Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 25
Linked Queues

FIT 1008
Introduction to Computer Science

MONASH University
Information Technology

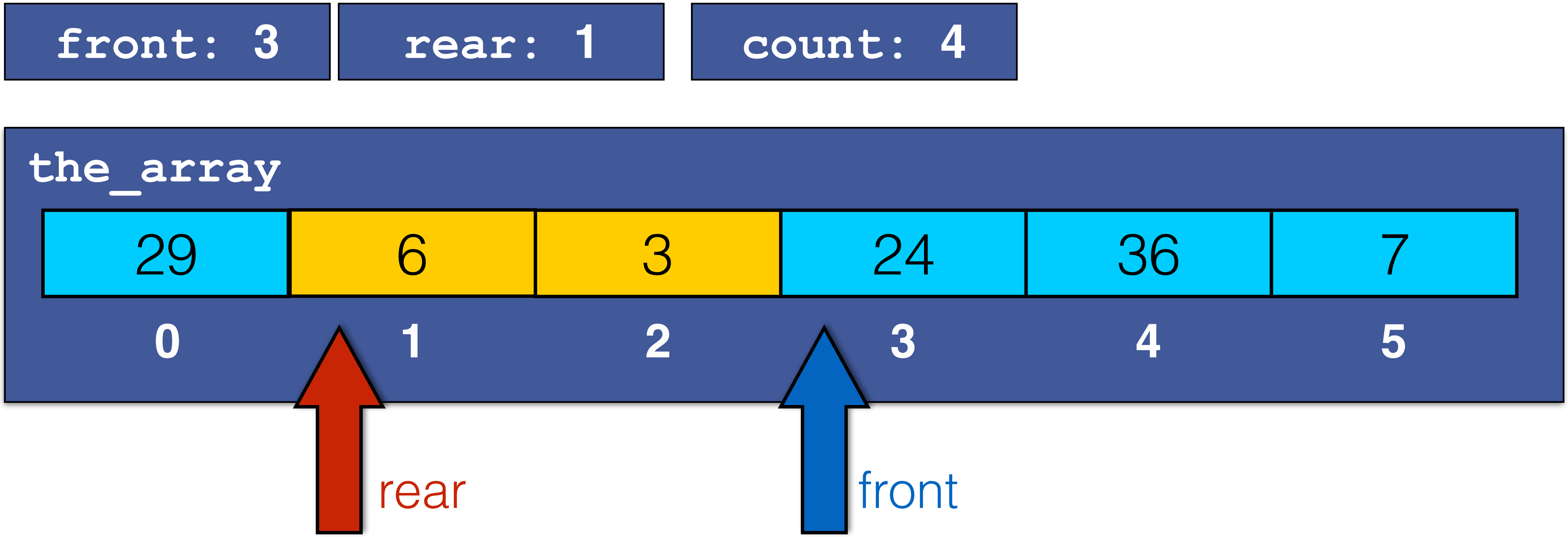
Objectives for these this lecture

- 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 queue to the left..”

Remember array-based queues?



front: 3

rear: 1

count: 4

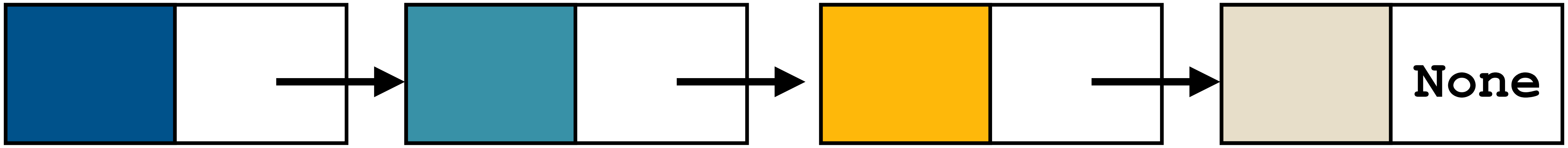
the_array

0	1	2	3	4	5
29	6	3	24	36	7

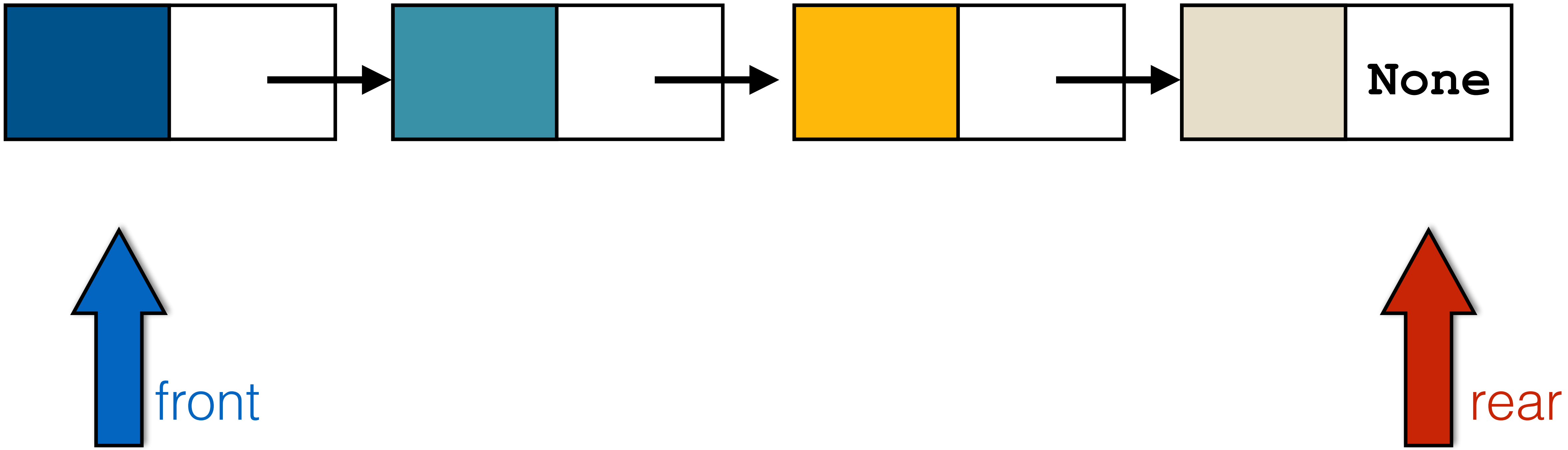
rear

front

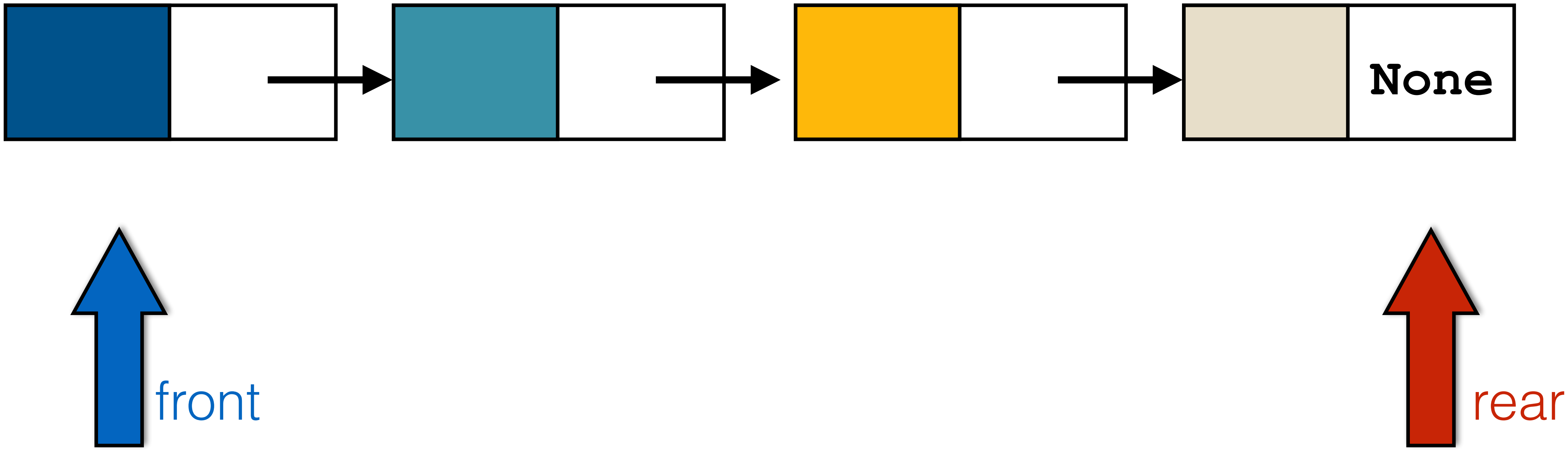
Linked Queue



Linked Queue

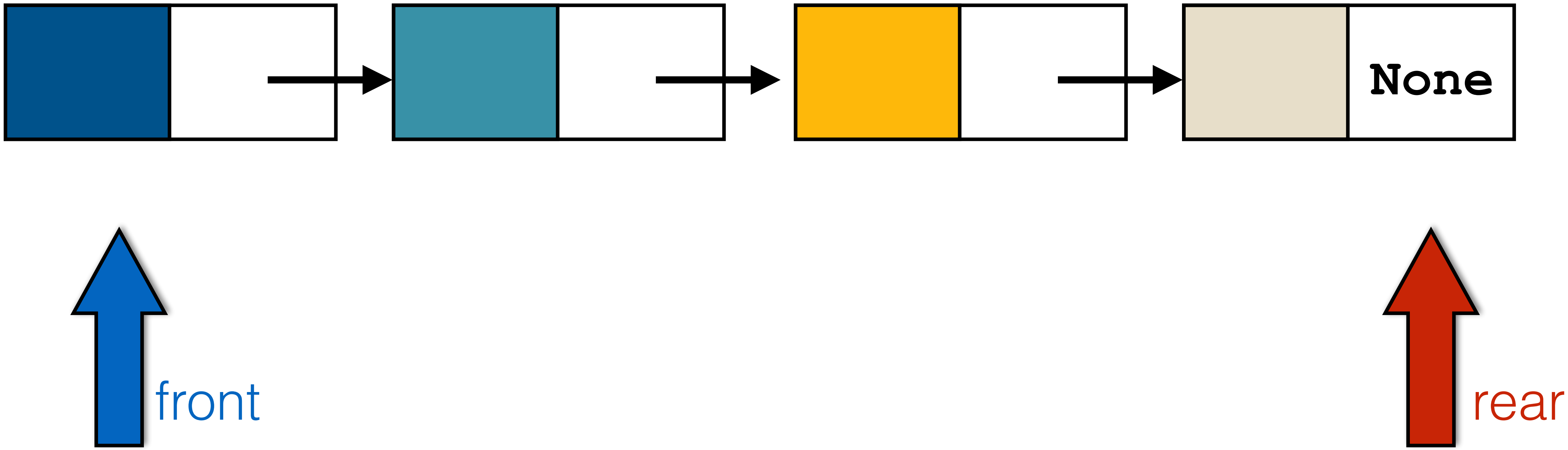


Linked Queue



Important: Rear now designates the last node

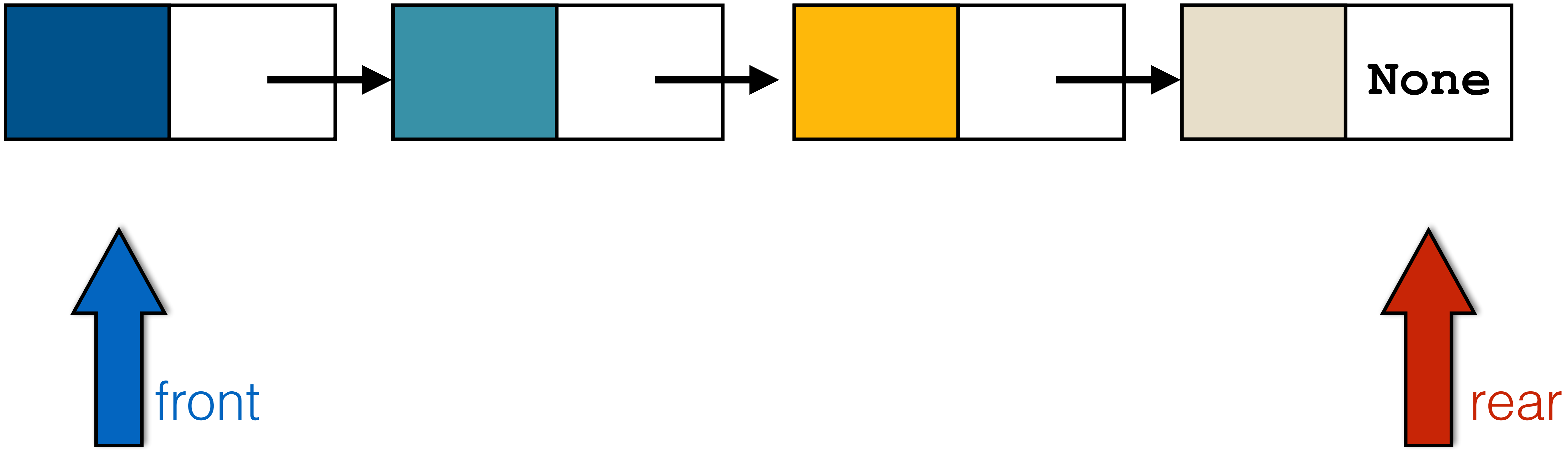
Linked Queue



Important: Rear now designates the last node

No need for circularity.

Linked Queue

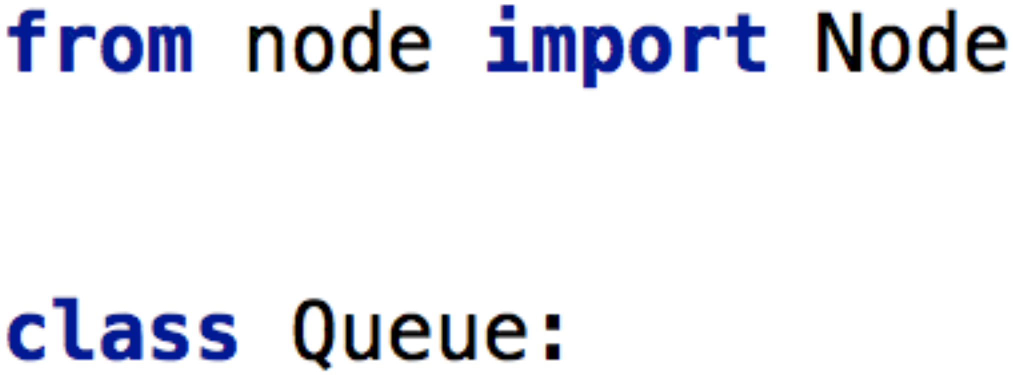


Important: Rear now designates the last node

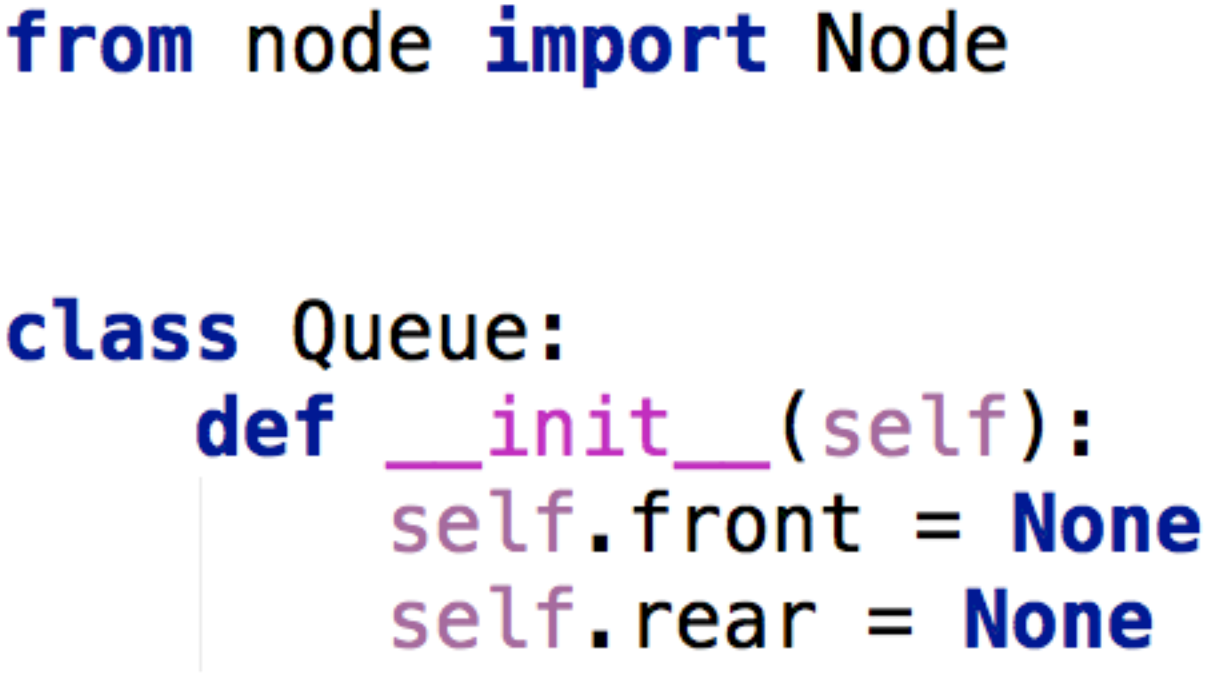
No need for circularity.

count is optional…

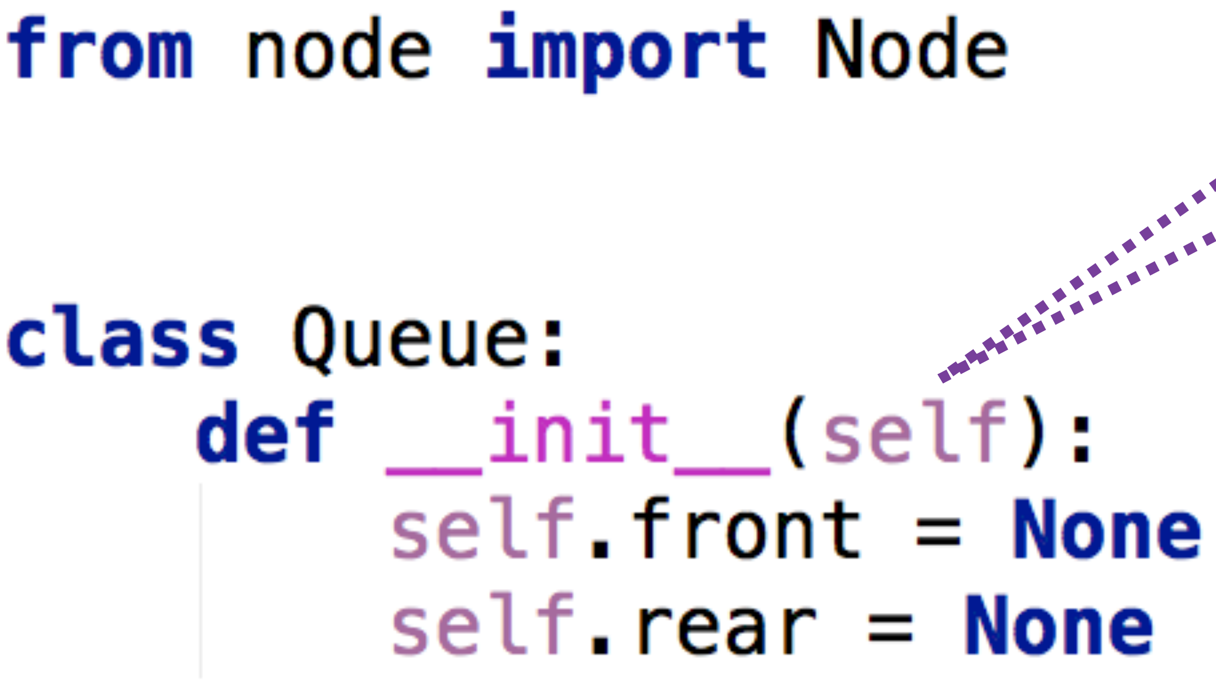
```
from node import Node
```



```python
from node import Node


class Queue:
```

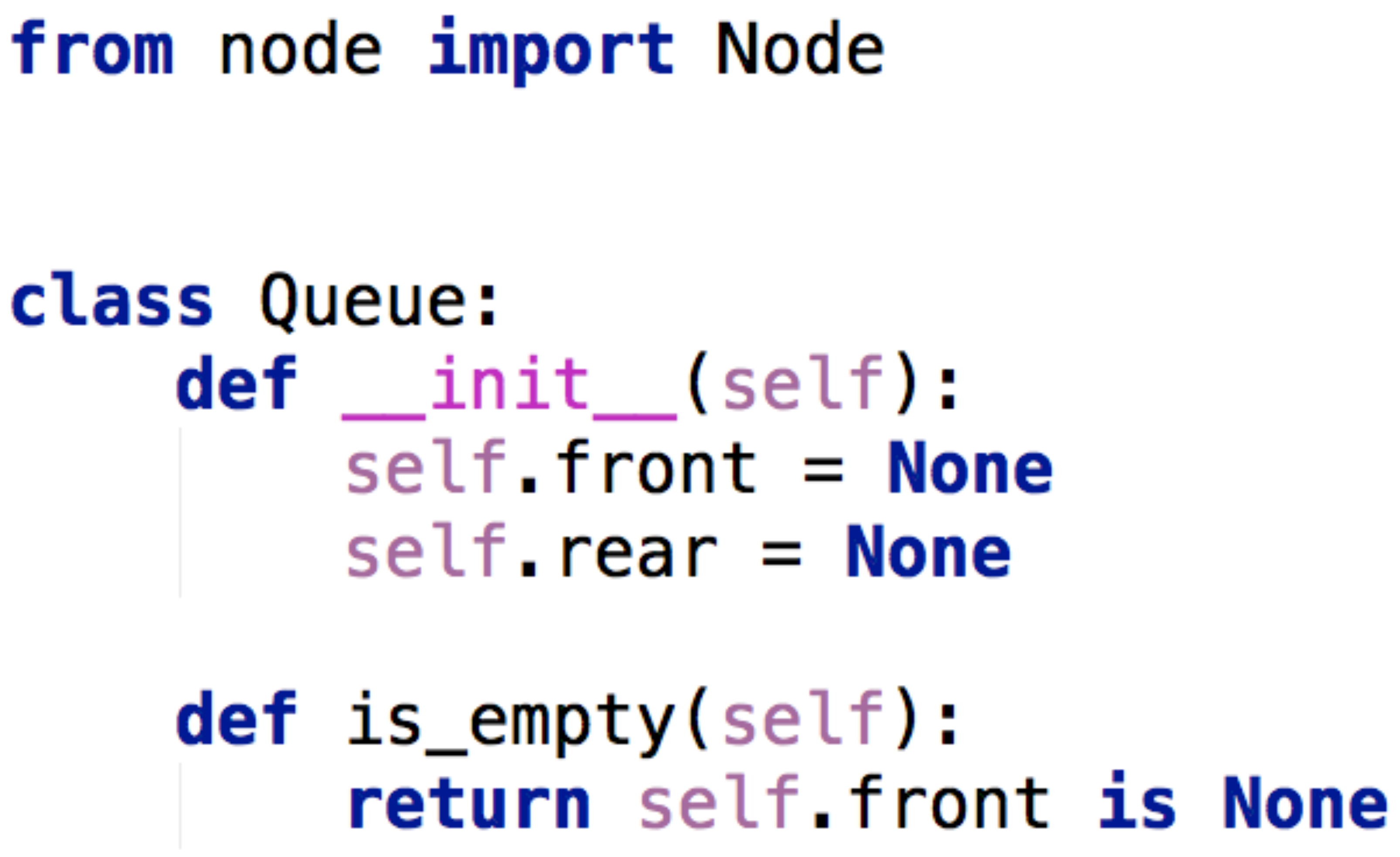


```python
from node import Node


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

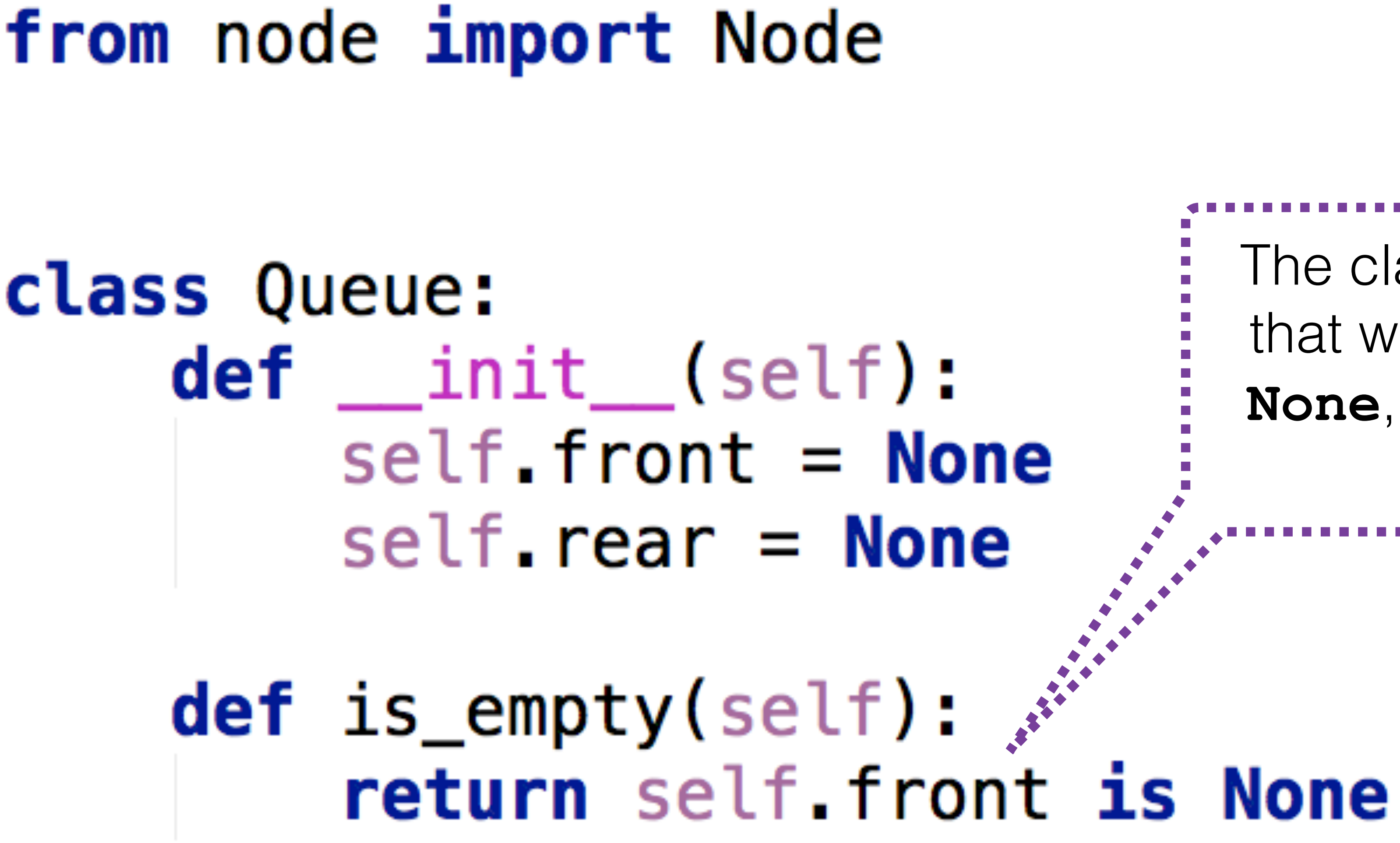


```python
from node import Node


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

No need for size when initialising the object

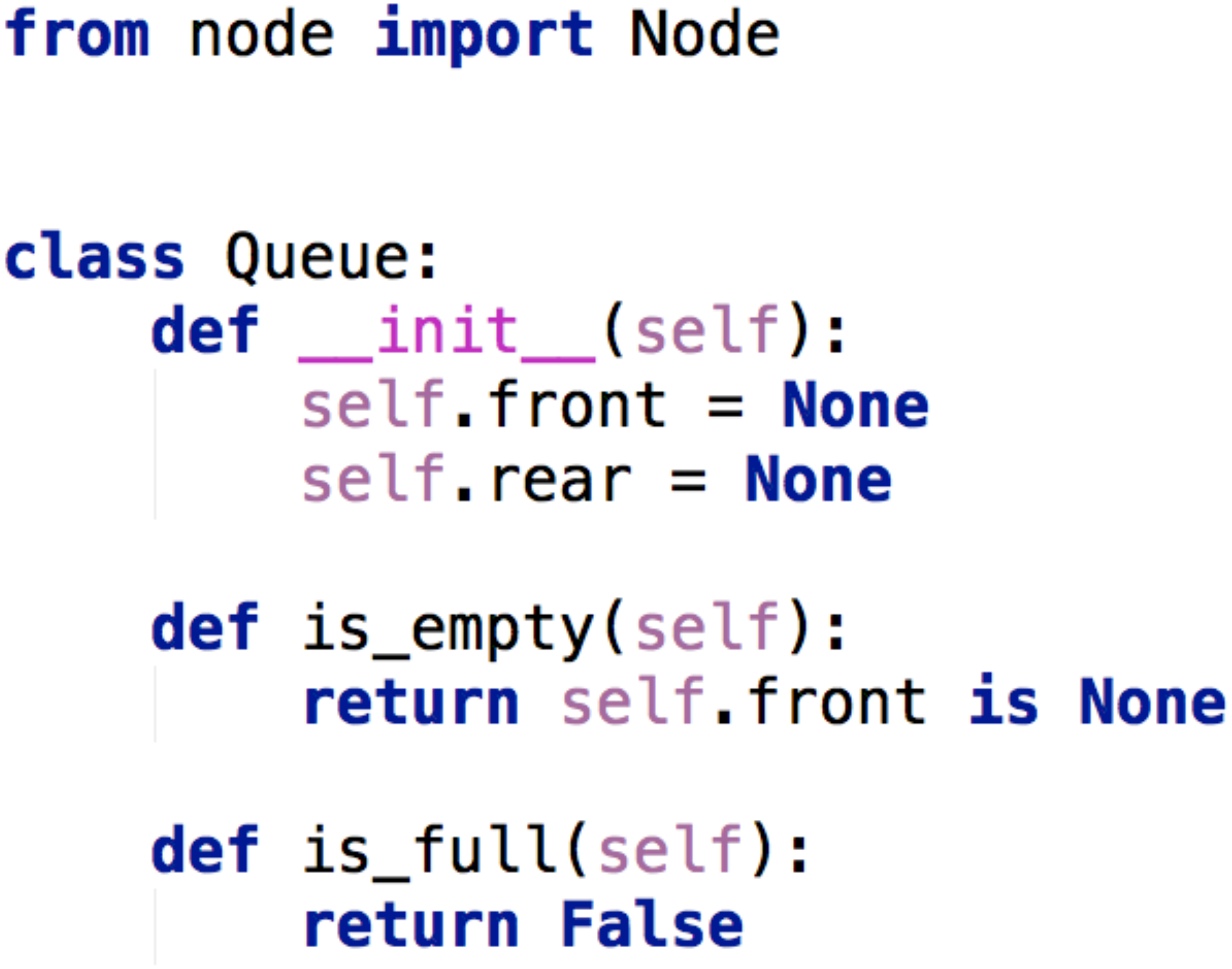


```python
from node import Node


class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

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

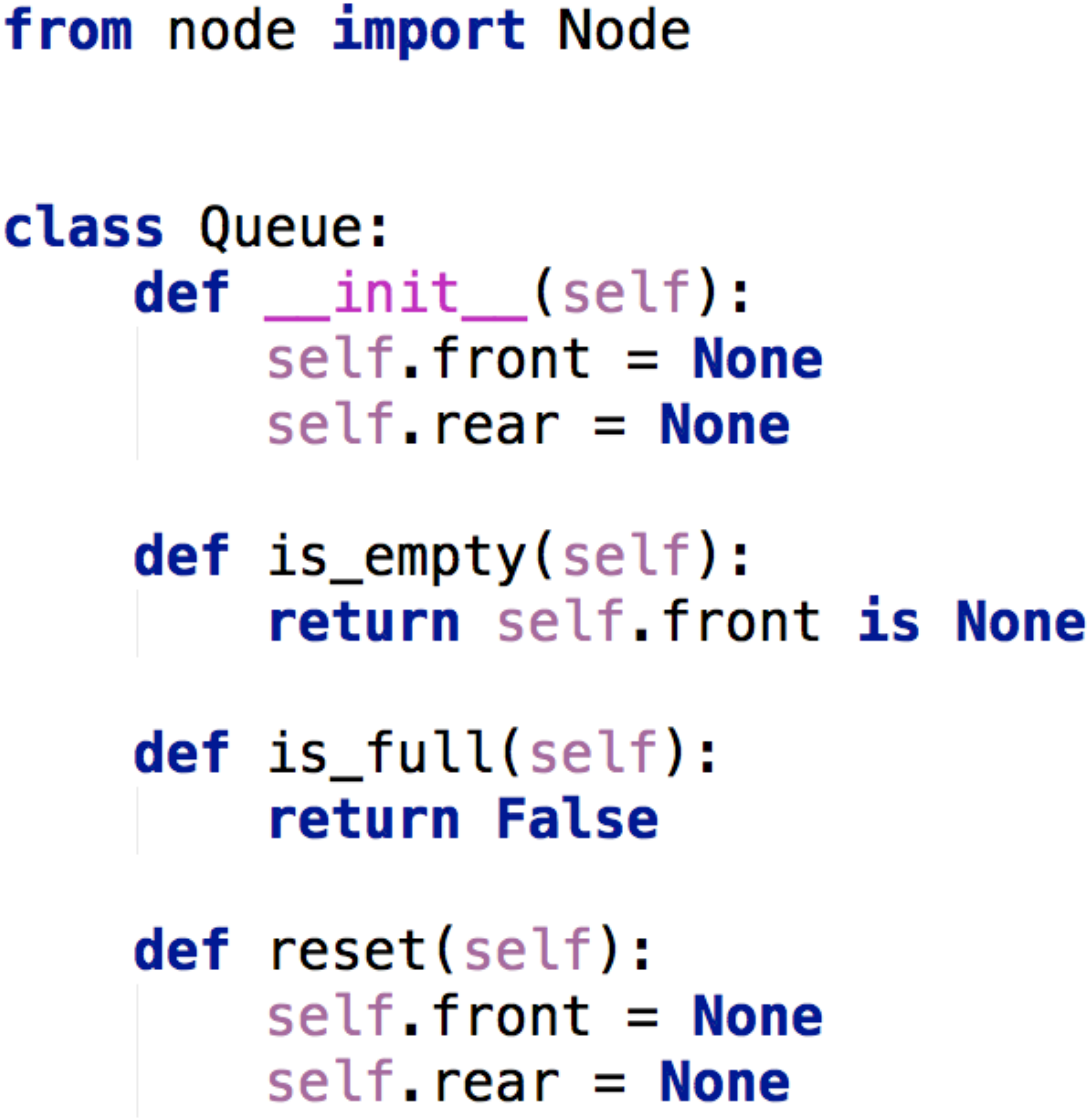


```python
from node import Node


class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

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

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



```python
from node import Node


class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

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

    def is_full(self):
        return False
```



```python
from node import Node


class Queue:
    def __init__(self):
        self.front = None
        self.rear = 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:

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

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

Linked implementation:

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

Linked implementation:

- Create a **new node** that contains item and points to 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

Linked implementation:

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

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

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.

Append: algorithm

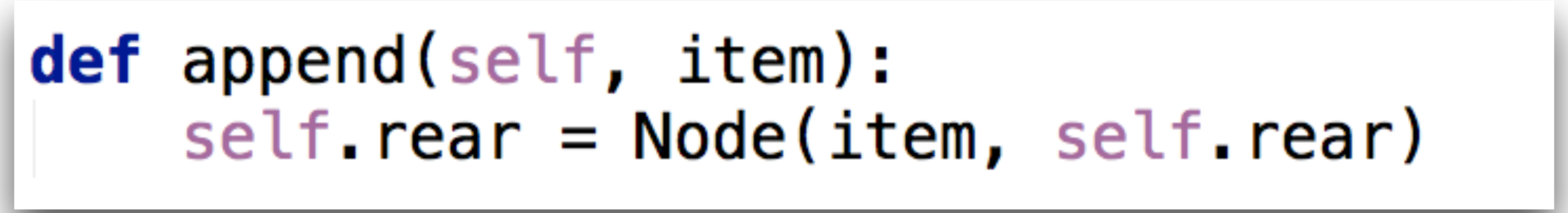
Circular array implementation:

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

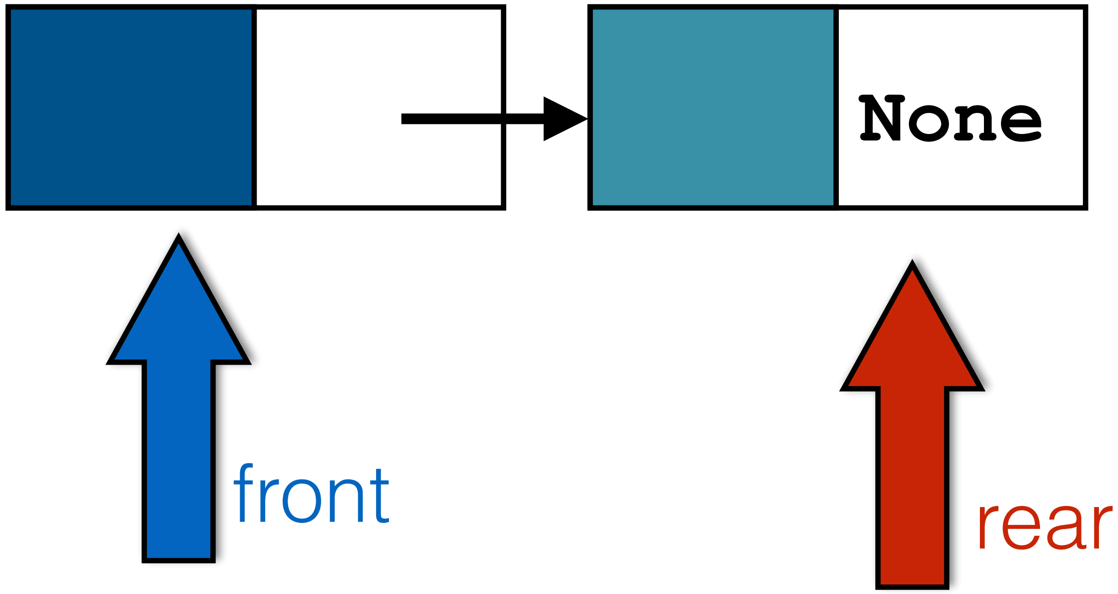
No need for is_full check.
If no more memory can be allocated the system will raise an exception.

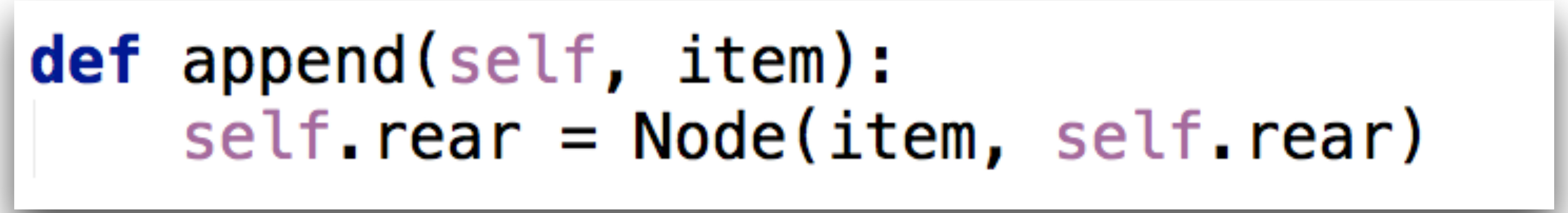
Linked implementation:

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

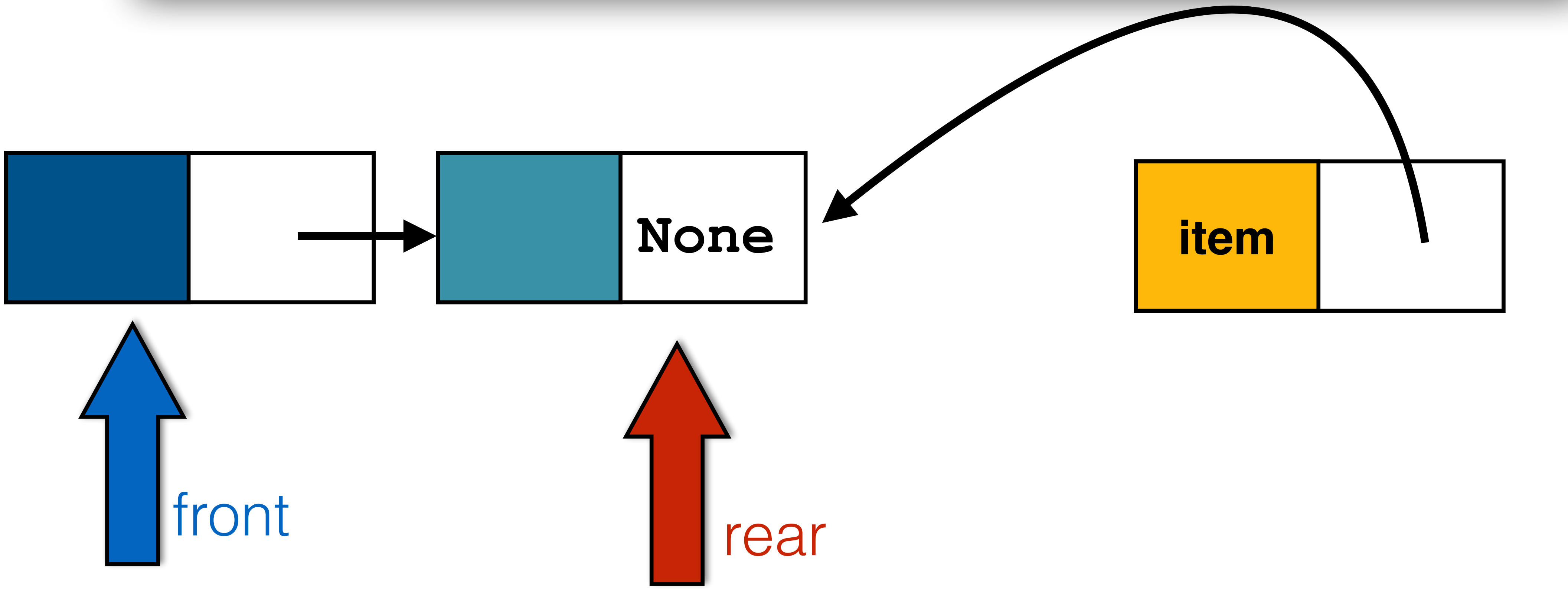


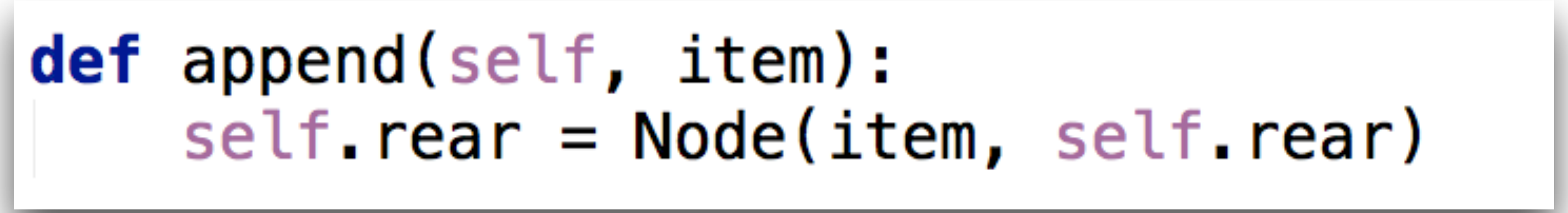
```
def append(self, item):
    self.rear = Node(item, self.rear)
```



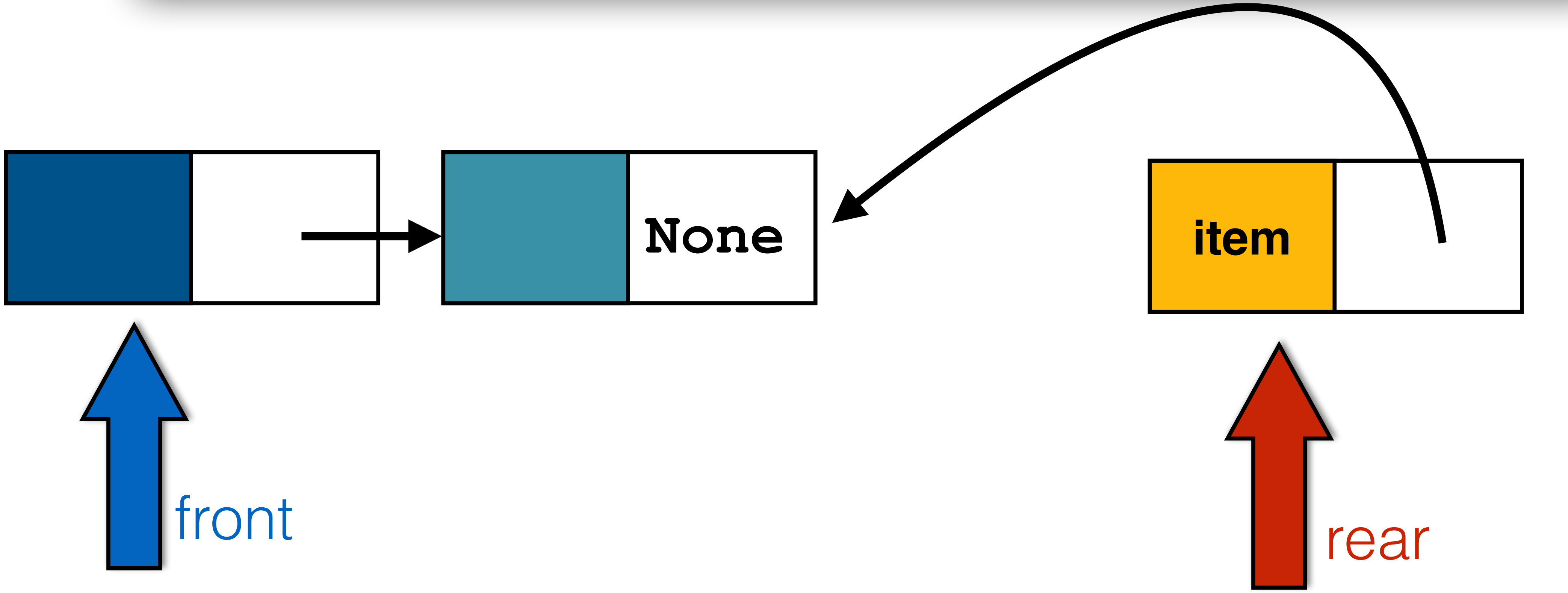


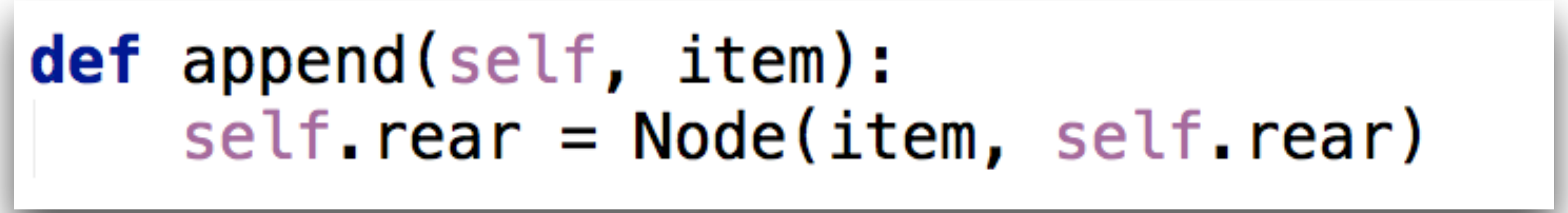
```
def append(self, item):
    self.rear = Node(item, self.rear)
```



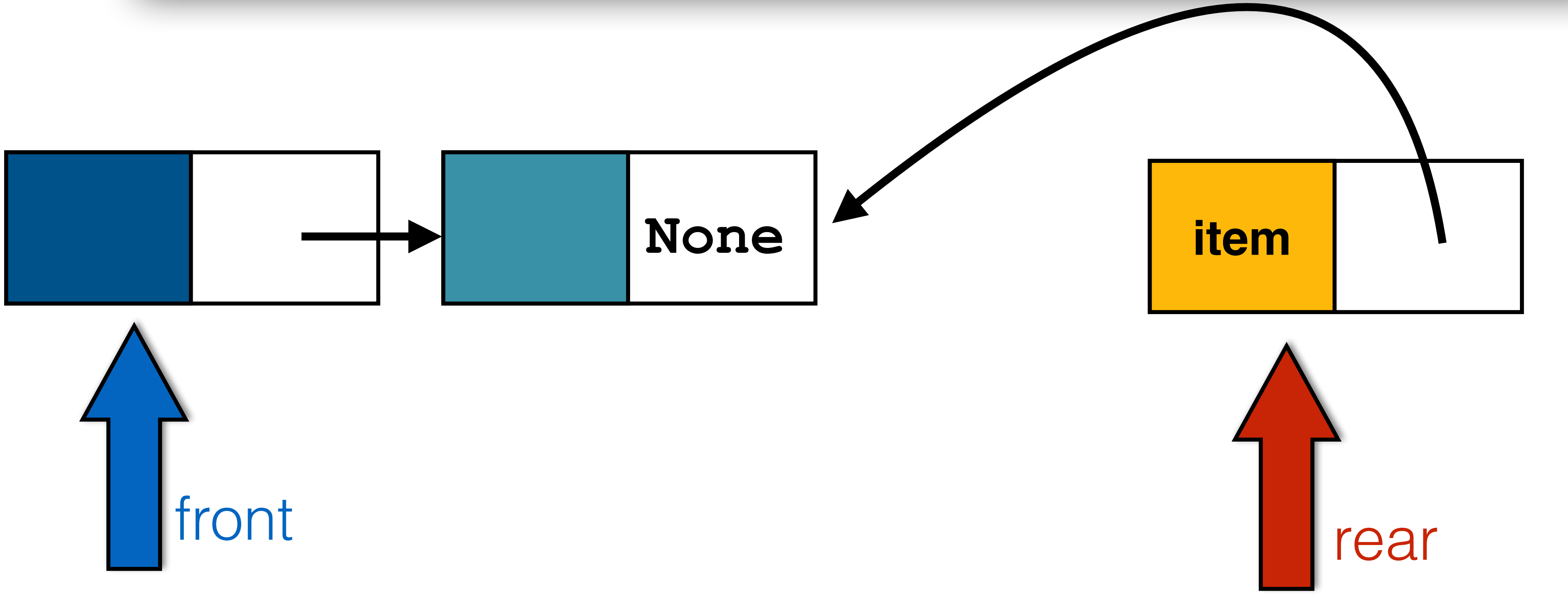


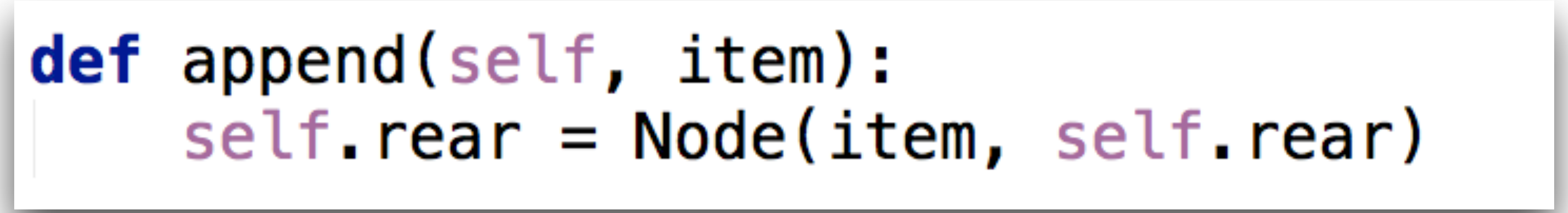
```python
def append(self, item):
    self.rear = Node(item, self.rear)
```



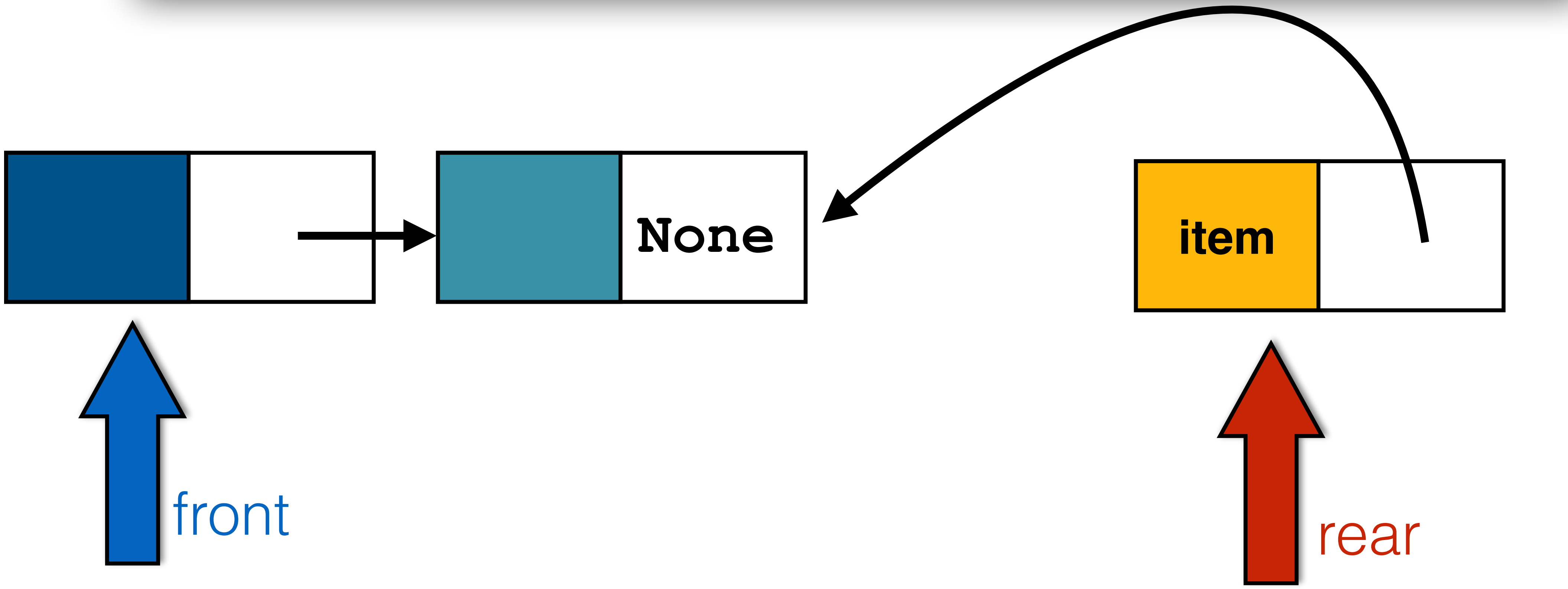


