CSC 1052 – Algorithms & Data Structures II:

Linked Queues

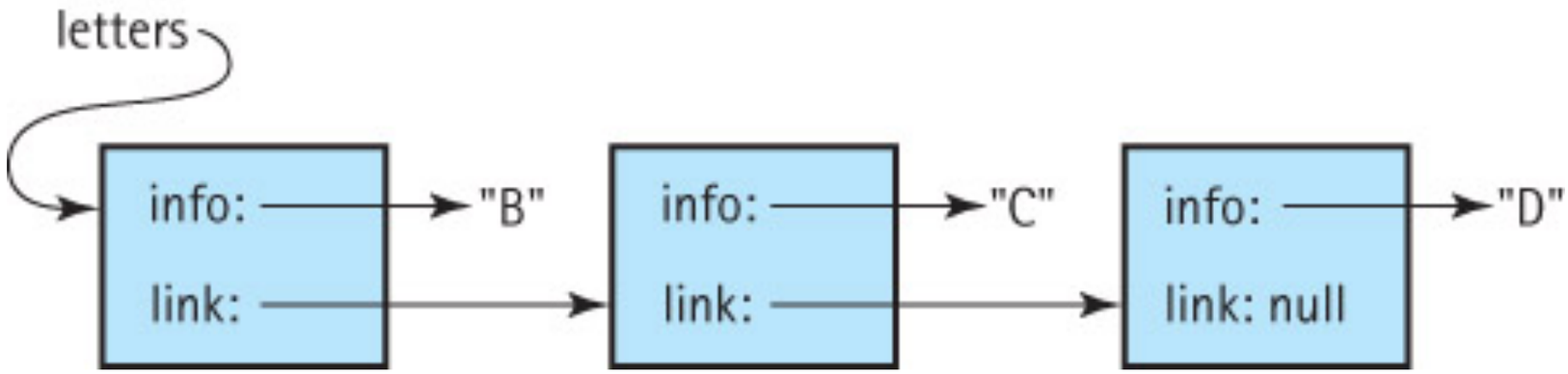
Professor Henry Carter
Spring 2017

Recap

- A queue simulates a waiting line, where objects are removed in the same order they are added
- The primary operations are:
  - Enqueue
  - Dequeue
  - Size, isEmpty, isFull
- Array-based implementation requires several technical tradeoffs
  - Fixed front vs floating front
  - Fixed size vs expanding size

Recall: Linked Lists

- Nodes containing data and a link to the next object
- Stack implementation was simple since only one end is accessed
- What could we change for the queue?
- How will this improve on the array implementation?



Code Setup

- Create a project
- Create the LinkedQueue class
- Create the QueueDriver to test operations

Linked Queue

- Maintains two pointers:
  - Head
  - Tail
- Maintains queue size for convenience

- How do we:
  - Initialize?
  - Implement enqueue?
  - Implement dequeue
  - Implement helper methods?

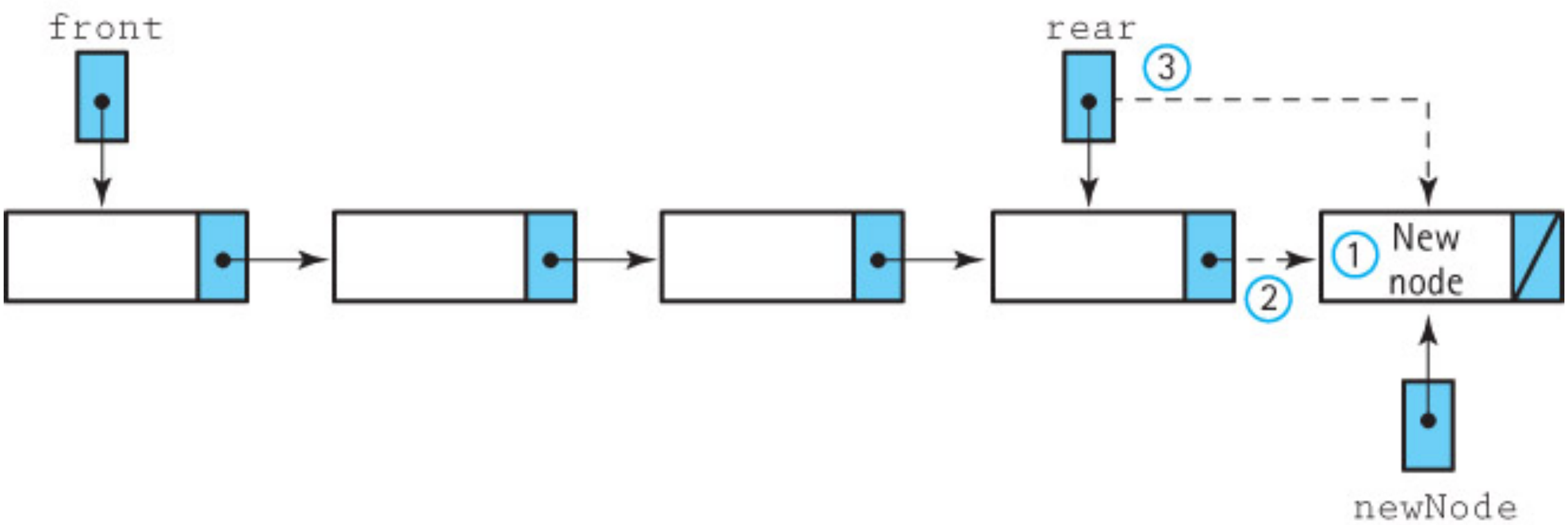
Initialization Code

```
package ch04.queues;
import support.LLNode;

public class LinkedQueue<T> implements QueueInterface<T>
{
  protected LLNode<T> front;     // reference to the front of this queue
  protected LLNode<T> rear;      // reference to the rear of this queue
  protected int numElements = 0; // number of elements in this queue

  public LinkedQueue()
  {
    front = null; rear = null;
  }
. . .
```

