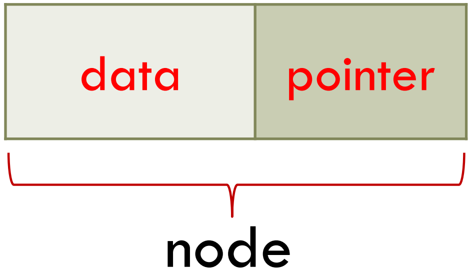
HIGH-LEVEL PROGRAMMING 2

Linked Lists

by Prasanna Ghali

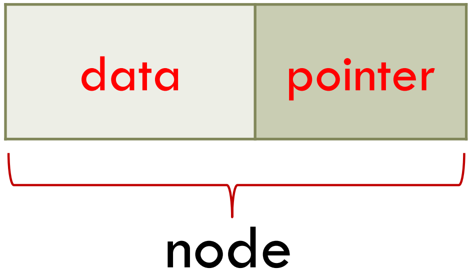
What is a Linked List? (1/2)

- Linked list is organized as group of dynamically allocated elements that are connected by pointers
- Element consists of *data* [some values encapsulated as a structure] and a *pointer* [to the next node in linked list]
- Linked list element commonly called *node*



What is a Linked List? (2/2)

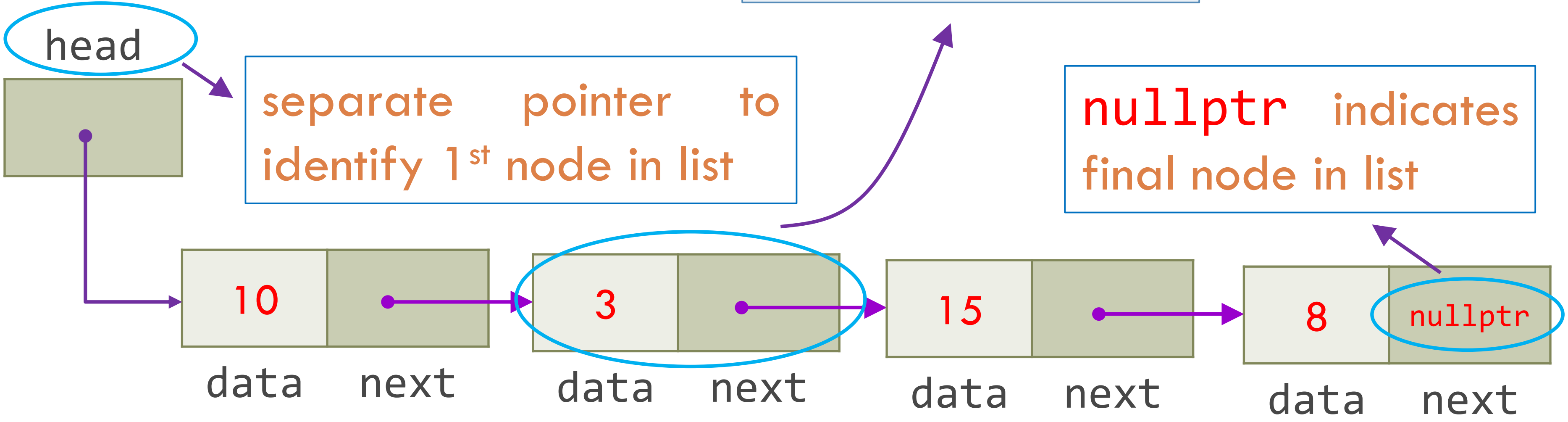
- Data in node can be anything
  - Single value or multiple values
  - Any type of object whose size is known at compile time
  - Includes `struct`, `class`, `union`, `char*` or other pointers
  - Also could be array of fixed size



Linked List of int Nodes

- □ Let's visualize *singly-linked list* creation for small number of nodes assuming each node encapsulates int data

