

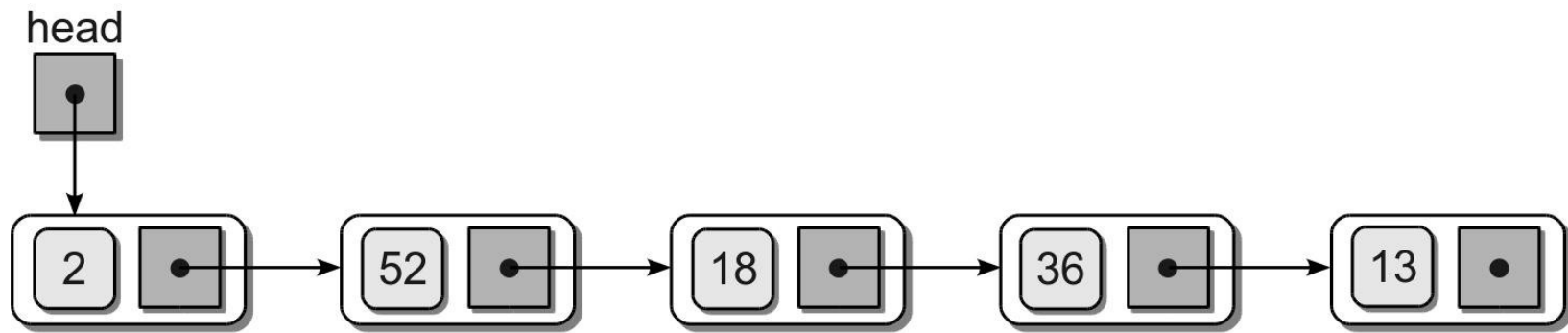
Linked Structures

Chapter 6



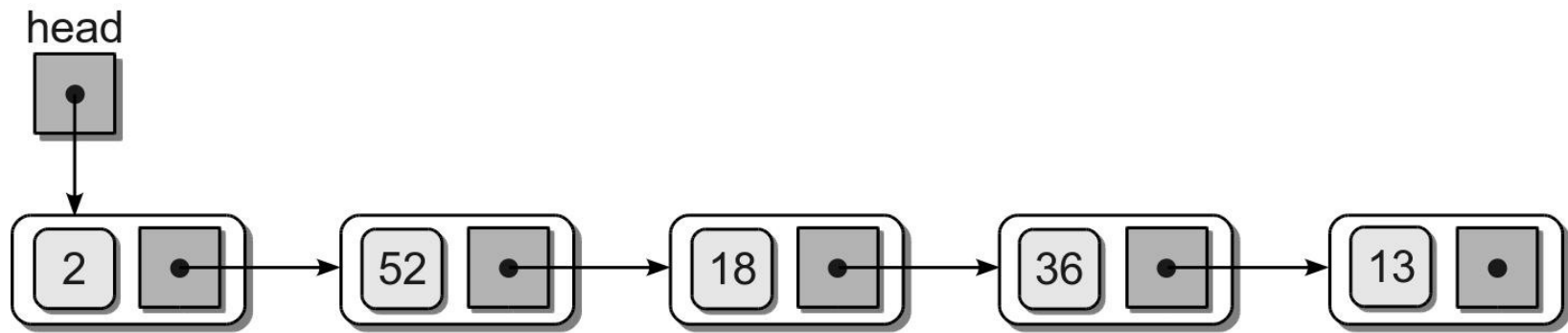
Linked Structure

- Constructed using a collection of objects called **nodes**.
- Each node contains data and at least one reference or **link** to another node.
- **Linked list** – a linked structure in which the nodes are linked together in linear order.



Linked List

- Terms:
 - **head** – first node in the list.
 - **tail** – last node in the list; link field has a null reference.



