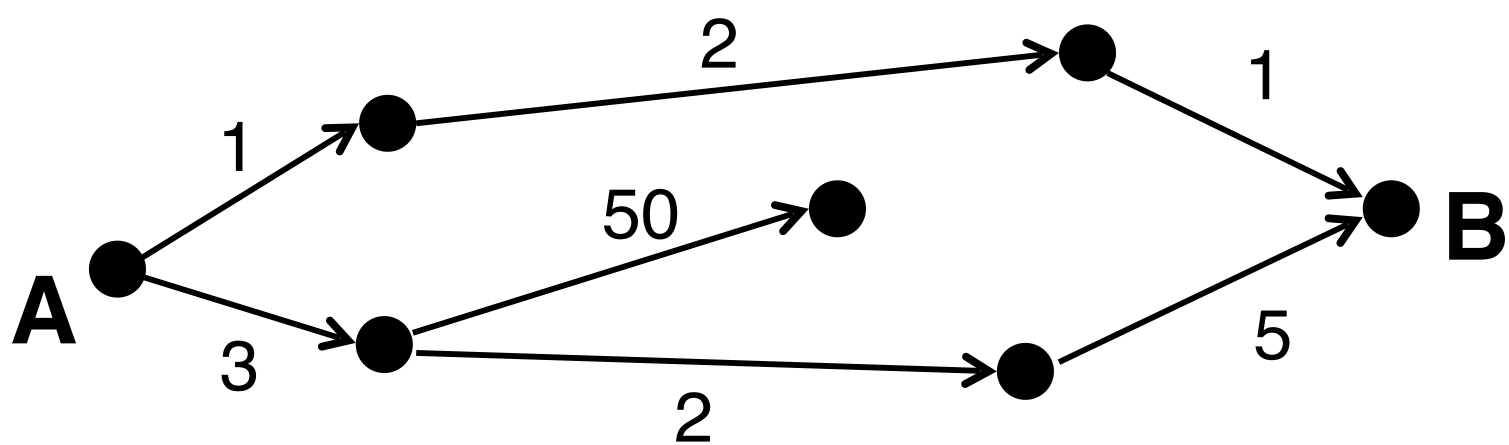
Maximum Flow

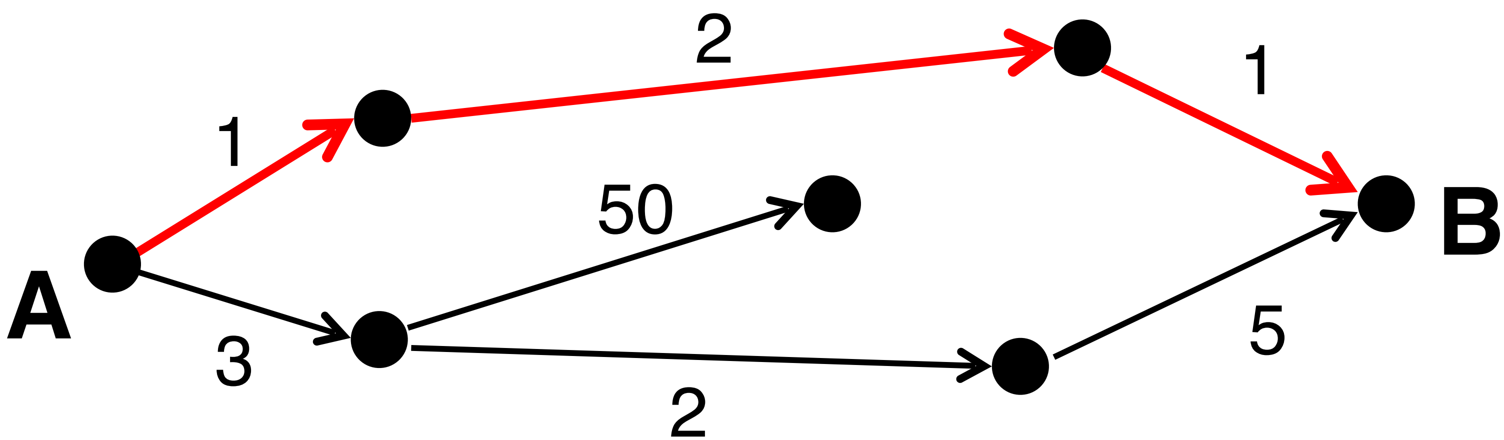
Problem Statement

Computer A wants to send packets to computer B. Each link in the network has a throughput of $\mathbf{w}_i$ packets/second. What is most packets A can send to B each second?



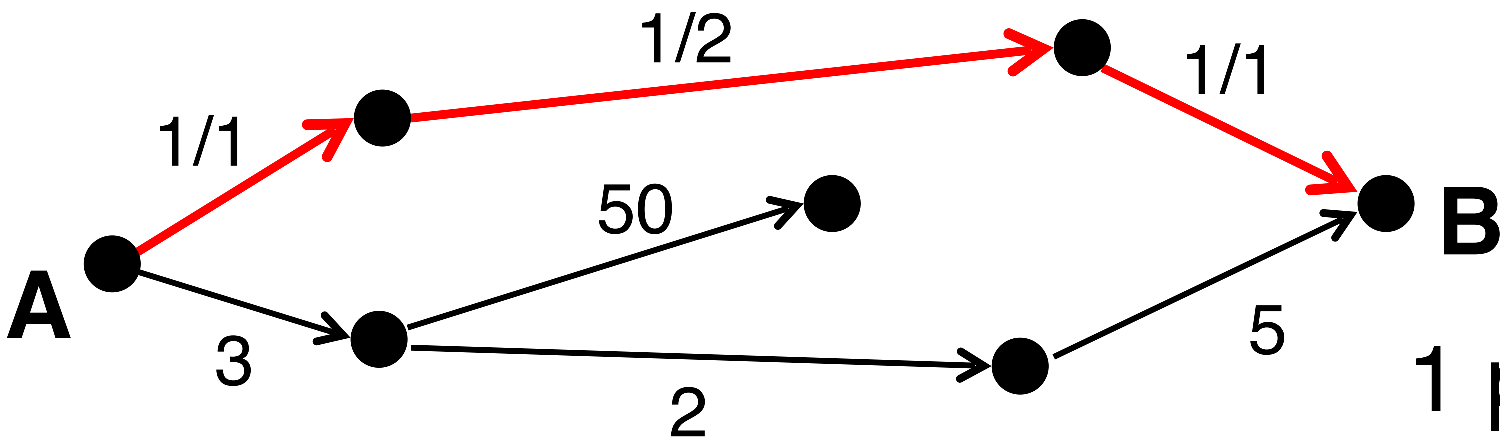
Problem Statement

Computer A wants to send packets to computer B. Each link in the network has a throughput of $\mathbf{w}_i$ packets/second. What is most packets A can send to B each second?



Problem Statement

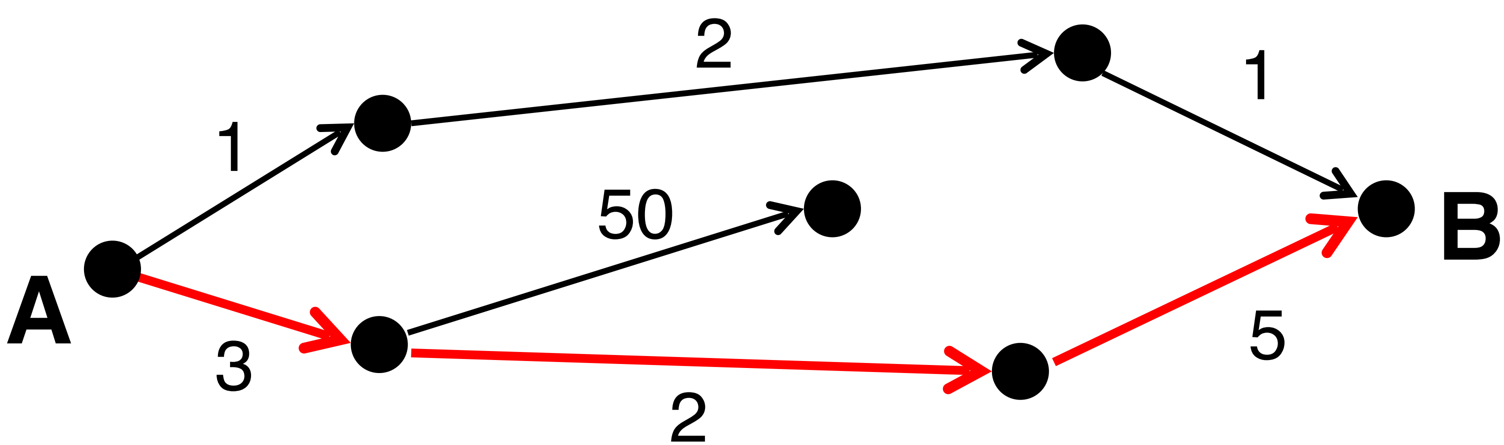
Computer A wants to send packets to computer B. Each link in the network has a throughput of $\mathbf{w}_i$ packets/second. What is most packets A can send to B each second?



1 packet/s

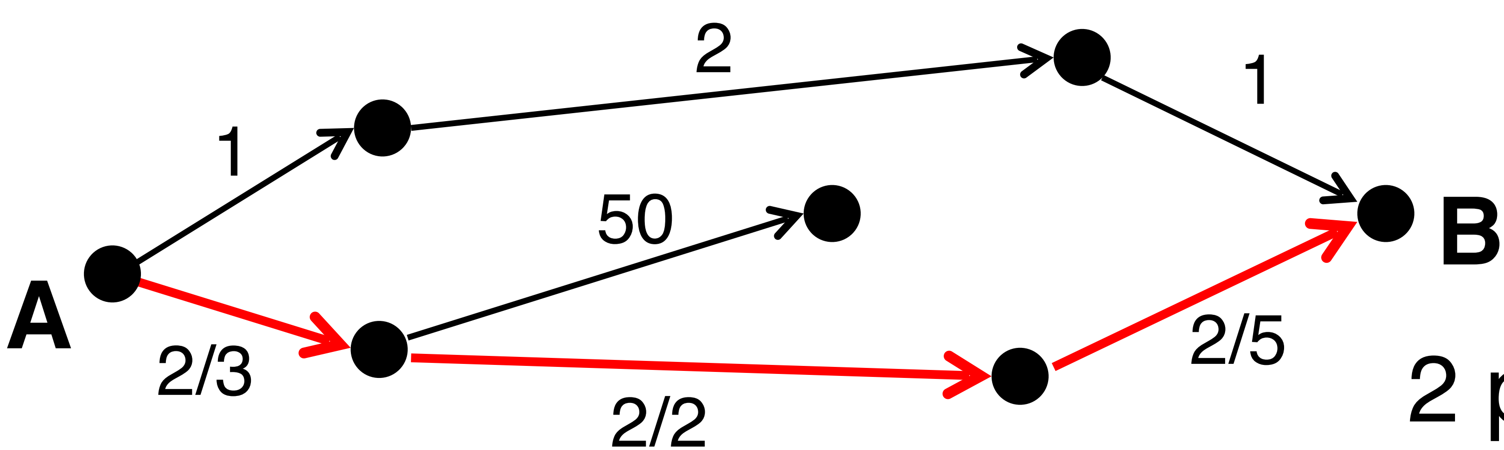
Problem Statement

Computer A wants to send packets to computer B. Each link in the network has a throughput of $\mathbf{w}_i$ packets/second. What is most packets A can send to B each second?