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

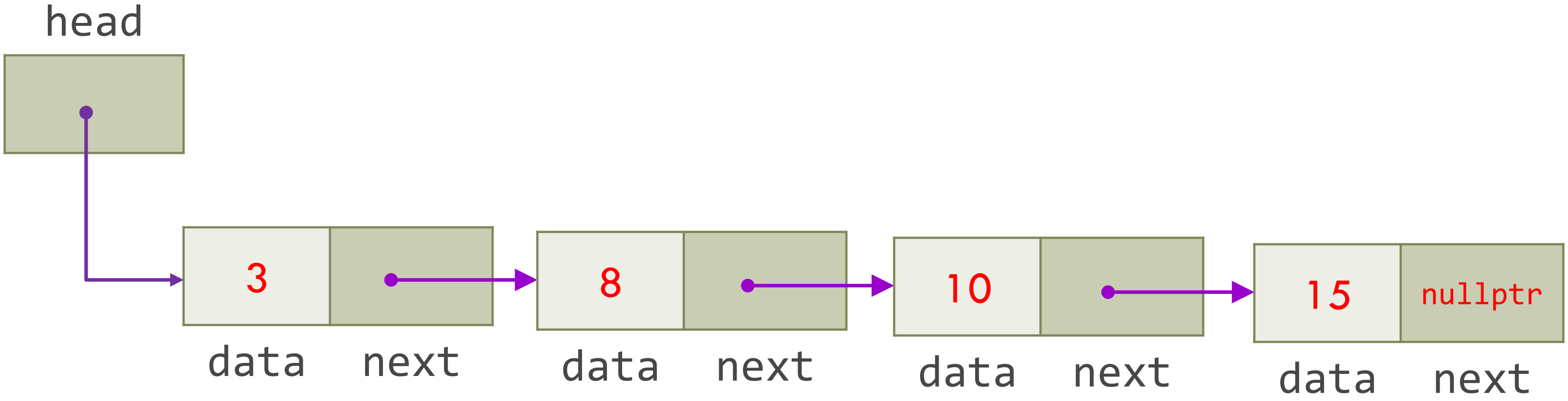


separate pointer to identify 1st node in list

nullptr indicates final node in list

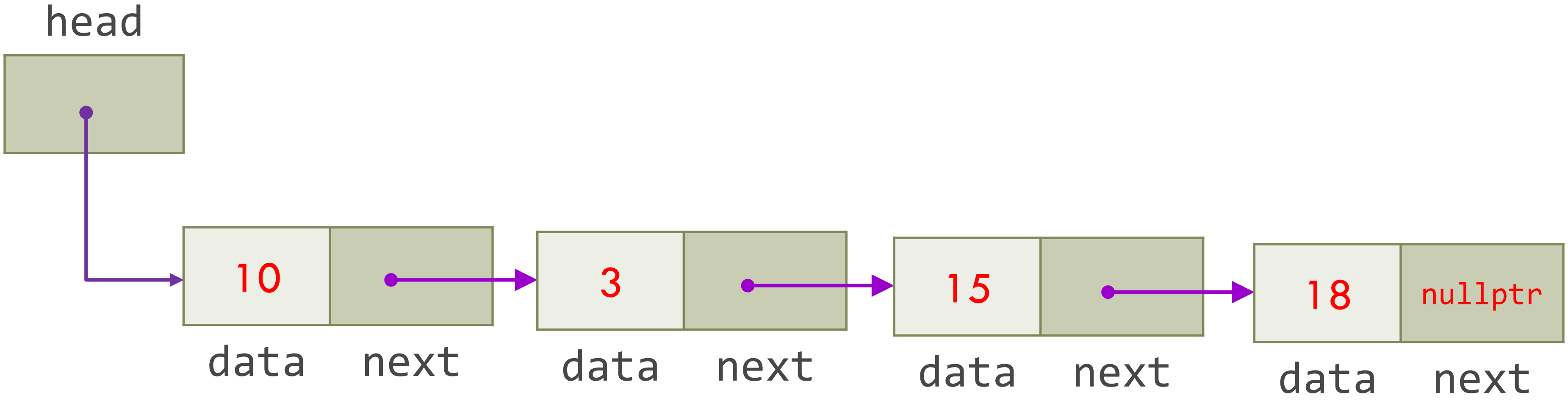
Usage of Linked Lists (1/6)

