Queue

Queue -Basics

- **Queue - List with access restrictions**
  - **FIFO – First-In First-Out**
  - **Double ended- Head and Tail (front and rear)**
  - **Insertion always to the tail**
  - **Deletion always from the head**

Queue -Implementations

- Array Based
  - Q [1..n]
  - Head, Tail - array indices

- Pointer Based
  - As a linked list
  - Head, Tail - pointers to the nodes at front and rear respectively

Queue - Array Based Implementation #1

- Array Based
  - Array ***Q [1..n]*** - an array of at most n elements
  - An attribute ***Q.head*** – index of the head element
  - An attribute ***Q.tail*** – index of the tail element
  - Elements from ***Q[Q.head..Q.tail]***

Queue – Implementation using Array #1

1	2	3	4	5	6	7	8	9	10	11	12
5	30	20	40								

Q.head = 1

Q.tail = 4

Queue – Implementation using Array #1

1	2	3	4	5	6	7	8	9	10	11	12
5	30	20	40								

⇧ Q.head = 1

⇧ Q.tail = 4

Enqueue(Q, 10)

1	2	3	4	5	6	7	8	9	10	11	12
5	30	20	40	10							

⇧ Q.head = 1

⇧ Q.tail = 5

Enqueue(Q, 15)

1	2	3	4	5	6	7	8	9	10	11	12
5	30	20	40	10	15						

Q.head = 1

Q.tail = 6

Dequeue(Q)

1	2	3	4	5	6	7	8	9	10	11	12
5	30	20	40	10	15						

Q.head = 2

Q.tail = 6

Dequeue(Q)

1	2	3	4	5	6	7	8	9	10	11	12
5	30	20	40	10	15						

Q.head = 3

Q.tail = 6

Queue -Operations

ENQUEUE (Q, x) // check the correctness

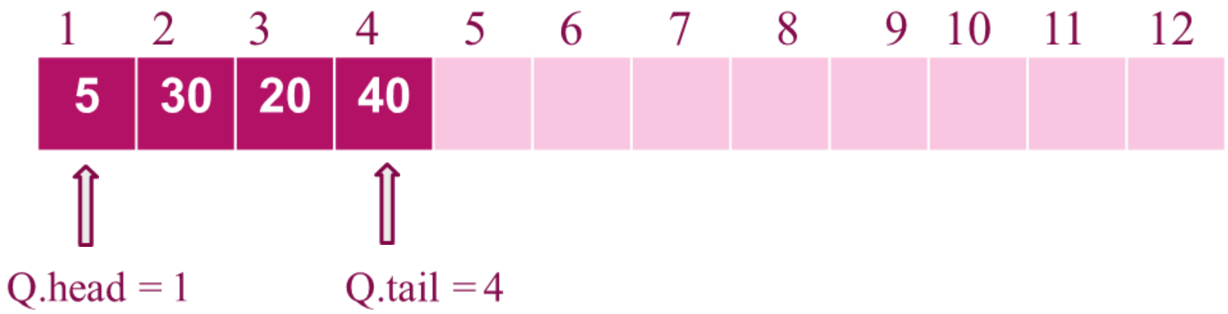
if (QUEUE-FULL(Q))

error “overflow”

else

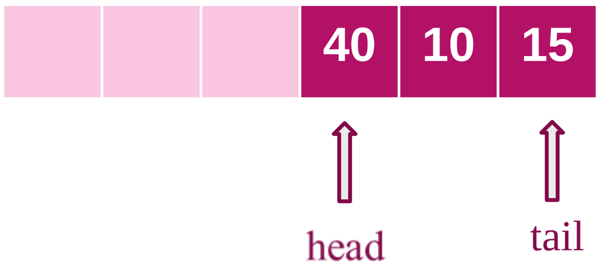
Q. tail= Q. tail + 1

Q [Q. tail] = x



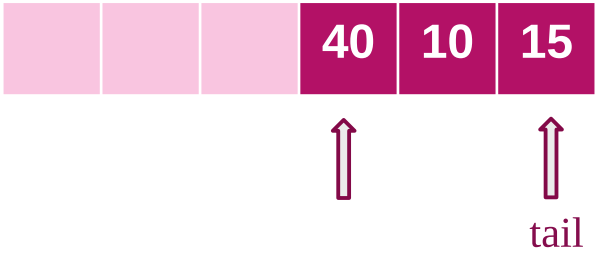
Queue -Operations

QUEUE-FULL(Q) ?



