

Lecture 21

Iterators

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.

Objectives for today's lecture

- To understand the importance of **iterators**
- To learn how to implement and use them
- To learn to make our classes iterable by creating iterators on them

We would like to be able to write code like:

```
for item in my_linked_list:  
    print(item)
```

For loops

```
for c in "abc":  
    print(c)
```

```
for item in ["apple", "pear", "plum"]:  
    print(item)
```

```
infile = open("example.txt")  
for line in infile:  
    print(line, end=' ')
```

Can we iterate over our own objects?

```
for c in "abc":  
    print(c)
```

a

b

c

```
>>> s = 'abc'
```

```
>>> itr = iter(s)
```

```
>>> itr
```

```
<str_iterator object at 0x10382f250>
```

```
>>> c = next(itr)
```

```
>>> c
```

```
'a'
```

```
>>> c = next(itr)
```

```
>>> c
```

```
'b'
```

```
>>> c = next(itr)
```

```
>>> c
```

```
'c'
```

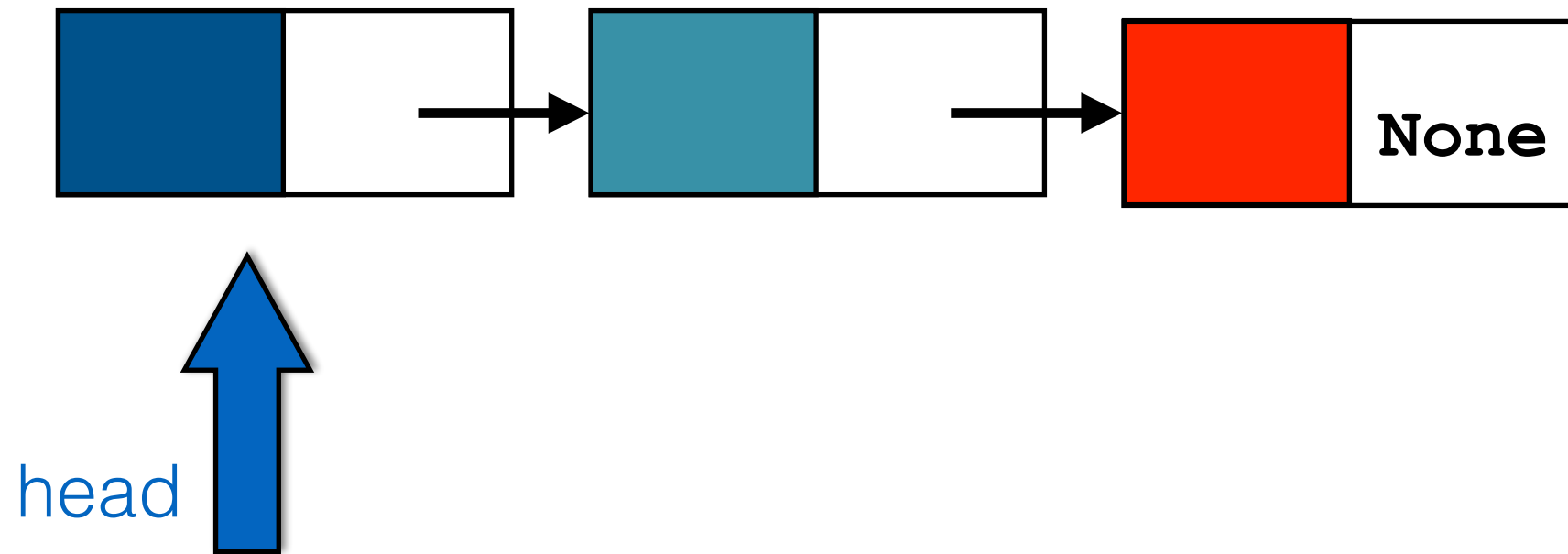
```
>>> c = next(itr)
```

```
Traceback (most recent call last):
```

```
File "<pyshell#12>", line 1, in <module>
```

```
c = next(itr)
```

```
StopIteration
```



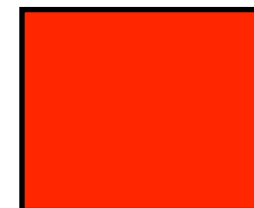
Next!



Next!



Next!



Next!

No more

Handled differently in
different languages.

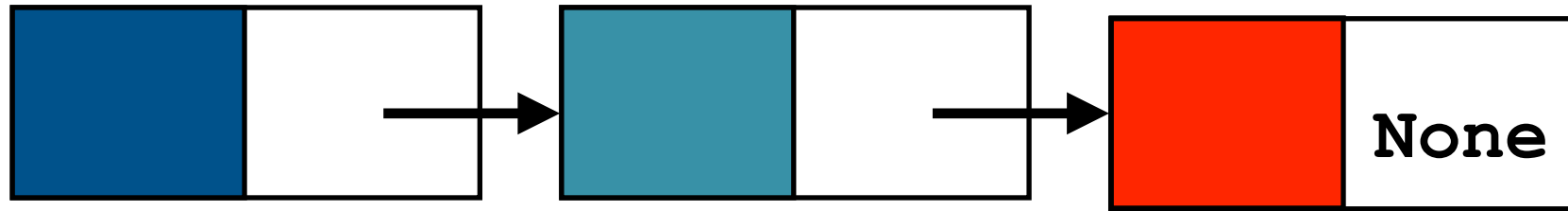
Python: raise
StopIteration

List iterator

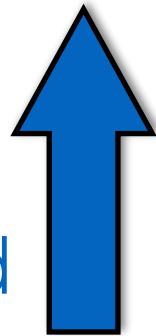
- We need to create an **iterator class** for the list.
A class with the methods:

```
__init__  
__iter__  
__next__
```

