Linked Lists

Peter Chapin

Vermont Technical College

About Dynamic Memory Allocation

```
#include <stdlib.h>

int *p = (int *)malloc( sizeof(int) );
```

C vs C++ vs Java

C

```
Widget *w = (Widget *)malloc( sizeof(Widget) );
initialize_widget( w, other, arguments, as, needed );
free( w );
```

Separate “constructor” function

C++

```
Widget *w = new Widget( other, arguments, as, needed );
delete w;
```

Invokes constructor method in class Widget

Java

```
Widget w = new Widget(other, arguments, as, needed);
```

C vs C++ vs Java

C

```
Widget *w1 = (Widget *)malloc( sizeof(Widget) );
Widget w2;
```

Uninitialized Widget object

C++

```
Widget *w1 = new Widget( other, arguments, as, needed );
Widget w2;
```

Initialized with default constructor (if available)

Java

```
Widget w1 = new Widget(other, arguments, as, needed);
Widget w2;
```

Not a real object. Initialized as a null reference.

Arrays

- An array is a sequence of items laid out in contiguous memory.
  - Each item is physically adjacent to the previous (and next).
  - Each item has the same size (and typically has the same type).
- Items are accessed using an integer "index" value.
  - Let *a* be the base address of an array. Let *s* be the size of each item. The address of item *i* is given by `a + i*s`.
- Using C notation:

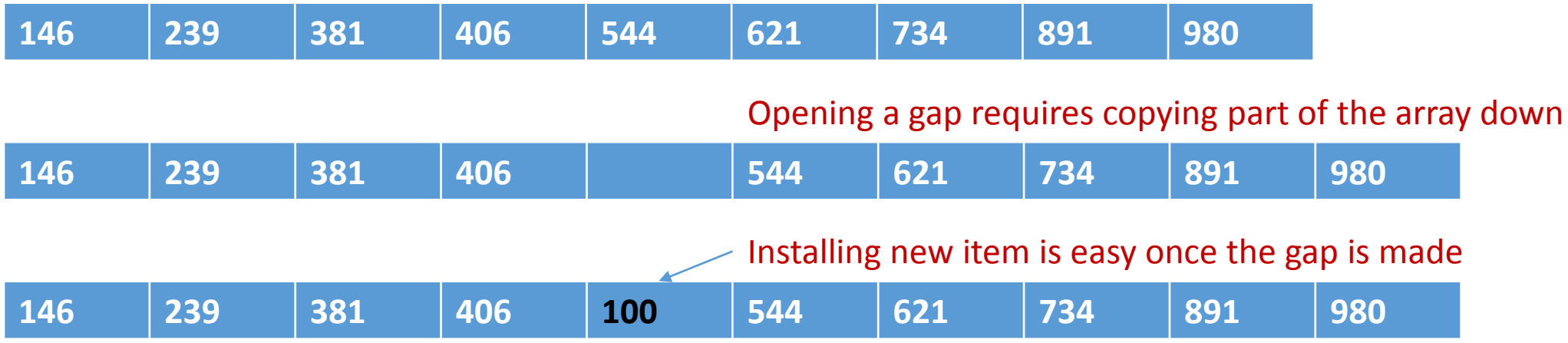
Compiler automatically applies scale factor *s*

```
*(a + i) == a[i]
```

Name of array is pointer to first element

Arrays

- Time to access an item is independent of the array's size
  - Said to be *constant time*
- However, inserting an item requires shifting down the array's contents. The time required is proportional to the array's size
  - Said to be *linear time*



Arrays

- In C arrays can't be resized after they are created
  - Opening a gap can't be done unless you have "extra" space pre-allocated.
  - Must maintain a record of how much space is actually being used
  - C strings (for example), mark the end with a null character.

