CS 242 - CS252

Doubly Linked Lists

Singly linked list

- We can efficiently (O(1)) do the following
  - insert a node at either end of a singly linked list
  - delete a node at the head of a list
- We are unable to efficiently delete a node at the tail of the list. *Why??*
- Given a reference to a node at any position, we can not delete it in efficient time.

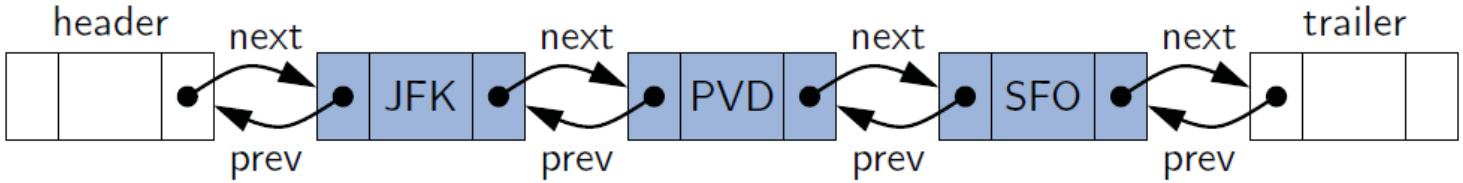
Doubly Linked Lists

- A linked list in which each node keeps an explicit reference to the node before it and a reference to the node after it
- *O*(1)-time update operations

- In defining the Node class
  - "next" for the reference to the node that follows it
  - "prev" for the reference to the node that precedes it

Header and trailer

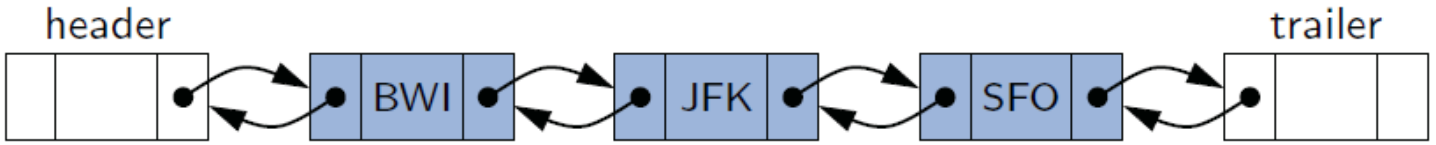
- Special nodes at both ends of the list.



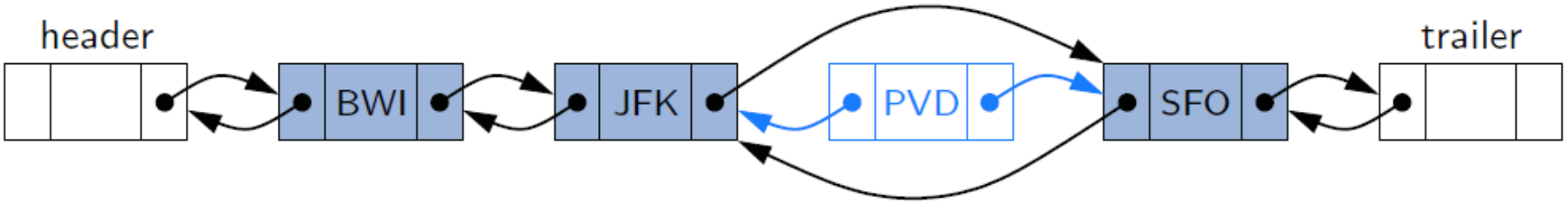
- Make implementation easier
  - Every insertion/deletion is between a pair of existing nodes.

