CS 32 Week 3
Discussion 1I

Srinath

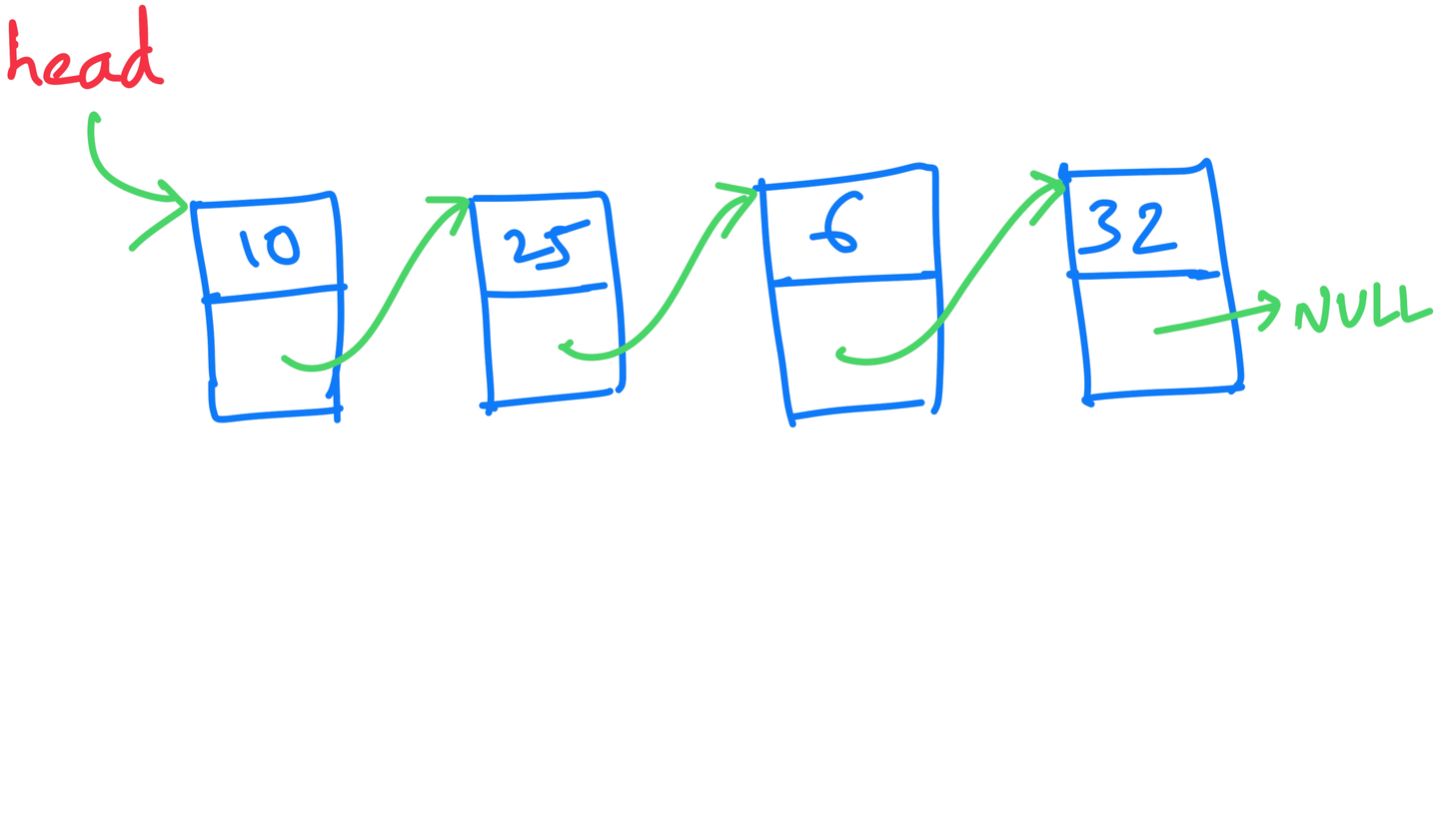
Topics

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

Linked Lists : Definition

Singly Linked List, Doubly Linked List

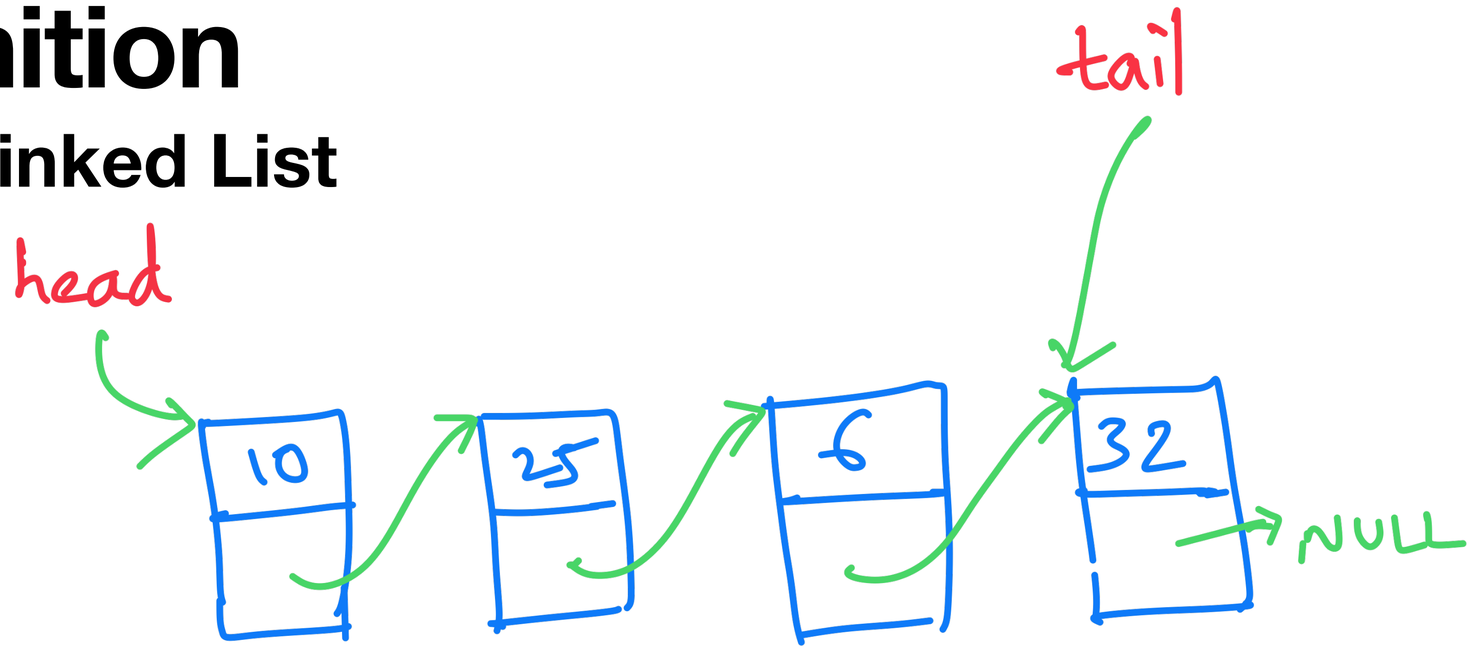
```
class Node{
        int m_val;
        Node* next;
        };

Node* head;
```



Linked Lists : Definition

Singly Linked List, Doubly Linked List

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

Node* head;
Node* tail;
```

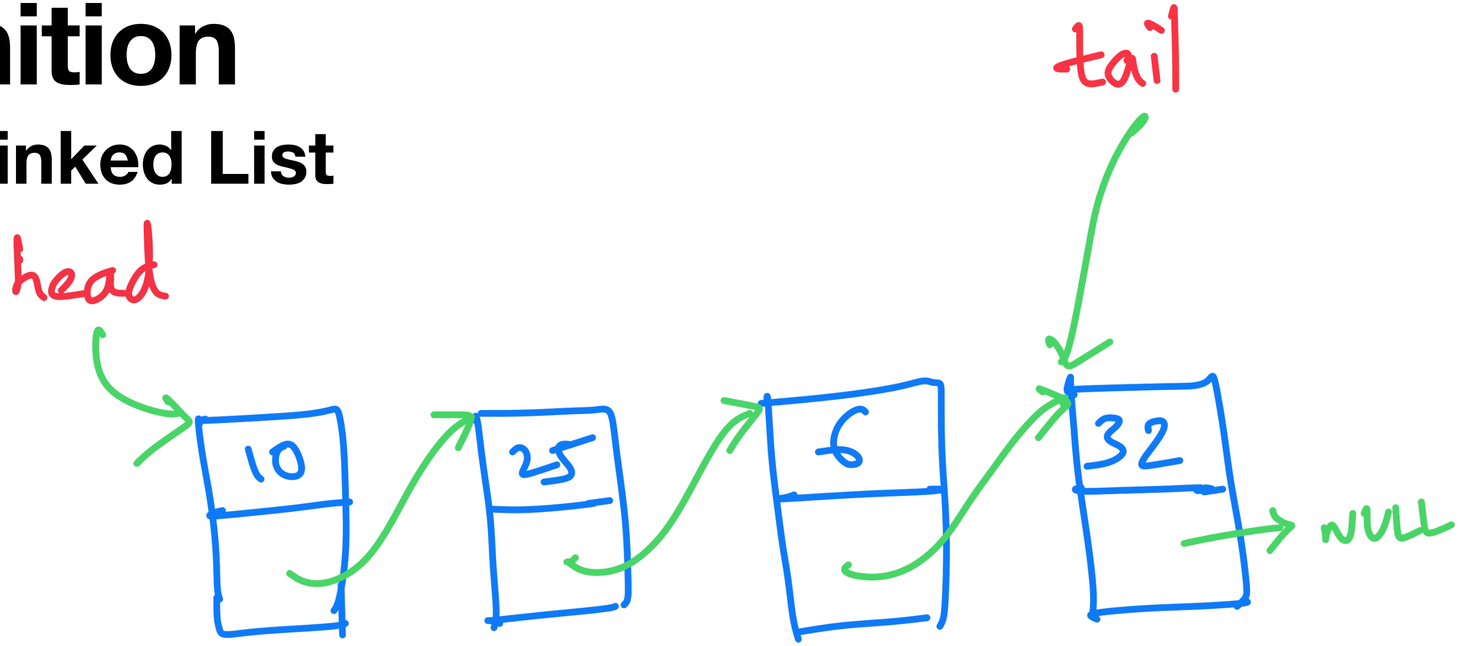


We can also use a tail pointer

Linked Lists : Definition

Singly Linked List, Doubly Linked List

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

Node* head;
Node* tail;
```



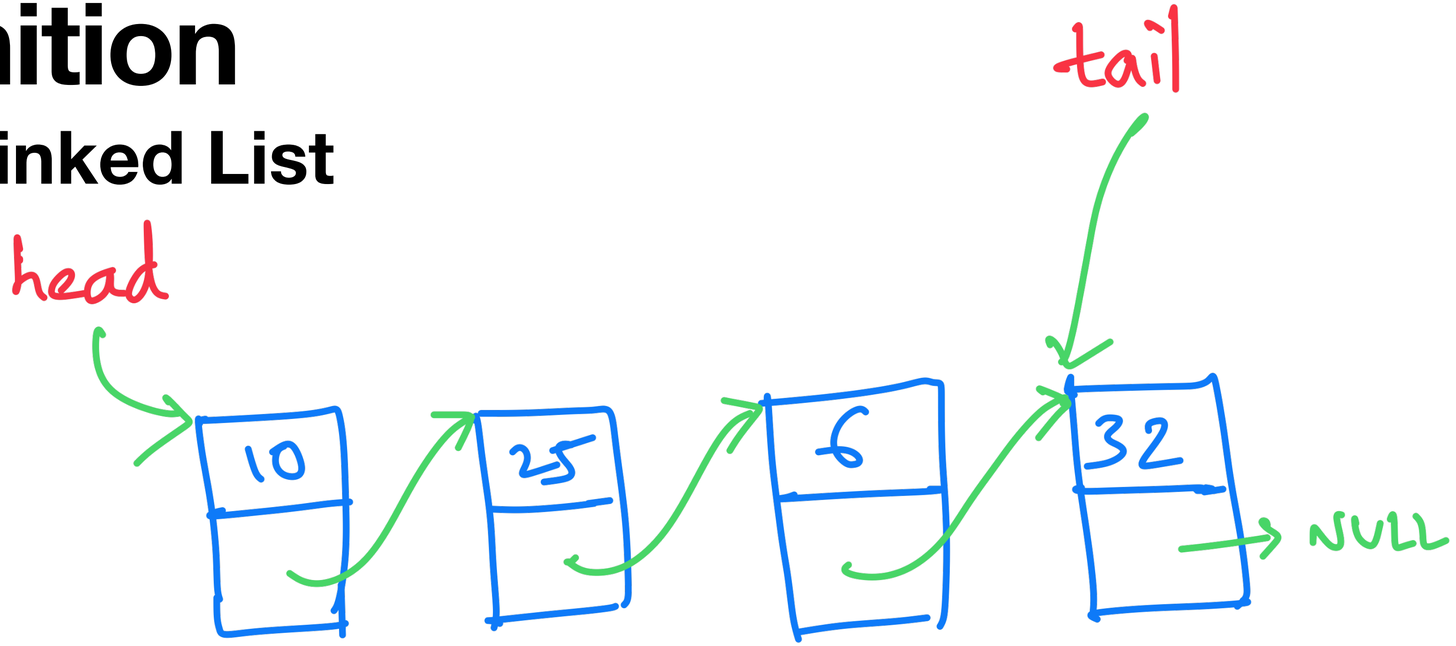
We can also use a tail pointer

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

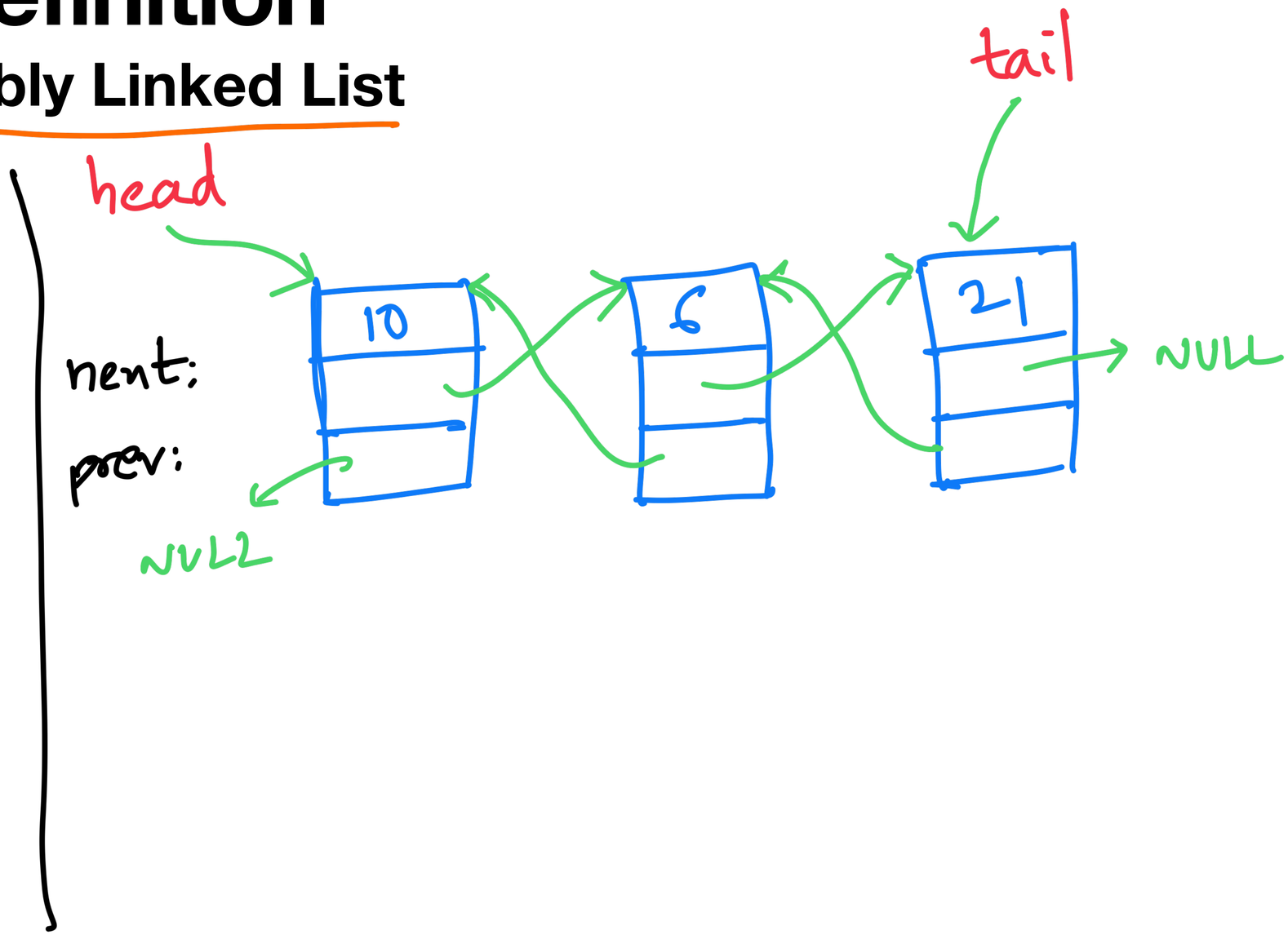
Empty list?

```
head = NULL;
tail = NULL;
```

Linked Lists : Definition

Singly Linked List, Doubly Linked List

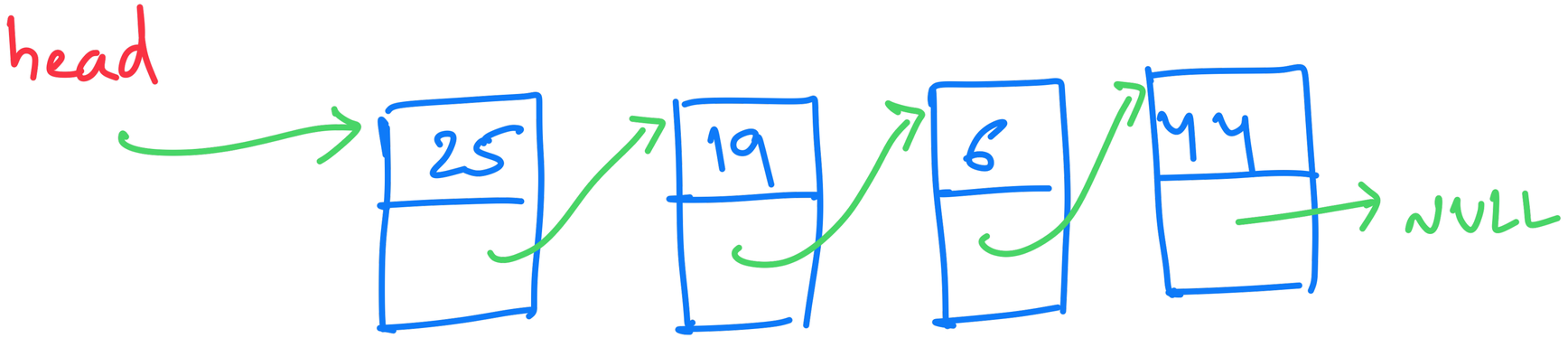
```
class Node{
    int m_val;
    Node* nent;
    Node* prev;
};

Node* head;
Node* tail;
```