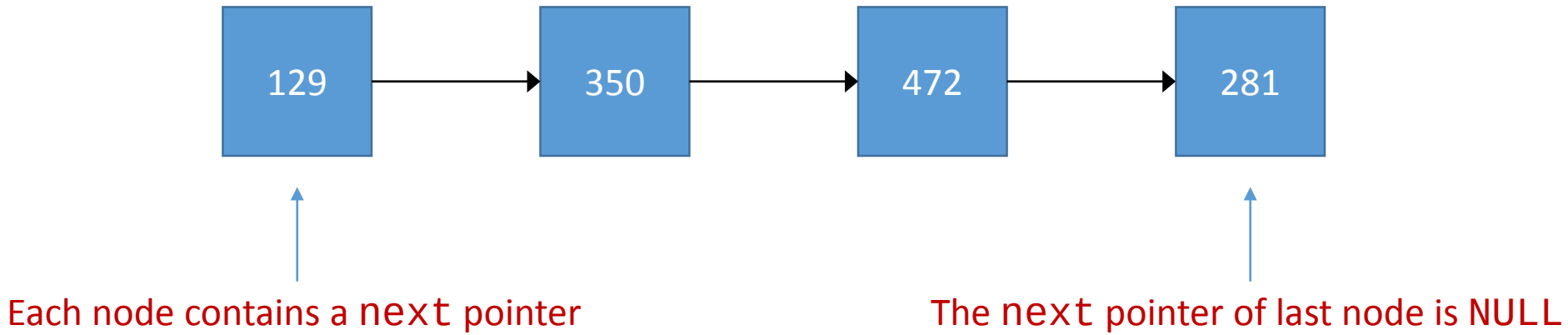
```
void insert_into( char *s, size_t position, char new_item )
{
    char *p = strchr( s, '\0' );
    while( p - s > position ) {
        *(p + 1) = *p;
         p--;
    }
    *(p + 1) = *p;
    *p = new_item;
}
```

Function assumes sufficient space exists

Linear time loop

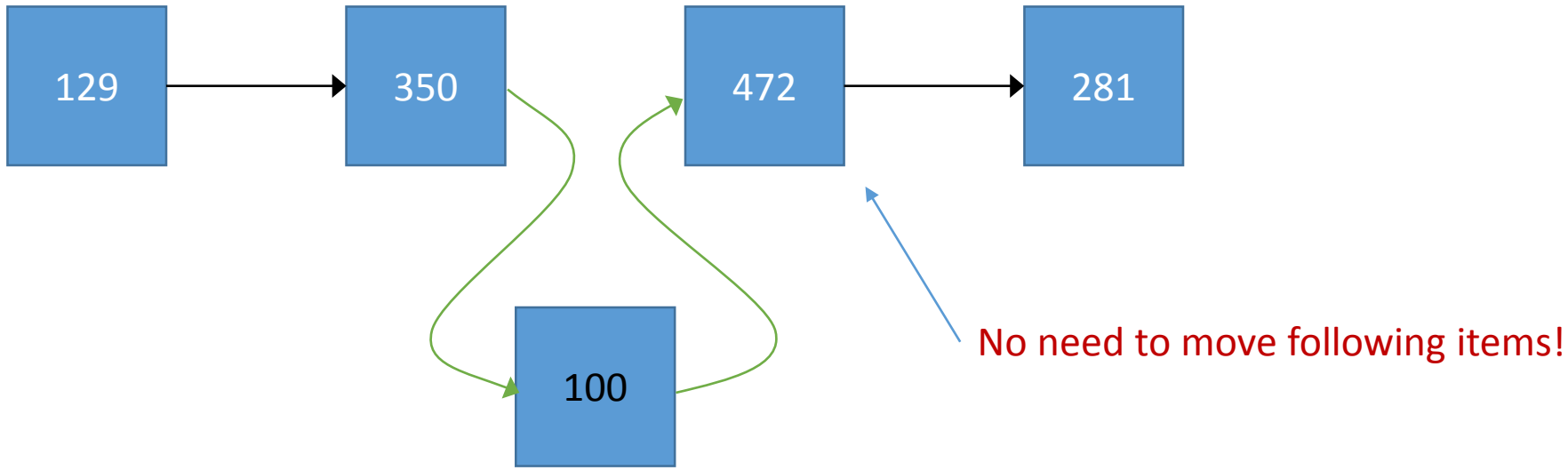
Linked Lists

- A sequence of items where each item is stored in its own *node*
  - Nodes are dynamically allocated and could be anywhere in the heap
  - Nodes contain a pointer to the next node in the sequence (a *link*)

