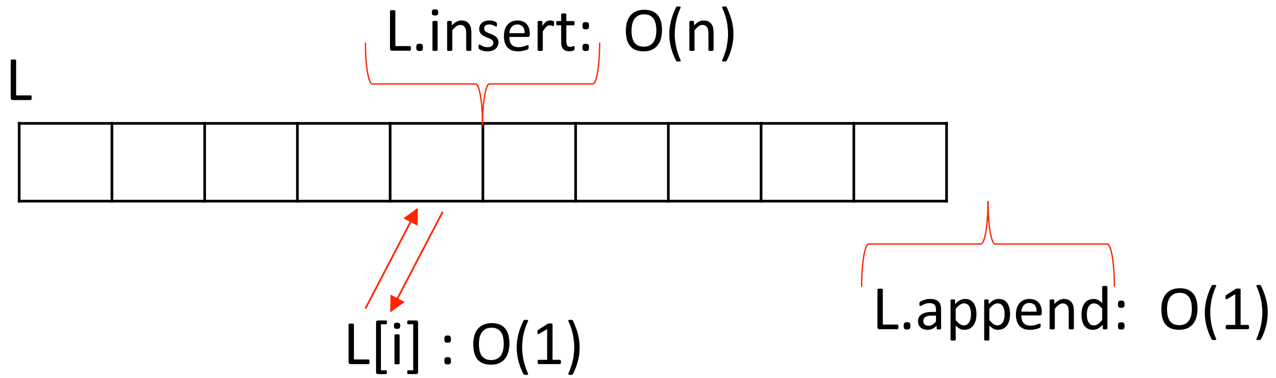
# CSc 120

# Introduction to Computer Programming II

*Adapted from slides*
*by Dr. Saumya Debray*

## 10: Linked Lists

# Python lists: reprise



L.insert: O(n)

L

L[i] : O(1)

L.append: O(1)

concatenating two lists: O(n)

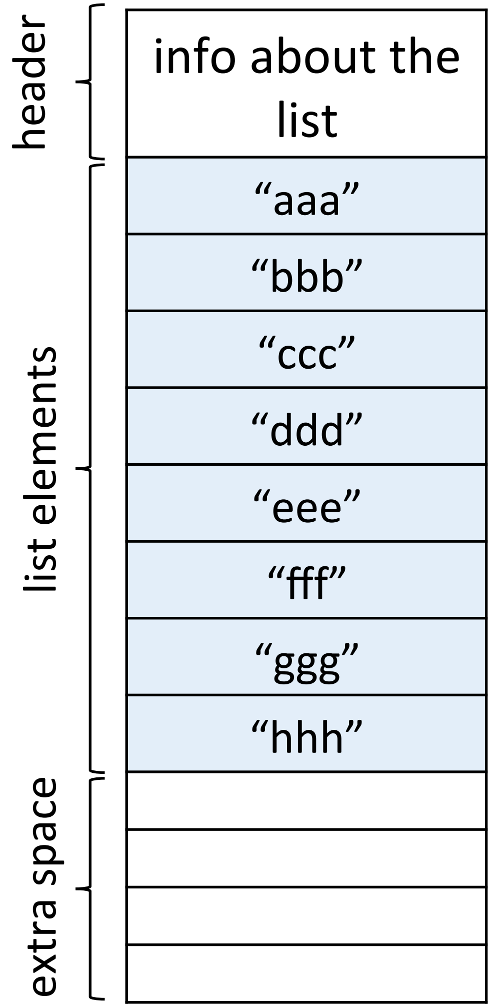
**Question**: Can we do insertion and concatenation in O(1) time?

(complexity of other operations may change).

⇒ **"Linked list"**

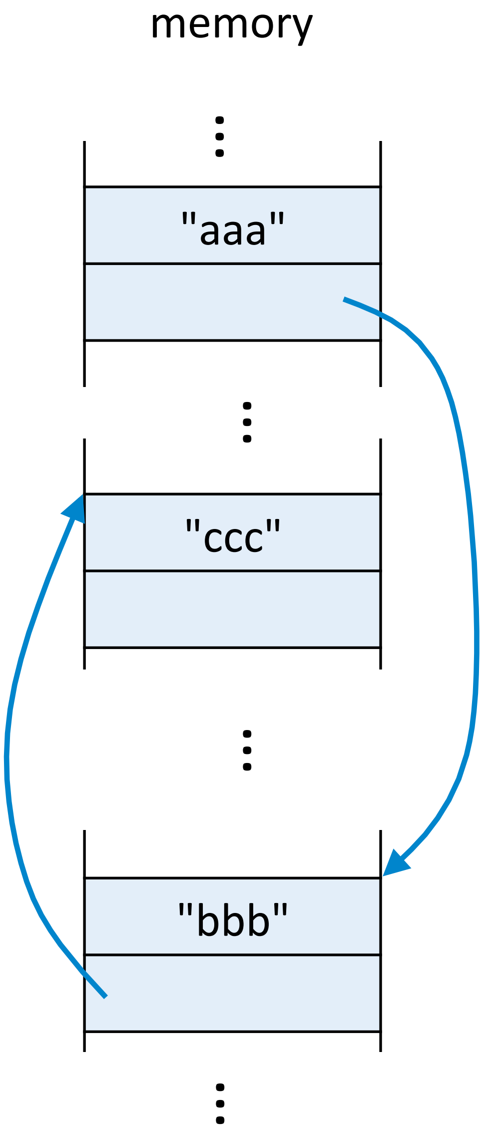
# Python lists: reprise

- Key feature: L[i] and L[i+1] are adjacent in memory
- This makes accessing L[i] very efficient
  - O(1)
- Insertion and concatenation require moving O(n) elements
  - O(n)



# Linked lists

- To get O(1) insertion and concatenation, we cannot afford to move O(n) list elements
- We have to relax the requirement that $i^{th}$ element is adjacent to $(i+1)^{st}$ element
  - any element can be anywhere in memory
- Each element has to tell us where to find the next element



# linked lists

# Linked lists

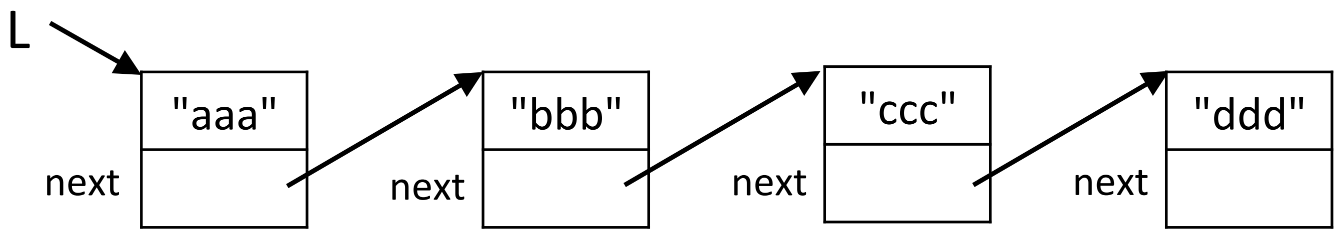
- Linked list:

  A collection of elements where each element has a value and a reference to the next element.

  There is at least one variable that references the beginning of the list.

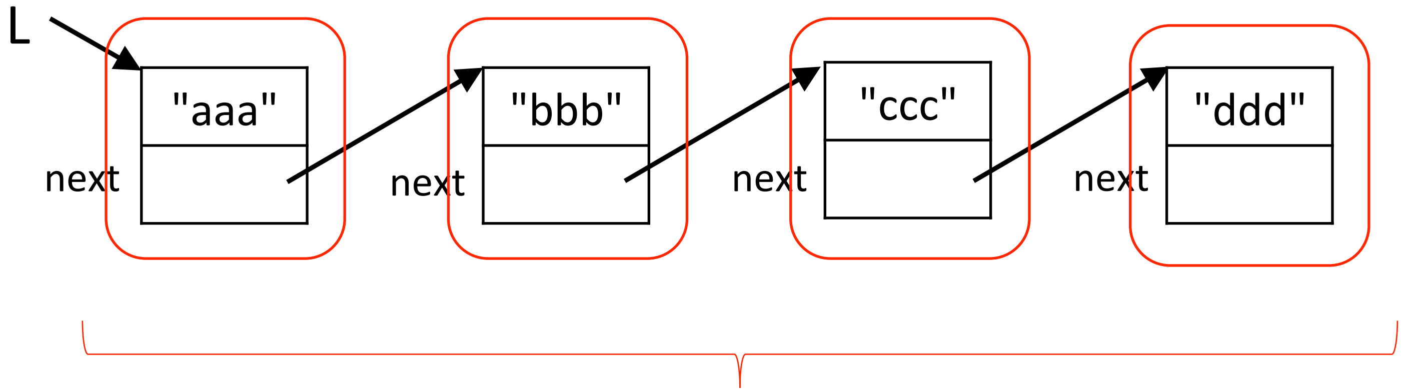
# Linked lists

Each element of the list has a reference to the next list element



# Linked lists

With each element of the list, keep a reference to the next list element

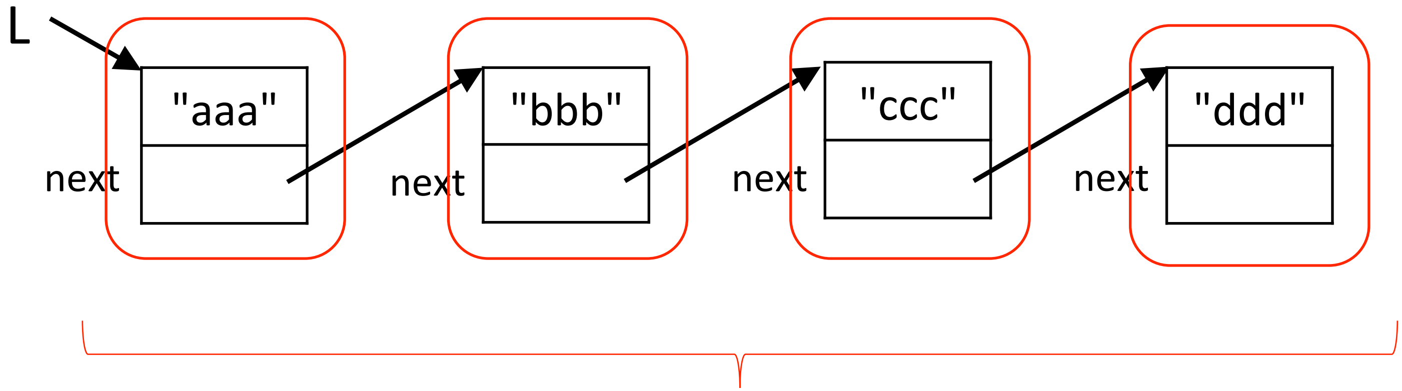


"nodes"

each node in the list has a reference to the next node

# Linked lists

With each element of the list, keep a reference to the next list element



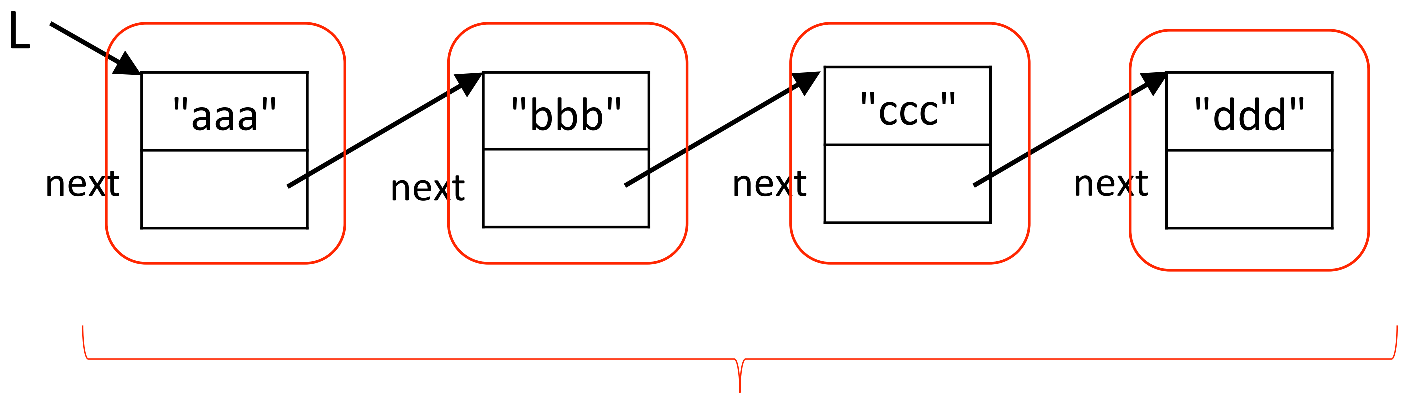
"nodes"

each node in the list has a reference to the next node

Let's explore this idea using a file for a "node"

# Linked lists

With each element of the list, keep a reference to the next list element



"nodes"

Let's explore this idea using a file for a "node"

each file has two lines:

- the first line is a value
- the second line is a reference to the next "node" (file)

# Linked lists

- Let's explore this idea using a file for a "node"
  - each file has two lines:
    - the first line is a value
    - the second line is a reference to the next "node" (file)

Sample file “node”: filename is 24.txt

value: aaa

next: 3.txt

# EXERCISE

- Exploring linked lists using files as nodes

How would we add the word "total" to our linked lists of files so that the sentence reads:

*The expert in anything was once a total beginner.*

# EXERCISE

- Exploring linked lists using files as nodes

  How would we add the word "total" to our linked lists of files so that the sentence reads:

  *The expert in anything was once a total beginner.*

  Create a new "node" (a new file)

  The first line is "total"

  The second line is 19.txt

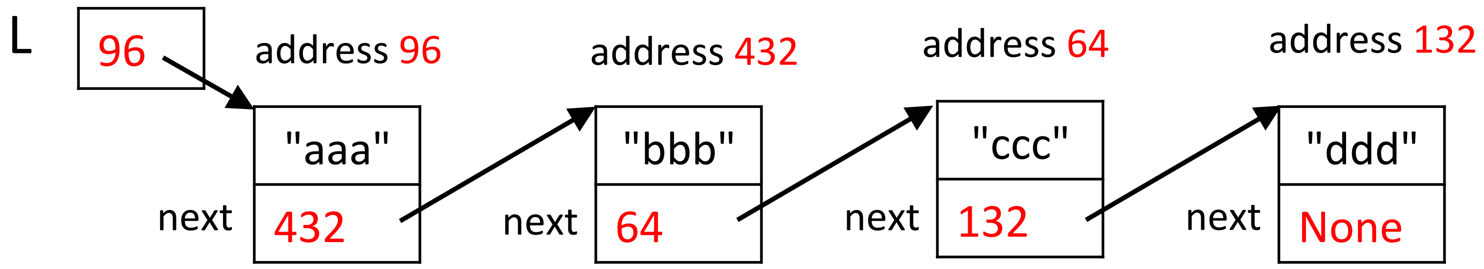
  What else do we have to do?

