LEC 12

CSE 123

Linked Lists with Recursion

Questions during Class?

Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Best boba in Seattle?

Instructor: James Wilcox

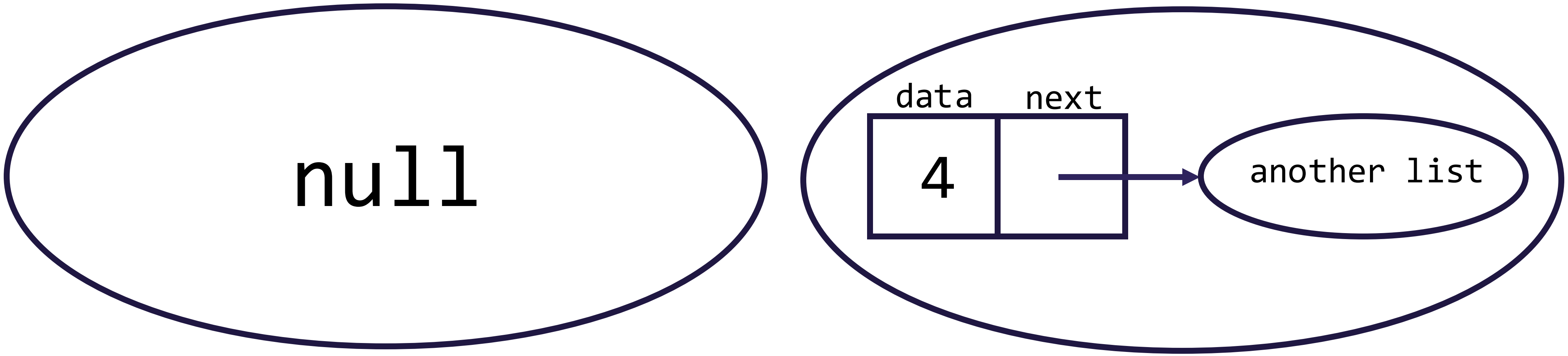
Announcements

- C2 due tonight 11/6
- R5 due Friday 11/8
- P2 out tomorrow 11/7
- Quiz 1 grades out early next week

- Apply to be a 12X TA!

Linked Lists

- A linked list is either:



Empty list

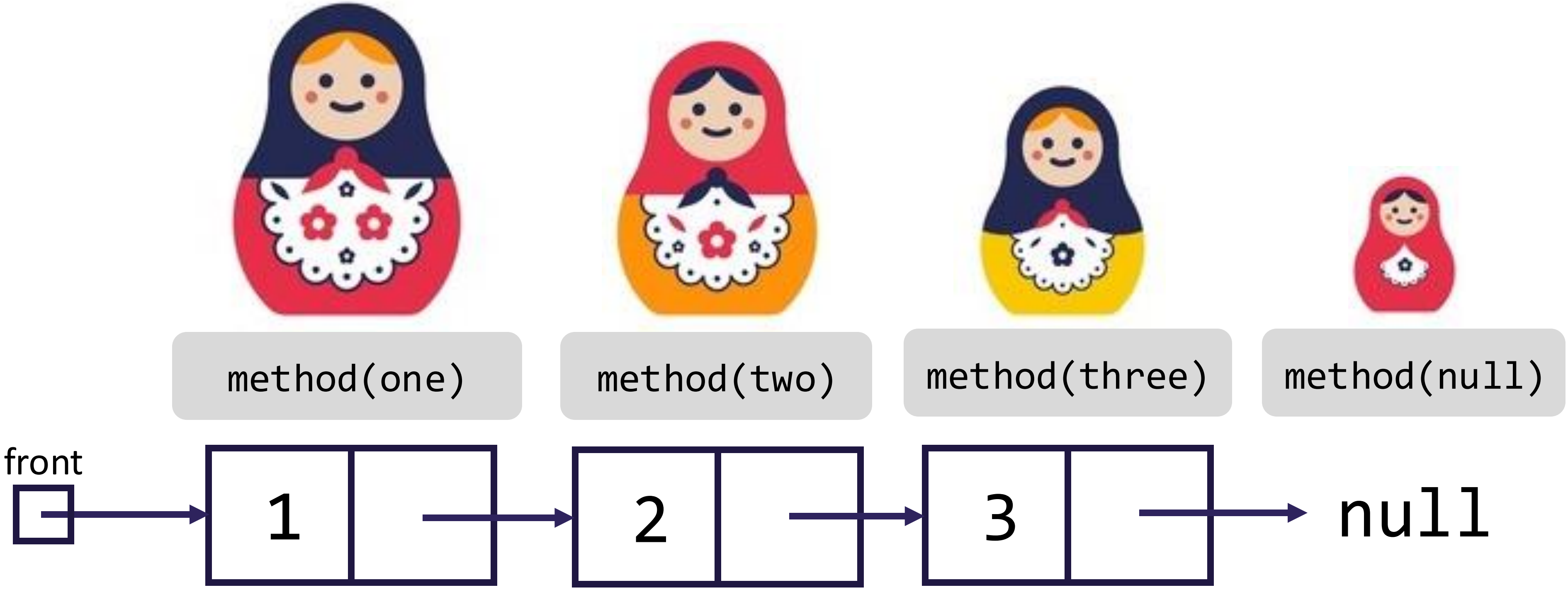
Node w/ another linked list

This is a recursive definition!

