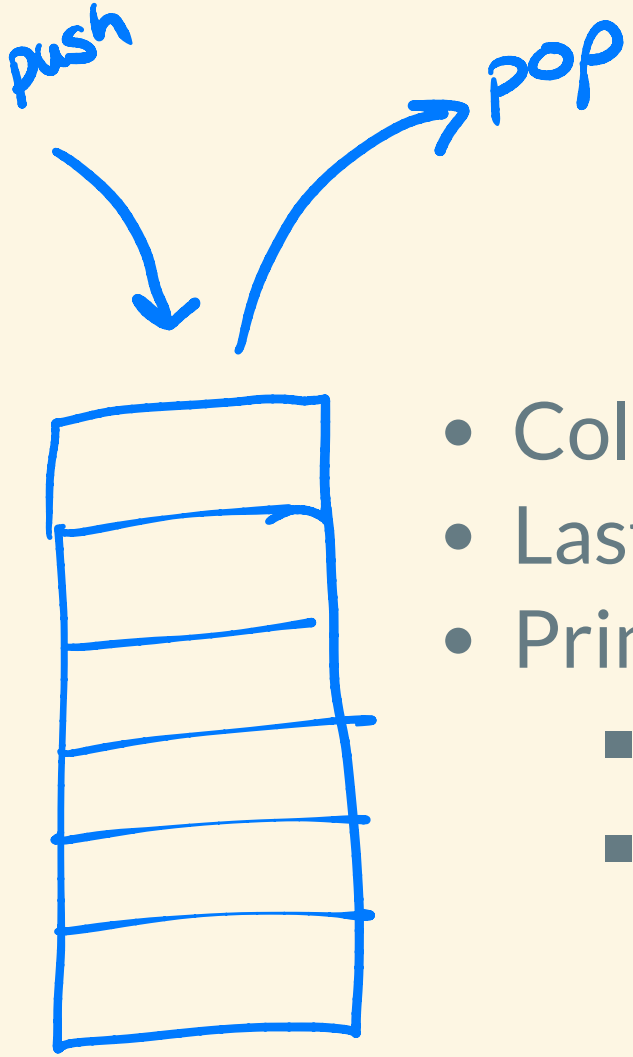
STACKS

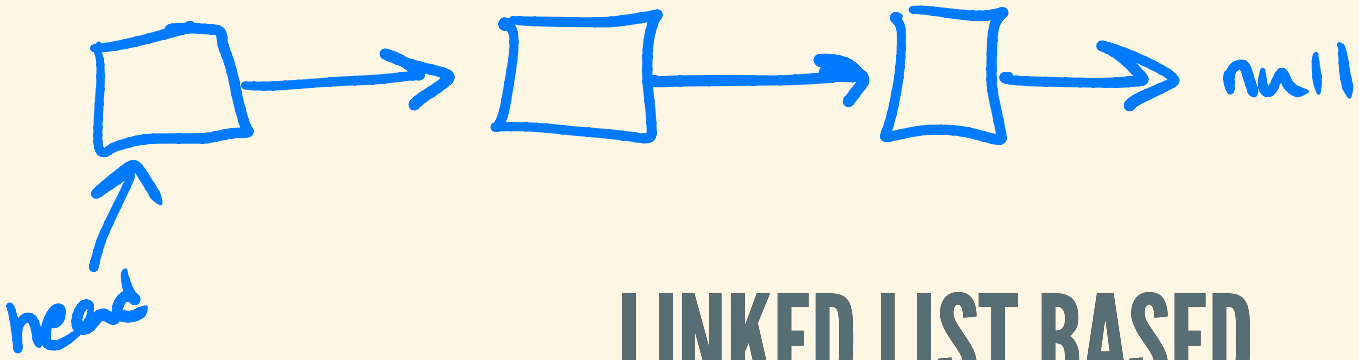


RECALL:

- Collection of objects
- Last in - first out (LIFO)
- Primary operations:
  - push (add to top)
  - pop (remove from top)

IMPELEMNTATIONS:

- Separate from the ADT
- The details of how we create the stack data structure
- Options:
  - Linked list based
  - Array based

