# Collaborative Delivery with Energy-Constrained Mobile Robots

**Andreas Bärtschi[1], Jérémie Chalopin[2], Shantanu Das[2], Yann Disser[3]**
**Barbara Geissmann[1], Daniel Graf[1], Arnaud Labourel[2], Matúš Mihalák[4]**

1 ETH Zürich
2 LIF, CNRS et Aix-Marseille Université
3 Technische Universität Berlin
4 Maastricht University

*23rd International Colloquium on Structural Information and Communication Complexity*
*19–21 July 2016 · Helsinki, Finland*

# Collaborative delivery with mobile agents

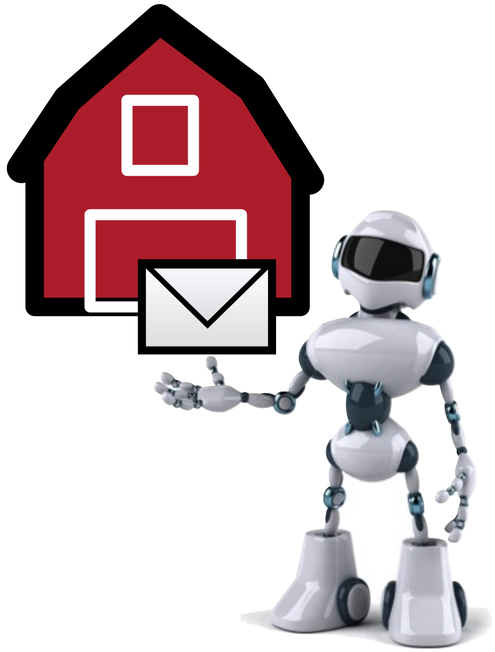


Image Sources: *BLADE Nano QX RTF Quadcopter*, http://paralebin.hol.es/robot/, http://www.generationrobots.com

# Collaborative delivery with mobile agents

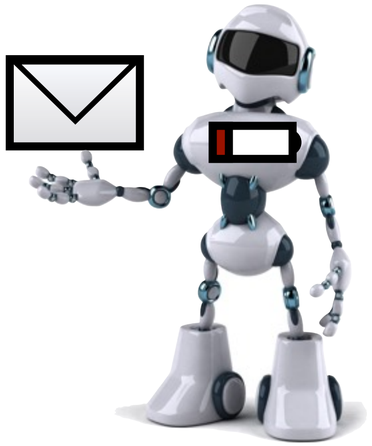


Image Sources: *BLADE Nano QX RTF Quadcopter*, http://paralebin.hol.es/robot/, http://www.generationrobots.com

# Collaborative delivery with mobile agents

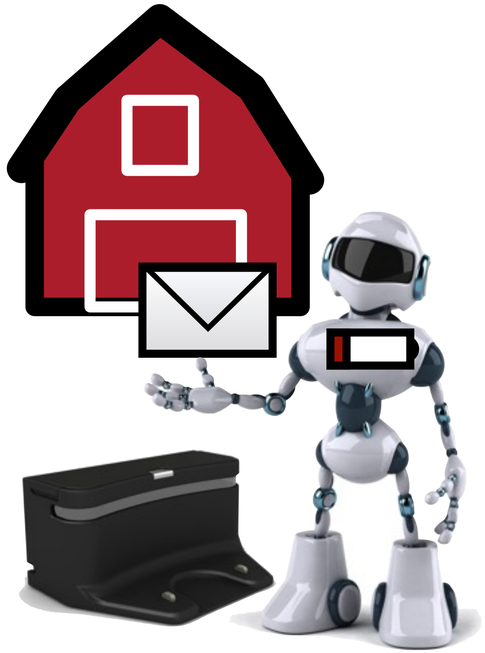
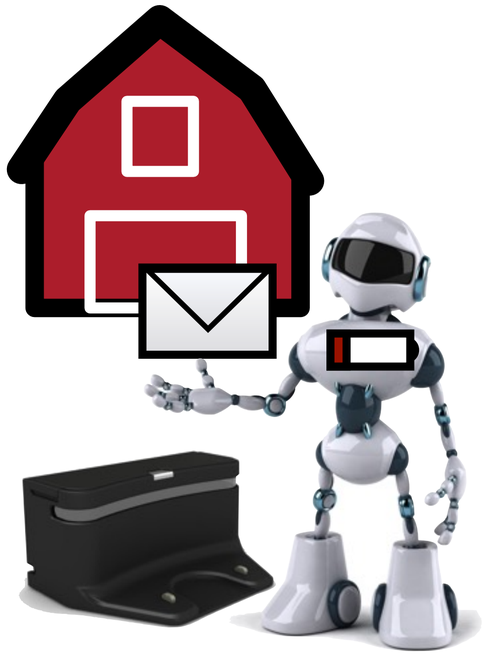


Image Sources: *BLADE Nano QX RTF Quadcopter*, http://paralebin.hol.es/robot/, http://www.generationrobots.com

# Collaborative delivery with mobile agents



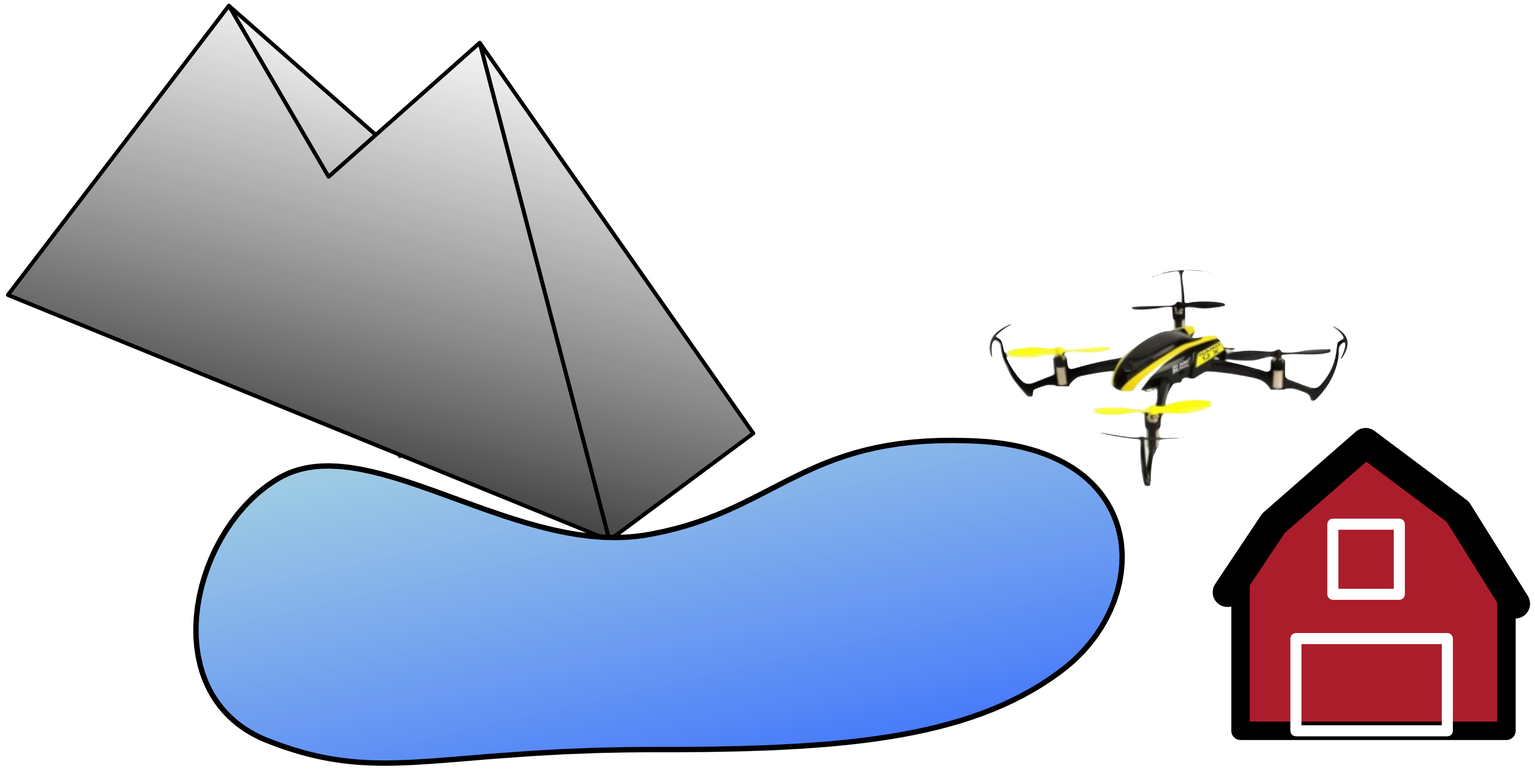


Image Sources: *BLADE Nano QX RTF Quadcopter*, http://paralebin.hol.es/robot/, http://www.generationrobots.com

# Collaborative delivery with mobile agents

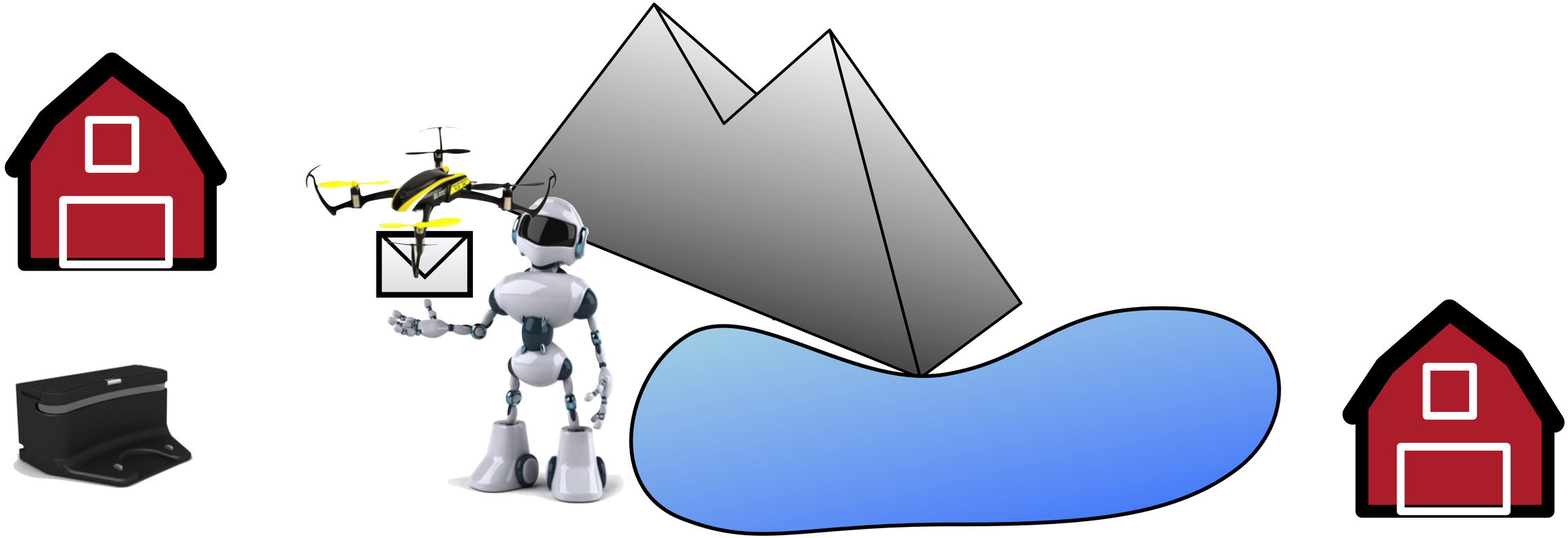


Image Sources: *BLADE Nano QX RTF Quadcopter*, http://paralebin.hol.es/robot/, http://www.generationrobots.com

# Collaborative delivery with mobile agents

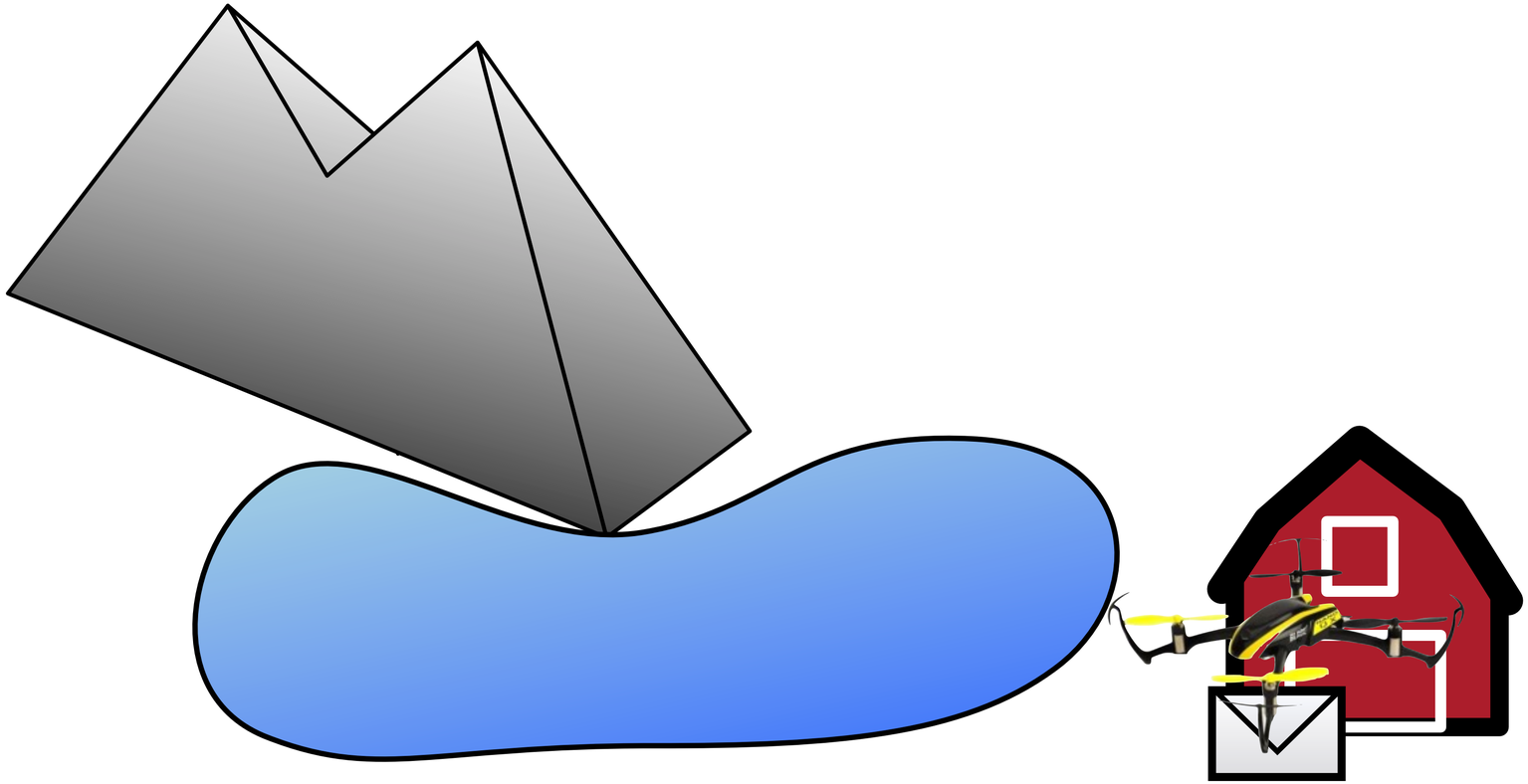


Image Sources: *BLADE Nano QX RTF Quadcopter*, http://paralebin.hol.es/robot/, http://www.generationrobots.com

## Setting

- undirected, edge-weighted graph with $n=|V|$, $m=|E|$
- single message from $s$ to $t$
- anyone can use any edge

## Assumptions

- global coordination
- handovers anywhere

## Agents

- $k$ agents each with
  - starting vertex $p_i$
  - battery capacity $B_i$
- agents have to return

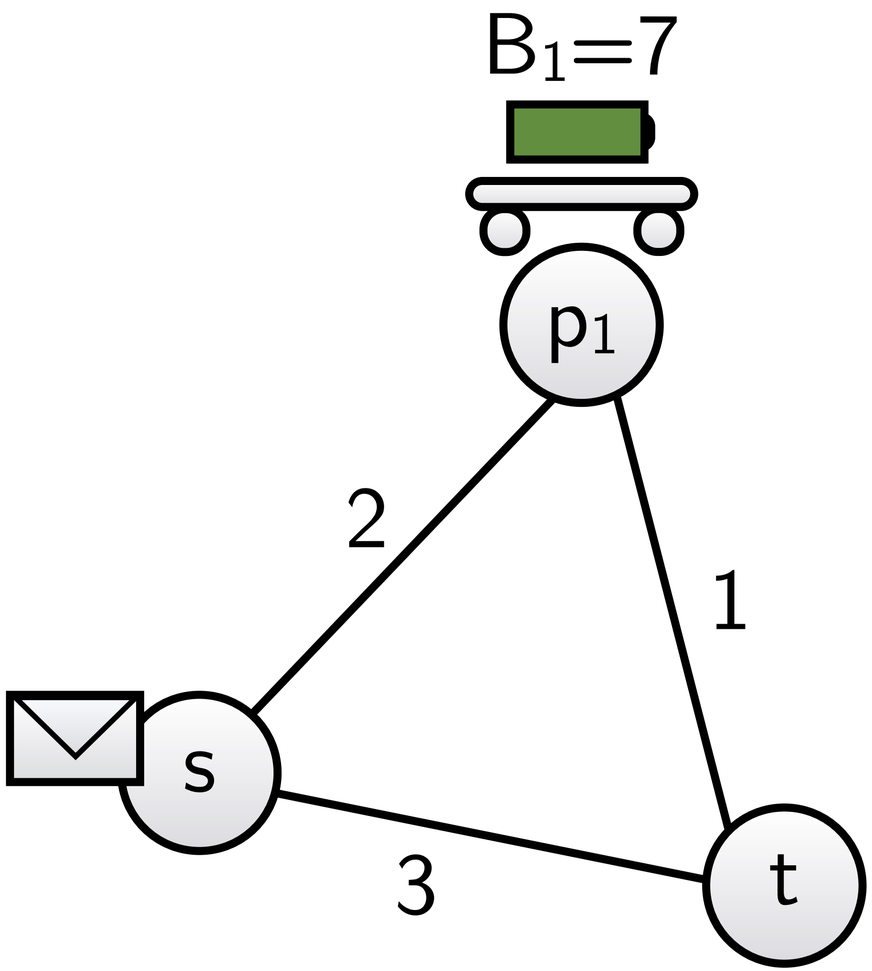
## Task

Decide feasibility!

If yes, compute a schedule.

