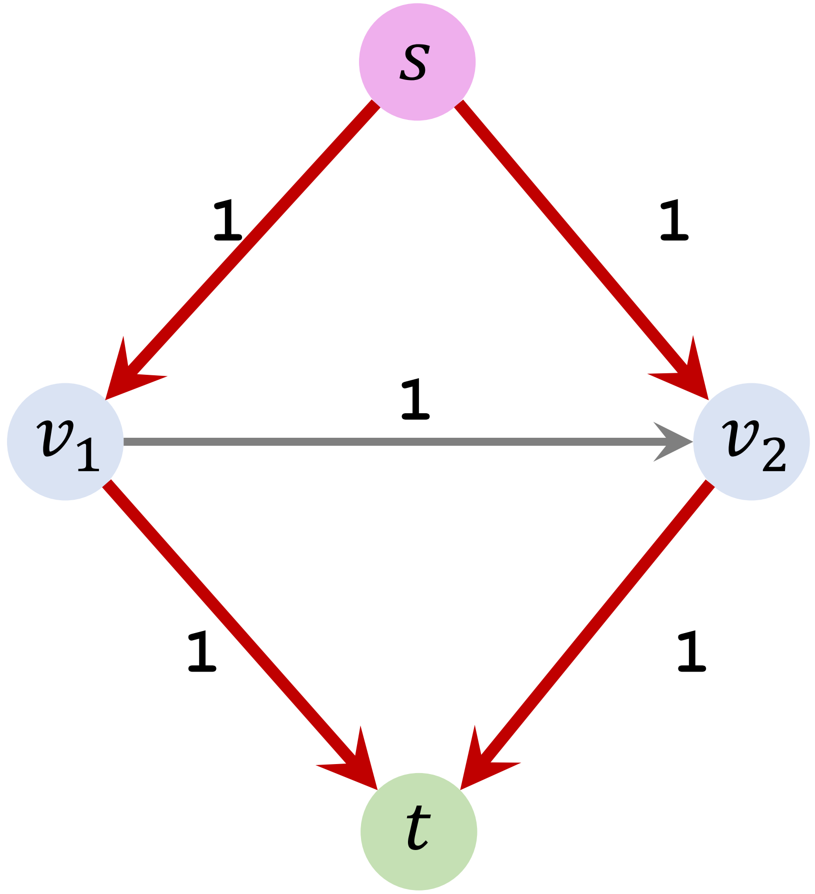
Ford-Fulkerson Algorithm

Shusen Wang

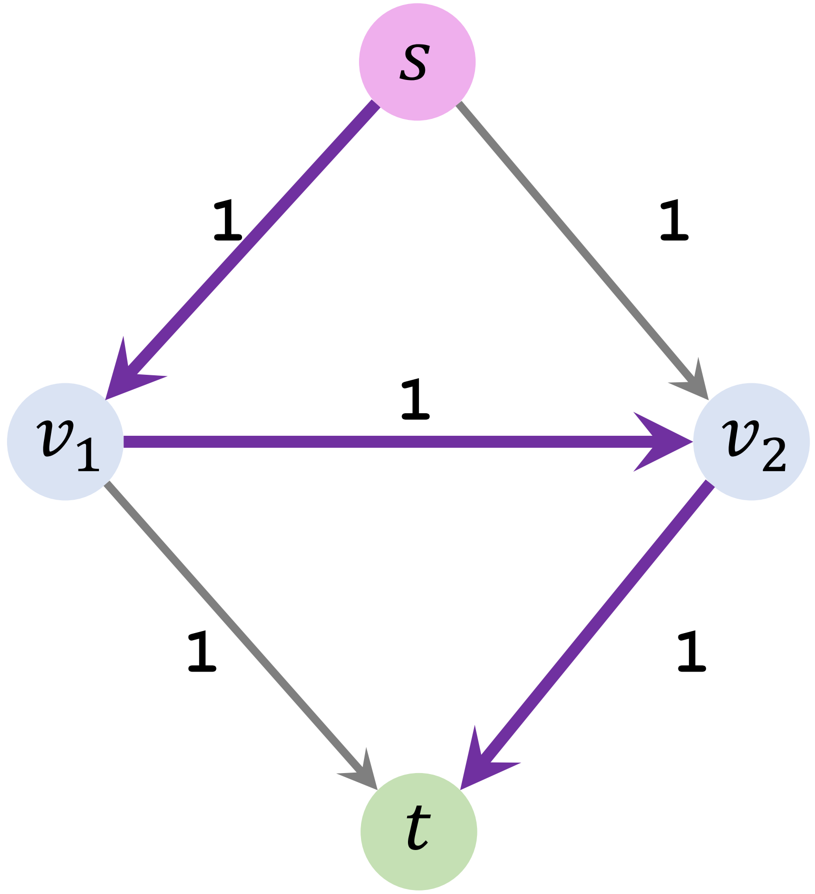
Problem with the naïve algorithm



Maximum Flow

- A selected path can be bad.
  - The amount of max flow is 2.

Problem with the naïve algorithm



Not Maximum Flow

- A selected path can be bad.
  - The amount of max flow is 2.
  - The amount of blocking flow is 2.

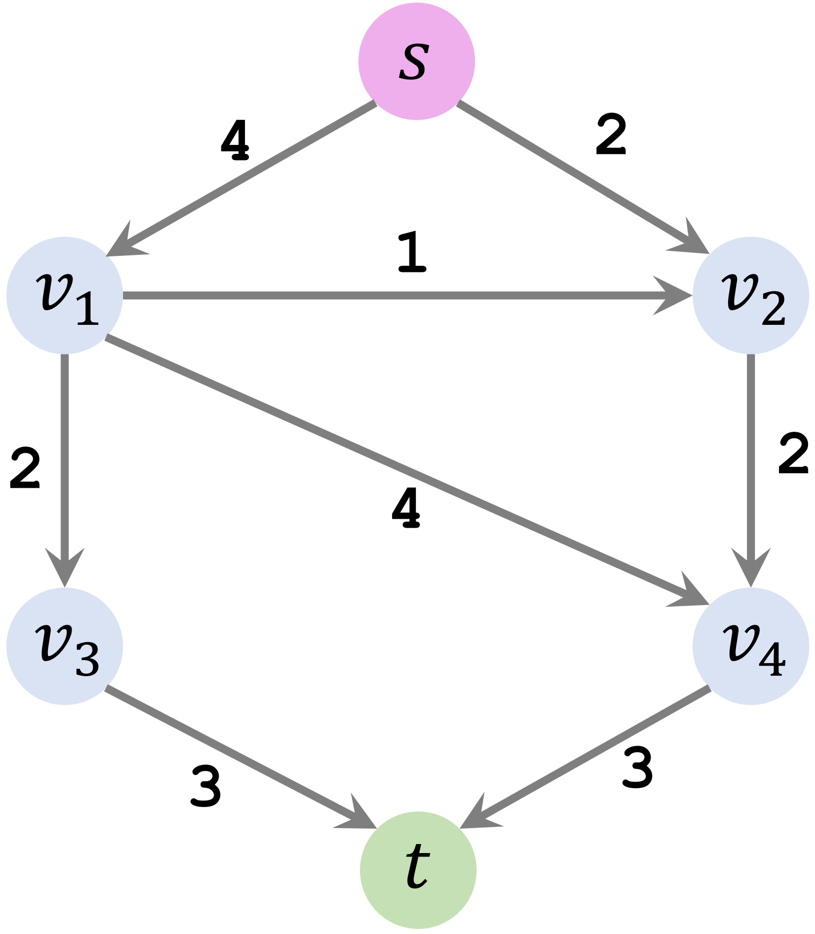
- Once a bad path is selected, the naïve algorithm cannot make corrections.

Ford-Fulkerson Algorithm

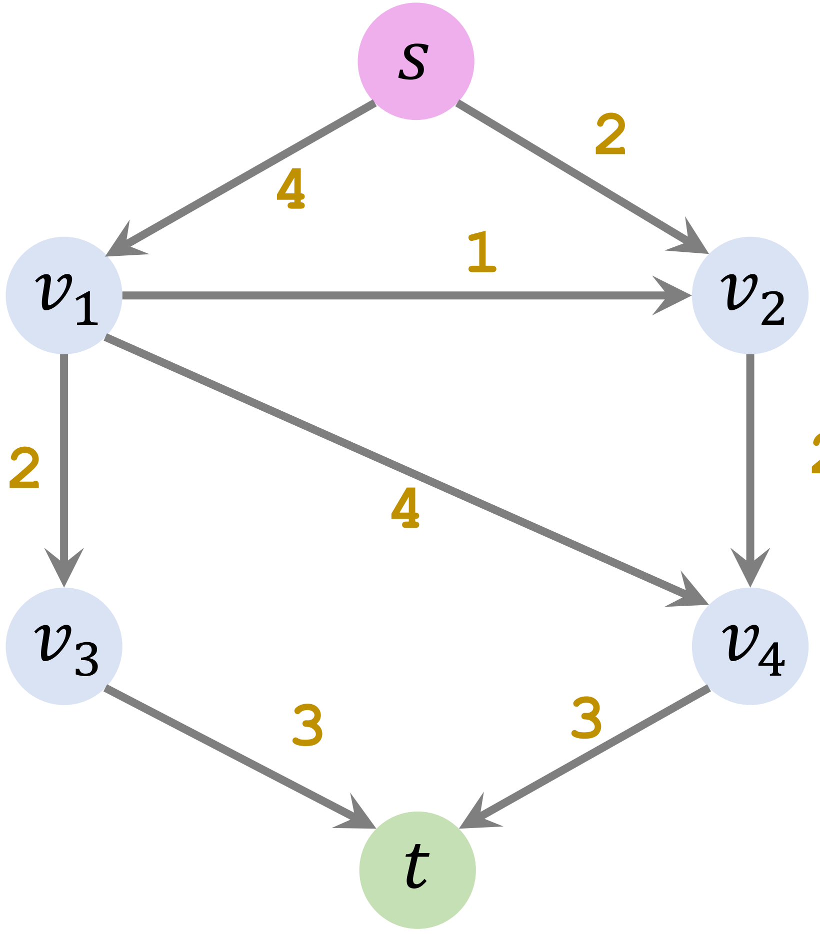
Reference

- L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8: 399–404, 1956.

Initialization

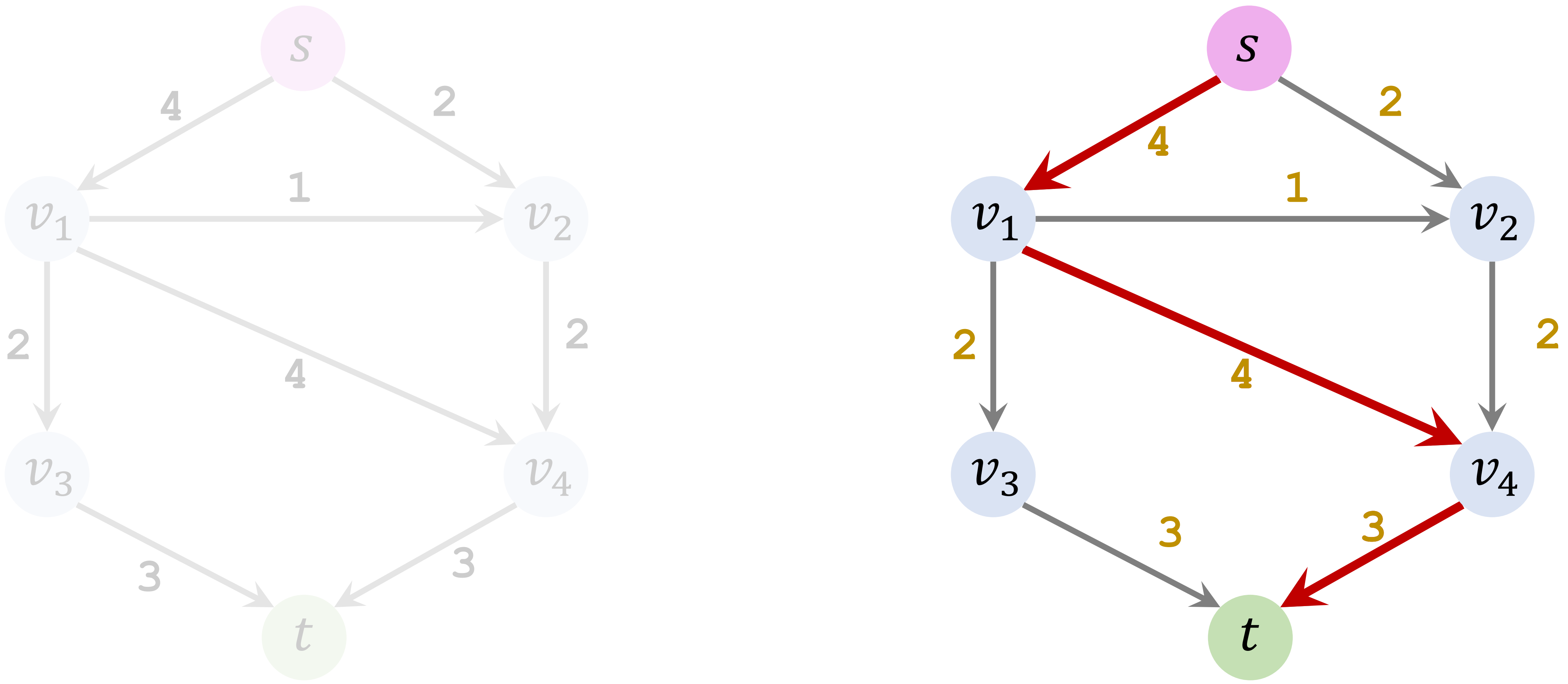


Original Graph



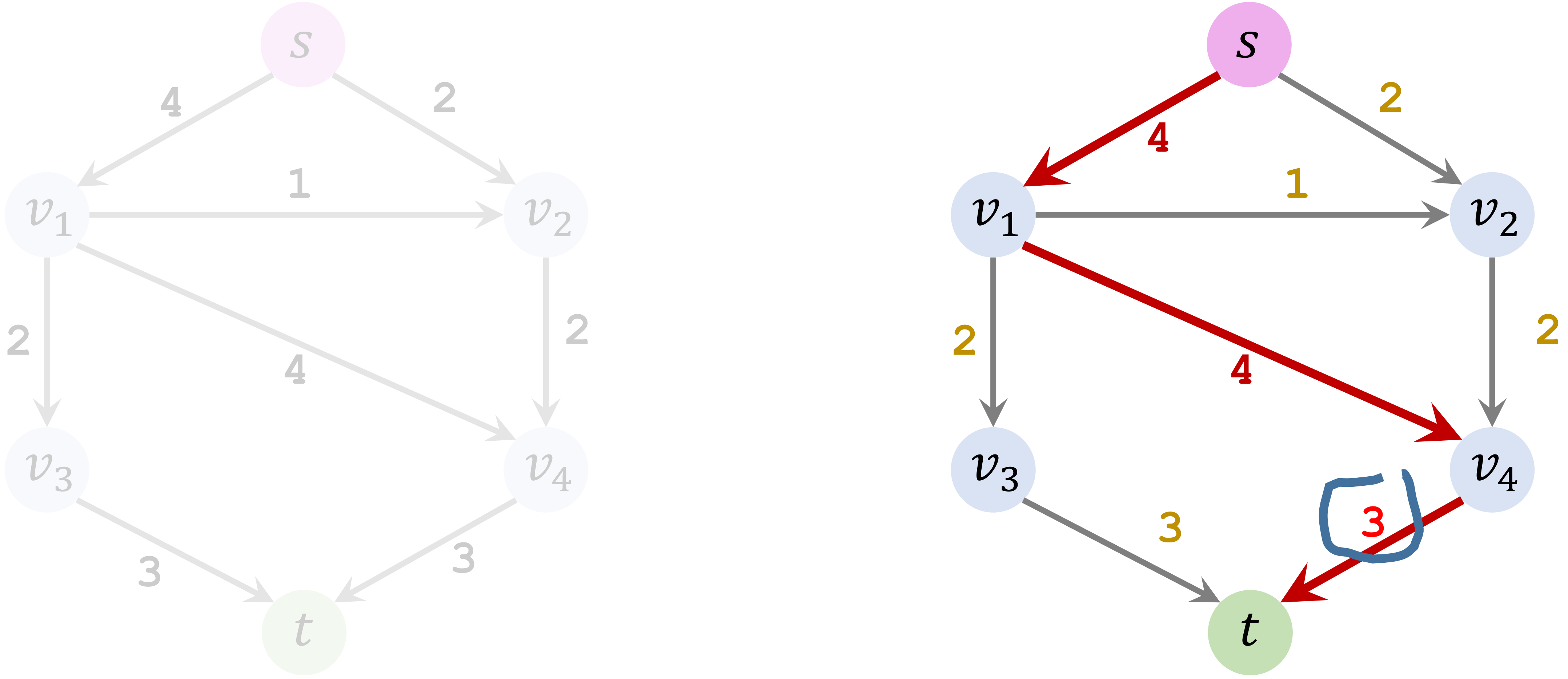
Residual Graph

Iteration 1: Find an augmenting path



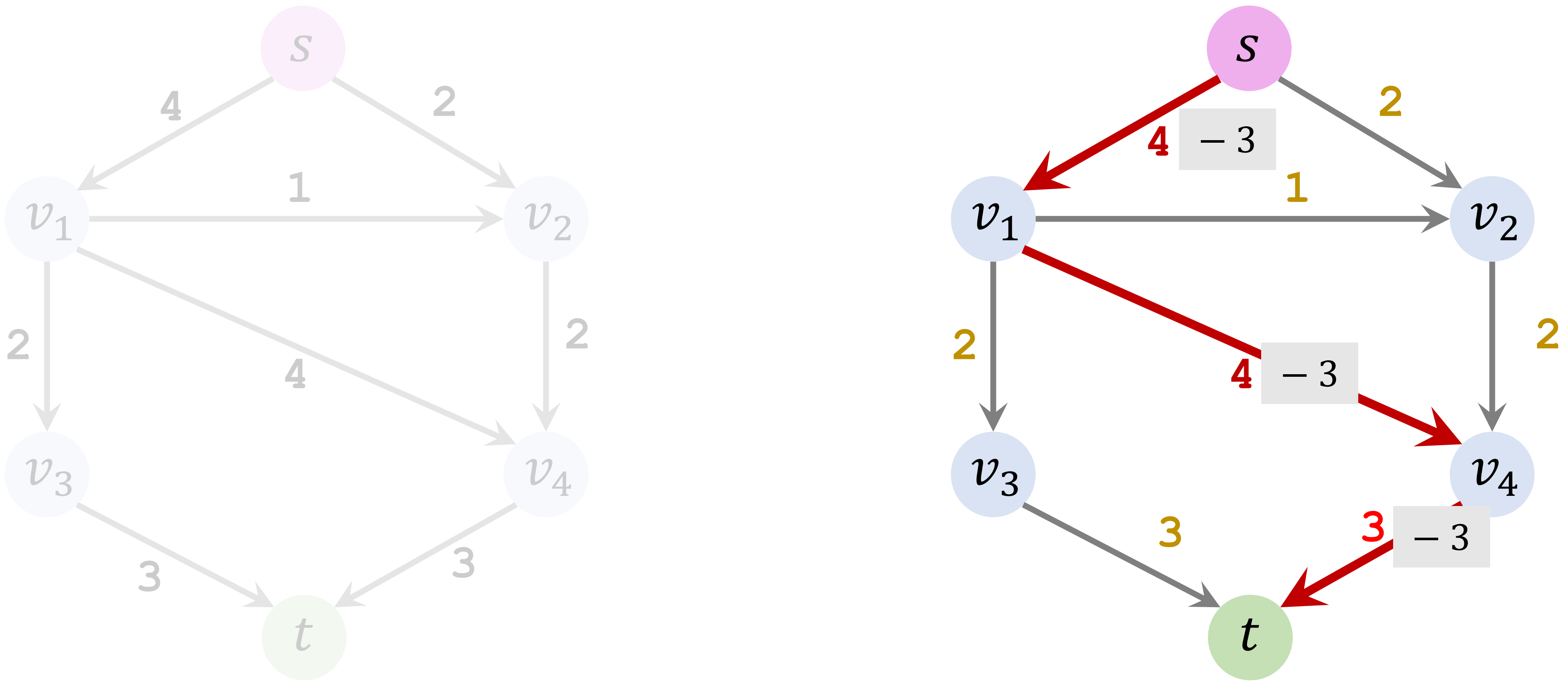
Found path $s \to v_1 \to v_4 \to t$.

