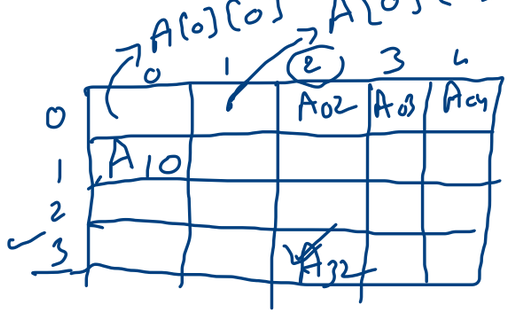
Unit-2:

Multidimensional Arrays:

2D Arrays

int a[4][5];

n rows x m columns.

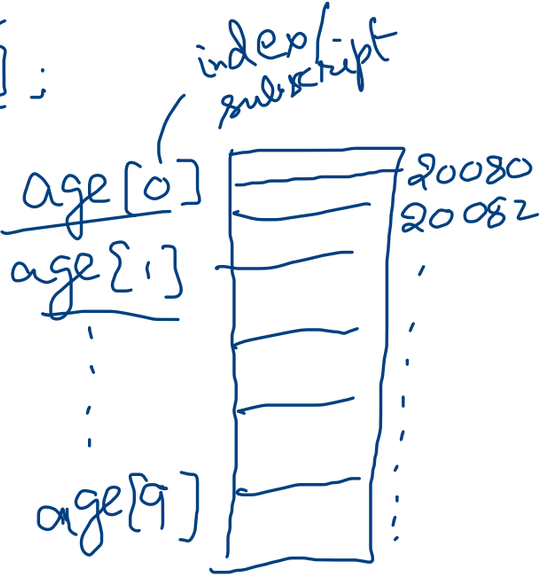


4 x 5 = 20

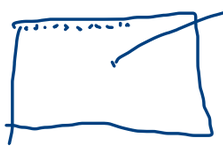
r x c

1D-Arrays.

int age[10];



2D Arrays



- Excel.
- Mathematical objects → Matrices
- Frame Buffer.

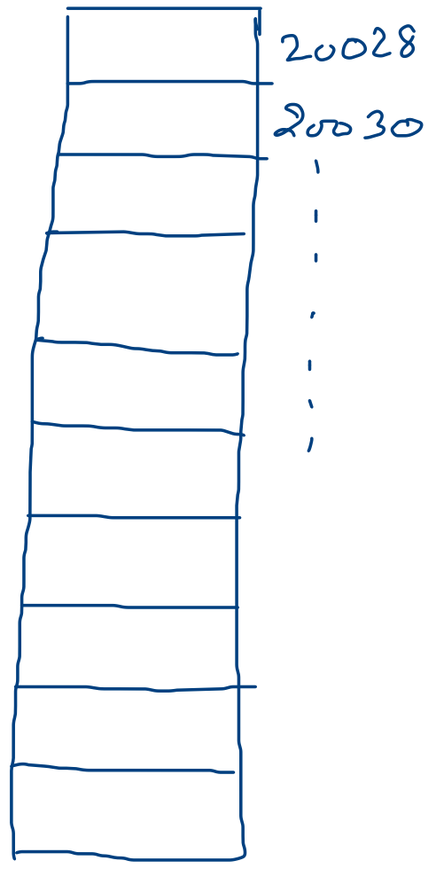
Pic Def.

Representing 2D Arrays in Memory:

- Row Major order.
- Column Major order.

2D

a[4][5]



Rep. of 2D Arrays.

(1) Row Major order:

A

	1	2	3	4
1	10	11	12	16
2	21	19	17	11
3	23	28	25	30

A_{11} A_{12} A_{33}

Logical representation

3X4 array A

$r = 3$

$c = 4$

Total elements = $3 \times 4 = 12$

used in various HLLs.

C, C++, Java, Pascal, Ada, Modula2, etc.

mapping

Row1

Row2

Row3

Physical representation

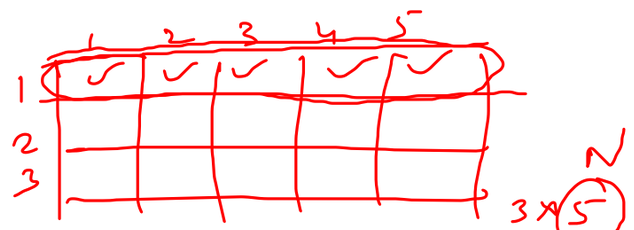
10	A[1,1]
11	A[1,2]
12	A[1,3]
16	A[1,4]
21	A[2,1]
19	A[2,2]
17	
11	
23	
28	
25	A[3,3]
30	A[3,4]

How to calculate the address of an element of a 2D-Array ? in case of Row-Major order?

Let there be an $M \times N$ array A (ie $A[M, N]$)

Let it's base address be Base(A)

s = Size of an element of the Array.



Q: To find address of specific element $A[i, j]$

multiply

(1) Address of i^{th} Row : ie How many rows to skip over ie $(i-1) \times$ Size of a row

$N \times s$

(2) Address of j^{th} Column: $(j-1) \times s$

$(i-1) \times N \times s$

$$\text{Address of } A[i,j] = \text{Base}(A) + (i-1) \times N \times s + (j-1) \times s$$

$$= \text{Base}(A) + s((i-1)N + (j-1))$$

Formula:

Address of $A[i,j] = \text{Base}(A) + s[(i-1)N + (j-1)]$

Example:

Address of $A[3,2]$ = ?

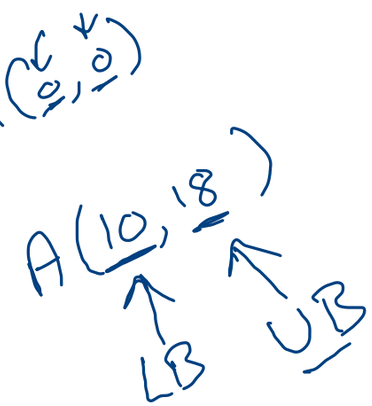
$= 200 + 2[(3-1)4 + (2-1)]$

$= 200 + 2[(2)4 + 1]$

$= 200 + 2(8+1)$

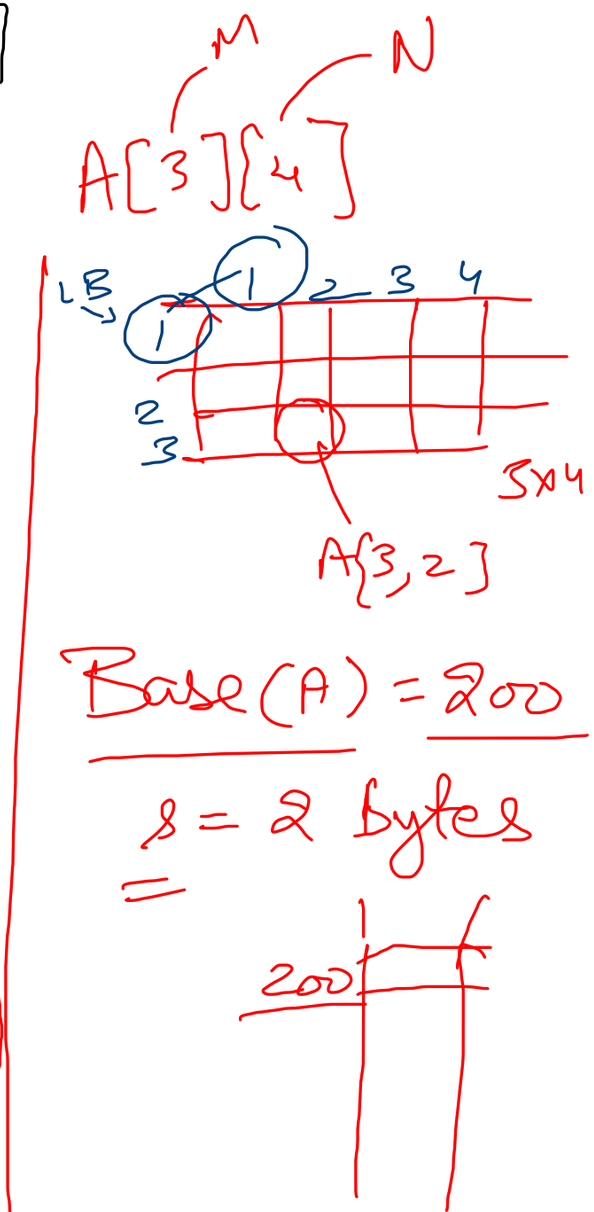
$= 200 + 18$

$= 218$



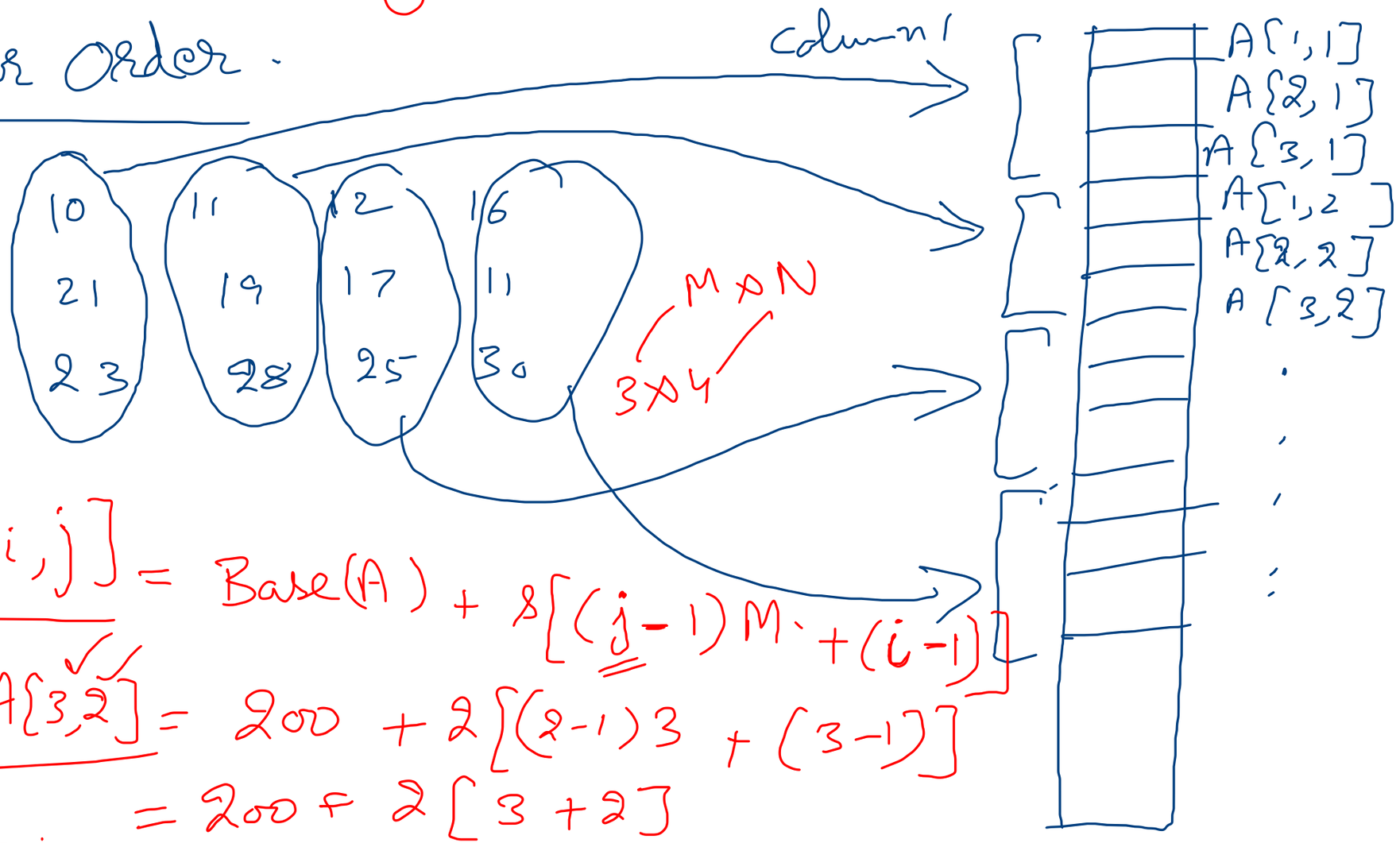
General Formula.

Address of $A[i,j] = \text{Base}(A) + s[(i-lb)N + (j-ub)]$



2nd Method of Representing 2D Arrays in Memory:

② Column Major Order.



Formula:

$$\text{Address of } A[i,j] = \text{Base}(A) + s\left[(j-1)M + (i-1)\right]$$

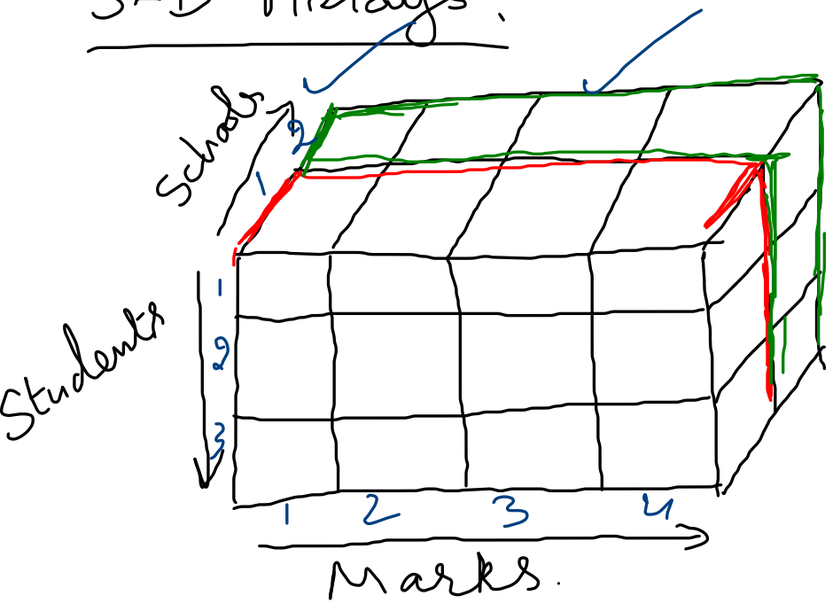
Eg: Address of $A[3,2]$ = $200 + 2[(2-1)3 + (3-1)]$

in column major order

$= 200 + 2[3+2]$

$= 210$

3-D Arrays:



Marks[2,3,4]

Marks → M

School 1	Marks[1,1,1]	M[1,1,2]	M[1,1,3]	M[1,1,4]
	M[1,2,1]	M[1,2,2]	M[1,2,3]	M[1,2,4]
	M[1,3,1]	M[1,3,2]	M[1,3,3]	M[1,3,4]

