Data Structures

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Queue (link based)

Goals

- Describe the structure of a queue and its operations at a logical level
- Demonstrate the effect of queue operations using a particular implementation of a queue
- Implement the Queue ADT, using both a an array-based implementation and a linked implementation
- Discuss Big O runtimes of operations for array-based and linked implementations.

Queue Linked Implementation

- Keep pointers to the front and rear elements.
- Use a node struct to hold the data and a pointer to the next element.

Queue (Linked) - Implementation

Queue

Here is the interface for the Queue ADT:

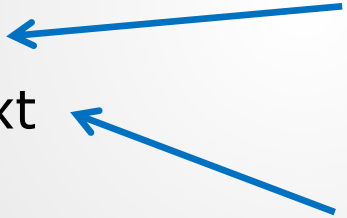
```
public interface Queue {
   boolean isEmpty();
   boolean isFull();
   void enqueue(int item) throws Exception;
   int dequeue() throws Exception;

   void makeEmpty();
}
```

The public interface of a queue should be the same for both the array-based and linked implementations

Node

- The linked stack data structure requires that we keep more information at EACH place inside of it.
- Each item in the queue will be a "Node" (not just the data).
- A node stores the data and a reference to the next node
- It should be defined as an inner class within the QueueLinked class.

```
class Node {
   Declare int data
   Declare Node next
}
```



Data for this node (change data type as necessary to store other types of data)

Points to next node in list

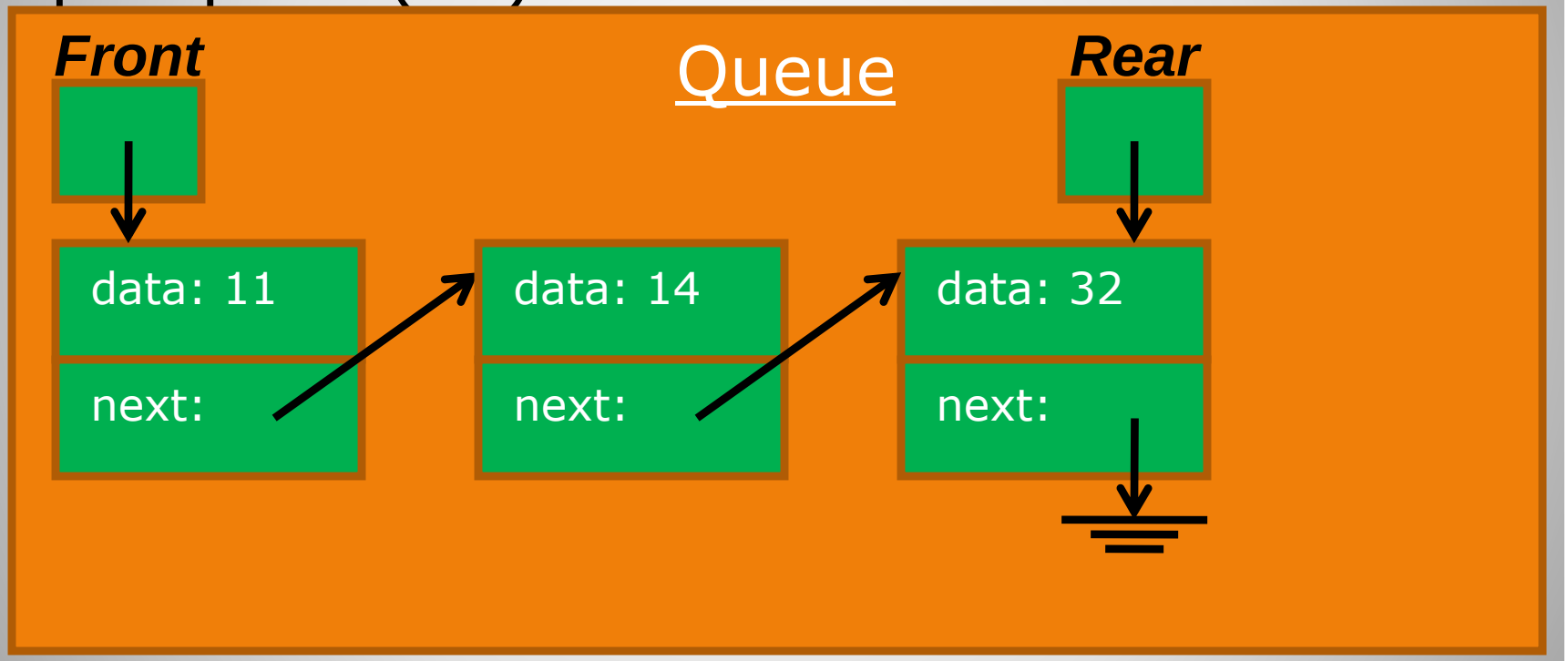
Queue (Linked)

- Link-based **private** members

```
class QueueLinked implements Queue {

   Declare Node front
   Declare Node rear

   // Public members go here…
}
```

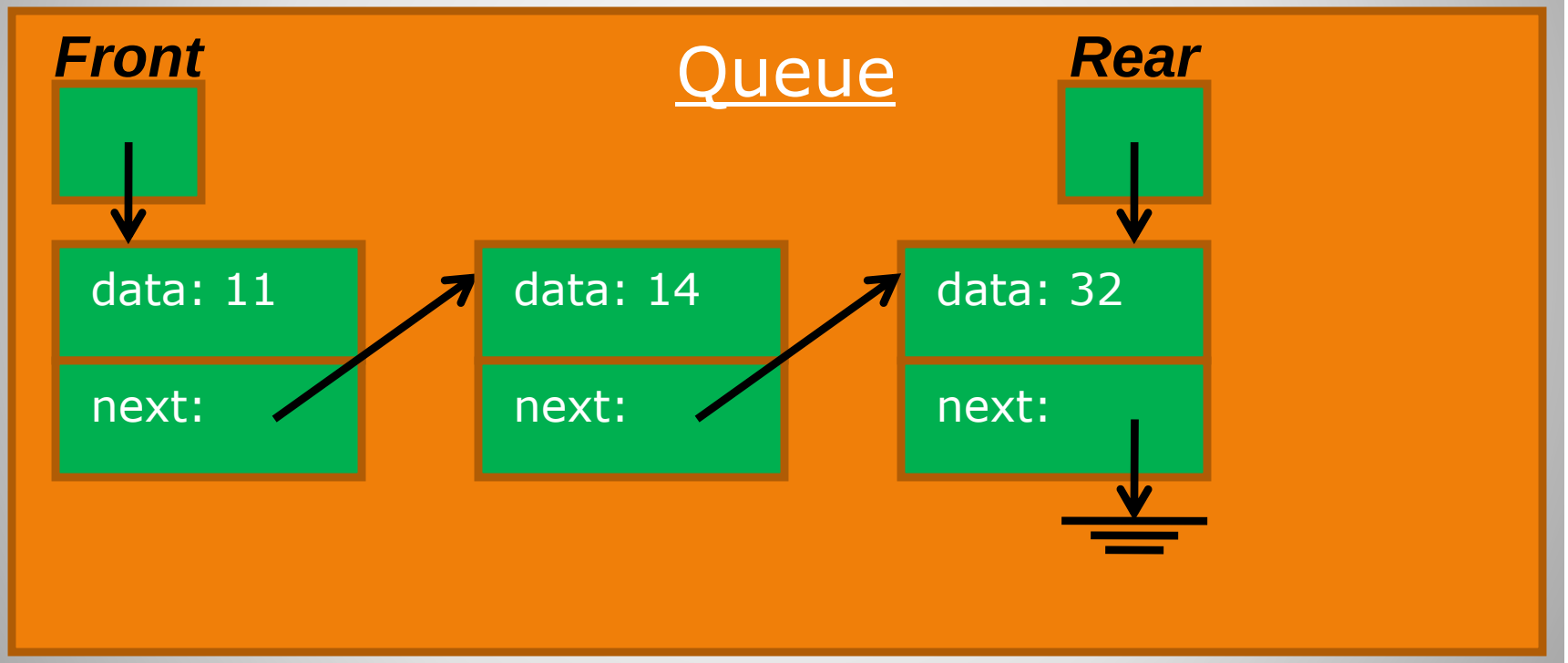
q.enqueue(11)
q.enqueue(14)
q.enqueue(32)

Assume the queue has 3 elements on to it



Queue (Linked)

Where would a new element go?
q.enqueue(77)



Queue (Linked)

Queue (Linked) - Enqueue

You ***MUST*** put the new item at the end of the queue.

