Text

0	1	2	3	4	5	6	7	8

L.insert(0,-1)

	0	1	2	3	4	5	6	7	8

L.insert(0,-1)

-1	0	1	2	3	4	5	6	7	8

L.insert(0,-1)

-1	0	1	2	3	4	5	6	7	8

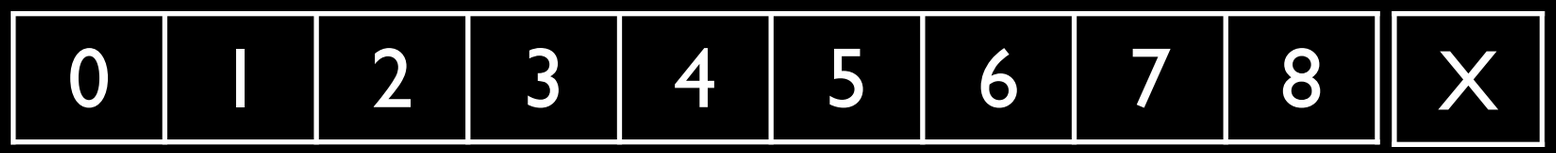
L.insert(0,-1)

linear

L.append(9)

L.append(9)

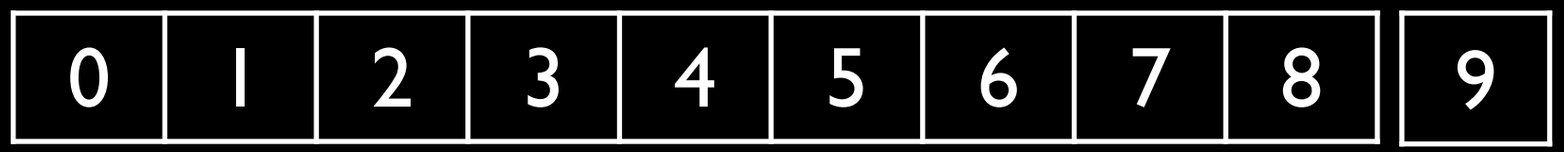
?



L.append(9)

?

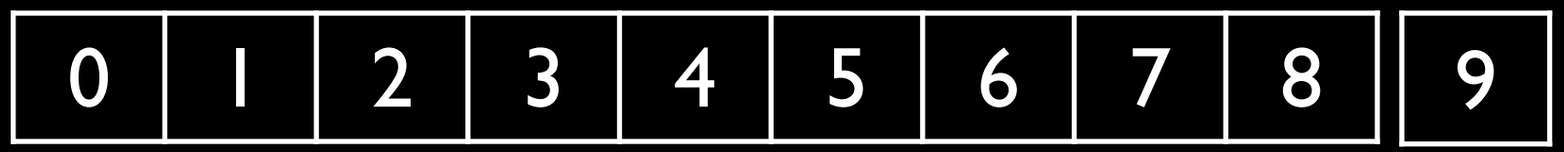
find new memory and copy old contents



L.append(9)

?

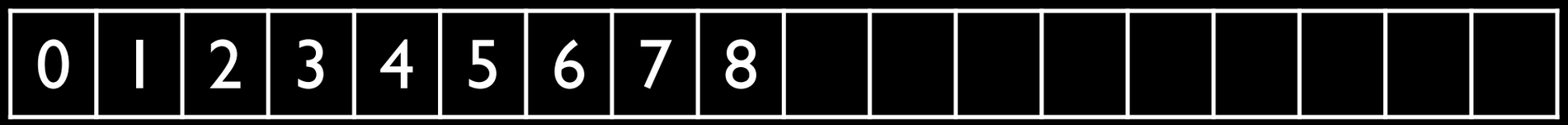
find new memory and copy old contents



L.append(9)

linear

allocate twice as much
memory as requested



L.append(9)

constant, but sometimes *linear*

depend on memory

del L[0]

1	2	3	4	5	6	7	8	

del L[0]

del L[0]

?

del L[0]

linear

0	1	2	3	4	5	6	7	8

del L[8]

del L[8]

?

del L[8]

constant

Lists

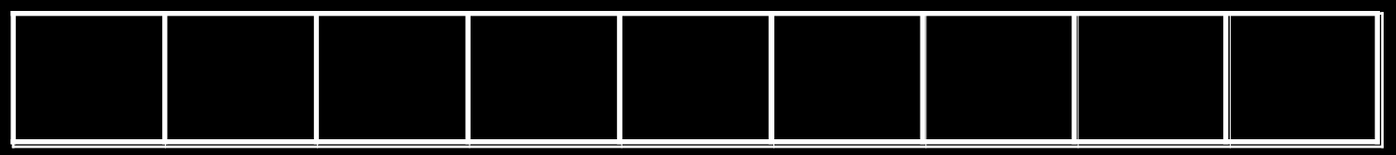
insert: *linear*

append: *constant*, but sometimes *linear*

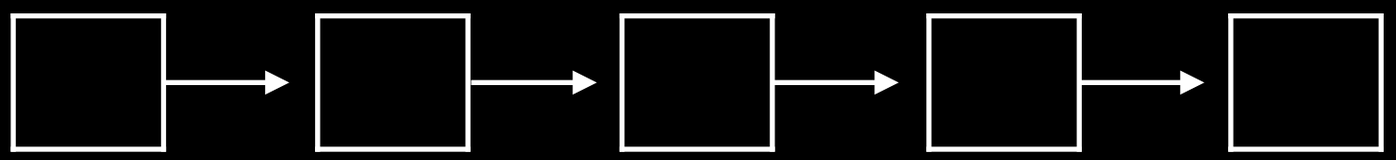
del: *linear*

access: *constant*

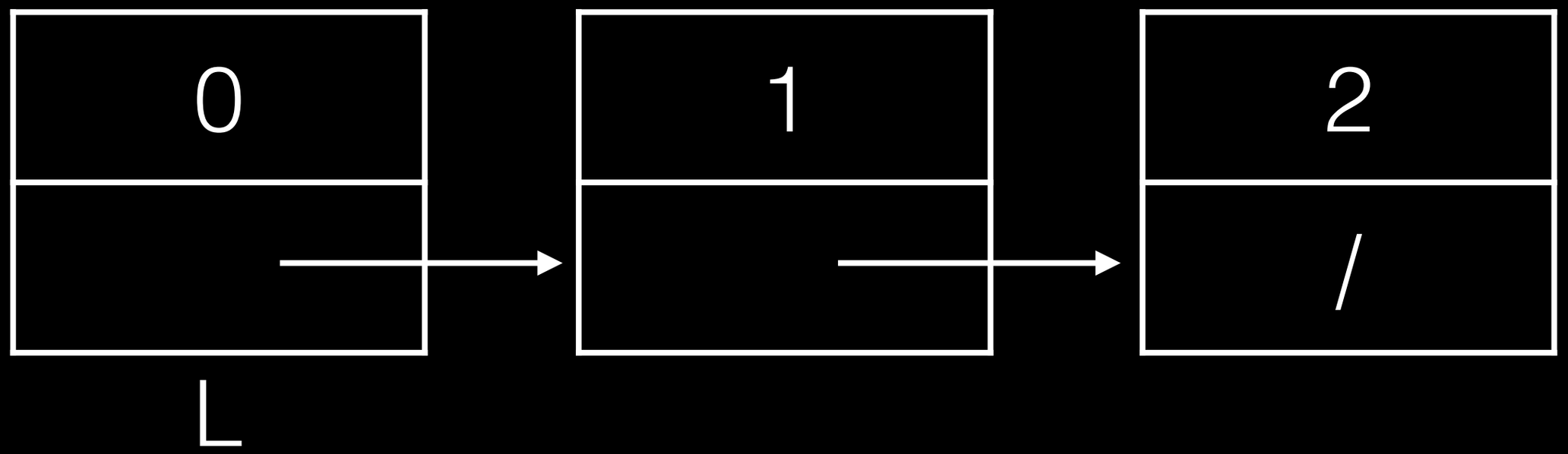
Lists
(bus)



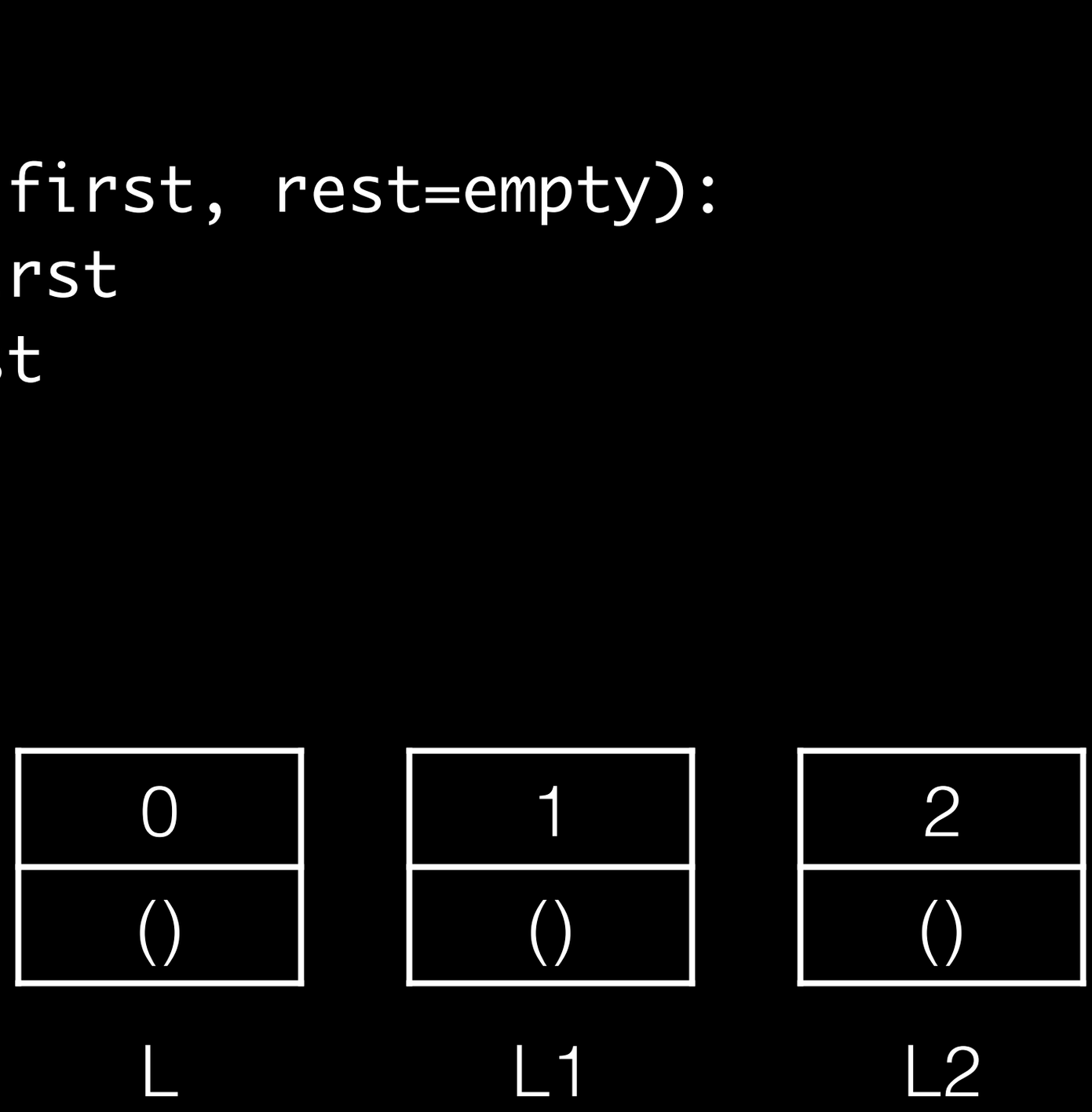
Linked Lists
(train)