Problem Statement

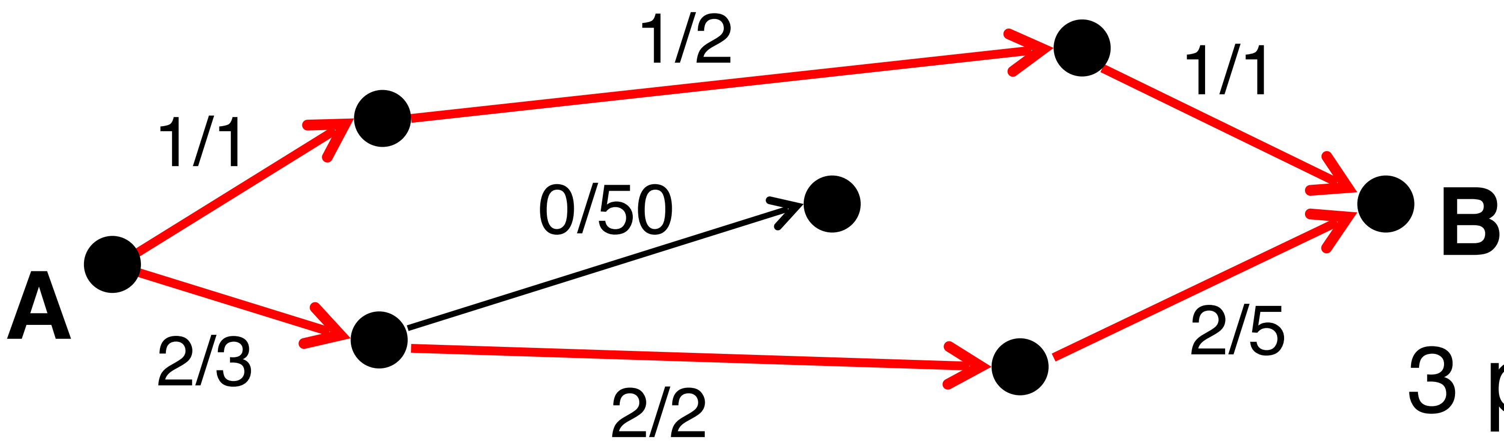
Computer A wants to send packets to computer B. Each link in the network has a throughput of $\mathbf{w}_i$ packets/second. What is most packets A can send to B each second?



2 packet/s

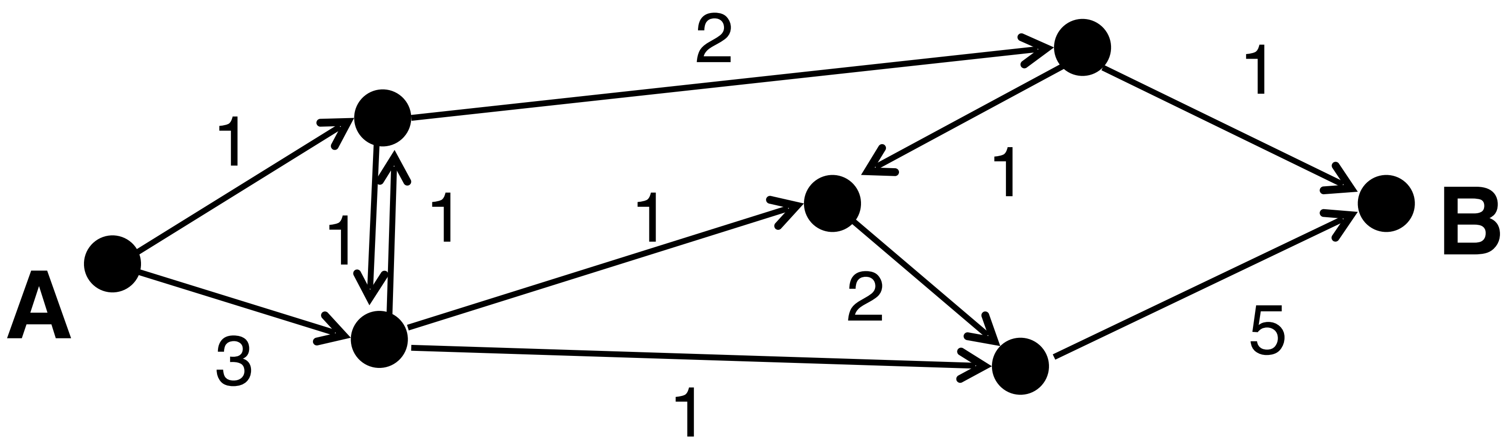
Problem Statement

Computer A wants to send packets to computer B. Each link in the network has a throughput of $\mathbf{w}_i$ packets/second. What is most packets A can send to B each second?

