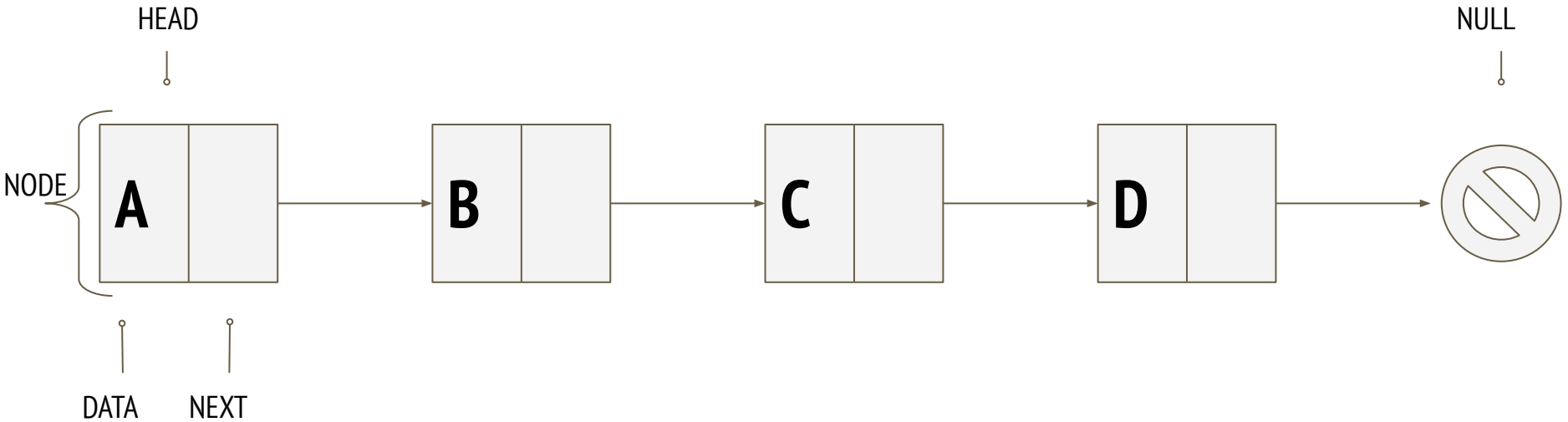

Linked Lists

Introduction

Linked List (Singly Linked List)

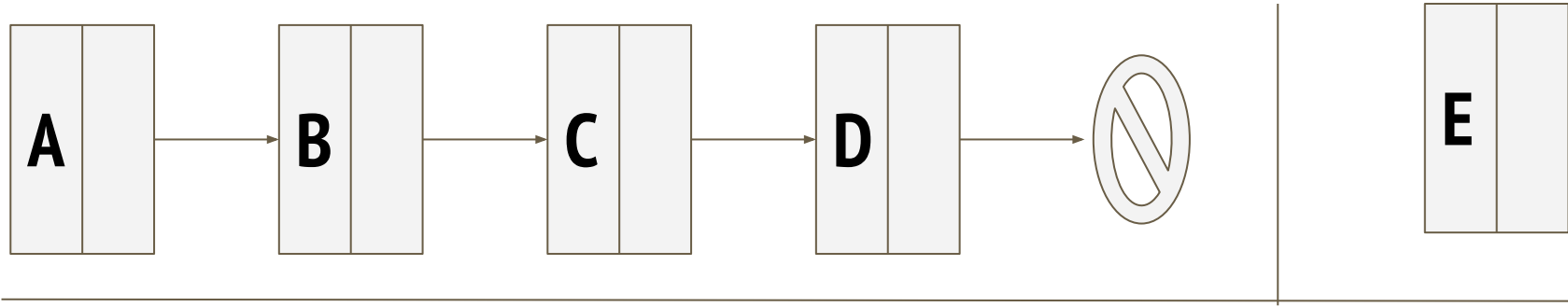


Array vs. Linked List

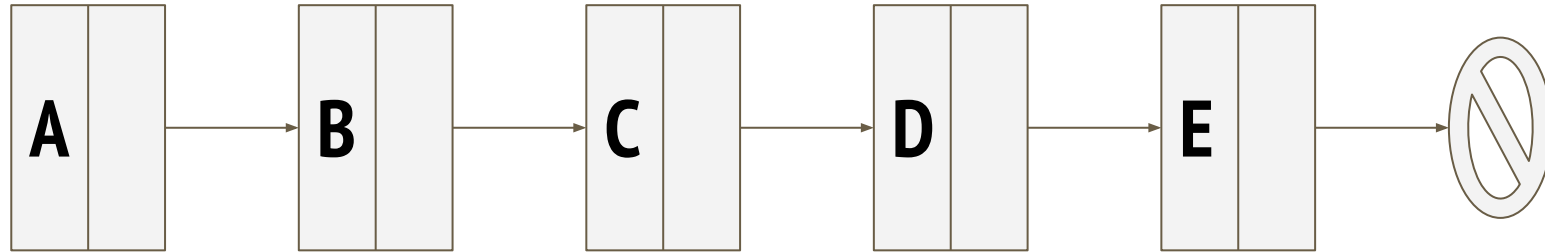
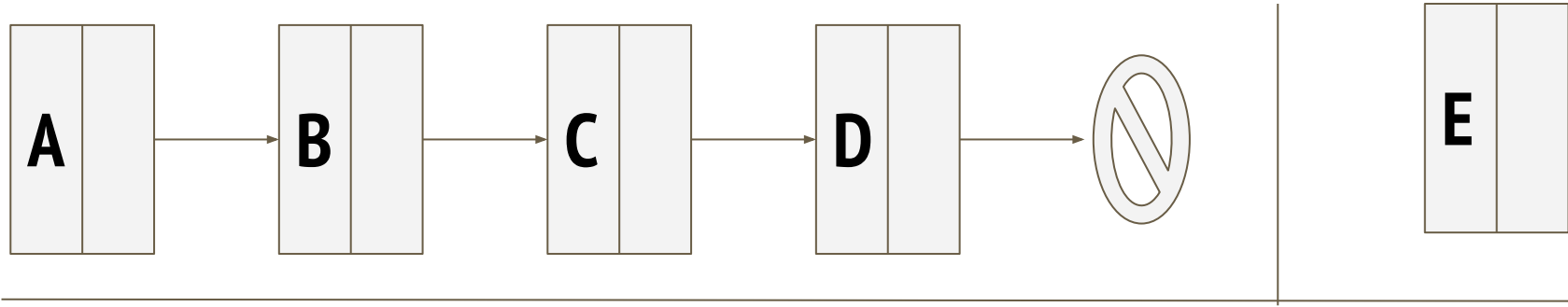
	Array	Linked List
Insertion/Deletion	$O(n)$	$O(1)$
Access Element	$O(1)$	$O(n)$

Linked Lists : Insertion

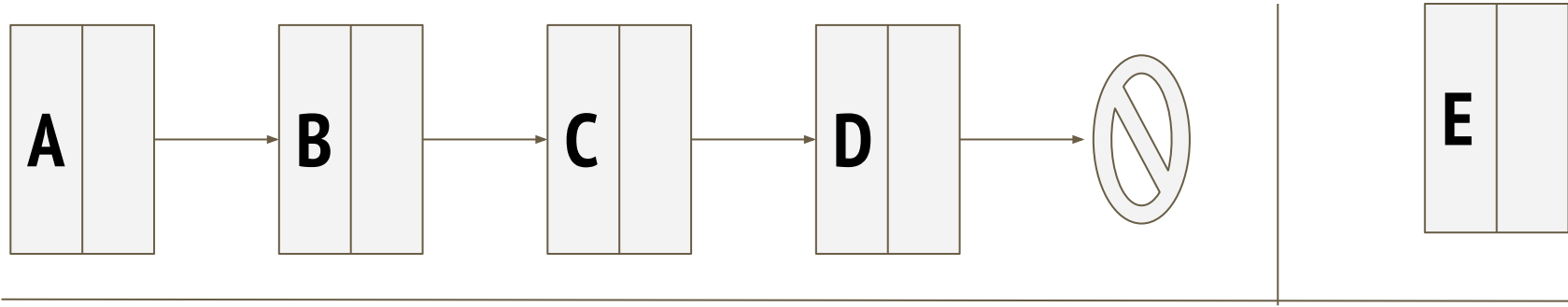
Singly Linked List: Append



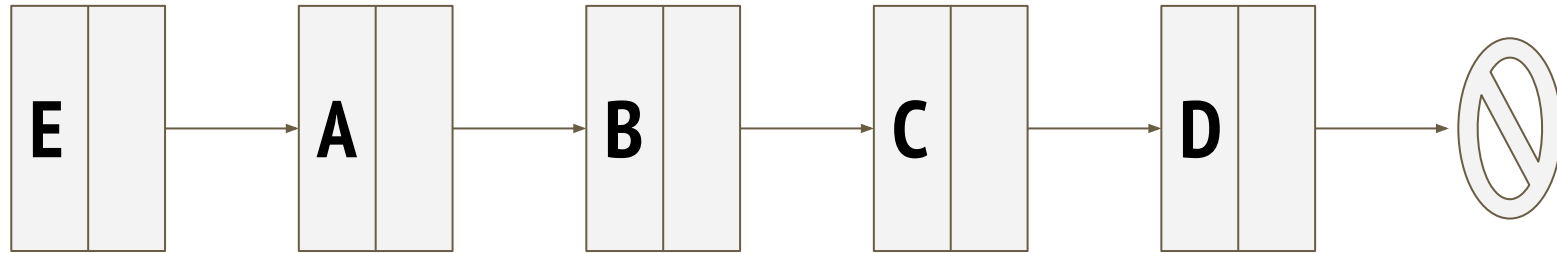
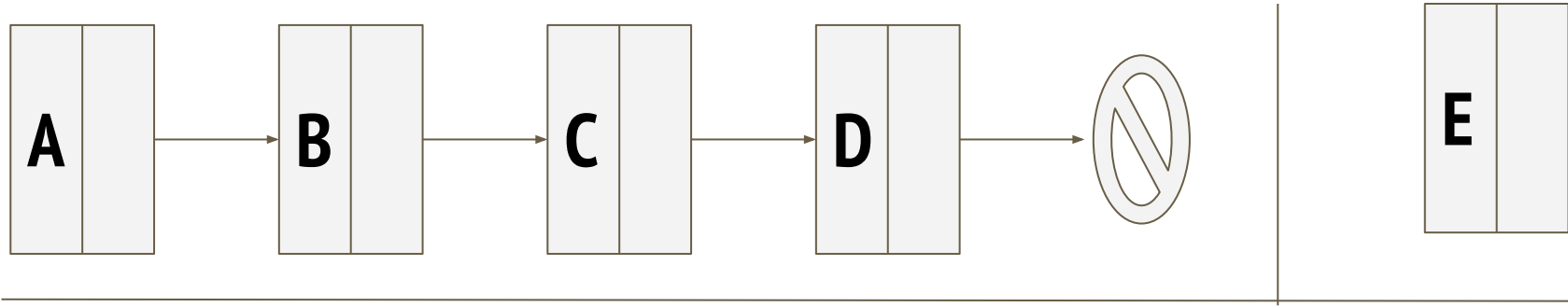
Singly Linked List: Append



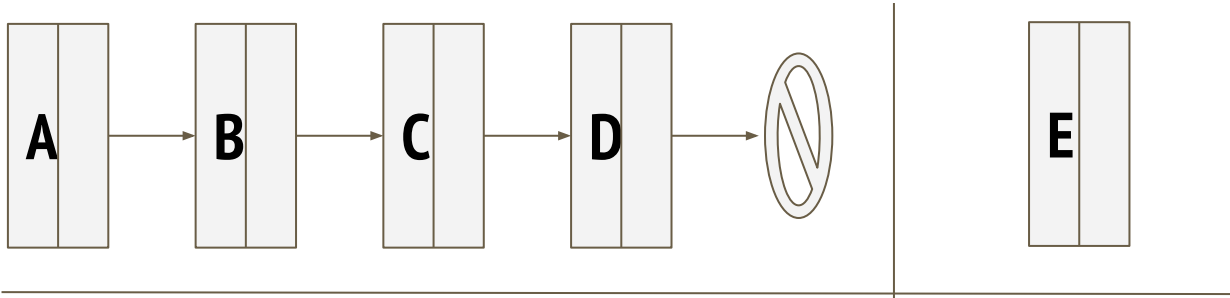
Singly Linked List: Prepend



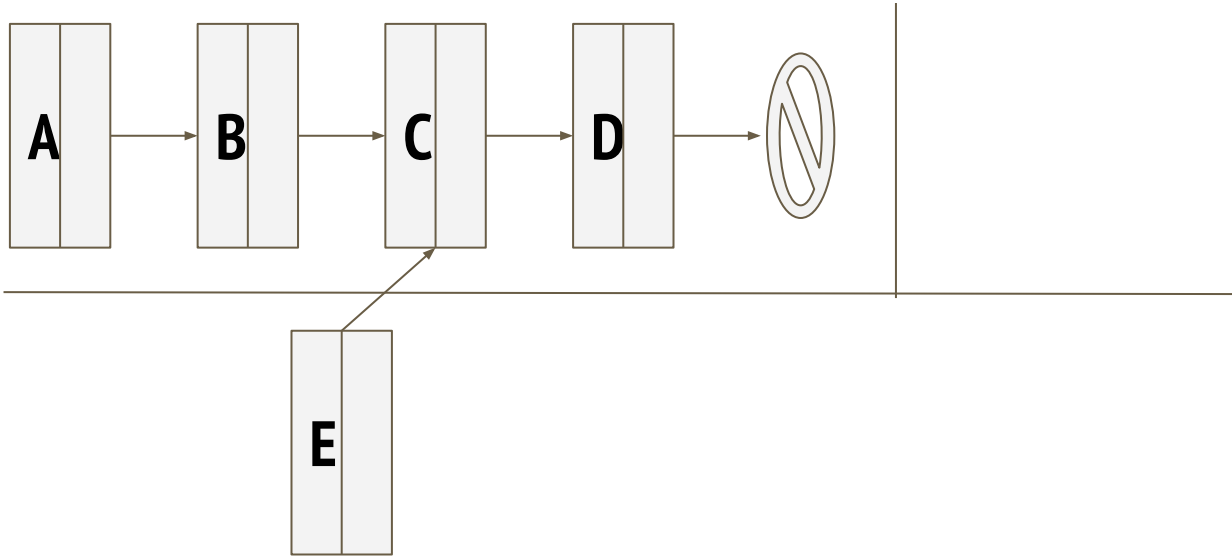
Singly Linked List: Prepend



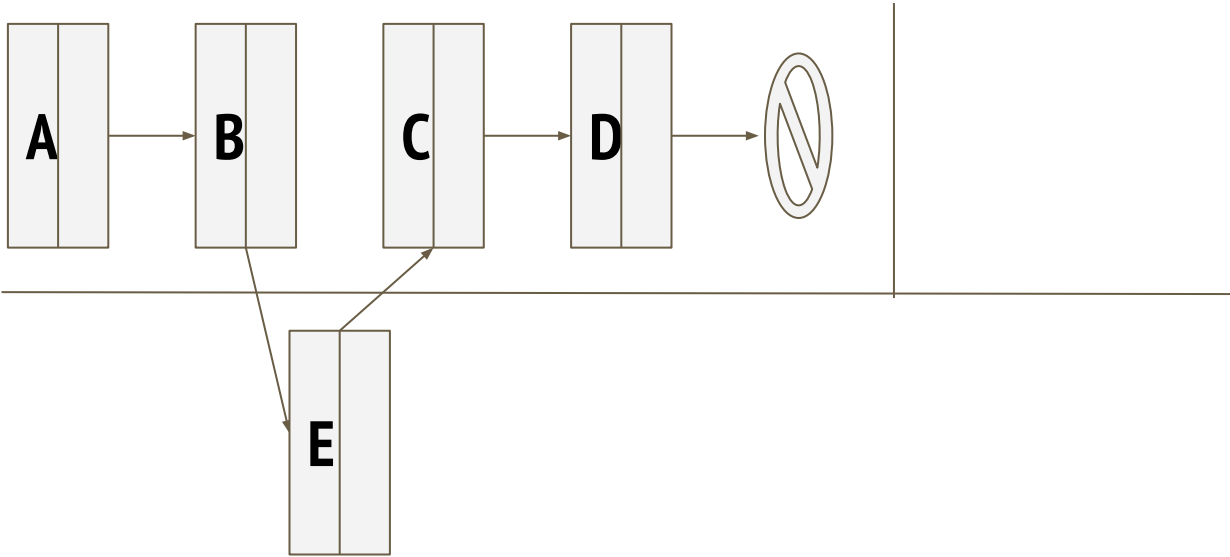
Singly Linked List: Insert after Node



Singly Linked List: Insert after Node



Singly Linked List: Insert after Node



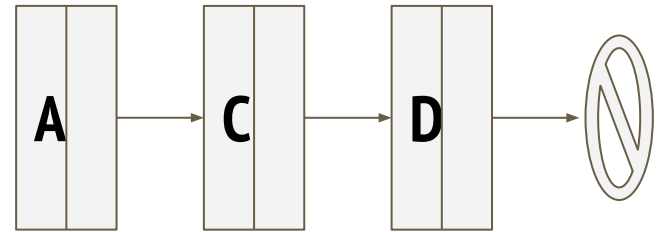
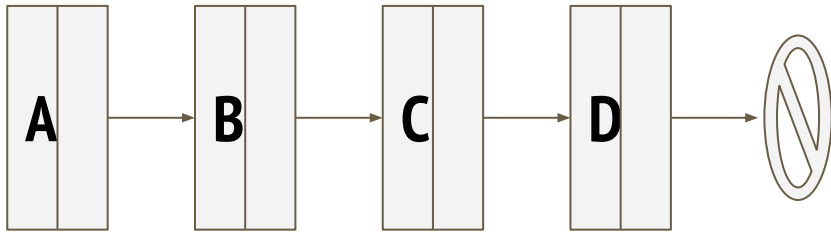
Linked Lists : Deletion

Singly Linked List: Delete node

Given a key (data field) delete node with this field.

Assume elements in linked list are unique.

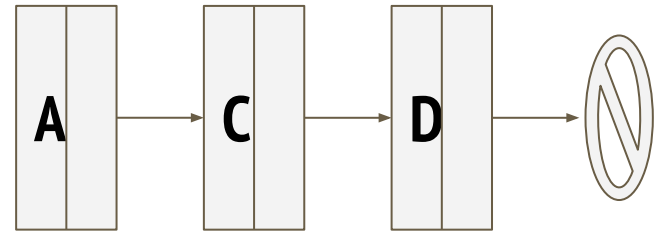
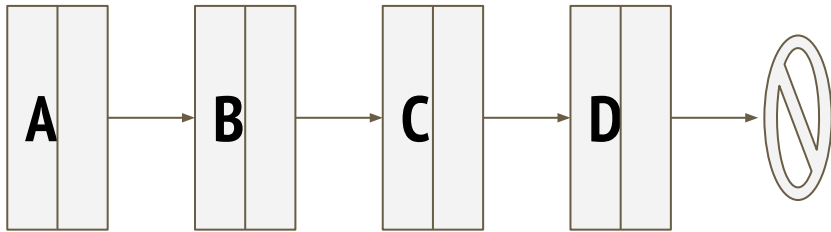
Example: Delete node with data field "B":



Singly Linked List: Delete node

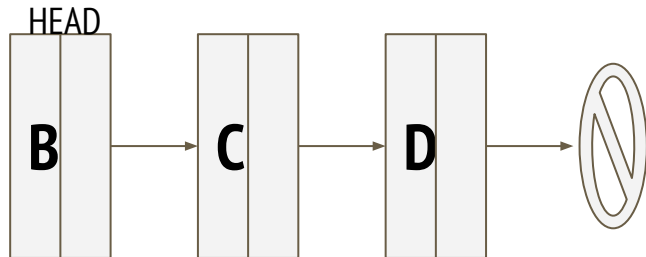
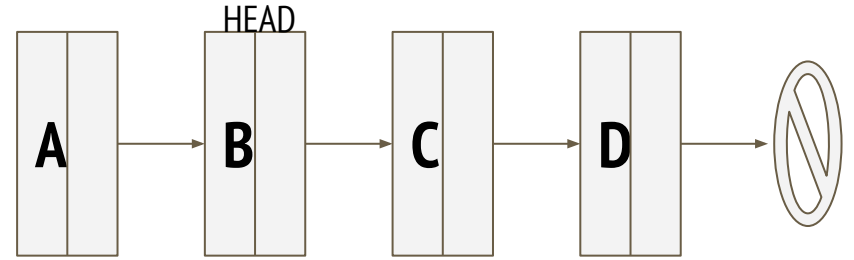
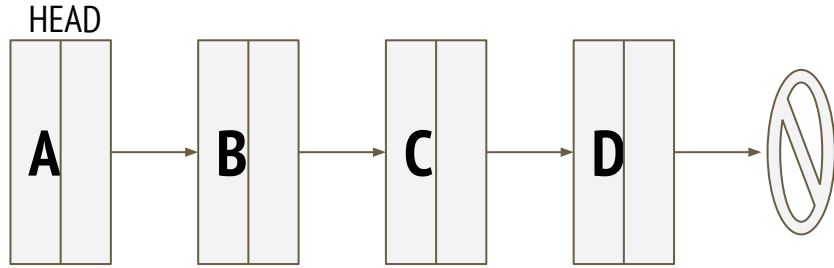
Two cases:

- Node to be deleted is head.
- Node to be deleted is not head.



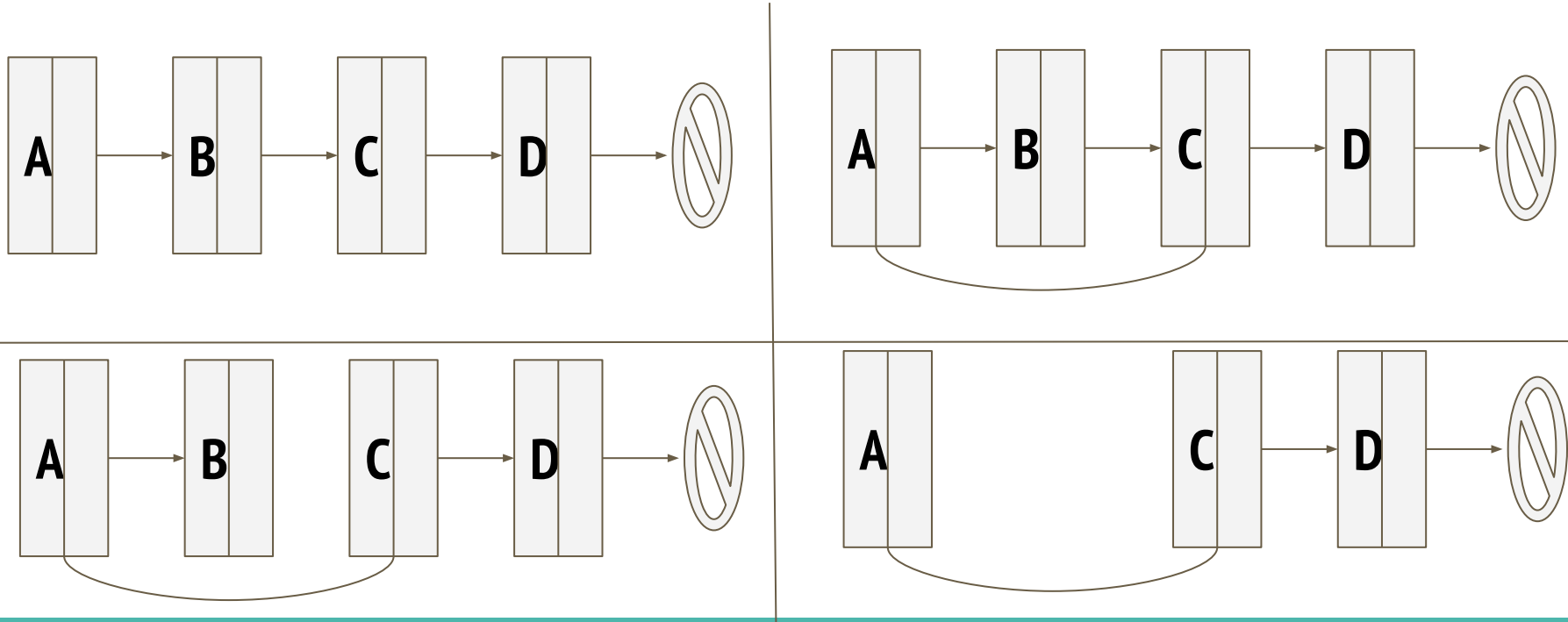
Singly Linked List: Delete node

Case 1: Node to be deleted is head



Singly Linked List: Delete node

Case 2: Node to be deleted is not head (say node with “B”)

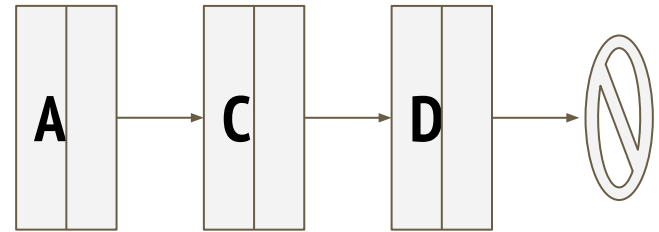
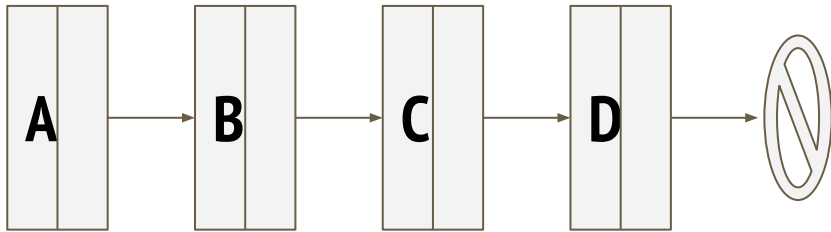


Singly Linked List: Delete node at position

Given a position, delete node with this position.