- Can be implemented without them
  - Save space
  - Need to consider special cases at the beginning and at the end

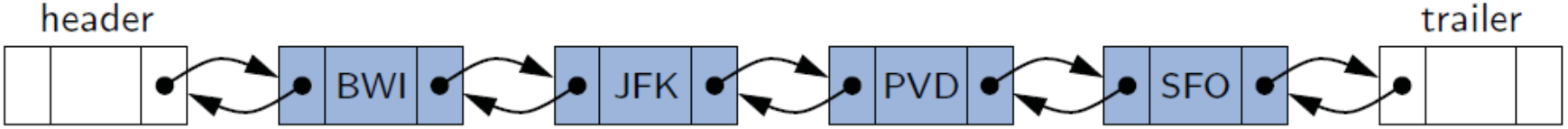
Inserting with a Doubly Linked List



(a)

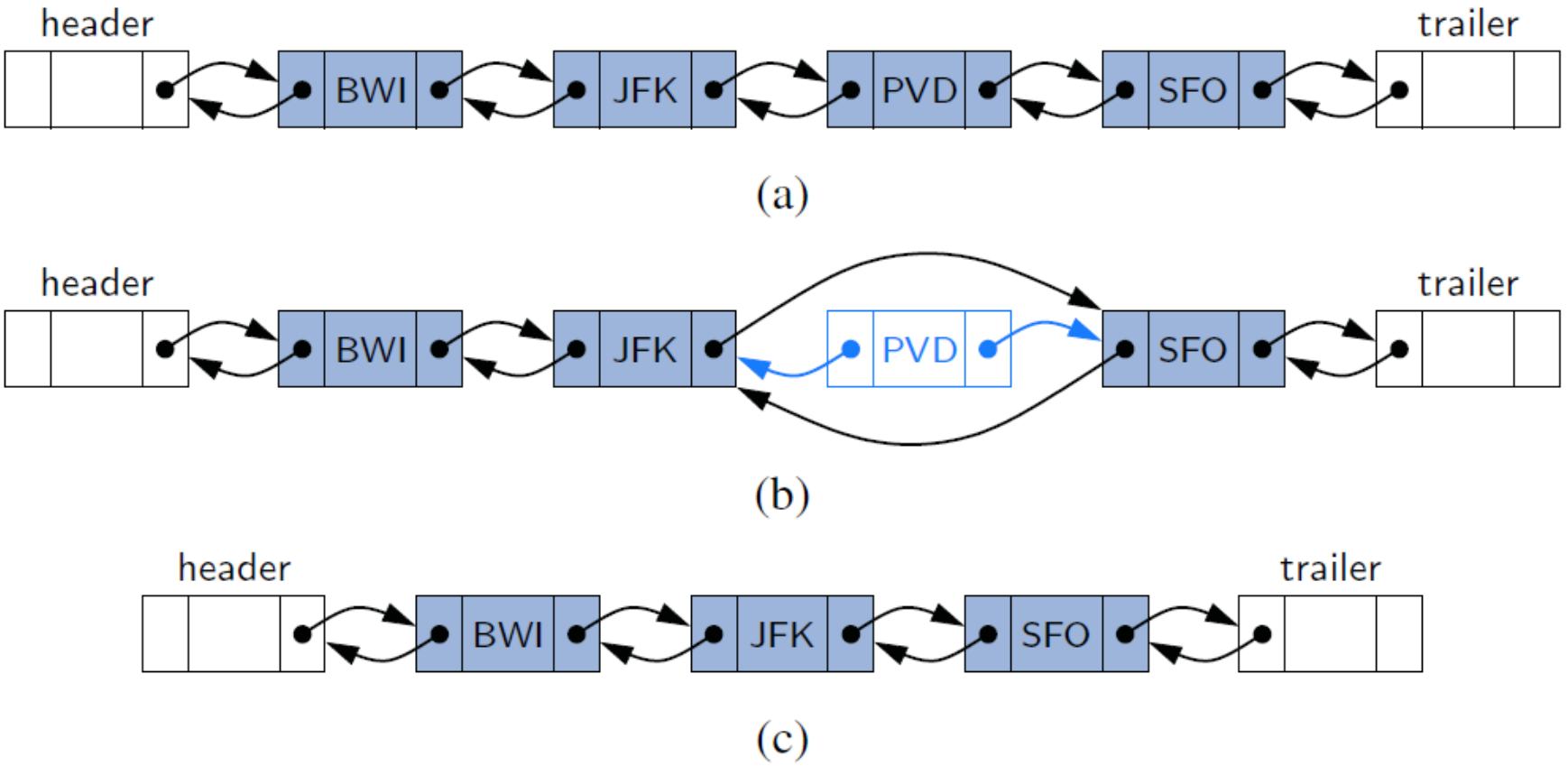


(b)



