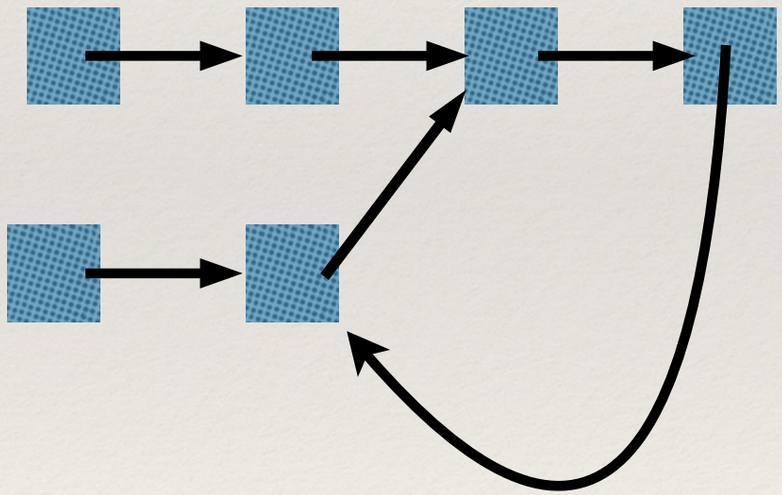
CS 61BL Lab 6

Ryan Purpura

What's Wrong With Yesterday's Linked Lists?

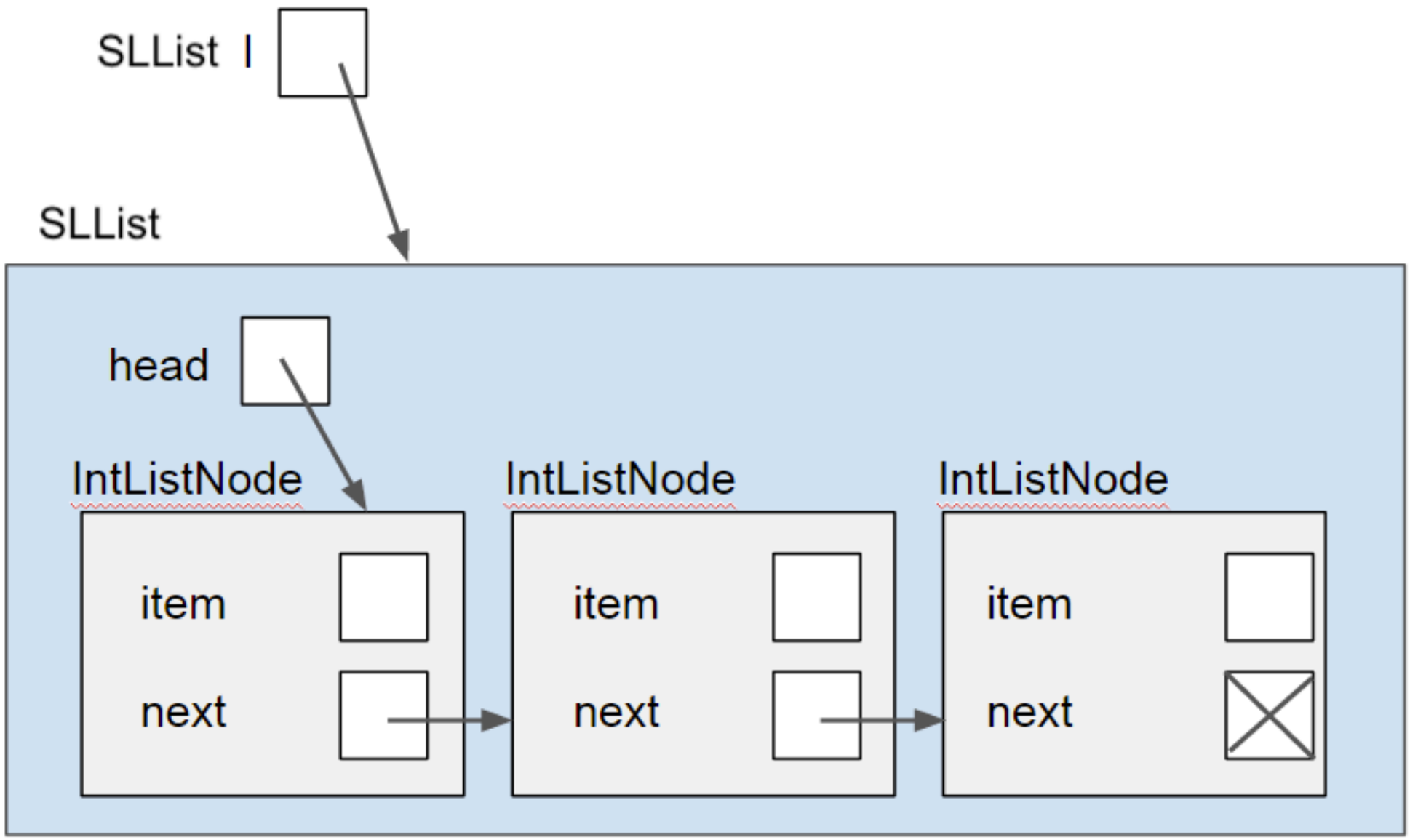
- What we want from a list is a linear collection of elements -- linked lists are no exception!