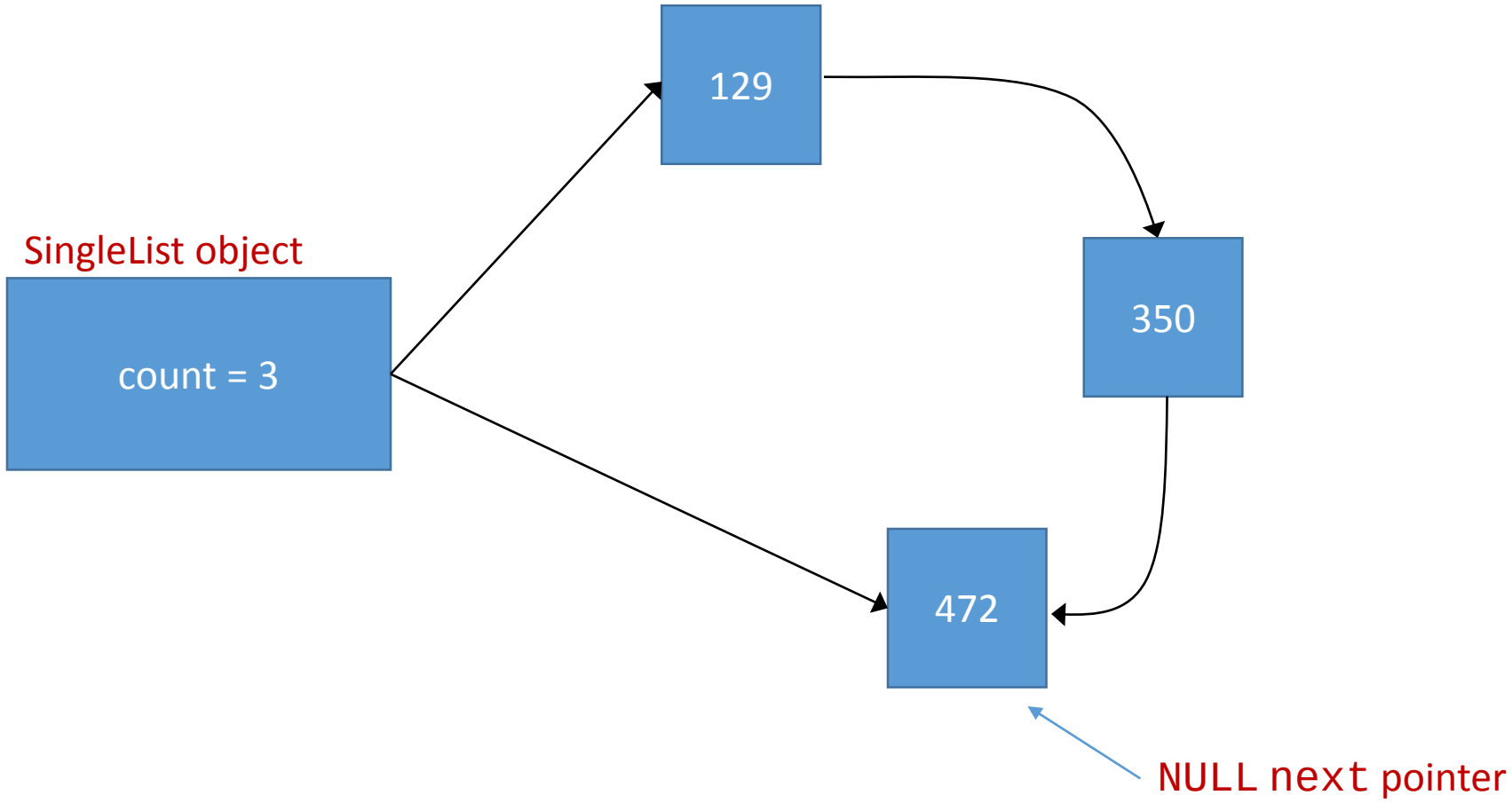


Linked Lists

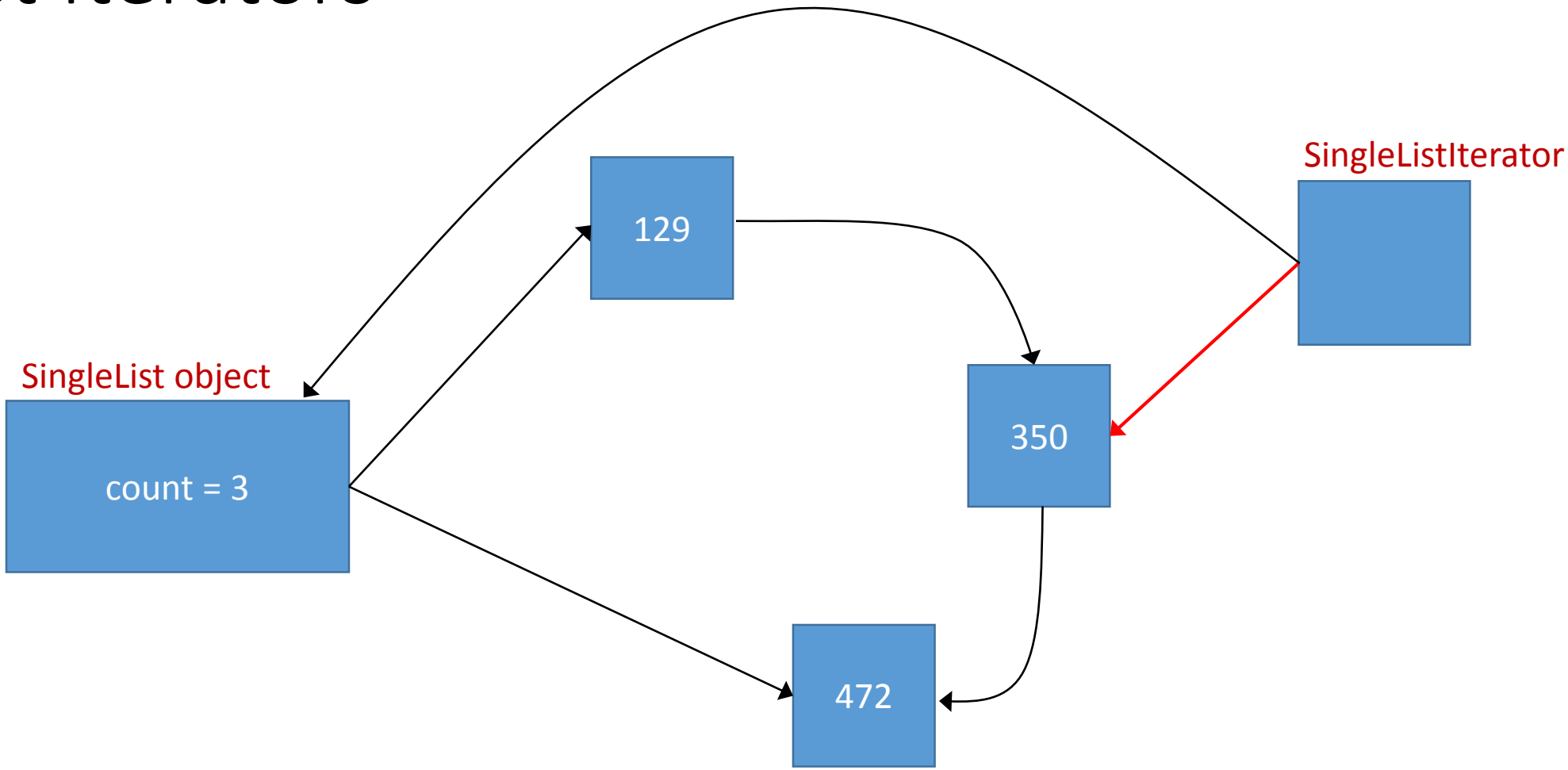
- Accessing an item requires accessing the previous item
  - Must use the previous item's next pointer to locate the next item (**linear time**)
- Inserting an item requires allocating it and then adjusting pointers