Node Definition

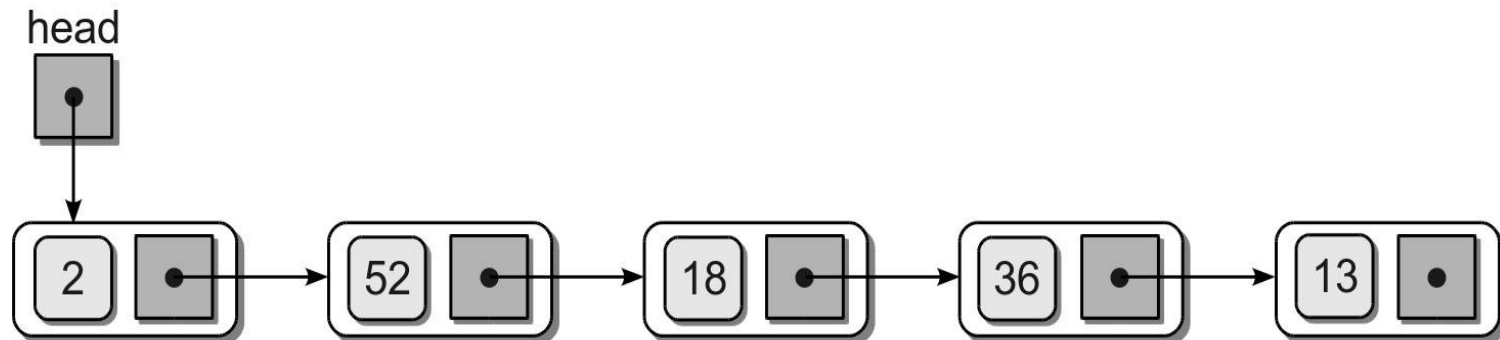
- The nodes are constructed from a simple storage class:

```
class _ListNode:  
    def __init__( self, data ):  
        self.data = data  
        self.next = None
```

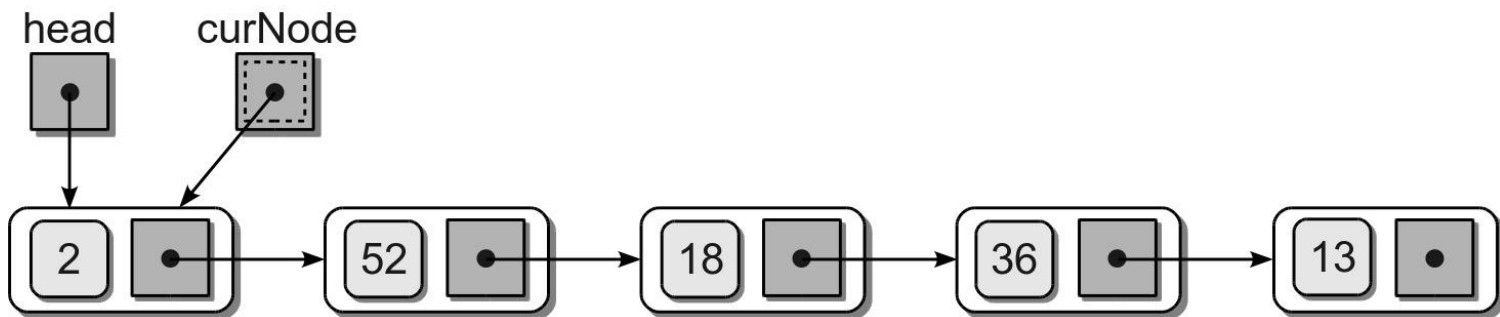


Traversing the Nodes

- We can traverse the nodes using a temporary external reference variable.



- Initialize a temporary reference to the head node.

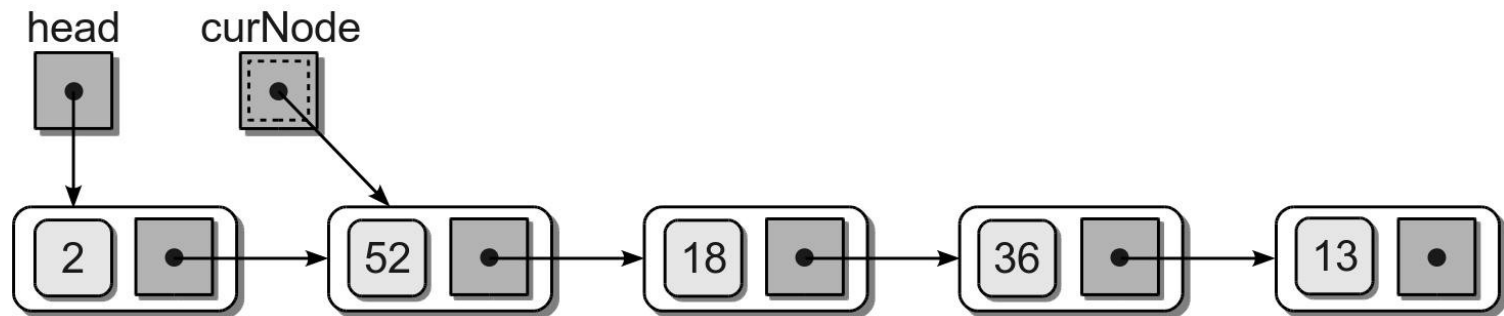


- Visit the node.



