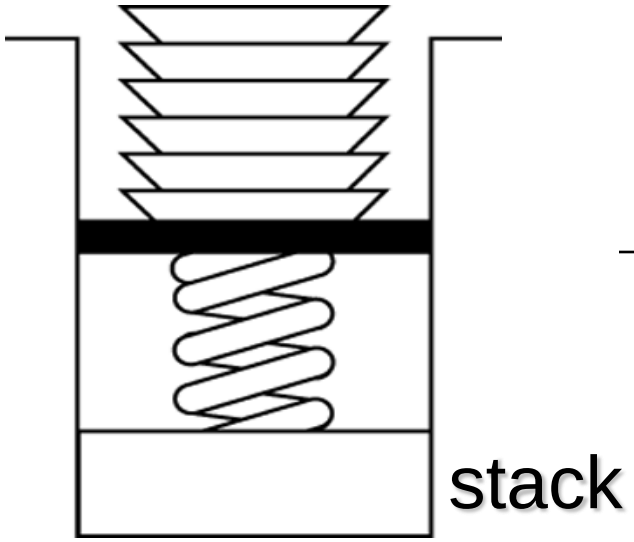
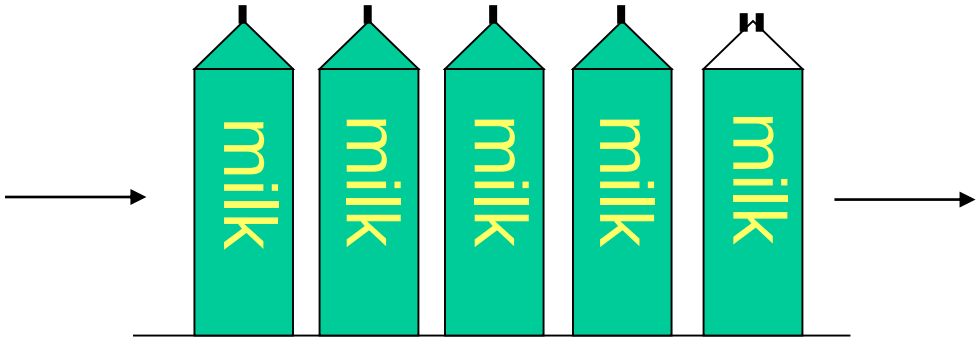
Ch. 7 Queue

- Stack
  - Removes the most recently added item
- Queue
  - Removes the least recently added item



stack



queue

Queue Examples

- 오래된 우유부터 먹기
- 식당에서 기다리는 사람들의 열
- Lotto 복권을 사려는 사람들의 열

✓ Stack: LIFO (Last-In-First-Out)

✓ Queue: FIFO (First-In-First-Out)

ADT *Queue* Operations

- Create an empty queue
- Determine whether a queue is empty
- Add a new item to the queue
- Remove from the queue the earliest added item
- Retrieve from the queue the earliest added item
- Remove all the items from the queue

✓ 본 chapter는 stack과 비교해가면서 공부할 것!

Queue Interface

```
public interface QueueInterface {

        public boolean isEmpty( );
        public void dequeueAll ( );
        public void enqueue(Object newItem);
        public Object dequeue( );
        public Object peek( );
}
```

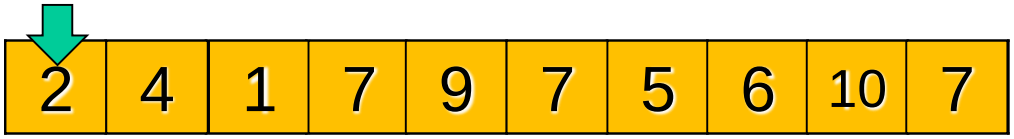
비교: stack

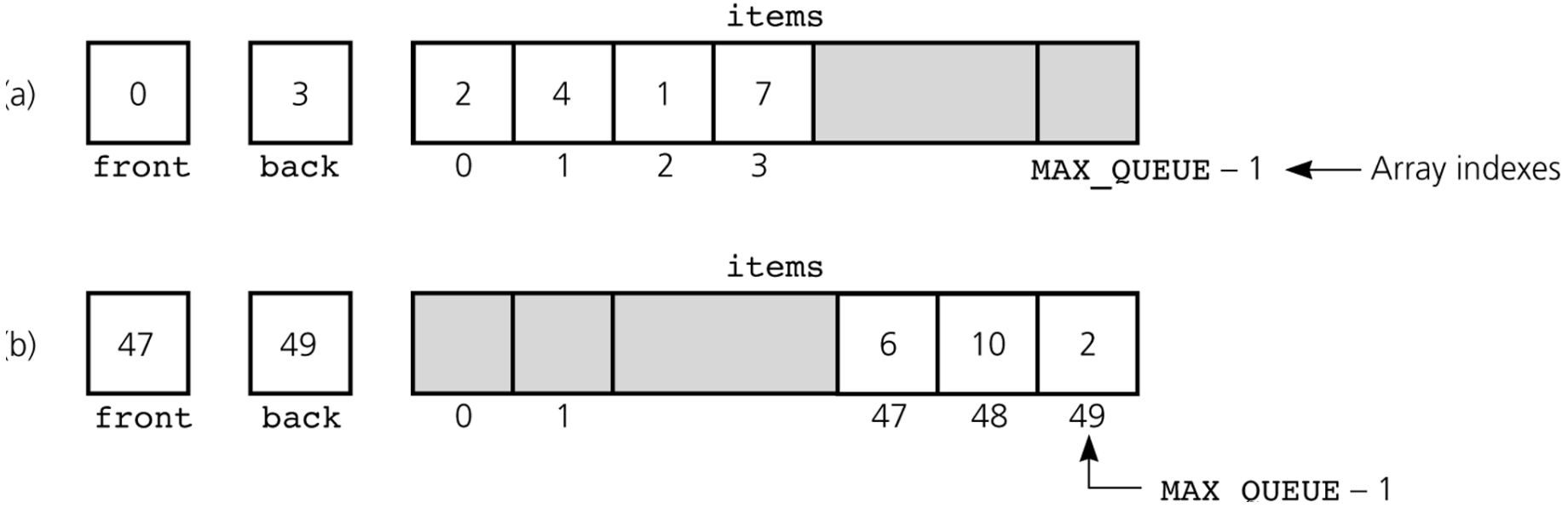
— isEmpty( );
— popAll ( );
— push(…);
— pop( );
— peek( );

- Notable fact
  - The only accessible item in the queue is the earliest added item

