Composition

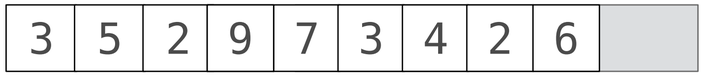
Announcements

List Efficiency

List Efficiency

Appending:

Removing (beginning/middle):



List Efficiency

Appending:

Removing (beginning/middle):

List Efficiency

Appending, assigning, and list comprehensions are fast:

Inserting (beginning/middle), slicing, and adding lists are slow:

Example: building long lists of perfect squares (numbers that are an integer times itself)

```
def using_list_comprehension(n):
    return [k * k for k in range(n)]

def using_append(n):
    s = []
    for k in range(n):
        s.append(k * k)
    return s

def using_assign(n):
    s = [0 for k in range(n)]
    for k in range(n):
        s[k] = k * k
    return s
```

```
def using_insert(n):
    s = []
    for k in range(n):
        s.insert(0, k * k)
    return s

def using_add(n):
    s = []
    for k in range(n):
        s = s + [k*k]
    return s
```

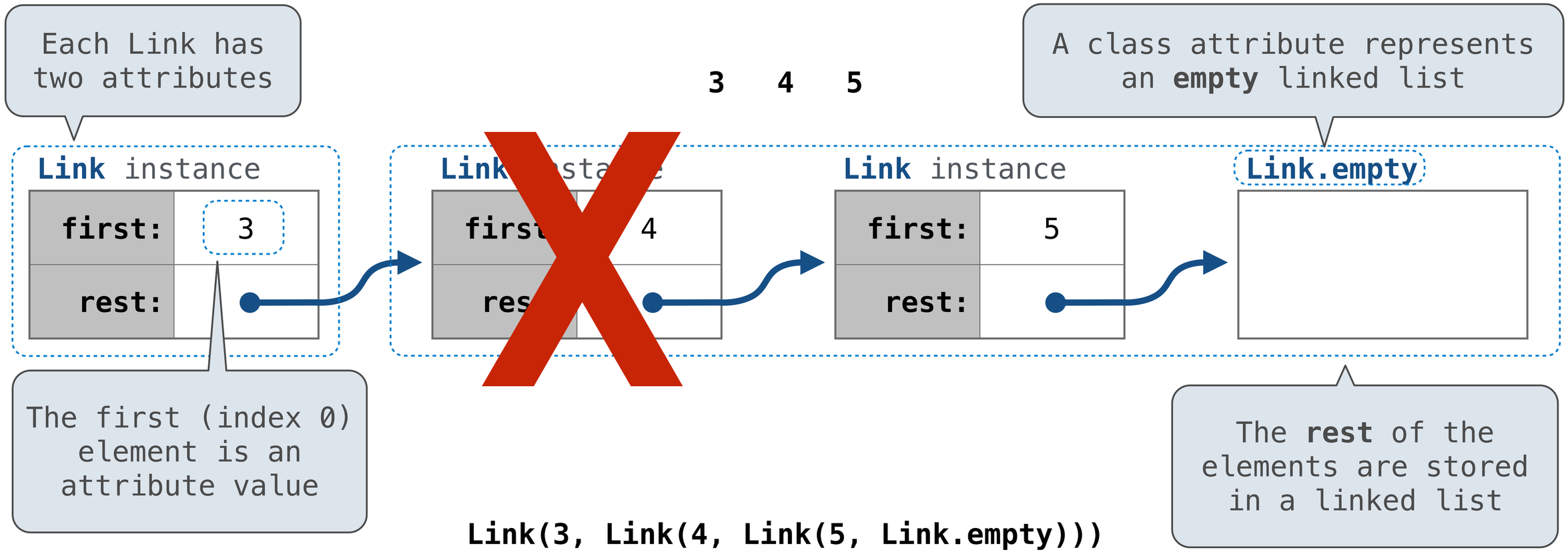
```
When n = 100,000
using_list_comprehension:  1.58 ms
using_append:              1.76 ms
using_assign:              2.58 ms
using_insert:          1,470    ms
using_add:             9,210    ms
```

Linked Lists

(Demo)

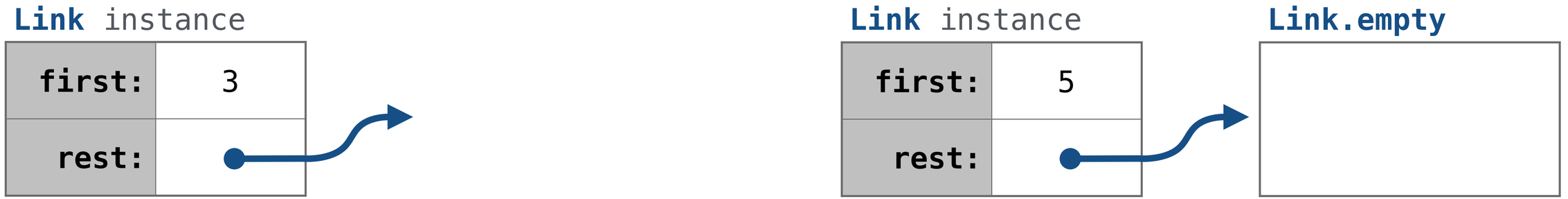
Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