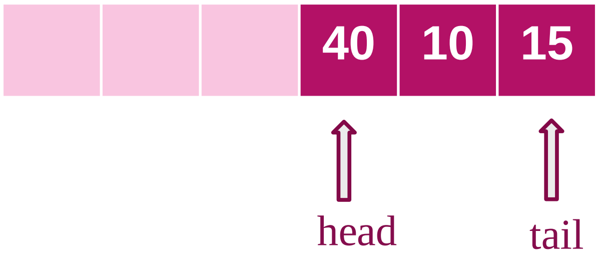
- **No further EnQueue possible even though the queue is not full**

Queue -Operations

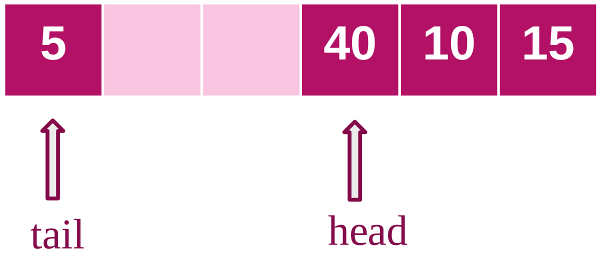


- No further EnQueue possible even though the queue is not full
  - Left Shift the element
    - Takes linear time

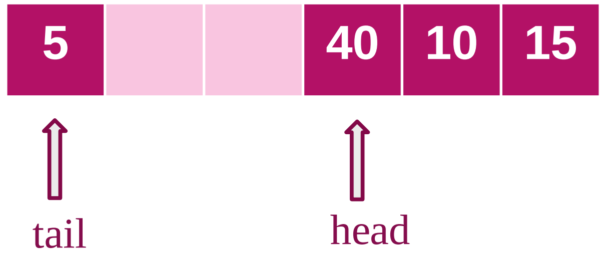
Queue -Operations



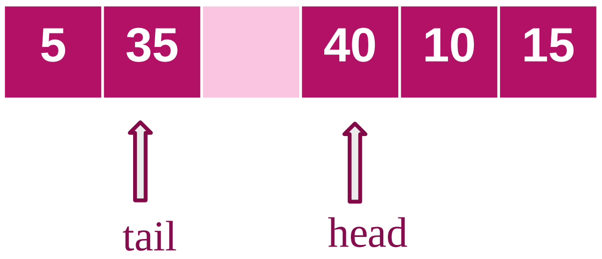
Start adding from left



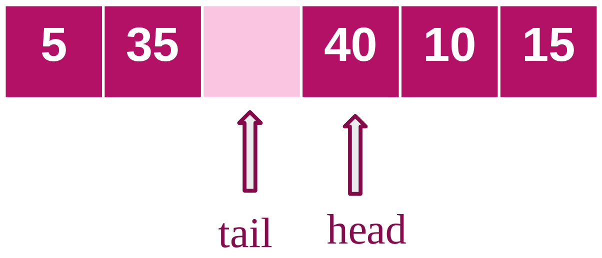
Queue -Operations



EnQueue(Q, 35)