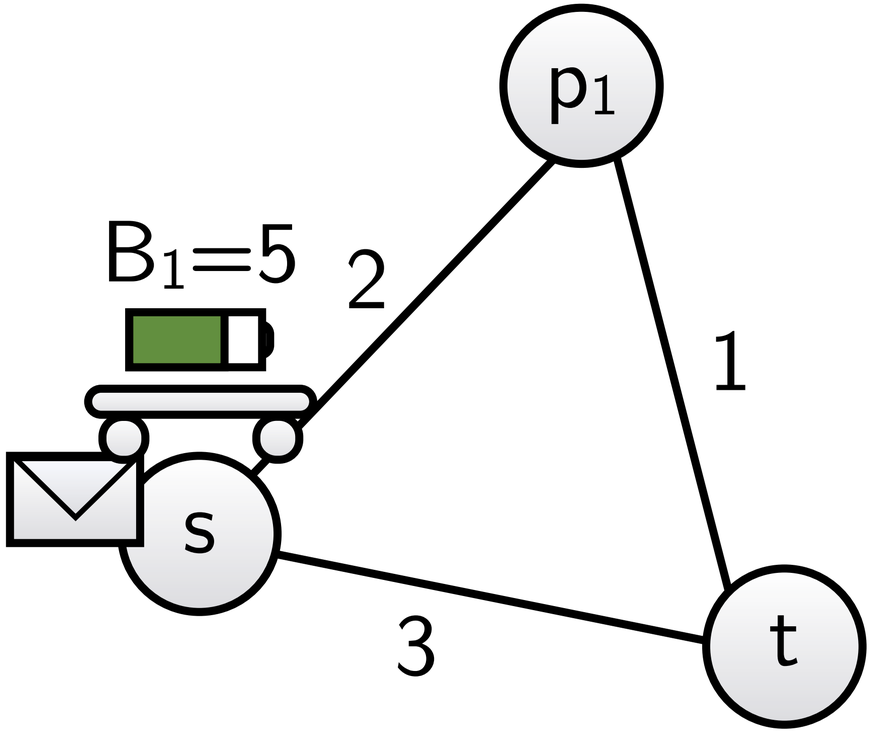
# Example





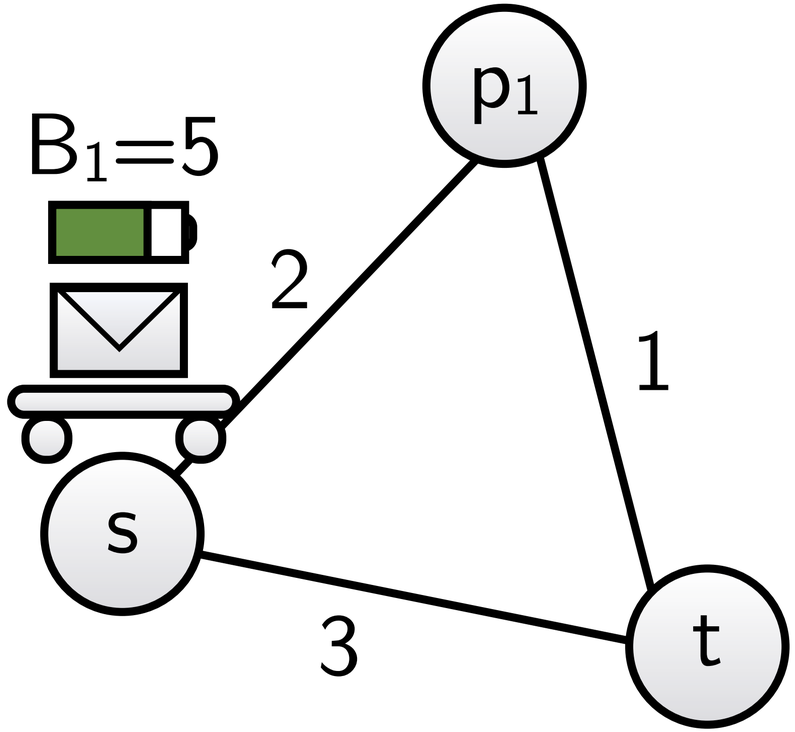
# Example





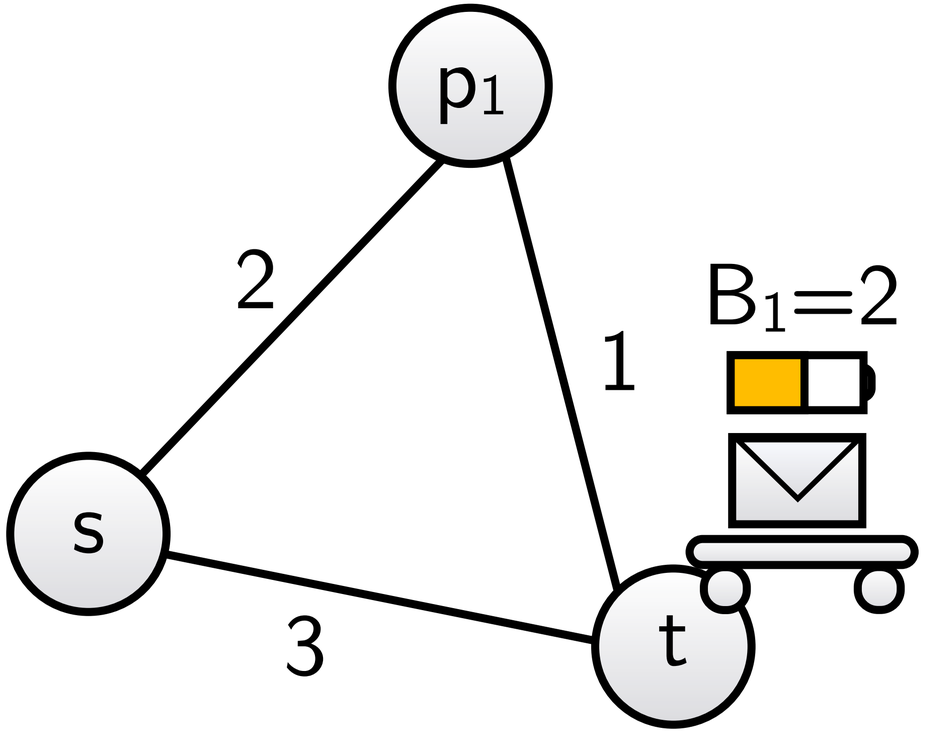
# Example





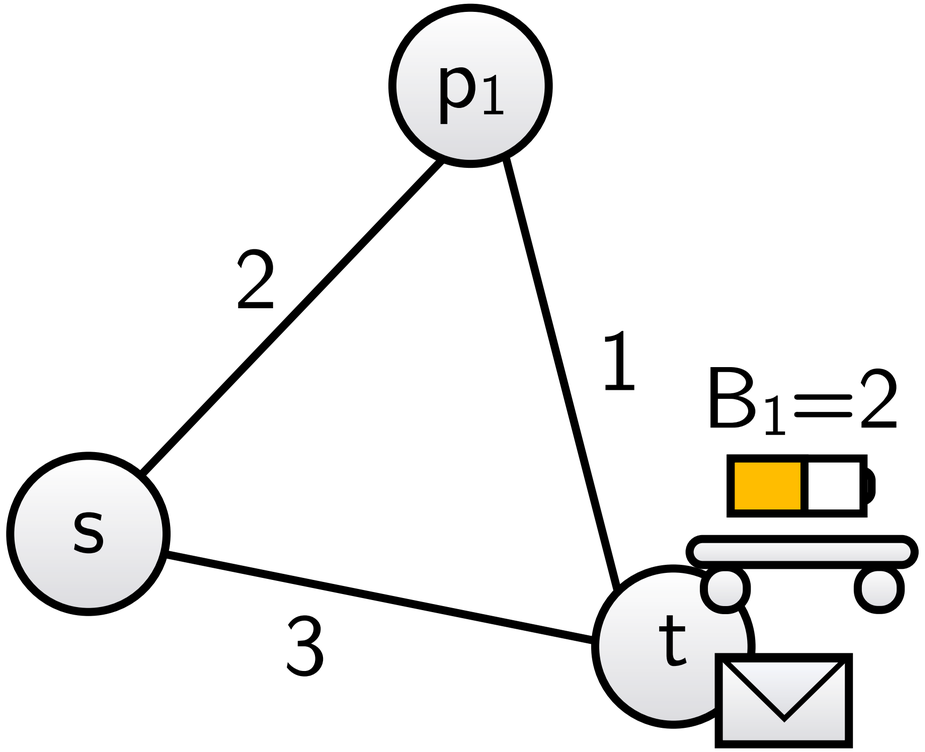
# Example





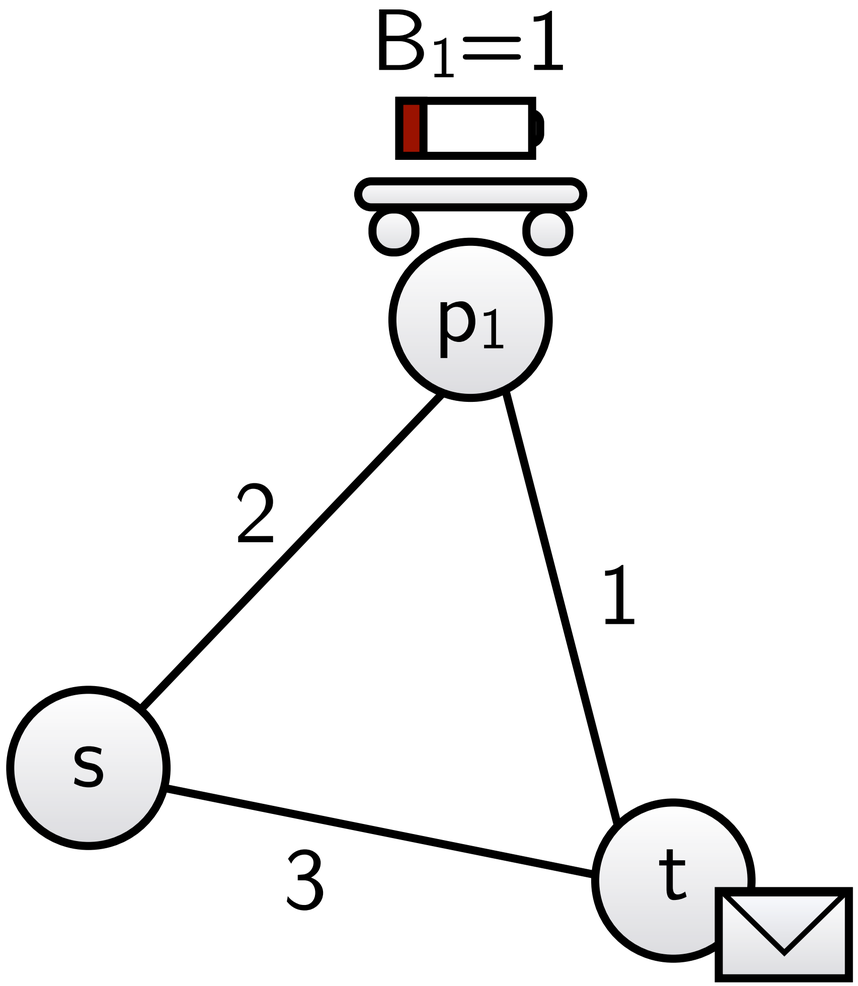
# Example





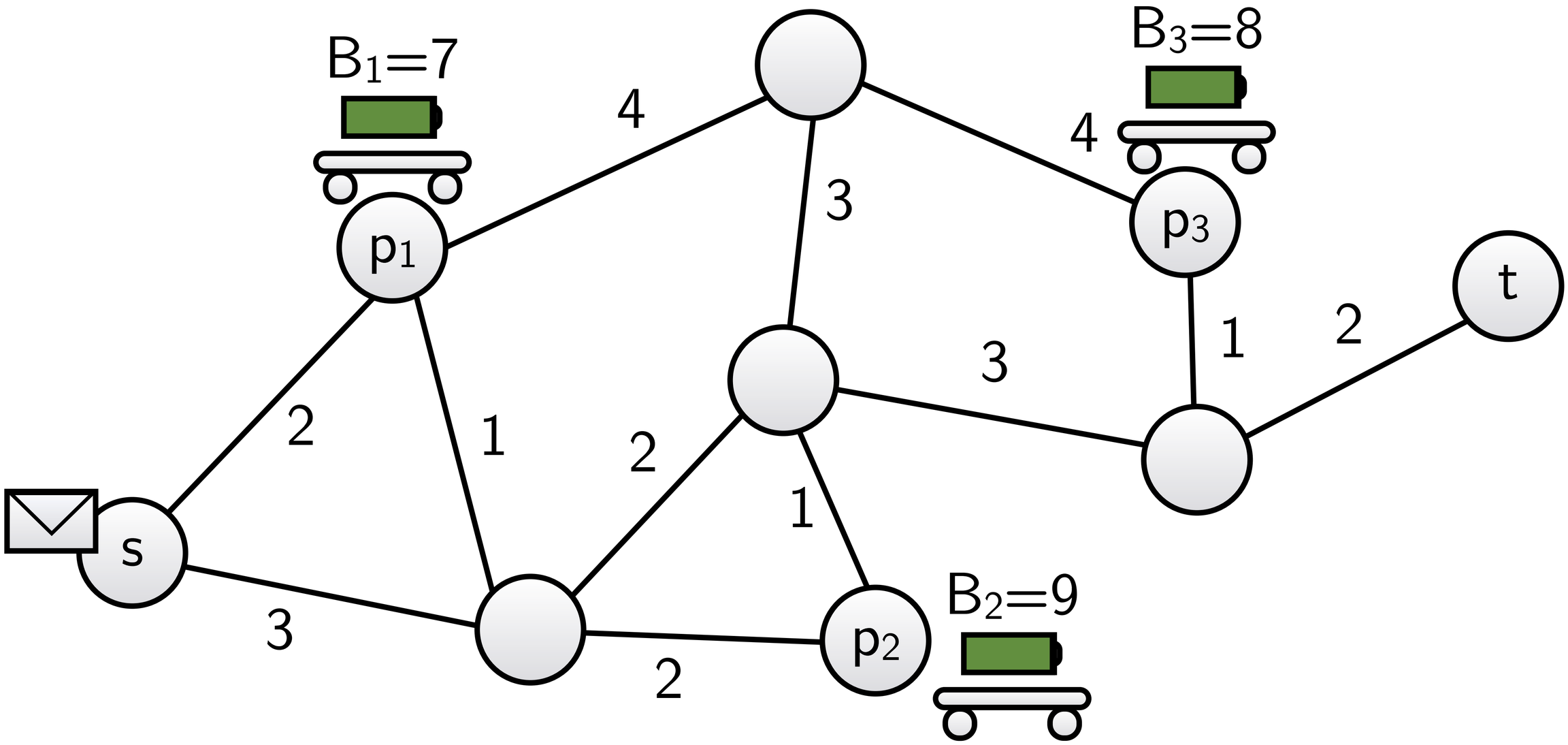
# Example





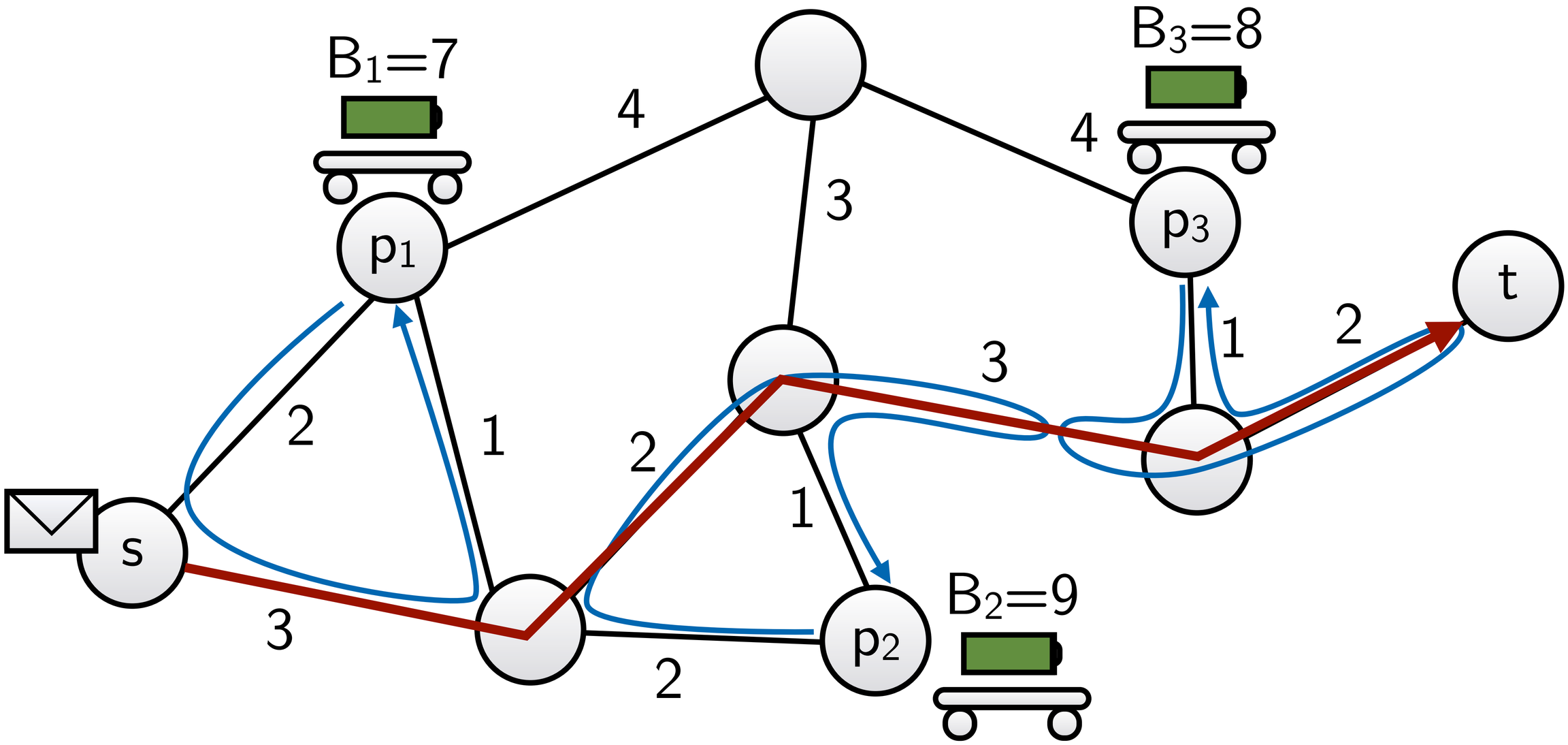
# Example





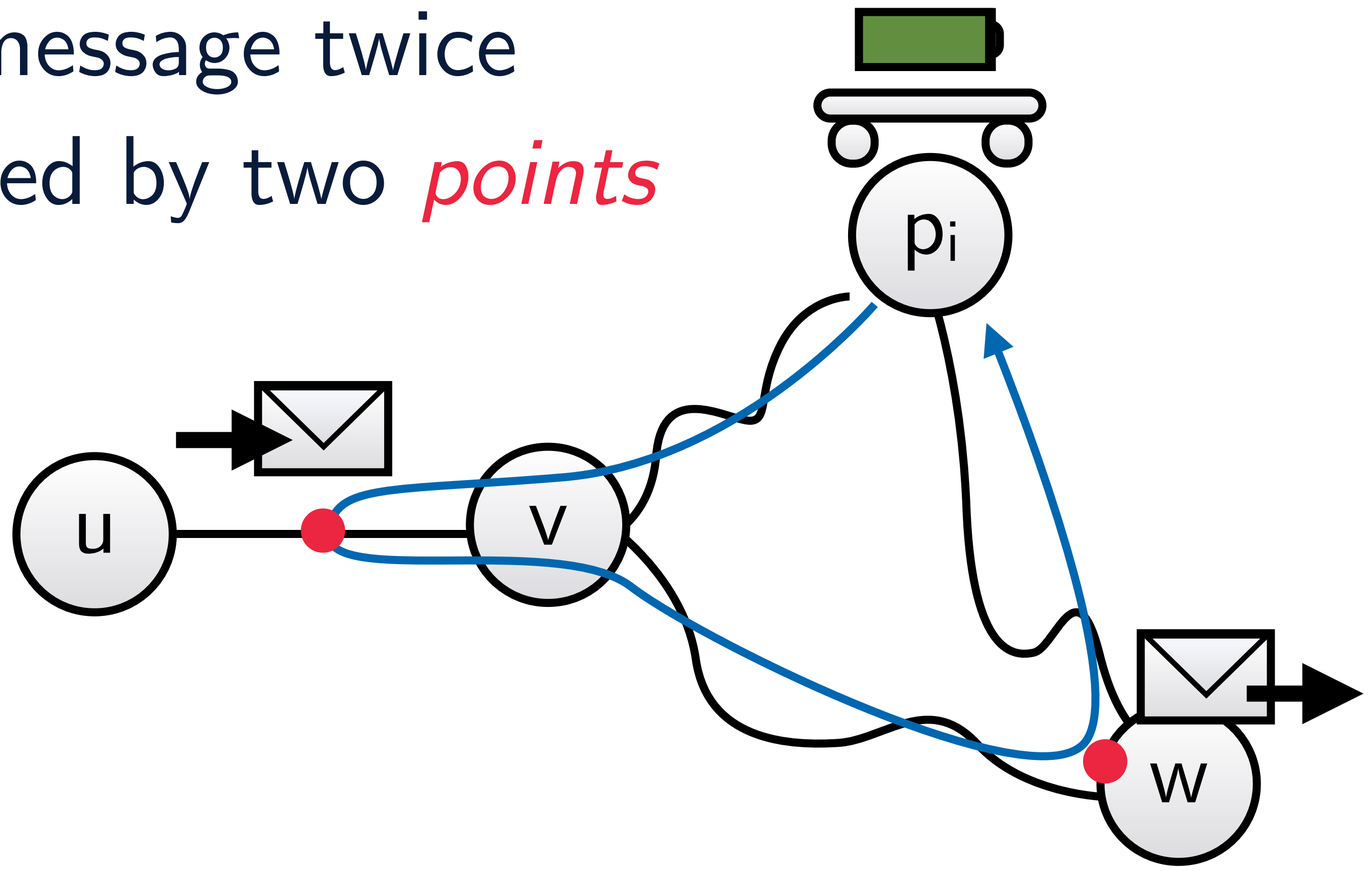
# Example





## Observations

- No agent picks up the message twice
- Path of an agent specified by two *points*

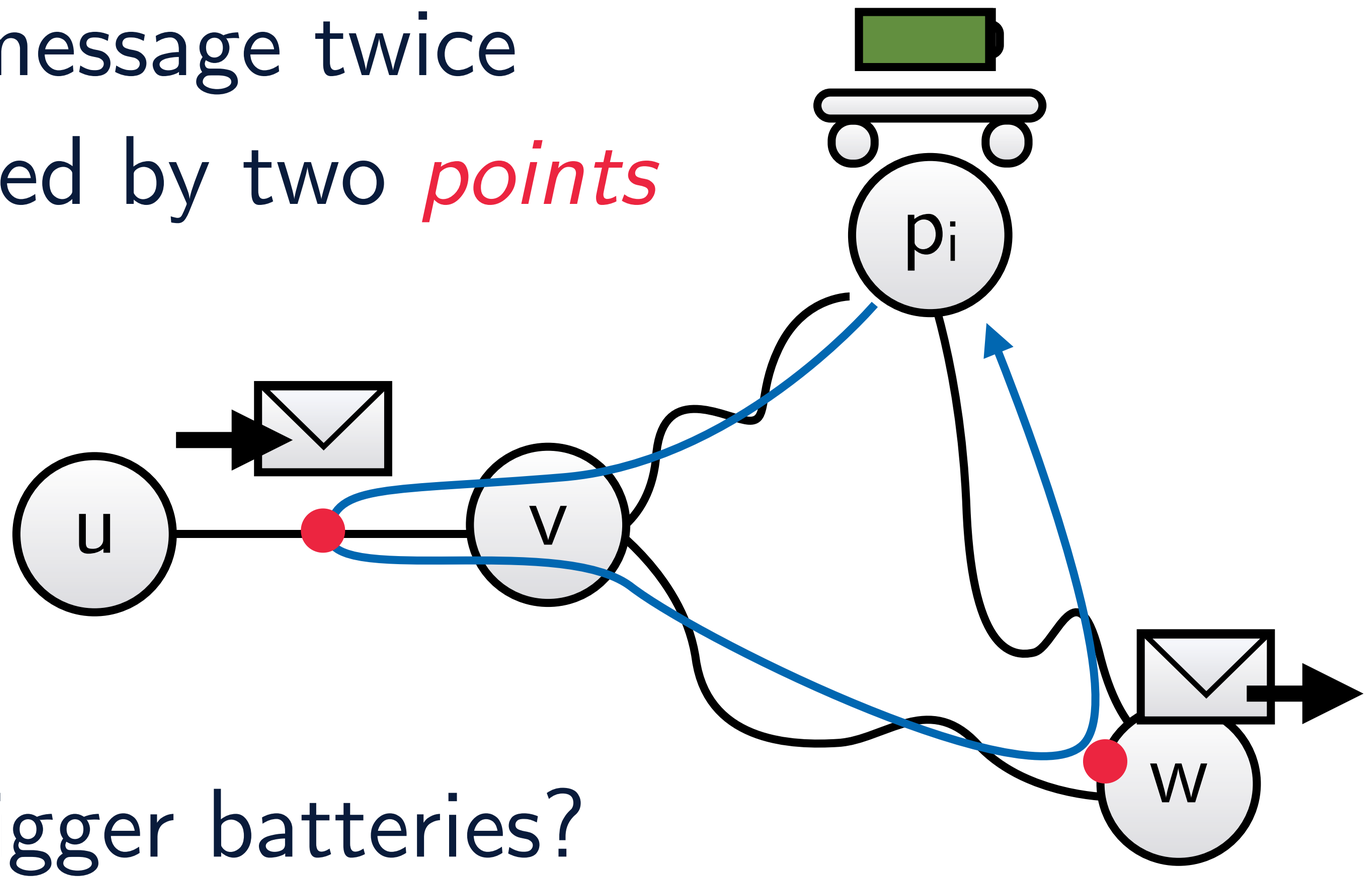


## Observations

- No agent picks up the message twice
- Path of an agent specified by two *points*

