

CS 32 Week 3 Discussion

Srinath

Topics

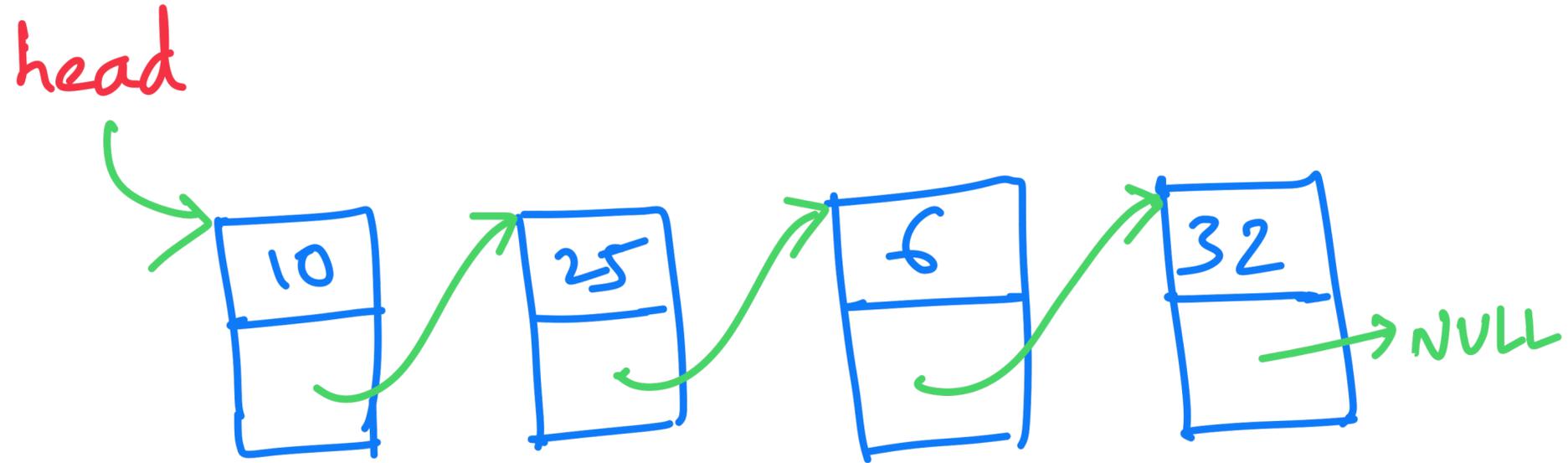
- Linked Lists
 - Definition
 - Traversal
 - Insertion
 - Deletion
 - Problems
- Worksheet 3

Linked Lists : Definition

Singly Linked List, Doubly Linked List

```
class Node {  
    int m_val;  
    Node* next;  
};
```

```
Node* head;
```



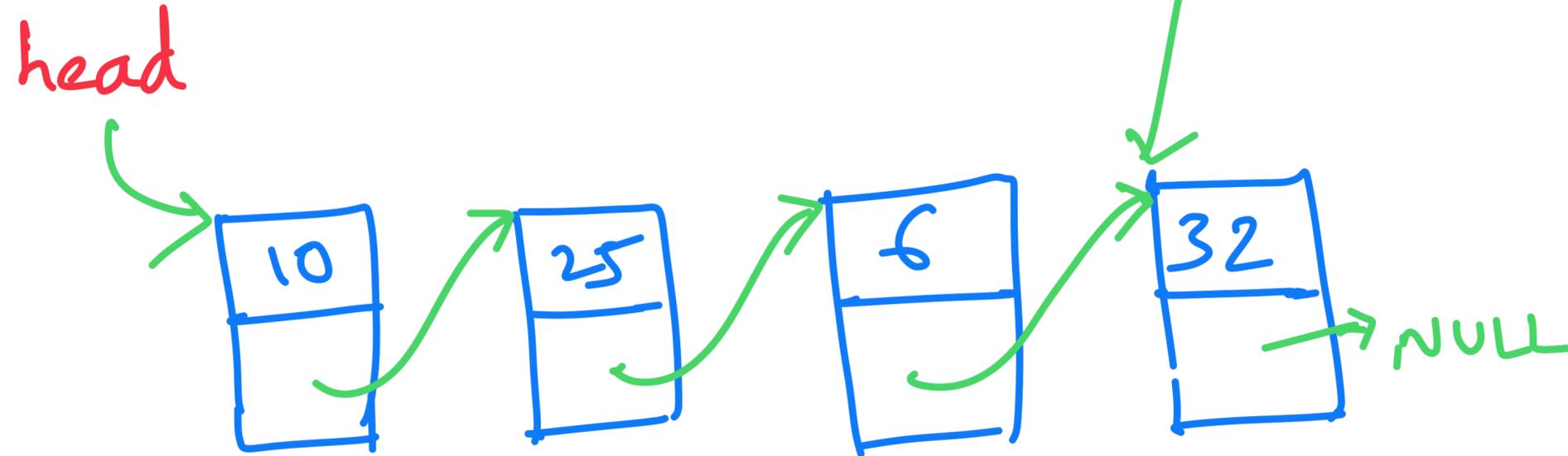
Linked Lists : Definition

Singly Linked List, Doubly Linked List

```
class Node {  
    int m_val;  
    Node* next;  
};
```

```
Node* head;
```

```
Node* tail;
```



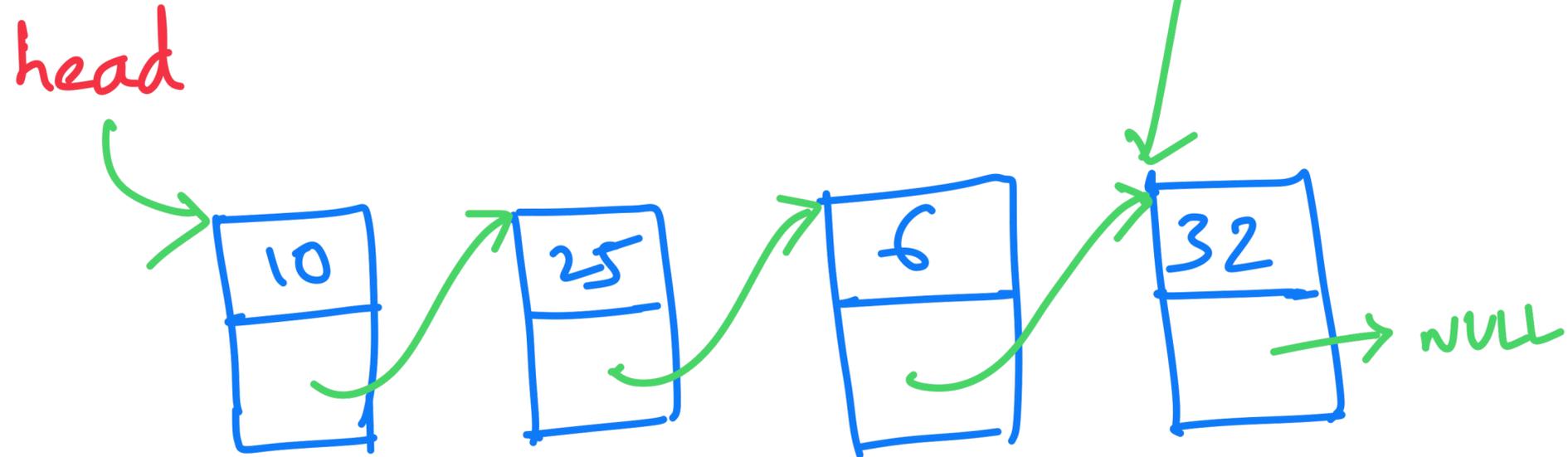
We can also use a **tail** pointer

Linked Lists : Definition

Singly Linked List, Doubly Linked List

```
class Node {  
    int m_val;  
    Node* next;  
};
```

```
Node* head;  
Node* tail;
```



We can also use a **tail** pointer

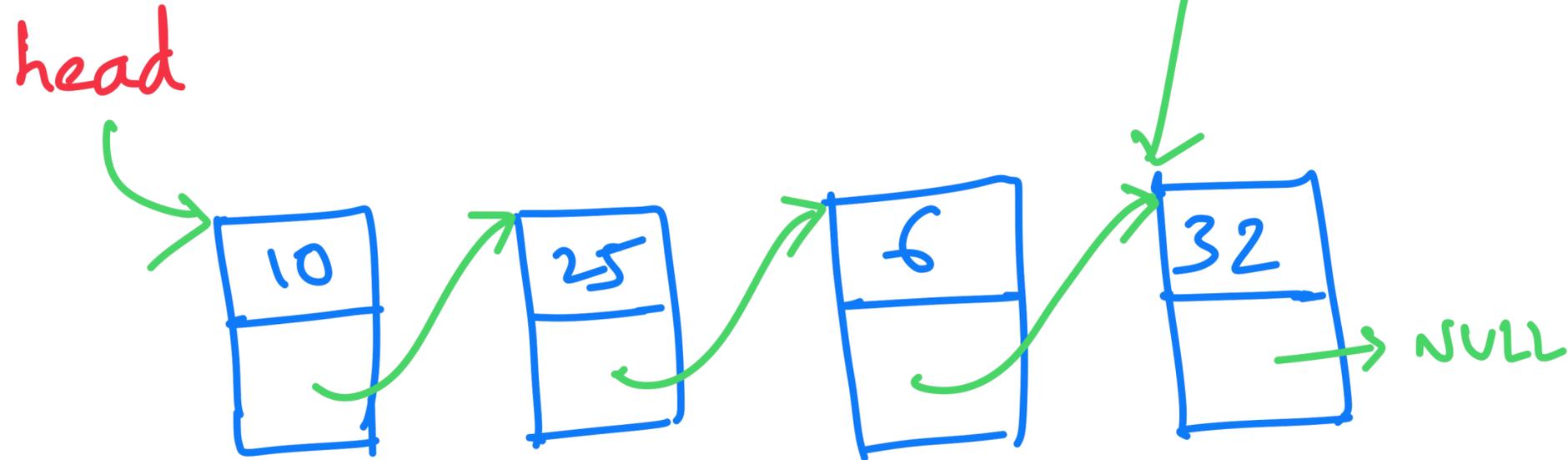
Empty list?

Linked Lists : Definition

Singly Linked List, Doubly Linked List

```
class Node {  
    int m_val;  
    Node* next;  
};
```

```
Node* head;  
Node* tail;
```



We can also use a **tail** pointer

Empty list?

```
head = NULL;
```

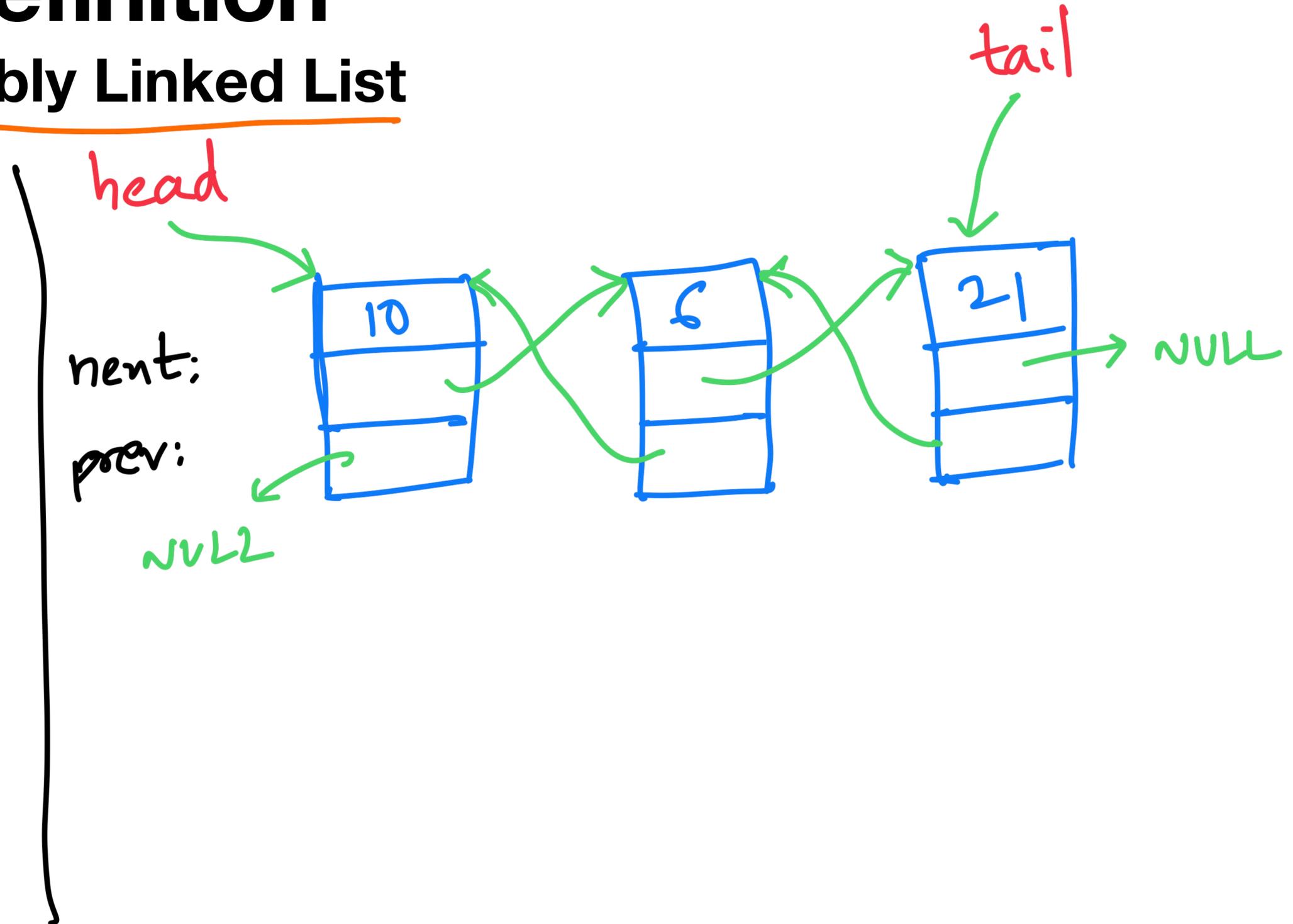
```
tail = NULL;
```

Linked Lists : Definition

Singly Linked List, Doubly Linked List

```
class Node {  
    int m_val;  
    Node* next;  
    Node* prev;  
};
```

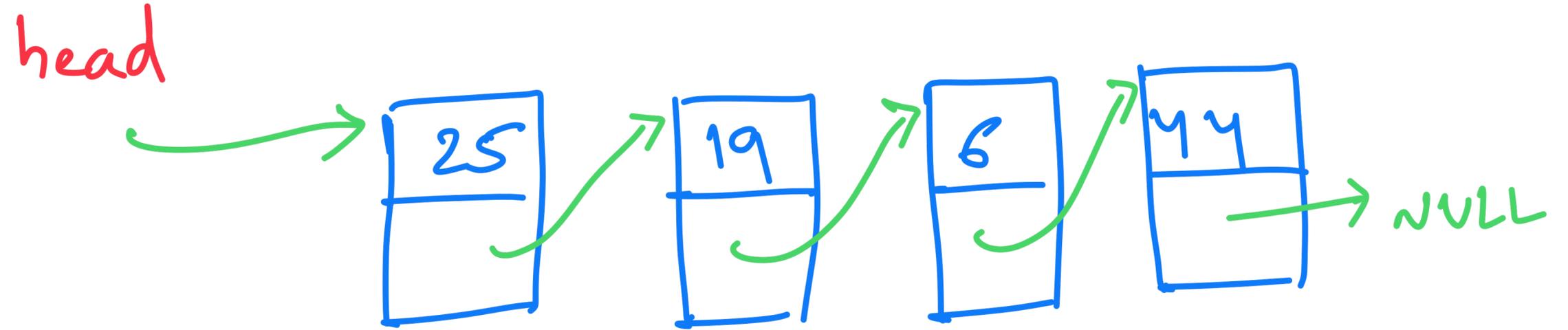
```
Node* head;  
Node* tail;
```



Linked Lists : Traversal

Printing all elements

Algorithm

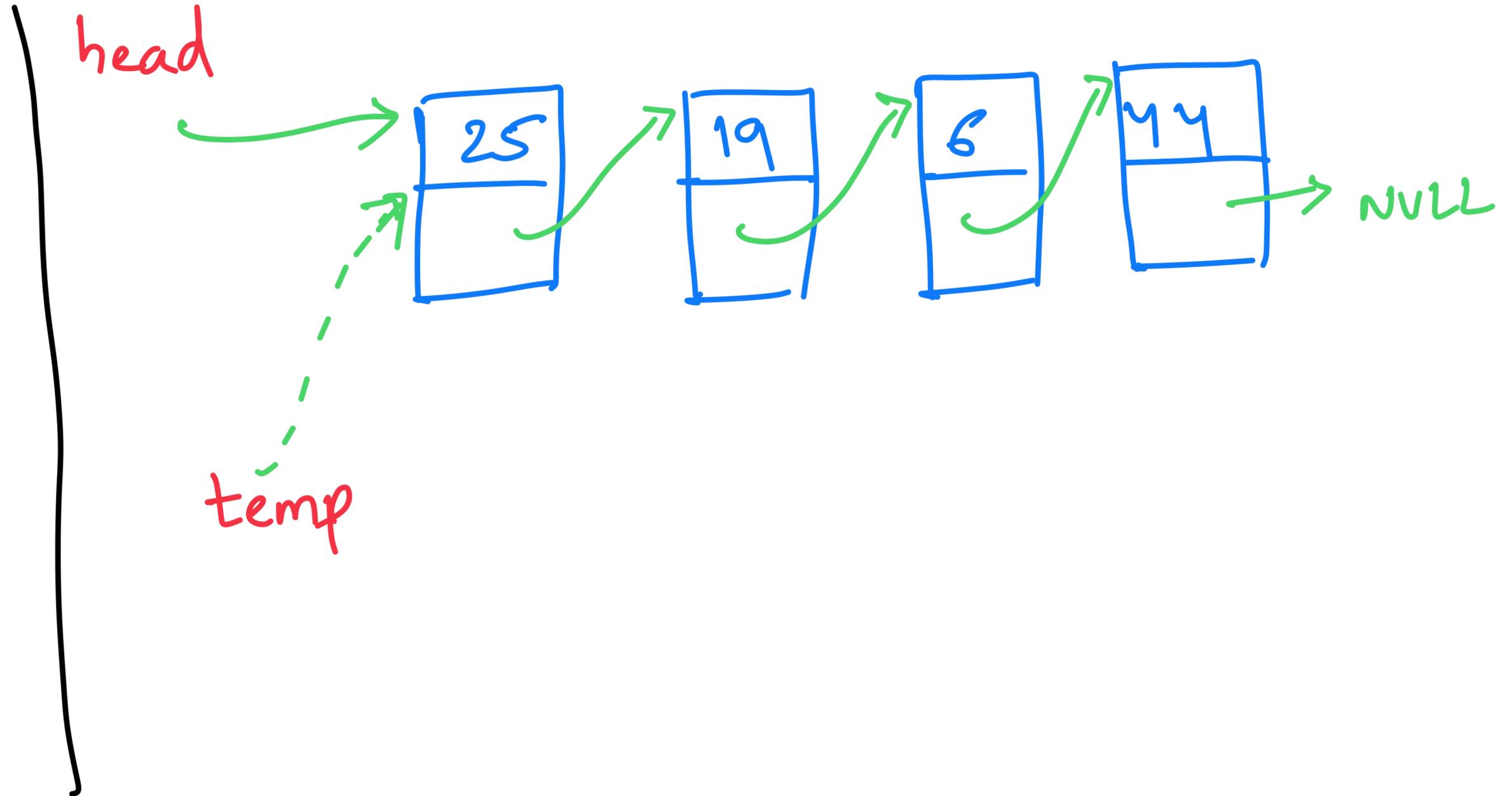


Linked Lists : Traversal

Printing all elements

Algorithm

```
Node* temp;  
temp = head;
```

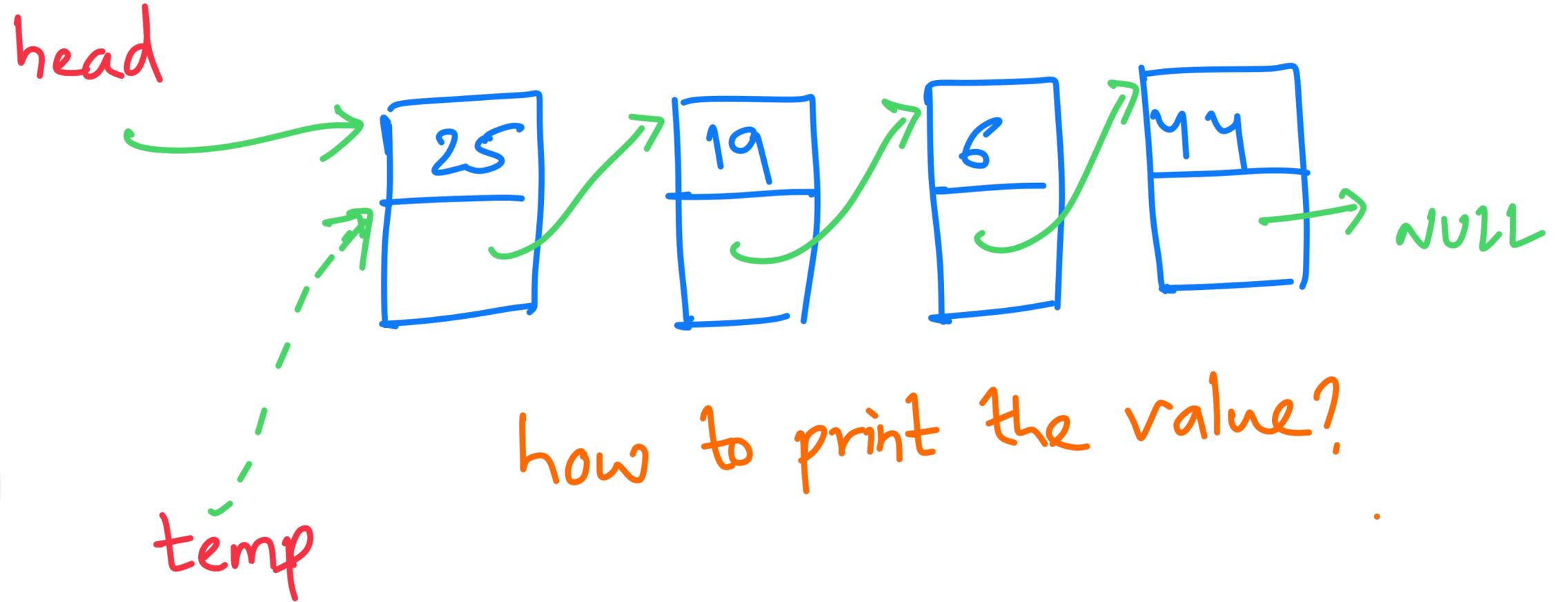


Linked Lists : Traversal

Algorithm

Printing all elements

Node* temp;
temp = head;

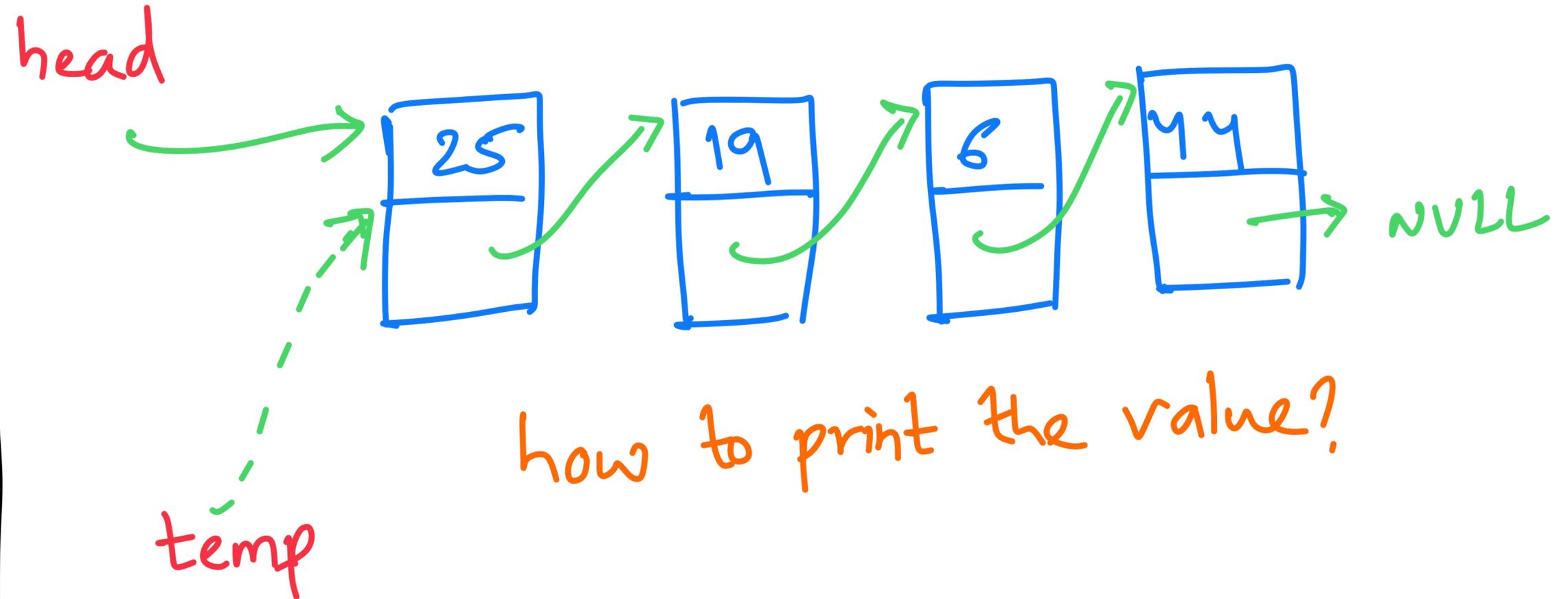


Linked Lists : Traversal

Algorithm

Printing all elements

```
Node* temp;  
temp = head;  
cout << temp->val;
```

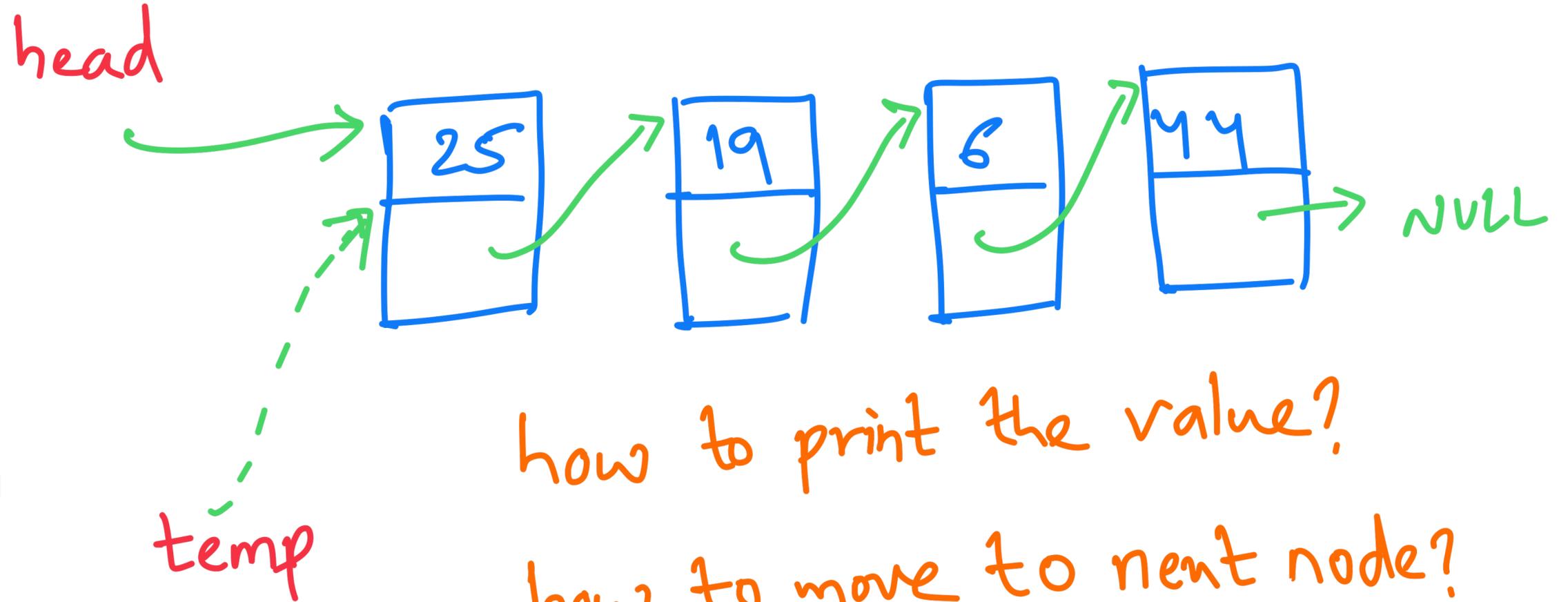


Linked Lists : Traversal

Algorithm

Printing all elements

```
Node* temp;  
temp = head;  
cout << temp->val;
```



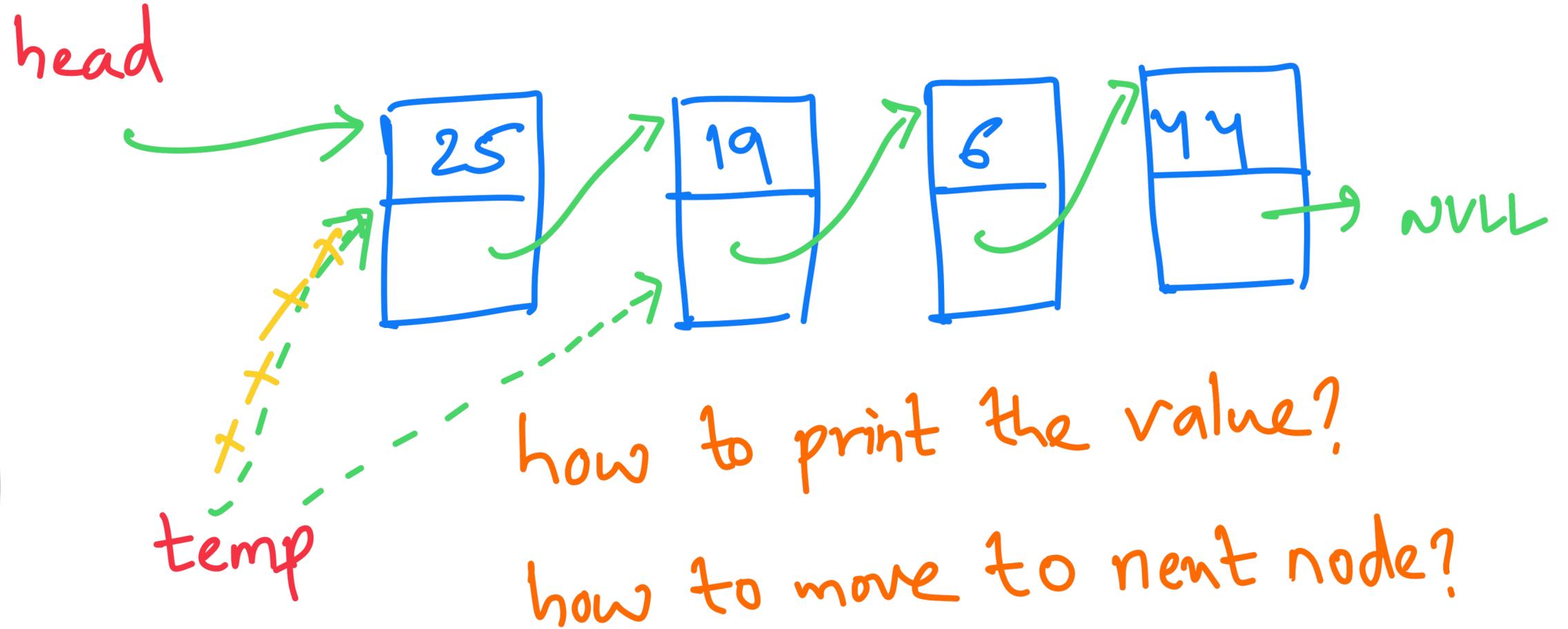
how to print the value?
how to move to next node?

Linked Lists : Traversal

Algorithm

Printing all elements

```
Node* temp;  
temp = head;  
cout << temp->val;  
temp = temp->next;
```

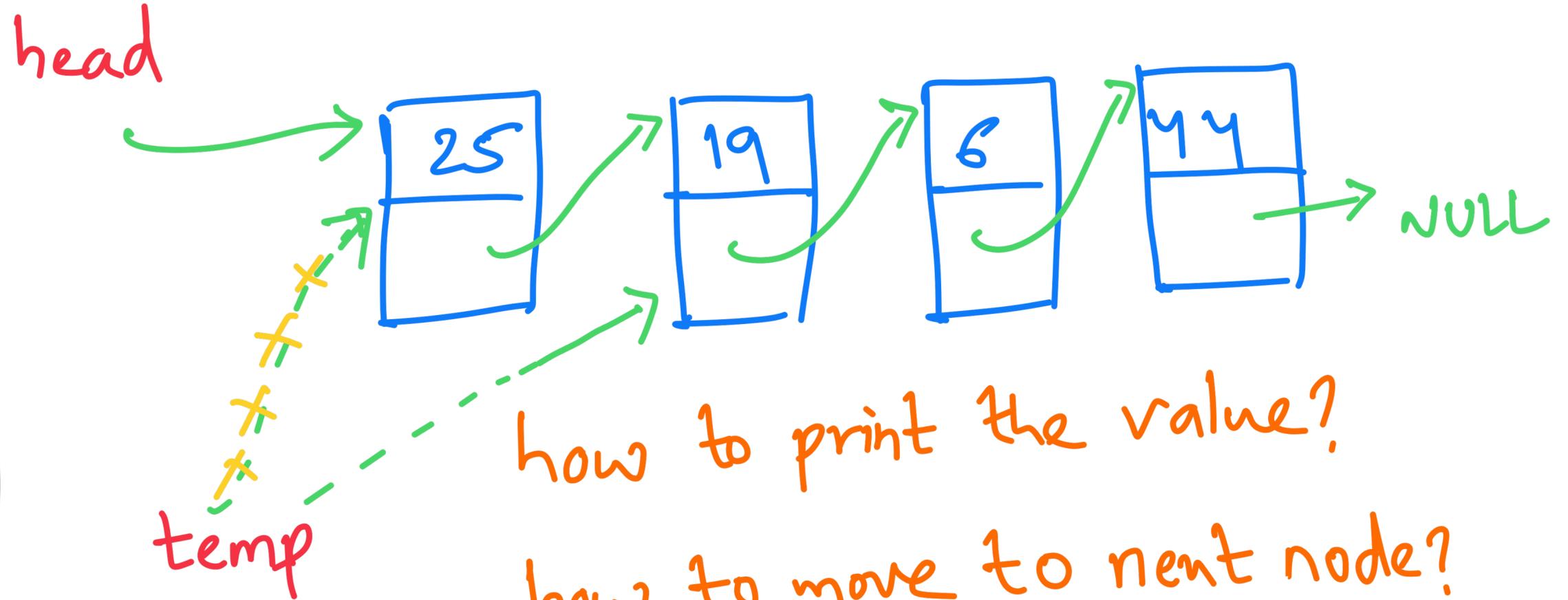


Linked Lists : Traversal

Algorithm

Printing all elements

```
Node* temp;  
temp = head;  
cout << temp->val;  
temp = temp->next;
```



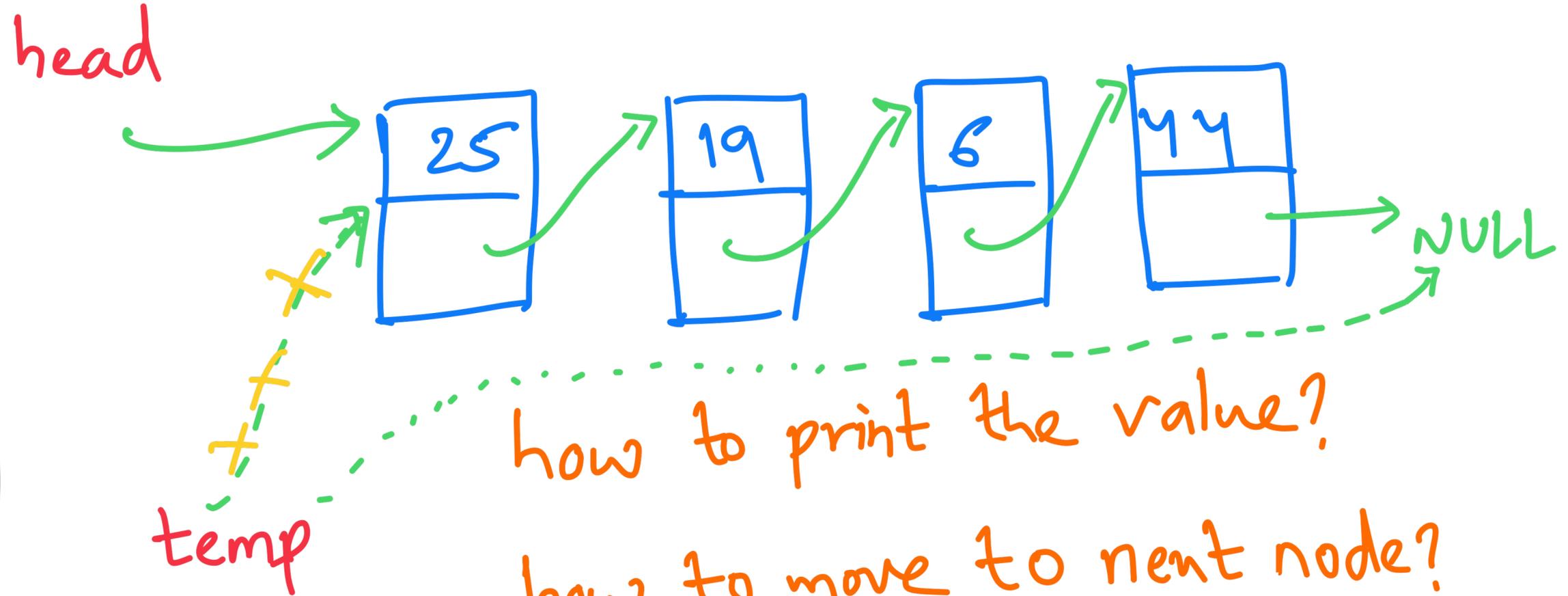
how to print the value?
how to move to next node?
Print again until?