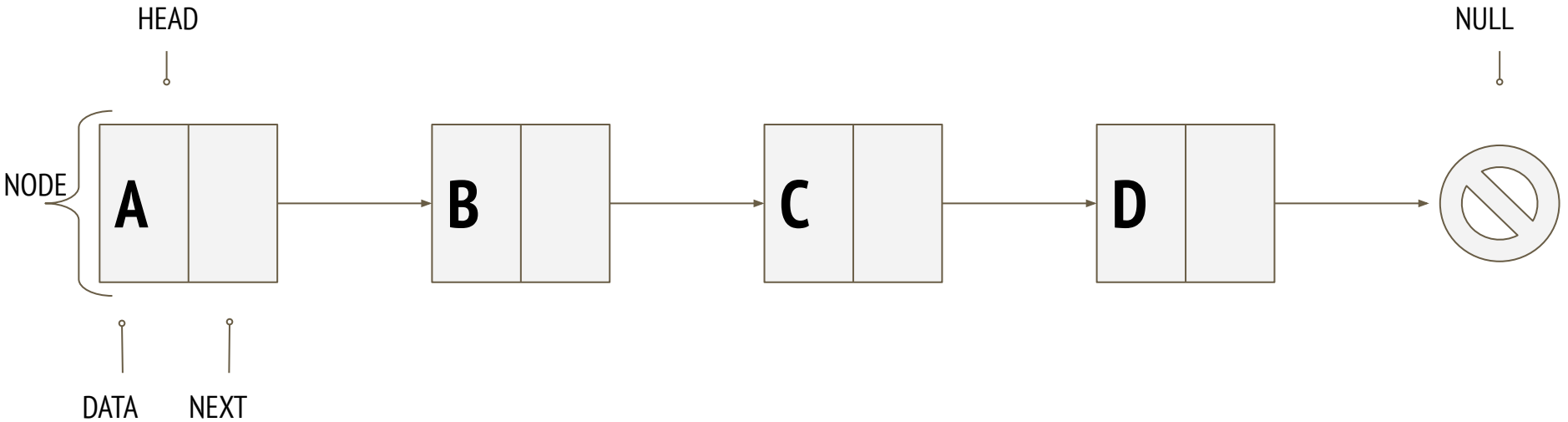
Assume elements in linked list are unique.

Example: Delete node with position 1



Linked Lists : Calculating Length

Singly Linked List: Calculating length

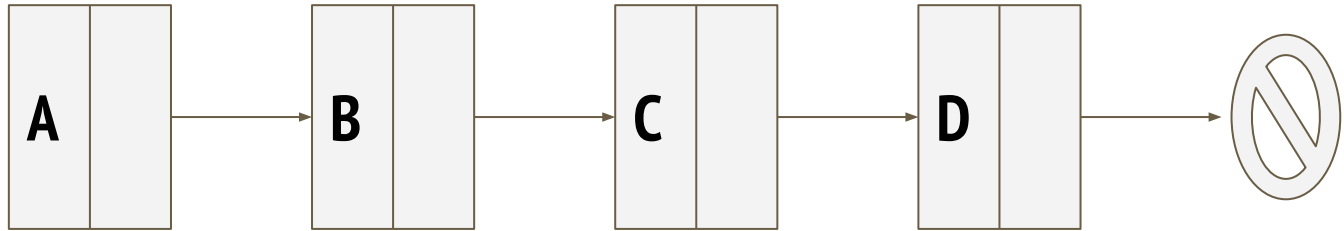


Linked Lists : Node Swap

Singly Linked List: Node Swap

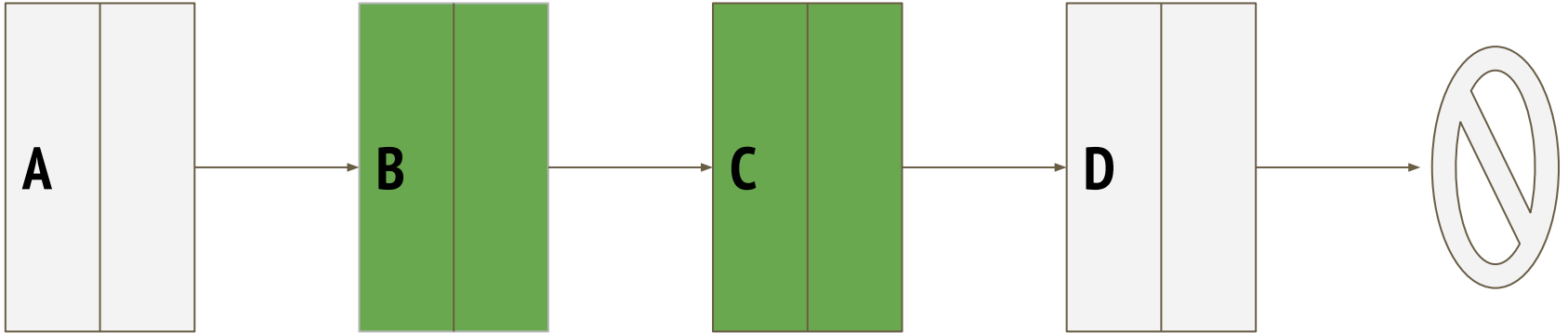
Node swap. Two cases: (Assume data entries are unique).

1. Node_1 and Node_2 are not head nodes.
2. Either Node_1 or Node_2 are head nodes.



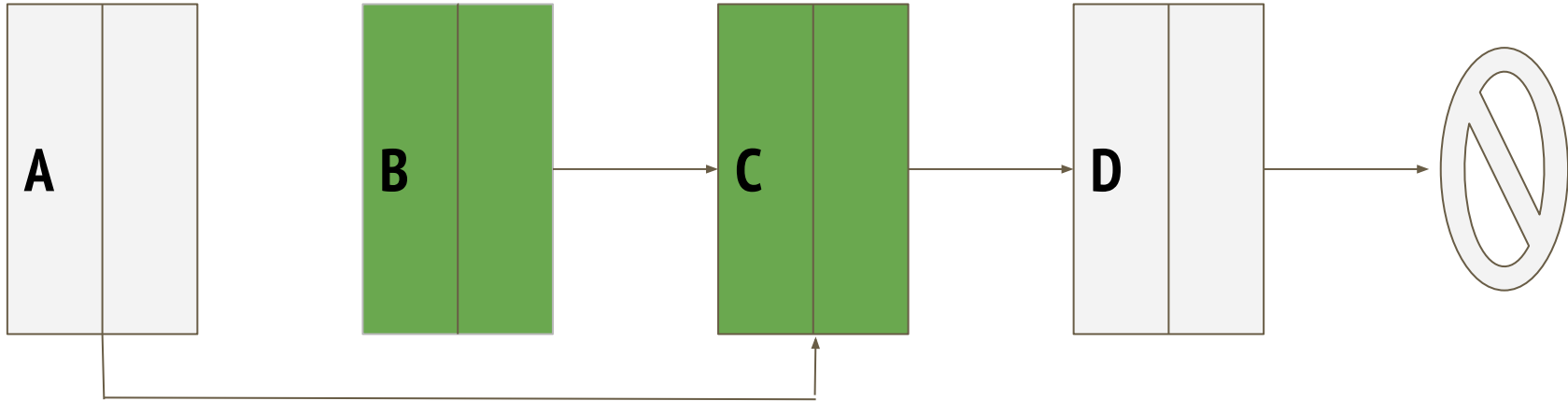
Singly Linked List: Node Swap - Case 1

Node_1 and Node_2 are not head nodes.



Singly Linked List: Node Swap - Case 1

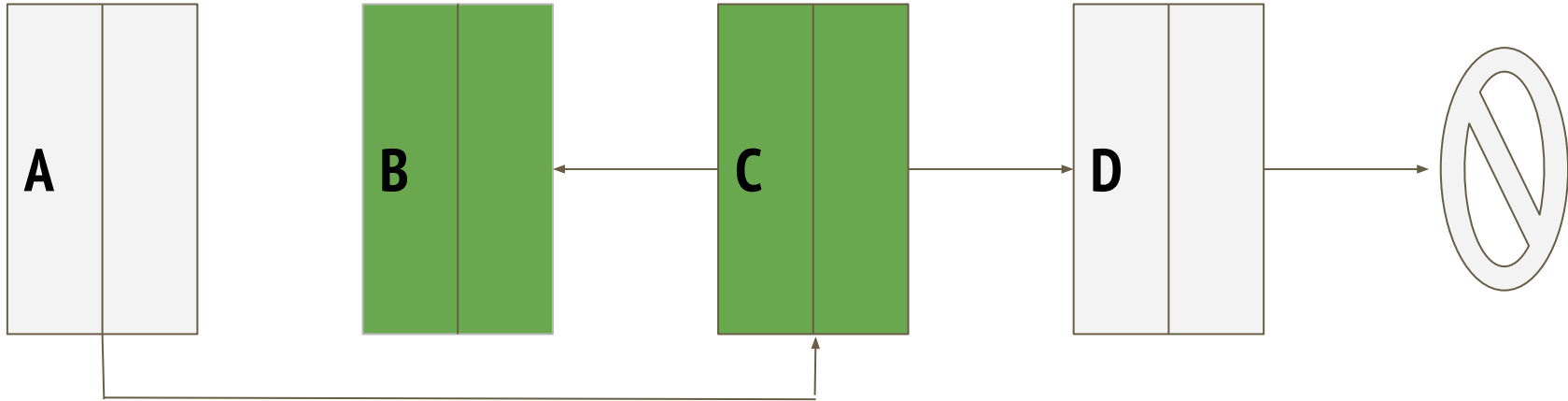
Node_1 and Node_2 are not head nodes.



`prev_1.next = curr_2`

Singly Linked List: Node Swap - Case 1

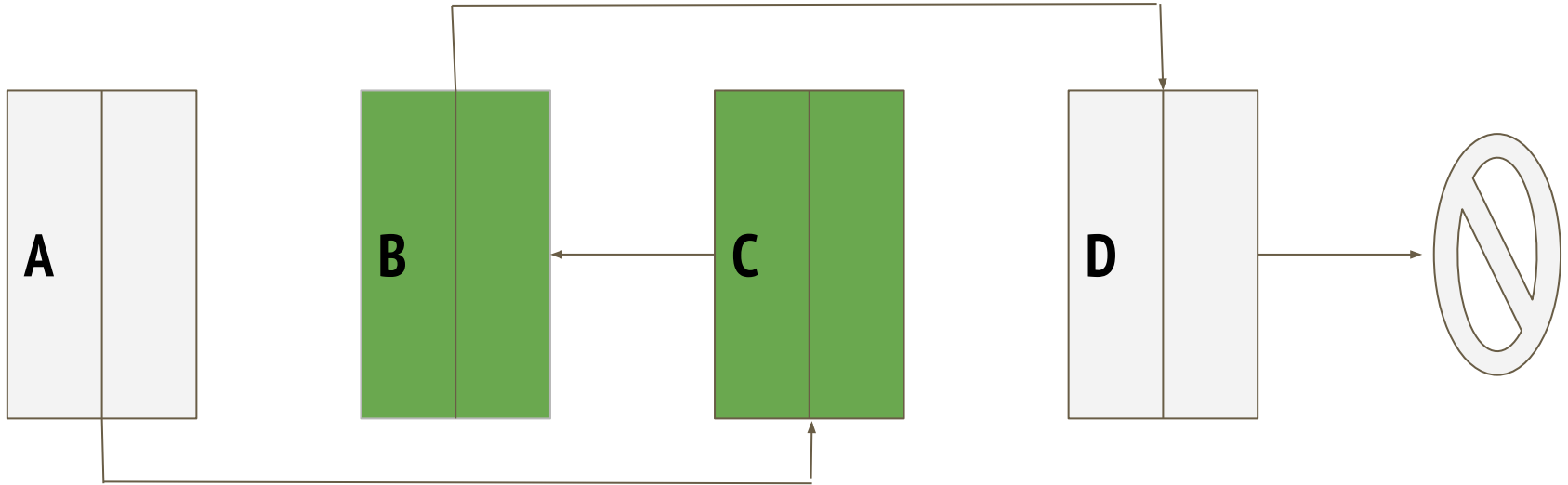
Node_1 and Node_2 are not head nodes.



`prev_2.next = curr_1`

Singly Linked List: Node Swap - Case 1

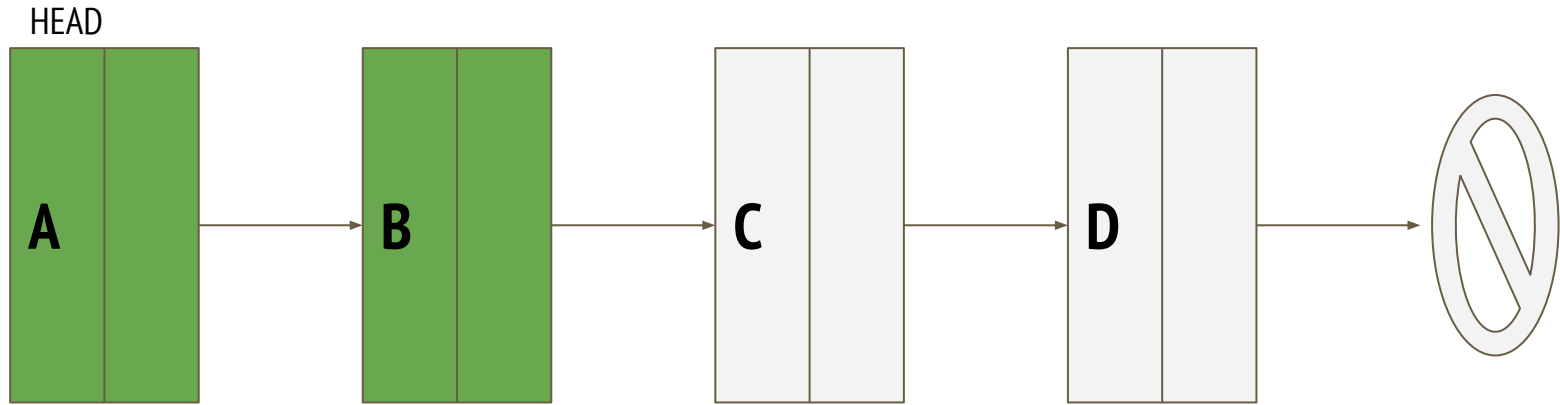
Node_1 and Node_2 are not head nodes.



```
curr_1.next, curr_2.next =  
curr_2.next, curr_1.next
```

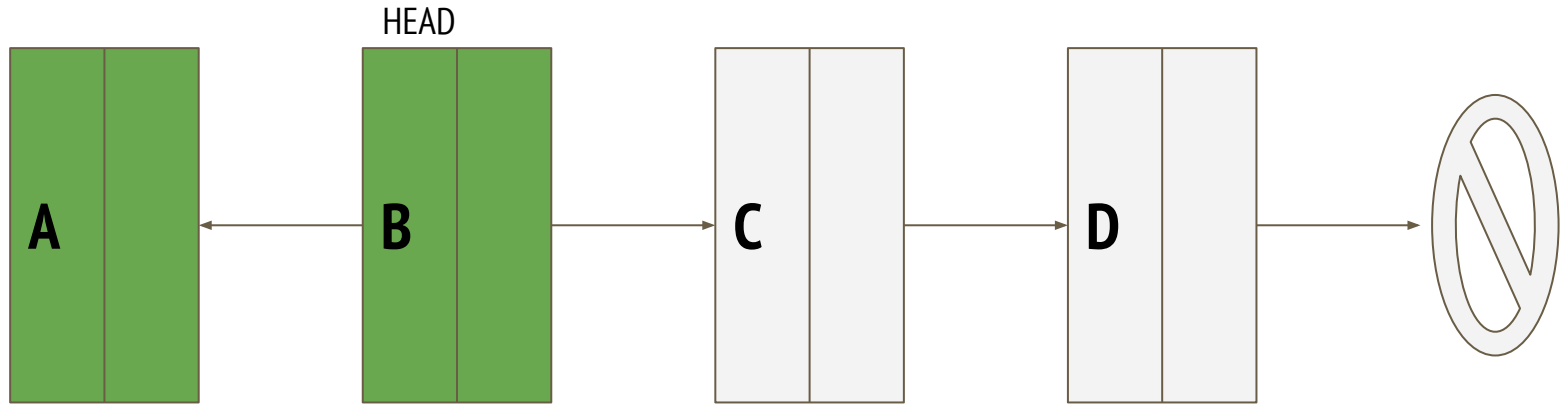
Singly Linked List: Node Swap - Case 2

Node_1 is a head node. Node_2 is not.



Singly Linked List: Node Swap - Case 2

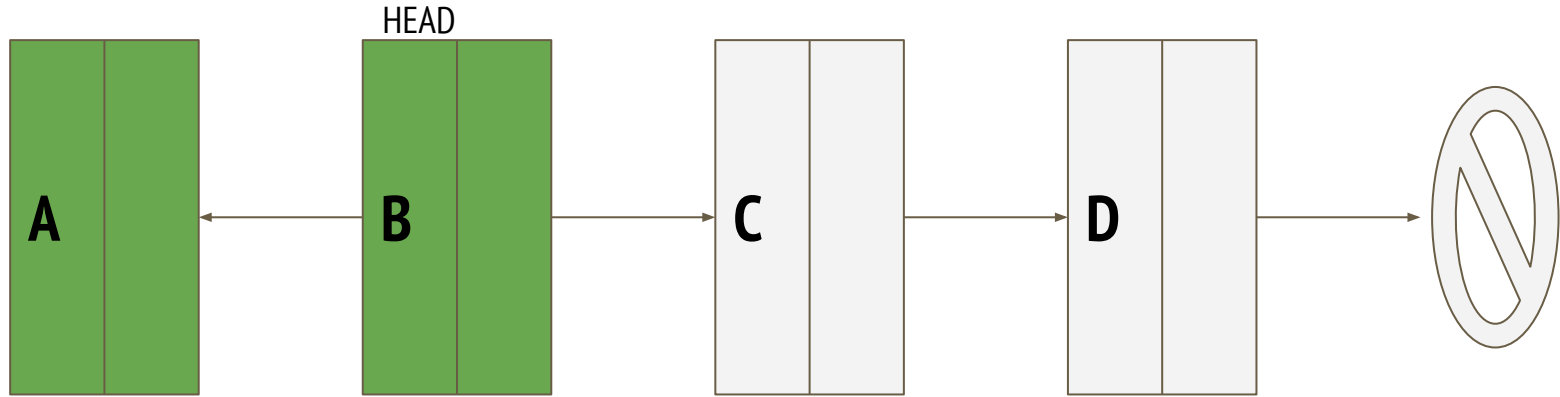
Node_1 is a head node. Node_2 is not.



```
self.head = curr_2
```

Singly Linked List: Node Swap - Case 2

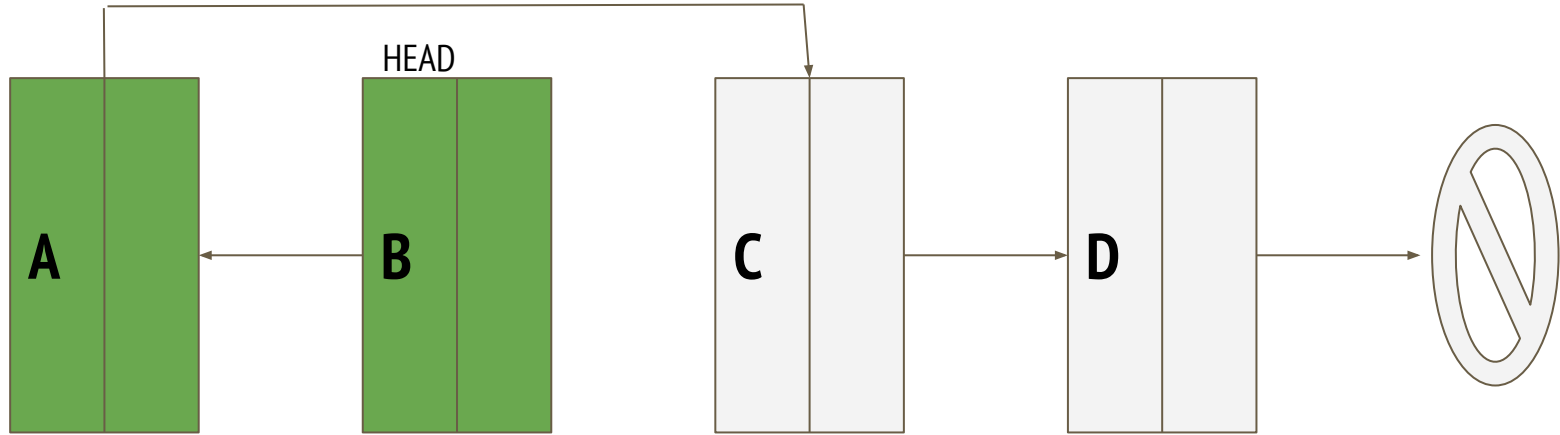
Node_1 is a head node. Node_2 is not.



```
prev_2.next = curr_1
```

