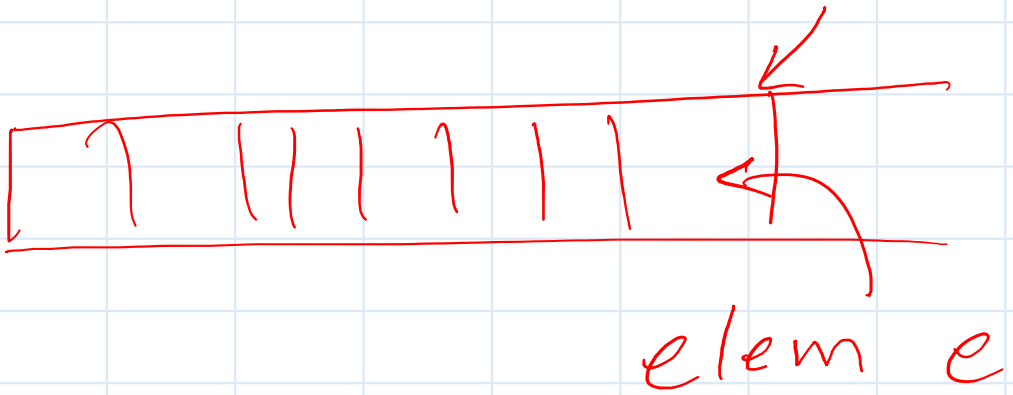
COL106 - Data Structures and Algorithms



Work Involved in Grow – Strategy 1

We start with an array of capacity 1.

Every time we run out of space grow by c

$\Rightarrow$ Over n push operations, we replace by $k = n/c$

total cost of n push operations $= T(n)$

$= n + c + 2c + 3c + \cdots + kc$ (n copies)

$= n + c\,\frac{k(k+1)}{2}$ $\quad k = O(n)$

$\equiv O(n + k^2)$

$\equiv O(n^2)$ $\Leftarrow$

Arithmetic series

Work Involved to Grow – Strategy 2

we start with an array with capacity 1

Double the size each time we run out

Total time for n push operations is $O(n)$

Effectively, each push operation is constant cost!

We replace the array $k = \log_2 n$ times

Total time $T(n)$ over n push operations is

$$T(n) = n + 1 + 2 + 4 + 8 + \cdots + 2^k$$

$$= n + 2^{k+1} - 1$$

$$= 3n - 1$$

$$\Rightarrow T(n) = O(n)$$

Array Implementation of List

⇒ Locating an element at an index i

Constant cost operation O(1)

⇒ Space used by the list

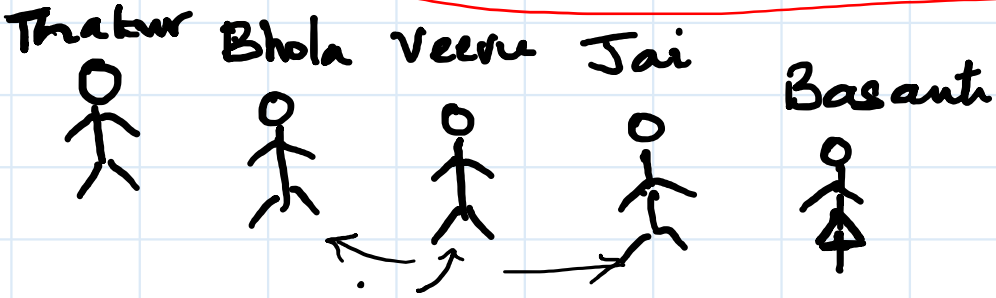
as much as largest it can be

⇒ add / remove operations

Can be as bad as requiring full array copy O(n)

Linked List

In an array an integer index is used determine the location of an element.



- Basanti is before Jai and there is no one before her
- Jai is before Veeru, Veeru before Bhola ----
- Thakur is the last in the list.