Linked Lists : Traversal

Algorithm

Printing all elements

```
Node* temp;  
temp = head;  
cout << temp->val;  
temp = temp->next;
```



how to print the value?
how to move to next node?
Print again until?
temp becomes NULL.

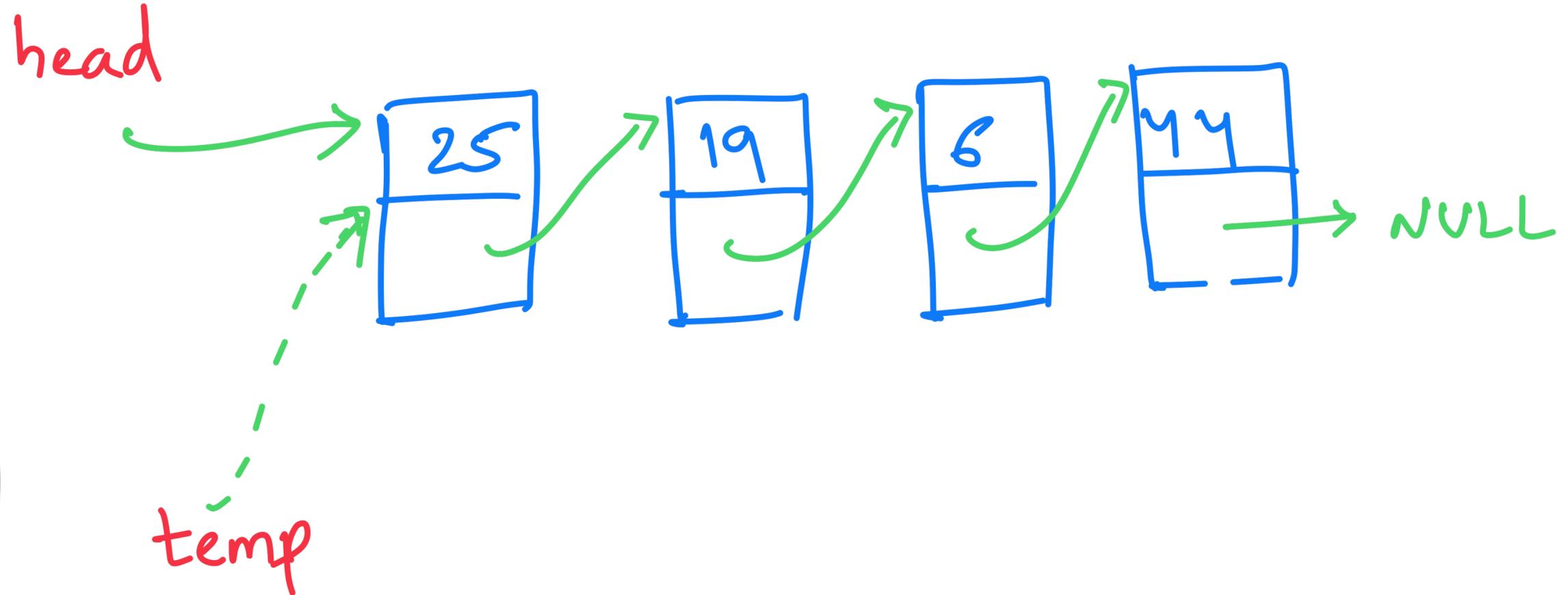
Linked Lists : Traversal

Printing all elements

Algorithm

```
Node* temp;  
temp = head;
```

```
while (temp != NULL)  
{  
    cout << temp->n-val;  
    temp = temp->nent;  
}
```

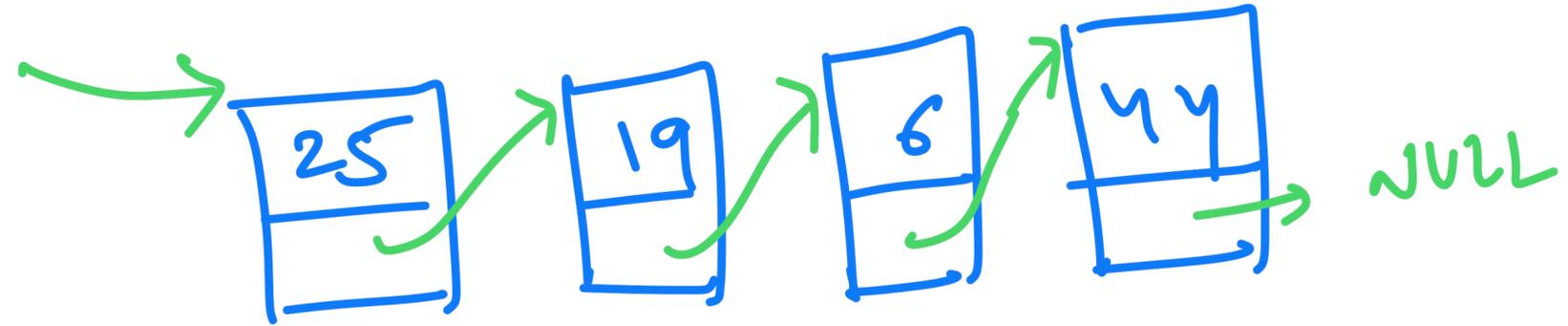


Linked Lists : Traversal

Find a given element

Find node representing 6?

head



Linked Lists : Traversal

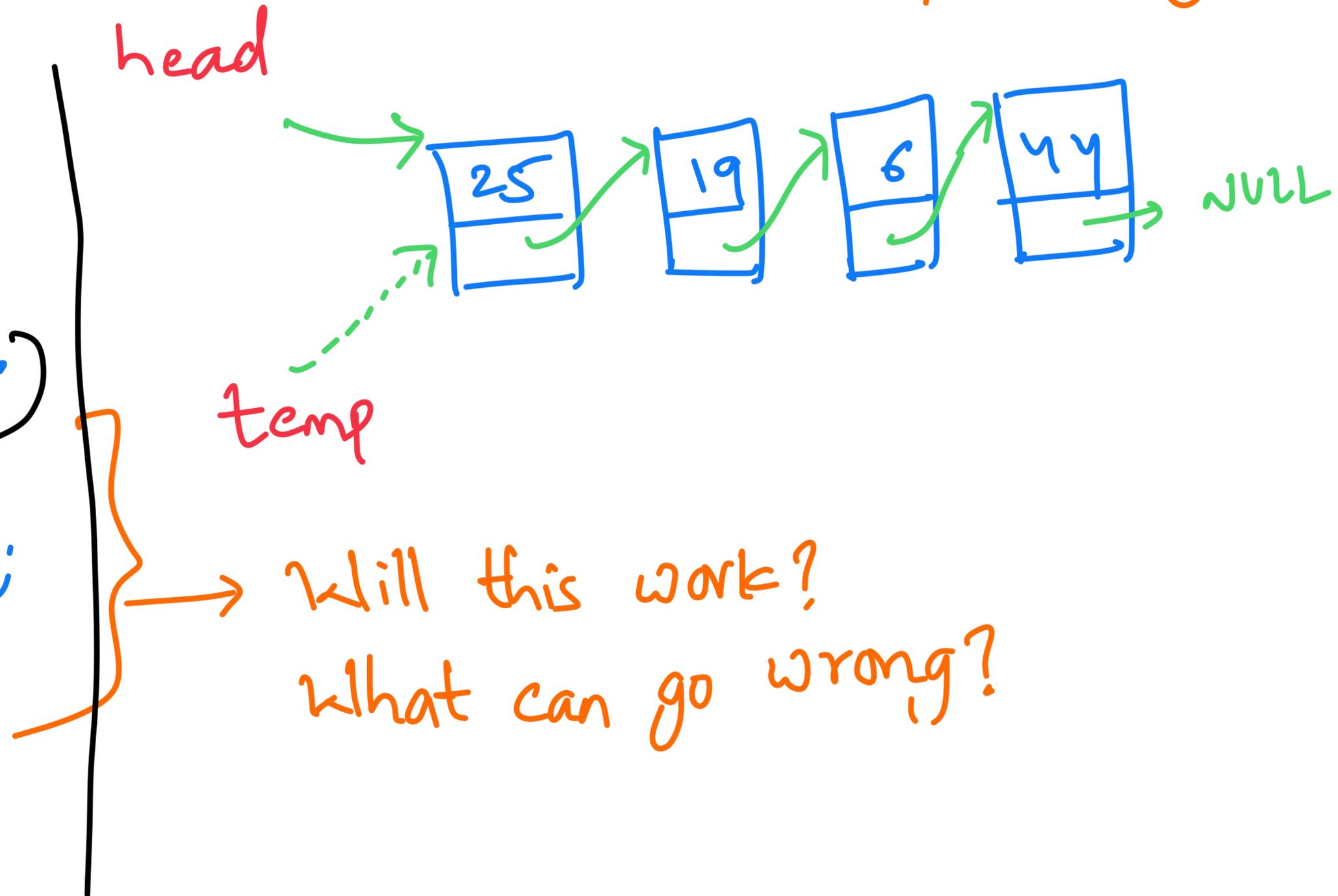
Find a given element

Find node representing 6?

```
Node* temp;  
temp = head;
```

```
while (temp->n-val != 6)  
{  
    temp = temp->next;  
}
```

```
return temp;
```



Linked Lists : Traversal

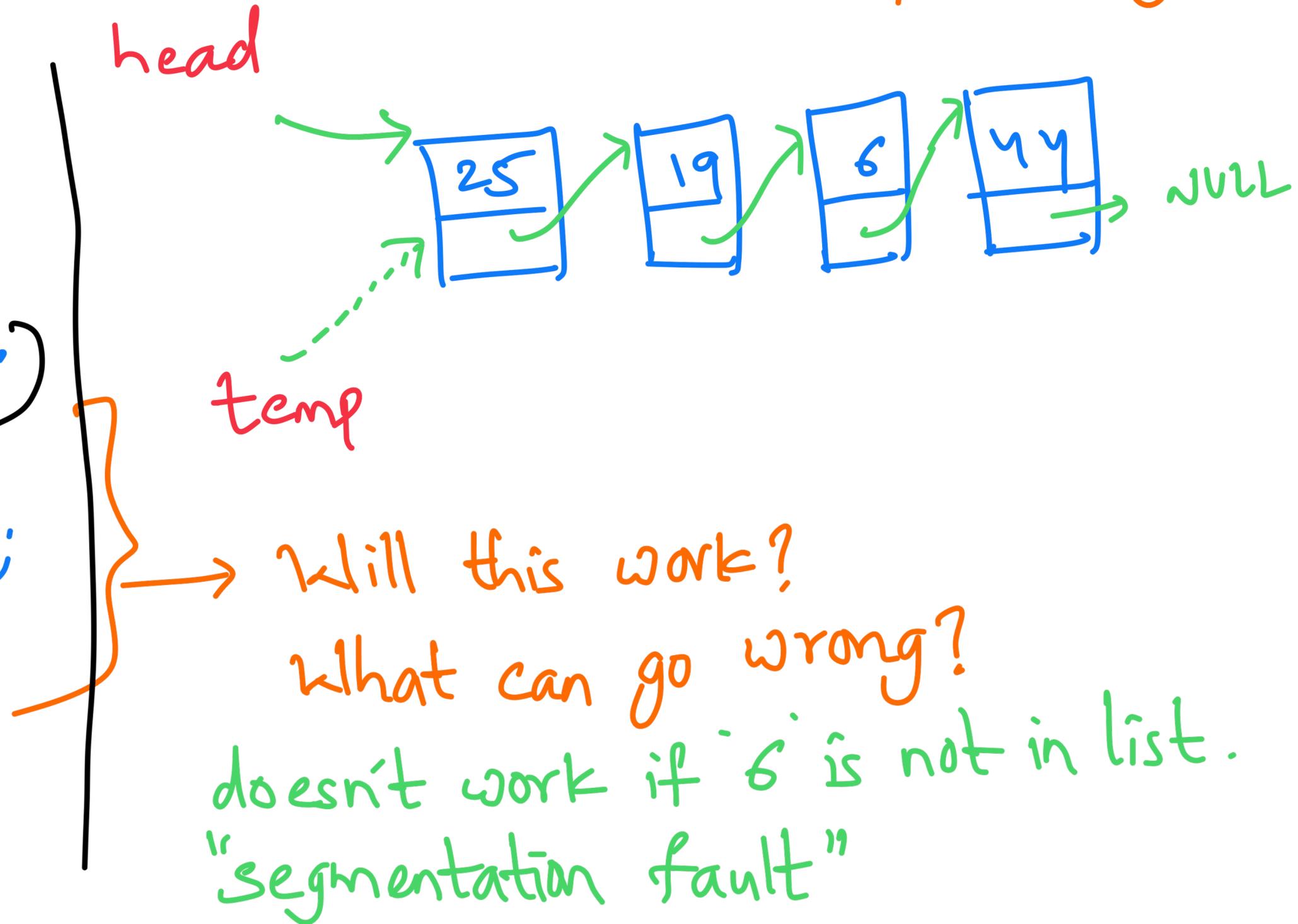
Find a given element

Find node representing 6?

```
Node* temp;  
temp = head;
```

```
while (temp->n-val != 6)  
{  
    temp = temp->next;  
}
```

```
return temp;
```

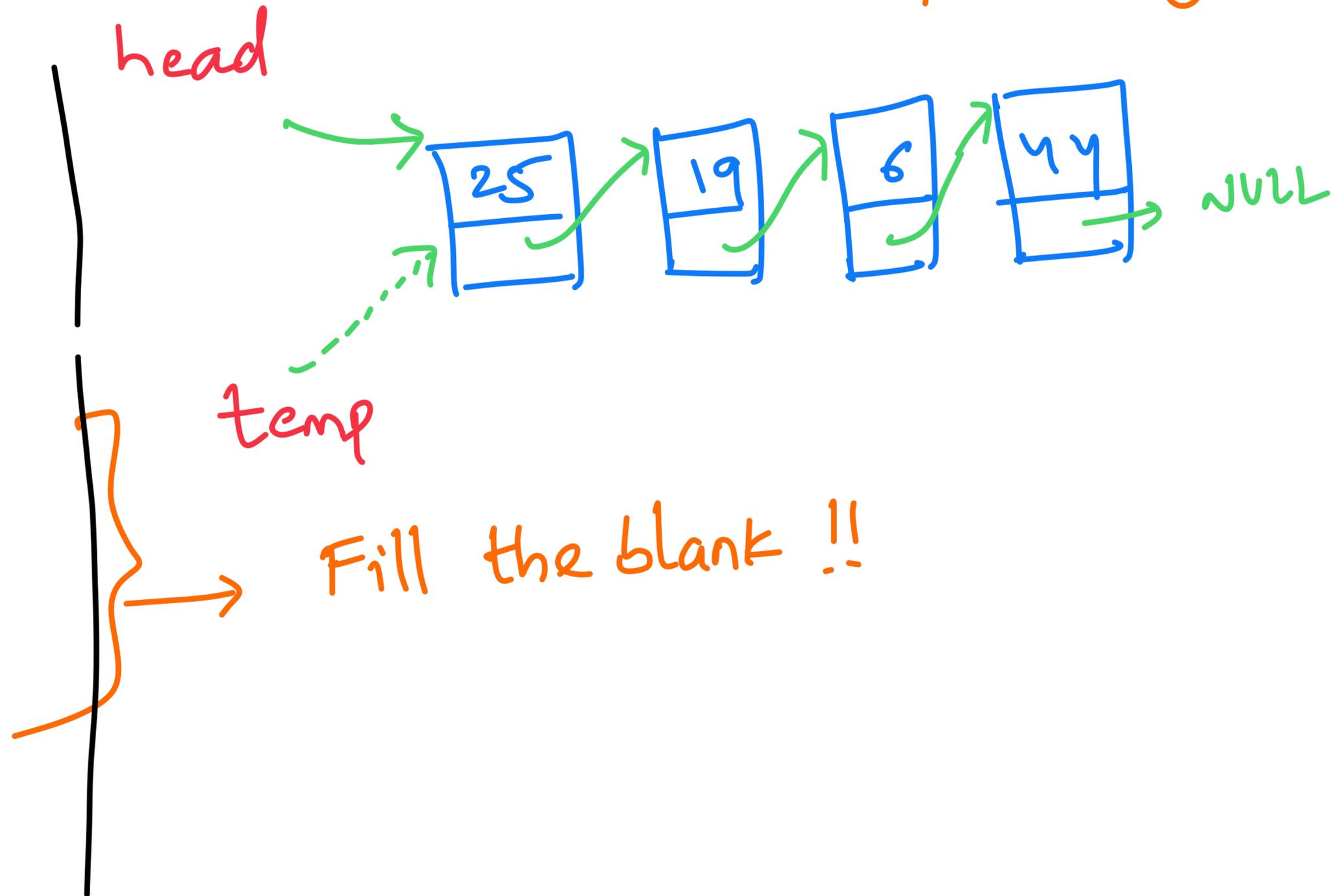


Linked Lists : Traversal

Find a given element

```
Node* temp;  
temp = head;  
while(temp != NULL){  
    -----  
    temp = temp->next;  
}  
return temp;
```

Find node representing 6?

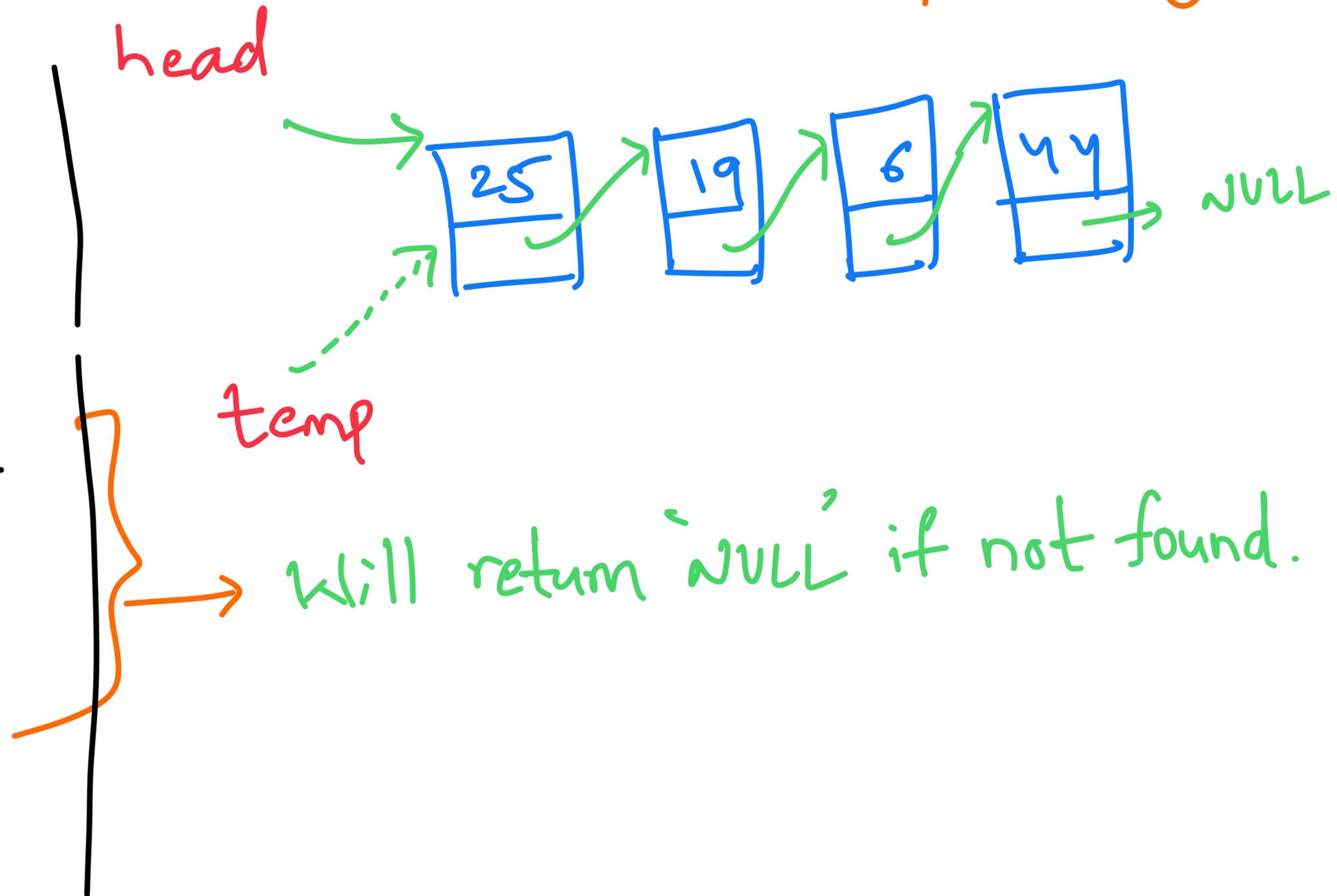


Linked Lists : Traversal

Find a given element

```
Node* temp;  
temp = head;  
while(temp != NULL){  
    if(temp->n-val == 6){  
        break;  
    }  
    temp = temp->next;  
}  
return temp;
```

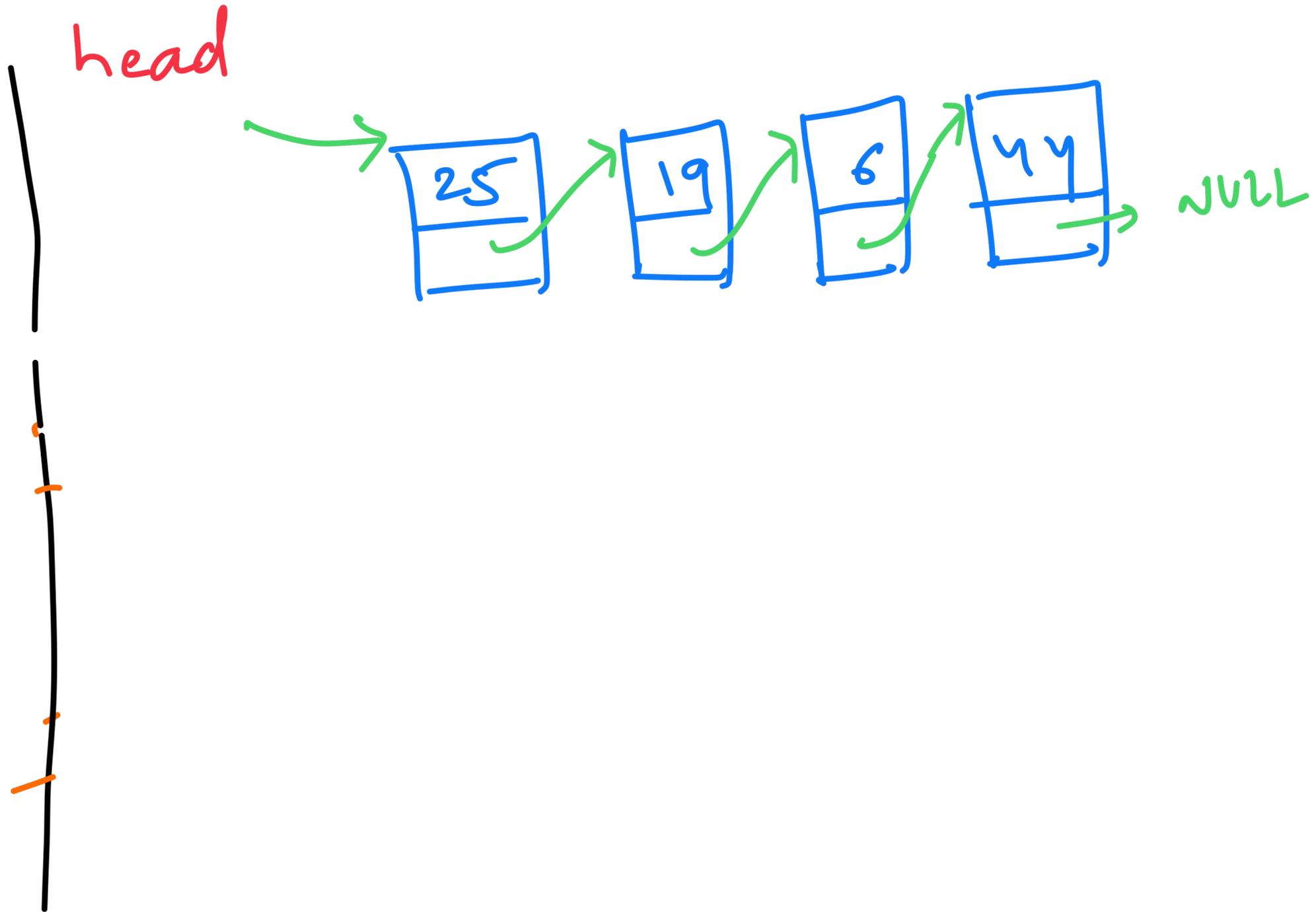
Find node representing 6?



Linked Lists : Traversal

Find element at position i

Find node at position 2 ?



Linked Lists : Traversal

Find element at position i

Find node at position 2?

```
Node* temp = head;
```

① -----

```
while(temp != NULL){
```

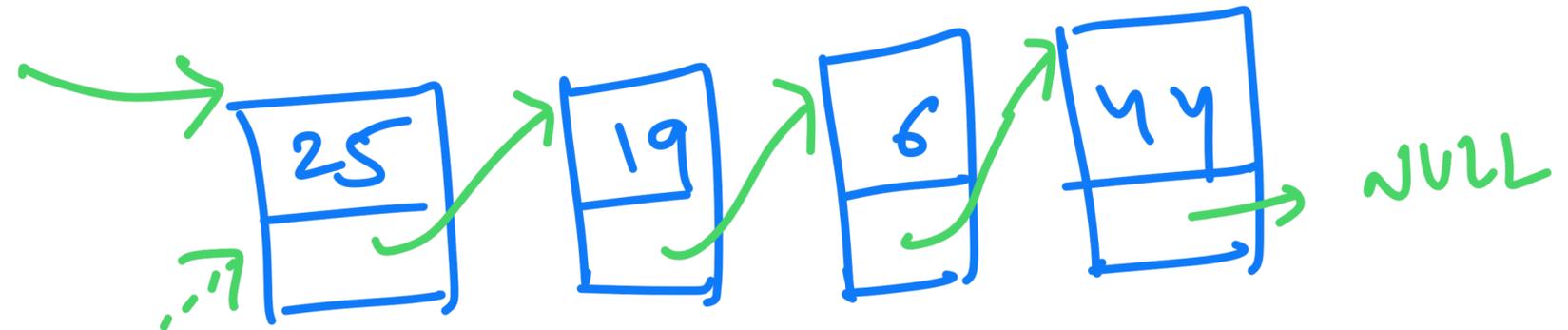
② -----

```
temp = temp->next;
```

```
}  
return temp;
```

head

temp



Fill the 3 blanks!!!

Linked Lists : Traversal

Find element at position i

Find node at position 2 ?

```
Node* temp = head;
```

```
int pos = 0;
```

```
while(temp != NULL){
```

② -----

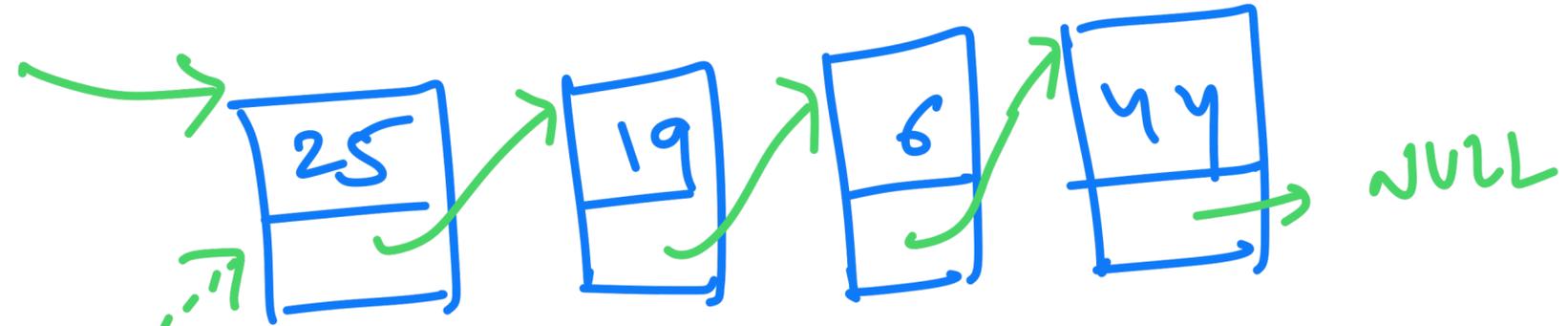
③ -----

```
temp = temp->next;
```

```
return temp;
```

head

temp



Fill the 3 blanks !!!

Linked Lists : Traversal

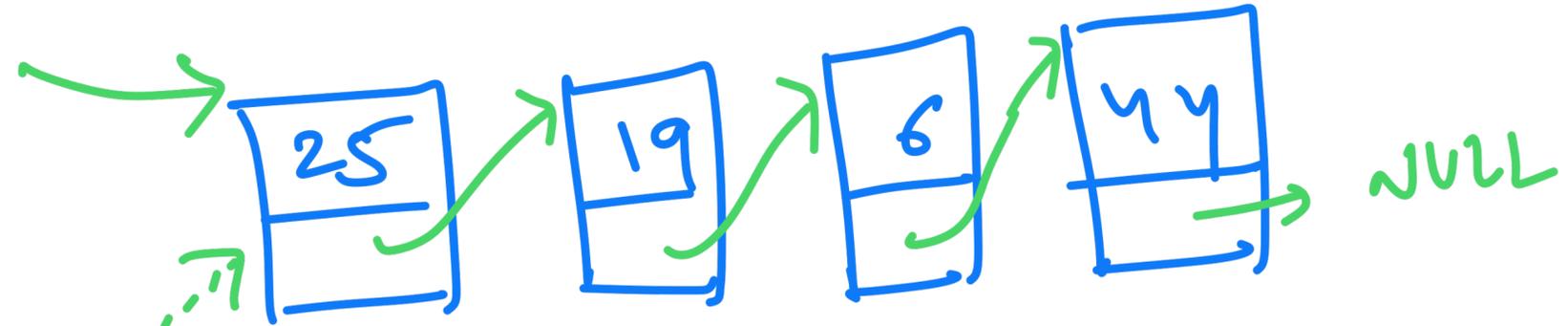
Find node at position 2 ?

Find element at position i

```
Node* temp = head;
int pos = 0;
while(temp != NULL){
    if(pos == 2) { break; }
    temp = temp->next;
}
return temp;
```

head

temp



③ - - - - -
temp = temp->next;
return temp;

Fill the 3 blanks!!!

Linked Lists : Traversal

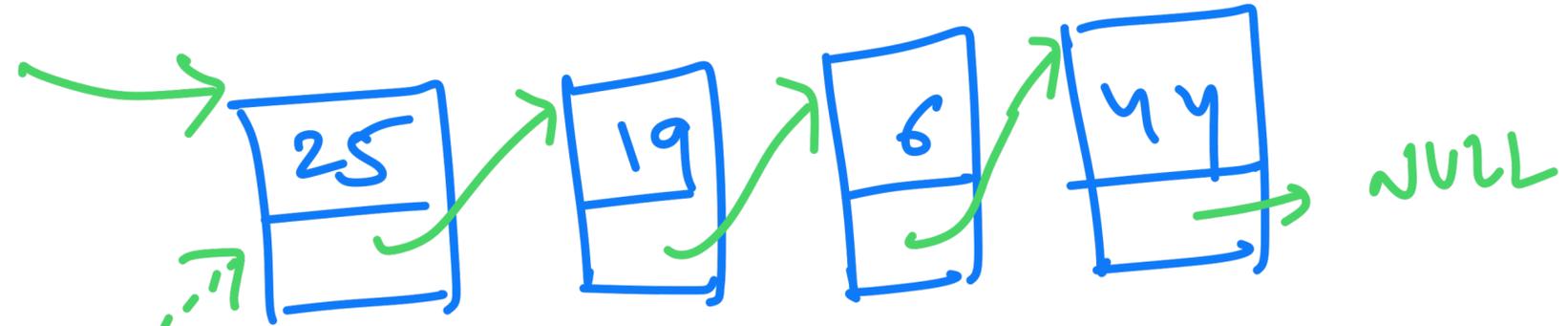
Find node at position 2 ?

Find element at position i

```
Node* temp = head;
int pos = 0;
while (temp != NULL) {
    if (pos == 2) { break; }
    pos++;
    temp = temp->next;
}
return temp;
```

head

temp



Fill the 3 blanks !!!

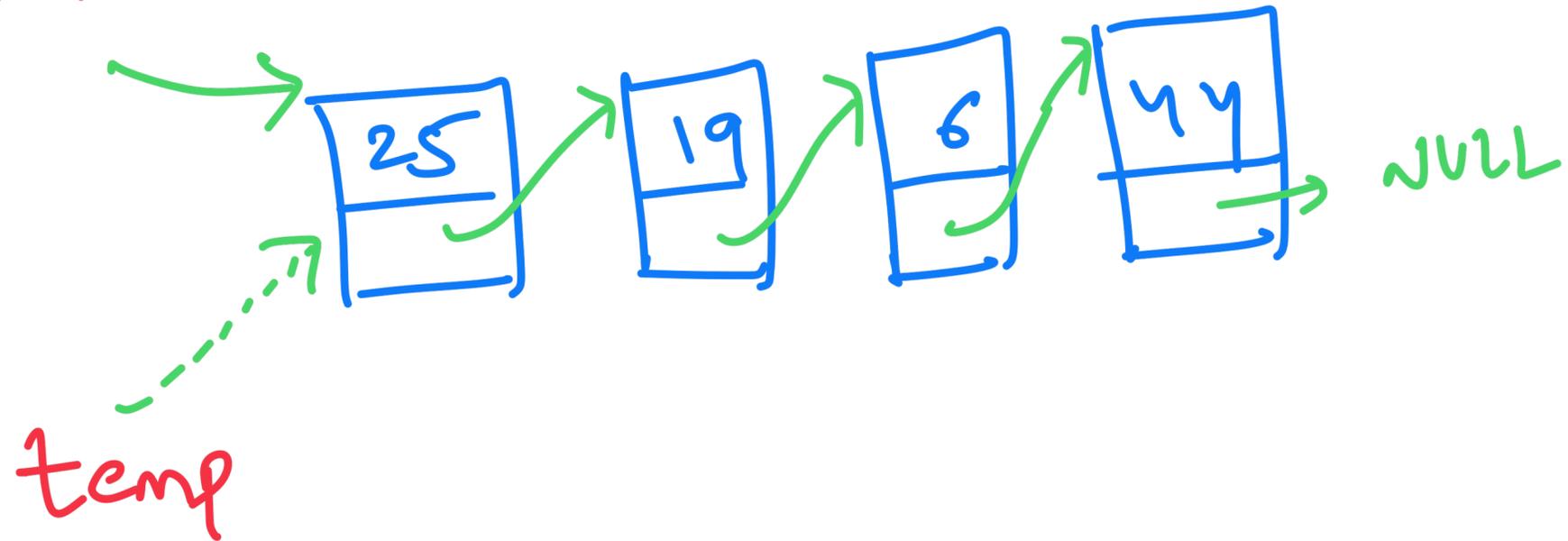
Linked Lists : Traversal

Find element at position i

```
Node* temp = head;
int pos = 0;
while(temp != NULL){
    if(pos == 2) { break; }
    pos++;
    temp = temp->next;
}
return temp;
```

Find node at position 2?

head

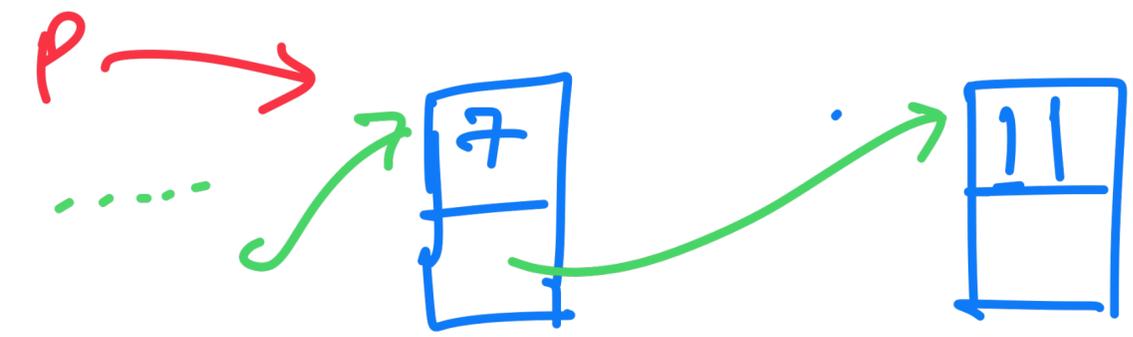


Will return 'NULL' if position is out of list bounds.