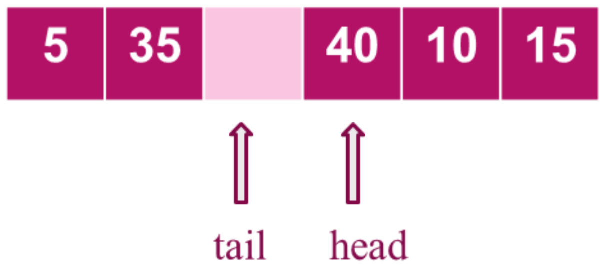
Queue -Operations

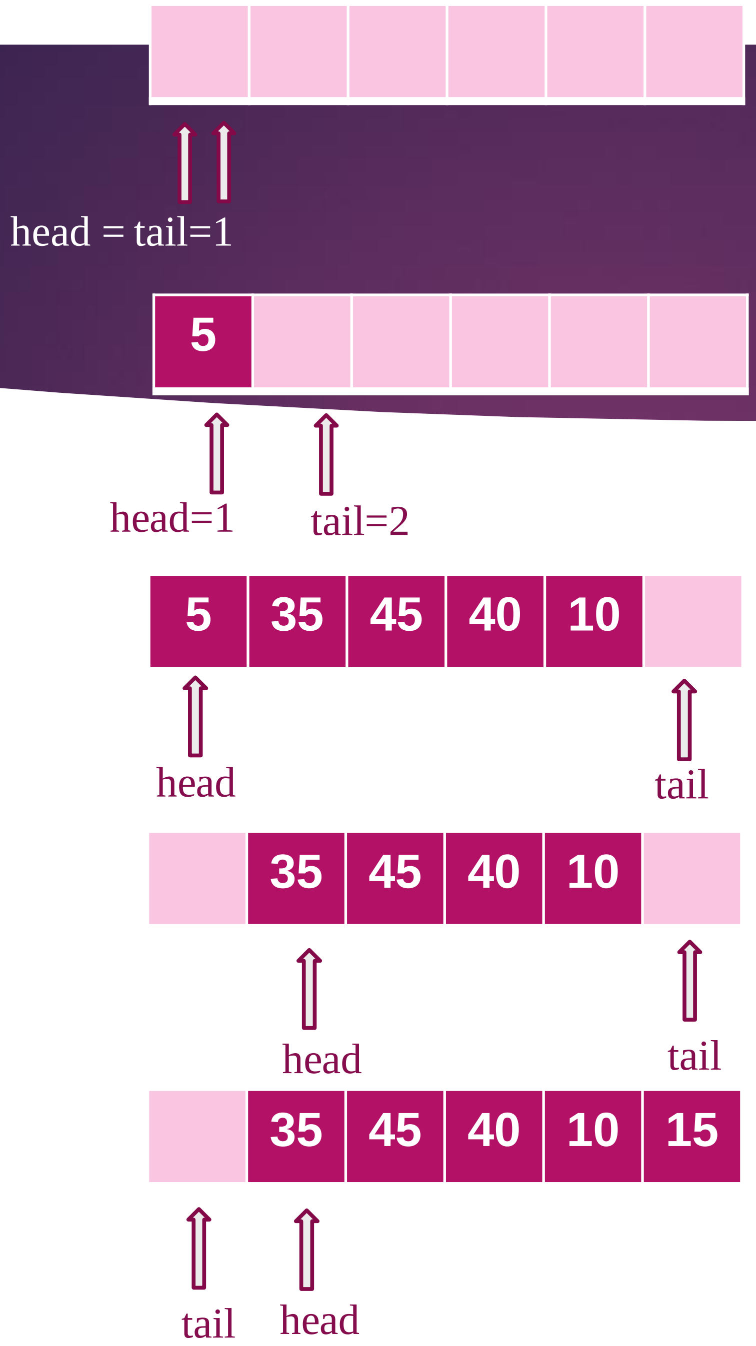


- One slot left vacant (maximum *n-1* elements can be queued in an *n* element array)
- Q.tail points to the next location where a new element can be added
- Makes checking QueueFull() / QueueEmpty() easier

Queue - Array Based Implementation

- Array Q[1..n]
- Q.head points to actual head
- Q.tail points to the next location for insertion
- Initially Q.head = Q.tail = 1



