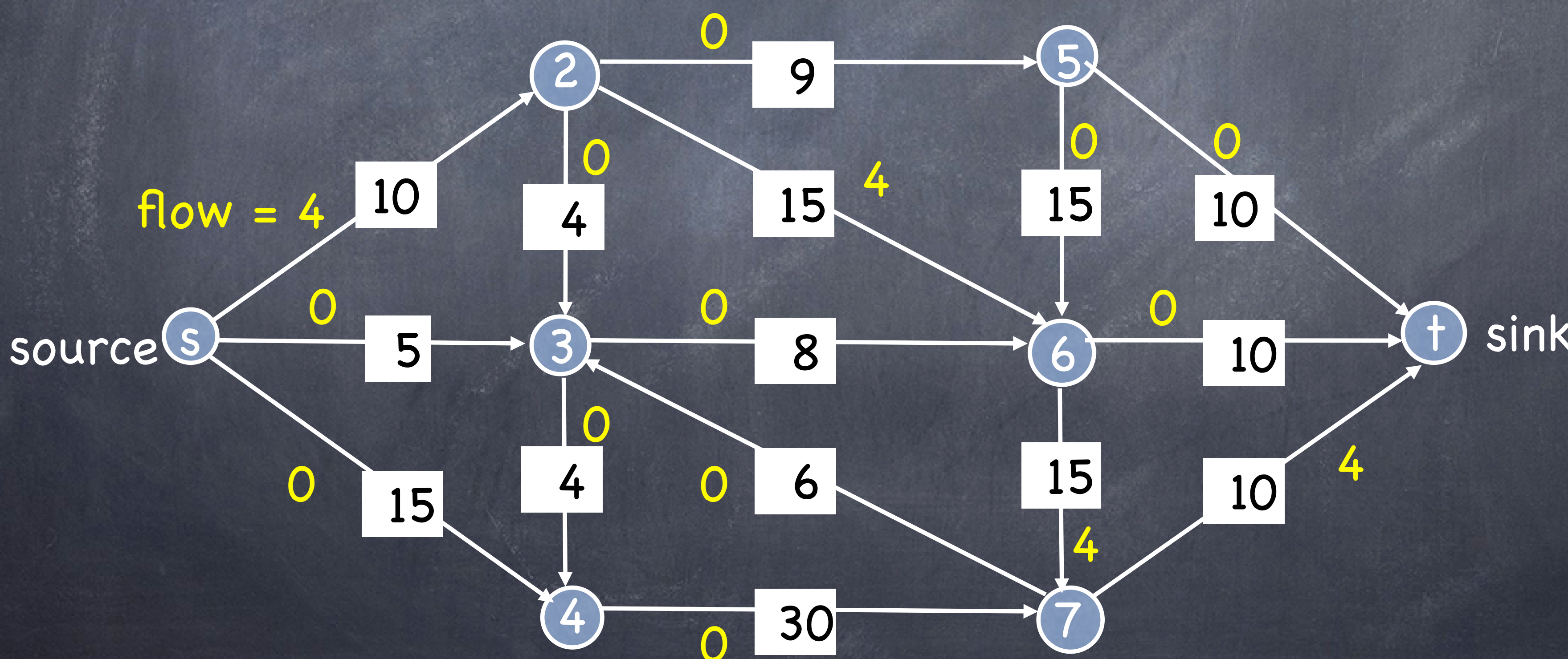
Today

- Flow review
- Augmenting paths
- Ford-Fulkerson Algorithm
- Intro to cuts (reason: prove correctness)

Flow Networks

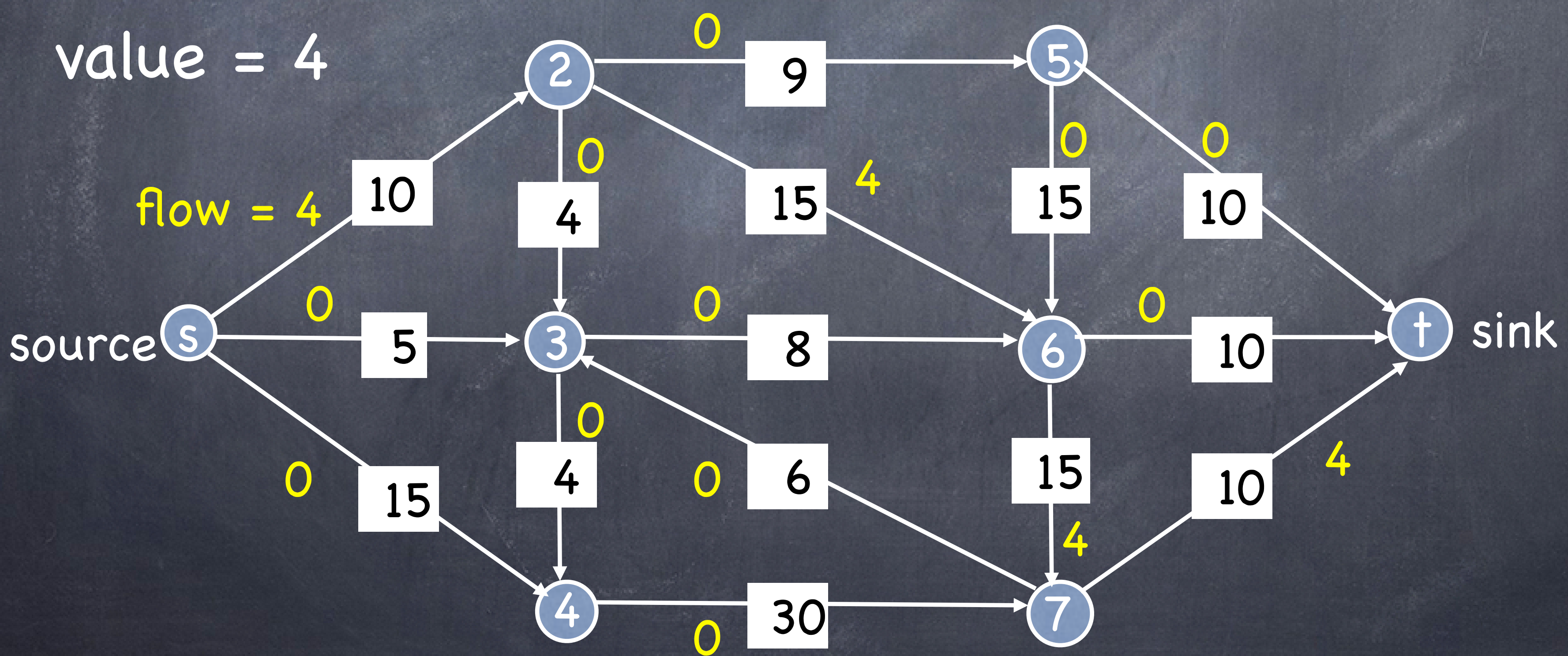
- s = source, t = sink.
- c(e) = capacity of edge e
- Capacity condition: $0 \leq f(e) \leq c(e)$
- Conservation condition: for $v \in V - \{s, t\}$:

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$



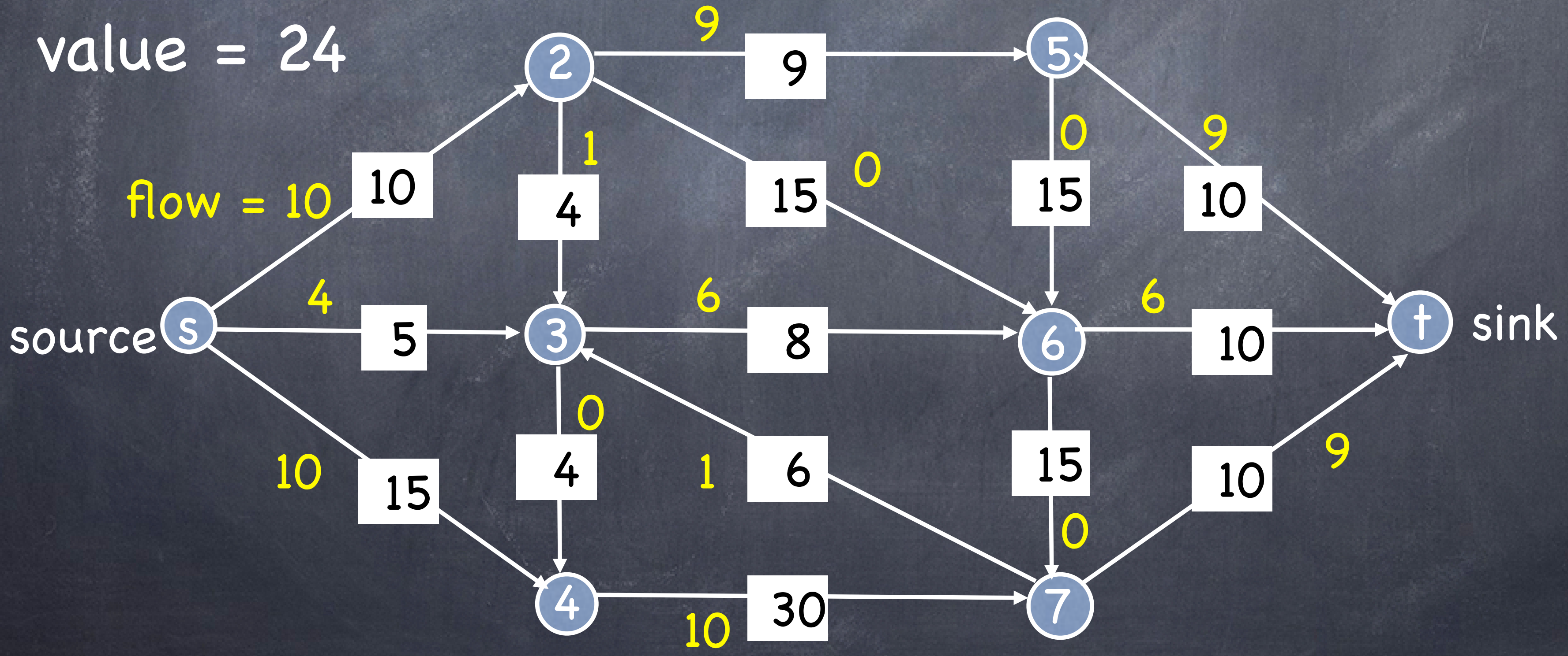
Flows

- The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$



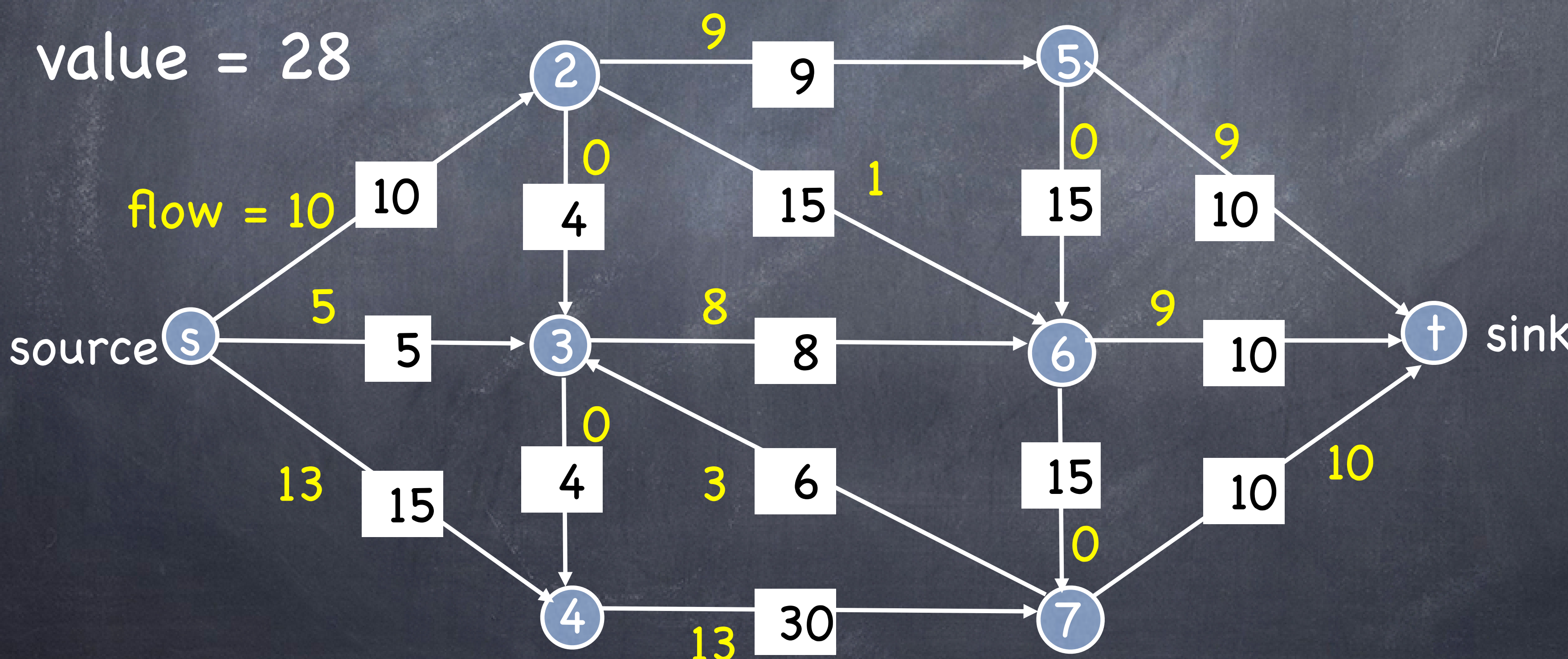
Flows

- The **value of a flow** f is: $v(f) = \sum_{e \text{ out of } s} f(e)$



Maximum Flow Problem

Find s-t flow of maximum value.

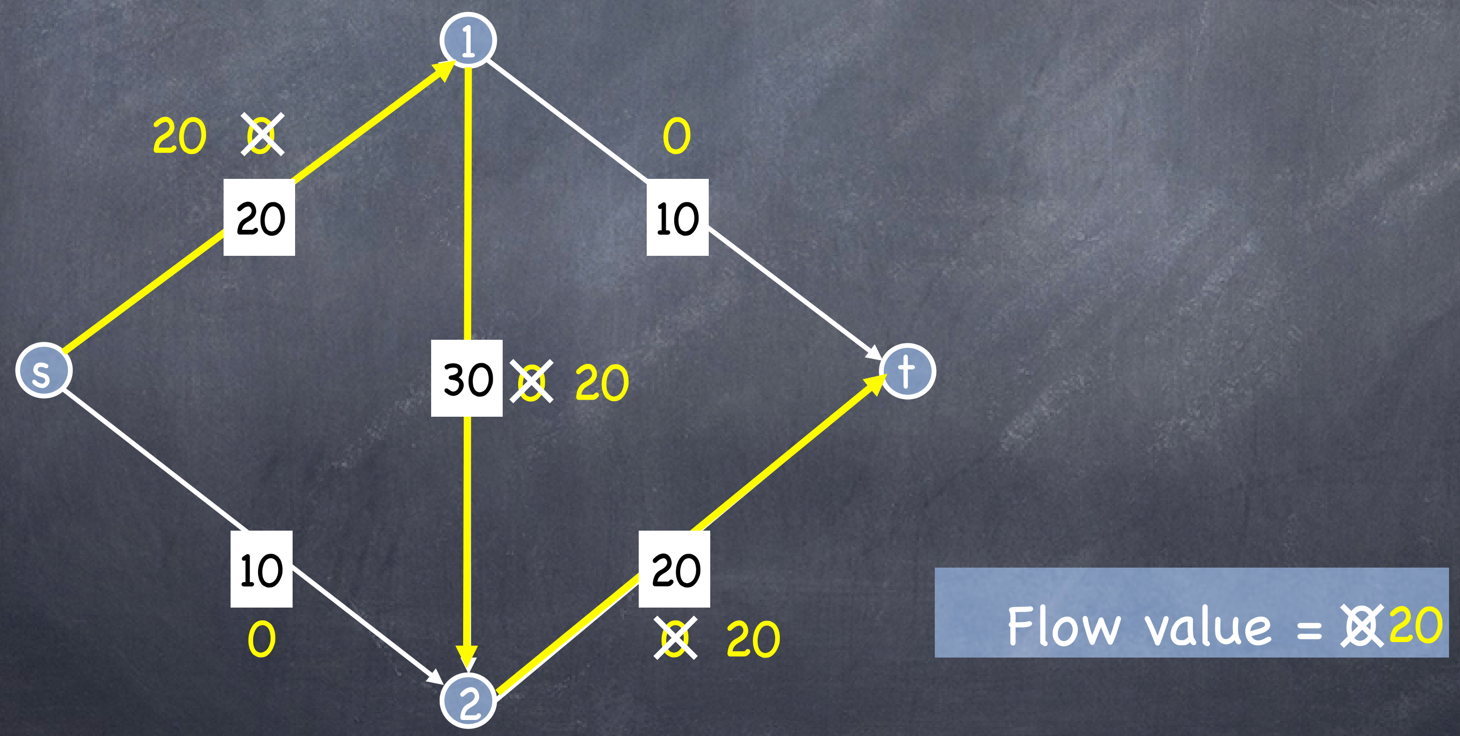


Towards a Max-Flow Algorithm

Key idea: repeatedly choose paths and "augment" the amount of flow on those paths as much as possible until capacities are met