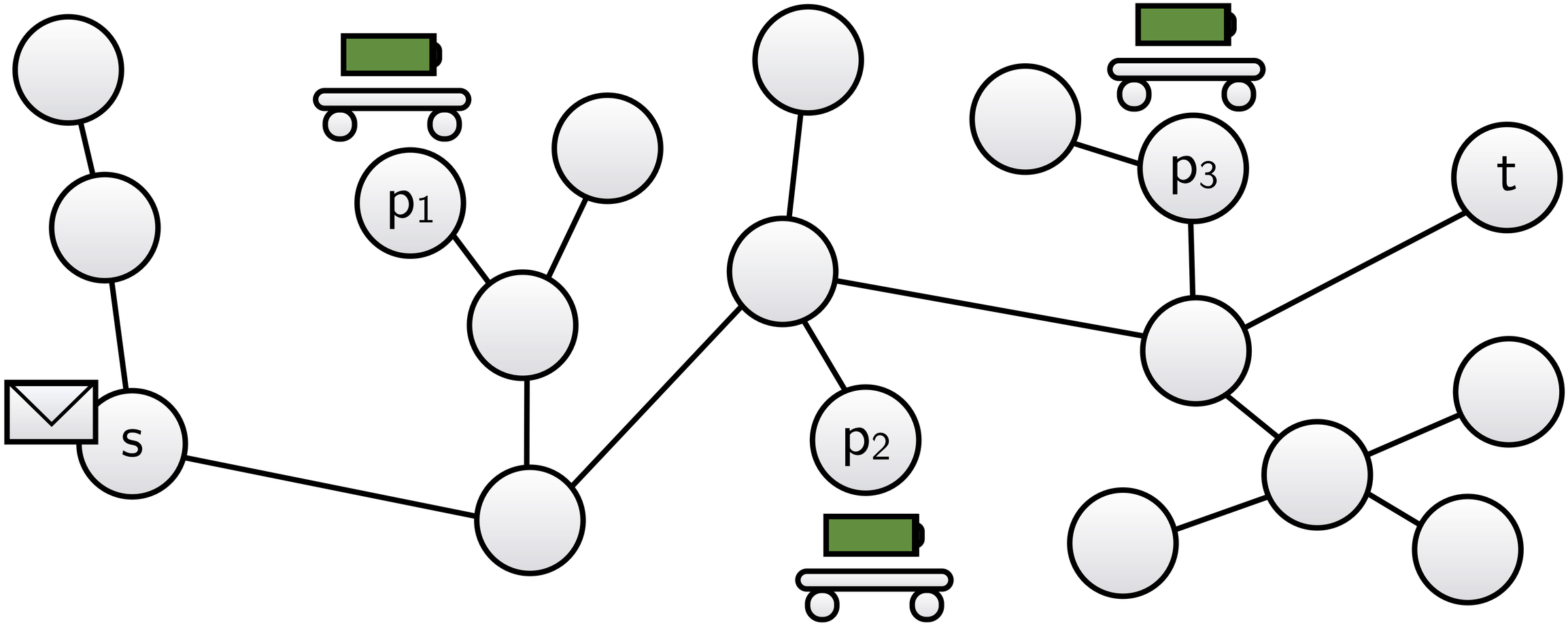
## Overview of the talk

- For trees?
- For all graphs?
- What can we do with bigger batteries?

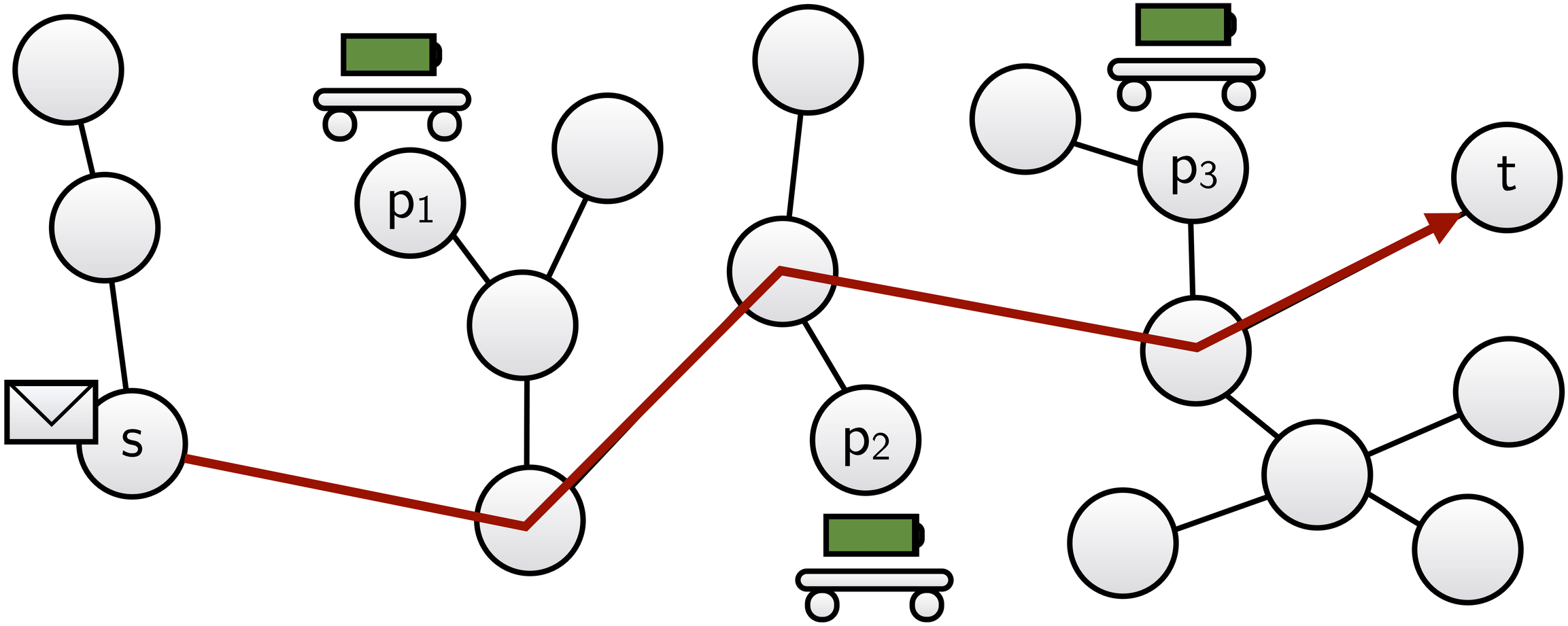


# Reduction to Paths

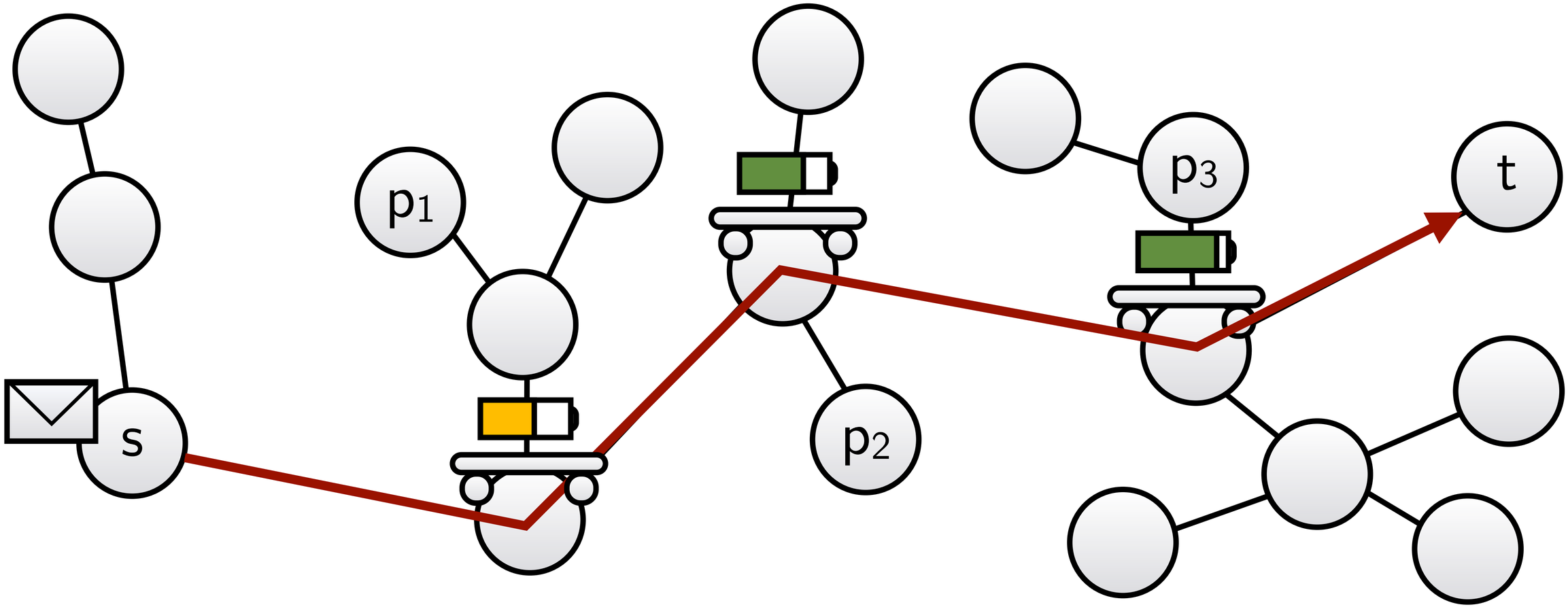




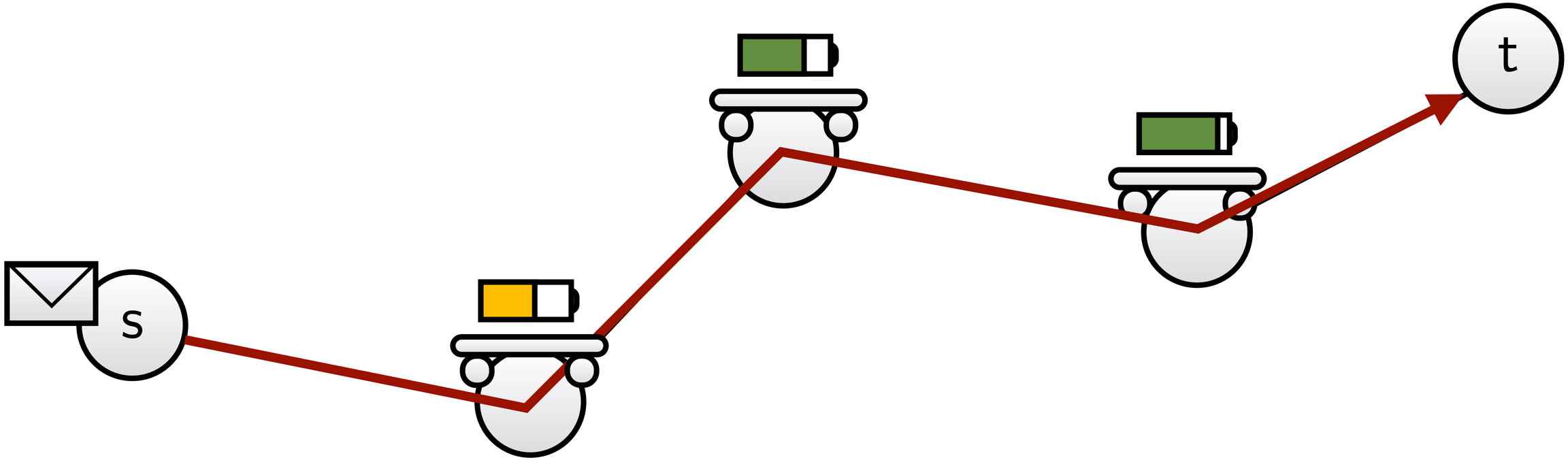
Move the agents to the $s$-$t$-path and adapt the budgets.



Move the agents to the $s$-$t$-path and adapt the budgets.



Move the agents to the $s$-$t$-path and adapt the budgets.



## Solving the Path Case

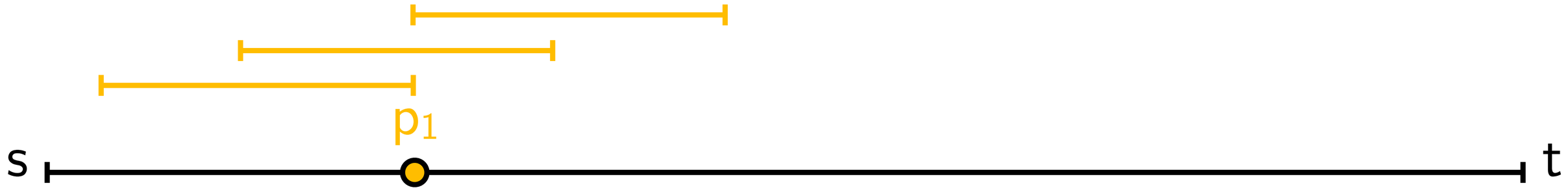
interval covering problem

- length $B_i/2$ containing $p_i$

## Solving the Path Case

interval covering problem

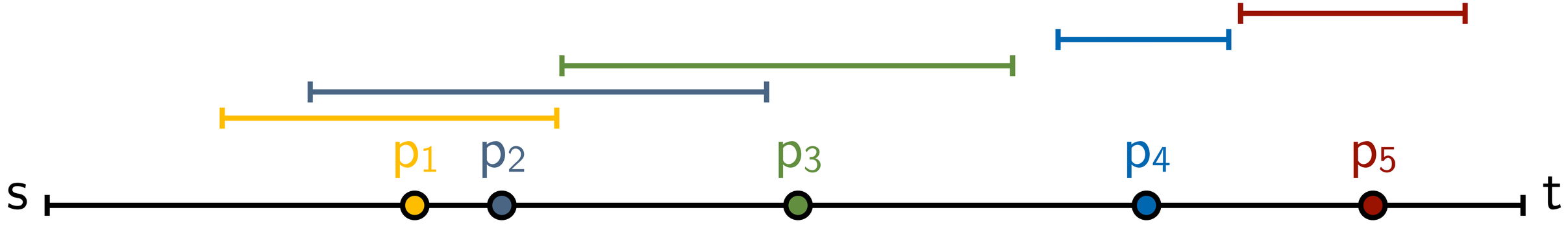
- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$



## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$



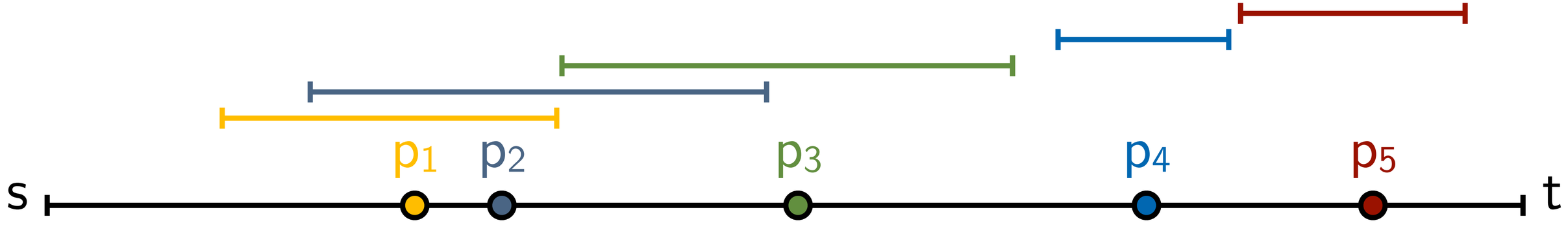
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



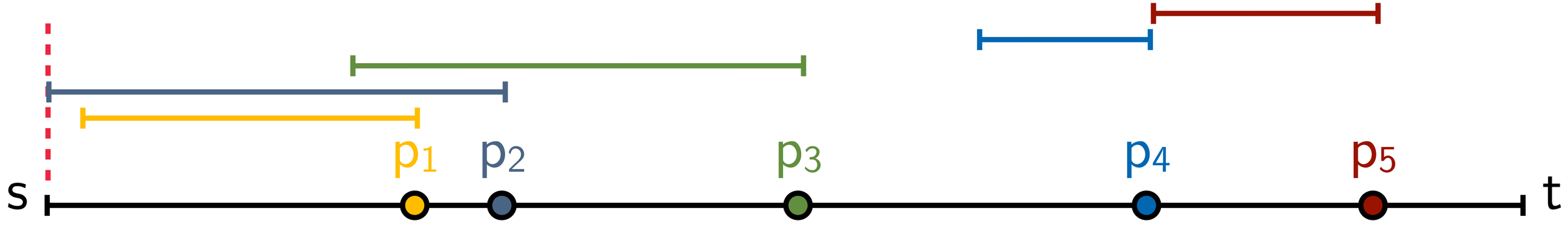
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



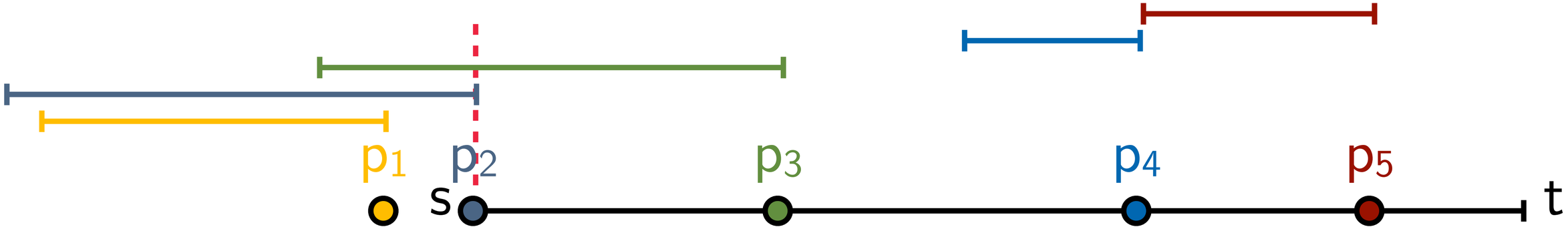
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



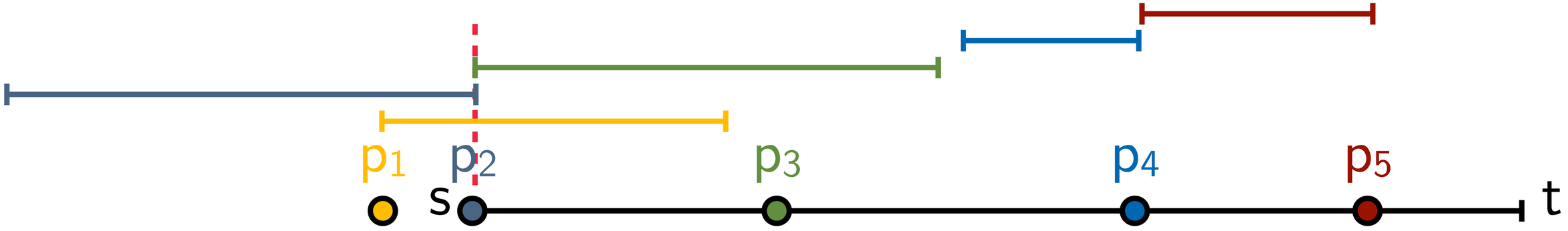
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



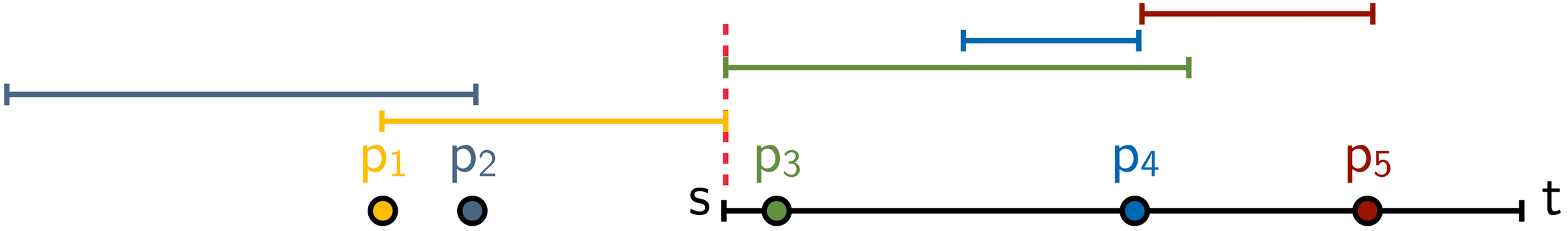
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



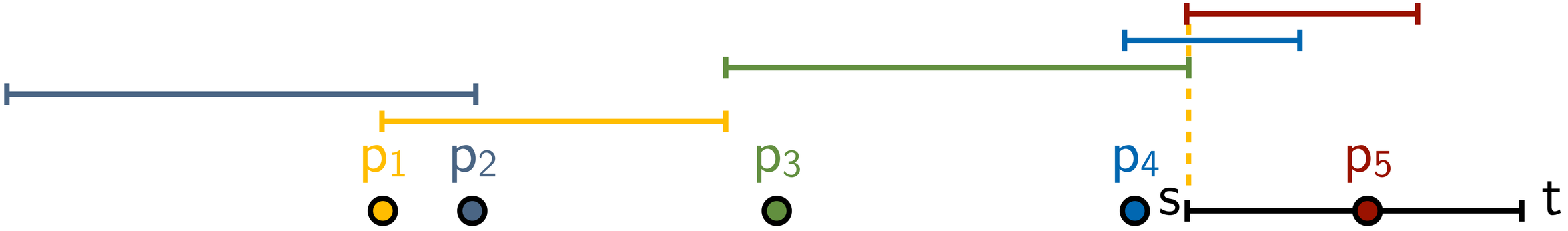
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



