NPTEL MOOC

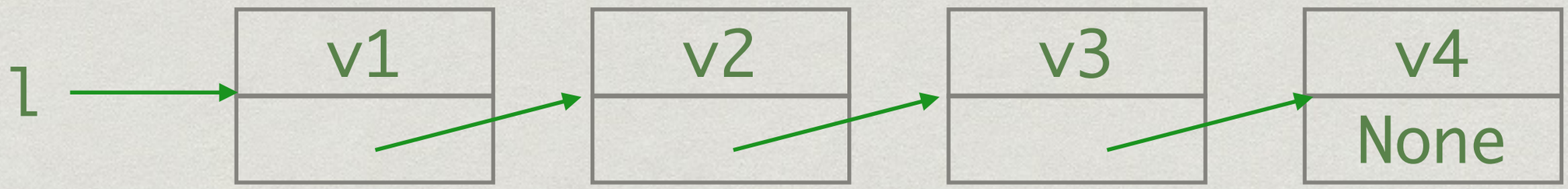
PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON

Week 7, Lecture 3

Madhavan Mukund, Chennai Mathematical Institute
http://www.cmi.ac.in/~madhavan

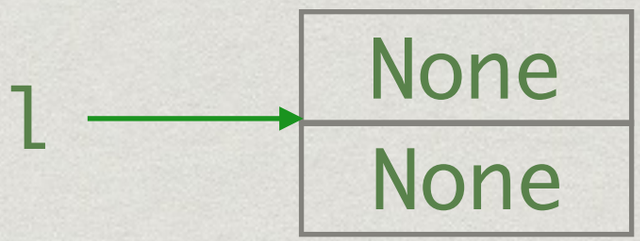
Designing our own list

- A list is a sequence of nodes
- Each node stores a value, points to next node



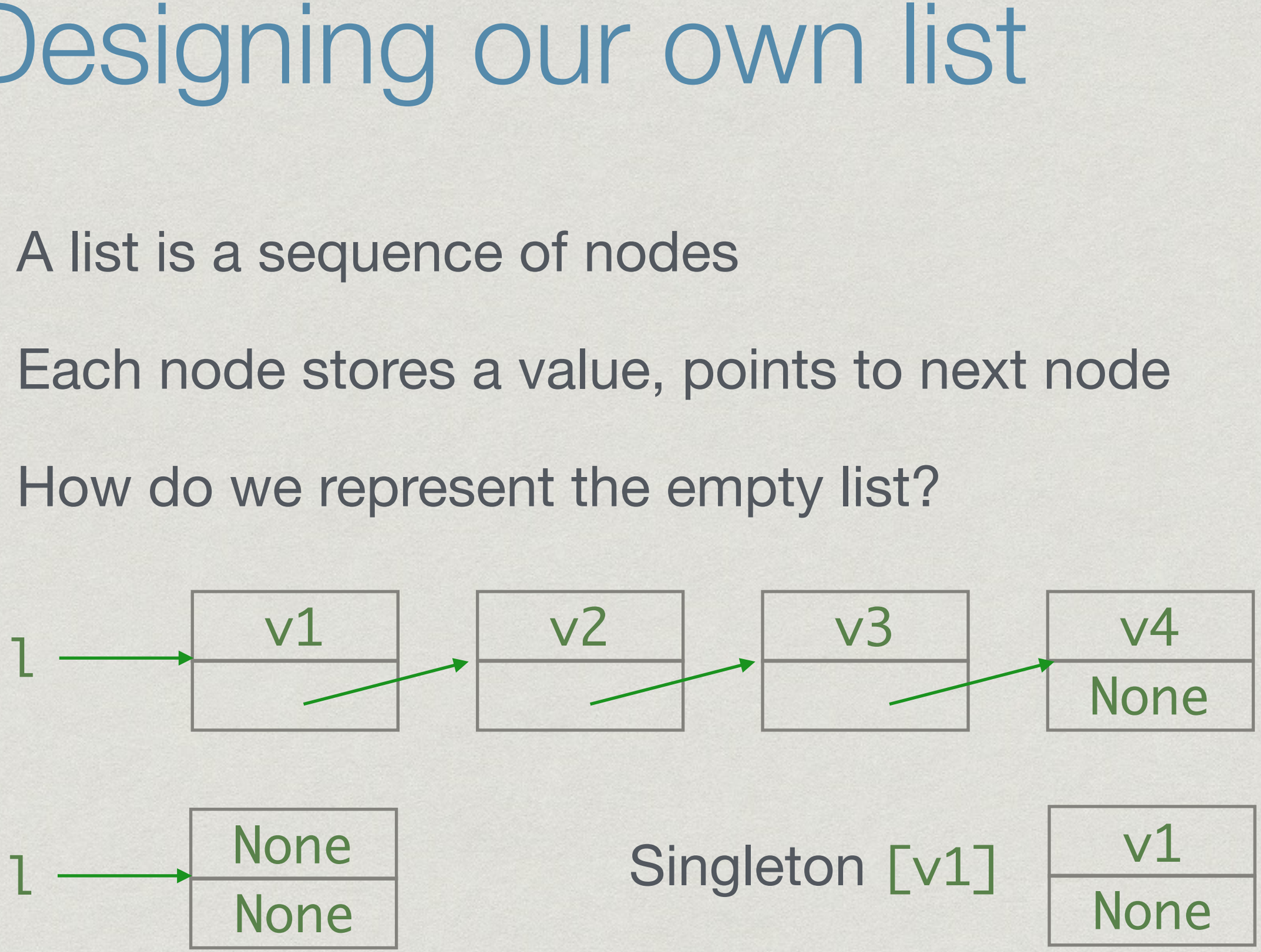
Designing our own list

- A list is a sequence of nodes
- Each node stores a value, points to next node
- How do we represent the empty list?



Designing our own list

- A list is a sequence of nodes
- Each node stores a value, points to next node
- How do we represent the empty list?



Class Node

```
class Node:

  def __init__(self,initval=None):
    self.value = initval
    self.next = None

  def isempty(self):
    return(self.value == None)
```

```
# Create empty list
l1 = Node()

# Create singleton
l2 = Node(5)
```

Class Node

```
class Node:
    def __init__(self,initval=None):
      self.value = initval
      self.next = None

    def isempty(self):
      return(self.value == None)
```

```
# Create empty list
l1 = Node()

# Create singleton
l2 = Node(5)

