NANYANG TECHNOLOGICAL UNIVERSITY

CE1007/CZ1007 DATA STRUCTURES

Lecture 5B: **Queues**

Dr. Owen Noel Newton Fernando

College of Engineering

School of Computer Science and Engineering

OUTLINE

- Examples of Queue
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Array-based Queue Implementation
- Worked examples: Applications

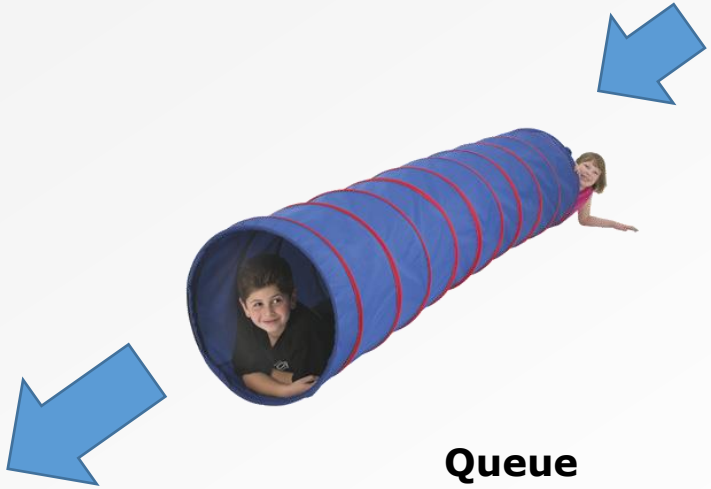
YOU SHOULD BE ABLE TO...

- Explain how a queue data structure operates
- Implement a queue using a linked list
- Choose a queue data structure when given an appropriate problem to solve
- You should also be able to implement a queue using an array (but we won't cover)

LINKED LIST, QUEUE & STACK

Linked List

Stack



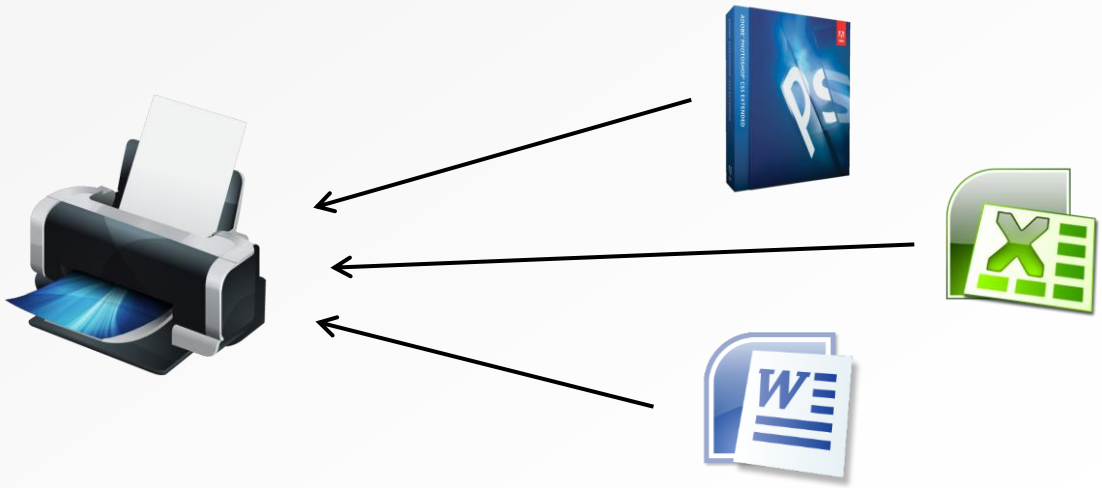
Queue

OUTLINE

- **Examples of Queue**
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Array-based Queue Implementation
- Worked examples: Applications

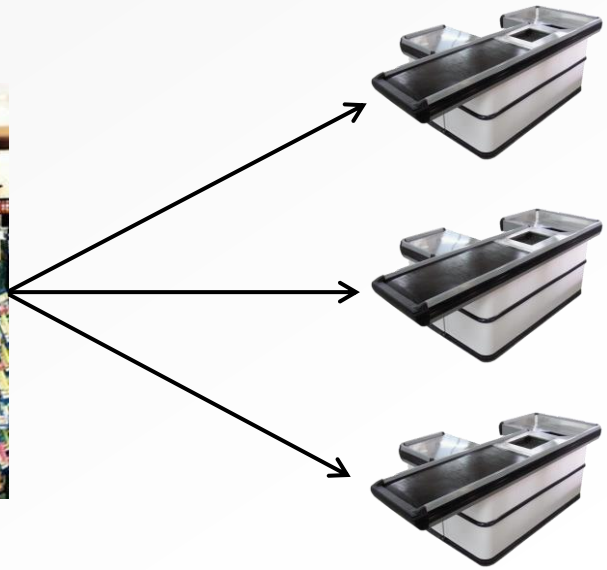
EXAMPLES OF QUEUE #1

- Write a printer driver application:
  - Print jobs may be sent to the printer driver at any time
  - A print job must be stored until it can be sent to the printer
  - Print jobs are sent in first-come, first-served order to the printer
  - Print jobs take some time to complete
    - When a print job completes, the next waiting print job should be sent to the printer