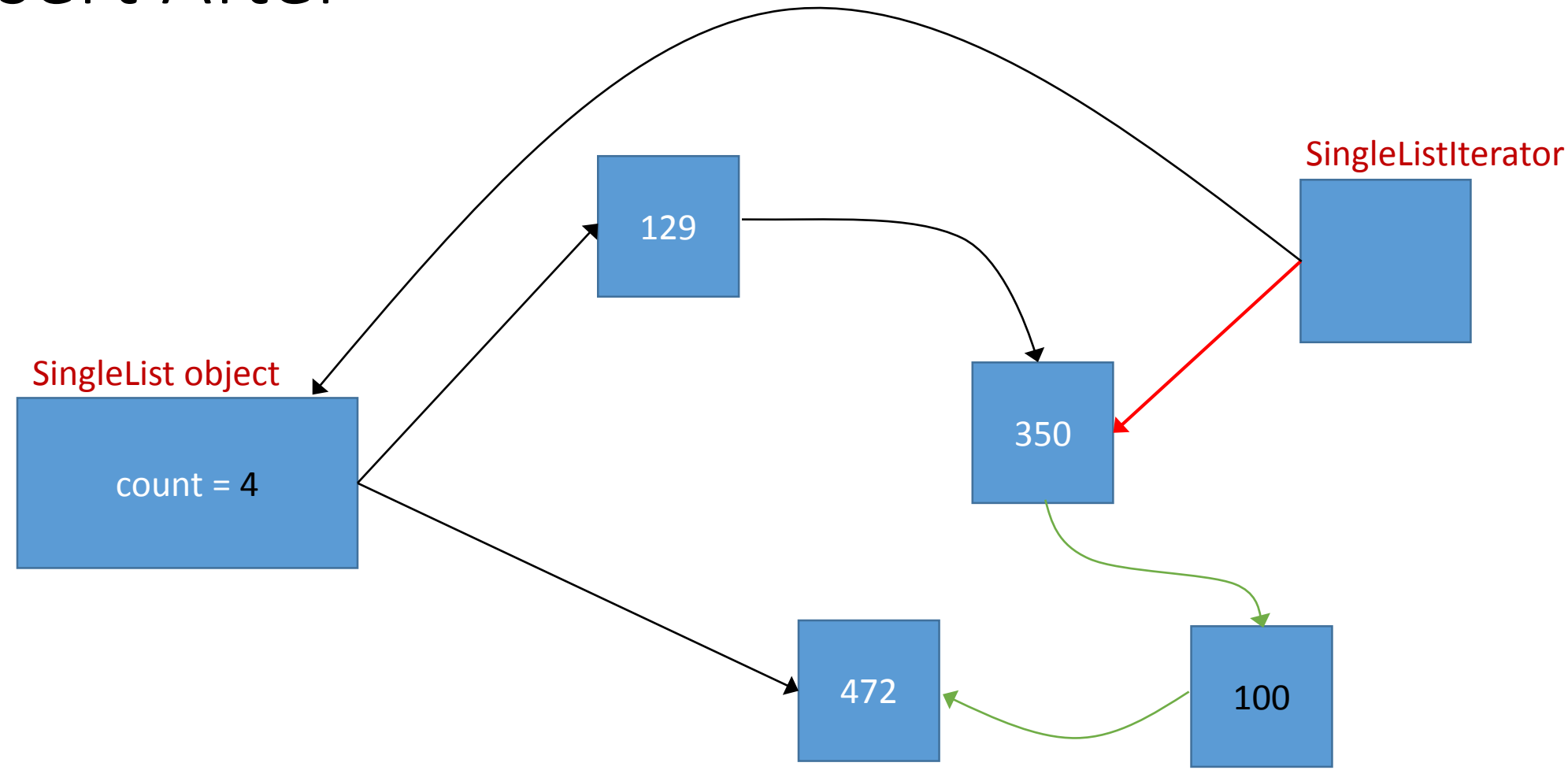
Information Hiding