A list is either empty or a node with another list!

Recursive Traversals w/ LinkedLists

- Guaranteed base case: empty list
  - Simplest possible input, should immediately know the return
- Guaranteed public / private pair
  - Need to know which sublist you're currently processing (i.e. `curr`)



`method(one)` `method(two)` `method(three)` `method(null)`

Modifying LinkedLists [Review]

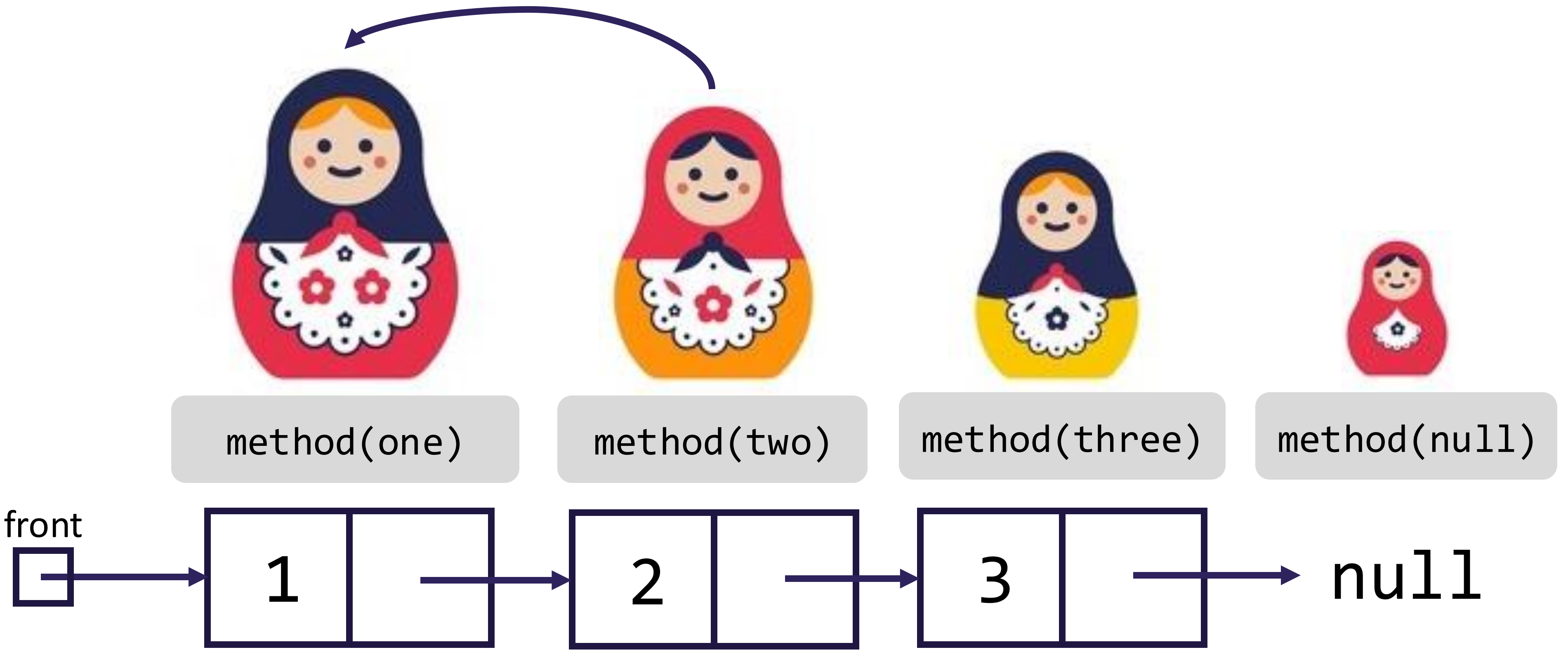
- Remember: using a `curr` variable to iterate over nodes

- Does changing `curr` actually update our chain?
  - What will? Changing `curr.next`, changing `front`
  - Need to **stop one early** to make changes

- Often a number of cases to watch out for:
  - M(iddle) – Modifying node in the middle of the list (general)
  - F(ront) – Modifying the first node
  - E(mpty) – What if the list is empty?
  - E(nd) – Rare, do we need to do something with the end of the list?

Modifying LinkedLists Recursively

- Much easier than iterative solutions!
- No longer need to stop one early
  - Can go right to the point you'd like to make the change



Modifying LinkedLists Recursively

- Much easier than iterative solutions!
- No longer need to stop one early
  - Can go right to the point you'd like to make the change
- How? Return the updated change and catch it!
  - Private pair returns `ListNode` type
  - `curr.next = change(curr.next)` / `front = change(front)`
  - Resulting solutions much cleaner than iterative cases
- We call this pattern **`x = change(x)`**