Data Examples

Announcements

Lists

Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

Operation	Example	Result

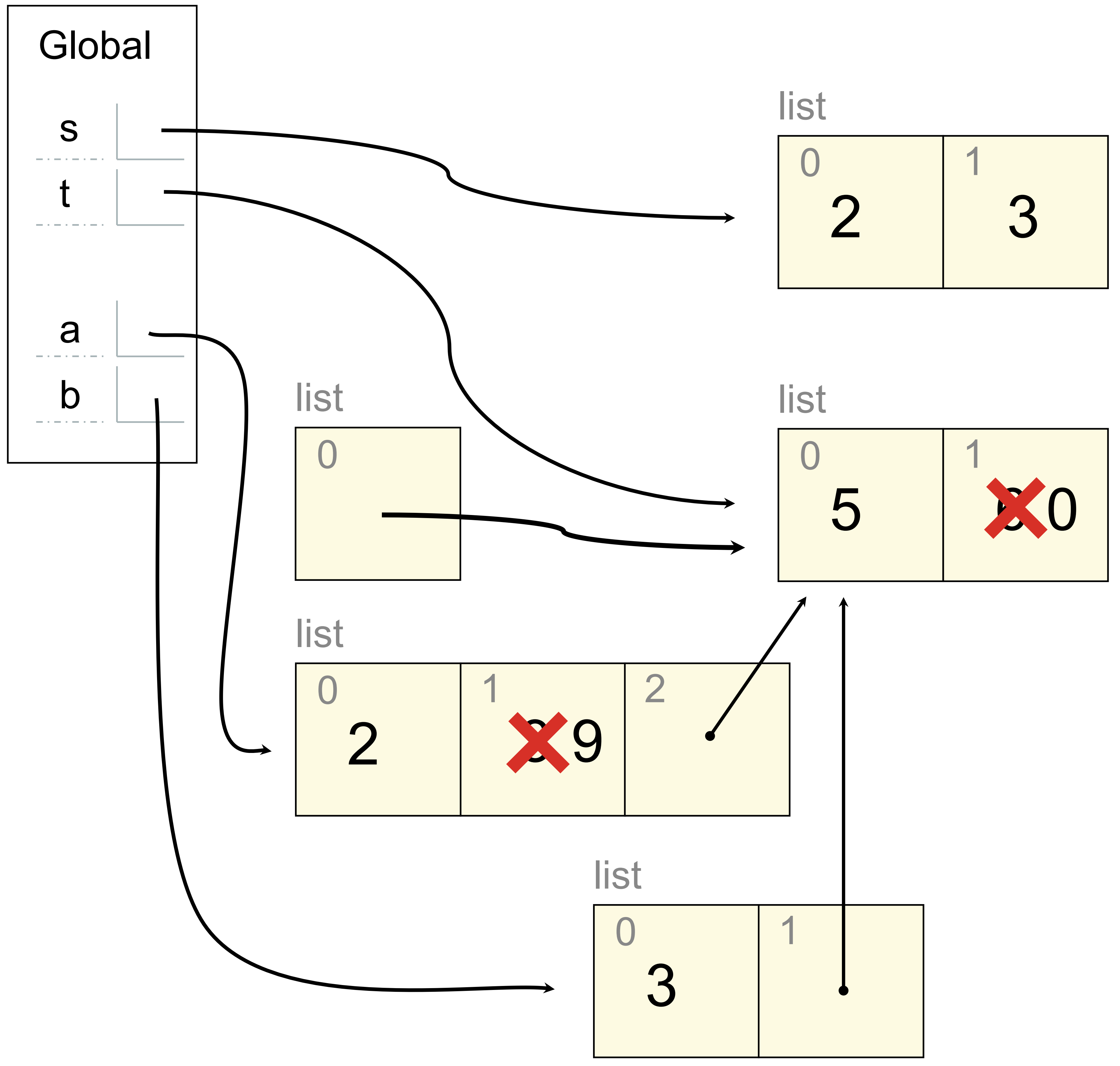
append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	t = list(s)
	s[1] = 0
		s → [2, 0]
		t → [2, 3]

