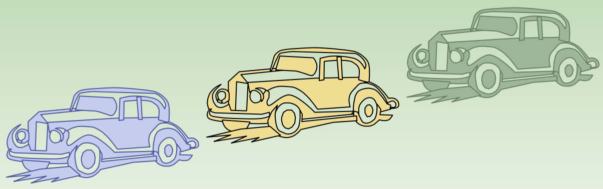
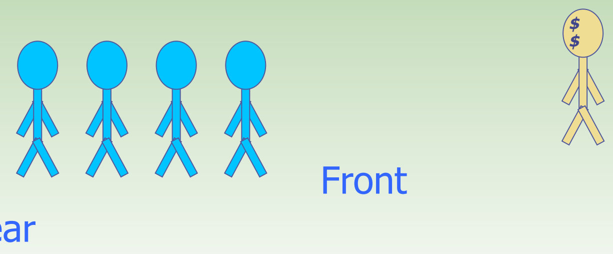
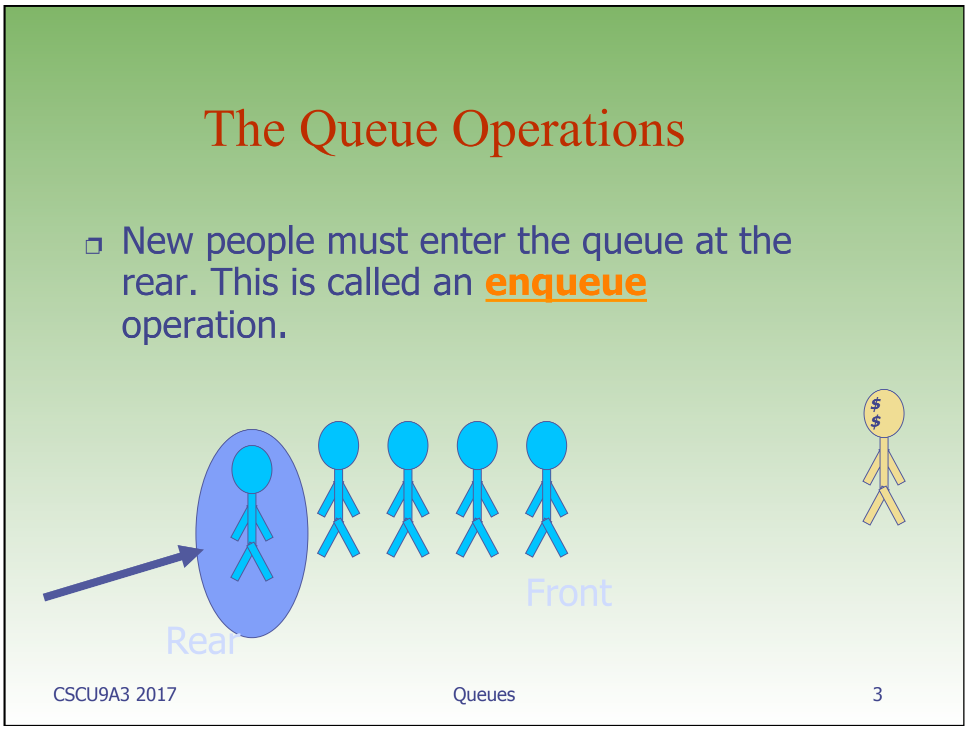
Queues



The Queue Operations

- A queue is like a line of people waiting for a bank teller. The queue has a **front** and a **rear**.



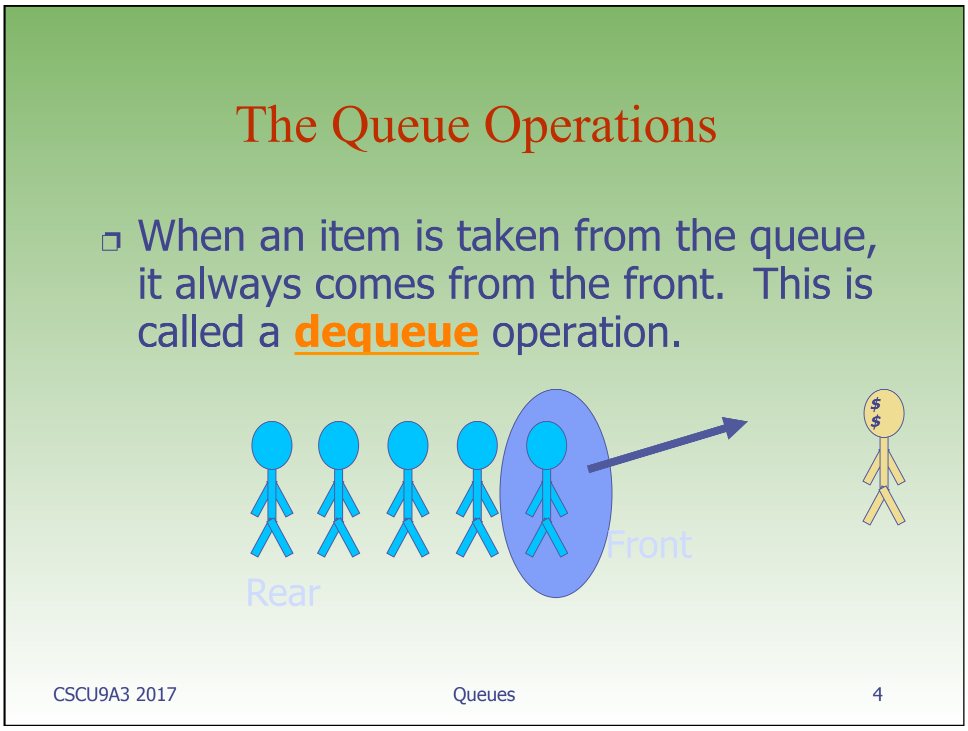


The Queue Operations

- New people must enter the queue at the rear. This is called an **enqueue** operation.

Front

Rear

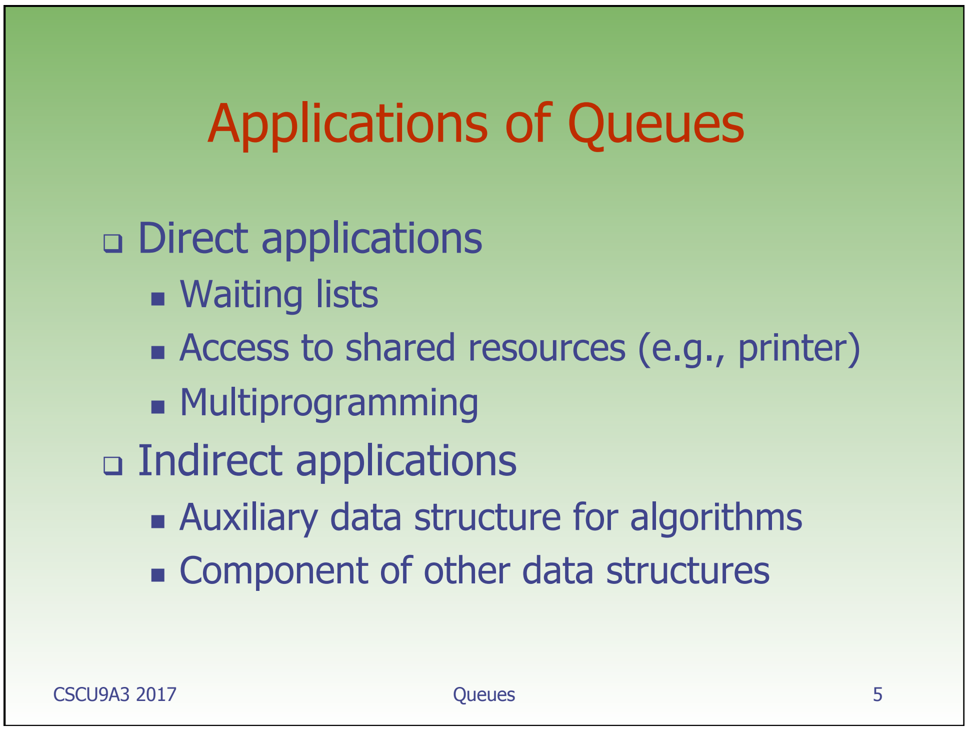


The Queue Operations

- When an item is taken from the queue, it always comes from the front. This is called a **dequeue** operation.