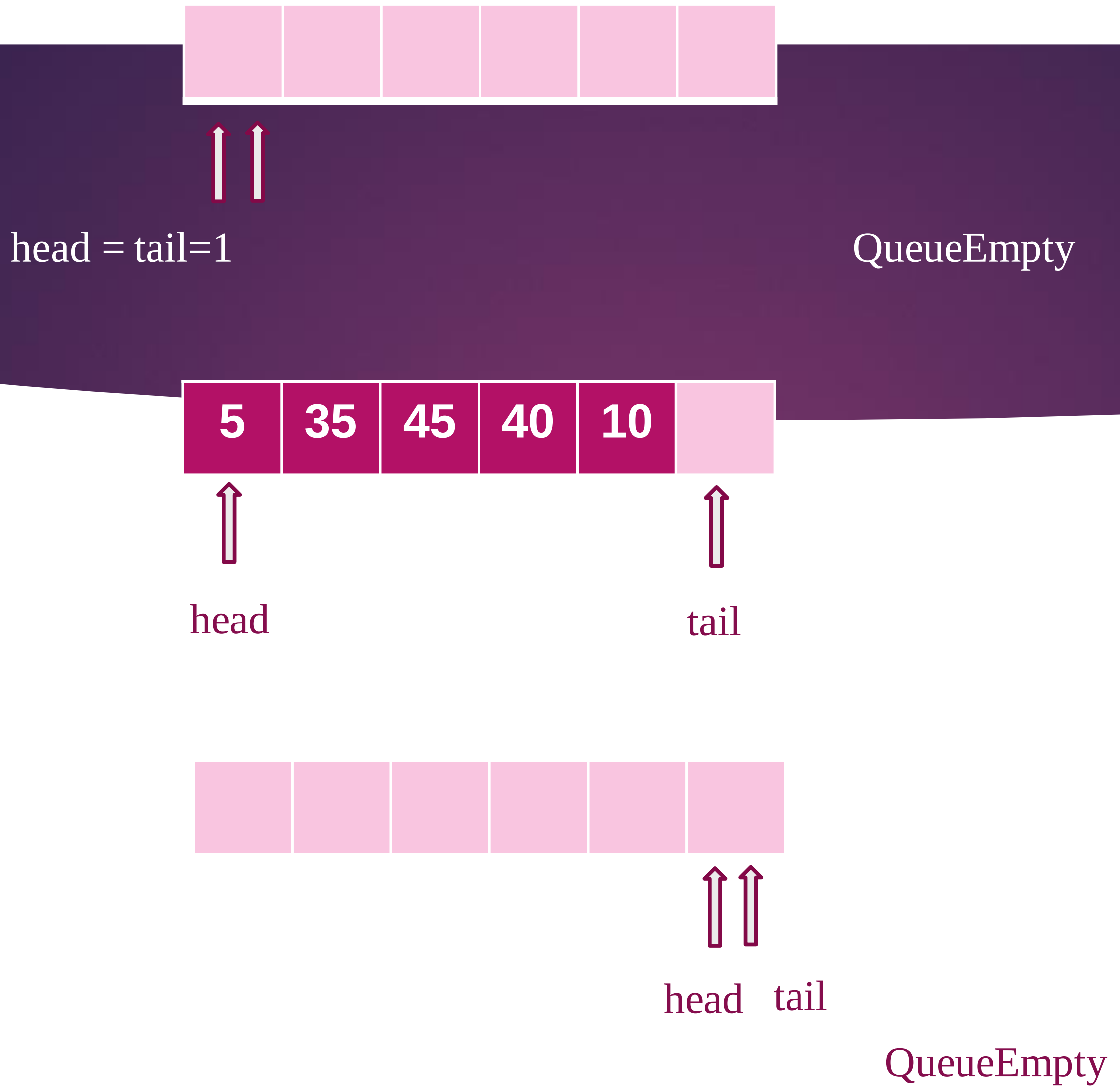


QueueFull

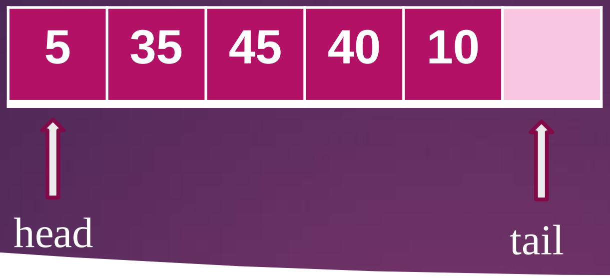
DeQueue(Q)