Iteration 1: Find an augmenting path

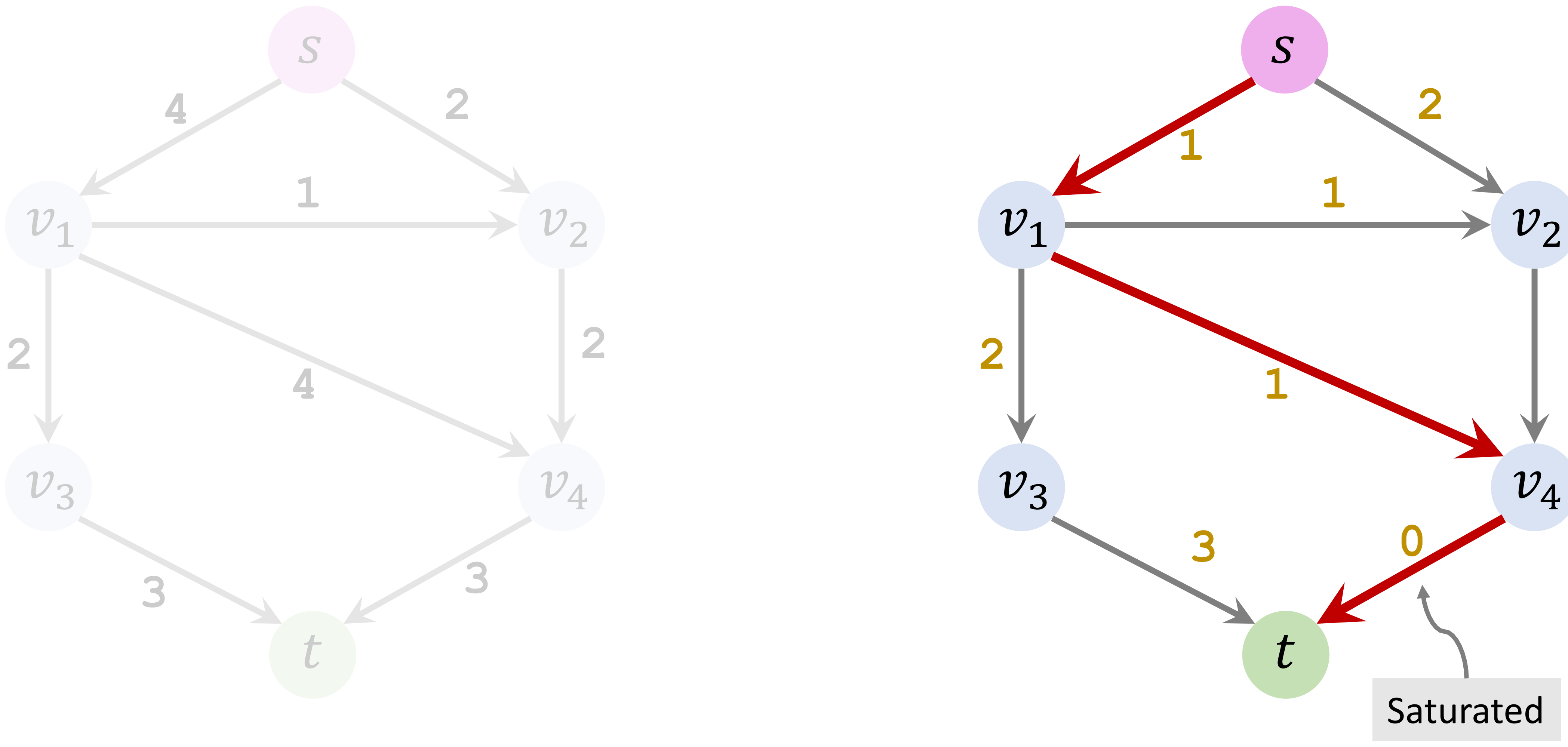


Found path $s \to v_1 \to v_4 \to t$. (Bottleneck capacity $= 3$.)

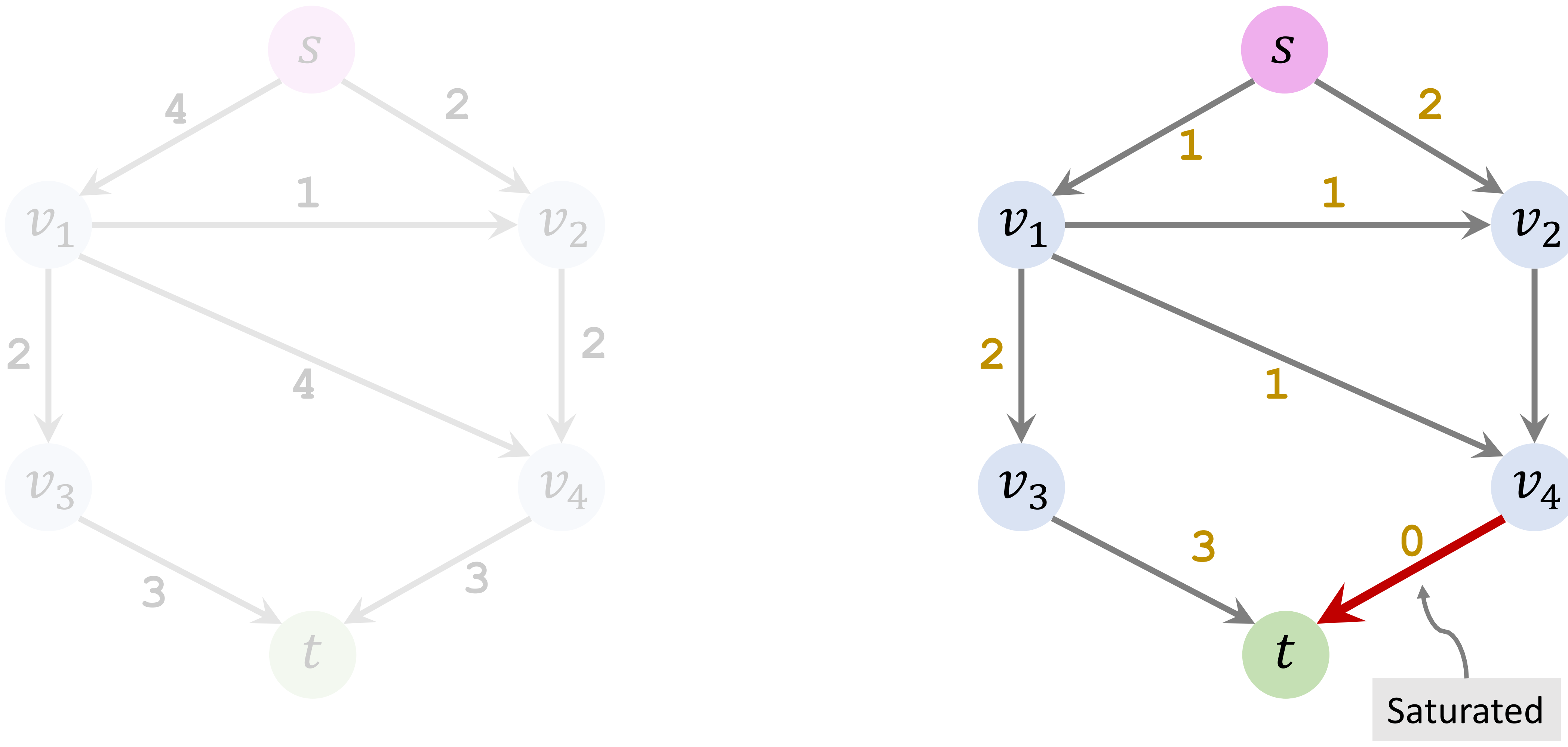
Iteration 1: Update residuals



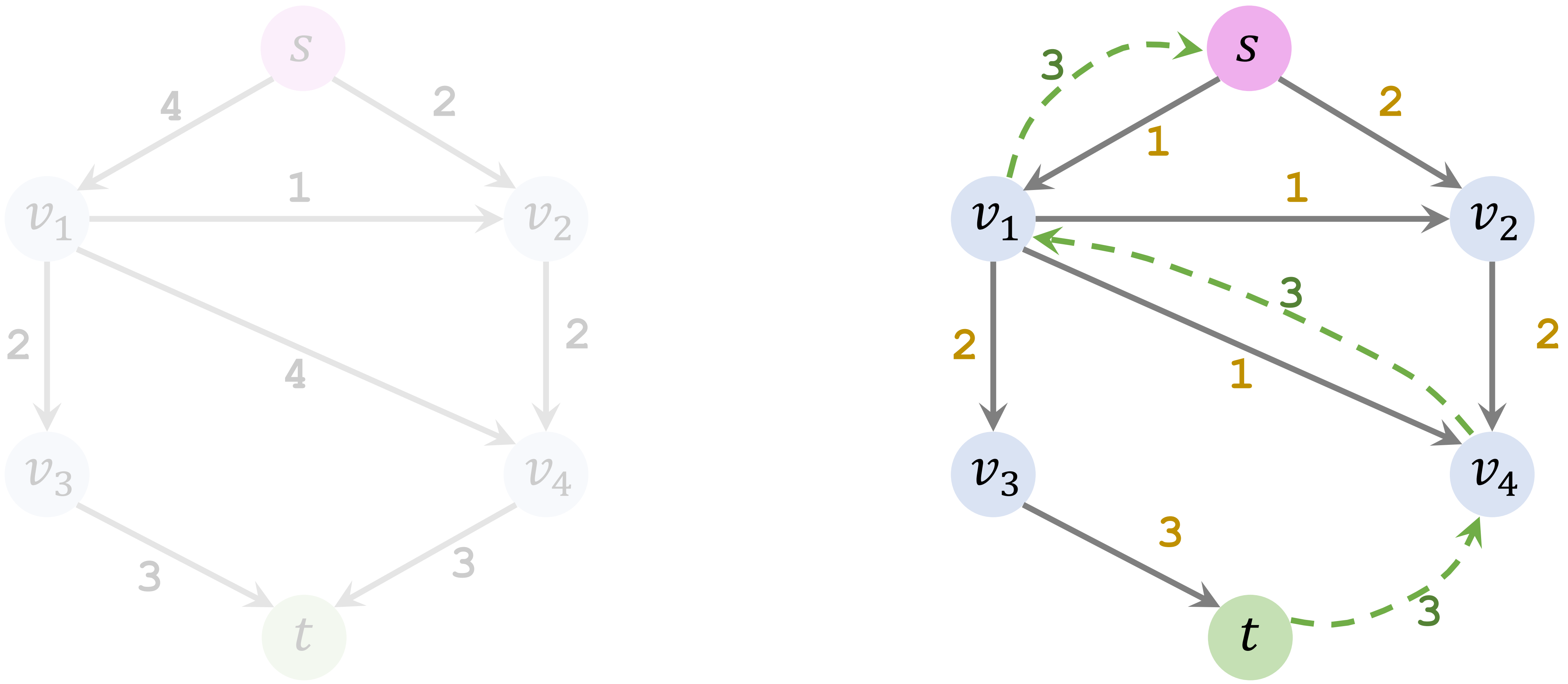
Iteration 1: Update residuals



Iteration 1: Remove saturated edges

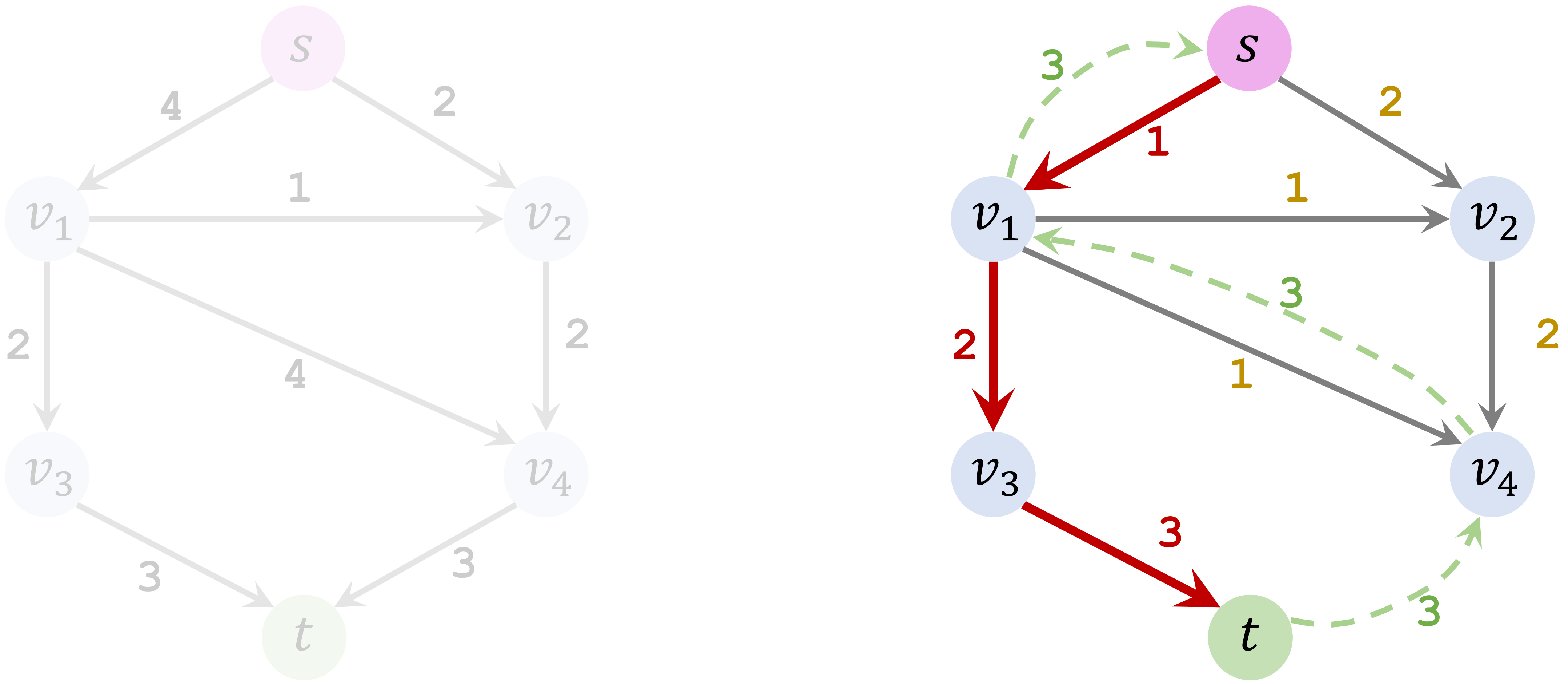


Iteration 1: Add a backward path



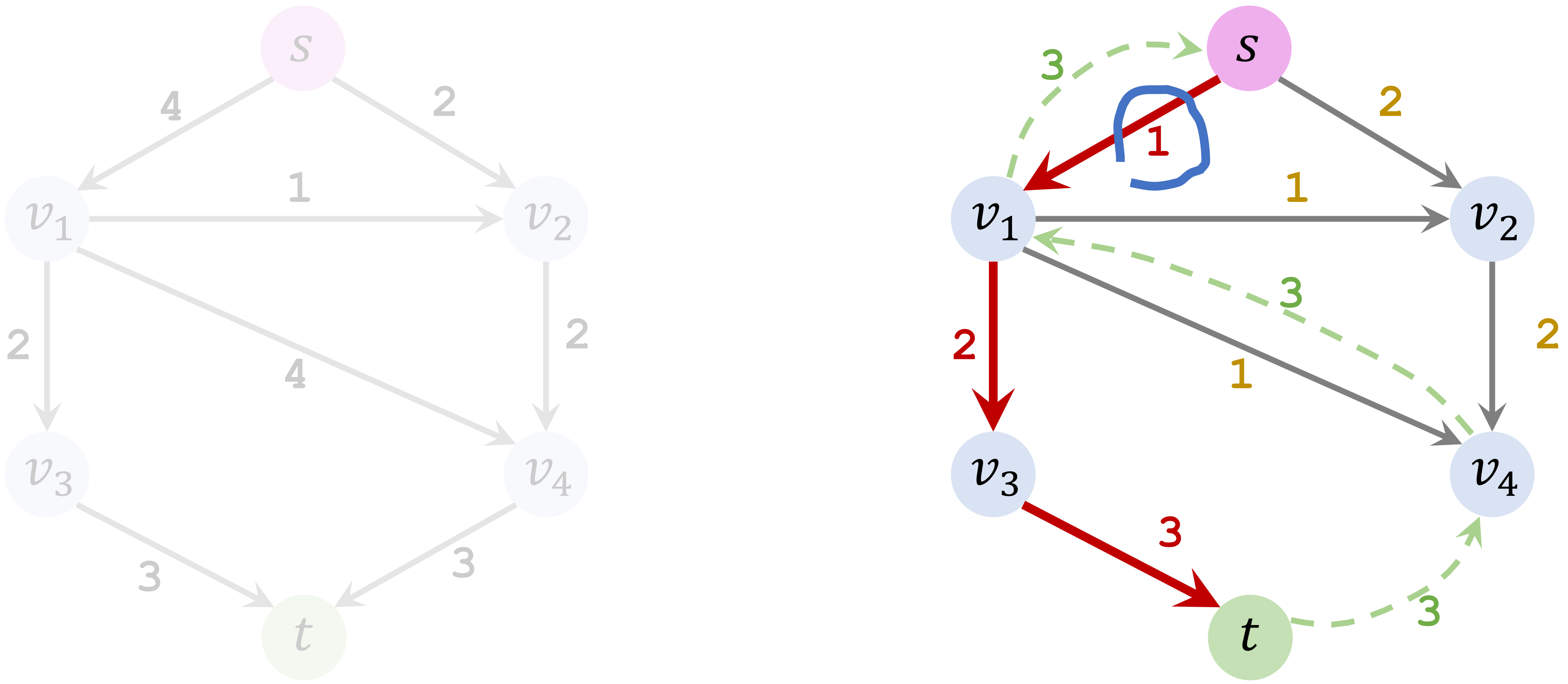
Add path $t \rightarrow v_4 \rightarrow v_1 \rightarrow s$ with capacity $= 3$. (Allow "undoing".)

