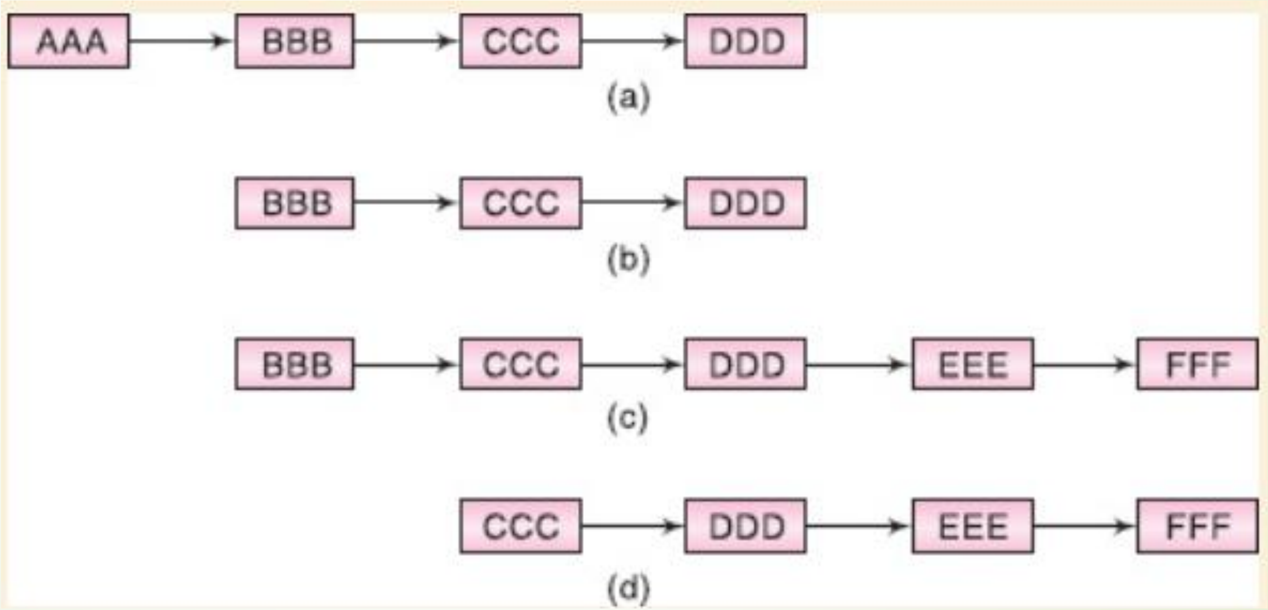
QUEUE

Queue

- A queue is a linear list of elements in which
  - deletion can take place only at one end called Front, and
  - Insertion takes place only at other end called Rear

Queue

- Queues are also known as First-In- First-Out (FIFO) list



Queue

- Queue are represented in two-ways
  - Linear Array
  - One-way Linked List

Array representation of Queue

- A queue is maintained by a
  - linear array QUEUE
  - Two pointer variable
    - FRONT : Containing the location of the front element of the queue
    - REAR : Containing
- FRONT == NULL indicates that the queue is empty

Queue

FRONT: 1
REAR: 4

AA	BB	CC	DD				...	
1	2	3	4	5	6	7		N

Delete an element

FRONT: 2
REAR: 4

	BB	CC	DD				...	
1	2	3	4	5	6	7		N

Whenever an element is deleted from the queue, the value of FRONT is increased by 1

FRONT = FRONT + 1

Queue

FRONT: 2

REAR: 4

	BB	CC	DD				...	
1	2	3	4	5	6	7		N

Insert an element

FRONT: 2

REAR: 5

	BB	CC	DD	EE			...	
1	2	3	4	5	6	7		N

Whenever an element is inserted into the queue, the value of REAR is increased by 1

REAR = REAR + 1

Queue

- REAR = N and Insert an element into queue

FRONT: 7
REAR: N

						XX	…	ZZ
1	2	3	4	5	6	7		N

Move the entire queue to the beginning of the array
Change the FRONT and REAR accordingly
Insert the element

This procedure is too expensive

Queue

- Queue is assumed to be circular
- QUEUE[1] comes after QUEUE[N]
- Instead of increasing REAR to N +1, we reset REAR = 1 and then assign

QUEUE[REAR] = ITEM

Queue

- FRONT = N and an element of QUEUE is Deleted

FRONT: N
REAR:

						XX	...	ZZ
1	2	3	4	5	6	7		N

We reset FRONT = 1, instead of increasing FRONT to N + 1

Queue

- QUEUE contain one element
  FRONT = REAR $\neq$ NULL

FRONT: 7
REAR: 7

						XX	...	
1	2	3	4	5	6	7		N

FRONT = NULL and REAR = NULL

Queue

			QUEUE 1	2	3	4	5
(a)	Initially empty:	FRONT: 0 REAR: 0					
(b)	A, B and then C inserted:	FRONT: 1 REAR: 3	A	B	C		
(c)	A deleted:	FRONT: 2 REAR: 3		B	C		
(d)	D and then E inserted:	FRONT: 2 REAR: 5		B	C	D	E
(e)	B and C deleted:	FRONT: 4 REAR: 5				D	E
(f)	F Inserted:	FRONT: 4 REAR: 1	F			D	E
(g)	D deleted:	FRONT: 5 REAR: 1	F				E
(h)	G and then H inserted:	FRONT: 5 REAR: 3	F	G	H		E
(i)	E deleted:	FRONT: 1 REAR: 3	F	G	H		
(j)	F deleted:	FRONT: 2 REAR: 3		G	H		
(k)	K inserted:	FRONT: 2 REAR: 4		G	H	K	
(l)	G and H deleted:	FRONT: 4 REAR: 4				K	
(m)	K deleted, QUEUE empty:	FRONT: 0 REAR: 0					

Queue: overflow

FRONT: 1
REAR: N

AA	BB	CC	DD	EE	FF	XX	...	ZZ
1	2	3	4	5	6	7		N

FRONT = 1 and REAR = N

FRONT: 7
REAR: 6

AA	BB	CC	DD	EE	FF	XX	...	ZZ
1	2	3	4	5	6	7		N

FRONT = REAR + 1

Algorithm to Insert in Q

1 If FRONT = 1 **and** REAR = N **or** if FRONT = REAR + 1 then Print: Overflow and Exit

2 If FRONT = NULL **then**

Set FRONT = 1 and REAR = 1

Else If REAR = N then

Set REAR = 1

Else

Set REAR = REAR + 1

3 Set QUEUE[REAR] = ITEM

4 Exit

Algorithm to Delete from Q

[1] If FRONT = NULL then Print: Underflow and Exit

[2] Set ITEM = QUEUE[FRONT]

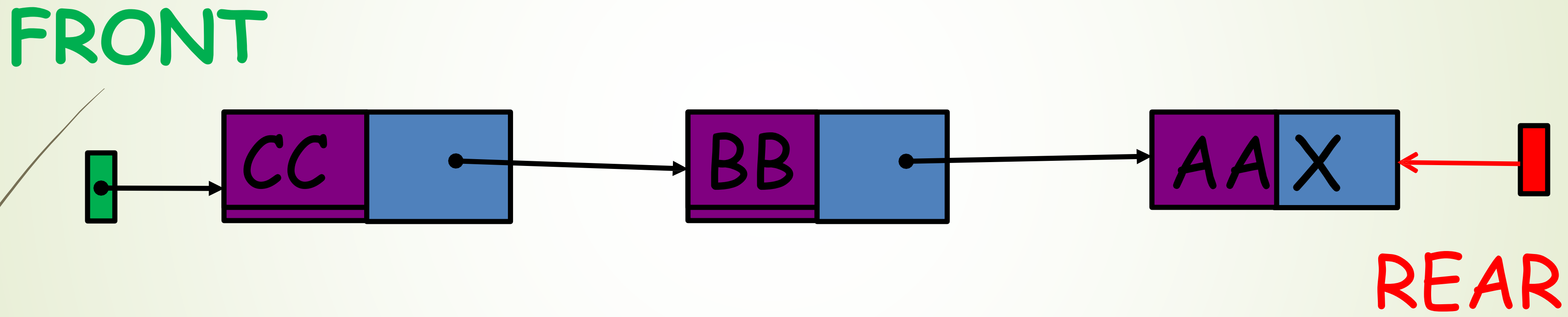
3 If FRONT = REAR **then**
Set FRONT = NULL and REAR = NULL
Else If FRONT = N then
Set FRONT = 1
Else
Set FRONT = FRONT + 1