- Unlike array, linked list can store elements without need for contiguous memory
  - However lack of random access makes finding specific element more expensive than array
  - Linear search is tolerable for small data
- Sorting not necessary if list is ordered



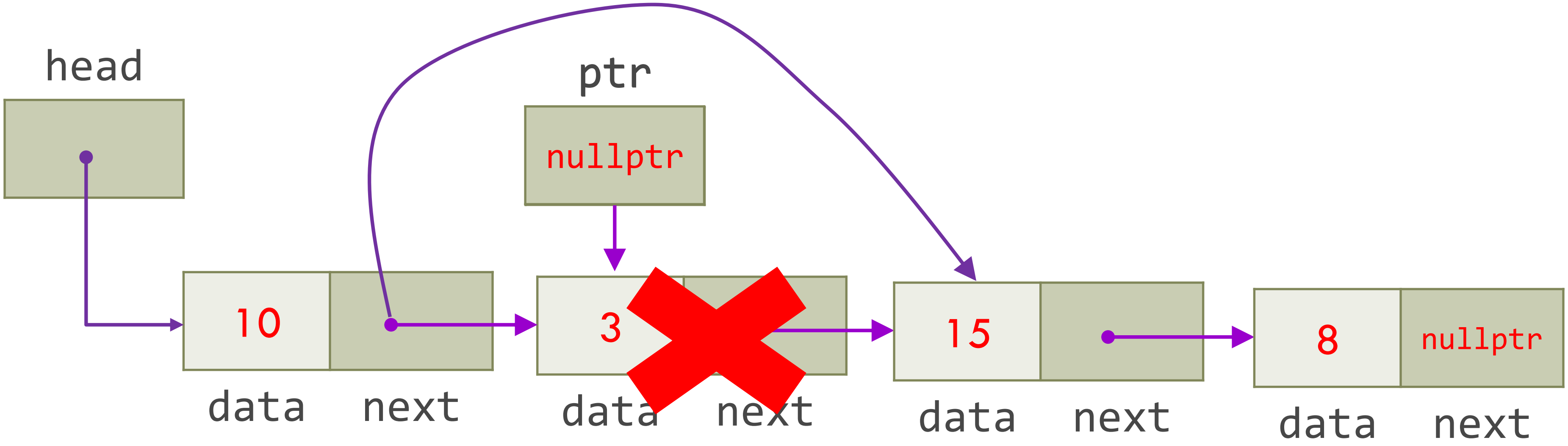
Usage of Linked Lists (2/6)