Linked Lists : Insertion

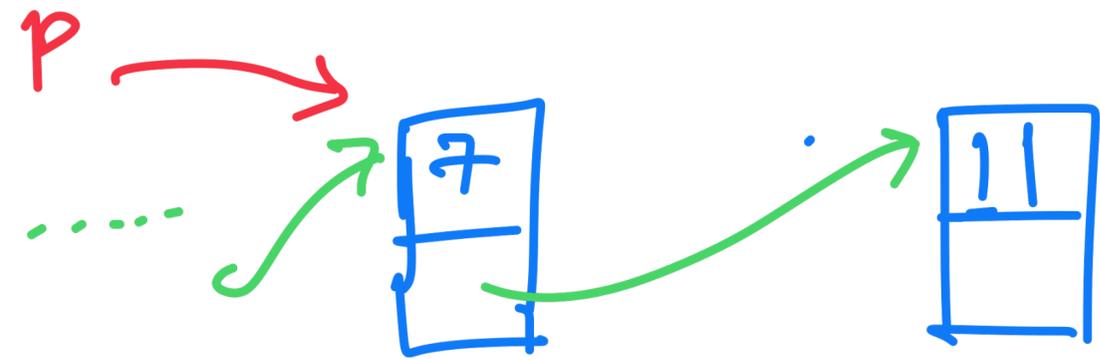
Insert element in between

insert 26?

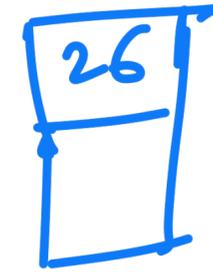


Linked Lists : Insertion

Insert element in between



insert 26?

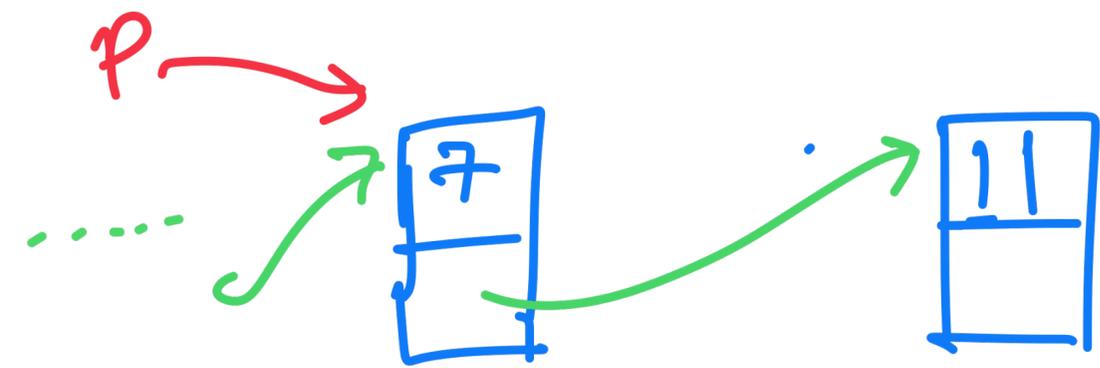


```
Node* n = new Node();
```

① create new node

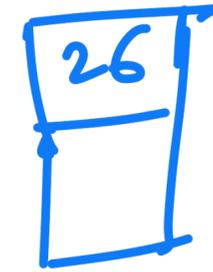
Linked Lists : Insertion

Insert element in between



insert 26?

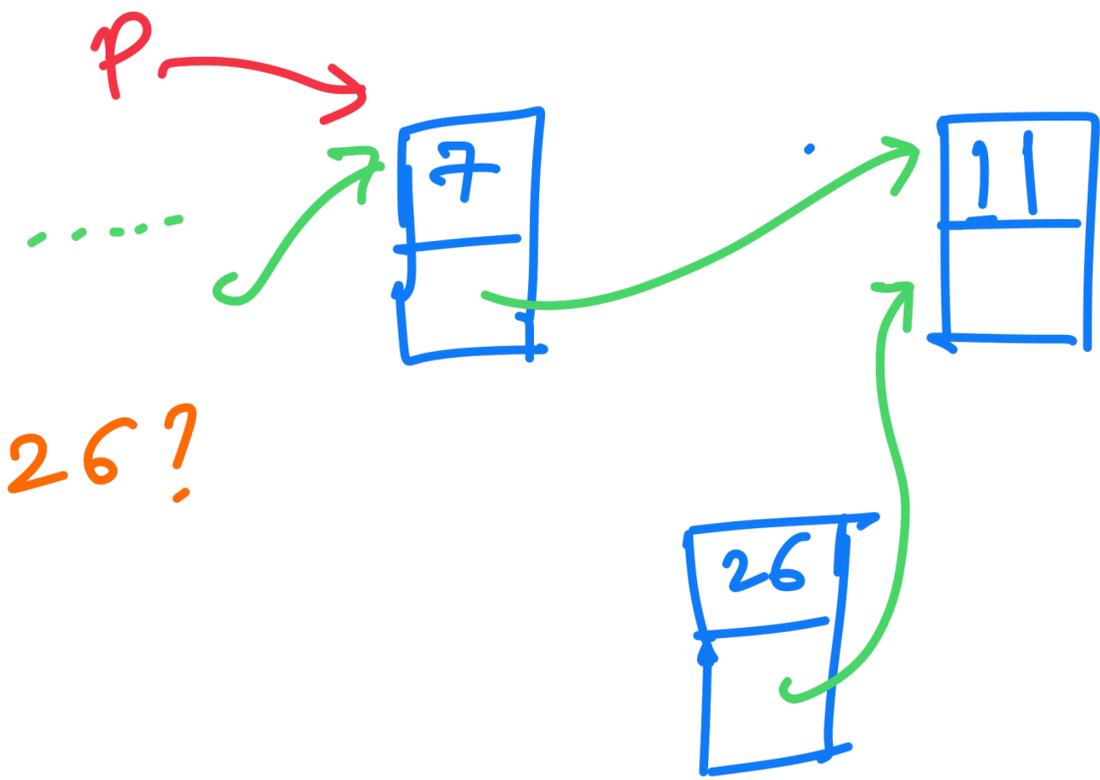
`Node* n = new Node();`



- ① create new node
- ② rewire it!!

Linked Lists : Insertion

Insert element in between



insert 26?

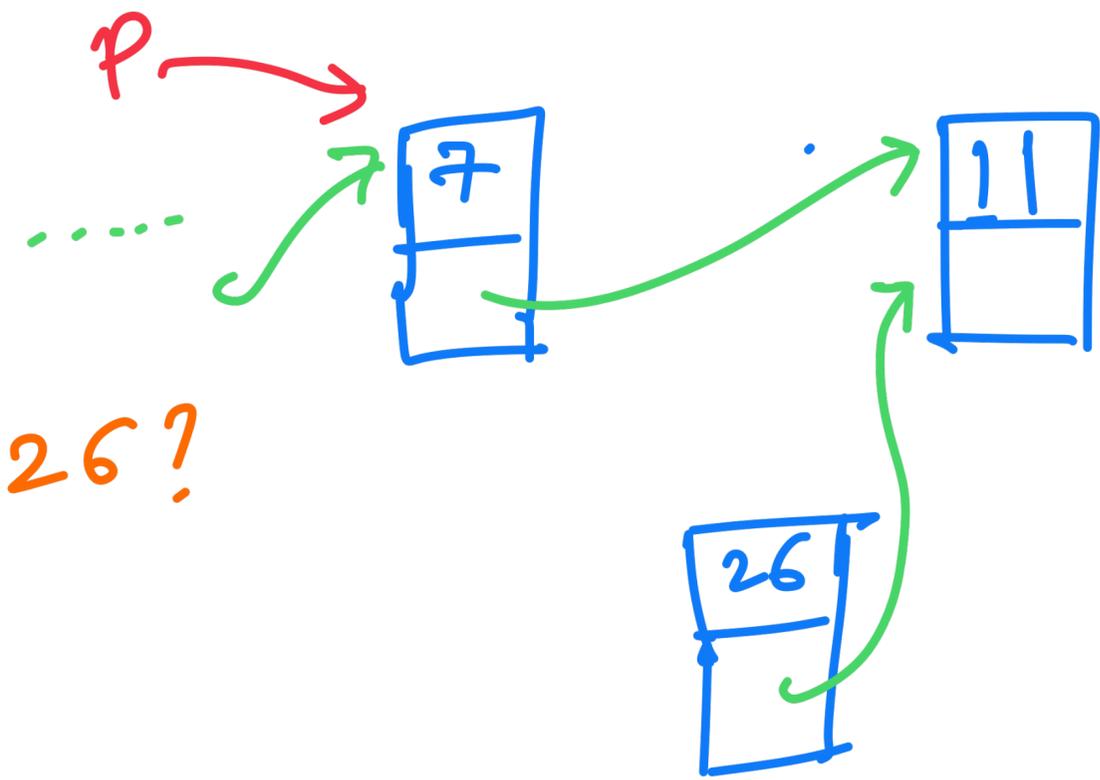
```
Node* n = new Node();
```

① - - - - -

- ① create new node
- ② rewire it!!

Linked Lists : Insertion

Insert element in between



insert 26?

```
Node* n = new Node();
```

```
① n->next = p->next;
```

① create new node

② rewire it!!

Linked Lists : Insertion

Insert element in between

`Node* n = new Node();`

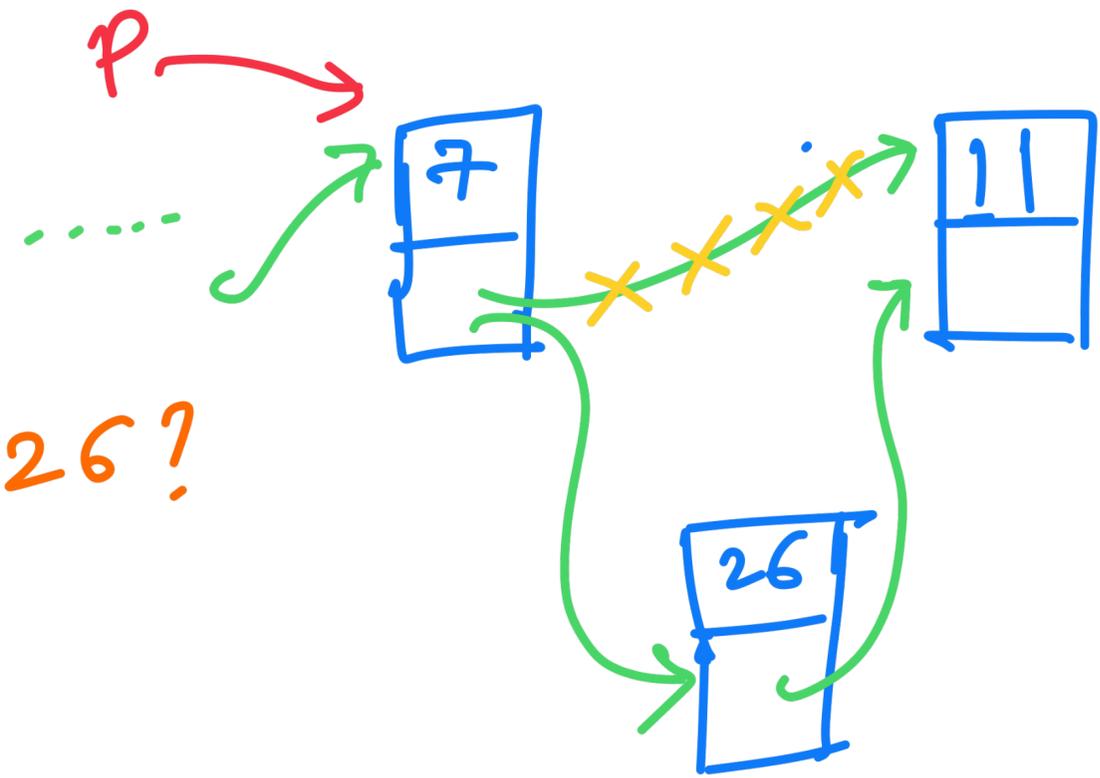
① `n->next = p->next;`

② `---`

insert 26?

① create new node

② rewire it!!



Linked Lists : Insertion

Insert element in between

$\text{Node}^* n = \text{new Node}();$

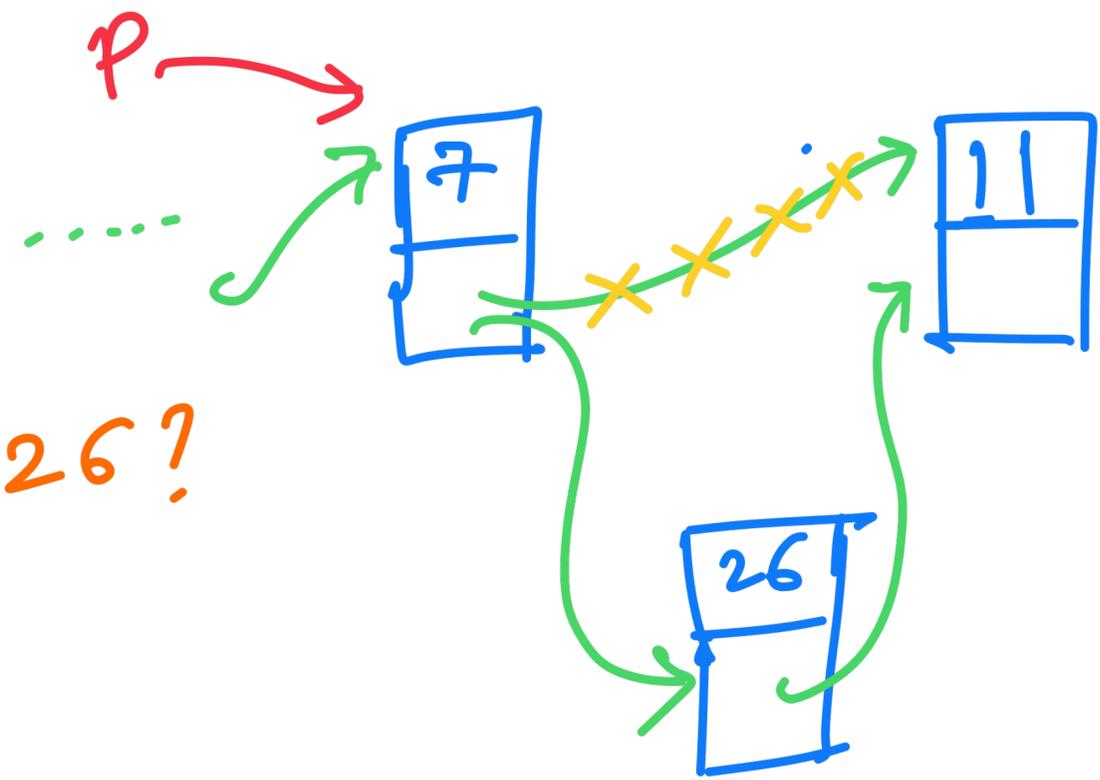
① $n \rightarrow \text{next} = p \rightarrow \text{next};$

② $p \rightarrow \text{next} = n;$

insert 26?

① create new node

② rewire it!!

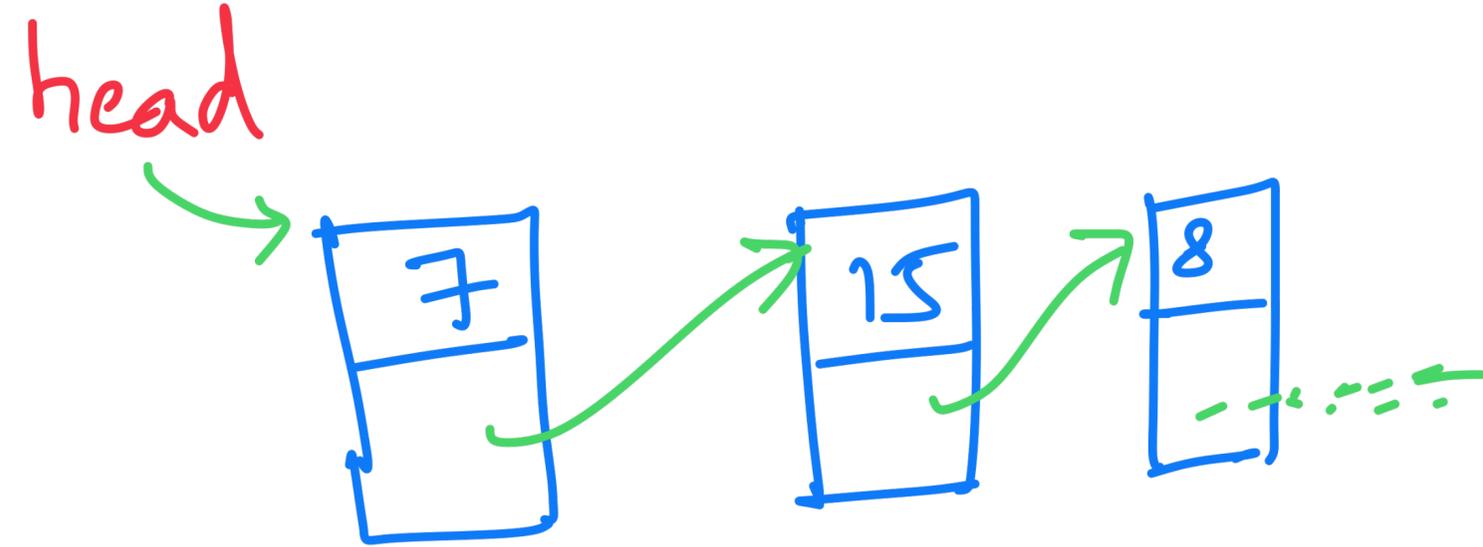


Linked Lists : Insertion

Insert element at beginning



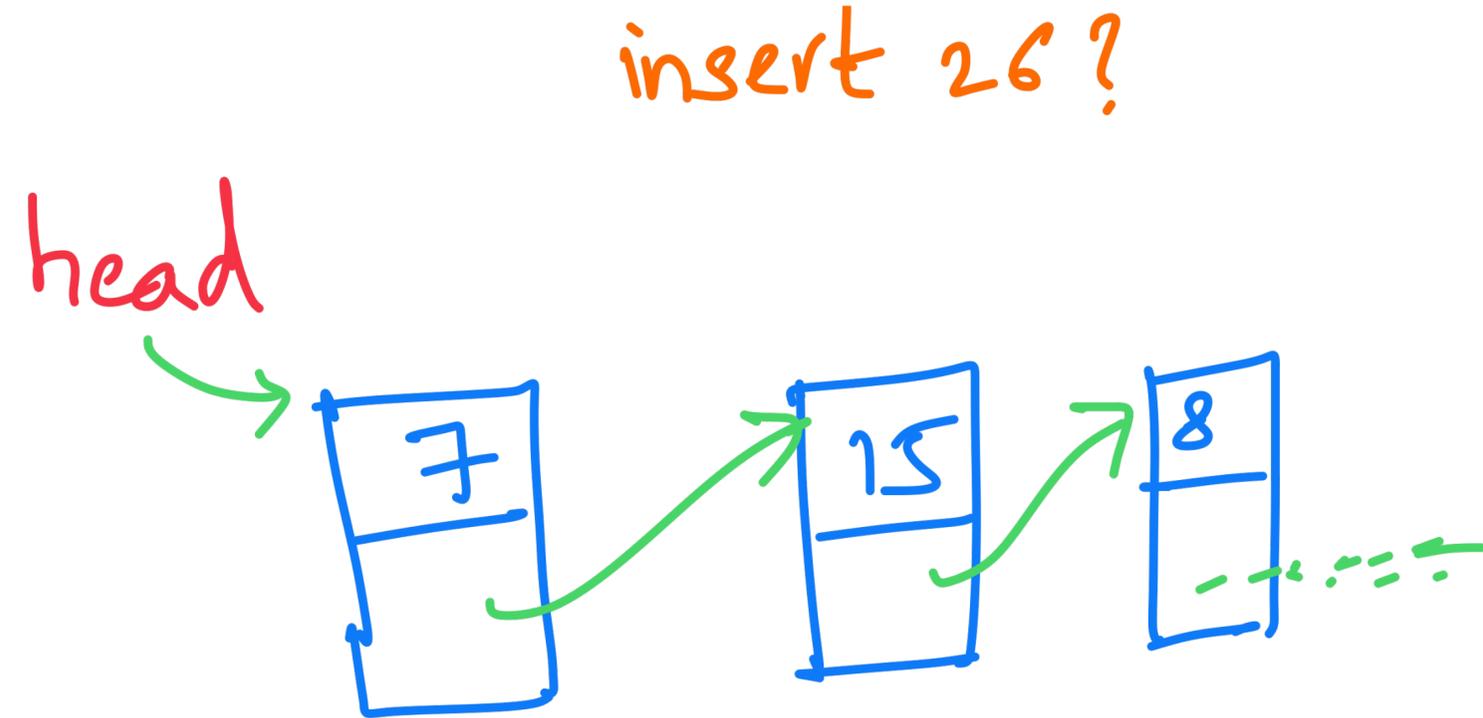
insert 26?



Linked Lists : Insertion

Insert element at beginning

`Node* n = new Node();`



① create new node

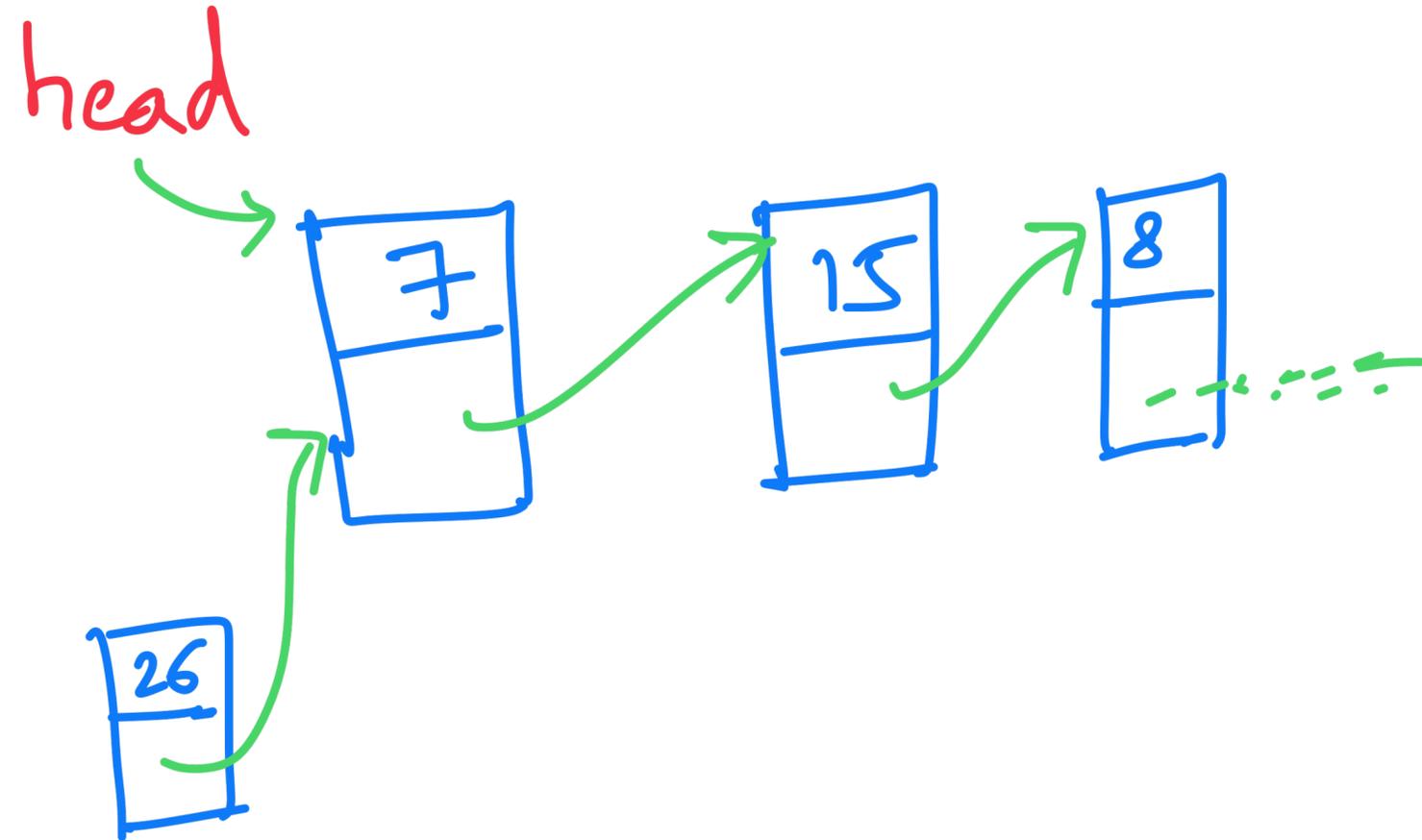
Linked Lists : Insertion

Insert element at beginning

`Node* n = new Node();`

① -----

insert 26?



① create new node

② rewire!!

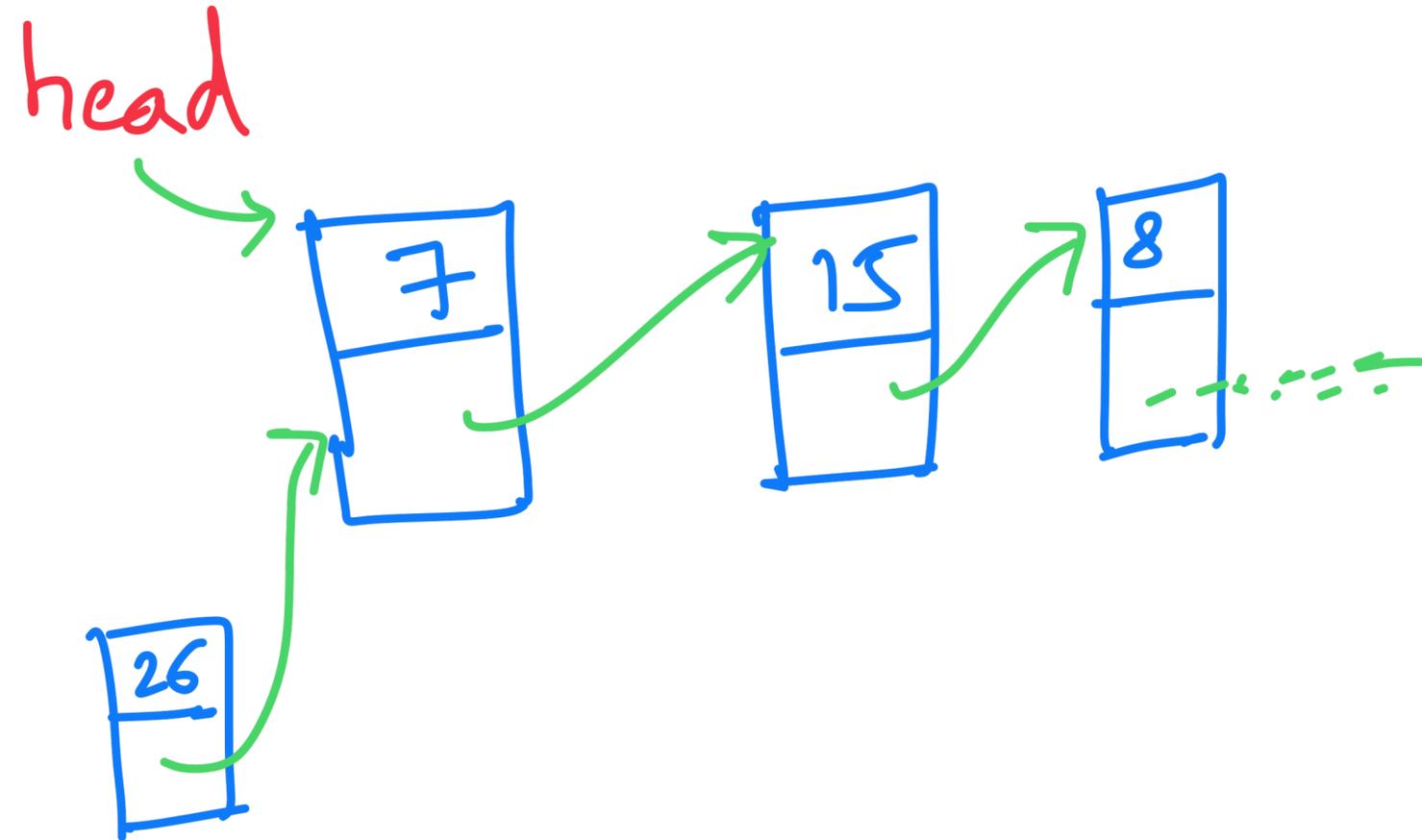
Linked Lists : Insertion

Insert element at beginning

`Node* n = new Node();`

① `n->next = head;`

insert 26?



① create new node

② rewire!!

Linked Lists : Insertion

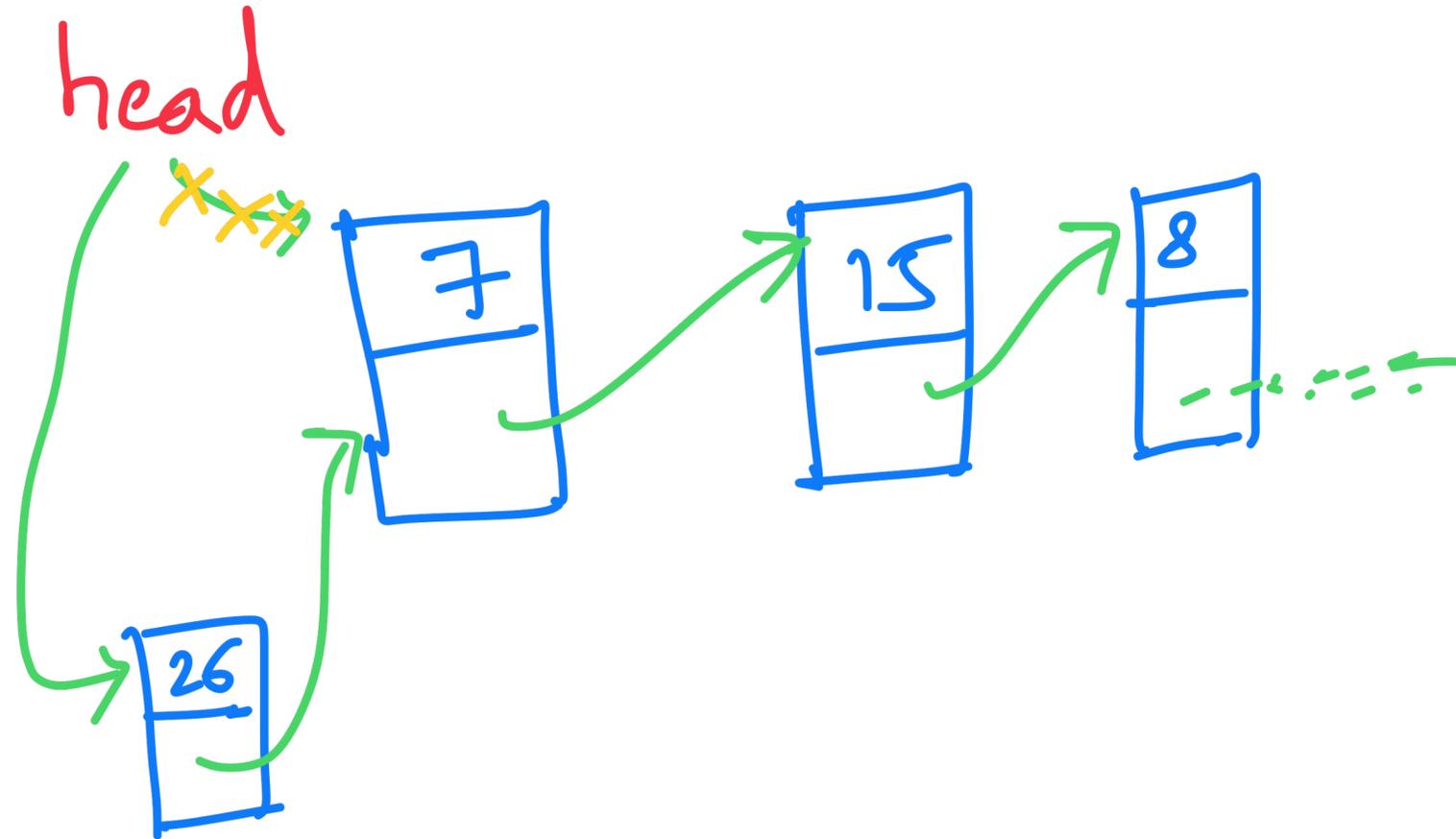
Insert element at beginning

insert 26?

$\text{Node}^* n = \text{new Node}();$

① $n \rightarrow \text{next} = \text{head};$

② $\text{head} = n;$

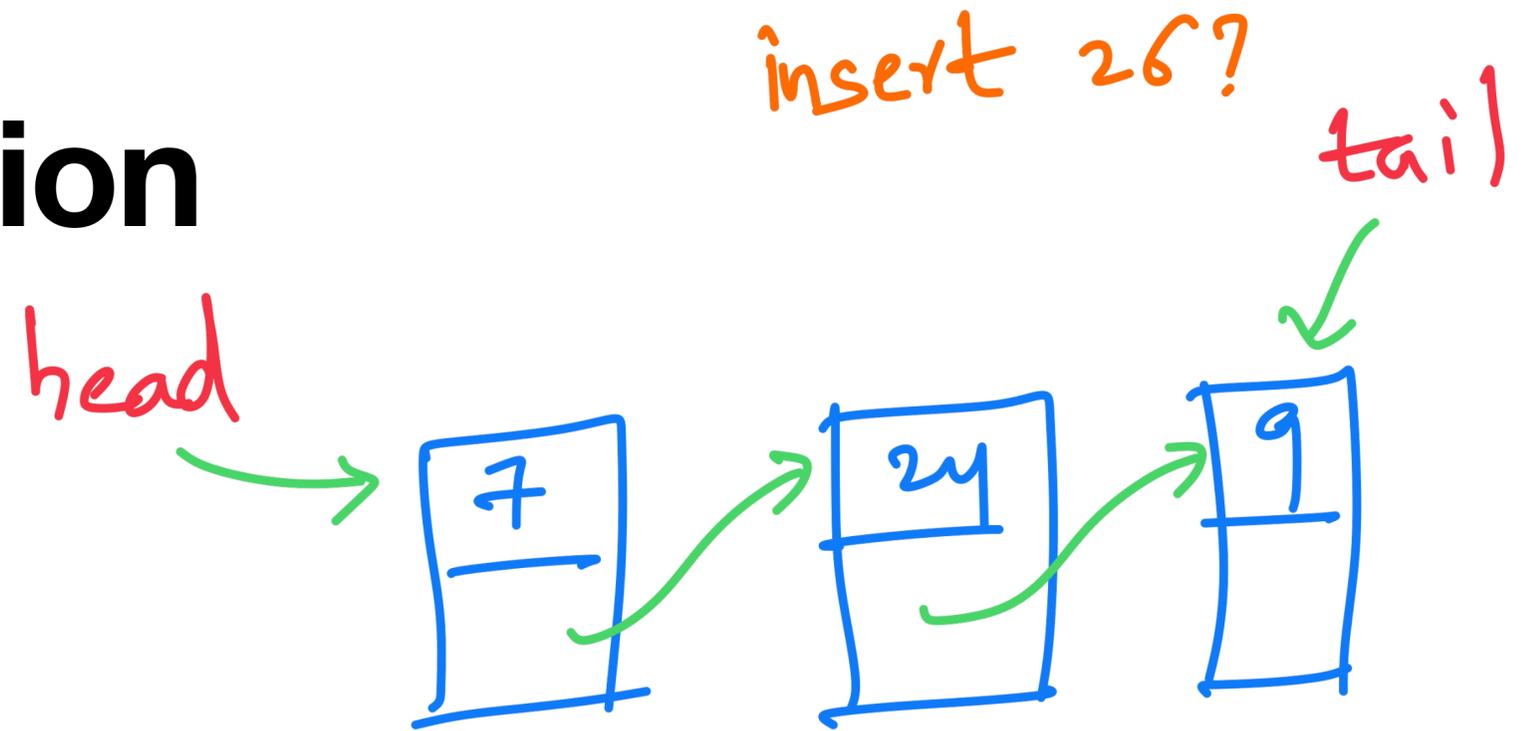


① create new node

② rewire!!