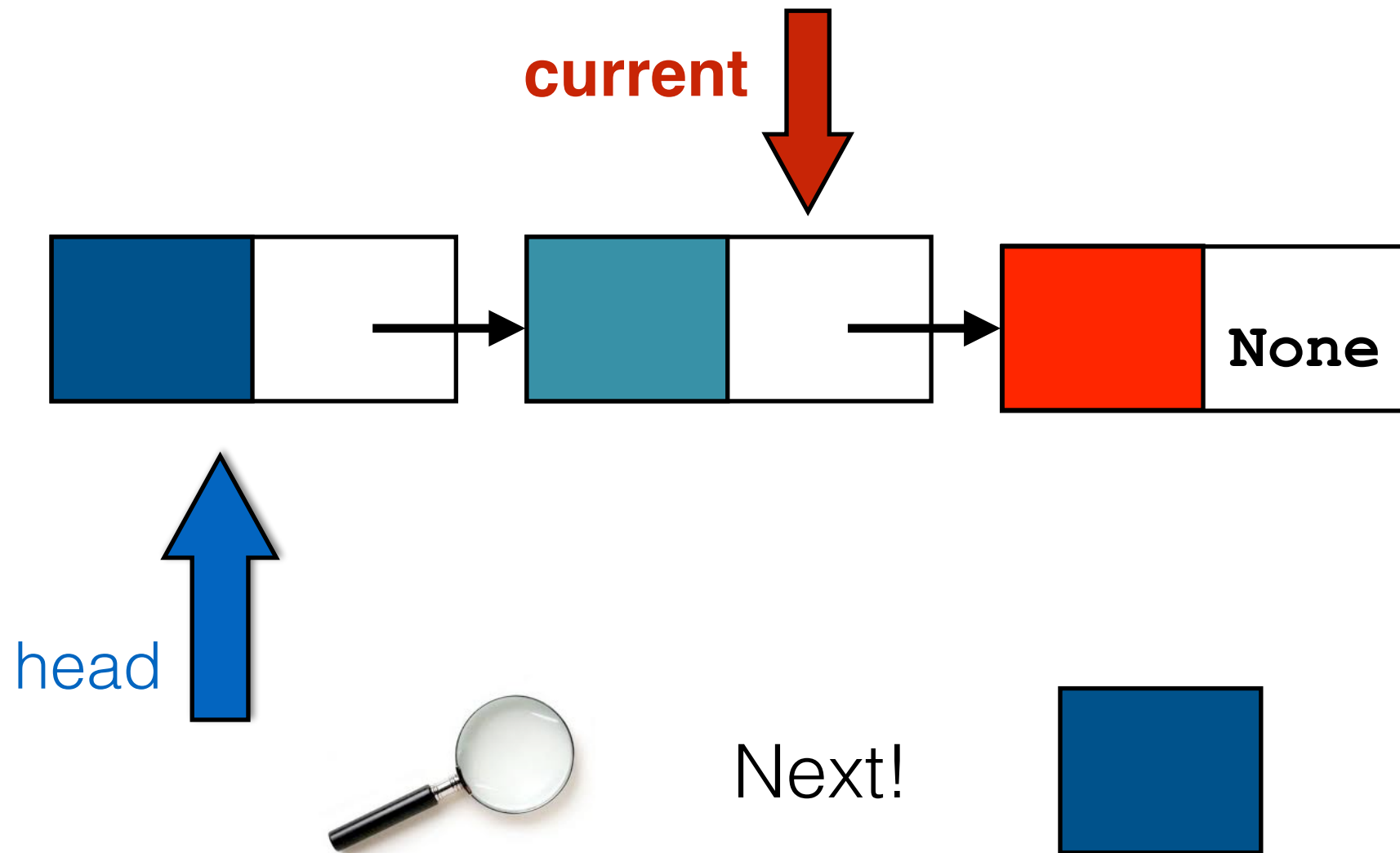
current

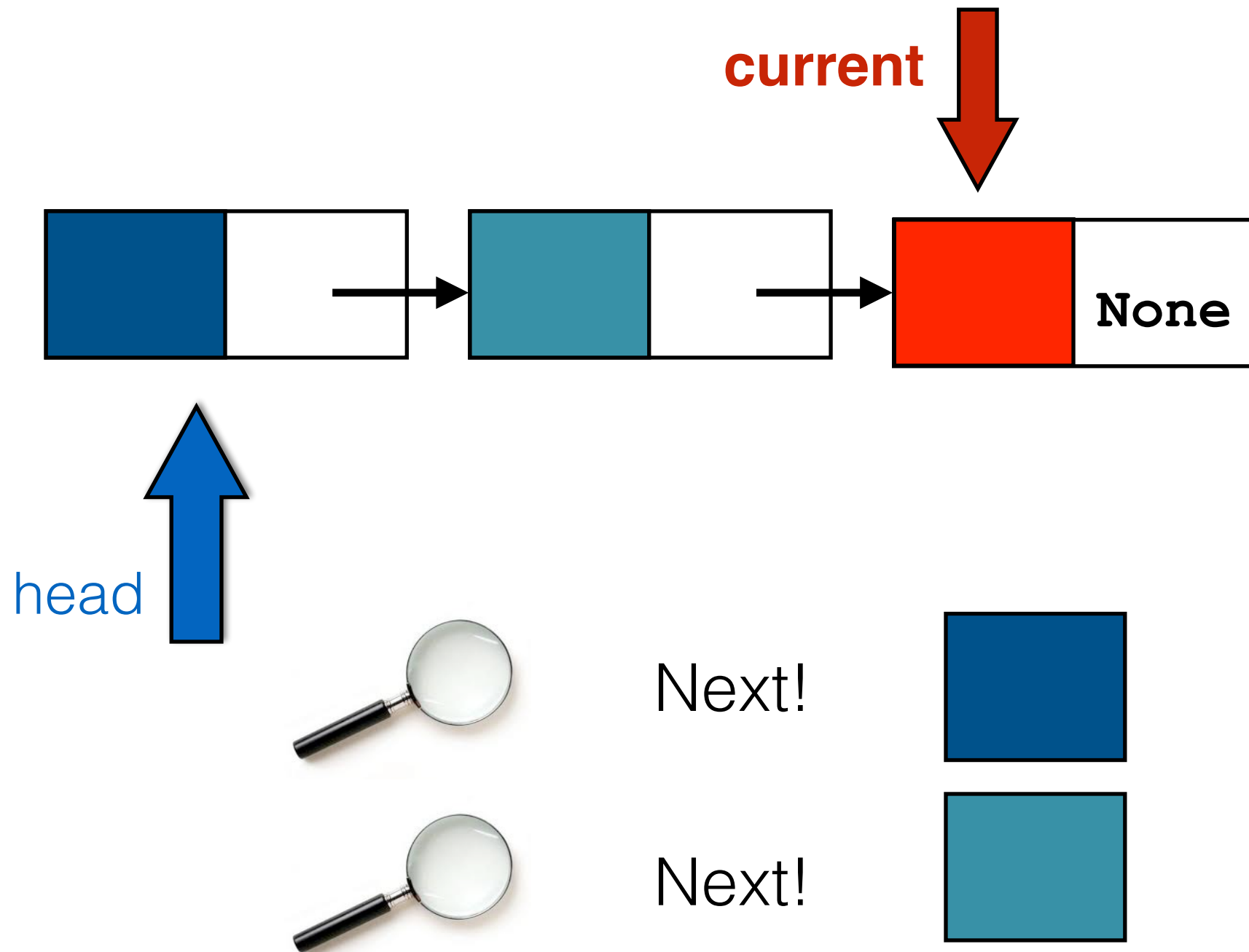


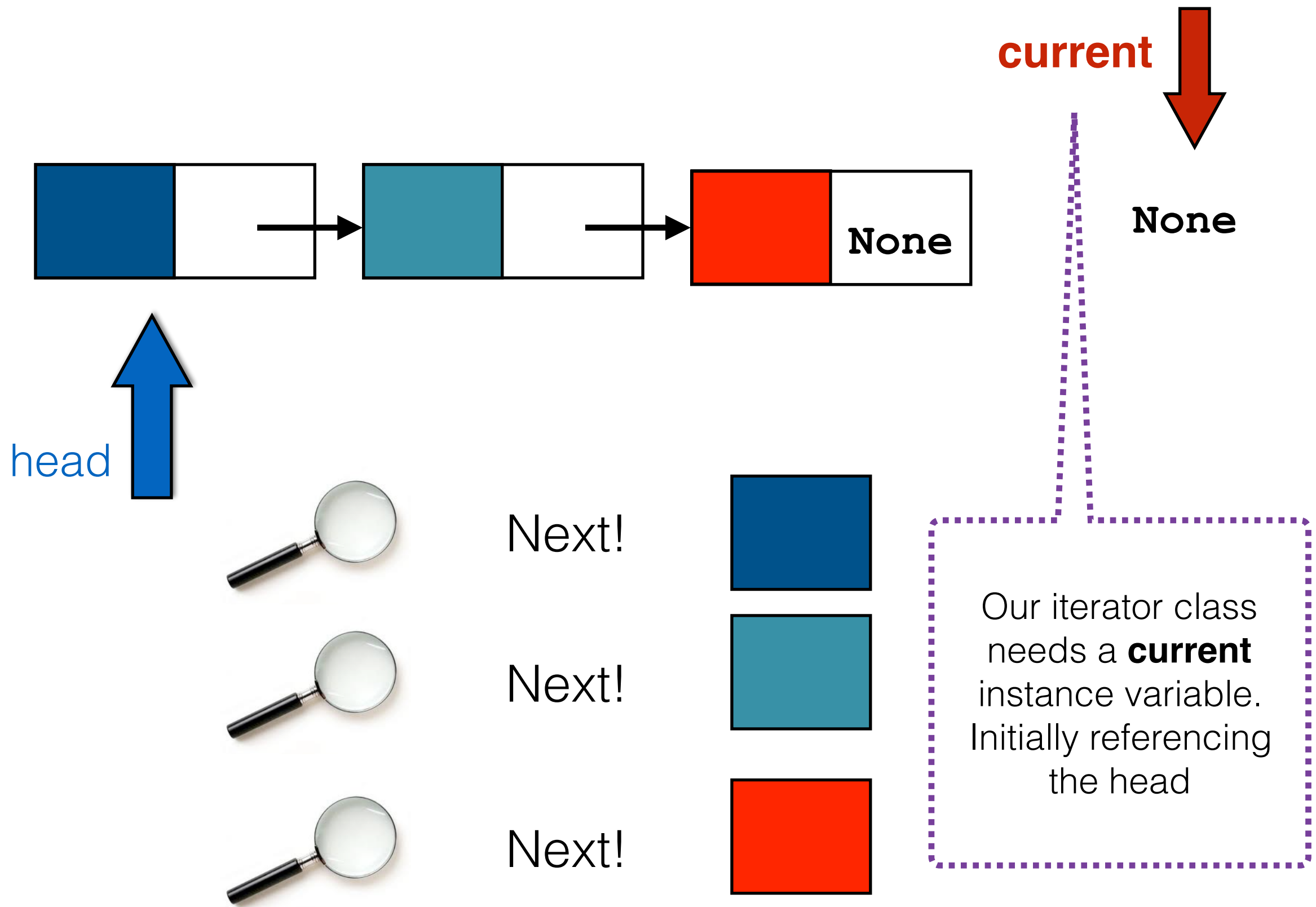
head



Next!







```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

How is my Iterator connected to the List class?

- The List class needs to have an **`__iter__`** method too
- Which returns a list iterator object initialised to the head of the list

```
class List:
    def __init__(self):
        self.head = None
        self.count = 0

    def __iter__(self):
        return ListIterator(self.head)
```

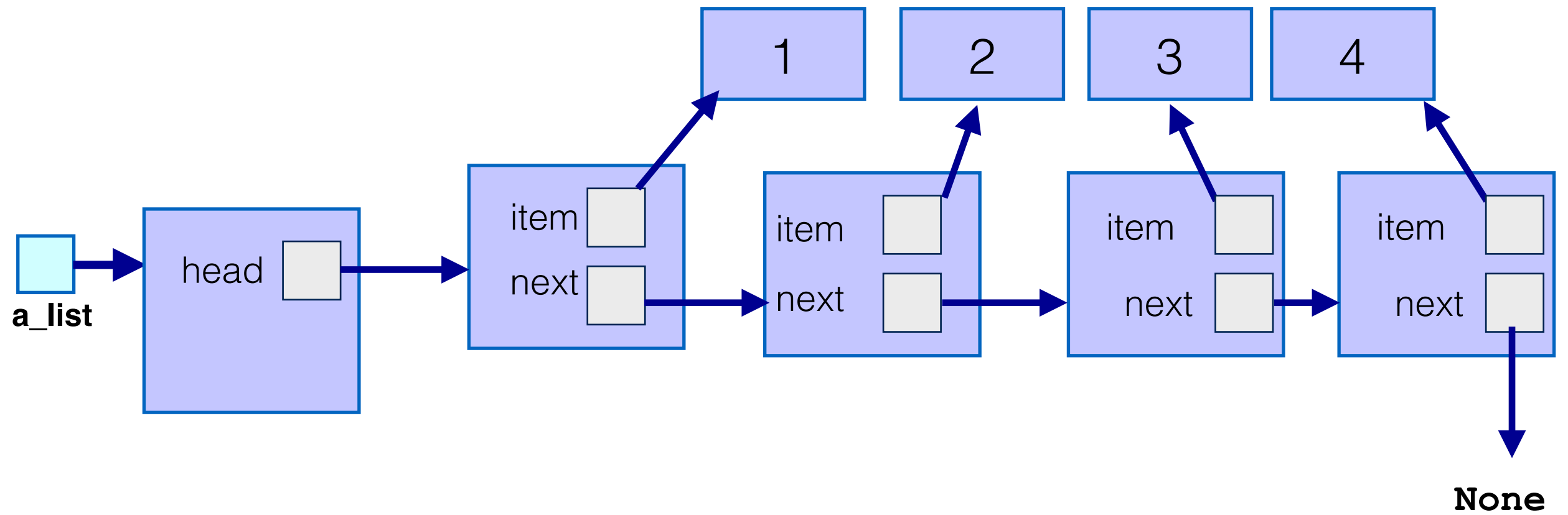
```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

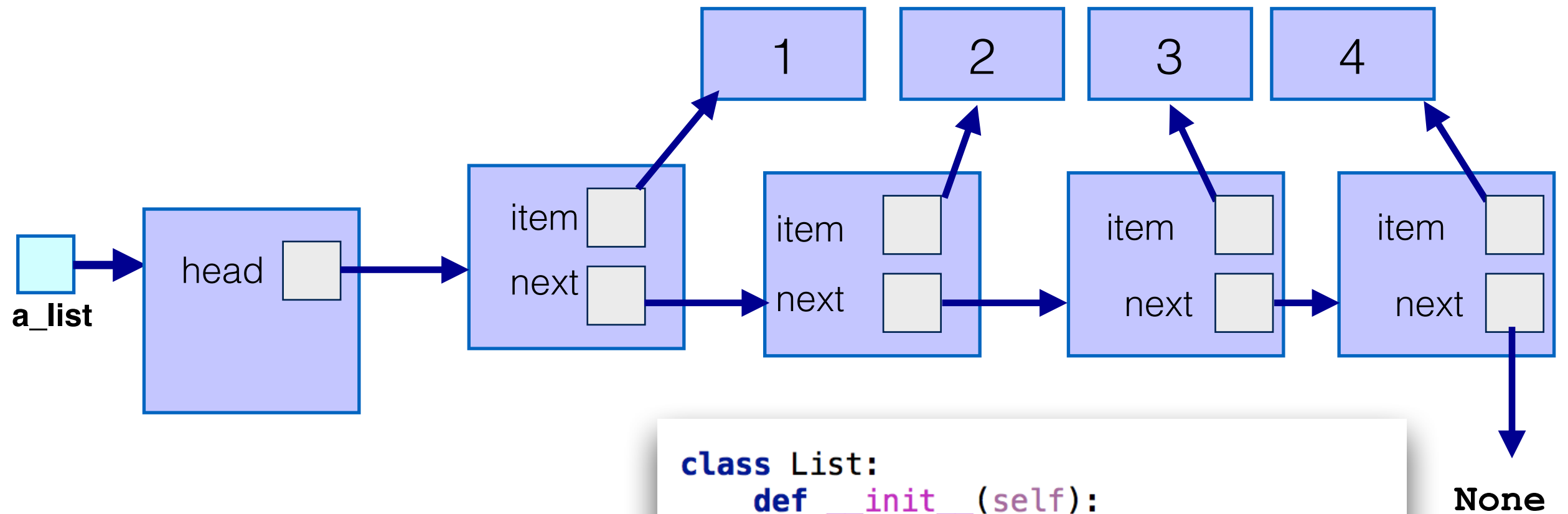
    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

```
>>> a_list = List()
```

```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
```