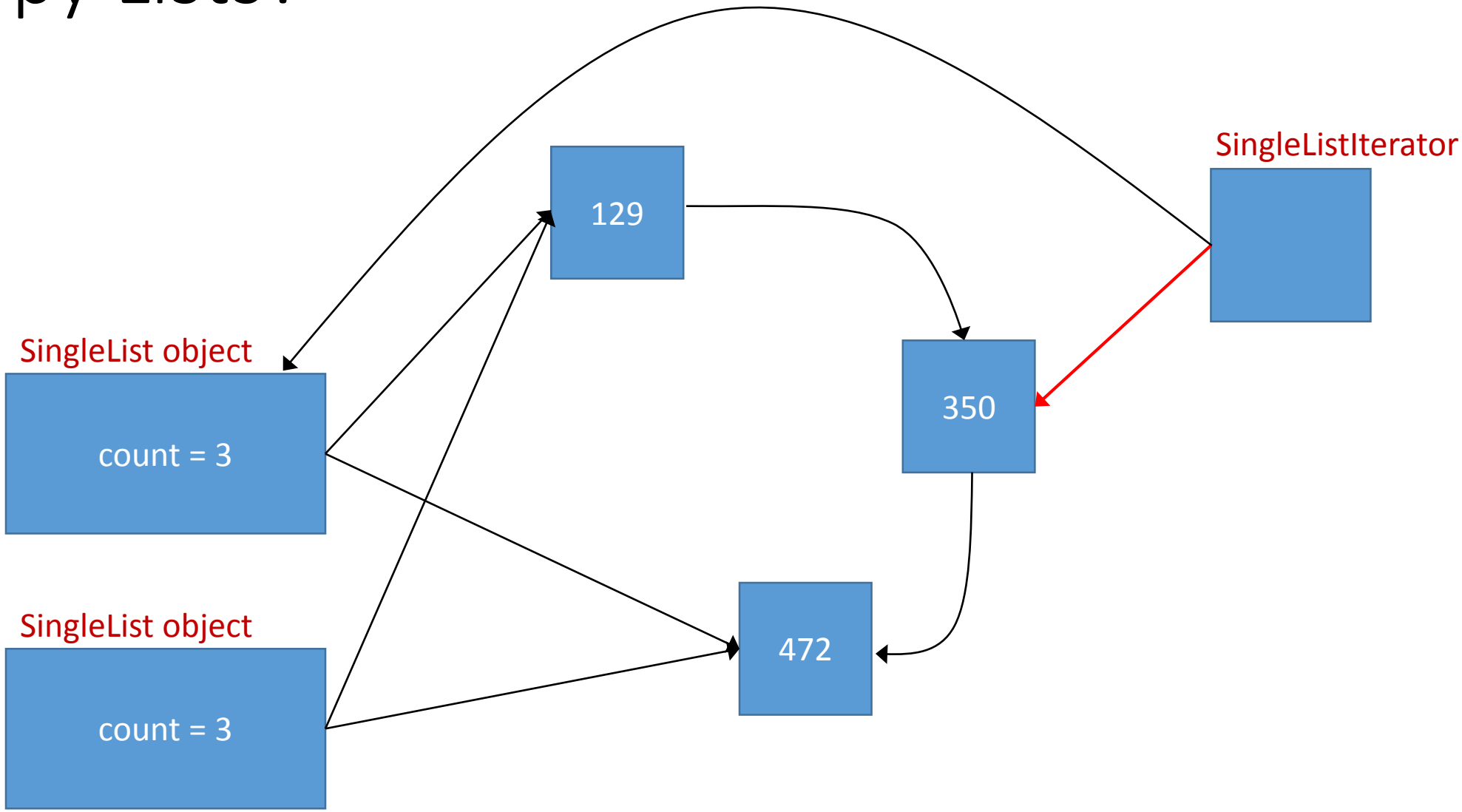
List Iterators



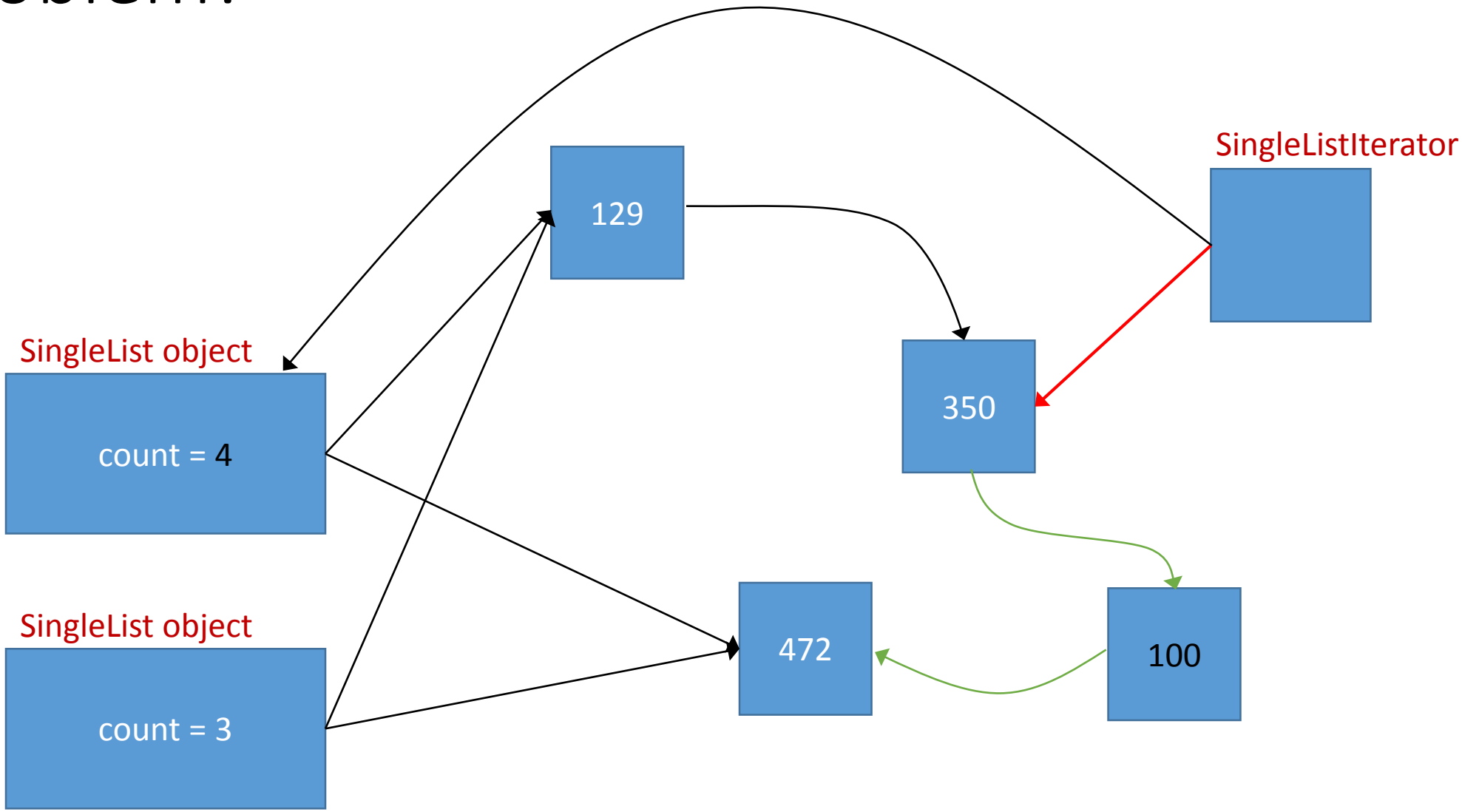
