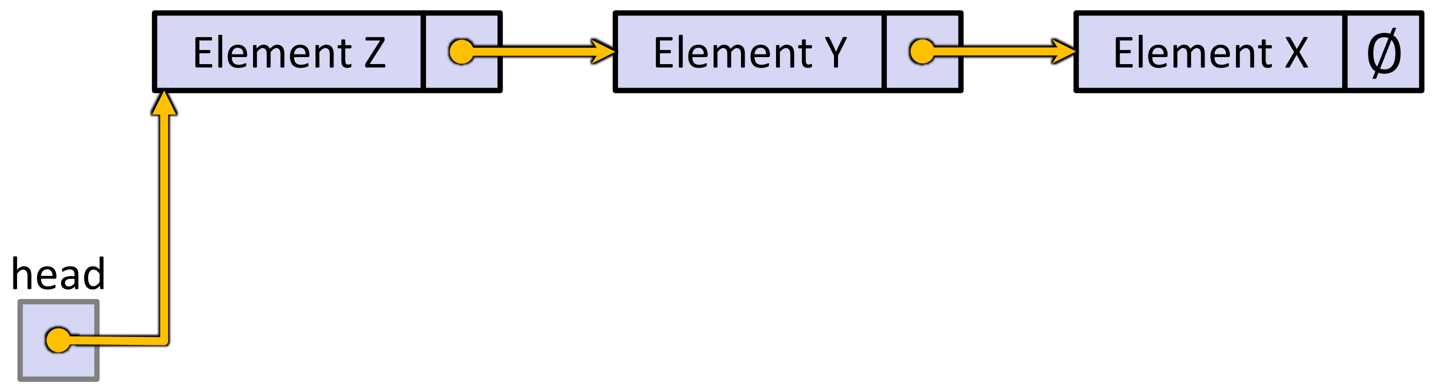
## Lecture Outline

- Implementing Linked List using C

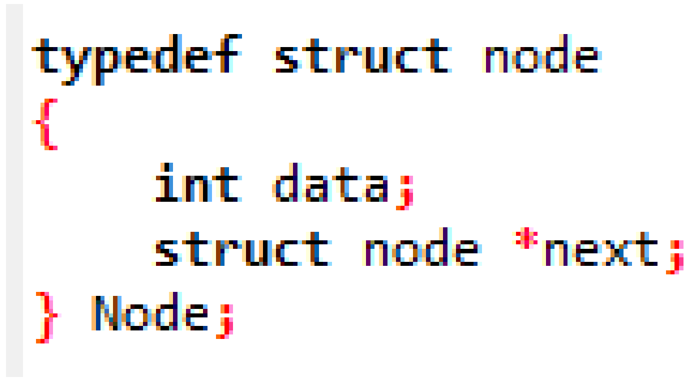
## Simple Linked List in C

- Each node in a linear, singly-linked list contains:
  - Some element as its payload
  - A pointer to the next node in the linked list
    - This pointer is NULL (or some other indicator) in the last node in the list

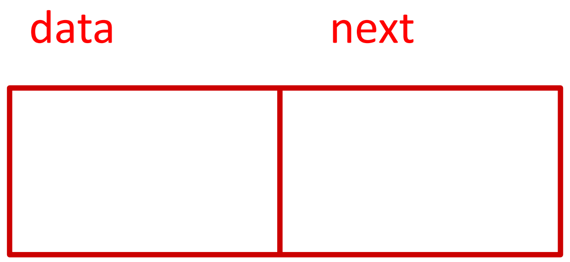


## Simple Linked List in C  from  Lab 7

- Let’s start with a node struct



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



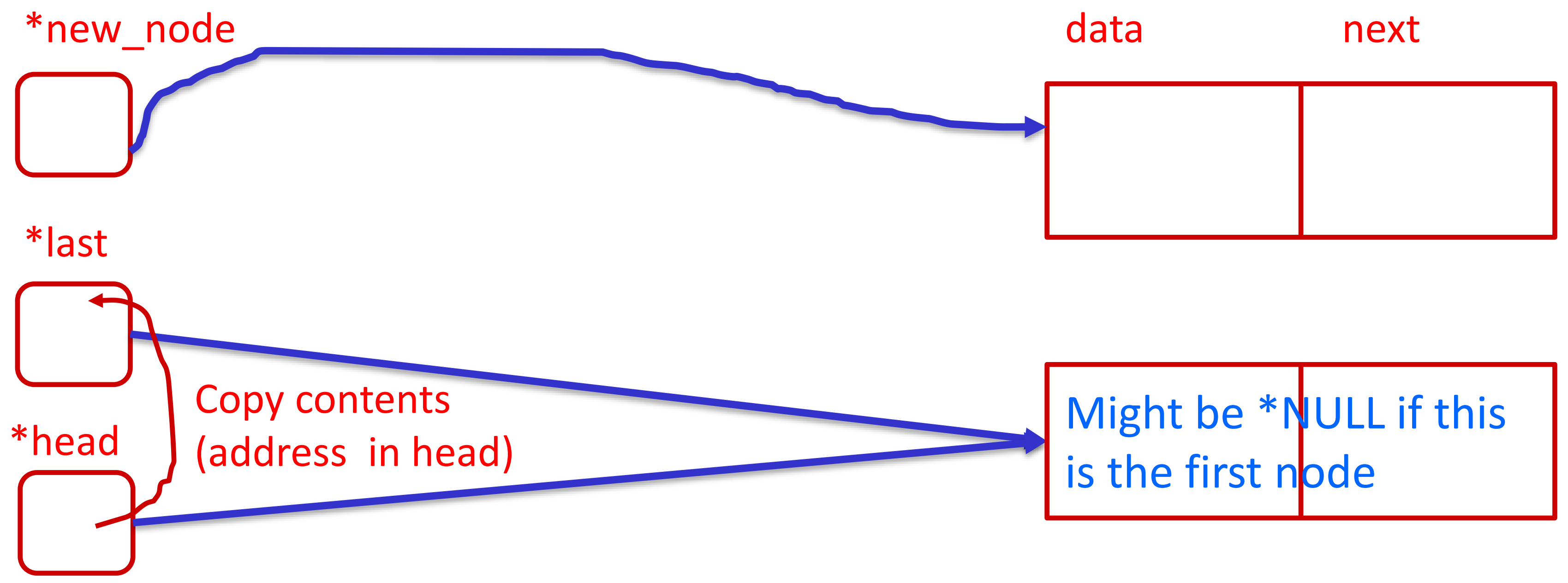
No allocation yet!

```
//Creating head and last nodes as global Node* s
Node *head = NULL;
```

## Appending a node

- void append(int new_data)

```
{/* 1. allocate node */
Node* new_node = (Node*) malloc(sizeof( Node));
Node *last = head;  //declare last for iteration
```



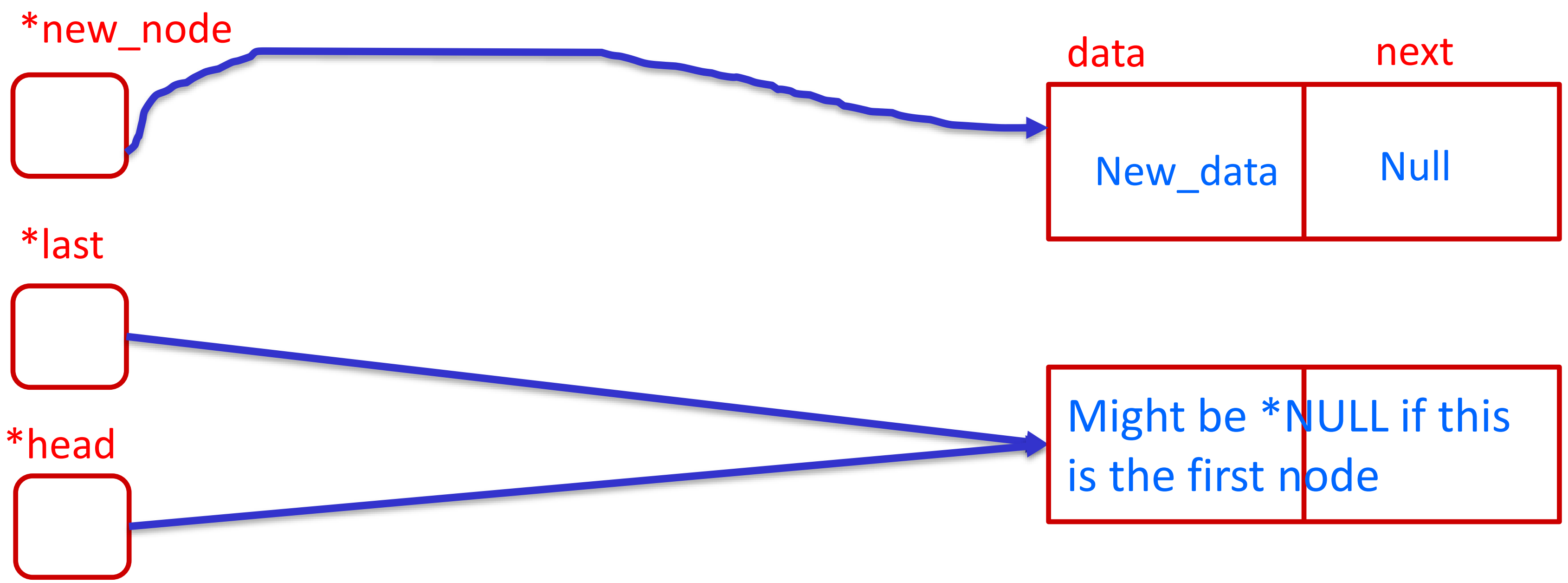
## Appending a node

- /* 2. put in the data  */

```
new_node->data  = new_data;
```

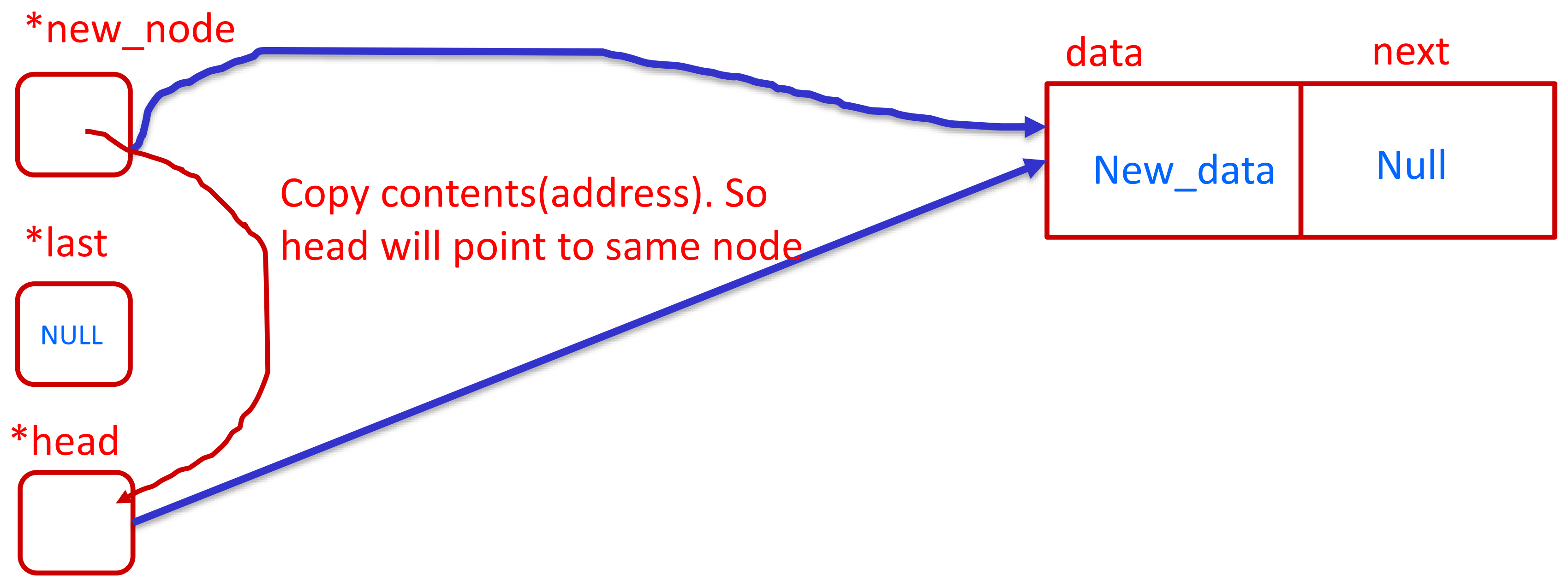
/* 3. This new node is going to be the last node, so make next of  it NULL*/

```
new_node->next = NULL;
```