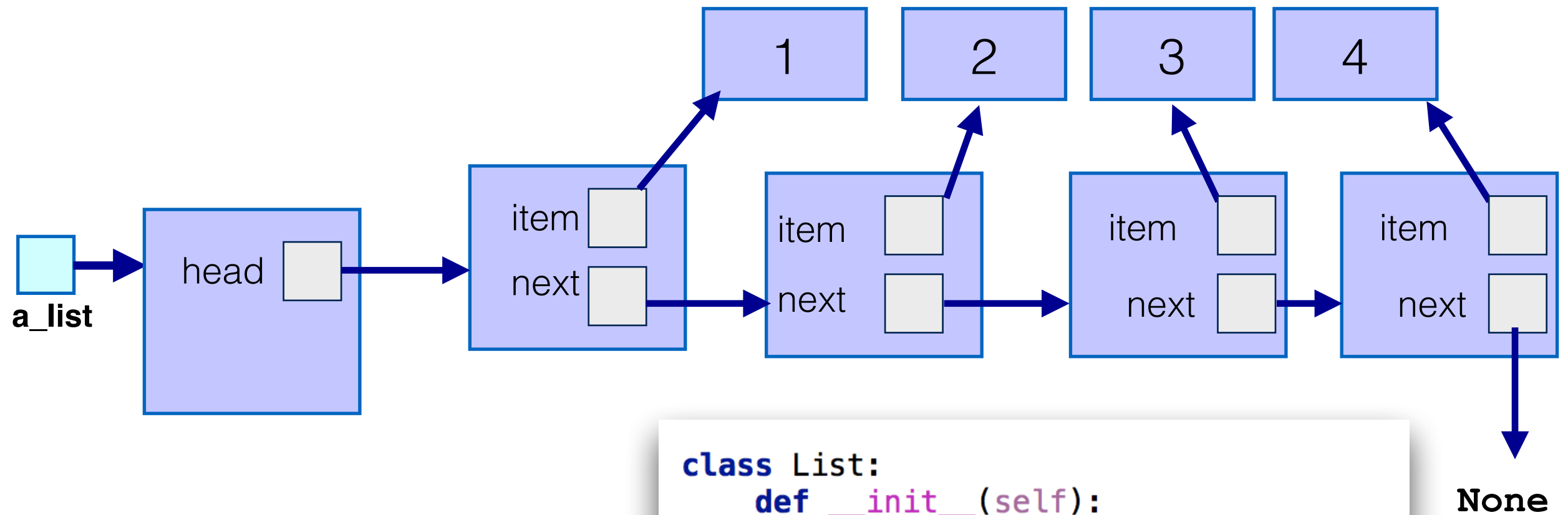
```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
```

```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
```

```
class List:
    def __init__(self):
        self.head = None
        self.count = 0

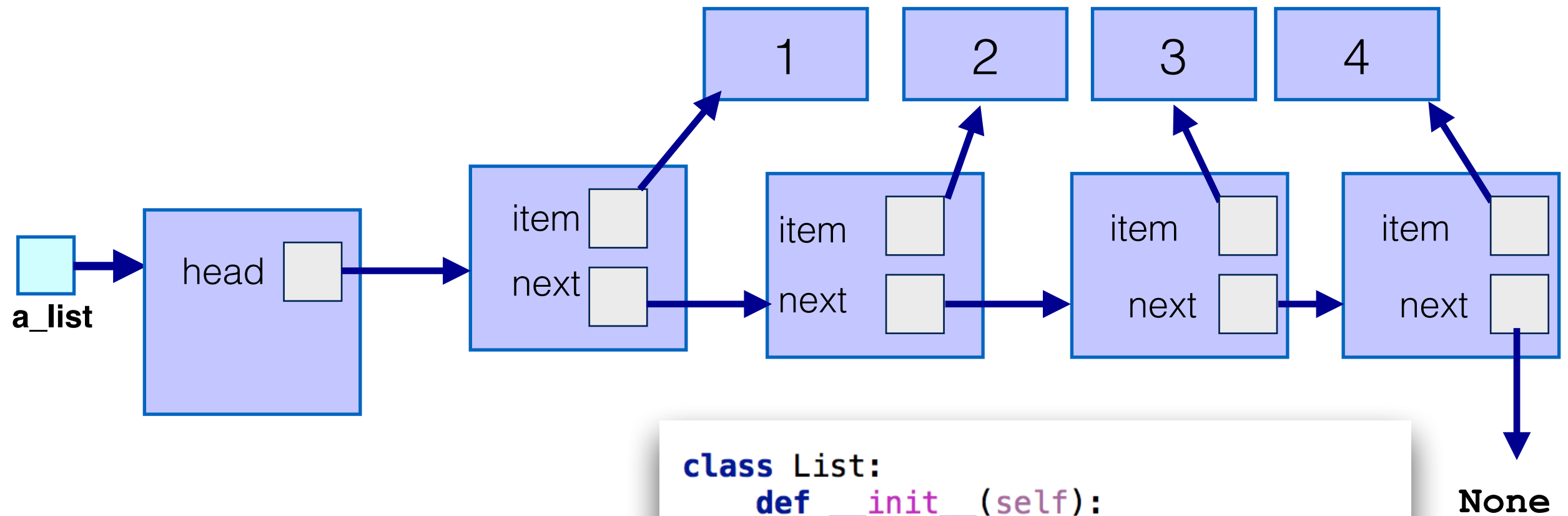
    def __iter__(self):
        return ListIterator(self.head)
```



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
```

```
class List:
    def __init__(self):
        self.head = None
        self.count = 0

    def __iter__(self):
        return ListIterator(self.head)
```



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
```

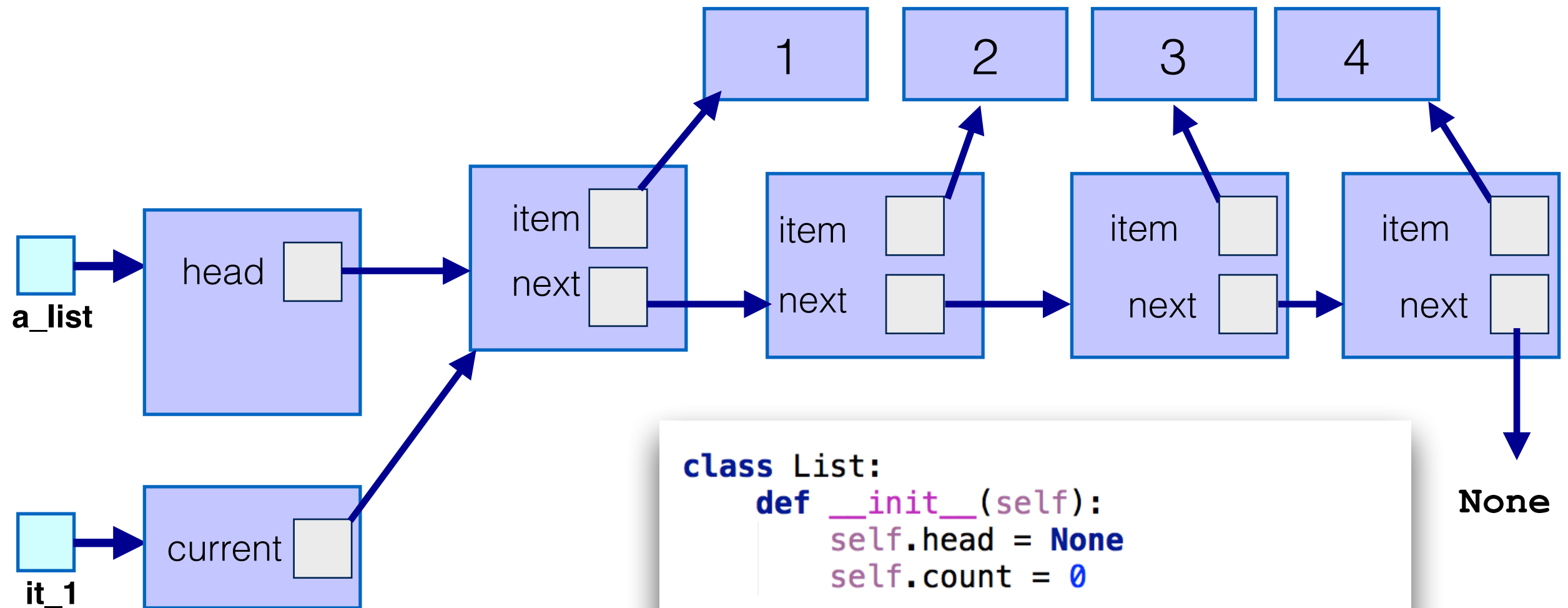
```
class List:
    def __init__(self):
        self.head = None
        self.count = 0

    def __iter__(self):
        return ListIterator(self.head)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
```

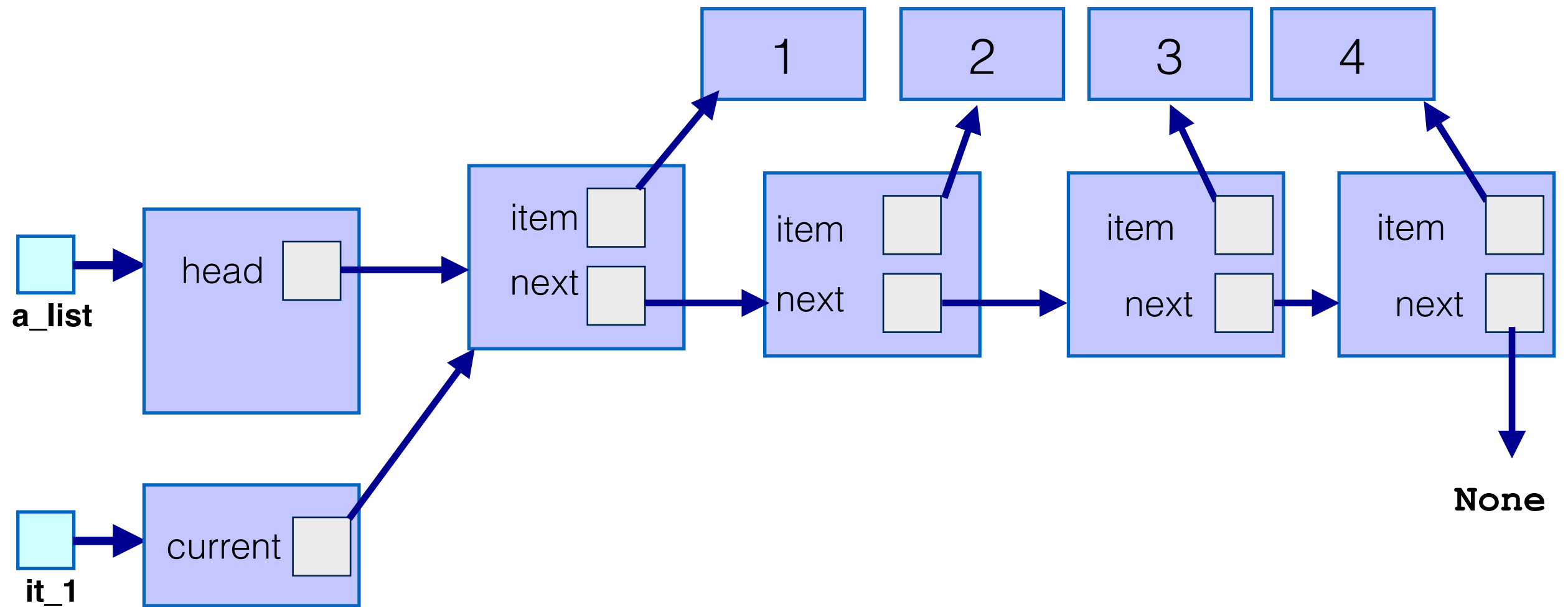
```
class List:
    def __init__(self):
        self.head = None
        self.count = 0

    def __iter__(self):
        return ListIterator(self.head)
```