```
def append(self, item):
    self.rear = Node(item, self.rear)
```

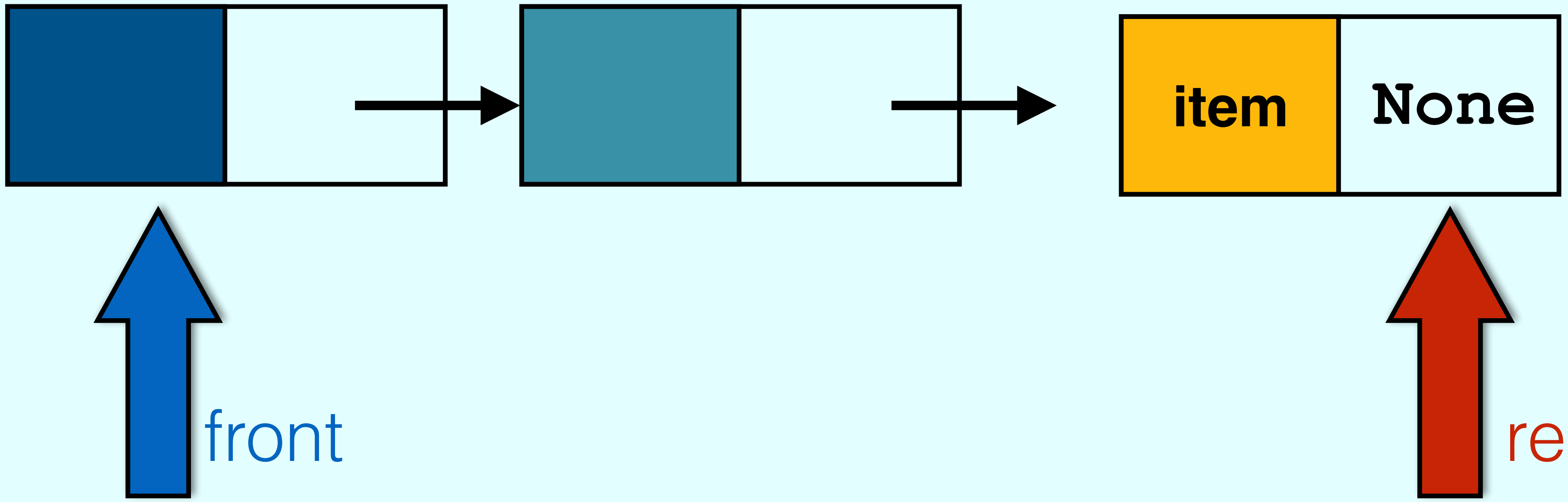


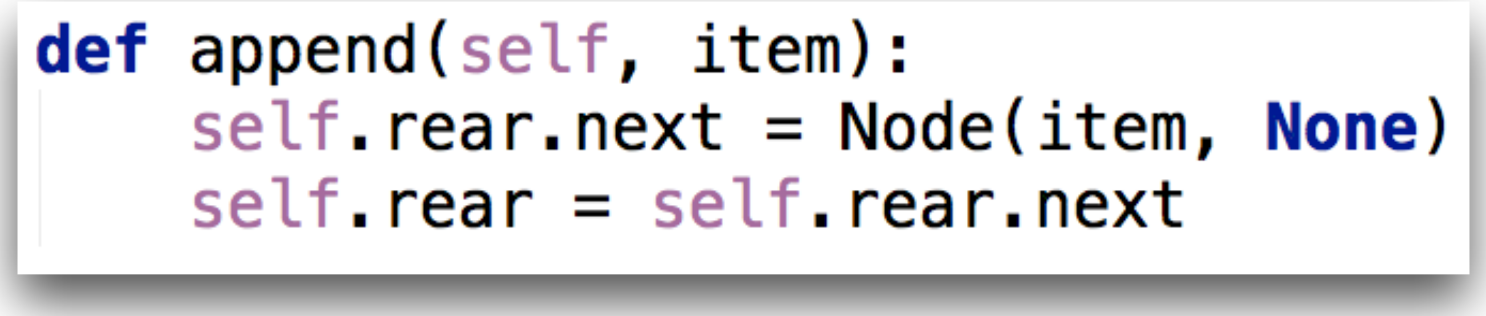


```python
def append(self, item):
    self.rear = Node(item, self.rear)
```

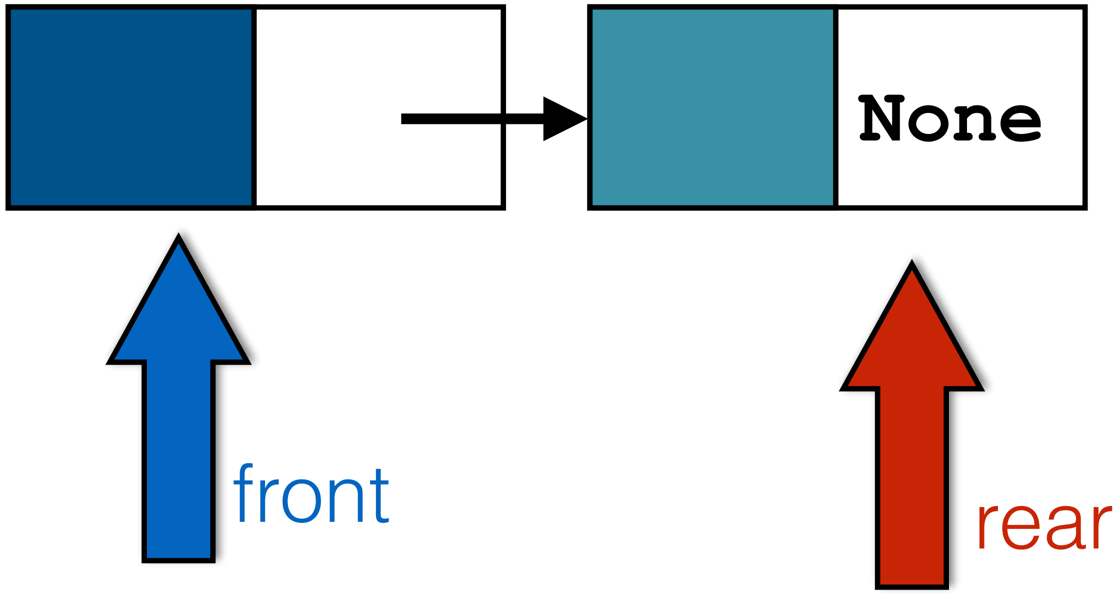


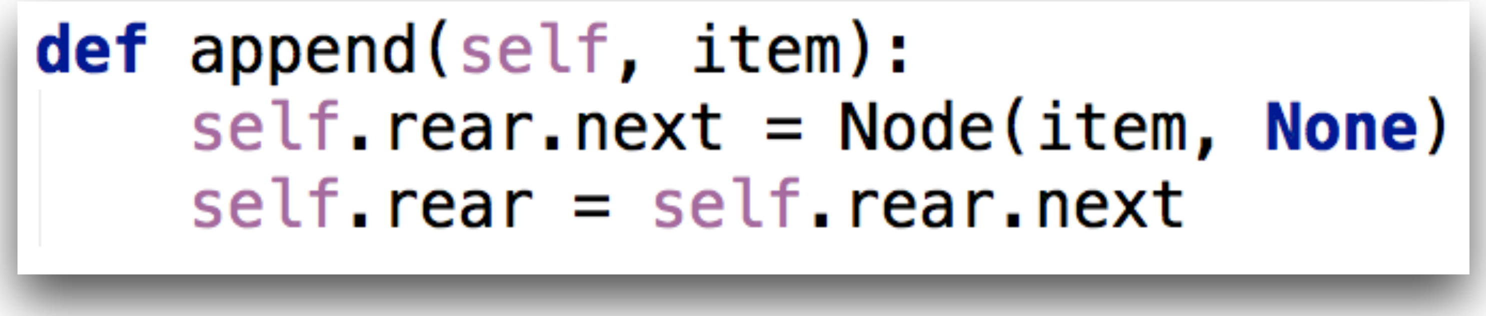
Goal



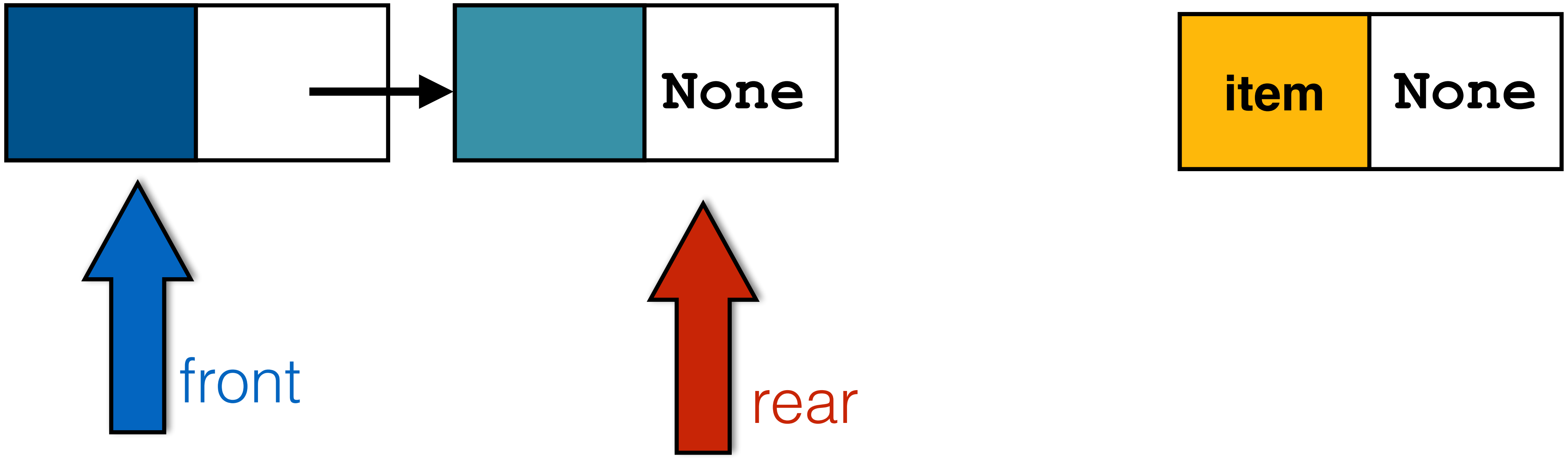


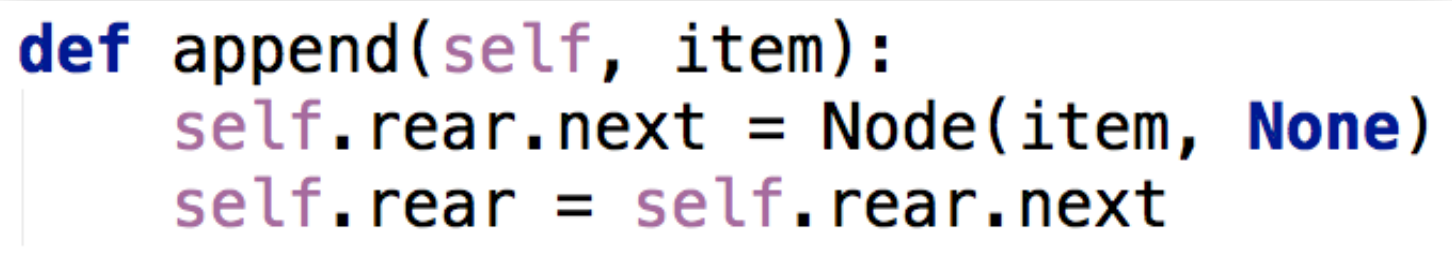
```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



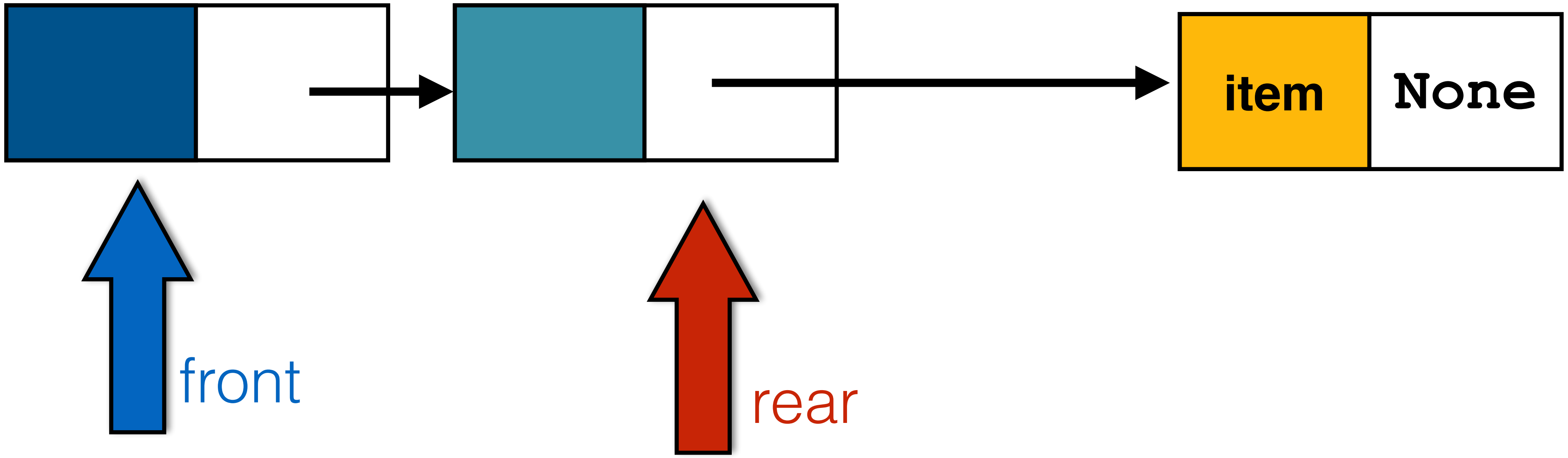


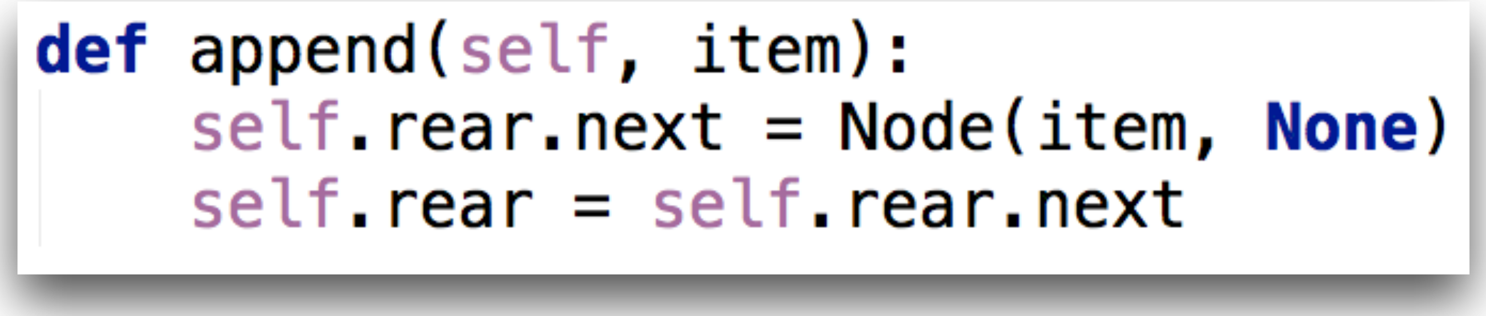
```python
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



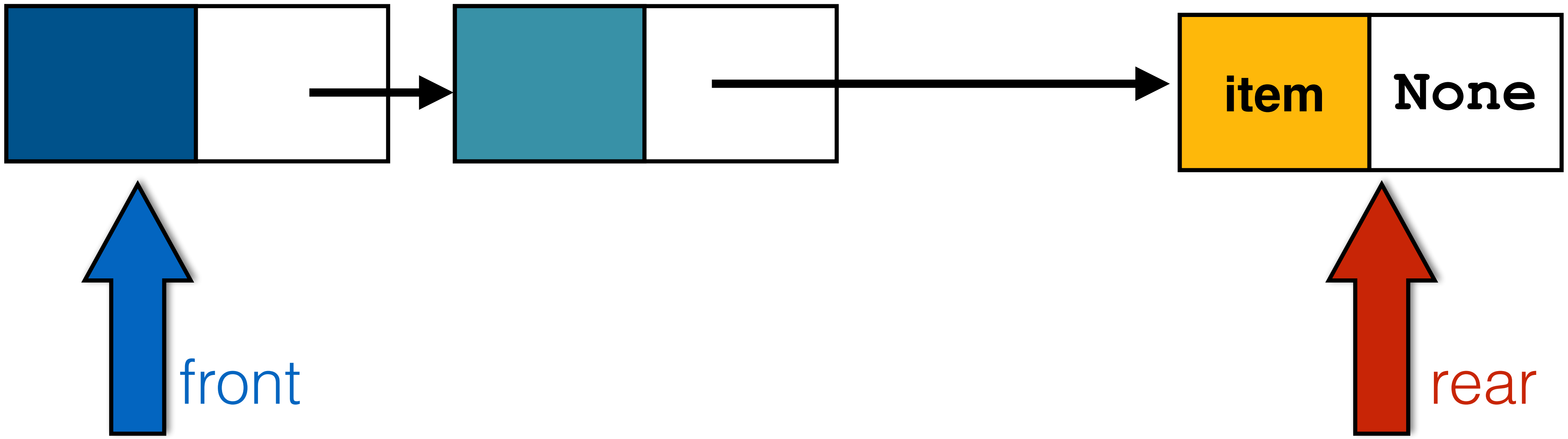


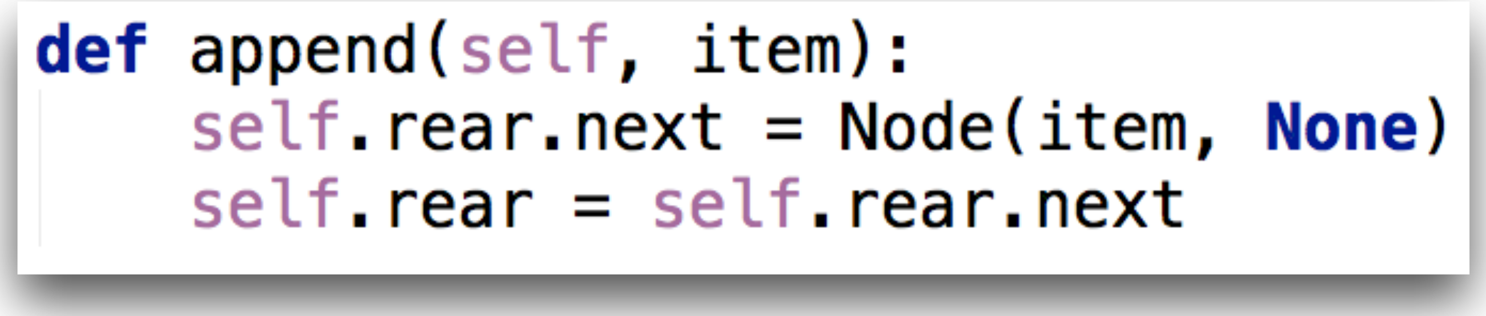
```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



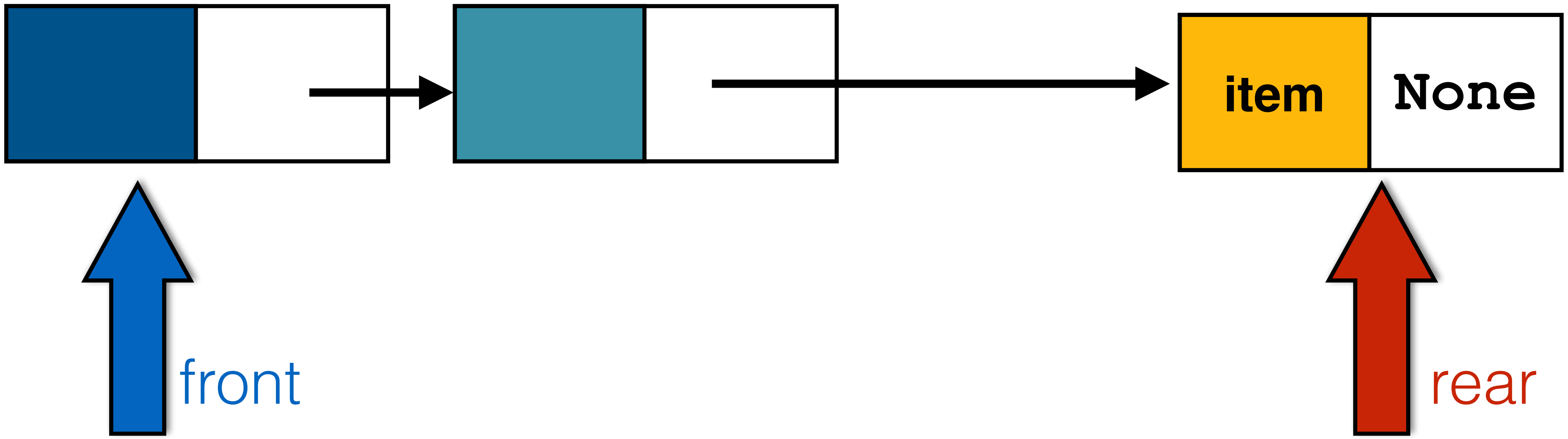


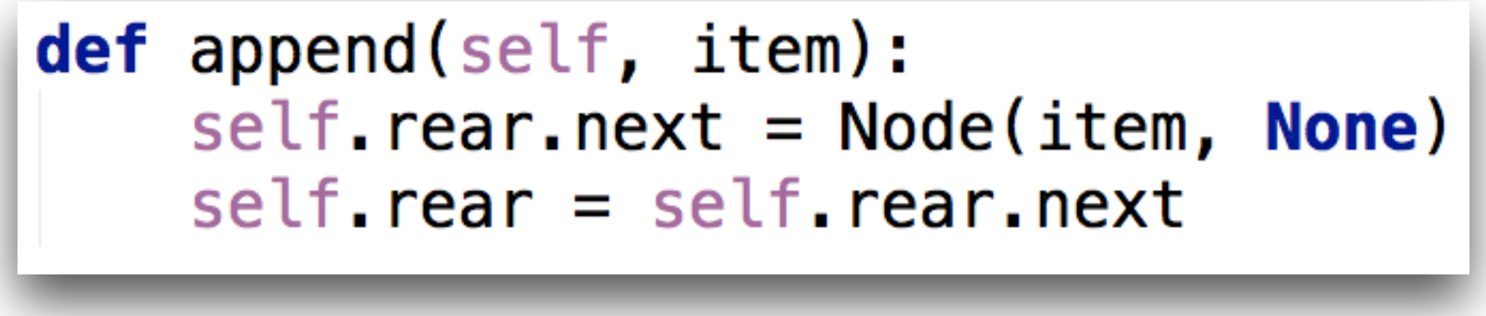
```python
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



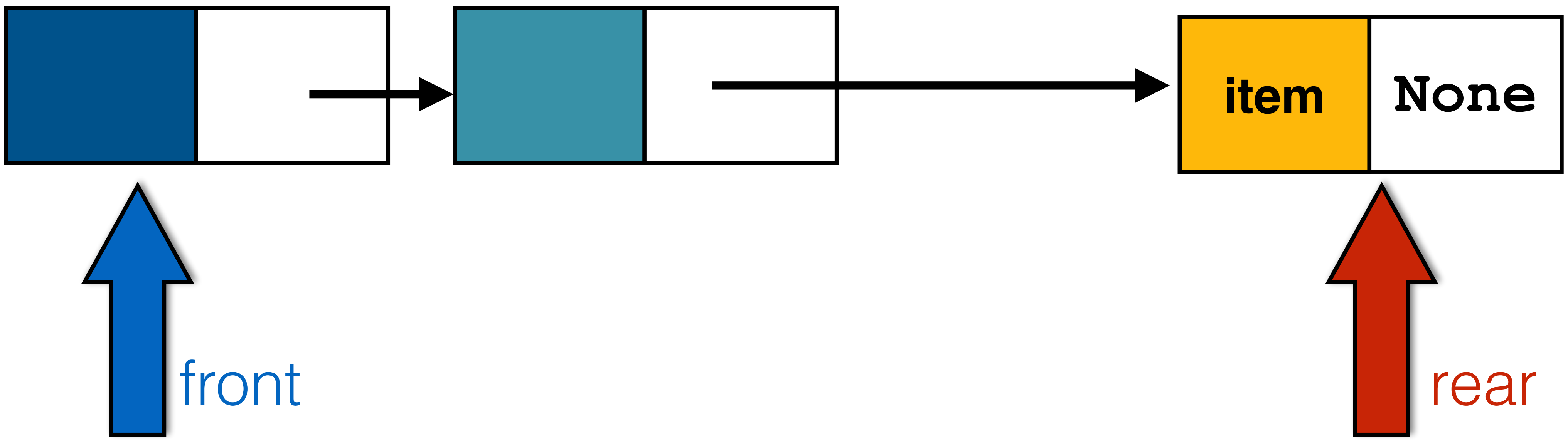


```python
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```

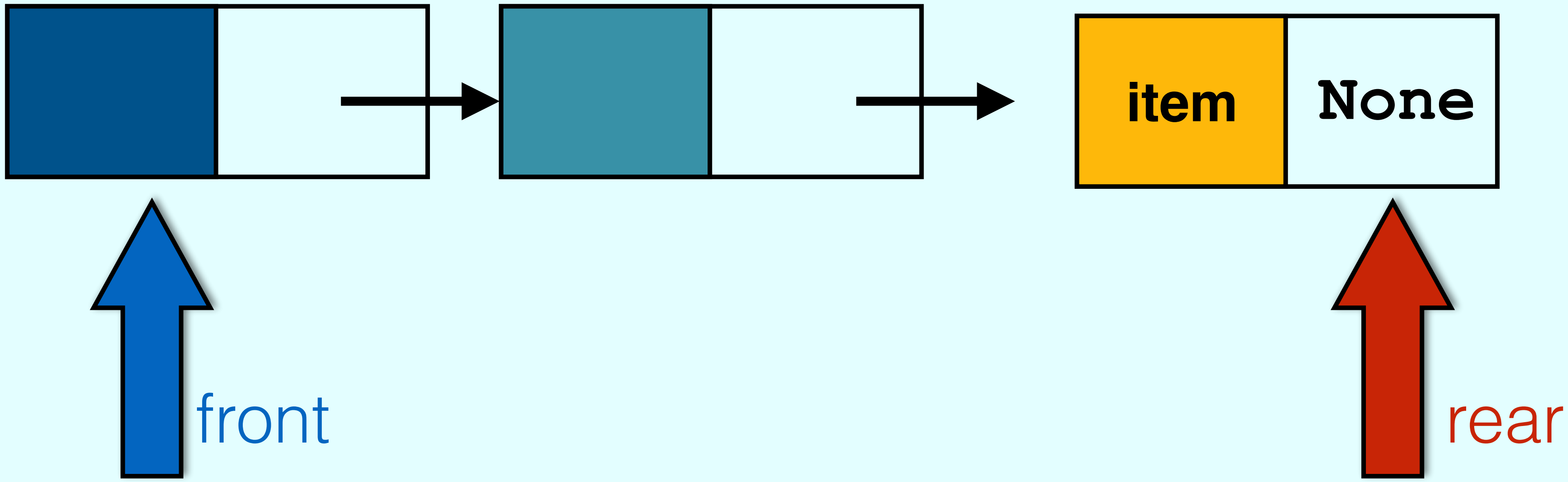




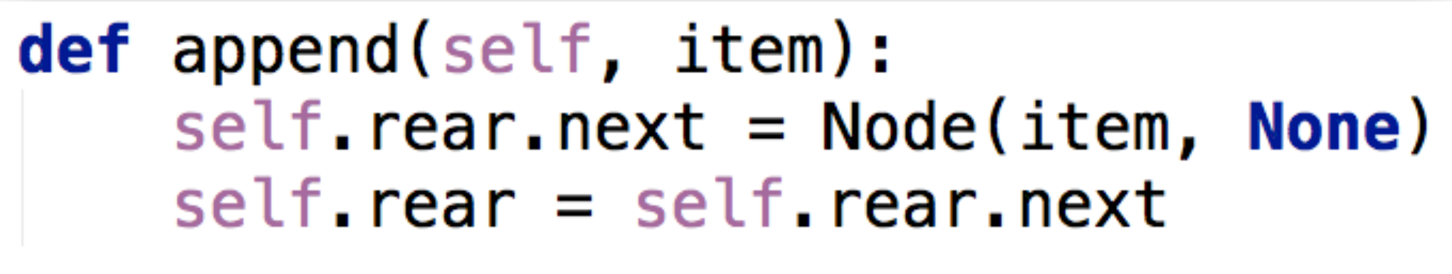
```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



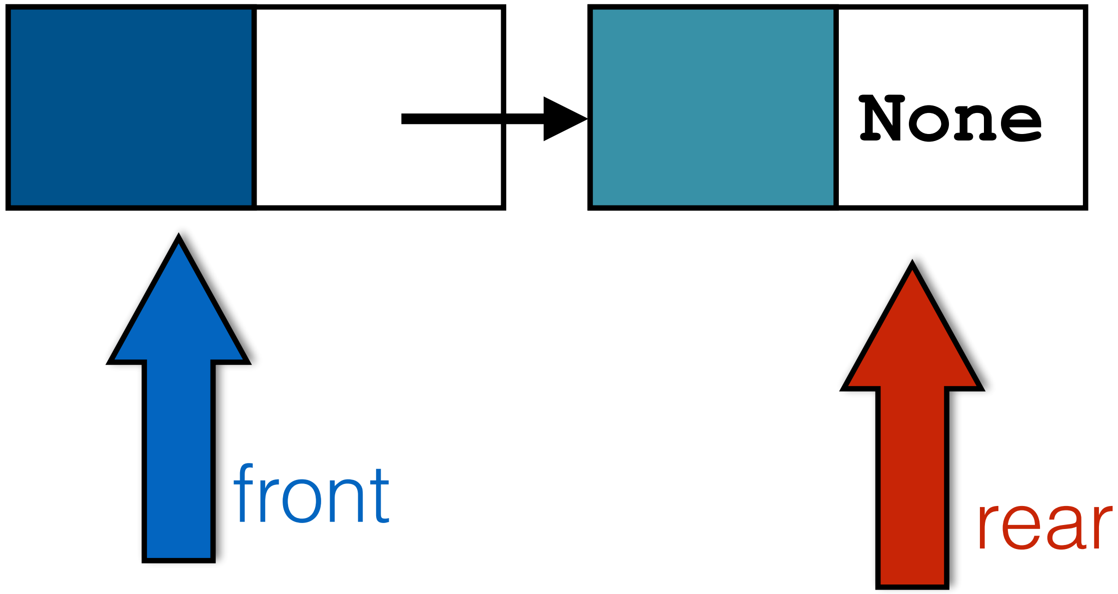
Goal

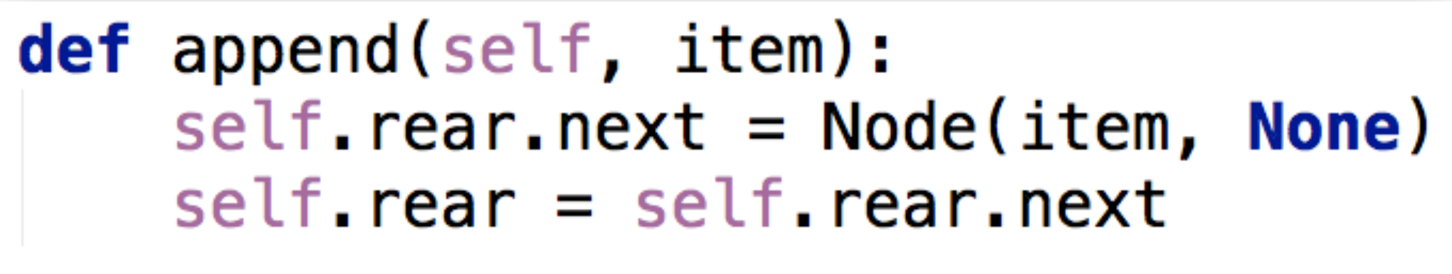


algorithm.

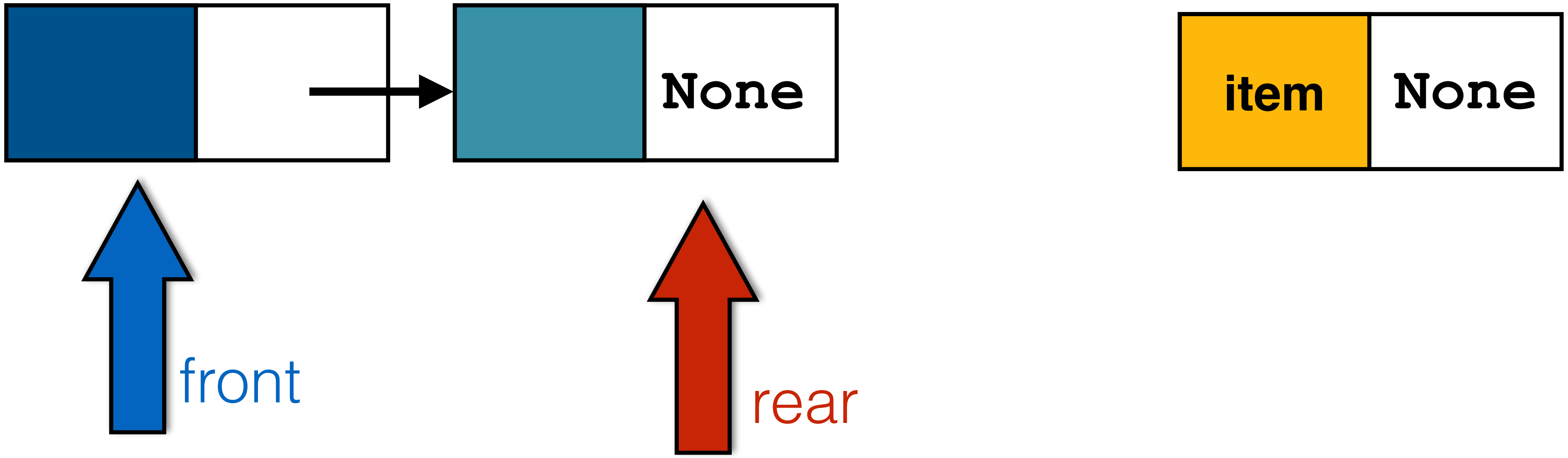


```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```

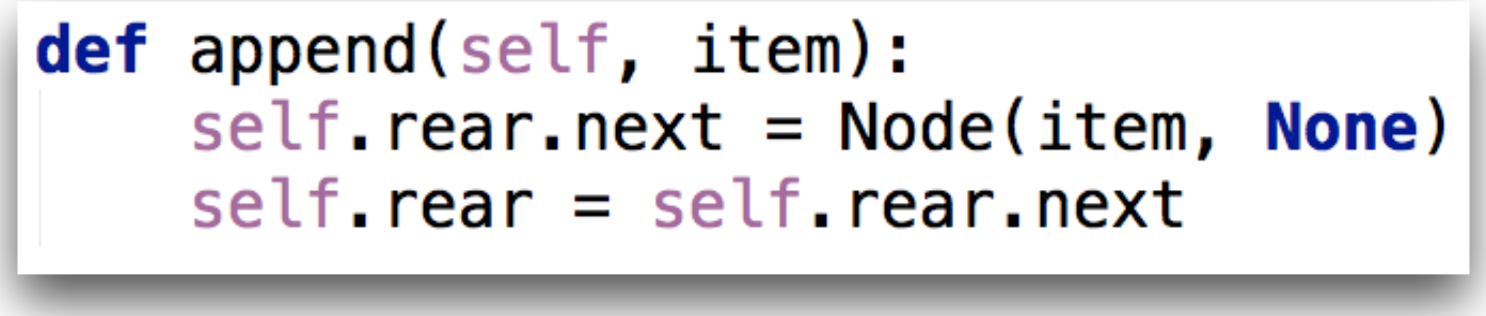




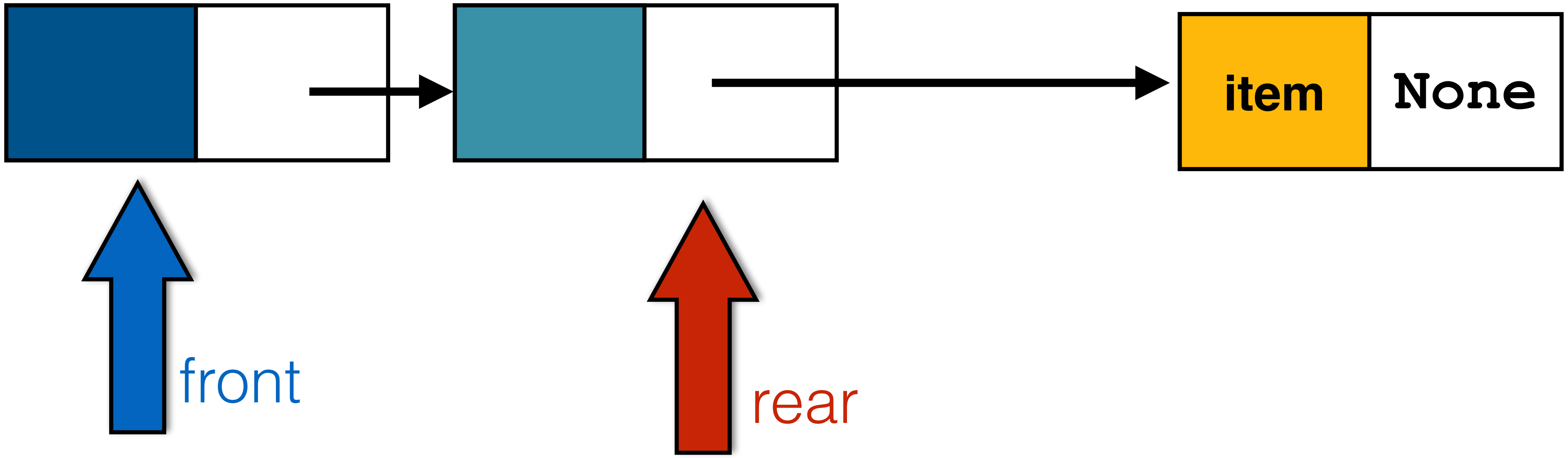
```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



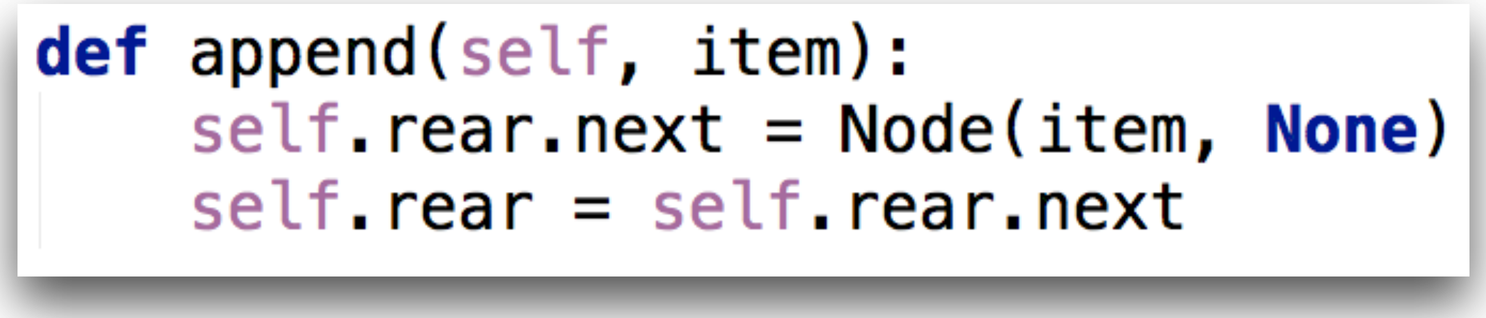
- Create a new node for item



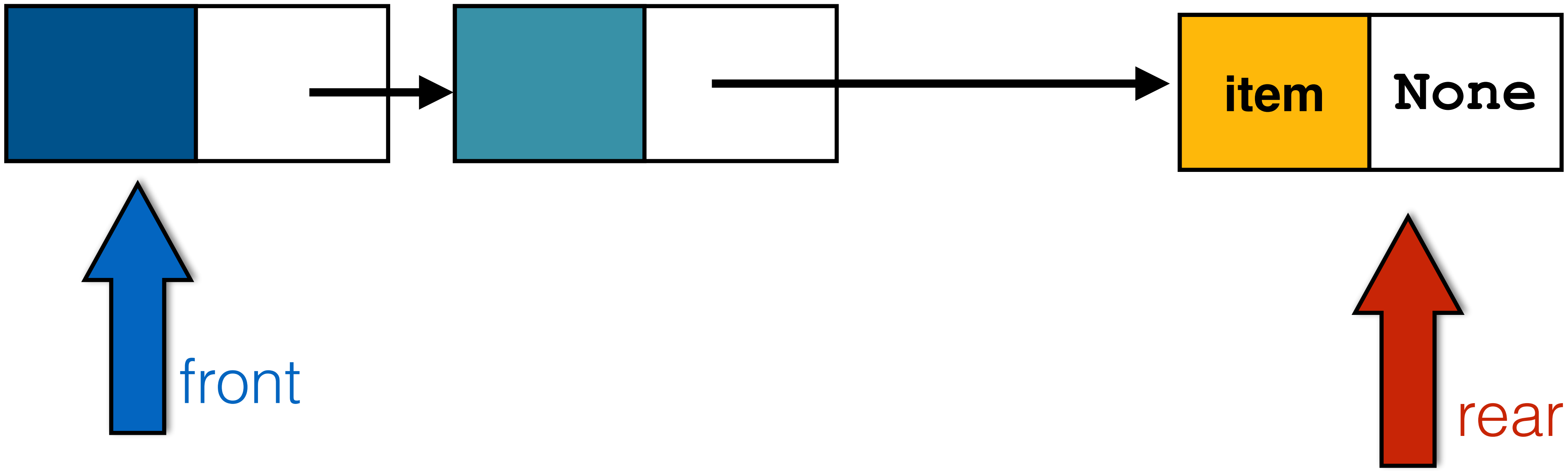
```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



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



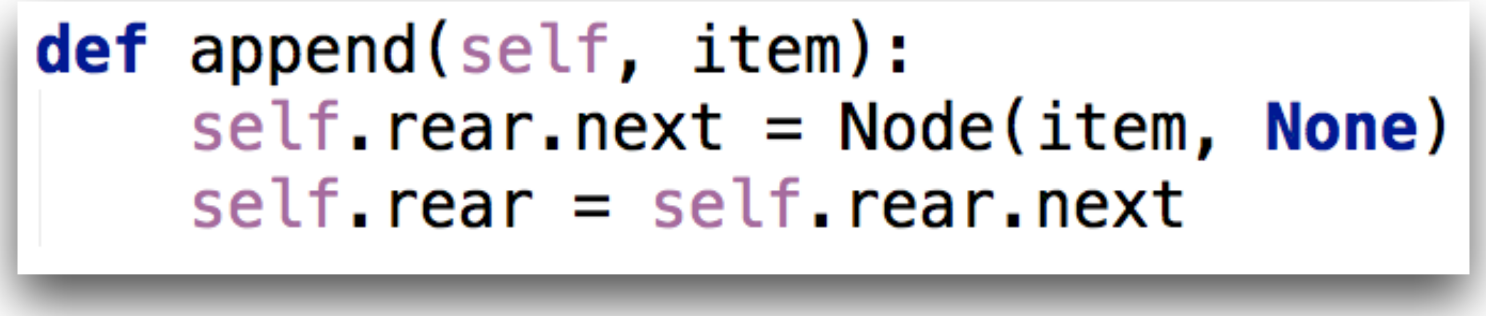
```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



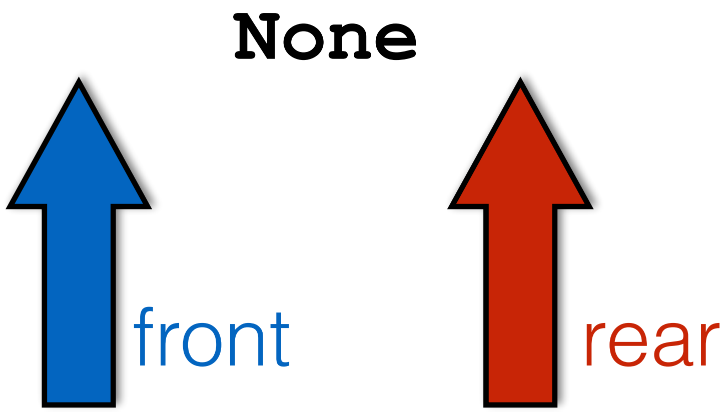
- Create a new node for item
- Make a link from current rear to new node
- The new node becomes the new rear

Looking good…

Boundary cases



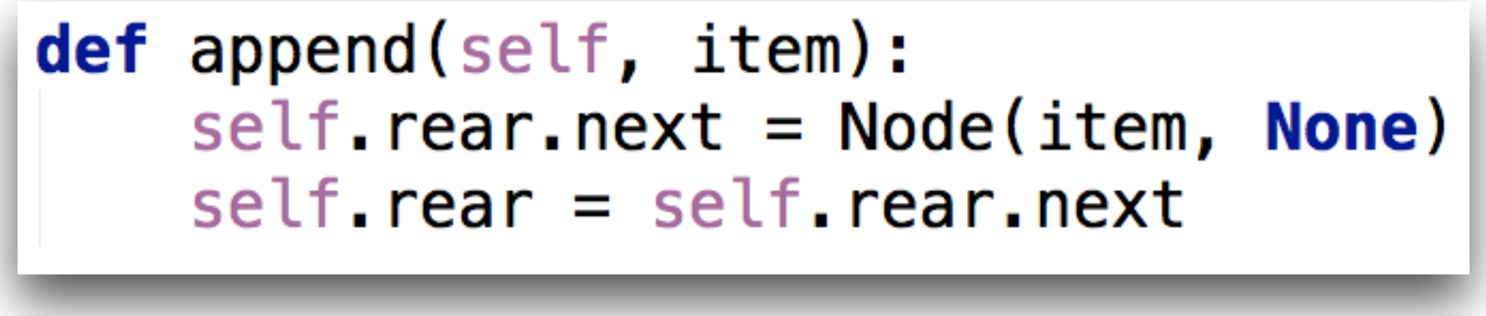
```
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```



None

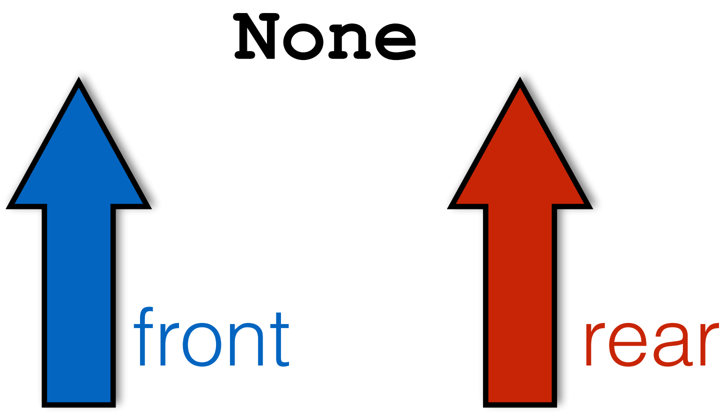
front

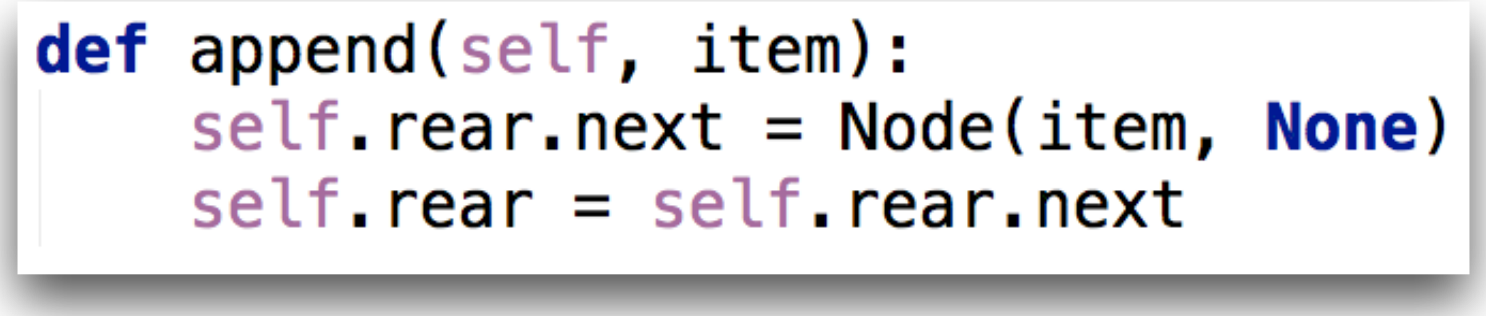
rear



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

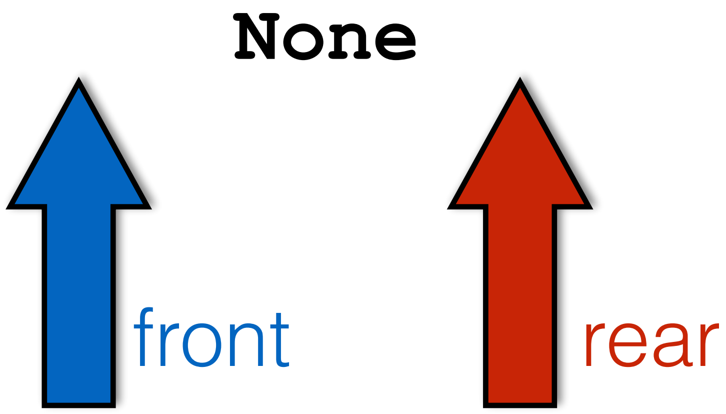
?



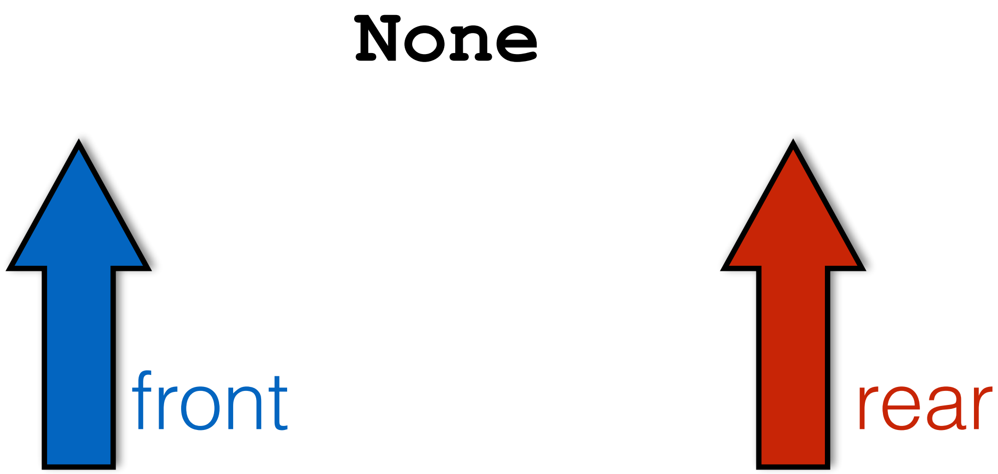


```python
def append(self, item):
    self.rear.next = Node(item, None)
    self.rear = self.rear.next
```

?