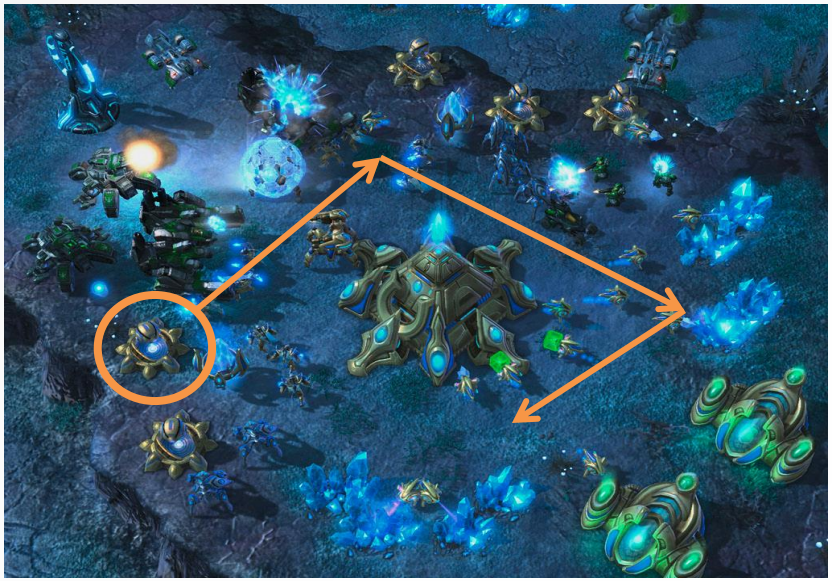
EXAMPLES OF QUEUE #2

- Supermarket checkout counter assignment
  - 1 checkout counter OR N checkout counters
  - Single queue of customers
  - First-come, first-served basis
    - Join the back of the queue and wait for your turn



EXAMPLES OF QUEUE #3

- Sequence of commands for a unit in a game
- Commands may be added to the sequence at any time
- Must be carried out in this order
  - Move there
  - Attack
  - Move there
  - Self-destruct



OUTLINE

- Examples of Queue
- **Queue data structure**
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Array-based Queue Implementation
- Worked examples: Applications

PREVIOUSLY

- Arrays
  - Random access data structure
  - **Access any element directly**
    - array[index]
- Linked lists
  - **Sequential access** data structure
  - Have to go through a particular sequence when accessing elements
    - temp->next until you find the right node
- Today, consider one **limited-access** sequential data structure

QUEUE DATA STRUCTURE

- A Queue is a data structure that operates like a real-world queue
  - Elements can only be added at the back
  - Elements can only be removed from the front
- Key: **First-In, First-Out (FIFO) principle**
  - Or, **Last-In, Last-Out (LILO)**
- Often built on top of some other data structure
  - Arrays, linked lists, etc.
  - We'll focus on a linked list-based implementation

QUEUE DATA STRUCTURE

- Core operations
  - Enqueue: Add an item to the back of the queue
  - Dequeue: Remove an item from the front of the queue
- Common helpful operations
  - Peek: Inspect the item at the front of the queue without removing it
  - IsEmptyStack: Check if the queue has no more items remaining
- Corresponding functions
  - **enqueue()**
  - **dequeue()**
  - **peek()**
  - **isEmptyQueue()**
- We'll build a queue assuming that it only deals with integers
  - But as with linked lists, can deal with any contents depending on your code

OUTLINE

- Examples of Queue
- Queue data structure
- **Queue implementation using linked lists**
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Array-based Queue Implementation
- Worked examples: Applications

QUEUE IMPLEMENTATION USING LINKED LISTS

- Recall that we defined a LinkedList structure

```
typedef struct _linkedlist{
     ListNode *head;
     int size;
} Linked List;
```

- Now, define a Queue structure
  - We'll build our queue on top of a linked list

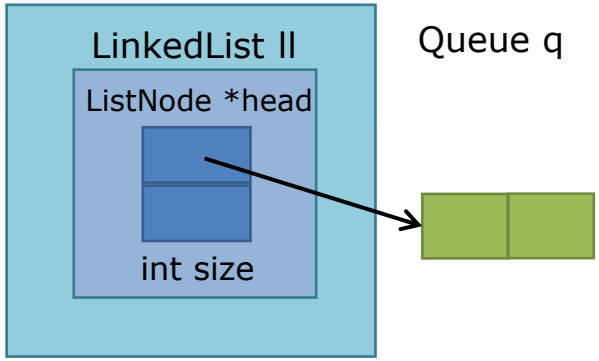
```
typedef struct _queue{
     LinkedList ll;
} Queue;
```