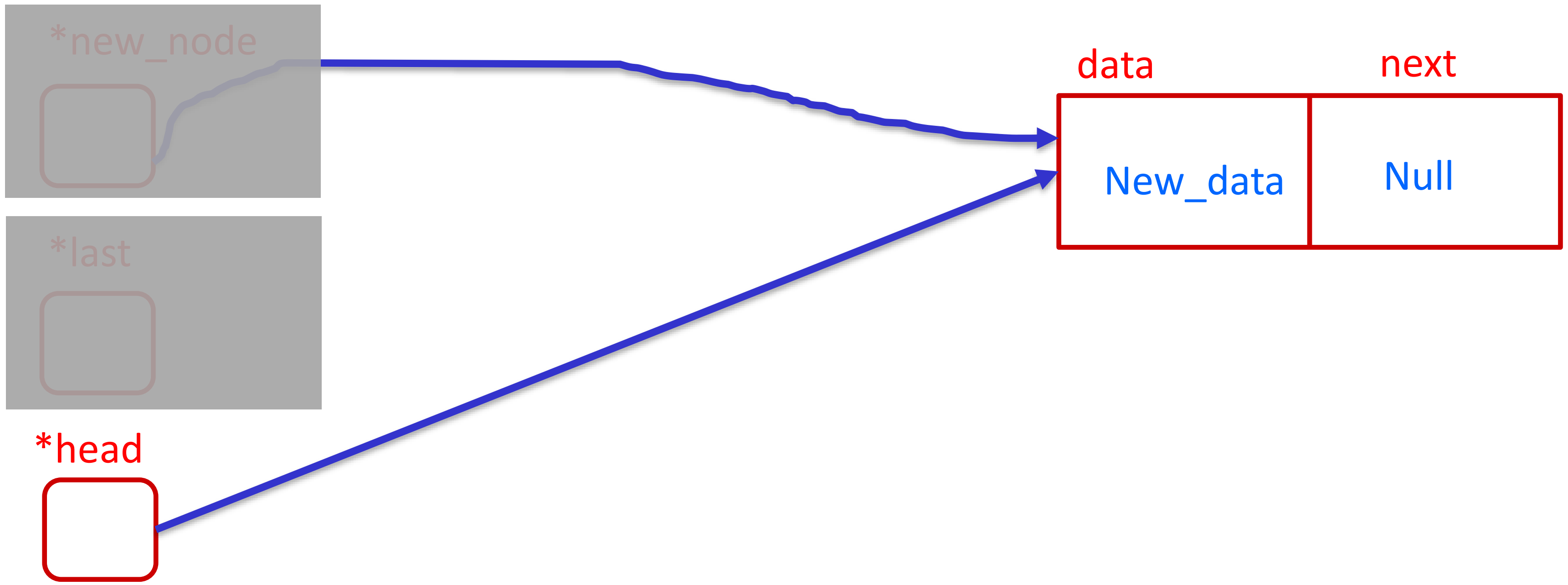
## Appending a node

- /* 4. If the Linked List is empty, then make the new node as head */
- if (head == NULL)  { head = new_node;   return; }



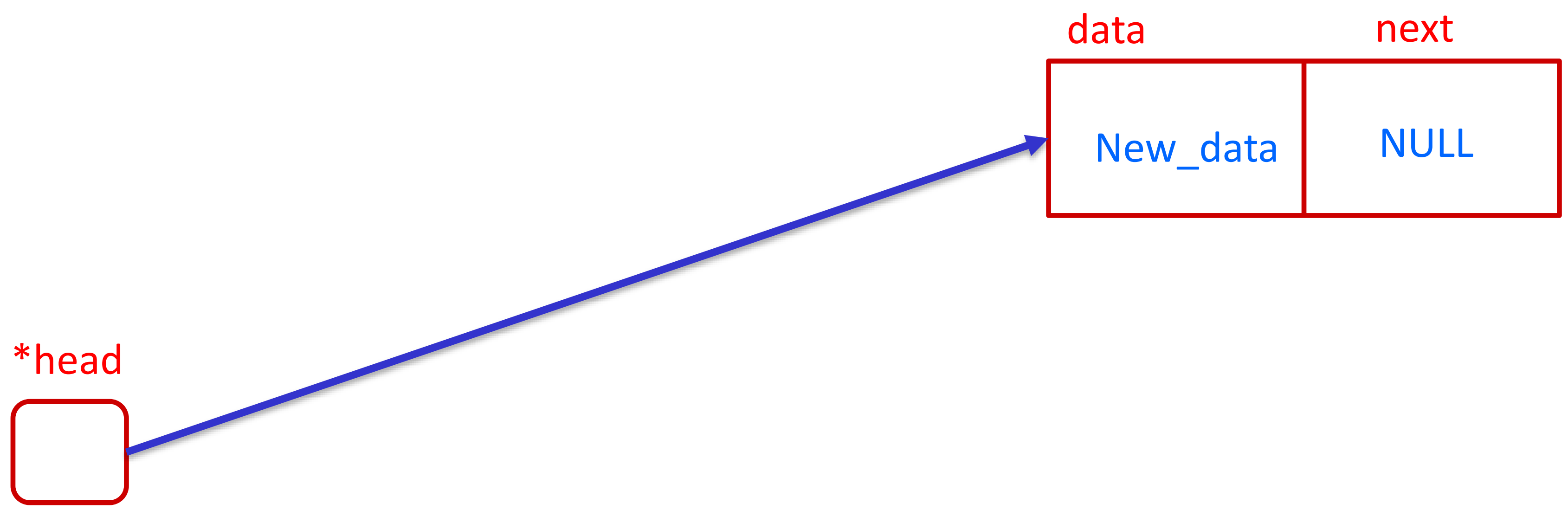
## Appending a node

- /* 4. If the Linked List was empty, then new node becomes head */
- if (head == NULL) { head = new_node; return; }



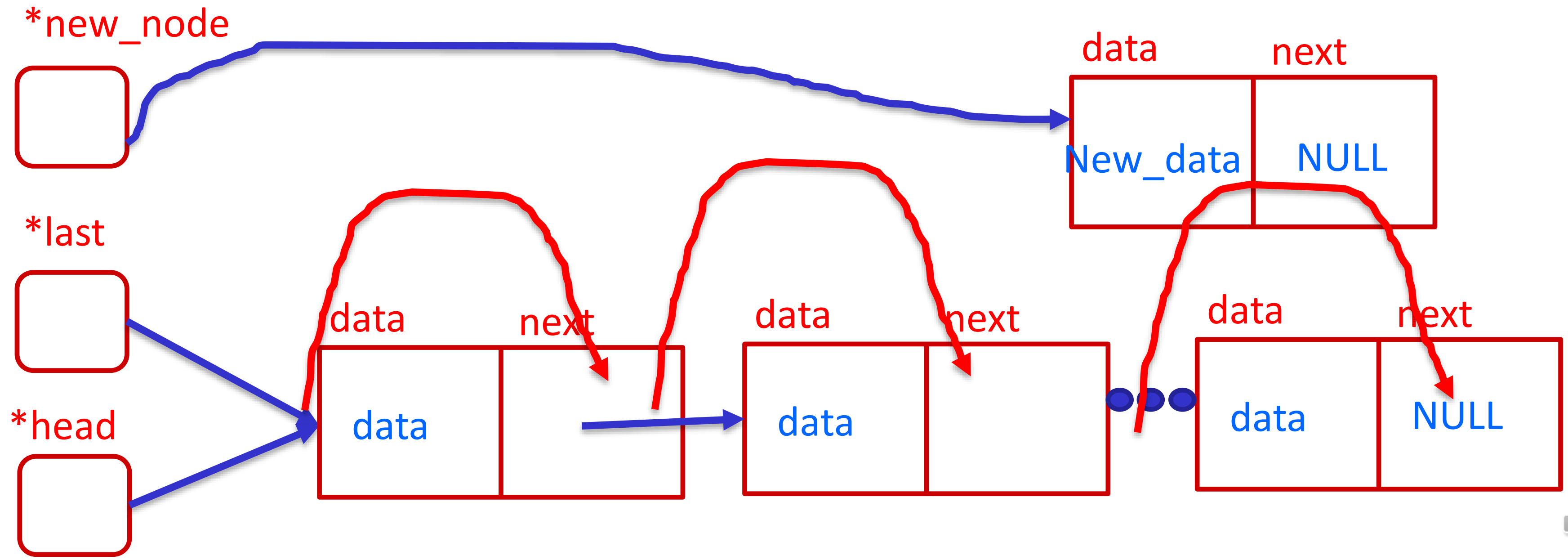
# Appending a node

- /* 4. If the Linked List was empty, then new node becomes head */
- if (head == NULL)  { head = new_node;   return; }



# Appending a node

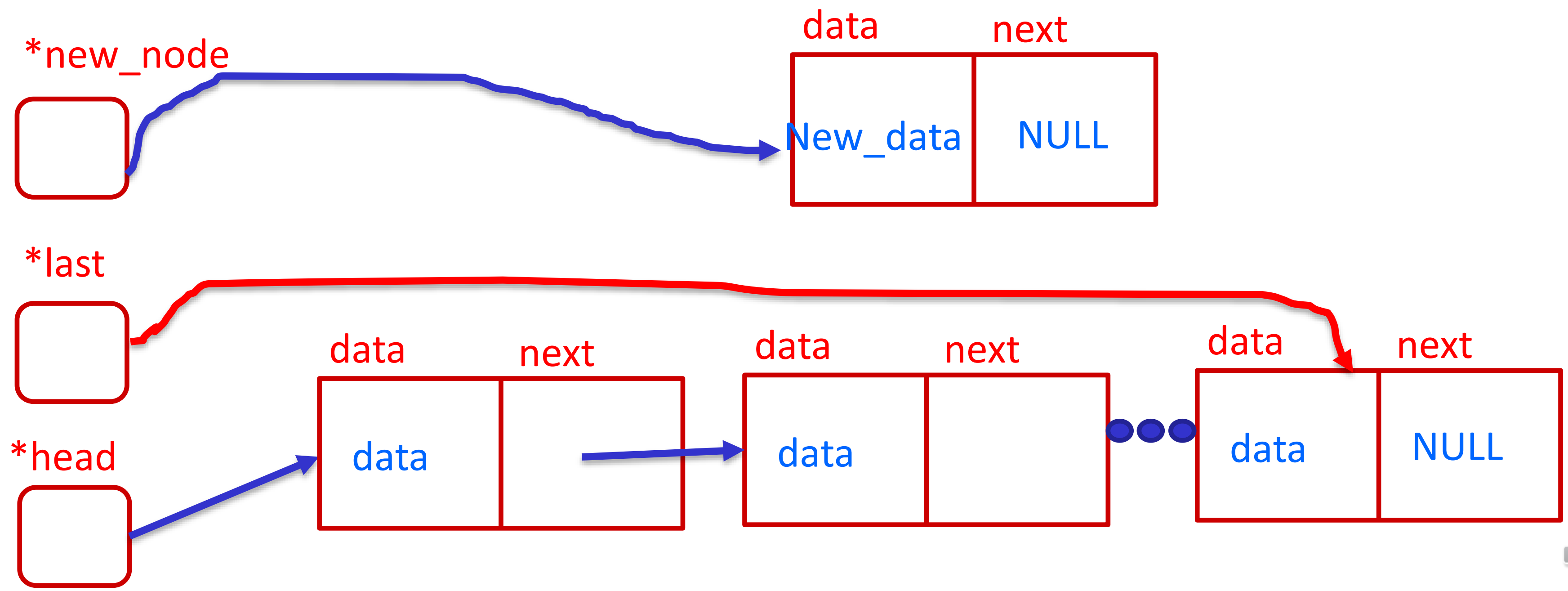
- /* 5. Else traverse till the last node */
-     while (last->next != NULL)
-         last = last->next;