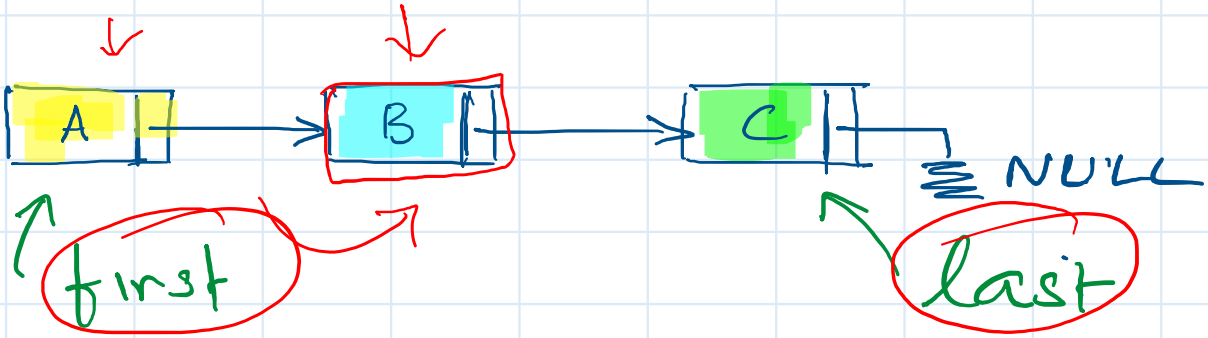
Above is another natural way to define a list.

Linked List

A position is an abstraction that ... is defined by which element is before / after it.

- With this view of position, inserting an element in the middle of a list is straightforward.
- There is no question of a bound on the list.

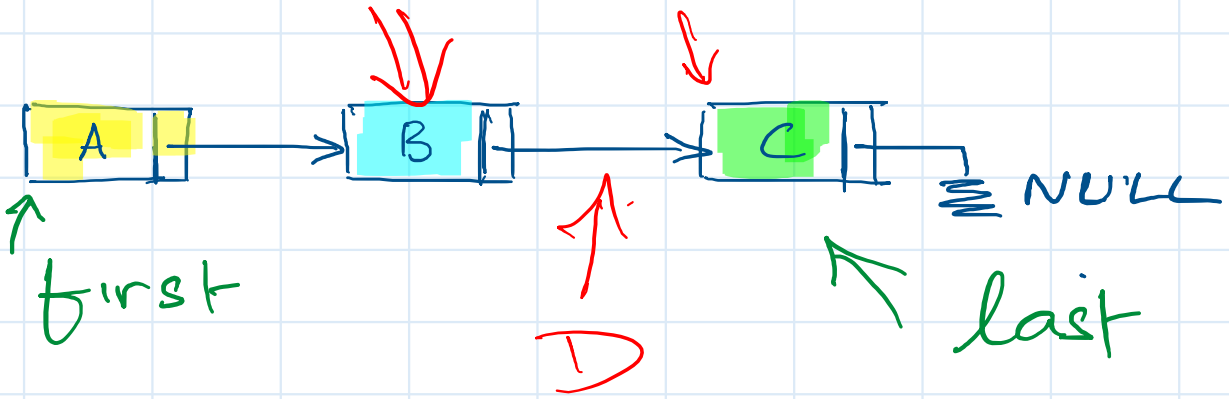
Singly Linked and Doubly Linked List



pos(A) is the predecessor of pos(B)
pos(B) is the successor of pos(A)

- Traverse a single linked list from head to tail
- Index of the position of an element in the list is to be computed.

Singly Linked and Doubly Linked List



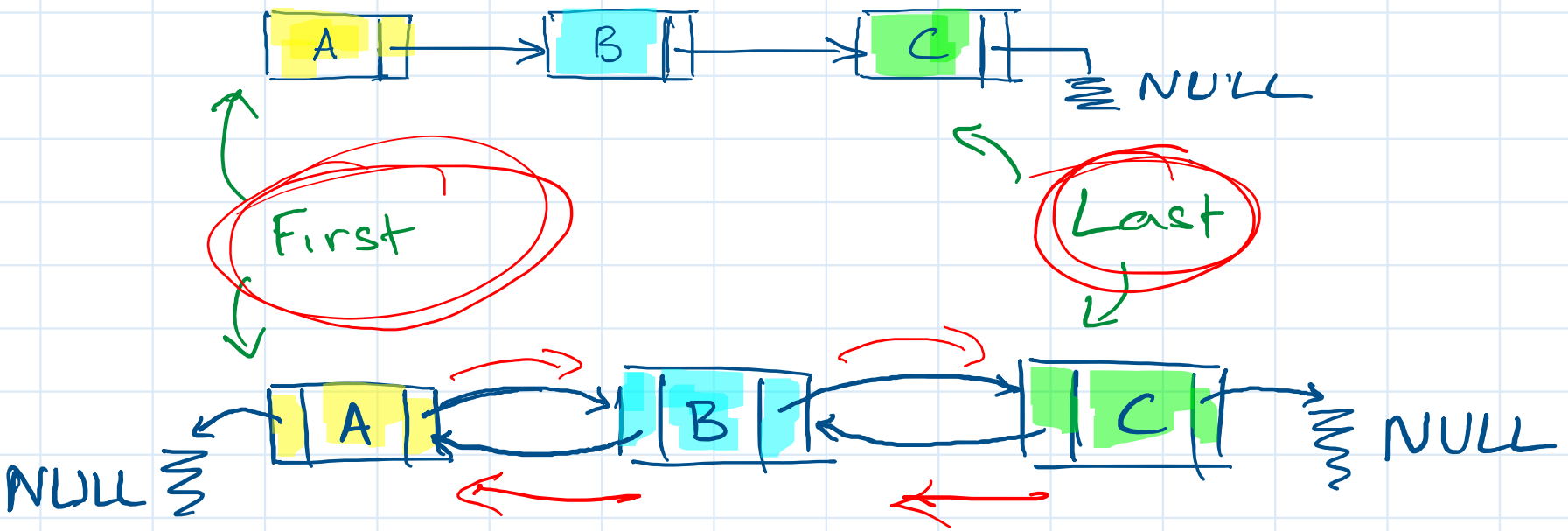
predecessor of the new element that you want to insert