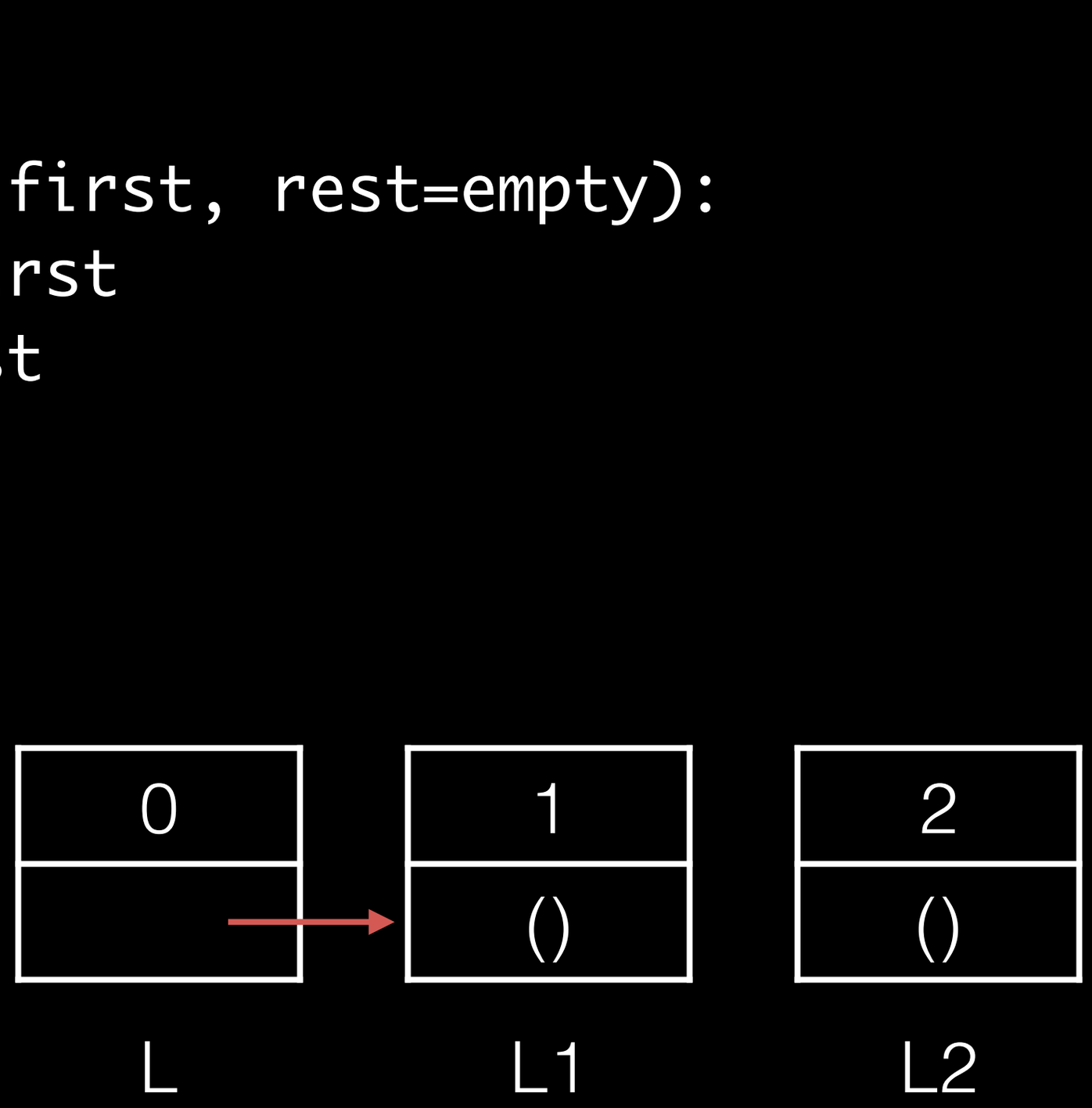
Linked Lists



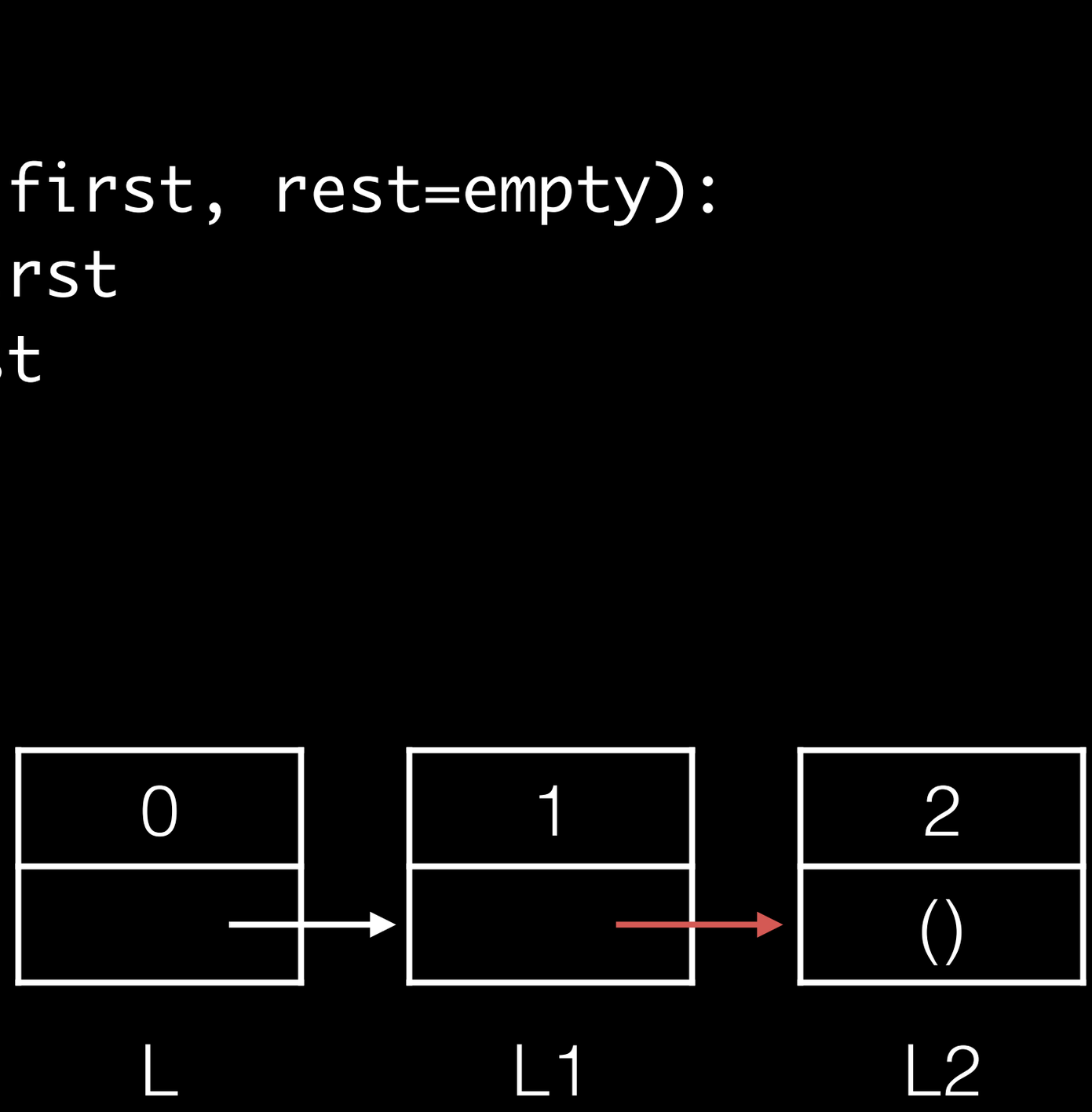
```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

L  = Link(0)
L1 = Link(1)
L2 = Link(2)

# connect nodes?
```



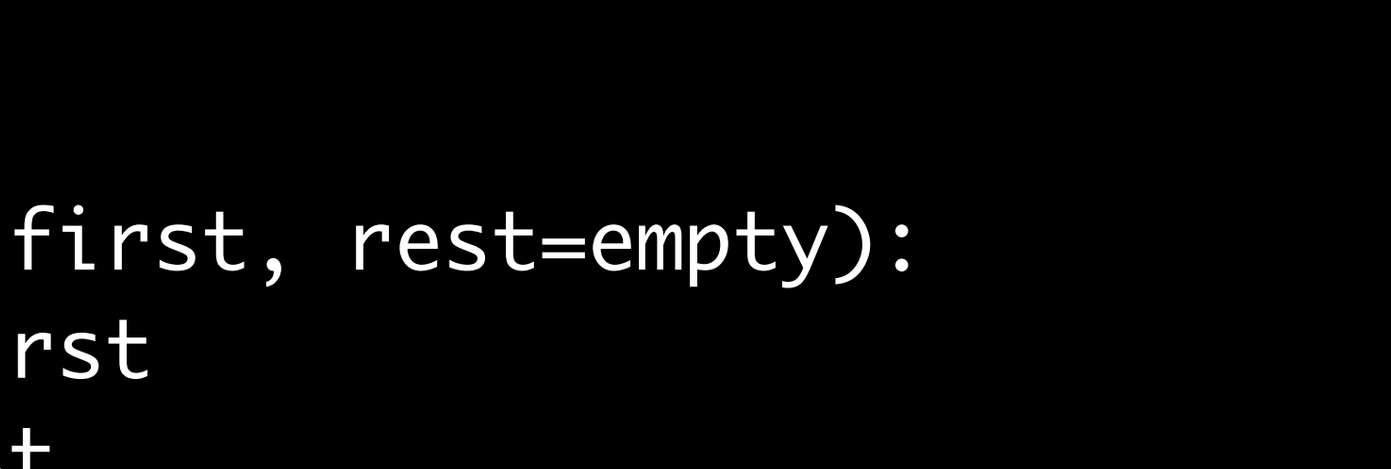
```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

L  = Link(0)
L1 = Link(1)
L2 = Link(2)

L.rest  = L1
```



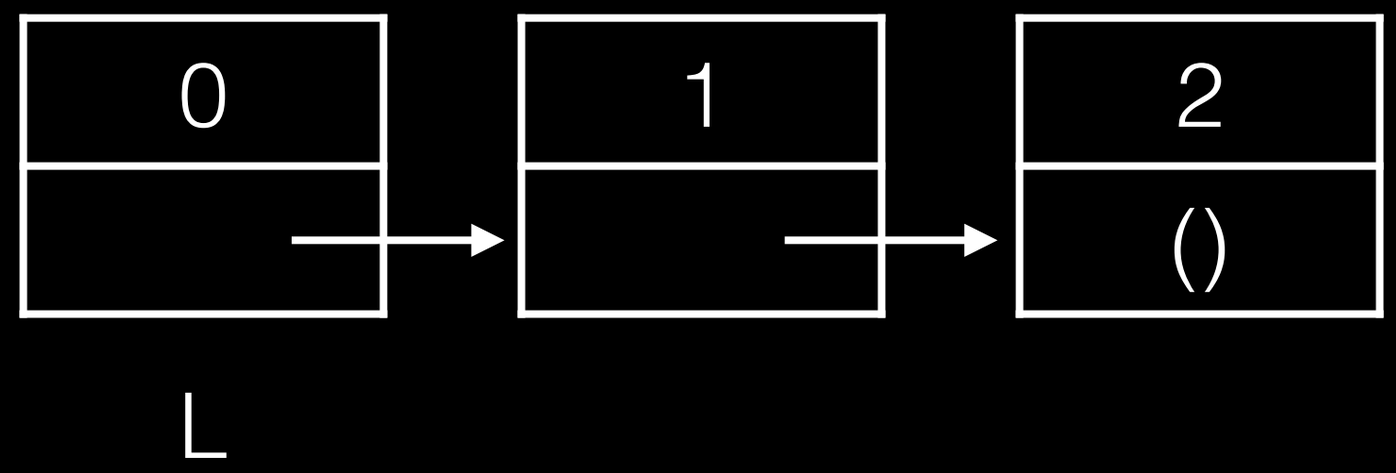
```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

L  = Link(0)
L1 = Link(1)
L2 = Link(2)

L.rest  = L1
L1.rest = L2
```



```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

L  = Link(0)
L1 = Link(1)
L2 = Link(2)

L.rest  = L1
L1.rest = L2
```