Student1

Student2

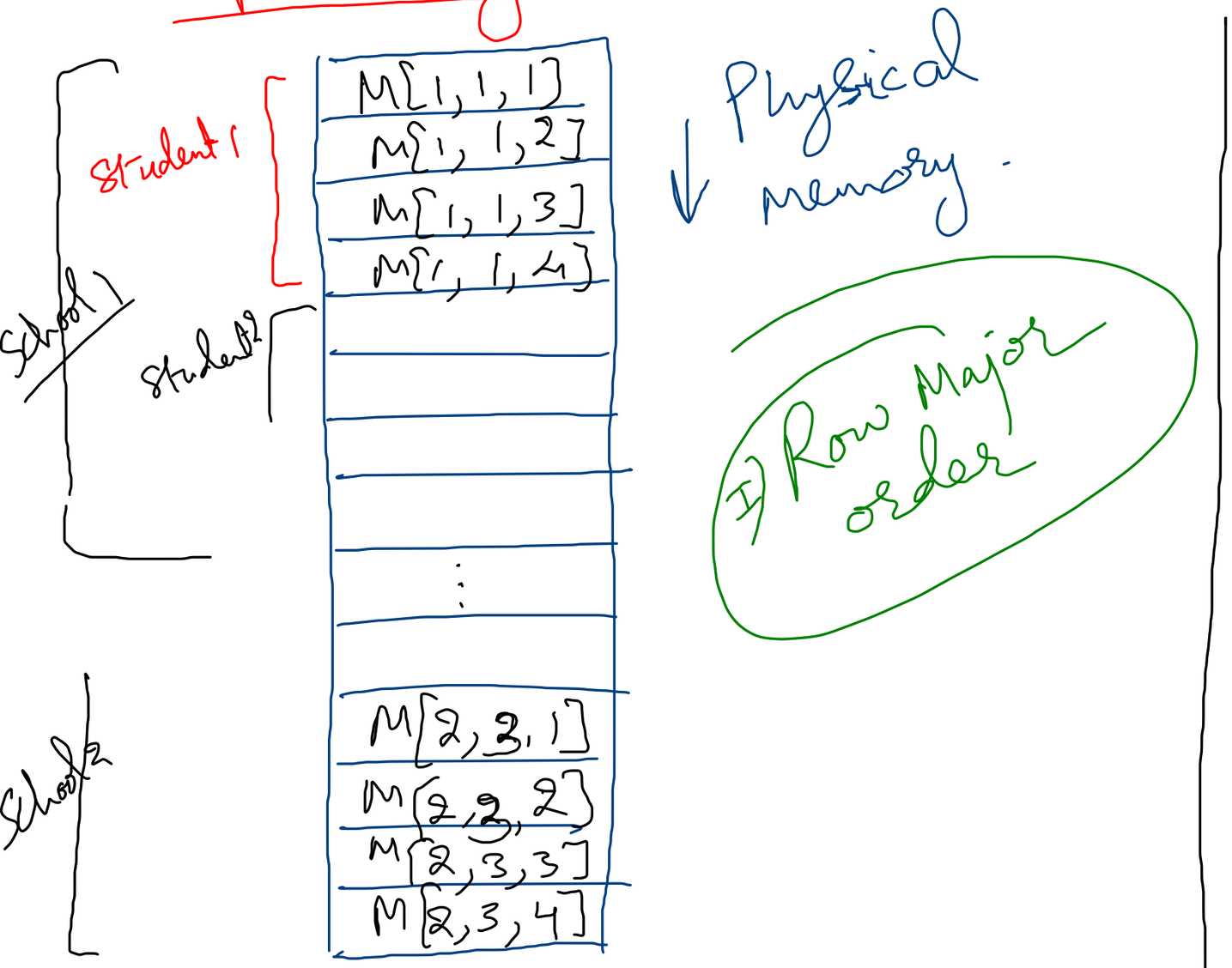
Student3

School 2	M[2,1,1]	M[2,1,2]	-	-
	-	-	-	-

complete it.

```
for(i=1; i<=M; i++)
  { for(j=1; j<=N; j++)
      { for(k=1; k<=Q, k++)
          { cin>> M[i][j][k];
          }
      }
  }
```

Representing 3D Arrays in memory:



Finding address of an element in 3D. Array.

$1 \le i \le M$, $1 \le j \le N$, $1 \le k \le Q$.

$(i-1)NQ + (j-1)Q$

Formula:

$$\text{Address of } A[i,j,k] = \text{Base}(A) + S[((i-1)N + (j-1))Q + (k-1)]$$

eg: $A[\underset{i}{2}, \underset{j}{2}, \underset{k}{3}]$

$B(A) = 200$

$S = 2$

$M = 2$

$N = 3$

$Q = 4$

$= 200 + 2[((2-1)3 + (2-1))4 + (3-1)]$

$= 200 + 2((3+1)4 + 2)$.

$= 200 + 2[16+2]$

$= 200 + 2(18) = 236$.

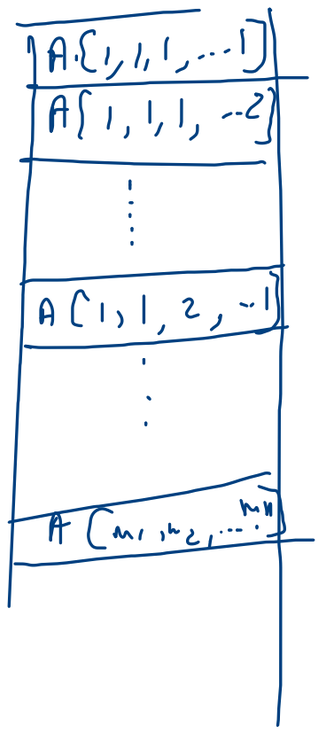
Generalization of Multidimensional Array:

↳ n-D array.

Range: $1 \le k_1 \le m_1, \; 1 \le k_2 \le m_2, \; \ldots \; 1 \le k_n \le m_n$.

Element: $A[k_1, k_2, \ldots k_n]$

Memory Rep. (Row Major order)



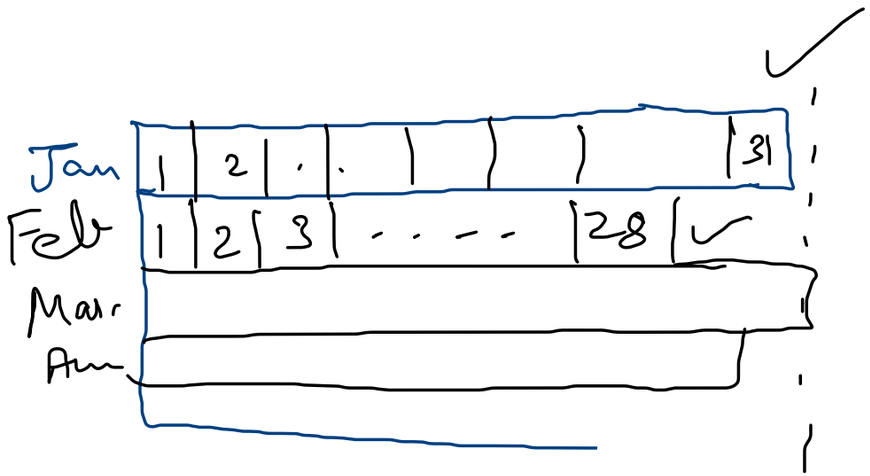
Formula for finding address of an element.

$$A[k_1, k_2, \ldots k_n] = \text{Base}(A) + \omega[(k_1-1)m_2m_3\ldots m_n + (k_2-1)m_3m_4\ldots m_n + \ldots + (k_{n-1}-1)m_n + (k_n-1)]$$

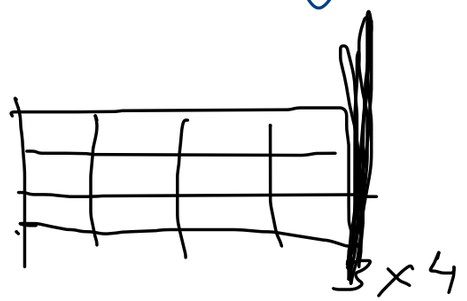
$$= \text{Base}(A) + \omega\left[\sum_{1 \le i \le n} p_i (k_i - 1)\right]$$

$$p_i = \prod_{i < x \le n} m_x$$

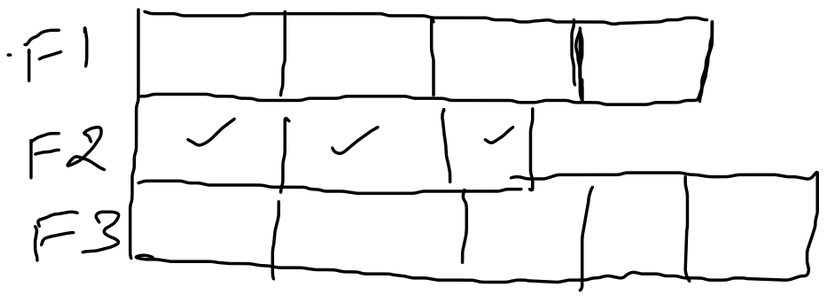
Jagged Arrays and Pointer Arrays:



eg:- calander for a particular year.



An array in which no. of elements in each row need not be of same size is called a jagged array.

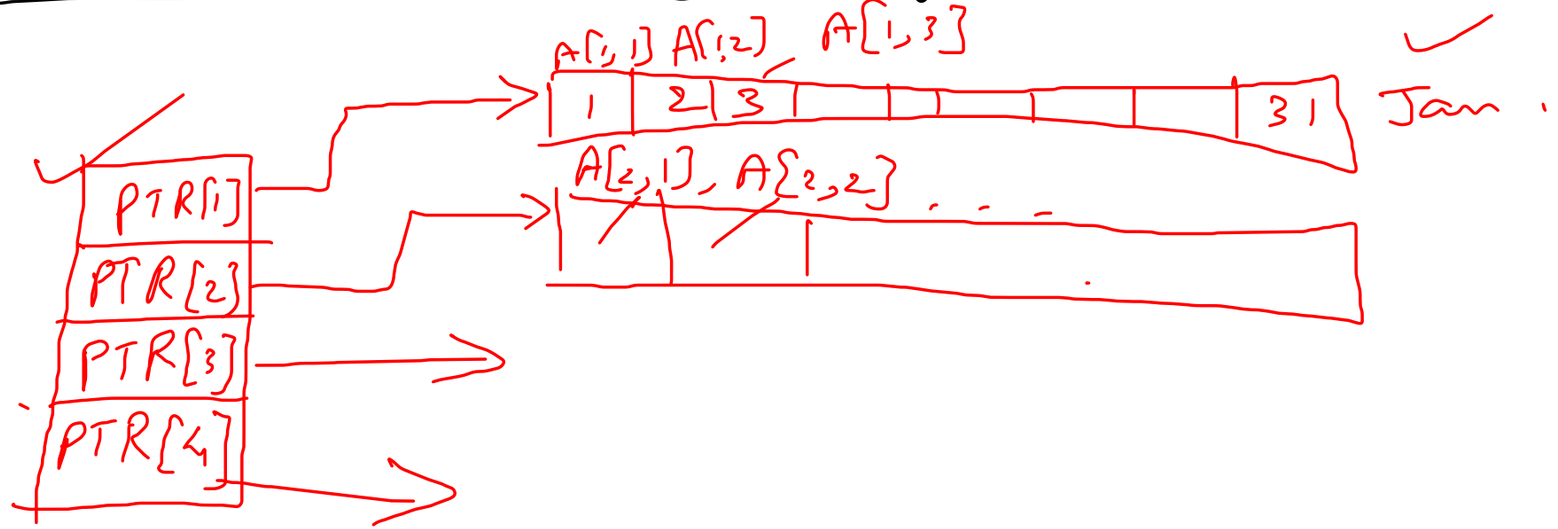


Storing Jagged arrays in Memory: (We must know abot: Pointer and array of pointers)

P is a var that points to an element of an array.

↳ contains address of an element of an array.

PTR is an array of pointers.



SPARSE MATRICES

1. Regular Sparse Matrices
   - i) Tridiagonal Matrix

$$\begin{bmatrix} 8 & 4 & 0 & 0 & 0 \\ 2 & -5 & 3 & 0 & 0 \\ 0 & 2 & 4 & 6 & 0 \\ 0 & 0 & -2 & 5 & 9 \\ 0 & 0 & 0 & 8 & 4 \end{bmatrix}$$

   - ii) Lower Triangular Matrix

$$\begin{bmatrix} 9 & 0 & 0 & 0 & 0 \\ 8 & -1 & 0 & 0 & 0 \\ -4 & 6 & 3 & 0 & 0 \\ 6 & 4 & 9 & 5 & 0 \\ 9 & 5 & 4 & 8 & 4 \end{bmatrix}$$

   - iii) Upper Triangular Matrix

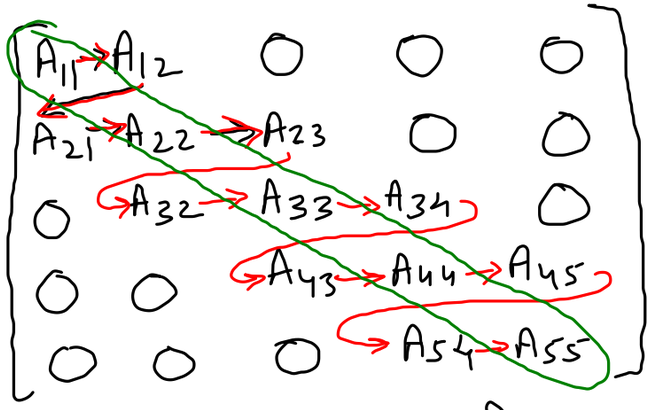
$$\begin{bmatrix} 1 & -2 & -8 & 4 & 6 \\ 0 & 3 & 2 & 9 & 5 \\ 0 & 0 & 5 & 1 & 3 \\ 0 & 0 & 0 & 7 & 4 \\ 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

2. Non-Regular / Irregular Sparse Matrices

$$\begin{bmatrix} 2 & 0 & 0 & 8 & 0 \\ -1 & 0 & 0 & 4 & 3 \\ 0 & 5 & 6 & 0 & 0 \\ 1 & 8 & 0 & 4 & 0 \\ 0 & 9 & 0 & 1 & 0 \end{bmatrix}$$

Storage Scheme for various Regular Sparse Matrices

(1) Storing Tridiagonal Matrix:



5 × 5

This may be stored in one dimensional array X as:

Row by Row

$X = \{A_{11}, A_{12}, A_{21}, A_{22}, A_{23}, A_{32}, A_{33}, A_{34}, A_{43}, A_{44}, A_{45}, A_{54}, A_{55}\}$

$X[1]$ $X[2]$ ..... $X[13]$

25 → 13

For $n \times n$ Tridiagonal Sparse matrix, we have following no. of non-zero elements:

3 Diagonals

Above middle ← Middle → Below middle.

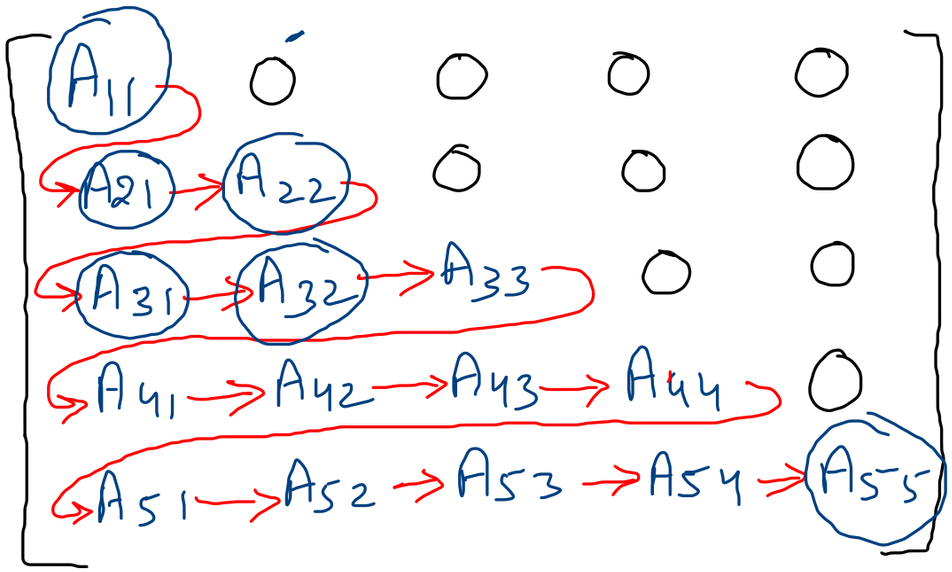
$n-1 + (n) + n-1$

$= 3n - 2.$

∴ In our example, no. of elements were

$= 3 \times 5 - 2 = 15 - 2 = 13$

(2) Storing Lower Triangular Matrix:



This may be stored in a one-Dimensional array Y as:

(Row-by-Row)

$Y = \{A_{11}, A_{21}, A_{22}, A_{31}, A_{32}, A_{33}, A_{41}, A_{42}, A_{43}, A_{44}, A_{51}, A_{52}, A_{53}, A_{54}, A_{55}\}$

$Y[1]$ $Y[2]$ ....