- Shines when nodes are inserted or deleted “on the fly” from anywhere in list



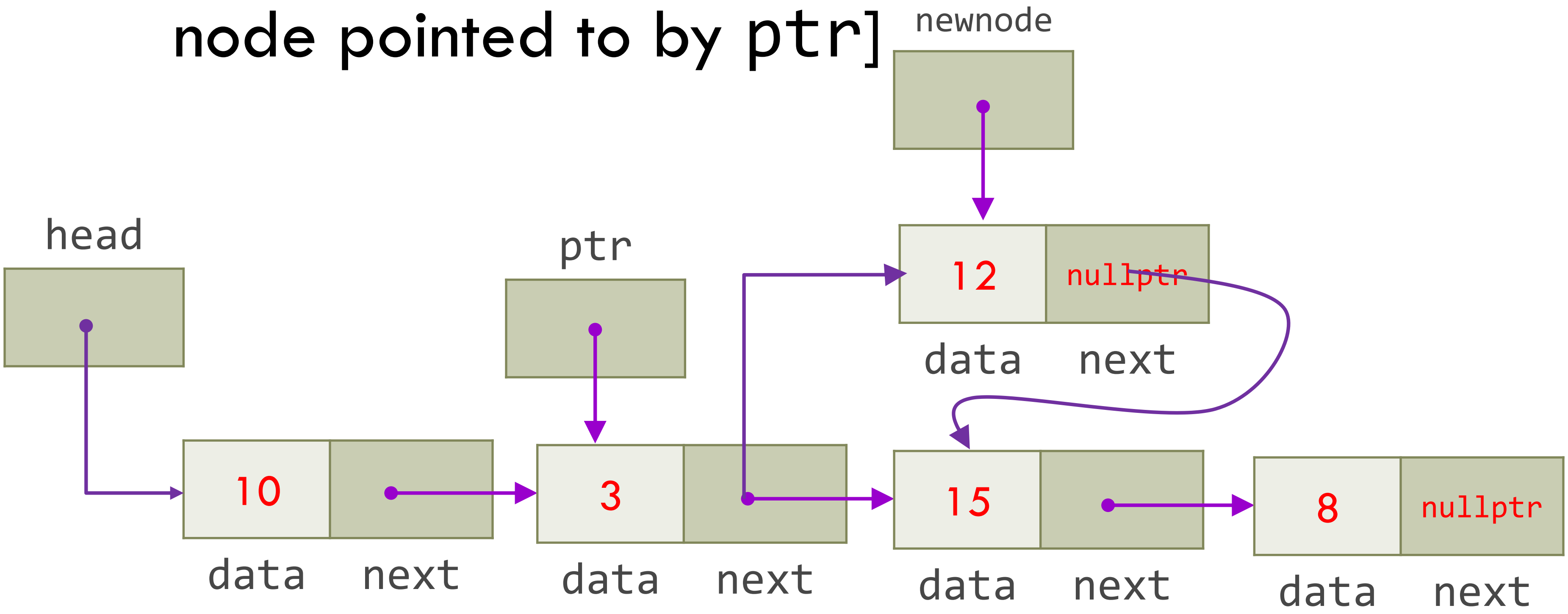
Usage of Linked Lists (3/6)

- Shines when nodes are inserted or deleted "on the fly" from anywhere in list
- **Let's see how to delete node "on the fly"** [the node pointed to by `ptr`]



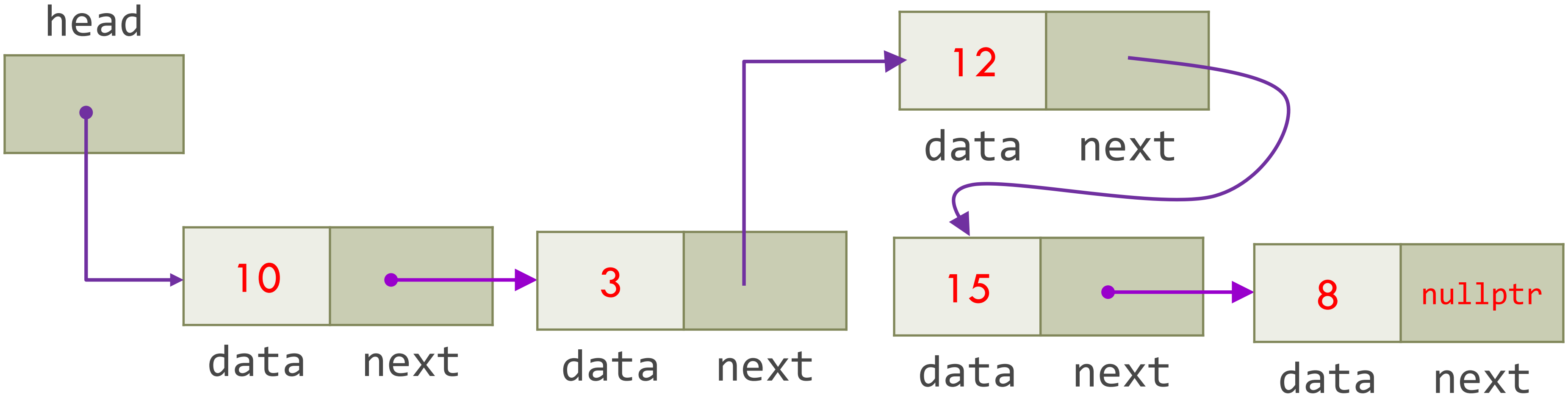
Usage of Linked Lists (4/6)

