Ford-Fulkerson Algorithm

Admin

- Assignment 6 is due next Wed 11 pm
    - Office and TA hours today
    - Come to get an early start
- Question 1 (Bookshelf dynamic programming question)
    - Review book partitioning question from class
    - Similar problem, different cost functions; no restriction on number of shelves/partitions

Story So Far

- Defined the max-flow and min-cut problem
- Relationship between flows and cuts. Let f be **any** s-t flow and (S, T) be **any** s-t cut then $v(f) \leq c(S, T)$
  - Key idea in proof: when $v(f) = c(S, T)$?
    - When $f_{in}(S) = 0$ and $f_{out}(S) = c(S, T)$
    - That is no flow is entering cut S and all edges leaving it are fully saturated
- Max-flow min-cut theorem. Given any flow network G, there exists a feasible (s, t)-flow f and a (s, t)-cut (S, T) s.t., $v(f) = c(S, T)$, and f is the max flow and (S, T) is the min cut

Towards a Max-Flow Algorithm

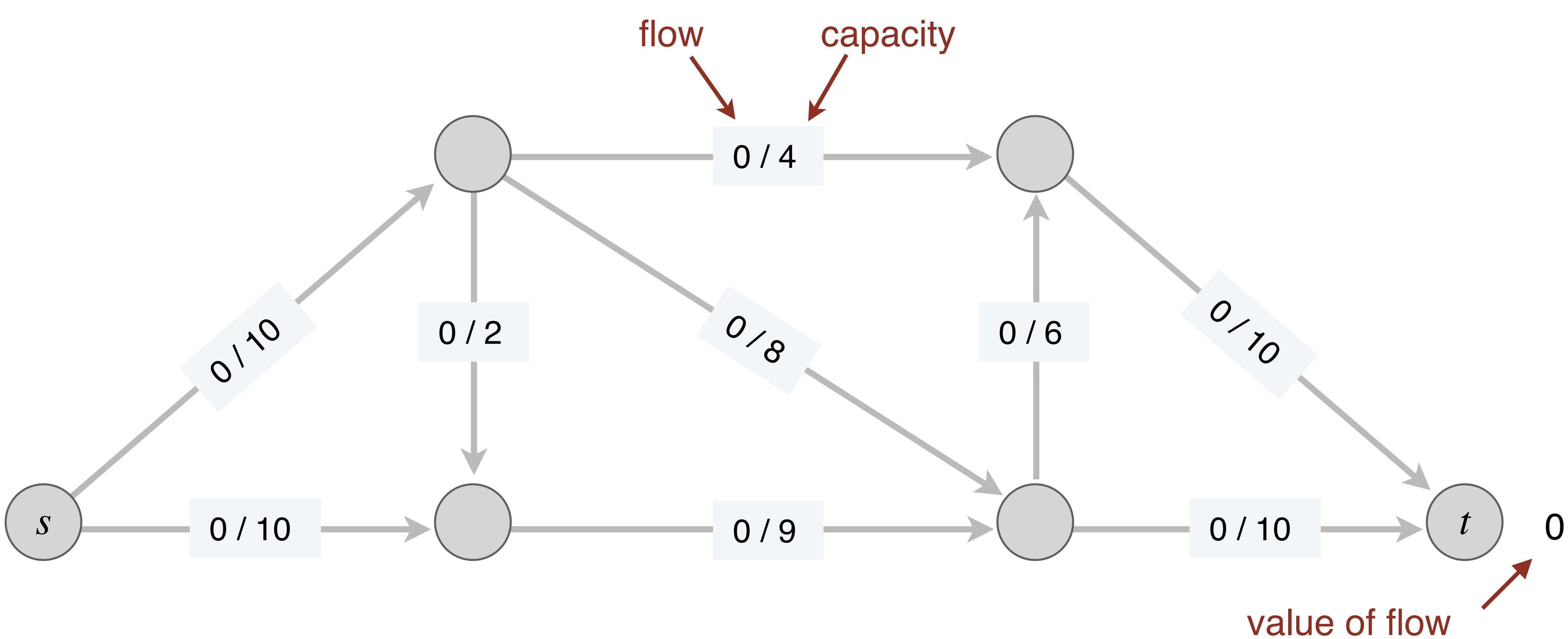
- Today: we will prove the max-flow min-cut theorem *constructively*
- We will design a max-flow algorithm and show that there is a s-t cut s.t. value of flow computed by algorithm $=$ capacity of cut
- Let's start with a greedy approach
    - Push as much flow as possible down a s-t path
    - This won't actually work
    - But gives us a sense of what we need to keep track off to improve upon it