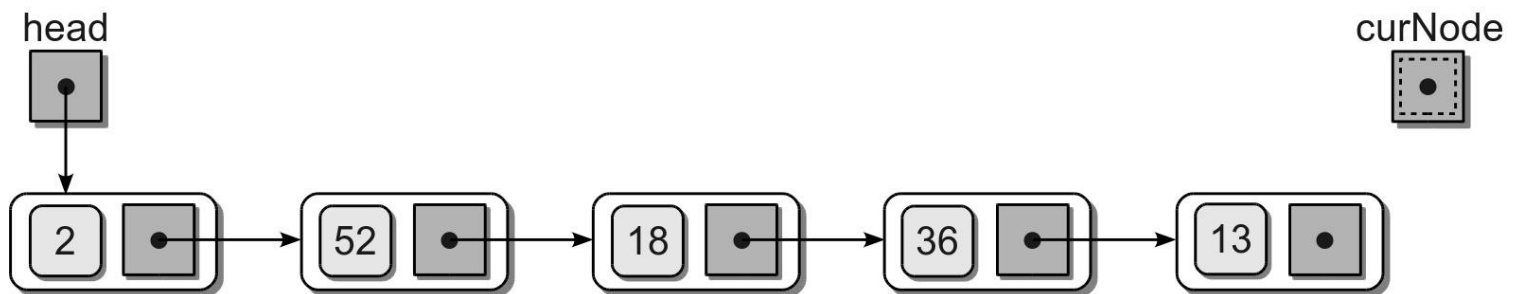
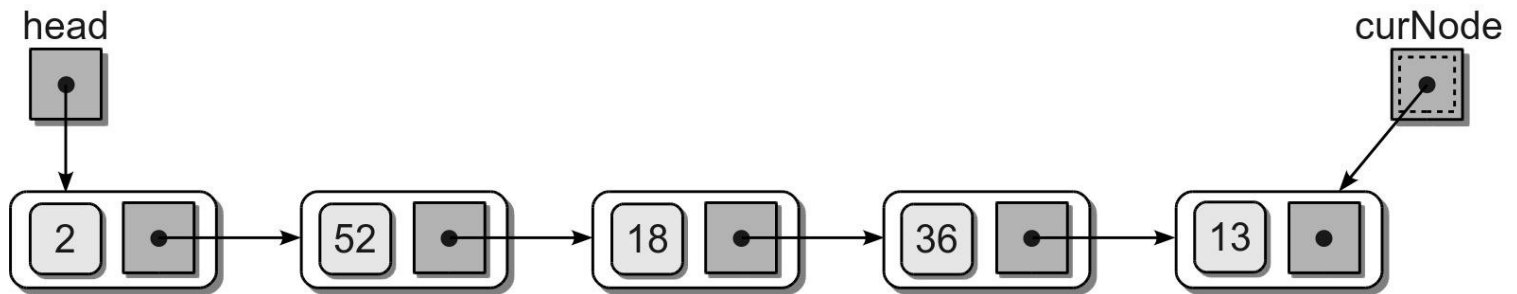
Traversing the Nodes

- Advance the temporary reference to the next node using the link field and visit that node.



Traversing the Nodes

- Repeat the process until the reference falls off the end of the list.



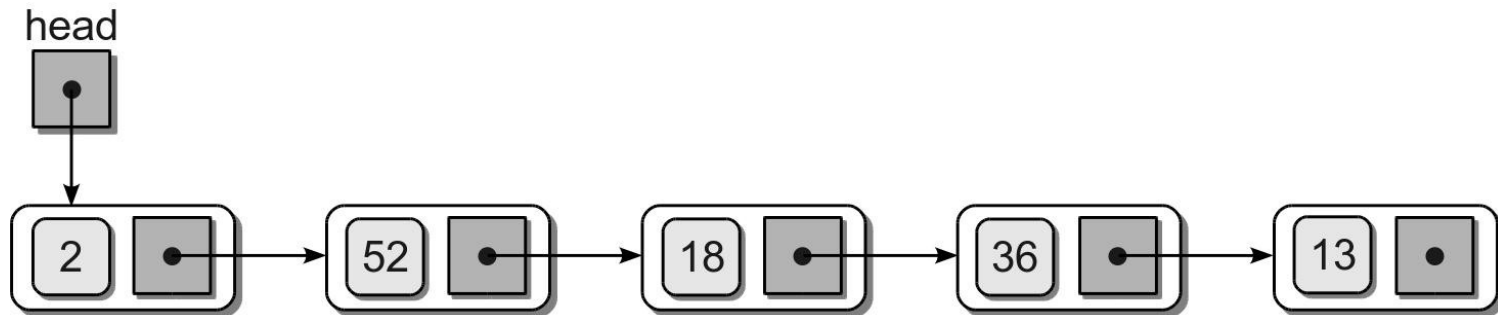
Searching

- We can perform a linear search to determine if the list contains a specific data item.



