PROGRAMMING CONCEPTS AND EMBEDDED PROGRAMMING IN C, C++ and JAVA:

Lesson-6: Data Structures: Lists

List

1) *list* difference with an *array;*
array –Array memory allocation- as per the index assigned to an element.

List

2) Each array element- at the consecutive memory addresses starting from the 0^{th} element address.

3) Each element value (or object) in an *array* is read or replaced or written by an addressing using only two values: 0^{th} element address pointer and index(ices).

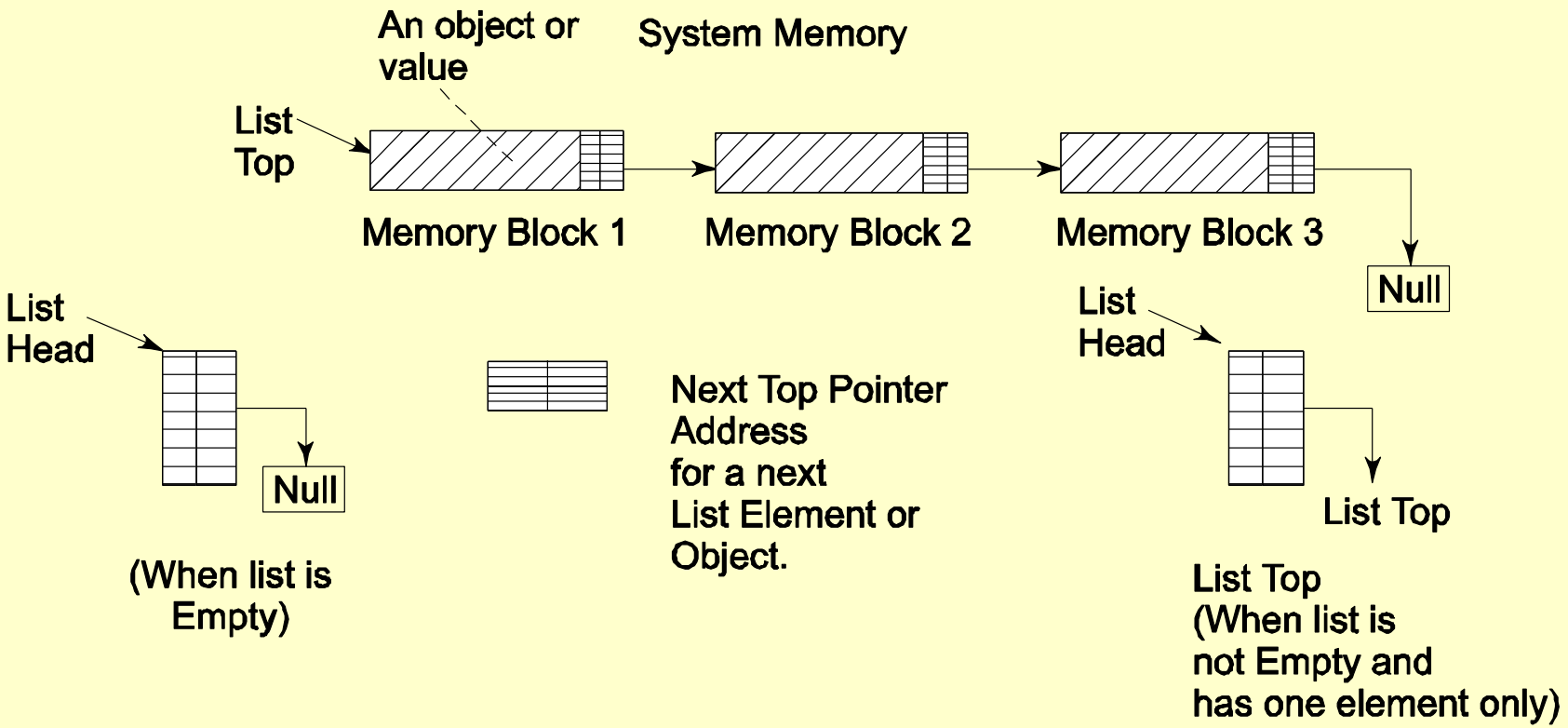
List

4) *list* each element- Must include along with an item a pointer, *LIST_NEXT*. Each element is at the memory address to which the predecessor list-element points.