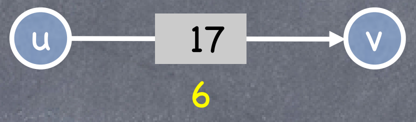
Towards a Max Flow Algorithm

- Problem: possible to get stuck at a flow that is not maximum, no more paths with excess capacity

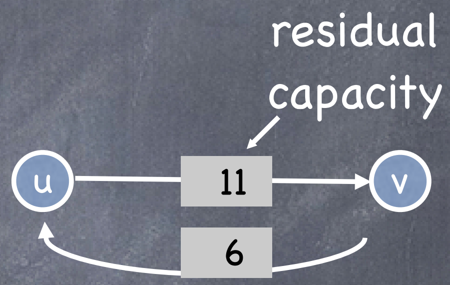


Residual Graph

- Original edge: $e = (u, v) \in E$.
  - Flow $f(e)$, capacity $c(e)$.



- Create two residual edges
  - Forward edge
    $e = (u, v)$ with capacity $c(e) - f(e)$
  - Backward/reverse edge
    $e' = (v, u)$ with capacity $f(e)$



- Residual graph: $G_f = (V, E_f)$.
  - E_f = edges with positive residual capacity
  - $E_f = \{e : f(e) < c(e)\} \cup \{e' : f(e) > 0\}$

Augmenting Path

- **Definition:** an s-t path P in G_f is an **augmenting path**
- **Idea:** use an augmenting path to augment flow in G
  - Increase flow on forward edges
  - Decrease flow on backward edges
- **Definition:** let bottleneck(P, f) be the minimum residual capacity (i.e., capacity in G_f) of any edge in P

Example on board

Augmenting Path

Use path P in G_f to to update flow f

```
Augment(f, P) {
    b = bottleneck(P, f)    // edge on P with least residual capacity
    foreach e = (u,v) ∈ P {
        if e is a forward edge
            f(e) = f(e) + b        // forward edge: increase flow
        else
            let e′ = (v, u)
            f(e′) = f(e′) - b      // backward edge: decrease flow
    }
    return f
}
```

Augmenting Path

Claim: Let f be a flow and let f′ = Augment(f, P). Then f′ is a flow.

Proof idea: verify capacity and conservation conditions
1) Capacity: by design of residual graph
2) Conservation: check that net change at each node is zero

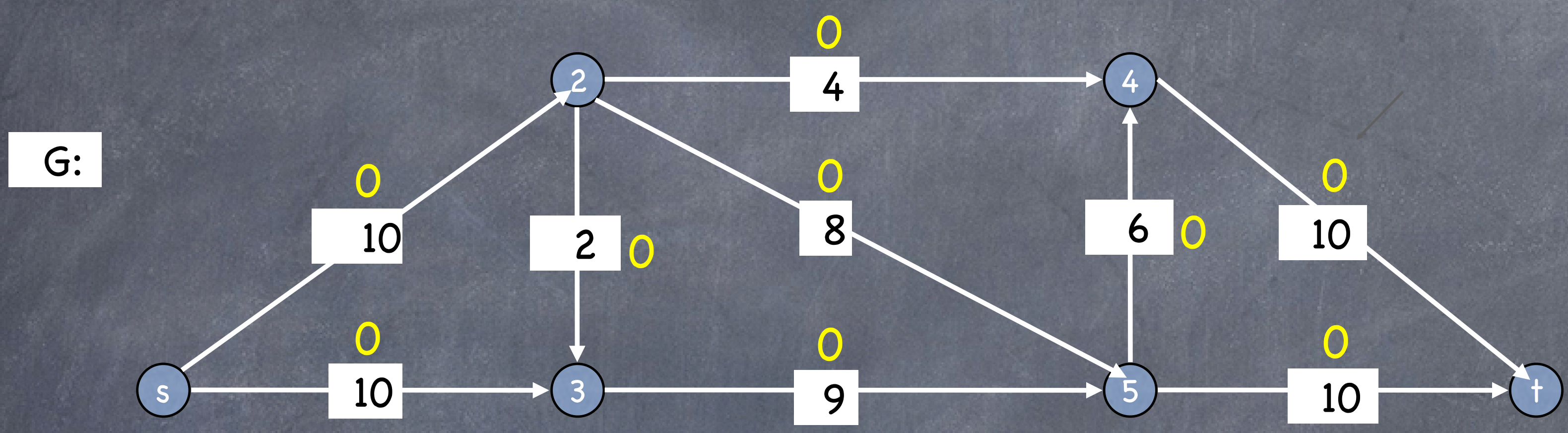
Proof sketch on board

Ford-Fulkerson Algorithm

Repeat: find an augmenting path, and augment!

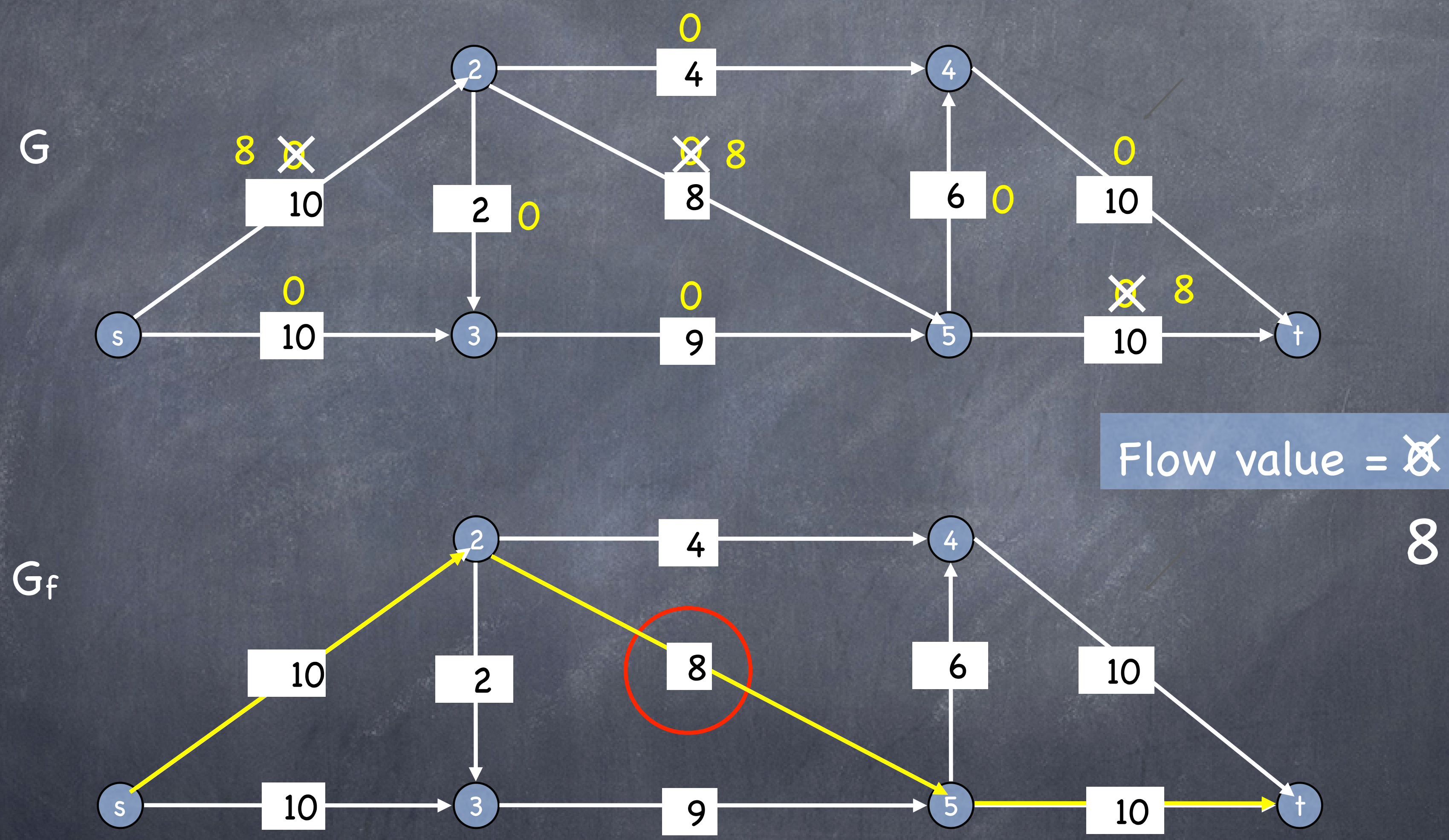
```
Ford-Fulkerson(G, s, t) {
   foreach e ∈ E  f(e) = 0 // initially, no flow
   Gf = copy of G          // residual graph = original graph

   while (there exists an s-t path P in Gf) {
      f = Augment(f, P)    // change the flow
      update Gf            // build a new residual graph
   }
   return f
}
```