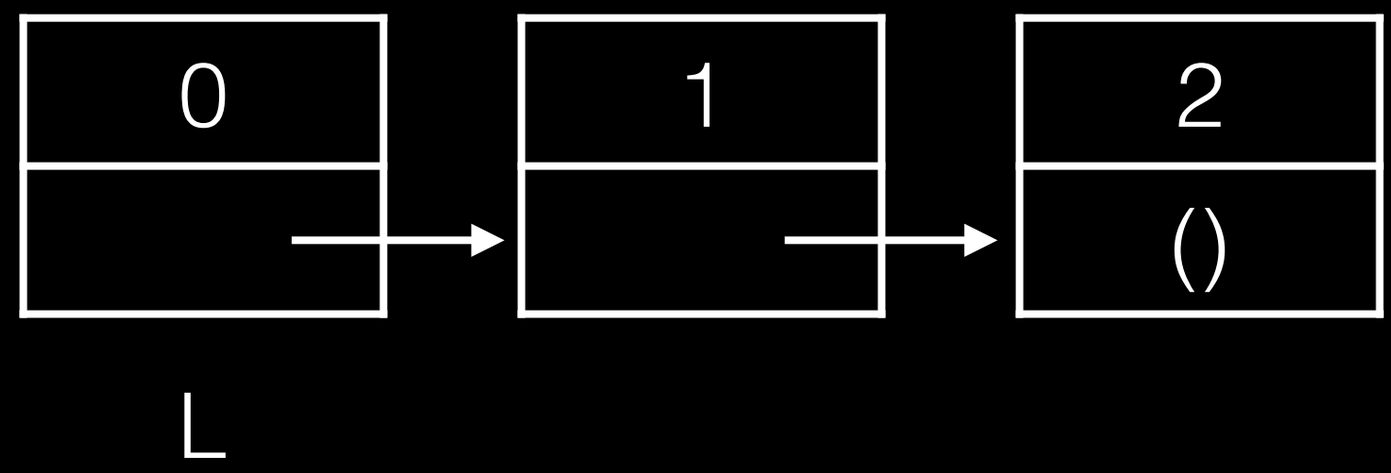


```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

L  = Link(0, Link(1, Link(2)))
```



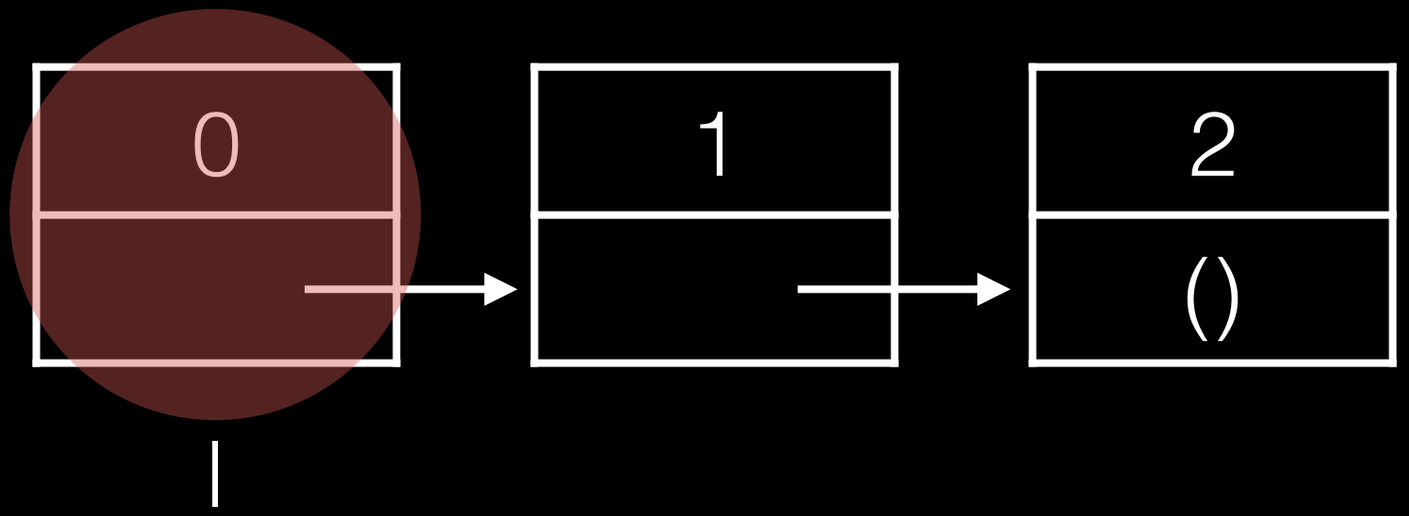
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
```



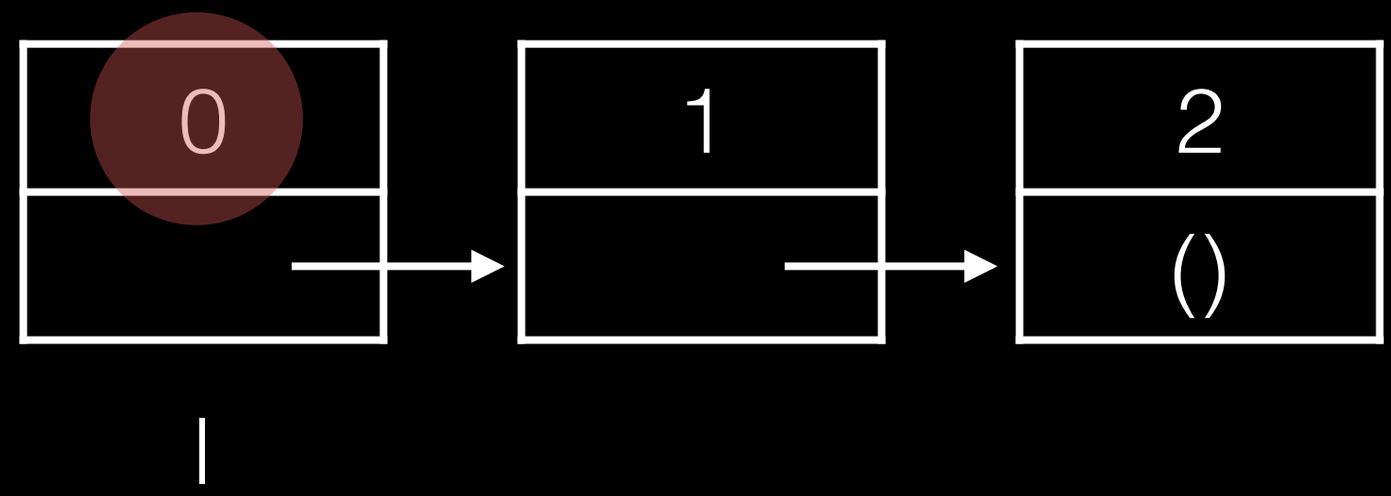
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
```



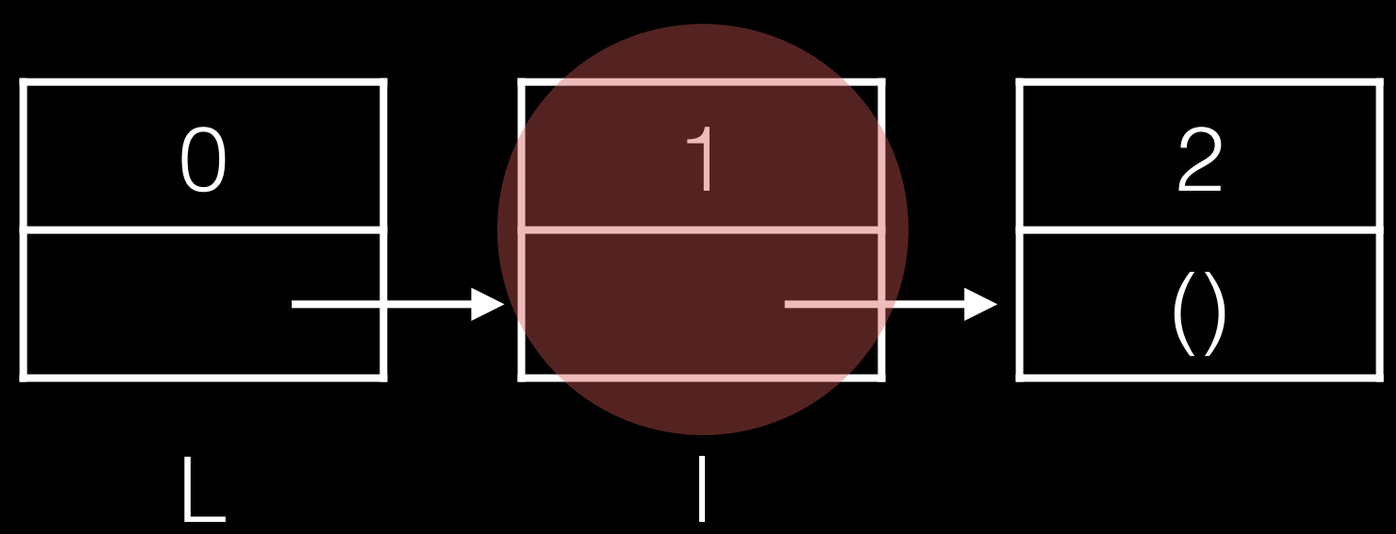
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
```



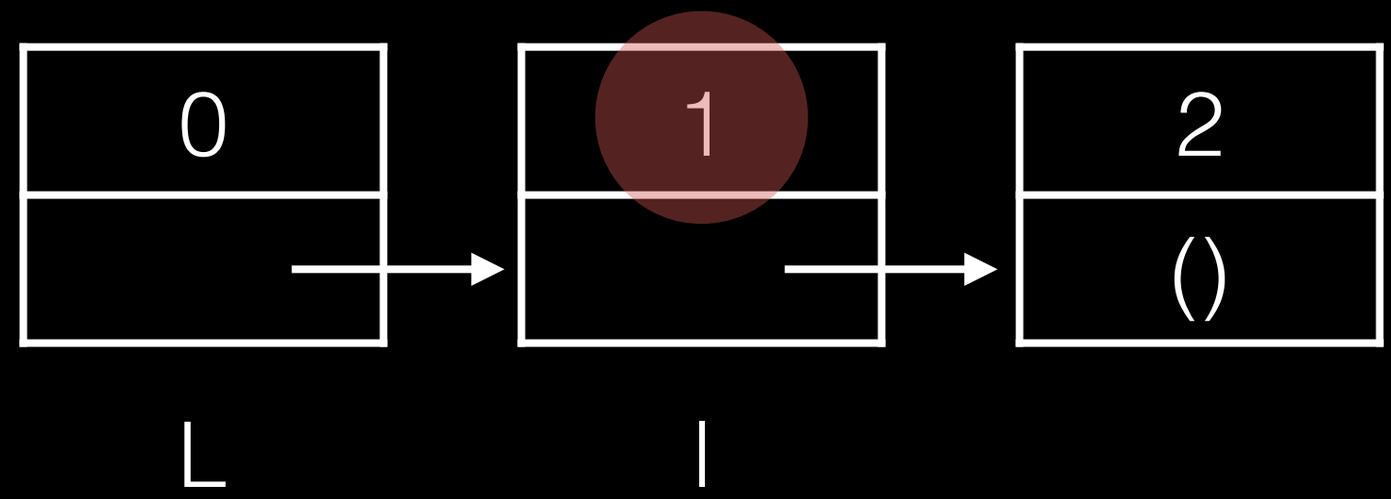
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
    l = l.rest
```



