Indian Institute of Information Technology Allahabad

Data Structures and Algorithms

Queues and Lists

Dr. Shiv Ram Dubey
Assistant Professor
Department of Information Technology
Indian Institute of Information Technology, Allahabad

Email: srdubey@iiita.ac.in Web: https://profile.iiita.ac.in/srdubey/

DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Queues and Linked Lists

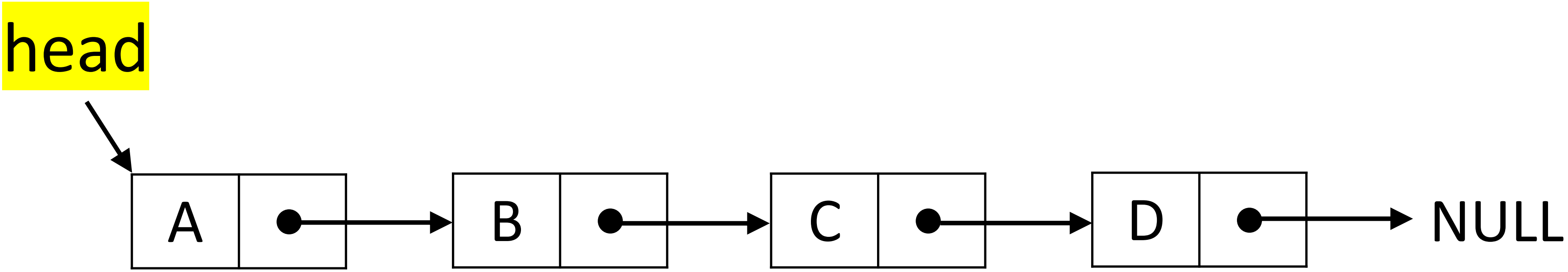
Queues

Linked Lists

Doubly Linked List

Double-Ended Queues

Circular List



Queues

- A queue differs from a stack in that its insertion and removal routines follows the **first-in-first-out (FIFO)** principle.
- Elements may be inserted at any time, but only the element which has been in the queue the longest may be removed.
- Elements are inserted at the rear (enqueued) and removed from the front (dequeued)

ONE WAY
Rear
Front

Queue Abstract Data Type (ADT)

The queue supports following fundamental methods:

- **New():***ADT* – Creates an empty queue
- **Enqueue(S**:*ADT*, **o**:element**)**:*ADT* – Inserts o at the rear of the queue
- **Dequeue(S**:*ADT***)**:*ADT* – Removes the element from the front of the queue, an error occurs when queue is empty, so need to take care.
- **Front(S**:*ADT***)**:*element* – Returns front element without removing it, an error occurs when queue is empty, so need to take care.

Queue Abstract Data Type (ADT)

These support methods should also be defined:

- **Size(S:**_ADT_**):**_integer_
- **IsEmpty(S**:_ADT_**)**:_Boolean_

Queue Abstract Data Type (ADT)

These support methods should also be defined:

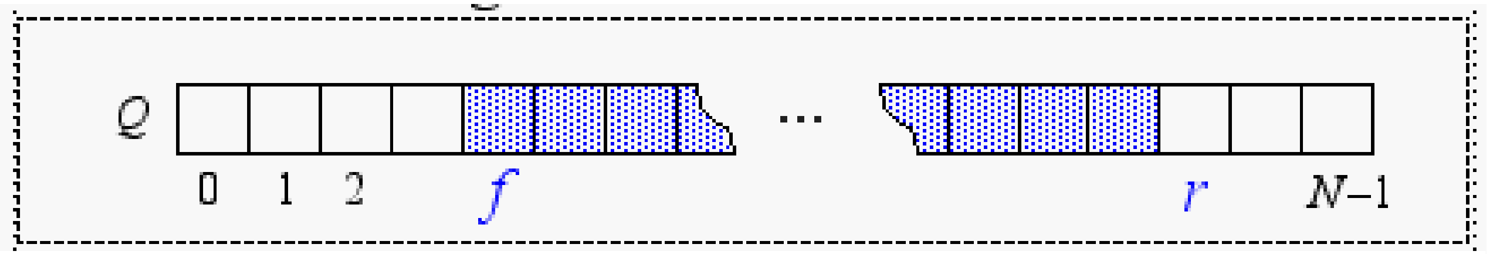
- **Size(S:***ADT***):***integer*
- **IsEmpty(S**:*ADT***)**:*Boolean*

Axioms:

- **Front(Enqueue(New(), v)) = v**
- **Dequeue(Enqueue(New(), v)) = New()**
- **Front(Enqueue(Q, w)) = Front(Enqueue(Enqueue(Q, w),v))**
- **Dequeue(Enqueue(Enqueue(Q, w),v)) = Enqueue(Dequeue(Enqueue(Q, w)),v)**

