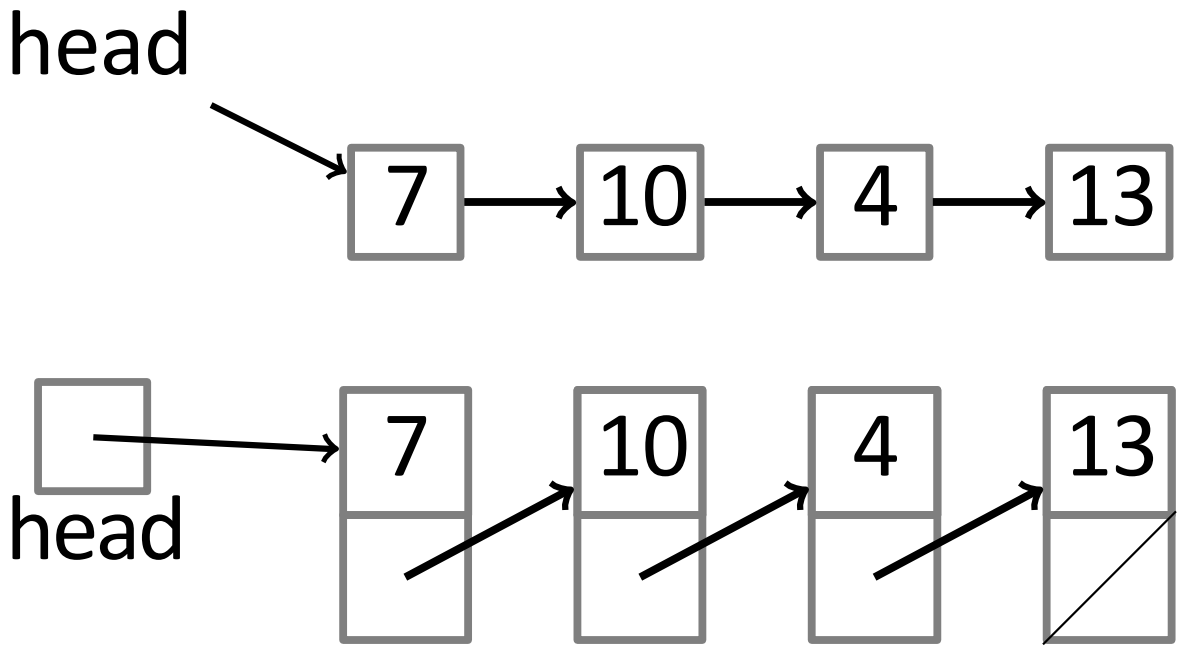
Singly-Linked List



Node contains:

- `key`
- `next` pointer

List API

PushFront(Key) add to front

List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item

List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item

List API

PushFront(Key)	add to front
Key TopFront()	return front item
PopFront()	remove front item
PushBack(Key)	add to back

also known as Append

List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item
`PushBack(Key)`	add to back
`Key TopBack()`	return back item

List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item
`PushBack(Key)`	add to back
`Key TopBack()`	return back item
`PopBack()`	remove back item

List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item
`PushBack(Key)`	add to back
`Key TopBack()`	return back item
`PopBack()`	remove back item
`Boolean Find(Key)`	is key in list?

List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item
`PushBack(Key)`	add to back
`Key TopBack()`	return back item
`PopBack()`	remove back item
`Boolean Find(Key)`	is key in list?
`Erase(Key)`	remove key from list

List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item
`PushBack(Key)`	add to back
`Key TopBack()`	return back item
`PopBack()`	remove back item
`Boolean Find(Key)`	is key in list?
`Erase(Key)`	remove key from list
`Boolean Empty()`	empty list?

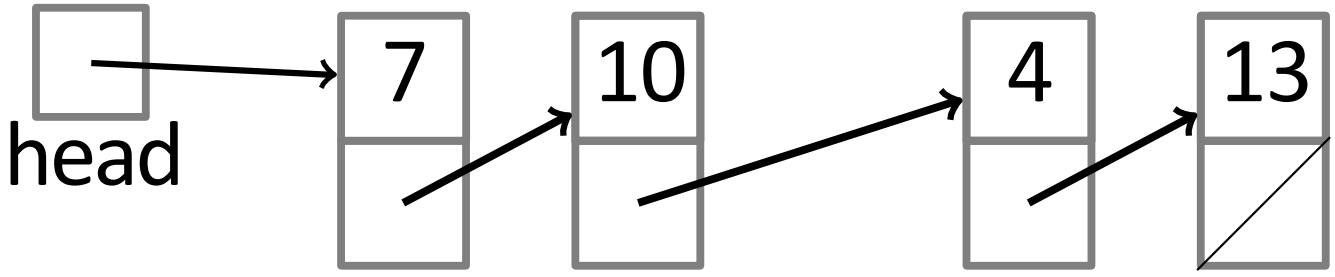
List API

`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item
`PushBack(Key)`	add to back
`Key TopBack()`	return back item
`PopBack()`	remove back item
`Boolean Find(Key)`	is key in list?
`Erase(Key)`	remove key from list
`Boolean Empty()`	empty list?
`AddBefore(Node, Key)`	adds key before node

List API

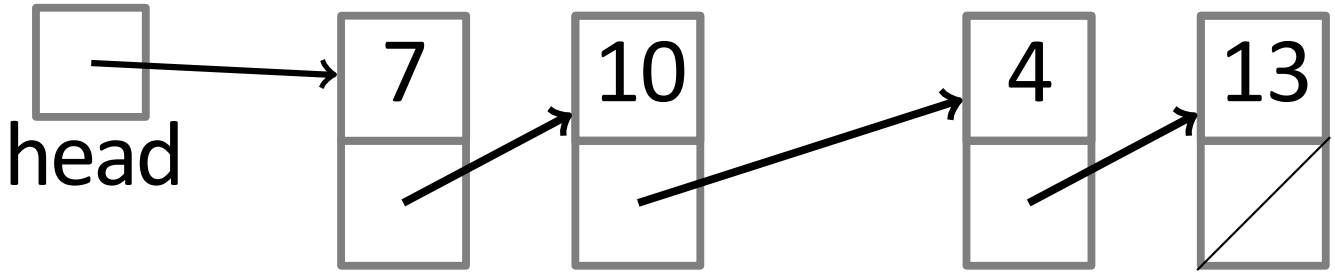
`PushFront(Key)`	add to front
`Key TopFront()`	return front item
`PopFront()`	remove front item
`PushBack(Key)`	add to back
`Key TopBack()`	return back item
`PopBack()`	remove back item
`Boolean Find(Key)`	is key in list?
`Erase(Key)`	remove key from list
`Boolean Empty()`	empty list?
`AddBefore(Node, Key)`	adds key before node
`AddAfter(Node, Key)`	adds key after node

