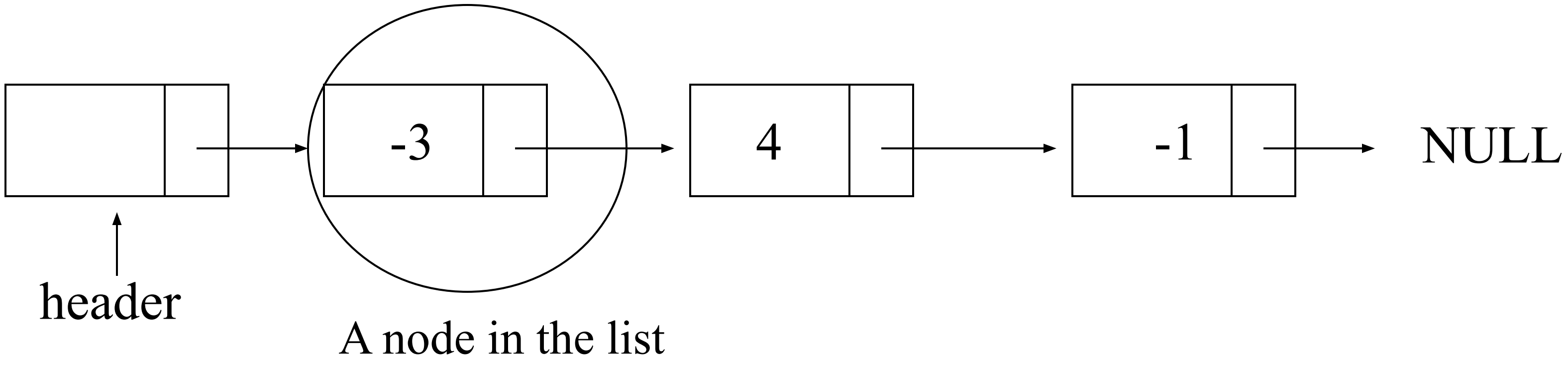
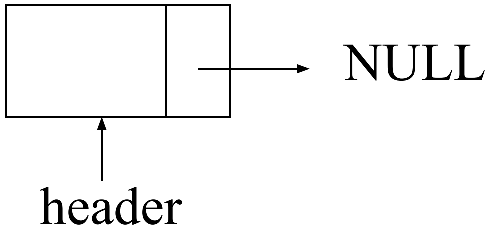
DATA STRUCTURES

Doubly and Circular Lists

Problems with Single Linked Lists





(An empty list)

Two problems – we can’t get back to the beginning of the list from the end, and we can’t go backwards through the list. So, circular linked lists and doubly linked lists were invented.

Other list flavors

- Doubly-linked list
  - Each node has a pointer to its successor and its predecessor.
    - Faster insert/delete, but more space.
- Circular list
  - The last node points back to the head.
- Sorted list
  - Items stored in sorted order.

Doubly-Linked Lists

- A common variation on linked lists is to have two references to other nodes within each node: one to the *next* node on the list, and one to the *previous* node

- Doubly-linked lists make some operations, such as deleting a tail node, more efficient

Doubly Linked List

- A doubly linked list is a linked list in which every node has a next pointer and a back pointer

- Every node (except the last node) contains the address of the next node, and every node (except the first node) contains the address of the previous node.

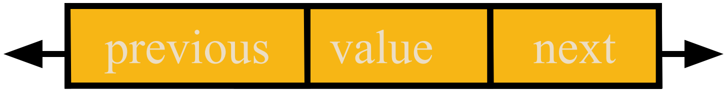
- A doubly linked list can be traversed in either direction

Doubly-Linked Lists

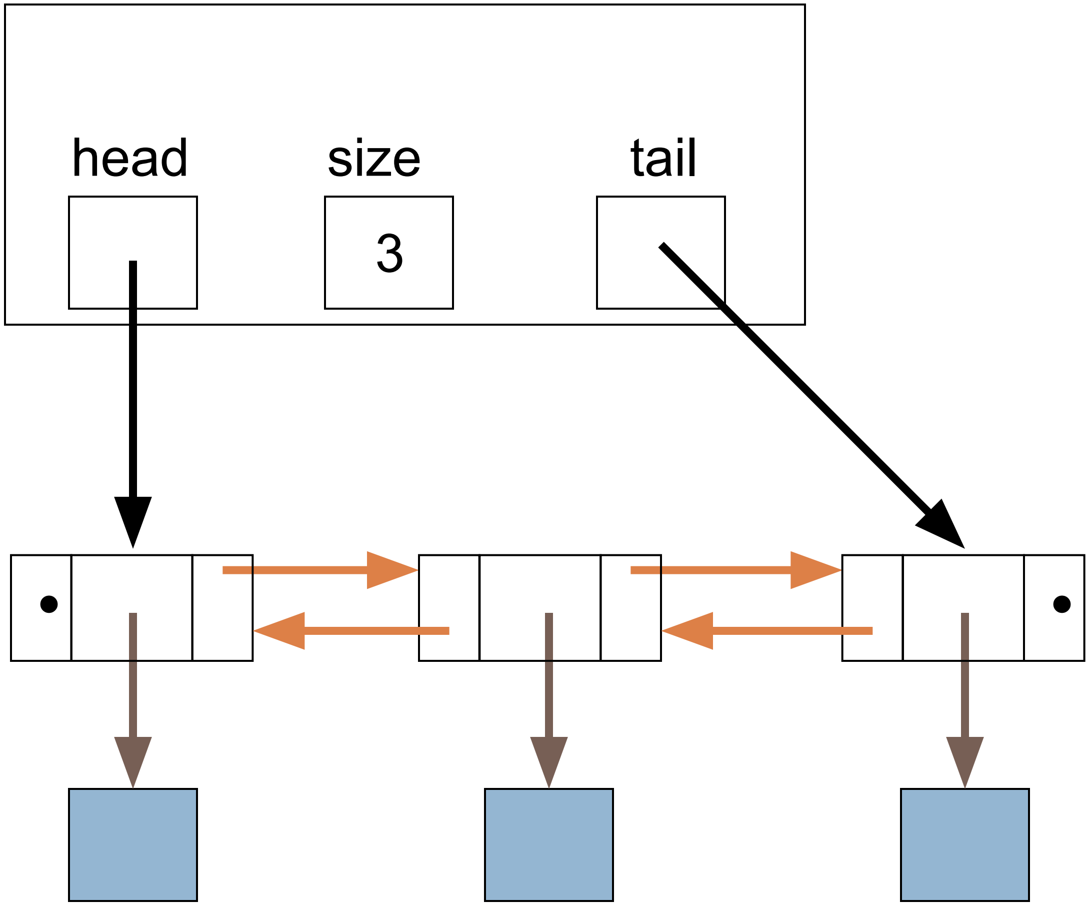
- Other operations, such as adding an item to an ordered linked list, are easier to program with doubly-linked lists
- *Tradeoffs:*
  - The *NodeType* class is more complex: extra instance variable, extra methods, revised constructors
  - Each node requires 4 additional bytes

