Linked List outline

- Why linked lists
- Linked lists
- Representation of Linked lists in memory
- Traversing a linked lists
- Memory allocation: Garbage collection
- Overflow and Underflow
- Basic operations of linked lists
  - Insert, find, delete, print, etc.
- Variations of linked lists
  - Circular linked lists
  - Doubly linked lists

Lecture – 5
on
Data Structure

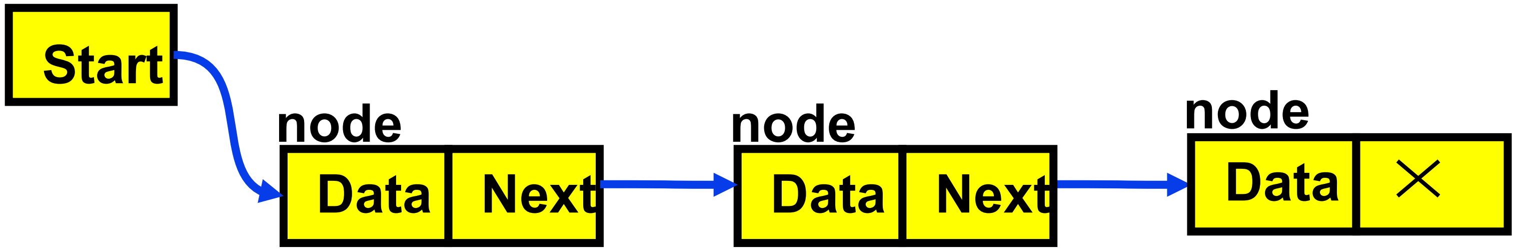
Linked Lists

Why linked lists

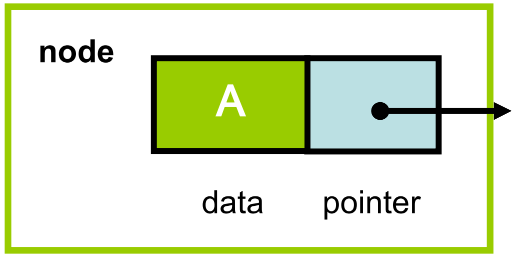
- Disadvantages of arrays as storage data structures:
  - Slow insertion in ordered array
  - Contiguous Size is required
  - Inefficient usage of Memory
- Linked lists solve some of these problems
- Linked lists are general purpose storage data structures.

Linked lists

•A linked list, or one way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.
•Each node is divided into two parts:
•The first part contains the information of the element, and
•The second part, called the link field or next pointer field, contains the address of the next node in the list.
•The pointer of the last node contains a special value,called the null pointer.
•A pointer variable – called START which contains the address of the first node.
•A special case is the list that has no nodes, such a list is called the null list or empty list and is denoted by the null pointer in the variable START.