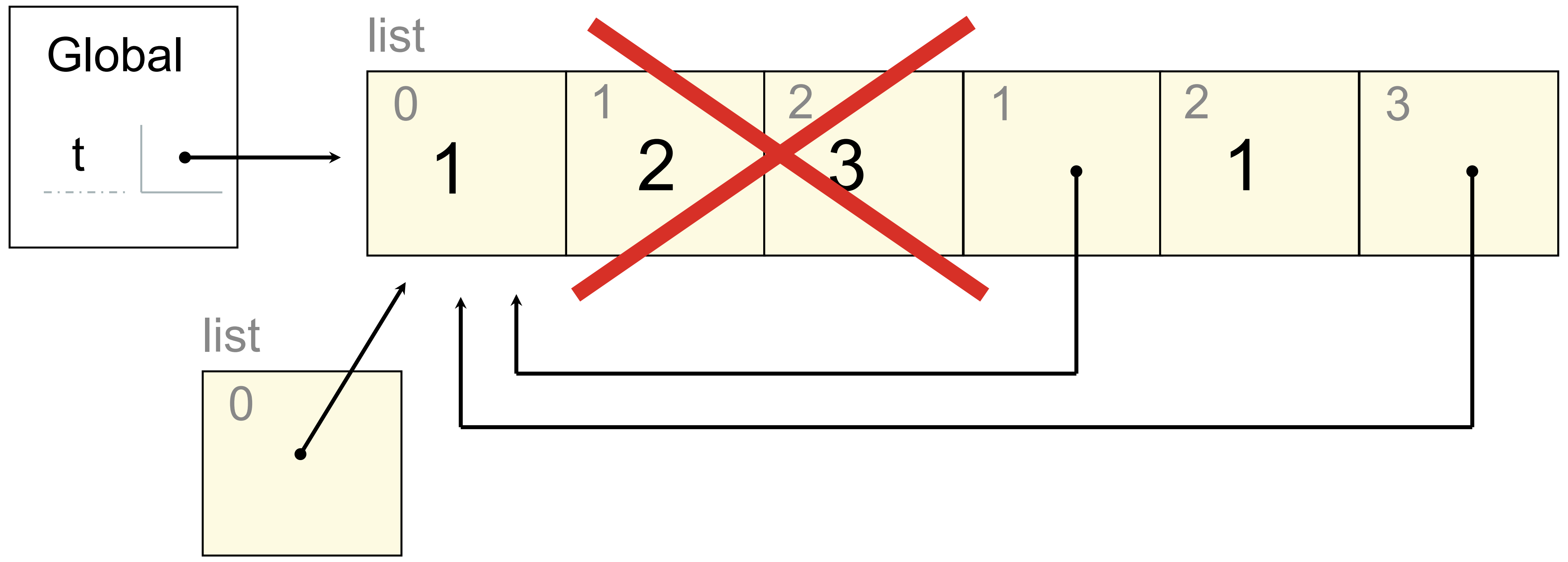
slice assignment replaces a slice with new values	s[0:0] = t s[3:] = t t[1] = 0	s → [5, 6, 2, 5, 6] t → [5, 0]