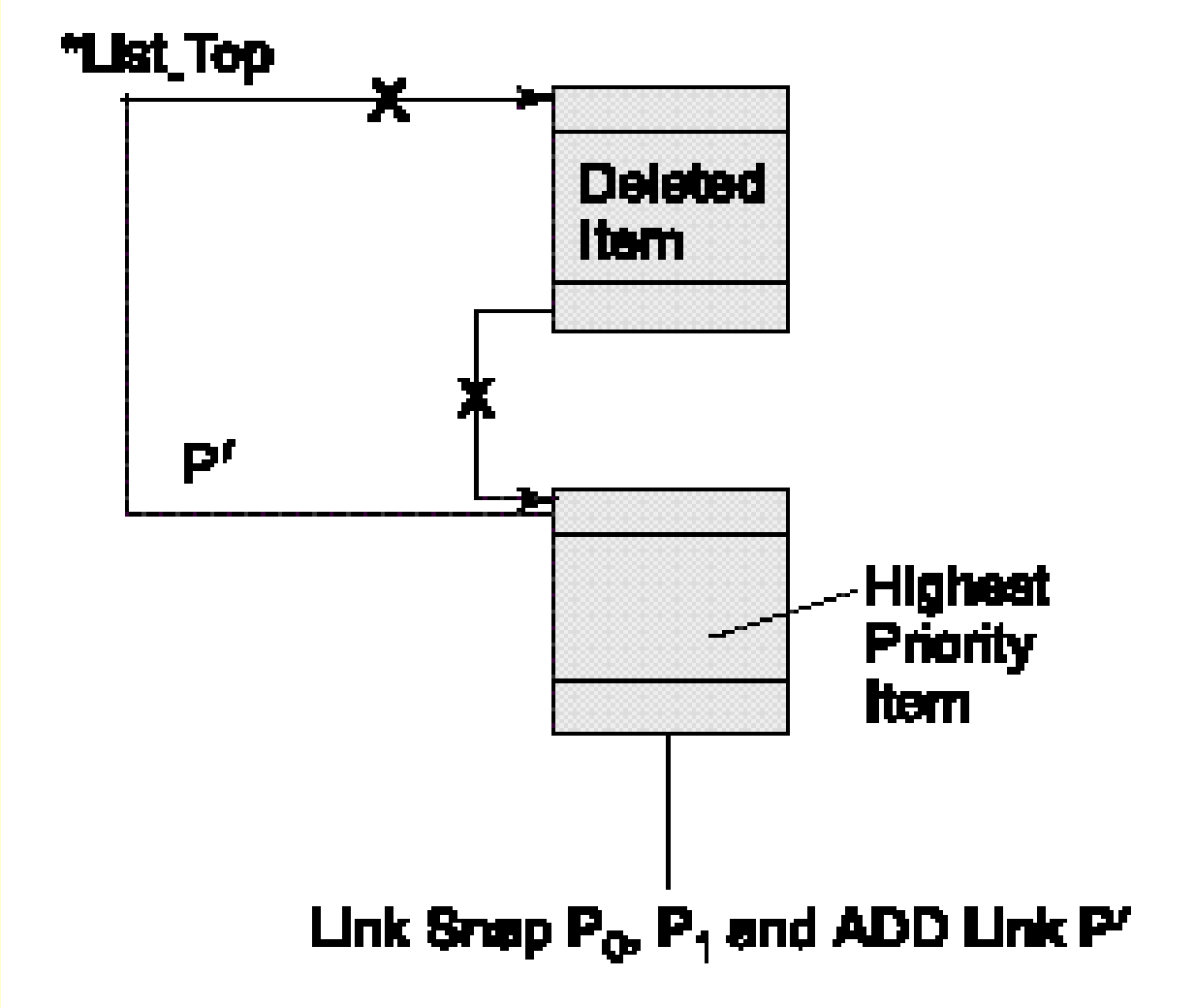
List

5) *LIST_NEXT* points to the next element in the *list*. *LIST_NEXT* points to NULL in element at end of the list.

