#9 Delayed Expressions

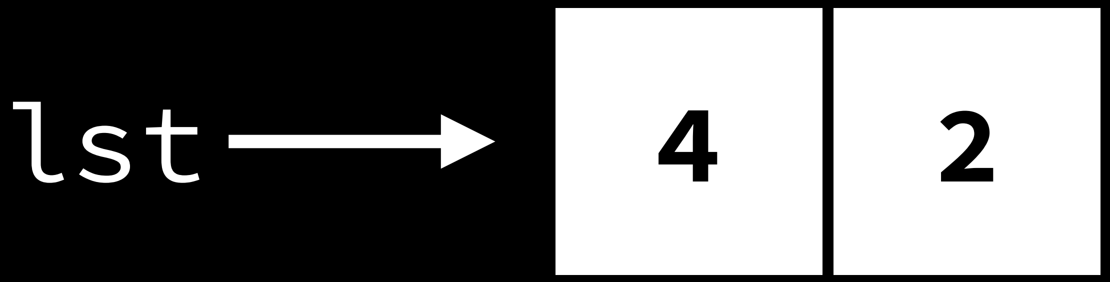
TA: Jerry Chen (jerry.c@berkeley.edu)

Have you seen the new Yelp, but for sequences? It's a great "iter rater."

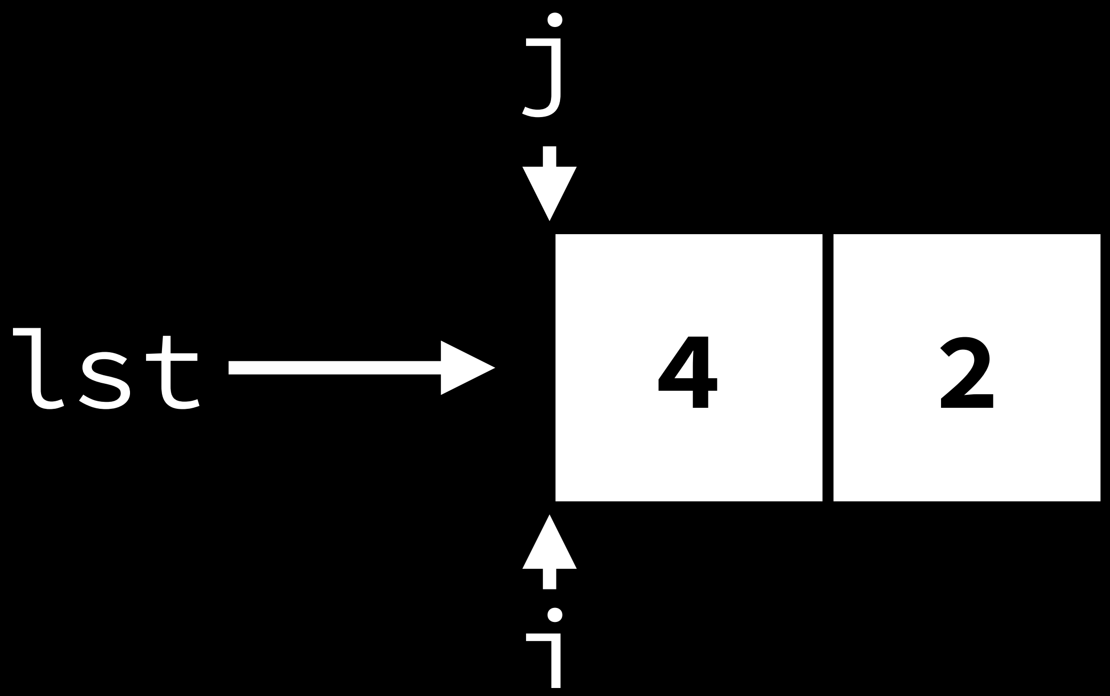
Iterators and Iterables

- **Iterable** - usually represents a sequence, can call iter on it to get an
- **Iterator** - represents a position in a sequence. Return next item by calling next