Singly Linked List: Node Swap - Case 2

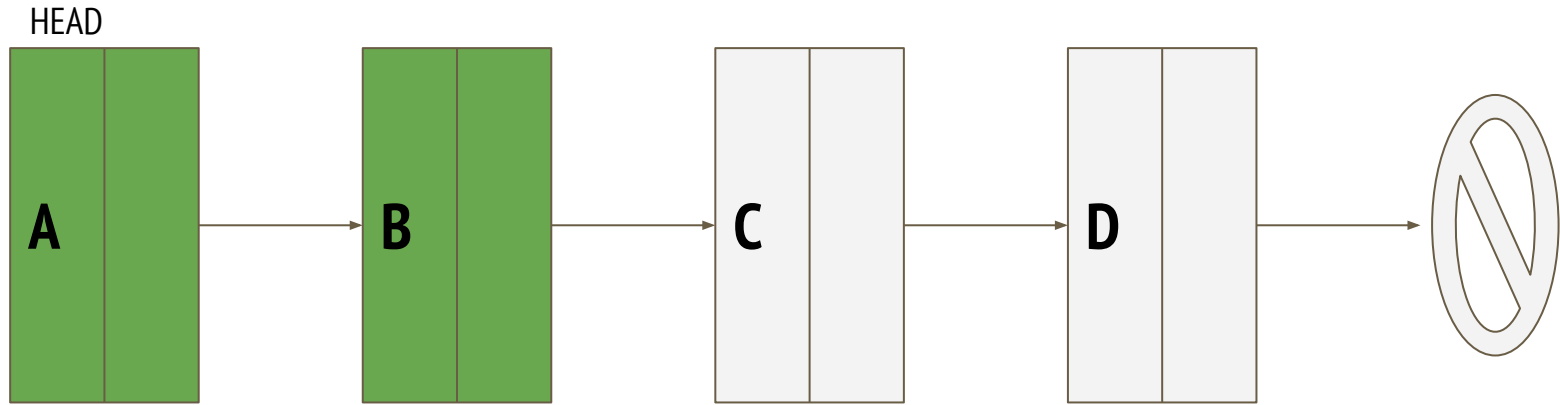
Node_1 is a head node. Node_2 is not.



```
curr_1.next, curr_2.next =  
curr_2.next, curr_1.next
```

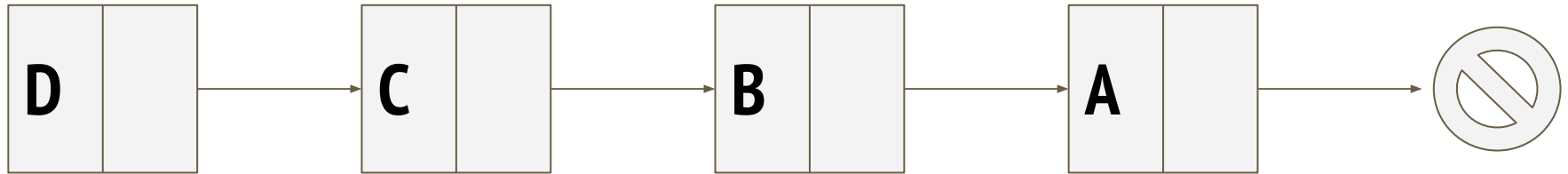
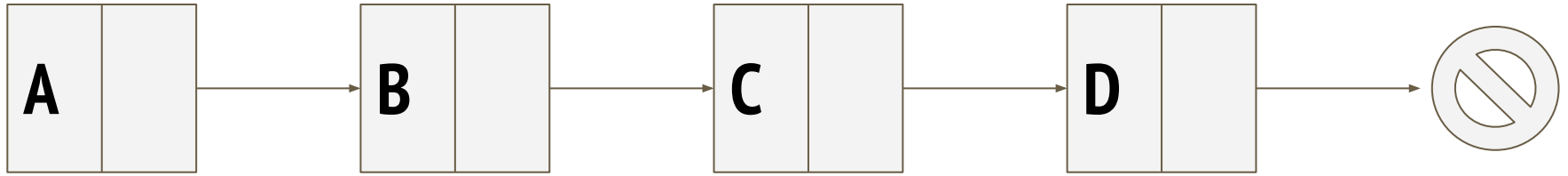
Singly Linked List: Node Swap - Case 3

Node_2 is a head node. Node_1 is not. (almost identical to case 2)



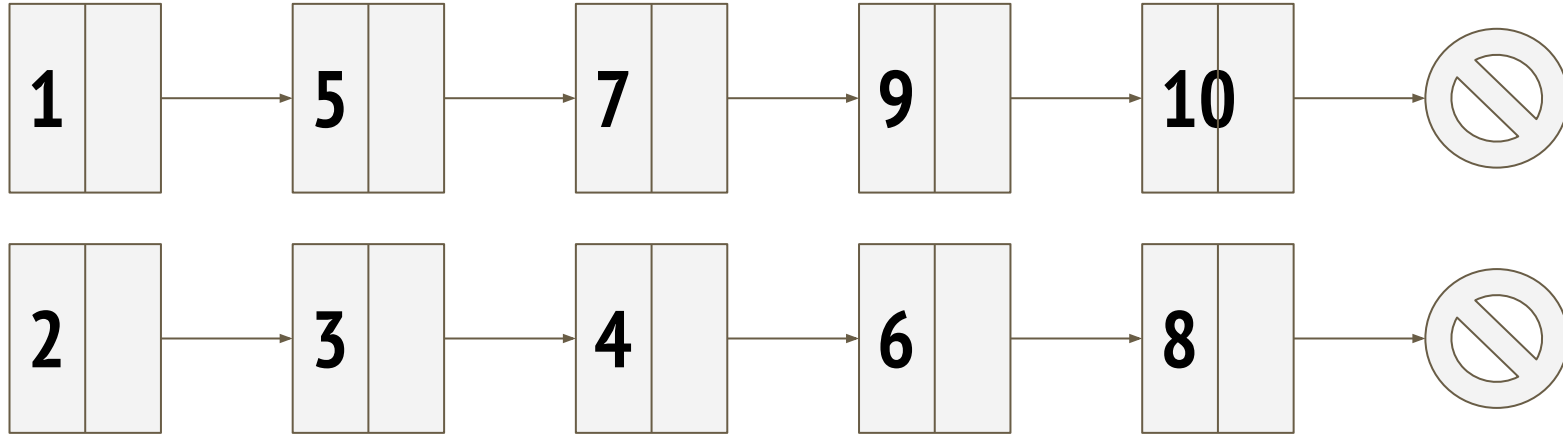
Linked Lists : Reverse List

Singly Linked List: Reverse List

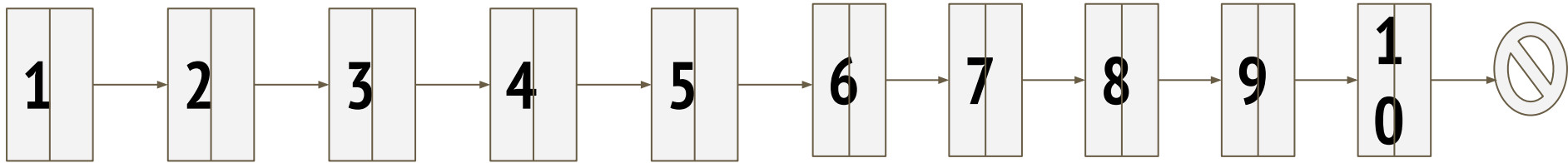
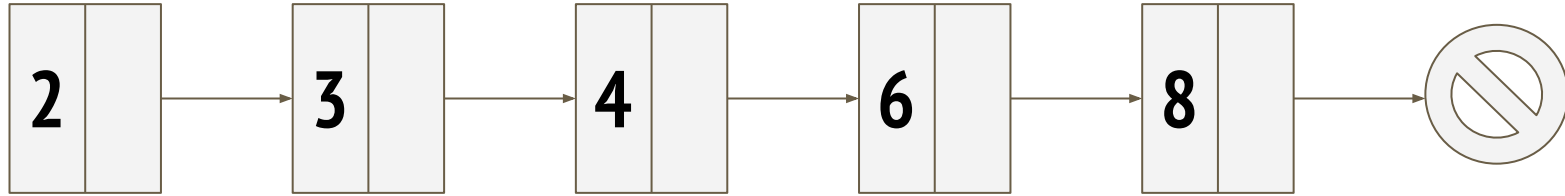
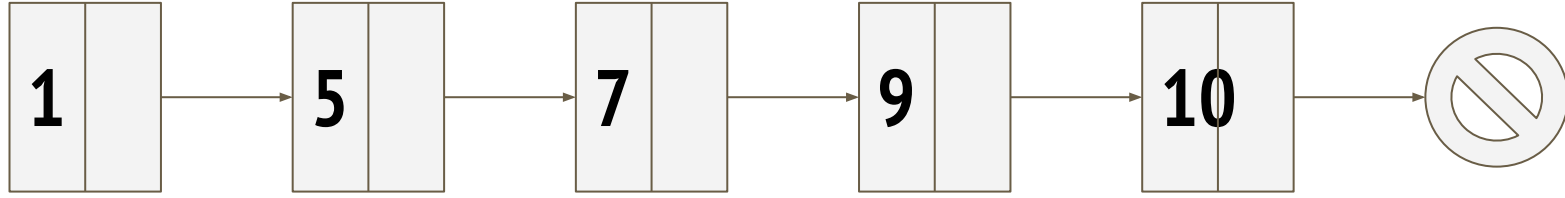


Linked Lists : Merge Two Sorted Linked Lists

Singly Linked List: Merge Two Sorted Linked Lists

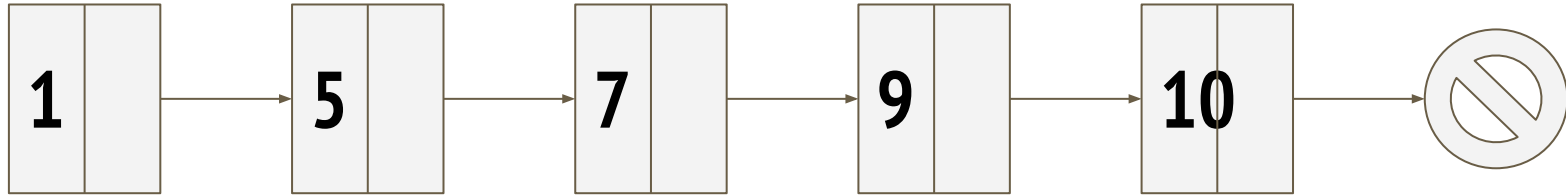


Singly Linked List: Merge Two Sorted Linked Lists



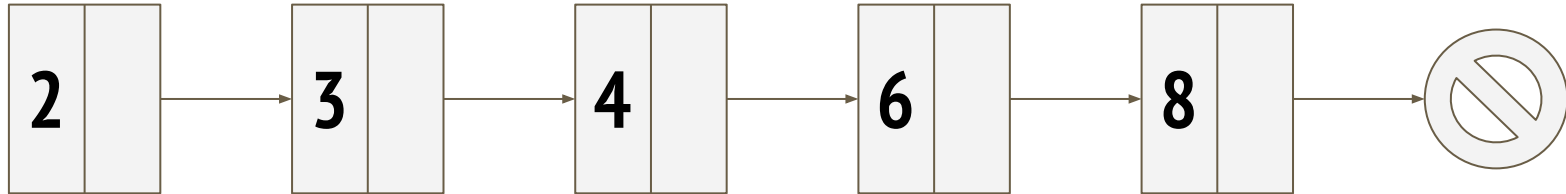
Singly Linked List: Merge Two Sorted Linked Lists

P

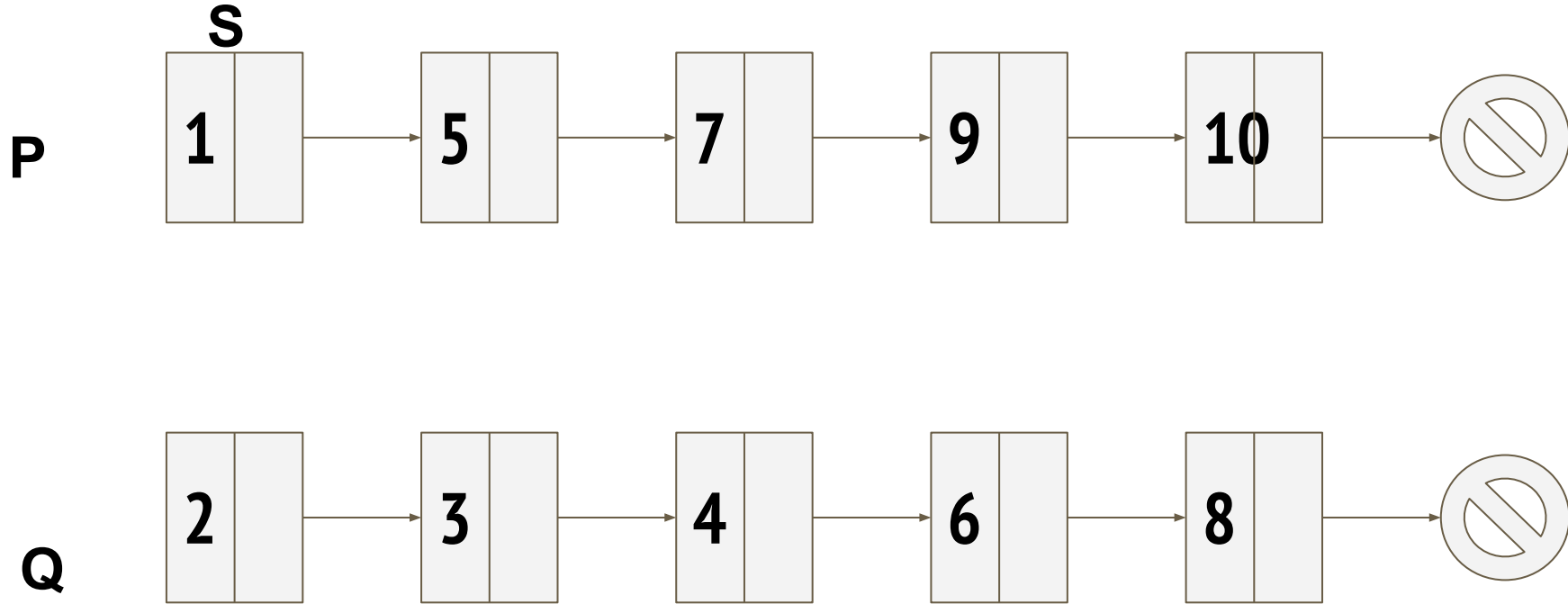


S

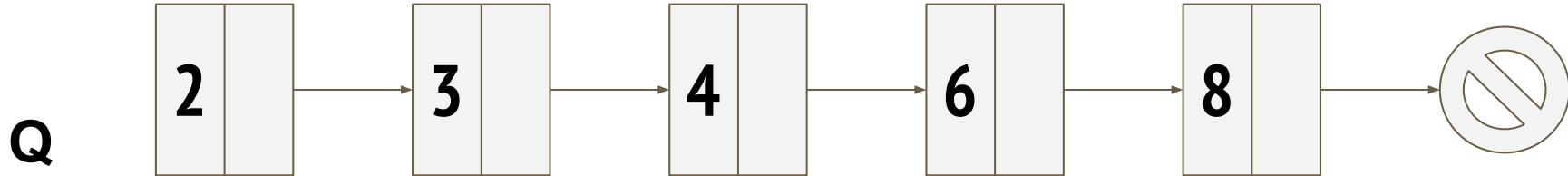
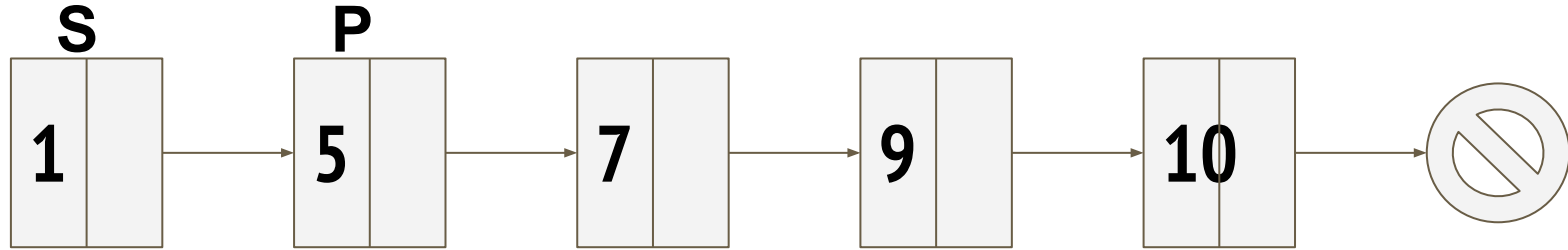
Q



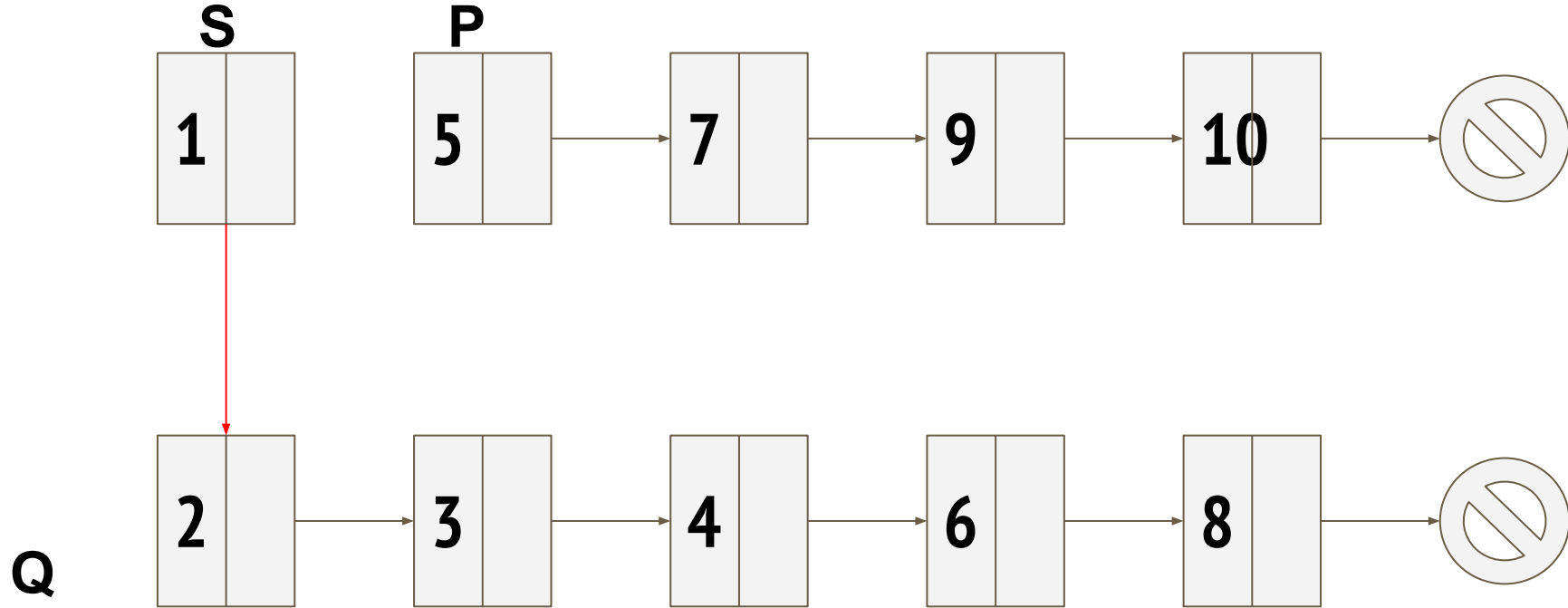
Singly Linked List: Merge Two Sorted Linked Lists



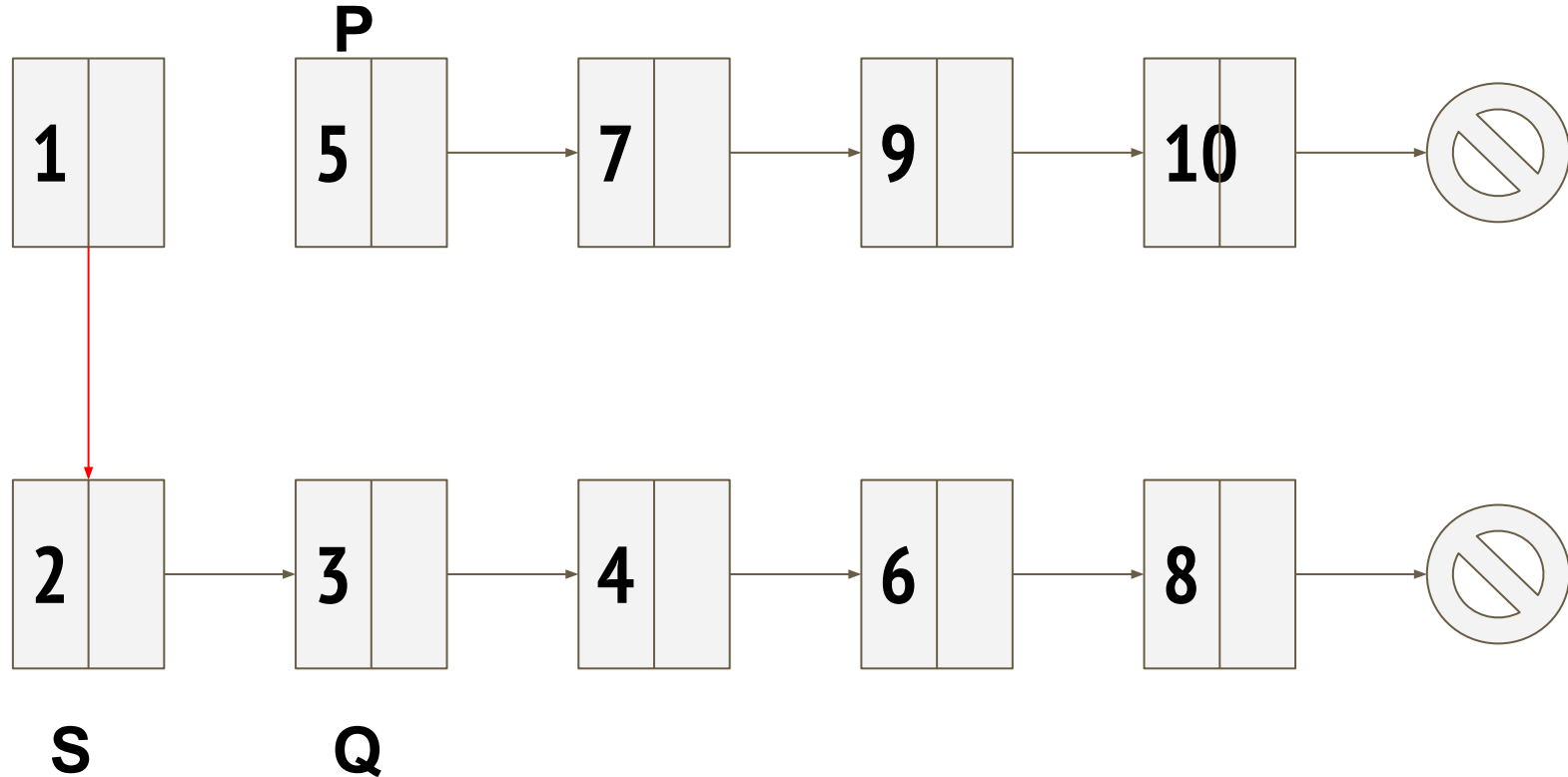
Singly Linked List: Merge Two Sorted Linked Lists