- □ Shines when nodes are inserted or deleted "on the fly" from anywhere in list
- □ Let's see how to insert node "on the fly" [after node pointed to by `ptr`]



Usage of Linked Lists (5/6)

- Shines when nodes are inserted or deleted "on the fly" from anywhere in list
- Let's see how to insert node "on the fly" [after node pointed to by `ptr`]



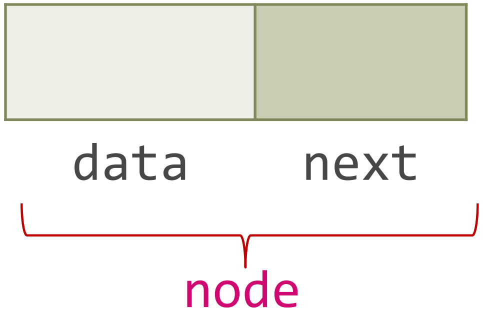
Usage of Linked Lists (6/6)

- *Doubly-linked list* is variation of singly-linked list
- Both types are used to create more advanced data structures such as *stack*, *queue*, *circular list*, …

Linked List Node Definition

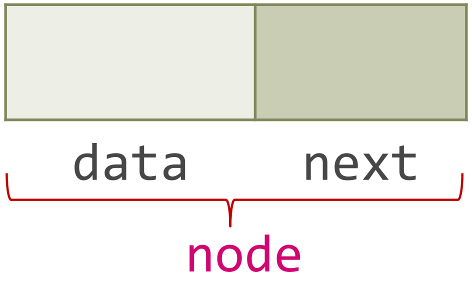
- Each node of linked list represented by following structure

```
struct node {
   type data;
   node *next;
};
```

