Linked List
(Data Structure)

Humayun Kabir

Professor, CS, Vancouver Island University, BC, Canada

Objectives

- Describe linked structures
- Compare linked structures to array-based structures
- Explore the techniques for managing a linked list
- Discuss the need for a separate node class to form linked structures

Array Limitations

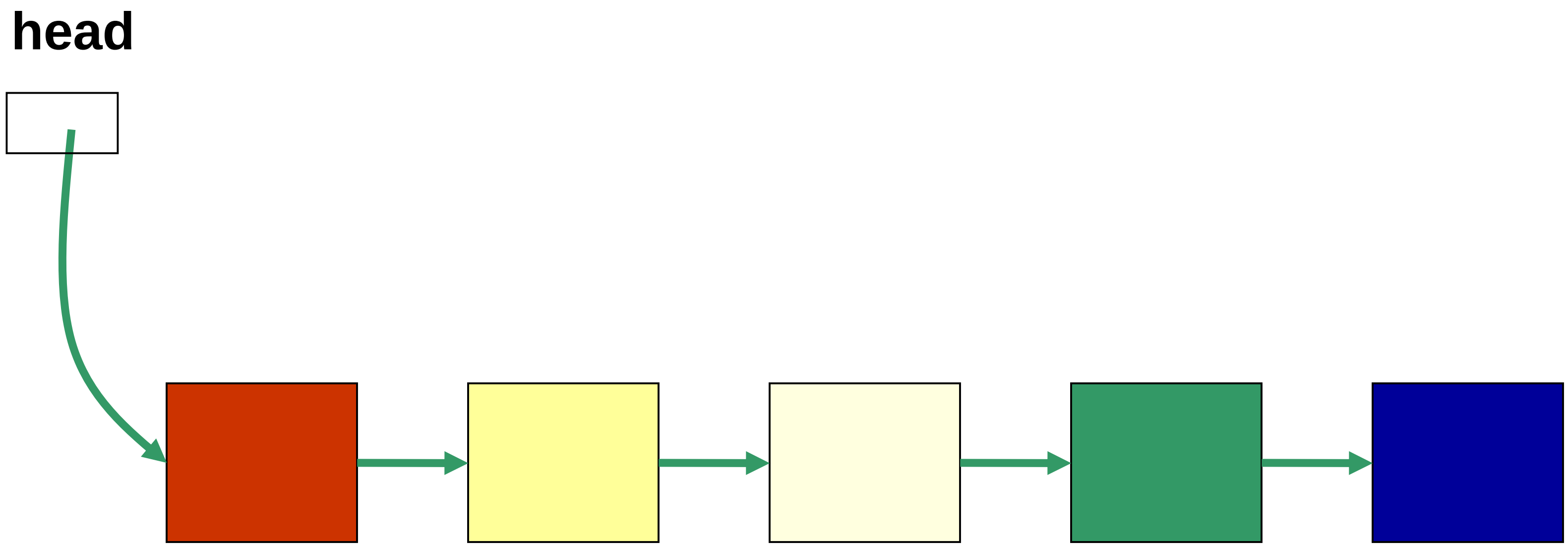
- What are the limitations of an array, as a data structure?
  - Fixed size
  - Physically stored in consecutive memory locations
  - To insert or delete items, may need to shift data

Linked Data Structures

- A ***linked*** data structure consists of items that are linked to other items
  - How? each item ***points to*** another item

- ***Singly linked list:*** each item points to the **next** item
- ***Doubly linked list:*** each item points to the **next** item *and* to the **previous** item

Conceptual Diagram of a Singly-Linked List



Advantages of Linked Lists

- The items do ***not*** have to be stored in consecutive memory locations: the successor can be anywhere physically
  - So, can insert and delete items without shifting data
  - Can increase the size of the data structure easily
- Linked lists can grow ***dynamically*** (i.e. at run time) – the amount of memory space allocated can grow and shrink as needed

Nodes

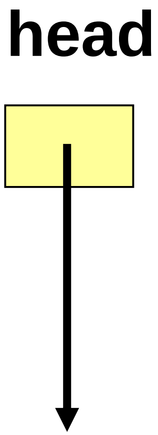
- A linked list is an ordered sequence of items called ***nodes***
  - A node is the basic unit of representation in a linked list
- A ***node*** in a ***singly linked list*** consists of two fields:
  - A ***data*** portion
  - A ***link (pointer)*** to the ***next*** node in the structure
- The first item (node) in the linked list is accessed via a ***head*** or ***front*** pointer
  - The linked list is defined by its head (this is its starting point)