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 take an example

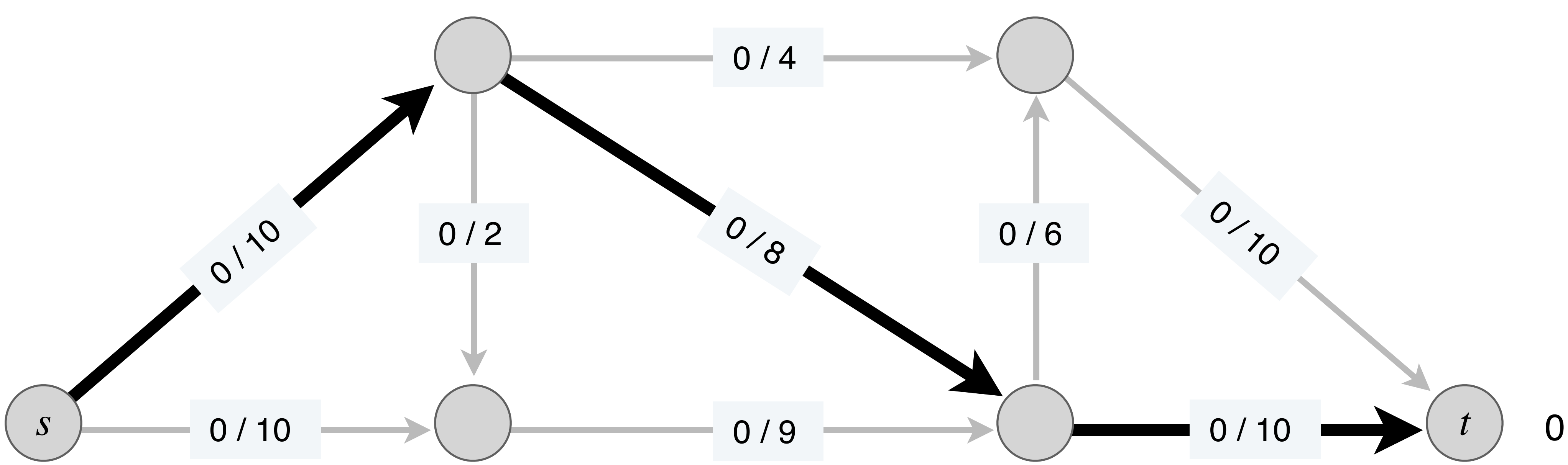
Towards a Max-Flow Algorithm

- 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



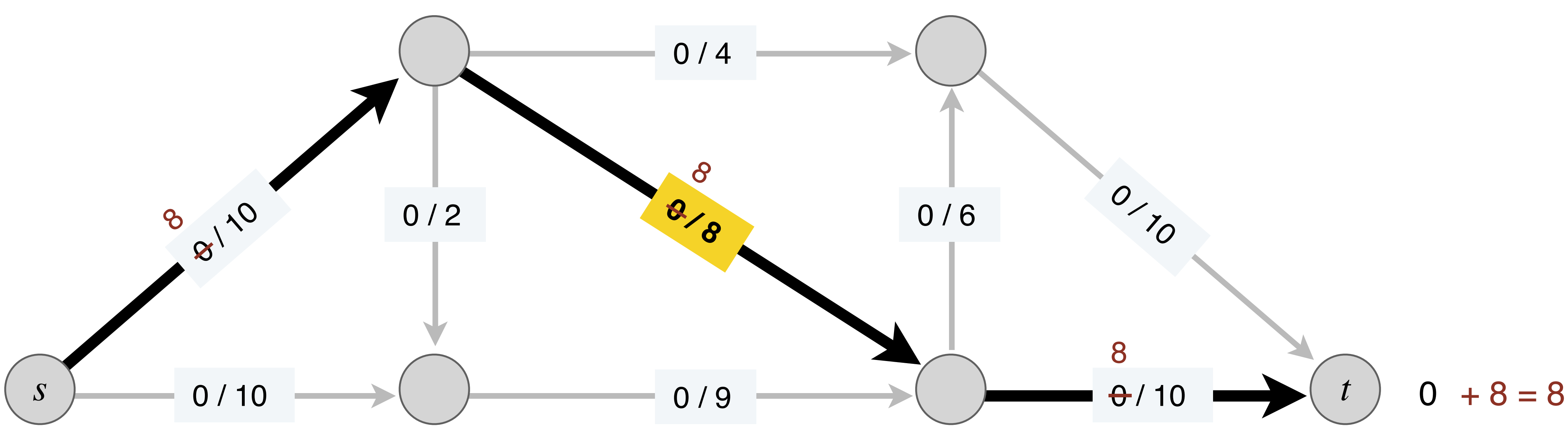
Towards a Max-Flow Algorithm

- 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



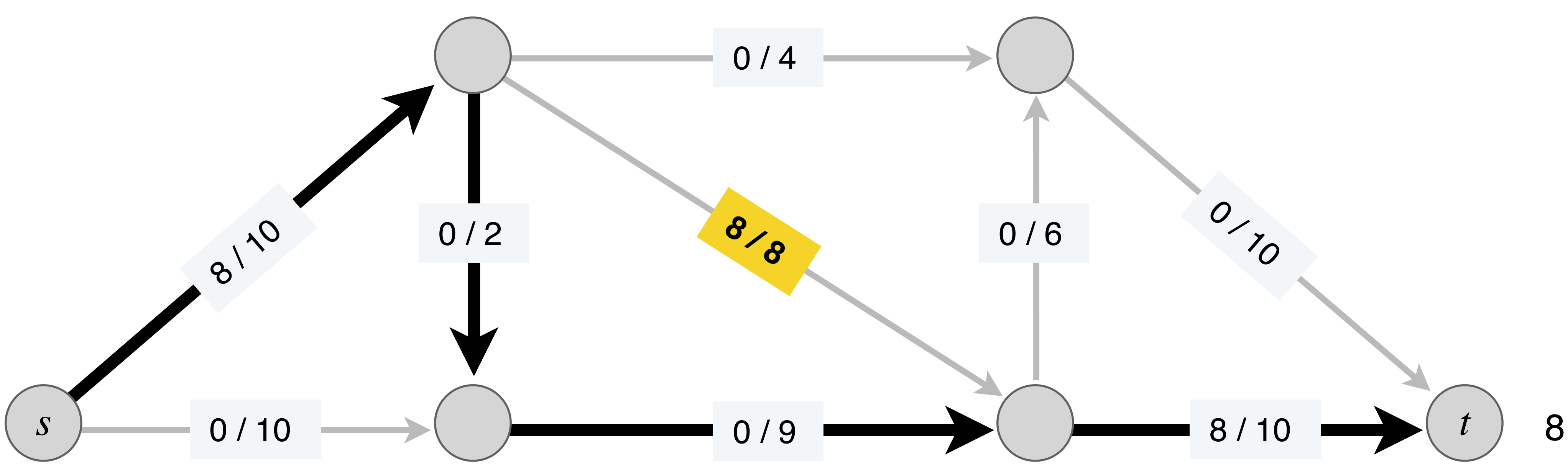
Towards a Max-Flow Algorithm

- 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



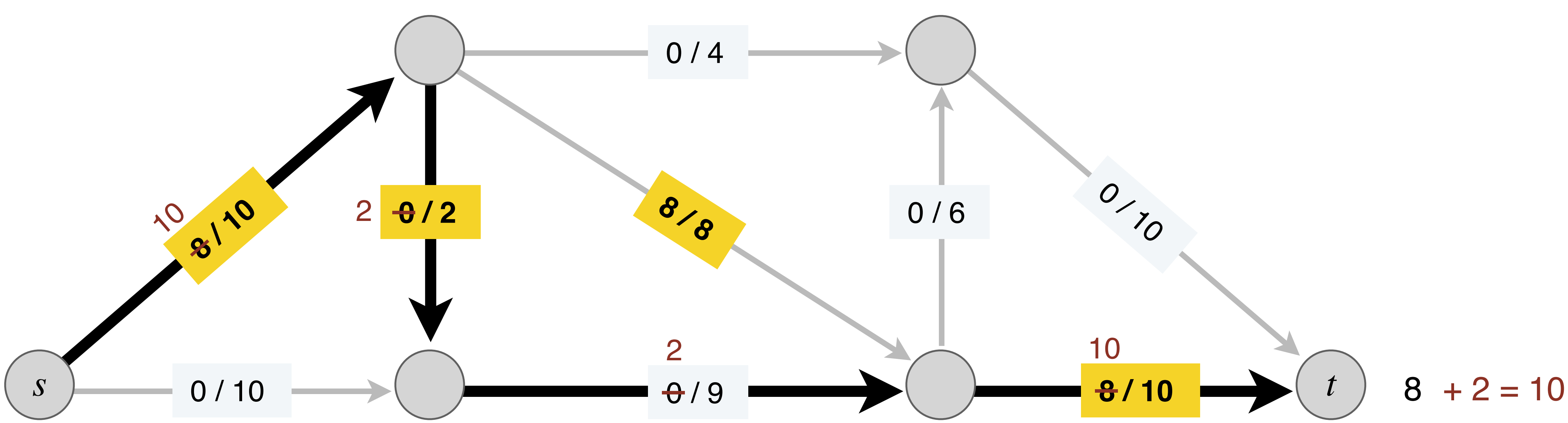
Towards a Max-Flow Algorithm

- 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



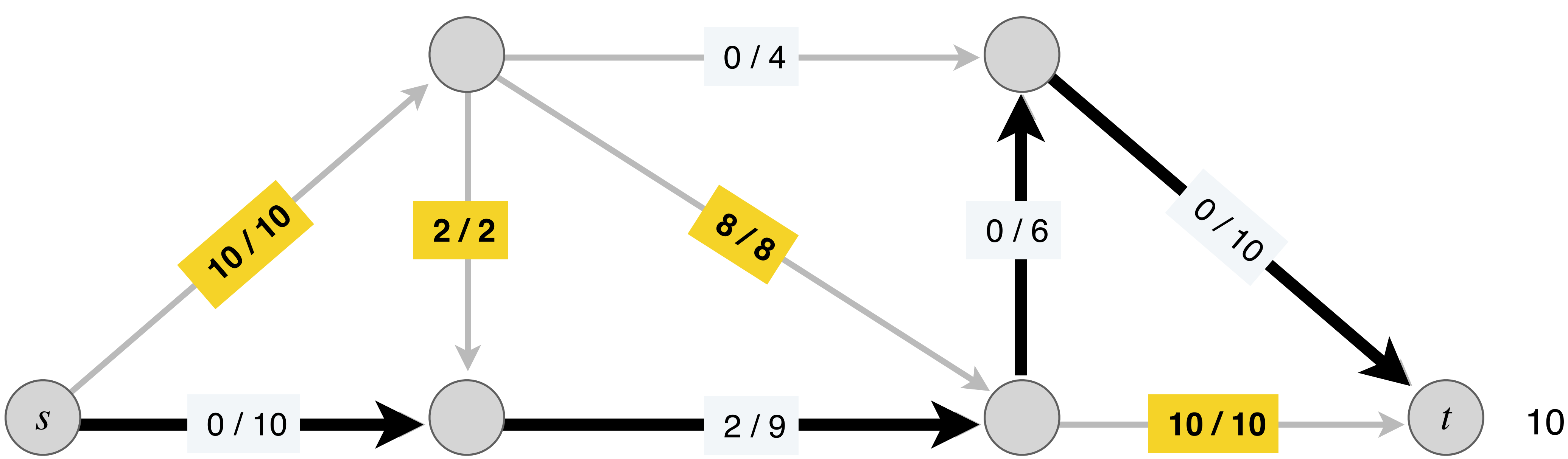
Towards a Max-Flow Algorithm

- 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

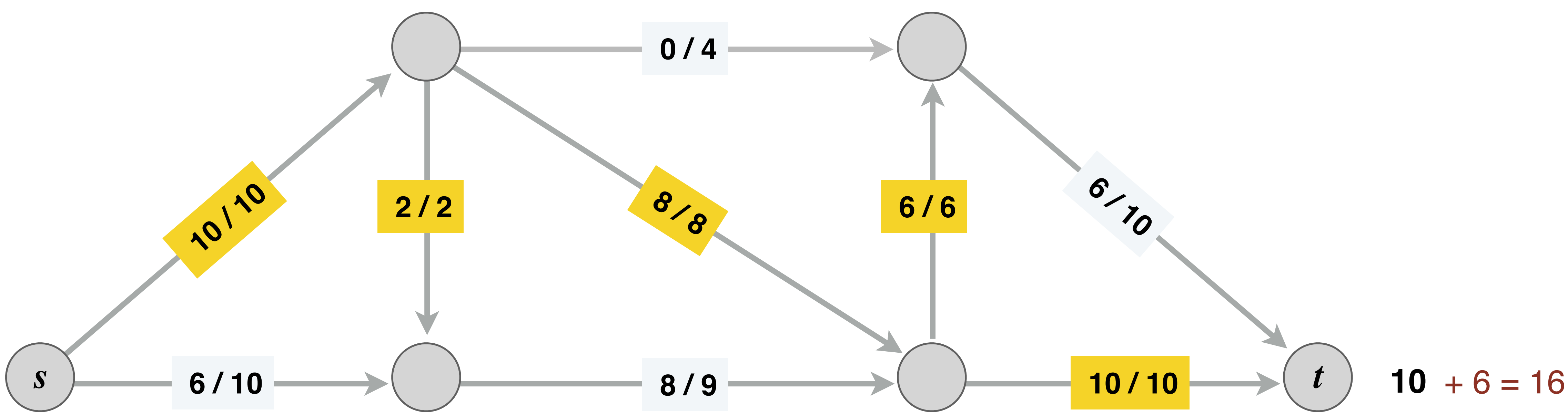


Towards a Max-Flow Algorithm

- 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

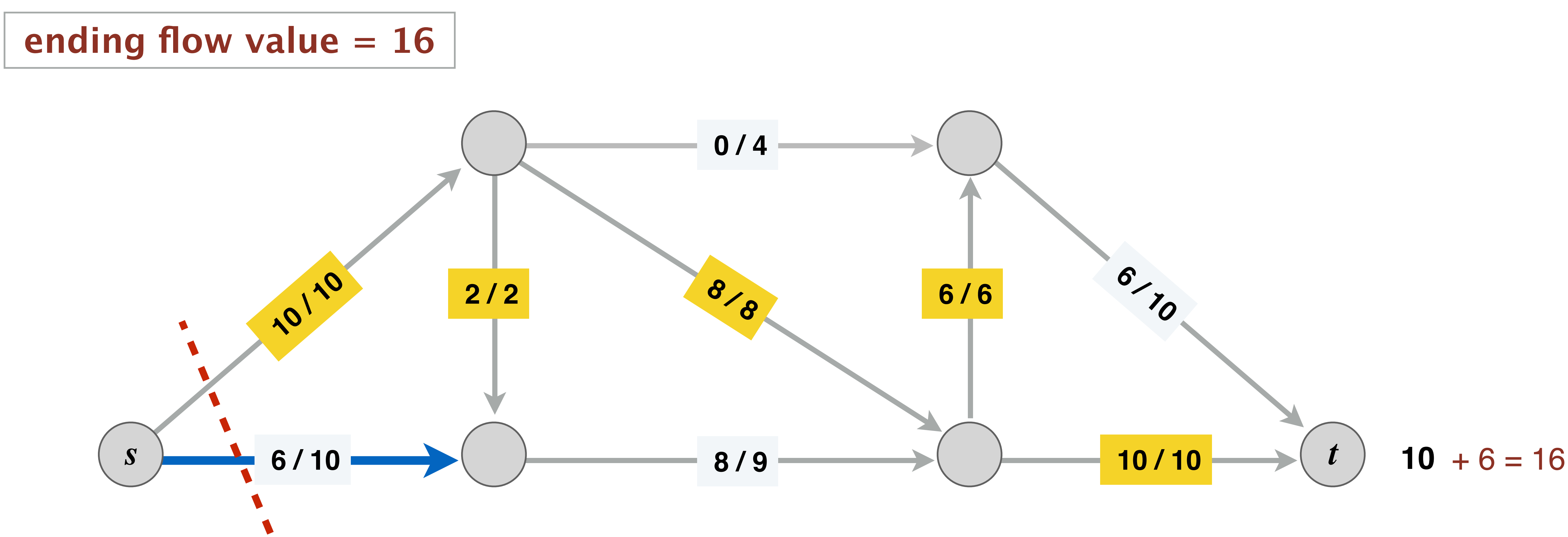
Is this the best we can do?

ending flow value = 16



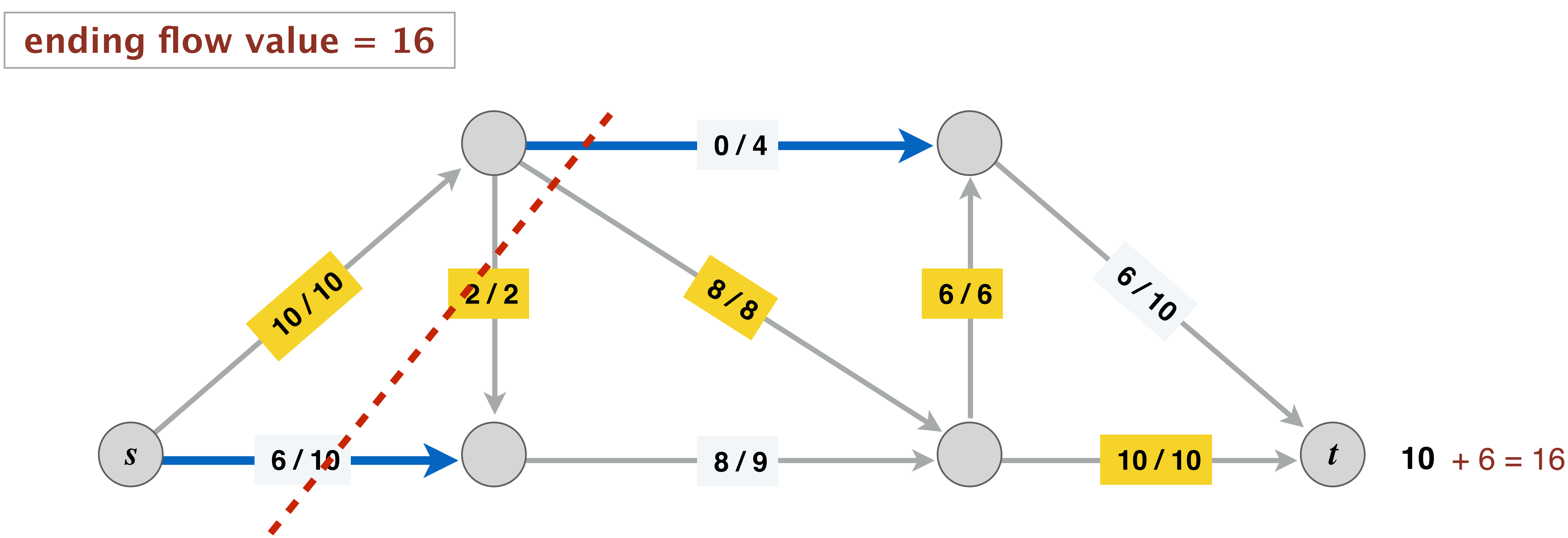
Towards a Max-Flow Algorithm

- 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



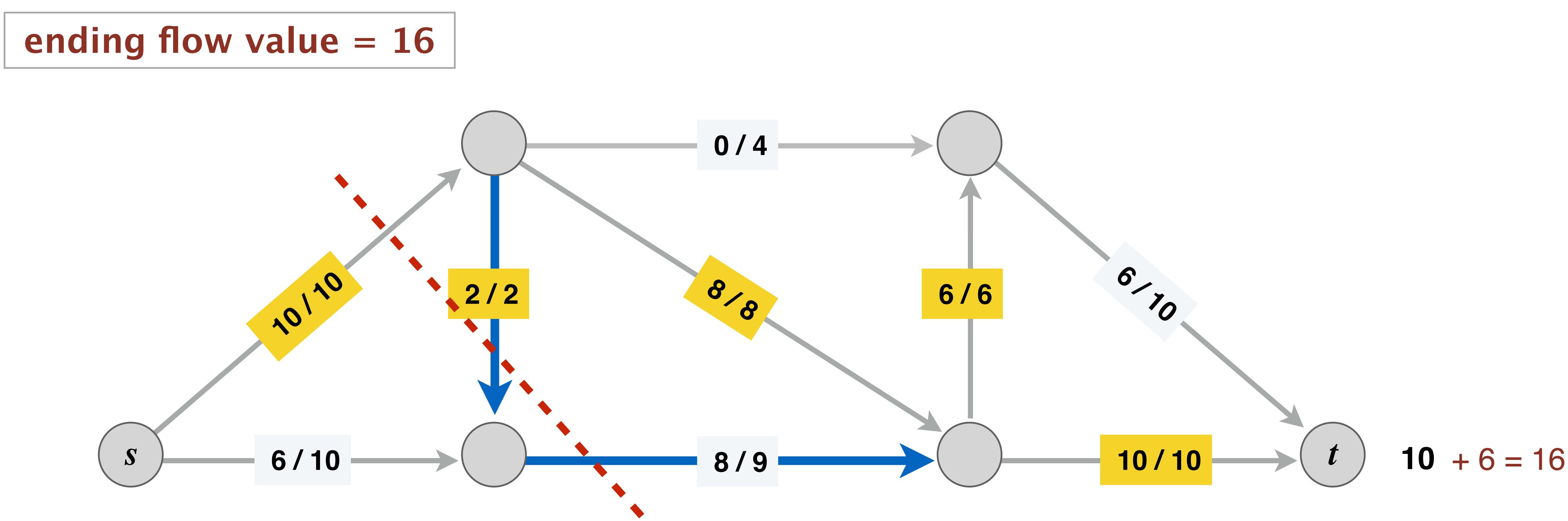
Towards a Max-Flow Algorithm

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



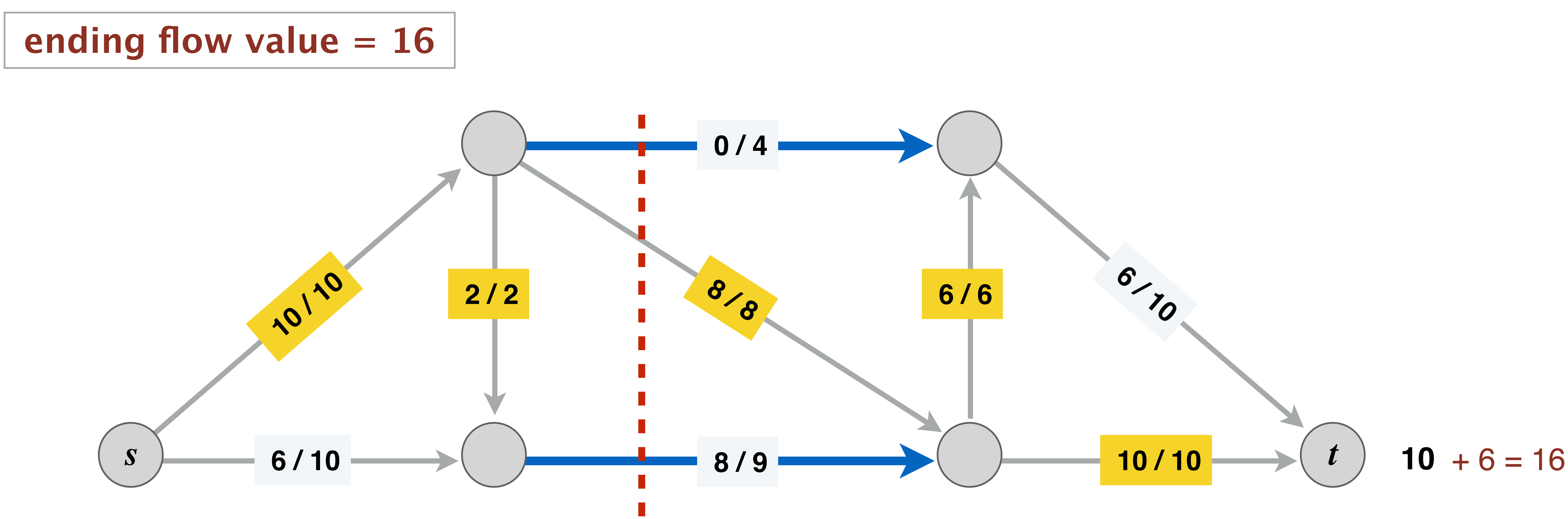
Towards a Max-Flow Algorithm

- 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

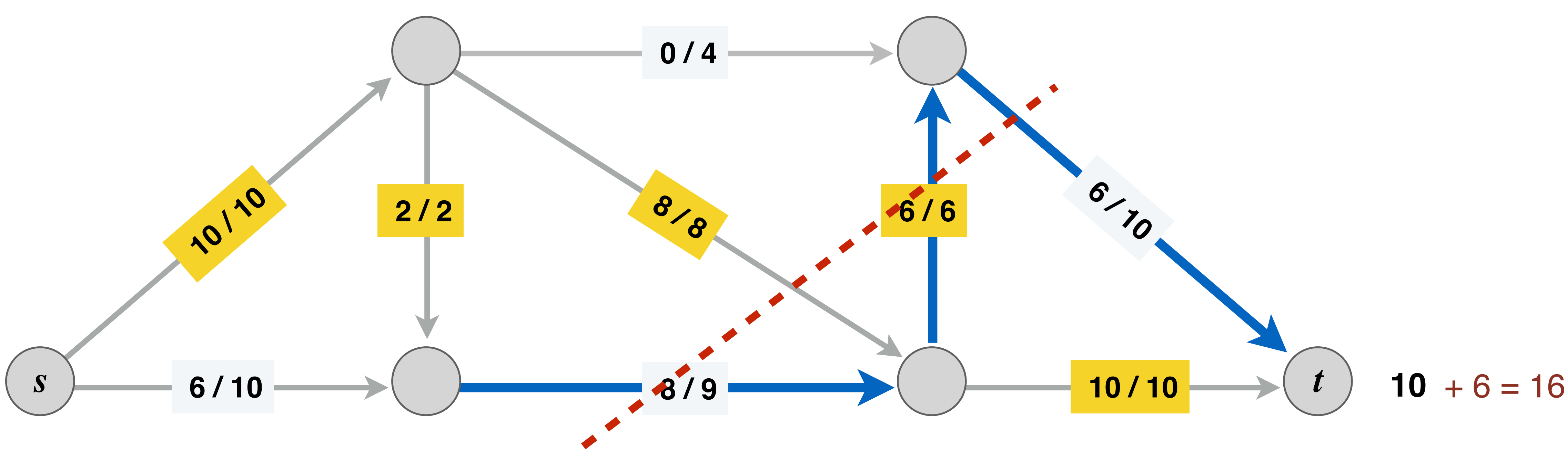
- Start with $f(e) = 0$ for each edge
- Find an $s \leadsto t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Towards a Max-Flow Algorithm