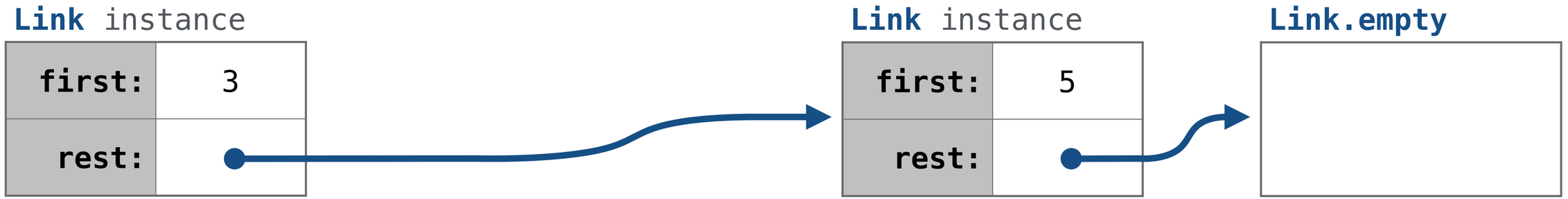
Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



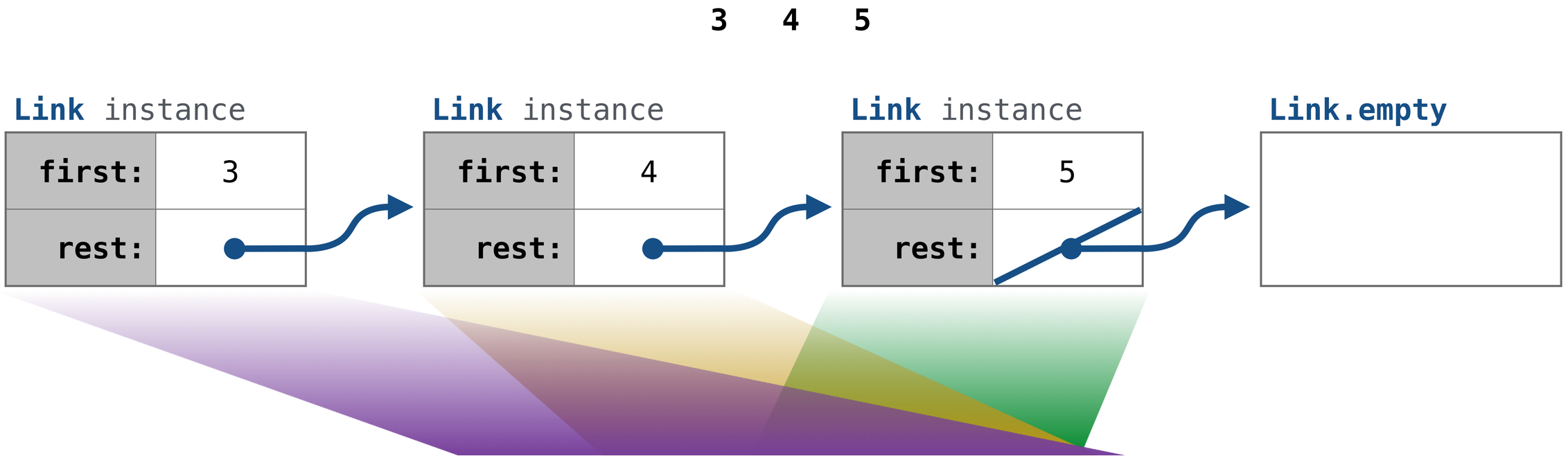
Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



The repr: `Link(3, Link(4, Link(5, Link.empty)))`

The str: (3 4 5) *but old midterms will have* <3 4 5> *instead*

The fast way to draw it:

Linked List Class

Linked list class: attributes are passed to `__init__`

```
class Link:

    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
```

A False value that looks empty when printed

Returns whether rest is a Link

help(isinstance): Return whether an object is an instance of a class or of a subclass thereof.

```
Link(3, Link(4, Link(5                )))
```

(Demo)

Discussion Questions

What will be displayed by...

```
>>> v = Link(1, Link(Link(2, Link(3)), Link(4)))
>>> print(v)
(1 (2 3) 4)
```

What expression starting with v. evaluates to 3?

```
v.rest.first.rest.first
```

Repeated Inserts

Double a List

```
def double(s, v):
    """Insert another v after each v in list s.

    >>> s = [2, 7, 1, 8, 2, 8]
    >>> double(s, 2)
    >>> s
    [2, 2, 7, 1, 8, 2, 2, 8]
    """
    i = 0
    while i < len(s):
        if s[i] == v:
            s.insert(i+1, v)
            i += 2
        else:
            i += 1
```

Double a Linked List

```
def double_link(s, v):
    """Insert another v after each v in linked list s.

    >>> t = Link(2, Link(7, Link(1, Link(8, Link(2, Link(8))))))
    >>> double_link(t, 2)
    >>> print(t)
    (2 2 7 1 8 2 2 8)
    """

    while s is not Link.empty:

        if s.first == v:

            s.rest = Link(v, s.rest)

            s = s.rest.rest

        else:

            s = s.rest
```