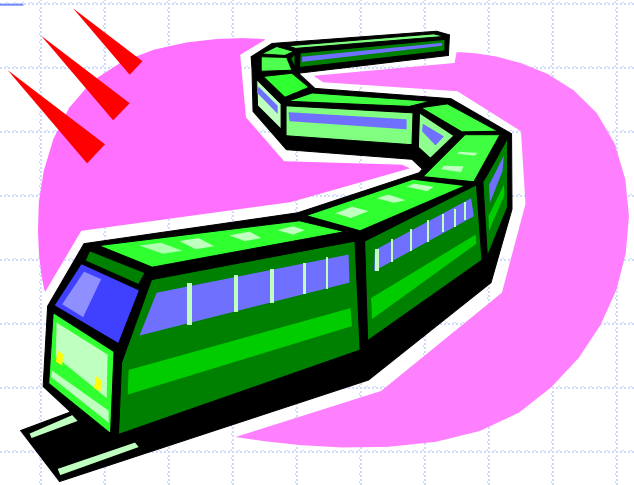
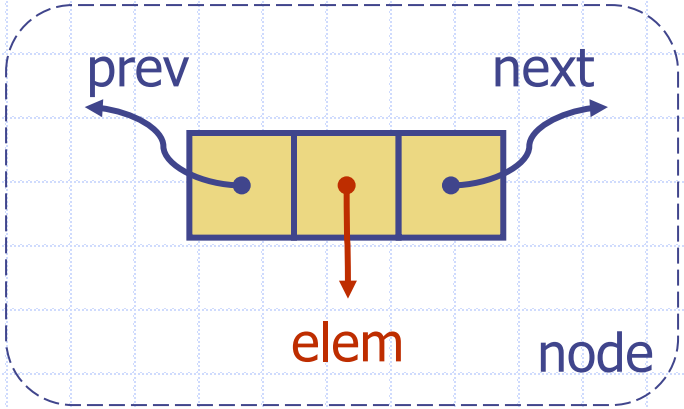
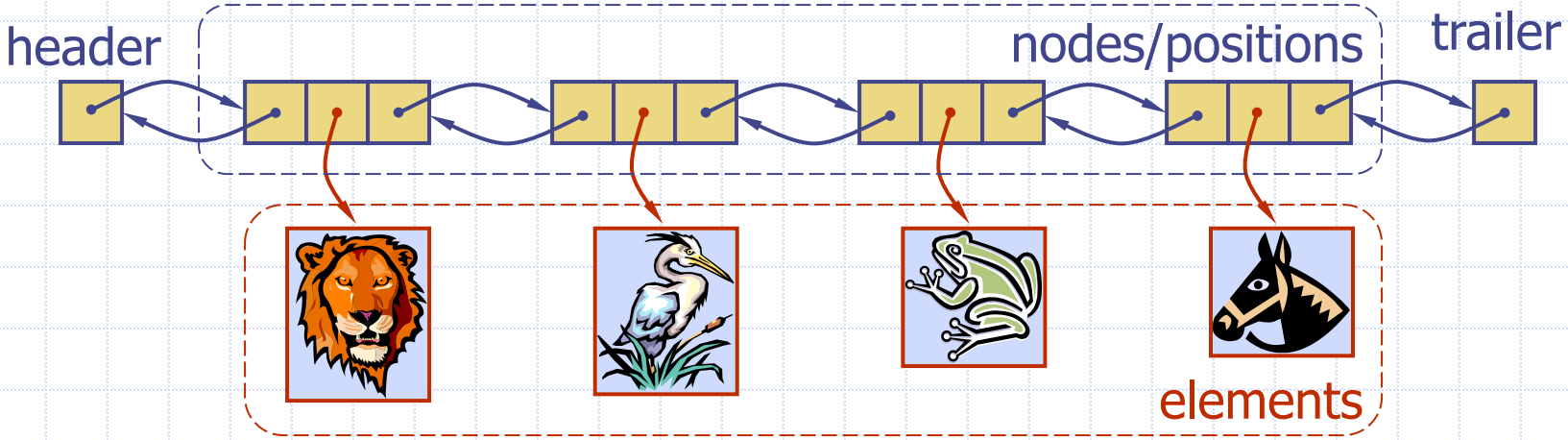
# Doubly-Linked Lists



