CS240 Fall 2014

Mike Lam, Professor

Image from http://www.boxtechnologies.com/box-solutions/retail-banking/

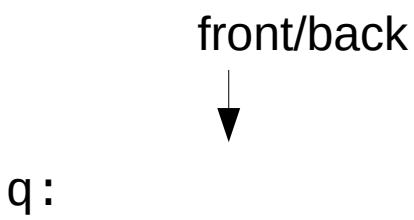
Queues

Queues

- First in, first out (FIFO) sequence data structure
- Basic operations
  - `Q.enqueue(e)`: add element e to back
  - `Q.dequeue()`: remove and return front element
  - `Q.first()`: return (but do not remove) front element
  - `Q.is_empty()`: return True if no elements
  - `len(Q)`: return number of elements

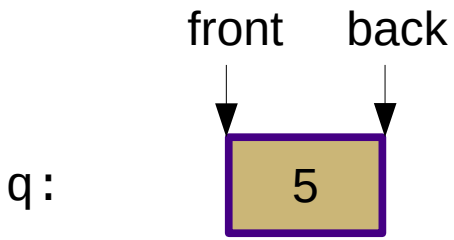
Queues

- `q = Queue()`



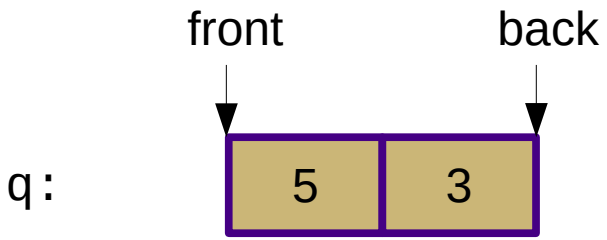
Queues

- `q.enqueue(5)`



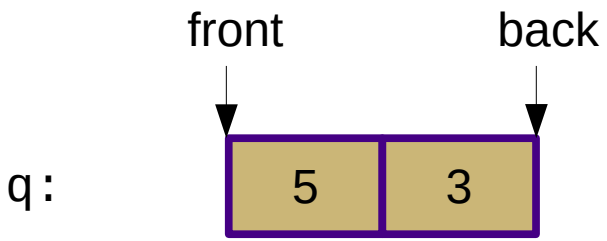
Queues

- `q.enqueue(3)`