Example



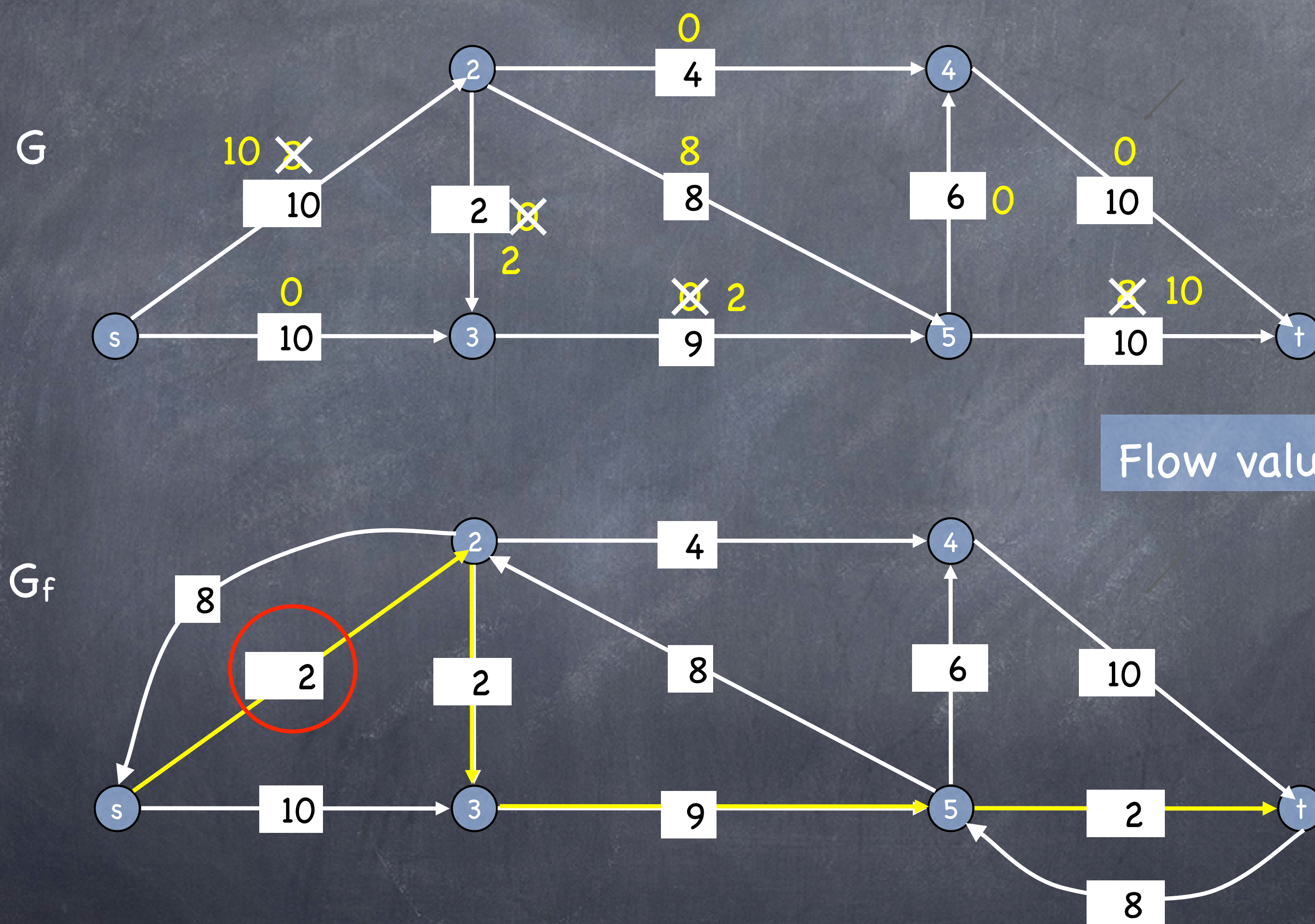
Flow value = 0

Example



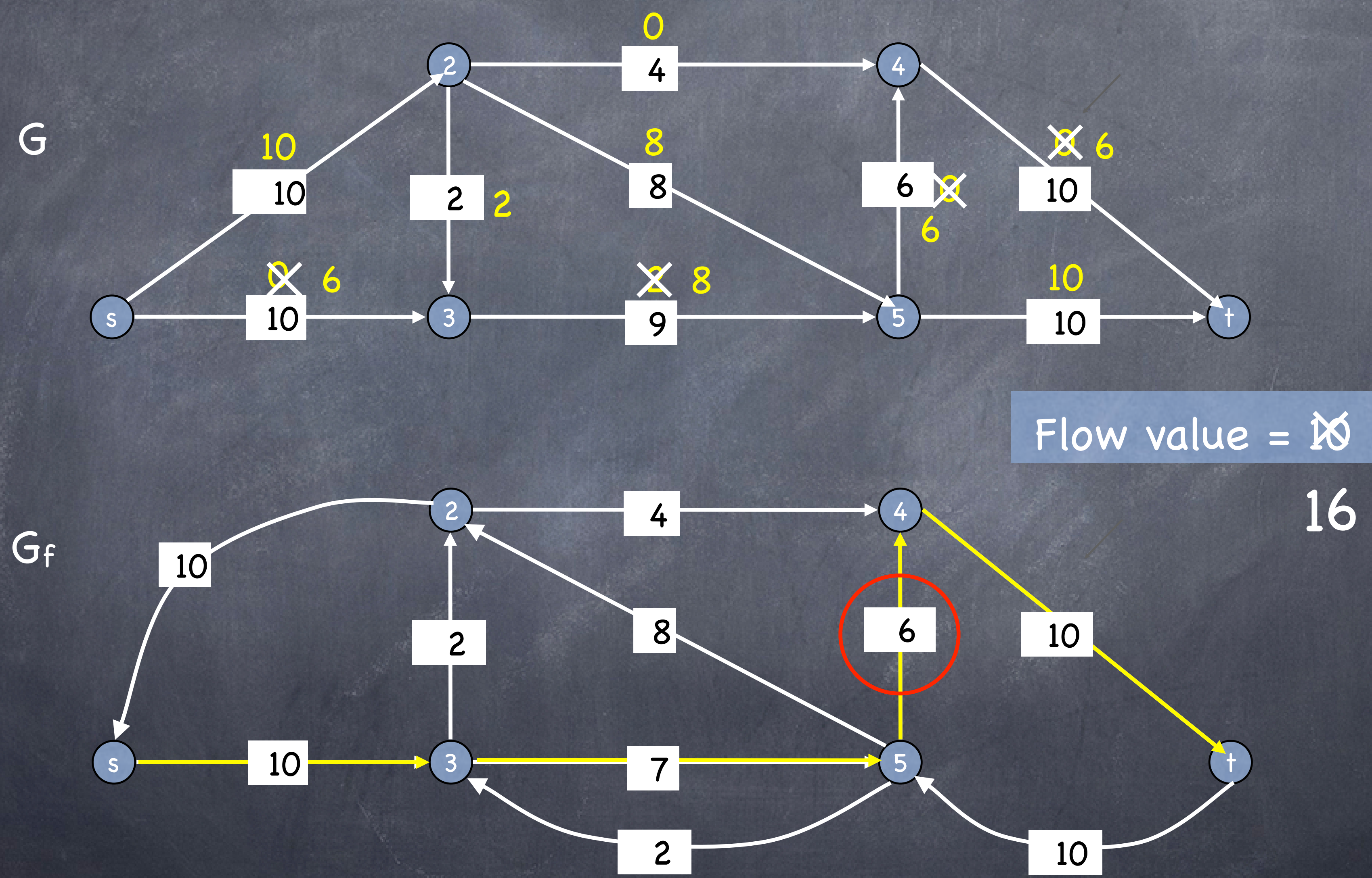
G
2
4
s
3
5
t
0
4
8
0
10
2
0
0
8
8
6
0
0
10
0
10
0
9
0
8
10
Flow value = 0
8
Gf
2
4
s
3
5
t
4
10
2
8
6
10
10
9
10