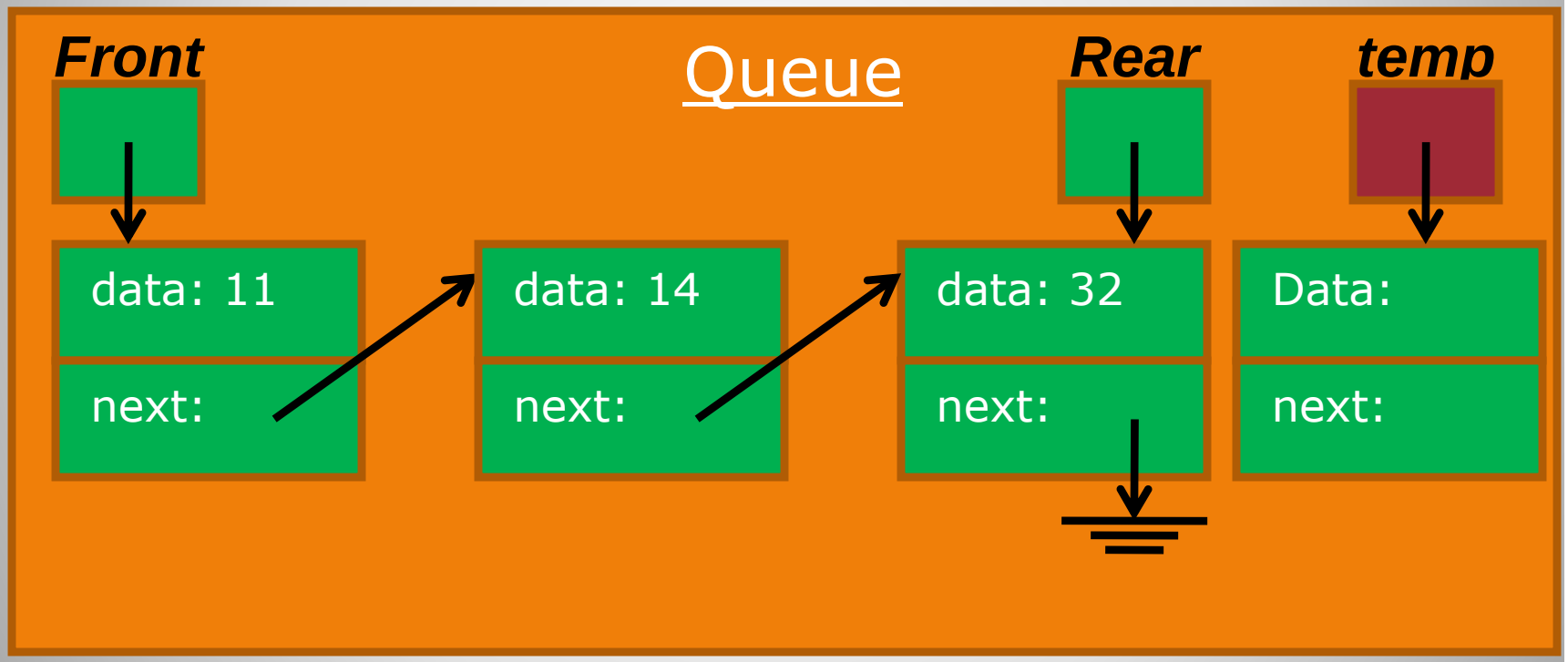
Enqueue Pseudocode

1. Create a new NodeType item (dynamically allocate).
2. Set the fields on the new NodeType item. This means setting the data item and the next pointer. The next pointer should be set to the current top.
3. Set the current last element to point to the new node.

(slightly different if queue is empty)

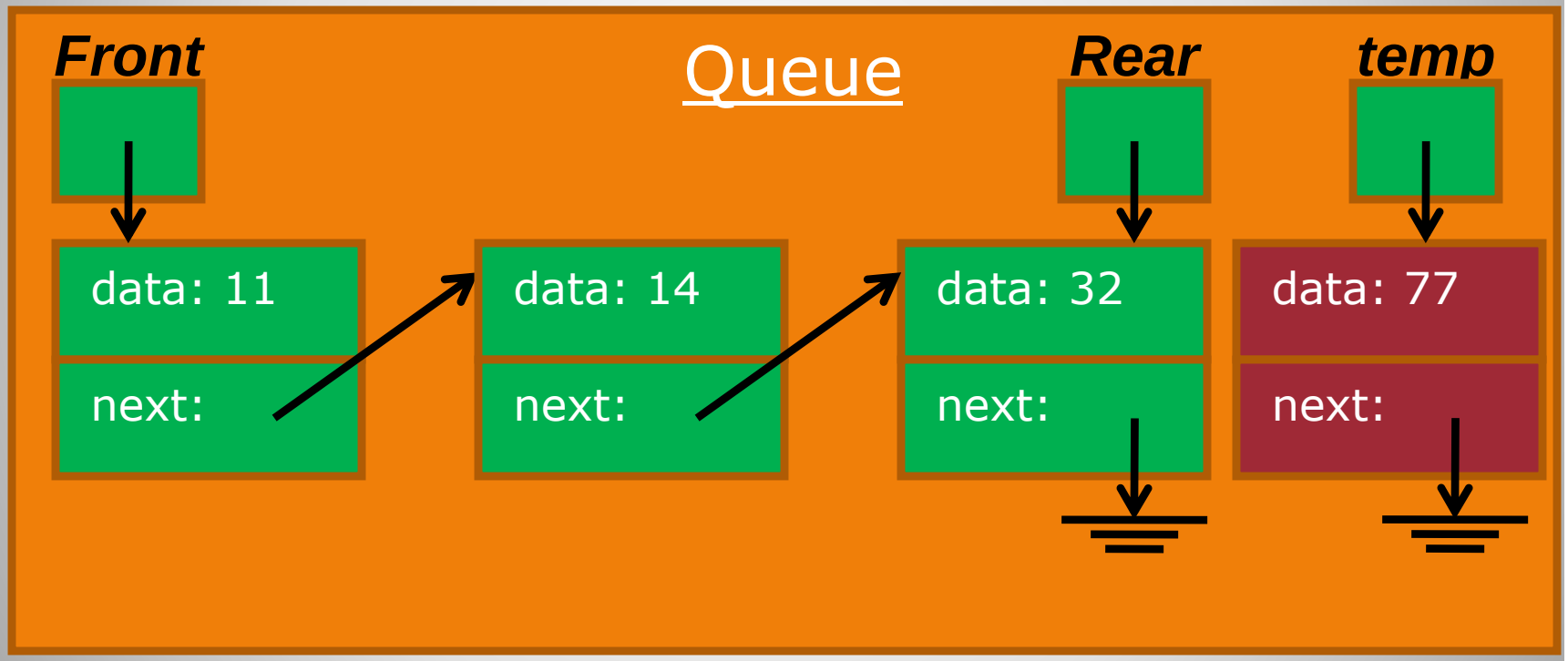
4. Set the rear to the new element.