l1.isempty()==True
l2.isempty()==False
```

Append a value **v**

- If list is empty, replace `None` by `v`
- If at last element of list (`next` is `None`)
  - Create a node with value `v`
  - Set `next` to point to new node
- Otherwise, recursively append to rest of the list

Append a value v

```
def append(self,v):
   if self.isempty():
      self.value = v
   elif self.next == None:
      newnode = Node(v)
      self.next = newnode
   else:
      (self.next).append(v)
   return
```

Append a value v

```
def append(self,v):
   if self.isempty():
      self.value = v
   elif self.next == None:
      newnode = Node(v)
      self.next = newnode
   else:
      self.next.append(v)
   return
```

Append a value iteratively

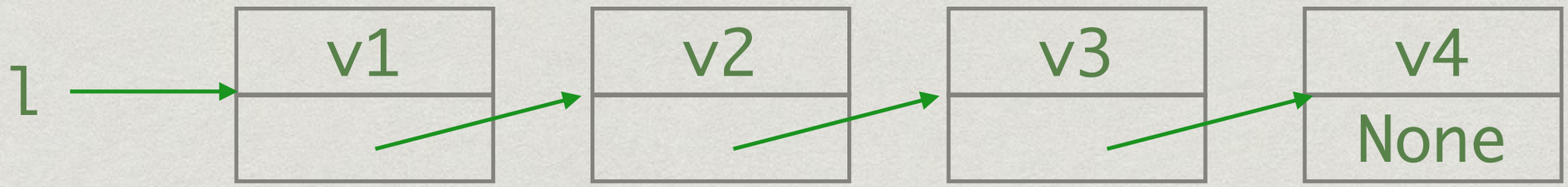
- If list is empty, replace None by v
- Scan the list till we reach the last element
- Append the element at the last element

Append value iteratively

```
def appendi(self,v):
    if self.isempty():
        self.value = v
        return

    temp = self
    while temp.next != None:
        temp = temp.next

    newnode = Node(v)
    temp.next = newnode

    return
```