Example

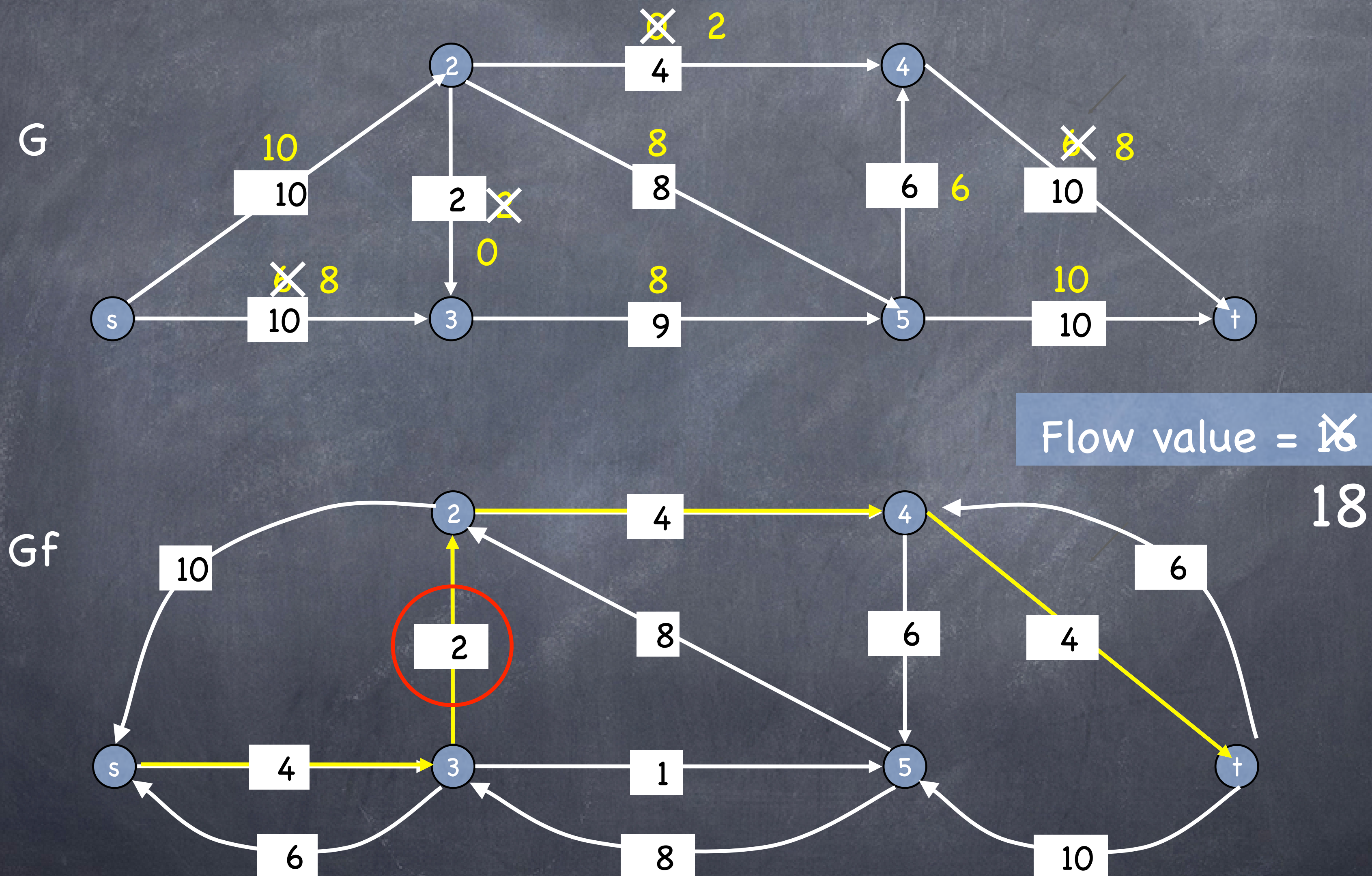


Flow value = ~~8~~ 10

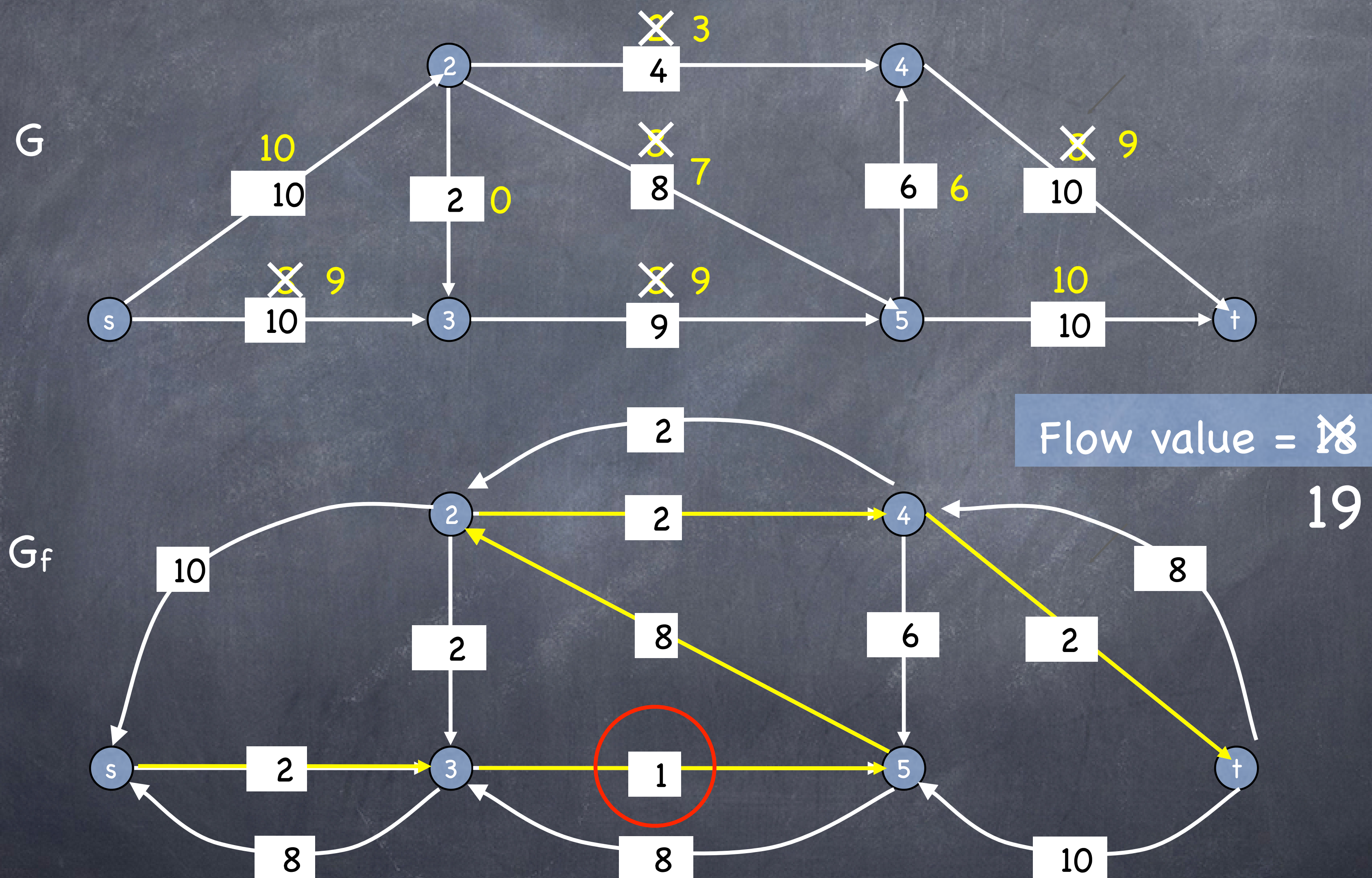
Example



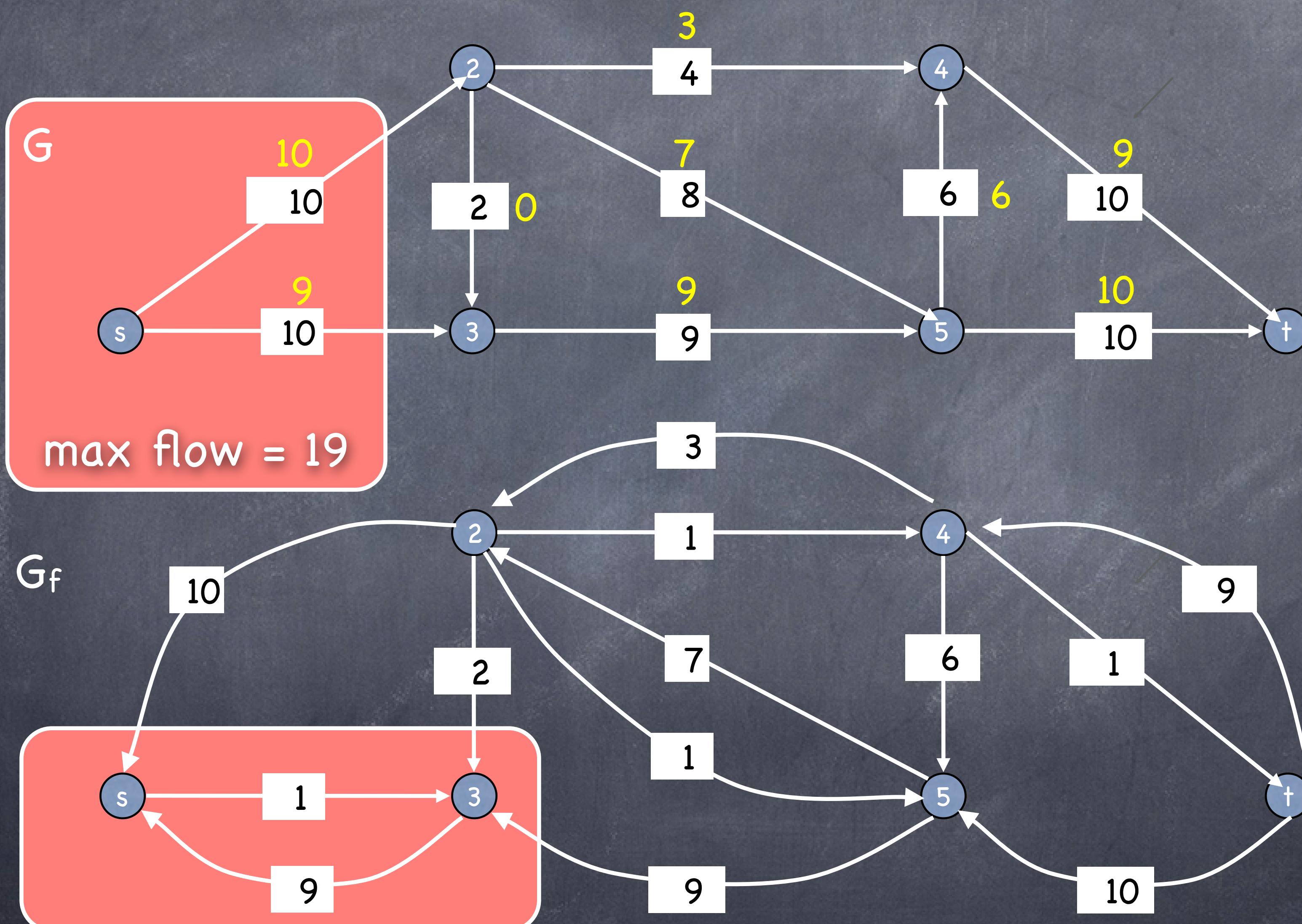
Example



Example



Example



G
max flow = 19
Gf
s
2
3
4
5
t
3
4
10
10
2
0
7
8
6
6
9
10
9
10
9
9
10
10
3
1
10
2
7
6
1
9
1
1
9
9
10

Termination

