

Test & Marking Guide
Available.

Are you doing the quizzes?

Lecture 25

Linked Queues

FIT 1008
Introduction to Computer Science



COMMONWEALTH OF AUSTRALIA
Copyright Regulations 1969
WARNING

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Where are we at?

- Know about Linked Structures
- Have implemented Nodes
- Have implemented Linked Stacks

Objectives for these this lecture

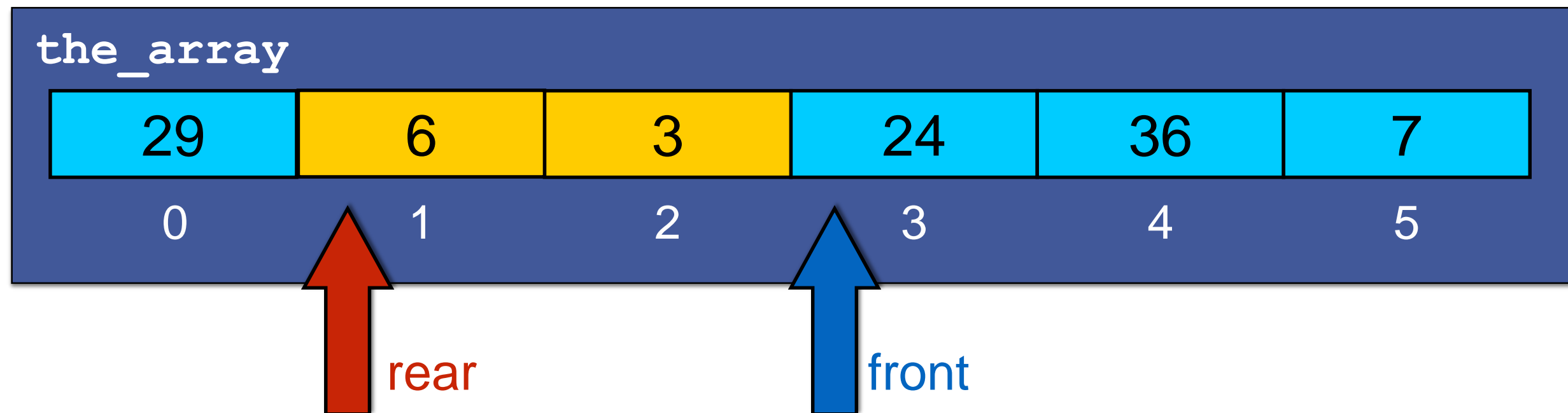
- To understand:
 - The concept of linked data structures
 - Their use in implementing queues
- To be able to:
 - Implement, use and modify linked queues.
 - Decide when it is appropriate to use them (rather than arrays)



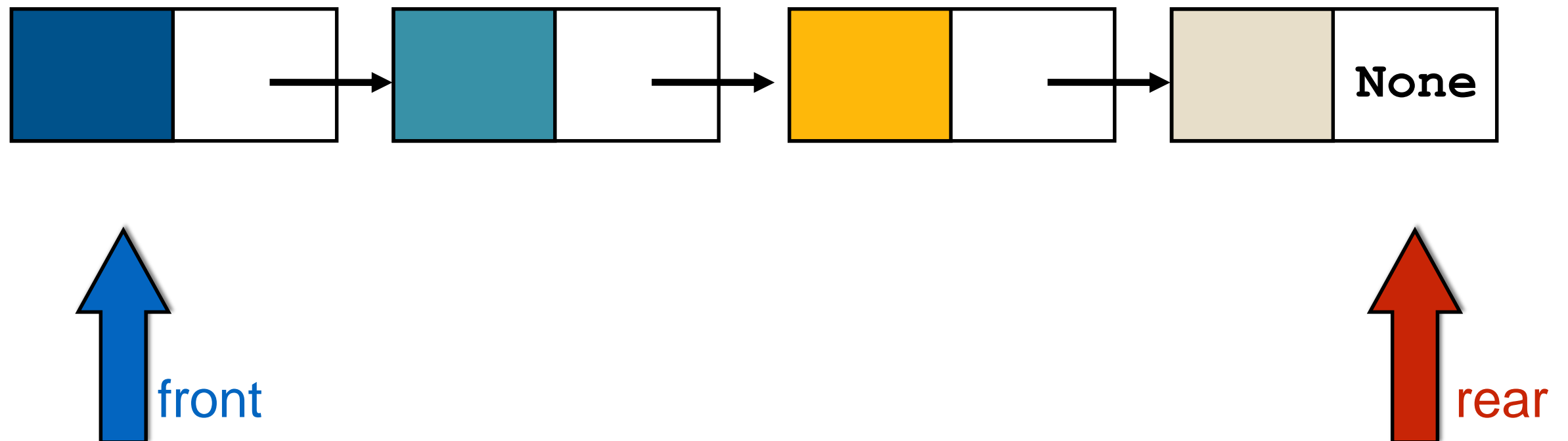
“Form an orderly Q to the left..”

Remember array-based queues?

front: 3 rear: 1 count: 4



Linked Queue



Important: Rear now designates the last node

No need for circularity.

count is optional...


```
from node import Node
```

No need for size when
initialising the object

```
class Queue:
```

```
    def __init__(self):  
        self.front = None  
        self.rear = None
```

The class must ensure
that when self.front is
None, self.rear is also
None.

```
    def is_empty(self):  
        return self.front is None
```

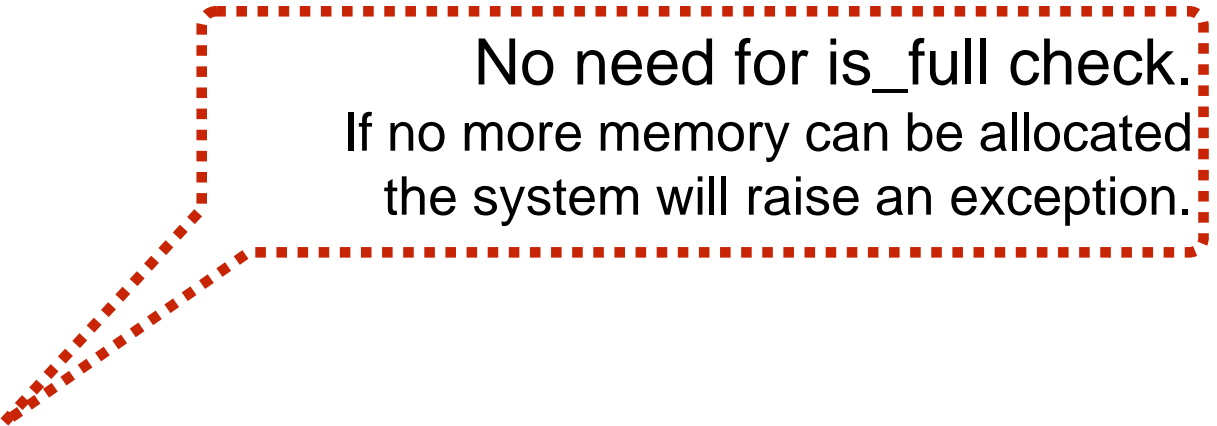
```
    def is_full(self):  
        return False
```

```
    def reset(self):  
        self.front = None  
        self.rear = None
```

Append: algorithm

Circular array implementation:

- If the array is full raise exception
- Else
 - Increase rear % length of the array
 - Add the item at the position designated by rear

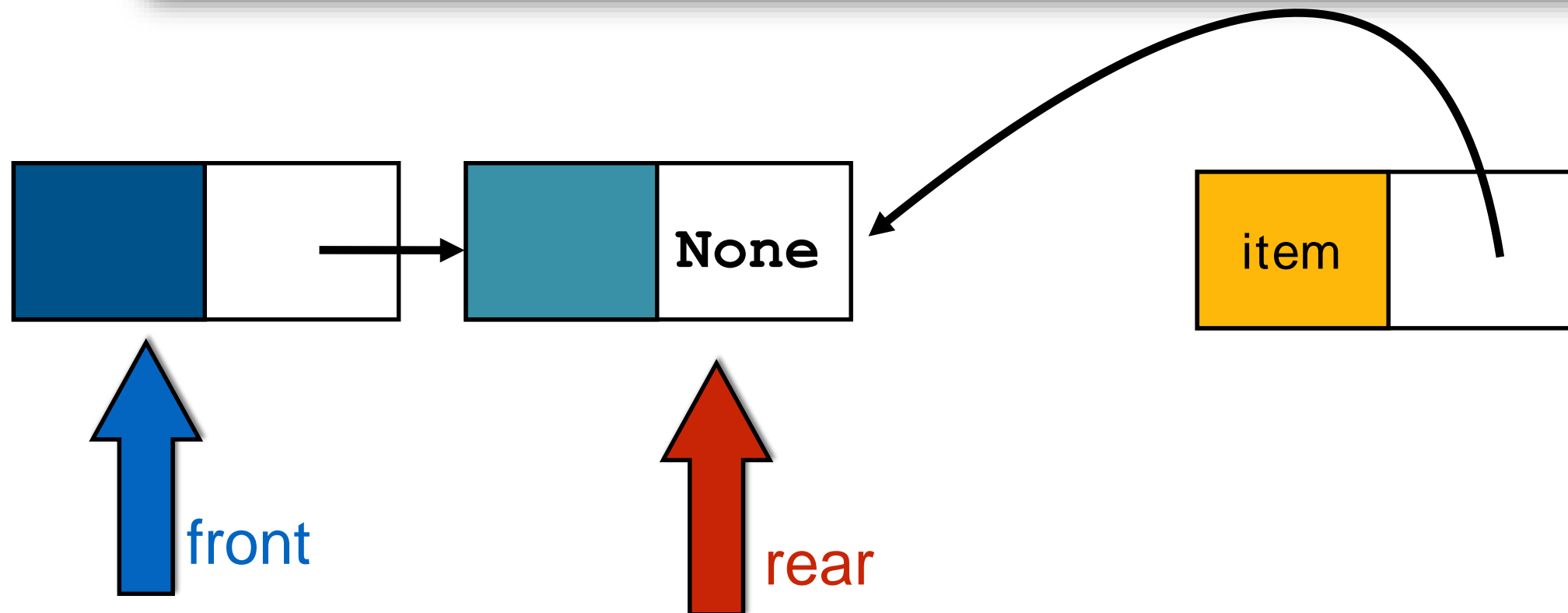


No need for is_full check.
If no more memory can be allocated
the system will raise an exception.

