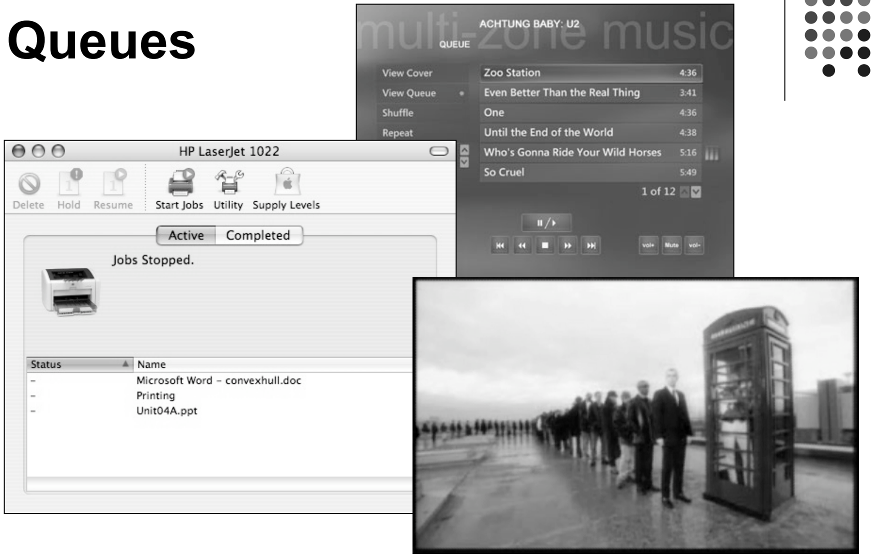
More Linear Data Structures

3B

Queues

Queues



Zoo Station	4:36
Even Better Than the Real Thing	3:41
One	4:36
Until the End of the World	4:38
Who's Gonna Ride Your Wild Horses	5:16
So Cruel	5:49

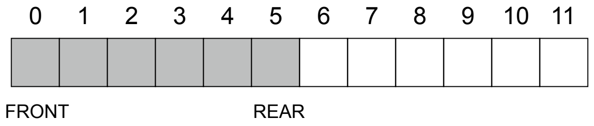
Jobs Stopped.

Status	Name
–	Microsoft Word – convexhull.doc
–	Printing
–	Unit04A.ppt

Queue Operations

```
public interface FIFOQueue<E> {
  public void enqueue(E element);
      // Insert element at rear of queue
  public boolean isEmpty();
      // Is the queue empty?
  public E dequeue();
      // Remove element from front of queue
  public E peek();
      // Examine element at front of queue
}
```

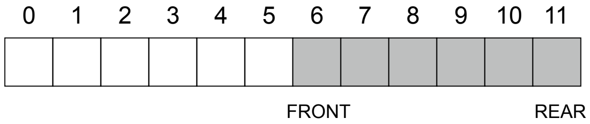
Queues using an array



Store the elements of the queue from front to rear in order in the array.

What happens if we store the front of the queue always in position 0?

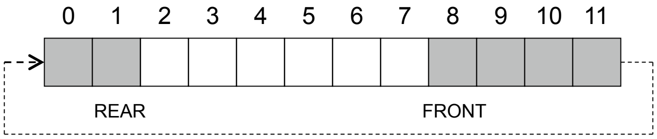
Queues using an array



Store the elements of the queue from front to rear in order in the array.

What happens if we store the rear of the queue always in last position in the array?

Queues using a “circular” array



Allow the queue to “wrap around” from the last cell of the array back to the first cell so shifting is not necessary.

If the array is full, we can reallocate the array and copy the queue data into the new array starting at position 0 again.

Array Implementation

Fields

```
public class ArrayQueue<E> implements
  FIFOQueue<E> {

  private E[] dataArray;
  private int front;
  private int rear;
  private int numElements;

  // methods (next slides)

}
```

indices of front and rear queue elements in array

Array Implementation

Constructor & isEmpty

```
  public ArrayQueue() {
   dataArray = (E[])new Object[1];
   front = -1;
   rear = -1;
   numElements = 0;
  }

  public boolean isEmpty() {
   return (numElements == 0);
  }
```

indicates an empty queue

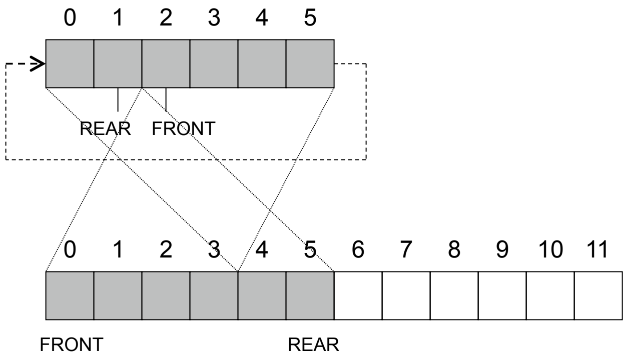
Array Implementation

enqueue

```
public void enqueue(E element) {
 if (______________________________________)
    reallocate();
 rear = (rear+1) % dataArray.length;
 dataArray[rear] = element;
 if (front == -1)
    front = rear;
 numElements++;
}
```