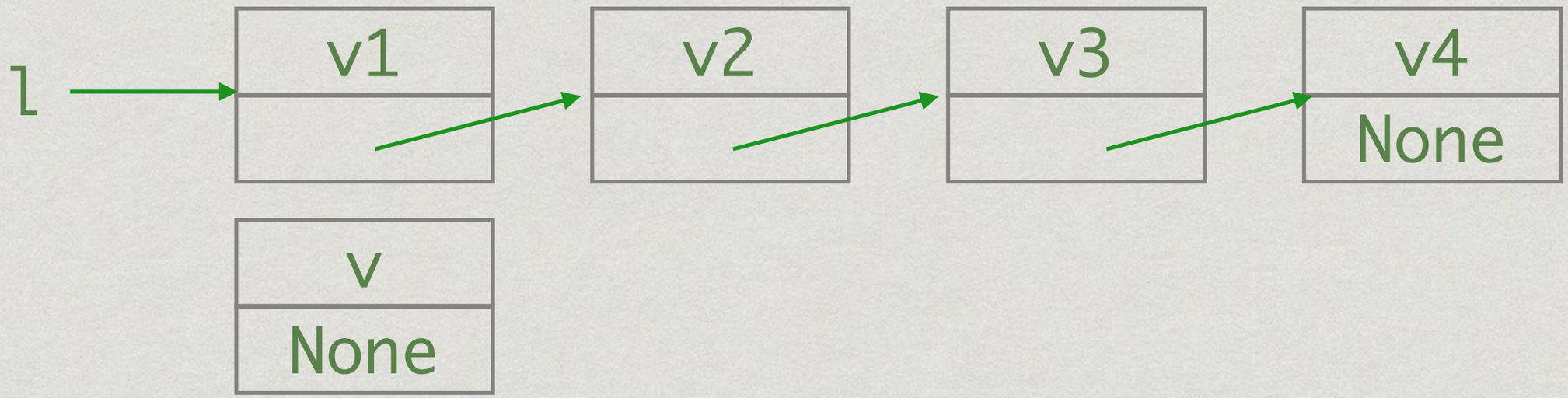
Insert a value v

- Want to insert v at the head of the list



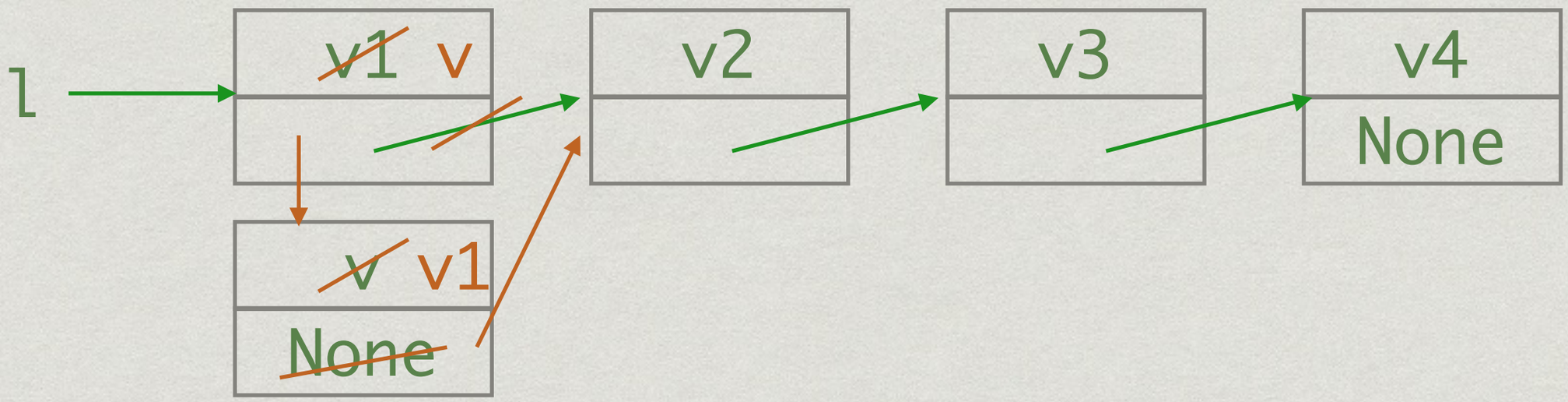
Insert a value v

- Want to insert v at the head of the list
- Create a new node with v
  - But we cannot change where `l` points to!