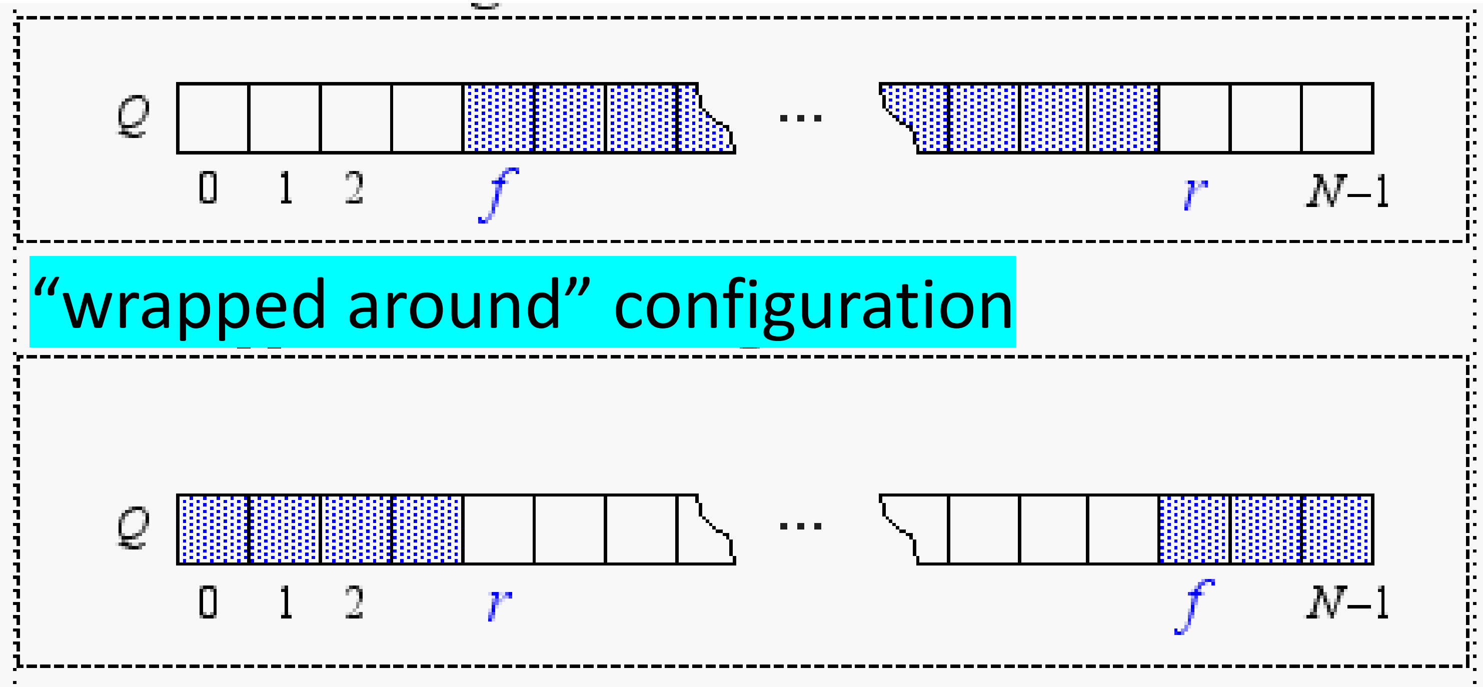
An Array-Based Queue

- Create a queue using an array in a circular fashion
- A maximum size N is specified, e.g. N = 1,000.
- The queue consists of an N-element array Q and two integer variables:
    - *f*, index of the front element (head – for dequeue)
    - *r*, index of the element after the rear one (tail – for enqueue)



An Array-Based Queue

- Create a queue using an array in a circular fashion
- A maximum size N is specified, e.g. N = 1,000.
- The queue consists of an N-element array Q and two integer variables:
  - *f*, index of the front element (head – for dequeue)
  - *r*, index of the element after the rear one (tail – for enqueue)

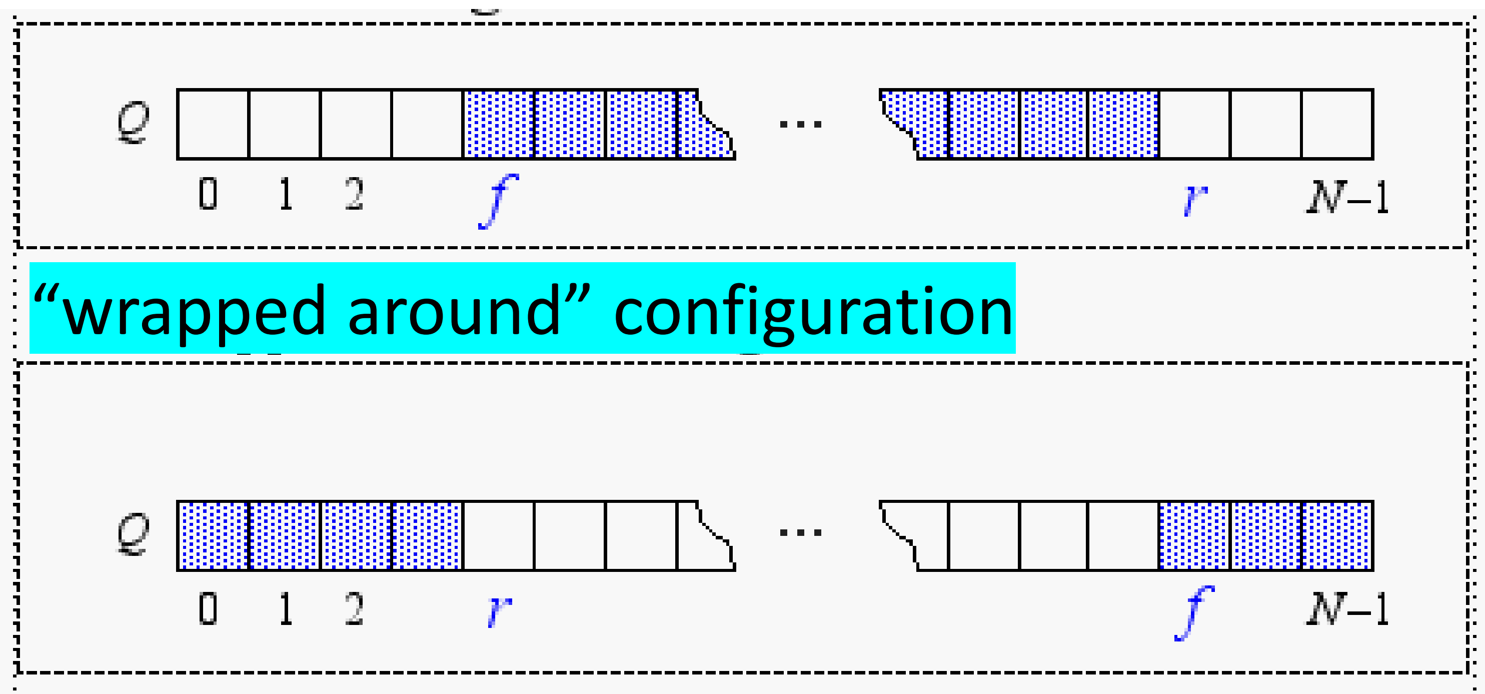


“wrapped around” configuration

An Array-Based Queue

Questions:

What does f==r mean?