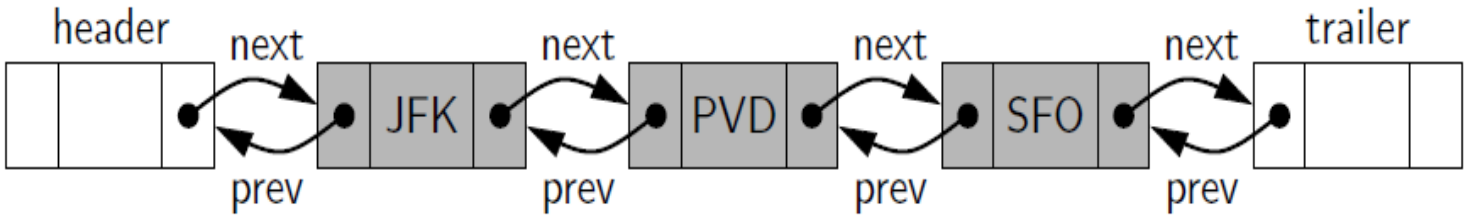
# Doubly Linked List

- A doubly linked list provides a natural implementation of the Node List ADT
- A Node stores:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes

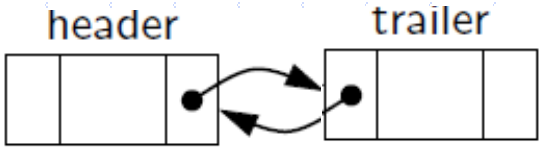


# Header and Trailer Sentinels




When Empty:



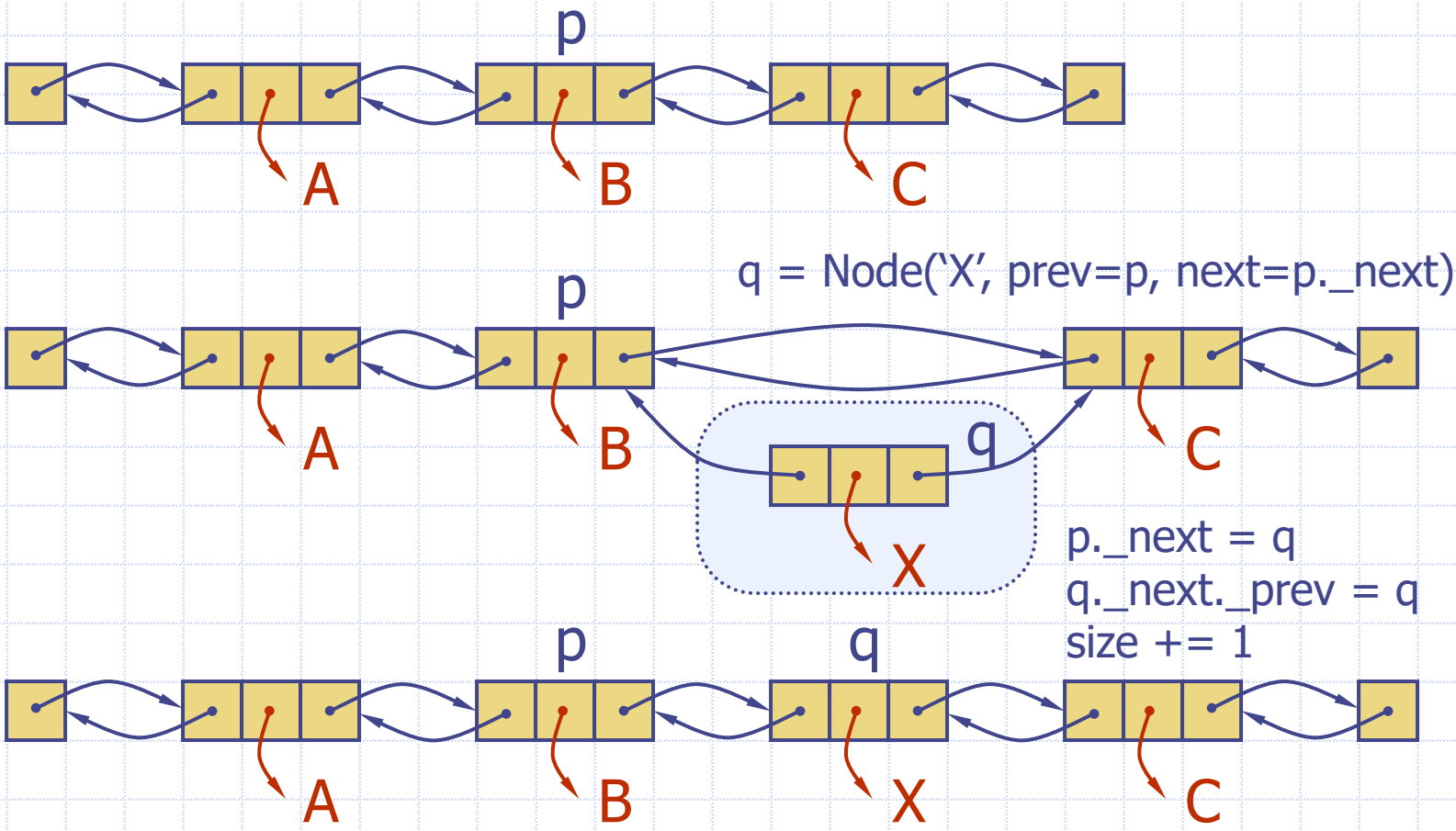
# Node Class for Double Link List

```
class _Node:
"""Lightweight, nonpublic class for storing a doubly linked node."""
    __slots__= _element , _prev , _next # streamline memory

  def __init__(self, element, prev, next): # initialize node's fields
      self._element = element # user's element
      self._prev = prev # previous node reference
      self._next = next # next node reference
```