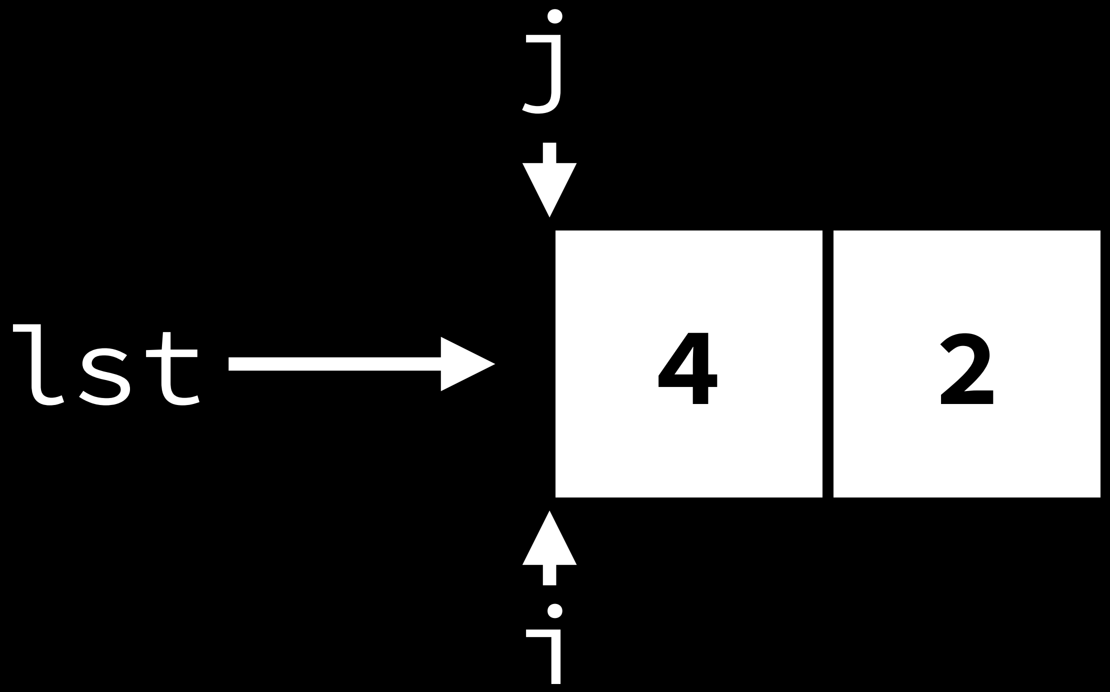
```
lst = [4, 2]
```



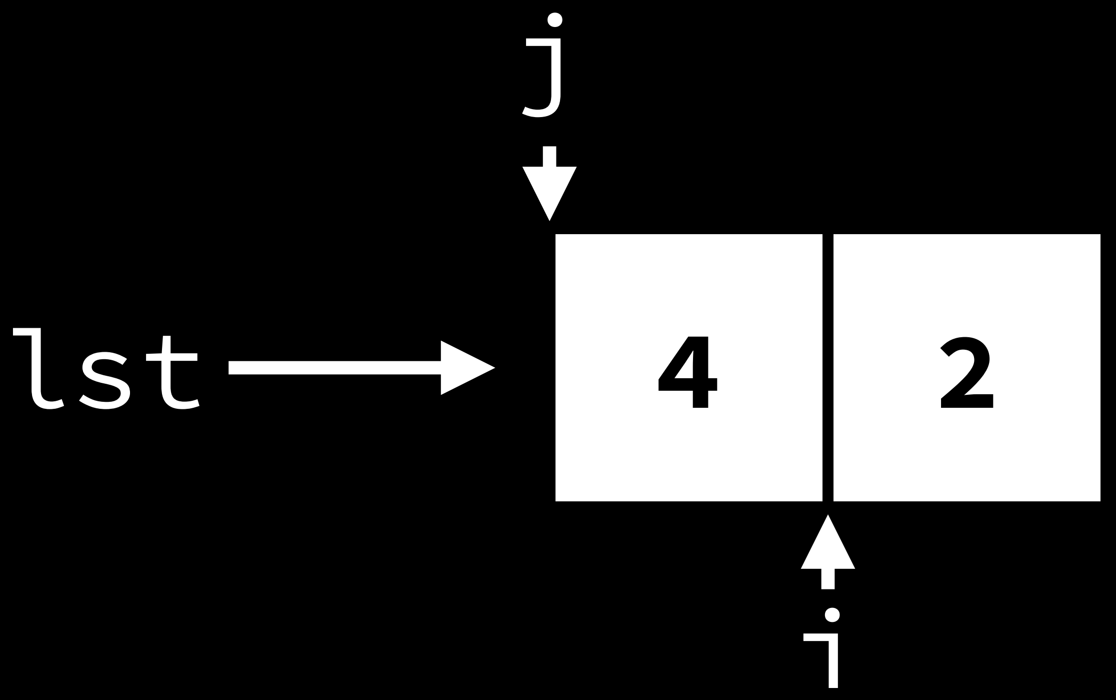
```python
i = iter(lst)
j = iter(lst)
```



