# Applications of Max Flow

# Maximum Flow Overview

Given a directed graph with special vertices $s$ and $t$, find a maximum flow from $s$ to $t$.

- The flow through each edge $(u, v)$ cannot exceed the capacity of $(u, v)$.
- For each node $v \in V \setminus \{s, t\}$, the total amount of flow entering $v$ must be equal to the total amount of flow leaving $v$.

Finding the maximum flow is equivalent to finding the min-cut: i.e., the partition of nodes in two disjoint sets $S$ and $T$ such that $s \in S, t \in T$ and the sum of capacities of edges connecting a node in $S$ to a node in $T$ are minimized.

Residual graph.

- For every edge $(u, v)$ in the original graph, create an opposite edge $(v, u)$. Initially, the *residual capacity* $c_f(u, v)$ equals the capacity of $(u, v)$, and $c_f(v, u) = 0$.
- Let $f(u, v)$ be the flow passing through edge $(u, v)$: $c_f(u, v)$ decreases by $f(u, v)$ and $c_f(v, u)$ increases by $f(u, v)$.

The following Ford Fulkerson Algorithm operates on the residual graph.

# Ford Fulkerson Algorithm

Start with $f(e) = 0$ for each edge $e$.
Construct Residual Graph $G_f$ for current flow $f(e) = 0$
While there exists some $s-t$ path $P$ in $G_f$

  Let $c_f(P)$ be the maximum amount of flow that can be pushed through $P$
  Push $c_f(P)$ units of flow along the edges $e \in P$
  Construct $G_f$ for new flow $f(e)$

Integrality property: if all edge capacities are integers, then there exists a max flow for which every flow value is an integer and the F-F algorithm constructs such a flow.

Complexity: Assuming that all capacities are integer, and we choose paths at random, the worst case cost is $O(Ef^*)$, where $f^*$ is the maximum flow (pseudo-polynomial complexity).

If we choose, shortest paths (in terms of number of edges), the cost is $O(VE^2)$.

# Max Flow Applications

The Max Flow setup can model (surprisingly) many (seemingly) unrelated problems.

The idea is to express the problem as a max flow and then feed individual instances into our max flow solver.

The examples seen below all share the property that they are integer flow problems, e.g., all capacities are integral, so running-time analyses can use Ford-Fulkerson for integral flows.

1. Maximum Bipartite Matching
2. Edge-Disjoint Paths
3. Circulations with Demands
4. Baseball Elimination

# Maximum Bipartite Matching

A graph $G = (V, E)$ is Bipartite if there exists partition
$V = X \cup Y$ with $X \cap Y = \emptyset$ and $E \subseteq X \times Y$.

A Matching is a subset $M \subseteq E$
such that $\forall v \in V$ at most one edge in $M$ is incident upon $v$.

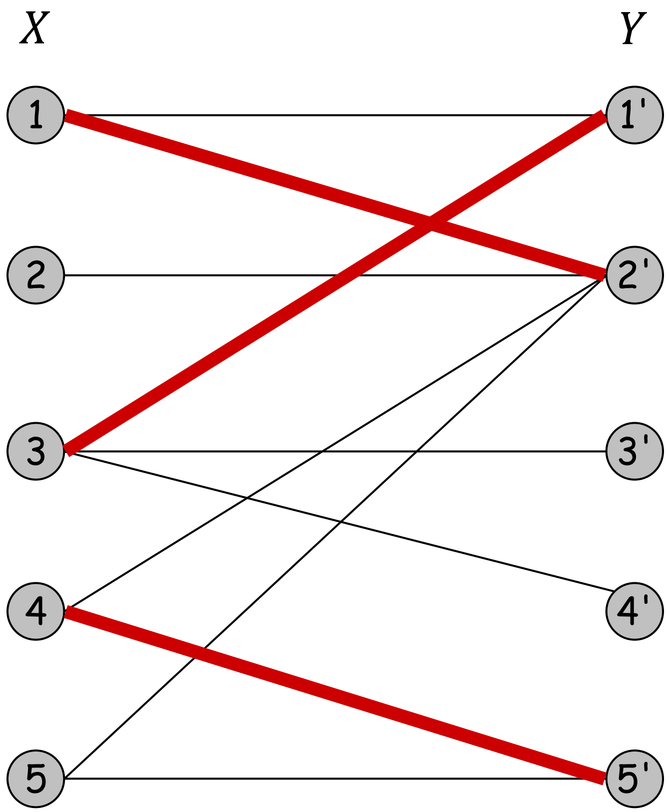
The Size $|M|$ is the number of edges in $M$.

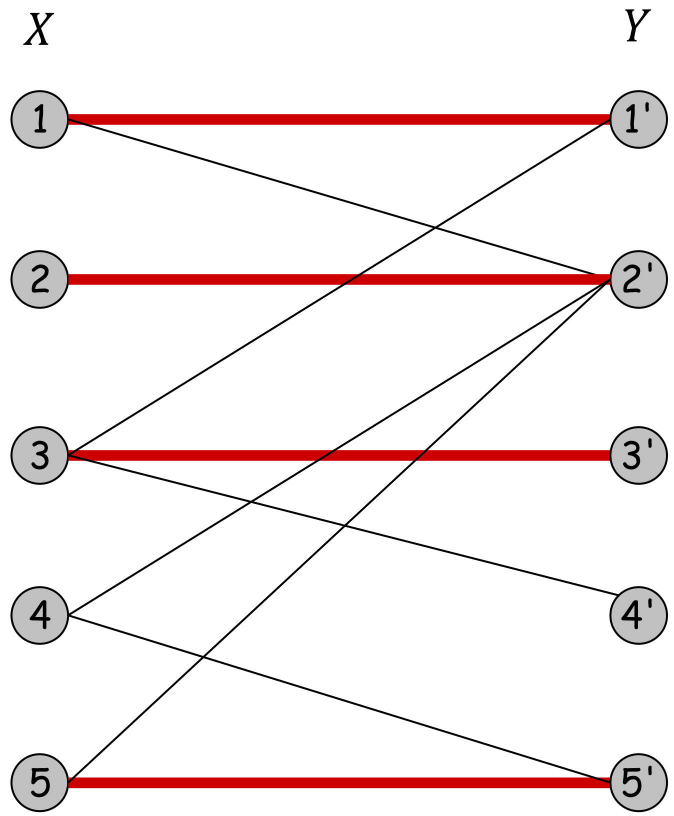
A Maximum Matching is matching $M$ such that
every other matching $M'$ satisfies $|M'| \leq |M|$.