# Insertion

- Insert a new node, q, between p and its successor.



# More Insertions Examples

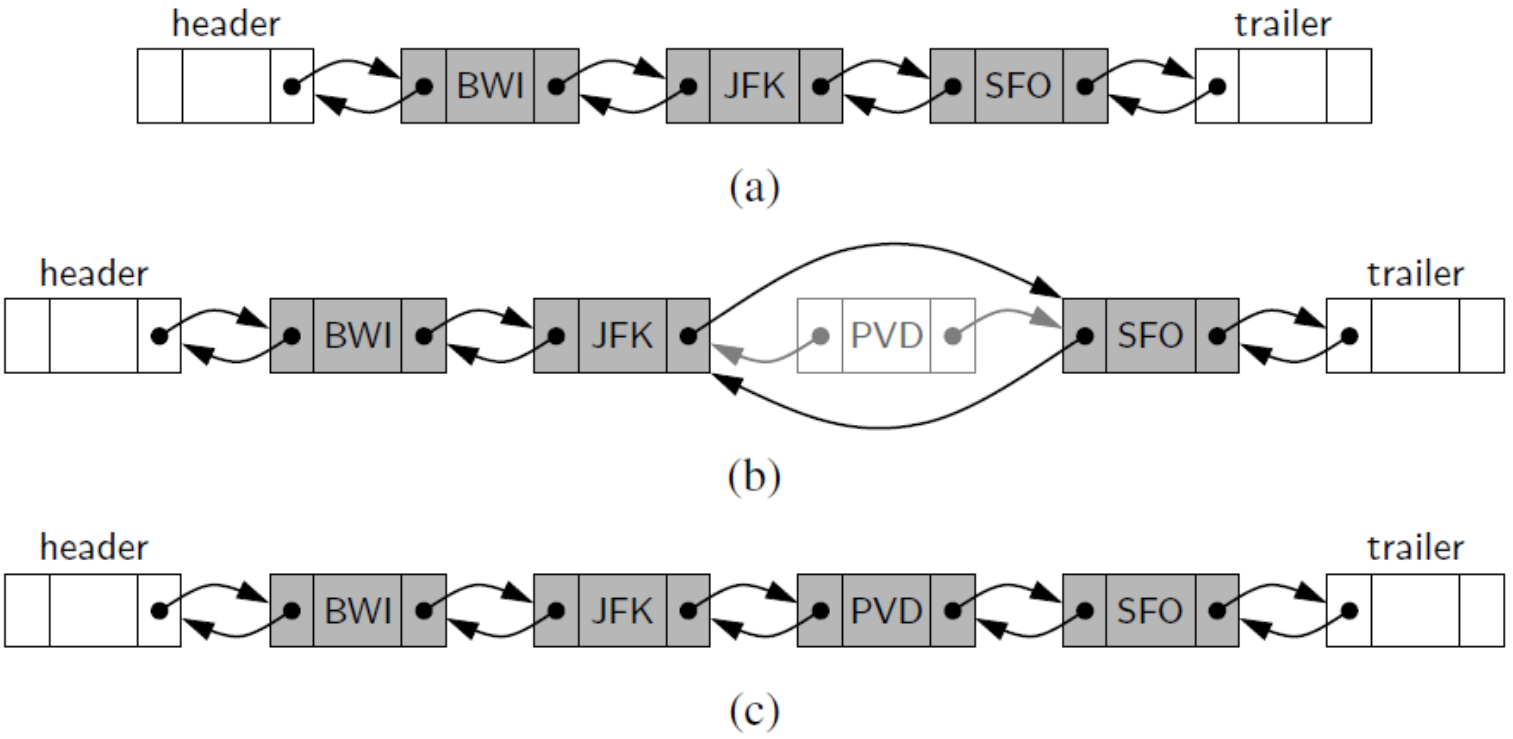



**Figure 7.11:** Adding an element to a doubly linked list with header and trailer sentinels: (a) before the operation; (b) after creating the new node; (c) after linking the neighbors to the new node.