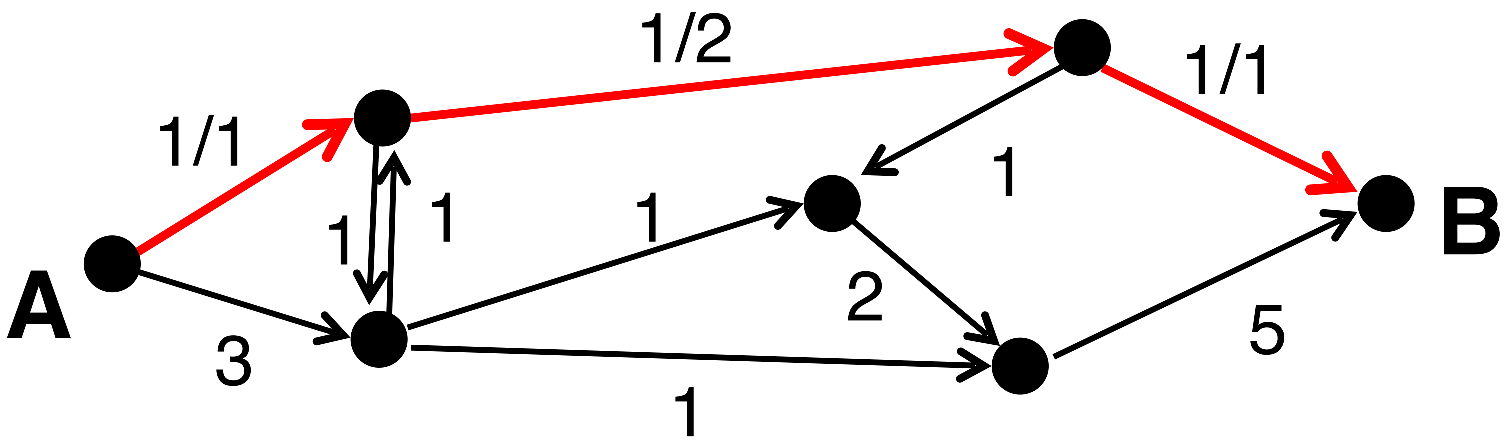


3 packet/s

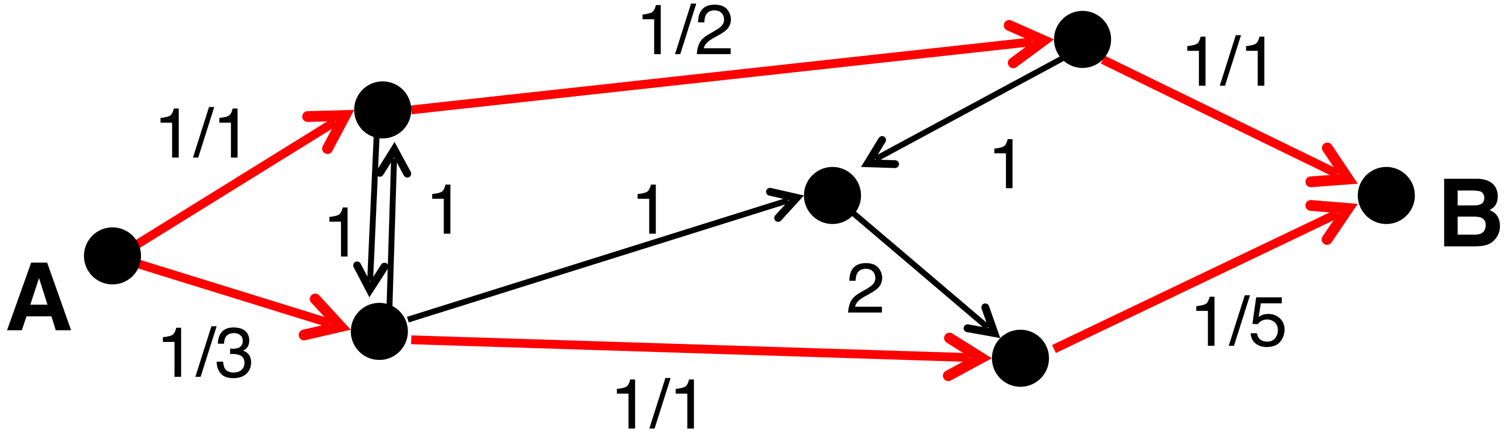
Flows Can Be More Complex



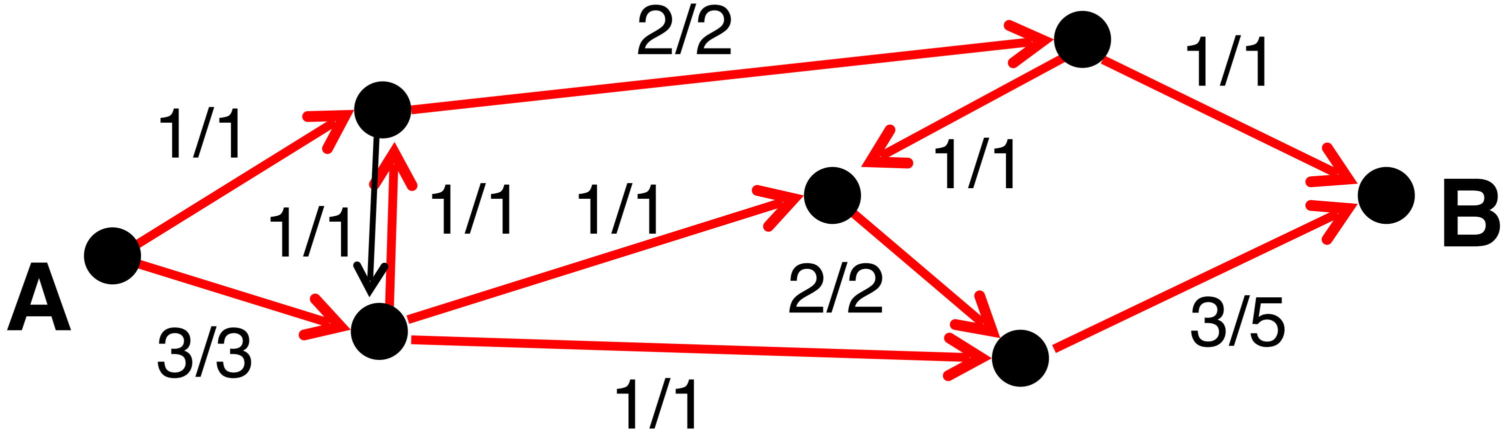
Flows Can Be More Complex



Flows Can Be More Complex



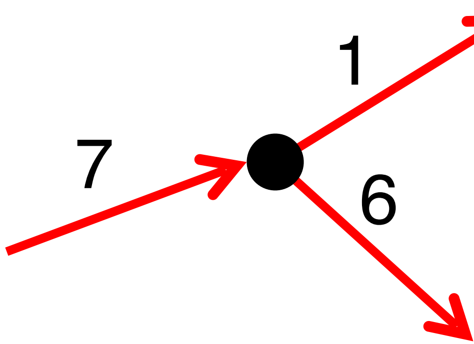
Flows Can Be More Complex



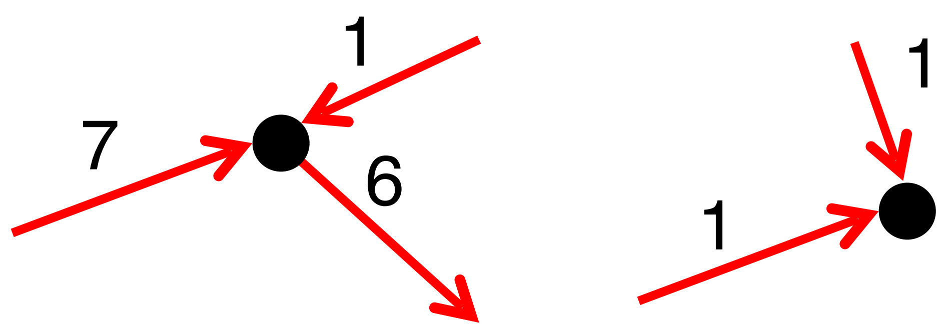
Network Flow: Formal Definition

Given a directed weighted graph, assign a flow to each edge so that

1. At each vertex, flow in == flow out



OK



WRONG

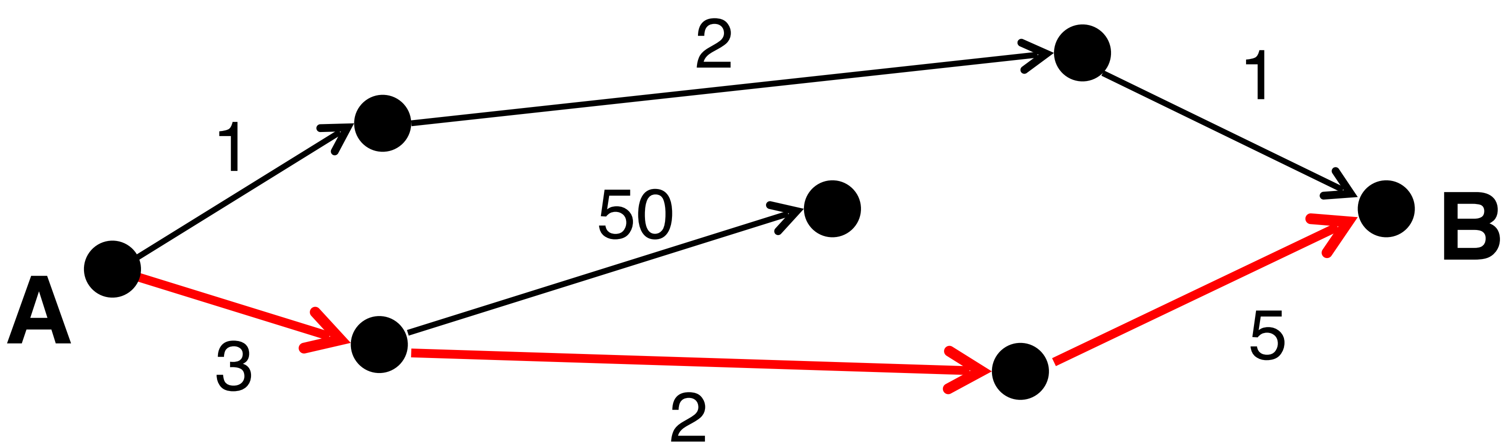
Network Flow: Formal Definition

Given a directed weighted graph, assign a flow to each edge so that

1. At each vertex, flow in == flow out
   - (except at **source** and **sink** vertex)
2. Capacity constraints satisfied

Greedy Algorithm?

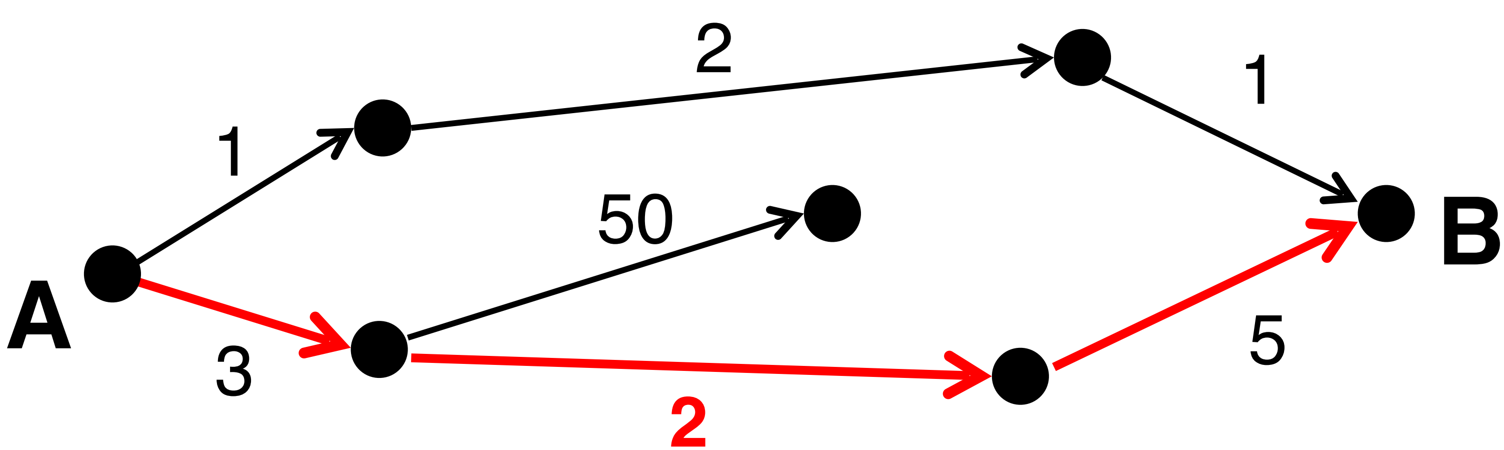
1. Find path from A to B (BFS)



Greedy Algorithm?

1. Find path from A to B (BFS)

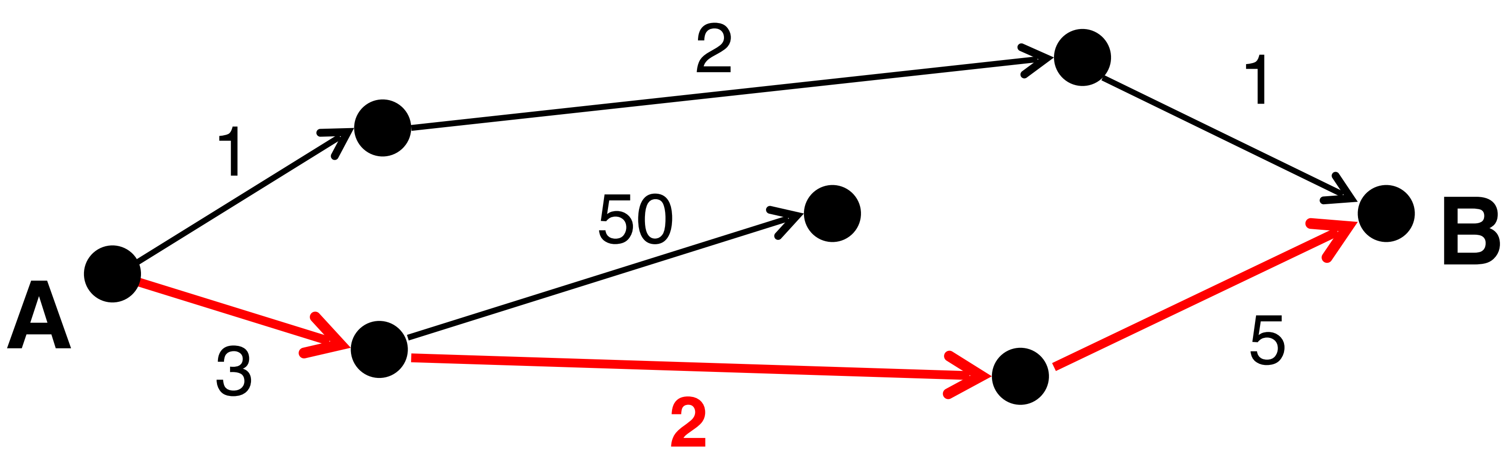
2. Find bottleneck capacity



Greedy Algorithm?

1. Find path from A to B (BFS)

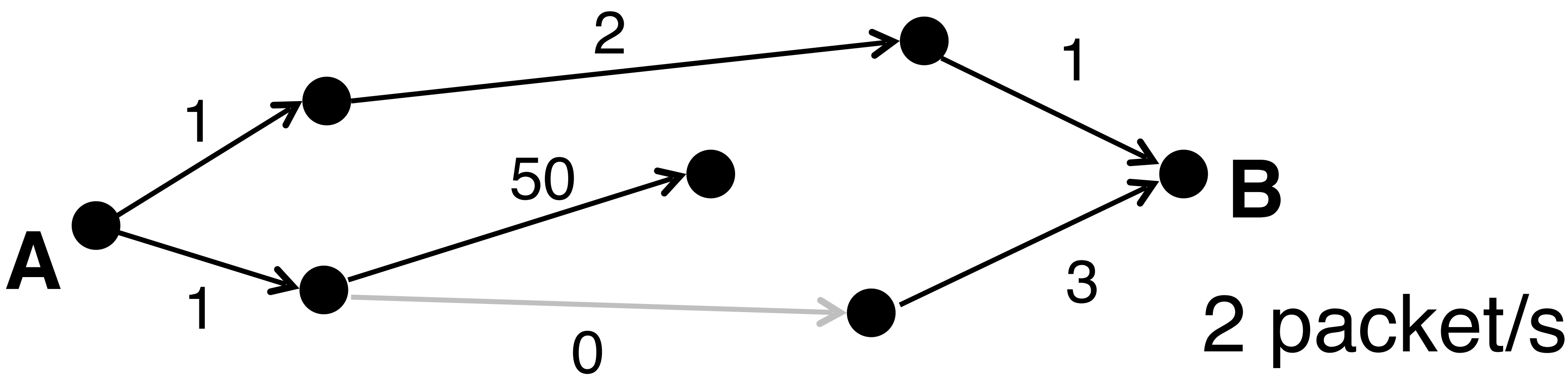
2. Find bottleneck capacity
3. Add to total / subtract from capacities



Greedy Algorithm?

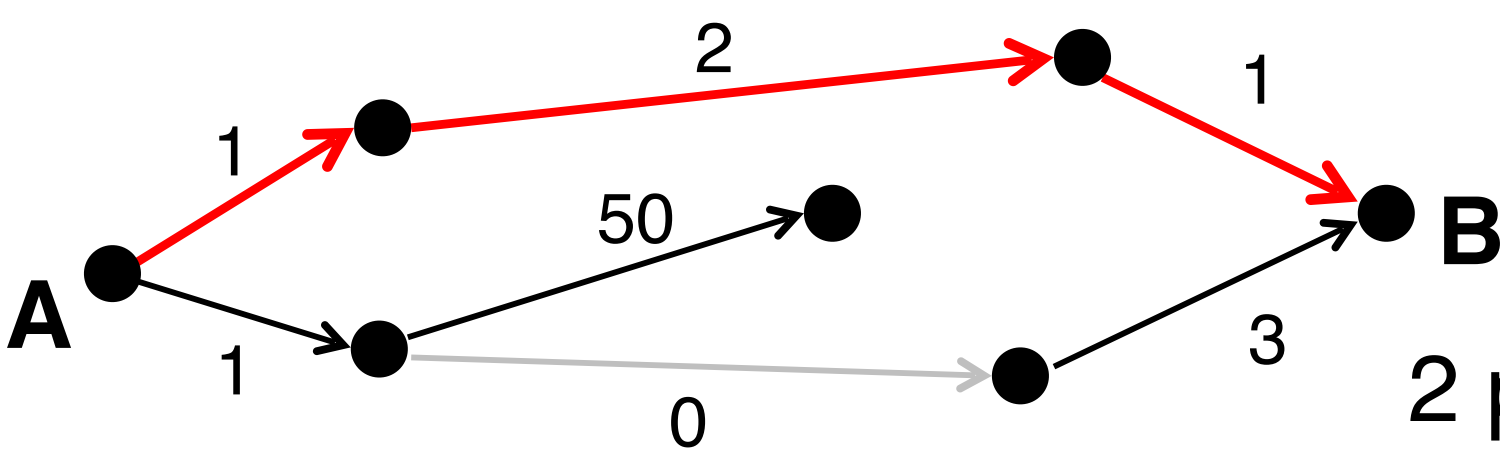
1. Find path from A to B (BFS)

2. Find bottleneck capacity
3. Add to total / subtract from capacities



Greedy Algorithm?