⊛ Adding an element.

(i) Create a position & add element
(ii) Find its right location in the list
(iii) Update its link to point to the **successor of its "predecessor"**.
(iv) Update its **predecessor** to point to itself

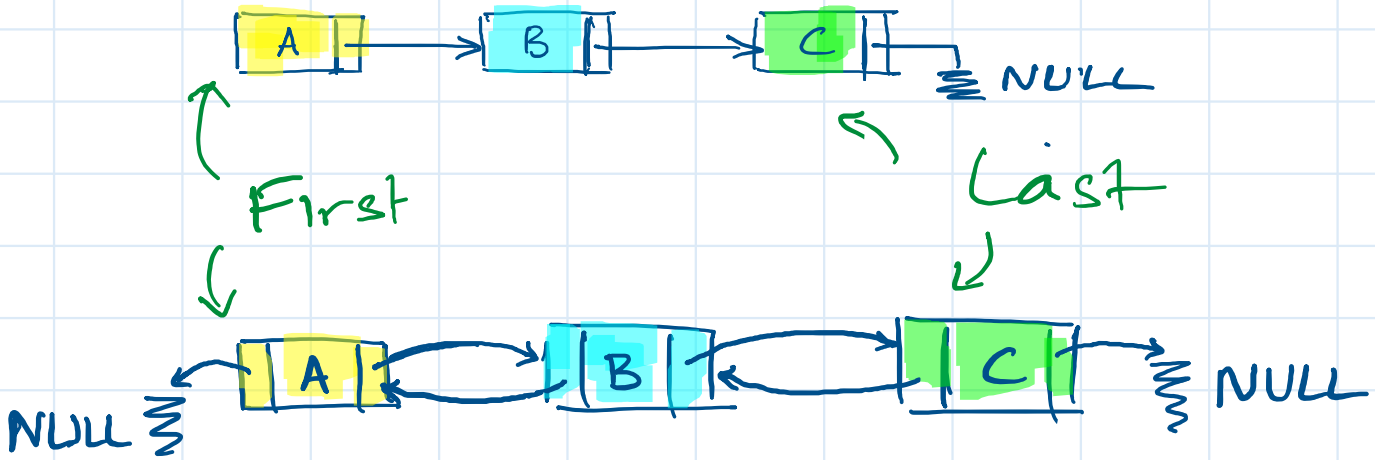
Add to the ~~head~~ of the list first

Singly Linked and Doubly Linked List



Doubly linked list allows you to traverse in both directions.

Single Linked and Double Linked List



⇒ Doubly linked list allows you to insert elements into the list before & after any position