Encapsulation

- Previously, the concept of a node and a list was blurred.
- We will improve this with a separate SLList (Singly-linked list) class and a Node class.
- The SLList will handle all of the logic regarding Nodes, and people who use the SLList class doesn't have to know anything about how it's implemented!

The Idea


SLList l
SLList
head
IntListNode
item
next
IntListNode
item
next
IntListNode
item
next

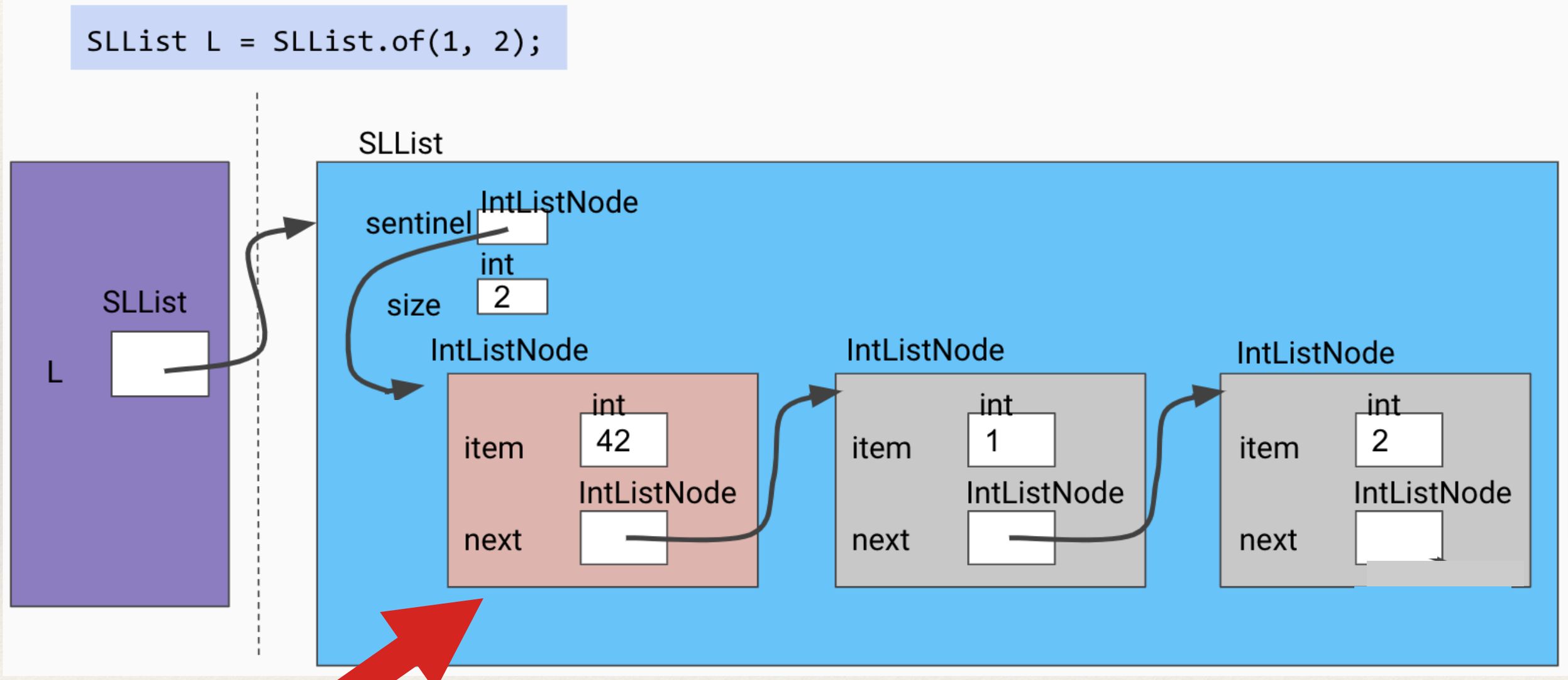
Null Checks

- Null checks suck
- When do we have null checks in SLList?
  - Checking if empty list (**`head`** will be null)
  - At the end of the list (the last node's **`next`** instance variable will be null)