List


An object or value
System Memory
List Top
Memory Block 1
Memory Block 2
Memory Block 3
Null
List Head
Null
(When list is Empty)
Next Top Pointer Address for a next List Element or Object.
List Head
List Top
List Top (When list is not Empty and has one element only)

A deletion into the list the first element



List

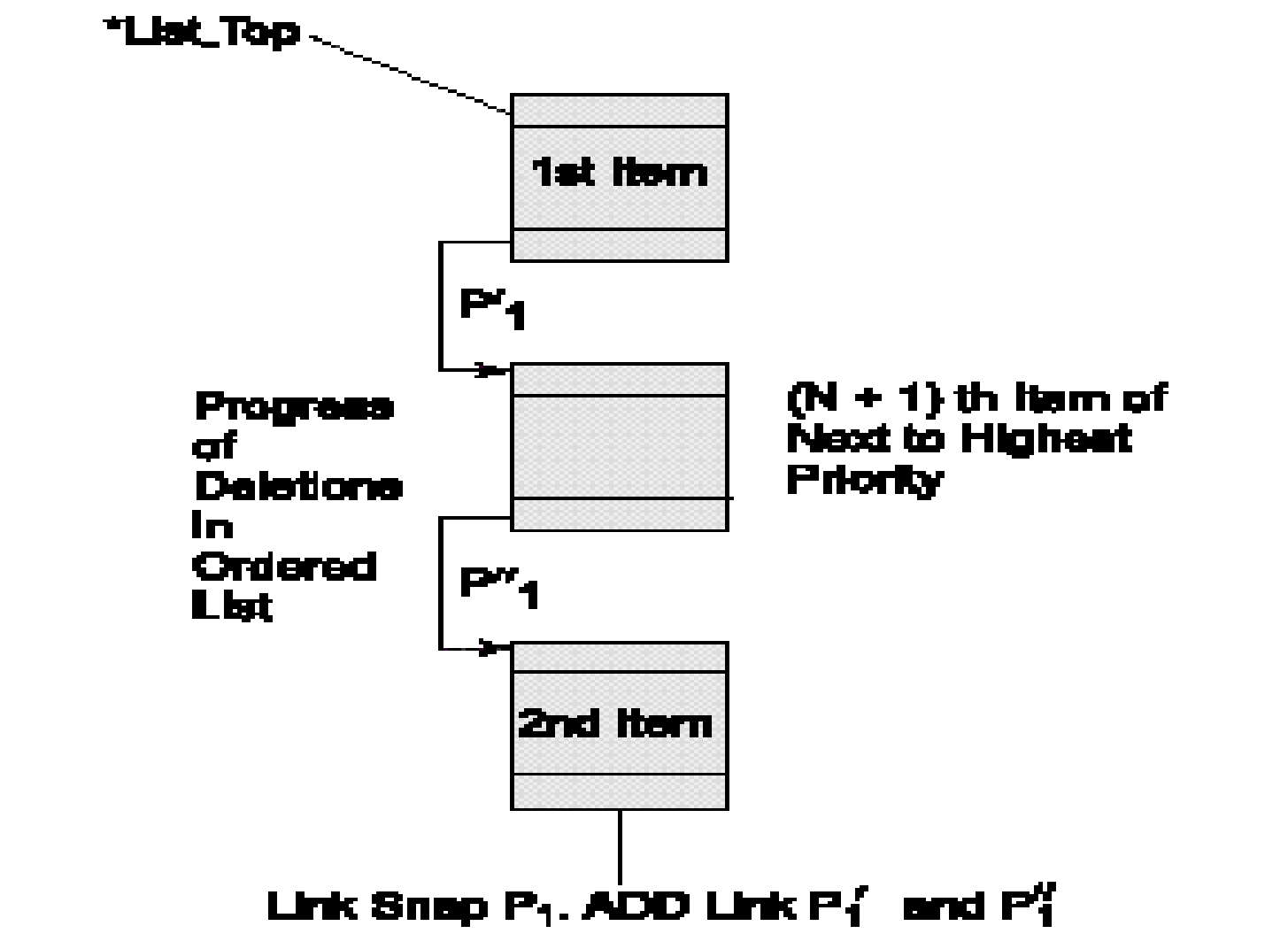
6) The memory-size of an item of an element can also vary. The address of the list element at the top is a pointer, LIST_TOP. Only by using the LIST_TOP and traversing through all the LIST_NEXT values for the preceding elements can an element be deleted or replaced or inserted between the two elements.

7) A *list* differs from a *queue* as follows: A *queue* is accessible and is readable as FIFO only.