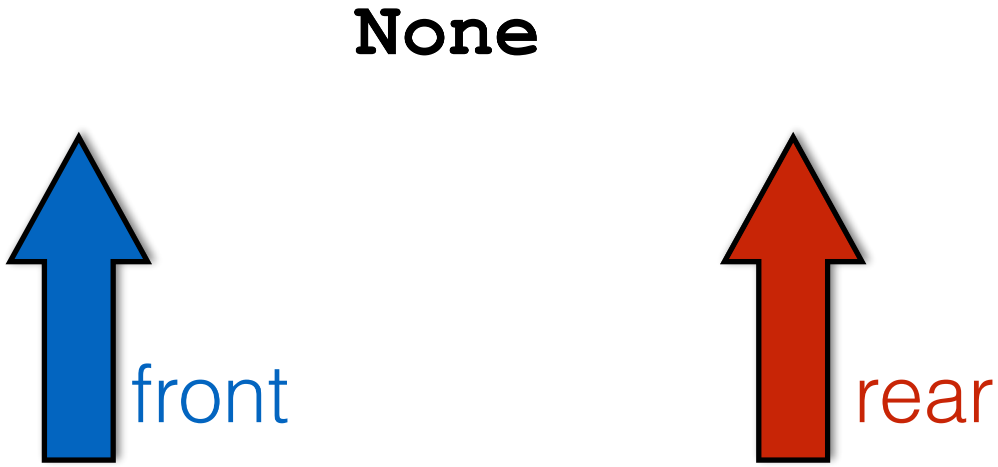
If the queue is empty
we need to do something with **front**



None

front

rear

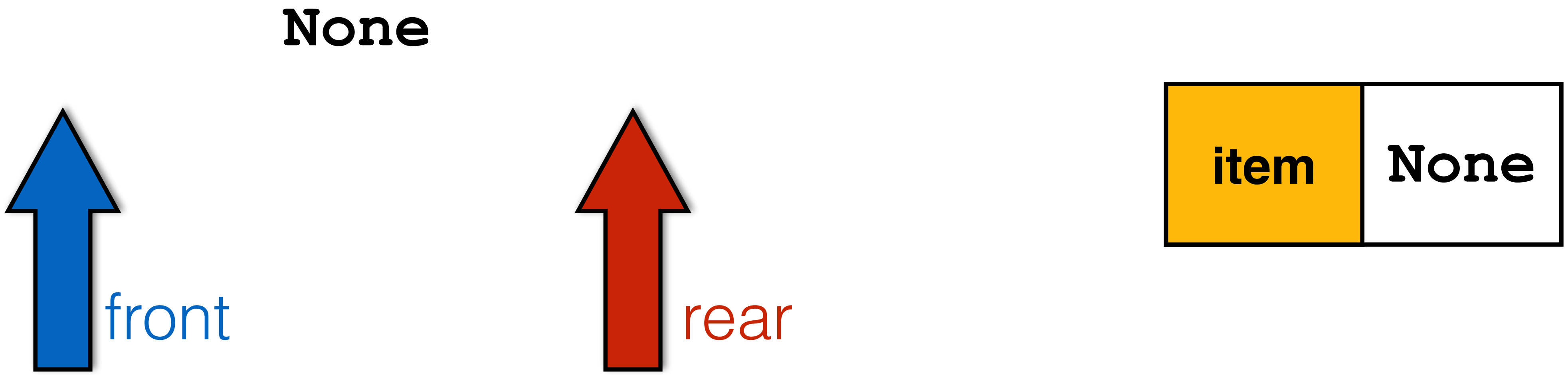


None

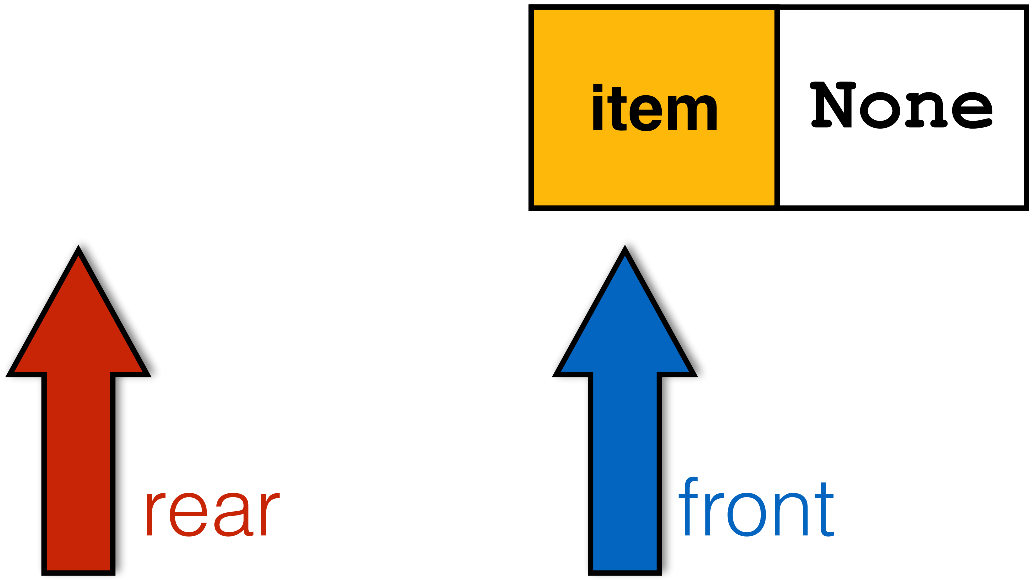
front

rear

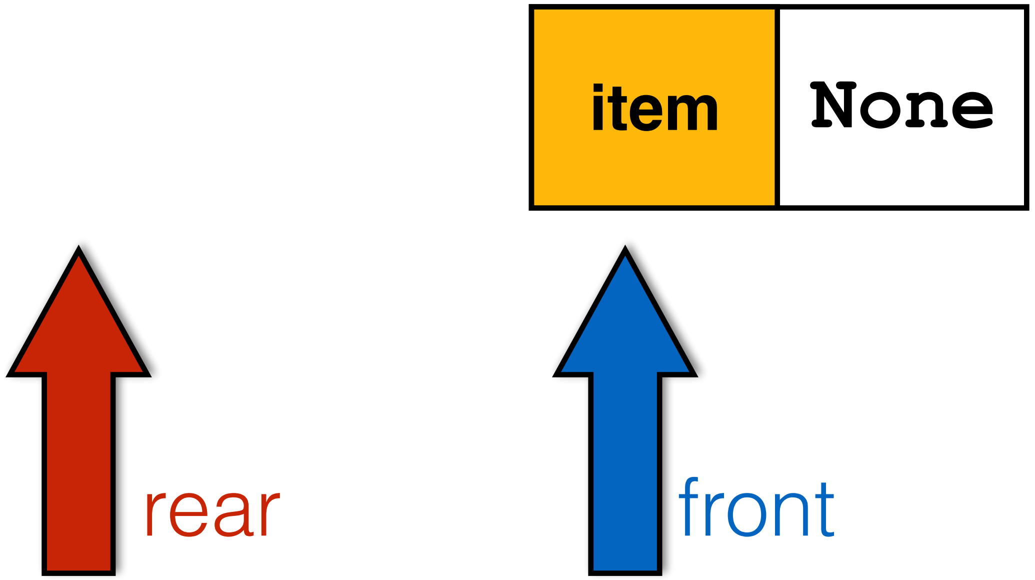
- Create a new node for item



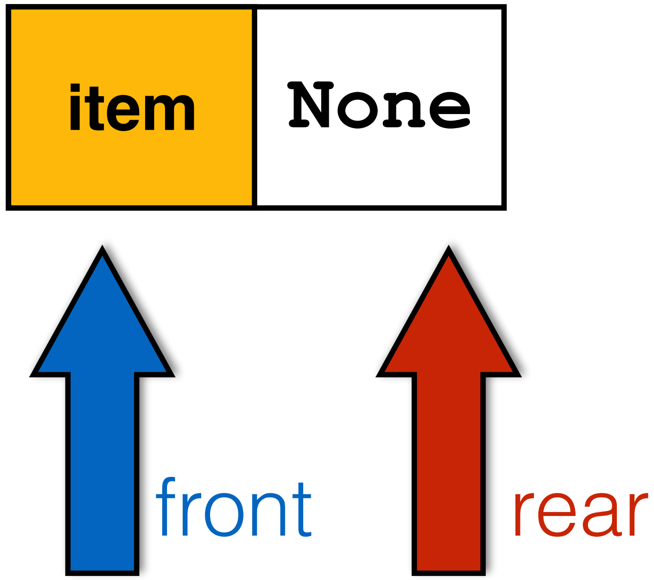
- Create a new node for item



- Create a new node for item
- If the queue is empty:
    - Make the new node be the front

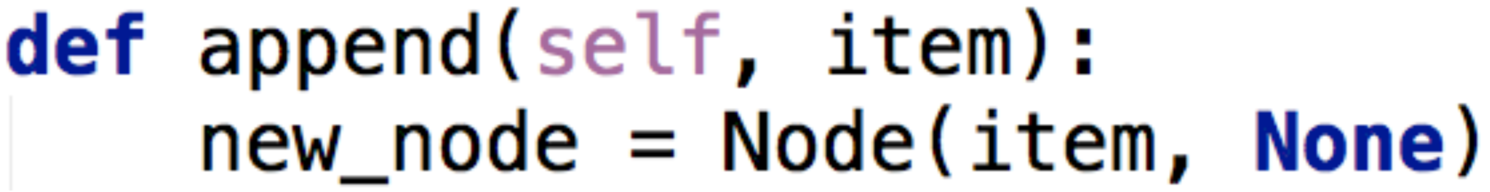


- Create a new node for item
- If the queue is empty:
  - Make the new node be the front
- If the queue is not empty:
  - Make a link from current rear to new node



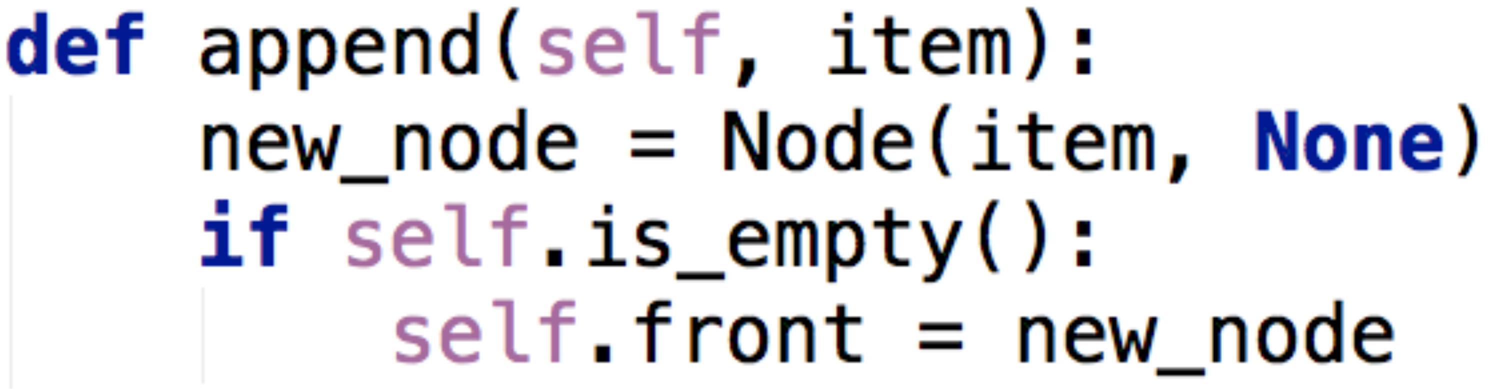
- Create a new node for item
- If the queue is empty:
    - Make the new node be the front
- If the queue is not empty:
    - Make a link from current rear to new node
- The new node becomes the new rear

```python
def append(self, item):
```



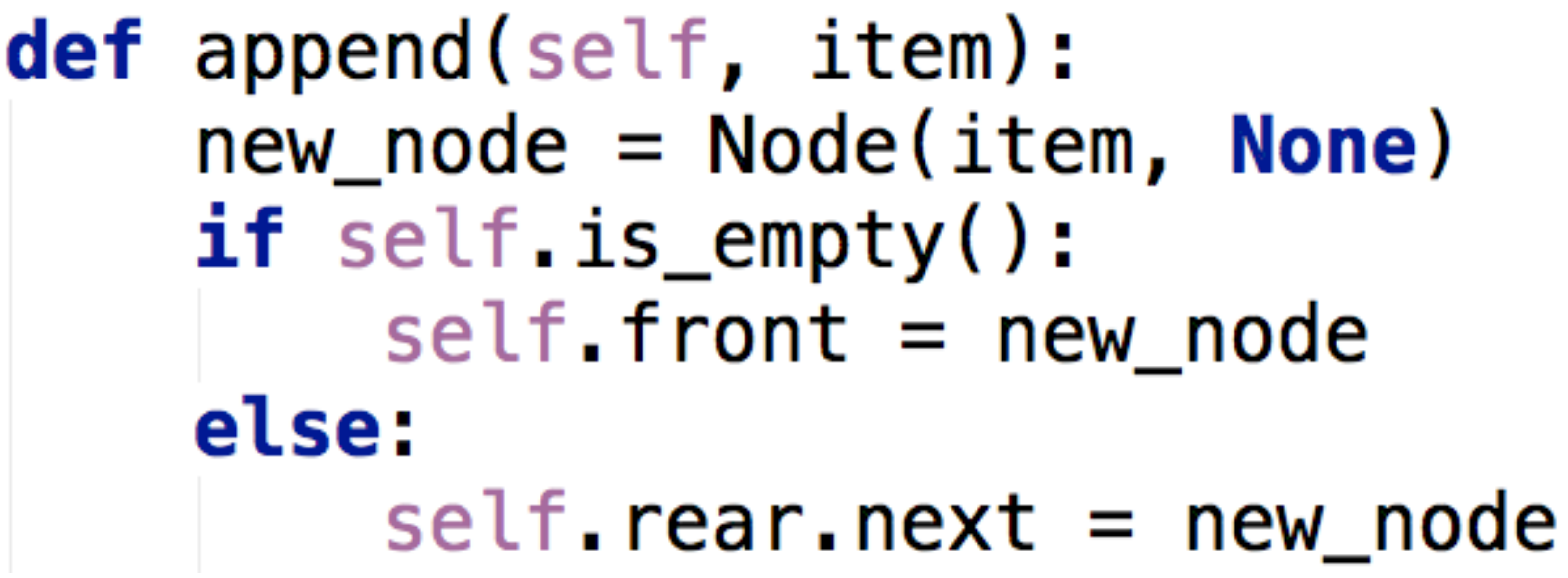
```
def append(self, item):
    new_node = Node(item, None)
```

- Create a new node for item



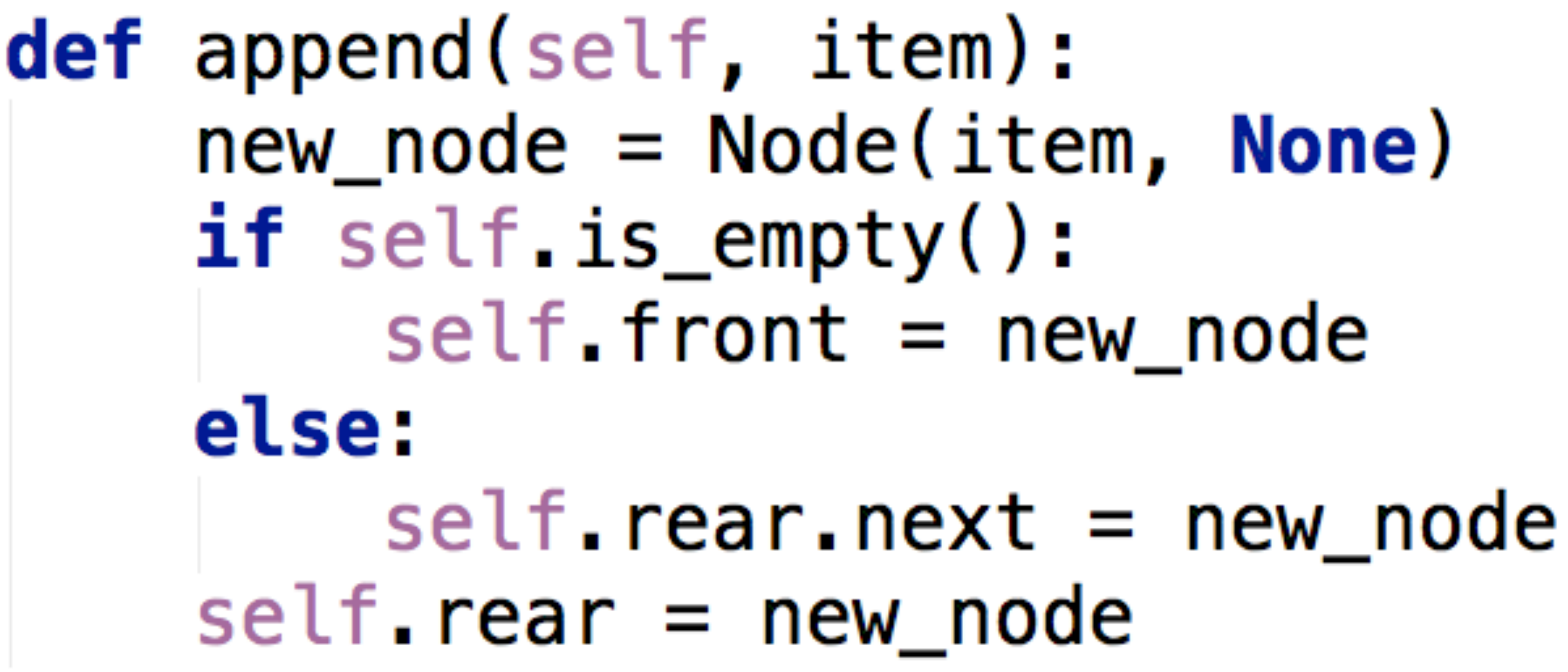
```python
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
```

- Create a new node for item
- If the queue is empty:
    - Make the new node be the front



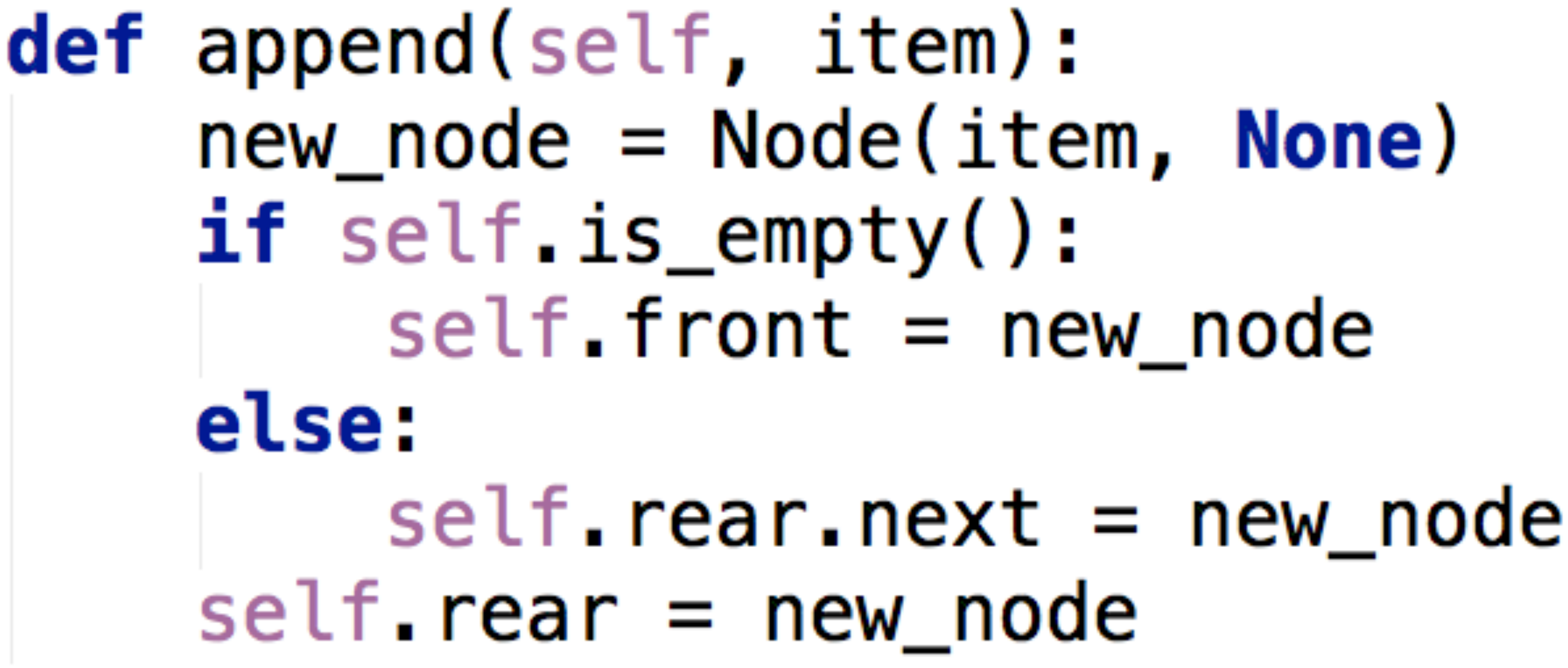
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
```

- 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

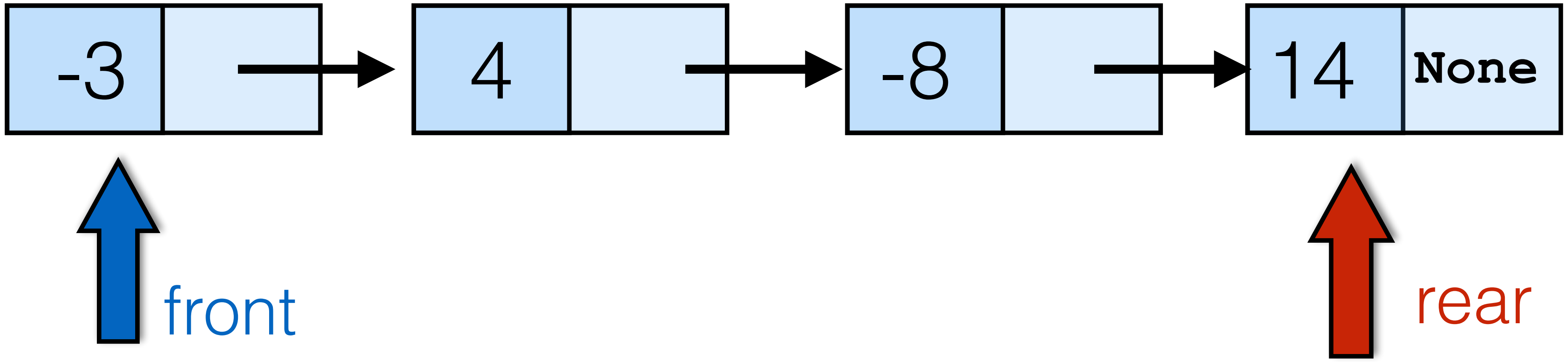


```python
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

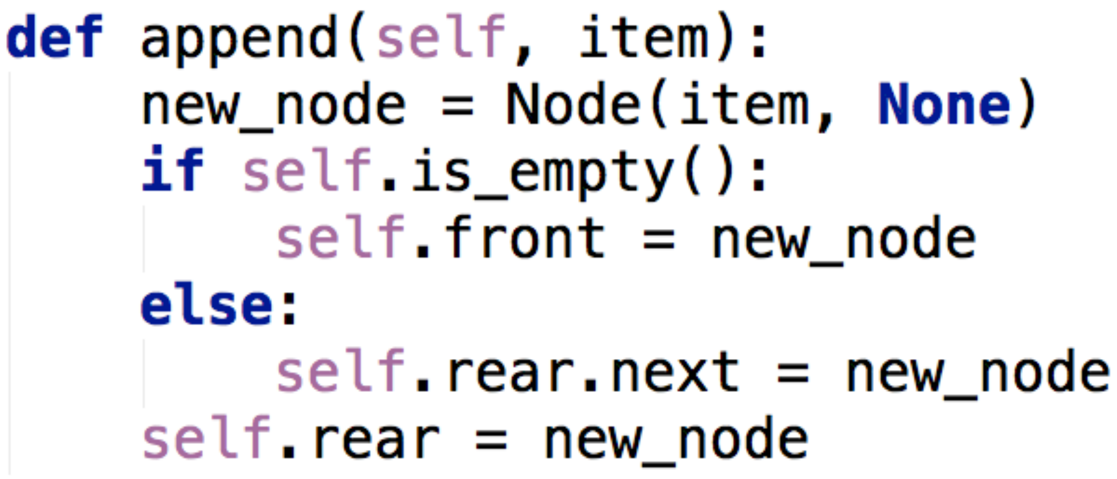
- Create a new node for item
- If the queue is empty:
    - Make the new node be the front
- If the queue is not empty:
    - Make a link from current rear to new node
- The new node becomes the new rear



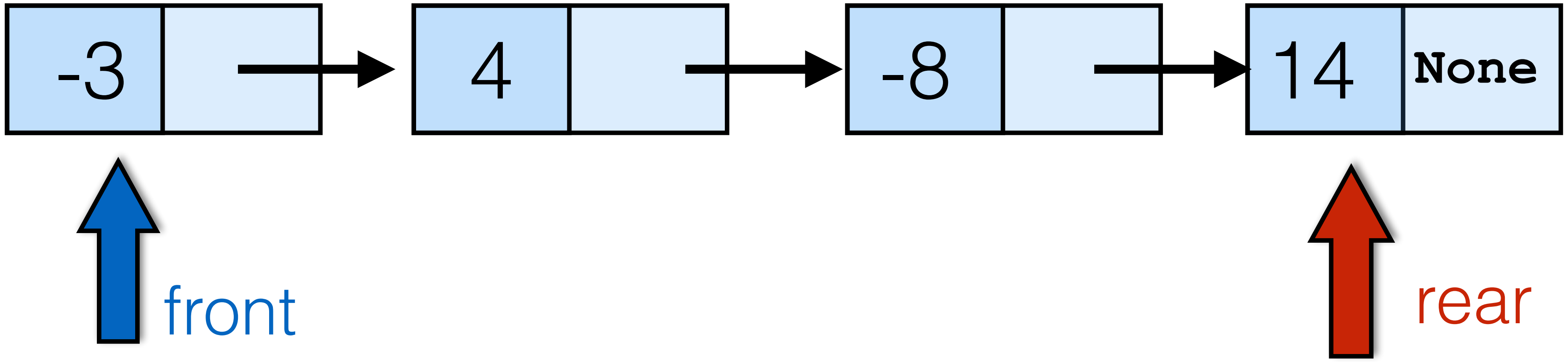
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



q.front.item = -3
q.rear.item = 14

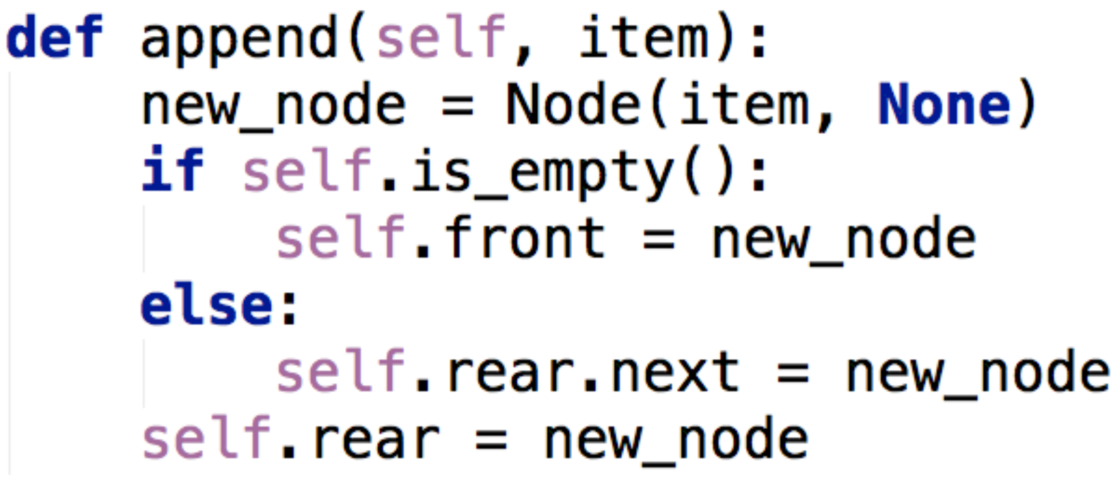


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

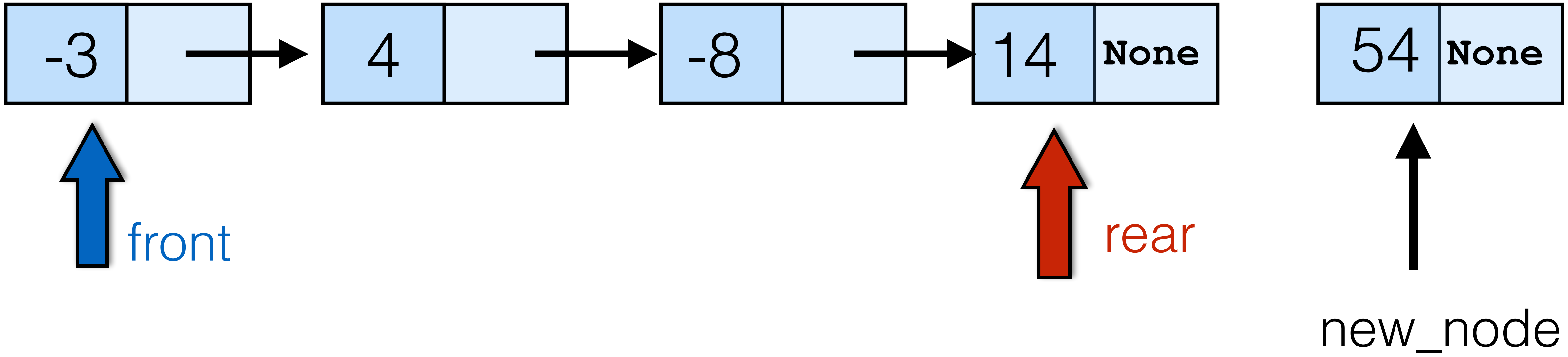


```
q.front.item = -3
q.rear.item = 14
```

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

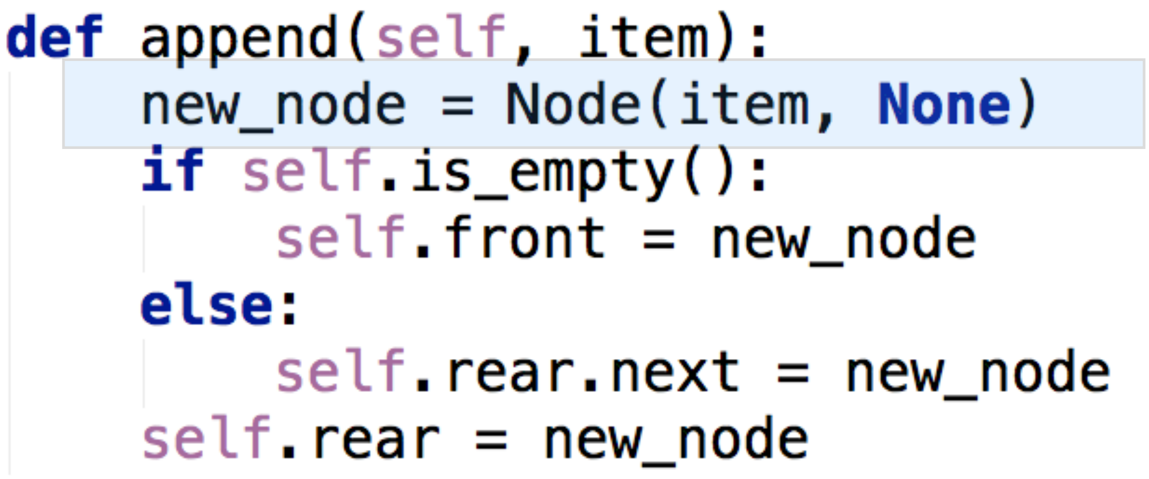


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

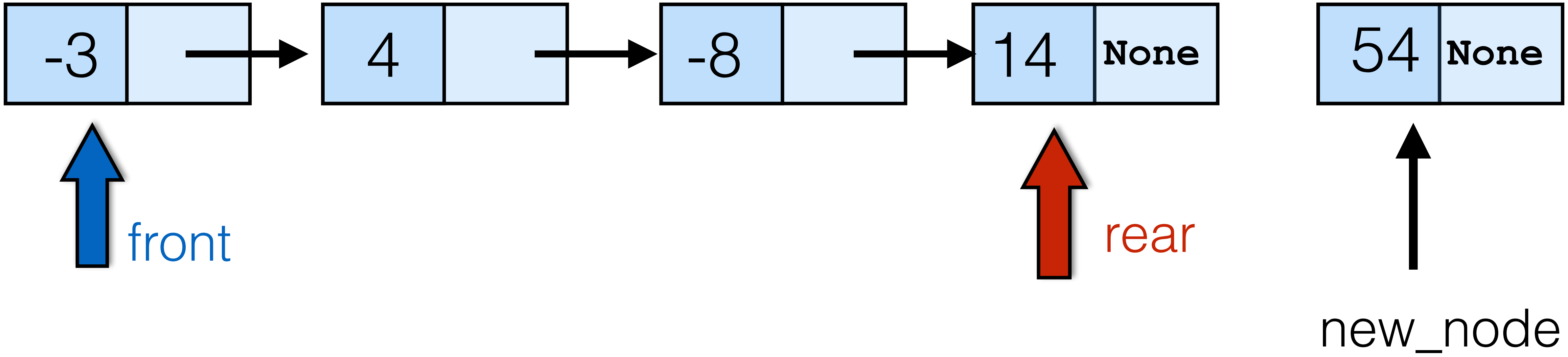


```
q.front.item = -3
q.rear.item = 14
```

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

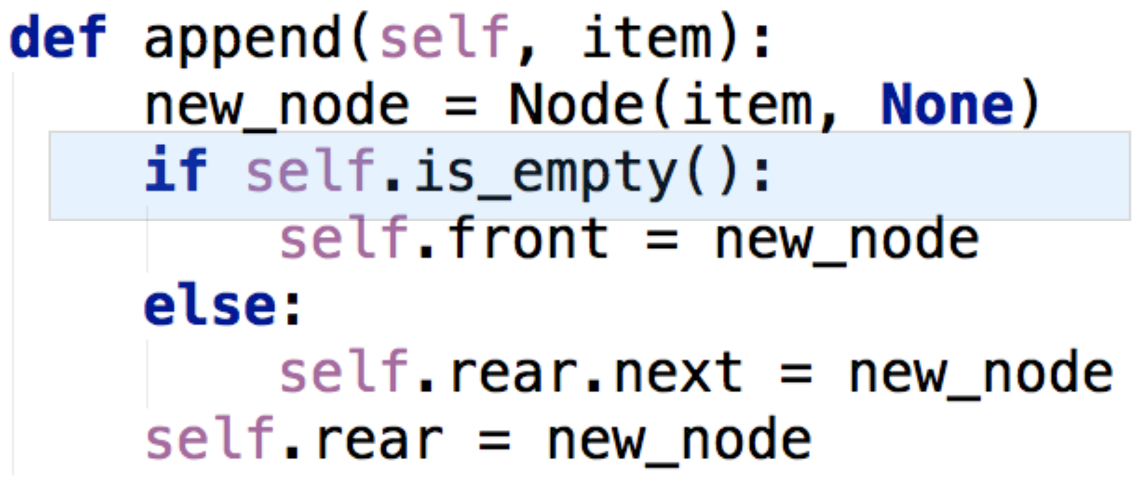


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

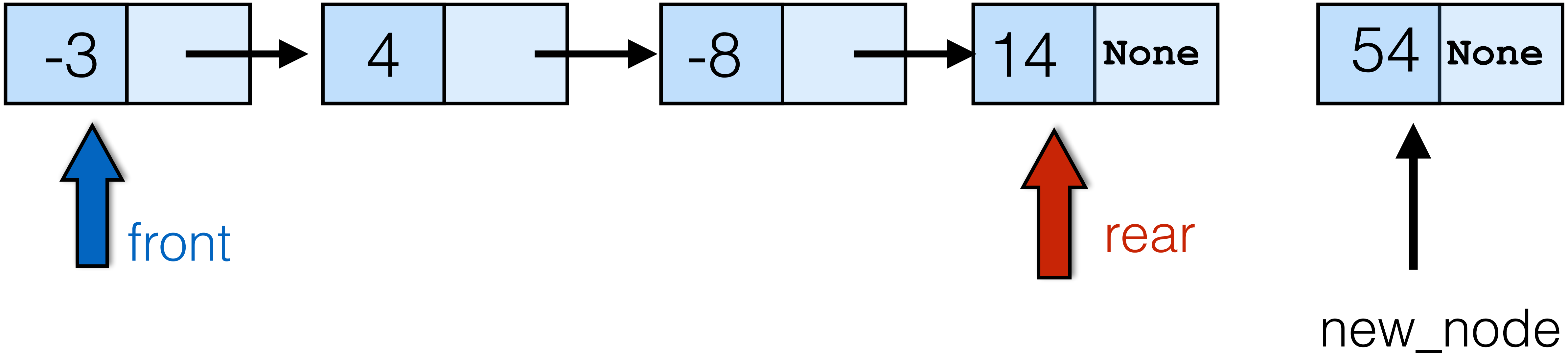


```
q.front.item = -3
q.rear.item = 14
```

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

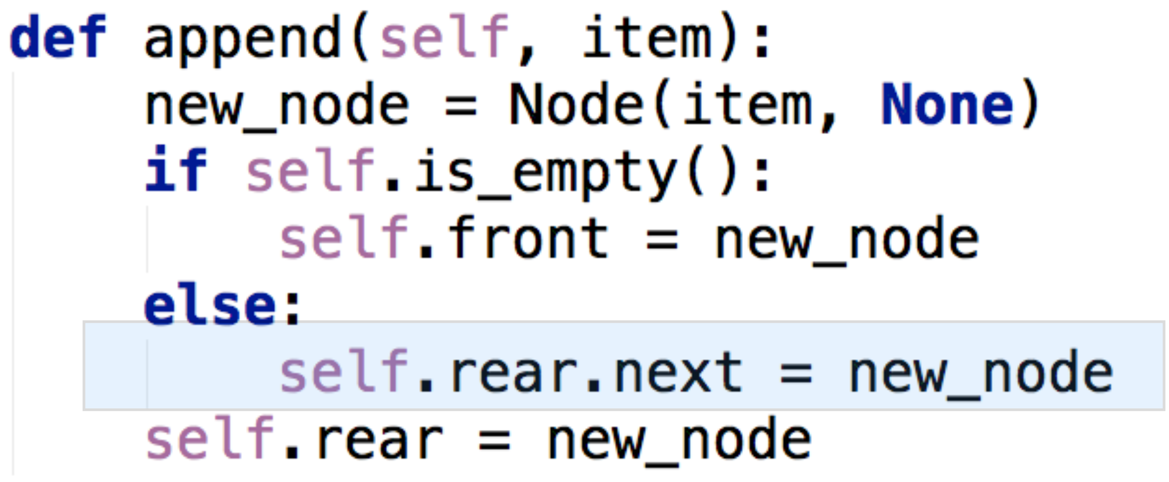


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

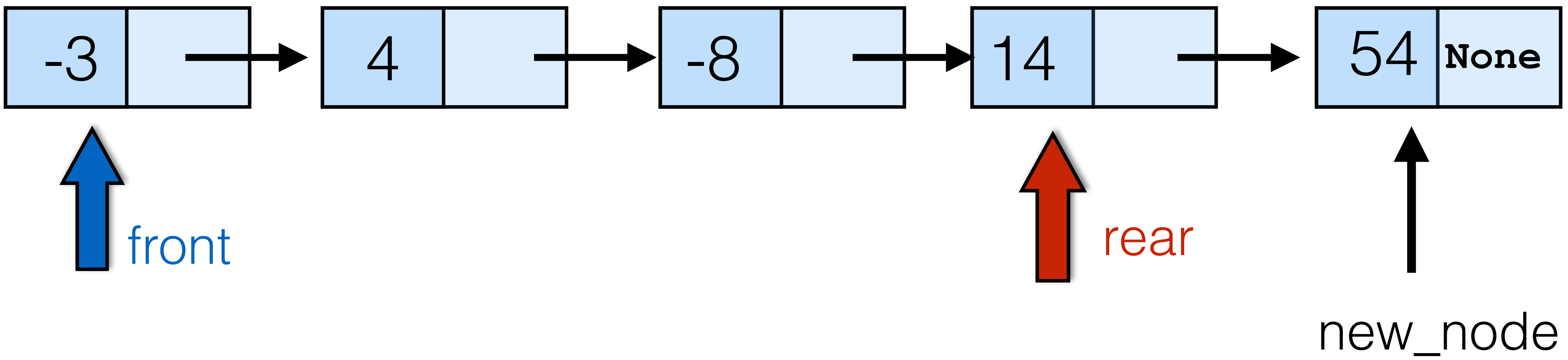


```
q.front.item = -3
q.rear.item = 14
```

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

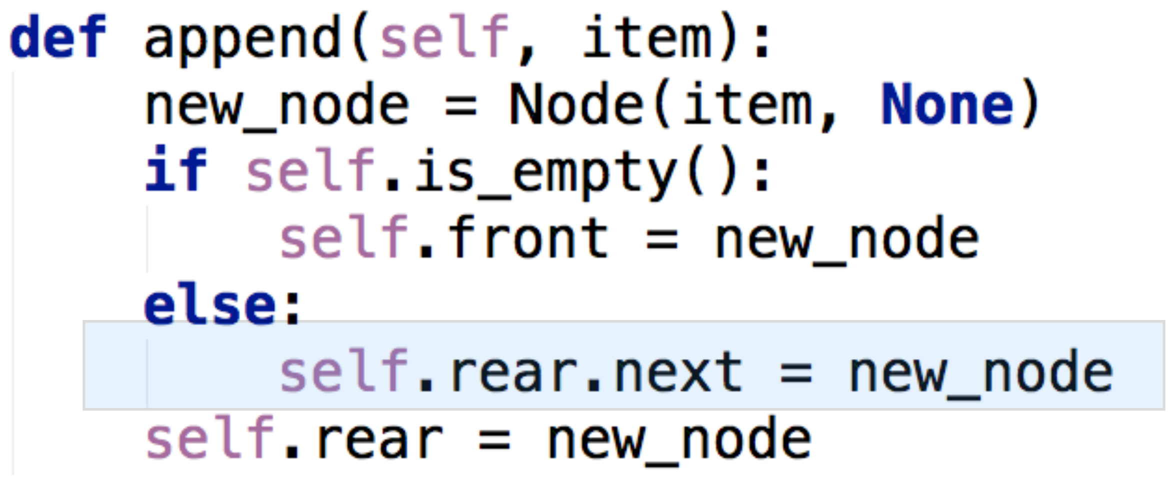


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

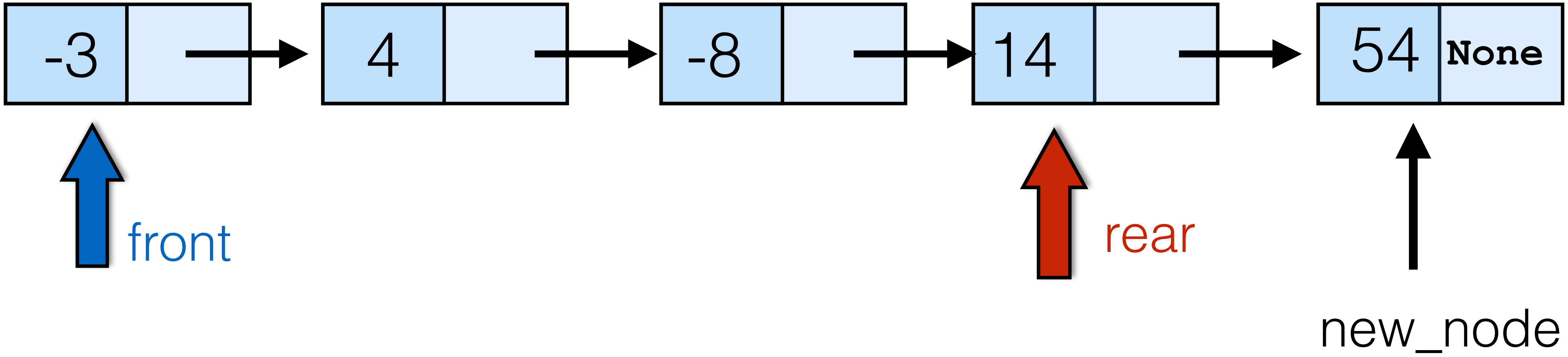


```
q.front.item = -3
 q.rear.item = 14
```

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

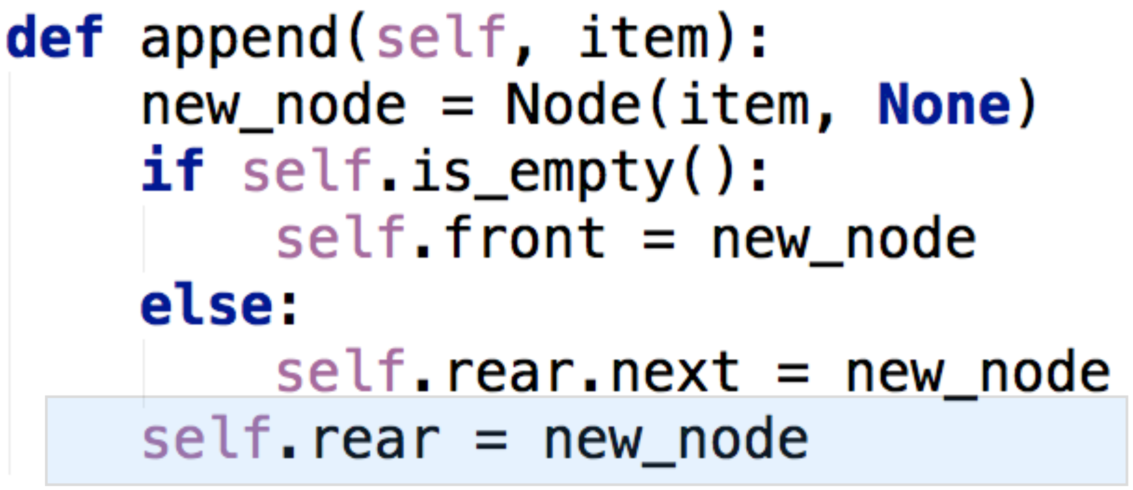


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

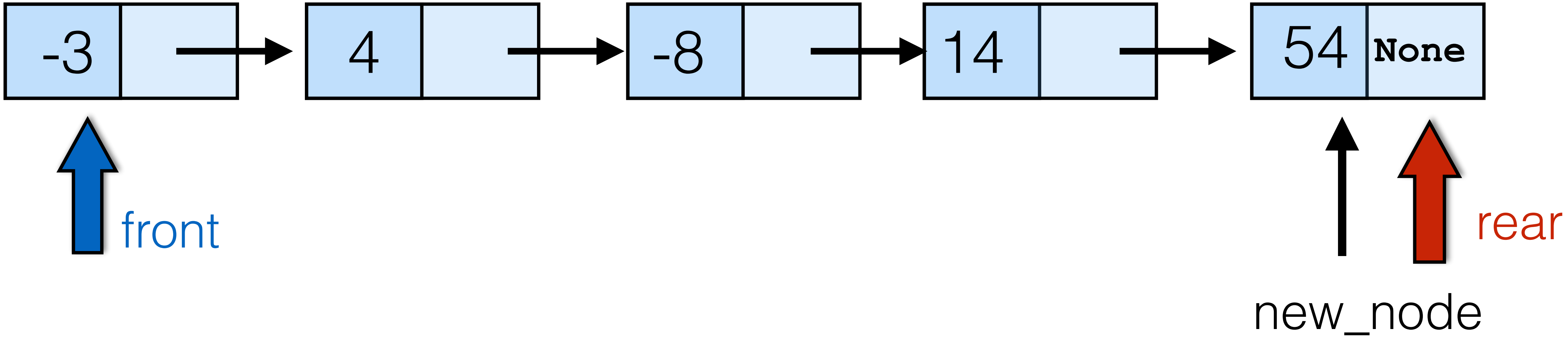


```
q.front.item = -3
q.rear.item = 14
```

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

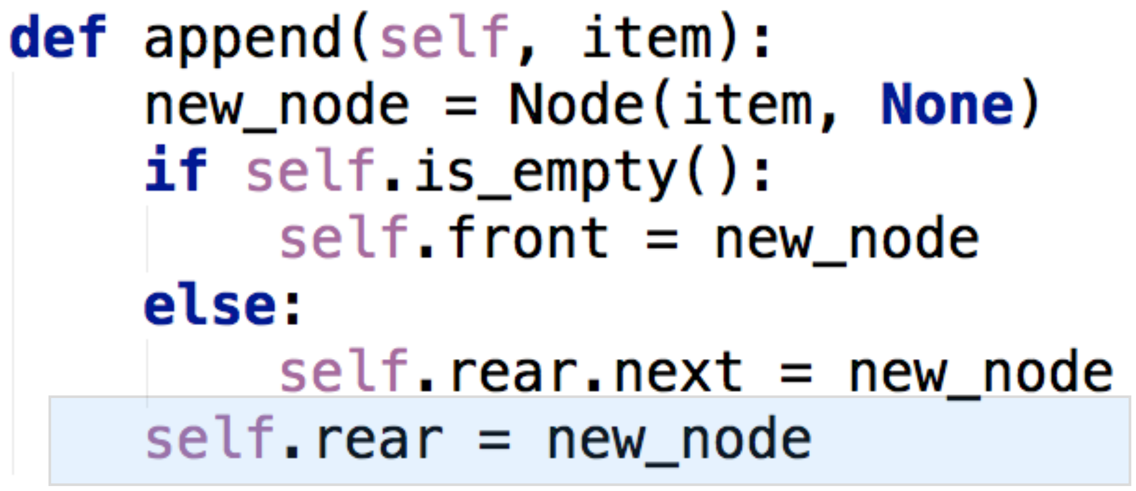


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

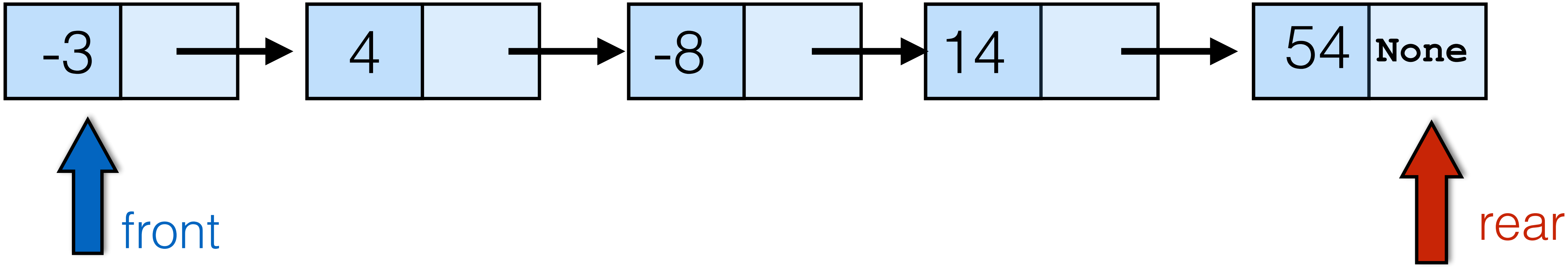


```
q.front.item = -3
q.rear.item = 14
```

