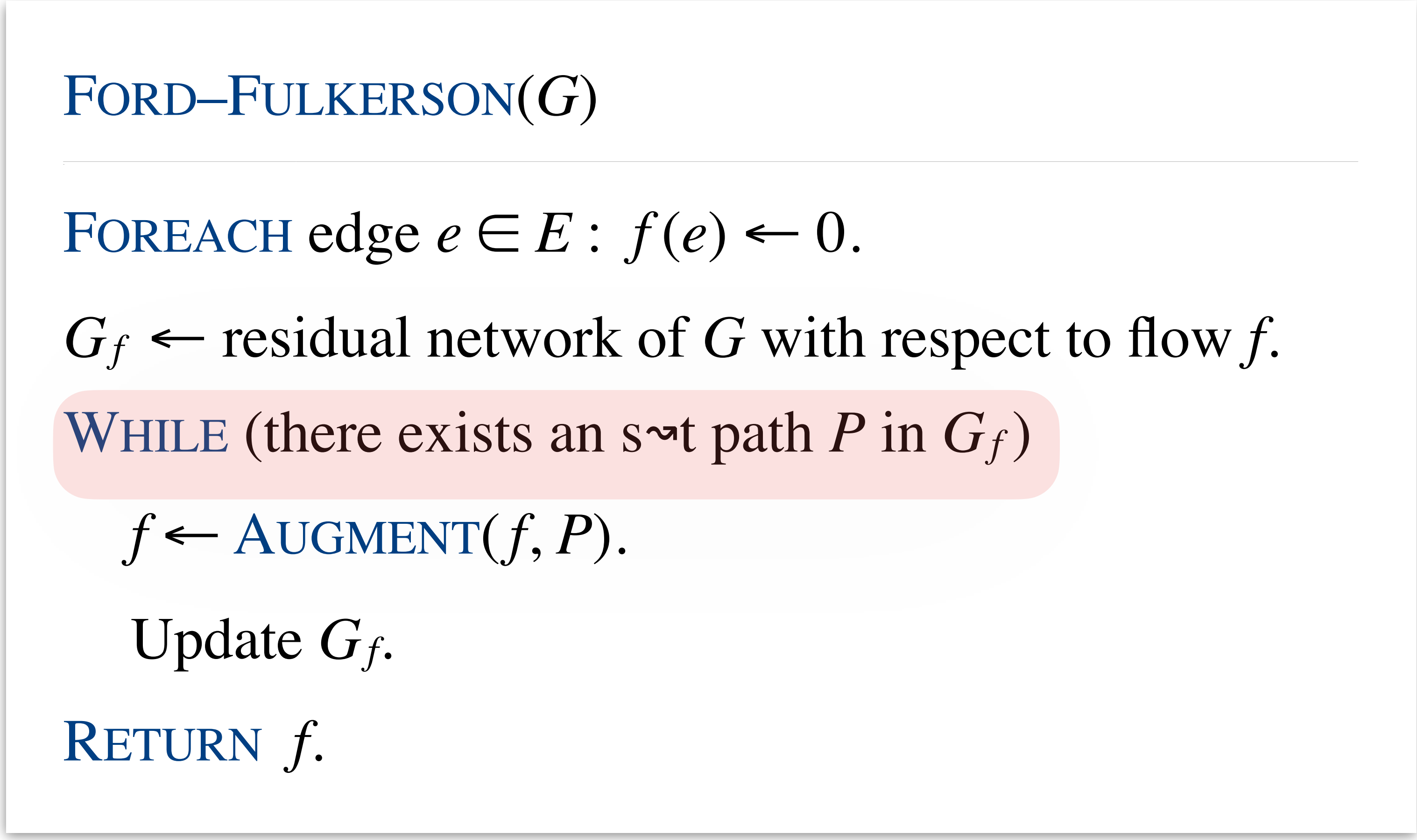
- Feasibility and value of flow:
  - Show that each time we update the flow, we are routing a feasible s-t flow through the network
  - And that value of this flow increases each time by that amount
- Optimality:
  - Final value of flow is the maximum possible
- Running time:
  - How long does it take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

Show this today, save rest for after P.S.