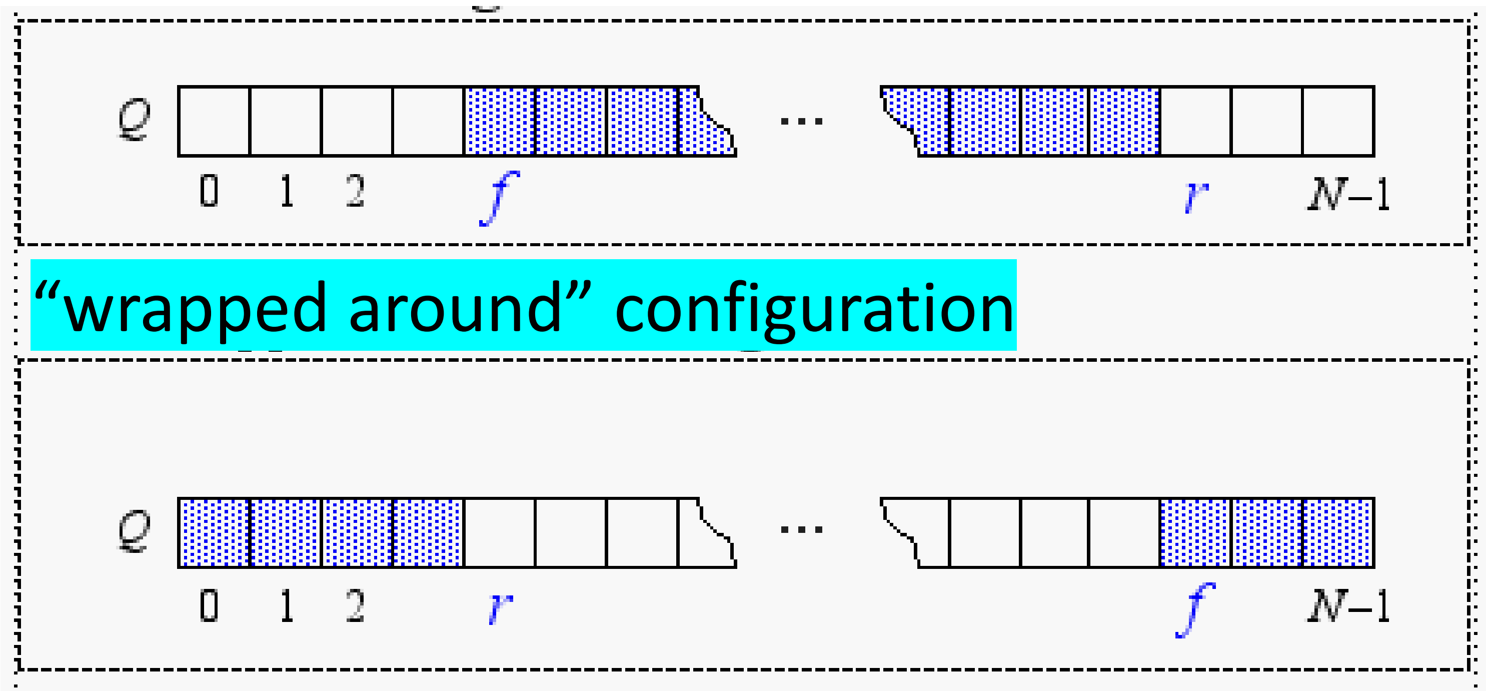
“wrapped around” configuration

An Array-Based Queue

Questions:

What does f==r mean?

Empty

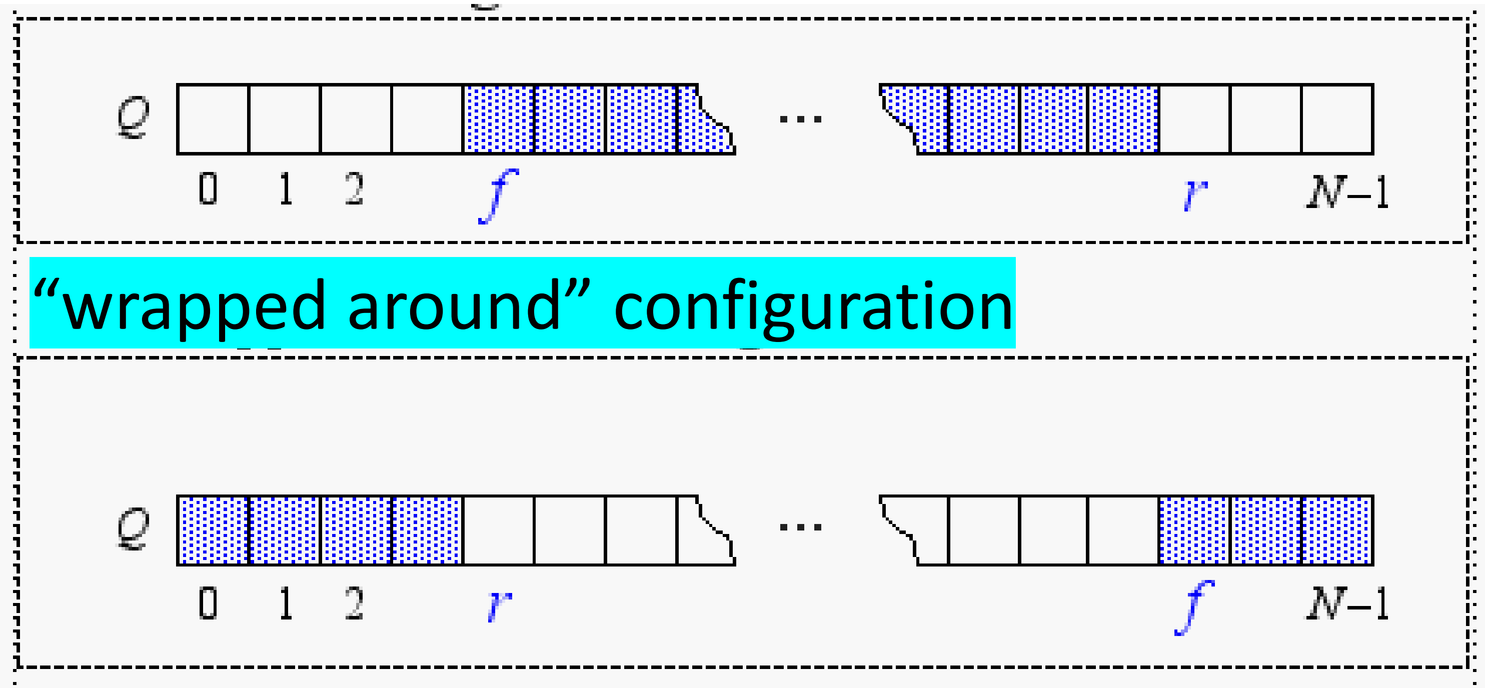


“wrapped around” configuration

An Array-Based Queue

Questions:

How do we compute the number of elements in the queue from f and r?



“wrapped around” configuration

An Array-Based Queue

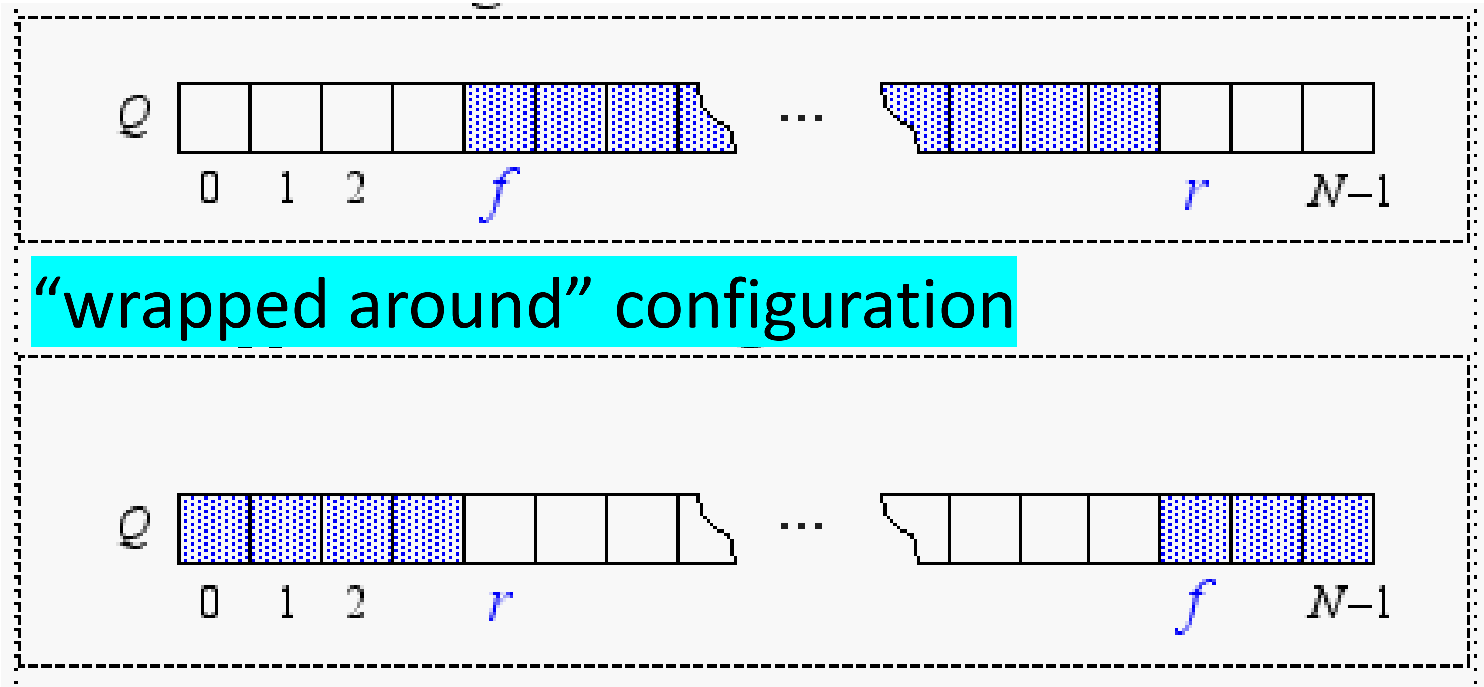
Questions:

How do we compute the number of elements in the queue from f and r?

- *if r > f,*

  #elements = r – f

- *if r < f,*
  - ***#elements = N – f + r***
- *if r == f,*
  - ***#elements = 0***



“wrapped around” configuration

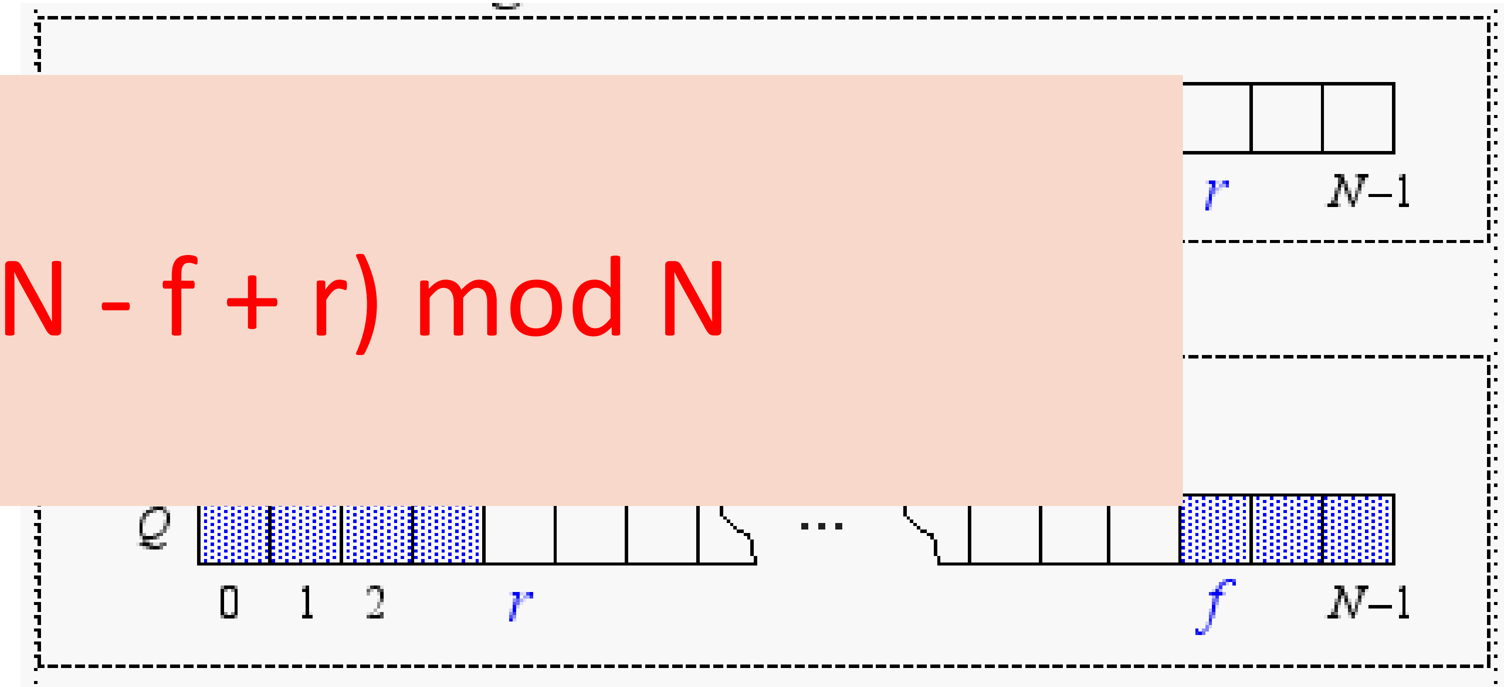
An Array-Based Queue

Questions:

How do we compute the number of elements in the queue from f and r?

- *if r > f*

  #ele
- *if r <*
  - ***#e***
- *if r == f,*
  - ***#elements = 0***



i.e., (N - f + r) mod N

An Array-Based Queue: Pseudo Code

```
Algorithm Size():
    return (N - f + r) mod N
```

```
Algorithm isEmpty():
    return (f == r)
```

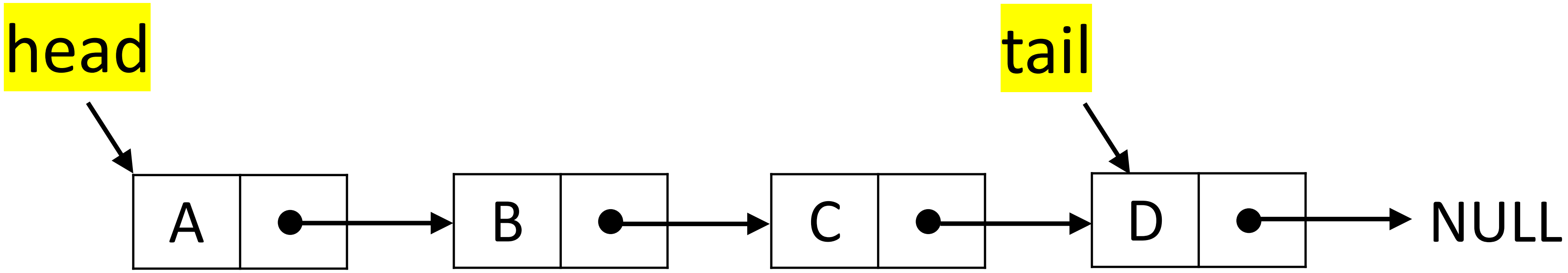
```
Algorithm Front():
    if isEmpty() then
        print “Queue is Empty”
        return NULL
    return Q[f]
```

```
Algorithm Dequeue():
    if isEmpty() then
        print “Queue is Empty”; return NULL
    temp = Q[f]
    Q[f] = null
    f = (f + 1) mod N
    return temp
```

```
Algorithm Enqueue(o):
    if Size() == N - 1 then
        print “Queue is Full”; return
    Q[r] = o
    r = (r + 1) mod N
```