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

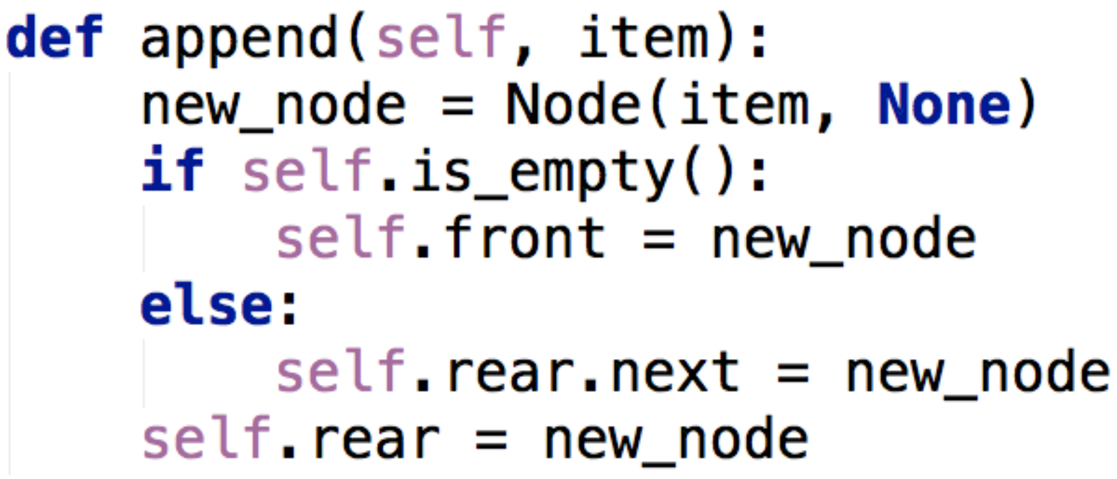


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

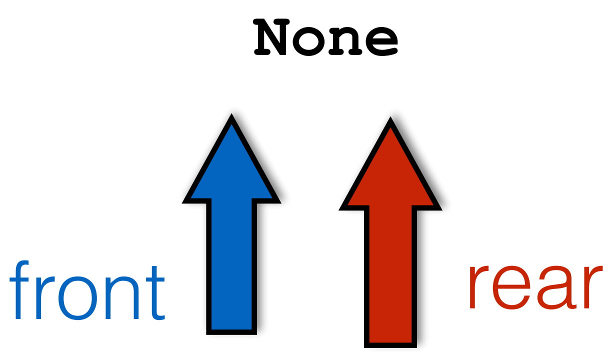


q.front.item = -3
q.rear.item = 14

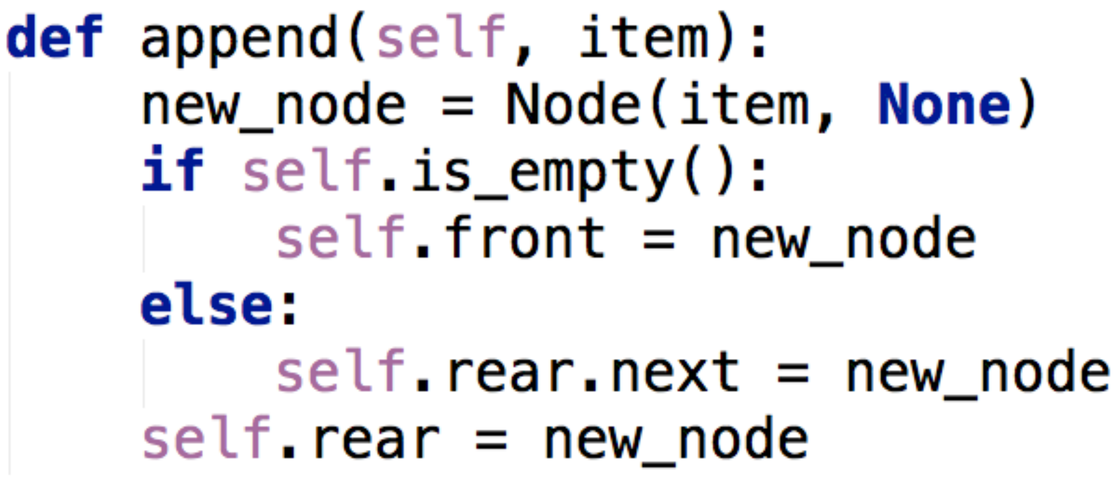
q.append(54)



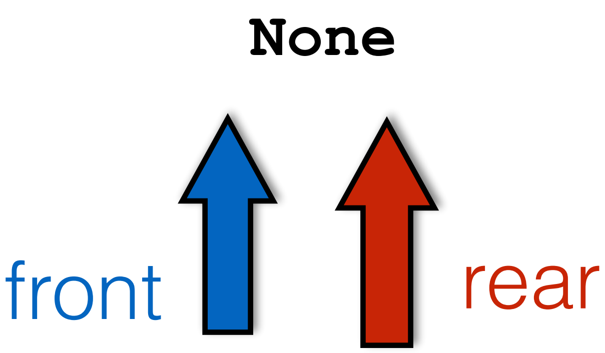
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



q.front = None
q.rear = None

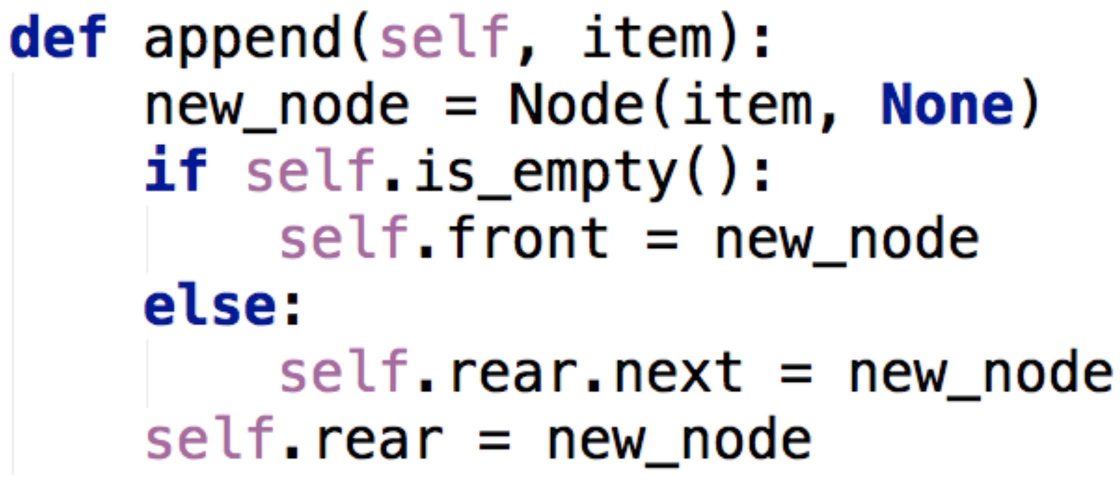


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

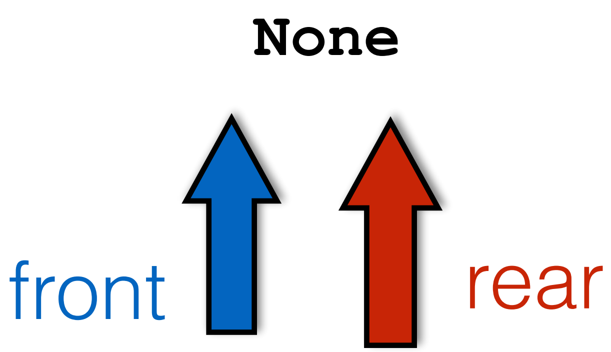


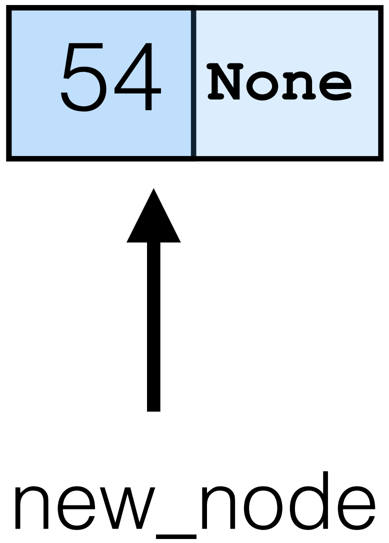
```
q.front = None
q.rear = None
```

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



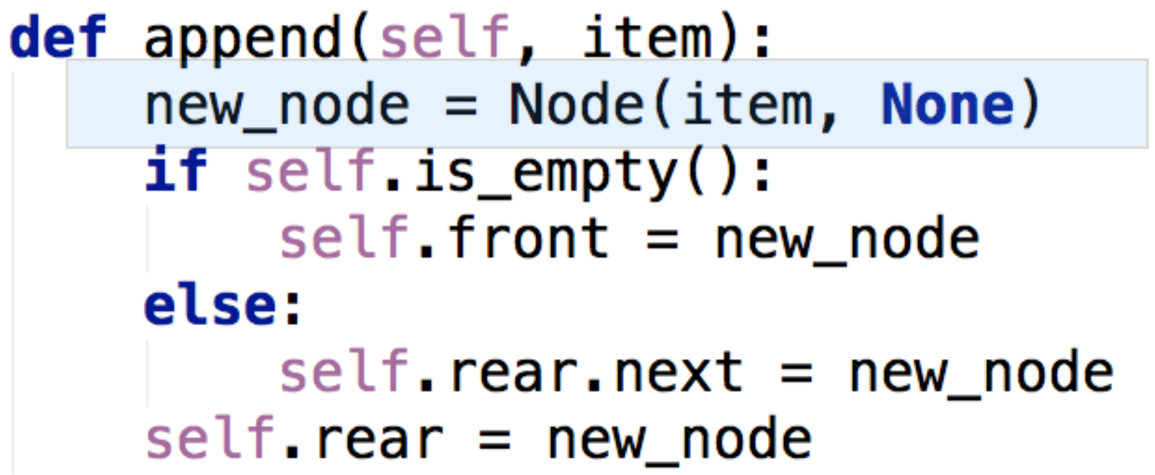
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



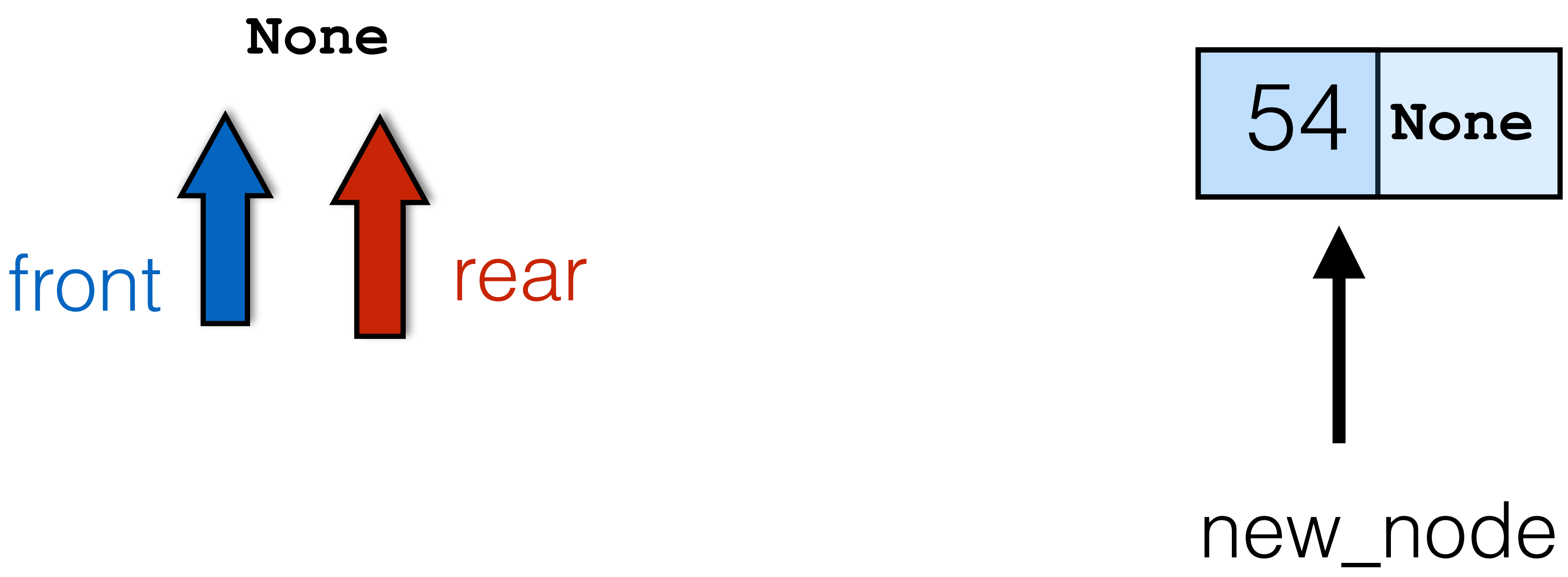


```
q.front = None
q.rear = None
```

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

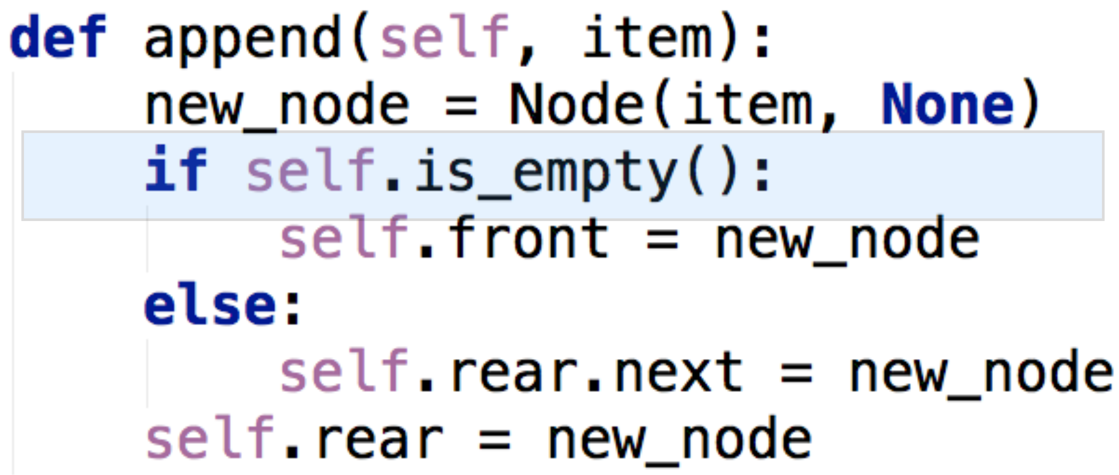


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

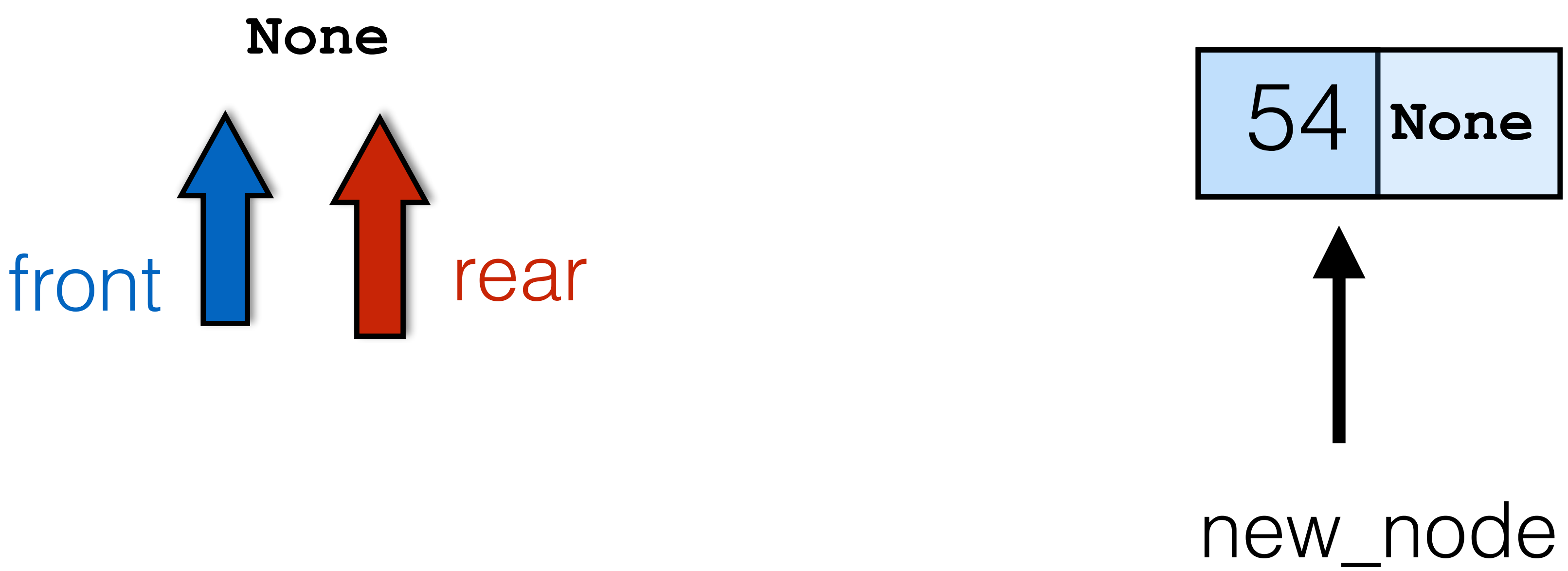


```
q.front = None
q.rear = None
```

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

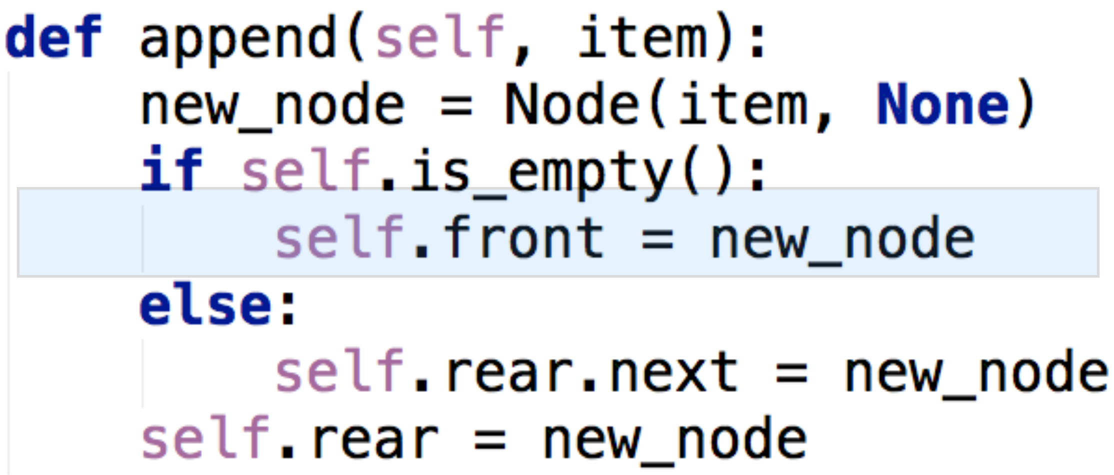


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

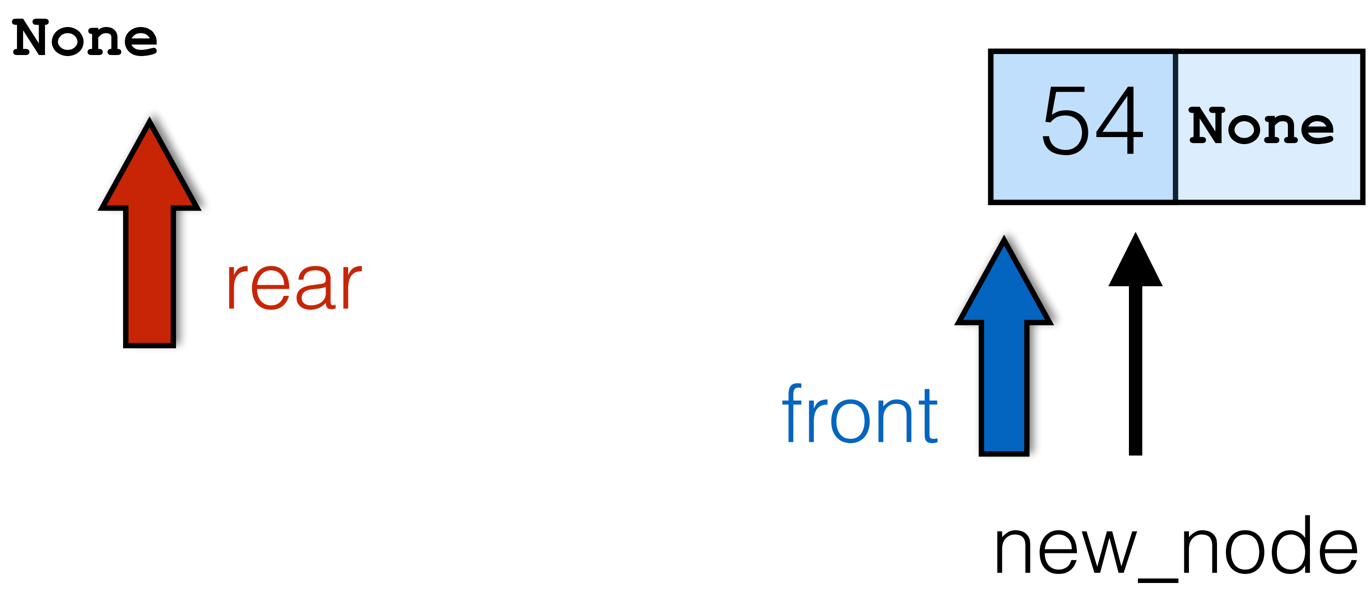


```
q.front = None
q.rear = None
```

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

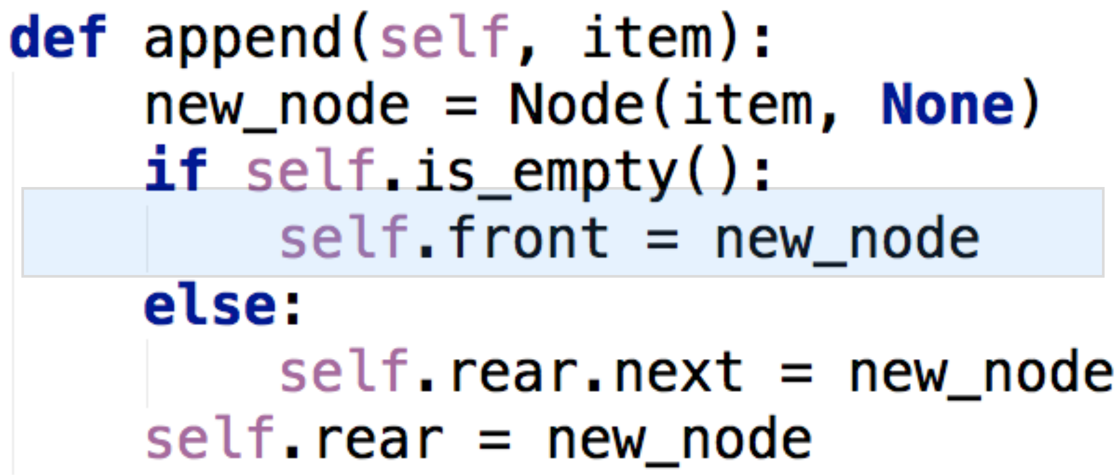


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

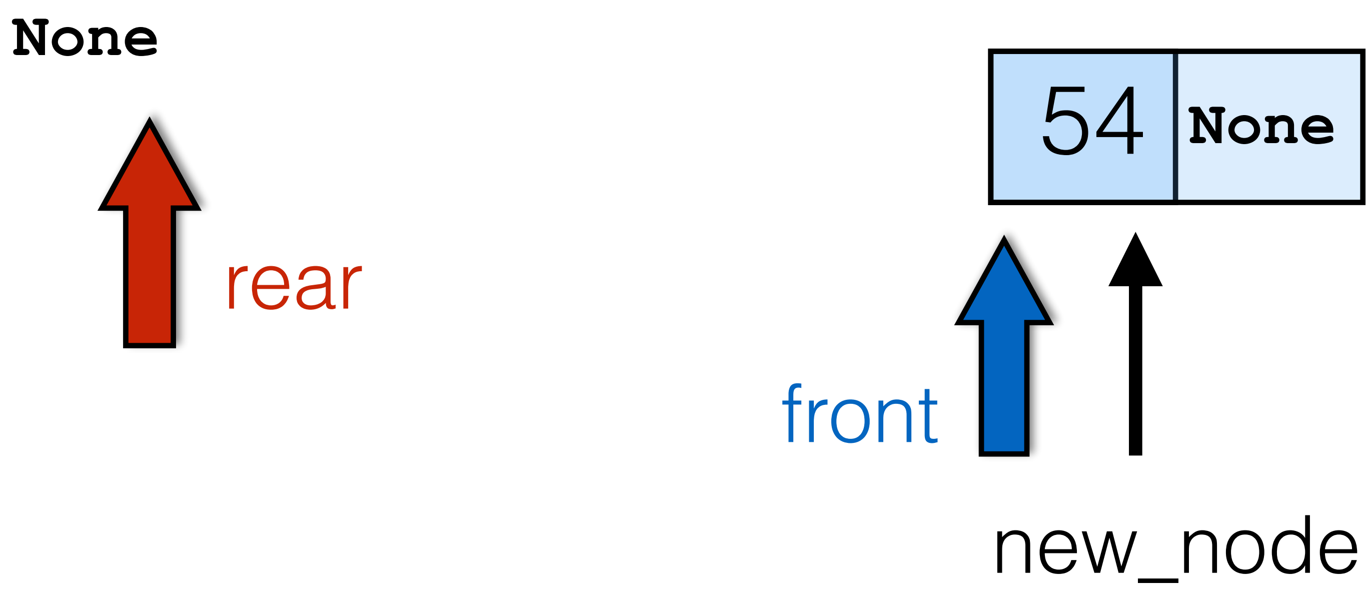


```
q.front = None
q.rear = None
```

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

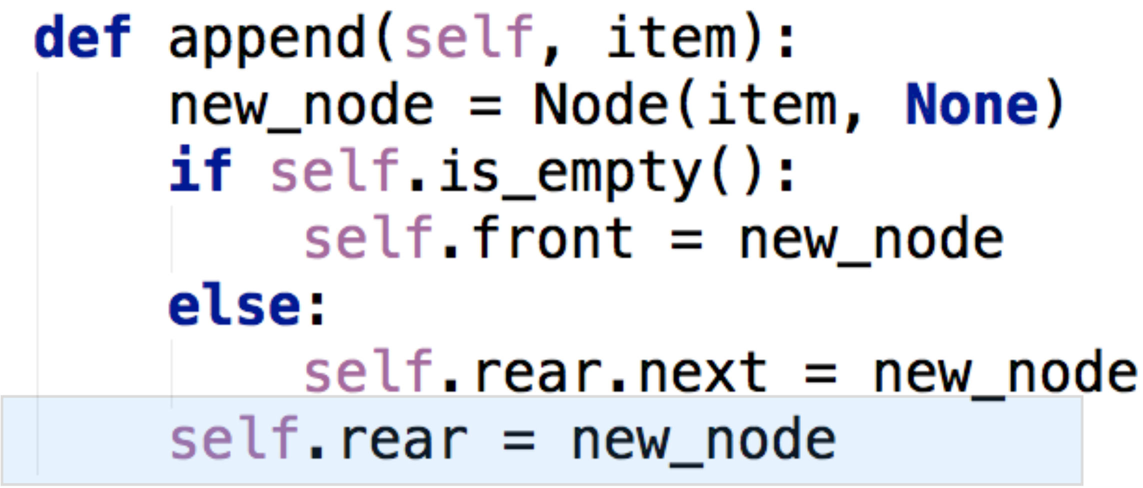


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

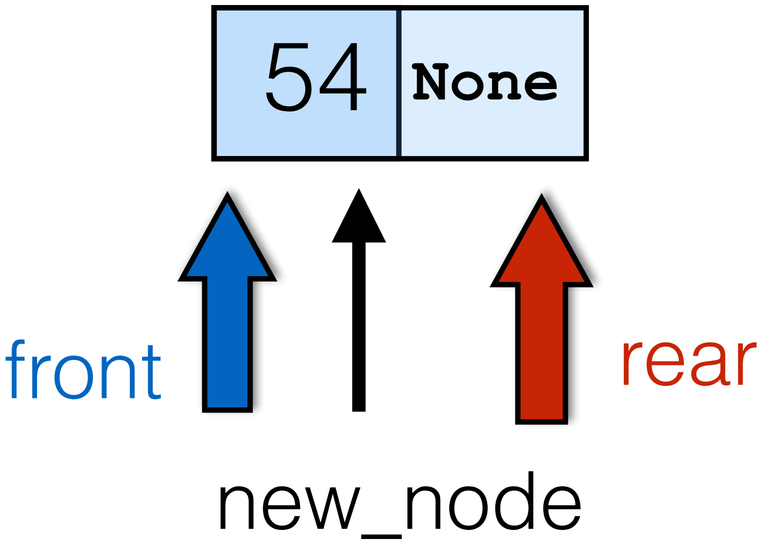


```
q.front = None
q.rear = None
```

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

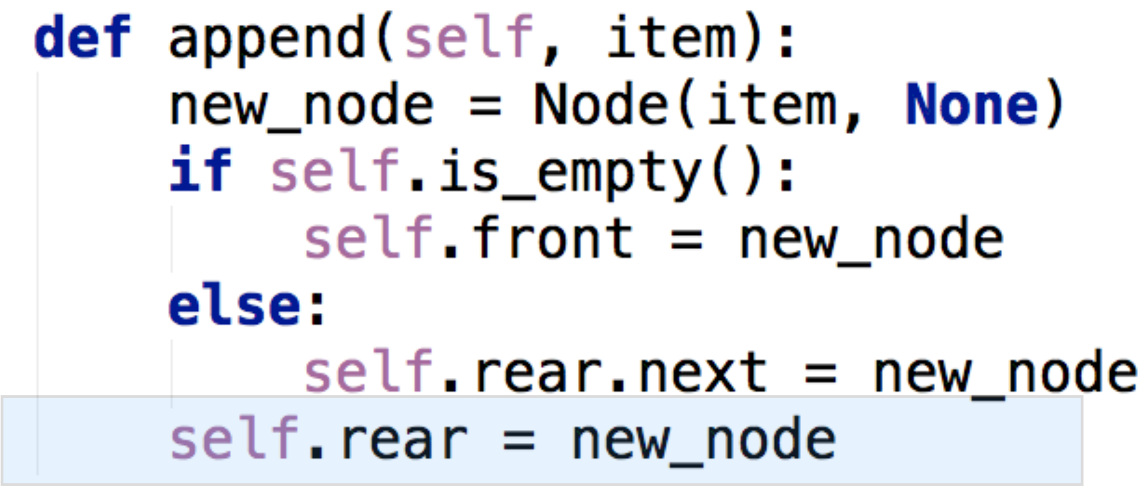


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

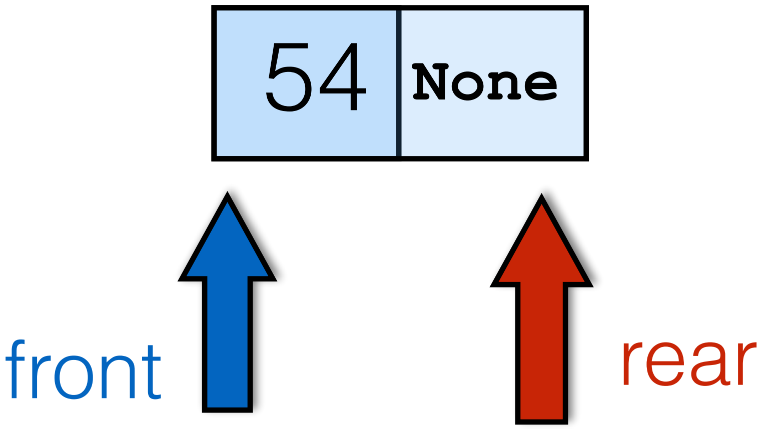


```
q.front = None
q.rear = None
```

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

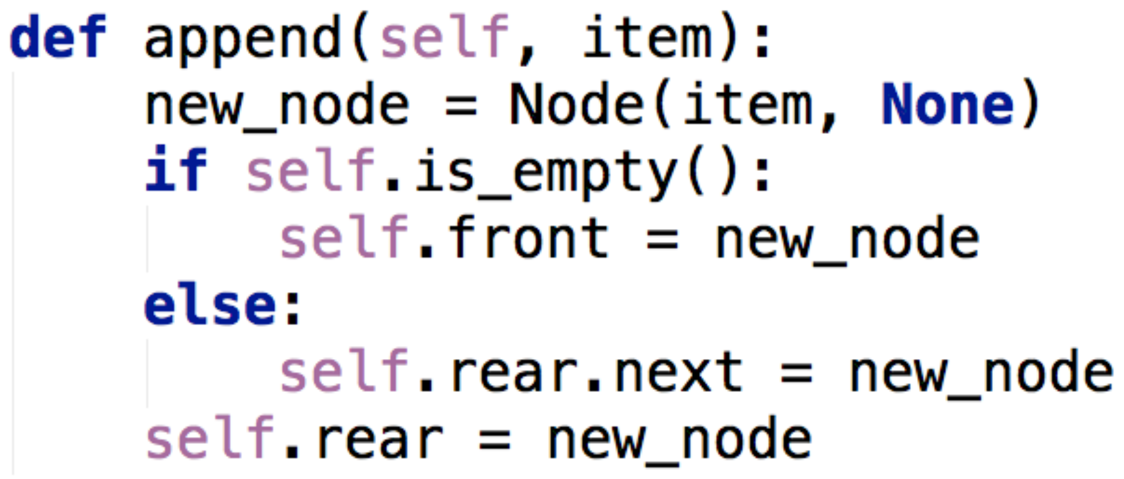


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

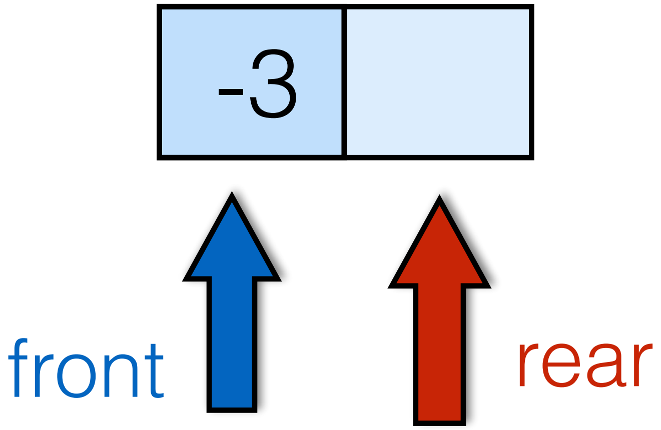


```
q.front = None
q.rear = None
```

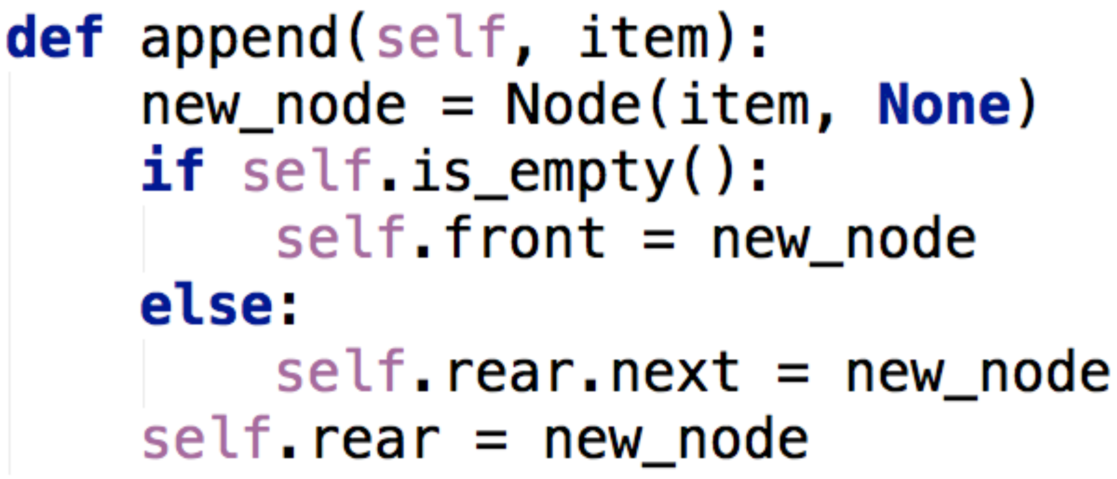
```
q.append(54)
```



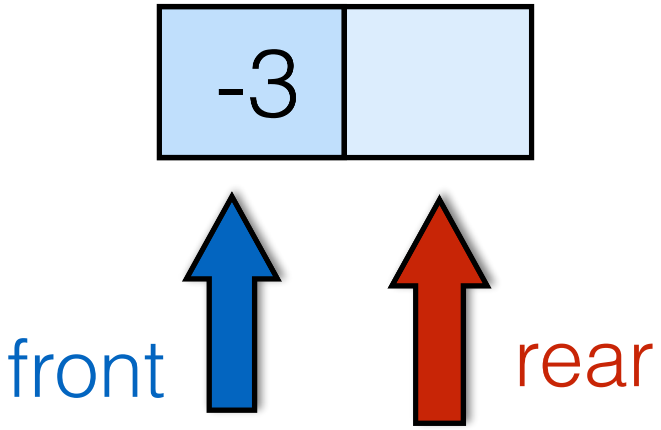
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



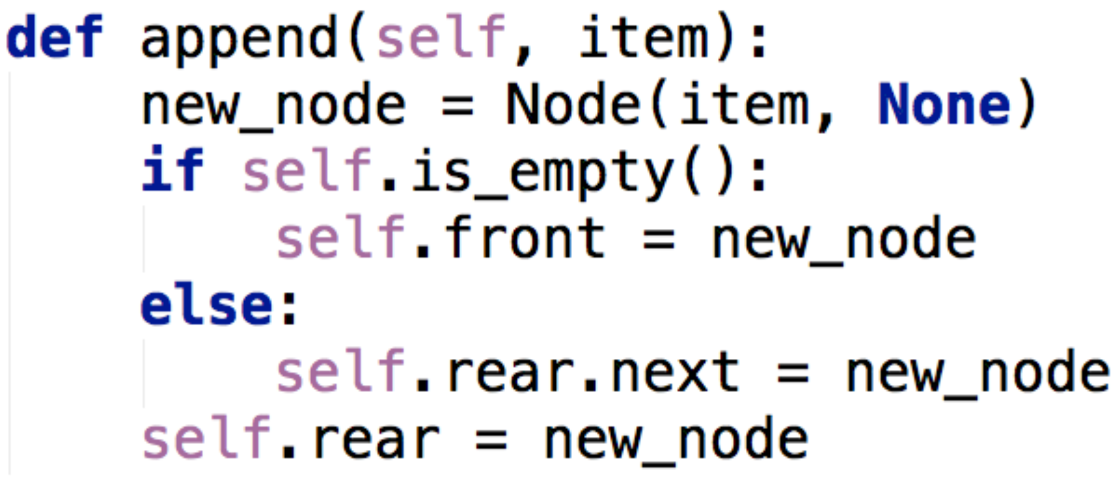
```
if q.front is q.rear
```



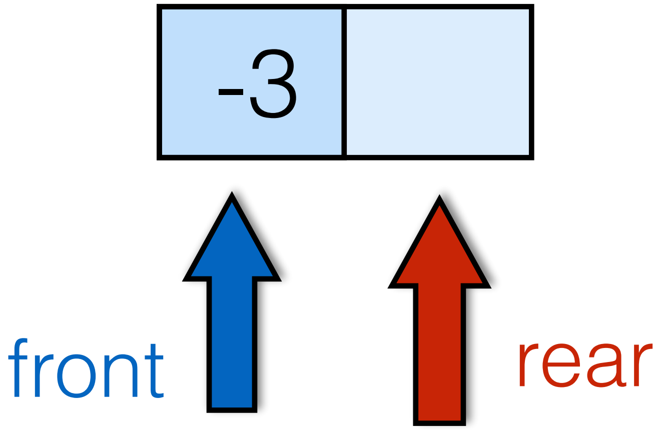
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



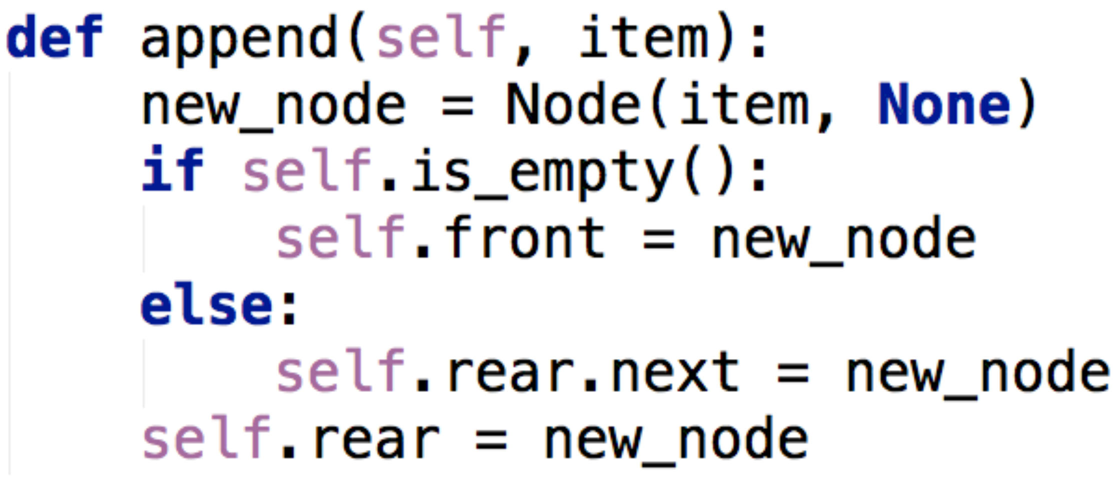
```
if q.front is q.rear
    q.append(54)
```



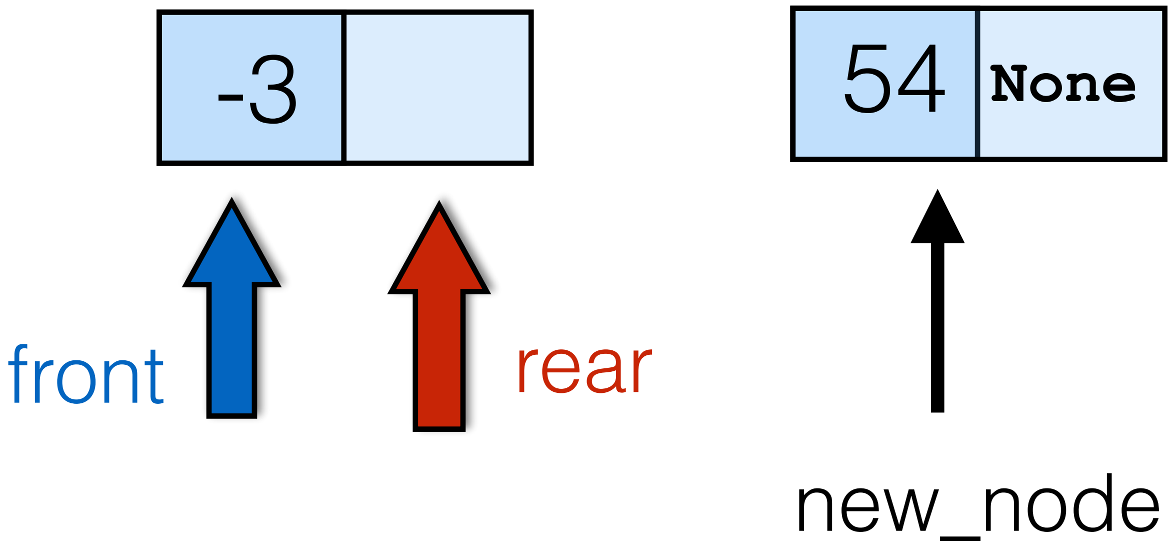
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



```
if q.front is q.rear
      q.append(54)
```

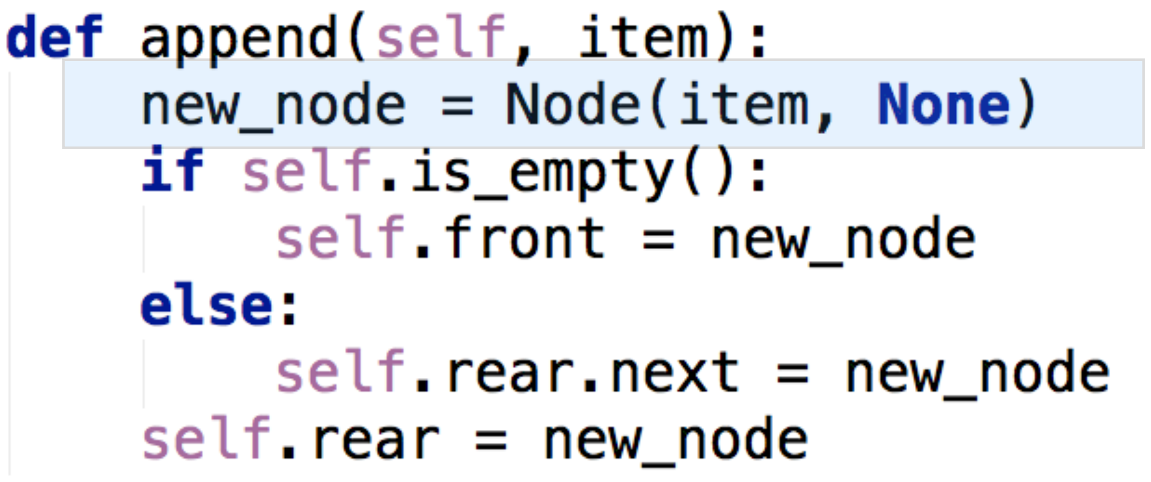


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

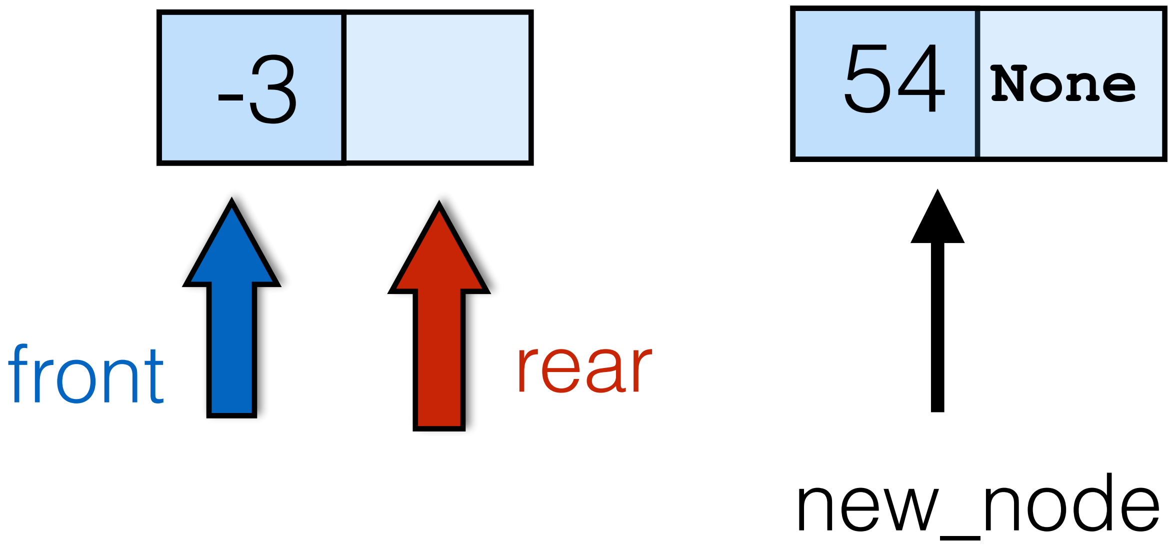


```
if q.front is q.rear
```

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

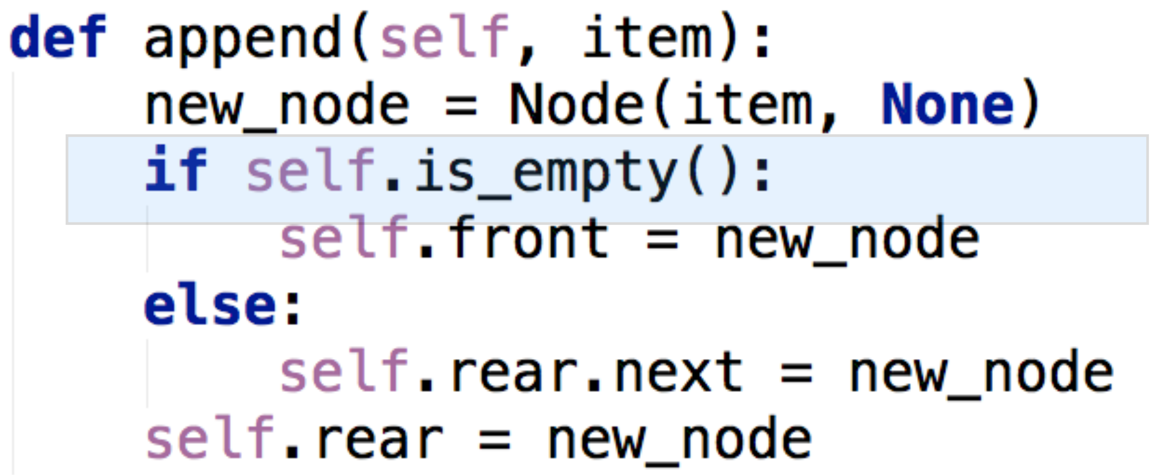


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

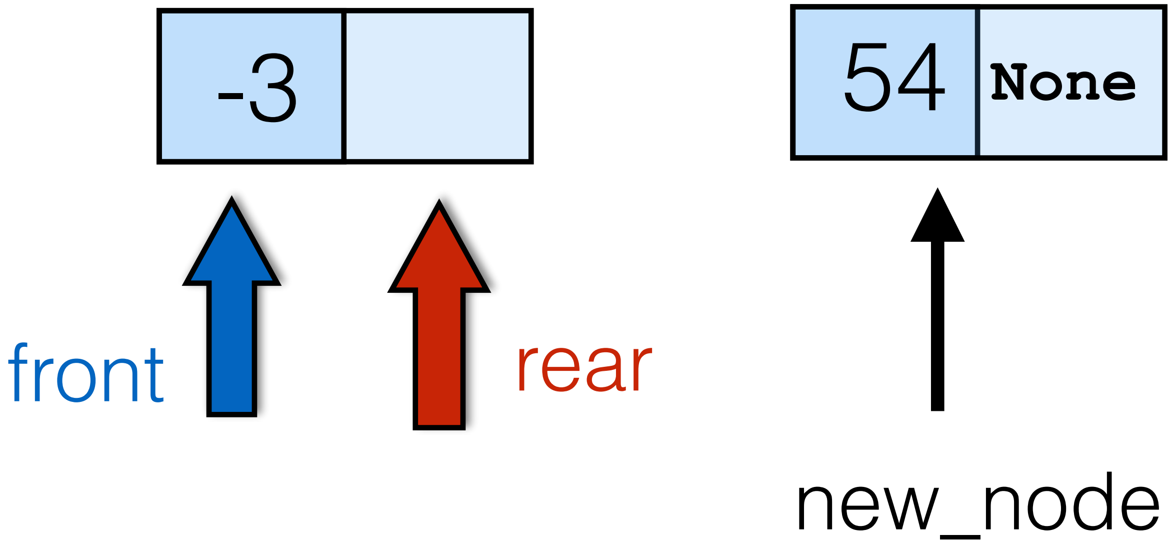


```
if q.front is q.rear
```

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

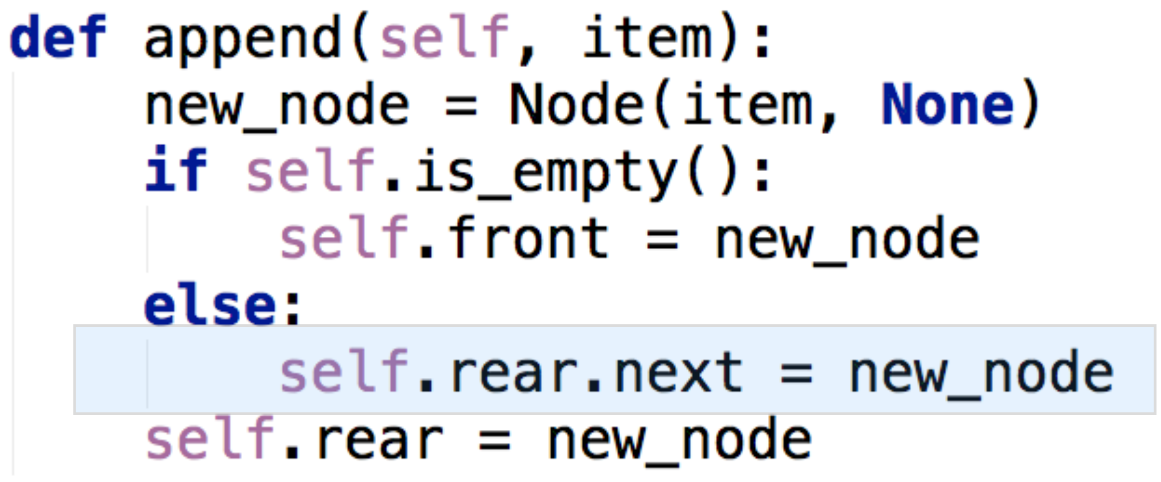


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

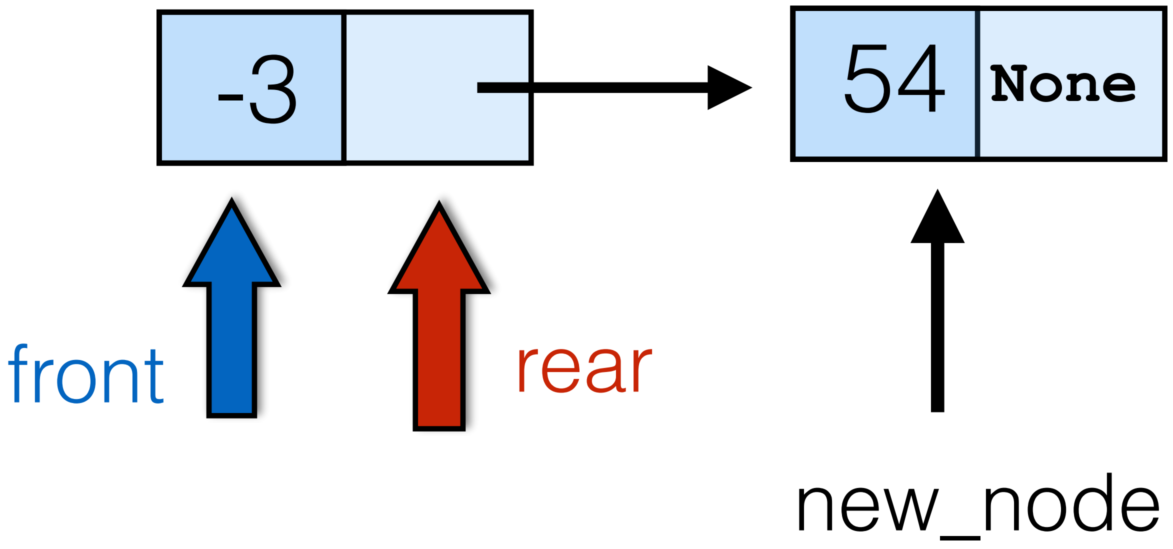


```
if q.front is q.rear
```

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

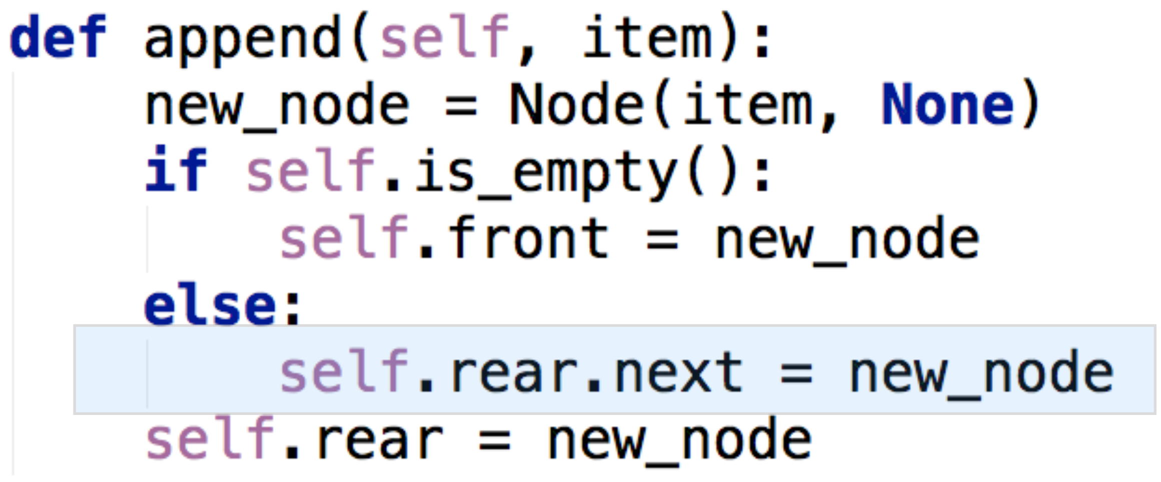


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

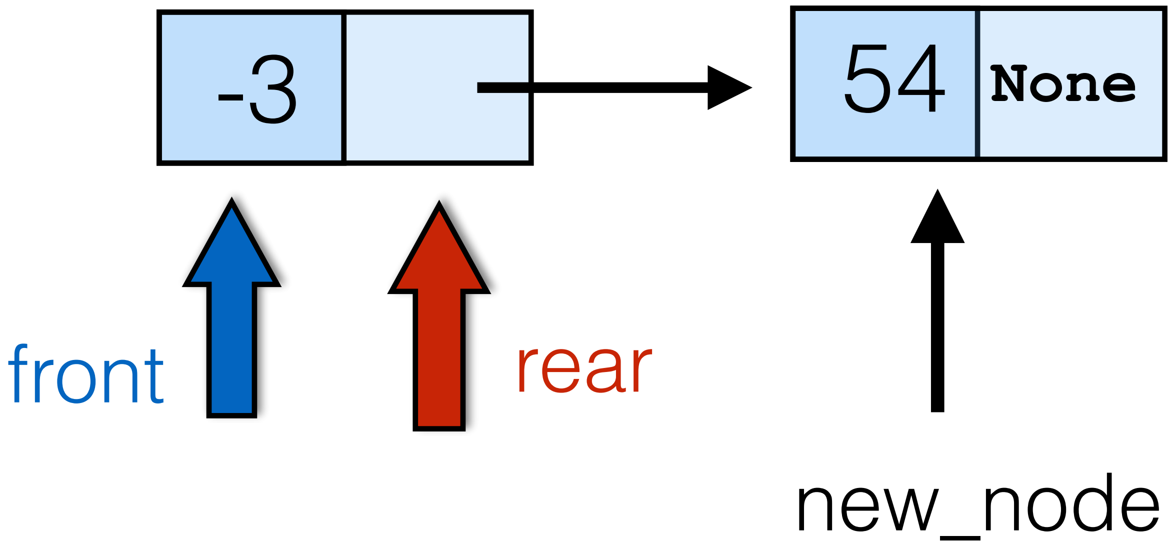


if q.front is q.rear

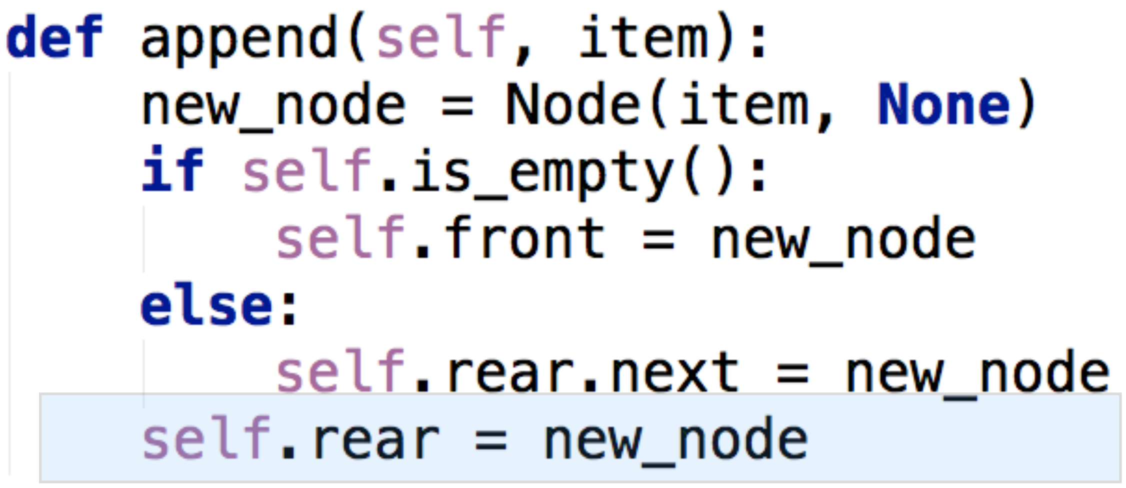
q.append(54)