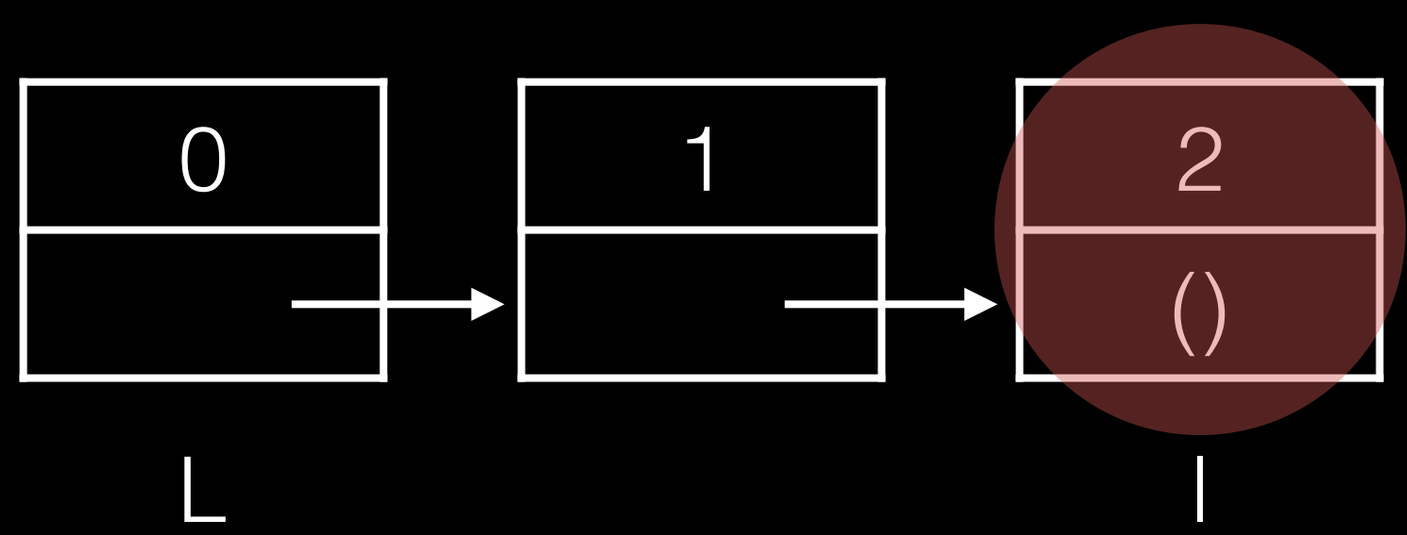
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
    l = l.rest
```



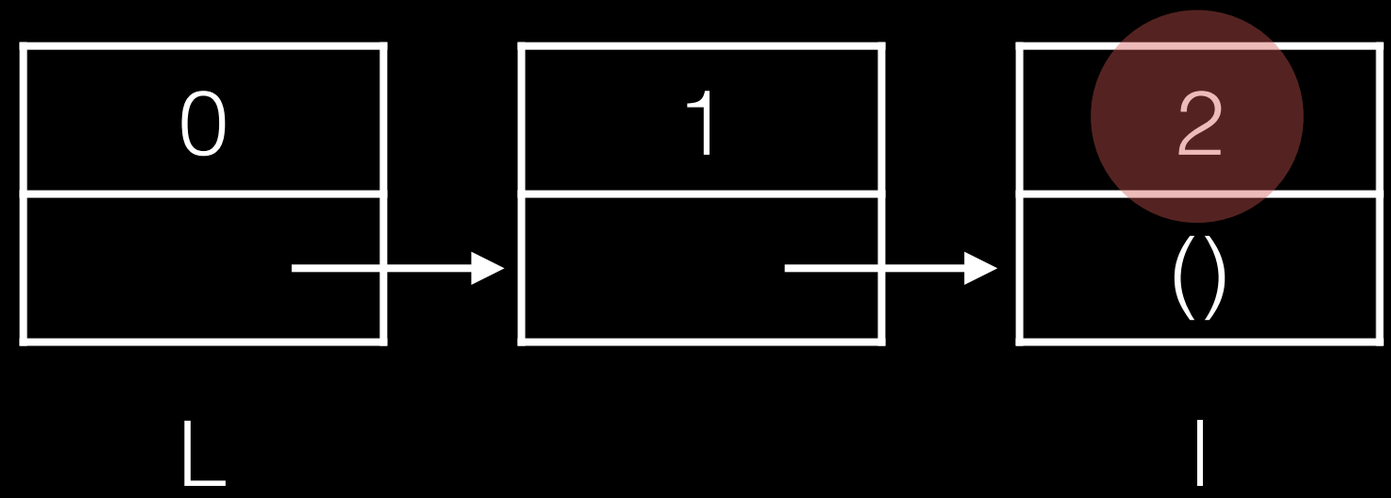
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
    l = l.rest
```



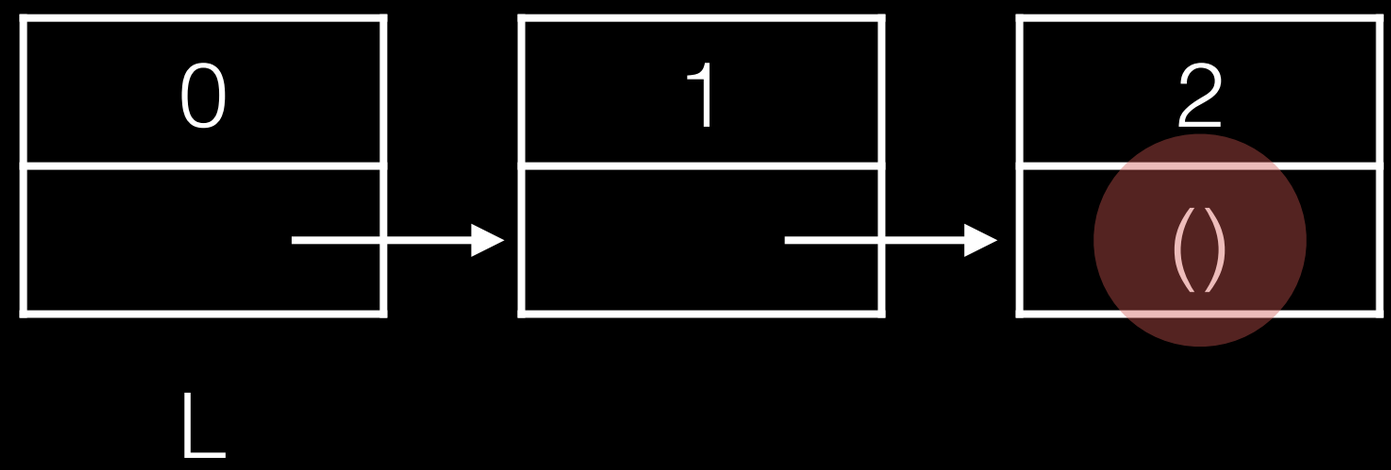
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
    l = l.rest
```



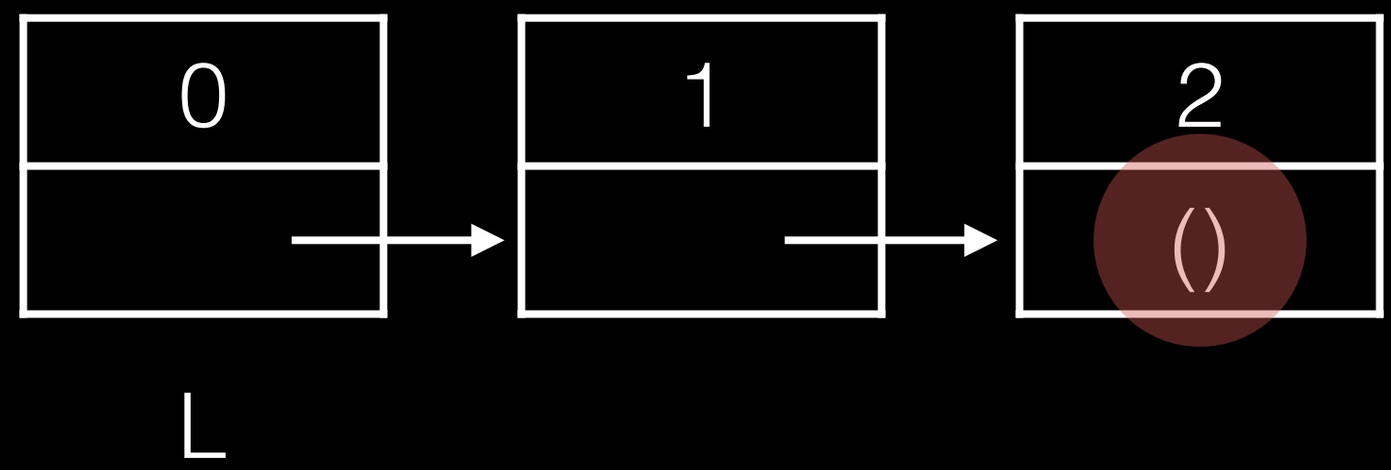
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
    l = l.rest
```



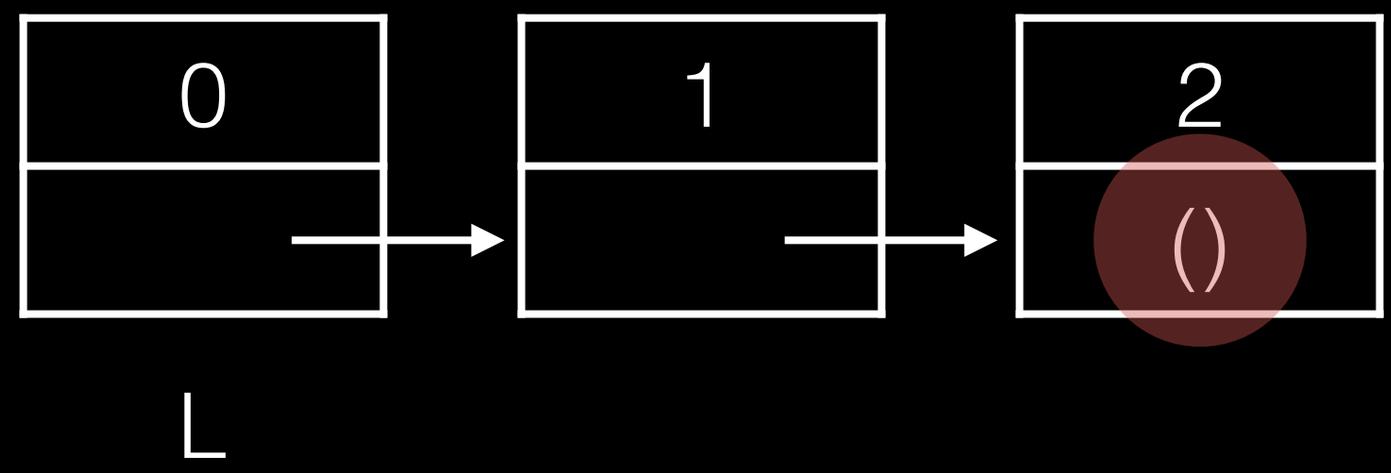
Linked Lists (print each element)

```
l = L
while ???:
    print(l.first)
    l = l.rest
```