Linked Lists : Traversal

Algorithm

Printing all elements

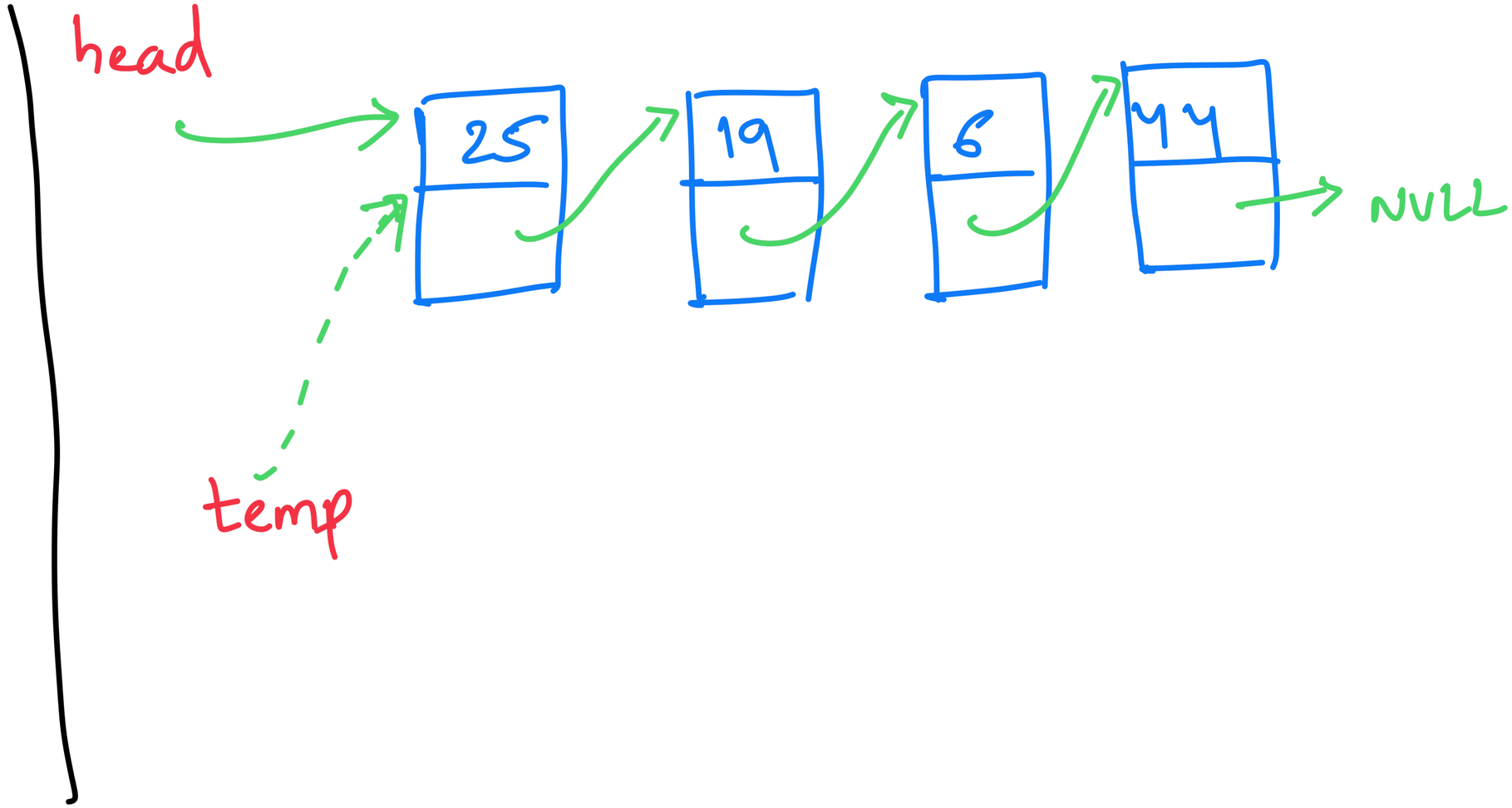


Linked Lists : Traversal

Printing all elements

Algorithm

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

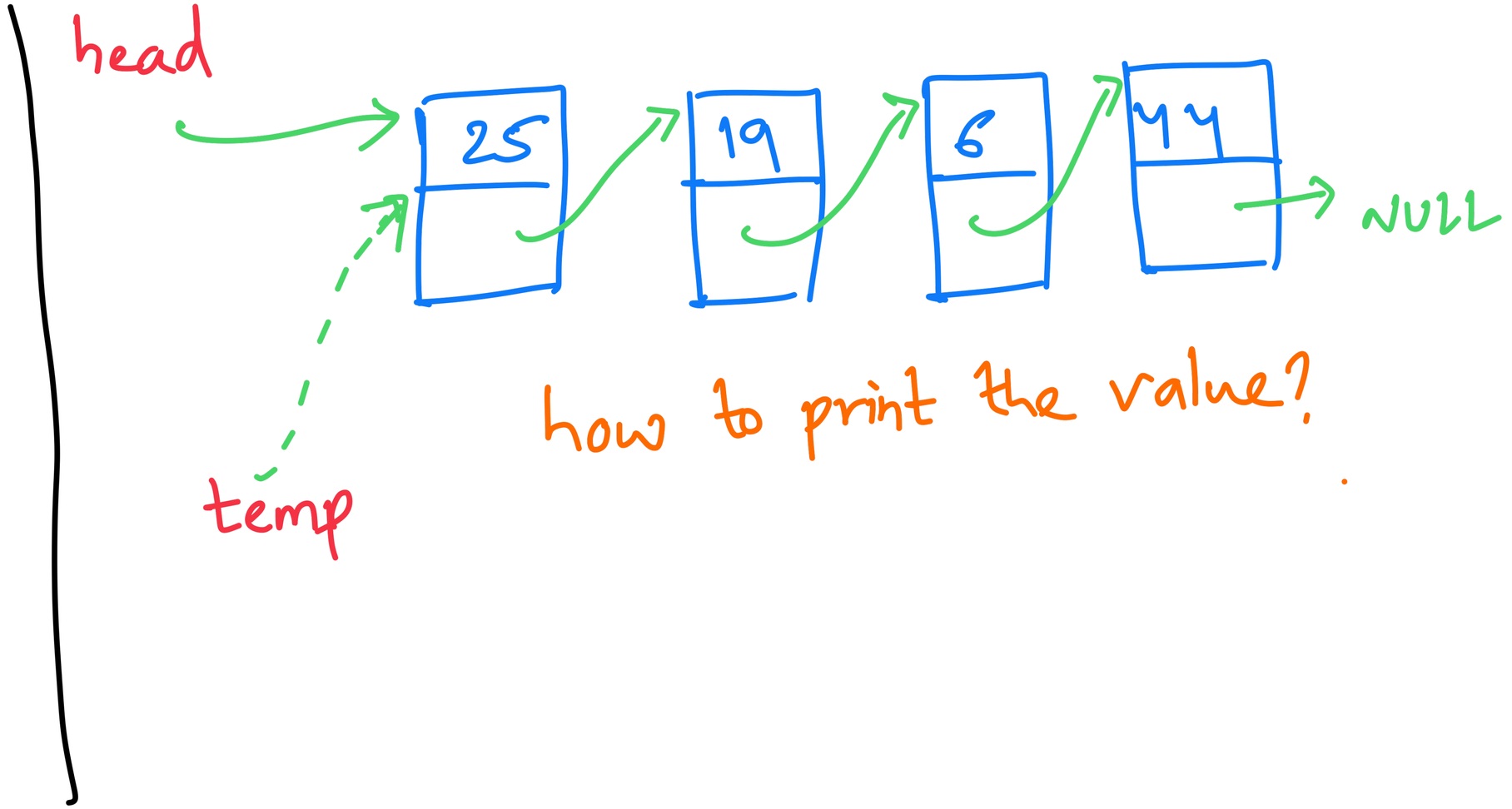


Linked Lists : Traversal

Printing all elements

Algorithm

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

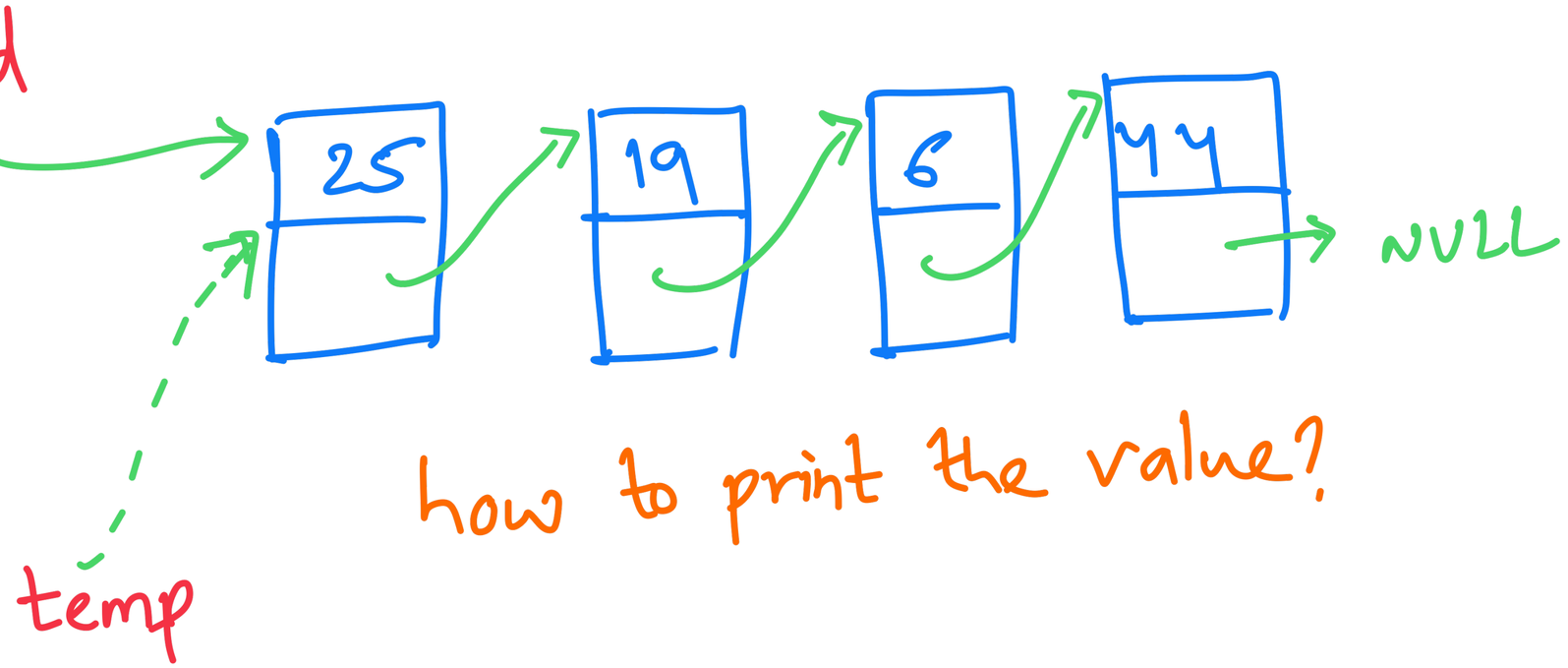


Linked Lists : Traversal

Printing all elements

Algorithm

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

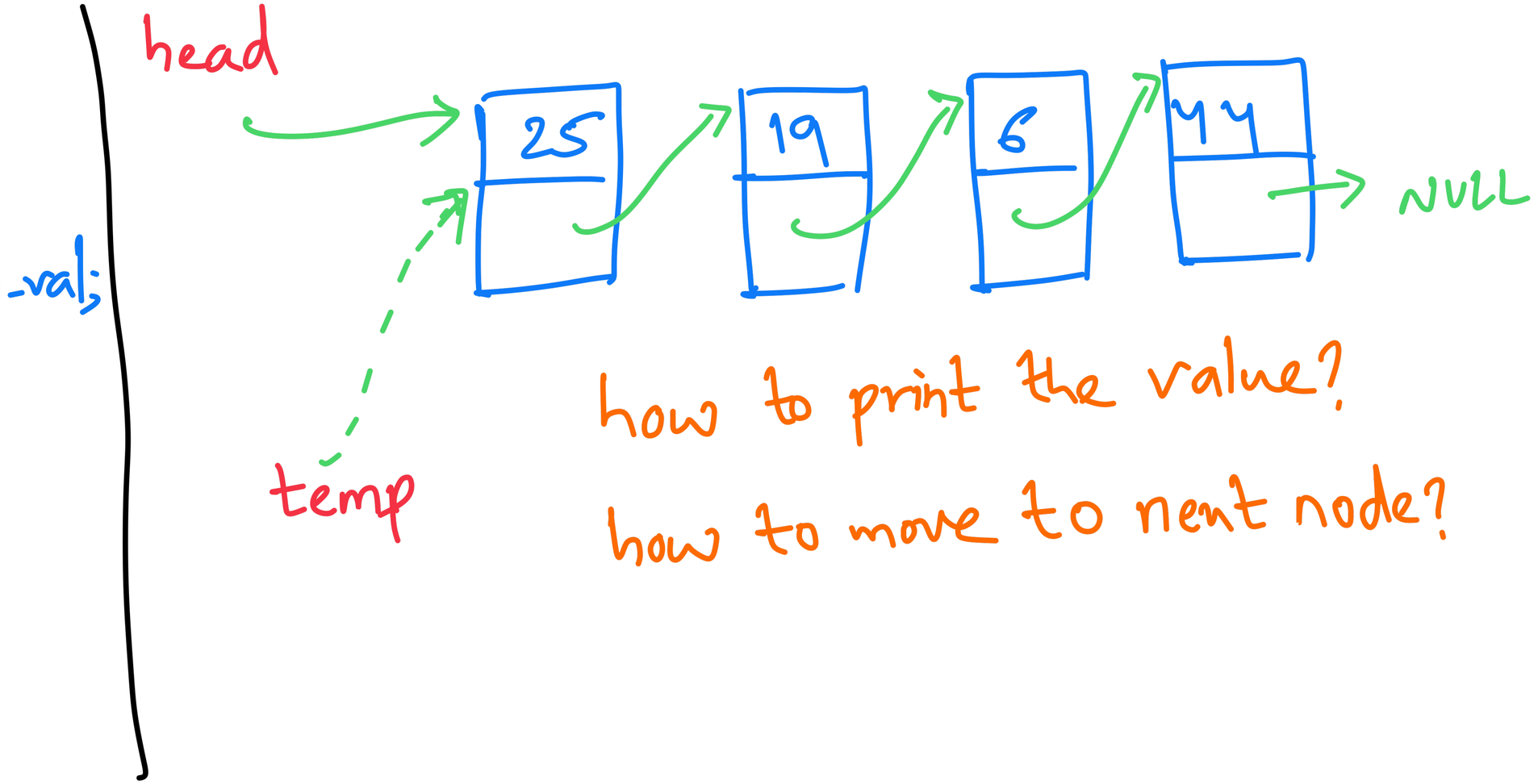


Linked Lists : Traversal

Printing all elements

Algorithm

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

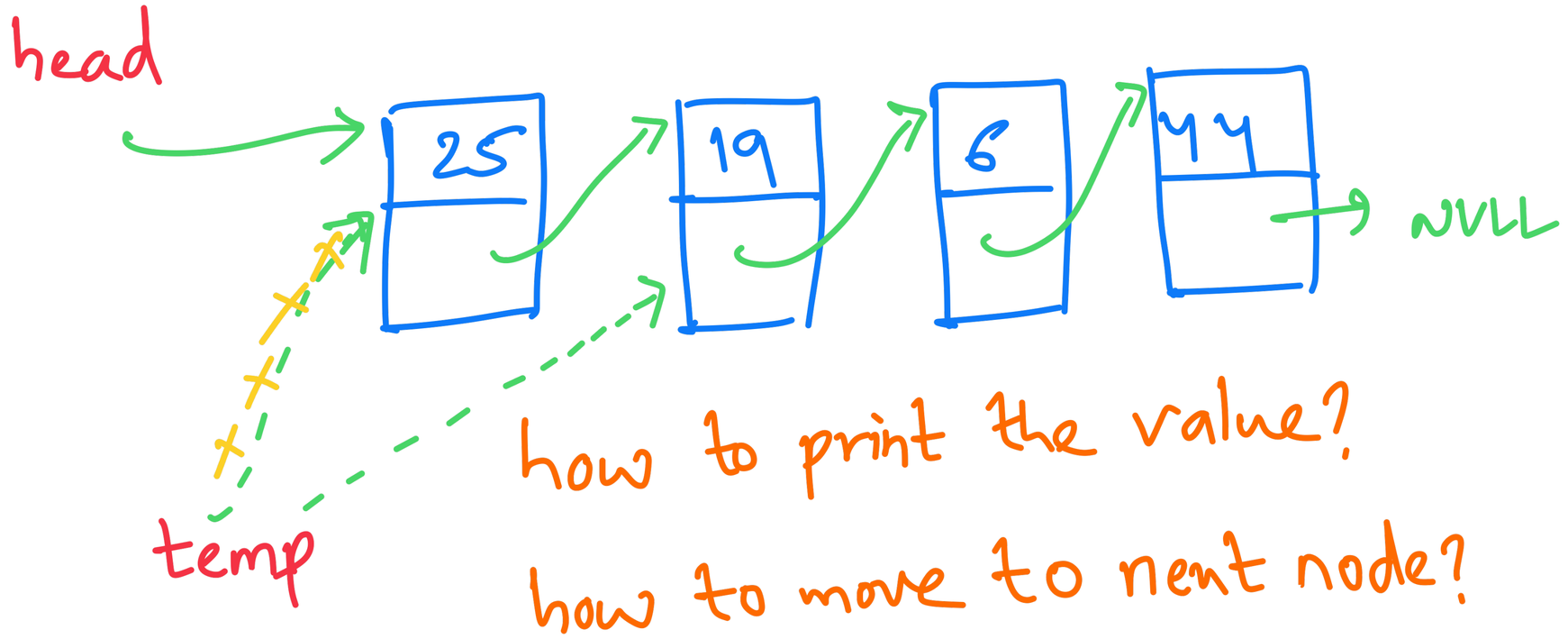


Linked Lists : Traversal

Printing all elements

Algorithm

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

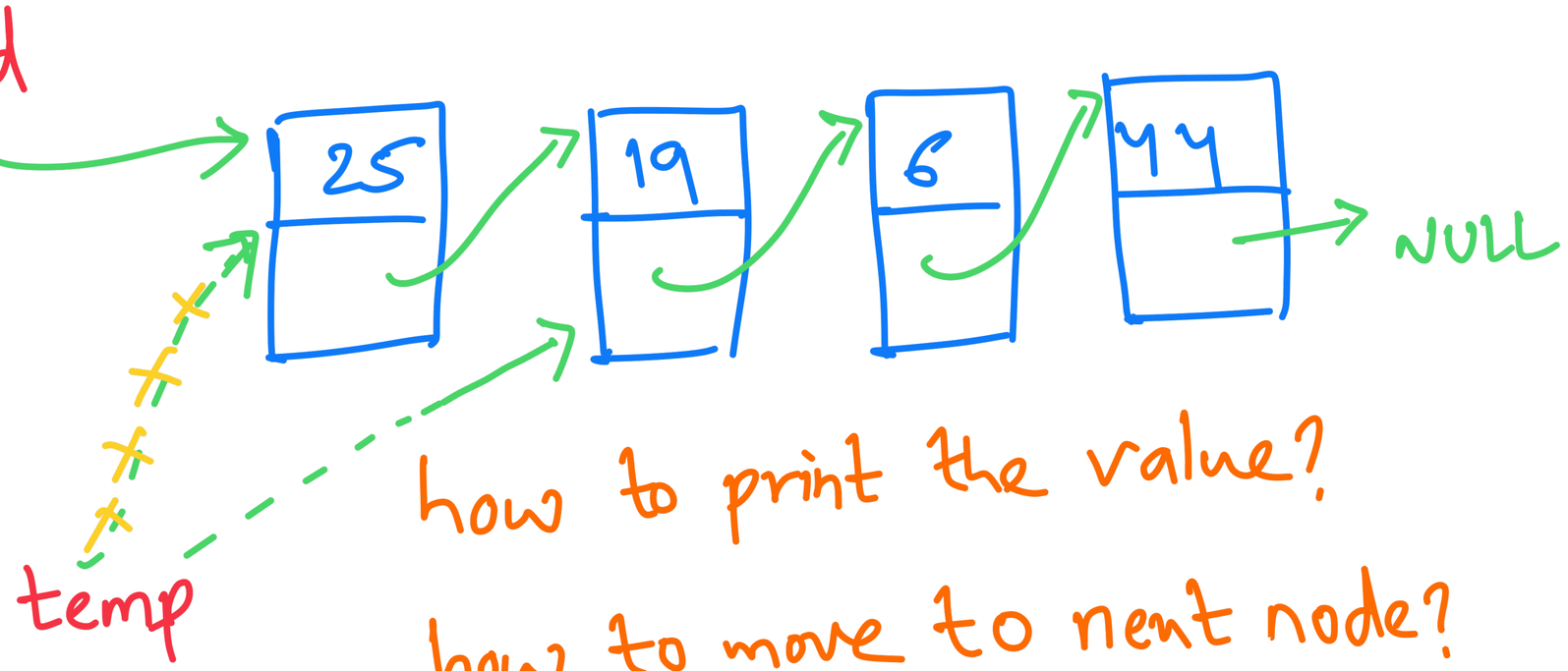


Linked Lists : Traversal

Printing all elements

Algorithm

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

temp = temp->next;
```



how to print the value?

how to move to next node?

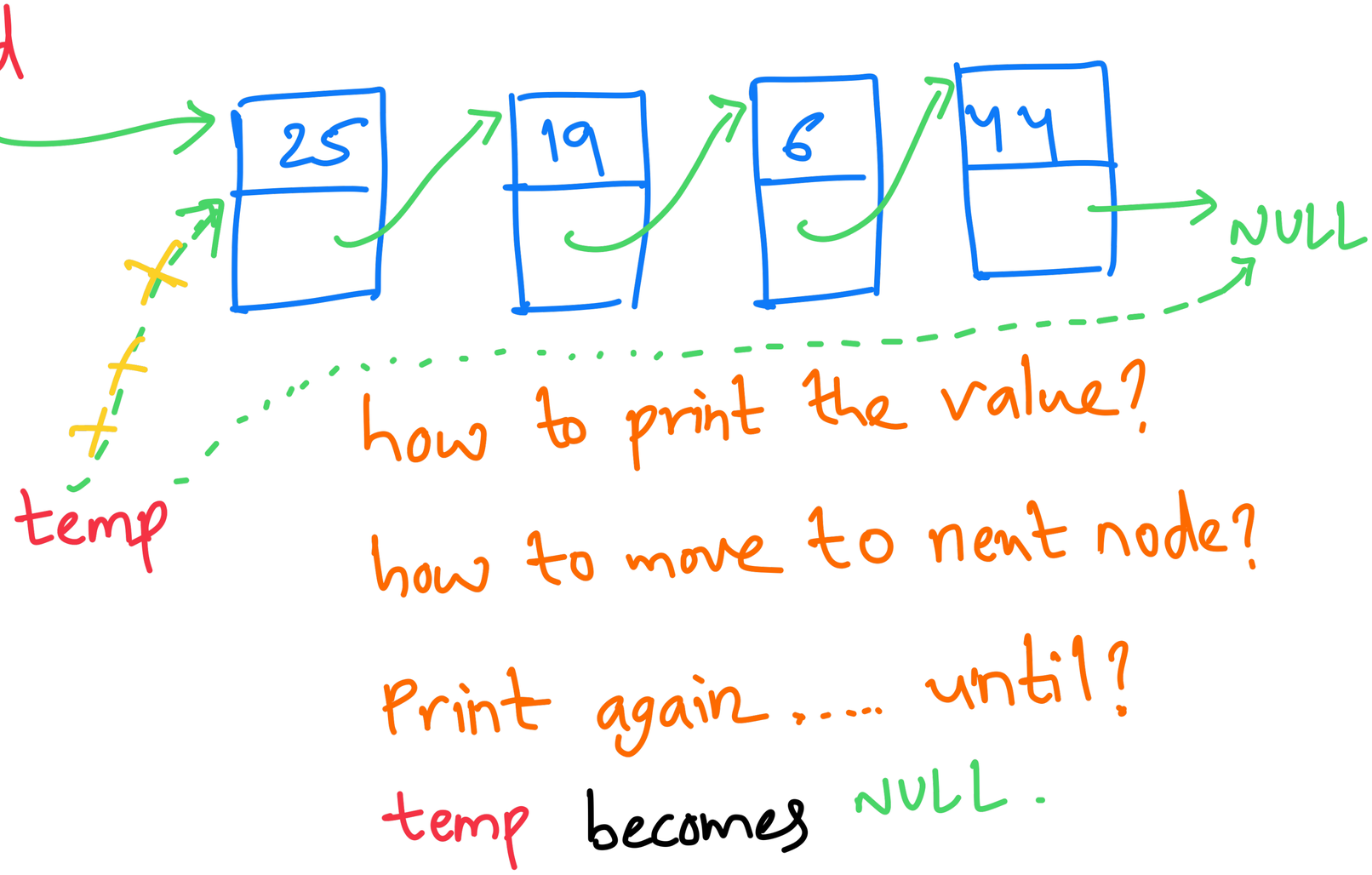
Print again ..... until?

Linked Lists : Traversal

Printing all elements

Algorithm

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

temp = temp->next;
```