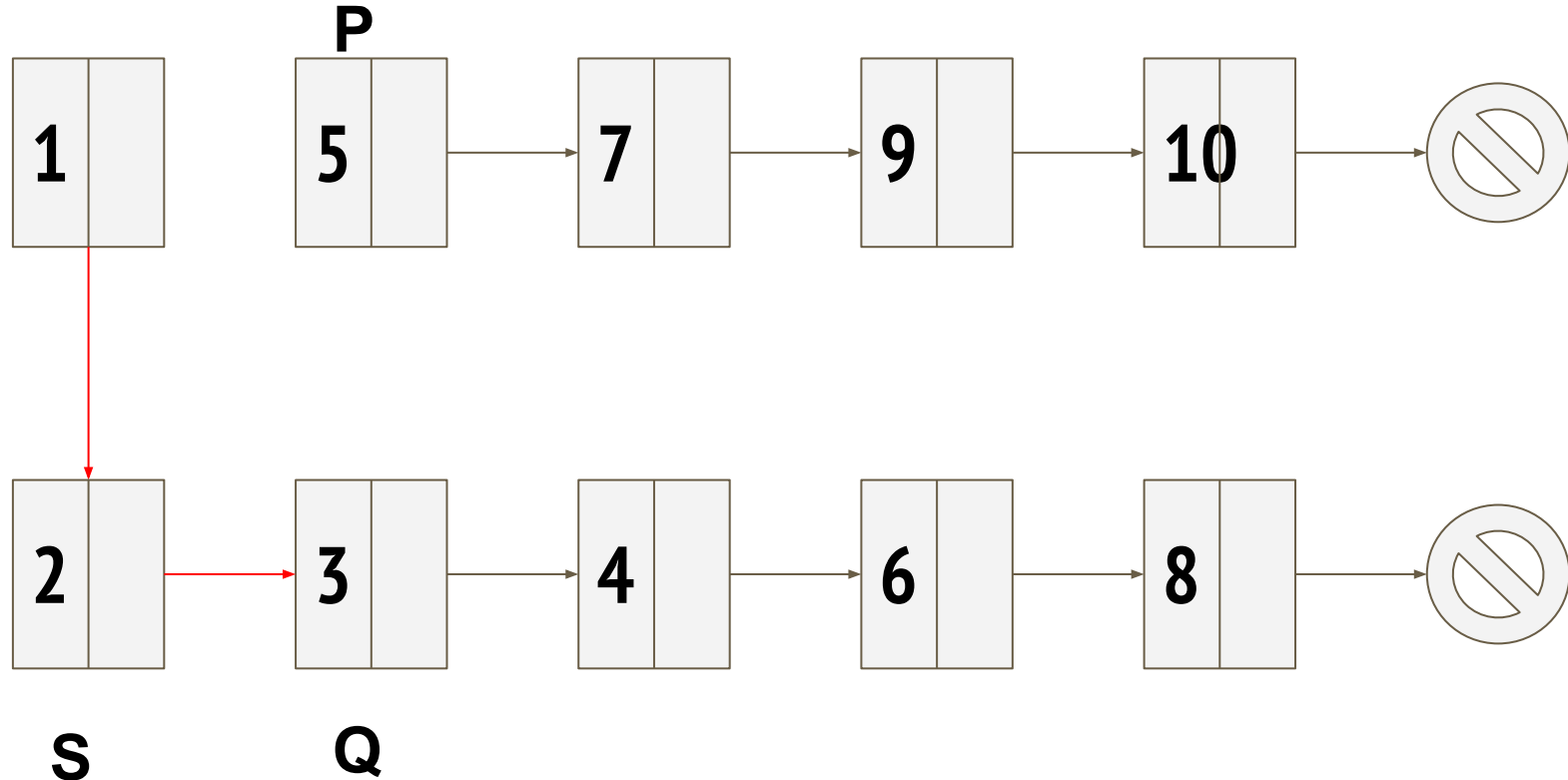
Singly Linked List: Merge Two Sorted Linked Lists



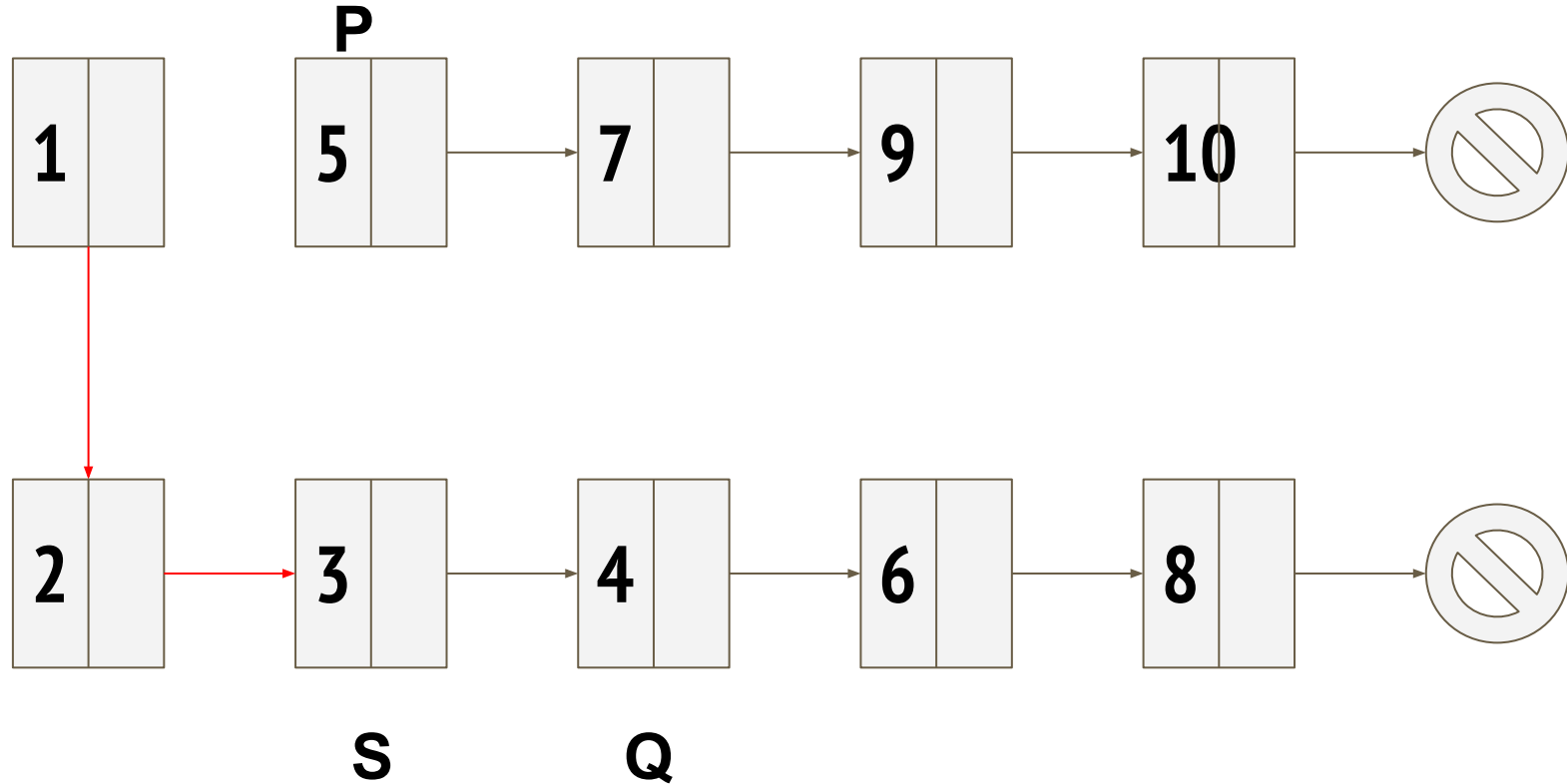
Singly Linked List: Merge Two Sorted Linked Lists



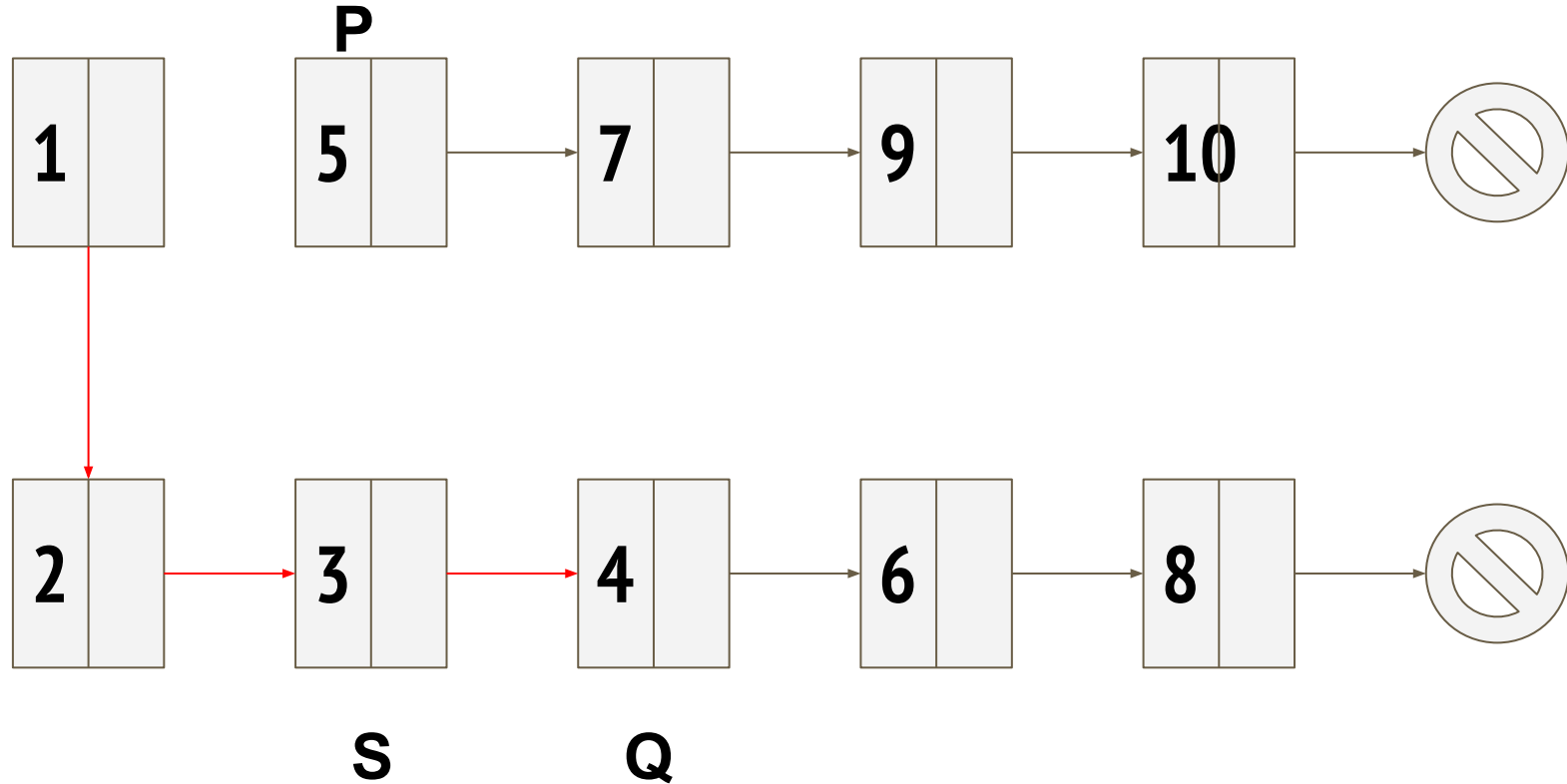
Singly Linked List: Merge Two Sorted Linked Lists



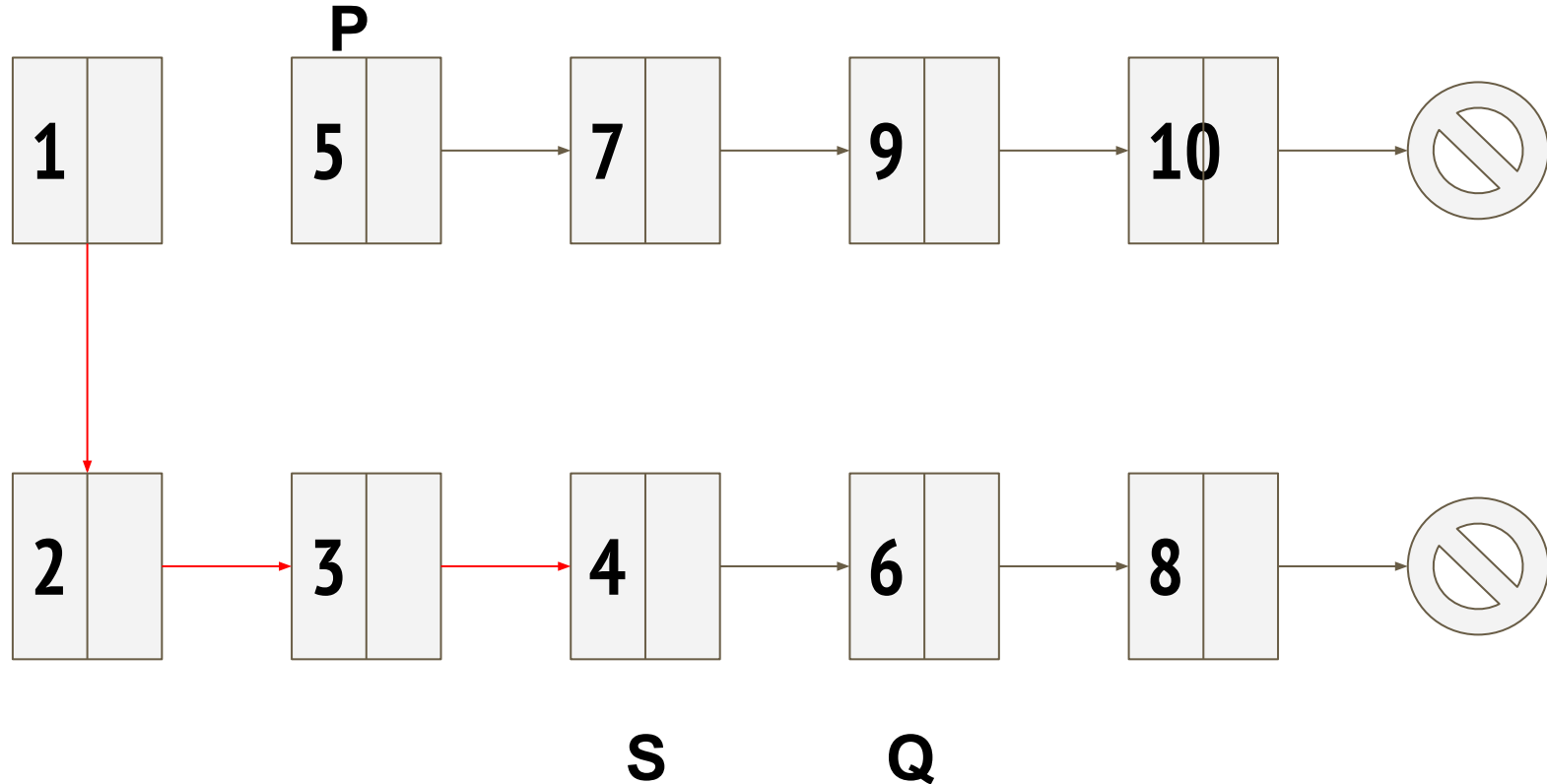
Singly Linked List: Merge Two Sorted Linked Lists



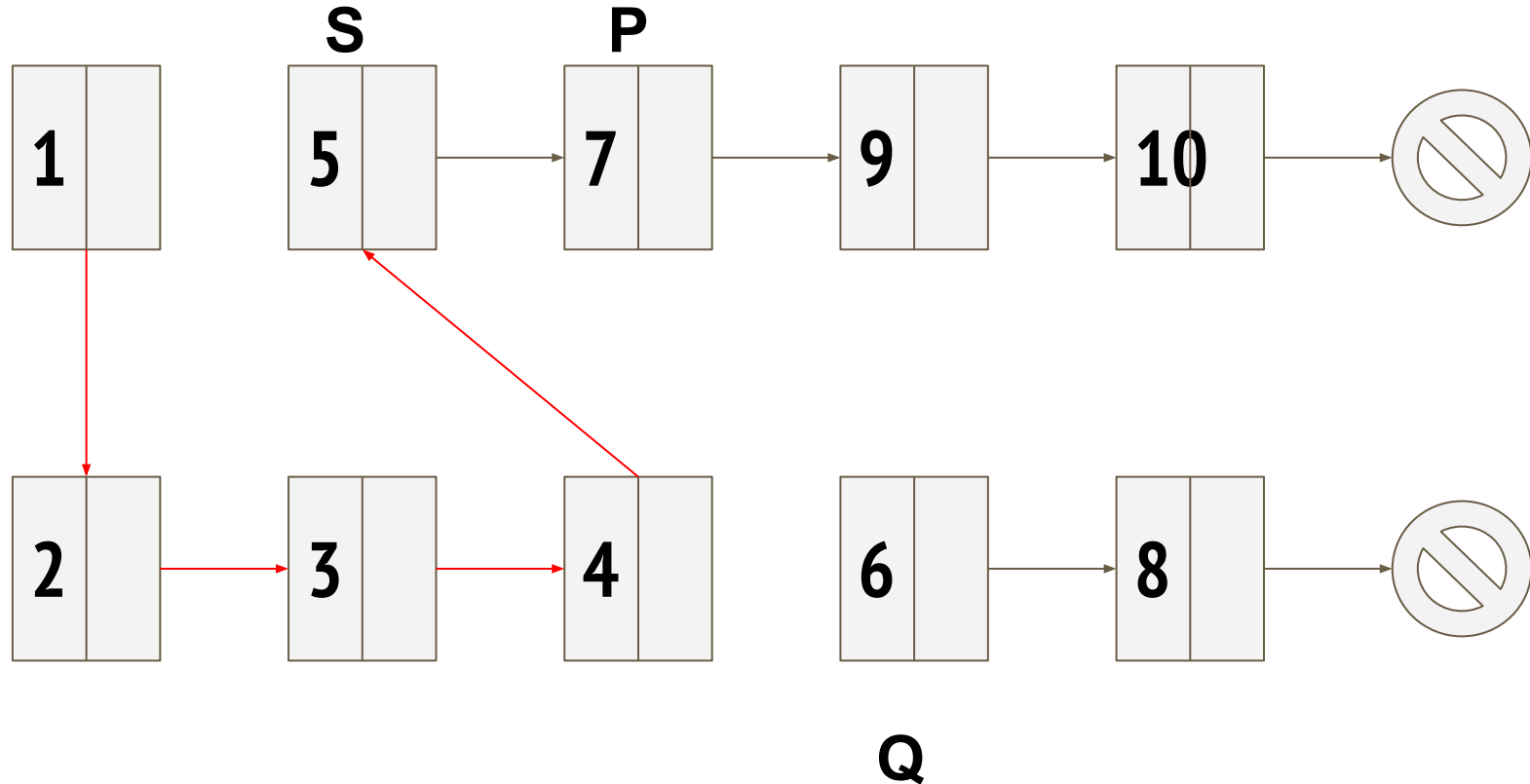
Singly Linked List: Merge Two Sorted Linked Lists



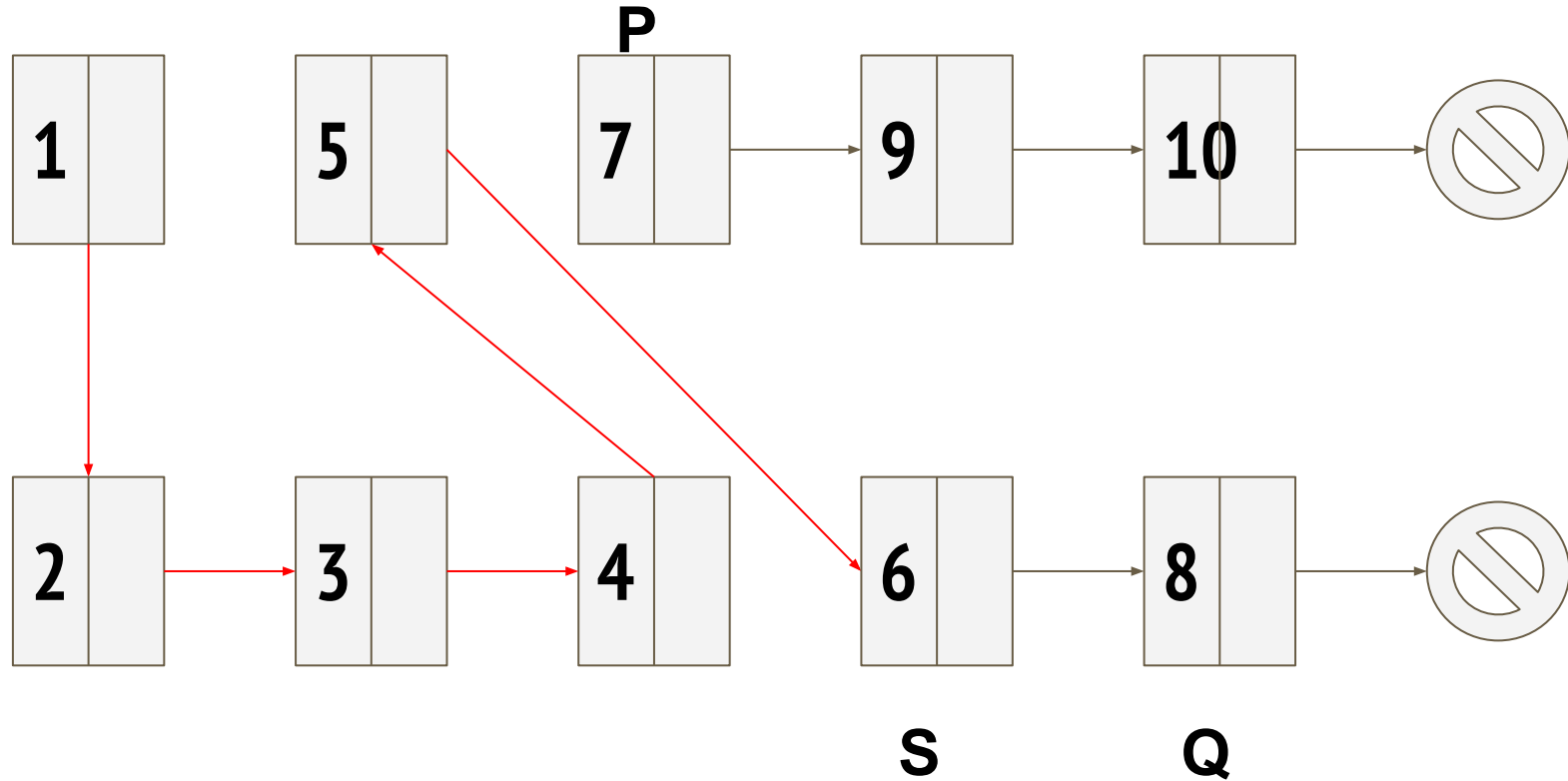
Singly Linked List: Merge Two Sorted Linked Lists



