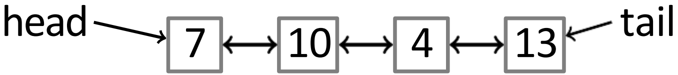
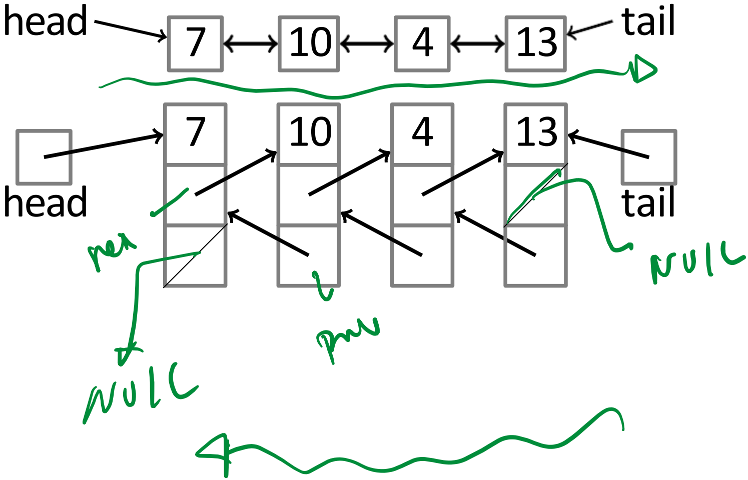
Doubly-Linked List

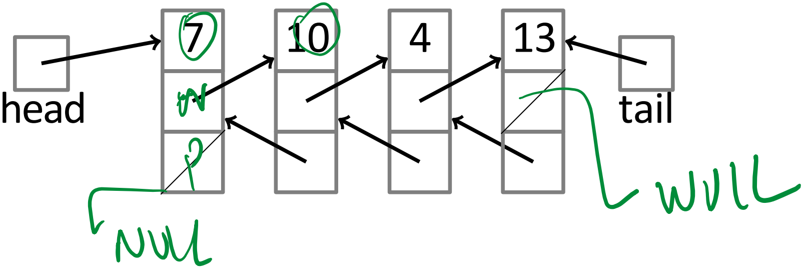


head 7 10 4 13 tail

Doubly-Linked List



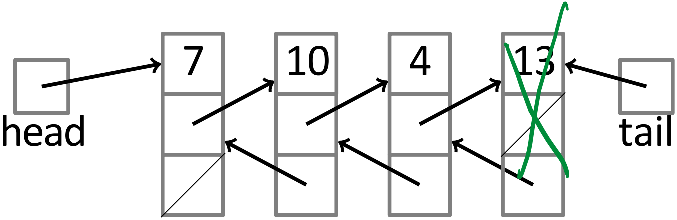
Doubly-Linked List



Node contains:

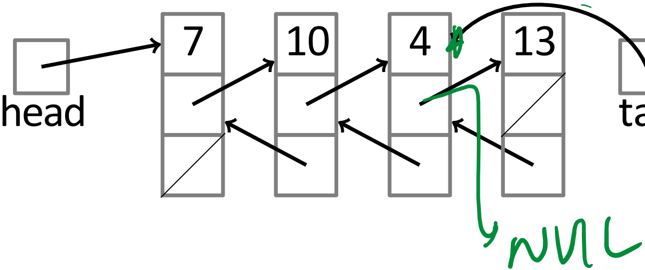
- key
- next pointer
- prev pointer

Doubly-Linked List



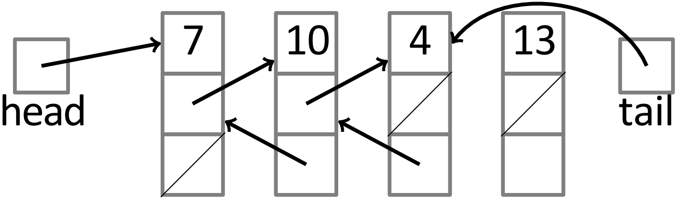
PopBack

Doubly-Linked List



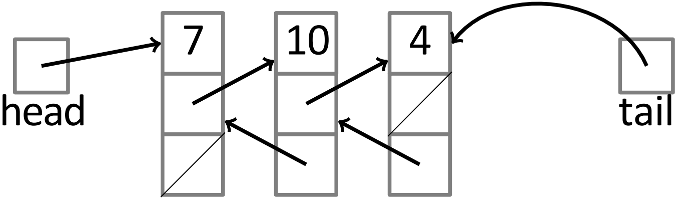
PopBack

Doubly-Linked List



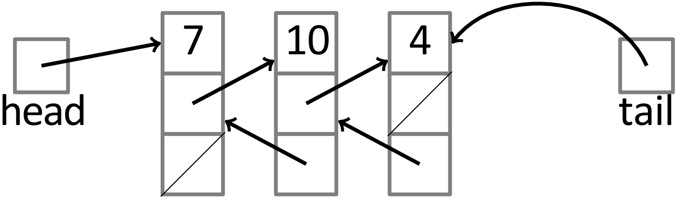
PopBack

Doubly-Linked List



PopBack

Doubly-Linked List



PopBack $O(1)$

Doubly-linked List

PopBack()

Doubly-linked List

PopBack()

```
if head = nil:  ERROR: empty list
```

Doubly-linked List

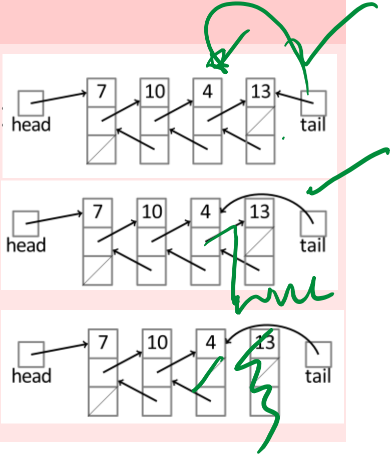
PopBack()

```
if $head$ = nil:  ERROR: empty list
if $head = tail$:
  $head \leftarrow tail \leftarrow$ nil
```

Doubly-linked List

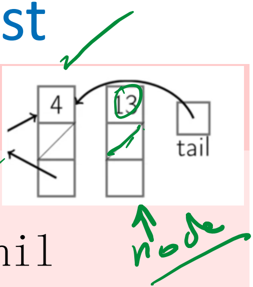
PopBack()

```
if $head$ = nil:   ERROR:
if $head = tail$:
    $head \leftarrow tail \leftarrow$ nil
else:
    $tail \leftarrow tail.prev$
    $tail.next \leftarrow$ nil
```



Doubly-linked List

PushBack(*key*)

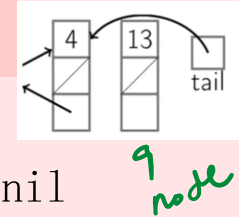


node ←new node

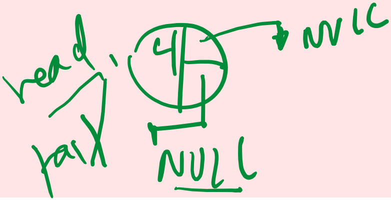
node.key ← *key* ; *node.next* =nil

Doubly-linked List

PushBack(*key*)



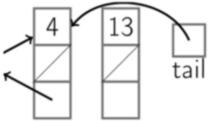
```
node ←new node   ✓
node.key ← key ;  node.next =nil
if tail = nil  (list empty single node)
    head ← tail ← node
    node.prev ←nil
```