Linked Lists : Insertion

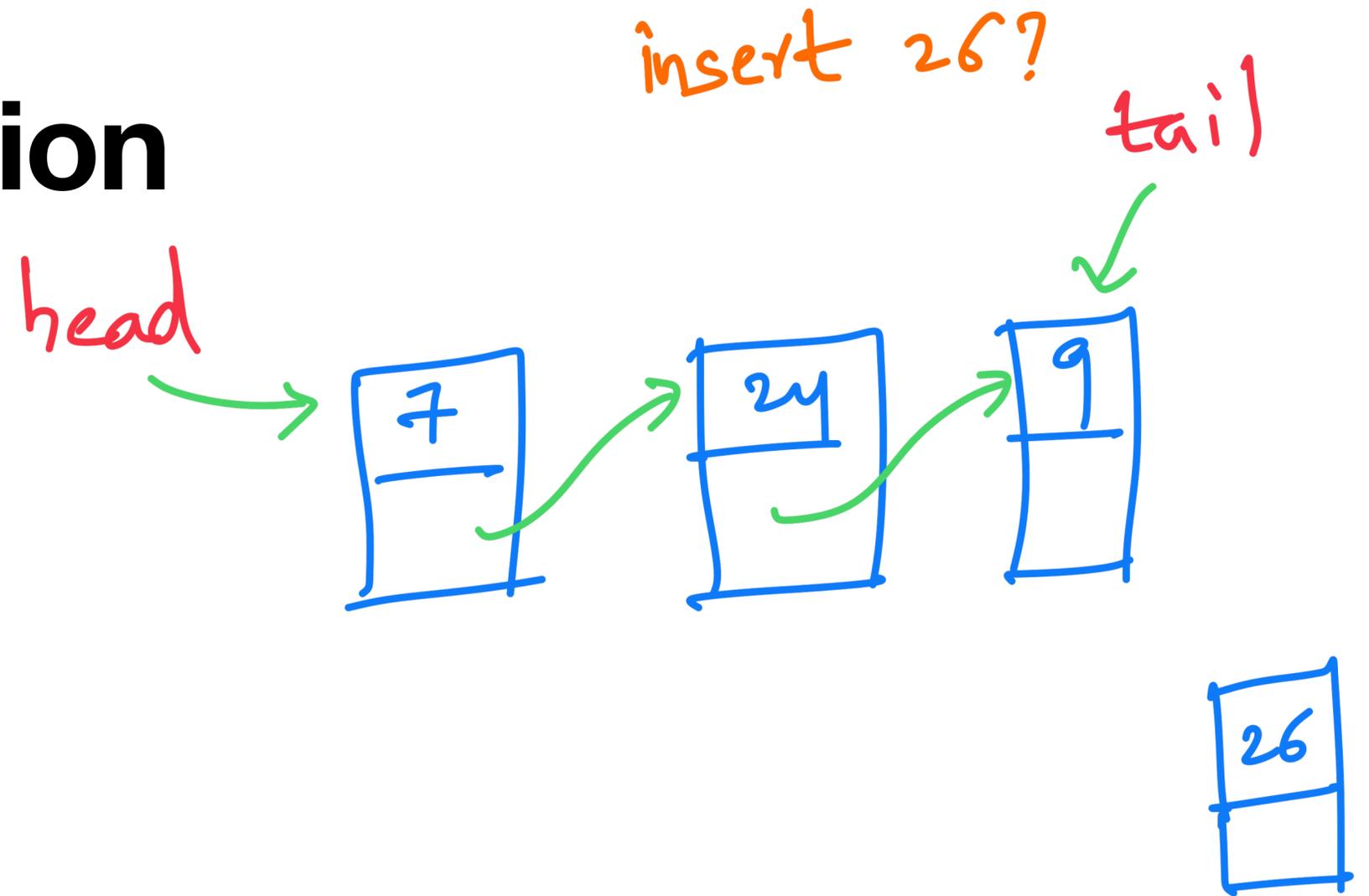
Insert element at end



Linked Lists : Insertion

Insert element at end

```
Node* n = new Node();
```

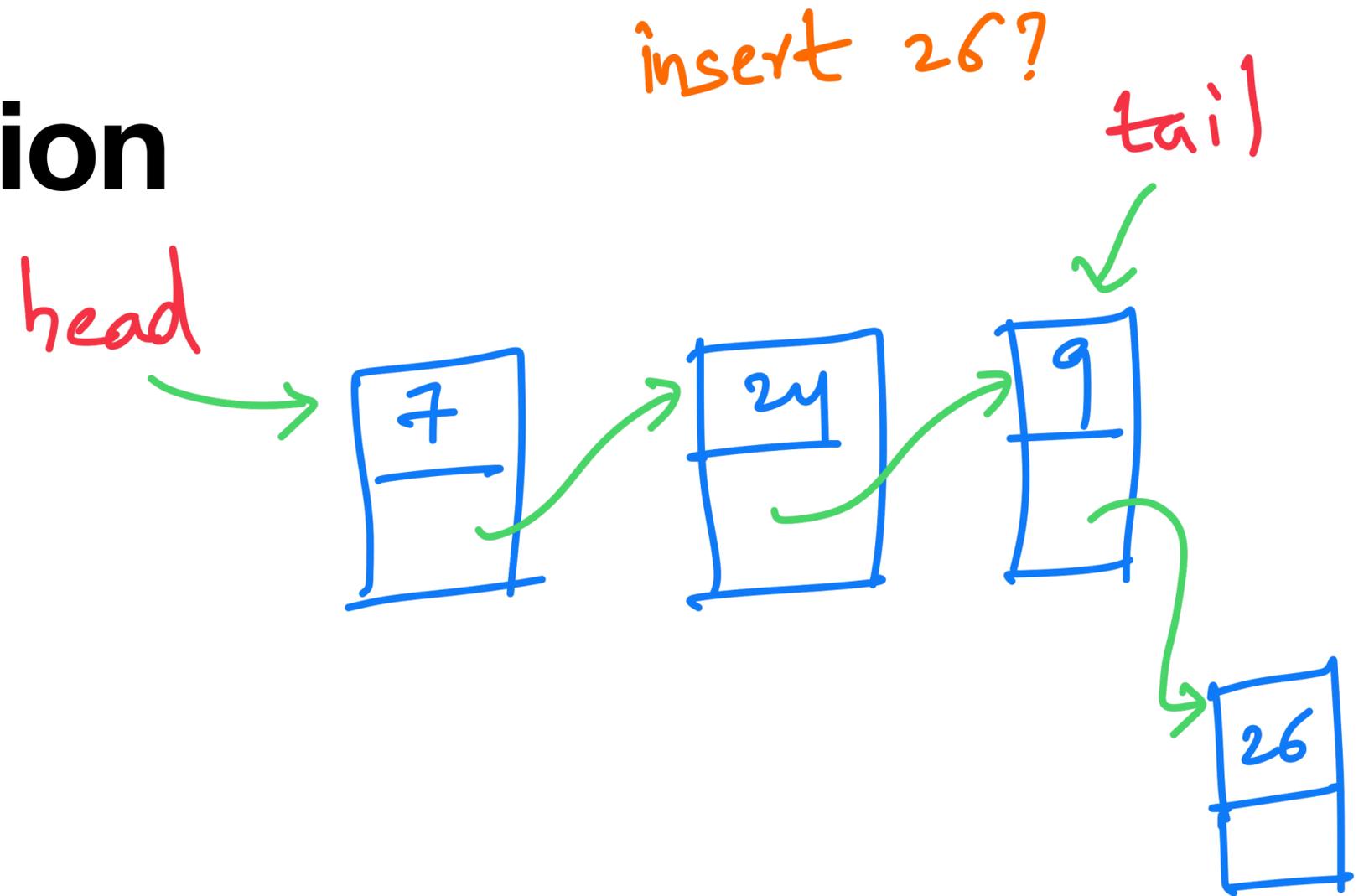


Linked Lists : Insertion

Insert element at end

`Node* n = new Node();`

① - - - - -

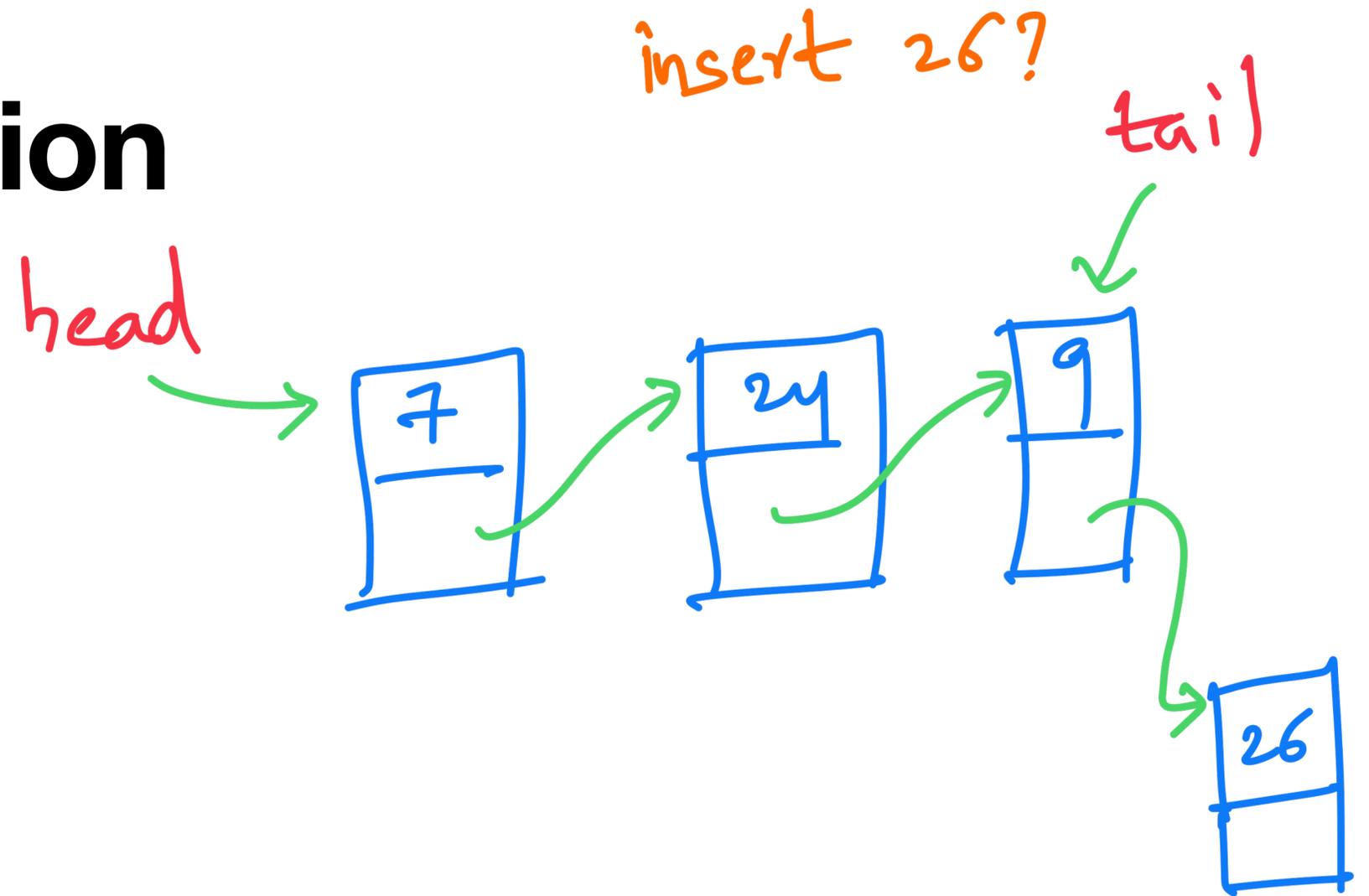


Linked Lists : Insertion

Insert element at end

`Node* n = new Node();`

① `tail->next = n;`



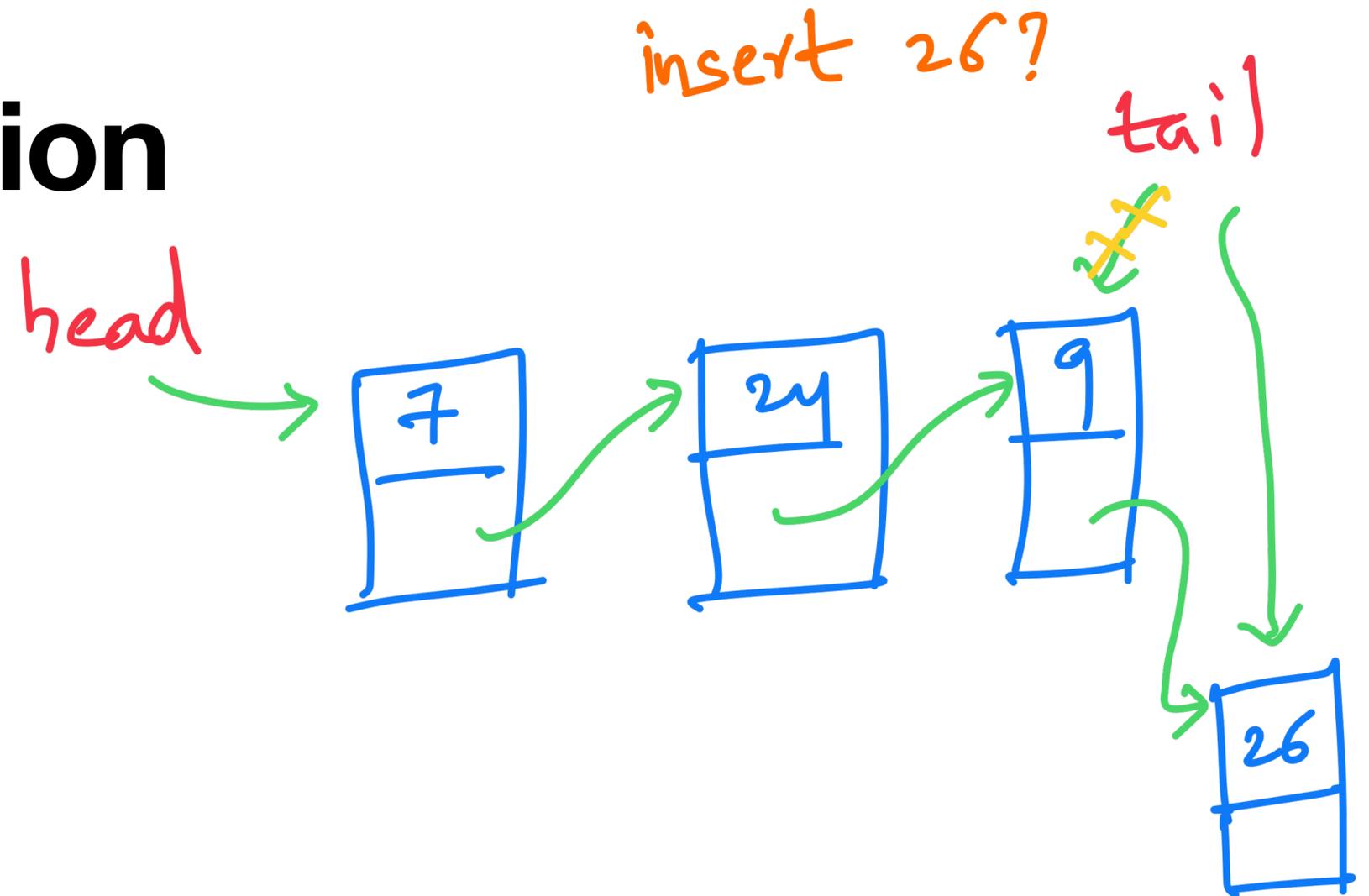
Linked Lists : Insertion

Insert element at end

`Node* n = new Node();`

① `tail->next = n;`

② _____



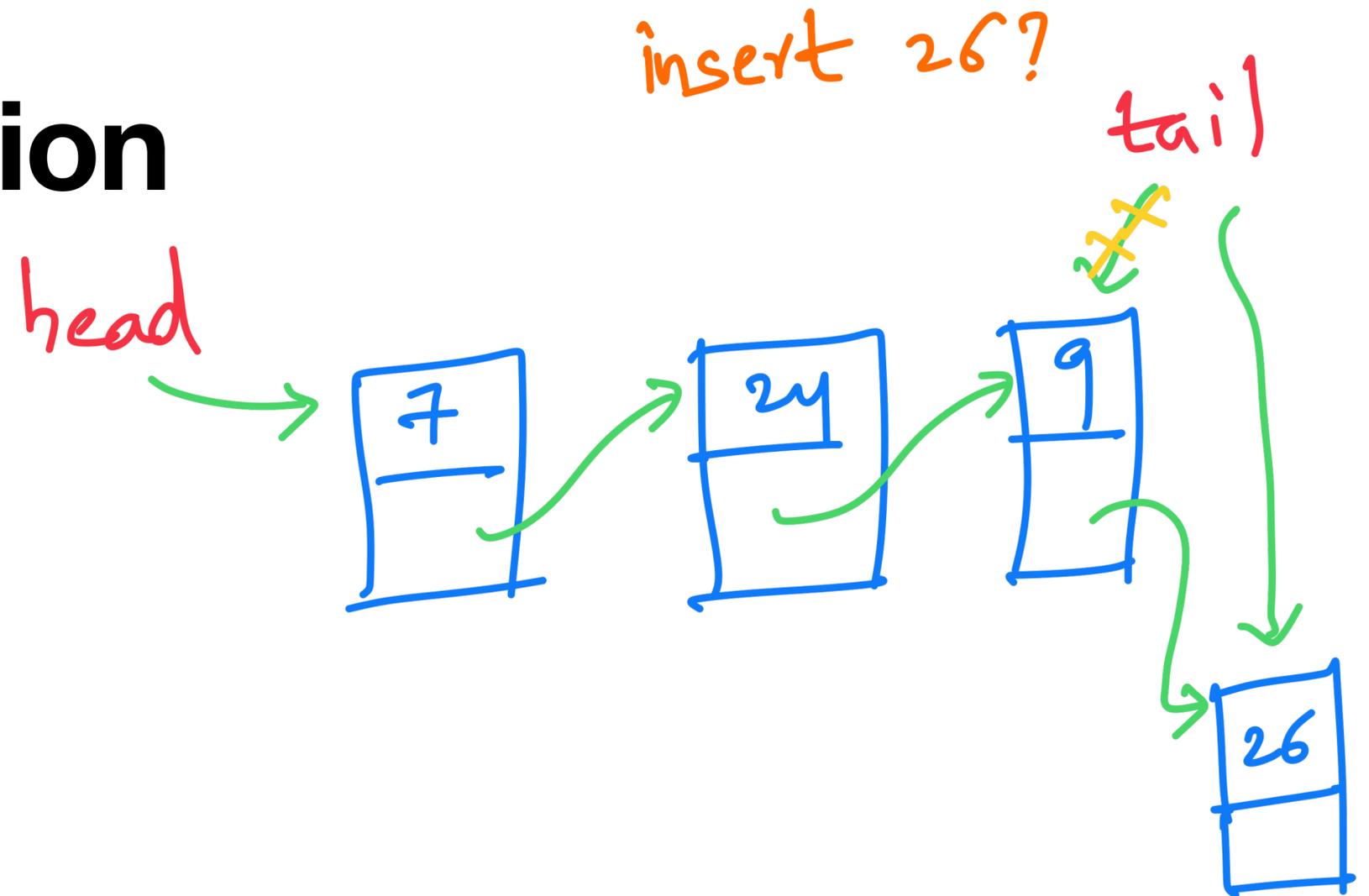
Linked Lists : Insertion

Insert element at end

`Node* n = new Node();`

① `tail->next = n;`

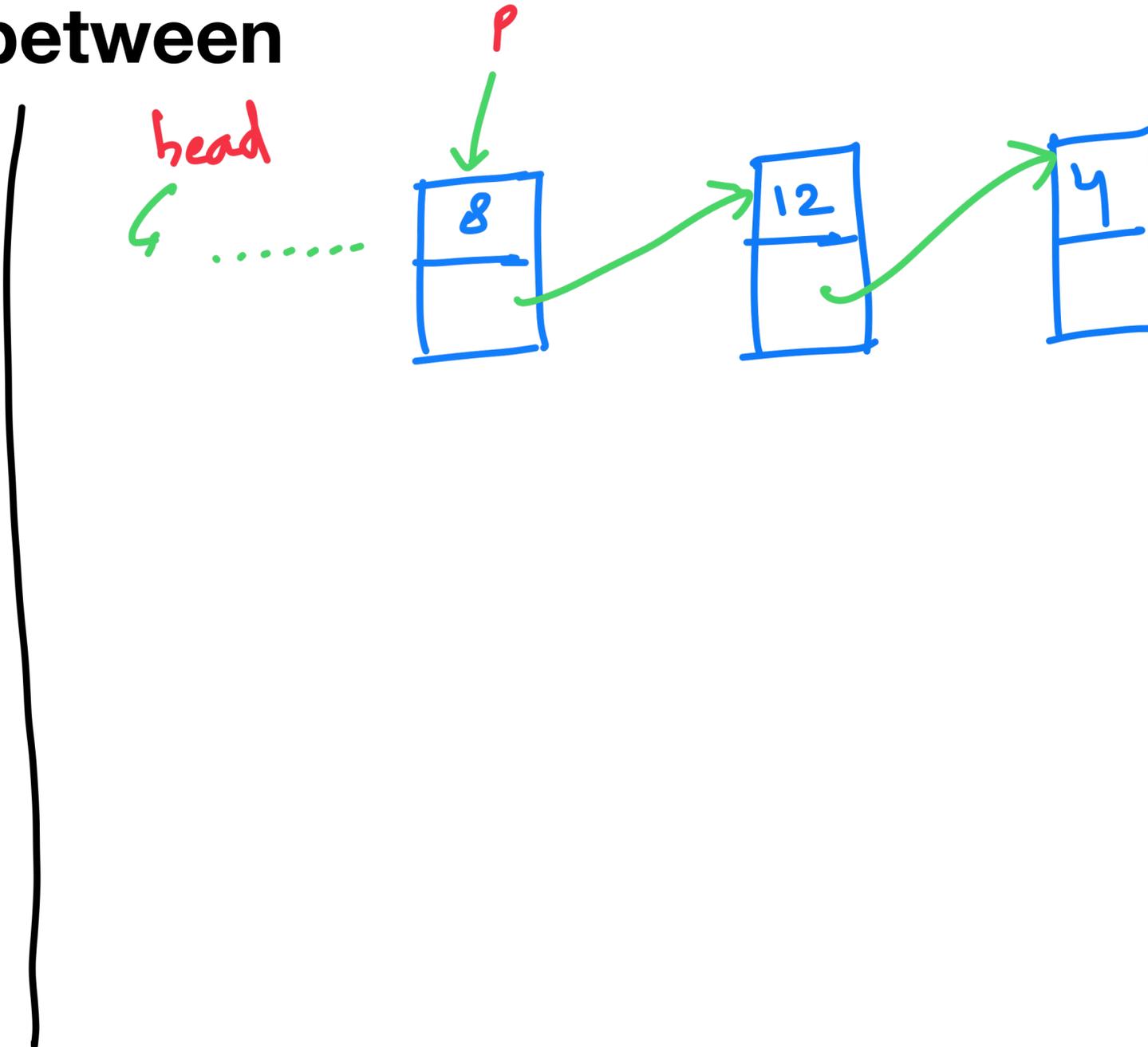
② `tail = n;`



Linked Lists : Deletion

Delete element in between

delete 12?

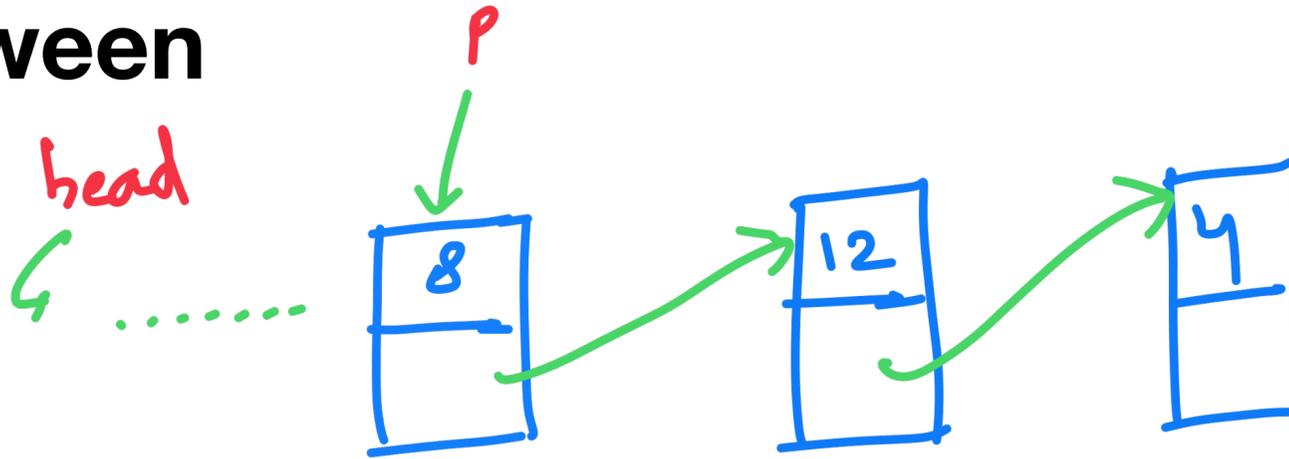


Linked Lists : Deletion

delete 12?

Delete element in between

①

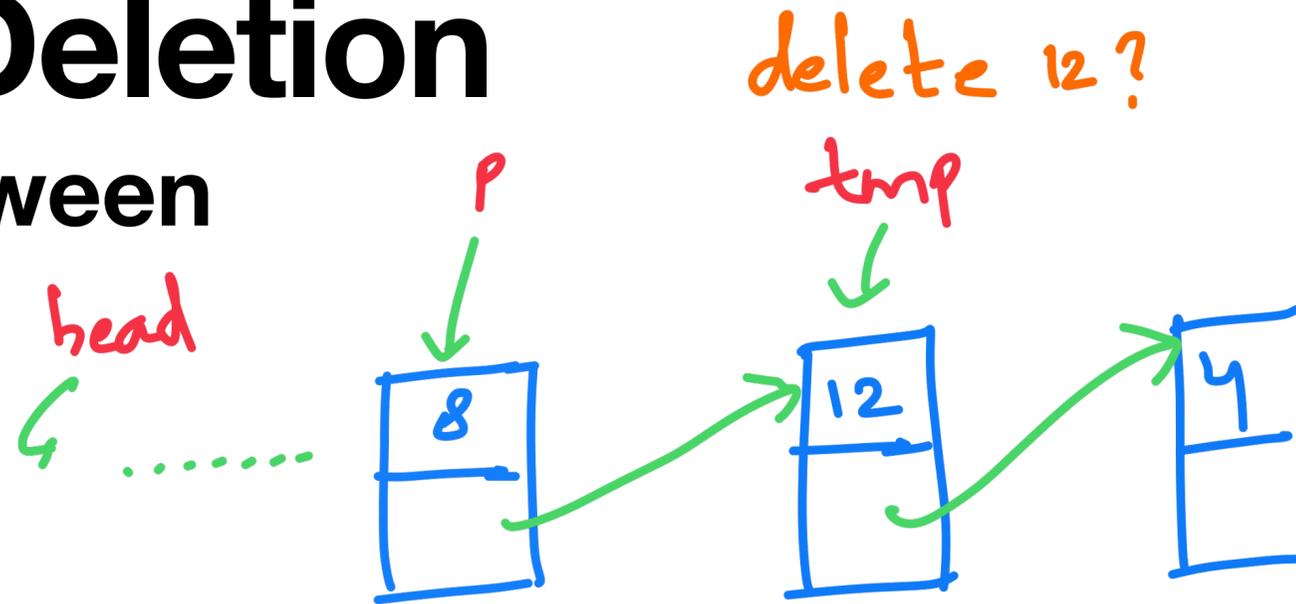


① Get the node to be deleted

Linked Lists : Deletion

Delete element in between

① node* tmp = P->next;



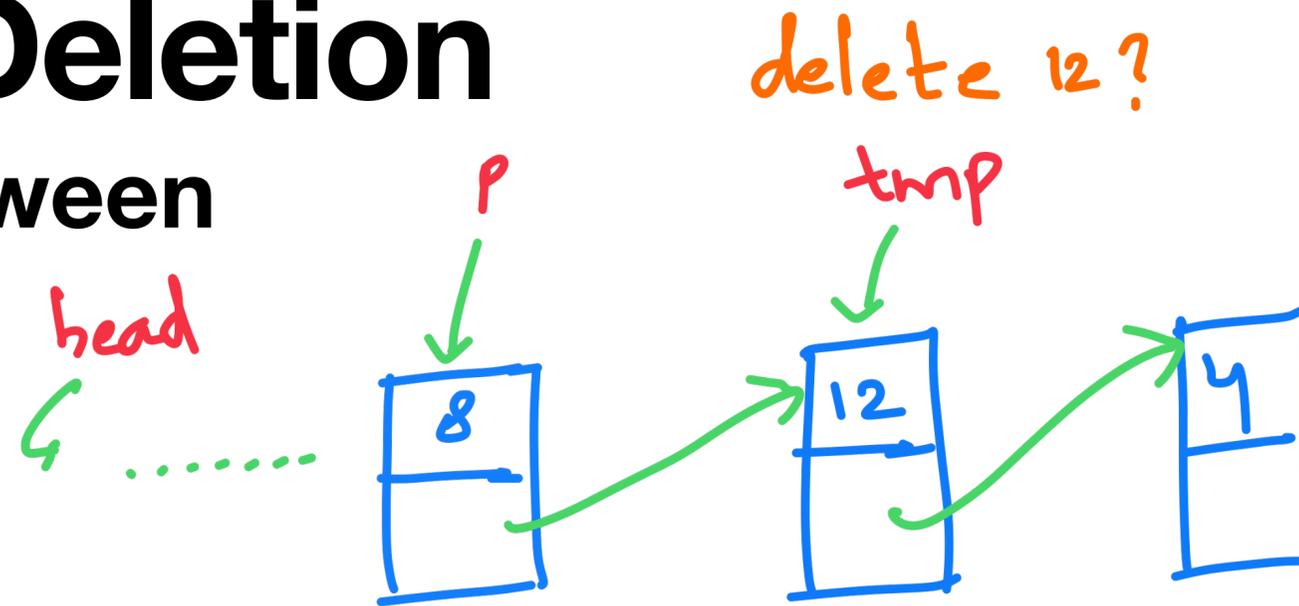
① Get the node to be deleted

Linked Lists : Deletion

Delete element in between

① `node* tmp = P->next;`

② -----



① Get the node to be deleted

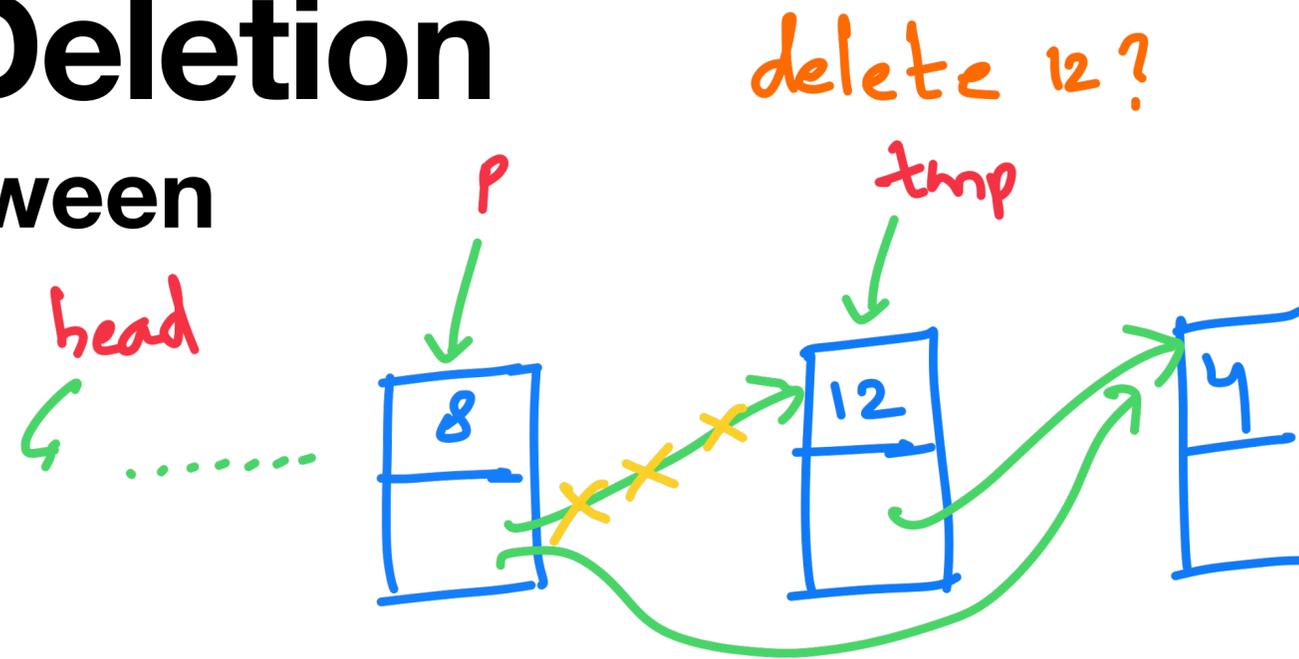
② rewire!

Linked Lists : Deletion

Delete element in between

① `node* tmp = P->next;`

② -----



① Get the node to be deleted

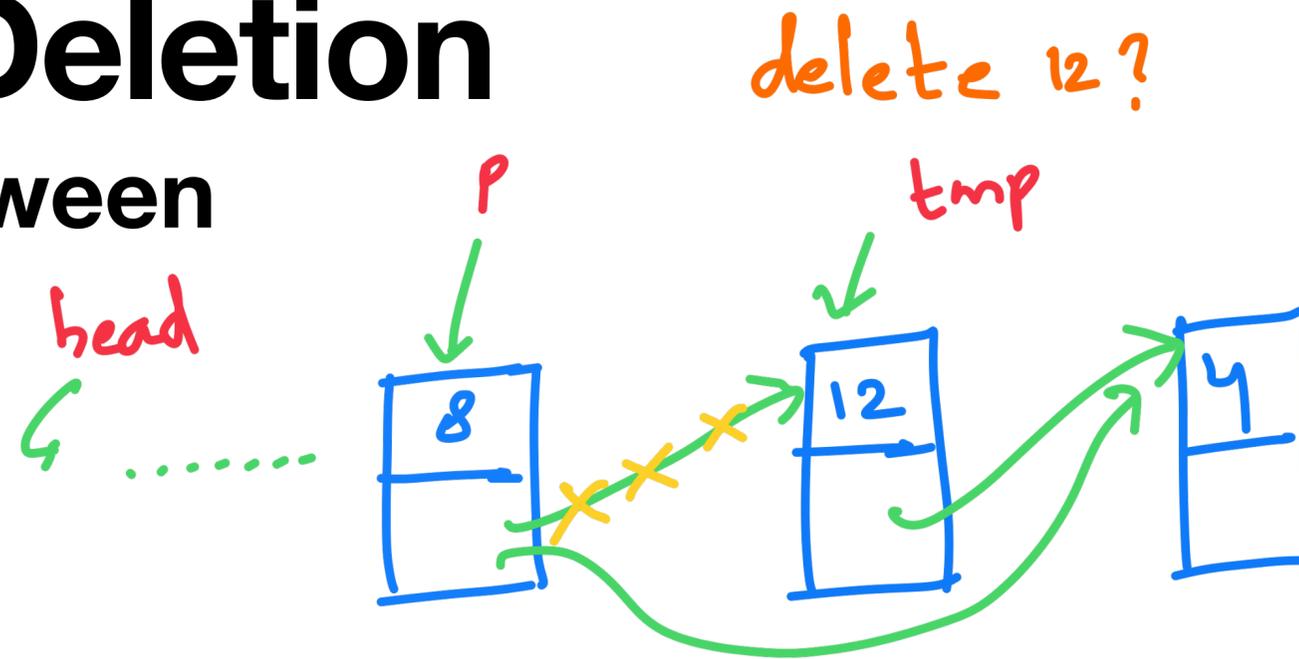
② rewire !

