CSCI 136
Data Structures & Advanced Programming

Singly Linked List Variants

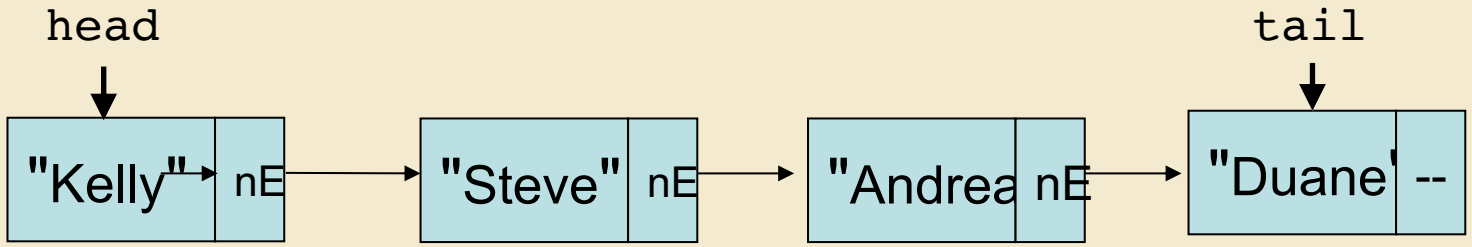
Singly Linked List Variants

Optimizing Singly Linked Lists

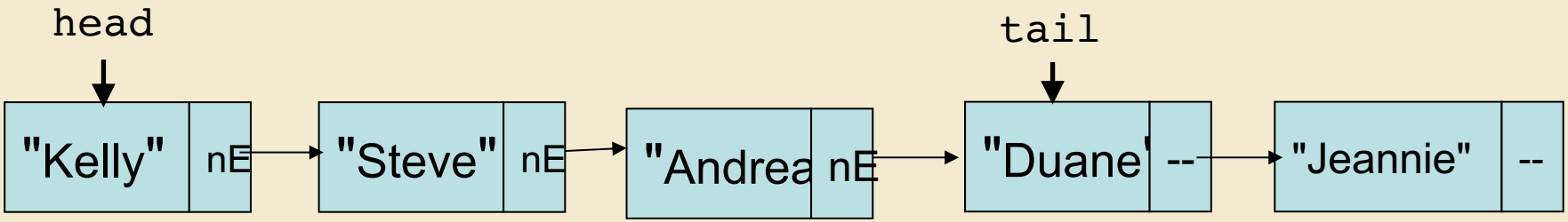
- Adding to the end of a linked list requires traversing the entire list : An O(n) operation.
- We can improve this by keeping a reference to the last element of the list ("A tail pointer")
- To add a new value to the end of the list

```
tail.setNext( new Node<E>( value ) )
tail = tail.nextelement;
count++;
```

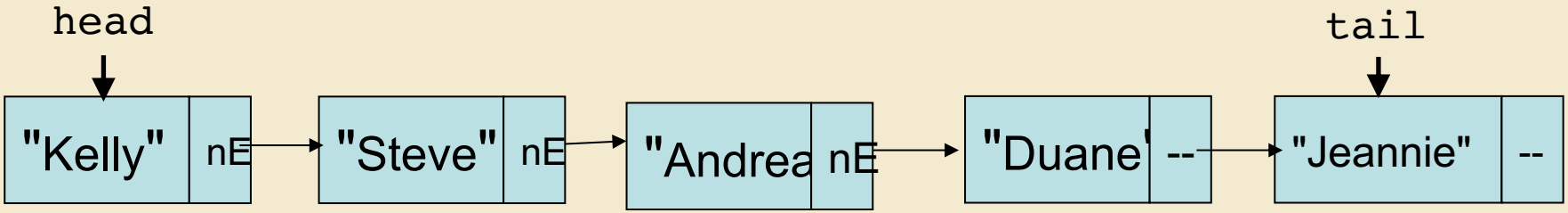
Adding a Tail Reference




```
tail.setNext( new Node<String>( "Jeannie" ) );
```




```
tail = tail.nextelement;
```



