



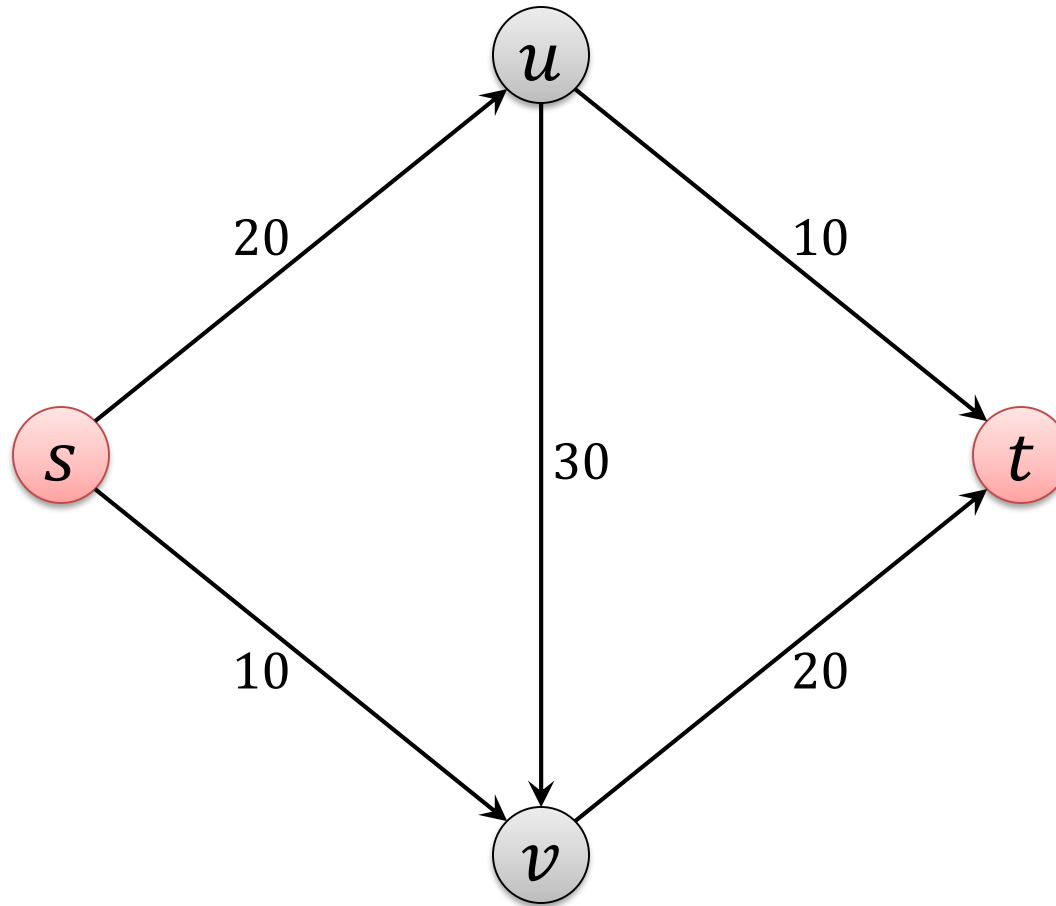
# Chapter 6

# Graph Algorithms

Algorithm Theory  
WS 2019/20

Fabian Kuhn

# Example: Flow Network

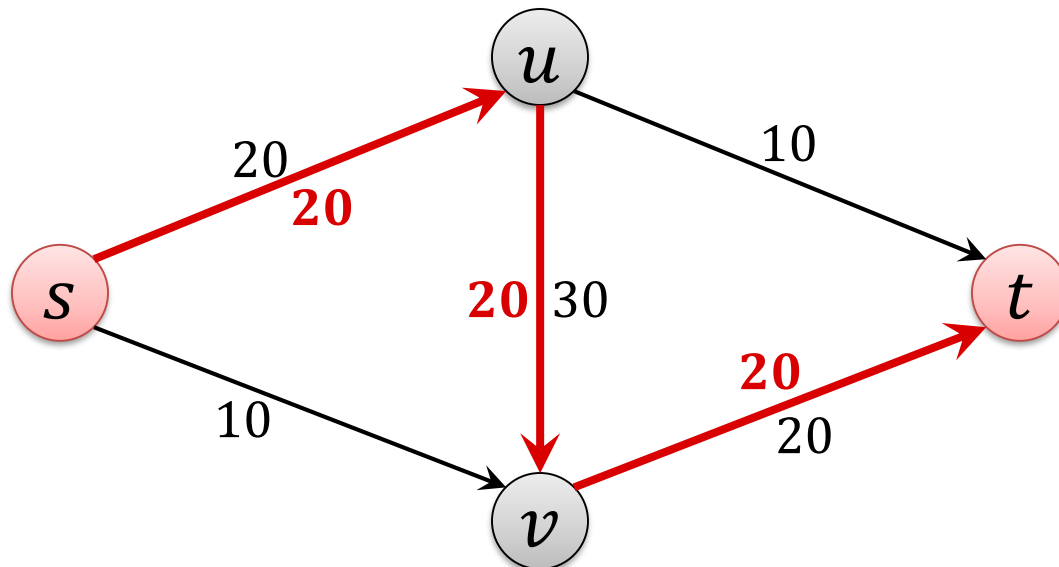


# Residual Graph

Given a flow network  $G = (V, E)$  with capacities  $c_e$  (for  $e \in E$ )

For a flow  $f$  on  $G$ , define **directed graph**  $G_f = (V_f, E_f)$  as follows:

- Node set  $V_f = V$
- For each edge  $e = (u, v)$  in  $E$ , there are two edges in  $E_f$ :
  - **forward edge**  $e = (u, v)$  with **residual capacity**  $c_e - f(e)$
  - **backward edge**  $e' = (v, u)$  with **residual capacity**  $f(e)$



# Augmenting Path

## Definition:

An **augmenting path**  $P$  is a (simple)  $s$ - $t$ -path on the **residual graph**  $G_f$  on which each edge has **residual capacity**  $> 0$ .