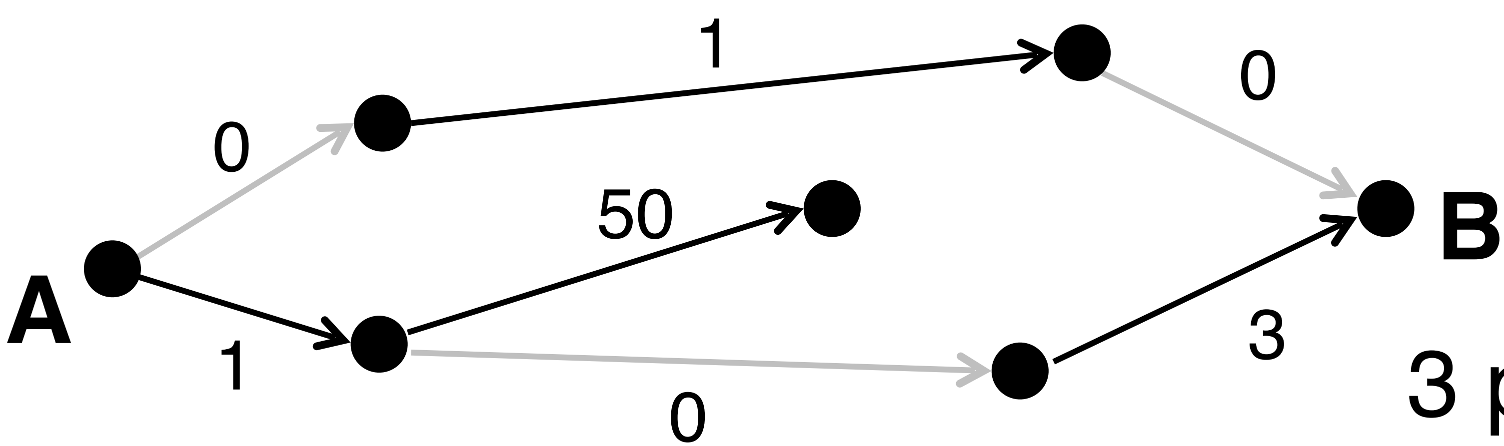
1. Find path from A to B (BFS)
   (ignore edges with no capacity)
2. Find bottleneck capacity
3. Add to total / subtract from capacities



2 packet/s

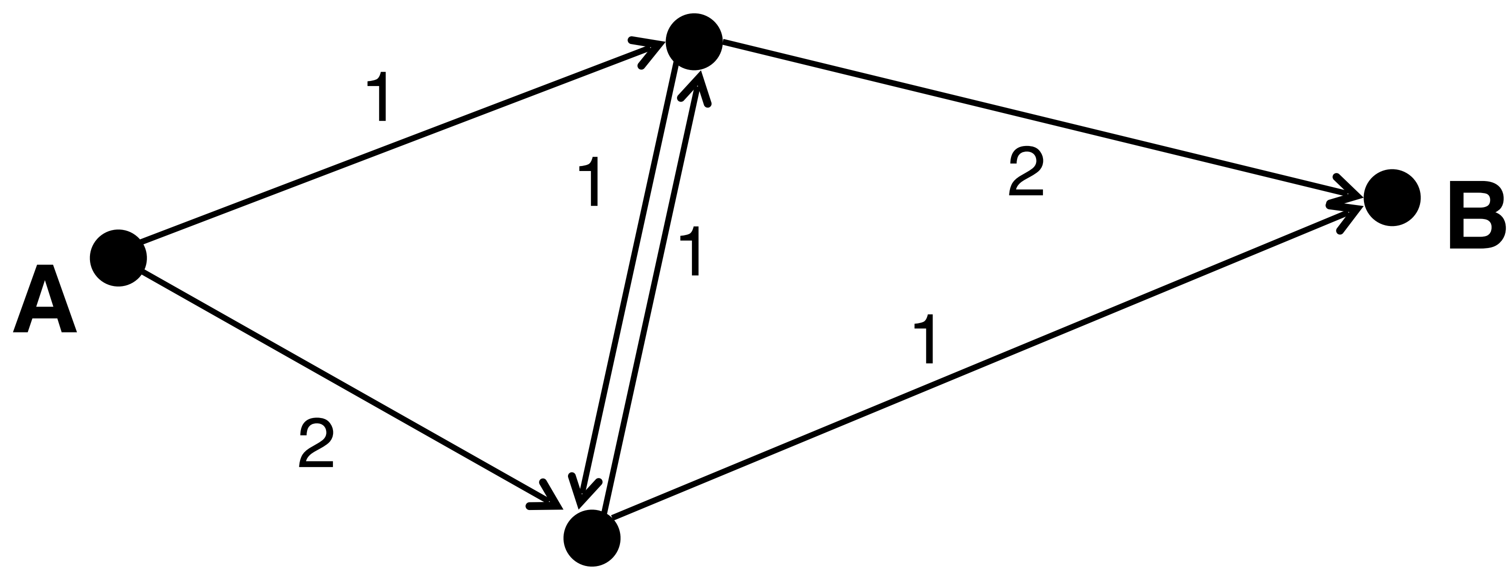
Greedy Algorithm?

1. Find path from A to B (BFS)
   (ignore edges with no capacity)
2. Find bottleneck capacity
3. Add to total / subtract from capacities



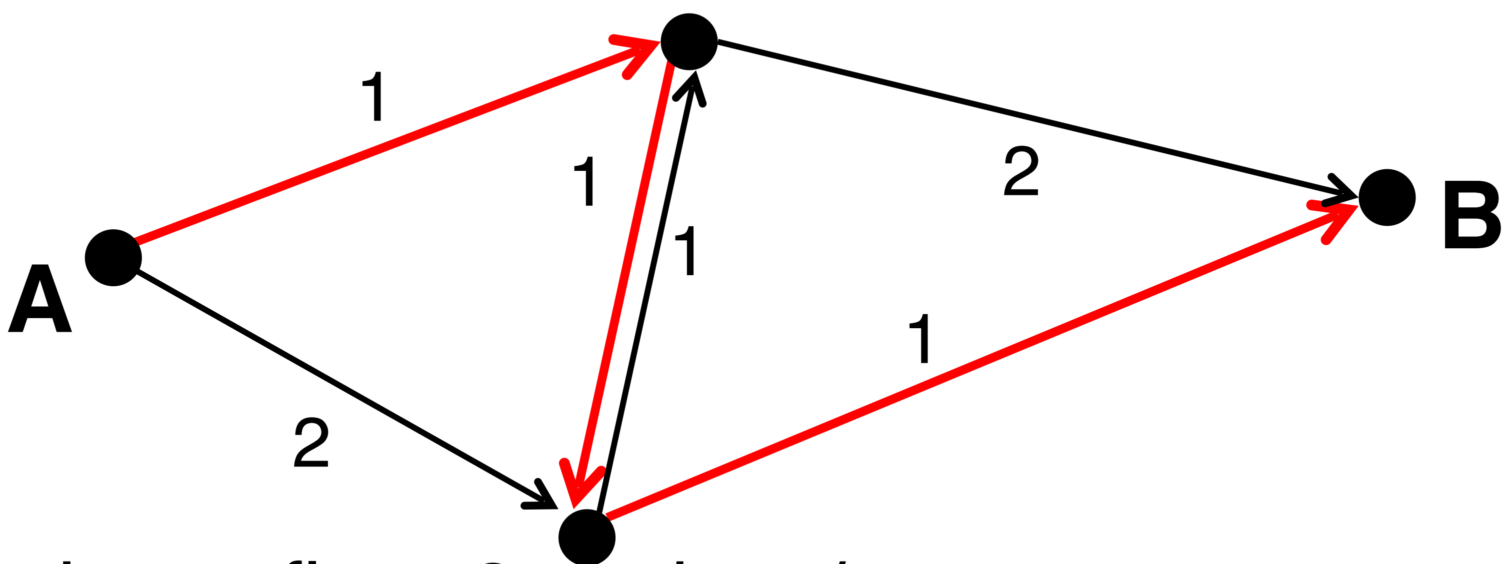
3 packet/s

Greedy Algorithm?



Maximum flow:

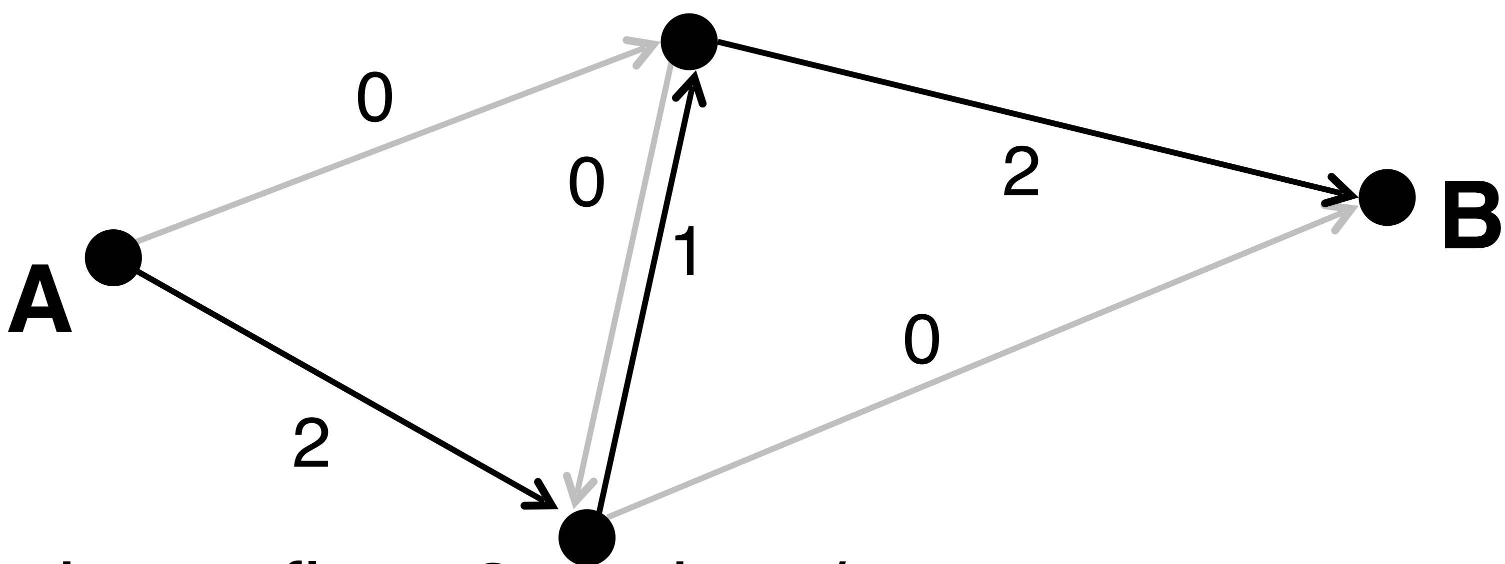
Greedy Algorithm?



Maximum flow: 3 packets/sec

Algorithm returns:

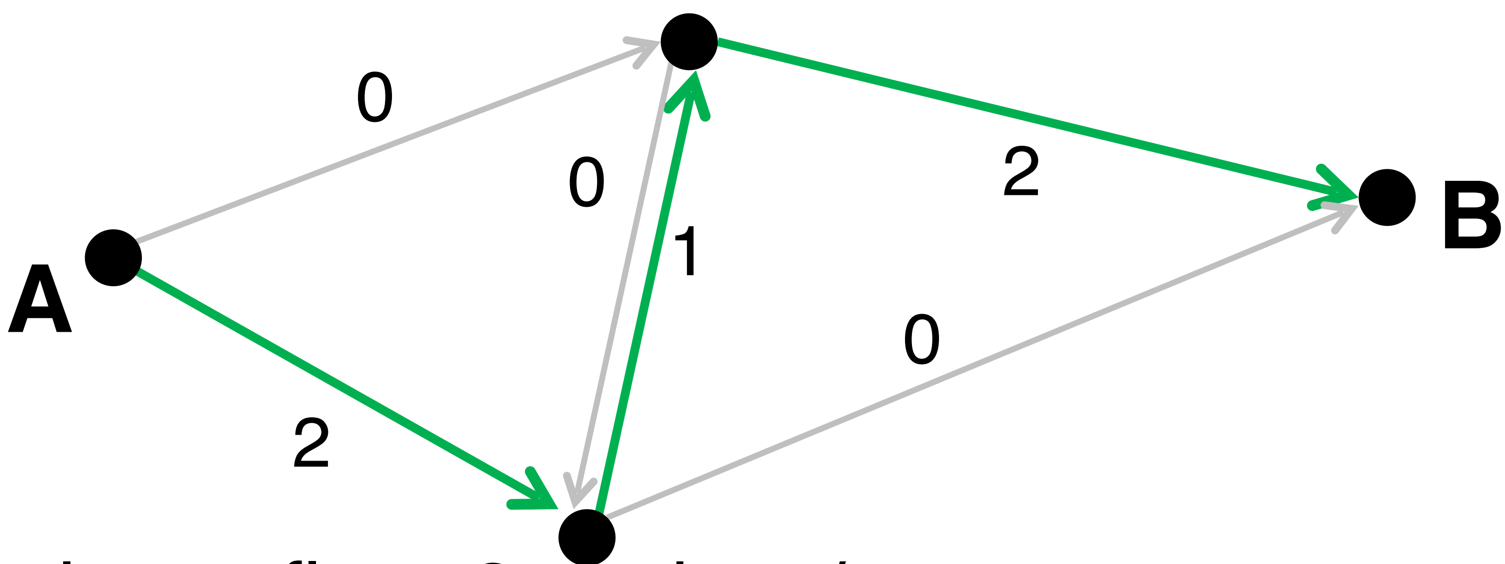
Greedy Algorithm?