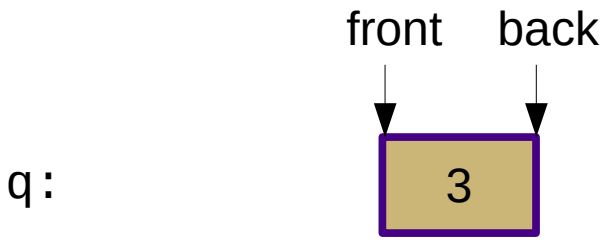
Queues

- `len(q) == 2`



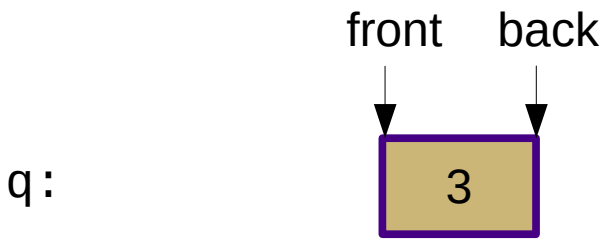
Queues

- `q.dequeue() == 5`



Queues

- `q.is_empty() == False`



Queues

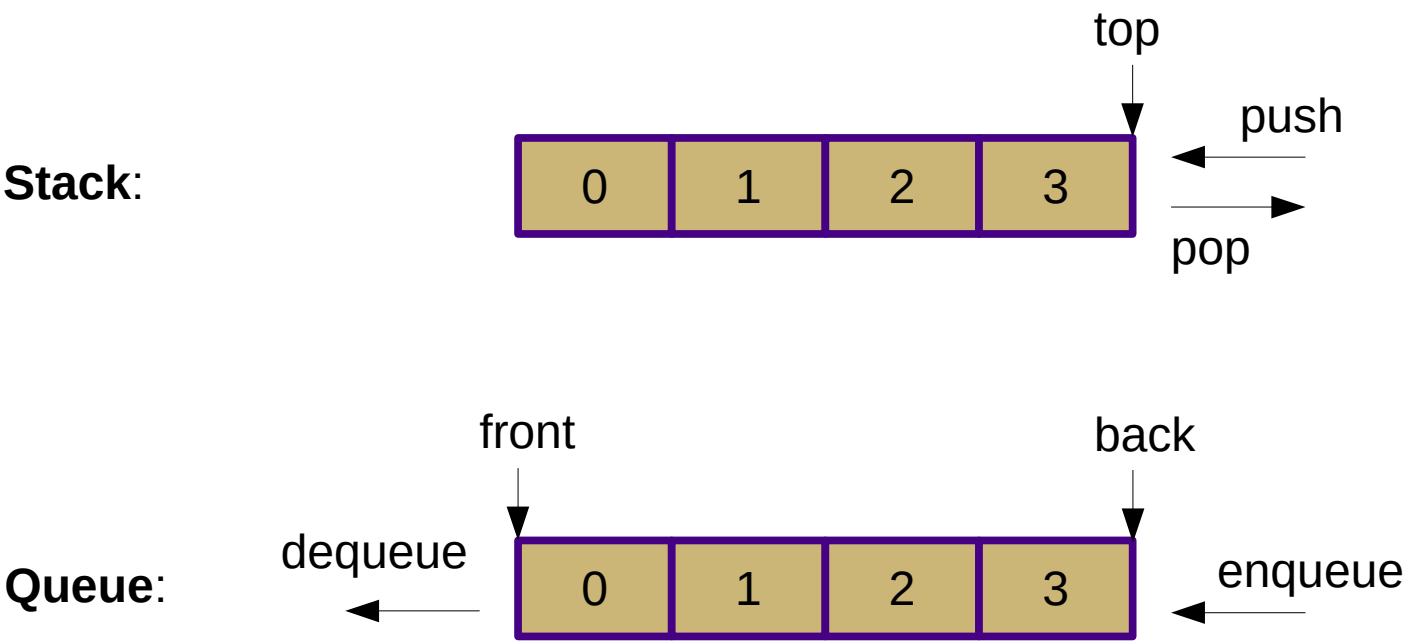
- `q.dequeue() == 3`

Queues

- `q.is_empty() == True`

Queue Implementation

- Challenge: operations manipulate both ends



Queue Implementation

- Bad implementation: **remove** front element for every dequeue()
  - Would need to shift/copy every other item
  - Θ(n)!
- Better idea: leave the elements alone
  - Keep track of the position of “front”

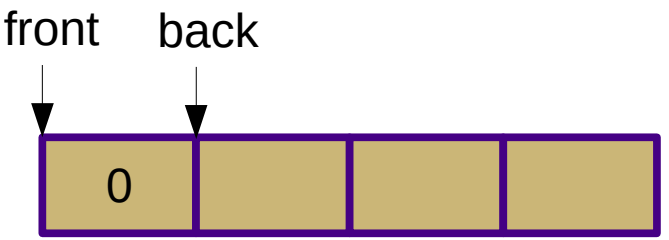
Queue Implementation

```
new queue:
```


```
enqueue(0)
```