Problem: Given bipartite graph $G$, find a Maximum Matching.

Applications: Assign jobs to people, tasks to machines, etc.

# Bipartite Matching Example




<u>Matching</u>

1-2', 3-1', 4-5'

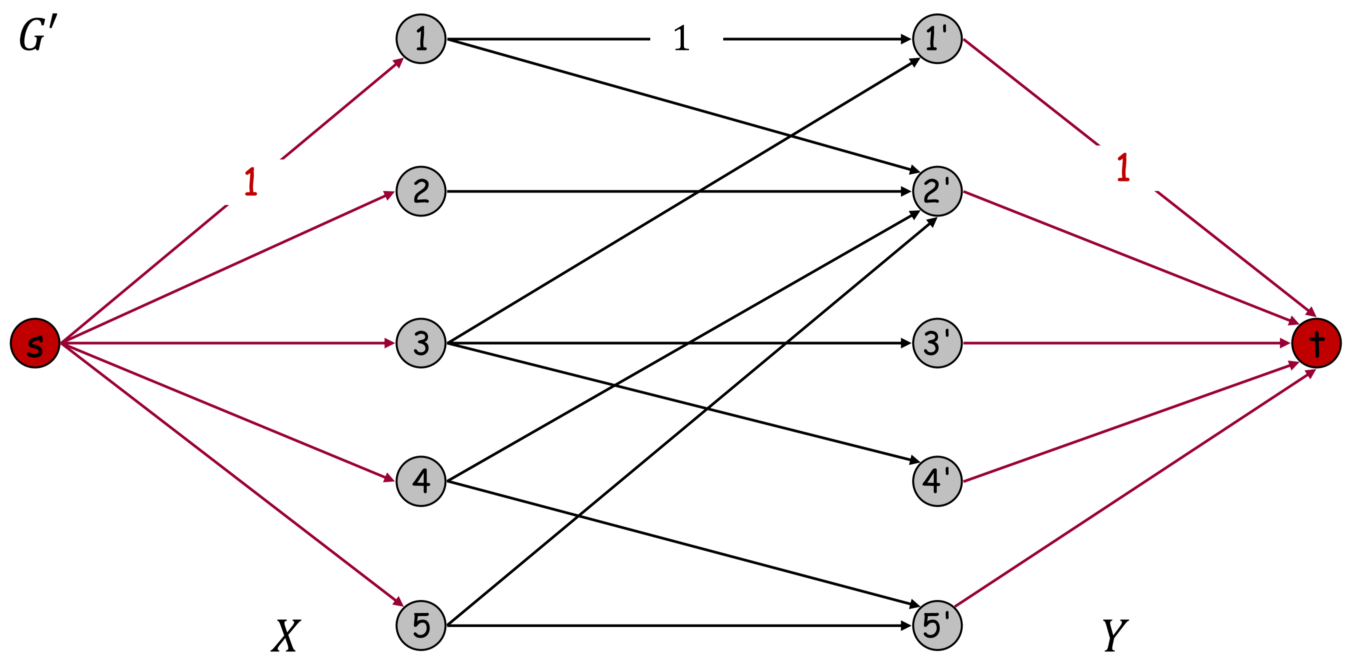
<u>Max Matching</u>

1-1', 2-2', 3-3' 4-4'

# From Bipartite Matching to Flow

## Max flow formulation.

- Create directed graph $G' = (X \cup Y \cup \{s, t\}, E')$.
- Direct all edges from $X$ to $Y$, and assign them capacity $1$.
- Add source $s$, and unit capacity edges from $s$ to each node in $X$.
- Add target $t$, and unit capacity edges from each node in $Y$ to $t$.