QUEUE IMPLEMENTATION USING LINKED LISTS

- Queue structure

```
typedef struct _queue{
        LinkedList ll;
} Stack;
```

- Again, wrap up a linked list and use it for the actual data storage
- Notice that the LinkedList already takes care of little things like keeping track of number of nodes, etc.
- There is one modification we need for a queue… KIV



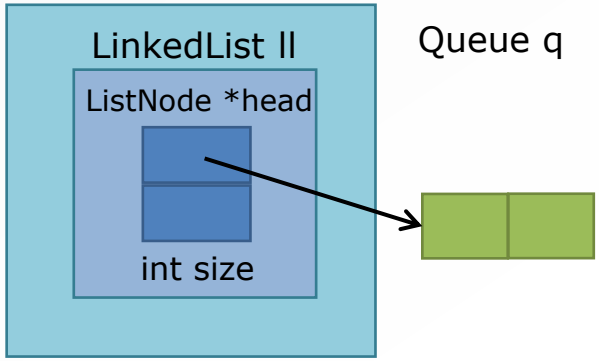
OUTLINE

- Examples of Queue
- Queue data structure
- Queue implementation using linked lists
- **Queue functions**
  - **enqueue()**
  - **dequeue()**
  - **peek()**
  - **isEmptyQueue()**
- Array-based Queue Implementation
- Worked examples: Applications

QUEUE FUNCTION: enqueue()

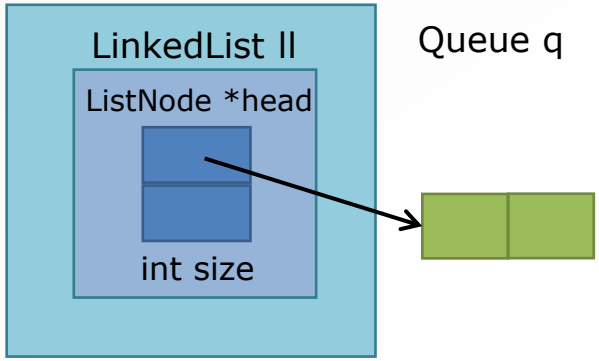
- enqueue() function is the **only** way to add an element to the queue data structure
- Only allowed to enqueue() at the **end**
- Using a linked list as the underlying data storage, the first linked list node represent the front of the queue

  (or represent the end of the queue)



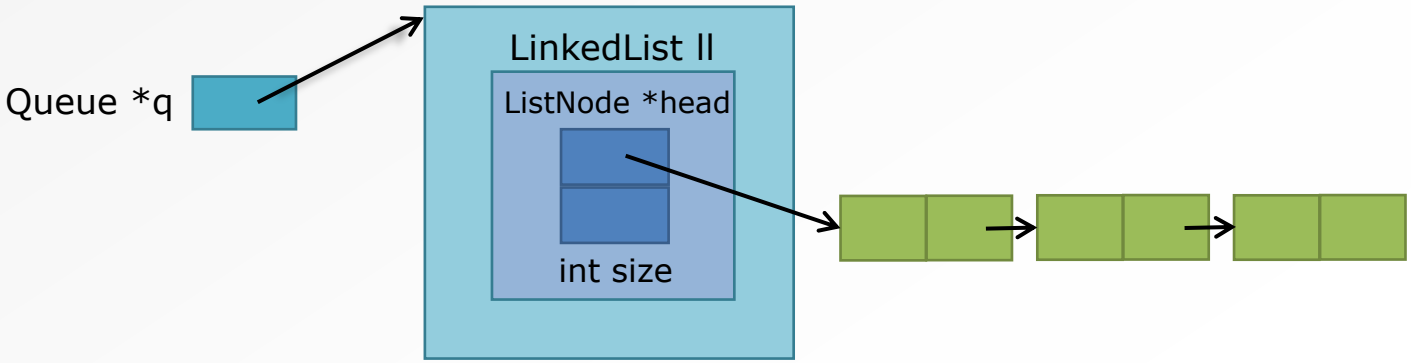
QUEUE FUNCTION: enqueue()

- **Write the enqueue() function**
  - Define the function prototype
  - Implement the function
- Answer is a few slides down, so don't look yet
- Requirements
  - Make use of the LinkedList functions we've already defined
  - Insert at the back only (what index position?)



