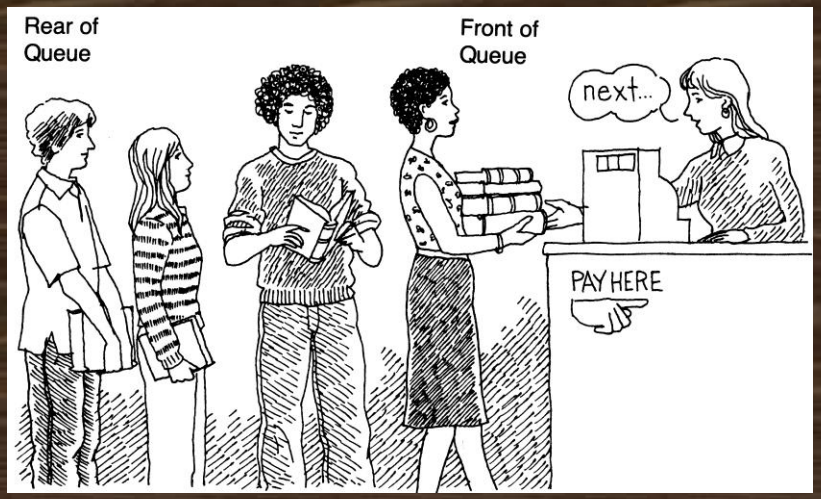
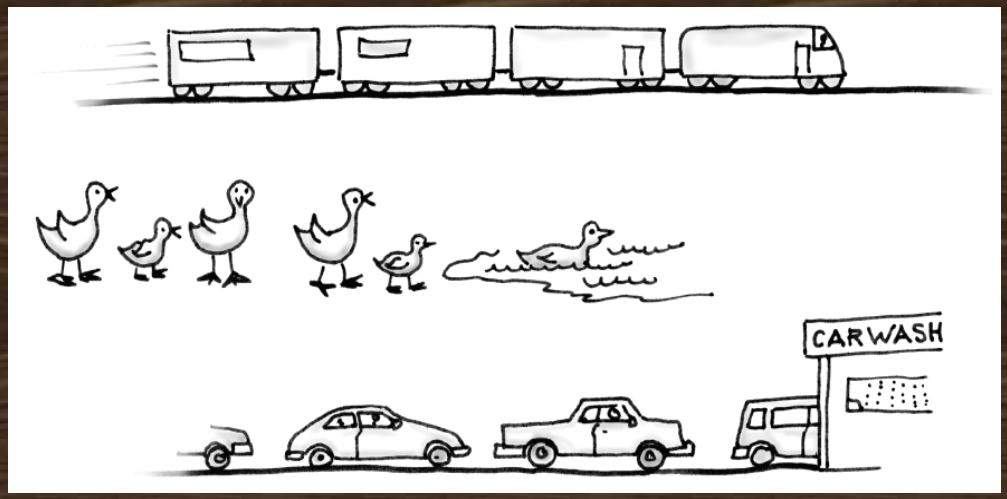
Queues

What is a queue?

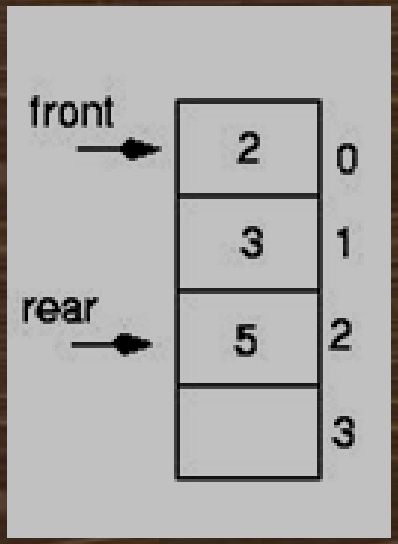
- It is an ordered group of homogeneous items.
- Queues have two ends:
  - Items are added at one end.
  - Items are removed from the other end.
- **FIFO property**: First In, First Out
  - The item added first is also removed first