Times for Some Operations



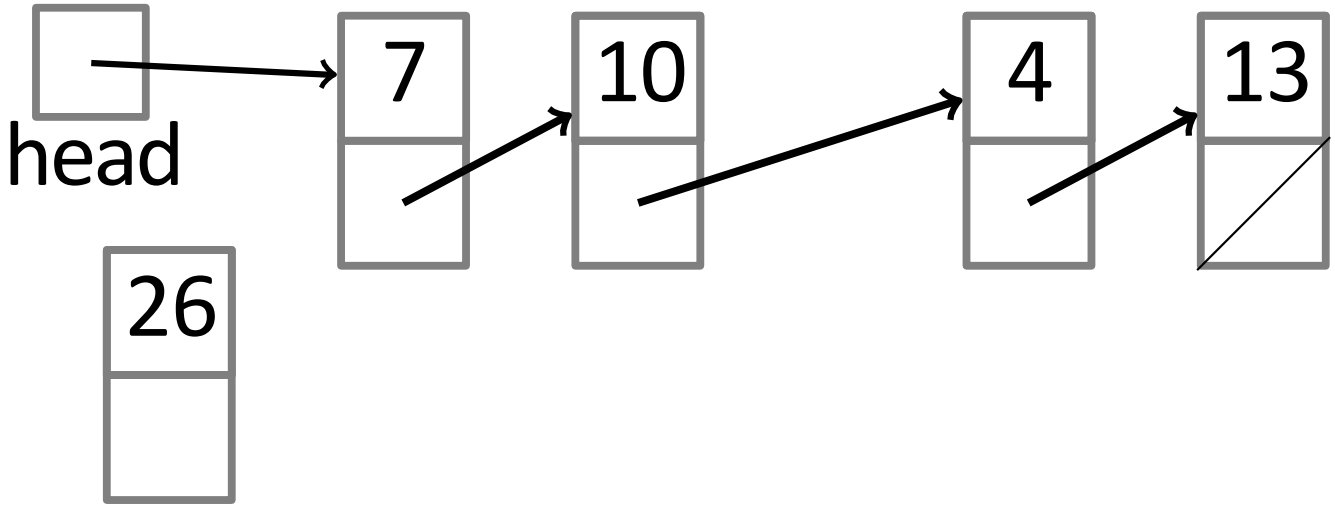
Times for Some Operations

PushFront



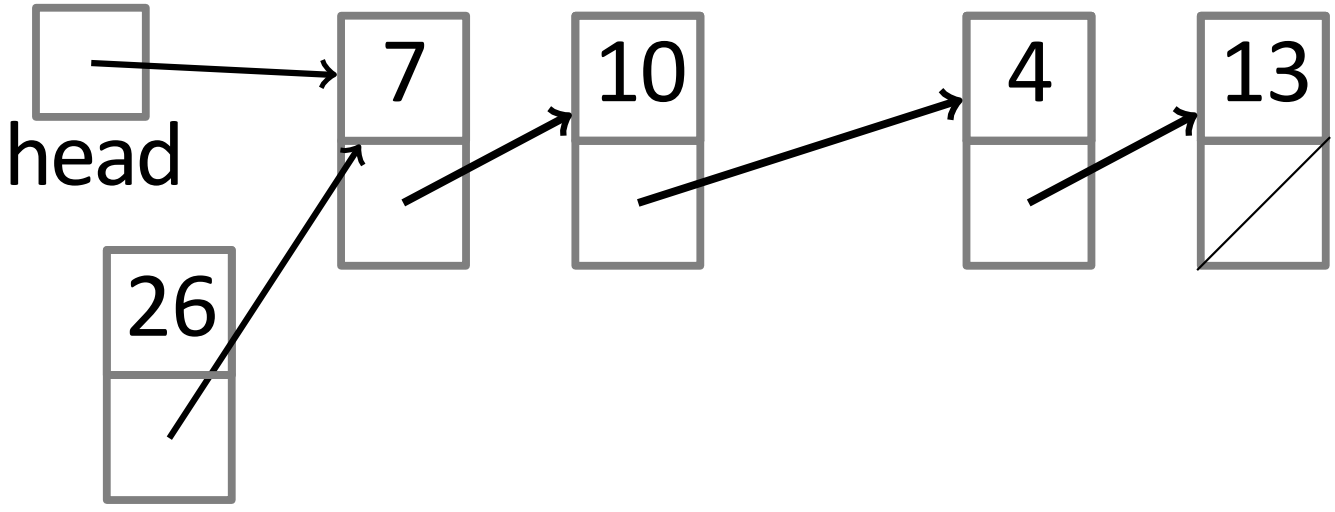
Times for Some Operations

PushFront



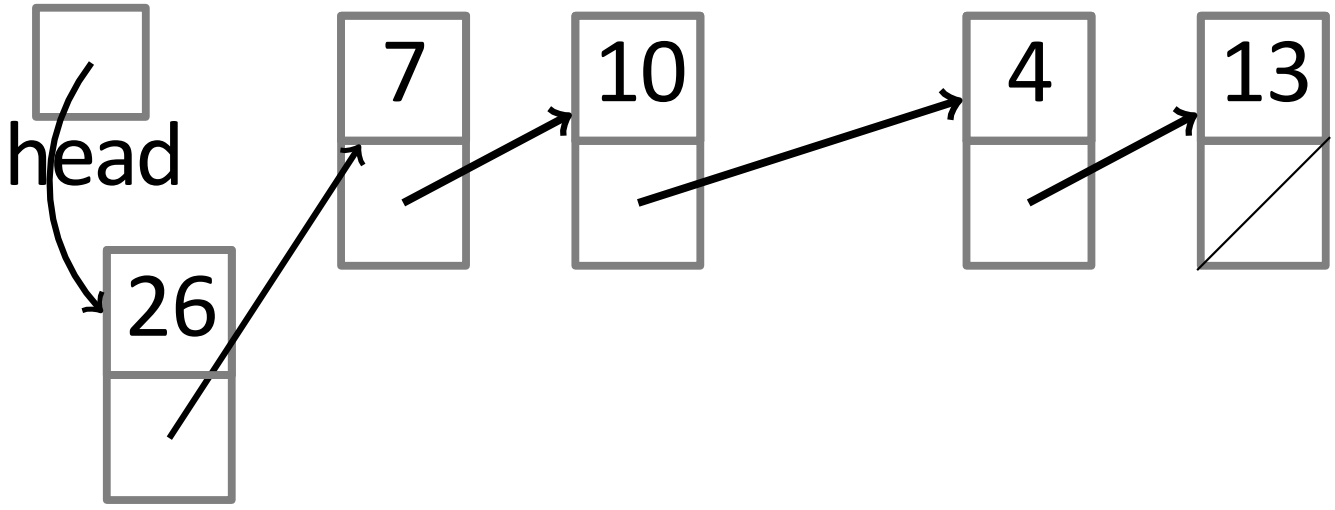
Times for Some Operations

PushFront



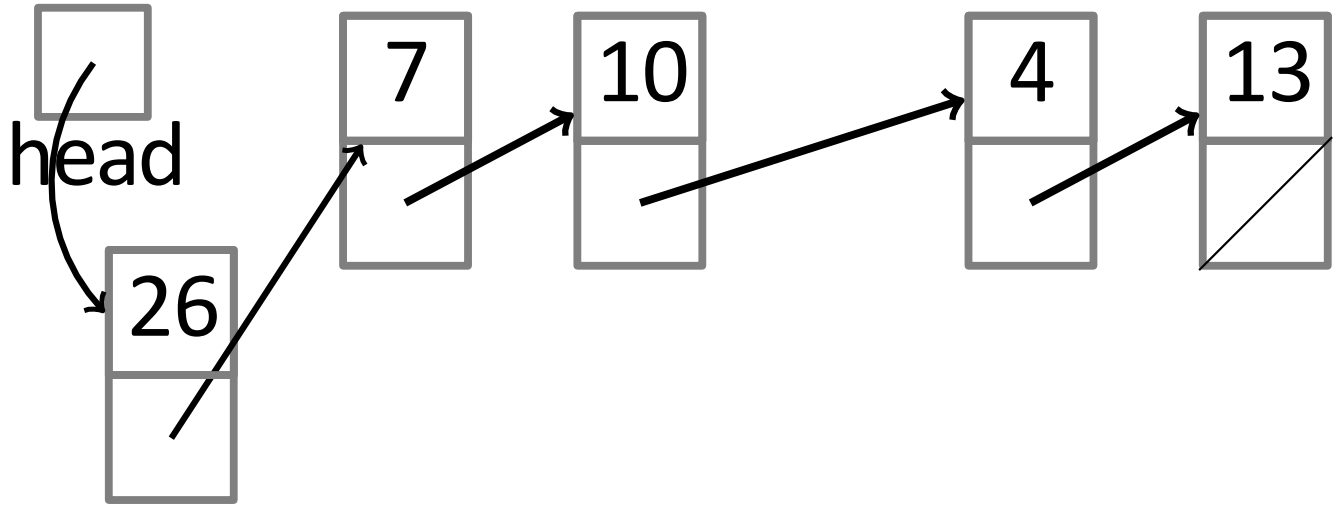
Times for Some Operations

PushFront $O(1)$



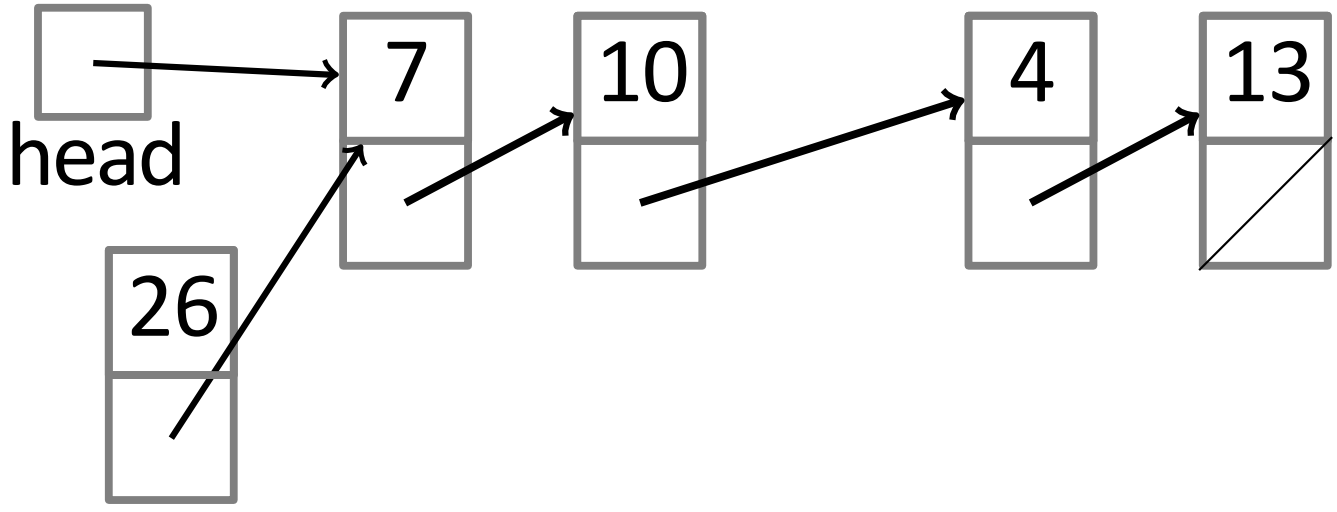
Times for Some Operations

PopFront