## Appending a node

- /* 5. Else traverse till the last node */

```
while (last->next != NULL)
 last = last->next;
```

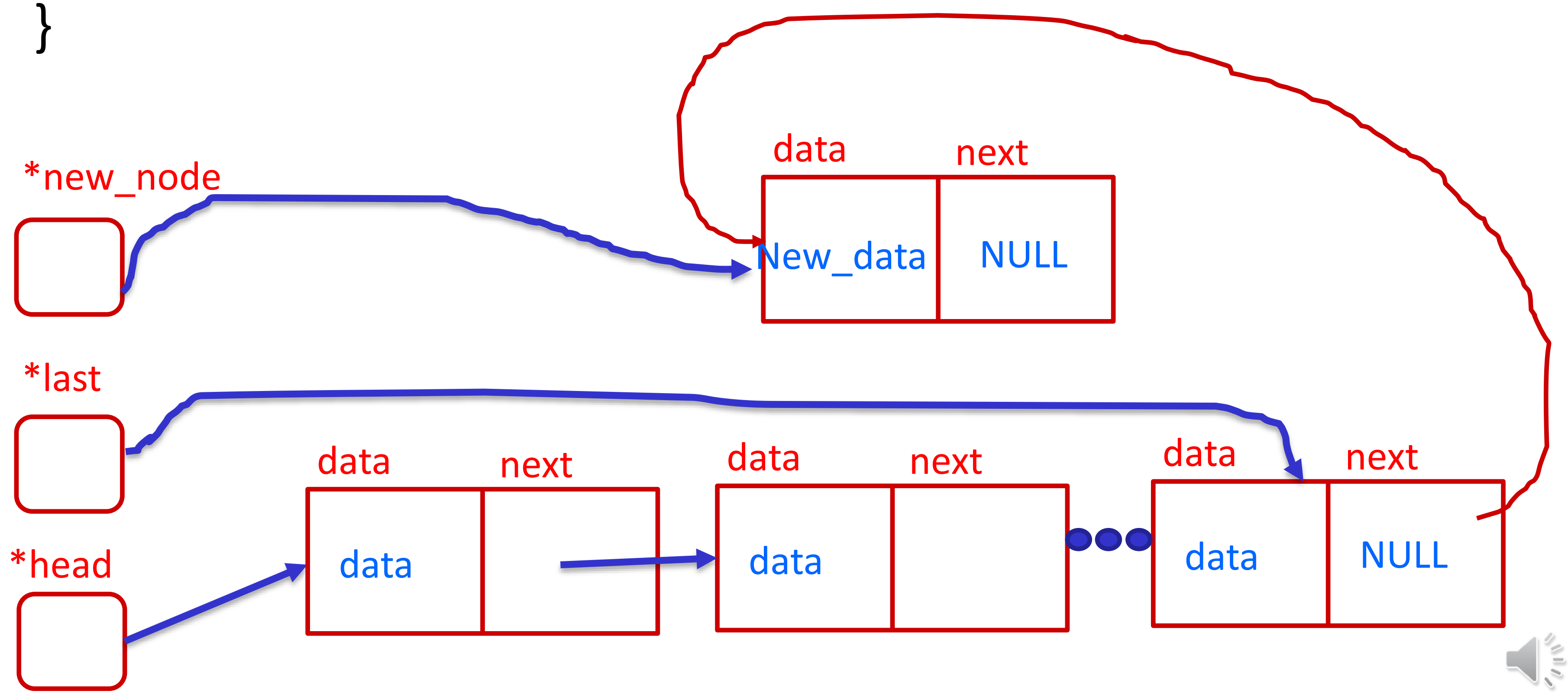


# Appending a node

- /* 6. Change the next of last node */

```
last->next = new_node;
return;
}
```

head* is global
new_node* and last* are local

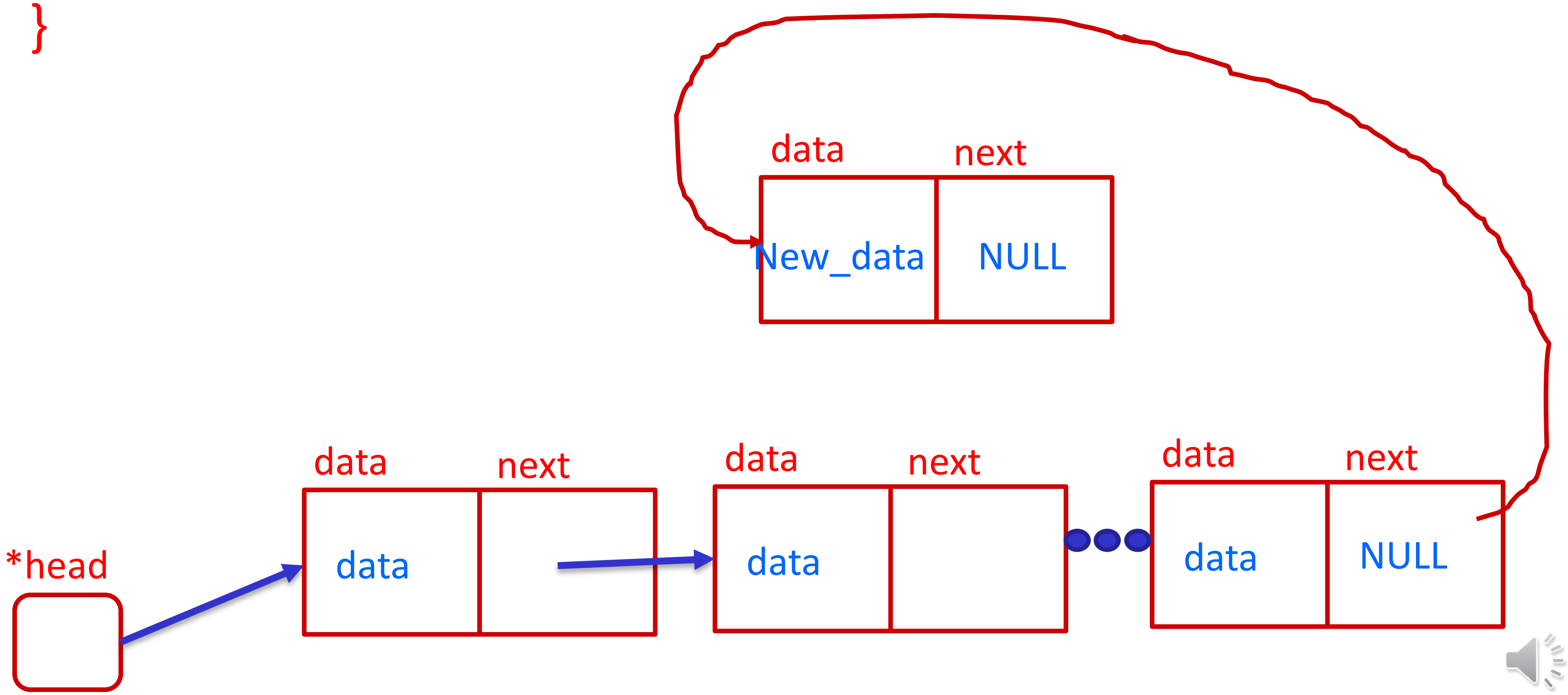


# Appending a node

- /* 6. Change the next of last node */

```
last->next = new_node;
return;
}
```

Only head* will remain



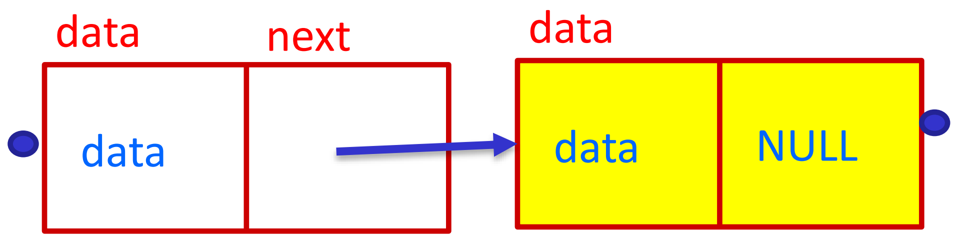
## Inserting a node after a given node

- Need pointer to previous node and new data as parameters
- void insert(int new_data, Node* prev_node)
- /*1. check if the given prev_node is NULL */

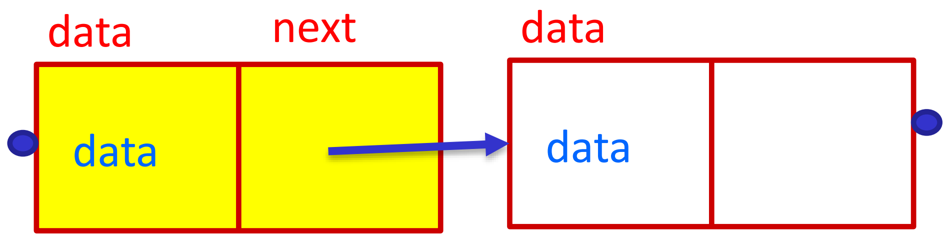
```
if (prev_node == NULL)
{
  printf("the given previous node cannot be NULL");
  return;
}
```

## Inserting a node after a given node

- If there exists a prev_node, there are two possibilities
  - Node at prev_node is the last  node