Linked Lists : Deletion

Delete element in between

① $\text{Node}^* \text{ tmp} = \text{P} \rightarrow \text{next};$

② $\text{P} \rightarrow \text{next} = \text{tmp} \rightarrow \text{next};$



① Get the node to be deleted

② rewire!

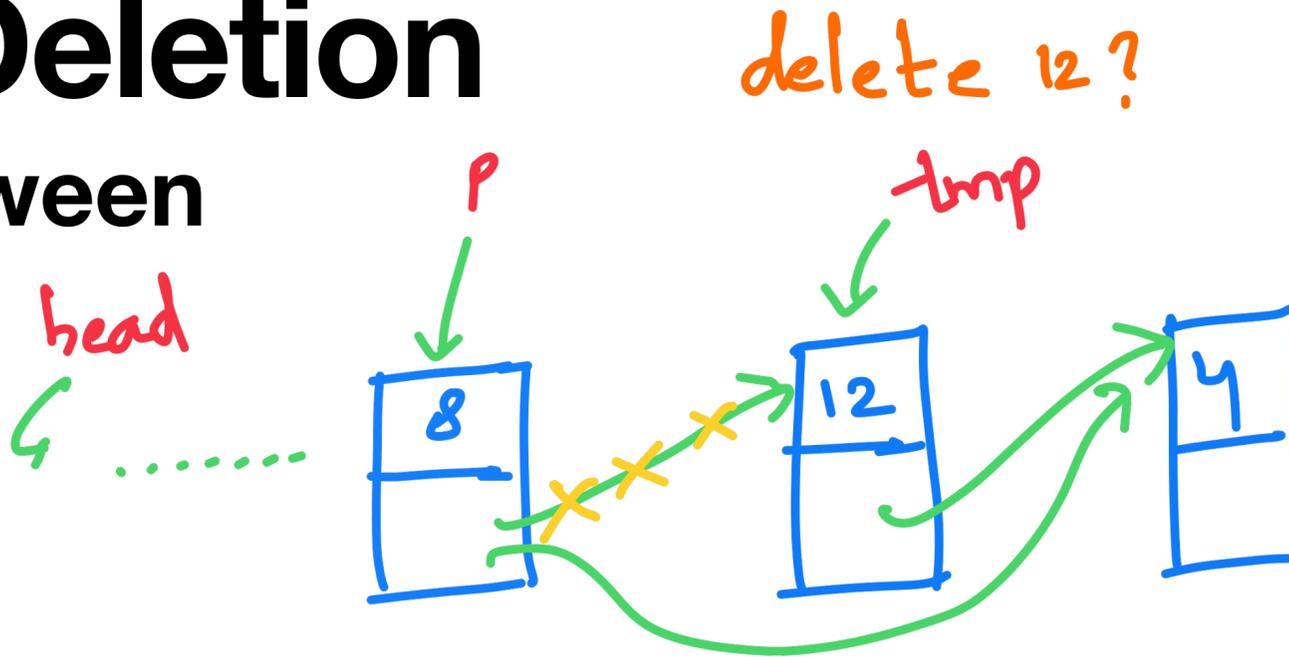
Linked Lists : Deletion

Delete element in between

① `node* tmp = P->next;`

② `P->next = tmp->next;`

③ -----



① Get the node to be deleted

② rewire!

③ -----

What's the third step?

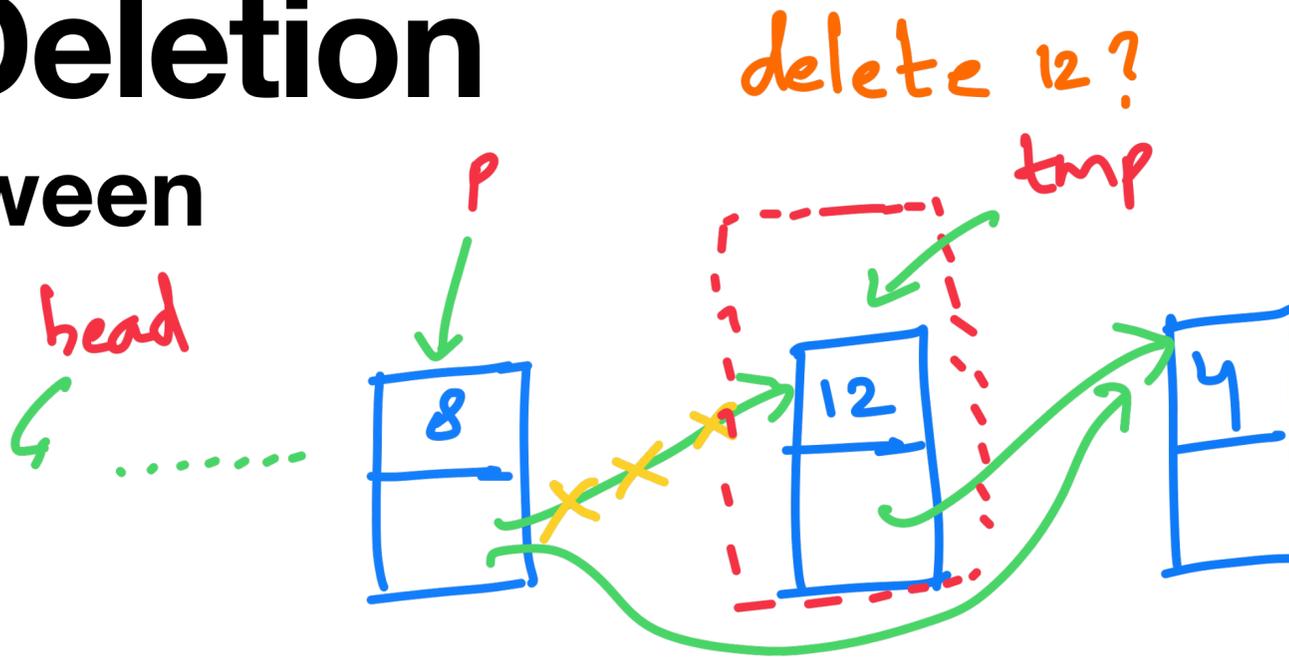
Linked Lists : Deletion

Delete element in between

① node* tmp = P->next;

② P->next = tmp->next;

③ delete tmp;



① Get the node to be deleted

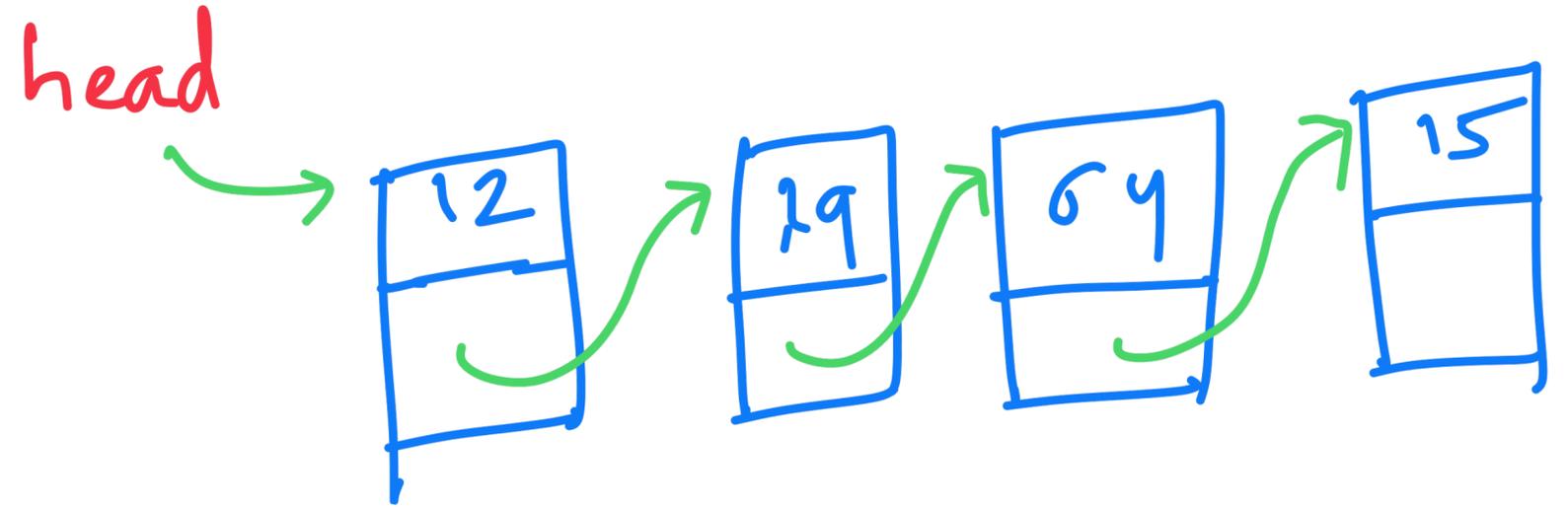
② rewire!

③ delete

Linked Lists : Deletion

Delete first element

delete 12?

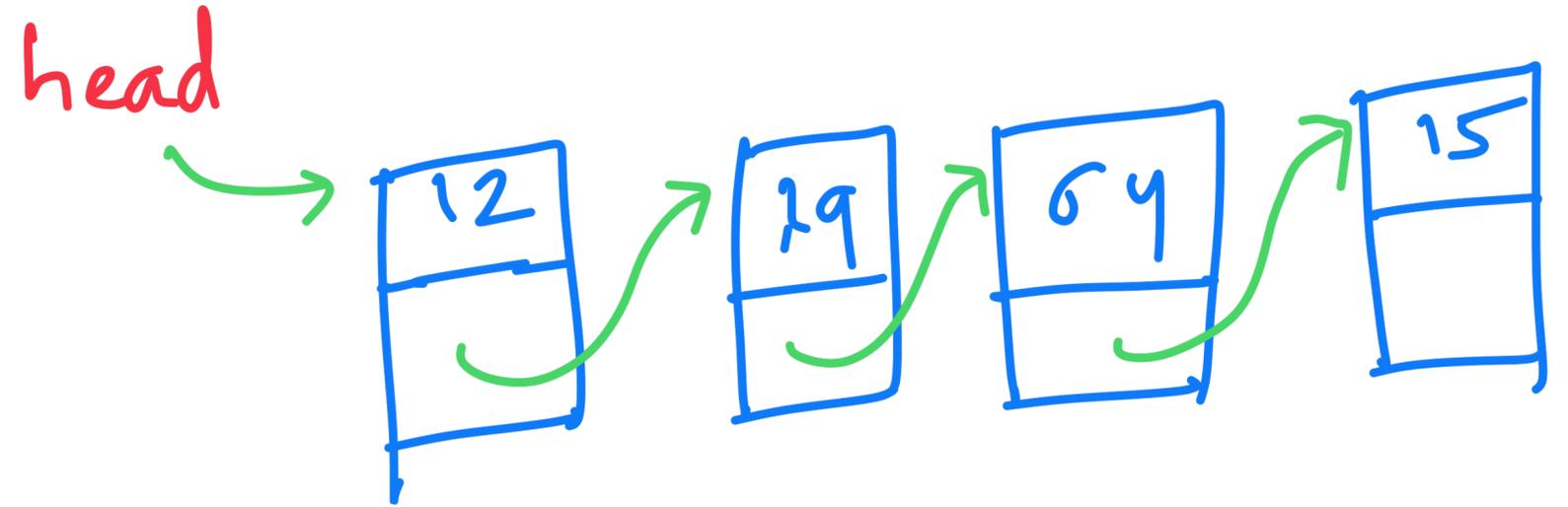


Linked Lists : Deletion

Delete first element



delete 12?



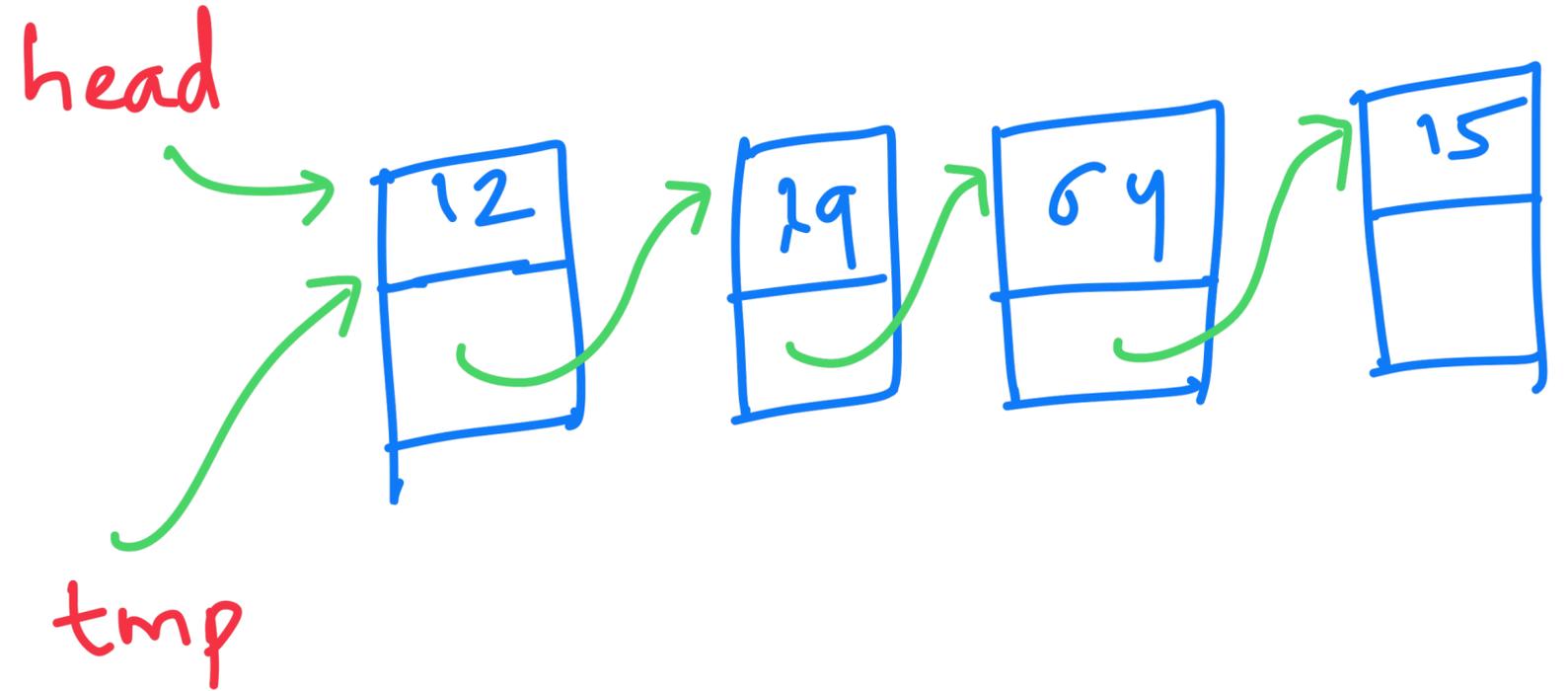
① Get the node to be deleted

Linked Lists : Deletion

Delete first element

① Node* tmp = head;

delete 12?



① Get the node to be deleted

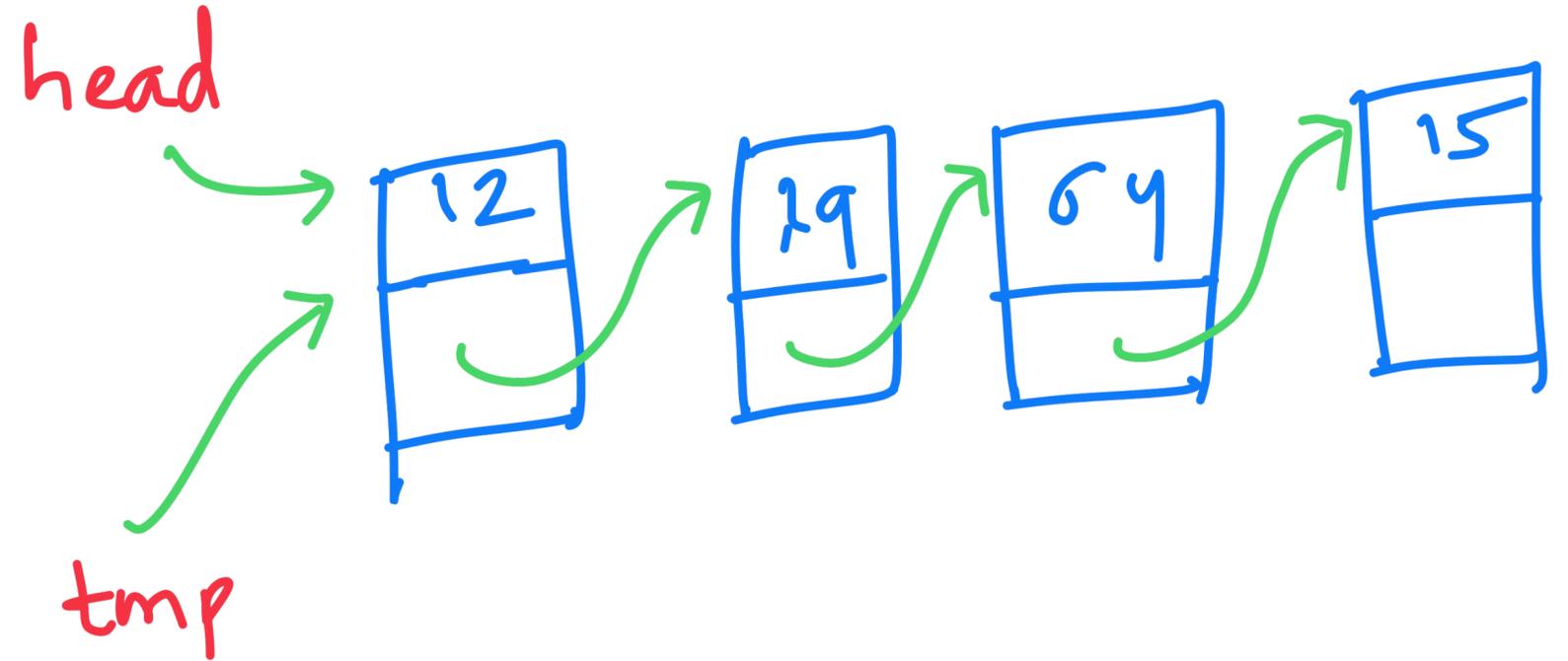
Linked Lists : Deletion

Delete first element

delete 12?

① `Node* tmp = head;`

② -----



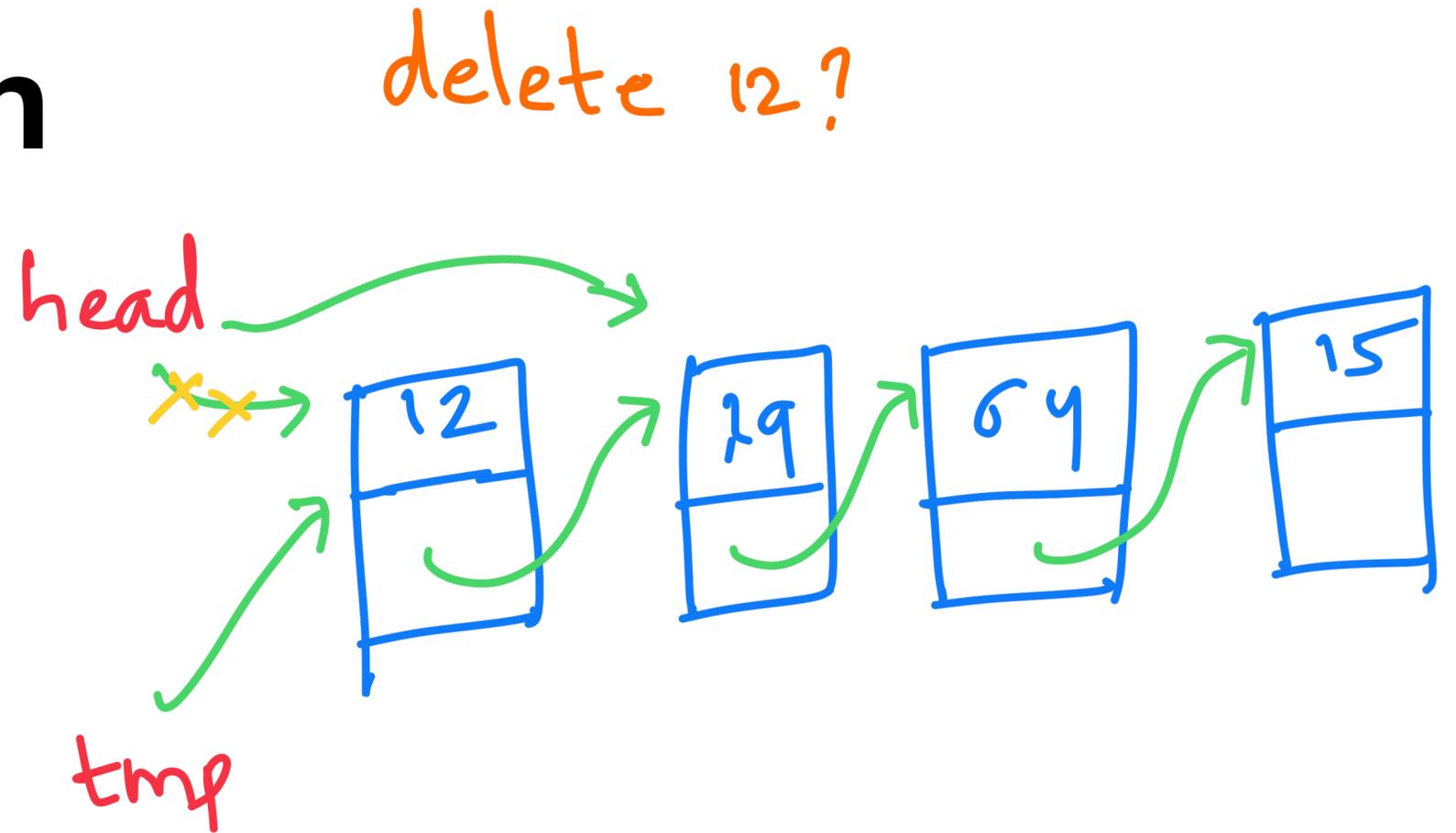
- ① Get the node to be deleted
- ② rewire!

Linked Lists : Deletion

Delete first element

① `Node* tmp = head;`

② -----

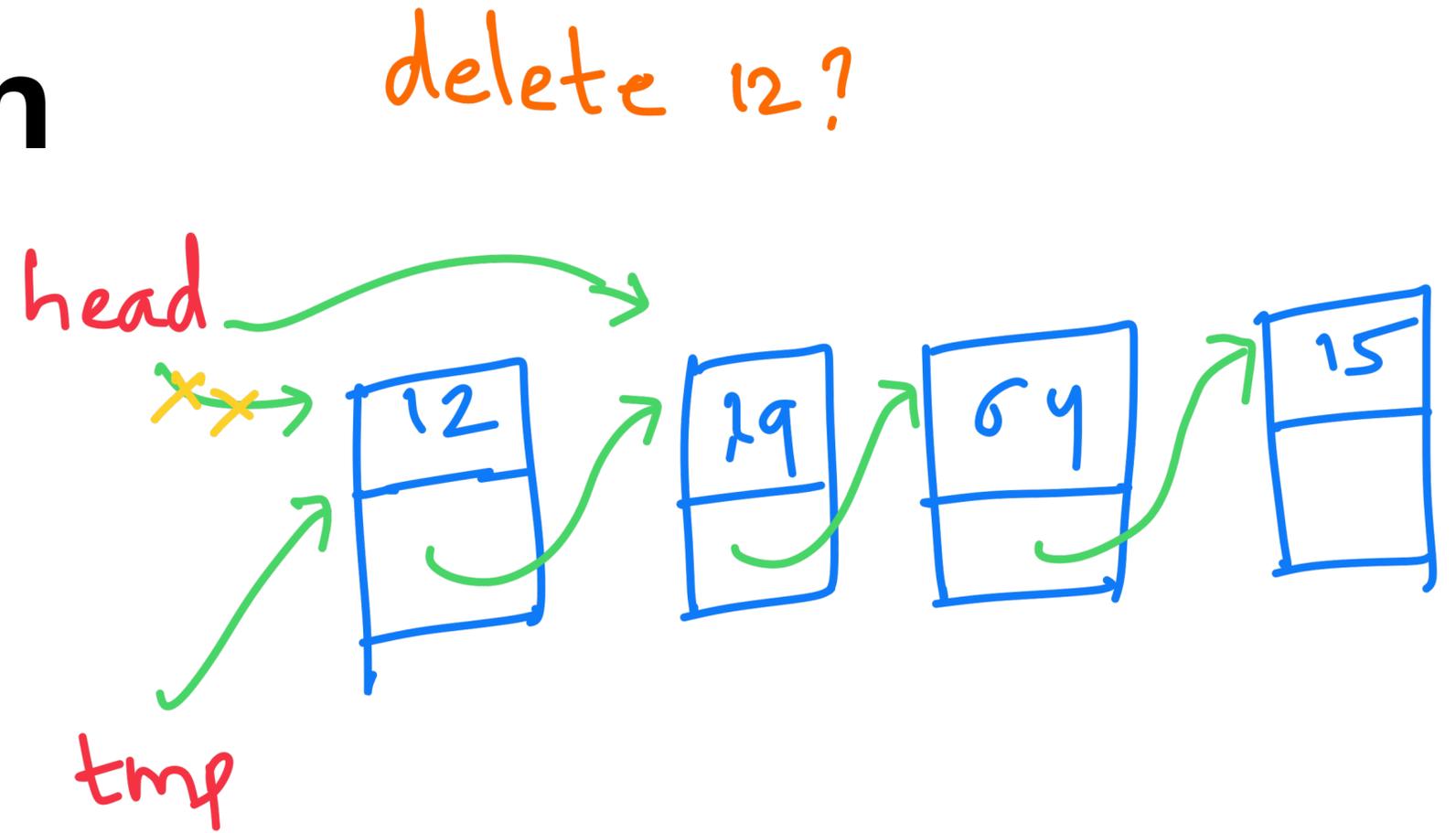


- ① Get the node to be deleted
- ② rewire!

Linked Lists : Deletion

Delete first element

- ① `Node* tmp = head;`
- ② `head = tmp->next;`

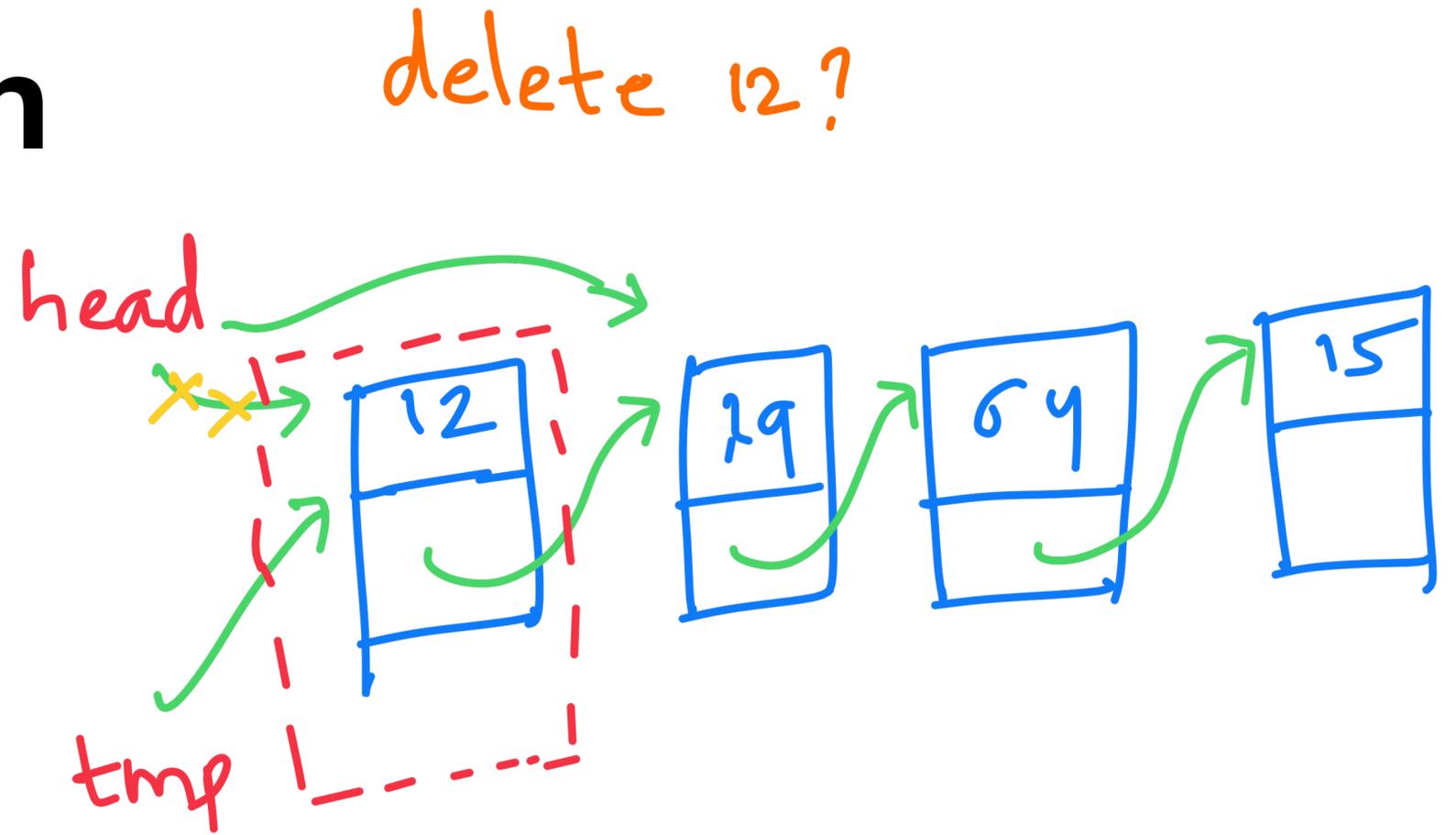


- ① Get the node to be deleted
- ② rewire!

Linked Lists : Deletion

Delete first element

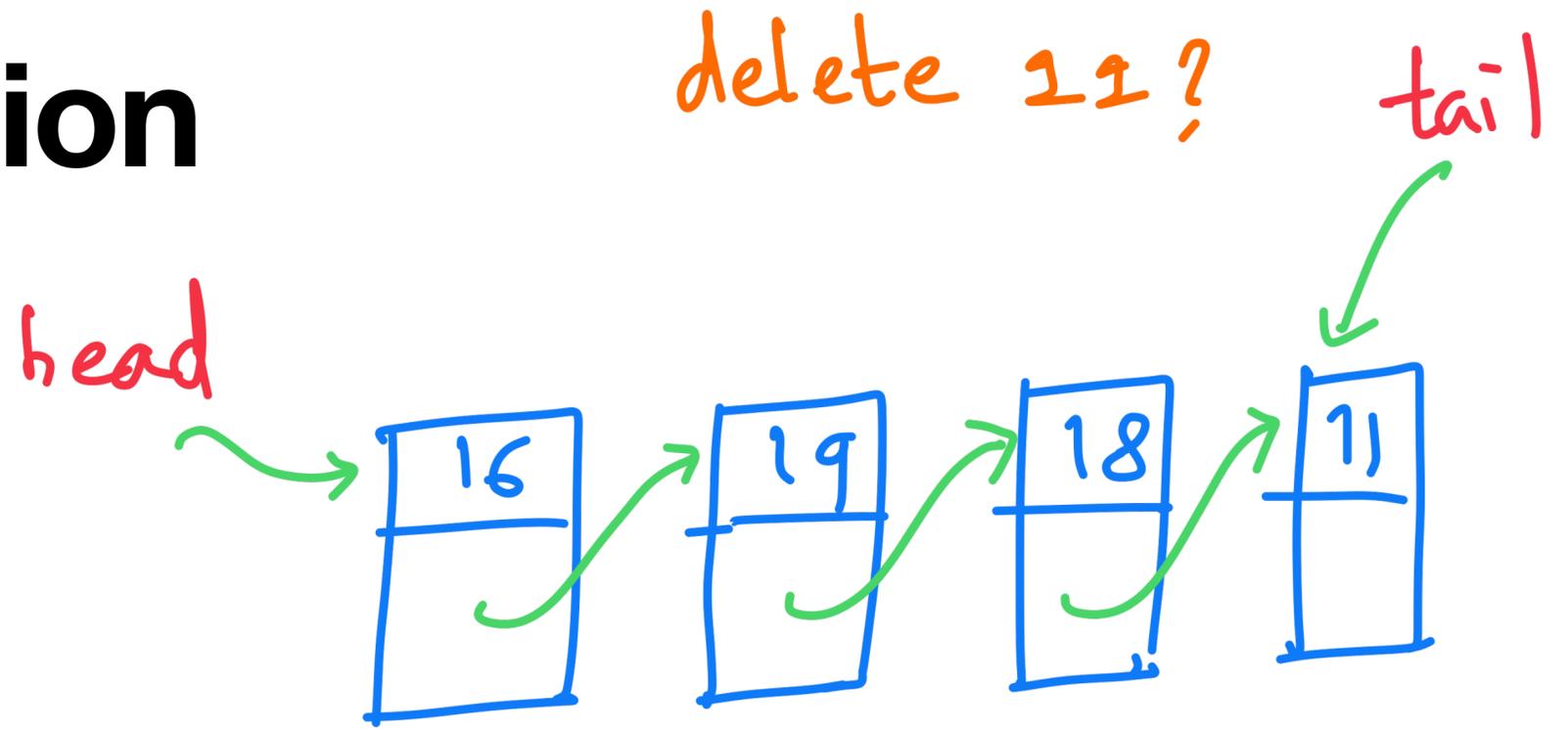
- ① `Node* tmp = head;`
- ② `head = tmp->next;`
- ③ `delete tmp;`



