Linked Lists

Agenda

Dynamic Sets

Operations on Dynamic Sets

Representation of Dynamic sets

Linked List - Introduction

Linked List Operations

Dynamic Sets

- Sets are fundamental to **Mathematics & Computer Science(CS)**
- **Mathematics** - sets are unchanging
- CS - sets are manipulated by algorithms
- Sets can
  - grow
  - shrink
  - change their size over time
- Such sets are called **Dynamic Sets**

Dictionary

- **Operations on dynamic sets**
  - Insert elements
  - Delete elements
  - Test membership

- **Dictionary -** Dynamic set that supports these operations

Totally Ordered Set

- **Examples:** Real numbers/natural numbers, alphabetic order of names
- Satisfies **Trichotomy property:** For any two elements **a and b** in the Totally Ordered Set, exactly one of the following must hold:
  - $a < b$
  - $a = b$
  - $a > b$
- **Totally ordered set**
  - Minimum/Maximum element of the set
  - Next element larger than a given element

Operations on Dynamic Sets

Grouped into **two categories**:

- **Queries**: return information on the sets

- **Modifying operations**: change the set

Depending on the application only few operations needed

Typical Queries on Dynamic Sets

- **SEARCH(S, k)**

  Input: A totally ordered set *S* and a key value *k*

  Output: Returns

  – a pointer *x* to an element in *S* such that *x.key* = *k*

  – NIL if no such element belongs to S

- **MINIMUM(S):**

  Input: A totally ordered set *S*

  Output:

  – Returns a pointer to the element of S with the smallest key

Typical Queries on dynamic sets

- **MAXIMUM(S)**

  Input: A totally ordered set *S*

  Output:

  – Returns a pointer to the element of S with the largest key

- **SUCCESSOR(S, x)**

  Input: An element *x* whose key is from a totally ordered set *S*

  Output: Returns

  – a pointer to the next larger element in *S*

  – NIL if *x* is the maximum element in *S*

Typical Queries on dynamic sets

- **PREDECESSOR(S,x)**

  Input: An element *x* whose key is from a totally ordered set *S*

  Output: Returns
  - a pointer to the next smaller element in S
  - NIL if x is the minimum element in S

- **SUCCESSOR(S, x) and PREDECESSOR(S,x)** are extended to sets with non-distinct keys

Modifying operations on Dynamic Sets

- **INSERT(S,x)**

  Input: A totally ordered set *S* and a pointer to *x* (Assume that an attribute of x is already initialized)

  Output:

  – Augments S with the element pointed by *x*

- **DELETE(S,x)**

  Input: A totally ordered set *S* and a pointer to *x* (not a key value)

  Output:

  – Removes x from S

Measuring Running time for the operations on Dynamic Sets

How do we measure the time taken to execute

a set operation?

In terms of the size of the set

Elementary Data Structures

- Representation of dynamic sets by simple data structures that use pointers :
  - Linked Lists
  - Stacks
  - Queues
  - Rooted Trees

Dynamic Sets

- Each element is represented by an object
  - A pointer to the object is used for examining and manipulating the objects' attributes

- Dynamic sets assume, one of the objects attribute is an identifying **key**

Dynamic Sets

- The object may contain **satellite data**, which are carried around in other object attributes
- **Object attributes** - manipulated by set operations
  - Attributes may contain data or pointers to other objects in the set
- Some dynamic sets - **keys from a totally ordered set**