  \- modify the file node for the value "a" to change its reference

# Linked lists

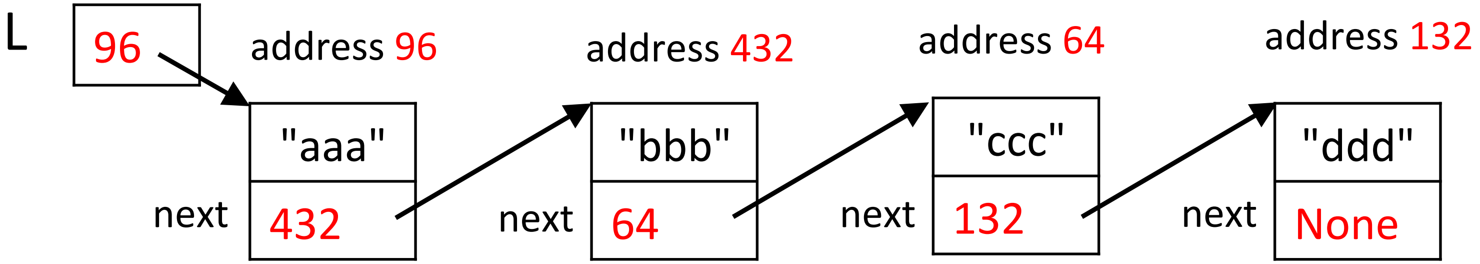
References are addresses in memory.

Here is the diagram with explicit addresses (simplified).



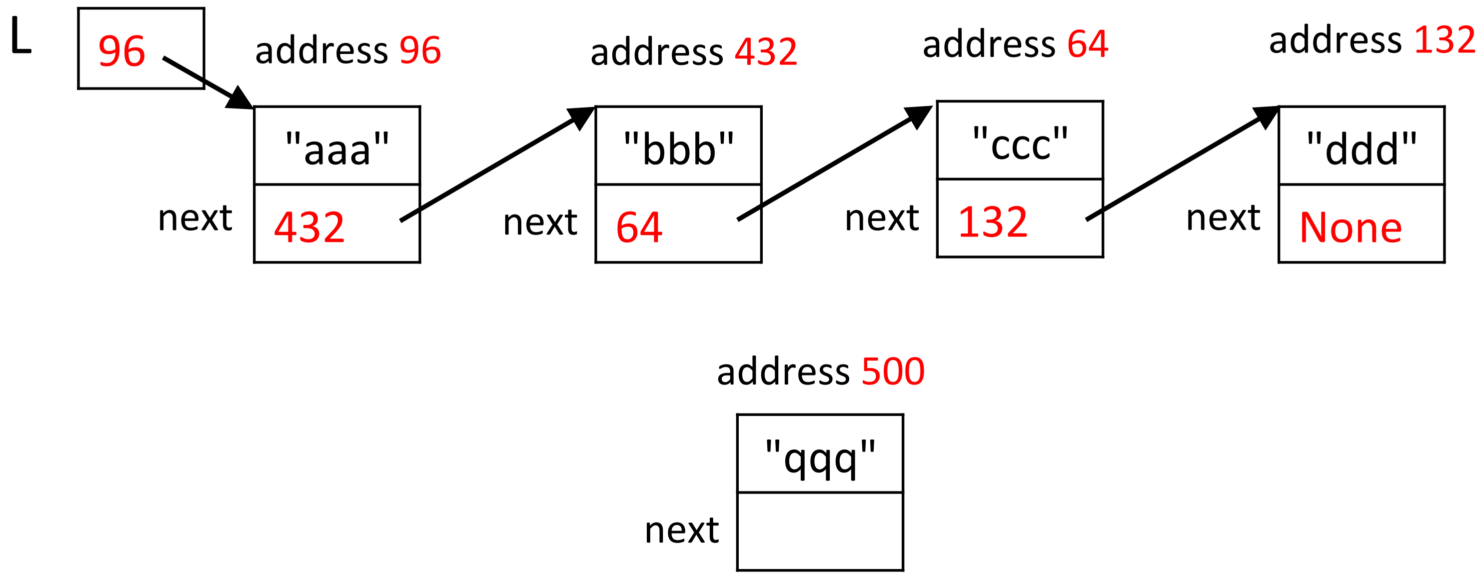
# Insertion

Consider inserting a new node into the linked list



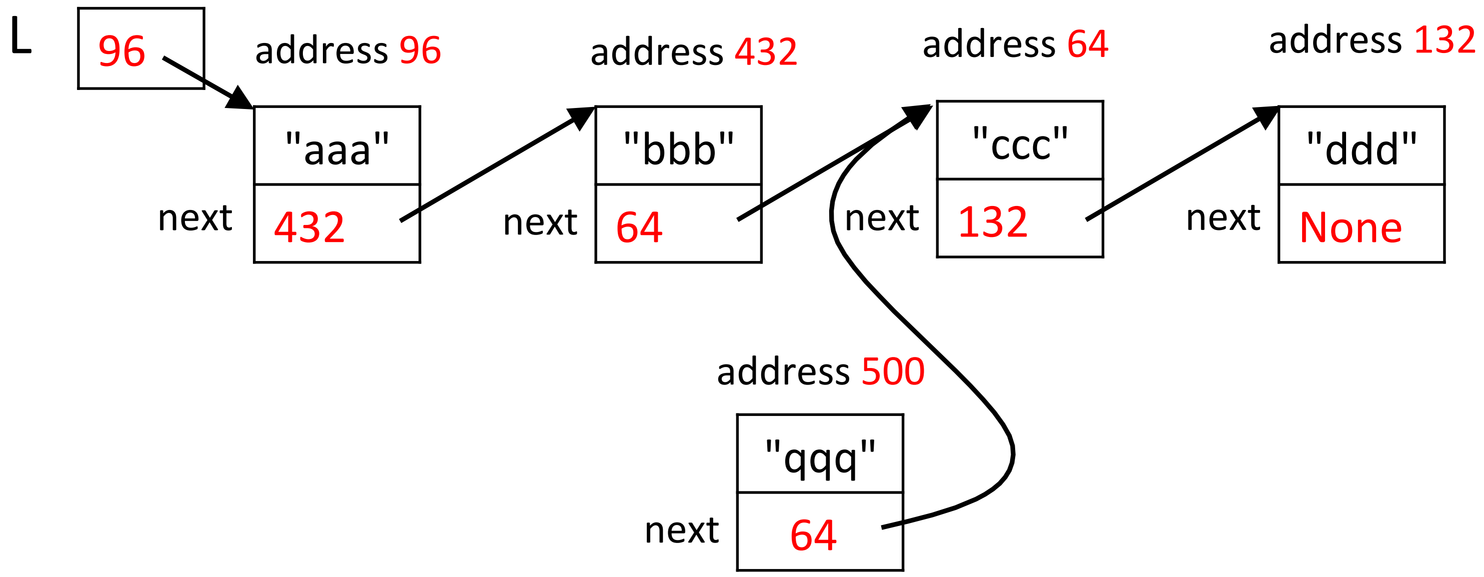
# Linked lists

Specifically, add a new node between "bbb" and "ccc". What do we change?



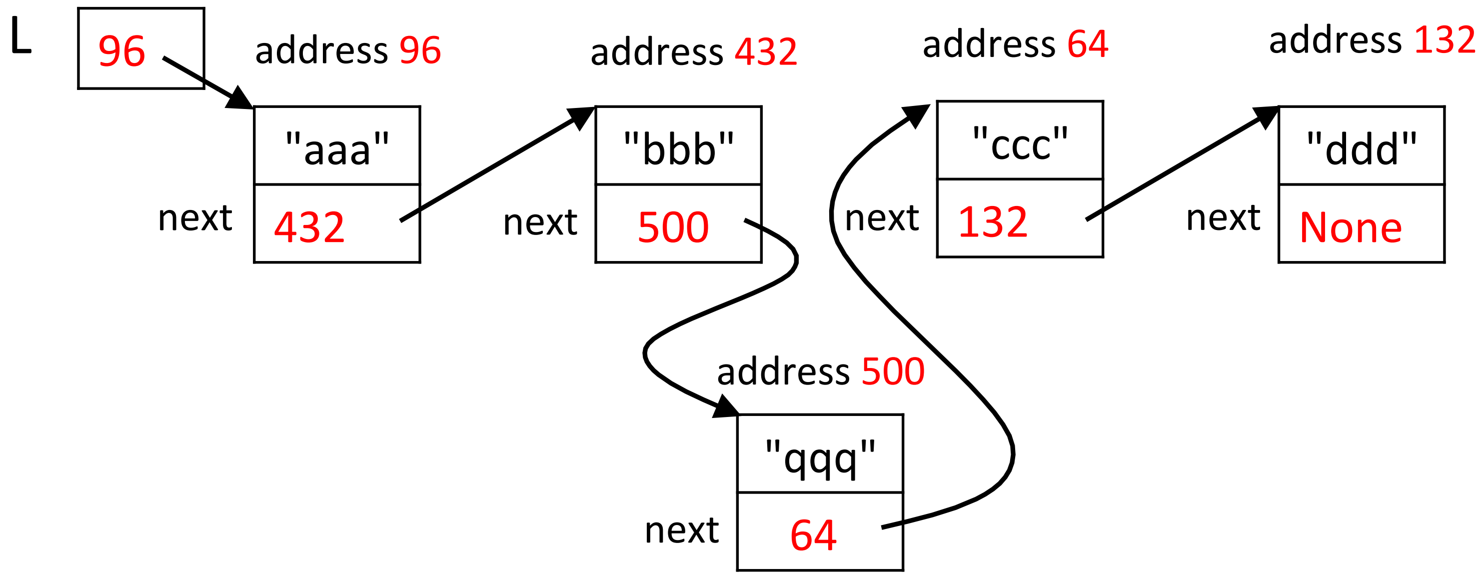
# Insertion

Specifically, add a new node between "bbb" and "ccc". What do we change?



# Insertion

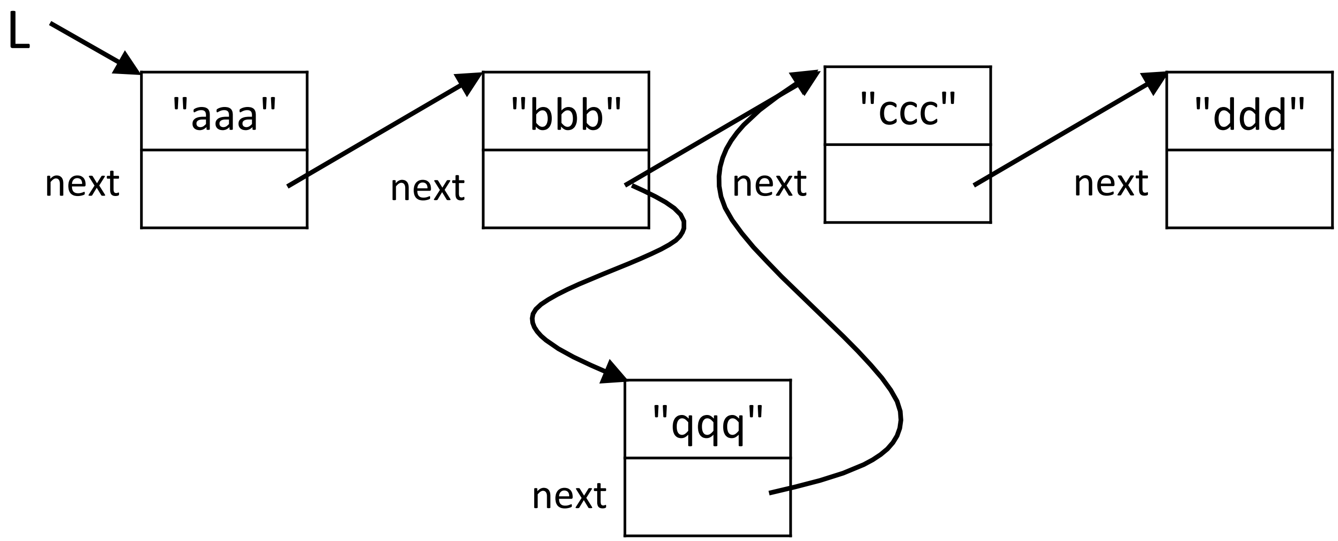
We want to add a new node between "bbb" and "ccc". What do we change?



# Insertion

O(1)*

Set the next references appropriately. What is the complexity of insertion?