- Assumption. All capacities are positive integers.
- Invariant. Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.
- Theorem. Let OPT = value of max flow. The algorithm terminates in at most OPT iterations, with OPT $\leq$ C, the total capacity of the edges leaving the source.
- Proof?

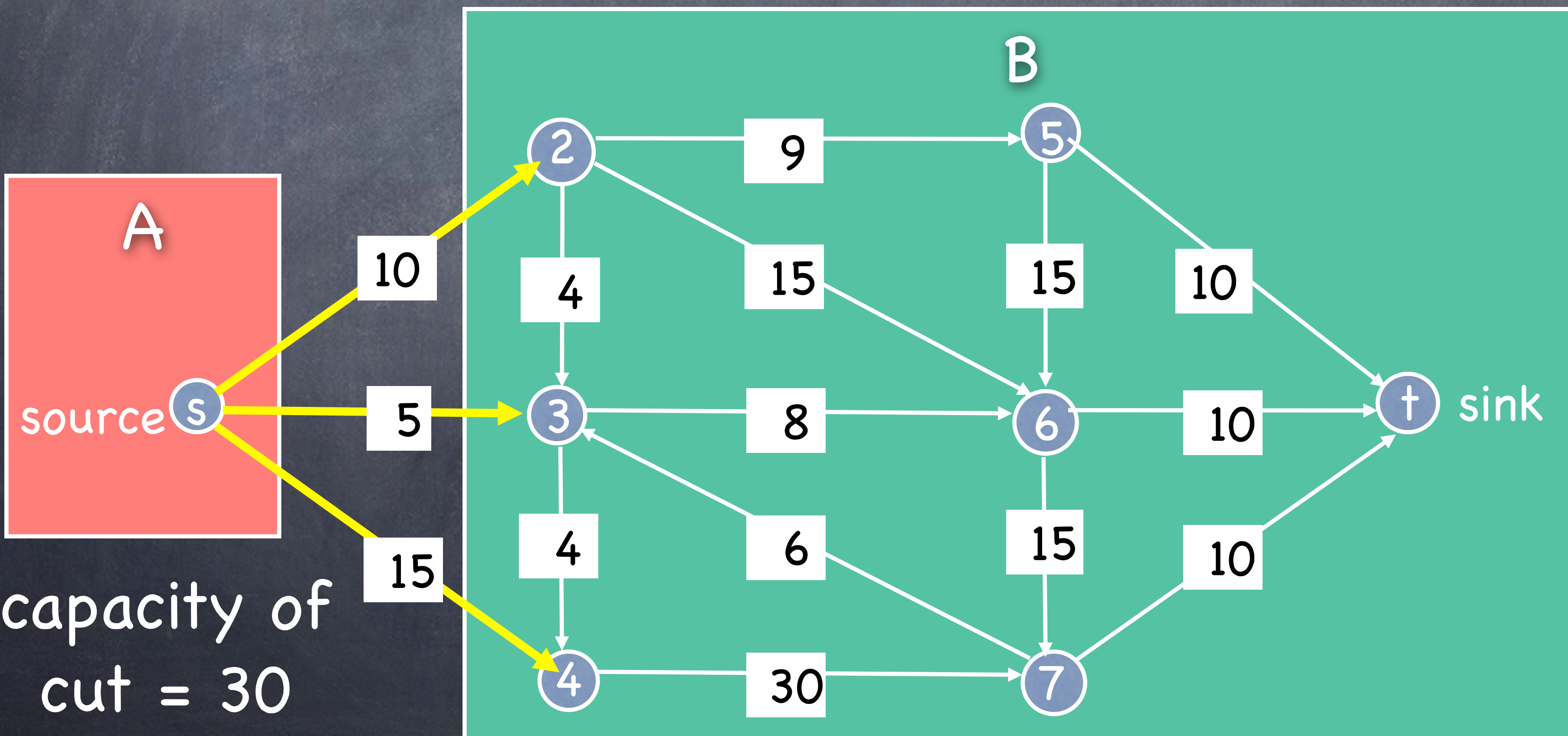
Running Time?