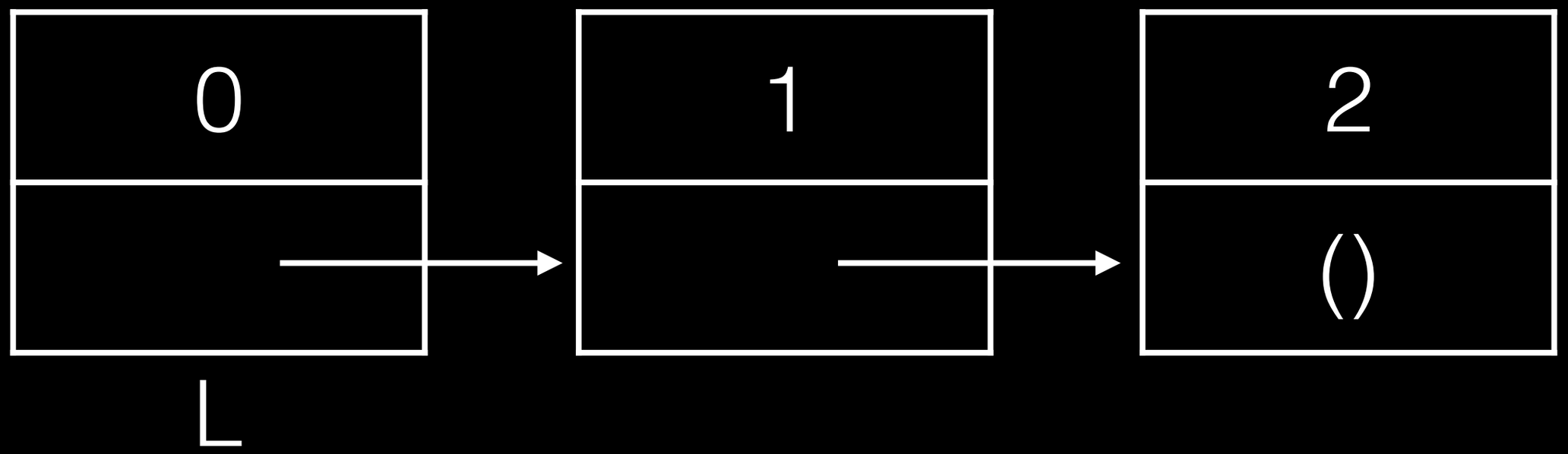


Linked Lists (print each element)

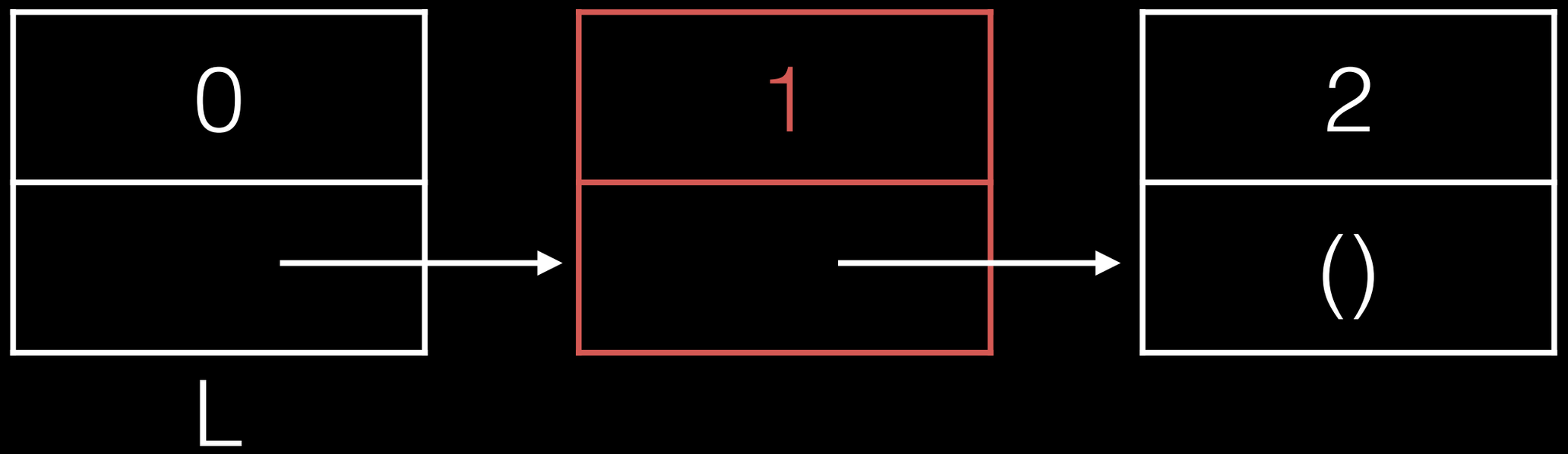
```
  l = L
● while l != empty:
      print(l.first)
      l = l.rest
```



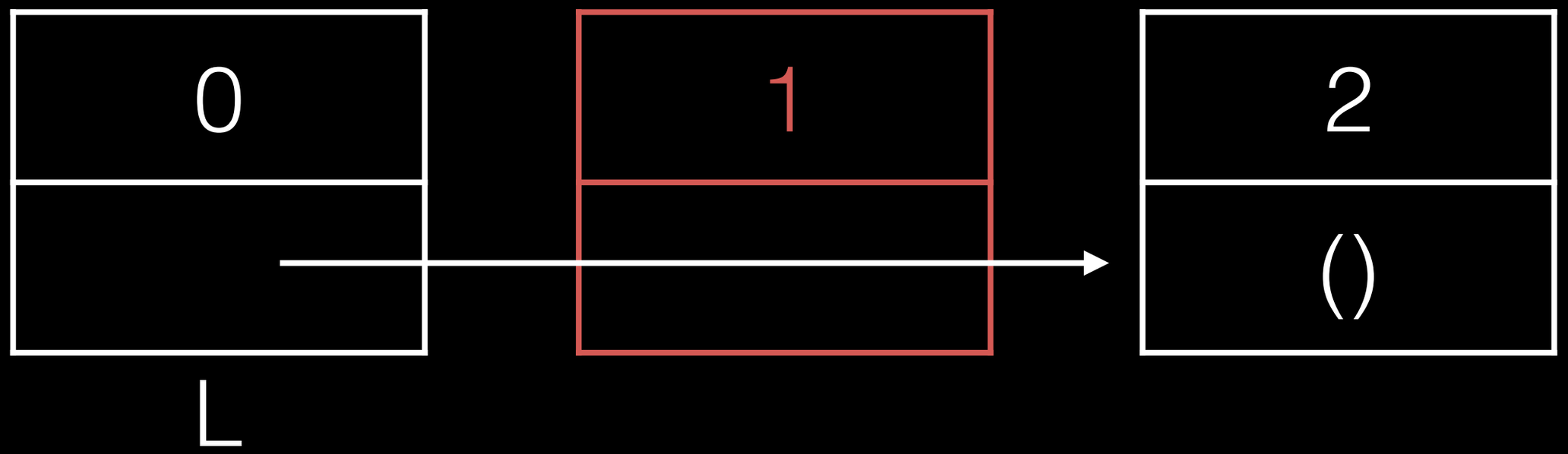
Linked Lists (deleting)



Linked Lists (deleting)



Linked Lists (deleting)



Linked Lists (deleting)

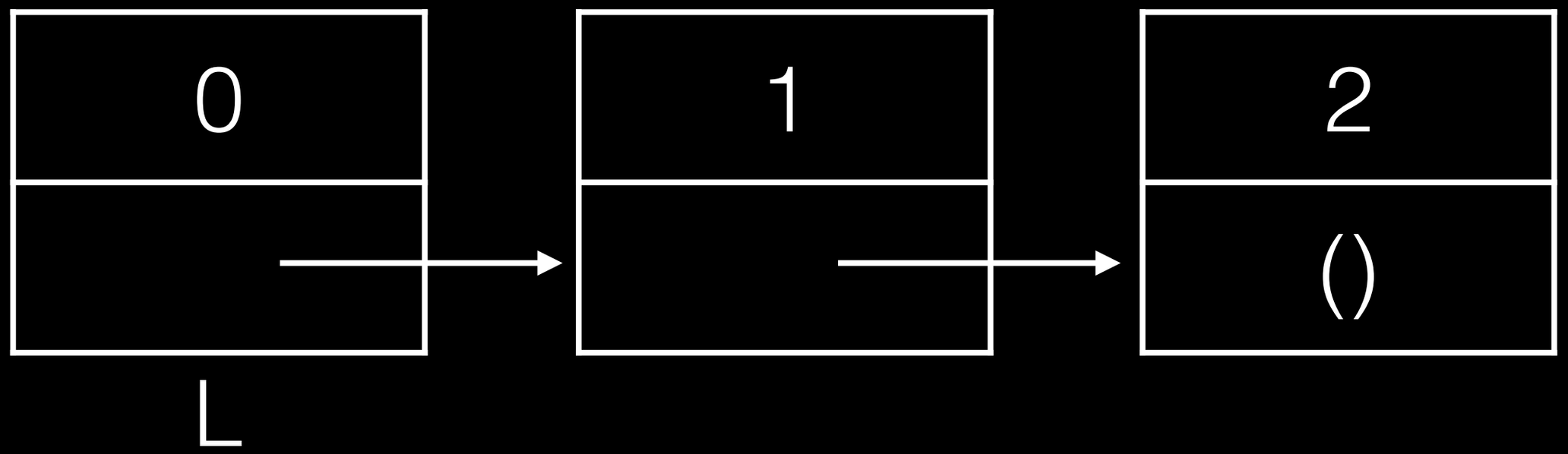
Linked Lists (deleting)

?