4 Exit

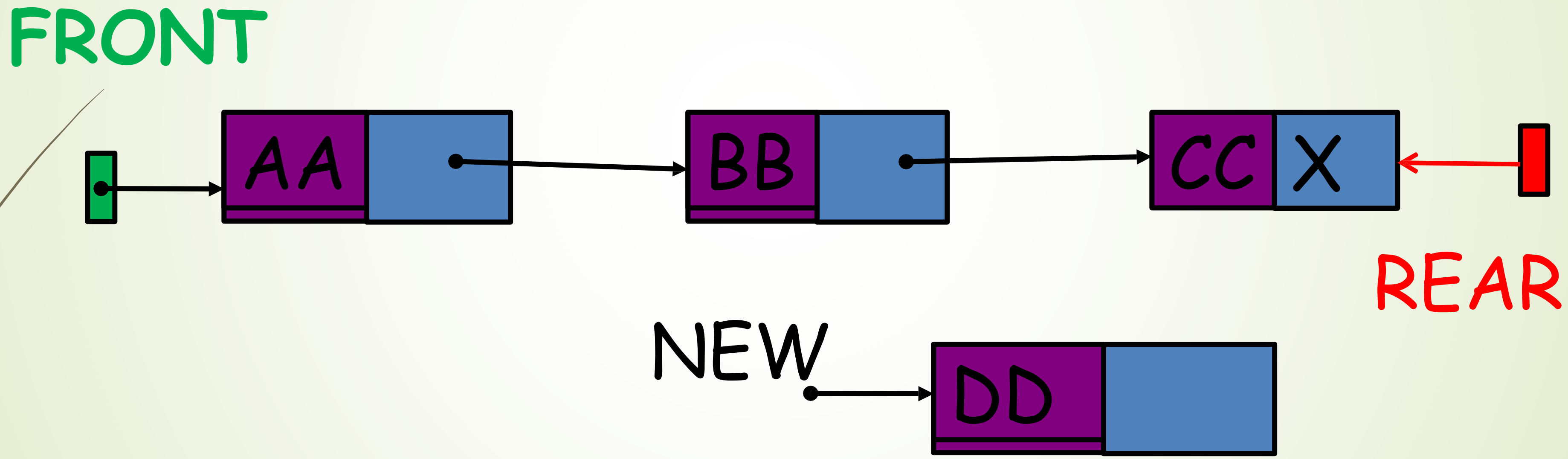
Linked List Representation of Queue

- A linked queue is a queue implemented as linked list with two pointer variable FRONT and REAR pointing to the nodes which is in the FRONT and REAR of the queue