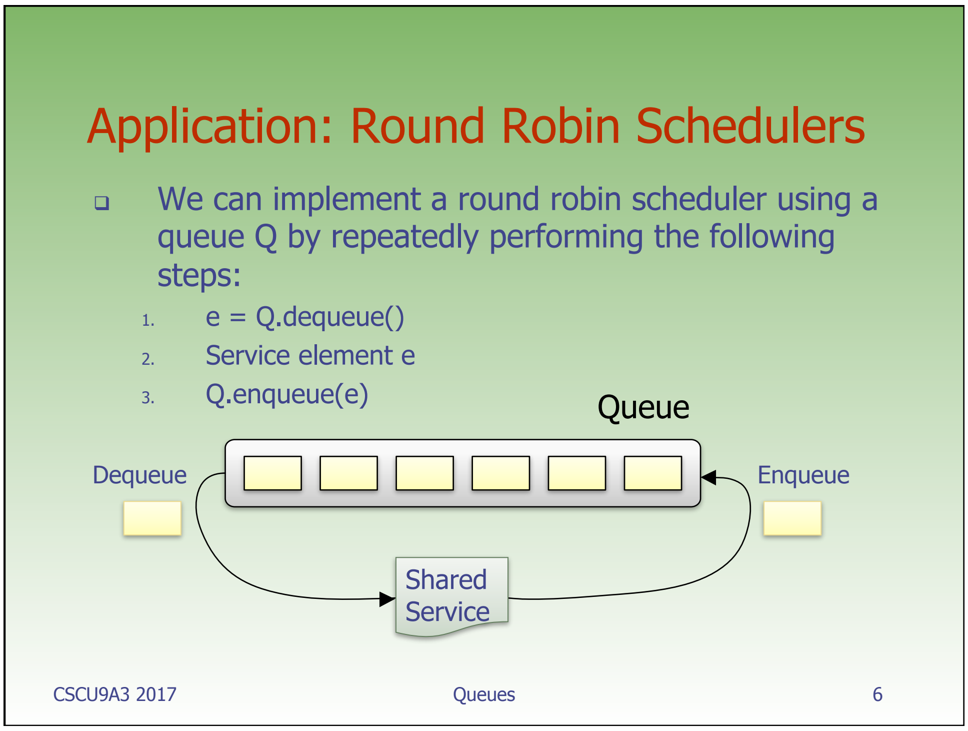
Front

Rear



Applications of Queues

- Direct applications
  - Waiting lists
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

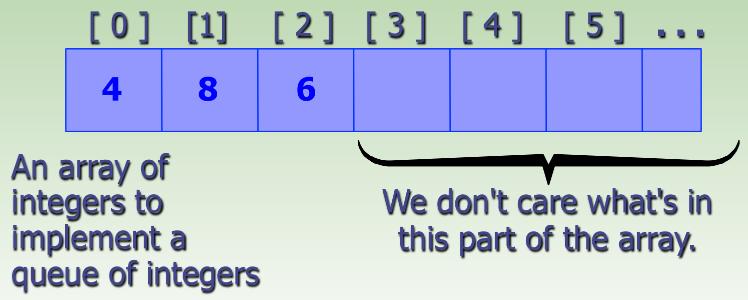


Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
  1. e = Q.dequeue()
  2. Service element e
  3. Q.enqueue(e)

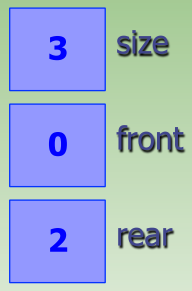
Implementation using an Array

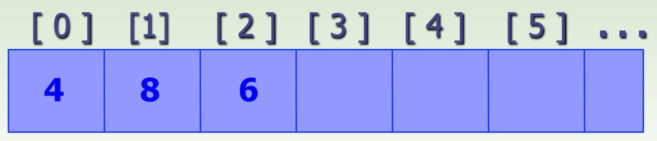
- A queue can be implemented with an array, as shown here. For example, this queue contains the integers 4 (at the front), 8 and 6 (at the rear).

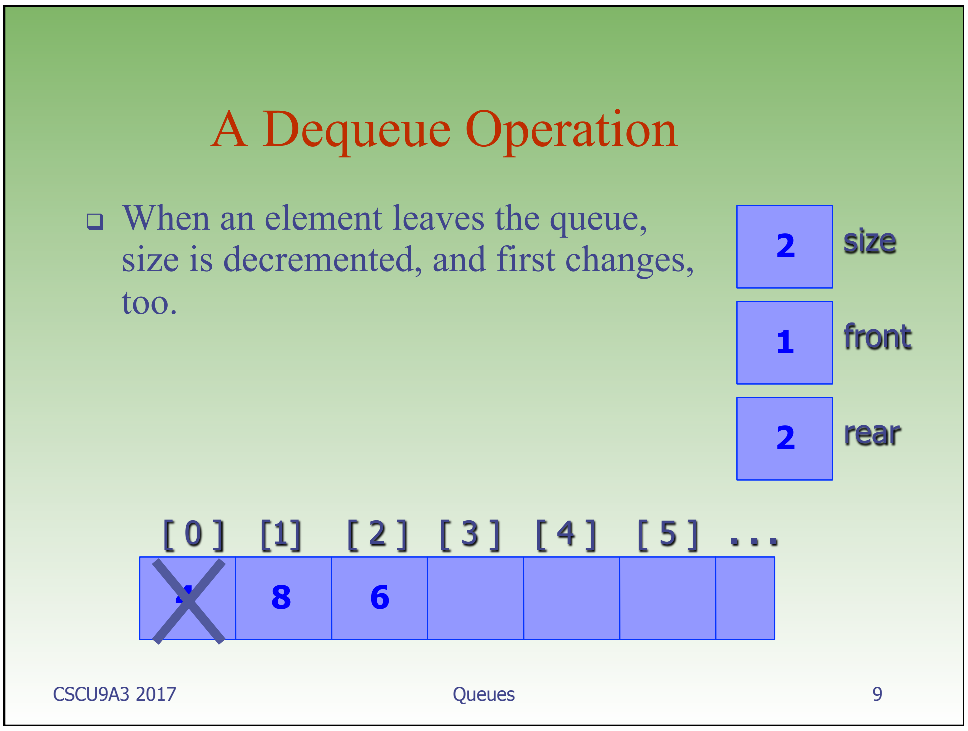

[0] [1] [2] [3] [4] [5] ...
4
8
6
An array of integers to implement a queue of integers
We don't care what's in this part of the array.