**bottleneck** $(P, f)$ : minimum residual capacity on any edge of the augmenting path  $P$

## Augment flow $f$ to get flow $f'$ :

- For every **forward edge**  $(u, v)$  on  $P$ :

$$f'((u, v)) := f((u, v)) + \mathbf{bottleneck}(P, f)$$

- For every **backward edge**  $(u, v)$  on  $P$ :

$$f'((v, u)) := f((v, u)) - \mathbf{bottleneck}(P, f)$$

# Ford-Fulkerson Algorithm

- Improve flow using an augmenting path as long as possible:
  1. Initially,  $f(e) = 0$  for all edges  $e \in E$ ,  $G_f = G$
  2. **while** there is an augmenting  $s$ - $t$ -path  $P$  in  $G_f$  **do**
  3.     Let  $P$  be an augmenting  $s$ - $t$ -path in  $G_f$ ;
  4.      $f' := \text{augment}(f, P)$ ;
  5.     update  $f$  to be  $f'$ ;
  6.     update the residual graph  $G_f$
  7. **end**;

# Ford-Fulkerson Running Time



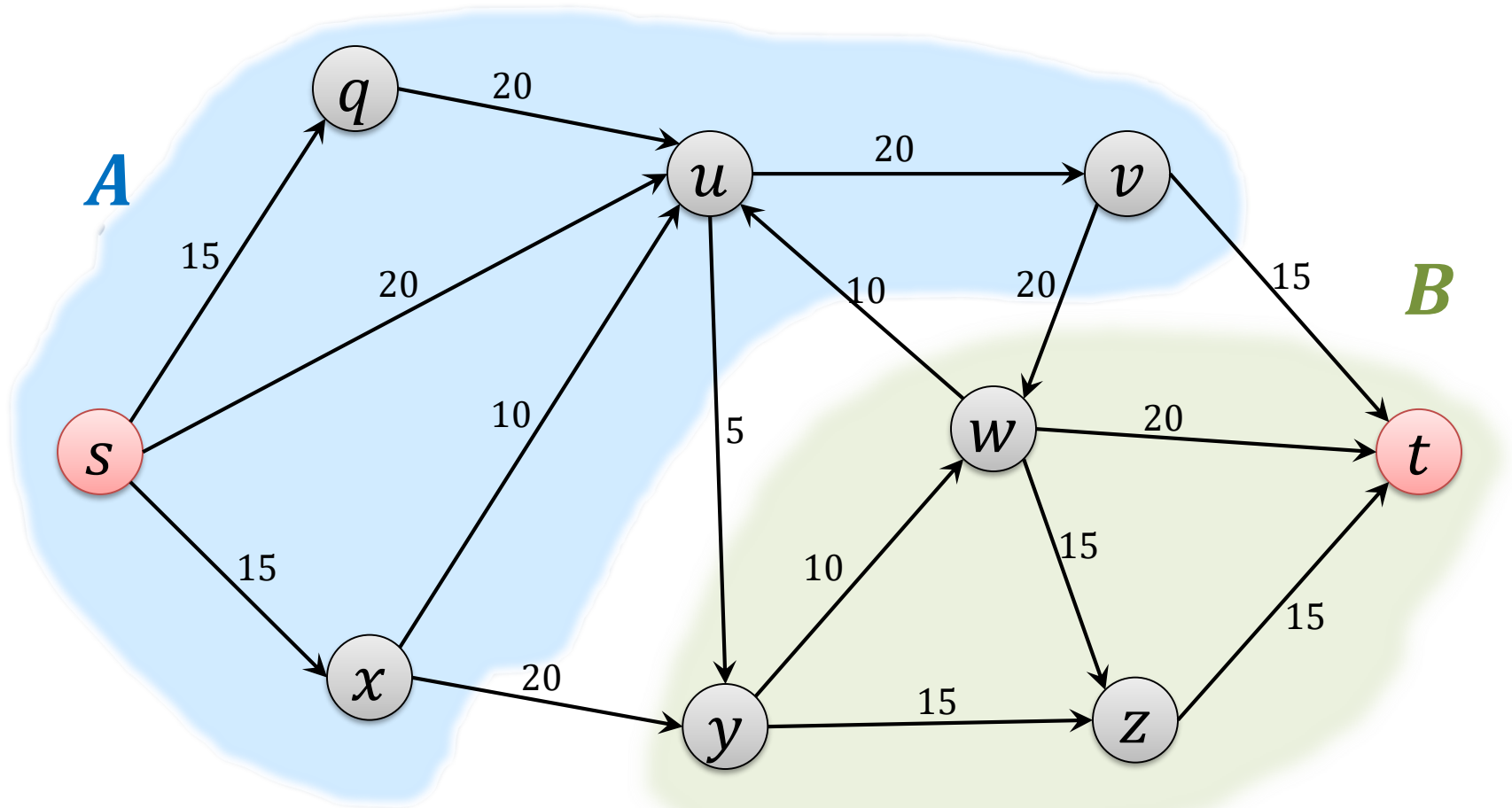
**Theorem:** If all edge capacities are integers, the Ford-Fulkerson algorithm can be implemented to run in  $O(mC)$  time.

**Proof:**

# $s$ - $t$ Cuts

## Definition:

An  $s$ - $t$  cut is a partition  $(A, B)$  of the vertex set such that  $s \in A$  and  $t \in B$