Iteration 2: Find an augmenting path



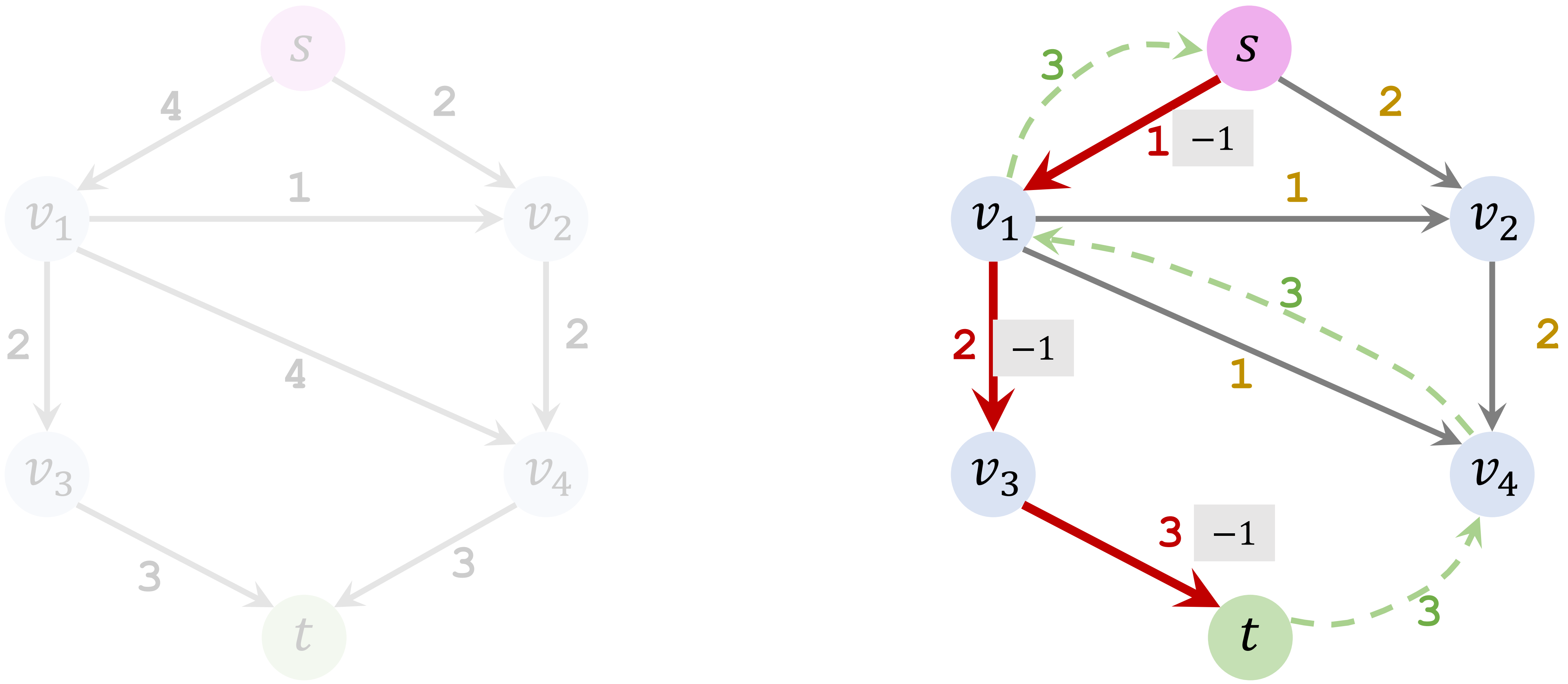
Found path $s \to v_1 \to v_3 \to t$.

Iteration 2: Find an augmenting path

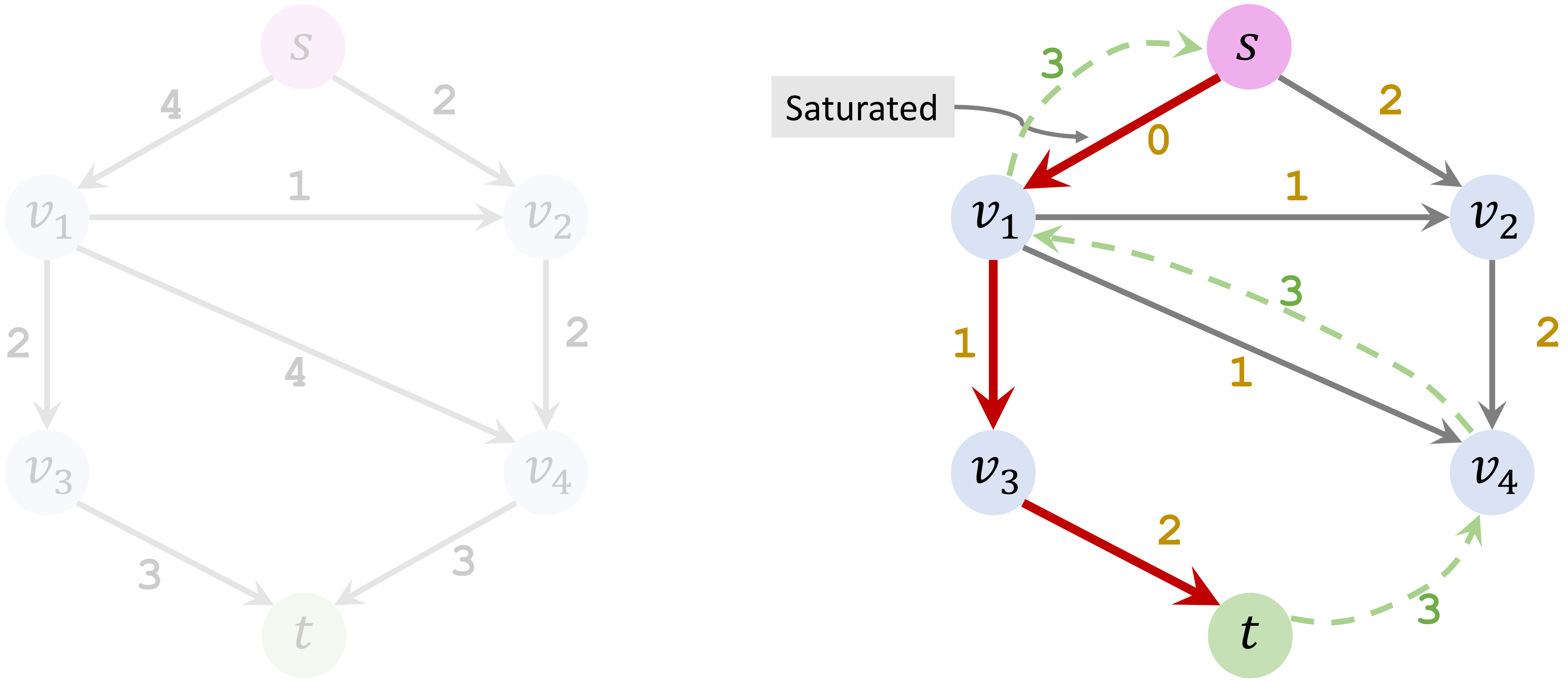


Found path $s \to v_1 \to v_3 \to t$. (Bottleneck capacity = 1.)

Iteration 2: Update residuals

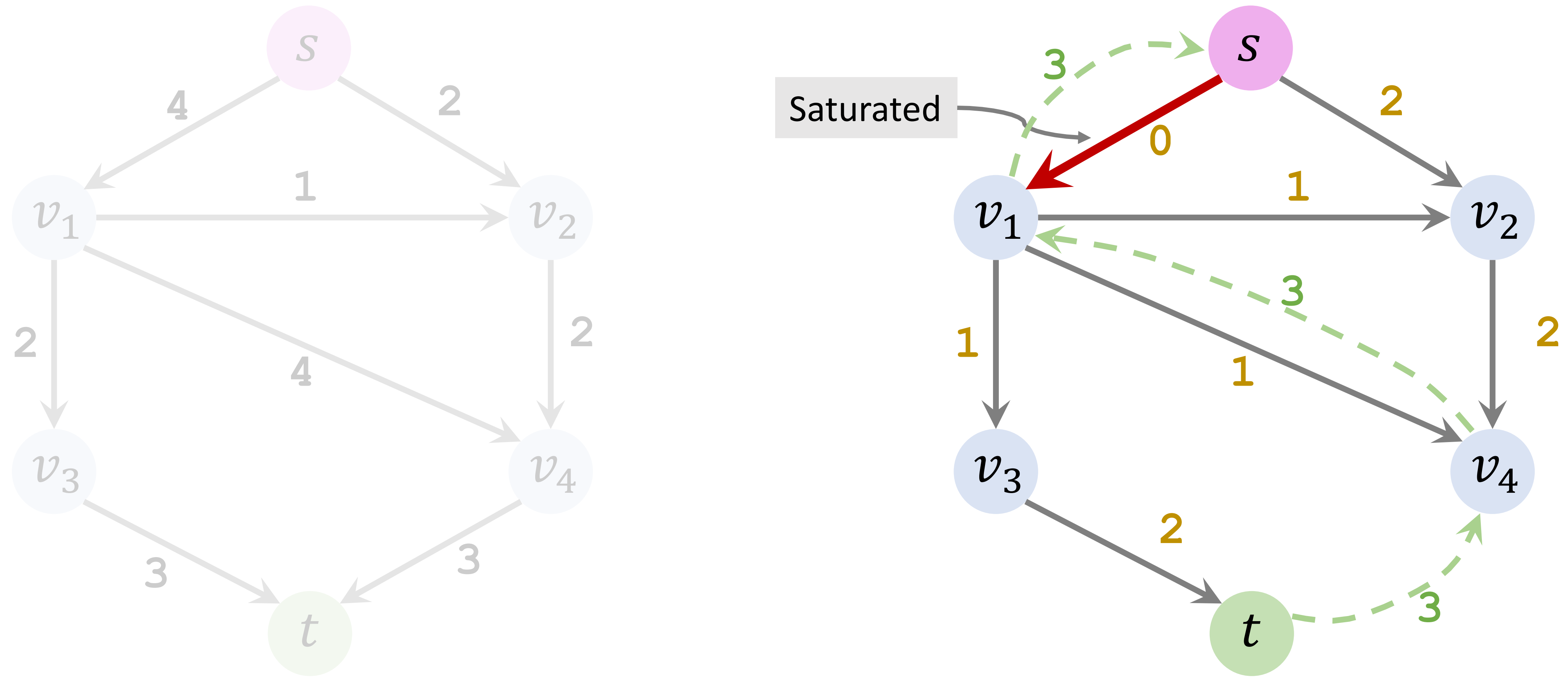


Iteration 2: Update residuals

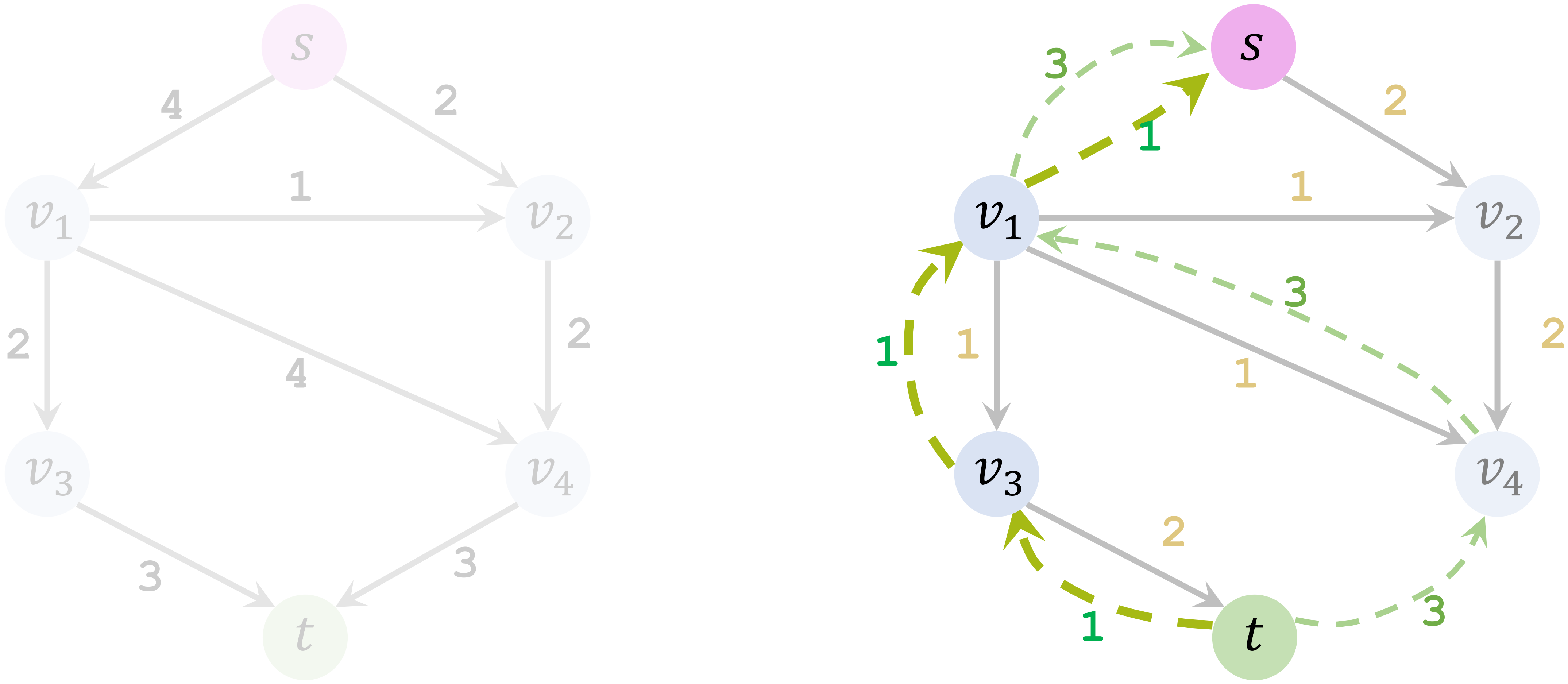


Found path $s \to v_1 \to v_3 \to t$. (Bottleneck capacity = 1.)

Iteration 2: Remove saturated edges

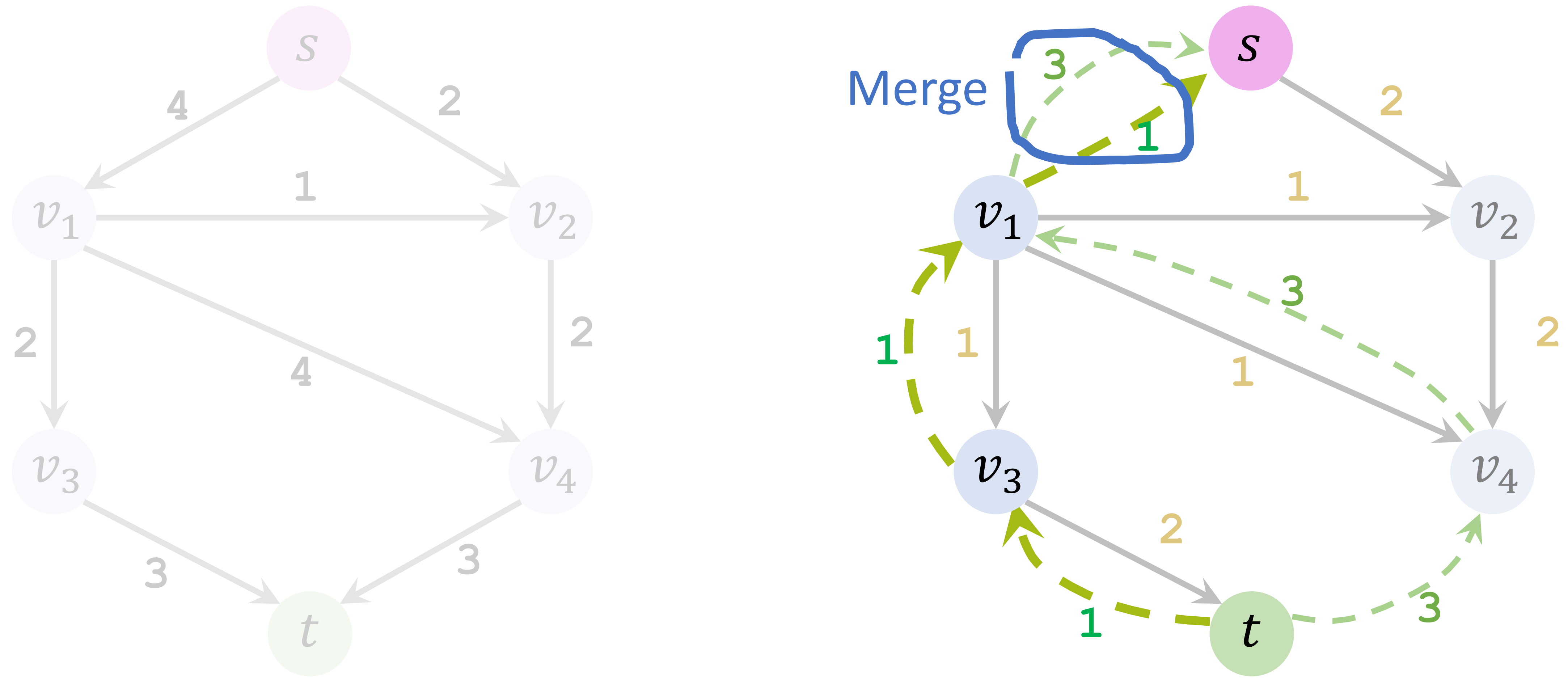


Iteration 2: Add a backward path



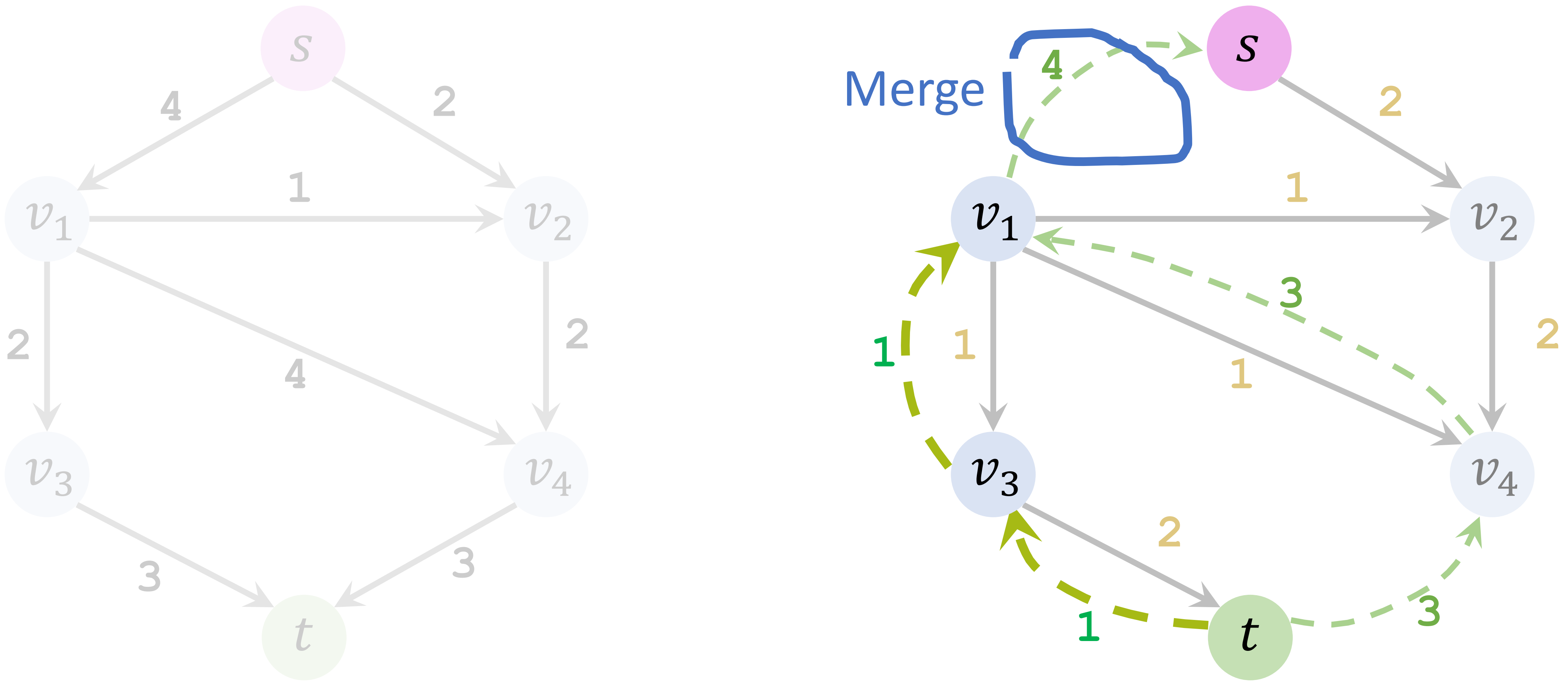
Add path $t \rightarrow v_3 \rightarrow v_1 \rightarrow s$ with capacity $= 1$.