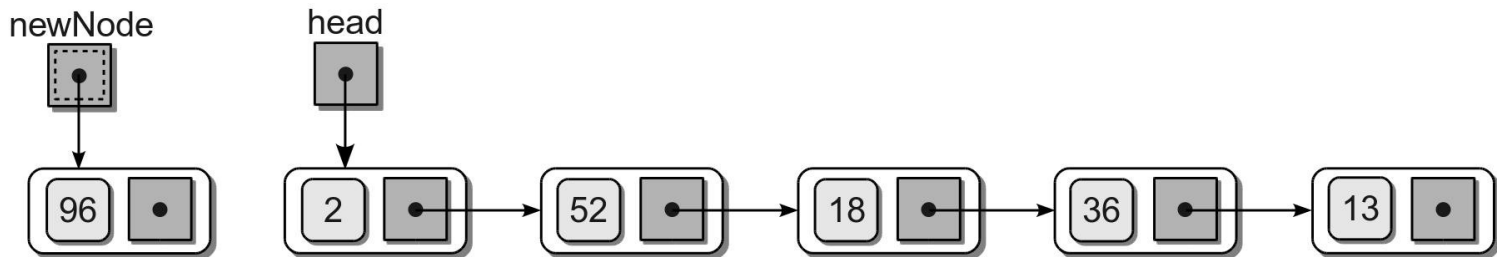
Prepending Nodes

- When working with an unsorted linked list, new values can be inserted at any point.
- We can prepend new items with little effort.
- **Example:** add value 96 to the sample list.

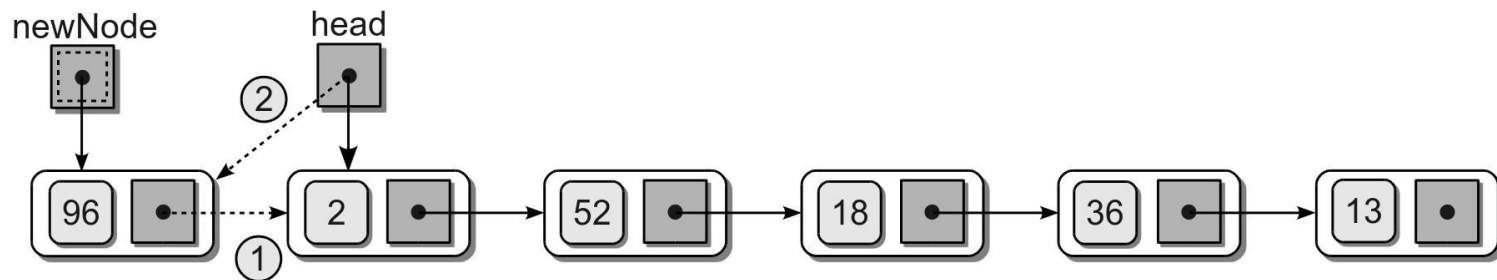


Prepending Nodes

- Create a new node for the new item.