Singly Linked Llist Node

```
struct Node {
        int data;
        Node *next;
};
```

Singly Linked List

```
Node* head = NULL;   //global pointer
```

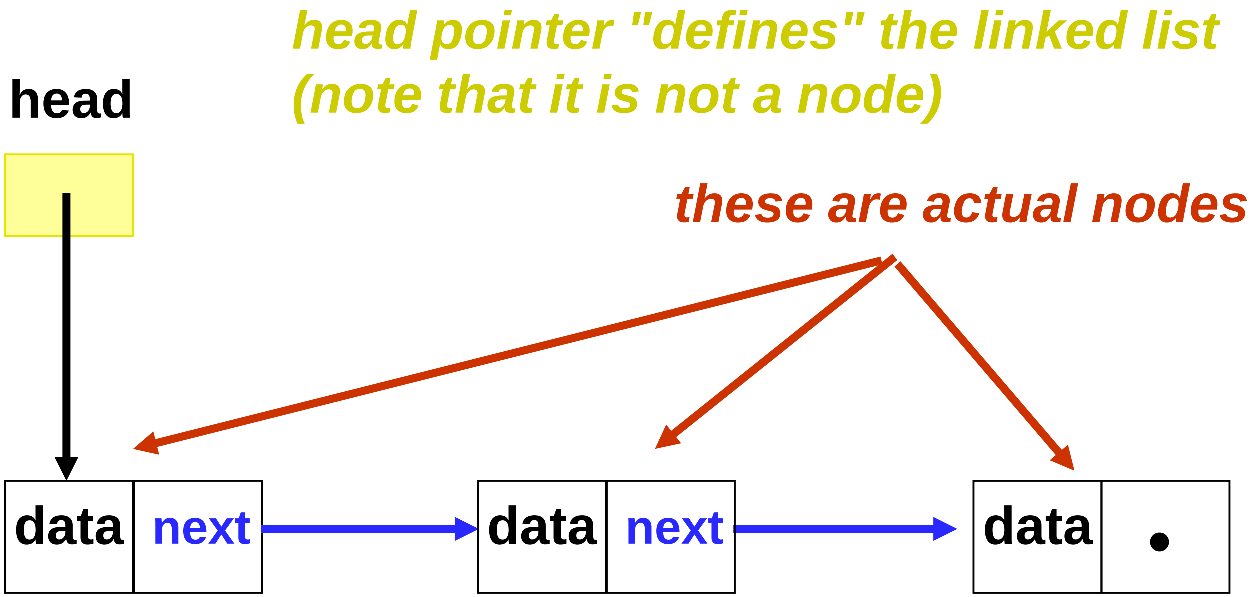


head

head pointer "defines" the linked list

(note that it is not a node)

Singly Linked List



head pointer "defines" the linked list
(note that it is not a node)

these are actual nodes

Linked List

Note*: we will hereafter refer to a singly linked list just as a "****linked list****"*

- ***Traversing the linked list***
  - How is the first item accessed?
  - The second?
  - The last?

- What does the last item point to?
  - We call this the ***null link***

Discussion

- How do we get to an item's successor?
- How do we get to an item's predecessor?
- How do we access, say, the 3rd item in the linked list?

- How does this differ from an array?

Searching in a Linked List

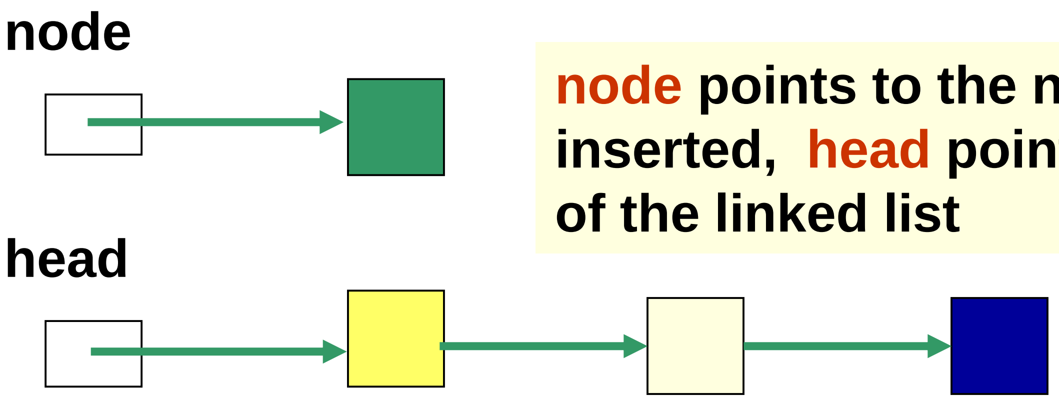
```
Node* searchLinkedList(int key) {
    Node* iterator = head; //assuming head is a global pointer
    while(iterator != NULL) {
        if (iterator->data == key ) {
            return iterator;
        }
        iterator = iterator->next;
    }
    return NULL;
}
```