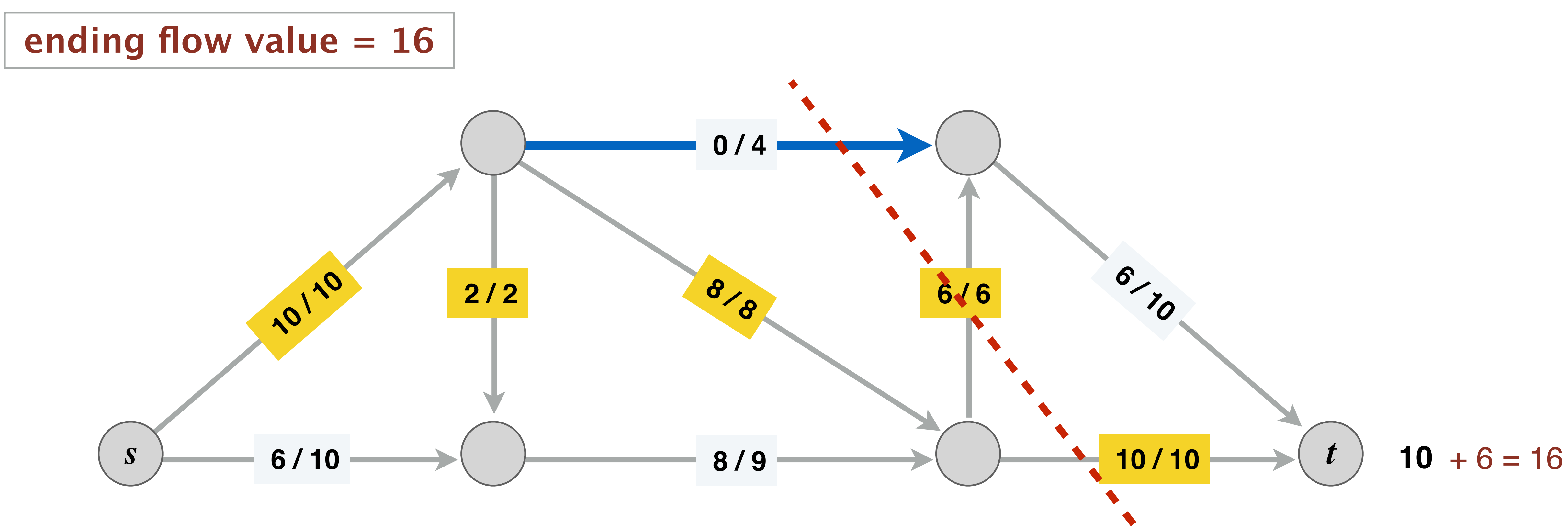
- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck

ending flow value = 16



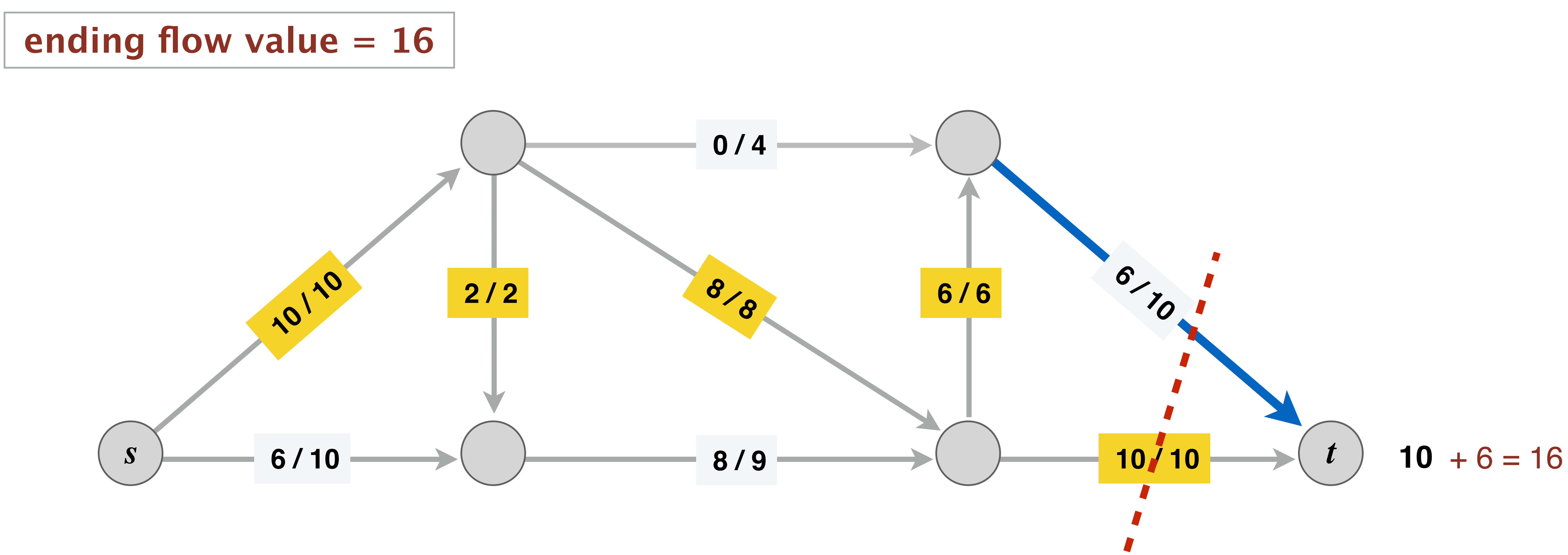
Towards a Max-Flow Algorithm

- 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



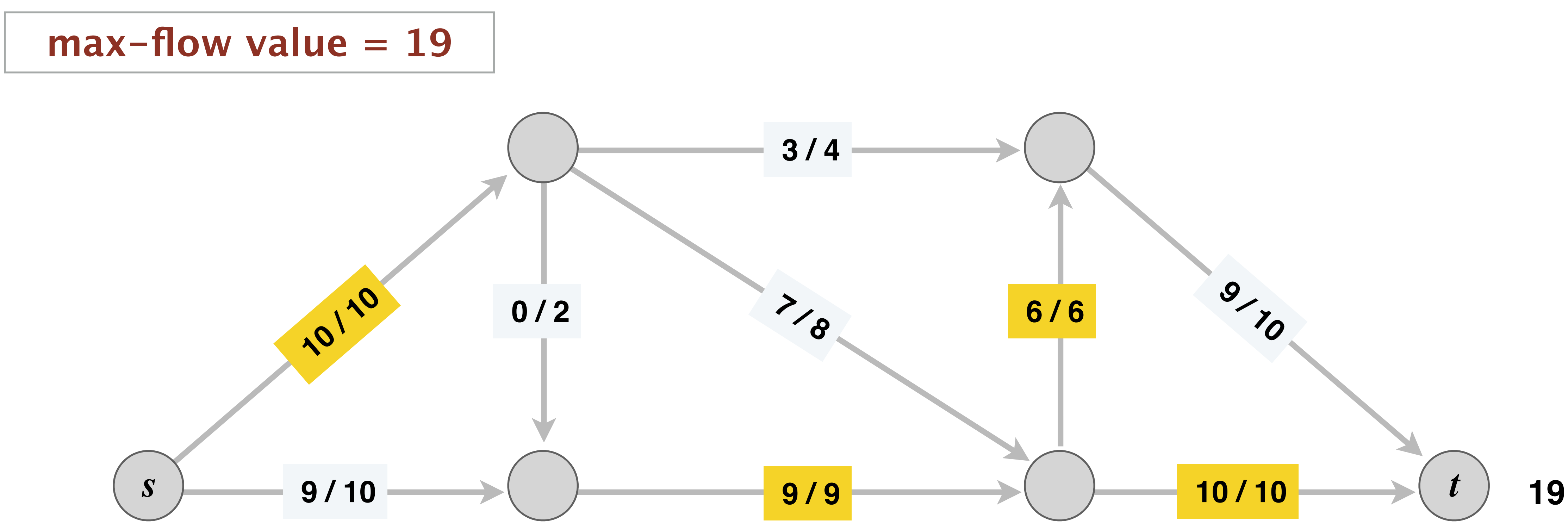
Towards a Max-Flow Algorithm

- 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



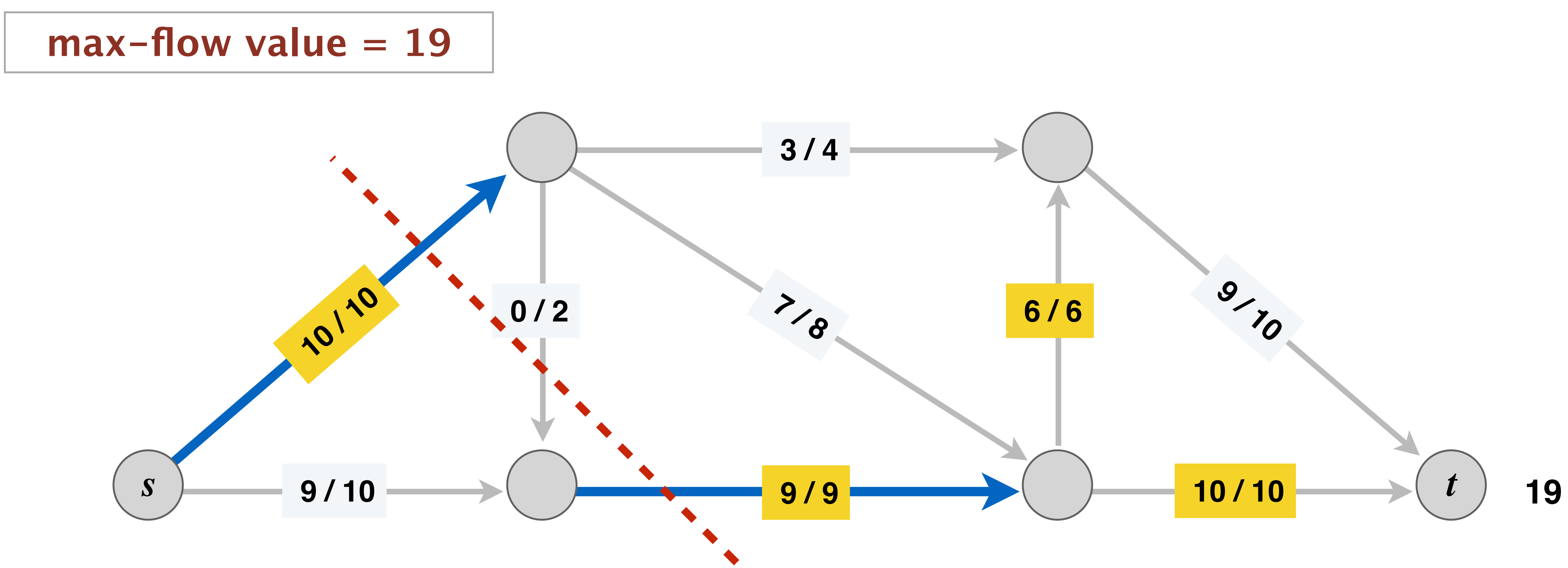
Towards a Max-Flow Algorithm

- 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



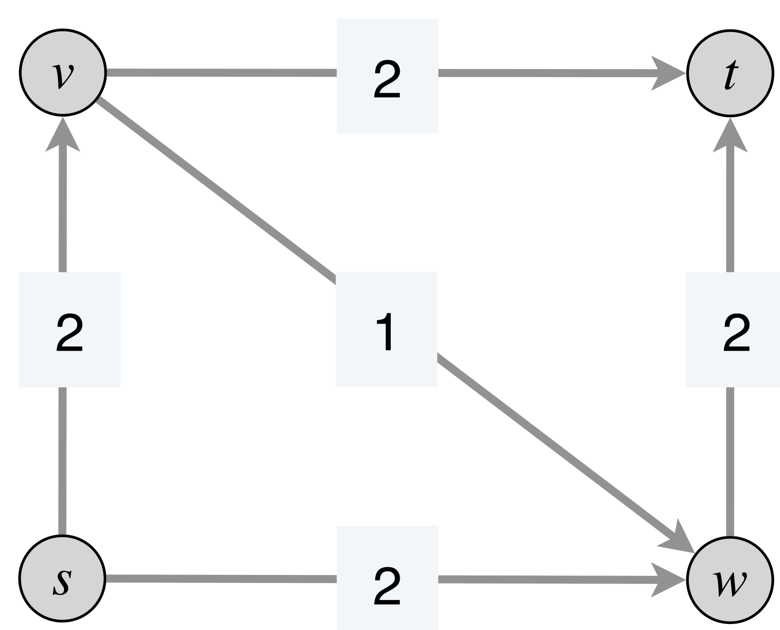
Towards a Max-Flow Algorithm

- 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



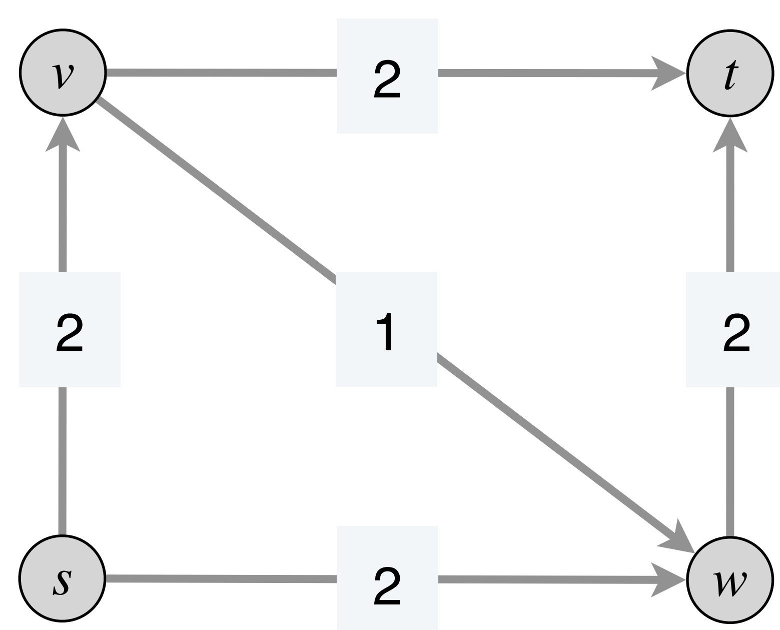
Why Greedy Fails

- **Problem**: greedy can never "undo" a bad flow decision
- Consider the following flow network



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



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

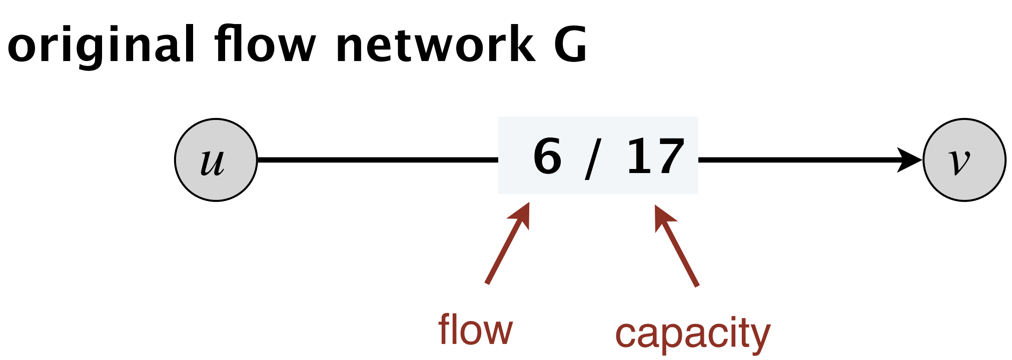
Ford-Fulkerson Algorithm

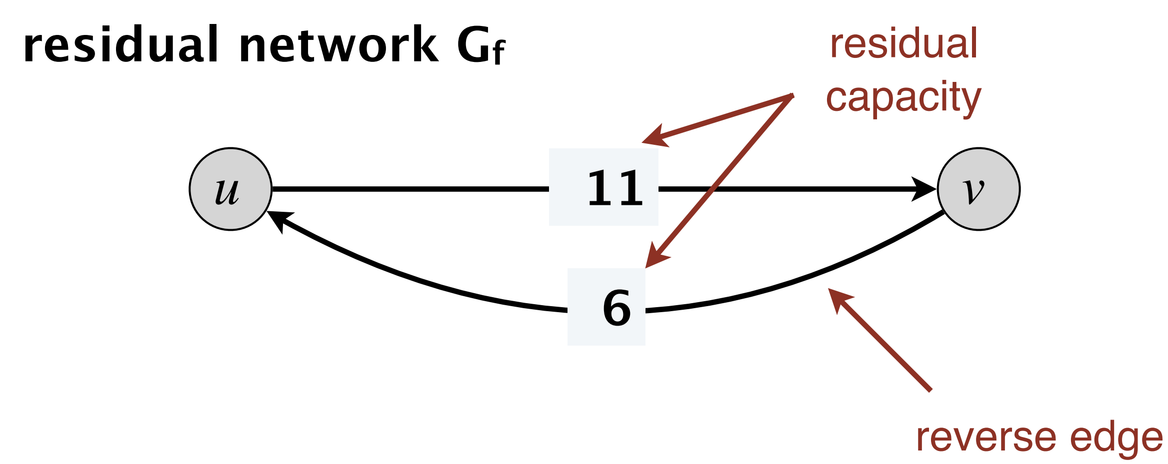
Ford Fulkerson: Idea

- 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
- 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:
    - Vertices in G_f same as G
    - **(Forward edge)** For $e \in E$ with residual capacity $c_r = c(e) - f(e) > 0$ create $e \in E_f$ with capacity c_r
    - **(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 could mean either
  - We increment existing flow on an edge
  - Or we decrement flow on an edge ("push back existing flow")

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.

The path P is 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.