Queue Implementation

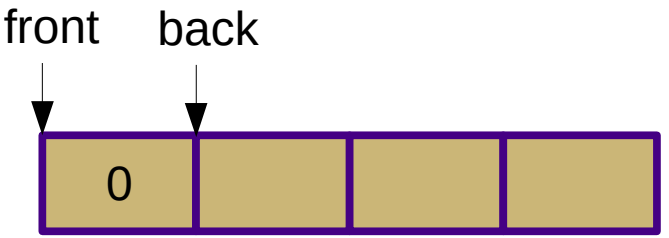
```
enqueue(0)
```



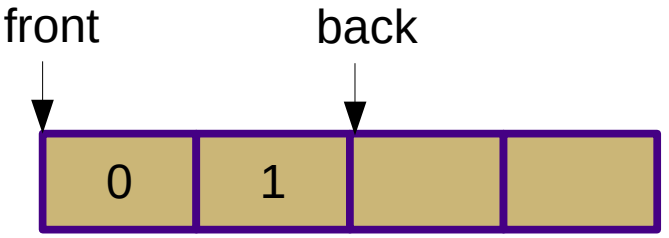

```
enqueue(1)
```

Queue Implementation

`enqueue(0)`



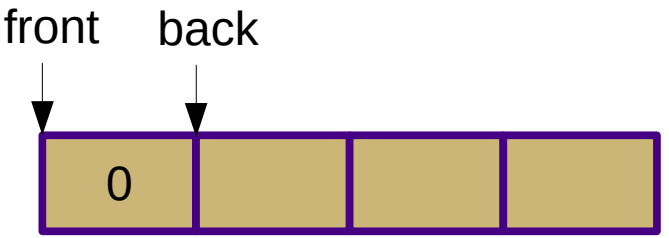
`enqueue(1)`



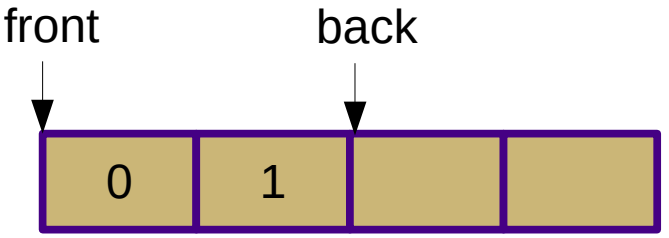
`enqueue(2)`

Queue Implementation

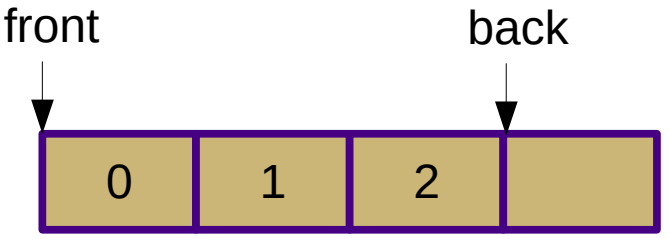
`enqueue(0)`



`enqueue(1)`



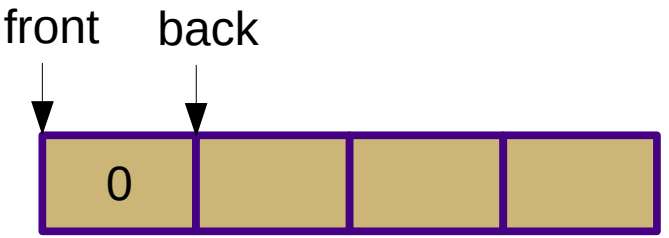
`enqueue(2)`



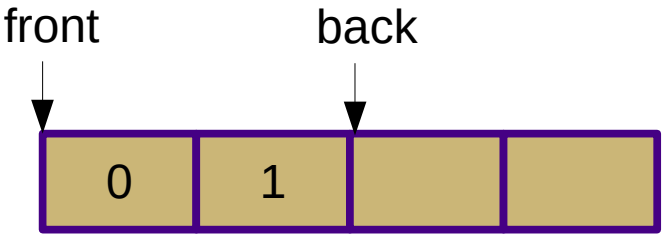
`enqueue(3)`

Queue Implementation

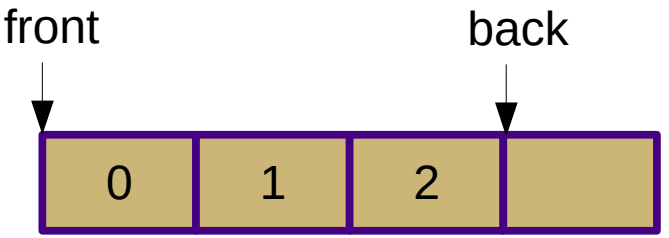
`enqueue(0)`



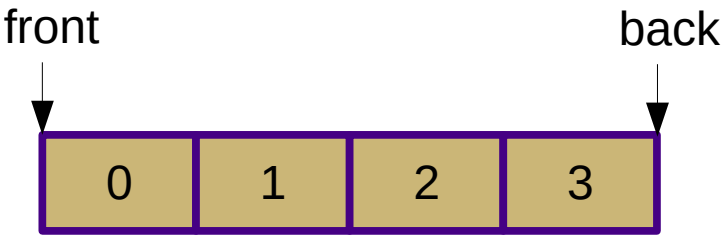