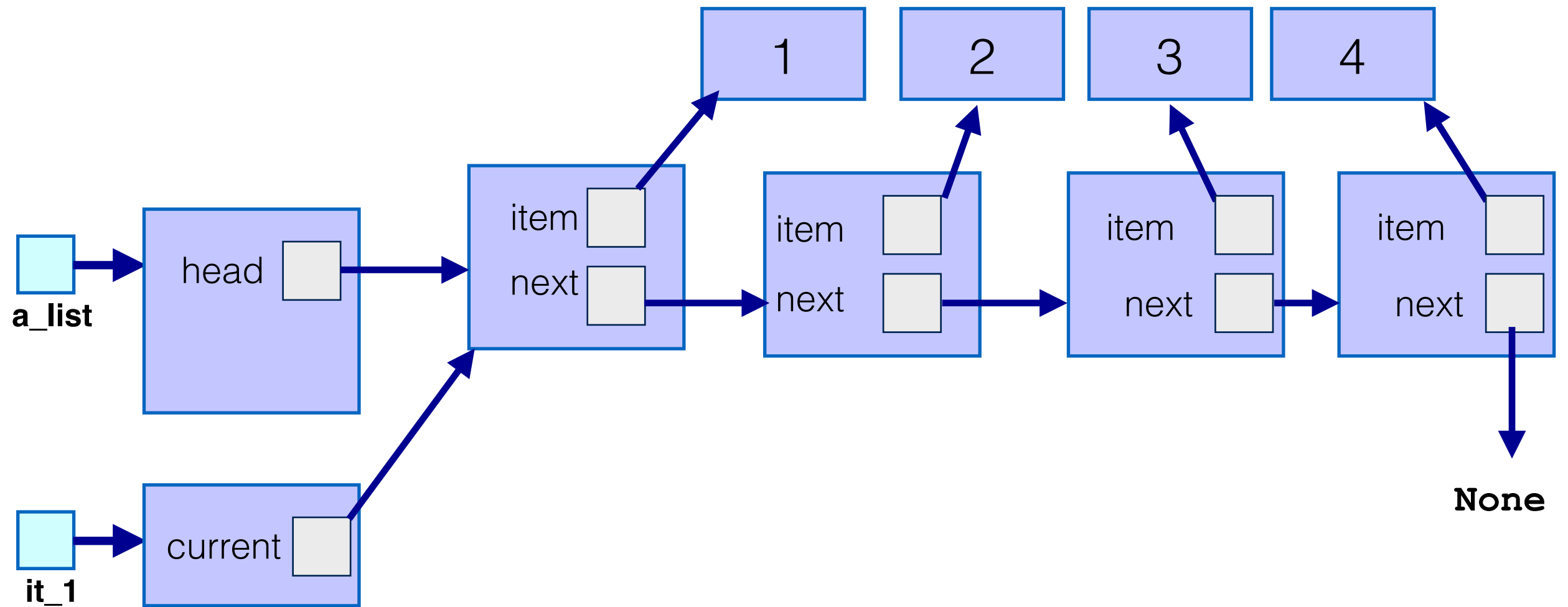
```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

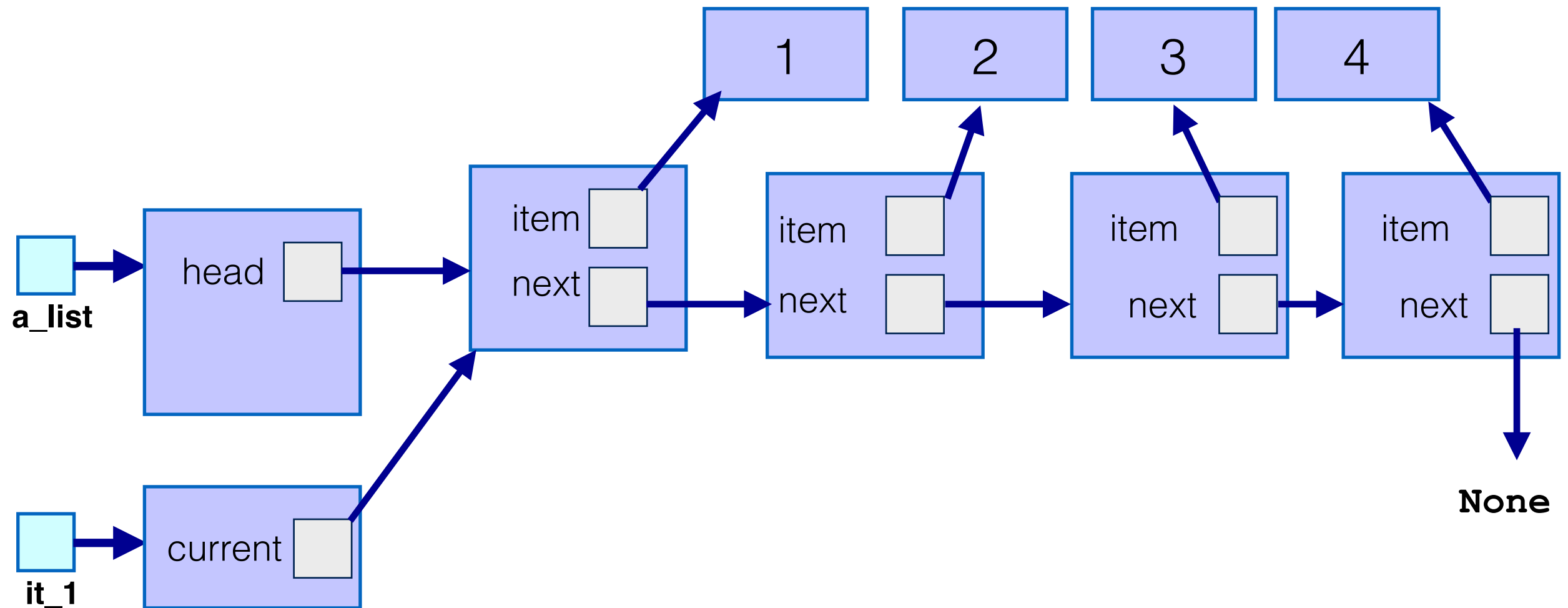
    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
```