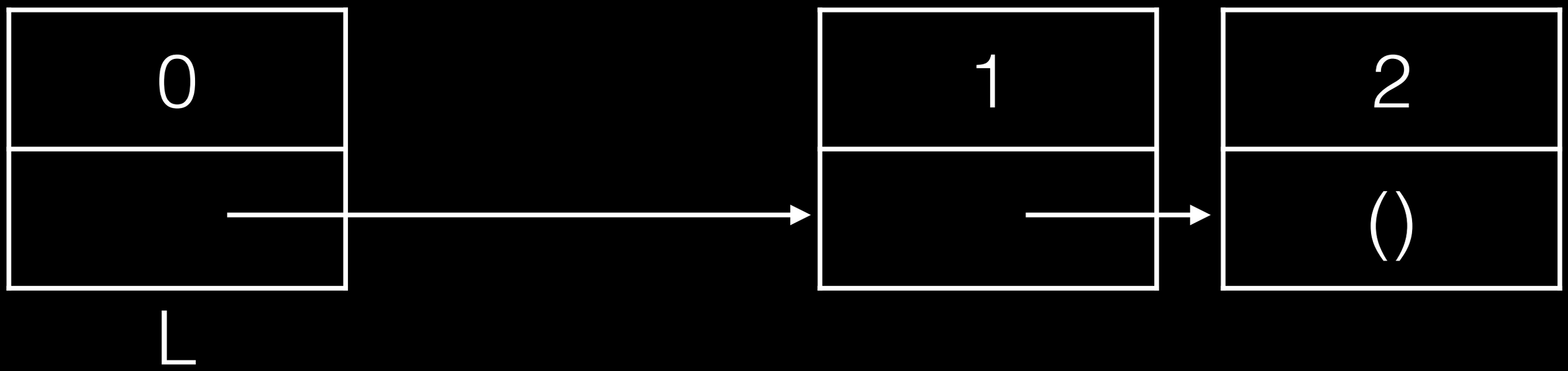
Linked Lists (deleting)

constant

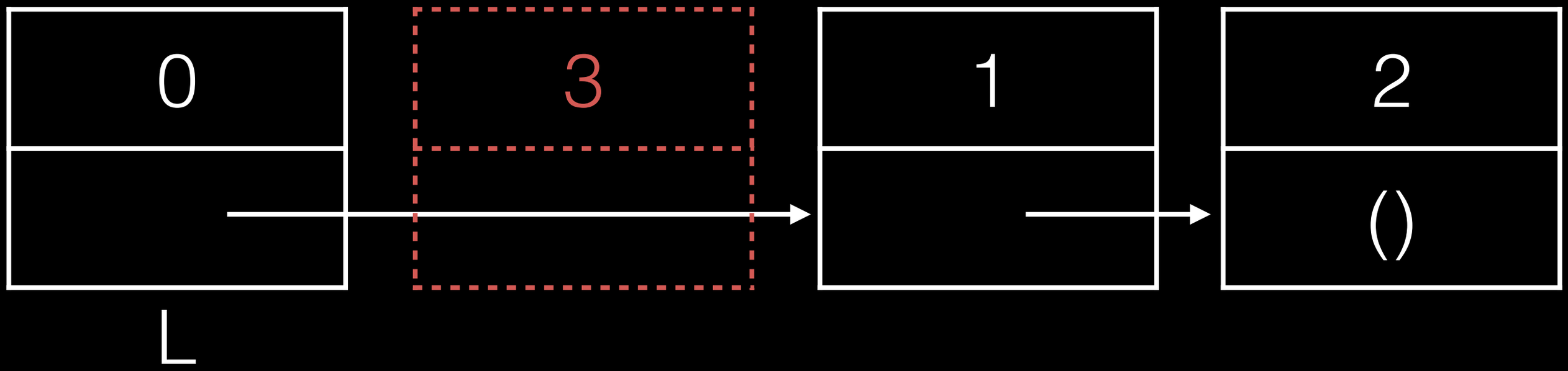
Linked Lists (inserting)



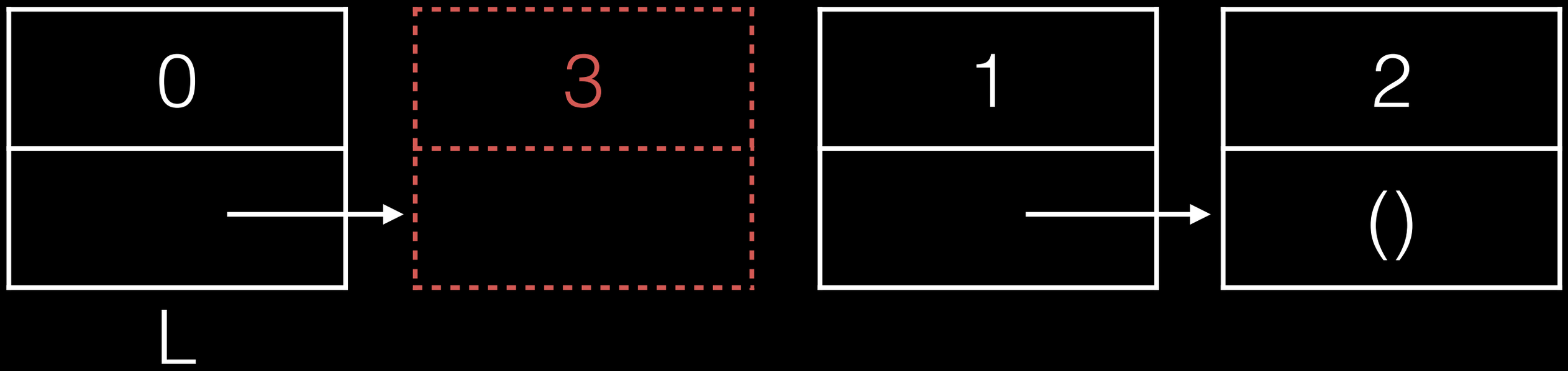
Linked Lists (inserting)



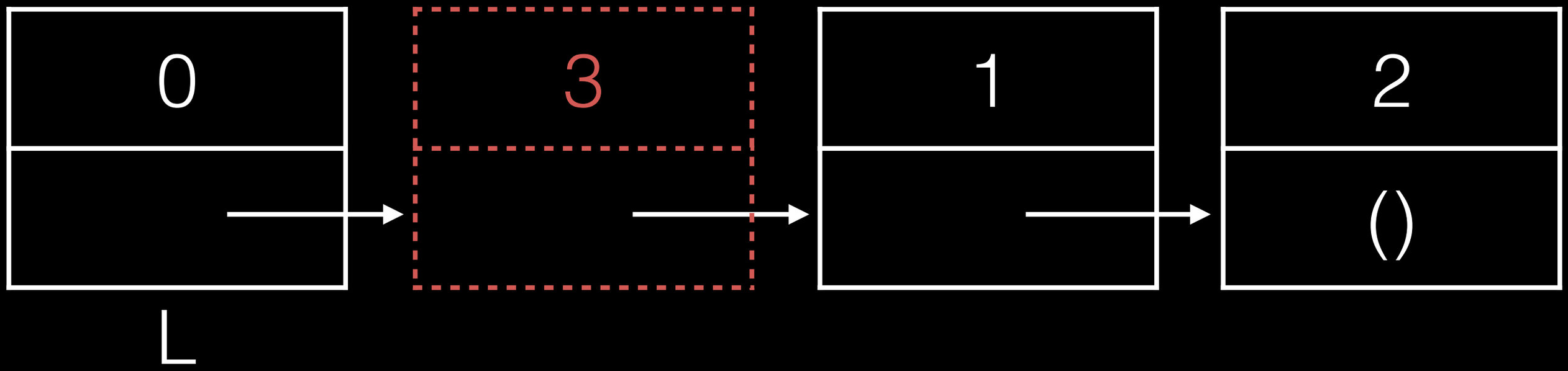
Linked Lists (inserting)



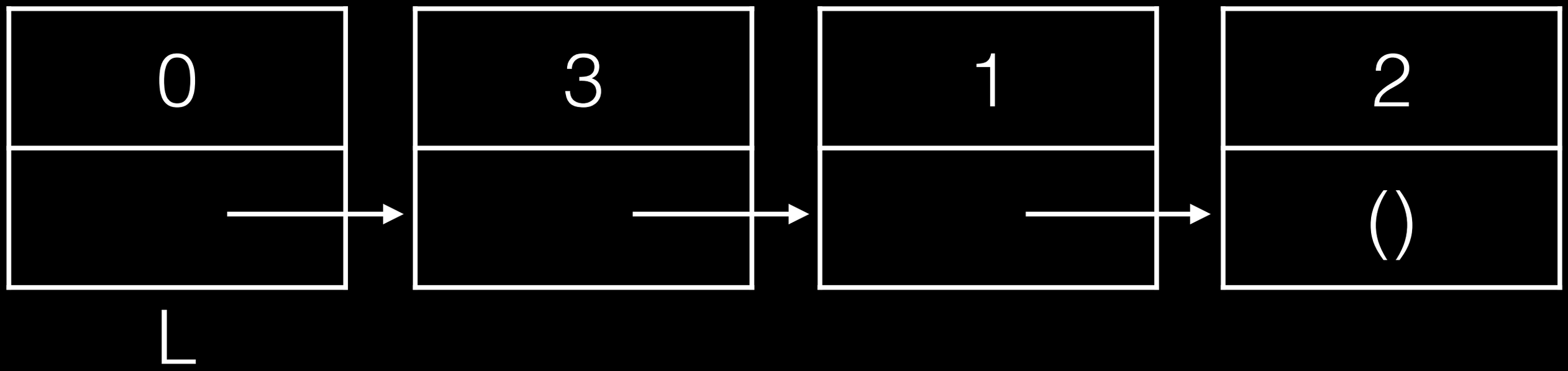
Linked Lists (inserting)



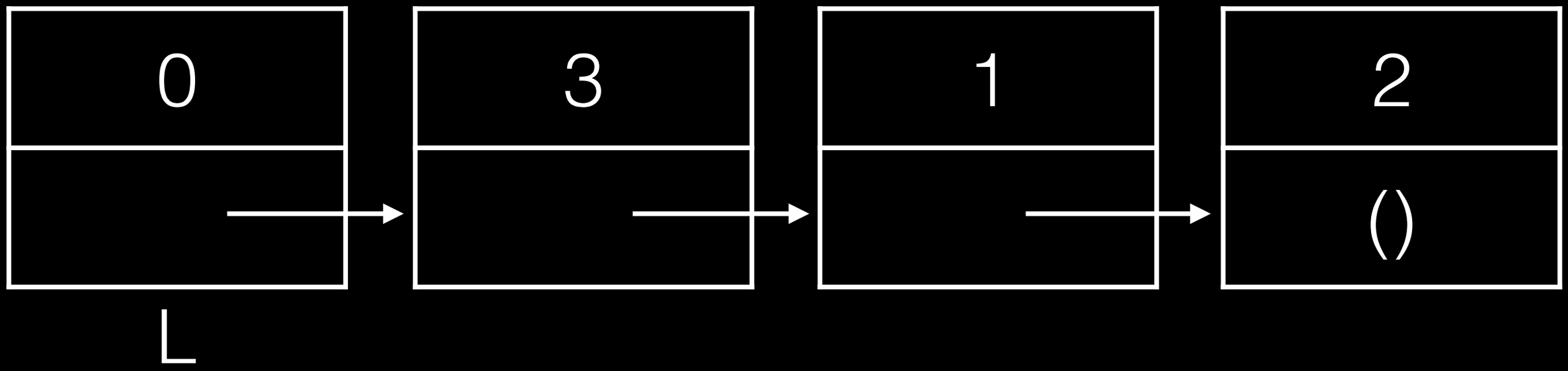
Linked Lists (inserting)



Linked Lists (inserting)

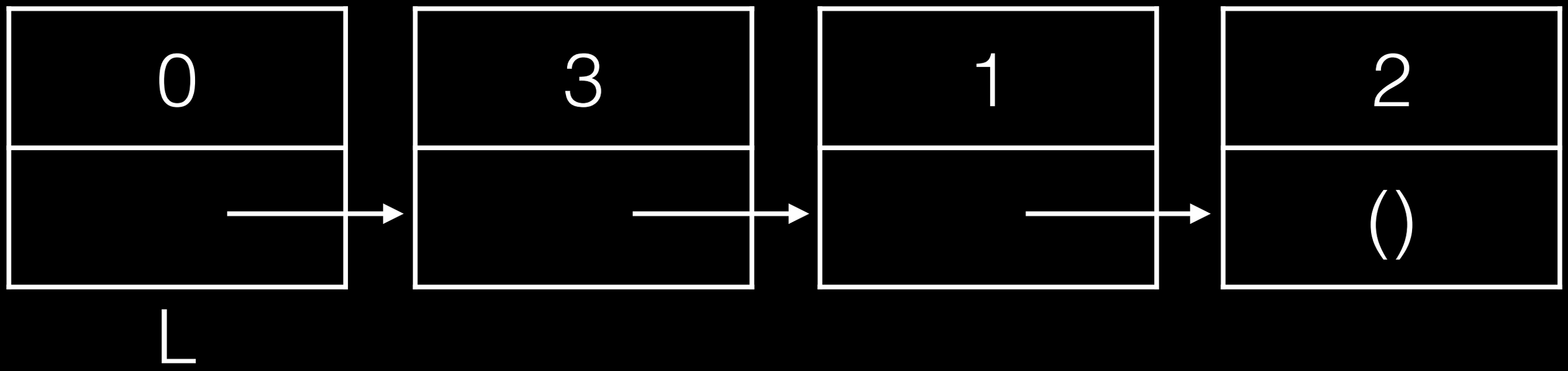


Linked Lists (inserting)