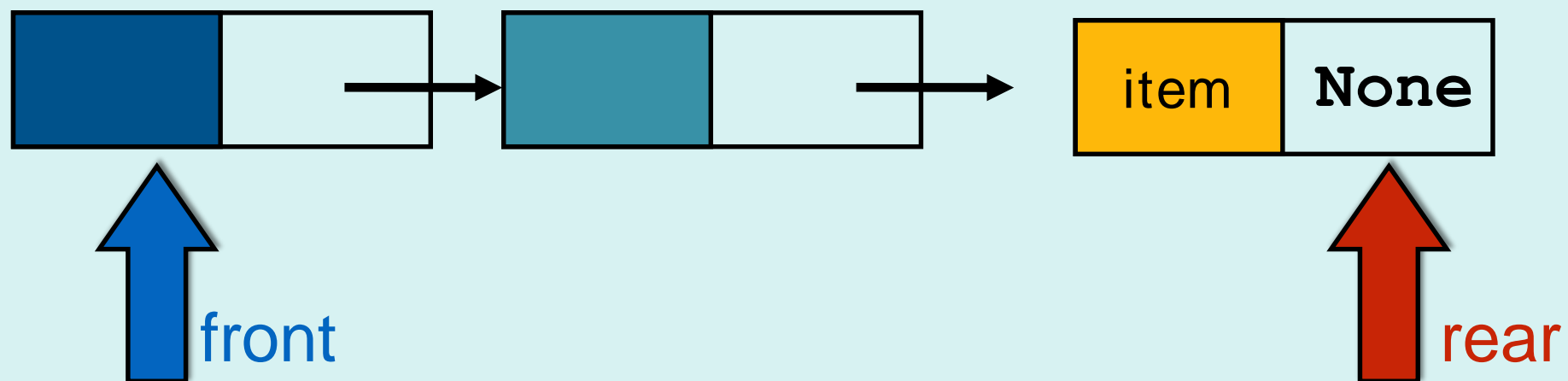
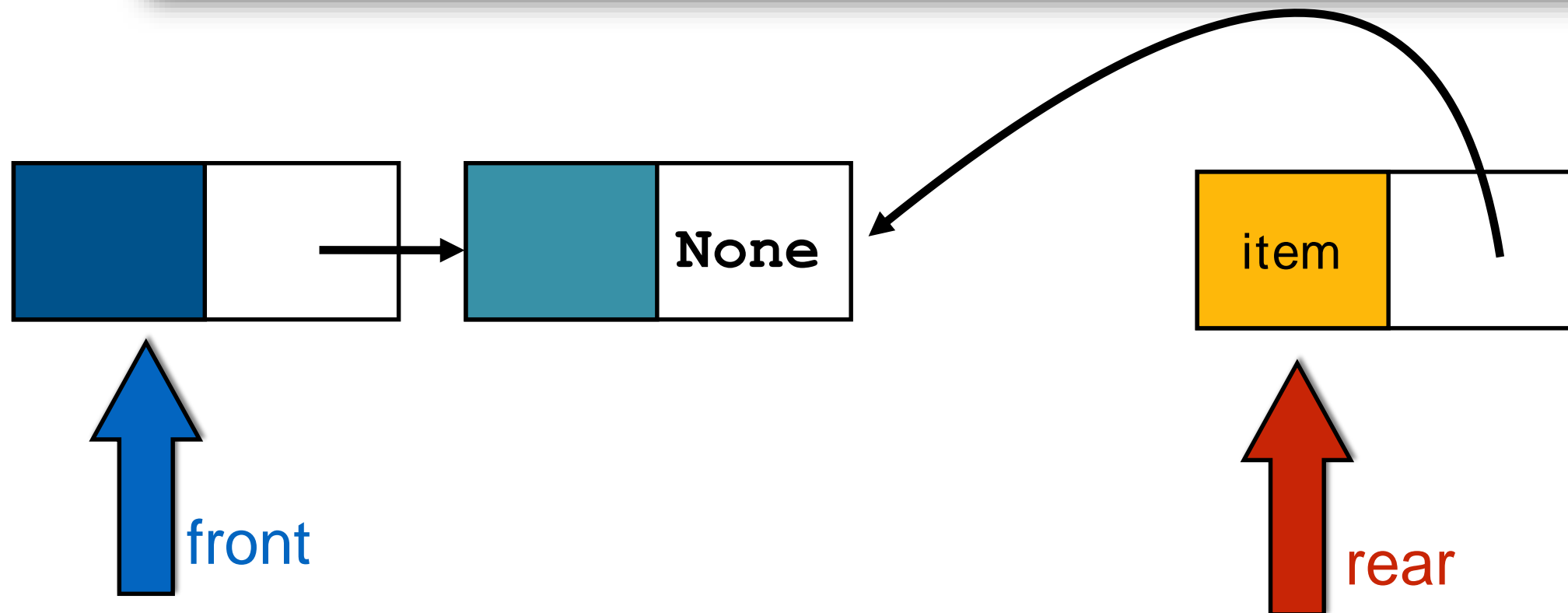
Linked implementation:

- Create a new node that contains item and points to None
- Link the current rear to it
- Change rear to point to new node.

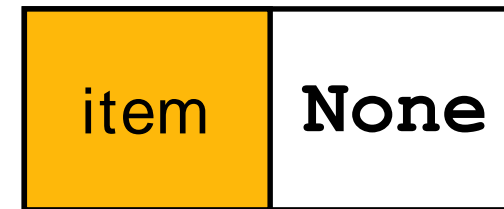
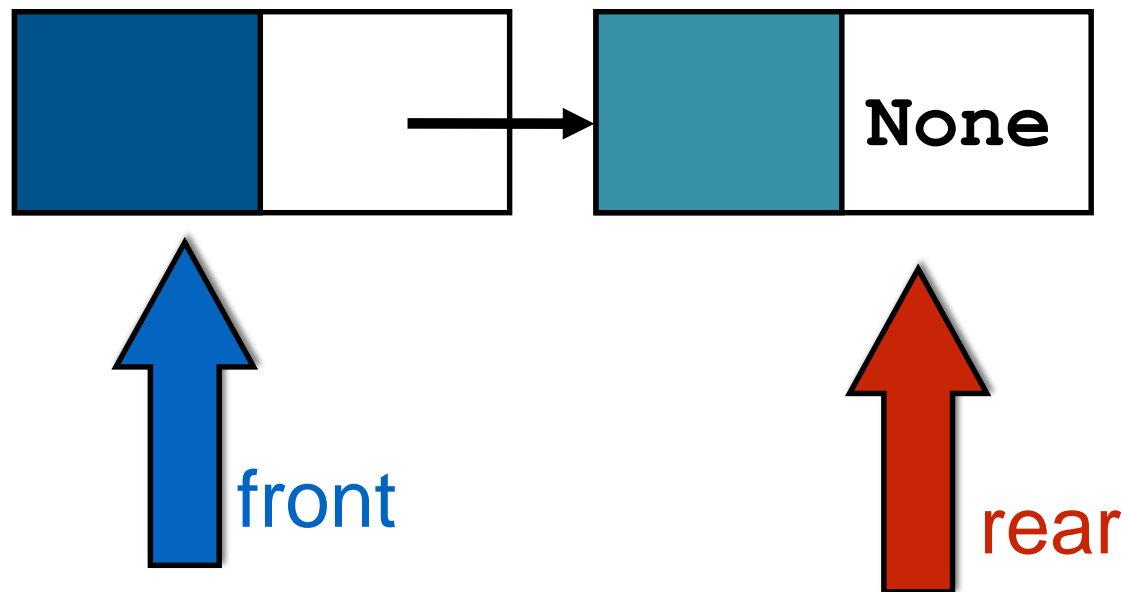
```
def append(self, item):  
    self.rear = Node(item, self.rear)
```



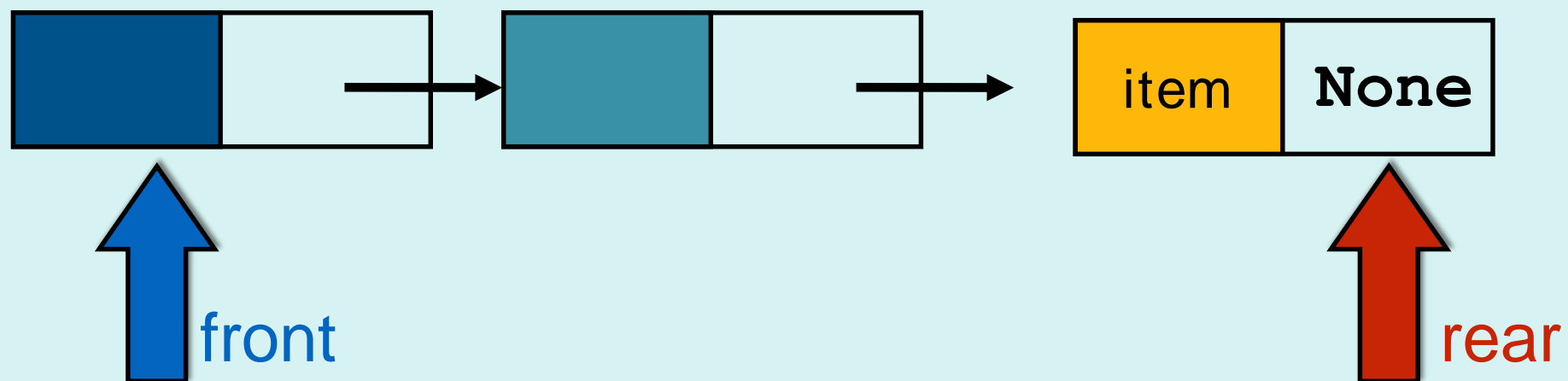
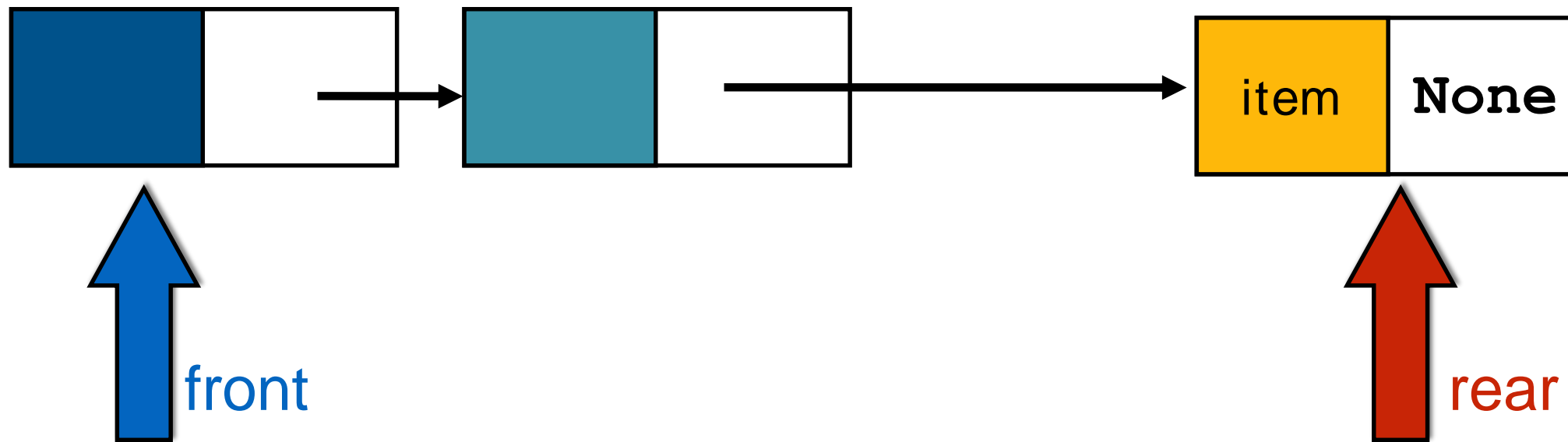
```
def append(self, item):  
    self.rear = Node(item, self.rear)
```



```
def append(self, item):  
    self.rear.next = Node(item, None)  
    self.rear = self.rear.next
```