Linked List Operations

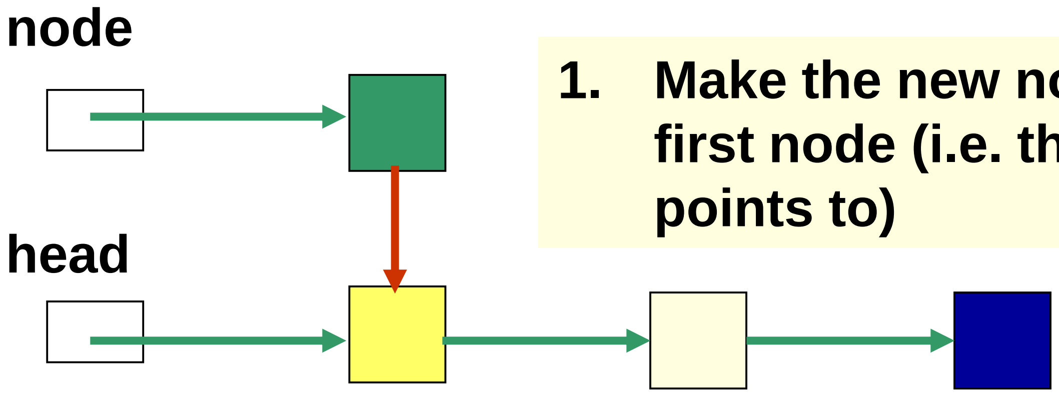
We will now examine linked list operations:

- ***Add*** an item to the linked list
    - We have 3 situations to consider:
        - insert a node at the head
        - insert a node in the middle
        - insert a node at the end
- ***Delete*** an item from the linked list
    - We have 3 situations to consider:
        - delete the node at the head
        - delete an interior node
        - delete the last node

Inserting a Node at the Front

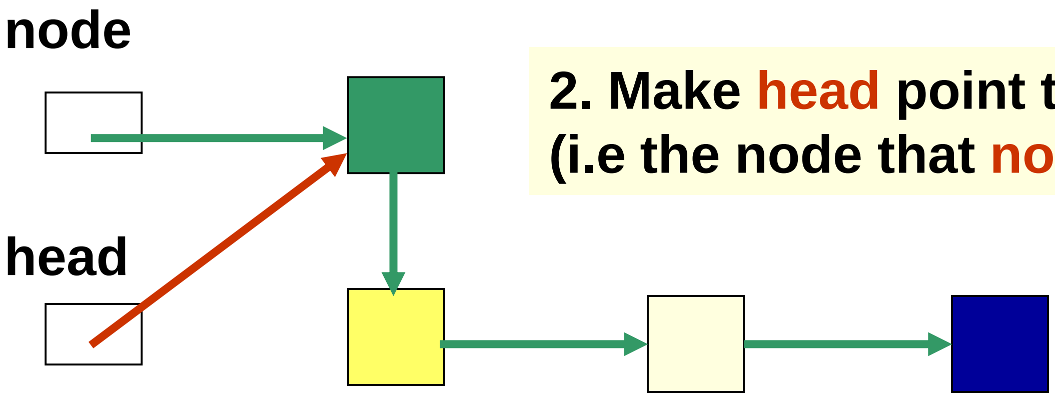


node points to the new node to be inserted, head points to the first node of the linked list