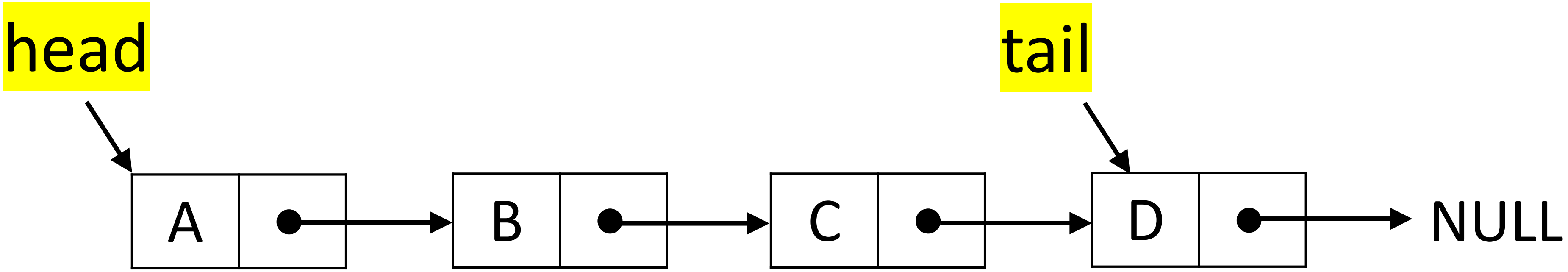
Implementing Queue with a Singly Linked List

- Nodes (*data, pointers*) connected in a chain by links

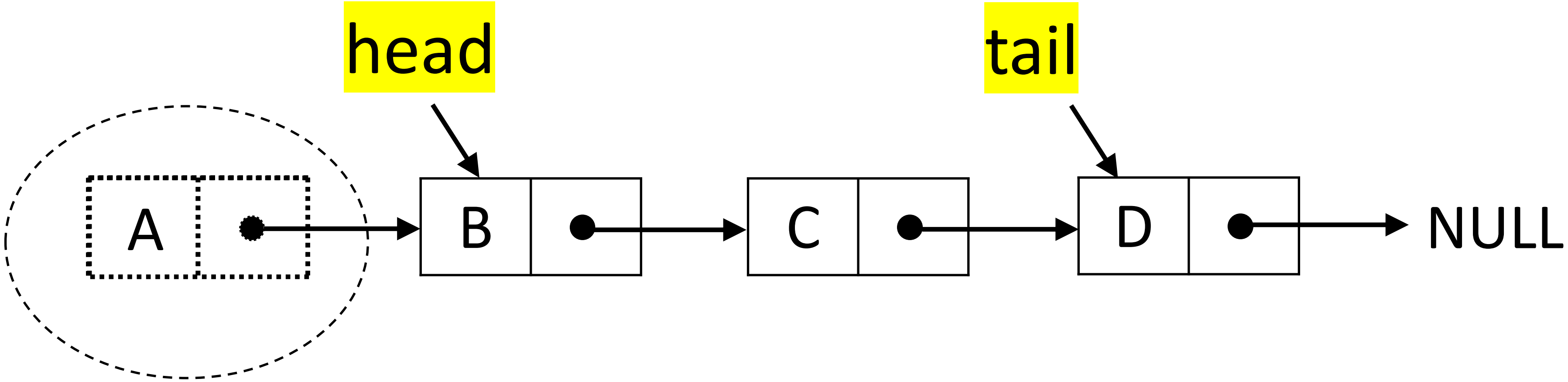


- The head of the list is the front of the queue, the tail of the list is the rear of the queue. ***Why not the opposite?***

Queue: Removing at the Head (Dequeue)



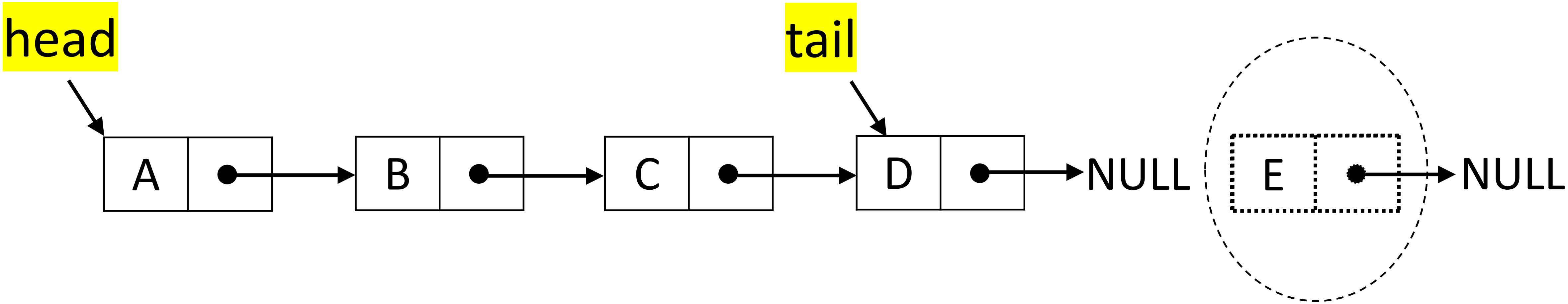
Advance head reference



Inserting at the head is just as easy

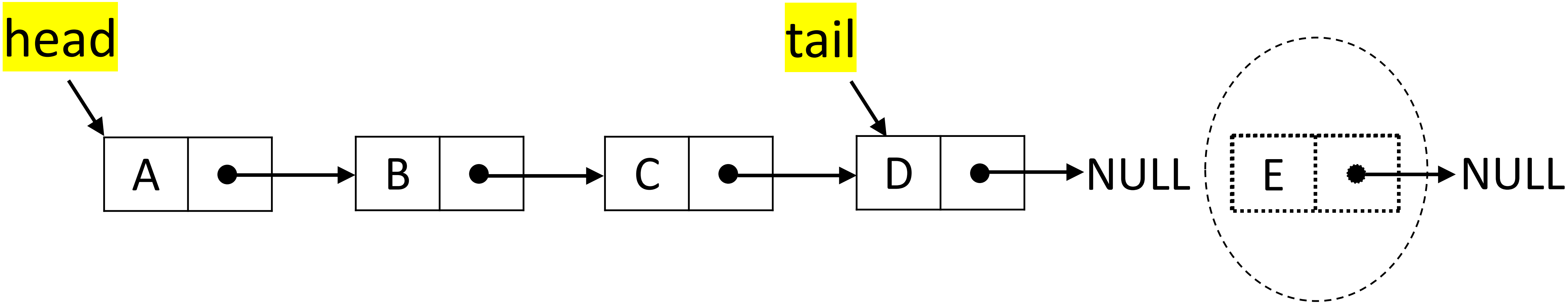
Queue: Inserting at the Tail (Enqueue)