Linear Data Structures

Array - order determined by the indices

Advantage

Disadvantage

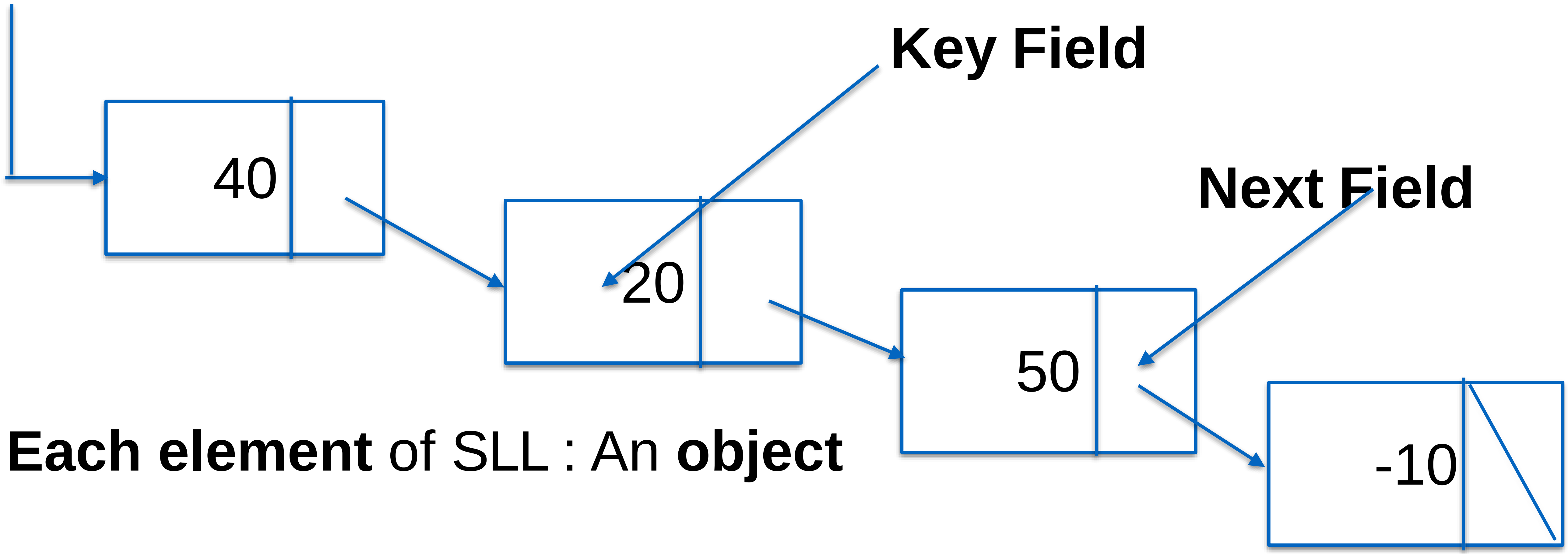
Linked Lists

- **Linked list** is a data structure in which objects are arranged in a linear order
  - Linear order is determined by a pointer in each object

- Provides a **simple, flexible representation for dynamic sets** and it supports all the operations (query & modifications)

- **Different types of linked list**:
  - Singly Linked List (SLL)
  - Doubly Linked List (DLL)
  - Circular Linked List (CLL)

SINGLY LINKED LIST (SLL)

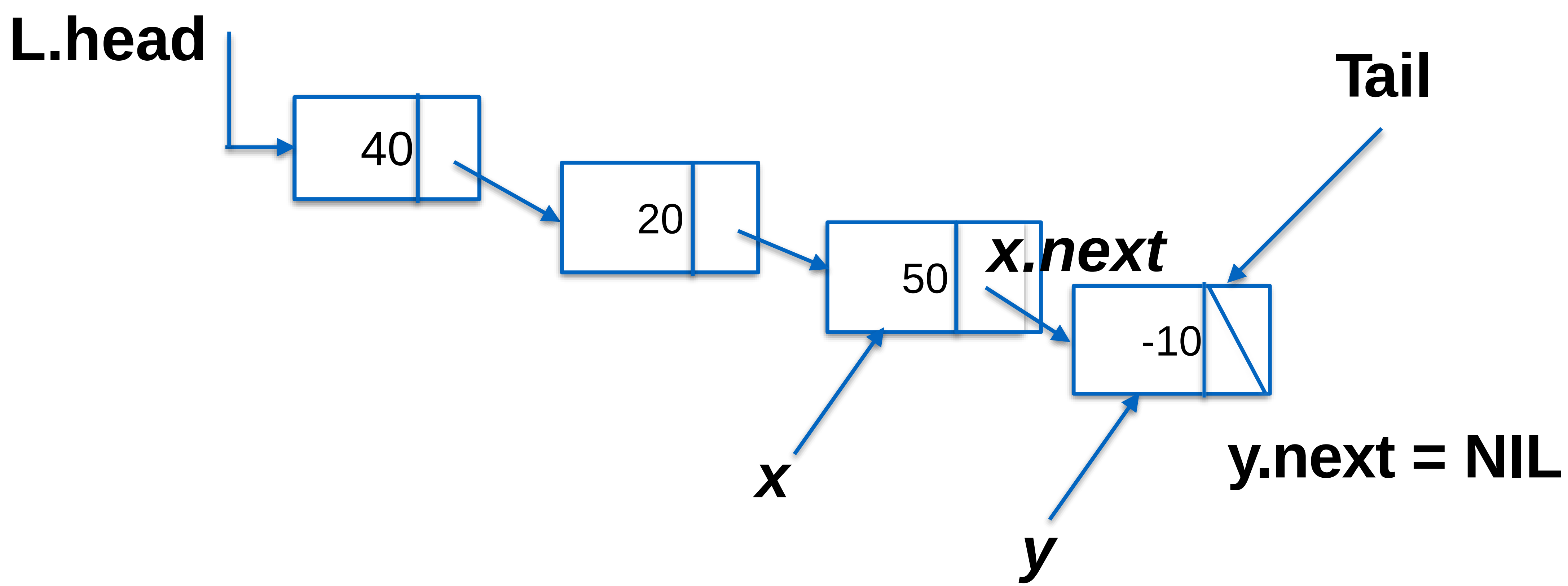


Each element of SLL : An **object**

Attributes: Key and a Next pointer

Object may also contain other **satellite data**

SINGLY LINKED LIST (SLL)