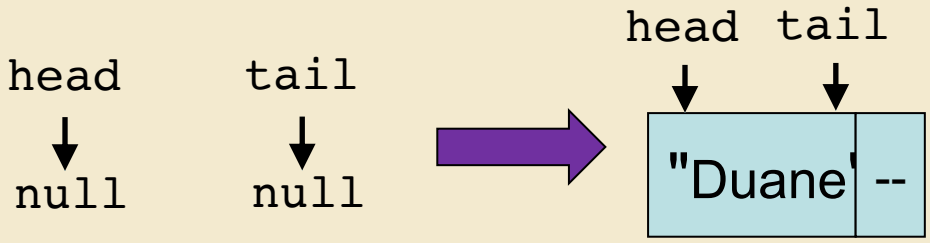

```
count++
```

Adding a Tail Reference

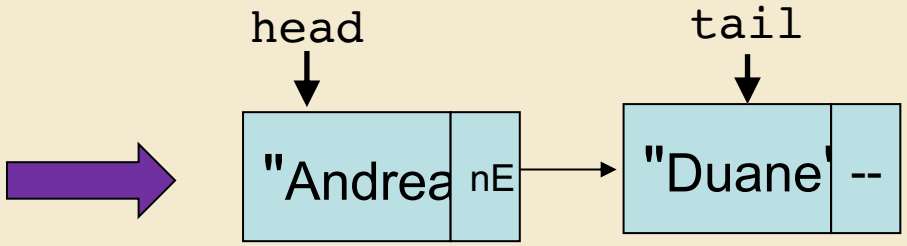
- A new instance variable (`tail`) is added to the list
- Result
  - `addLast` and `getLast` are fast: Now O(1) instead of O(n)
  - `removeLast` is not improved
    - We need to know element before tail so we can reset `tail` reference to previous element in list
- Side effects
  - Two references must be maintainted: `head` and `tail`
  - Potential for confusion
    - `head == tail` could mean an empty list or list of size 1

AddFirst

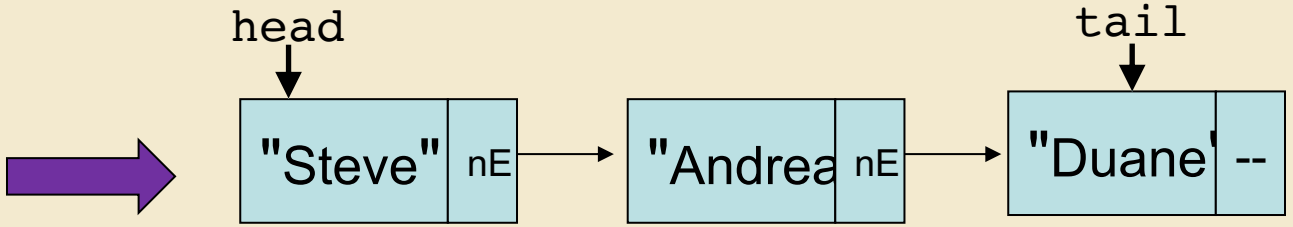
Adding to front of empty list: `myList.addFirst("Duane");`



Adding to front of one-element list: `myList.addFirst("Andrea");`



Adding to front of a longer list: `myList.addFirst("Steve");`



AddFirst

```
public void addFirst(E value) {
  // if empty list
  if(size() == 0) {
    head = new Node<E>(value);
    tail = head;
  // if not empty list
  else {
    head = new Node<E>(value, head);
  }
  count++;
}
```

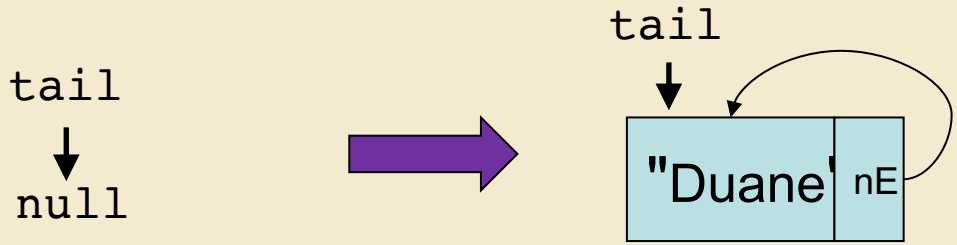
CircularlyLinkedLists

Consider the Singly-Linked List structure with tail reference.