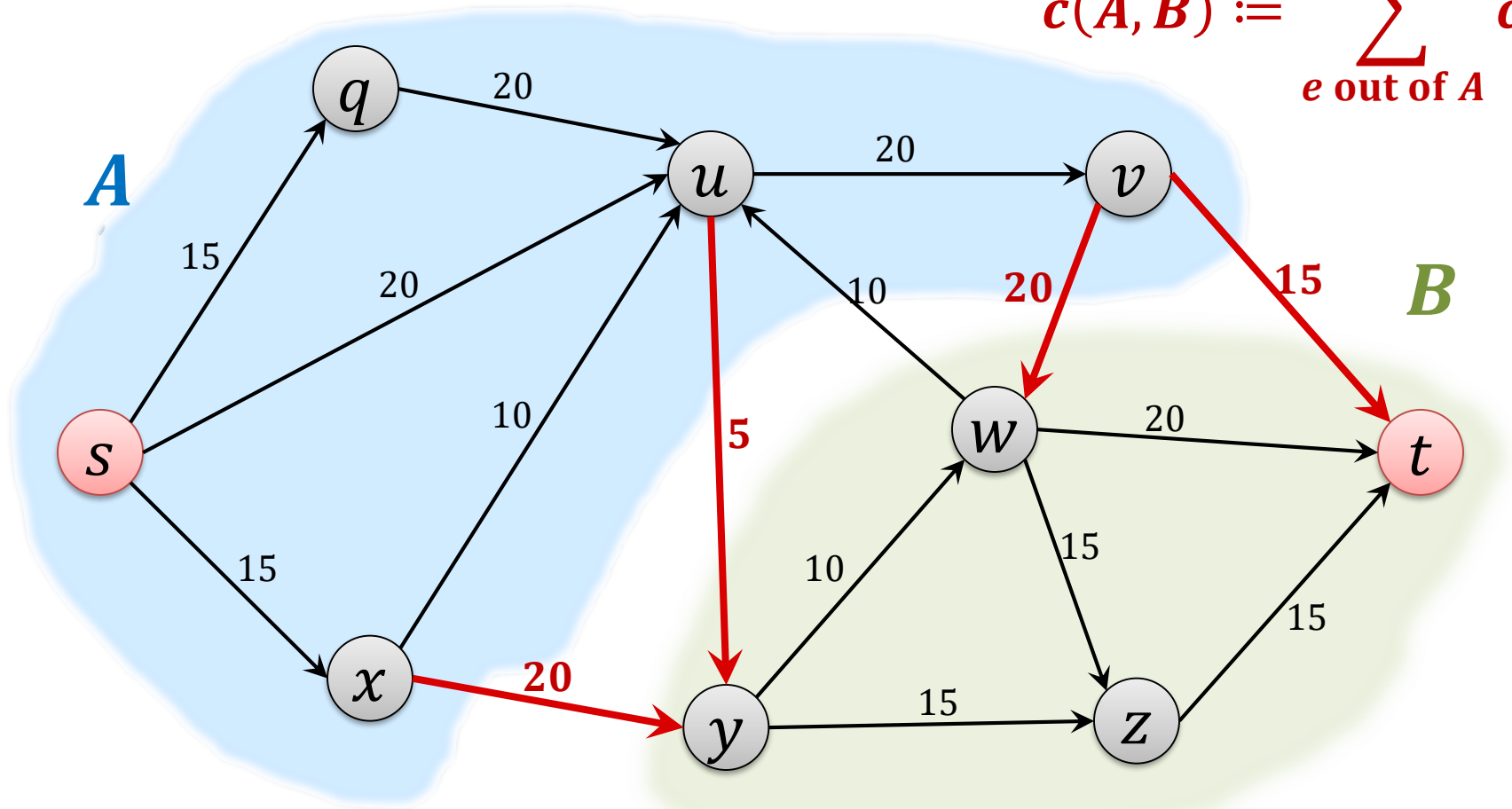


# Cut Capacity

## Definition:

The **capacity**  $c(A, B)$  of an  $s$ - $t$ -cut  $(A, B)$  is defined as

$$c(A, B) := \sum_{e \text{ out of } A} c_e.$$





# Max-Flow Min-Cut Theorem

## Theorem: (Max-Flow Min-Cut Theorem)

In every flow network, the maximum value of an  $s$ - $t$  flow is equal to the minimum capacity of an  $s$ - $t$  cut.

## Proof:

# Strongly Polynomial Algorithm

- Time of regular Ford-Fulkerson algorithm with integer capacities:

$$O(mC)$$

- Time of algorithm with scaling parameter:

$$O(m^2 \log C)$$

- $O(\log C)$  is polynomial in the size of the input, but not in  $n$
- Can we get an algorithm that runs in time polynomial in  $n$ ?
- Always picking a **shortest augmenting path** leads to running time

$$O(m^2 n)$$

- also works for arbitrary real-valued weights

# Other Algorithms

- There are many other algorithms to solve the maximum flow problem, for example:
- **Preflow-push algorithm:**
  - Maintains a preflow ( $\forall$  nodes: inflow  $\geq$  outflow)
  - Alg. guarantees: As soon as we have a flow, it is optimal
  - Detailed discussion in 2012/13 lecture
  - Running time of basic algorithm:  $O(m \cdot n^2)$
  - Doing steps in the “right” order:  $O(n^3)$
- **Current best known complexity:  $O(m \cdot n)$** 
  - For graphs with  $m \geq n^{1+\epsilon}$  [King,Rao,Tarjan 1992/1994]  
(for every constant  $\epsilon > 0$ )
  - For sparse graphs with  $m \leq n^{16/15-\delta}$  [Orlin, 2013]

# Maximum Flow Applications

- Maximum flow has many applications
- Reducing a problem to a max flow problem can even be seen as an important algorithmic technique
- Examples:
  - related network flow problems
  - computation of small cuts
  - computation of matchings
  - computing disjoint paths
  - scheduling problems
  - assignment problems with some side constraints
  - ...

# Undirected Edges and Vertex Capacities

## Undirected Edges:

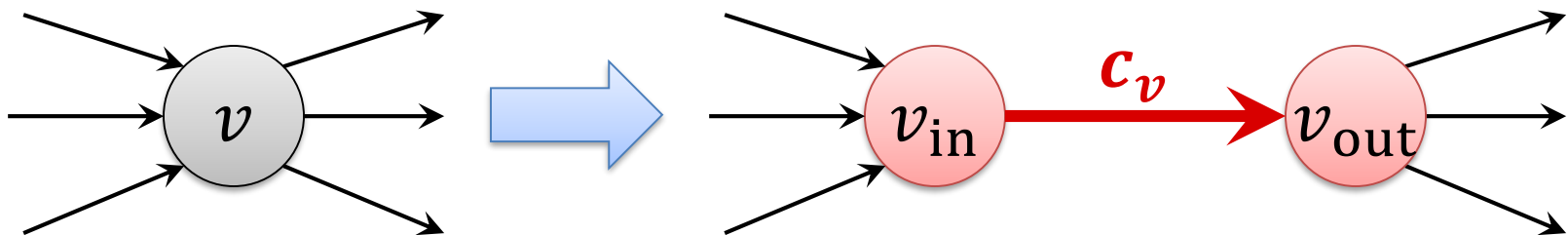
- Undirected edge  $\{u, v\}$ : add edges  $(u, v)$  and  $(v, u)$  to network