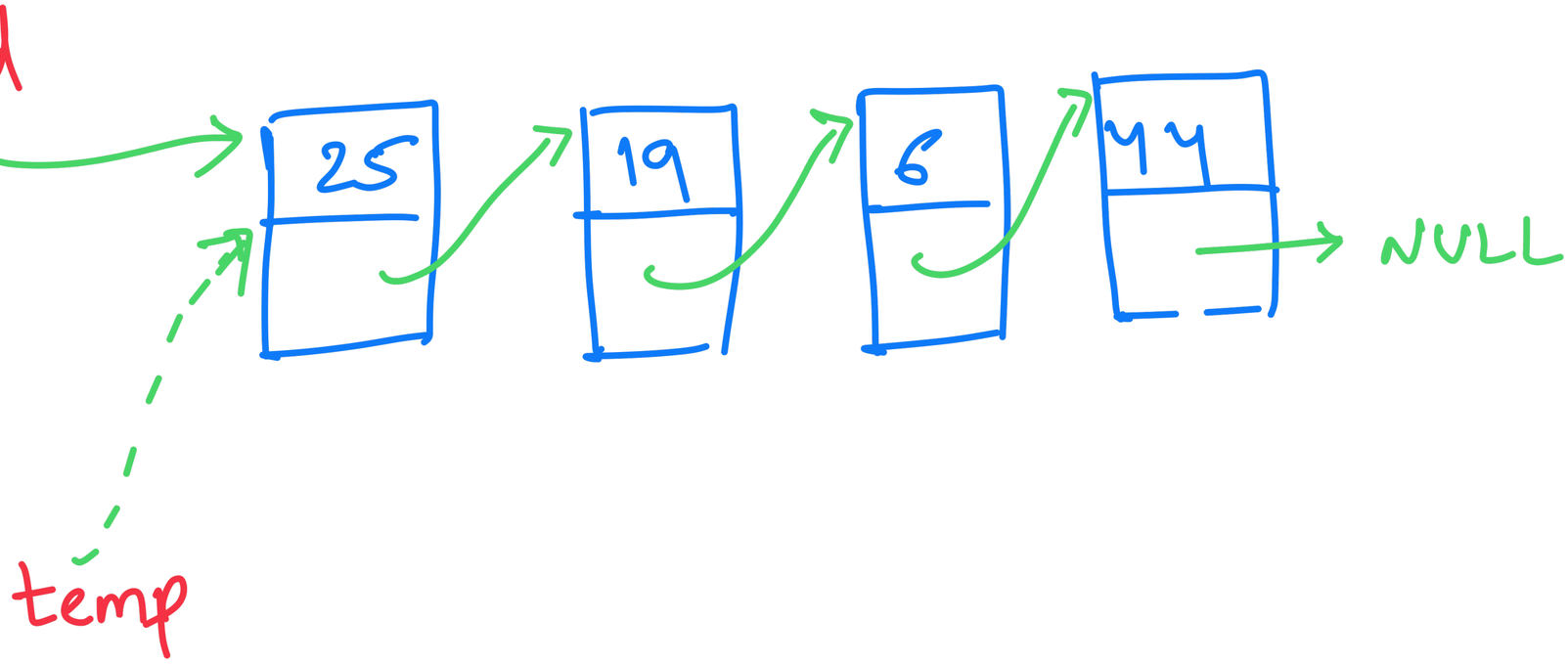


Linked Lists : Traversal

Printing all elements

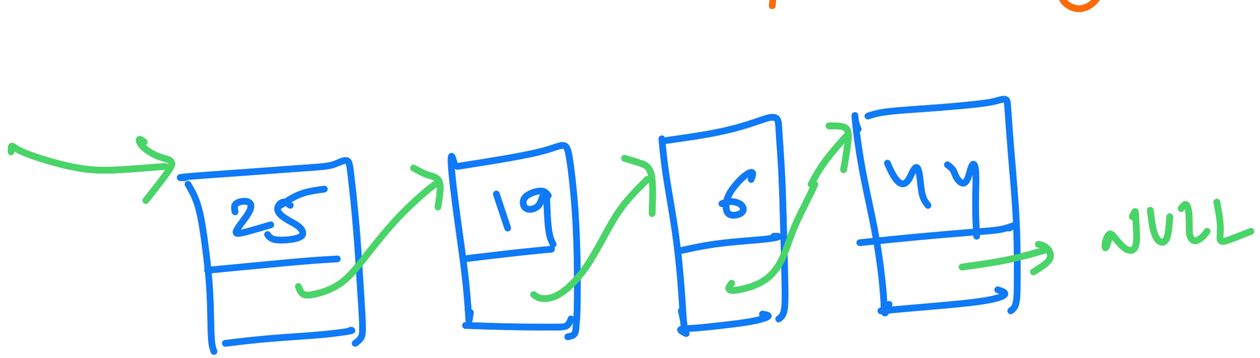
Algorithm

```
Node* temp;
temp = head;

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



Linked Lists : Traversal

Find a given element

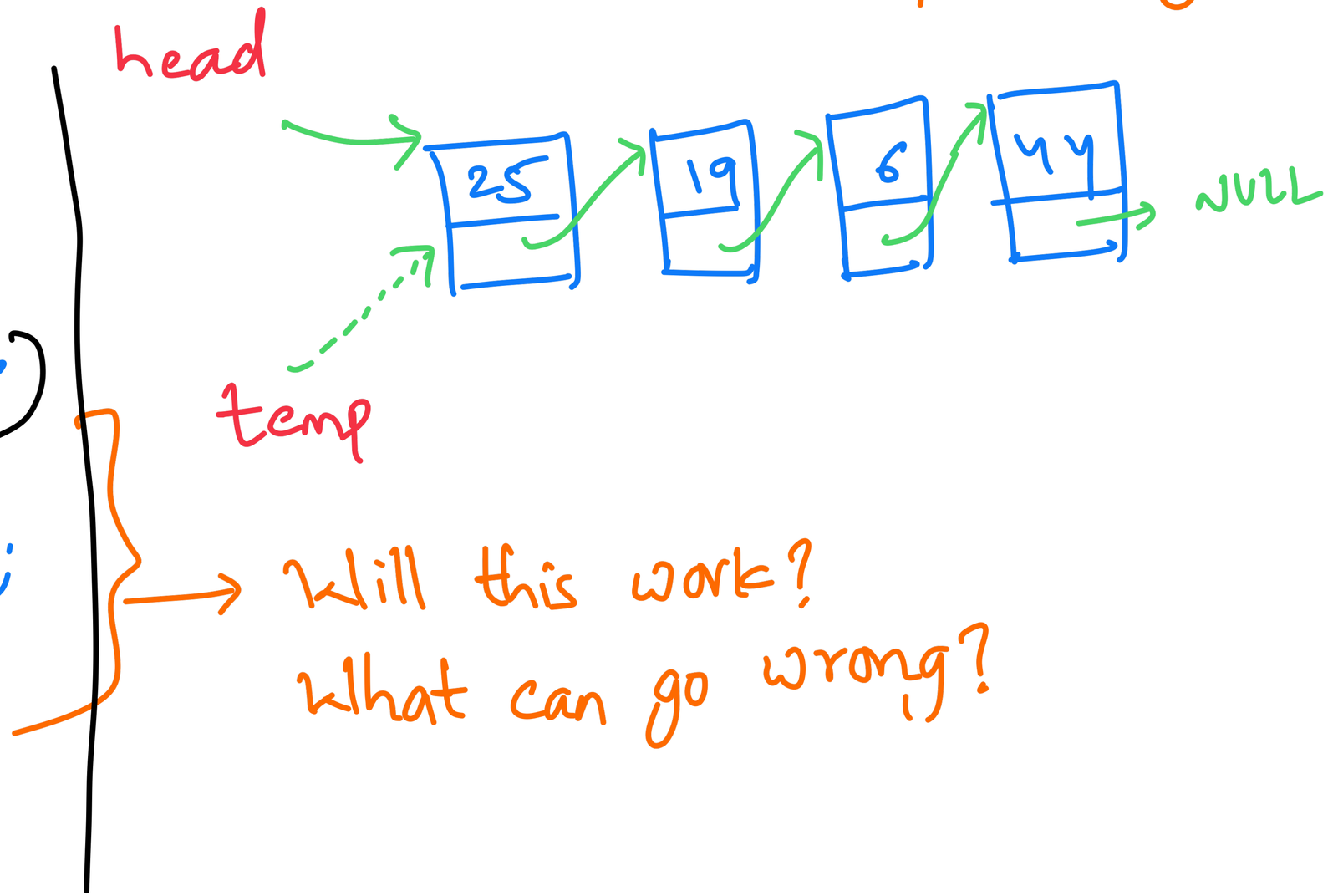


Linked Lists : Traversal

Find a given element

```
Node* temp;
temp = head;

while (temp->n_val != 6)
{
    temp = temp->next;
}

return temp;
```

Find node representing 6?

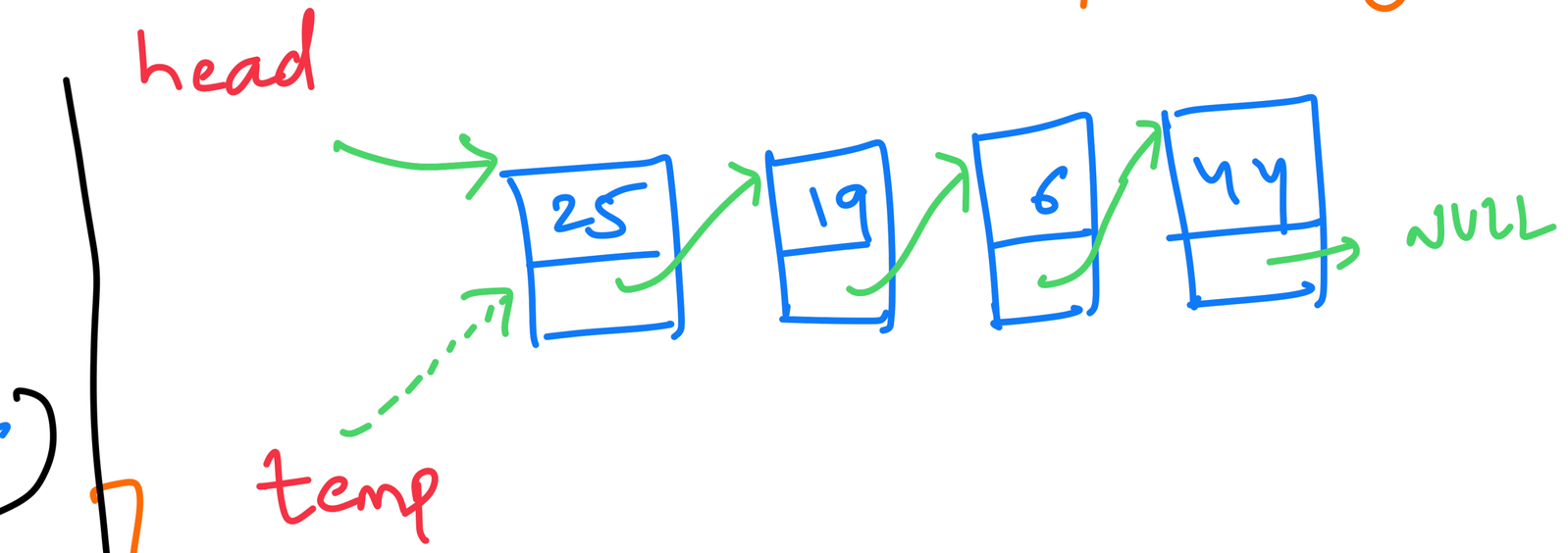


Linked Lists : Traversal

Find a given element

```
Node* temp;
temp = head;

while (temp->n_val != 6)
{
    temp = temp->next;
}

return temp;
```

Find node representing 6?



Will this work?

What can go wrong?

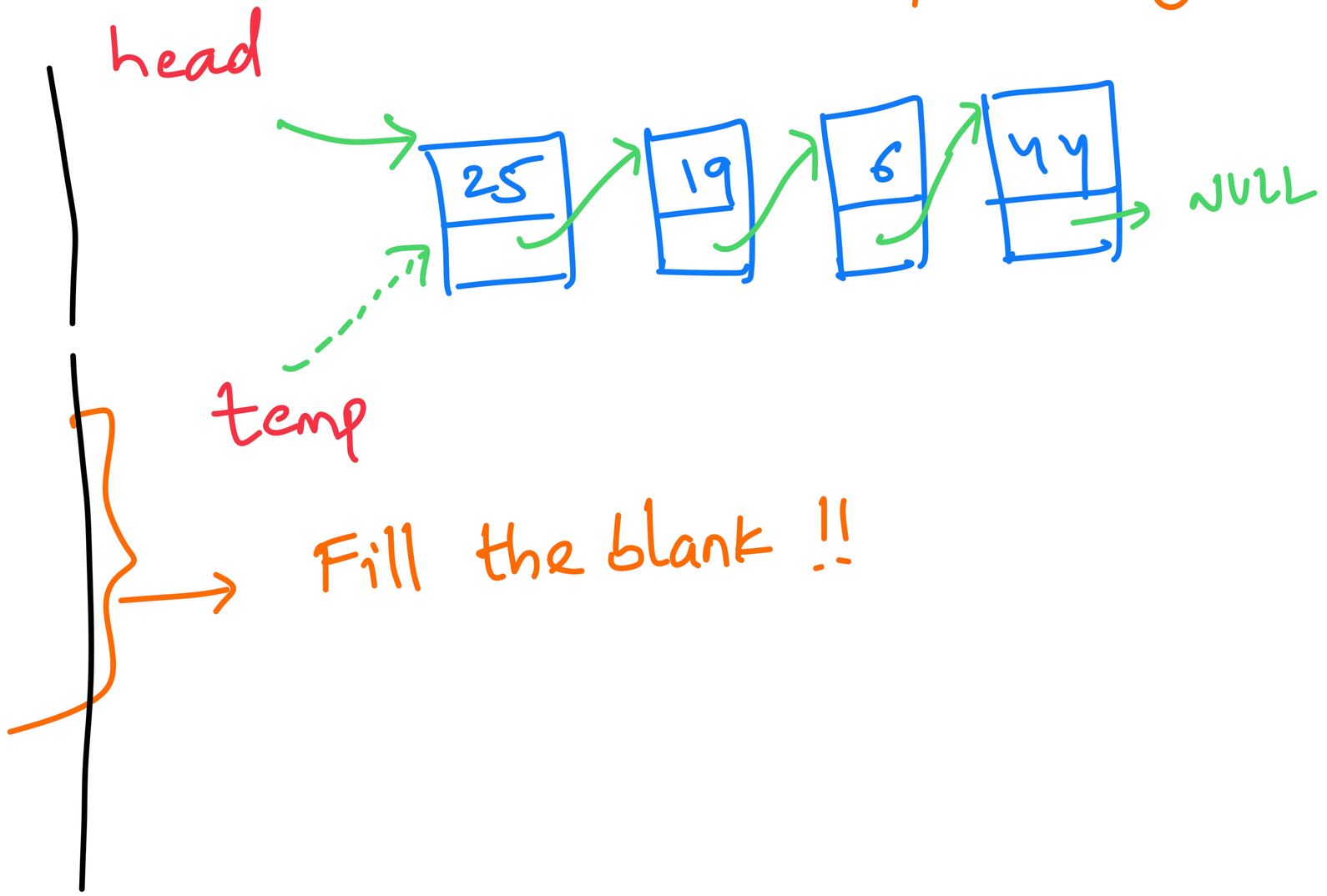
doesn't work if '6' is not in list.

"segmentation fault"

Linked Lists : Traversal

Find a given element

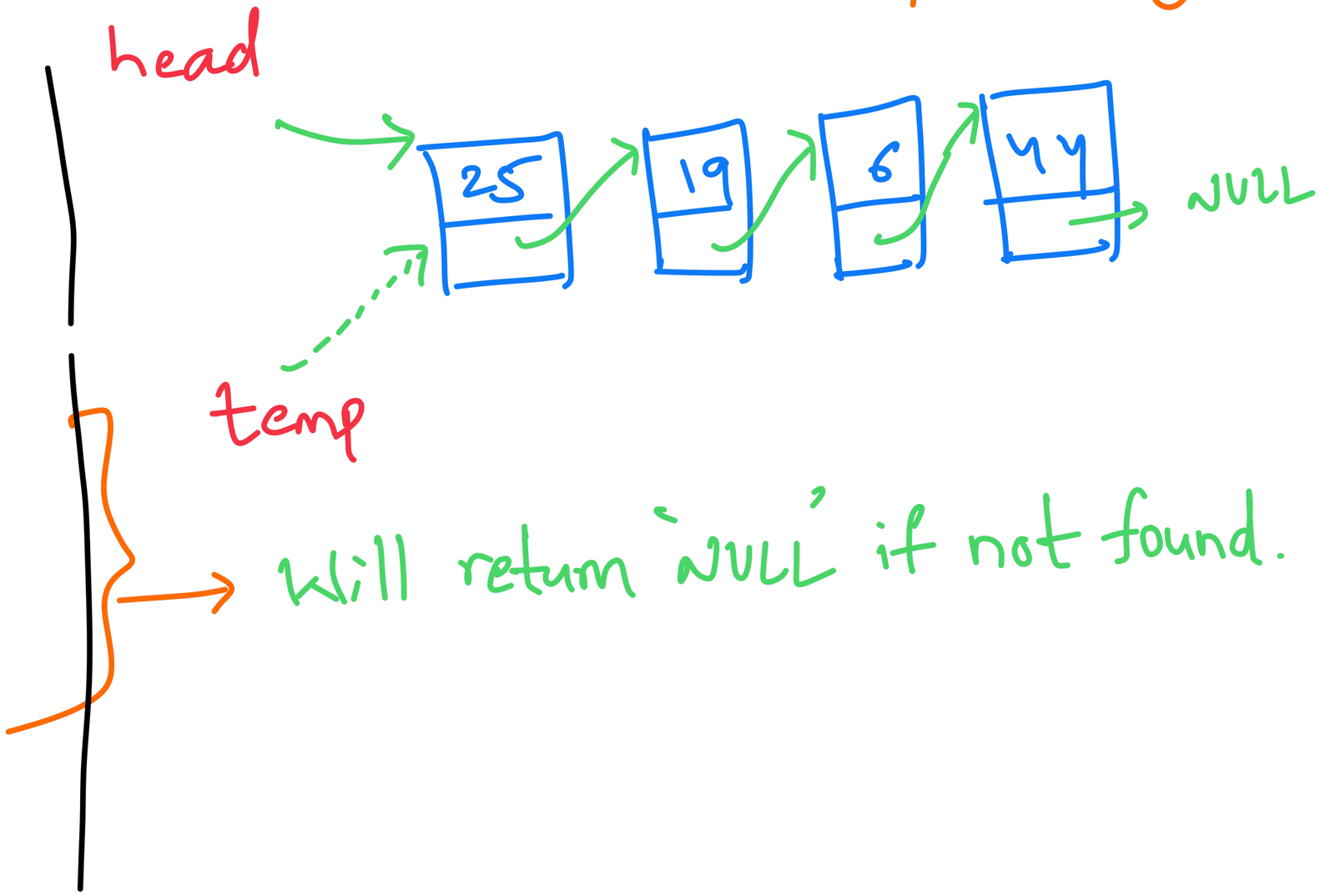
```
Node* temp;
temp = head;
while(temp != NULL){

    - - - - - -

    temp = temp->next;
}
return temp;
```



Linked Lists : Traversal

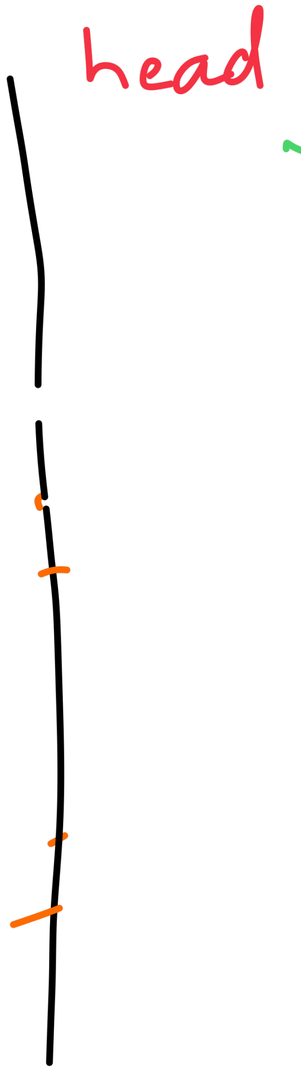
Find a given element

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

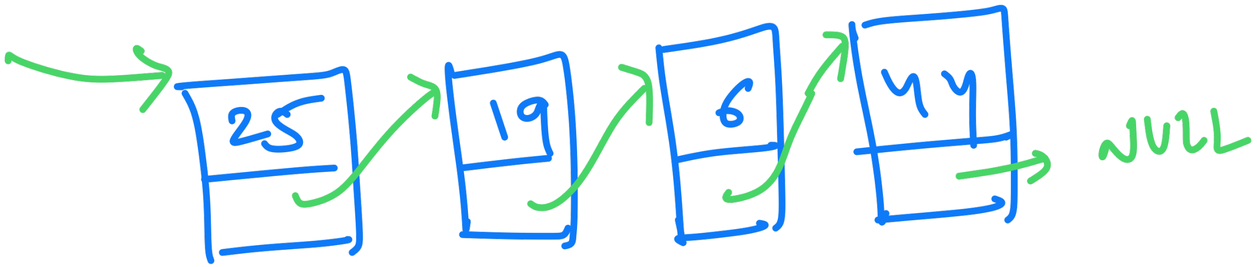


Linked Lists : Traversal

Find element at position i



Find node at position 2 ?



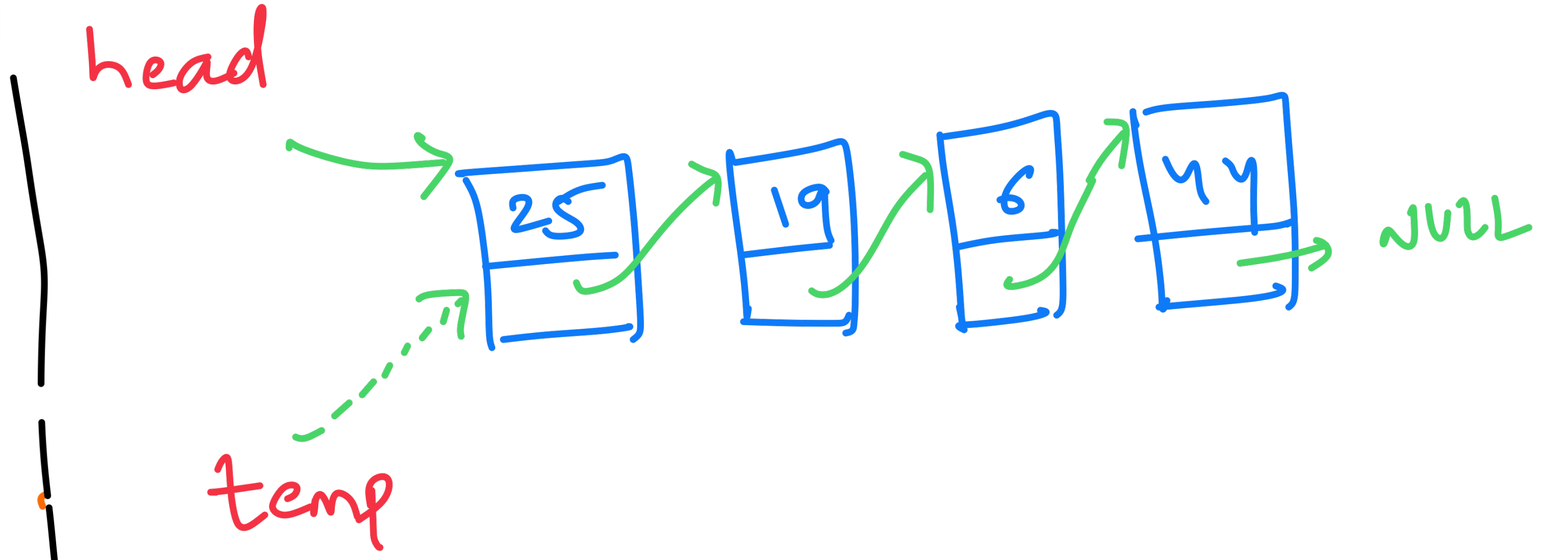
Linked Lists : Traversal

Find element at position i

```
Node* temp = head;
①  _ _ _ _ _ _ _ _ _
while(temp != NULL){
②  _ _ _ _ _ _ _ _ _
③  _ _ _ _ _ _ _ _
    temp = temp->next;
}
return temp;
```

Fill the 3 blanks !!!

Find node at position 2 ?

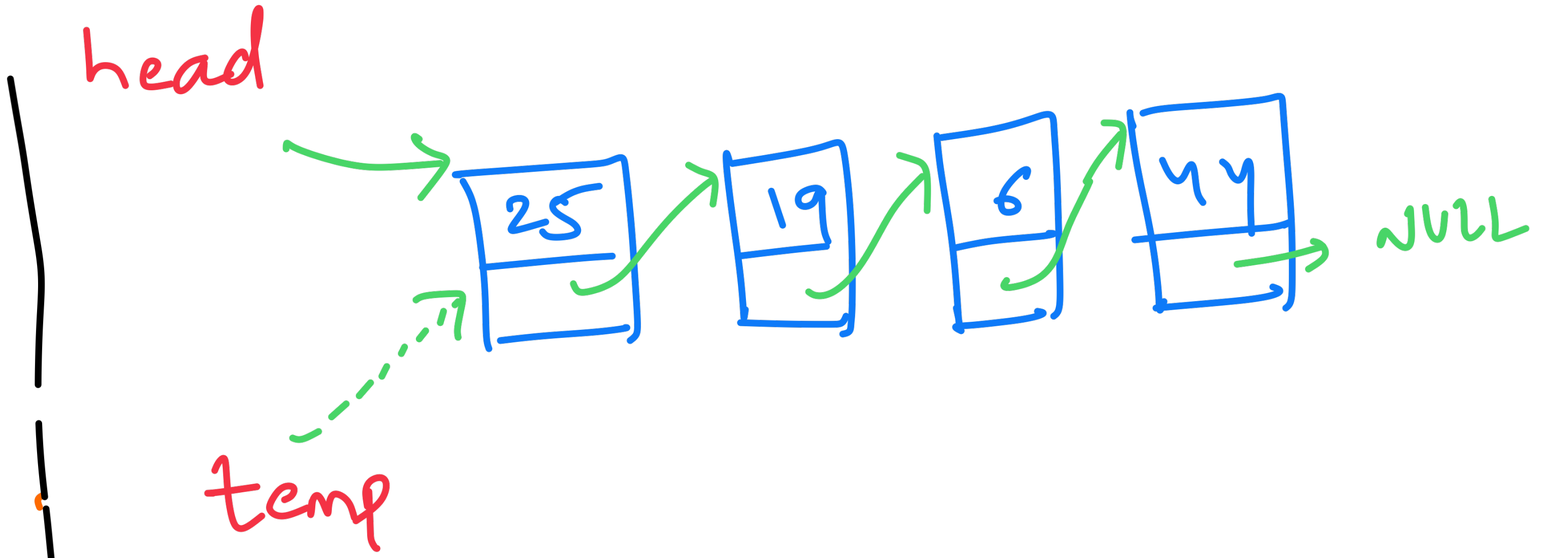


Linked Lists : Traversal

Find element at position i