- An attribute **L.head** points to the **first element** of the list.
  If **L.head = NIL, the list is empty**
- Given an element *x* in the list, ***x.next*** points to its **successor** in the linked list
- If **x.next = NIL**, the element x has **no successor** and is therefore the last element, or **tail**, of the list.

Types of Linked List

Sorted List - If a **list is sorted**, the **linear order** of the list corresponds to the **linear order of keys** stored in elements of the list

– **Minimum** element: head of the list

– **Maximum** element: tail of the list

Unsorted List - If the list is **unsorted**, the elements can **appear in any order**.

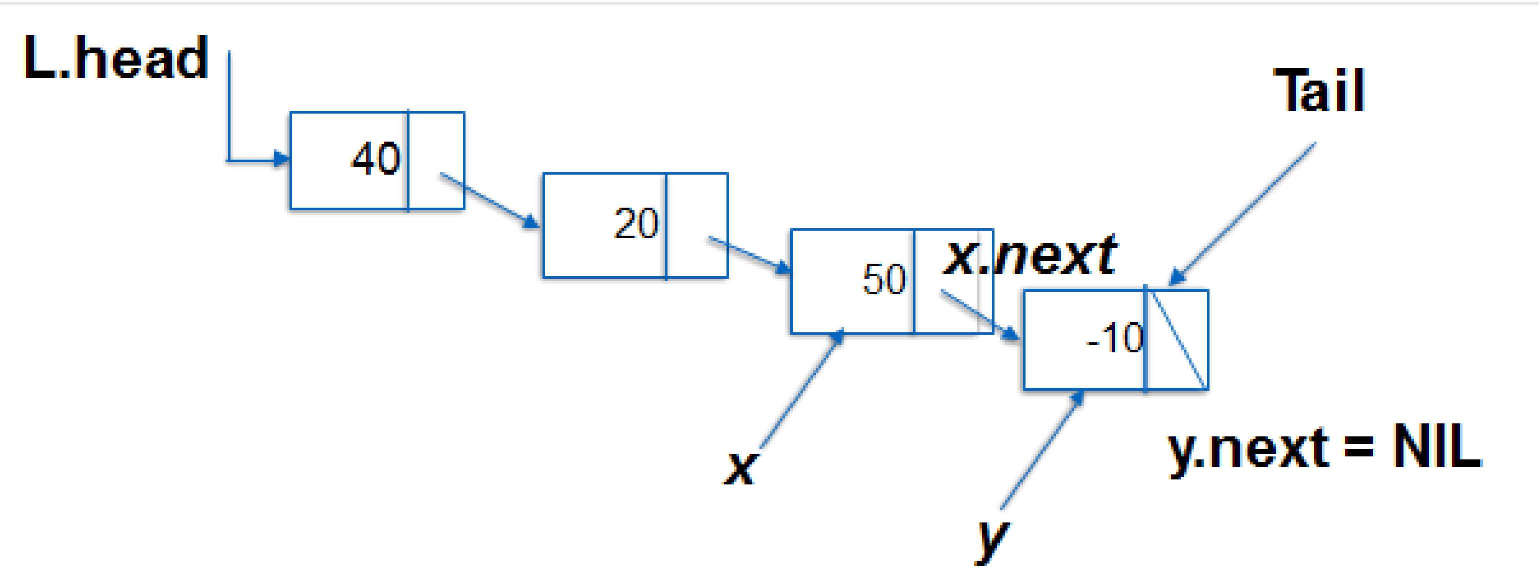
Search Operation

The procedure **LIST-SEARCH (L,k)** finds the first element with **key k** in **list L** by a simple linear search, returning a pointer to this element.

If **no object with key k** appears in the list, then the procedure **returns NIL**.