$Y[14]$ $Y[15]$

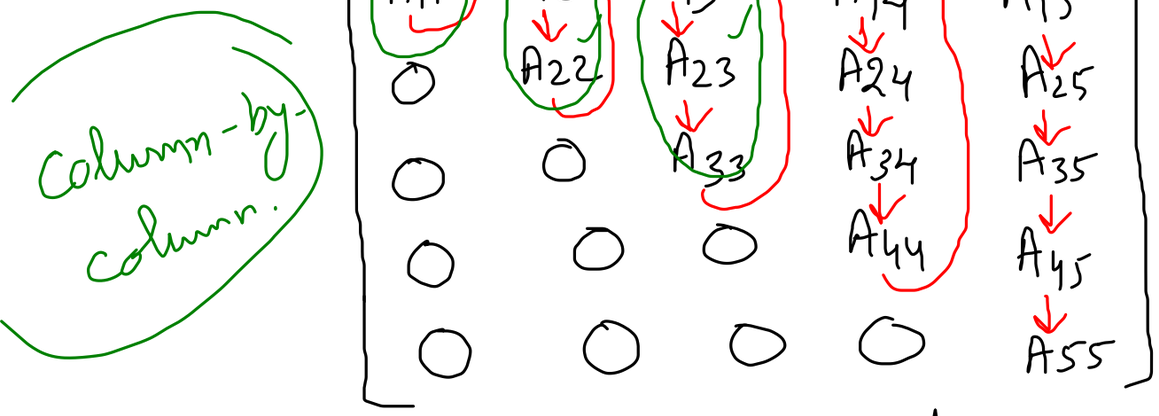
No. of elements in this case:

$$1+2+3+4+\ldots.(n-1)+n$$

$$= \frac{n(n+1)}{2}$$

$$= \frac{5(5+1)}{2} = \frac{5(6)}{2} = \frac{30}{2} = 15$$

(3) Storing Upper Triangular Matrix



This may be stored in one dimensional array Z as:

$Z = \{A_{11}, A_{12}, A_{22}, A_{13}, A_{23}, A_{33} \cdots A_{45}, A_{55}\}$

$Z[1]$ $Z[2]$ ... $Z[14]$ $Z[15]$

No. of elements in this case:

$= 1 + 2 + 3 + \cdots n$

$= \frac{n(n+1)}{2}$

In our example:

$\frac{5(5+1)}{2} = \frac{30}{2} = 15$

Storage Scheme for Irregular Sparse Matrices:

(1) Using storage by Index Method

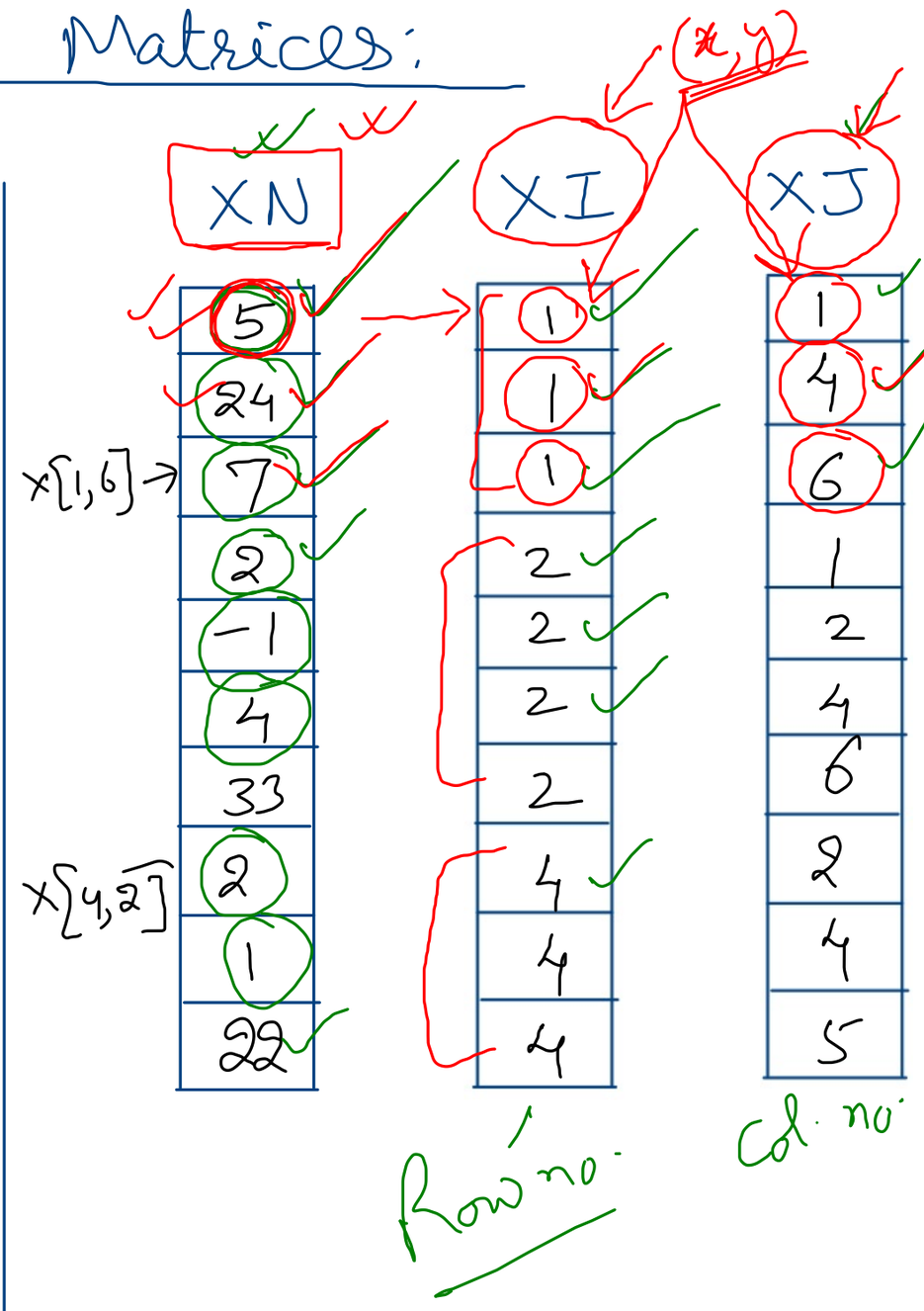
Take 3 one-Dim-arrays as XN, XI, XJ

- XN → Contains non-zero elements stored contigeously row by row
- XI → contains row no. of each non-zero element
- XJ → Contains column no. of each non-zero element

Example:

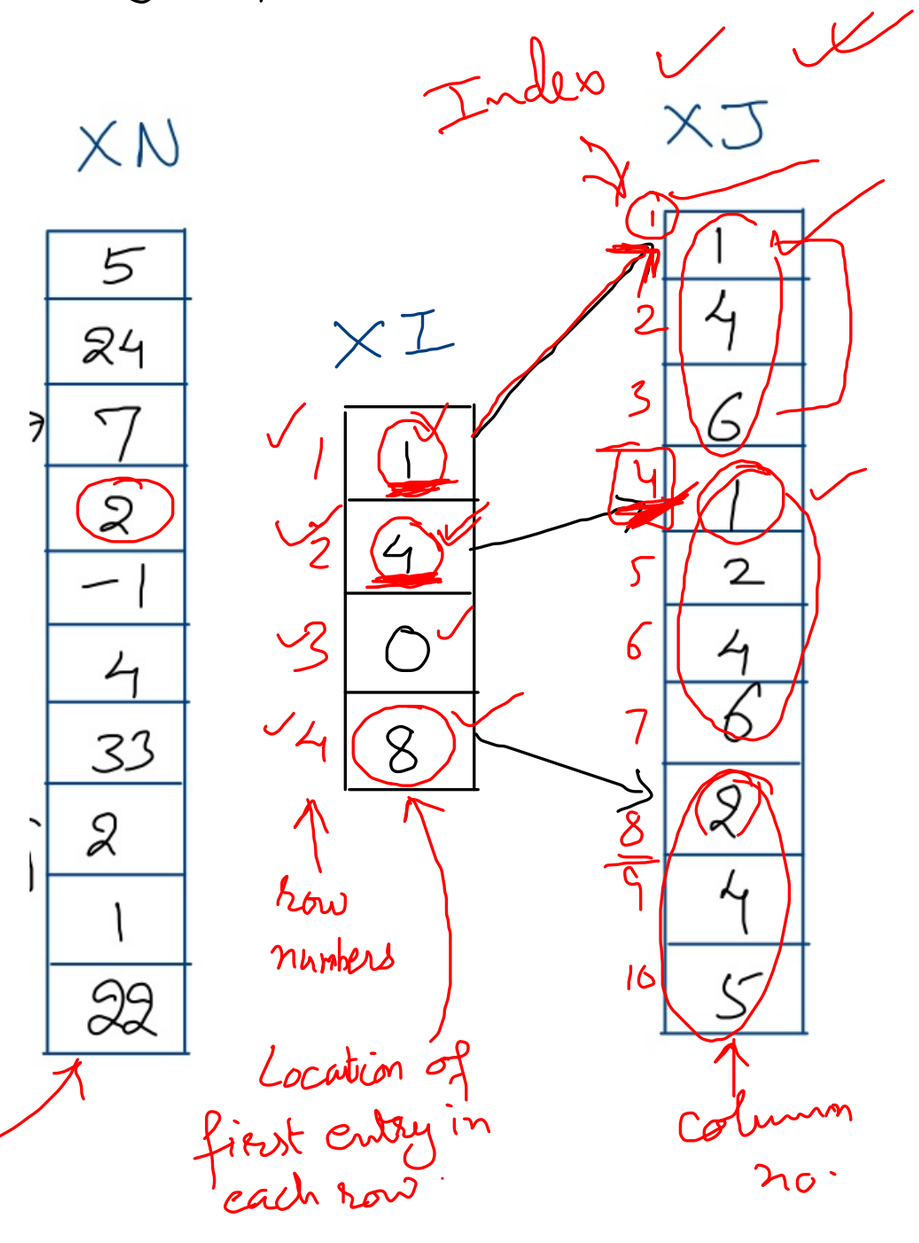
Consider a 4×6 sparse matrix X as follows:

$$X = \begin{bmatrix} 5 & 0 & 0 & 24 & 0 & 7 \\ 2 & -1 & 0 & 4 & 0 & 33 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 22 & 0 \end{bmatrix}$$



(2) Using Compressed row storage format :

$$\begin{bmatrix} 5 & 0 & 0 & 24 & 0 & 7 \\ 2 & -1 & 0 & 4 & 0 & 33 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 22 & 0 \end{bmatrix}$$

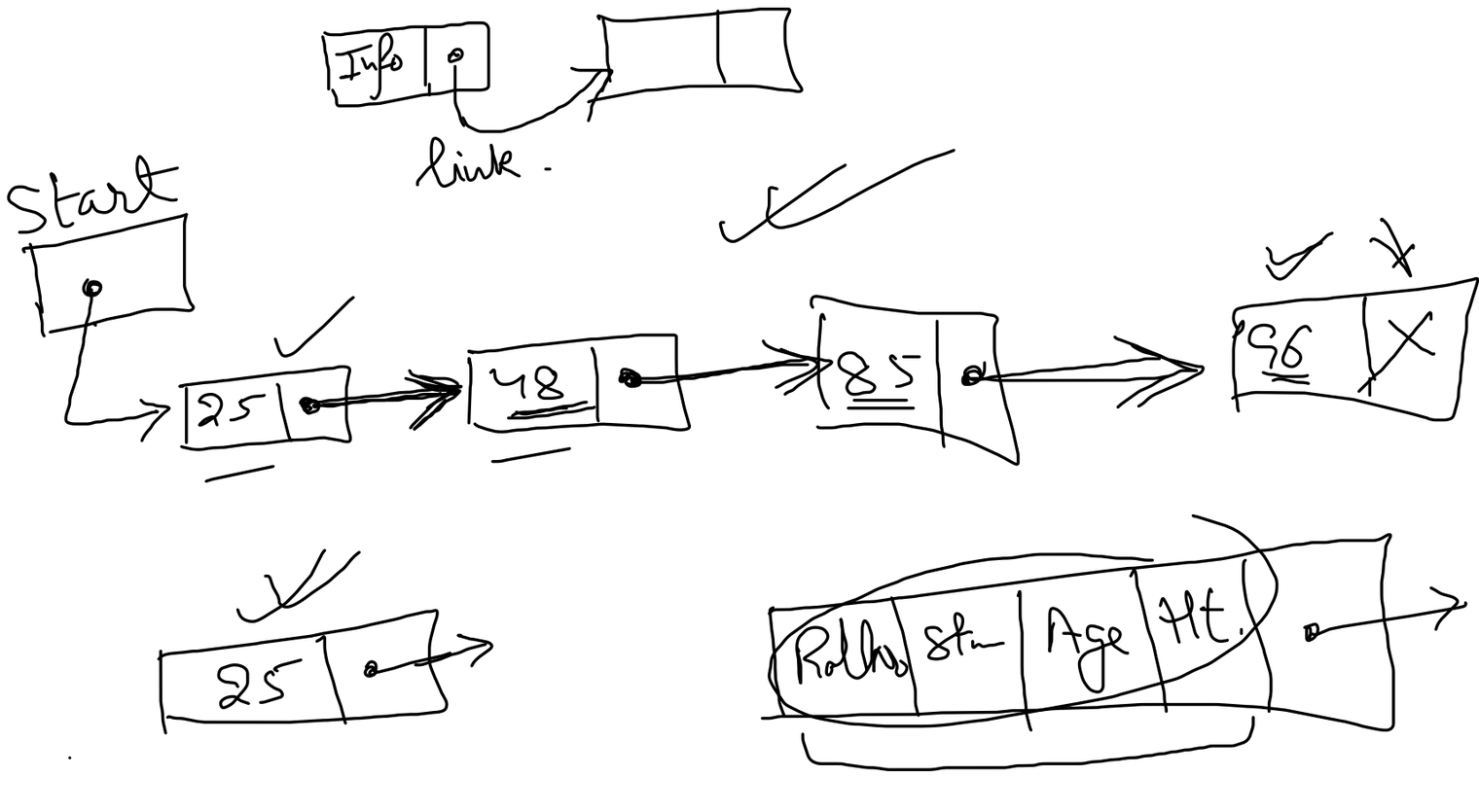


4 - 1 = 3

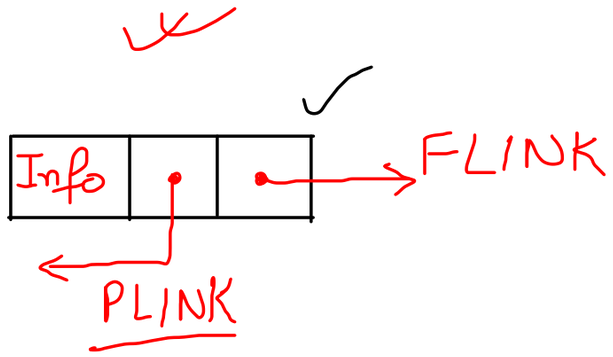
8 - 4 = 4

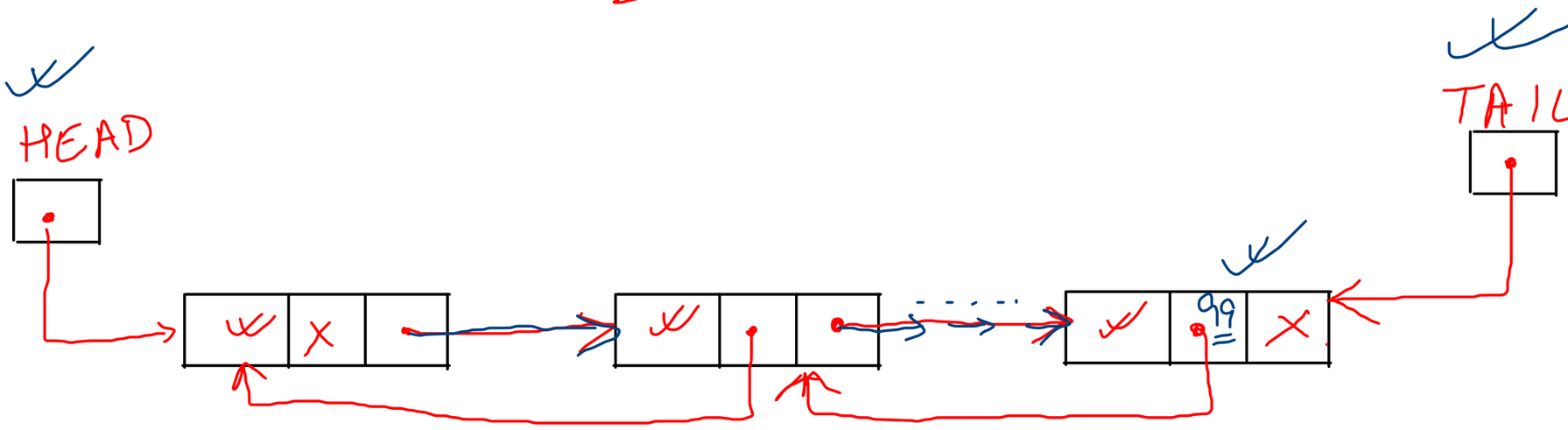
Linked Lists:

- Doubly Linked List
  - Doubly Header Linked Lists
  - Doubly Linked Circular Lists
  - Doubly Circular Header Linked Lists
  - Operations on Doubly Linked Lists
- Multi Linked Lists



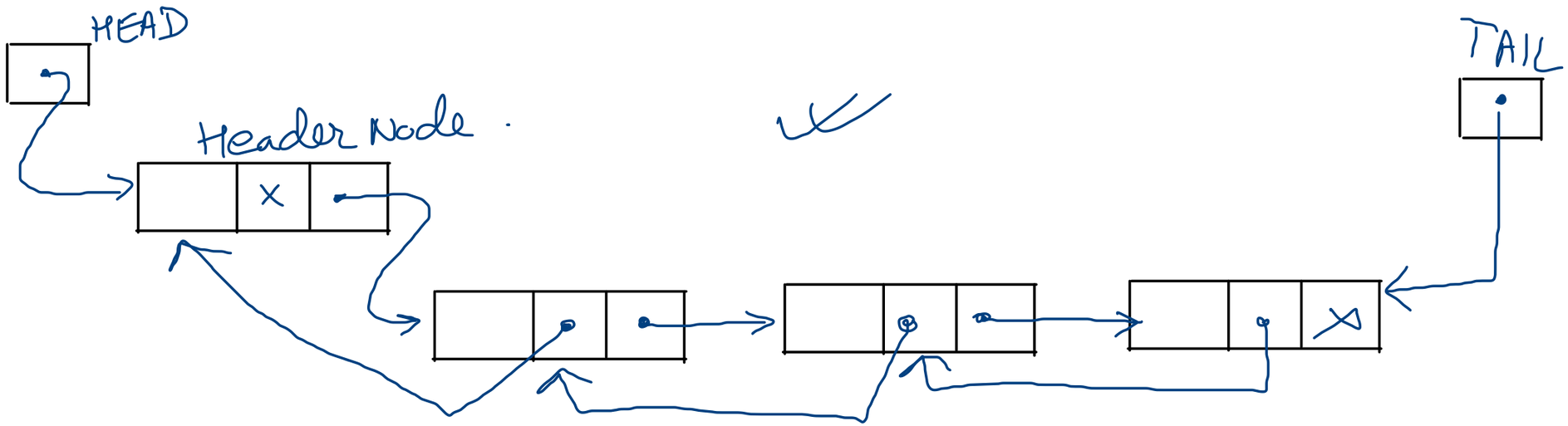
Doubly Linked List -

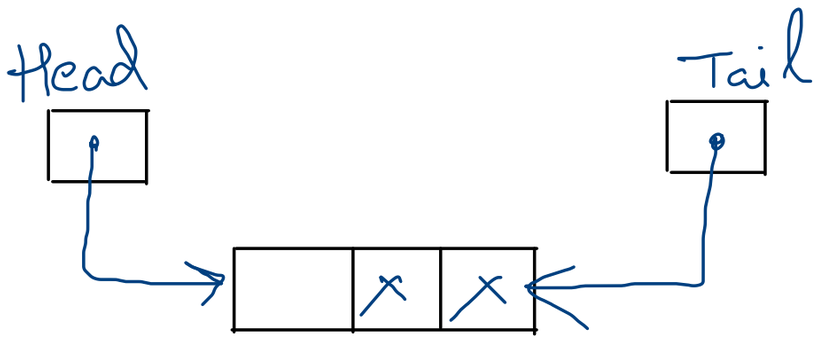




Types of Doubly Linked List

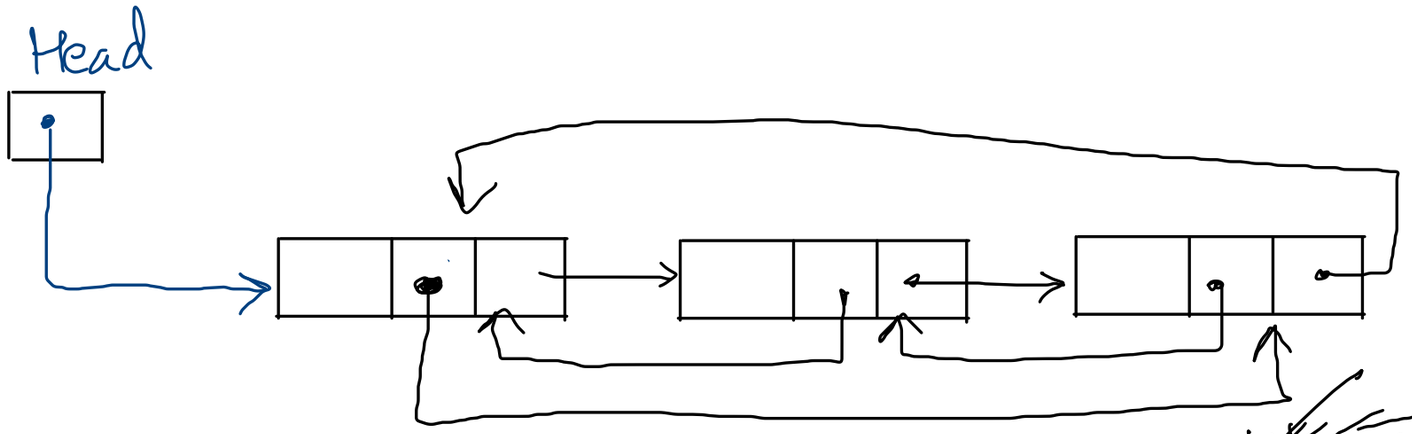
1. Doubly Header Linked List

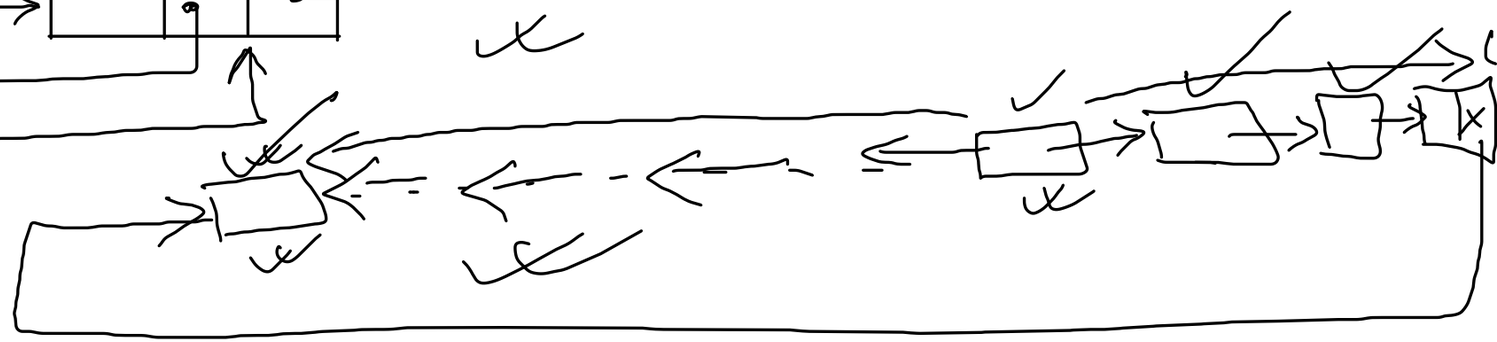




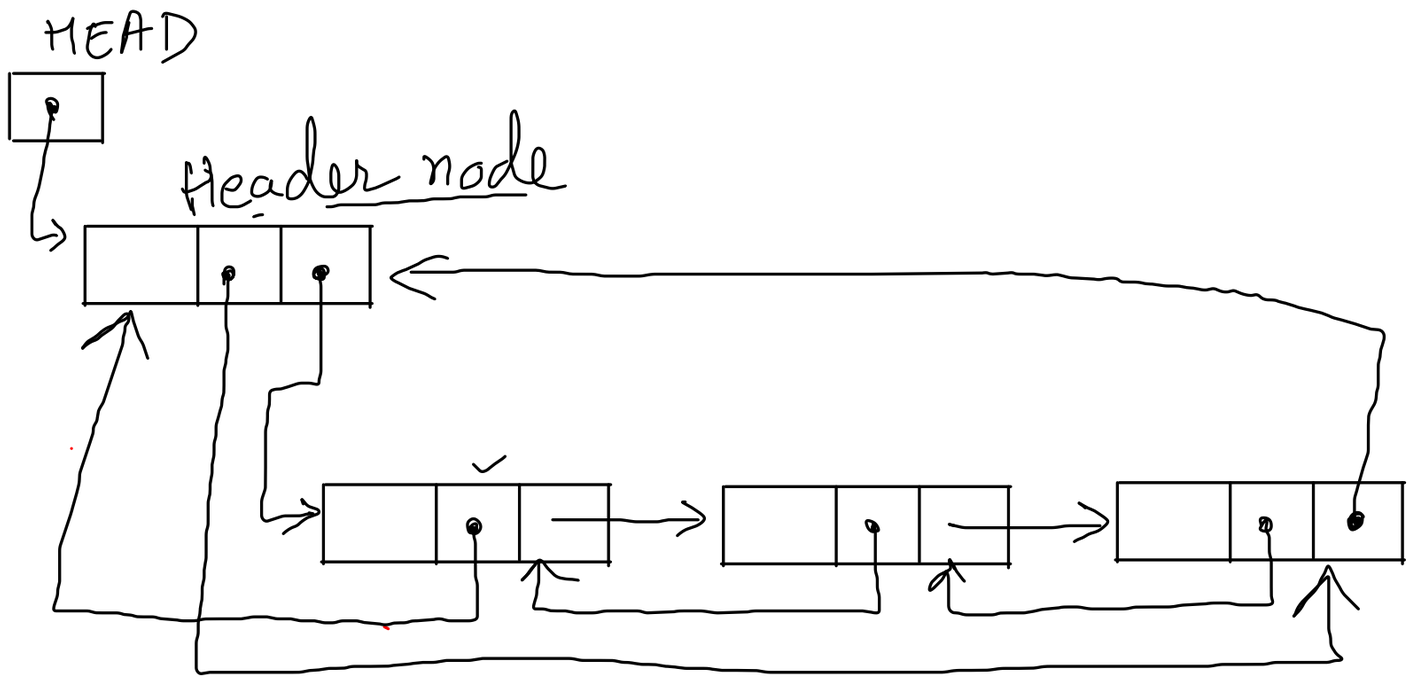
An empty Doubly linked list with header node.

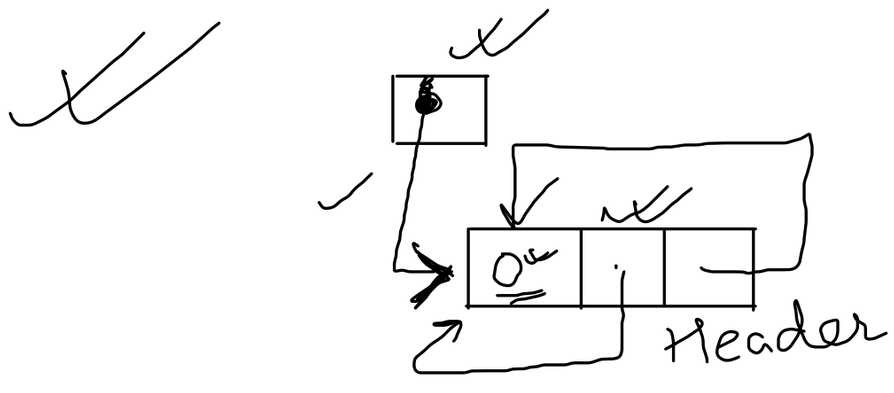
2. Doubly Linked Circular List





3. Doubly Circular Header Linked List





Operations on Doubly Linked Lists:

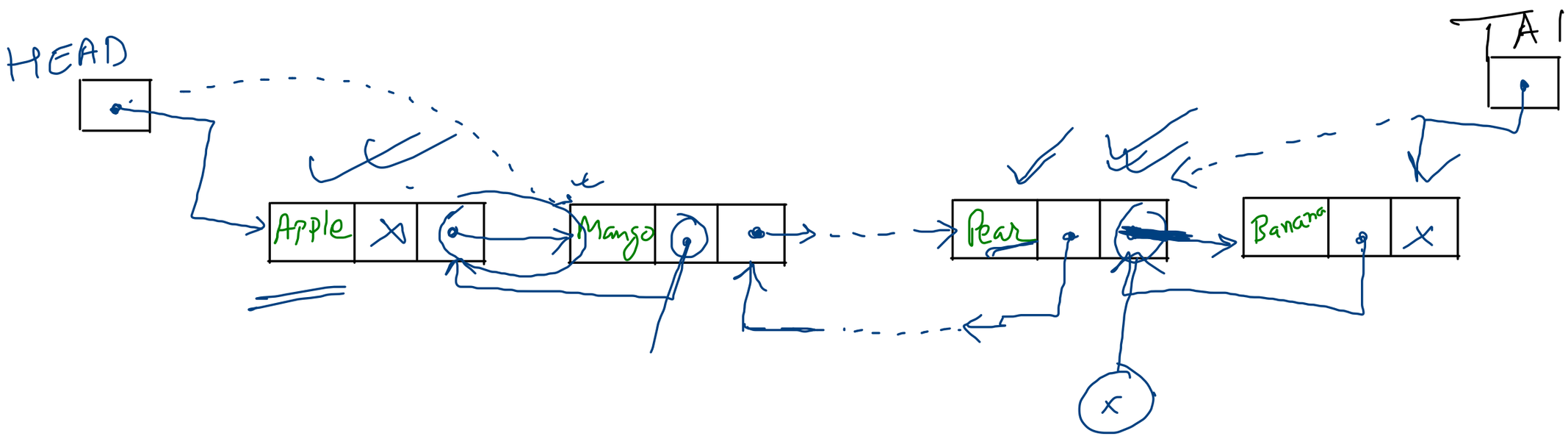
(1) Traversing (2) Searching (3) Deletion (4) Insertion.

