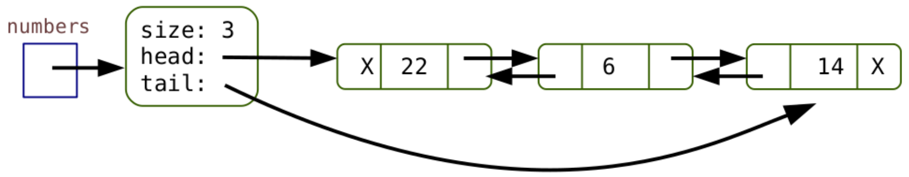
Doubly-Linked Lists

Java's implementation of `LinkedList` stores two references in each node: one for the *previous*, and one for the *next*. In addition, both the head and the tail are stored in the wrapper class.



Questions (10 min) Start time:

1. How many operations are required to add an element *at the start* of this list?

In general, 3 operations: one to create the element, one to copy the reference to the next node, and one to change the reference from the head to the new element. (If it's the first node, then the tail reference would also need to be set.)

2. How many operations are required to add an element *at the end* of this list?

In general, 3 operations: one to create the element, one to create the reference to the previous node, and one to change the reference from the tail to the new element.

3. How much memory is required for each node? How does that amount compare with using an `ArrayList`?

Each element requires two references, so there is $N \times 8$ bytes of extra overhead (where N is the length of the list).

4. What problems of singly-linked lists do doubly-linked lists solve? (In other words, what do the `previous` and `tail` make possible?)

They allow for quick access to things at the end of the list, so adding/removing at either end is fast. They also allow for the list to be traversed backwards.

5. If your program requires a `List` collection, how would you decide which implementation to use? (`ArrayList` vs `LinkedList`)

If adding items at the end and accessing random elements, use ArrayList. If inserting items in the middle and only accessing at the two ends, use LinkedList.