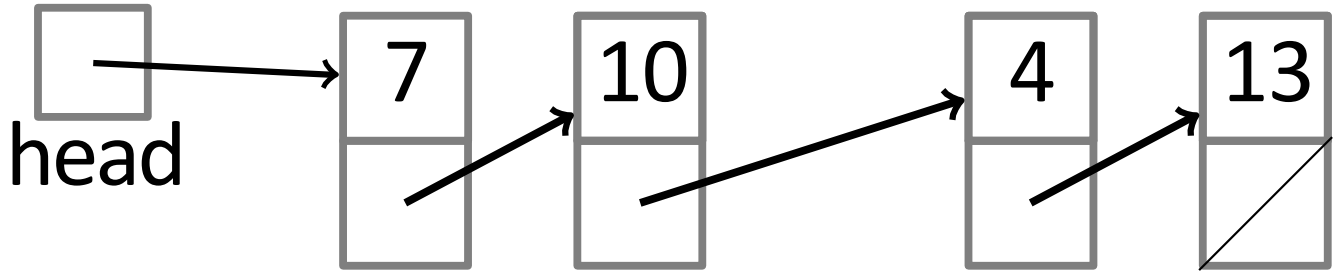
Times for Some Operations

PopFront



Times for Some Operations

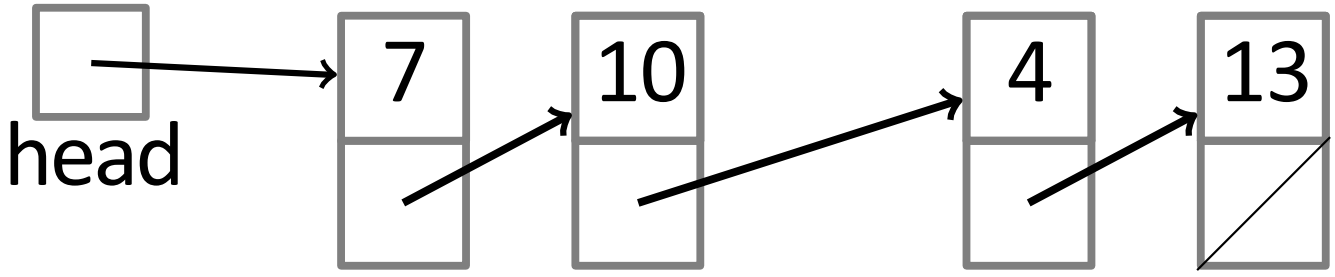
`PopFront` $O(1)$



Times for Some Operations

PushBack

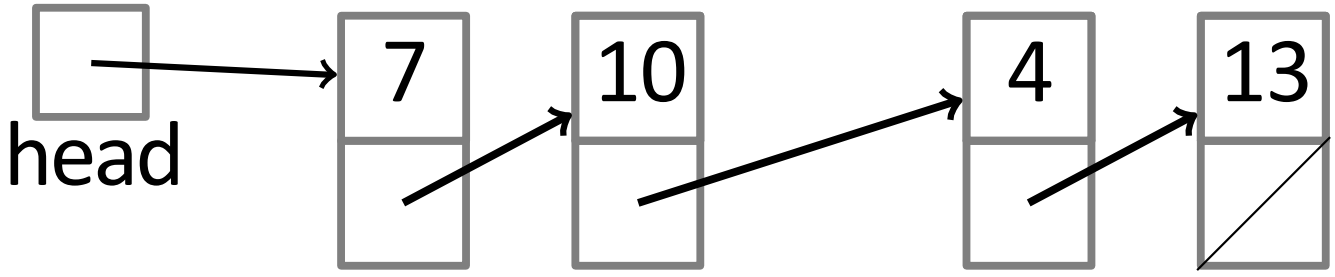
(no tail)



Times for Some Operations

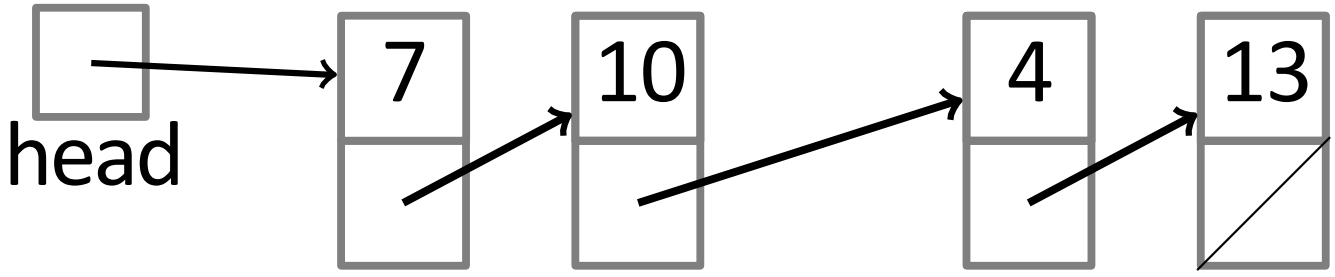
PushBack $O(n)$

(no tail)



Times for Some Operations

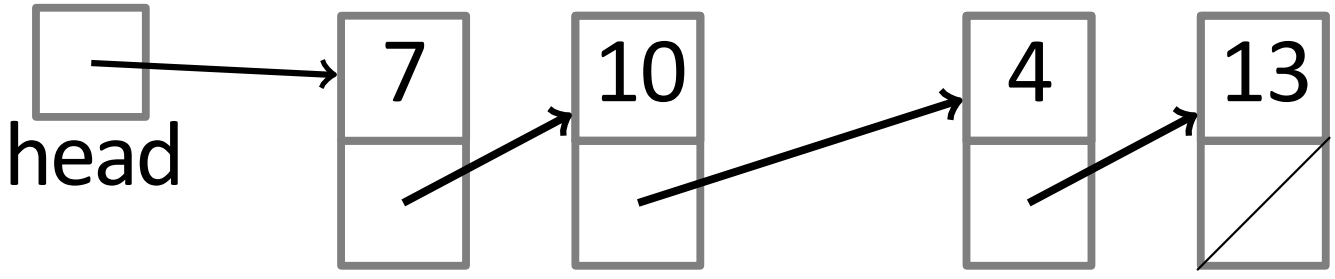
PopBack
(no tail)