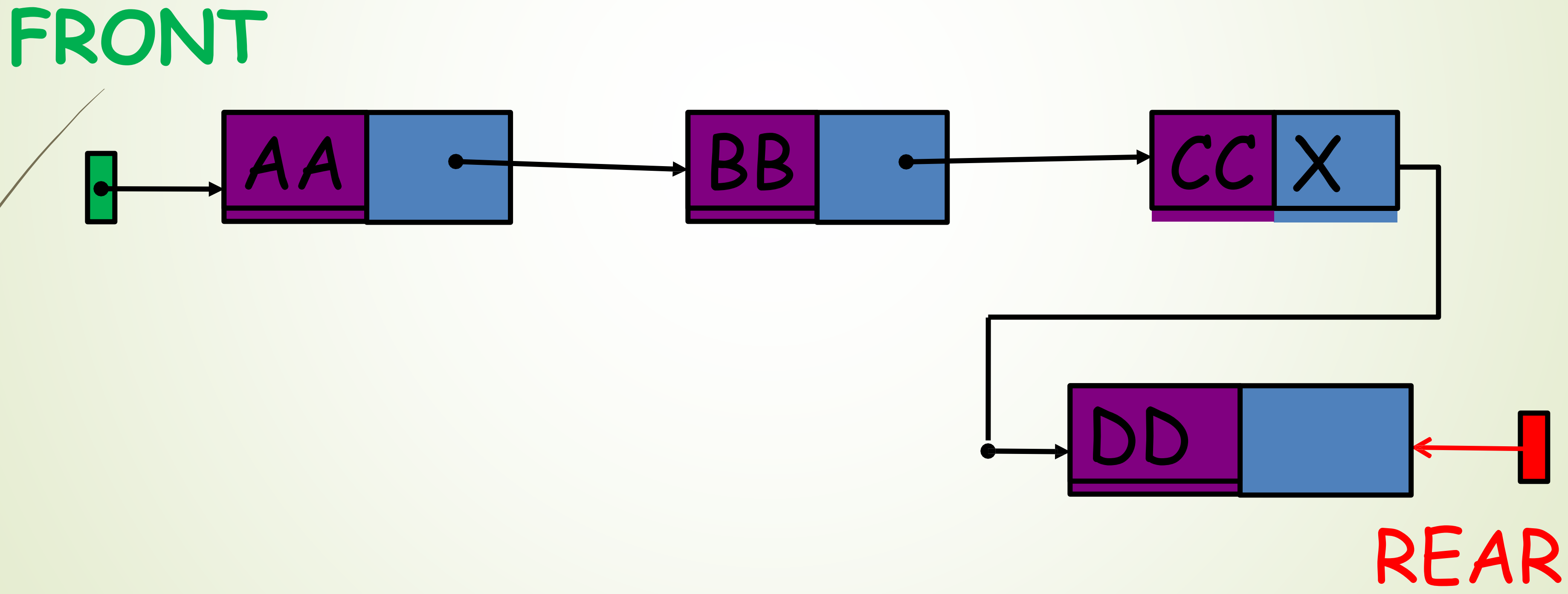
Linked List Representation of Queue



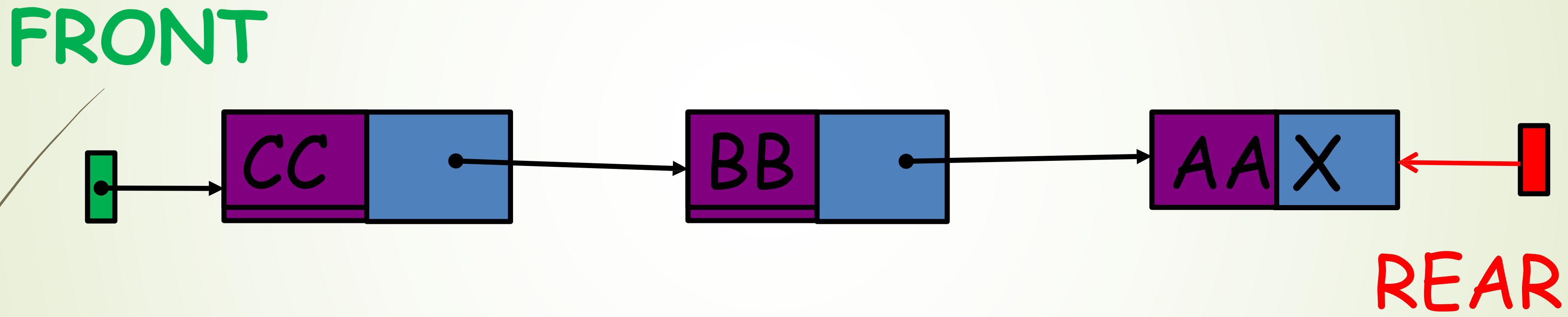
Insertion in a Queue



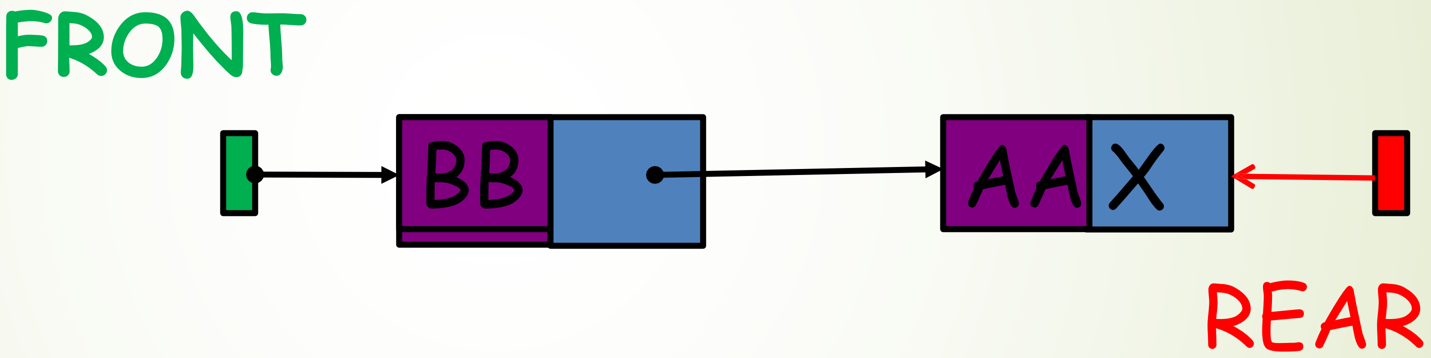
Insertion in a Queue



Delete from a Queue



Delete from a Queue



Linked Queue

- No need to check for overflow condition during insertion
- No need to view it as circular for efficient management of space

