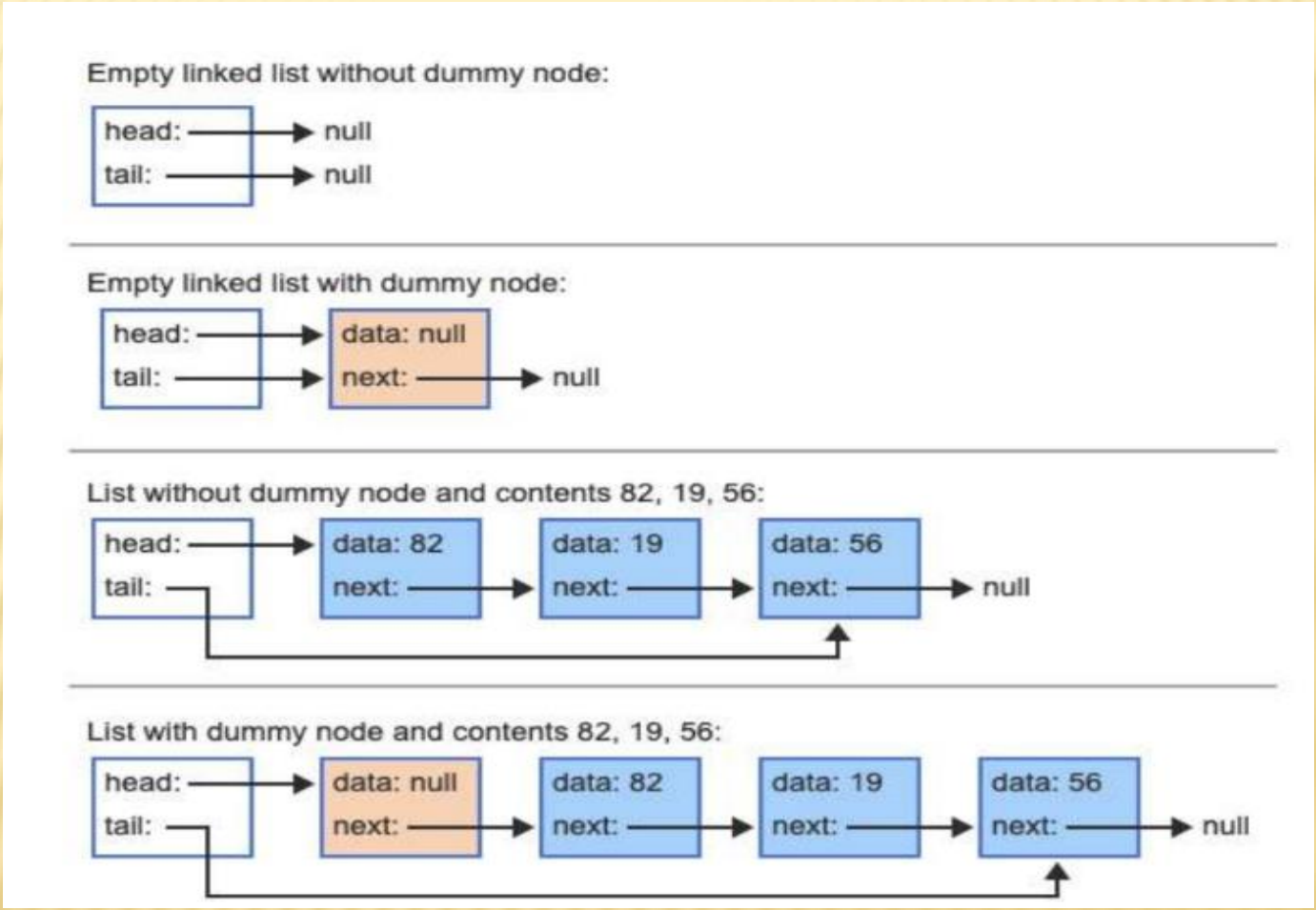
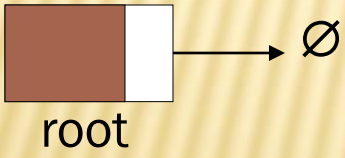
LINK LIST

LINKED-LIST EXAMPLE

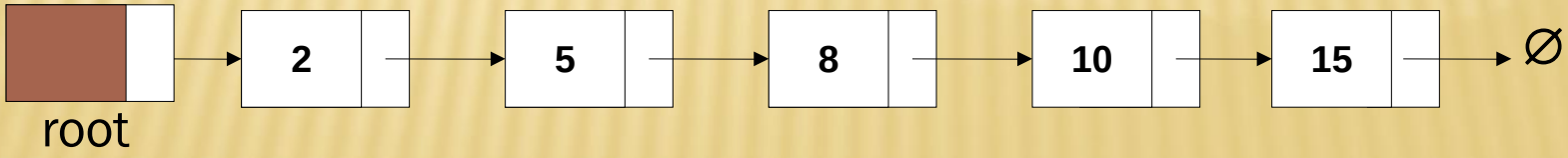


ELIMINATING BOUNDARY CONDITIONS – DUMMY HEAD NODE

```
template <class type>
class List{
    Node<type>  root;       //instead of Node<type> * root;
public:
    List();
    bool isEmpty();
    void print();
    bool insert(type info);
    void makeEmpty();
    bool Delete(type info);
    List() {root.next = NULL;};
};
```



Empty list



```
template <class type>
void List<type>::print(){
    Node <type>* current = root.next;
    while(current){
        cout<<current->data<<endl;
        current = current -> next;
    }
}
```