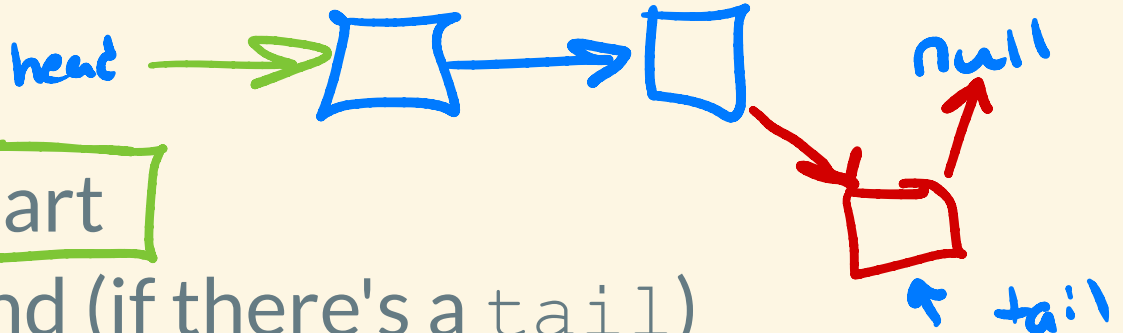


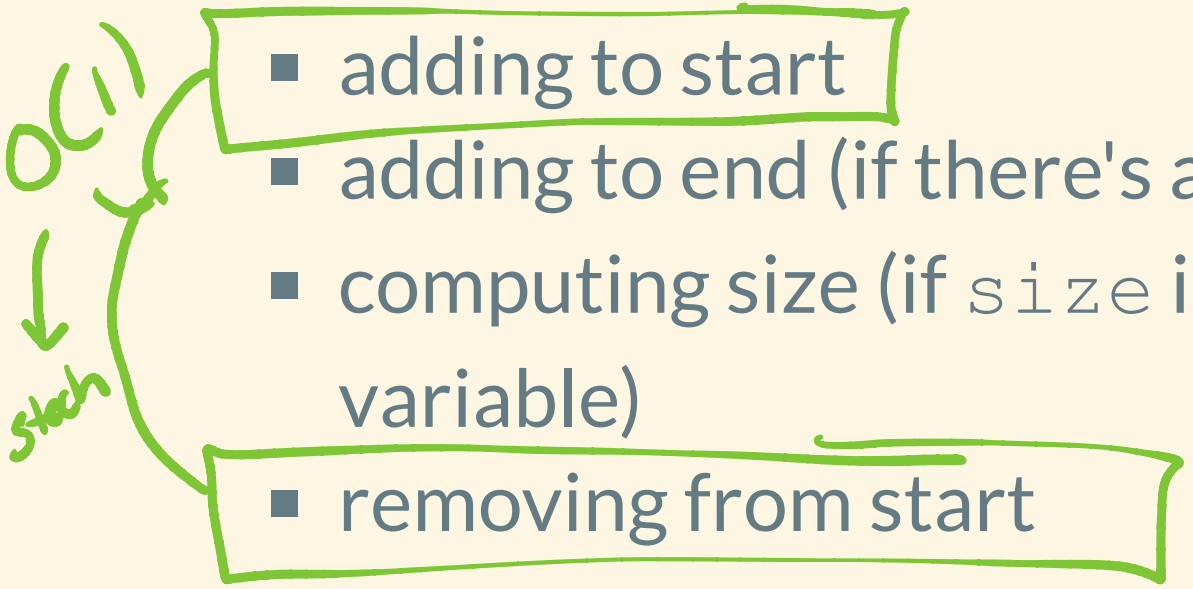
LINKED LIST BASED

- Store stack as linked list
- How could we best represent stack using linked list for operations to be as efficient as possible?
  - Can `push` be O(1)?
  - Can `pop` be O(1)?
  - Possible to implement so they are both O(1)?