QUEUE FUNCTION: enqueue()

```
void enqueue(Queue *q,                          ){


}
```



QUEUE FUNCTION: enqueue()

- First linked list node corresponds to the front of the queue
- Last linked list node corresponds to the back of the queue
- Enqueueing a new item → adding a new node to the end of the linked list

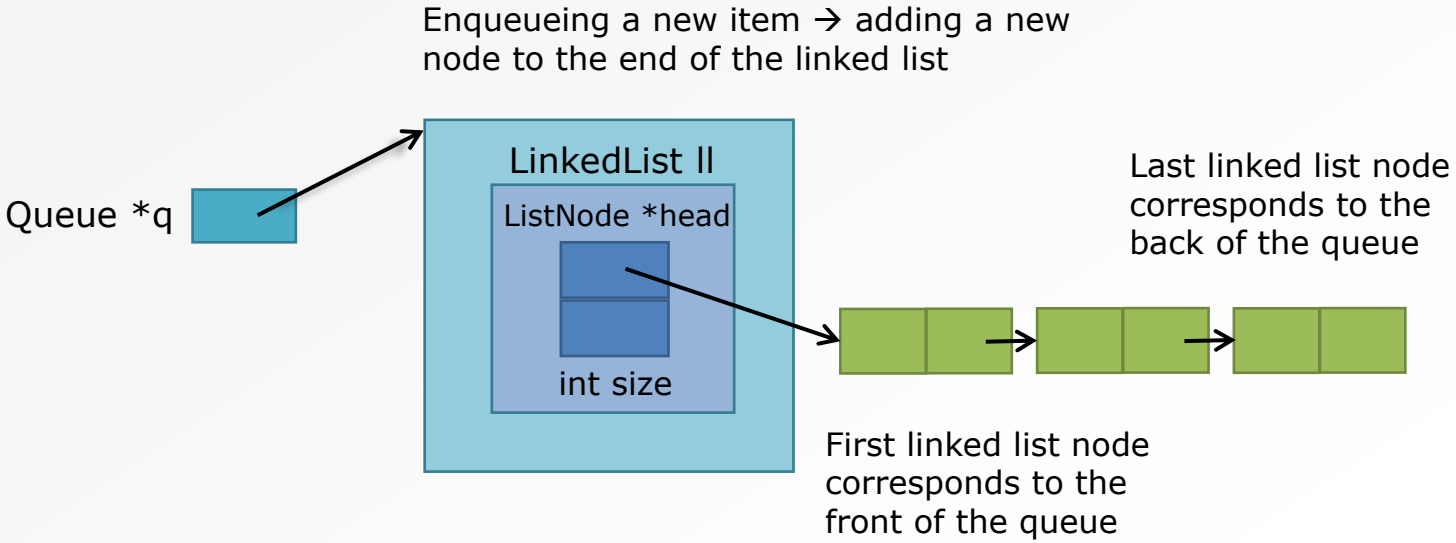
```
void enqueue(Queue *q, int item){
    insertNode(&(q->ll), q->ll.size, item);
}
```

  - Notice that this could be a very inefficient operation if the queue is long
- Need to use a tail pointer to make the operation efficient
  - Gives us direct access to the current last node of the linked list
- Also note that the inefficient version still works

```
int insertNode(LinkedList *ll, int index, int value);
```

QUEUE FUNCTION: enqueue()

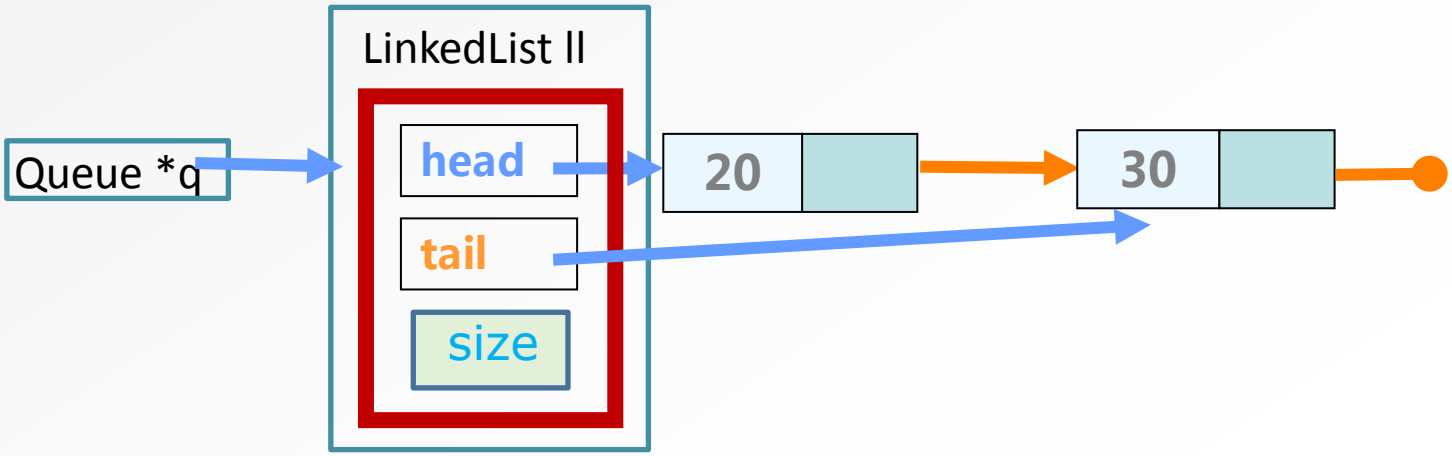
```
void enqueue(Queue *q,      int item      ){

  insertNode(&(q->11), q->11.size, item);

}
```