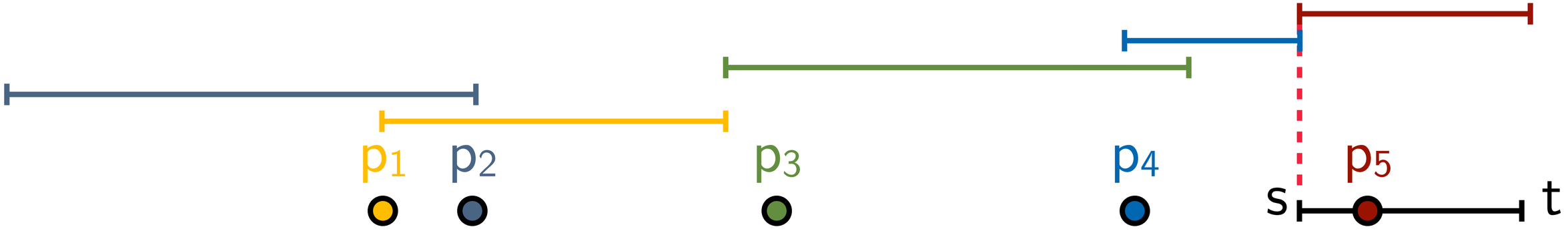
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



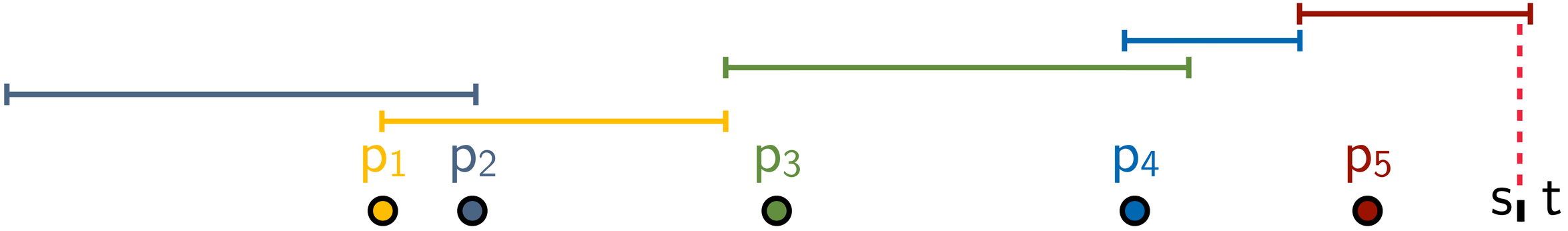
## Solving the Path Case

interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



## Solving the Path Case

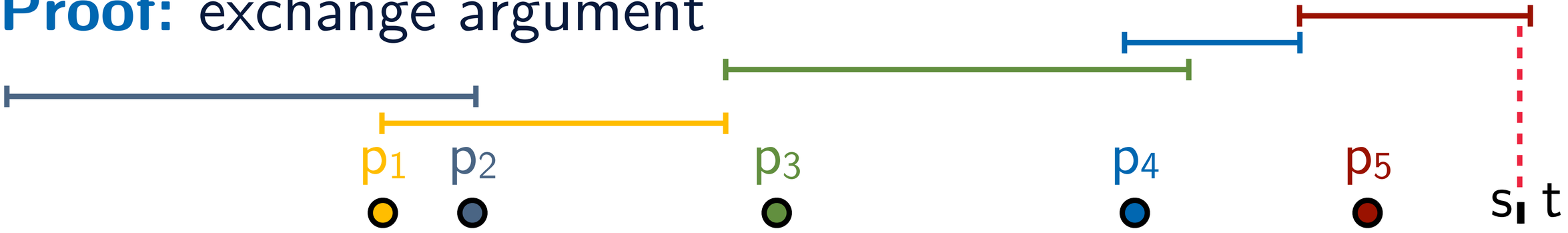
interval covering problem

- length $B_i/2$ containing $p_i$
- range: from $l_i = p_i - B_i/2$ to $r_i = p_i + B_i/2$

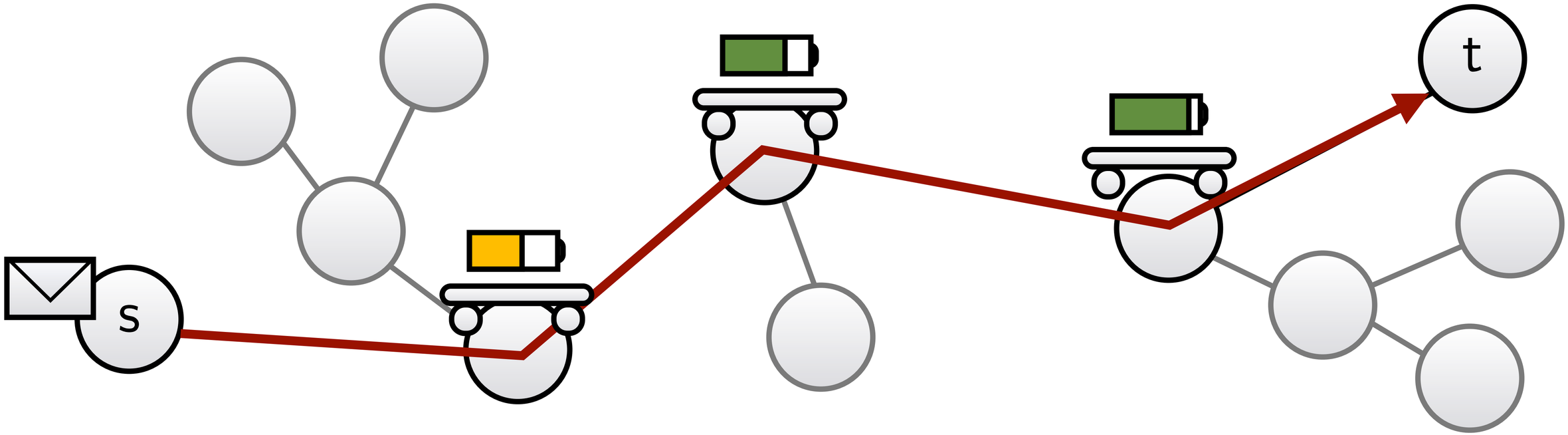
**Proof:** exchange argument

## Greedy Algorithm

- already covered up to $s$
- pick interval to minimize $r_i$ among all with $l_i \leq s < r_i$
- Set $s' = min(r_i, s + B_i/2)$



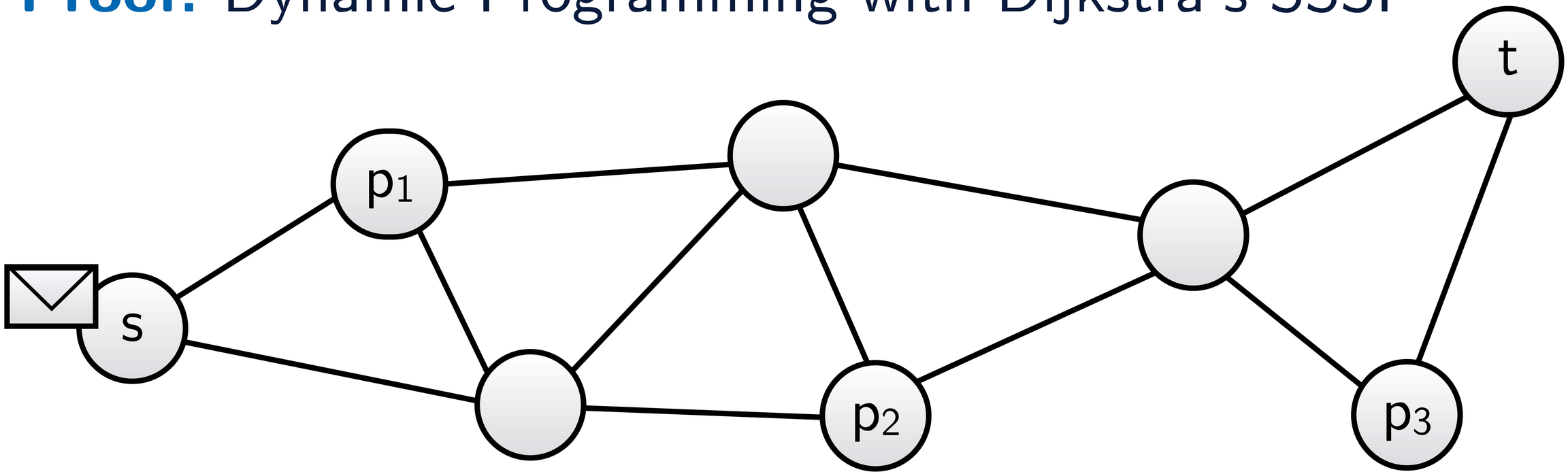
**Theorem:** There is a $O(n \cdot k + k \cdot \log k)$ algorithm for trees.




Surprising as non-returning setting is weakly NP-hard.
[Chalopin, Jacob, Mihalák, Widmayer, 2014]

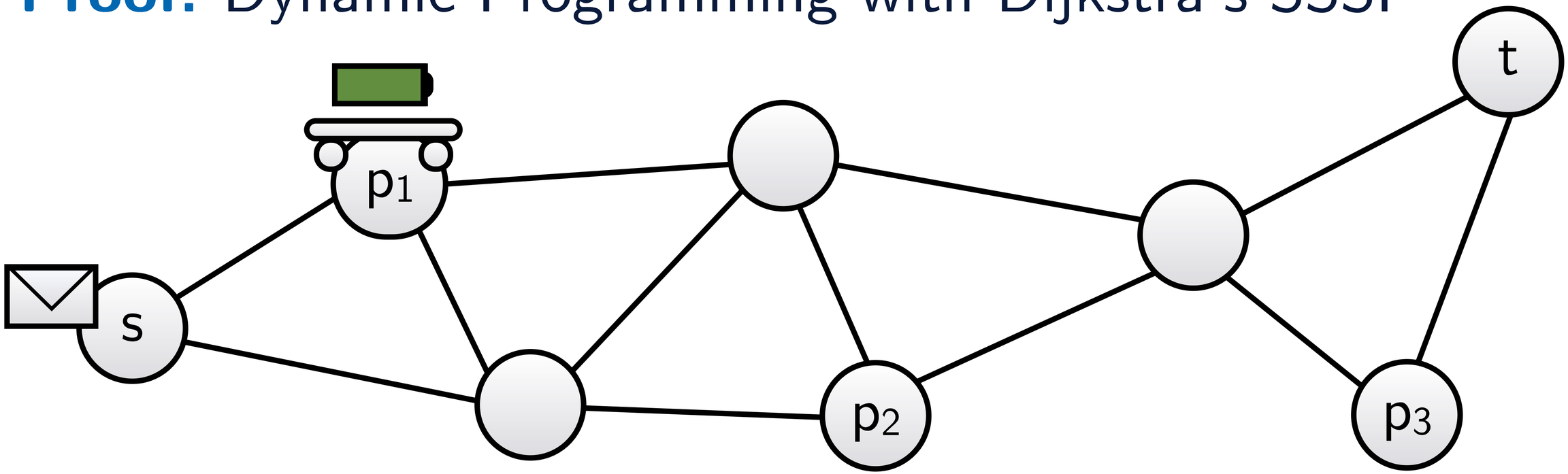
**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP



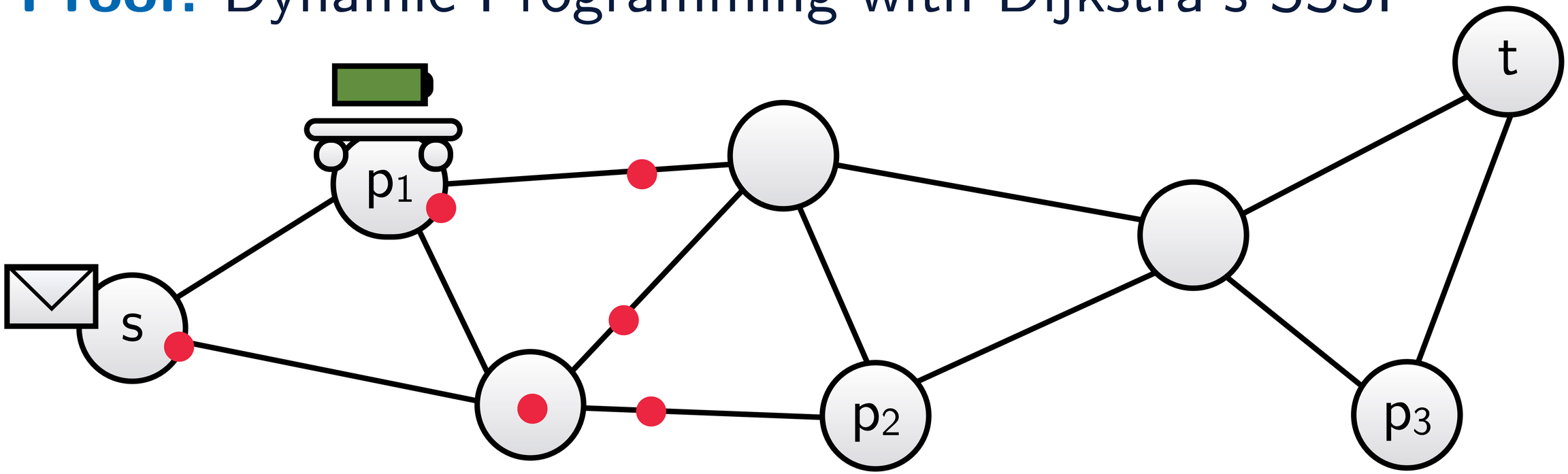
**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP



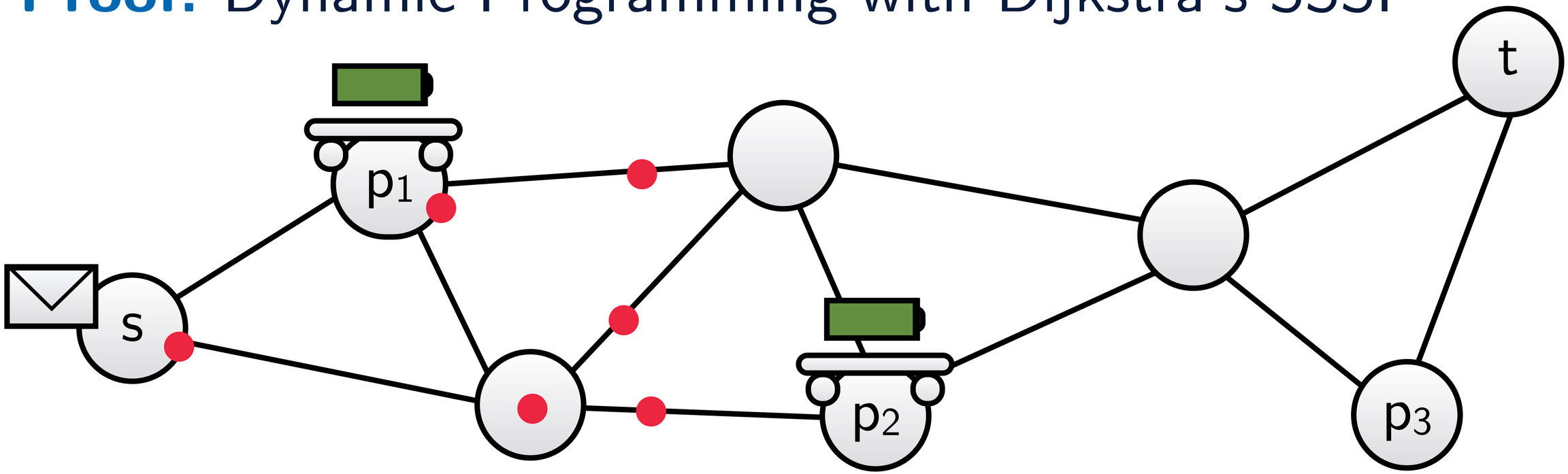
**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP



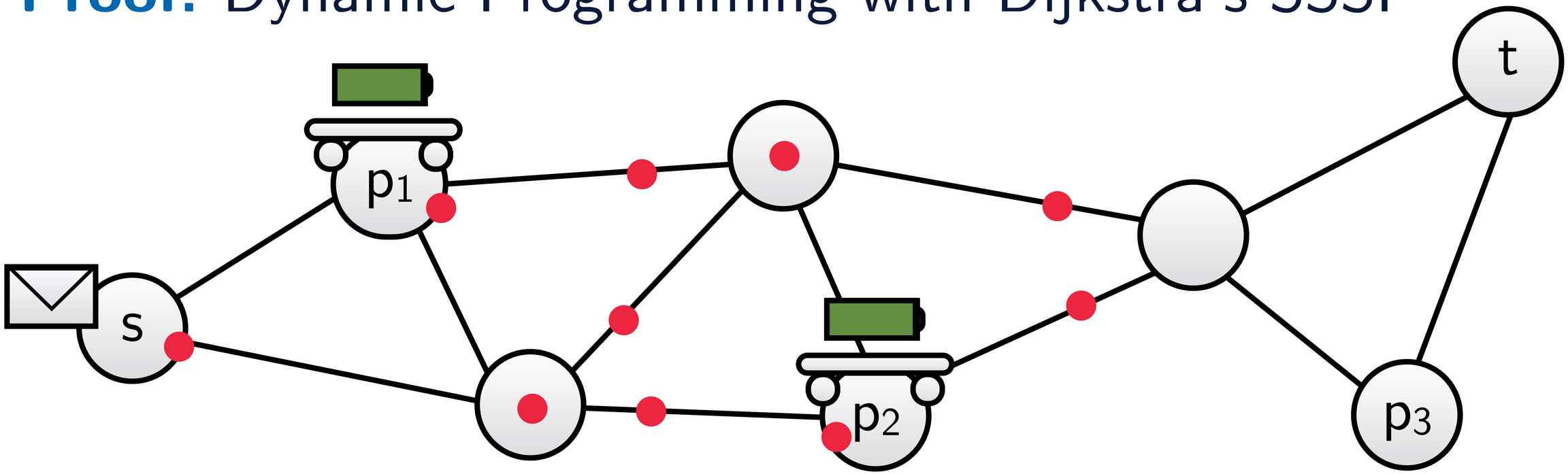
**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP



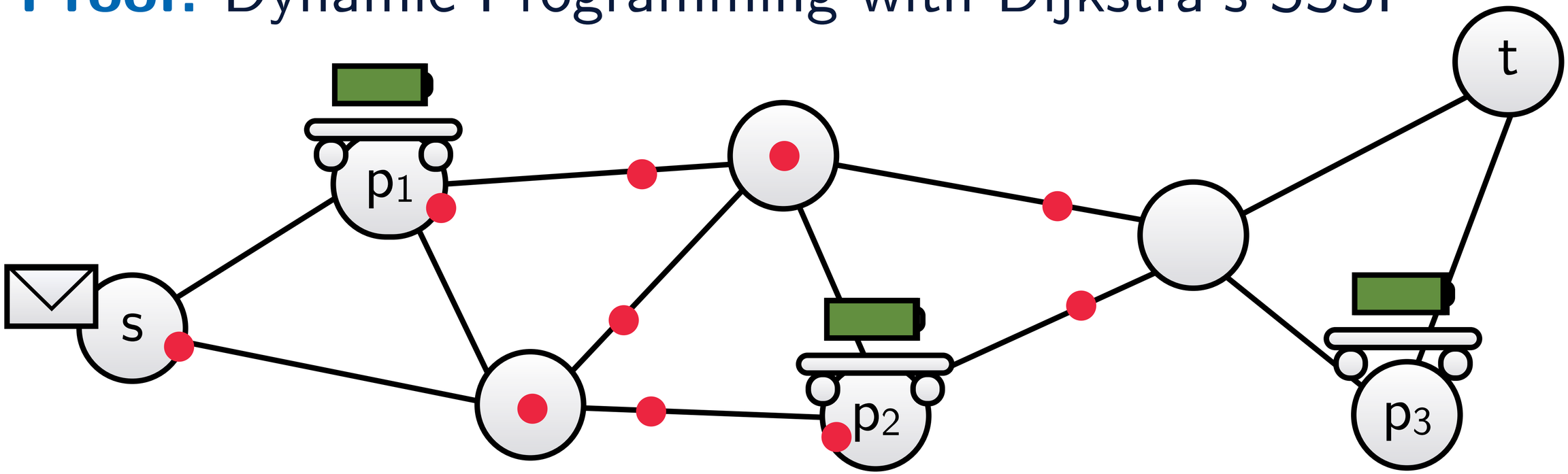
**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP



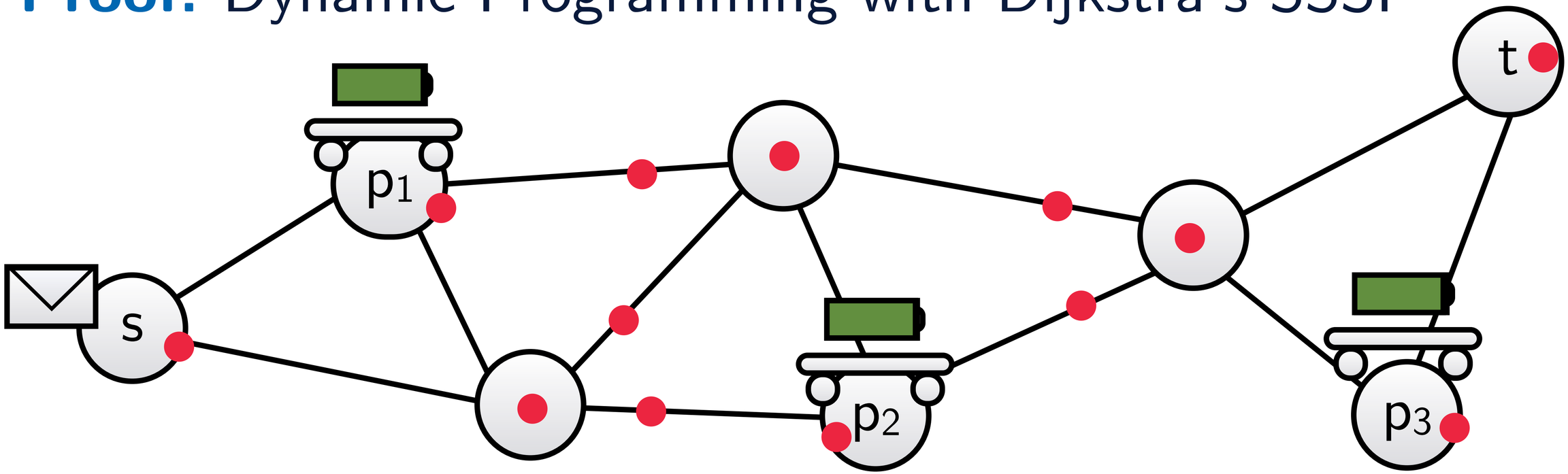
**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP



**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP



**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP

**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP

**Corollary:** For constantly many agents, we can do it in poly(n,m) by brute forcing the order of the agents.

**Theorem:** If agents are restricted to carry the message in a fixed order, we can decide in time $O(k(n+m)(n \cdot \log n + m))$.