Enqueue

- What needs to happen?
- What are the edge cases?
- What is the expected OOG?

Enqueue Illustration



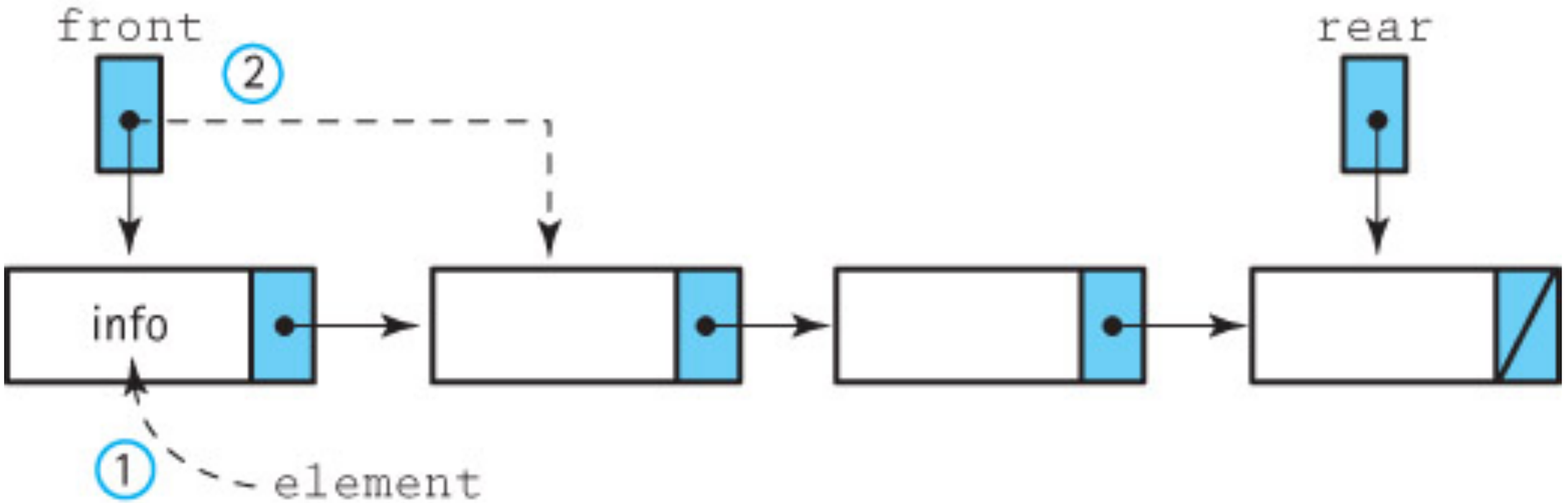
Enqueue Code

```
public void enqueue(T element)
// Adds element to the rear of this queue.
{
  LLNode<T> newNode = new LLNode<T>(element);
  if (rear == null)
    front = newNode;
  else
    rear.setLink(newNode);
  rear = newNode;
  numElements++;
}
```

Dequeue

- What needs to happen?
- What are the edge cases?
- What is the expected OOG?

