Lists

Linear list

a sequence of elements that are in a certain order

- each element has a position
- element order in list is discussed based on position

A linear list is called ***circular*** if it is considered that the predecessor of the first node is the last node and last node successor is the first node

List and Position

Position extended

- valid location of an element
- $\perp$

Order in list

➔ positions

a) (sorted) Position values : ex. indexes are Integers

b) Ordering is kept by the association to each element of link(s) to the element(s) preceding(/ following) it

← Node

Remark: Position values - not sorted

Indexed Lists

Choices for position: index

ADT: List(Index; TE)= IndexedList(TE)

$\mathcal{D}_{\text{List}}$(Index; TE)

Index = {i | i $\in$ **N** }

different internal representations:

- array representation
- linked representation

Linked Lists

order is determined by positions informations stored with each element

Node:

- elements are stored in nodes
- each node know the position of the next (/previous) node in the list

Informations to store in a node (choices)

- one link ➔ singly linked list
  SLNode = {***(e; n)*** | ***e***:TE and ***n*** : **position**(*next*(SLNode))}
  next, link
- two links ➔ doubly linked list
  DLNode = {(***e; n, p)*** | ***e***:TE and ***n*** : **position** *next*(DLNode))
  and ***p*** : **position** (*prev*(DLNode))
  }

singly linked list

- knowing the position of the head of the list (the first element), we can access all elements in the list

  → *list*: keeps only the position of the first element
- a position of an element in the list → sublist
- ⊥ - *empty list*

singly linked list

representations for SLL

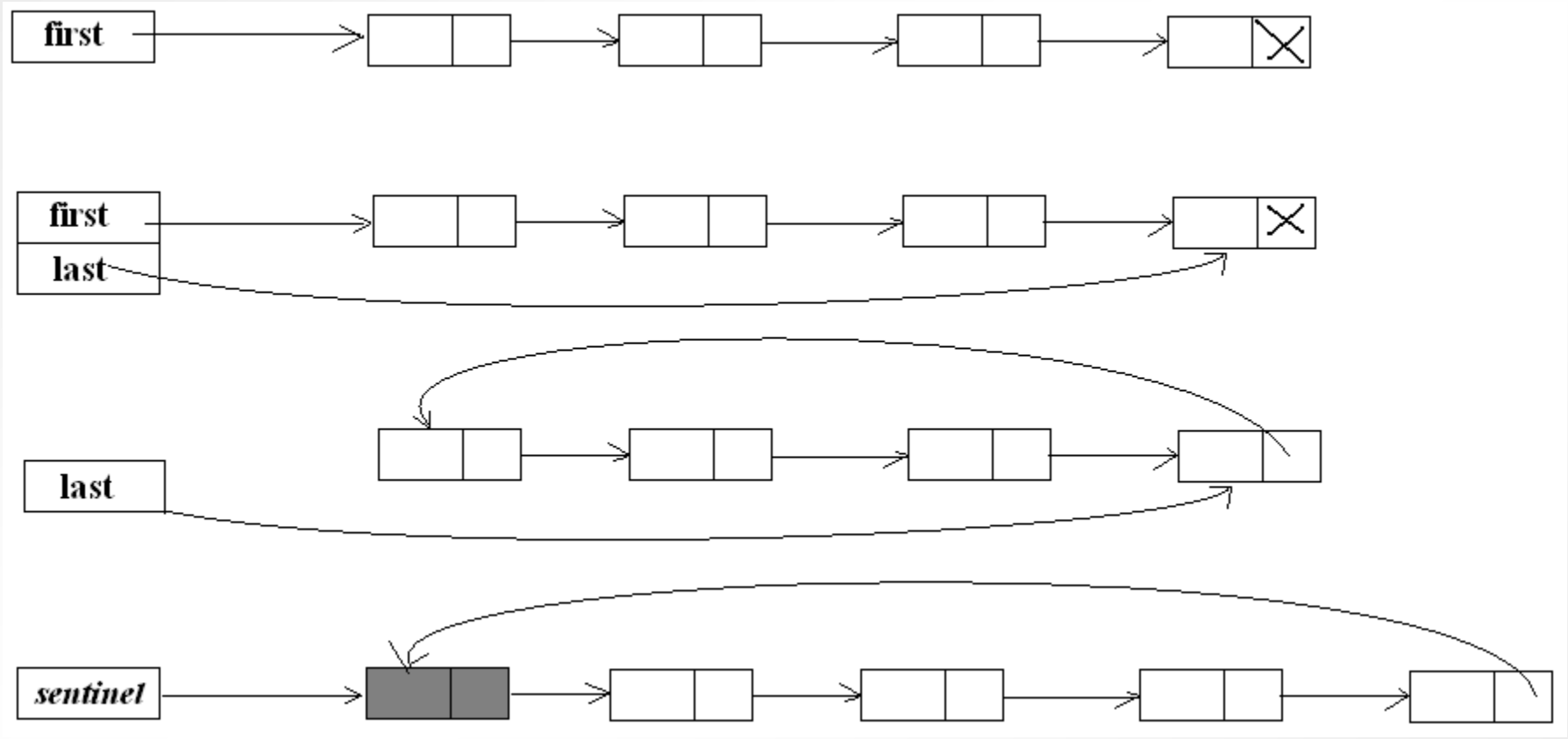
- using dynamic memory allocation
  - for each node individually