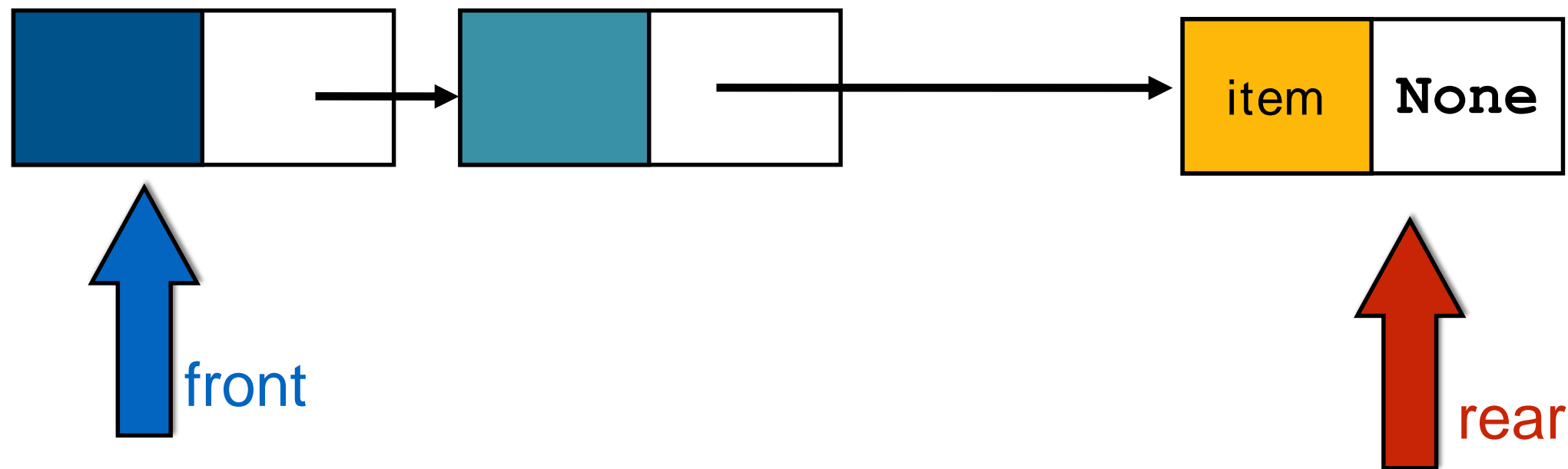


```
def append(self, item):  
    self.rear.next = Node(item, None)  
    self.rear = self.rear.next
```

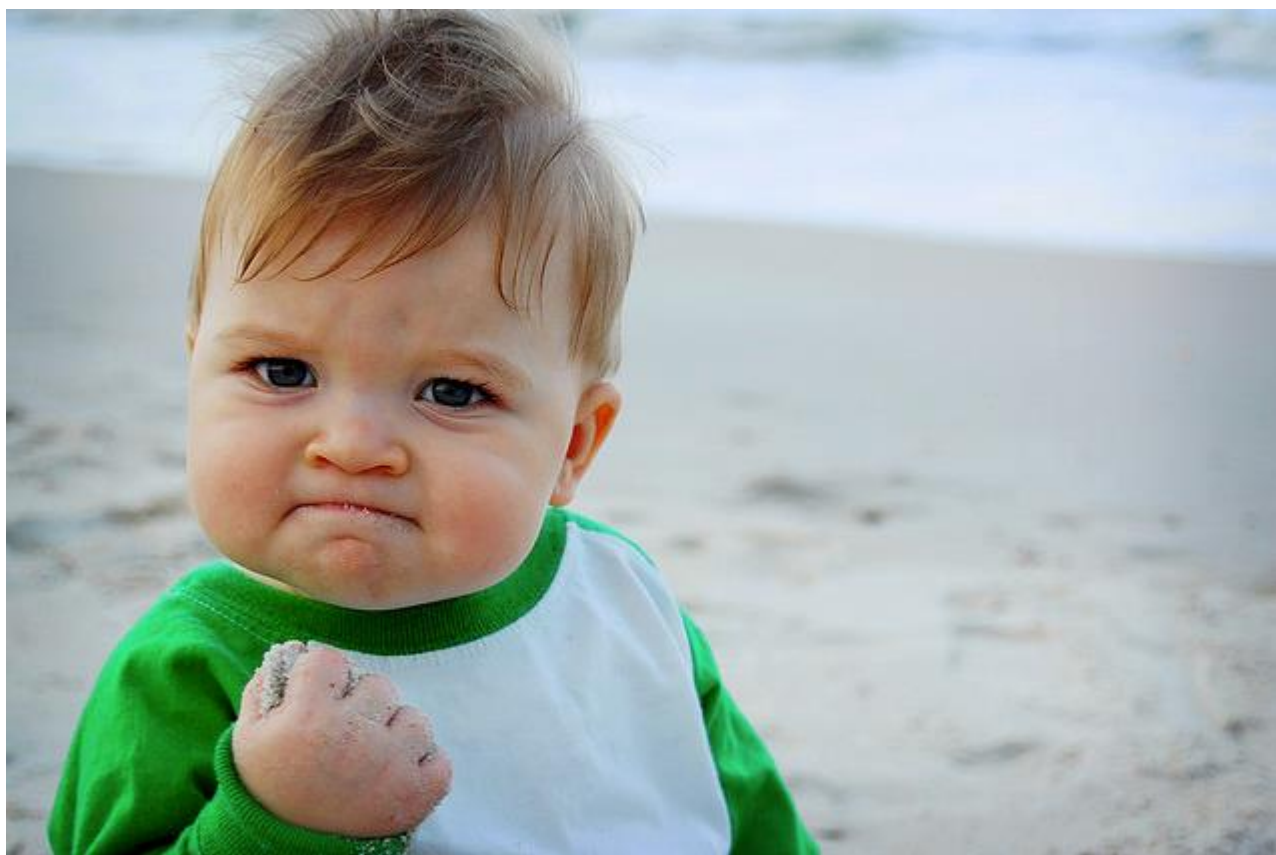


algorithm.

```
def append(self, item):  
    self.rear.next = Node(item, None)  
    self.rear = self.rear.next
```

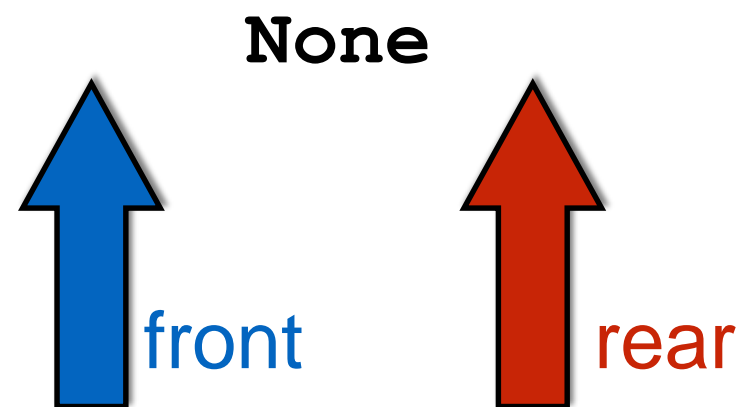


- Create a new node for item
- Make a link from current rear to new node
- The new node becomes the new rear

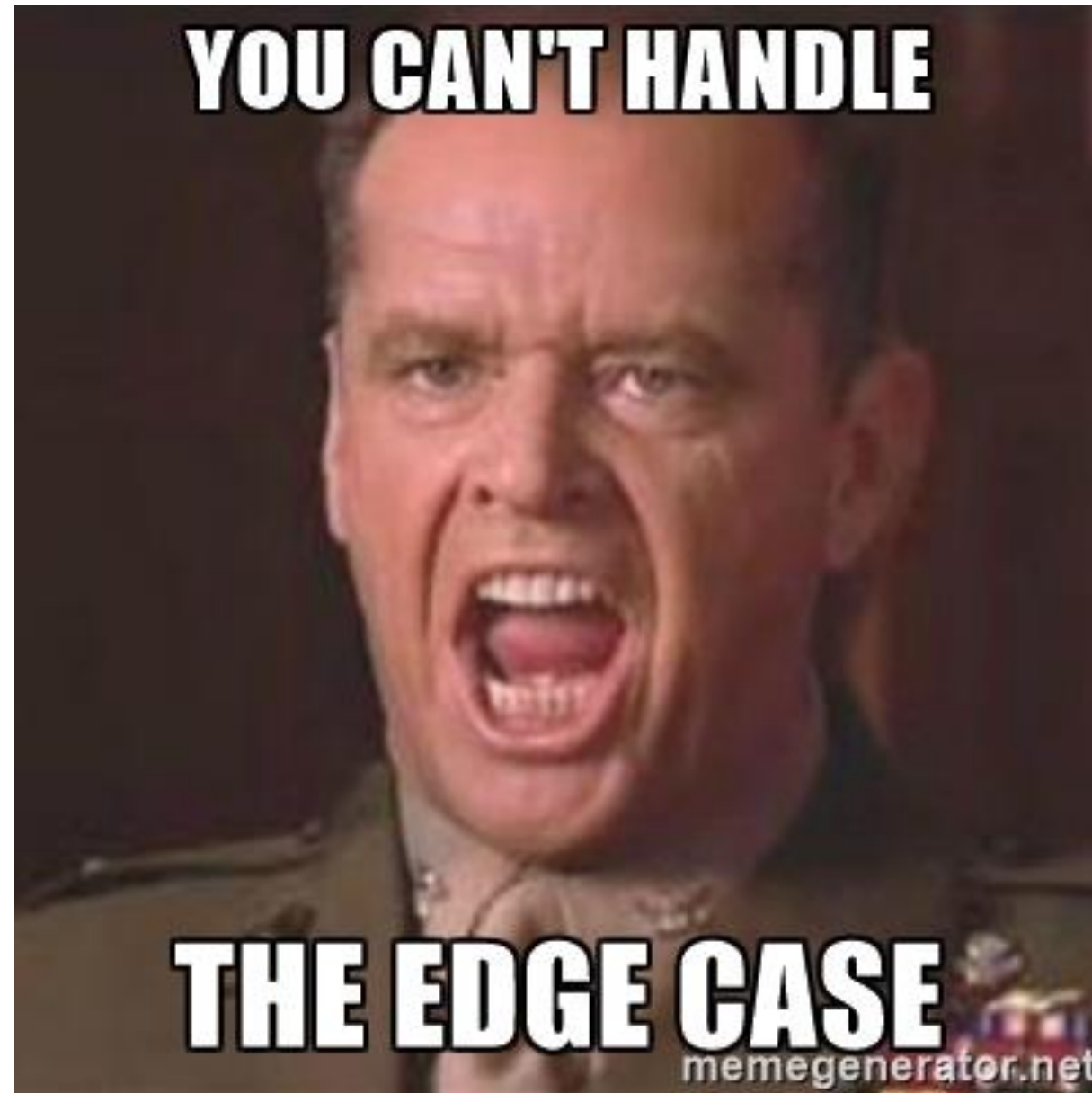


Boundary cases

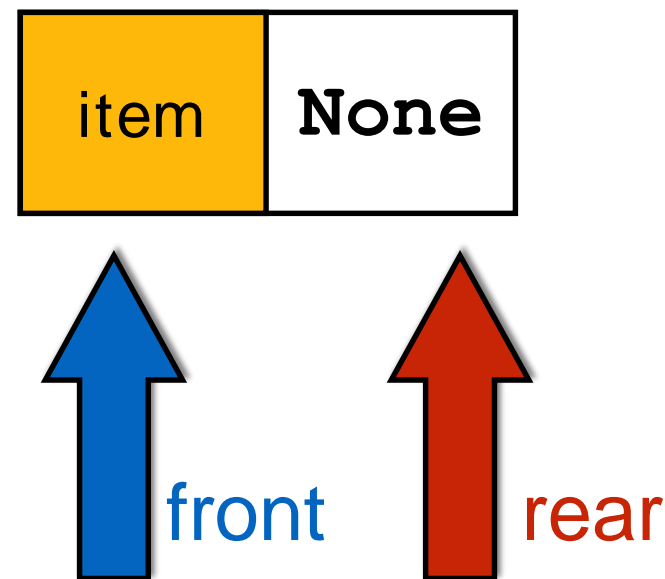
```
def append(self, item):  
    self.rear.next = Node(item, None)  
    self.rear = self.rear.next
```



