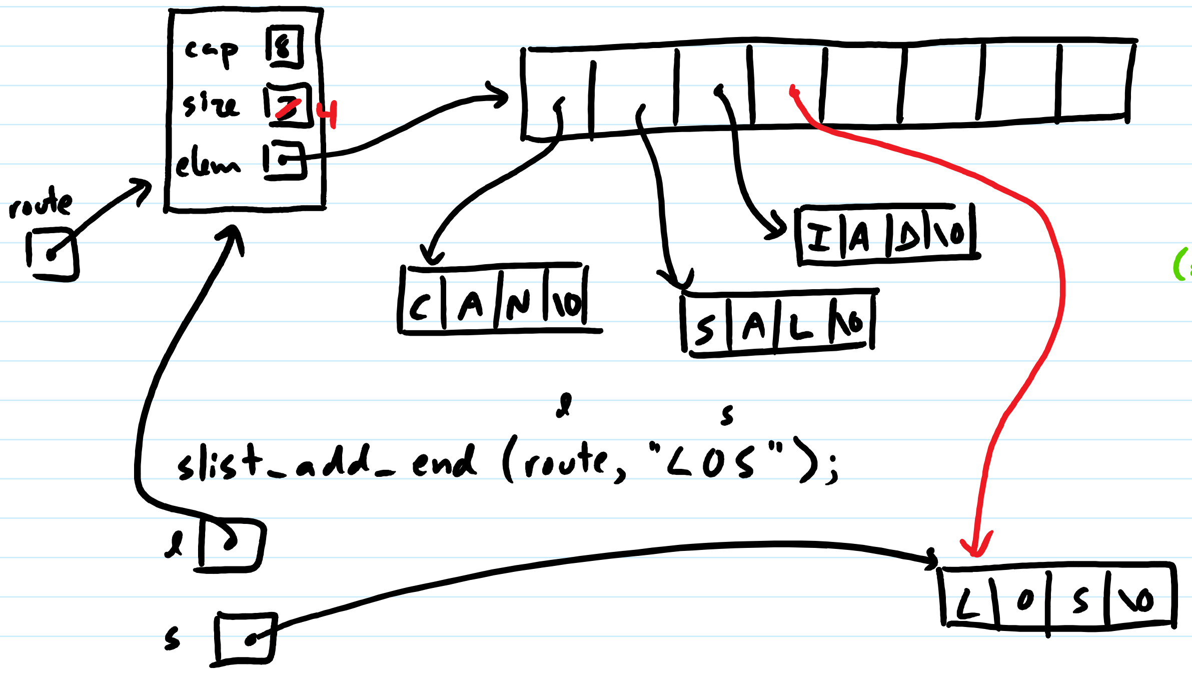
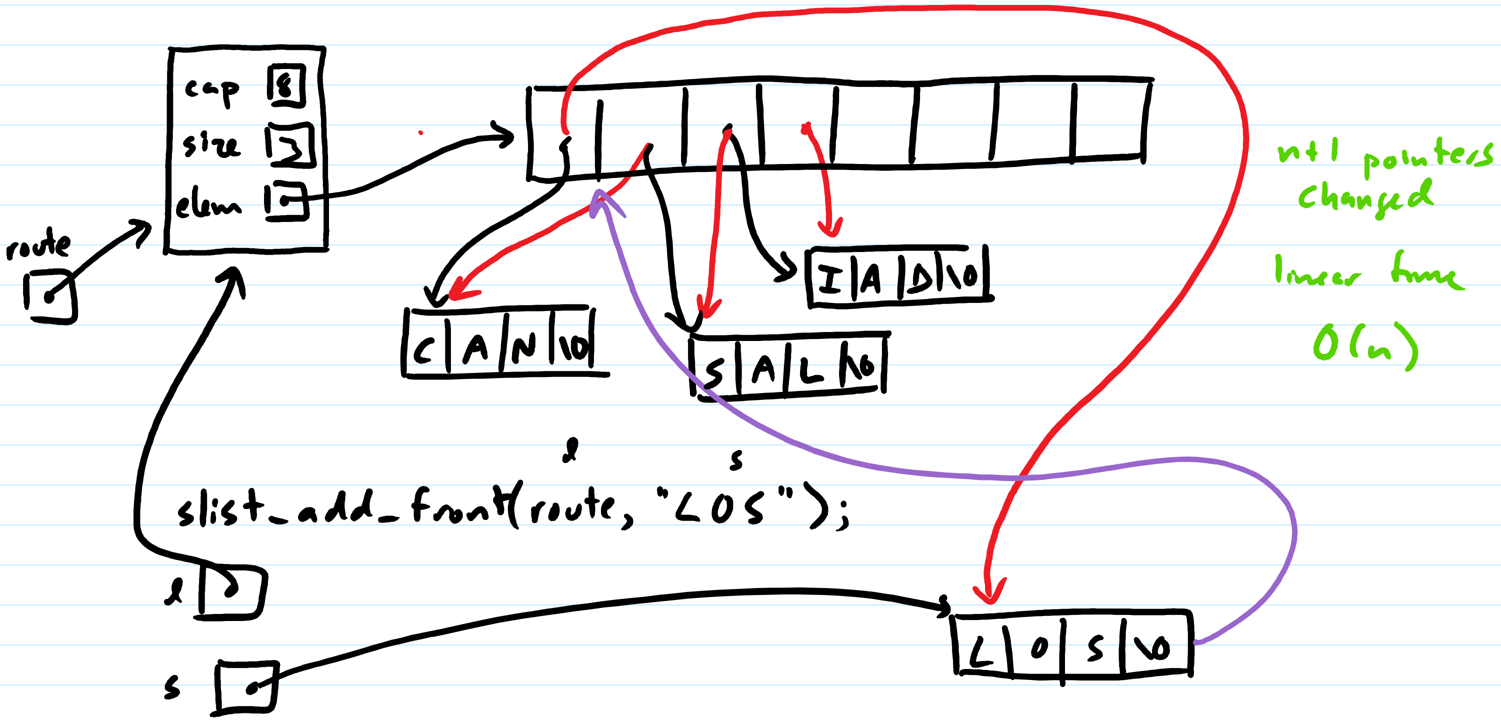


```
slist_add_end(route, "LOS");
```