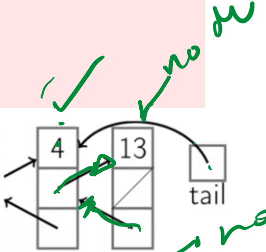


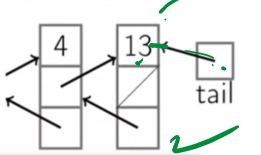
Doubly-linked List

PushBack(*key*)

```
node ← new node
node.key ← key;  node.next = nil
if tail = nil: (list empty single node)
    head ← tail ← node
    node.prev ← nil
else:
    tail.next ← node
    node.prev ← tail
    tail ← node
```



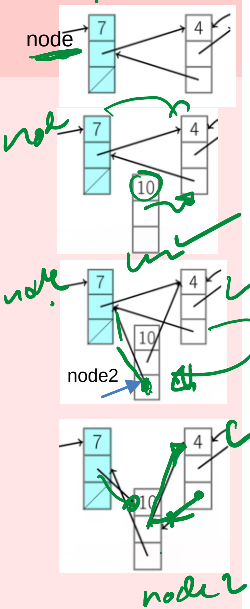




Doubly-linked List

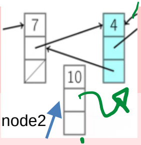
```
AddAfter(node, key)
node2 ← new node
node2.key ← key
node2.next ← node.next
node2.prev ← node
node.next ← node2
if node2.next ≠ nil:
    node2.next.prev ← node2
if tail = node:
    tail ← node2
```

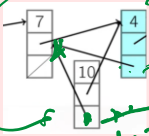
one node



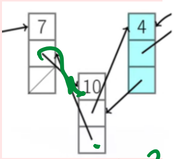
Doubly-linked List

```
AddBefore(node, key)
                  10
node2 ← new node
node2.key ← key
node2.next ← node
node2.prev ← node.prev
node.prev ← node2
if node2.prev ≠ nil:
    node2.prev.next ← node2
if head = node:
    (if we are adding before the head)
    head ← node2
```