1. Make the new node point to the first node (i.e. the node that head points to)

Inserting a Node at the Front



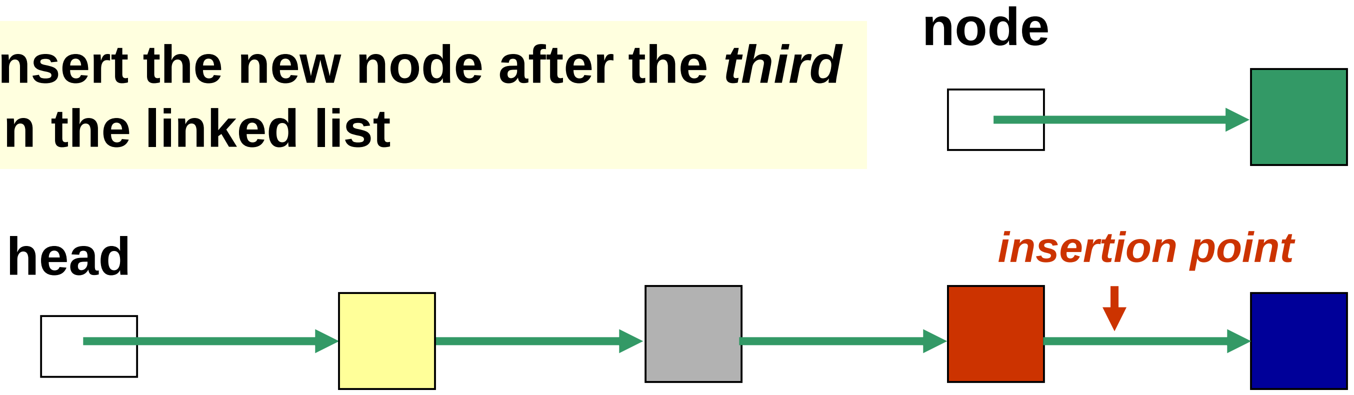
2. Make head point to the new node (i.e the node that node points to)

Inserting a Node at the Front

```
void insertNodeAtFront(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->next = NULL;
        //assuming head is a global pointer
        newNode->next = head;
        head = newNode;
}
```