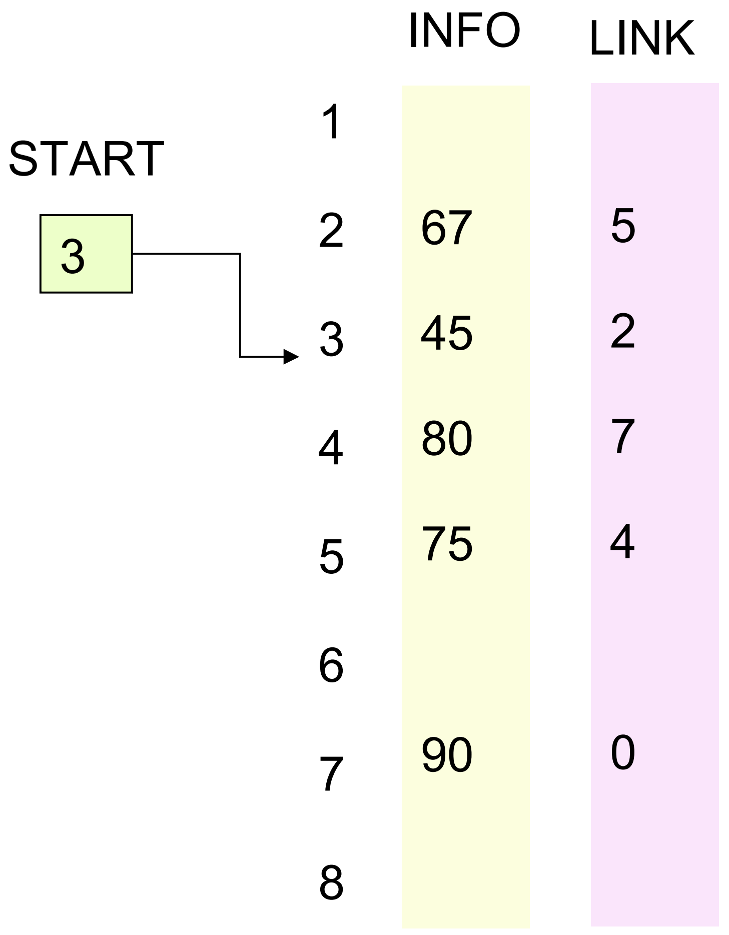
Linked list with 3 nodes



linked lists

- A linked list organizes a collection of data items (elements) such that elements can easily be added to and deleted from any position in the list.
- Only references To next elements are updated in insertion and deletion operations.
- There is no need to copy or move large blocks of data to facilitate insertion and deletion of elements.
- Lists grow dynamically.

Representation of Linked lists in memory



	INFO	LINK
1		
2	67	5
3	45	2
4	80	7
5	75	4
6		
7	90	0
8		

START=3, INFO[3]=45

LINK[3]=2, INFO[2]=67

LINK[2]=5, INFO[5]=75

LINK[5]=4, INFO[4]=80

LINK[4]=7, INFO[7]=90

LINK[7]=0, NULL value, So the list has ended

Traversing a linked lists

LIST be a linked list in memory stored in linear arrays INFO and LINK with START pointing to the first element and NULL indicating the end of LIST.

We want to traverse LIST in order to process each node exactly once.

Pointer variable PTR points to the node that is currently being processed.

LINK[PTR] points to the next node to be processed.

Thus update PTR by the assignment

PTR : =LINK[PTR]

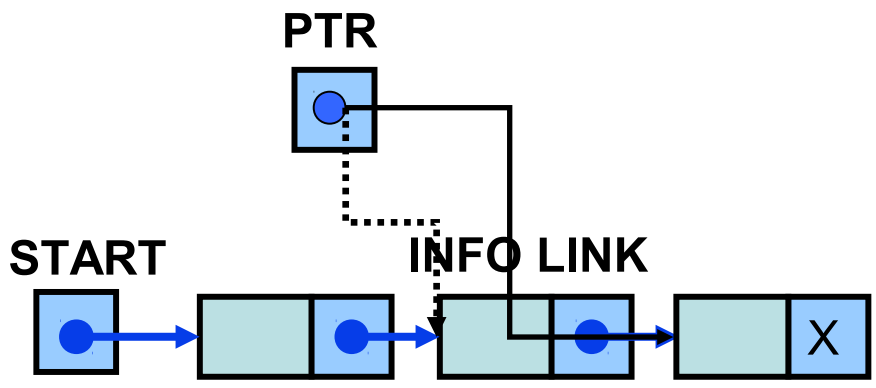


Fig : PTR : = LINK[PTR]

Traversing a linked lists

For printing the information at each node of a linked list, must traverse the list.