8) An insertion of an element in the *list* can be done anywhere within it only by traversing through LIST_NEXT pointers at all the preceding elements

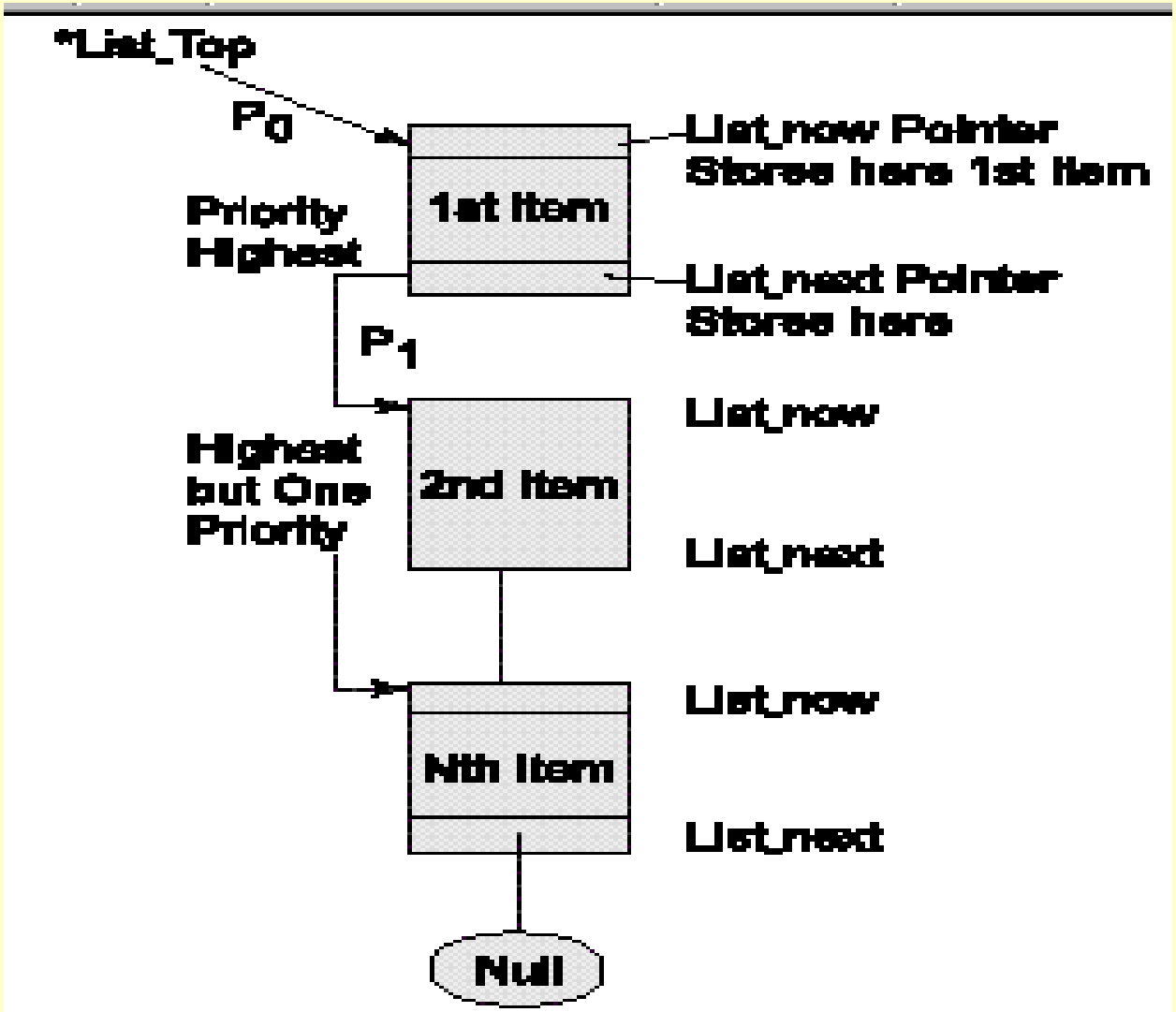
9) An insertion is always at the *tail* in *queue*. Also, an element can be read and deleted from anywhere in the *list* only by traversing through the list. It is always from the *head* in the *queue*.

An insertion into the list after the first element



10) Each element of the *ordered list* - Always as per the order of the priority assigned to its items. Priority can be in order of task execution or alphabetical order or timeout period left in case of the list of the active timers or first time entry then the next in a chosen criteria for ordering in list.