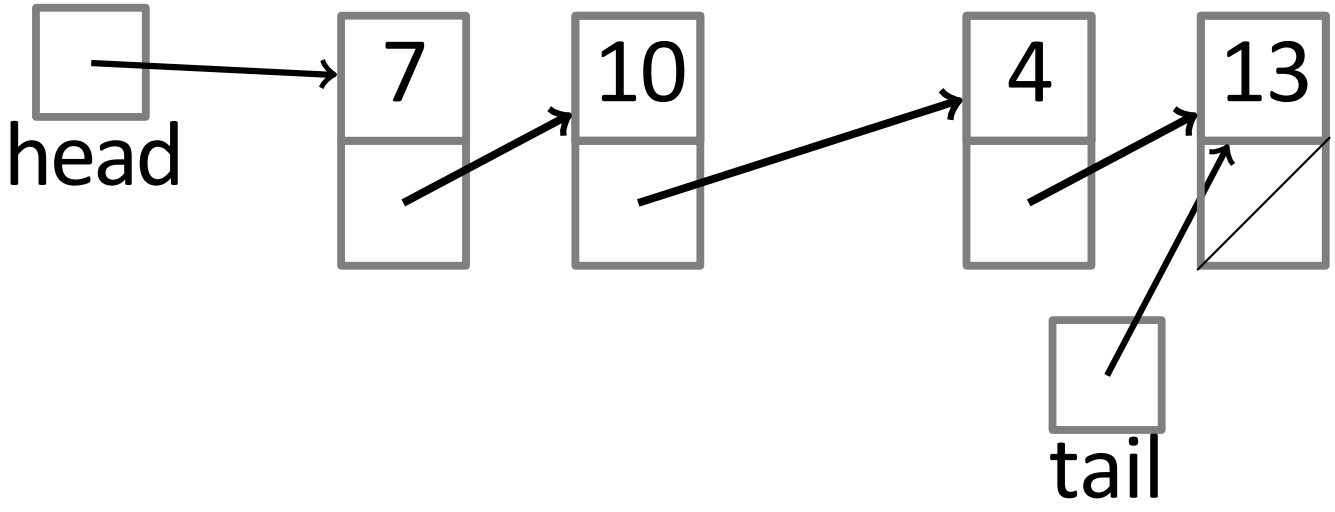
Times for Some Operations

PopBack $O(n)$

(no tail)



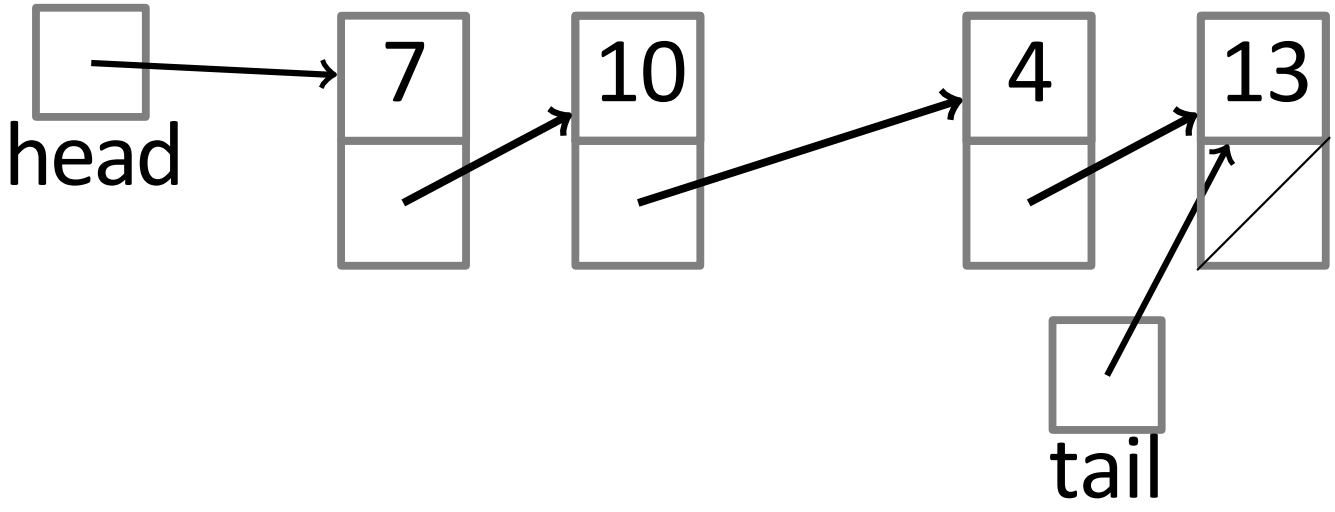
Times for Some Operations



Times for Some Operations

PushBack

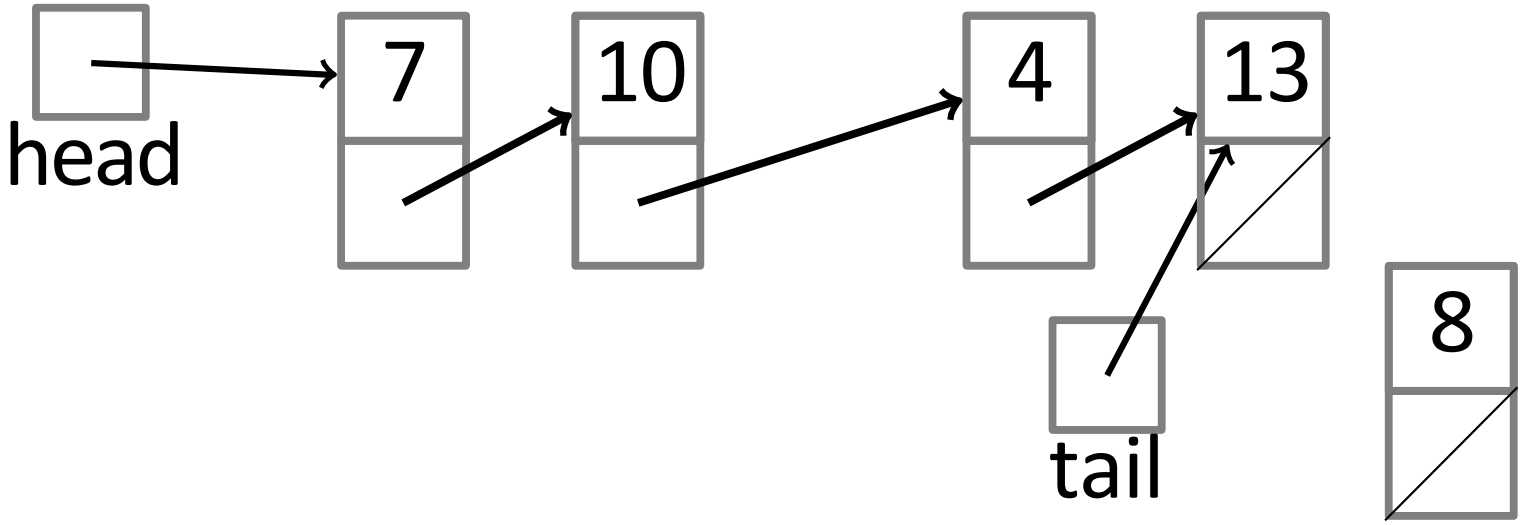
(with tail)