If the queue is empty
we need to do something with front

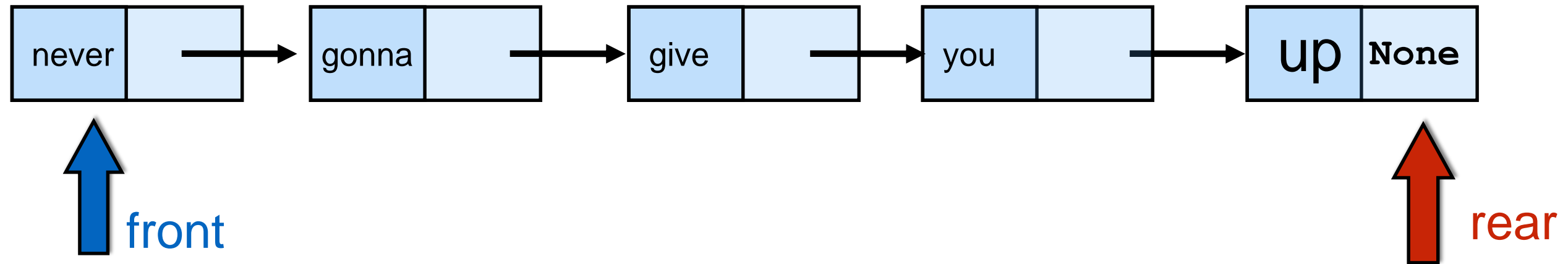


(with that code)



- Create a new node for item
- If the queue is empty:
 - Make the new node be the front
- If the queue is not empty:
 - Make a link from current rear to new node
- The new node becomes the new rear

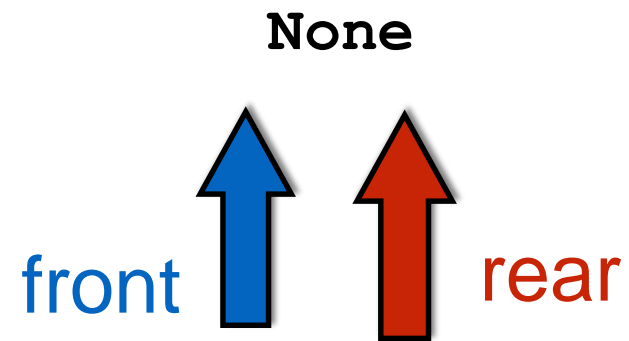
```
def append(self, item):  
    new_node = Node(item, None)  
    if self.is_empty():  
        self.front = new_node  
    else:  
        self.rear.next = new_node  
    self.rear = new_node
```



```
q.front.item = "never"  
q.rear.item = "you"
```

```
q.append("up")
```

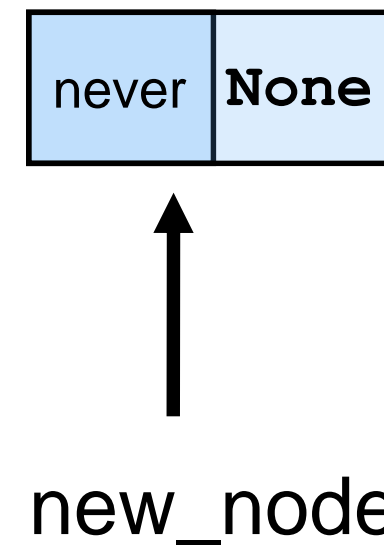
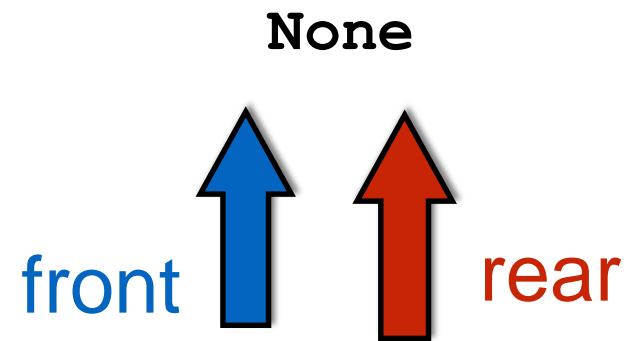
```
def append(self, item):  
    new_node = Node(item, None)  
    if self.is_empty():  
        self.front = new_node  
    else:  
        self.rear.next = new_node  
    self.rear = new_node
```



```
q.front = None  
q.rear = None
```

```
q.append("never")
```

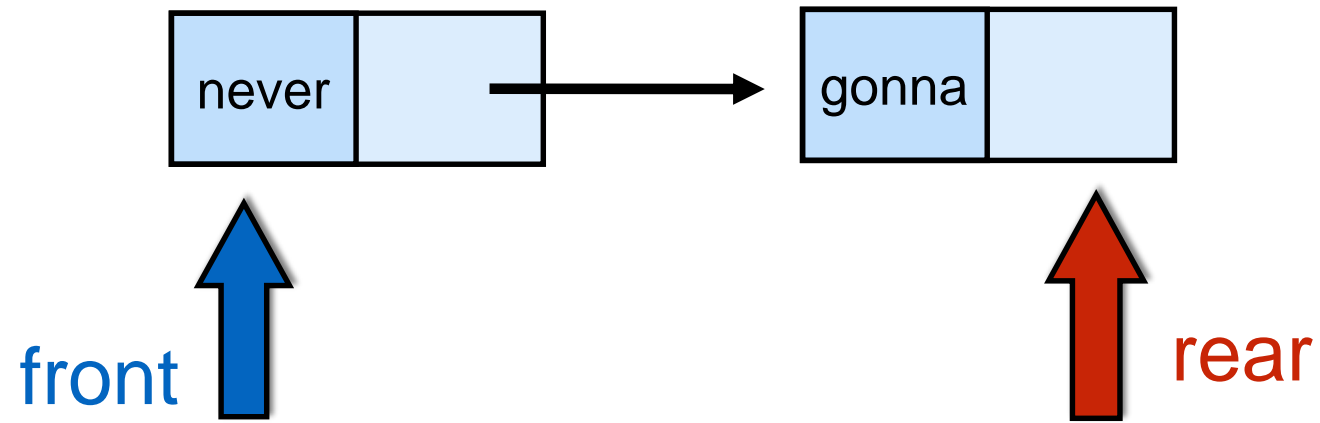
```
def append(self, item):  
    new_node = Node(item, None)  
    if self.is_empty():  
        self.front = new_node  
    else:  
        self.rear.next = new_node  
    self.rear = new_node
```

```
q.front = None  
q.rear = None
```

```
q.append("never")
```

```
def append(self, item):  
    new_node = Node(item, None)  
    if self.is_empty():  
        self.front = new_node  
    else:  
        self.rear.next = new_node  
    self.rear = new_node
```



```
if q.front==q.rear
```

```
    q.append(54)
```

```
def append(self, item):  
    new_node = Node(item, None)  
    if self.is_empty():  
        self.front = new_node  
    else:  
        self.rear.next = new_node  
    self.rear = new_node
```

Useful to check cases

- A few nodes.
- Empty.
- Single node.

Serve: algorithm

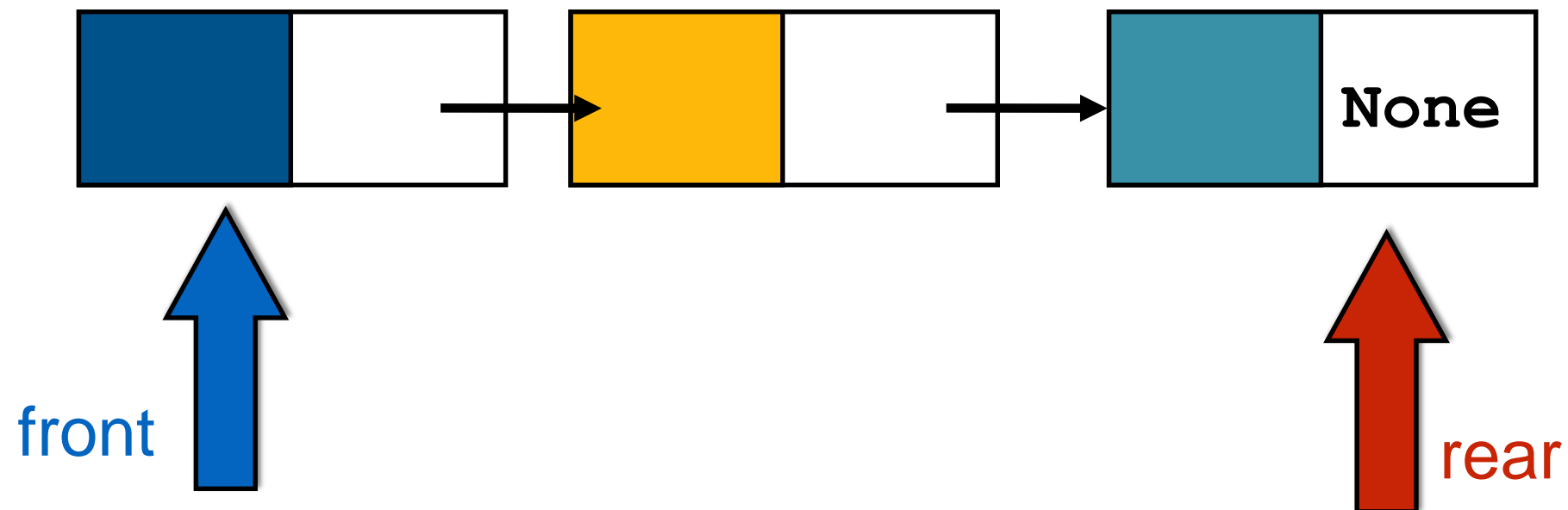
Circular array implementation:

- If the array is empty raise exception
- Else
 - Remember item to return
 - Increase front % length of the array
 - Return the item