Insertion

1 NEW -> INFO = ITEM

NEW -> LINK = NULL

2 If (FRONT = NULL) then

FRONT = REAR = NULL

else

Set REAR -> LINK = NEW

REAR = NEW

3 Exit

Insertion

LINKQ_INSERT(INFO,LINK, FRONT, REAR,AVAIL,ITEM)

1. [Available space?] If AVAIL = NULL, then Write
   OVERFLOW and Exit
2. [Remove first node from AVAIL list]
   Set NEW := AVAIL and AVAIL := LINK[AVAIL]
3. Set INFO[NEW] := ITEM and LINK[NEW]=NULL
   [Copies ITEM into new node]
4. If (FRONT = NULL) then FRONT = REAR = NEW
   [If Q is empty then ITEM is the first element in the queue Q]
   else set LINK[REAR] := NEW and REAR = NEW
   [REAR points to the new node appended to the end of the list]
5. Exit.

Deletion

1 If (FRONT = NULL) then
Print: Underflow, and Exit

2 FRONT = FRONT -> LINK

3 Exit

Deletion

LINKQ_DELETE (INFO, LINK, FRONT, REAR, AVAIL, ITEM)

1. [Linked queue empty?] if (FRONT = NULL) then Write: UNDERFLOW and Exit
2. Set TEMP = FRONT [If linked queue is nonempty, remember FRONT in a temporary variable TEMP]
3. ITEM = INFO (TEMP)
4. FRONT = LINK (TEMP) [Reset FRONT to point to the next element in the queue]
5. LINK(TEMP) =AVAIL and AVAIL=TEMP [return the deleted node TEMP to the AVAIL list]
6. Exit.

Deque

- A deque is a linear list in which elements can be added or removed at either end but not in the middle

- Deque is implemented by a circular array DEQUE with pointers **LEFT** and **RIGHT** which points to the two end of the deque

Deque

- LEFT = NULL indicate deque is empty

LEFT: 4
RIGHT: 7

			AA	BB	CC	DD	
1	2	3	4	5	6	7	8

LEFT: 7
RIGHT: 2

YY	ZZ					WW	XX
1	2	3	4	5	6	7	8

Variation of deque

- There are two variation of deque

1 Input-restricted queue: Deque which allows insertions at only one end of the list but allows deletion at both ends of the list

2 Output-restricted queue: Deque which allows deletion at only one end of the list but allows insertion at both ends of the list