Insert a value v

- Want to insert v at the head of the list
- Create a new node with v
  - But we cannot change where `l` points to!
- Instead, swap the contents of v with the current first node

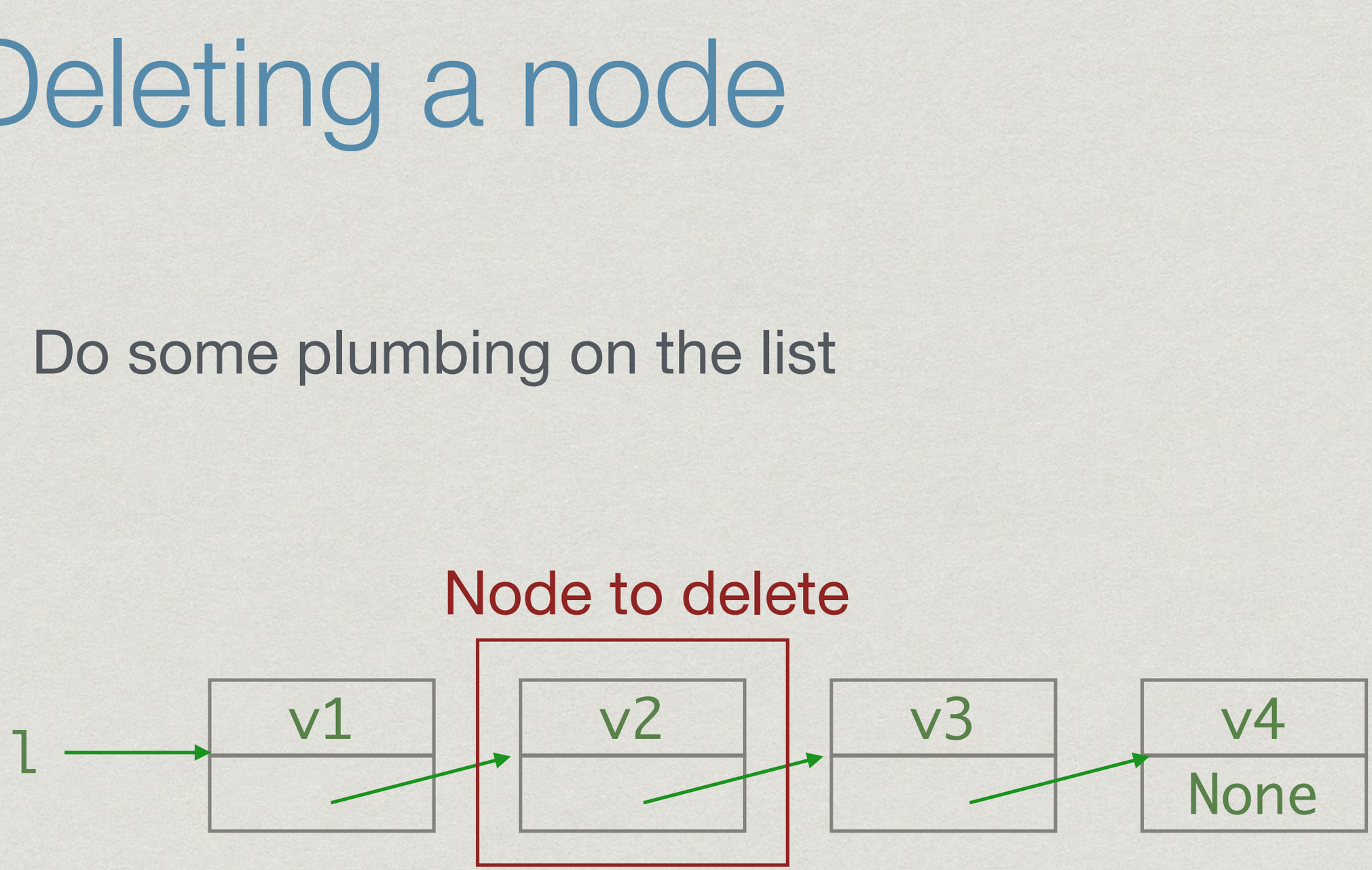


Insert a value v

```
def insert(self,v):

  if self.isempty():
    self.value = v
    return

  newnode = Node(v)

  # Exchange values in self and newnode
  (self.value, newnode.value) =
                          (newnode.value, self.value)
  (self.next, newnode.next) = (newnode, self.next)

  return
```