EnQueue (Q, 15)

QueueFull



QueueEmpty :   Q.head = Q.tail



QueueFull

QueueFull

Think about how do we write QueueFull : **Q.head = (….. Q.tail….) ?**

Example

n=6, Q is initially empty

ENQUEUE (Q, 8)

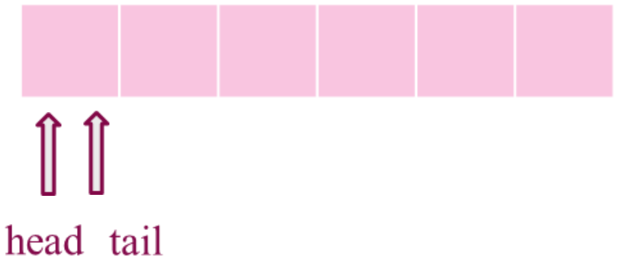
ENQUEUE (Q, 2)

ENQUEUE (Q, 3)

DEQUEUE(Q)

ENQUEUE (Q, 16)

DEQUEUE(Q)



Queue -Operations

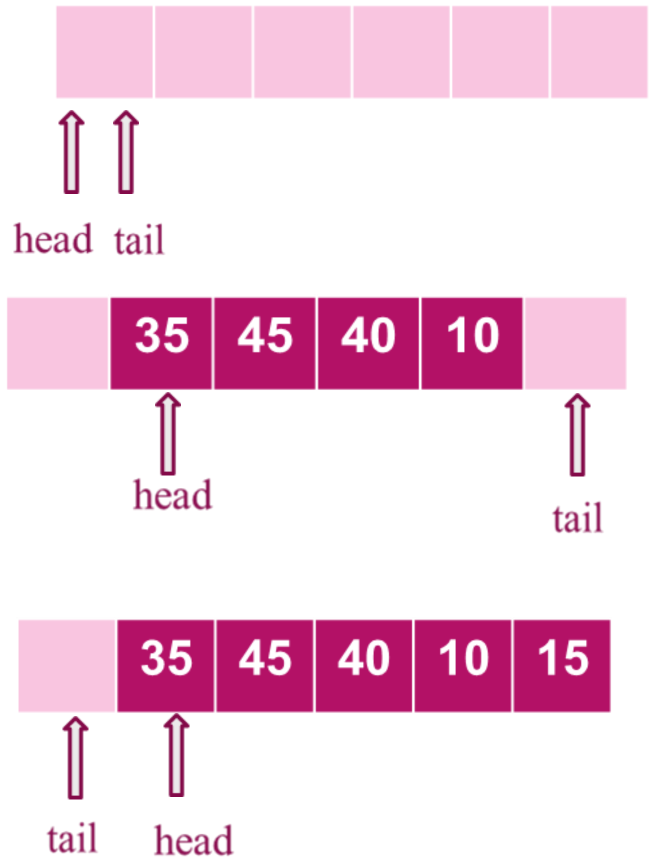
ENQUEUE (Q, x) //CLRS: ignoring oveflow

Q[Q.tail]=x

if Q.tail == Q.length

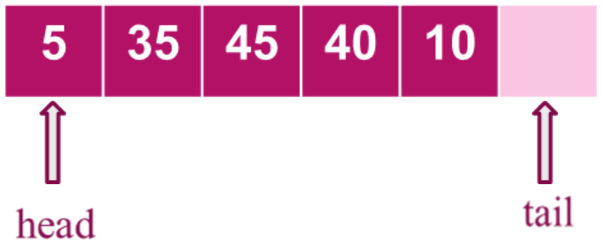
Q.tail =1

else Q.tail = Q.tail+1



Queue -Operations

```
DEQUEUE (Q)   //CLRS: ignoring underflow
    x = Q[Q. head]
    if (Q.head == Q.length)
        Q.head=1
    else Q.head = Q.head+1
    return x
```