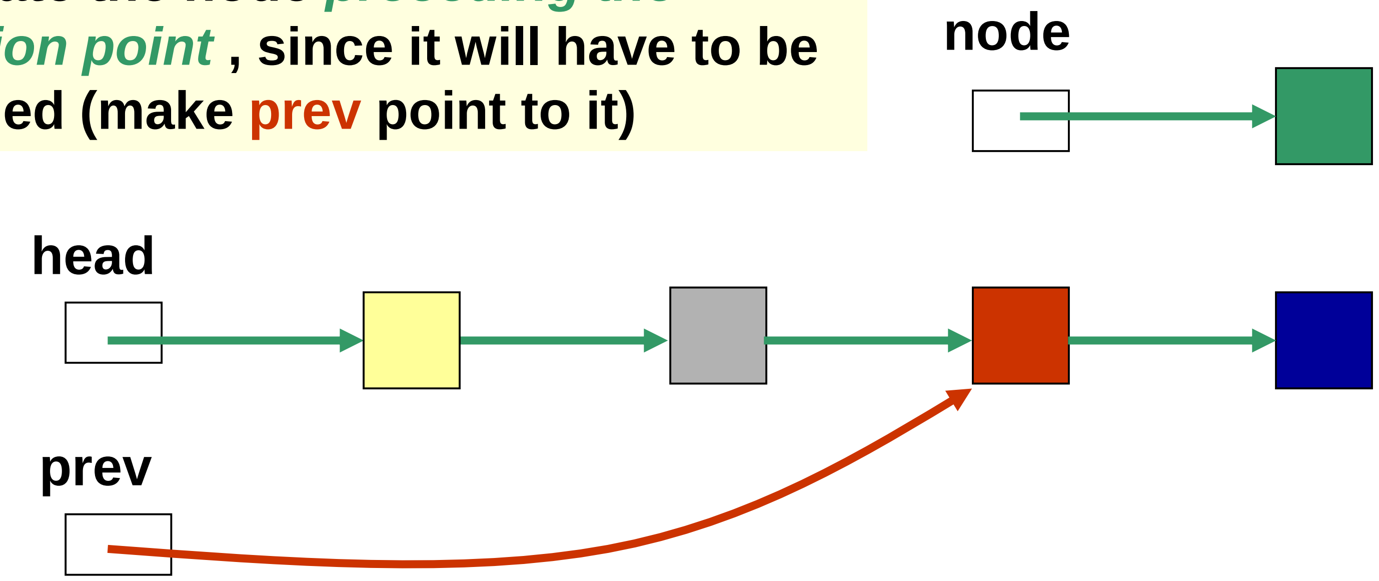
Inserting a Node in the Middle

Let's insert the new node after the *third* node in the linked list



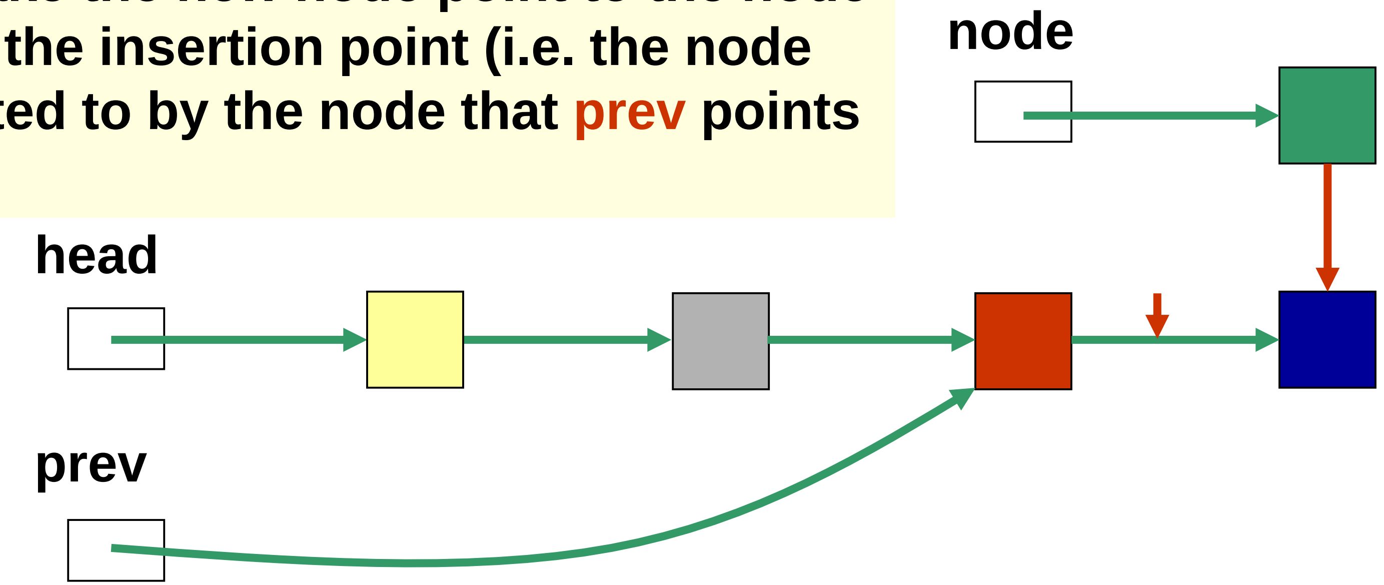
Inserting a Node in the Middle

1. Locate the node *preceding the insertion point* , since it will have to be modified (make prev point to it)



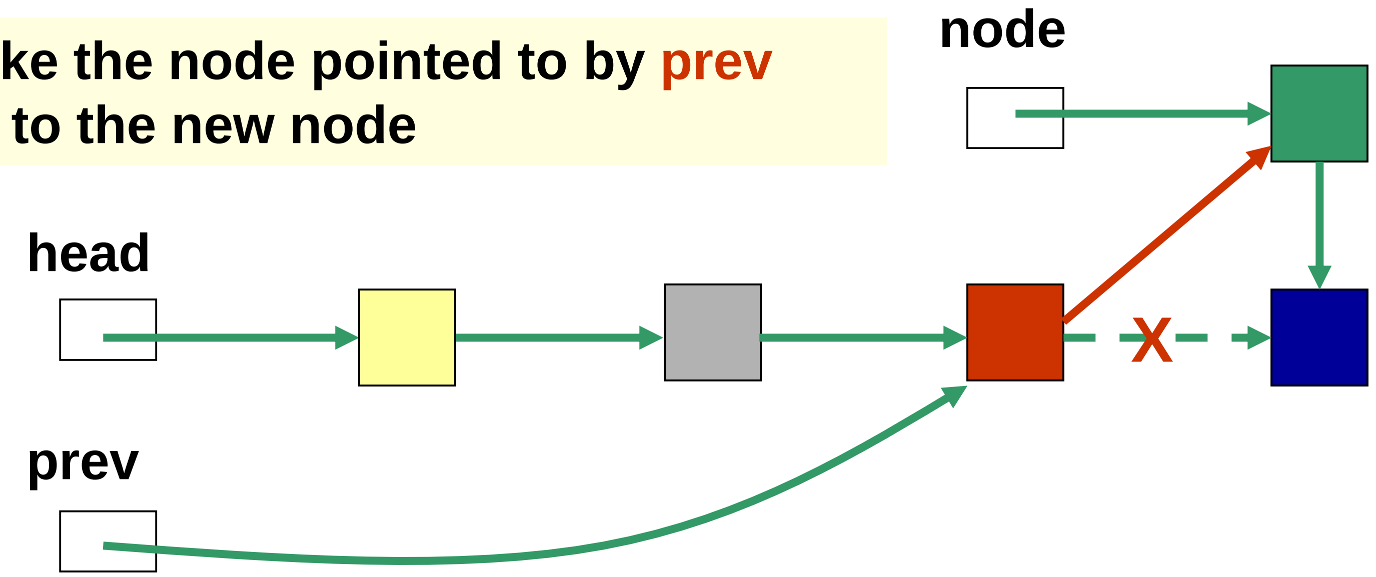
Inserting a Node in the Middle

2. Make the new node point to the node after the insertion point (i.e. the node pointed to by the node that prev points to)



Inserting a Node in the Middle

3. Make the node pointed to by prev point to the new node



Inserting a Node in the Middle

```
void insertNodeAtMiddle(Node *node, int after) {
        if(node == NULL) return; //sanity check
        Node* prev = head; //assuming head is a global pointer
        while(prev != NULL) {