1. Create a new Node item (dynamically allocate).



Queue (Linked)

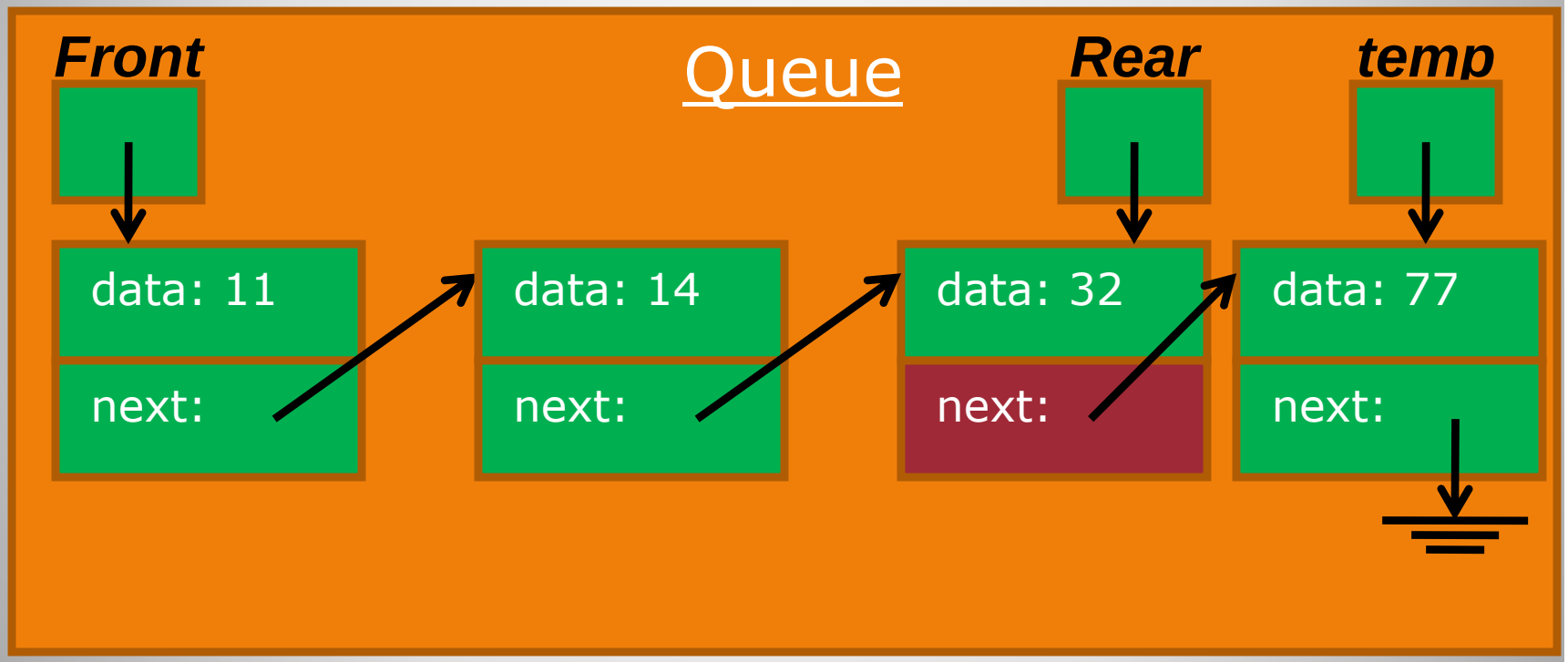
2. Set the fields on the new Node item. This means setting the data item and the next pointer. The next pointer should be set to null because this new node will become the last element in the queue.



Queue (Linked)

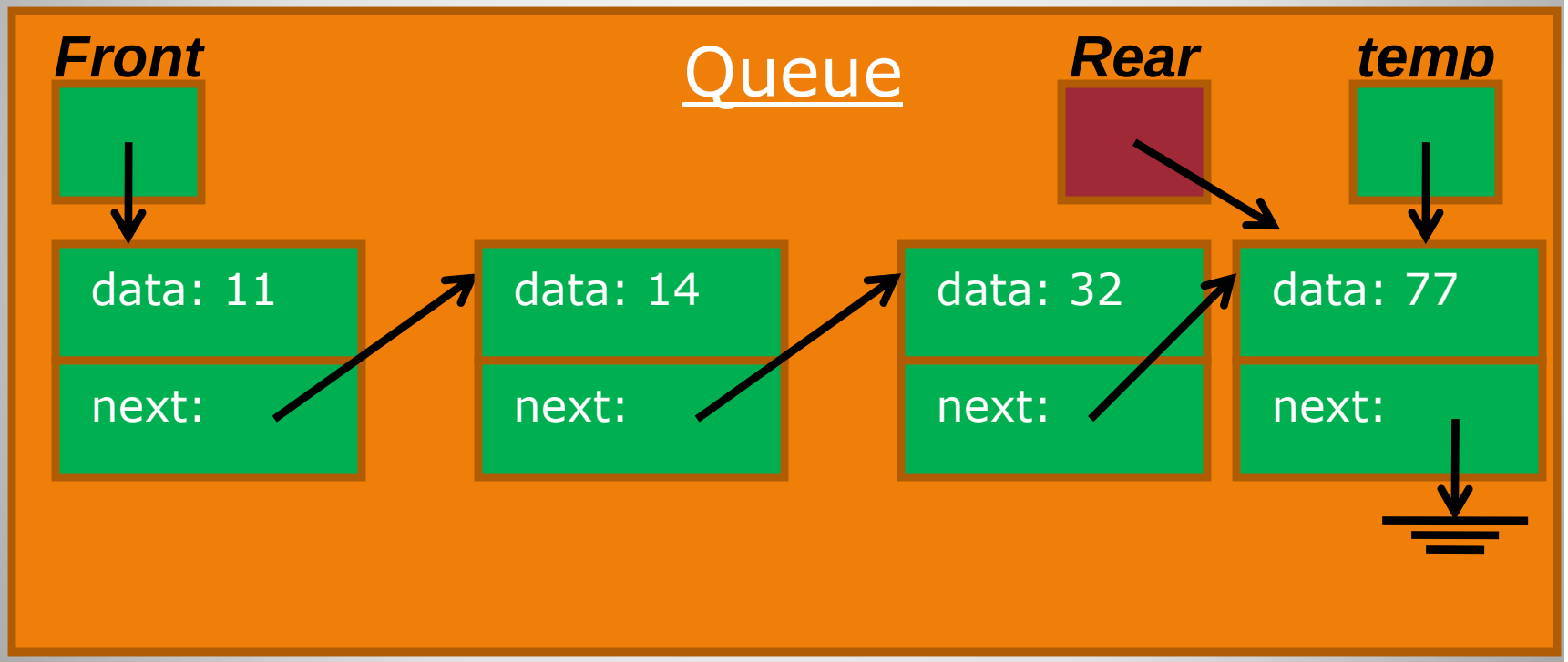
3. Set the current last element to point to the new node.