INSERTION

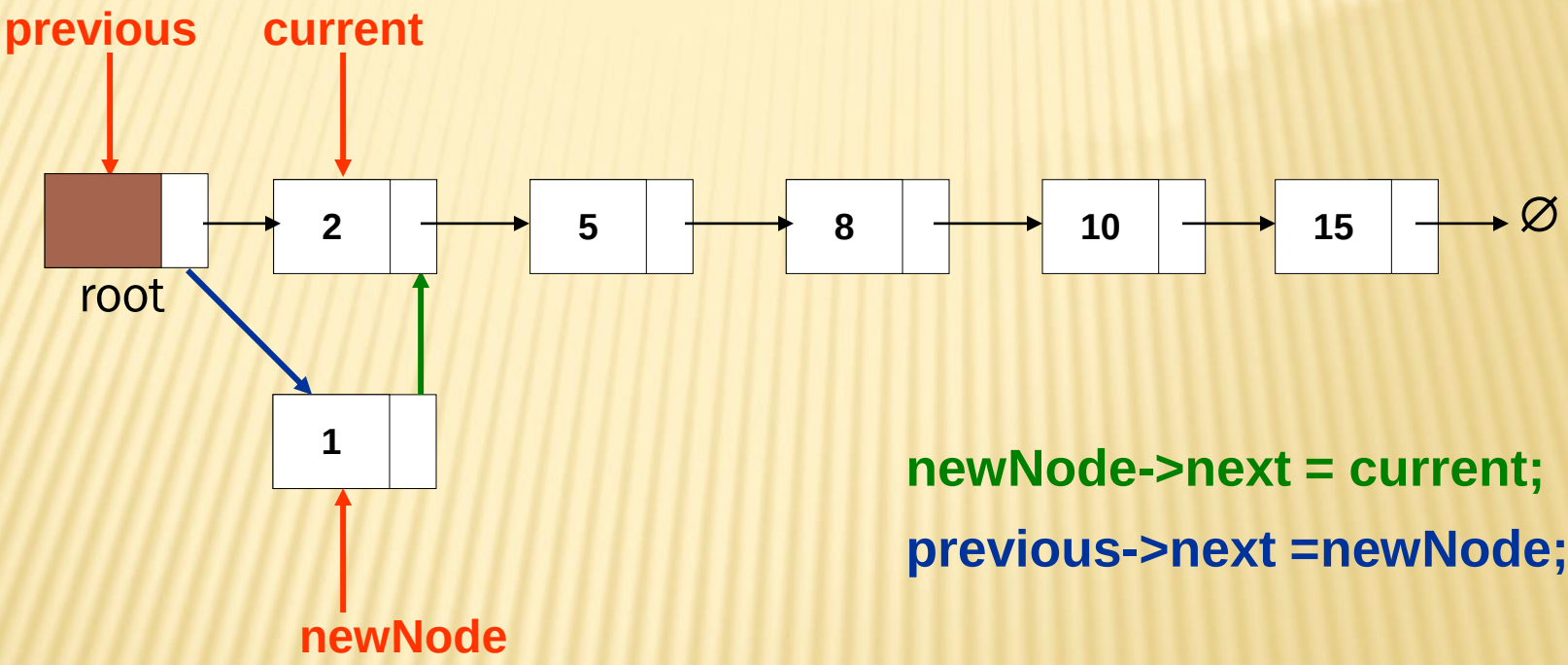
```
template <class type>
bool List<type>::insert(type info){

	if(!isFull()){
		Node<type> * newNode  = new Node<type>(info);
		Node<type> * previous = &root;
		Node<type> * current;
		current = previous->next;

		while(current && current->data < info){
			previous = current;
			current= current->next;
		}
		/*if(!previous){
			newNode->next = root;
			root = newNode;
		}
		else {*/

			newNode -> next = current;
			previous -> next = newNode;
		//}
		return true;
	}
	return false;
}
```

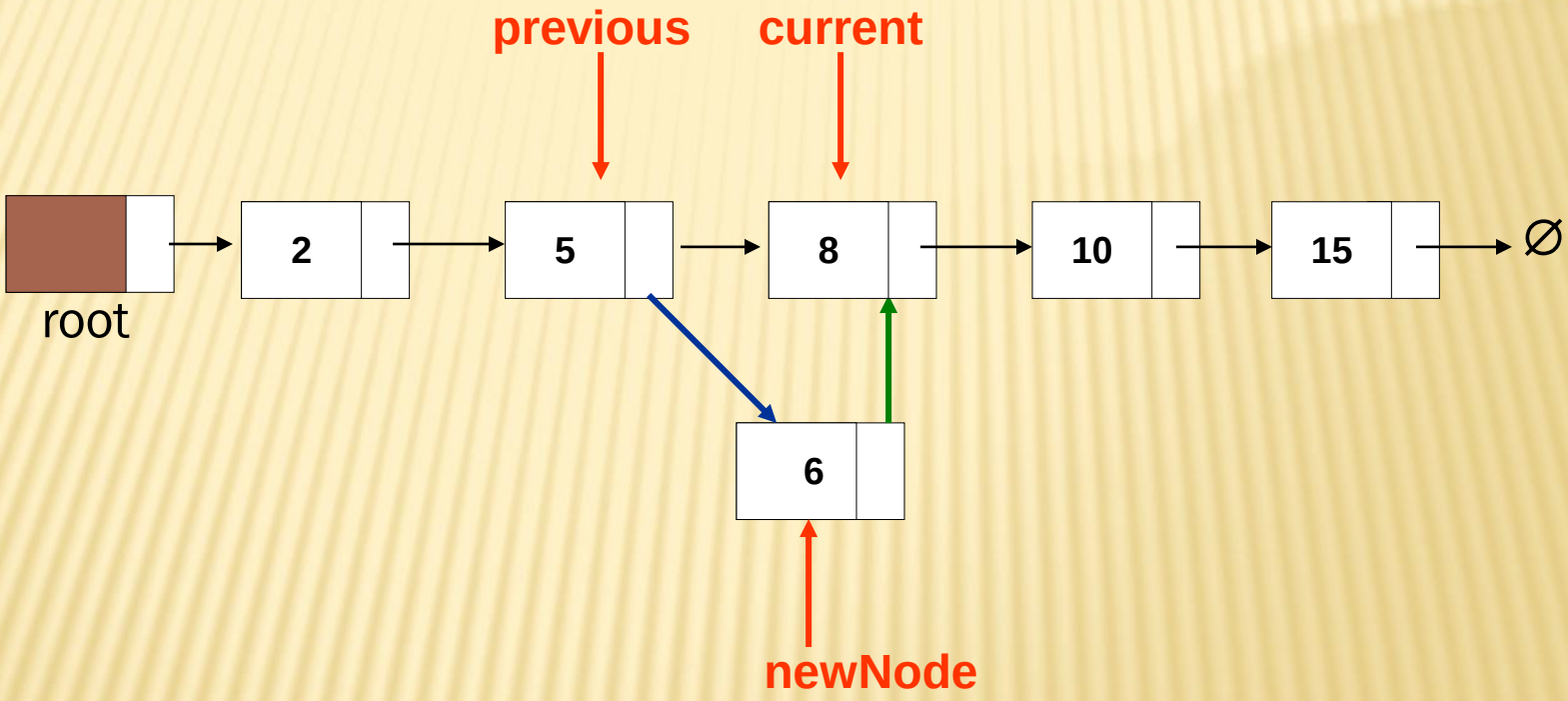
LINKED-LIST
INSERTING A NODE AT START



newNode->next = current;

previous->next =newNode;