Deque

LEFT: 2
RIGHT: 4

	A	C	D		
1	2	3	4	5	6

F is added to the right

LEFT: 2
RIGHT: 5

	A	C	D	F	
1	2	3	4	5	6

Deque

LEFT: 2
RIGHT: 5

	A	C	D	F	
1	2	3	4	5	6

Two Letters on right is deleted

LEFT: 2
RIGHT: 3

	A	C			
1	2	3	4	5	6

Deque

LEFT: 2
RIGHT: 3

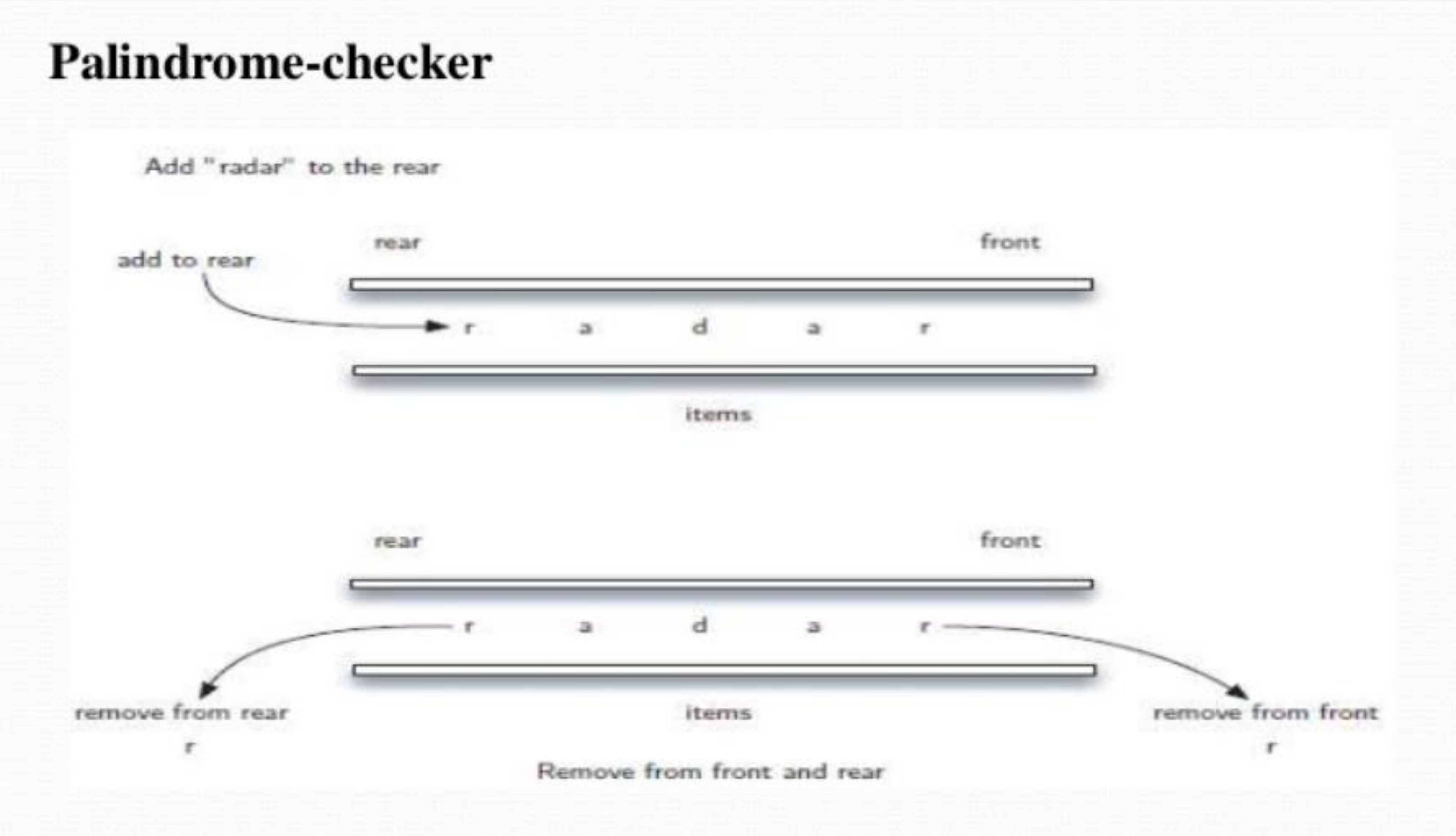
	A	C			
1	2	3	4	5	6

K, L and M are added to the Left

LEFT: 5
RIGHT: 3

K	A	C		M	L
1	2	3	4	5	6

Applications of deque



Applications of deque

A-Steal job scheduling algorithm

- The A-Steal algorithm implements task scheduling for several processors(multiprocessor scheduling).
- The processor gets the first element from the deque.
- When one of the processor completes execution of its own threads it can steal a thread from another processor.
- It gets the last element from the deque of another processor and executes it.

Priority Queue

- A priority queue is a collection of elements such that each elements has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules:

1 Elements of higher priority is processed before any elements of lower priority

2 Two elements with the same priority are processed according to the order in which they were added to the queue

Two types

- Ascending priority queue
  - only lowest priority element can be removed
- Descending priority queue
  - only highest priority element can be removed

Priority

- Intrinsic: based on one or several fields
  - Integer or alphabetical order
  - Ex: telephone directory ordered by last name
- External priority
  - An external value defines the priority of the element
- If time of insertion is used as priority of element
  - A stack may be viewed as descending priority queue whose element are ordered by time of insertion
  - A queue may be viewed as ascending priority queue whose element are ordered by time of insertion