Example Queue Operations

Operation	Queue after operation
	Front
`queue.createQueue()`	
`queue.enqueue(5)`	5
`queue.enqueue(2)`	5 2
`queue.enqueue(7)`	5 2 7
`queueFront = queue.peek()`	5 2 7 (`queueFront` is 5)
`queueFront = queue.dequeue()`	2 7 (`queueFront` is 5)
`queueFront = queue.dequeue()`	7 (`queueFront` is 2)

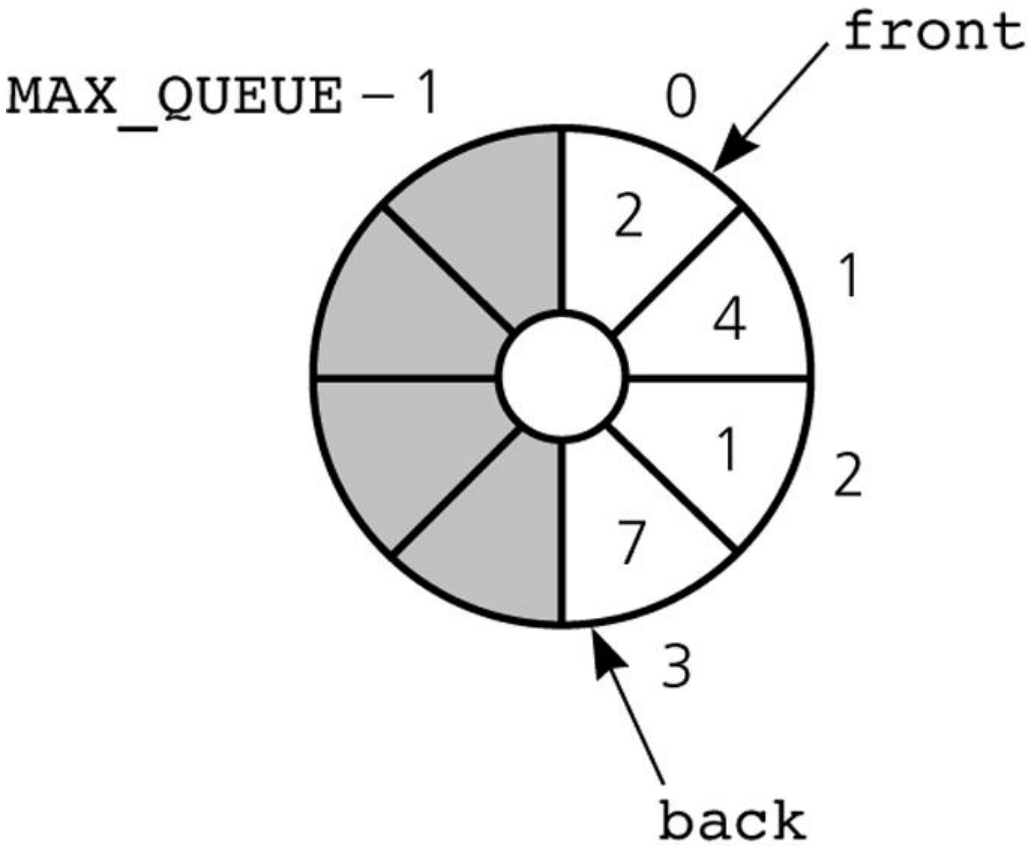




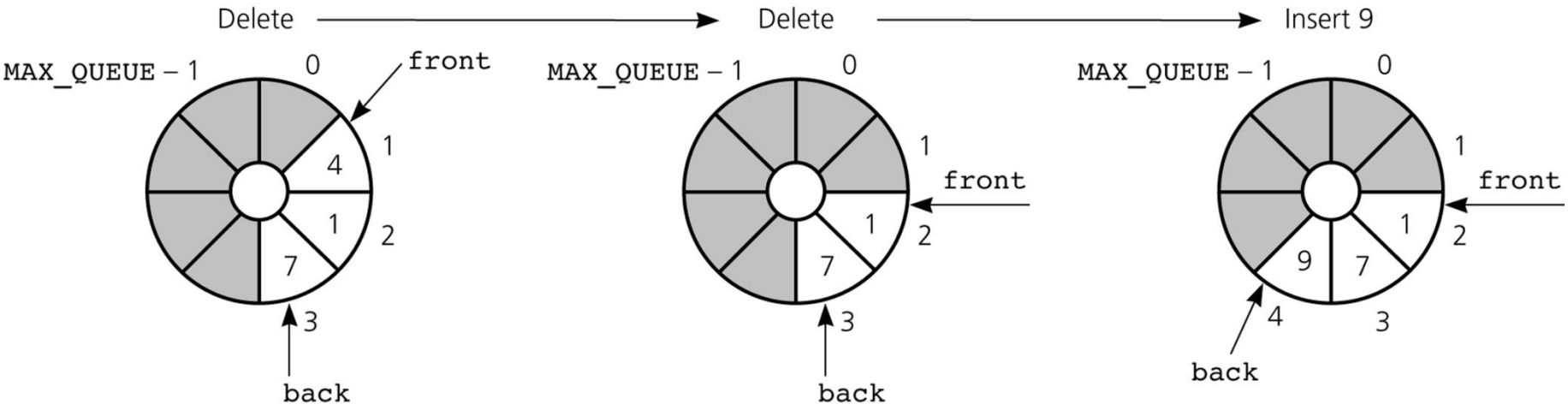
(a) A naive array-based implementation of a queue