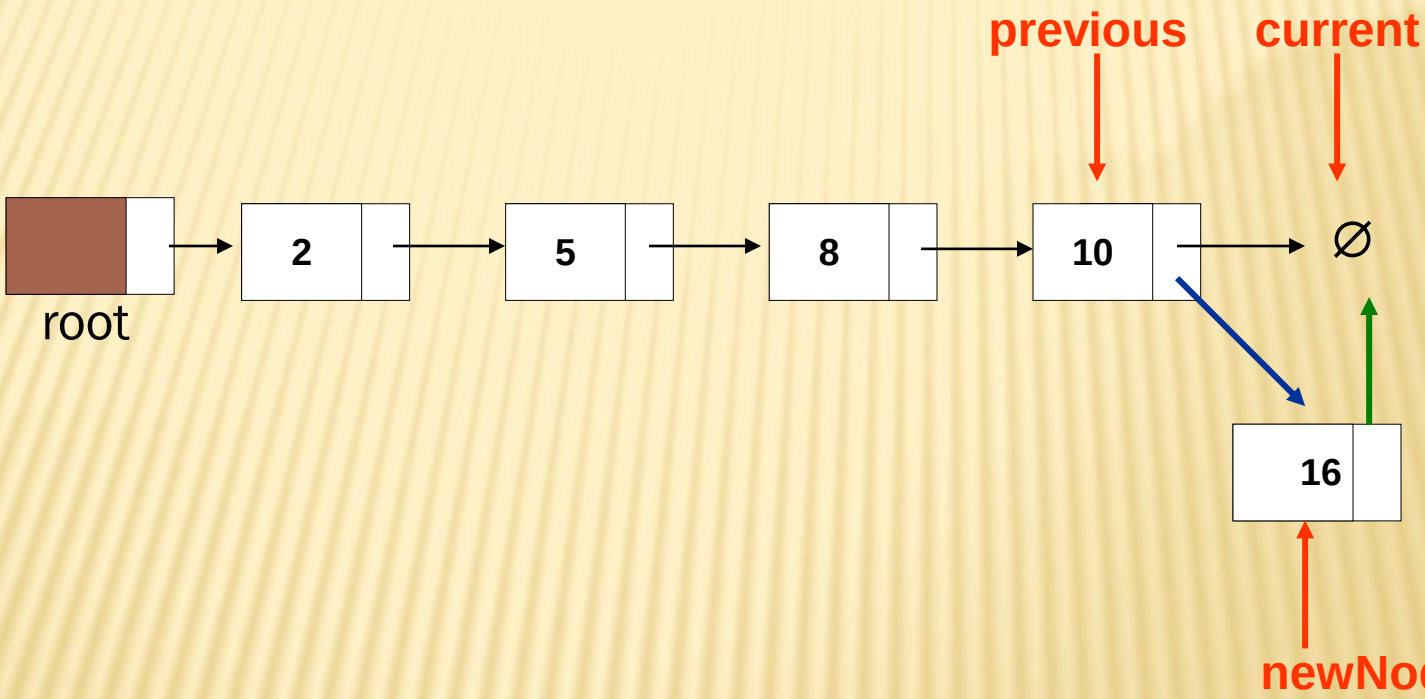
LINKED-LIST
INSERTING A NODE IN MIDDLE



newNode->next = current;

previous->next =newNode;

LINKED-LIST
INSERTING A NODE AT END



newNode->next = current;

previous->next =newNode;

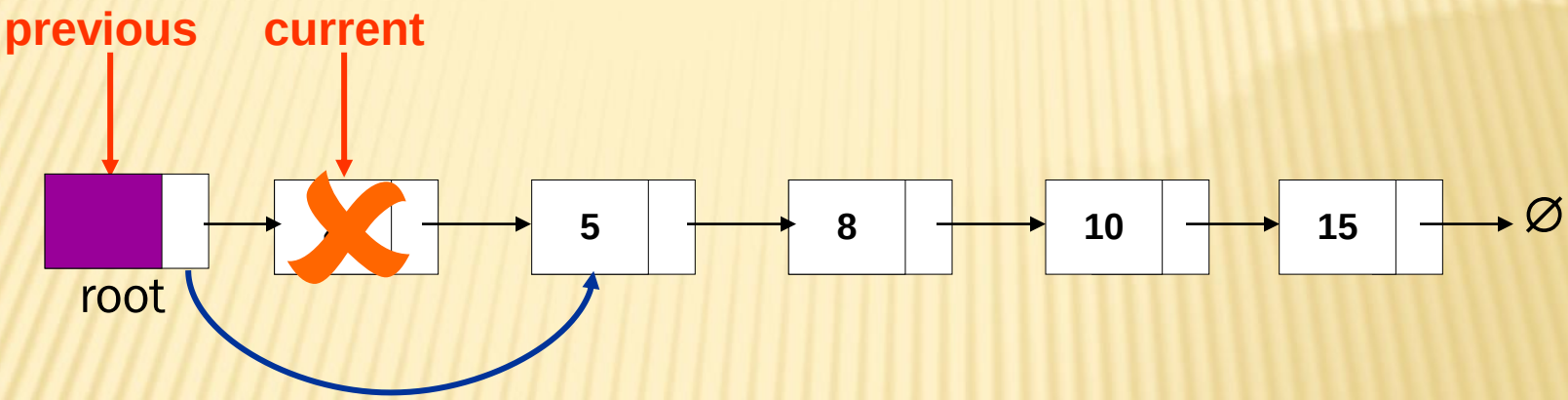
```
template <class type>
bool List<type>::Delete(type info){

	if(!isEmpty()){
		Node<type> * previous = &root;
		Node<type> * current;
		current = previous->next;
		while(current && current->data != info){
			previous = current;
			current= current->next;
		}

		/*if(!previous){
			root = current->next;
		}
		else{*/

			if(current){
				previous->next = current->next;
			}
			else
				return false;			//}
		delete current;
		return true;
	}
	return false;
}
```

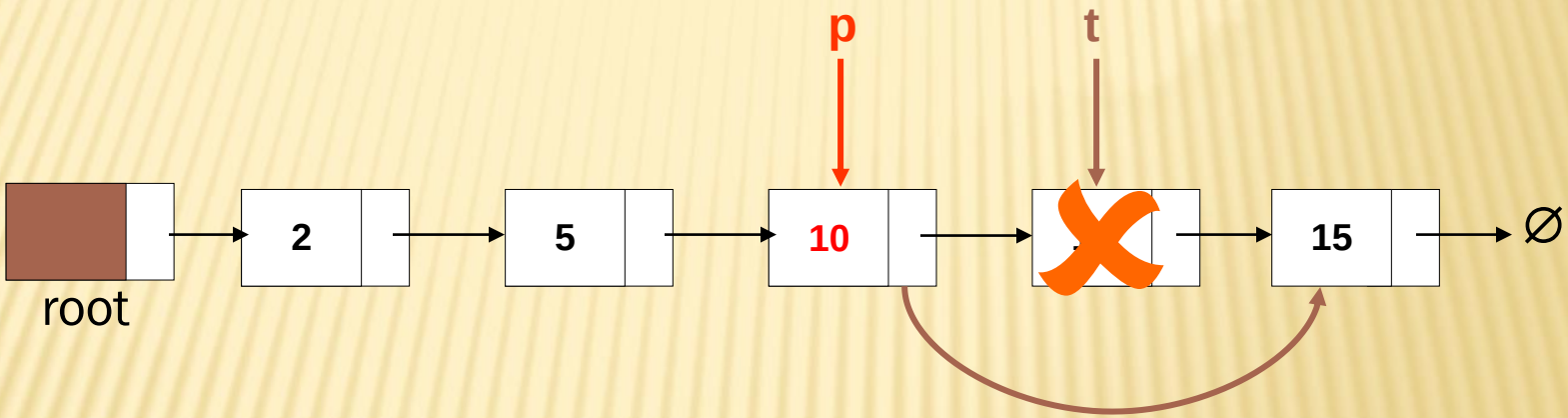
LINKED-LIST DELETING A NODE



previous->next = current->next;

delete current;

LINKED-LIST DELETING THE NODE POINTED TO BY A POINTER ‘P’, HAVING NO PREVIOUS POINTER




```
p->data = p->next->data;
t = p->next;
p->next = p->next->next;
delete t;
```

Boundary Conditions

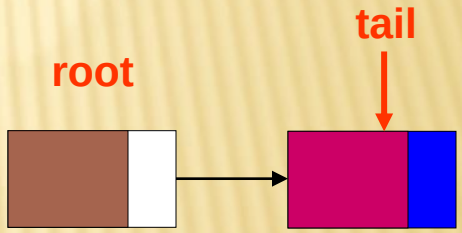
- Can we delete the first node?
- Can we delete the last node?