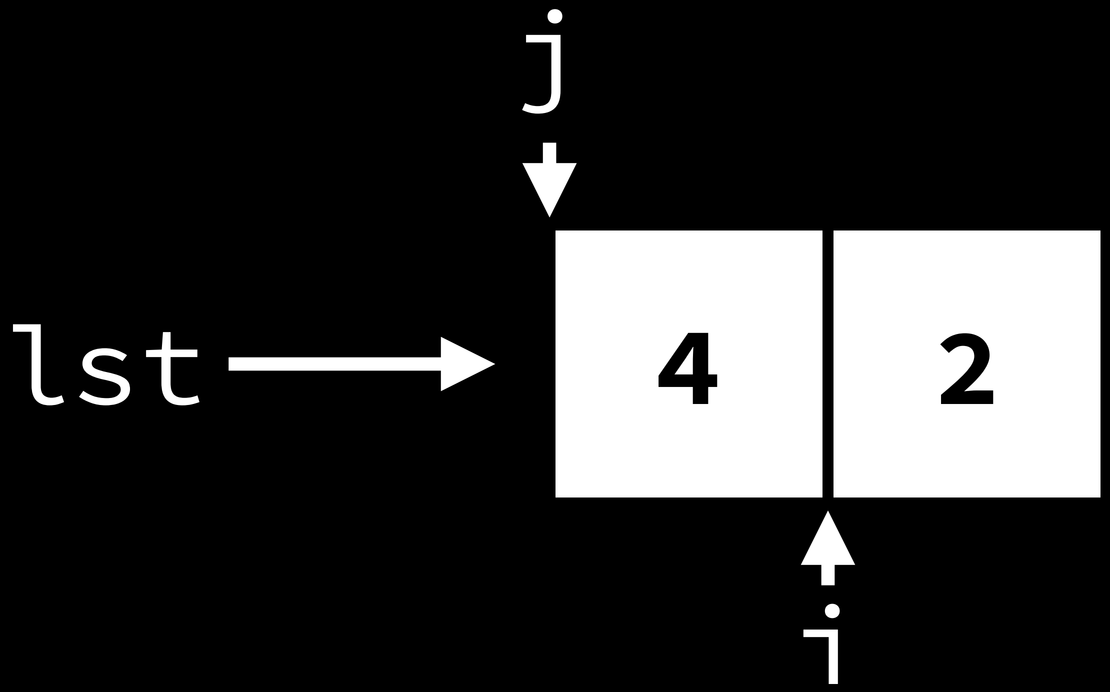
```
>>> next(i)
```



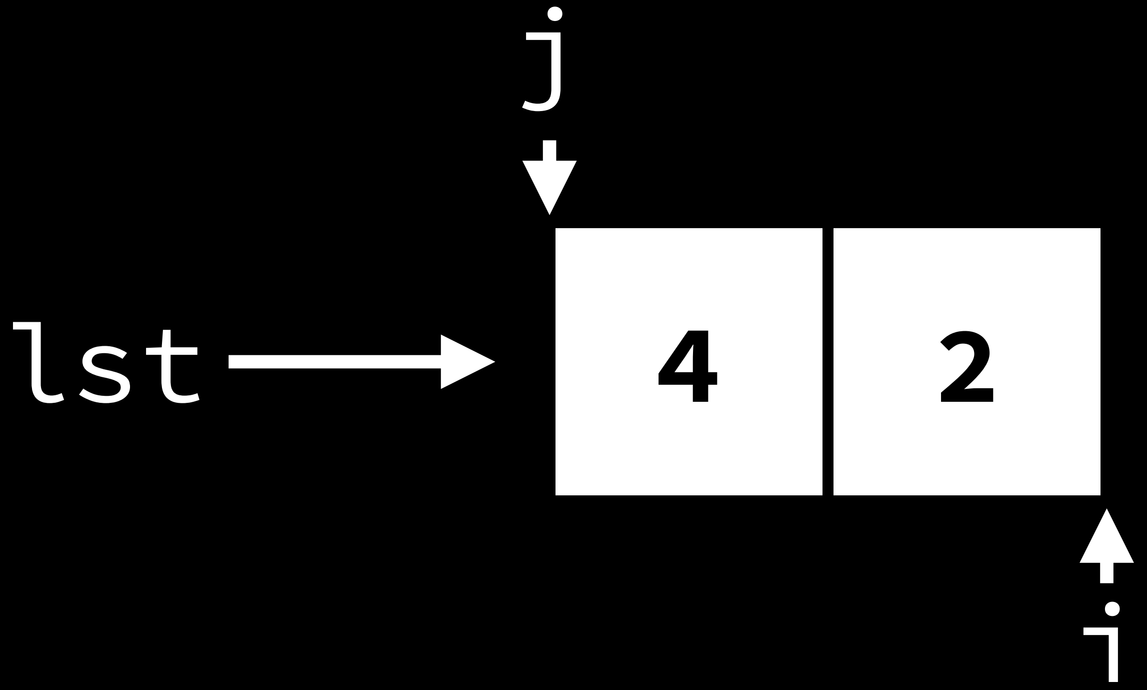
```
>>> next(i)
```



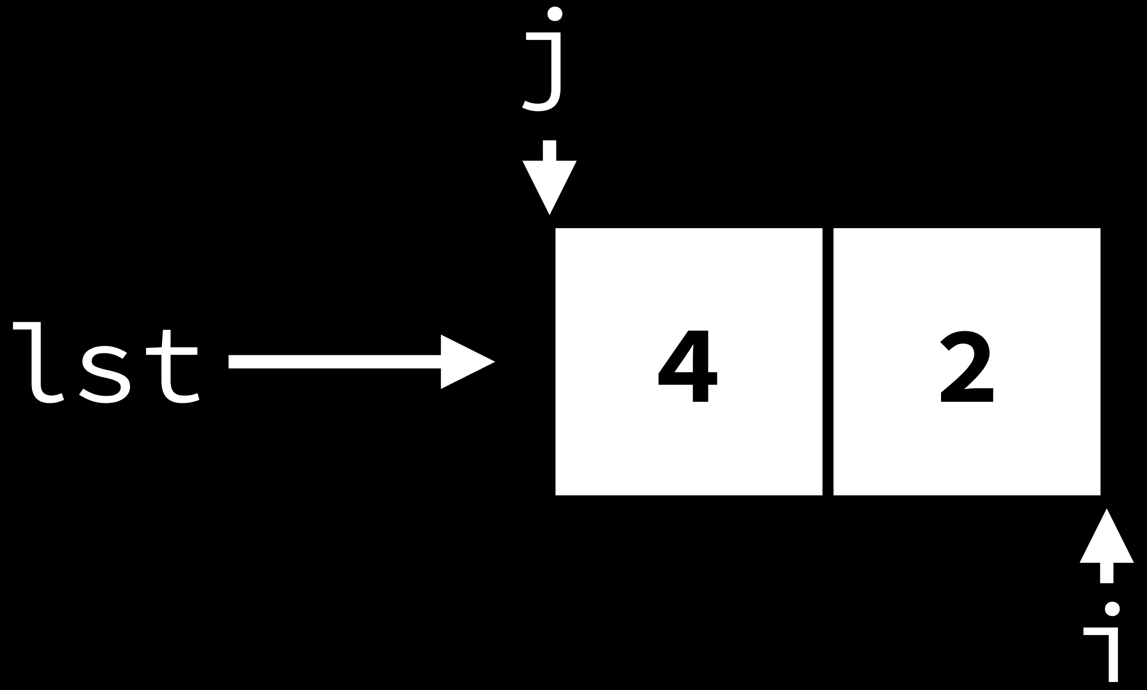
```
>>> next(i)
4
```



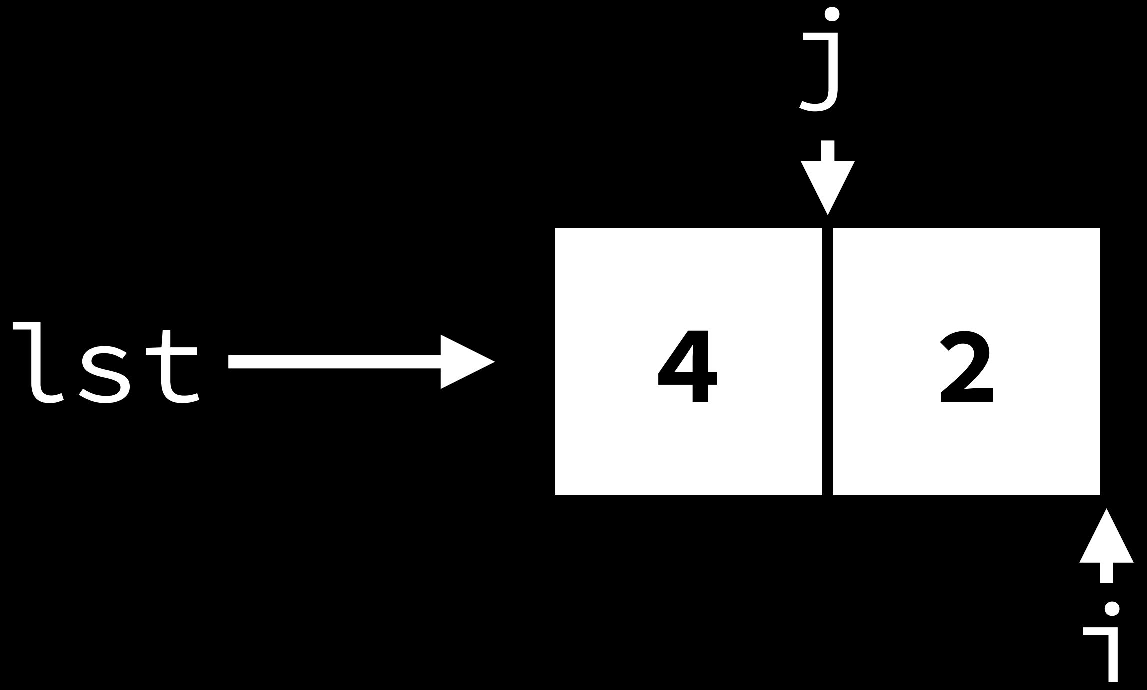
```
>>> next(i)
2
```



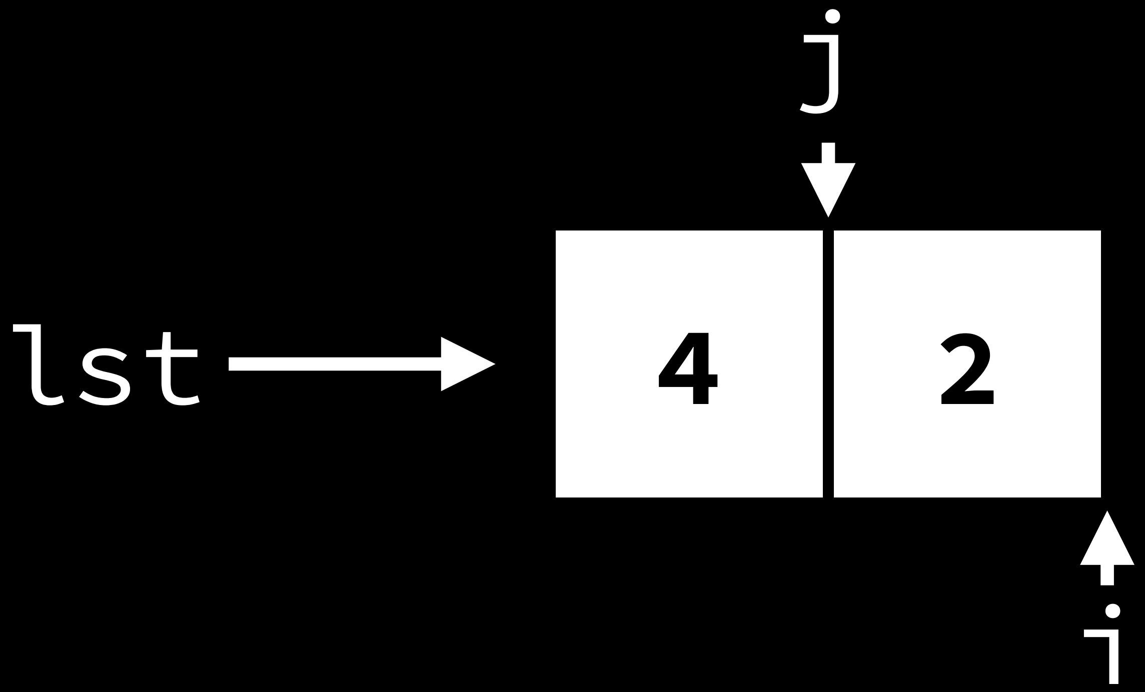
```
>>> next(j)
```



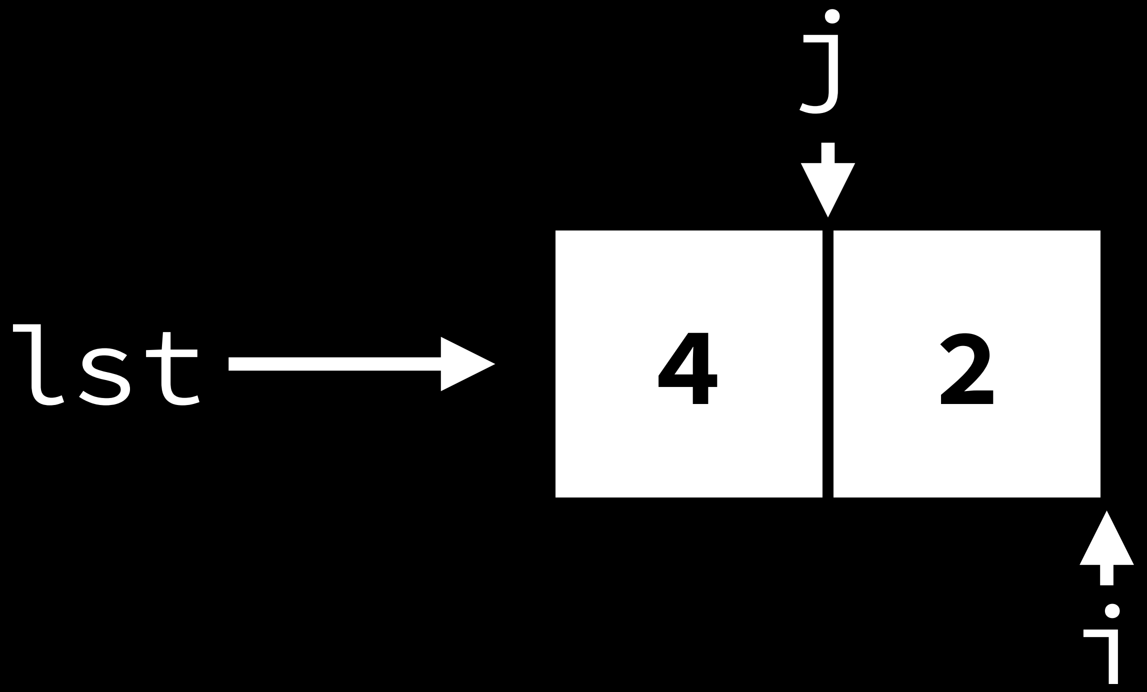
```
>>> next(j)
```



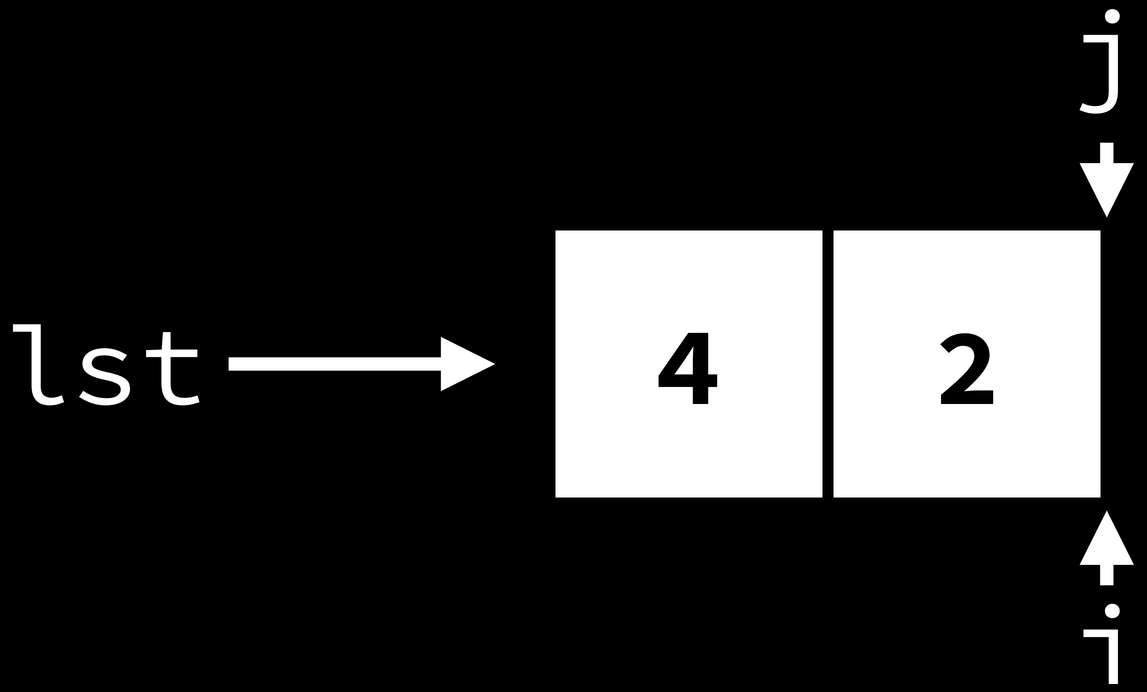
```
>>> next(j)
4
```



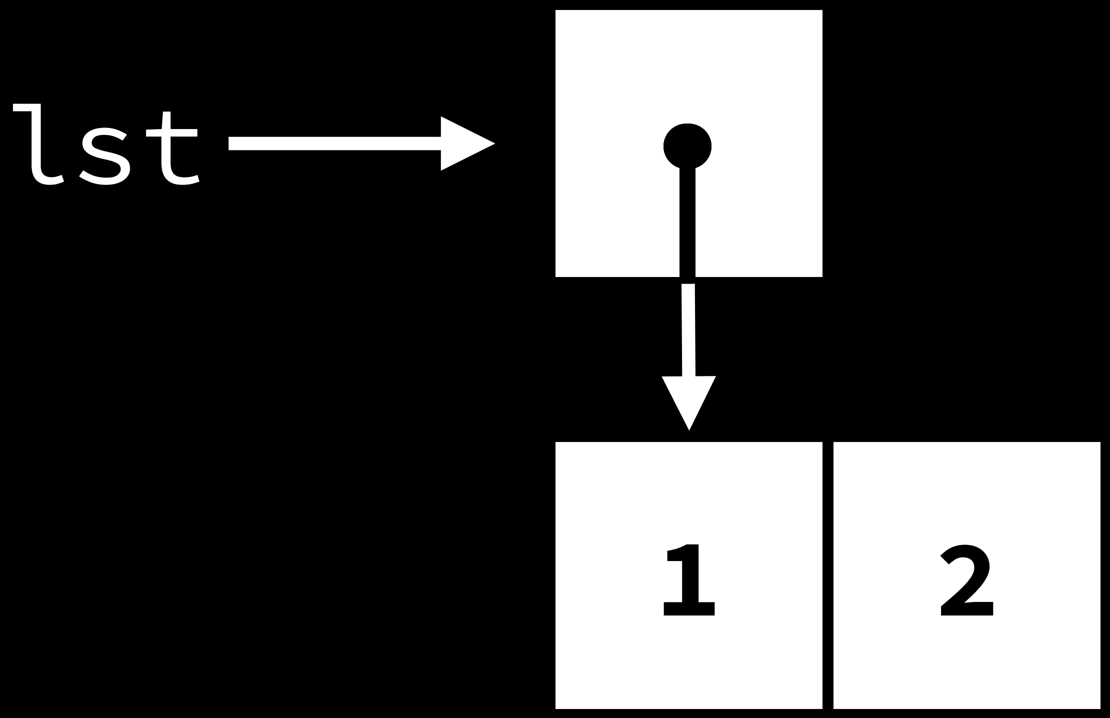
```
>>> next(i)