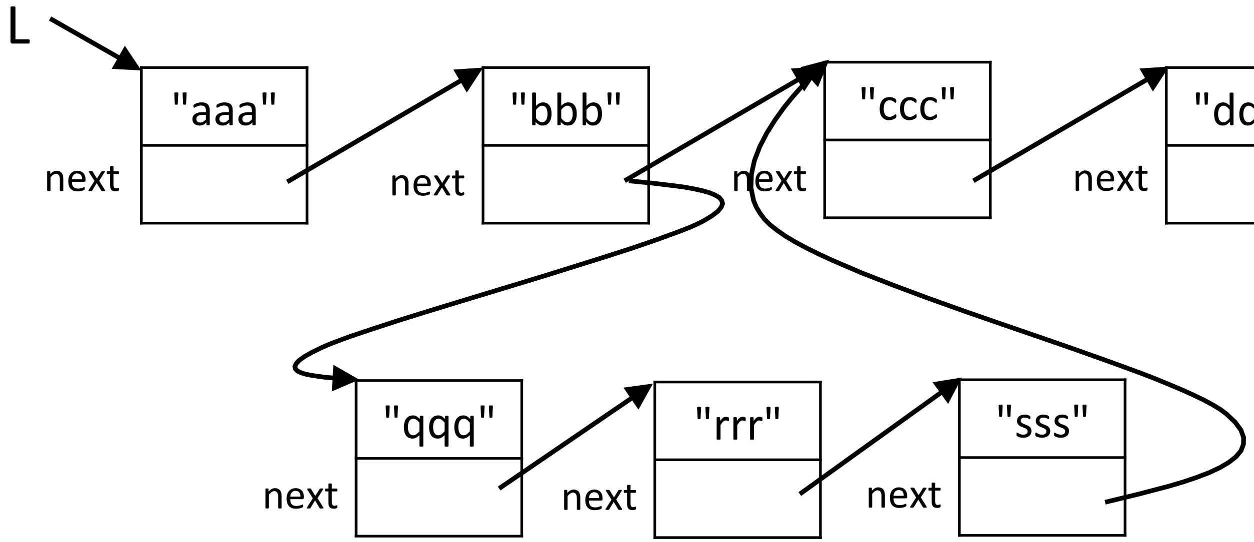


*assuming we have a reference to the node of insertion

# Insertion

O(1)

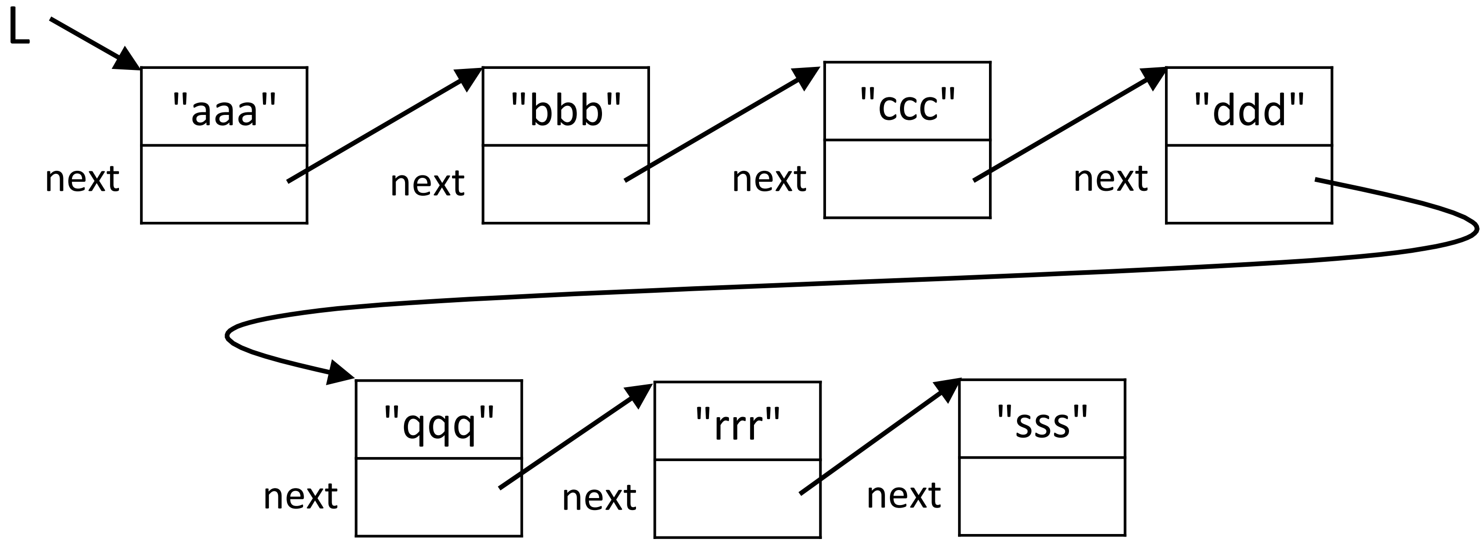
To insert an element into a linked list: set next references appropriately



# Concatenation

O(1)*

To concatenate two linked lists: set next reference of end of first list to refer to beginning of second list



* once we have a reference to the end of the first list

# implementation

# Nodes: Implementation

```
class Node:
    def __init__(self, value):
        self._value = value  # reference to the object at that node
        self._next = None  # reference to the next node in the list
```

*Getters*:

```
def value(self):
    return self._value
```

```
def next(self):
    return self._next
```

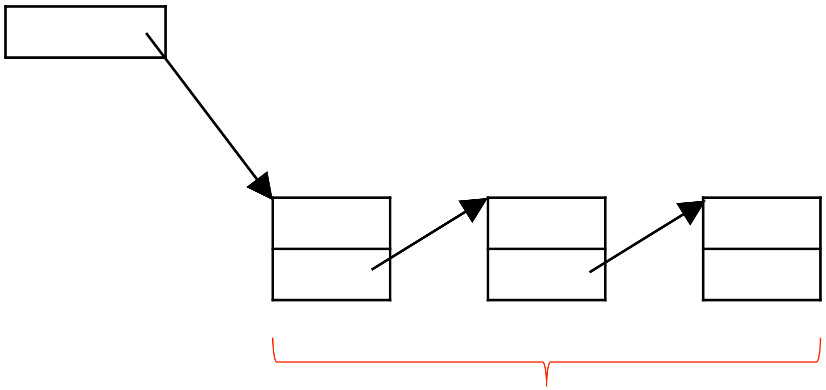
*Setters*:

```
def set_value(self, value):
    self._value = value
```

```
def set_next(self, next):
    self._next = next
```

# Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes

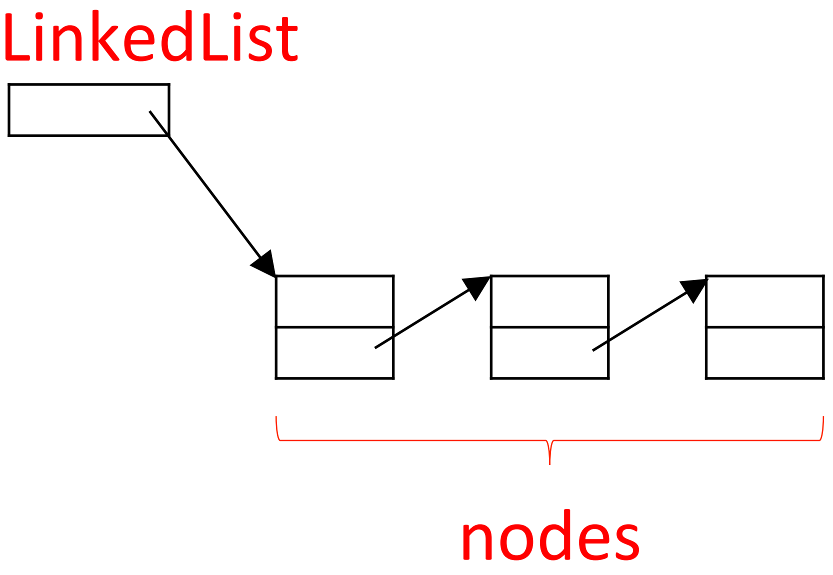


LinkedList

nodes

# Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes



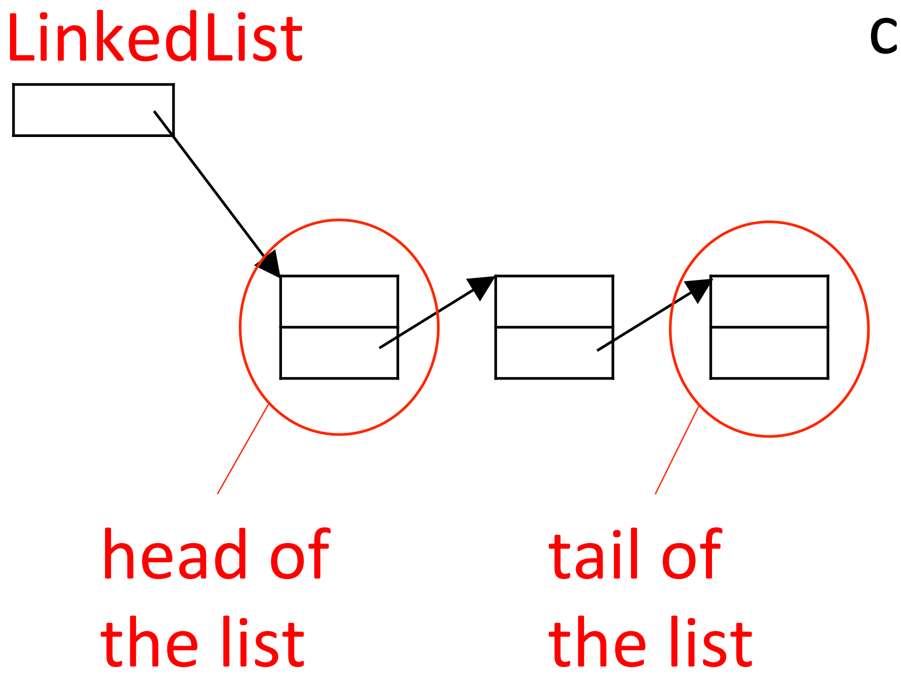
LinkedList

nodes

```
class LinkedList:
    def __init__(self):
        self._head = None
```

# Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes



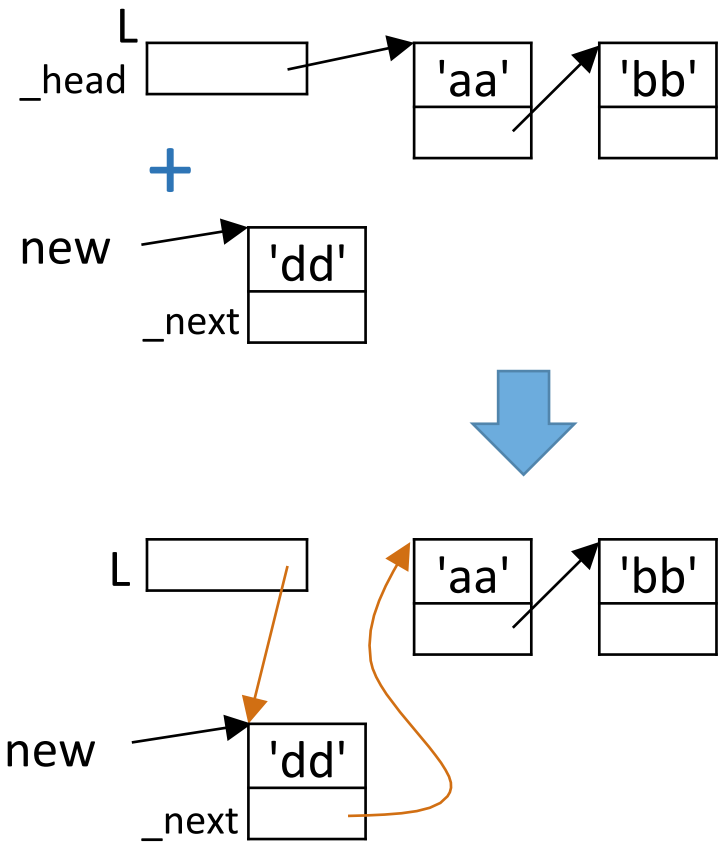
```
class LinkedList:
    def __init__(self):
        self._head = None
```

# Linked Lists: Implementation

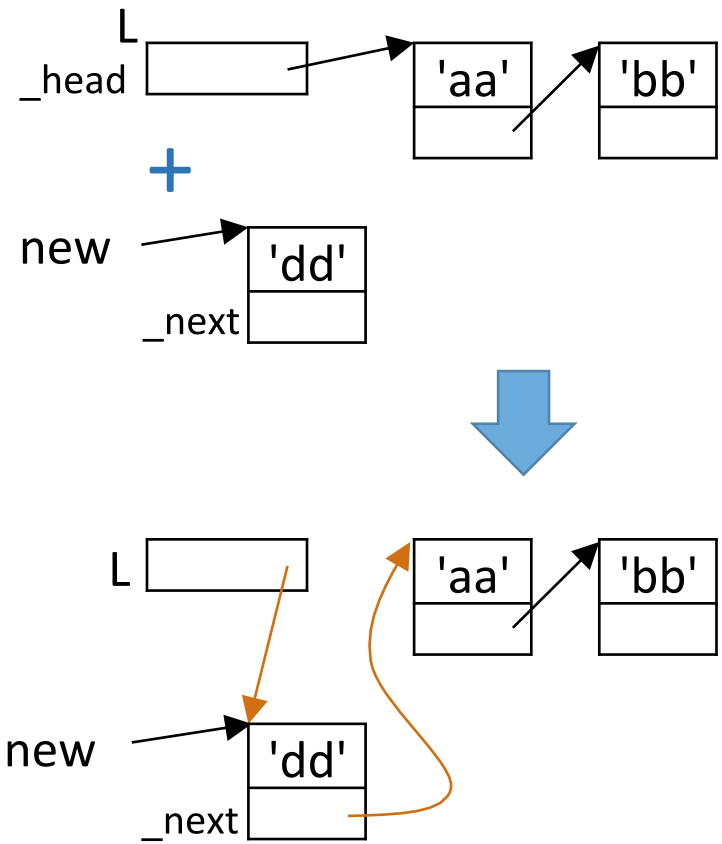
```
class LinkedList:
    def __init__(self):
        self._head = None

    def is_empty(self):
        return self._head == None

    def head(self):
        return self._head
```