```
Node* temp = head;
int pos = 0;
while(temp != NULL){
(2) ---------
(3) ---------
    temp = temp->next;
}
return temp;
```

Find node at position 2 ?



Fill the 3 blanks !!!

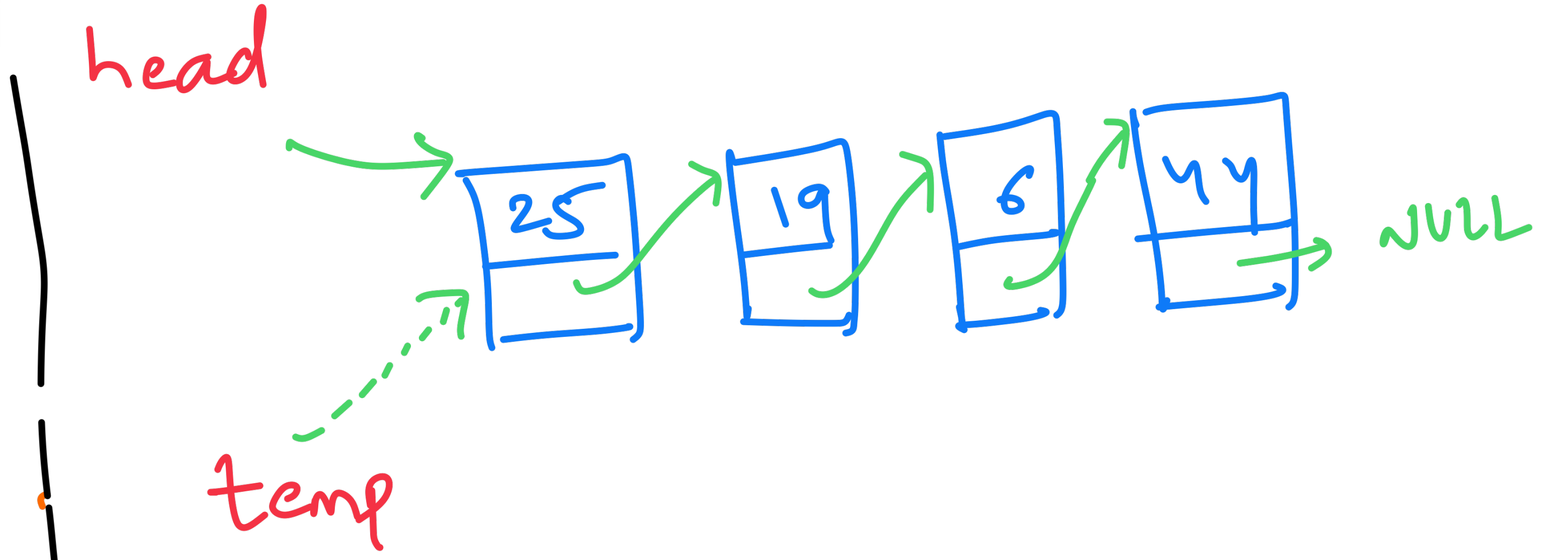
Linked Lists : Traversal

Find element at position i

```
Node* temp = head;
int pos = 0;
while(temp != NULL){
    if(pos == 2){break;}
③ - - - - - - - -
    temp = temp->next;
}
return temp;
```

Fill the 3 blanks !!!

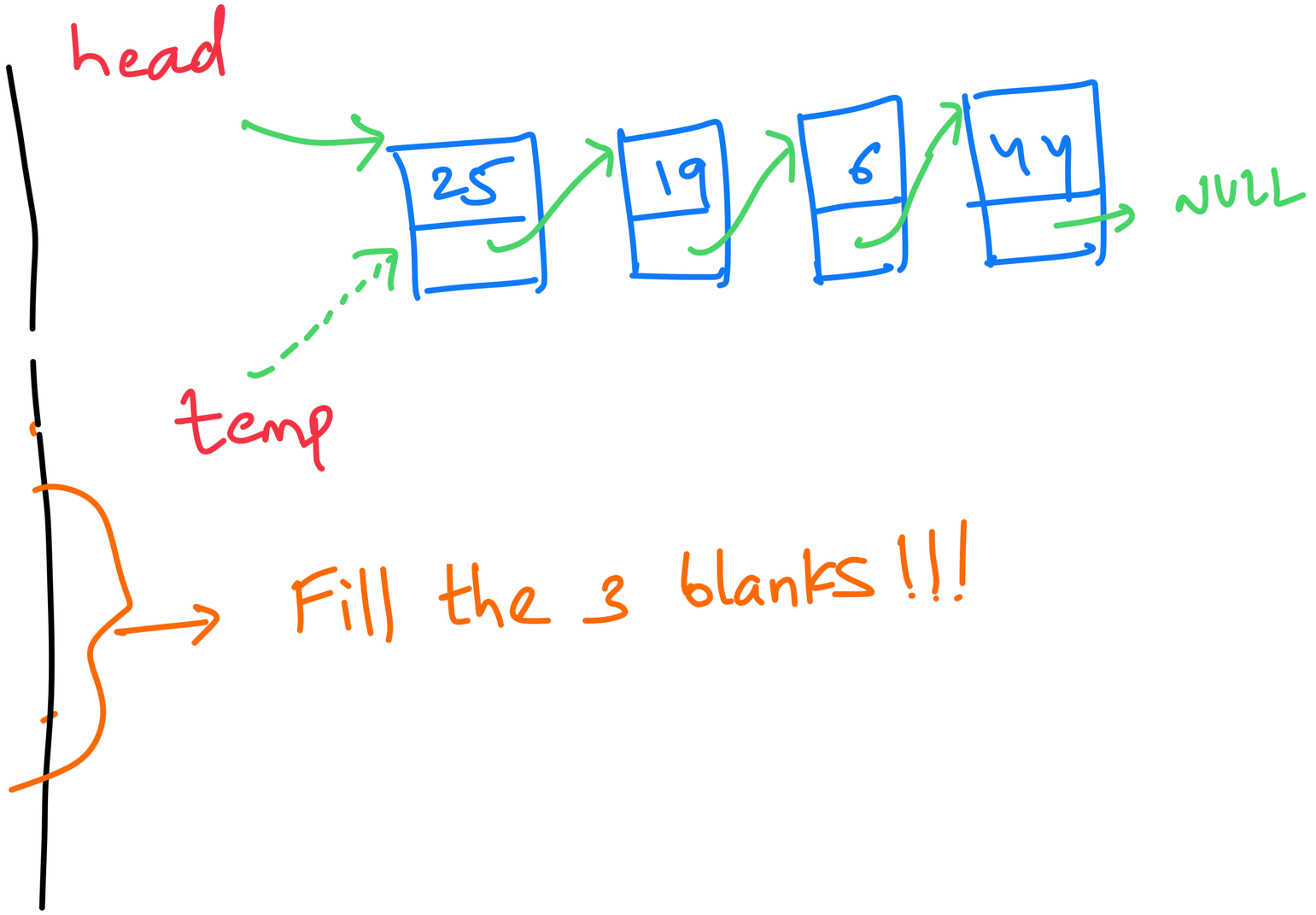
Find node at position 2 ?



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;
```

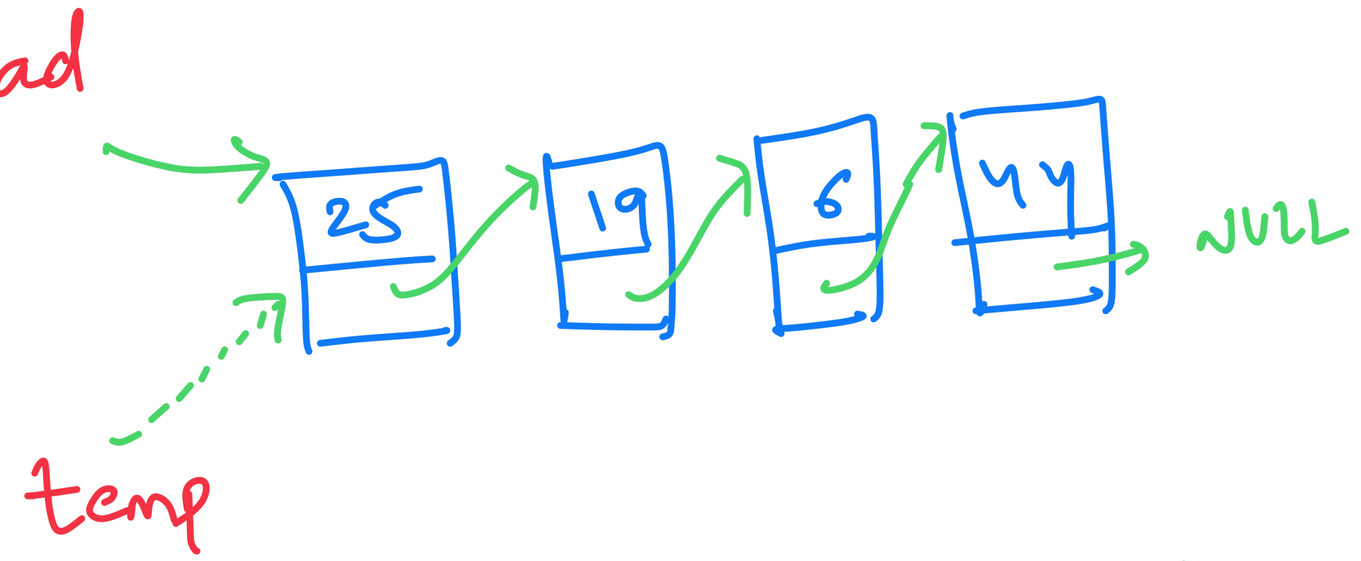


Linked Lists : Traversal

Find element at position i

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

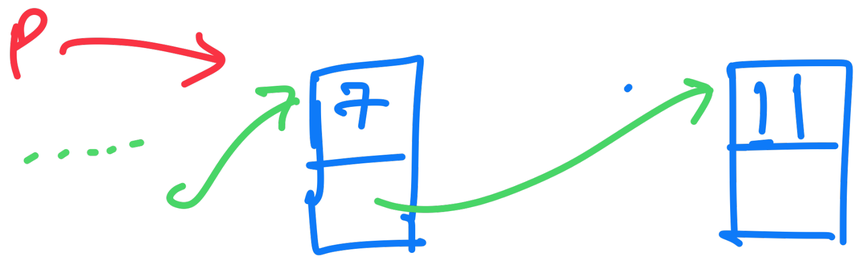
Find node at position 2 ?



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

Linked Lists : Insertion

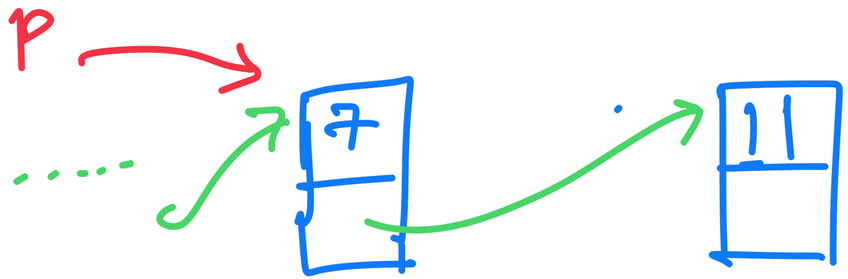
Insert element in between



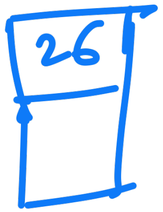
insert 26?

Linked Lists : Insertion

Insert element in between



insert 26?

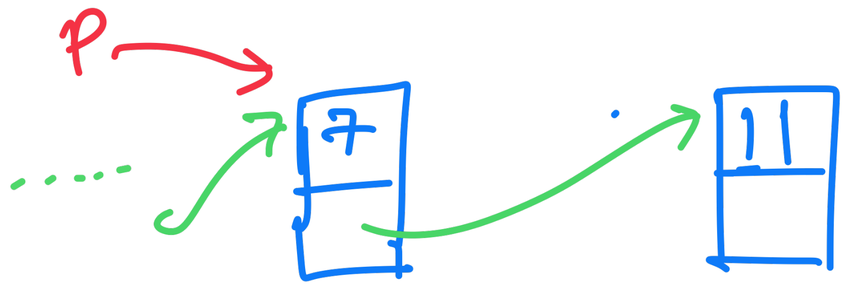



```
Node* n = new Node();
```

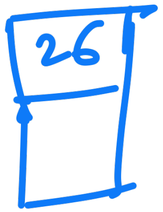
① create new node

Linked Lists : Insertion

Insert element in between



insert 26?



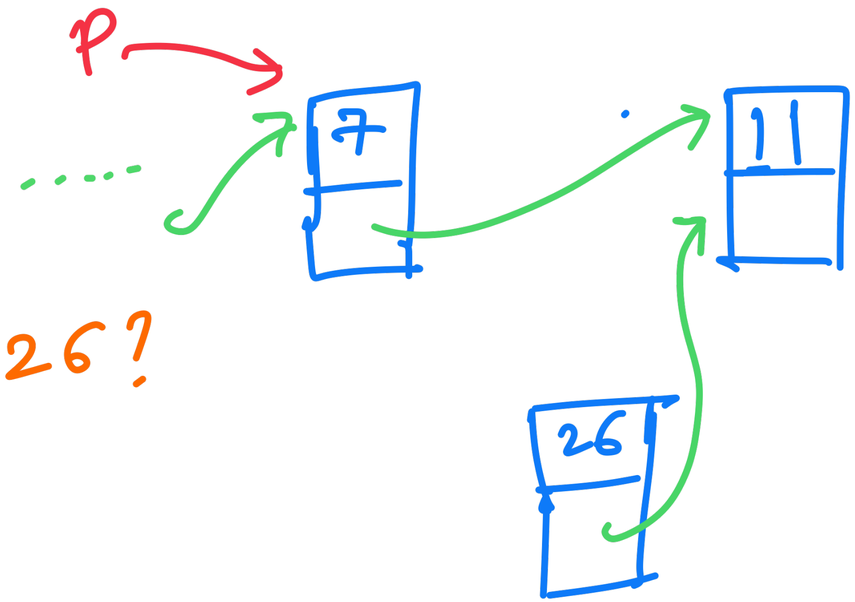

```
Node* n = new Node();
```

① create new node

② rewire it!!

Linked Lists : Insertion

Insert element in between



insert 26?

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

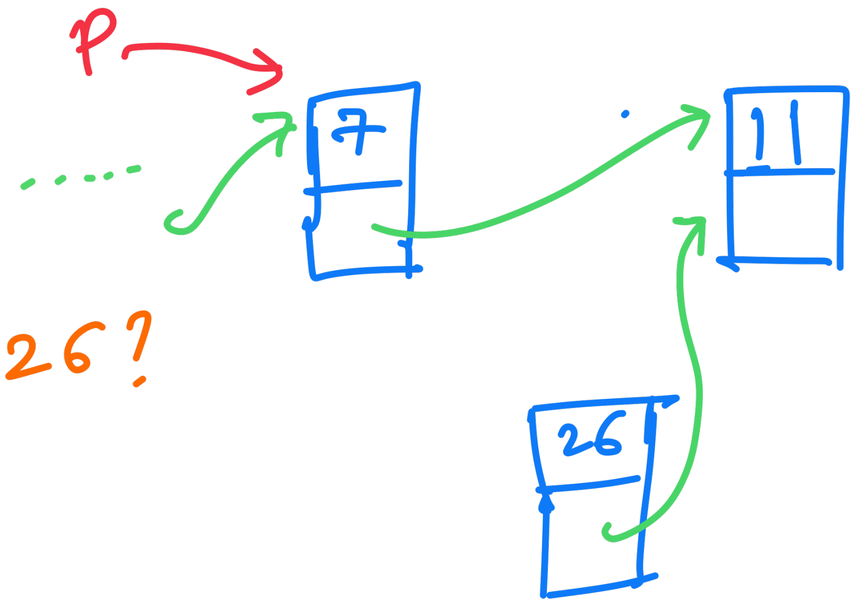
① _ _ _ _ _ _ _ .

① create new node

② rewire it!!

Linked Lists : Insertion

Insert element in between



insert 26?

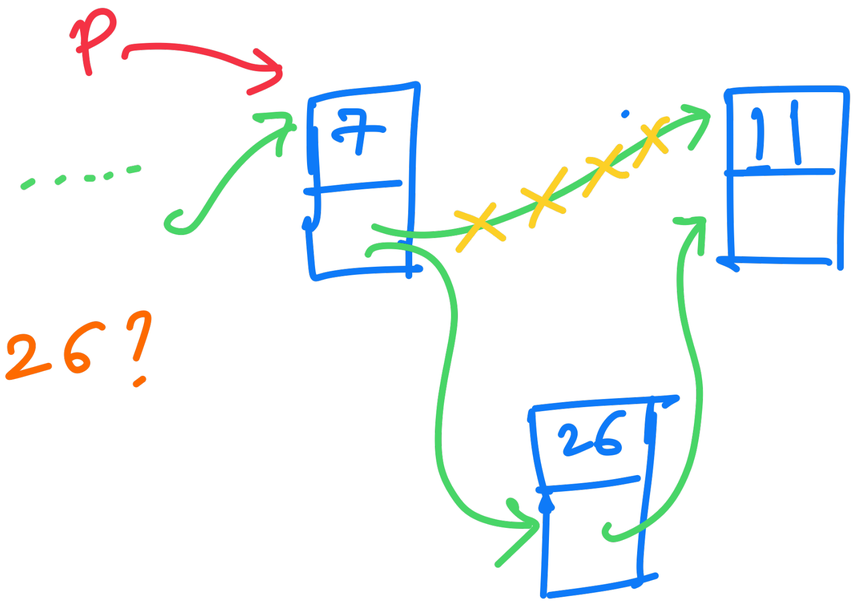
```
Node* n = new Node();
n->next = p->next;
```

① create new node

② rewire it!!

Linked Lists : Insertion

Insert element in between



insert 26?

```
Node* n = new Node();
n->next = p->next;
```

① _ _ _ _ _ _ _ _ _

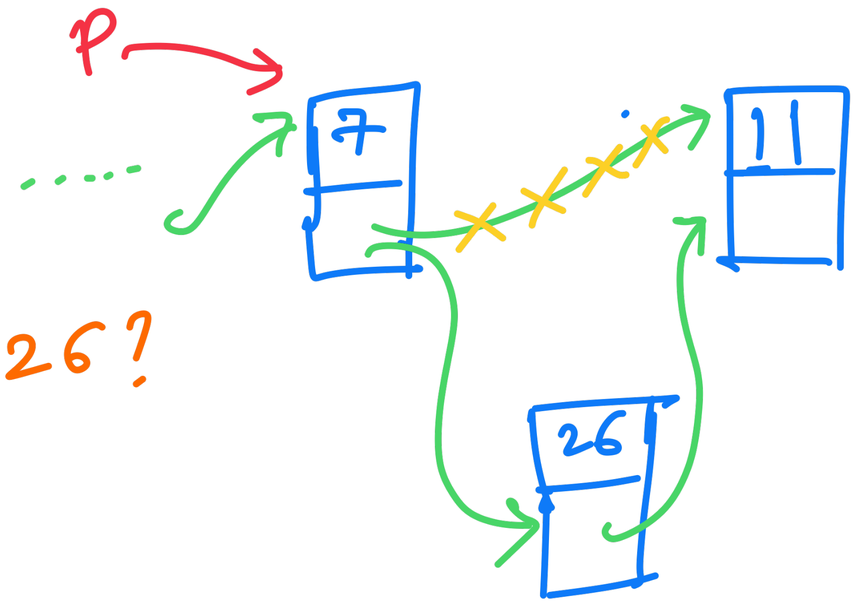
② _ _ _ _ _ _ _ _ _

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;`

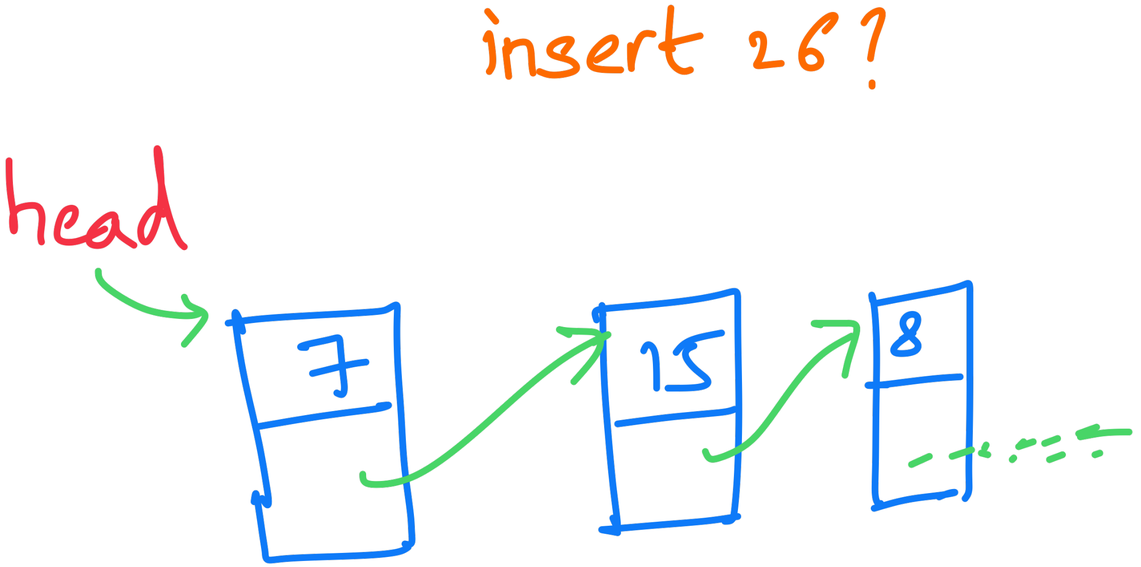
② `p->next = n;`

① create new node

② rewire it!!

Linked Lists : Insertion

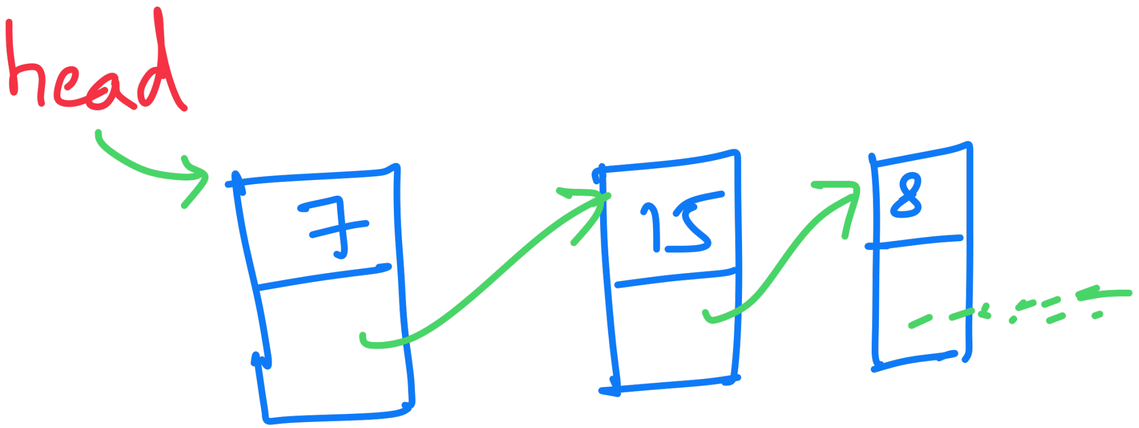
Insert element at beginning



Linked Lists : Insertion

Insert element at beginning

insert 26 ?