Ordered List (Priority List)



Standard Functions used in an Ordered List

1. LELInsert – Insert an element into the list as pointed by *ListNext and reallocate the LIST_NEXT pointer addresses

Standard Functions used in an Ordered List

2. LELInsertLast – Insert an element into the list with this element pointing to NULL and reallocate the LIST_NEXT pointer address of the earlier last element to this element address
3. LELInsertPrev – Insert an element into the list at the previous element address

Standard Functions used in an Ordered List

4. LELDelete – Delete an item within the list
5. LELdeleteLast – Deletes the last element, earlier last but one now points to NULL.
6. LELSearch – Search an item
7. LElprioritySearch- Search the priority of an item

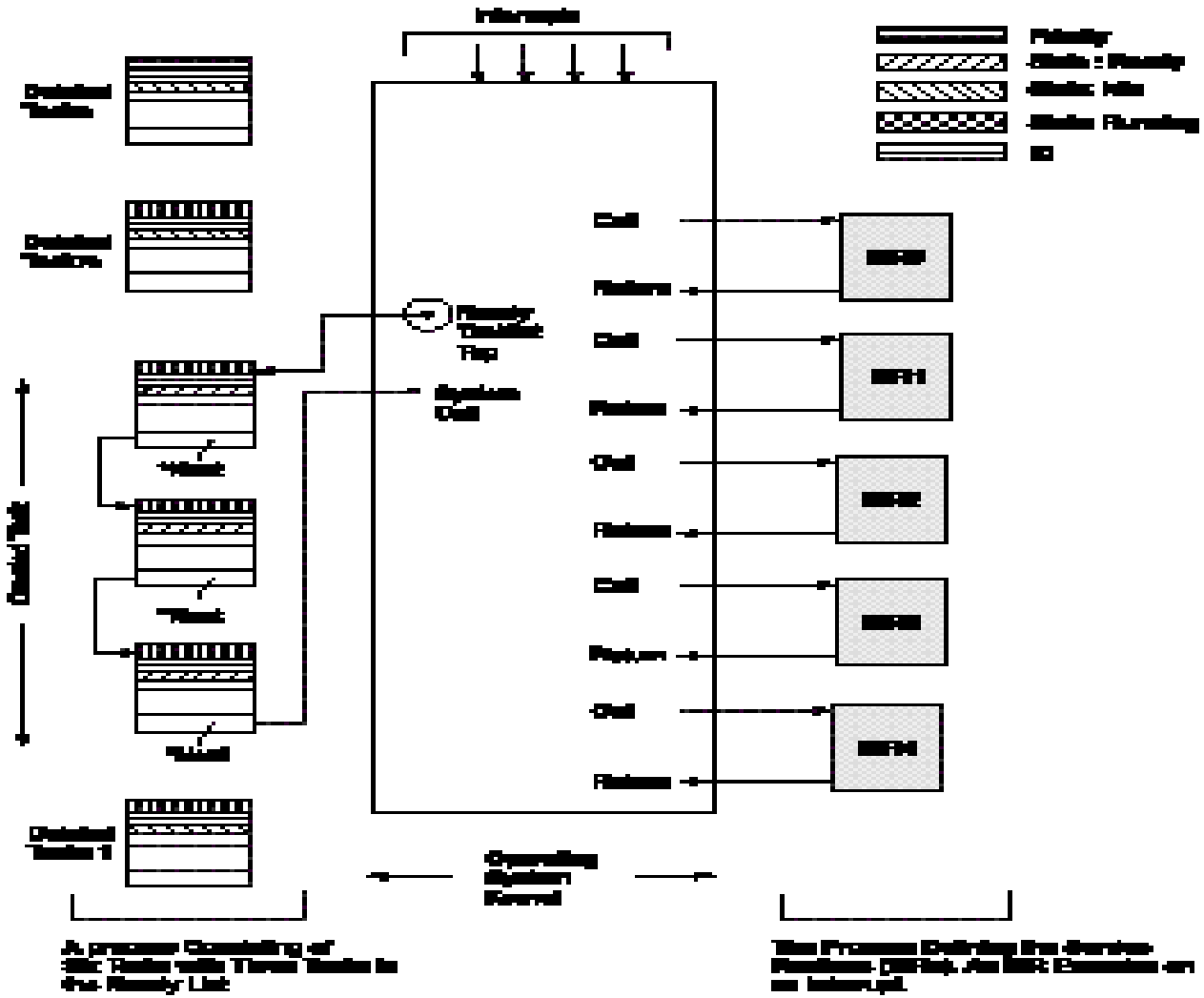
Standard Functions used in an Ordered List

8. isLNotEmpty– Return true or false after the check for the queue not empty.