**Proof:** Dynamic Programming with Dijkstra's SSSP

**Corollary:** For constantly many agents, we can do it in poly(n,m) by brute forcing the order of the agents.

**Question:** What about many agents?

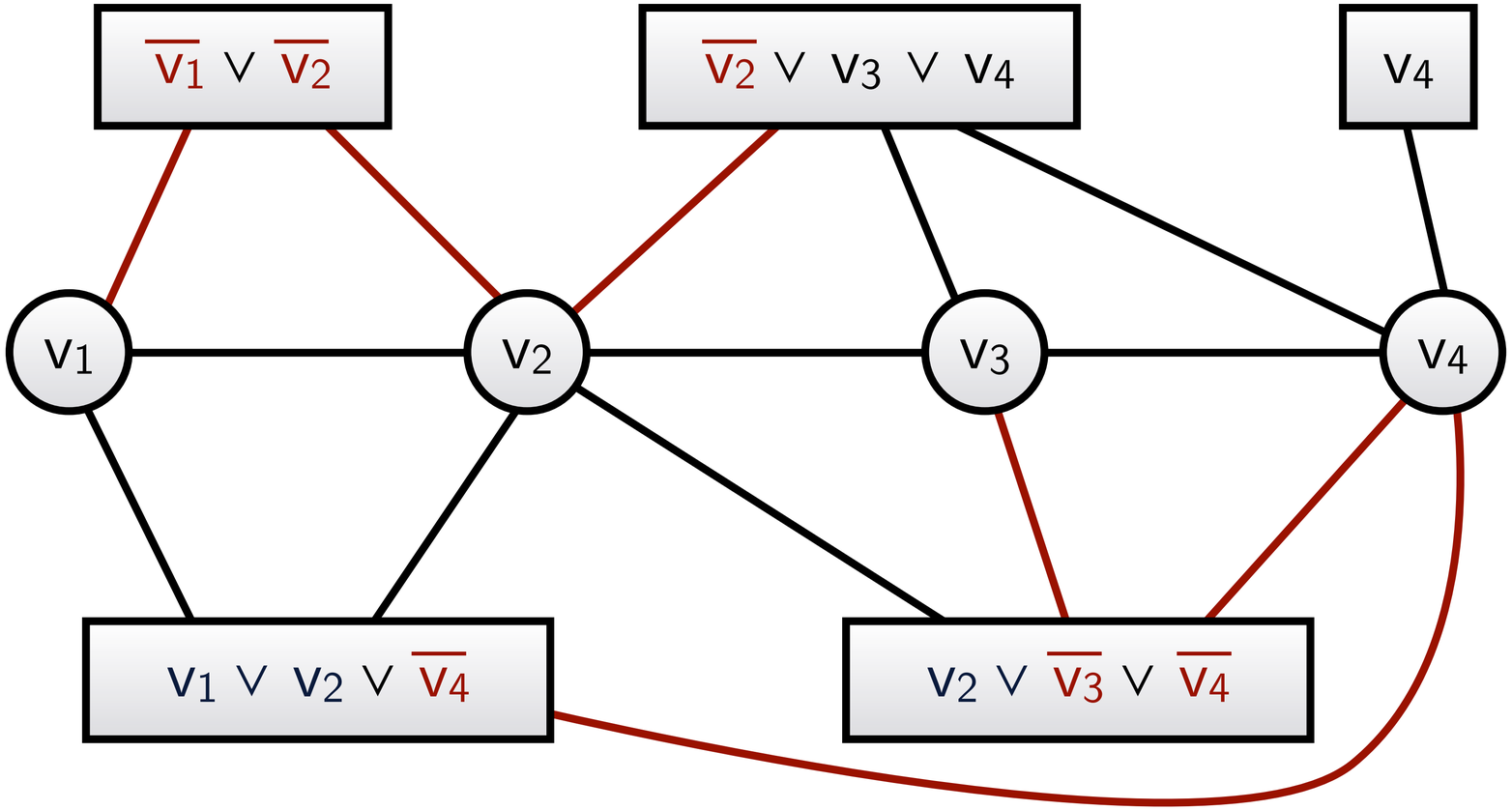
**Theorem:** Our problem is strongly NP-hard even on planar graphs.

**Proof Sketch:** Reduction from Planar-3SAT.

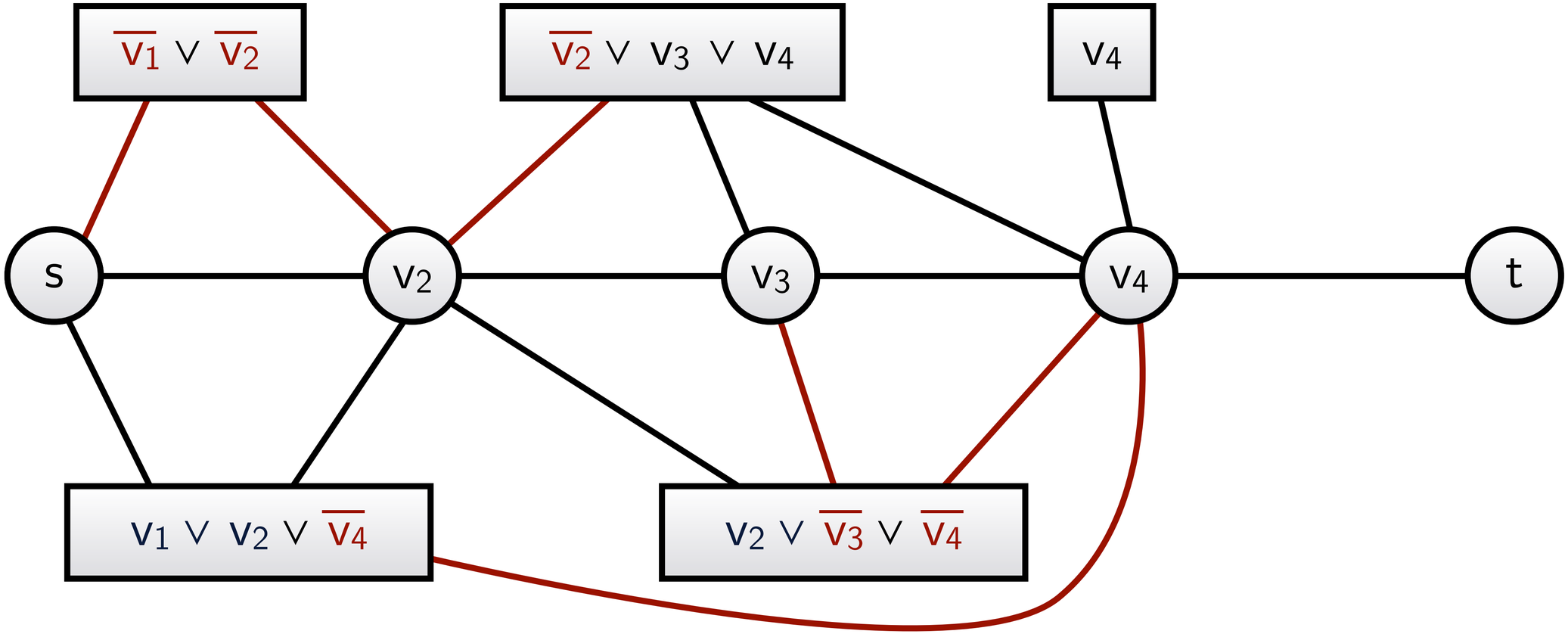
**Proof Sketch:** Reduction from Planar-3SAT.

**Proof Sketch:** Reduction from Planar-3SAT.

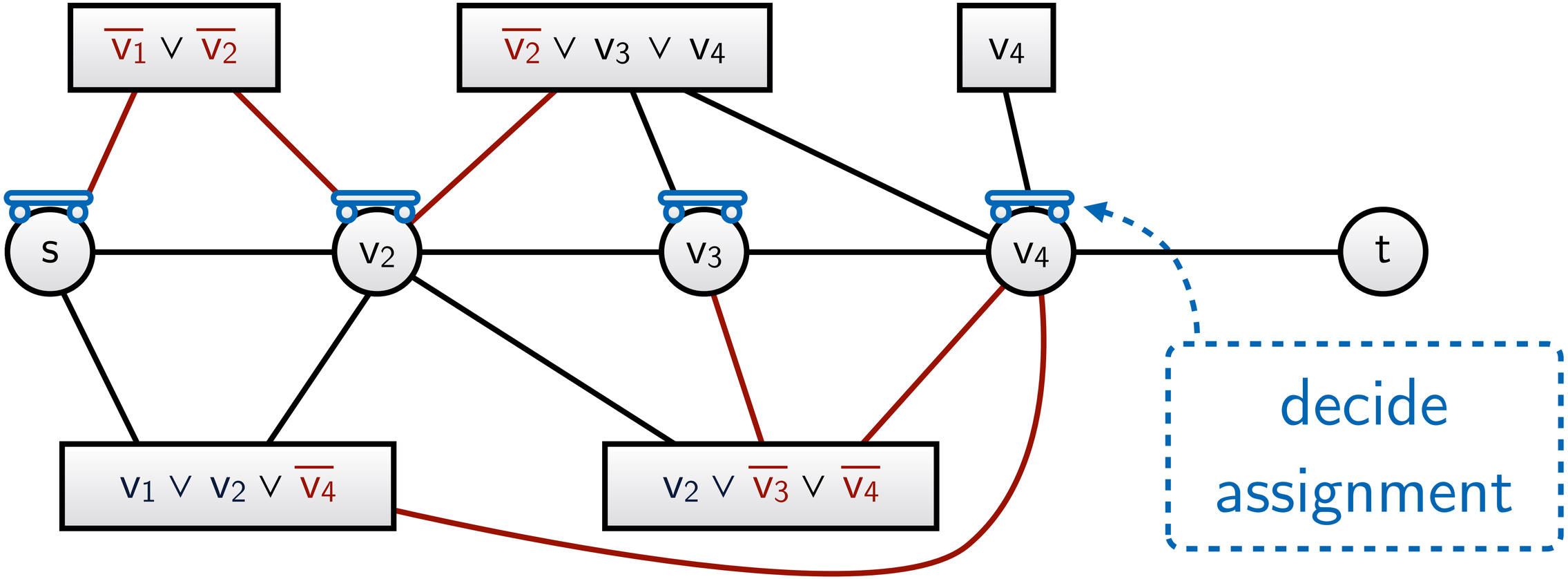
**Proof Sketch:** Reduction from Planar-3SAT.



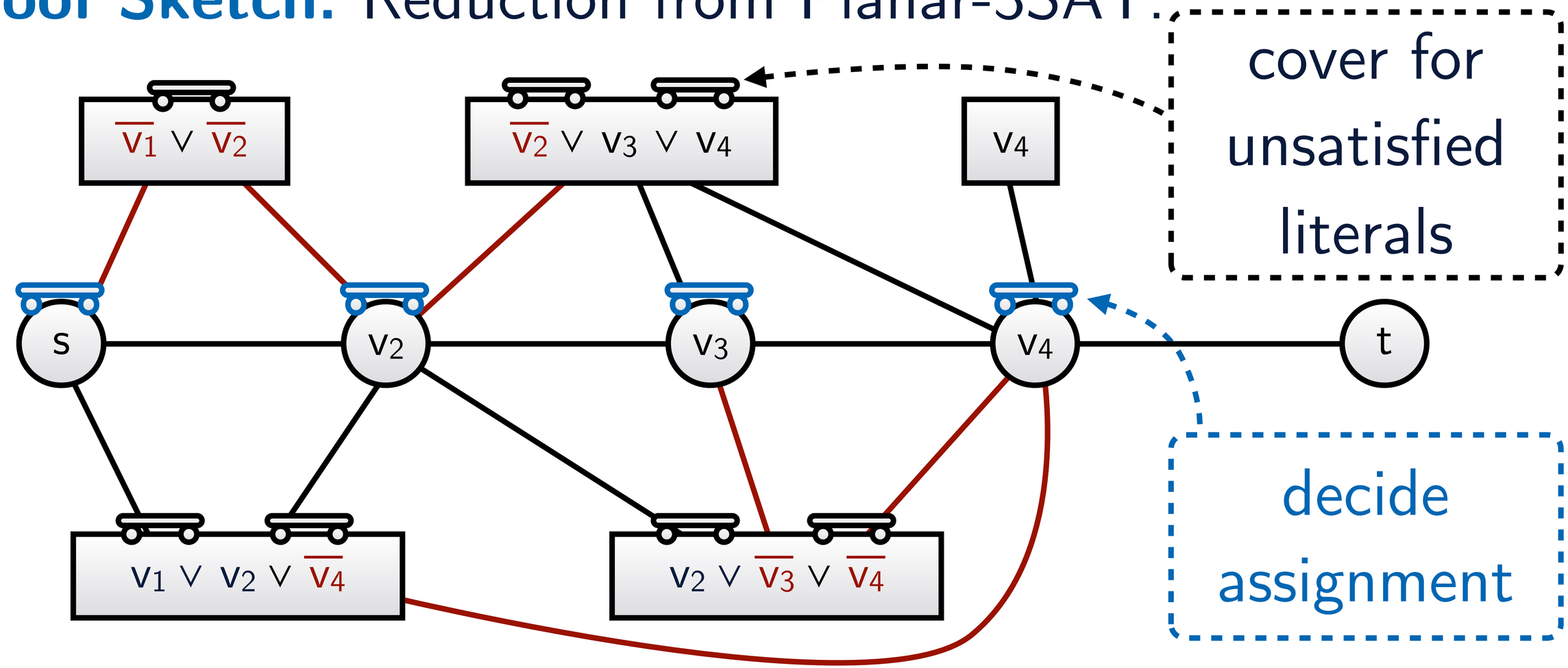
**Proof Sketch:** Reduction from Planar-3SAT.



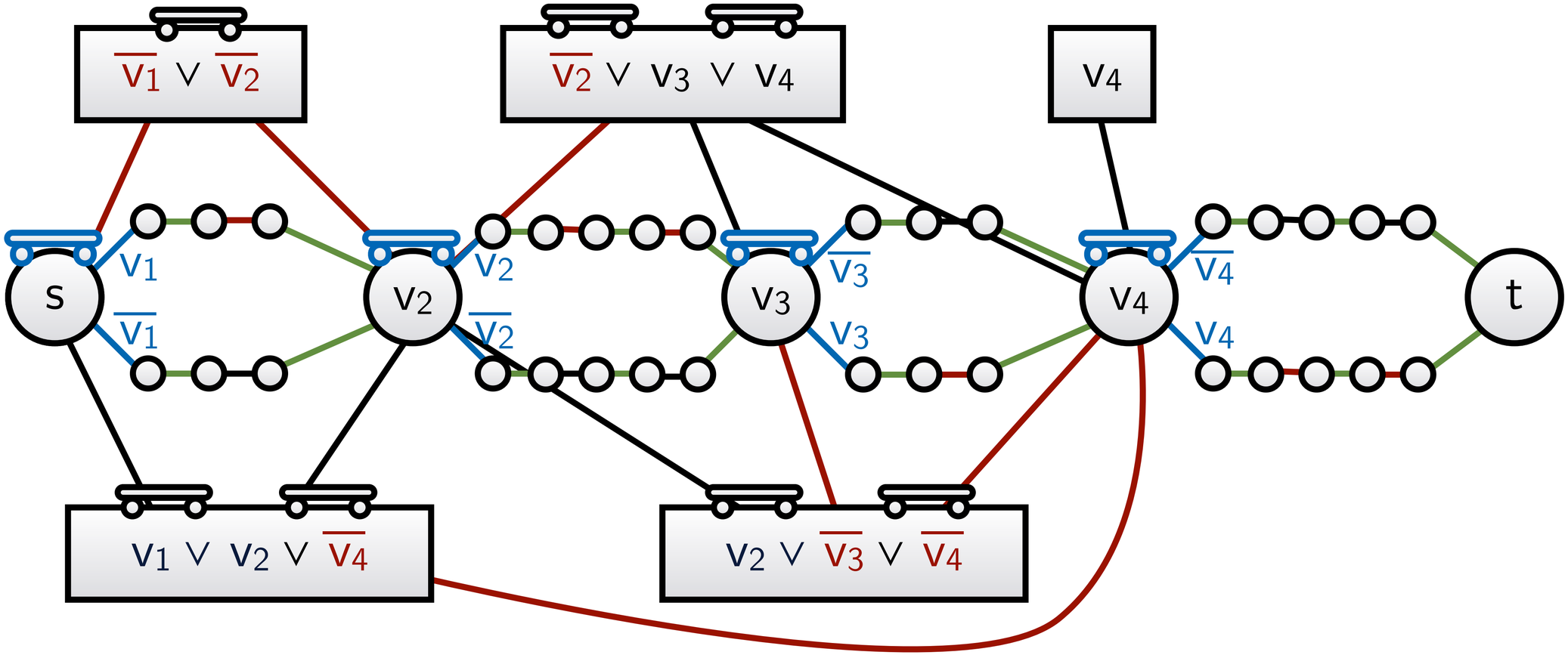
**Proof Sketch:** Reduction from Planar-3SAT.



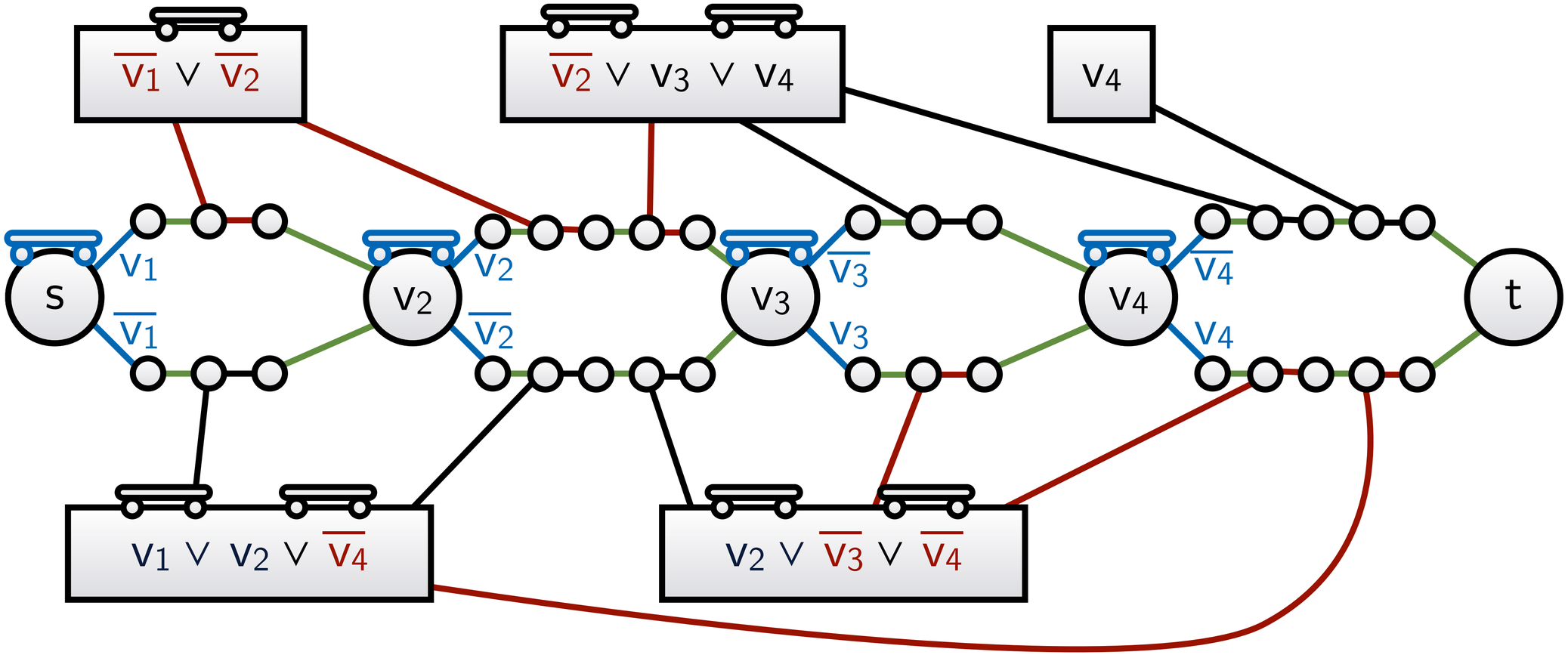
**Proof Sketch:** Reduction from Planar-3SAT.