Updating flow in G

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
- 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)

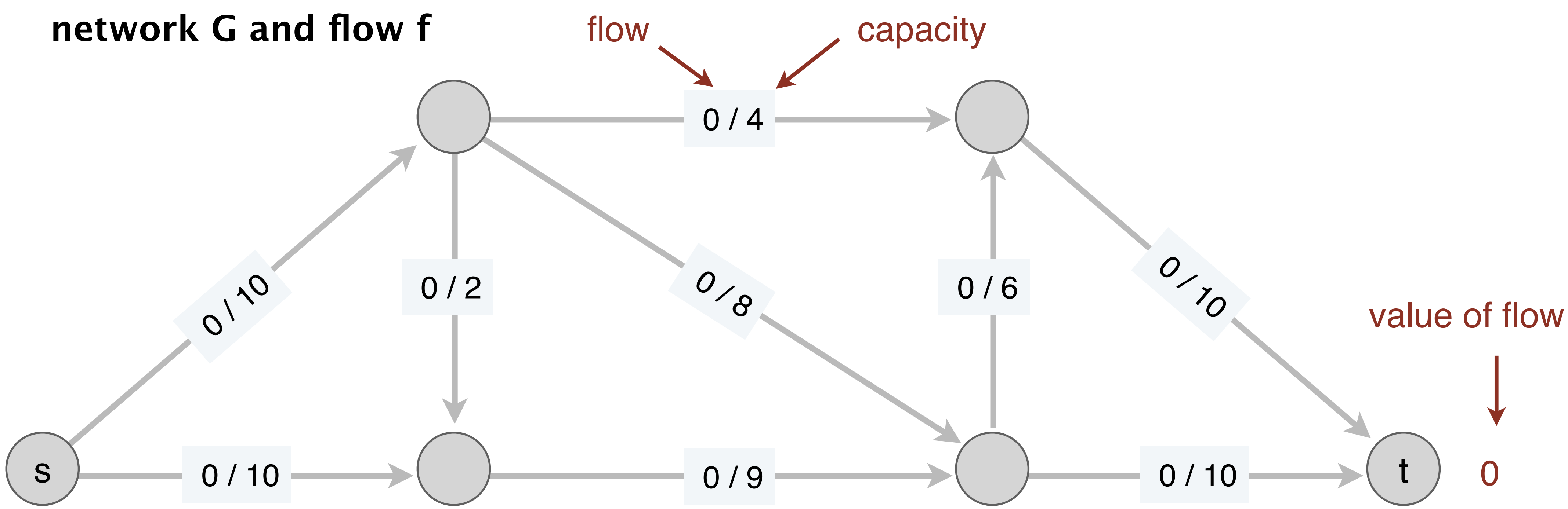
  $f \leftarrow$ AUGMENT(f, P).

  Update G_f.

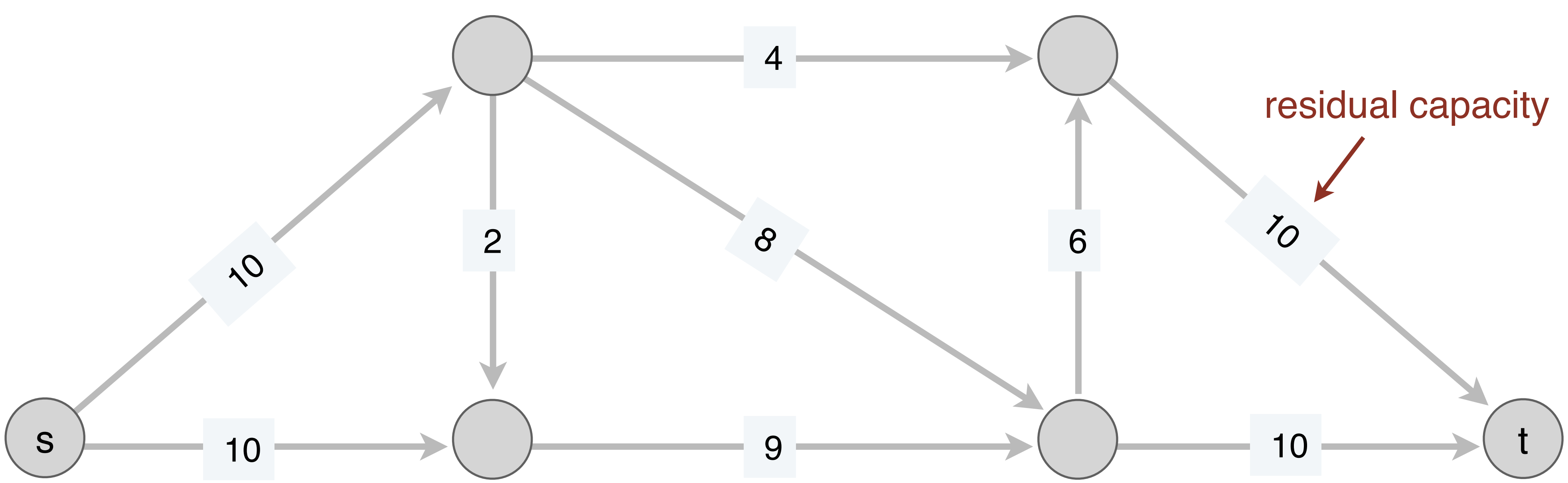
RETURN f.