(b) Rightward drift can cause the queue to appear full

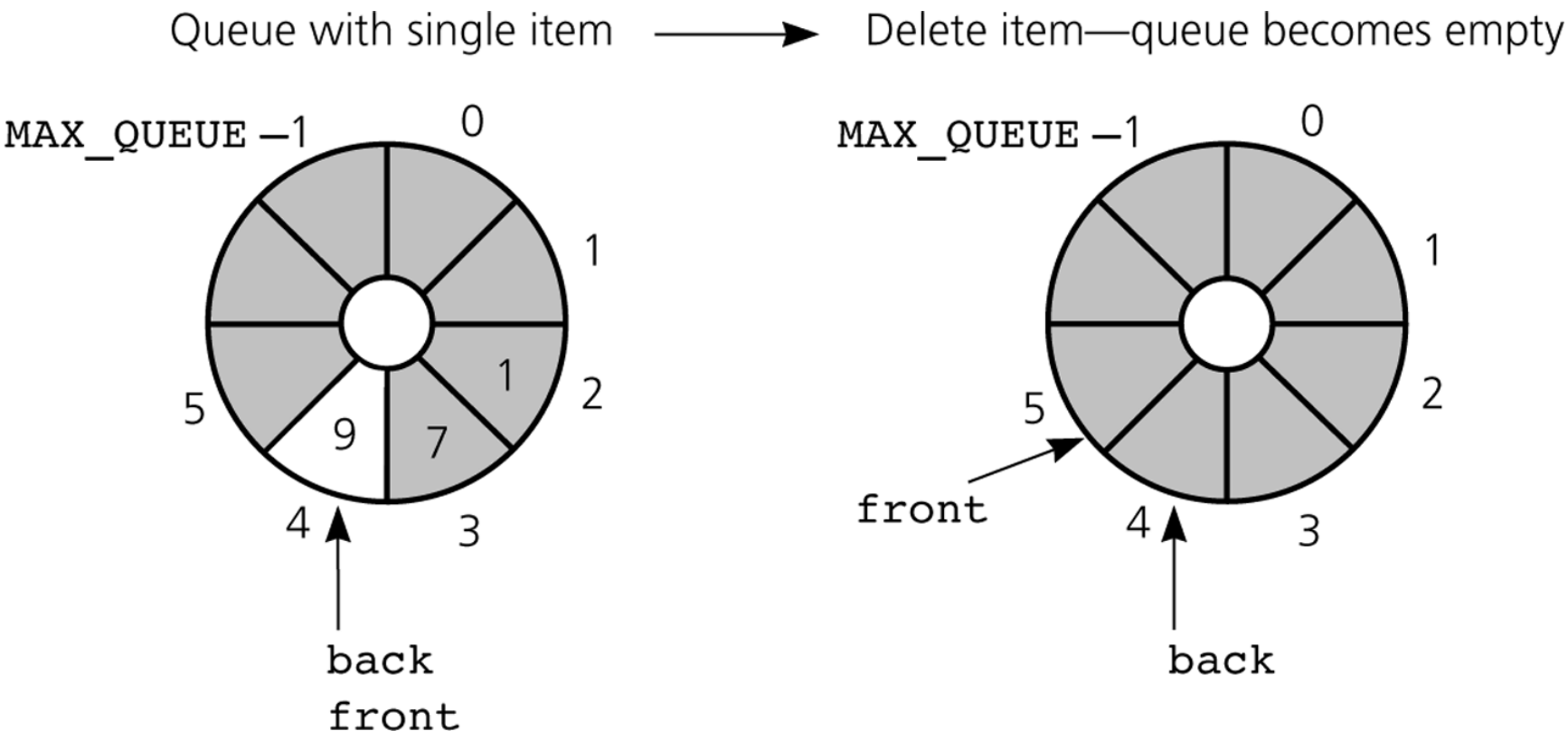
A circular implementation of a queue



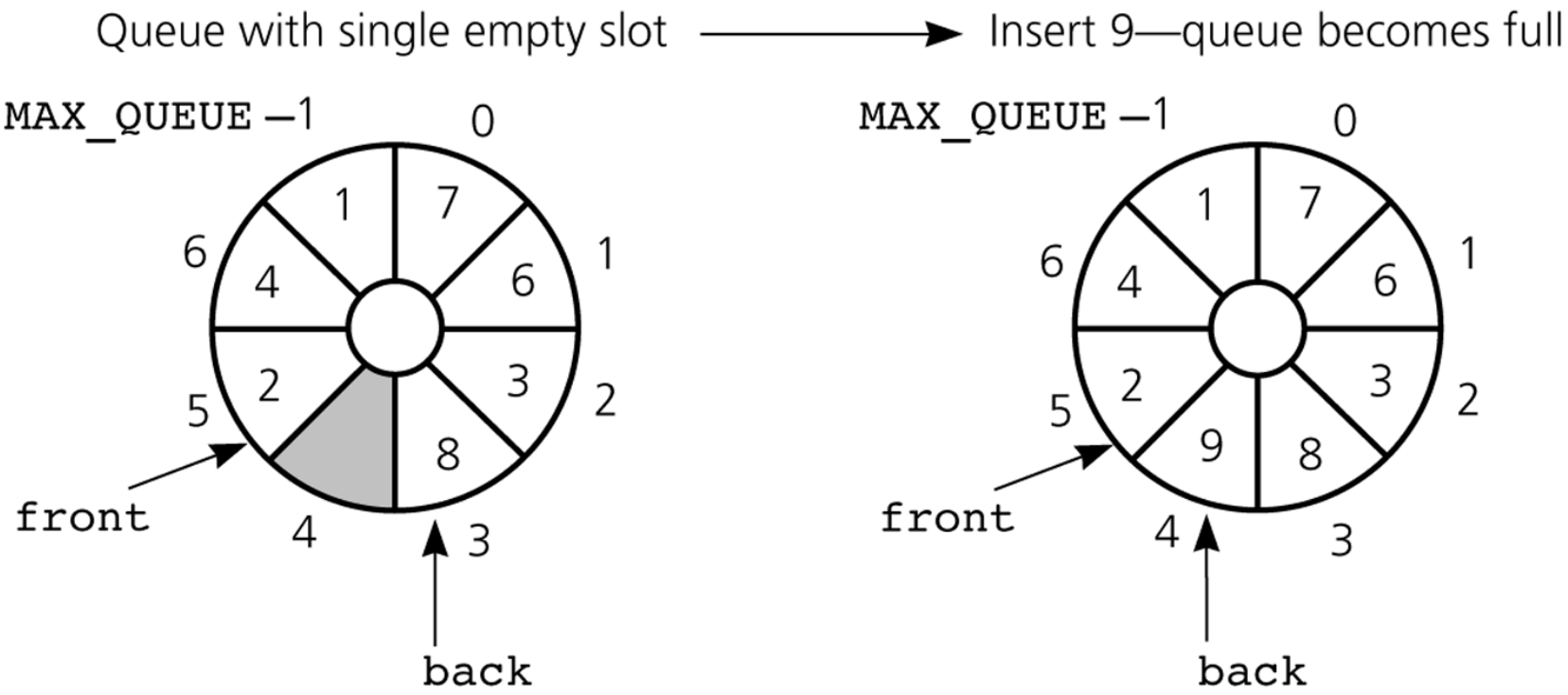
The effect of some operations of the queue



front passes ***back*** when the queue becomes empty



back catches up to ***front*** when the queue becomes full



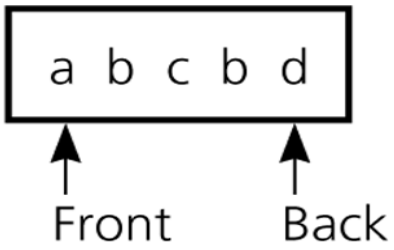
Example:Recognizing Palindromes

Palindromes =

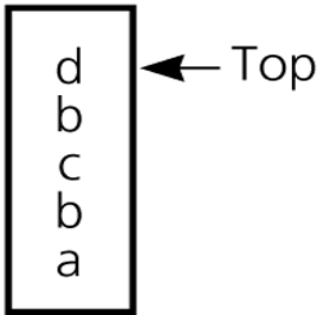
{***w*** | ***w*** reads the same
left to right
as right to left}