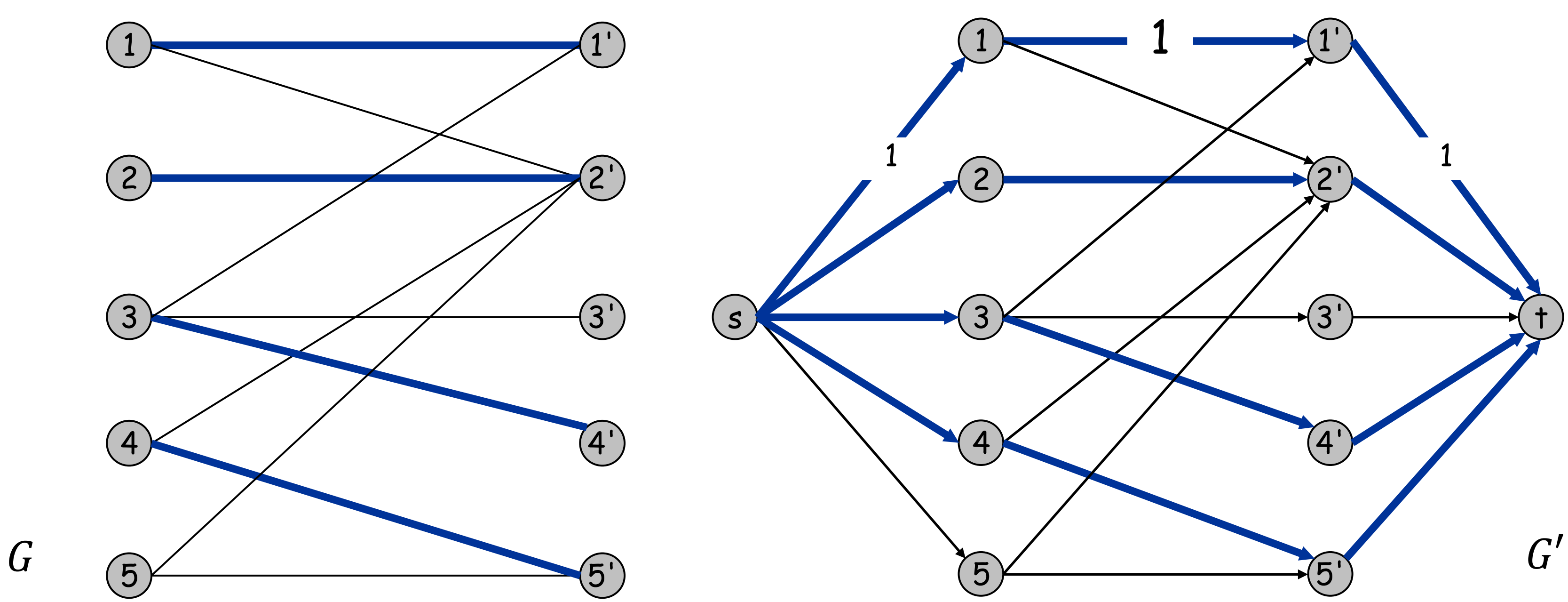


# Maximum Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in $G$ = value of max flow in $G'$.

Pf. A max matching with cardinality $k$ in G ⇒ a flow of value $k$ in $G'$

- Given a matching $M$ of cardinality $k$.
- Consider flow $f$ that sends 1 unit along each of the $3k$ edges.
  - $(s, x), (x, y), (y, t)$, where $(x, y)$ is an edge in the matching
- $f$ is a flow, and has value $k$.

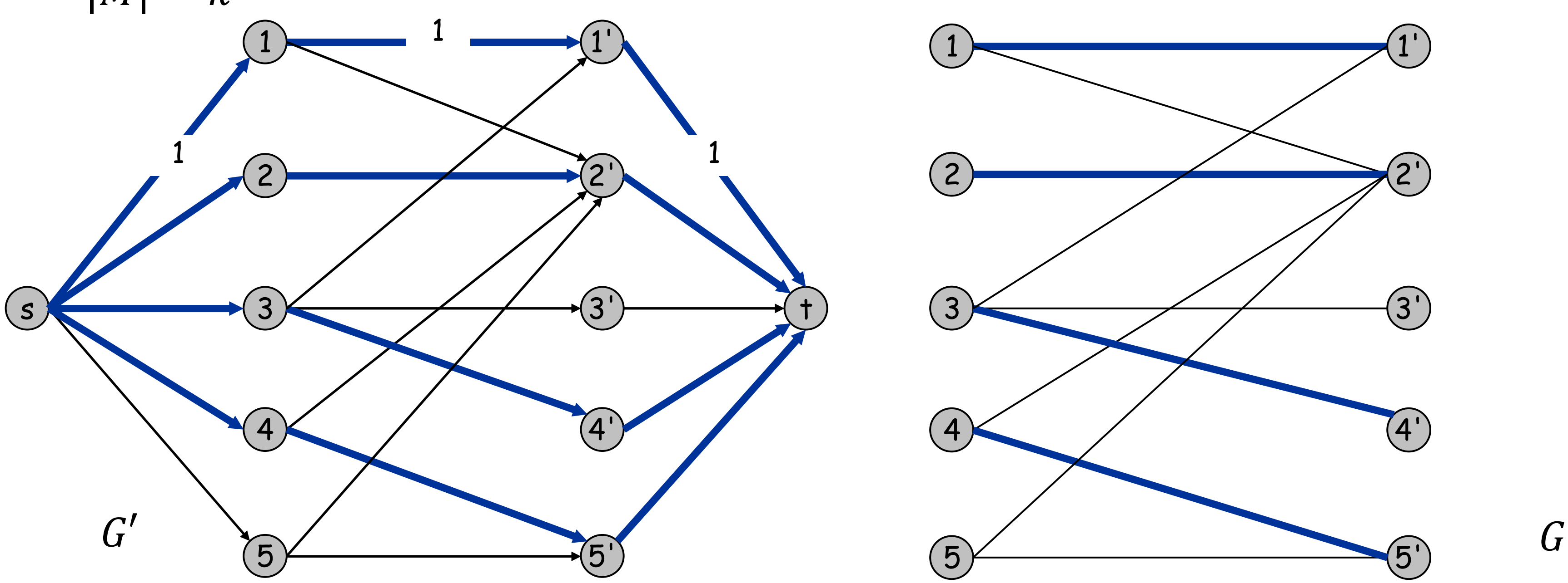


# Maximum Bipartite Matching: Proof of Correctness

**Theorem.** Max matching in $G$ = value of max flow in $G'$.

**Pf.** A max flow of value $k$ in $G'$ $\Rightarrow$ a matching with cardinality $k$ in G

- Let $f$ be a max flow in $G'$ of value $k$ computed by Ford-Fulkerson
- $f(e) = 1$ or $0$ for every edge $e$ (because of integrality of F-F solution)
- Consider $M$ = set of edges from $X$ to $Y$ with $f(e) = 1$.
  - each node in $X$ and $Y$ participates in at most one edge in $M$
  - $|M| = k$



# Maximum Bipartite matching: Running time

Algorithm:

- Run F-F on the constructed graph $G'$
- Report all original edges from $G$ that have flow 1
- Correct by analysis on previous page
- Correct no matter how augmenting paths are chosen

Running time: $O(VE)$

- Each iteration increases $|f|$ by 1.
- $|f^*| \leq V/2$
- Each iteration takes $O(E)$ time.

Specialized algorithms

- $O(\sqrt{V}E)$ [Hopcroft–Karp, 1973]
- $O(V^{2.376})$ using matrix multiplication [Mucha-Sankowski, 2004]
- $O(E^{10/7})$ [Madry, 2013]
- Now all subsumed by the $O\left(E^{1+o(1)} \log U\right)$ algorithm in 2022.

# Bipartite Matching : Feasible Schedule

Assume $n$ roommates $r_1, \ldots, r_n$.

For fairness, every day $d_1, \ldots, d_n$ a different roommate is supposed to cook dinner.