Implementation using an Array

- The easiest implementation also keeps track of the number of items in the queue and the index of the first element (at the front of the queue), the last element (at the rear).







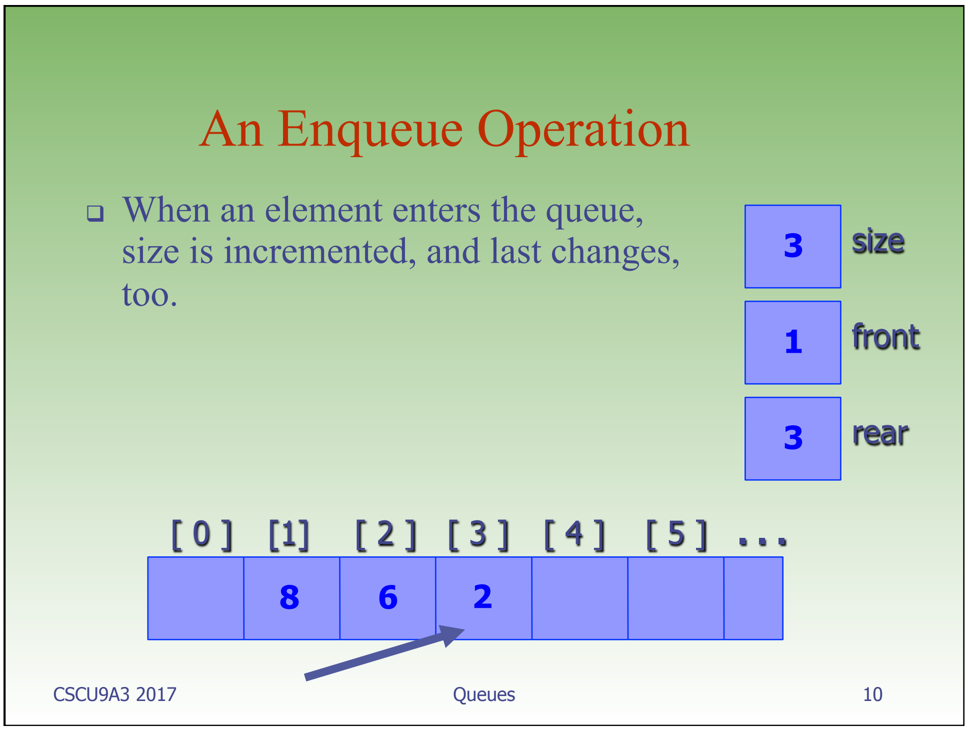
A Dequeue Operation

- When an element leaves the queue, size is decremented, and first changes, too.

2	size
1	front
2	rear

[0]	[1]	[2]	[3]	[4]	[5]	...
~~4~~	8	6				

CSCU9A3 2017 Queues 9



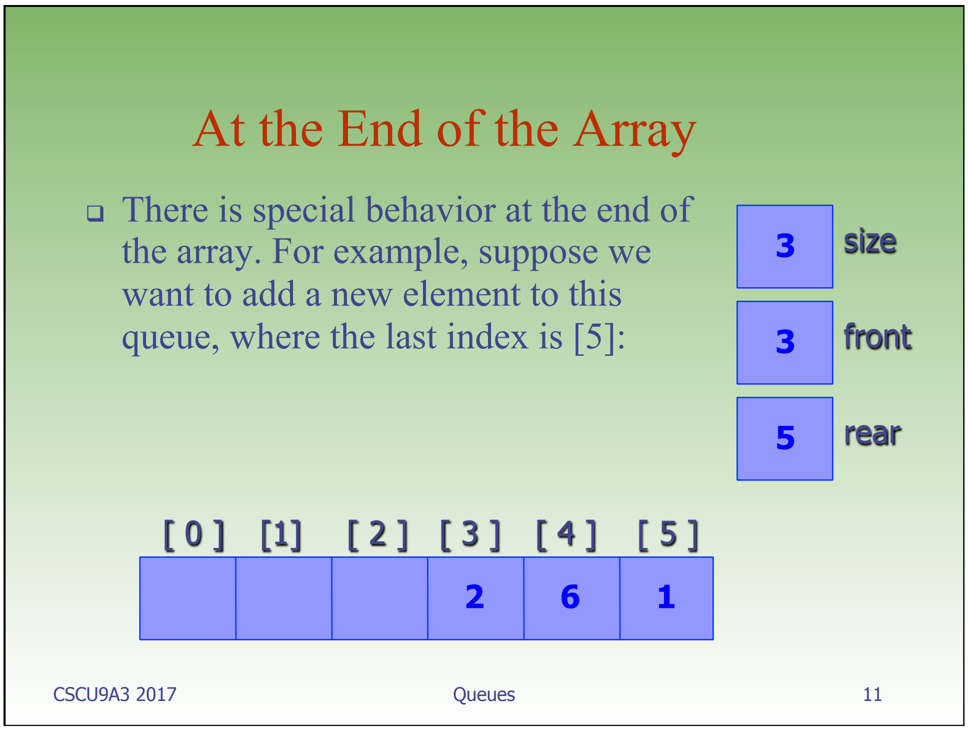
An Enqueue Operation

- When an element enters the queue, size is incremented, and last changes, too.