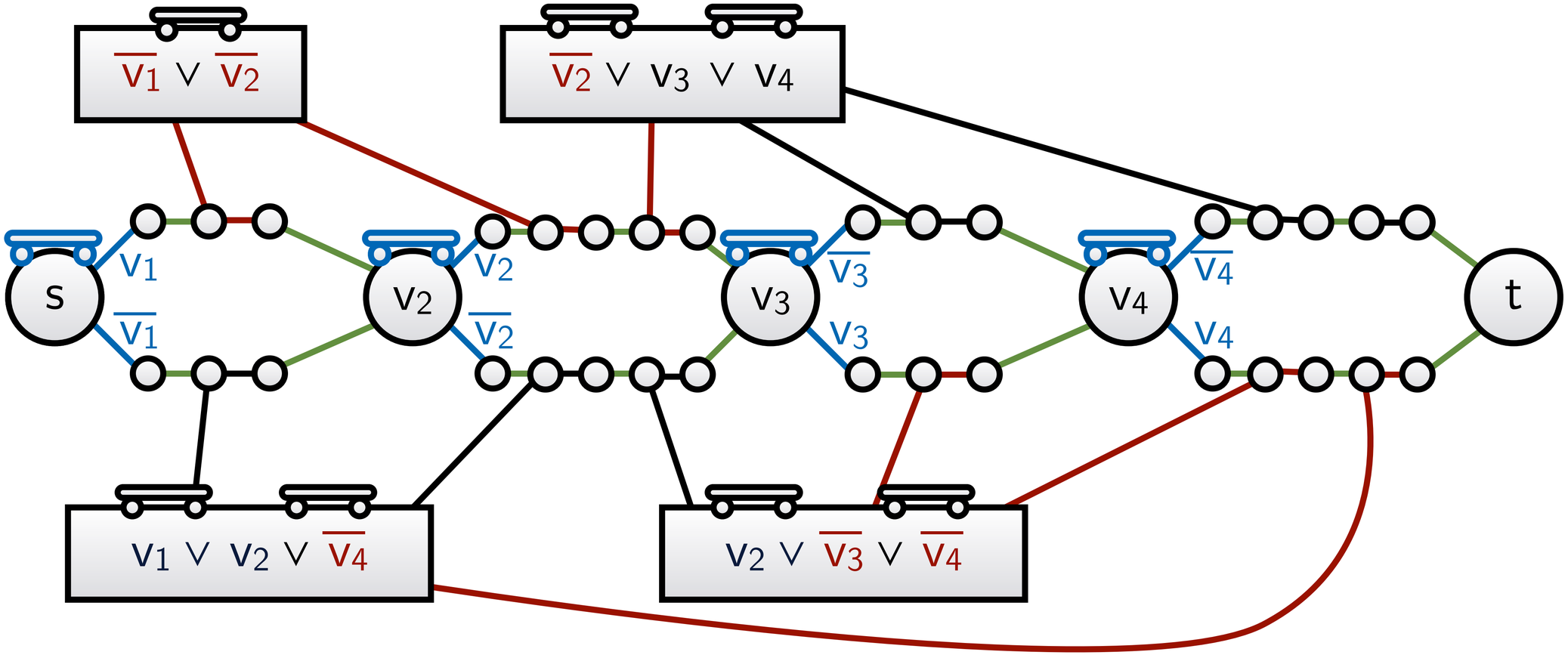
**Proof Sketch:** Reduction from Planar-3SAT.



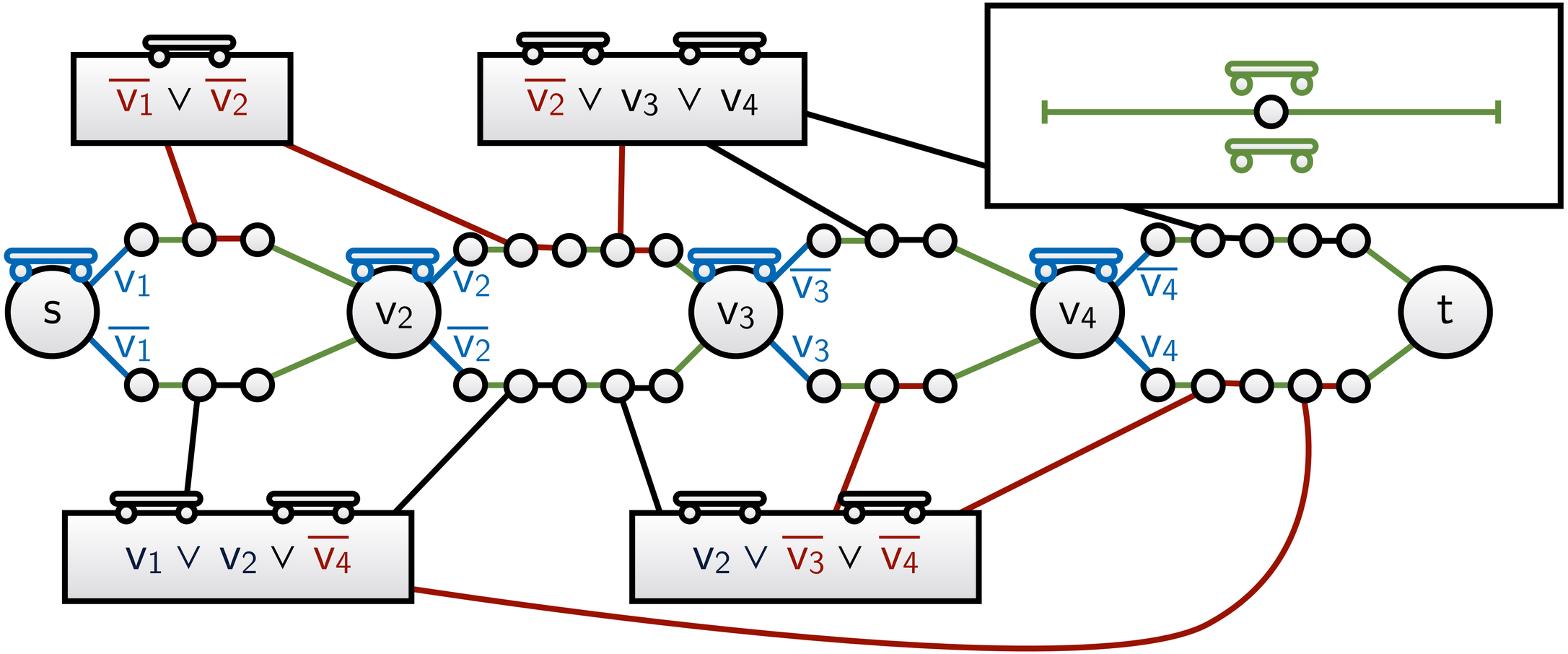
**Proof Sketch:** Reduction from Planar-3SAT.



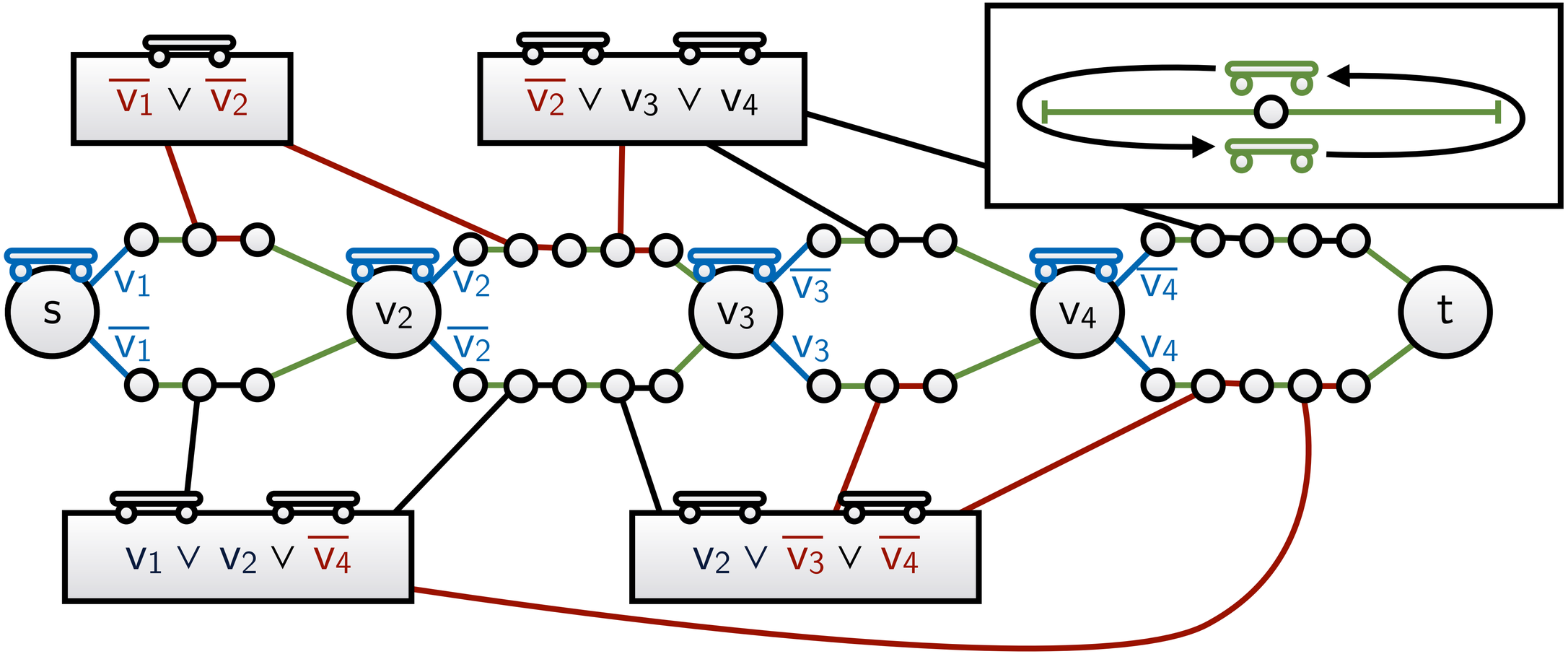
**Proof Sketch:** Reduction from Planar-3SAT.



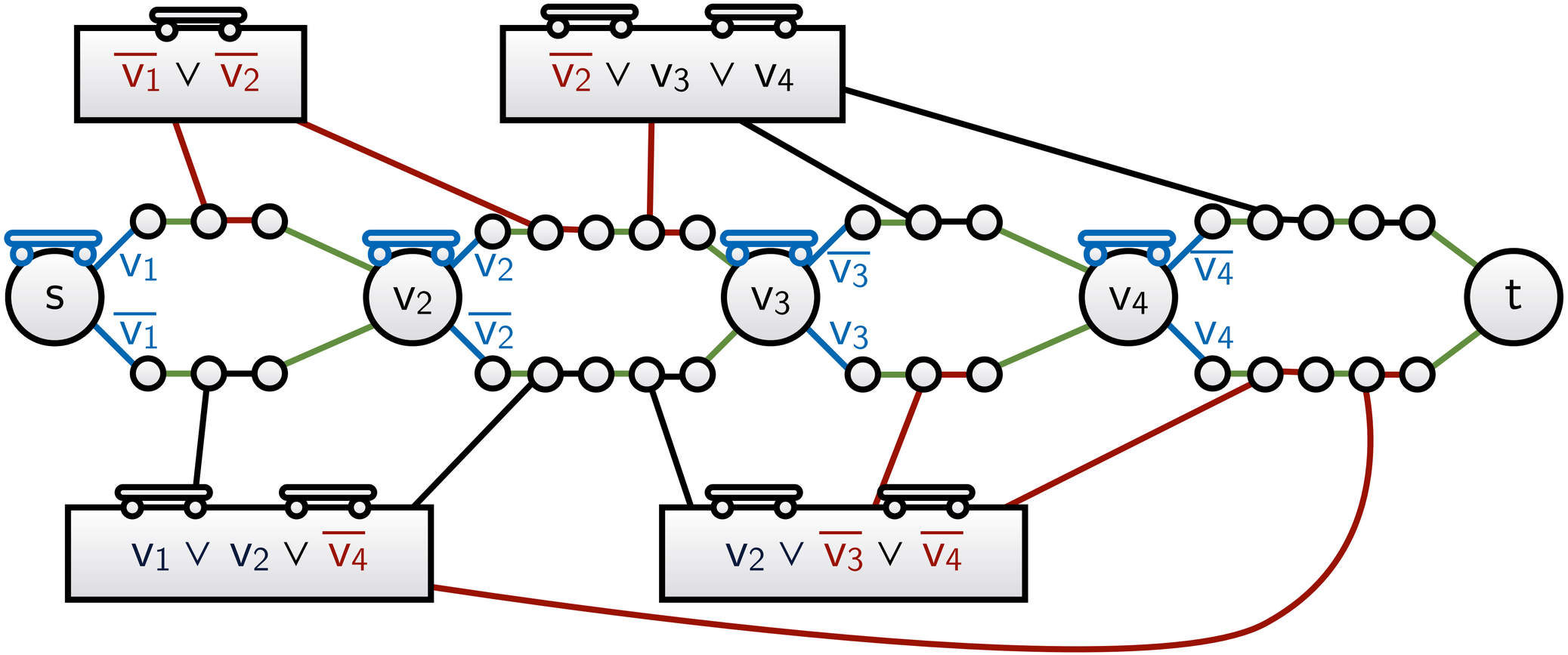
**Proof Sketch:** Reduction from Planar-3SAT.



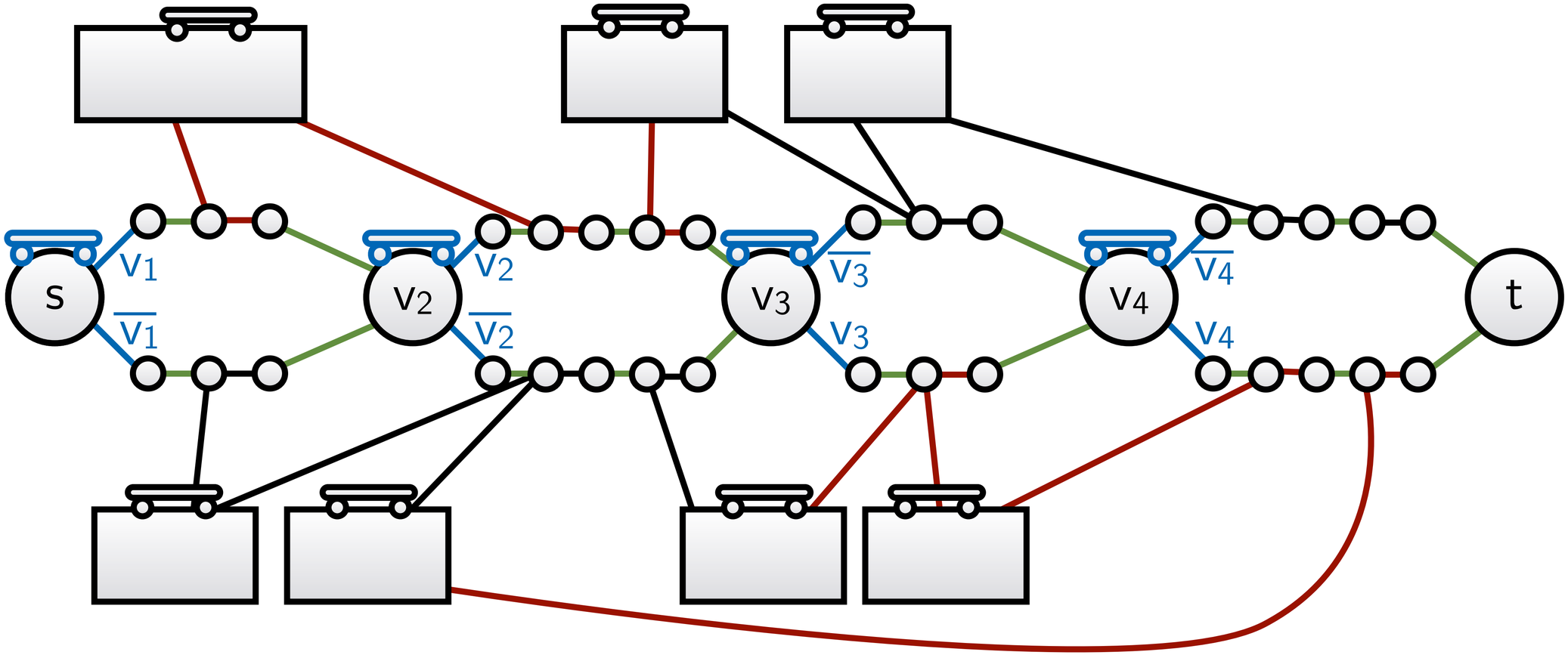
**Proof Sketch:** Reduction from Planar-3SAT.



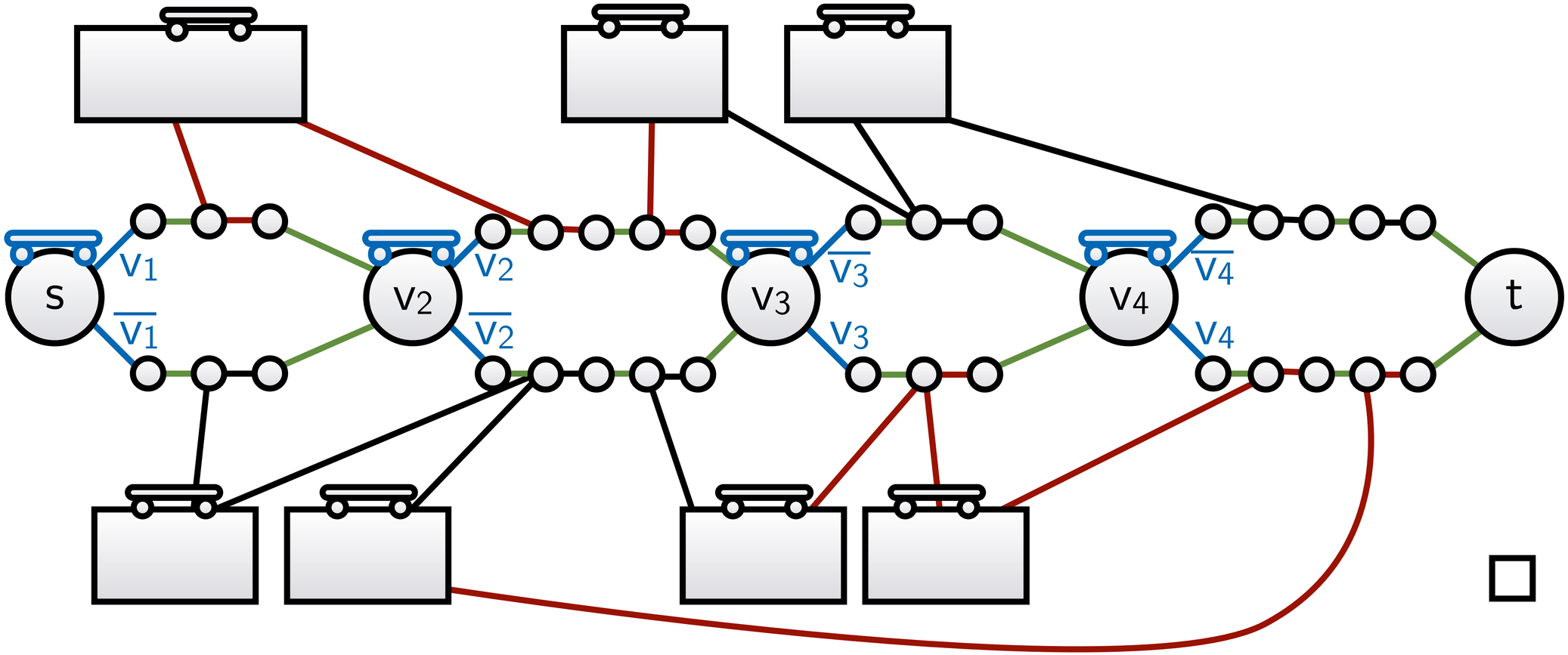
**Proof Sketch:** Reduction from Planar-3SAT.



**Proof Sketch:** Reduction from Planar-3SAT.



**Proof Sketch:** Reduction from Planar-3SAT.



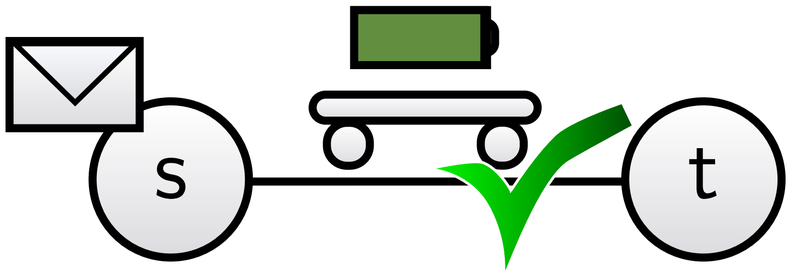

□

# Resource Augmentation / Definition

What is a 2-resource-augmented algorithm?

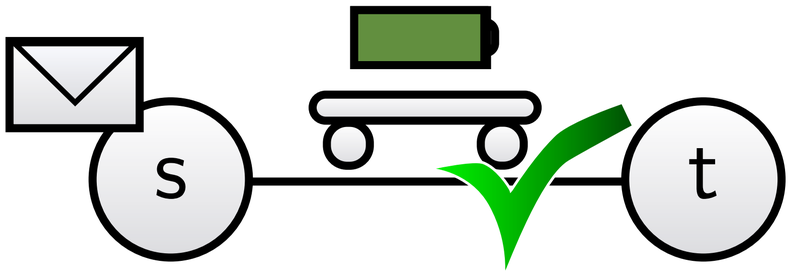
What is a 2-resource-augmented algorithm?

**Given:**

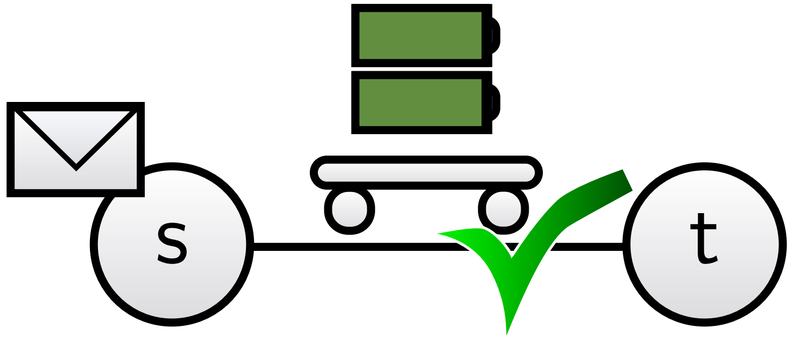


What is a 2-resource-augmented algorithm?