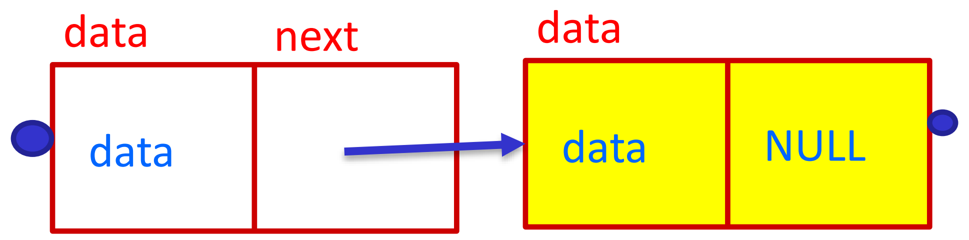


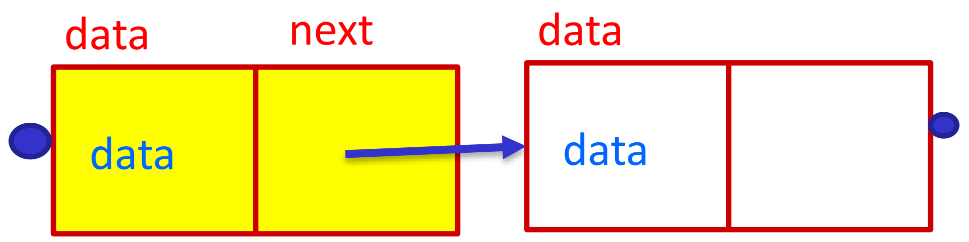
  - Or not



## Inserting a node after a given node

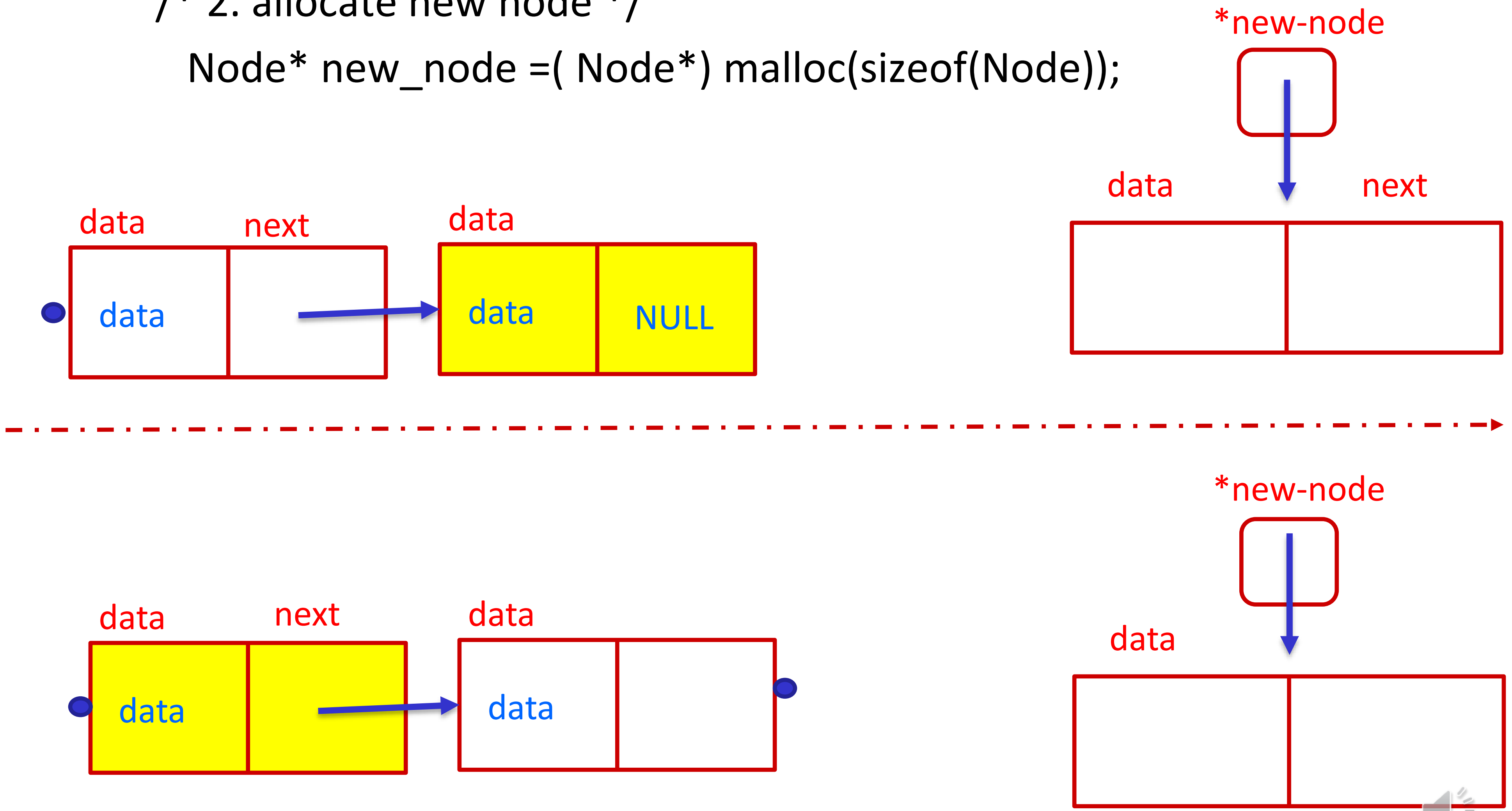
- But no need for if/else statement





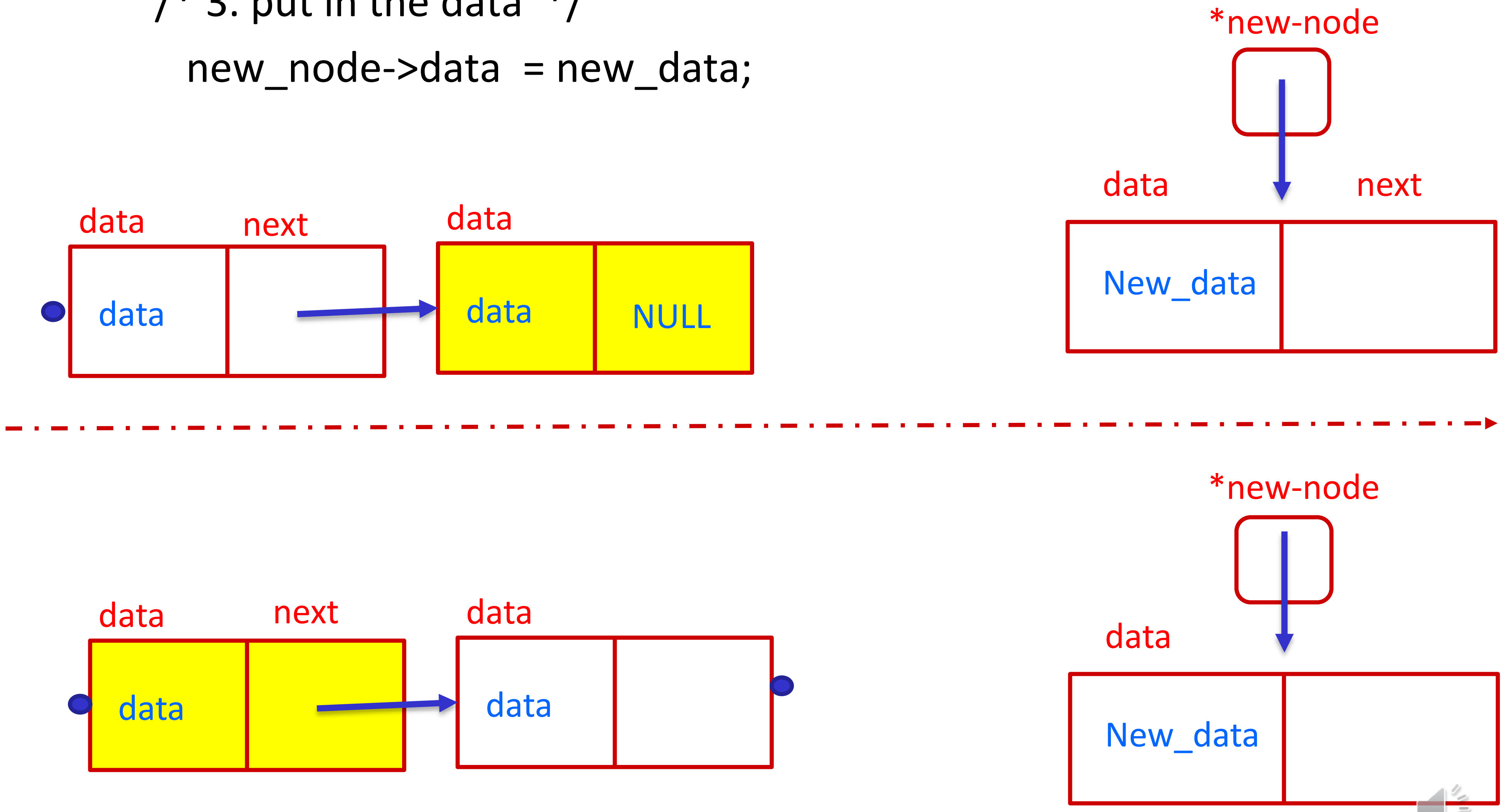
# Inserting a node after a given node

```
/* 2. allocate new node */
   Node* new_node =( Node*) malloc(sizeof(Node));
```



## Inserting a node after a given node

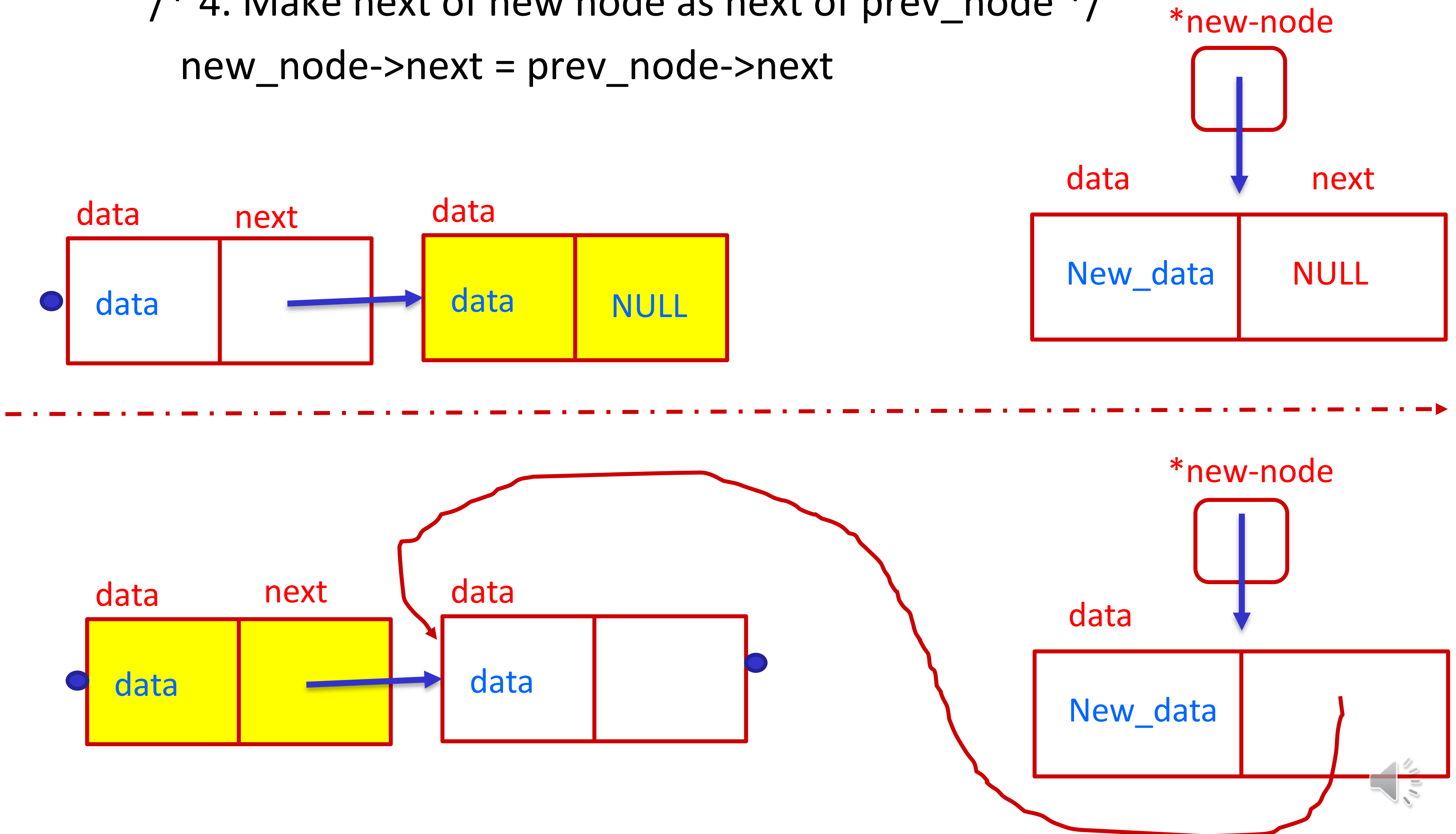
```
/* 3. put in the data  */
  new_node->data  = new_data;
```