3	size
1	front
3	rear

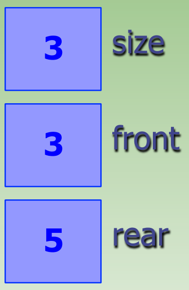
[0]	[1]	[2]	[3]	[4]	[5]	...
	8	6	2			

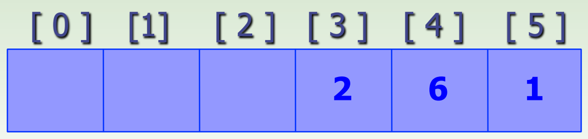
CSCU9A3 2017 Queues 10

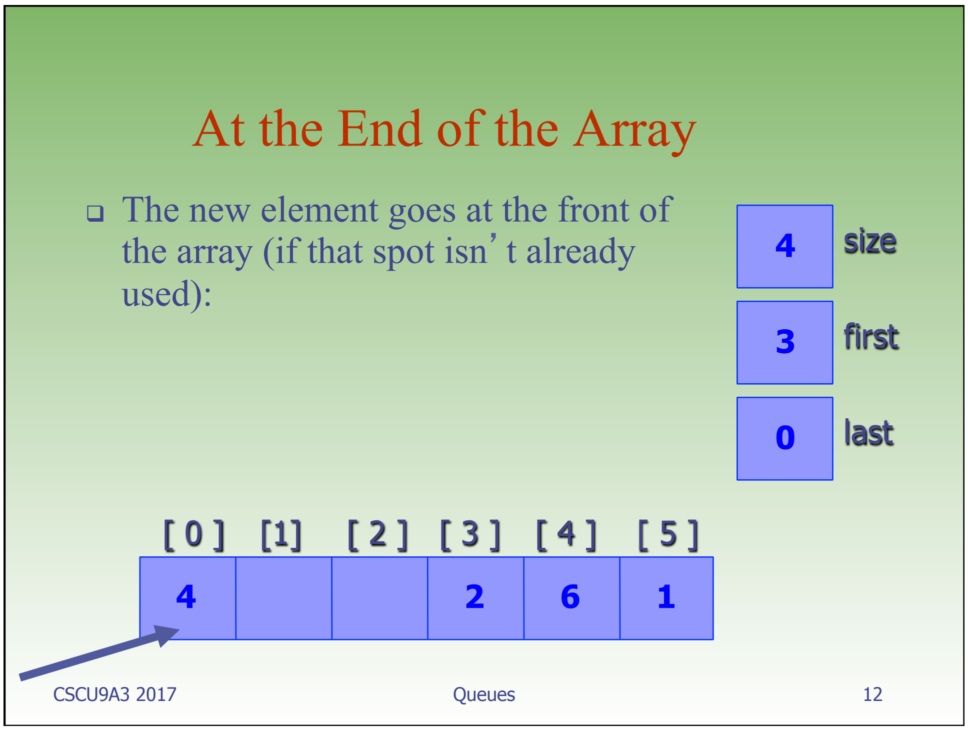


At the End of the Array

- There is special behavior at the end of the array. For example, suppose we want to add a new element to this queue, where the last index is [5]:

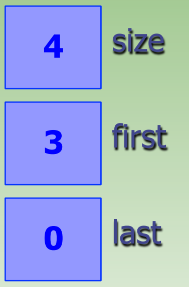


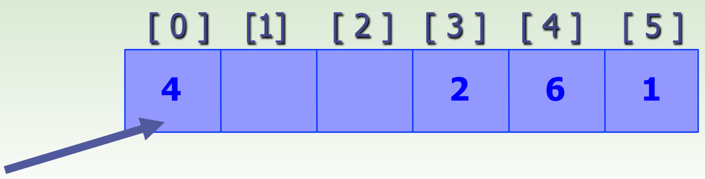


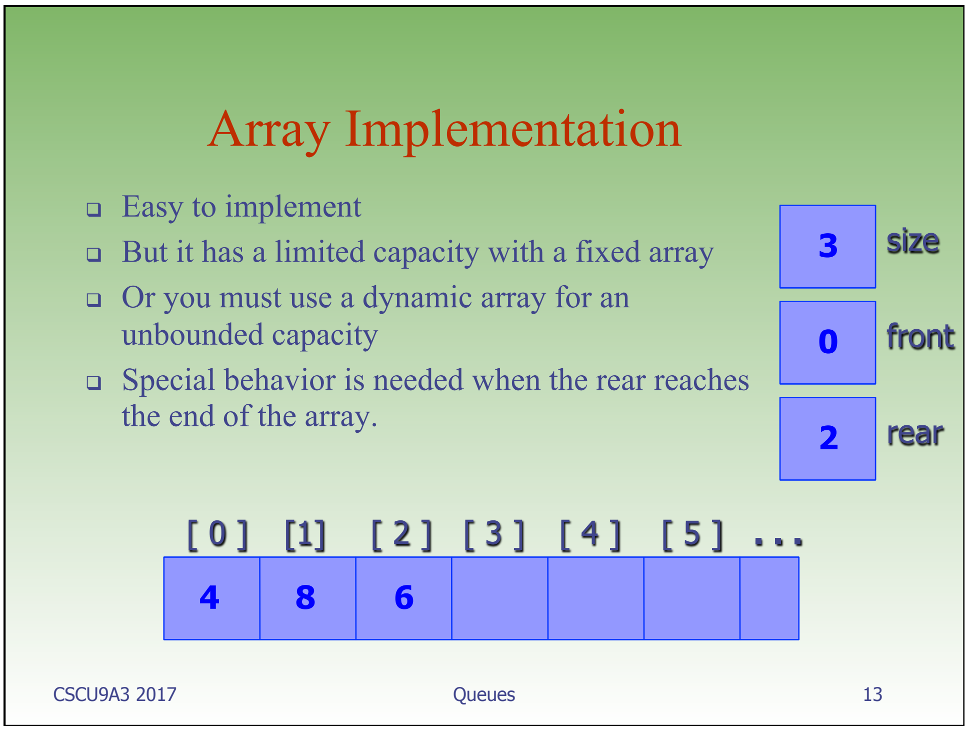


At the End of the Array

- The new element goes at the front of the array (if that spot isn’t already used):