(slightly different if queue is empty)



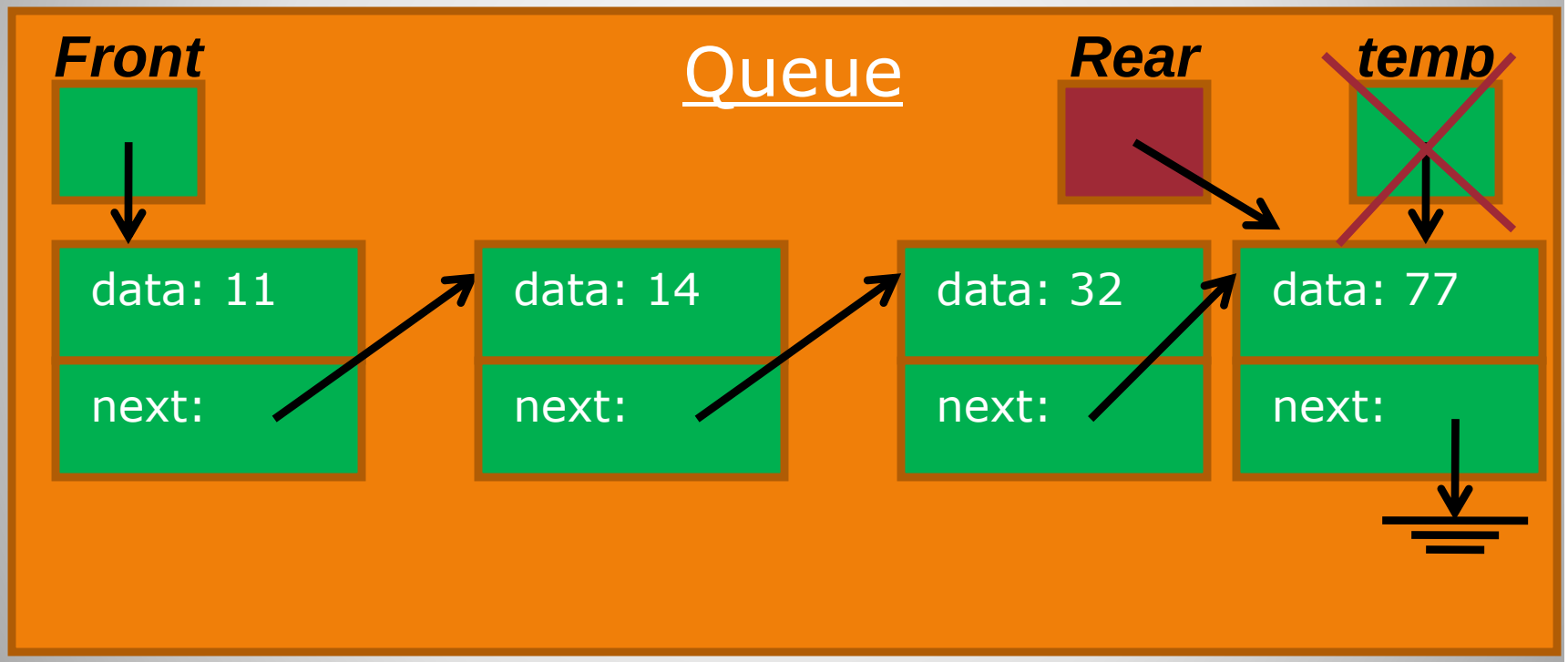
Queue (Linked)

4. Set the rear to the new element.



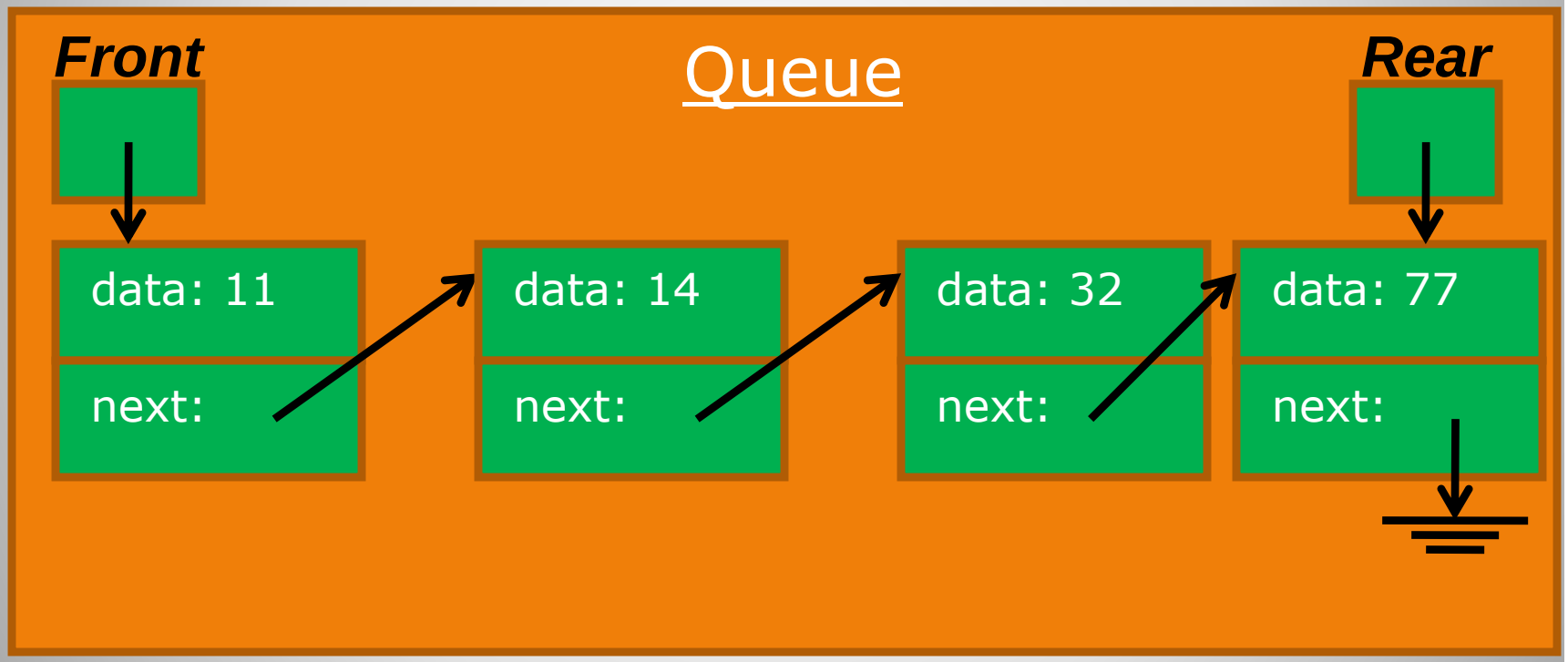
Queue (Linked)

When the enqueue method ends the temp pointer will disappear.



Queue (Linked)

This picture is ***LOGICALLY EQUIVALENT*** to the previous slide!!!