## Vertex Capacities:

- Not only edges, but also (or only) nodes have capacities
- Capacity  $c_v$  of node  $v \notin \{s, t\}$ :

$$f^{\text{in}}(v) = f^{\text{out}}(v) \leq c_v$$

- Replace node  $v$  by edge  $e_v = \{v_{\text{in}}, v_{\text{out}}\}$ :



# Minimum $s$ - $t$ Cut

**Given:** undirected graph  $G = (V, E)$ , nodes  $s, t \in V$

**$s$ - $t$  cut:** Partition  $(A, B)$  of  $V$  such that  $s \in A, t \in B$

**Size of cut  $(A, B)$ :** number of edges crossing the cut

**Objective:** find  $s$ - $t$  cut of minimum size

# Edge Connectivity

**Definition:** A graph  $G = (V, E)$  is  **$k$ -edge connected** for an integer  $k \geq 1$  if the graph  **$G_X = (V, E \setminus X)$  is connected** for every edge set

$$X \subseteq E, |X| \leq k - 1.$$

**Goal:** Compute **edge connectivity  $\lambda(G)$**  of  $G$   
(and edge set  $X$  of size  $\lambda(G)$  that divides  $G$  into  $\geq 2$  parts)

- minimum set  $X$  is a minimum  $s$ - $t$  cut for some  $s, t \in V$ 
  - Actually for all  $s, t$  in different components of  $G_X = (V, E \setminus X)$
- Possible algorithm: fix  $s$  and find min  $s$ - $t$  cut for all  $t \neq s$

# Minimum $s$ - $t$ Vertex-Cut

**Given:** undirected graph  $G = (V, E)$ , nodes  $s, t \in V$

**$s$ - $t$  vertex cut:** Set  $X \subset V$  such that  $s, t \notin X$  and  $s$  and  $t$  are in different components of the sub-graph  $G[V \setminus X]$  induced by  $V \setminus X$

**Size of vertex cut:**  $|X|$

**Objective:** find  $s$ - $t$  vertex-cut of minimum size

- Replace undirected edge  $\{u, v\}$  by  $(u, v)$  and  $(v, u)$
- Compute max  $s$ - $t$  flow for edge capacities  $\infty$  and node capacities

$$c_v = 1 \text{ for } v \neq s, t$$

- Replace each node  $v$  by  $v_{\text{in}}$  and  $v_{\text{out}}$ :
- Min edge cut corresponds to min vertex cut in  $G$



# Vertex Connectivity

**Definition:** A graph  $G = (V, E)$  is  **$k$ -vertex connected** for an integer  $k \geq 1$  if the sub-graph  $G[V \setminus X]$  **induced by  $V \setminus X$  is connected** for every edge set

$$X \subseteq V, |X| \leq k - 1.$$

**Goal:** Compute **vertex connectivity  $\kappa(G)$**  of  $G$   
(and node set  $X$  of size  $\kappa(G)$  that divides  $G$  into  $\geq 2$  parts)

- Compute minimum  $s$ - $t$  vertex cut for all  $s$  and all  $t \neq s$

# Edge-Disjoint Paths

**Given:** Graph  $G = (V, E)$  with nodes  $s, t \in V$

**Goal:** Find as many edge-disjoint  $s$ - $t$  paths as possible

**Solution:**

- Find max  $s$ - $t$  flow in  $G$  with **edge capacities**  $c_e = 1$  for all  $e \in E$

Flow  $f$  induces  **$|f|$  edge-disjoint paths:**

- Integral capacities  $\rightarrow$  can compute integral max flow  $f$
- Get  $|f|$  edge-disjoint paths by greedily picking them
- Correctness follows from flow conservation  $f^{\text{in}}(v) = f^{\text{out}}(v)$

# Vertex-Disjoint Paths

**Given:** Graph  $G = (V, E)$  with nodes  $s, t \in V$

**Goal:** Find as many internally vertex-disjoint  $s$ - $t$  paths as possible

**Solution:**

- Find max  $s$ - $t$  flow in  $G$  with **node capacities**  $c_v = 1$  for all  $v \in V$

Flow  $f$  induces  **$|f|$  vertex-disjoint paths:**

- Integral capacities  $\rightarrow$  can compute integral max flow  $f$
- Get  $|f|$  vertex-disjoint paths by greedily picking them
- Correctness follows from flow conservation  $f^{\text{in}}(v) = f^{\text{out}}(v)$

# Menger's Theorem

## **Theorem: (edge version)**

For every graph  $G = (V, E)$  with nodes  $s, t \in V$ , the size of the minimum  $s$ - $t$  (edge) cut equals the maximum number of pairwise edge-disjoint paths from  $s$  to  $t$ .

## **Theorem: (node version)**

For every graph  $G = (V, E)$  with nodes  $s, t \in V$ , the size of the minimum  $s$ - $t$  vertex cut equals the maximum number of pairwise internally vertex-disjoint paths from  $s$  to  $t$ .

- Both versions can be seen as a special case of the max flow min cut theorem

# Baseball Elimination

Team $i$	Wins $w_i$	Losses $\ell_i$	To Play $r_i$	Against = $r_{ij}$				
				NY	Balt.	T. Bay	Tor.	Bost.
New York	81	69	12	-	2	5	2	3
Baltimore	79	77	6	2	-	2	1	1
Tampa Bay	79	74	9	5	2	-	1	1
Toronto	76	80	6	2	1	1	-	2
Boston	71	84	7	3	1	1	2	-

- Only wins/losses possible (no ties), winner: team with most wins
- Which teams can still win (as least as many wins as top team)?
- Boston is eliminated (cannot win):
  - Boston can get at most 78 wins, New York already has 81 wins
- If for some  $i, j$ :  $w_i + r_i < w_j \rightarrow$  team  $i$  is eliminated
- **Sufficient** condition, **but not** a **necessary** one!