Iteration 2: Add a backward path



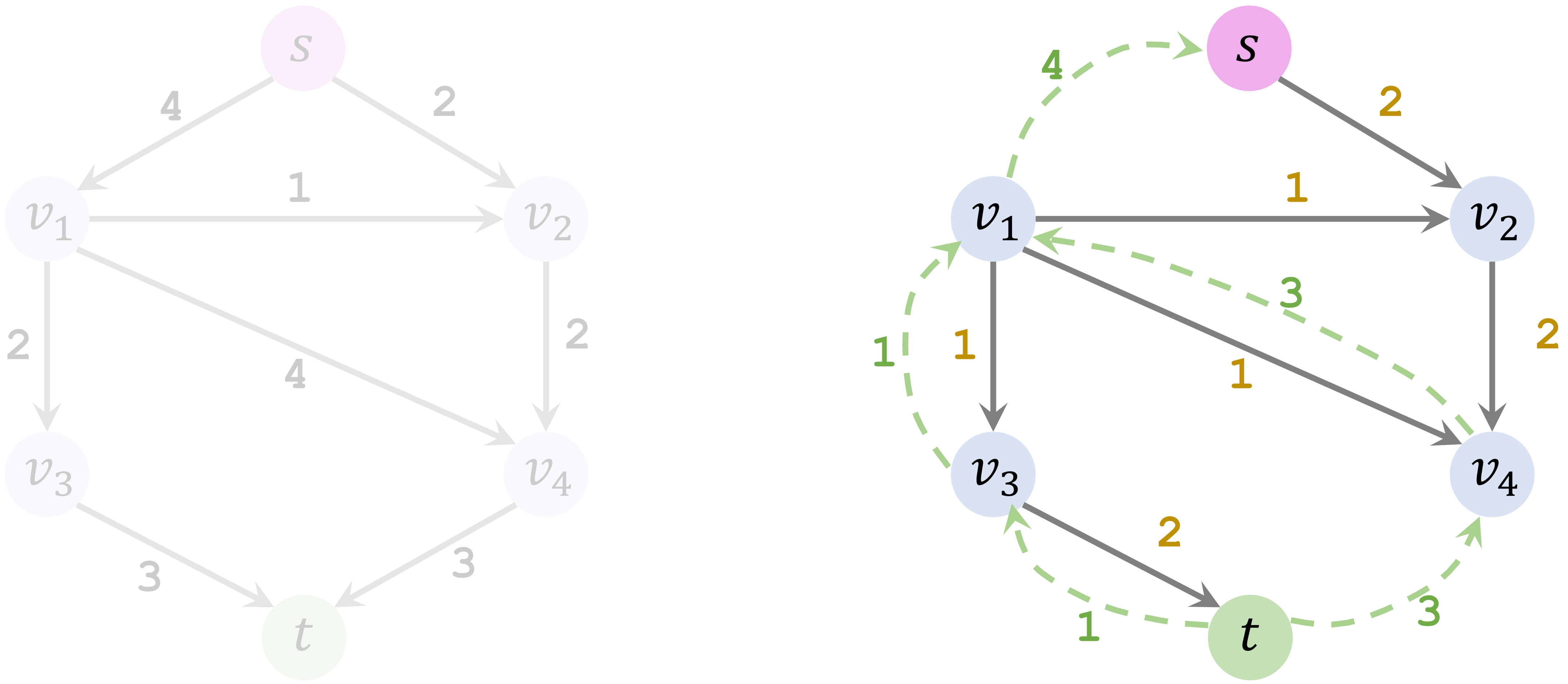
Add path $t \rightarrow v_3 \rightarrow v_1 \rightarrow s$ with capacity $= 1$.

Iteration 2: Add a backward path

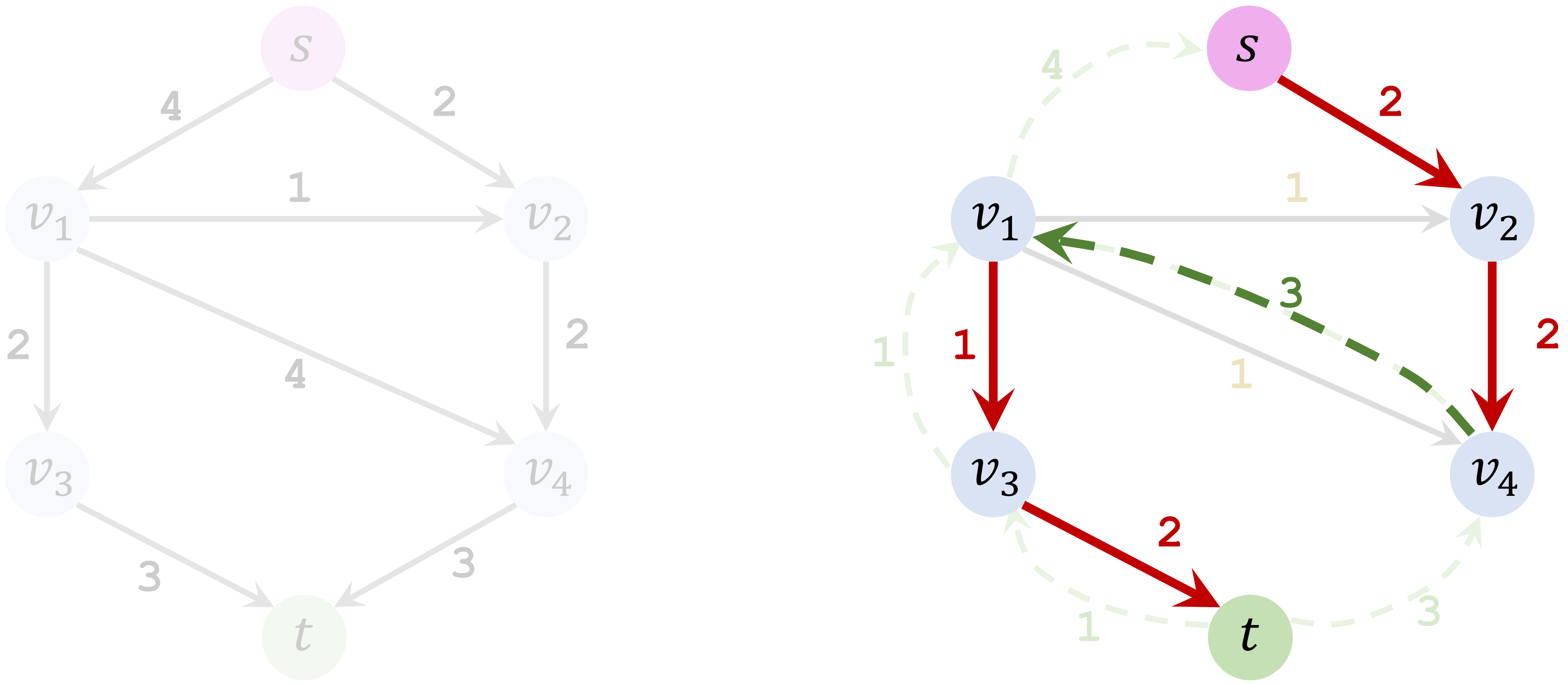


Add path $t \to v_3 \to v_1 \to s$ with capacity $= 1$.

Iteration 3: Find an augmenting path

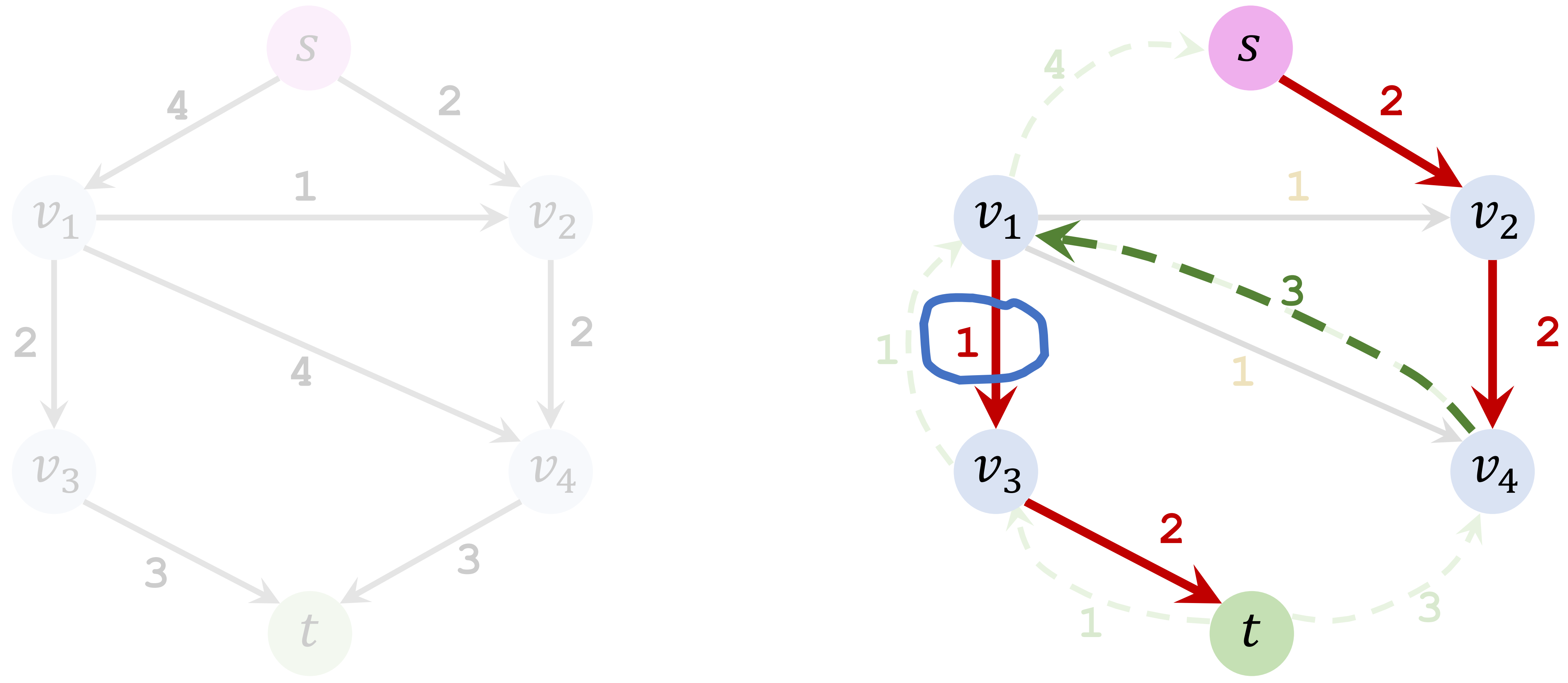


Iteration 3: Find an augmenting path



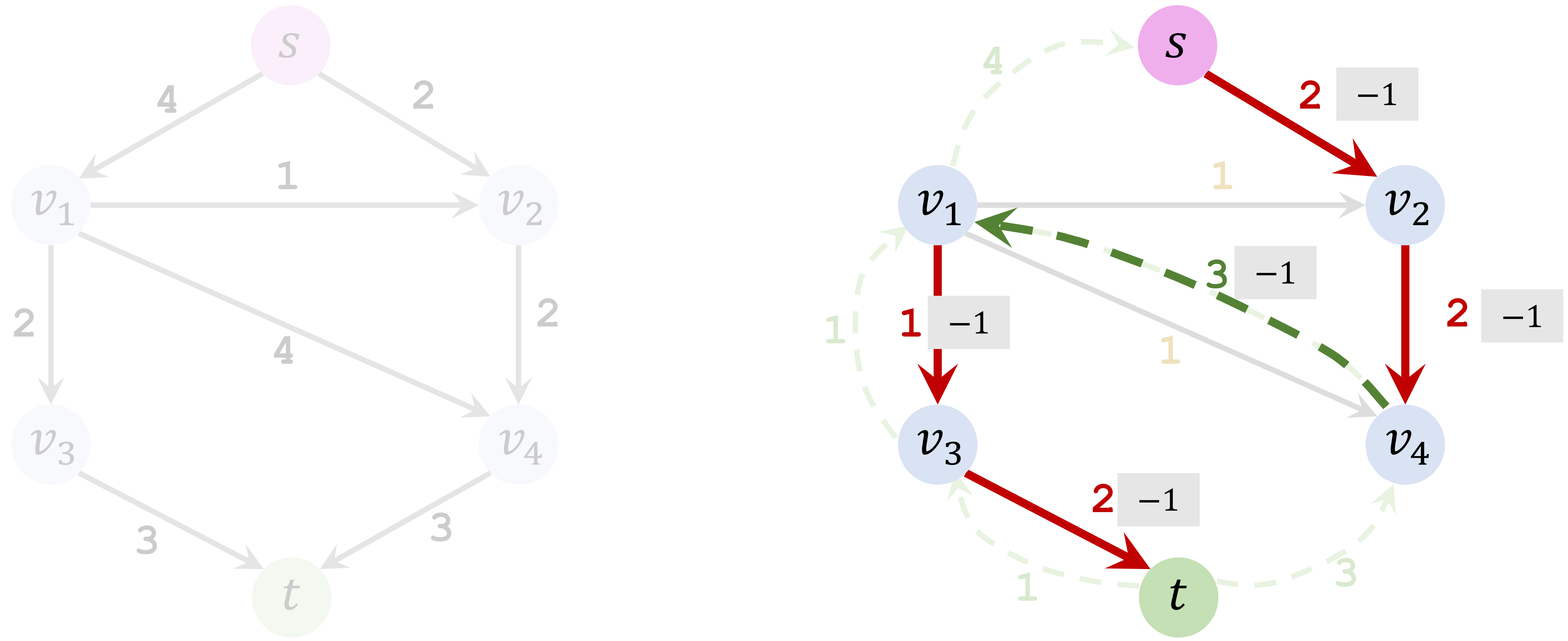
Found path $s \to v_2 \to v_4 \to v_1 \to v_3 \to t$.

Iteration 3: Find an augmenting path

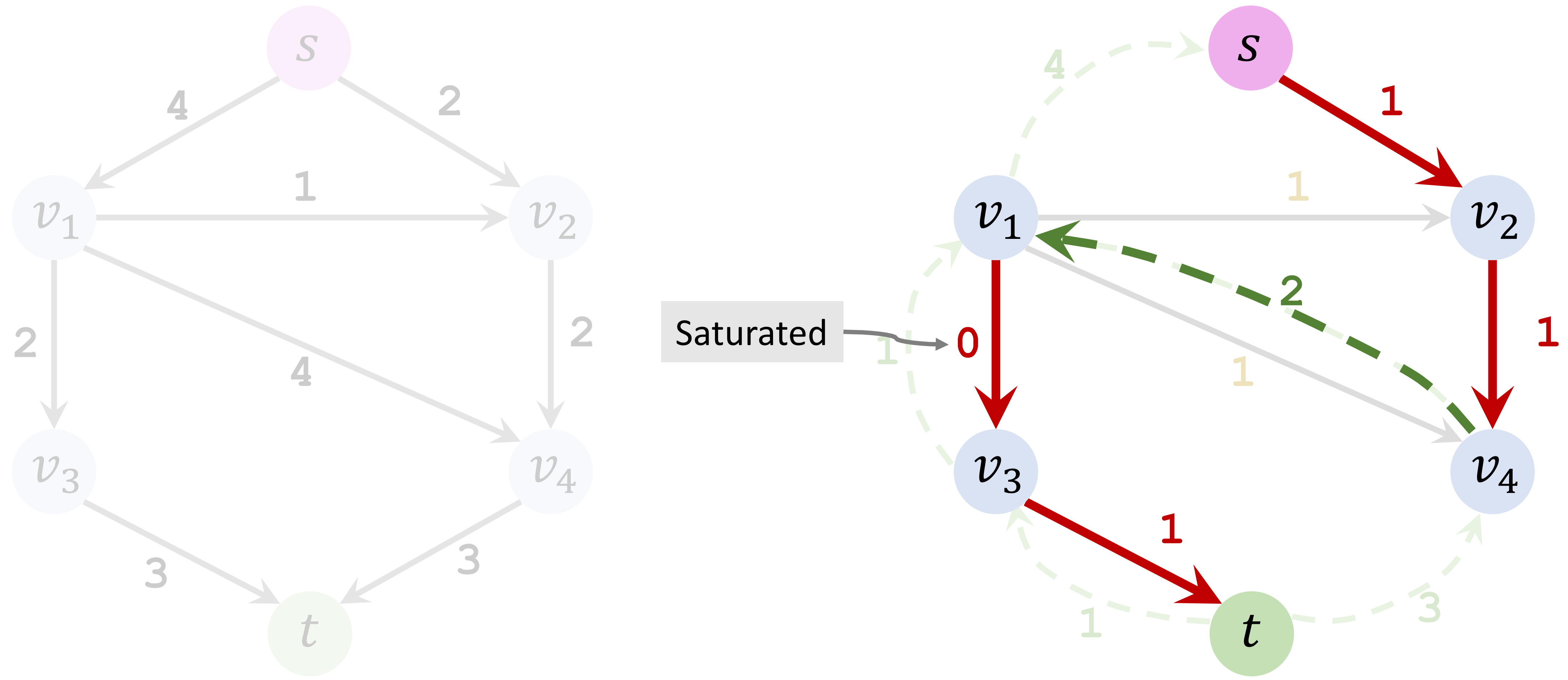


Found path $s \to v_2 \to v_4 \to v_1 \to v_3 \to t$. (Bottleneck capacity = 1.)