`enqueue(1)`



`enqueue(2)`

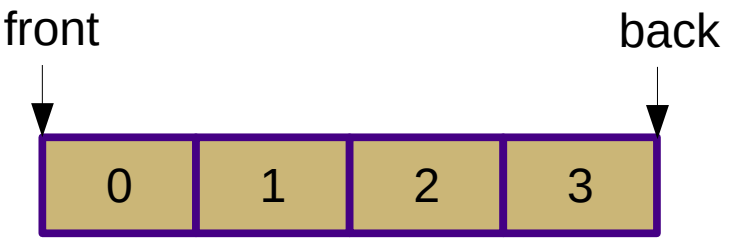


`enqueue(3)`



Queue Implementation

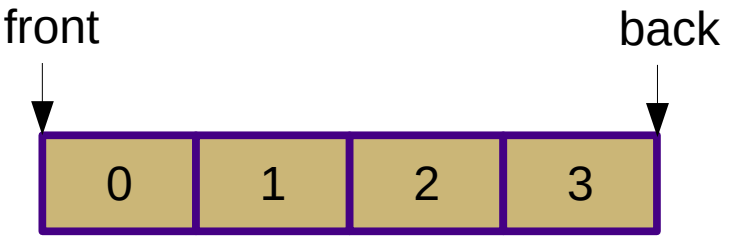
```
enqueue(0); enqueue(1)
enqueue(2); enqueue(3)
```



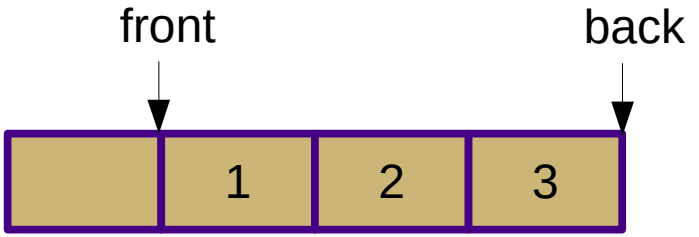

```
dequeue()
```

Queue Implementation

```
enqueue(0); enqueue(1)
enqueue(2); enqueue(3)
```



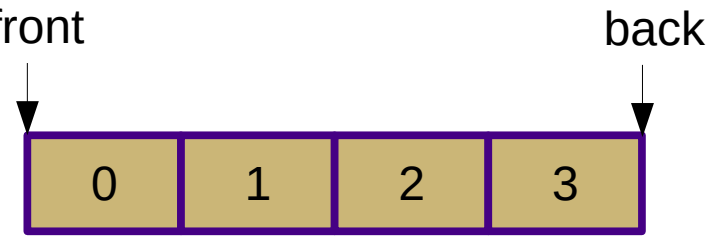

```
dequeue()
```



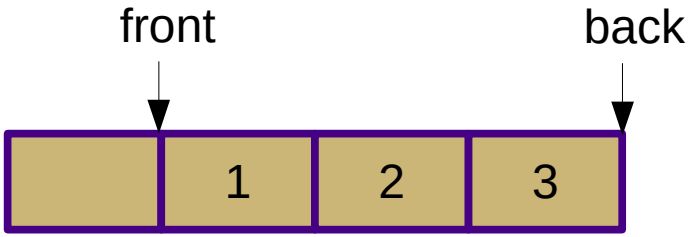

```
dequeue()
```

Queue Implementation

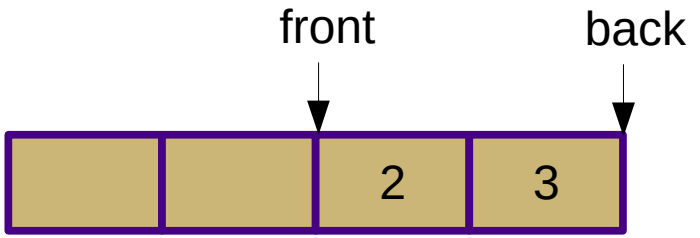
```
enqueue(0); enqueue(1)
enqueue(2); enqueue(3)
```




```
dequeue()
```