- ① Get the node to be deleted
- ② rewire!
- ③ delete

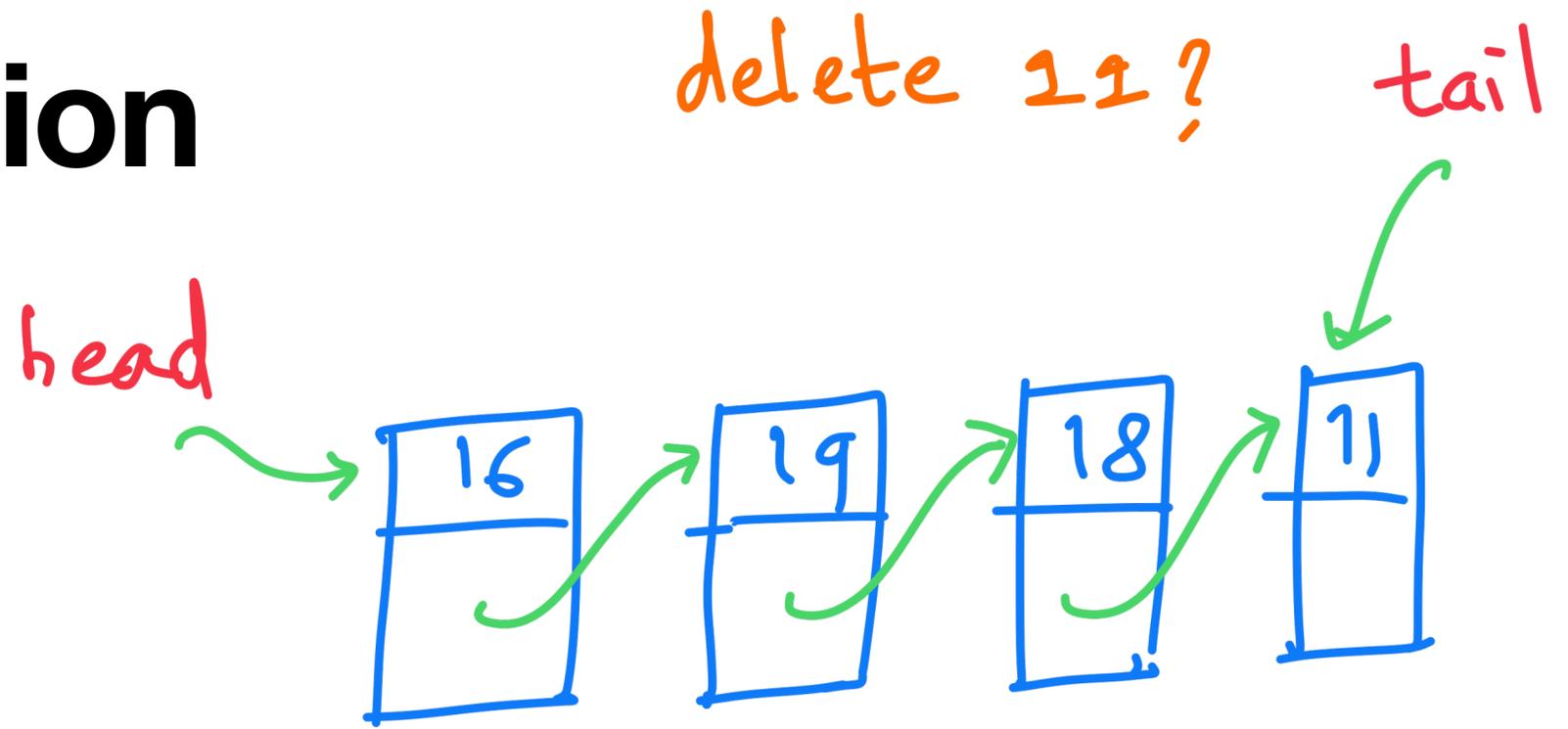
Linked Lists : Deletion

Delete last element



Linked Lists : Deletion

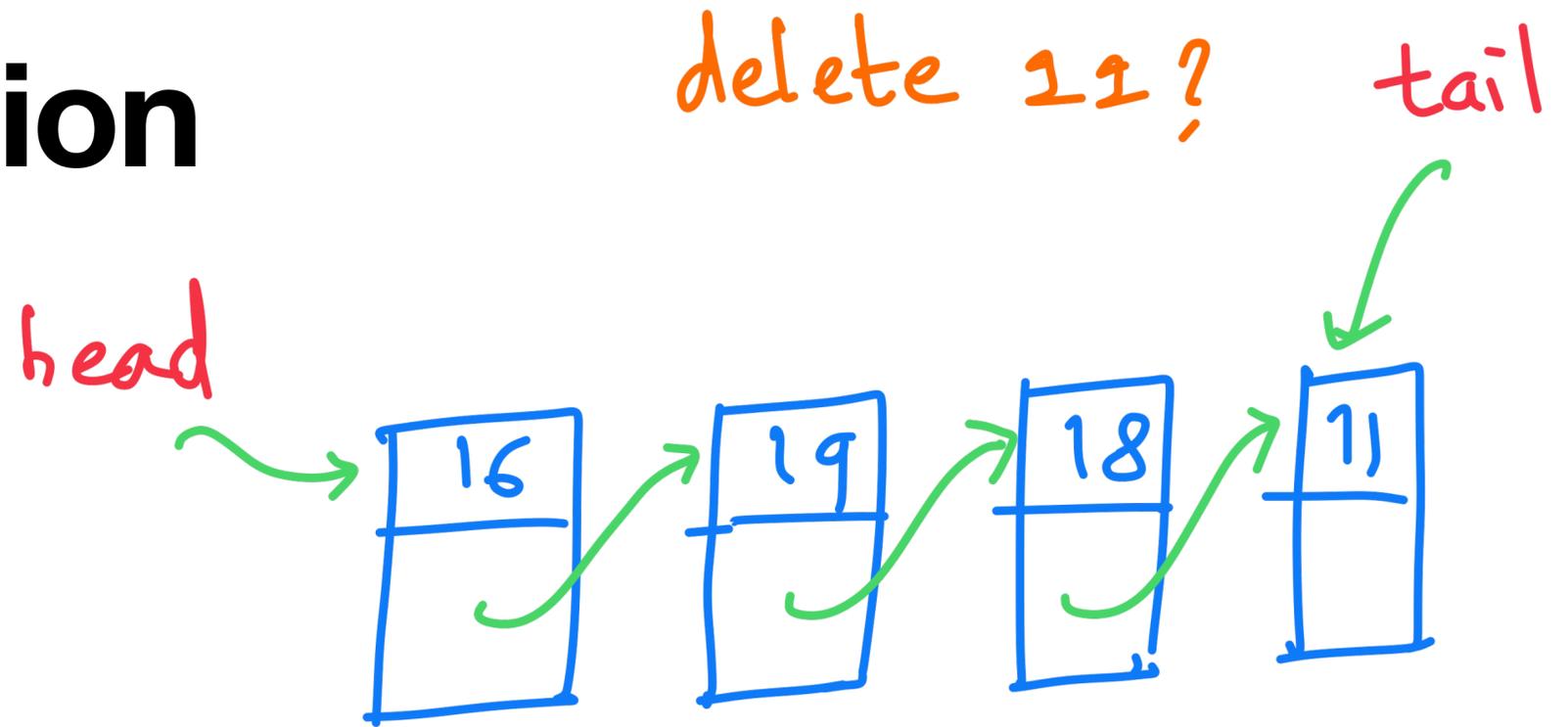
Delete last element



① Get the node to be deleted

Linked Lists : Deletion

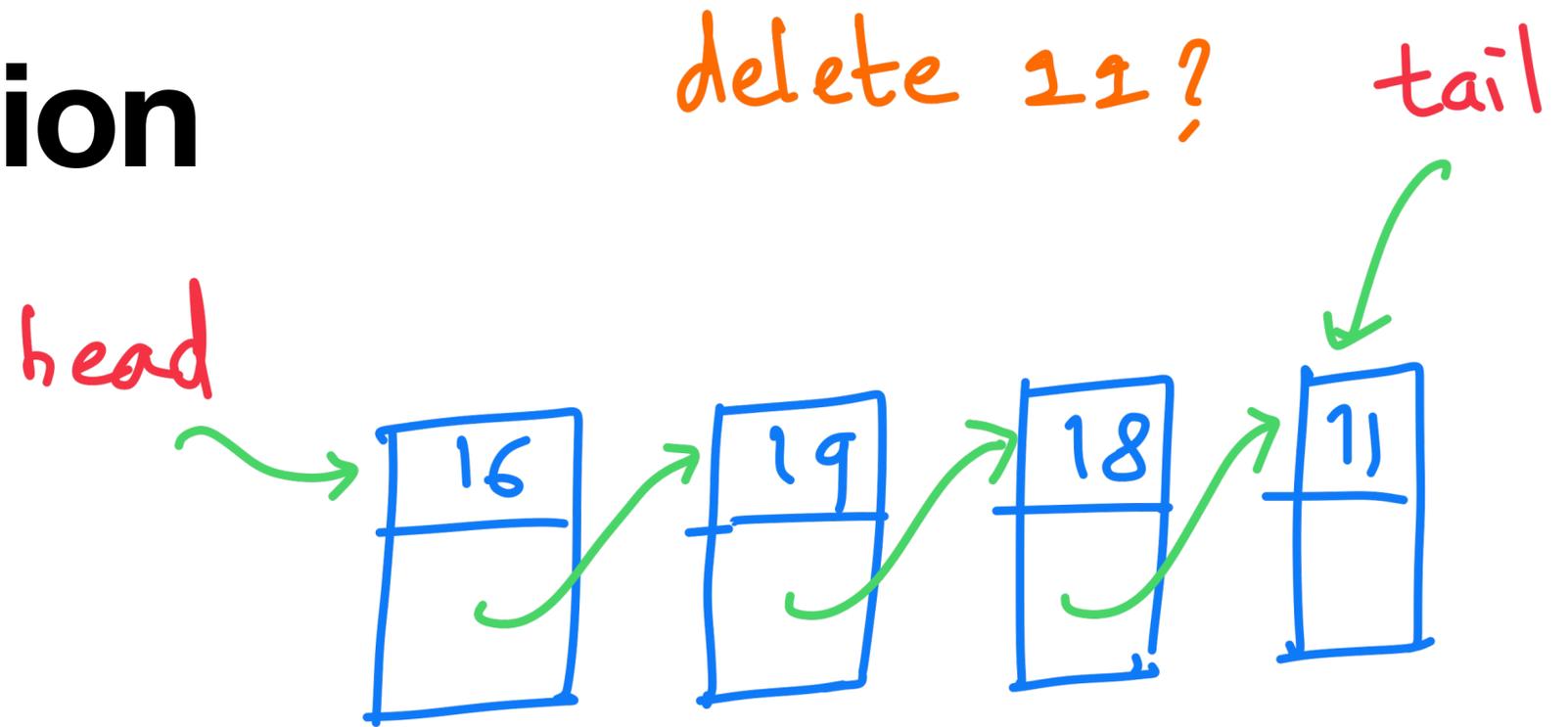
Delete last element



- ① Get the node to be deleted
it's the tail

Linked Lists : Deletion

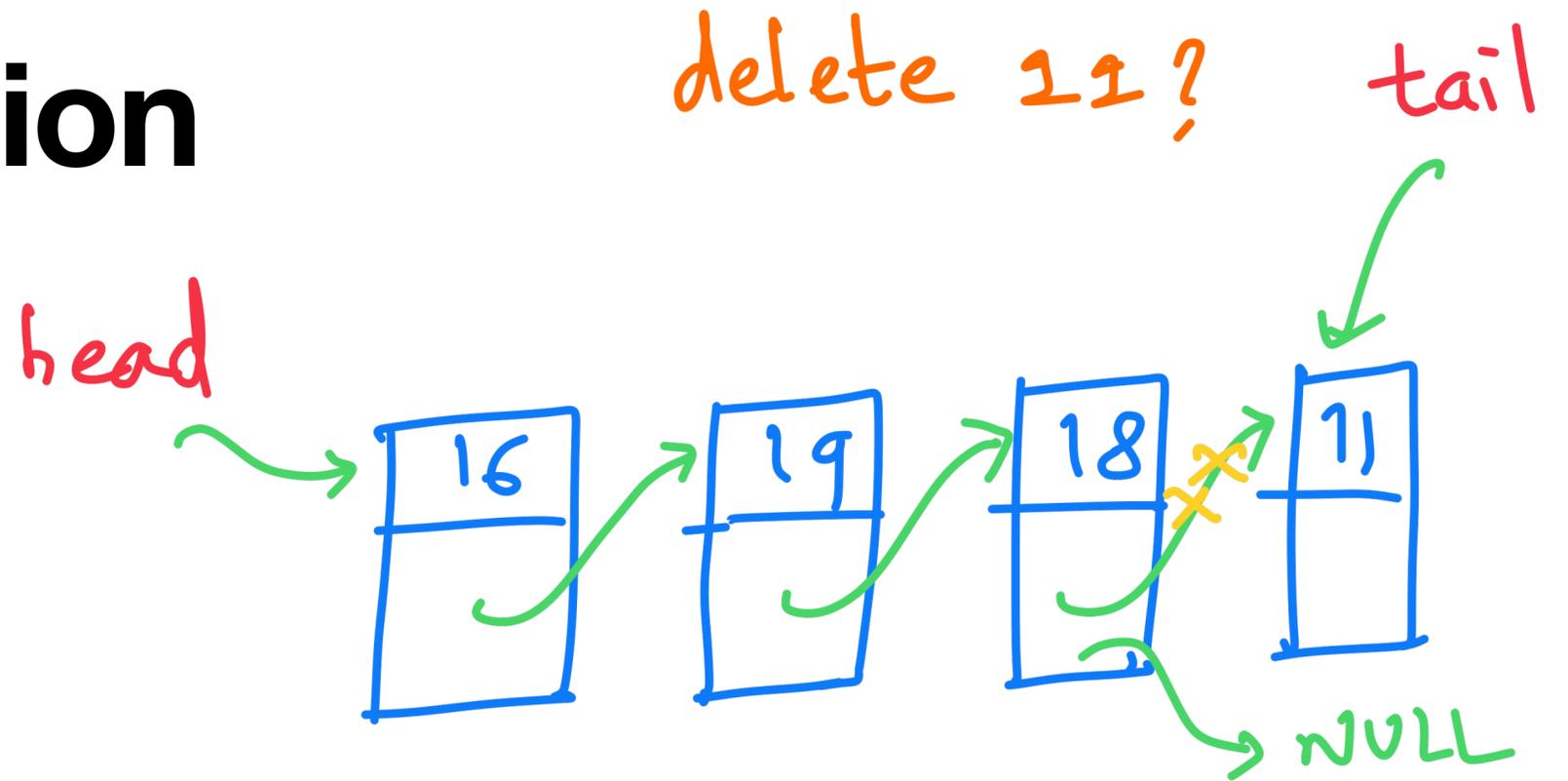
Delete last element



- ① Get the node to be deleted
it's the tail
- ② rewire

Linked Lists : Deletion

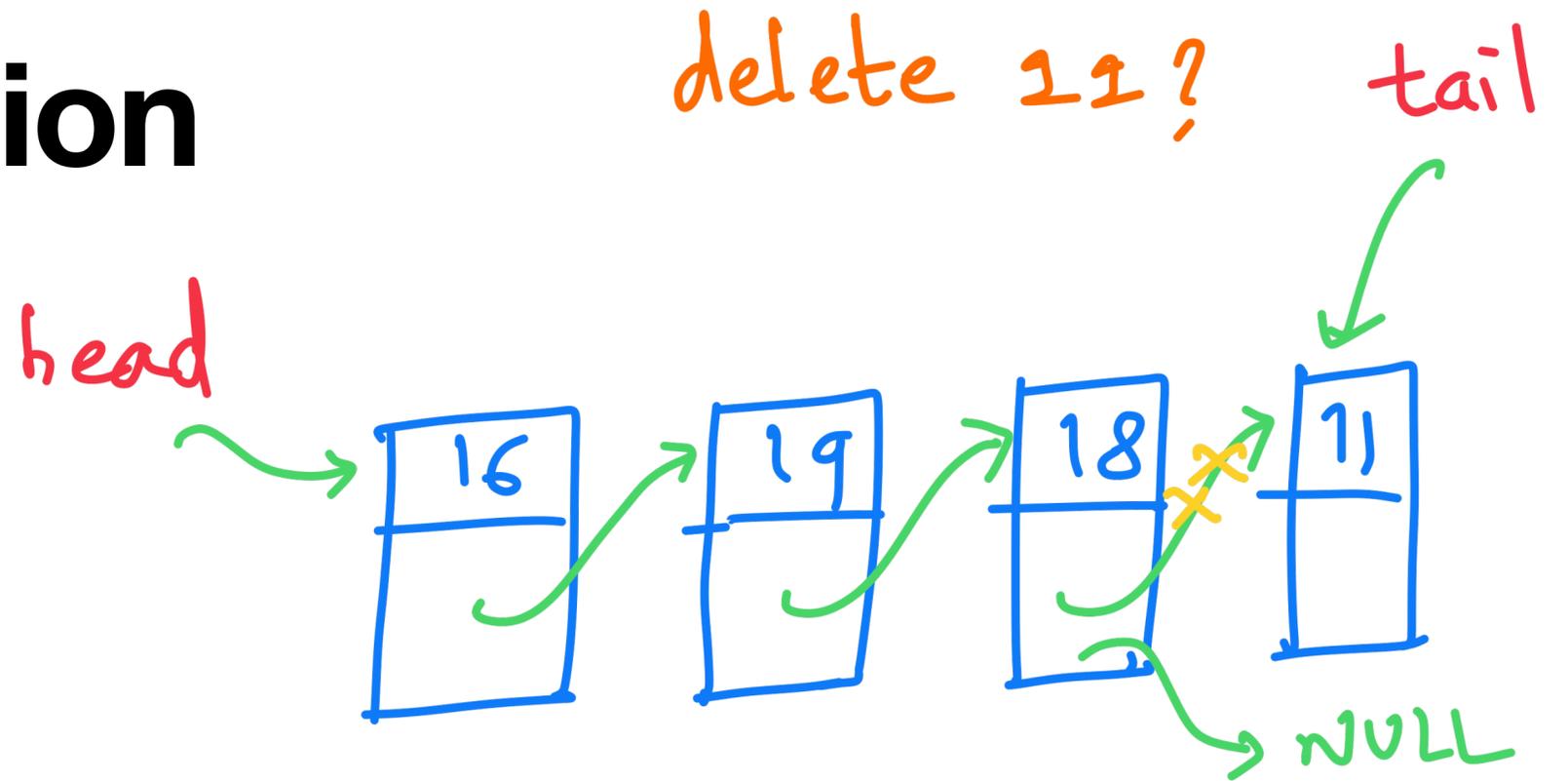
Delete last element



- ① Get the node to be deleted
it's the tail
- ② rewire

Linked Lists : Deletion

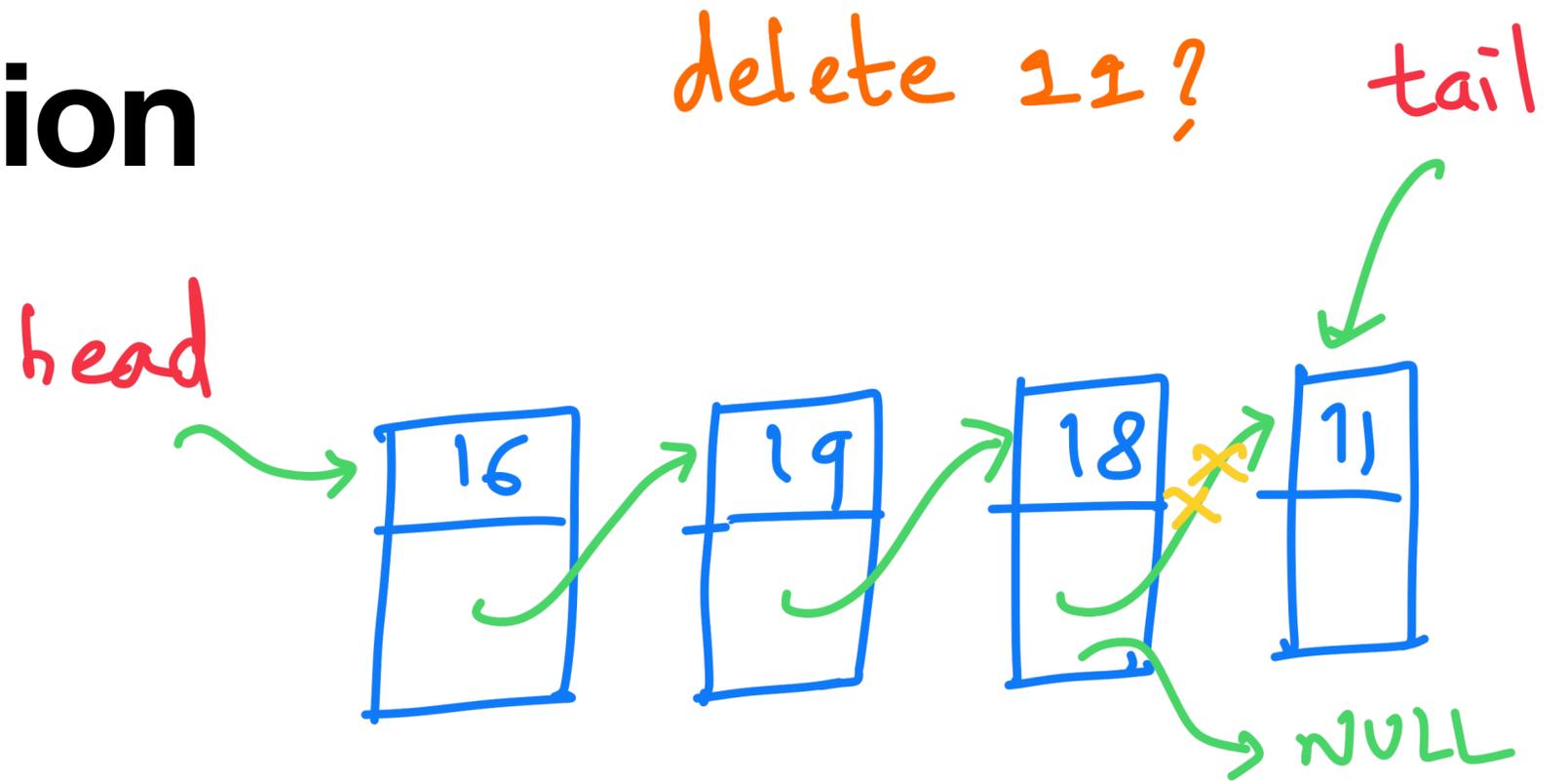
Delete last element



- ① Get the node to be deleted
it's the tail
- ② rewiring
but we need node '18'

Linked Lists : Deletion

Delete last element

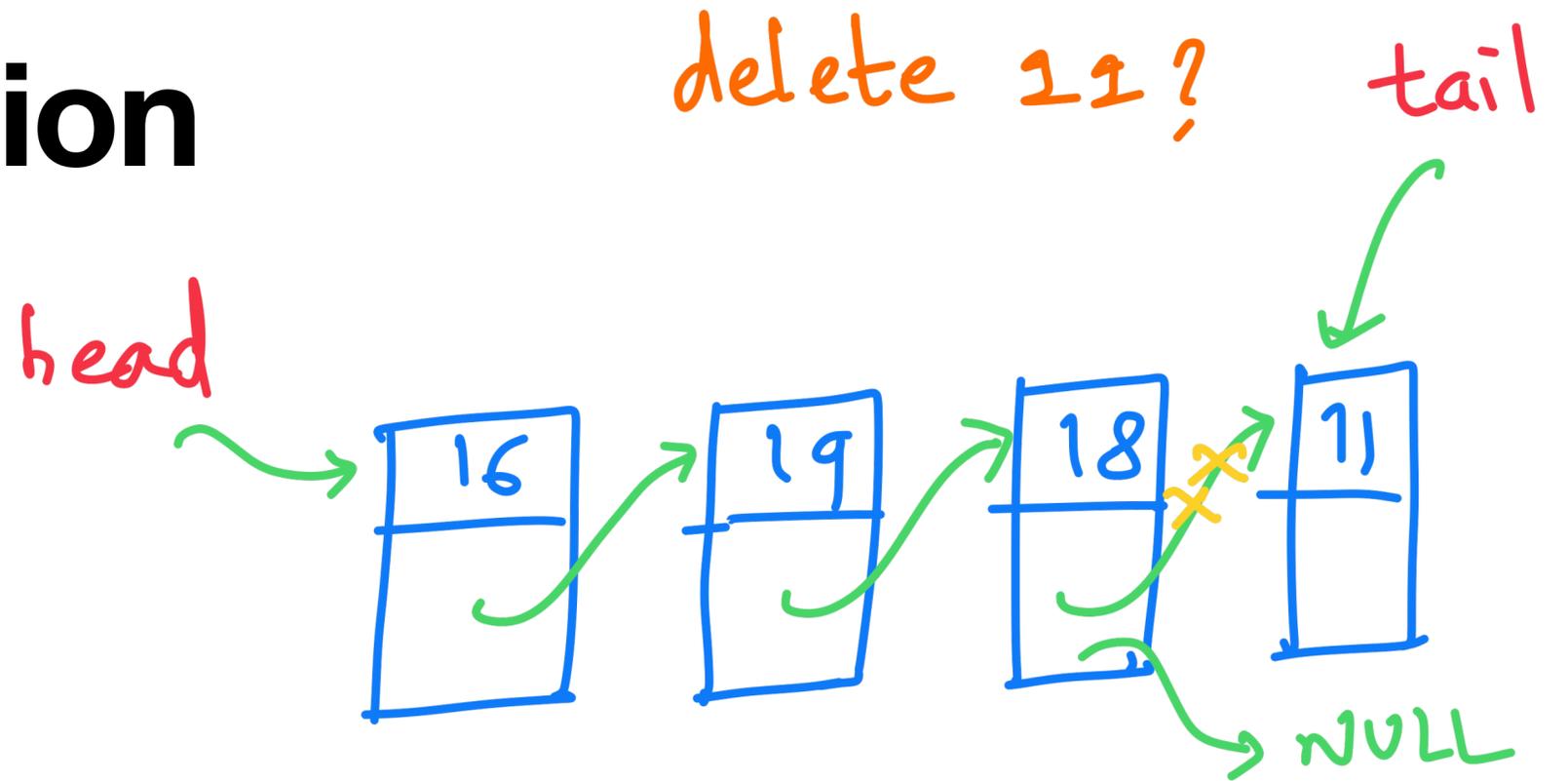


① Get the node to be deleted
it's the tail

② rewire
but we need node '18'
how to get it?

Linked Lists : Deletion

Delete last element

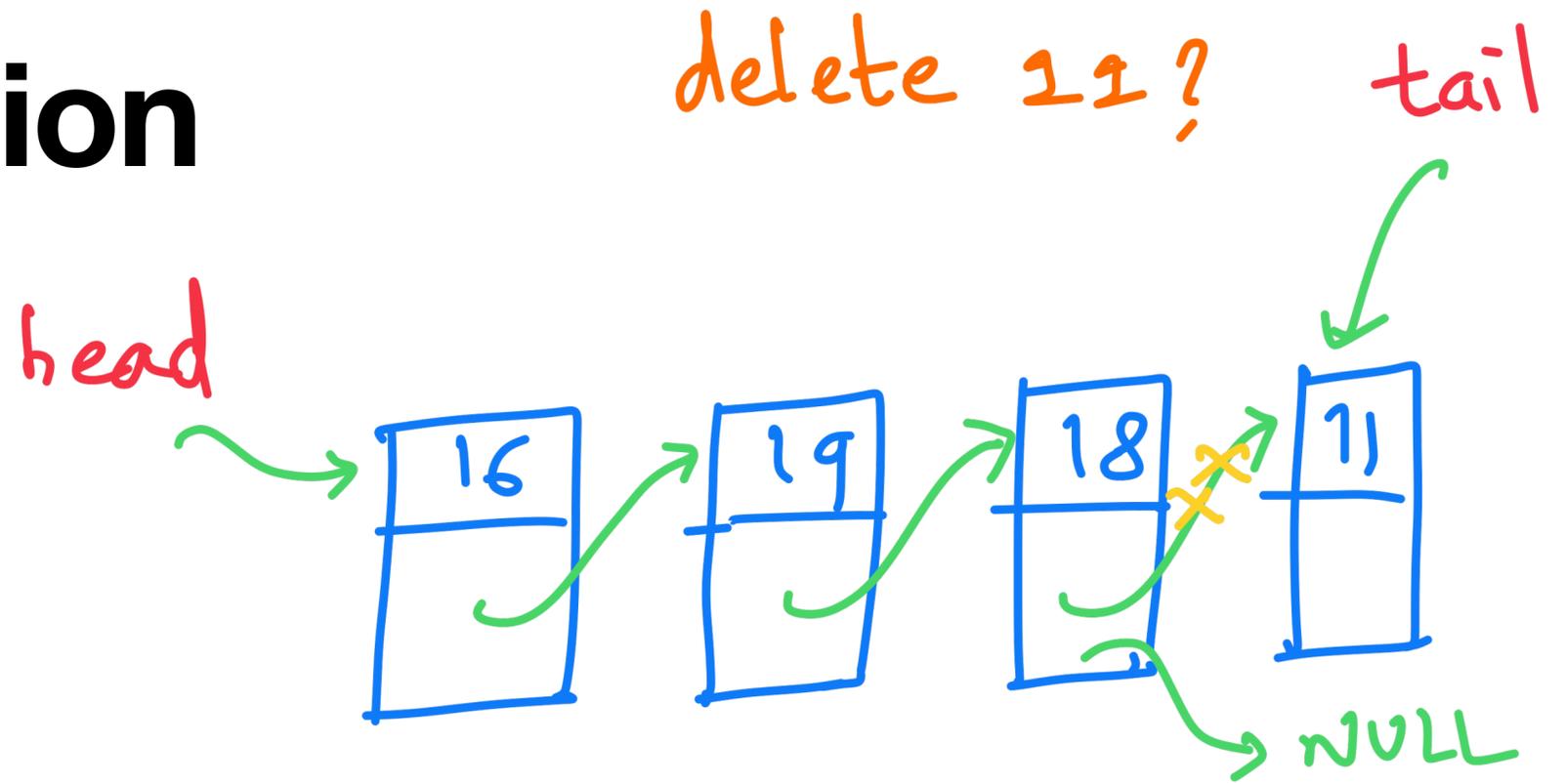


- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!

Linked Lists : Deletion

Delete last element

```
Node* n = head;  
while (n != NULL) {  
  ① If ..... break;  
  n = n->next;  
}
```

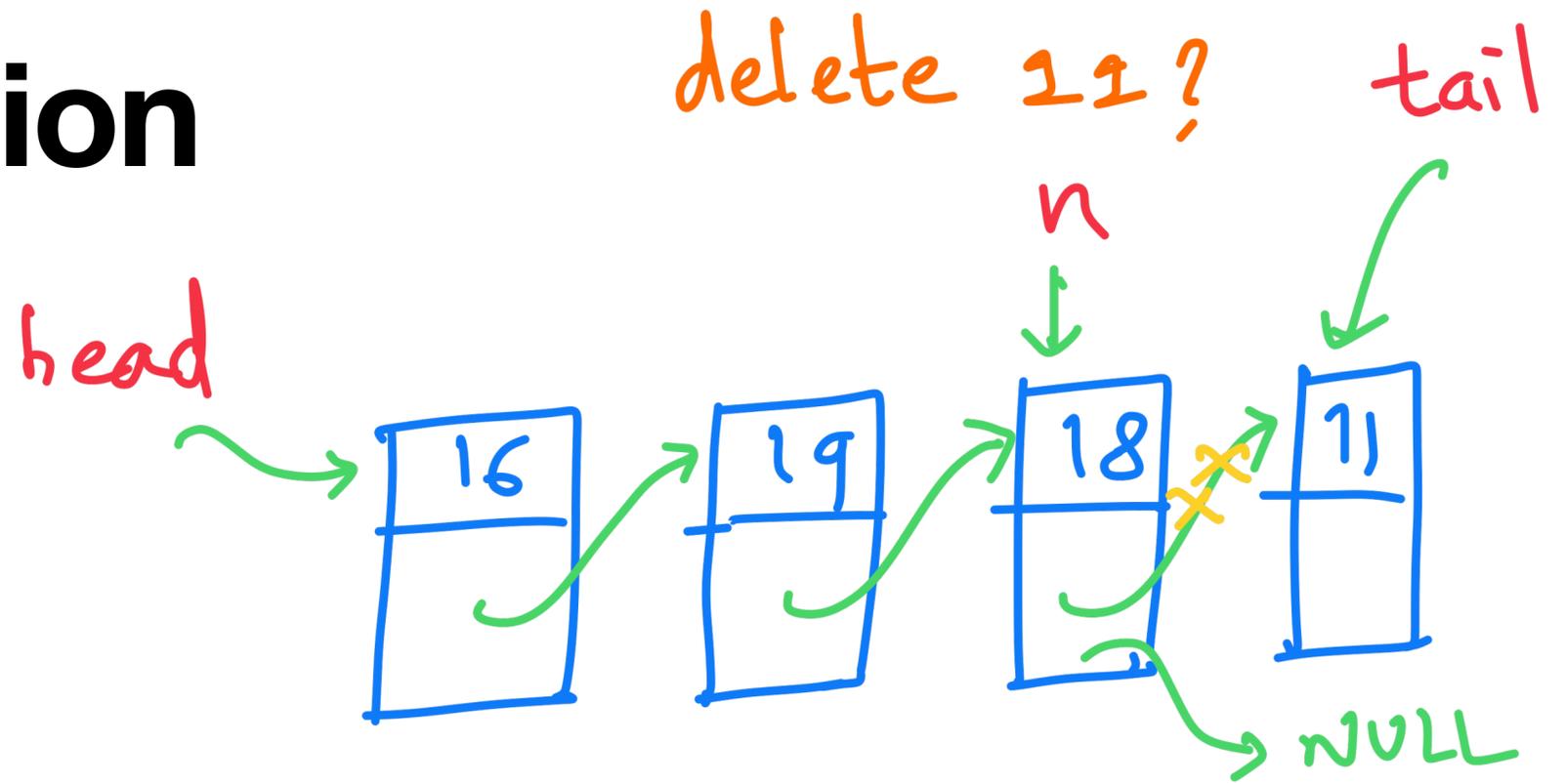


- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!

Linked Lists : Deletion

Delete last element

```
Node* n = head;  
while (n != NULL) {  
  ① If ..... break;  
  n = n->next;  
}
```

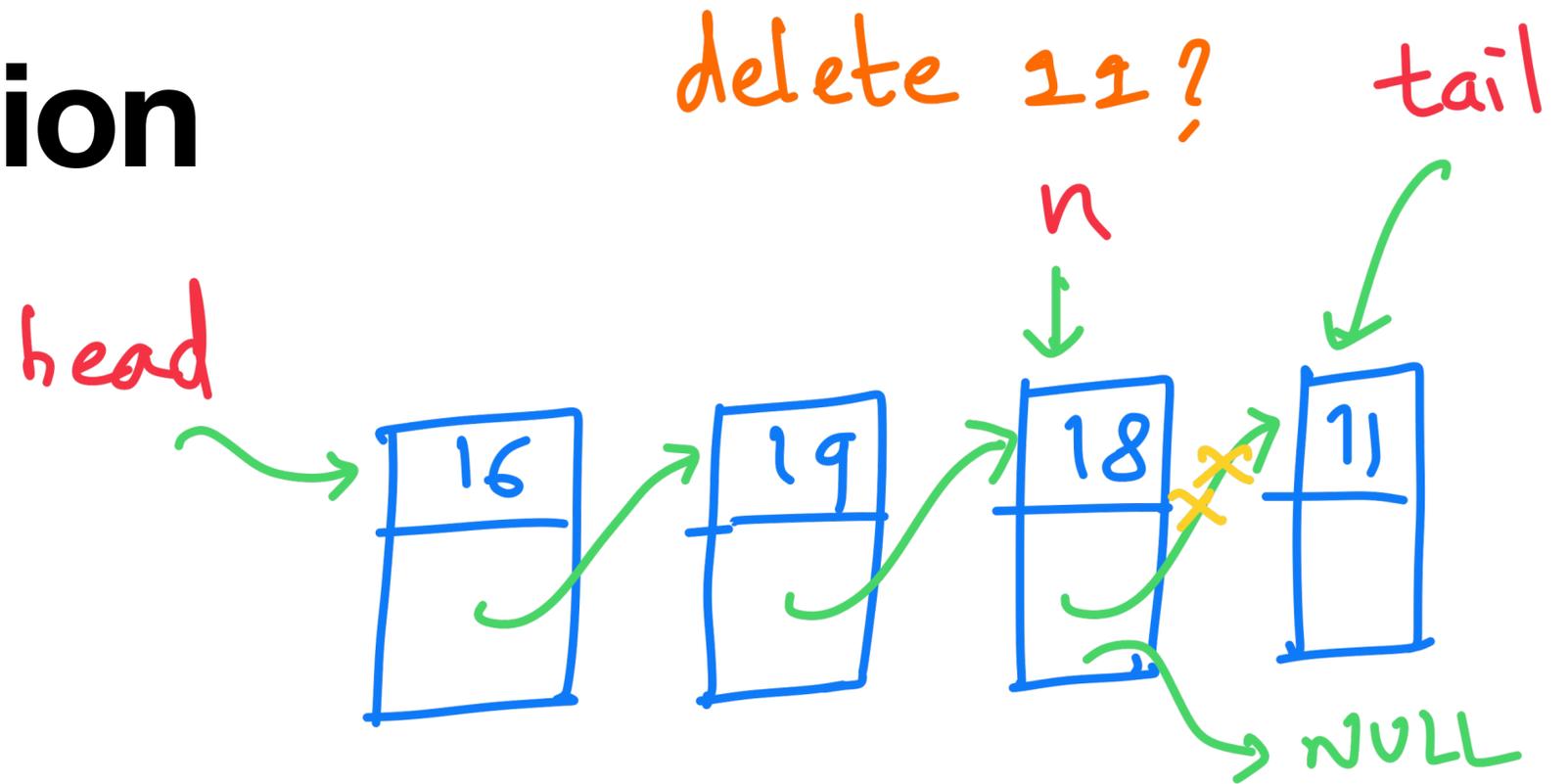


- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!