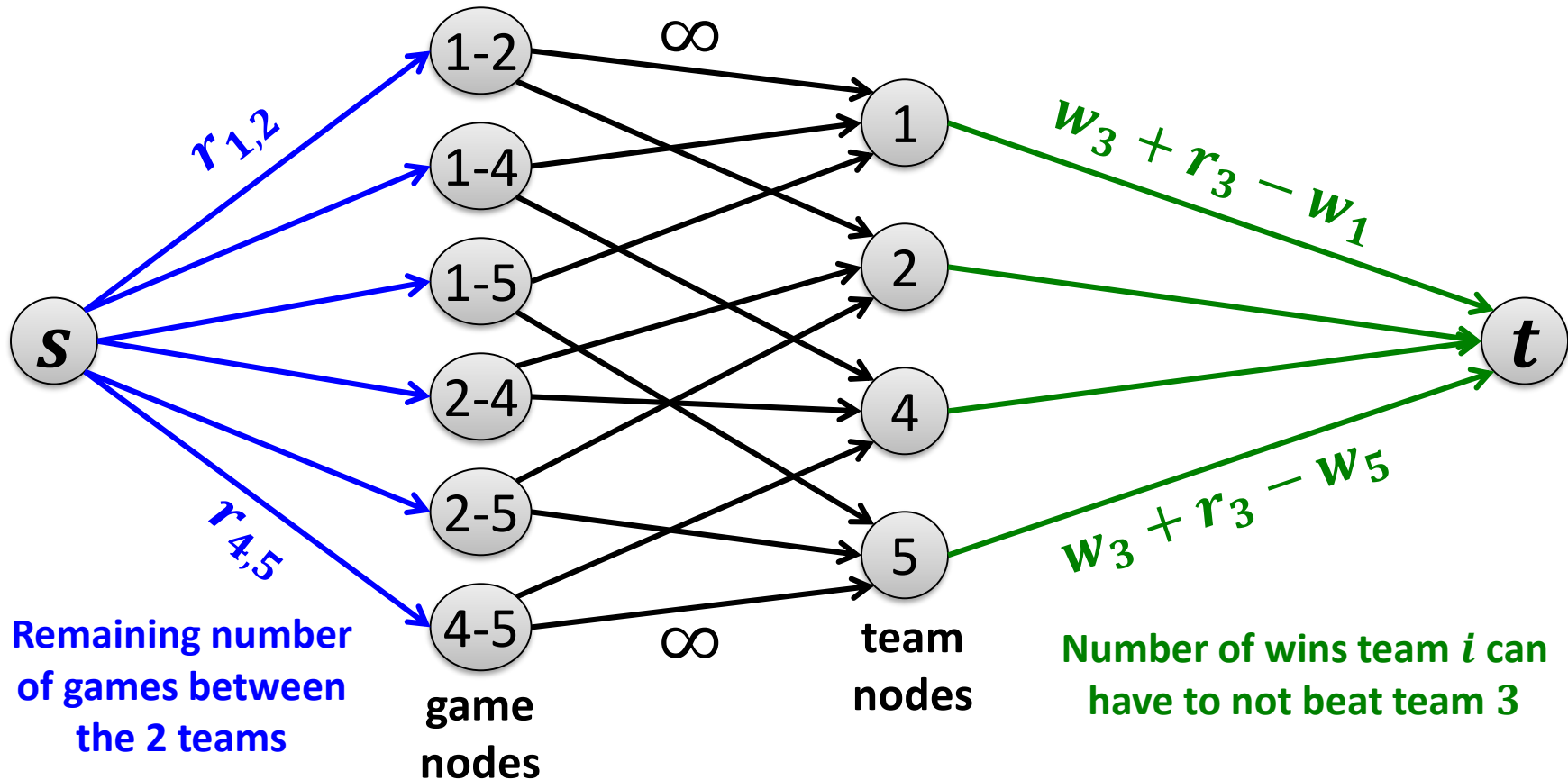
# Baseball Elimination

Team $i$	Wins $w_i$	Losses $\ell_i$	To Play $r_i$	Against = $r_{ij}$				
				NY	Balt.	T. Bay	Tor.	Bost.
New York	81	69	12	-	2	5	2	3
Baltimore	79	77	6	2	-	2	1	1
Tampa Bay	79	74	9	5	2	-	1	1
Toronto	76	80	6	2	1	1	-	2
Boston	71	84	7	3	1	1	2	-

- Can Toronto still finish first?
- Toronto can get  $82 > 81$  wins, but:  
 NY and Tampa have to play 5 more times against each other  
 → if NY wins two, it gets 83 wins, otherwise, Tampa has 83 wins
- Hence: Toronto cannot finish first
- How about the others? How can we solve this in general?

# Max Flow Formulation

- Can team 3 finish with most wins?