String: abcbd

Queue:



Stack:



```
boolean isPal(w)
{
        for i = 1 to w.length( ) {
                queue.enqueue(ith character of w);
                stack.push(ith character of w);
        }
        // start to compare
        while (!queue.isEmpty( )) {
                if (queue.dequeue( ) != stack.pop( )) return false;
        }
        // finished w/ empty queue (and empty stack)
        return true;
}
```

Queue Implementation

```
public interface QueueInterface {

        public boolean isEmpty( );
        public void dequeueAll ( );
        public void enqueue(Object newItem);
        public Object dequeue( );
        public Object peek( );
}
```

세가지 구현

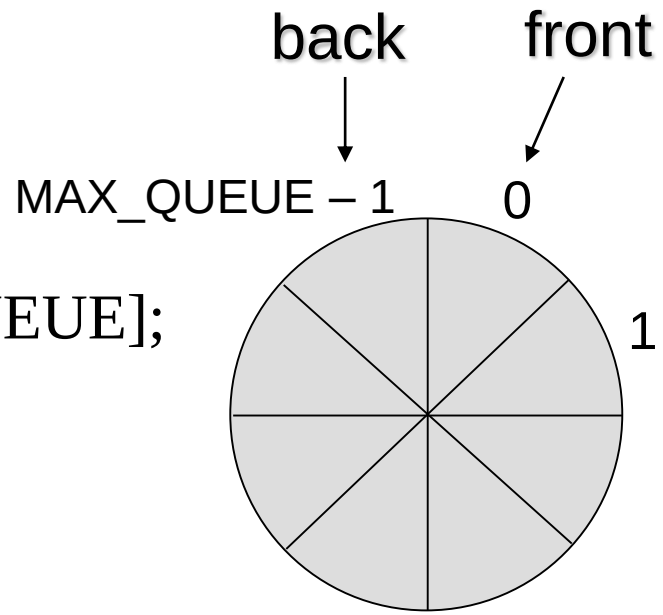
- Array Based
- Reference Based
- Reusing List

✓ Interface는 전혀 변하지 않는다

```
public interface QueueInterface {
        public boolean isEmpty( );
        public void dequeueAll ( );
        public void enqueue(Object newItem);
        public Object dequeue( );
        public Object peek( );
}
```

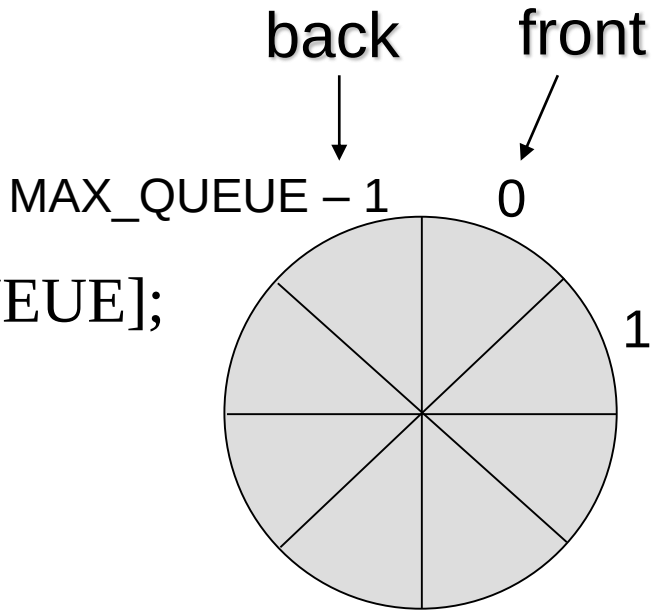
Array-Based Implementation (Circular)

```
public class QueueArrayBased implements QueueInterface {

    final int MAX_QUEUE = 50;
    private Object items[ ];
    private int front, back, numItems;

    public QueueArrayBased( ) {
        items = new Object[MAX_QUEUE];
        front = 0;
        back = MAX_QUEUE – 1;
        numItems = 0;
    }
```



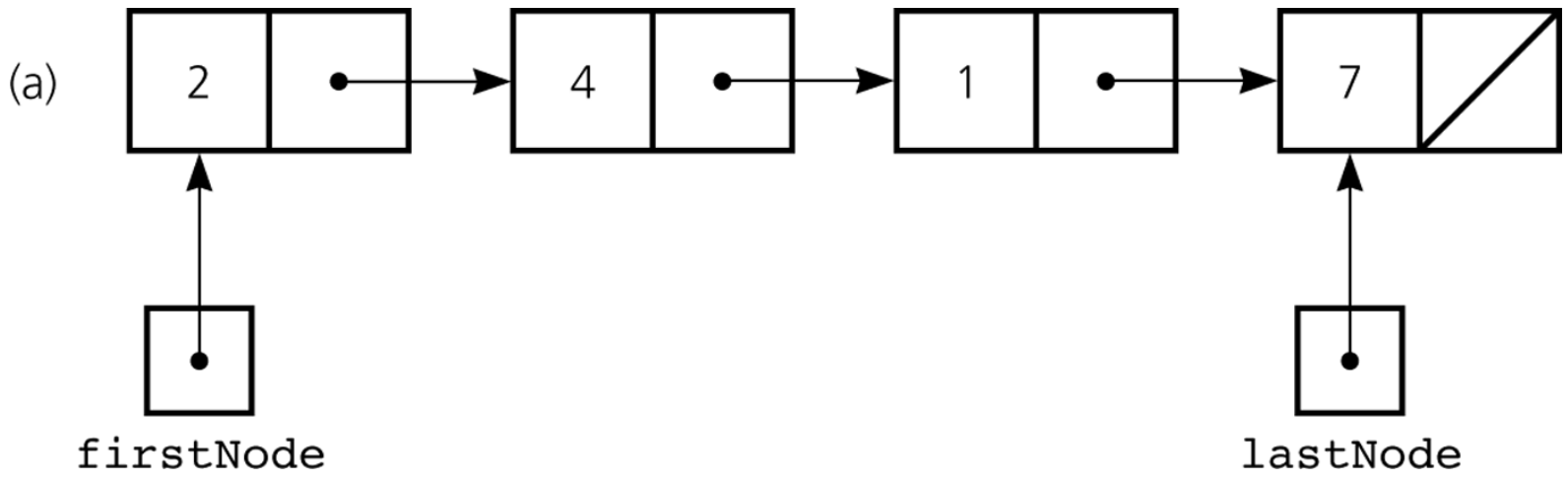
```
public boolean isEmpty( ) {
        return (numItems == 0);
}
public boolean isFull ( ) {
        return (numItems ==  MAX_QUEUE);
}
public void enqueue(Object newItem) {
        if (!isFull( )) {
                back = (back+1) % MAX_QUEUE;
                items[back] = newItem;
                ++numItems;
        } else {exception 처리}
}
```

```
public Object dequeue( ) {
        if (!isEmpty( )) {
                Object queueFront = items[front];
                front = (front+1) % MAX_QUEUE;
                --numItems;
                return queueFront;
        } else {exception 처리}
}

public void dequeueAll ( ) {
        items = new Object[MAX_QUEUE];
        front = 0;
        back = MAX_QUEUE – 1;
        numItems = 0;
}
```