Doubly-linked Lists - properties

- Allow sequential access to the list in **both** directions
- Each element points to
  - **Next** element
  - **Previous** element

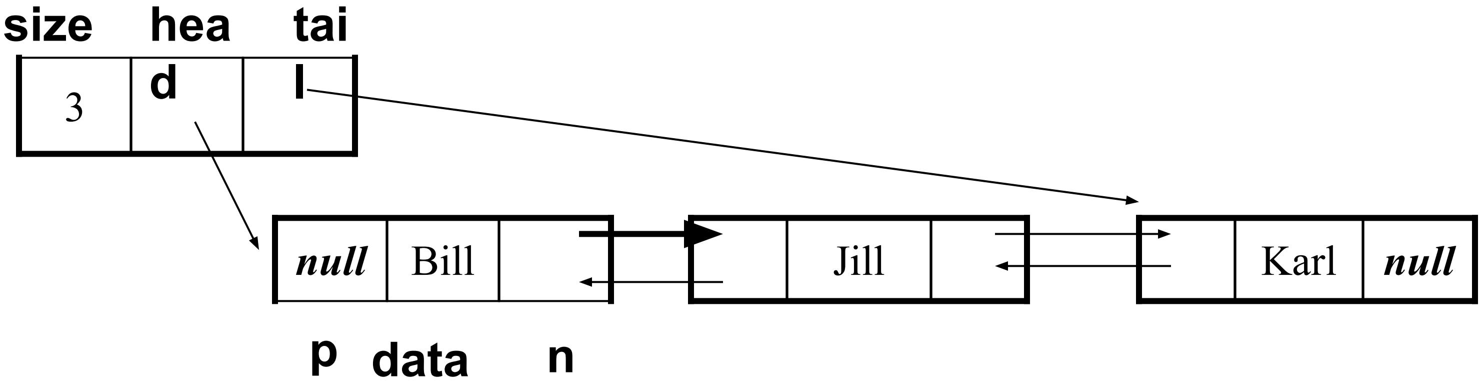


Doubly-Linked List

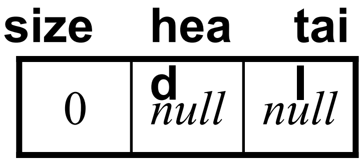


Example - Doubly-linked Lists

- Non-empty doubly linked list



- Empty doubly linked list



Advantages

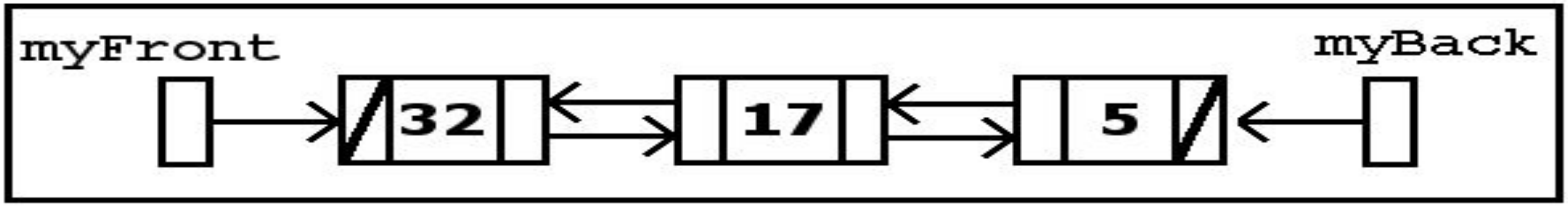
- The doubly linked list eliminates the need for the “pPrevious” pointer since each node has pointers to both the previous and next nodes in the list.

- You can go to the previous node easily
  - Traversal is in both directions.

- Implementations: ALT+TAB and ALT+SHIFT+TAB (Window Browsing)
  - Picture Viewers, Power Point Presentations

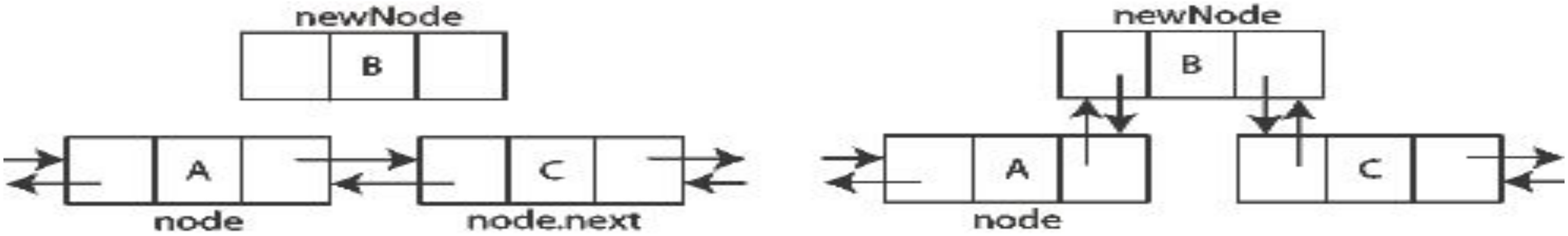
Doubly-linked lists

- add a `prev` pointer to our `Node` class
- allows backward iteration
- some methods need to be modified
  - when adding or removing a node, we must fix the `prev` and `next` pointers to have the correct value!
  - can make it easier to implement some methods such as `remove`



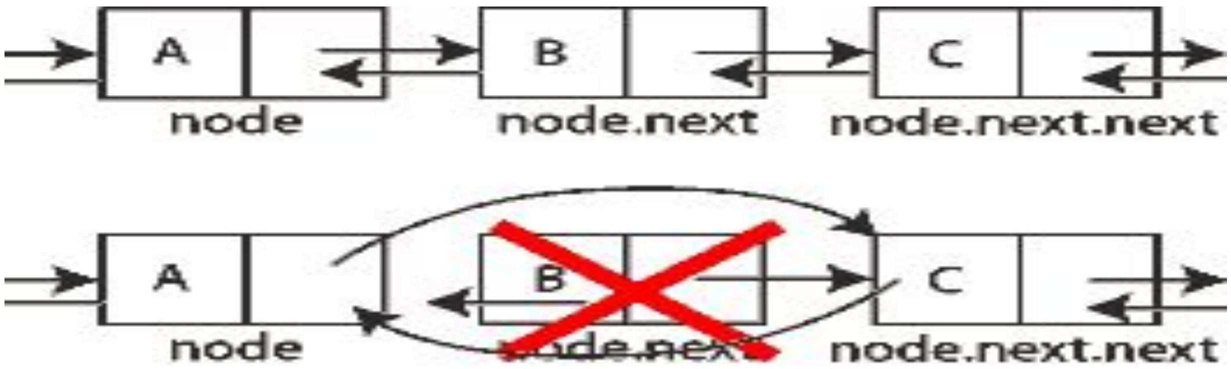
Linked list add operation

- When adding a node to the list at a given index, the following steps must be taken:
  - Advance through the list to the node just before the one with the proper index.
  - Create a new node, and attach it to the nodes that should precede and follow it.
    - How many 'arrows' (references) will need to be changed?