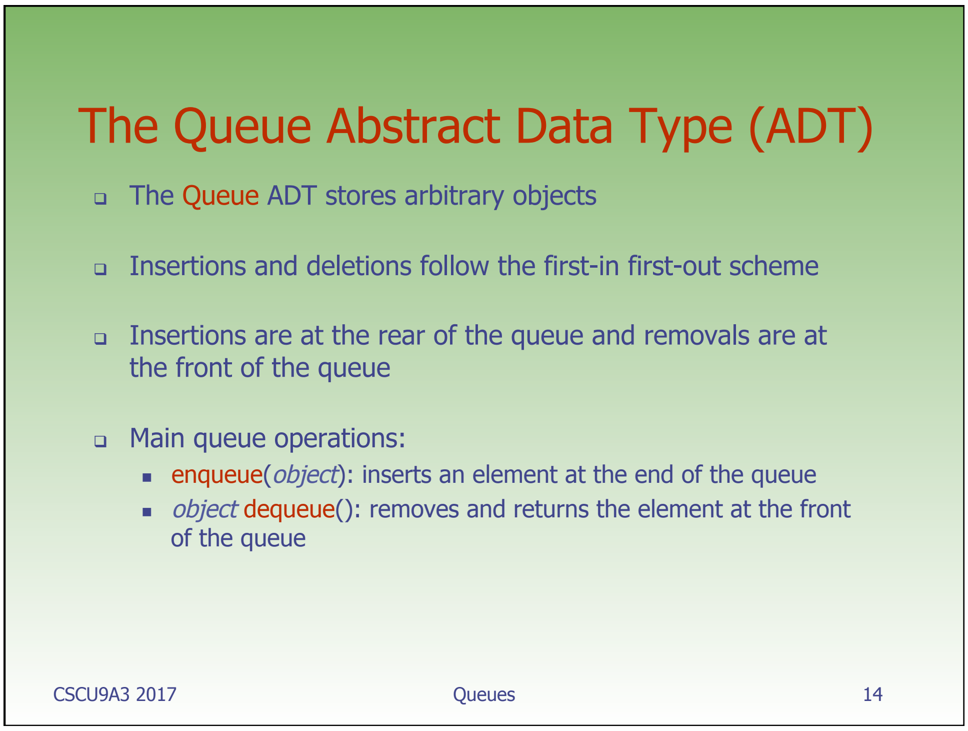




Array Implementation

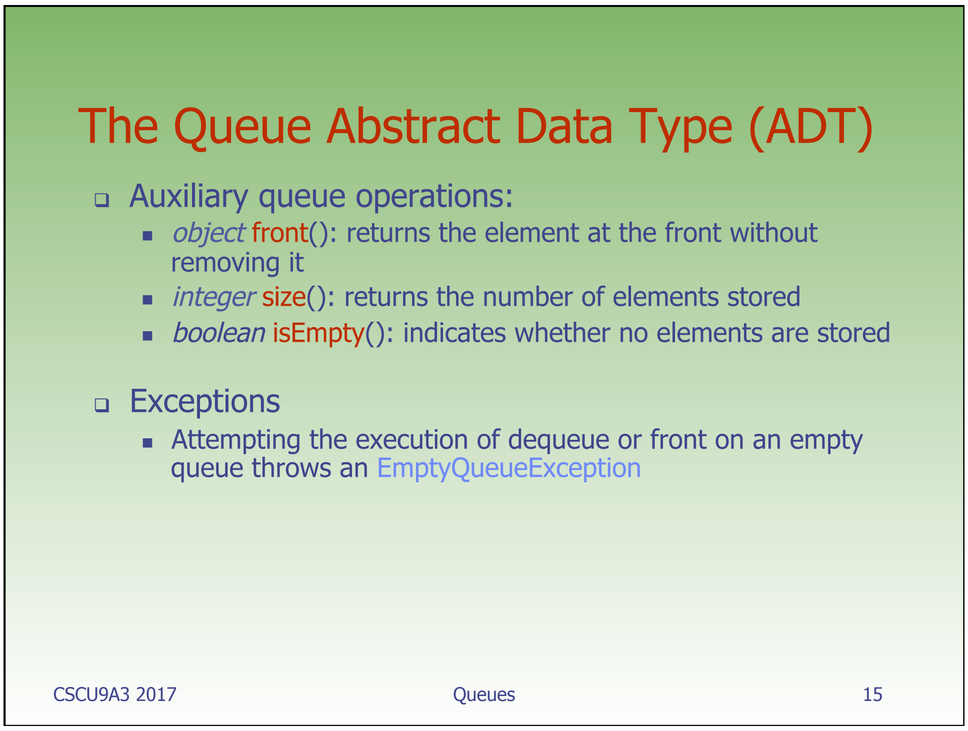
- Easy to implement
- But it has a limited capacity with a fixed array
- Or you must use a dynamic array for an unbounded capacity
- Special behavior is needed when the rear reaches the end of the array.

3 size
0 front
2 rear
[0] [1] [2] [3] [4] [5] ...
4 8 6



The Queue Abstract Data Type (ADT)

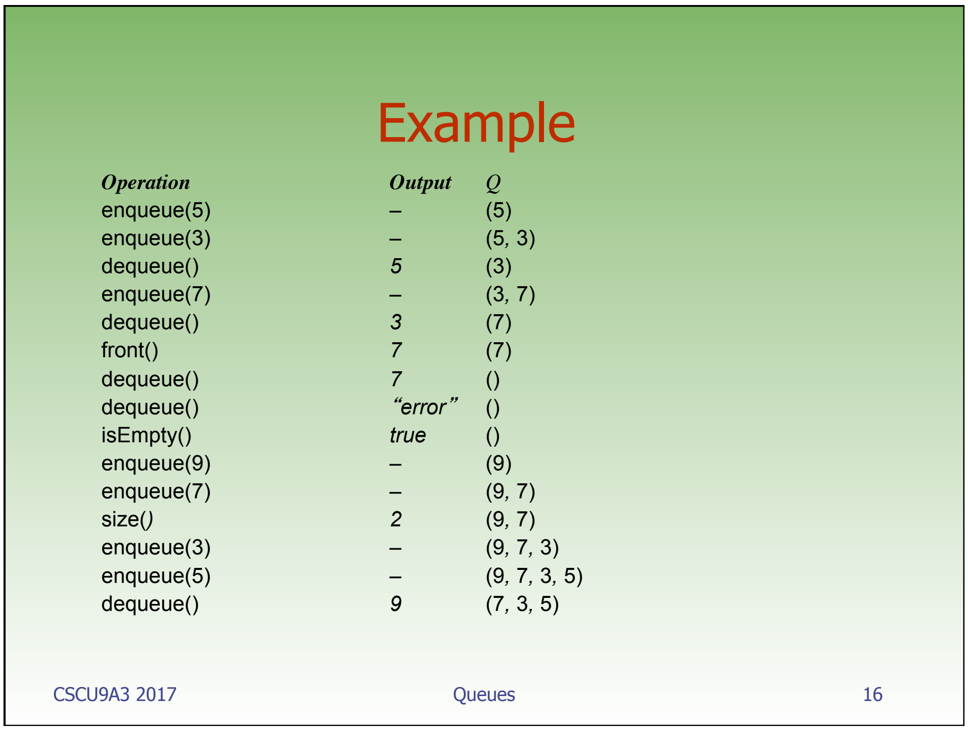
- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme
- Insertions are at the rear of the queue and removals are at the front of the queue
- Main queue operations:
  - enqueue(*object*): inserts an element at the end of the queue
  - *object* dequeue(): removes and returns the element at the front of the queue



The Queue Abstract Data Type (ADT)

- Auxiliary queue operations:
  - *object* front(): returns the element at the front without removing it
  - *integer* size(): returns the number of elements stored
  - *boolean* isEmpty(): indicates whether no elements are stored
- Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an EmptyQueueException

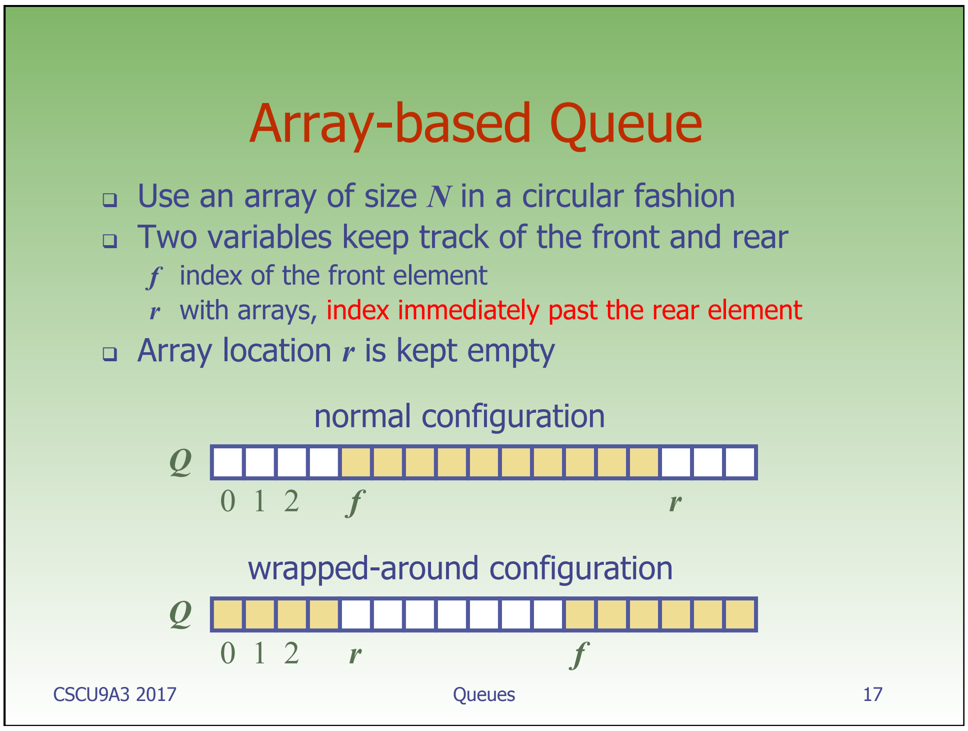
CSCU9A3 2017 Queues 15



Example

Operation	***Output***	***Q***
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	*5*	(3)
enqueue(7)	–	(3, 7)
dequeue()	*3*	(7)
front()	*7*	(7)
dequeue()	*7*	()
dequeue()	*"error"*	()
isEmpty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	*2*	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	*9*	(7, 3, 5)