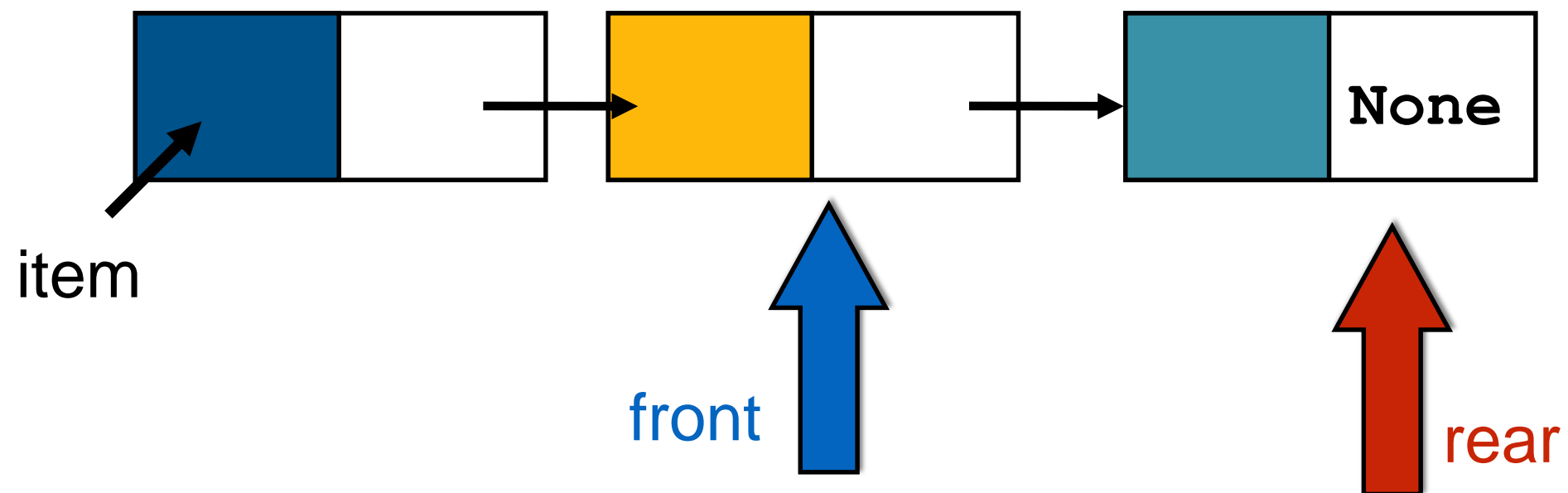
Linked implementation:

- If the array is empty raise exception
- Else
 - Remember item to return
 - Change front to point to the next node
 - Return the item

```
def serve(self):  
    item = self.front.item  
    self.front.next = self.front  
    return item
```



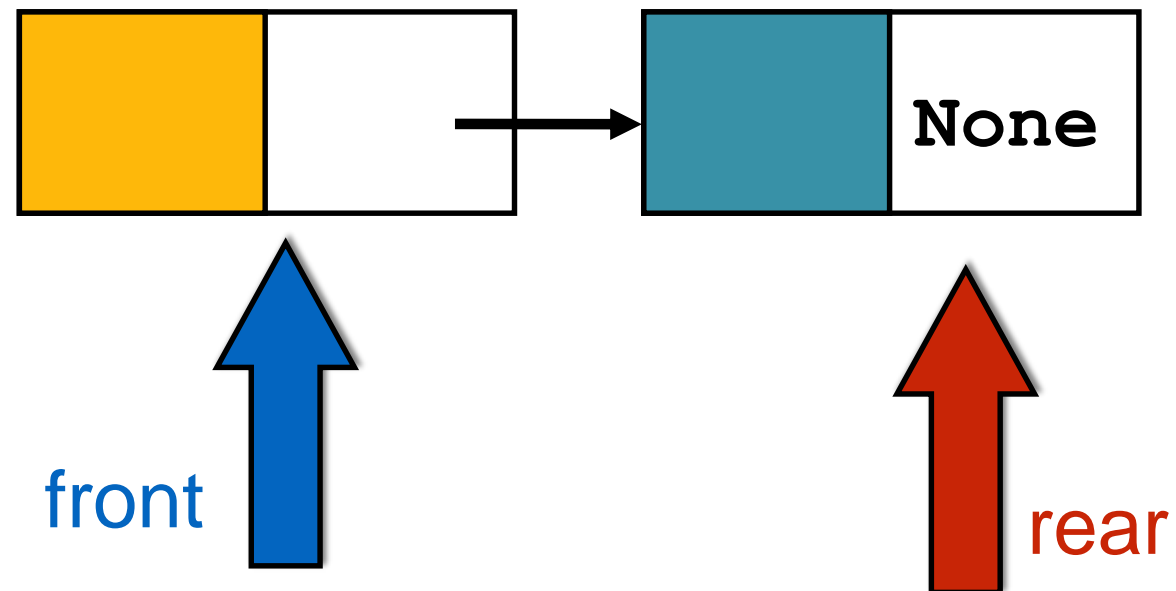
```
def serve(self):  
    item = self.front.item  
    self.front = self.front.next  
    return item
```



return



```
def serve(self):  
    item = self.front.item  
    self.front = self.front.next  
    return item
```