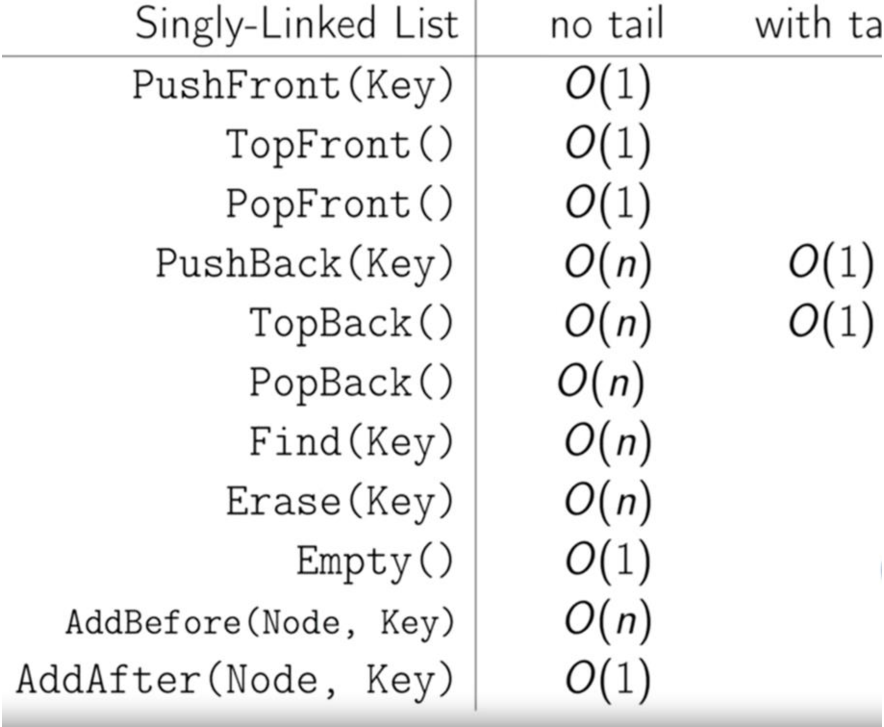
node2



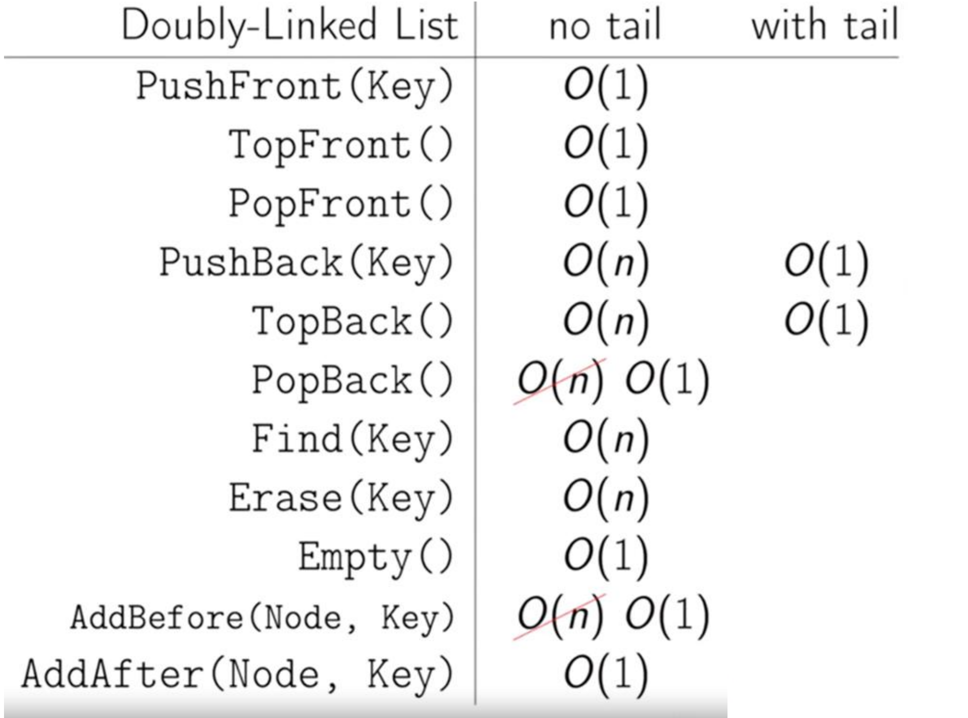
node

node 2

head



Singly-Linked List	no tail	with ta
`PushFront(Key)`	$O(1)$	
`TopFront()`	$O(1)$	
`PopFront()`	$O(1)$	
`PushBack(Key)`	$O(n)$	$O(1)$
`TopBack()`	$O(n)$	$O(1)$
`PopBack()`	$O(n)$	
`Find(Key)`	$O(n)$	
`Erase(Key)`	$O(n)$	
`Empty()`	$O(1)$	
`AddBefore(Node, Key)`	$O(n)$	
`AddAfter(Node, Key)`	$O(1)$	



Doubly-Linked List	no tail	with tail
`PushFront(Key)`	$O(1)$	
`TopFront()`	$O(1)$	
`PopFront()`	$O(1)$	
`PushBack(Key)`	$O(n)$	$O(1)$
`TopBack()`	$O(n)$	$O(1)$
`PopBack()`	$\cancel{O(n)}$ $O(1)$	
`Find(Key)`	$O(n)$	
`Erase(Key)`	$O(n)$	
`Empty()`	$O(1)$	
`AddBefore(Node, Key)`	$\cancel{O(n)}$ $O(1)$	
`AddAfter(Node, Key)`	$O(1)$	

Summary

- Constant time to insert at or remove from the front.
- With tail and doubly-linked, constant time to insert at or remove from the back.
- $O(n)$ time to find arbitrary element.
- List elements need not be contiguous.
- With doubly-linked list, constant time to insert between nodes or remove a node.