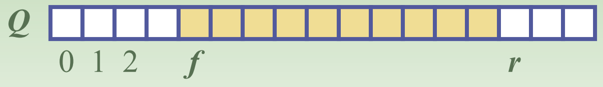
CSCU9A3 2017 Queues 16



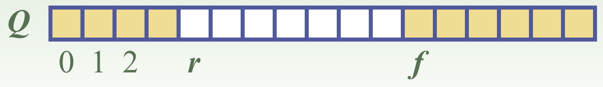
Array-based Queue

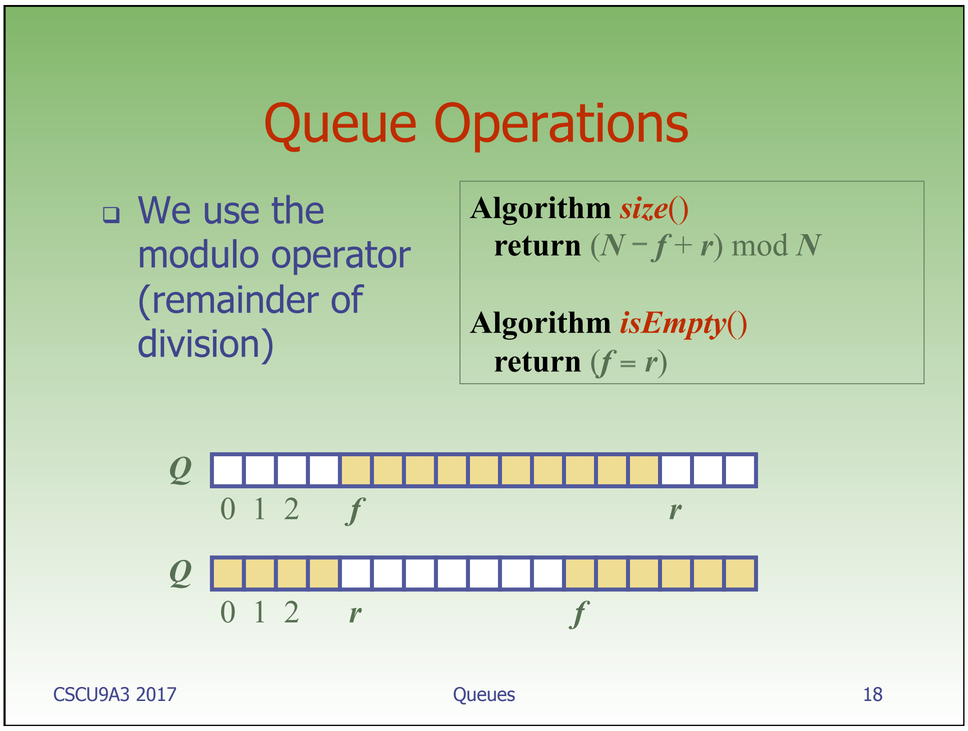
- Use an array of size N in a circular fashion
- Two variables keep track of the front and rear
  - f index of the front element
  - r with arrays, index immediately past the rear element
- Array location r is kept empty

normal configuration



wrapped-around configuration



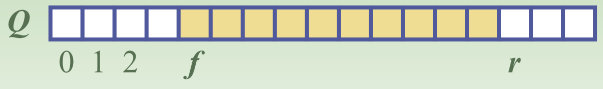


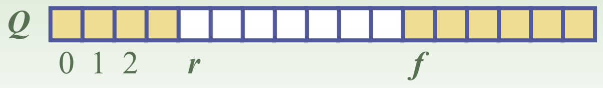
Queue Operations

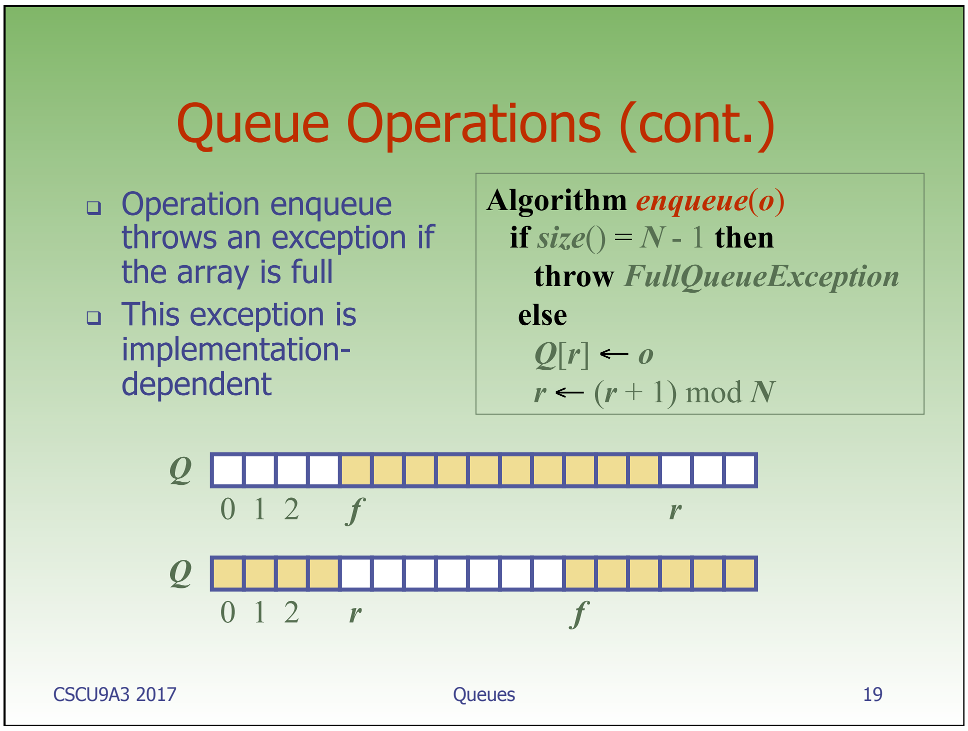
- We use the modulo operator (remainder of division)

```
Algorithm size()
    return $(N - f + r) \bmod N$

Algorithm isEmpty()
    return $(f = r)$
```