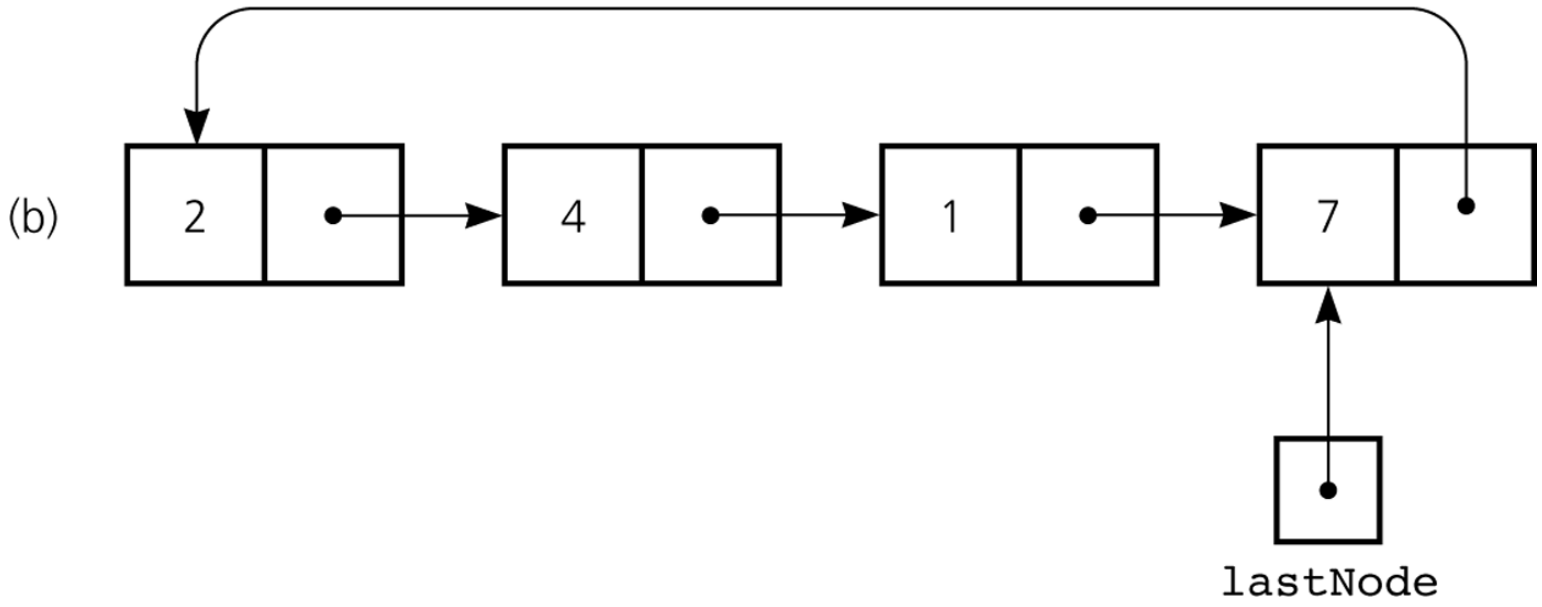


```
        public Object peek( ) {
                if (!isEmpty( )) return items[front];
                else {exception 처리}
        }
} // class QueueArrayBased
```

Reference-Based Implementation

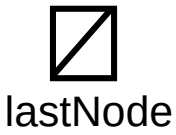


A reference-based implementation of a queue:
a linear linked list with two external references



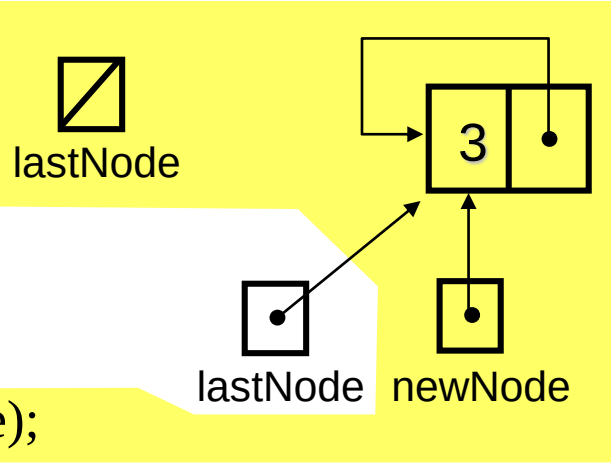
A reference-based implementation of a queue:
a circular linear linked list with one external reference

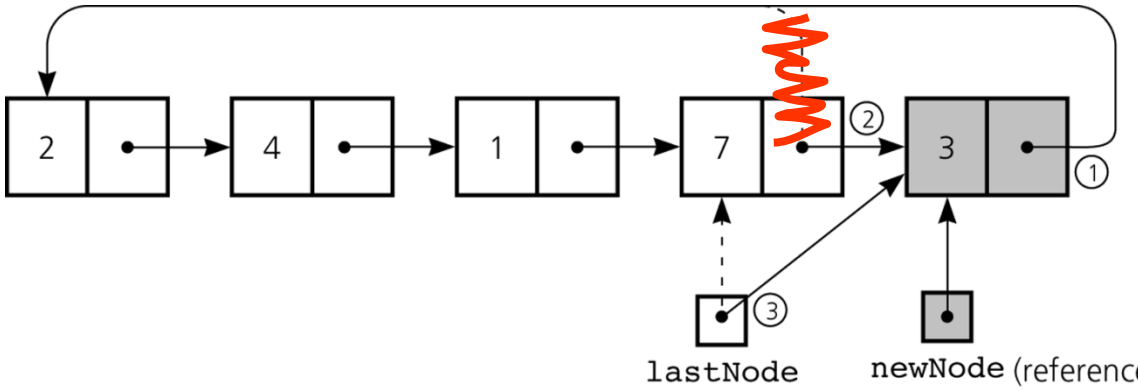
```
public class QueueReferenceBased implements QueueInterface{
        private Node lastNode;

        public QueueReferenceBased( ) {
                lastNode = null;
        }
```