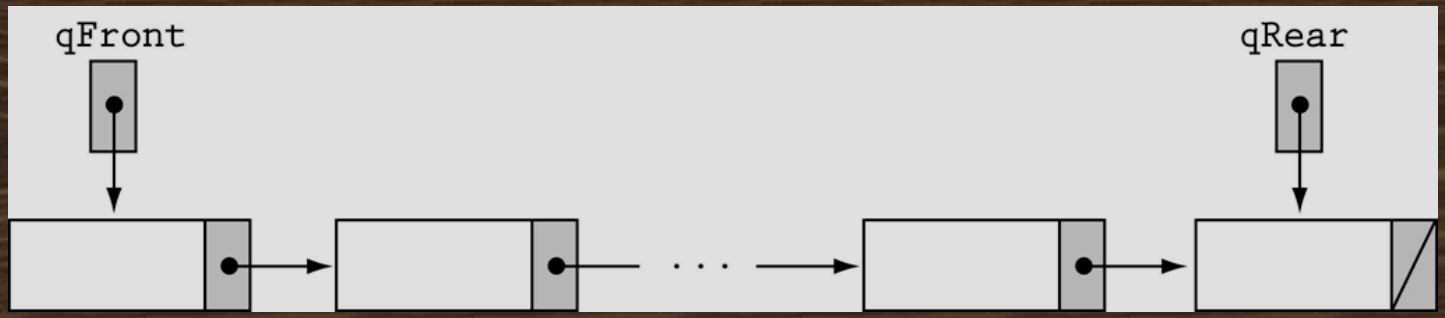


Queue Implementations

Array-based



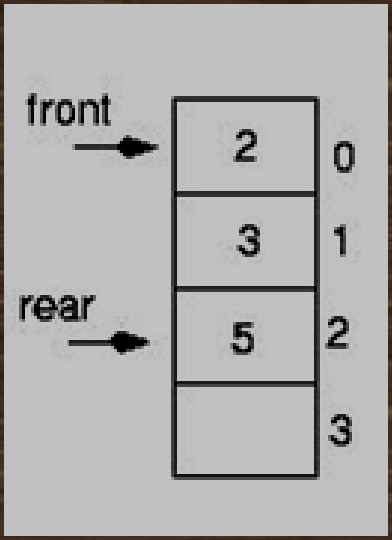
Linked-list-based



Array-based Implementation

```
template<class ItemType>
class QueueType {
 public:
    QueueType(int);
    ~QueueType();
    void MakeEmpty();
    bool IsEmpty() const;
    bool IsFull() const;
    void Enqueue(ItemType);
    void Dequeue(ItemType&);
```

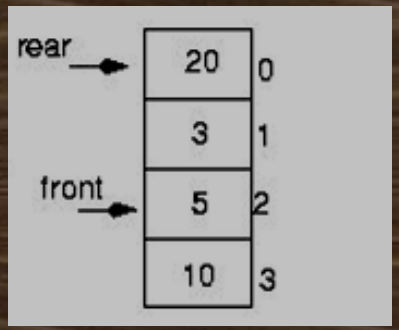
```
 private:
    int front, rear;
    ItemType* items;
    int maxQue;
};
```



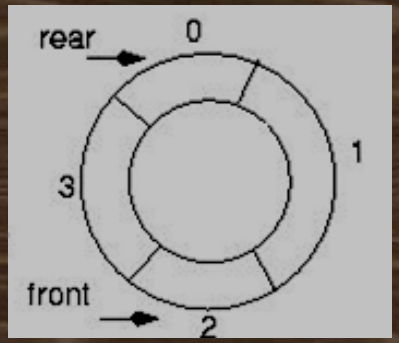
Implementation Issues

- *Optimize* memory usage.
- Conditions for a *full* or *empty* queue.
- Initialize *front* and *rear*.