Crazy Idea 1

- **Just don't let the list be empty!**
- Then head will never be null.
- To implement this, we insert a dummy node (contains no data for our list) at the beginning called a **sentinel node.** (The SLList will now have a reference to the sentinel node instead of **head**)
- The sentinel's **next** instance variable is the front of the list.

The Idea

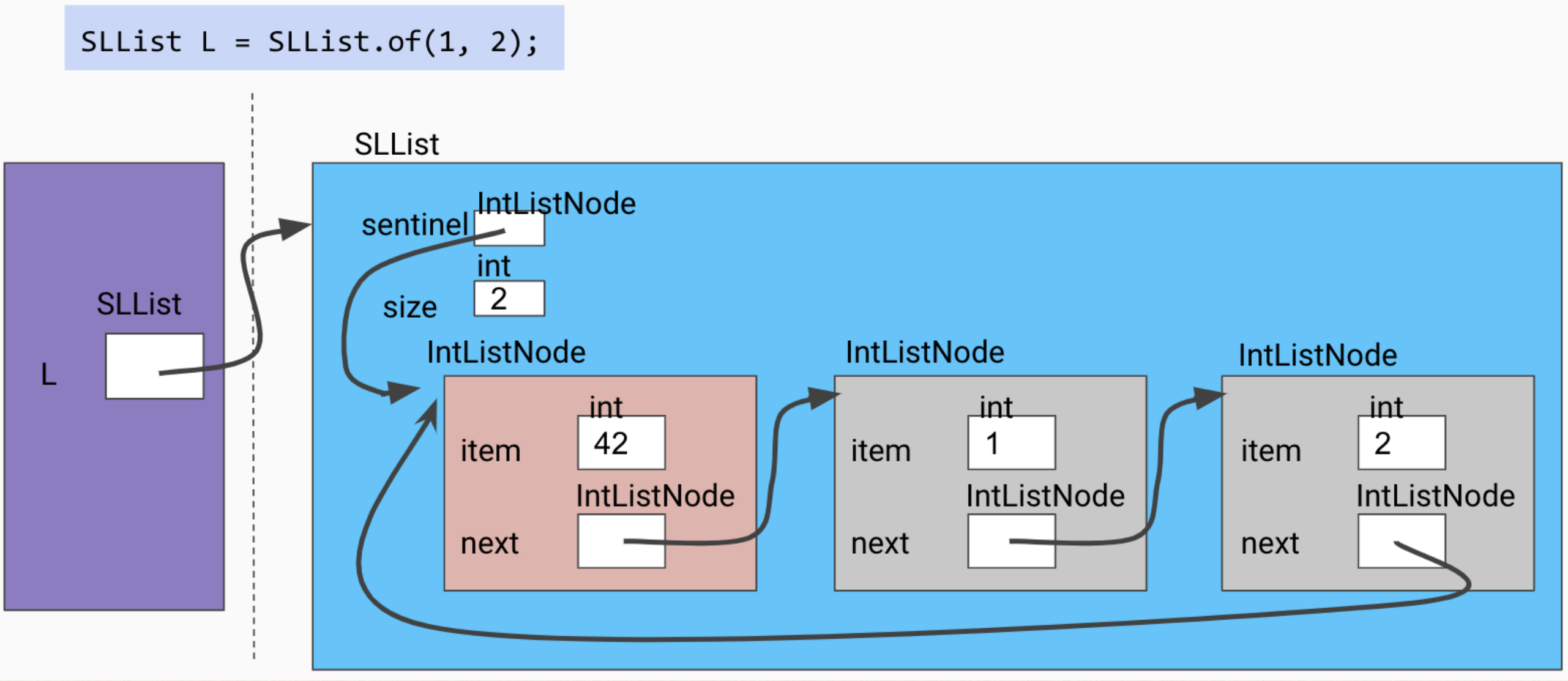


sentinel node

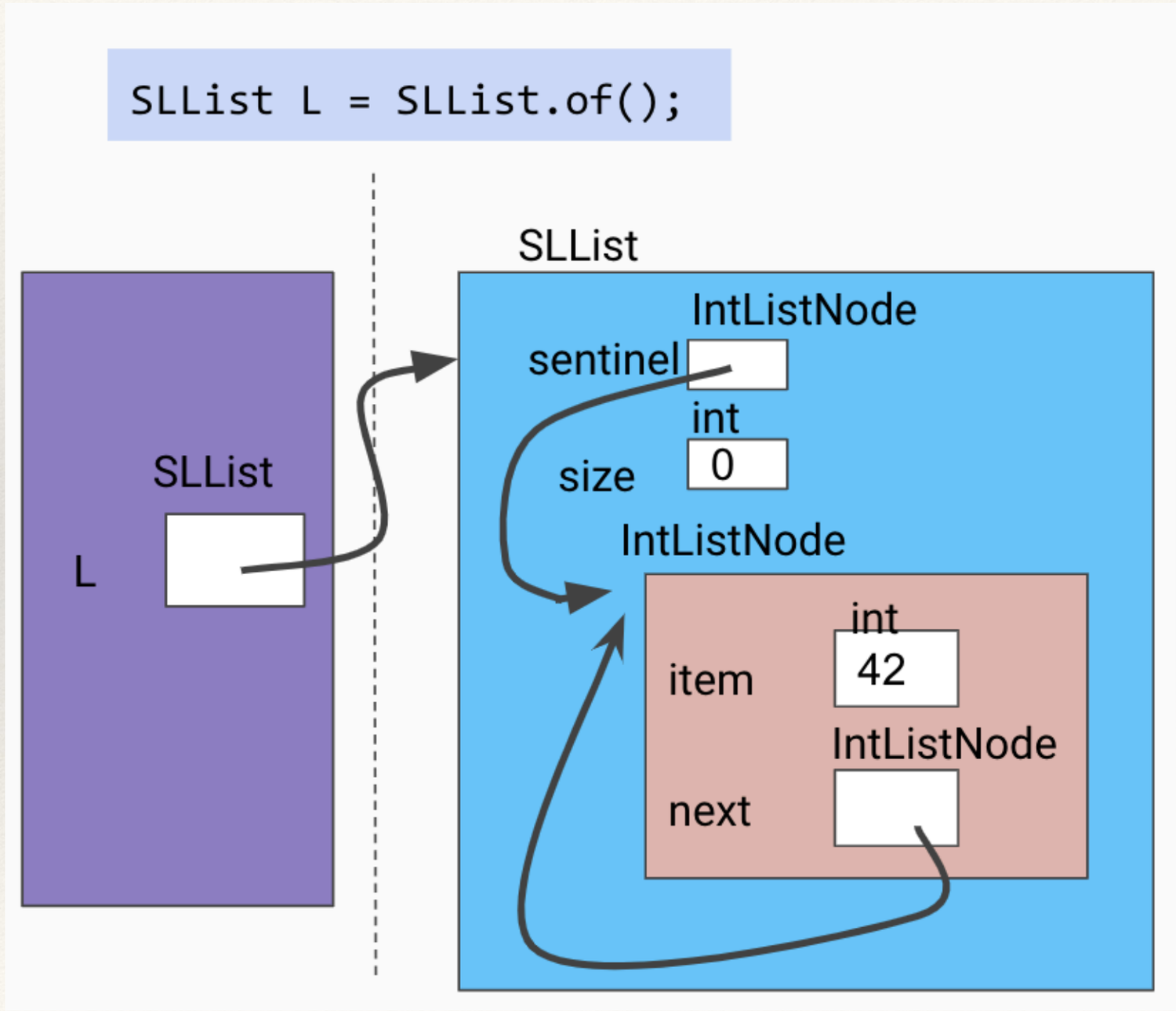
Crazy Idea 2

- Make the last element's next instance variable point at the sentinel.
- This eliminates the null pointer at the end of the list.

The Idea


SLList L = SLList.of(1, 2);
SLList
L
SLList
IntListNode
sentinel
int
size
2
IntListNode
item
int
42
IntListNode
next
IntListNode
item
int
1
IntListNode
next
IntListNode
item
int
2
IntListNode
next

What would an empty list look like?



Are the null checks gone now?

- ***How do we know that our linked list is empty?***
  - Before, we checked if **`head`** is null.
  - Now, we check that sentinel's **`next`** points to itself
- ***How do we know if we are at the last element?***
  - Before, we checked if the next element is null.
  - Now we check if the **`next`** element points to the sentinel