(c)

Deleting with a Doubly Linked List

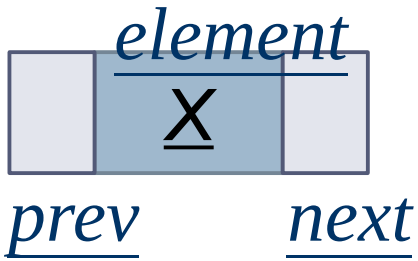


Implementing a Doubly Linked List Class

size( ): Returns the number of elements in the list.
isEmpty( ): Returns true if the list is empty, and false otherwise.
first( ): Returns (but does not remove) the first element in the list.
last( ): Returns (but does not remove) the last element in the list.
addFirst(e): Adds a new element to the front of the list.
addLast(e): Adds a new element to the end of the list.
removeFirst( ): Removes and returns the first element of the list.
removeLast( ): Removes and returns the last element of the list.

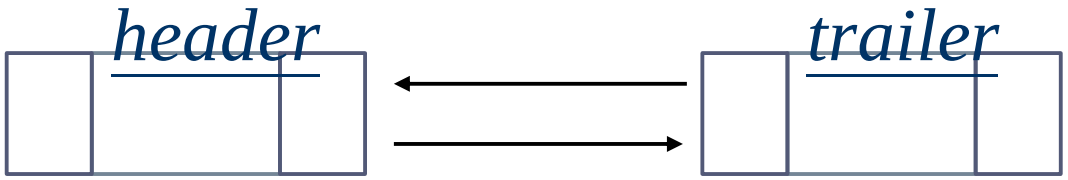
Private Class Node

```
1  /** A basic doubly linked list implementation. */
2  public class DoublyLinkedList<E> {
3    //---------------- nested Node class ----------------
4    private static class Node<E> {
5      private E element;                  // reference to the element stored at this node
6      private Node<E> prev;               // reference to the previous node in the list
7      private Node<E> next;               // reference to the subsequent node in the list
8      public Node(E e, Node<E> p, Node<E> n) {
9        element = e;
10       prev = p;
11       next = n;
12     }
13     public E getElement( ) { return element; }
14     public Node<E> getPrev( ) { return prev; }
15     public Node<E> getNext( ) { return next; }
16     public void setPrev(Node<E> p) { prev = p; }
17     public void setNext(Node<E> n) { next = n; }
18   } //---------- end of nested Node class ----------
```



The Attributes and the Constructor

```
20    // instance variables of the DoublyLinkedList
21    private Node<E> header;                    // header sentinel
22    private Node<E> trailer;                   // trailer sentinel
23    private int size = 0;                      // number of elements in the list
24    /** Constructs a new empty list. */
25    public DoublyLinkedList( ) {
26      header = new Node<>(null, null, null);   // create header
27      trailer = new Node<>(null, header, null); // trailer is preceded by header
28      header.setNext(trailer);                  // header is followed by trailer
29    }
```



Access Methods

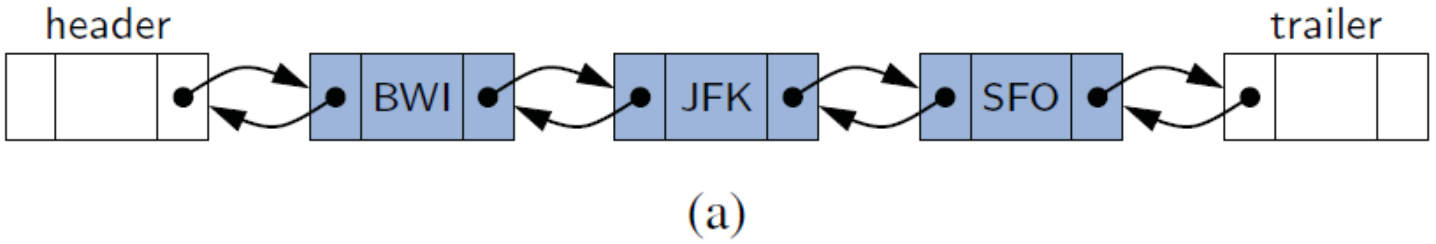
```
30    /** Returns the number of elements in the linked list. */
31    public int size( ) { return size; }
32    /** Tests whether the linked list is empty. */
33    public boolean isEmpty( ) { return size == 0; }
34    /** Returns (but does not remove) the first element of the list. */
35    public E first( ) {
36      if (isEmpty( )) return null;
37      return header.getNext( ).getElement( );          // first element is beyond header
38    }
39    /** Returns (but does not remove) the last element of the list. */
40    public E last( ) {
41      if (isEmpty( )) return null;
42      return trailer.getPrev( ).getElement( );         // last element is before trailer
43    }
```