Create a new node

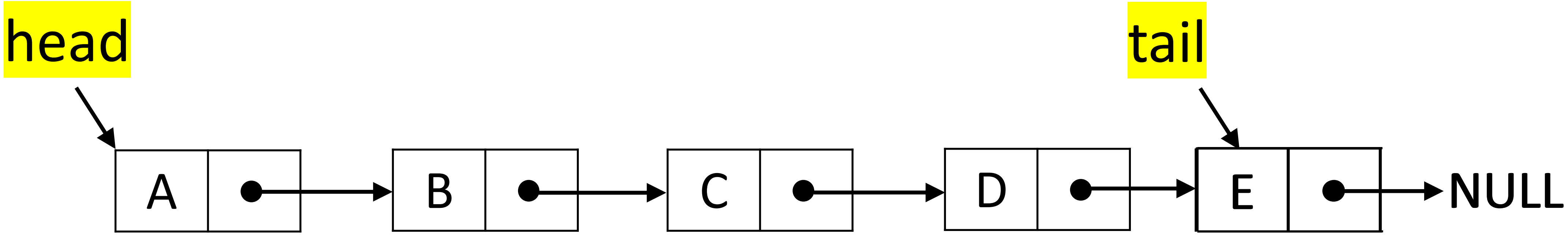


Queue: Inserting at the Tail (Enqueue)

Create a new node



Chain it and move the tail reference



How about removing at the tail?

Double-Ended Queue

- A double-ended queue, or deque, supports insertion and deletion from the front and back.
- The Deque supports following fundamental methods:
    - **insertFirst(S**:*ADT*, **o**:element**)**:*ADT* - Inserts e at the beginning of deque.
    - **insertLast(S**:*ADT*, **o**:element**)**:*ADT* - Inserts e at the end of deque.
    - **removeFirst(S**:*ADT***)**:*ADT* - Removes the first element.
    - **removeLast(S**:*ADT***)**:*ADT* - Removes the last element.
    - **first(S**:*ADT***)**:*element* - Return the first element.
    - **last(S**:*ADT***)**:*element* - Return the last element.

Implementing Deques

With Singly Linked Lists

- Not a good idea
  - As deletion at tail is costly

Implementing Deques

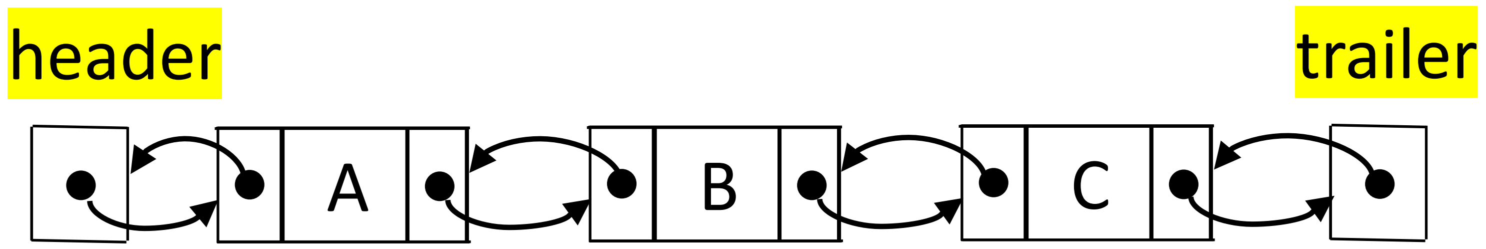
With Singly Linked Lists

- Not a good idea
  - As deletion at tail is costly

Solution: Use Doubly Linked List

Implementing Deques with Doubly Linked Lists

- Deletions at the tail of a singly linked list cannot be done in constant time.
- To implement a deque, we use a doubly linked list with special header and trailer nodes

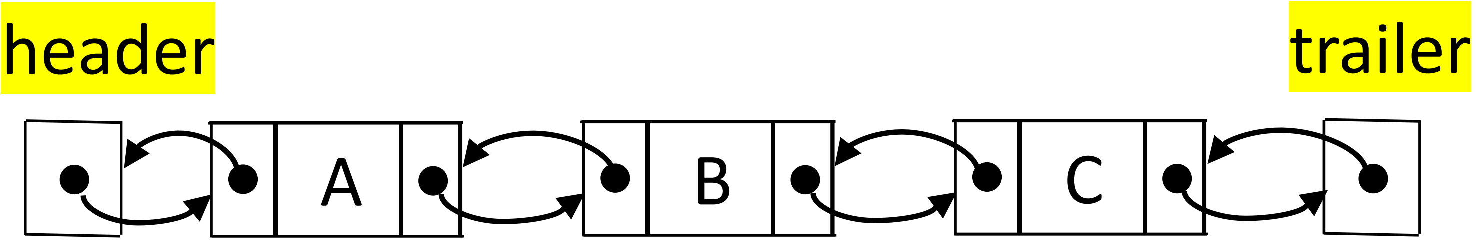


- A node of a doubly linked list has a next and a prev link.
- By using a doubly linked list, all the methods of a deque run in O(1) time.

Implementing Deques with Doubly Linked Lists

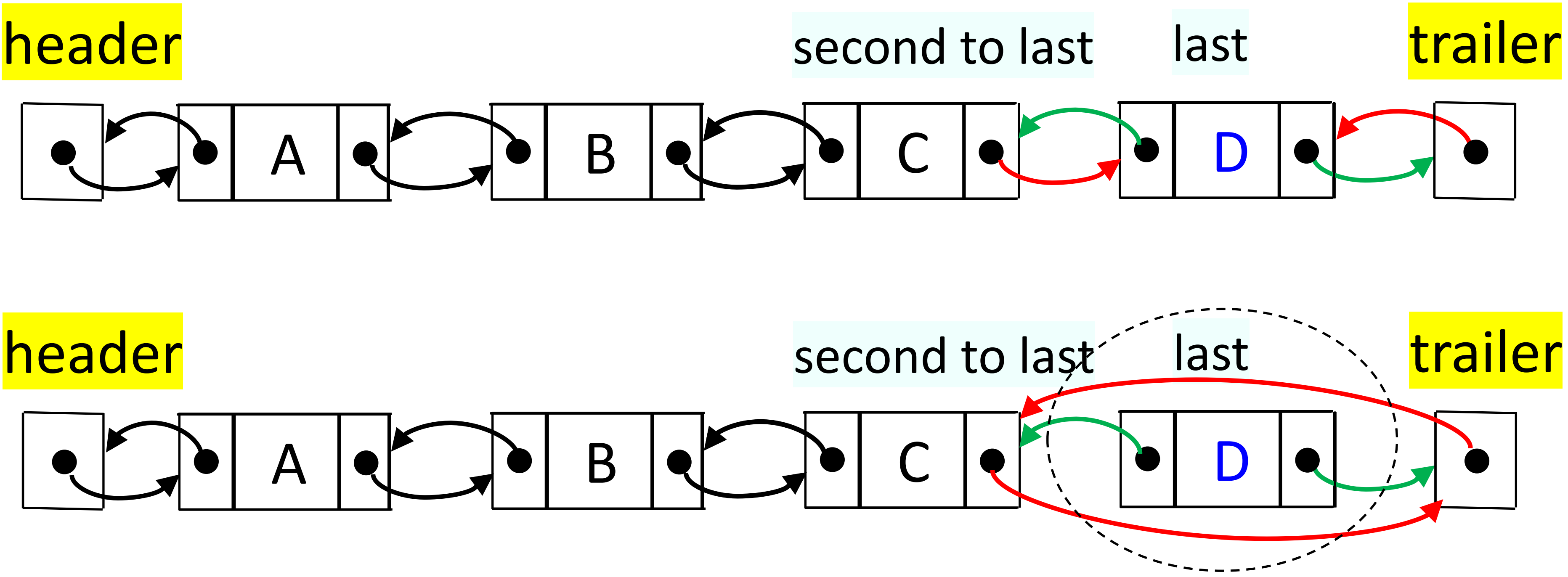
- When implementing a doubly linked lists, we add two special nodes to the ends of the lists: the header and trailer nodes.
    - The ***header*** node goes before the first list element. It has a valid next link but a null prev link.
    - The ***trailer*** node goes after the last element. It has a valid prev reference but a null next reference.