LINKED-LIST

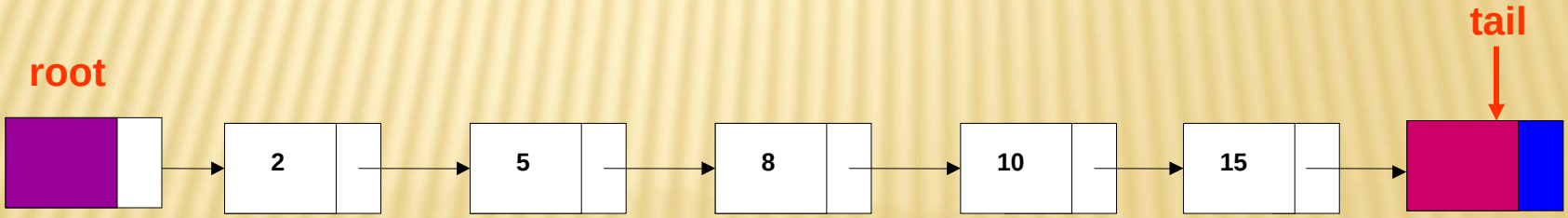
DELETING THE NODE POINTED TO BY A POINTER ‘P’, HAVING NO PREVIOUS POINTER

DUMMY TAIL NODE

```
Class LinkedList {
    private:
        Node root;
        Node *tail;
    public:
        LinkedList() {
            tail = new Node;
            root.next = tail;
        }
    …
};
```



Empty list

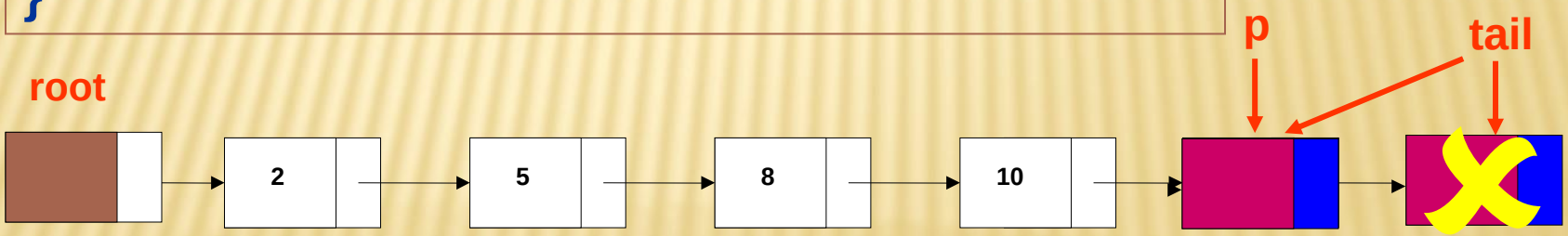


Why do we have tail as pointer to a node whereas root is a node?

LINKED-LIST

DELETING THE NODE POINTED TO BY A POINTER ‘P’, HAVING NO PREVIOUS POINTER

```
if (p->next == tail) {      // deleting the last node
      delete tail;
      tail = p;
}
else {                      // otherwise
      p->data = p->next->data;
      t = p->next;
      p->next = p->next->next;
      delete t;
}
```



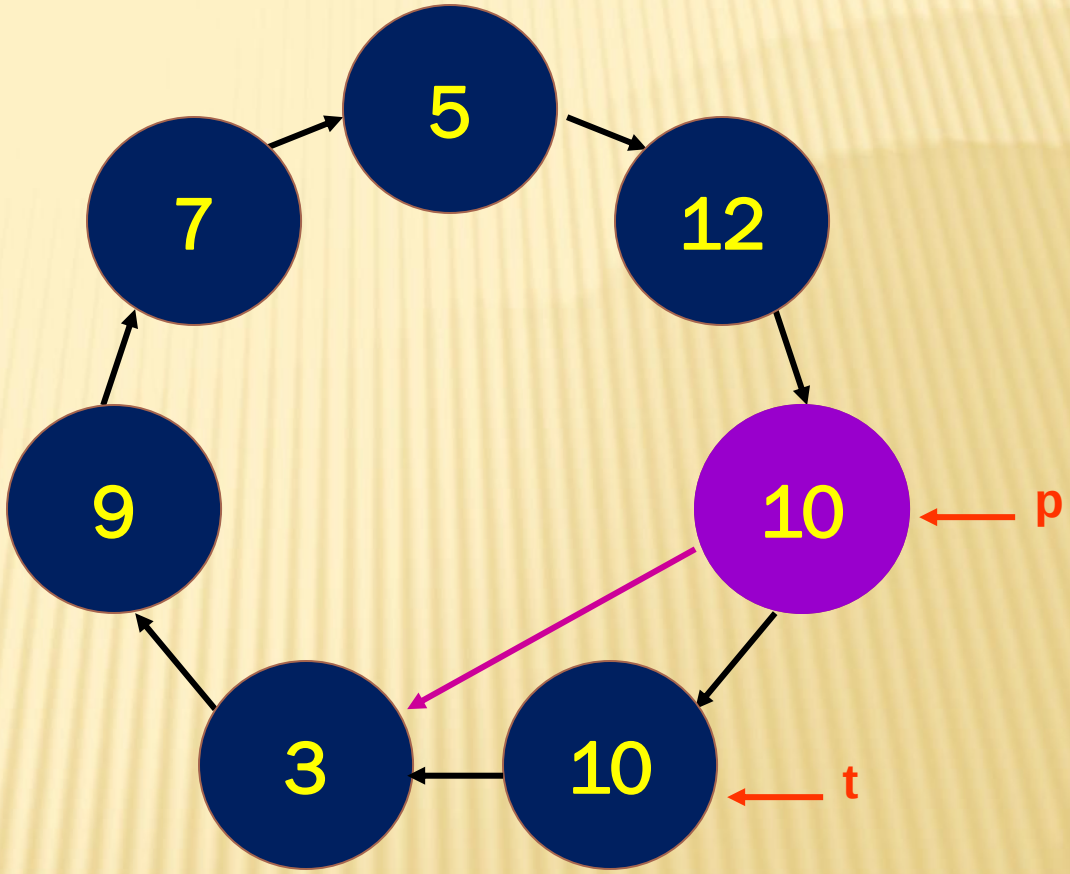
Linked-List

Deleting the Node pointed to by a pointer ‘p’, having no previous pointer

```
p->data = p->next->data;
t = p->next;
p->next = t->next;
delete t;
```

Boundary Conditions

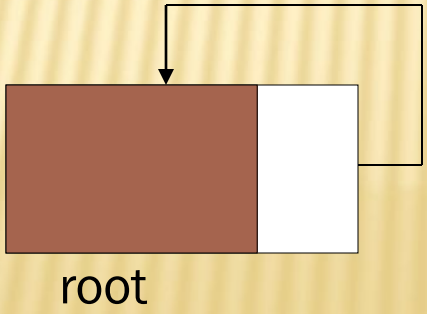
- Where is the starting point?
- What happens when we delete the node pointed to by the root?