## Inserting a node after a given node

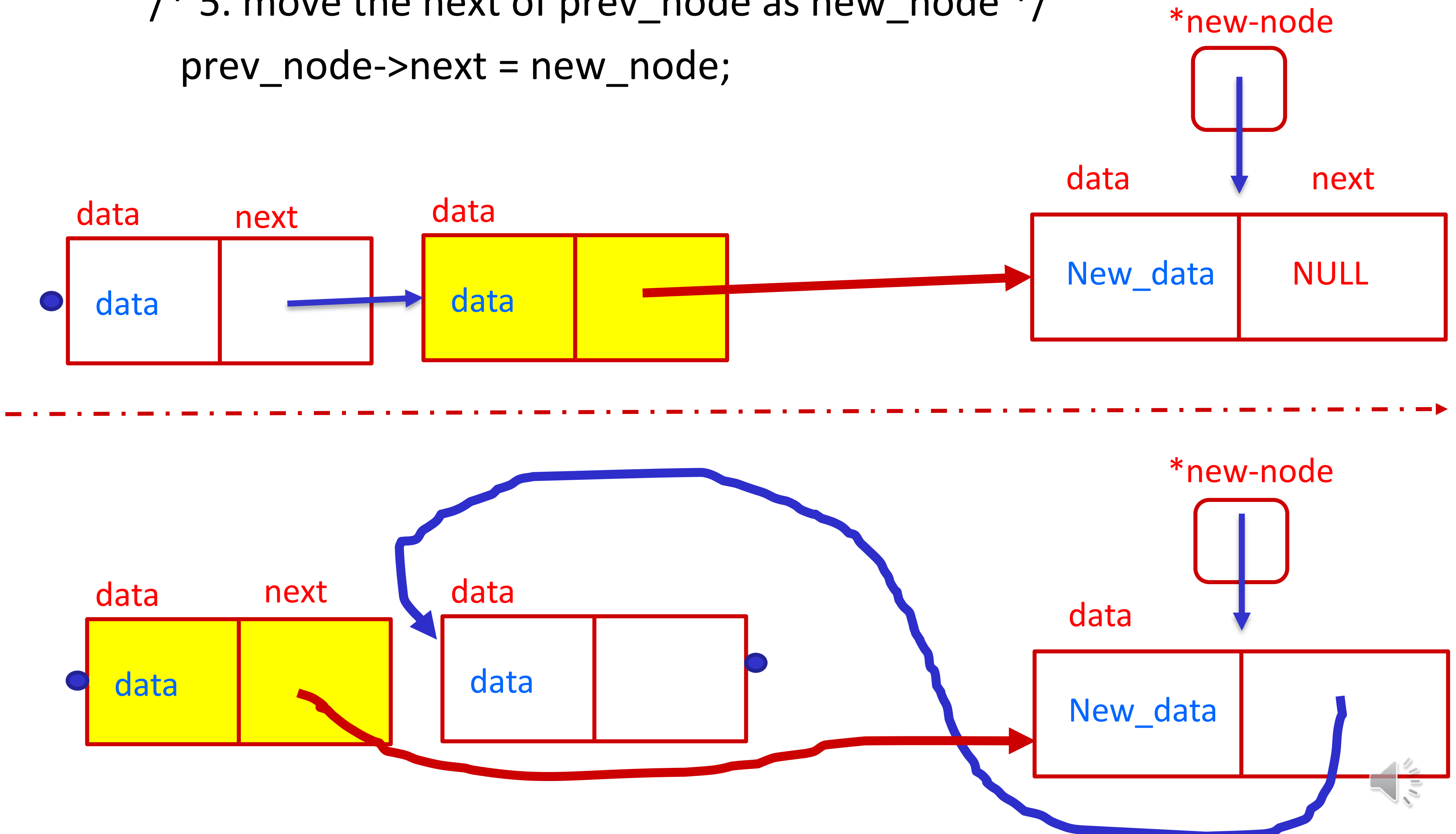
/* 4. Make next of new node as next of prev_node */

```
new_node->next = prev_node->next
```



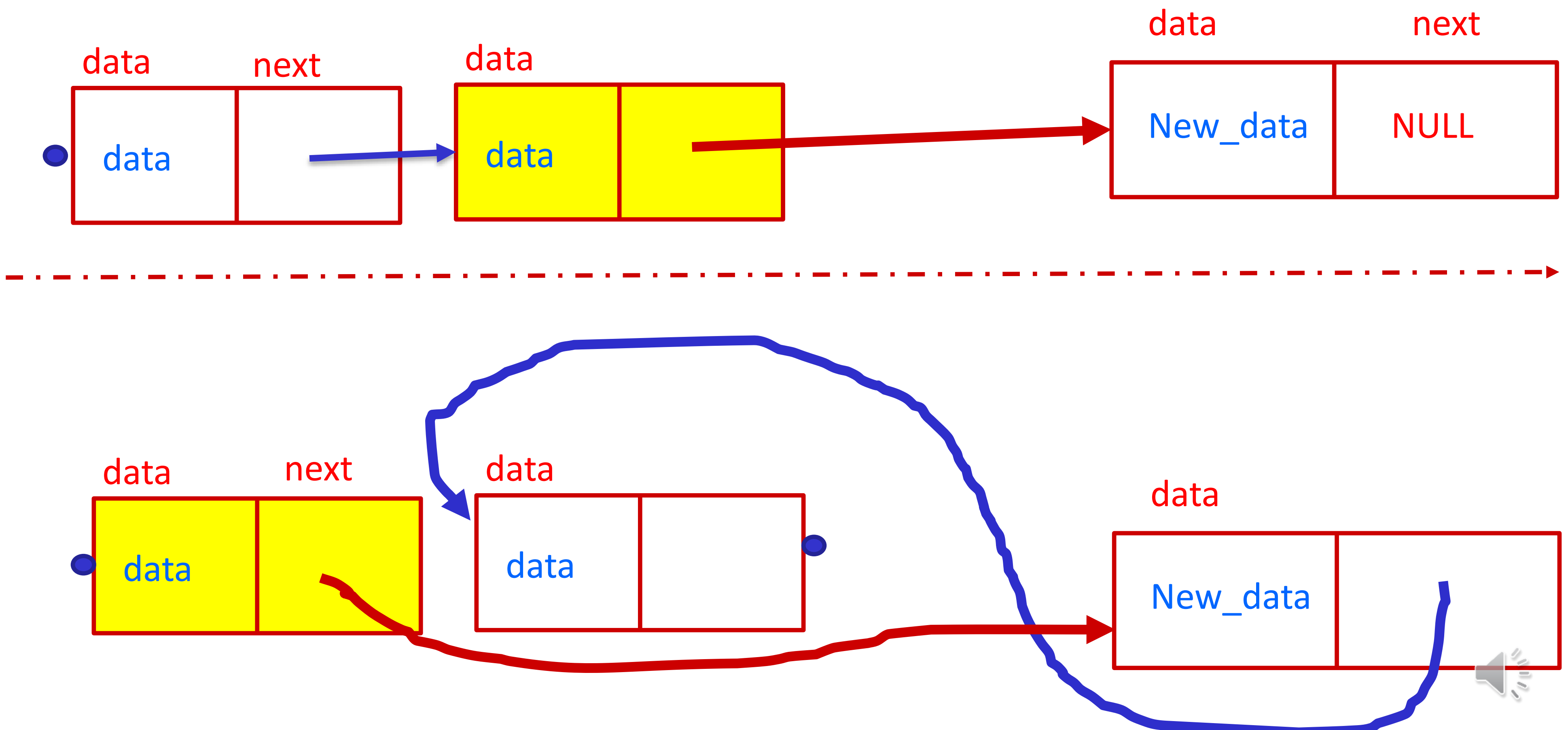
## Inserting a node after a given node

```
/* 5. move the next of prev_node as new_node */
  prev_node->next = new_node;
```