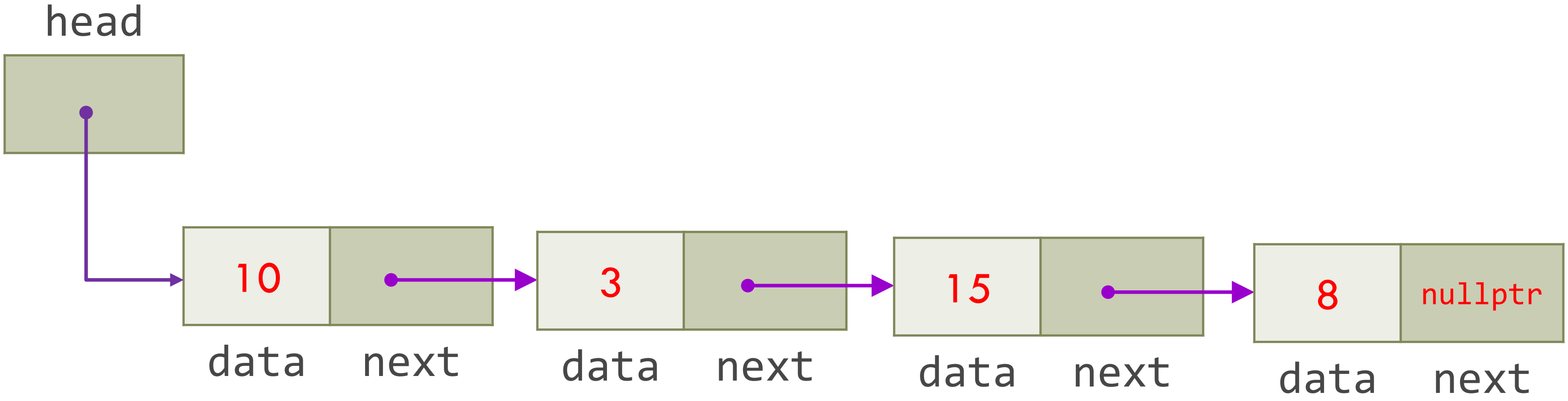


Linked List of int Nodes

- Here each node encapsulates int data



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



Inserting Value

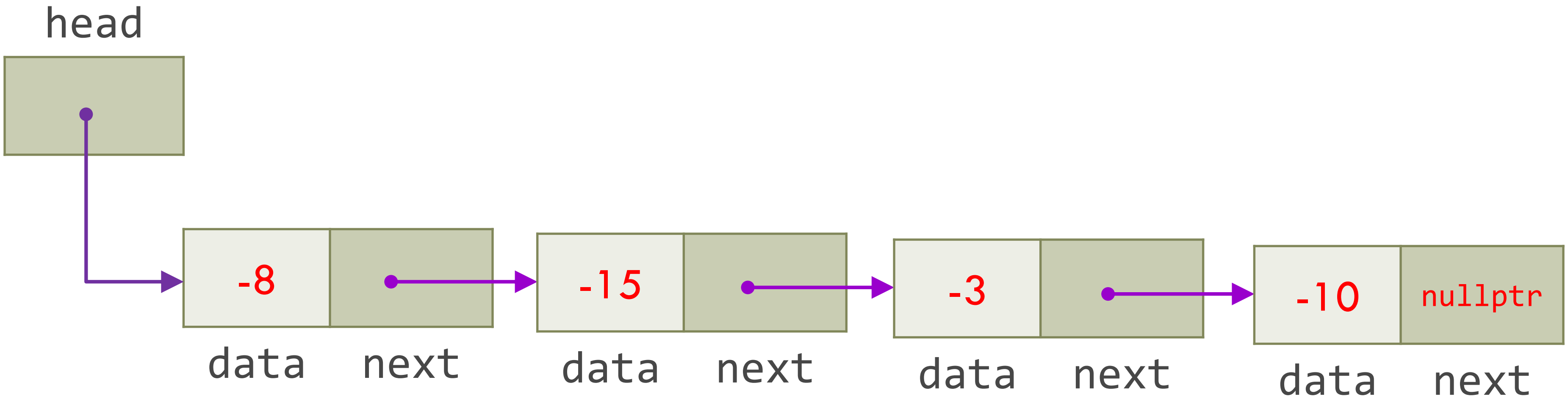
- Desired insertion location can be one of three places: in front of first node (`push_front`); after last node (`push_back`); after specific position (`insert_after`)
- In all cases, you've to worry about empty linked list

Inserting Value: push_front

- Client wants to do this:

```
node *head{nullptr};

push_front(&head, -10);
push_front(&head, -3);
push_front(&head, -15);
push_front(&head, -8);
```

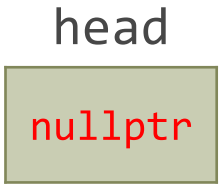
- Singly-linked list interface should do this:



Inserting Value: push_front

- We’ve defined head:

```
node *head{nullptr};
```

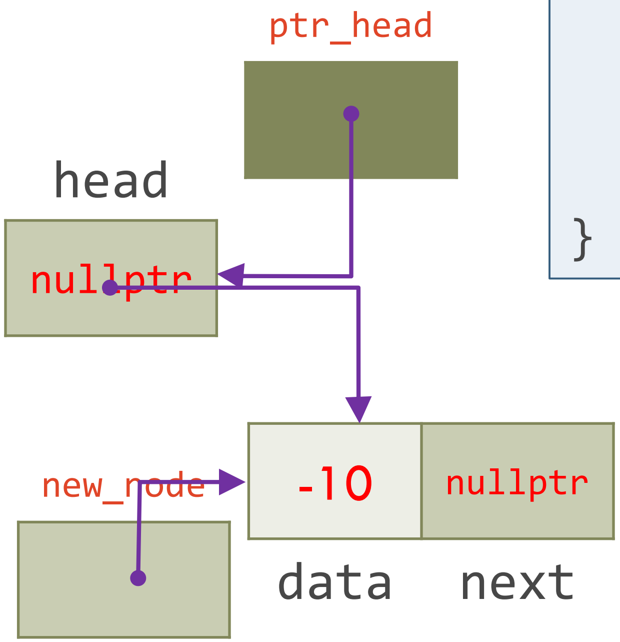


Inserting Value: push_front

- We make call to push_front:

```
node *head{nullptr};
push_front(&head, -10);
```

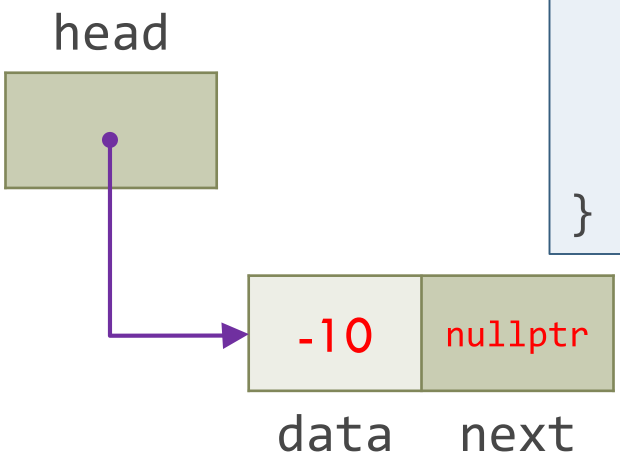
```
void push_front(node **ptr_head, int value) {
  node *new_node {new node{value, nullptr}};

  if (*ptr_head) {
    new_node->next = *ptr_head;
  }
  *ptr_head = new_node;
}
```



Inserting Value: push_front

- □ After 1st call to push_front:

```
node *head{nullptr};
push_front(&head, -10);
```



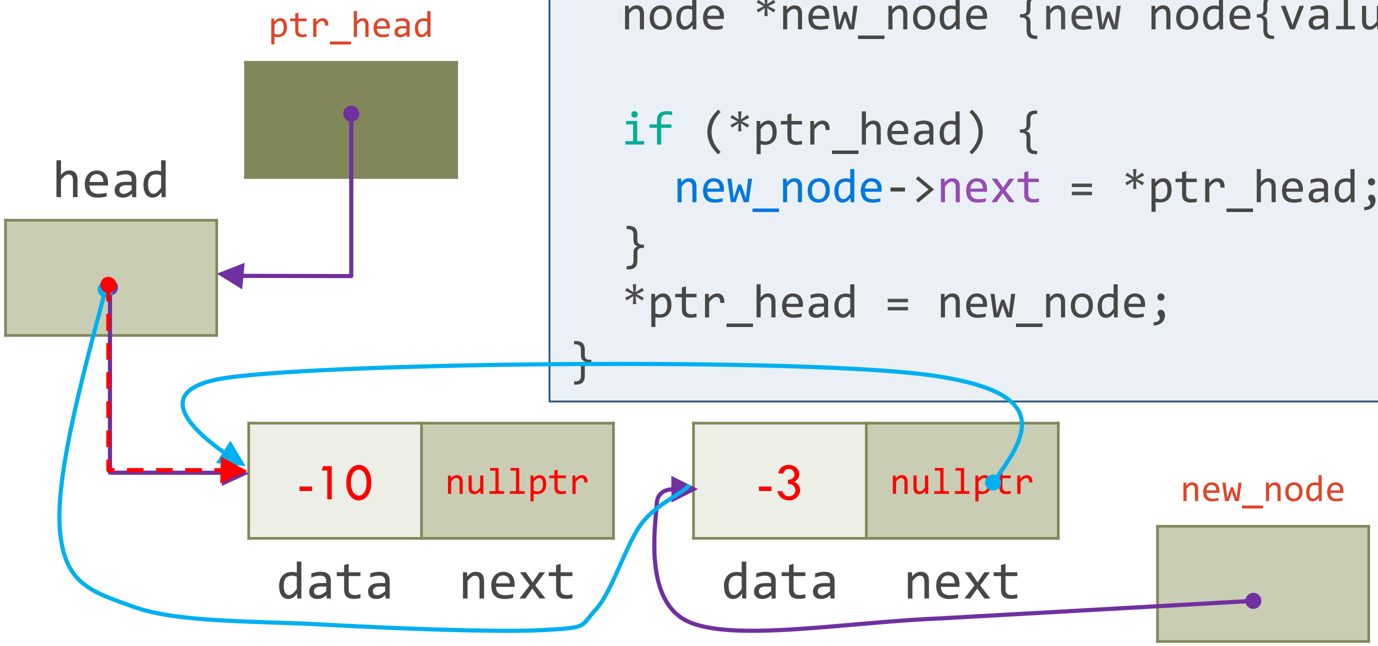
```
void push_front(node **ptr_head, int value) {
  node *new_node {new node{value, nullptr}};

  if (*ptr_head) {
    new_node->next = *ptr_head;
  }
  *ptr_head = new_node;
}
```