Public update methods

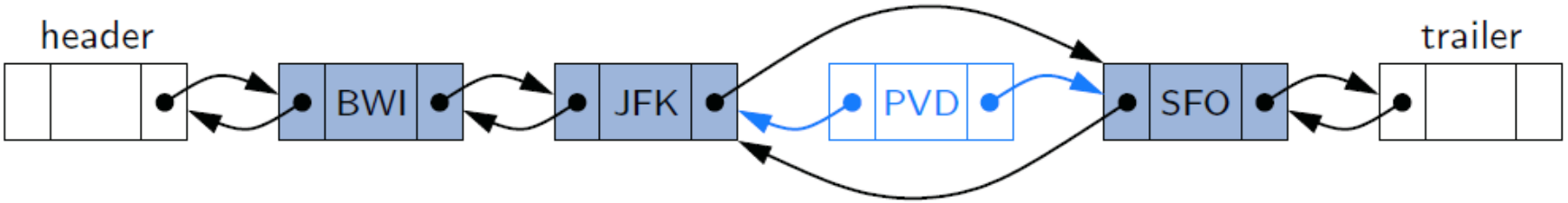
```
44    // public update methods
45    /** Adds element e to the front of the list. */
46    public void addFirst(E e) {
47      addBetween(e, header, header.getNext( ));          // place just after the header
48    }
49    /** Adds element e to the end of the list. */
50    public void addLast(E e) {
51      addBetween(e, trailer.getPrev( ), trailer);        // place just before the trailer
52    }

64    // private update methods
65    /** Adds element e to the linked list in between the given nodes. */
66    private void addBetween(E e, Node<E> predecessor, Node<E> successor) {
67      // create and link a new node
68      Node<E> newest = new Node<>(e, predecessor, successor);
69      predecessor.setNext(newest);
70      successor.setPrev(newest);
71      size++;
72    }
```

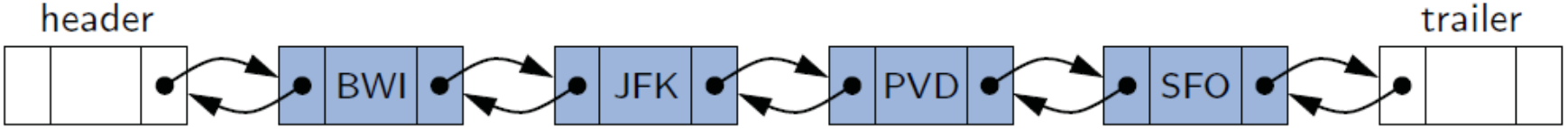
Inserting with a Doubly Linked List



(a)



(b)