  Terminology (*often*) : dynamic linked storage
- represented over an array (semi-static)

  - *space management in the array !!*
- …. others

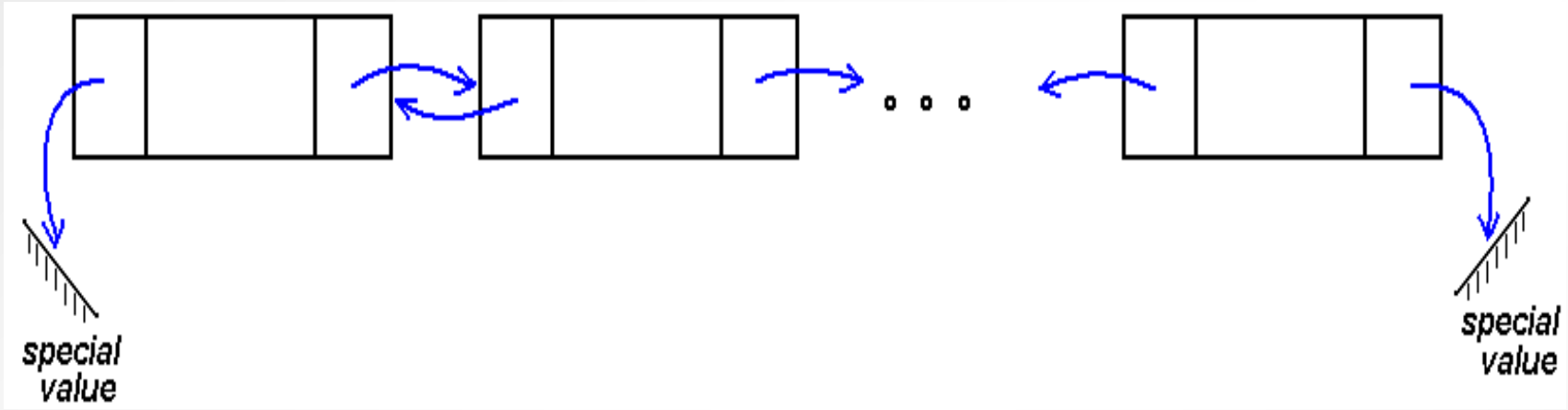
Ex.: Design operation *addAfter*

Singly-linked list representations (variations)



? Design SLList with fast addLast

Doubly-Linked list



- can access elements if we know one of the two ends
- *but*: keep Position s of both ends (usually)
  reason: access in both direction

representations for DLL

- based on dynamic memory allocation dynamic linked storage
- represented over an array
  - over a static vector (semi-static) / over a dynamic vector

Doubly-linked and circular list variations