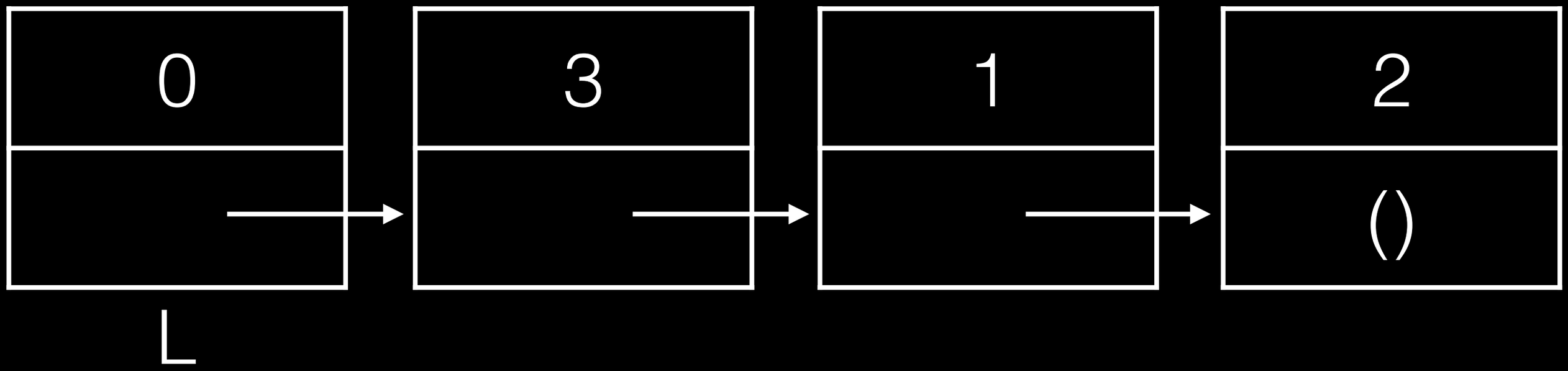
?

Linked Lists (inserting)

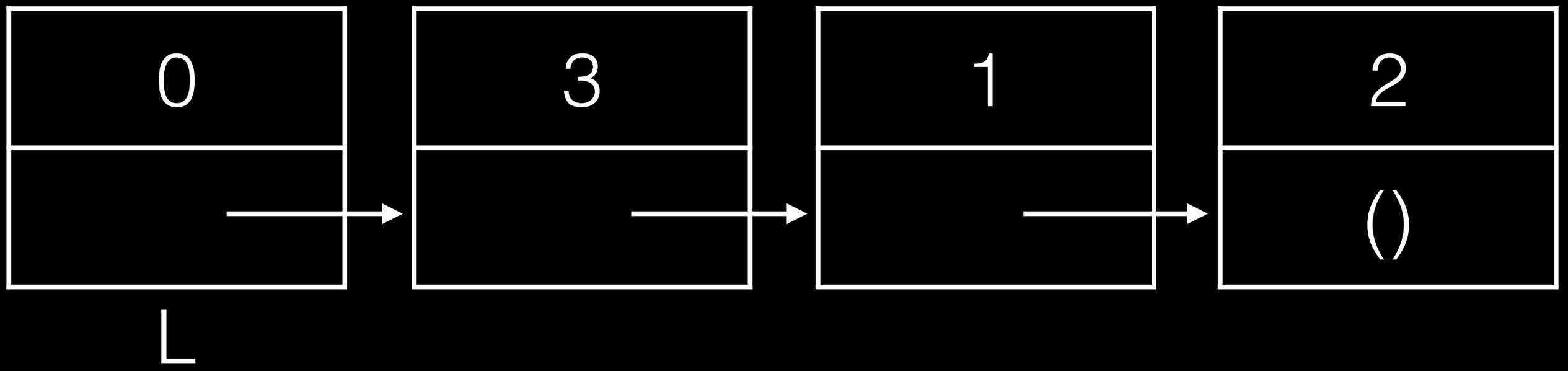


constant

Linked Lists (access)

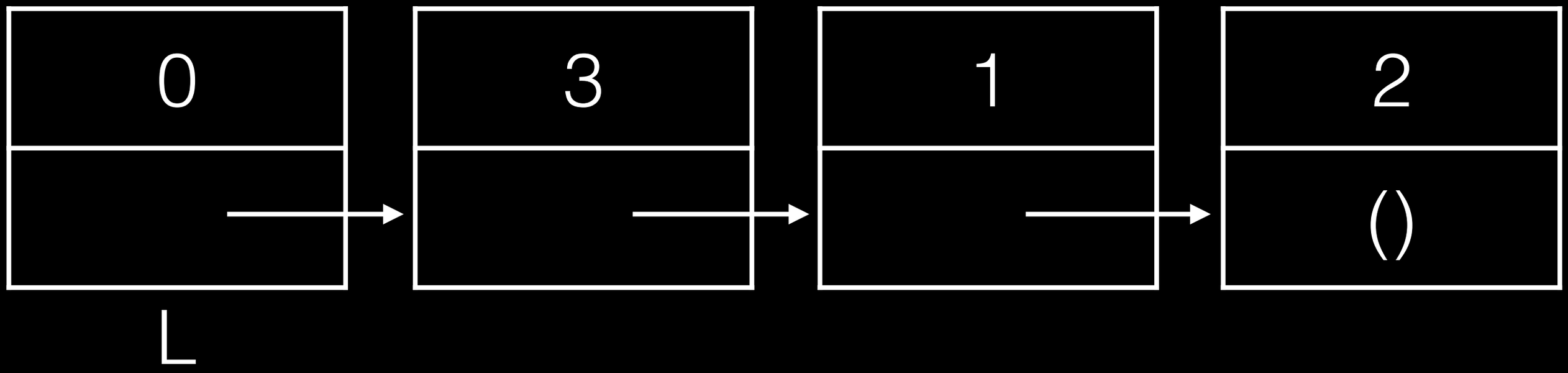


Linked Lists (access)



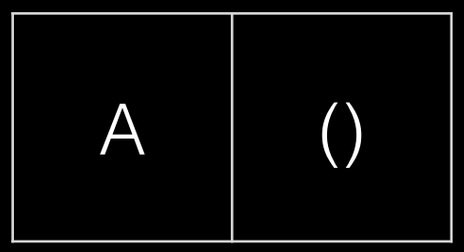
?

Linked Lists (access)

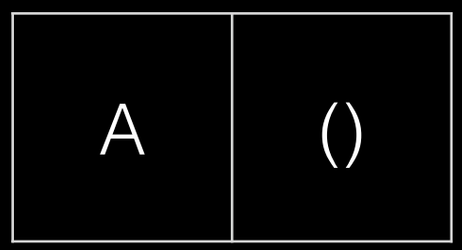


linear

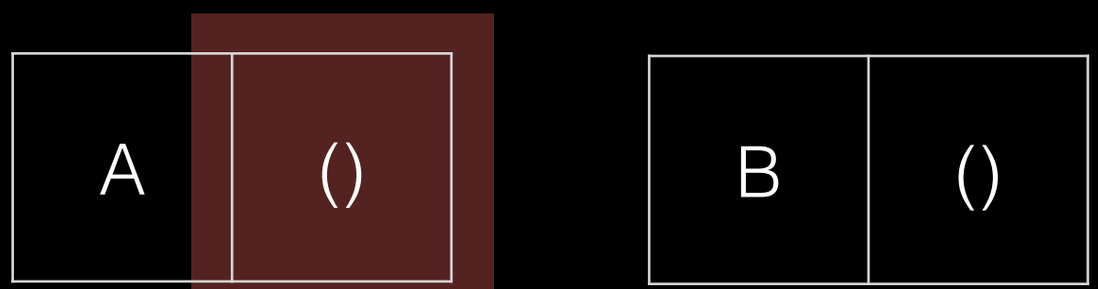
```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
```



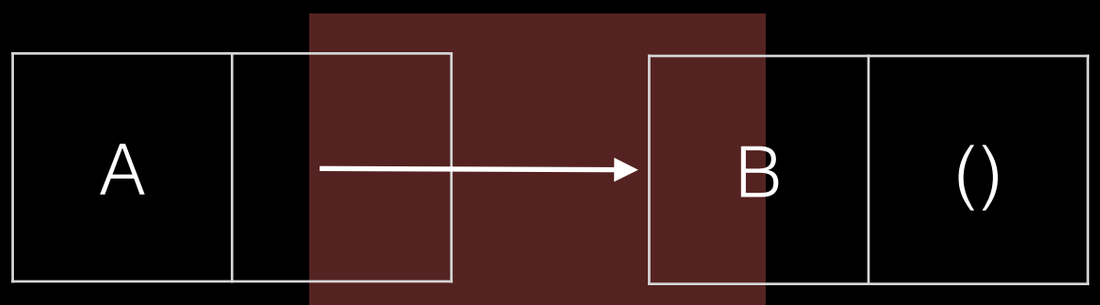
```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
```



```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
```

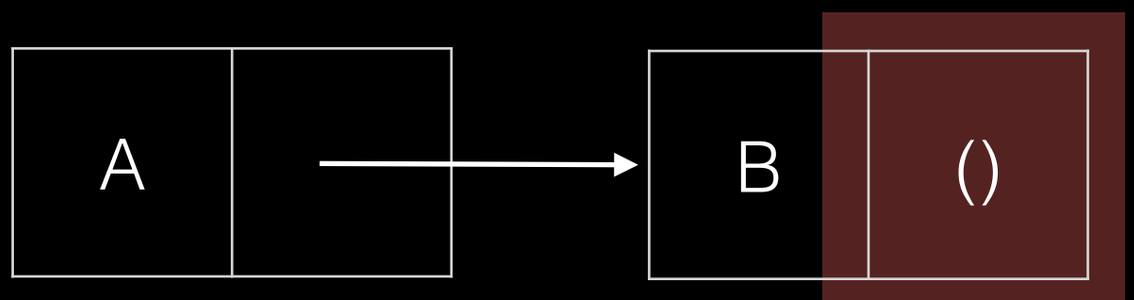


```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
```

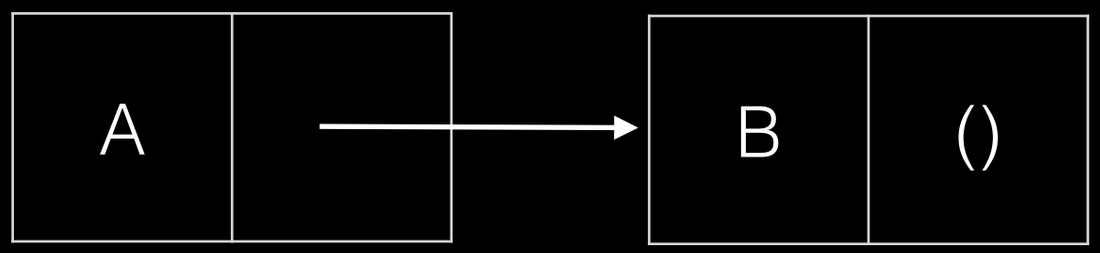


```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp
```