- There are at most C augment operations. How long does it take for each?
  - Find a residual path O(m+n)
  - Compute bottleneck capacity O(m)
  - Update flow O(m)
  - Update residual graph O(m)

Total running time: O(C(m+n))

Cuts

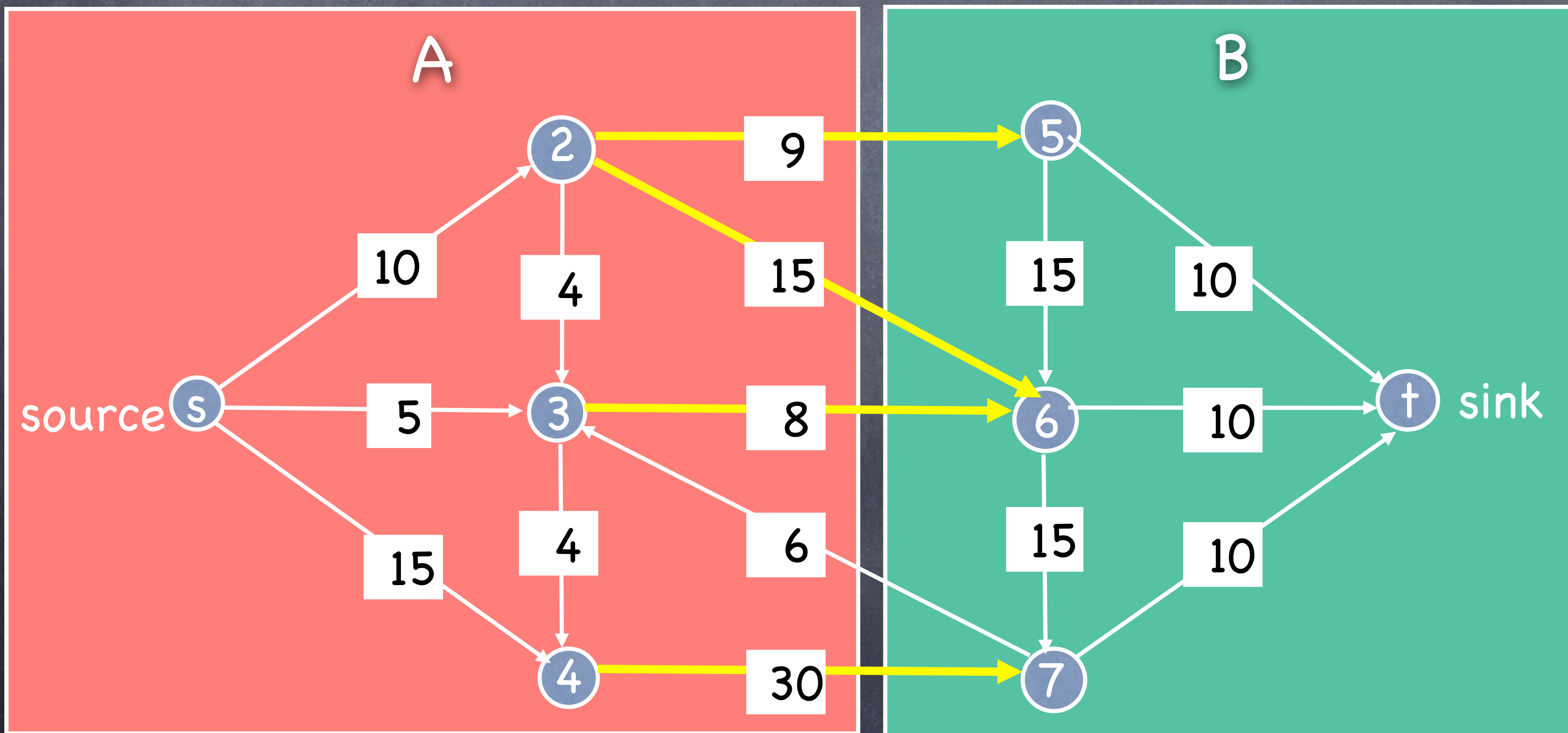
- An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$.
- The capacity of a cut (A, B) is $c(A,B) = \sum_{e \text{ out of } A} c(e)$



capacity of cut = 30

Cuts

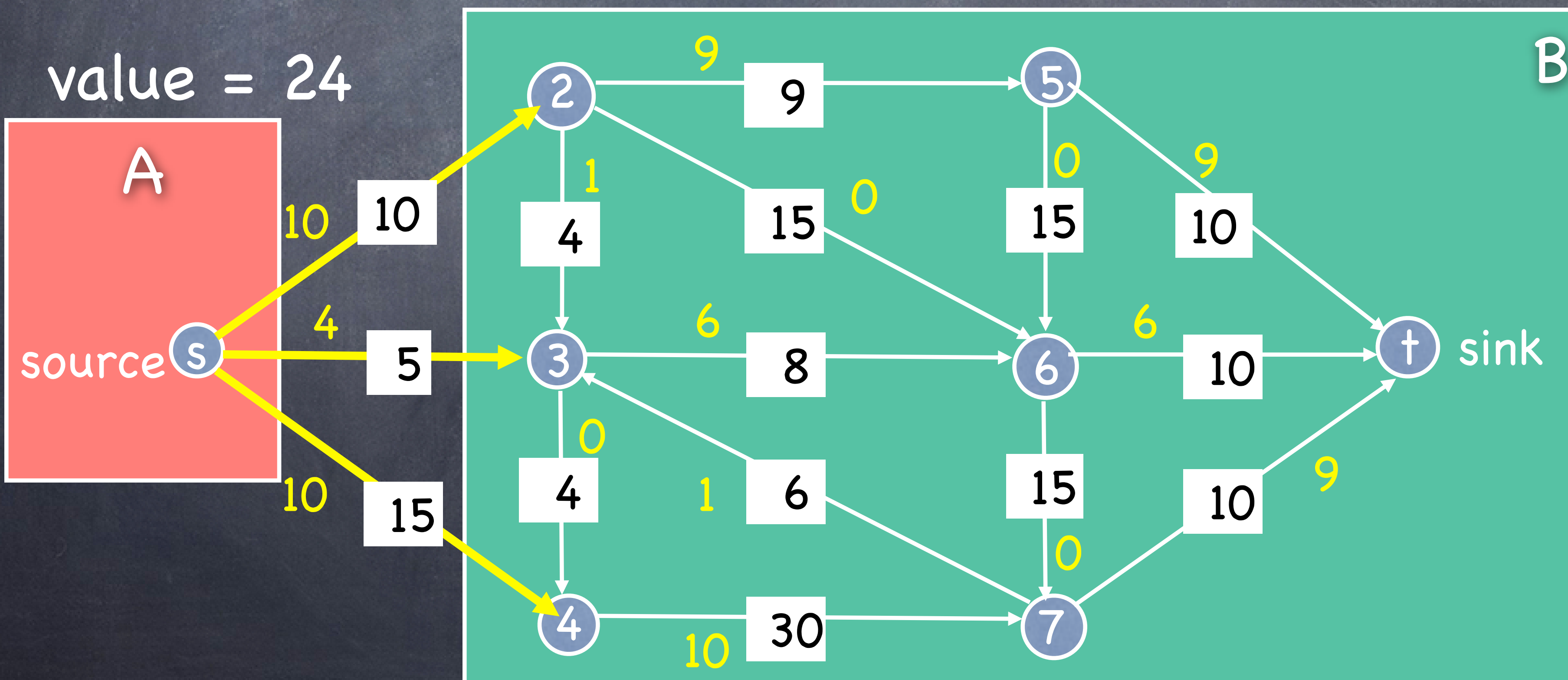
capacity of cut = 9 + 15 + 8 + 30 = 62
(Capacity is sum of weights on edges leaving A.)



