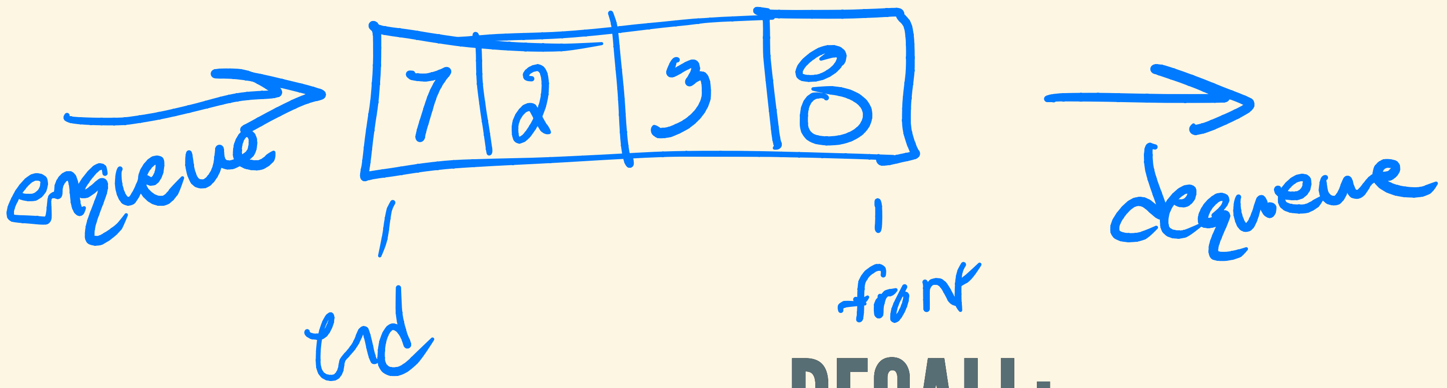
QUEUES



RECALL:

- Collection of elements
- First in - first out (FIFO)
- Primary operations:
  - enqueue (add to end)
  - dequeue (remove from front)

IMPELEMENTATIONS:

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

LINKED LIST BASED

- Store queue as linked list
- How can we efficiently represent queue using linked list?
    - Can `enqueue` be O(1)?
    - Can `dequeue` be O(1)?
    - Possible to implement so they are both O(1)?

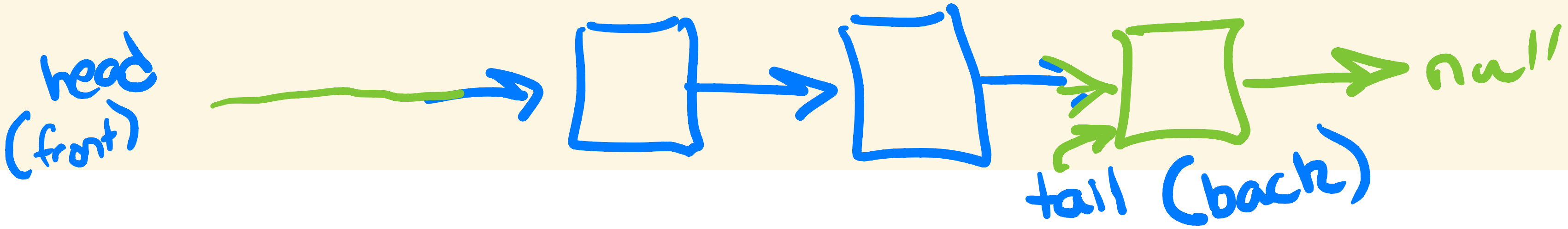
LINKED LIST BASED

dequeue: removing from head of list

enqueue: adding to tail of list

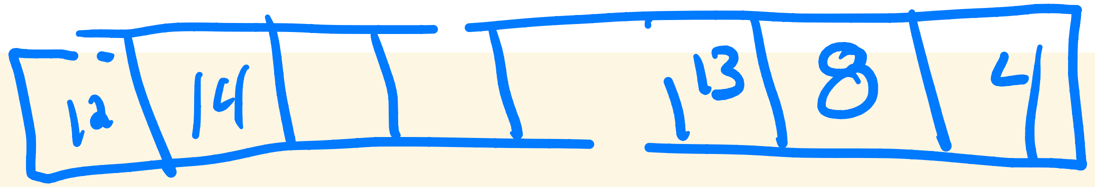
- Think about what operations with linked list were O(1):
    - adding to start
    - adding to end (if there's a `tail`)
    - computing size (if `size` is stored as instance variable)
    - removing from start
- FIFO: need add at one end, remove at other

  back (above "one end"), front (above "other")



ARRAY BASED

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



ARRAY BASED

front = 5

end = 2

- Don't want to shift elements (not efficient)
- Idea: store front and end index
- Problem:
  - add and remove shifts both to one end of the array
  - eventually run out of usable space (but empty space in array)
- Fix:
  - circular array