- Team 3 can finish first iff all source-game edges are saturated

# Reason for Elimination

AL East: Aug 30, 1996

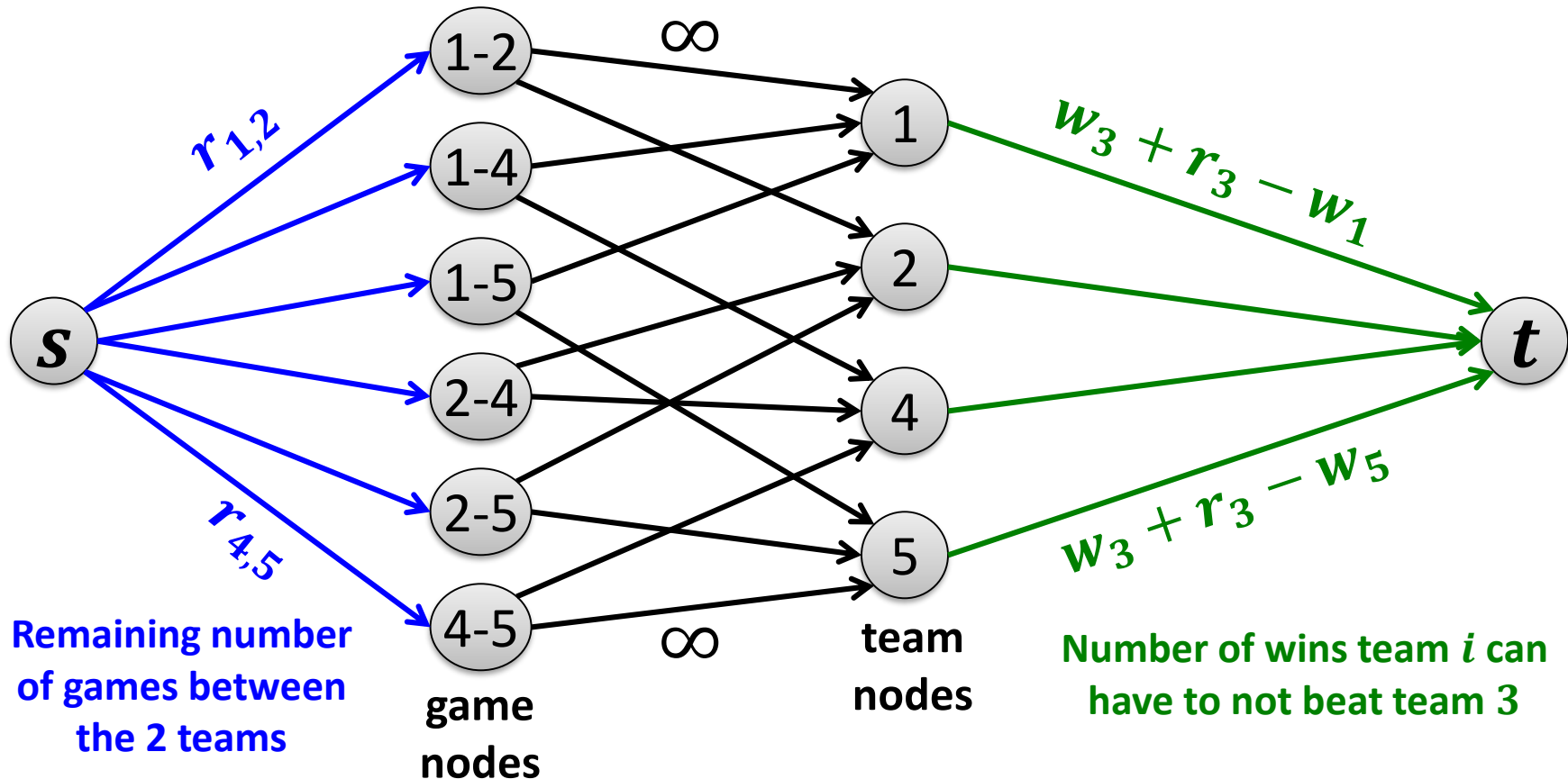
Team $i$	Wins $w_i$	Losses $\ell_i$	To Play $r_i$	Against = $r_{ij}$				
				NY	Balt.	Bost.	Tor.	Detr.
New York	75	59	28	-	3	8	7	3
Baltimore	71	63	28	3	-	2	7	4
Boston	69	66	27	8	2	-	0	0
Toronto	63	72	27	7	7	0	-	0
<b>Detroit</b>	49	86	27	3	4	0	0	-

- Detroit could finish with  $49 + 27 = 76$  wins
- Consider  $R = \{\text{NY, Bal, Bos, Tor}\}$ 
  - Have together already won  $w(R) = 278$  games
  - Must together win at least  $r(R) = 27$  more games
- On average, teams in  $R$  win  $\frac{278+27}{4} = 76.25$  games



# Reason for Elimination

- Can team 3 finish with most wins?



- Team 3 cannot finish first  $\Leftrightarrow$  min cut of size  $<$  “all blue edges”

# Reason for Elimination

**Certificate of elimination:**

$$R \subseteq X, \quad w(R) := \underbrace{\sum_{i \in R} w_i}_{\text{\#wins of nodes in } R}, \quad r(R) := \underbrace{\sum_{i, j \in R} r_{i, j}}_{\text{\#remaining games among nodes in } R}$$

Team  $x \in X$  is eliminated by  $R$  if

$$\frac{w(R) + r(R)}{|R|} > w_x + r_x.$$

# Reason for Elimination

**Theorem:** Team  $x$  is eliminated if and only if there exists a subset  $R \subseteq X$  of the teams  $X$  such that  $x$  is eliminated by  $R$ .

## Proof Idea:

- Minimum cut gives a certificate...
- If  $x$  is eliminated, max flow solution does not saturate all outgoing edges of the source.
- Team nodes of unsaturated source-game edges are saturated
- Source side of min cut contains all teams of saturated team-dest. edges of unsaturated source-game edges
- Set of team nodes in source-side of min cut give a certificate  $R$

# Circulations with Demands

**Given:** Directed network with positive edge capacities

**Sources & Sinks:** Instead of one source and one destination, several sources that generate flow and several sinks that absorb flow.