Insert After



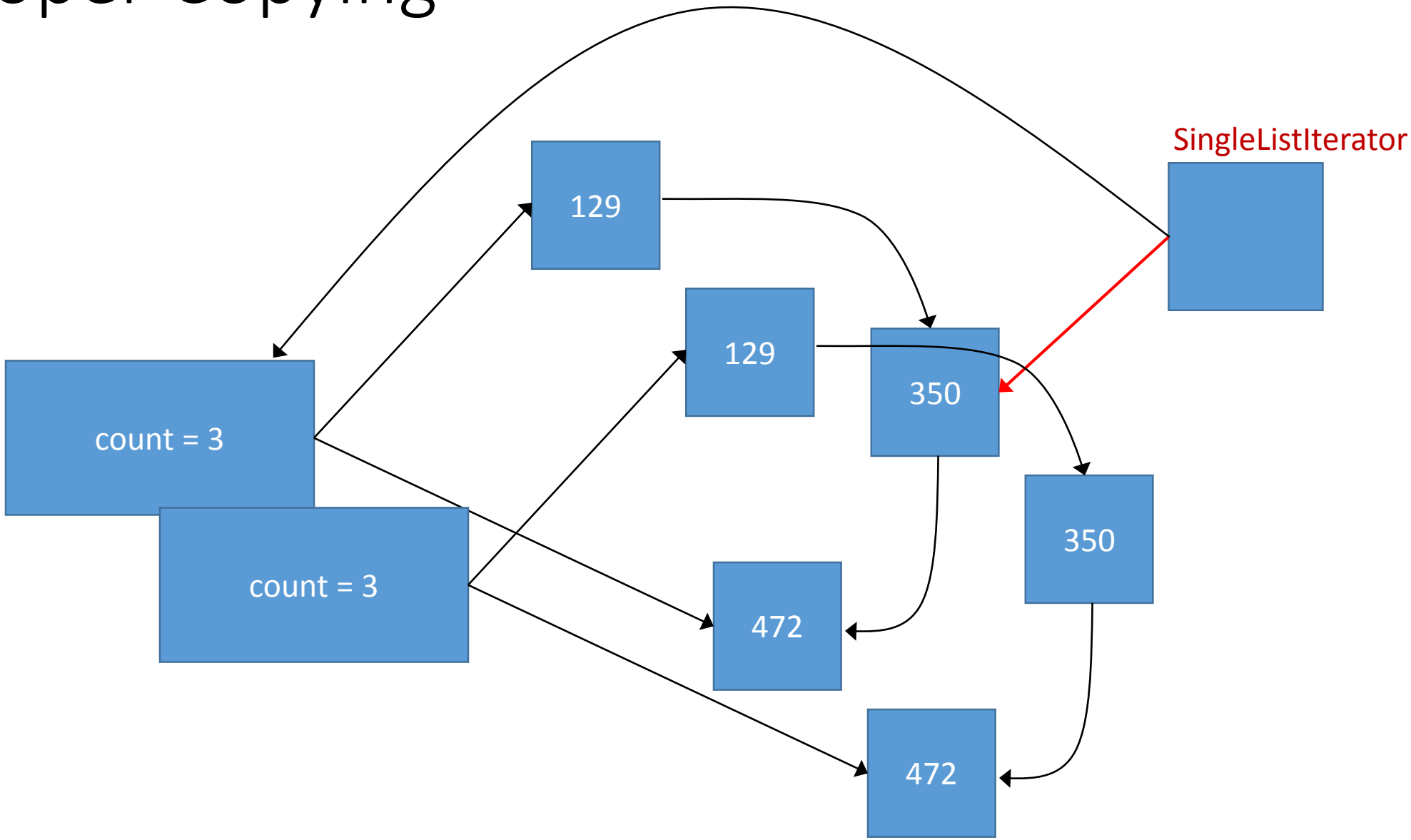
Copy Lists?