(1) Traversing in Doubly Linked list.

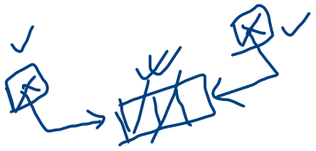
- Can be traversed in forward as well as the backward direction.
- During forward traversal, we start using HEAD pointer and follow the chain of links using forward pointers FLINK.
- During backward traversal, we traverse in the backward direction using TAIL pointer and follow the chain of PLINK pointers.

2) Searching in Doubly Linked List

- Involves finding location LOC of the node containing desired ITEM from Doubly Linked List.
- we can begin searching in either direction.



(3) Deletion from Doubly Linked List:



Possible situations

a) List contains only a single node

This results in empty list with both HEAD and TAIL pointers set to NULL

Algo:

HEAD = TAIL = NULL

b) Node to be deleted is leftmost node.

We need to change the HEAD pointer.

Algo:

HEAD ← FLINK(HEAD)

PLINK(HEAD) = NULL

c) Node to be deleted is the rightmost node

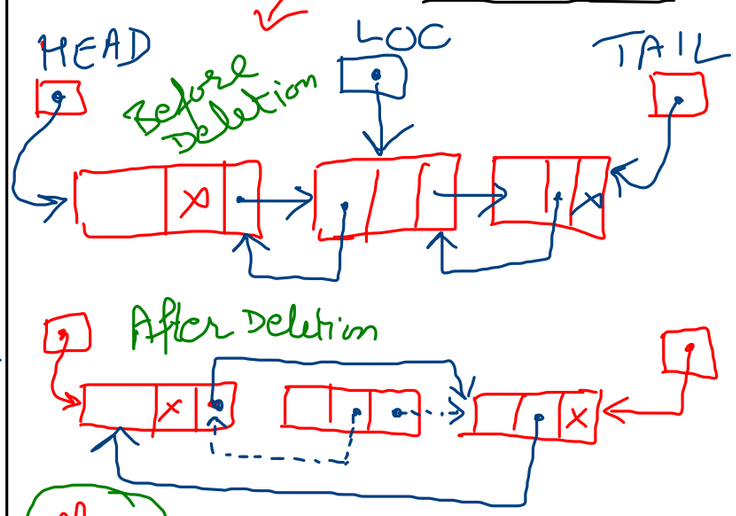
We need to change the TAIL pointer

Algo:

TAIL = PLINK(TAIL)

FLINK(TAIL) = NULL

d) Node appears somewhere in middle.



Algo

FLINK(PLINK(LOC)) = FLINK(LOC)

PLINK(FLINK(LOC)) = PLINK(LOC)

Algorithm: Deletion of a node from a doubly Linked List. (all cases in one algo.)

DOUBLEL L_Del [HEAD, TAIL, LOC)

1) [Empty List]

If TAIL = NULL then
write "Underflow"
return
endif

2) [Delete Node]

If HEAD = TAIL then
HEAD = TAIL = NULL
(only one node in the list.)

else if LOC = HEAD, then
HEAD = FLINK(HEAD)
PLINK(HEAD) = NULL
(To delete leftmost node.)

elseif LOC = TAIL then
TAIL = PLINK(TAIL)
FLINK(TAIL) = NULL
(To delete rightmost node.)

else
FLINK(PLINK(LOC)) = FLINK(LOC)
PLINK(FLINK(LOC)) = PLINK(LOC)
(node to be deleted is somewhere in the middle of list.)

<Endif>

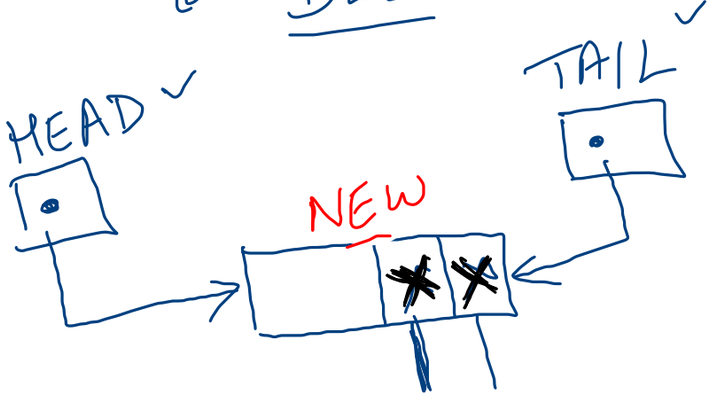
3) [Return deleted node to free storage list]

FLINK(LOC) = Avail
AVAIL = LOC

4) Return.

(4) Insertion of a node in Doubly Linked List (DLL)

Case-1: In case there is no node in the DLL.



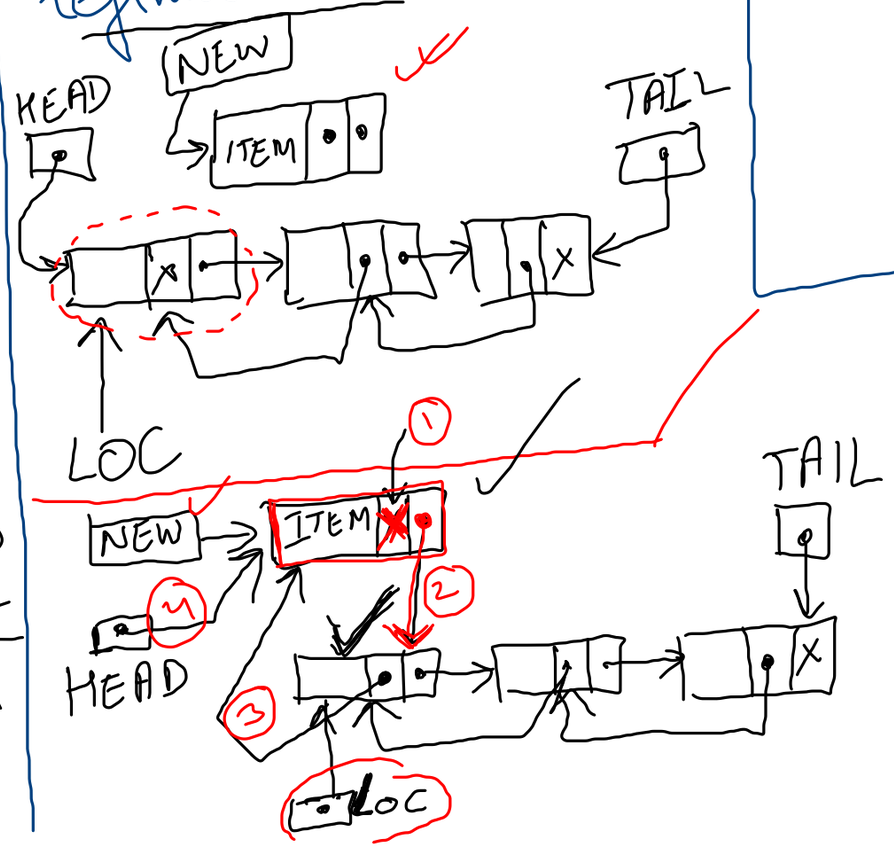
Algo:

If HEAD = NULL

i) PLINK(NEW) = FLINK(NEW) = NULL

ii) HEAD = TAIL = NEW.

Case-2: Inserting a node to the left of leftmost node.



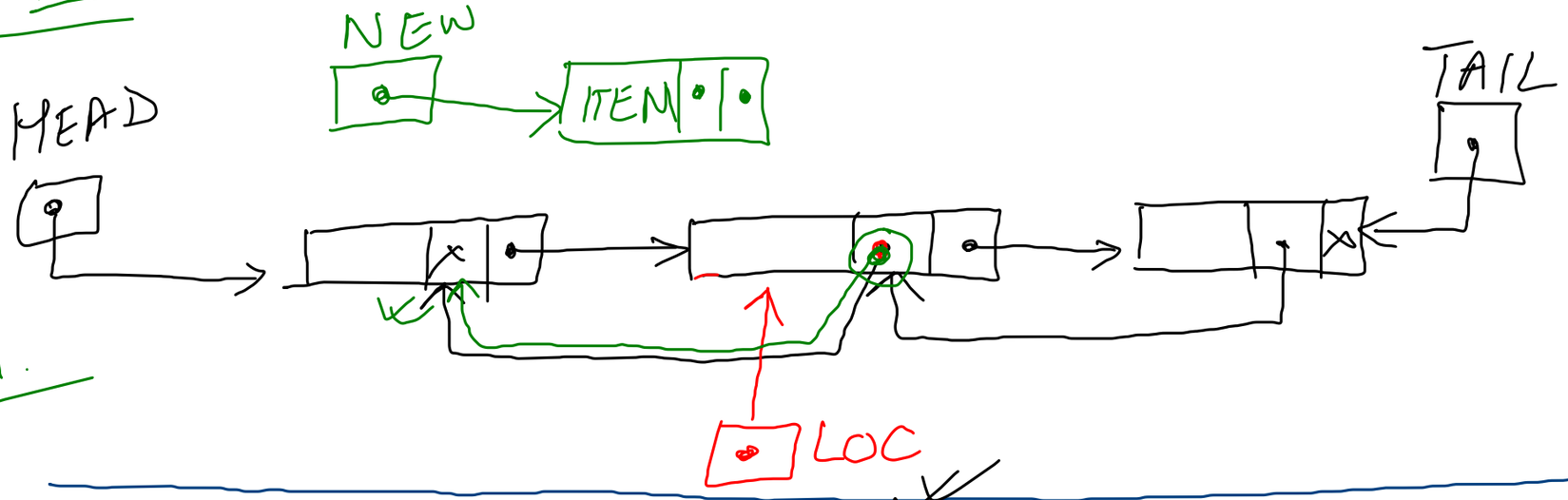
Case-3: Inserting a node somewhere in the middle of the list before a specified LOC.

See Next Page

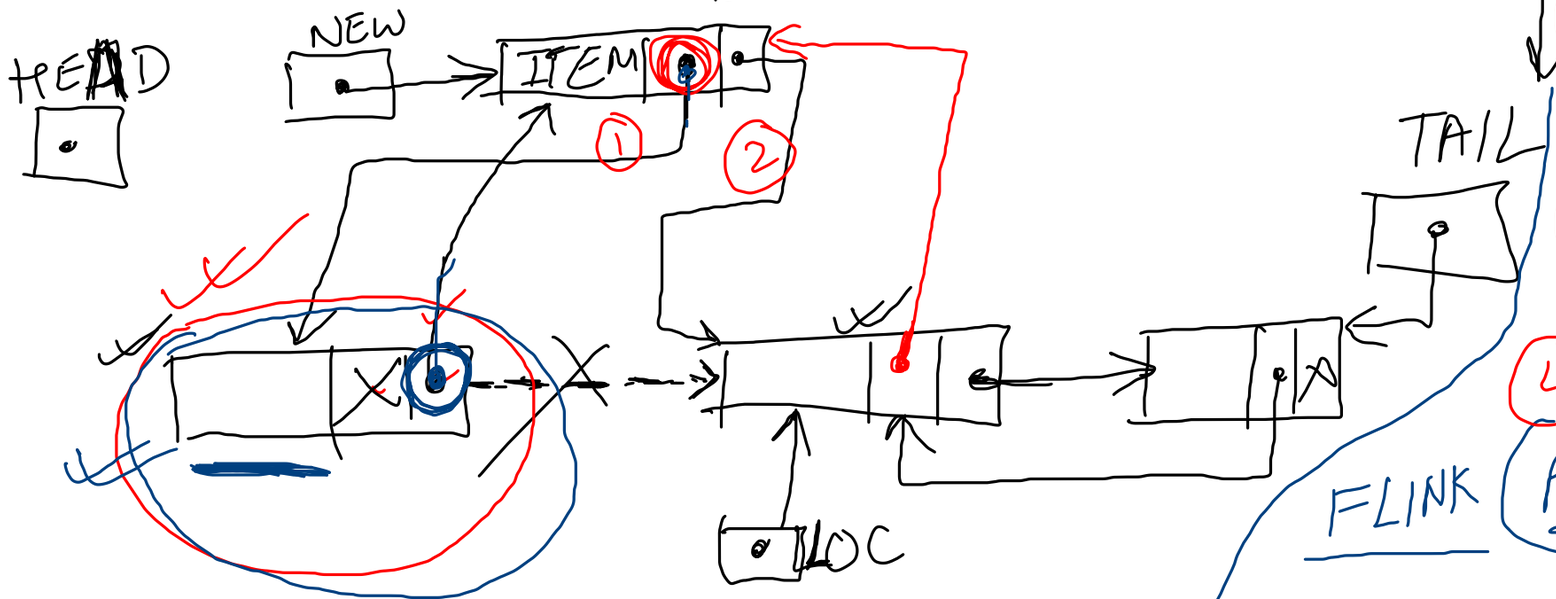
1. PLINK(NEW) = NULL
2. FLINK(NEW) = LOC
3. PLINK(LOC) = NEW
4. HEAD = NEW.

Case - III

Before Insertion.



After Insertion.



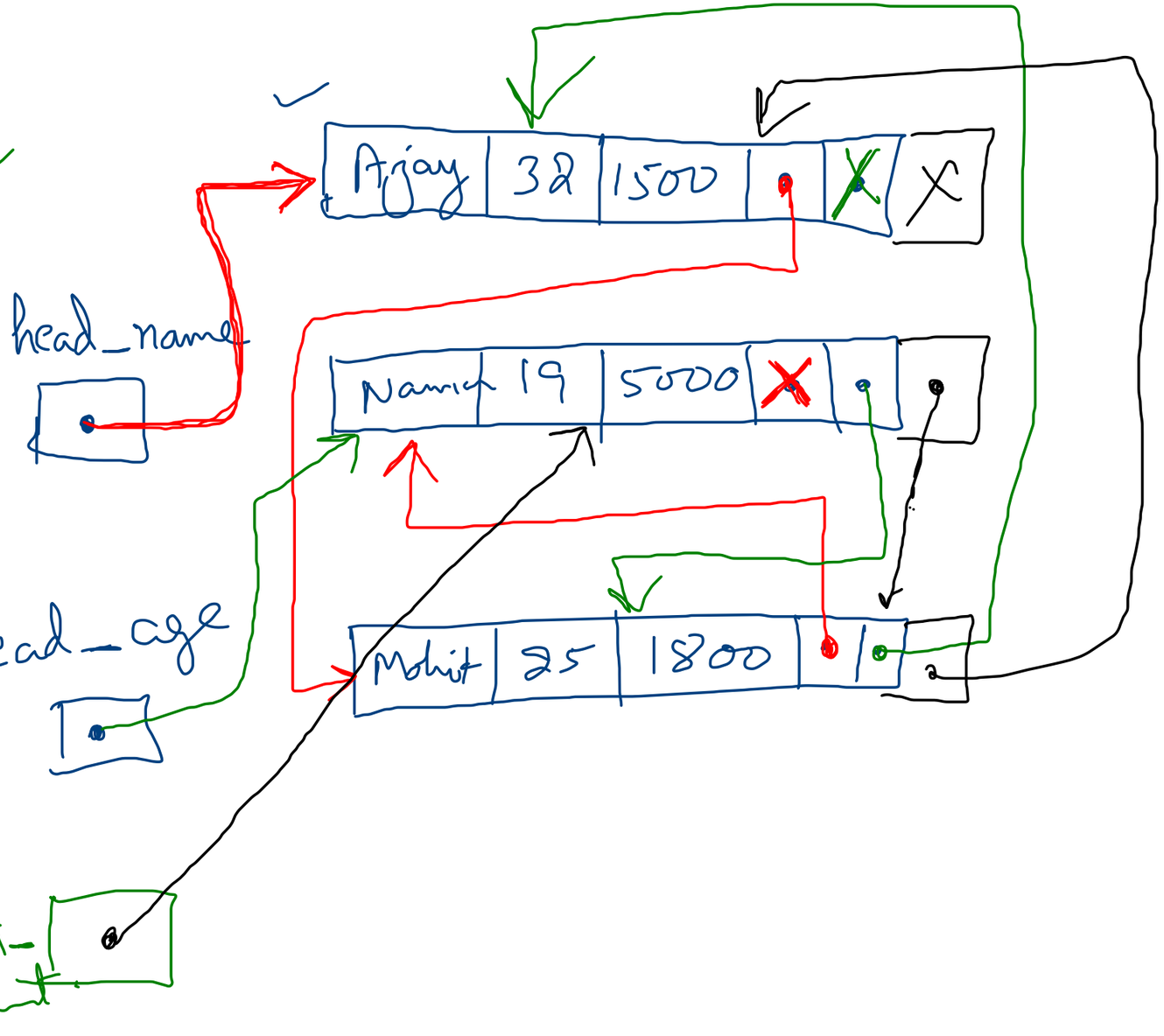
Algo:

1. PLINK(NEW) = PLINK(LOC)
2. FLINK(NEW) = LOC
3. PLINK(LOC) = NEW
4. FLINK(PLINK(NEW)) = NEW.

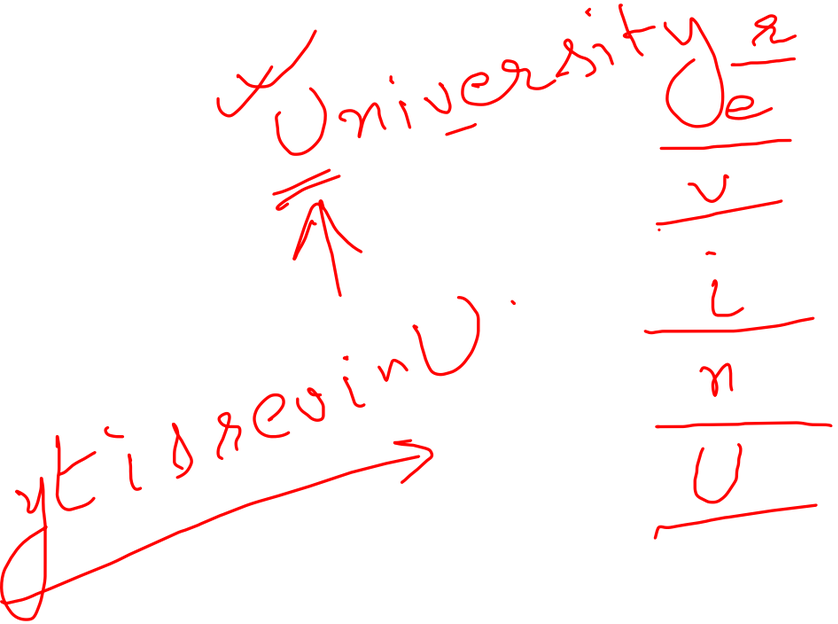
MULTI-LINKED LISTS :

Example:

Name	Age	Amount
Ajay	32	1500
Naman	19	5000
Mohit	25	1800



STACKS:



Application of stacks

1) Reversing of data.
2) Recursion.
3) Evaluation of Arithmetic Exp.
4) Backtracking.
5) Dilimiter matching.
6) Sorting data using quicksort.
7) Processing of function calls.
8) Adding very large numbers.

Evaluation of Arithmetic Expressions.

+, -, *, /, ^

A+B*C

A*(B-C)/D

5*(10-3)/2.

5*7/2.

35/2. = 17.5

5+10x3

= 15x3 = 45

5+10x3 = 35

A = 5

B = 10

C = 3

D = 2

^ ^

op.	Associativity
^	R -> L
*, /	L -> R
+, -	L -> R.

NOTATIONS for Arithmetic Expressions:

1. Infix
   - A+B,
   - C-D,
   - A+B*C
   - A*(B-C)
   - (A+B)/C - E*F + G*2
2. Prefix
   - +AB
   - -CD
   - *DF
   - ⋮

   Polish Notation.
3. Postfix
   - AB+
   - CD-
   - DF/

   Reverse Polish Notation.

Converting Infix expression to Postfix Expression.

Step-1: Fully paranthesize the given Infix Ex.

Step-2: Convert the subexp. in the innermost paranthases into postfix notion.. Repeat for all sub-Exps.

step-3 Remove all paranthesis.

Infix	Postfix
A * B	AB*
(A+B)/C	AB+C/
A * B ^ C	ABC^*

$A + B / C * D$ (Infix Exp.)

Step-1

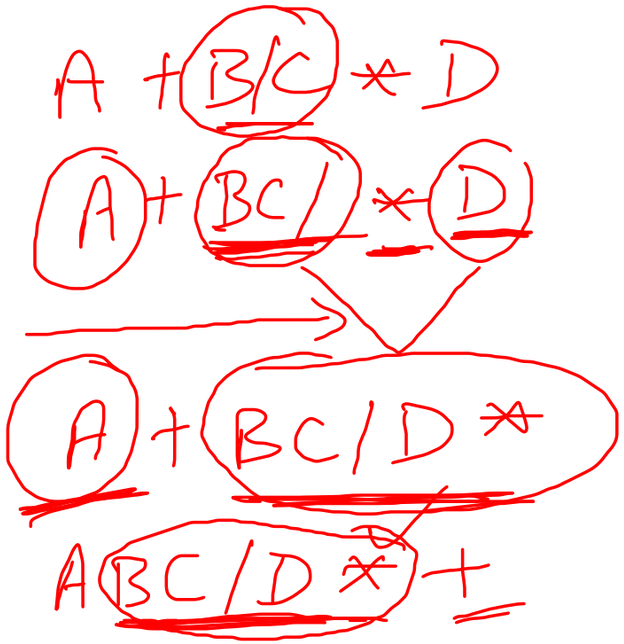
$(A+((B/C)*D))$

$(A+((BC/)*D))$

$(A+(BC/D*))$

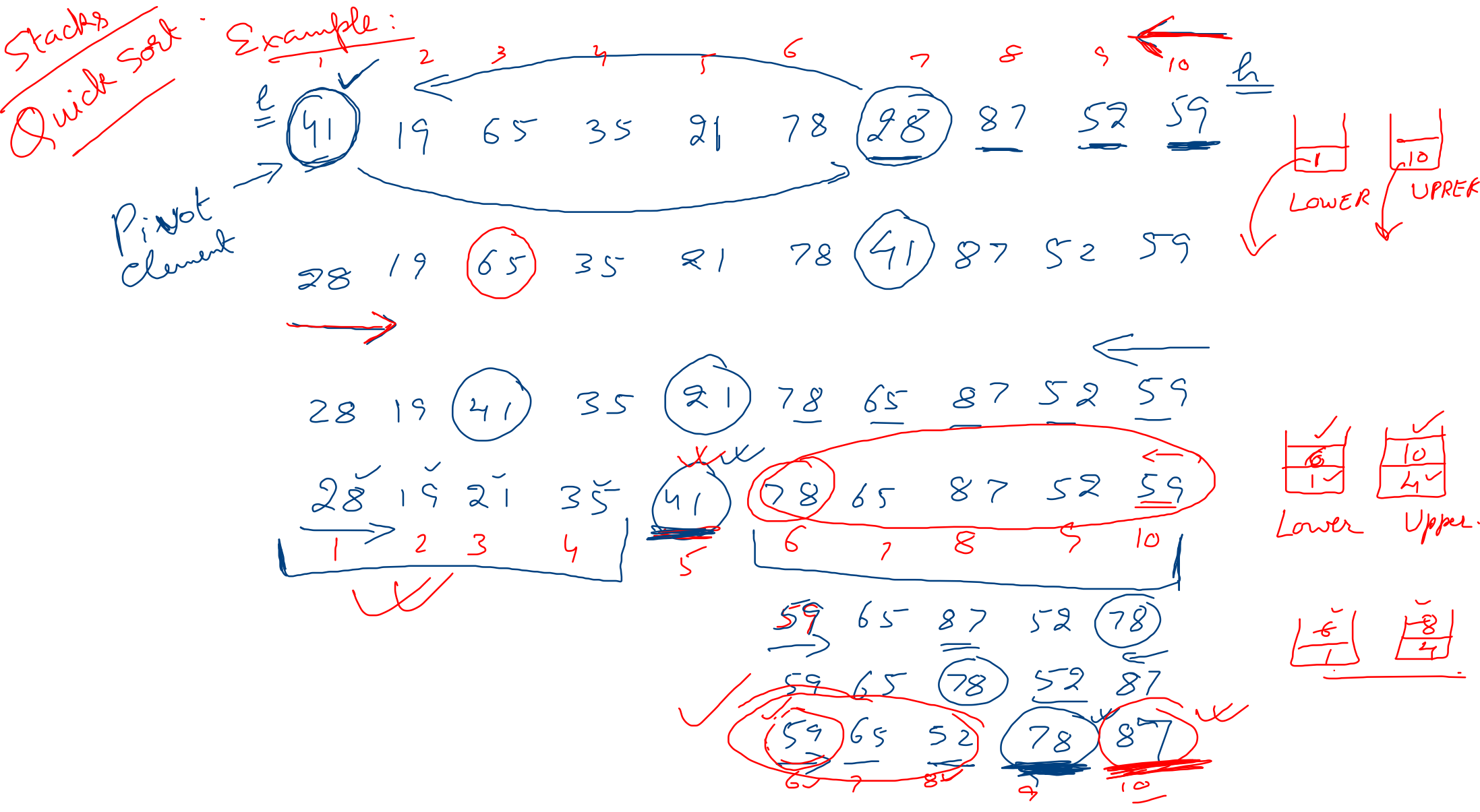
$((ABC/D*)+)$

$ABC/D*+$


A + B/C * D
A + BC/ * D
A BC/D * +
A BC/D * +

Stacks
Quick sort

Example:



Complexity of Quick Sort

Worst Case

$n+(n-1)+(n-2)\text{-----}+1$

$= \frac{n(n+1)}{2}$

$= \frac{n^2+1}{2}$

$= \frac{1}{2}(n^2)+\frac{1}{2}$

$O(n^2)$

Best Case

$O(n \cdot \log n)$

Average case.

$O(n \cdot \log_2 n)$.

Queue :

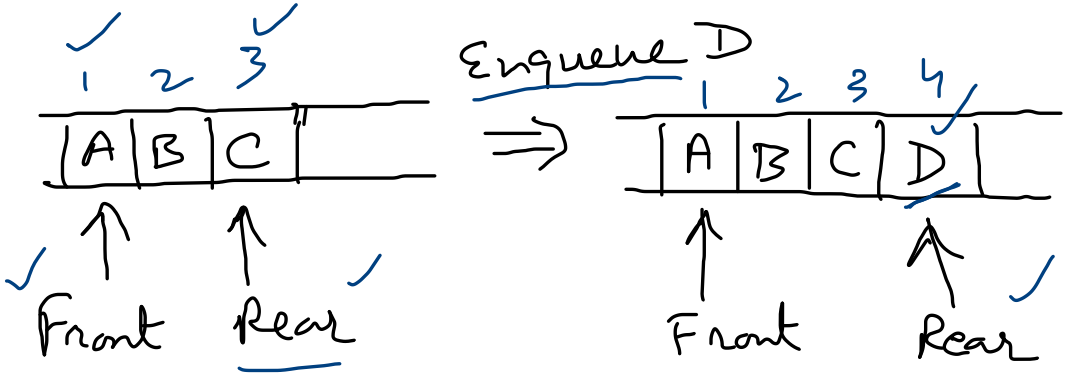
- Linear Data Structure
- Follows FIFO property.

eg:
- Printer maintains a queue
- Timesharing systems
- File servers in computer network
- keys pressed from the keyboard one after the other.
- Information packets in computer n/ws.

Operations on Queues

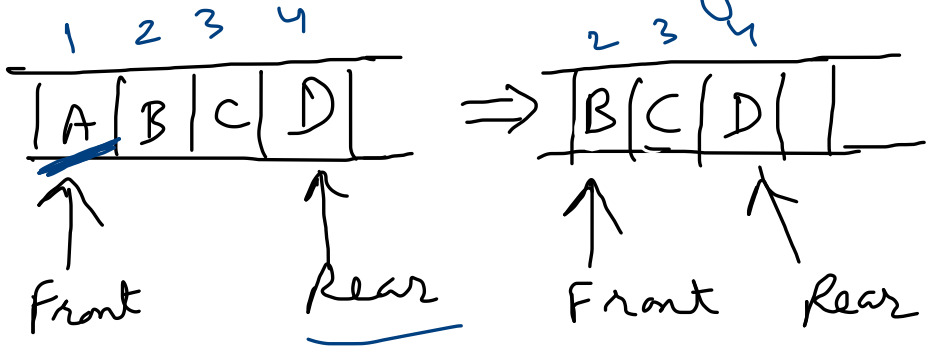
(1) ENQUEUE

Inserting an element



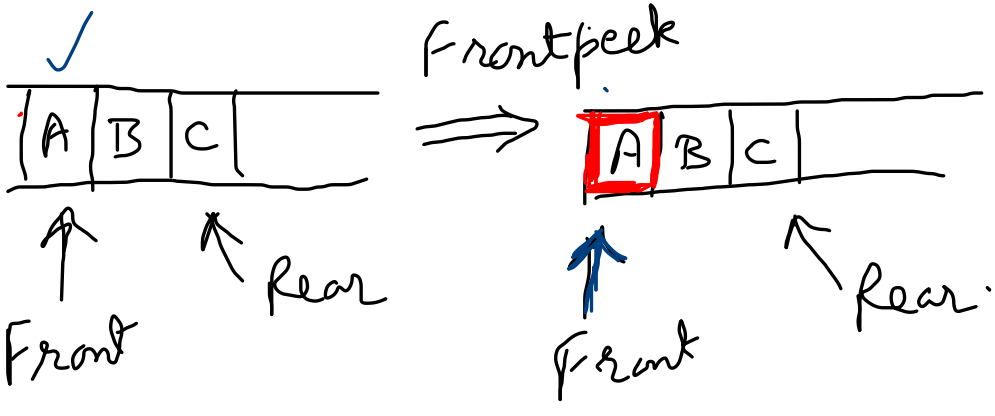
(2) DEQUEUE

Delete an element from queue



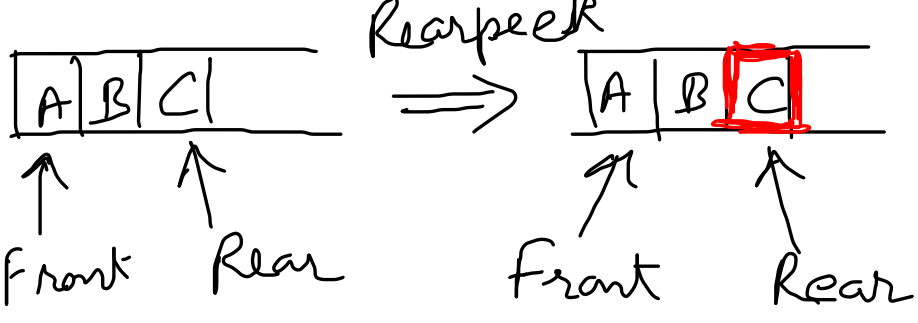
(3) FRONTPEEK

Inspecting first element in queue w/o removing it



(4) REARPEEK

Inspecting last element w/o removing it.