                if(prev->data == after) {
                        node->next = prev->next;
                        prev->next = node;
                        break;
                }
                prev = prev->next;
        }
}
```

Inserting a Node in the Middle

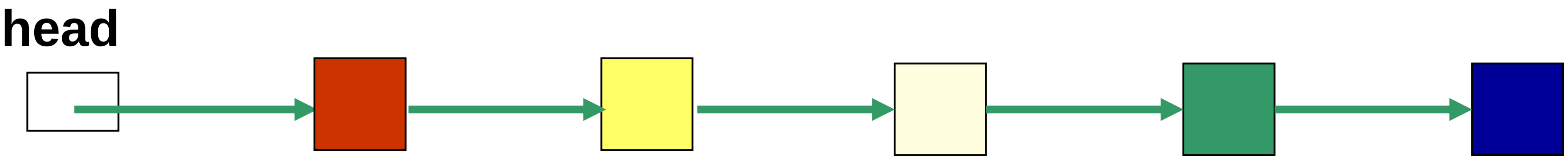
```
void insertNodeAtMiddle(Node *node, int before) {
    if(node == NULL) return; //sanity check
    Node* prev = head; //assuming head is a global pointer
    while(prev != NULL && prev->next != NULL) {
        if(prev->next->data == before) {
            node->next = prev->next;
            prev->next = node;
            break;
        }
        prev = prev->next;
    }
}
```

Discussion

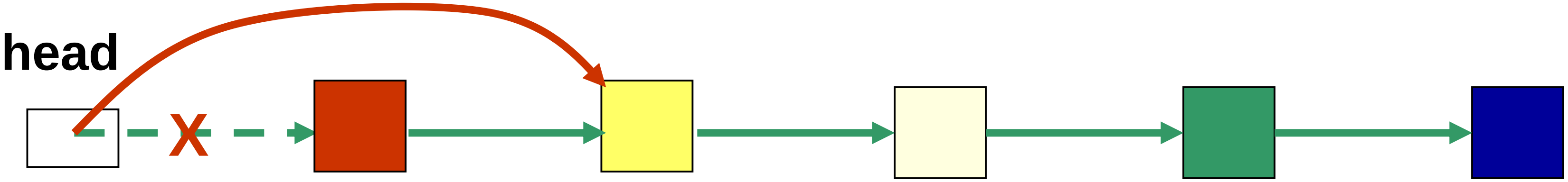
- Inserting a node at the head is a special case; why?
- Is inserting a node at the end a special case?