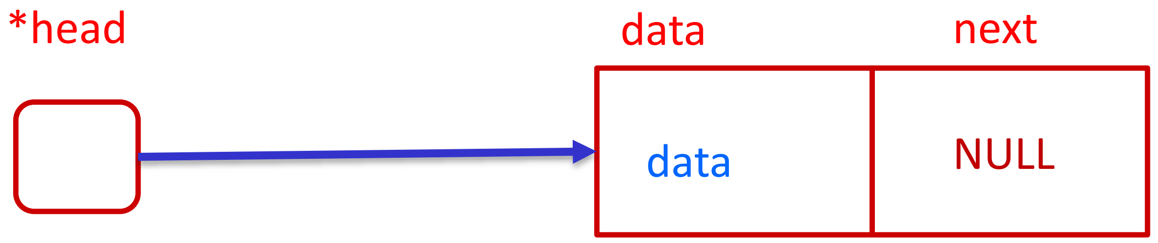
# Inserting a node after a given node

}

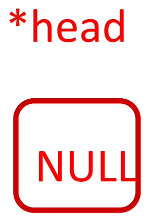


# Adding a node to the beginning (push)

- Two possibilities
  - List is not empty

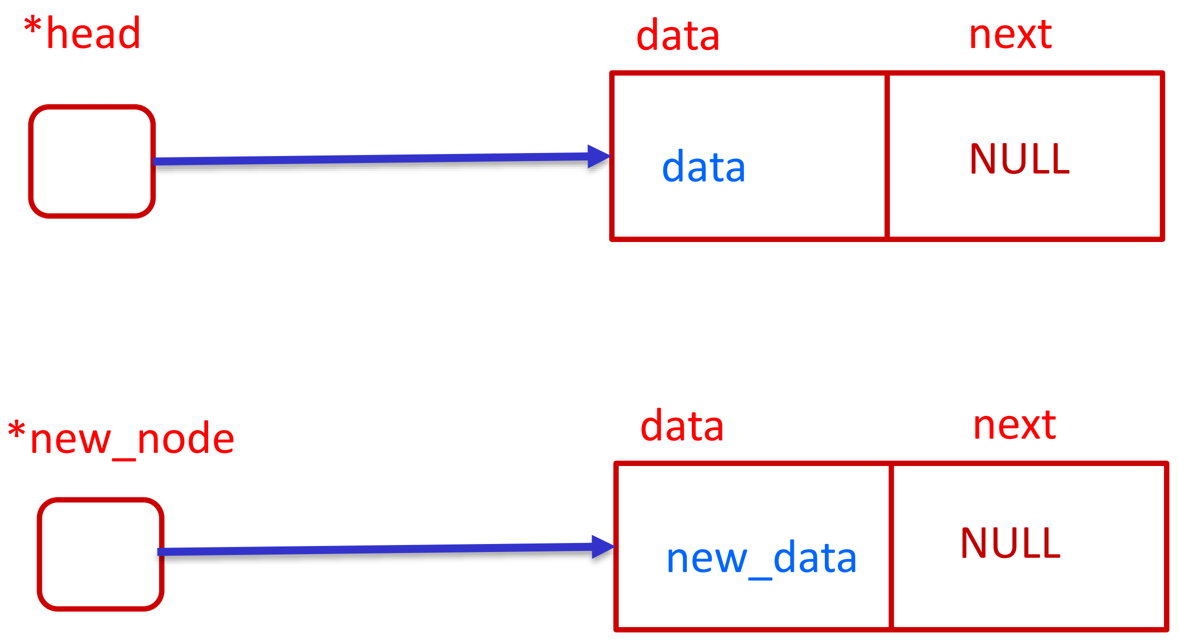


  - List is empty



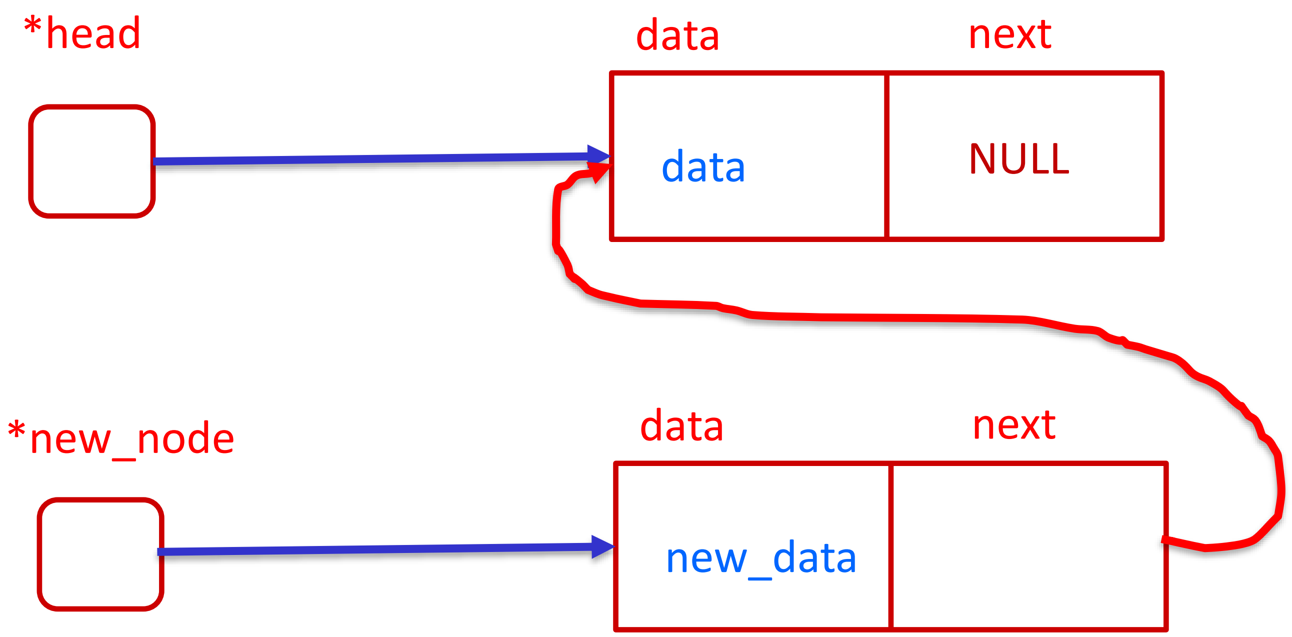
# Adding a node to the beginning

- Two possibilities
  - List is not empty



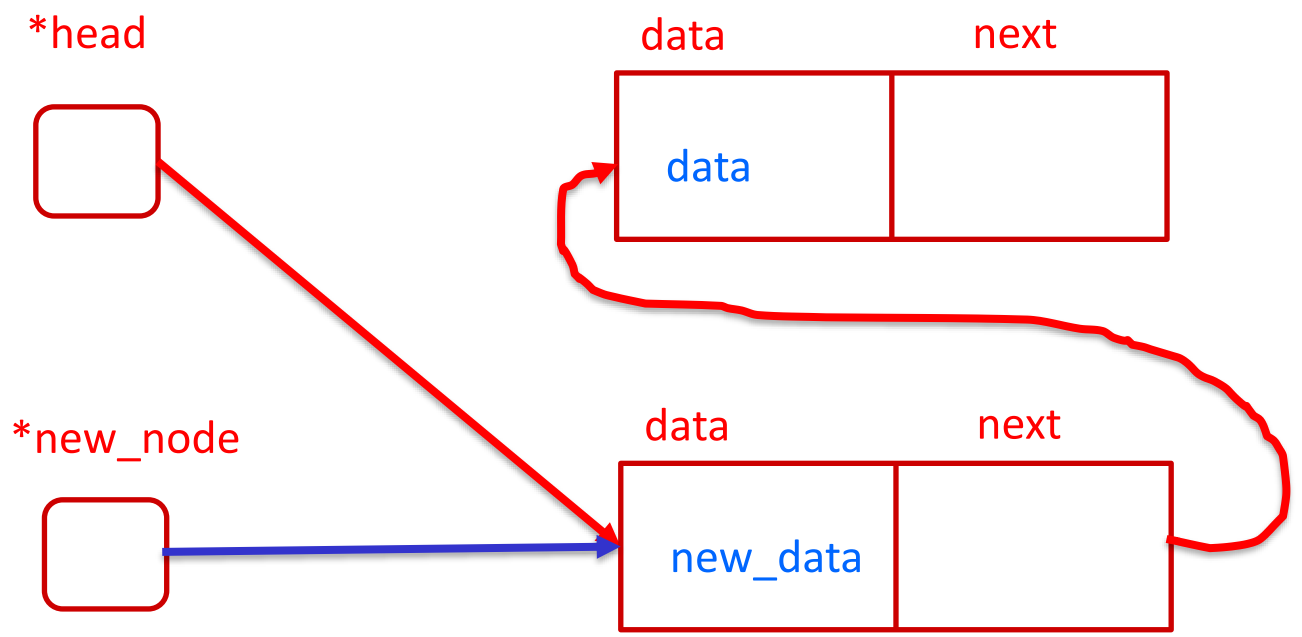
## Adding a node to the beginning

- Two possibilities
  - List is not empty



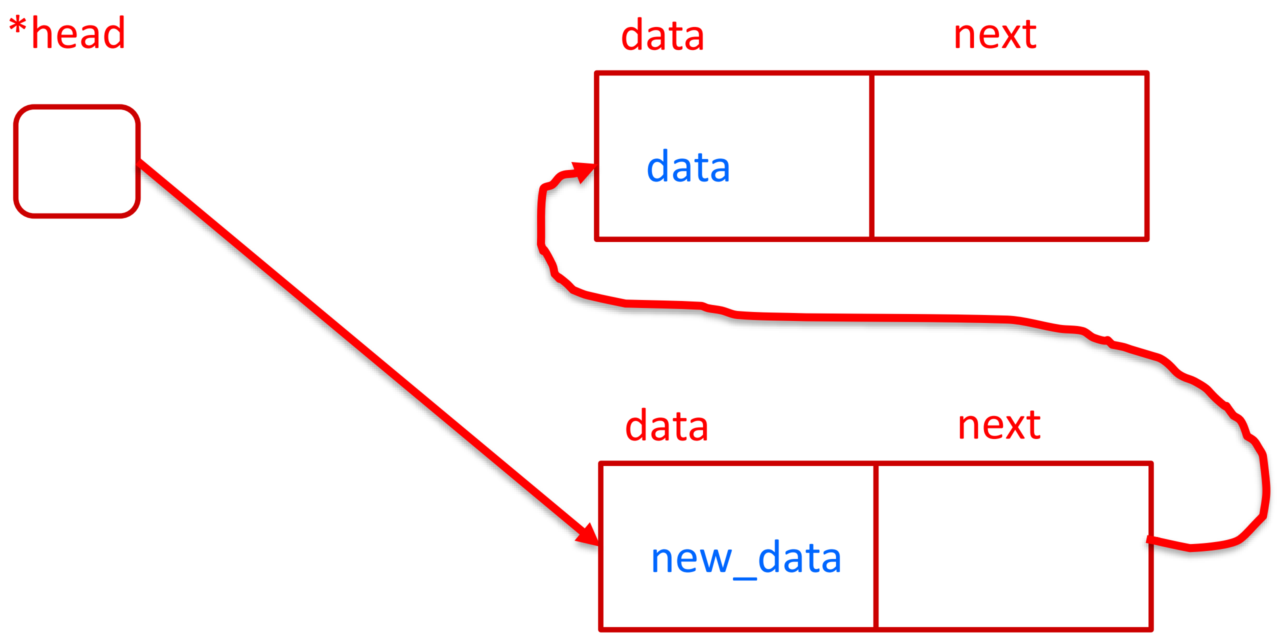
# Adding a node to the beginning

- Two possibilities
  - List is not empty



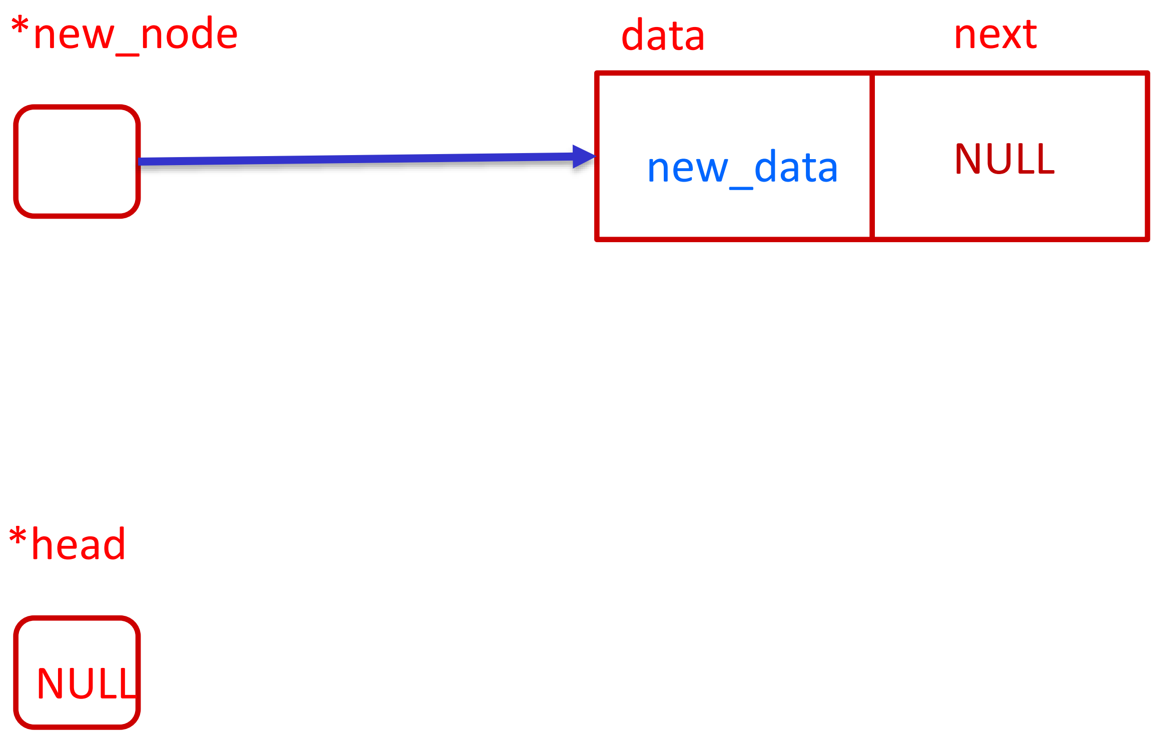
## Adding a node to the beginning

- Two possibilities
  - List is not empty



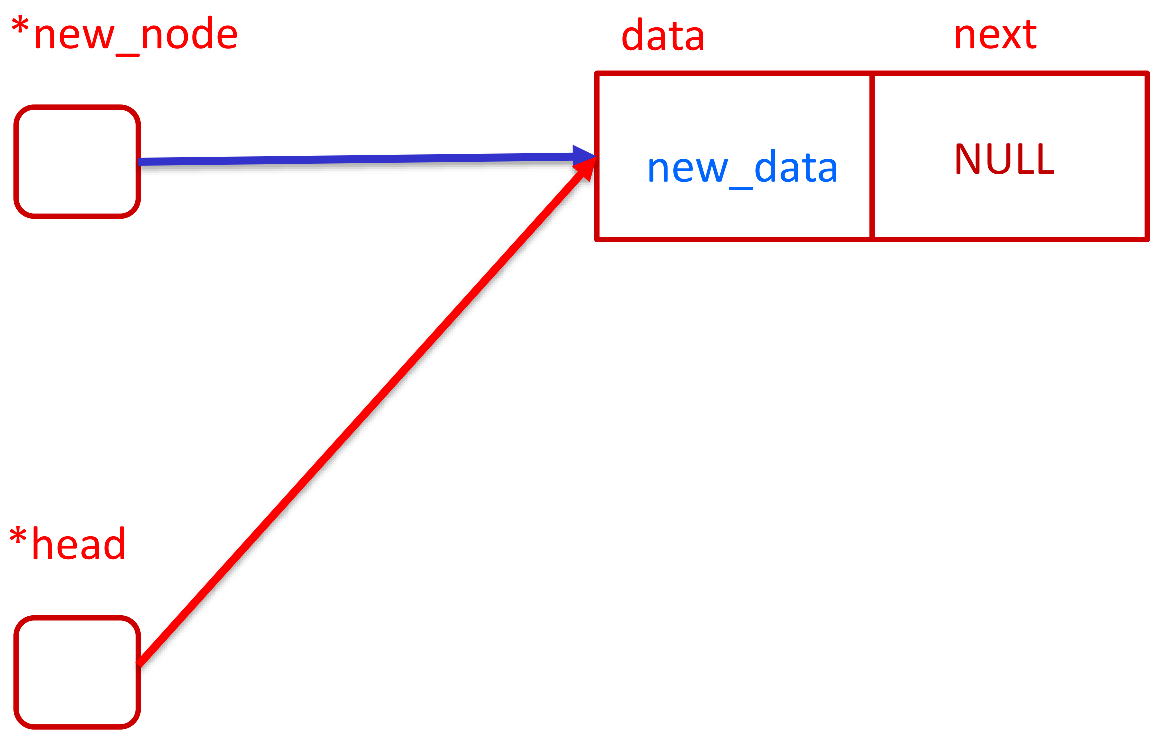
# Adding a node to the beginning (push)