Overflow/Underflow

- Write Algorithms for
  - QUEUE-EMPTY()
  - QUEUE-FULL()

Stack, Queue – Pointer based implementation

Stack - Basics

- List of elements, INSERT / DELETE only at one end of the list
  - Access restriction
  - Last-in, First-out (LIFO)
  - The last inserted element is the first one to be removed

Stack – Pointer Based Implementation

- **As a linked list of nodes**
- **Top – pointer to the front node**
- **PUSH() – Insert to front of the list**
- **POP()- Delete the front node**

Stack – Pointer Based Implementation

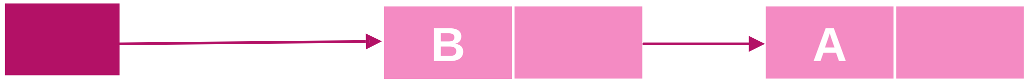
- Stack S is a linked list
- Attribute S.top
  - points to the top element (node at the front of the list)
- PUSH(S, x)
  - Insert node x as the new top node
- POP(S)
  - Removes the node pointed to by S.top

S.top

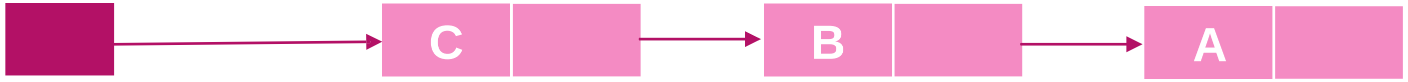
NIL

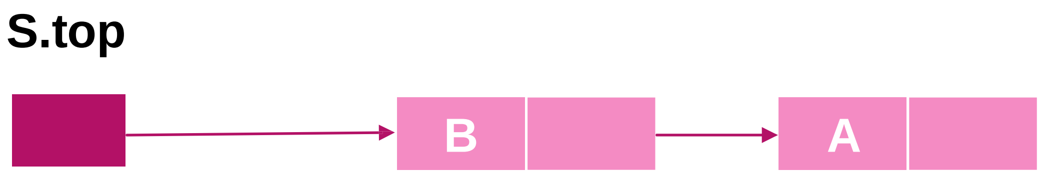
PUSH (S, ‘A’)

PUSH (S, ‘B’)

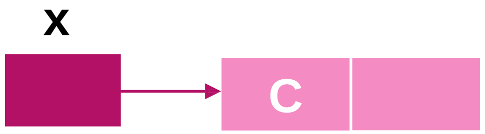


PUSH (S, ‘C’)



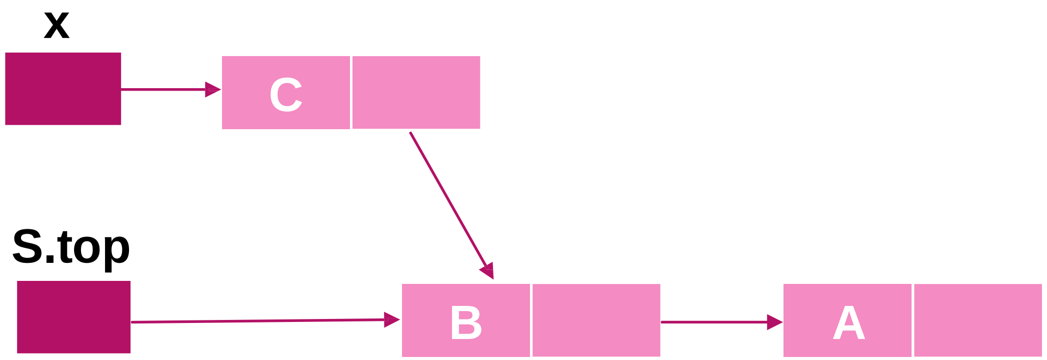


PUSH (S, 'C')