Linked Lists : Deletion

Delete last element

```
Node* n = head;  
while (n != NULL) {  
  ① if (n->next == tail) {  
    break;  
  }  
  n = n->next;  
}
```

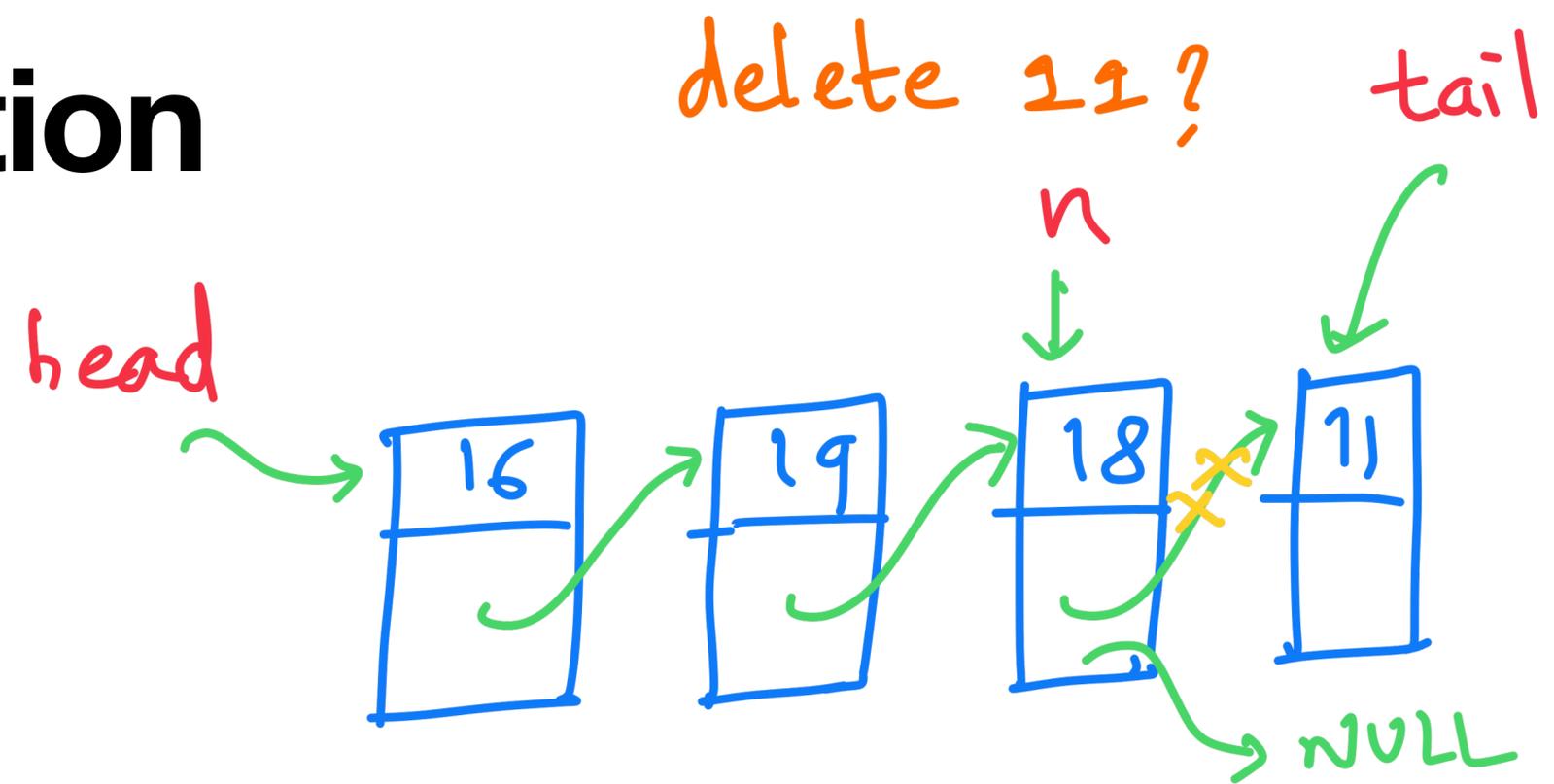


- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!

Linked Lists : Deletion

Delete last element

```
Node* n = head;
while (n != NULL) {
  ① if (n->next == tail) {
    break;
  }
  n = n->next;
}
② n->next = NULL;
```



- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!

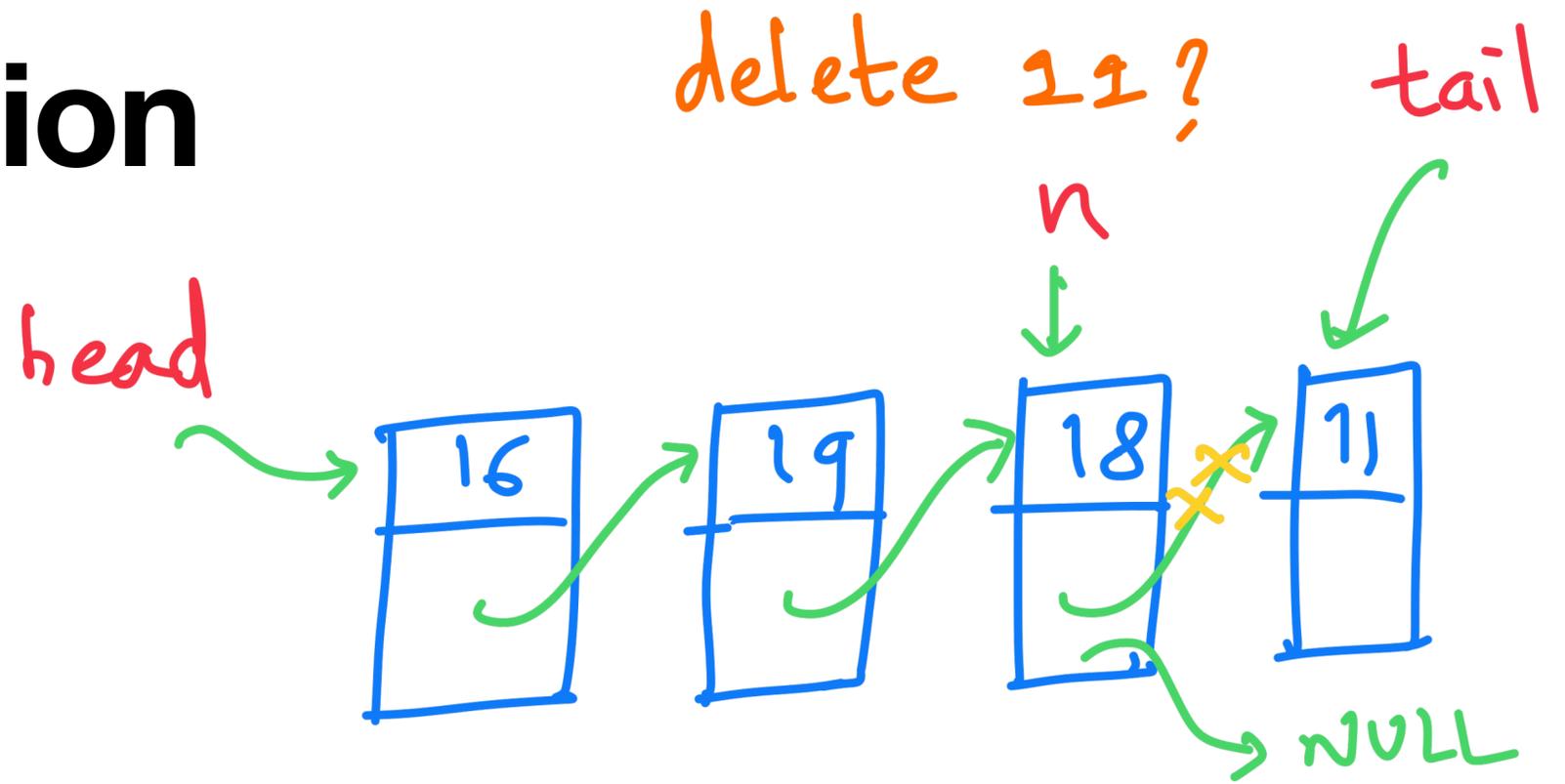
Linked Lists : Deletion

Delete last element

```
Node* n = head;  
while (n != NULL) {  
  ① if (n->next == tail) {  
    break;  
  }  
  n = n->next;  
}
```

```
② n->next = NULL;
```

```
③ -----
```



- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!
- ③ delete

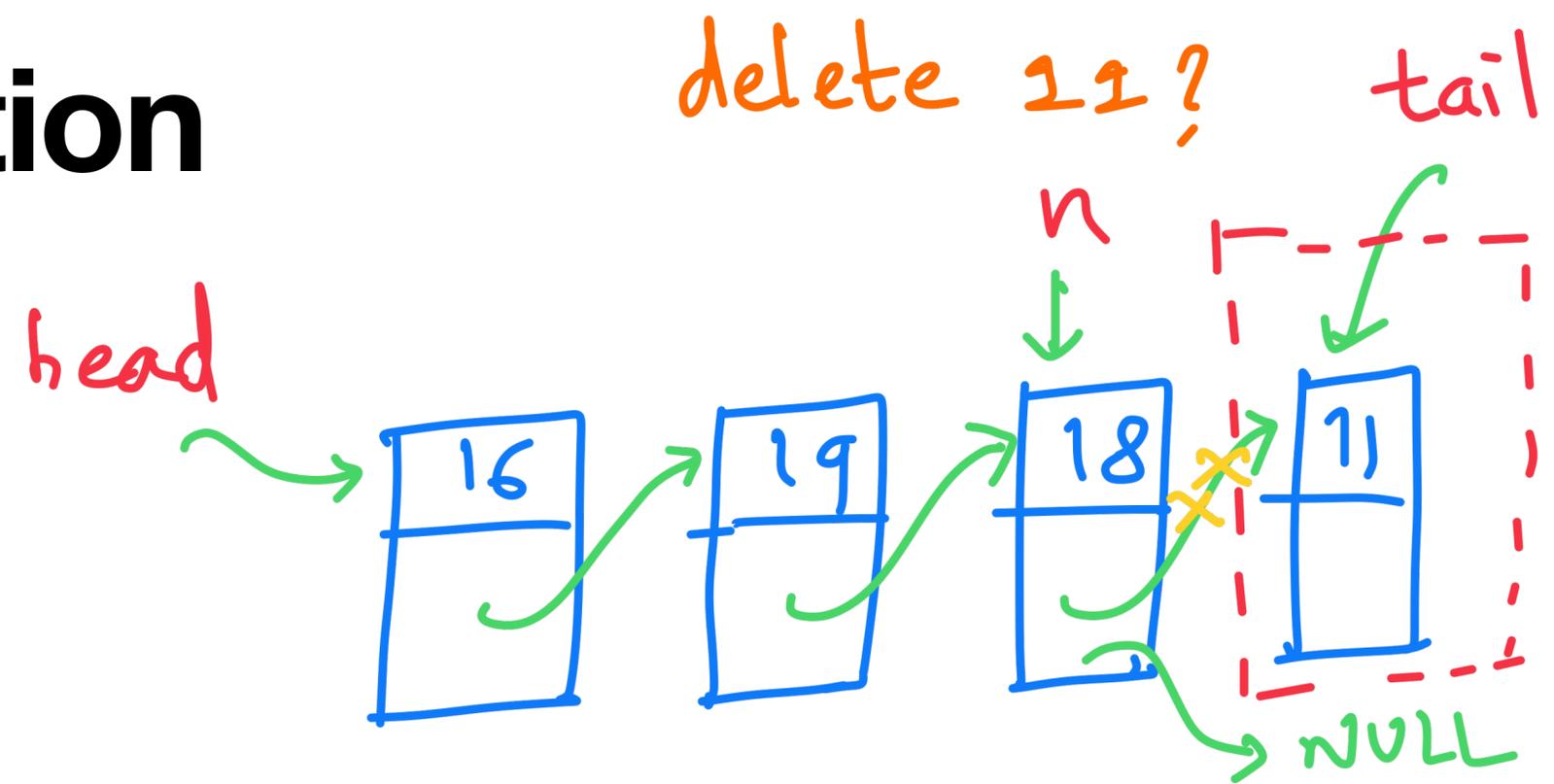
Linked Lists : Deletion

Delete last element

```
Node* n = head;
while (n != NULL) {
  ① if (n->next == tail) {
    break;
  }
  n = n->next;
}
```

```
② n->next = NULL;
```

```
③ delete tail;
```

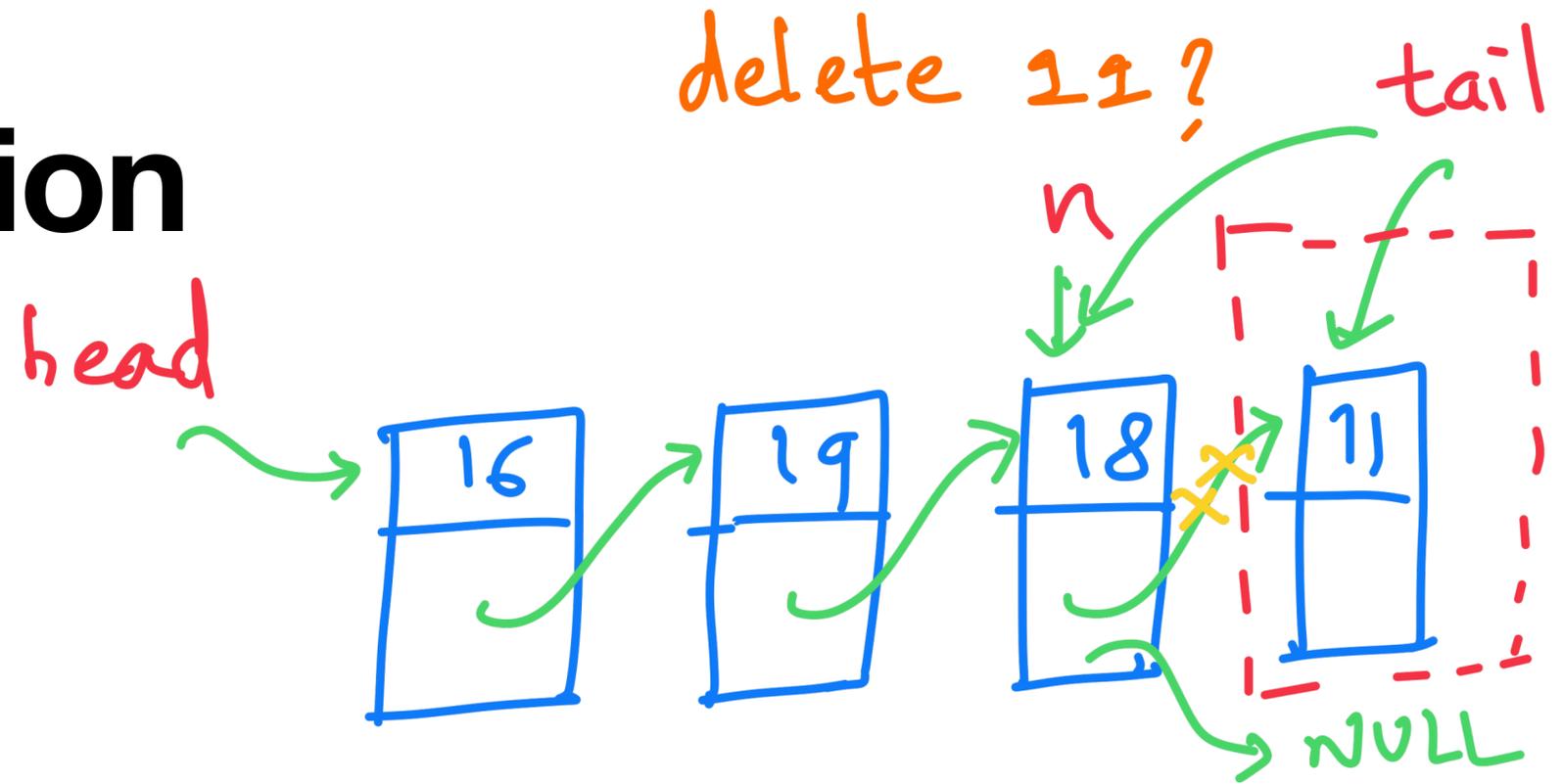


- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!
- ③ delete

Linked Lists : Deletion

Delete last element

```
Node* n = head;
while (n != NULL) {
  ① if (n->next == tail) {
    break;
  }
  n = n->next;
}
② n->next = NULL;
③ delete tail;
④ tail = n;
```



- ① Get the node to be deleted
it's the tail
- ② rewire
but we need node '18'
how to get it? - Traversal!
- ③ delete
- ④ reset tail.

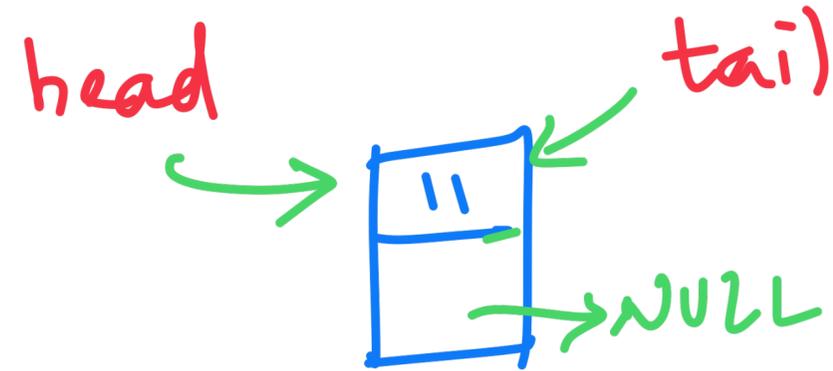
Linked Lists : Deletion

Delete last element

```
Node* n = head;  
while (n != NULL) {  
  ① if (n->next == tail) {  
    break;  
  }  
  n = n->next;  
}  
② n->next = NULL;  
③ delete tail;  
④ tail = n;
```

delete 11?

What if we have only 1 element?



} → Will this still work?

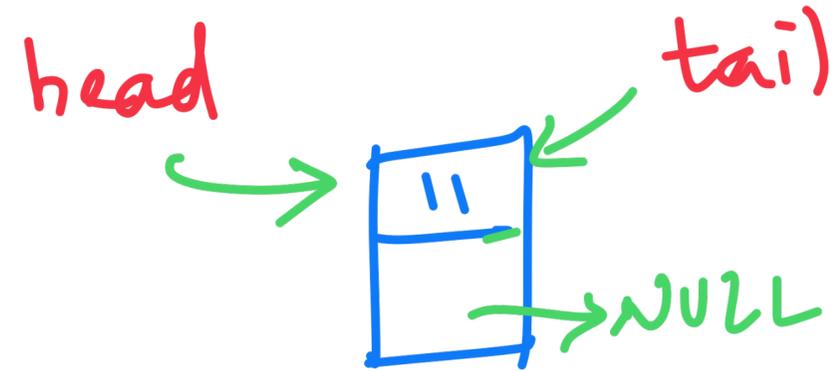
Linked Lists : Deletion

Delete last element

```
Node* n = head;
while (n != NULL) {
  ① if (n->next == tail) {
    break;
  }
  n = n->next;
}
② n->next = NULL;
③ delete tail;
④ tail = n;
```

delete 11?

What if we have only 1 element?



} → Will this still work? - NO,
"segmentation fault" at ②.

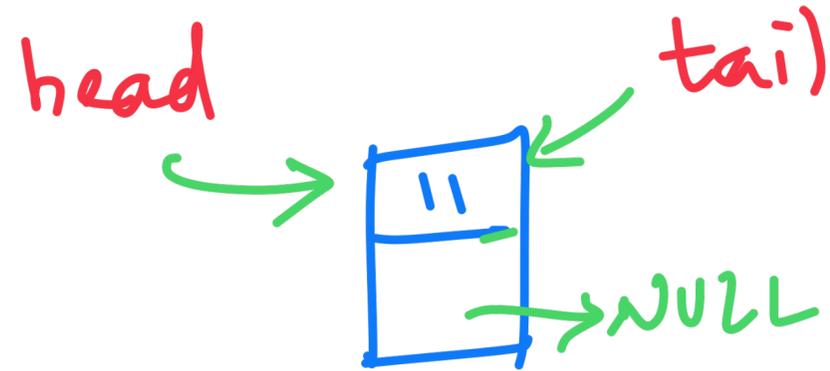
Linked Lists : Deletion

Delete last element

```
Node* n = head;
while (n != NULL) {
  ① if (n->next == tail) {
    break;
  }
  n = n->next;
}
② n->next = NULL;
③ delete tail;
④ tail = n;
```

delete 11?

What if we have only 1 element?



Will this still work? - NO,
"segmentation fault" at ②.
Need to check if head == tail
and also if head == NULL.

Linked Lists : Deletion

Delete last element

Full code →

→ empty list

```
if (head == NULL) return;
```

```
if (head == tail) { → one element case
```

```
    delete tail;
```

```
    tail = NULL;
```

```
    head = NULL;
```

```
    return; }
```

```
else {
```

```
    Node* n = head;
```

```
    while (n != NULL) {
```

```
        ① if (n->next == tail) {
```

```
            break;
```

```
        }
```

```
        n = n->next;
```

```
    }
```

```
    ② n->next = NULL;
```

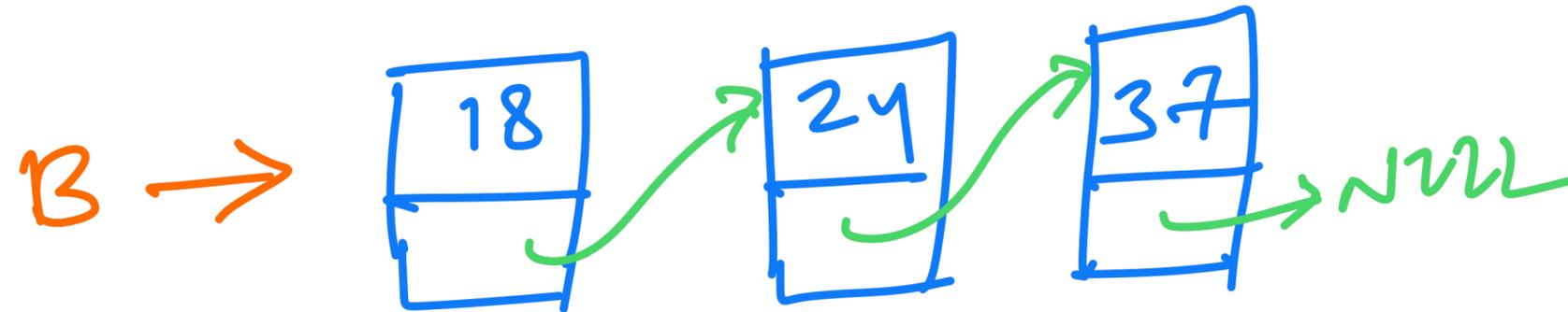
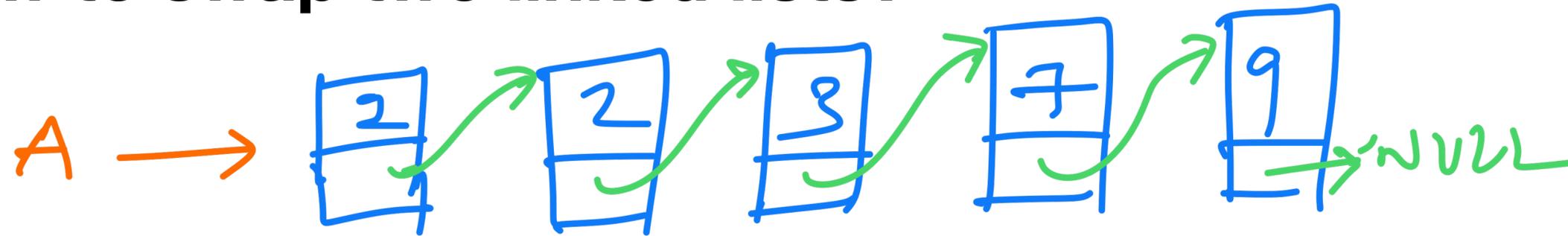
```
    ③ delete tail;
```

```
    ④ tail = n; }
```

→ multiple element case.

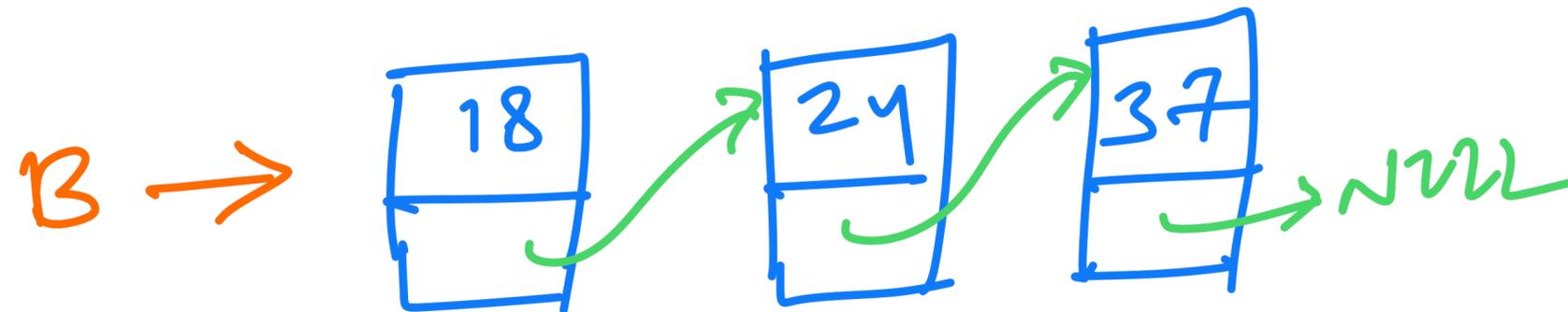
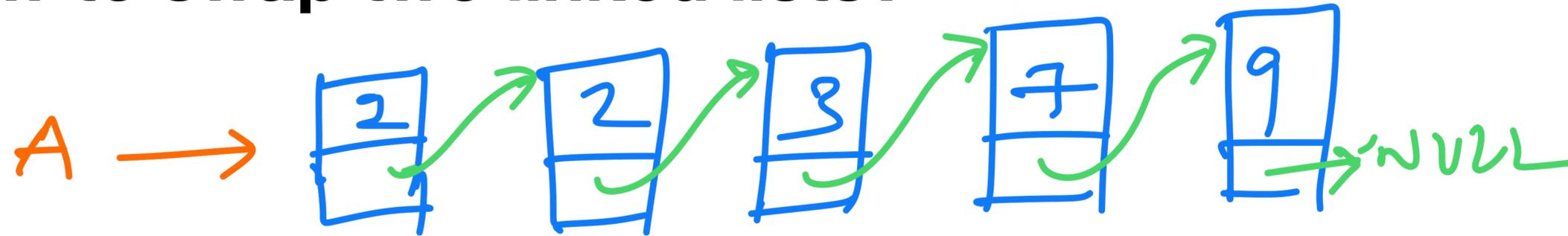
Linked Lists : Problems

How to swap two linked lists?



Linked Lists : Problems

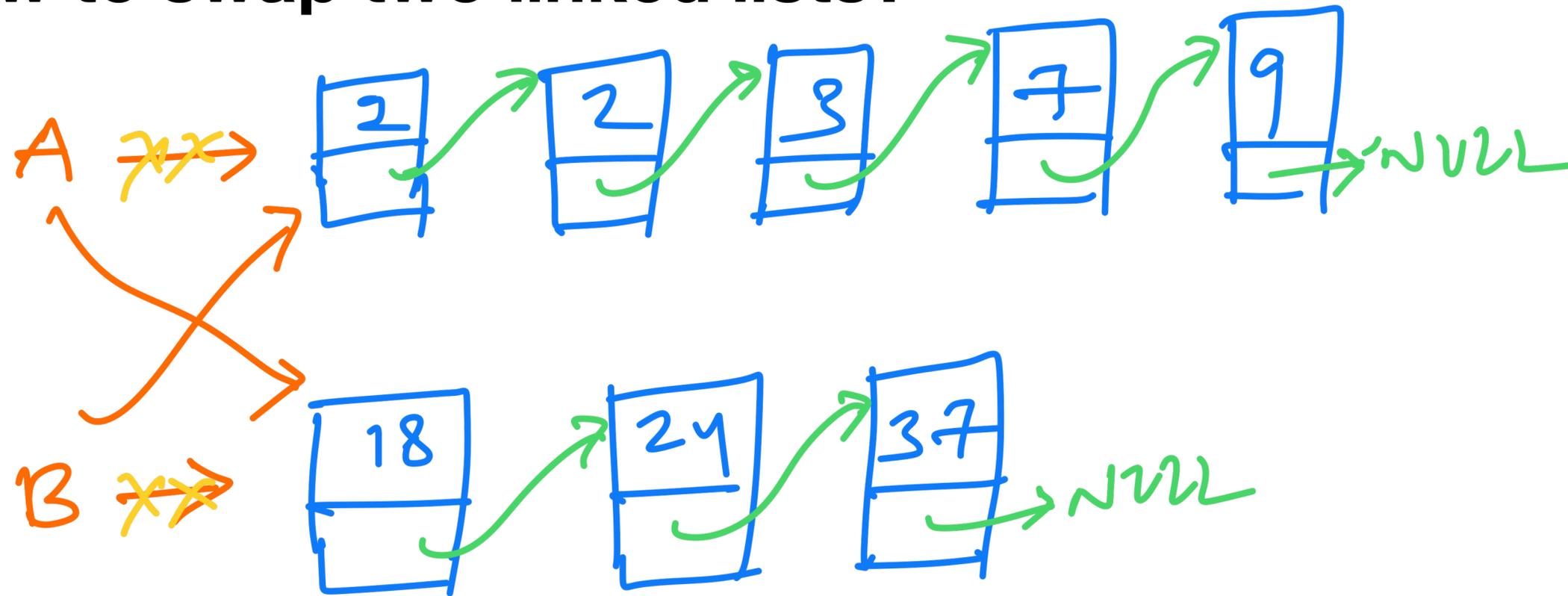
How to swap two linked lists?



Just swap the pointers !!

Linked Lists : Problems

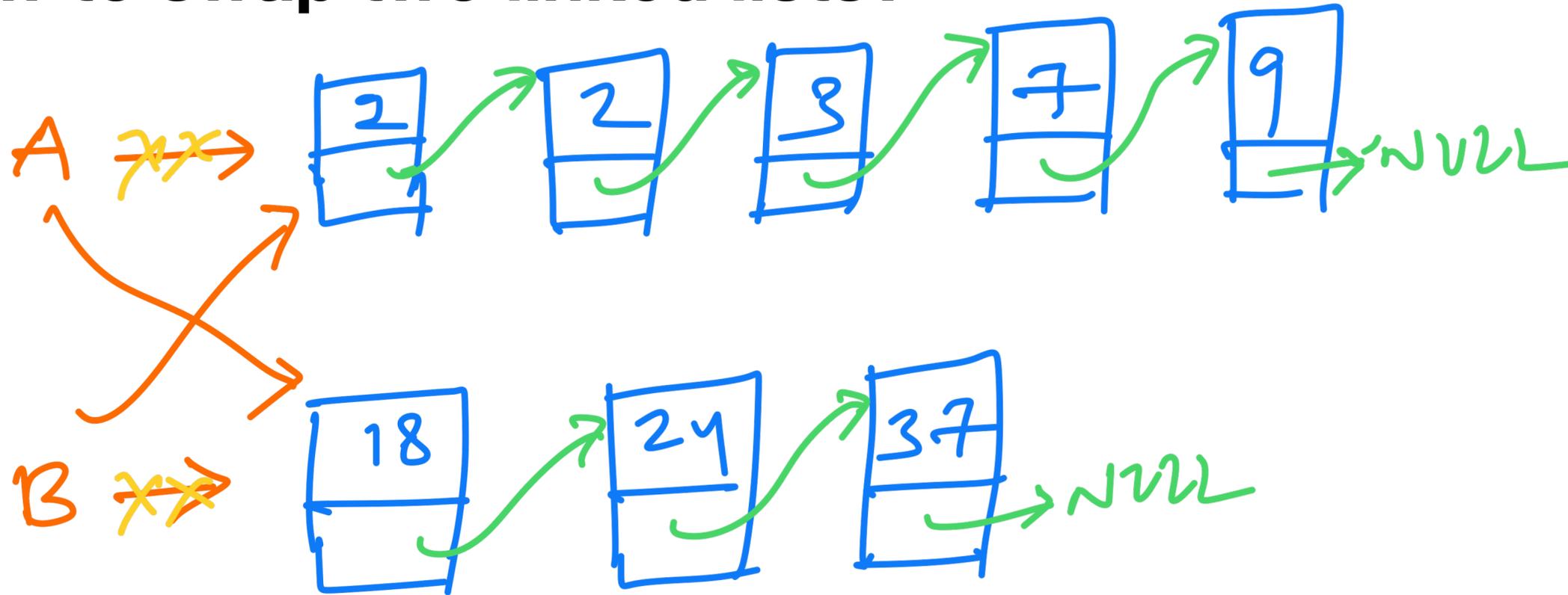
How to swap two linked lists?



Just swap the pointers !!

Linked Lists : Problems

How to swap two linked lists?

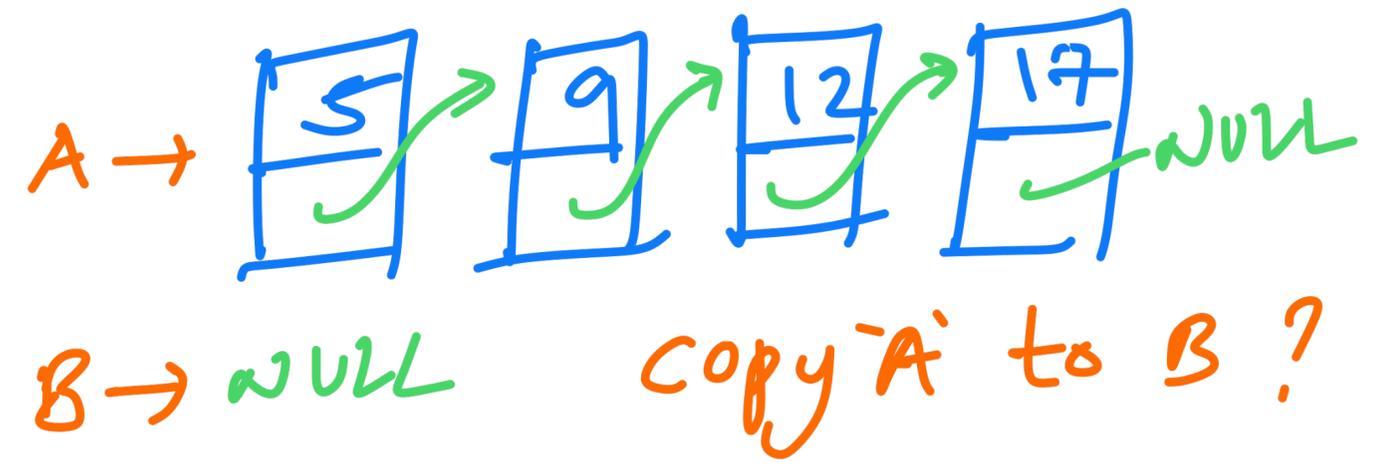


```
Node* tmp = A;  
A = B;  
B = tmp;
```

Just swap the pointers !!

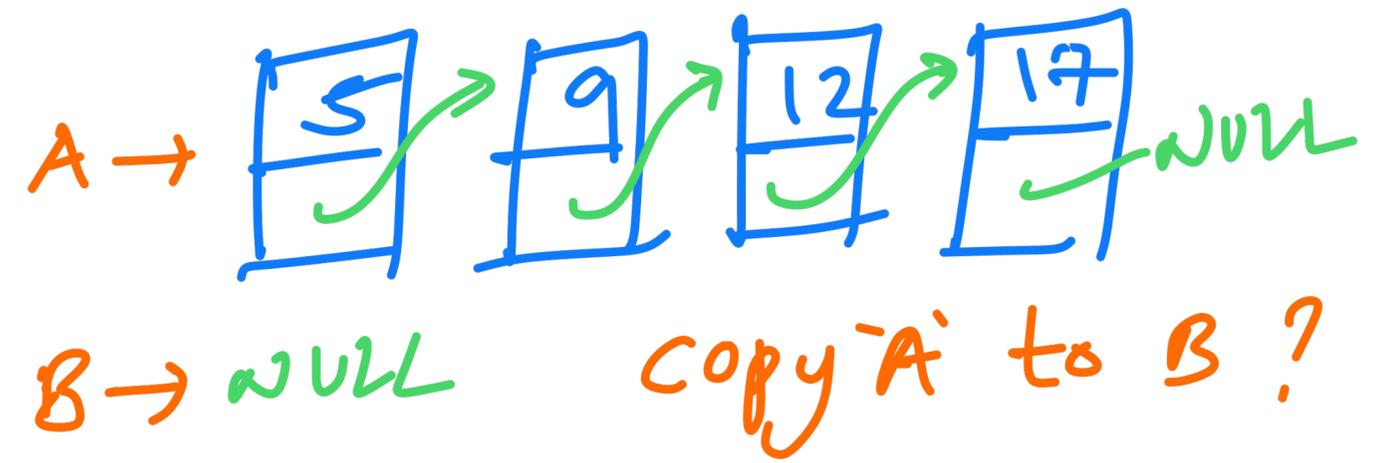
Linked Lists : Problems

Copy a linked list



Linked Lists : Problems

Copy a linked list



① Go through each element of A

Linked Lists : Problems

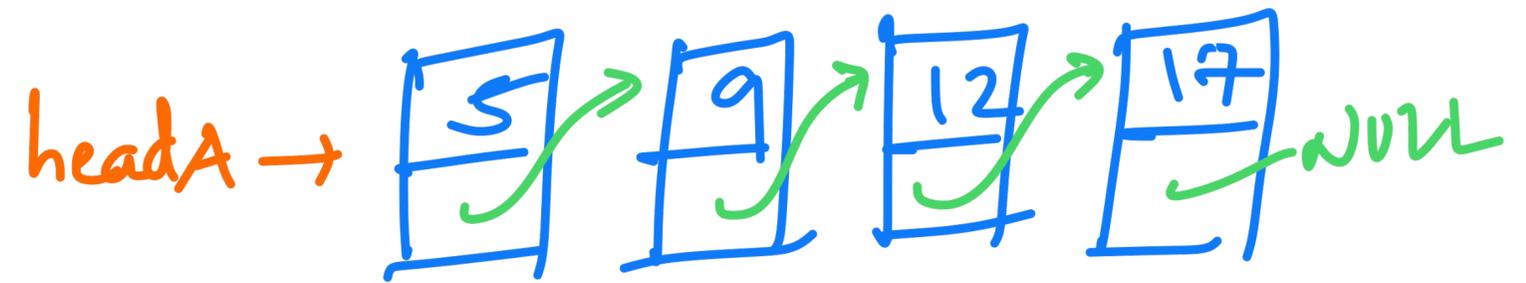
Copy a linked list

```
Node* tmpA = headA;
```

```
while (tmpA != NULL) {
```

```
    tmpA = tmpA->next;
```

```
}
```



headB → NULL copy 'A' to B ?