Times for Some Operations

PushBack

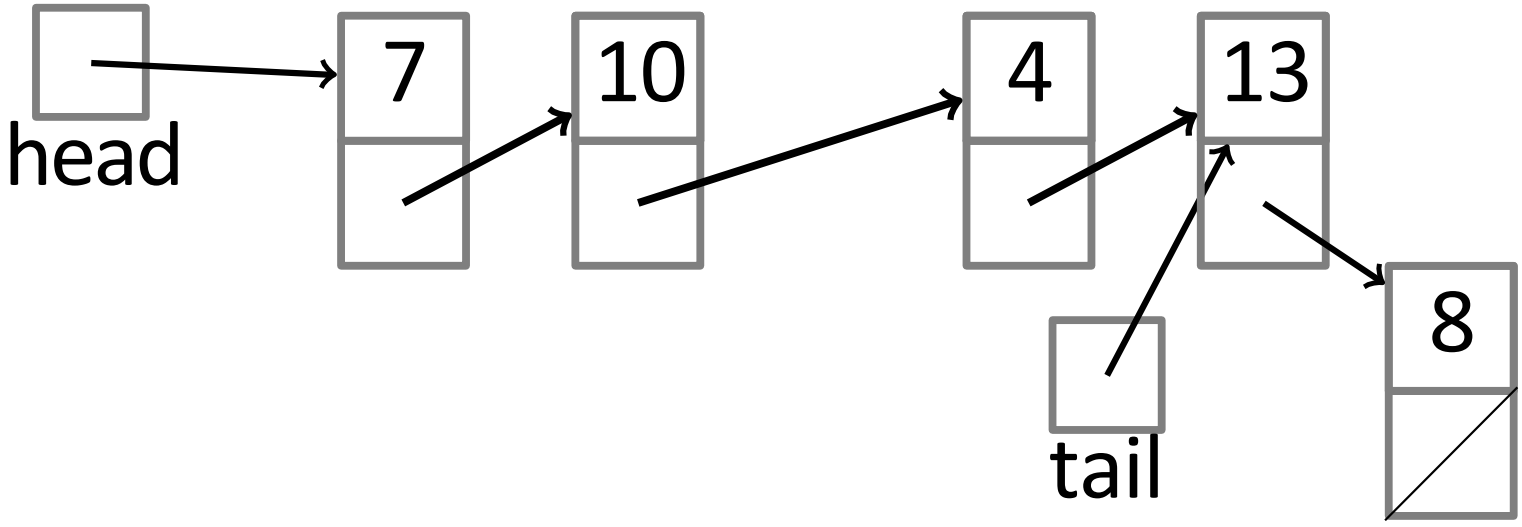
(with tail)



Times for Some Operations

PushBack

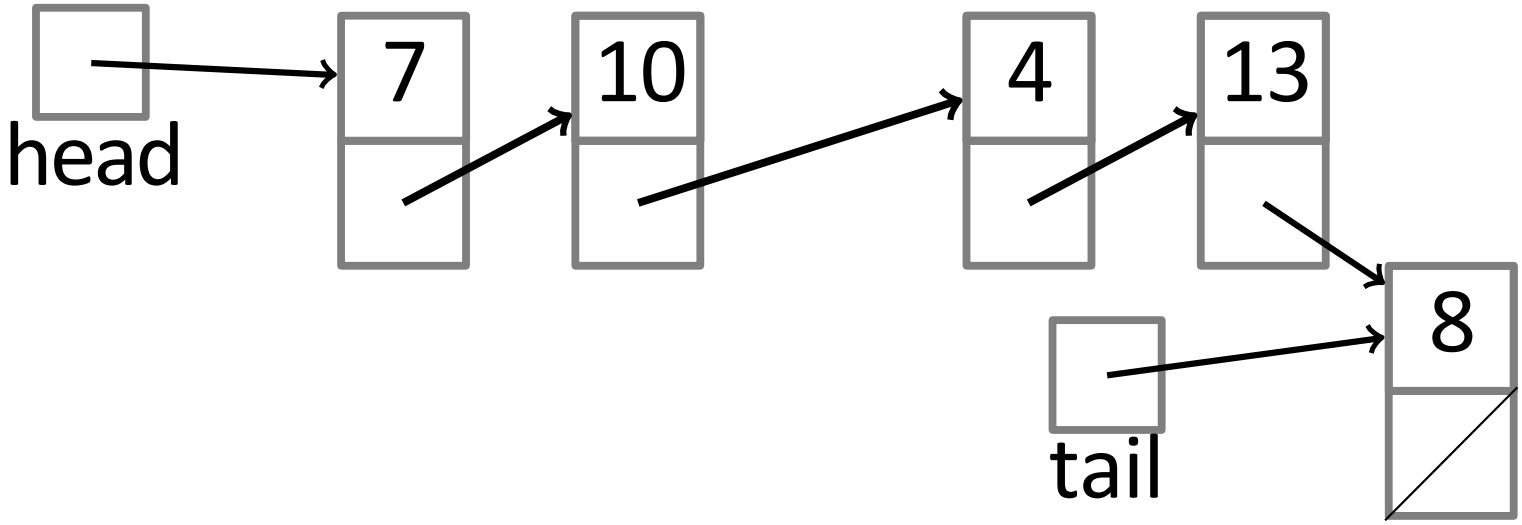
(with tail)



Times for Some Operations

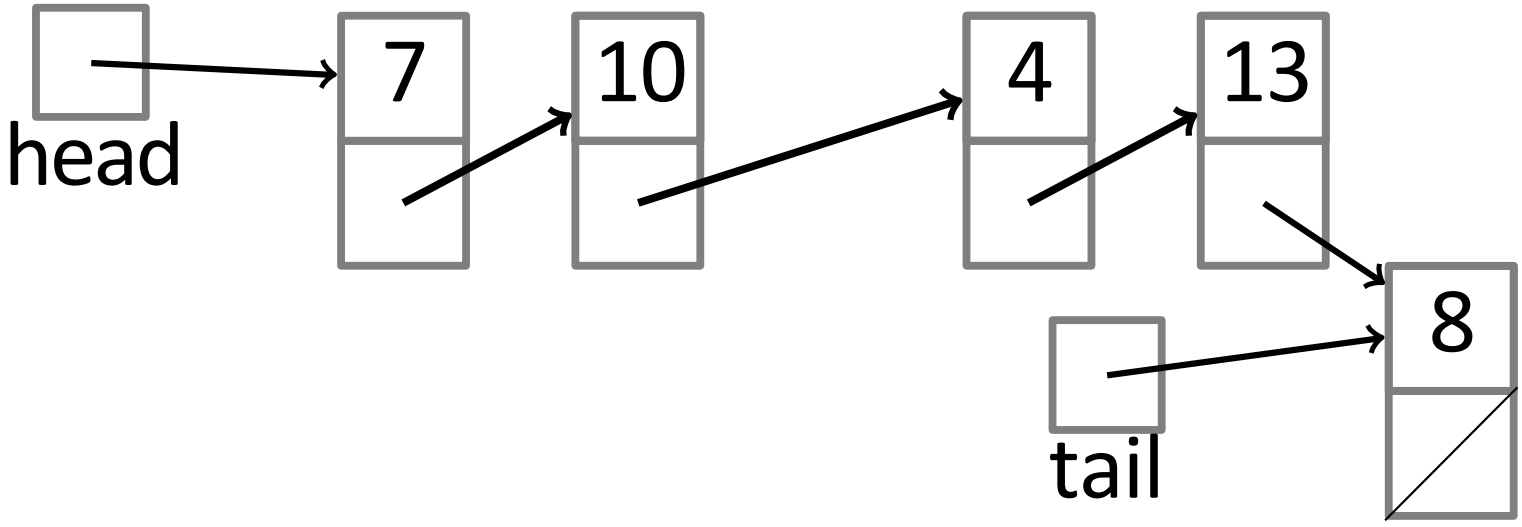
PushBack $O(1)$

(with tail)