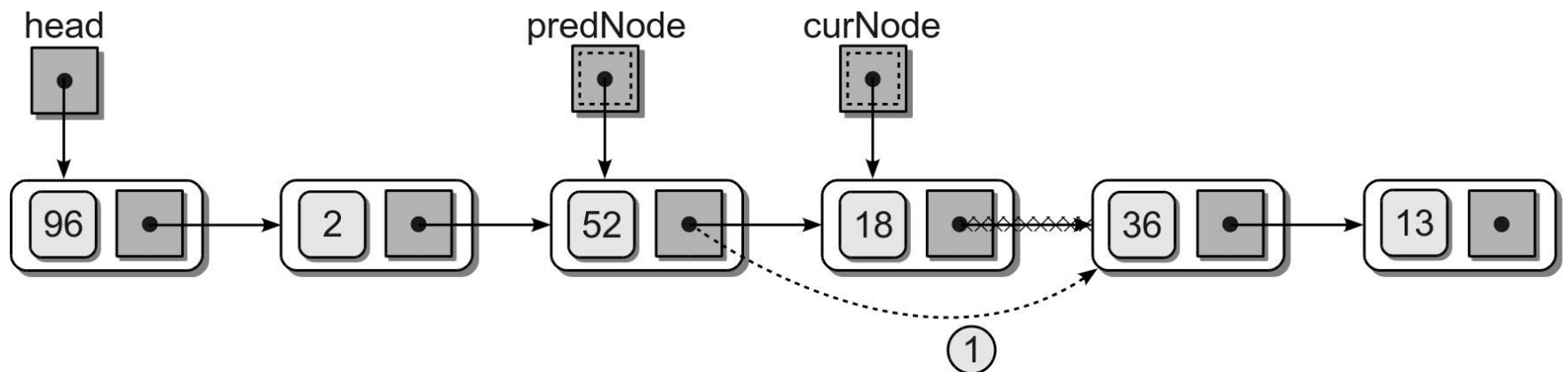


- Connect the new node to the list.

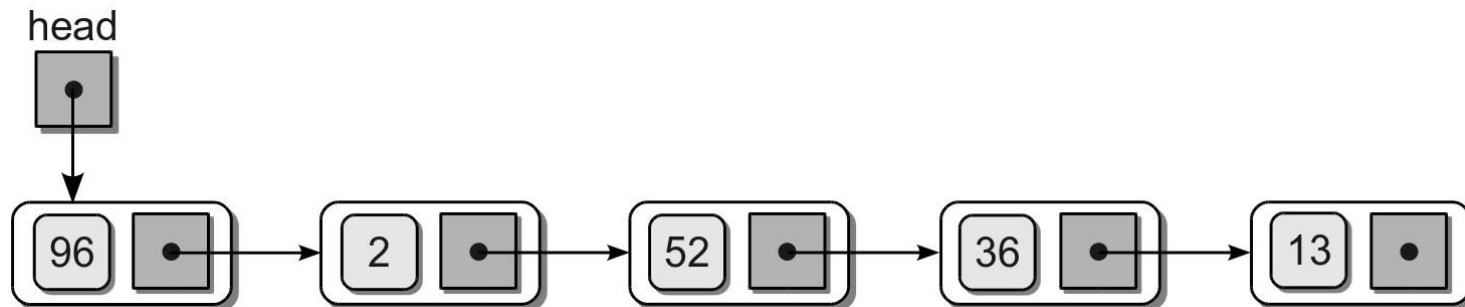


Removing Nodes

- Removing a node from the middle of the list requires a second external reference.

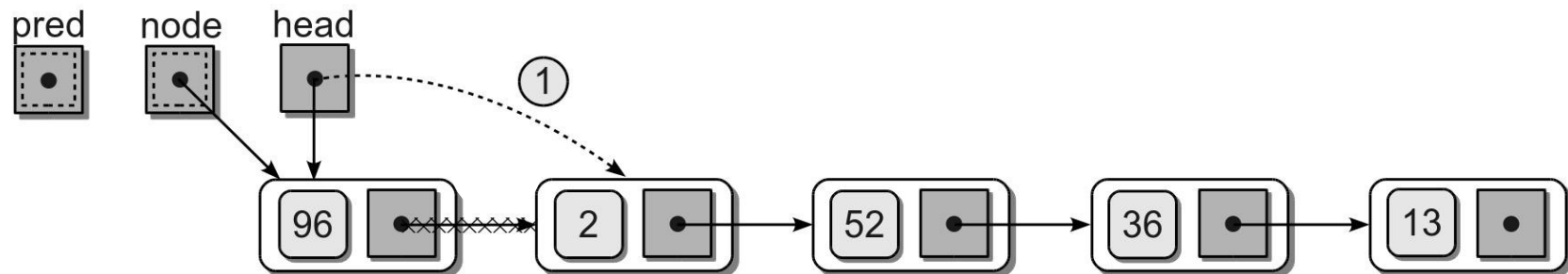


- Resulting list.