However, due to other obligations, some roommates are unable to cook on certain days.

Let $c_{i,j} = true$, if $r_i$ can cook on day $d_j$.

Describe an algorithm to determine if is possible to have a feasible schedule such that each roommate cooks exactly once during the $n$ days.

# Bipartite Matching: Feasible Schedule

$c_{i,j} = true$, if $r_i$ can cook on day $d_j$.

Describe an algorithm to determine if is possible to have a feasible schedule such that each roommate cooks exactly once during the $n$ days.

Solution: This is a matching problem.

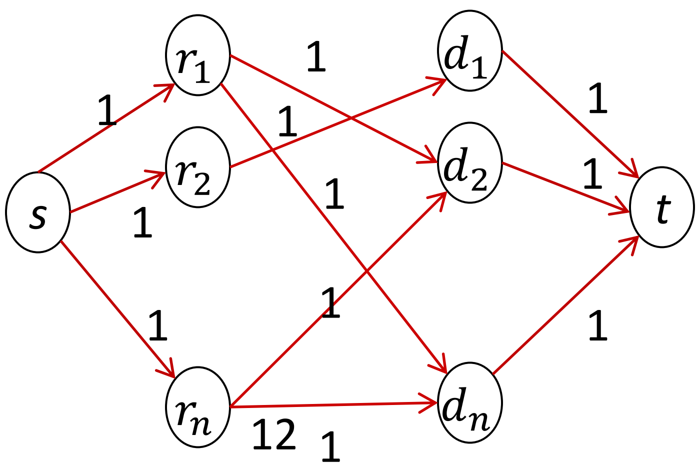
Create **bipartite graph** in which each roommate $r_1, \ldots, r_n$ and each day $d_1, \ldots, d_n$ are nodes.

Construct edge $(r_i, d_j)$ iff $c_{i,j} = true$.

Add source node $s$ with outgoing edges to all roommates $r_1, \ldots, r_n$,
and terminal node $t$ with incoming edges from all days $d_1, \ldots, d_n$.
Set all edge capacities equal 1.

A feasible schedule exists if and only if
The bipartite graph has a perfect matching, i.e.,
A matching touching every vertex.

This happens iff
the max $s-t$ flow has value $n$.



# Bipartite Matching: Balanced Assignment

Your company wishes to assign $n$ customers $c_1, \dots, c_n$ to $k$ facilities $f_1, \dots, f_k$.

Each customer can only be served by some facility in his vicinity:

$c_{i,j} = true$ means that customer $c_i$ can be served by facility $f_j$.

An **assignment** of customers to facilities is balanced,

if each facility serves the same number $n/k$ of customers (assume that $n/k$ is integer).

Given the constraints $c_{i,j}$, describe an algorithm to determine if is possible to construct a balanced assignment

# Bipartite Matching: Balanced Assignment

$c_{i,j} = true$ means that customer $c_i$ can be served by facility $f_j$.
Given constraints $c_{i,j}$, describe an algorithm to determine if is possible to construct a balanced assignment

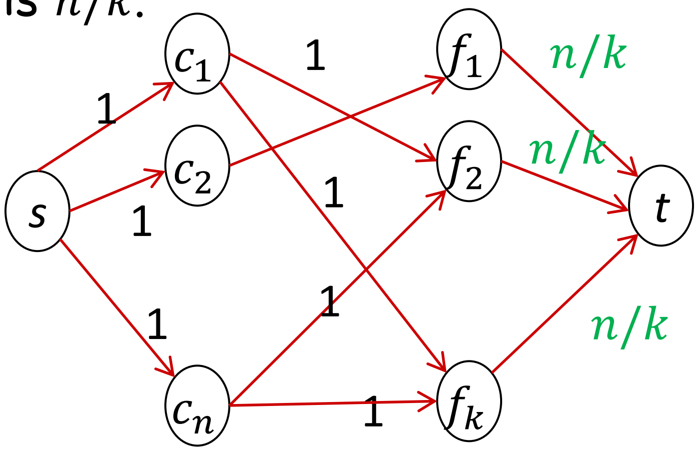
Solution: Create a bipartite graph.

Each customer $c_1, \ldots, c_n$ and each facility $f_1, \ldots, f_k$ are nodes.

Edge $(c_i, f_j)$ exists iff $c_{i,j} = true$.

Add source $s$ connected to all customers $c_1, \ldots, c_n$,
and terminal node $t$ with incoming edges from all facilities $f_1, \ldots, f_k$.
All edge capacities = 1,
except for the edges $(f_j, t)$ whose capacity is $n/k$.

A balanced assignment exists
if and only if
maximum $s{-}t$ flow has value $n$.



# Bipartite Matching: Constrained Assignment

Your company now wishes to assign $n$ customers $c_1, \ldots, c_n$ to $k$ facilities $f_1, \ldots, f_k$.

Each customer can only be served by some facility in his vicinity:
$c_{i,j} = true$ means that customer $c_i$ can be served by facility $f_j$

An **assignment** of customers to facilities is constrained,
so that facility $f_i$ can serve $n_i$ customers where $\sum_{i=1}^{k} n_i = n$.

Given the constraints $c_{i,j}$ and the $n_i$, describe an algorithm to determine if is possible to construct a constrained assignment that serves all of the customers and, if such an assignment exists, to construct it.

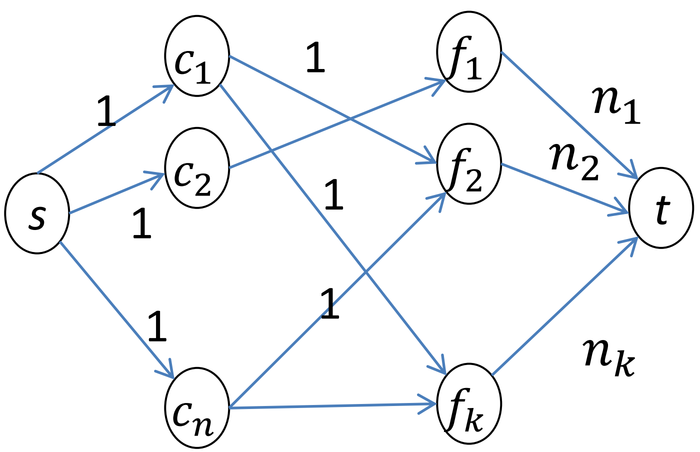
# Bipartite Matching: Constrained Assignment

$c_{i,j} = true$ means that customer $c_i$ can be served by facility $f_j$.
Facility $f_i$ serves at most $n_i$ customers where $\sum_{i=1}^{k} n_i = n$.