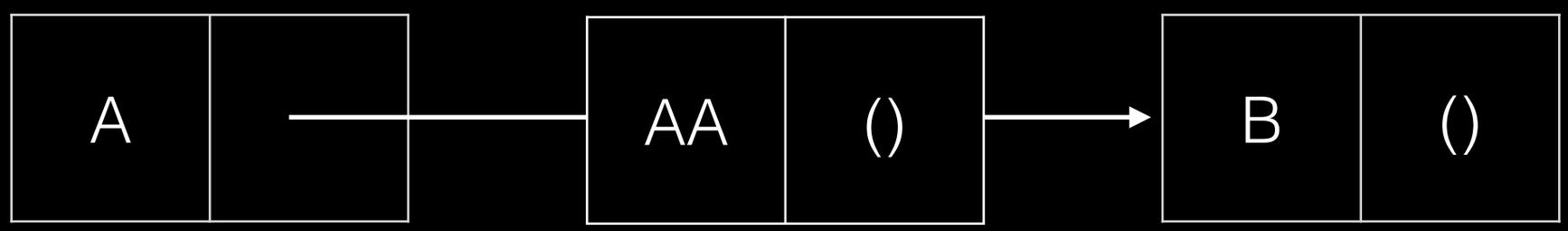
```
L = Link("A")
L.insertAfter( Link("B") )
```



```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
```



```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
L.insertAfter( Link("AA") )
```

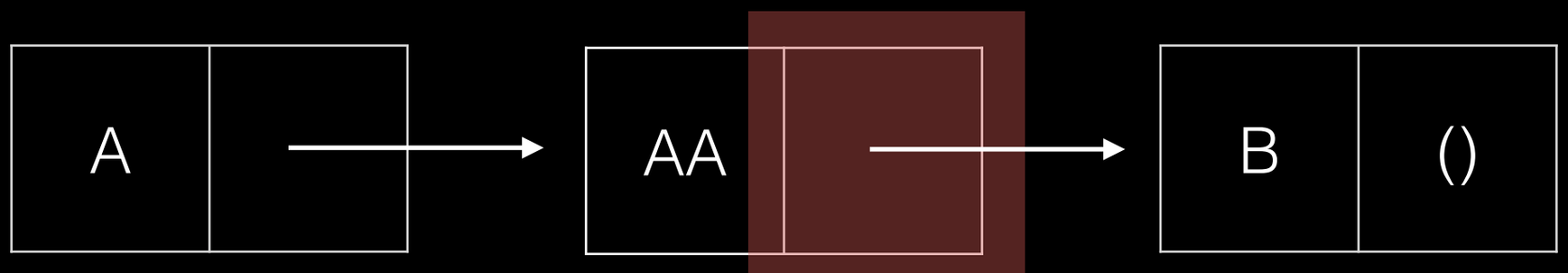


```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
L.insertAfter( Link("AA") )
```

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
L.insertAfter( Link("AA") )
```

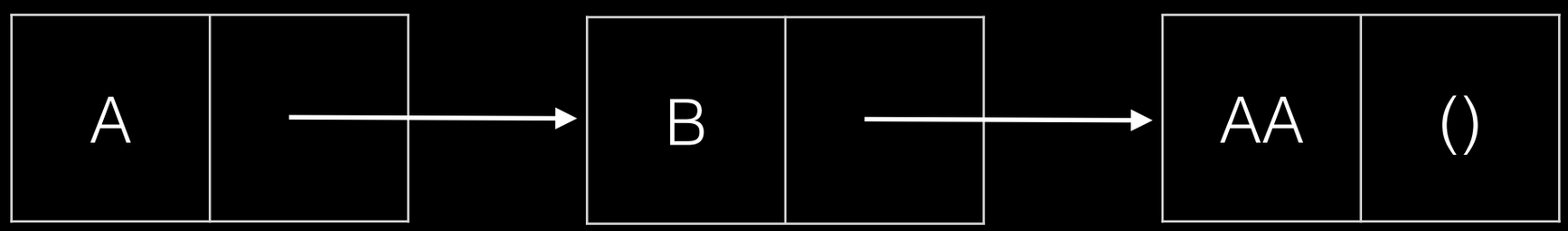
```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp
```

```
L = Link("A")
L.insertAfter( Link("B") )
L.insertAfter( Link("AA") )
```

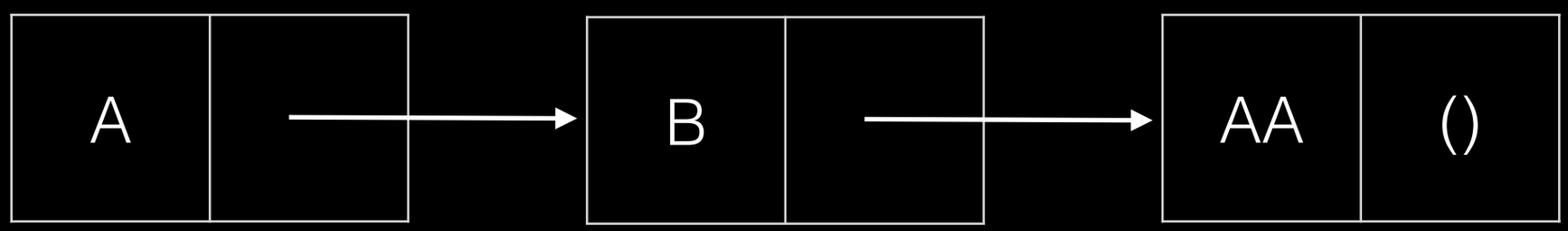


```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
L.insertAfter( Link("AA") )
```

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
```



```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    # insert a node (l) after a node
    def insertAfter(self, l):
        temp = self.rest
        self.rest = l
        l.next = temp

L = Link("A")
L.insertAfter( Link("B") )
L.rest.insertAfter( Link("AA") )
```



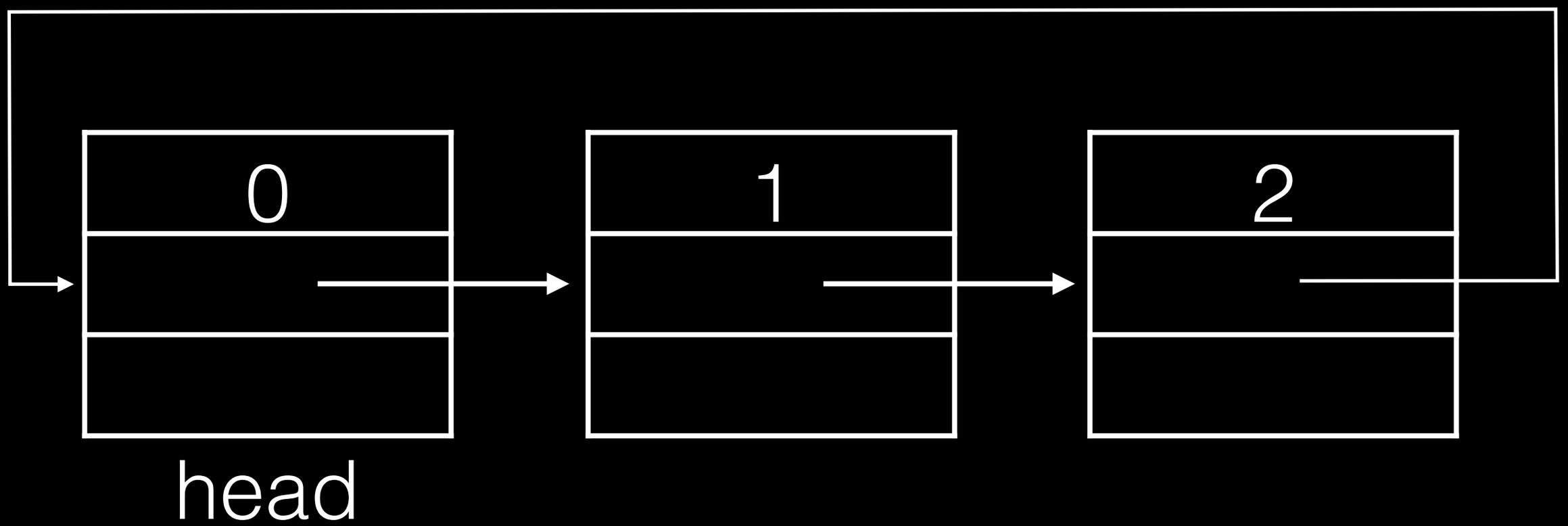
Lists

insert:	linear
append:	constant, sometimes linear
delete:	linear
find:	linear
access:	constant

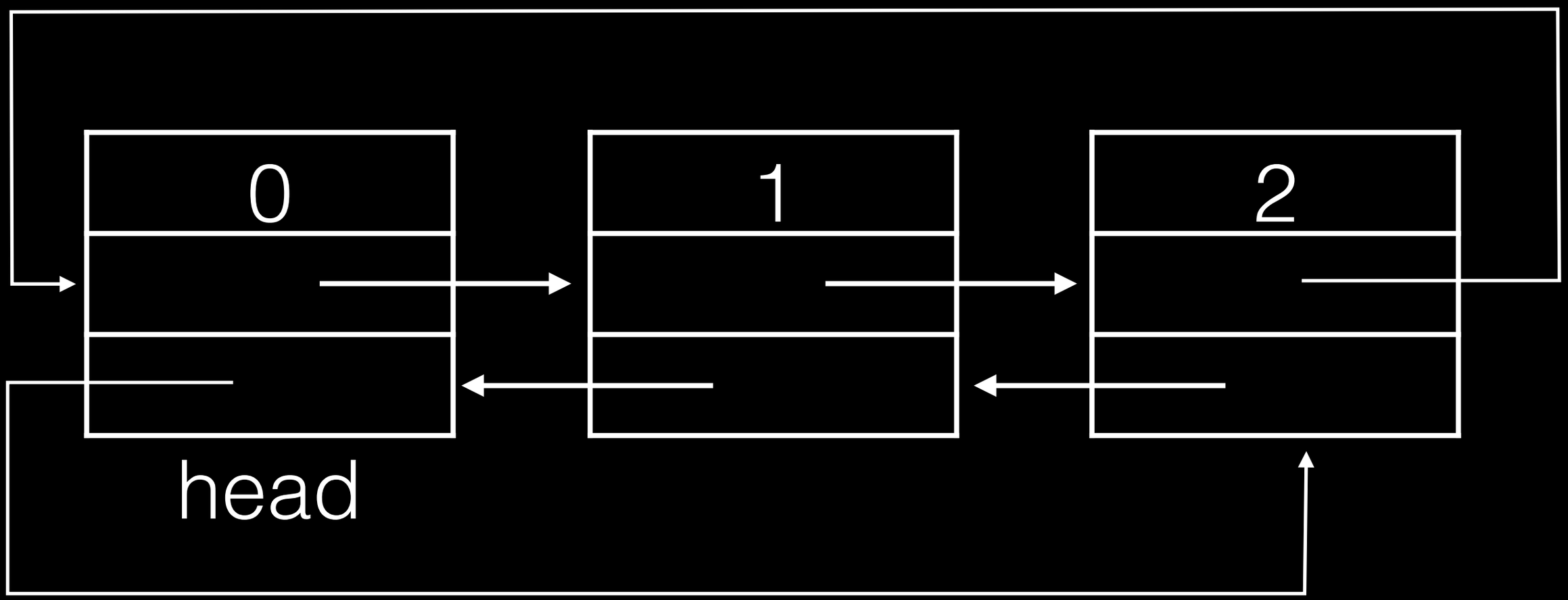
Linked Lists

insert:	constant
append:	constant
delete:	constant
find:	linear
access:	linear

Circular, Doubly Linked Lists



Circular, Doubly Linked Lists



```
class Dlink:
    def __init__(self, data):
        self.data = data
        self.next = self
        self.prev = self
```