Maximum flow: 3 packets/sec

Algorithm returns: 1 packet/sec

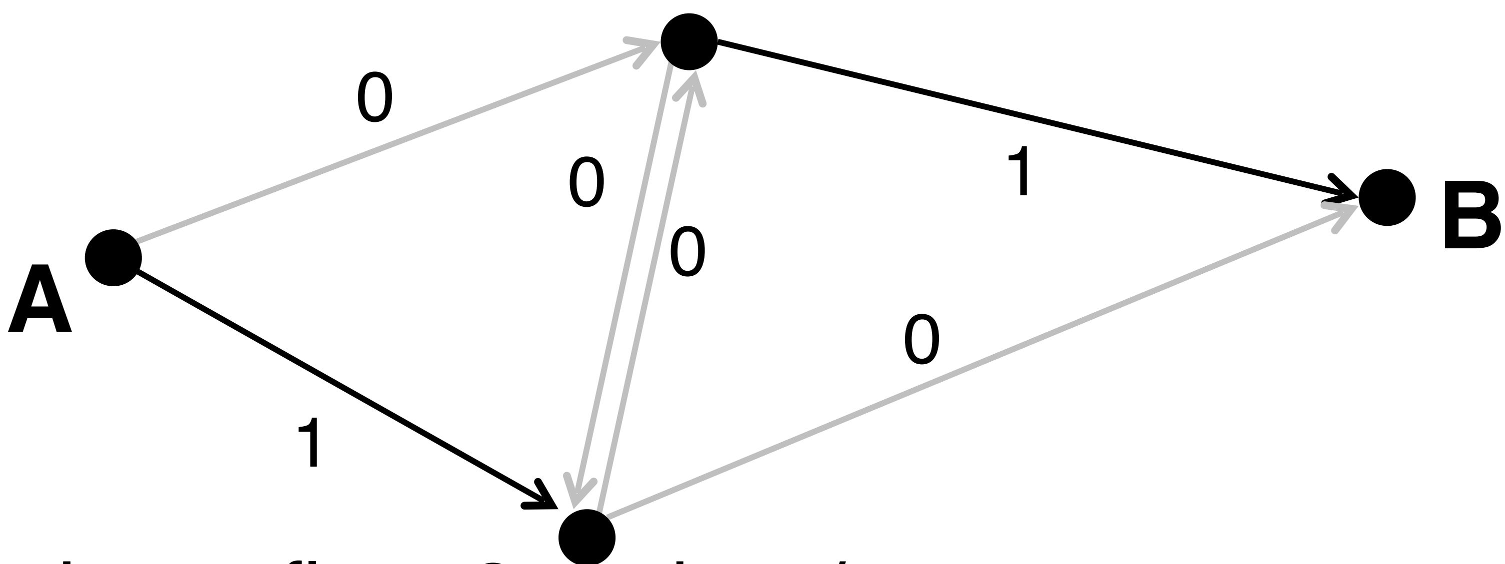
Greedy Algorithm?



Maximum flow: 3 packets/sec

Algorithm returns: 1 packet/sec

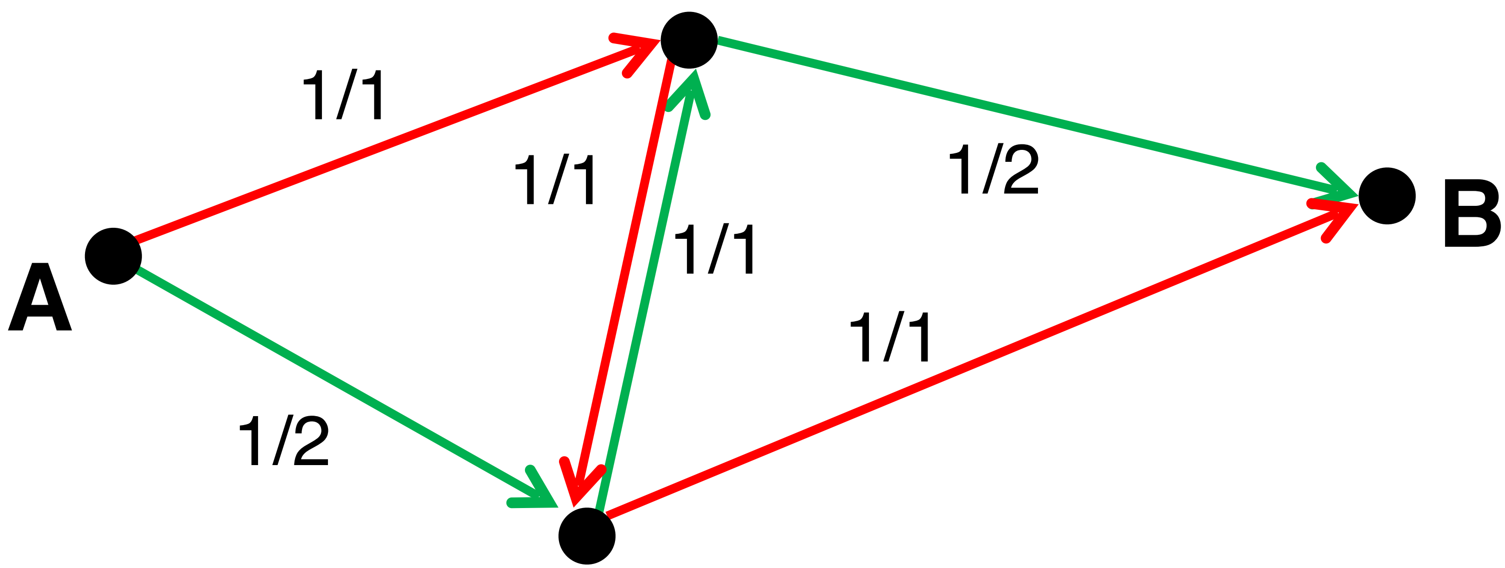
Greedy Algorithm?