constant time
(# steps doesn't
depend on
# items in list)

$O(1)$

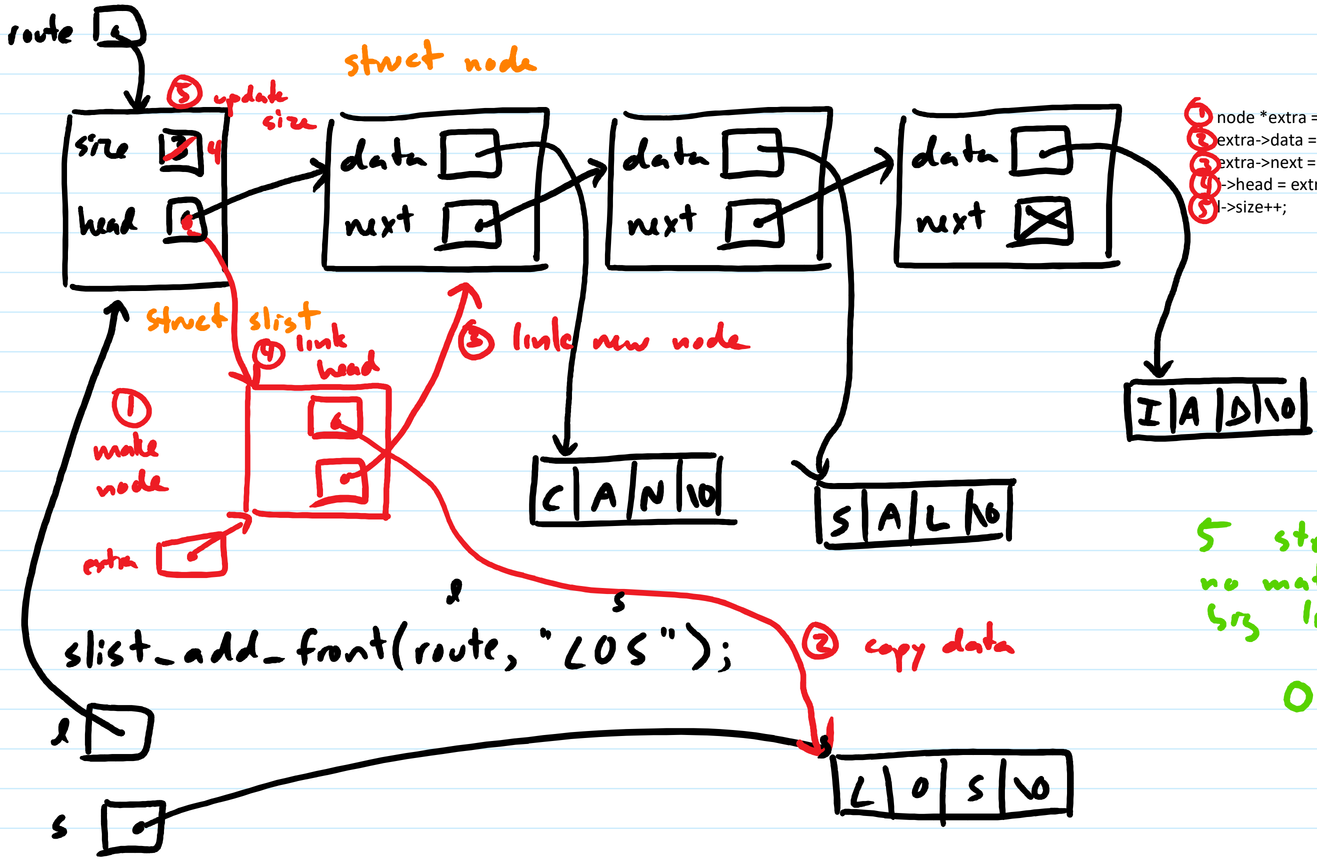


slist_add_front(route, "LOS");