Ford-Fulkerson Example

network G and flow f

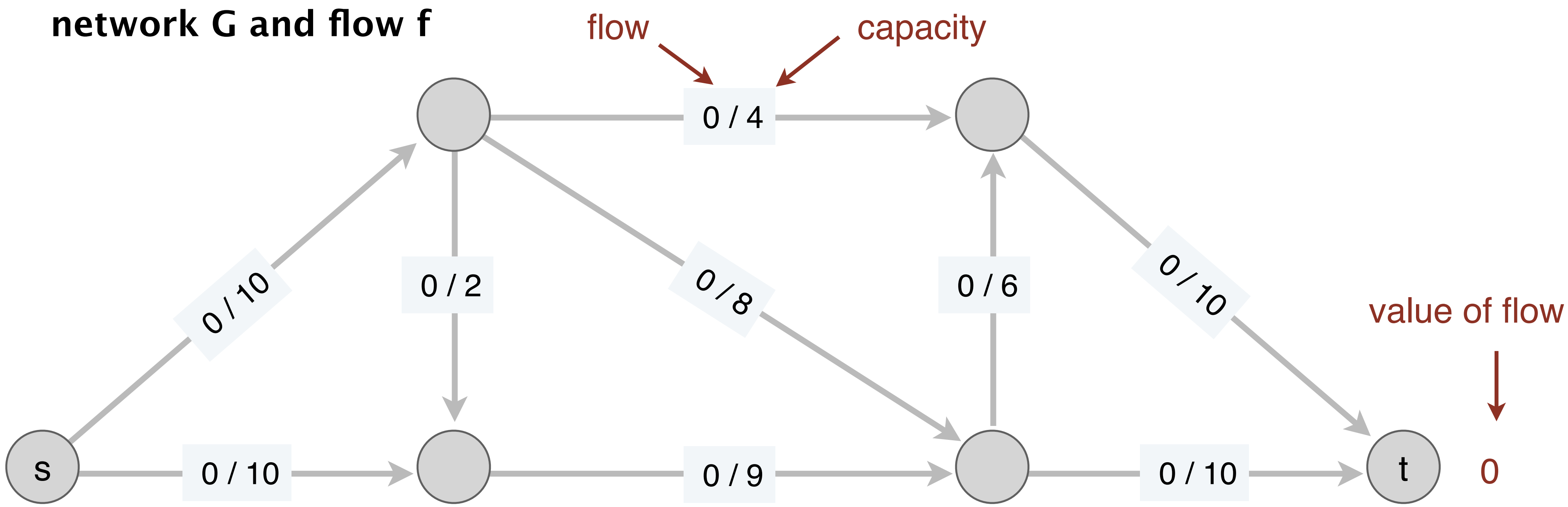


residual network G_f

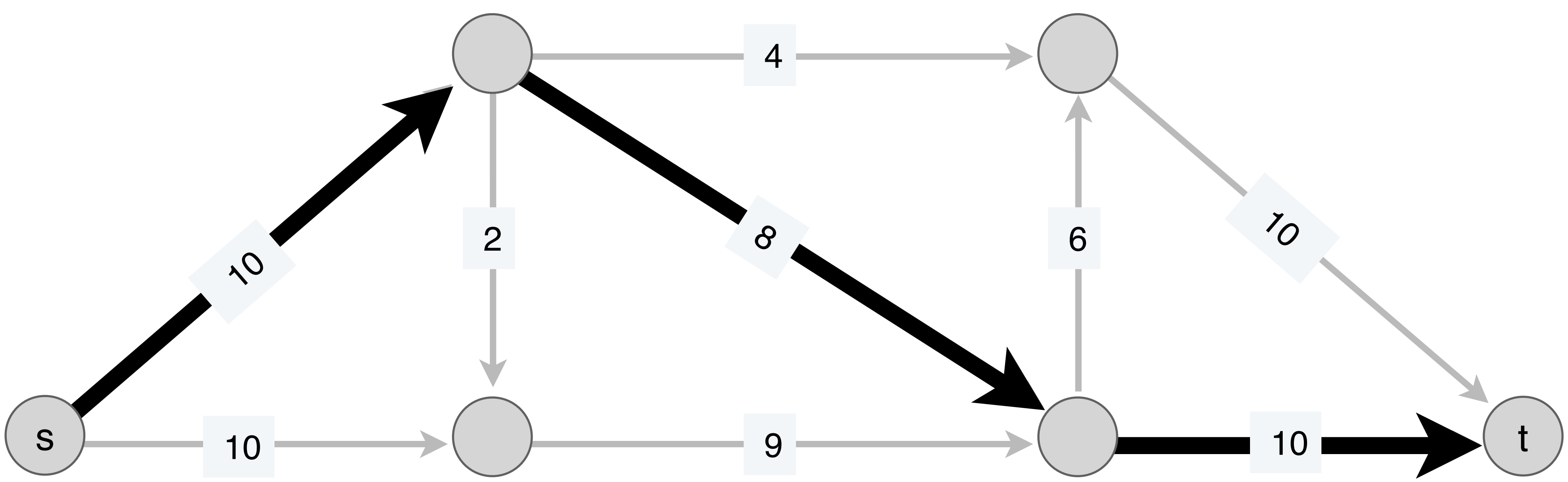


Ford-Fulkerson Example

network G and flow f

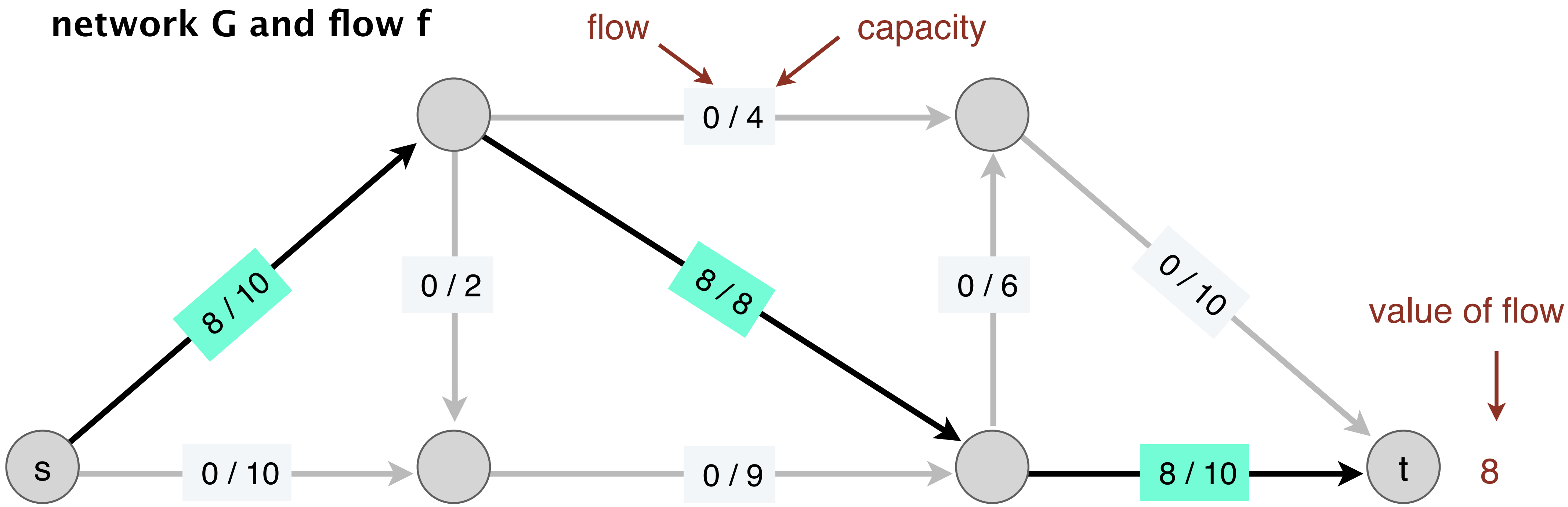


P in residual network G_f

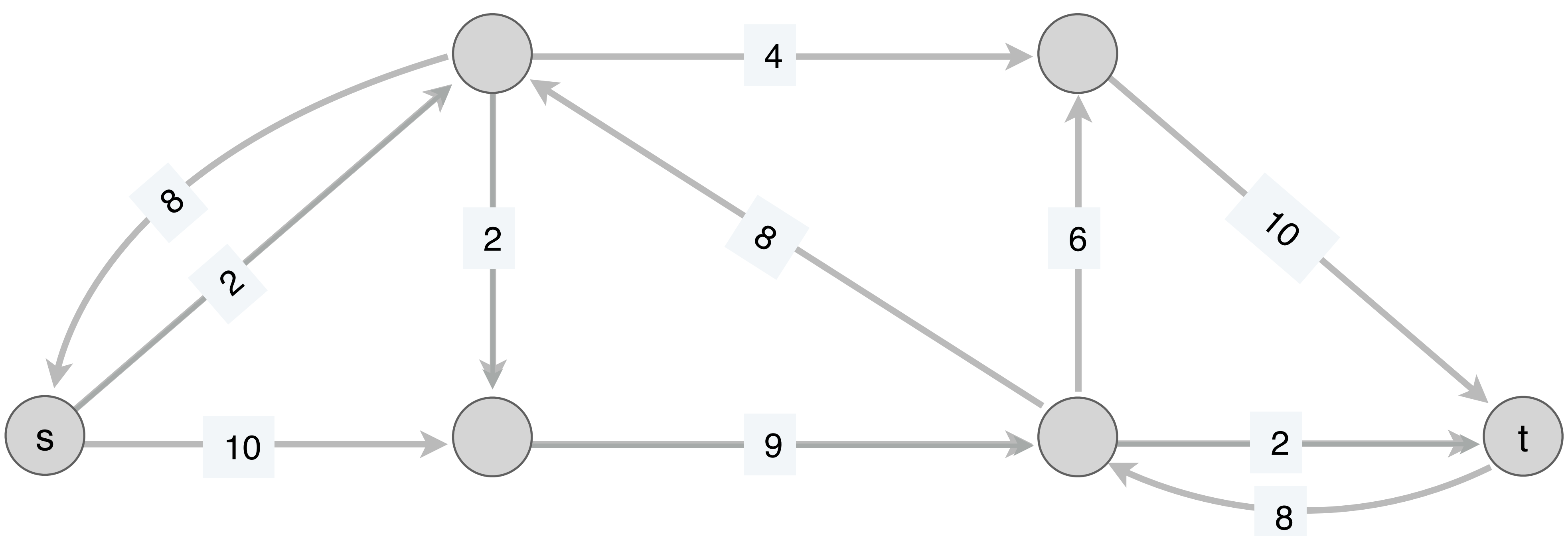


Ford-Fulkerson Example

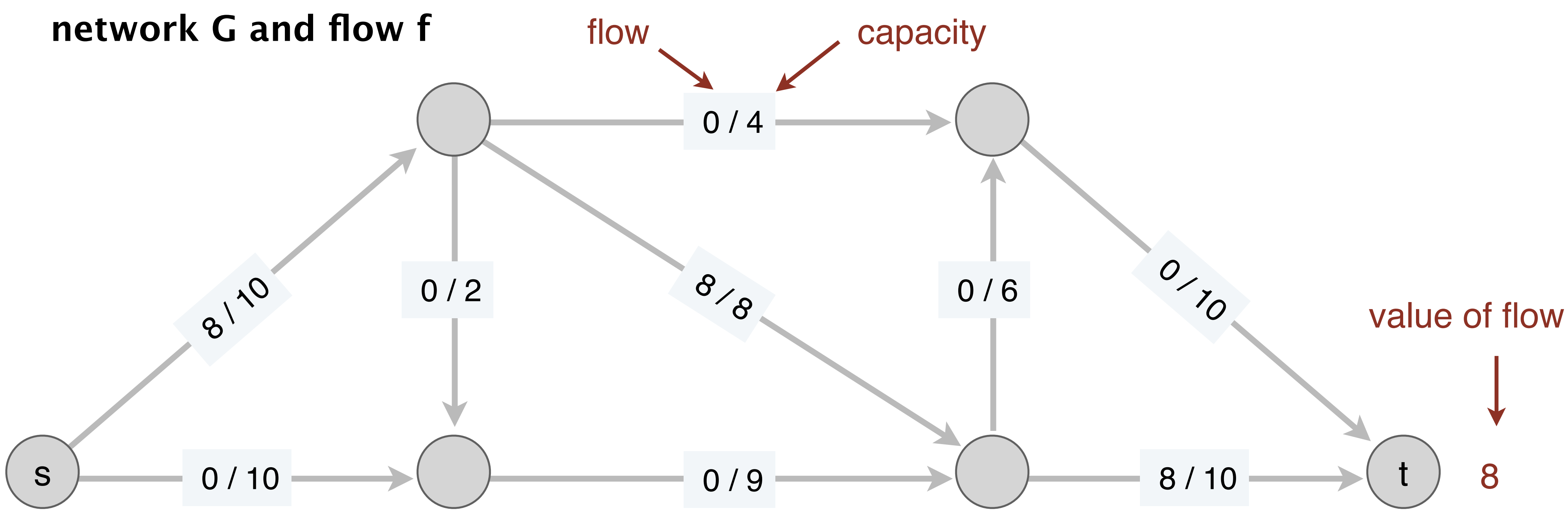
network G and flow f

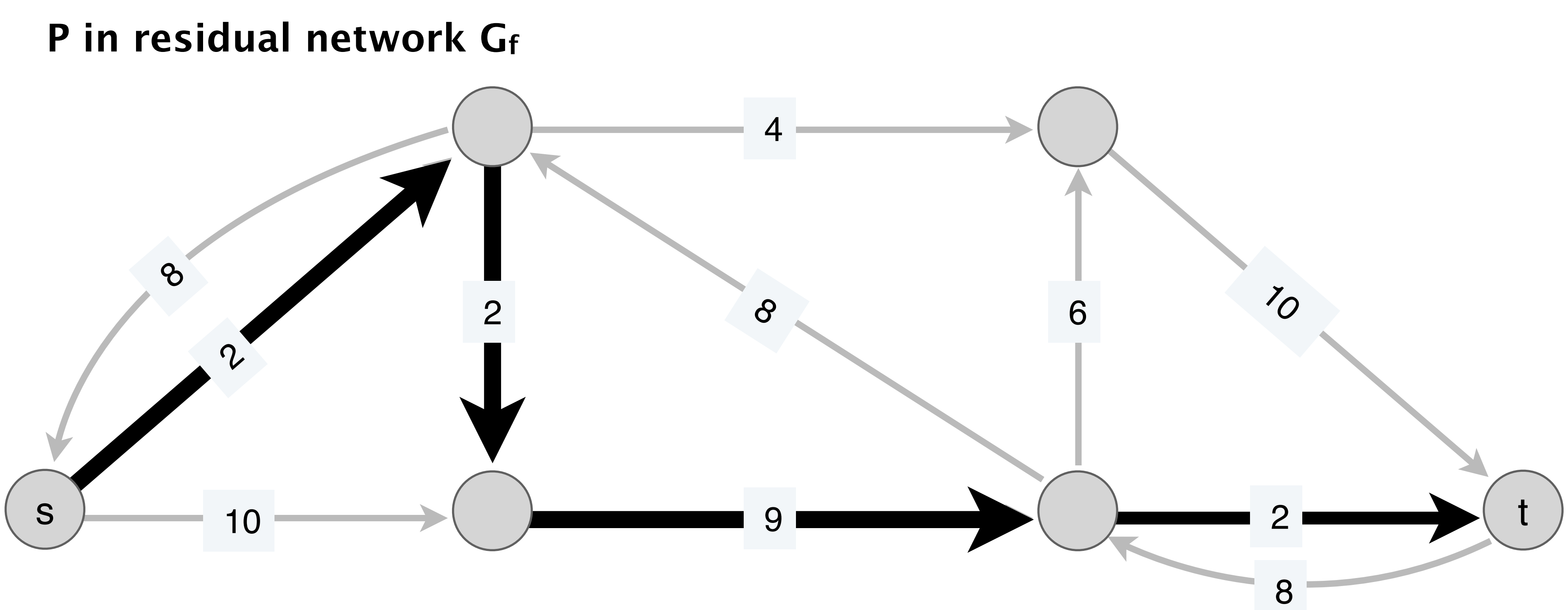


residual network G_f

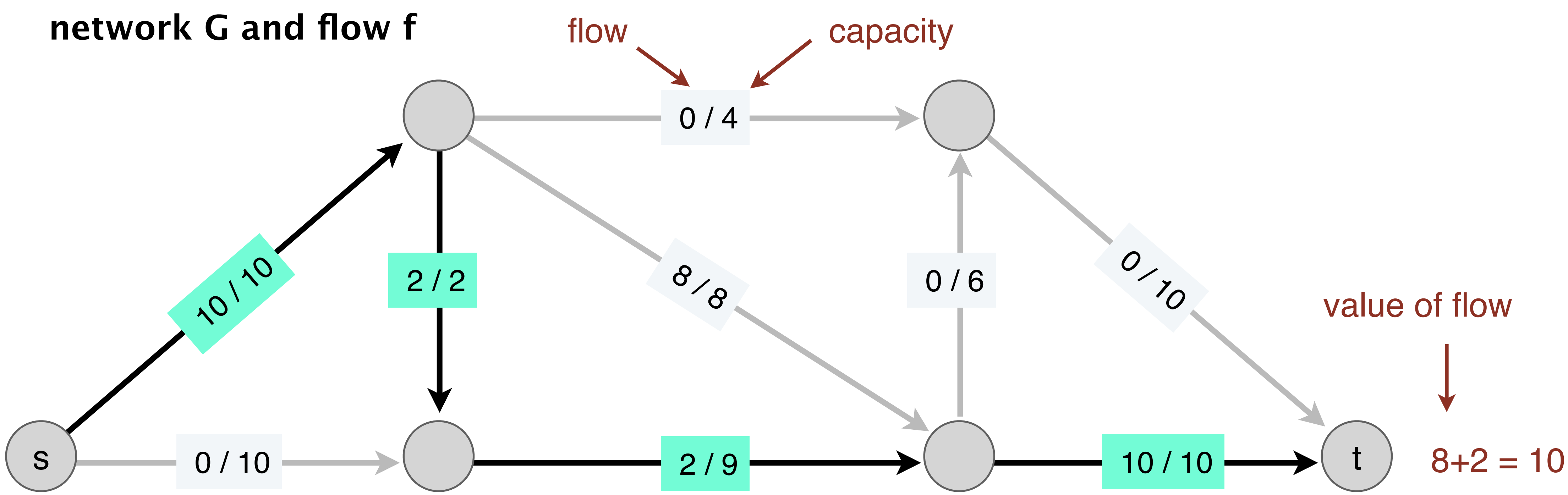


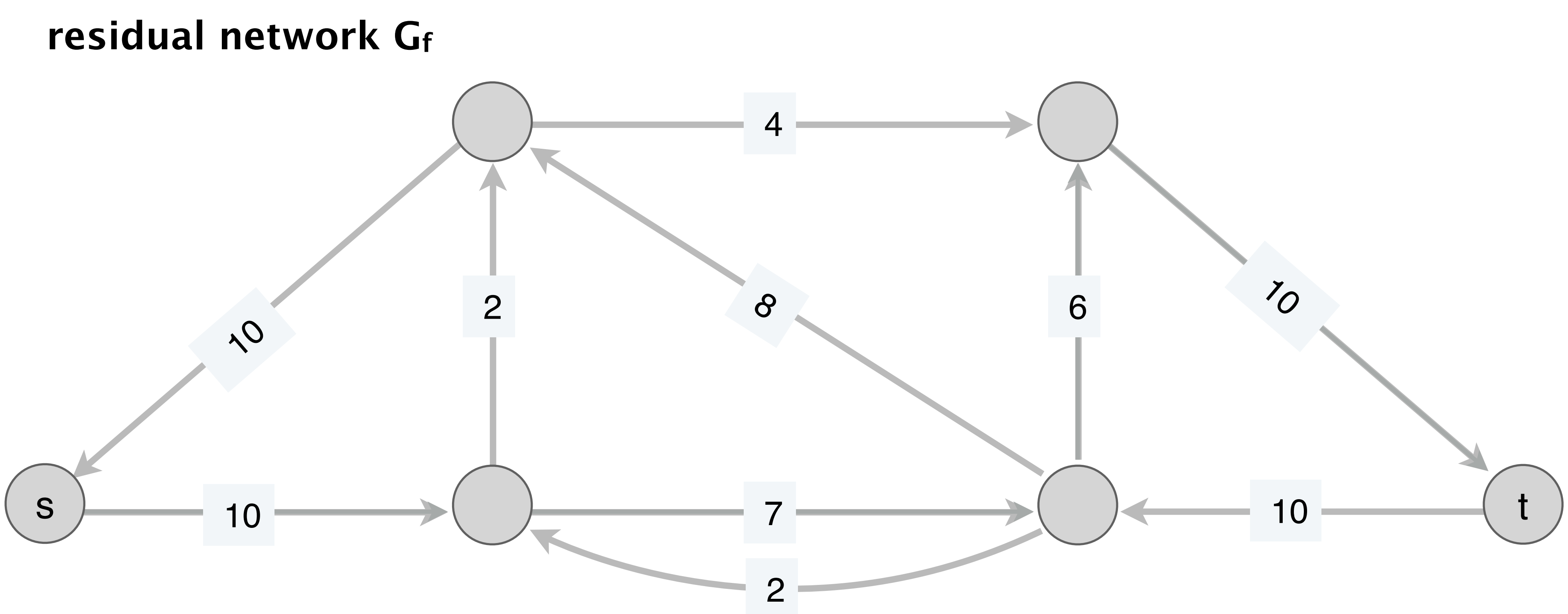
Ford-Fulkerson Example





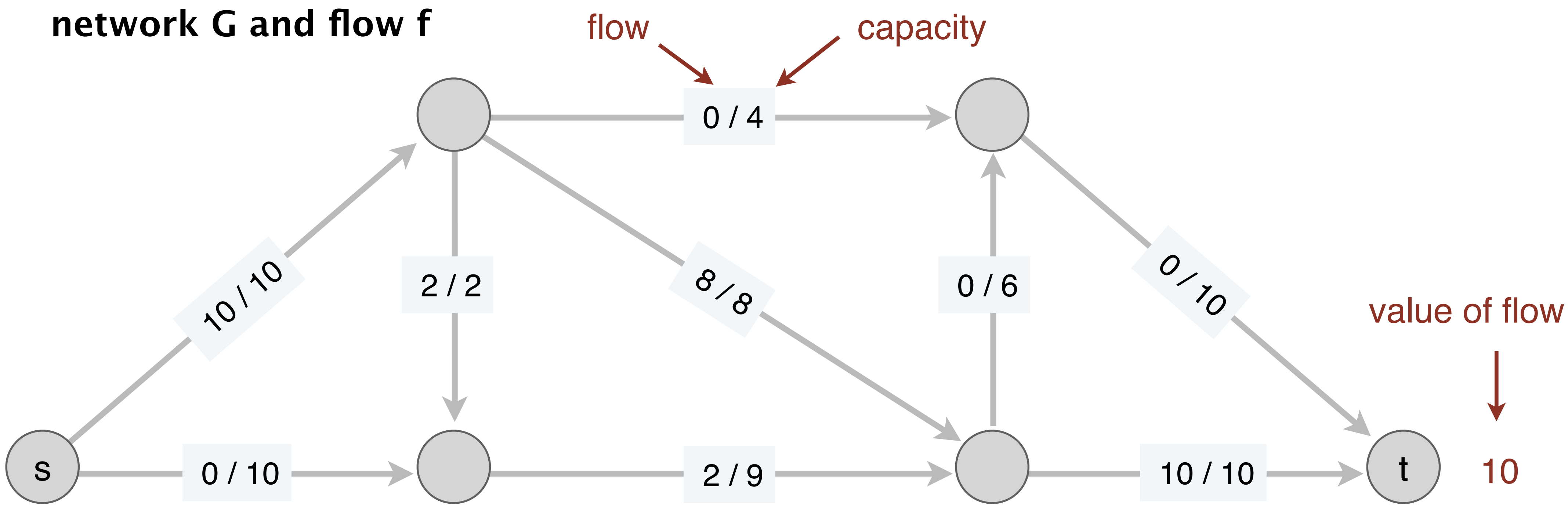
Ford-Fulkerson Example



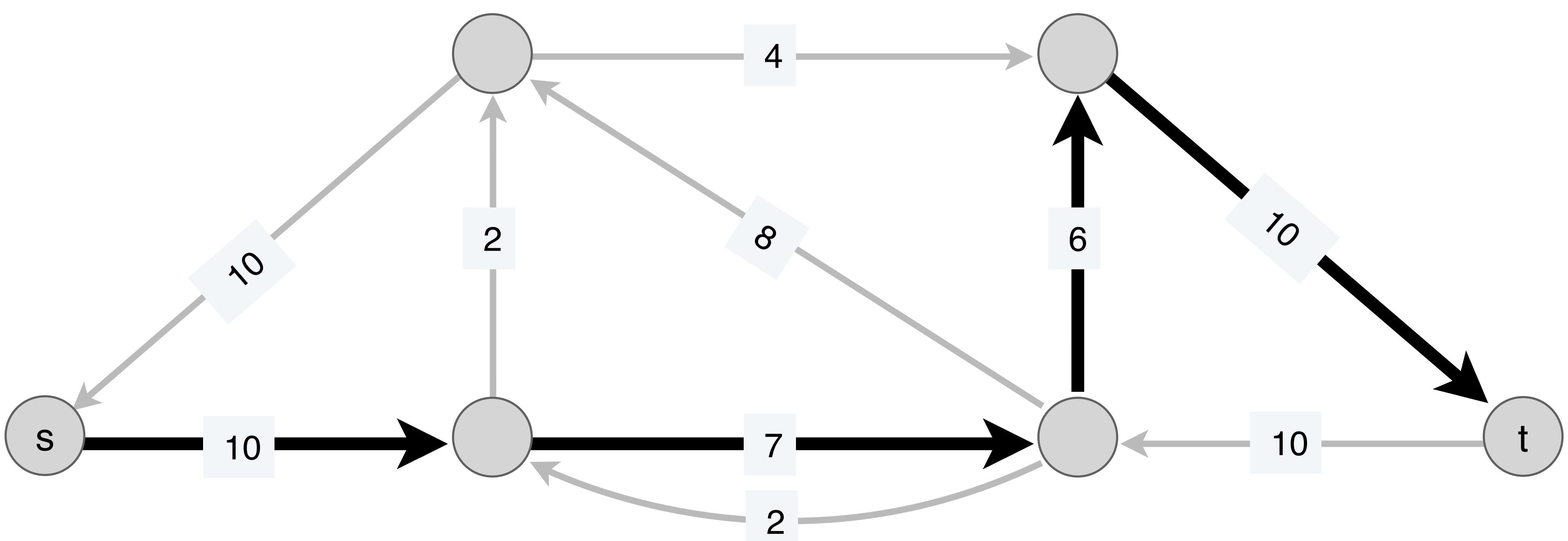


Ford-Fulkerson Example

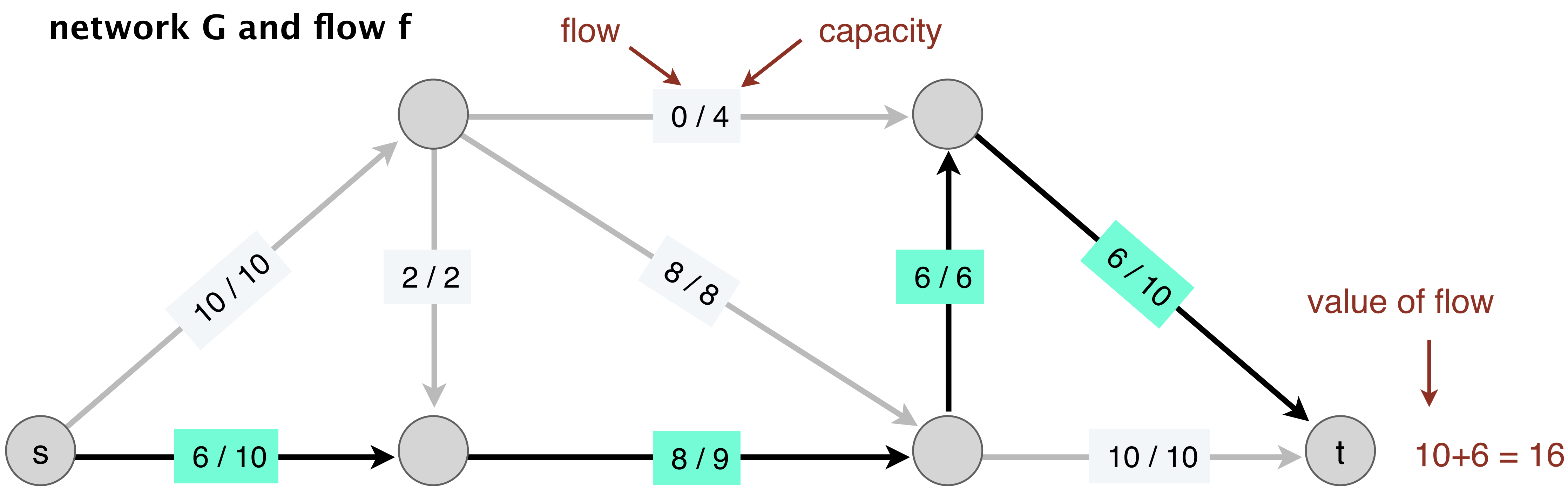
network G and flow f

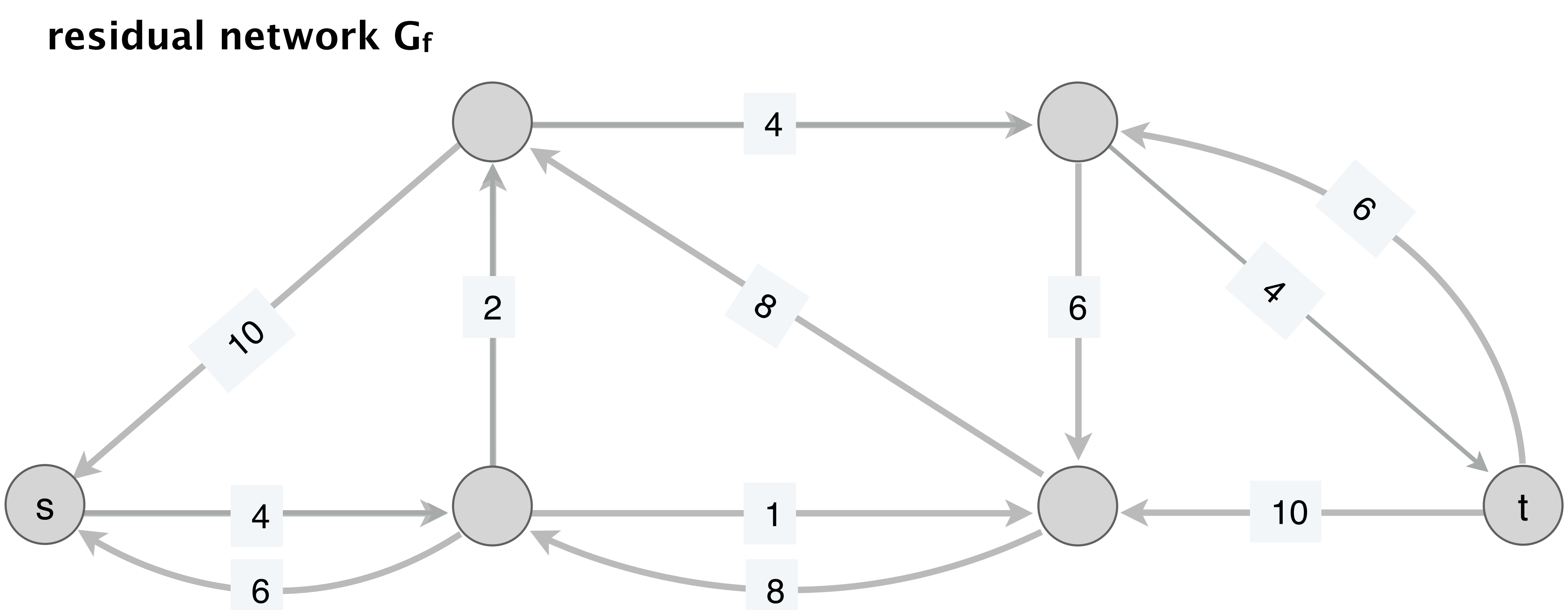


P in residual network G_f

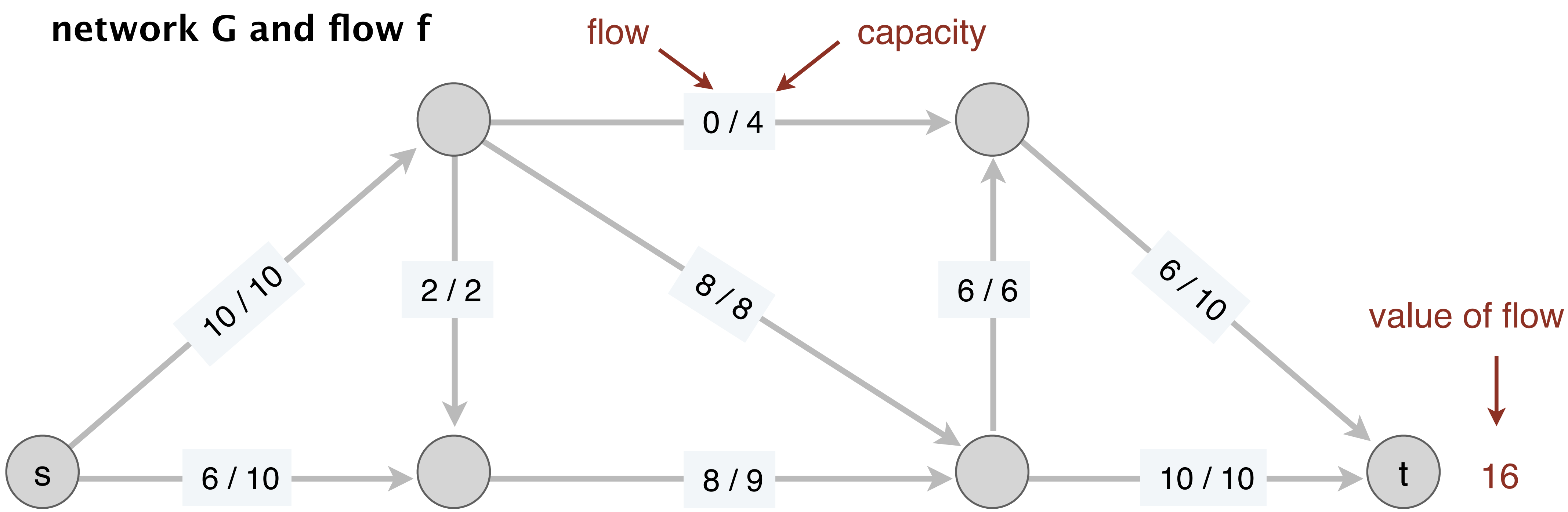


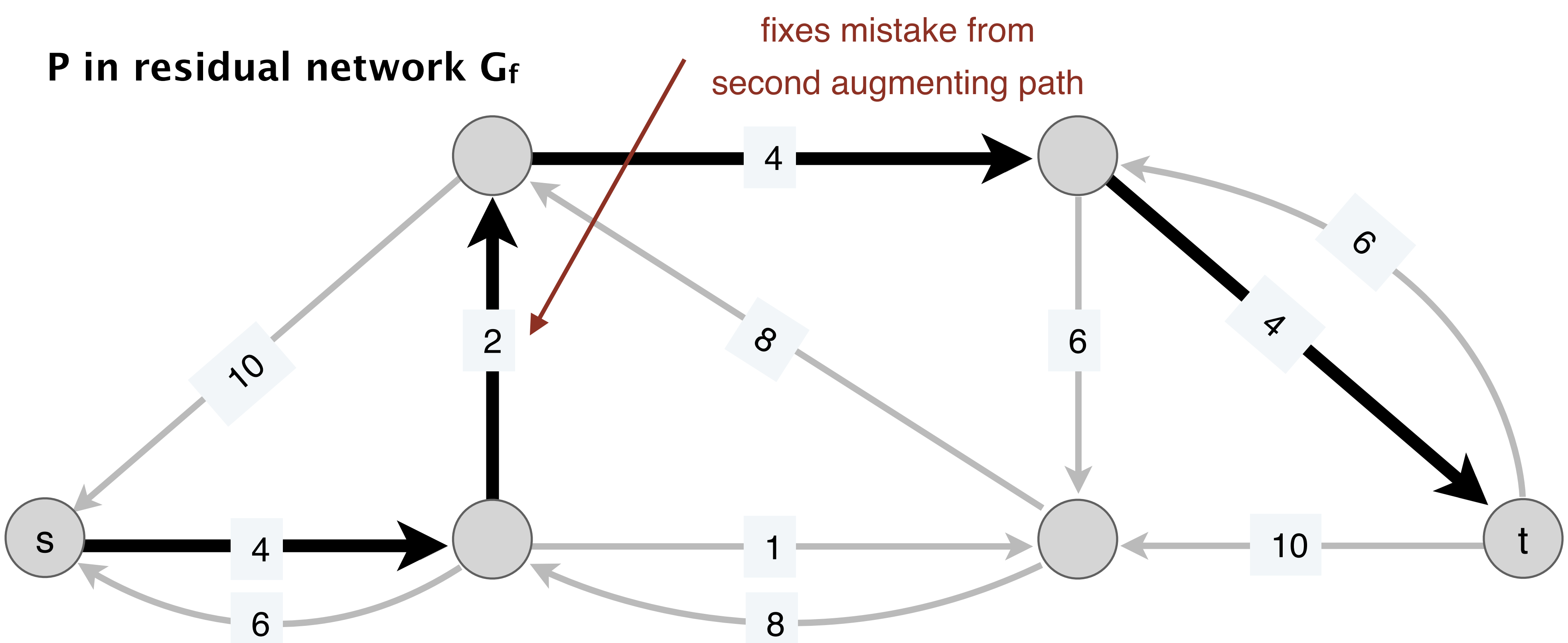
Ford-Fulkerson Example



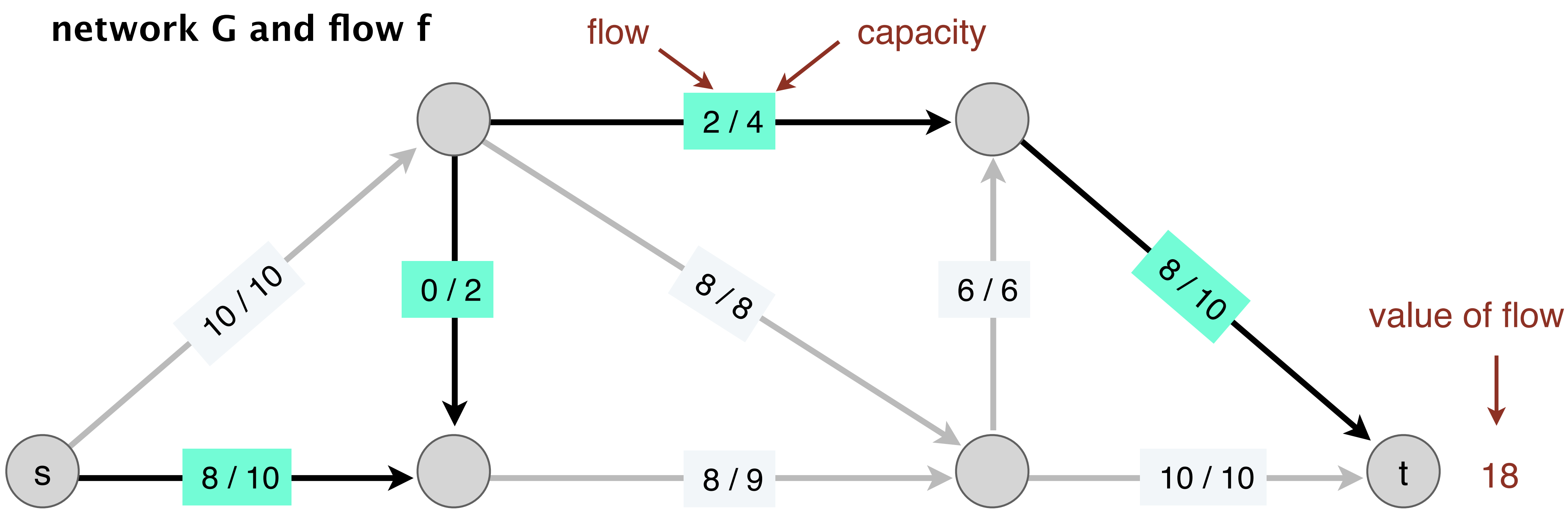


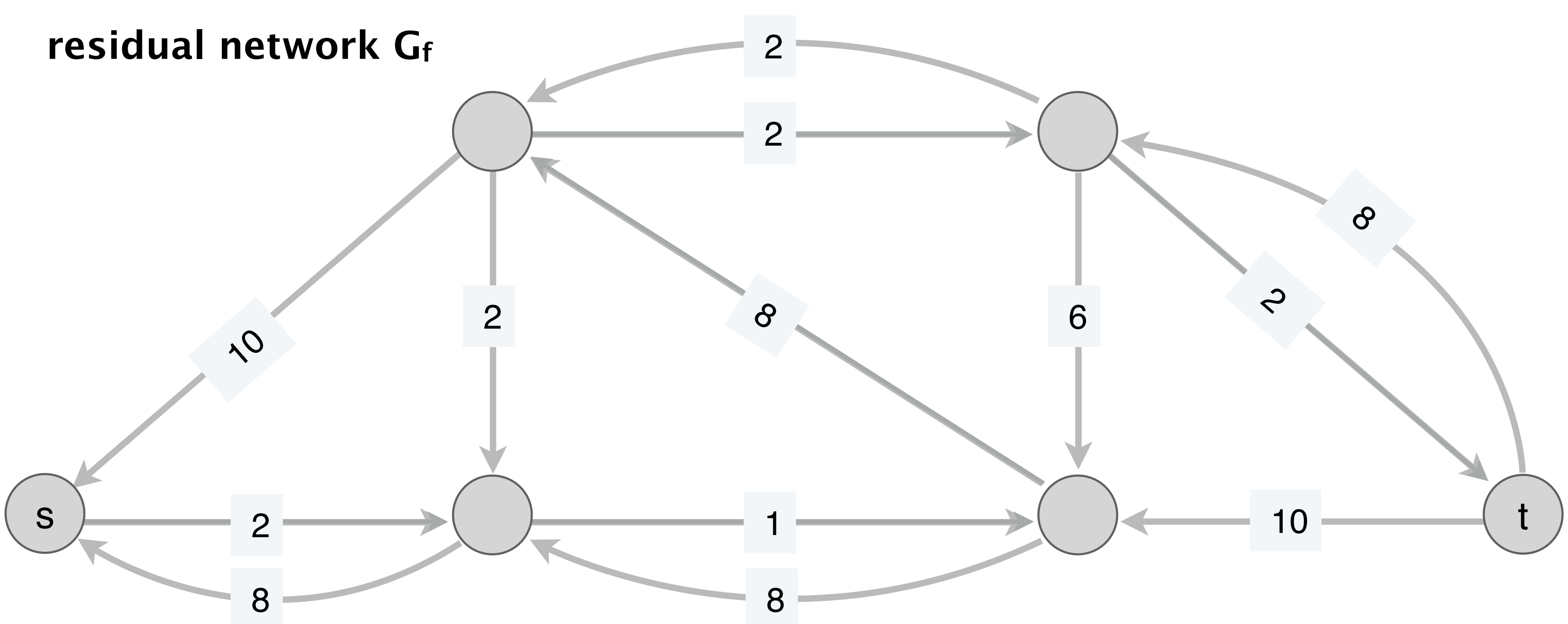
Ford-Fulkerson Example





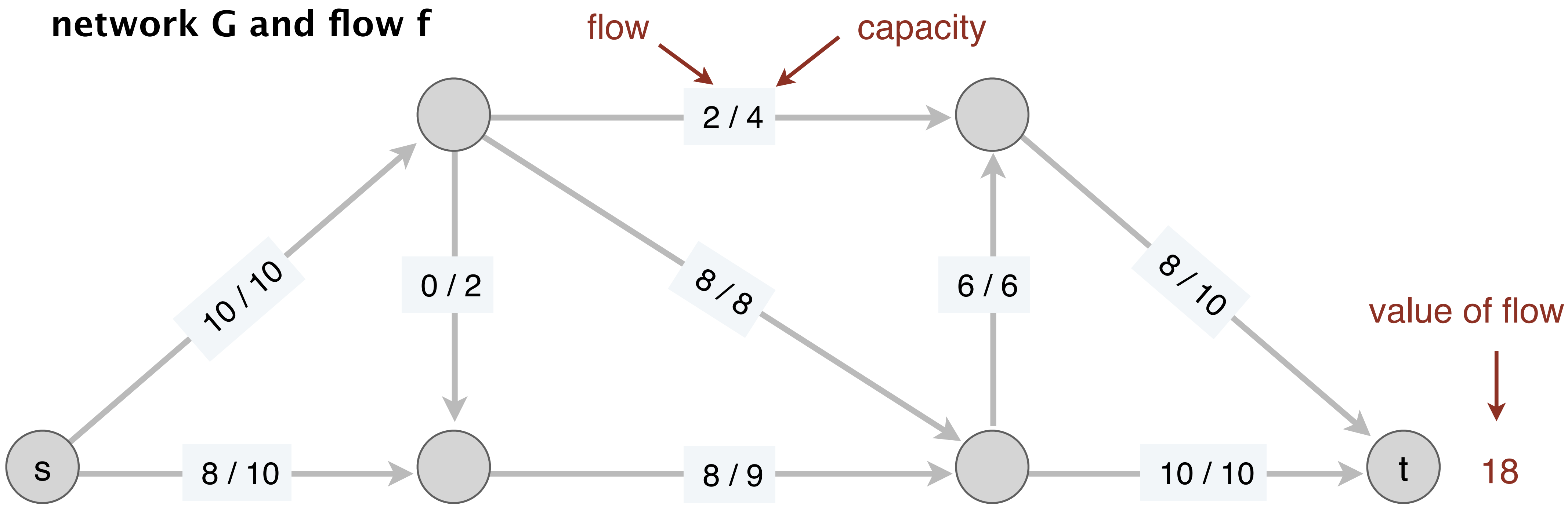
Ford-Fulkerson Example



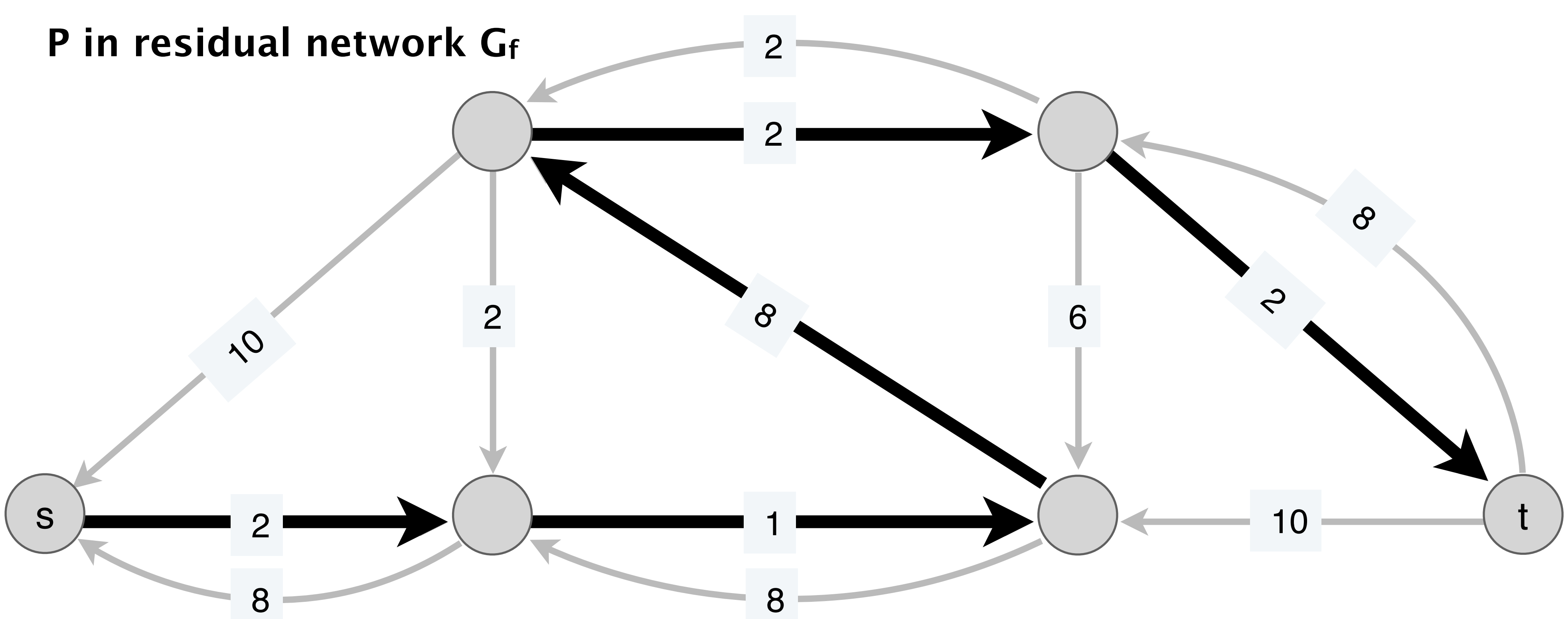


Ford-Fulkerson Example

network G and flow f

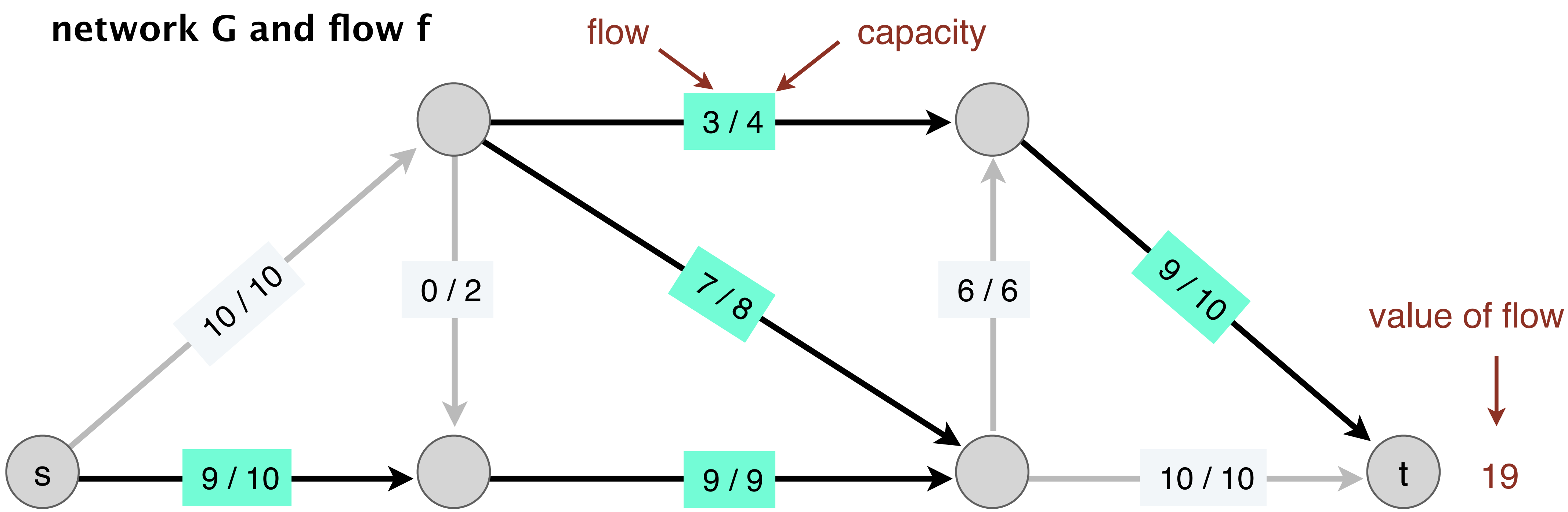


P in residual network G_f

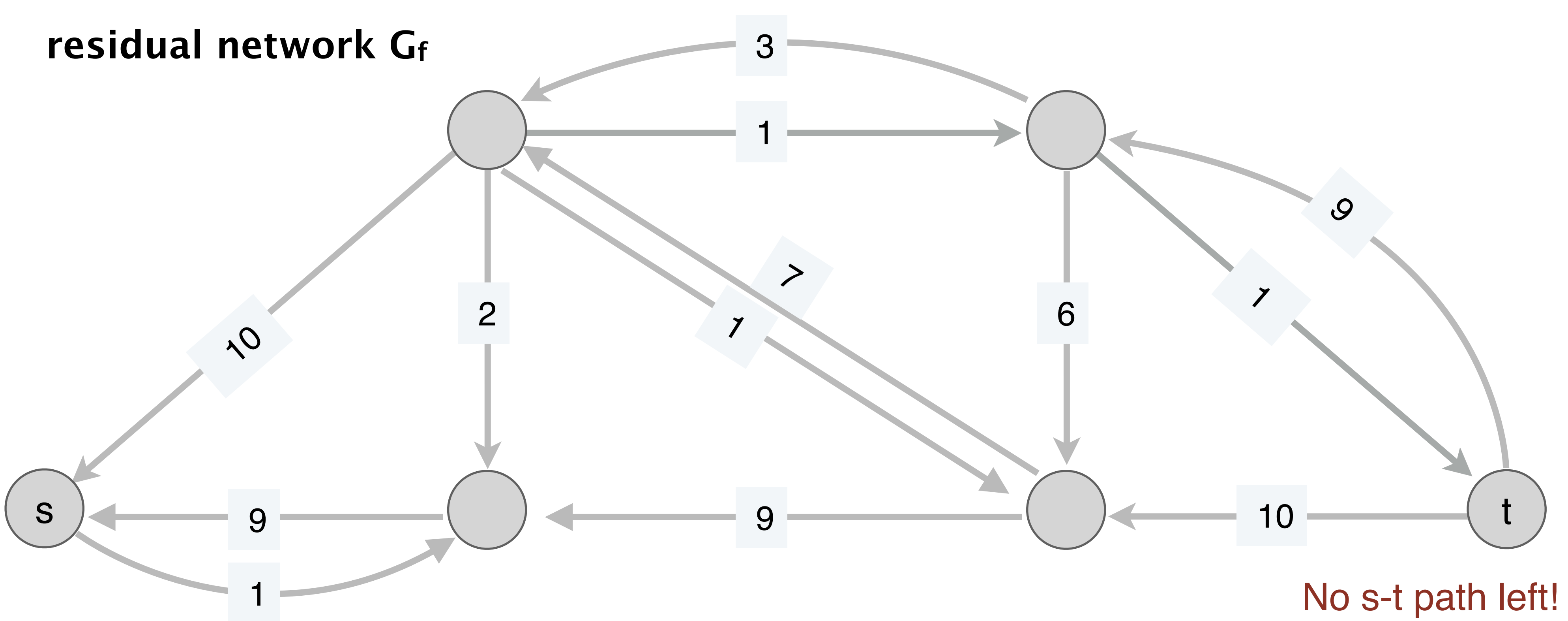


Ford-Fulkerson Example

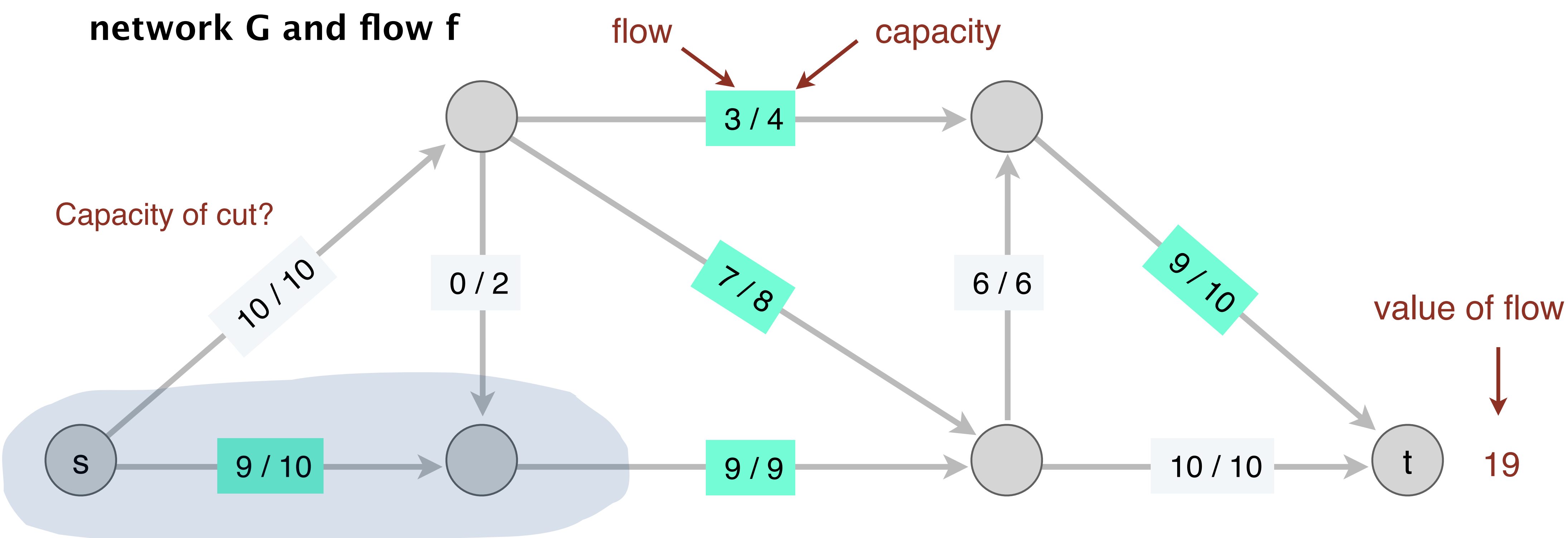
network G and flow f

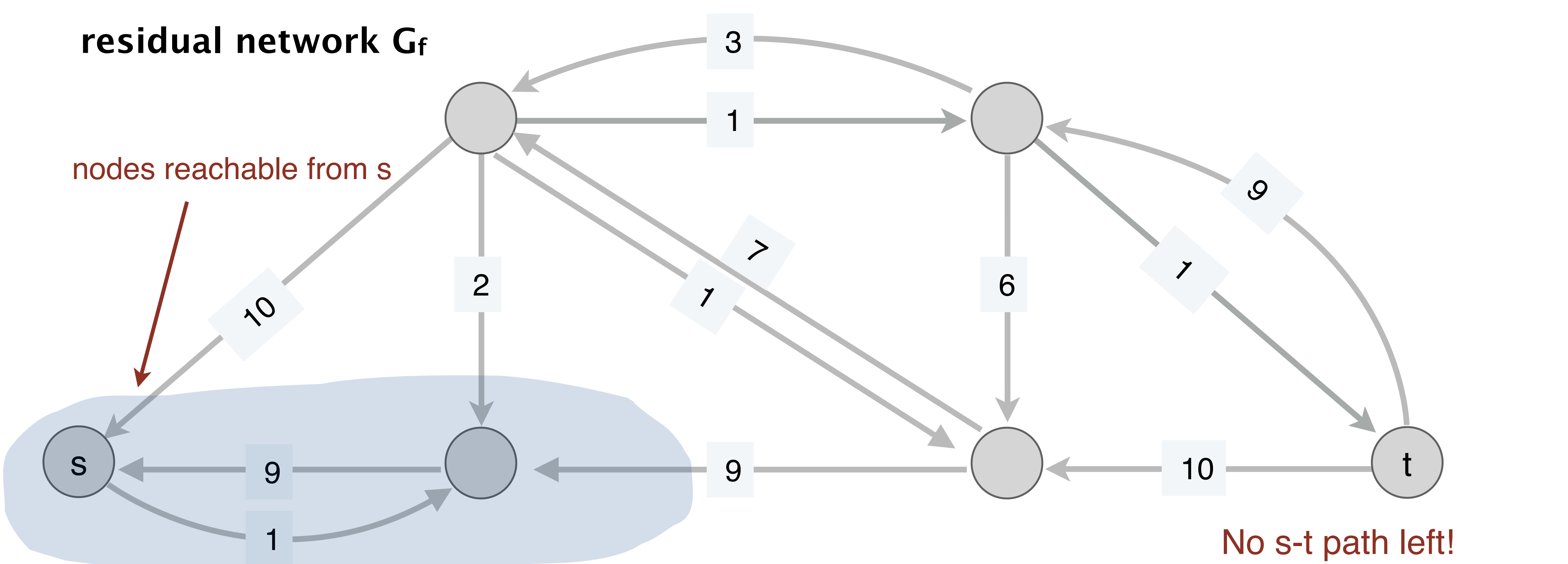


residual network G_f



Ford-Fulkerson Example





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

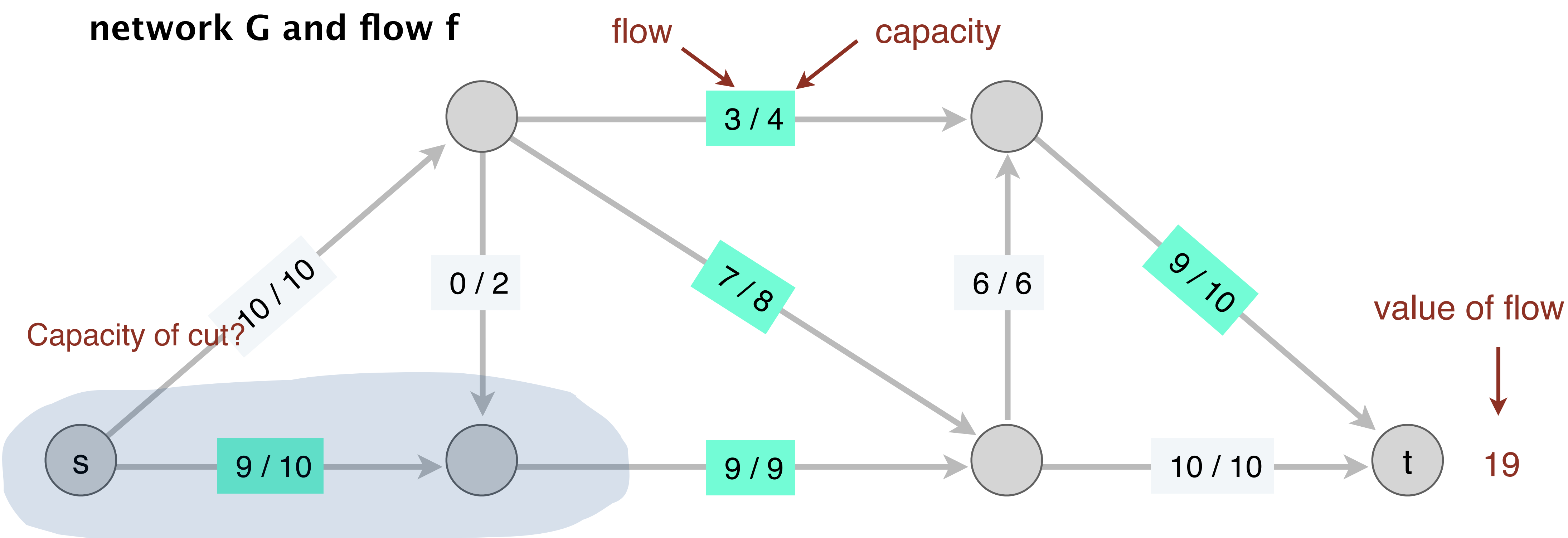
Ford-Fulkerson Optimality

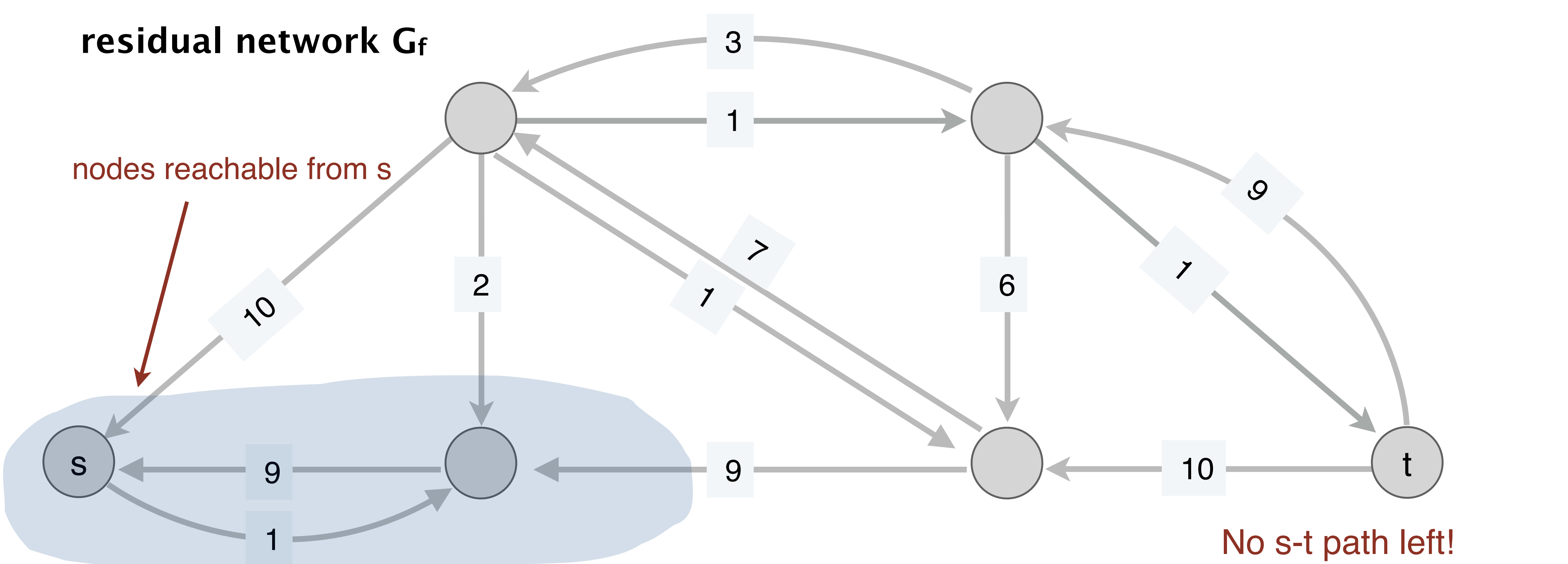
- **Recall**: If f is any feasible s-t flow and (S, T) is any s-t cut then $v(f) \leq c(S, T)$.