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 $s \rightsquigarrow t$ path $P$ in $G_f$)
    $f \leftarrow$ AUGMENT($f$, $P$).
    Update $G_f$.
RETURN $f$.
```

Performance Questions:

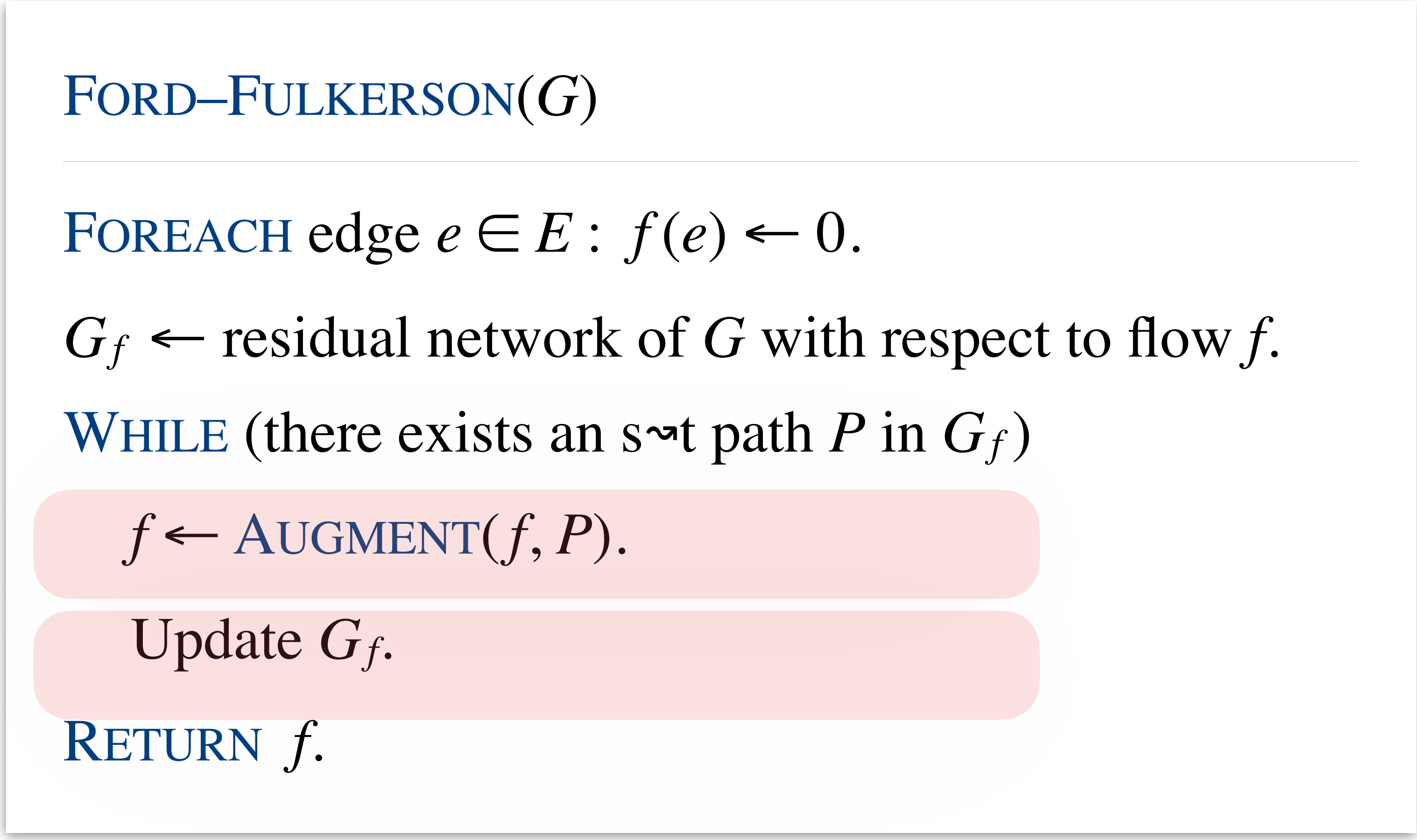
- Does the while loop terminate?
- If it terminates, can we bound the number of iterations?
- What is the Big-O running time of the whole algorithm?

Ford-Fulkerson Running Time

Recall we proved that with each call to AUGMENT, we increase **value of the s-t flow** by $b = \text{bottleneck}(G_f, P)$

- **Assumption**. We assumed 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 $v(f) \leq nC$
- 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 $f$.
```

We know there are $O(nC)$ iterations. How many operations per iteration?

- Cost to find an augmenting path in G_f?
- Cost to augment flow on path?
- Cost to 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