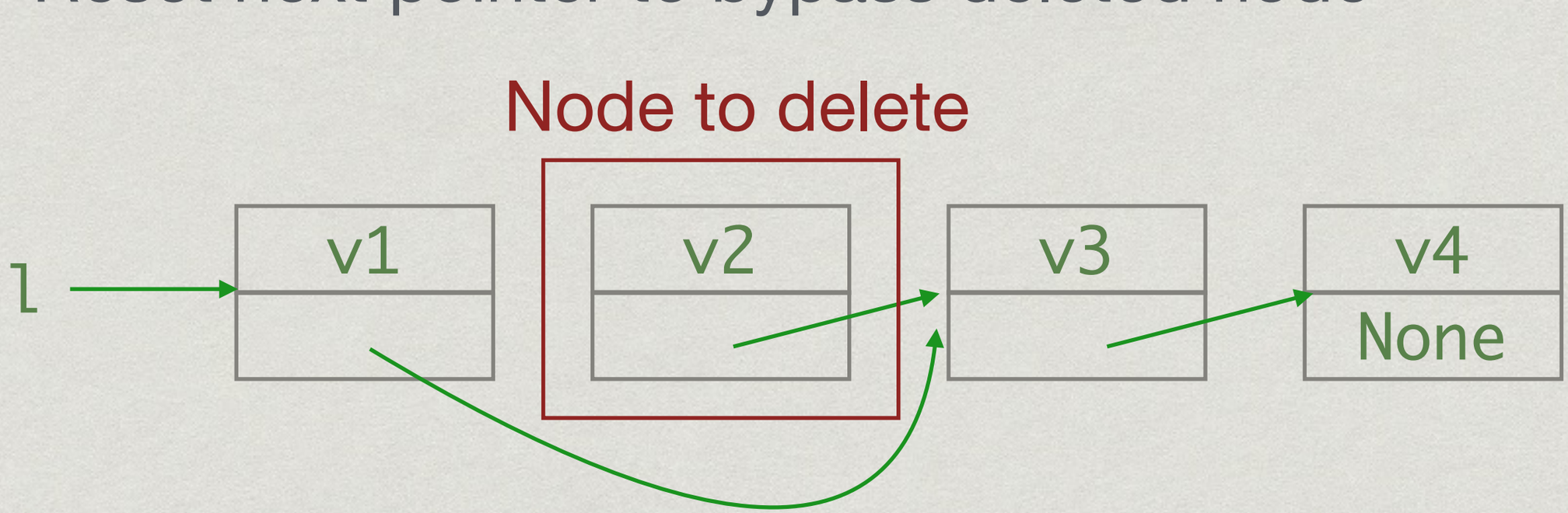
Deleting a node

- Do some plumbing on the list



Deleting a node

- Do some plumbing on the list
- Reset next pointer to bypass deleted node

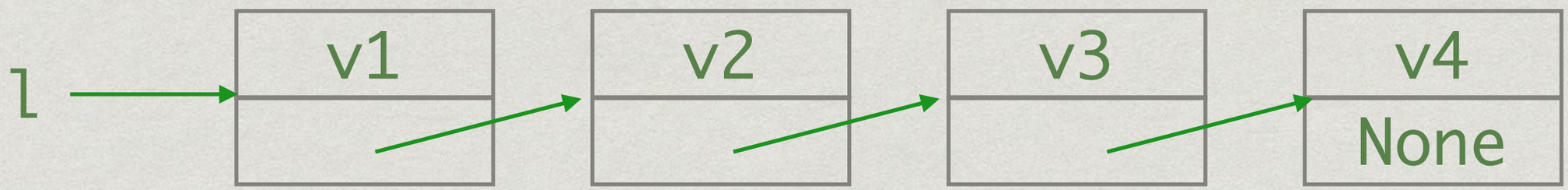


Delete a value v

- Remove first occurrence of v
- Scan list for first v
- If `self.next.value == v`, bypass `self.next`
  - `self.next = self.next.next`
- What if first value in the list is v?