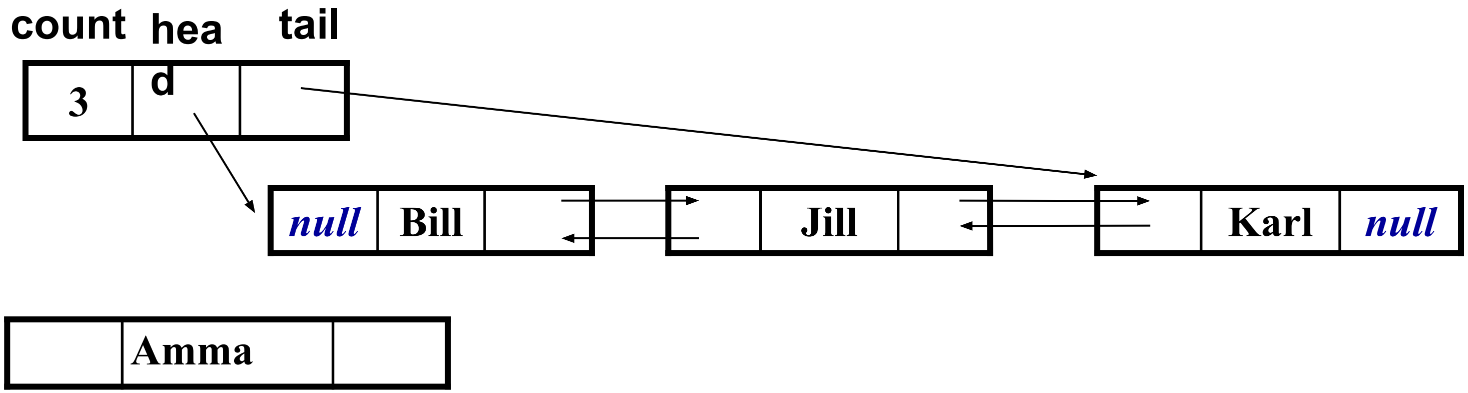
Linked list remove operation

- When removing a node from the list at a given index, the following steps must be taken:
  - Advance through the list to the node with the proper index.
  - Detach it from the nodes that used to precede and follow it.
    - How many 'arrows' (references) will need to be changed?



addFirst Method

Before:



count
hea
d
tail

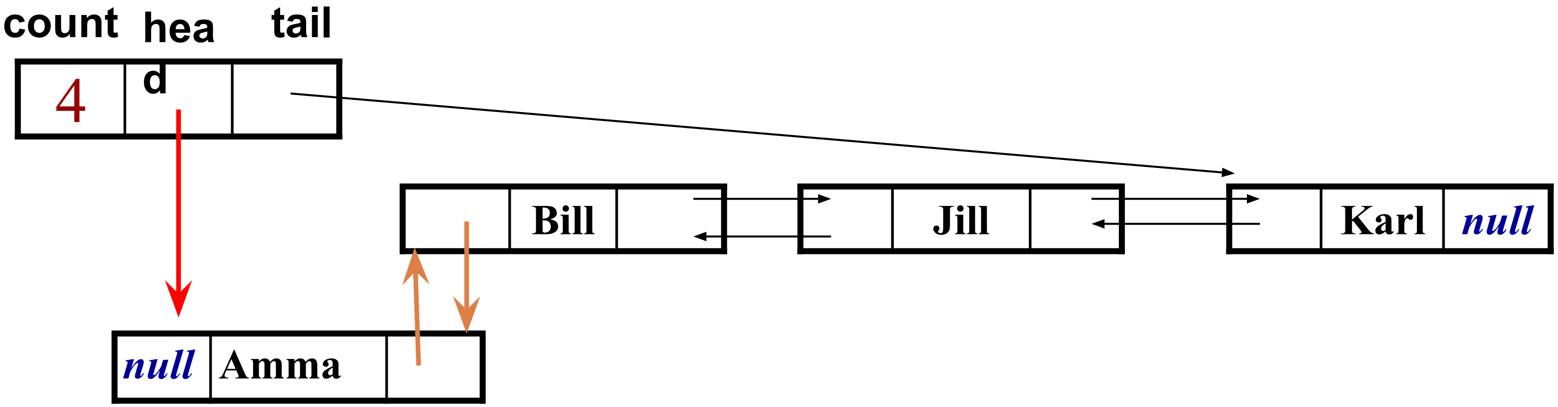
3

null
Bill
Jill
Karl
null

Amma

New value

After:



count
hea
d
tail

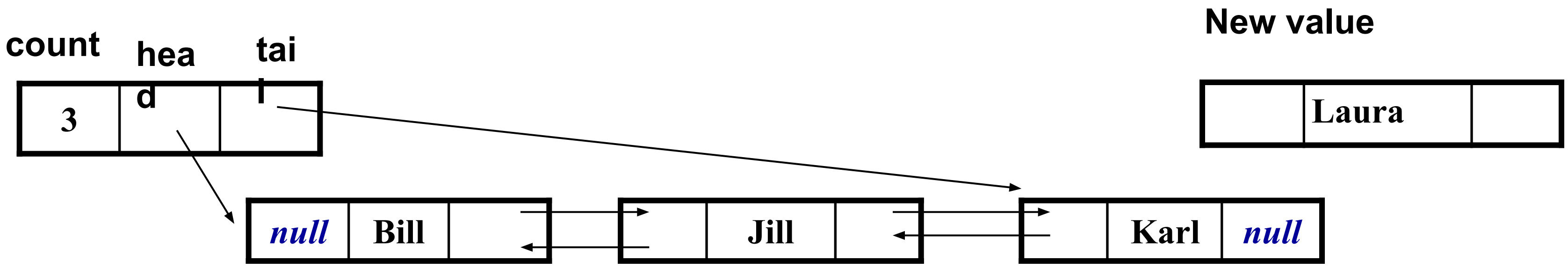
4

Bill
Jill
Karl
null

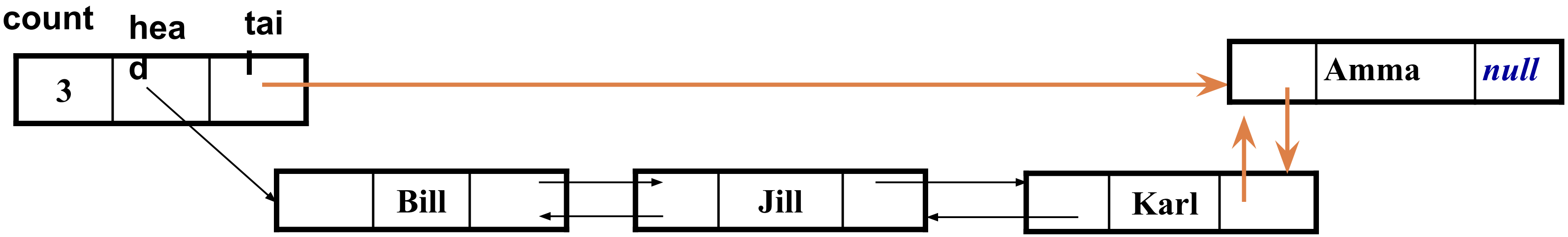
null
Amma

addLast method

Before:

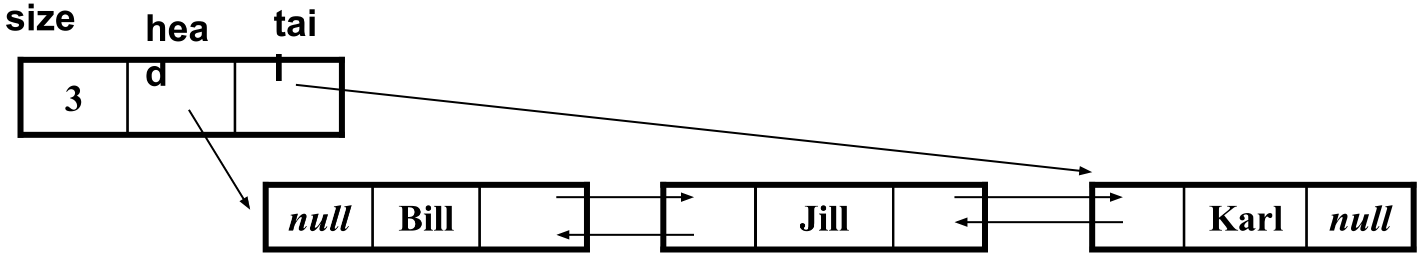


After:



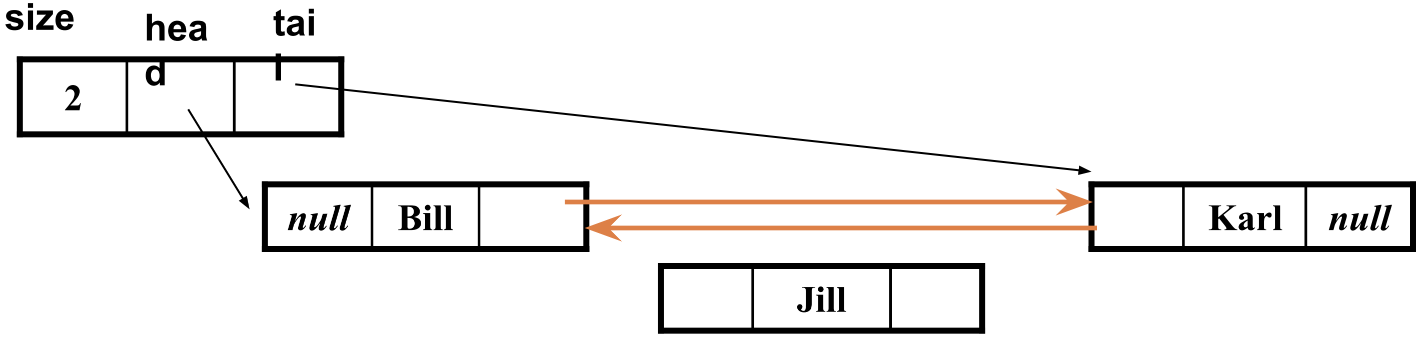
remove method

- Case 1: remove a middle element
  - Before:



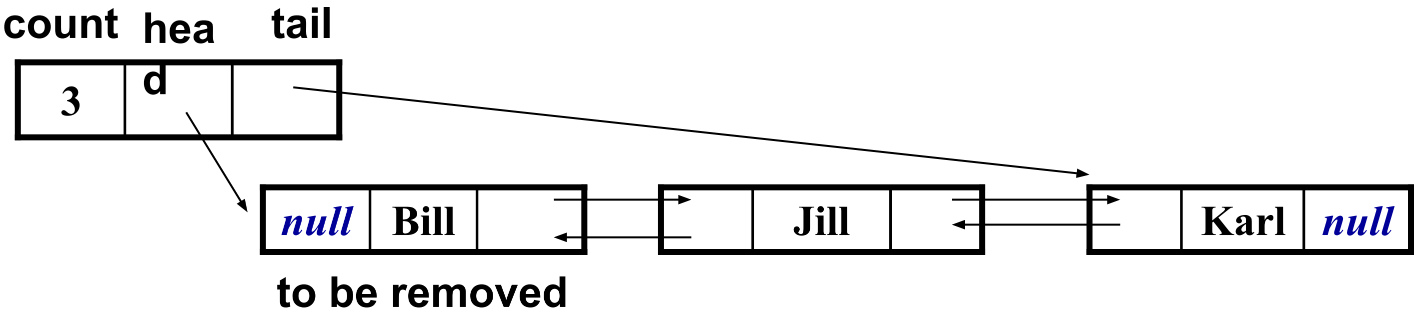
  - After:

to be removed

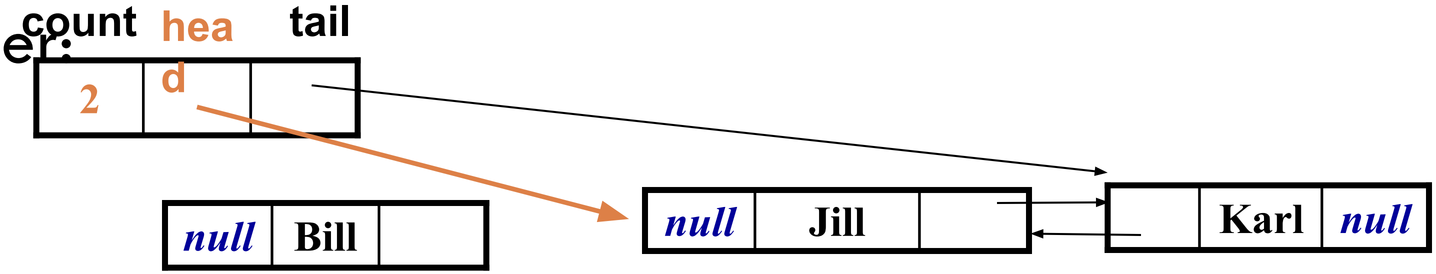


remove method

- Case 2: remove head element
  - Before:

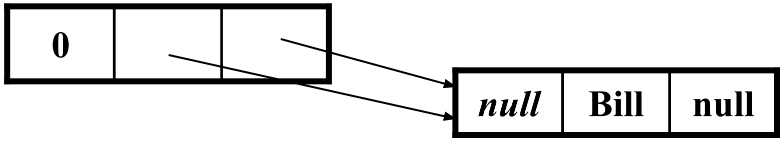


  - After:



remove method

- Case 3: remove the only element
  - Before:

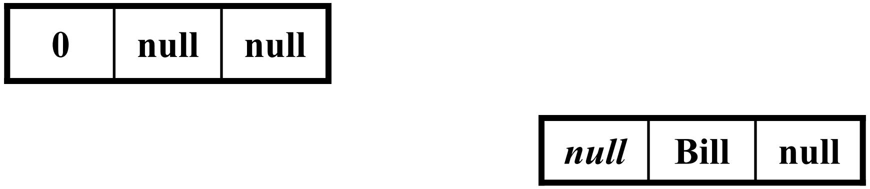


0

null Bill null

to be removed

  - After



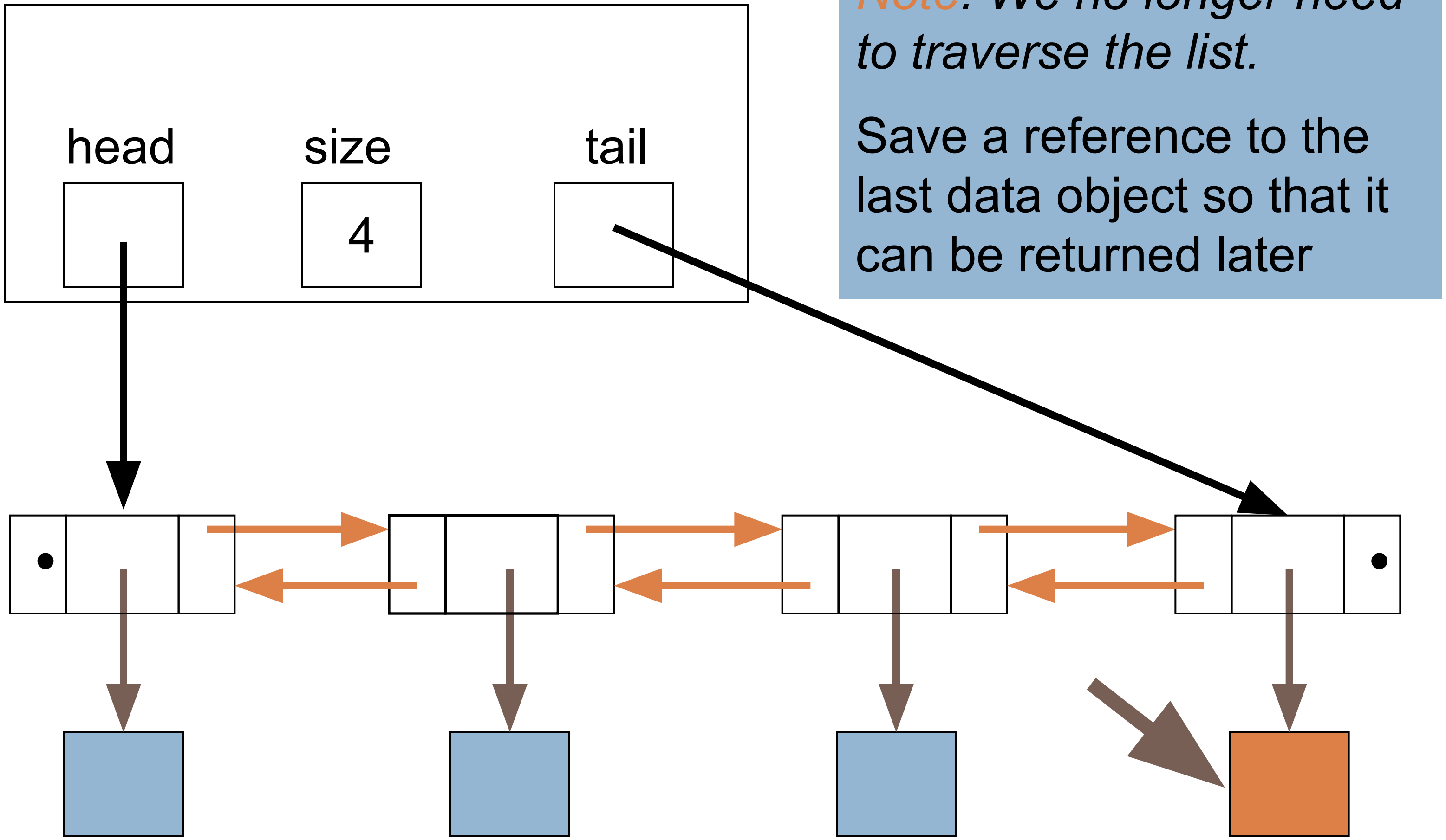
0 null null

null Bill null

To Remove Last Item From a Doubly-Linked List

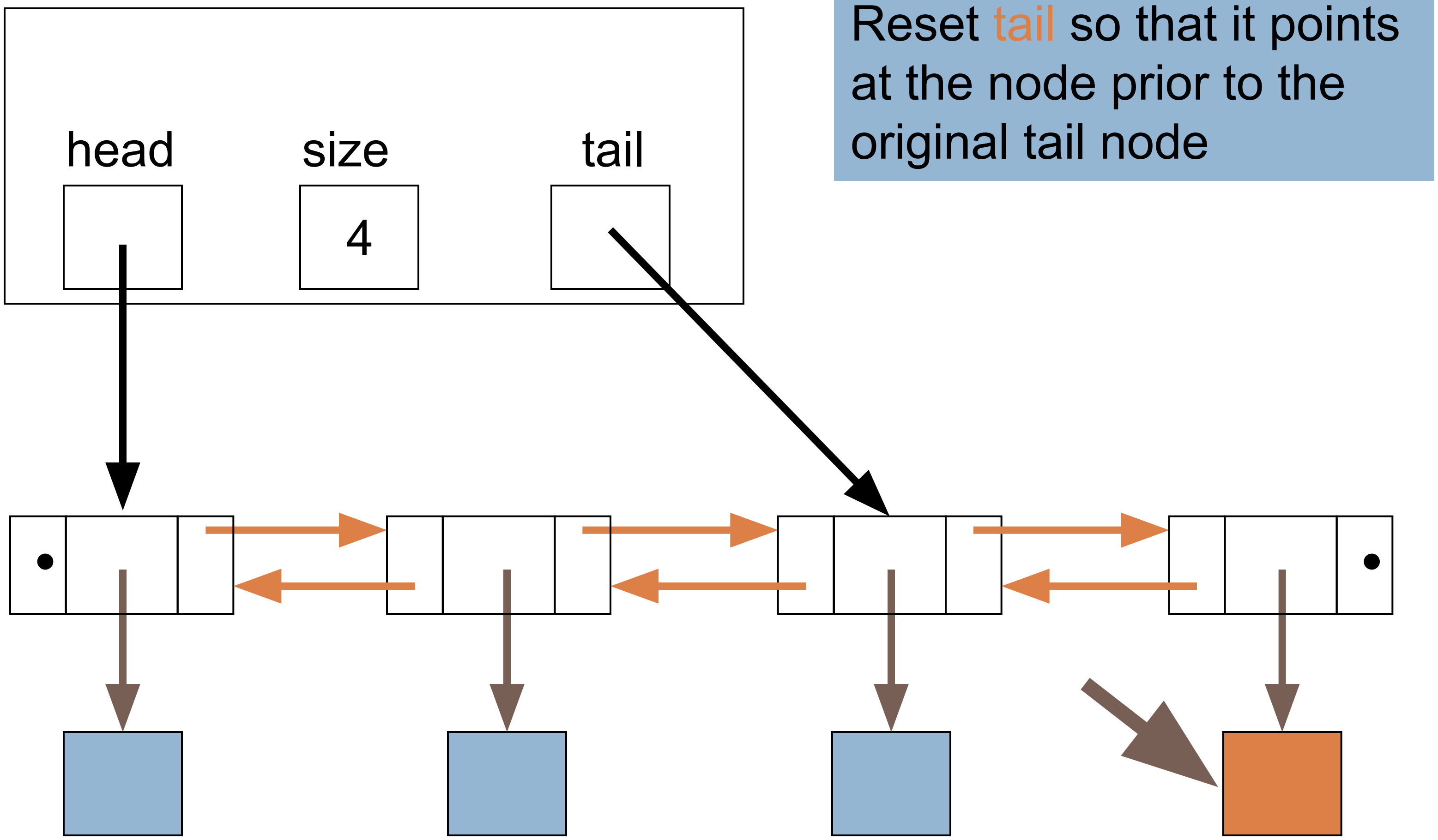
Note: We no longer need to traverse the list.