Queue (Linked)

```
enqueue(int item) throws Exception
    if (queue is full)
        throw exception "Full Stack"
    else
        Declare Node temp
        Set temp to new node instance
        Set temp.data to item
        Set temp.next to null
        if (rear equals null)
            Set front to temp
        else
            Set rear.next to temp


        Set rear to temp
```

1. Allocate a new node

2. Set values on the new node

3. If rear is null it is adding to an empty queue so front must also be set to the new node

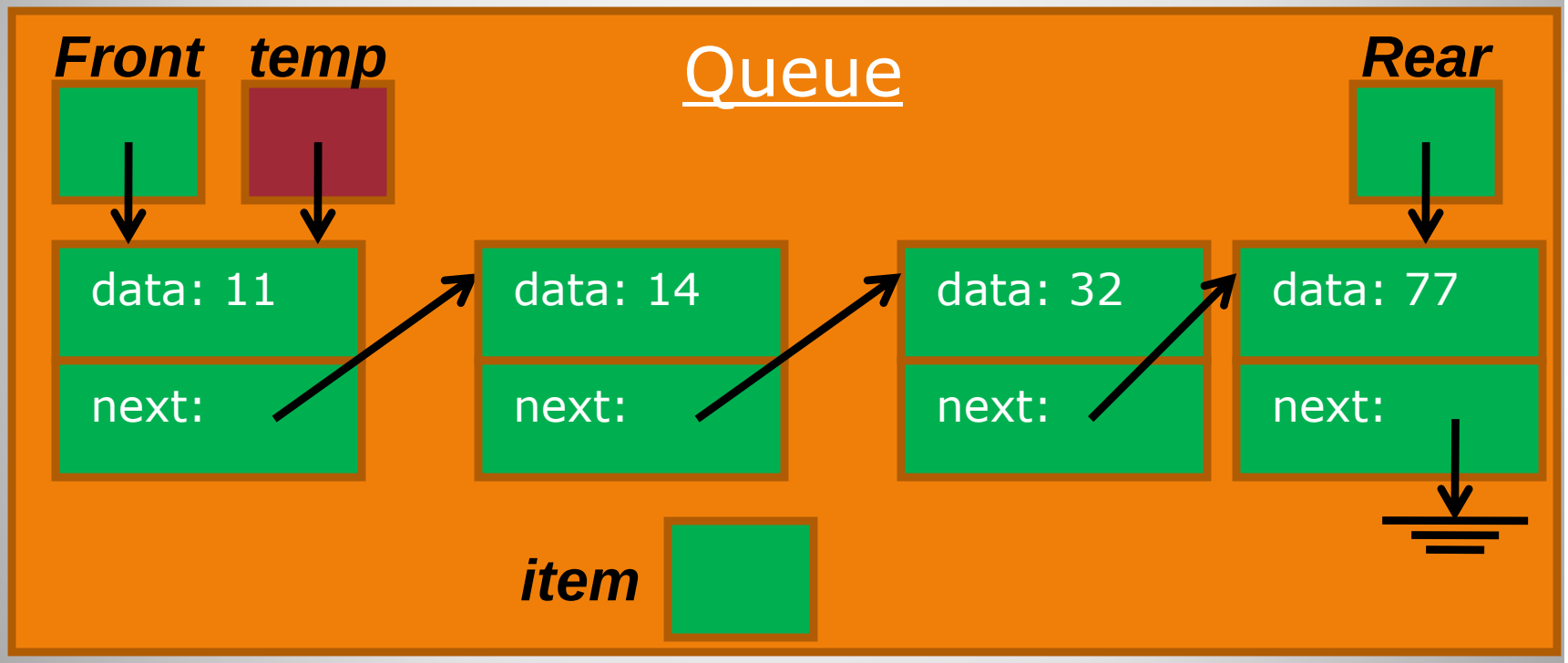
4. Make the new element the rear

Dequeue Pseudocode

1. Get a temp pointer to the front node.
2. Set the data on the item that will be sent back.
3. Set the front pointer to the second node in the queue.
4. If the queue is now empty set the rear pointer to nullptr.
5. Release memory for the original first node in the queue.