- We will show that the Ford-Fulkerson algorithm terminates in a flow that achieves equality, that is,
- Ford-Fulkerson finds a flow f^* and there exists a cut (S^*, T^*) such that, $v(f^*) = c(S^*, T^*)$
- 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 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 $f(e)$?

Recall: Ford-Fulkerson Example





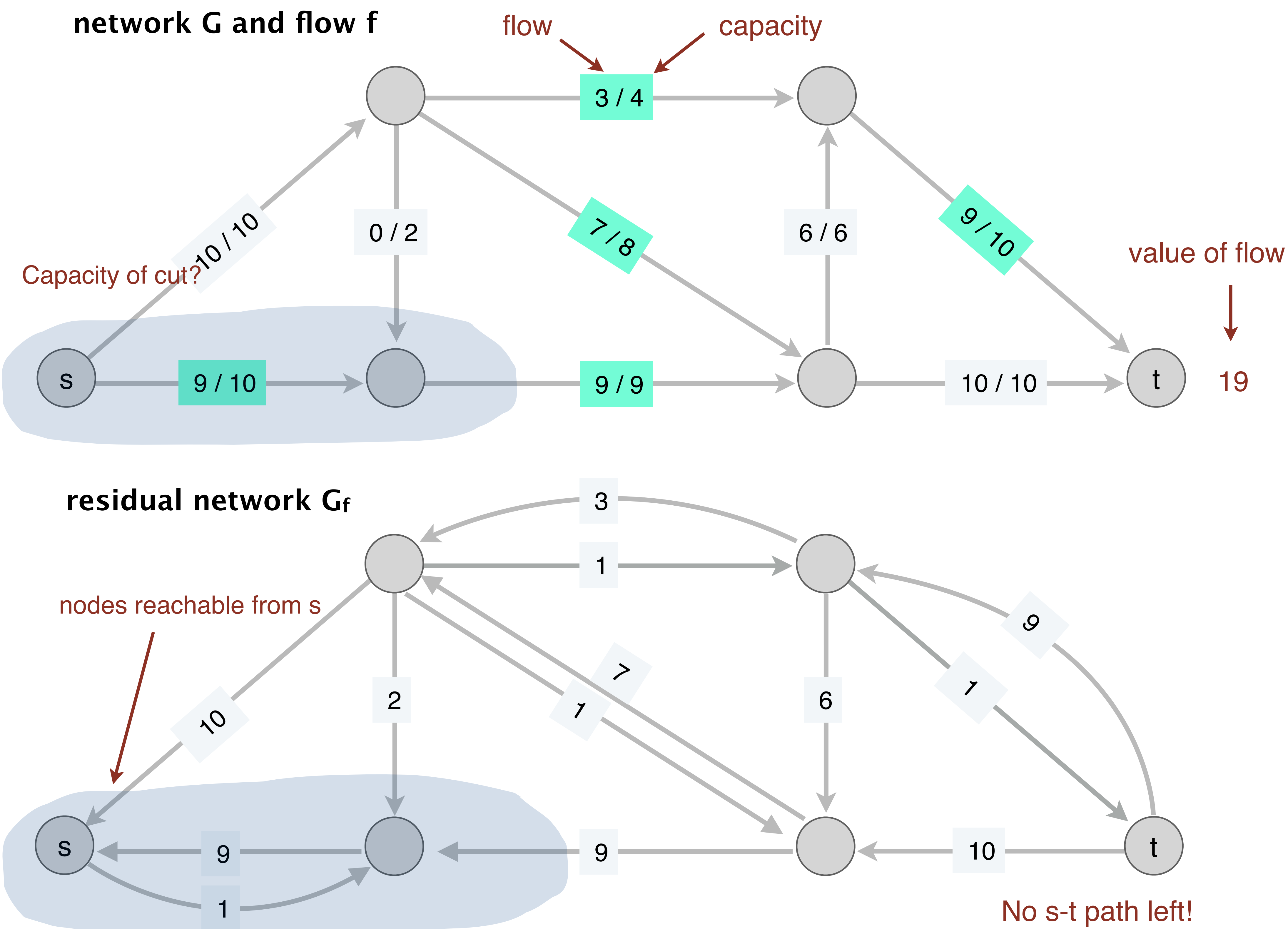
Ford-Fulkerson Optimality

- **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 $S^* = \{v \mid v \text{ is reachable from } s \text{ in } G_f\}$, $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = w \to v$ 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 $S^* = \{v \mid v \text{ is reachable from } s \text{ in } G_f\}$, $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = w \to v$ with $v \in S^*, w \in T^*$, then what can we say about $f(e)$?
  - $f(e) = 0$

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 $S^* = \{v \mid v \text{ is reachable from } s \text{ in } G_f\}$, $T^* = V - S^*$
- 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 $b = \text{bottleneck}(G_f, P)$
- **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 ∎

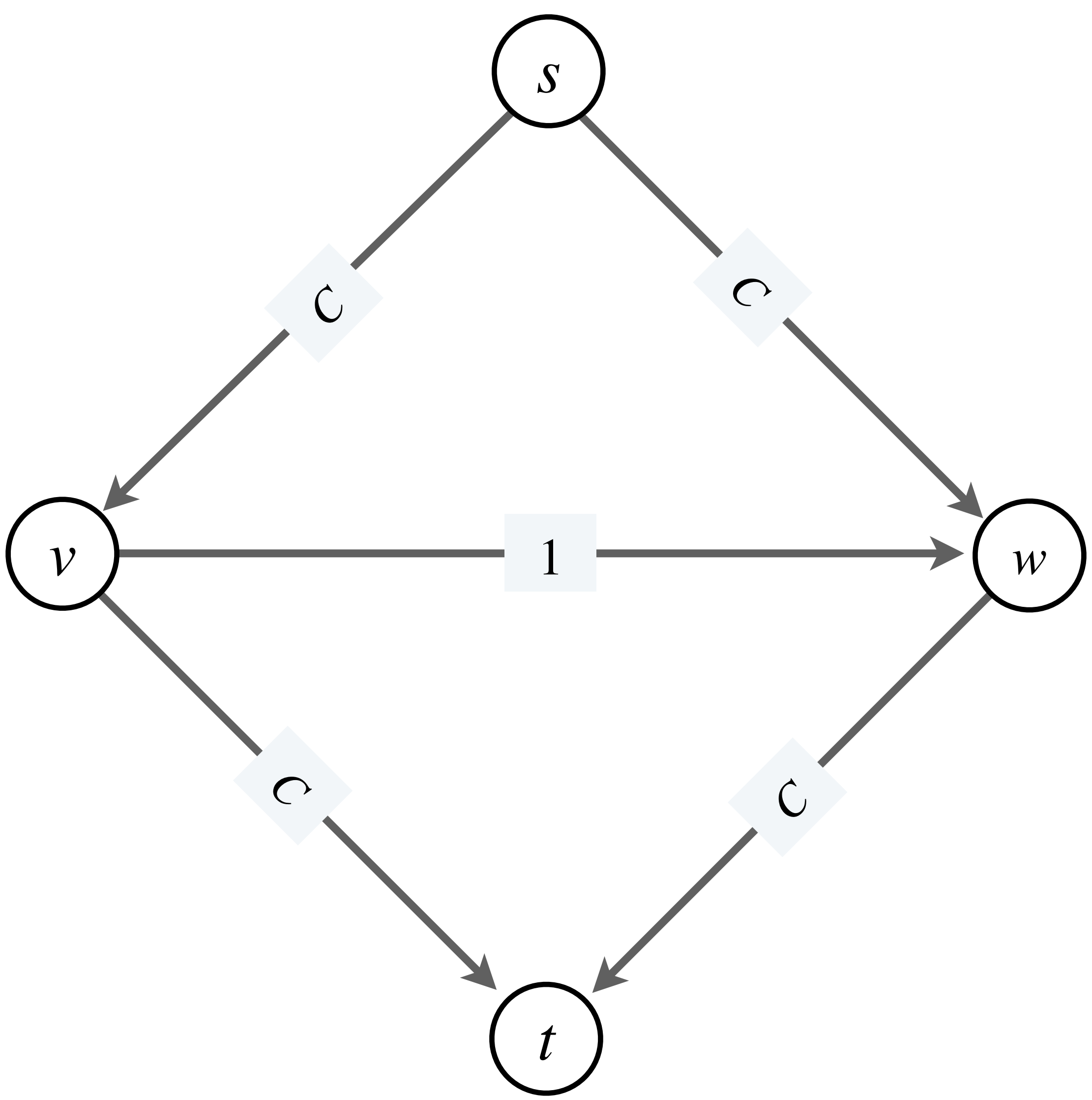
[Digging Deeper] Polynomial time?

- Does the Ford-Fulkerson algorithm run in time polynomial in the input size?
- Running time is $O(nmC)$, where $C = \max_{u} c(s \to u)$
- What is the input size?
    - n vertices, m edges, m capacities
    - C represents the magnitude of the maximum capacity leaving the source node
    - How many bits to represent C?
- Let us take an example

[Digging Deeper] Polynomial time?

- **Question**. Does the Ford-Fulkerson algorithm run in polynomial-time in the size of the input? ← ~ m, n, and $\log C$