n+1 pointers changed

linear time

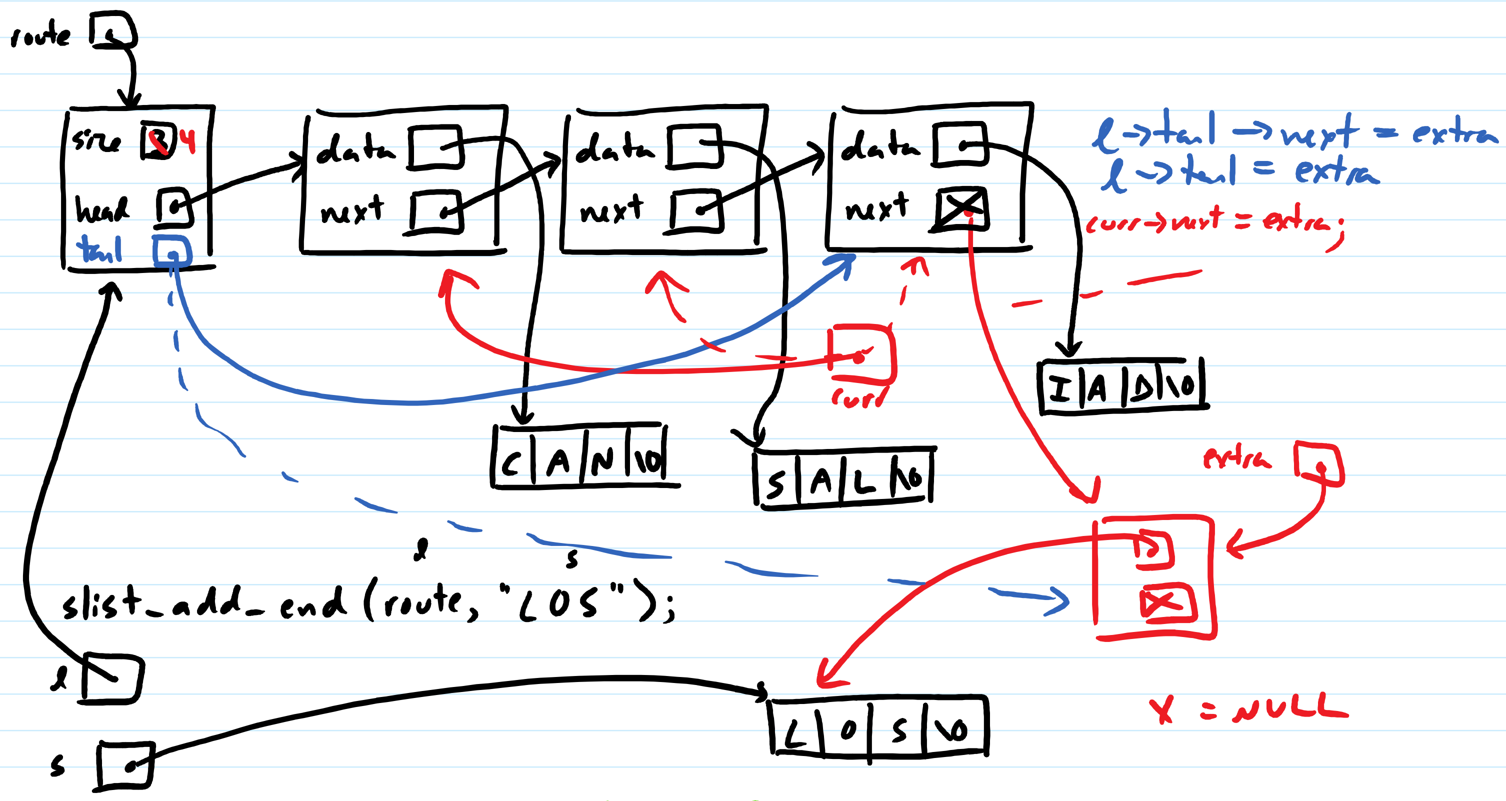
$O(n)$



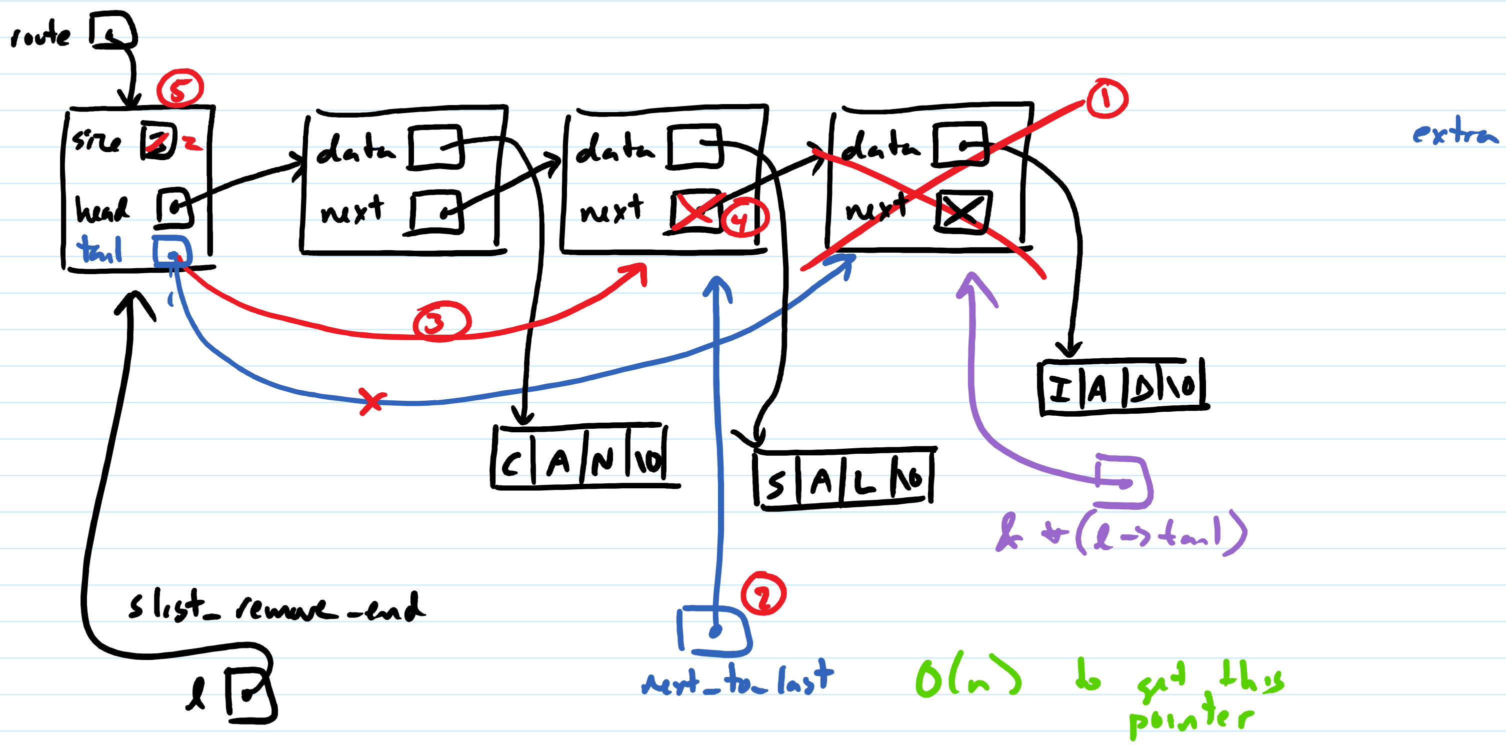
```
1 node *extra = malloc(sizeof(node));
2 extra->data = s;
3 extra->next = l->head;
4 l->head = extra;
5 l->size++;
```