QUEUE FUNCTION: enqueue()

```
void enqueue(Queue *q, int item){
    insertNode(&(q->ll), q->ll.size, item);
}
```

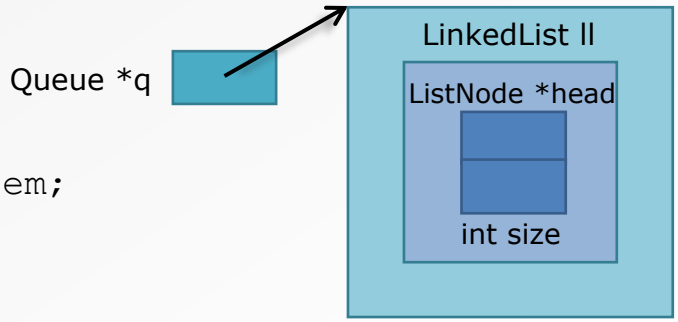
- Notice that this could be a very inefficient operation if the queue is long
- **Need to use a tail pointer** to make the operation efficient
  - Gives us direct access to the last node of the current linked list
- Also note that the inefficient version still works



QUEUE FUNCTION: dequeue()

- Dequeueing a value is a two-step process
  - Get the value of the node at the front of the linked list
  - Remove that node from the linked list

```
int dequeue(Queue *q){
     int item;
     item = ((q->ll).head)->item;
     removeNode(&(q->ll), 0);
     return item;
}
```



- Need a temporary int **item** variable to hold the stored value because we can't get it after we remove the front node

```
int removeNode(LinkedList *ll, int index);
```

QUEUE FUNCTION: peek()

- No change for items in the queue
- Peek at the value at the front of the queue
  - Get the value of the node at the front of the linked list
    - Without removing the node

```
int peek(Queue *q){
    return ((q->ll).head)->item;
}
```

QUEUE FUNCTION: isEmptyQueue()

- Check to see if number of nodes == 0
- Make use of the built-in size variable in the LinkedList struct

```
int isEmptyQueue(Queue *q){
    if ((q->ll).size == 0) return 1;
    return 0;
}
```

OUTLINE

- Examples of Queue
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- **Array-based Queue Implementation**
- Worked examples: Applications

ARRAY IMPLEMENTATION FOR QUEUES

- A Queue can be implemented with an array because we can only add an item to the back and remove an item from the front
- New C structure:

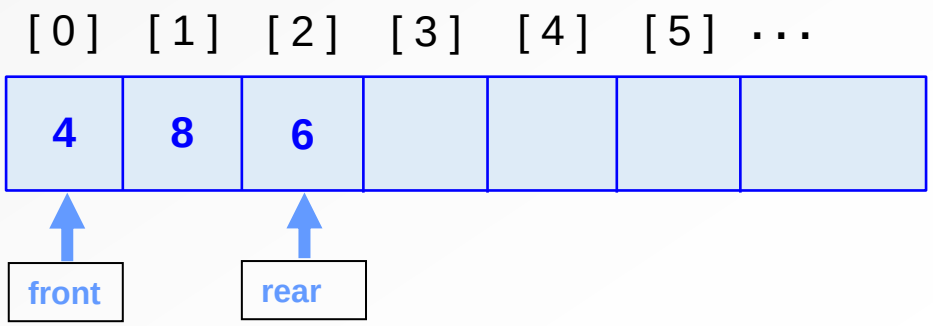
```
typedef struct {

   int num[MAX];
   int front;
   int rear;
   int size;
} queue;
```

The array can store other type of data, such as char, string, etc.