Circular Linked-List

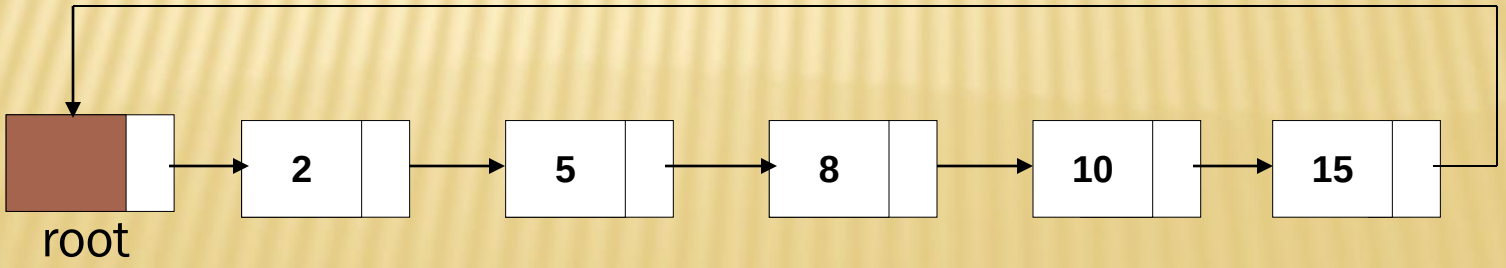
Circular Linked List

```
ClassCircularList {
    private:
        Node root;          // instead of Node *root
    public:
        CircularList() { root.next = &root; }
        void insert (int data);
        void delete (int data);
        void print ();
        ~CircularList();
};
```



```
void CircularList::insert(int data)
{
    Node *newNode = new Node;
    newNode->data = data;
    Node *current, *previous;
    previous = &root;
    current = previous->next;
    while ( current != &root && current->data < data ) {
        previous = current;
        current = current->next ;

    }
    newNode -> next = current;
    previous->next = newNode;
}
```



ADVANTAGES

- Every node is accessible from any node.
- That is, from a given node, all nodes can be reached by simply following the links.
- Example: Round Robin Queue for resource sharing such as process scheduling

DOUBLY LINKED-LIST

- Some times it is important to be able to go back and forth from a given point in the list.
- We can achieve it with the help of doubly linked lists.

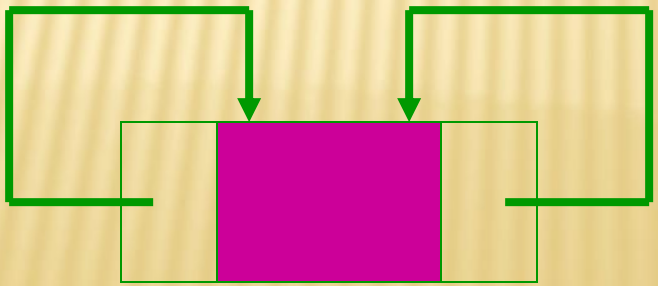
```
struct DoubleListNode {
    int                 data;
    DoubleListNode   *next;
    DoubleListNode   *previous;
};
```

In a doubly linked list, at any node pointed to by *"p"* we have the following:

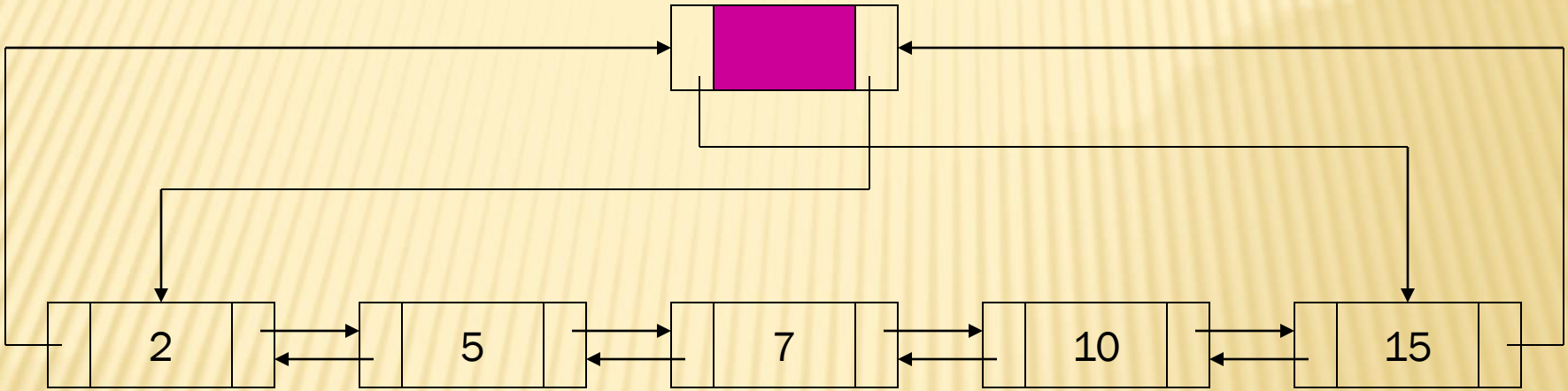
p == p->next->previous
p == p->previous->next