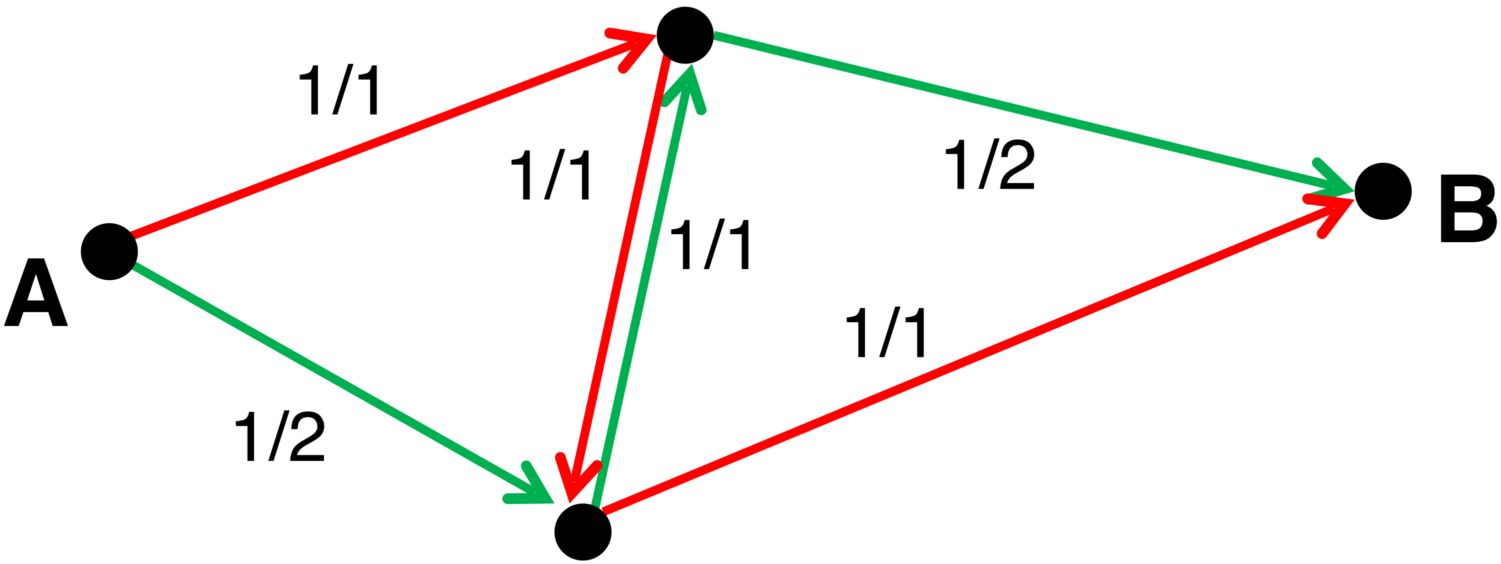
Maximum flow: 3 packets/sec

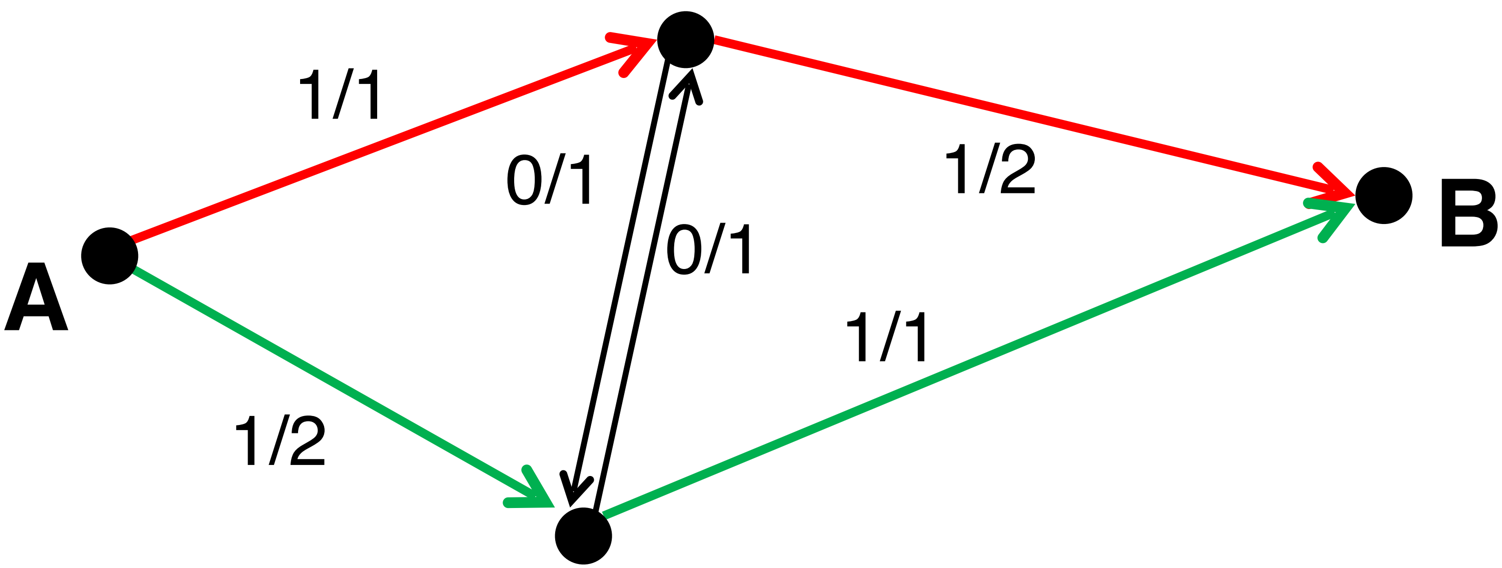
Algorithm returns: 2 packet/sec

Key Insight

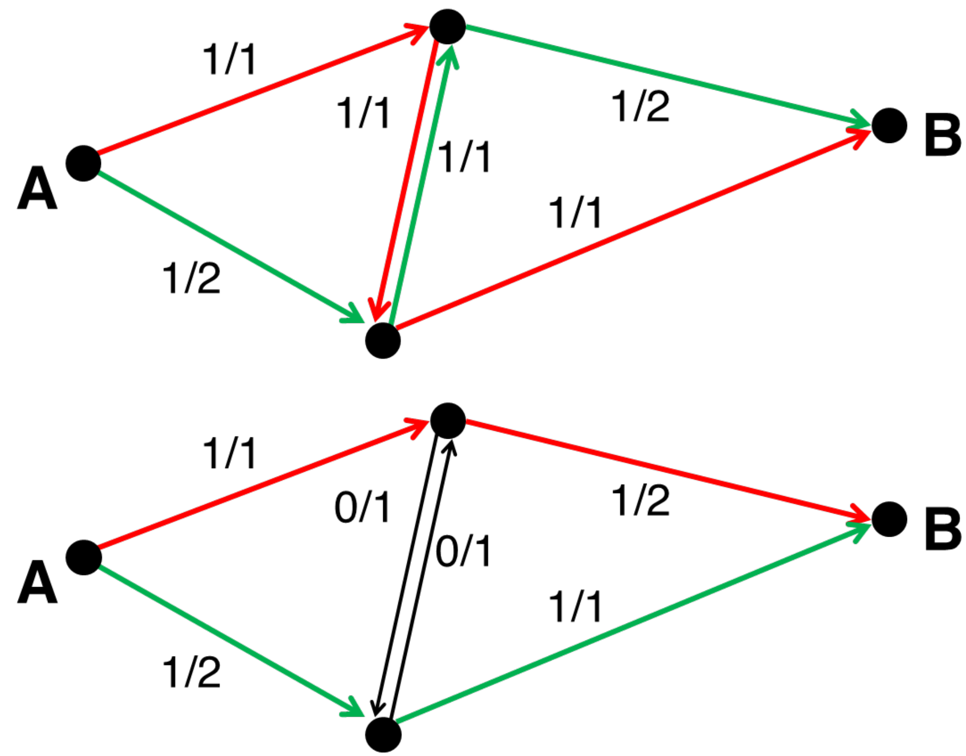


Key Insight





Key Insight

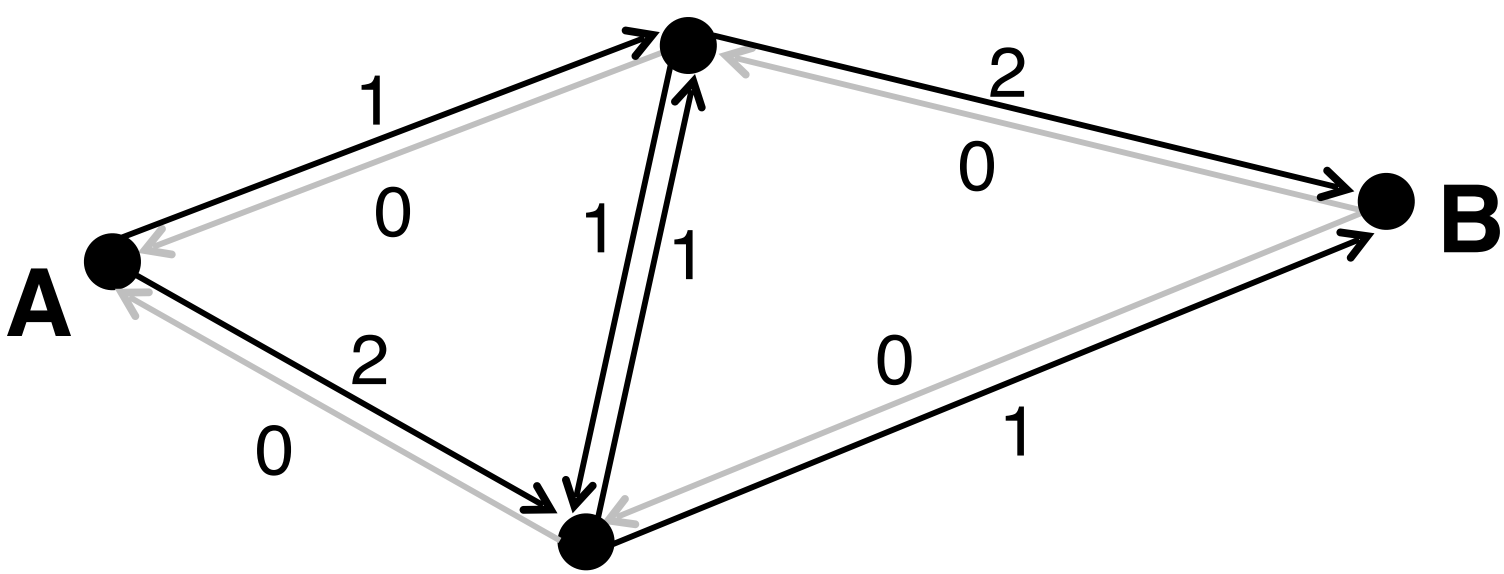


Every flow creates **virtual capacity** in the opposite direction

- increasing flow in opposite direction == decreasing flow in forward direction

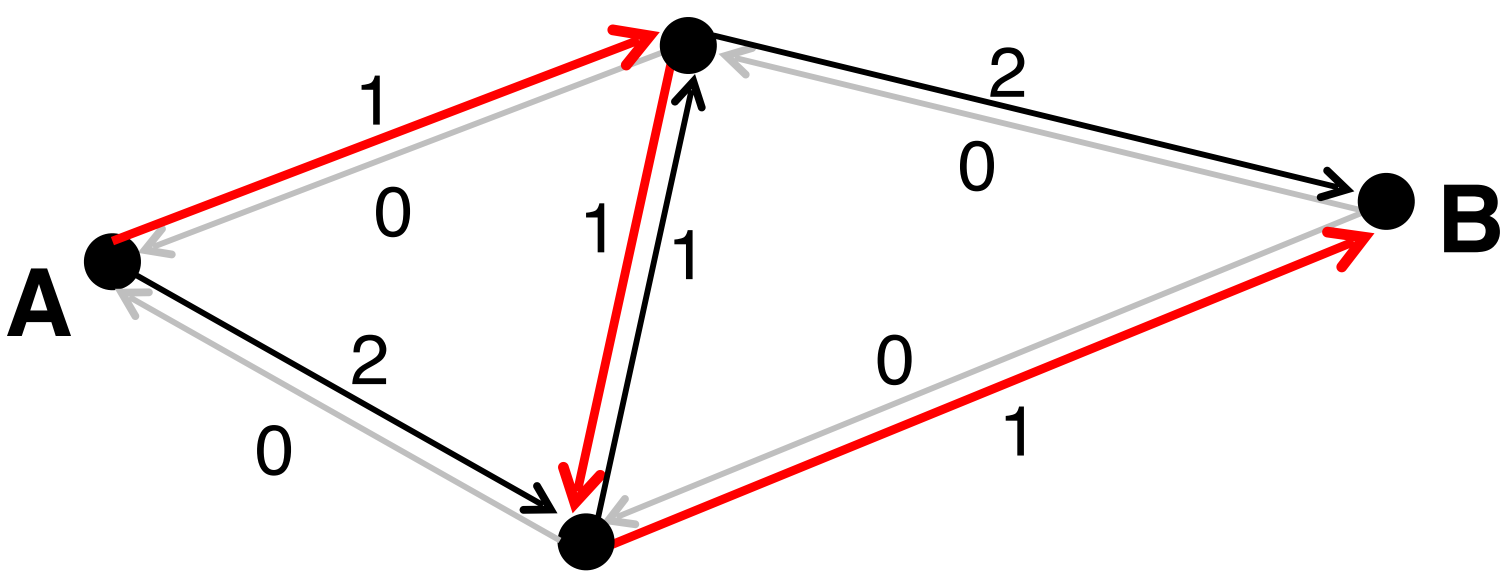
Ford–Fulkerson Algorithm

1. Find path from A to B (BFS)
2. Find bottleneck capacity
3. Add to total / subtract from capacities



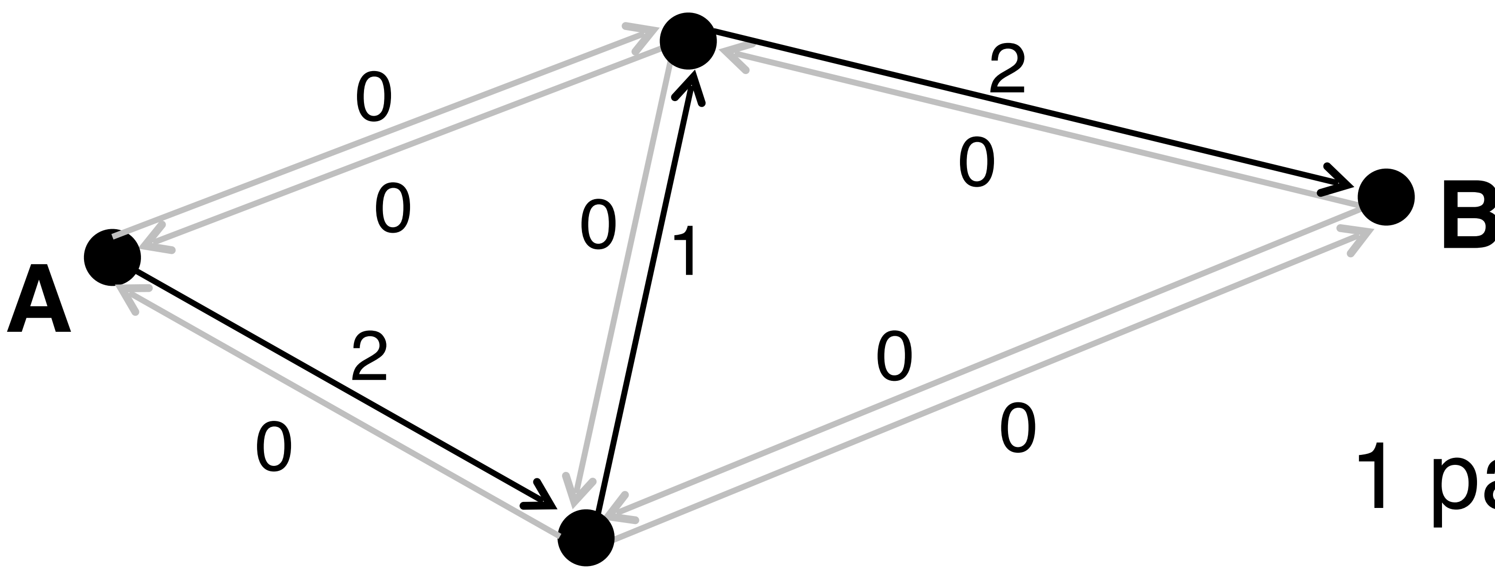
Ford–Fulkerson Algorithm

1. Find path from A to B (BFS)
2. Find bottleneck capacity
3. Add to total / subtract from capacities



Ford–Fulkerson Algorithm

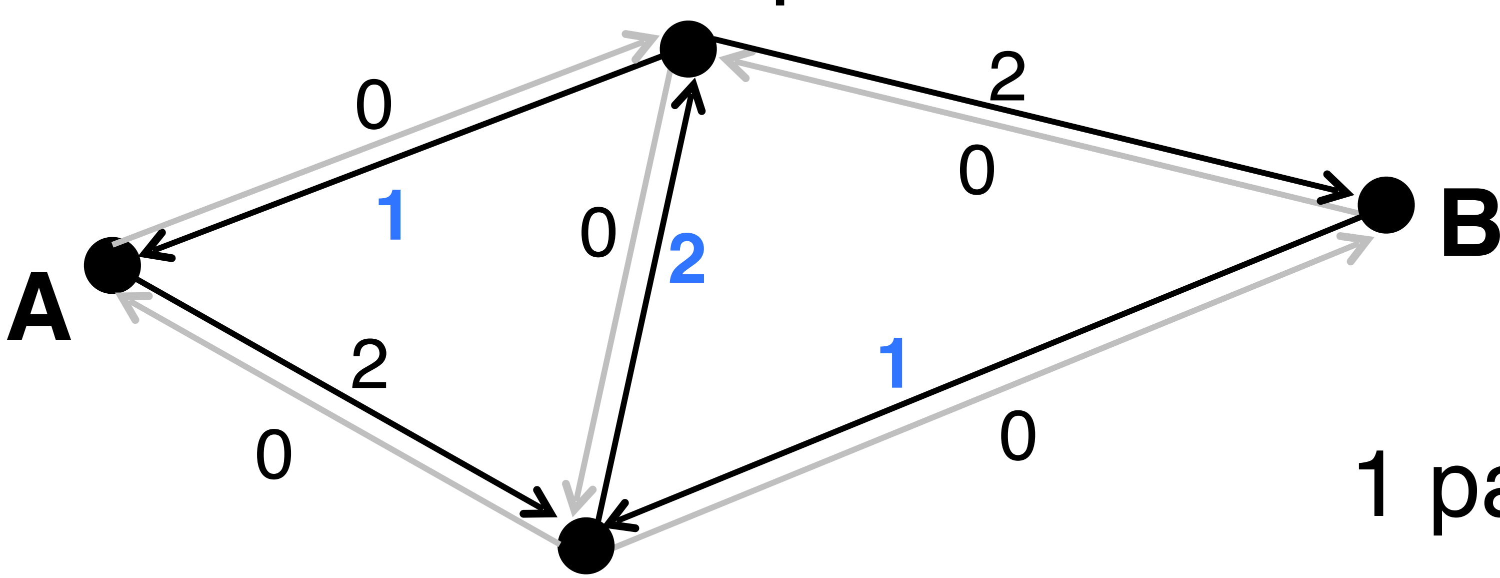
1. Find path from A to B (BFS)
2. Find bottleneck capacity
3. Add to total / subtract from capacities