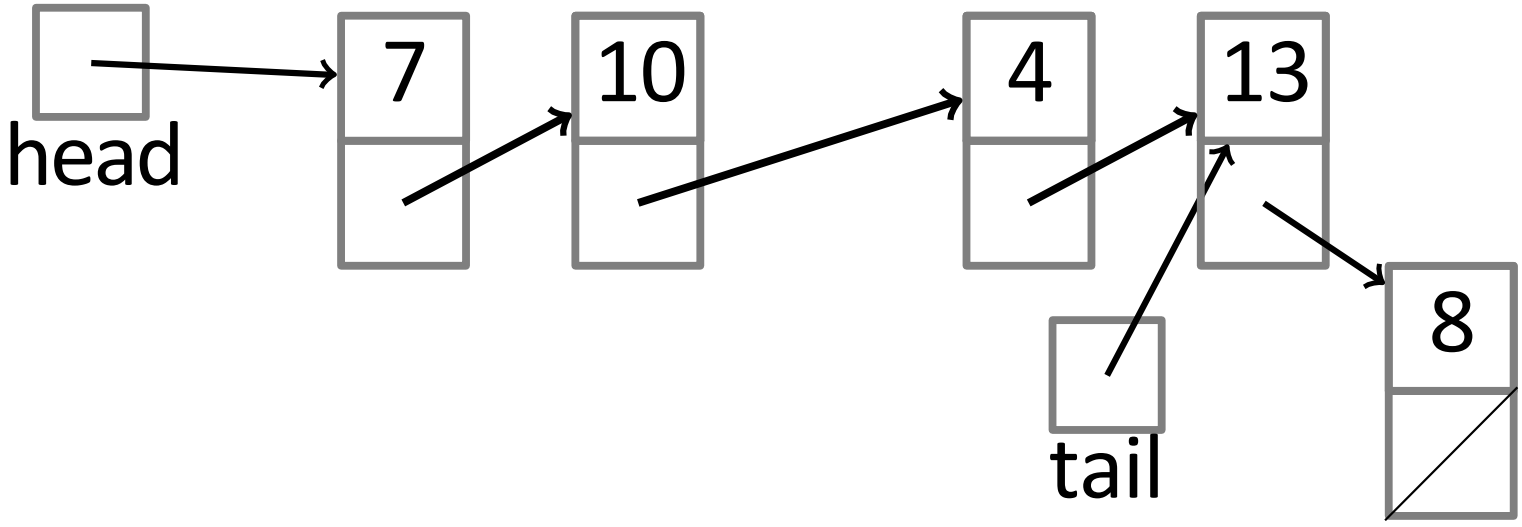
Times for Some Operations

PopBack
(with tail)



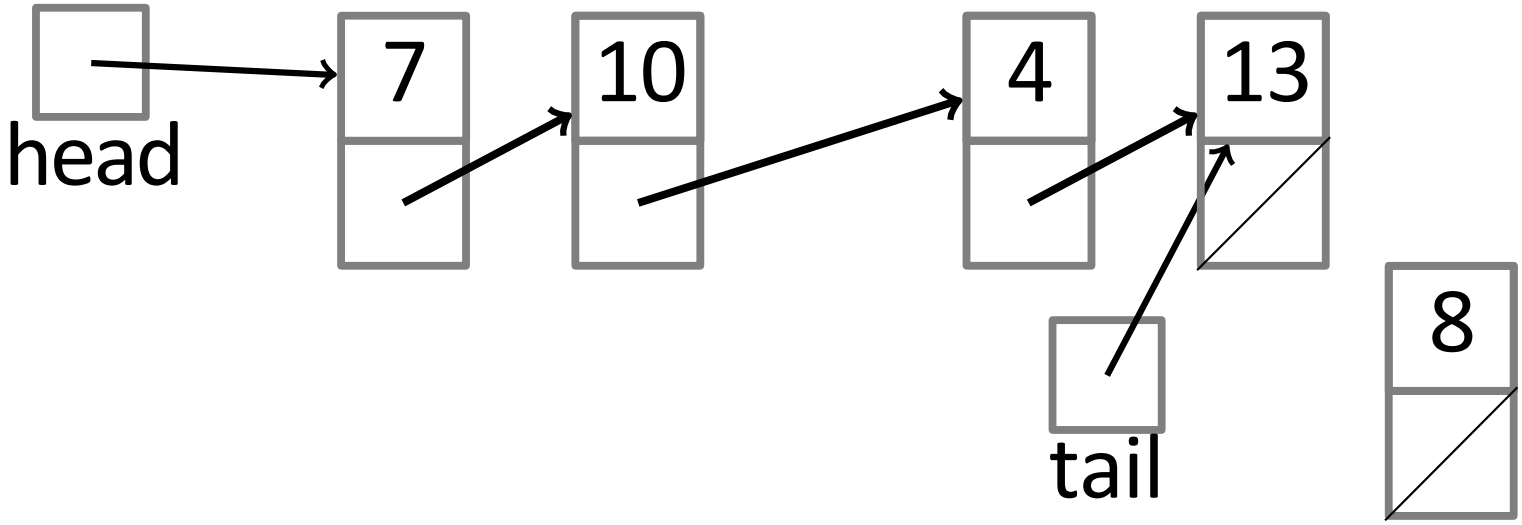
Times for Some Operations

PopBack
(with tail)



Times for Some Operations

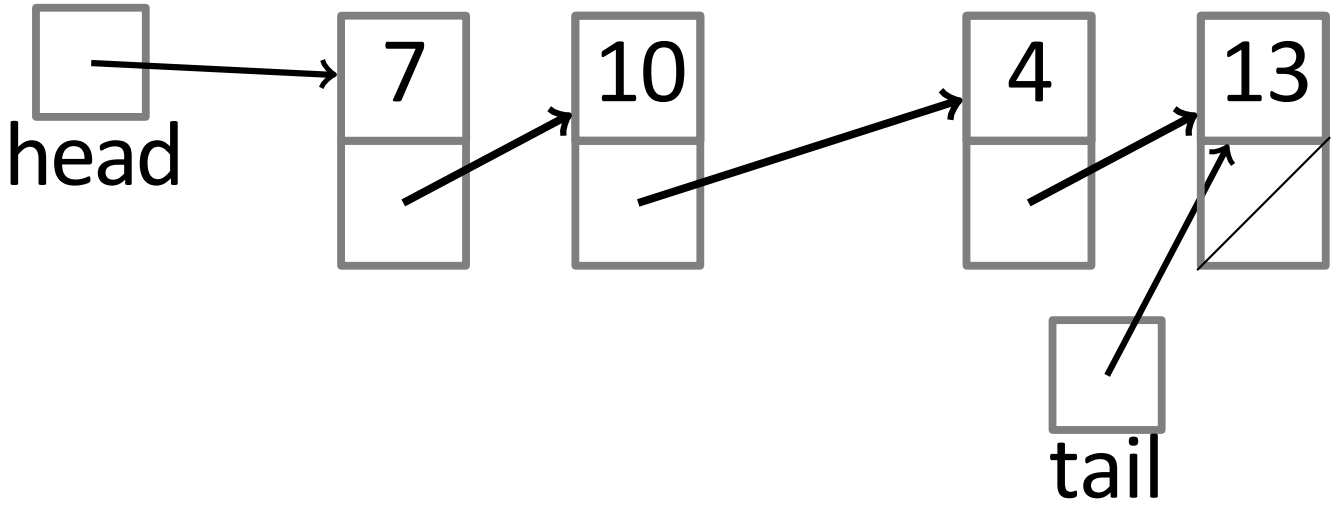
PopBack
(with tail)



Times for Some Operations

PopBack $O(n)$

(with tail)



Singly-linked List

PushFront(*key*)

node ←new node

node.key ← *key*

node.next ← *head*

head ← *node*

if *tail* = nil:

tail ← *head*

```
void insert_start(int value)
{
        node *temp=new node;
        temp->data=value;
        temp->next=head;
        head=temp;
}
```

Singly-linked List

PopFront()

```
if head = nil:
   ERROR: empty list
head ← head.next
if head = nil:
   tail ← nil
```

```
void delete_first()
{
        node *temp=new node;
        temp=head;
        head=head->next;
        delete temp;
}
```

Singly-linked List

PushBack(*key*)

node ←new node

node.key ← *key*

node.next =nil

Singly-linked List

PushBack(*key*)

node ←new node

node.key ← *key*

node.next =nil

if *tail* = nil:

    head ← *tail* ← *node*

Singly-linked List

PushBack(*key*)

```
node ←new node
node.key ← key
node.next =nil
if tail = nil:
    head ← tail ← node
else:
    tail.next ← node
    tail ← node
```

Singly-linked List

PopBack()

Singly-linked List

PopBack()

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

Singly-linked List

PopBack()

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

Singly-linked List

PopBack()

```
if head = nil:   ERROR: empty list
if head = tail :
   head ← tail ←nil
else:
   p ← head
   while p.next.next≠ nil:
      p ← p.next
```