# Insert at front

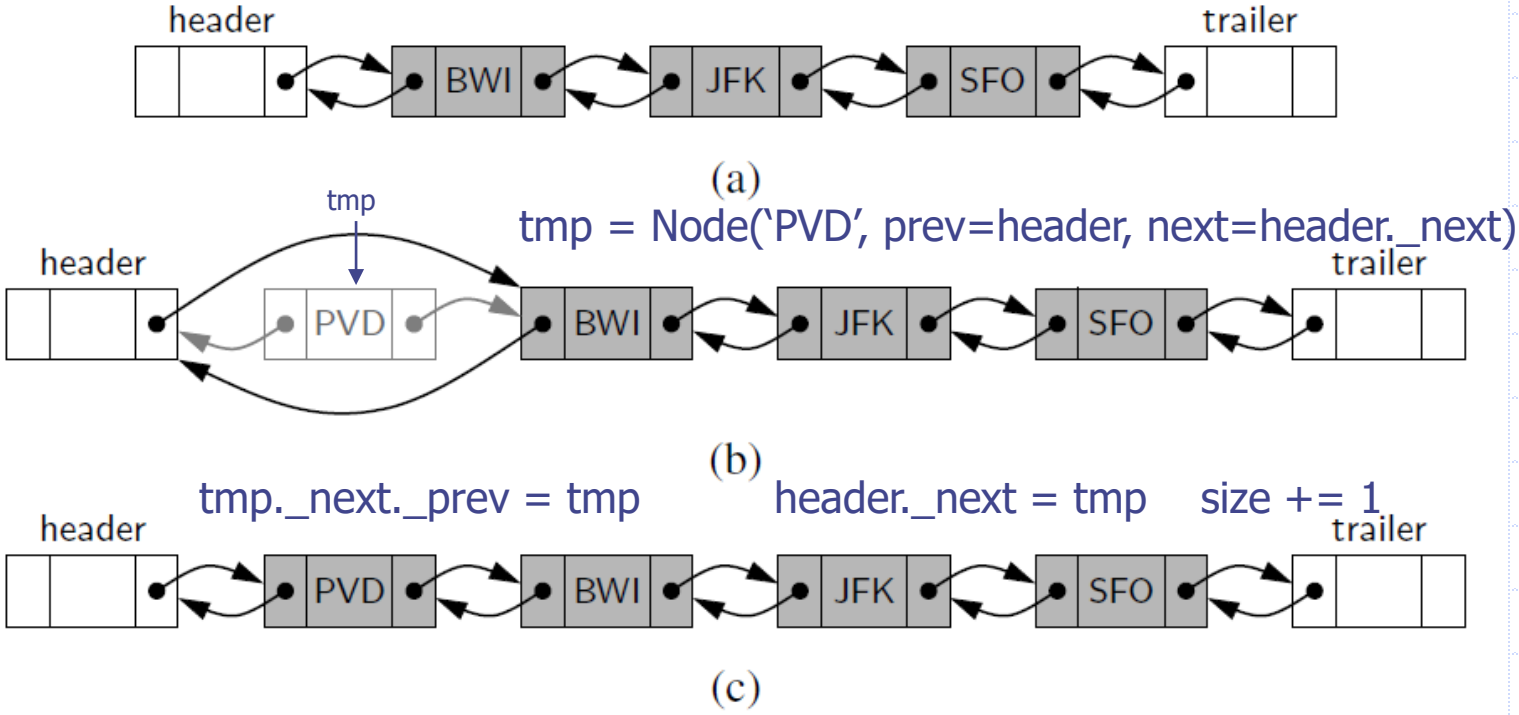