LINKED LIST BASED

- Think about what operations with linked list were O(1):



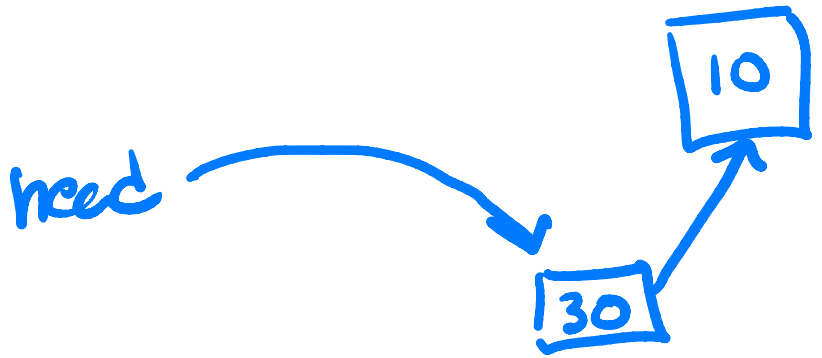


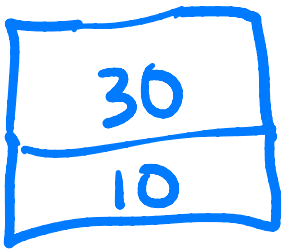
  - adding to start
  - adding to end (if there's a `tail`)
  - computing size (if `size` is stored as instance variable)
  - removing from start

- LIFO: want to remove the last one we added

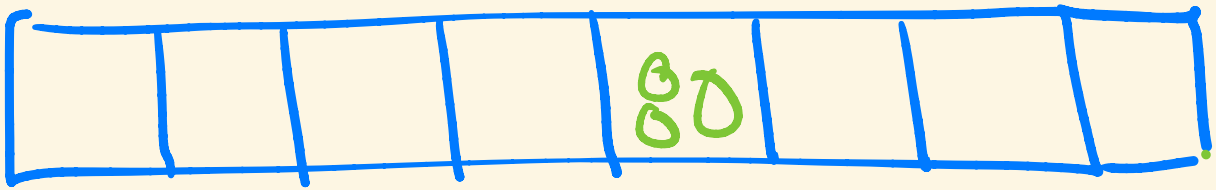
Stack

underlying linked list



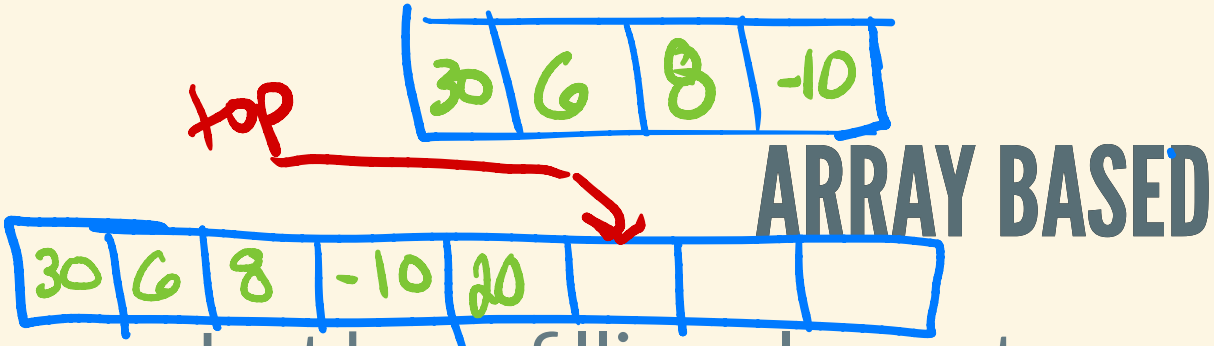


push (10)
push (30)
push (-20)
pop ()



ARRAY BASED

- Store stack using an array
- How can we represent to be as efficient as possible?
  - Can `push` be O(1)?
  - Can `pop` be O(1)?
  - Possible to implement so they are both O(1)?



ARRAY BASED

push (30)
push (6)
push (8)
push (-10)
push (20)

- Just keep filling elements
  - Keep track of index representing `top`
  - Setting value of element is O(1) → same for poppins
- Problem: everytime we resize need to create new and copy over
  - Don't resize everytime we add
  - Grab a chunk more (typically 2x) when we need to resize
  - O(n), but happens rarely