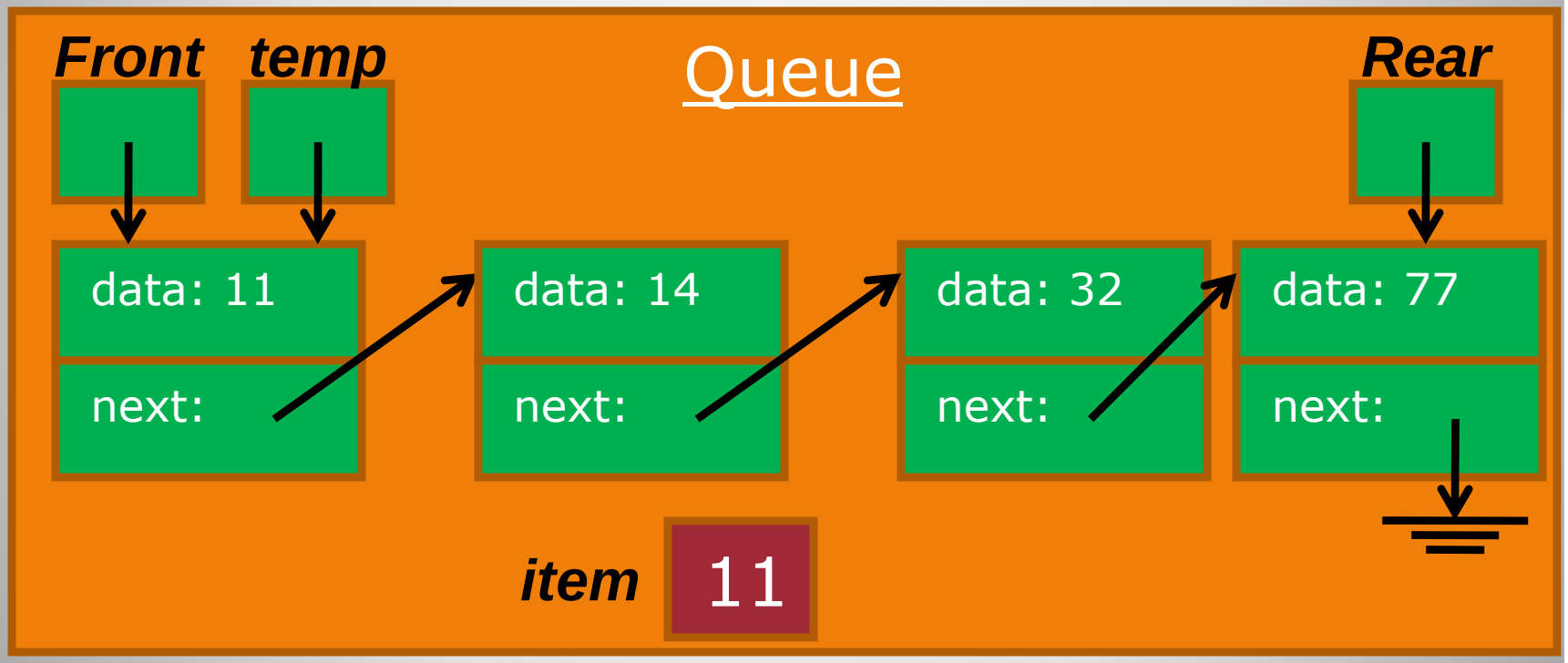
Queue (Linked) - Dequeue

1. Get a temp pointer to the front node.



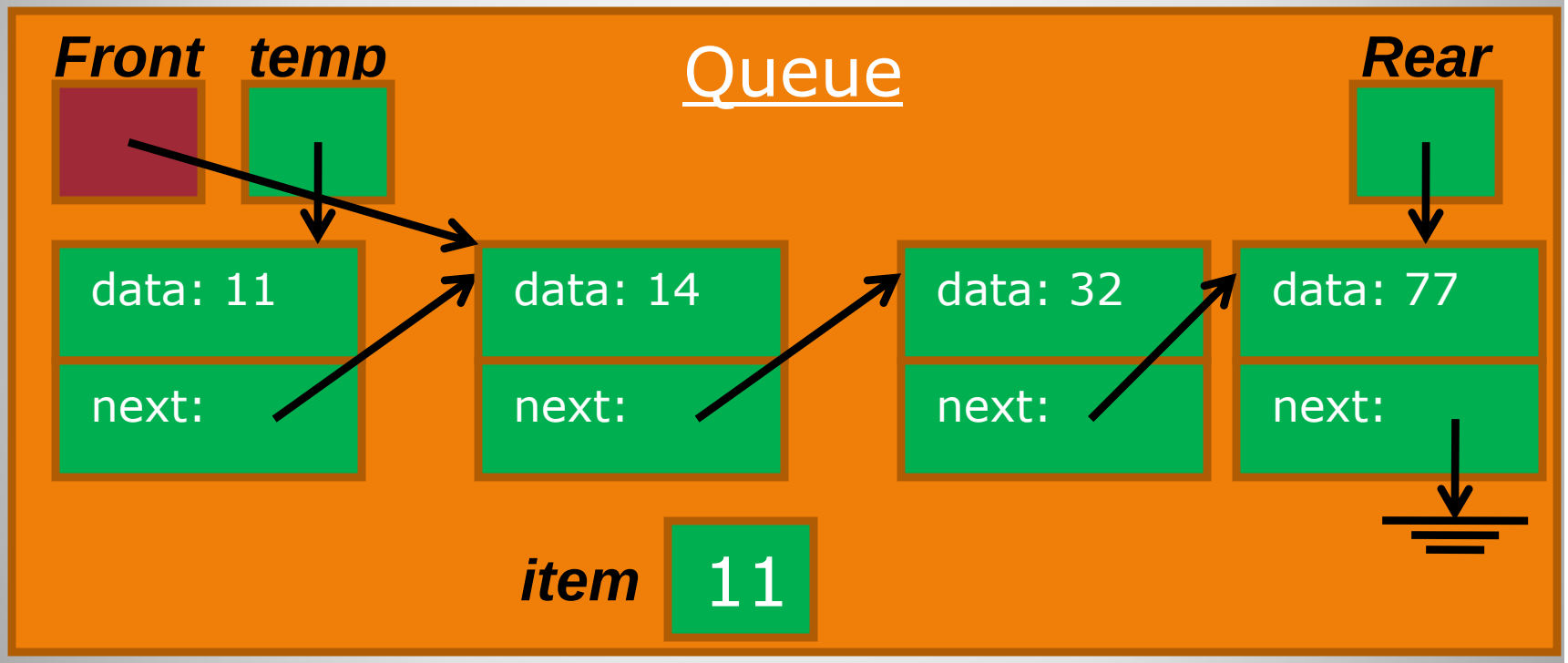
Queue (Linked)

2. Set the data on the item that will be sent back.



Queue (Linked)

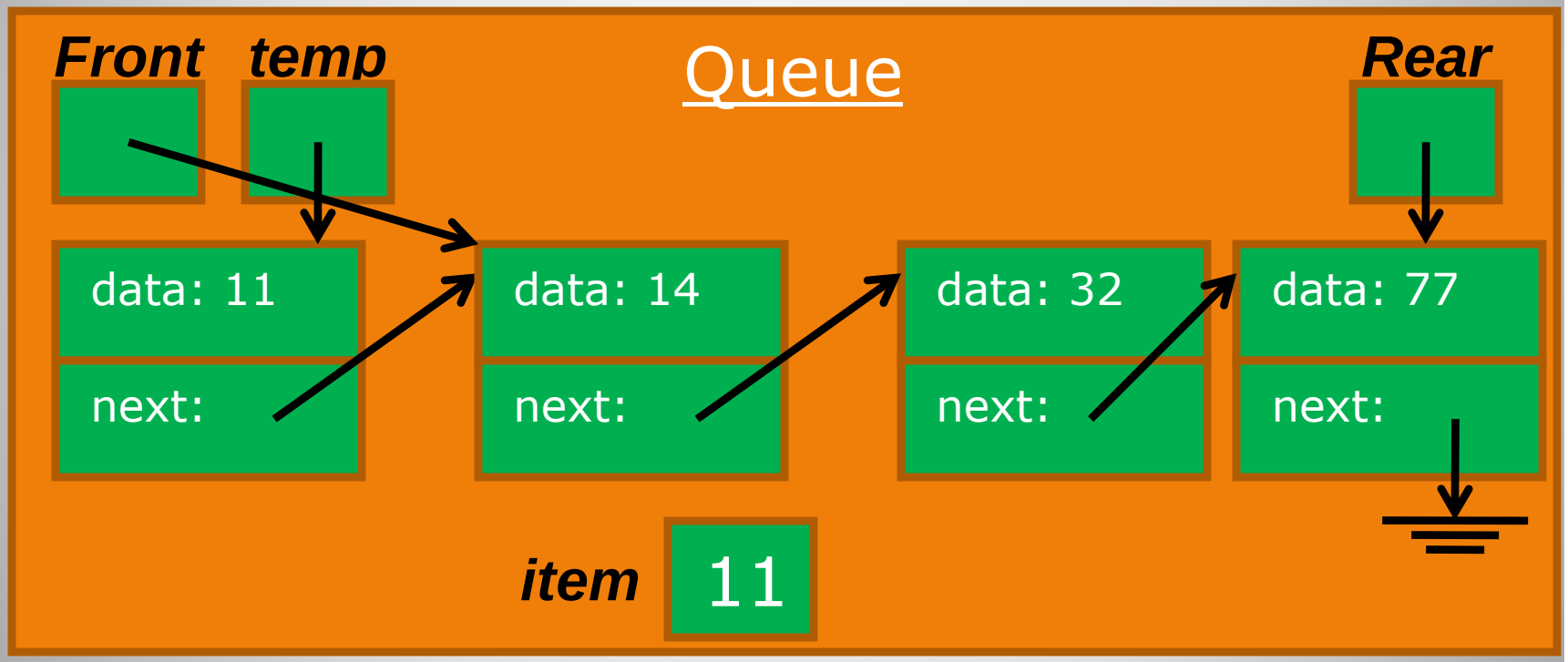
3. Set the front pointer to the second node in the queue.