NOTE***: the header and trailer nodes are sentinel or "dummy" nodes because they do not store elements. Here's a diagram of our doubly linked list:***



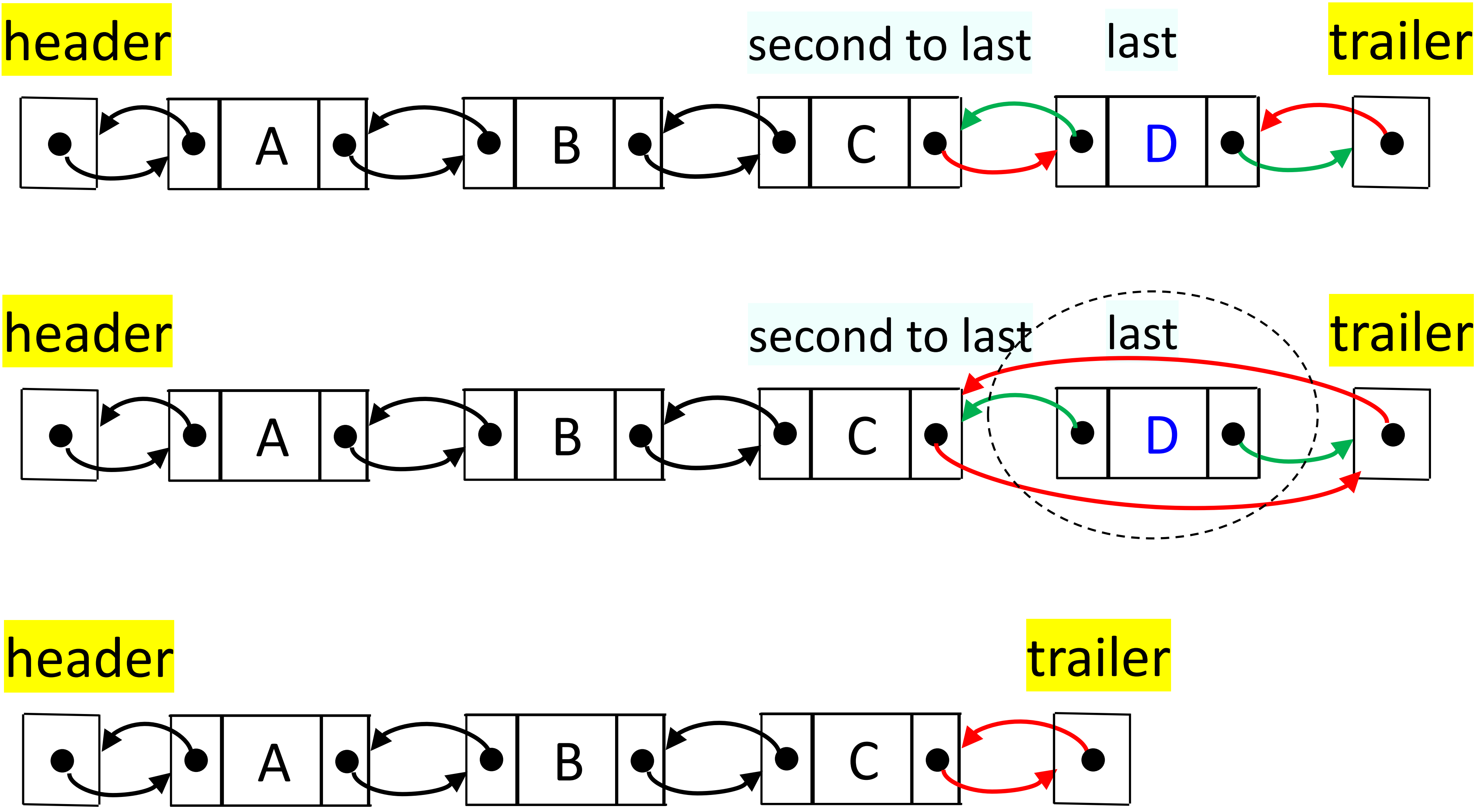
Implementing Deques with Doubly Linked Lists

Here’s a visualization of the code for removeLast().



Implementing Deques with Doubly Linked Lists

Here’s a visualization of the code for removeLast().



Implementing Stacks and Queues with Deques

Implementing ADTs using implementations of other ADTs as building blocks

Implementing Stacks and Queues with Deques

Implementing ADTs using implementations of other ADTs as building blocks

Stacks with Deques:

Stack Method	**Deque Implementation**
size()	size()
isEmpty()	isEmpty()
top()	last()
push(e)	insertLast(e)
pop()	removeLast()

Implementing Stacks and Queues with Deques

Implementing ADTs using implementations of other ADTs as building blocks

Stacks with Deques:

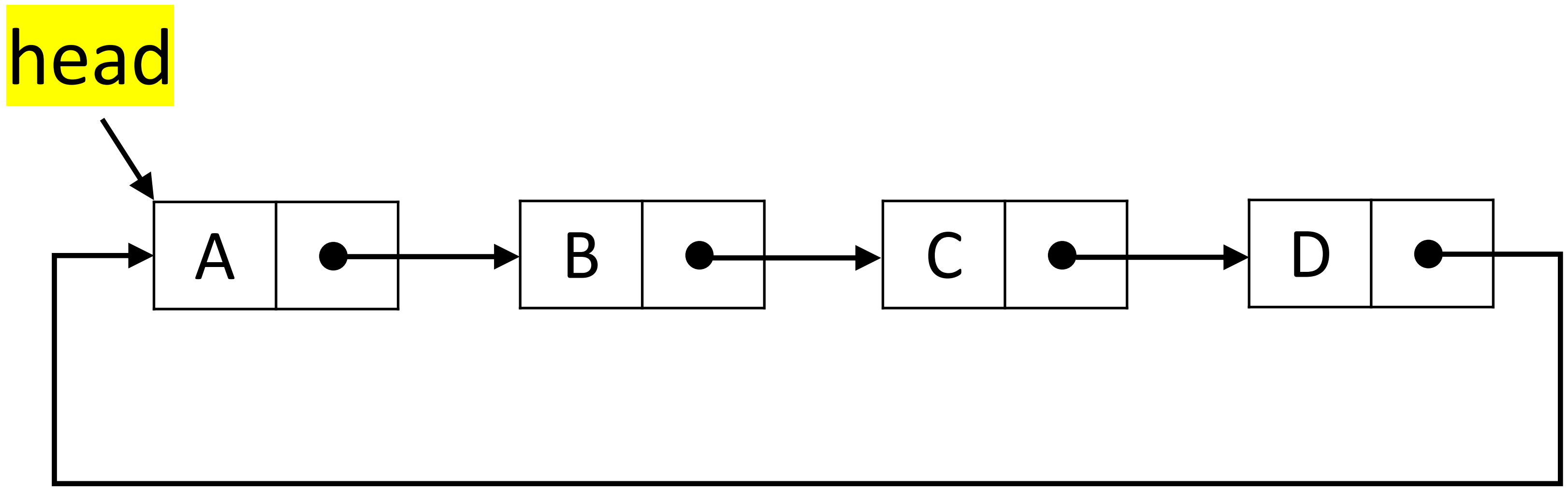
Stack Method	Deque Implementation
size()	size()
isEmpty()	isEmpty()
top()	last()
push(e)	insertLast(e)
pop()	removeLast()

Queues with Deques:

Queue Method	Deque Implementation
size()	size()
isEmpty()	isEmpty()
front()	first()
enqueue(e)	insertLast(e)
dequeue()	removeFirst()

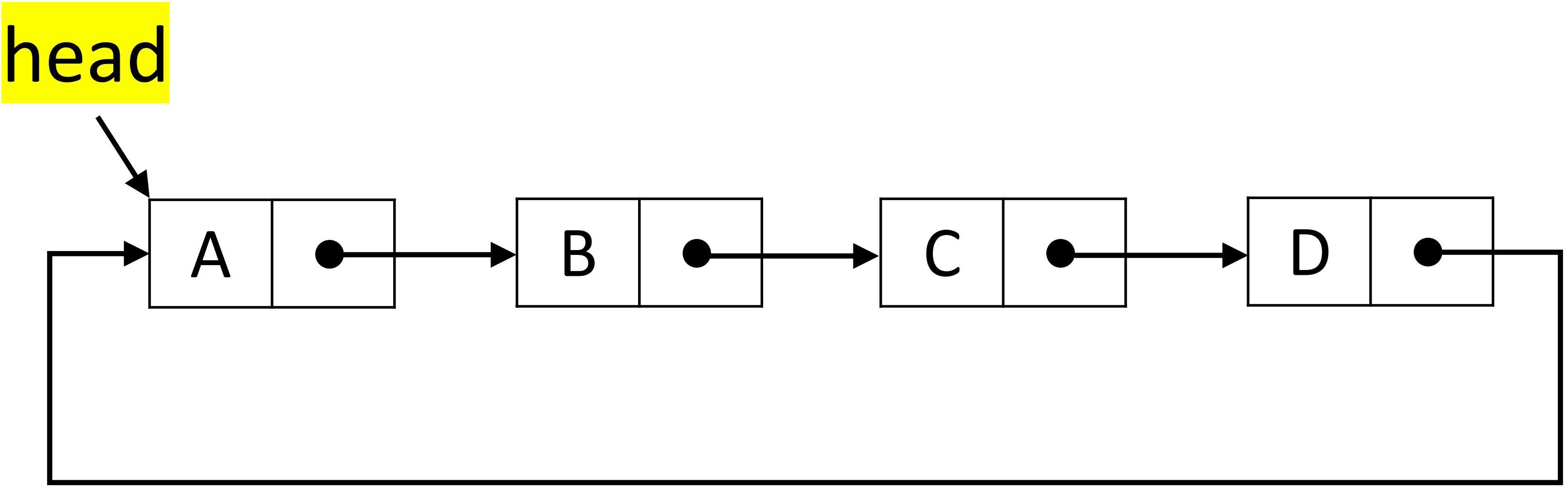
Circular Lists

- No end and no beginning of the list, only one pointer as an **entry point**
- *Circular doubly linked list with a sentinel* is an elegant implementation of a stack or a queue



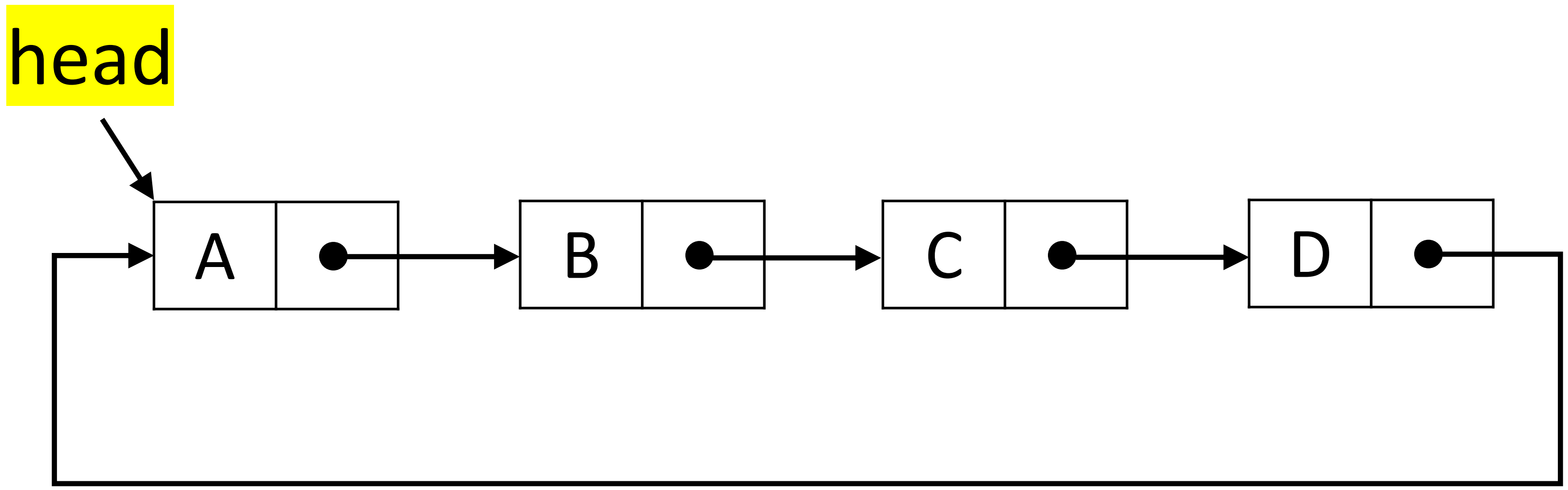
Circular Lists

- **Insertion a node F at head:** create a new node, insert between A and B, copy A to this new node and replace A of head node with F.



Circular Lists

- **Deleting the head node:** copy node B to node A and delete original node B.



Acknowledgement

- IIT Delhi

Thank You