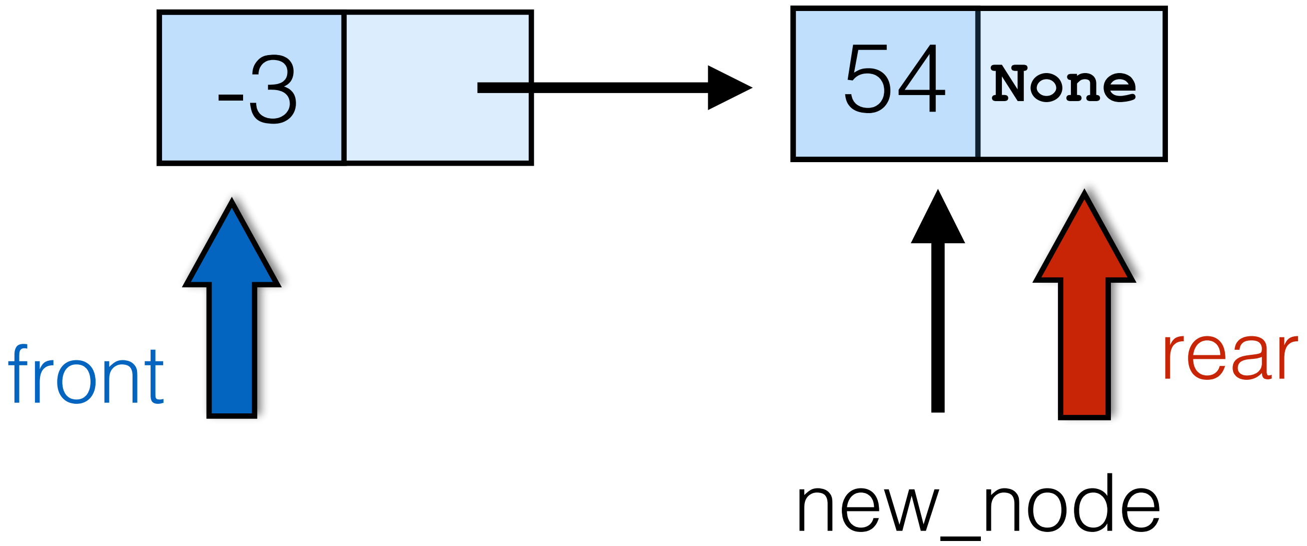
```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



```
if q.front is q.rear
          q.append(54)
```

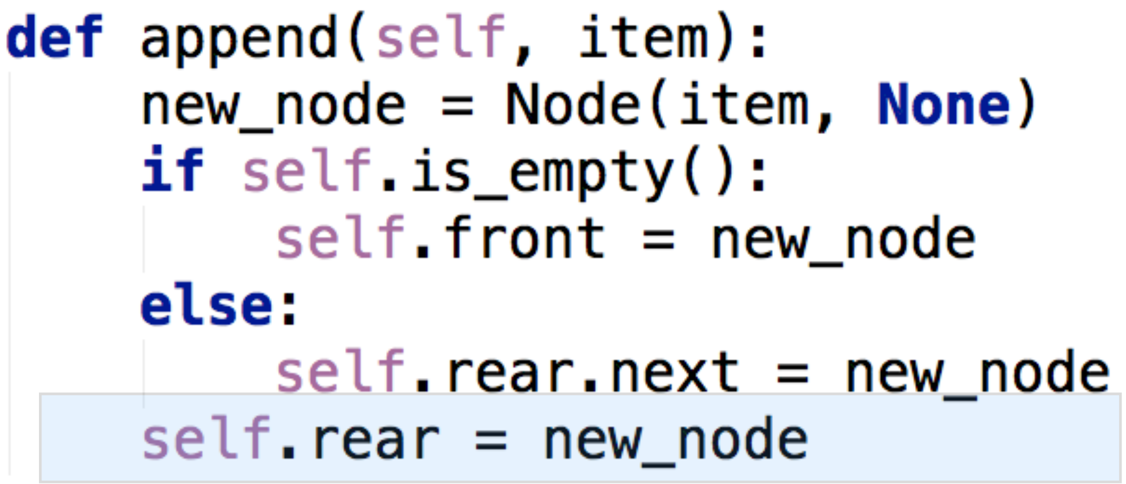


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```

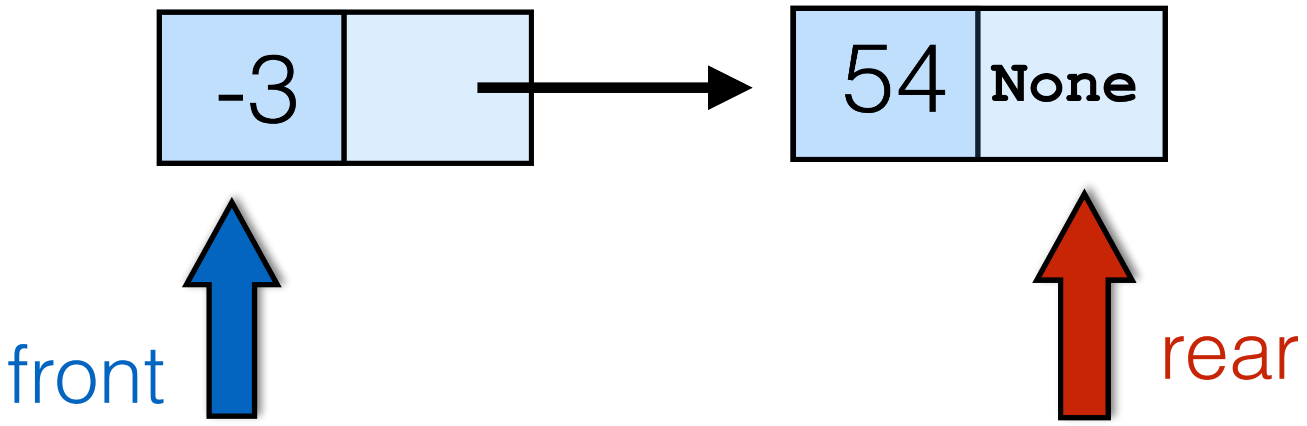


if q.front is q.rear

q.append(54)

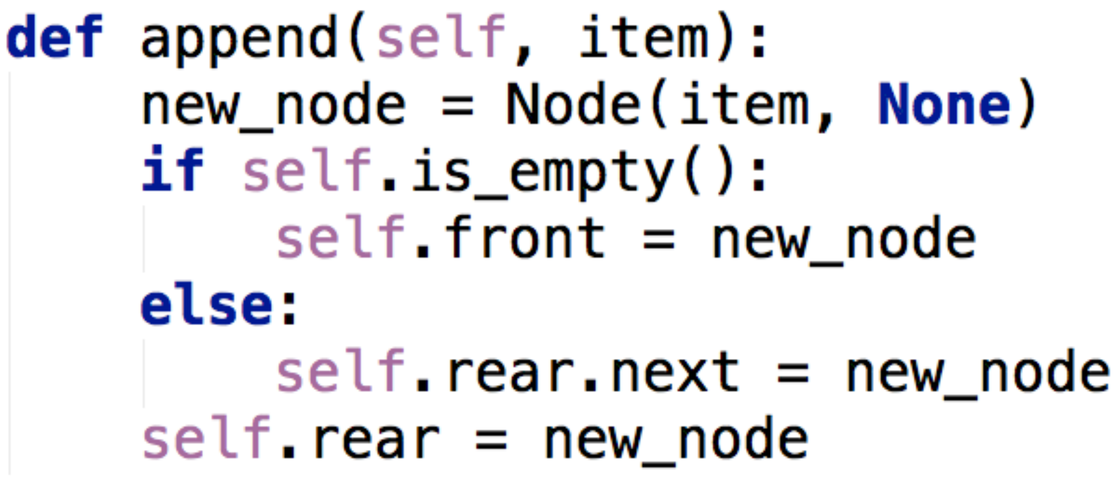


```
def append(self, item):
    new_node = Node(item, None)
    if self.is_empty():
        self.front = new_node
    else:
        self.rear.next = new_node
    self.rear = new_node
```



```
if q.front is 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:

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

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:

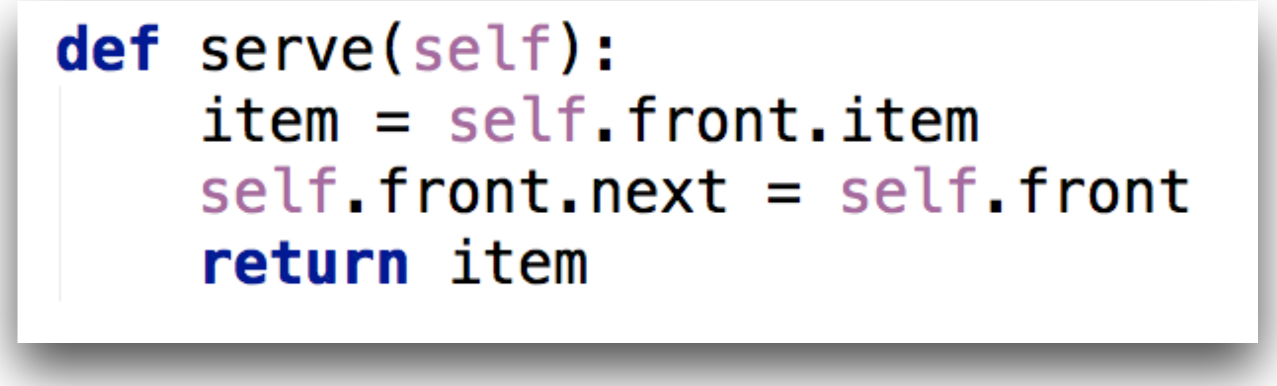
Serve: algorithm

Circular array implementation:

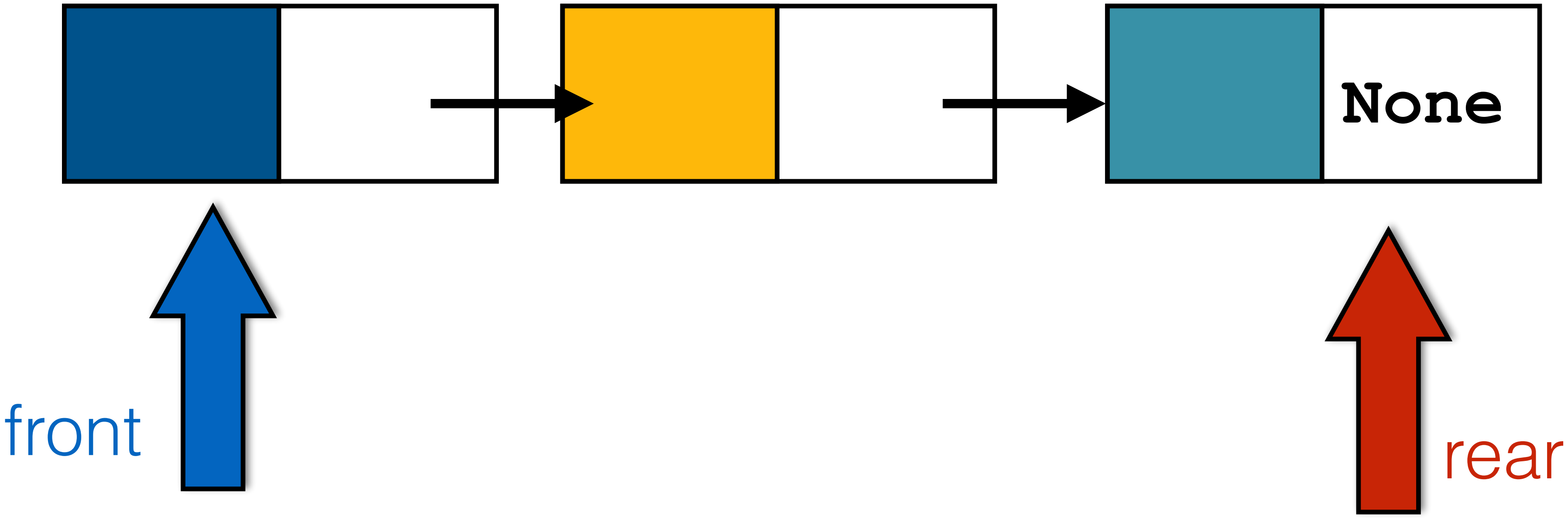
- If the array is empty raise exception
- Else
  - Remember item to return
  - Increase front % length of the array
  - Return the item

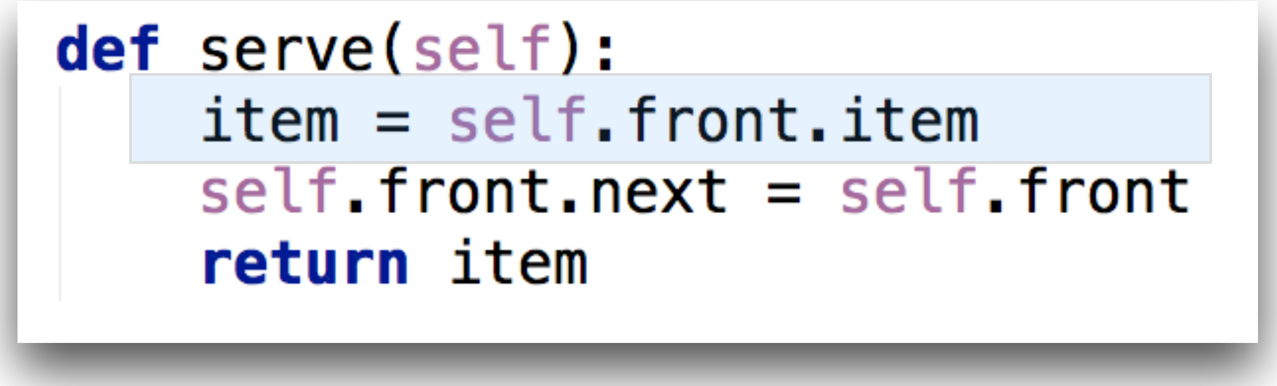
Linked implementation:

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

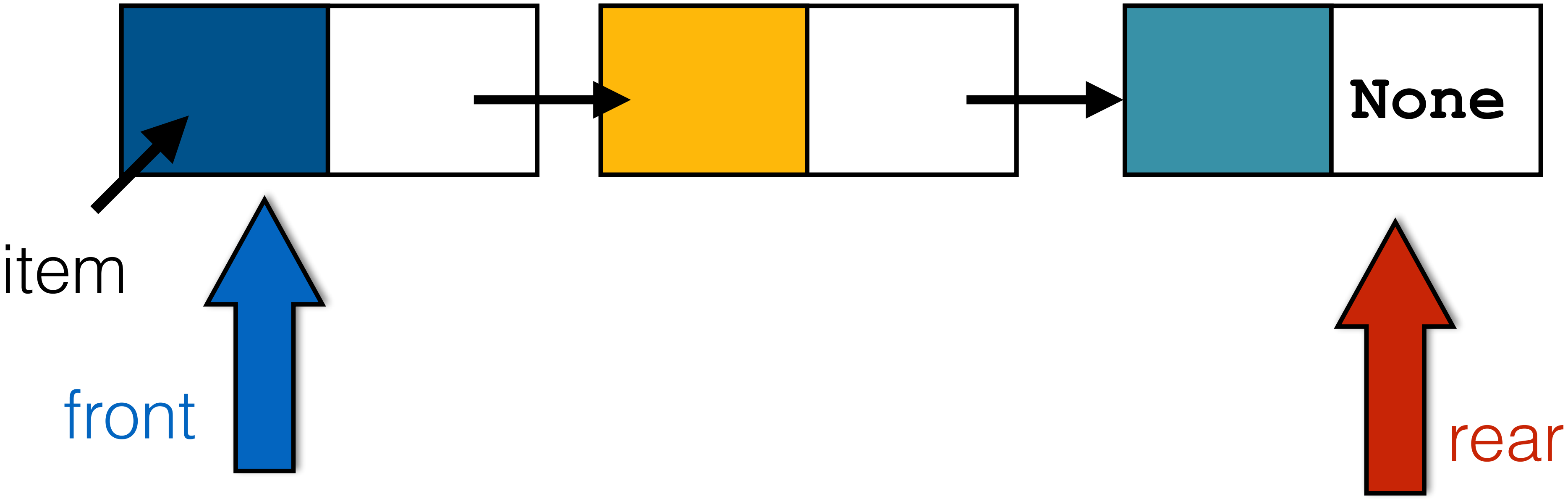


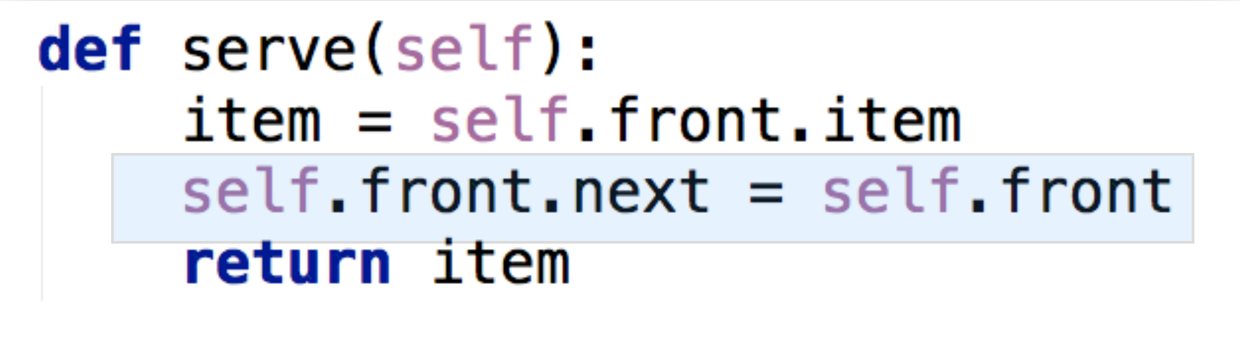
```python
def serve(self):
    item = self.front.item
    self.front.next = self.front
    return item
```



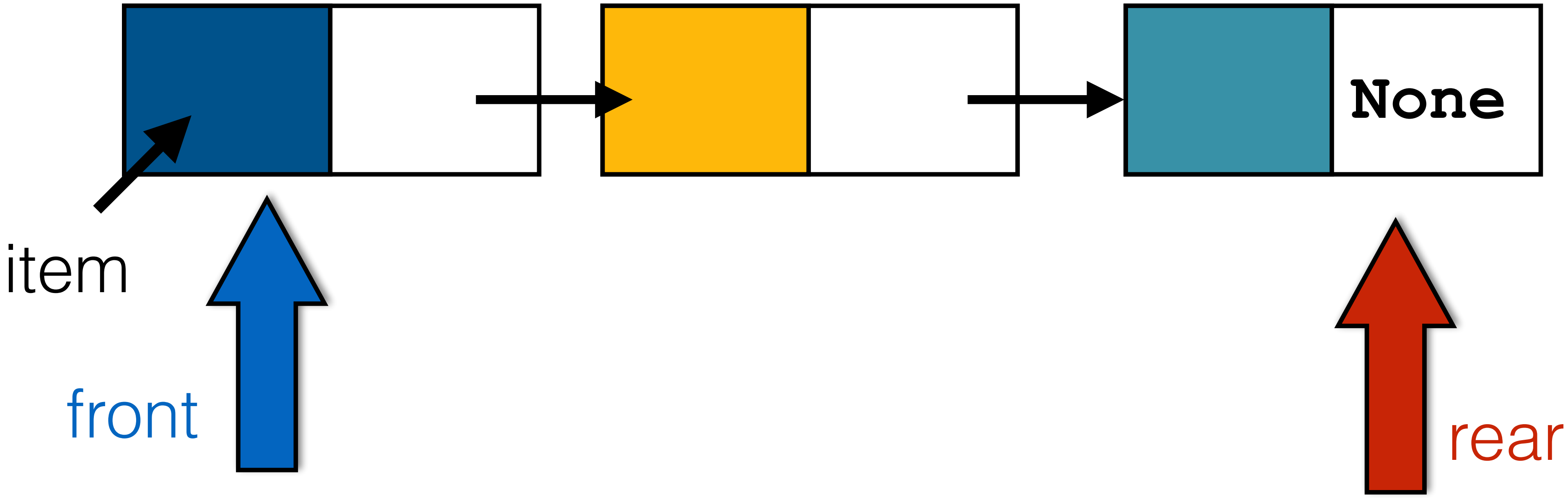


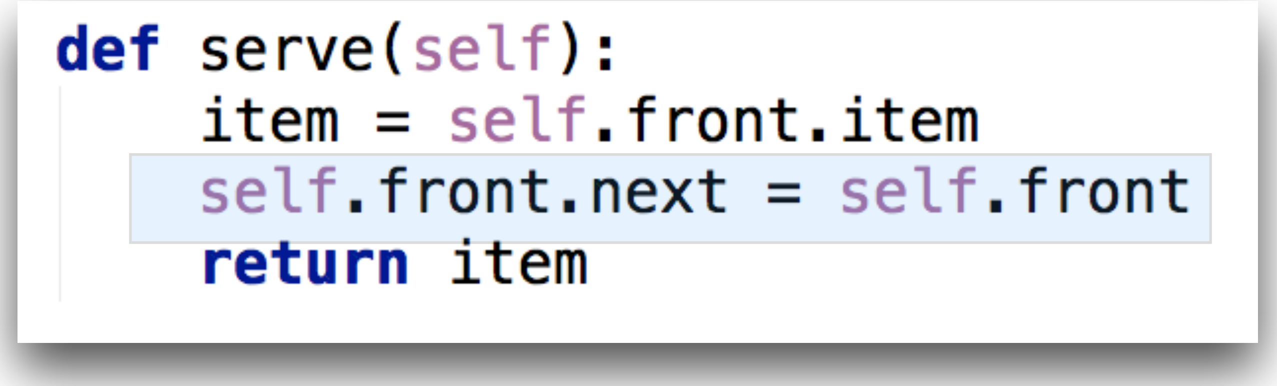
```python
def serve(self):
    item = self.front.item
    self.front.next = self.front
    return item
```



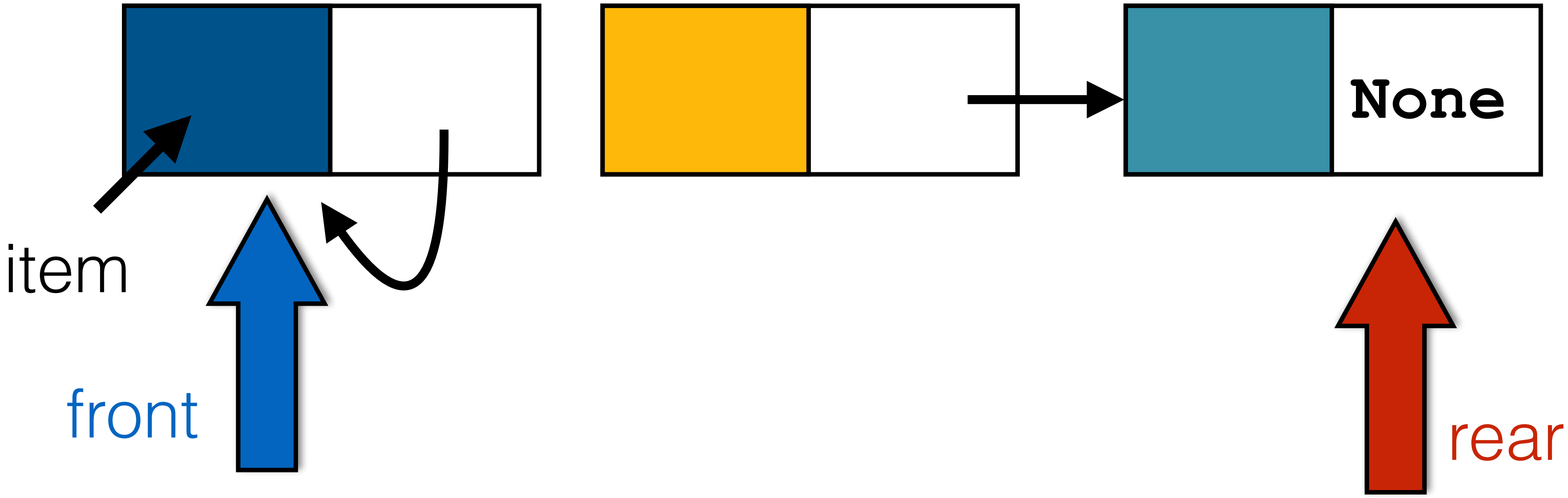


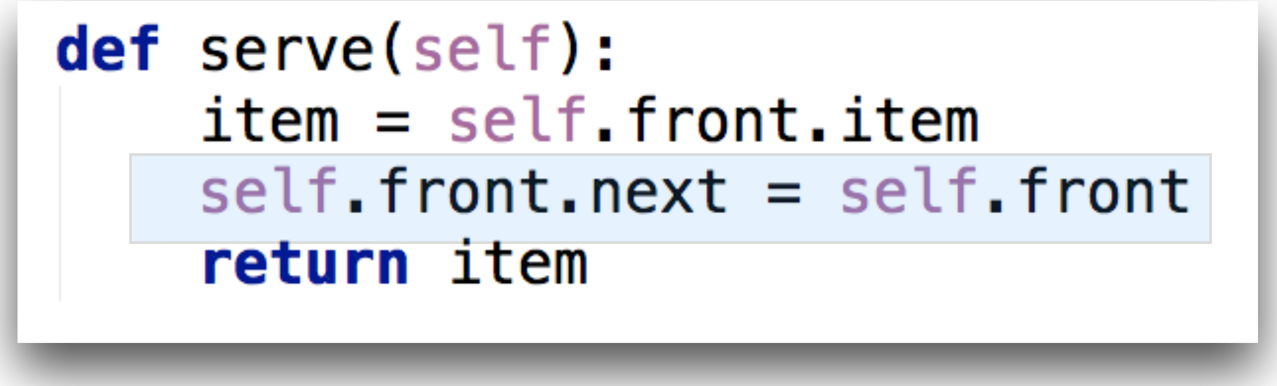
```
def serve(self):
    item = self.front.item
    self.front.next = self.front
    return item
```



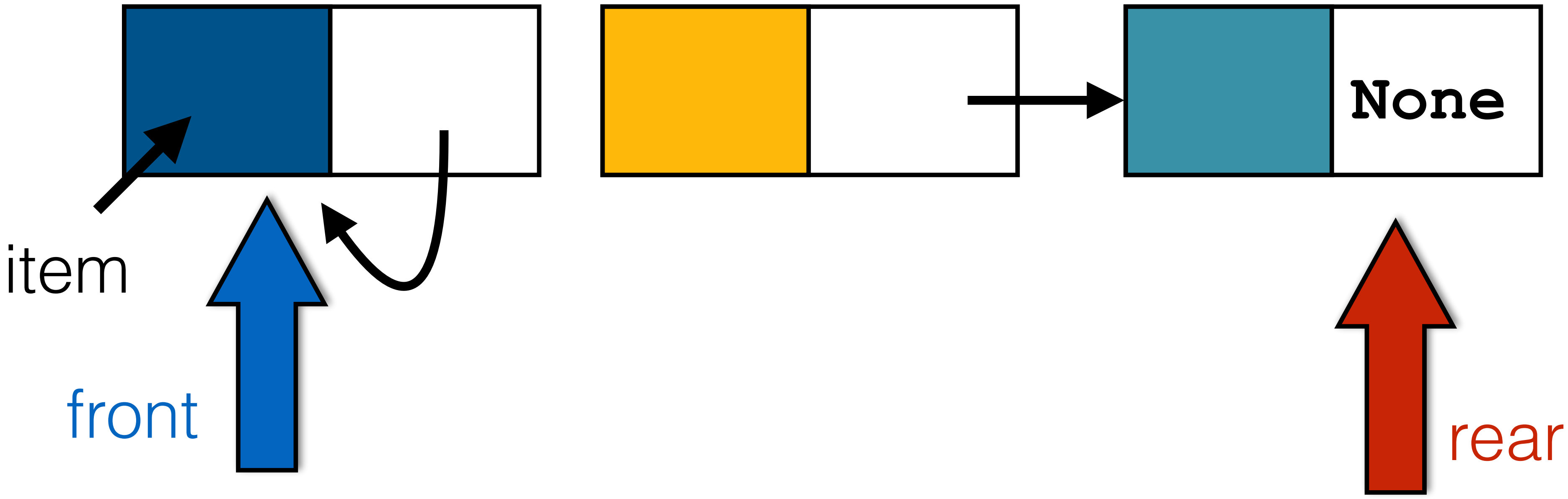


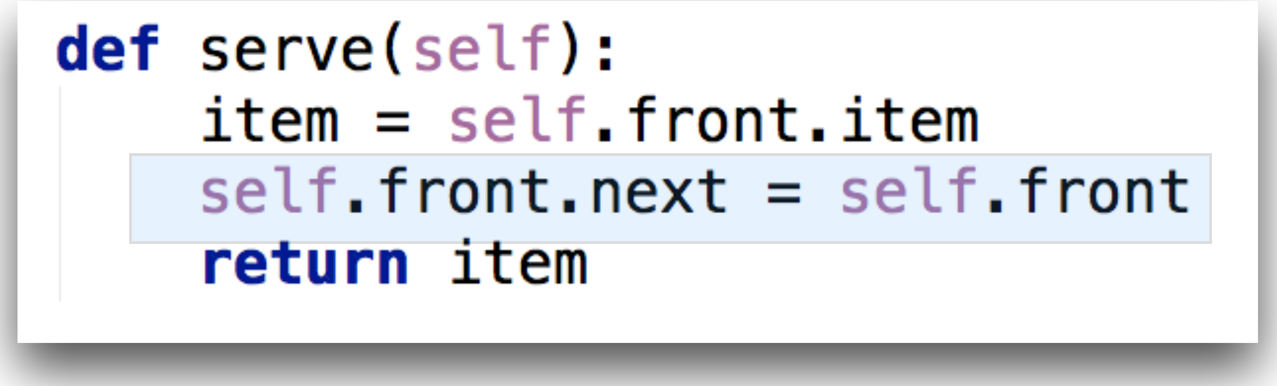
```python
def serve(self):
    item = self.front.item
    self.front.next = self.front
    return item
```





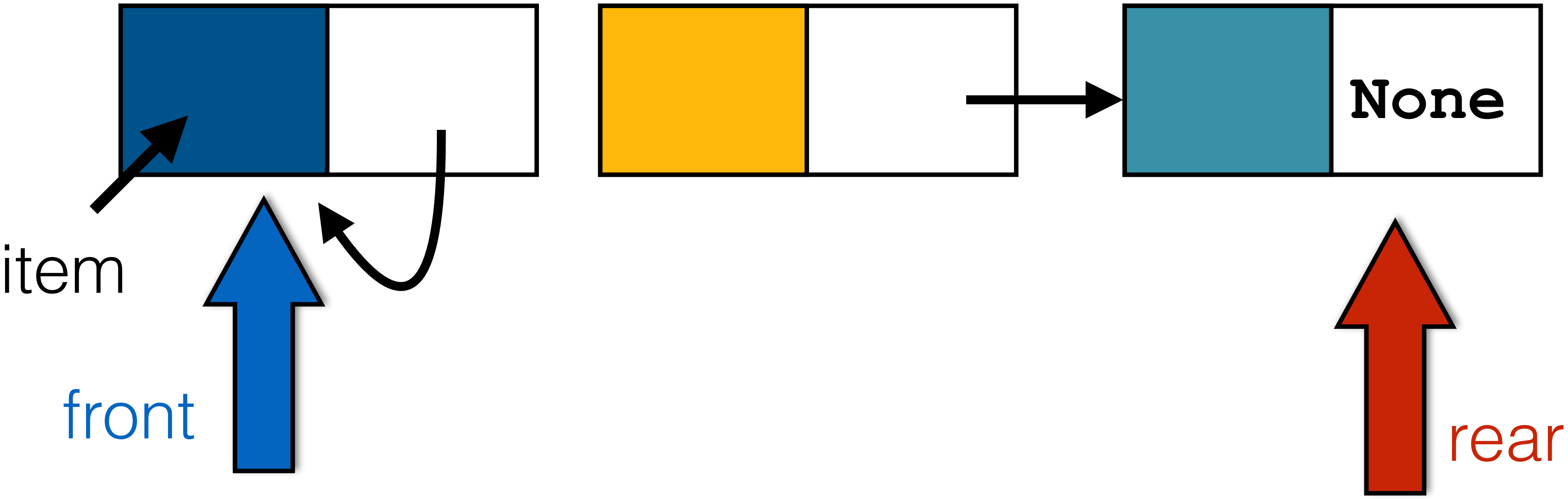
```
def serve(self):
    item = self.front.item
    self.front.next = self.front
    return item
```



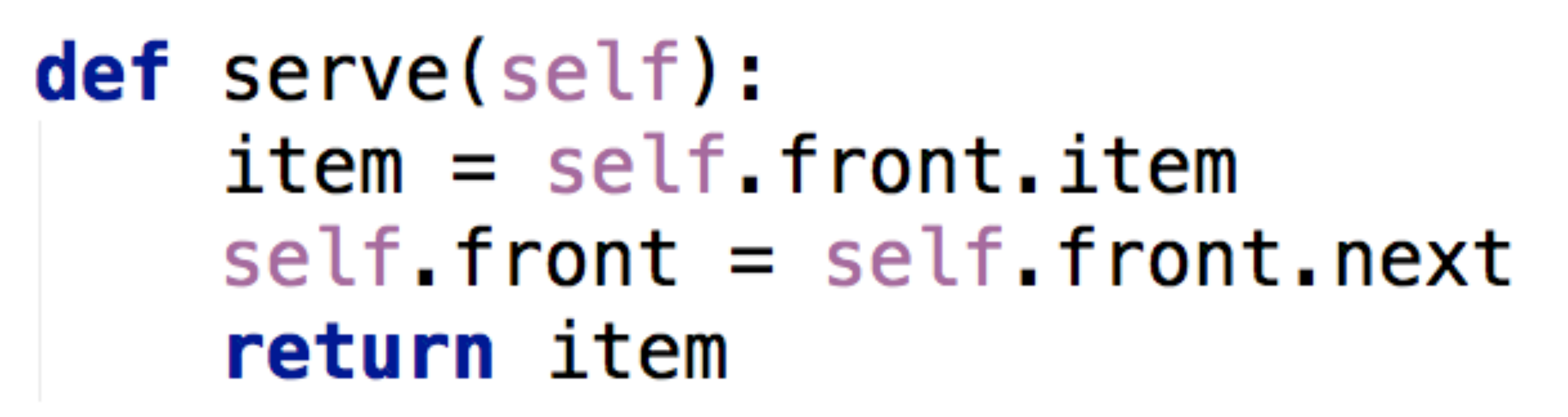


```
def serve(self):
    item = self.front.item
    self.front.next = self.front
    return item
```

?



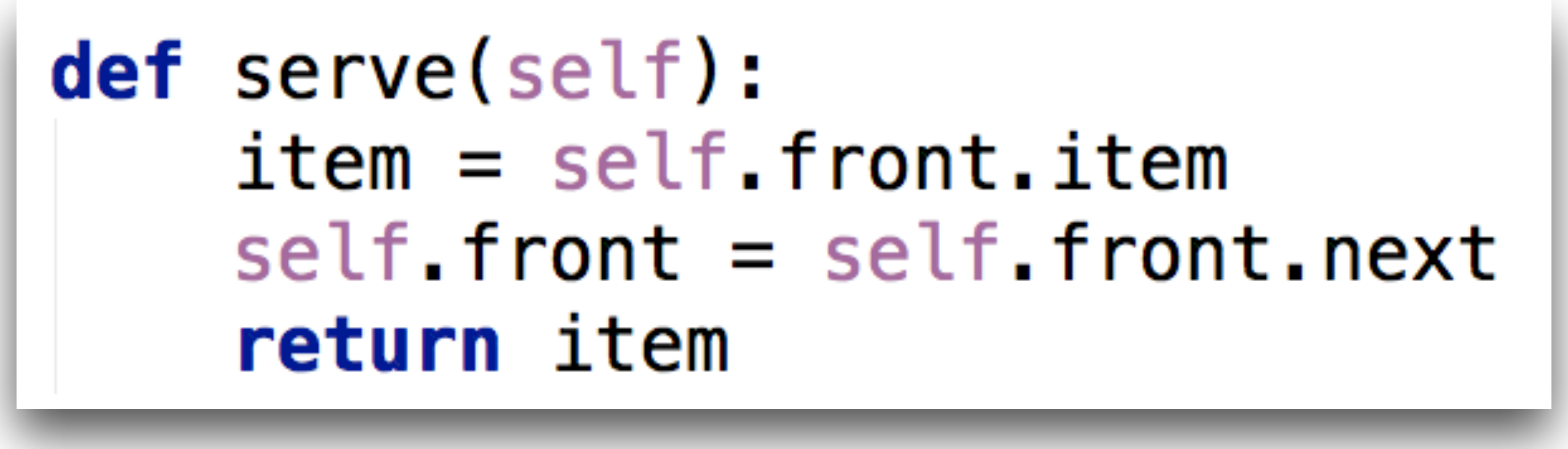
Is the following code correct?



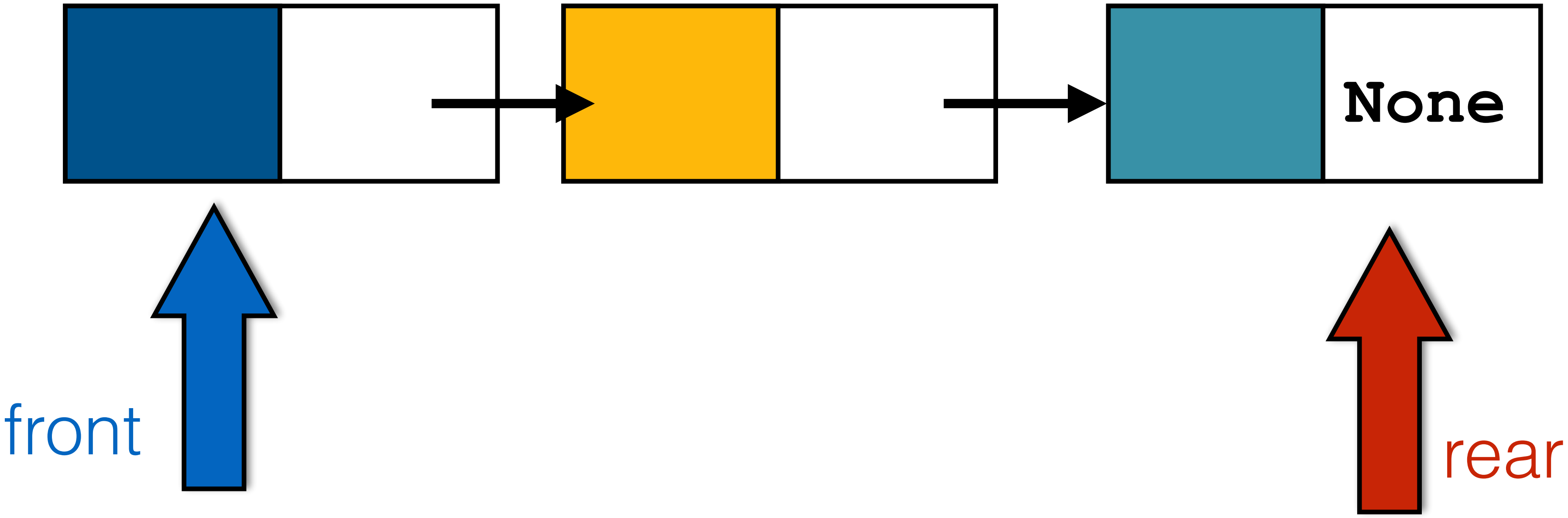
```
def serve(self):
    item = self.front.item
    self.front = self.front.next
    return item
```

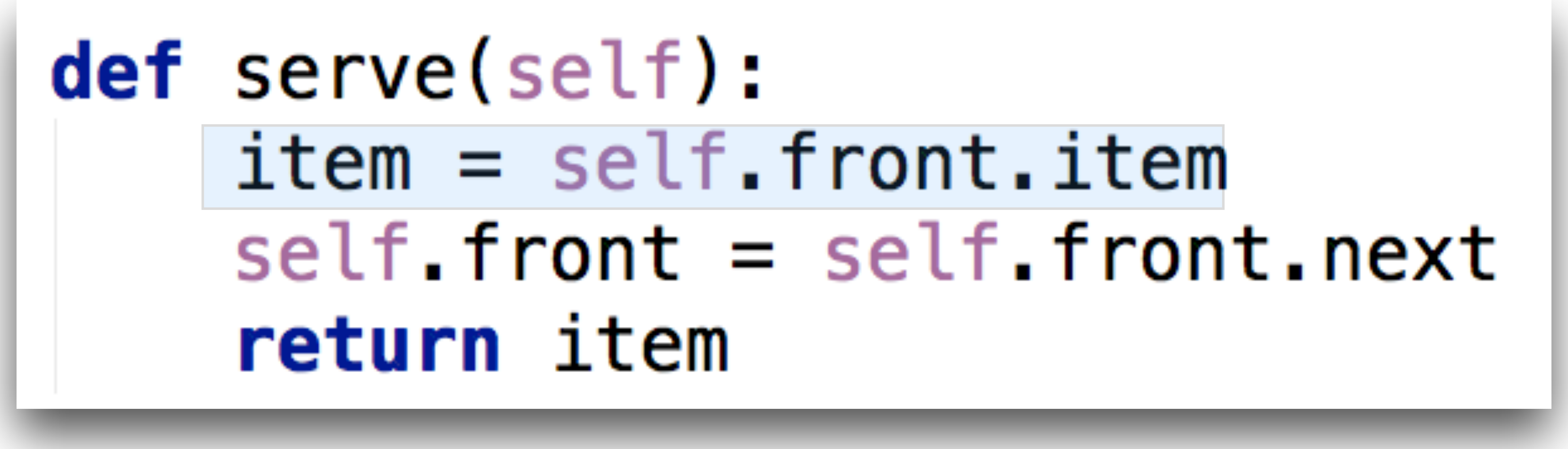
A) Yes

B) No

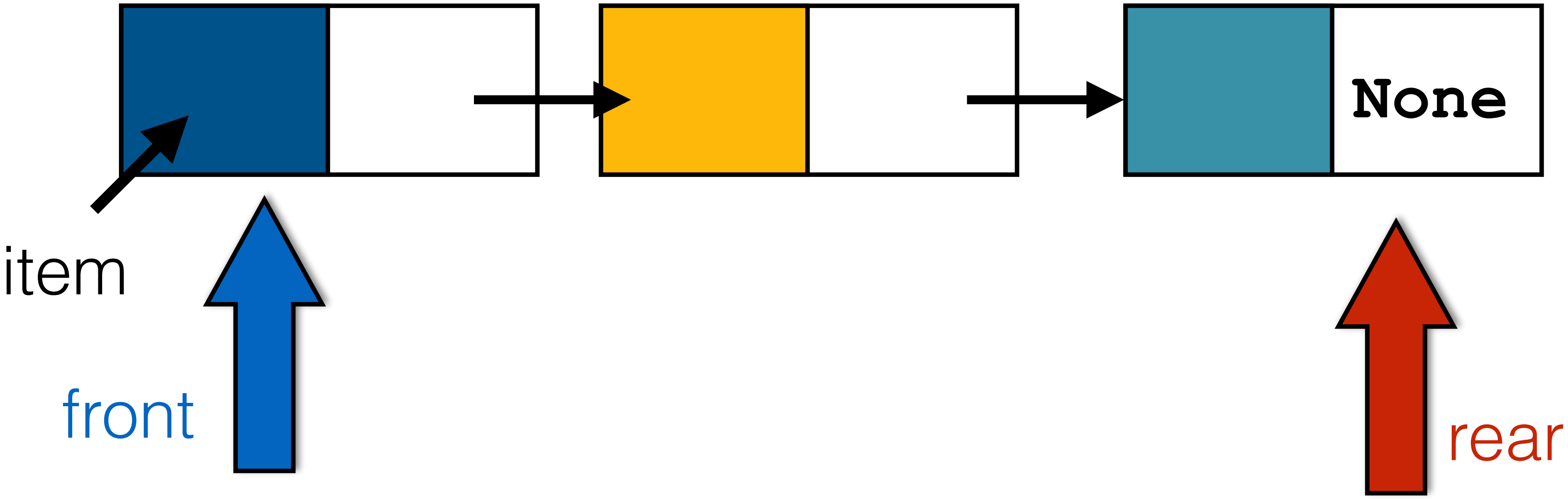


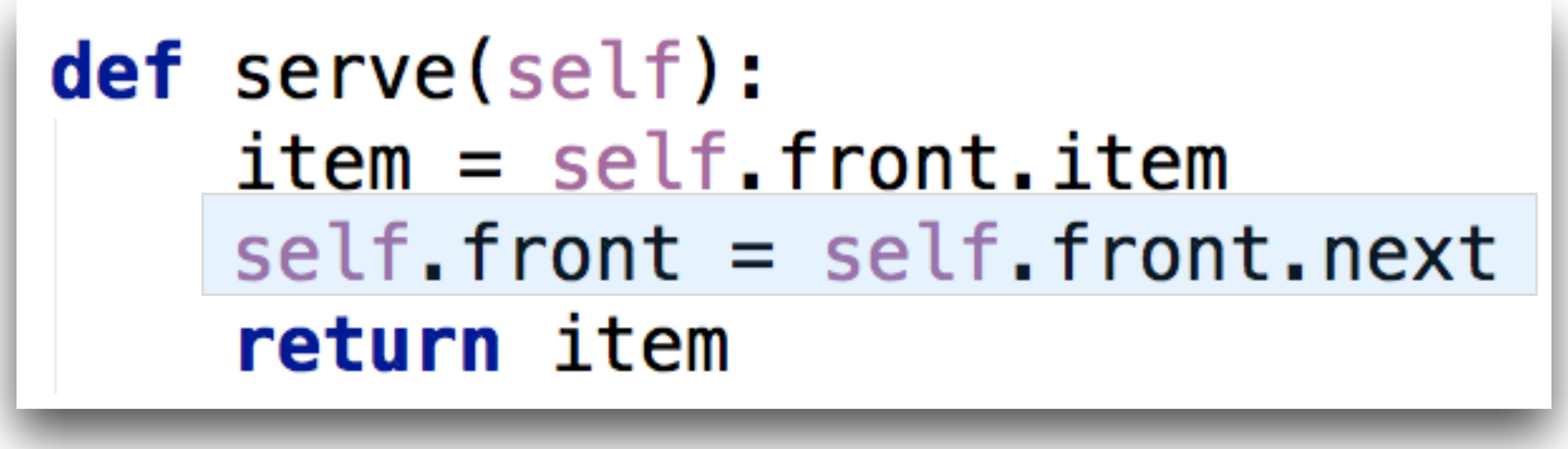
```
def serve(self):
    item = self.front.item
    self.front = self.front.next
    return item
```