# addition
at the head of the list

# Adding a node at the head

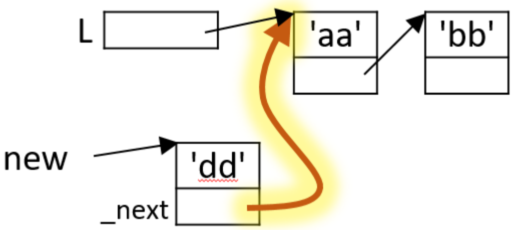


# Adding a node at the head

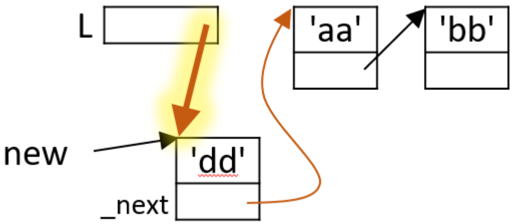


Sequence of operations for an add method:

1. new._next = L._head



2. L._head = new



# Adding a node at the head

```
class LinkedList:
  def __init__(self):
     self._head = None

  # add a node new at the head of the linked list
  def  add(self, new):
     new._next = self._head
     self._head = new
```

O(1)

# Creating a linked list: Example

```
class Node:
   def _ _init_ _(self, value):
      self._value = value
      self._next = None

   . . .
class LinkedList:
   def _ _init_ _(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

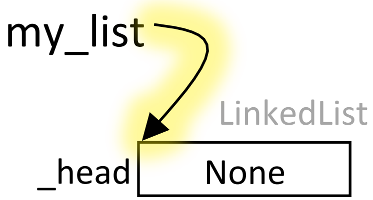
| infile.txt |
|---|
| aa |
| bb |
| cc |

# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |

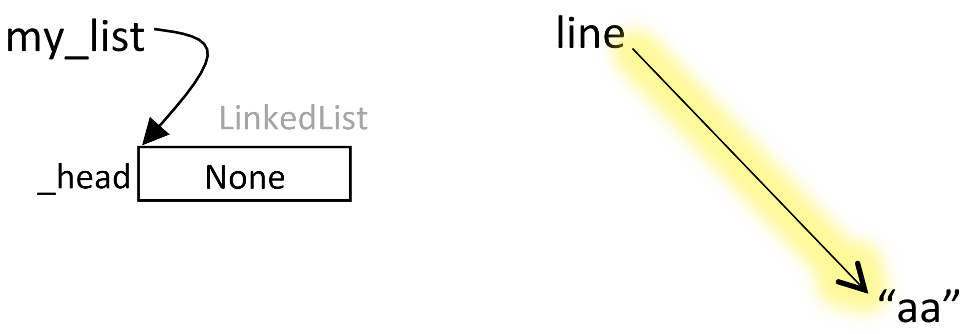


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |



my_list

LinkedList

_head None

line

"aa"

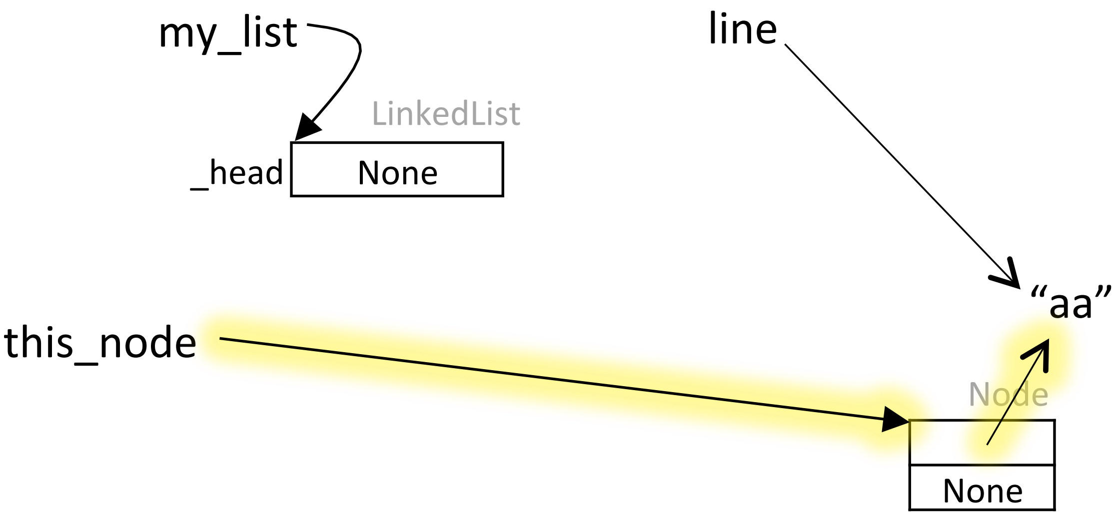
# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

infile.txt

| |
|---|
| aa |
| bb |
| cc |

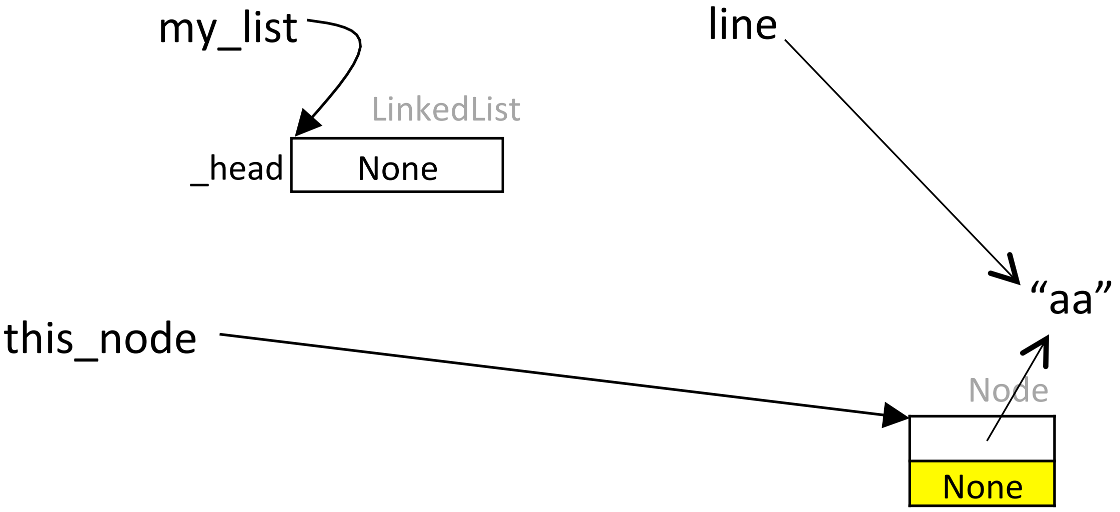


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
→     new._next = self._head
      self._head = new
```

```
  infile = open(“infile.txt”)
  my_list = LinkedList()
  for line in infile:
     this_node = Node(line)
→    my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |

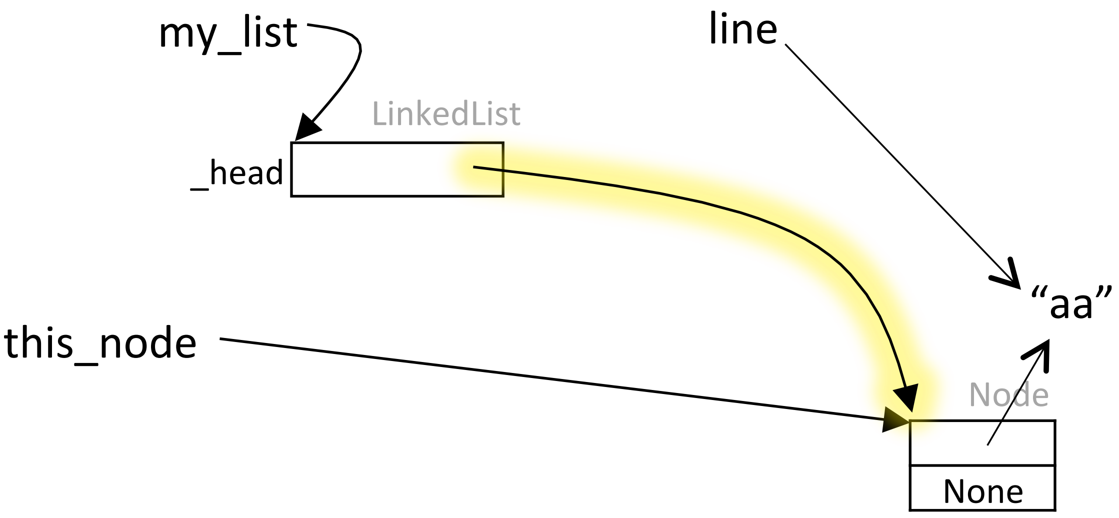


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

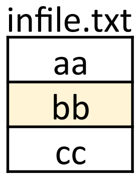
| infile.txt |
|---|
| aa |
| bb |
| cc |

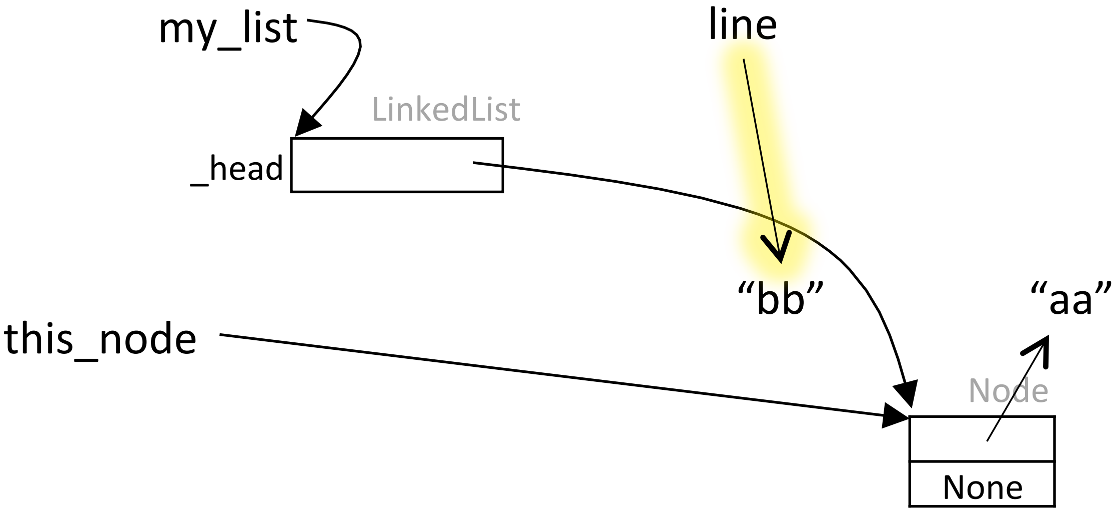


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```



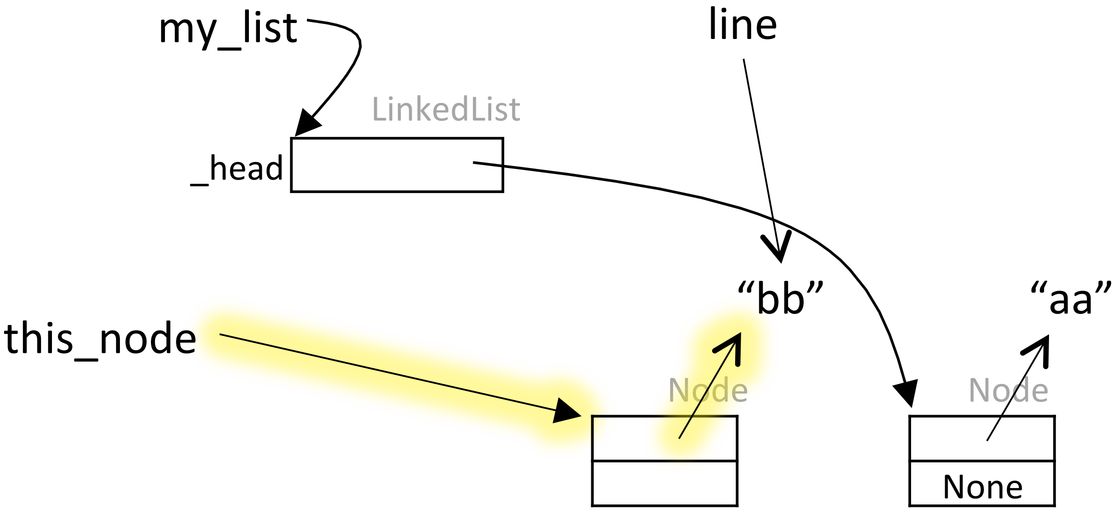


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |

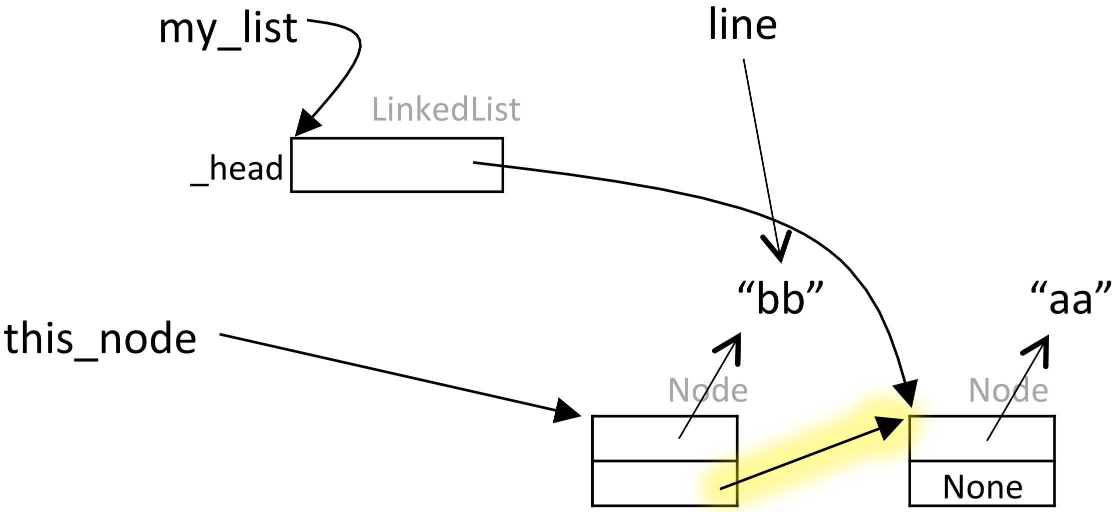


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

  . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |

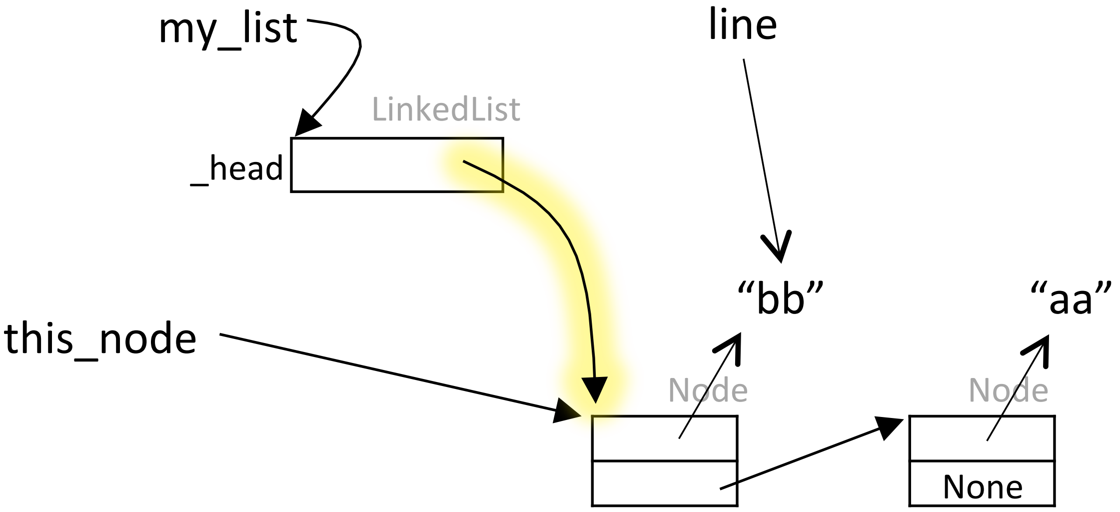


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |



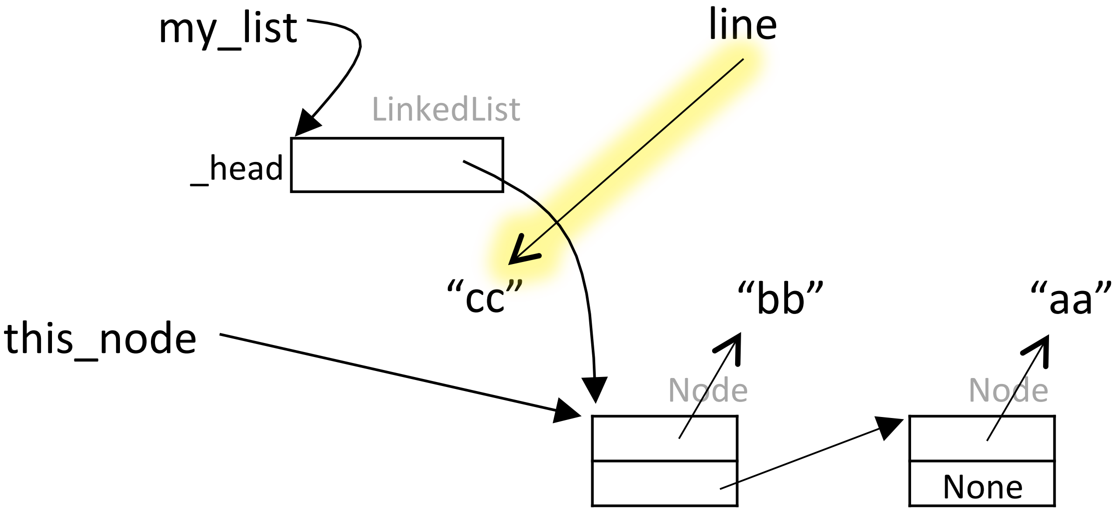
# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

infile.txt

| |
|---|
| aa |
| bb |
| cc |

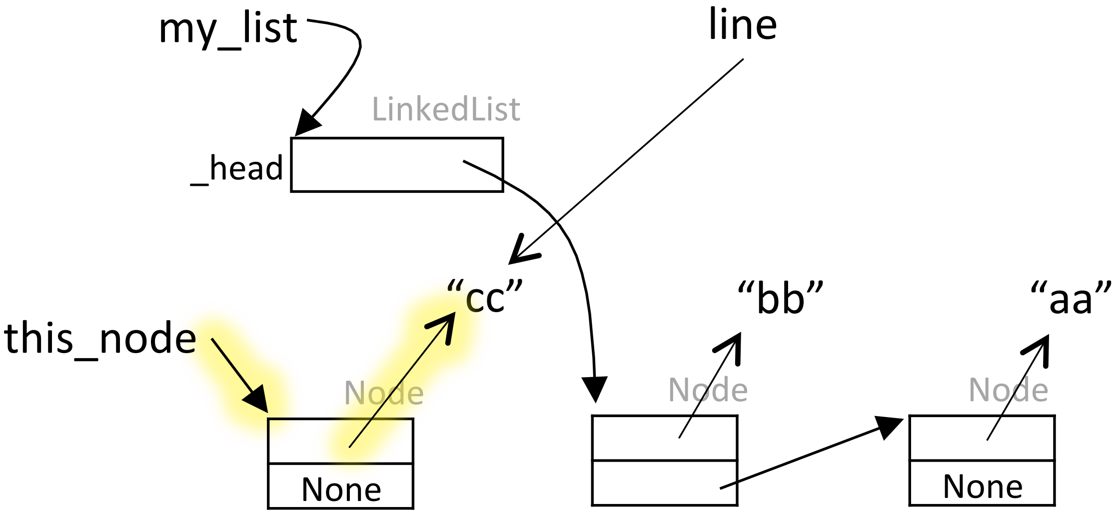


# Creating a linked list: Example