Singly-linked List

PopBack()

```
if head = nil:   ERROR: empty list
if head = tail :
   head ← tail ←nil
else:
   p ← head
   while p.next.next≠ nil:
      p ← p.next
   p.next ← nil;  tail ← p
```

Singly-linked List

AddAfter(*node, key*)

node2 ←new node

node2.key ← *key*

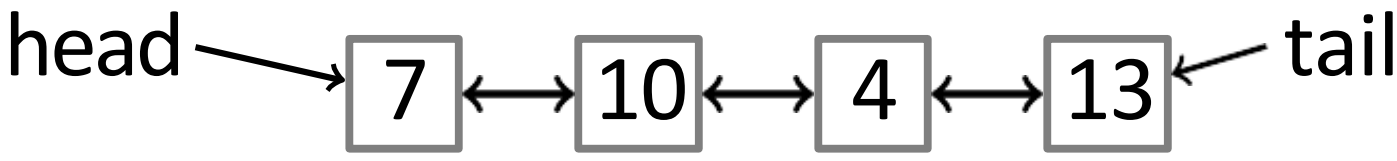
node2.next = *node.next*

node.next = *node2*

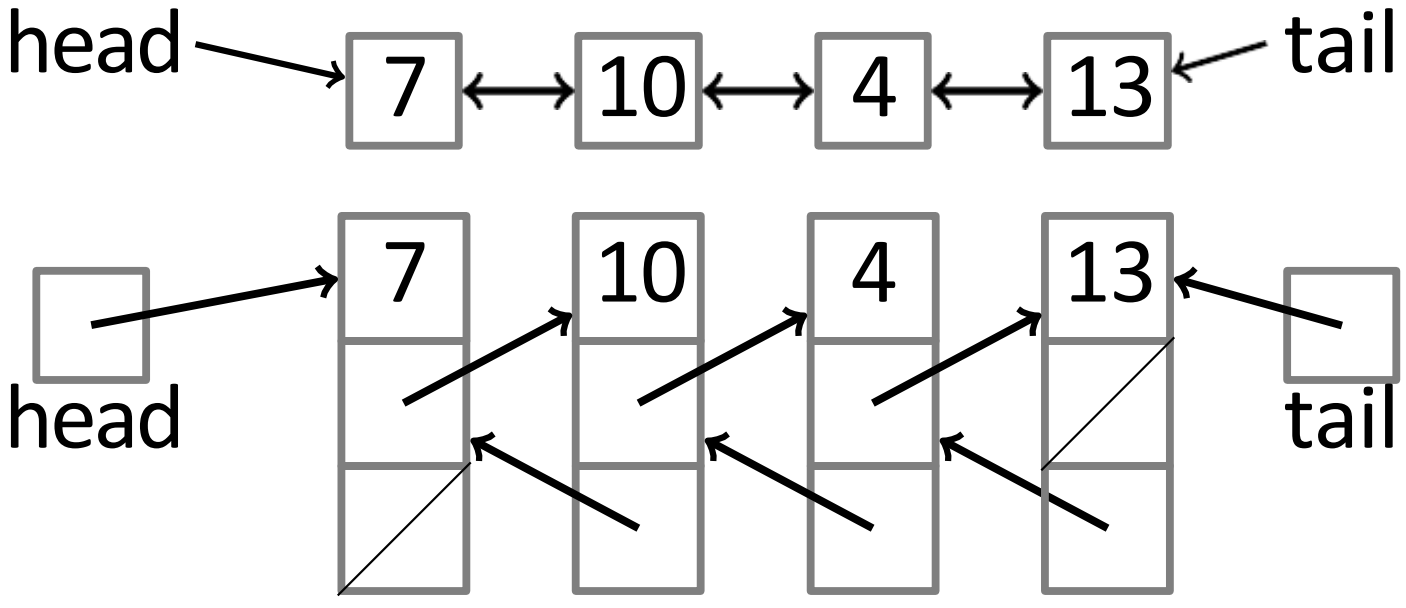
if *tail* = *node*:

    tail ← *node2*

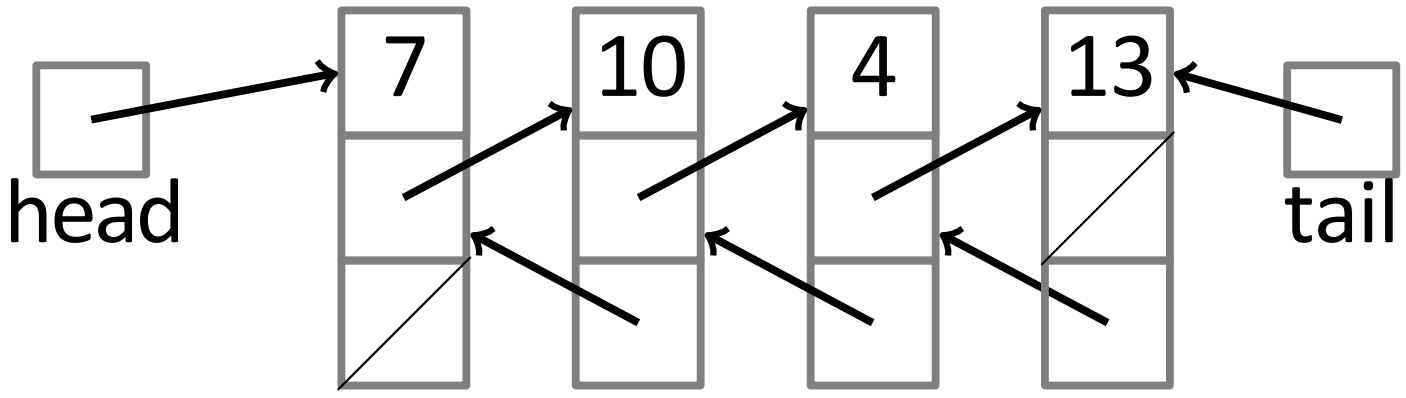
Doubly-Linked List



Doubly-Linked List



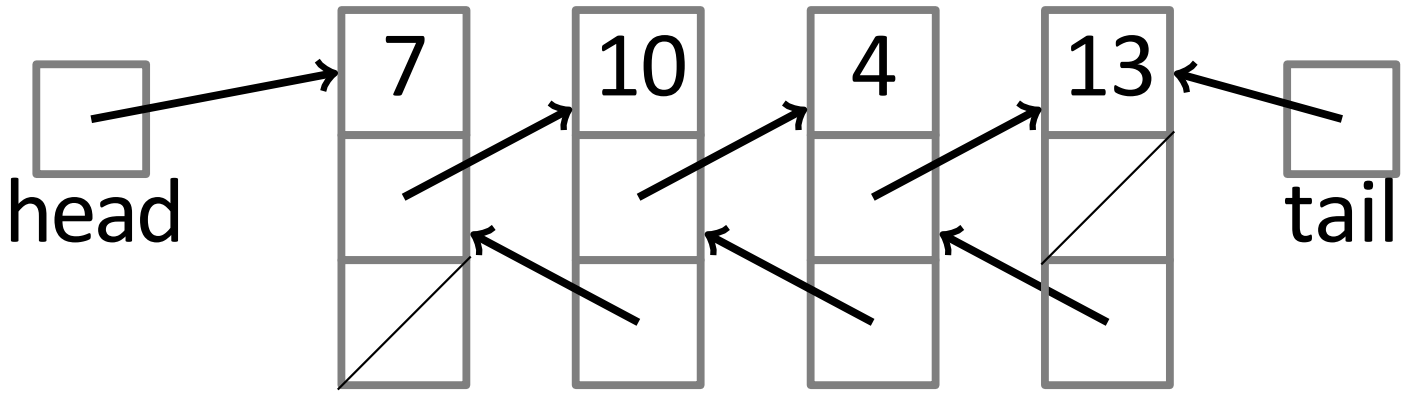
Doubly-Linked List



Node contains:

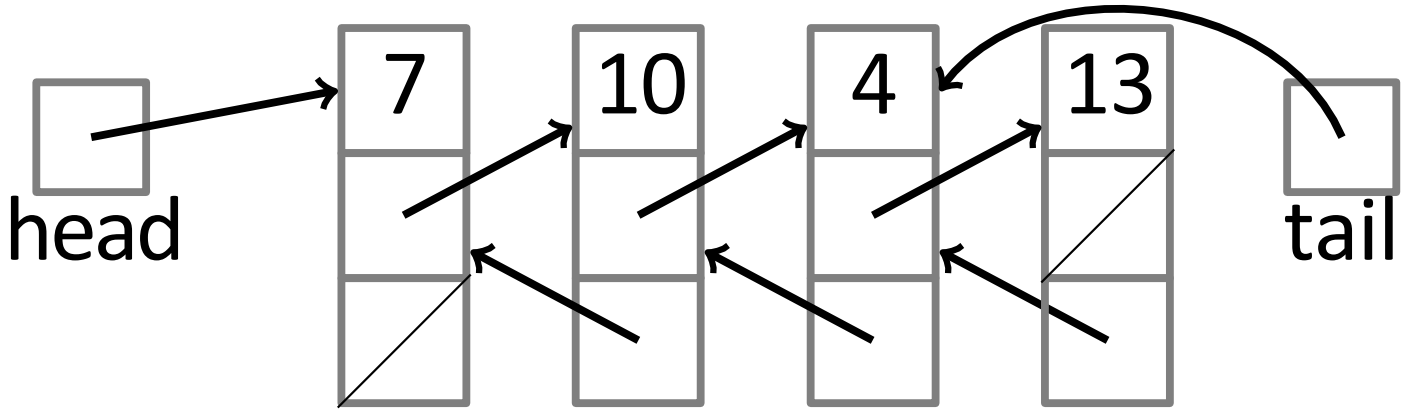
- `key`
- `next`
- pointer

Doubly-Linked List



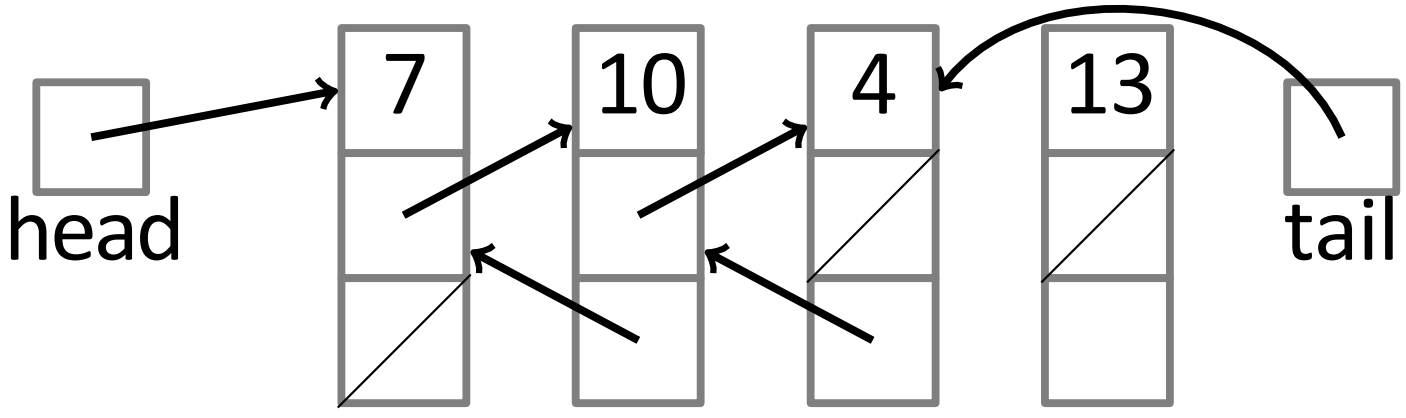
PopBack

Doubly-Linked List



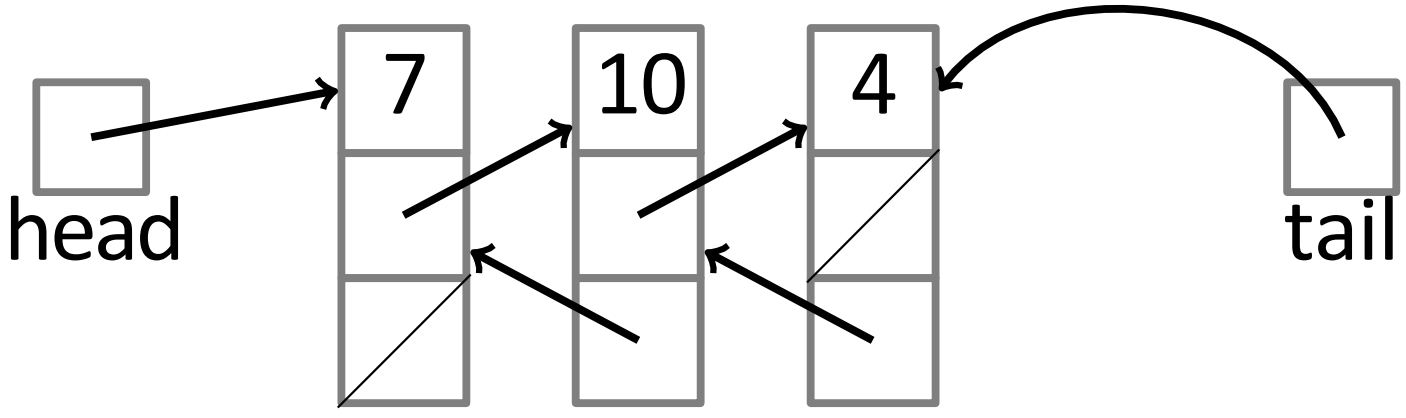
PopBack

Doubly-Linked List



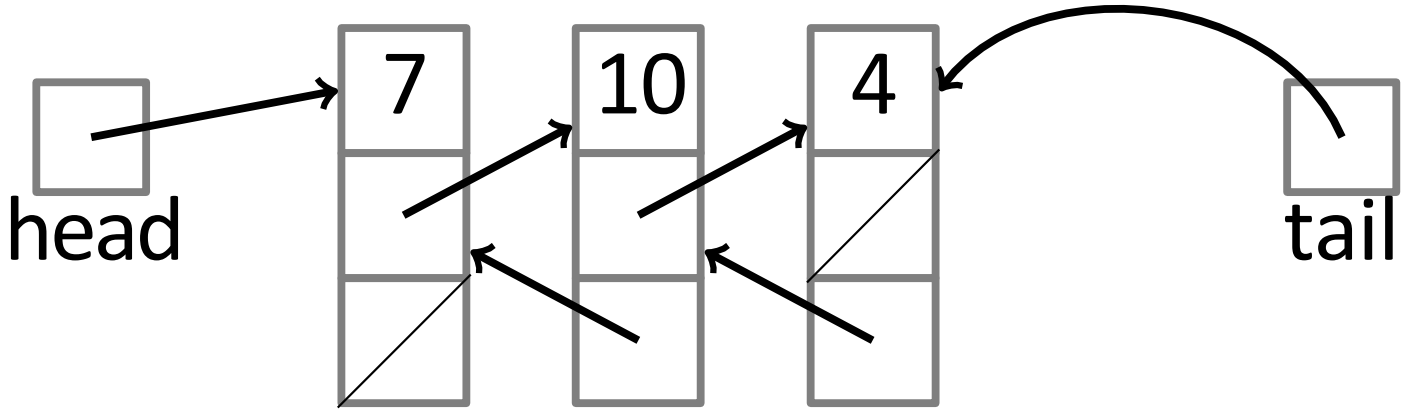
PopBack

Doubly-Linked List



PopBack

Doubly-Linked List



PopBack $O(1)$

Doubly-linked List

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:

    head ← *tail* ← *node*

    node.prev ←nil

Doubly-linked List

PushBack(*key*)

```
node ←new node
node.key ← key ; node.next =nil
if tail = nil:
    head ← tail ← node
    node.prev ←nil
else:
    tail .next ← node
    node.prev ← tail
    tail ← node
```

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 ← tail ←nil
```

Doubly-linked List

PopBack()

```
if head = nil:   ERROR: empty list
if head = tail :
   head ← tail ←nil
else:
   tail ← tail .prev
   tail .next ←nil
```

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
```

Doubly-linked List

AddBefore(*node, key*)

```
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:
    head ← node2
```