- **Answer**. No. if max capacity is C, the algorithm can take $\geq C$ iterations. Consider the following example.



- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- …
- $s \to v \to w \to t$
- $s \to w \to v \to t$

each augmenting path sends only 1 unit of flow (# augmenting paths = $2C$)

[Digger Deeper] Pseudo-Polynomial

- Input graph has n nodes and $m = O(n^2)$ edges, each with capacity c_e
- $C = \max_{e \in E} c(e)$, then $c(e)$ takes $O(\log C)$ bits to represent
- Input size: $\Omega(n \log n + m \log n + m \log C)$ bits
- Running time: $O(nmC) = O(nm2^{\log C})$
    - Exponential in the *size* of C
- Such algorithms are called **pseudo-polynomial**
    - If the running time is polynomial in the **magnitude** but **not size** of an input parameter.
    - We saw this for knapsack as well!

Non-Integral Capacities?

- If the capacities are rational, can just multiply to obtain a large integer (massively increases running time)

- If capacities are irrational, Ford-Fulkerson can run infinitely!

    - Improvement at each step can be arbitrarily small

    - Can create bad instances where it doesn't terminate in finite steps

Network Flow:
Beyond Ford Fulkerson

Edmond and Karp’s Algorithms

- Ford and Fulkerson’s algorithm does not specify which path in the residual graph to augment
- Poor worst-case behavior of the algorithm can be blamed on bad choices on augmenting path
- **Better choice of augmenting paths.** In 1970s, Jack Edmonds and Richard Karp published two natural rules for choosing augmenting paths
    - Widest path first: paths with largest bottleneck capacity
    - Shortest (in terms of edges) augmenting paths first (Dinitz independently discovered & analyzed this rule)