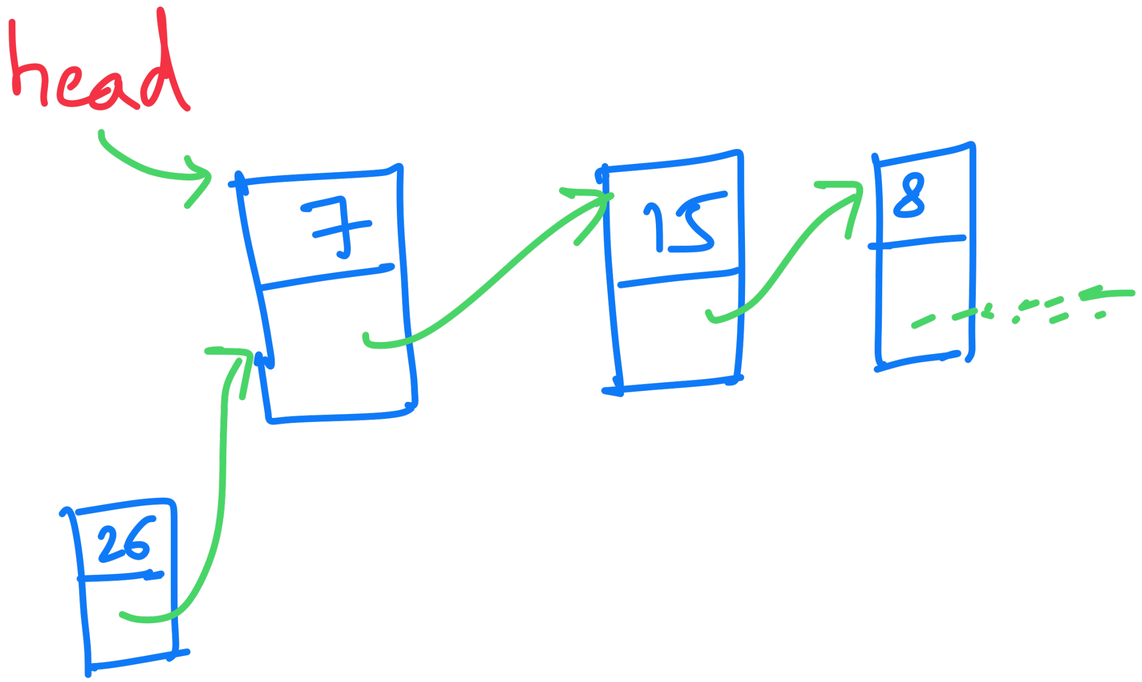

```
Node* n = new Node();
```

① create new node

Linked Lists : Insertion

Insert element at beginning

insert 26 ?




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

① _ _ _ _ _ _ _ _ .

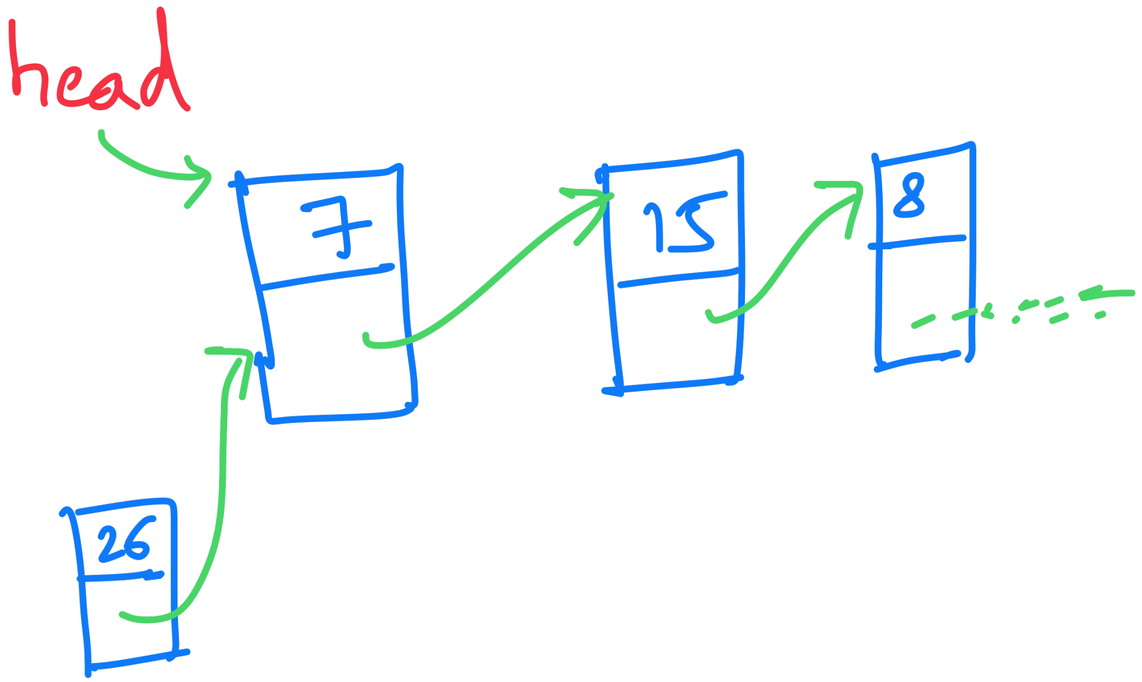
① create new node

② rewire !!

Linked Lists : Insertion

Insert element at beginning

insert 26 ?




```
Node* n = new Node();
① n->next = head;
```

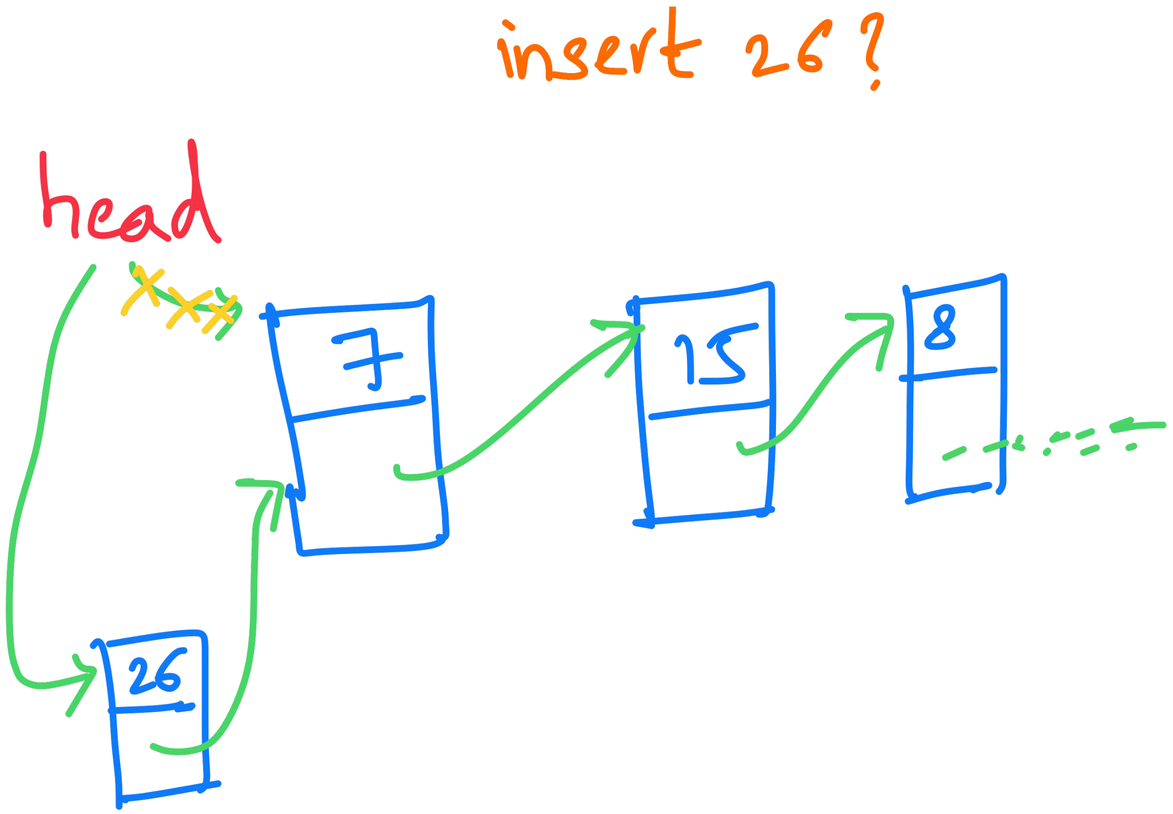
① create new node

② rewire !!

Linked Lists : Insertion

Insert element at beginning

```
Node* n = new Node();
n->next = head;
head = n;
```

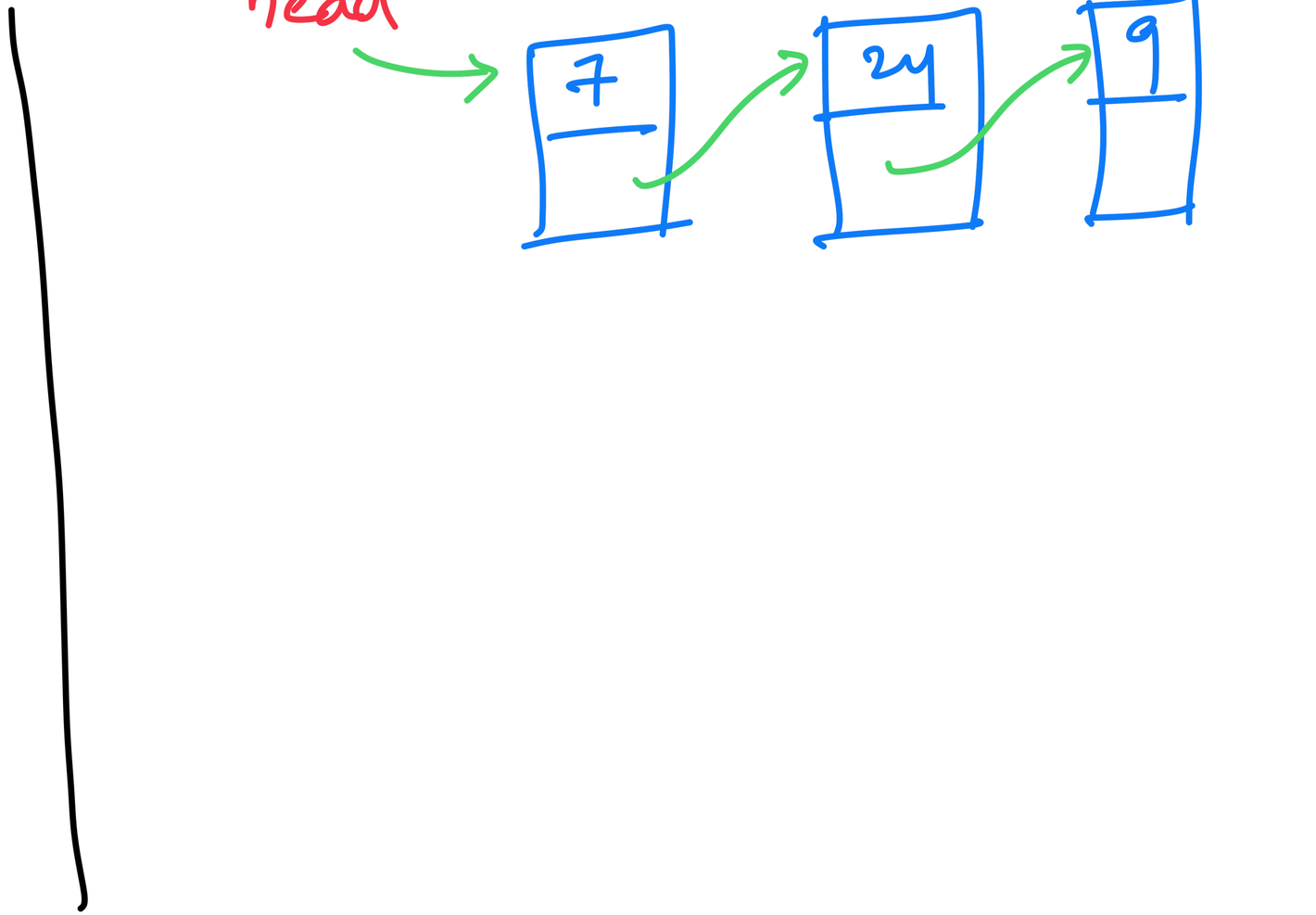


① create new node

② rewire !!