Dequeue Illustration



Dequeue Code

```
public T dequeue()
// Throws QueueUnderflowException if this queue is empty,
// otherwise removes front element from this queue and returns it.
{
  if (isEmpty())
    throw new QueueUnderflowException("Dequeue attempted on empty queue.");
  else
  {
    T element;
    element = front.getInfo();
    front = front.getLink();
    if (front == null)
      rear = null;
    numElements--;
    return element;
  }
}
```

Exceptions!

- Throw an underflow exception for dequeue on an empty queue
- Do we need an overflow exception?

Helpers

- isEmpty()
  - What do we check?
- isFull()
  - What do we return?
- Size()
  - What do we return?

Variation: Circular Linked List

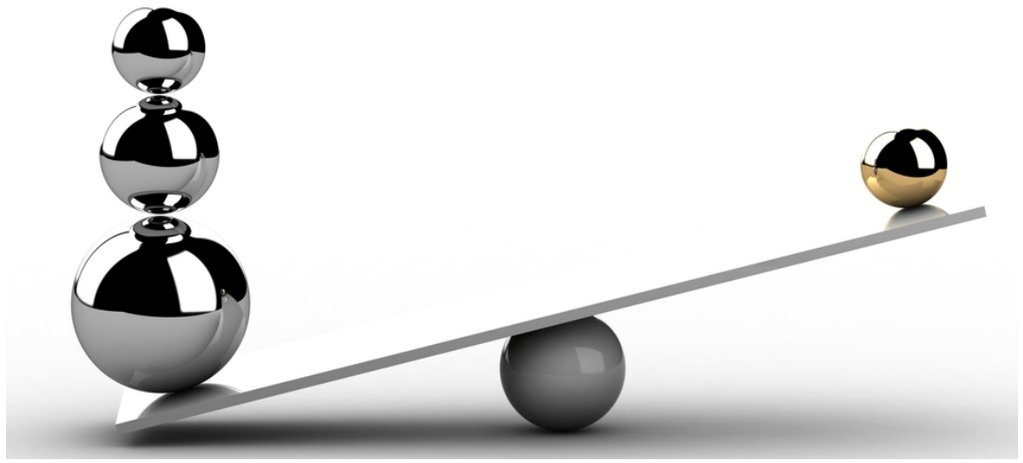
- Could we reduce the pointer overhead to 1?
- A circularly linked queue links the tail node back to the head
- Which node (head or tail) do we keep a pointer to?

Exercise

- Exercise: convert your LinkedQueue to a circularly linked queue

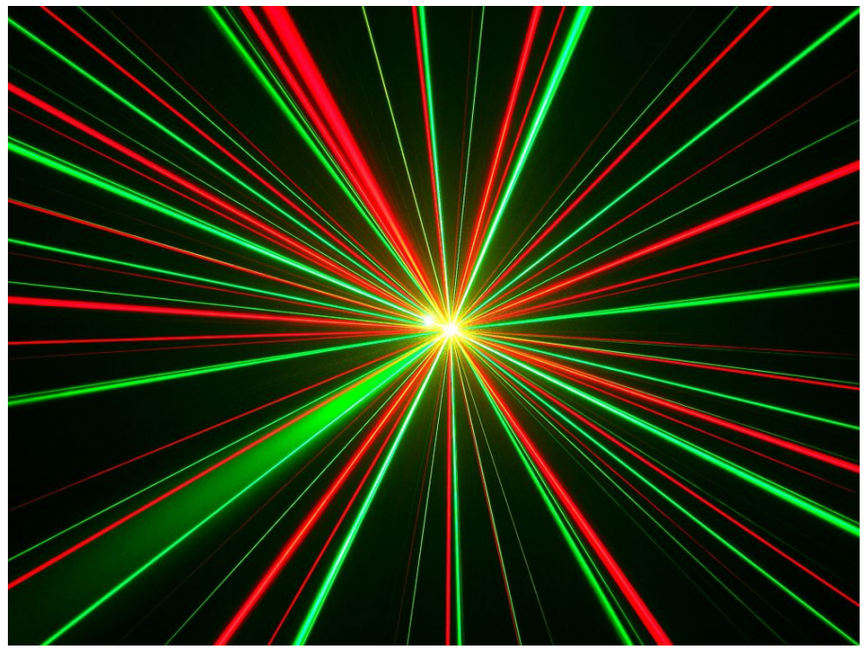
Efficiency Comparison

- What was the tradeoff between arrays and linked lists?
- Is the tradeoff different here?
- Is O(1) equivalent for both?
- What about initialization costs?



Space Efficiency: How Many Pointers?