Describe an algorithm to determine if is possible to construct a constrained assignment given the constraints $c_{i,j}$ and values $n_i$

Solution: Create a bipartite graph in which each customer $c_1, \ldots, c_n$ and each facility $f_1, \ldots, f_k$ are nodes.

Edge $(c_i, f_j)$ exists iff $c_{i,j} = true$.
Add source $s$ with outgoing edges to customers $c_1, \ldots, c_n$
terminal $t$ with incoming edges from all facilities $f_1, \ldots, f_k$
All edge capacities equal 1, except for the edges $(f_i, t)$ whose capacity is $n_i$



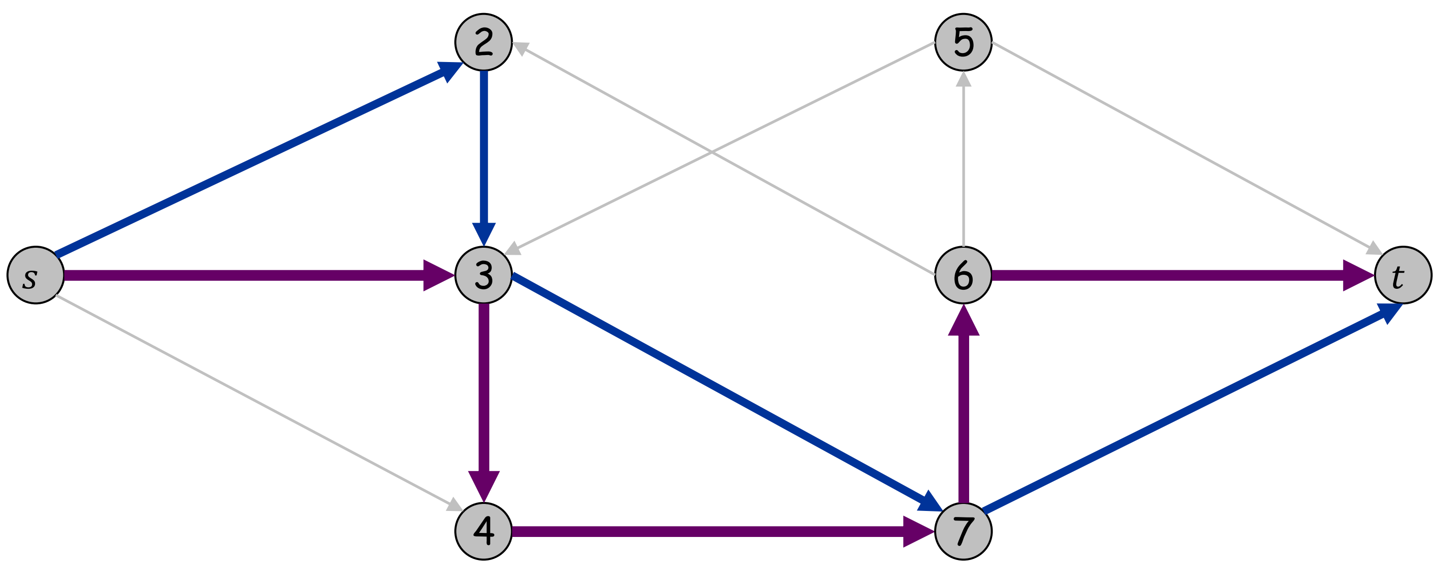

A constrained assignment exists
if and only if
maximum $s-t$ flow has value $n$.

# Edge Disjoint Paths

Disjoint path problem. Given a directed graph $G = (V, E)$ and two nodes $s$ and $t$, find the max number of edge-disjoint $s$–$t$ paths.

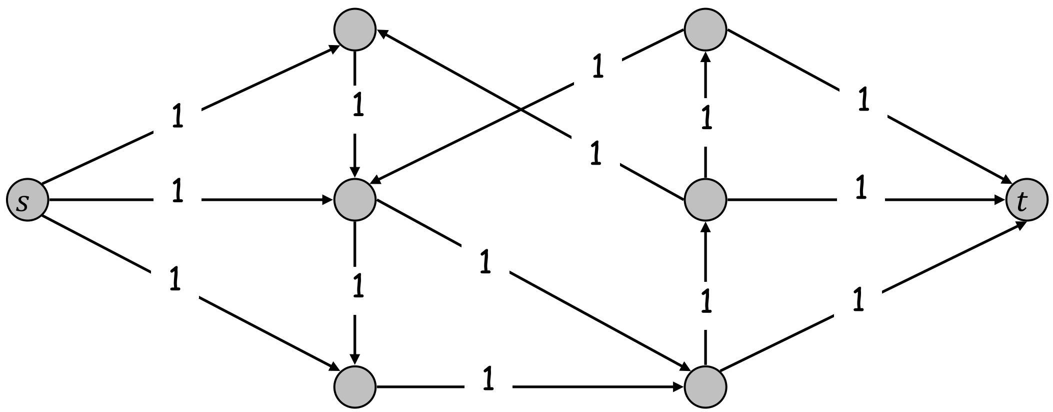
Def. Two paths are edge-disjoint if they have no edge in common.

Application: Communication networks.



# Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



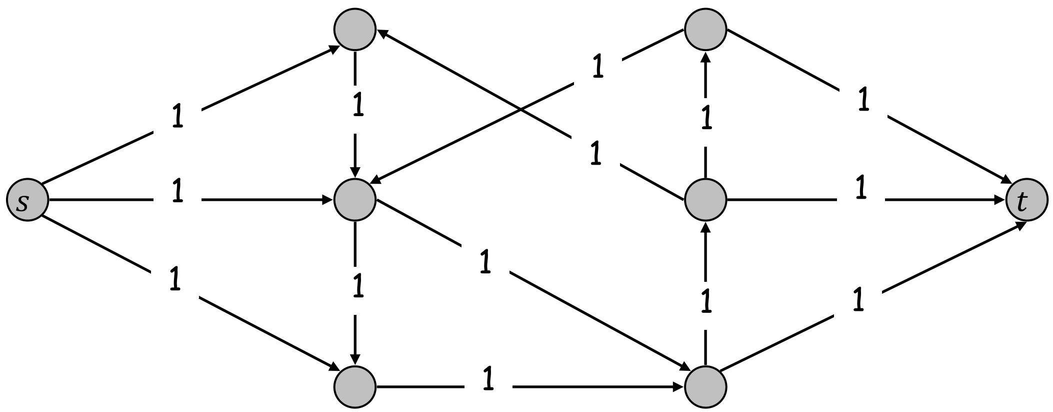

Theorem. Max number edge-disjoint $s$–$t$ paths equals max flow value.

Proof. max $k$ edge-disjoint $s$–$t$ paths ⇒ flow of value $k$

- Suppose there exists $k$ edge-disjoint paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if $e$ participates in some path $P_i$; else set $f(e) = 0$.
- Since paths are edge-disjoint, $f$ is a flow of value $k$.

# Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.




Proof. max flow of value $k \Rightarrow k$ edge-disjoint $s$–$t$ paths

- Let $f$ be a max flow in $G'$ of value $k$ computed by Ford-Fulkerson
- $f(e) = 1$ or $0$ for every edge $e$ (integrality property).

- Consider any edge $(s, u)$ with $f(s, u) = 1$.
  - By conservation, there exists edge $(u, v)$ with $f(u, v) = 1$
  - Continue to find the next unused edge out of $v$ until reaching $t$.

- After finding one path, flow value decreases by 1.
- Repeat the process $k$ times to find $k$ edge-disjoint paths.
- The proof above also provides an **algorithm**.

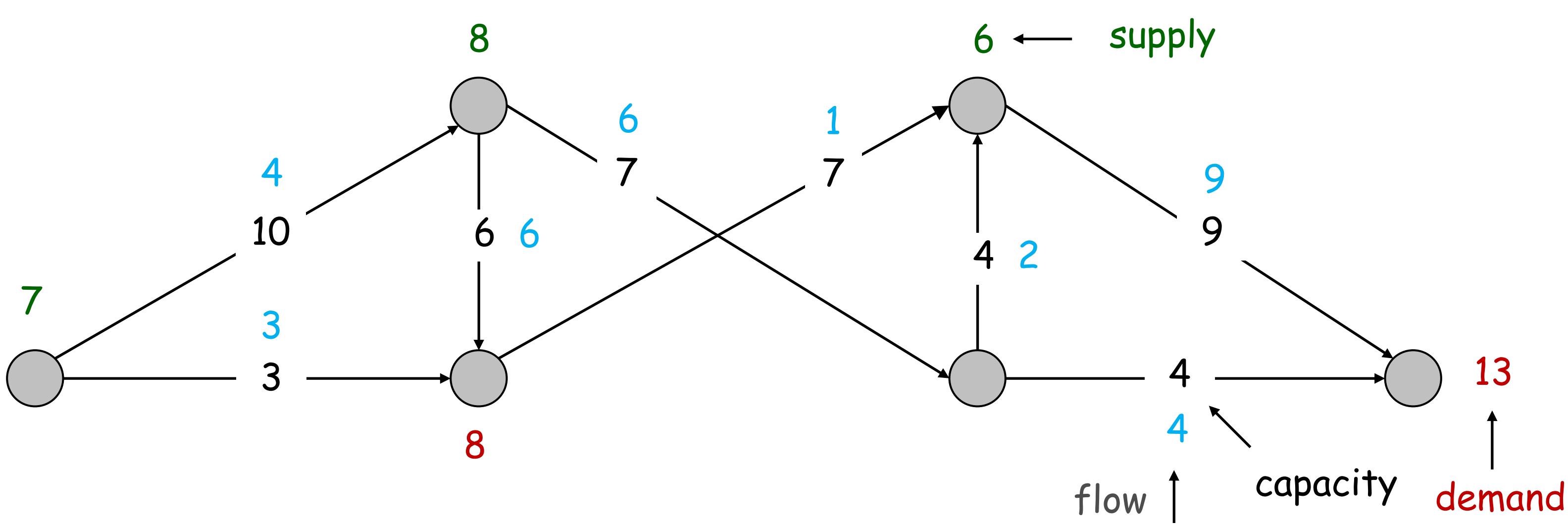
# Circulation with Demands

Input: A directed connected graph $G = (V, E)$, in which

- every edge $e \in E$ has a capacity $c(e)$;
- a number of source vertices $s_1, s_2, \ldots$, each with a supply of $sup(s_i)$ and a number of target vertices $t_1, t_2, \ldots$, each with a demand of $dem(t_i)$;
- $\sum_i sup(s_i) \geq \sum_i dem(t_i)$

Output: A flow $f$ that meets capacity and conservation conditions, and

- At each source vertex $s_i$, $\sum_{e \text{ out of } s_i} f(e) - \sum_{e \text{ into } s_i} f(e) \leq sup(s_i)$;
- At each target vertex $t_i$, $\sum_{e \text{ into } t_i} f(e) - \sum_{e \text{ out of } t_i} f(e) = dem(t_i)$.