Save a reference to the last data object so that it can be returned later



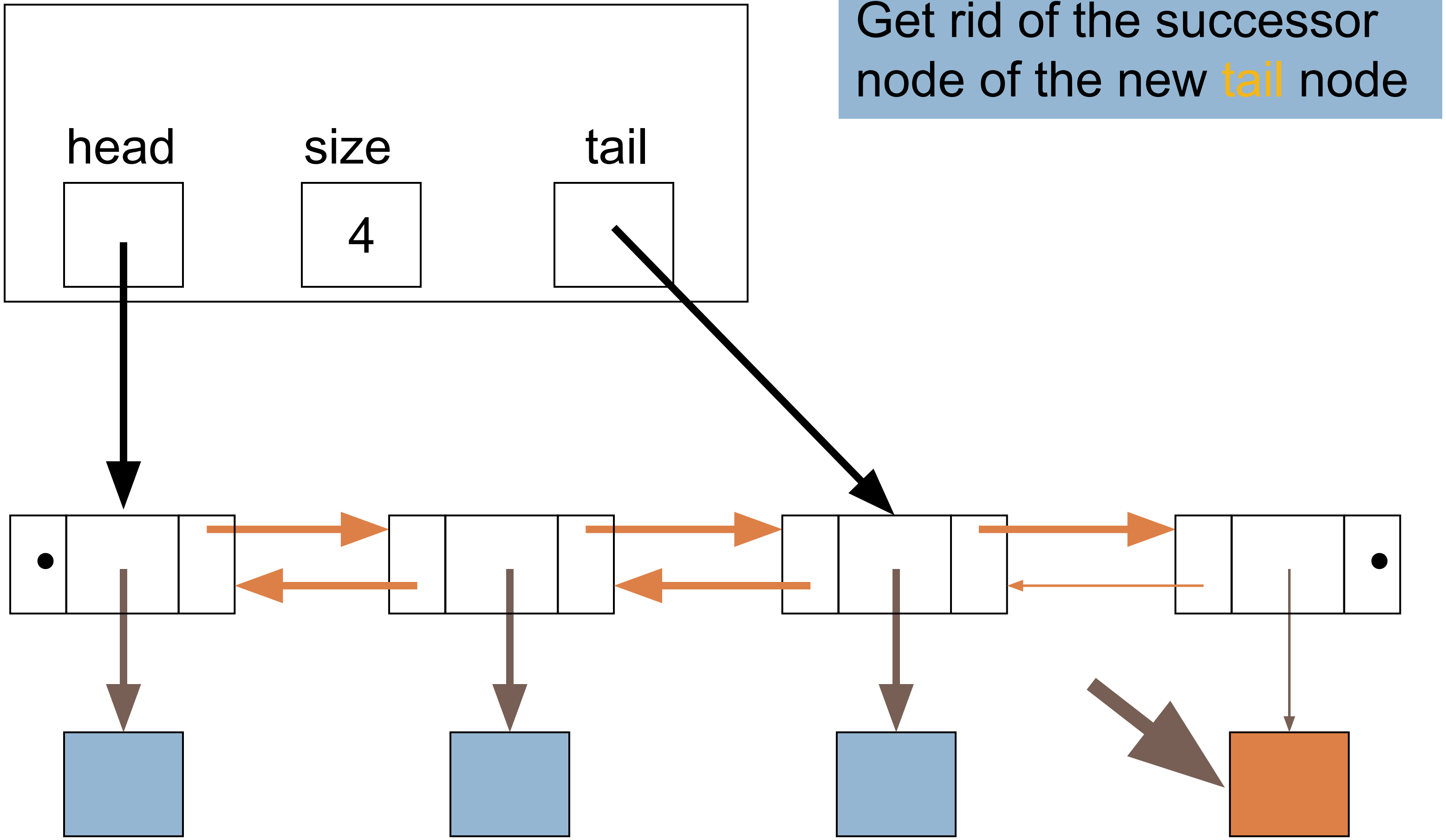
To Remove Last Item From a Doubly-Linked List