**Supply & Demand:** sources have supply values, sinks demand values

**Goal:** Compute a flow such that source supplies and sink demands are exactly satisfied

- The circulation problem is a feasibility rather than a maximization problem

# Circulations with Demands: Formally

**Given:** Directed network  $G = (V, E)$  with

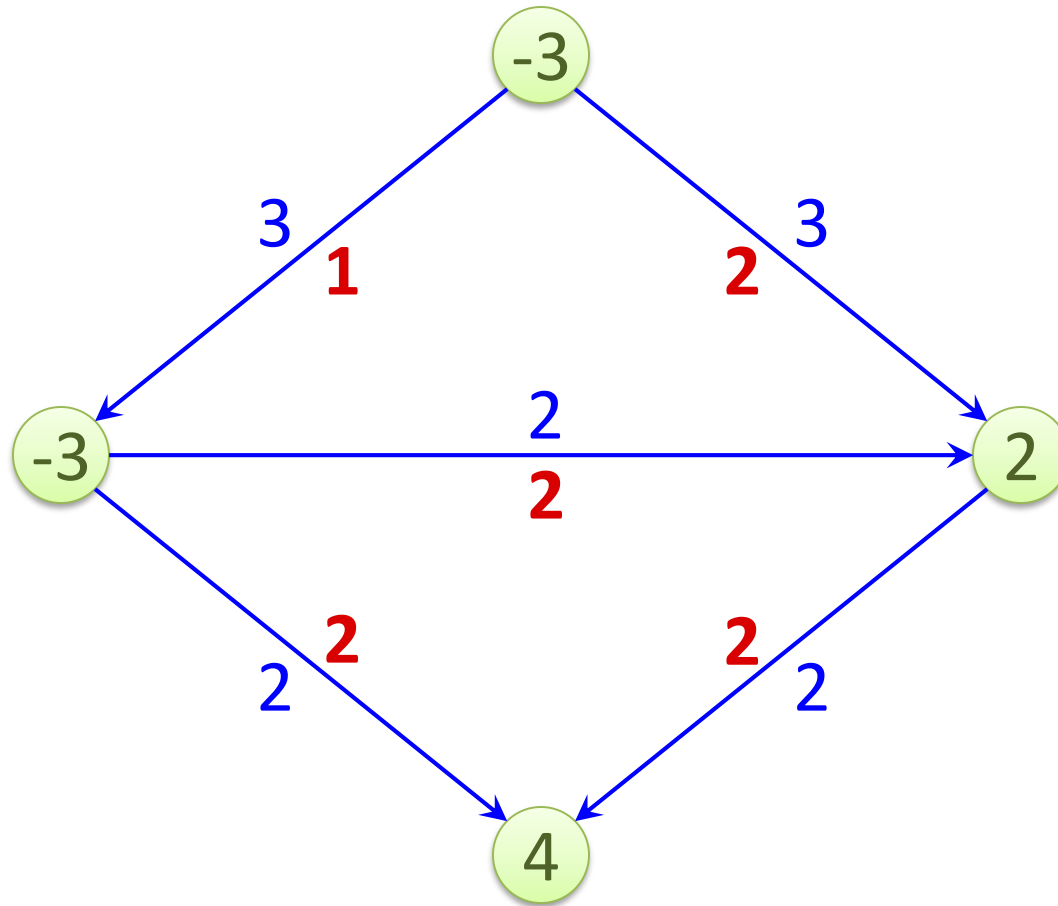
- Edge capacities  $c_e > 0$  for all  $e \in E$
- Node demands  $d_v \in \mathbb{R}$  for all  $v \in V$ 
  - $d_v > 0$ : node needs flow and therefore is a sink
  - $d_v < 0$ : node has a supply of  $-d_v$  and is therefore a source
  - $d_v = 0$ : node is neither a source nor a sink

**Flow:** Function  $f: E \rightarrow \mathbb{R}_{\geq 0}$  satisfying

- *Capacity Conditions:*  $\forall e \in E: 0 \leq f(e) \leq c_e$
- *Demand Conditions:*  $\forall v \in V: f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$

**Objective:** Does a flow  $f$  satisfying all conditions exist?  
If yes, find such a flow  $f$ .

# Example



# Condition on Demands

**Claim:** If there exists a feasible circulation with demands  $d_v$  for  $v \in V$ , then

$$\sum_{v \in V} d_v = 0.$$

**Proof:**

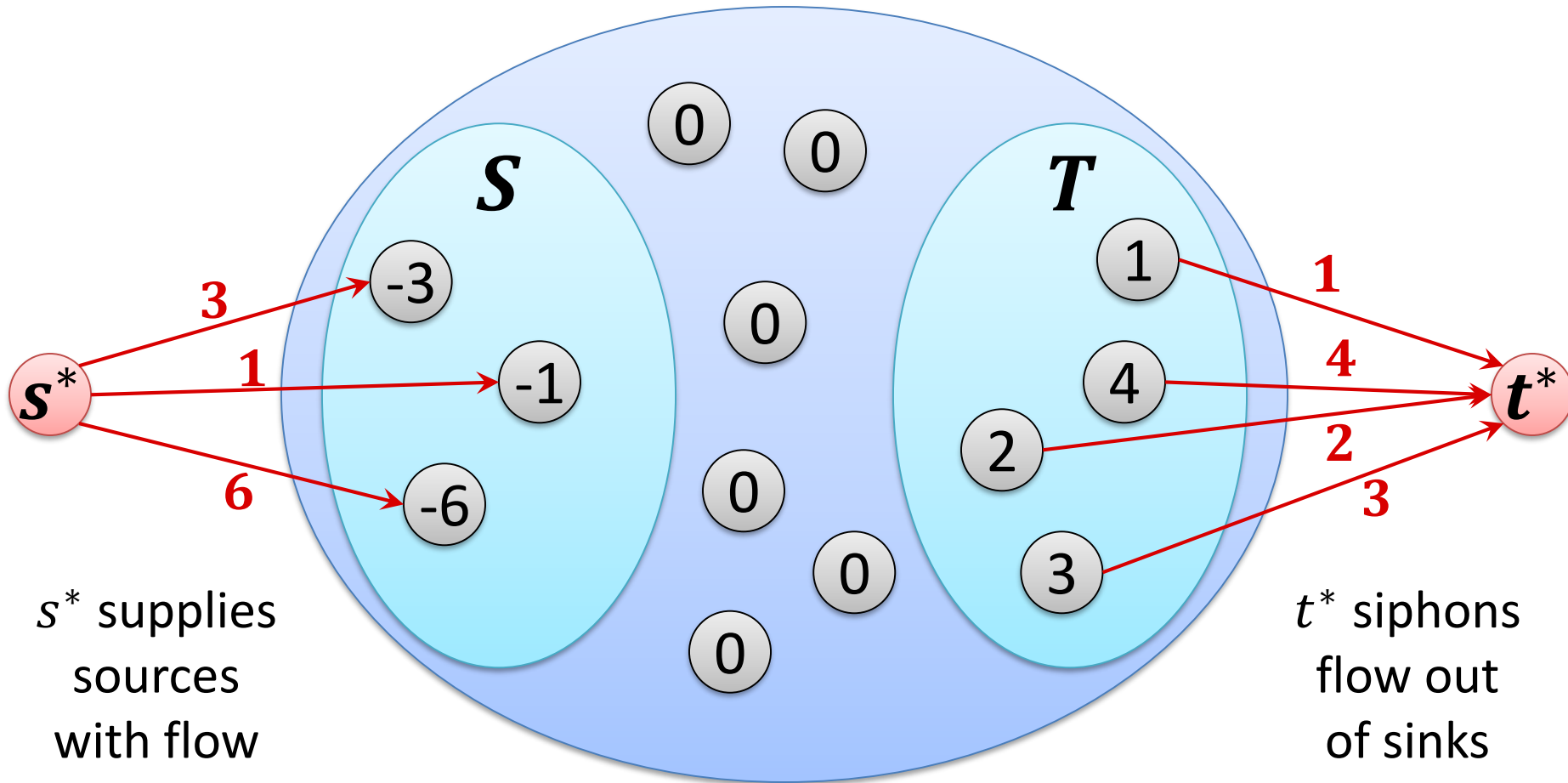
- $\sum_v d_v = \sum_v (f^{\text{in}}(v) - f^{\text{out}}(v))$
- $f(e)$  of each edge  $e$  appears twice in the above sum with different signs  $\rightarrow$  overall sum is 0