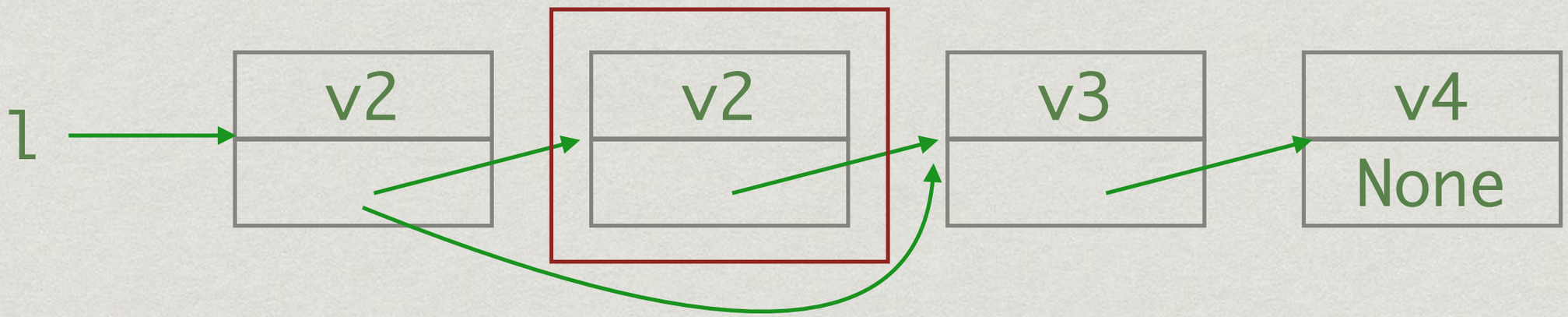
Deleting first value in list

- `l.delete(v1)`
- Cannot delete the node that `l` points to
- Reassigning name in function creates a new object



Deleting first value in list

- `l.delete(v1)`
- Cannot delete the node that `l` points to
  - Reassigning name in function creates a new object
- Instead, copy v2 from next node and delete second node!



Delete a value v

```
def delete(self,v):
  if self.isempty():
    return

  if self.value == v: # value to delete
                      # is in first node
    if self.next == None
      self.value = None
    else:
      self.value = self.next.value
      self.next = self.next.next
      return
```

Delete a value v

```
def delete(self,v):
  if self.isempty():
    return

  if self.value == v: # value to delete
                      # is in first node
    . . .

  temp = self # find first v to delete
  while temp.next != None:
    if temp.next.value == v:
      temp.next = temp.next.next
      return
    else:
      temp = temp.next

  return
```

Delete a value v

```
def delete(self,v):
  if self.isempty():
    return

  if self.value == v: # value to delete is in first node
    if self.next == None
      self.value = None
    else:
      self.value = self.next.value
      self.next = self.next.next
      return

  temp = self # first v to delete
  while temp.next != None:
    if temp.next.value == v:
      temp.next = temp.next.next
      return
    else:
      temp = temp.next

  return
```

Delete value v, recursively

- If v occurs in first node, delete as before
- Otherwise, if there is a next node, recursively delete v from there
  - If `next.value == v` and `next.next == None`, `next.value` becomes `None`
  - If so, terminate the list here

Delete value v, recursively

```
def deleter(self,v):
  if self.isempty():
    return

  if self.value == v: # value to delete is in first node
    if self.next == None
      self.value = None
    else:
      self.value = self.next.value
      self.next = self.next.next
      return

  else: # recursive delete
    if self.next != None:
      self.next.deleter(v)
      if self.next.value == None:
        self.next = self.next.next

  return
```

Printing out the list

```
def __str__(self):
  selflist = []

  if self.value == None:
    return(str(selflist))

  temp = self
  selflist.append(temp.value)

  while temp.next != None:
    temp = temp.next
    selflist.append(temp.value)

  return(str(selflist))
```