Reset tail so that it points at the node prior to the original tail node

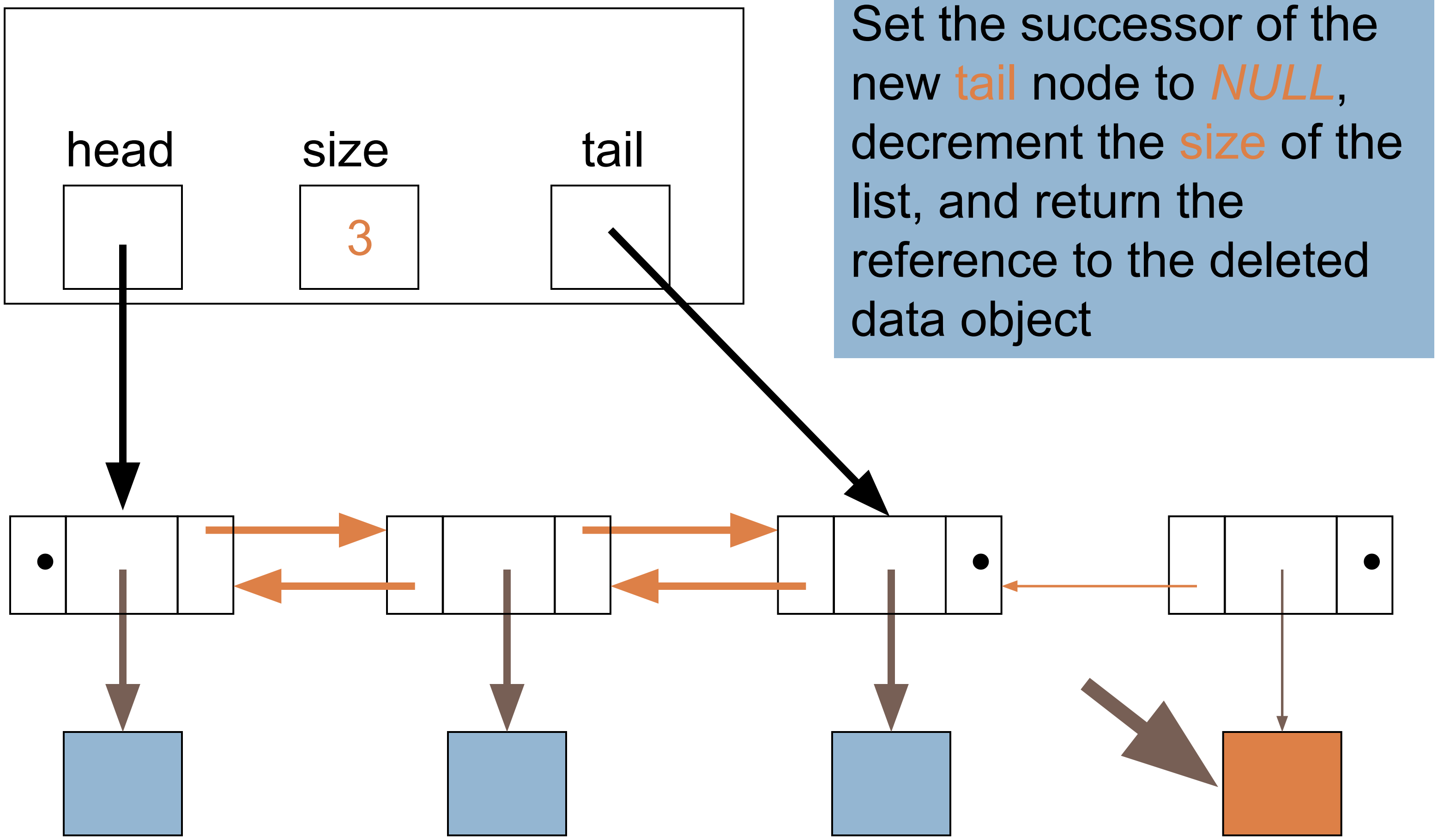


To Remove Last Item From a Doubly-Linked List

Get rid of the successor node of the new tail node



To Remove Last Item From a Doubly-Linked List



Set the successor of the new tail node to *NULL*, decrement the size of the list, and return the reference to the deleted data object

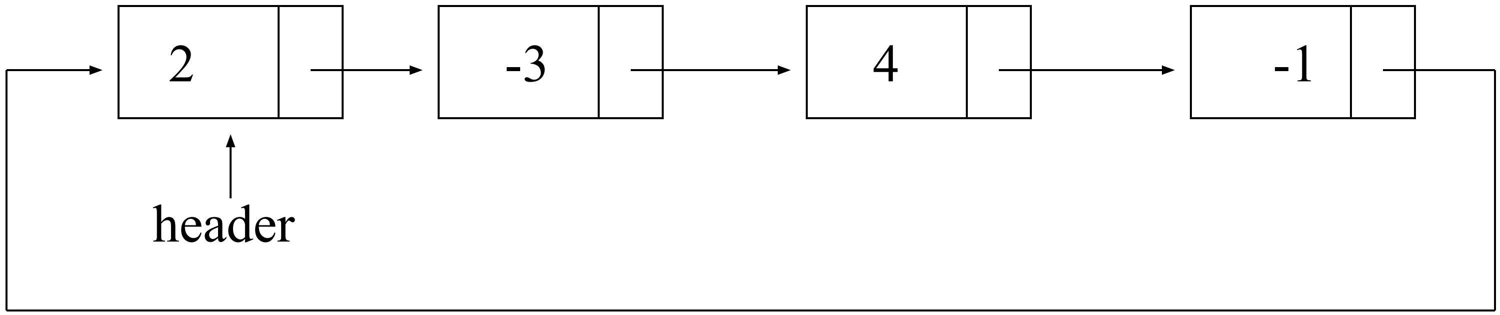
CIRCULAR LISTS

Circular Linked List

- A linked list in which the last node points to the first node is called a circular linked list

- In a circular linked list with more than one node, it is convenient to make the pointer first point to the last node of the list

A Circular Linked List



Have the last node link back to the first (or the header).

Circular Linked Lists

- Circular linked lists avoid the use of null references in their nodes
- Can be useful for certain algorithms
- Can also make programming simpler: fewer special cases to consider
- Successor of tail node is the head node; in doubly-linked list

Circular linked lists

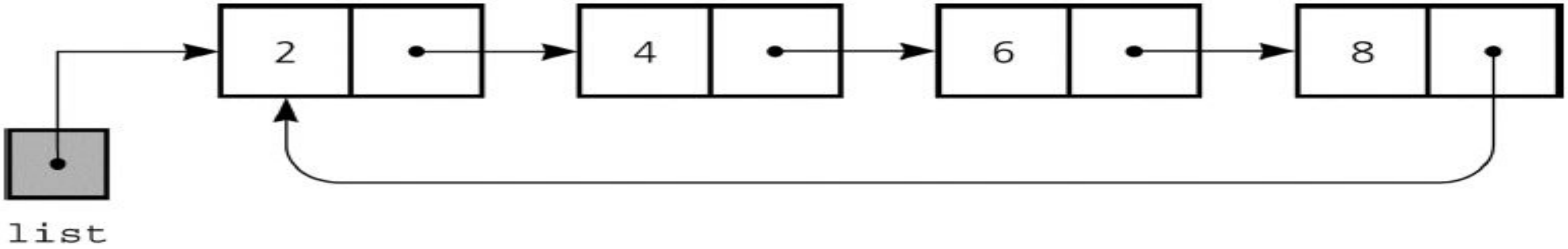
- Insertions and deletions into a circular linked list follow the same logic patterns used in a singly linked list except that the last node points to the first node.

- Therefore, when inserting or deleting the last node, in addition to updating the tail pointer, we must also point the link field of the new last node to the first node.

Circular linked list

- Advantage is that we can start searching from any node of the linked list and get to any other node.

- No logical head and tail pointers. We follow the conventions as first element inserted into the list is given status of head and last element entered is called as a tail.