Widest Augmenting Paths First

- Ford Fulkerson can be improved with a greedy algorithm way of choosing augmenting paths:
  - Choose the augmenting path with largest bottleneck capacity
- Largest bottleneck path can be computed in $O(m \log n)$ time in a directed graph
  - Similar to Dijkstra's analysis
- How many iterations if we use this rule?
  - Won't prove this: but takes $O(m \log C)$ iterations
- Overall running time is $O(m^2 \log n \log C)$ (polynomial time!)
  - Still depends on C though

Shortest Augmenting Paths First

- Choose the augmenting path with the smallest # of edges
- Can be found using BFS on G_f in $O(m + n) = O(m)$ time
- Surprisingly, this resulting a polynomial-time algorithm independent of the actual edge capacities !
- Analysis looks at "level" of vertices in the BFS tree of G_f rooted at s —levels only grow over time
- Analyzes # of times an edge $u \to v$ disappears from G_f
- Takes $O(mn)$ iterations overall
- Thus overall running time is $O(m^2 n)$

Progress on Network Flows

1951	$O(m\, n^2\, C)$	Dantzig
1955	$O(m\, n\, C)$	Ford–Fulkerson
1970	$O(m\, n^2)$	Edmonds–Karp, Dinitz
1974	$O(n^3)$	Karzanov
1983	$O(m\, n \log n)$	Sleator–Tarjan
1985	$O(m\, n \log C)$	Gabow
1988	$O(m\, n \log (n^2 / m))$	Goldberg–Tarjan
1998	$O(m^{3/2} \log (n^2 / m) \log C)$	Goldberg–Rao
2013	$O(m\, n)$	Orlin
2014	$\tilde{O}(m\, n^{1/2} \log C)$	Lee–Sidford
2016	$\tilde{O}(m^{10/7}\, C^{1/7})$	Mądry
2017	$\tilde{O}(m^{10/7} \log W)$	Cohen-Mądr

For unit capacity networks

Summary

- Given a flow network with integer capacities, the **maximum flow and minimum cut** can be computed in $O(mn)$ time.
- **Next.** Network flow applications!

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)