Deleting the First Node

head points to the first node in the linked list, which points to the second node



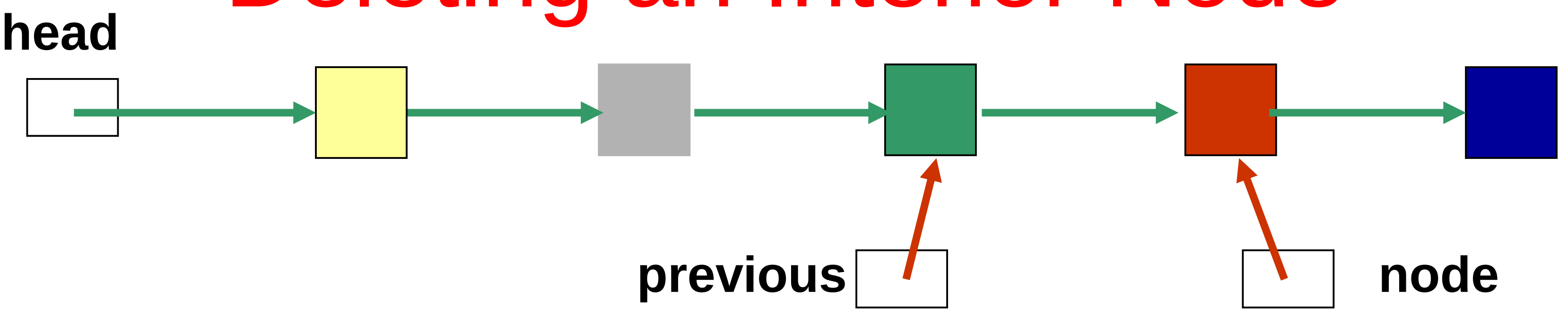
Make head point to the second node (i.e. the node pointed to by the first node)



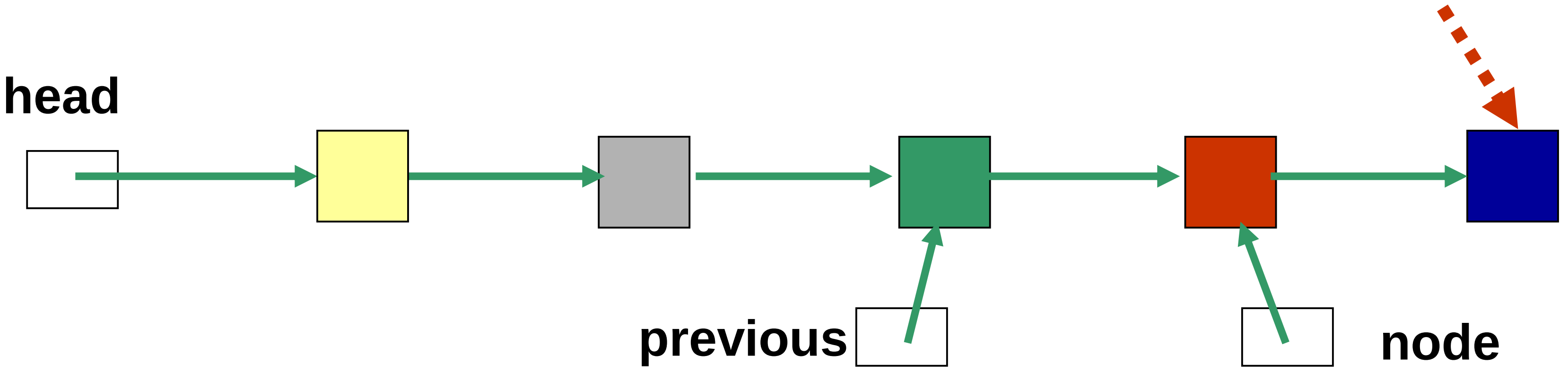
Deleting the First Node

```
void deleteFirstNode() {
        if( head == NULL) {  //assuming head is a global pointer
                return;
        }
        Node* node = head;
        head = node->next;
        delete node;
}
```