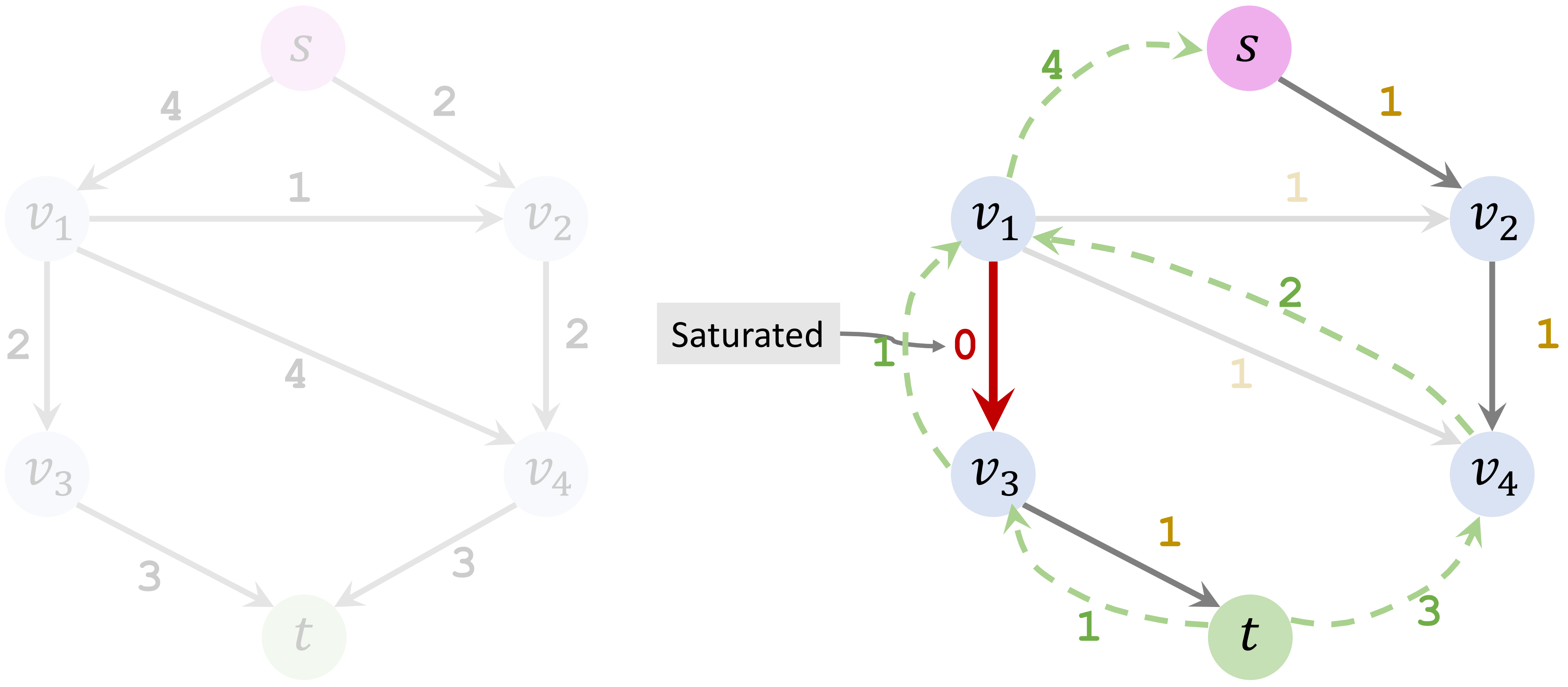
Iteration 3: Update residuals



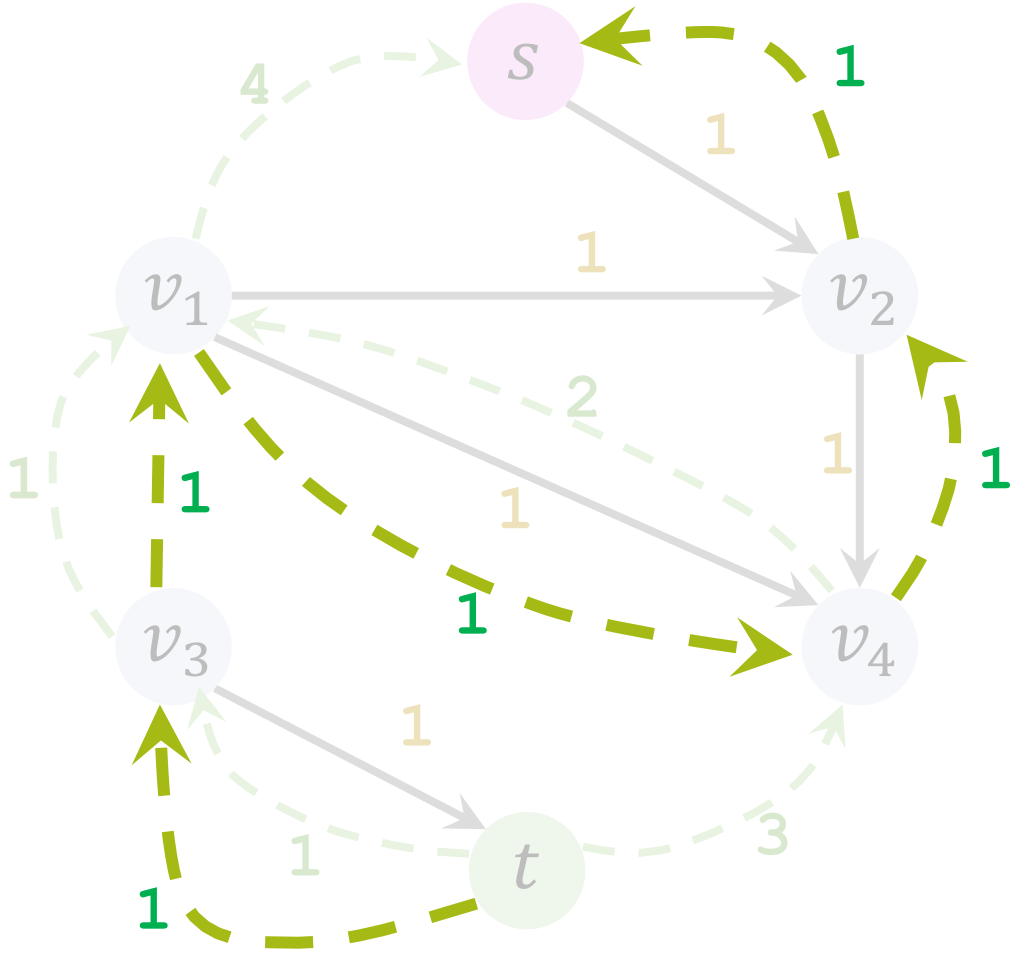
Iteration 3: Update residuals



Iteration 3: Remove saturated edges

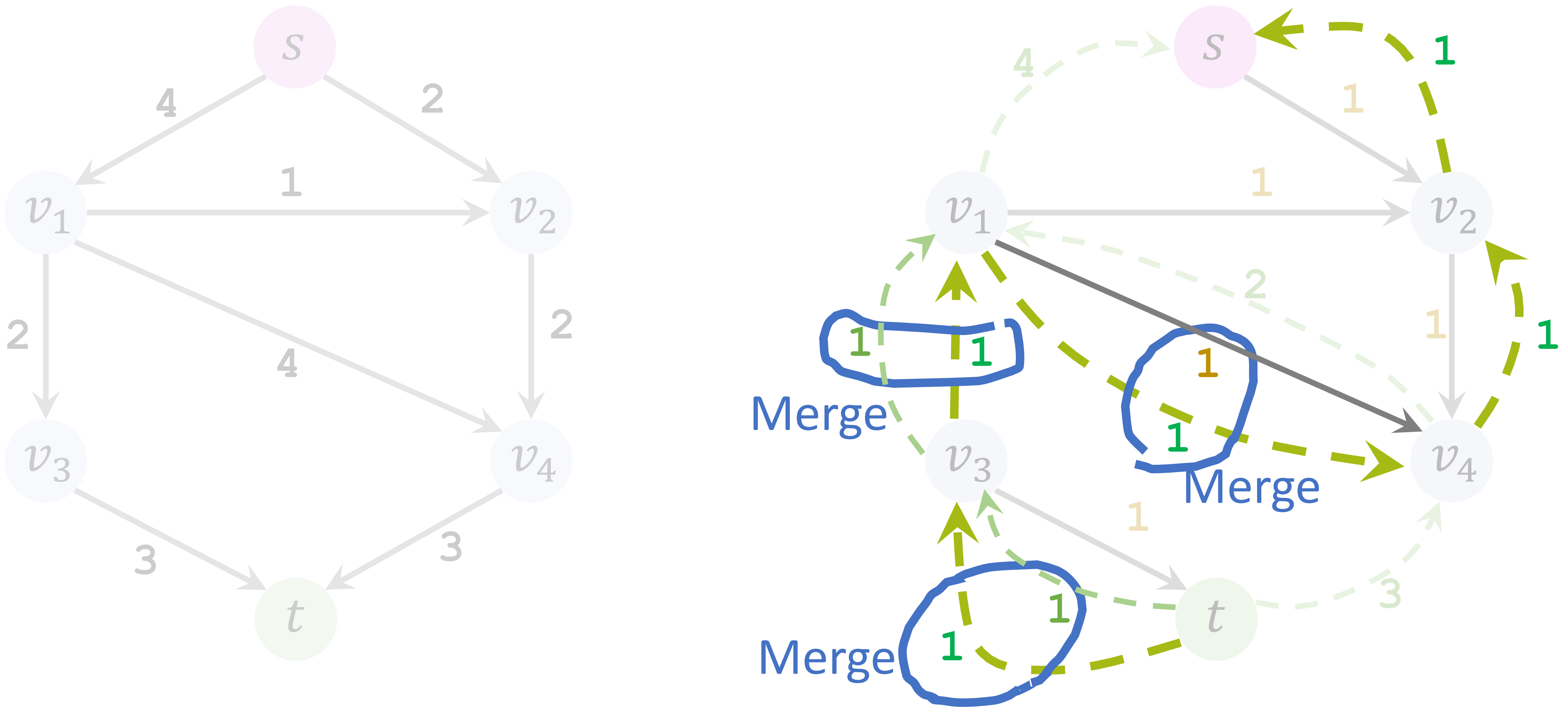


Iteration 3: Add a backward path

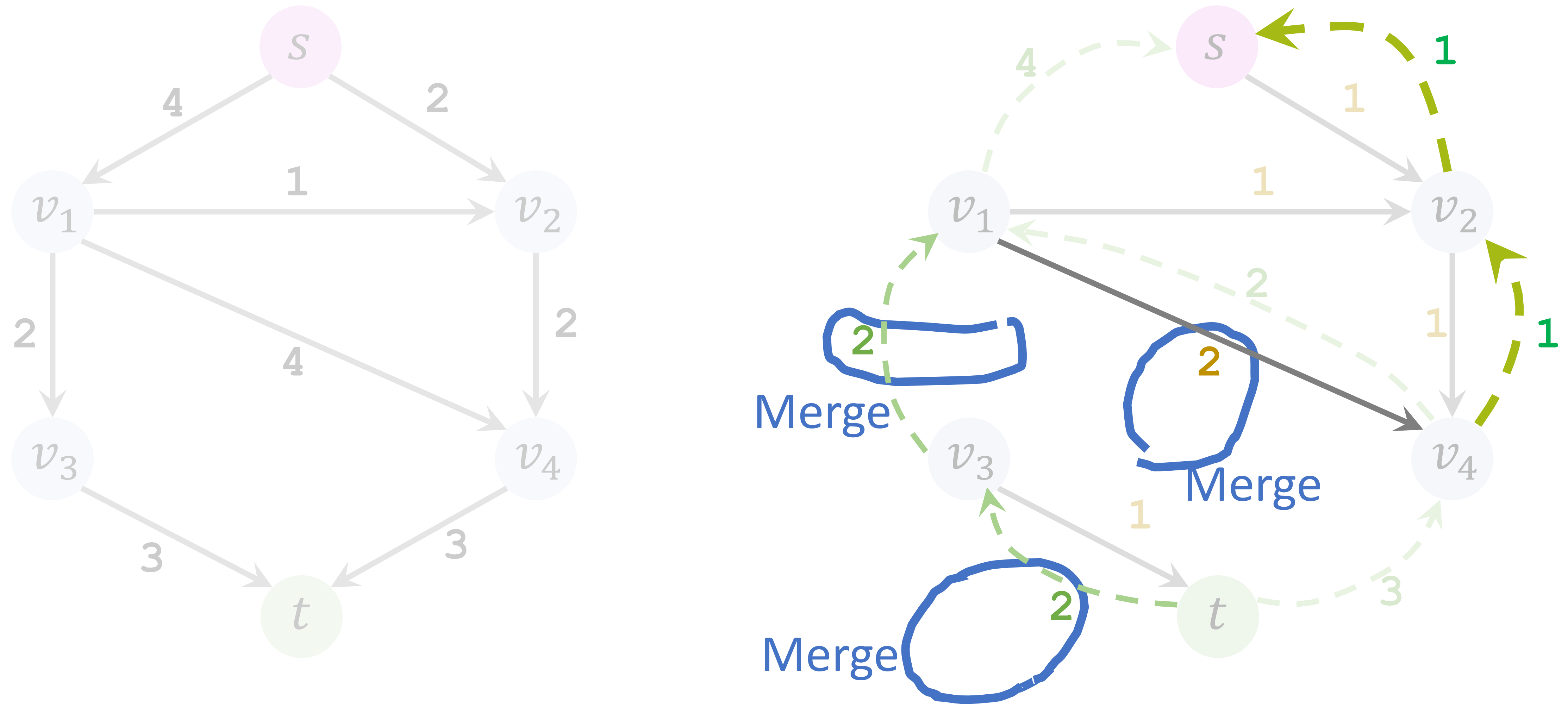


Add backward path $t \to v_3 \to v_1 \to v_4 \to v_2 \to s$ with capacity $= 1$.

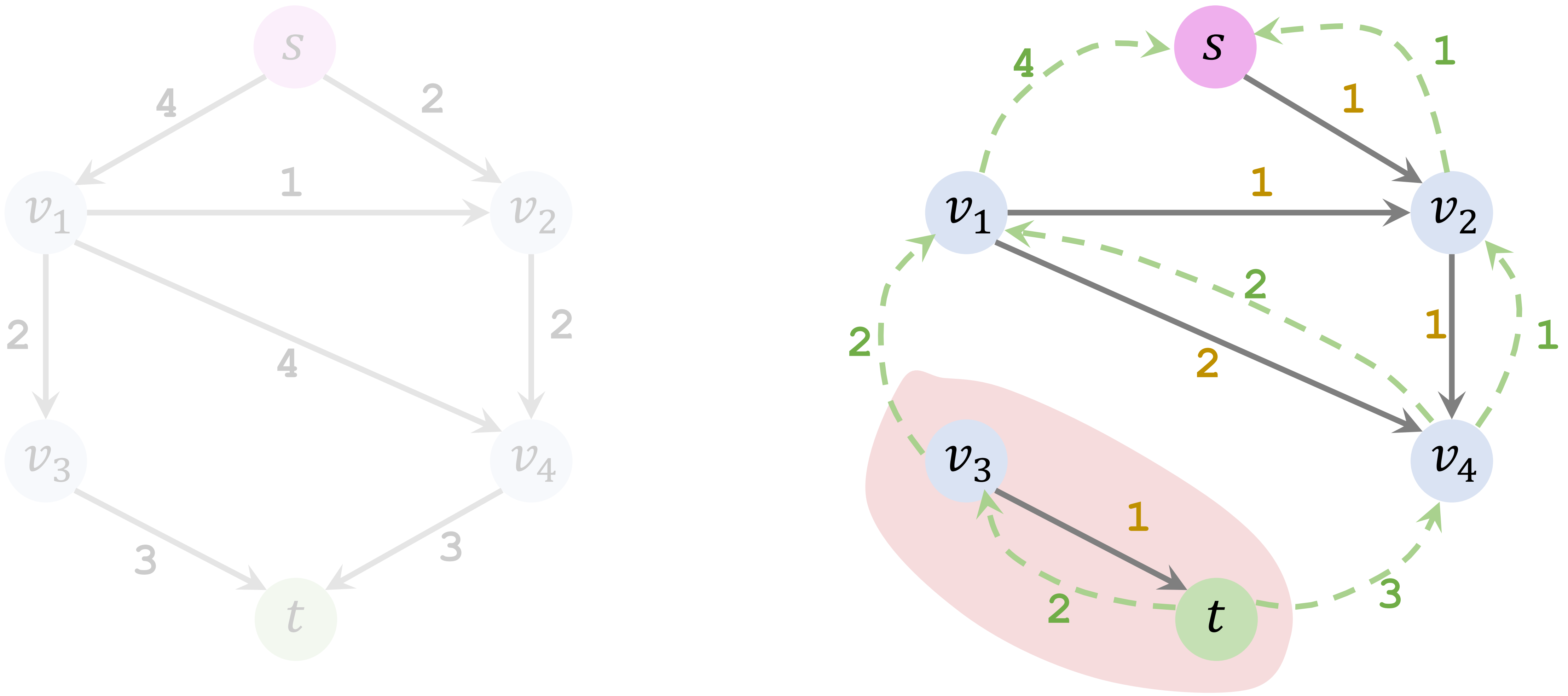
Iteration 3: Add a backward path



Iteration 3: Add a backward path

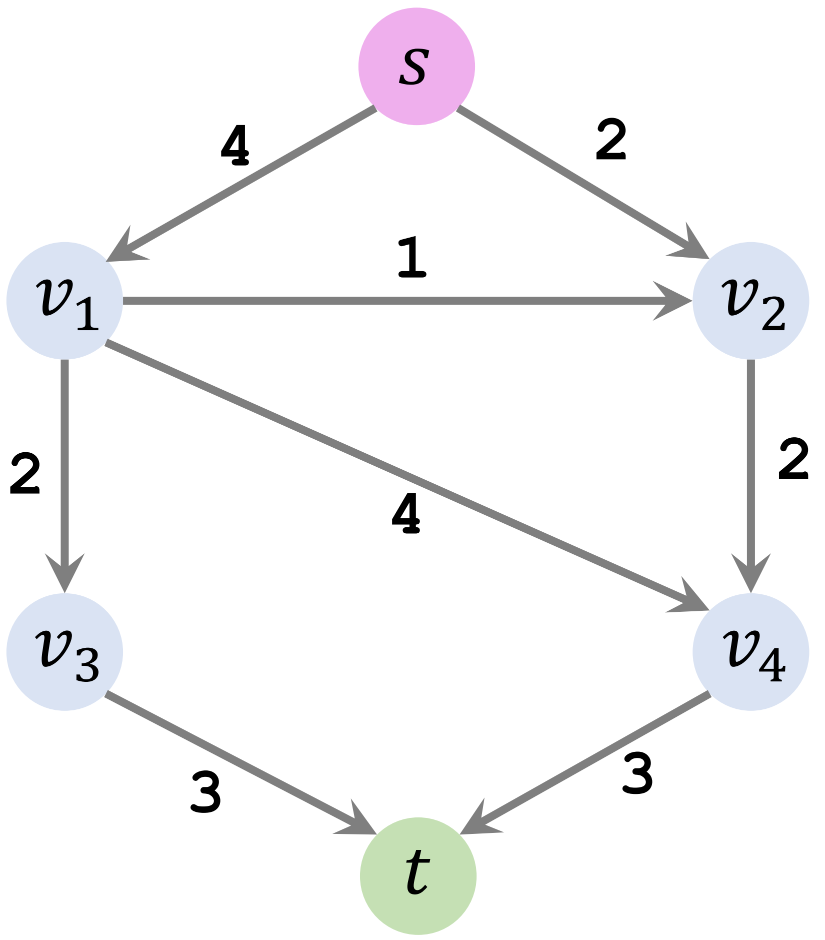


Iteration 4: Find an augmenting path

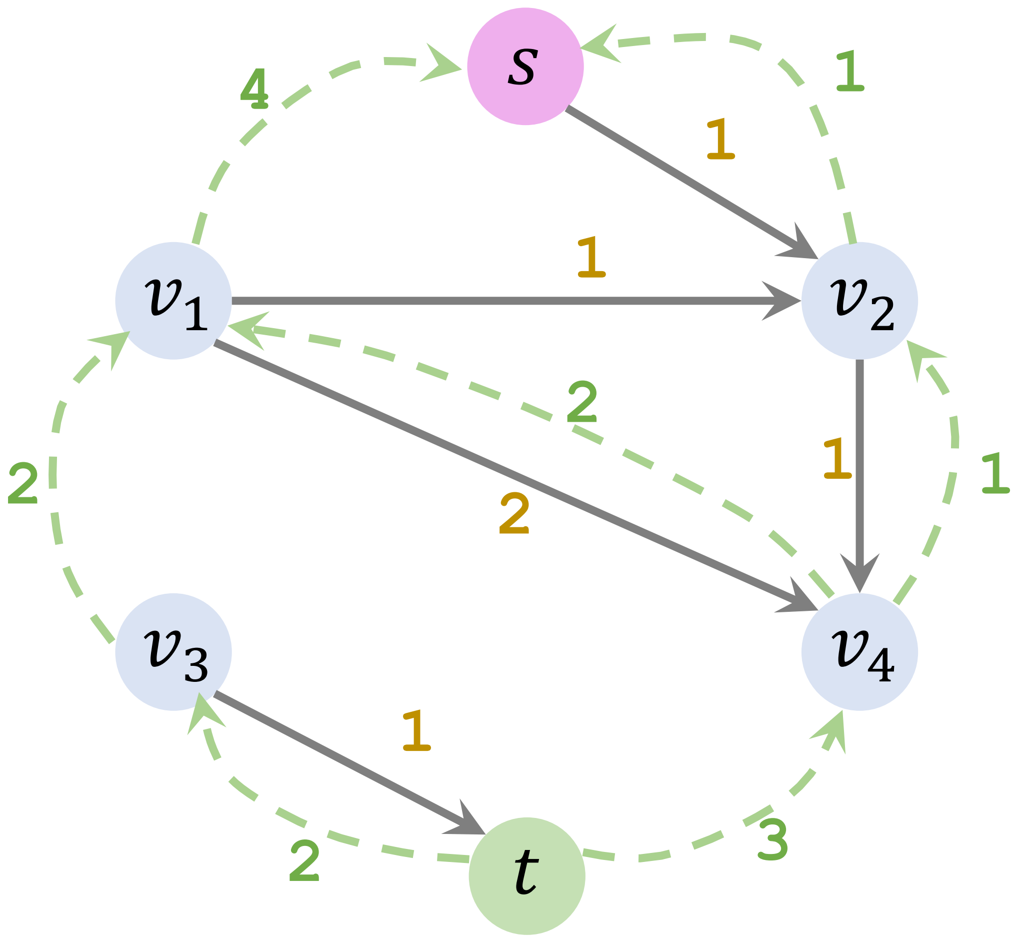


Cannot find any path from source to sink.