5 steps no matter how big list is

O(1)

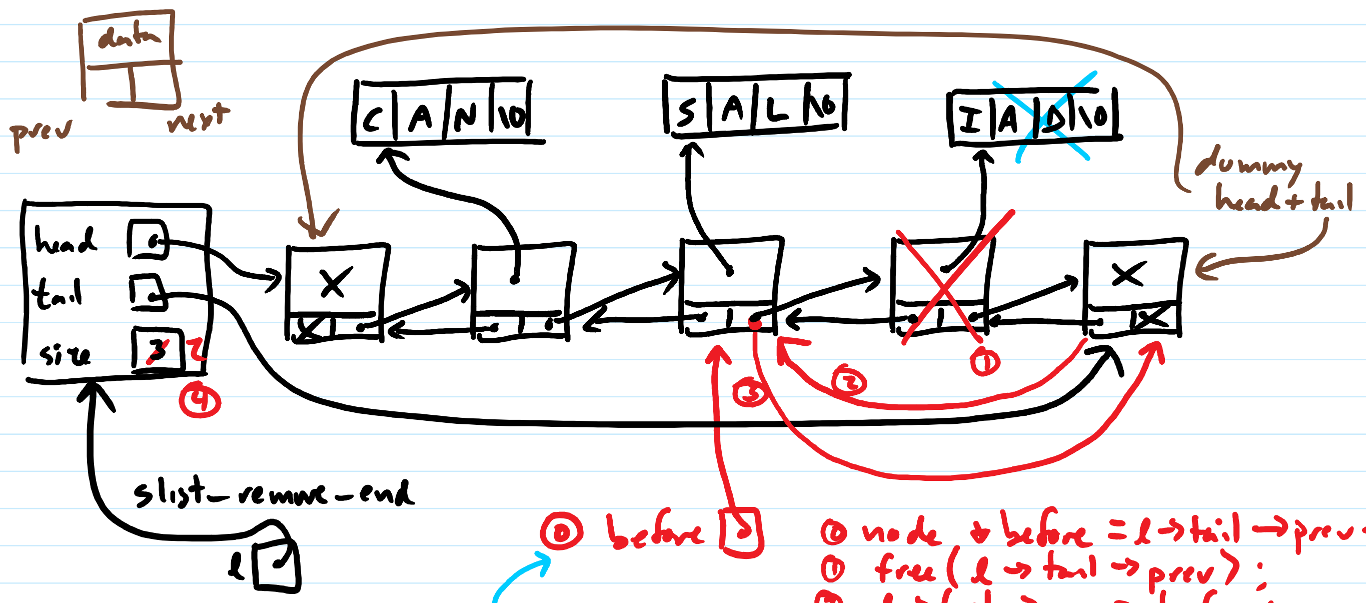


O(n) from loop to get to last node



O(n) to get this pointer

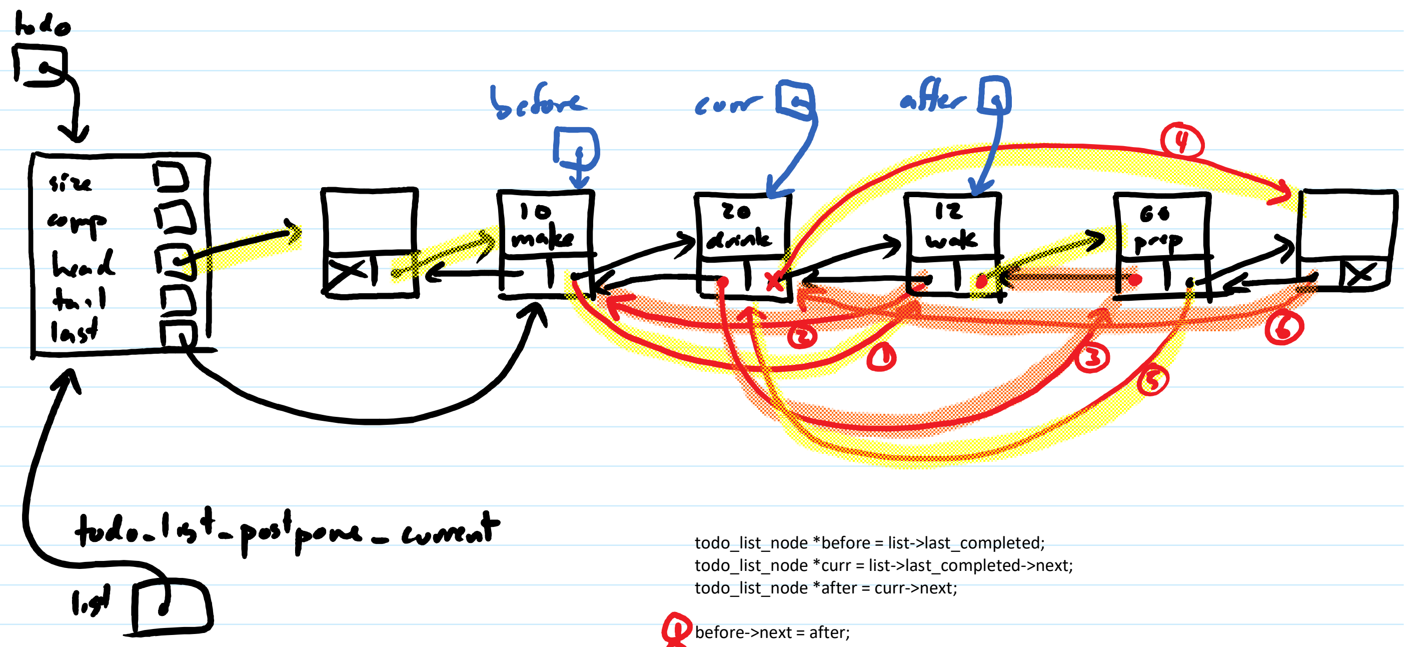
1 go to last node



⓪ node *before = l->tail->prev->prev;
① free(l->tail->prev);
② l->tail->prev = before;
③ before->next = l->tail;
④ l->size--;

½ free(l->tail->prev->data);
(if strings belong to list)

O(1)



```
todo_list_node *before = list->last_completed;
todo_list_node *curr = list->last_completed->next;
todo_list_node *after = curr->next;

1 before->next = after;
2 after->prev = before;

3 curr->prev = list->tail->prev;
4 curr->next = list->tail;
5 list->tail->prev->next = curr;
6 list->tail->prev = curr;
```

forward: 10 make 12 walk 60 prep 20 drink

backward: 20 drink 60 prep 12 walk 10 make

| | | array list | doubly-linked list | fancy array list |
|---|---|---|---|---|
| add to | back | O(1) if no resize | O(1) | O(1) |
| | front | O(n) | O(1) | O(1) |
| remove from | back | O(1) | O(1) | O(1) |
| | front | O(n) | O(1) | O(1) |
| add/remove at index | | O(n) | O(n) | |
| get | | O(1) | O(n) | |
| size | | O(1) | O(1) | |
| sort | | O(n log n) modified quicksort or heapsort | O(n log n) merge sort | |