Deleting an Interior Node

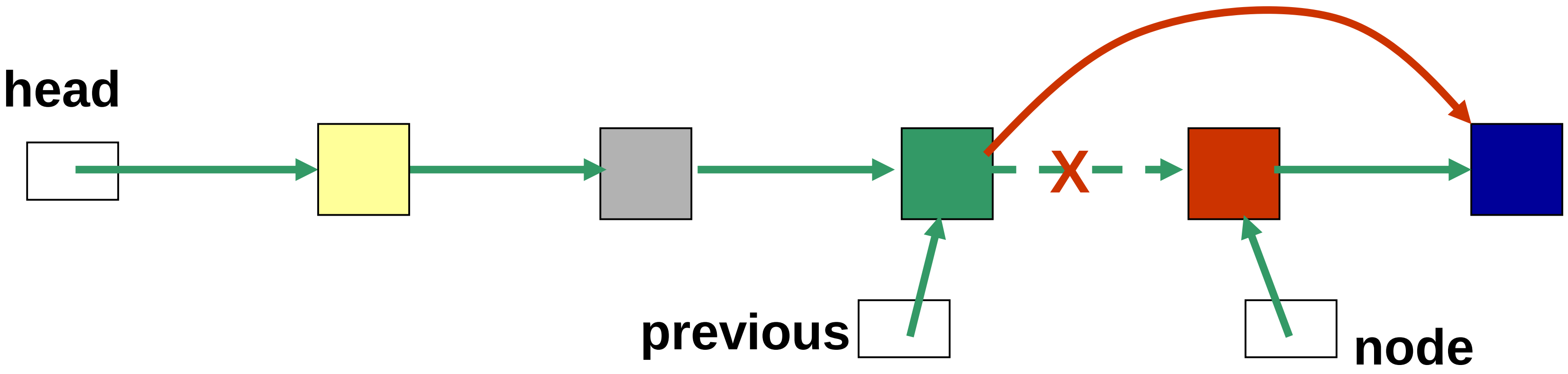


1. Traverse the linked list so that previous points to the node prior to the one to be deleted



2. We need to get the node *next to the one to be deleted*

Deleting an Interior Node



3. Make the node that previous points to, point to the node next to the one that to be deleted

Deleting an Interior Node

```
void deleteNode(Node* node) {
        if(node == NULL) return;
        if (head == node) {                        //assuming head is a global pointer
                deleteFirstNode();
                return;
        }
        Node* prev = head;
        while( prev != NULL && prev->next != NULL) {
                if(prev->next == node) {
                        prev->next = node->next;
                        delete node;
                        break;
                }
                prev = prev->next;
        }
}
```

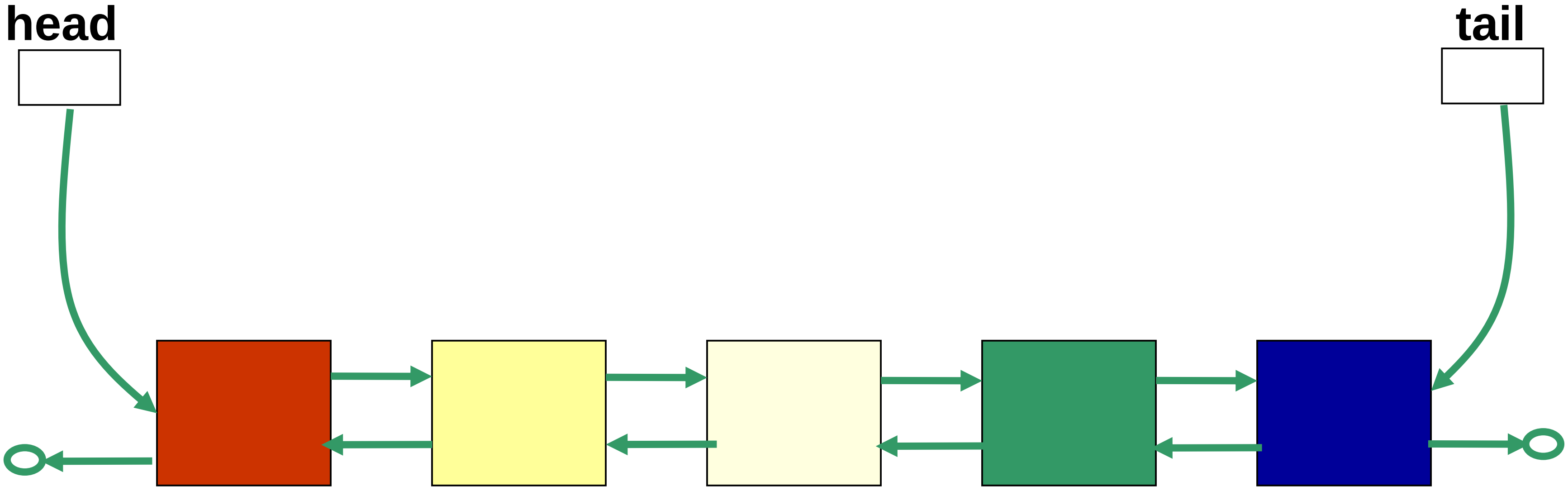
Discussion

- Deleting the node at the front is a special case; why?
- Is deleting the last node a special case?

Doubly Linked Llist Node

```
struct Node {
        int data;
        Node* prev;
        Node* next;
};
```

Conceptual Diagram of a Doubly-Linked List



Conceptual Diagram of a Circular-Linked List