End of Procedure

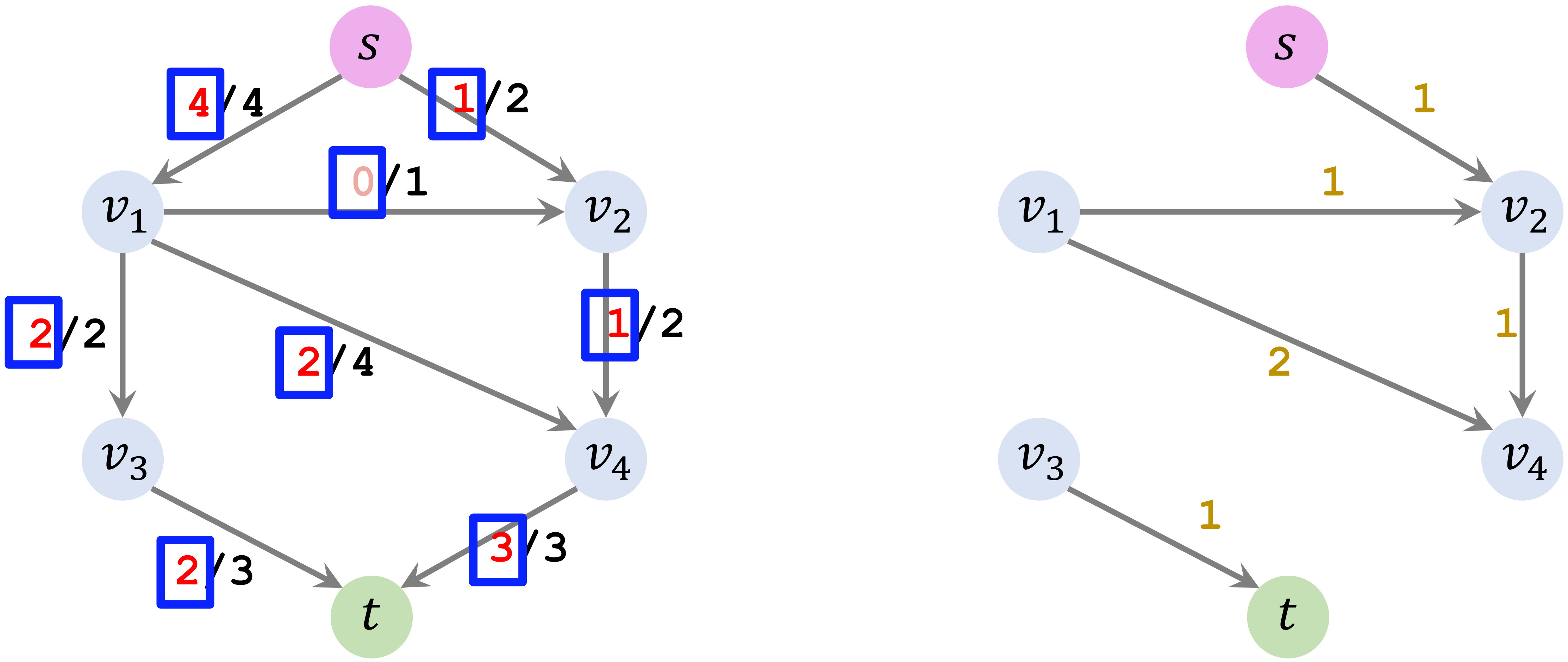


Original Graph



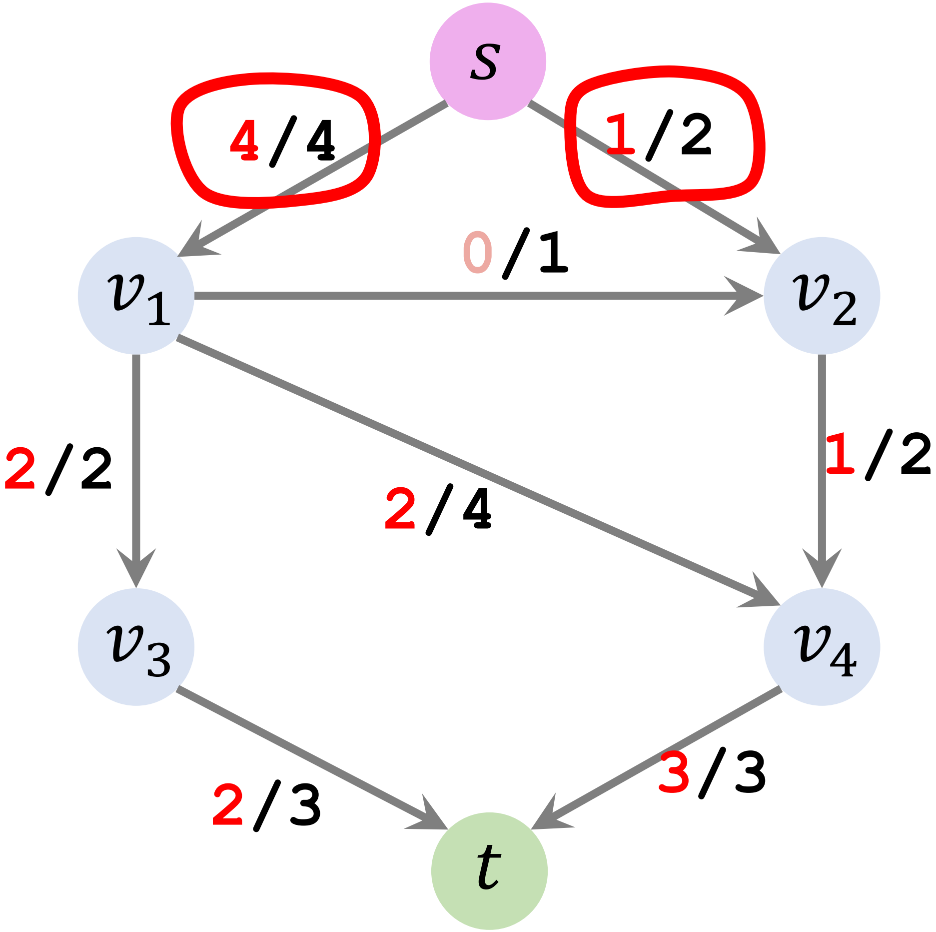
Residual Graph

End of Procedure



Flow = Capacity − Residual.

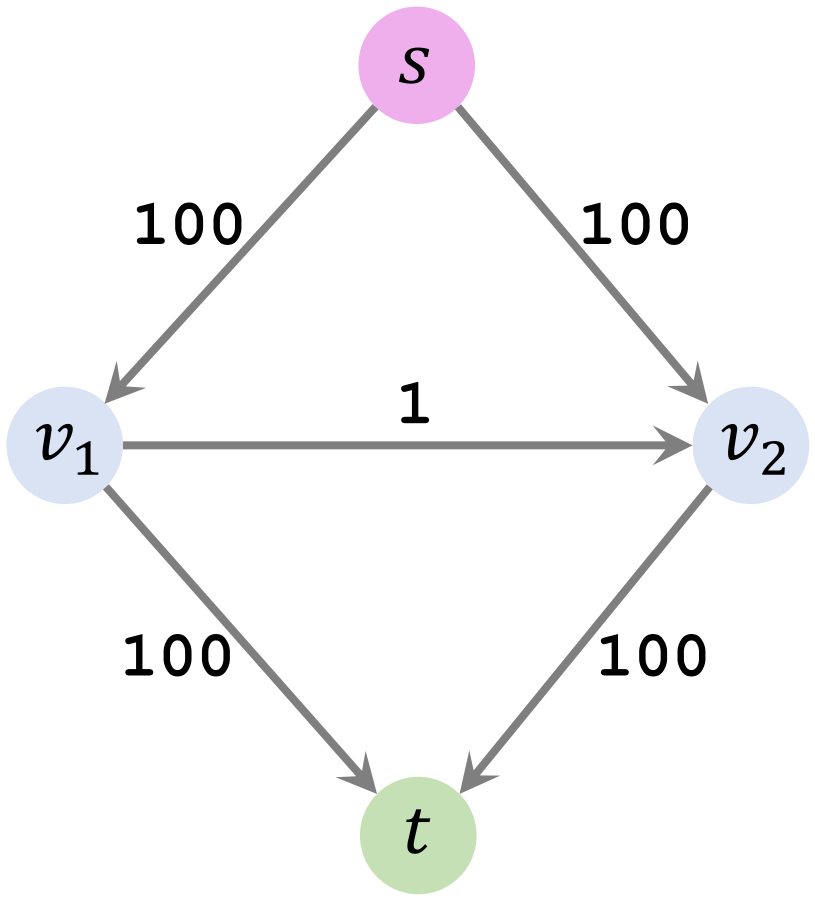
End of Procedure



Max Flow = 5. (Why? The flow leaving the source sum to 5.)

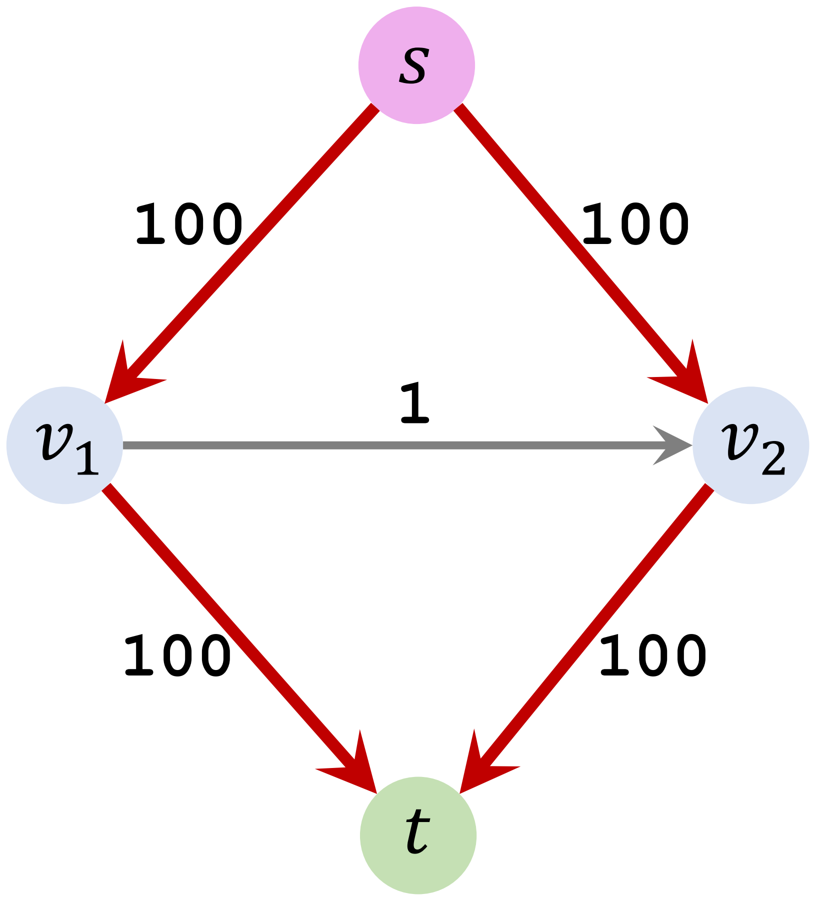
Worst-Case Time Complexity

A bad case for Ford-Fulkerson algorithm



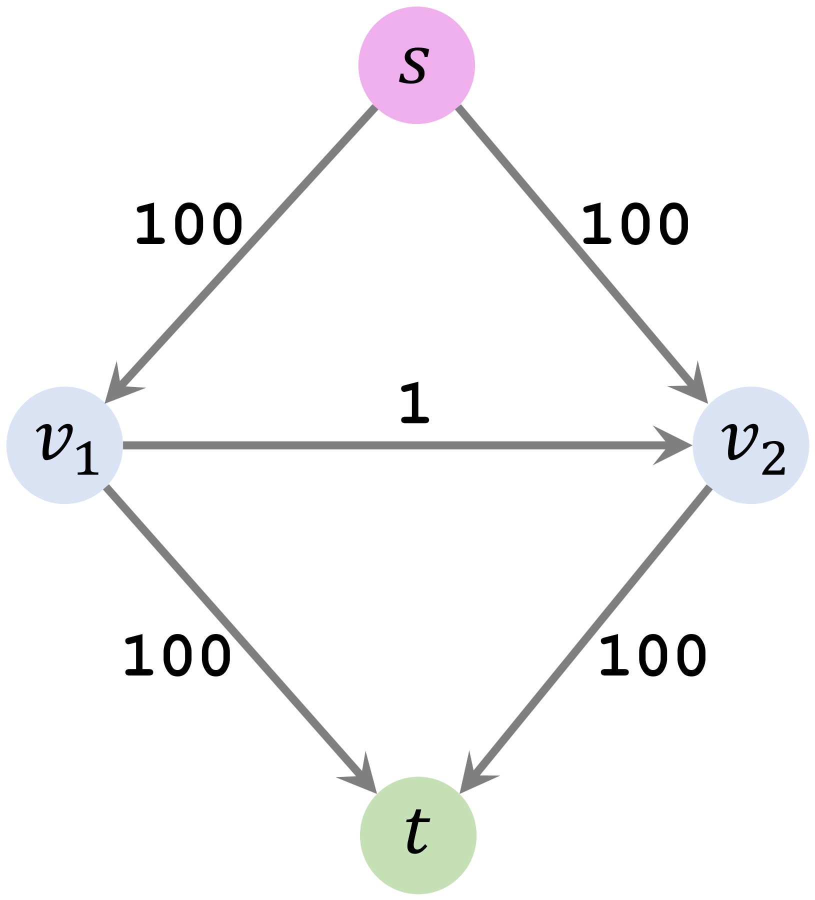
- The amount of the max flow is 200.

A bad case for Ford-Fulkerson algorithm

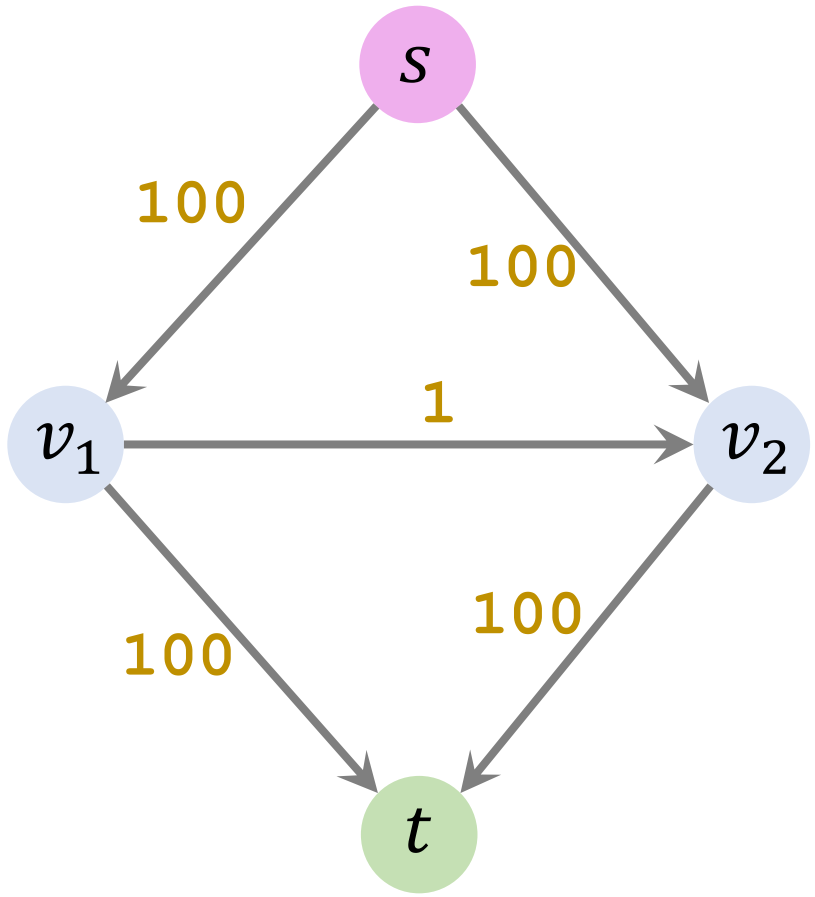


- The amount of the max flow is 200.
- Ford-Fulkerson algorithm takes a long time to find the right answer.

Initialization

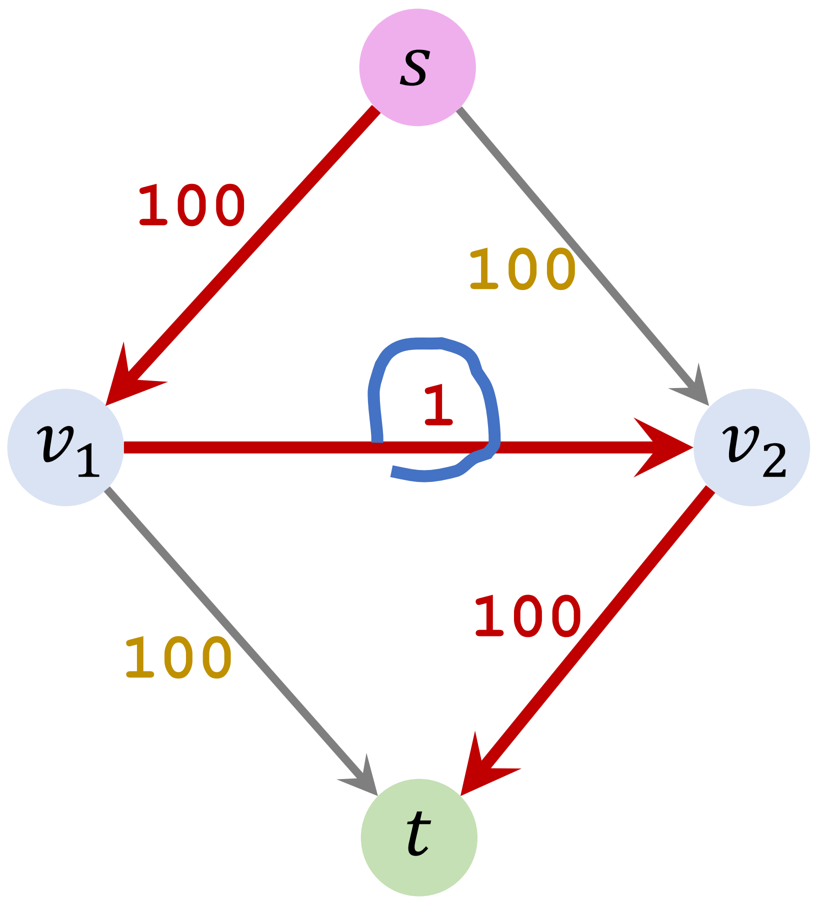


Original Graph



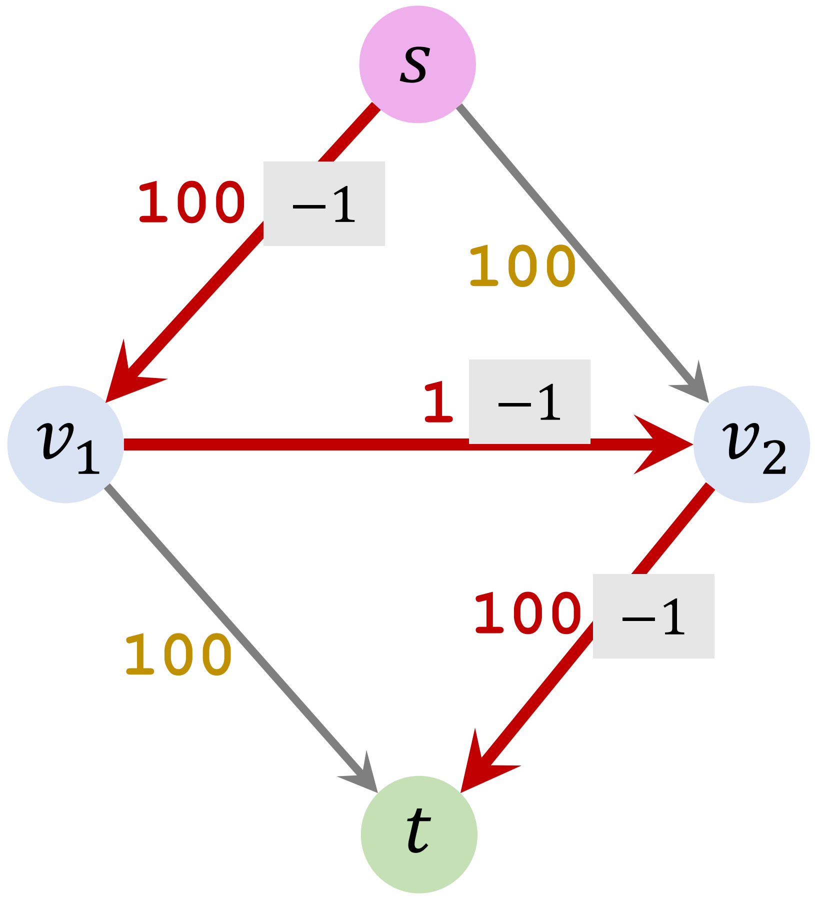
Residual Graph

Iteration 1: Find an augmenting path

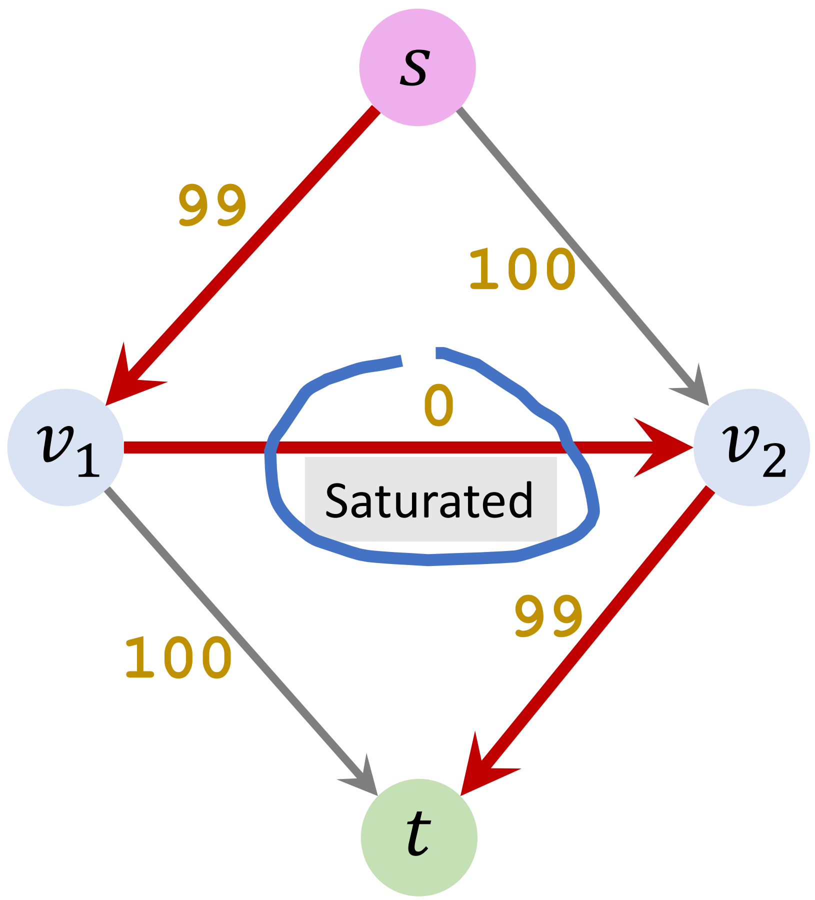


Found path $s \to v_1 \to v_2 \to t$. (Bottleneck capacity = 1.)