```
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

    . . .

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |

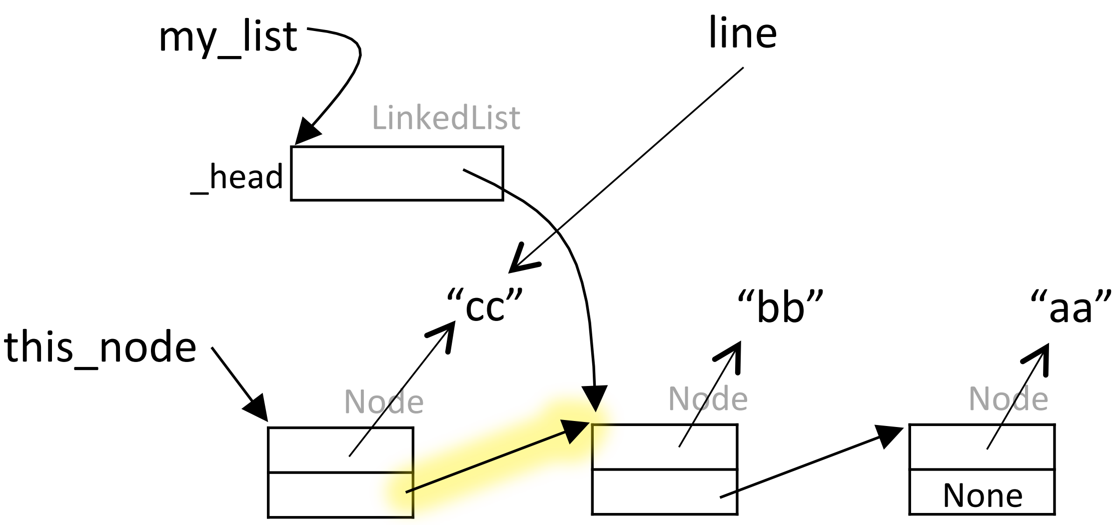


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |

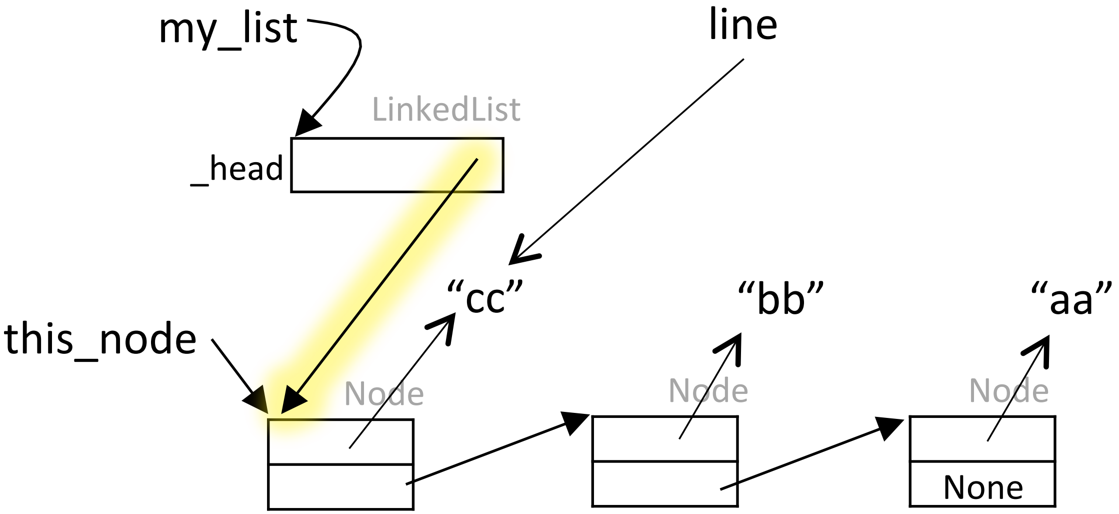


# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

| infile.txt |
|---|
| aa |
| bb |
| cc |



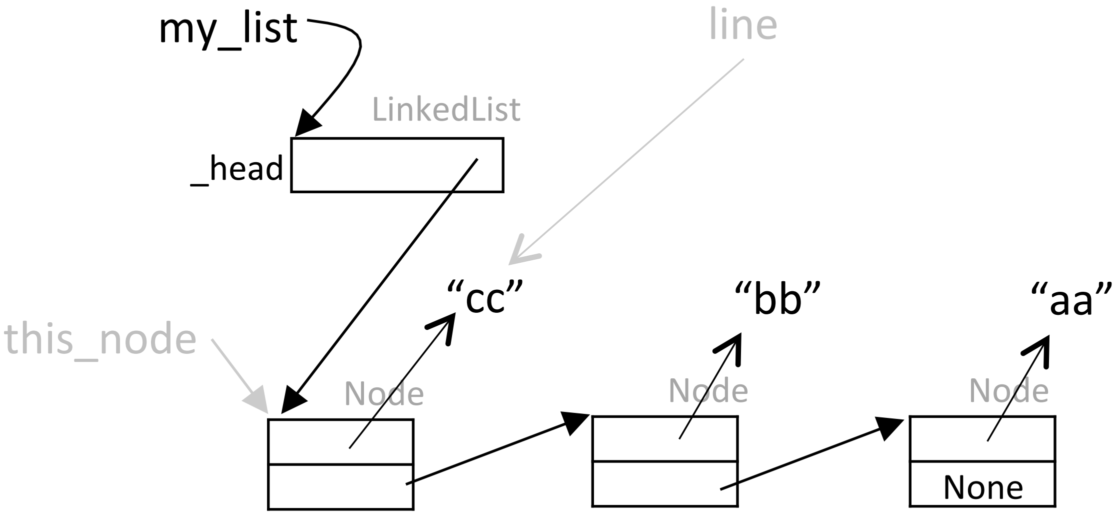
# Creating a linked list: Example

```
class Node:
   def __init__(self, value):
      self._value = value
      self._next = None

   . . .

class LinkedList:
   def __init__(self):
      self._head = None

   def add(self, new):
      new._next = self._head
      self._head = new
```

```
infile = open(“infile.txt”)
my_list = LinkedList()
for line in infile:
   this_node = Node(line)
   my_list.add(this_node)
```

infile.txt

```
aa
bb
cc
```

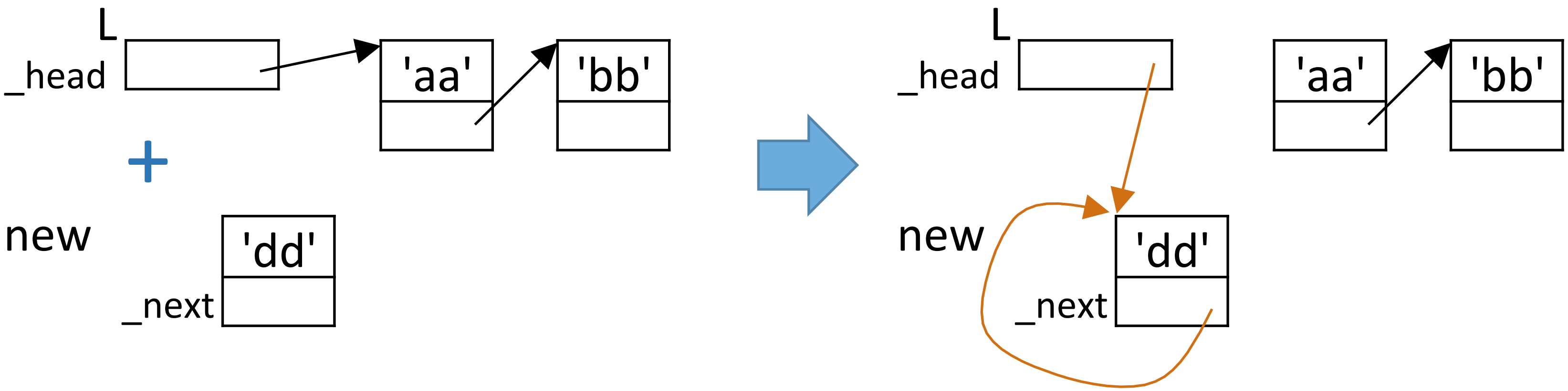


# Adding a node at the head

Changing the order of assignments does not work:

```
def broken_add(self, new):
        self._head = new
        new._next = self._head
```

```
def  add(self, new):
        new._next = self._head
        self._head = new
```



# appending
# to the tail of the list

# Adding a node at the tail

To add a node X at the end (i.e., tail) of a list L:

1. find the last element Y of L
2. Y._next = X

# Adding a node at the tail

To add a node X at the end (i.e., tail) of a list L:

1. find the last element Y of L — O(n)
2. Y._next = X — O(1)

# Adding a node at the tail

To add a node X at the end (i.e., tail) of a list L:

1. find the last element Y of L
2. Y._next = X

Gotchas to watch out for:

- what if there is no last element?
  - how can we tell?
  - what should we do?