✓ Class *Node*: Ch.4, 강의 노트 P.8 참조

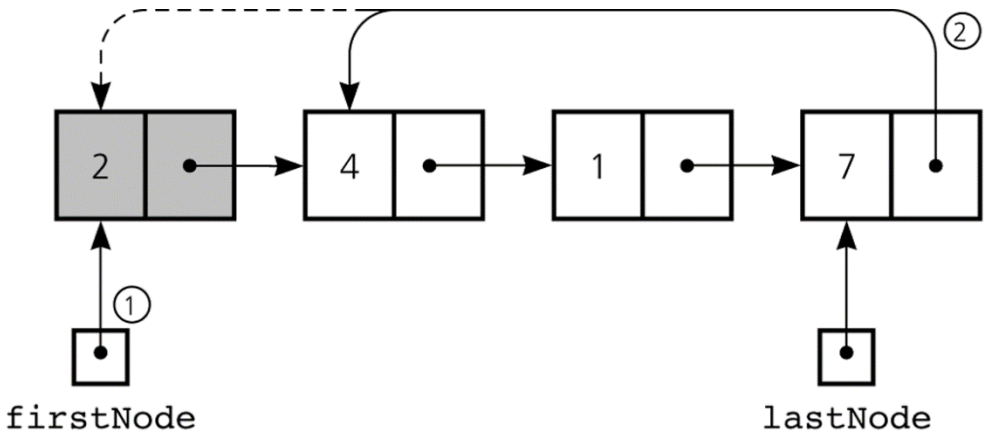
```
public boolean isEmpty( ) {
        return (lastNode == null);
}
public void enqueue(Object newItem) {
        Node newNode = new Node(newItem);
        if (isEmpty( )) newNode.setNext(newNode);
        else {
                newNode.setNext(lastNode.getNext( ));
                lastNode.setNext(newNode);
        }
        lastNode = newNode;
}
```






```
1. newNode.setNext(lastNode.getNext());
2. lastNode.setNext(newNode);
3. lastNode = newNode;
```

```
public Object dequeue( ) {
        if (!isEmpty( )) {
                Node firstNode = lastNode.getNext( );
                if (firstNode == lastNode) { // special case?
                        lastNode = null;  // only one node in queue
                } else {  // more than one item
                        lastNode.setNext(firstNode.getNext( ));
                }
                return  firstNode.getItem( );
        } else {exception 처리}
}
```




```
1. firstNode = lastNode.getNext();
2. lastNode.setNext(firstNode.getNext());
```

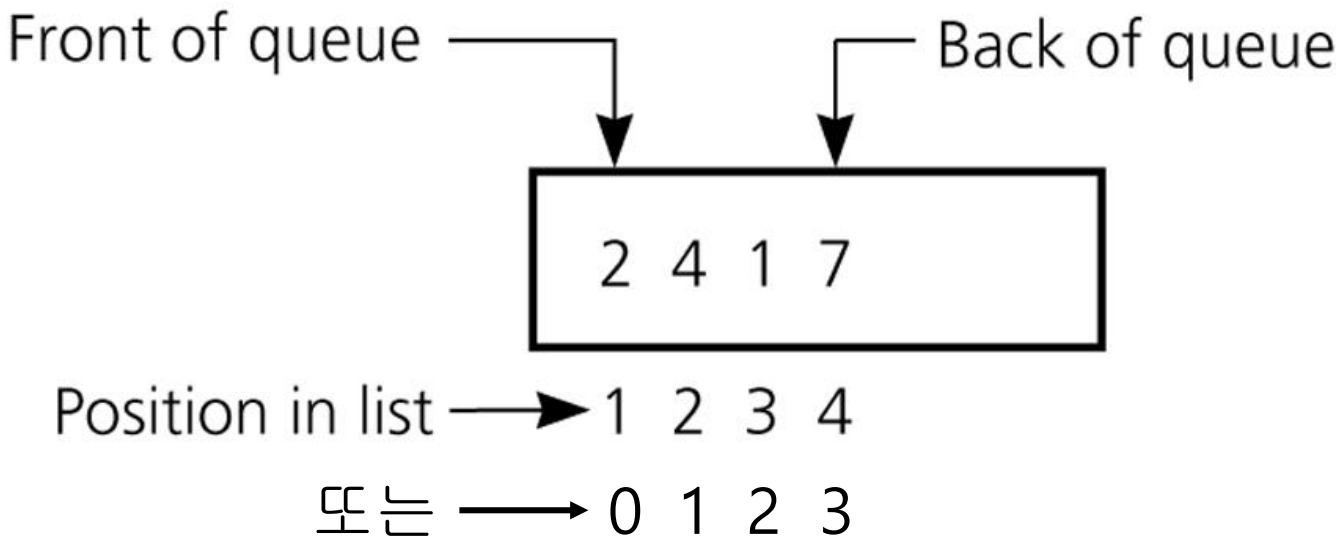
```
    public void dequeueAll ( ) {
                lastNode = null;
    }
    public Object peek( ) {
                if (!isEmpty( )) {
                        // lastNode.getNext( ) is the first node
                        return lastNode.getNext( ).getItem( );
                } else {exception 처리}
    }
} // class QueueRefenceBased
```

ADT List-Based Implementation

```
public class QueueListBased implements QueueInterface {
        private ListInterface list;

        public QueueListBased( ) {
                list = new ListReferenceBased( );
        }
```