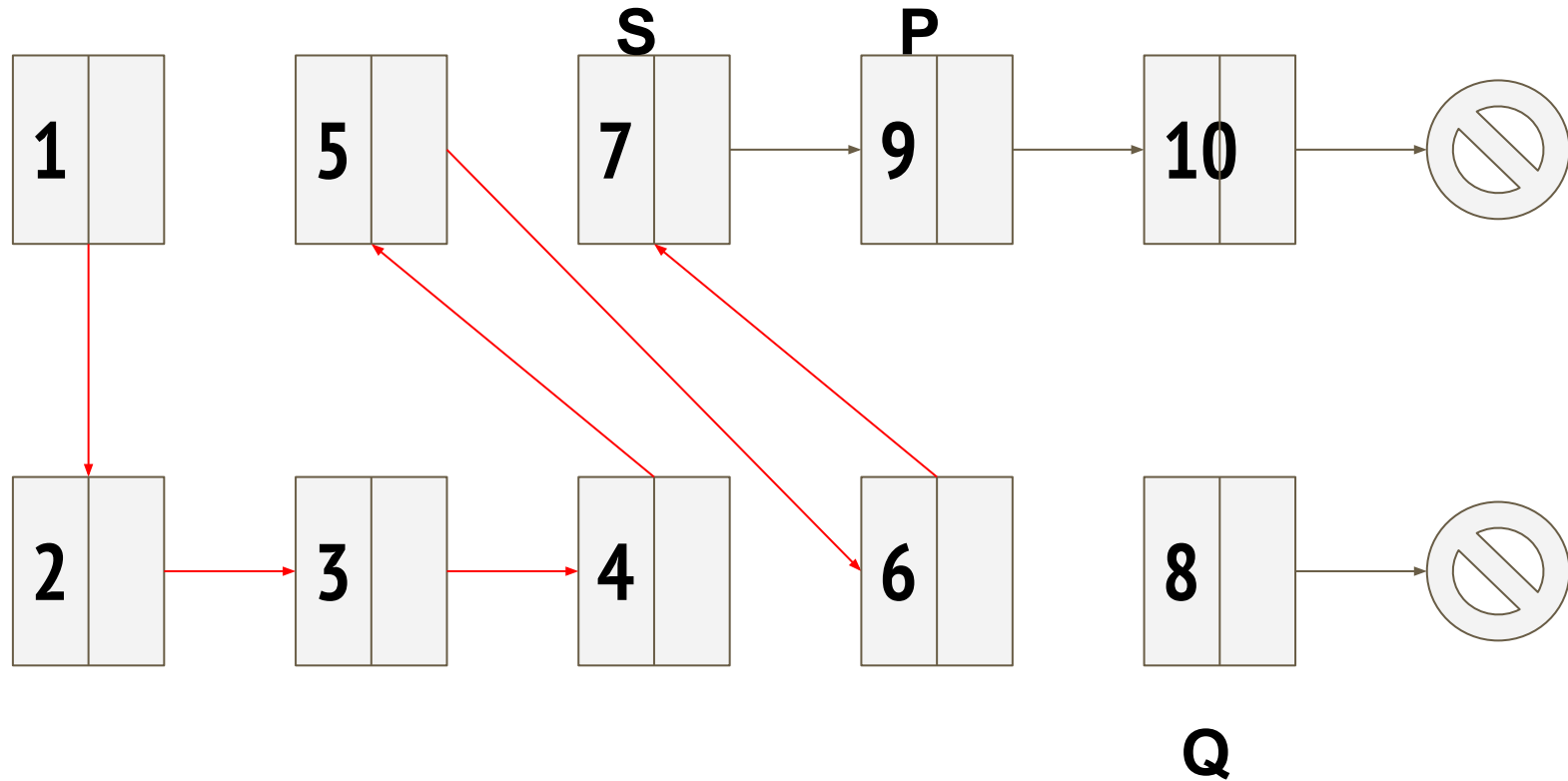
Singly Linked List: Merge Two Sorted Linked Lists



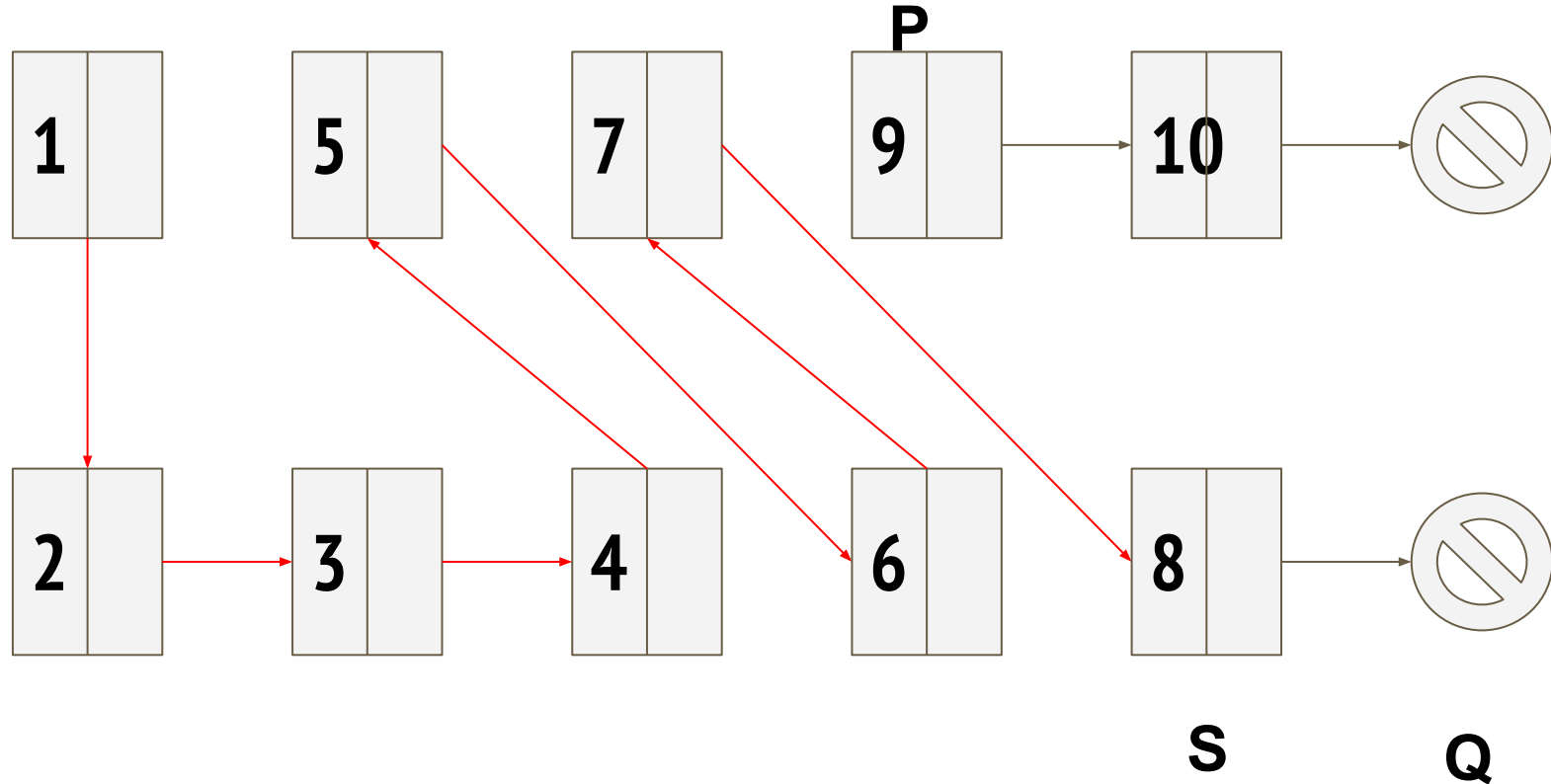
Singly Linked List: Merge Two Sorted Linked Lists



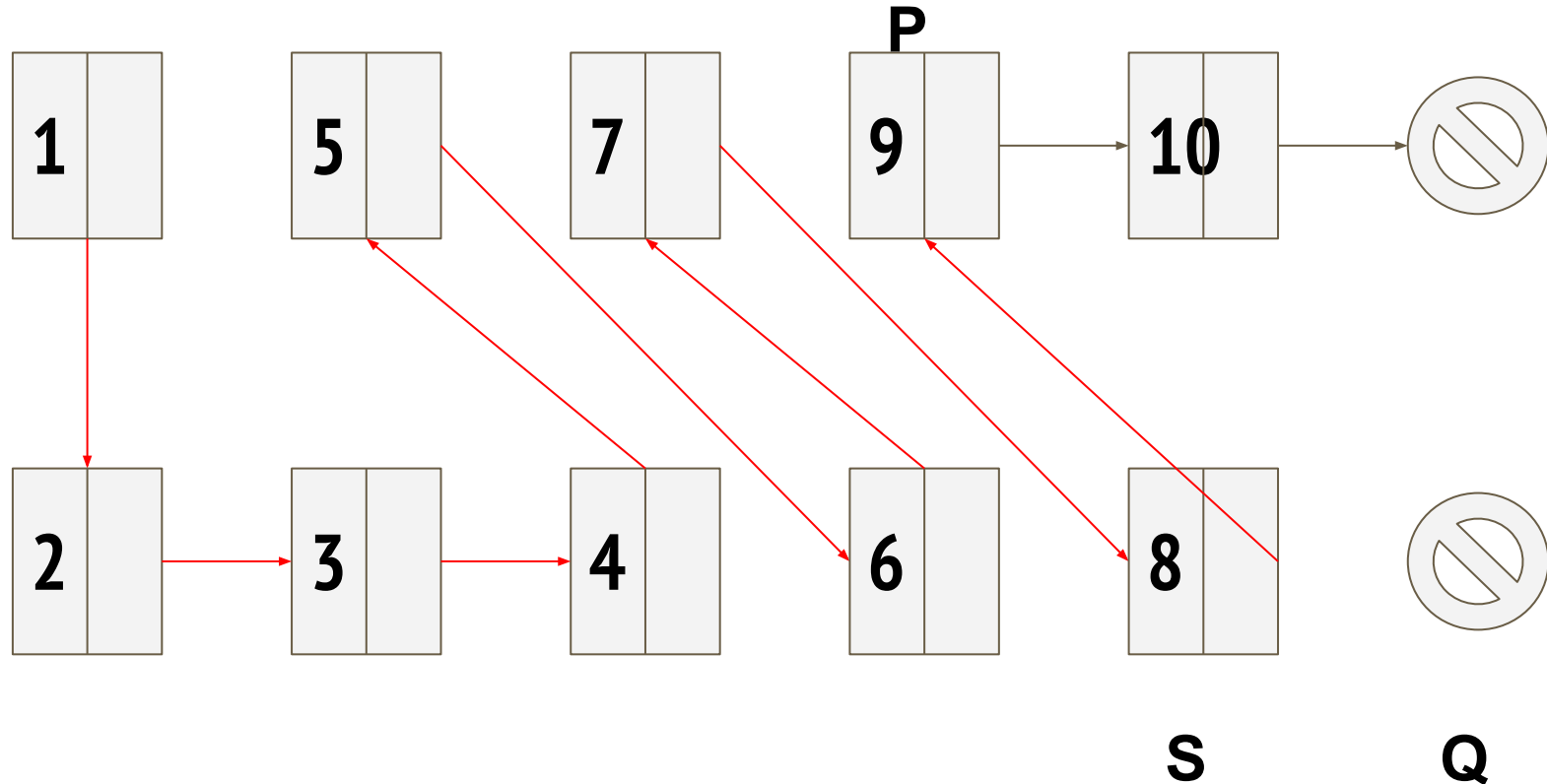
Singly Linked List: Merge Two Sorted Linked Lists



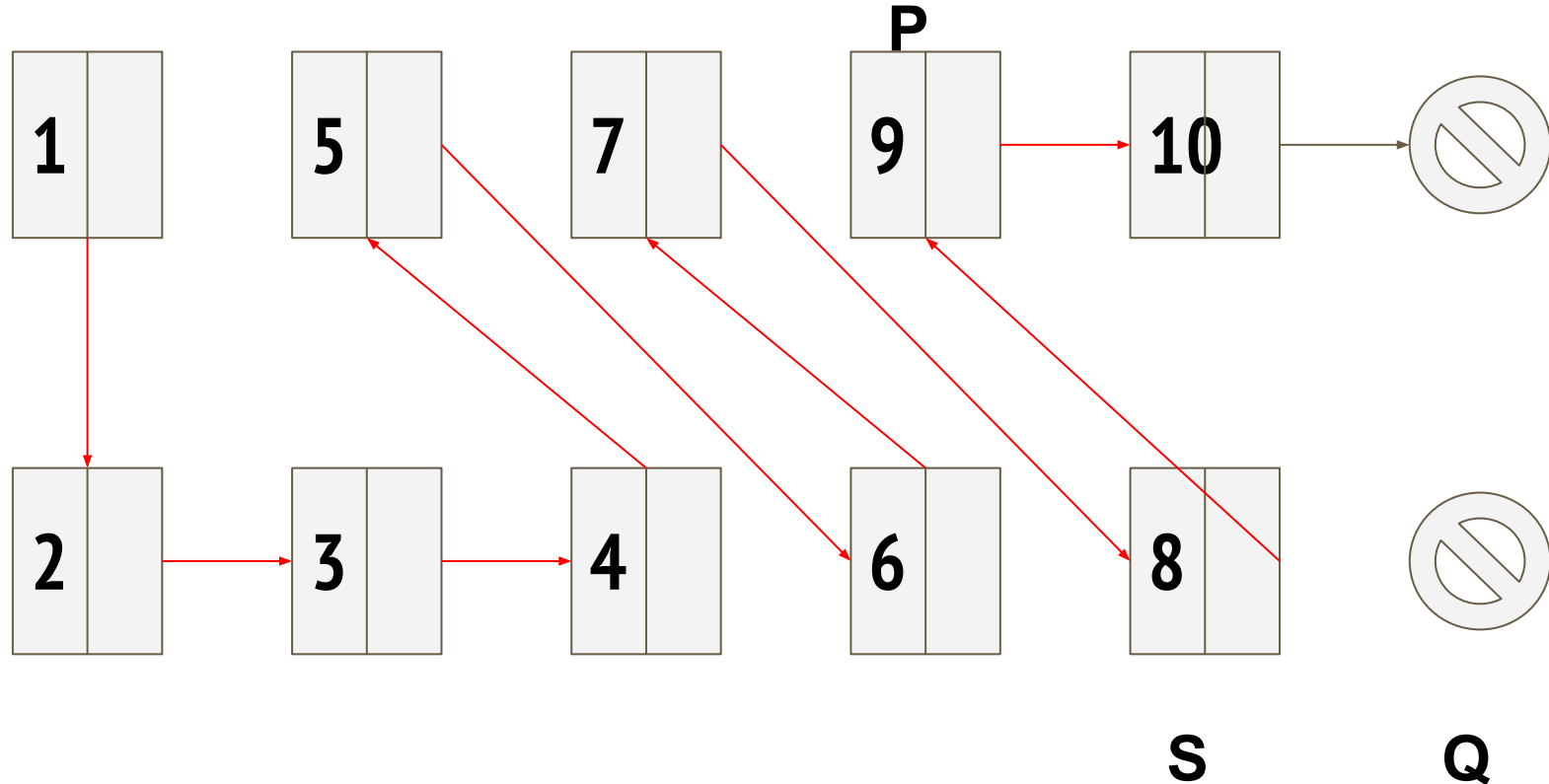
Singly Linked List: Merge Two Sorted Linked Lists



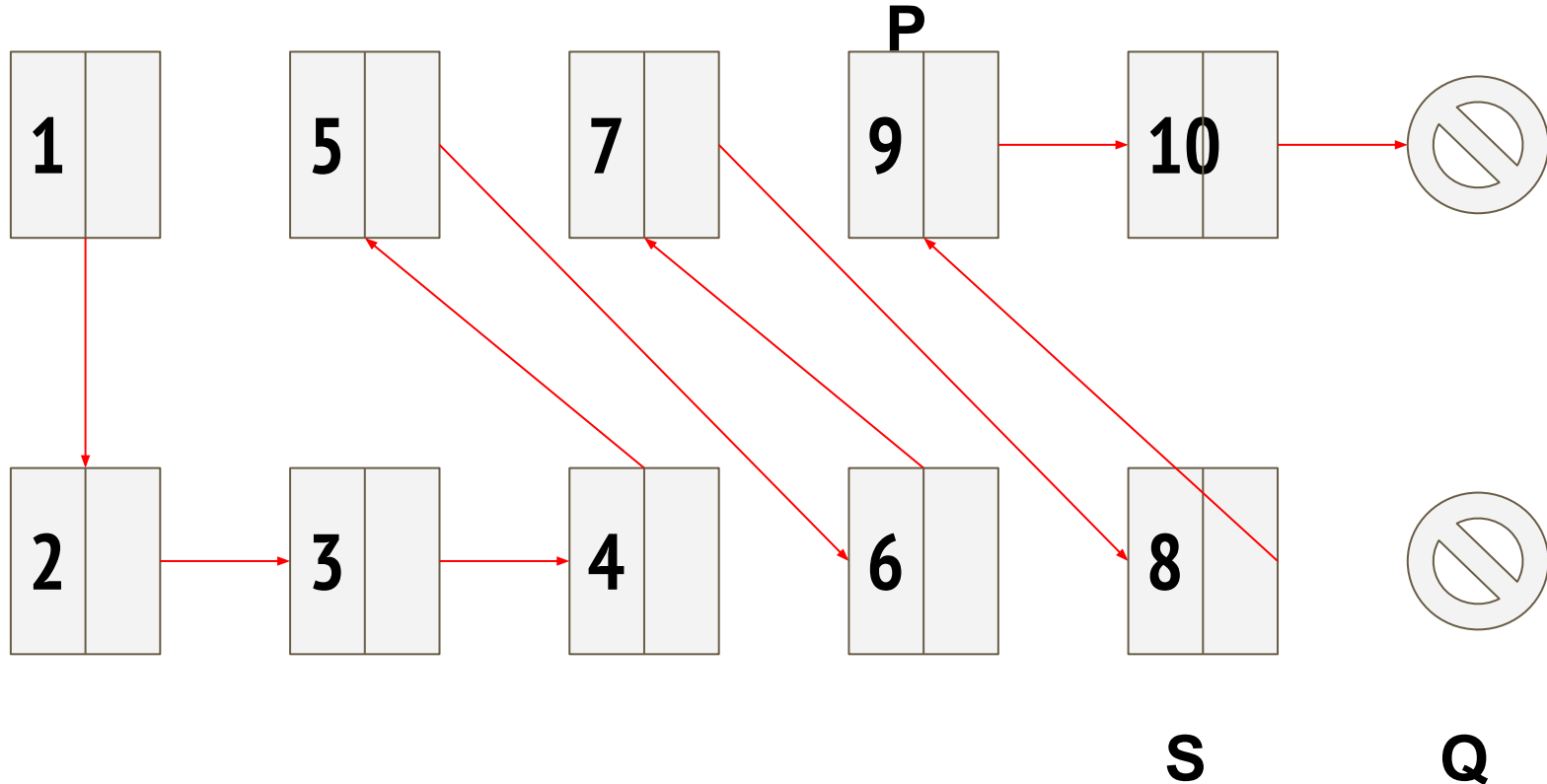
Singly Linked List: Merge Two Sorted Linked Lists



Singly Linked List: Merge Two Sorted Linked Lists

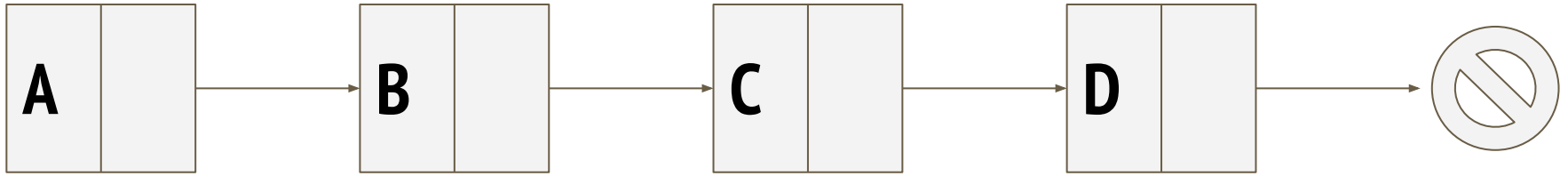


Singly Linked List: Merge Two Sorted Linked Lists

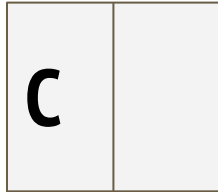


Linked Lists : Find Nth-to-last Node

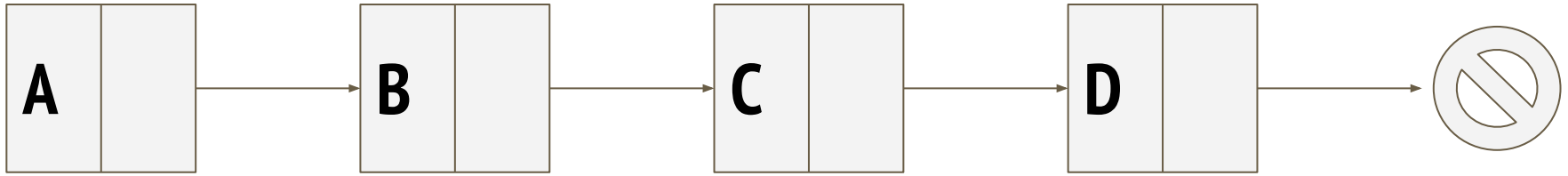
Singly Linked List: Nth-to-last Node



Second to last node: ($n = 2$)

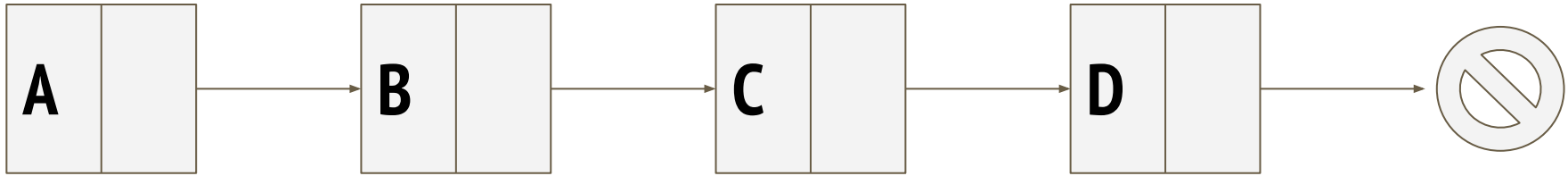


Singly Linked List: Nth-to-last Node -- Method 1



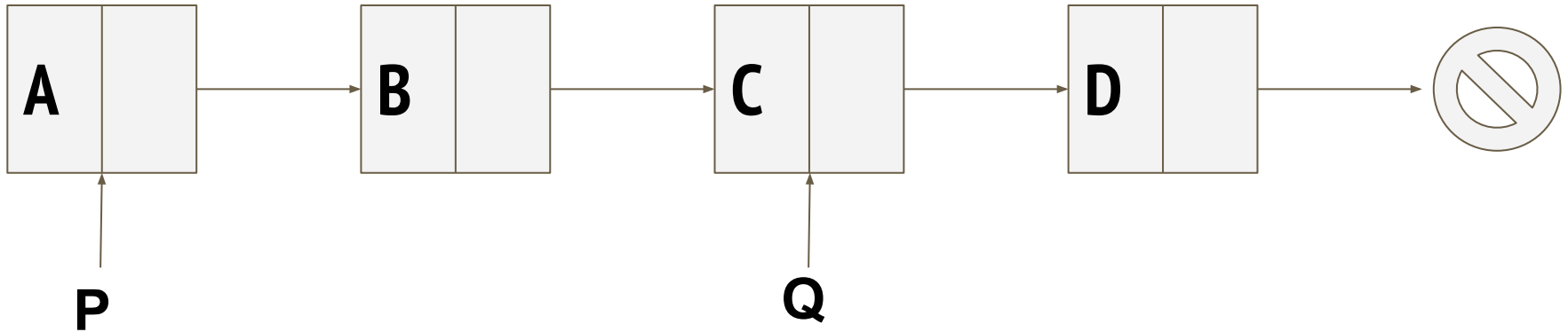
1. Calculate length of linked list.
2. Count down from the total length until "n" is reached.