① Go through each element of A
→ Traversal

Linked Lists : Problems

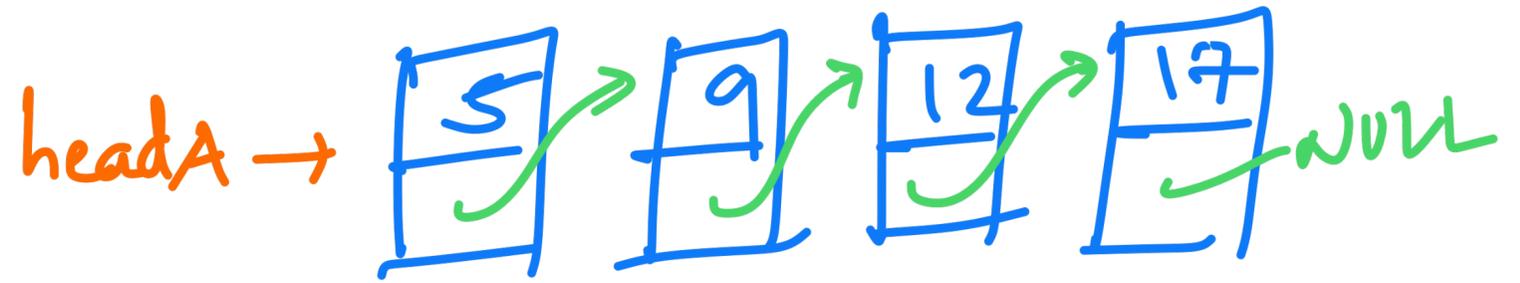
Copy a linked list

```
Node* tmpA = headA;
```

```
while(tmpA != NULL){
```

```
    tmpA = tmpA->next;
```

```
}
```



headB → NULL copy 'A' to B ?

- ① Go through each element of A
→ Traversal
- ② create copy of the element

Linked Lists : Problems

Copy a linked list

```
Node* tmpA = headA;
```

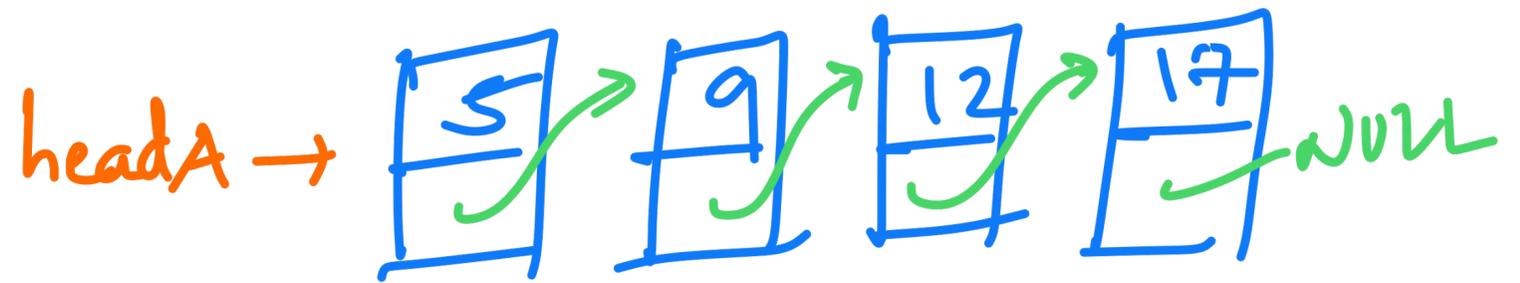
```
while (tmpA != NULL) {
```

```
    Node* cpy = new Node;
```

```
    cpy->n-val = tmpA->n-val;
```

```
    tmpA = tmpA->next;
```

```
}
```



headB → NULL copy 'A' to B ?

- ① Go through each element of A
→ Traversal
- ② create copy of the element

Linked Lists : Problems

Copy a linked list

```
Node* tmpA = headA;
```

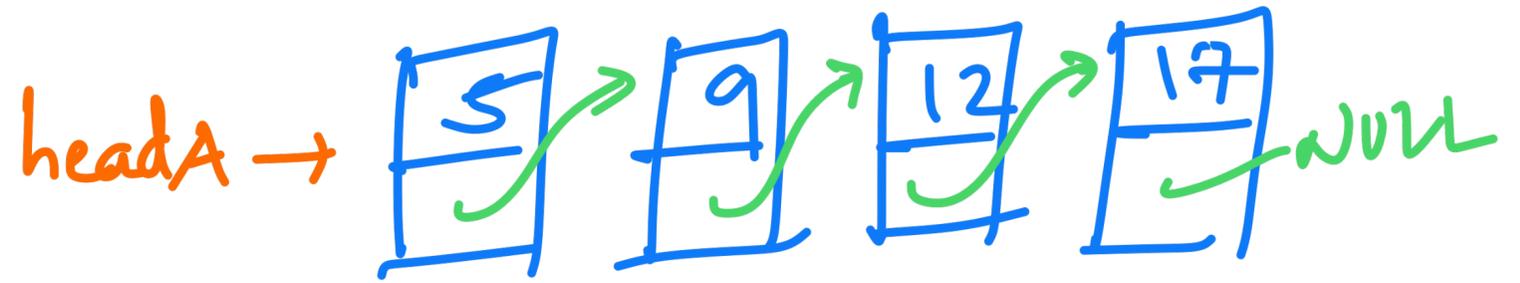
```
while (tmpA != NULL) {
```

```
    Node* cpy = new Node;
```

```
    cpy->n-val = tmpA->n-val;
```

```
    tmpA = tmpA->next;
```

```
}
```



headB → NULL copy 'A' to B ?

- ① Go through each element of A
→ Traversal
- ② create copy of the element
- ③ Add copy to B list.

Linked Lists : Problems

Copy a linked list

```
Node* tmpA = headA;
```

```
while(tmpA != NULL){
```

```
    Node* cpy = new Node;
```

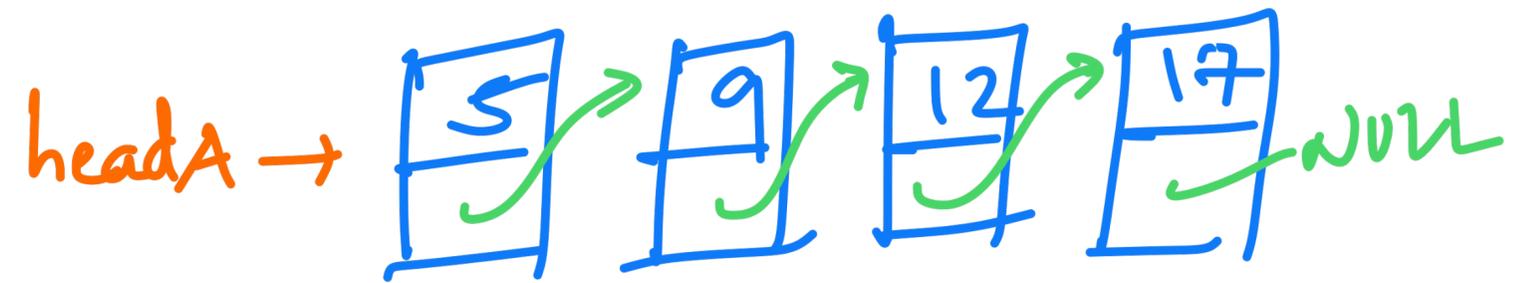
```
    cpy->n-val = tmpA->n-val;
```

```
    if(headB == NULL){
```

```
        headB = cpy;
```

```
        tailB = cpy; }
```

⋮

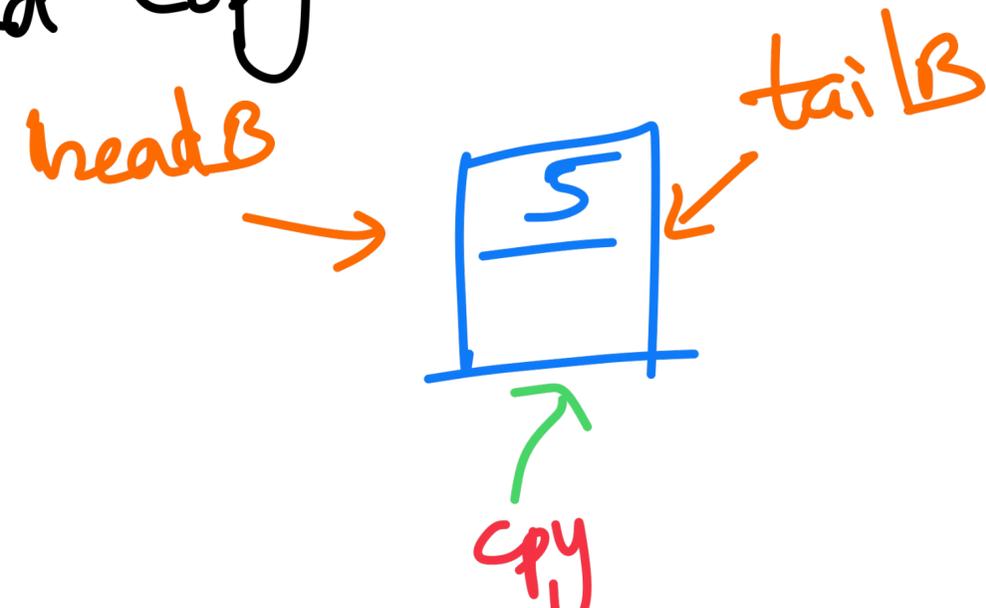


headB → NULL copy 'A' to B ?

① Go through each element of A
→ Traversal

② create copy of the element

③ Add copy to B list.



Linked Lists : Problems

Copy a linked list

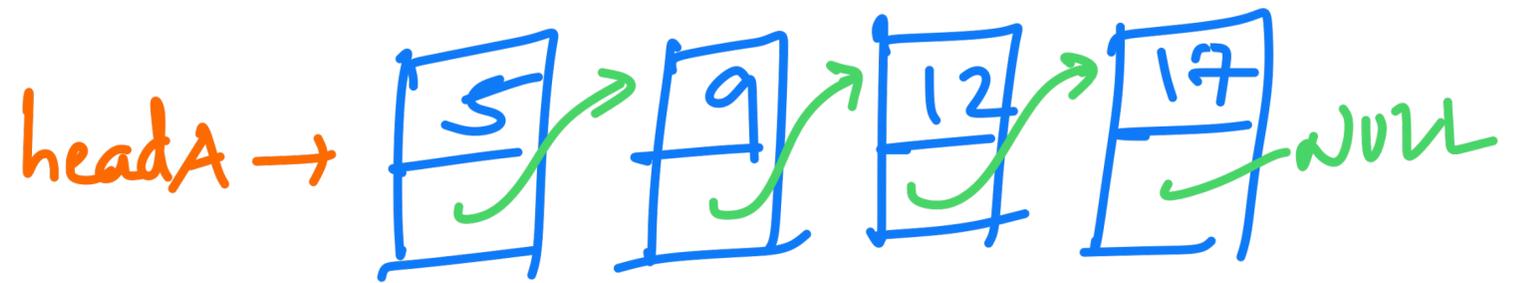
```
Node* tmpA = headA;
```

```
while(tmpA != NULL){  
    Node* cpy = new Node;  
    cpy->val = tmpA->val;
```

```
    if(headB == NULL){  
        headB = cpy;  
        tailB = cpy;
```

```
    } else {  
        tailB->next = cpy;  
        tailB = cpy;
```

```
    }  
    tmpA = tmpA->next;  
}
```

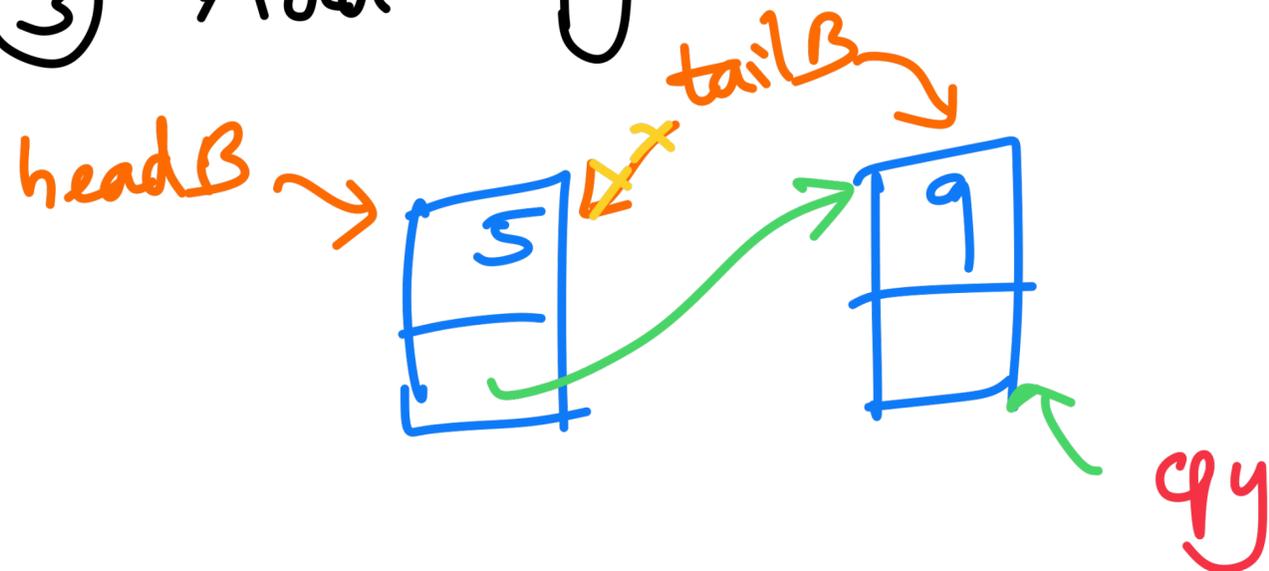


headB → NULL copy 'A' to B ?

① Go through each element of A
→ Traversal

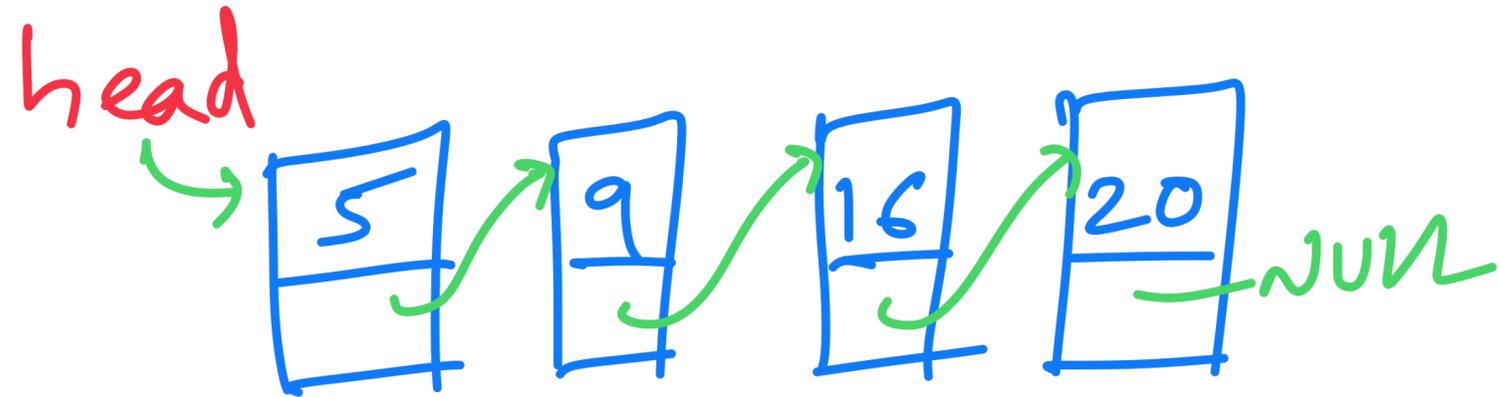
② create copy of the element

③ Add copy to B list.



Linked Lists : Problems

Delete a linked list

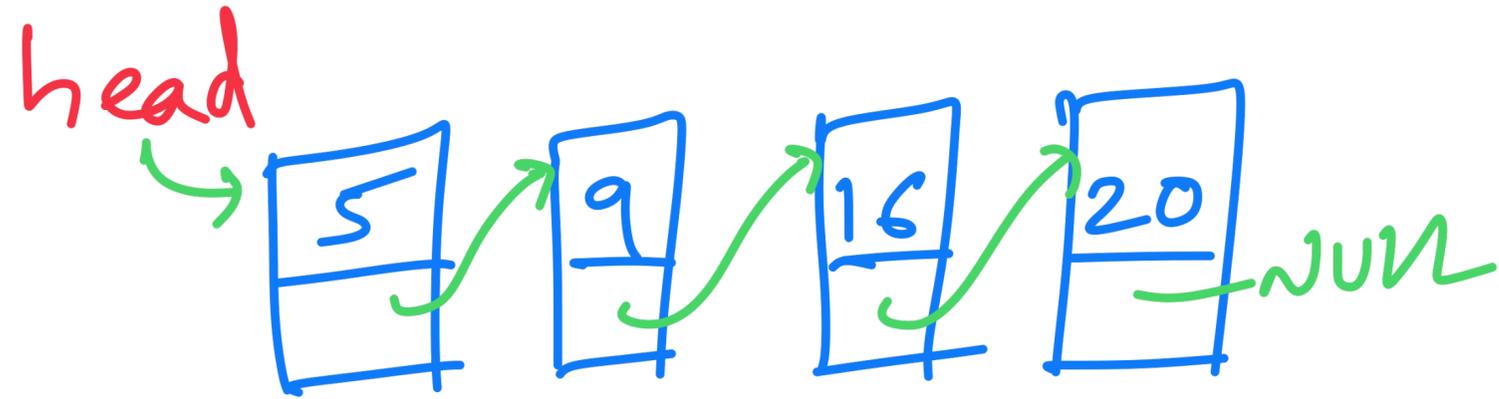


delete all nodes .

Any ideas?

Linked Lists : Problems

Delete a linked list



delete all nodes.

Any ideas?

→ Traversal !!

```
Node* tmp = head;
```

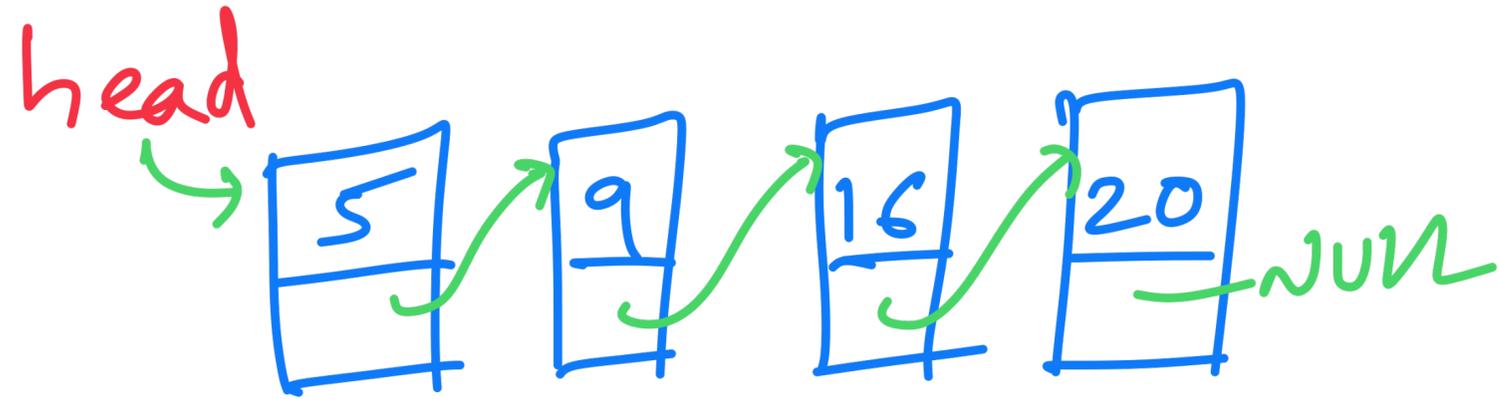
```
while (tmp != NULL) {
```

```
    tmp = tmp->next;
```

```
}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

Any ideas?

→ Traversal !!

```
Node* tmp = head;
```

```
while (tmp != NULL) {
```

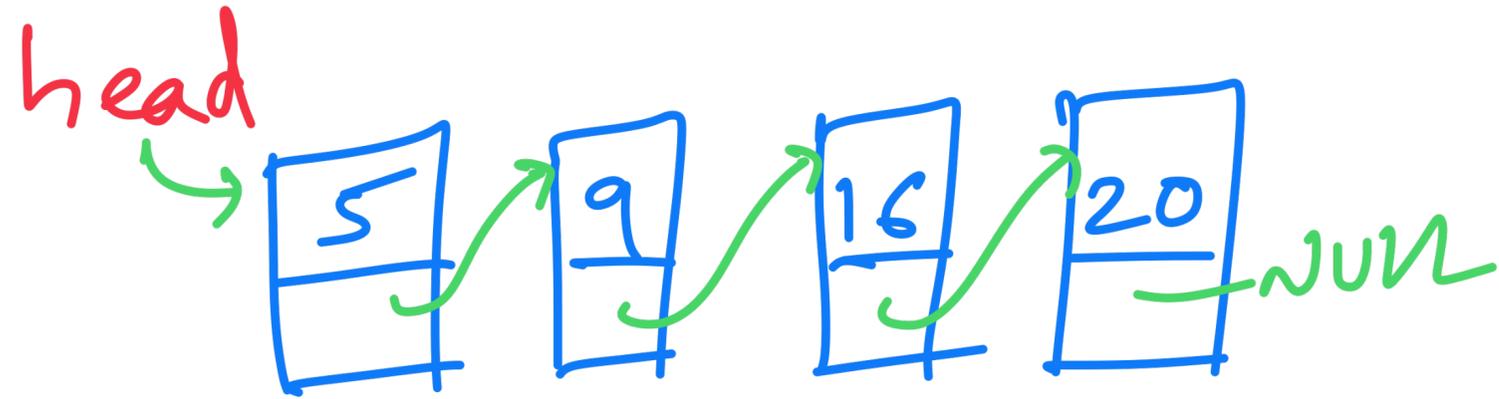
① -----

```
tmp = tmp->next;
```

```
}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

Any ideas?

→ Traversal !!

```
Node* tmp = head;
```

```
while (tmp != NULL) {
```

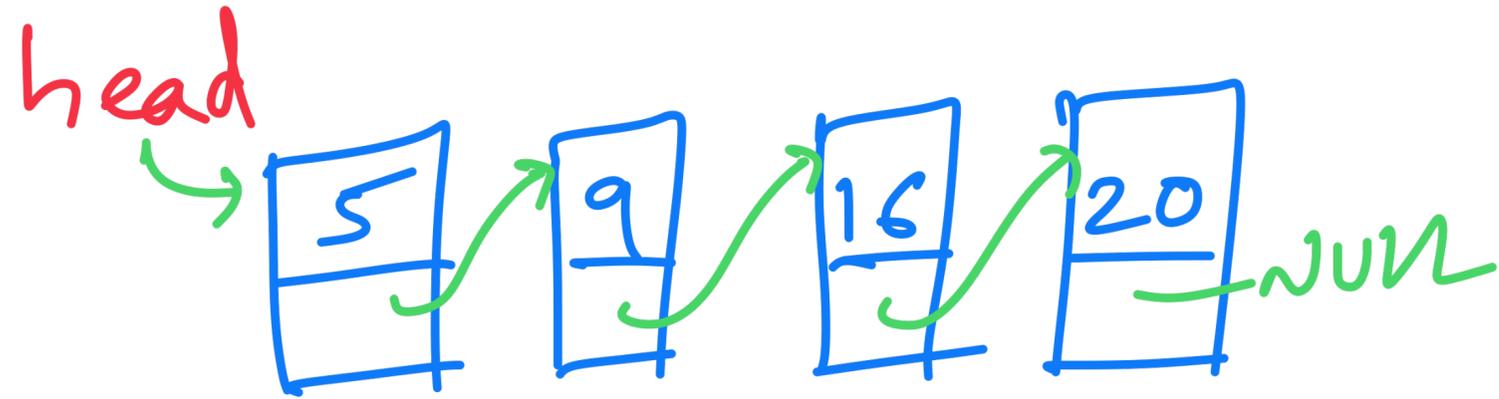
① delete tmp ??

```
tmp = tmp->next;
```

```
}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

Any ideas?

→ Traversal !!

```
Node* tmp = head;
```

```
while (tmp != NULL) {
```

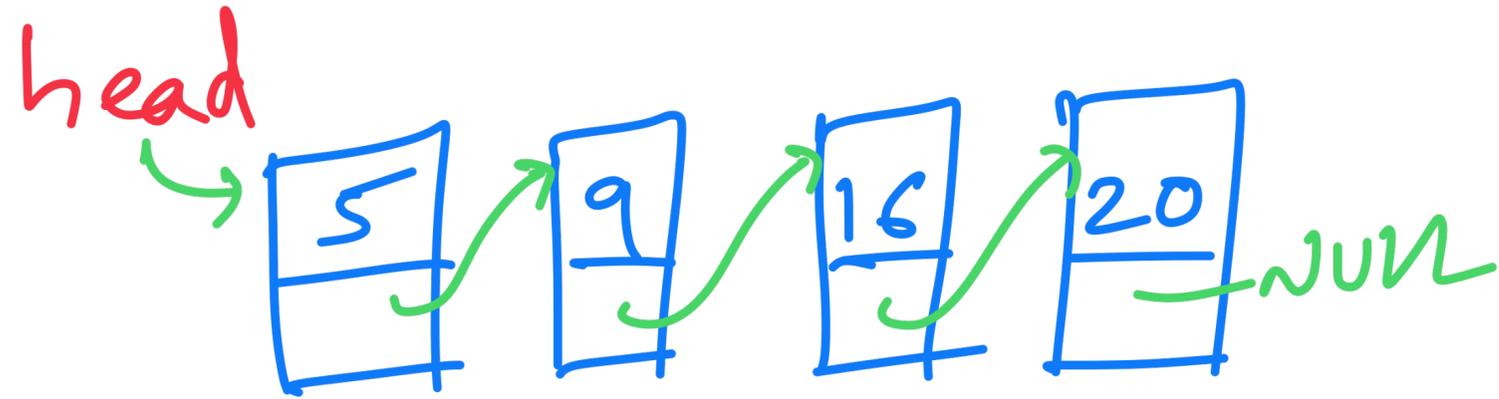
① delete tmp ?? → "No"

```
tmp = tmp->next;
```

```
}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

Any ideas?

→ Traversal !!

```
Node* tmp = head;
```

```
while (tmp != NULL) {
```

```
    ① -----
```

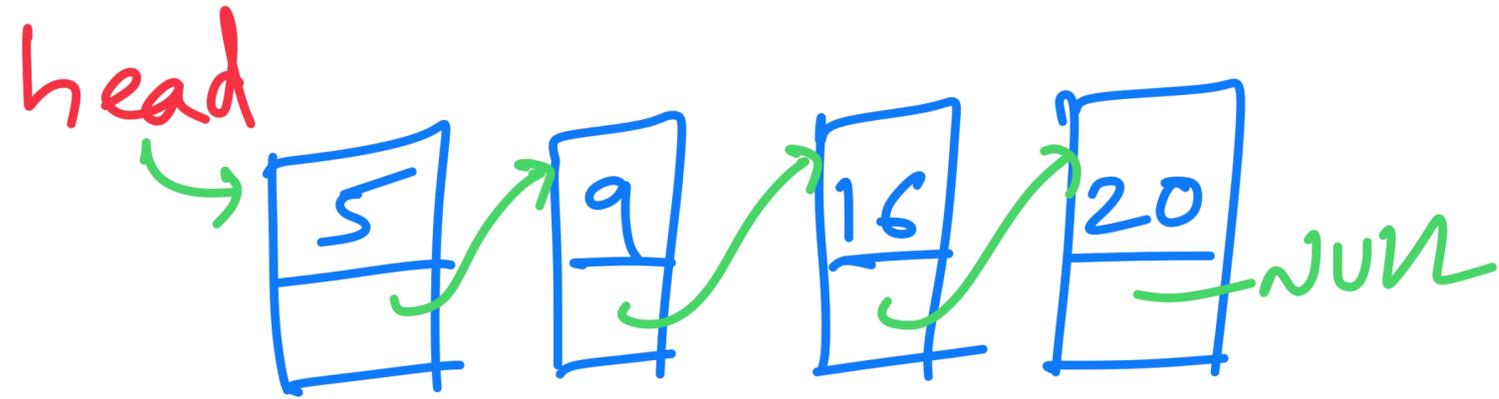
```
    tmp = tmp->next;
```

```
    ② delete tmp ??
```

```
}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

Any ideas?

→ Traversal !!

```
Node* tmp = head;
```

```
while (tmp != NULL) {
```

① -----

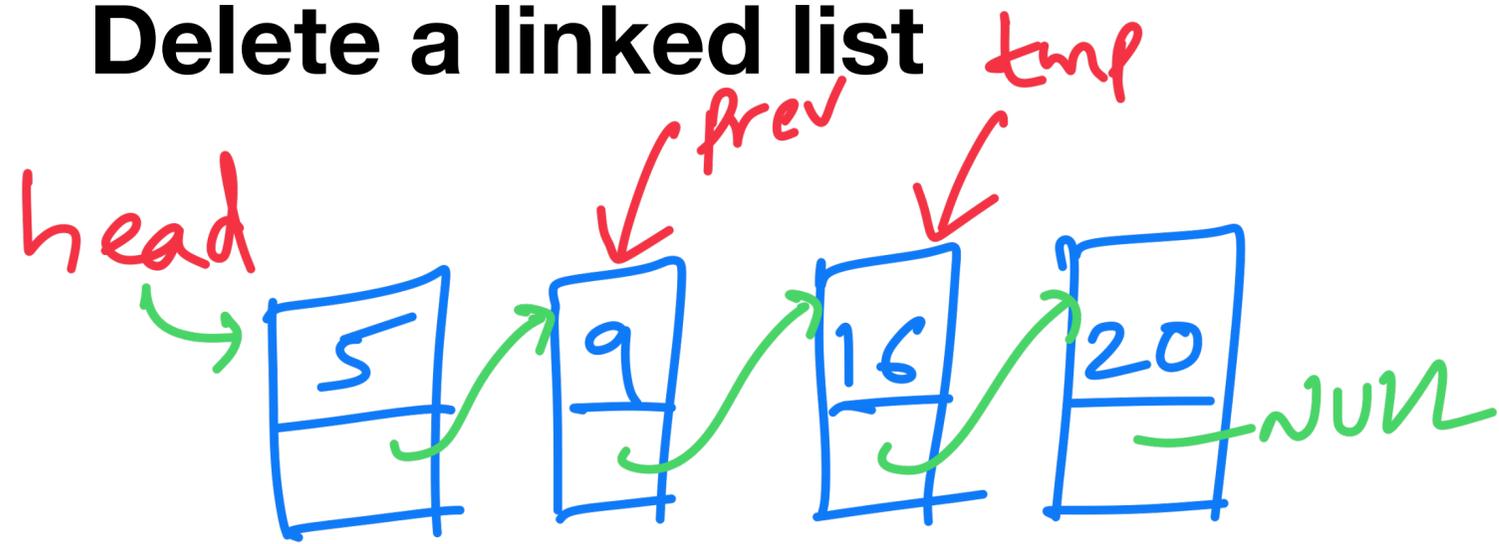
```
tmp = tmp->next;
```

② delete tmp ?? → "No"

```
}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

Any ideas?

→ Traversal !!

```
Node* tmp = head;
```

```
while (tmp != NULL) {
```

```
    ① Node* prev = tmp;
```

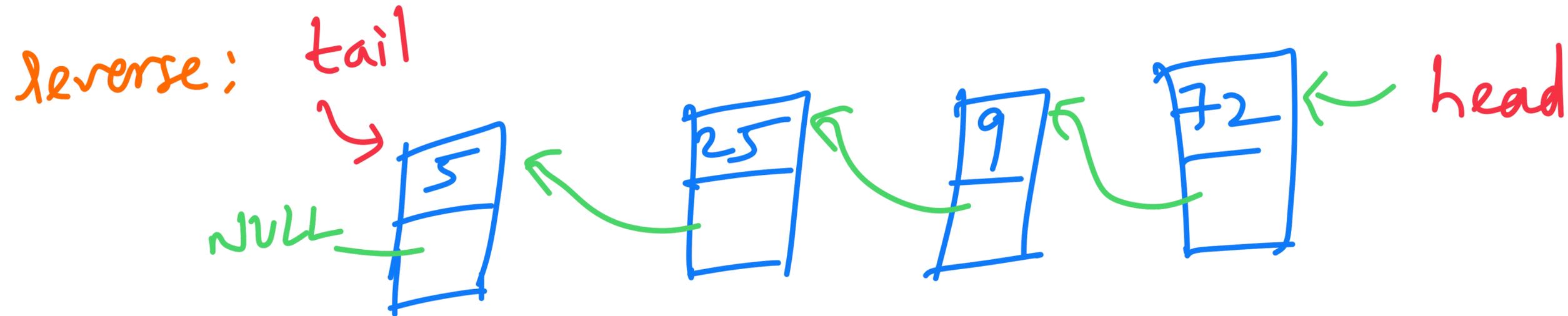
```
    tmp = tmp->next;
```

```
    ② delete prev;
```

```
}
```

Linked Lists : Problems

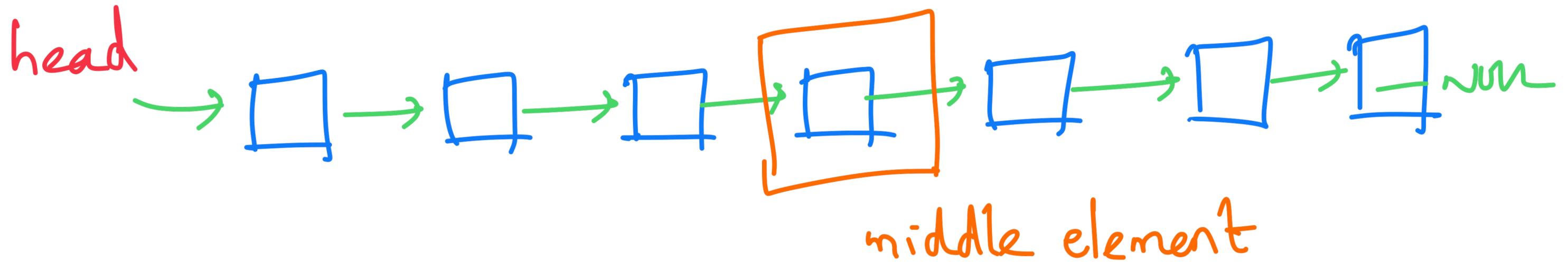
Reverse a linked list



→ Try It !!

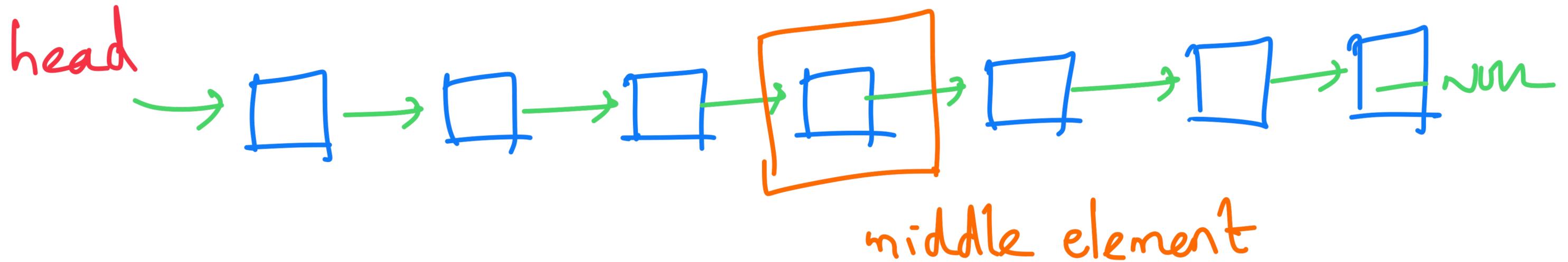
Linked Lists : Problems

Find approximate middle element in one pass



Linked Lists : Problems

Find approximate middle element in one pass



Hint ; Use two pointers

Linked Lists : Guidelines

Edge cases and best practices

- Draw Pictures
- When using `p->`, make sure `p` is initialized and not NULL
- Do not write `p++` or `p--` on pointers
- Always check for special cases : middle, front, back, empty list, one-element list