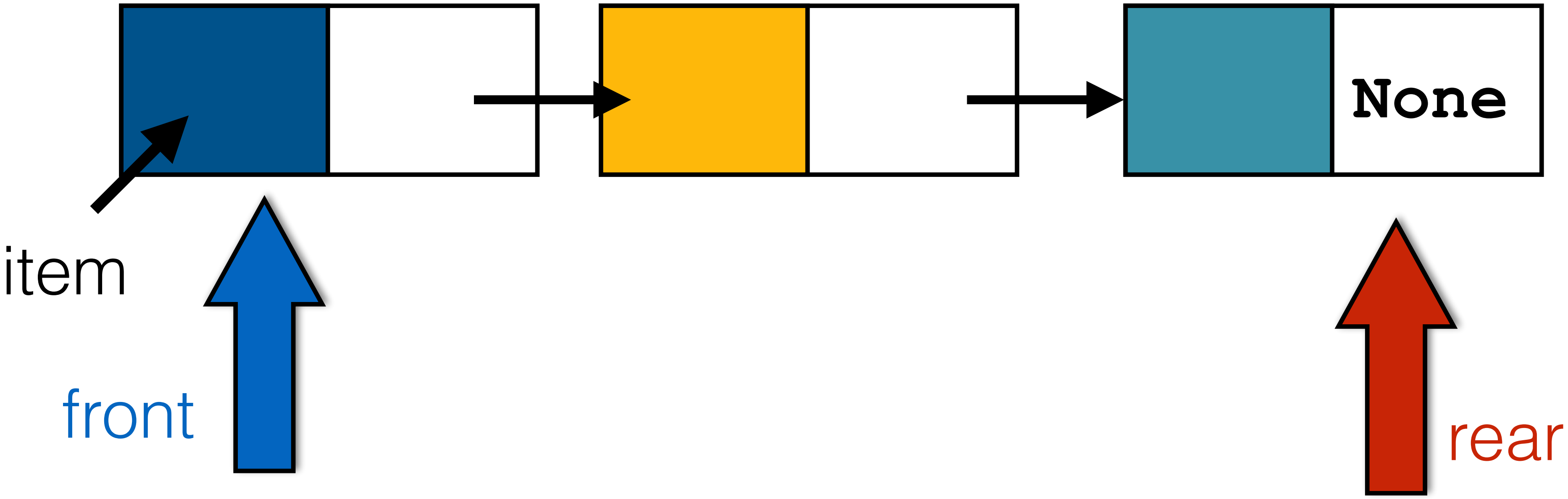


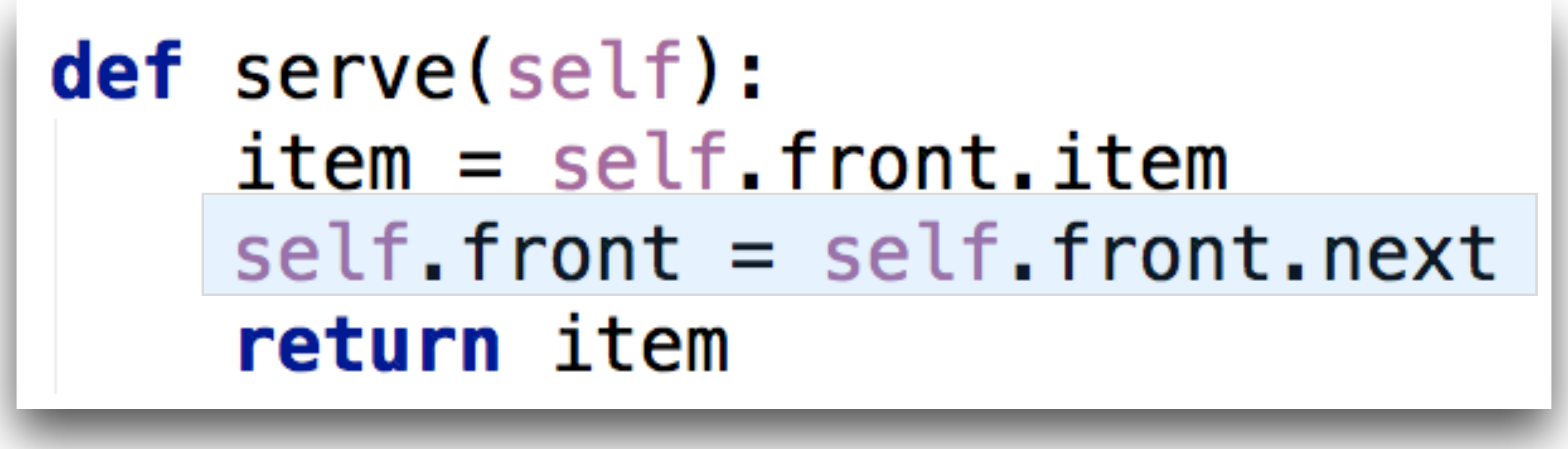
```python
def serve(self):
    item = self.front.item
    self.front = self.front.next
    return item
```



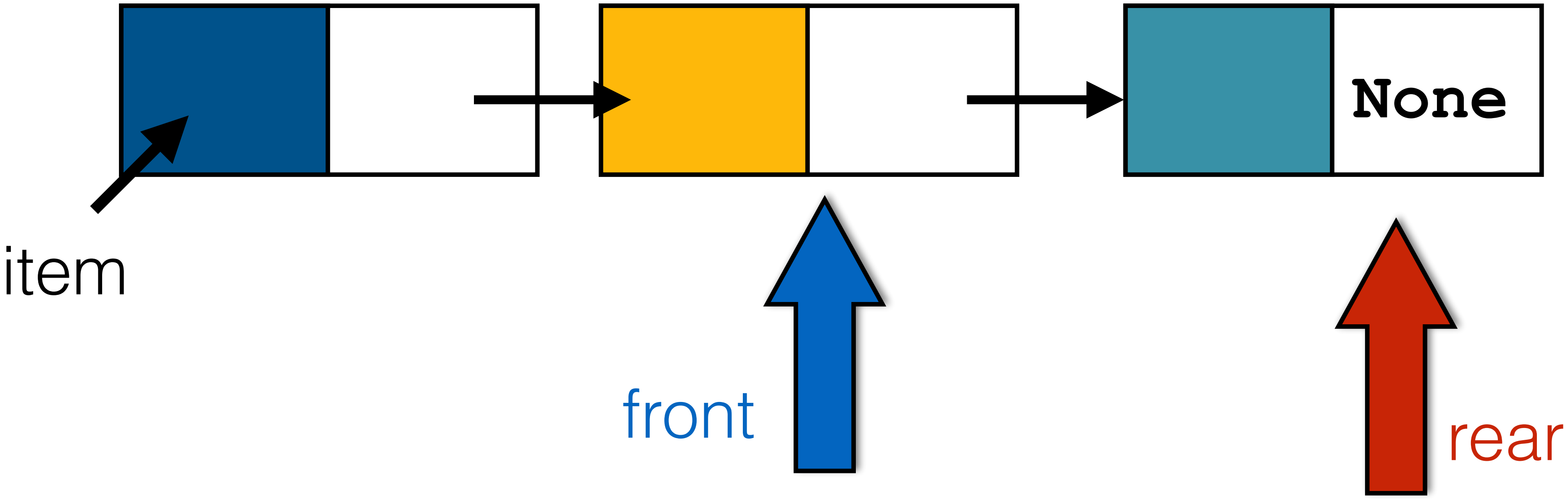


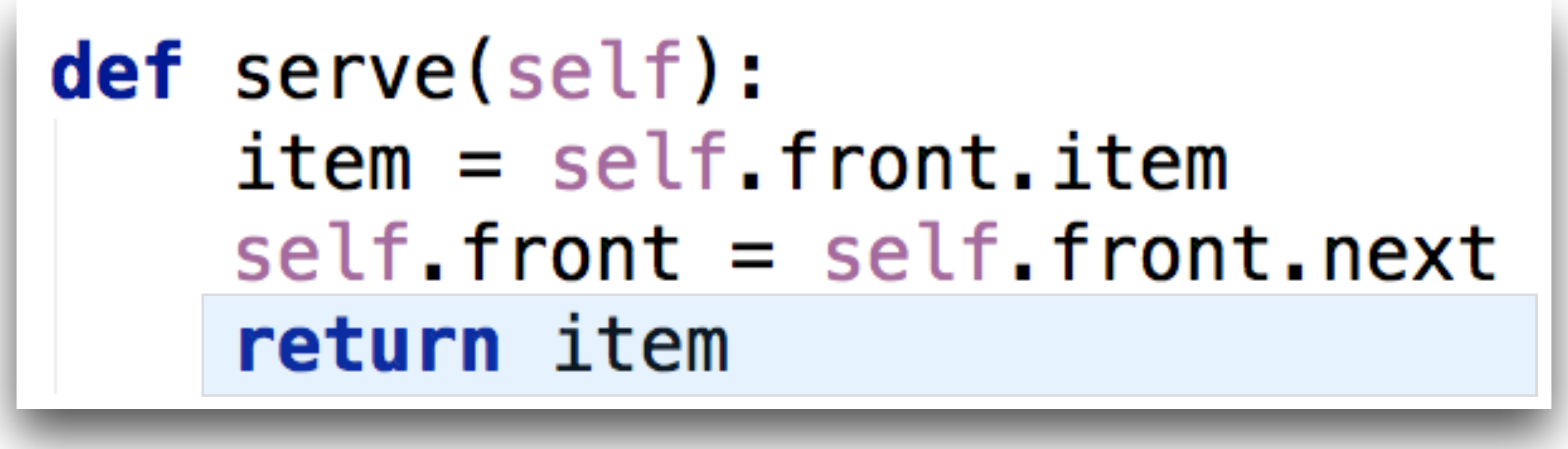
```python
def serve(self):
    item = self.front.item
    self.front = self.front.next
    return item
```



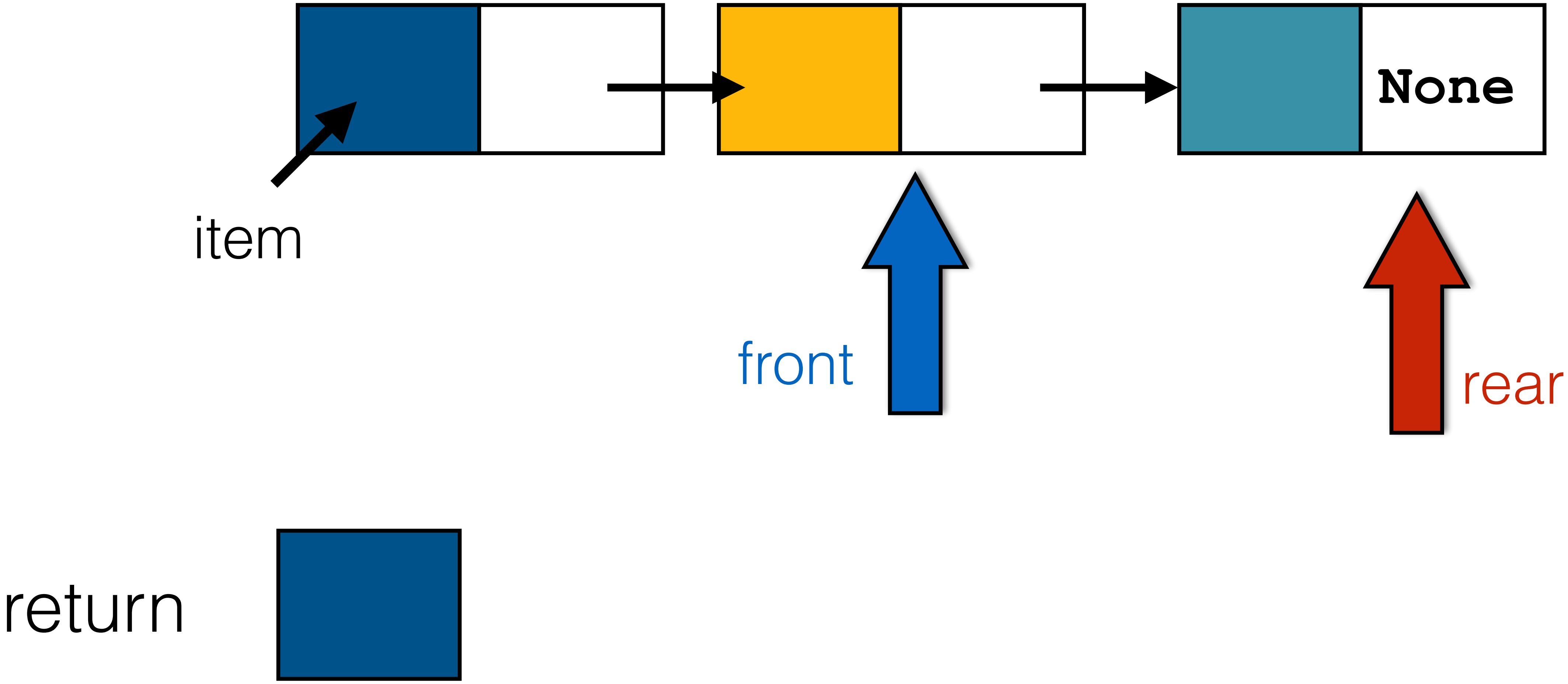


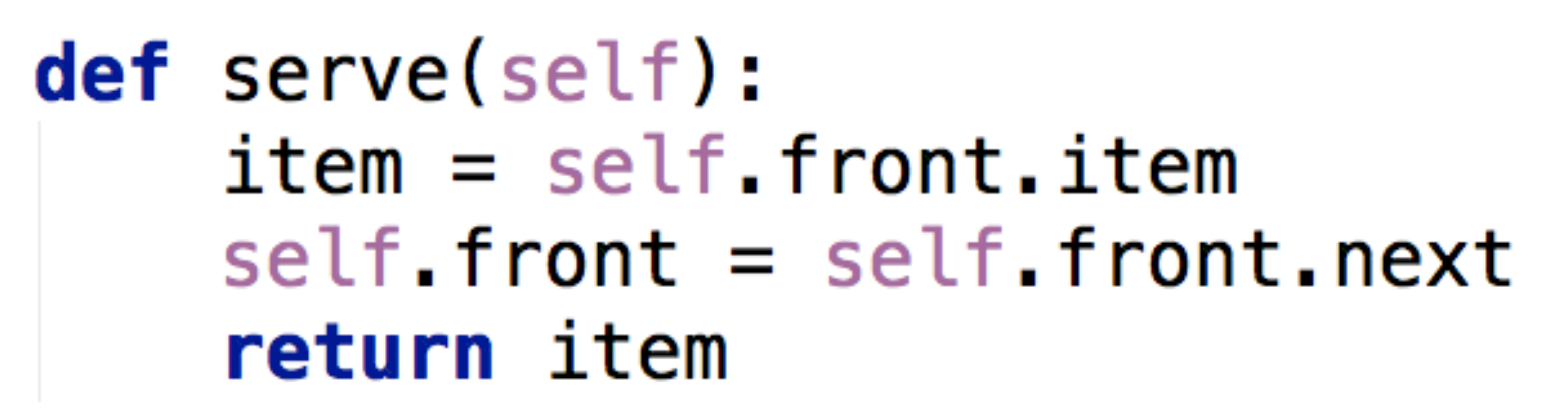
```python
def serve(self):
    item = self.front.item
    self.front = self.front.next
    return item
```



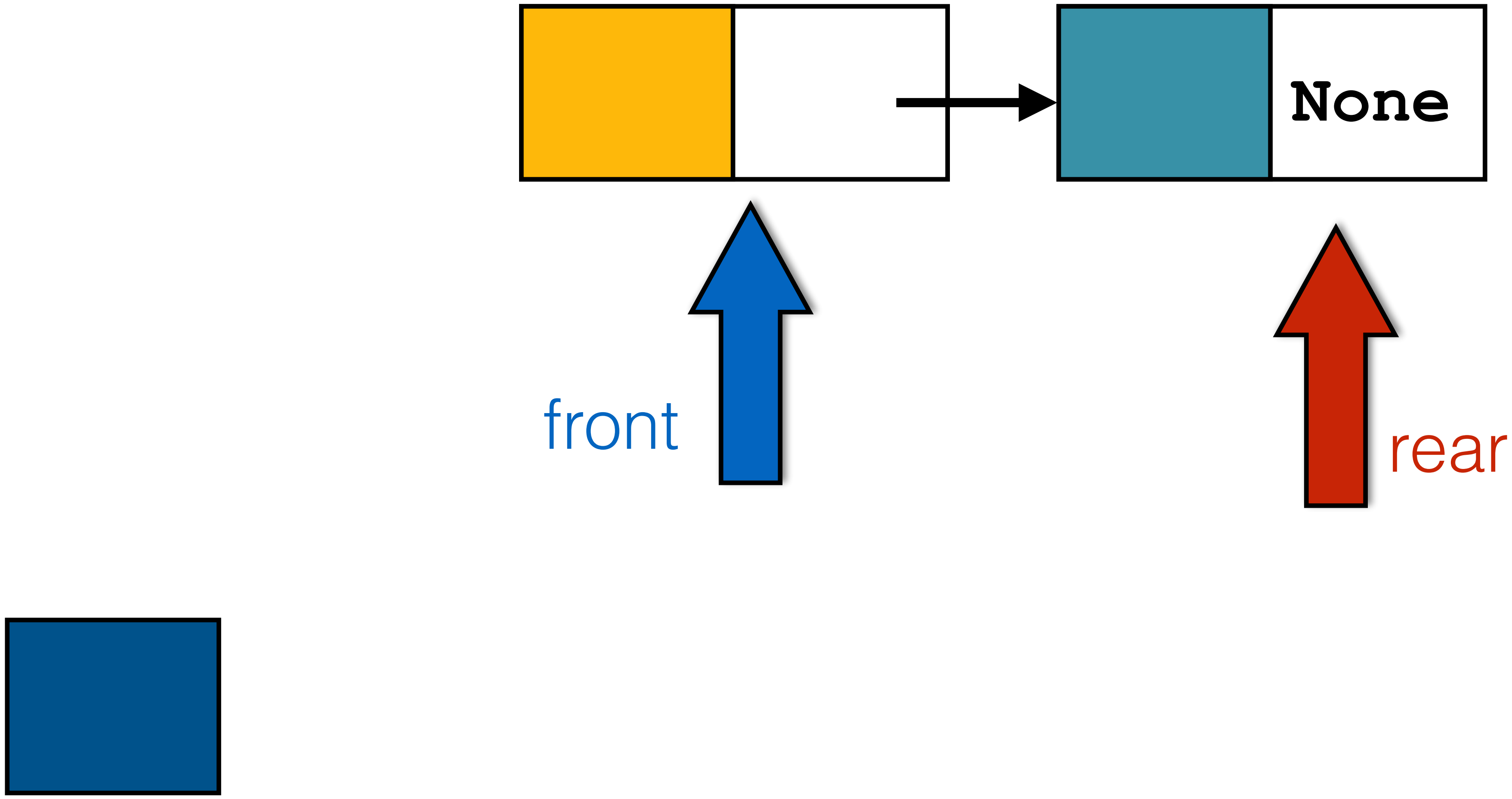


```
def serve(self):
    item = self.front.item
    self.front = self.front.next
    return item
```





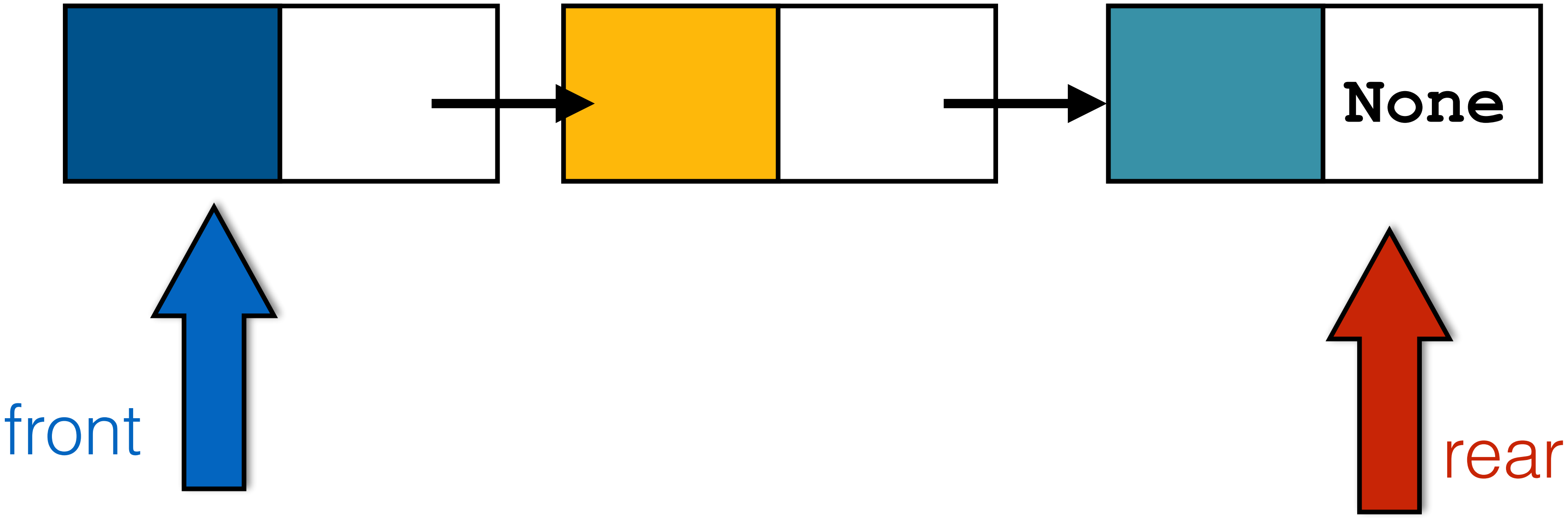
```
def serve(self):
    item = self.front.item
    self.front = self.front.next
    return item
```

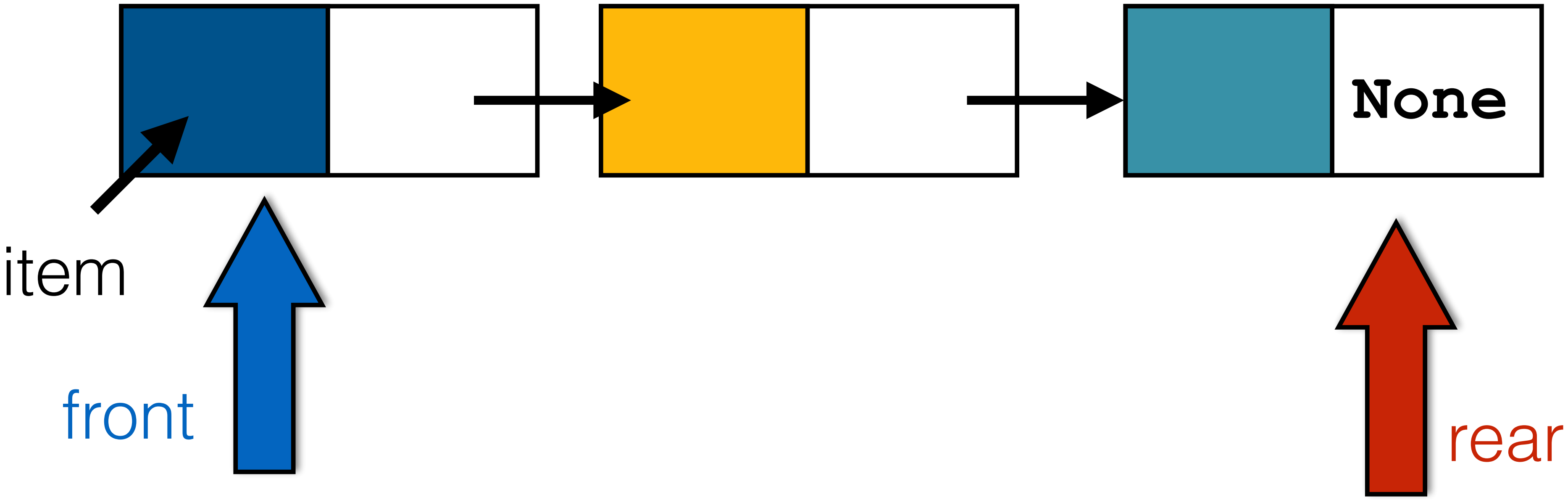


return

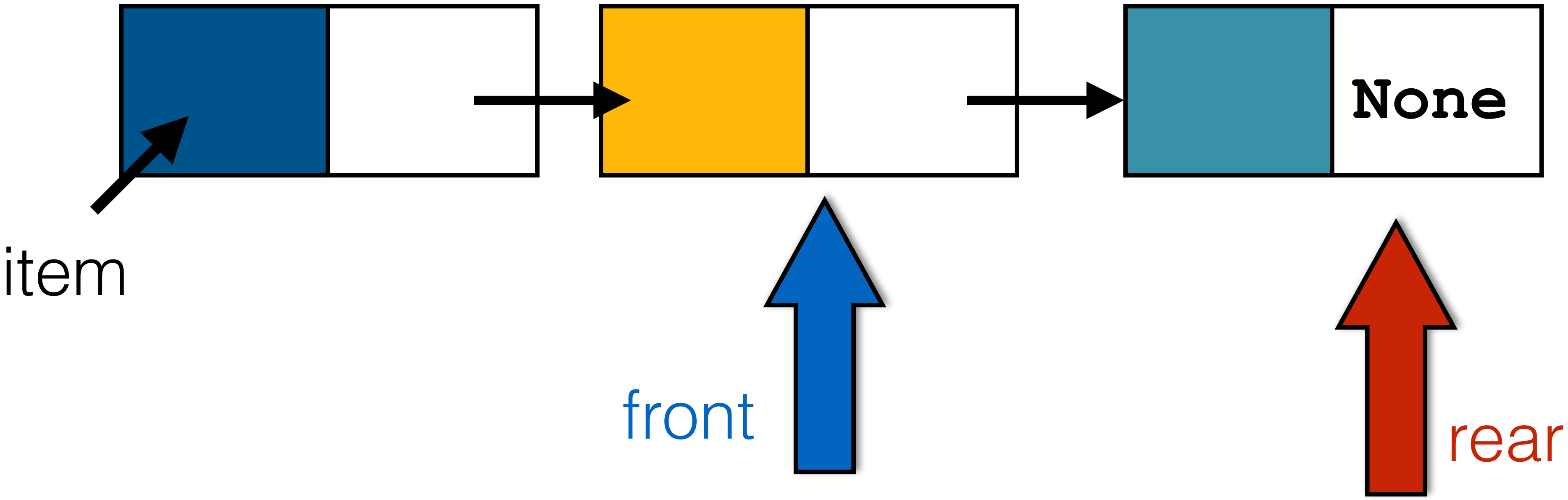
Looking good…

algorithm

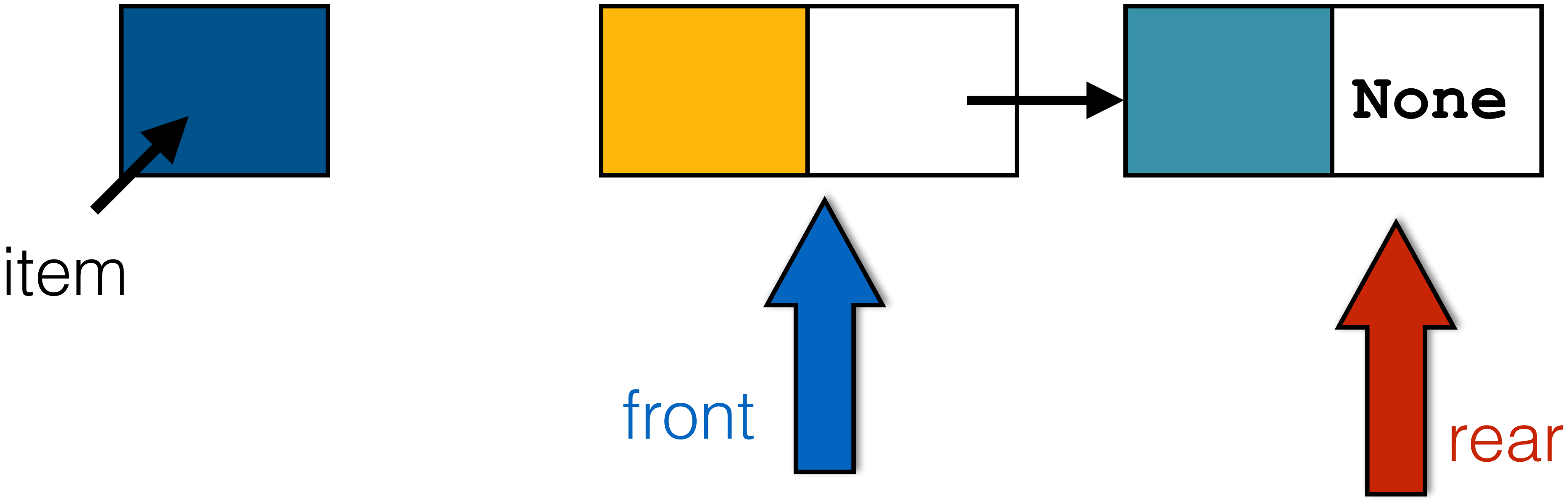

None
front
rear



- Remember the item in the front node.



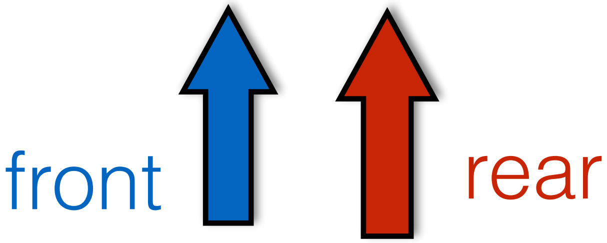
- Remember the item in the front node.
- Make the next node the new front



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

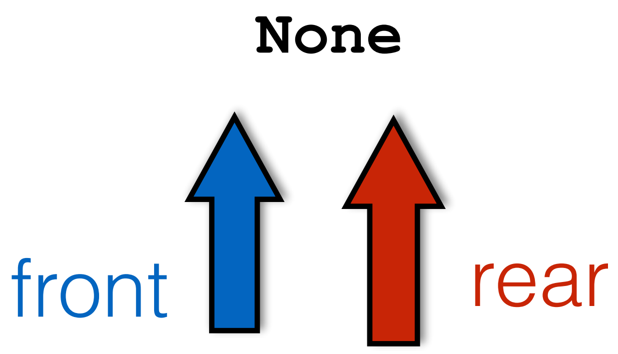
Boundary cases…

None



front

rear

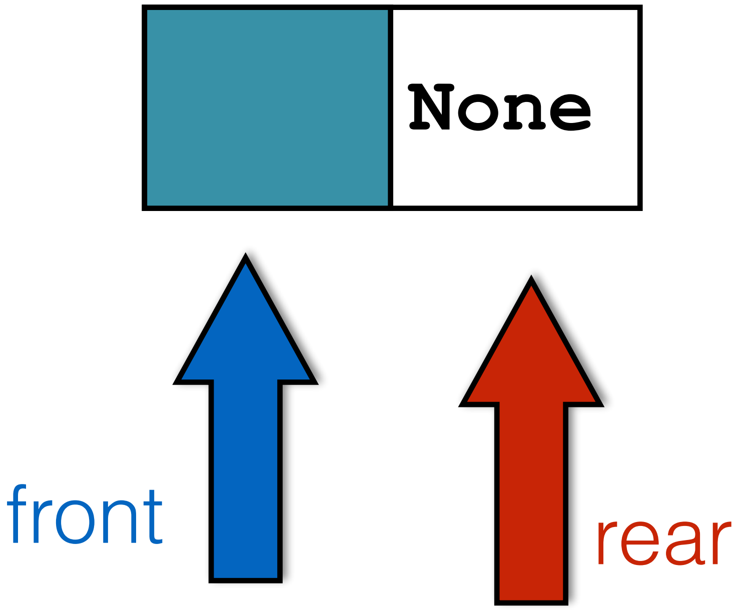


`None`

front

rear

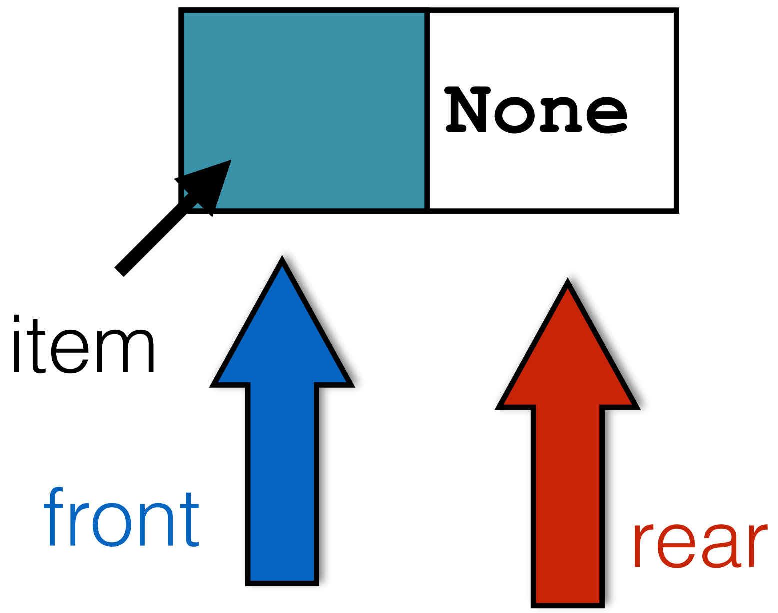
If the queue is empty we need to raise an Exception



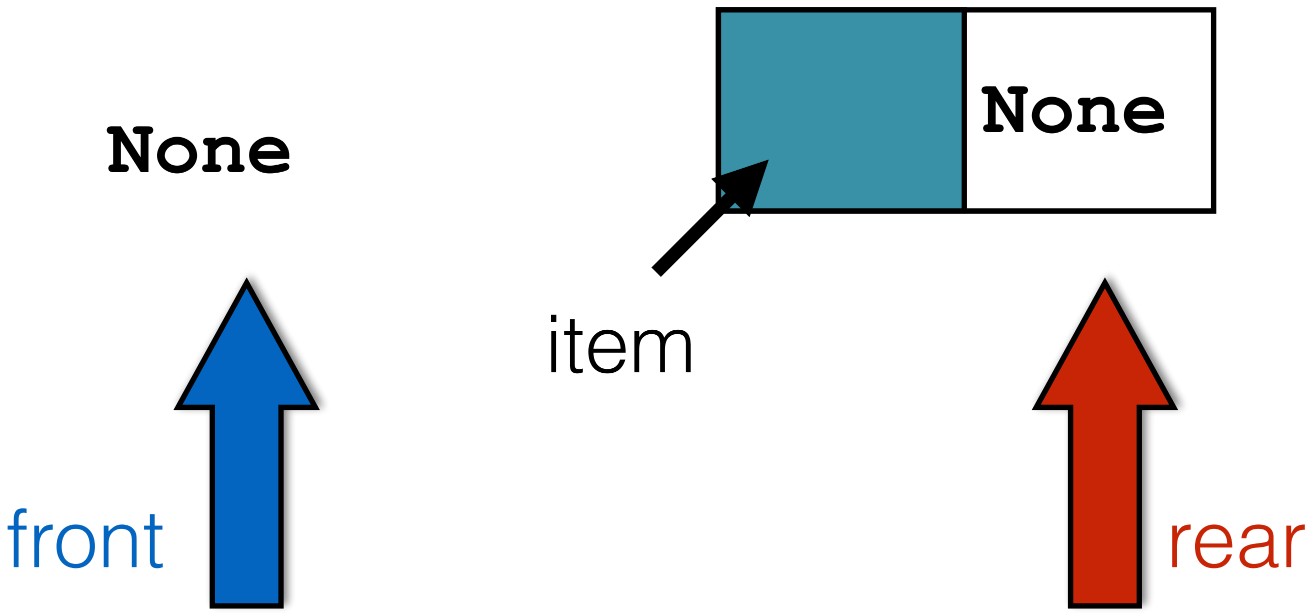
None

front

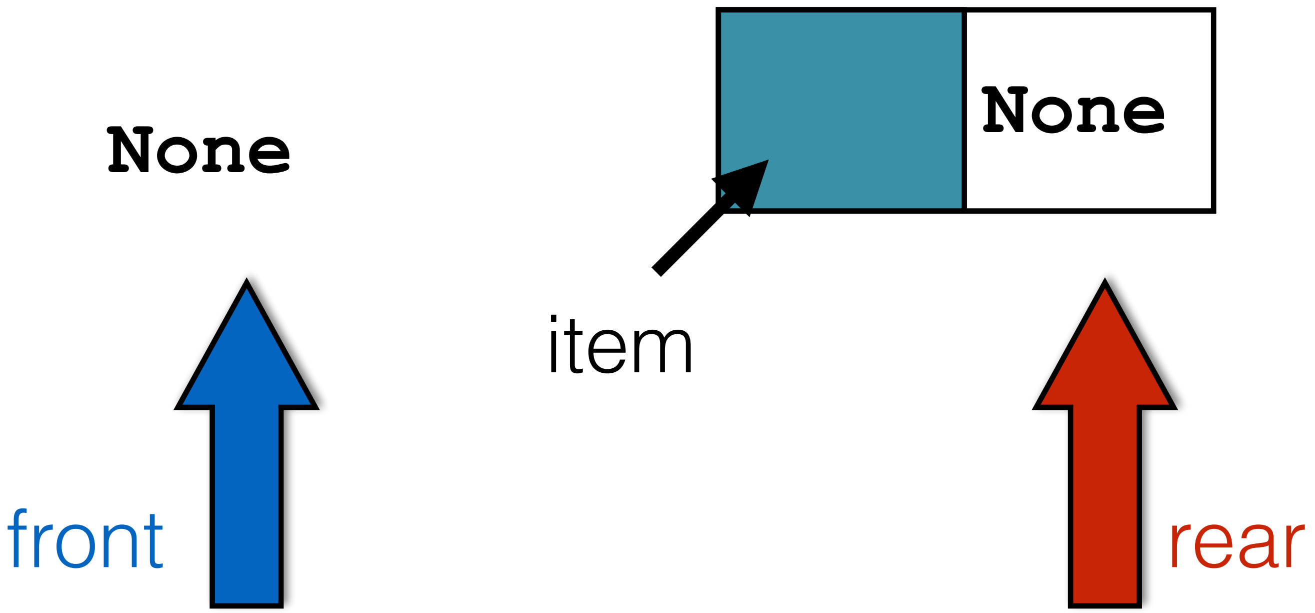
rear



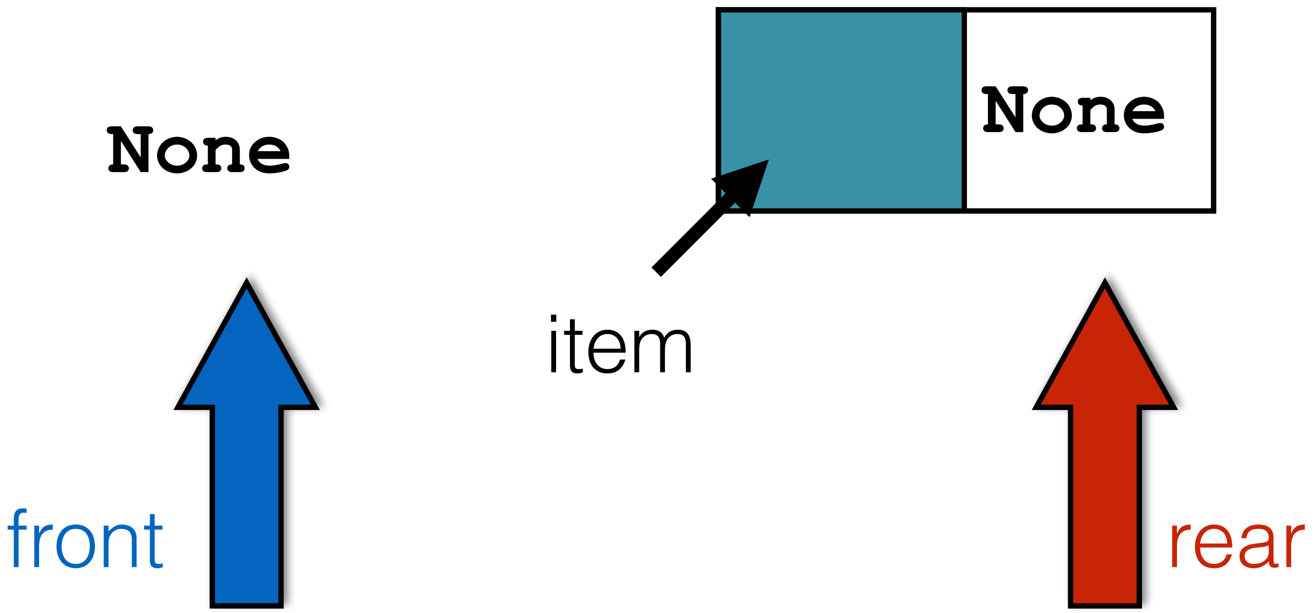
- Remember the item in the front node.



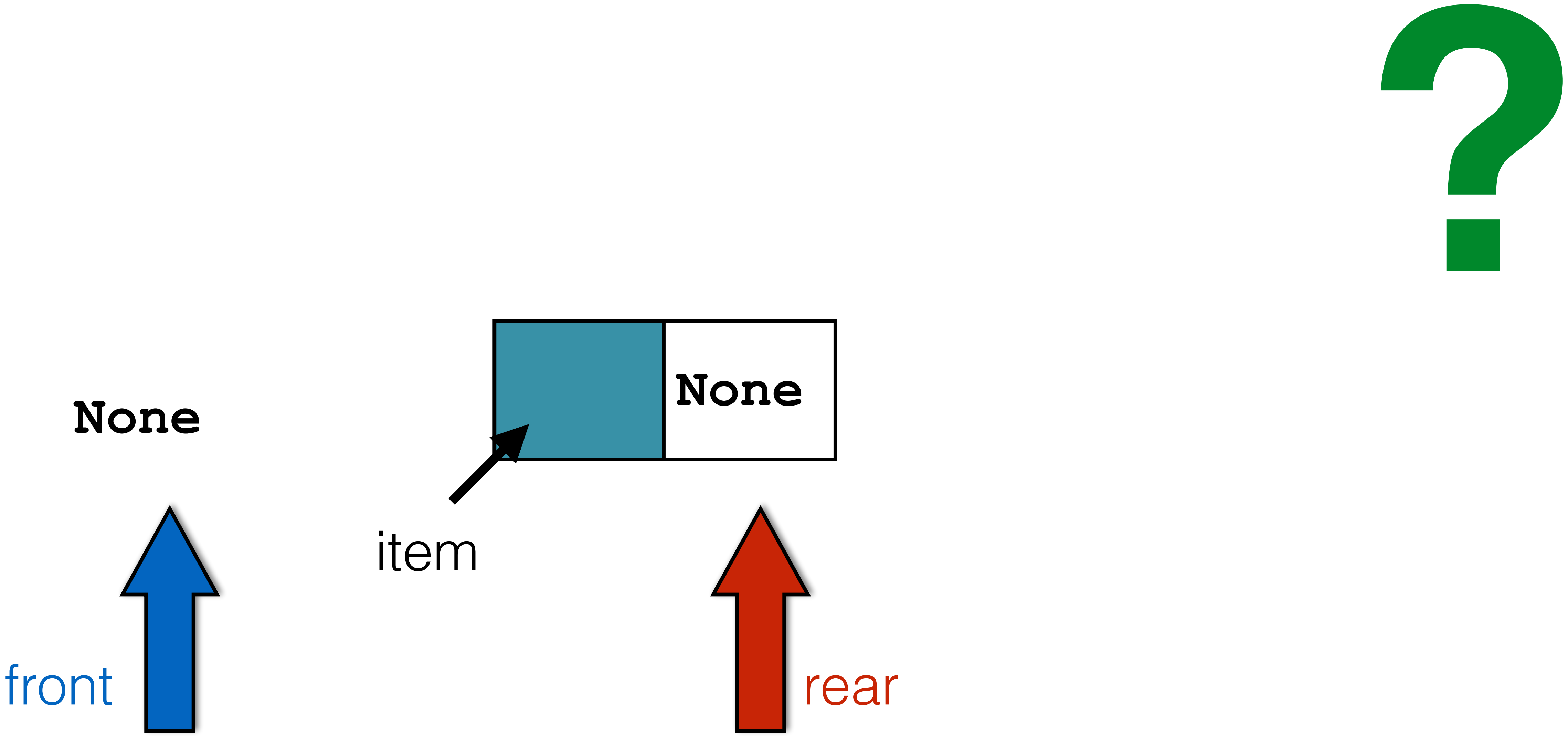
- Remember the item in the front node.
- Make the next node the new front



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

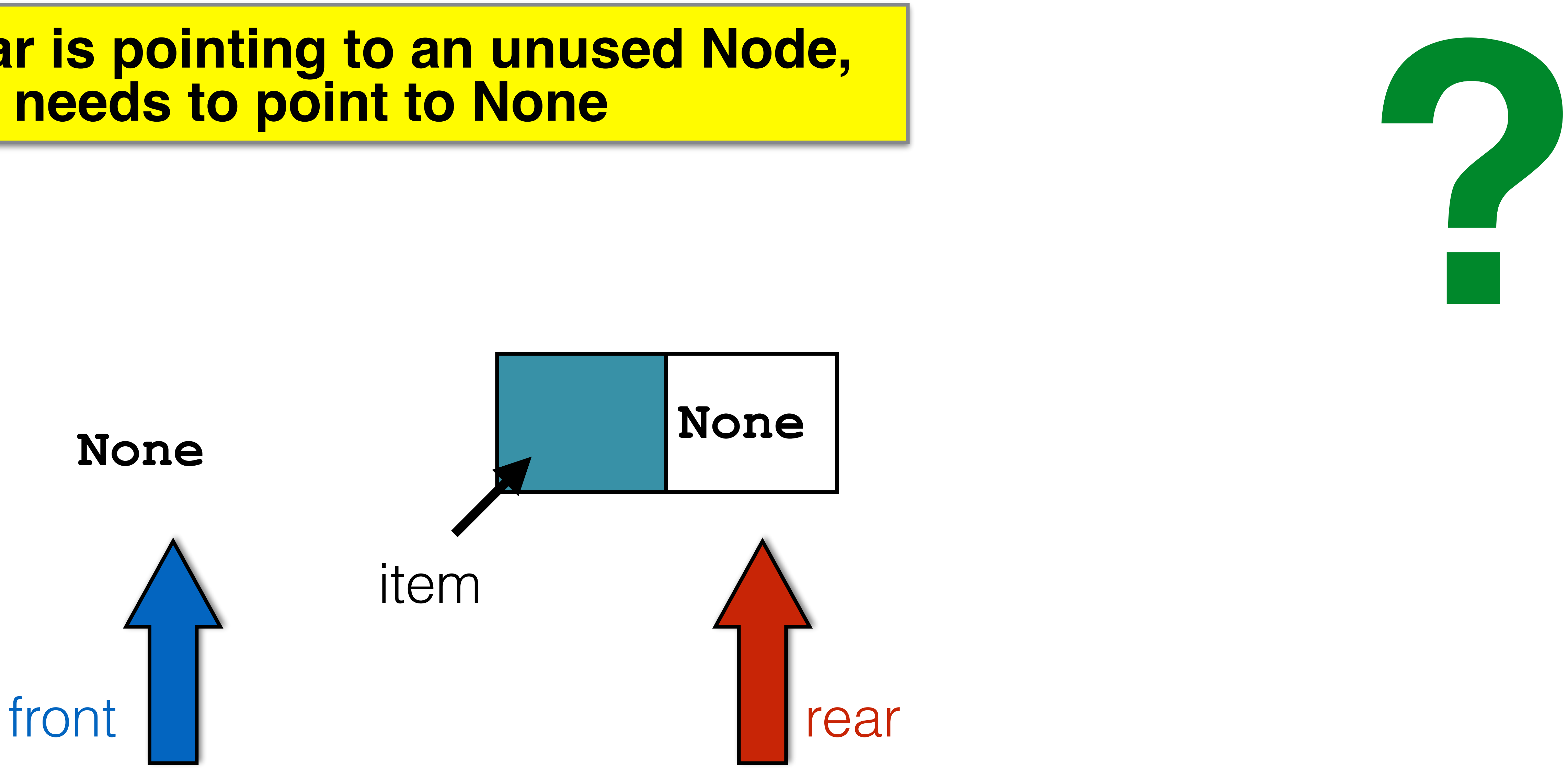


- Remember the item in the front node.
- Make the next node the new front
- Return the item

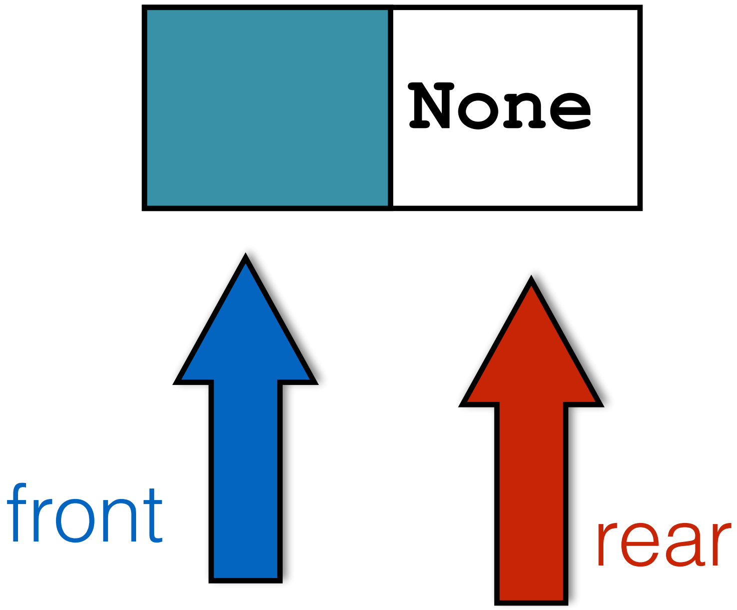


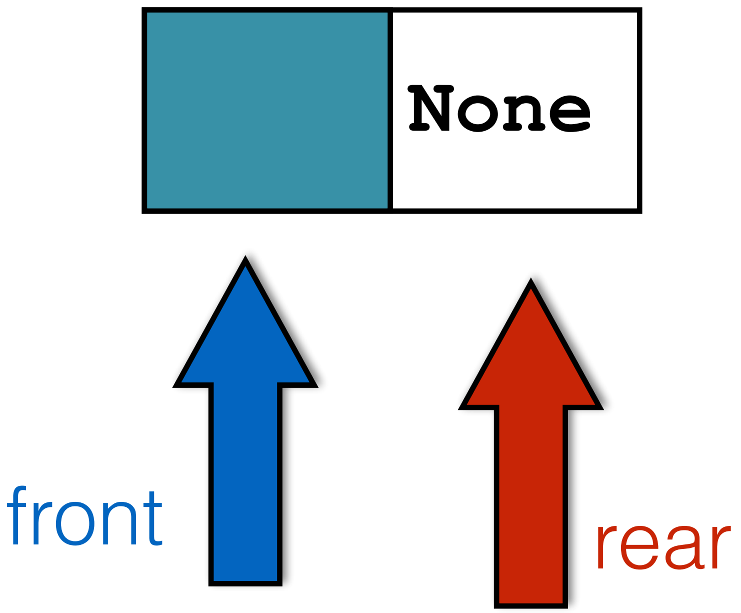
- Remember the item in the front node.
- Make the next node the new front
- Return the item

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

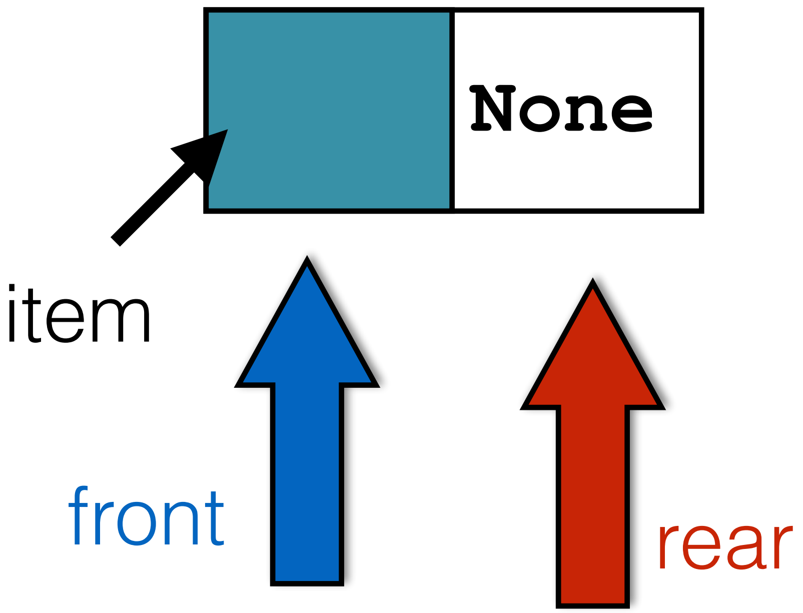


- Remember the item in the front node.
- Make the next node the new front
- Return the item

