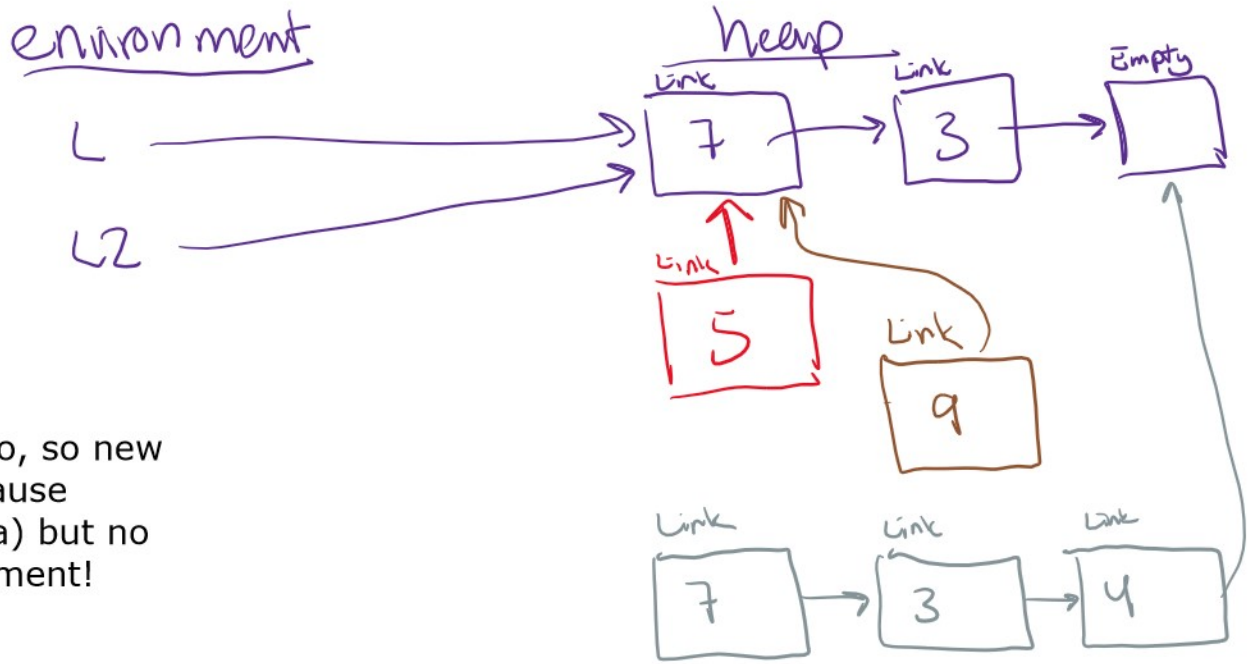
Immutable vs. Mutable lists in memory

Friday, September 23, 2022 10:54 AM

L is immutable, already contains 7,3

```
L2 = L
L2.addFirst(5)
L2.addFirst(9)
L.addLast(4)
```

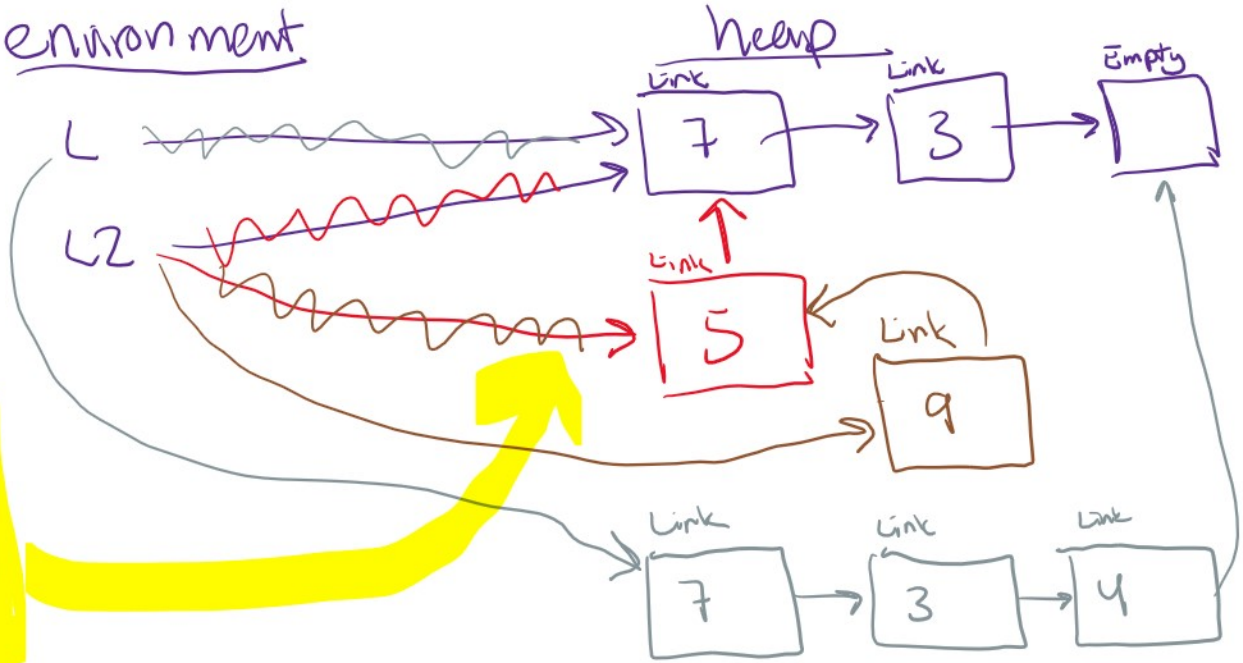
Does not change what names point to, so new objects are created in the heap (because addFirst and addLast return new data) but no way to get to them from the environment!



A variation: change what names point to

```
L2 = L
L2 = L2.addFirst(5)
L2 = L2.addFirst(9)
L = L.addLast(4)
```

Immutable: changes cannot happen to existing objects in the heap (but we can change which objects the names in the environment point to)



Now let's try this with mutable lists! Lmu is mutable, contains 3, 7

Since addFirst and addLast are void methods that alter the data in the heap, we want Lmu to always know about these changes without us writing code to change what Lmu points to in the heap. How do we do this when we're adding and removing nodes? We create separate MutableList and Node objects, where the MutableList has access to the start node. Because MutableList is mutable, what "start" refers to can change, and we will be able to access those changes from Lmu without changing what Lmu points to.

```
Lmu.addFirst(5)
Lmu.addFirst(9)
Lmu.addLast(4)
```