- Two possibilities
  - List is empty



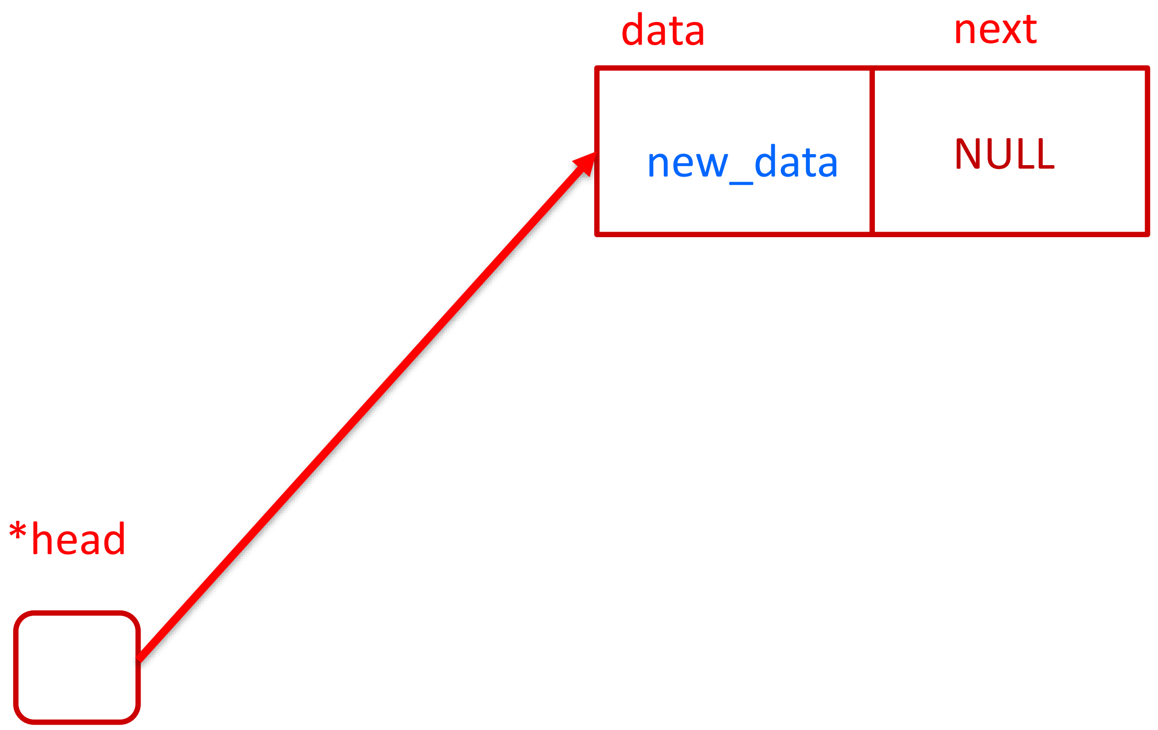
## Adding a node to the beginning (push)

- Two possibilities
  - List is empty



## Adding a node to the beginning (push)

- Two possibilities
  - List is empty



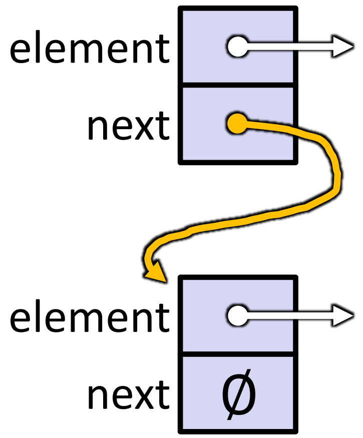
## Additional lab work

- Length of  linked list
- Delete first occurrence of a number

# A Generic Linked List

- Let's generalize the linked list element type
  - Let customer decide type (instead of always `int`)
  - Idea: let them use a generic pointer (*i.e.* a `void*`)

```
typedef struct node_st {
  void* element;
  struct node_st* next;
} Node;
Node* head;

void Push(void* e) {
  Node* n = (Node*) malloc(sizeof(Node));
  n->element = e;
  //…

}
```

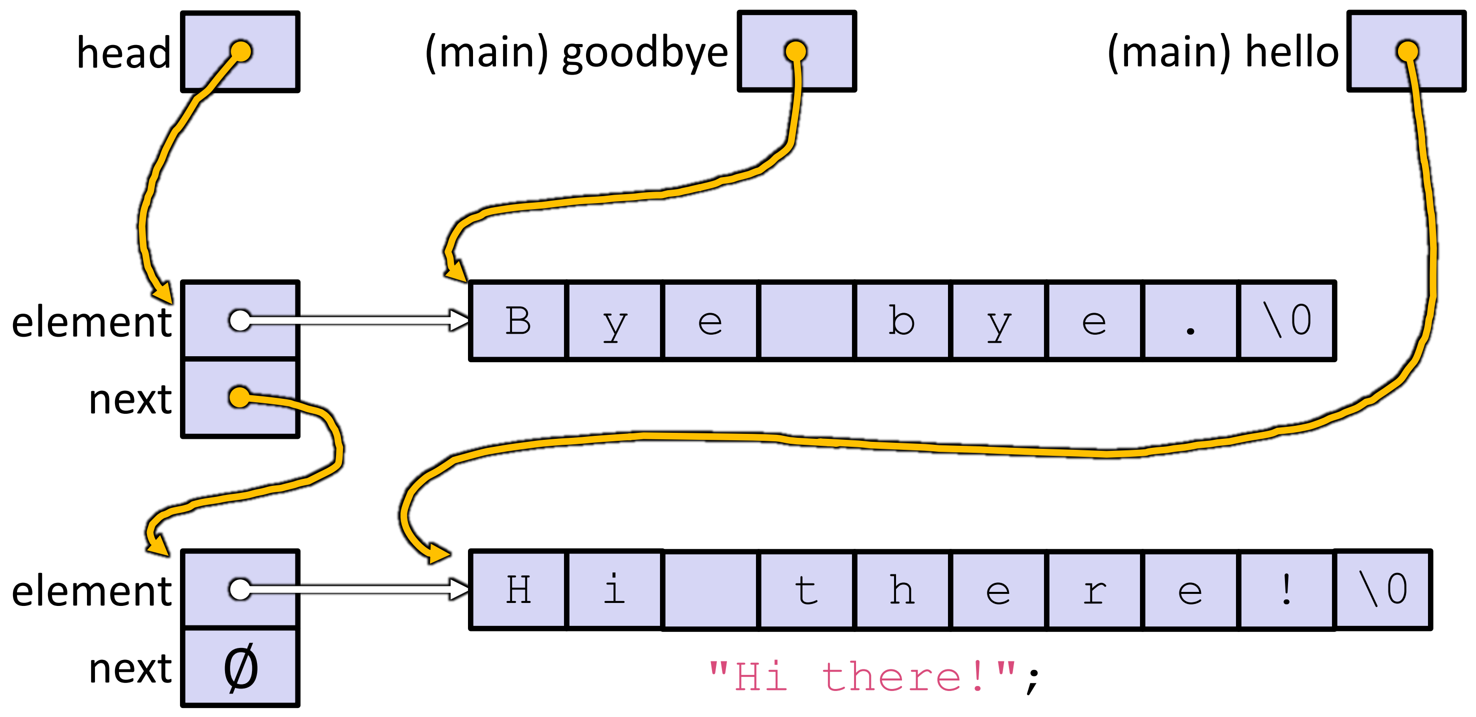


# Using a Generic Linked List

- Type casting needed to deal with `void*` (raw address)
  - Before pushing, need to convert to `void*`
  - Convert back to data type when accessing

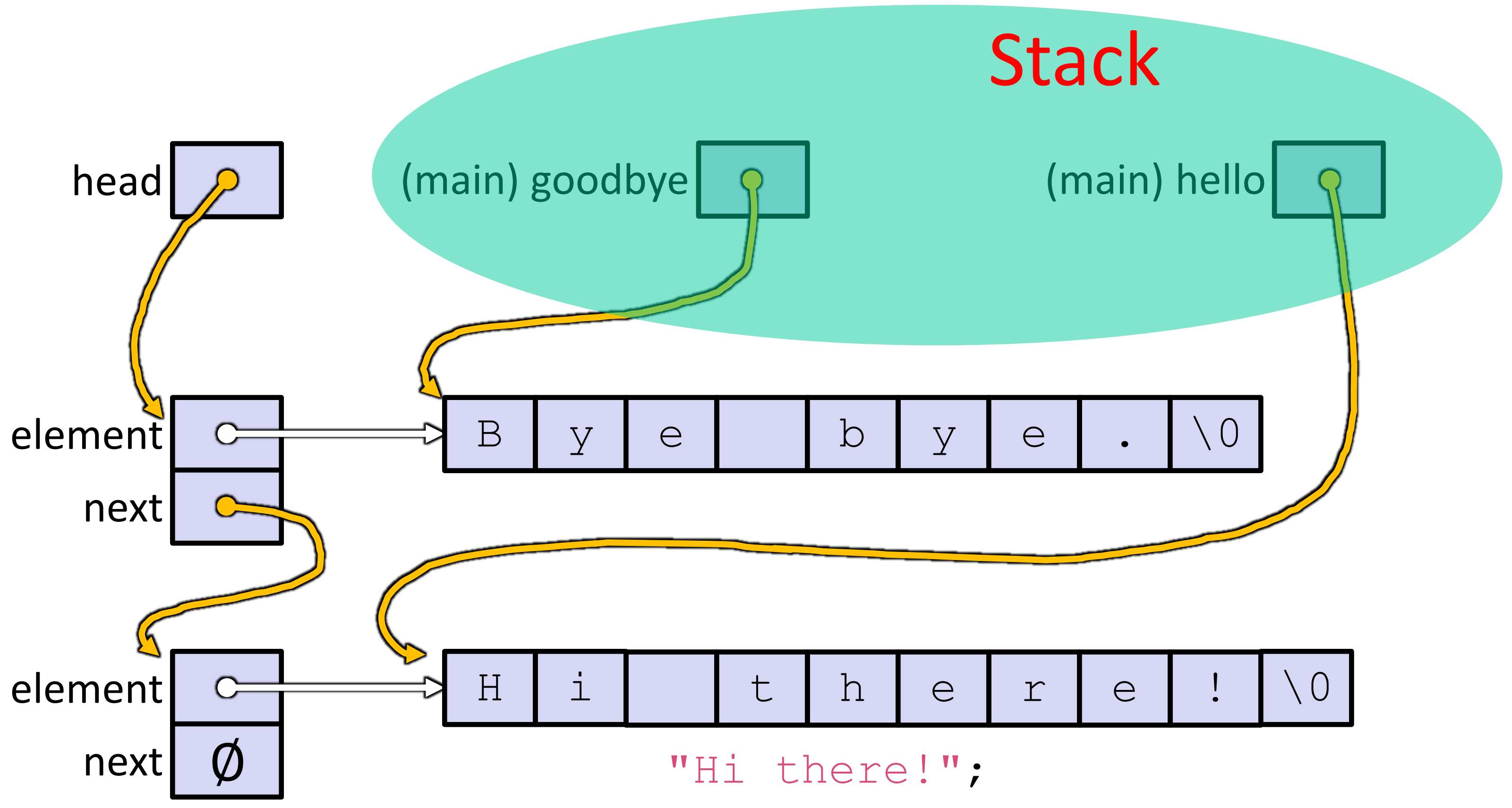
```
typedef struct node_st {
  void* element;
  struct node_st* next;
} Node;
Node* head;

void Push(void* e);    // assume last slide's code

int main(int argc, char** argv) {
  char* hello = "Hi there!";
  char* goodbye = "Bye bye.";
  head = NULL;
  Push((void*) hello);
  Push((void*) goodbye);
  printf("payload: '%s'\n", (char*) ((head->next)->element) );
  return 0;
}
```