Singly Linked List: Nth-to-last Node -- Method 2



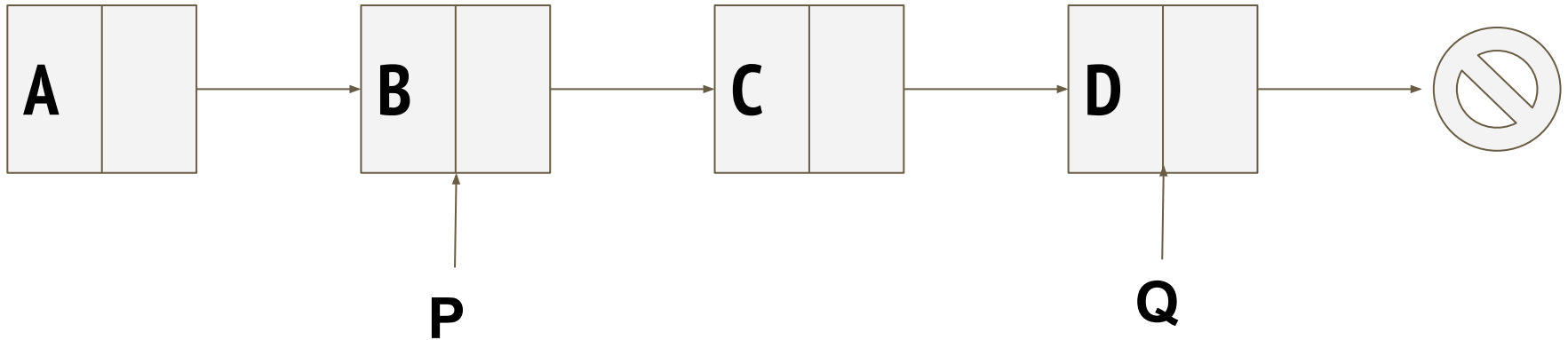
1. Two pointers:
 - a. P: head node
 - b. Q: n nodes beyond head node

Singly Linked List: Nth-to-last Node -- Method 2



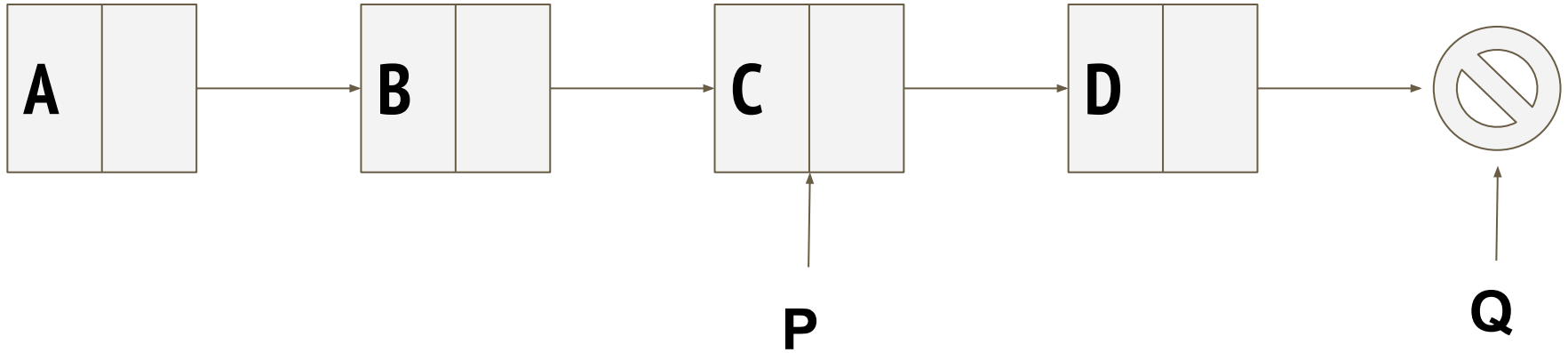
Example: $n = 2$

Singly Linked List: Nth-to-last Node -- Method 2



Example: $n = 2$

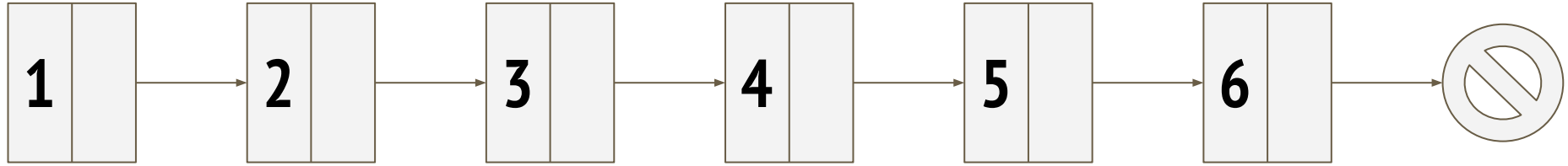
Singly Linked List: Nth-to-last Node -- Method 2



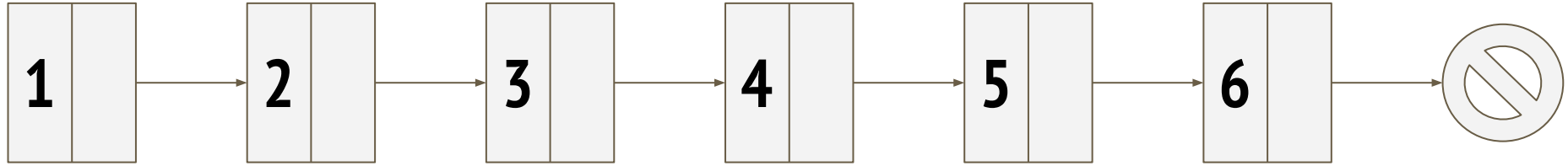
Example: $n = 2$

Linked Lists : Rotate List

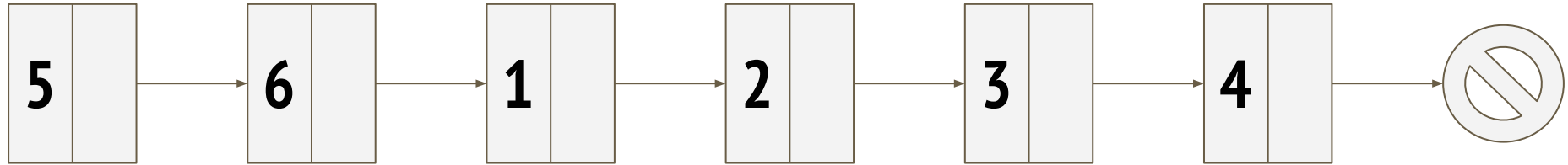
Singly Linked List: Rotate



Singly Linked List: Rotate

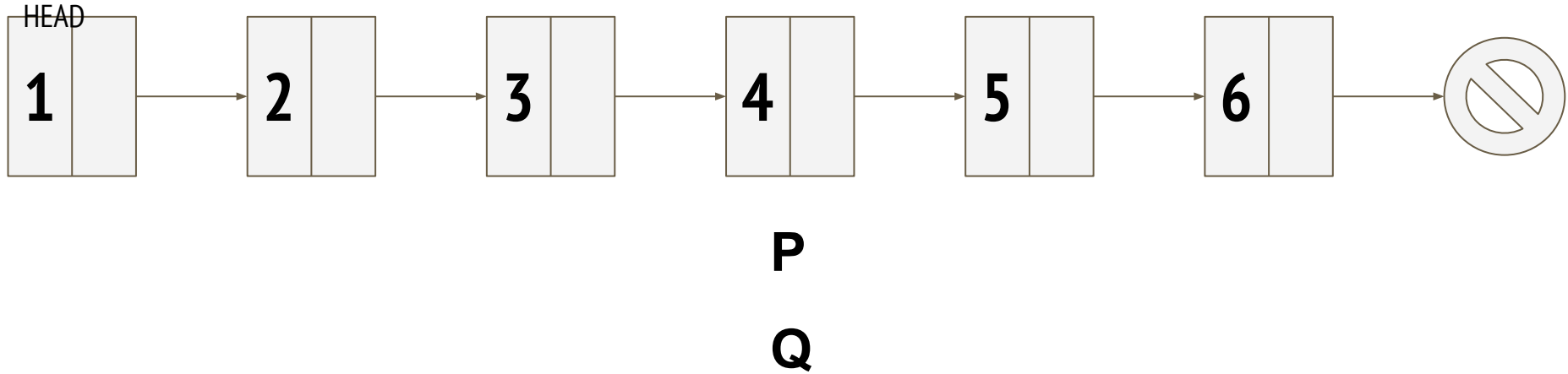


Example: $k = 4$



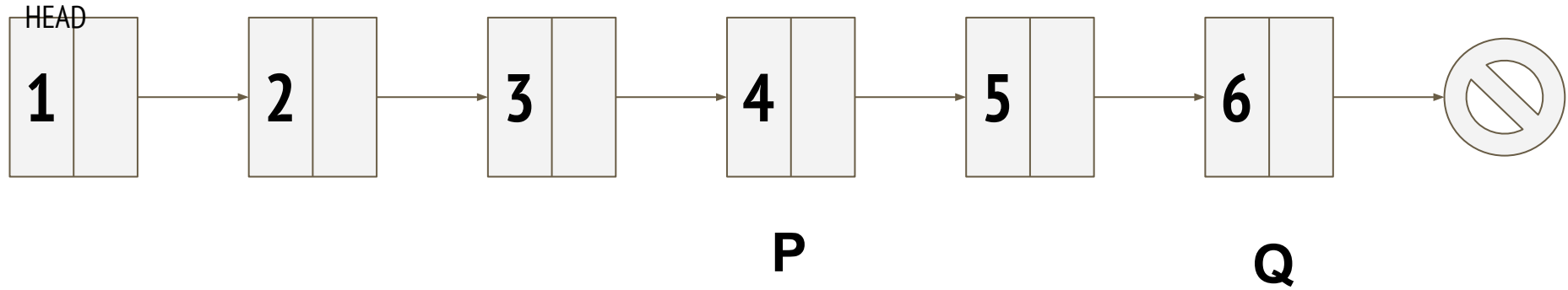
Singly Linked List: Rotate

Example -- $k = 4$



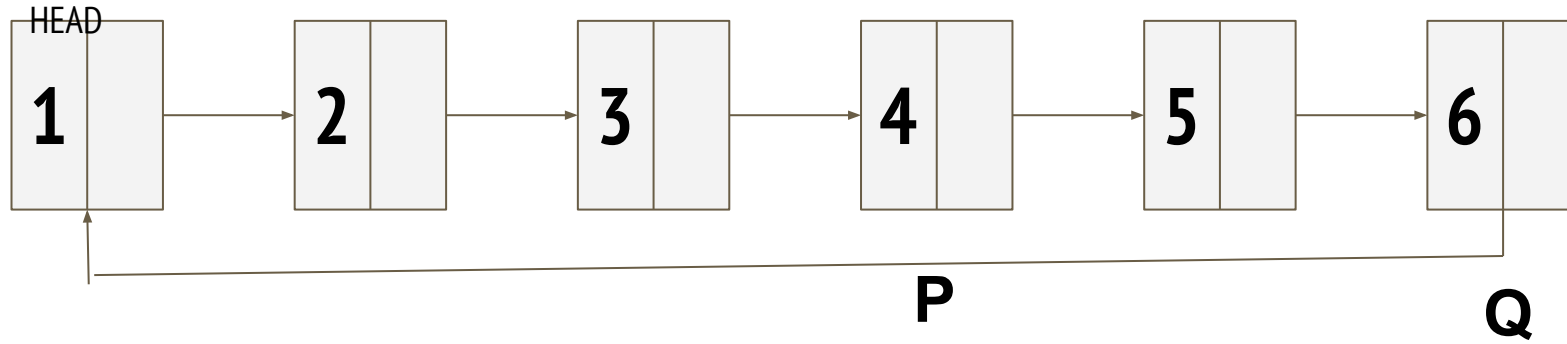
Singly Linked List: Rotate

Example -- $k = 4$



Singly Linked List: Rotate

Example -- $k = 4$



Circular Linked Lists



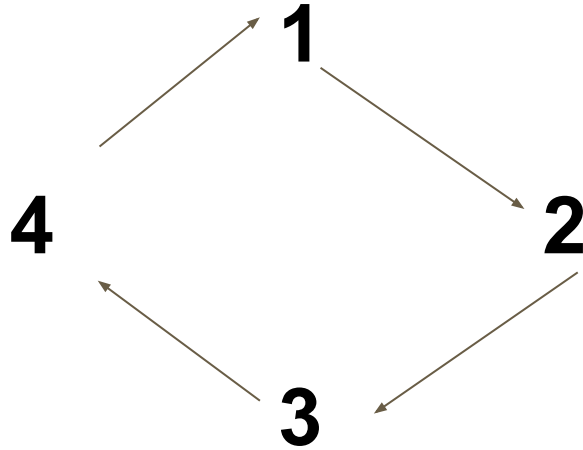
Introduction



Circular Linked Lists: Josephus Problem

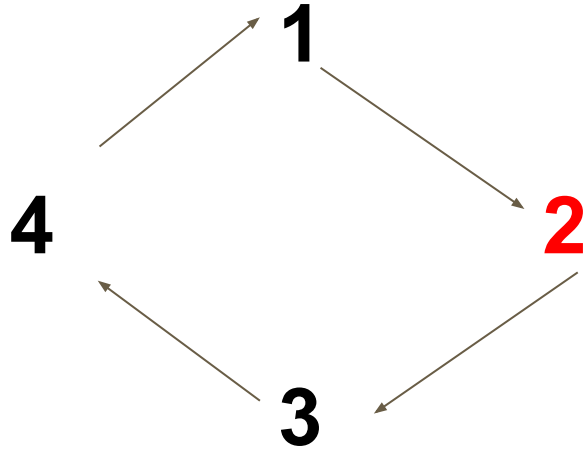
Circular Linked List: Josephus Problem

Step size = 2



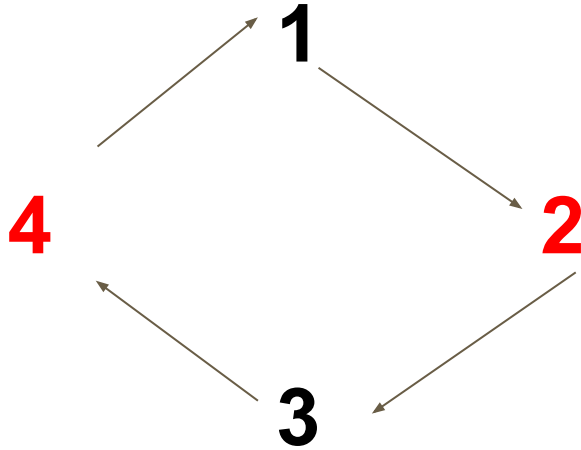
Circular Linked List: Josephus Problem

Step size = 2



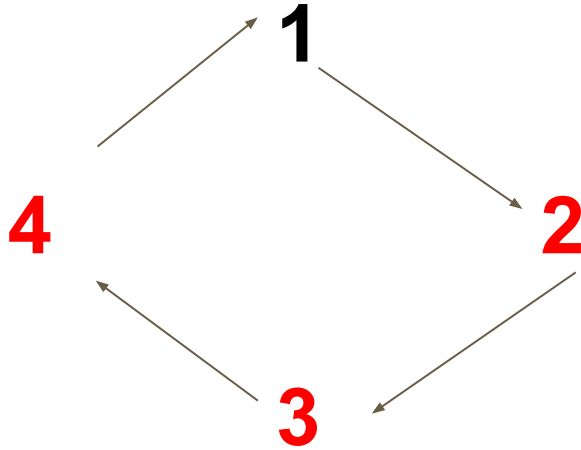
Circular Linked List: Josephus Problem

Step size = 2



Circular Linked List: Josephus Problem

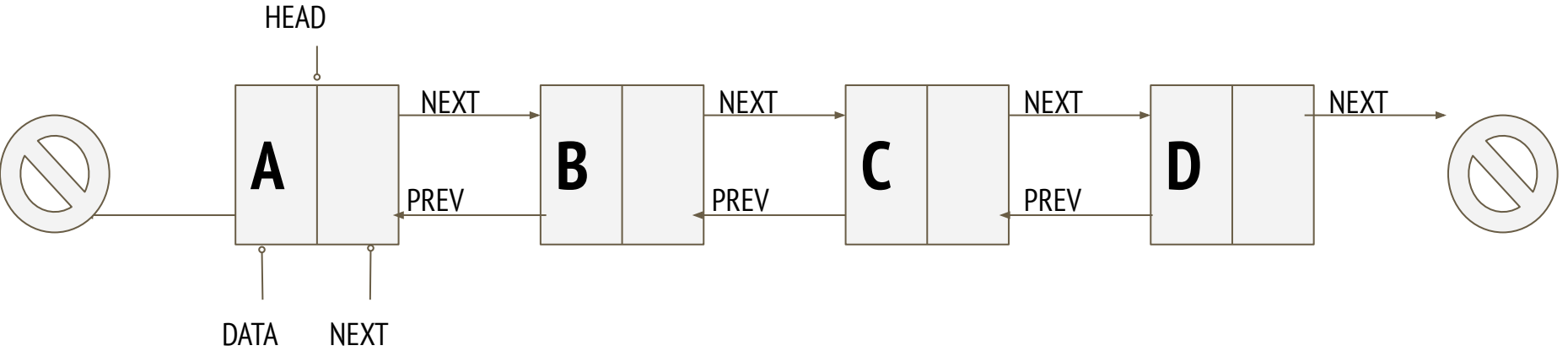
Step size = 2



Doubly Linked Lists

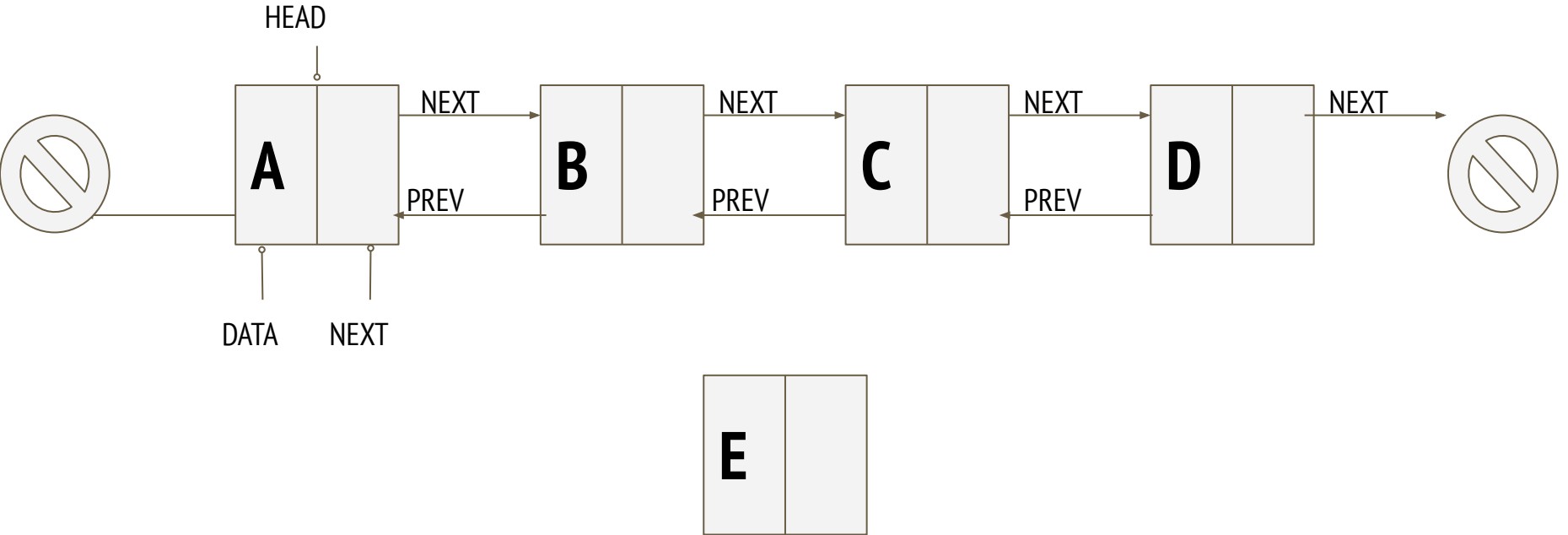
Introduction

Doubly Linked List



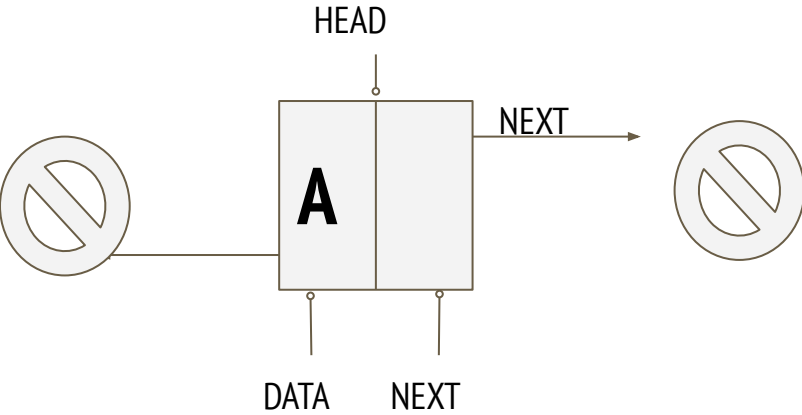
Doubly Linked Lists: Add after node

Doubly Linked List



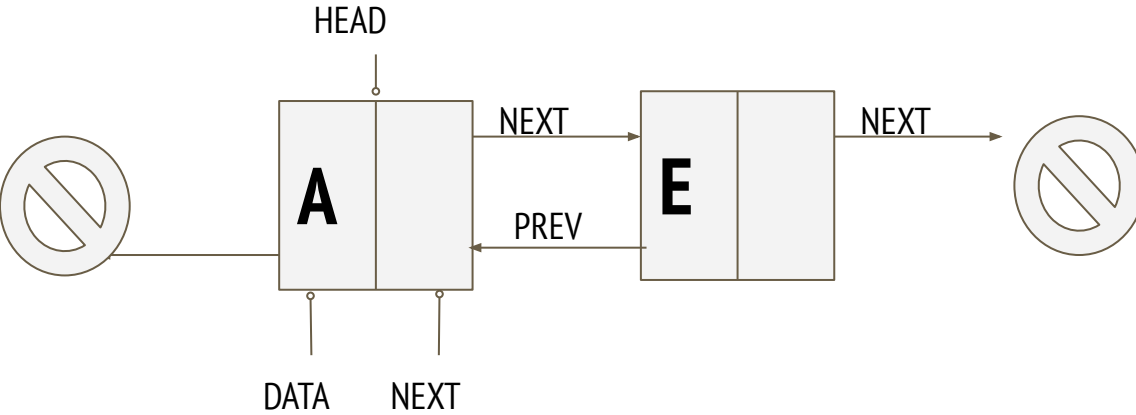
Doubly Linked List

Add after node with data "A"



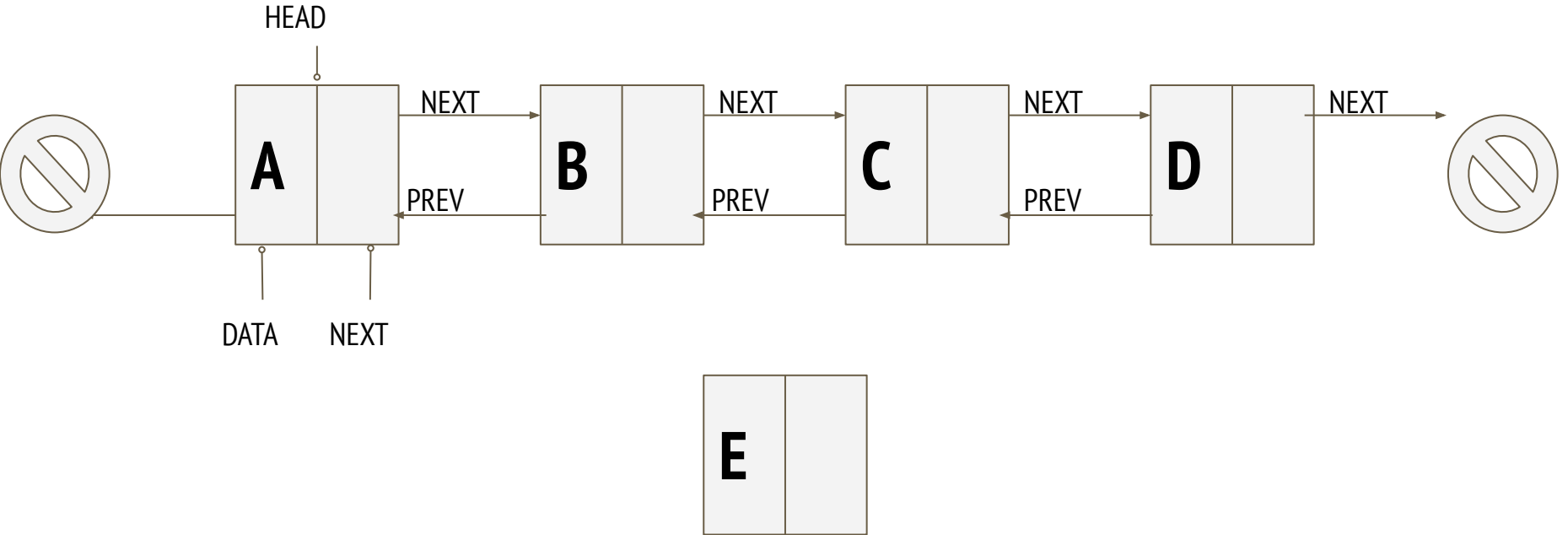
Doubly Linked List

Add after node with data "A"
Call to "append" function.