Queue (Linked)

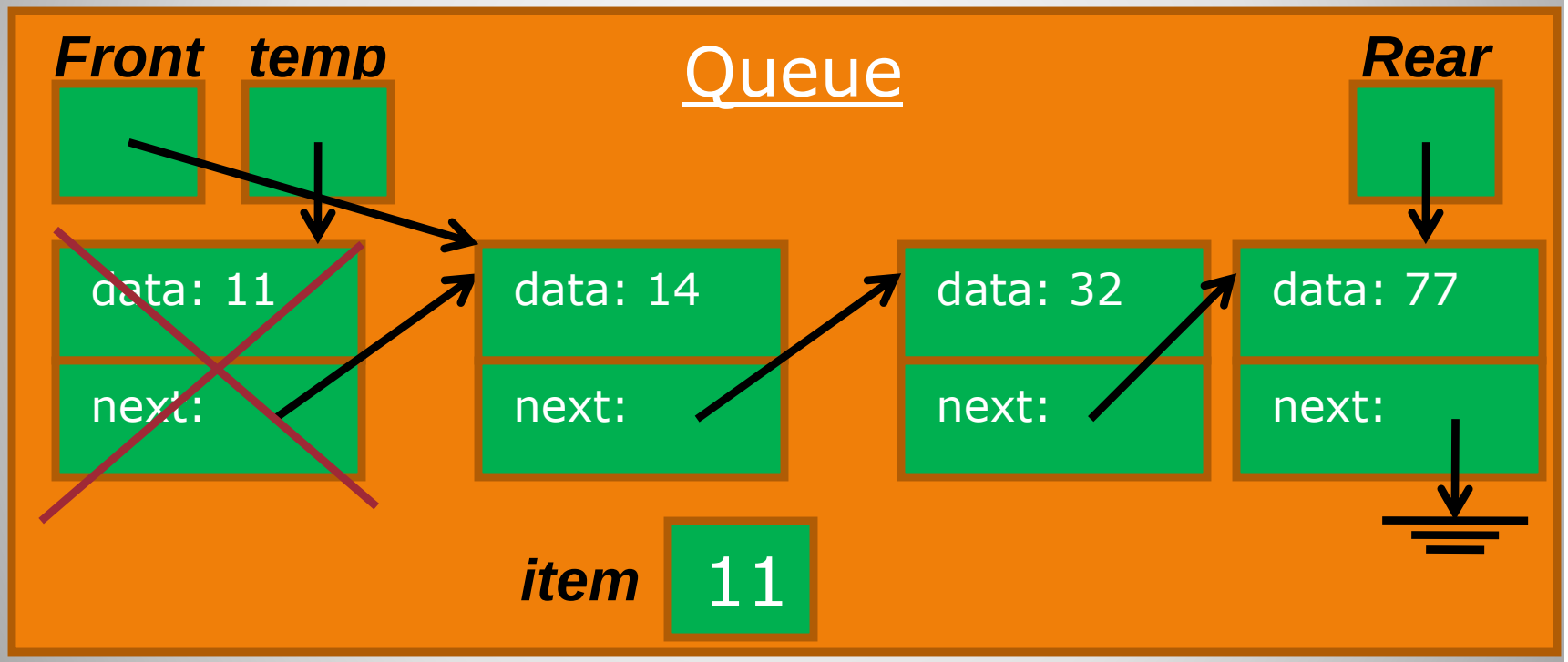
4. If the queue is now empty set the rear pointer to nullptr.

THE QUEUE IS NOT EMPTY SO NOTHING CHANGES ON THIS STEP.



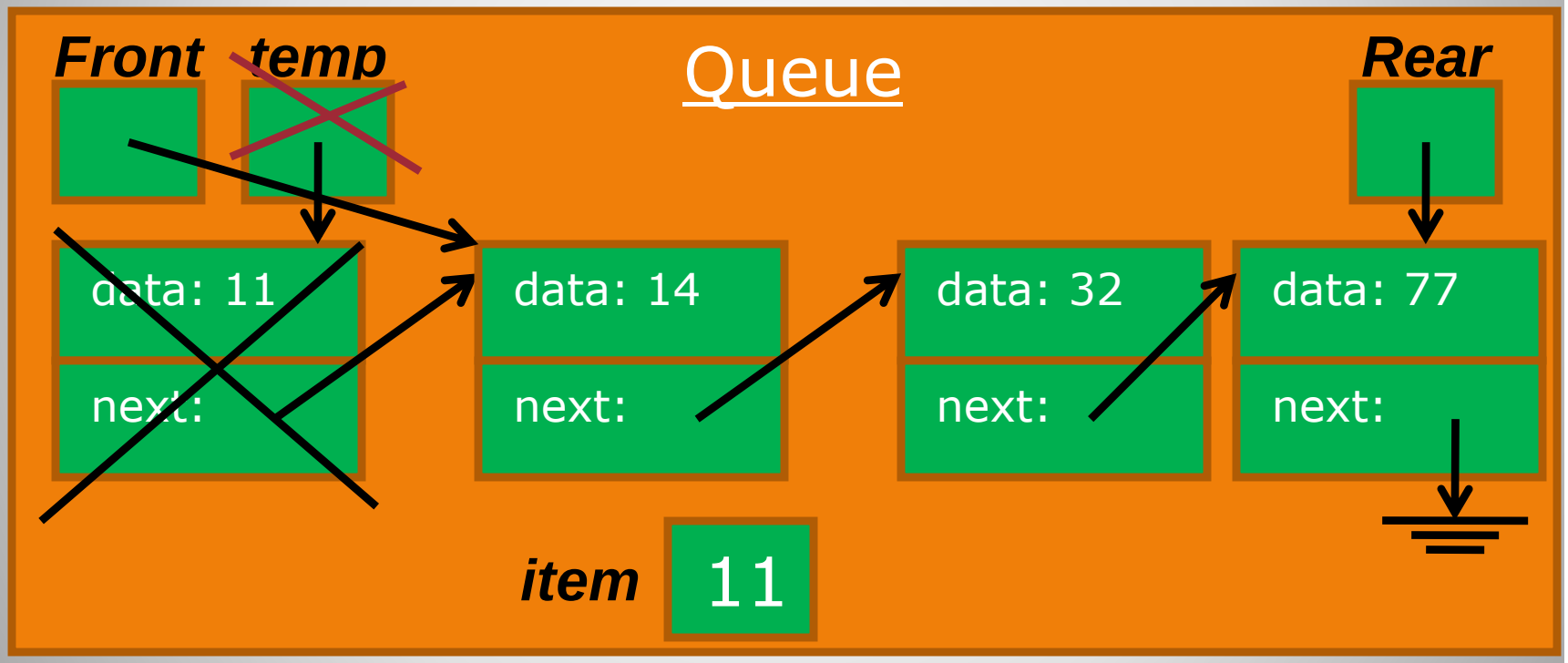
Queue (Linked)