# Resulting Memory Diagram



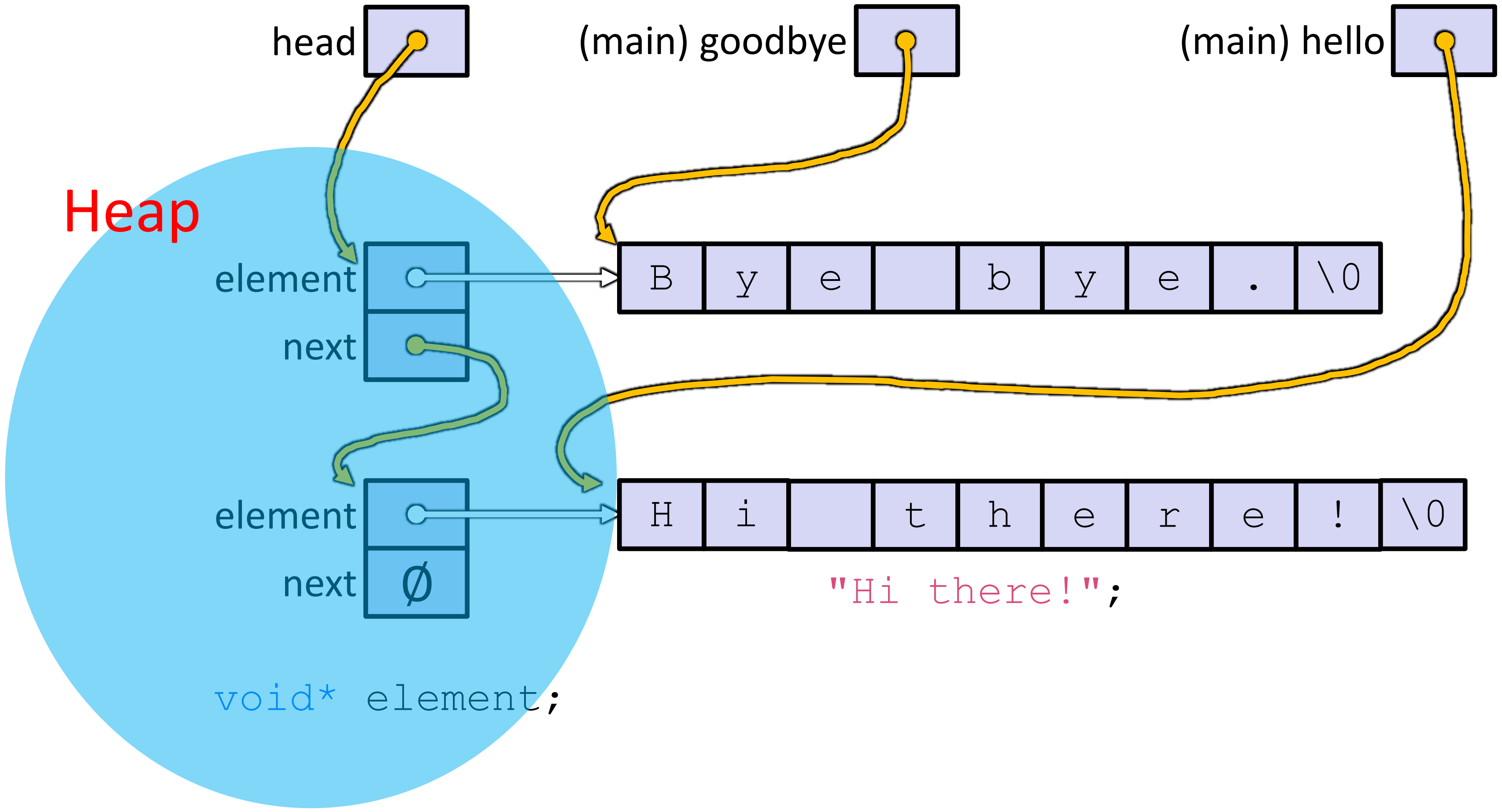
```
void* element;
```

# Resulting Memory Diagram

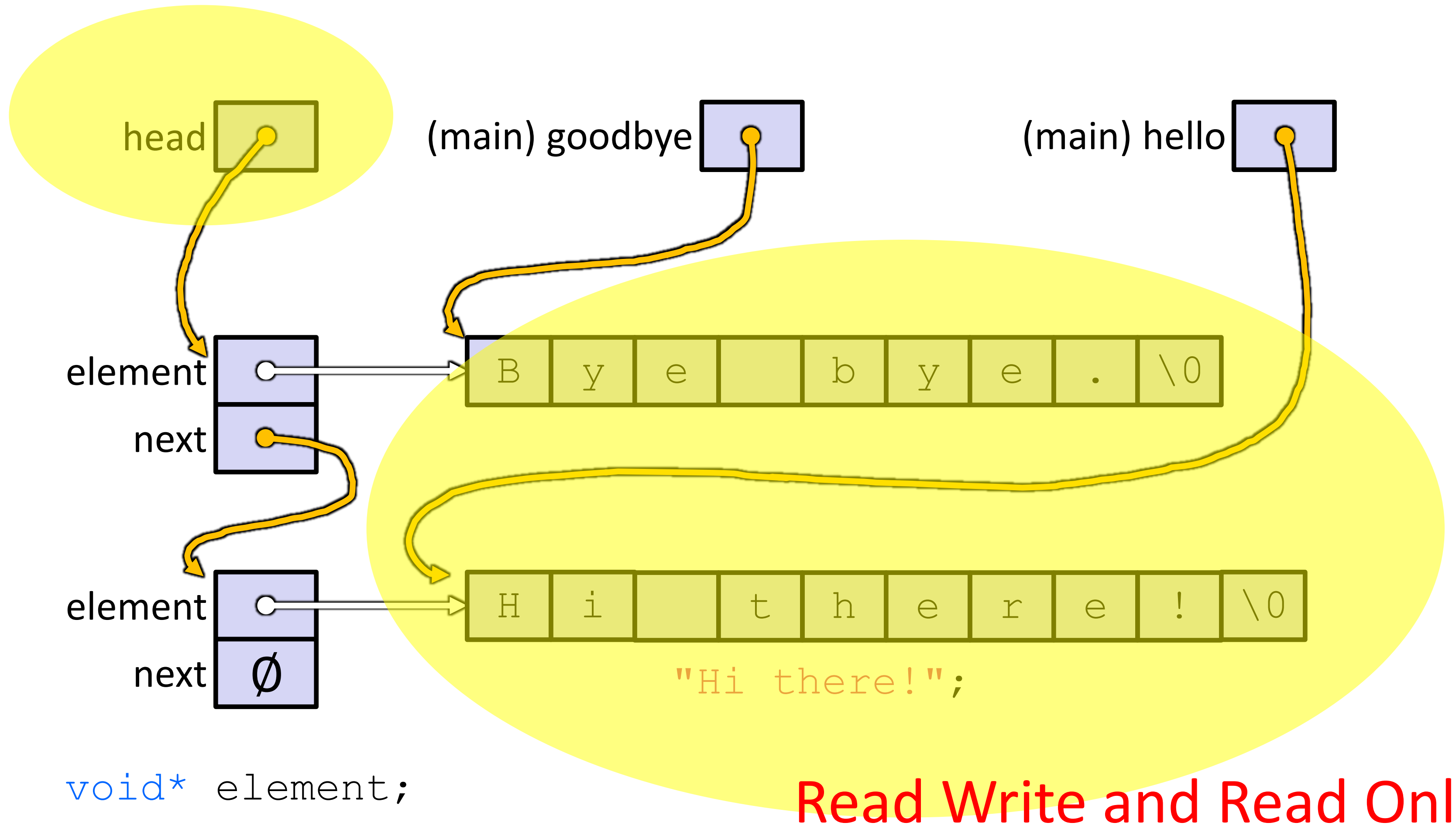


```
void* element;
```

## Resulting Memory Diagram



## Resulting Memory Diagram



```
void* element;
```

Read Write and Read Only Data

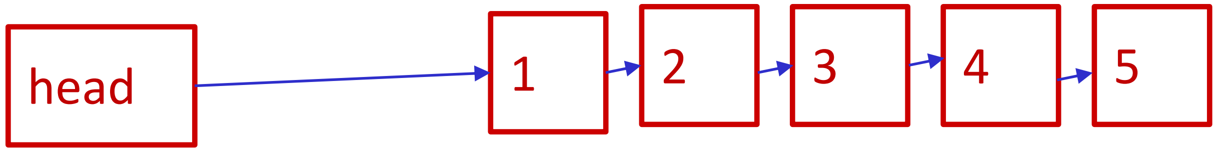
literals, globals

## Let’s do some exercise

This Photo by Unknown Author is licensed under CC BY-NC-ND

## Questions

- Linked list is organized as follows



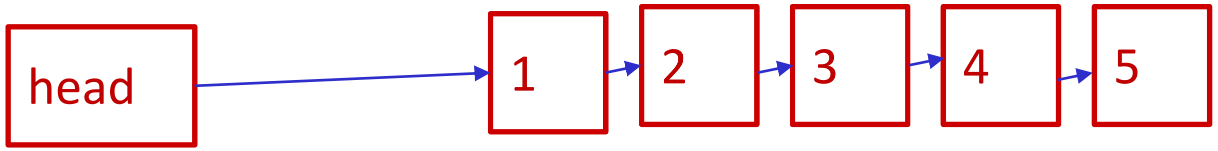
- What will be printed when fun1(head) is called?
- What will be printed when fun2(head) is called?
- Check the link for entire program https://ide.geeksforgeeks.org/lut8zH5XnG

```
void fun1(struct Node* head)
{
    if(head == NULL)
        return;

    fun1(head->next);
    printf("%d ", head->data);
}
```

```
void fun2(struct Node* start)
{
    if(start == NULL)
        return;
    printf("%d ", start->data);

    if(start->next != NULL )
        fun2(start->next->next);
    printf("%d ", start->data);
}
```

# Questions

- Linked list is organized as follows



- What will be printed when fun1(head) is called?
- What will be printed when fun2(head) is called?
- Check the link for entire program https://ide.geeksforgeeks.org/lut8zH5XnG

```
void fun1(struct Node* head)
{
    if(head == NULL)
        return;

    fun1(head->next);
    printf("%d ", head->data);
}
```

```
void fun2(struct Node* start)
{
    if(start == NULL)
        return;
    printf("%d ", start->data);

    if(start->next != NULL )
        fun2(start->next->next);
    printf("%d ", start->data);
}
```