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
```

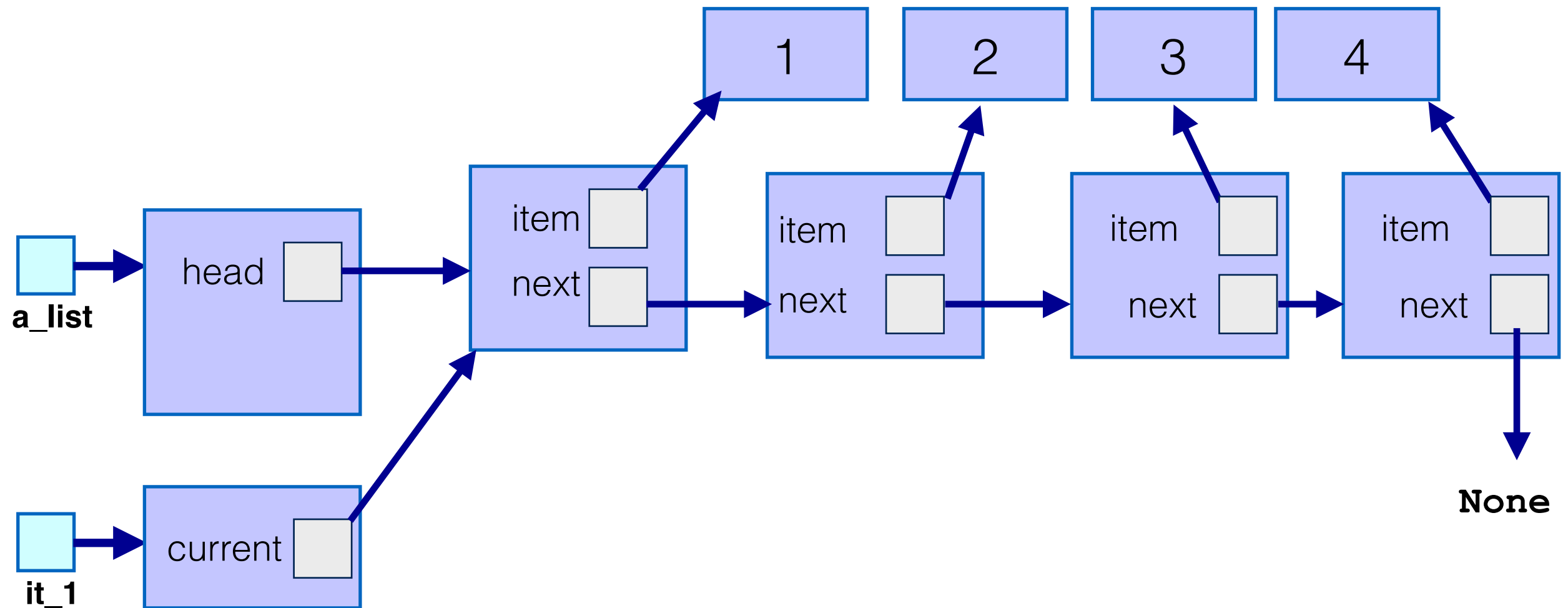


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

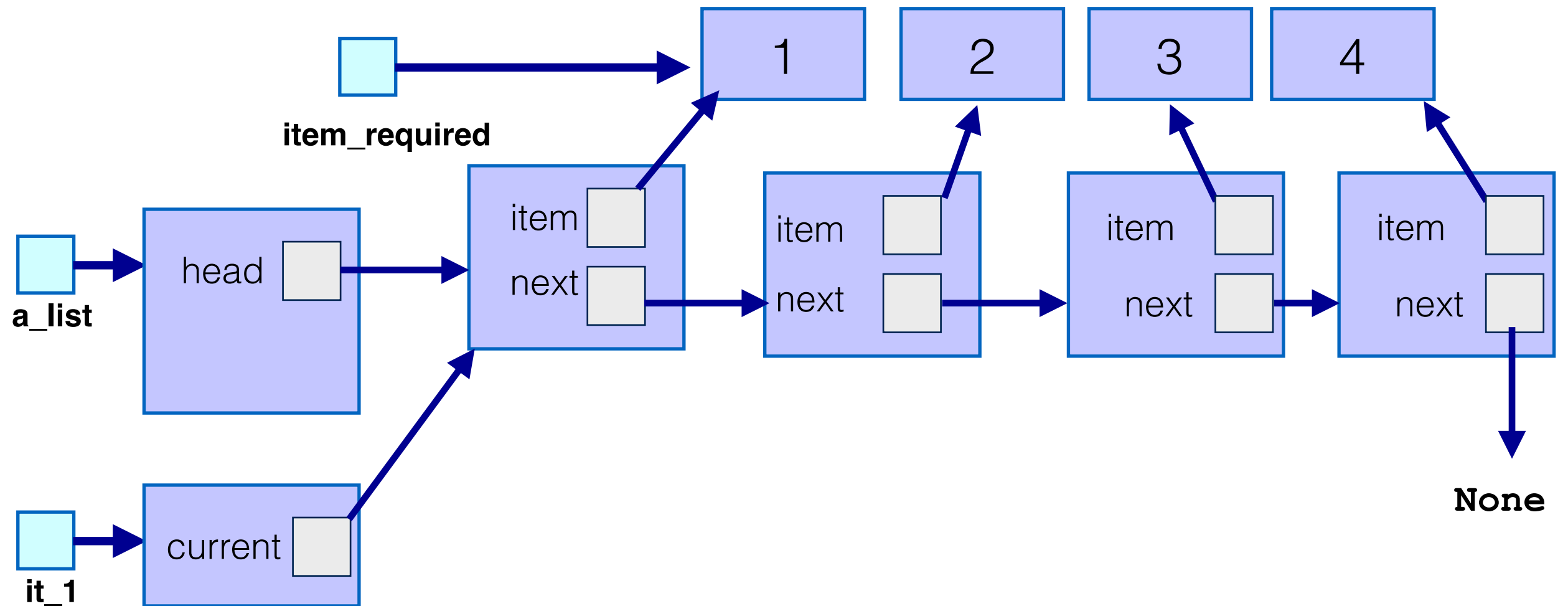


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

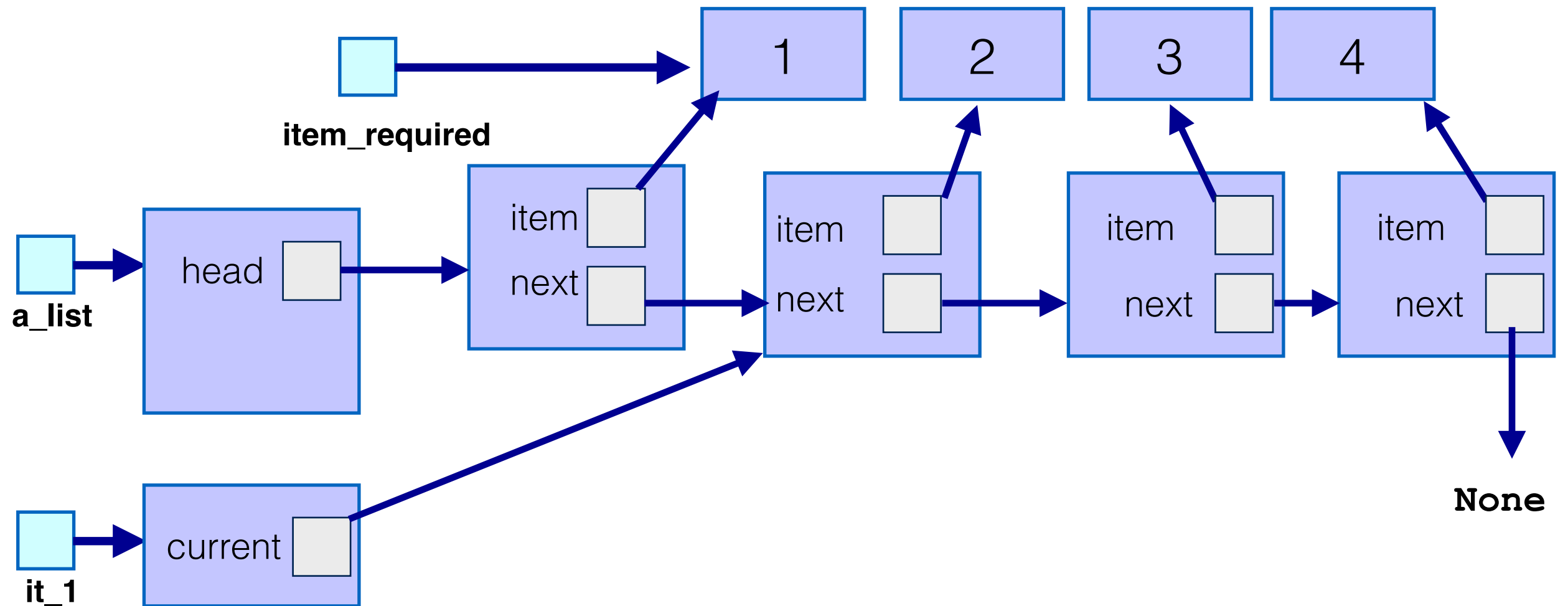



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

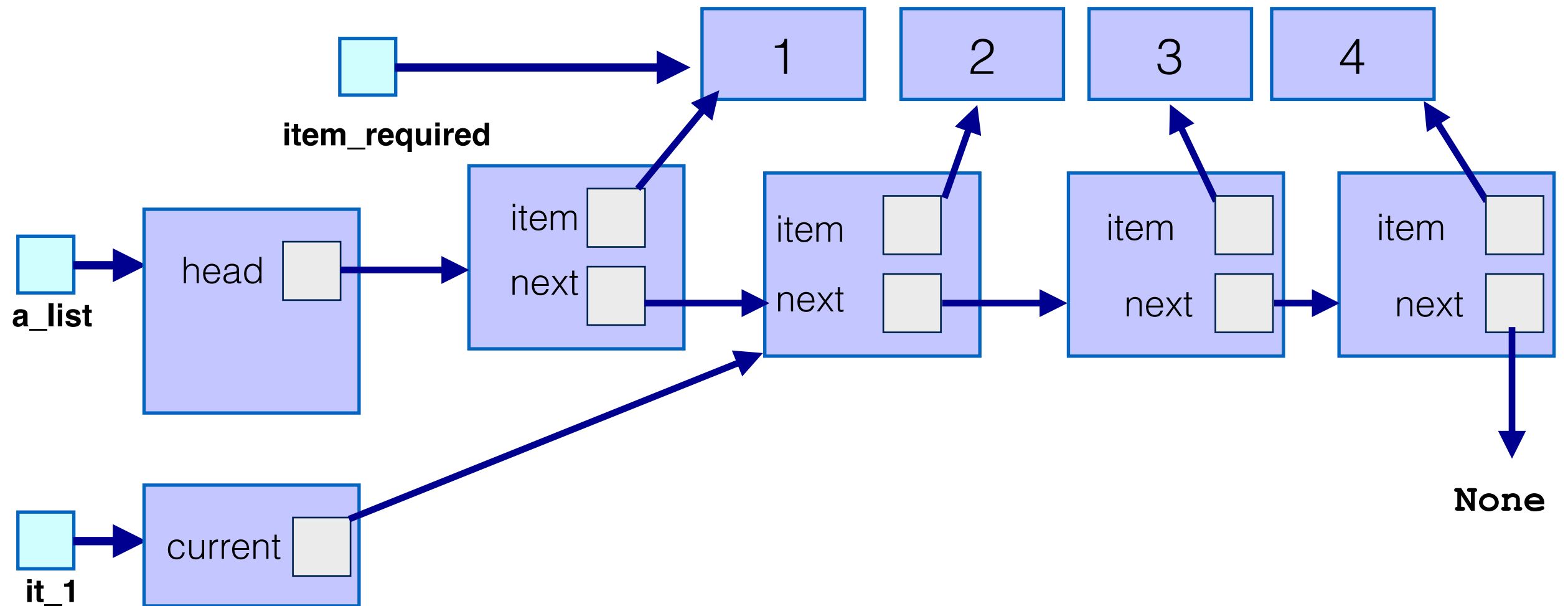


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

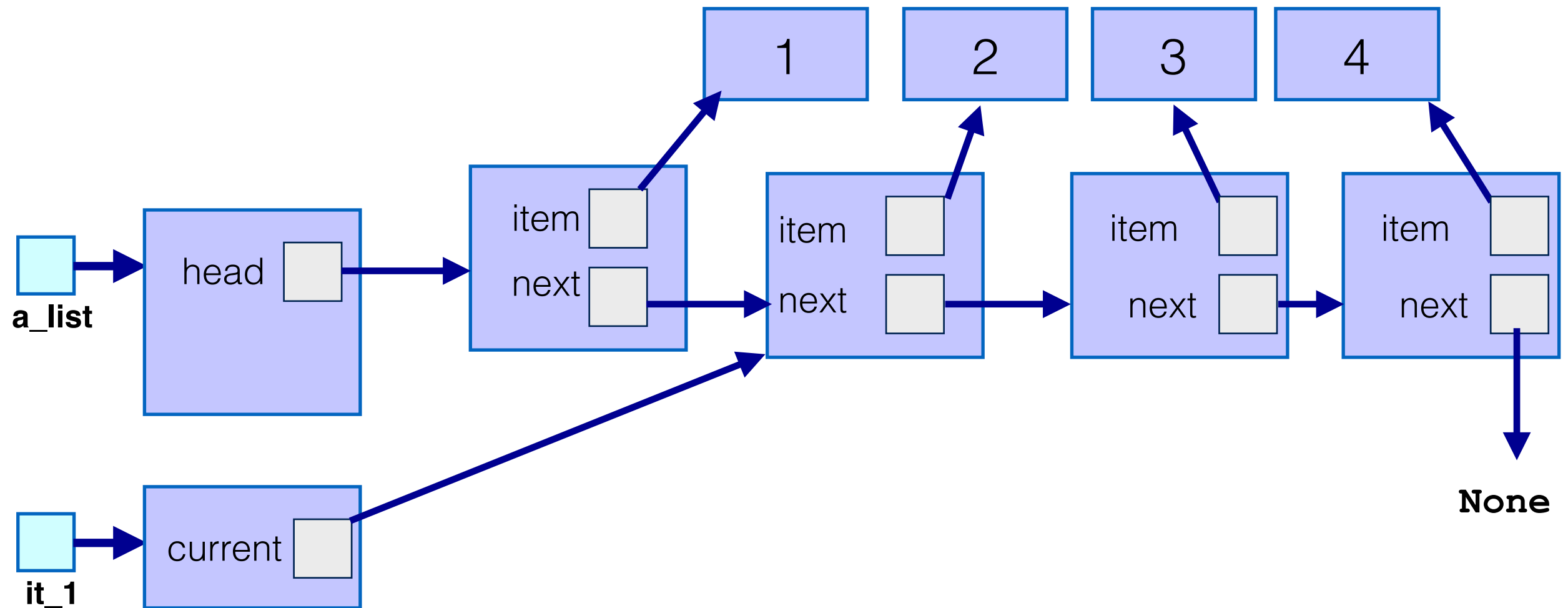