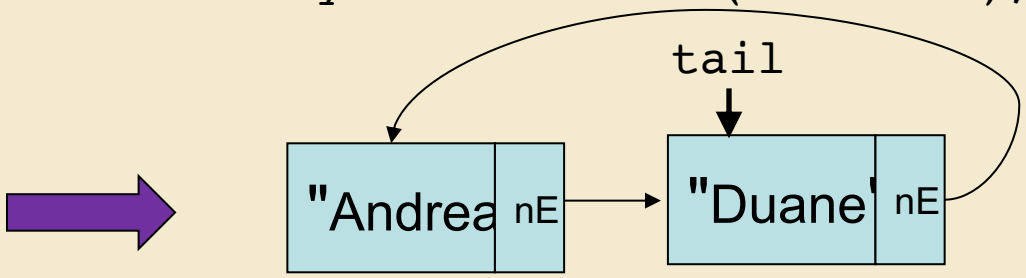
- Notice: The implementation never uses the fact that the `tail` node has a null `nextElement` reference.
- Idea: Have the `nextElement` reference of the `tail` node refer to the *first node* (`head`) of the list
- Results:
  - `head` reference is no longer needed, just use `tail.nextElement` instead!
  - ALL operations on `head` are fast!
  - `addLast()` is still fast
  - Only modest additional complexity in implementation
  - Can "cyclically reorder" (rotate) the list by changing `tail` reference

AddFirst on Circular List

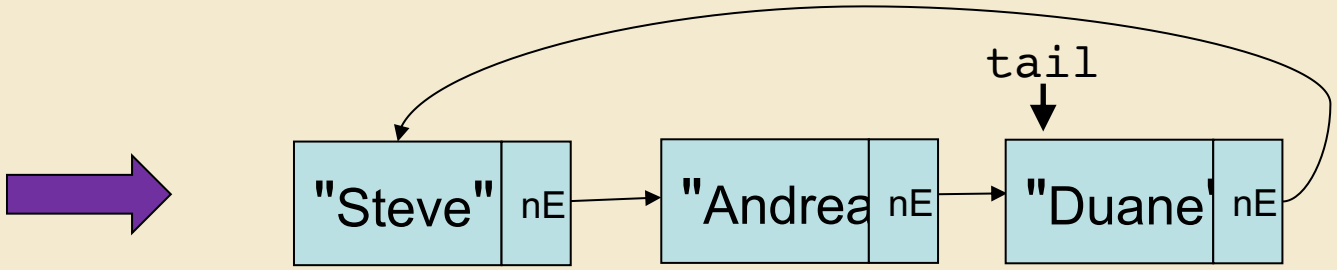
Adding to front of empty list: `myList.addFirst("Duane");`



Adding to front of one-element list: `myList.addFirst("Andrea");`



Adding to front of a longer list: `myList.addFirst("Steve");`



Circular AddFirst

```
public void addFirst(E value) {
  // if empty list
  if(size() == 0) {
    tail = new Node<E>(value);
    tail.setNext(tail);
  // if not empty list
  else {
    tail.setNext(new Node<E>(value, tail.next());
  }
  count++;
}
```

Summary

Adding an additional piece of information to a singly list list can speed up some operations.

- A tail reference speeds up adding to end of list
  - but not removing from end of list
- A link from tail node to head node
  - Removes need for head reference
  - Doesn't reduce efficiency of any method
  - Allows for list rotation
- In a future video, we'll see how adding further node references can provide additional improvements in efficiency