LIST-SEARCH (L,k)

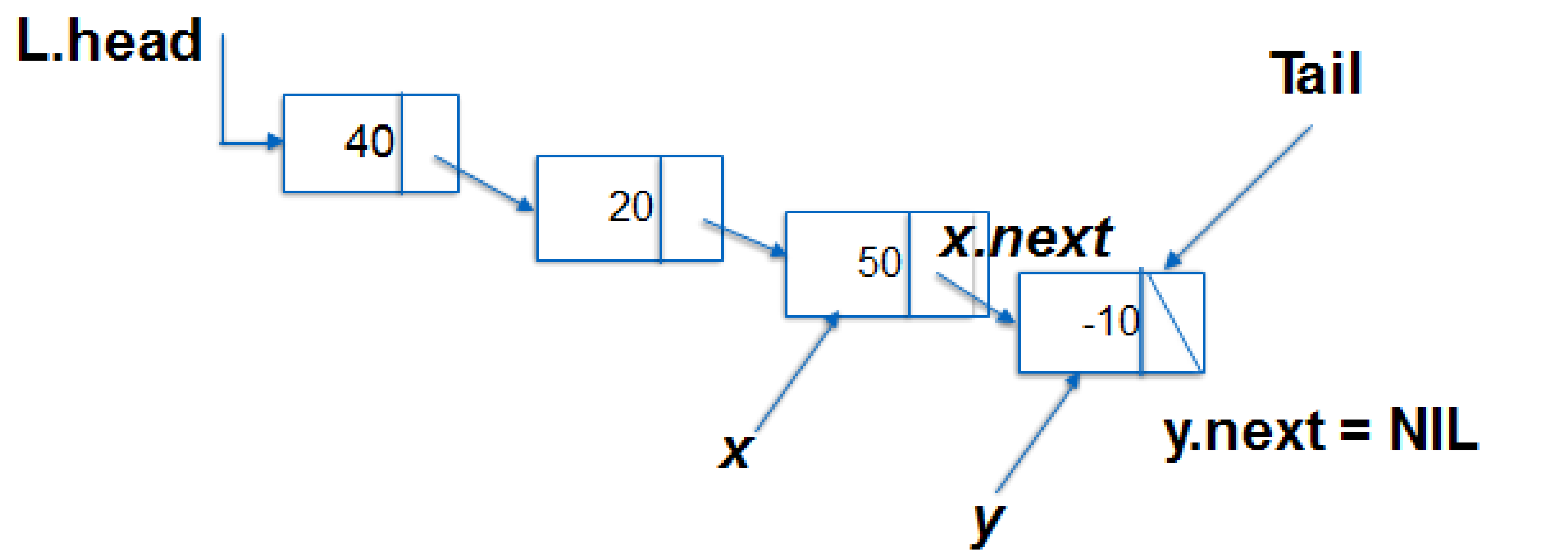
```
LIST-SEARCH(L, k)
    x = L.head
    while x ≠ NIL and x.key ≠ k
        x = x.next
    return x
```



LIST-SEARCH (L,k)

Consider a SLL with **n objects**, What is the running time of LIST-SEARCH ?