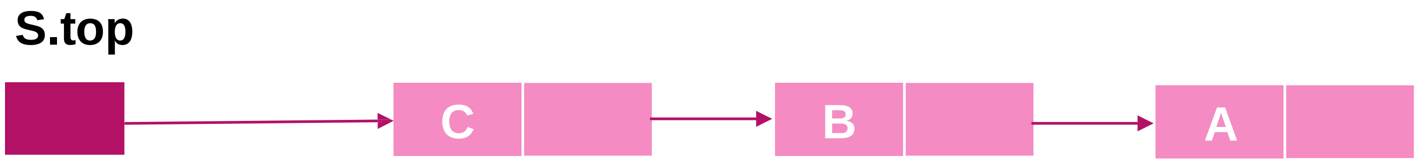


Create new node x

x.next = S.top



S.Top = x



Stack - Pointer Based Implementation

PUSH (S, x)

x. next = S.top

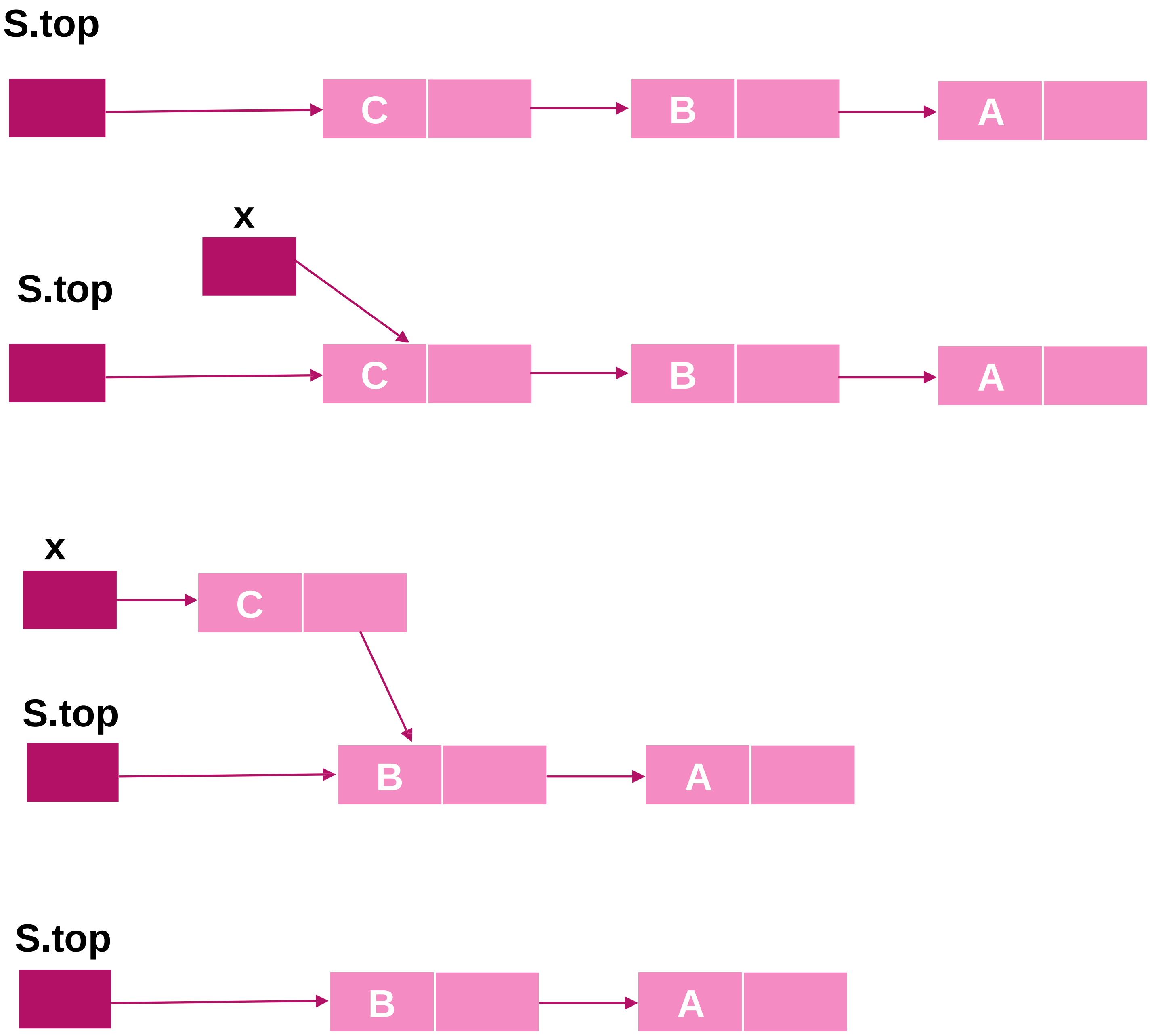
S.top = x

Think about what has to be done, If S.top is NIL

POP (S)

x = S.top

S.top= S.top.next



Stack - Pointer Based Implementation

```
  POP (S)
if (STACK-EMPTY(S))
     error "underflow"
x = S.top
S.top = S.top.next
return x;
```

Stack - Pointer Based Implementation

```
STACK-EMPTY (S)
    if S.top == NIL
        return true
    return false
```

Stack - Pointer Based Implementation

- **Time Complexity of operations**
  - **PUSH ?**
  - **POP ?**
  - **STACKEMPTY() ?**

Stack - Pointer Based Implementation

- **Time Complexity of operations**
  - **PUSH - O(1)**
  - **POP - O(1)**
  - **STACKEMPTY() – O(1)**

Implementation Details

```
struct node {      //a node in the stack
        ElemType elem;
        struct node *next
        };

struct stack {
         struct node * top;
        };
```

Queue - Basics

- **Queue - List with access restrictions**
  - **FIFO – First-In First-Out**
  - **Double ended- Head and Tail (front and rear)**
  - **Insertion always to the tail**
  - **Deletion always from the head**

Queue – Pointer Based Implementation

- As a linked list of nodes
- head – pointer to the first node (at the front end)
- tail – pointer to the last node (at the rear end)
- ENQUEUE() – insert to end of the list
- DEQUEUE()- delete the first node