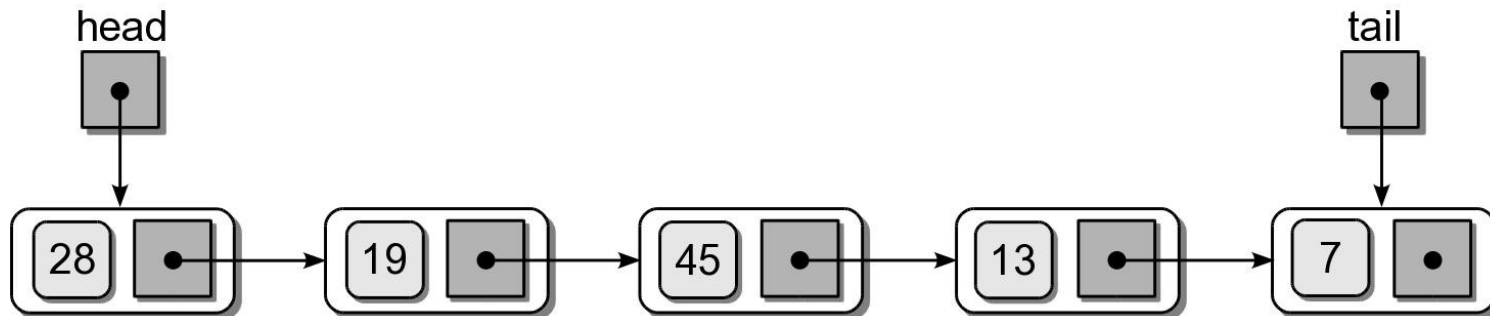
Removing Nodes

- Removing the first node is a special case.
- The head reference must be repositioned to reference the next node in the list.



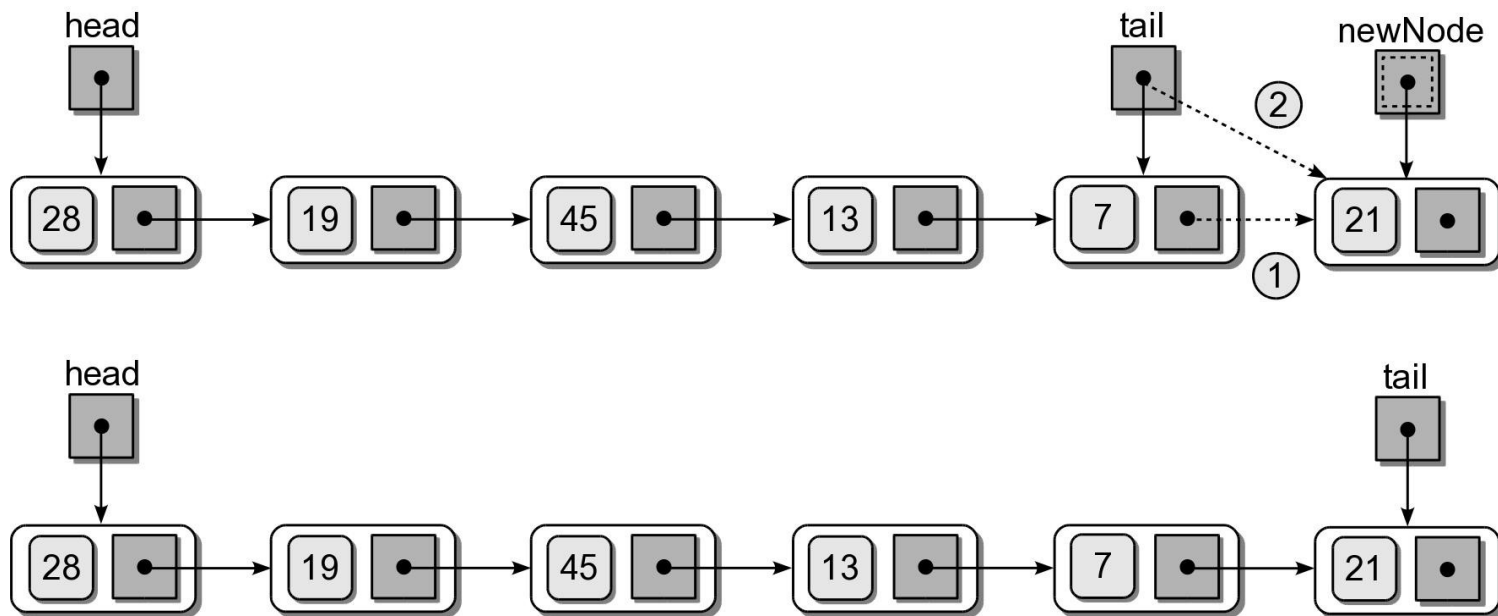
Using a Tail Reference

- Some applications require items be appended to the end of the linked list.
 - **tail reference** – a second external reference indicating the tail or last node in the list.



Appending Nodes

- Must manage the tail reference as nodes are added/removed.
 - **Example:** append 21 to the list.



Appending Nodes

- Given the head and tail reference, we can add an item to a linked list.