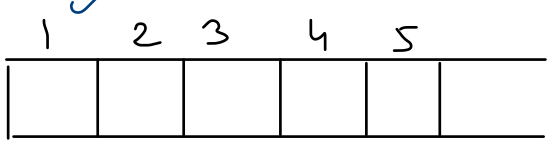


Representation of Queue.

1. ARRAY Representation
2. CIRCULAR Queue (Improved array representation of Queue)
3. LINKED Representation of Queue.

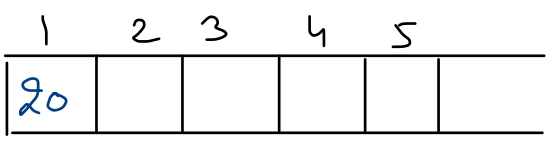
① Array Representation of Queue.

Enqueue

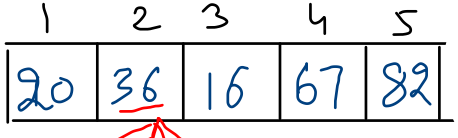


Front = 1
Rear = 0

Queue is empty



Front = 1, Rear = 1.



Front = 1

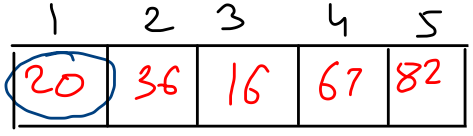
Rear = 5

MaxSize = 5

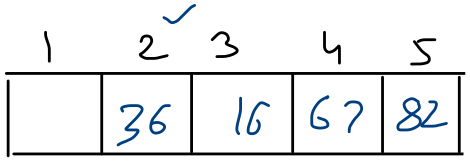
ENQUEUE (Q, Front, Rear, MaxSize, Item)

1) [check the queue for overflow]
   If Rear = MaxSize. then
      write "overflow"
   Endif
2) Rear = Rear + 1
3) Q[Rear] = Item
4) Return.

Dequeue



Front = 1, Rear = 5



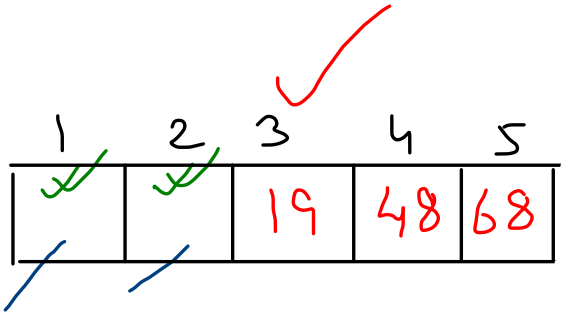
Front = 2, Rear = 5

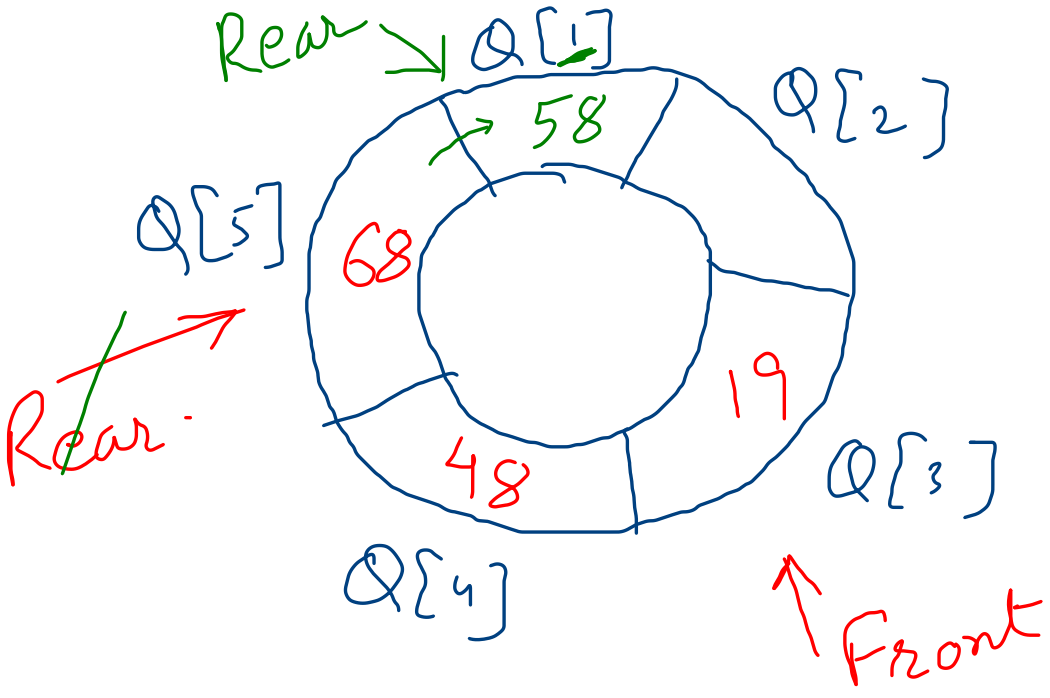
Dequeue (Q, Front, Rear, Maxsize)

1) [check for Underflow]
   If Rear = 0 then
   Write "Underflow"
   Return.
   endif.
2) Item = Q [Front].

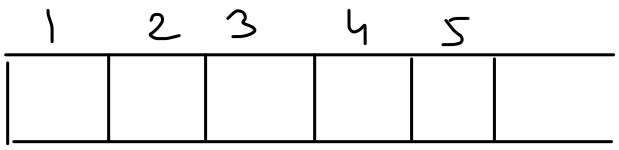
3) If Front < Rear then — multiple elements present in queue
   Front = Front + 1
   Elseif Front = Rear — single element
   Front = 1, Rear = 0.
   End if.
4) Return(Item)

② CIRCULAR QUEUE

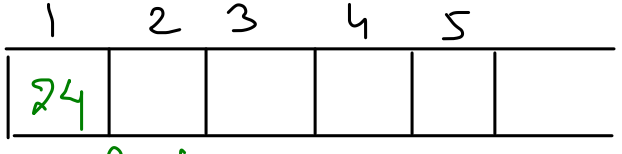




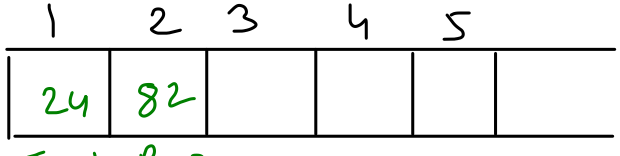
Example of Circular queue



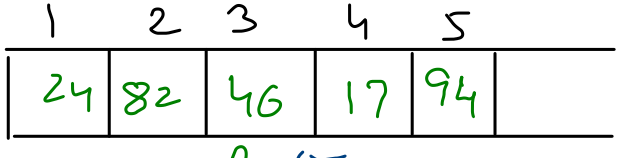
F = 1 R = 0



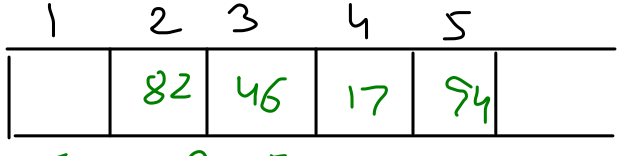
F = 1, R = 1



F = 1, R = 2

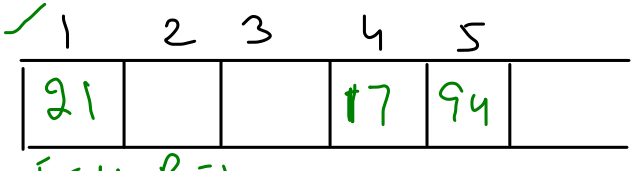


F = 1, R = 5

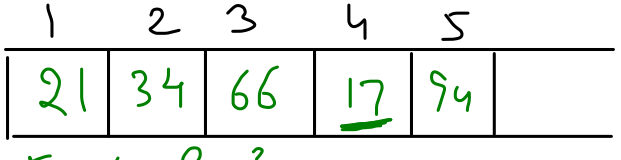


F = 2, R = 5

F = 4, R = 5



F = 4, R = 1

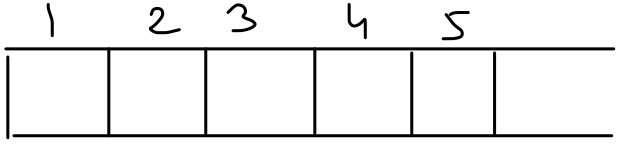


F = 4, R = 3

F = 5, R = 3

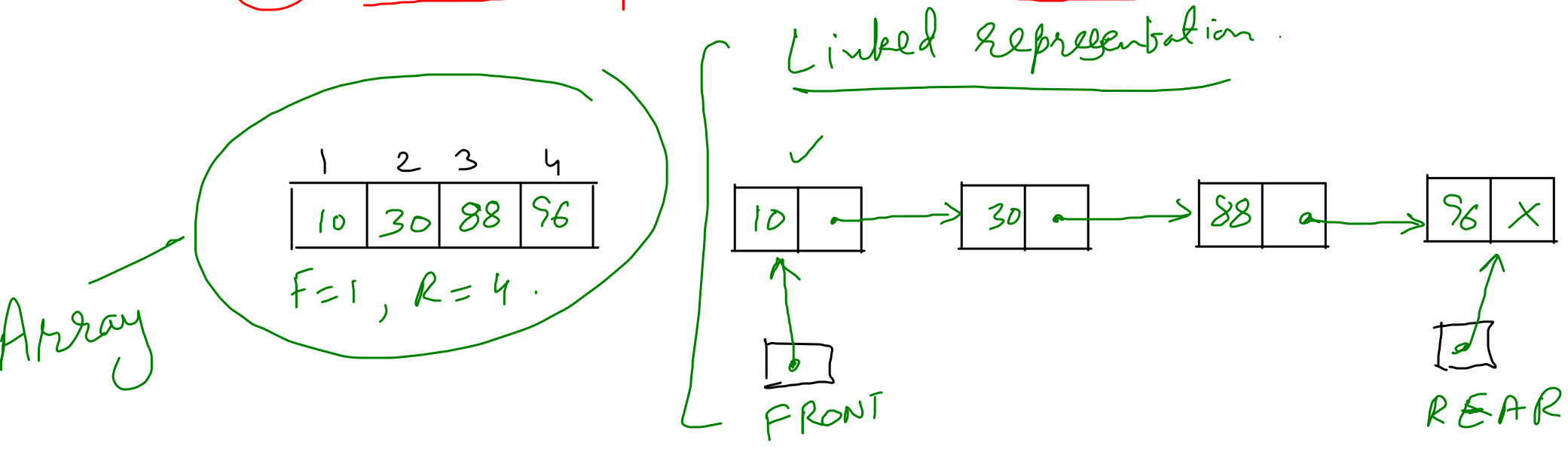
F = 1, R = 66

F = 3, R = 3

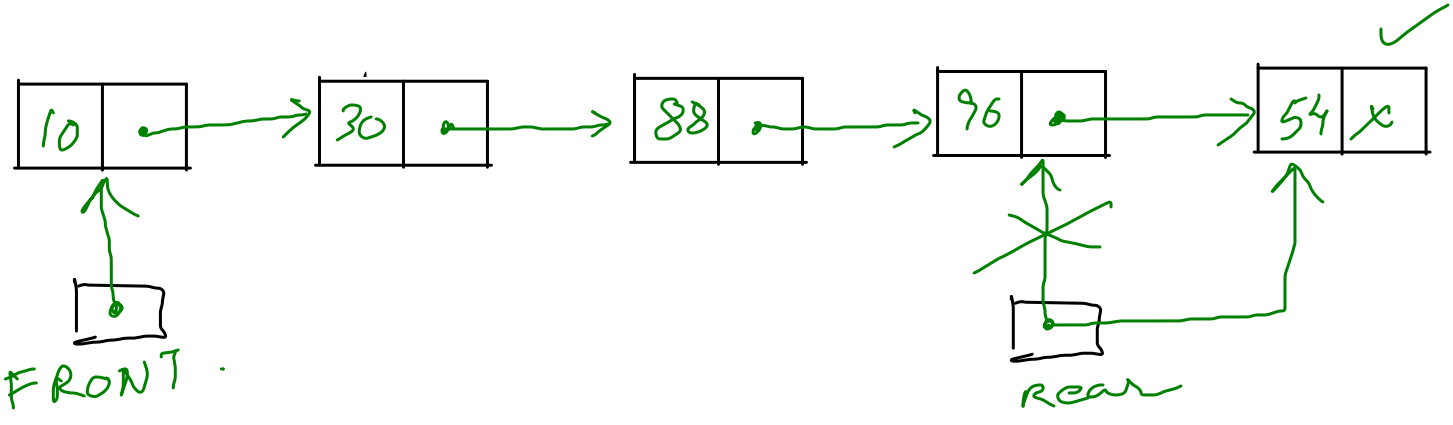


F = 1, R = 0

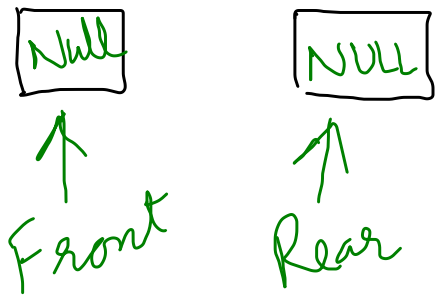
(3) Linked Representation of Queue.

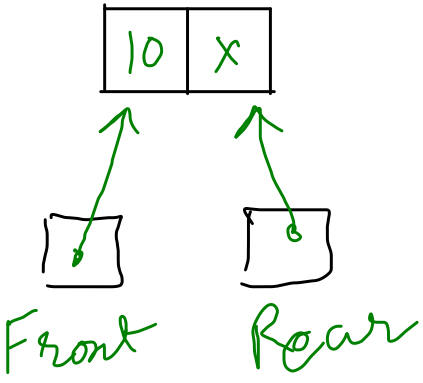


Enqueue:

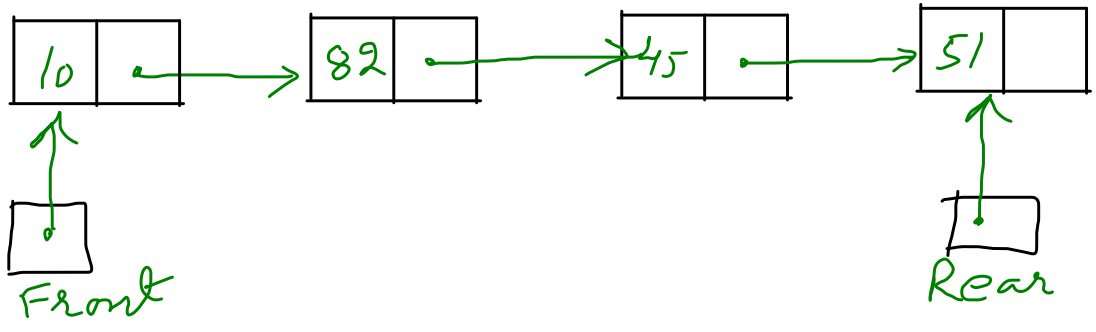


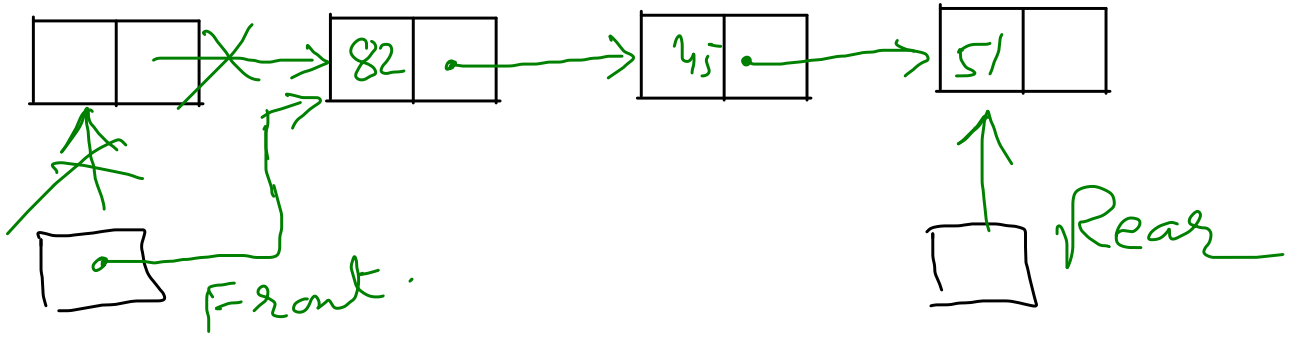
✓ If inserted node is first node



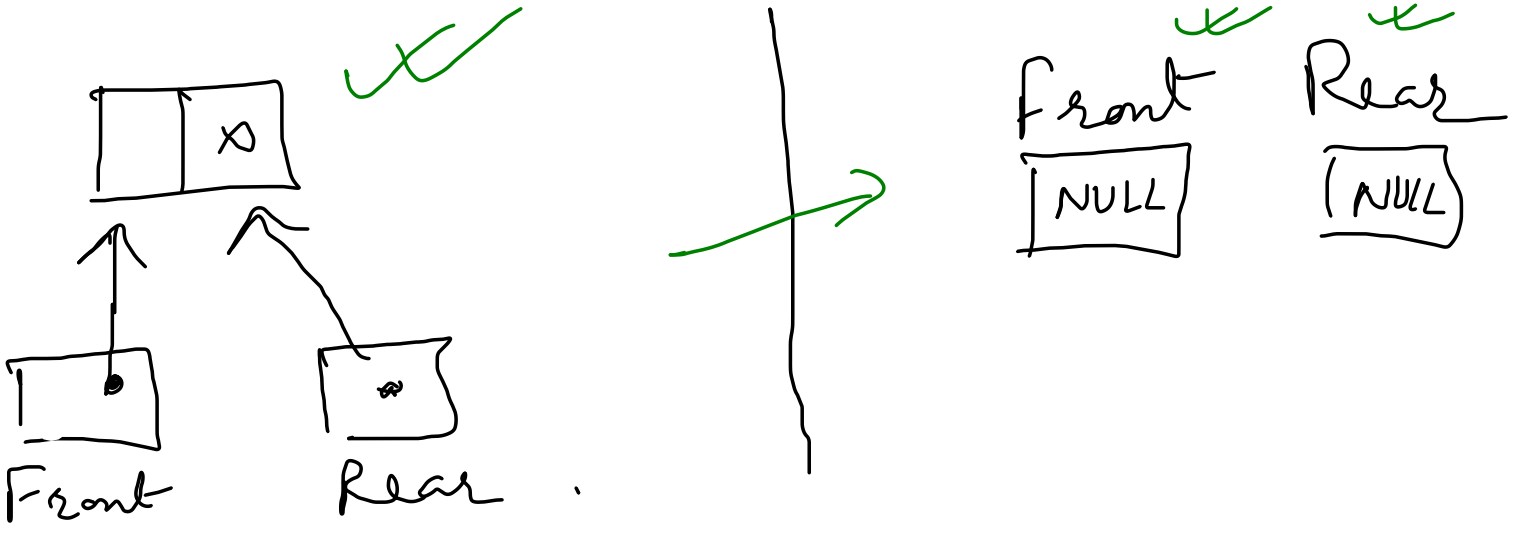


Dequeue





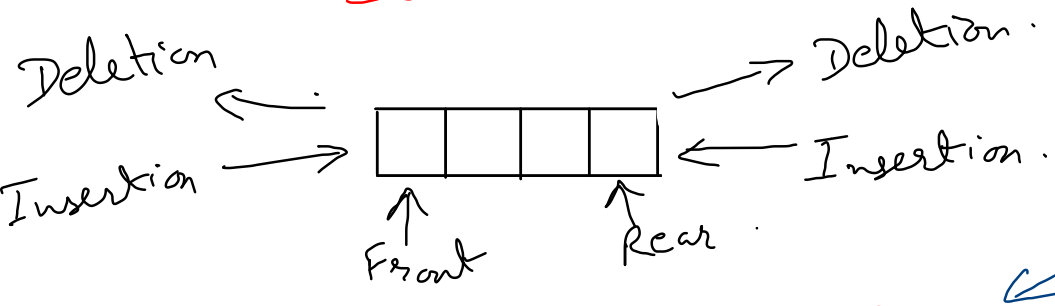
Dequeue operation on linked queue with only one node:



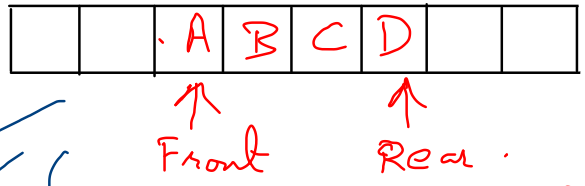
⊛ DEQUE

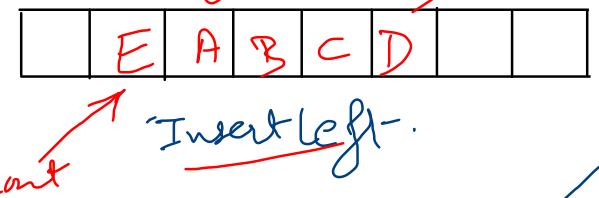
Double Ended Queue.

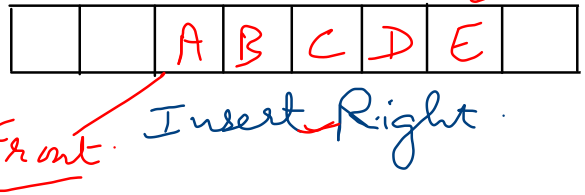
Allows insertion and deletion at either end. (but not from middle).

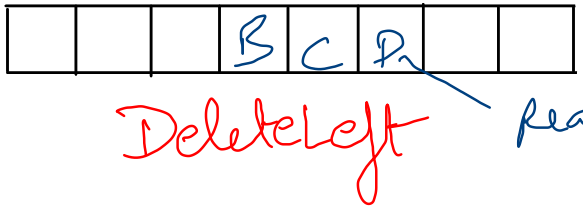


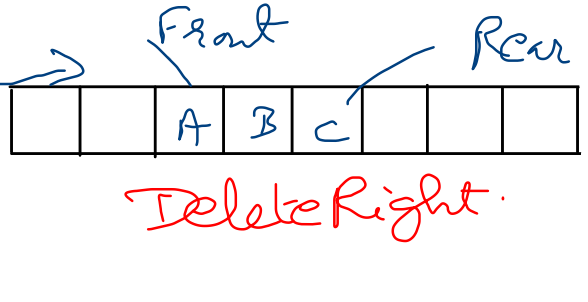
- Insert Left.
- Insert Right
- Delete Left
- Delete Right.











PRIORITY QUEUE

In certain situations, elements need not be processed in the order in which they arrive, but in accordance with the priority assigned to them.

Q_1 Q_2 Q_n =

50 pages — 30 pages — 1 page.

3 2 1

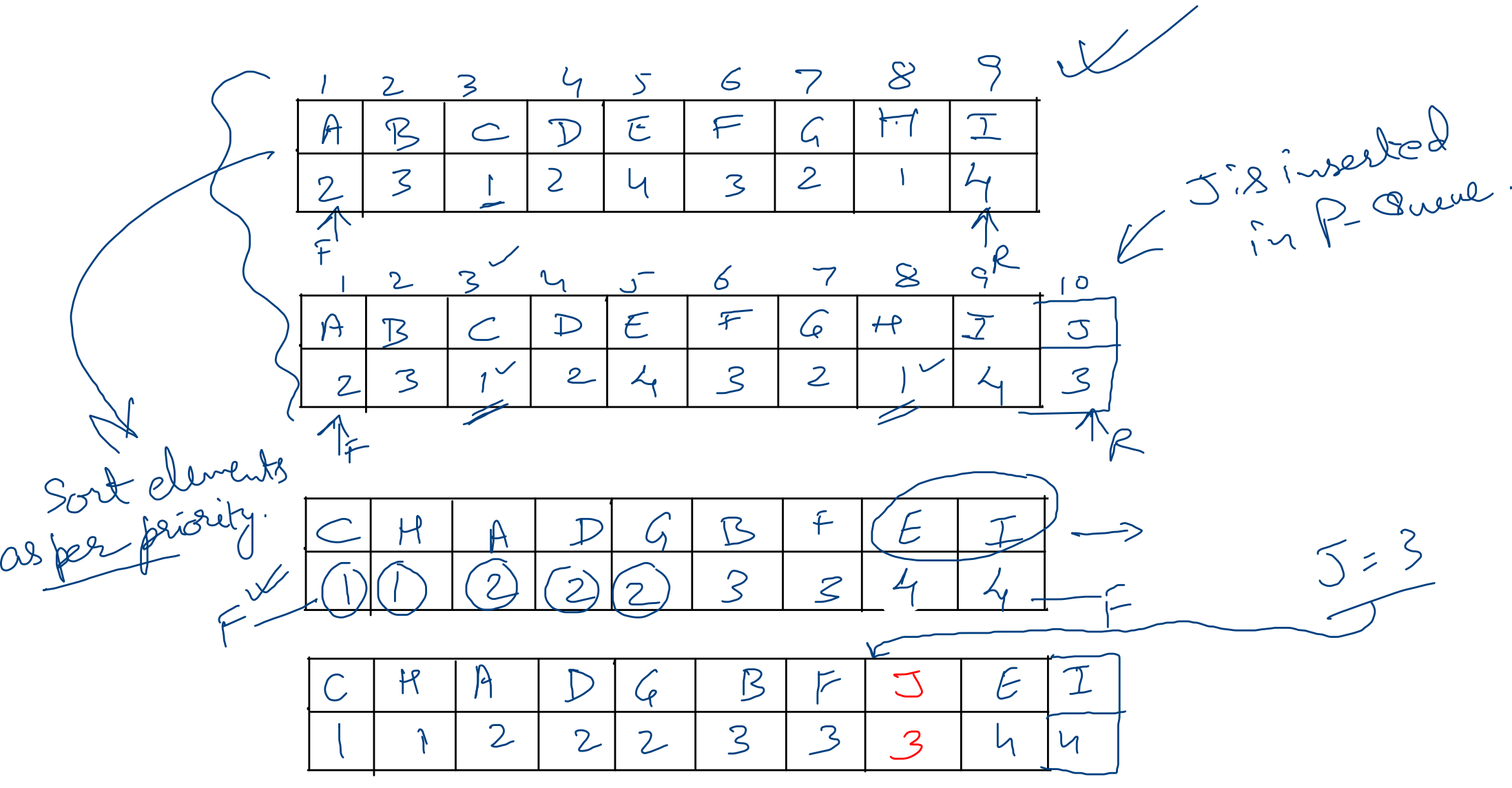
		A	B	.	. . .	X	. . .	Z
		P_1	P_2			P_k		P_n

Front — Rear

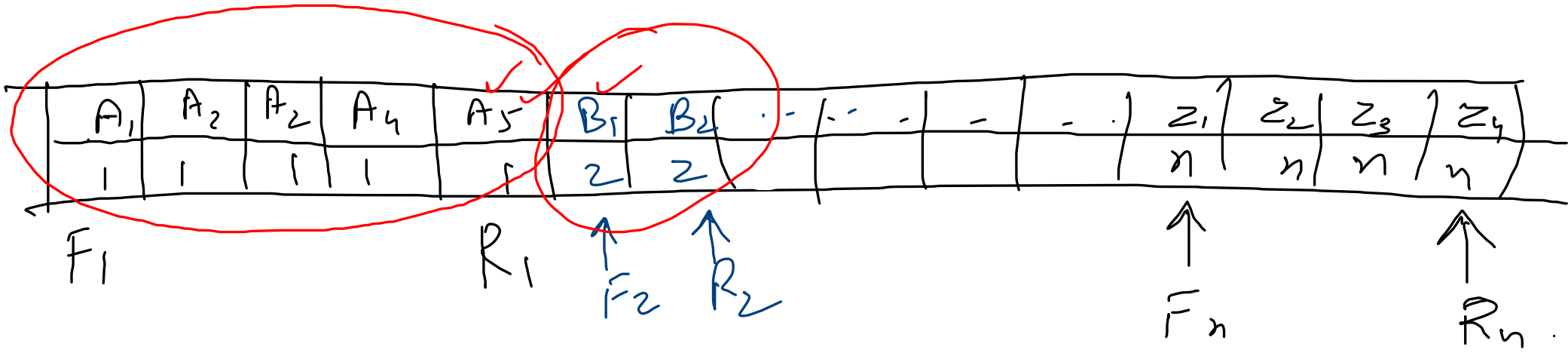
A	B	C	D	E	F	G	H	I
2	3	1	2	4	3	2	1	4

Front — Rear

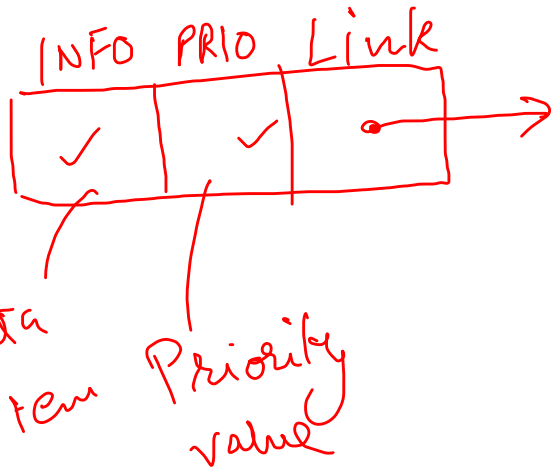
9) Array Representation of Priority Queues.

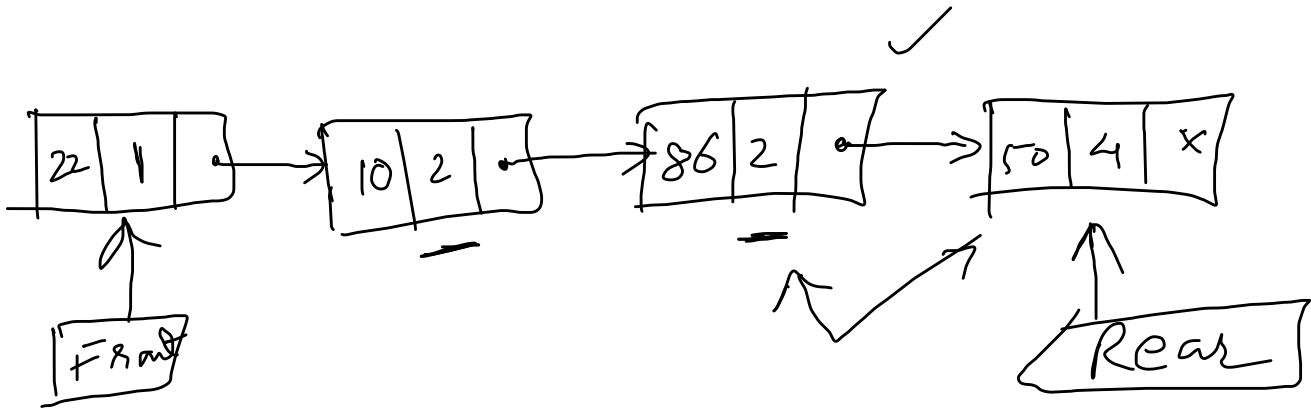


b) Multiuser Representation.



c) Linked Representation





Inserting

to the existing list.