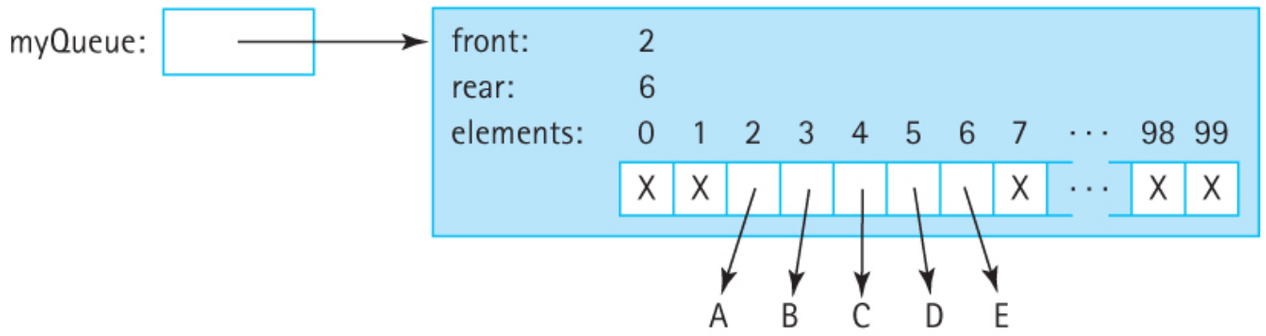
- Are required per element?
- Are allocated at initialization?
- When do we “break even”?



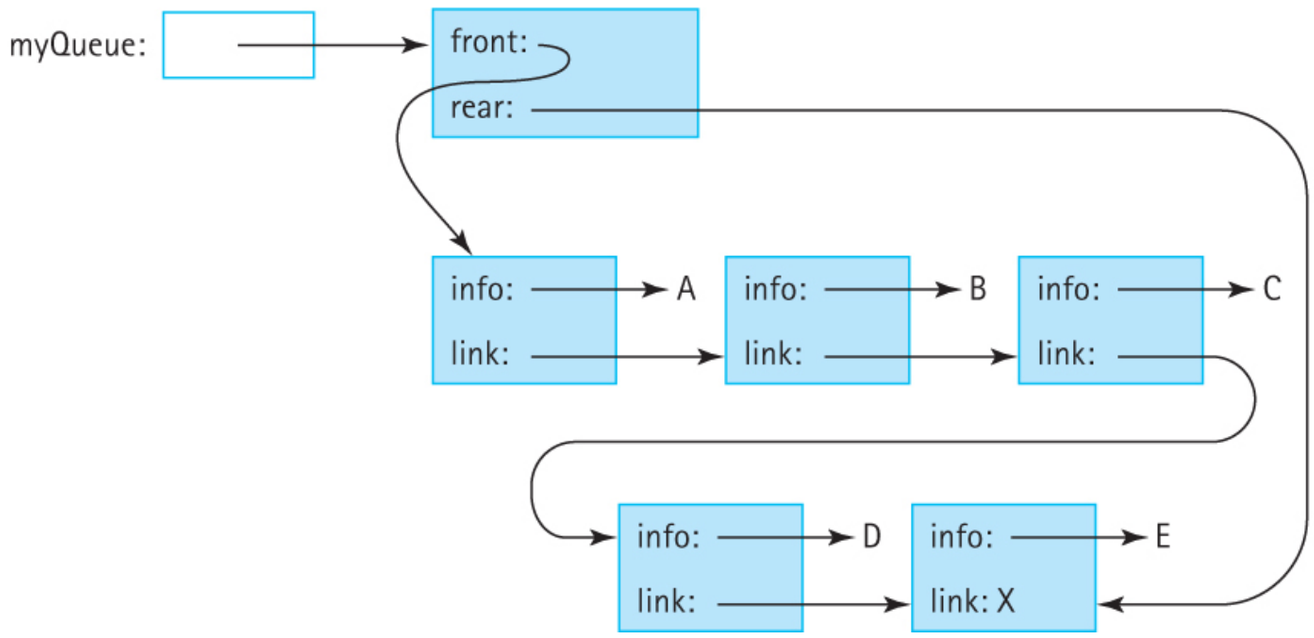
Space Efficiency Illustration

Queues with
Maximum size 100 (Array-Based)
Current size 5
x = null

Array–Based Implementation



Linked Implementation



Recap

- Linked Lists allow for convenient implementation of queues
- Maintaining a head and tail simplifies implementation (but is not necessary)
- Array/Linked List tradeoffs between time and space are consistent with previous examples
  - Large queue relative to max size: array
  - Small or variable size queue: linked list

Next Time...

- Dale, Joyce, Weems Chapter 4.7-4.8
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/