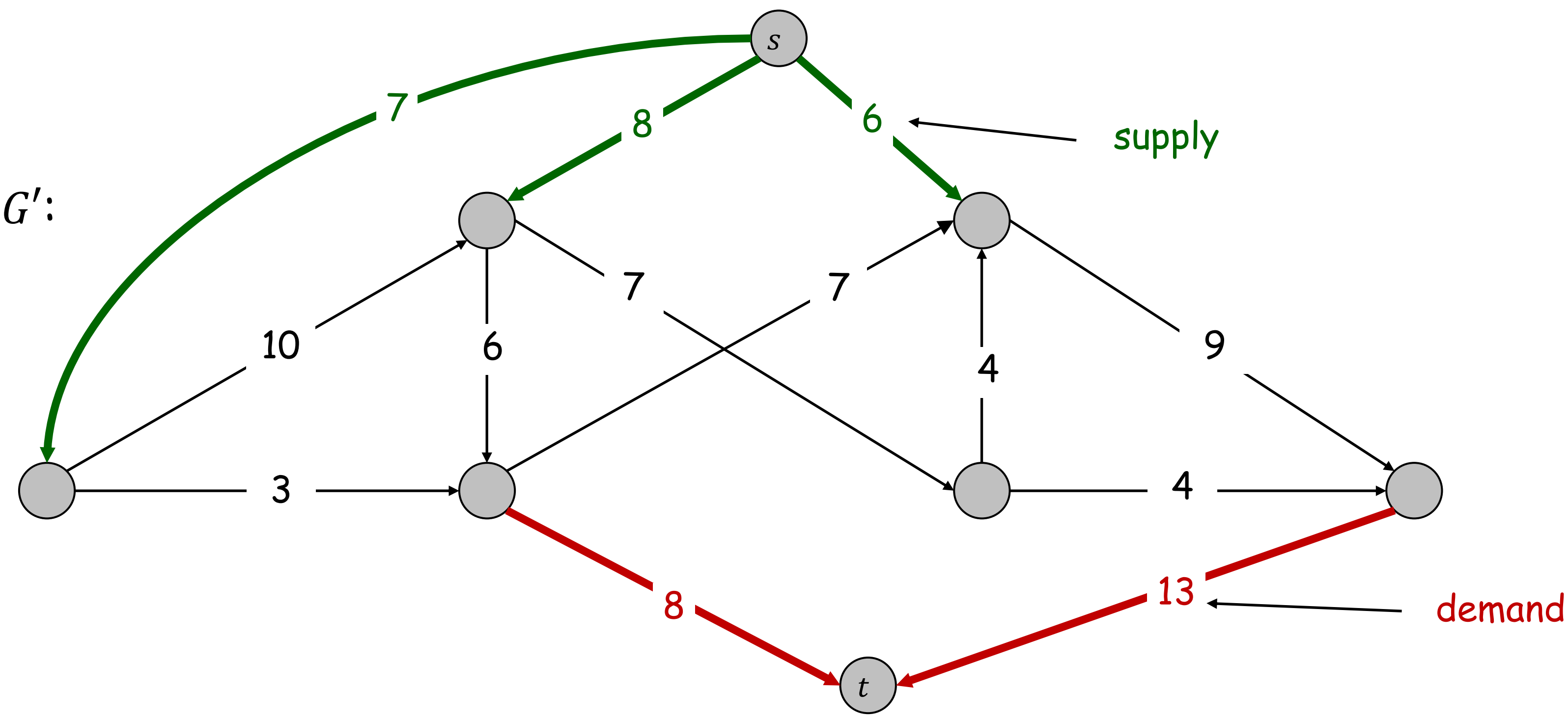


# Solving Circulation with Demands using Max Flow

Algorithm:

- Add a "super source" $s$ and a "super target" $t$.
- Add an edge from $s$ to each $s_i$ with capacity $sup(s_i)$.
- Add an edge from each $t_i$ to $t$ with capacity $dem(t_i)$.
- Compute the max flow $f$.
- If $|f| = \sum_i dem(t_i)$, then return $f$; else return "no solution".

$G'$:



# Baseball (Basketball) Elimination

| Team $i$ | Wins $w_i$ | To play $r_i$ | Remaining Against = $r_{ij}$ | | | |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 1 | 3 | 2 | - | 1 | 1 | 0 |
| 2 | 2 | 3 | 1 | - | 1 | 1 |
| 3 | 2 | 3 | 1 | 1 | - | 1 |
| 4 | 0 | 2 | 0 | 1 | 1 | - |

Rule: Order teams by the number of wins. Each win is 1 point. There are no ties in the games. Losses are 0 points.

Q: Does Team 4 still have a chance to finish in first place (tie is OK)?

A: No, obviously. Even if it wins last 2 games, it will have 2 points, whereas team 1 has already 3 points.

# Basketball Elimination

| Team $i$ | Wins $w_i$ | To play $r_i$ | Remaining Against = $r_{ij}$ | | | |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 1 | 3 | 2 | - | 1 | 1 | 0 |
| 2 | 2 | 3 | 1 | - | 1 | 1 |
| 3 | 2 | 3 | 1 | 1 | - | 1 |
| 4 | 1 | 2 | 0 | 1 | 1 | - |

Q: Does Team 4 still have a chance to finish in first place (tie is OK)?

A: No, because

- Team 4 has to win both remaining games against team 2 and 3.
- Team 1 has to lose both remaining games against team 2 and 3.
- Then 2 and 3 will both have 3 wins.
- The game between team 2 and 3 will give one of them one more win.

# Baseball Elimination: Definition

**Input:**

- $n$ teams: $1, 2, \dots, n$
- One particular team, say $n$ (without loss of generality)
- Team $i$ has won $w_i$ games already
- Team $i$ and $j$ still need to play $r_{ij}$ games
- Team $i$ has a total of $r_i = \sum_j r_{ij}$ games to play

**Output:**

- "Yes", if there is an outcome for each remaining game such that team $n$ finishes with the most wins (tie is OK).
- "No", if no such possibilities.