- Best Case
- Worst Case
- Average Case

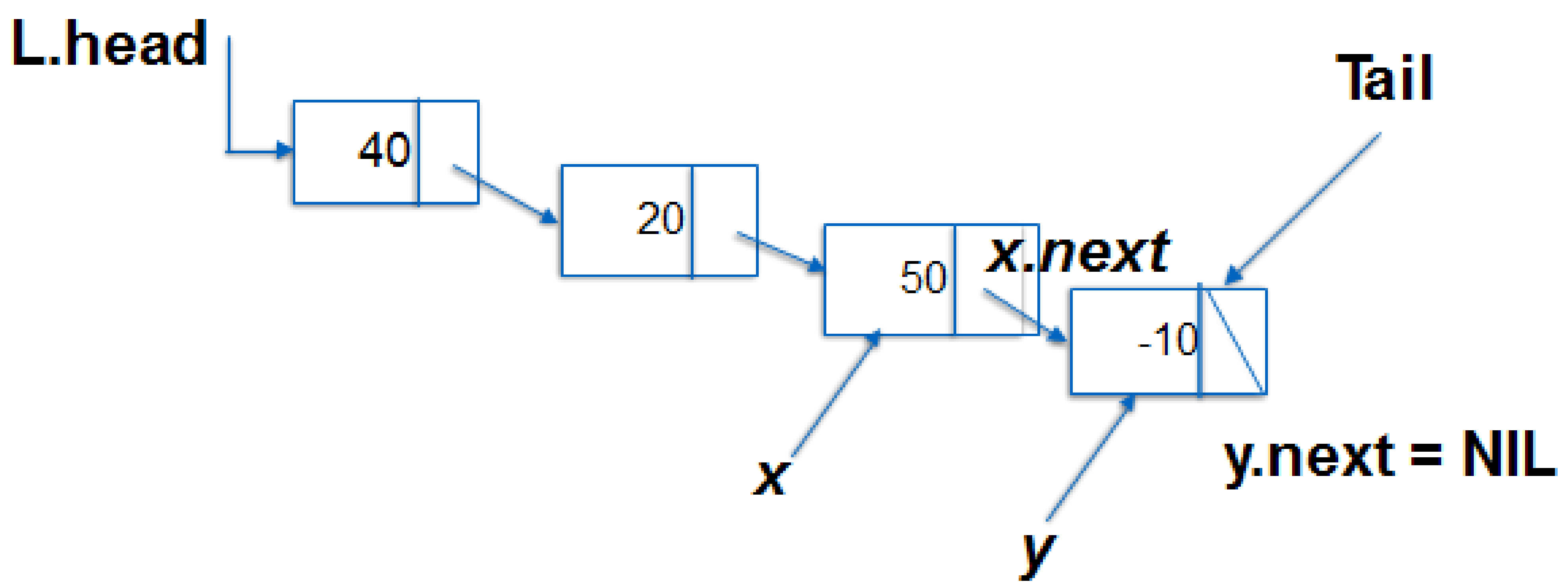


Insertion of a node in a linked list

1. Insertion at the beginning of the list.

2. Insertion at the end of the list

3. Inserting a new node except the above-mentioned positions.



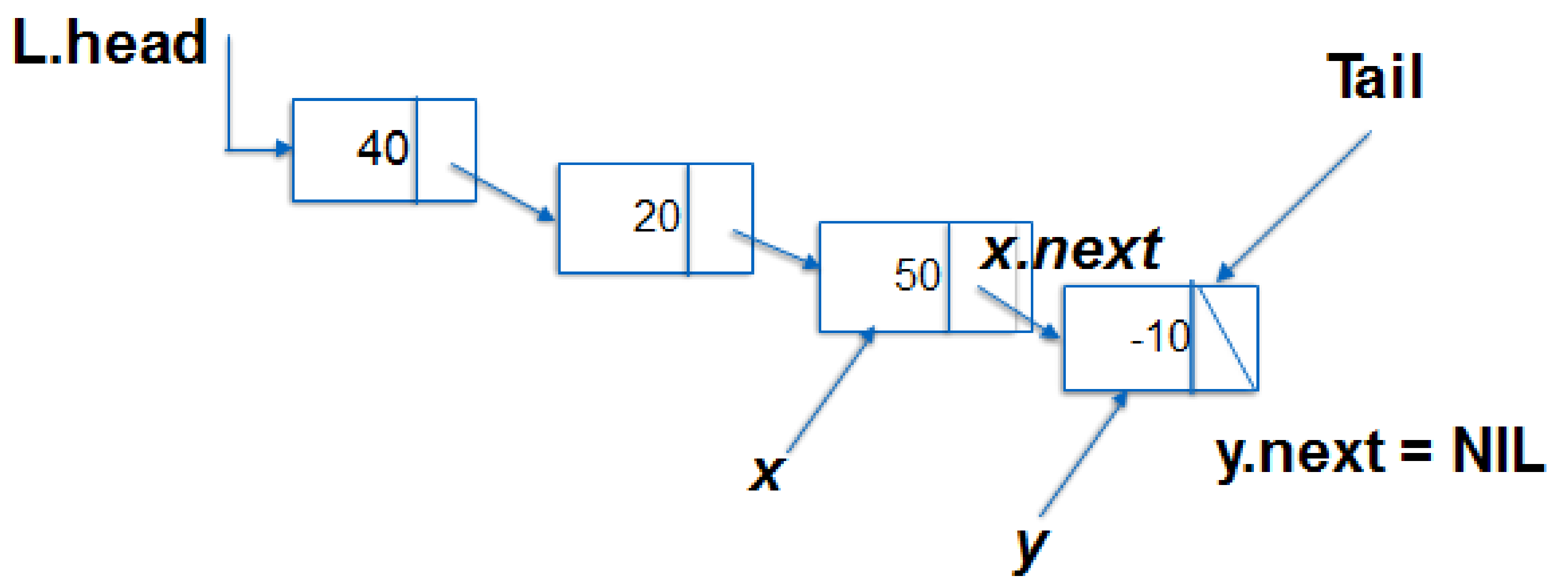
LIST-INSERT (L,x)

Steps to insert the element x in the front of the SLL:

1. x.next = L.head

2. L.head = x

What is the running time to insert the element in the front of the list?



"The best way to learn a new programming language is by writing programs in it."

- Dennis Ritchie