StopIteration
```



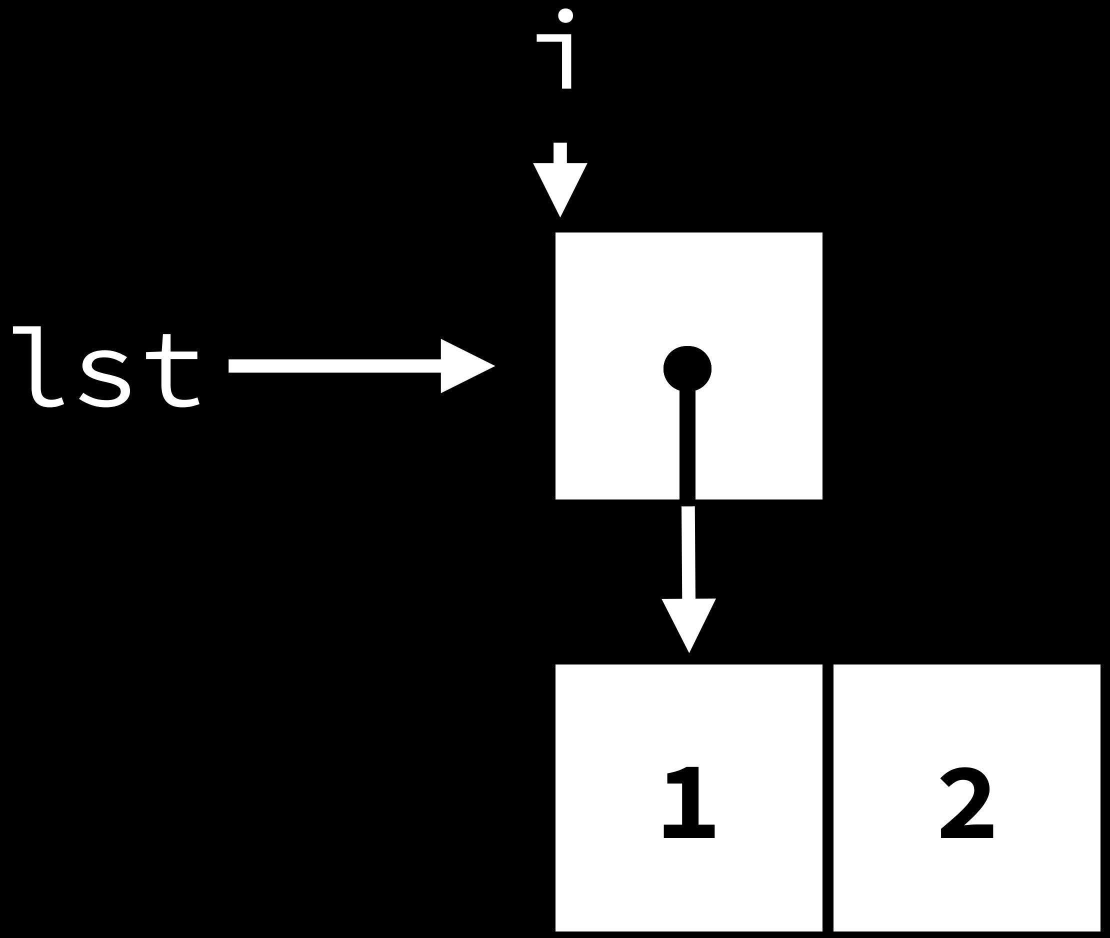
```
>>> next(j)
2
```



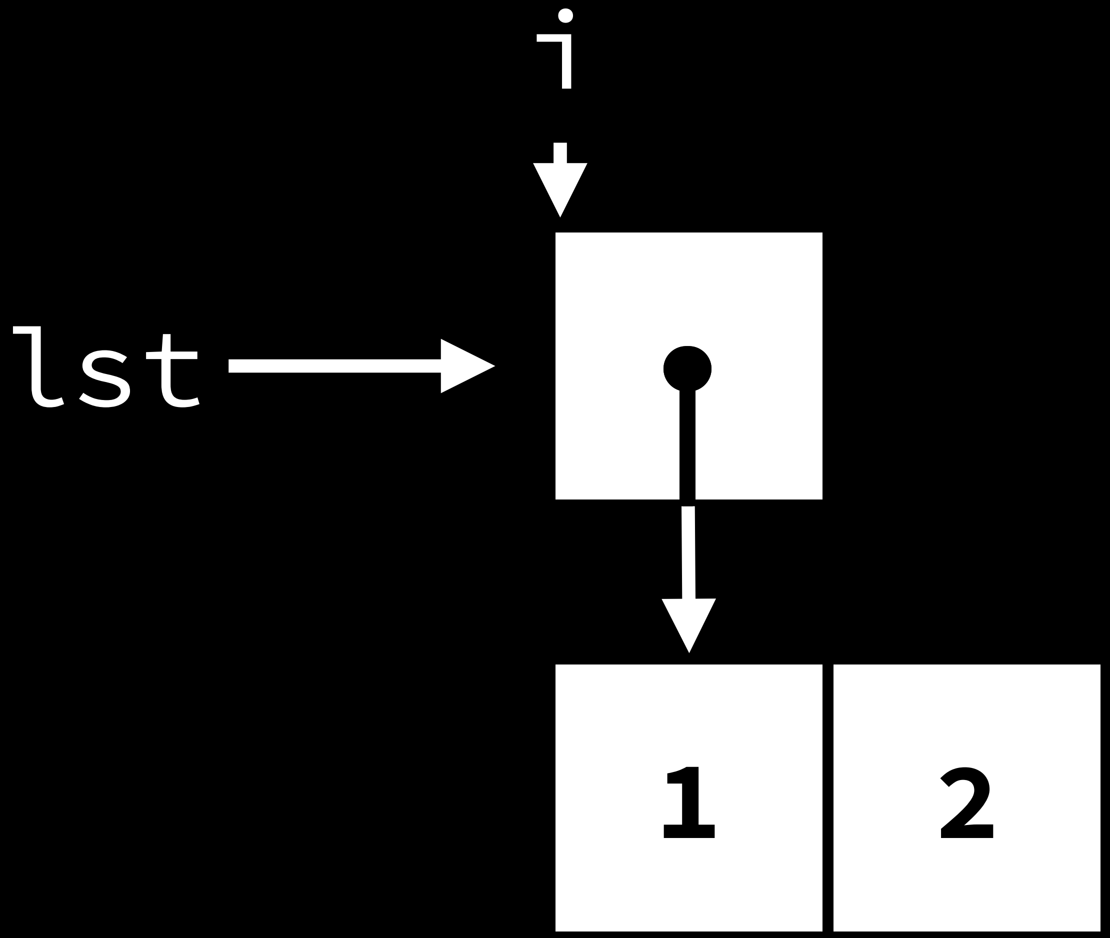
```python
lst = [[1, 2]]
```



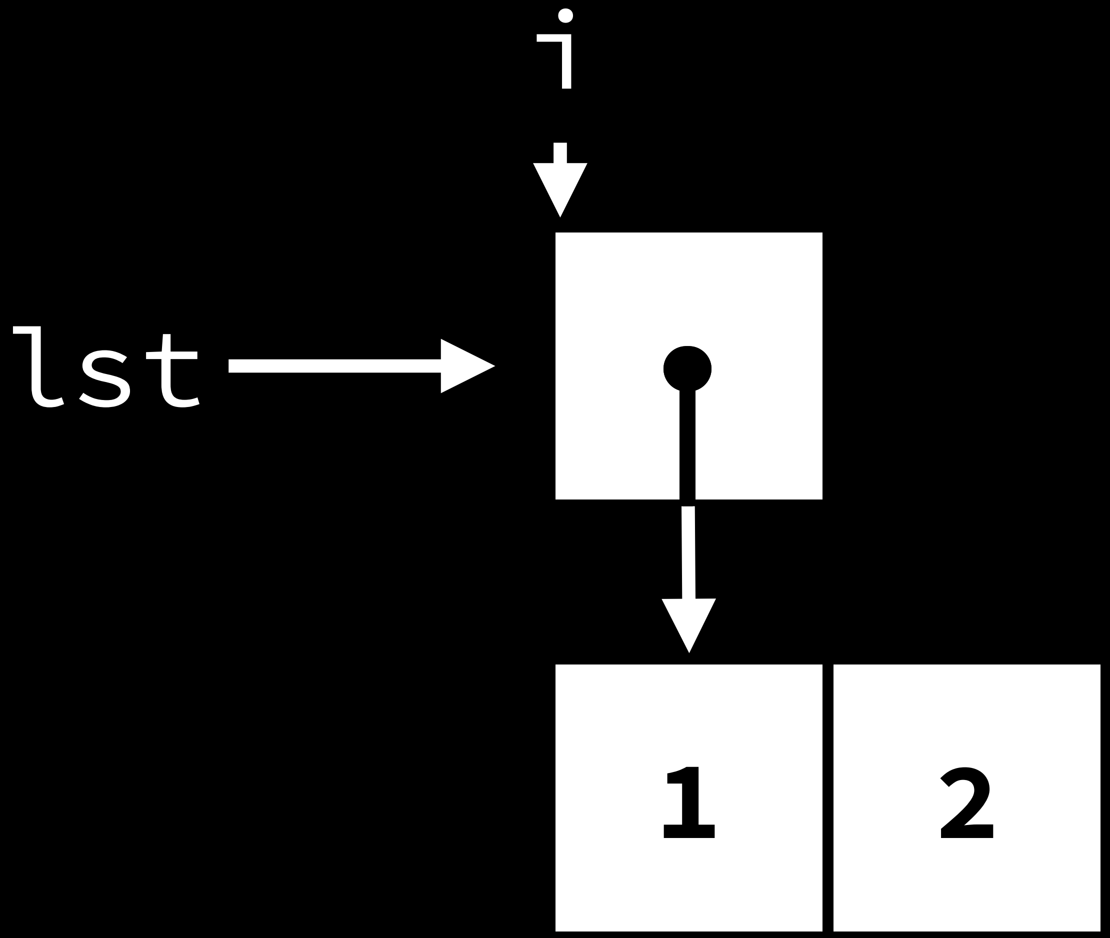
```
i = iter(lst)
```



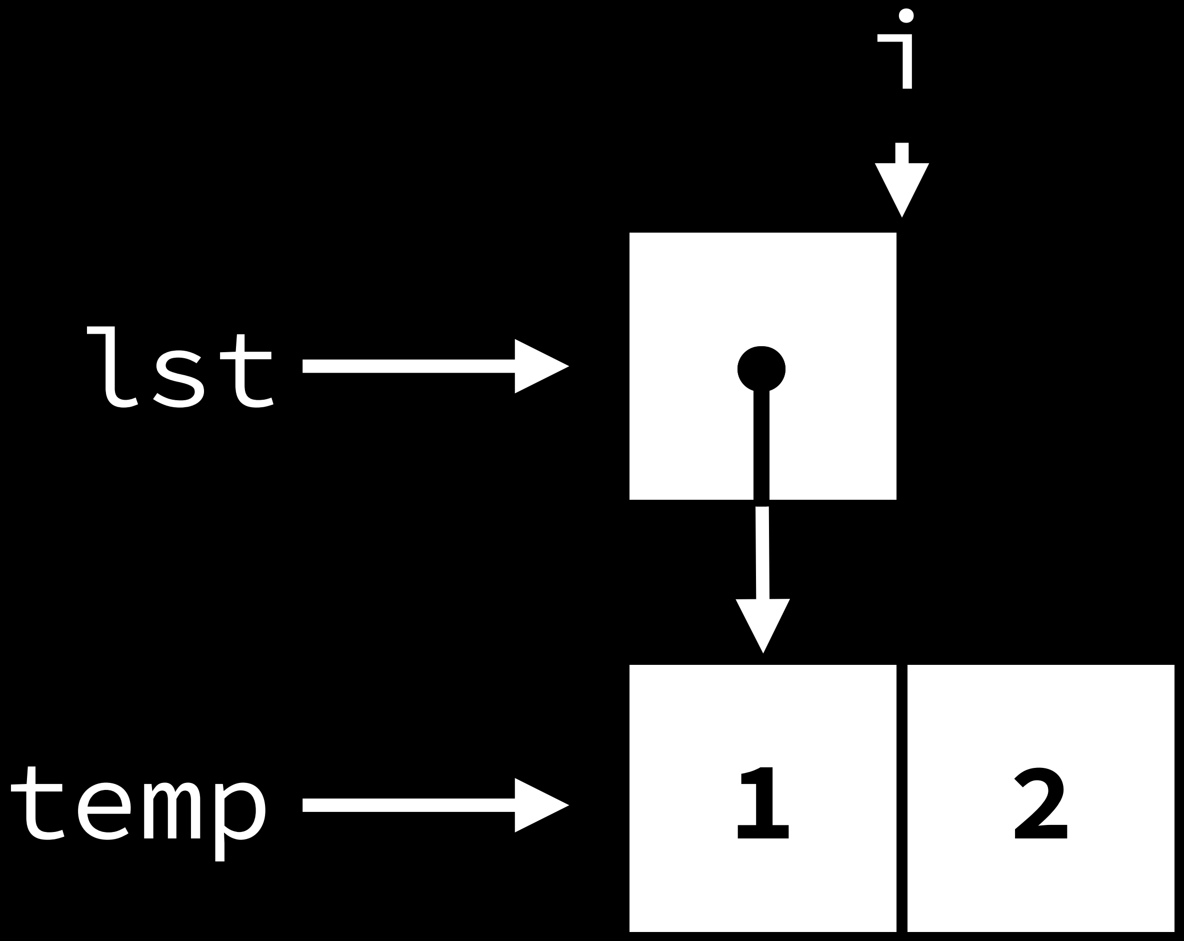
```
j = iter(next(i))
```



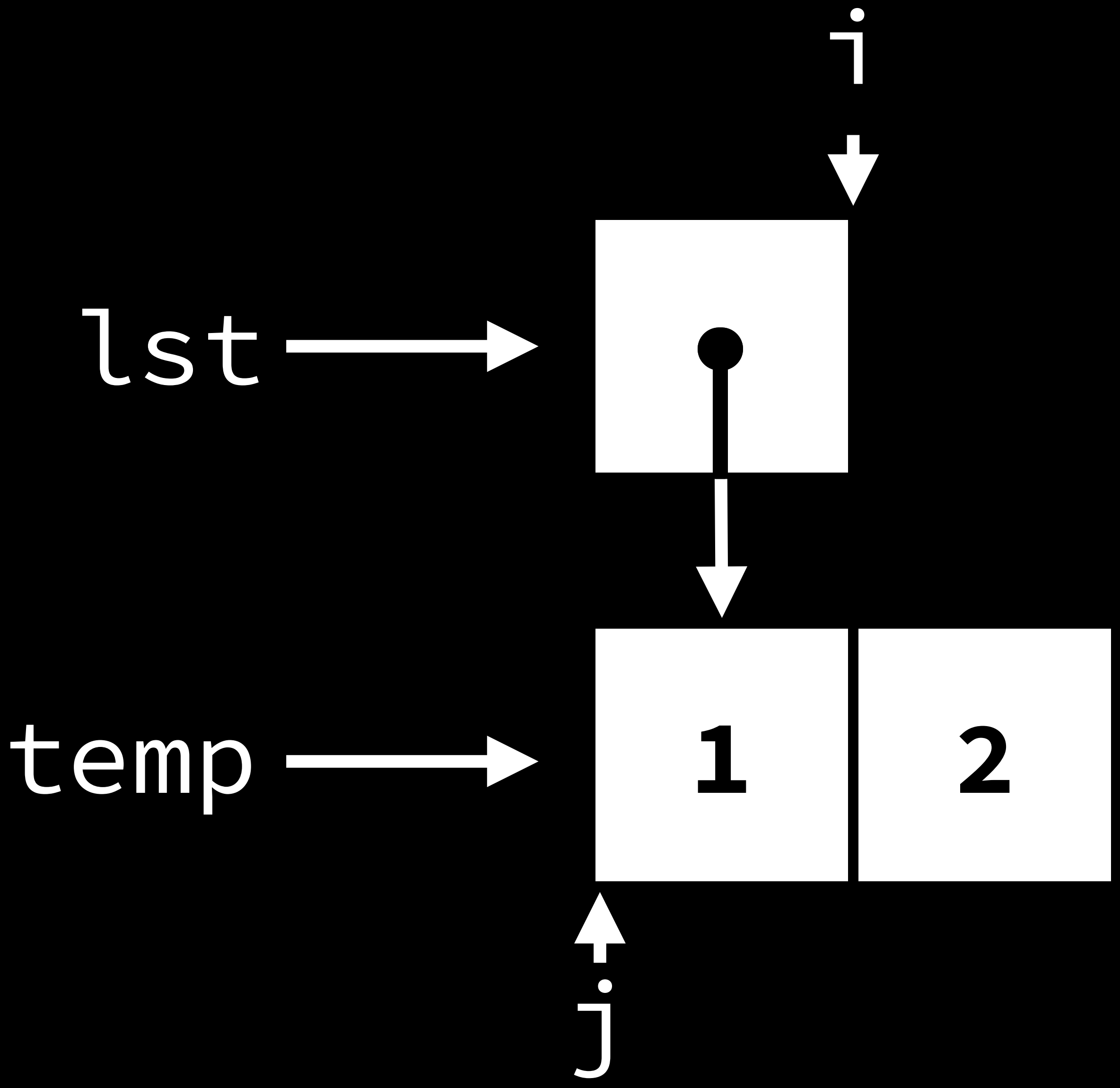
```
temp = next(i)
j = iter(temp)
```



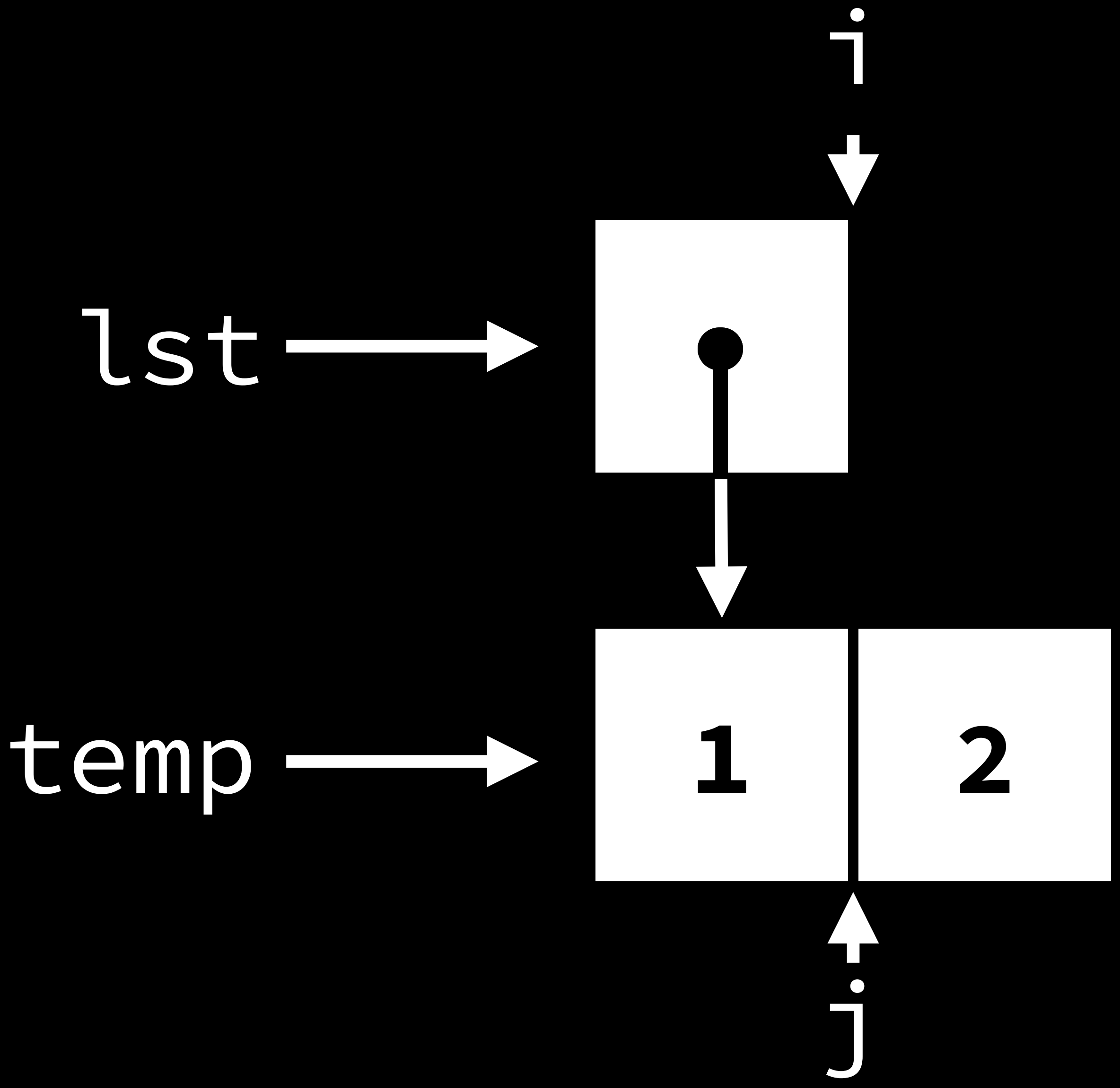
```
temp = next(i)
j = iter(temp)
```



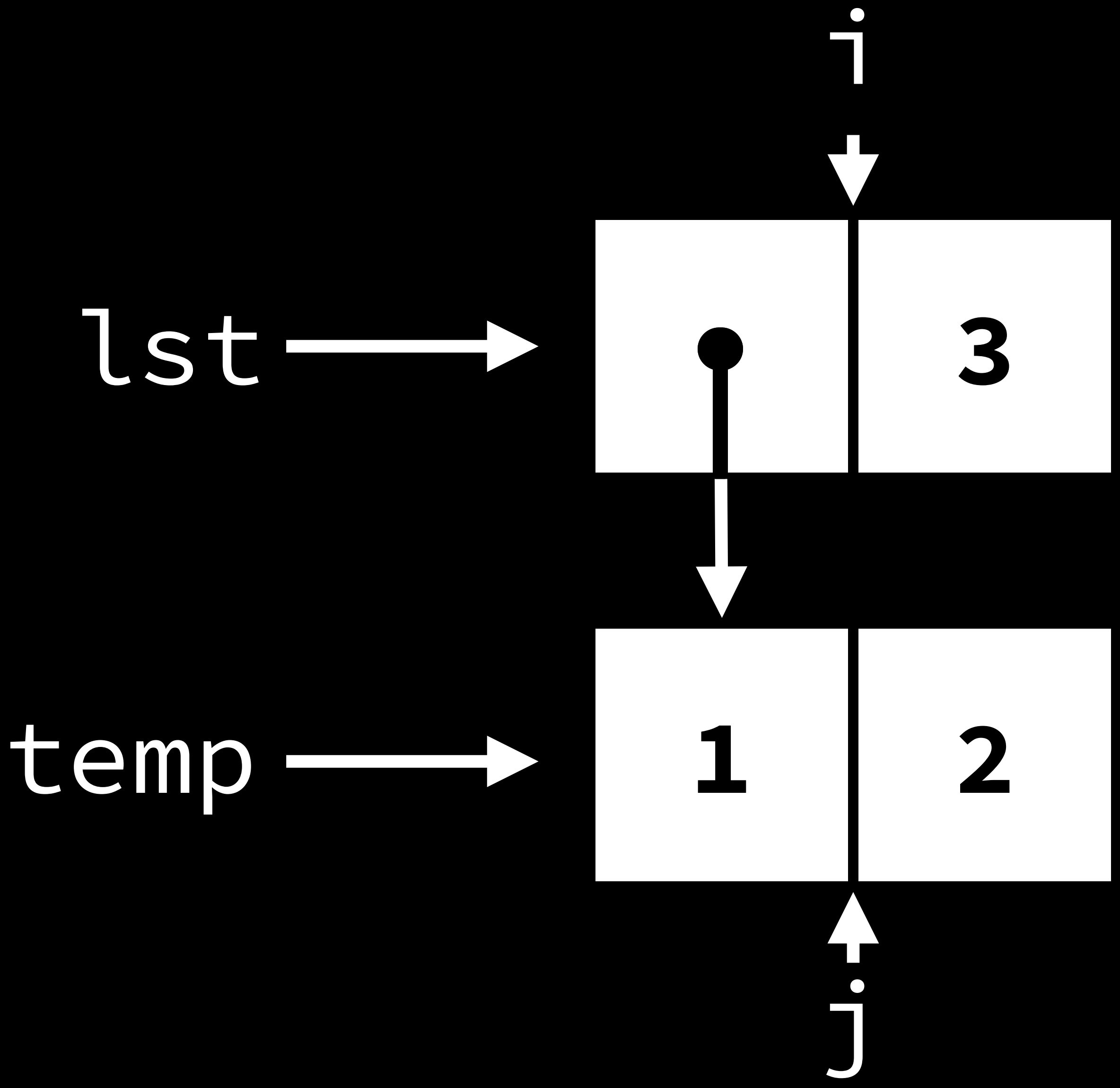
```