5. Release memory for the original first node in the queue.



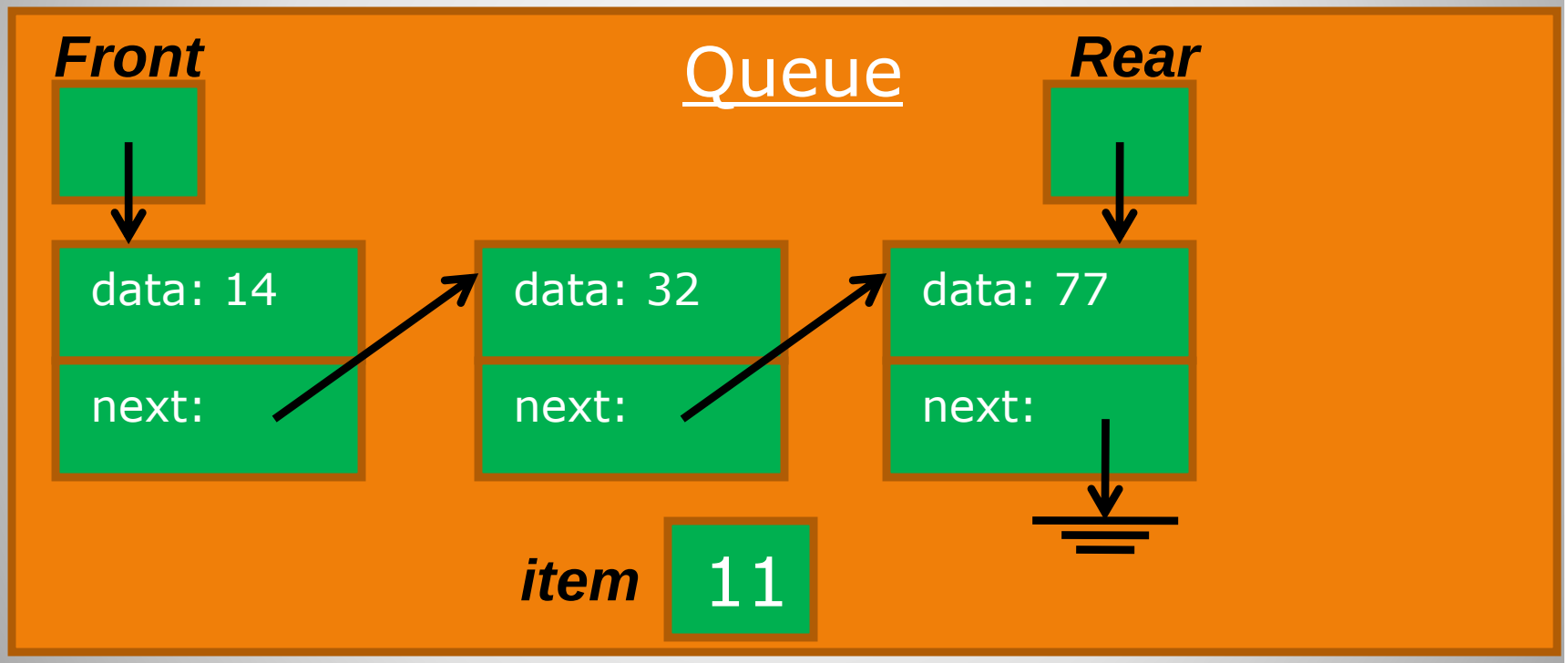
Queue (Linked)

The temp pointer will disappear when the Dequeue method ends.



Queue (Linked)

This picture is ***LOGICALLY EQUIVALENT*** to the previous slide!!!



Queue (Linked)

```
dequeue() returns int throws Exception
        Declare int item

        if (queue is empty)
            throw exception "Empty Stack"
        else
            Declare Node temp
            Set temp to front

            Set item to front.data
            Set front to front.next

            if (front equals null)
                Set rear to null

            Set temp to null

        return item
```

1. Get a temp pointer to front

2. Get the data to return

3. Move front to the second element

4. If front is null the the queue is now empty to set rear to null

5. Release memory for the old front node (will become a candidate for garbage collection)

Queue (Linked)

makeEmpty()
    Set front to null
    Set rear to null

Deletes <u>ALL</u> of the elements in the queue.

Make front and rear null.

All nodes in the queue are now unreferenced so they will become candidates for garbage collection

```
isFull() returns boolean
    Declare Node location
    try
        Set location to new Node instance
        Set location to null
        return false
    catch OutOfMemoryError exception
        return true
```

Same as Linked version of Unsorted List

Check to see if you can allocate memory.

If you can then queue is not full so return false.

If you cannot allocate memory, then queue is full.

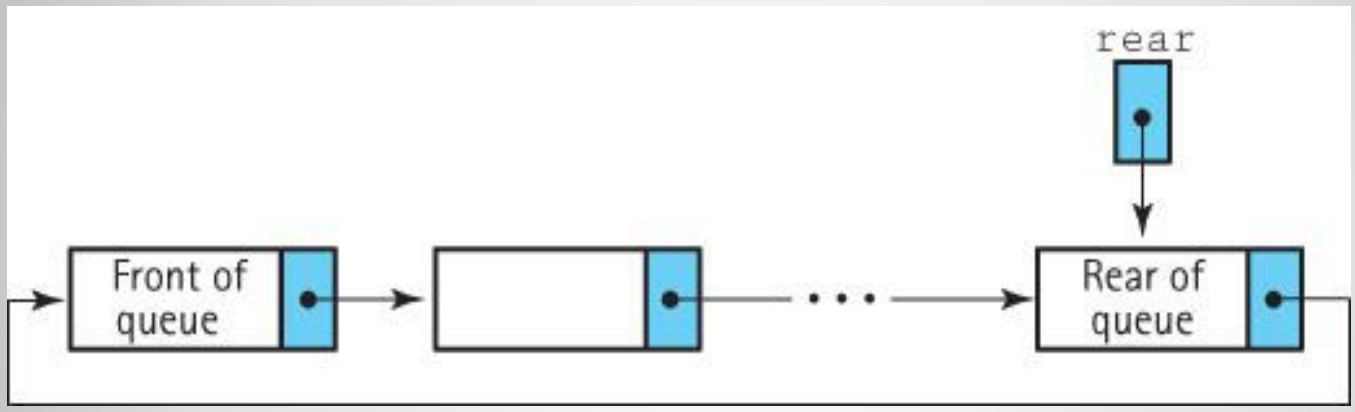
Queue (Linked)

Circular Queue

- Last element points to first element.
- Only stores one pointer – rear
- For example…

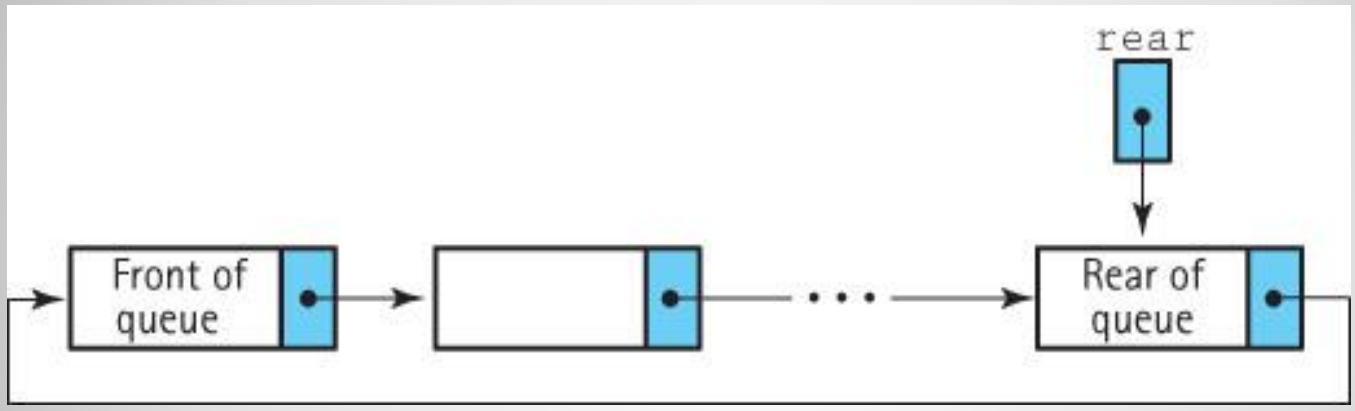
Circular Queue

A circular linked queue uses only one external pointer: rear



How do you access front?

A circular linked queue uses only one external pointer: rear



How do you access front?
Answer: Front is the next element after rear.

Circular Queue

- What are the Big-O runtimes for the array-based and linked implementations of a queue?

Big-O Comparison – Queue

Operation	Cost
makeEmpty	O(1)
isFull	O(1)
isEmpty	O(1)
enqueue	O(1)
dequeue	O(1)
Constructor	O(1)

Big-O Comparison – Queue(Linked)

- **End of Slides**

End of Slides