Lists in Lists in Lists in Environment Diagrams

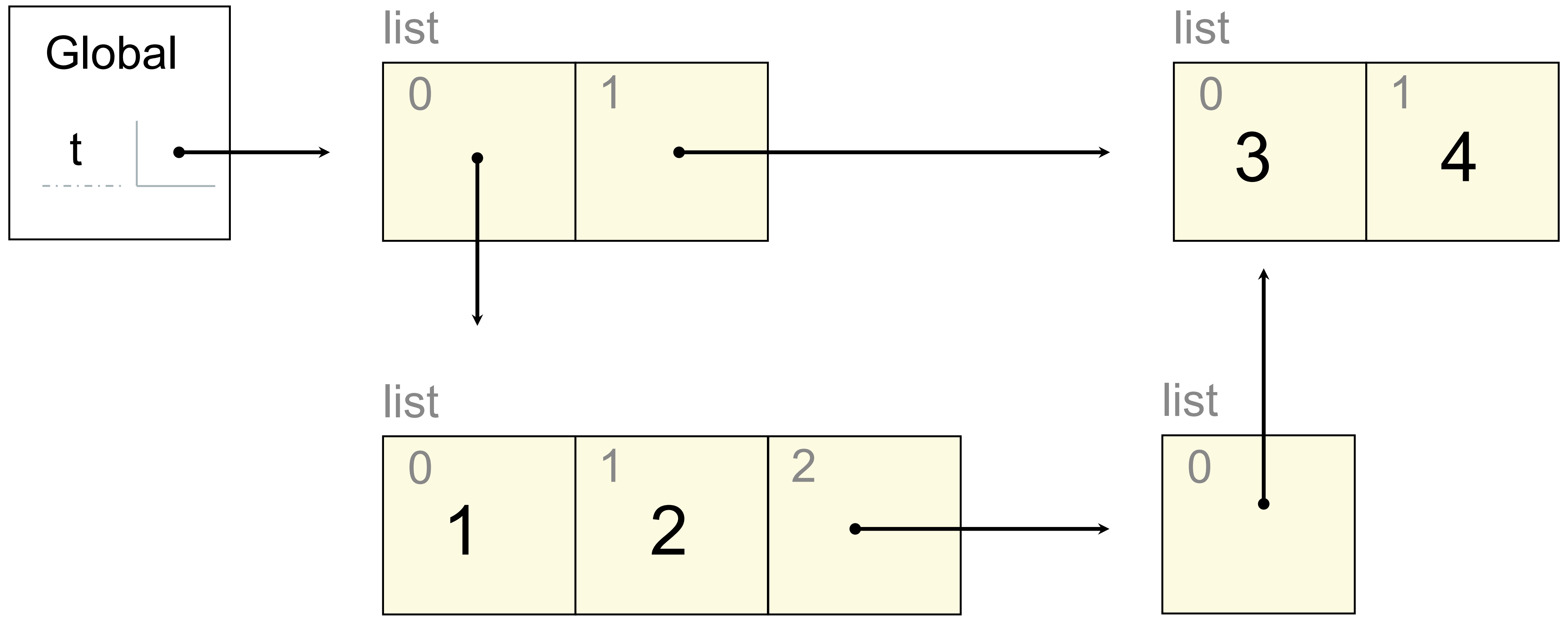
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```

[t] evaluates to:



[1, [...], 1, [...]]

```
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```



[[1, 2, [[3, 4]]], [3, 4]]

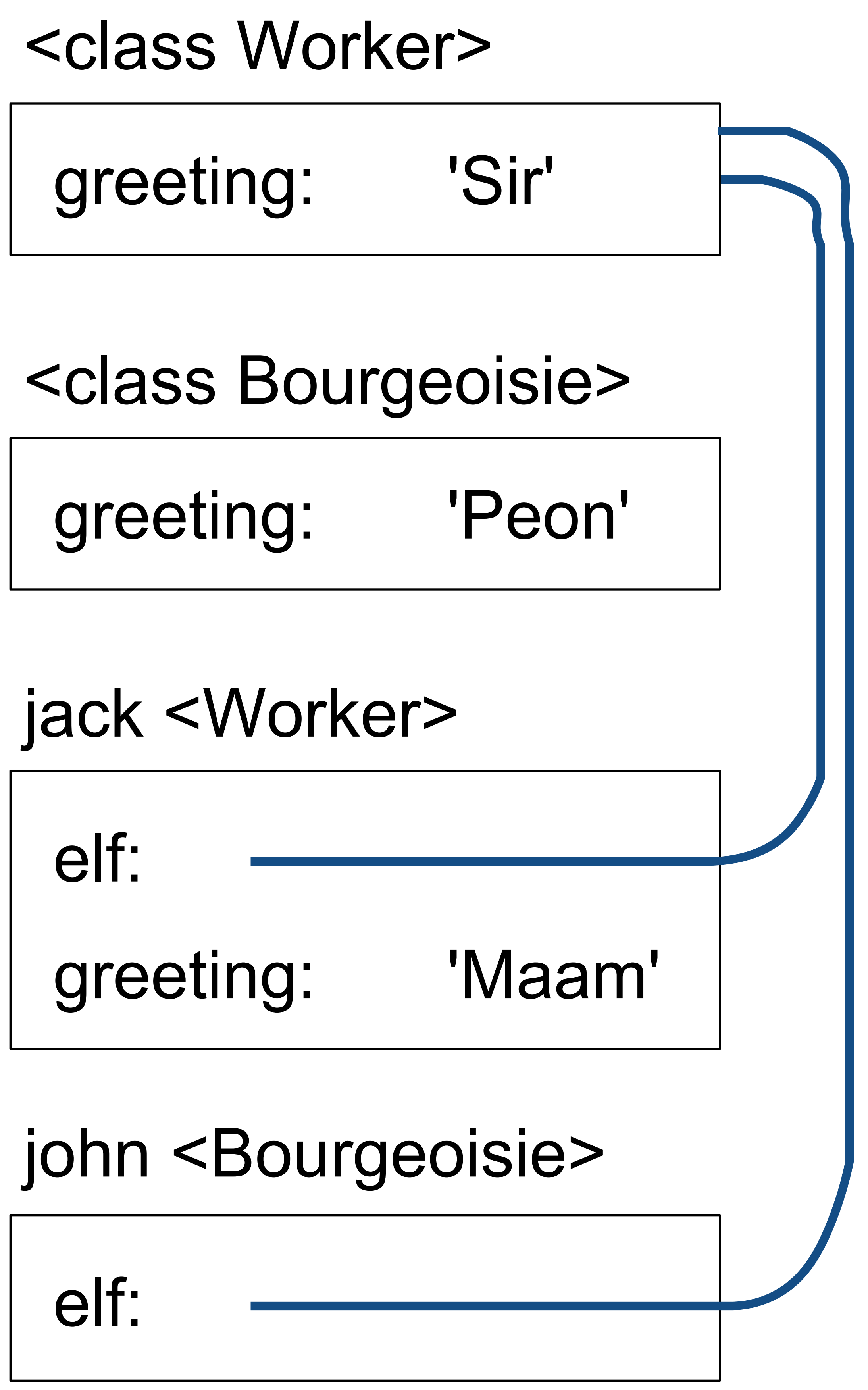
Objects

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()
Peon, I work
'I gather wealth'

>>> john.elf.work(john)
'Peon, I work'
```