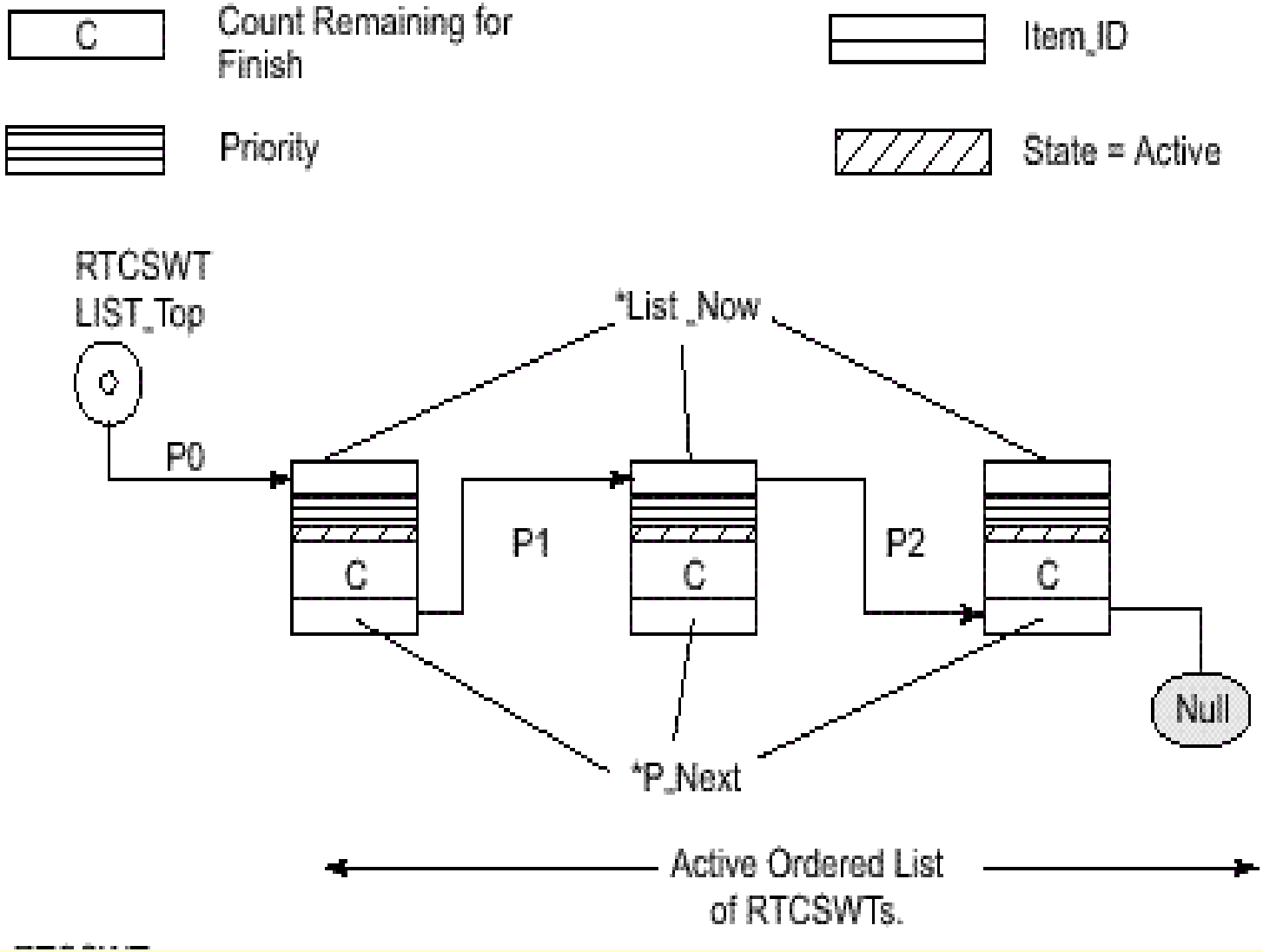
Application Example of a List

1. List of Tasks that are active (not blocked and not finished).
2. List of software timers which are not yet timed out and to which clock inputs are to be periodically given by real time clock.