**EXERCISE**

# EXERCISE

- Consider a linked list whose value attributes consist of strings.

- Write a method `replace(arg1, arg2)` that replaces the value attributes of all nodes that equal `arg1` with `arg2`.

# finding the $n^{th}$ element

# Finding the nth element

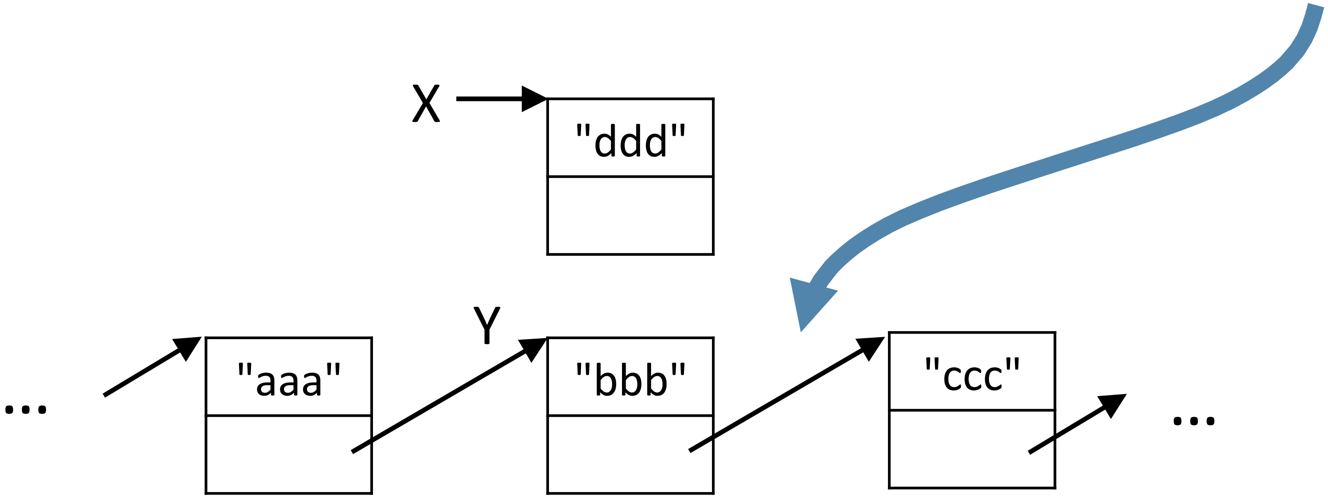
```
class LinkedList:

    # return the node at position n of the linked list
    def  get_element(self, n):
        elt = self._head
         while elt != None and n > 0:
            elt = elt._next
            n –= 1
        return elt
```

O(n)

# insertion

# Inserting a node

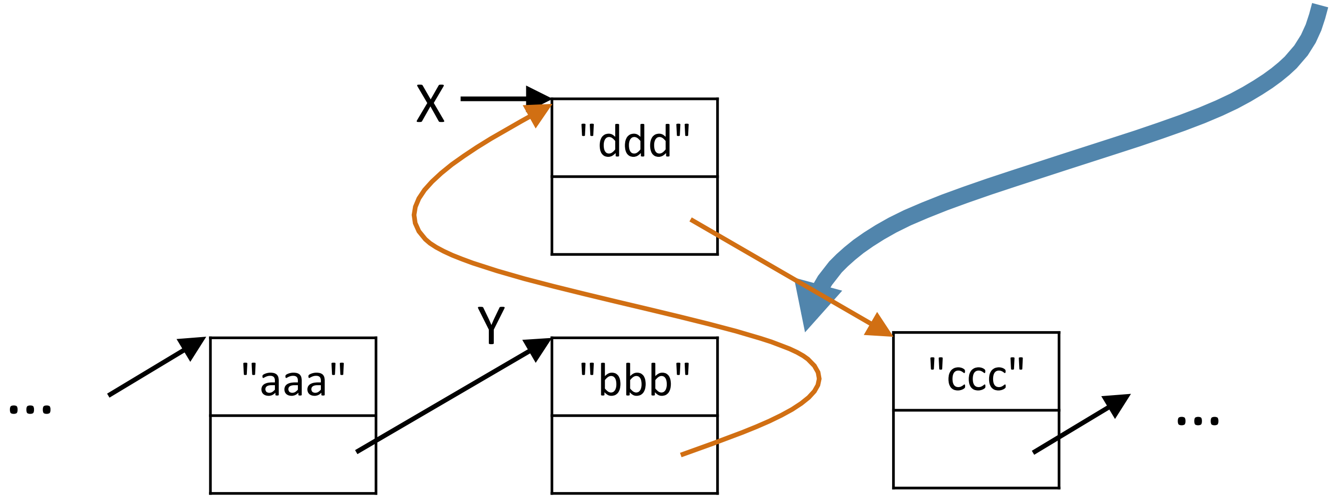
Suppose we want to insert a node X into a list here:



Then we have to adjust the next-node reference on the node Y just before that position

# Inserting a node

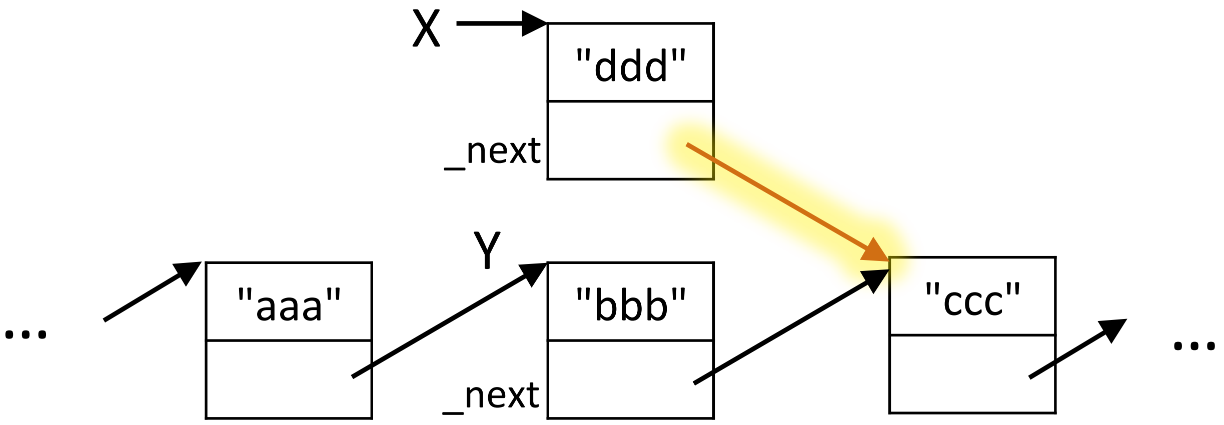
Suppose we want to insert a node X into a list here:



Then we have to adjust the next-node reference on the node Y just before that position

# Inserting a node

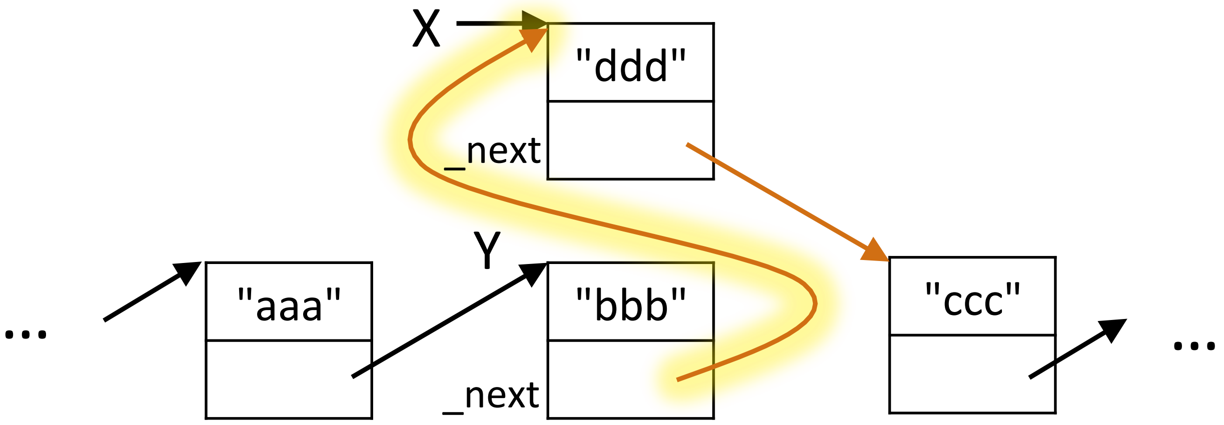
The order of operations is important:



1. X._next = Y._next

# Inserting a node

The order of operations is important:



1. X._next = Y._next
2. Y._next = X

# Inserting a node

Inserting a node X at position *n* in a list L:

1. find the node Y at position *n*−1
   - iterate *n*−1 positions from the head of the list*
2. insert X after Y
   - adjust next-node references as in previous example

```
Y =  L._head
for i in range(n-1):
    Y = Y._next
```

O(n)

```
X._next = Y._next
Y._next = X
```

O(1)

* do something sensible if the list has fewer than n-1 nodes

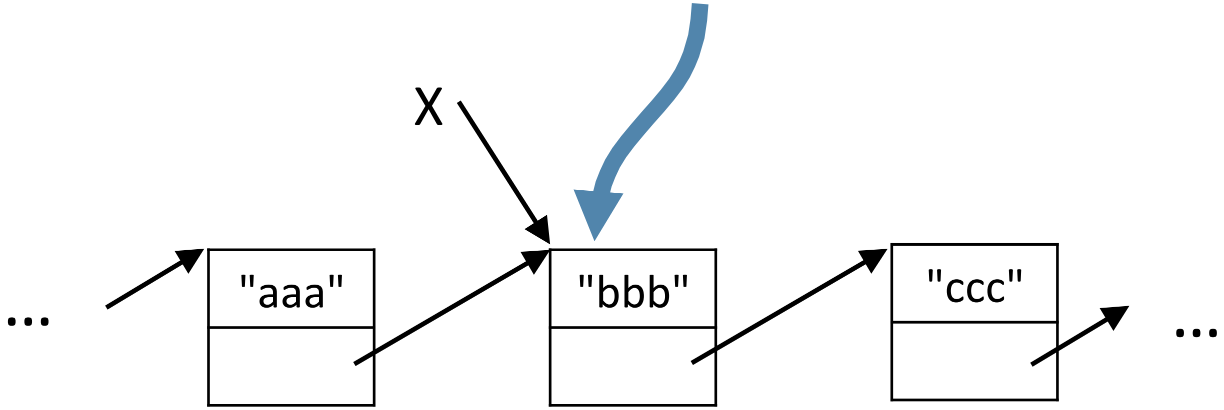
# Inserting a node

```
class LinkedList:

    # insert a node new at position n
    def  insert(self, new, n):
        if n == 0:
            self.add(new)
        else:
            prev = self.get_element(n–1)
            new.next = prev.next
            prev.next = new
```

# deletion

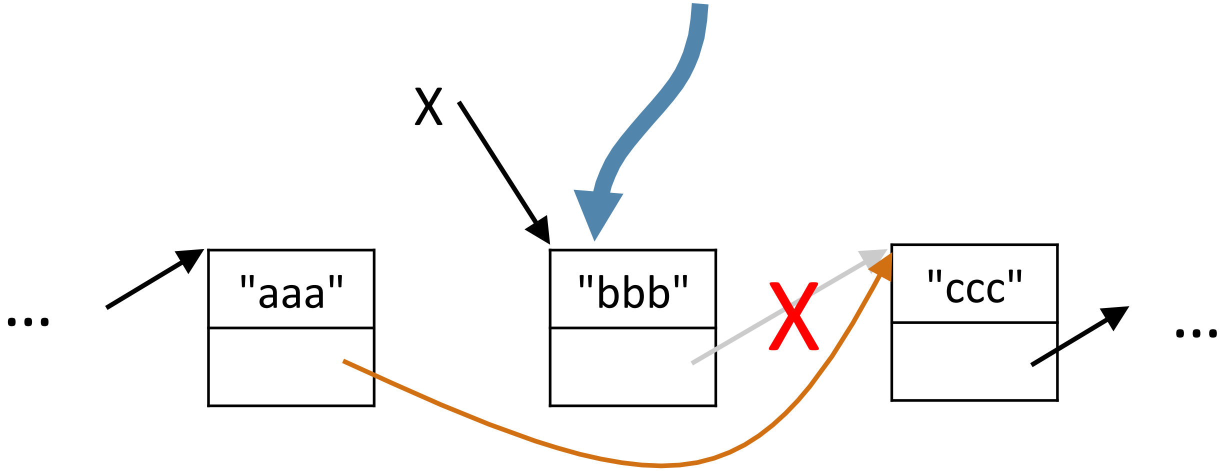
# Deleting a node

Suppose we want to delete this node:



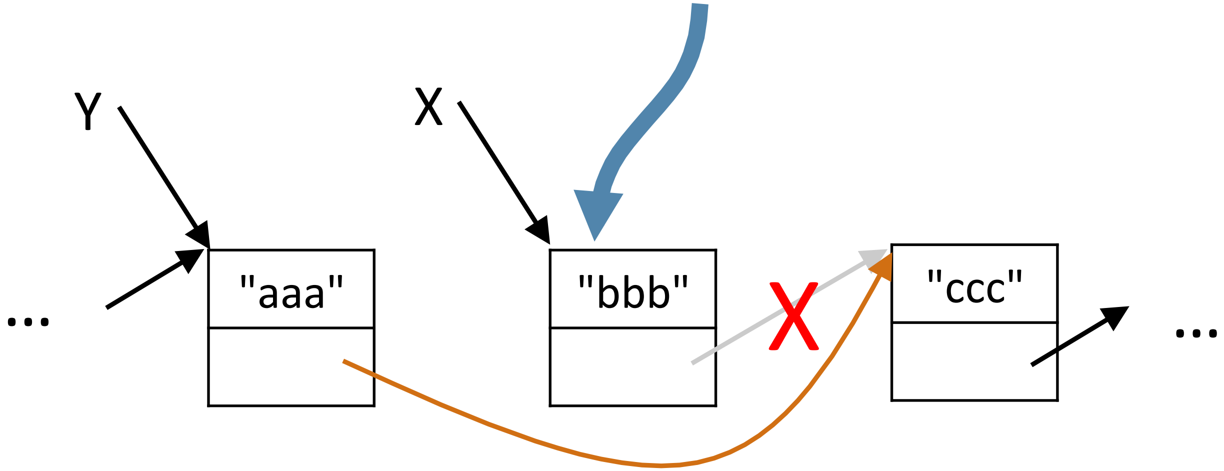
# Deleting a node

Suppose we want to delete this node:



# Deleting a node

Suppose we want to delete this node:



1. find the node Y just before X (i.e., Y._next == X)
2. Y._next = X._next
3. X._next = None

O(n)

O(1)

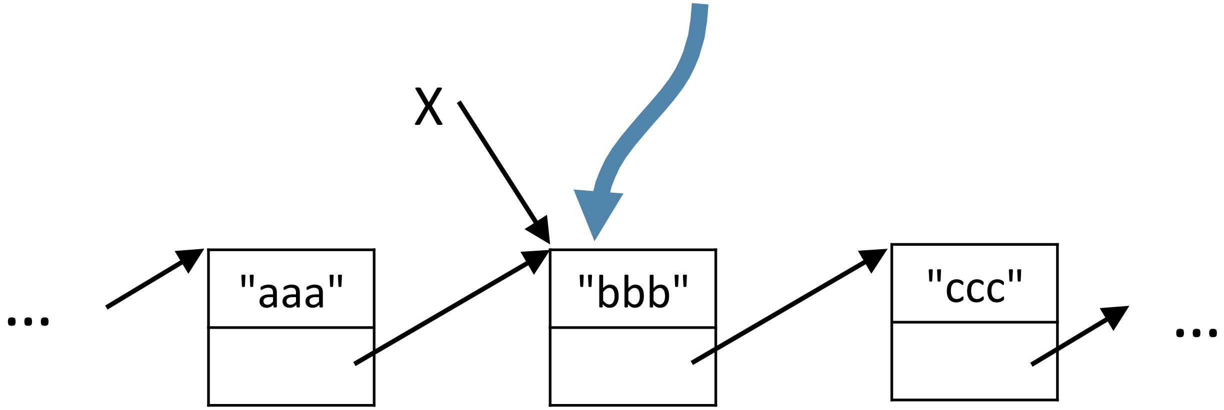
# Deleting a node

```
class LinkedList:
    # delete a node X
    def delete(self, X):
        if self._head == X:          # X is the head of the list
            self._head = X._next
        else:
            Y = self._head
            while Y._next != X:
                Y = Y._next
            Y._next = X._next
        X.next = None
```

# deletion (revisited)

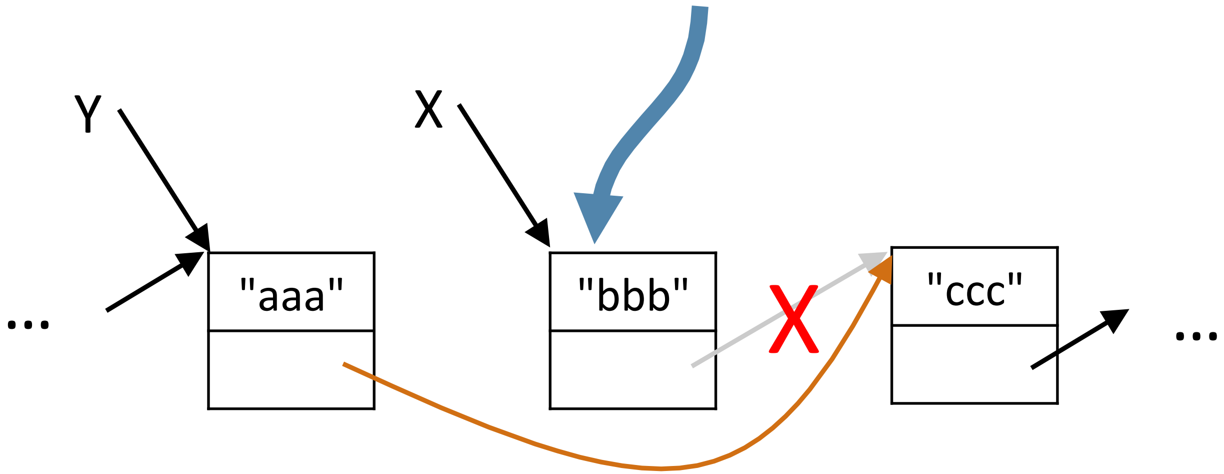
# Deleting a node

Suppose we want to delete this node:



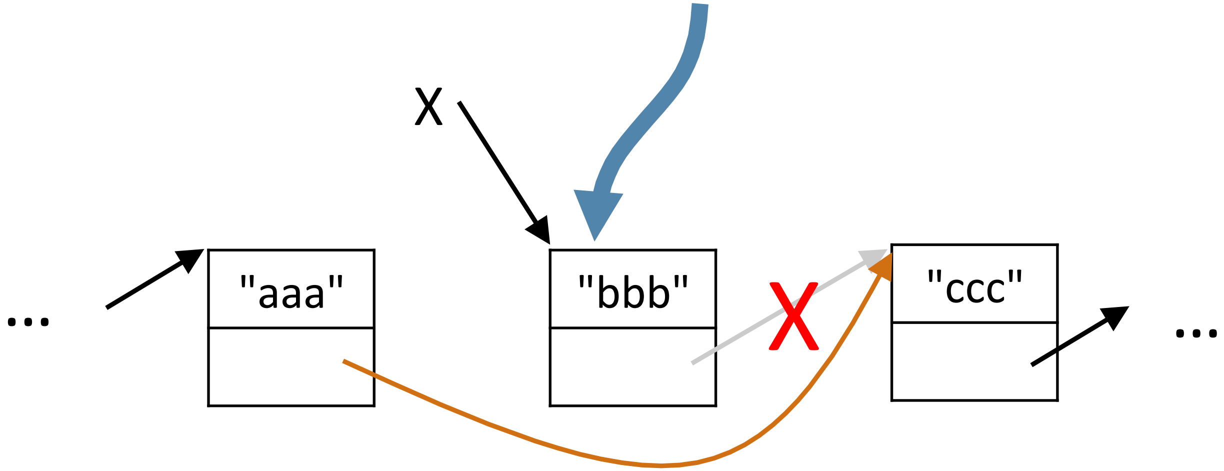
# Deleting a node

Suppose we want to delete this node:



# Deleting a node

Suppose we want to delete this node:

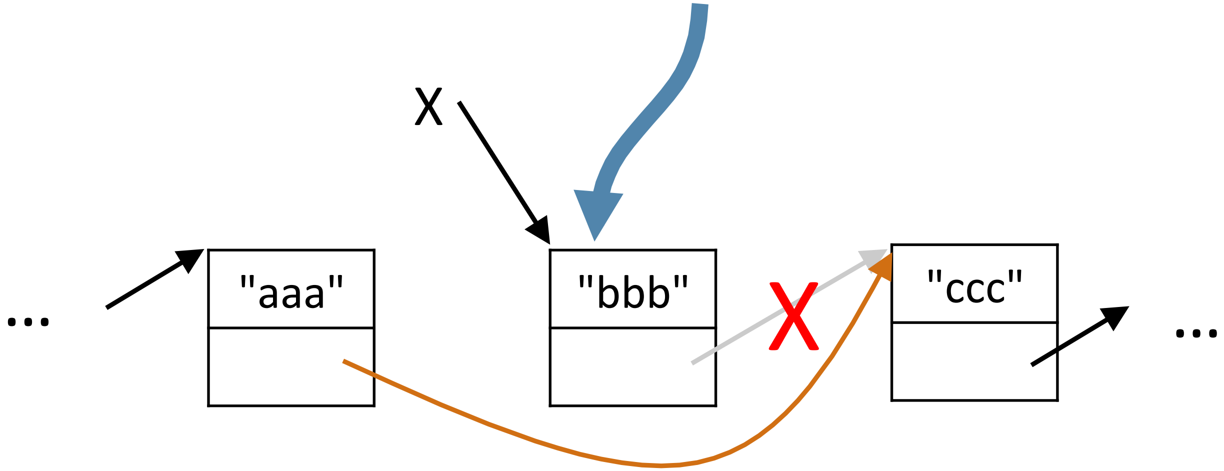


```
def  delete(self, x):
        r = self._head
         while r != None:
               if r == x:
                  <delete node x>
                   return
               r = r._next
```

- Does this code pattern work for delete?
- It worked for len, replace, count_vowels …

# Deleting a node

Suppose we want to delete this node:



```
def  delete(self, x):
        r = self._head
         while r != None:
             if r == x:
                 <delete node x>
                  return
             r = r._next
```

- No, does not work
- We need a reference to the previous node

# Deleting a node

```
class LinkedList:
    # delete a node X
    def delete(self, X):
        if self._head == X:            # X is the head of the list
            self._head = X._next
        else:
            Y = self._head
            while Y._next != X:
                Y = Y._next
            Y._next = X._next
        X.next = None
```

# concatenation

# Concatenating two linked lists

```
class LinkedList:
    # concatenate list2 at the end of the list
    def concat(self, list2):
        if self._head == None:      # list is empty
            self._head = list2._head
        else:
            tail = self._head
            while tail._next != None:
                tail = tail._next
            tail.next = list2._head
```

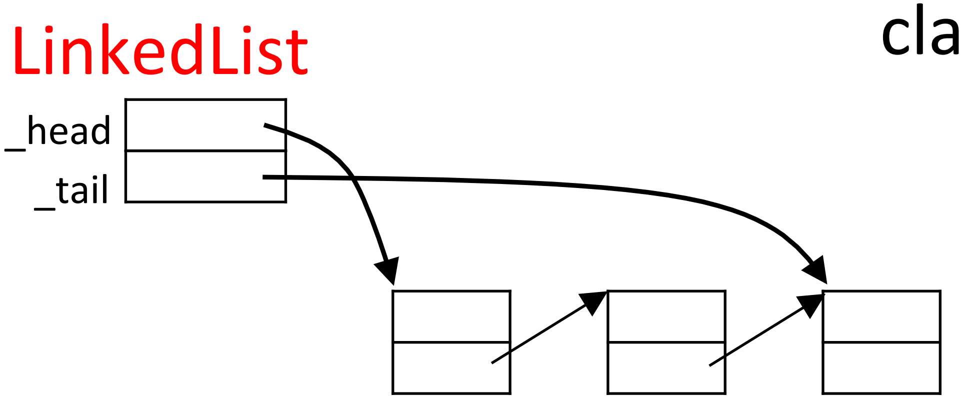
O(n)

O(1)