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

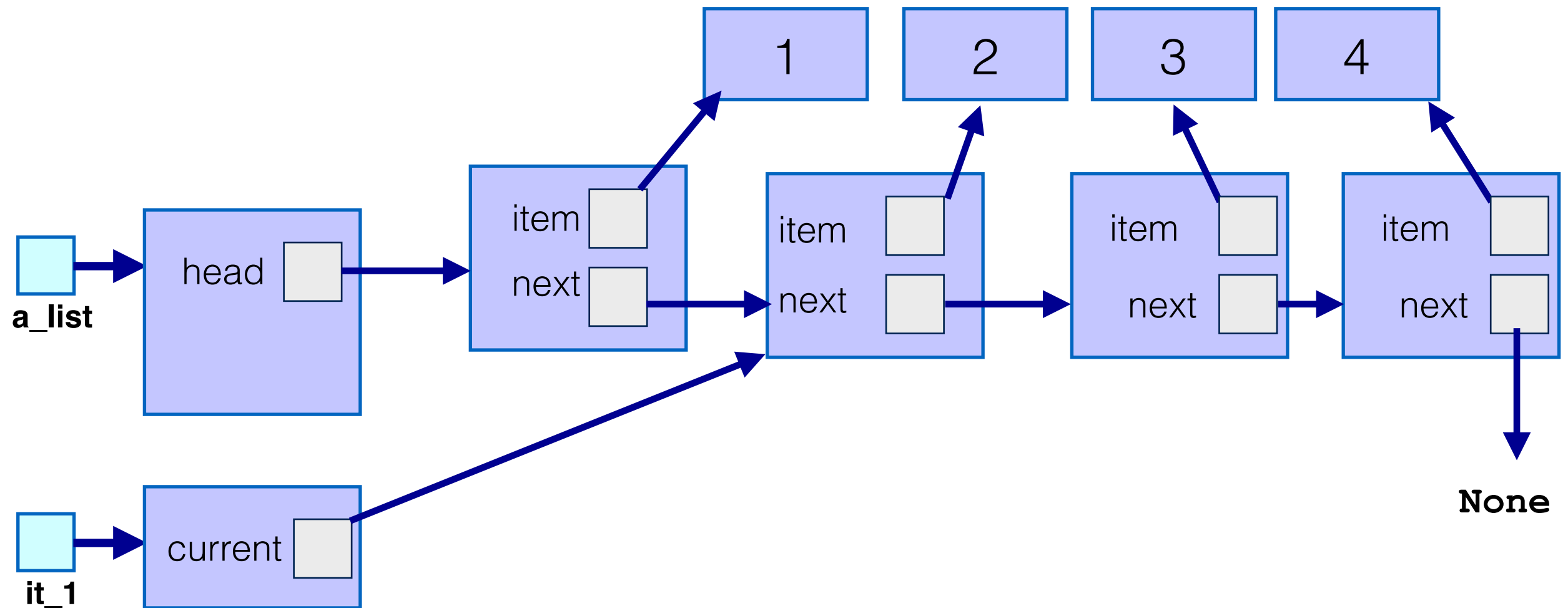


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

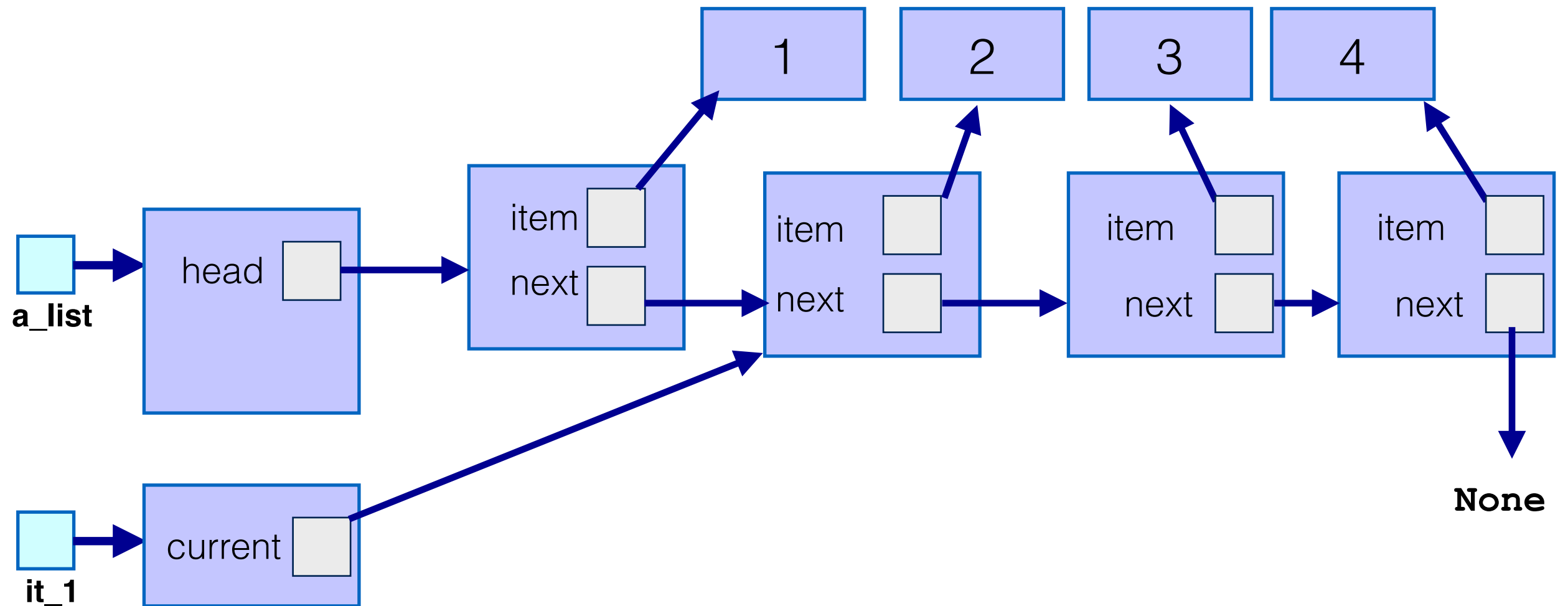


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

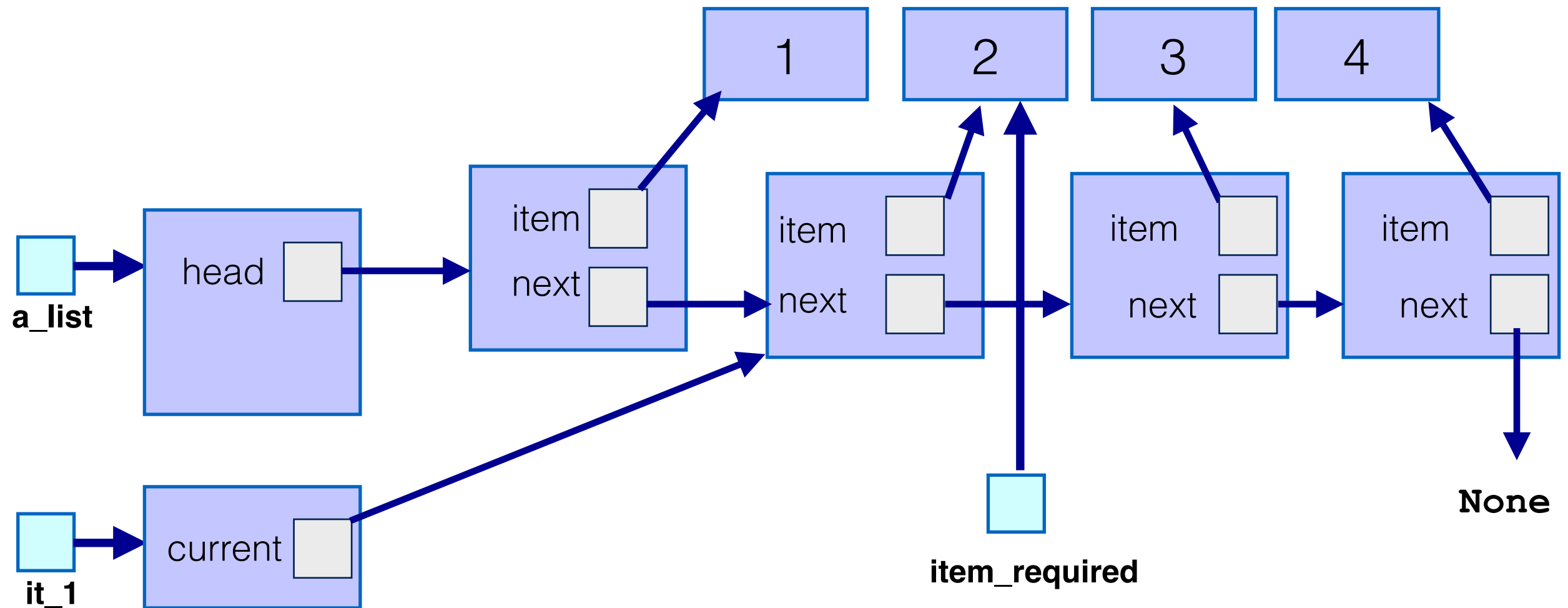


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

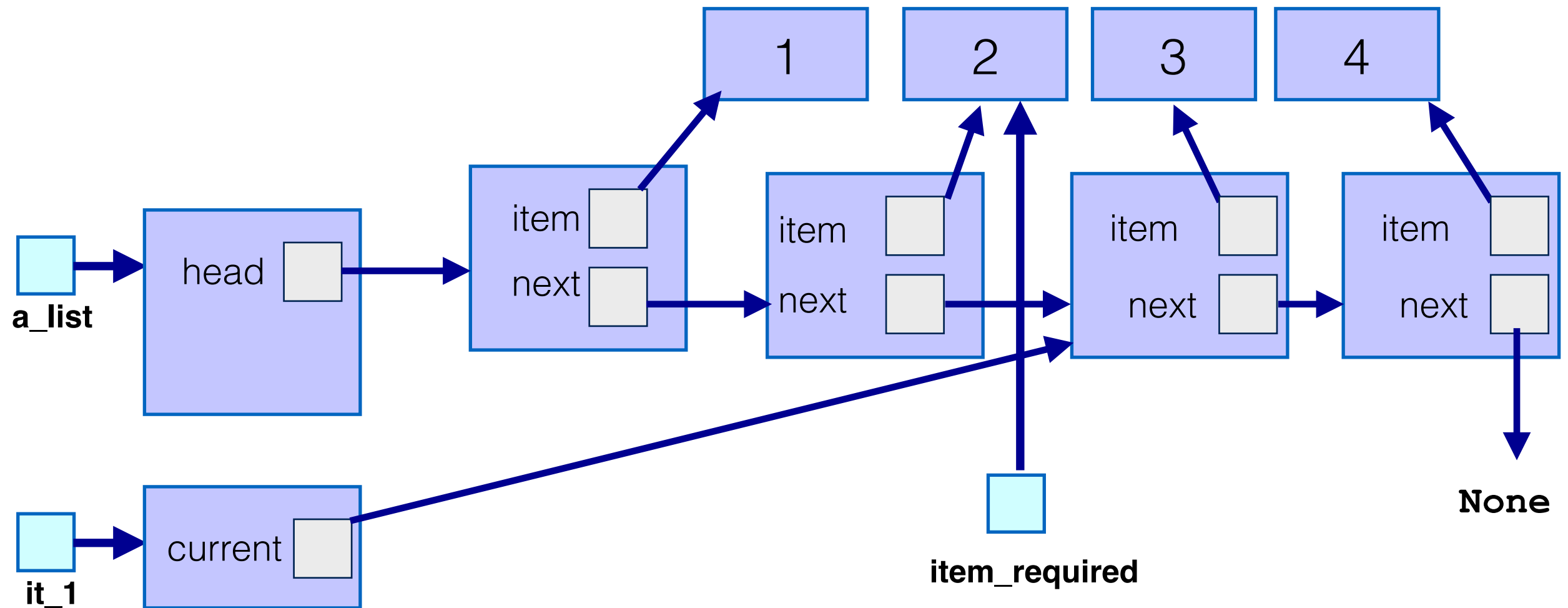


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

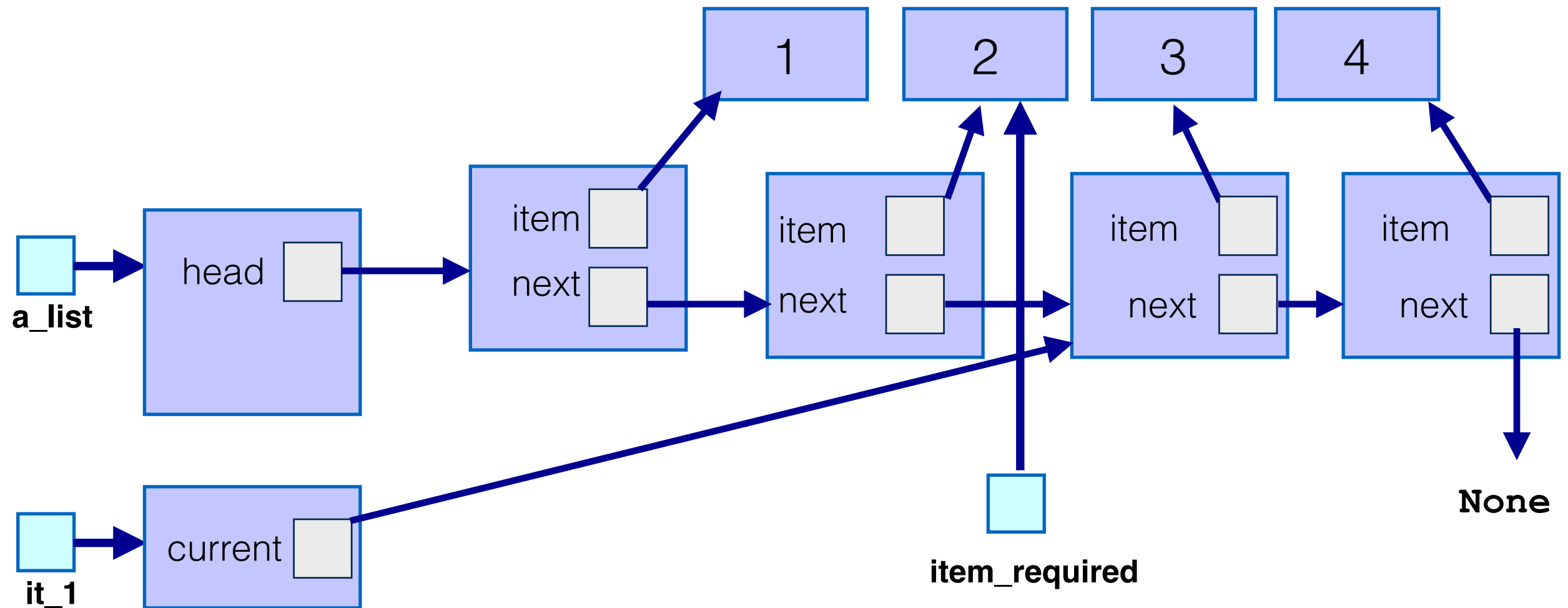