1 packet/s

Ford–Fulkerson Algorithm

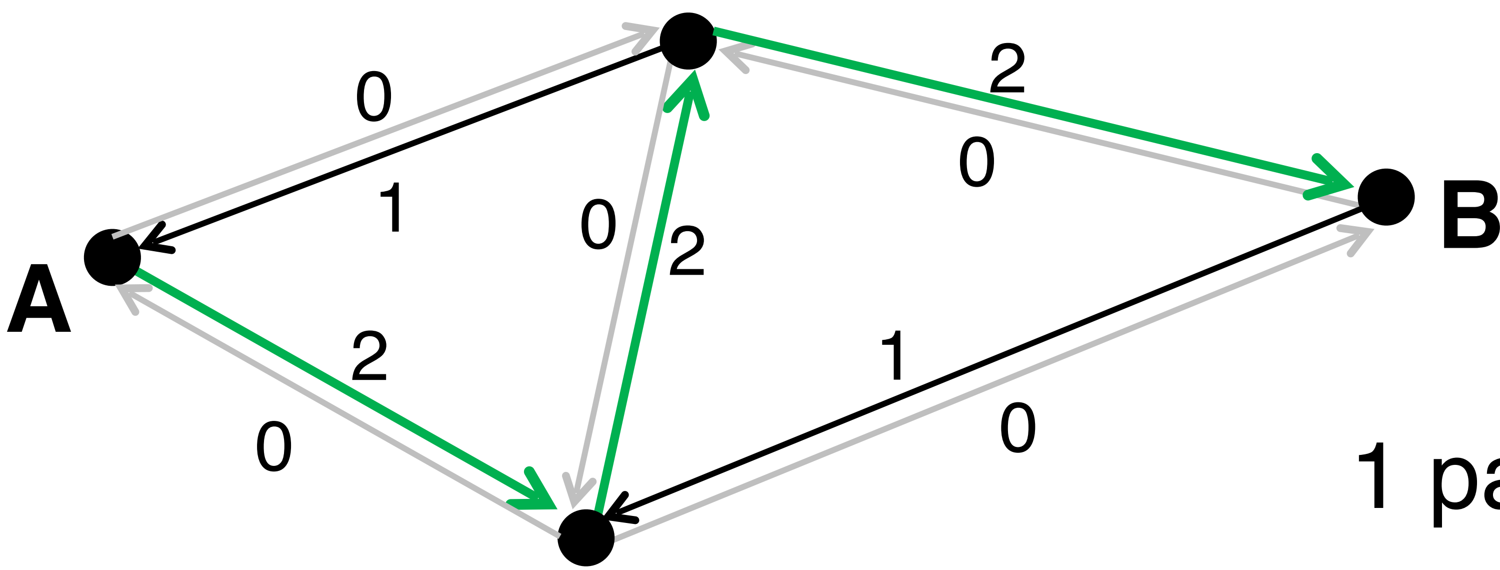
1. Find path from A to B (BFS)
2. Find bottleneck capacity
3. Add to total / subtract from capacities
4. Add to **reverse** capacities



1 packet/s

Ford–Fulkerson Algorithm

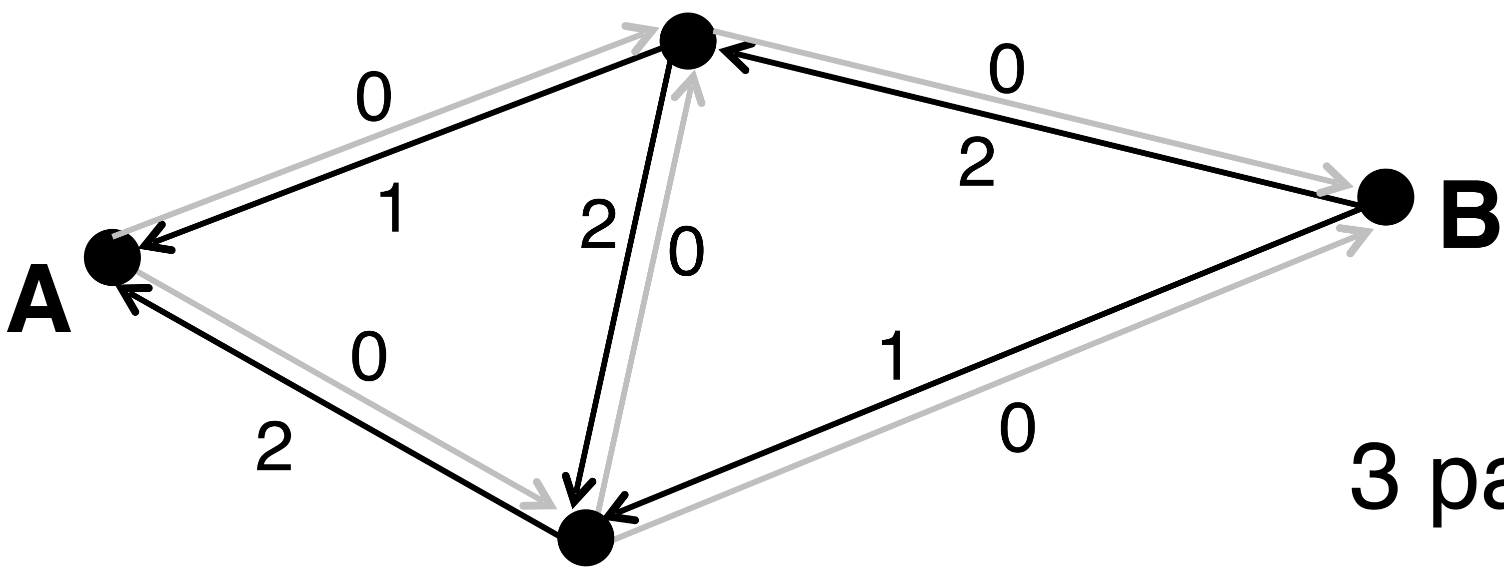
1. Find path from A to B (BFS)
2. Find bottleneck capacity
3. Add to total / subtract from capacities
4. Add to **reverse** capacities



1 packet/s

Ford–Fulkerson Algorithm

1. Find path from A to B (BFS)
2. Find bottleneck capacity
3. Add to total / subtract from capacities
4. Add to **reverse** capacities



3 packet/s

Ford–Fulkerson Algorithm

1. Find path from A to B (BFS)
2. Find bottleneck capacity
3. Add to total / subtract from capacities
4. Add to **reverse** capacities

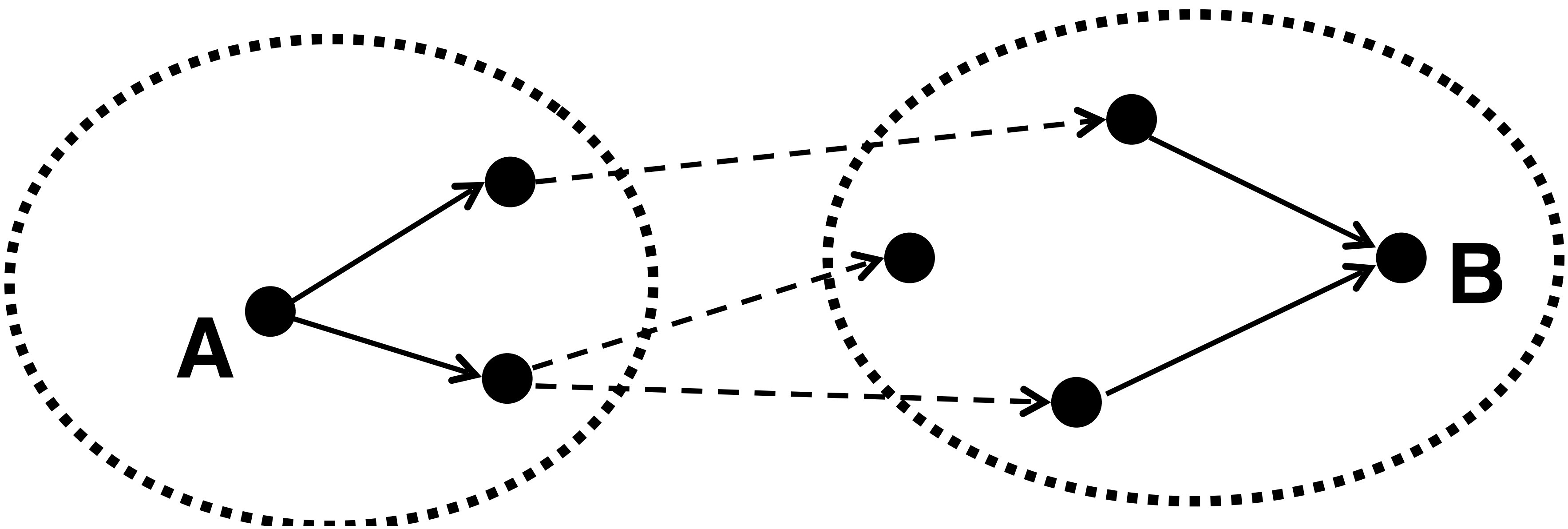
Every iteration increases total flow

- terminates for integer capacities

Graph Cut

A **graph cut** consists of:

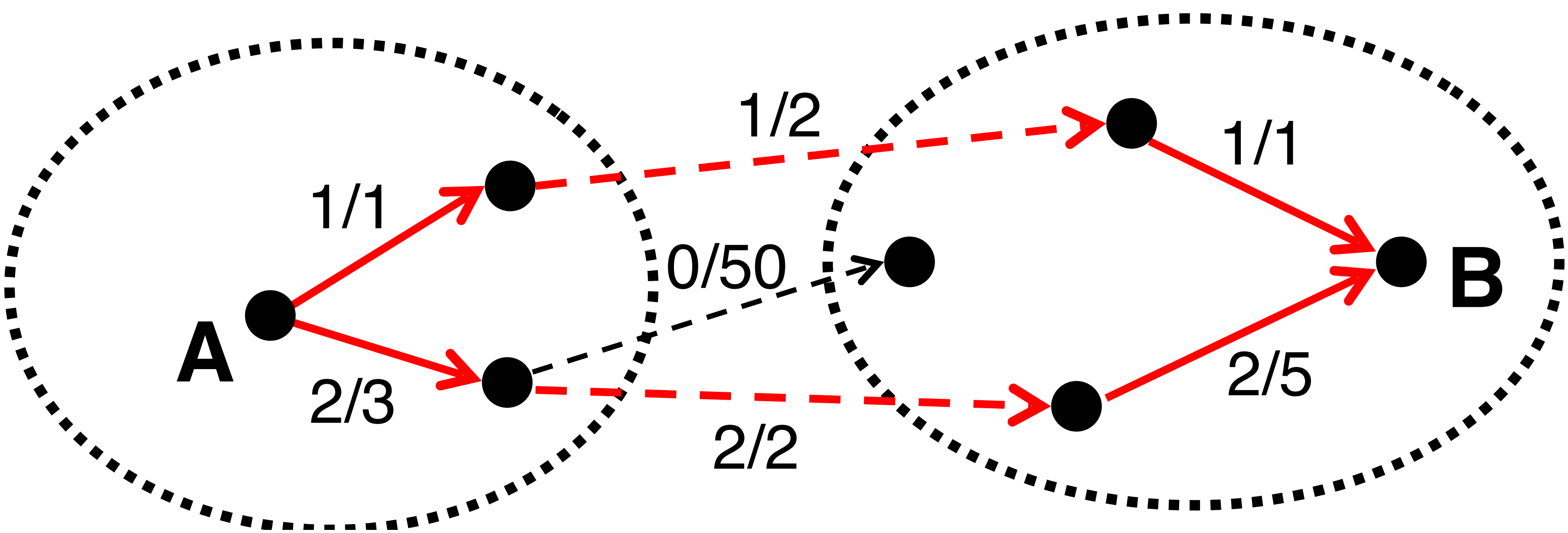
- group all vertices into two sets
- edges connecting the sets are the **cut**