The array is of fixed size: a queue of maximum MAX elements

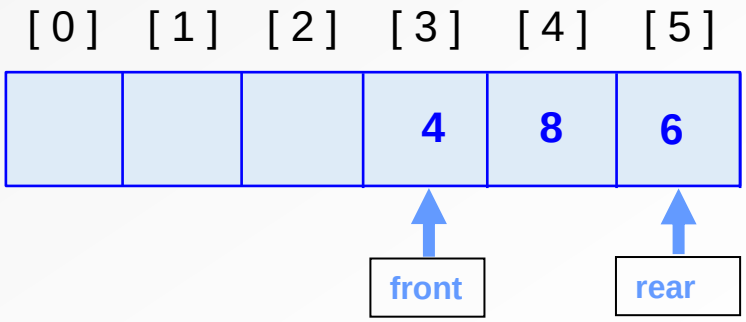
Need to maintain track of both **front** and **rear**

- Functions: Enqueue(), Dequeue(), Peek(), isEmptyQueue(), isFullQueue()



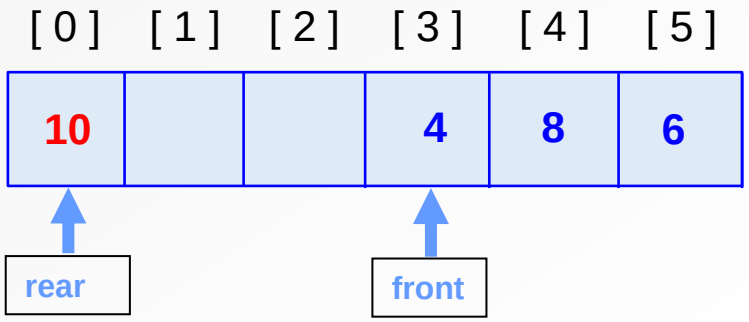
CAN WE ADD AN ITEM TO THE QUEUE?

- The size of the array is MAX=6;
- Rear = 5 = MAX-1, so we can't add one ☹
- But the array has 3 available elements… it is a waste!
- Wrap the array → circular queue



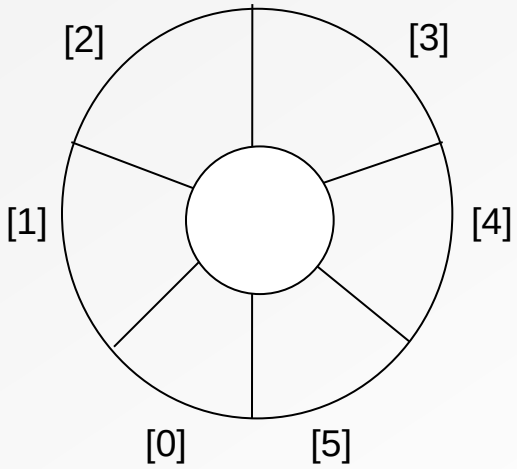
CAN WE ADD AN ITEM TO THE QUEUE?

- The size of the array is MAX=6;
- Rear = 5 = MAX-1, so we can't add one ☹
- But the array has 3 available elements… it is a waste!
- Wrap the array → circular queue



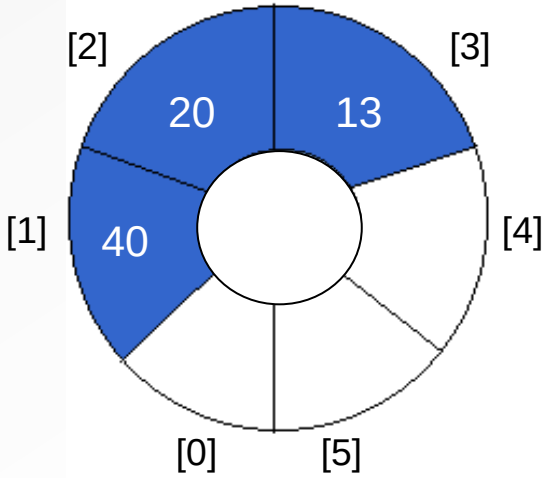
CIRCULAR ARRAY FOR QUEUE

- When queue reaches end of array
  - Add subsequent entries to beginning
- Array behaves as though it were circular
  - First location follows last one