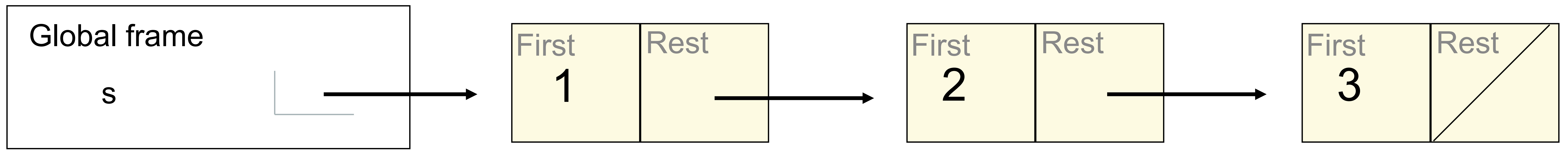
Linked Lists

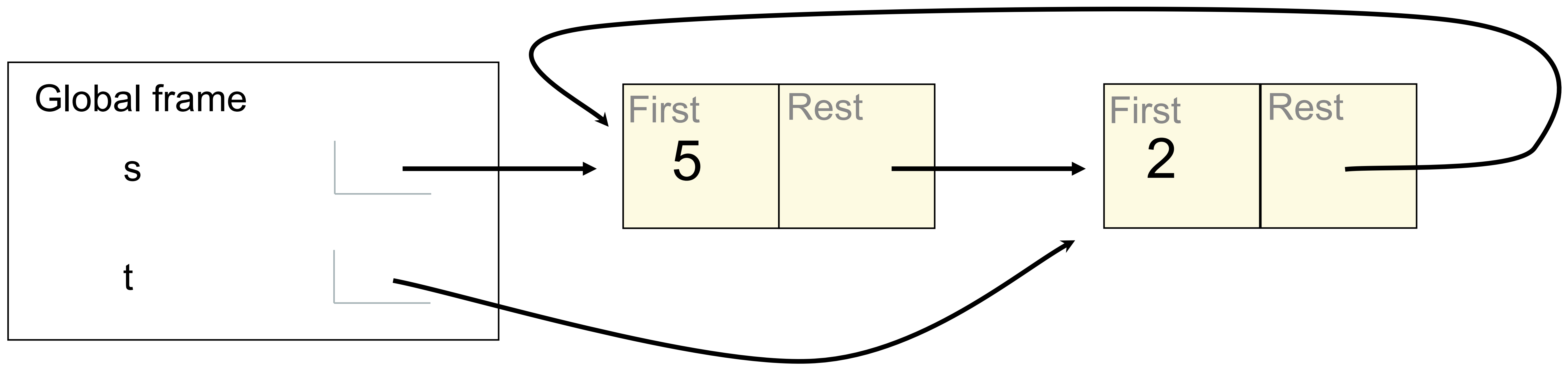
Recursive Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.rest.first
2
```





Note: The actual environment diagram is much more complicated.

Trees

Morse Code

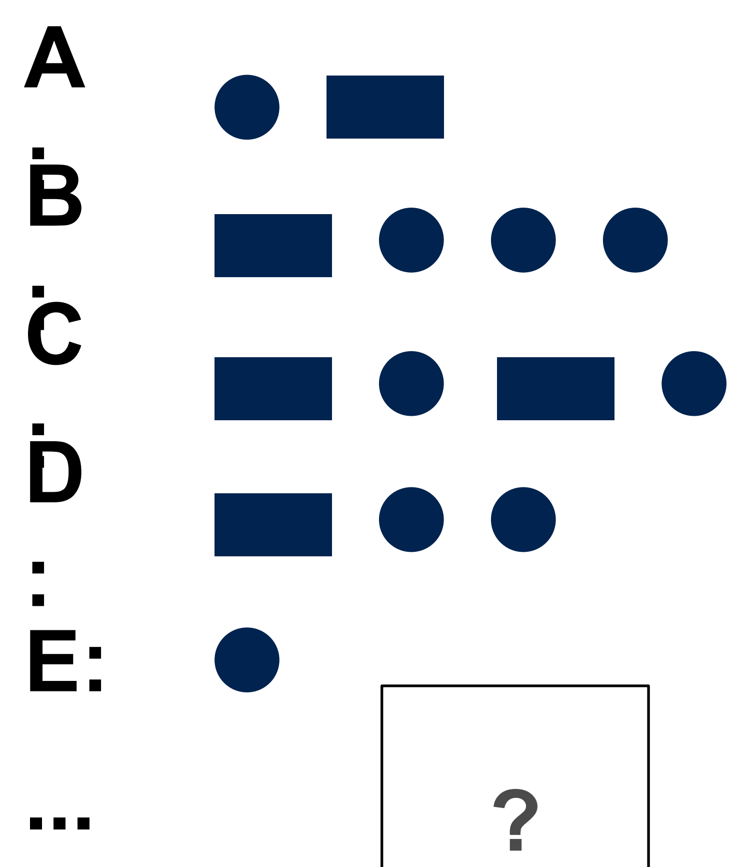
Morse code is a signaling protocol that transmits messages by sequences of signals

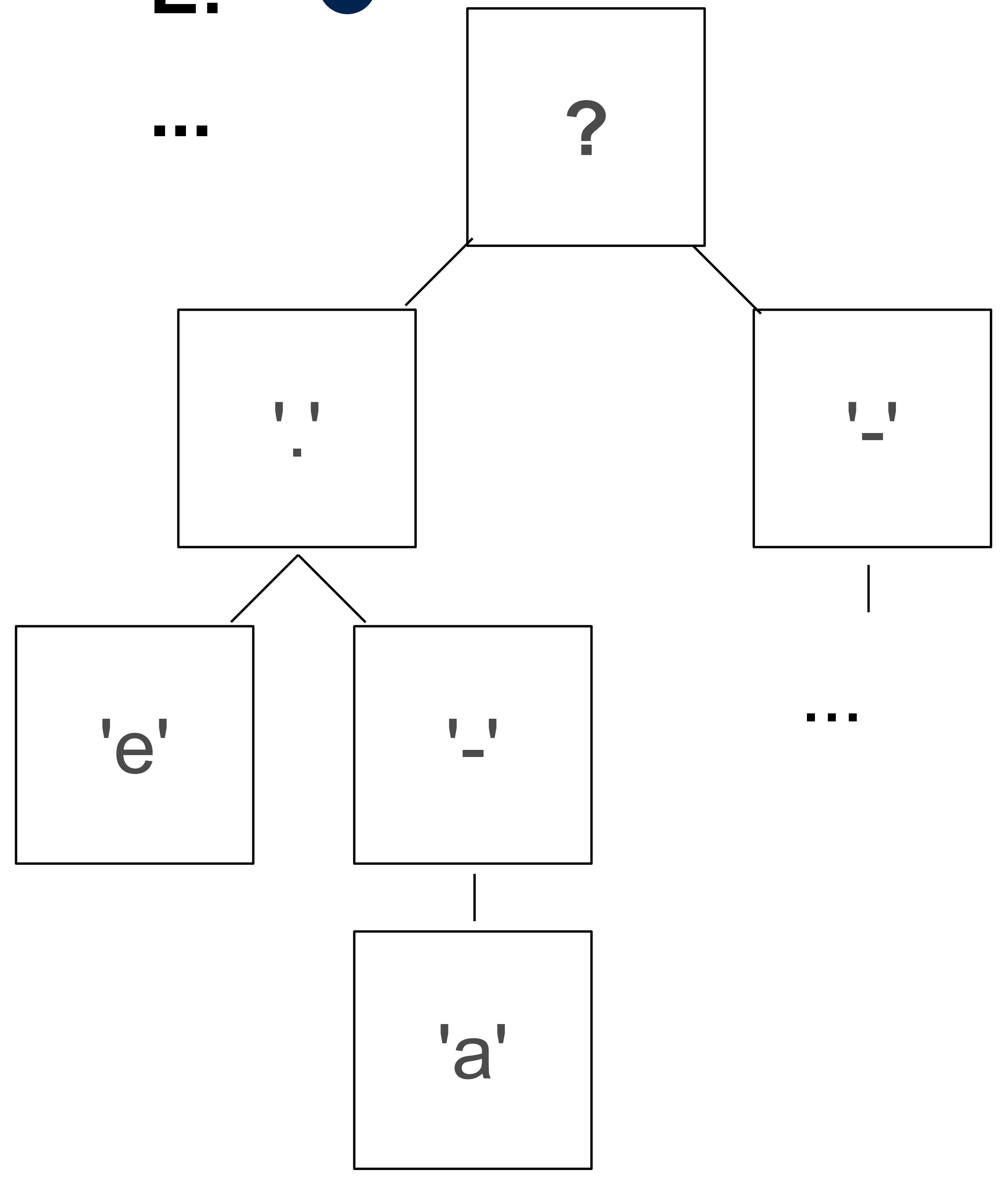
Problem: Implement **morse** so that **decode** works correctly

```
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

    >>> t = morse(abcde)
    >>> [decode(s, t) for s in ['-..', '.', '-.-.', '.-', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    """
    for signal in signals:
        tree = [b for b in tree.branches if b.label == signal][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].label
```

```
def morse(code):
    ....
```

```
decode('.', t)
```





(Demo)