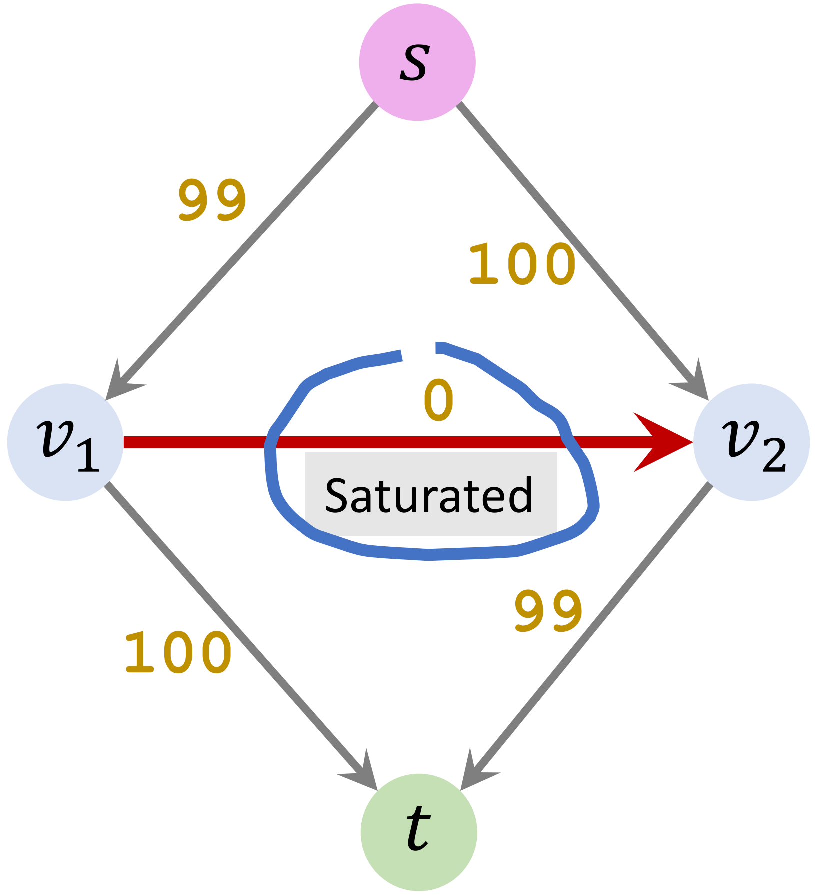
Iteration 1: Update residuals



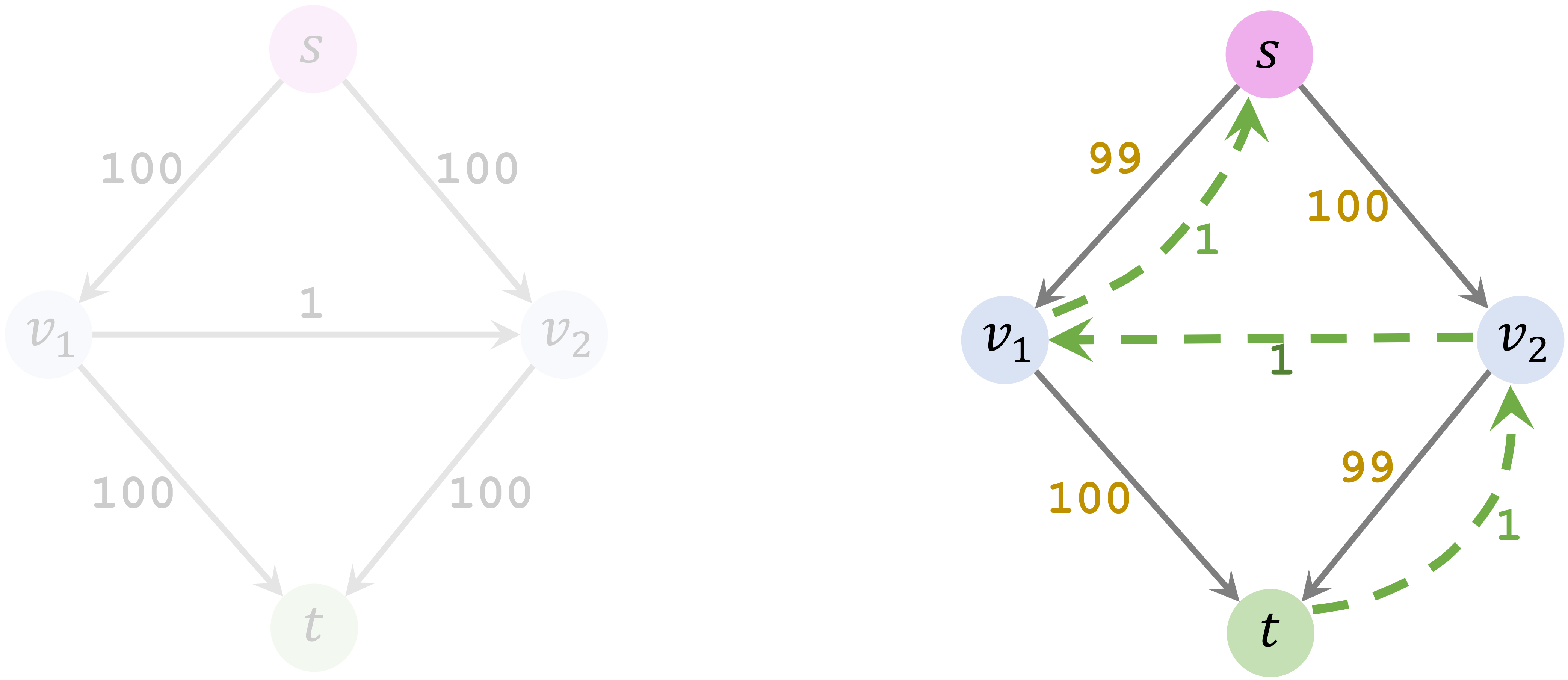
Iteration 1: Update residuals



Iteration 1: Remove saturated edges

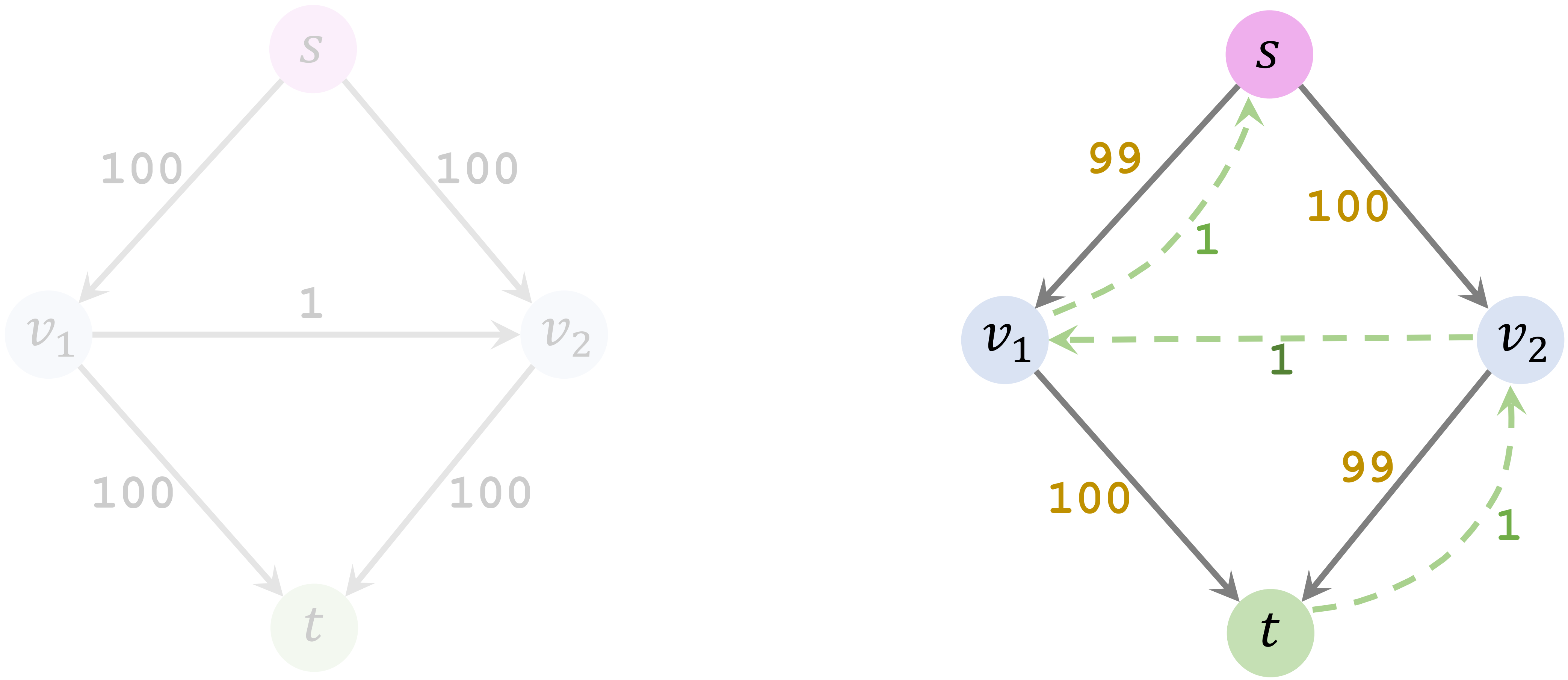


Iteration 1: Add backward path

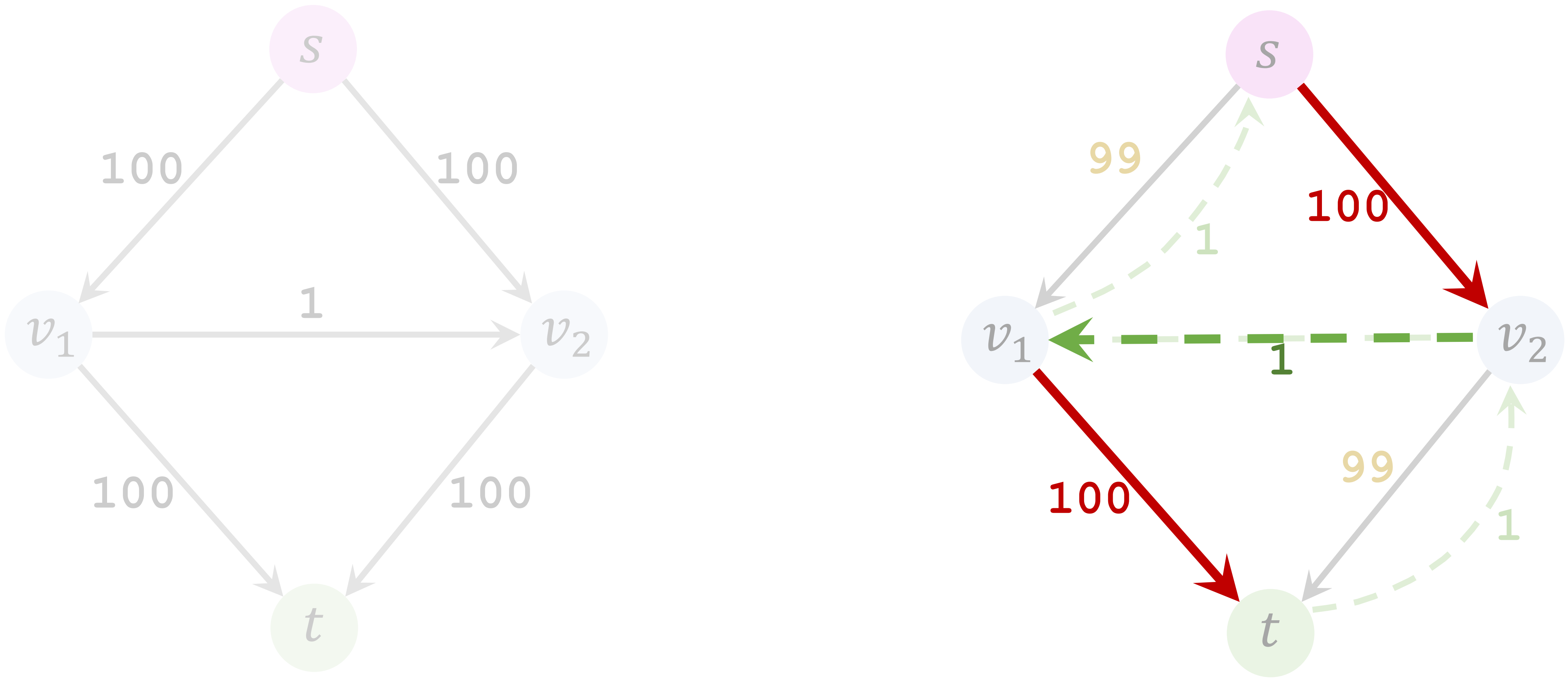


Add backward path $t \to v_2 \to v_1 \to s$ with capacity $= 1$.

Now, the flow is 1

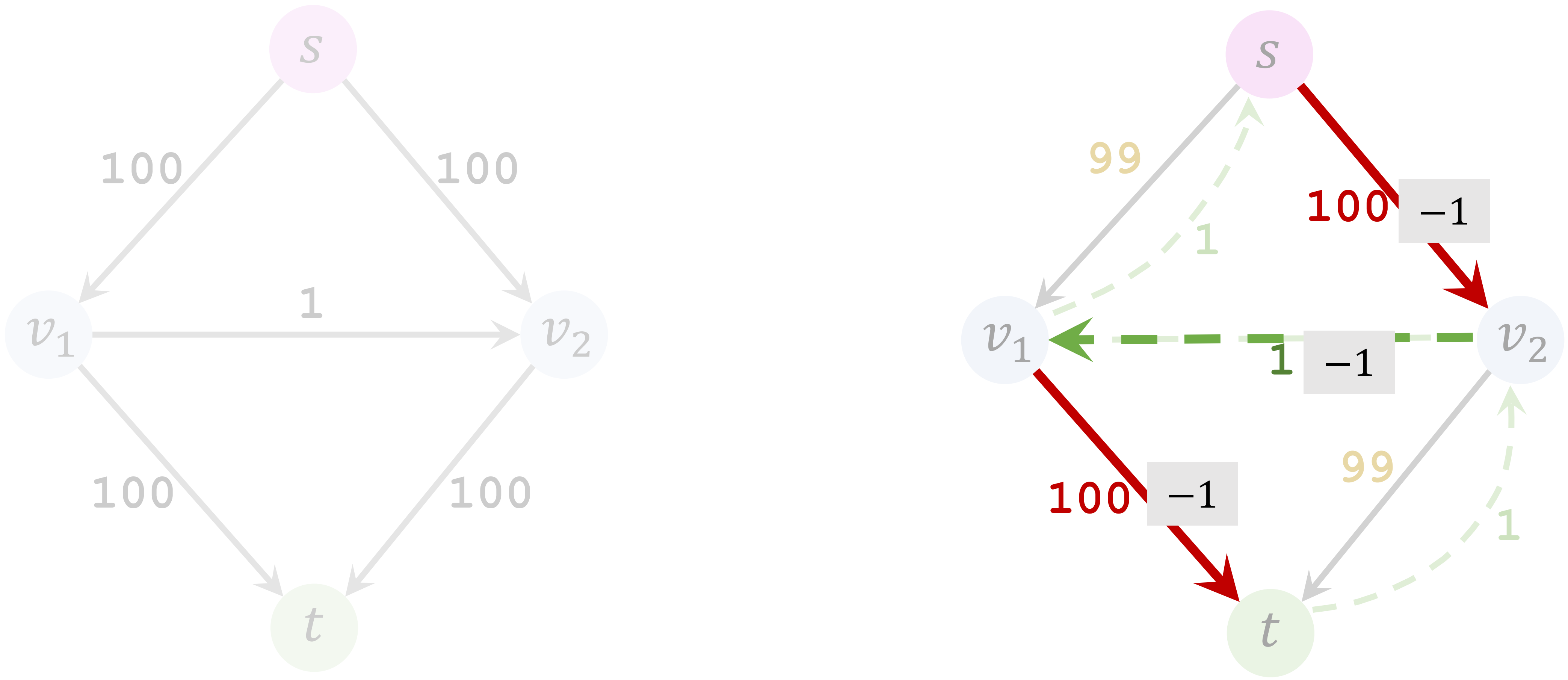


Iteration 2: Find an augmenting path



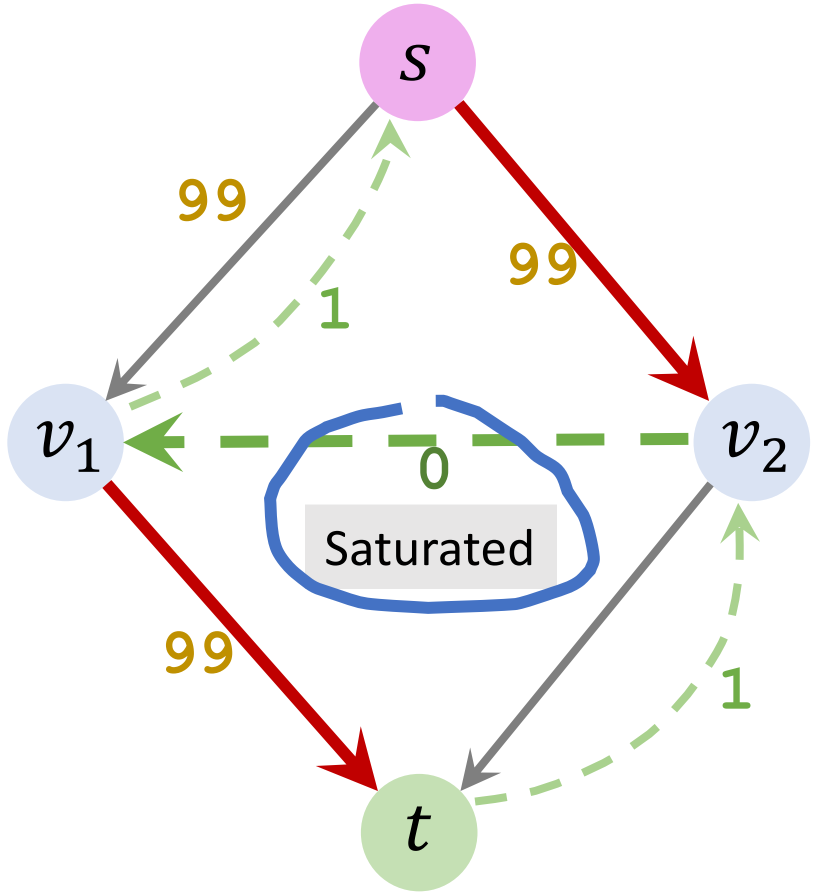
Found path $s \to v_2 \to v_1 \to t$. (Bottleneck capacity = 1.)