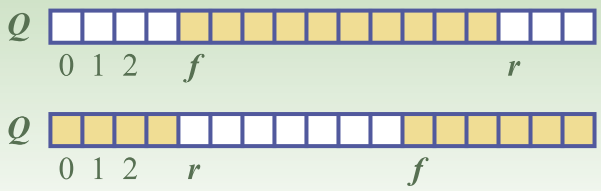


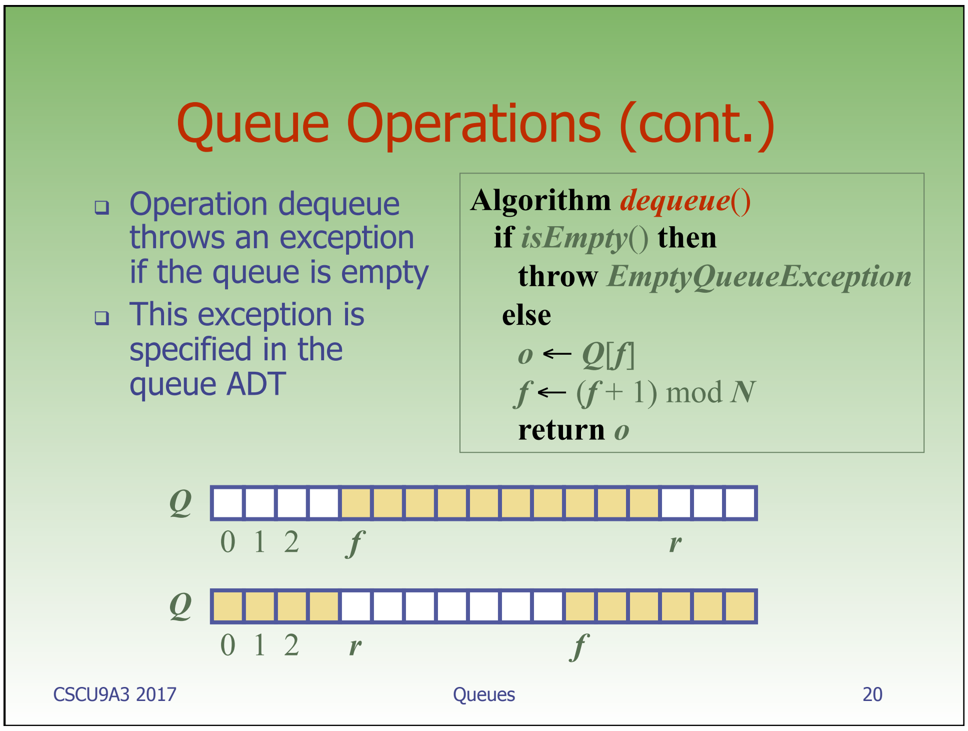


Queue Operations (cont.)

- Operation enqueue throws an exception if the array is full
- This exception is implementation-dependent

```
Algorithm enqueue(o)
  if size() = N - 1 then
    throw FullQueueException
  else
    Q[r] ← o
    r ← (r + 1) mod N
```

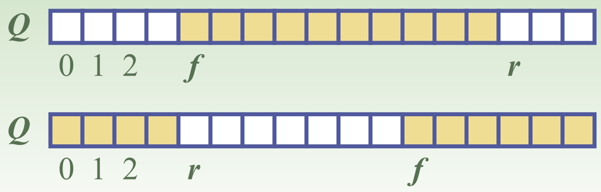


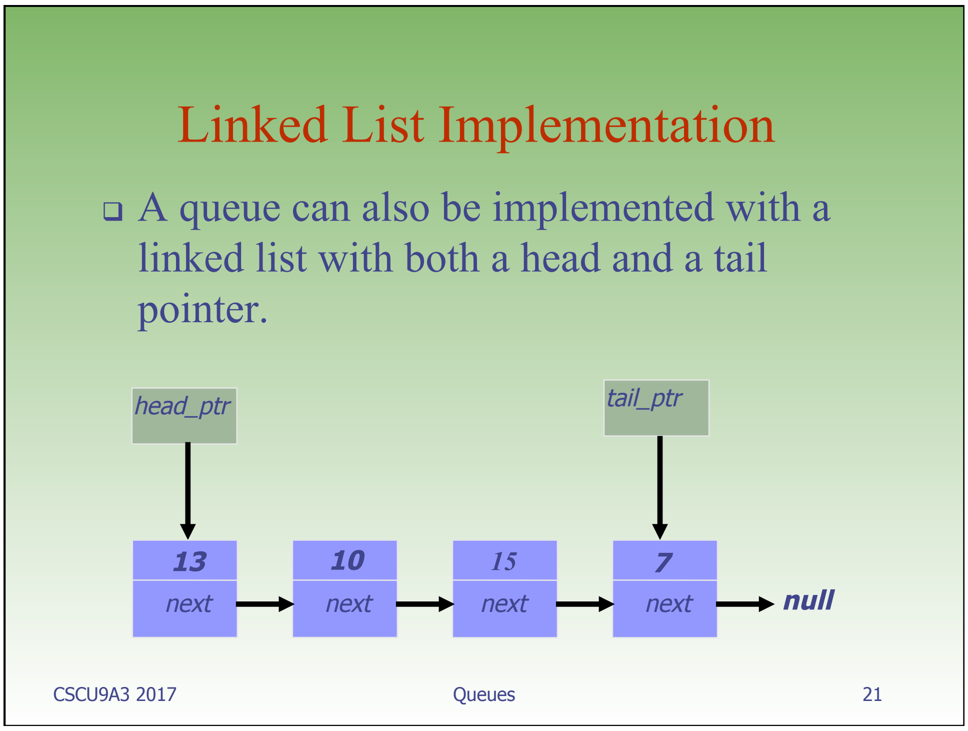


Queue Operations (cont.)

- Operation dequeue throws an exception if the queue is empty
- This exception is specified in the queue ADT

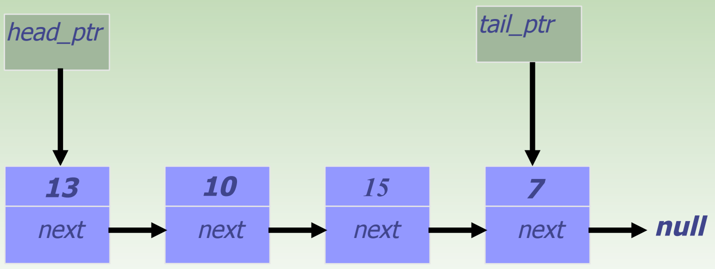
```
Algorithm dequeue()
  if isEmpty() then
    throw EmptyQueueException
  else
    o ← Q[f]
    f ← (f + 1) mod N
    return o
```



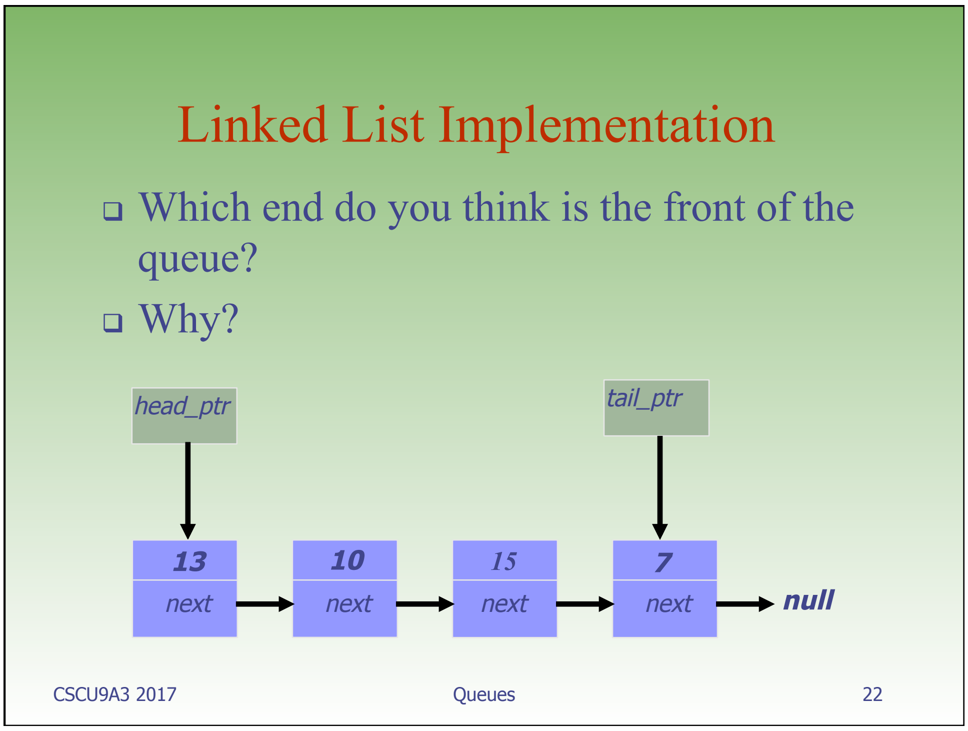


Linked List Implementation

- A queue can also be implemented with a linked list with both a head and a tail pointer.