Queue – Pointer Based Implementation

- **Queue Q as a linked list**
- **Attribute Q.head**
  - **points to the first element (node at the front of the list)**
- **Attribute Q.tail**
  - **points to the last element (node at the rear of the list)**
- **ENQUEUE(Q, x)**
  - **Insert node x at the tail**
- **DEQUEUE(Q)**
  - **Removes the node at the head**

Queue - Pointer Based Implementation

```
 ENQUEUE (Q, x)
   if (QUEUE-EMPTY (Q))
       Q.front = x
else
       Q.tail.next = x
Q.tail = x
```

Queue - Pointer Based Implementation

```
DEQUEUE (Q)
    if (QUEUE-EMPTY (Q))
        error "underflow"
    else
        x = Q. head
        if (Q.head == Q.tail)
            Q.tail = NIL
        Q.head = Q.head.next
        return x;
```

Queue - Pointer Based Implementation

```
QUEUE-EMPTY (Q)
    if Q. head == NIL
        return true
    return false
```

Implementation Details

```
struct node {      //a node in the queue
      ElemType elem;
      struct node *next
      };

struct Queue {
       struct node * head;
       struct node * tail;
      };
```

Stack - Pointer Based Implementation

- **Time Complexity of operations**
  - **ENQUEUE ?**
  - **DEQUEUE ?**
  - **QUEUE-EMPTY() ?**

Stack - Pointer Based Implementation

- **Time Complexity of operations**
  - **ENQUEUE - O(1)**
  - **DEQUEUE - O(1)**
  - **QUEUE-EMPTY() – O(1)**

Reference

T H Cormen, C E Leiserson, R L Rivest, C Stein *Introduction to Algorithms,* 3rd ed., PHI, 2010