Algorithm 5.1 : PRINT(INFO, LINK, START)

Algorithm Prints the information at each node of the list.

```
1. Set PTR : =START.
2. Repeat steps 3 and 4 while PTR : ≠ NULL:
3. Write : INFO[PTR].
4. Set PTR : =LINK[PTR].
5. Exit.
```

Traversing a linked lists

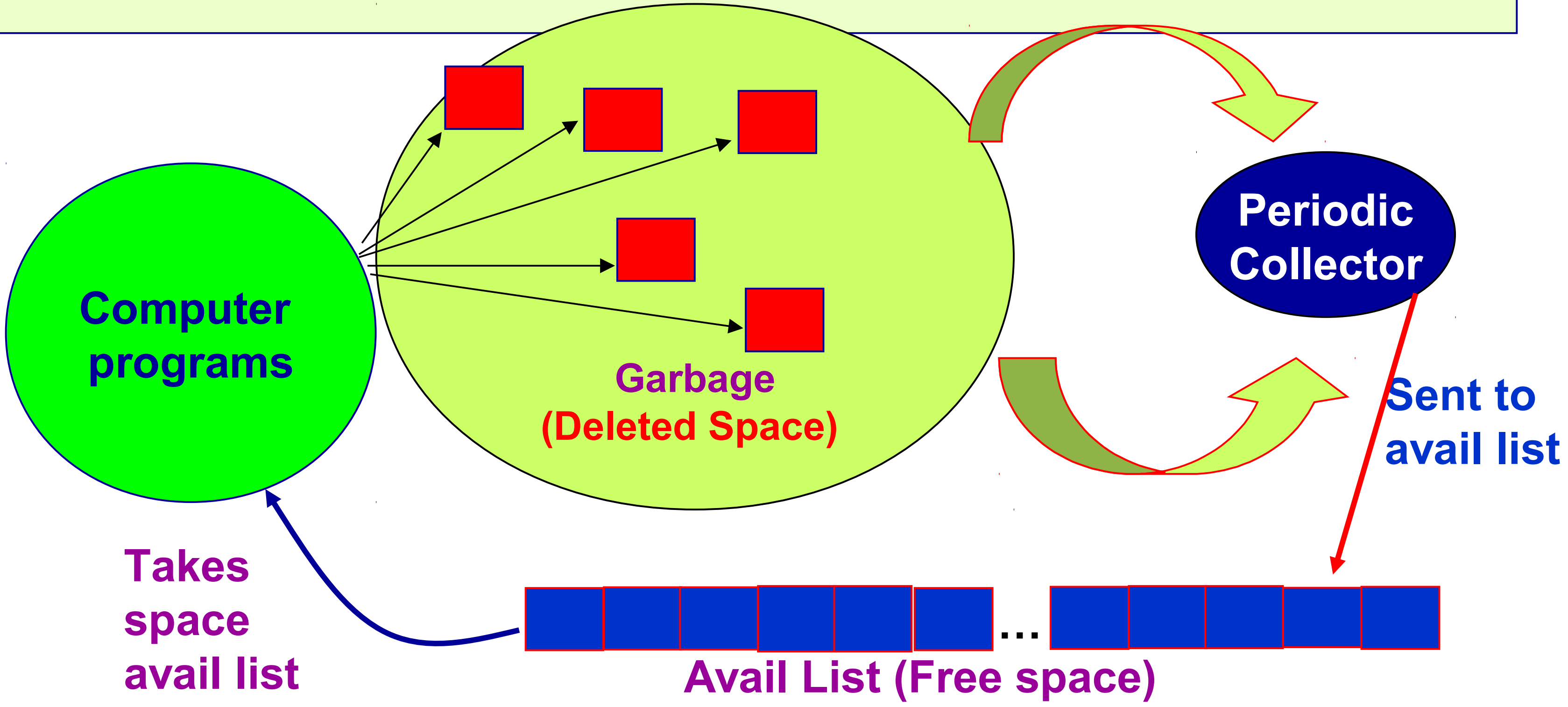
For Finding the number NUM of elements in a linked list, must traverse the list.