Array Implementation

reallocate



Array Implementation

reallocate

```
private void reallocate() {
 E[] newArray =
      (E[])new Object[numElements*2];
 int j = front;
 for (int i=0; i<numElements; i++) {
    newArray[i] = dataArray[j];
    j = _________________________ ;
 }
 front = 0;
 rear = numElements-1;
 dataArray = newArray;
}
```

Array Implementation

dequeue

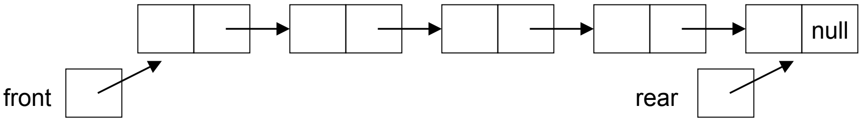
```
public E dequeue() {
 if (__________________________)
   throw new NoSuchElementException();
 E element = dataArray[front];
 dataArray[front] = null;
 if (front == rear) {
    ______________________________________
 } else
    ______________________________________
 numElements--;
 return element;
}
```

Array Implementation

peek

```
public E peek() {
    if (____________________________)
      throw new NoSuchElementException();
    return dataArray[front];
}
```

Queues using a singly-linked list



Store the elements of the stack from top to bottom in order in the list.

Why do we need an additional reference to the tail?

Linked List Implementation

Fields

```
public class ListQueue<E> implements
  FIFOQueue<E> {

  private Node<E> front;
  private Node<E> rear;


  // methods (next slides)

}
```

references to nodes with front and rear queue elements in list

Linked List Implementation

Constructor & isEmpty

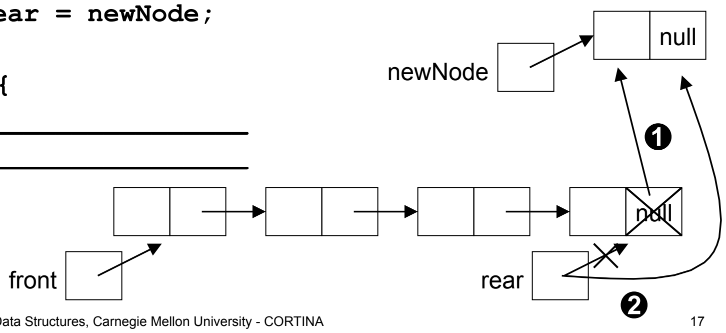
```
  public ListQueue() {
      front = null;
      rear = null;
  }


  public boolean isEmpty() {
      return (front == null);
  }
```

indicates an empty queue

Linked List Implementation

enqueue

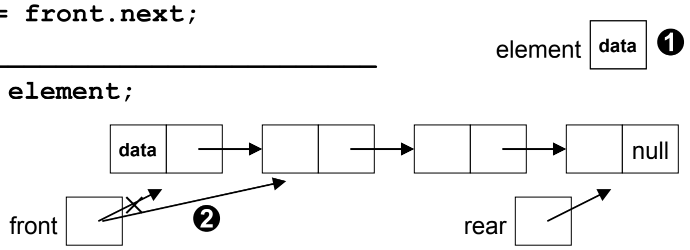
```
public void enqueue(E element) {
 Node<E> newNode = new Node<E>(element);
 if (rear != null) {
  ❶ rear.next = newNode;
  ❷ rear = newNode;
 }
 else {
     ________________________
     ________________________
 }
}
```



Linked List Implementation

dequeue

```
public E dequeue() {
 if (front == null)
   throw new NoSuchElementException();
❶ E element = front.data;
❷ front = front.next;
 __________________________________
 return element;
}
```



Linked List Implementation

peek

```
public E peek() {
    if (front == null)
      throw new NoSuchElementException();
    return front.data;
}
```

Other uses for queues

- Printer queues
- Packet router
- Simulating a queuing system
  - Supermarket checkout lanes
  - Highway traffic congestion models
  - Internet traffic
- Priority Queues: emergency room