✓Class *ListReferenceBased*: Ch.4, 강의 노트 P.13 참조

An implementation that uses the ADT list



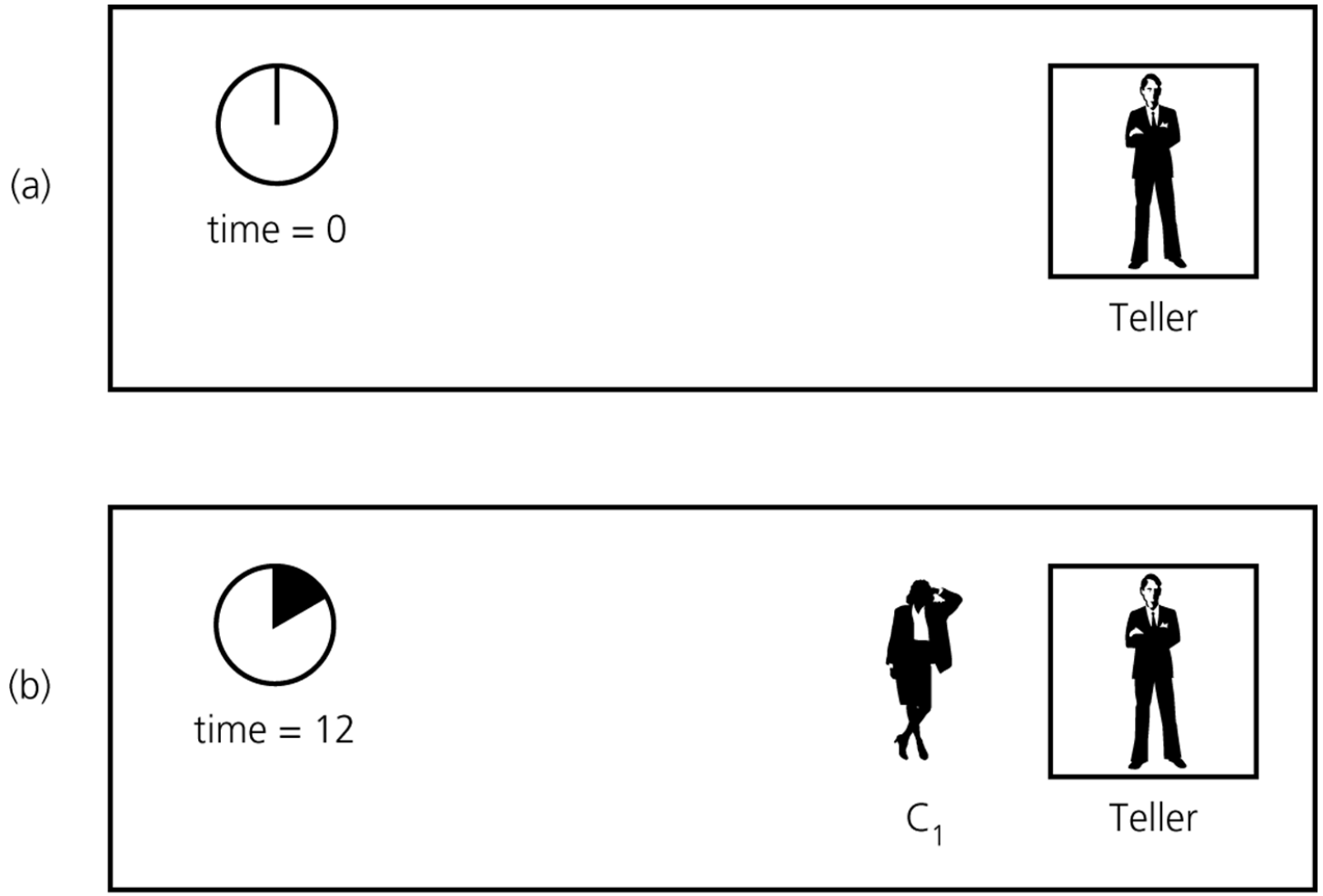
```
public boolean isEmpty( ) {
        return list.isEmpty( );
}
public void enqueue(Object newItem) {
        list.add(list.size( )+1, newItem);
}
public Object dequeue( ) {
        if (!list.isEmpty( )) {
                Object queueFront = list.get(1);
                list.remove(1);
                return queueFront;
        } else {exception 처리}
}
```

```
        public void dequeueAll( ) {
                list.removeAll( );
        }
        public Object peek( ) {
                if (!isEmpty( )) return list.get(1);
                else {exception 처리}
        }
} // class QueueListBased
```

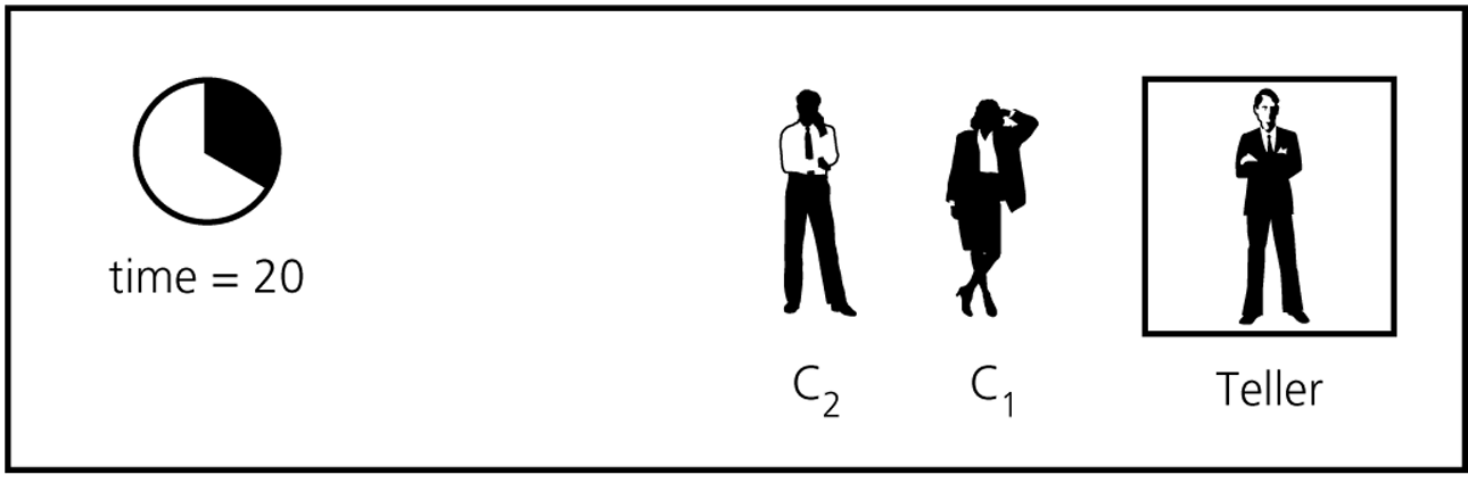
- ✓ ADT list-based version
  reuses the class ListReferenceBased.
- ✓ The reuse made the code simple.