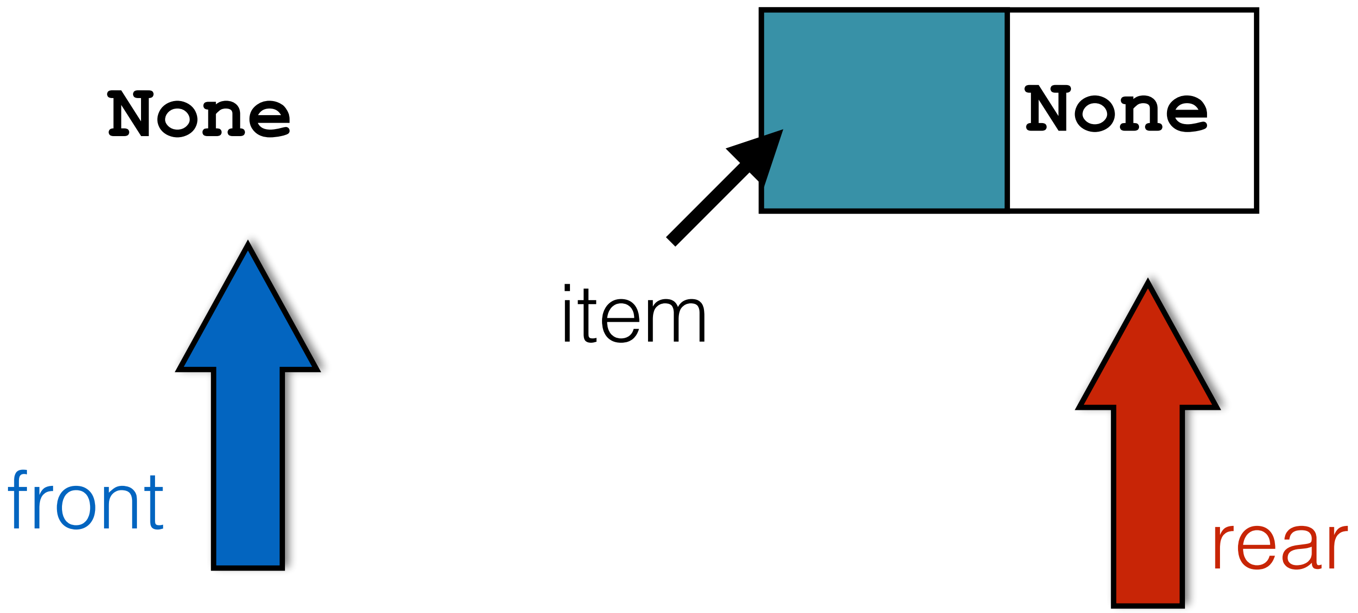

None
front
rear



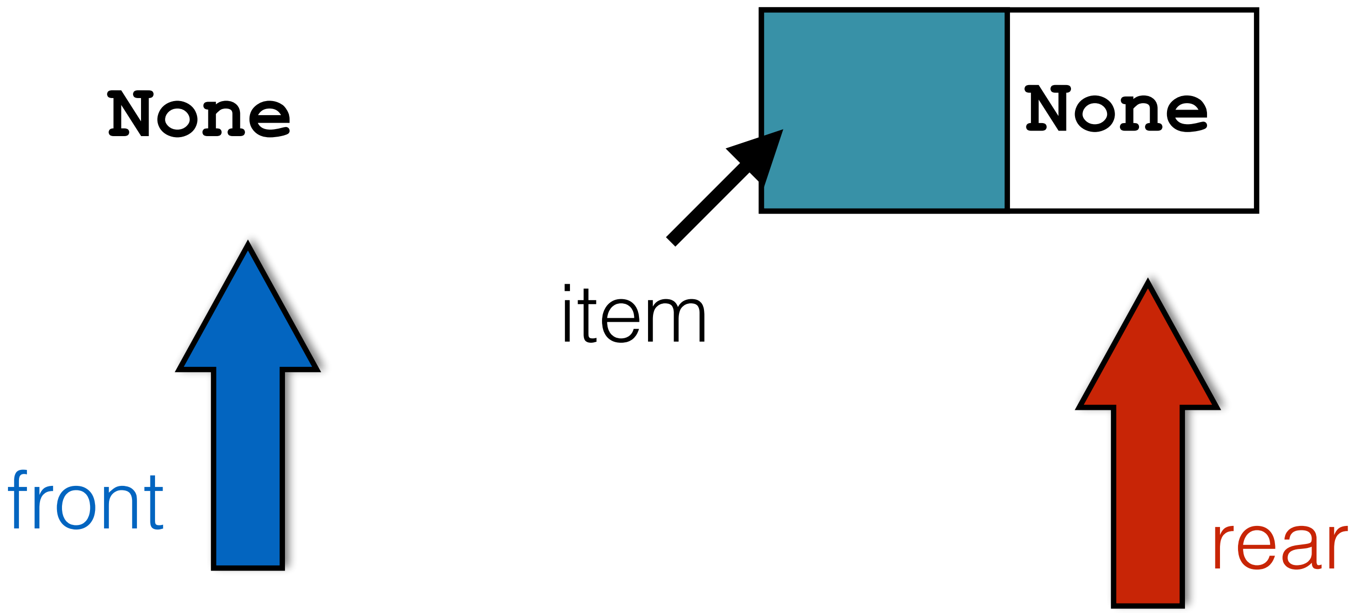
- If the queue is empty we raise an Exception



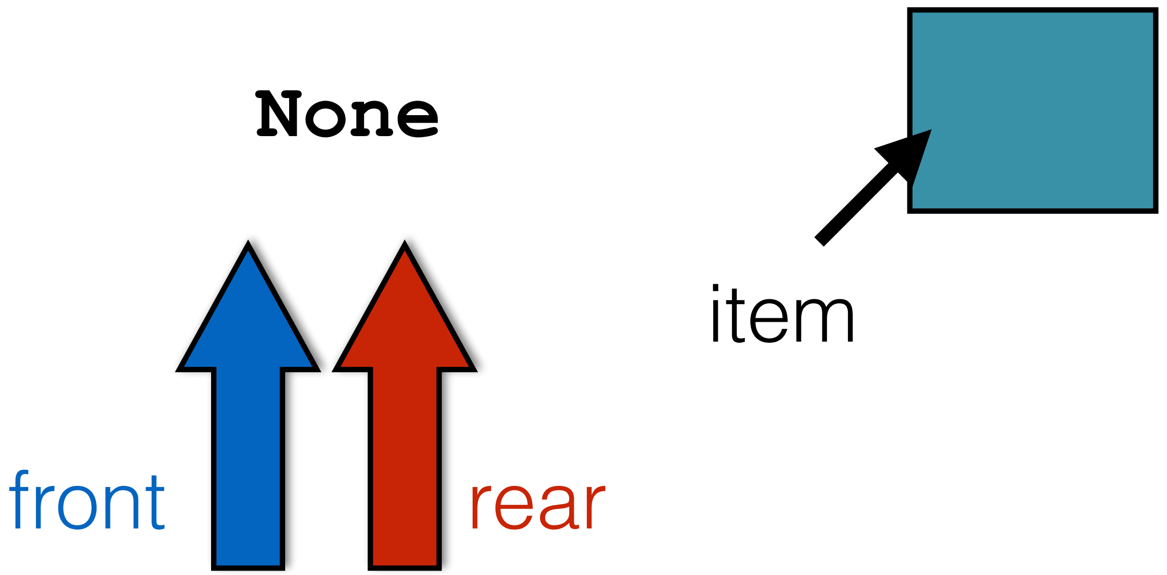
- If the queue is empty we raise an Exception
- Remember the item in the front node.



- If the queue is empty we raise an Exception
- Remember the item in the front node.
- Make the next node the new front

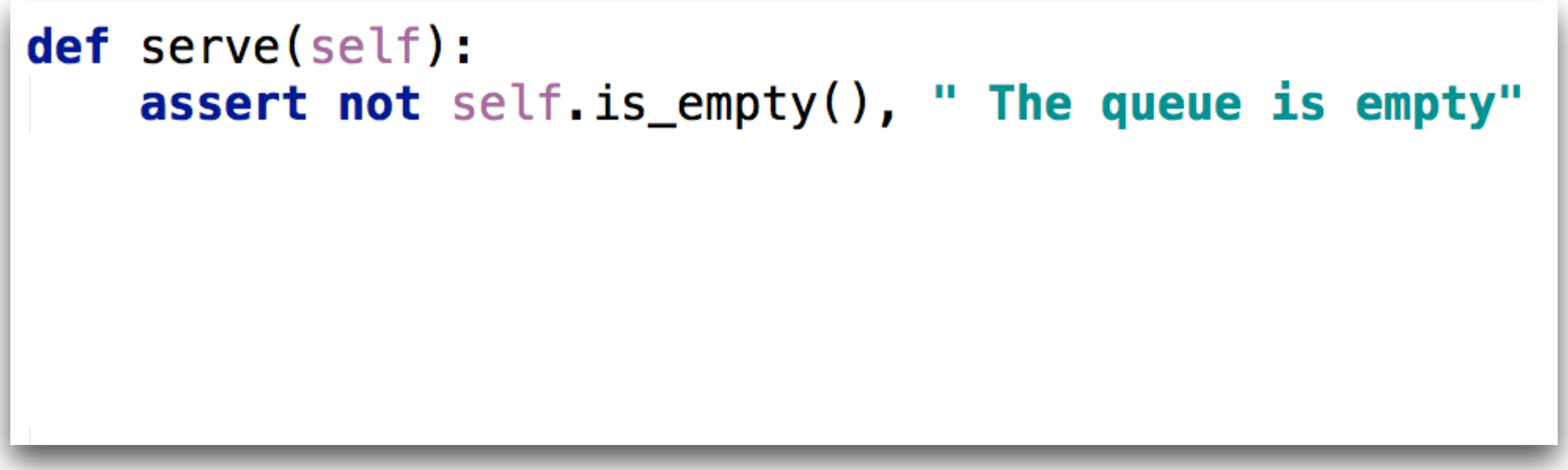


- If the queue is empty we raise an Exception
- Remember the item in the front node.
- Make the next node the new front
- If front is pointing to None (i.e., queue is <u>now</u> empty)
  - Point rear to None



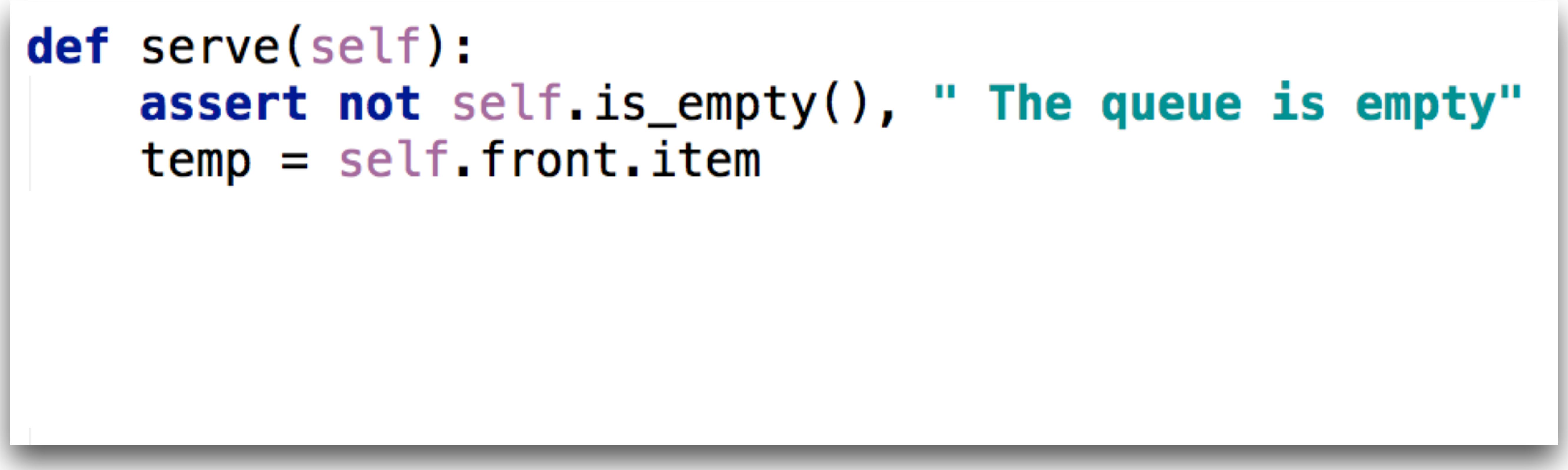
- If the queue is empty we raise an Exception
- Remember the item in the front node.
- Make the next node the new front
- If front is pointing to None (i.e., queue is <u>now</u> empty)
    - Point rear to None
- Return the item

```python
def serve(self):
```



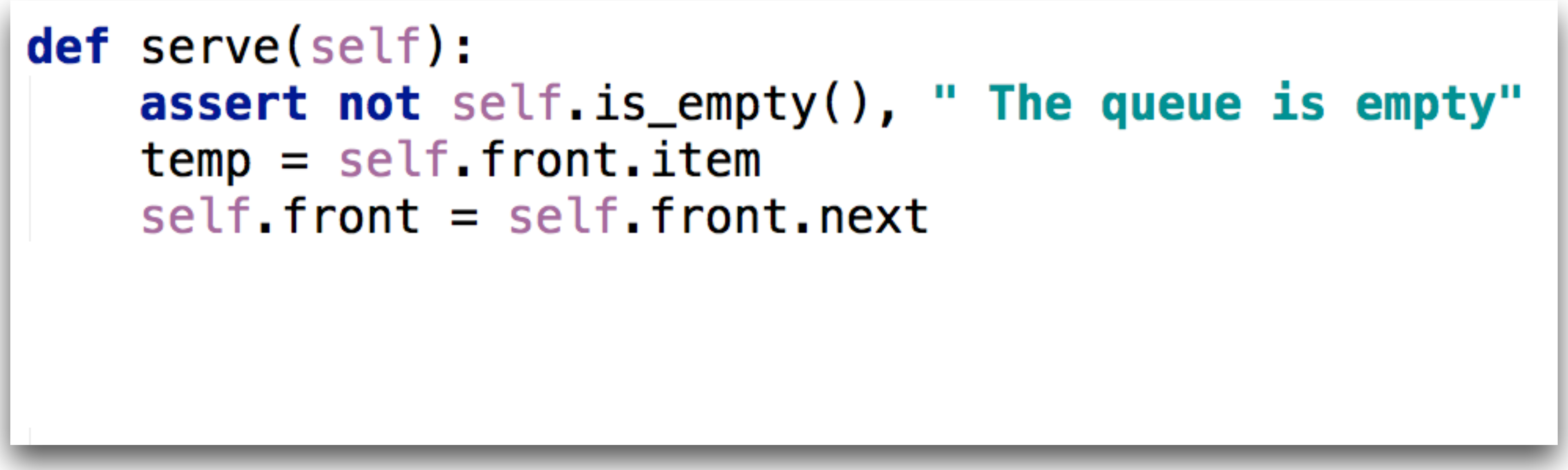
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
```

- If the queue is empty we raise an Exception



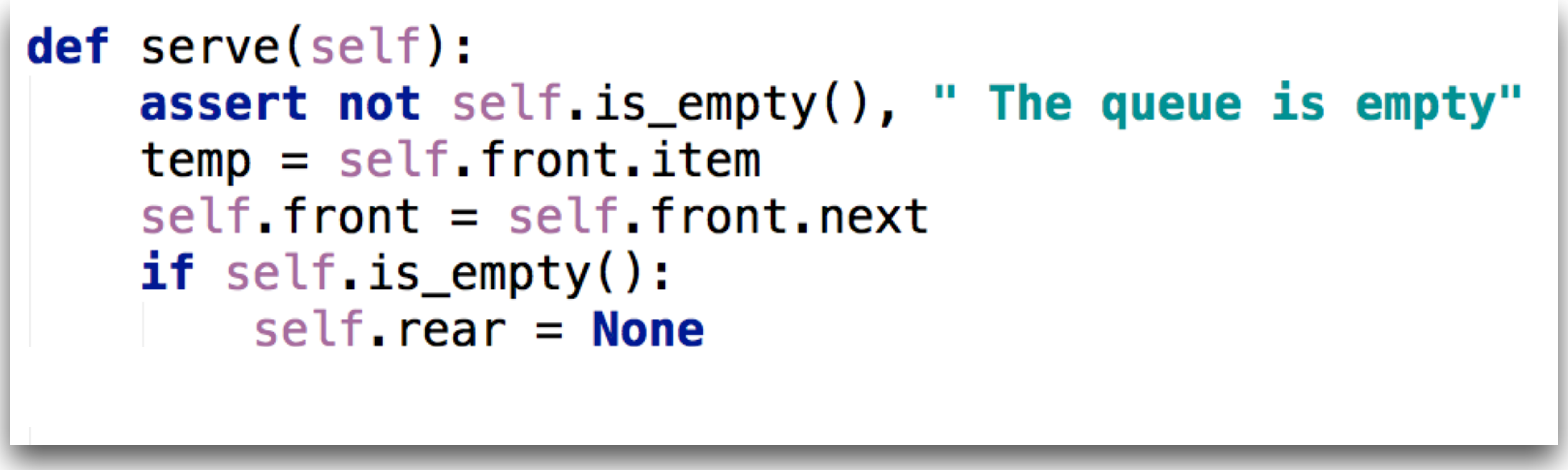
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
```

- If the queue is empty we raise an Exception
- Remember the item in the front node.



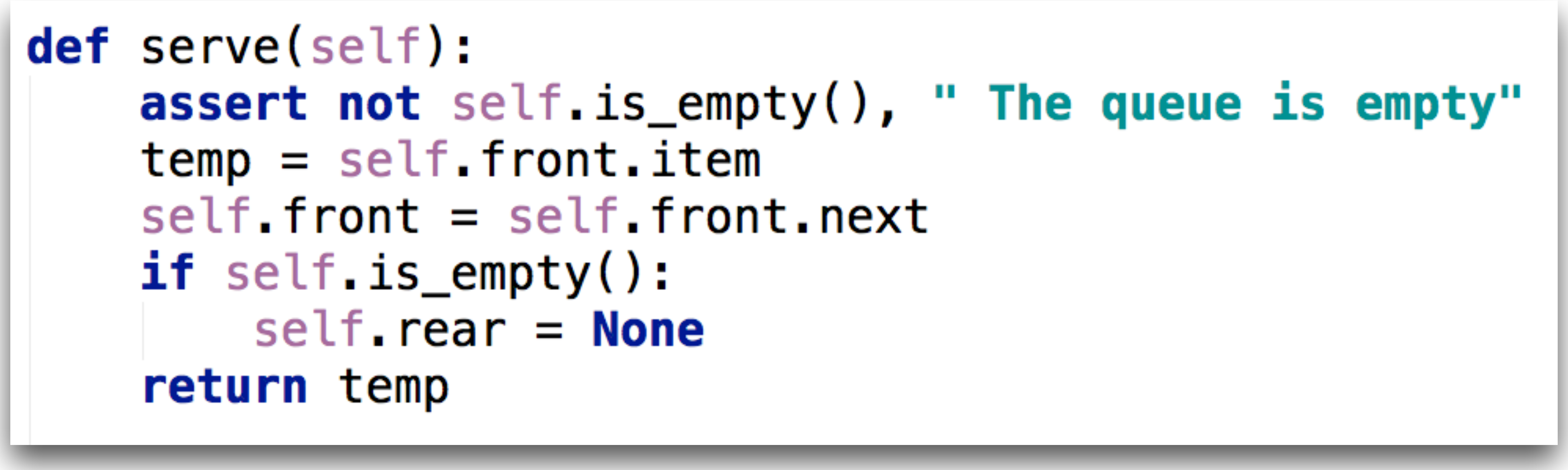
```python
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
```

- If the queue is empty we raise an Exception
- Remember the item in the front node.
- Make the next node the new front



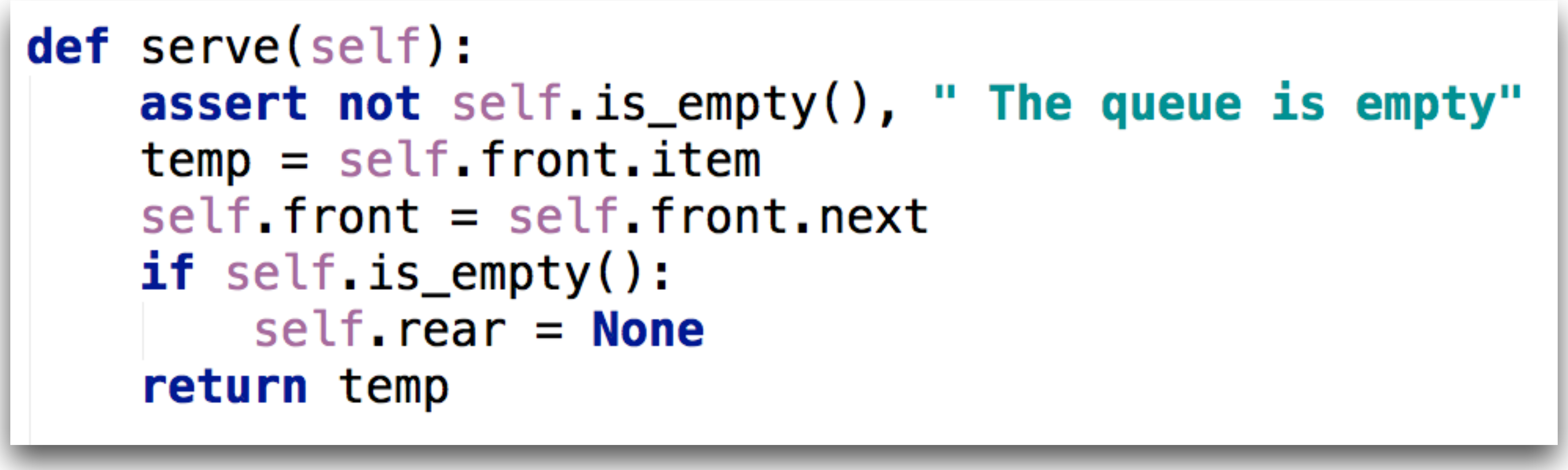
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
```

- 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

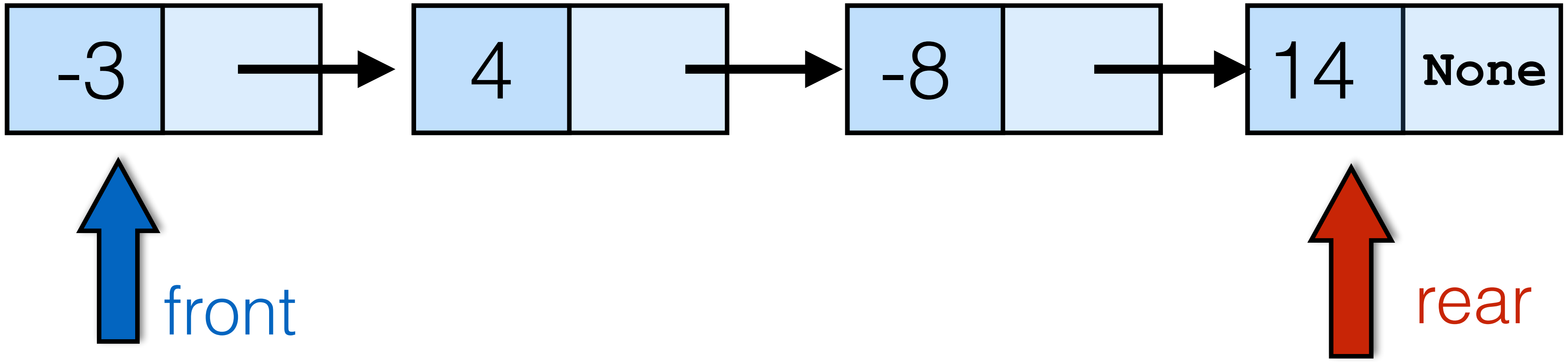


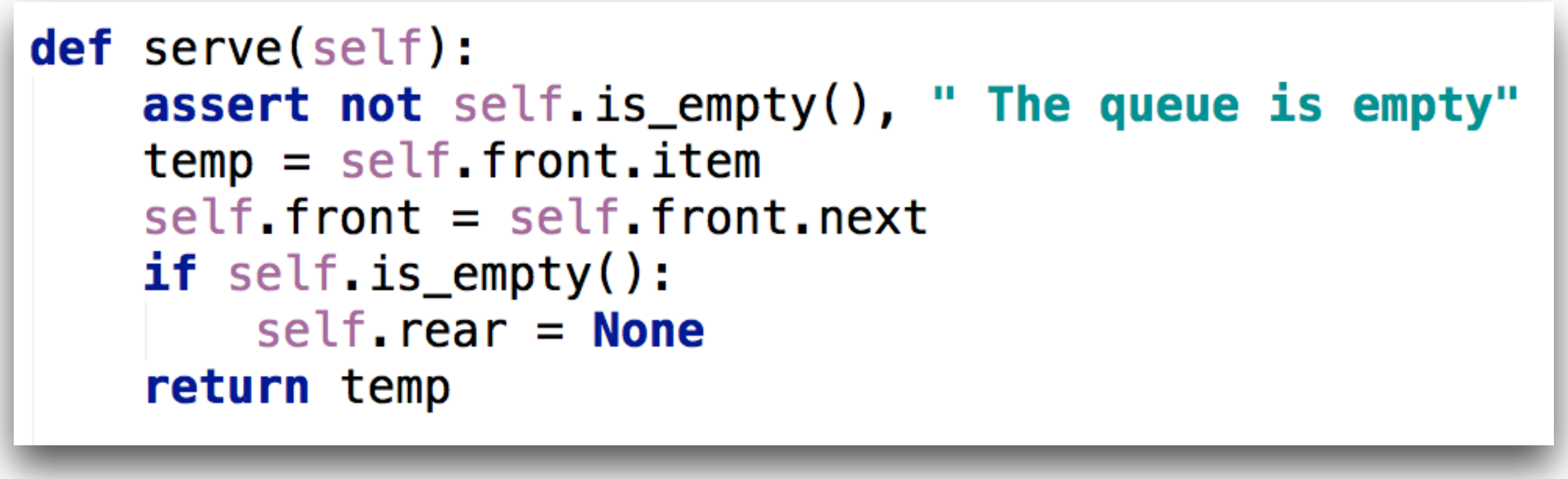
```python
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```

- 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

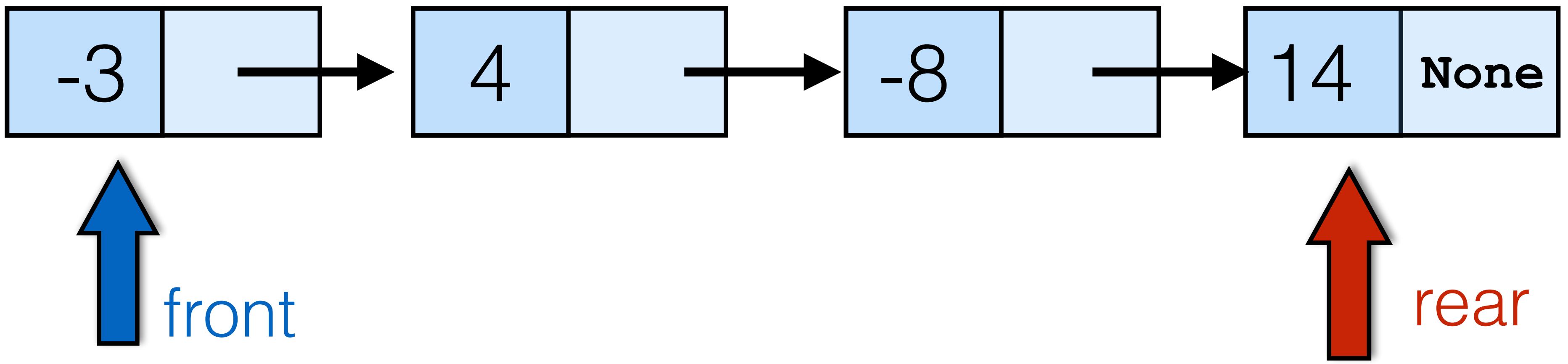


```python
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```

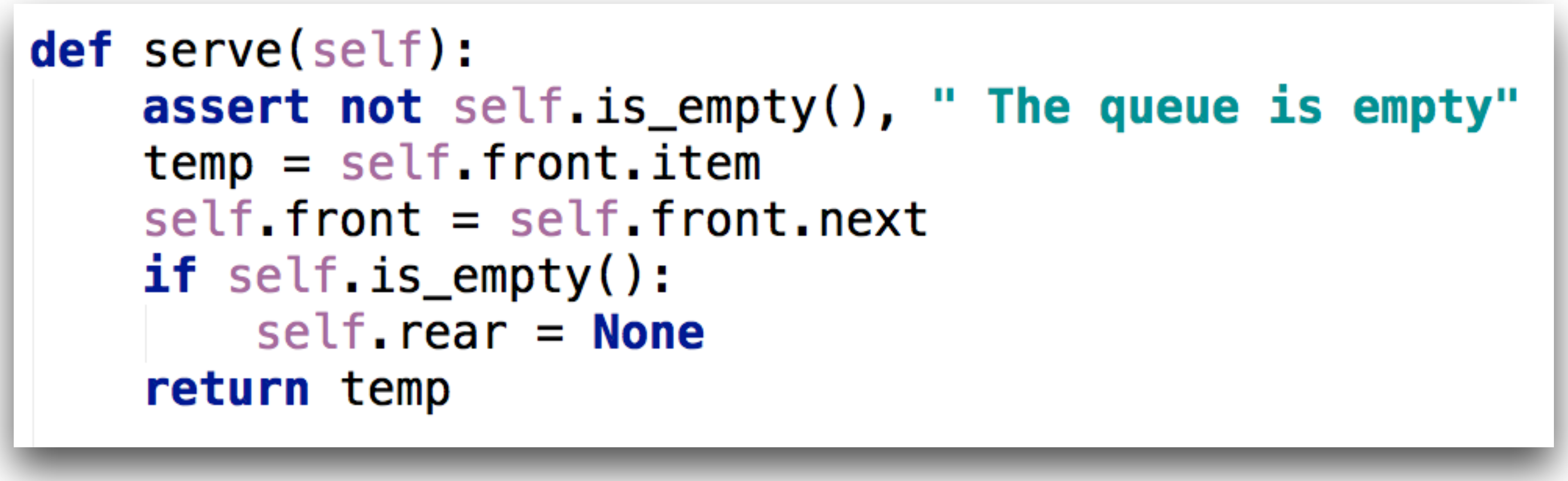




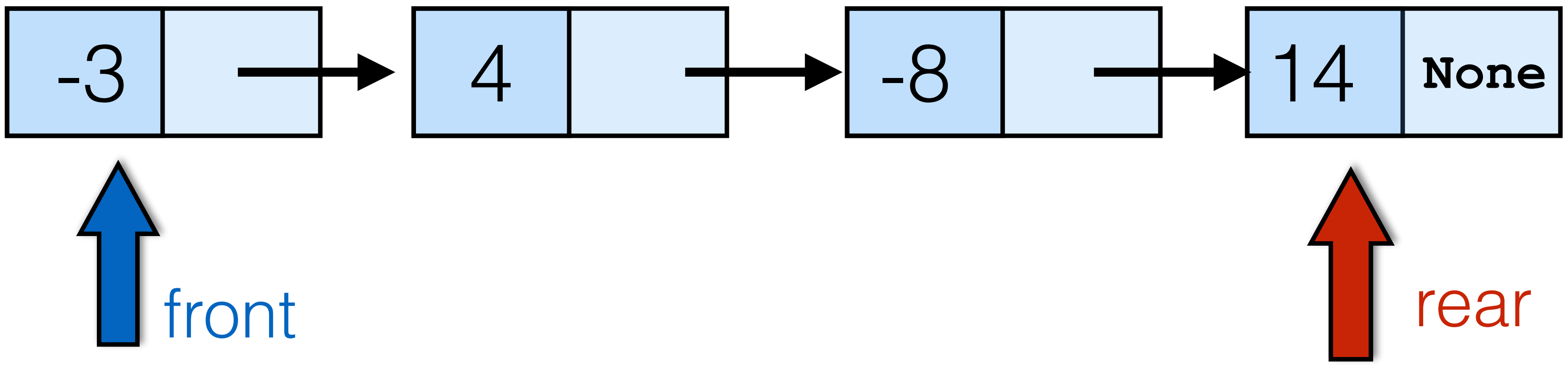
```python
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



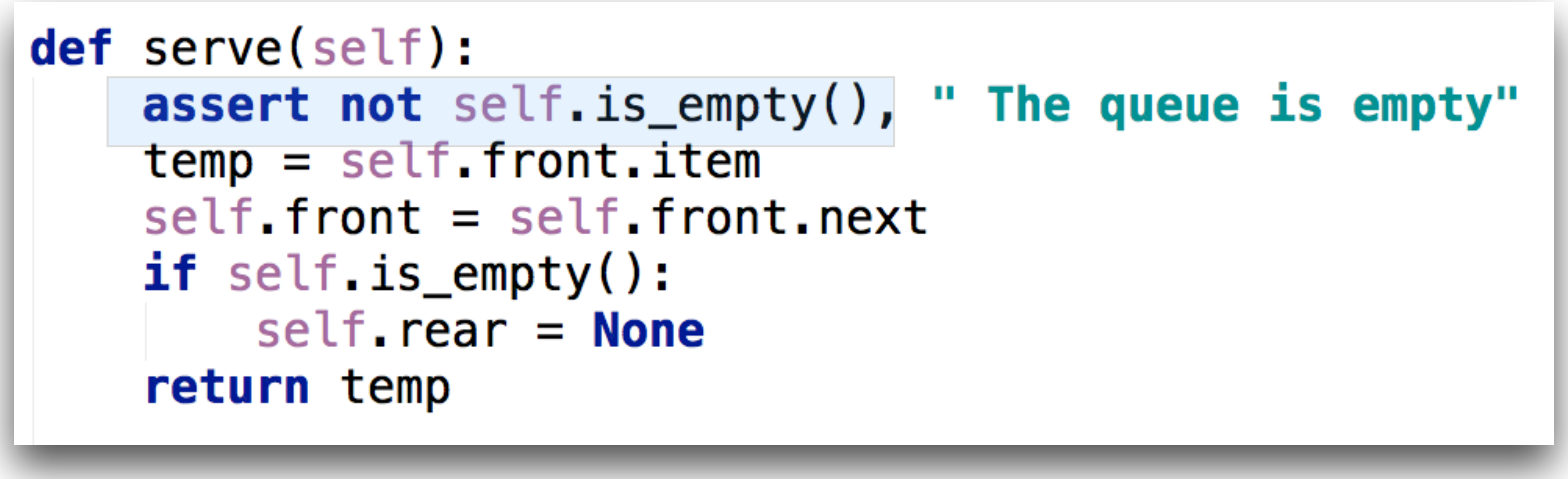
```
q.serve()
```



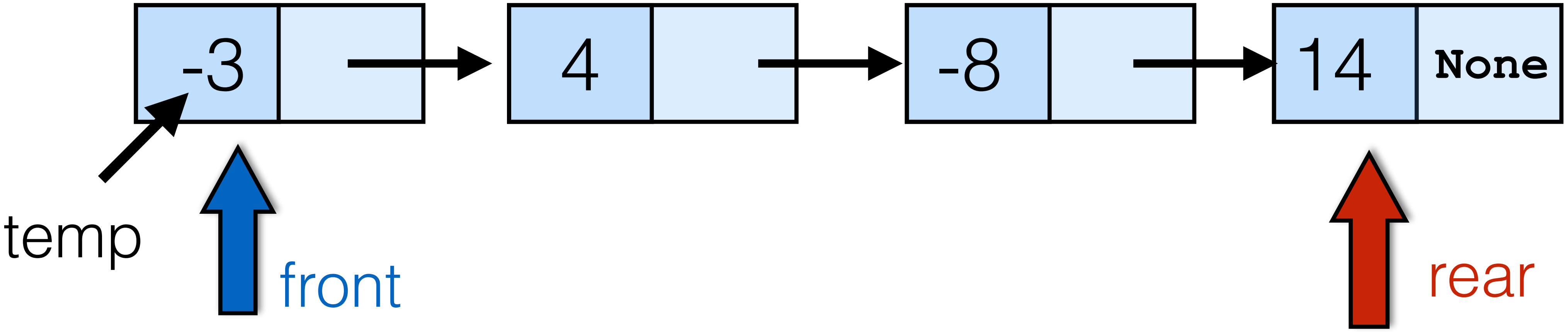
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



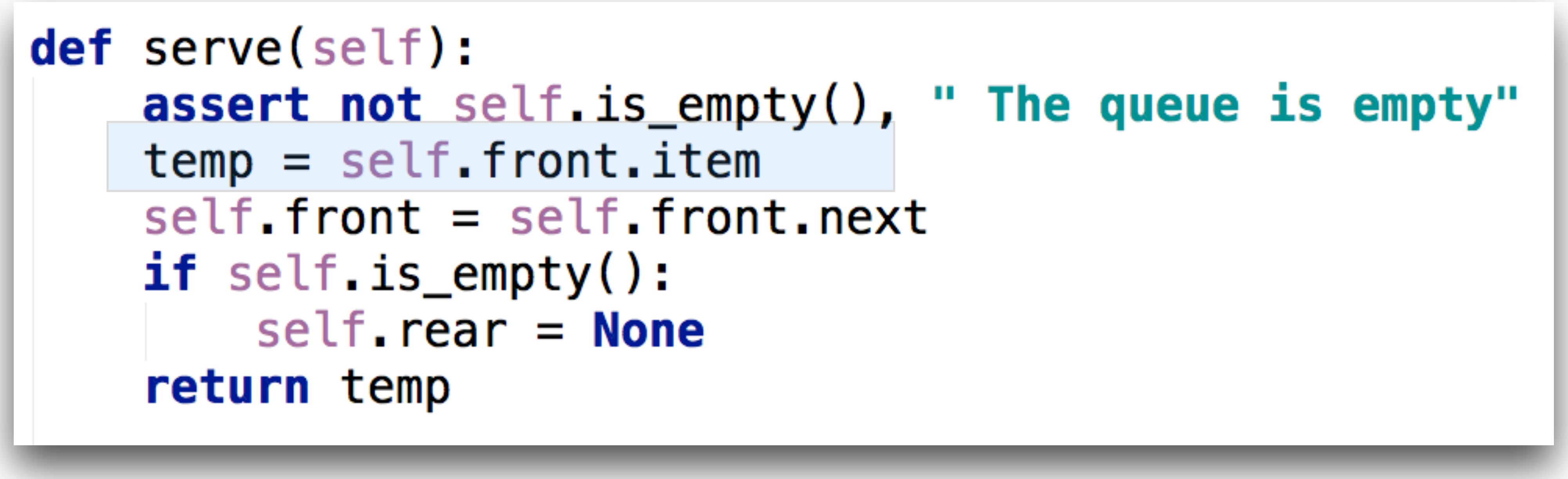
q.serve()



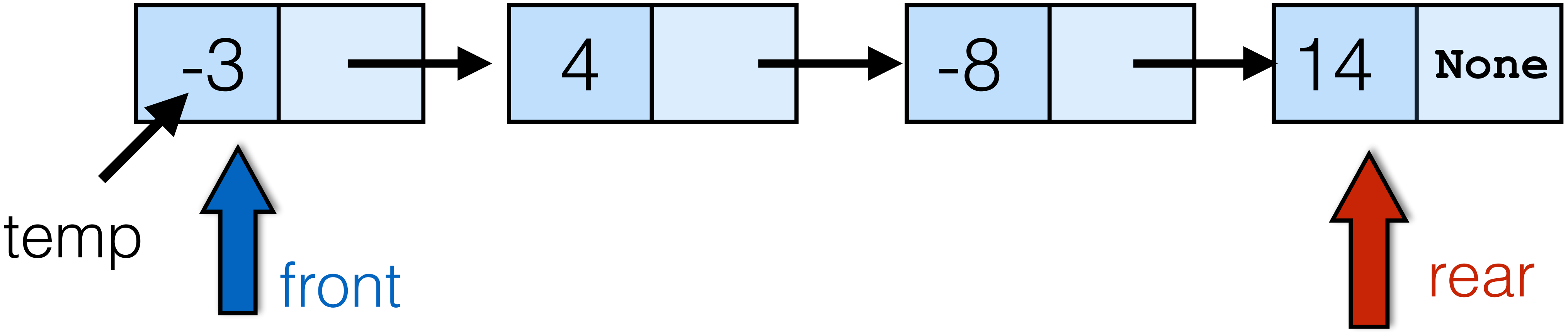
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



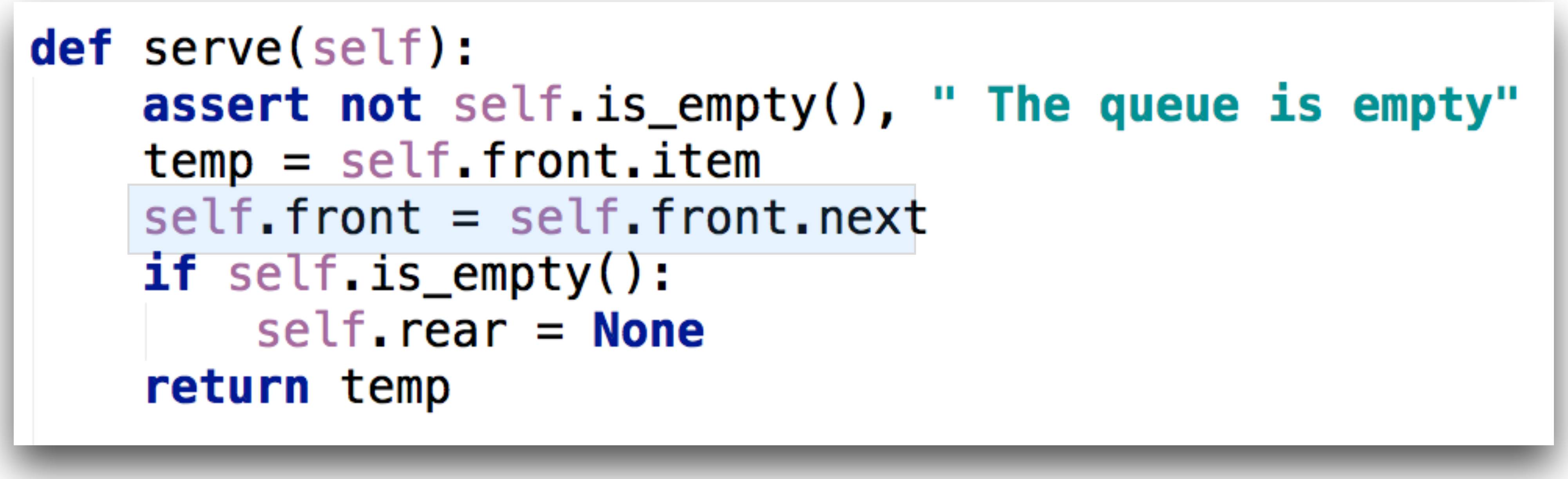
q.serve()



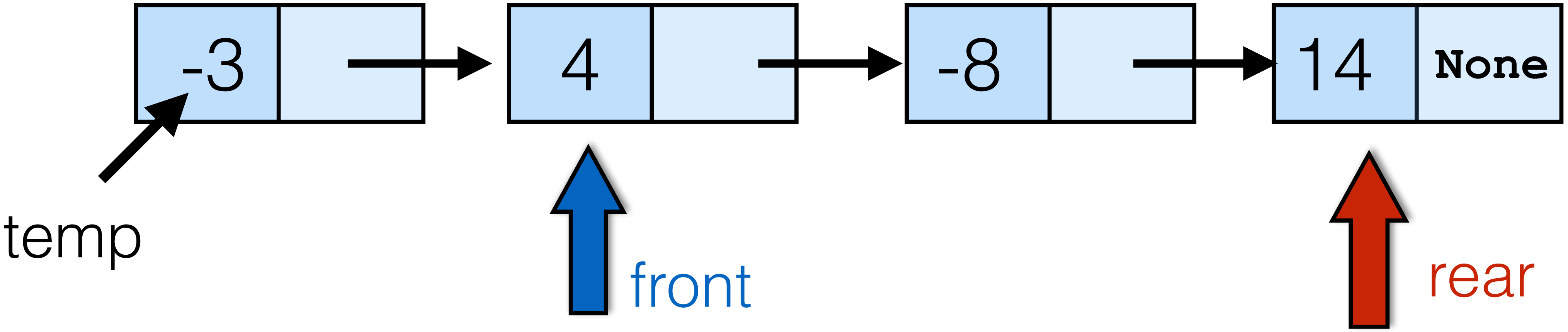
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



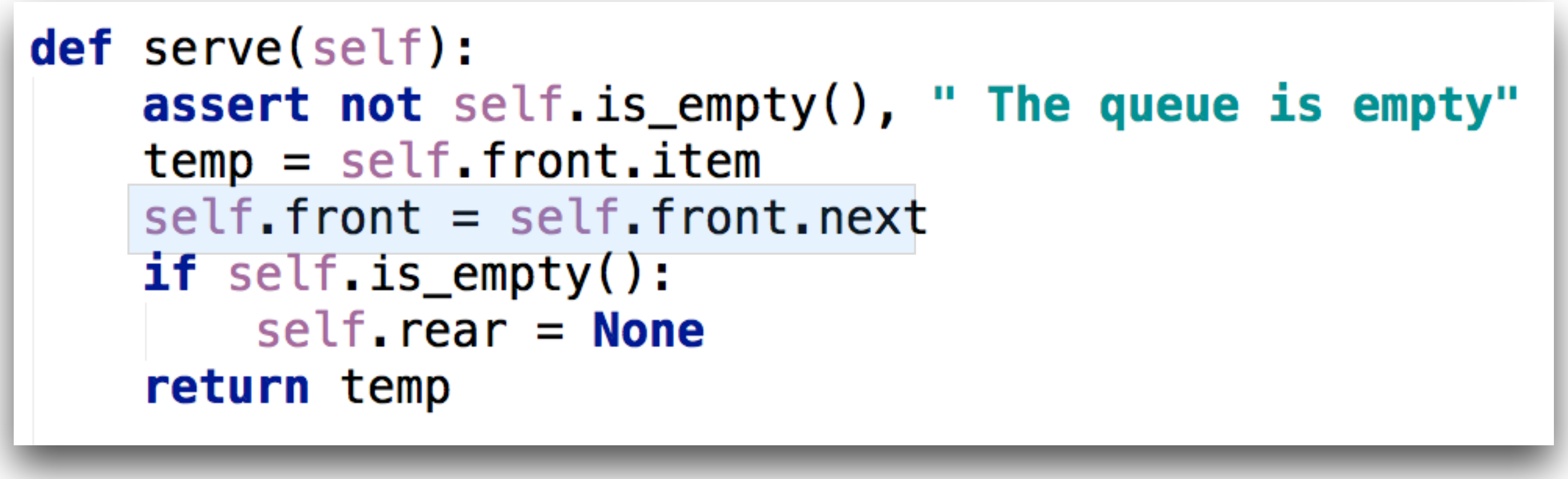
q.serve()



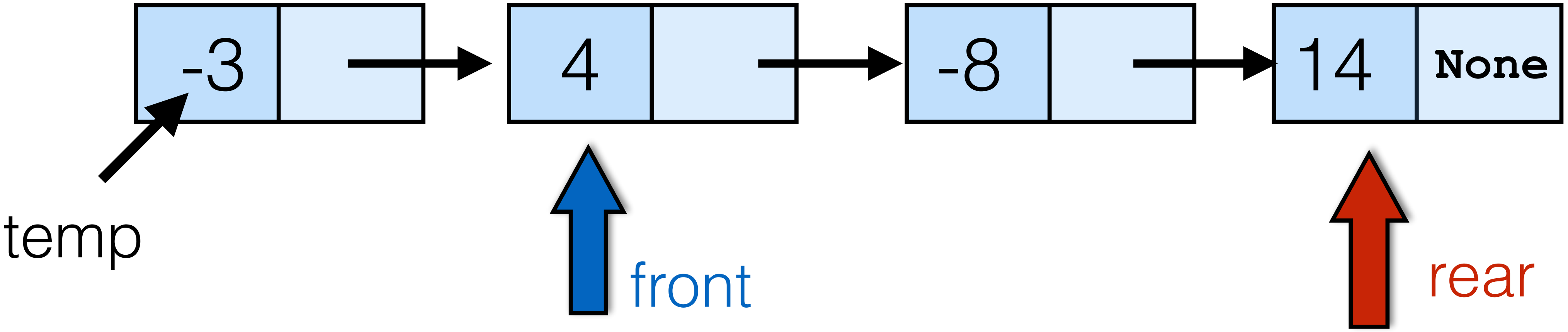
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



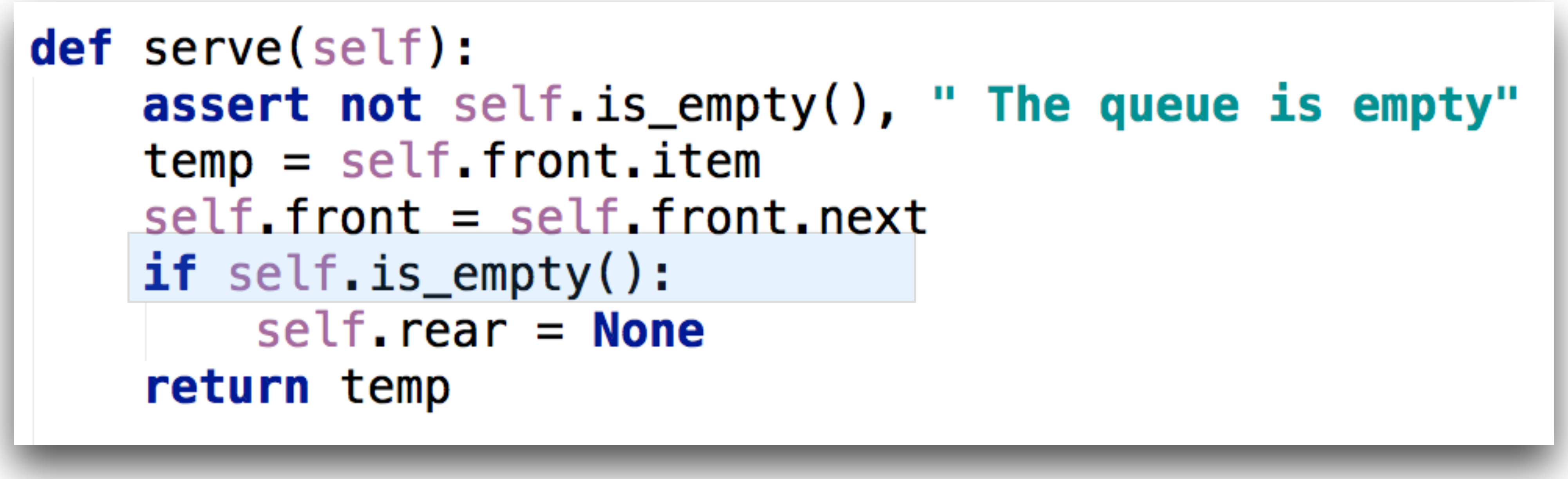
q.serve()



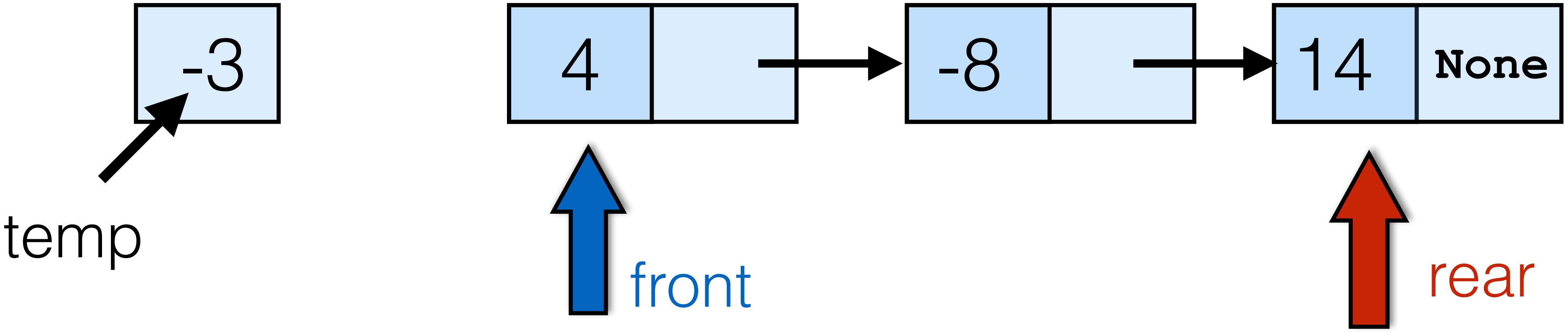
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