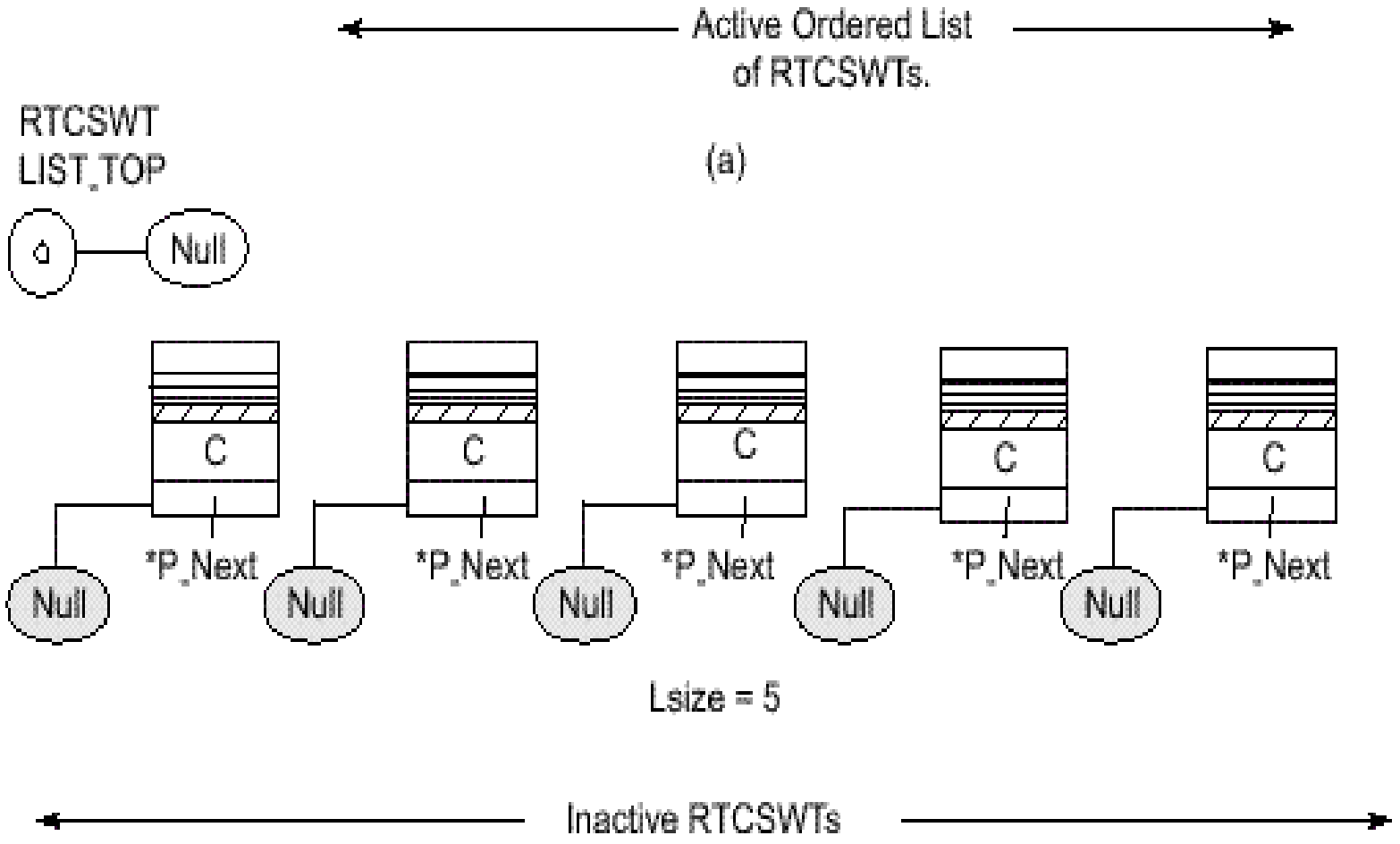
Priority List of Active Tasks at an operating system



List of software timers



Priority List of active software timers



Summary

We learnt

- List
- Priority -wise ordered list.

We learnt

- List Related functions for constructing a list
- inserting an element into it
- finding an element from it
- deleting an element from it and
- destructing of the list.

We learnt

- List data-structure differences with the array and queue structures

End of Lesson 6 of Chapter 7

on

Data Structures: Lists