Inserting Value: push_front

- We make 2nd call to push_front:

```
node *head{nullptr};
push_front(&head, -10);
push_front(&head, -3);
```

```
void push_front(node **ptr_head, int value) {
  node *new_node {new node{value, nullptr}};

  if (*ptr_head) {
    new_node->next = *ptr_head;
  }
  *ptr_head = new_node;
}
```

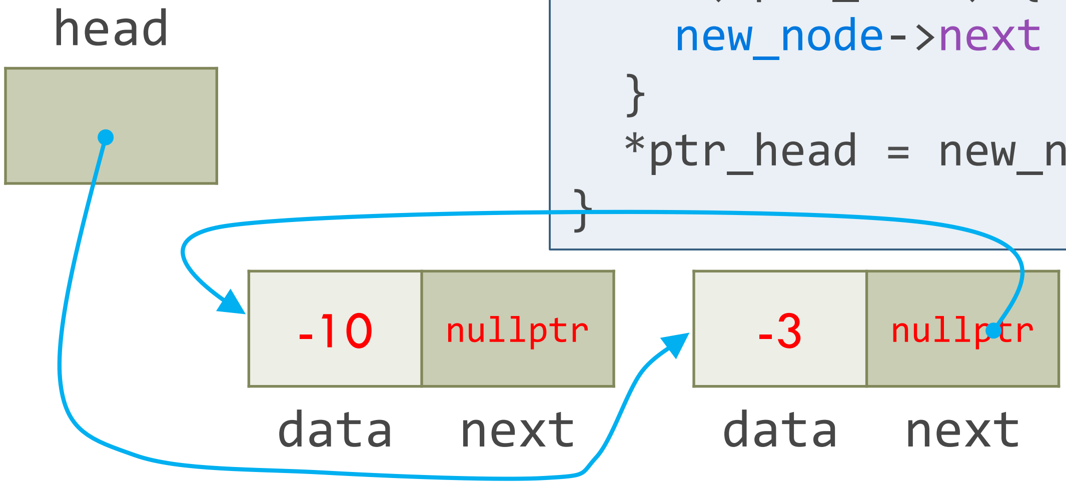


Inserting Value: push_front

- □ After 2nd call to push_front:

```
node *head{nullptr};
push_front(&head, -10);
push_front(&head, -3);
```

```
void push_front(node **ptr_head, int value) {
  node *new_node {new node{value, nullptr}};

  if (*ptr_head) {
    new_node->next = *ptr_head;
  }
  *ptr_head = new_node;
}
```



Iterating Through Linked List

- See *sll.cpp* for `push_back`, `size`, and `print` functions