Circular Linked Lists

- Last node references the first node
- Every node has a successor
- No node in a circular linked list contains *NULL*

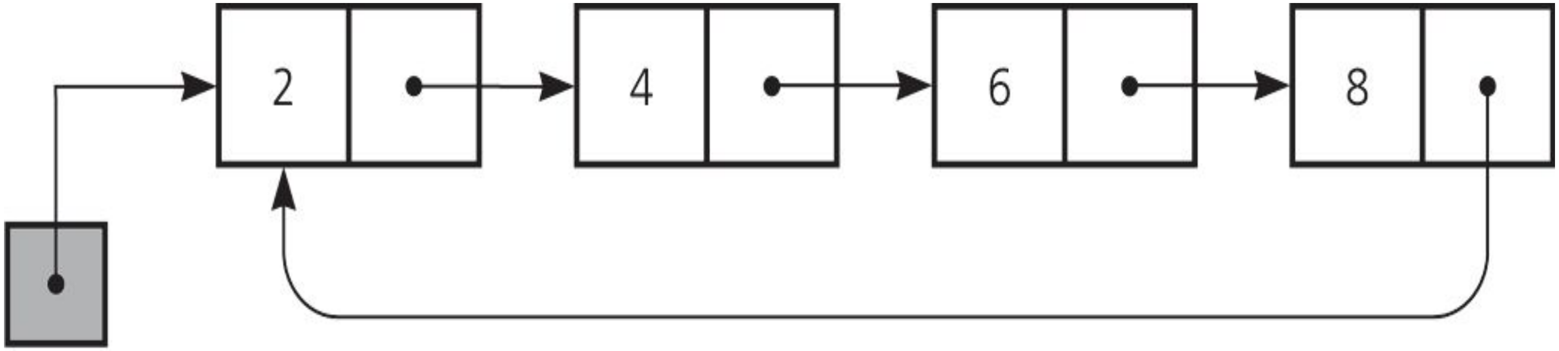
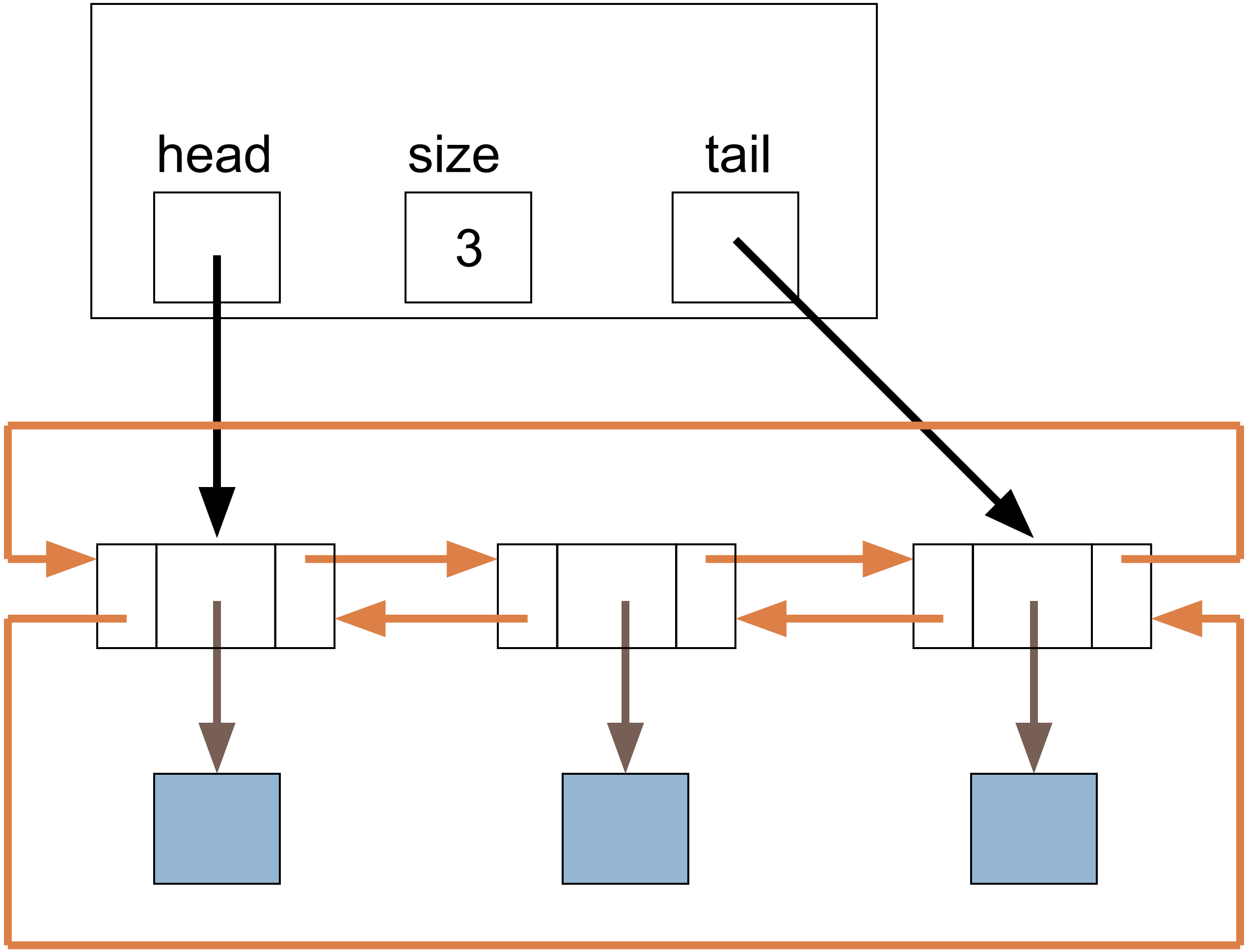


Figure 4.25 A circular linked list

Circular Doubly-Linked List



Header node

- Advantages of using header node

  - We can keep track of no of nodes in the list which is required for some operations.
  - We can eliminate sentinel testing and there by reduce the no of cases to be tested during insertion and deletion.
  - Structure of header node may be same as the nodes in the list or it we can define different structure.

Note: Refer to class notes for detailed discussion on different operations with different type of linked list.

Difference between single and double linked list

Single Linked list	Double linked list
-It is one way list with which we can traverse in only one direction(Forward) - Need to keep track of the address of Predecessor during new insertion or deletion Which is a tedious job. -Occupies less memory as there is only on link for each node - Need to take care of manipulating only one link.	-It is two way list with which we can traverse in both the directions(Forward and backward) - Since every node as address of predecessor and successor, we need not have to keep track of the address of predecessor during new insertion or deletion. -Occupies more memory as there are two links for each node -Extra care must be taken to manipulate both the links Which is a tedious job.

- Even though double linked list has certain problems with extra link manipulation and extra memory, It can be used to perform some of the operations efficiently.
  - Example
    - Insertion of a new node to the immediate left or right
    - Deletion of a node from immediate left or right of a node

There are many applications for which double linked list is required.