⇒ Space overhead due to additional references

Java (JDK) LinkedList implementation

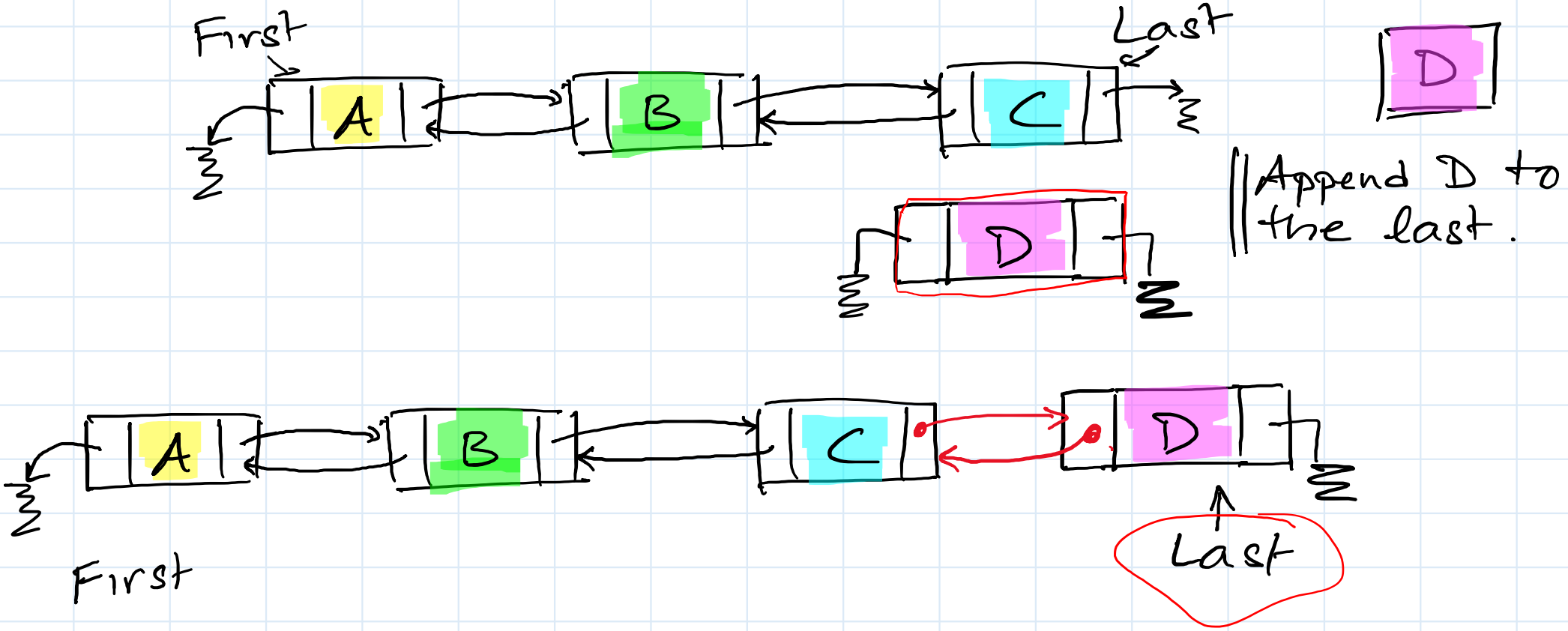
```
public class LinkedList<E> implements List<E>...{
        int size;
        Node<E> first;
        Node<E> last;
        private static class Node<E> {
                E item;
                Node<E> prev; Node<E> next;
                Node (Node<E> prev, E element, Node<E> next) {
                        this.item = element;
                        this.prev = prev;
                        this.next = next;
                }
        }
        ...
}
```

Implementing the Position abstraction

Node() {
}

item = null;
prev = null;
next = null;

Append an Element to Doubly Linked List



Unlinking a Position from the List

```
E unlink(Node<E> x){
  E element = x.item;
  Node<E> next = x.next;
  Node<E> prev = x.prev;
  if (prev == null){
     first = next;
  } else {
     prev.next = next;
     x.prev = null;
  }
  if (next == null) {
     last = prev;
  } else {
     next.prev = prev;
     x.next = null
  }
  x.item = null;
  size--;
  return element;
}
```

x.next

java.util.LinkedList

Array vs LinkedList Implementation of List ADT

Feature	Array	LinkedList
Space Usage	O (N) – N is the maximum possible size Even in growable arrays there is wasted space	O(n) – number of elements in the list
Given integer position, get the element	O(1) – allows random access	O(i) - for locating an element at integer position I
Inserting an element in the middle (given the position)	O(n) – copy all the subsequent elements	O(1)
Deleting an element in the middle (given the position)	O(n) – copy all subsequent elements	O(1)

LinkedList → frequent insert/delete, no need for random access

Array → fast random accesses, stable (relatively) list

What drives the choice of implementation?

1. Efficiency of the algorithm

   ⇒ will the implementation choice help in speeding up the algorithm.

   ⇒ add new methods, improve state / data "structure", storage and access.

2. Resource usage (usually memory).