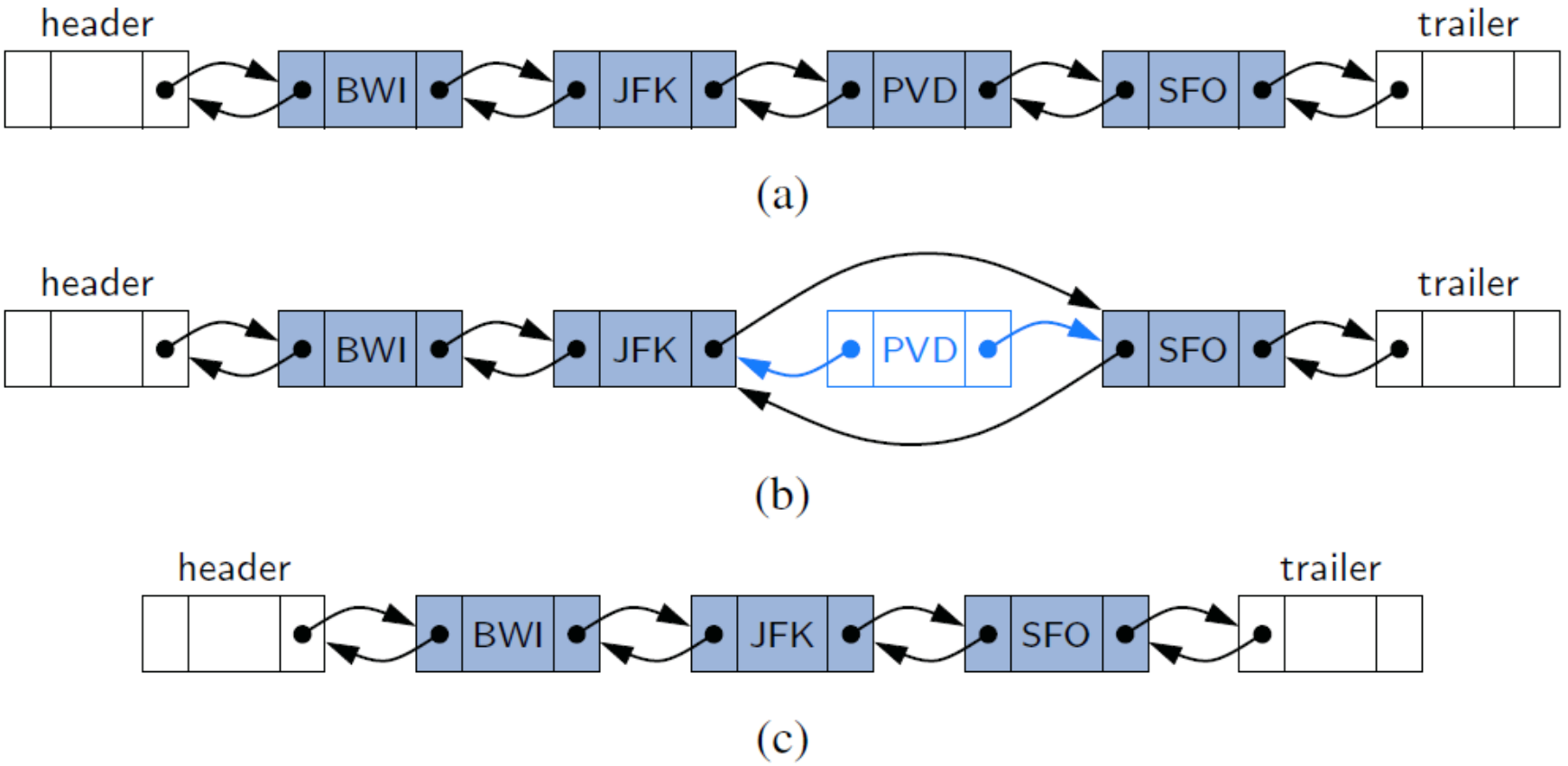
(c)

Public update methods

```
53    /** Removes and returns the first element of the list. */
54    public E removeFirst( ) {
55      if (isEmpty( )) return null;                    // nothing to remove
56      return remove(header.getNext( ));               // first element is beyond header
57    }
58    /** Removes and returns the last element of the list. */
59    public E removeLast( ) {
60      if (isEmpty( )) return null;                    // nothing to remove
61      return remove(trailer.getPrev( ));              // last element is before trailer
62    }

73    /** Removes the given node from the list and returns its element. */
74    private E remove(Node<E> node) {
75      Node<E> predecessor = node.getPrev( );
76      Node<E> successor = node.getNext( );
77      predecessor.setNext(successor);
78      successor.setPrev(predecessor);
79      size--;
80      return node.getElement( );
81    }
```

Deleting with a Doubly Linked List



Exercise

- Describe in pseudo-code how to swap two nodes *x* and *y* in a singly linked list L given references only to *x* and *y*. Repeat this exercise for the case when L is a doubly linked list. What are the running times of each of these methods in terms of *n*, the number of nodes in L?