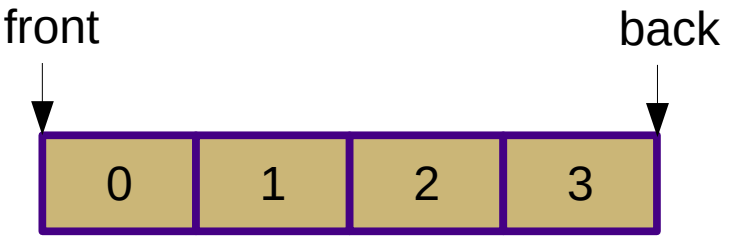

```
dequeue()
```



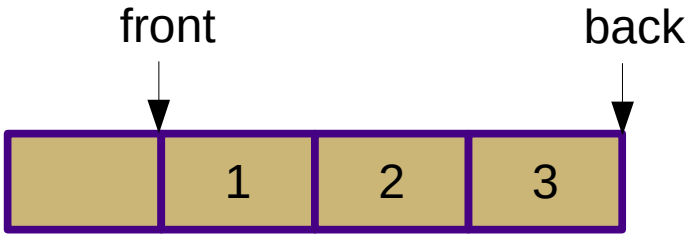

```
enqueue(4)
```

Queue Implementation

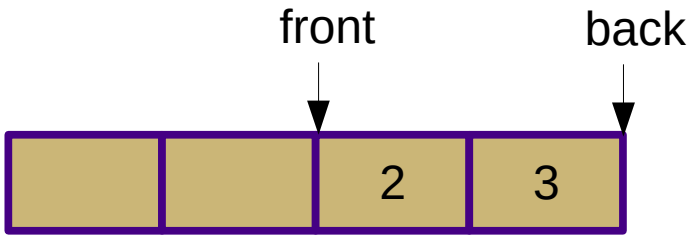
```
enqueue(0); enqueue(1)
enqueue(2); enqueue(3)
```



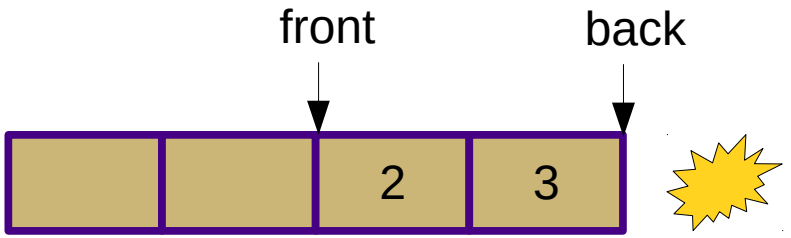

```
dequeue()
```




```
dequeue()
```