Algorithm 5.1 : COUNT(INFO, LINK, START, NUM)

```
1. Set NUM: =0.
2. . Set PTR : =START.
3. Repeat steps 4 and 5 while PTR : ≠ NULL:
4. Set NUM : =NUM+1.
5. Set PTR : =LINK[PTR].
6. Exit.
```

Memory allocation: Garbage collection

- Memory space can be reused if a node is deleted from a list
  - i.e deleted node can be made available for future use
- The operating system of a computer may periodically collect all the deleted space on to the free storage list. Any technique which does this collection called garbage collection



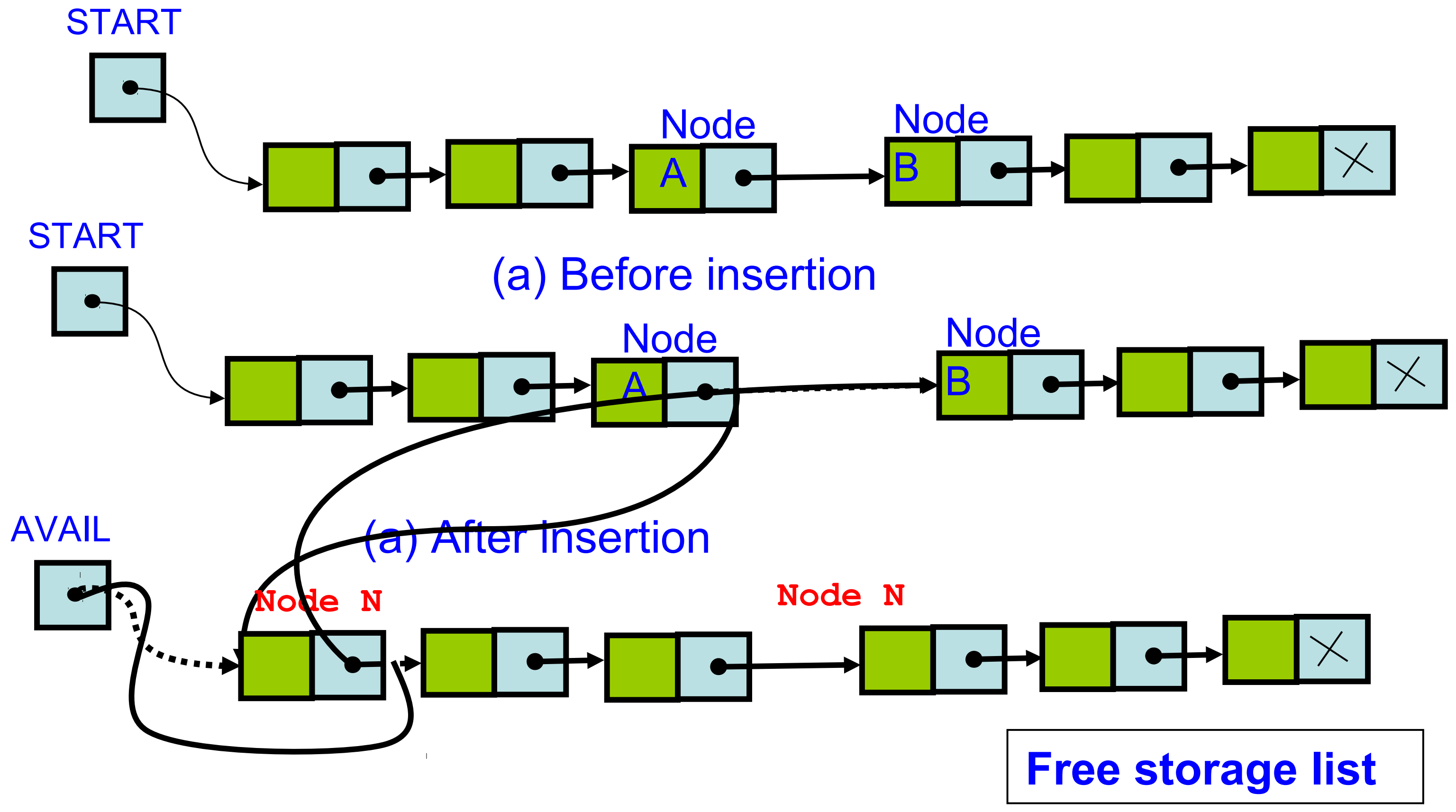
Overflow and Underflow

- **Overflow**:
  - Sometimes data are inserted into a data structure but there is no available space.
  - This situation is called overflow
  - Example: In linked list overflow occurs when
    - AVAIL= NULL and
    - There is an insertion operation
- **Underflow**:
  - Situation:
    - Want to delete data from data structure that is empty.
  - Example: In linked list overflow occurs when
    - START = NULL and
    - There is an deletion operation

Insertion into a linked list

- Node N is to be inserted in to the list between nodes A and B
- Three pointer fields are changed as follows:

1. The next pointer field of node A now points to the new node N, to which AVAIL previously pointed.
2. AVAIL now point to the second node in the free pool, to which node N previously pointed.
3. The next pointer field of node N now points to node B, to which node A previously pointed.



Inserting a new node

- Possible cases of Insert Node

  1. Insert into an empty list
  2. Insert in front
  3. Insert at back
  4. Insert in middle

- But, in fact, only need to handle two cases:

  1. Insert as the first node (Case 1 and Case 2)
  2. Insert in the middle or at the end of the list (Case 3 and Case 4)