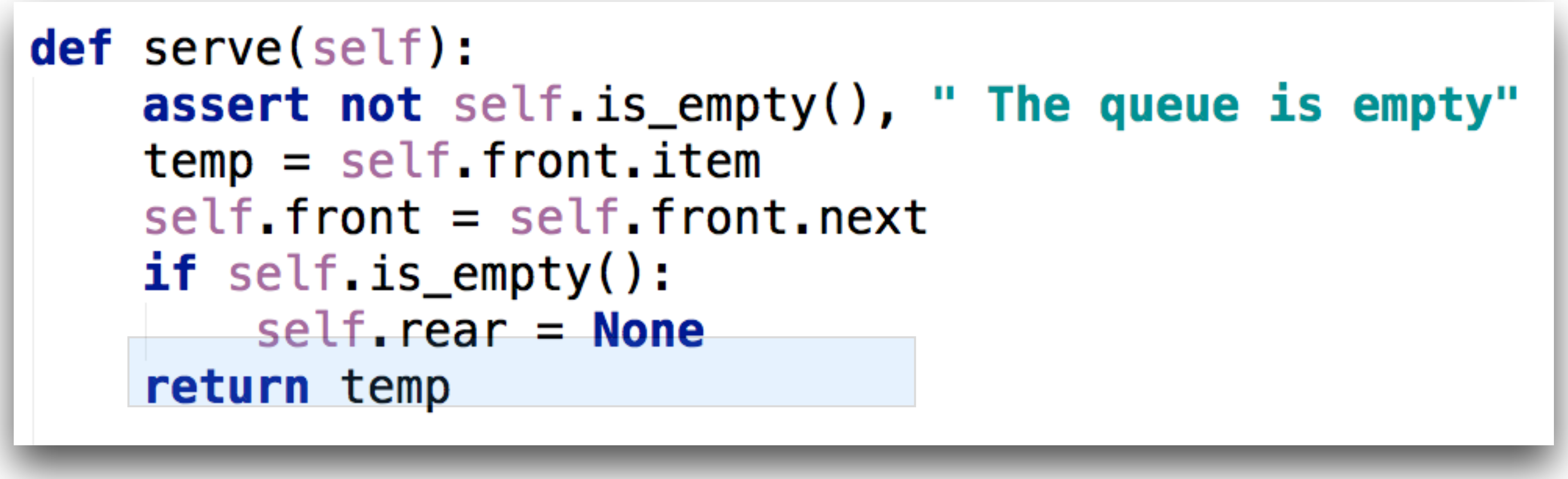
```
q.serve()
```



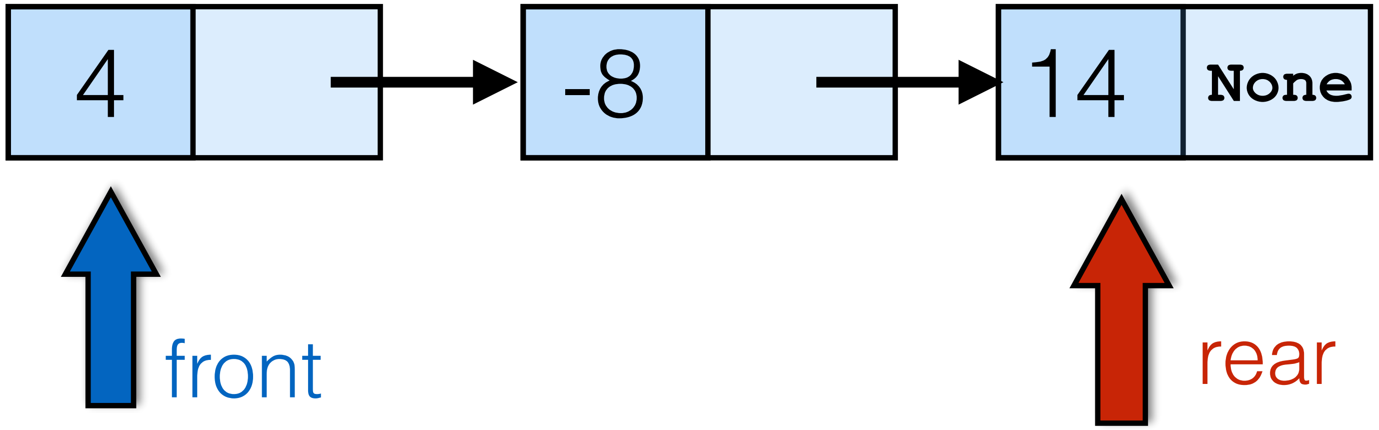
```python
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



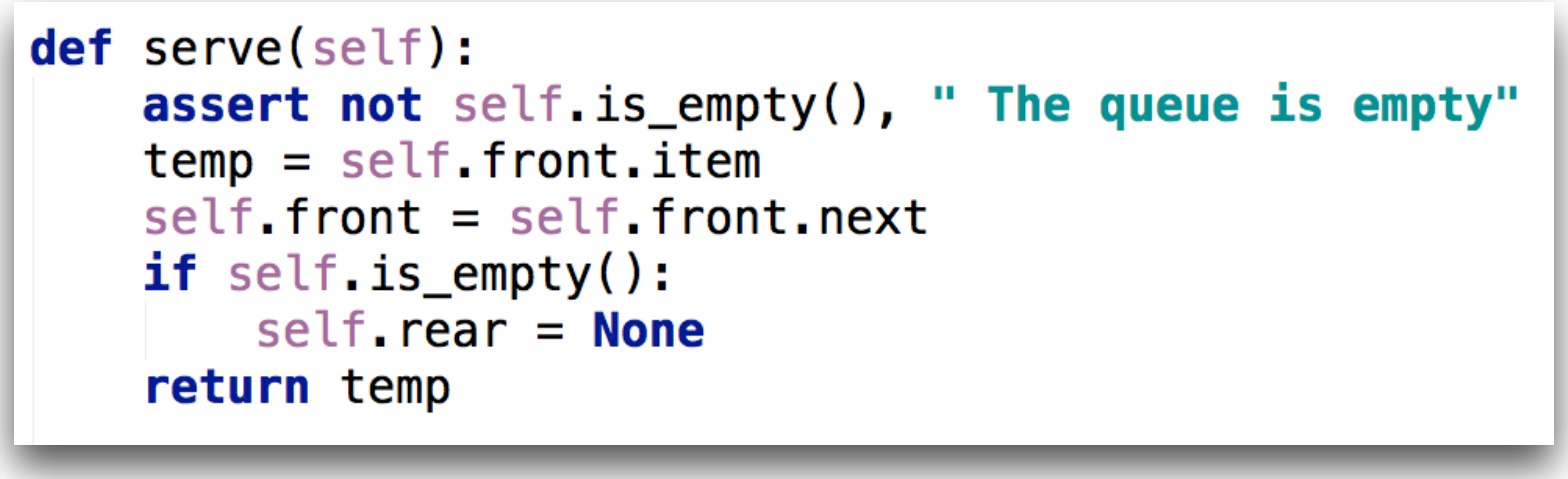
q.serve()



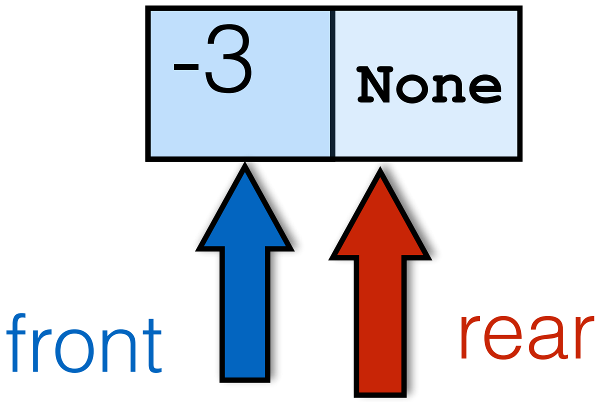
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```

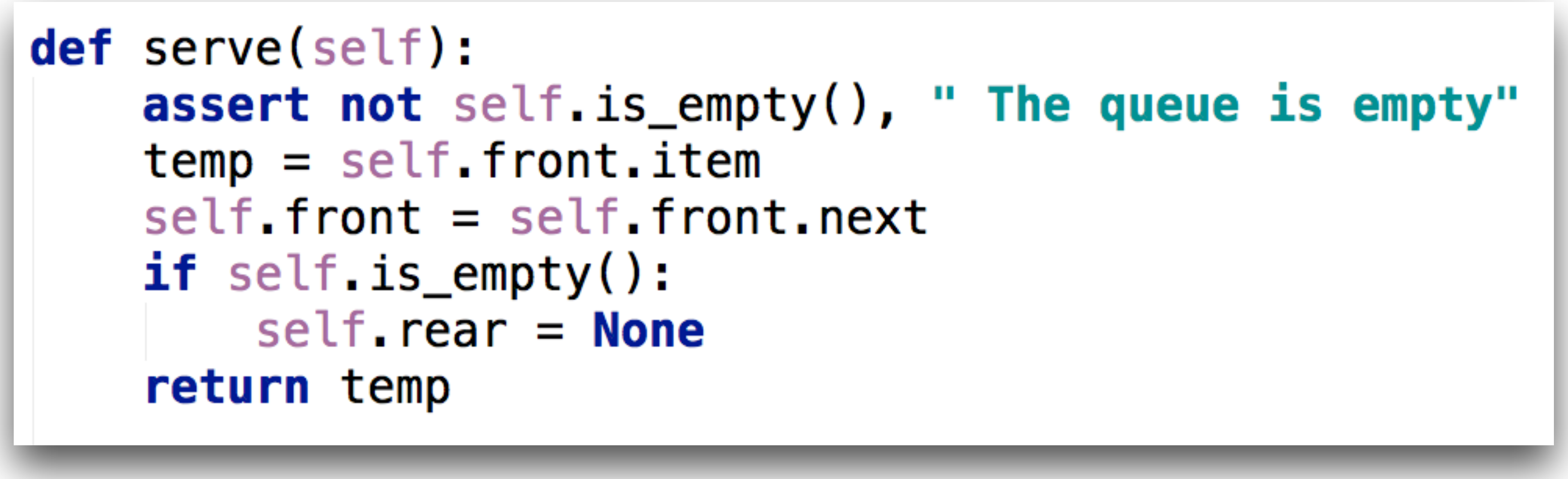


q.serve()

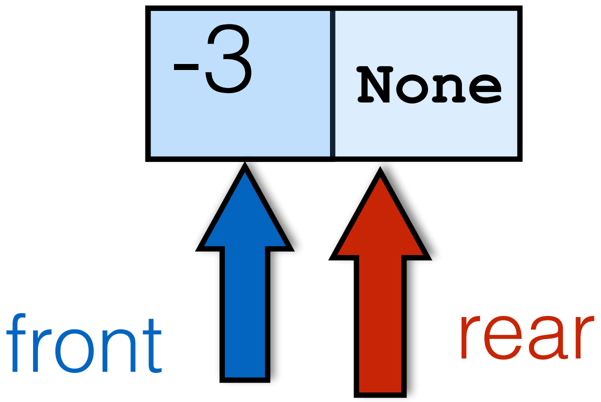


```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```

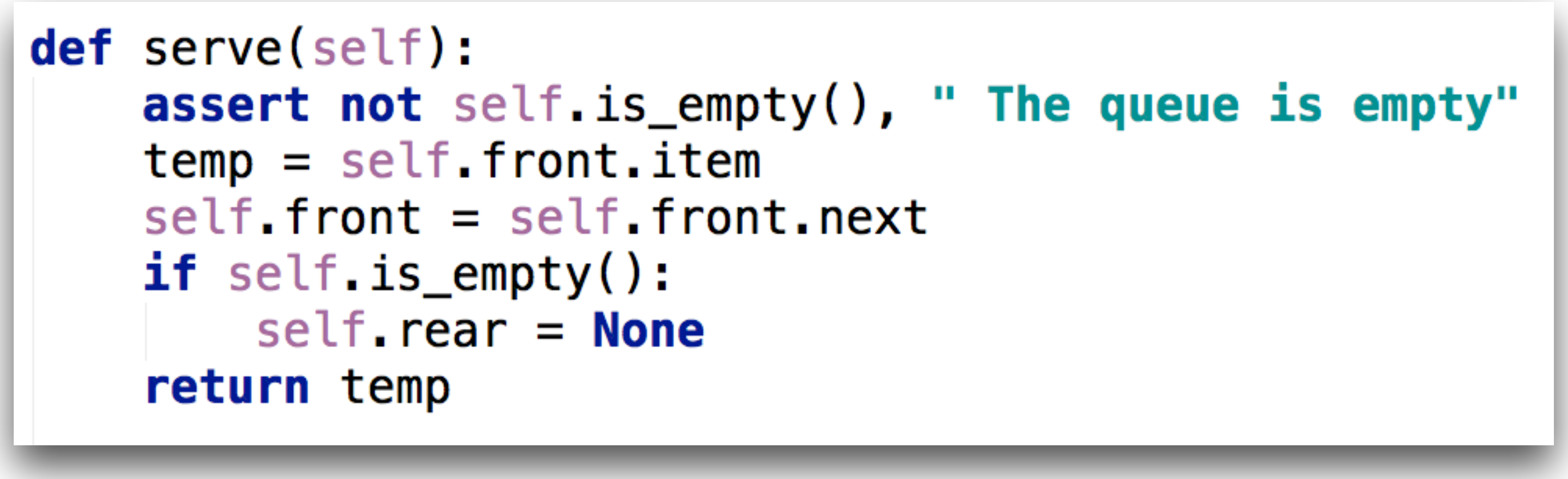




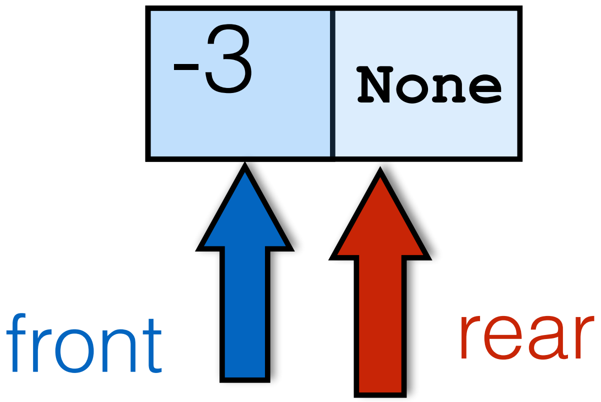
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



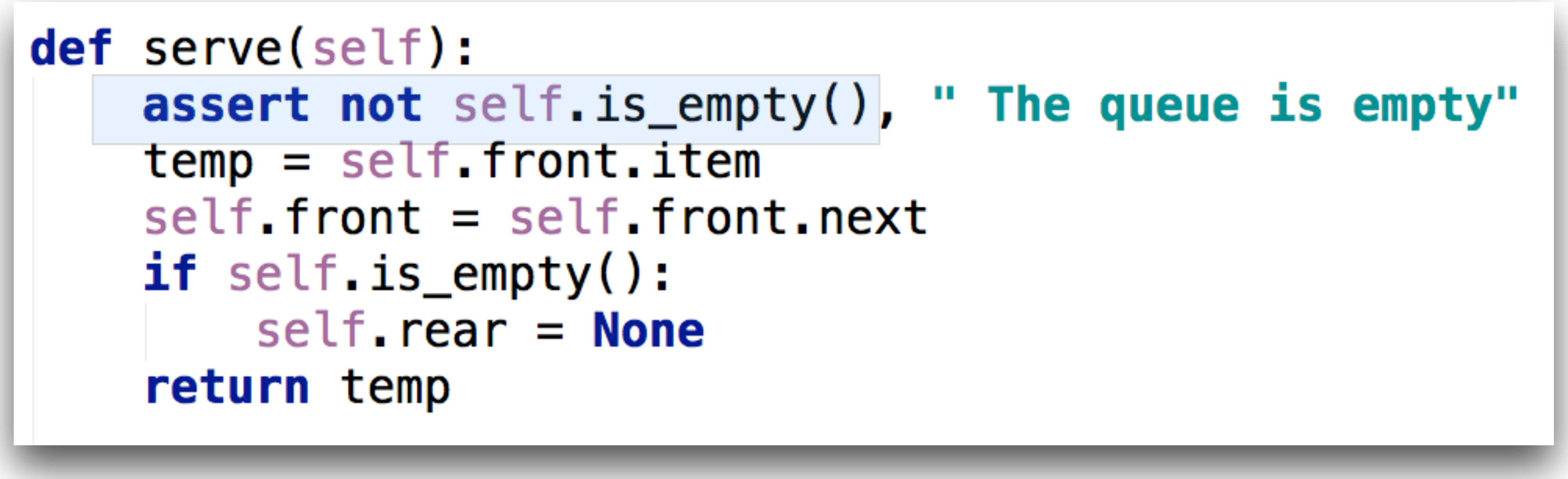
q.serve()



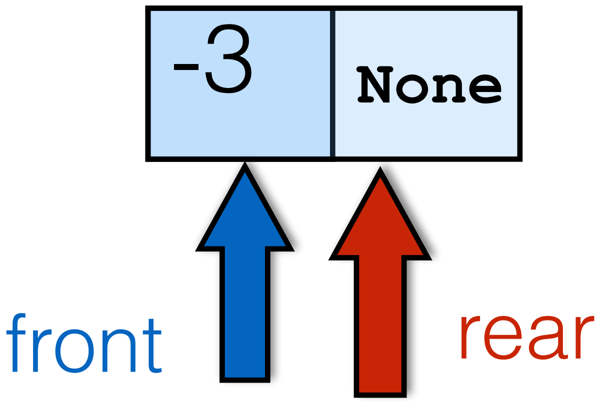
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



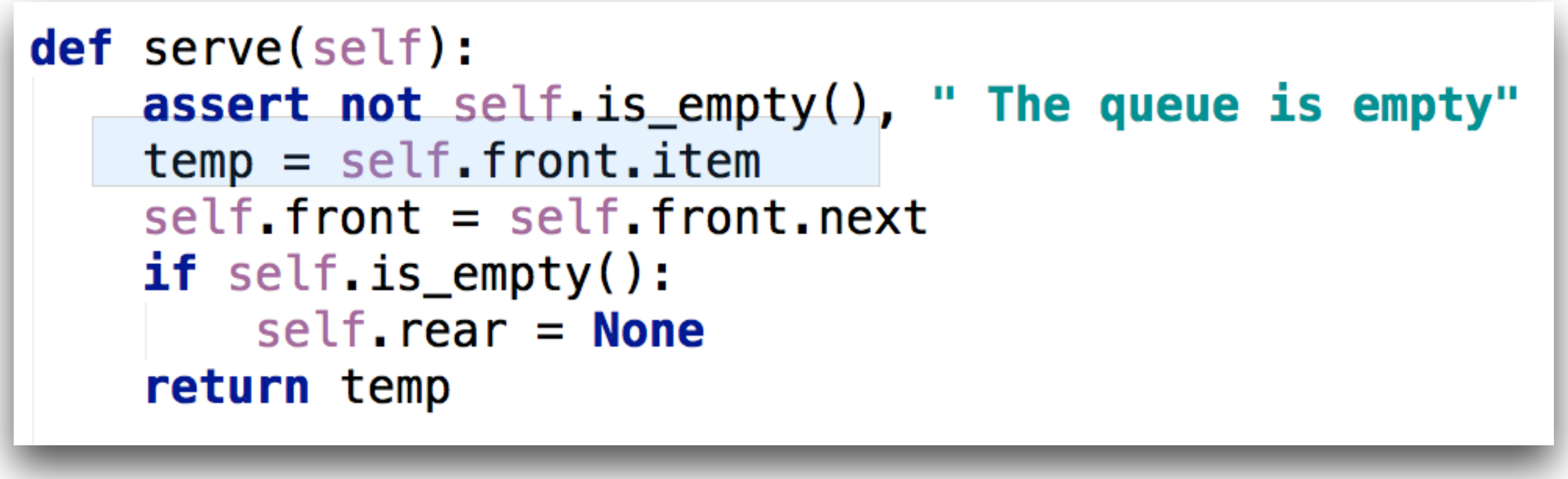
q.serve()



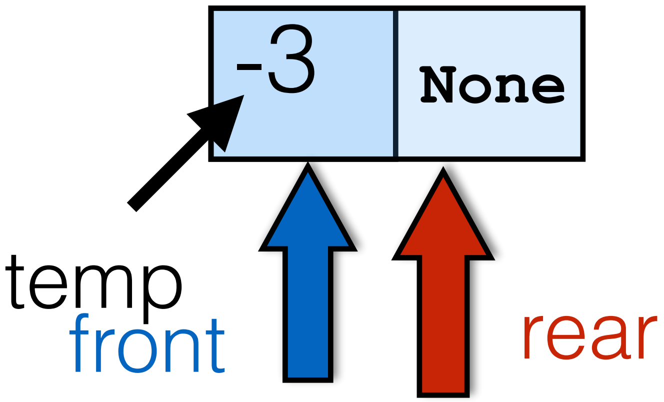
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



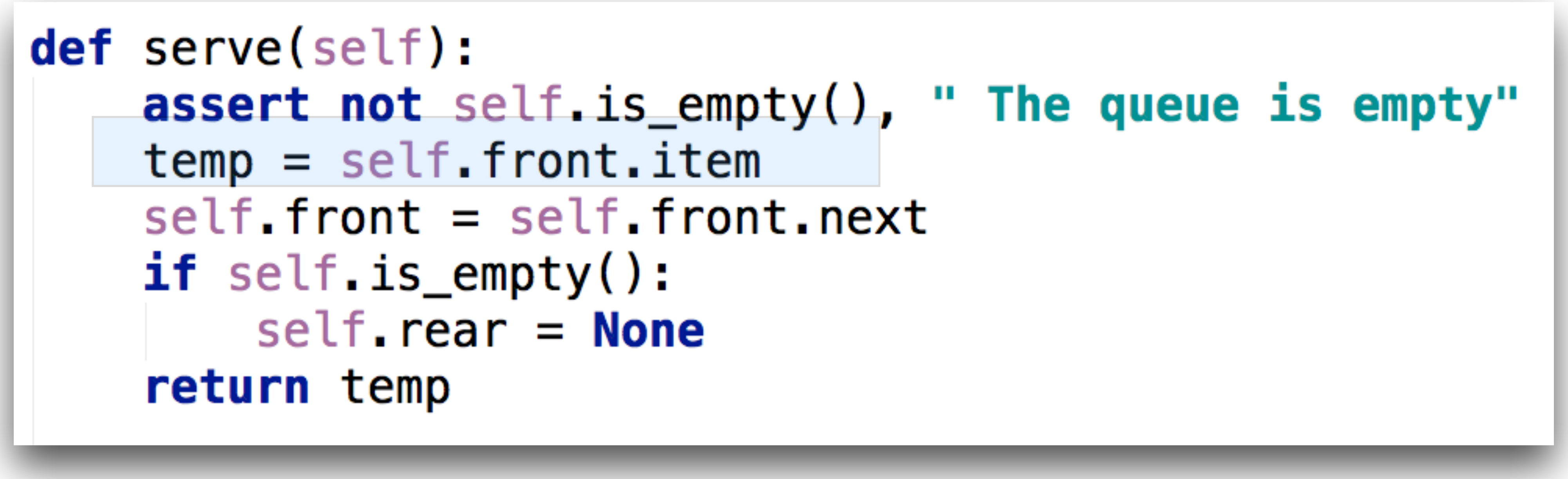
q.serve()



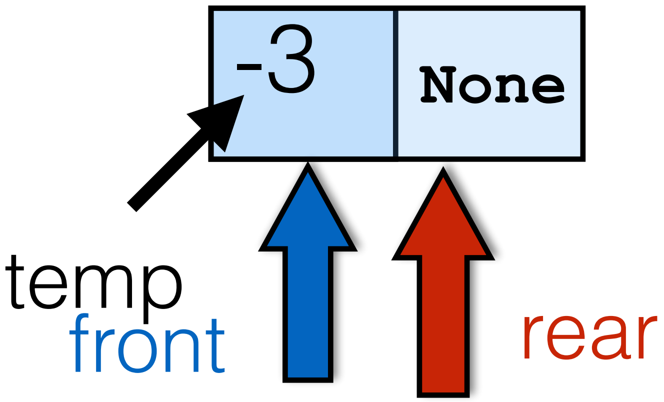
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



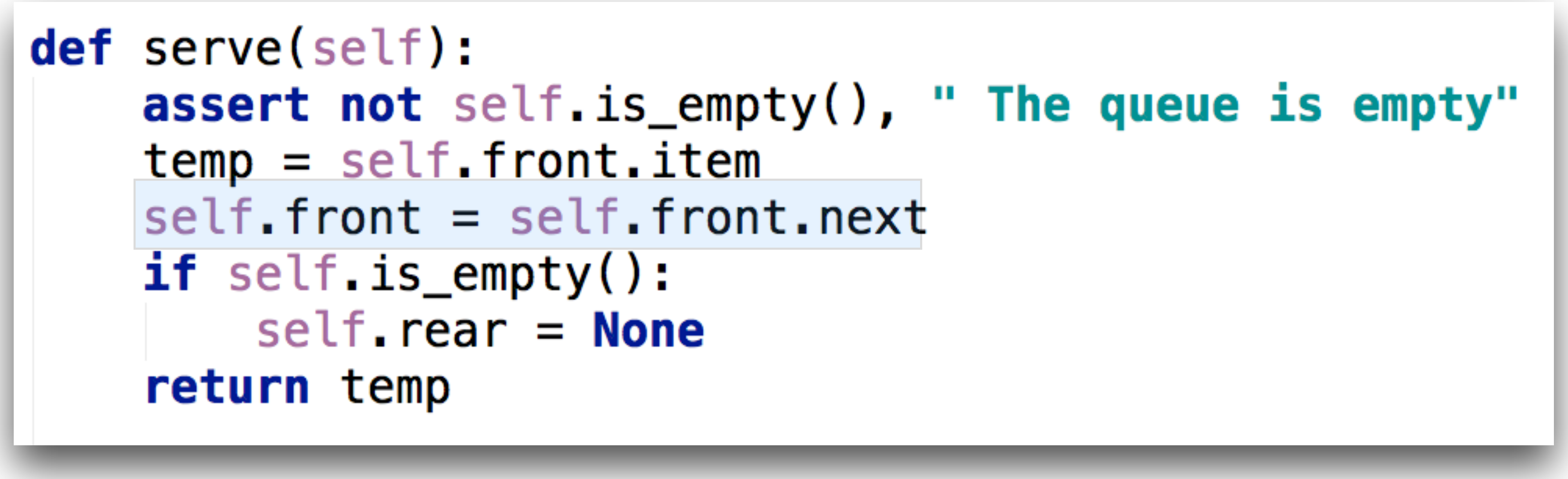
```
q.serve()
```



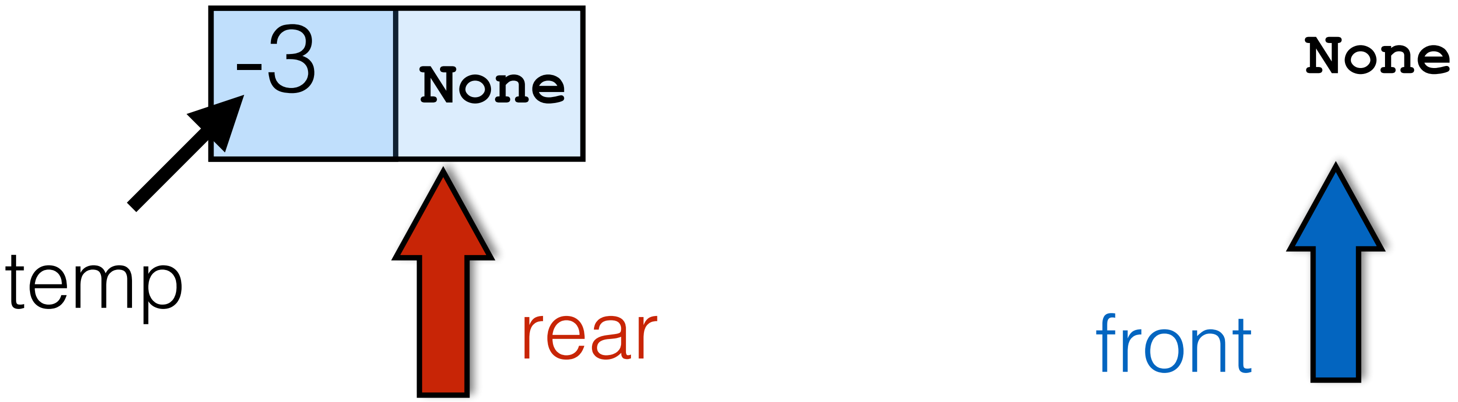
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



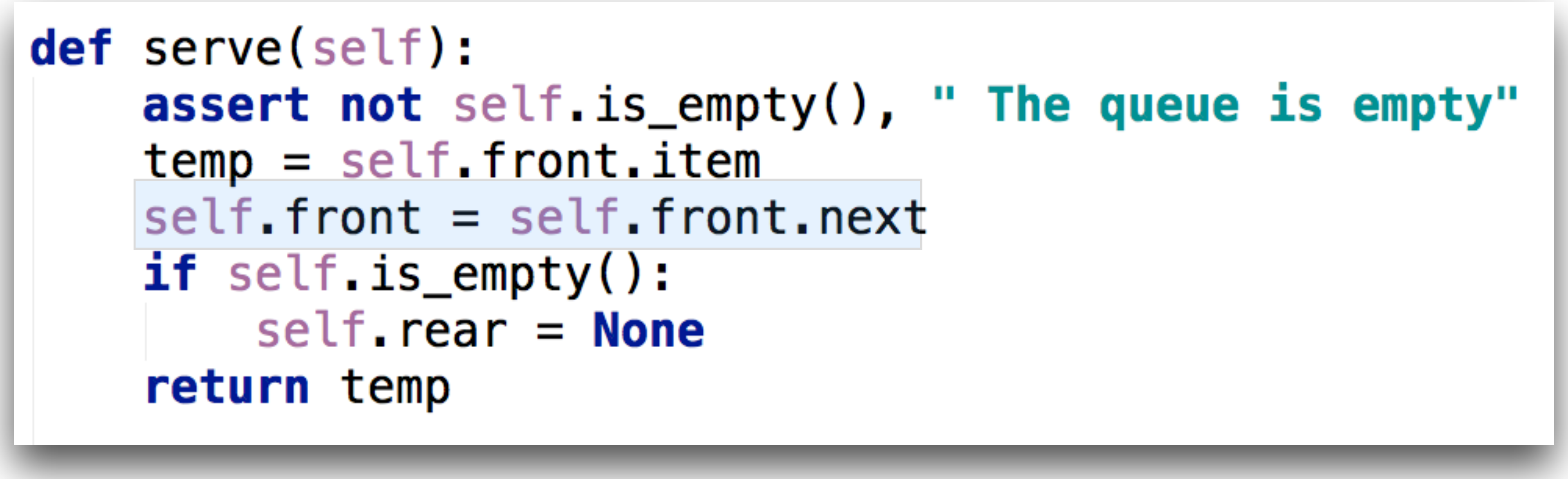
q.serve()



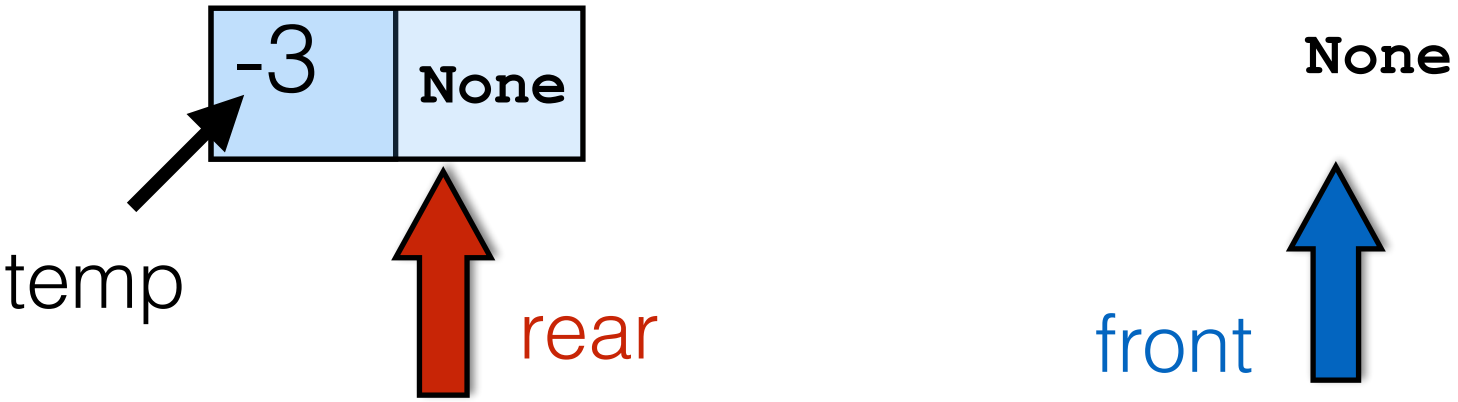
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



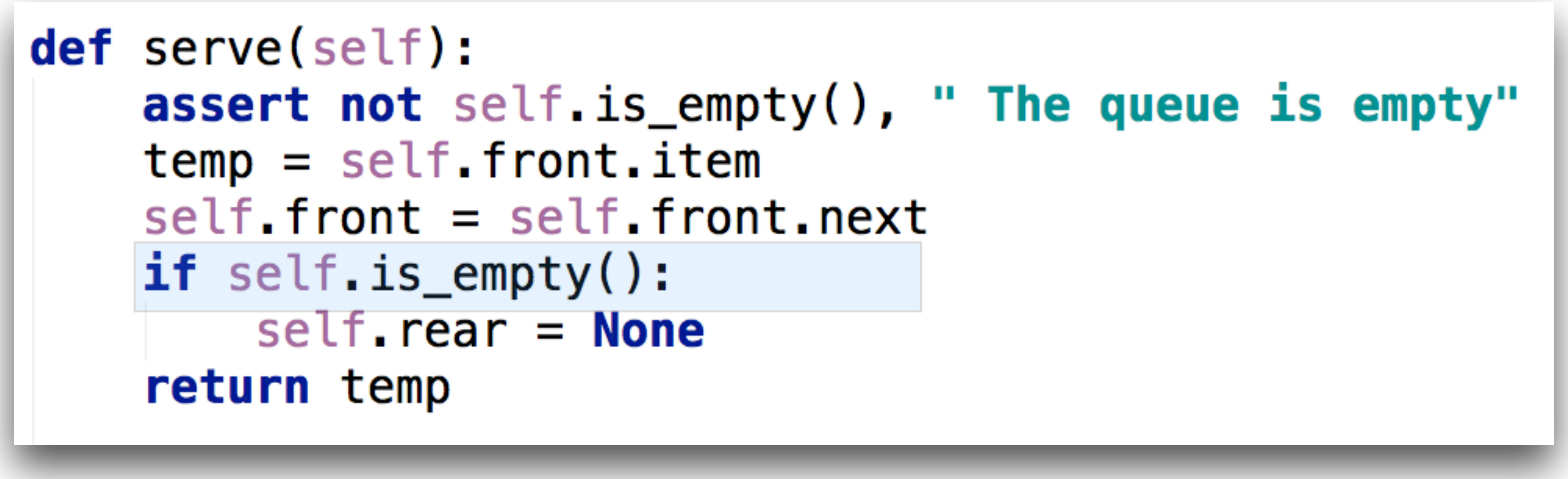
q.serve()



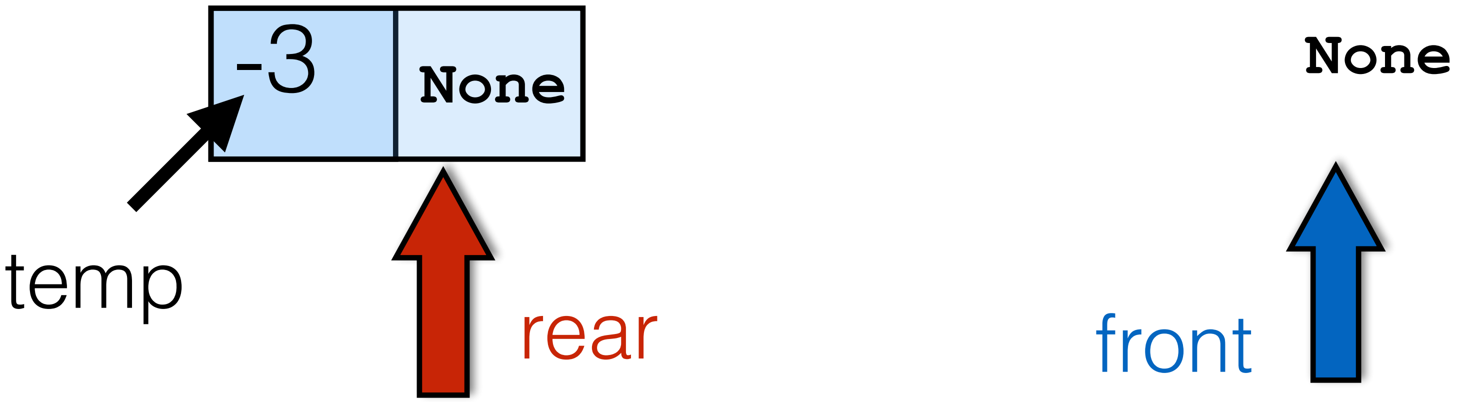
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



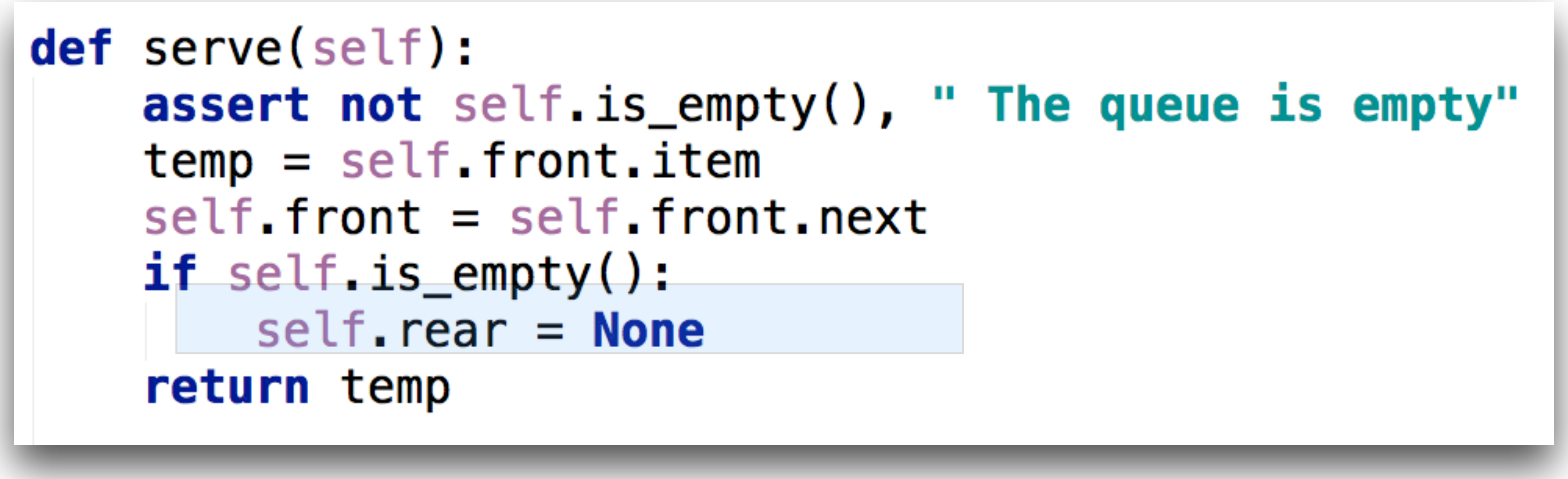
q.serve()



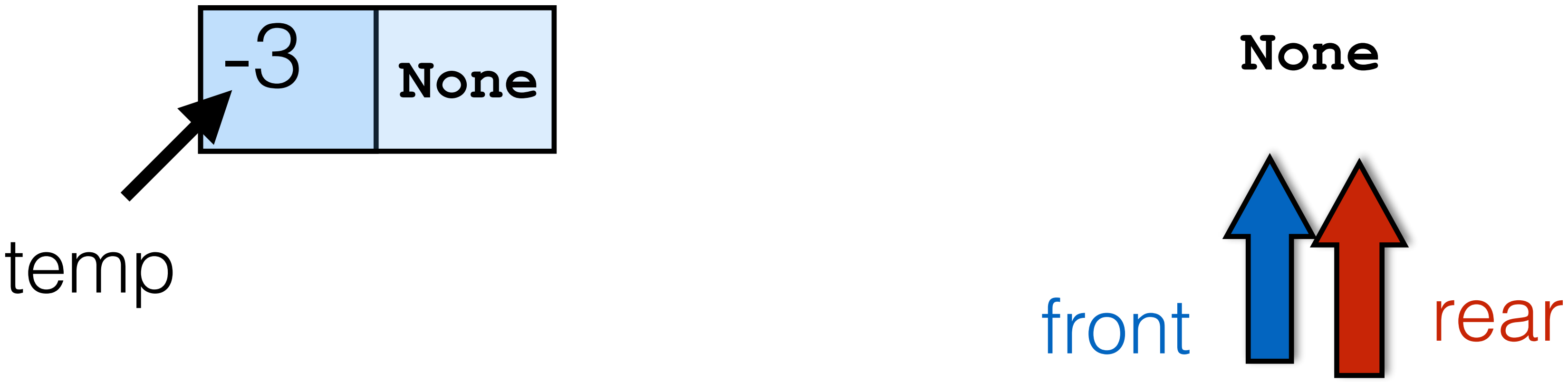
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



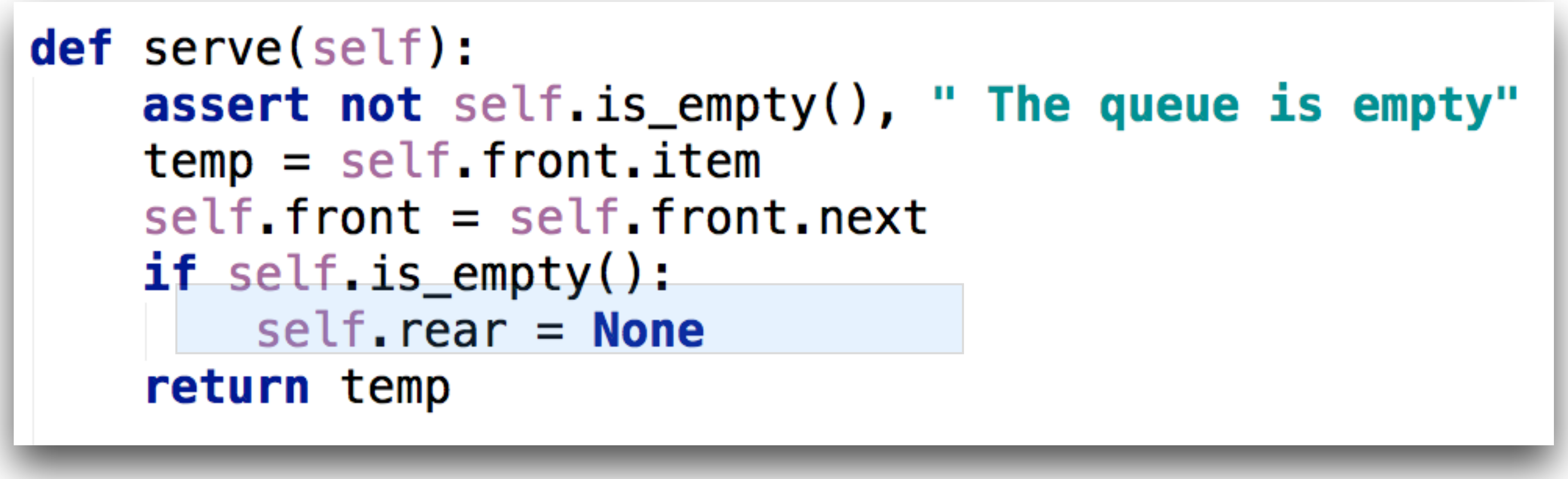
q.serve()



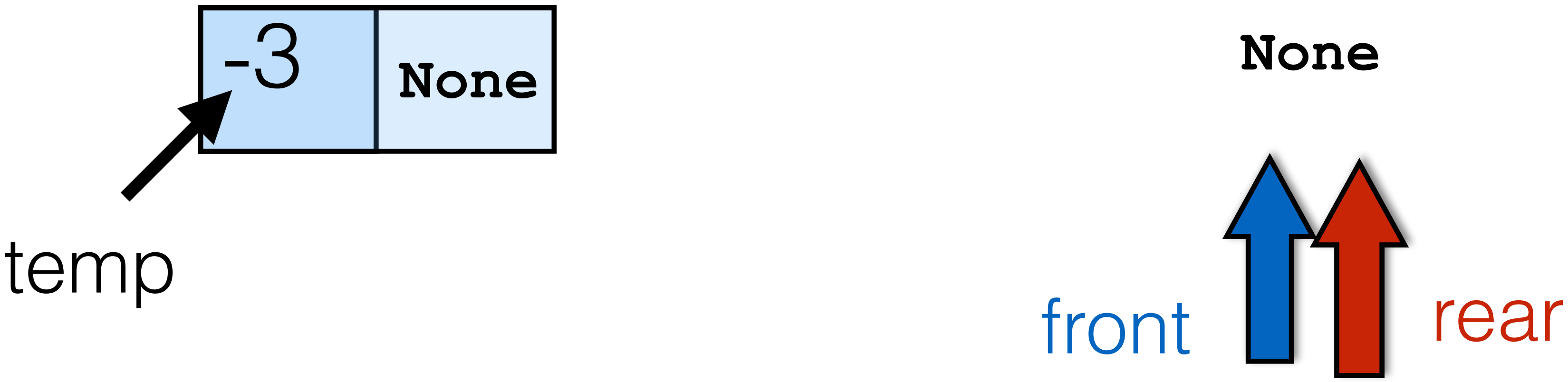
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



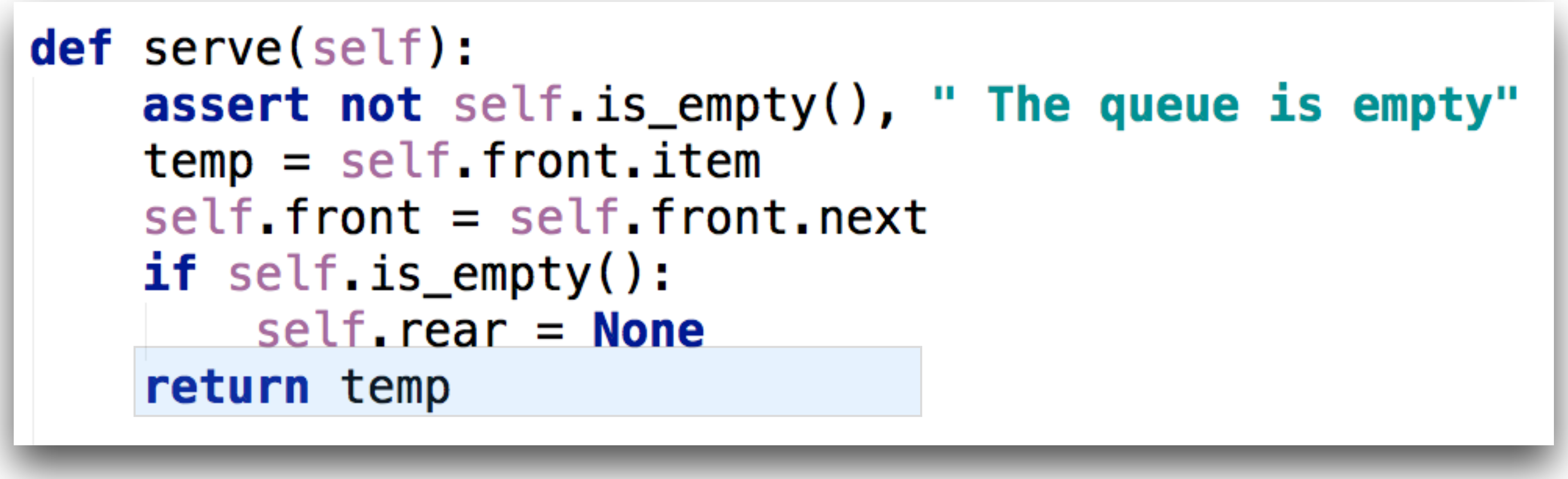
q.serve()



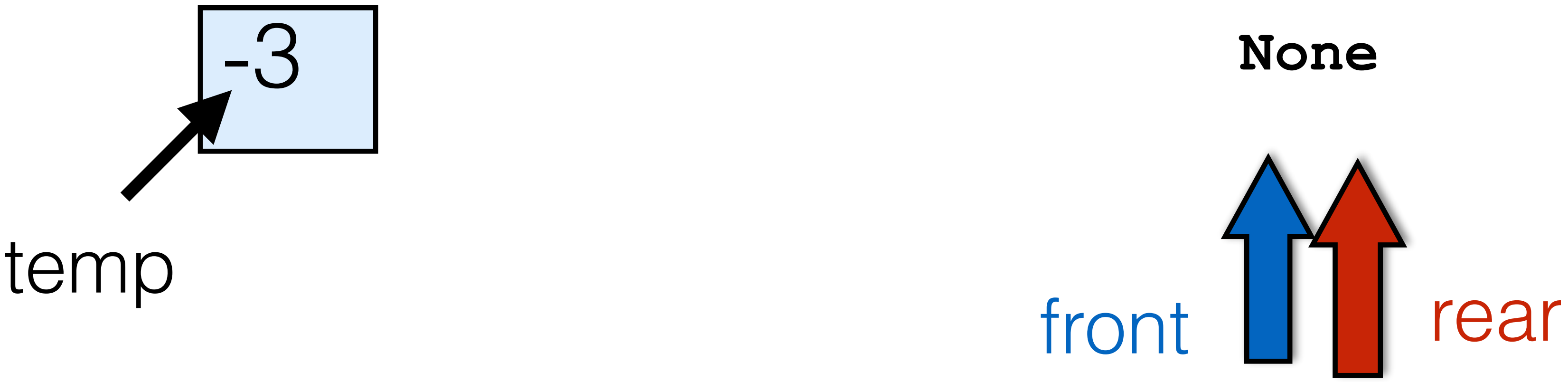
```
def serve(self):
    assert not self.is_empty(), " The queue is empty"
    temp = self.front.item
    self.front = self.front.next
    if self.is_empty():
        self.rear = None
    return temp
```



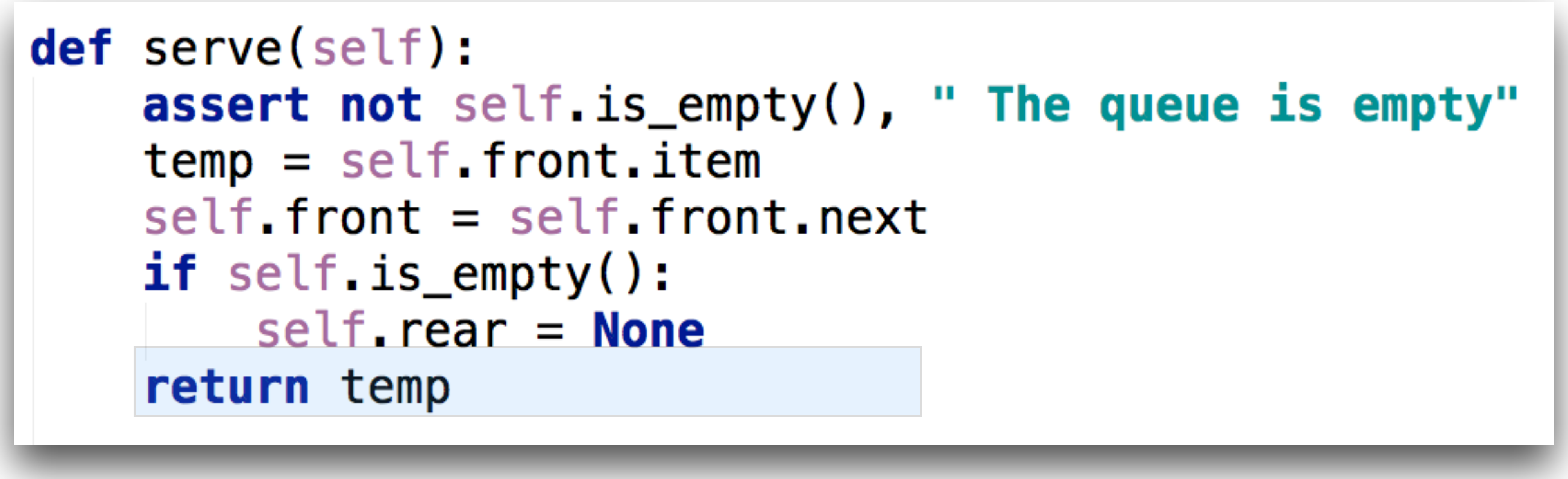
q.serve()



```
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
```

Summary

- Queues implemented with linked data structures