Representation of Priority Queue

1 One-way List
   a) Ordered
   b) unordered

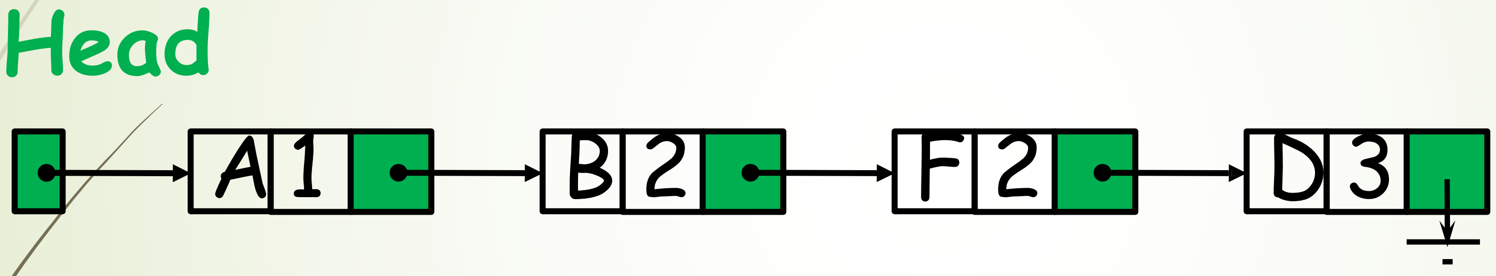
2 Array
   a) Single
      i. Ordered
      ii. unordered
   b) Multiple

One-Way (ordered) List Representation of a Priority Queue

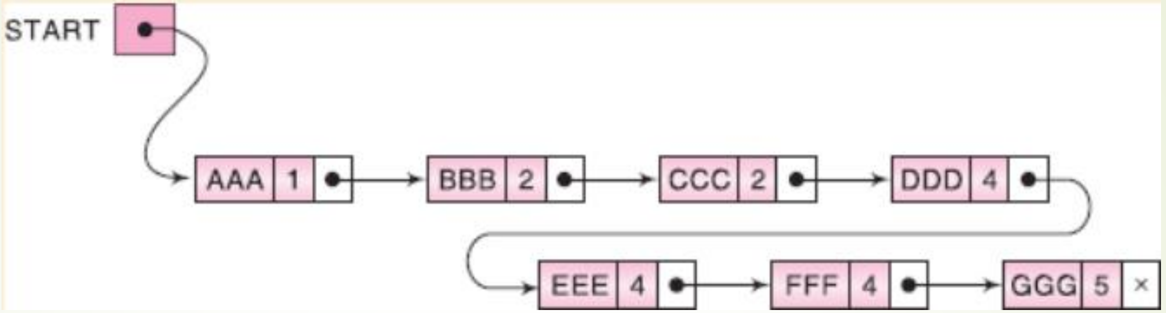
1 Each node in the list will contain three items of information: **an information field INFO, a priority number PRN, and a link number LINK**

2 A node X precedes a node Y in the list
(a) when X has **higher priority** than Y or
(b)when both have same priority but X was **added to the list before** Y

Priority Queue



Priority Queue



START 5

AVAIL 2

	INFO	PRN	LINK
1	BBB	2	6
2			7
3	DDD	4	4
4	EEE	4	9
5	AAA	1	1
6	CCC	2	3
7			10
8	GGG	5	0
9	FFF	4	8
10			11
11			12
12			0

Insertion and Deletion

- Deletion : Delete the first node in the list.

- Insertion: Find the location of Insertion
and add an ITEM with priority number N

a. Traverse the list until finding a node X whose priority exceeds N. Insert ITEM in front of node X

b. If no such node is found, insert ITEM as the last element of the list

(Multiple) Array representation of Priority Queue

- Multiple arrays are used and each array is maintained as circular queue
- Separate queue for each level of priority
- Each queue will appear in its own circular array and must have its own pair of pointers, FRONT and REAR
- If each queue is given the same amount space then a 2D queue can be used

FRONT REAR

1	2	2
2	1	3
3	0	0
4	5	1
5	4	4

QUEUE

	1	2	3	4	5
1		AA			
2	BB	CC	DD		
3					
4	FF			DD	EE
5			GG		

Deletion Algorithm for Ascending Queue

1 Find the smallest K such that FRONT[K] $\neq$ NULL

2 Delete and process the front element in row K of QUEUE

3 Exit

Insertion Algorithm

Insert a new element ITEM with priority M

1 Insert ITEM as the rear element in row M of QUEUE

2 Exit