# maintaining a tail reference

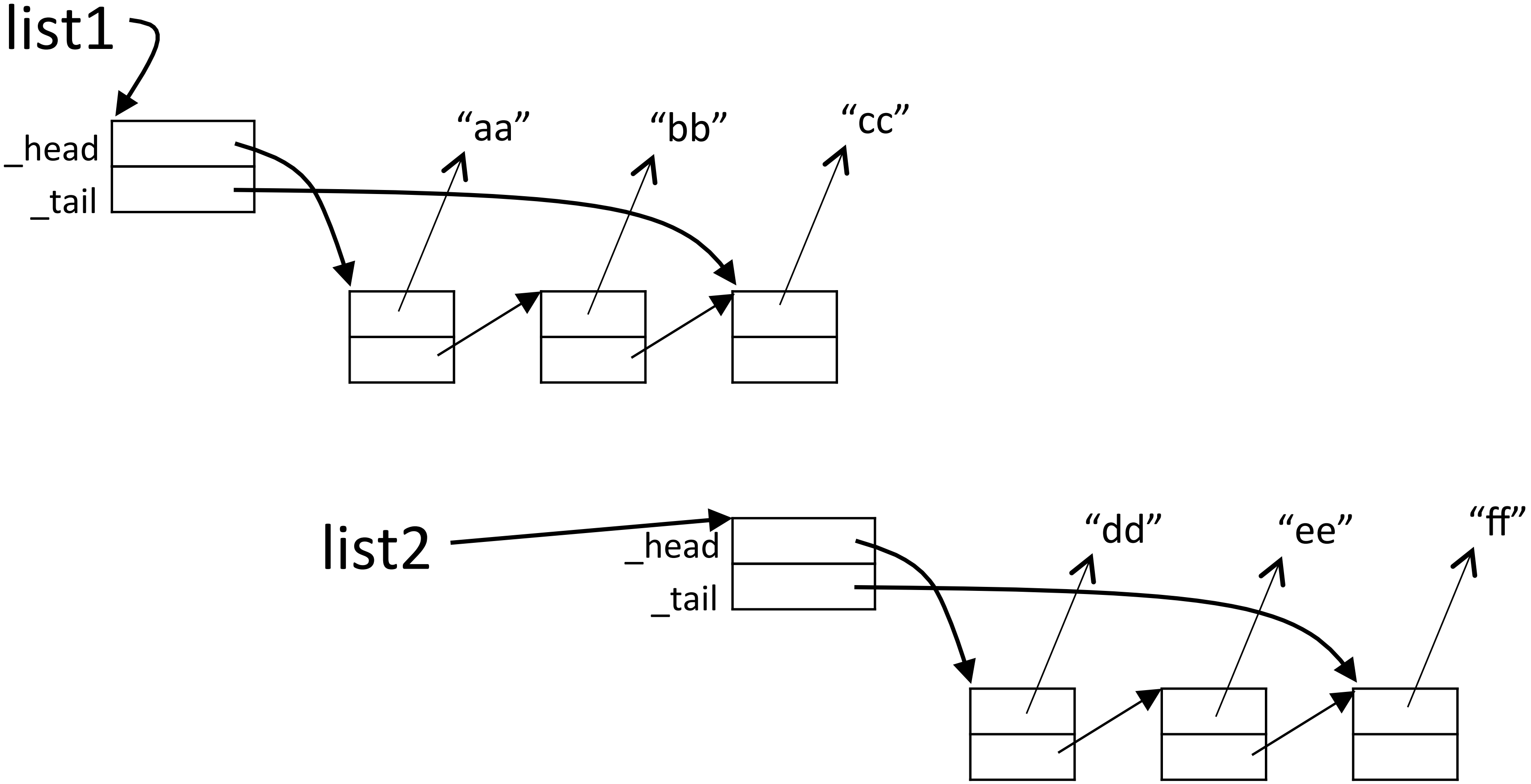
# Maintaining a tail reference

A variation is to also maintain a reference to the tail of the list

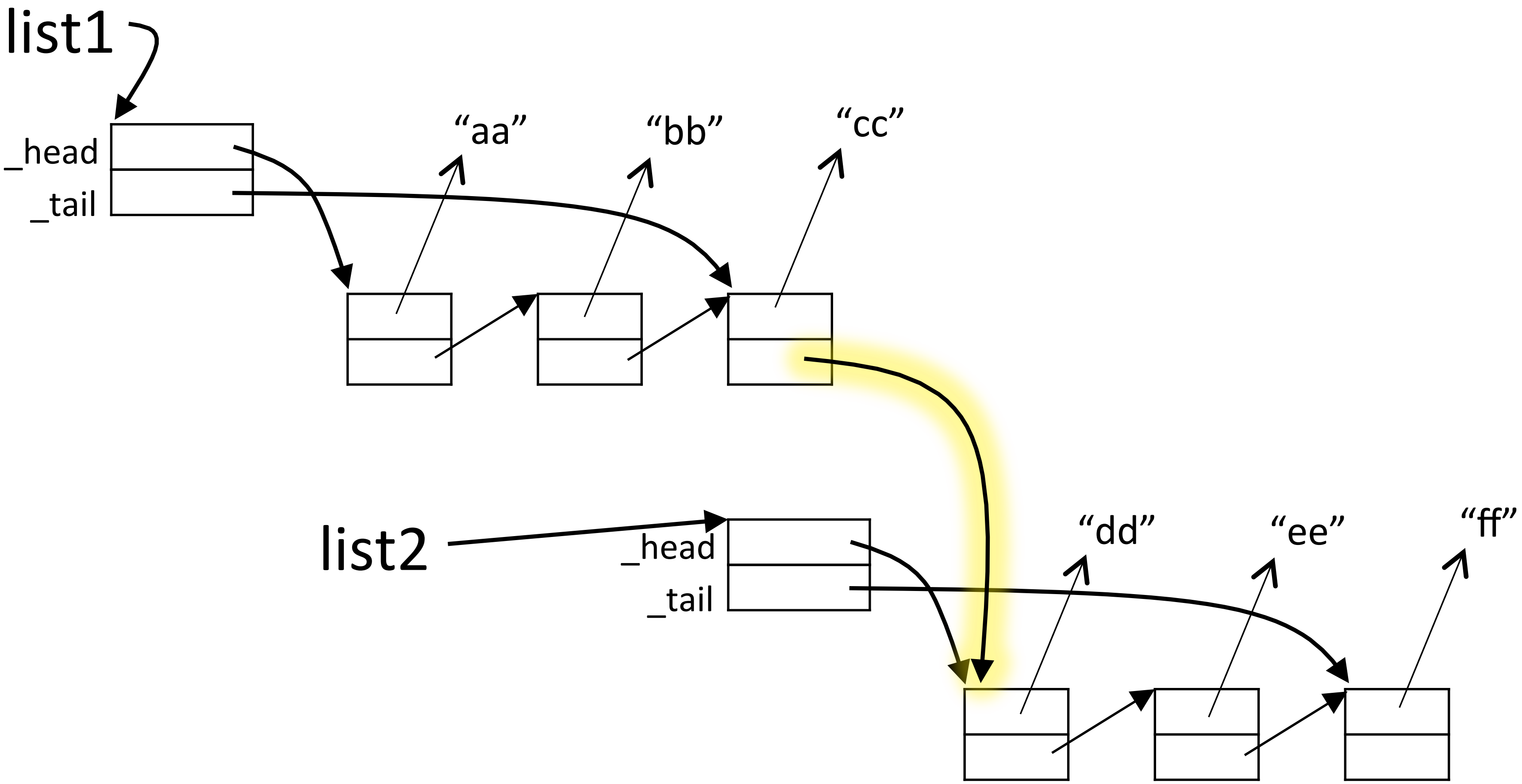


```
class LinkedList:
    def __init__(self):
        self._head = None
        self._tail = None
```

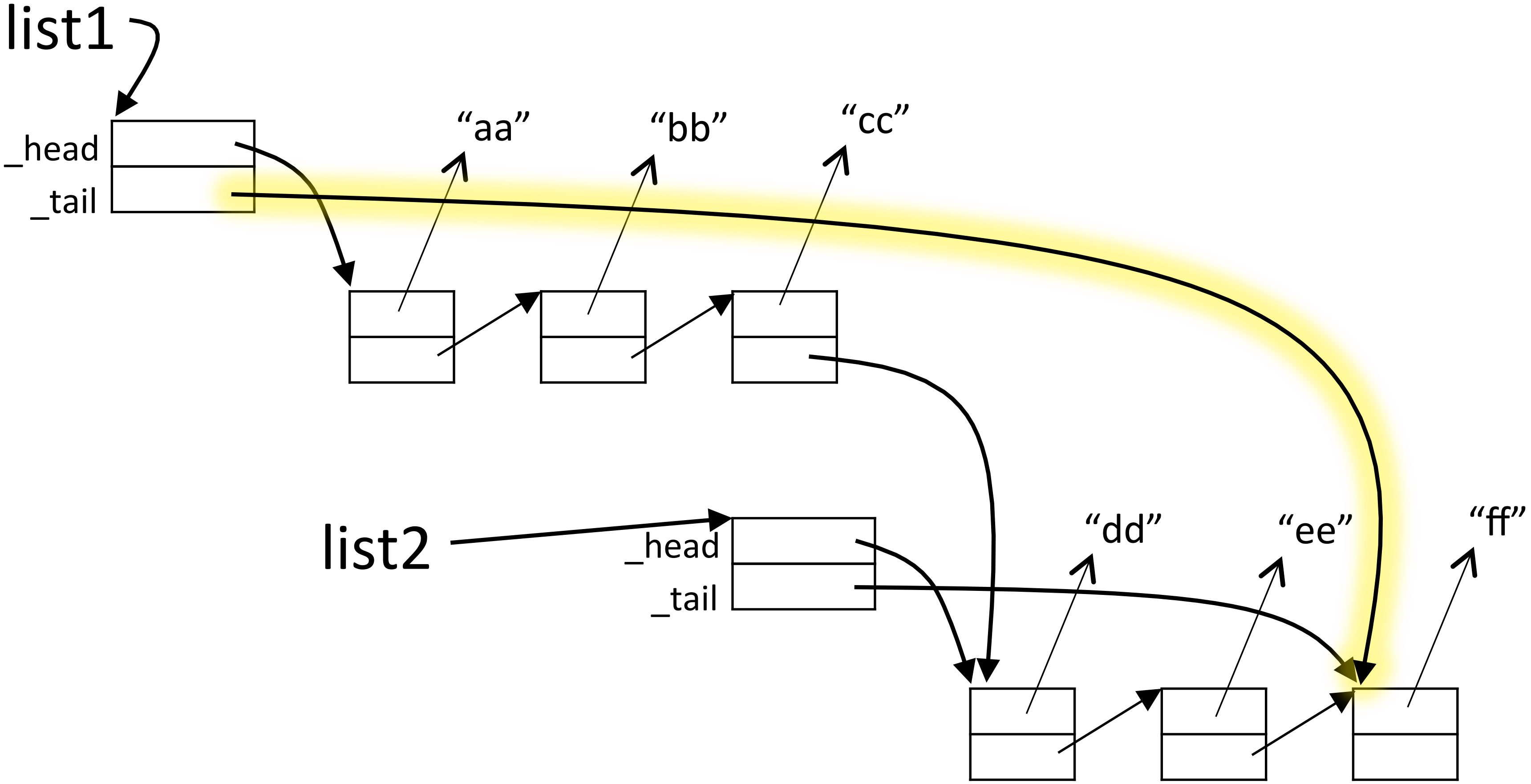
# Tail references and concatenation



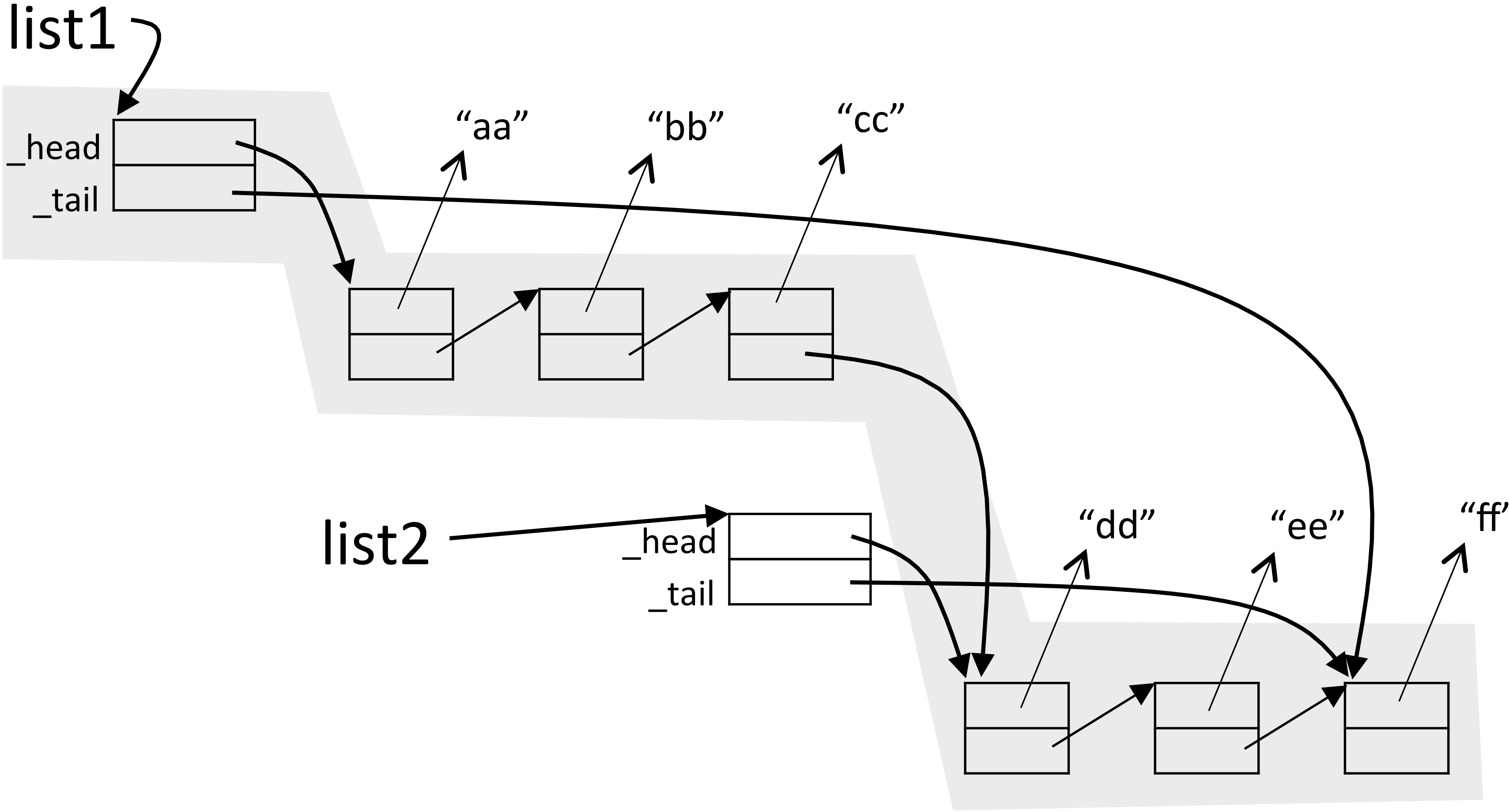
# Tail references and concatenation



# Tail references and concatenation



# Tail references and concatenation



# Maintaining a tail reference

- Concatenation and append become O(1):

```
def concat(self, list2):
    if self._head == None:
        self._head = list2._head
        self._tail = list2._tail
    else:
        self._tail._next = list2._head
        self._tail = list2._tail
```

- All linked list operations must now make sure that the tail reference is kept properly updated

# Linked lists: summary

| Operation | Without tail reference | With tail reference |
|---|---|---|
| add to front of list | O(1) | |
| append to end of list | O(n) | O(1) |
| find nth element | O(n) | |
| insert | O(1) if prev. node is available<br>O(n) otherwise | |
| delete | O(1) if prev. node is available<br>O(n) otherwise | |
| concatenate | O(n) | O(1) |