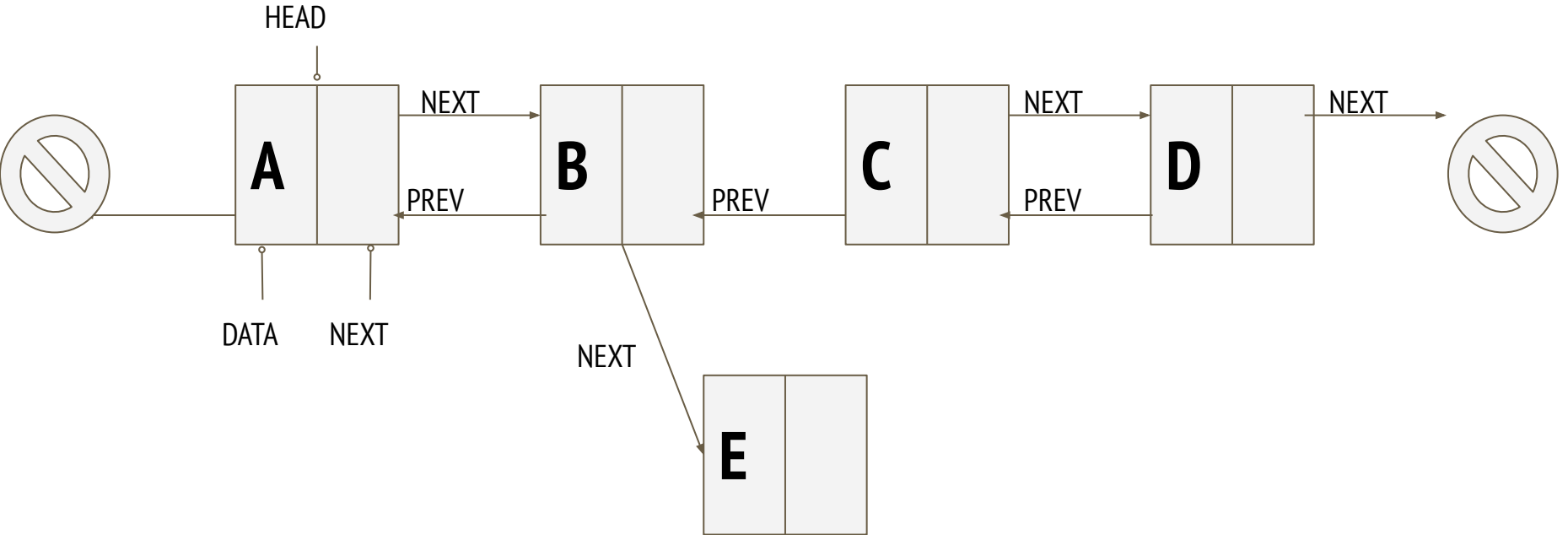
Doubly Linked List

Add after node with data "B"



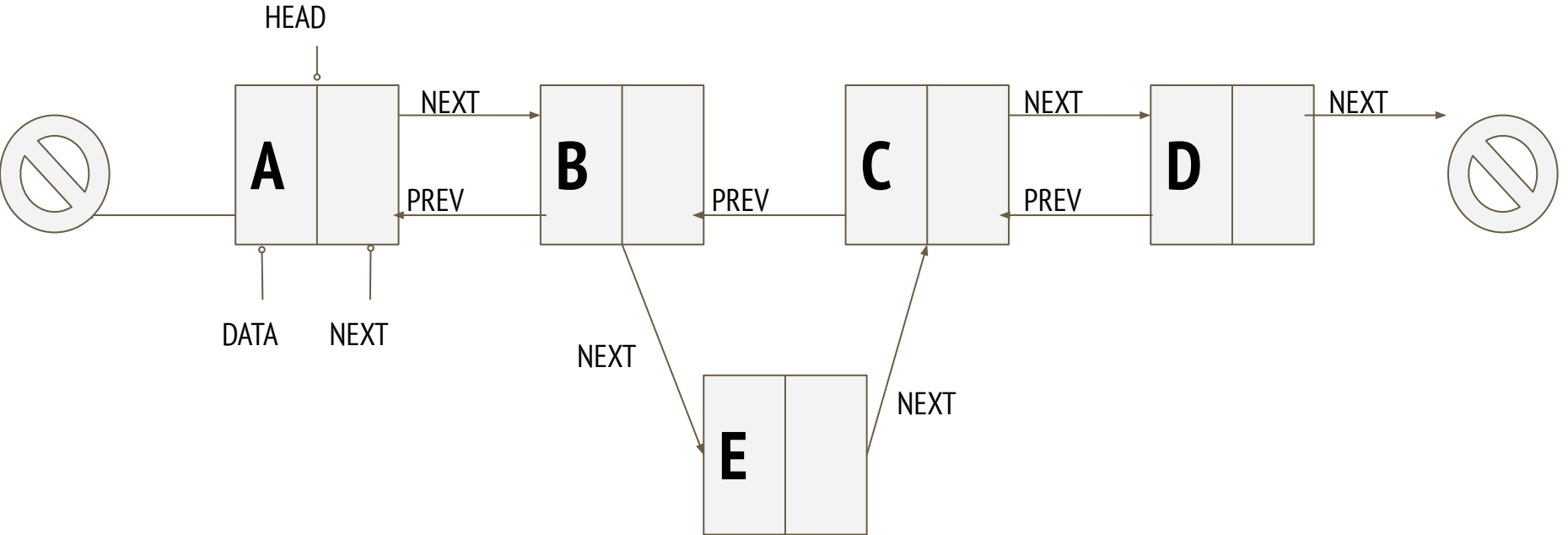
Doubly Linked List

Add after node with data "B"



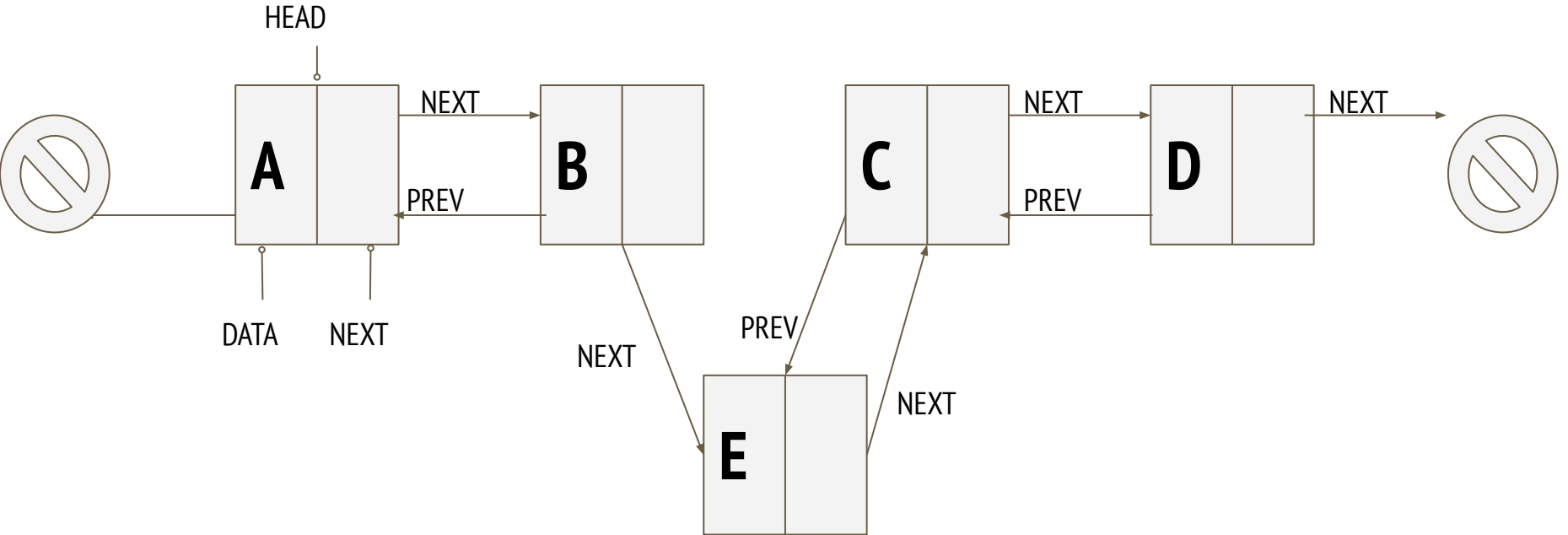
Doubly Linked List

Add after node with data "B"



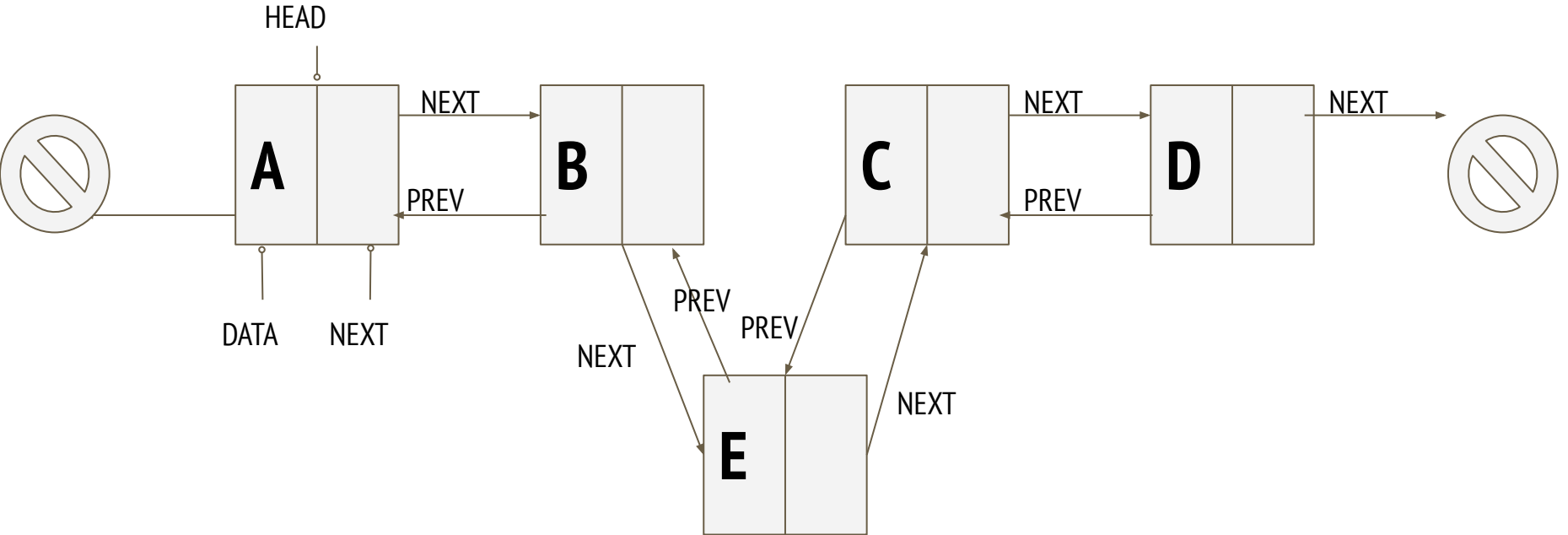
Doubly Linked List

Add after node with data "B"



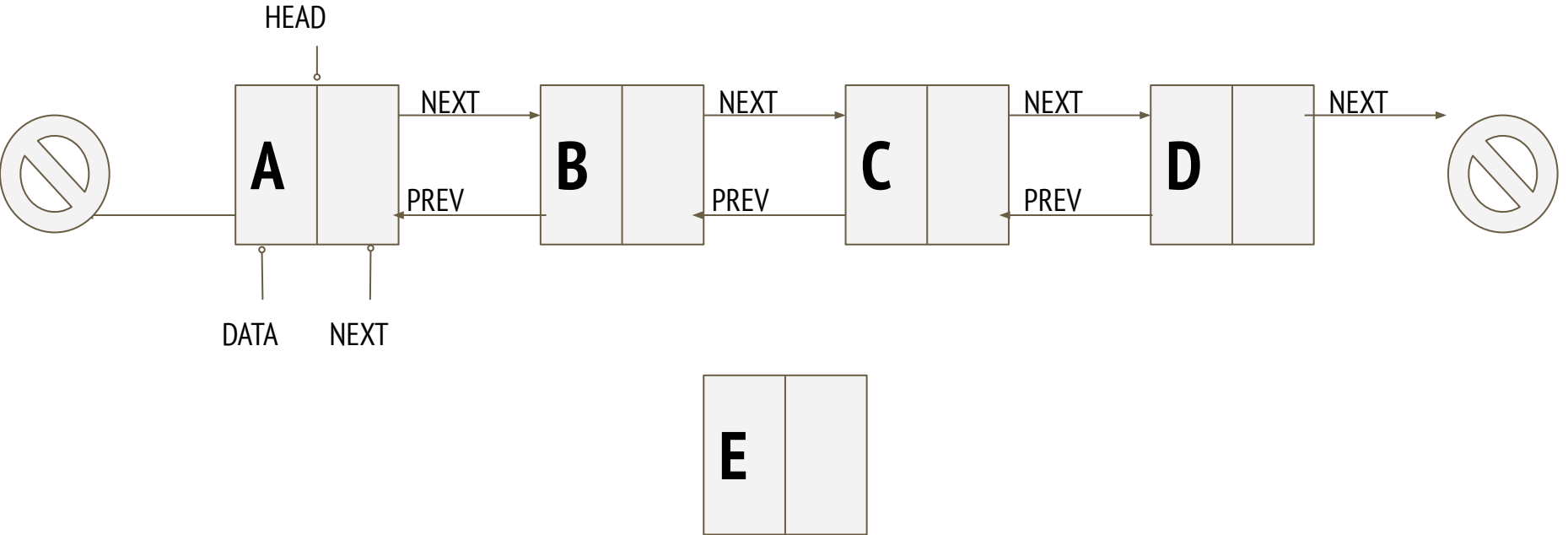
Doubly Linked List

Add after node with data "B"



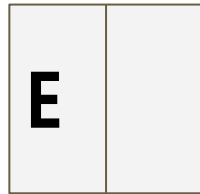
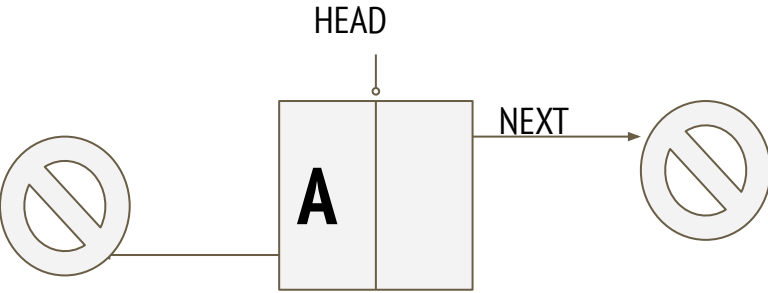
Doubly Linked Lists: Add before node

Doubly Linked List



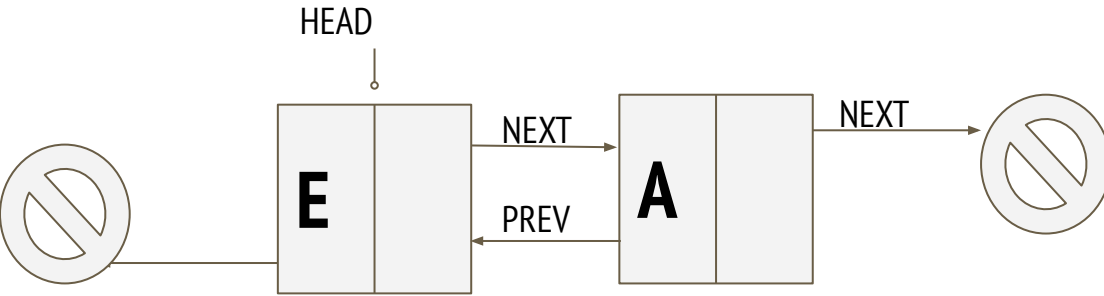
Doubly Linked List

Add before node with data "A"



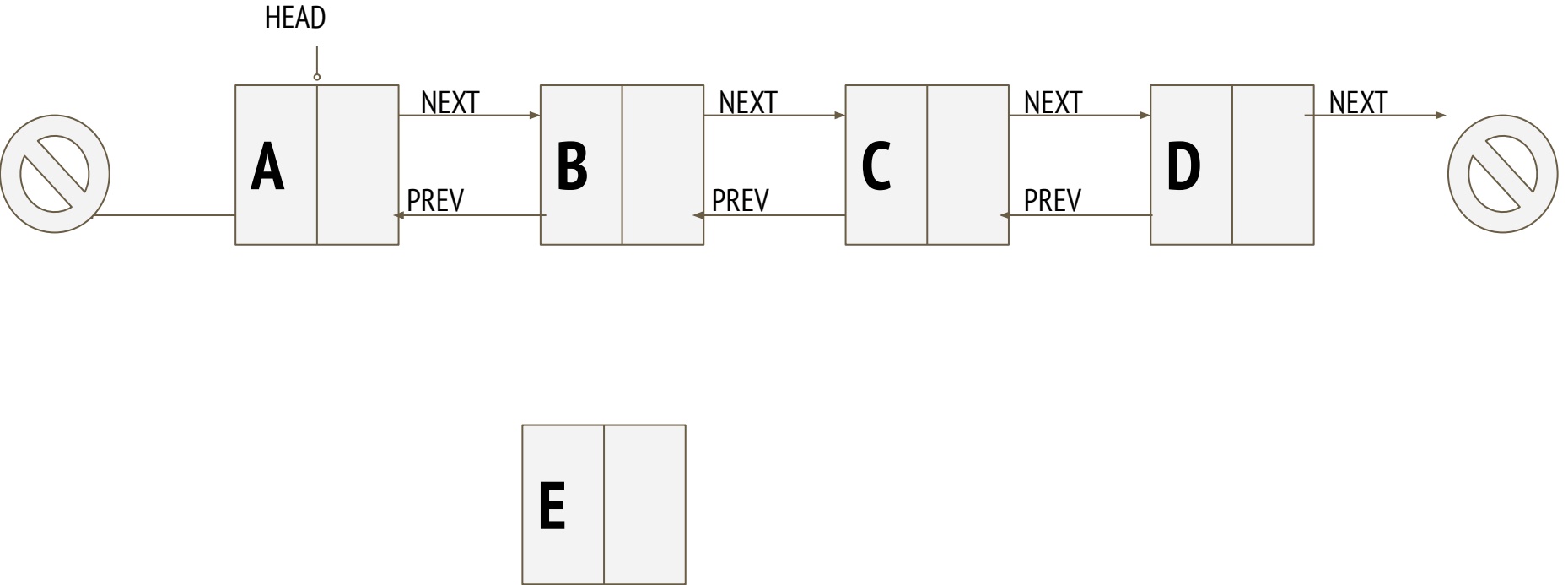
Doubly Linked List

Add before node with data "A"
Call to "prepend" function.



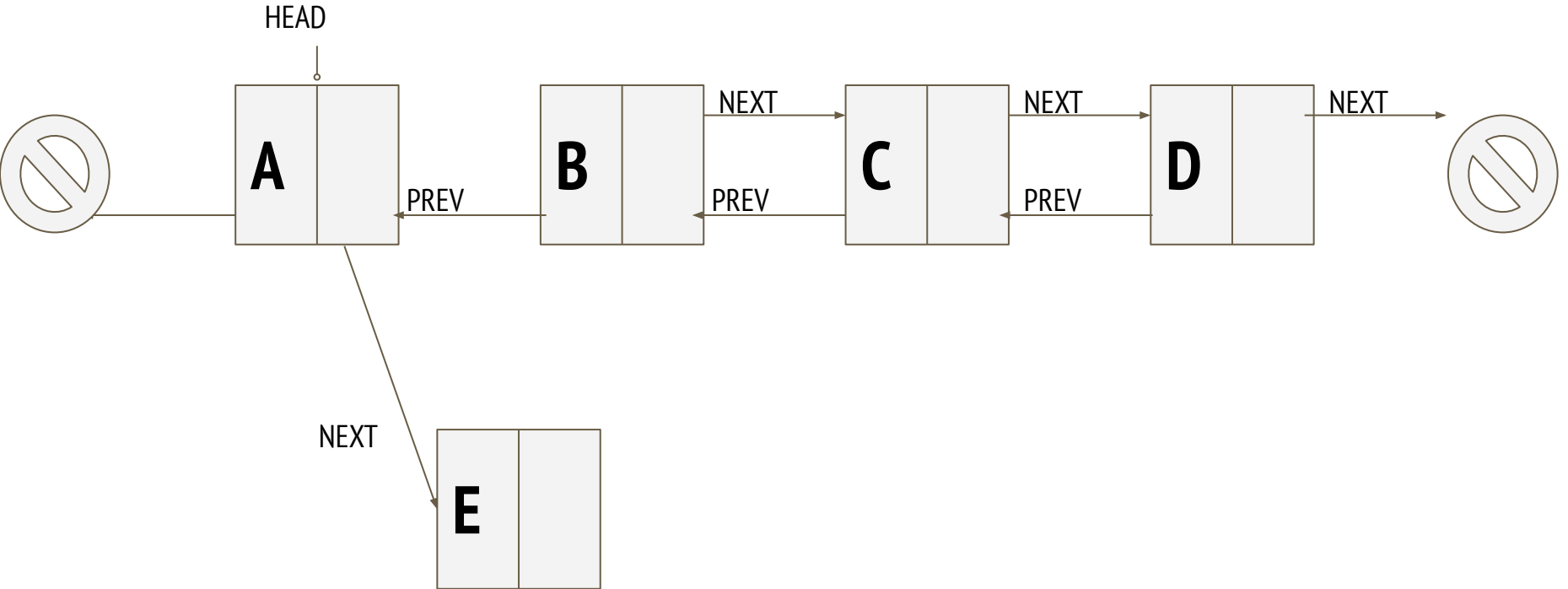
Doubly Linked List

Add before node with data "B"