**Brute-force algorithm:**

- For each remaining game, consider two possible outcomes.
- Try all $2^r$ possible combinations, where $r = \sum_{i,j} r_{ij}$

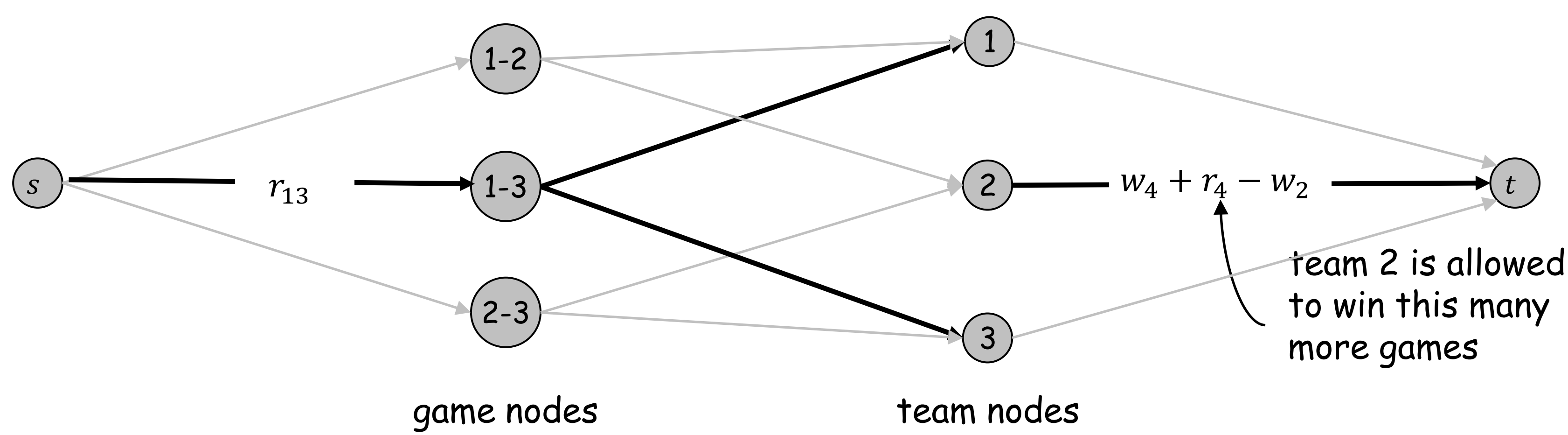
# Baseball Elimination: Max Flow Formulation

Can team $n$ (in this example team 4) finish with most wins?

- Assume team $n$ wins all remaining games $\Rightarrow w_n + r_n$ wins.
- All other teams must have $\leq w_n + r_n$ wins.

Flow network construction:

- A source $s$ and a target $t$
- A game node for each $(i, j)$; and an edge from $s$ to it with capacity $r_{ij}$
- A node for each team $i = 1, 2, \dots, n-1$; and an edge from it to $t$ with capacity $w_n + r_n - w_i$
- Game node $(i, j)$ has edges to team node $i$ and $j$, with capacity $\infty$



# Baseball Elimination: Max Flow Formulation

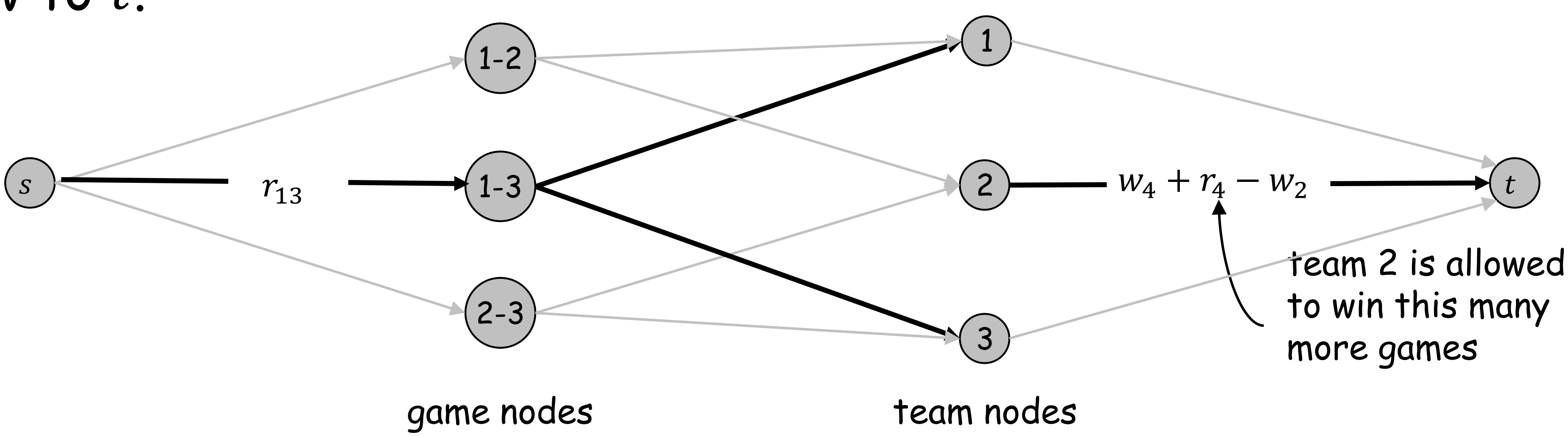
Claim: There is a way for team $n$ to finish in the first place iff the max flow has value $r = \sum_{i,j} r_{ij}$.

Proof: "⇒": Suppose there is an outcome for each remaining game such that team $n$ finishes the first. First set $f\big(s, (i,j)\big) = r_{ij}$ for all $(i,j)$.

For each game node $(i,j)$:

- if $i$ wins $x$ games against $j$, set $f\big((i,j), i\big) = x$ and $f\big((i,j), j\big) = r_{ij} - x$;

Team $i$ wins $\leq w_n + r_n - w_i$ games, so it can send all incoming flow to $t$.



# Baseball Elimination: Max Flow Formulation

Proof: "$\Leftarrow$": Suppose the max flow $f$ has $|f| = r$. It must saturate all edges out of $s$.

Look at each game node $(i, j)$.

- Let team $i$ win $f\big((i,j), i\big)$ games against $j$
- Let team $j$ win $f\big((i,j), j\big)$ games against $i$
- Note: $f\big((i,j), i\big) + f\big((i,j), j\big) = r_{ij}$ and they must be integers

Team node $i$ receives $\leq w_n + r_n - w_i$ units of flow, each corresponding to one win, so it cannot beat team $n$.