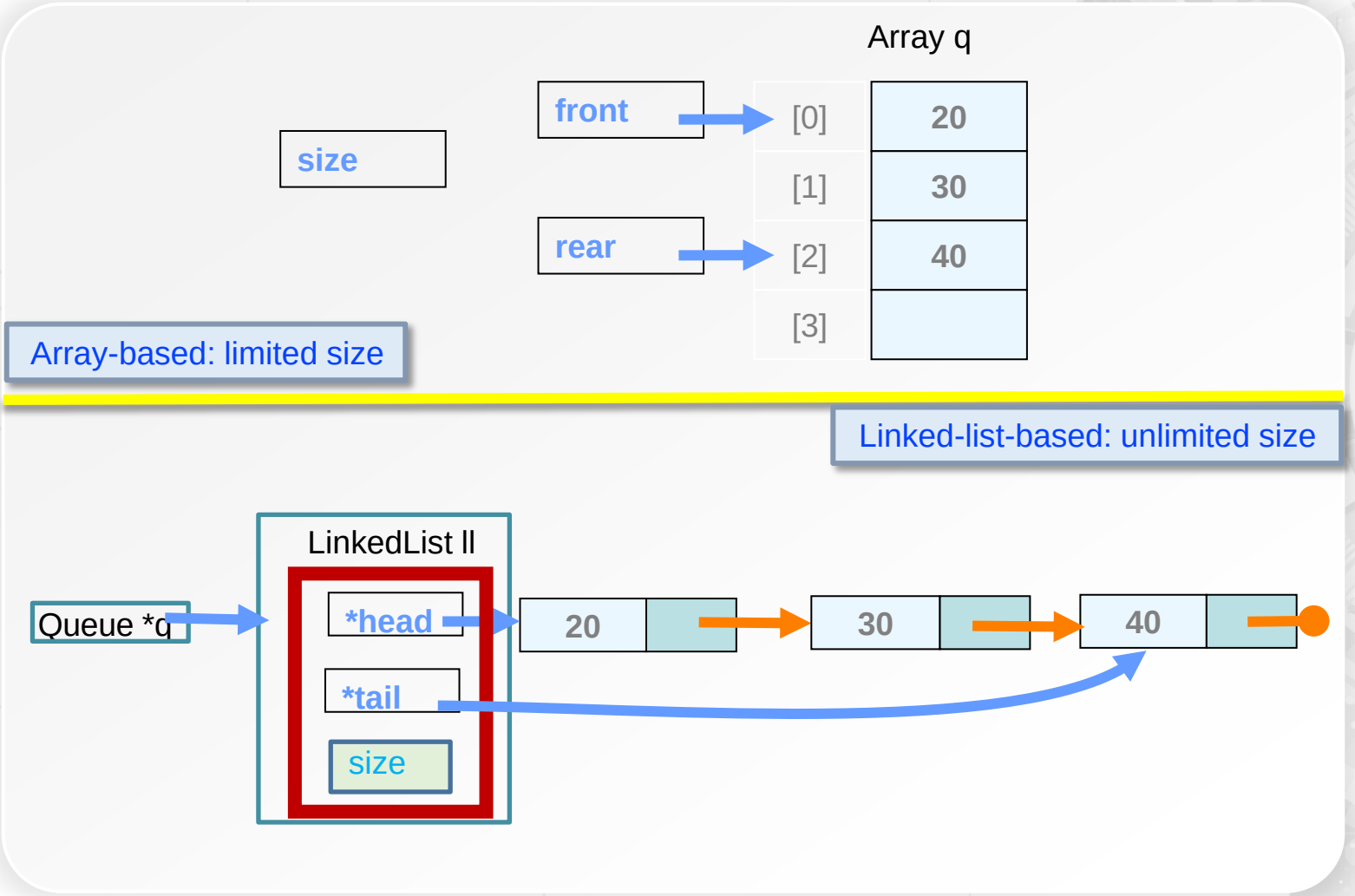
EMPTY QUEUE

front = 0
rear = 0



front = 1
rear = 3

ARRAY- VS LINKED-LIST-BASED QUEUE IMPLEMENTATIONS

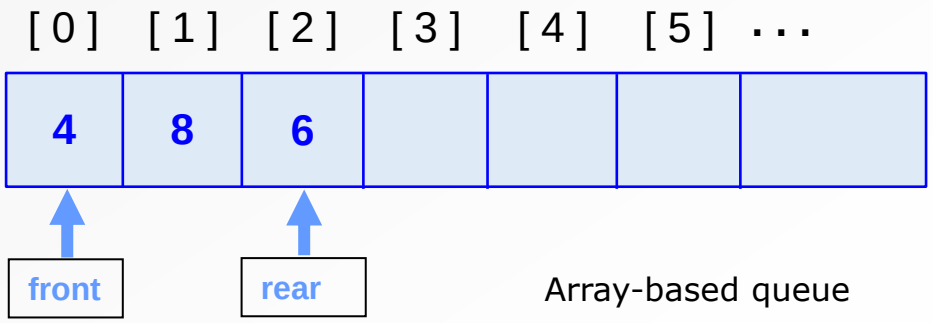


ARRAY- VS LINKED-LIST-BASED QUEUE IMPLEMENTATIONS

- Array-based implementation is simple but:
  - The size of the queue must be determined when the queue object is declared.
  - Space is wasted if we use less elements.
  - Cannot "enqueue" more elements than the array can hold.
- Linked-list-based queue alleviates these problems but time requirements might increase.