temp = next(i)
j = iter(temp)
```



`next(j)`



`lst.append(3)`



Generators

- **Generator functions** return a generator when called
- A **generator** is a special iterator

```python
def gen_naturals():
    print("Entered")
    current = 0
    while True:
        print("Before yield")
        yield current
        print("After yield")
        current += 1
```

```python
def gen_naturals():
    print("Entered")
    current = 0
    while True:
        print("Before yield")
        yield current
        print("After yield")
        current += 1
```

```python
def gen_naturals():
    print("E")
    current = 0
    while True:
        print("B")
        yield current
        print("A")
        current += 1
```

```
>>> gen =
gen_naturals()
```

```python
def gen_naturals():
    print("E")
    current = 0
    while True:
        print("B")
        yield current
        print("A")
        current += 1
```

```
>>> gen =
gen_naturals()
>>> next(gen)
E
B
0
```

```python
def gen_naturals():
    print("E")
    current = 0
    while True:
        print("B")
        yield current
        print("A")
        current += 1
```

$\longrightarrow$ (at `print("A")`)

```
>>> gen =
gen_naturals()
>>> next(gen)
E
B
0
>>> next(gen)
A
B
1
```

```python
def gen_naturals():
    print("E")
    current = 0
    while True:
        print("B")
        yield current
        print("A")
        current += 1
```

```
>>> gen =
gen_naturals()
>>> next(gen)
E
B
0
>>> next(gen)
A
B
1
```

```python
def gen_naturals():
    print("E")
    current = 0
    while True:
        print("B")
        yield current
        print("A")
        current += 1
```

```
>>> gen =
gen_naturals()
>>> next(gen)
E
B
0
>>> next(gen)
A
B
1
```

```python
def gen_naturals():
    print("E")
    current = 0
    while True:
        print("B")
        yield current
        print("A")
        current += 1
```

Streams

Being lazy pays off

Lists, but better

Streams

New primitives

`cons-stream` allows us to delay evaluation of **`second`**

`cdr-stream` forces that evaluation