Linked Lists : Insertion

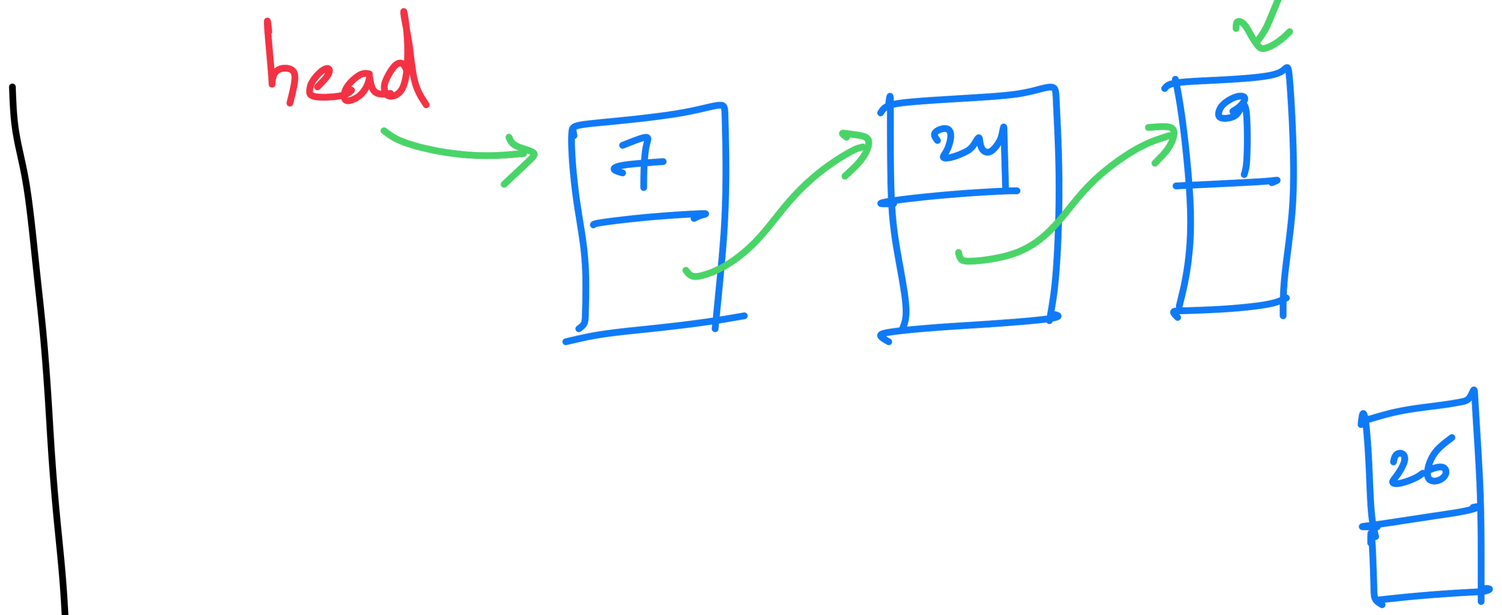
Insert element at end



Linked Lists : Insertion

Insert element at end

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

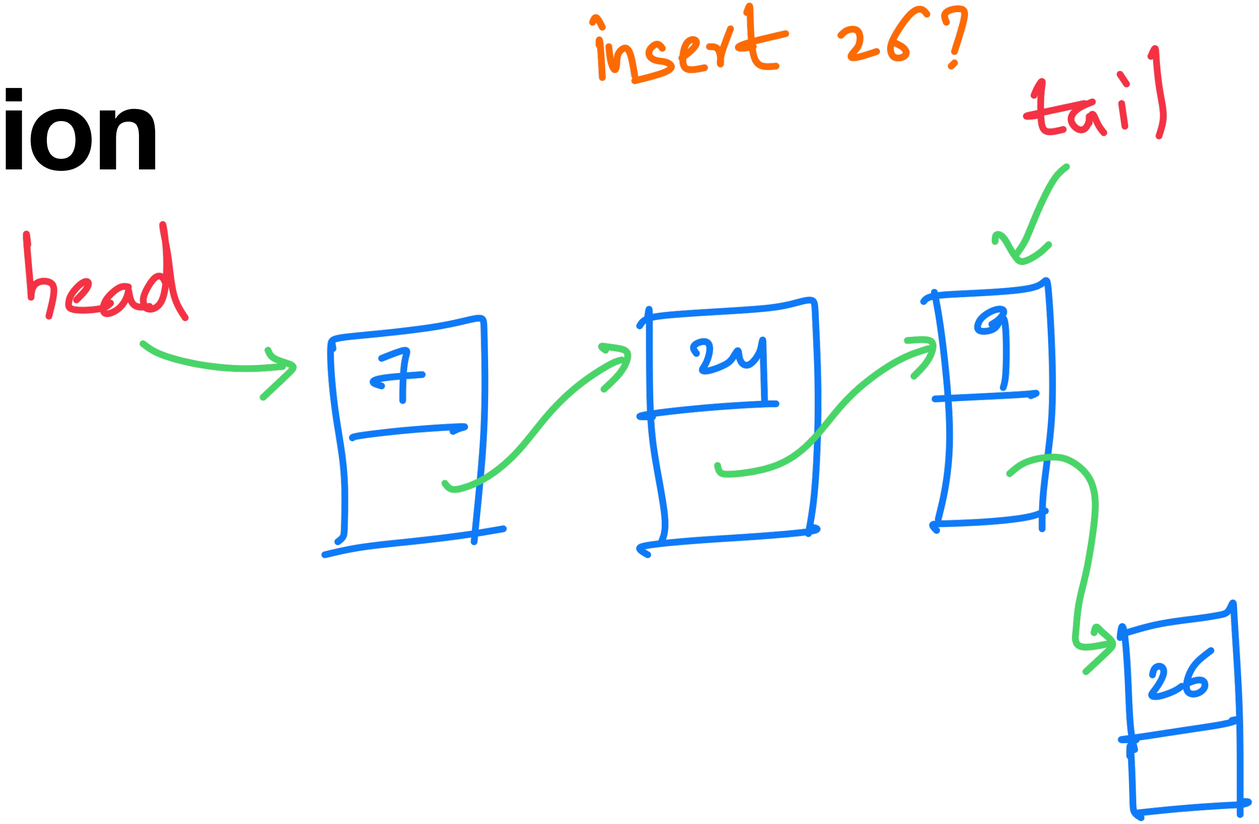


Linked Lists : Insertion

Insert element at end

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

① _ _ _ _ _ _ _

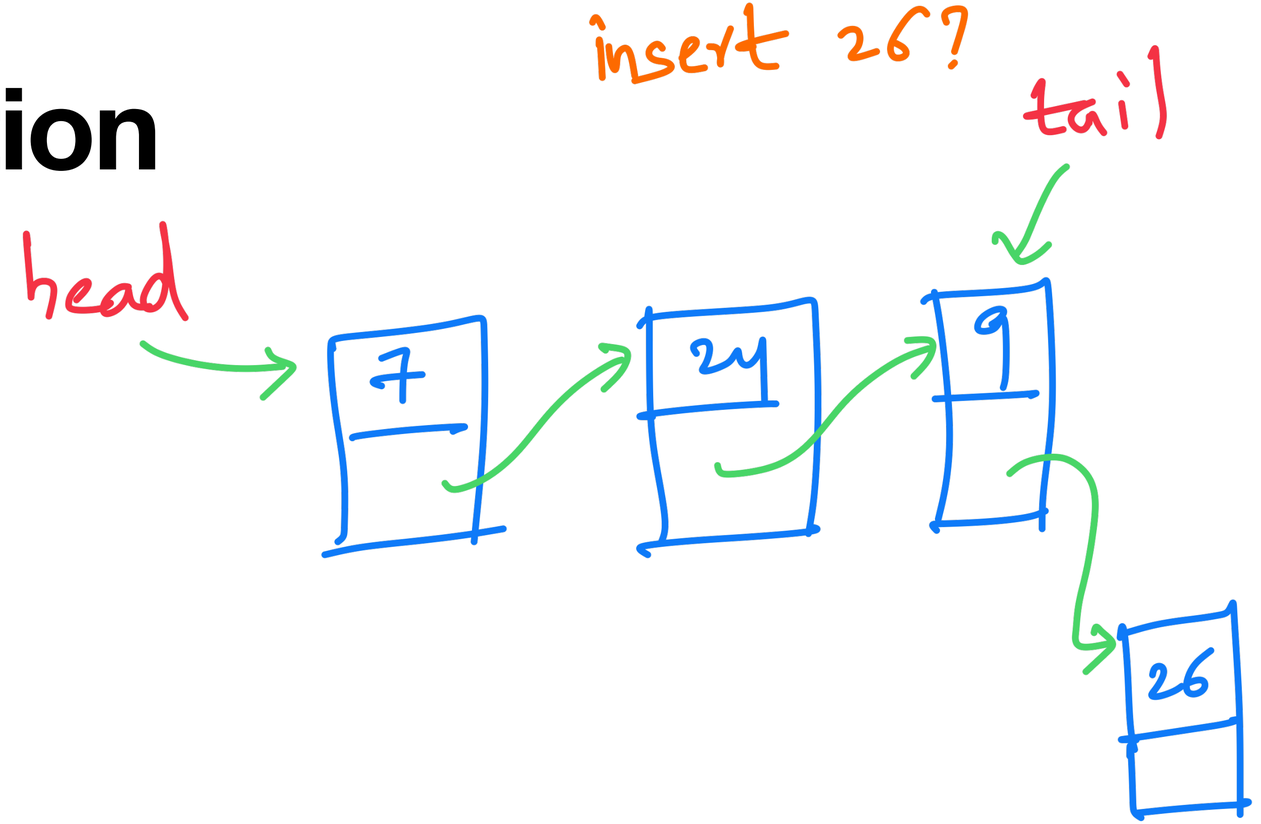


Linked Lists : Insertion

Insert element at end

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

① `tail->next = n;`



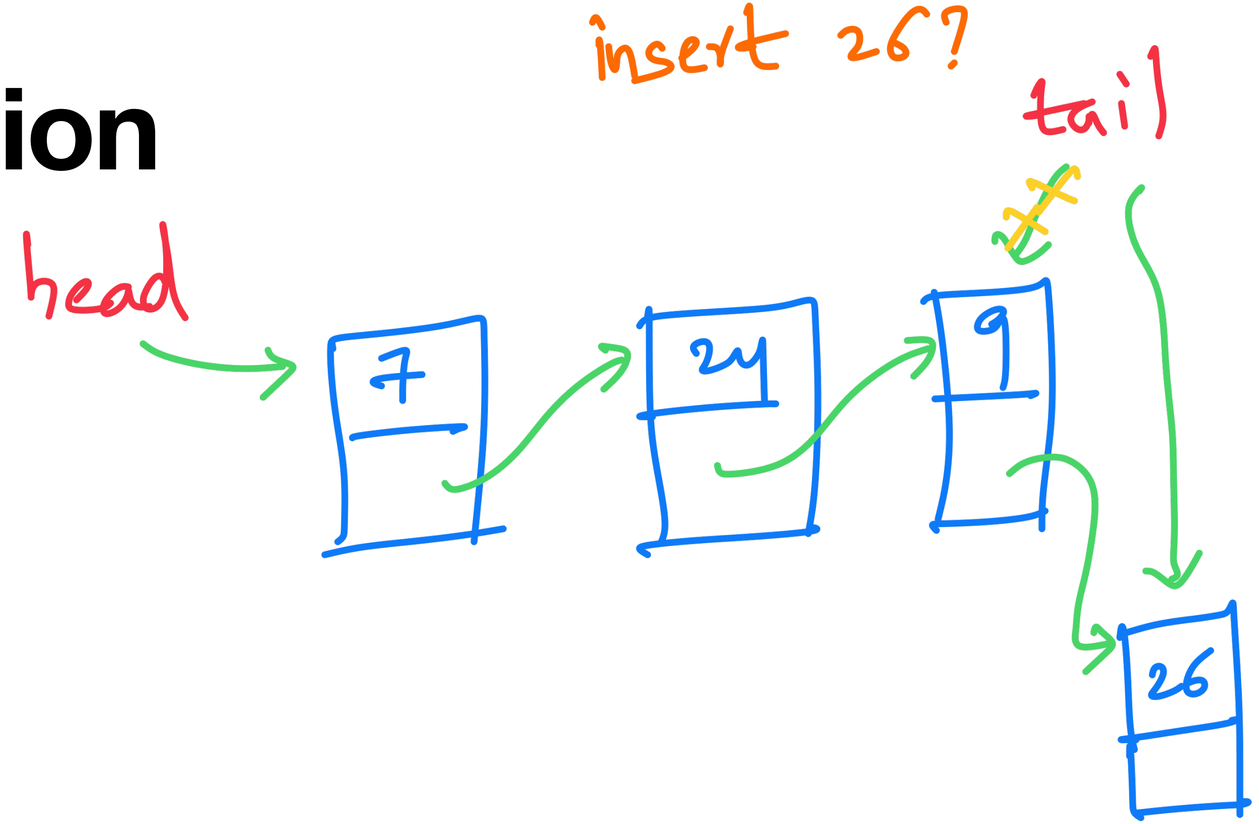
Linked Lists : Insertion

Insert element at end

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

① `tail->next = n;`

②



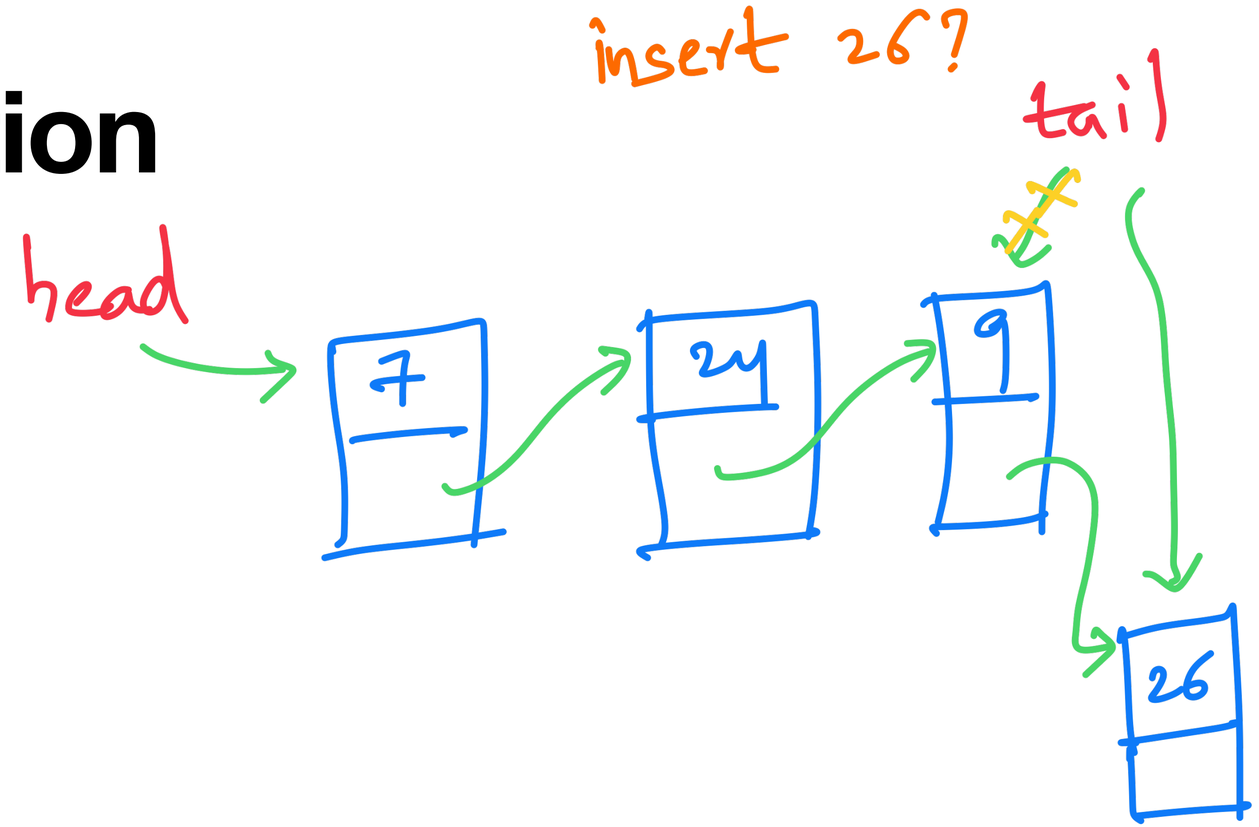
Linked Lists : Insertion

Insert element at end

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

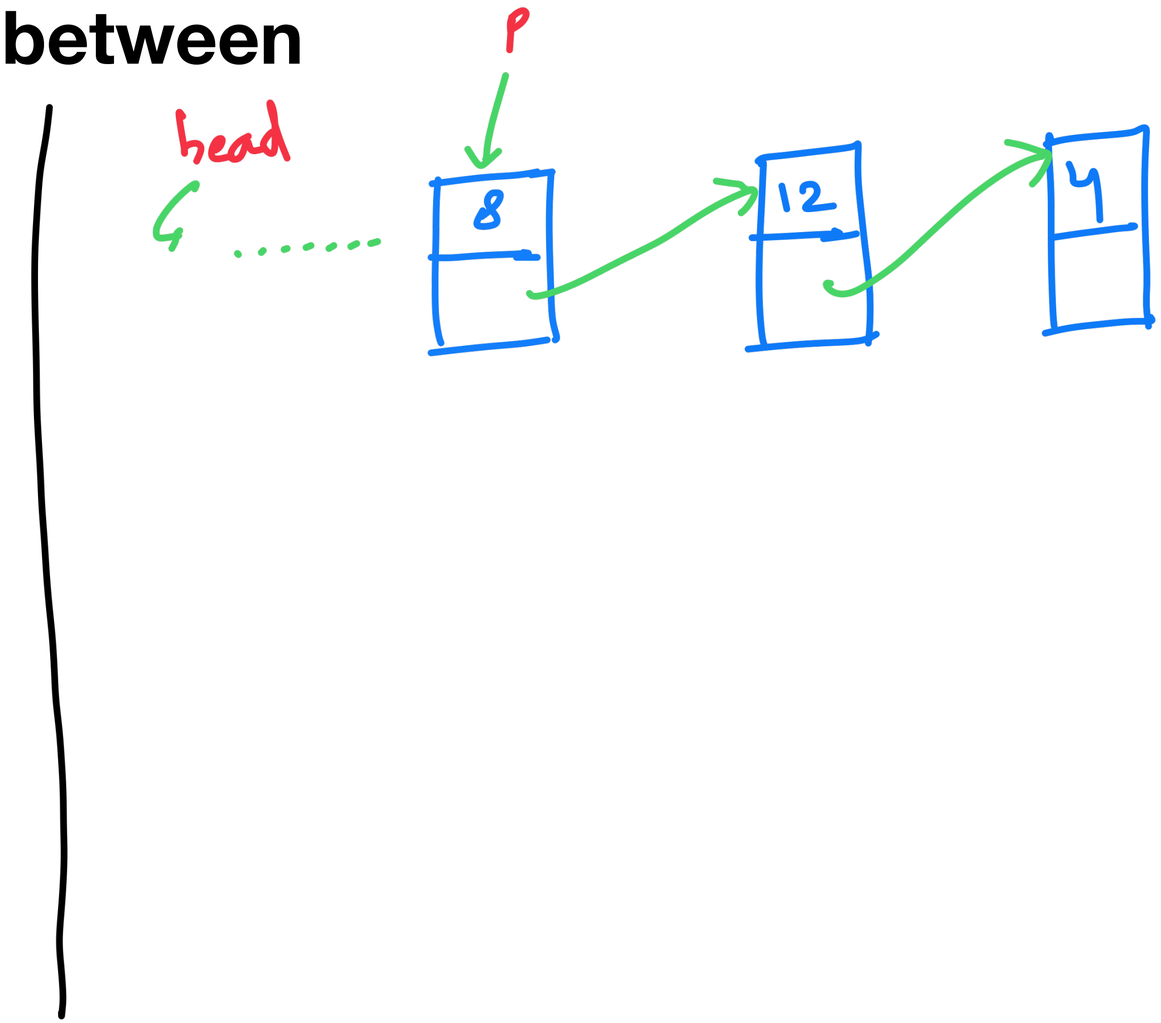
① `tail->next = n;`

② `tail = n;`



Linked Lists : Deletion

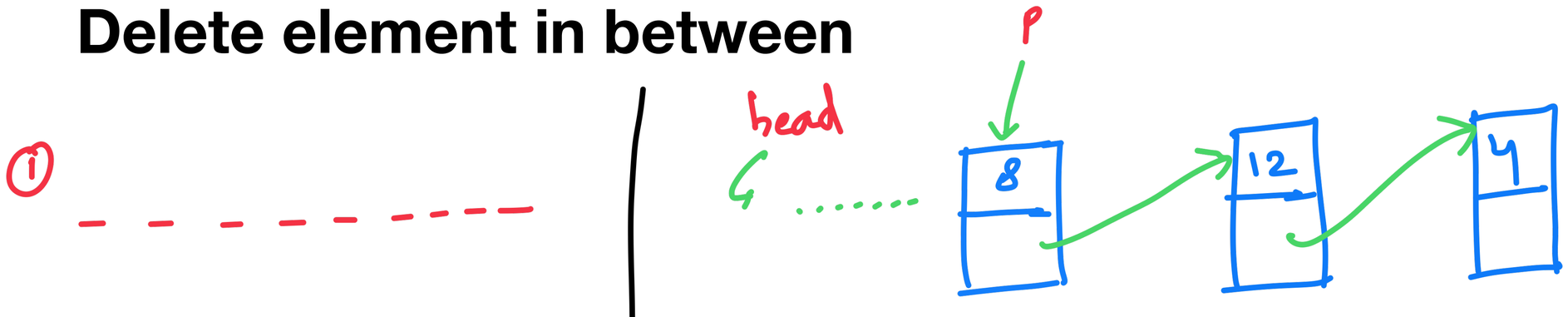
Delete element in between



Linked Lists : Deletion

Delete element in between

delete 12 ?

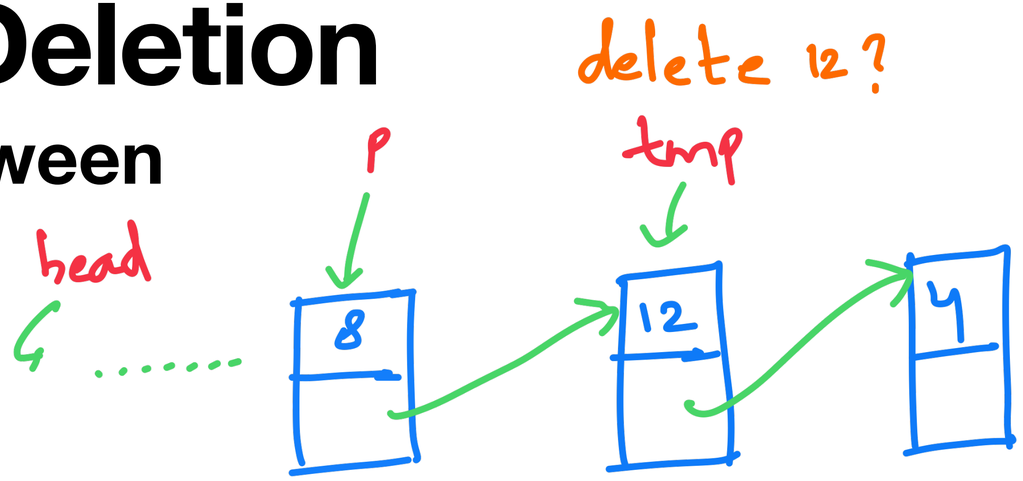


① Get the node to be deleted

Linked Lists : Deletion

Delete element in between

① Node* tmp = P->next;



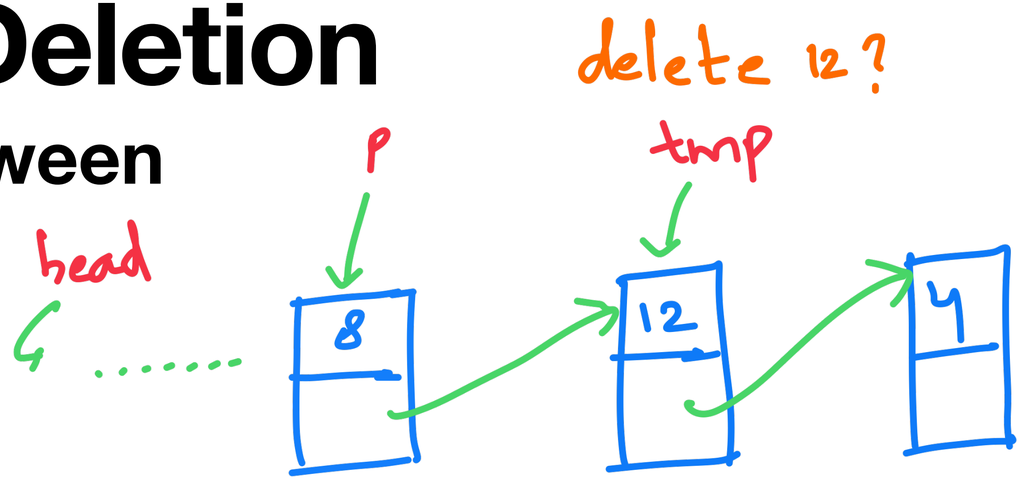
① Get the node to be deleted

Linked Lists : Deletion

Delete element in between

① Node* tmp = p->next;

②



① Get the node to be deleted

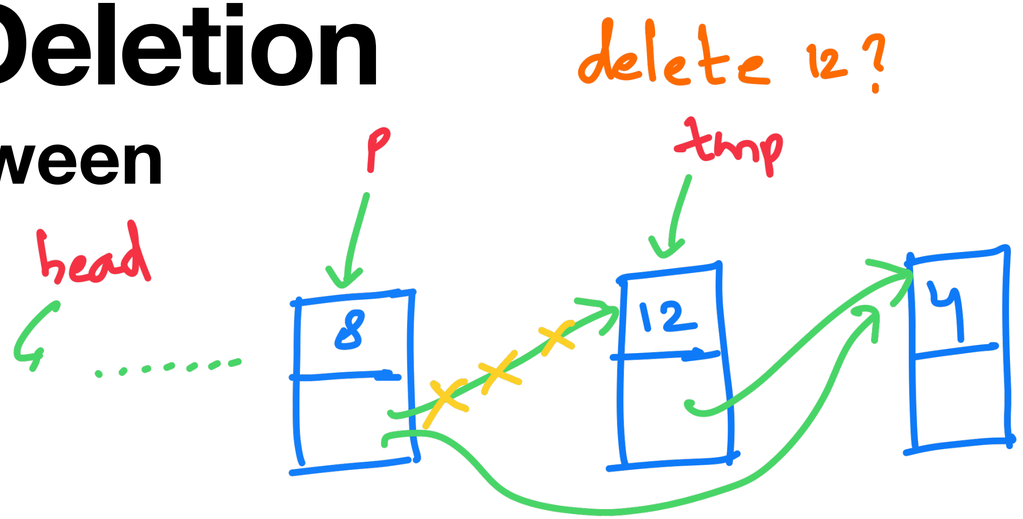
② rewire !

Linked Lists : Deletion

Delete element in between

① Node* tmp = P->next;

②



① Get the node to be deleted

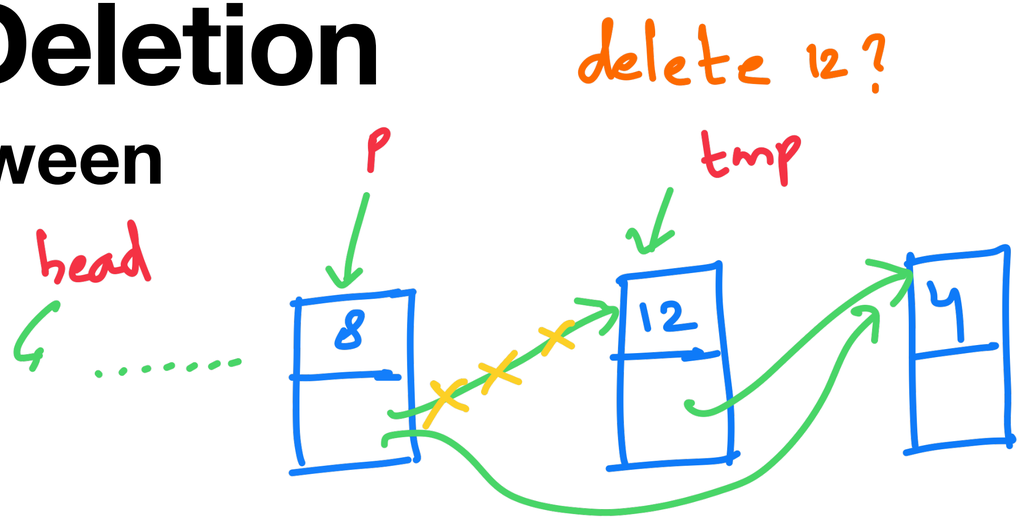
② rewire !

Linked Lists : Deletion

Delete element in between

① `Node* tmp = p->next;`

② `p->next = temp->next;`



① Get the node to be deleted

② rewire !

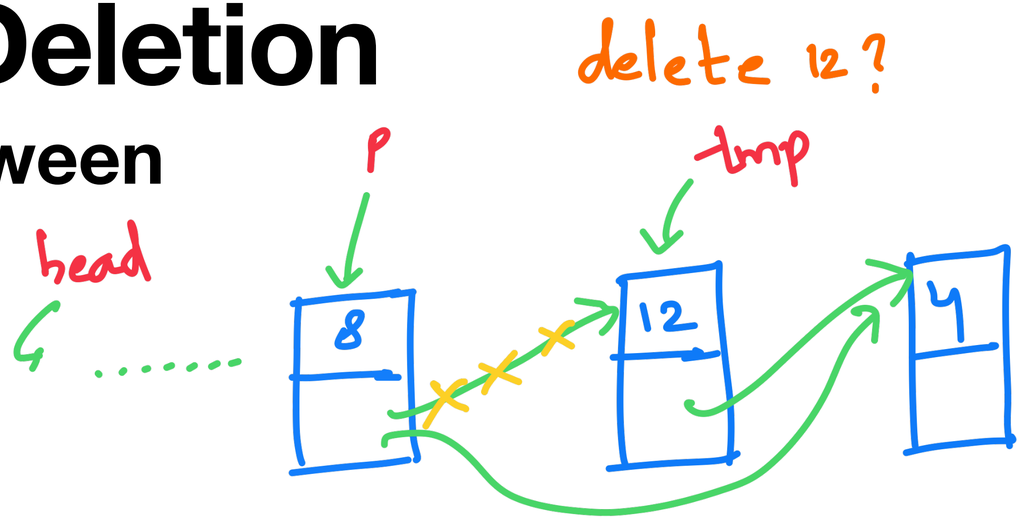
Linked Lists : Deletion

Delete element in between

① `Node* tmp = p->next;`

② `p->next = temp->next;`

③ _ _ _ _ _ _ _



① Get the node to be deleted

② rewire!

③ ------

What's the third step?

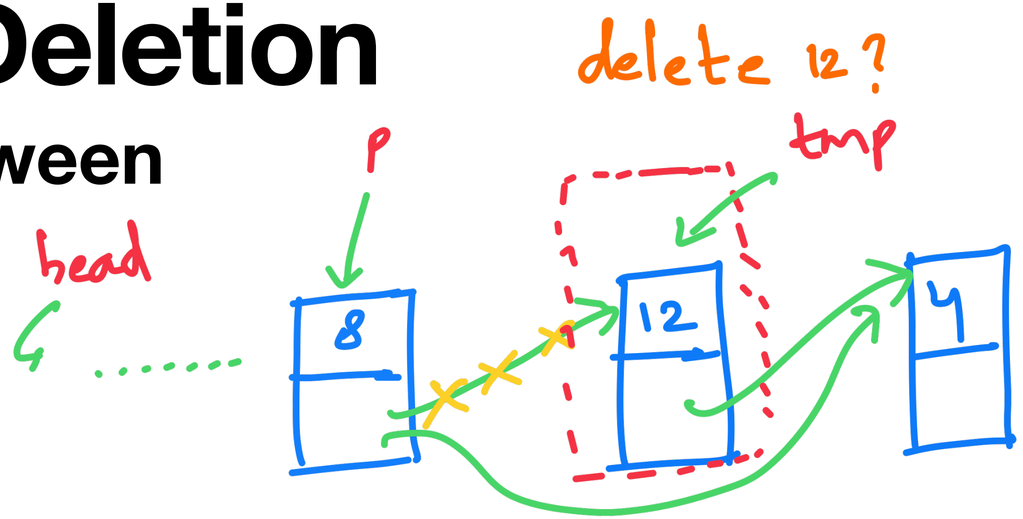
Linked Lists : Deletion

Delete element in between

① `Node* tmp = p->next;`

② `p->next = temp->next;`

③ `delete tmp;`



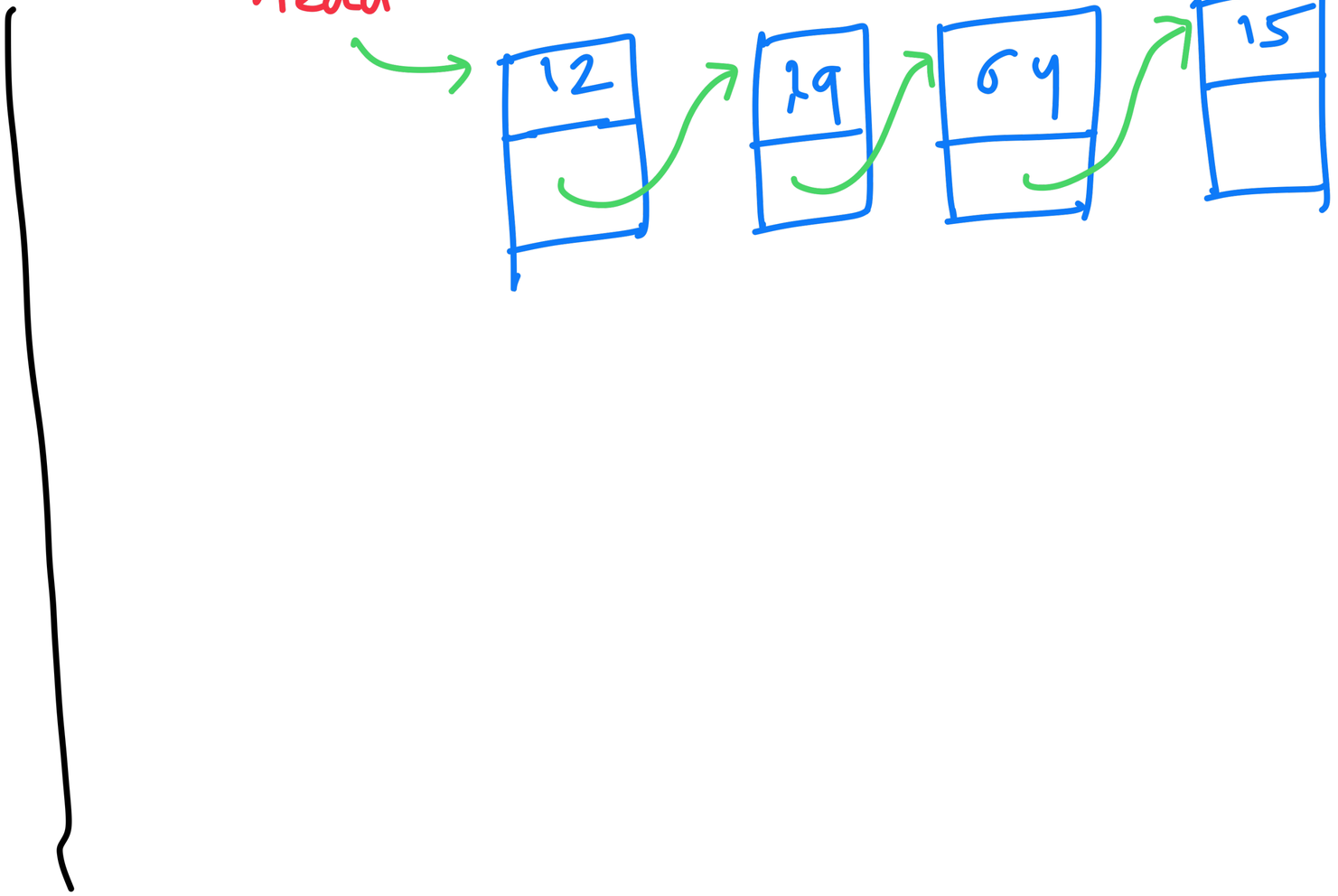
① Get the node to be deleted

② rewire !