**Given:**

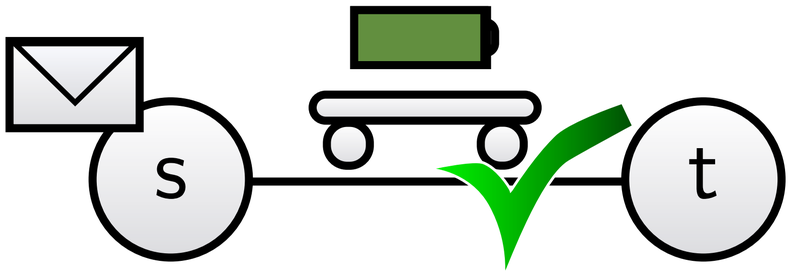


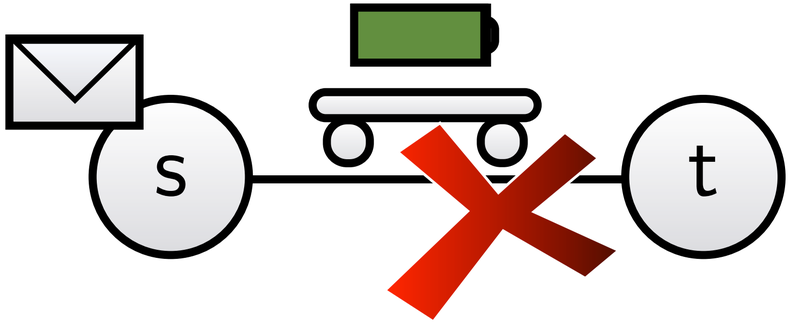

**Algorithm:**



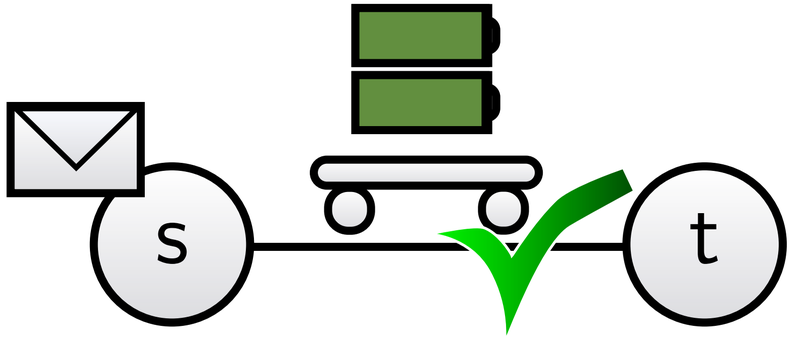
What is a 2-resource-augmented algorithm?

**Given:**



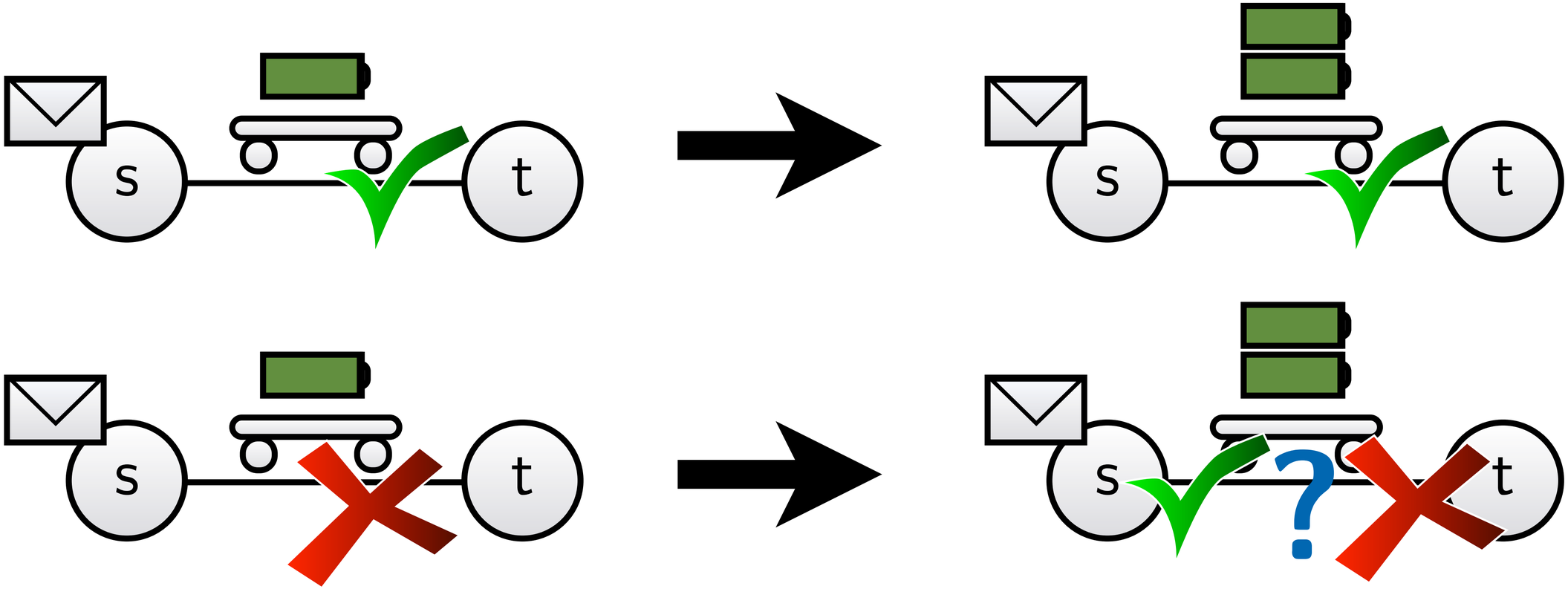




**Algorithm:**



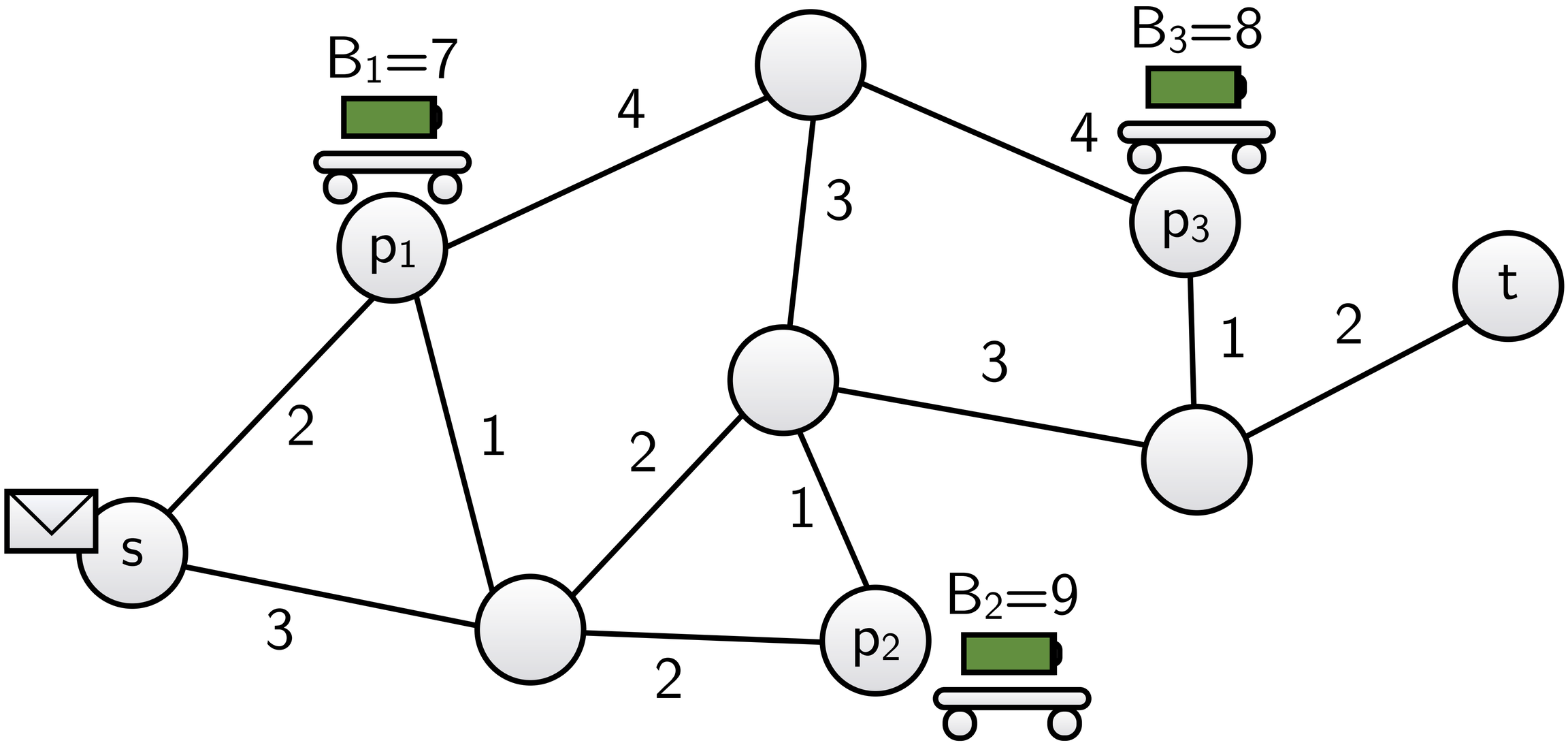
What is a 2-resource-augmented algorithm?

**Given:** **Algorithm:**



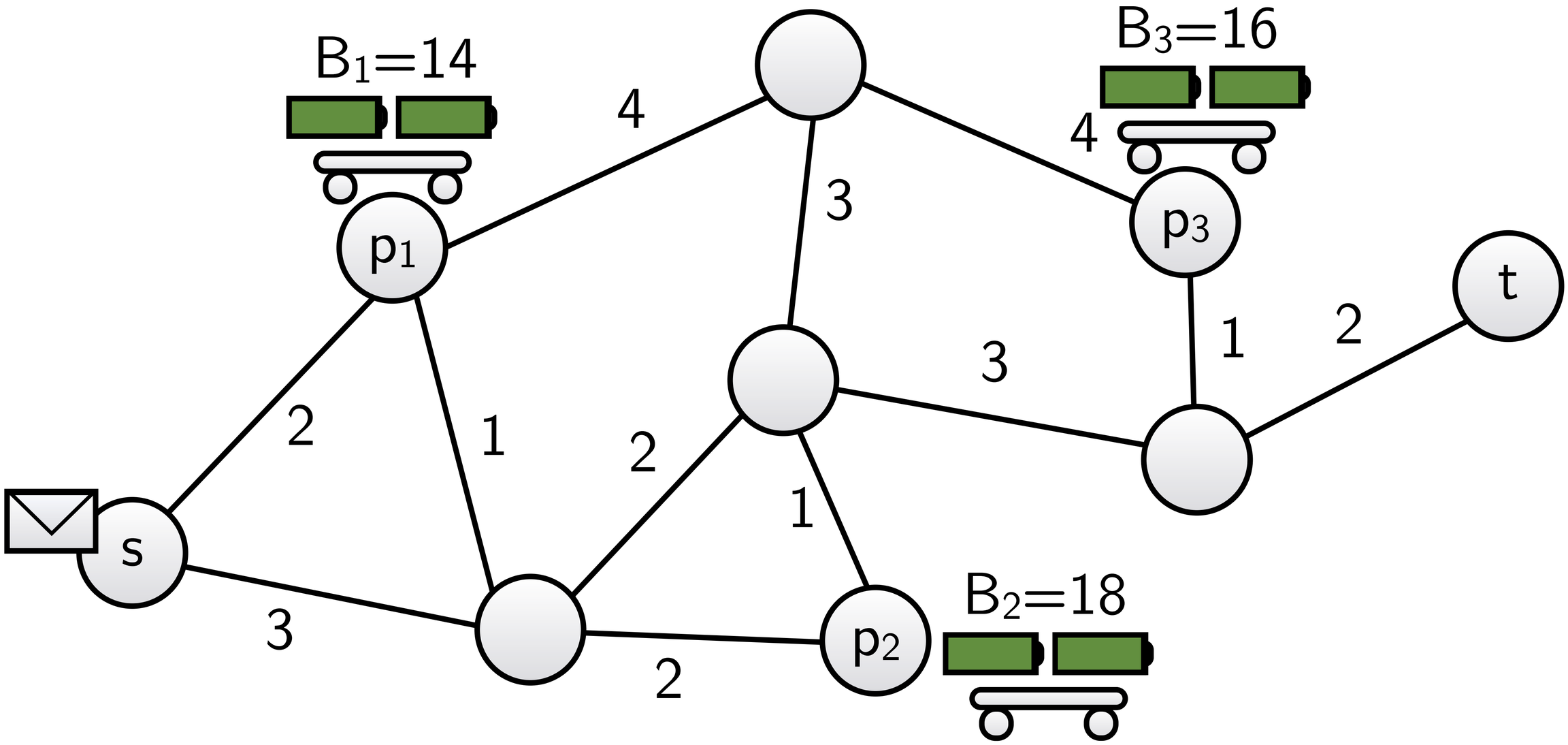
# Resource Augmentation / Example

ETH zürich



# Resource Augmentation / Example

ETH zürich



# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

**Proof Sketch:** Balls of radius $B_i/2$.

# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

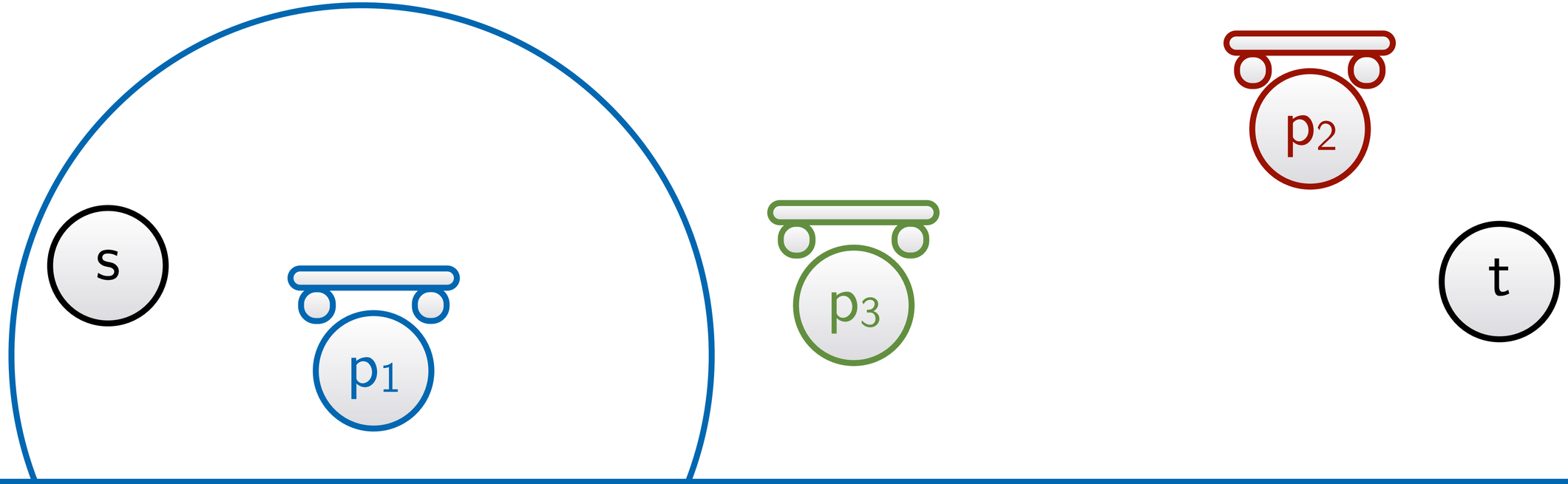
**Proof Sketch:** Balls of radius $B_i/2$.

# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

**Proof Sketch:** Balls of radius $B_i/2$.

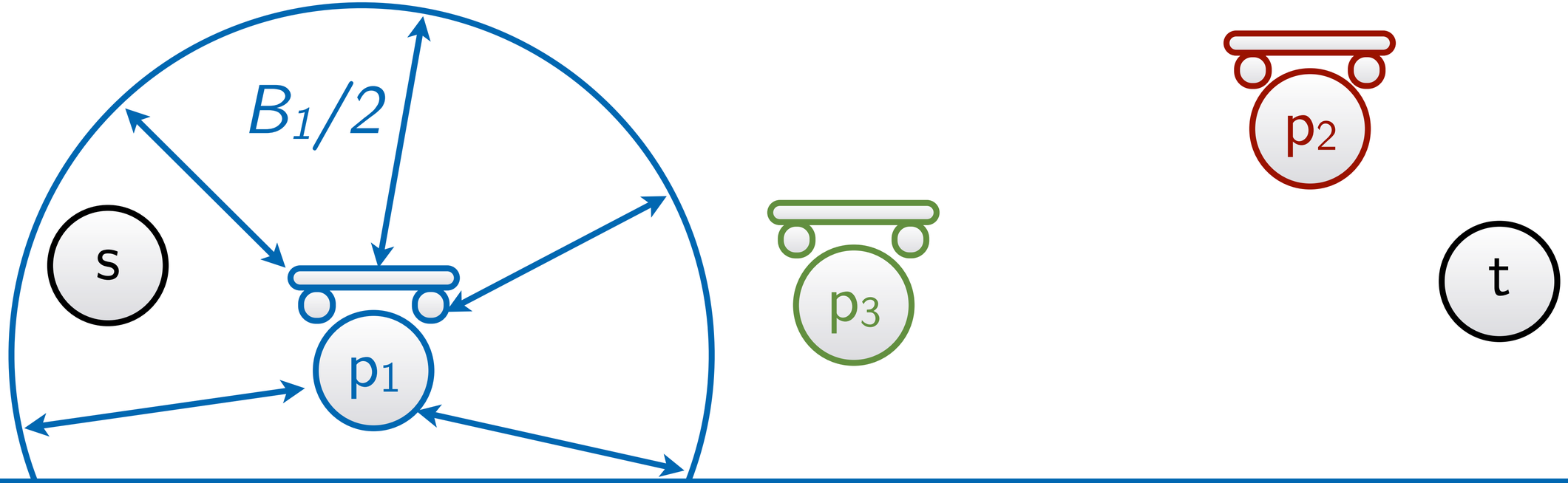


# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

**Proof Sketch:** Balls of radius $B_i/2$.

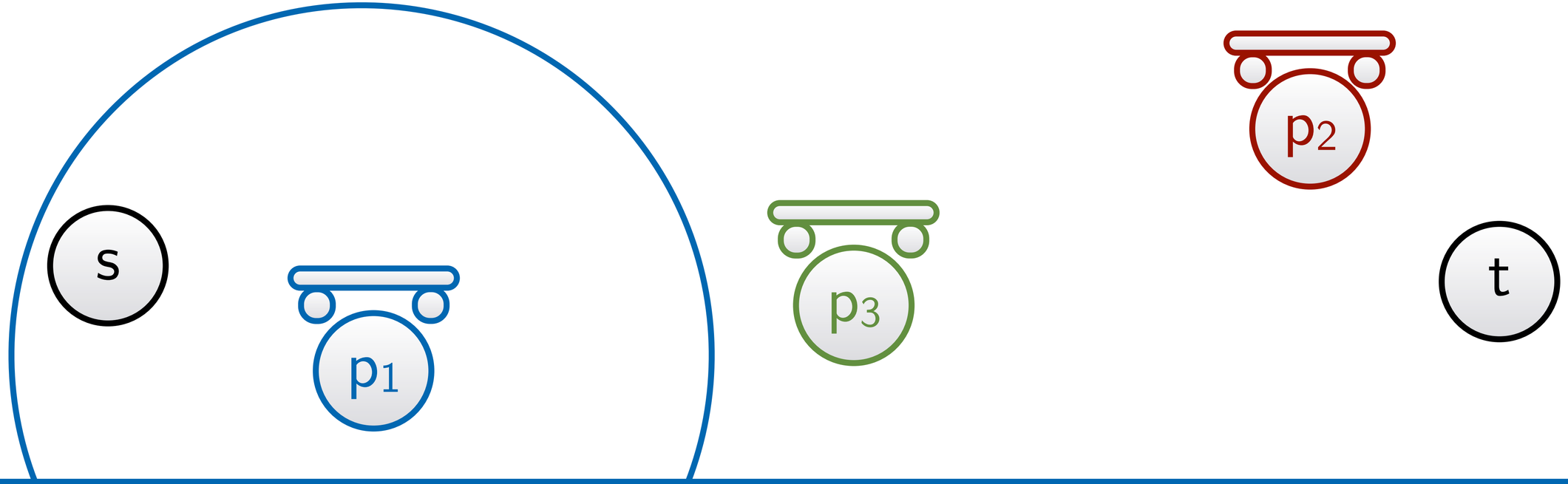


# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

**Proof Sketch:** Balls of radius $B_i/2$.

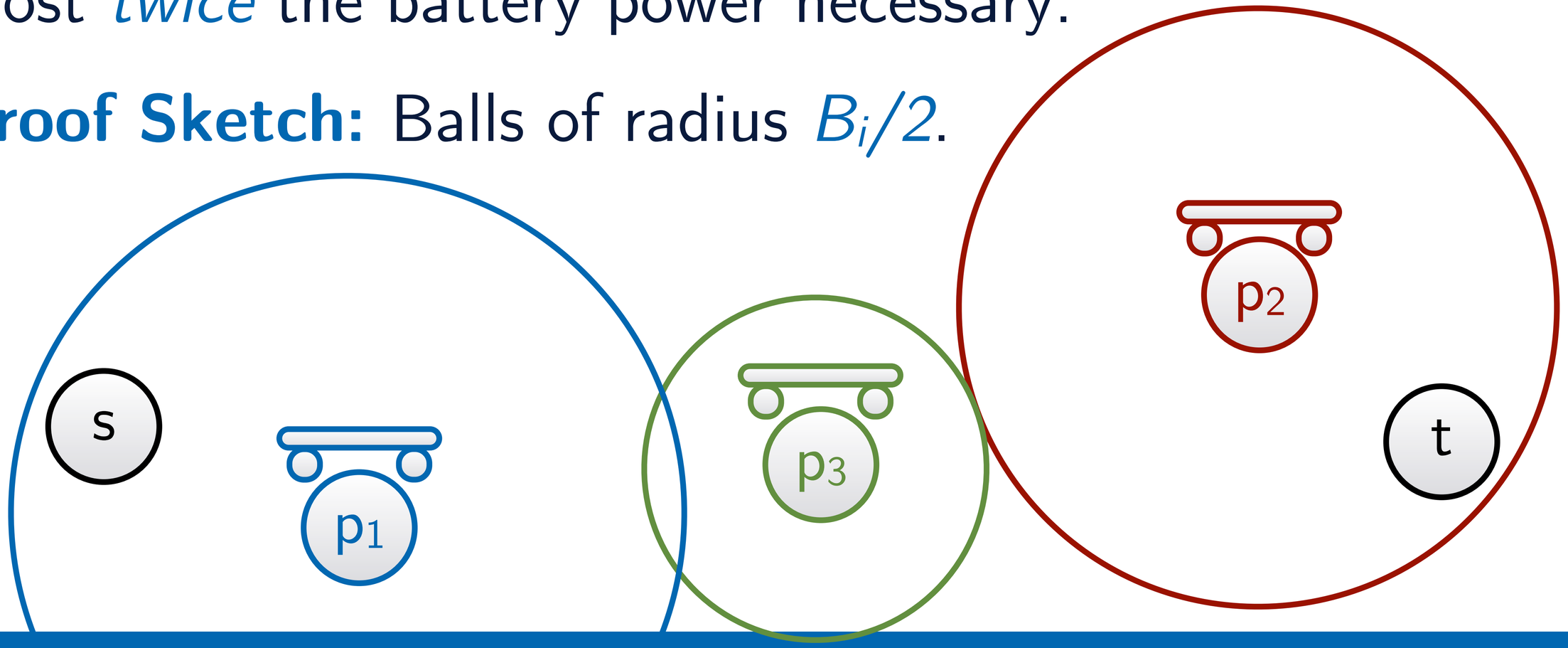


# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

**Proof Sketch:** Balls of radius $B_i/2$.

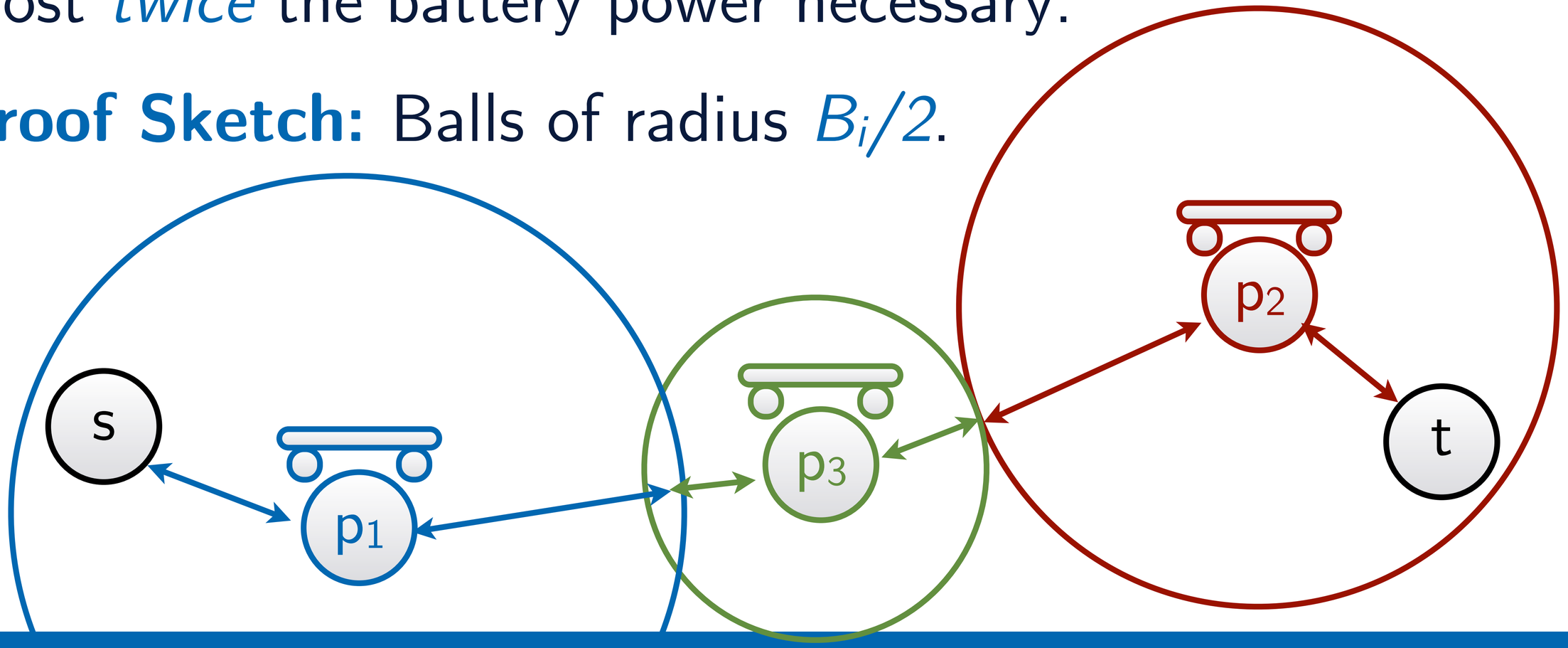


# Resource Augmentation / Factor 2

ETH zürich

**Theorem:** We can do it if we are allowed to use at most *twice* the battery power necessary.

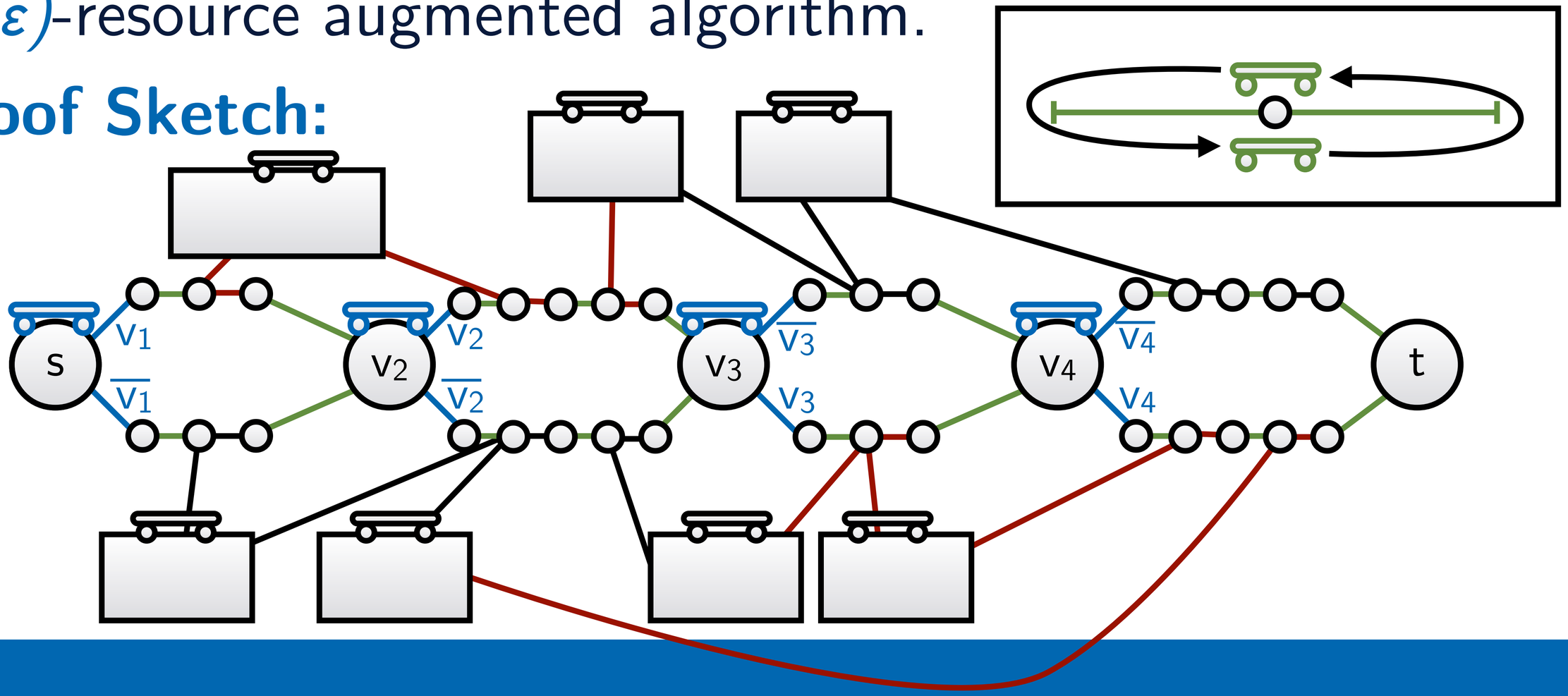
**Proof Sketch:** Balls of radius $B_i/2$.



**Theorem:** Unless $P=NP$, there is no polynomial-time $(2\text{-}\varepsilon)$-resource augmented algorithm.

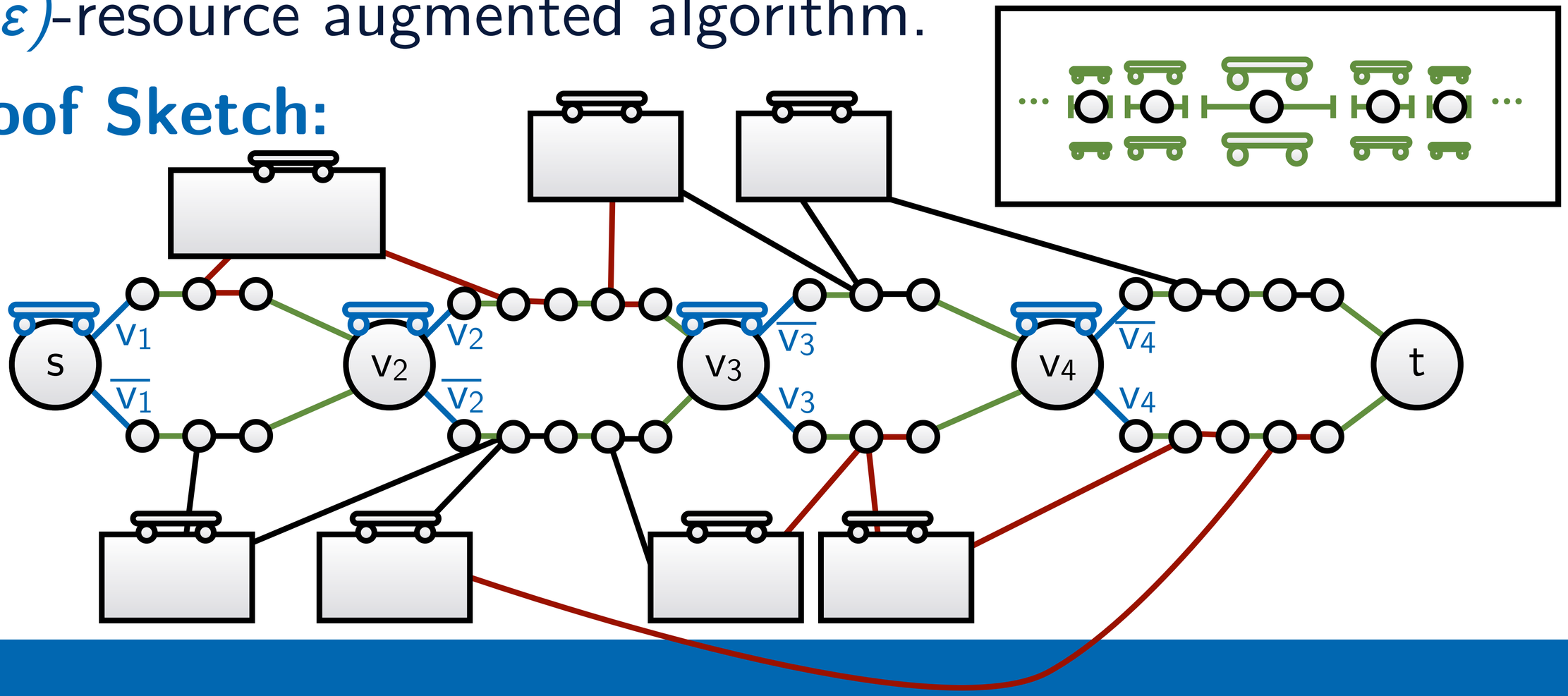
**Theorem:** Unless $P=NP$, there is no polynomial-time $(2\text{-}\varepsilon)$-resource augmented algorithm.

**Proof Sketch:**



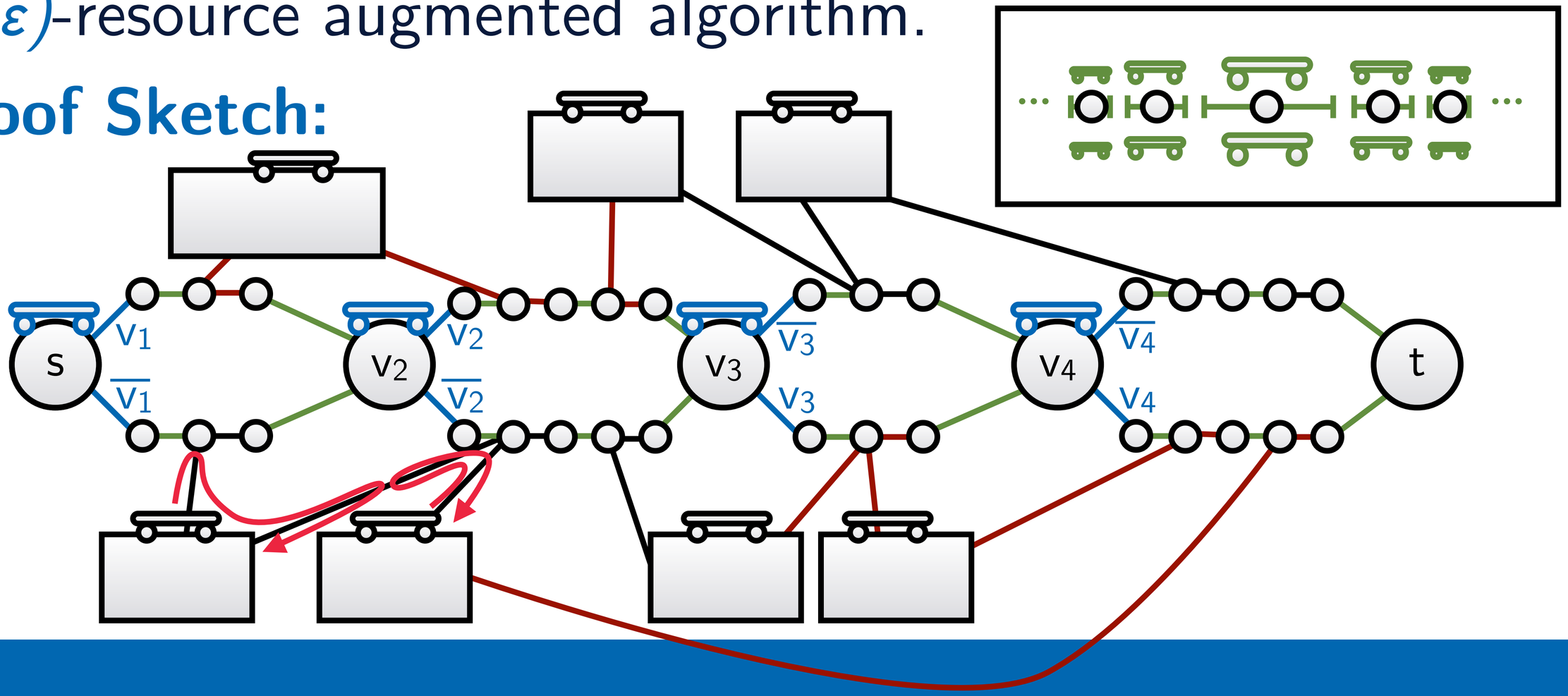
**Theorem:** Unless $P=NP$, there is no polynomial-time $(2\text{-}\varepsilon)$-resource augmented algorithm.

**Proof Sketch:**



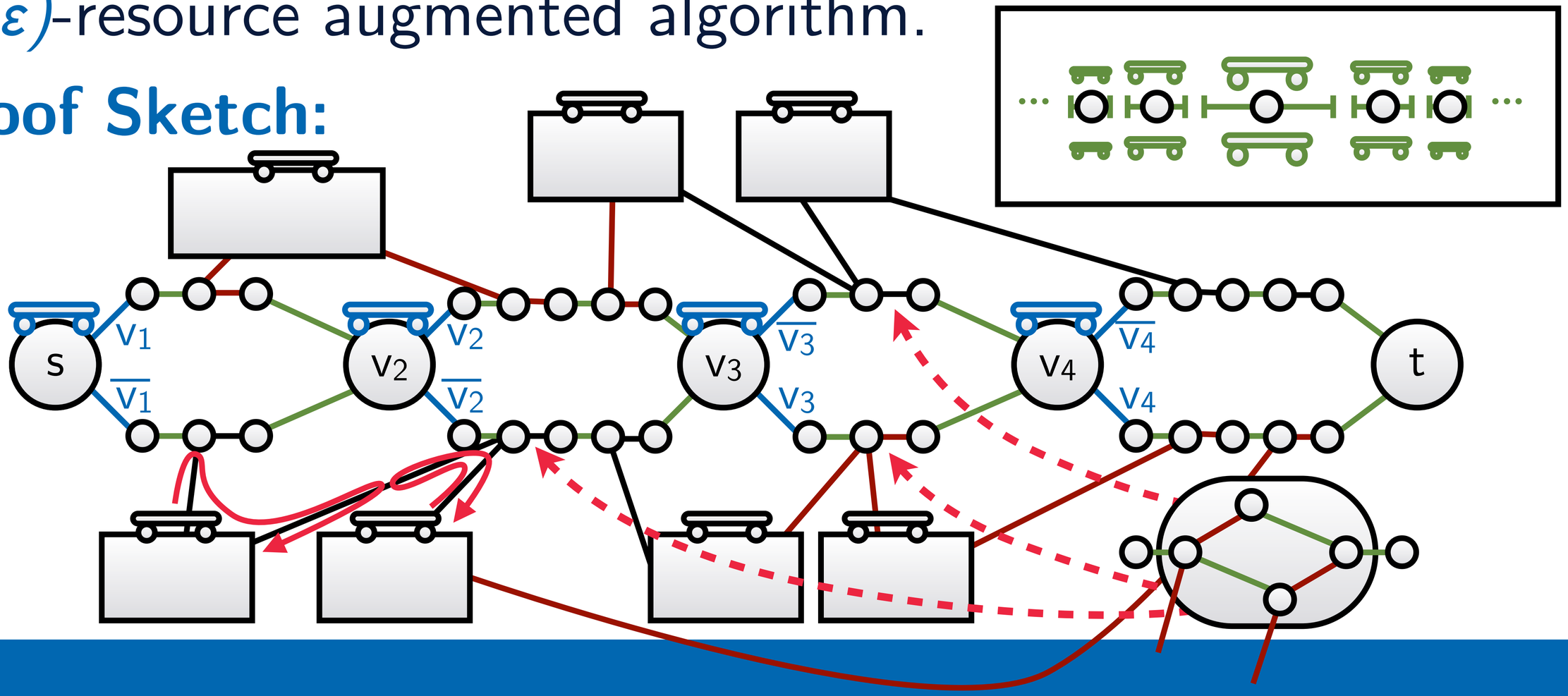
**Theorem:** Unless $P=NP$, there is no polynomial-time $(2\text{-}\varepsilon)$-resource augmented algorithm.

**Proof Sketch:**



**Theorem:** Unless $P=NP$, there is no polynomial-time $(2\text{-}\varepsilon)$-resource augmented algorithm.

**Proof Sketch:**



**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

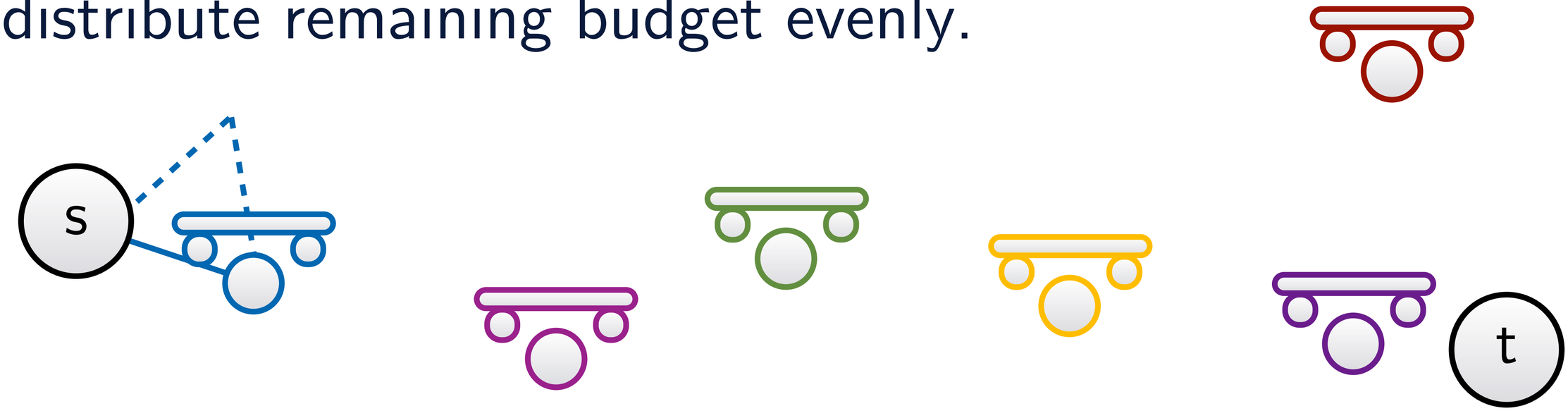
**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.



**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

**Theorem:** If all agents have equal budget, there is an *(2-2/k)*-resource augmented polynomial-time algorithm.

**Proof Sketch:** Guess first and last agent.

Redistribute remaining budget evenly.

## Contributions

- fast, greedy algorithm for paths and trees
- strong NP-hardness for planar graphs
- 2-resource-augmentation algorithm and it is tight
- more for non-returning case

## Future Work

- multiple messages
  - on a line, or
  - with fixed agent order

**Thank you!**