**Total supply = total demand:**

$$\text{Define } D := \sum_{v: d_v > 0} d_v = \sum_{v: d_v < 0} -d_v$$

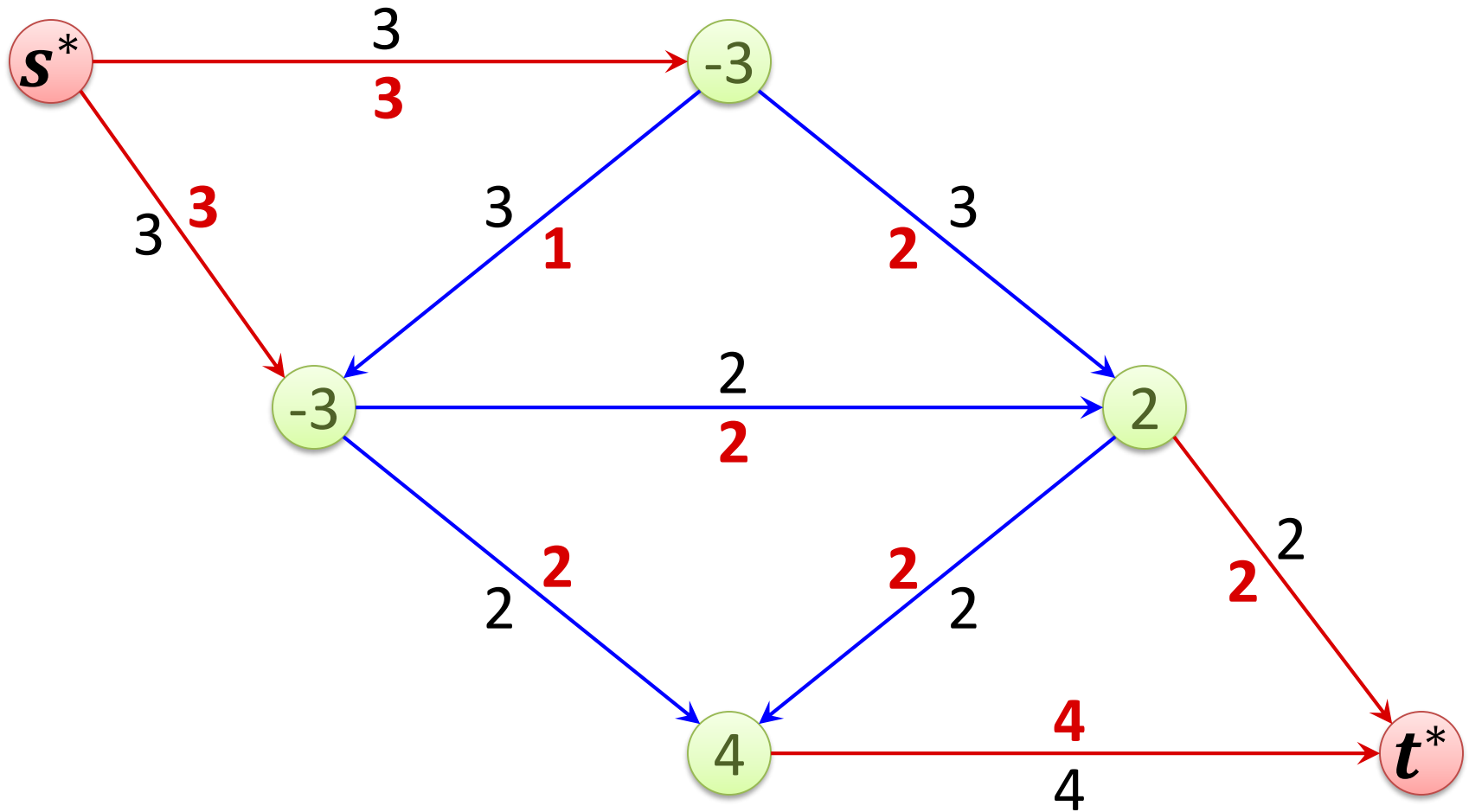
# Reduction to Maximum Flow

- Add “super-source”  $s^*$  and “super-sink”  $t^*$  to network





# Example



# Formally...

**Reduction:** Get graph  $G'$  from graph as follows

- Node set of  $G'$  is  $V \cup \{s^*, t^*\}$
- Edge set is  $E$  and edges
  - $(s^*, v)$  for all  $v$  with  $d_v < 0$ , capacity of edge is  $-d_v$
  - $(v, t^*)$  for all  $v$  with  $d_v > 0$ , capacity of edge is  $d_v$

**Observations:**

- Capacity of min  $s^*$ - $t^*$  cut is at most  $D$  (e.g., the cut  $(s^*, V \cup \{t^*\})$ )
- A feasible circulation on  $G$  can be turned into a feasible flow of value  $D$  of  $G'$  by saturating all  $(s^*, v)$  and  $(v, t^*)$  edges.
- Any flow of  $G'$  of value  $D$  induces a feasible circulation on  $G$ 
  - $(s^*, v)$  and  $(v, t^*)$  edges are saturated
  - By removing these edges, we get exactly the demand constraints

# Circulation with Demands

**Theorem:** There is a feasible circulation with demands  $d_v, v \in V$  on graph  $G$  if and only if there is a flow of value  $D$  on  $G'$ .

- If all capacities and demands are integers, there is an integer circulation

The **max flow min cut theorem** also implies the following:

**Theorem:** The graph  $G$  has a feasible circulation with demands  $d_v, v \in V$  if and only if for all cuts  $(A, B)$ ,

$$\sum_{v \in B} d_v \leq c(A, B).$$