```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```

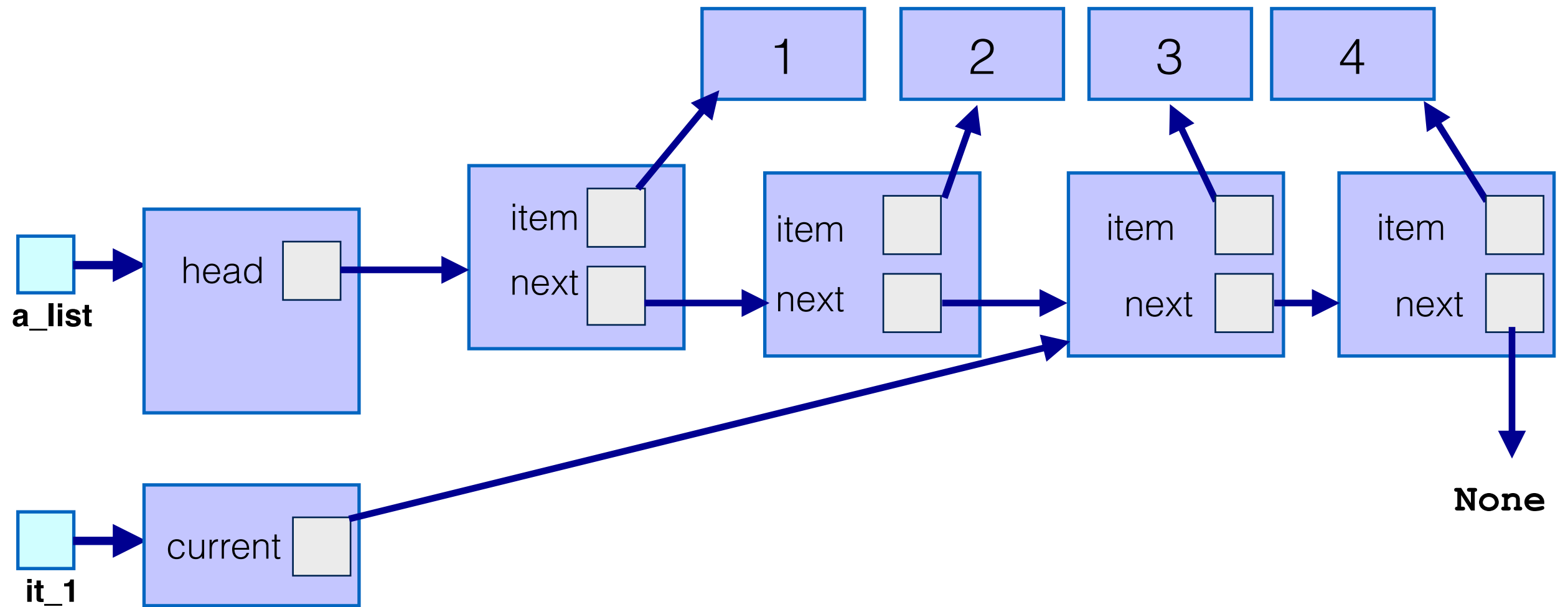



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
class ListIterator:
    def __init__(self, head):
        self.current = head

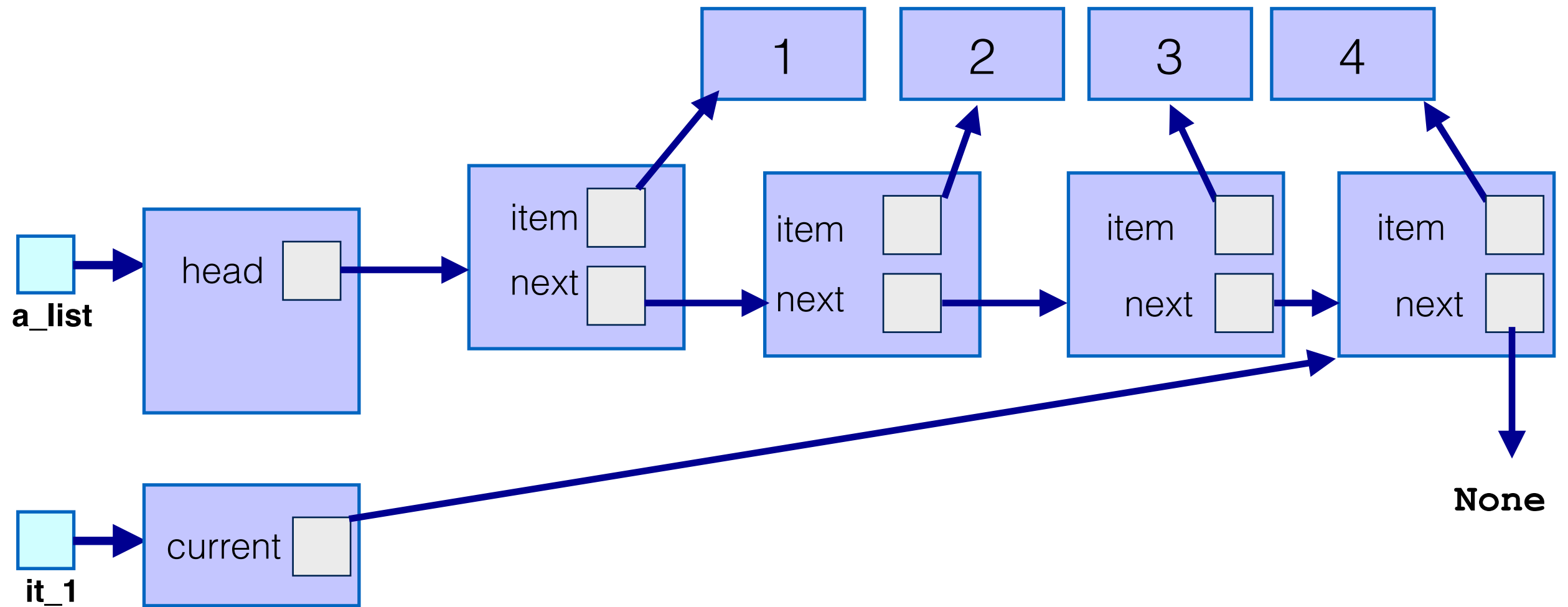
    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
```



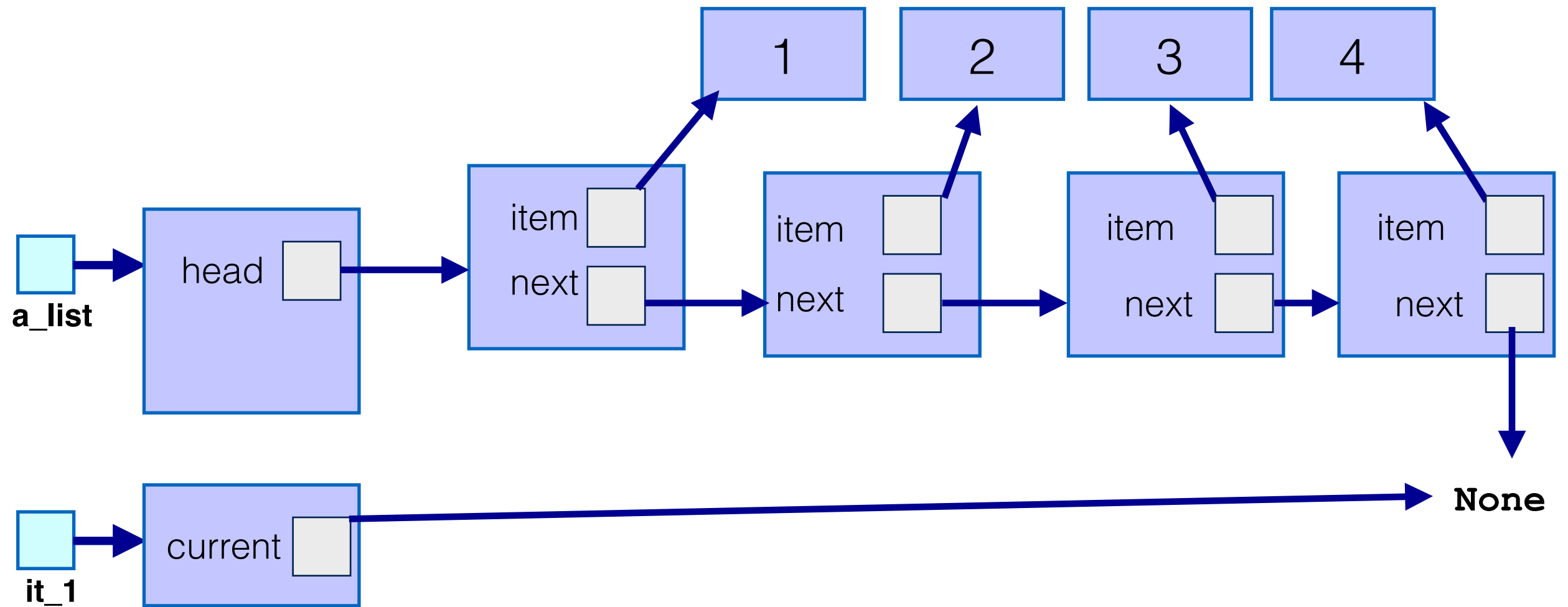
```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