Graph Cut

Given a network with maximum flow **N**,

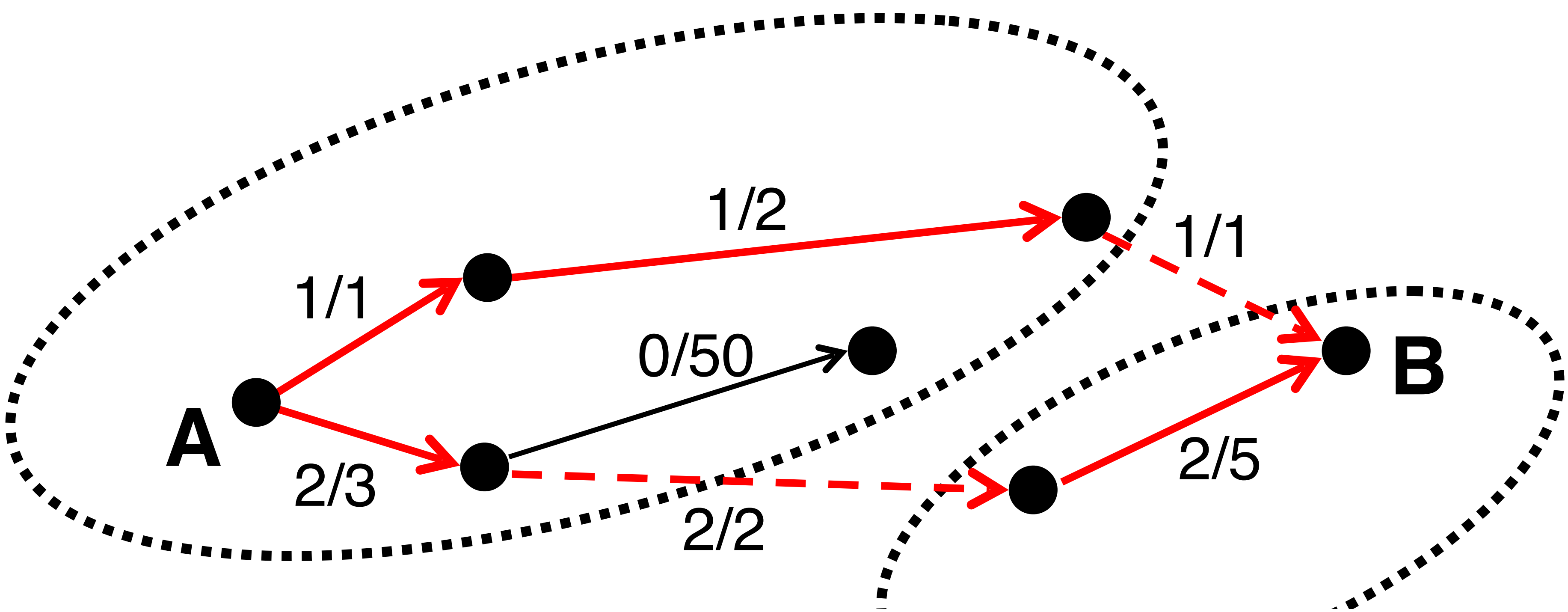
1. the flow across **every** cut is **N**



Graph Cut

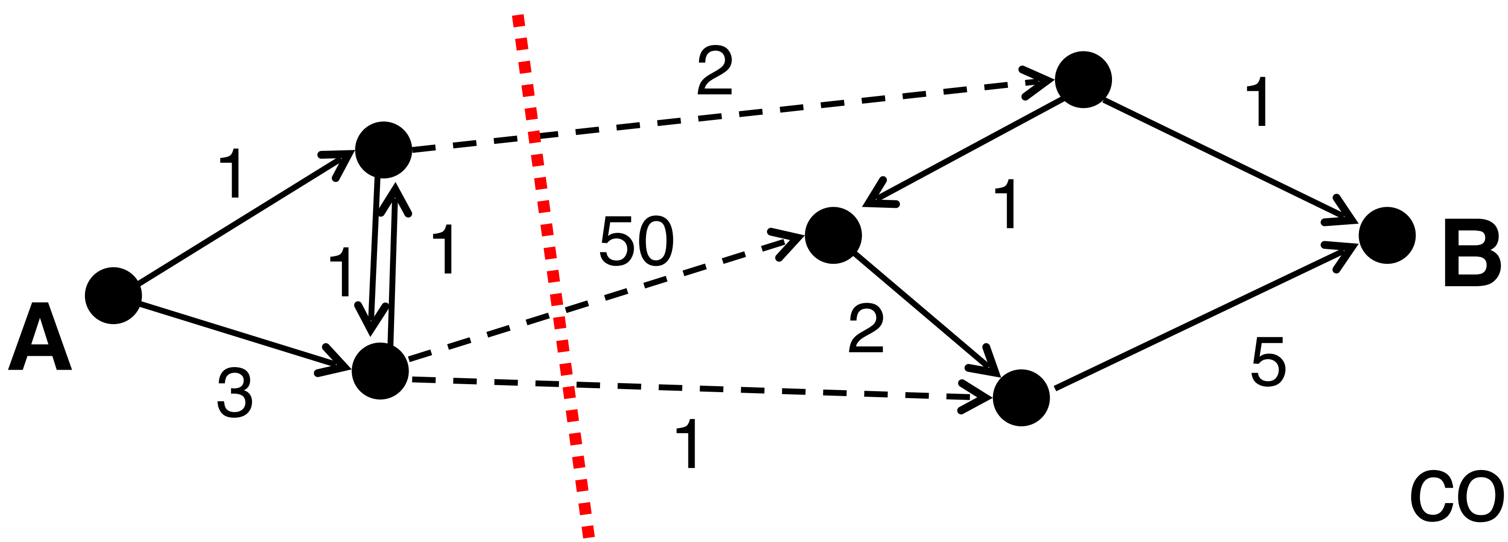
Given a network with maximum flow **N**,

1. the flow across **every** cut is **N**
2. for some cut, flow maxes capacity



Minimum Cut

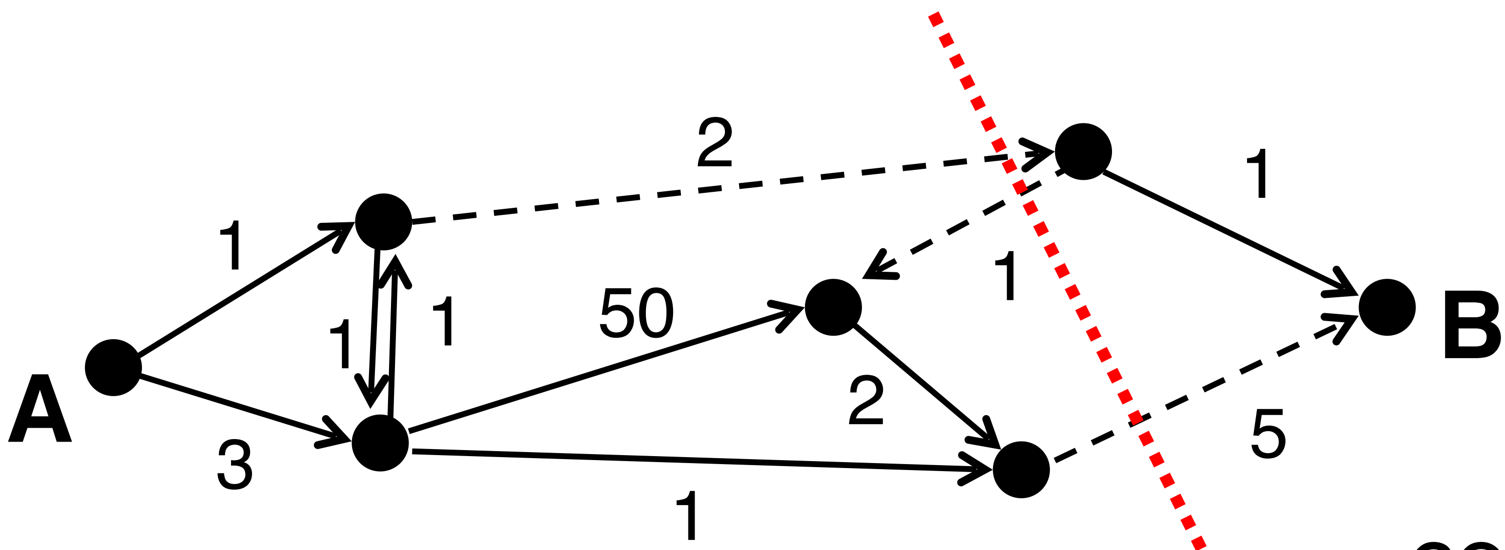
Given a weighted graph with source **A** and sink **B**, find cut that minimizes sum of weights of edges going from source side to sink side



cost: 53

Minimum Cut

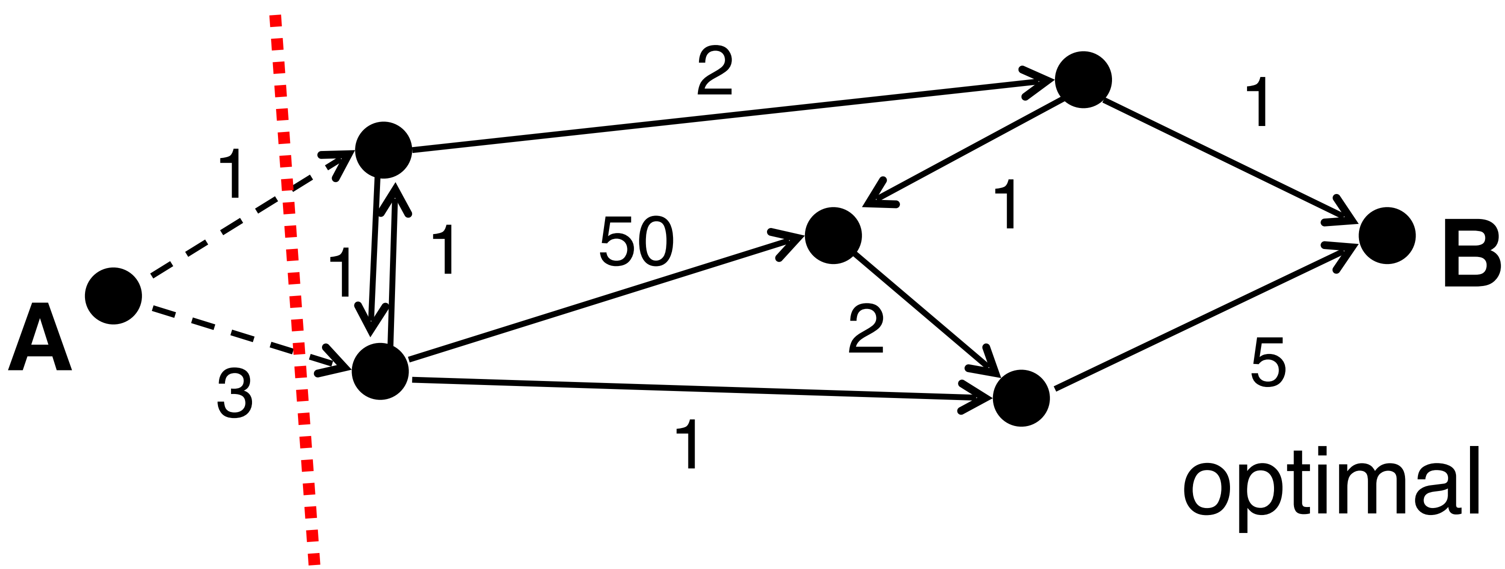
Given a weighted graph with source **A** and sink **B**, find cut that minimizes sum of weights of edges going from source side to sink side



cost: 7

Minimum Cut

Given a weighted graph with source **A** and sink **B**, find cut that minimizes sum of weights of edges going from source side to sink side



optimal cost: 4

Min Cut / Max Flow

From previous remarks, solving max flow is **same** as solving min cut

Many surprising applications

- seam carving
- bipartite matching (queens problem)