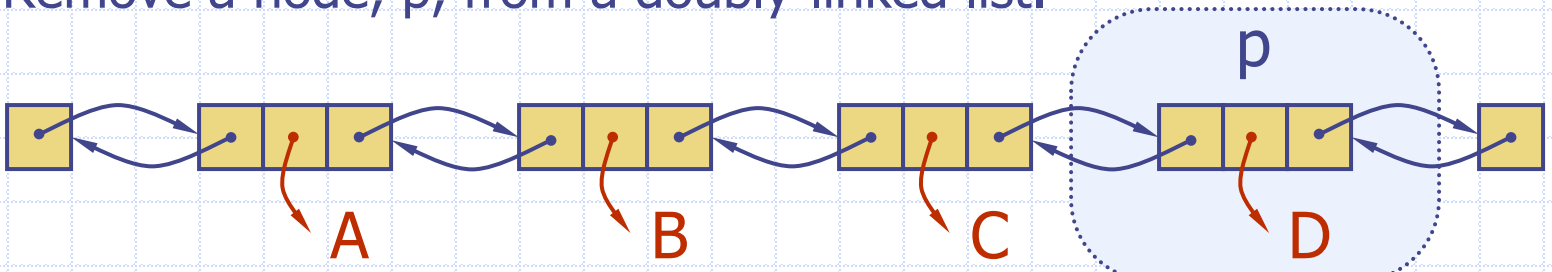
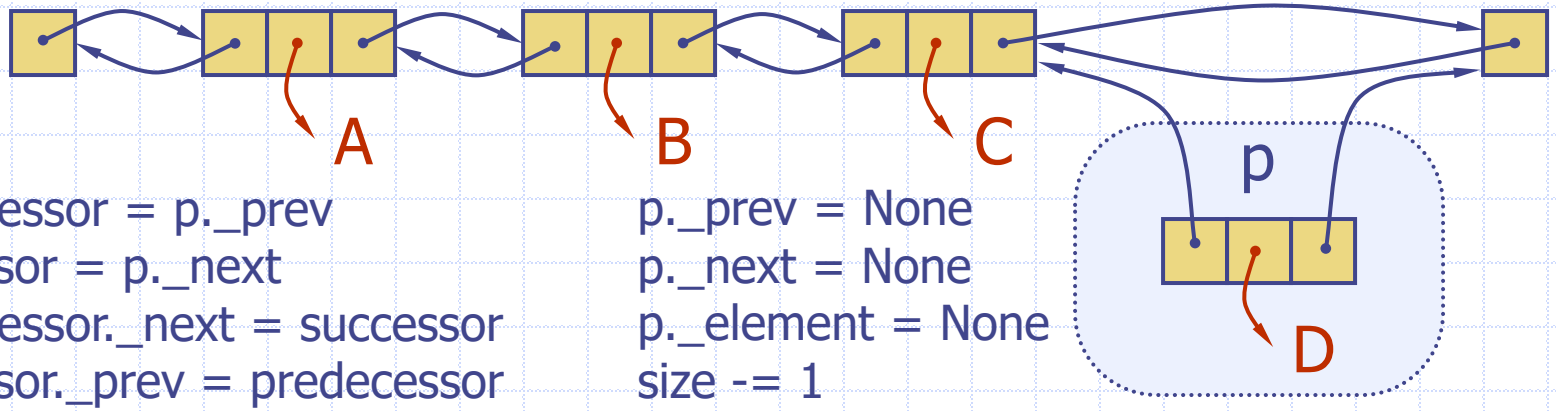