REMARKS ON ARRAY-BASED IMPLEMENTATION

- **Easy to implement**, simple coding
- **For memory usage**
  - Save memory: If size of the queue is predetermined, no extra space for pointers.
  - Waste of memory: if we use less elements.
  - Cannot add(enqueue) more elements than the array can hold. it has a limited capacity with a fixed array



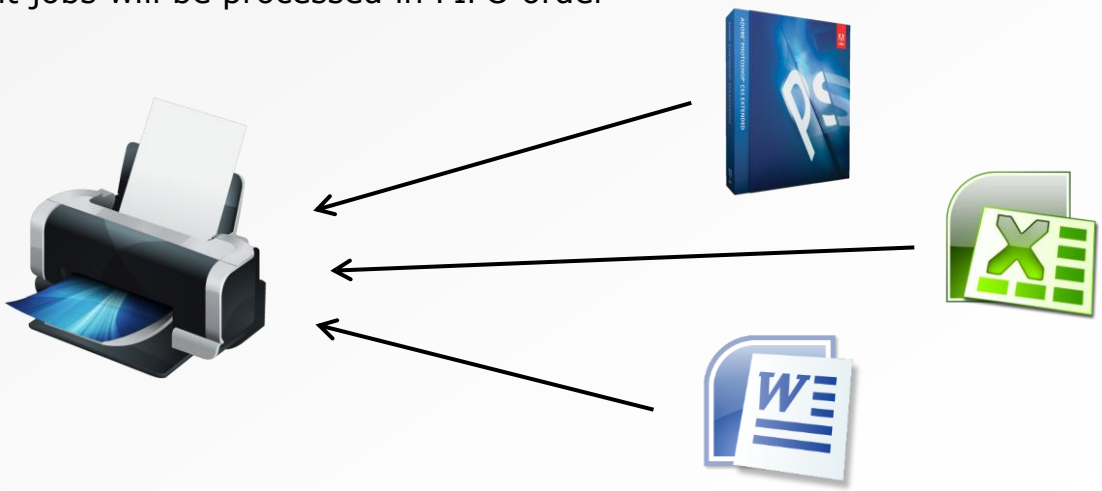
Array-based queue

OUTLINE

- Examples of Queue
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Array-based Queue Implementation
- **Worked examples: Applications**

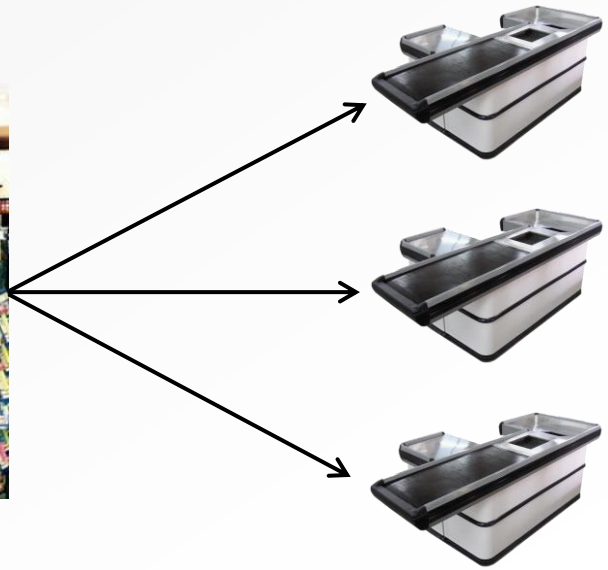
EXAMPLES OF QUEUE #1

- Application sends print job to driver by calling **addPrintJob()**
  - This will **enqueue()** the print job
- When printer finishes the current print job, it calls **getNextPrintJob()**
  - This will **dequeue()** from the queue
- Neither the application nor the printer has to care about other waiting print jobs, etc.
- All print jobs will be processed in FIFO order



EXAMPLES OF QUEUE #2

- To checkout, join the queue at the back
- When any of the checkout counters becomes available, it calls **getNextCustomer()**
- First-come, first-served order of processing guaranteed
- Checkout counters don't have to care about all other waiting customers



SIMPLE APPLICATION

- Simple application
  - Enqueue some integers
  - Dequeue and print


```
1    int main(){
2        Queue q;
3        q.ll.head = NULL;
4        q.ll.tail = NULL;
5
6        enqueue(&q, 1);
7        enqueue(&q, 2);
8        enqueue(&q, 3);
9        enqueue(&q, 4);
10       enqueue(&q, 5);
11       enqueue(&q, 6);
12
13       while (!isEmptyQueue(&q))
14           printf("%d ", dequeue(&q));
15   }
```

OUTLINE

- Examples of Queue
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Array-based Queue Implementation
- Worked examples: Applications

EXERCISE

- Try to re-write insertNode() and removeNode() functions for a Queue implementation using a LinkedList with a tail pointer
- Q:
  - Assume a queue has n items stored at any point
  - How many steps of computation are saved when using a LinkedList with a tail pointer vs a LinkedList without one?