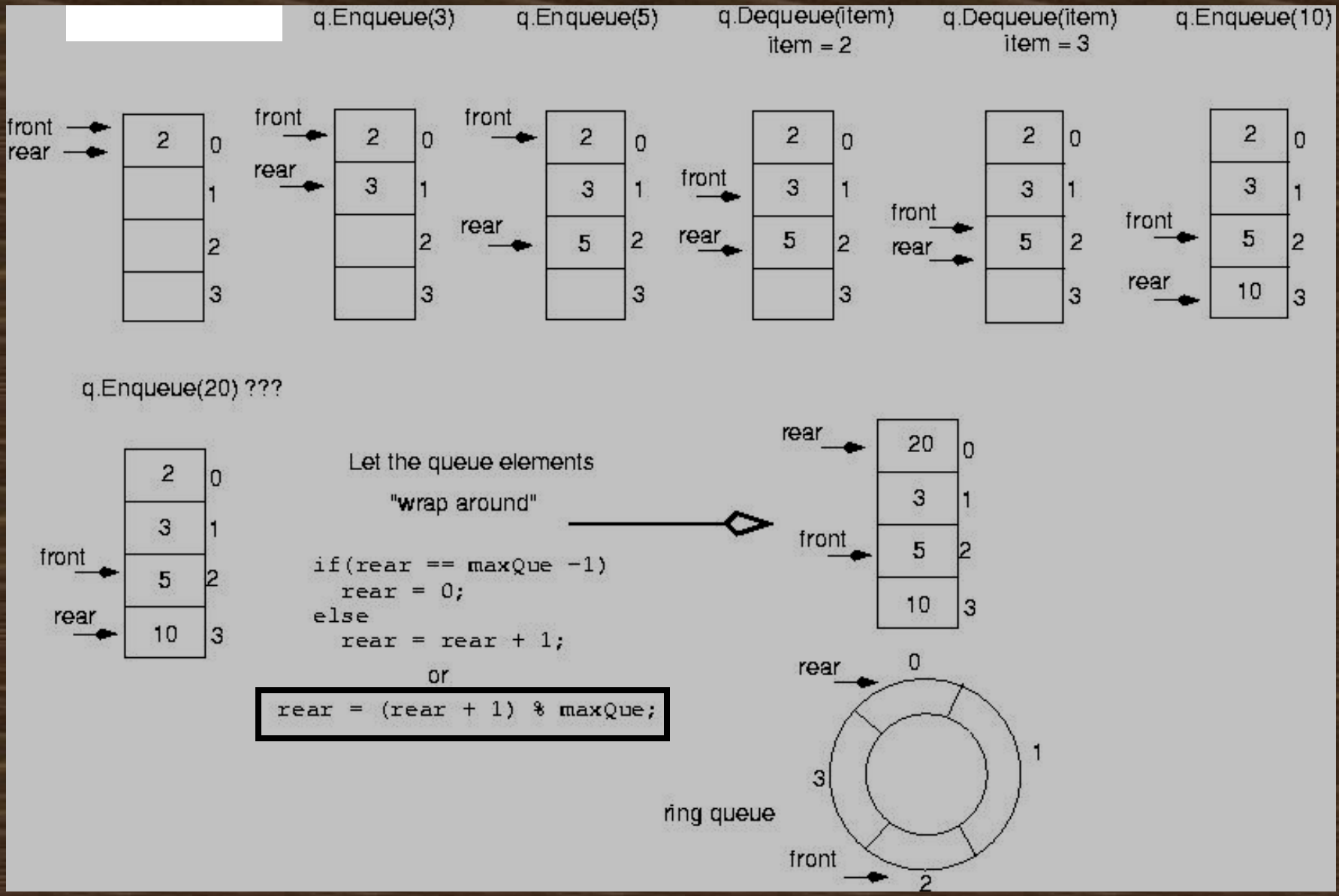
linear array



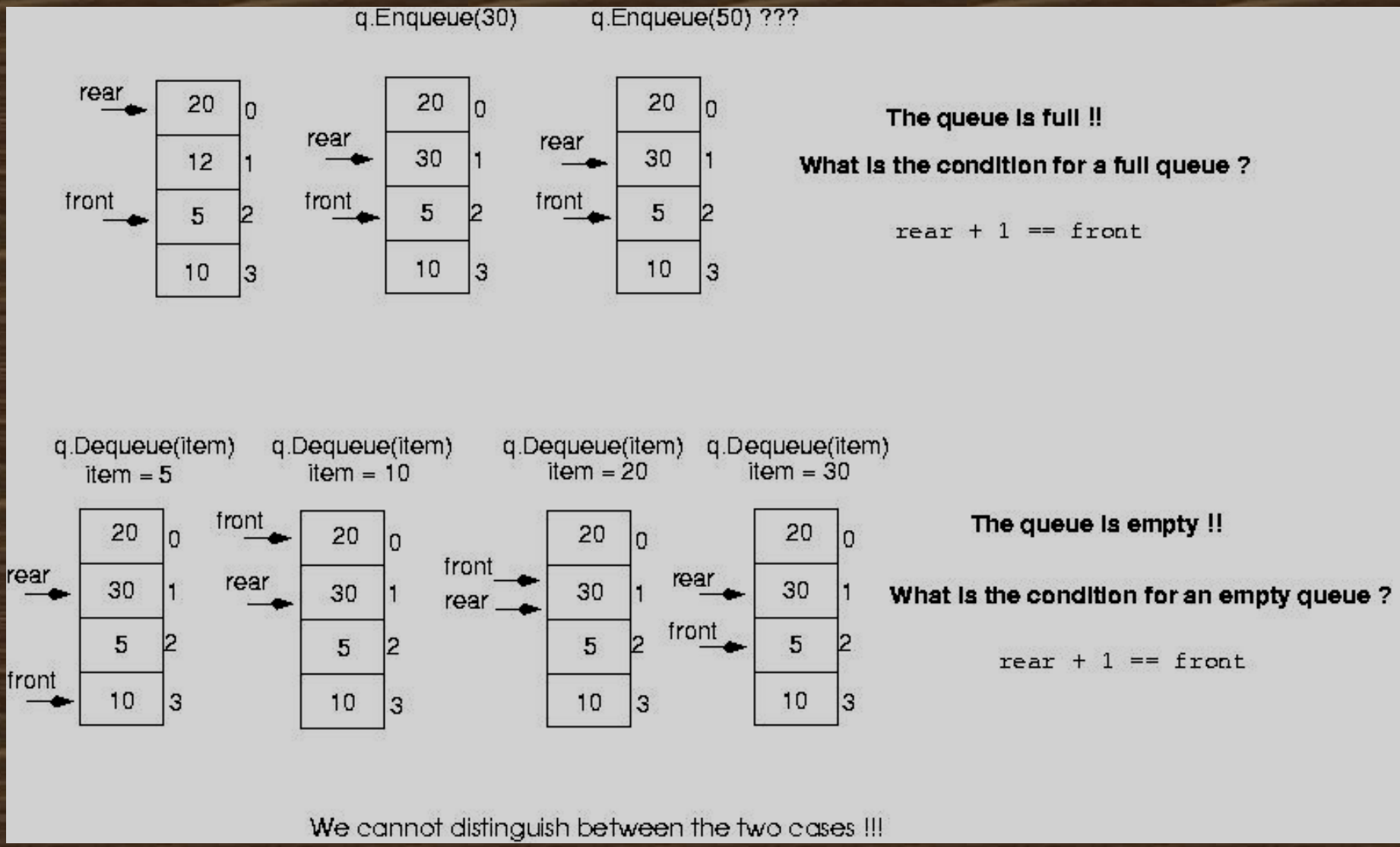
circular array



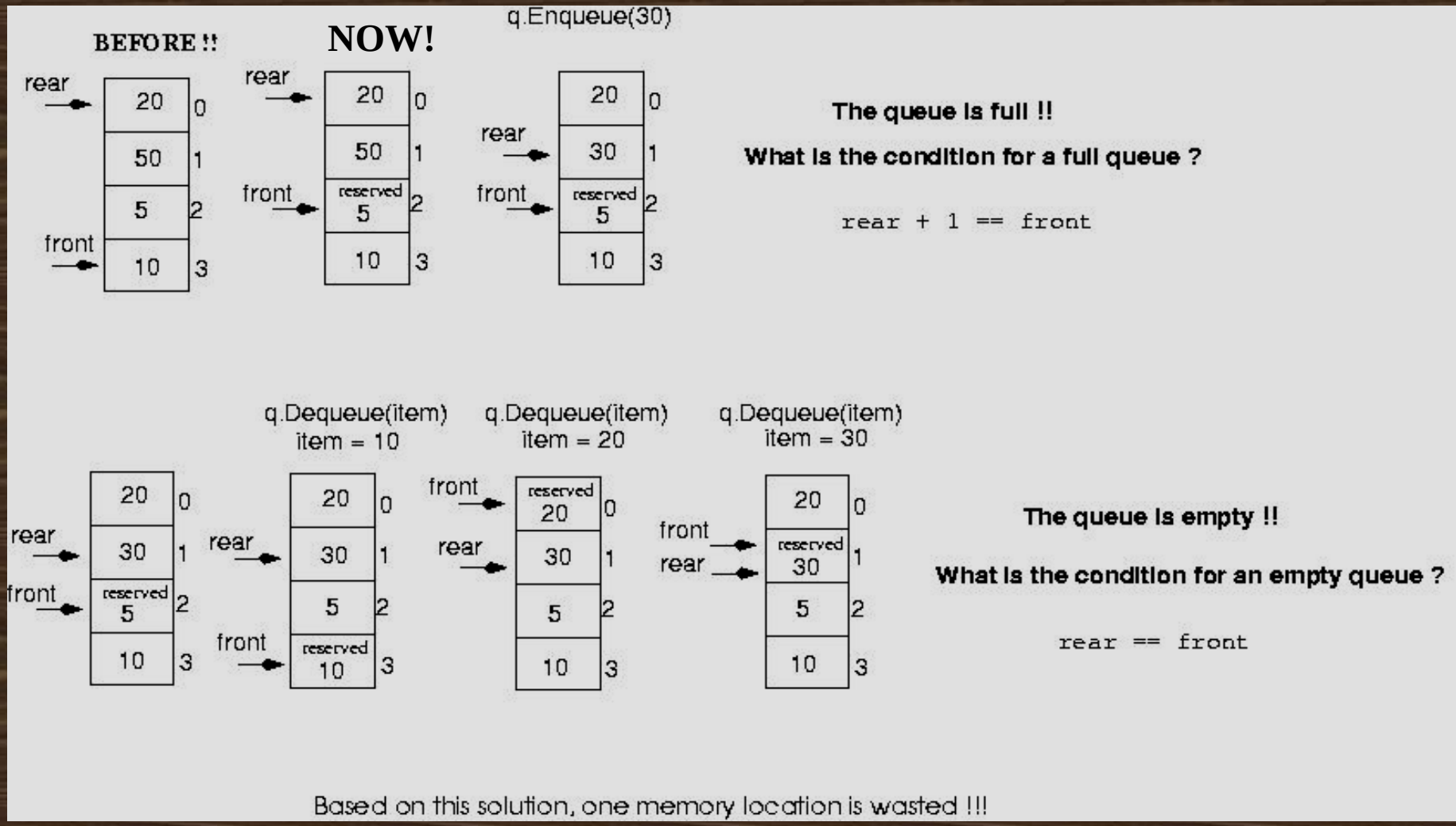
Optimize memory usage



Full/Empty queue conditions



"Make *front* point to the element **preceding** the front element in the queue!"



Operation on Queue

1.Enqueue (ItemType newItem)

- *Function*: Adds newItem to the rear of the queue.
- *Preconditions*: Queue has been initialized and is not full.
- *Postconditions*: newItem is at rear of queue.

2. Queue overflow

- The condition resulting from trying to add an element onto a full queue.

```
if(!q.IsFull())
  q.Enqueue(item);
```

Array-based Implementation (cont.)

```
template<class ItemType>
void QueueType<ItemType>::Enqueue
   (ItemType newItem)
{
 rear = (rear + 1) % maxQue;
 items[rear] = newItem;
}
```

O(1)

Dequeue (ItemType& item)

- *Function*: Removes front item from queue and returns it in item.
- *Preconditions*: Queue has been initialized and is not empty.
- *Postconditions*: Front element has been removed from queue and item is a copy of removed element.

Queue underflow

- The condition resulting from trying to remove an element from an empty queue.

```
if(!q.IsEmpty())
    q.Dequeue(item);
```

Array-based Queue Implementation (cont.)

```
template<class ItemType>
void QueueType<ItemType>::Dequeue
   (ItemType& item)
{
 front = (front + 1) % maxQue;
 item = items[front];
}
```

O(1)

Linked-list-based Implementation

- Allocate memory for each new element dynamically
- Link the queue elements together
- Use two pointers, *qFront* and *qRear*, to mark the front and rear of the queue