**Figure 7.12:** Adding an element to the front of a sequence represented by a doubly linked list with header and trailer sentinels: (a) before the operation; (b) after creating the new node; (c) after linking the neighbors to the new node.

# Deletion

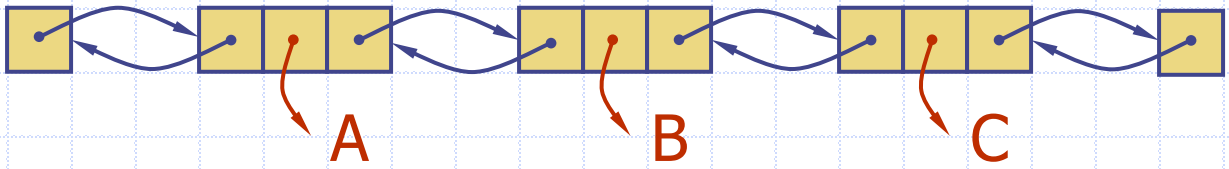
- Remove a node, p, from a doubly-linked list.




predecessor = p._prev
successor = p._next
predecessor._next = successor
successor._prev = predecessor

p._prev = None
p._next = None
p._element = None
size -= 1



# More Deletions

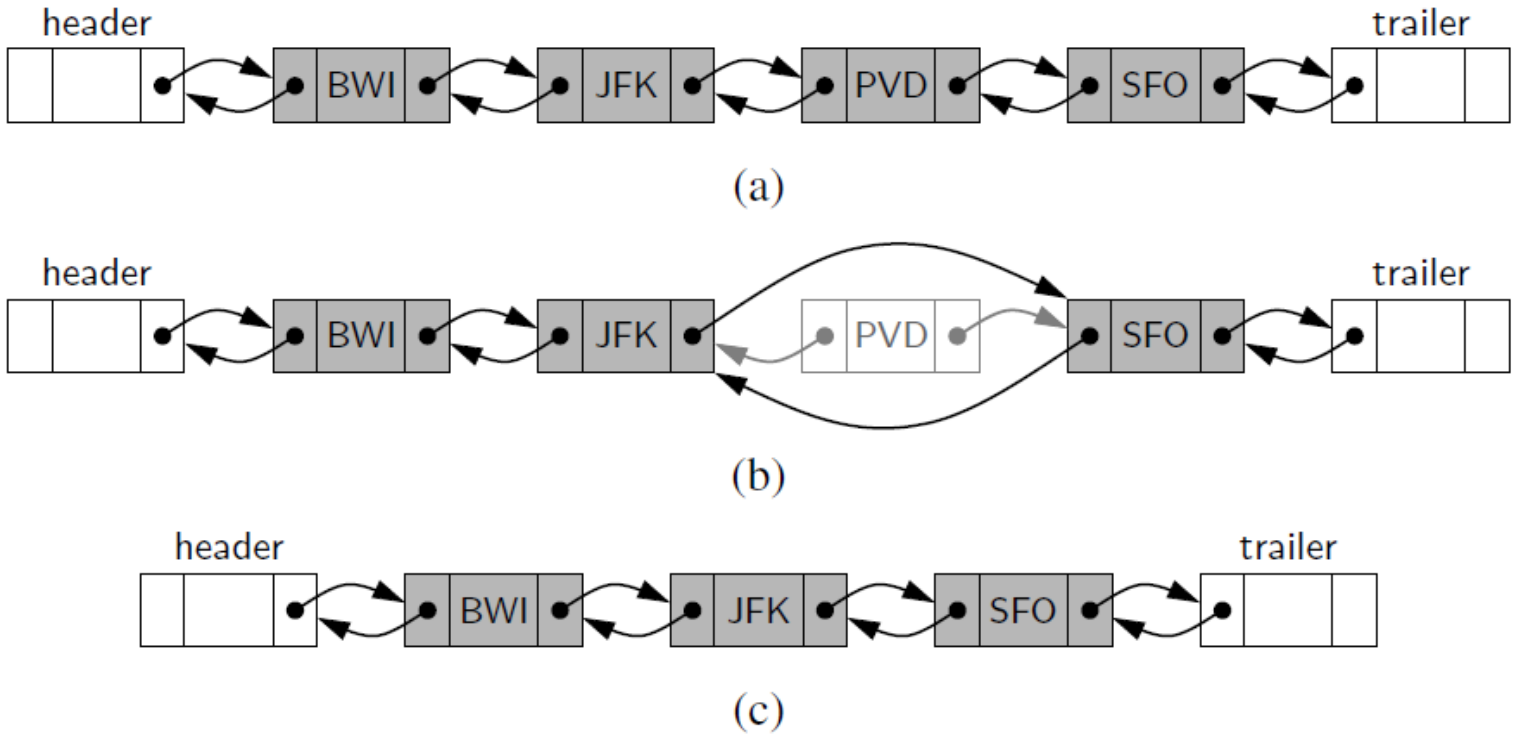



**Figure 7.13:** Removing the element PVD from a doubly linked list: (a) before the removal; (b) after linking out the old node; (c) after the removal (and garbage collection).

# Doubly-Linked List in Python

```
1  class _DoublyLinkedBase:
2    """A base class providing a doubly linked list representation."""
3
4    class _Node:
5      """Lightweight, nonpublic class for storing a doubly linked node."""
6      (omitted here; see previous code fragment)
7
8    def __init__(self):
9      """Create an empty list."""
10     self._header = self._Node(None, None, None)
11     self._trailer = self._Node(None, None, None)
12     self._header._next = self._trailer                # trailer is after header
13     self._trailer._prev = self._header                # header is before trailer
14     self._size = 0                                    # number of elements
15
16   def __len__(self):
17     """Return the number of elements in the list."""
18     return self._size
19
20   def is_empty(self):
21     """Return True if list is empty."""
22     return self._size == 0
23
```

```
24   def _insert_between(self, e, predecessor, successor):
25     """Add element e between two existing nodes and return new node."""
26     newest = self._Node(e, predecessor, successor)   # linked to neighbors
27     predecessor._next = newest
28     successor._prev = newest
29     self._size += 1
30     return newest
31
32   def _delete_node(self, node):
33     """Delete nonsentinel node from the list and return its element."""
34     predecessor = node._prev
35     successor = node._next
36     predecessor._next = successor
37     successor._prev = predecessor
38     self._size -= 1
39     element = node._element                          # record deleted element
40     node._prev = node._next = node._element = None   # deprecate node
41     return element                                   # return deleted element
```

# Performance

- In a doubly linked list
  - The space used by a list with $n$ elements is $O(n)$
  - The space used by each position of the list is $O(1)$
  - All the standard operations of a list run in $O(1)$ time