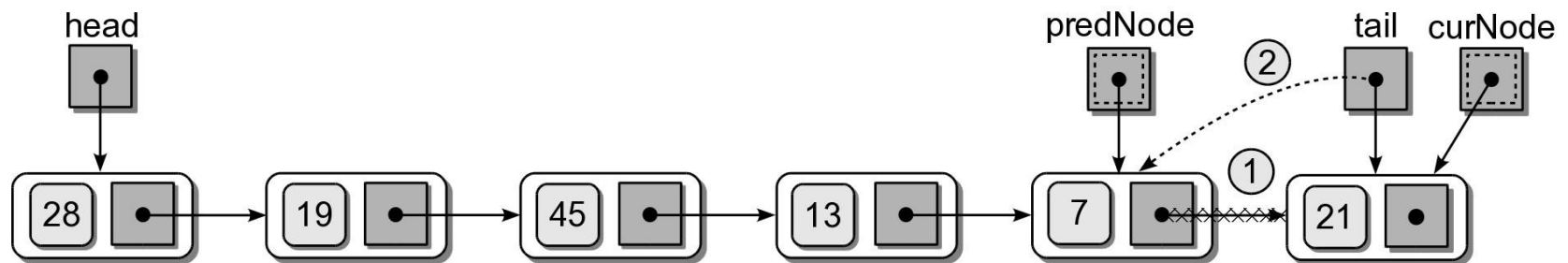
```
newNode = ListNode( item )
if self._head is None :
    self._head = newNode
    self._tail = newNode
else :
    self._tail.next = newNode
    self._tail = newNode
```

What is the time complexity to append a node to a linked list, if no tail reference is used?



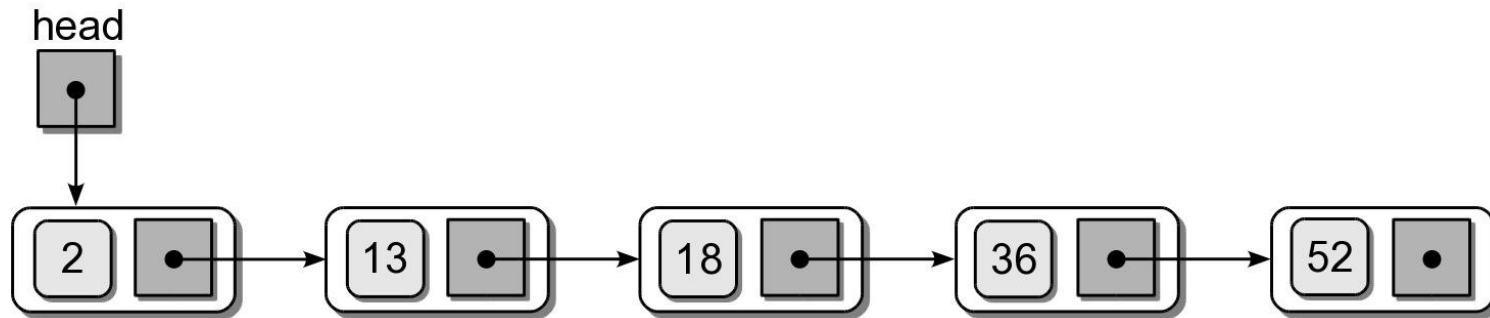
Removing Nodes

- If the tail node is removed, the tail reference has to be adjusted.



The Sorted Linked List

- The items in a linked list can be maintained in sorted order.



Sorted List: Searching

- Searching a sorted list is similar to that of an unsorted list.

```
def sortedSearch( head, target ) :  
    curNode = head  
  
    # Stop early when a larger value is encountered.  
while curNode is not None and \  
        target <= curNode.data :  
    if curNode.data == target :  
        return True  
    else :  
        curNode = node.next  
  
return False
```