Iteration 2: Update residuals



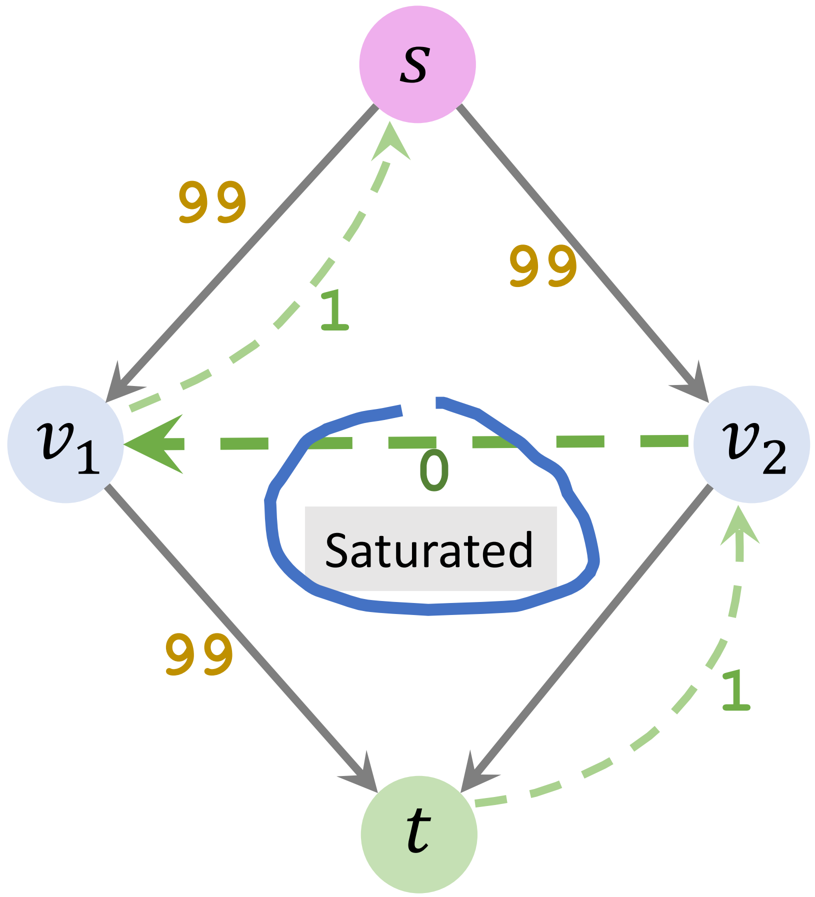
Found path $s \to v_2 \to v_1 \to t$. (Bottleneck capacity = 1.)

Iteration 2: Update residuals

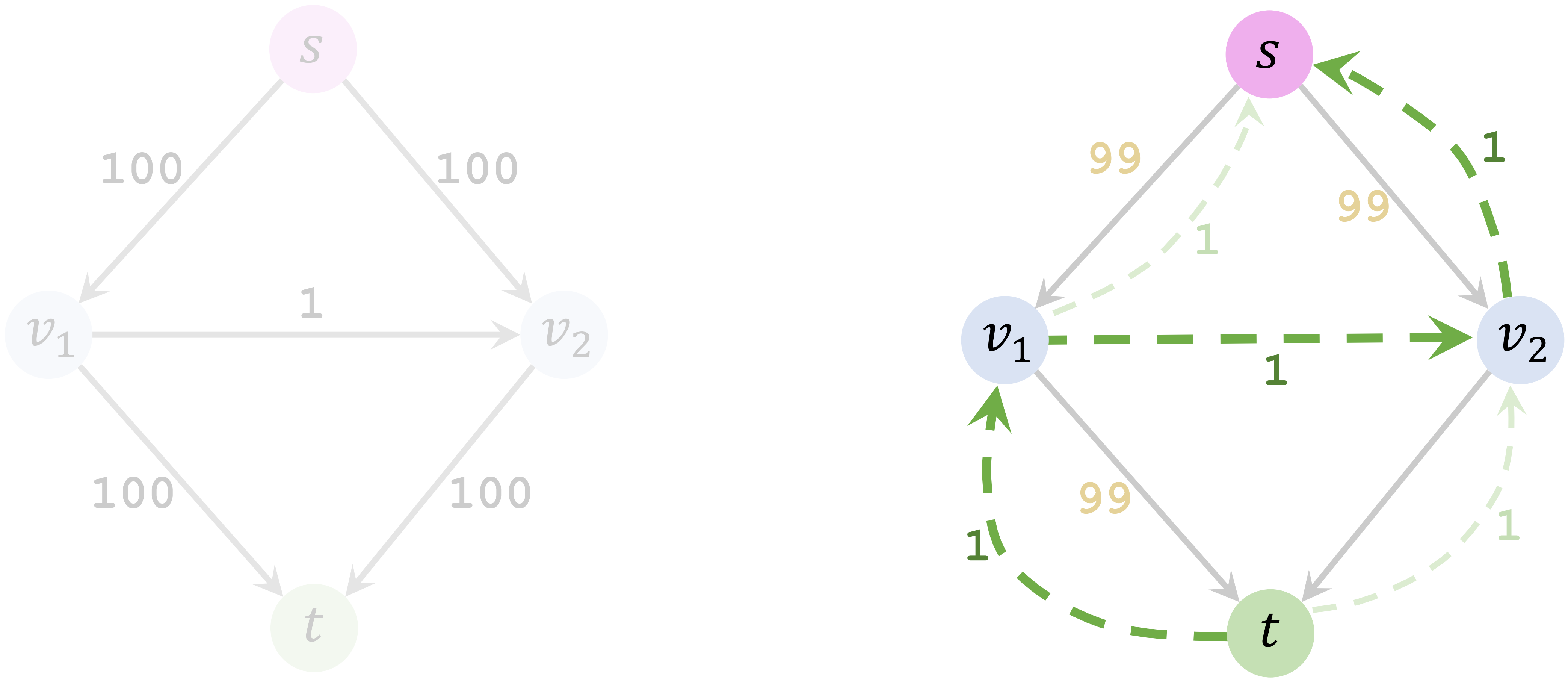


Found path $s \to v_2 \to v_1 \to t$. (Bottleneck capacity $= 1$.)

Iteration 2: Remove saturated edges

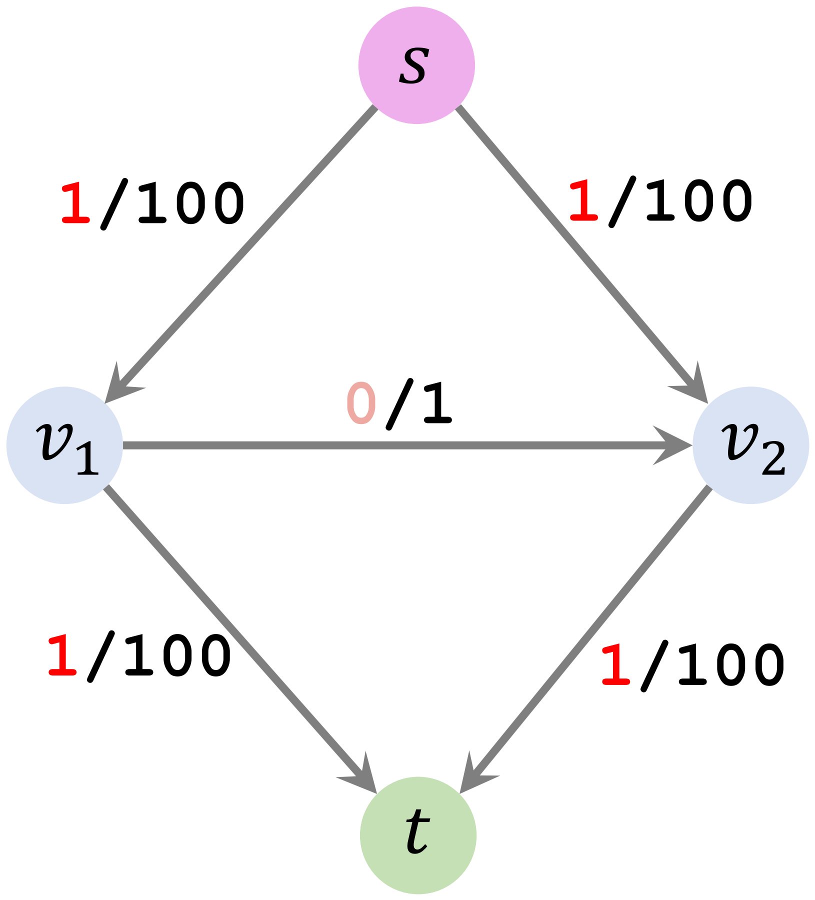


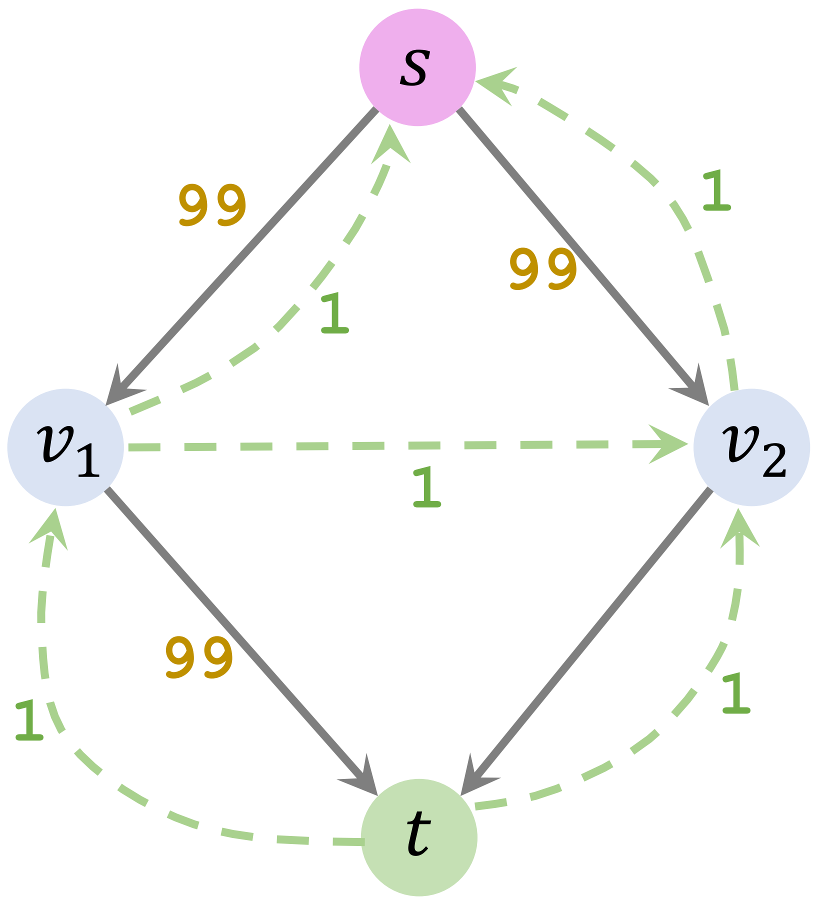
Iteration 2: Add backward path



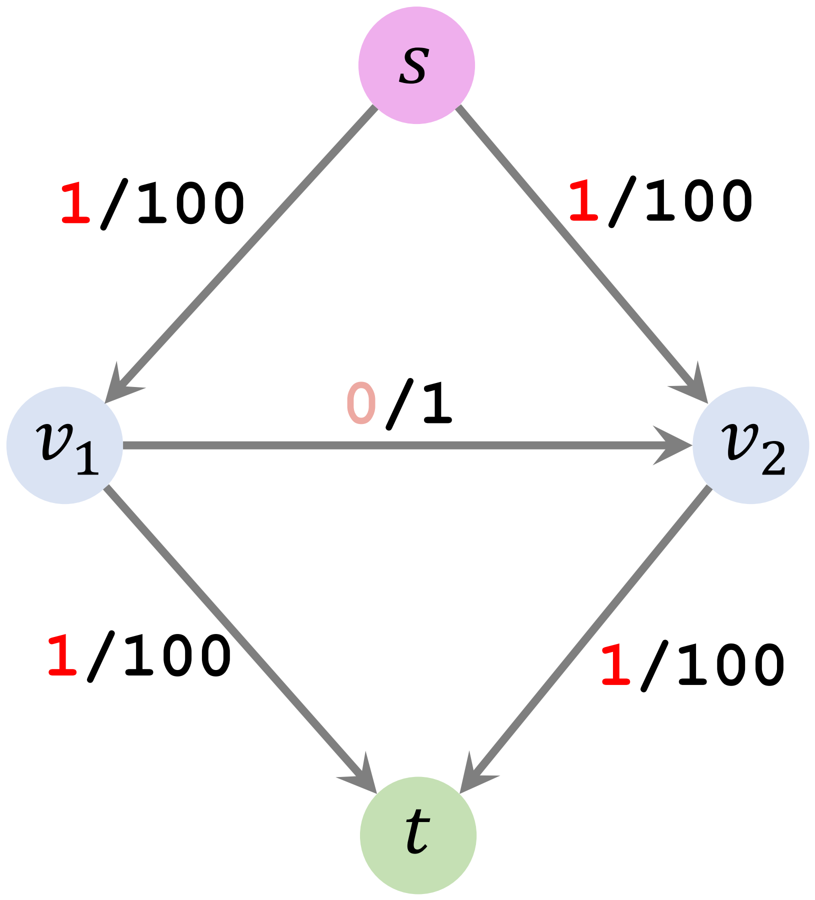
Add backward path $t \to v_1 \to v_2 \to s$ with capacity $= 1$.

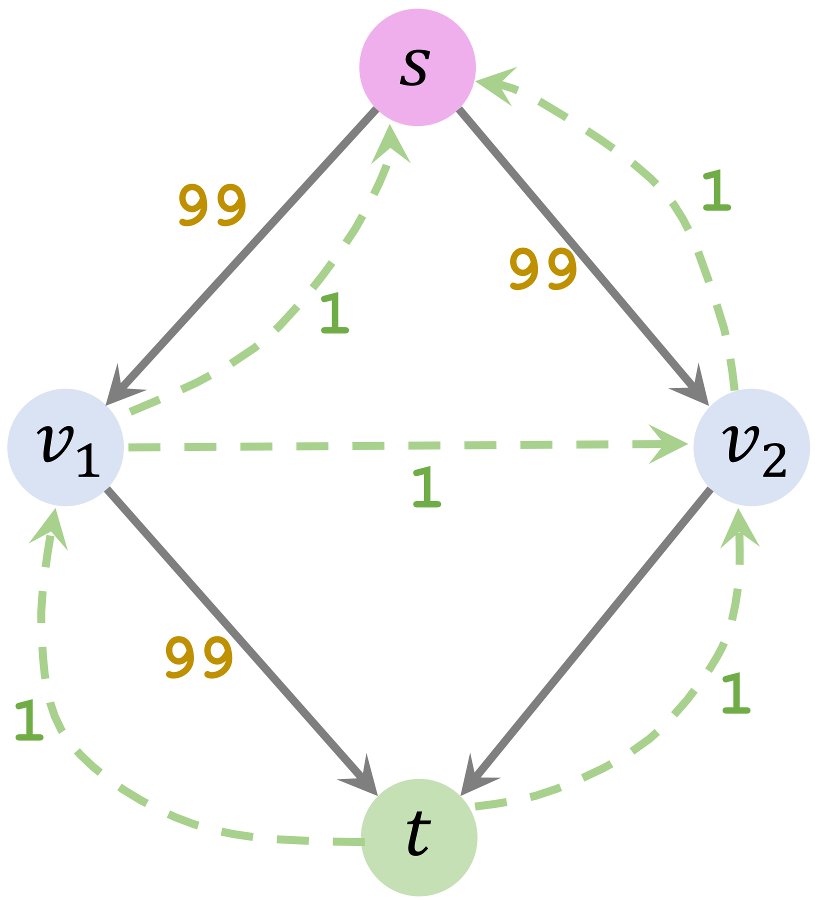
Now, the flow is 2





Slow improvement…





In every iteration, the flow leaving the source increases by 1.

Worst-Case Iteration Complexity

- Ford-Fulkerson algorithm always improve the flow in every iteration.
- Thus, $\texttt{\#Iterations} \leq \texttt{MaxFlow}$.

- In our example, each iteration increases the flow by only 1.
- Thus, $\texttt{\#Iterations} = \texttt{MaxFlow}$.

- In sum, the worst-case $\texttt{\#Iterations}$ is equal to $\texttt{MaxFlow}$.

Worst-Case Time Complexity

- Let m be the number of edges.
- It takes $O(m)$ time to find a path in unweighted graph.
  (Ignore the weights in the residual graph.)
- Thus, the per-iteration time complexity is $O(m)$.

Worst-Case Time Complexity

- Let m be the number of edges.
- It takes $O(m)$ time to find a path in unweighted graph. (Ignore the weights in the residual graph.)
- Thus, the per-iteration time complexity is $O(m)$.

- Let the maximum flow be f.
- The worst-case time complexity is $O(f \cdot m)$.
- (In practice, the time complexity is not so bad.)

Edmonds–Karp Algorithm

Reference

- Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM.* 19 (2): 248–264, 1972.

Edmonds–Karp Algorithm

- Edmonds-Karp algorithm is a special case of Ford-Fulkerson algorithm.
- Edmonds-Karp algorithm uses the shortest path from source to sink. (Apply weight 1 to all the edges in the residual graph.)
- Everything else is the same as Ford-Fulkerson algorithm.

Edmonds–Karp Algorithm

- Edmonds-Karp algorithm has $O(m^2 \cdot n)$ time complexity.
- m: number of edges.
- n: number of vertices.

Summary

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to be the capacities.

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to be the capacities.
2. While augmenting path can be found:
    a. Find an augmenting path (in the residual graph.)
    b. Find the bottleneck capacity x on the augmenting path.
    c. Update the residuals. (Along the path, $\text{Residual} = \text{Residual} - x$.)
    d. Add a backward path. (Along the path, edge weights are all x.)

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to be the capacities.
2. While augmenting path can be found:
   a. Find an augmenting path (in the residual graph.)
   b. Find the bottleneck capacity x on the augmenting path.
   c. Update the residuals. (Along the path, $\text{Residual} = \text{Residual} - x$.)
   d. Add a backward path. (Along the path, edge weights are all x.)

Time complexity: $O(f \cdot m)$. (f is the max flow; m is #edges.)

Edmonds–Karp Algorithm

1. Build a residual graph; initialize the residuals to the capacity.
2. While augmenting path can be found:
    a. Find the shortest augmenting path (in the residual graph.)
    b. Find the bottleneck capacity b on the augmenting path.
    c. Update the residuals. (Along the path, $\text{Residual} = \text{Residual} - b$.)
    d. Add a backward path. (Along the path, edge weights are all b.)

Time complexity: $O(m^2 \cdot n)$. (m is #edges; n is #vertices.)

Thank You!