Queue Application

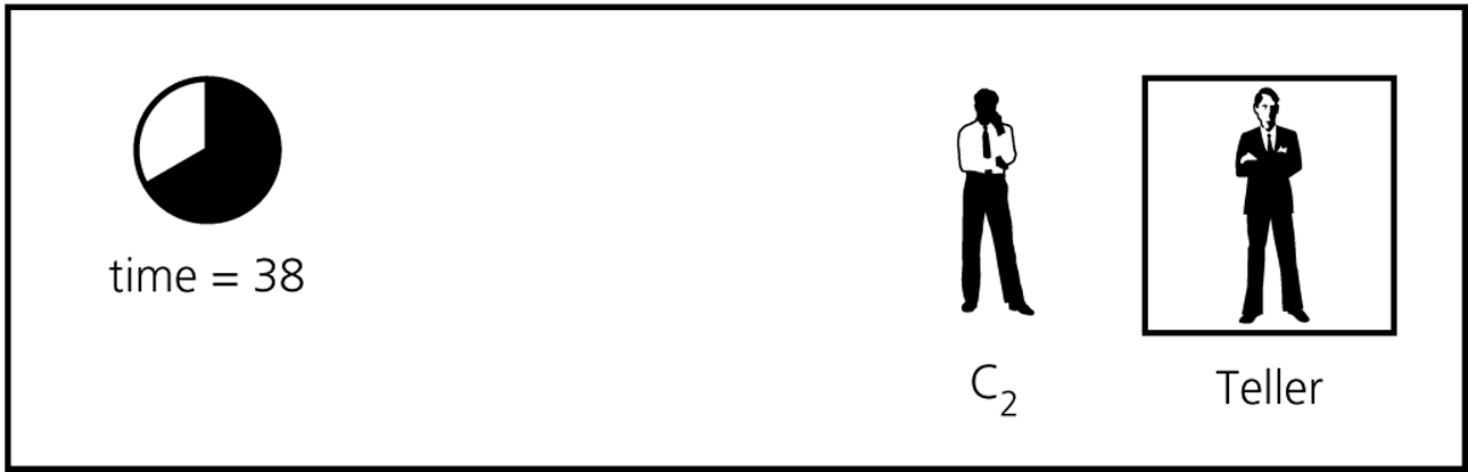
Bank



(c)

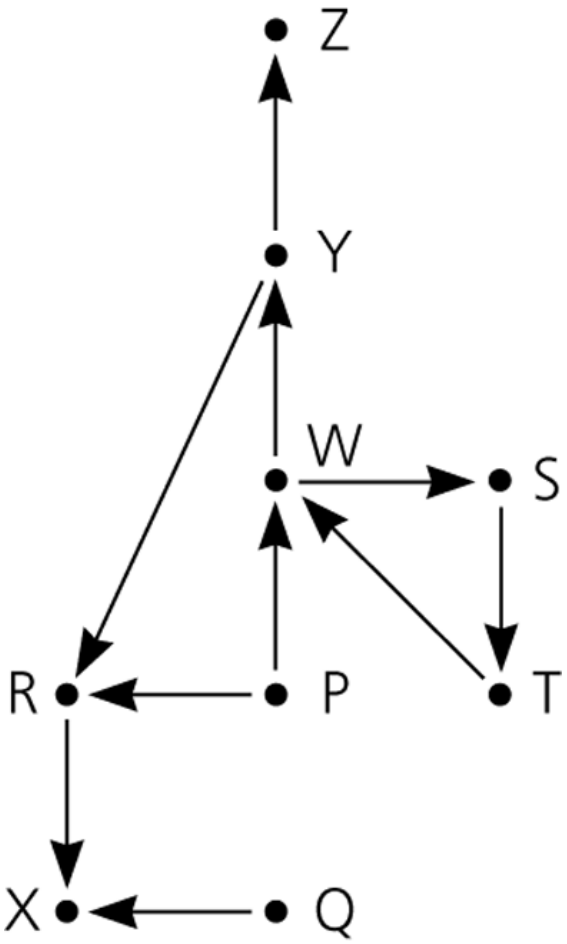


(d)



Queue Application

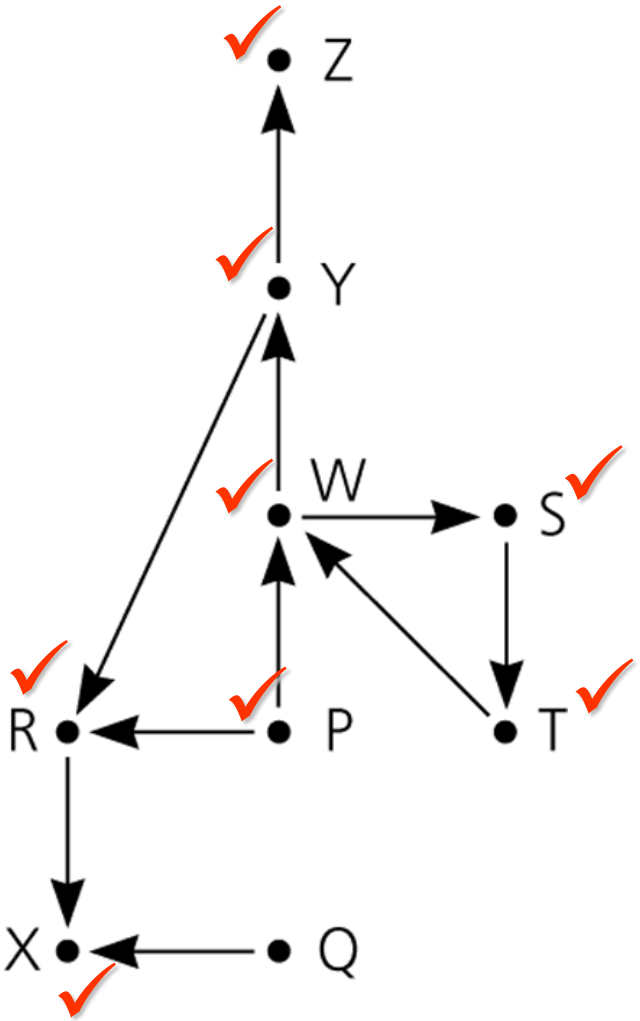
A Search Problem in Flight Map



↑ : direct flight

Is there a path
from city A to city B?

```
searchS(originCity, destinationCity)
{       // All cities are marked UNVISITED in the beginning
        queue.enqueue(originCity);
        mark[originCity] = VISITED;

        while (!queue.isEmpty( )) {
                fCity = queue.dequeue( );
                for all unvisited cities C that is directly reachable from fCity
                {
                        if (C != destinationCity) {
                                queue.enqueue(C);
                                mark[C] = VISITED;
                        } else {
                                return true;
                        }
                }
        }
        // There is no path
        return false;
}
```



P

RW

WX

XSY

SY

YT

TZ

Z

DFS : BFS

- Depth-First Search (DFS)
  - 앞의 stack 방식의 search
- Breadth-First Search (BFS)
  - 앞의 queue 방식의 search
- Covered in Chapter 15