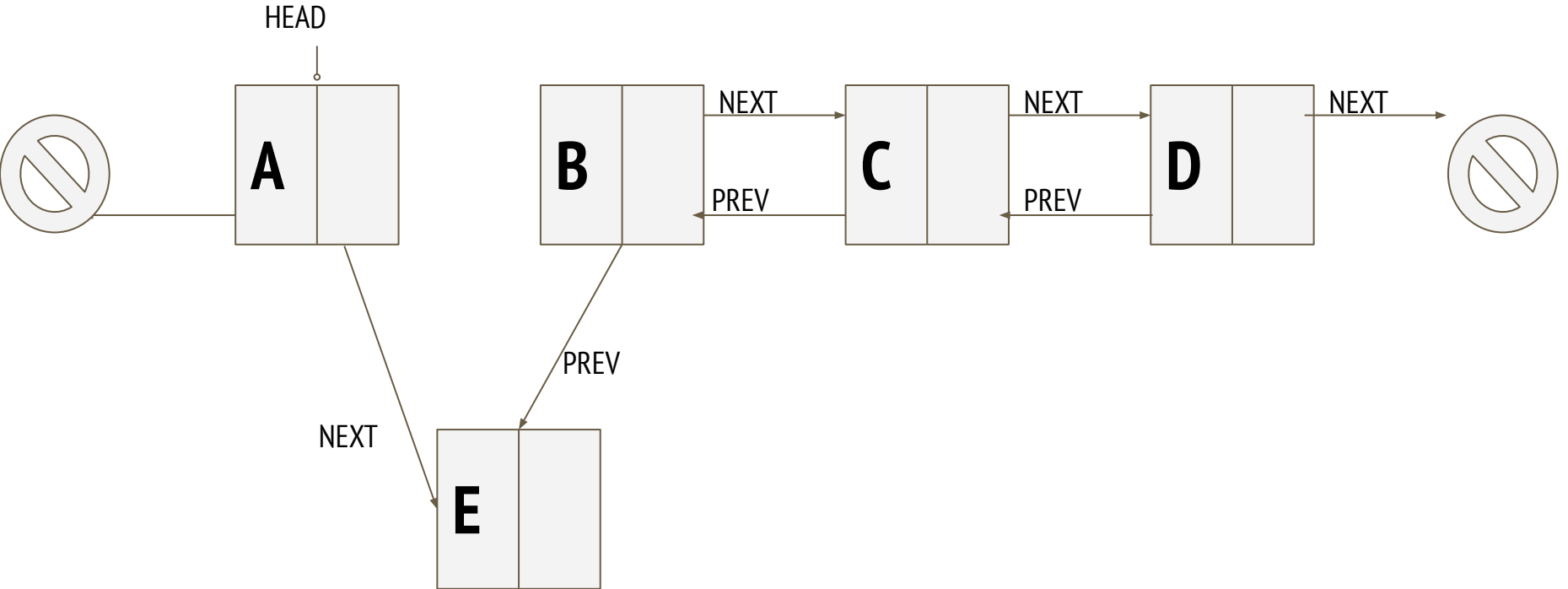
Doubly Linked List

Add before node with data "B"



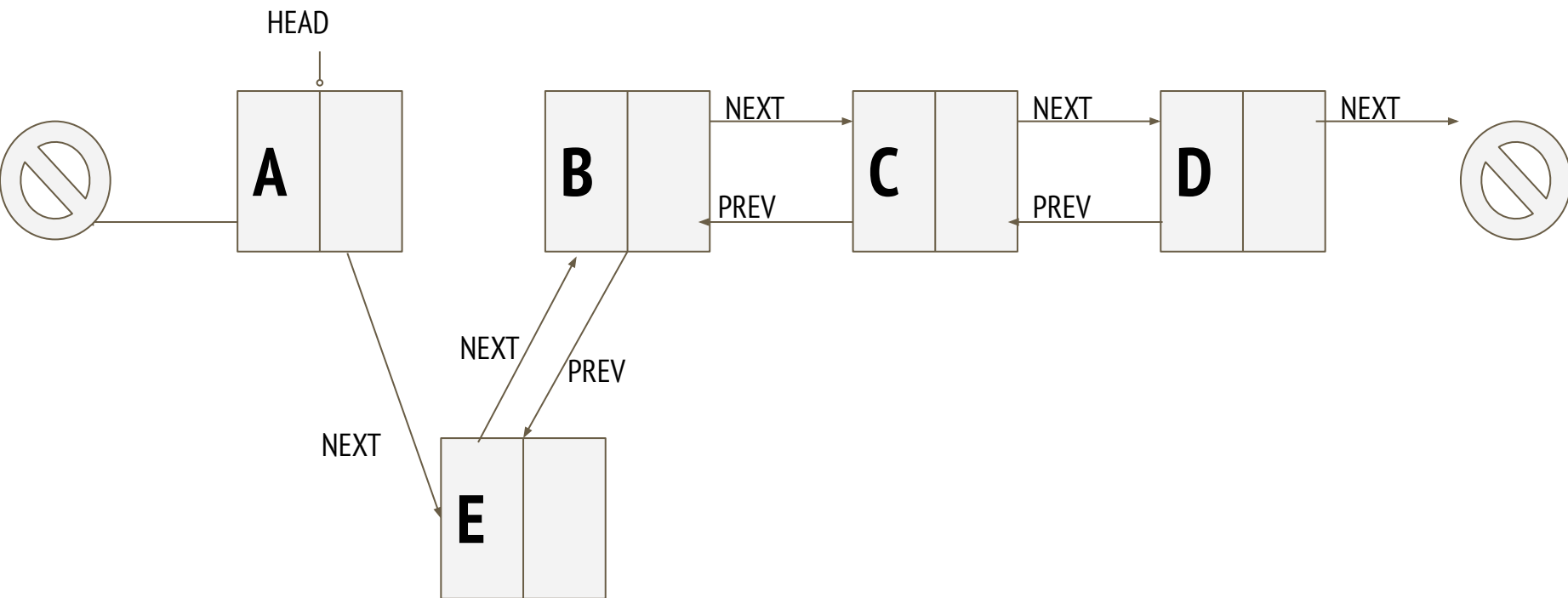
Doubly Linked List

Add before node with data "B"



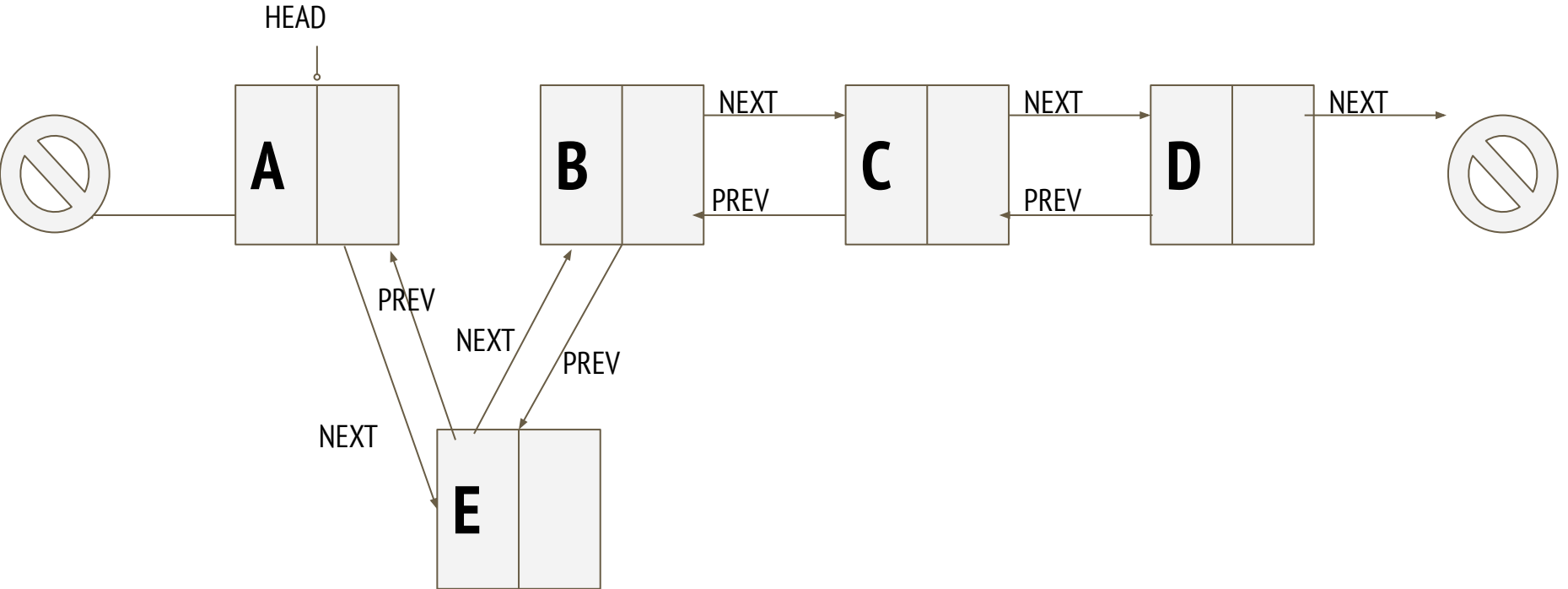
Doubly Linked List

Add before node with data "B"



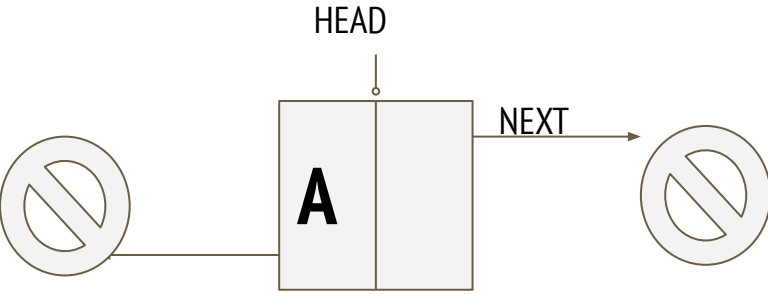
Doubly Linked List

Add before node with data "B"

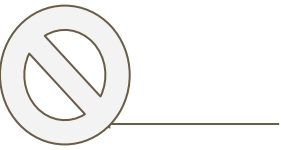


Doubly Linked Lists: Delete node

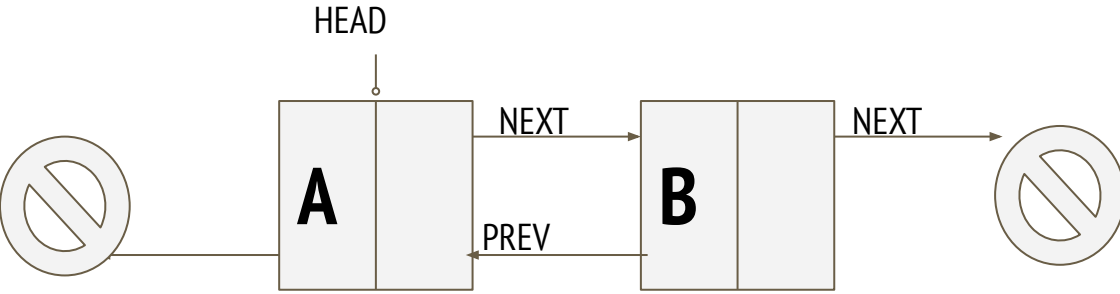
Doubly Linked List: Delete -- Case 1



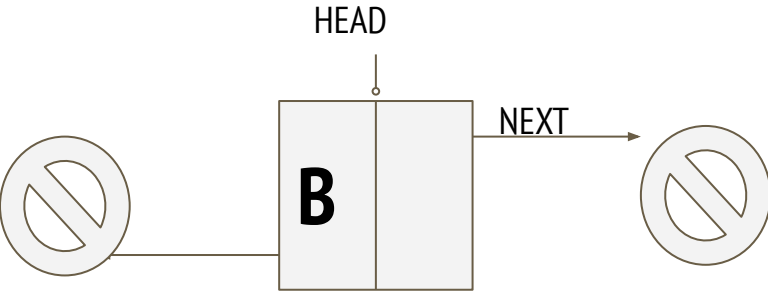
Doubly Linked List: Delete -- Case 1



Doubly Linked List: Delete -- Case 2

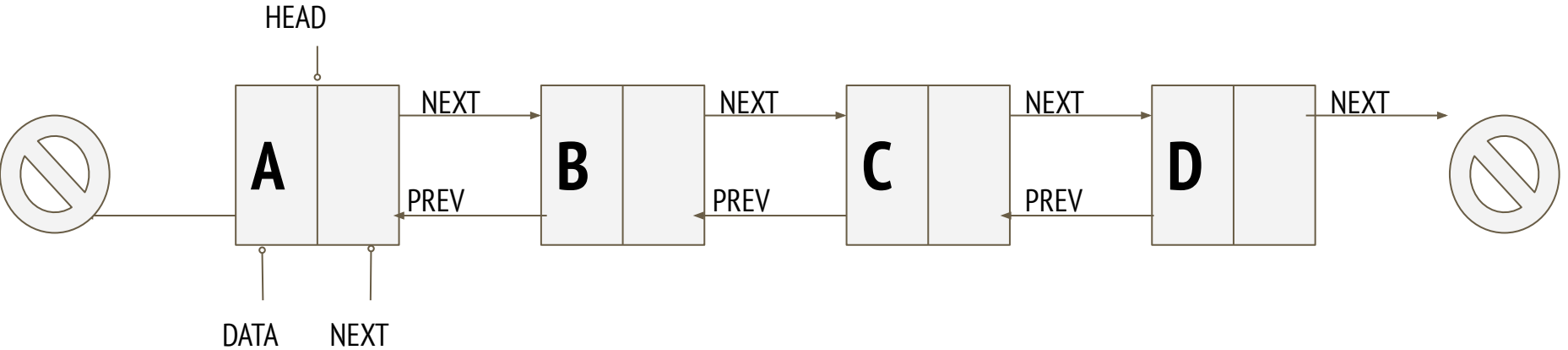


Doubly Linked List: Delete -- Case 2



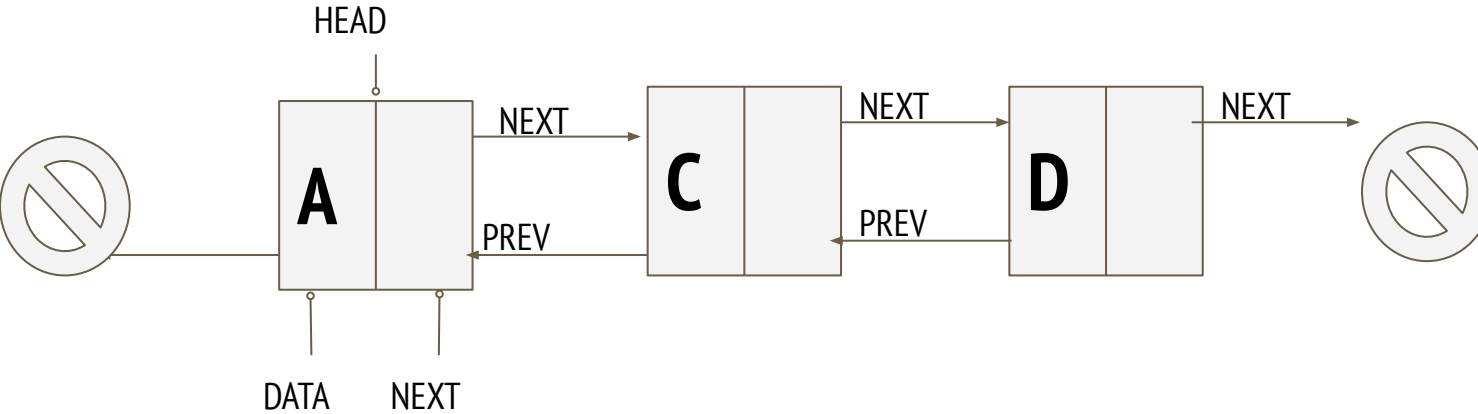
Doubly Linked List: Delete -- Case 3

Cur.next is not None

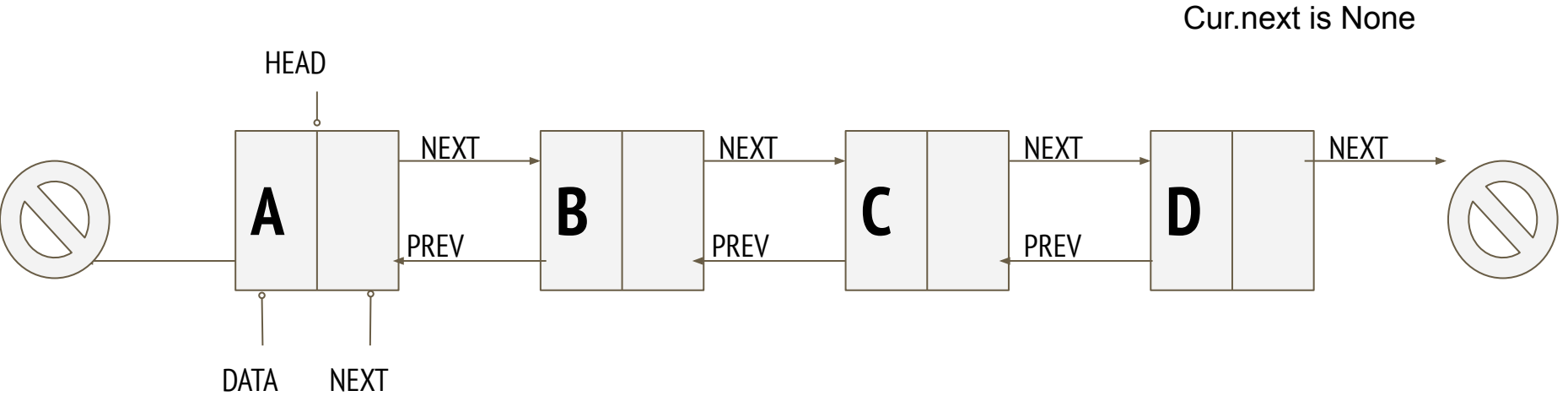


Doubly Linked List: Delete -- Case 3

Cur.next is not None

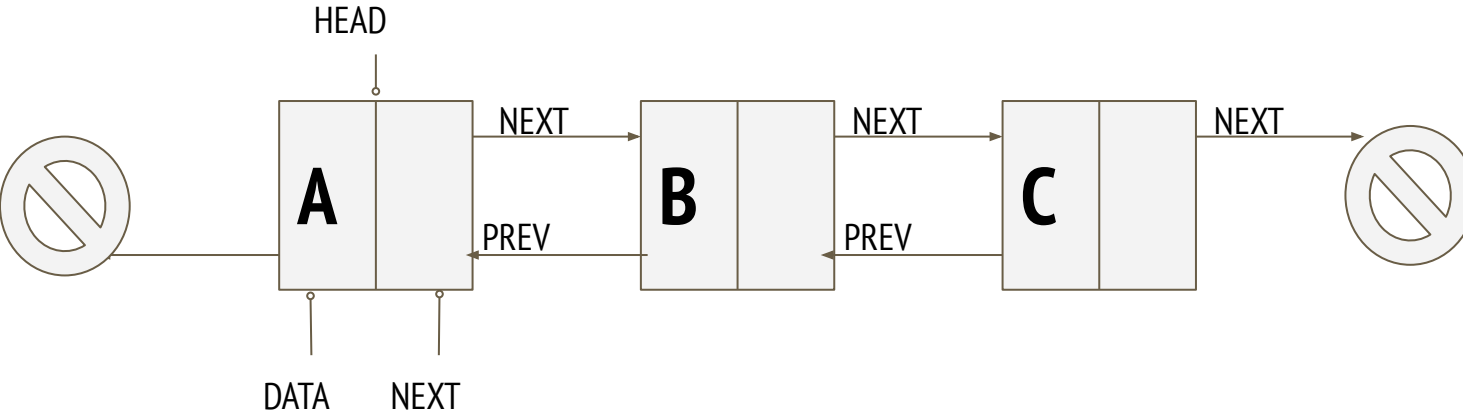


Doubly Linked List: Delete -- Case 4



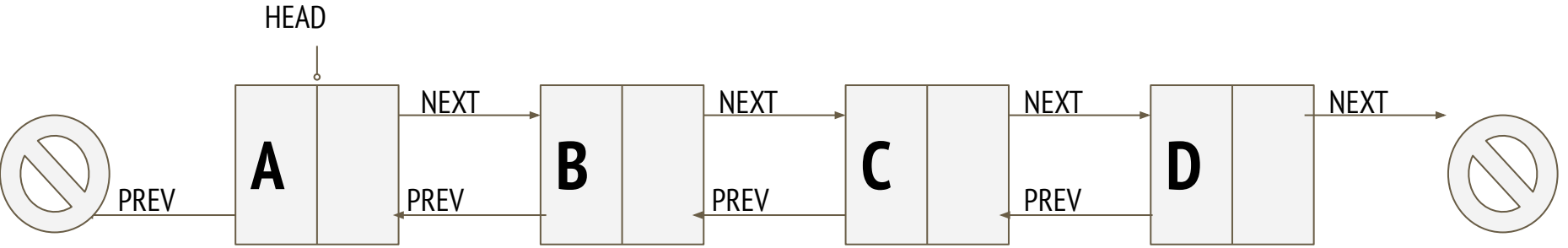
Doubly Linked List: Delete -- Case 4

Cur.next is None

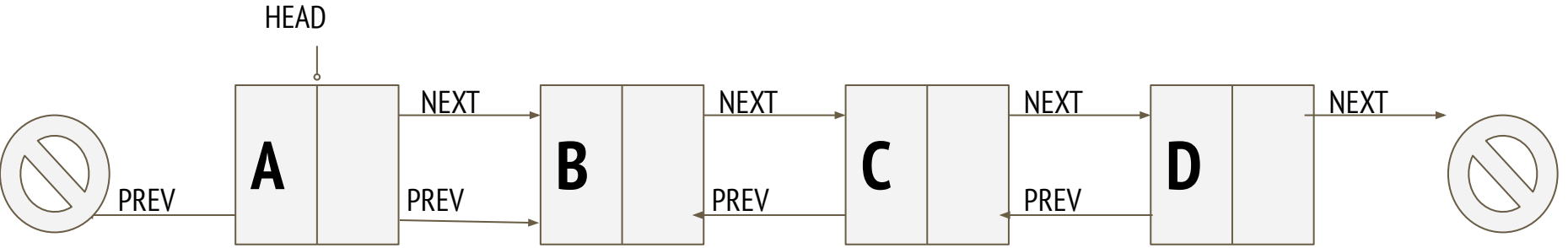


Doubly Linked Lists: Reverse

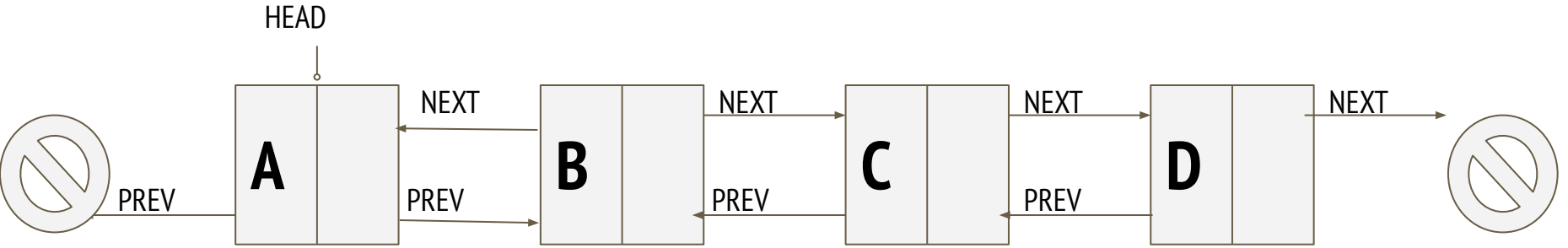
Doubly Linked List: Reverse



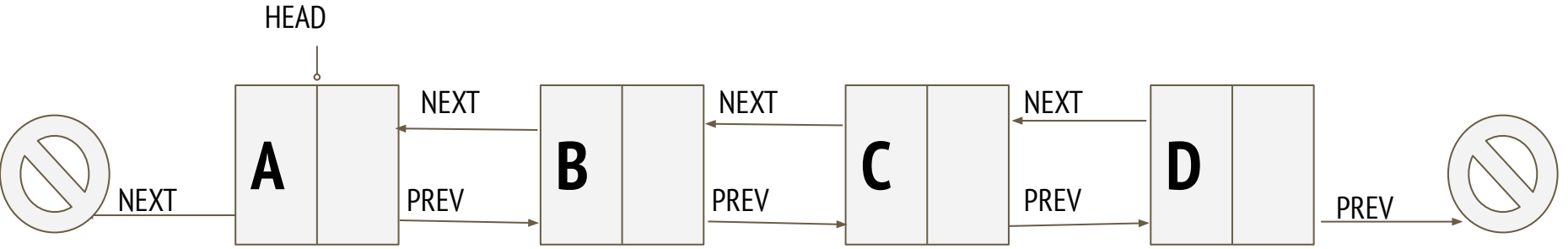
Doubly Linked List: Reverse



Doubly Linked List: Reverse



Doubly Linked List: Reverse



Doubly Linked List: Reverse