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

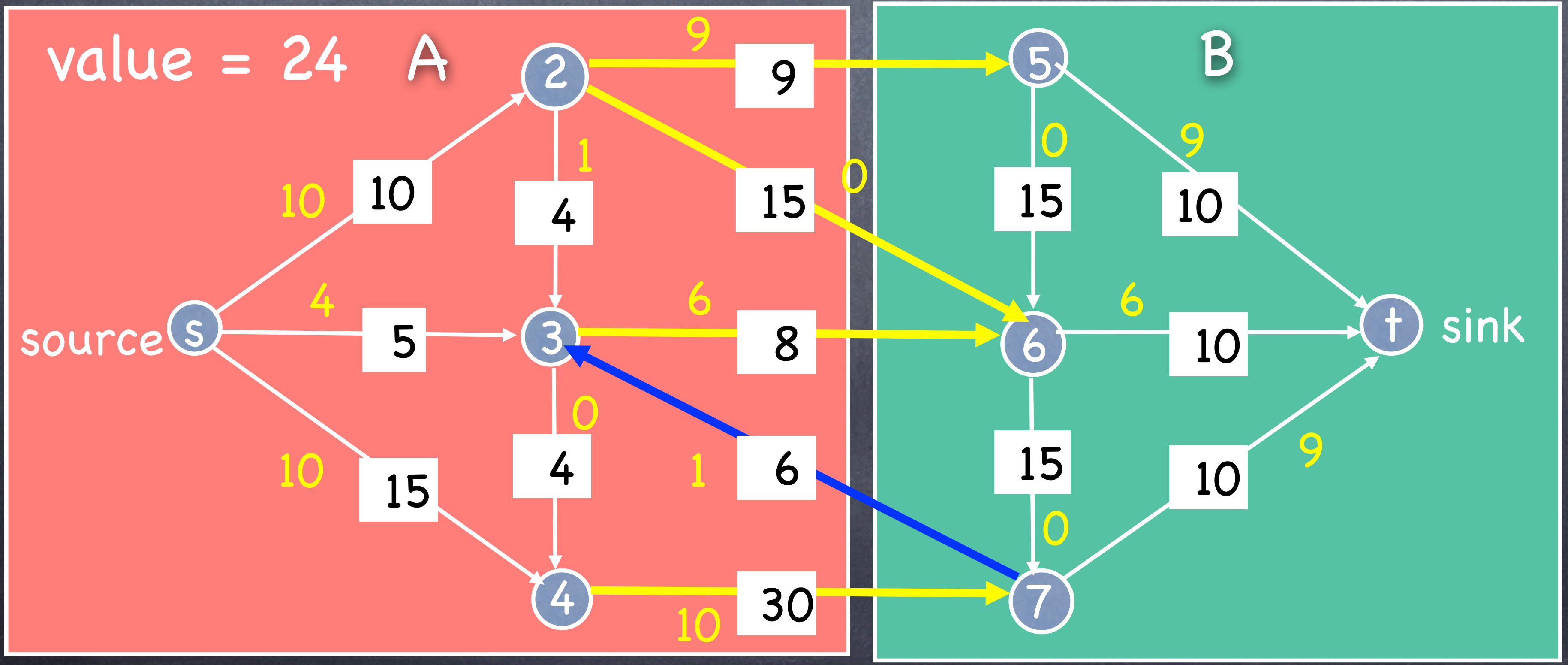
$$\sum_{e \text{ out of } a} f(e) - \sum_{e \text{ into } a} f(e) = v(f)$$



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = v(f)$$

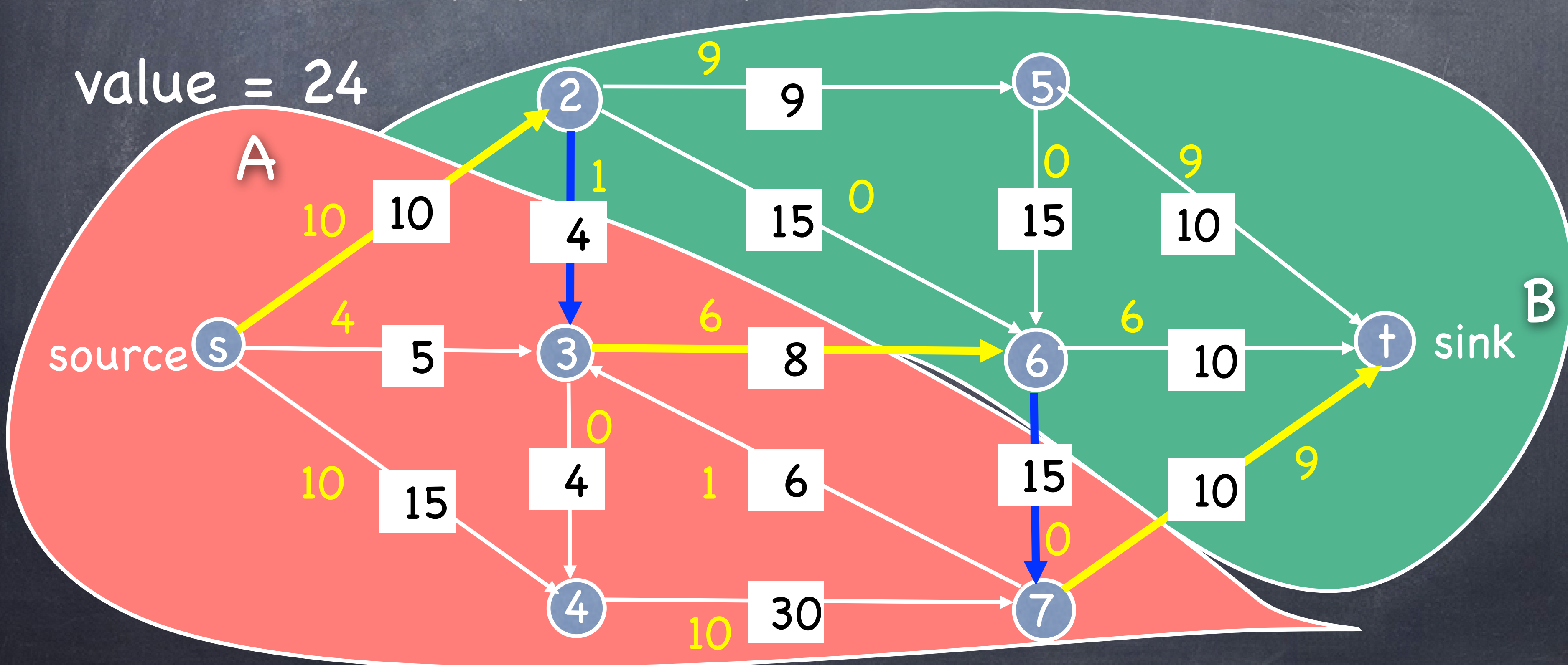


Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } a} f(e) - \sum_{e \text{ into } a} f(e) = v(f)$$

value = 24



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then $\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = v(f)$.

Proof:

$v(f) = \sum_{e \text{ out of } s} f(e)$ by definition

$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) \right)$ by flow conservation, all terms except v = s are 0

$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$ if both endpoints of e are in A, there will be canceling terms for that edge

Max-Flow Min-Cut

- There is a deep connection between flows and cuts in networks
- Next time, we will prove that Ford-Fulkerson is correct by proving the Max-Flow Min-Cut Theorem