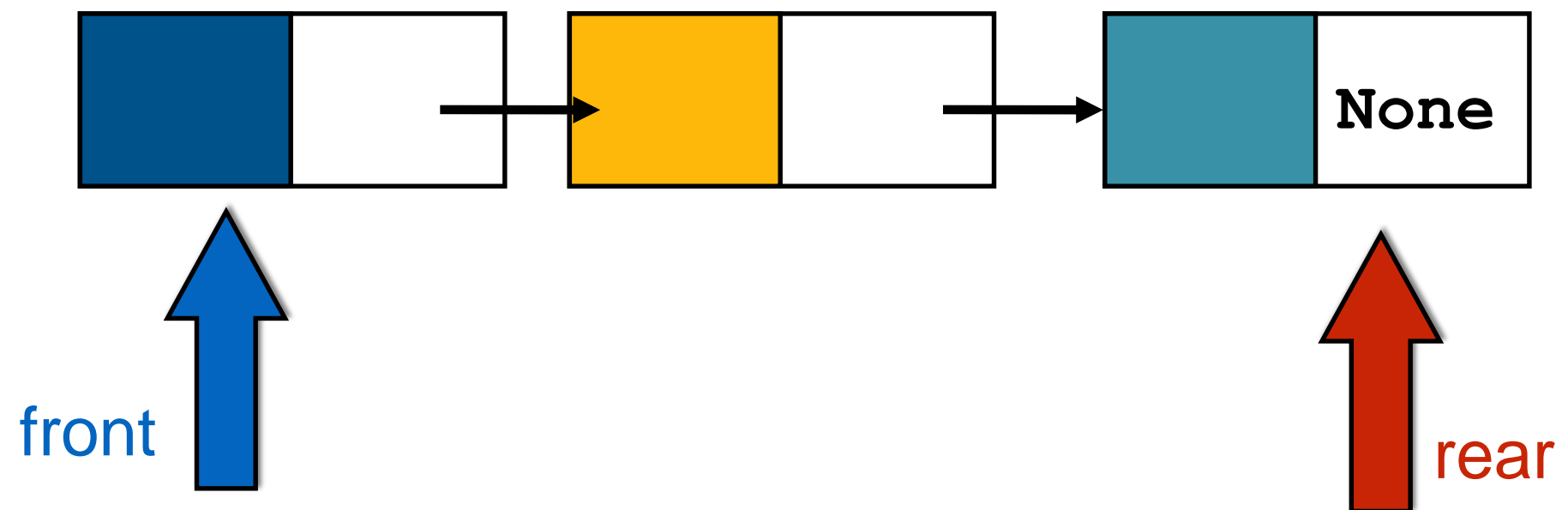
return

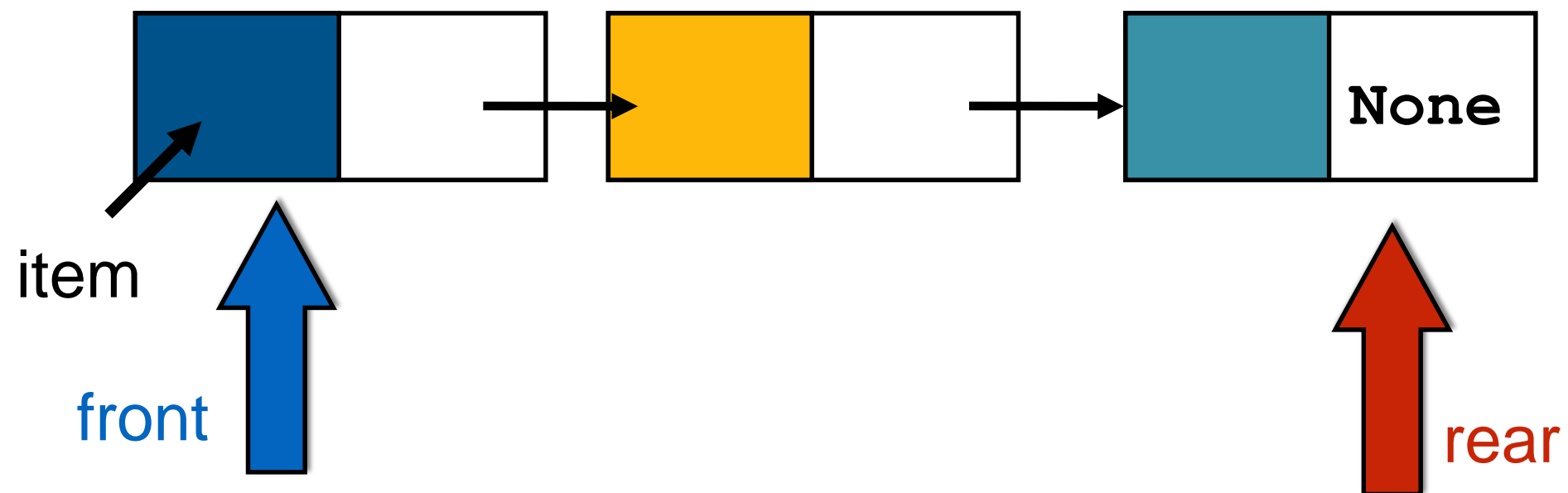




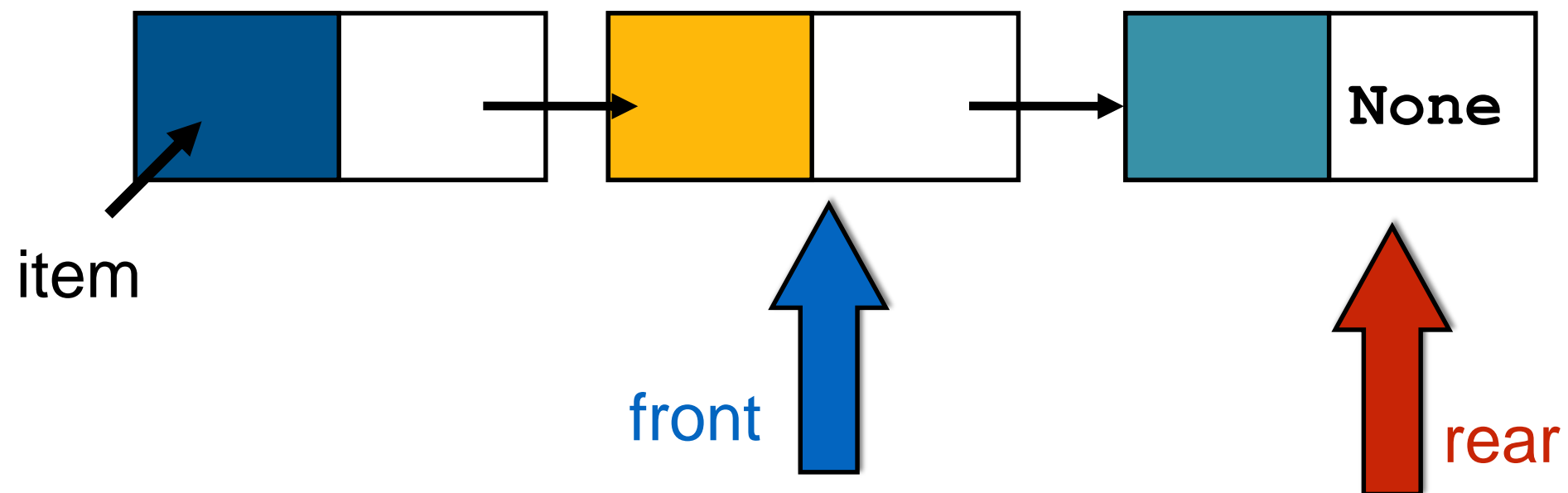
176414941

algorithm

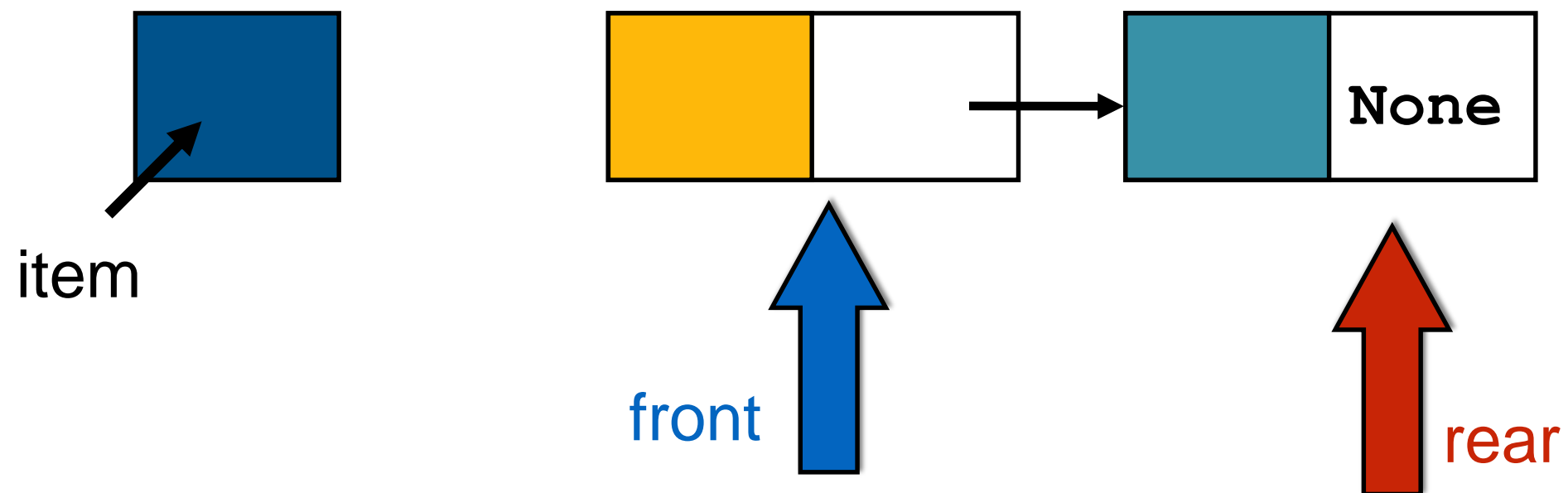




- Remember the item in the front node.

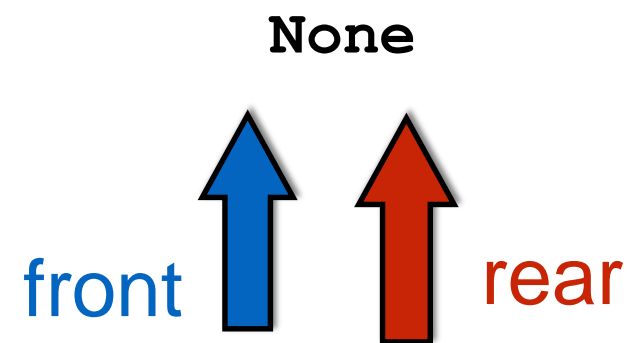


- Remember the item in the front node.
- Make the next node the new front

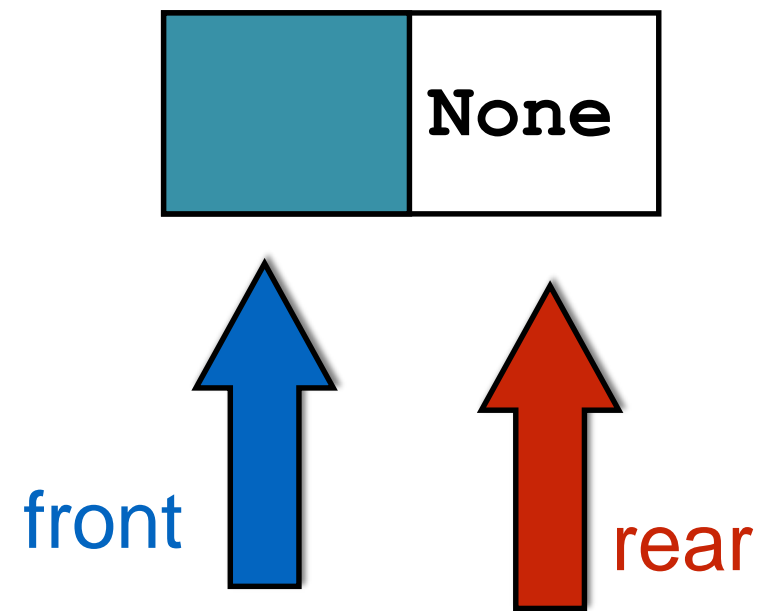


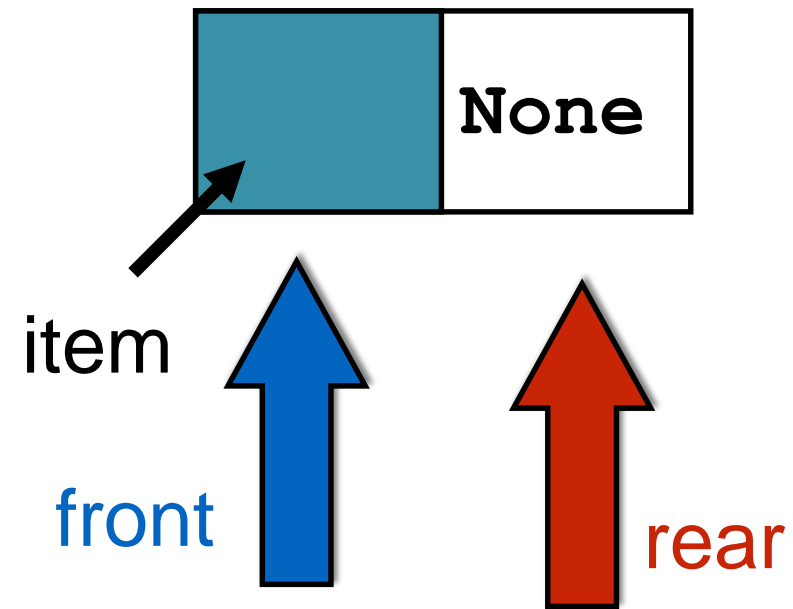
- Remember the item in the front node.
- Make the next node the new front
- Return the item

Boundary cases...

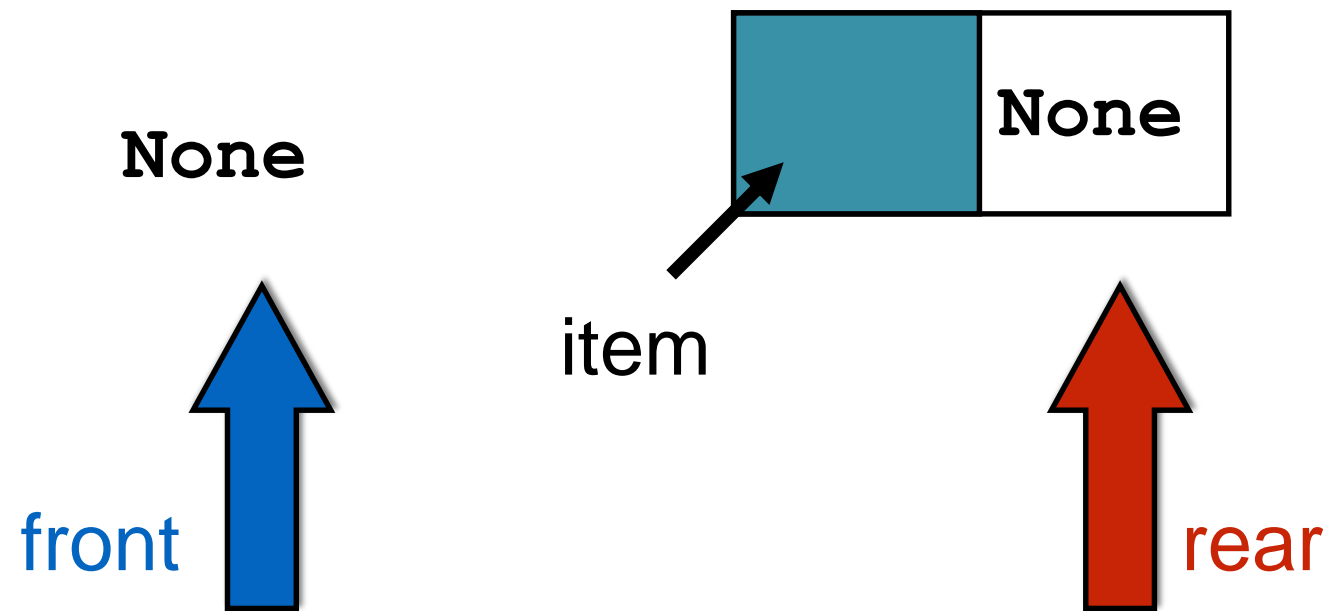


If the queue is empty we need to raise
an Exception

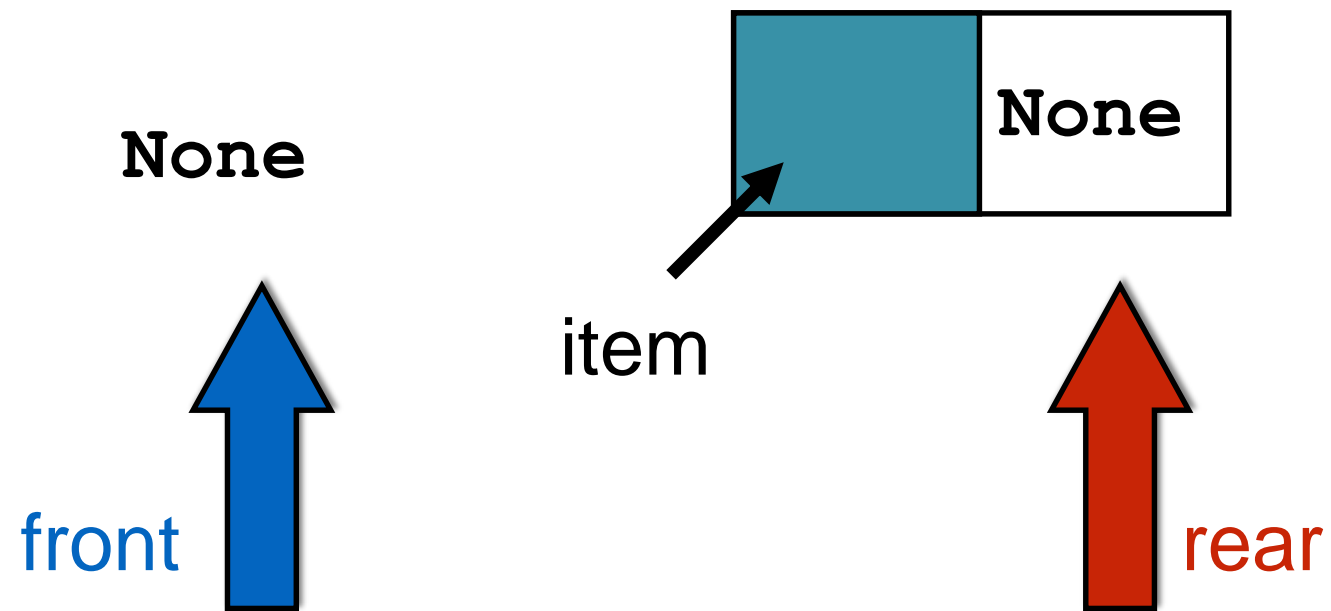




- Remember the item in the front node.