③ delete

Linked Lists : Deletion

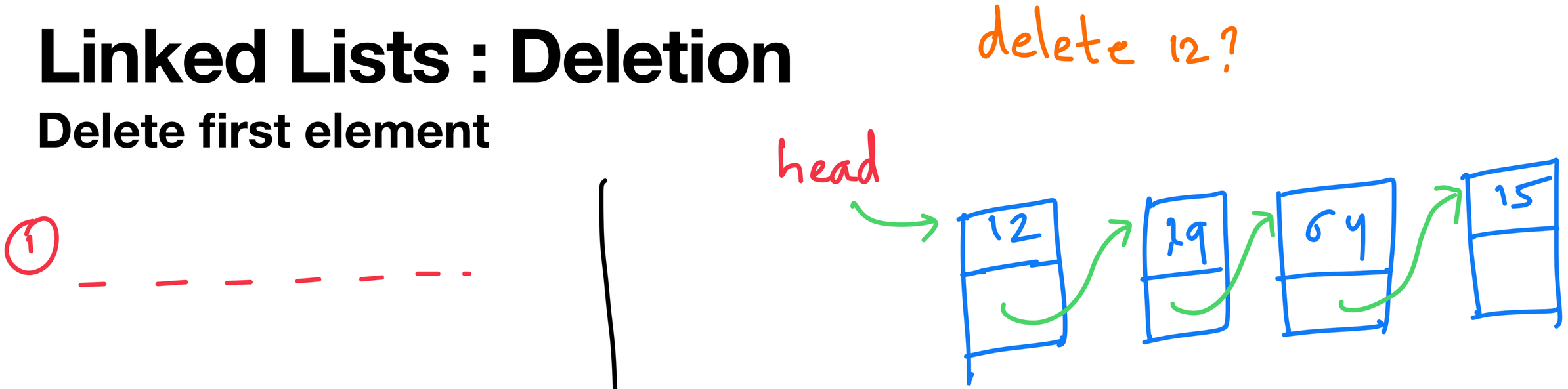
Delete first element



Linked Lists : Deletion

Delete first element

delete 12?



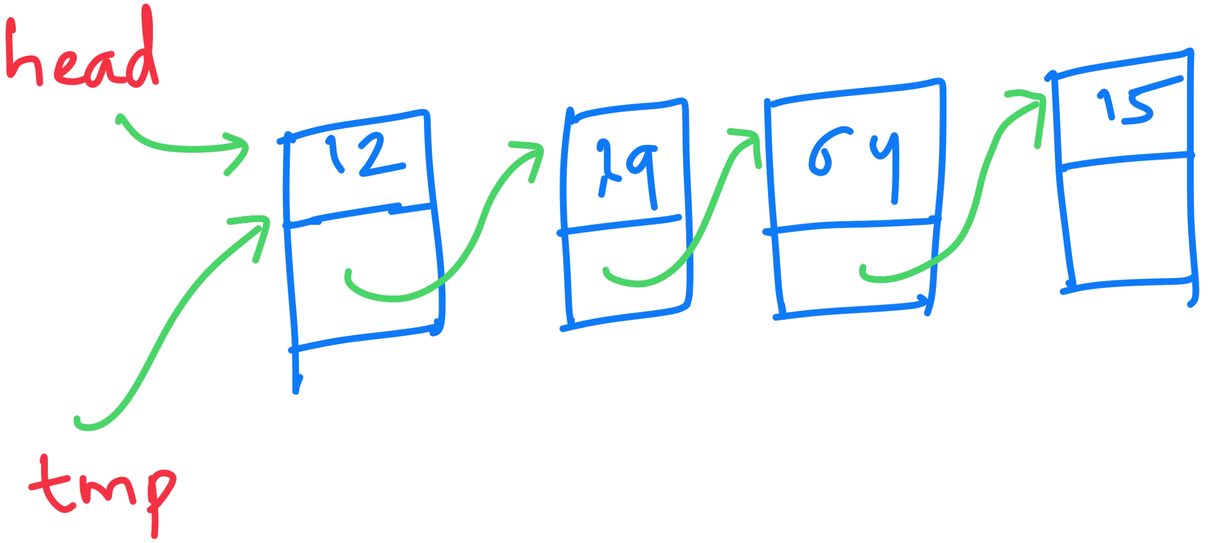
① Get the node to be deleted

Linked Lists : Deletion

Delete first element

delete 12?

① Node* tmp = head;

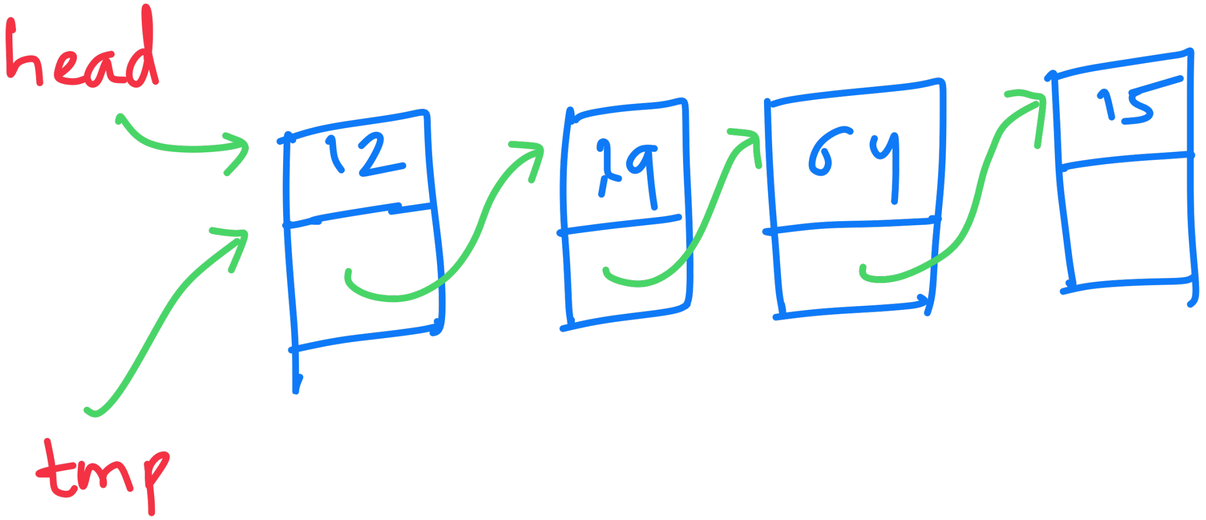


① Get the node to be deleted

Linked Lists : Deletion

Delete first element

delete 12?



① Node* tmp = head;

②

① Get the node to be deleted

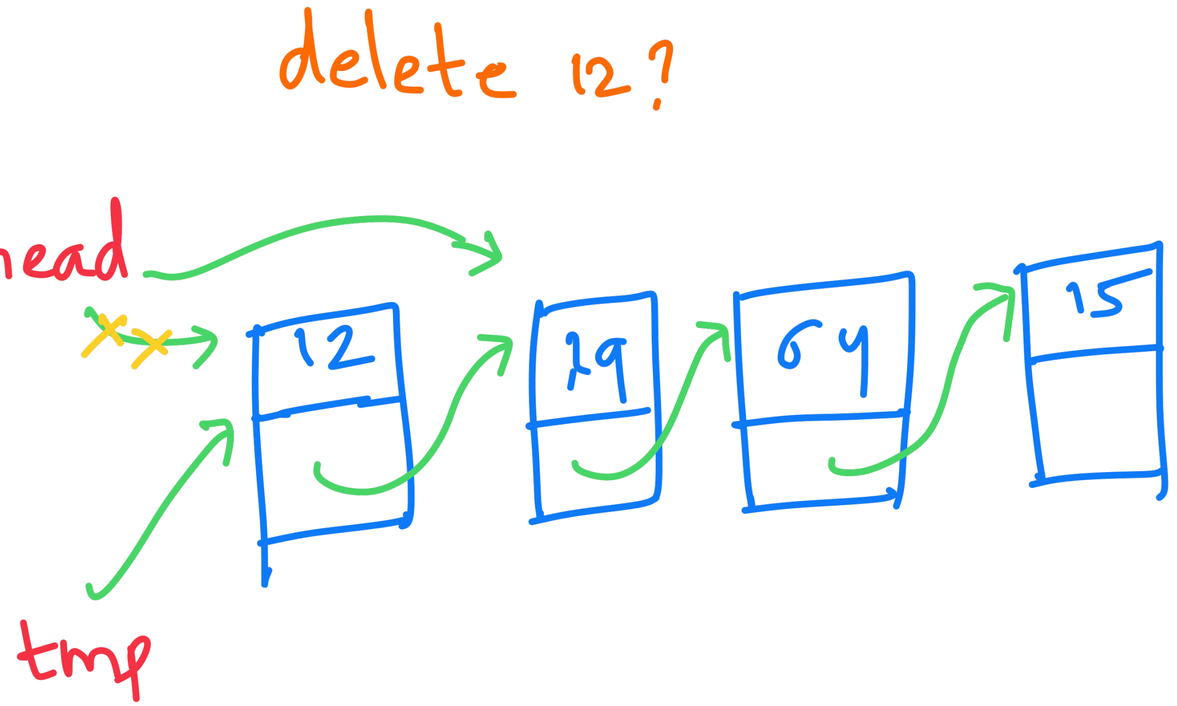
② rewire!

Linked Lists : Deletion

Delete first element

① `Node* tmp = head;`

②



① Get the node to be deleted

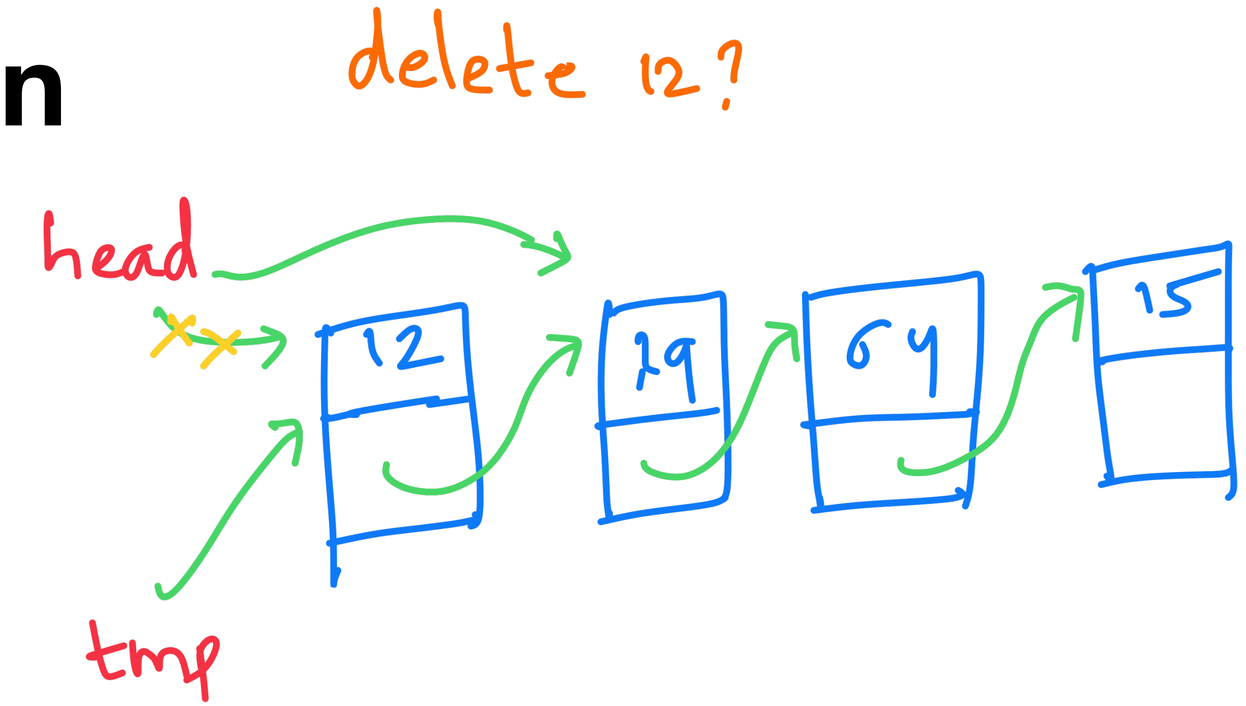
② rewire!

Linked Lists : Deletion

Delete first element

① `Node* tmp = head;`

② `head = tmp->next;`



① Get the node to be deleted

② rewire!

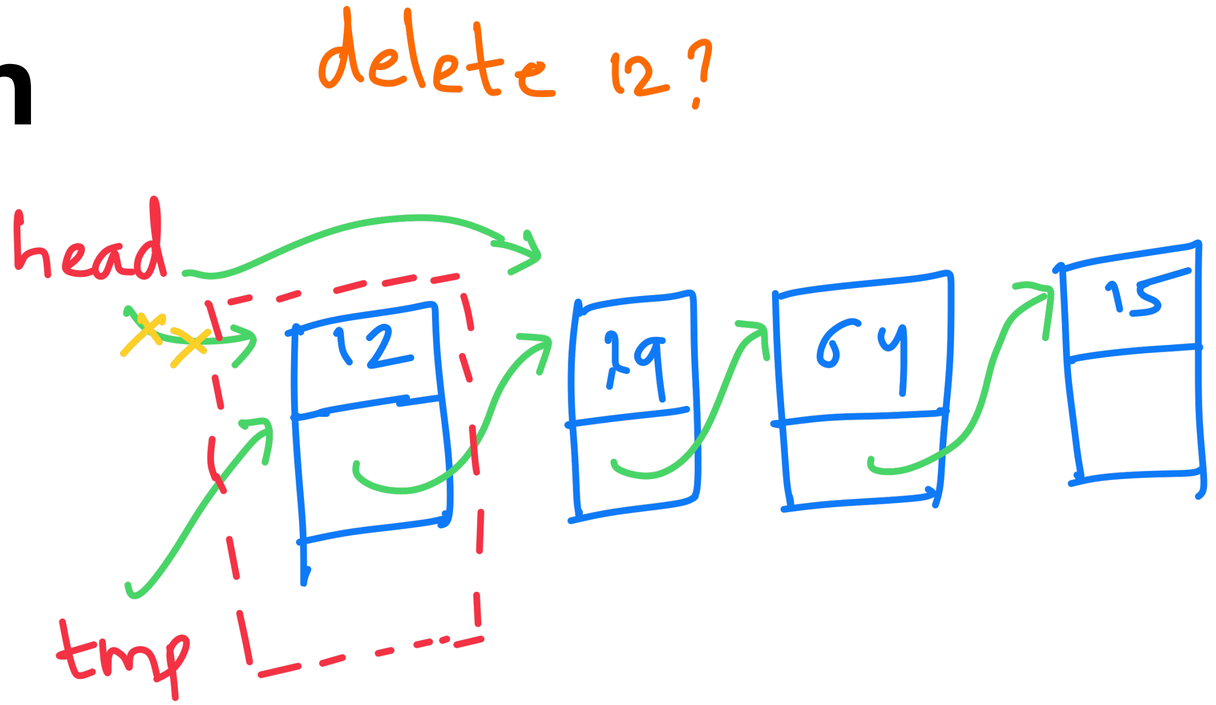
Linked Lists : Deletion

Delete first element

① `Node* tmp = head;`

② `head = tmp->next;`

③ `delete tmp;`



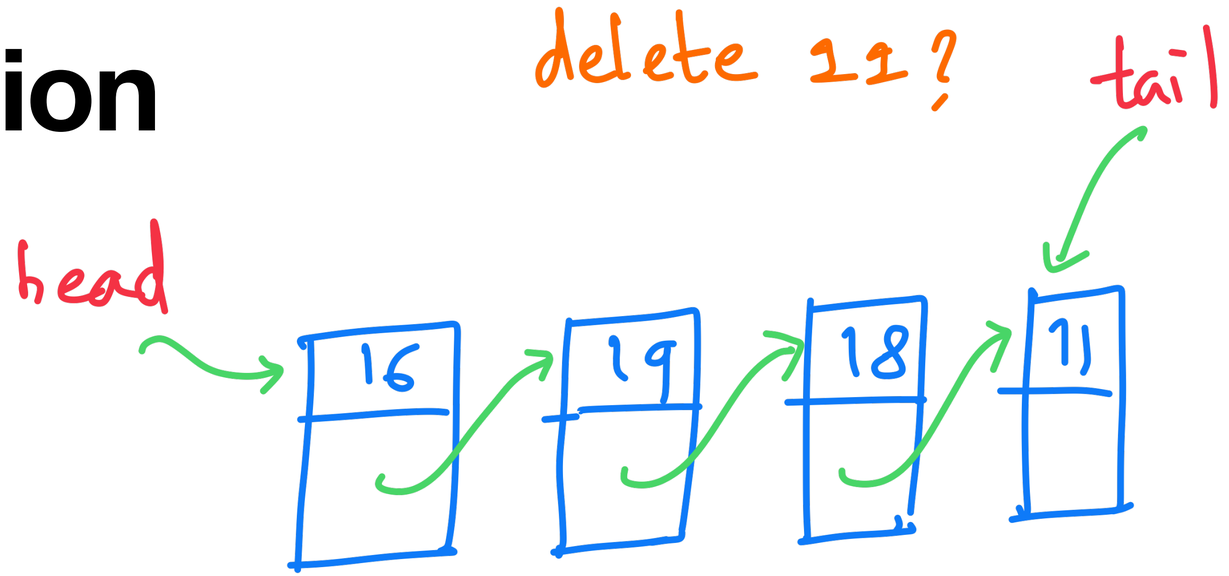
① Get the node to be deleted

② rewire!