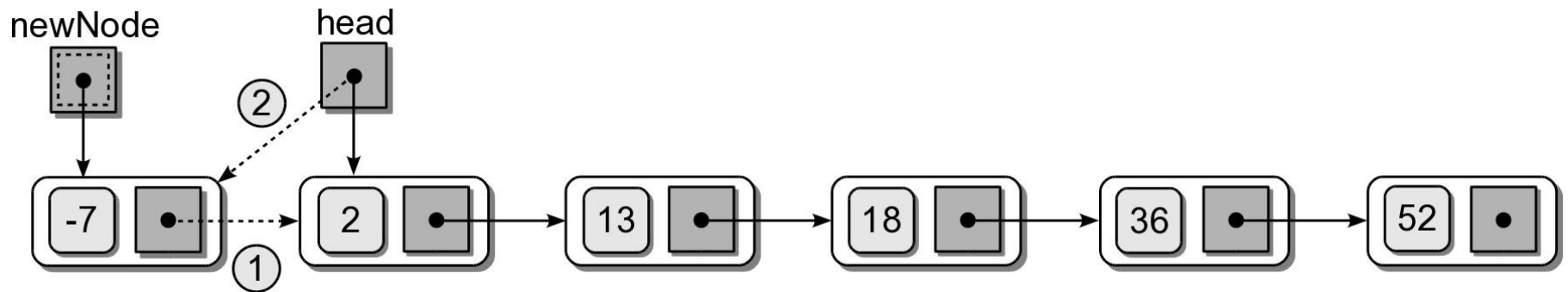
Sorted List: Insert

- Adding a new node to a sorted list requires locating the correct position within the list.
 - Locating the position is similar to the removal operation.
 - Use a second temporary reference for the predecessor.
- There are 3 possible cases.
 - front
 - middle
 - back



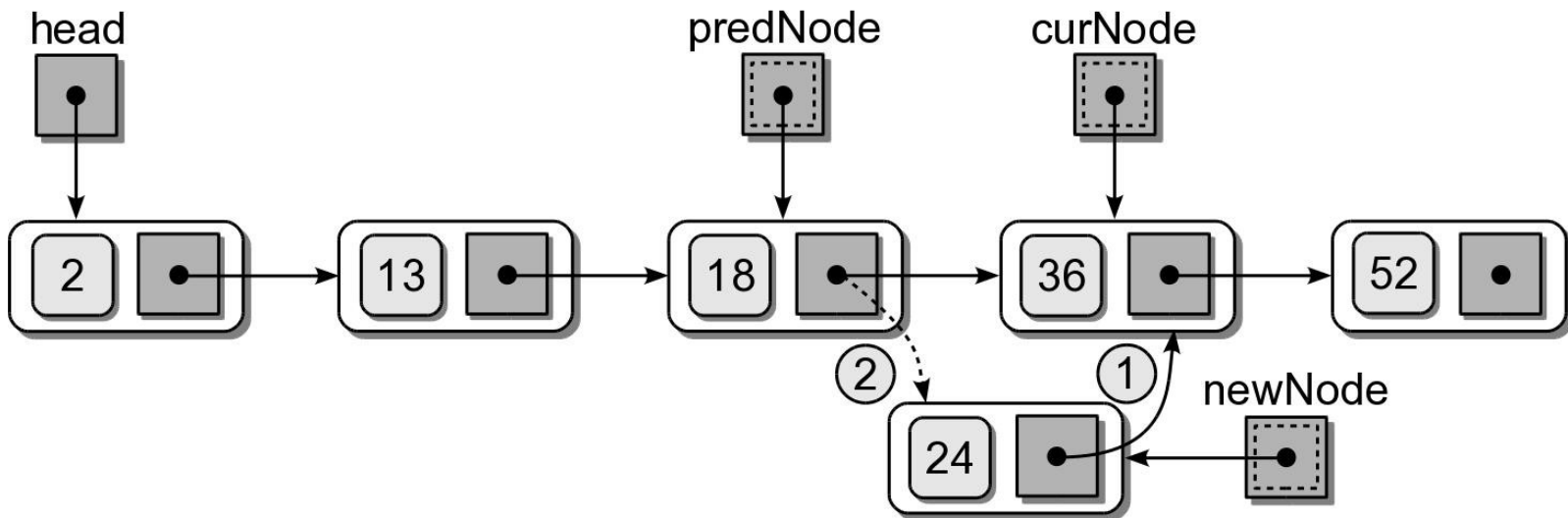
Sorted List: Insert

- (1) Insert at the front.



Sorted List: Insert

- (2) Insert in the middle.



Singly Linked List / Python List Comparison

Operation	Linked List	Python List
append(item)		
insert(0, item)		
pop(0)		
pop(i)		
__getitem__(i)		
__setitem__(i, item)		

- When does a Linked List make more sense than a contiguous representation?



Singly Linked List / Python List Comparison

Operation	Linked List	Python List
append(item)	$O(n)^*$	$O(1)^{**}$
insert(0, item)	$O(1)$	$O(n)$
pop(0)	$O(1)$	$O(n)$
pop(i)	$O(n)$	$O(n)$
__getitem__(i)	$O(n)$	$O(1)$
__setitem__(i, item)	$O(n)$	$O(1)$

- When does a Linked List make more sense than a contiguous representation?

* $O(1)$ with tail reference.

** Amortized