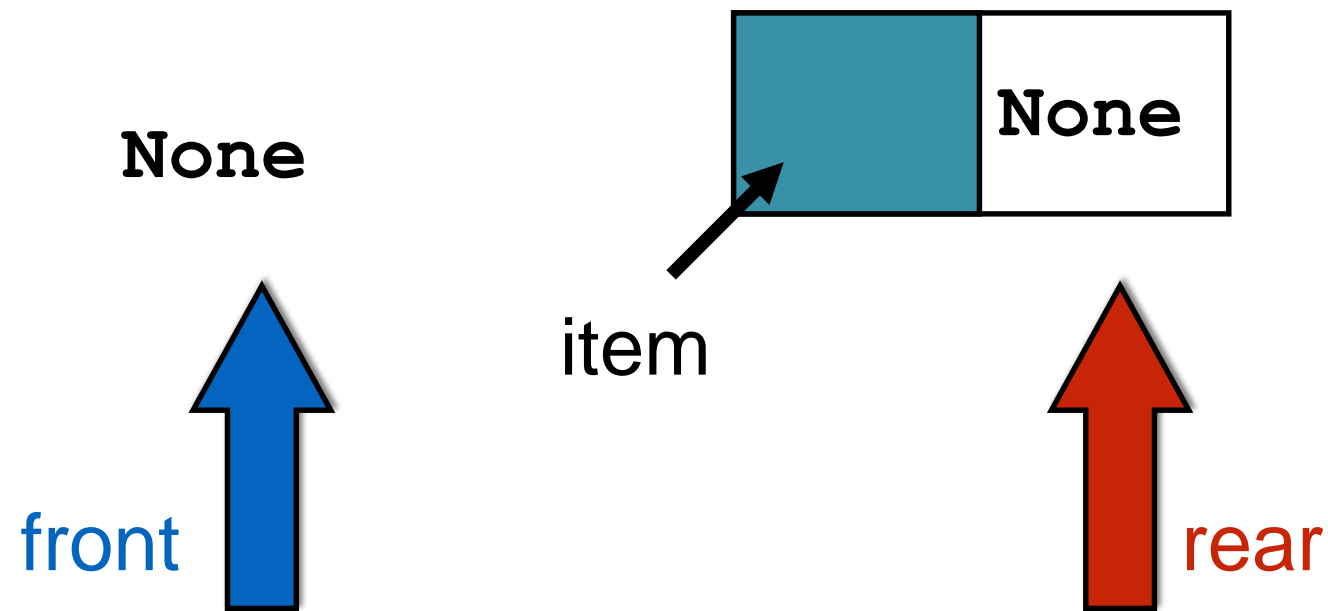


- Remember the item in the front node.
- Make the next node the new front

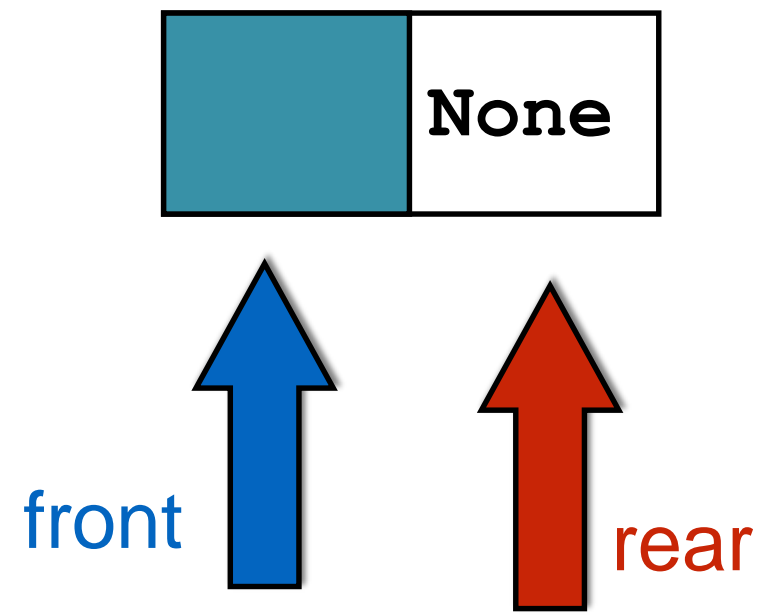


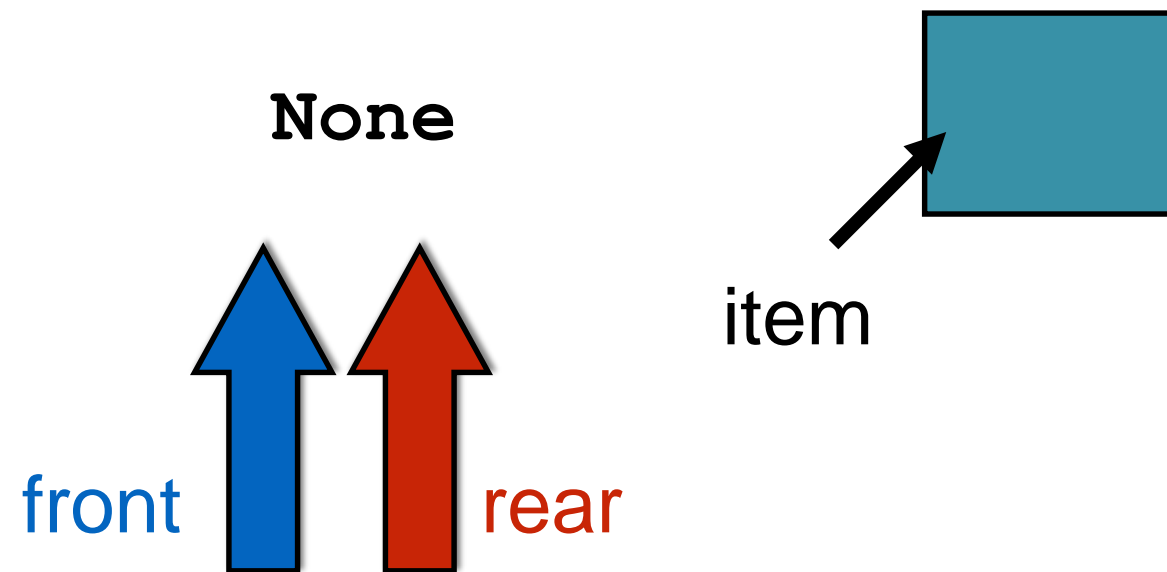
- Remember the item in the front node.
- Make the next node the new front
- Return the item

Rear is pointing to an unused Node,
but needs to point to None



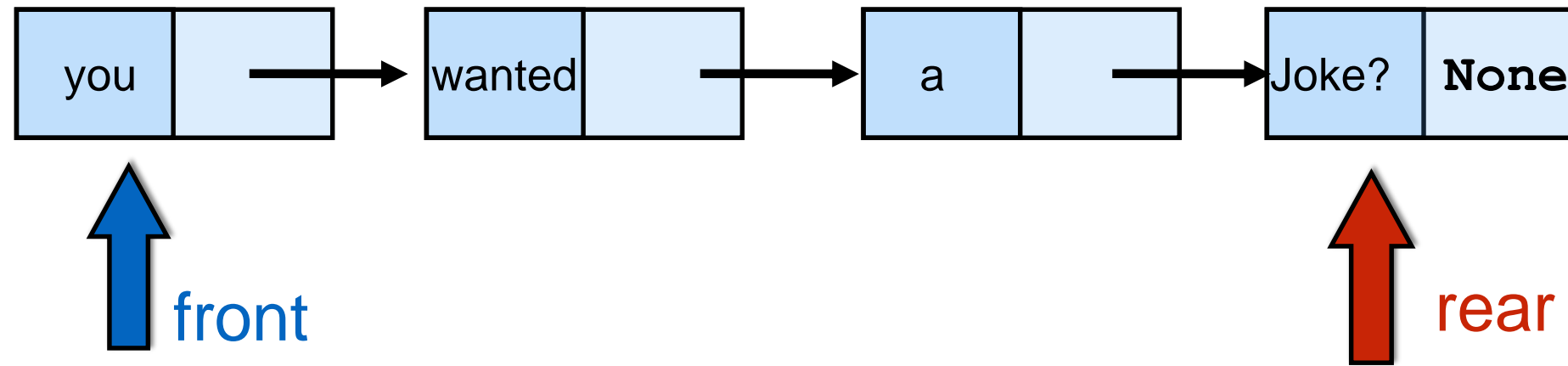
- Remember the item in the front node.
- Make the next node the new front
- Return the item





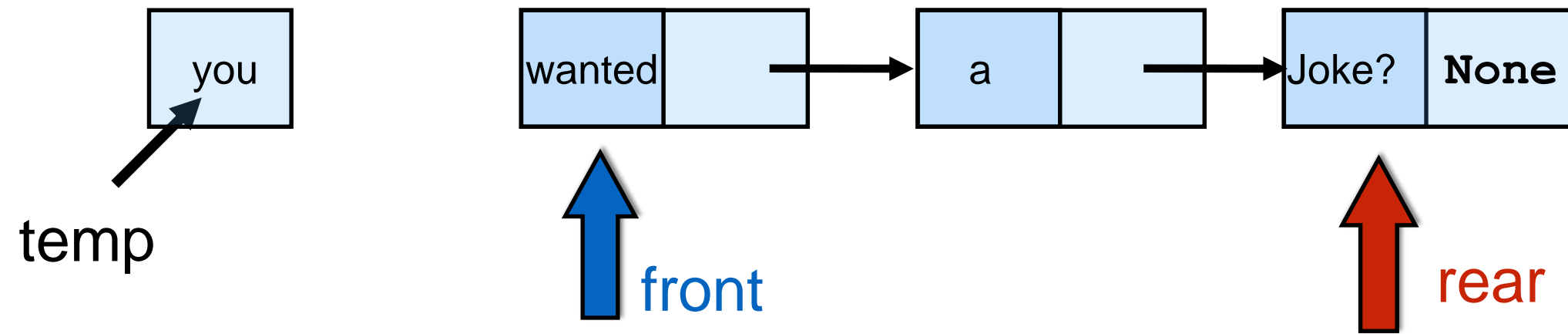
- If the queue is empty we raise an Exception
- Remember the item in the front node.
- Make the next node the new front
- If front is pointing to None (i.e., queue is now empty)
 - Point rear to None
- Return the item

```
def serve(self):  
    assert not self.is_empty(), " The queue is empty"  
    temp = self.front.item  
    self.front = self.front.next  
    if self.is_empty():  
        self.rear = None  
    return temp
```