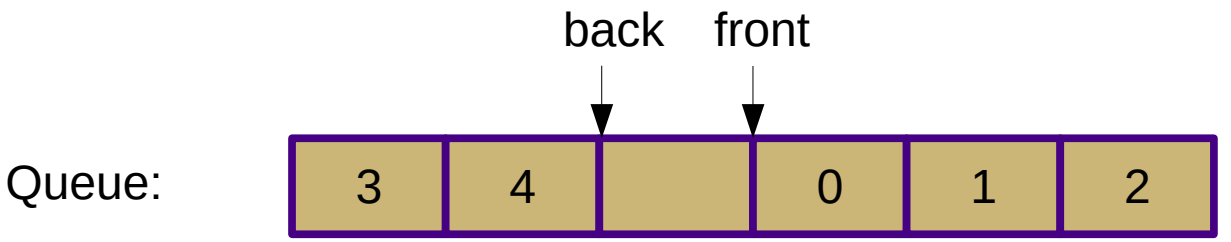



```
enqueue(4)
```



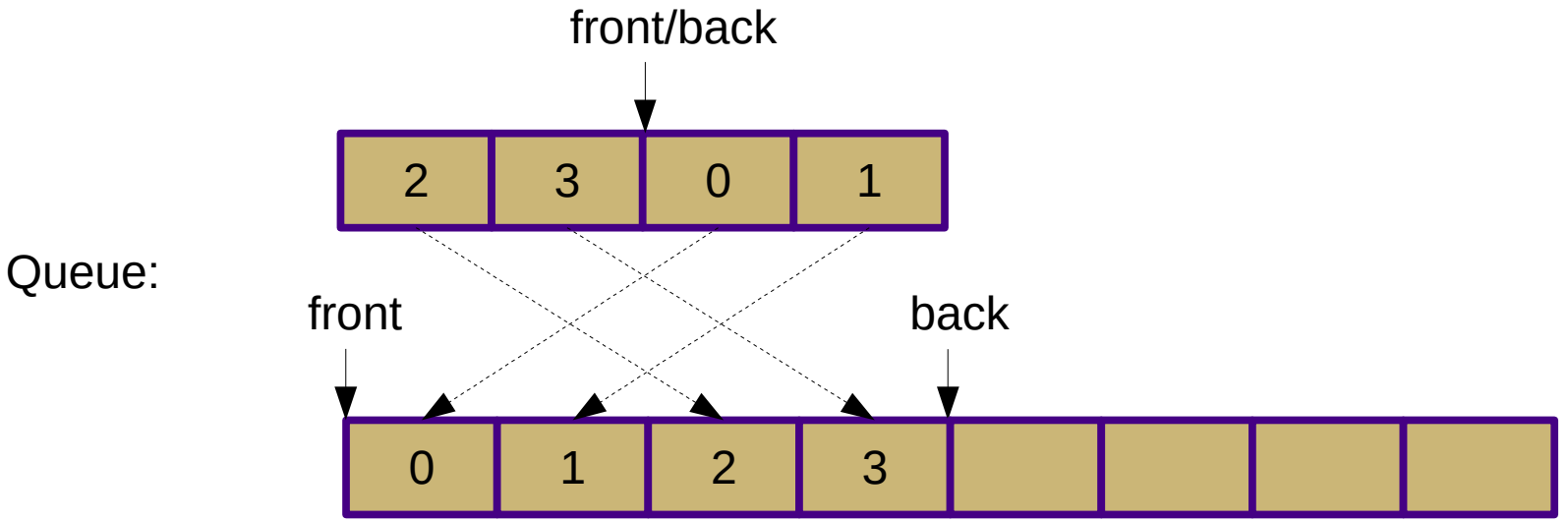
Queue Implementation

- Problem: end of array reached, but it's not full
  - Need to “wrap around” and re-use space
  - Use a *circular* array
    - Instead of `data[i]`
    - use `data[(front + i) % cap]`



Queue Implementation

- What about resizing?
  - Chance to “reset” front location



Queue Implementation

- Using Array from PA2
  - `from t_array import Array`
  - Creation: `a = Array(<capacity>)`
  - Get length:`len(a)`
  - Access:`a[i]`
  - Modify: `a[i] = x`
  - Clean up:`a.free()`

Queue Analysis

Operation	Running Time
`Q.enqueue(x)`	
`Q.dequeue()`	
`Q.front()`	
`Q.is_empty()`	
`len(Q)`	

* = amortized

Queue Analysis

Operation	Running Time
`Q.enqueue(x)`	$O(1)$*
`Q.dequeue()`	$O(1)$*
`Q.front()`	$O(1)$
`Q.is_empty()`	$O(1)$
`len(Q)`	$O(1)$

* = amortized

Queues

- Applications
  - Process scheduling
  - Printer queue
  - Web server responses
  - I/O buffering

Iterators

- Reading: Sections 1.8 and 2.3.4
- Lazy evaluation for iteration over data structures
- Need to use optional *iterable* parameter in Set constructor
- Need to provide Set iterator for PA2
  - Two choices for implementation
    - Iterator classes (easier to understand)
    - Generator method (easier to code)
  - Demo code on Piazza

Queue Implementation

- Using Array from PA2
  - `from t_array import Array`
  - Creation: `a = Array(<capacity>)`
  - Get length:`len(a)`
  - Access:`a[i]`
  - Modify:`a[i] = x`
  - Clean up:`a.free()`

PA2 General Hints

- Don't be overwhelmed
  - Most functions are <10 lines of code
  - Don't over-complicate things
- Test early and often
  - Write a function, write a test
  - Advice: don't try the provided unit tests until you have finished most of the project
- Reuse code
  - Many functions can be implemented by calling others
  - This is easier **and** better!

Stacks

- Changes to stack.py
  - Resize checks new_cap against _len, not _cap
    - Allow shrinking as well as expanding, although this has not been implemented
  - Set top element to None in pop()
    - Visualization is now more intuitive
    - Aids Python garbage collector