DOUBLY LINKED-LIST

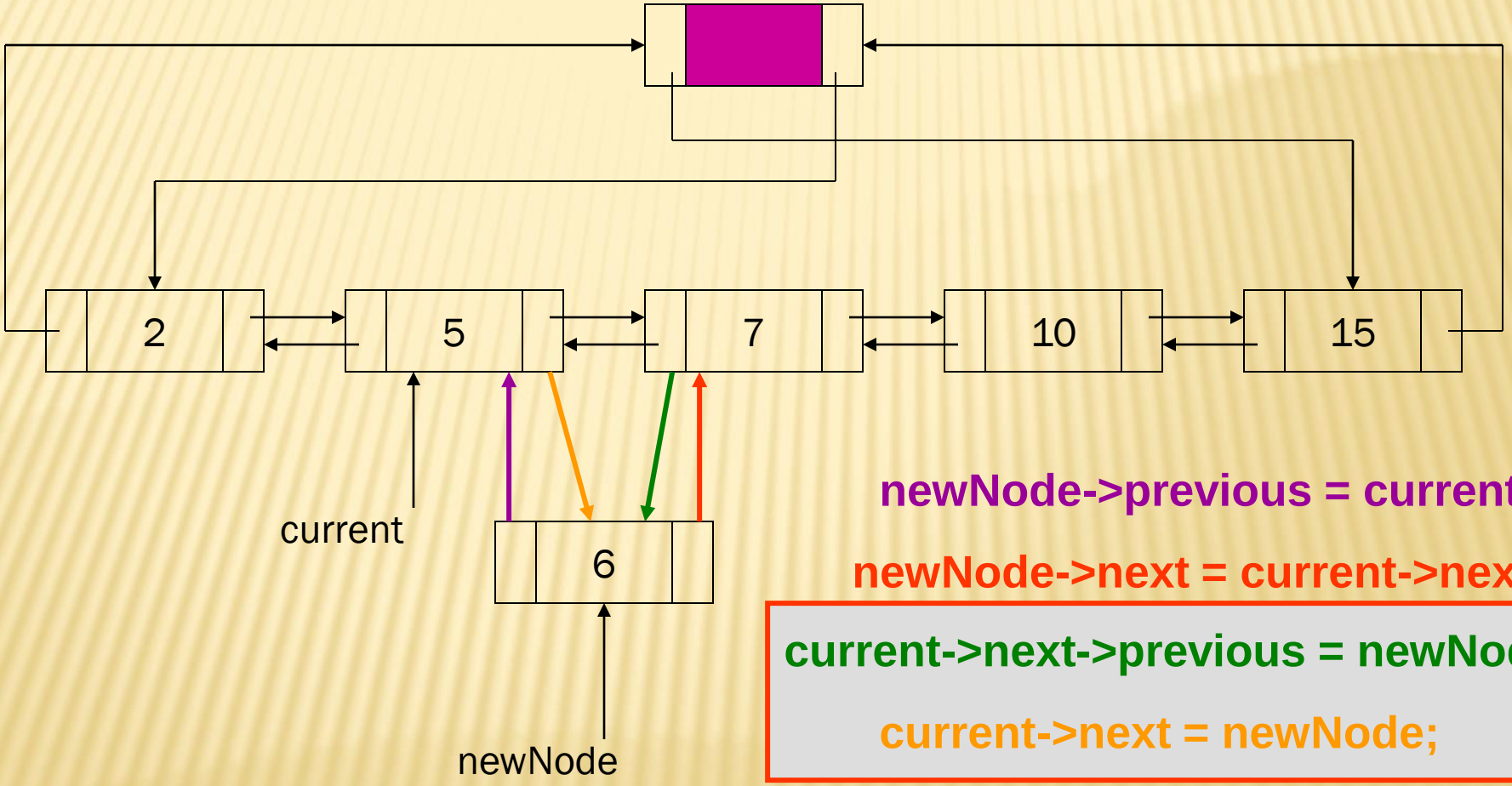
```
Class DoublyLinkedList {
    private:
        DoubleListNode head;
    public:
        DoublyLinkedList() { head.next = head.previous = &head; }
        void add (int data);
        void remove (int data);
        void print ();
        ~DoublyLinkedList();
};
```



DOUBLY LINKED-LIST



DOUBLY LINKED-LIST INSERTING A NODE



newNode->previous = current;

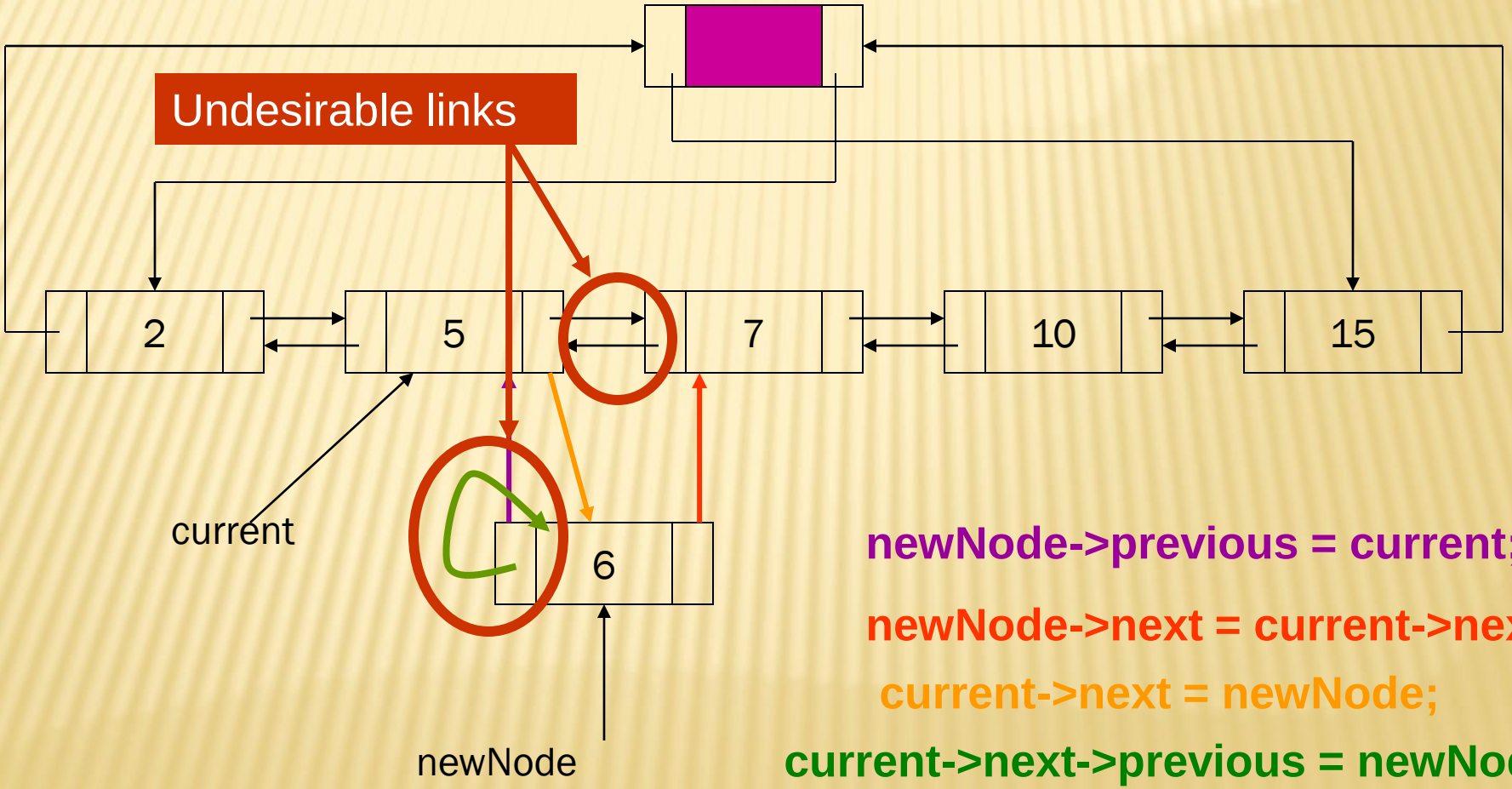
newNode->next = current->next;

current->next->previous = newNode;

current->next = newNode;

Order is very important!