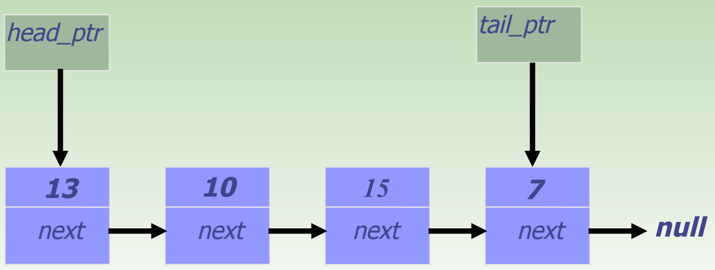


CSCU9A3 2017 Queues 21

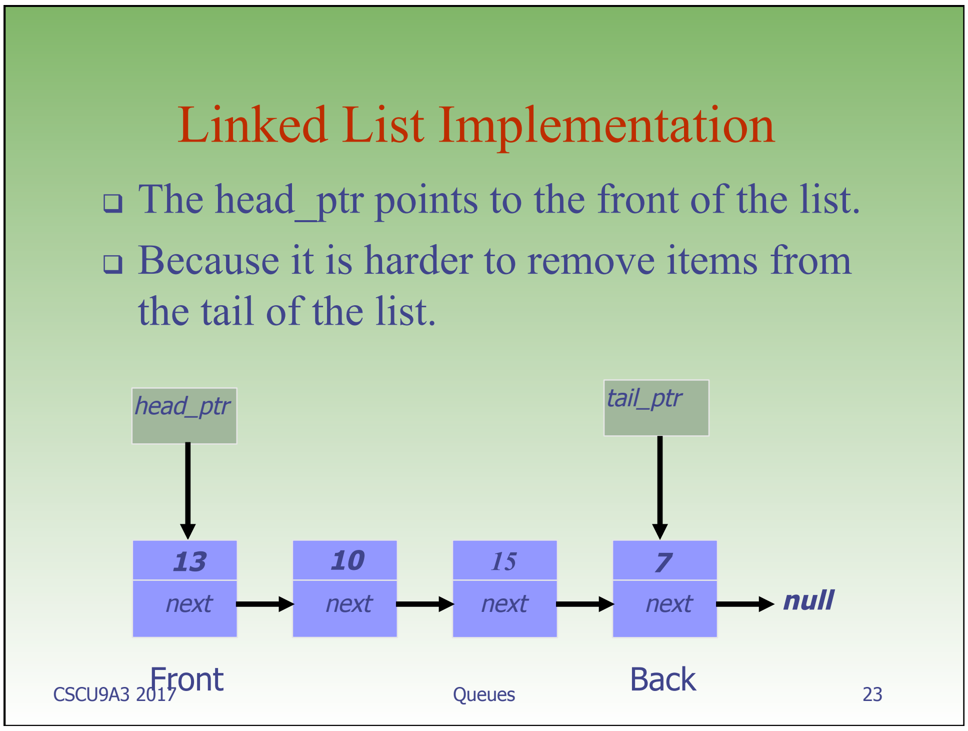


Linked List Implementation

- Which end do you think is the front of the queue?
- Why?

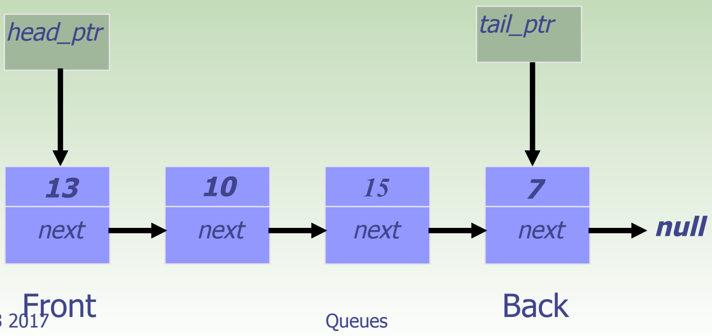


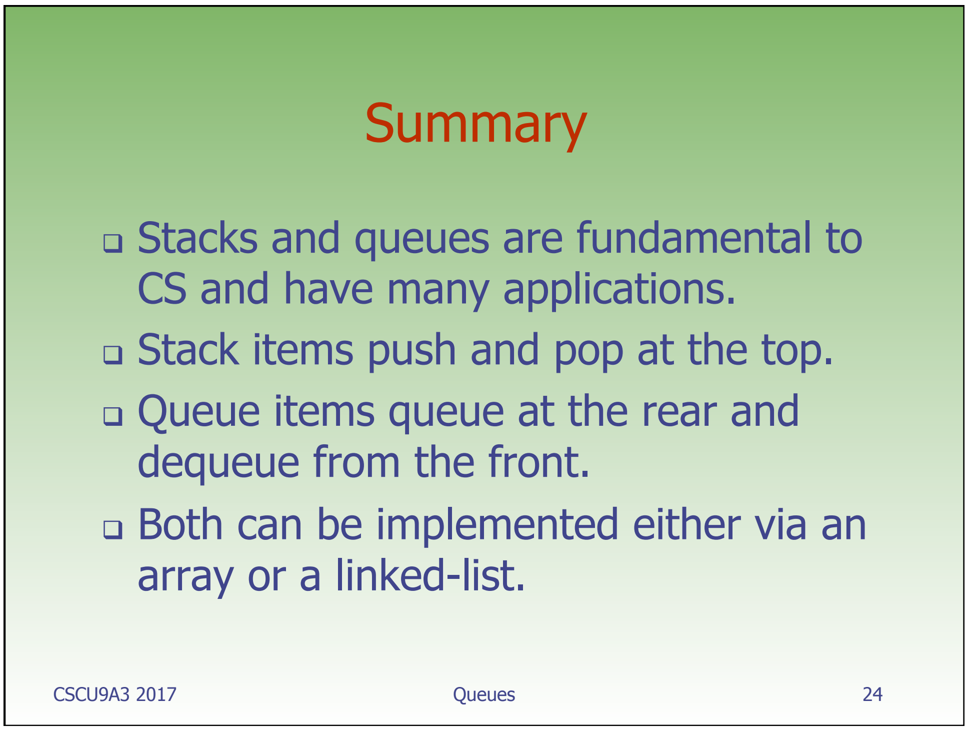
CSCU9A3 2017 Queues 22



Linked List Implementation

- The head_ptr points to the front of the list.
- Because it is harder to remove items from the tail of the list.





Summary

- Stacks and queues are fundamental to CS and have many applications.
- Stack items push and pop at the top.
- Queue items queue at the rear and dequeue from the front.
- Both can be implemented either via an array or a linked-list.