SingleListIterator
129
SingleList object
count = 3
350
SingleList object
count = 3
472

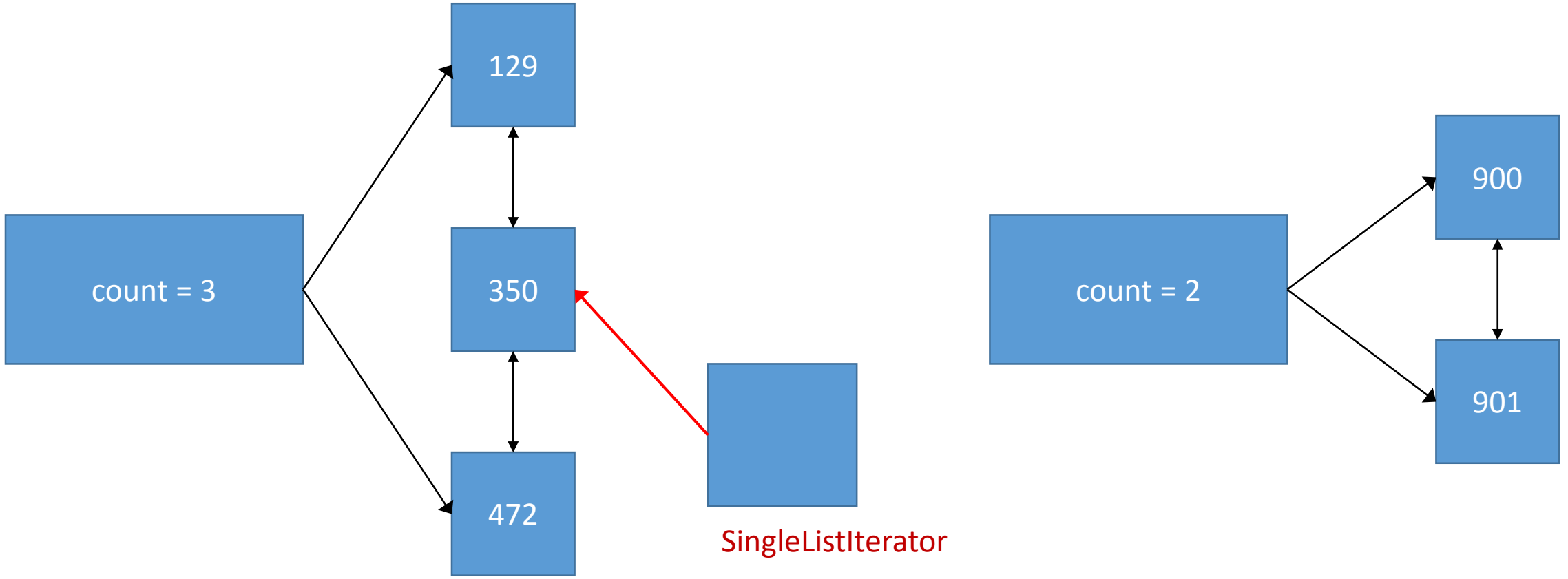
Problem!


SingleListIterator
129
SingleList object
count = 4
350
SingleList object
count = 3
472
100

Proper Copying



DoubleList_splice_before



DoubleList_splice_before