Inserting a new node

```
INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

1.  [OVERFLOW?] If AVAIL=NULL, then print OVERFLOW and exit

2.  Set NEW= AVAIL and AVAIL=LINK[AVAIL]

3.  Set INFO[NEW]= ITEM

4.  IF LOC = NULL then [Insert as first Node]
    Set LINK[NEW]= START and START=NEW.

  Else: [Insert after node with location LOC]
    Set LINK[NEW]= LINK [LOC] and LINK[LOC]= NEW

5.  Exit.
```

Check for available memory

Inserting a new node

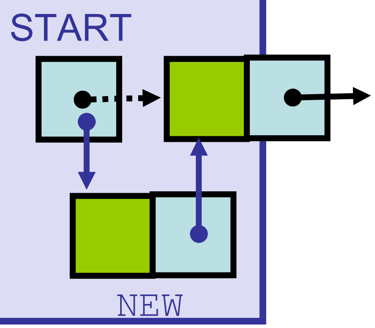
```
INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

1. [OVERFLOW?] If AVAIL=NULL, then print OVERFLOW and exit

2. Set NEW= AVAIL and AVAIL=LINK[AVAIL]

3. Set INFO[NEW]= ITEM

4. IF LOC = NULL then [Insert as first Node]
   Set LINK[]= START and START=NEW.

  Else: [Insert after node with location LOC]
   Set LINK[NEW]= LINK [LOC] and LINK[LOC]= NEW

5. Exit.
```

Create a new node

Inserting a new node

```
INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

1.  [OVERFLOW?] If AVAIL=NULL, then print OVERFLOW and exit

2.  Set NEW= AVAIL and AVAIL=LINK[AVAIL]

3.  Set INFO[NEW]= ITEM

4.  IF LOC = NULL then [Insert as first Node]
            Set LINK[NEW]= START and START=NEW.

    Else: [Insert after node with location LOC]
            Set LINK[NEW]= LINK [LOC] and LINK[LOC]= NEW

5.  Exit.
```

Insert as first element



Inserting a new node

```
INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

1. [OVERFLOW?] If AVAIL=NULL, then print OVERFLOW and exit

2. Set NEW= AVAIL and AVAIL=LINK[AVAIL]

3. Set INFO[NEW]= ITEM

4. IF LOC = NULL then [Insert as first Node]
        Set LINK[NEW]= START and START=NEW.

   Else: [Insert after node with location LOC]
        Set LINK[NEW]= LINK [LOC] and LINK[LOC]= NEW

5. Exit.
```

Insert after currNode