2



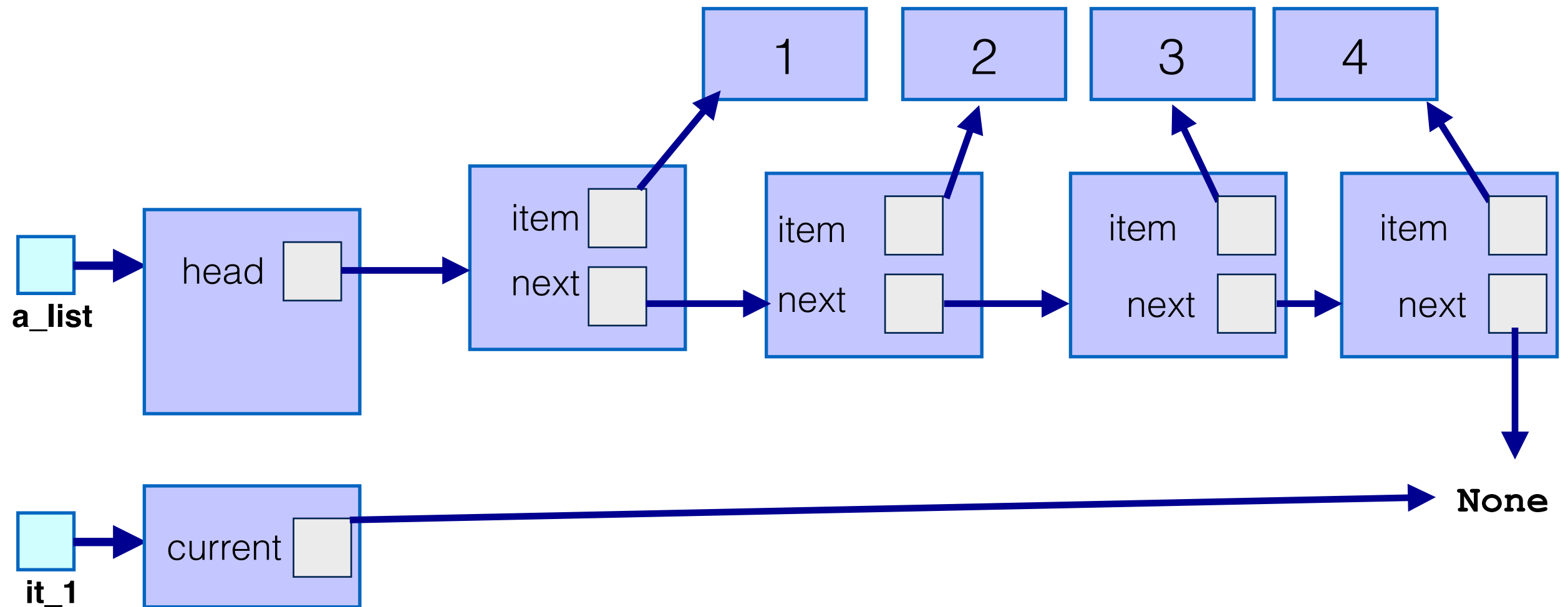
```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
2
>>> next(it1)
3
```



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
2
>>> next(it1)
3
>>> next(it1)
4
```



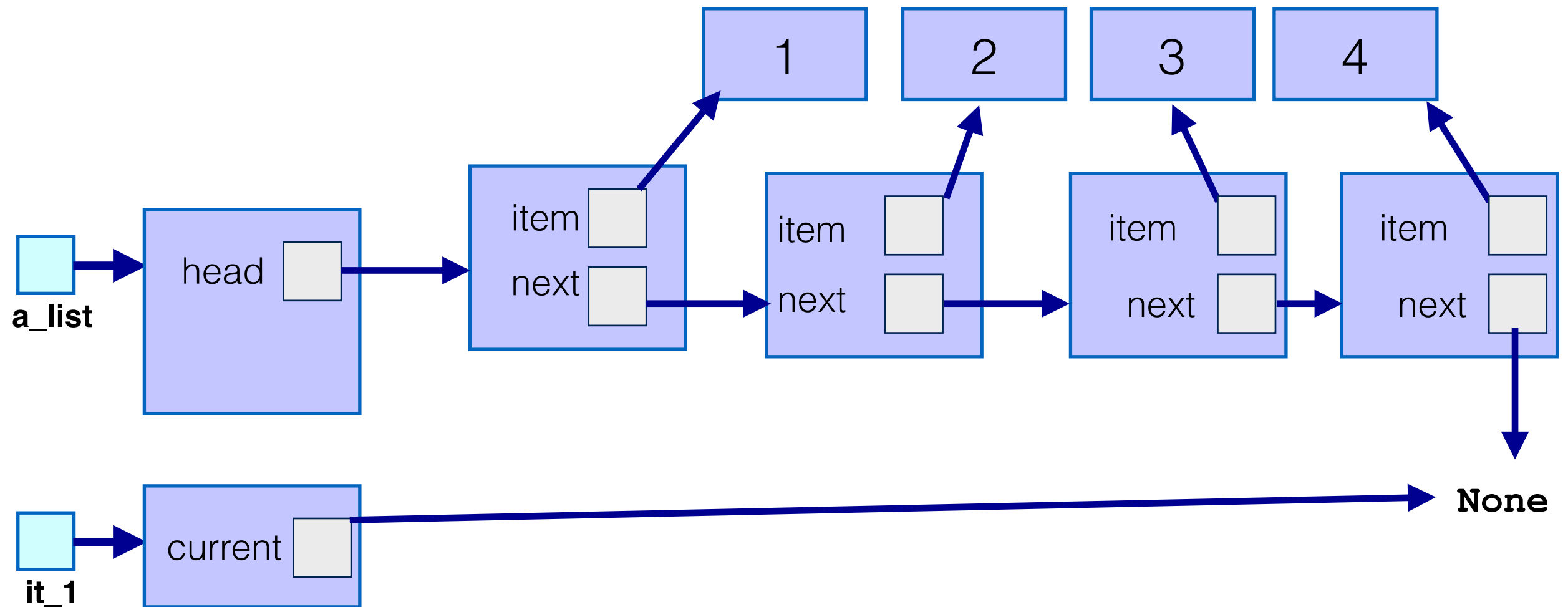
```
>>> a_list = List()
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required

>>> next(it1)
```

```
2
>>> next(it1)
3
>>> next(it1)
4
>>> next(it1)
```

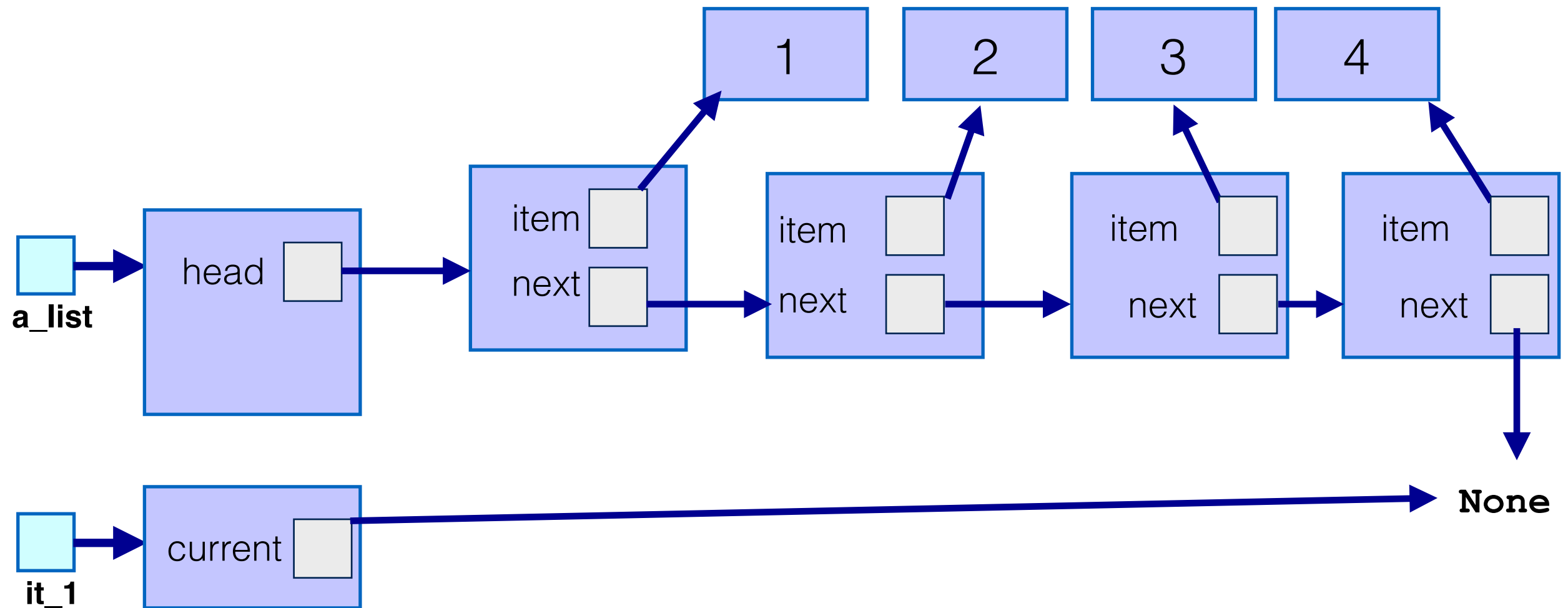


```
>>> a_list = List()
class ListIterator:
    def __init__(self, head):
        self.current = head

    def __iter__(self):
        return self

    def __next__(self):
        if self.current is None:
            raise StopIteration
        else:
            item_required = self.current.item
            self.current = self.current.next
            return item_required
>>> next(it1)
```

```
2
>>> next(it1)
3
>>> next(it1)
4
>>> next(it1)
Traceback ...:
  File ... in __next__
    raise StopIteration
```



```
>>> a_list = List()
>>> a_list.insert(0, 4)
>>> a_list.insert(0, 3)
>>> a_list.insert(0, 2)
>>> a_list.insert(0, 1)
>>> it1 = iter(a_list)
>>> next(it1)
1
>>> next(it1)
```

```
2
>>> next(it1)
3
>>> next(it1)
4
>>> next(it1)
Traceback ...:
  File ... in __next__
    raise StopIteration
```

Iterables and Iterators

- We have made our List class **iterable**: it implements an `__iter__` method that returns an Iterator on the list
- Objects of the ListIterator class are **iterators**: they implement `__iter__` and `__next__` methods

So let's use it

- Define `all_positive(a_list)` which returns **True** if all items are > 0 .
- You are a user: **outside** the class, no access to internals

```
def all_positive(a_list):  
    for item in a_list:  
        if item < 0:  
            return False  
    return True
```

List comprehensions

```
>>> A = [3*x for x in range(10)]  
>>> A  
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]  
>>> B = [x for x in A if x % 2 == 0]  
>>> B  
[0, 6, 12, 18, 24]
```

List comprehensions allow you to:

- Performing some operation for every element
- Selecting a subset of elements that meet a condition
- AND return a list

```
def all_positive(a_list):  
    for item in a_list:  
        if item < 0:  
            return False  
    return True
```

```
def all_positive(a_list):  
    return [] == [e for e in a_list if e <= 0]
```

maximum_item

- Define **maximum_item(a_list)** to find maximum of the items in a_list
- Assume you are a **user**: outside the class, no access to internals
- Need to use an iterator.

```
def max(a_list):  
    try:  
        it = iter(a_list)  # construct an iterator  
        max_val = next(it) # get the first element  
        for item in it:    # traverse the rest of the list  
            if max_val < item:  
                max_val = item  
        return max_val  
    except StopIteration:  
        raise Exception("The list is empty")
```

Fibonacci Numbers

0. 1. 1, 2, 3, 5, 8, 13, ...

```
class Fibonacci:
    def __init__(self, maximum):
        self.maximum = maximum
        self.count = 0
        self.a = 0
        self.b = 1

    def __iter__(self):
        return self

    def __next__(self):
        next_fib = self.a
        self.count += 1
        if self.count > self.maximum:
            raise StopIteration
        self.a = self.b
        self.b = next_fib + self.b
        return next_fib
```

```
>>> for i in Fibonacci(10):
...     print(i, end=" ")
...
0 1 1 2 3 5 8 13 21 34 >>>
```

Summary

- How to make lists iterable by implementing an iterator for them
- How to construct a simple traversal iterator
- How to use iterators to define functions
- How to use iterators to generate sequences of items