DOUBLY LINKED-LIST INSERTING A NODE



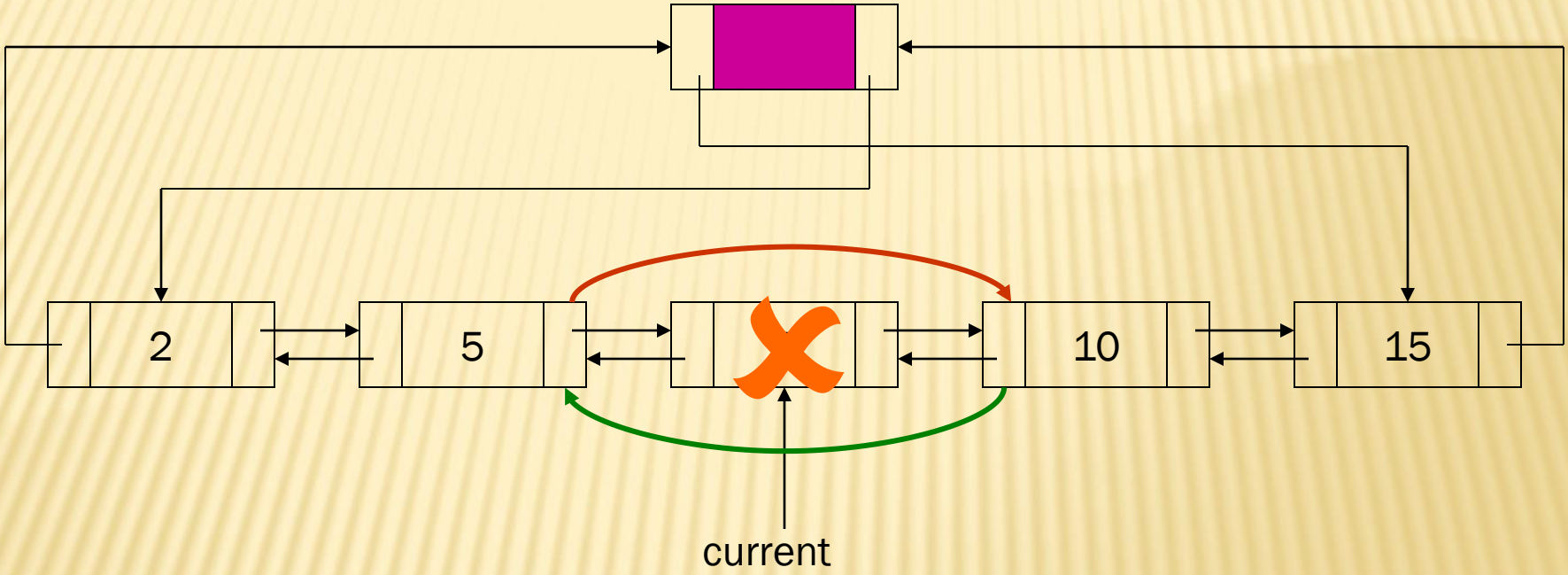
newNode->previous = current;

newNode->next = current->next;

current->next = newNode;

current->next->previous = newNode;

DOUBLY LINKED-LIST DELETING A NODE



current->previous->next = current->next;

current->next ->previous = current->previous;

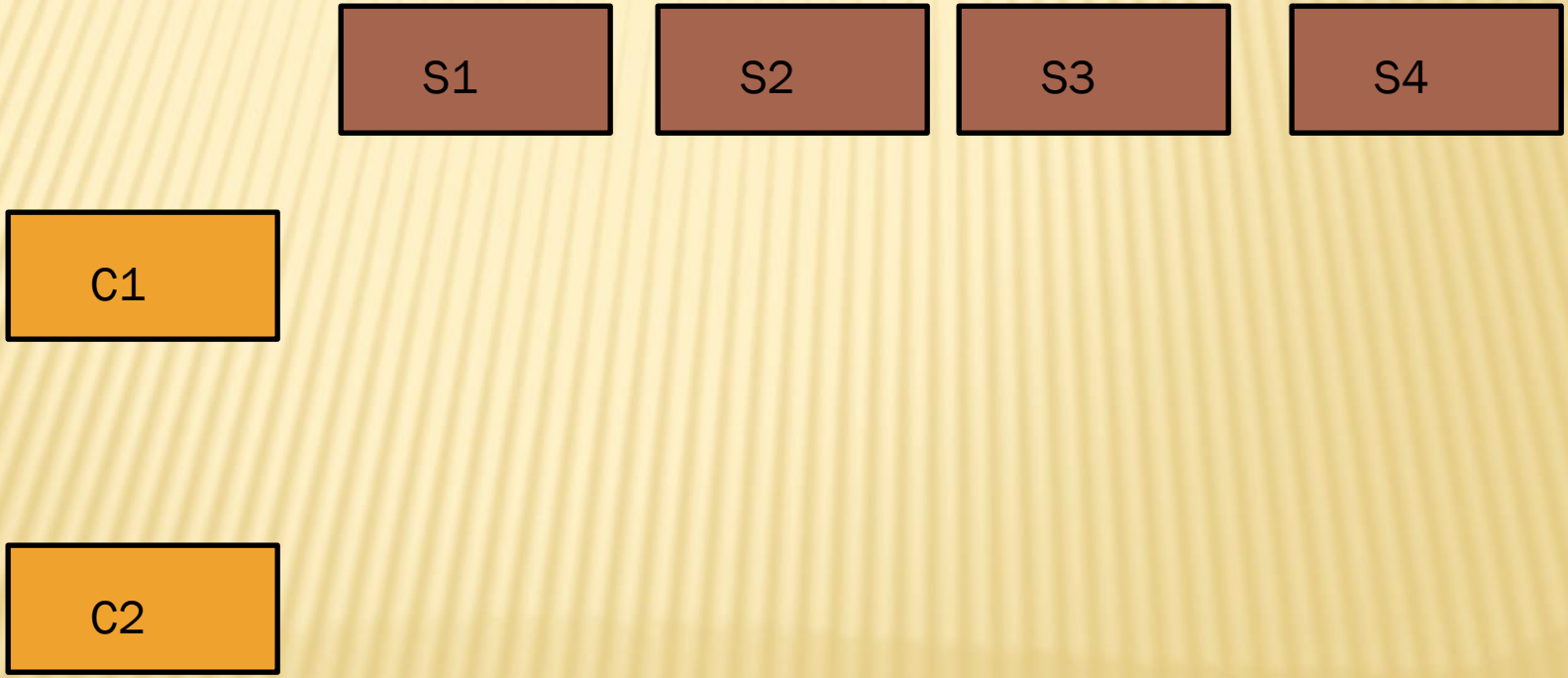
delete current;

EXAMPLE

- A university with 40,000 students & 2,500 courses needs to be able to generate two types of reports. The first report lists the registration for each class, & the second report lists the classes that each student is registered to.