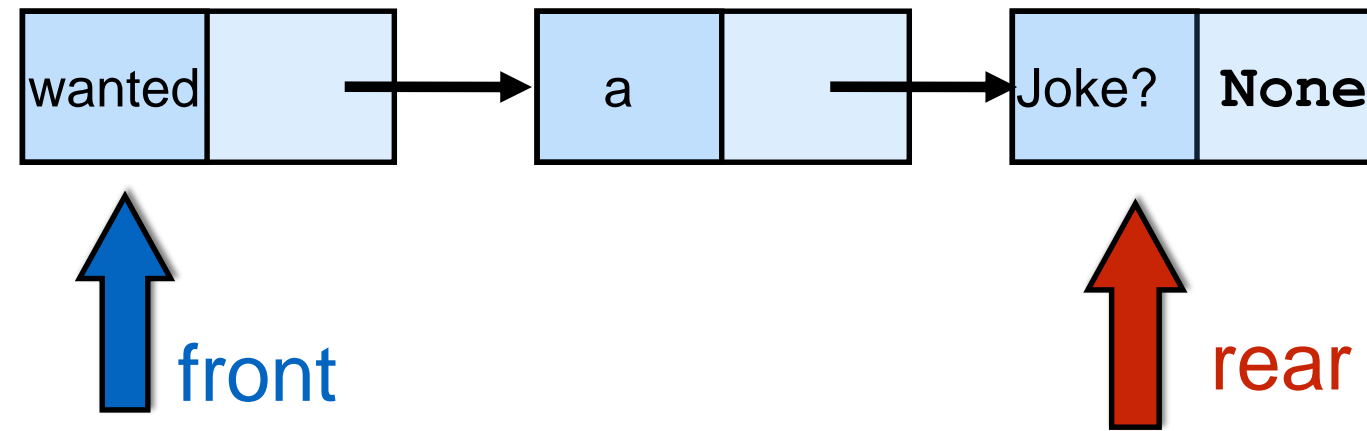
`q.serve()`

```
def serve(self):  
    assert not self.is_empty(), " The queue is empty"  
    temp = self.front.item  
    self.front = self.front.next  
    if self.is_empty():  
        self.rear = None  
    return temp
```

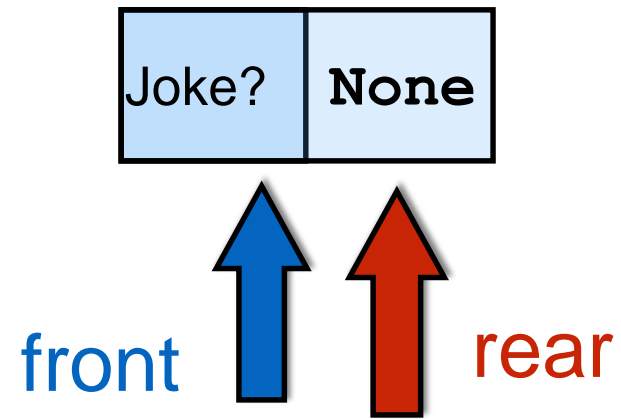
`q.serve()`

```
def serve(self):  
    assert not self.is_empty(), " The queue is empty"  
    temp = self.front.item  
    self.front = self.front.next  
    if self.is_empty():  
        self.rear = None  
    return temp
```



```
q.serve()
```

```
def serve(self):  
    assert not self.is_empty(), " The queue is empty"  
    temp = self.front.item  
    self.front = self.front.next  
    if self.is_empty():  
        self.rear = None  
    return temp
```



```
def serve(self):  
    assert not self.is_empty(), " The queue is empty"  
    temp = self.front.item  
    self.front = self.front.next  
    if self.is_empty():  
        self.rear = None  
    return temp
```



`q.serve()`

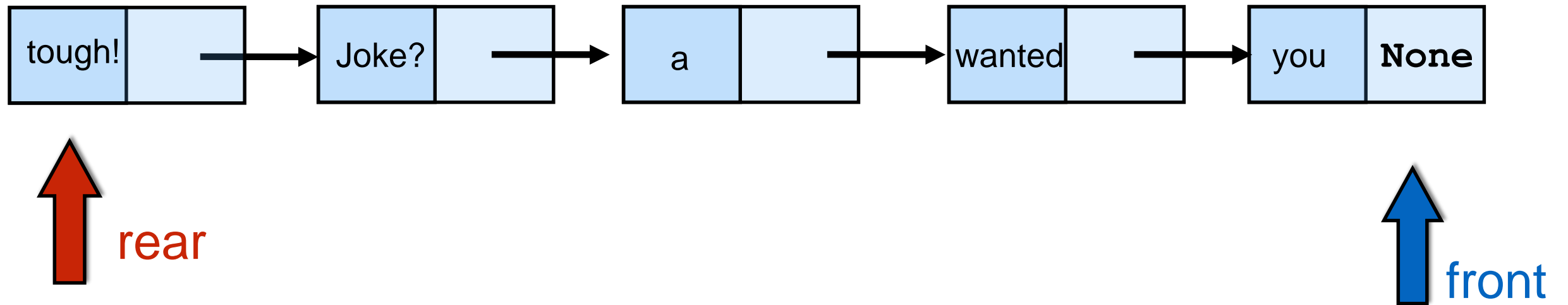
```
def serve(self):  
    assert not self.is_empty(), " The queue is empty"  
    temp = self.front.item  
    self.front = self.front.next  
    if self.is_empty():  
        self.rear = None  
    return temp
```



`q.serve()`

```
def serve(self):  
    assert not self.is_empty(), "The queue is empty"  
    temp = self.front.item  
    self.front = self.front.next  
    if self.is_empty():  
        self.rear = None  
    return temp
```

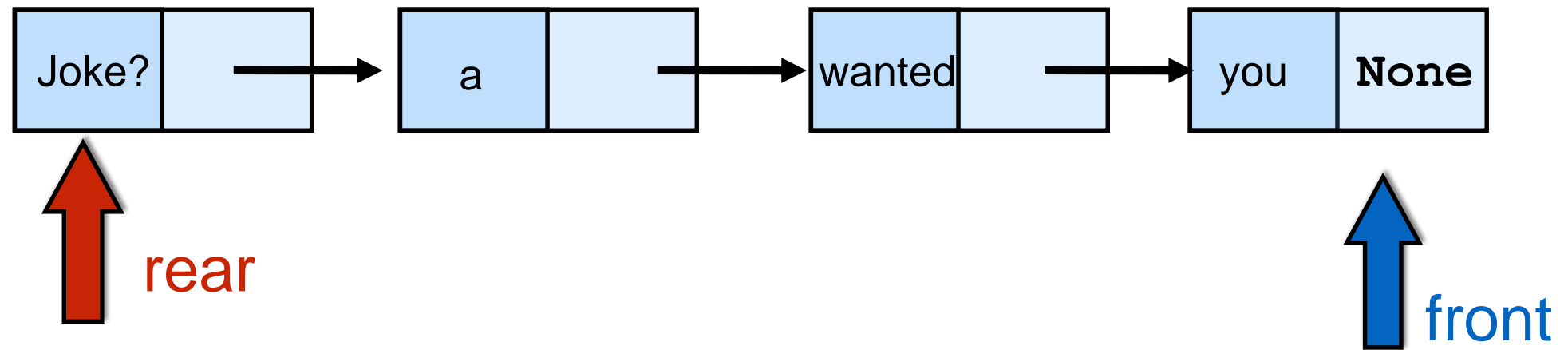
append



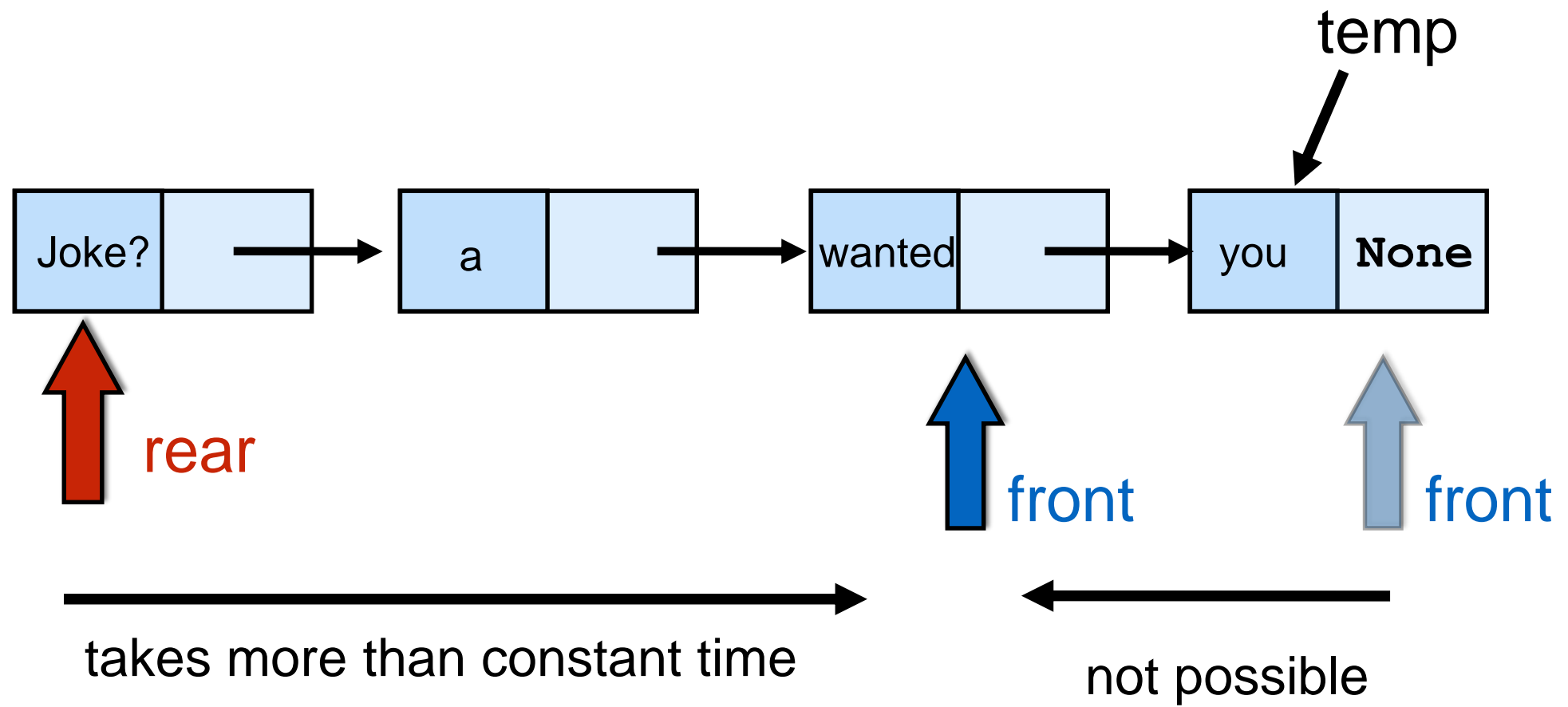
```
q.append("tough!")
```

Seems to work.

serve



`q.serve()`



`q.serve()`

Summary

- Queues implemented with linked data structures