EXAMPLE

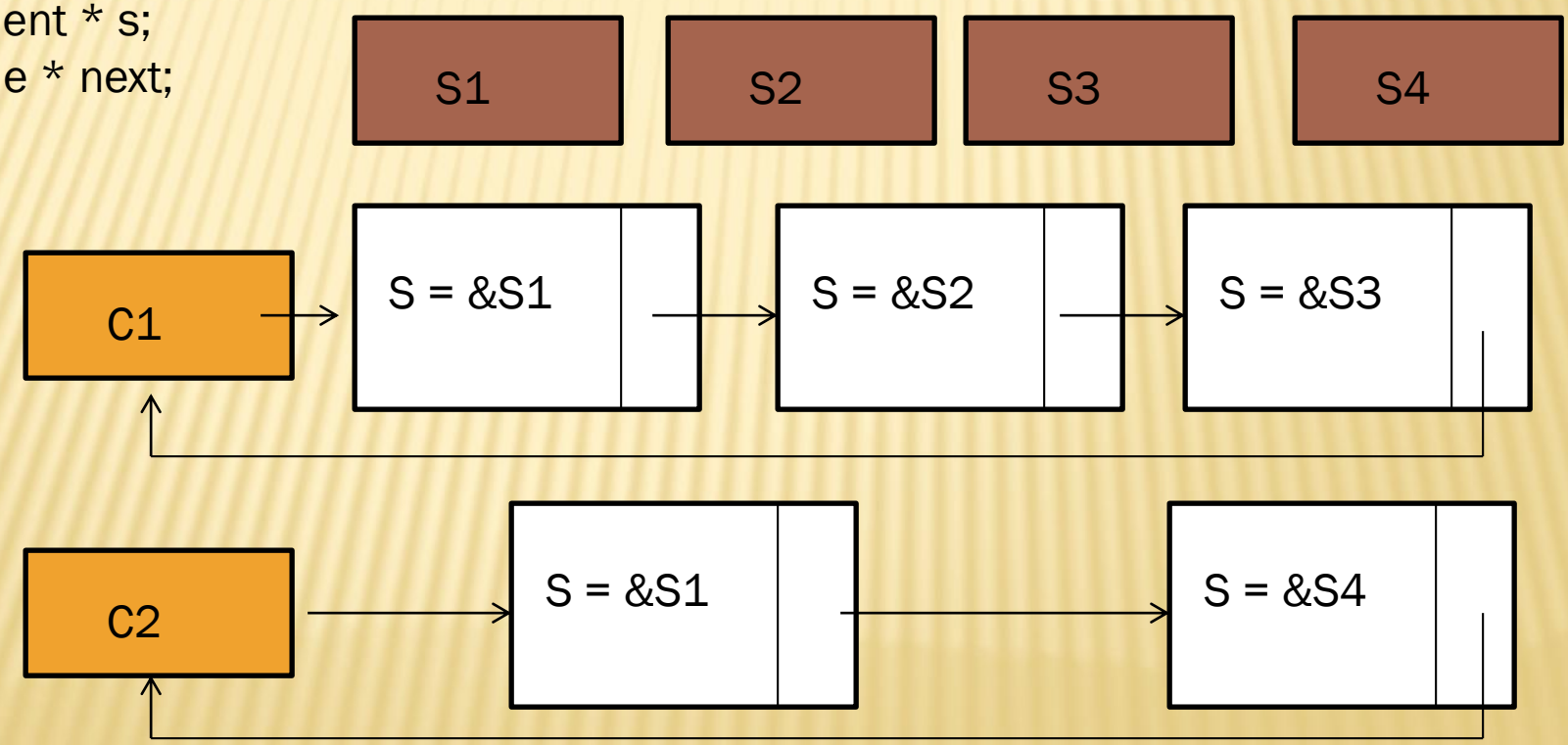
- Suppose we have 2 courses C1 & C2 and 4 students S1,S2,S3,S4



EXAMPLE- REGISTRATION OF EACH COURSE

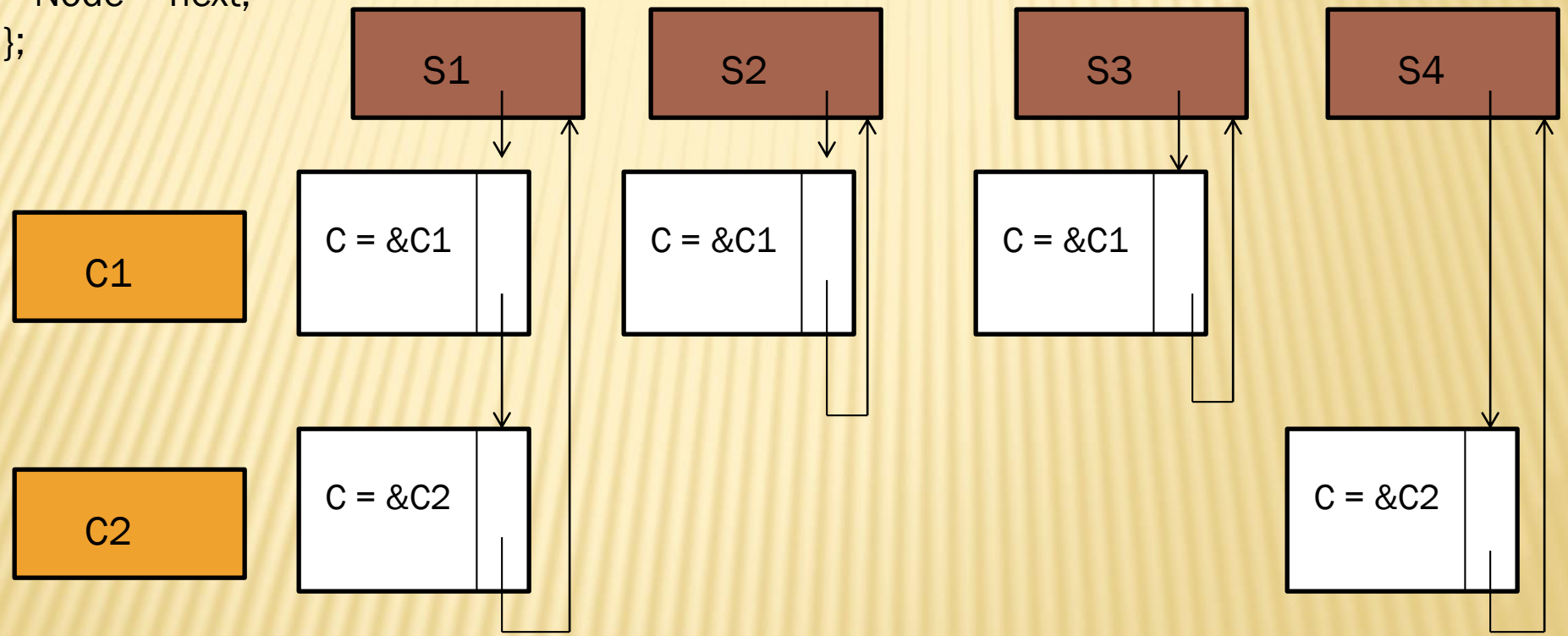
- Suppose S1, S2 & S3 are registered in C1 & S1 & S4 are in C2

```
Class Node{
  Student * s;
   Node * next;
};
```



EXAMPLE- COURSES IN WHICH EACH STUDENT IS REGISTERED

```
Class Node{
  Course* c;
   Node * next;
};
```



EXAMPLE- MULTILISTS

```
Class Node{
  Course* c;
  Student *s;
   Node * nextS;
   Node *nextC;
};
```