③ delete

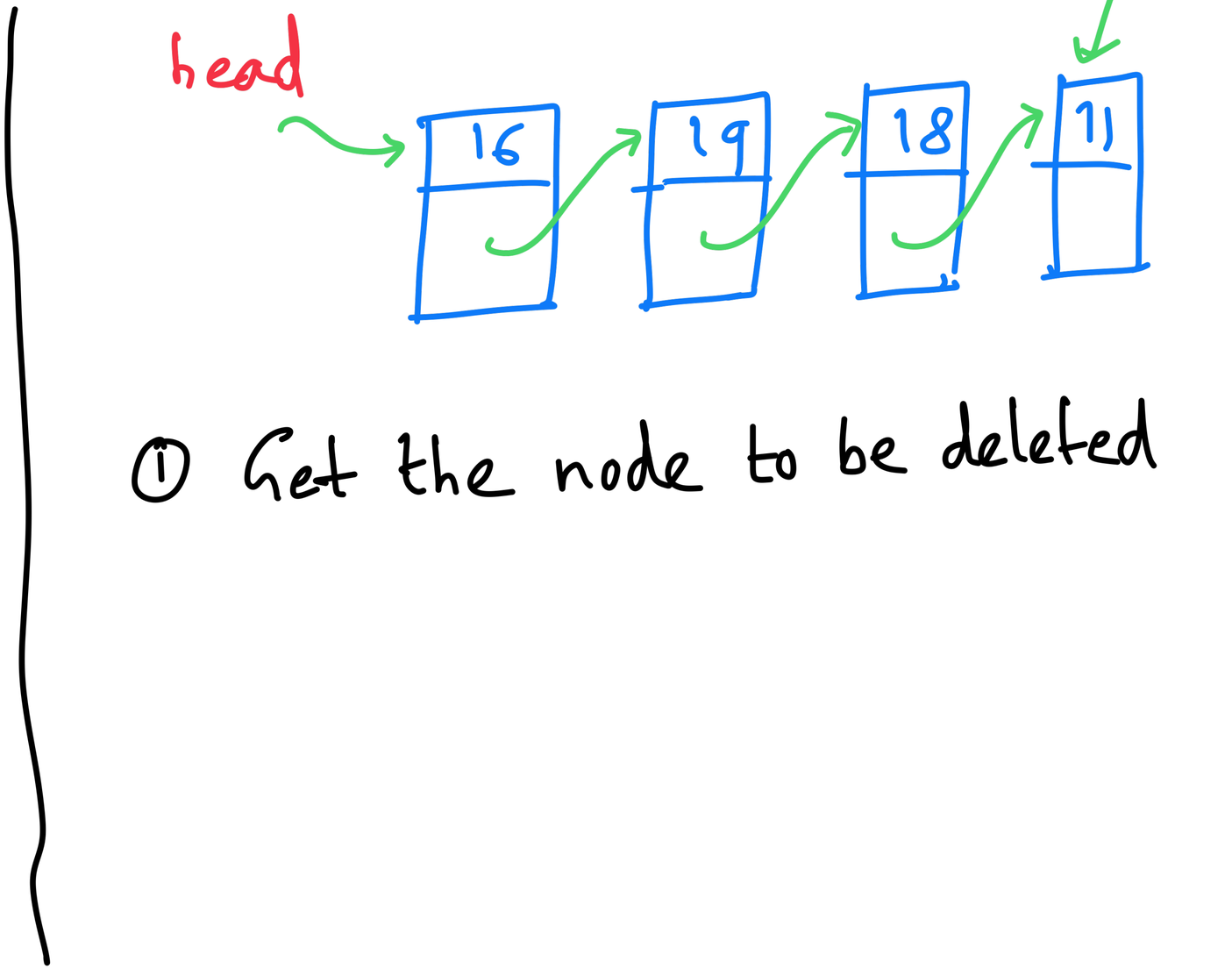
Linked Lists : Deletion

Delete last element



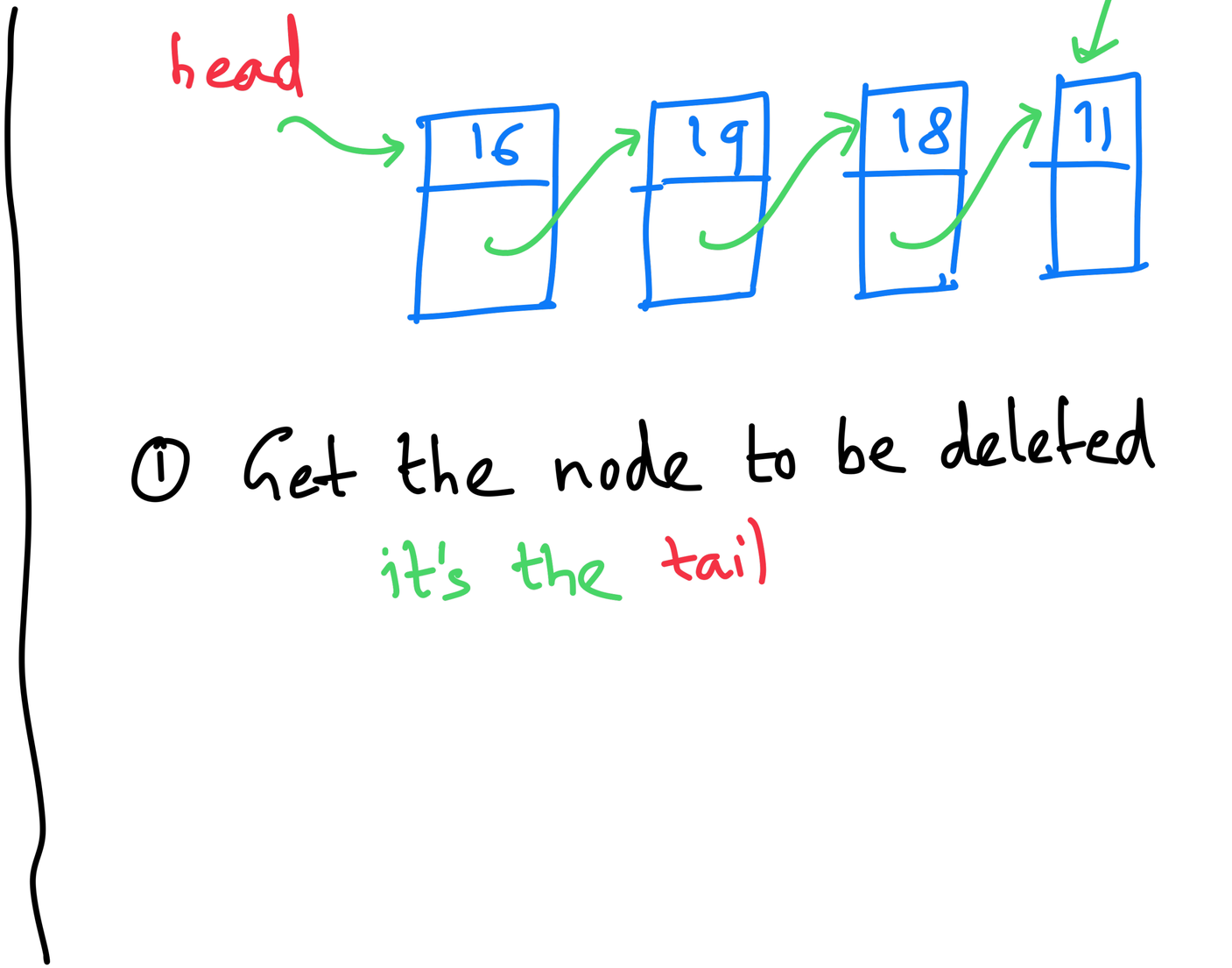
Linked Lists : Deletion

Delete last element



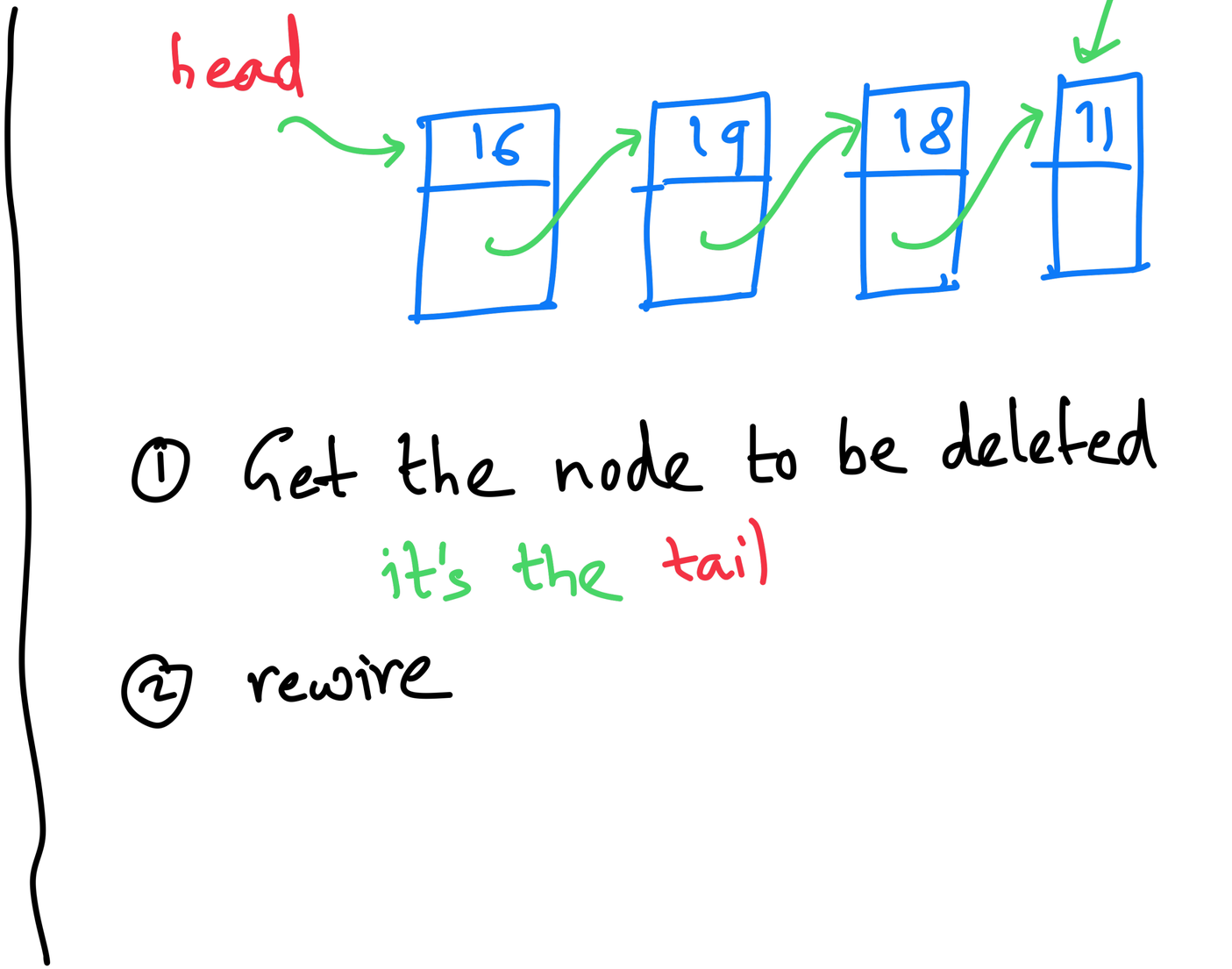
Linked Lists : Deletion

Delete last element



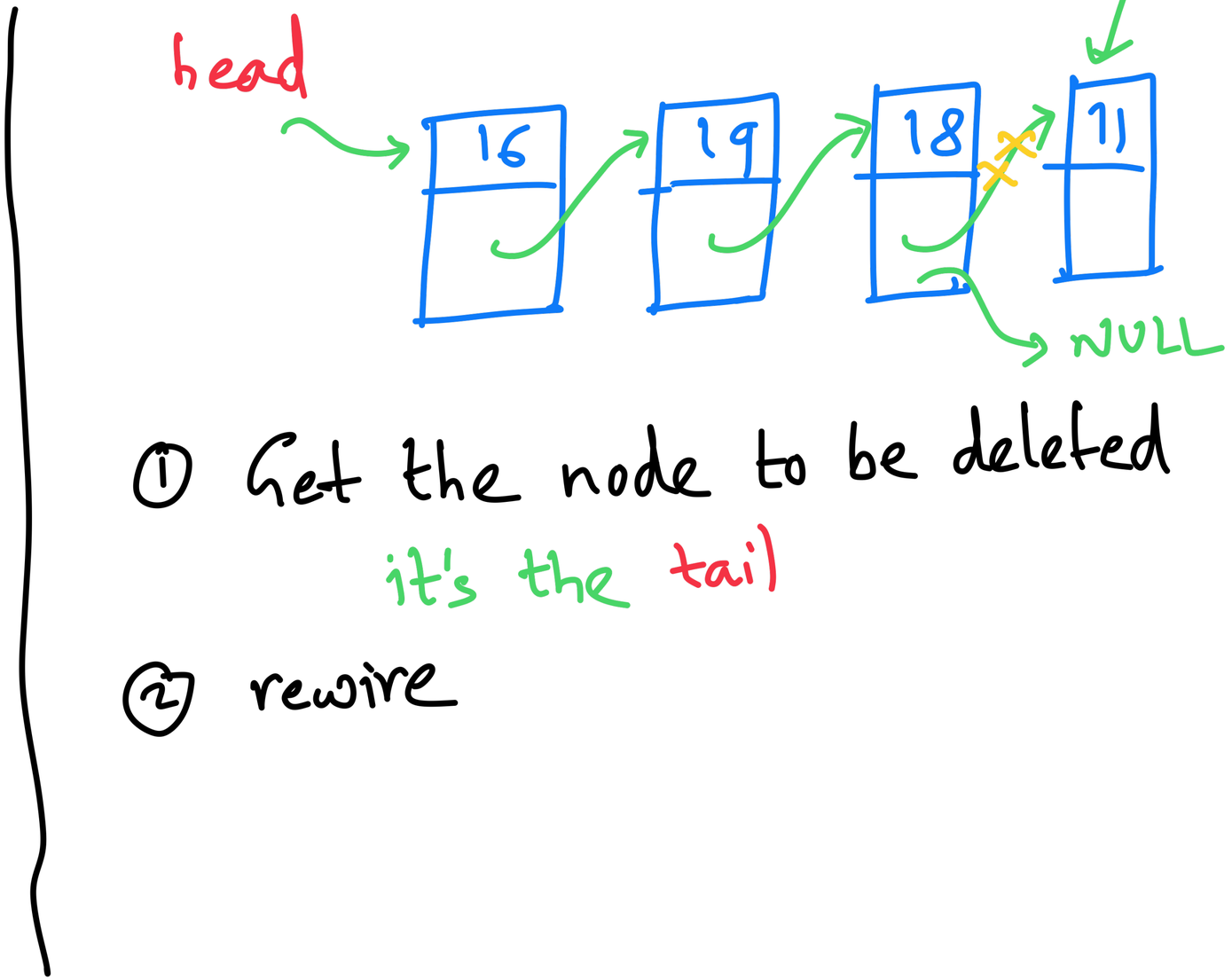
Linked Lists : Deletion

Delete last element



Linked Lists : Deletion

Delete last element



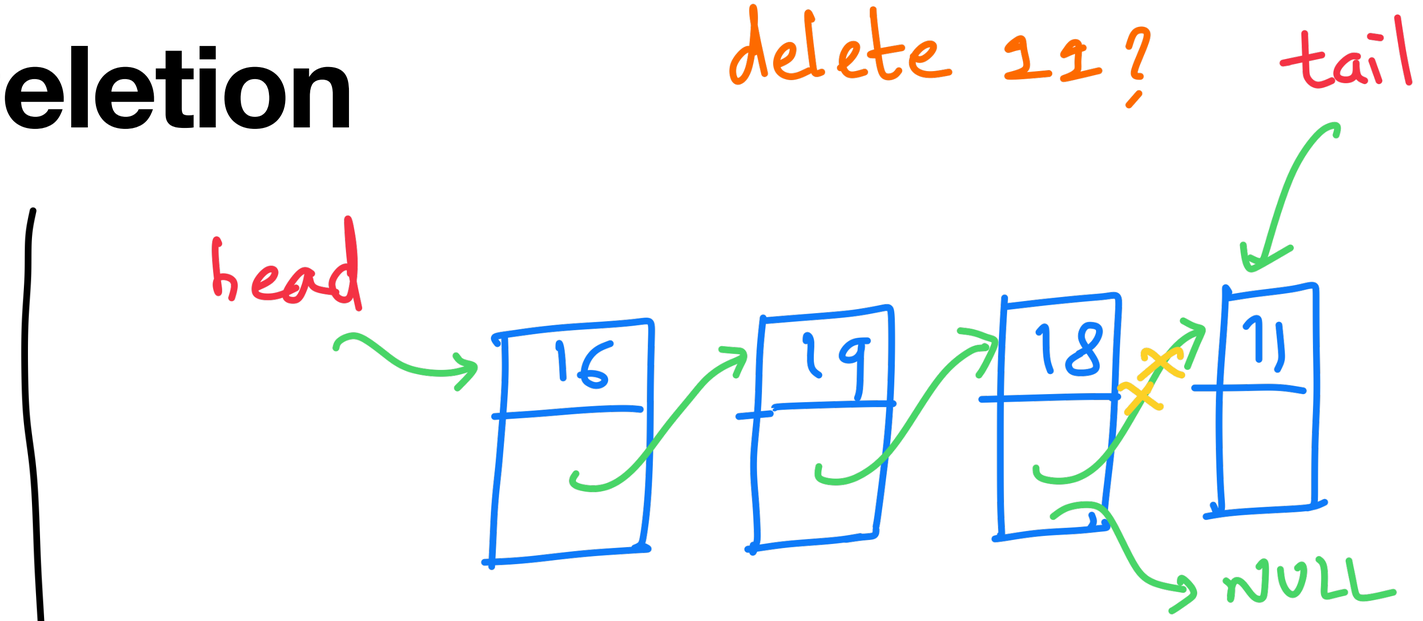
① Get the node to be deleted

it's the tail

② rewire

Linked Lists : Deletion

Delete last element



① Get the node to be deleted

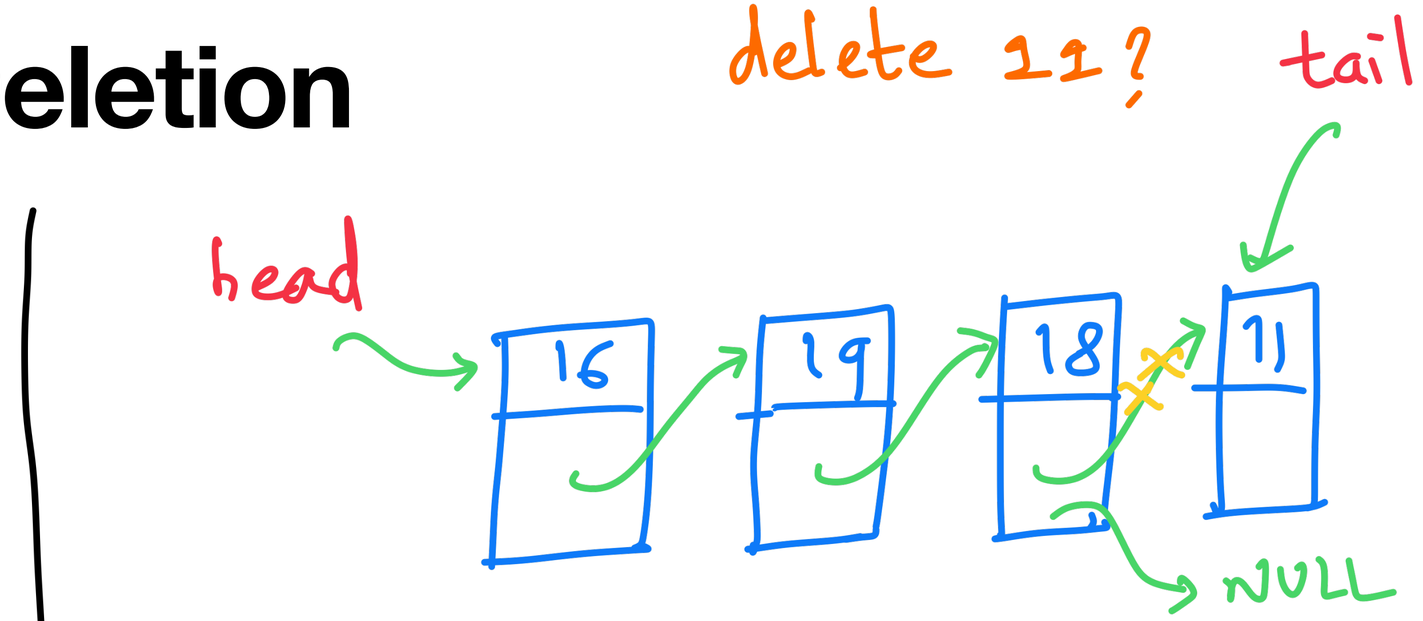
it's the tail

② rewire

but we need node 18

Linked Lists : Deletion

Delete last element



① Get the node to be deleted

it's the tail

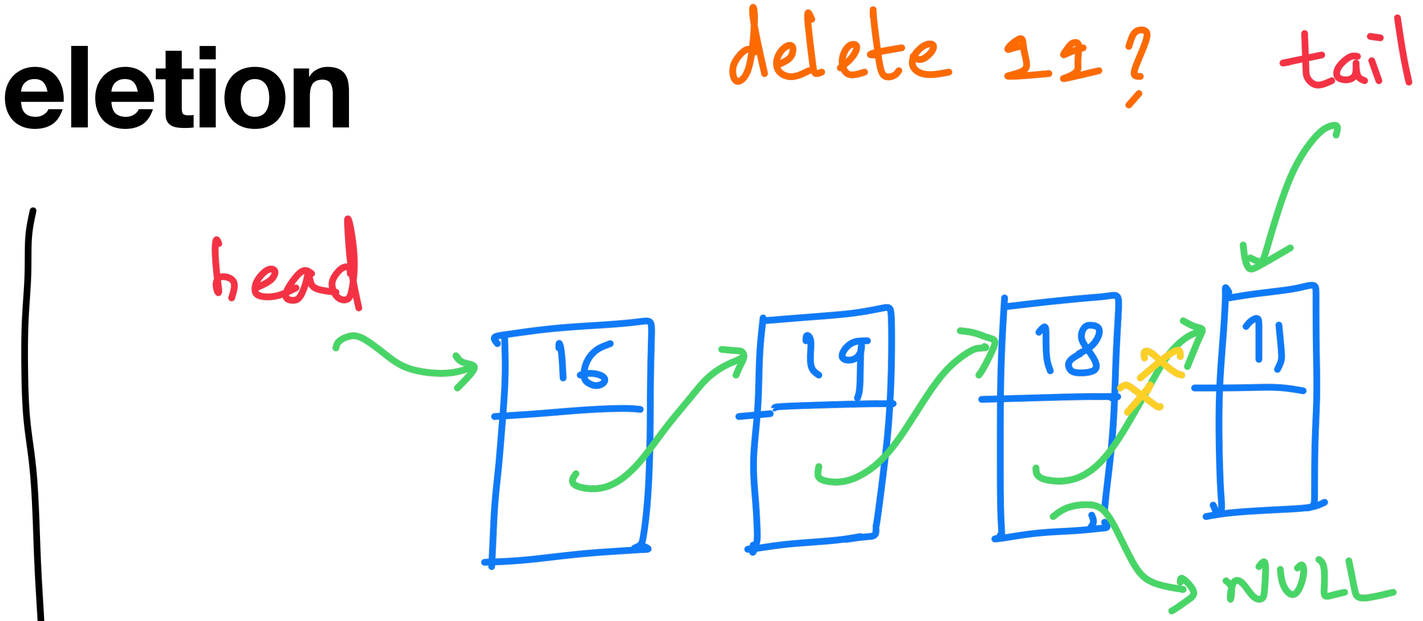
② rewire

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

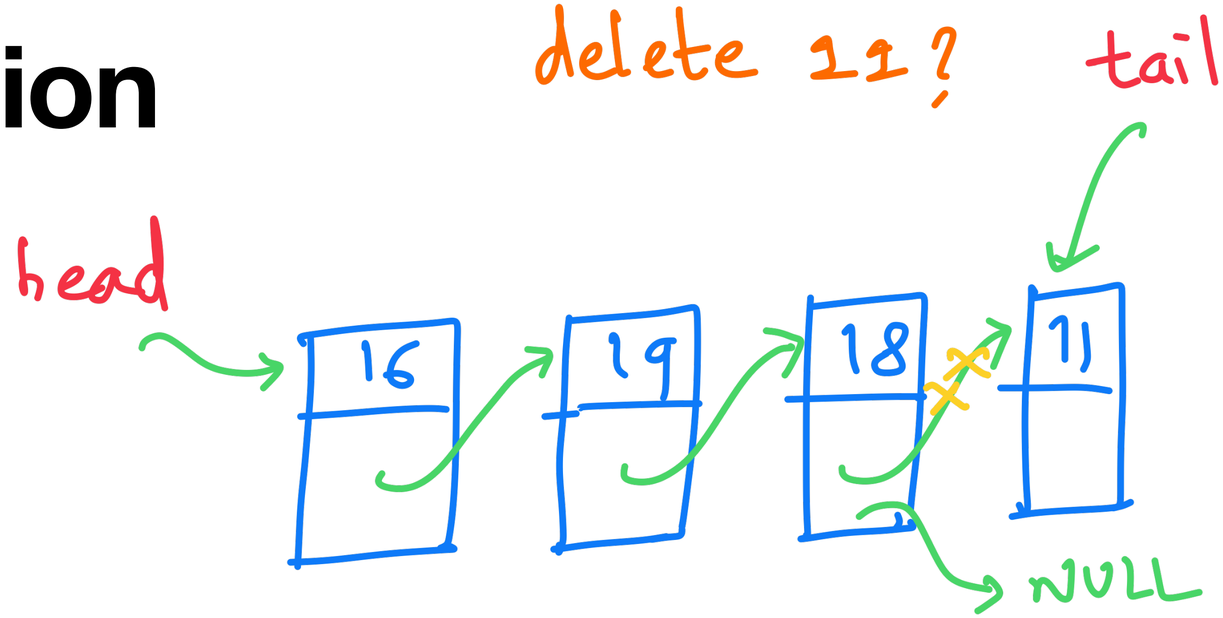
but we need node 18

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

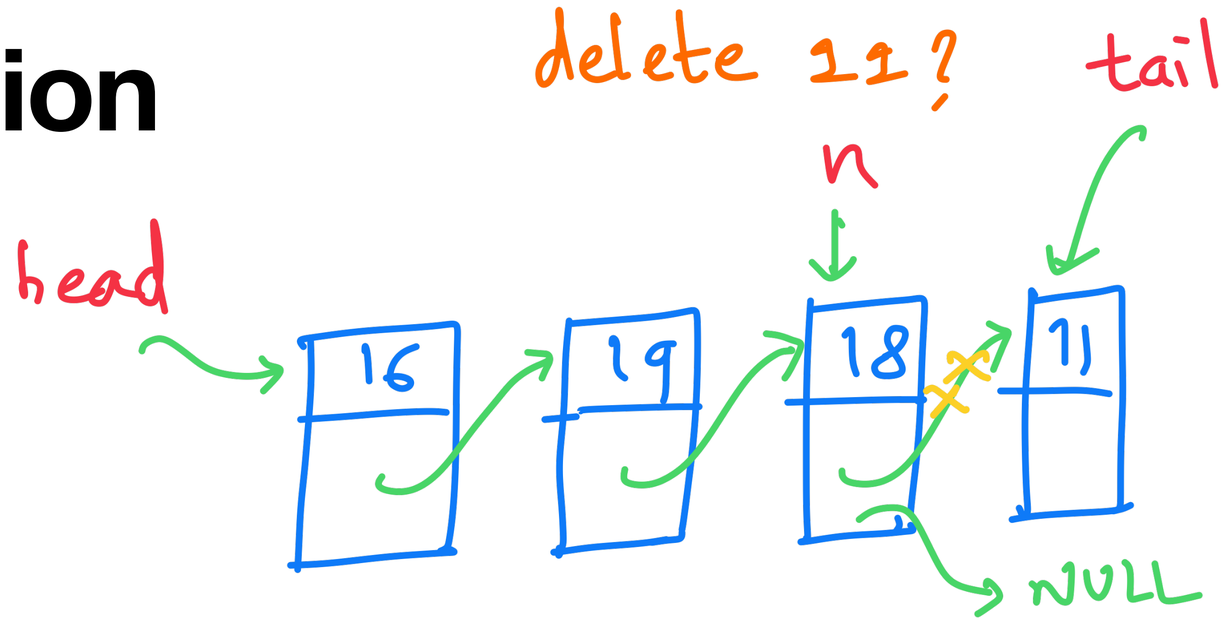
but we need node 18

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

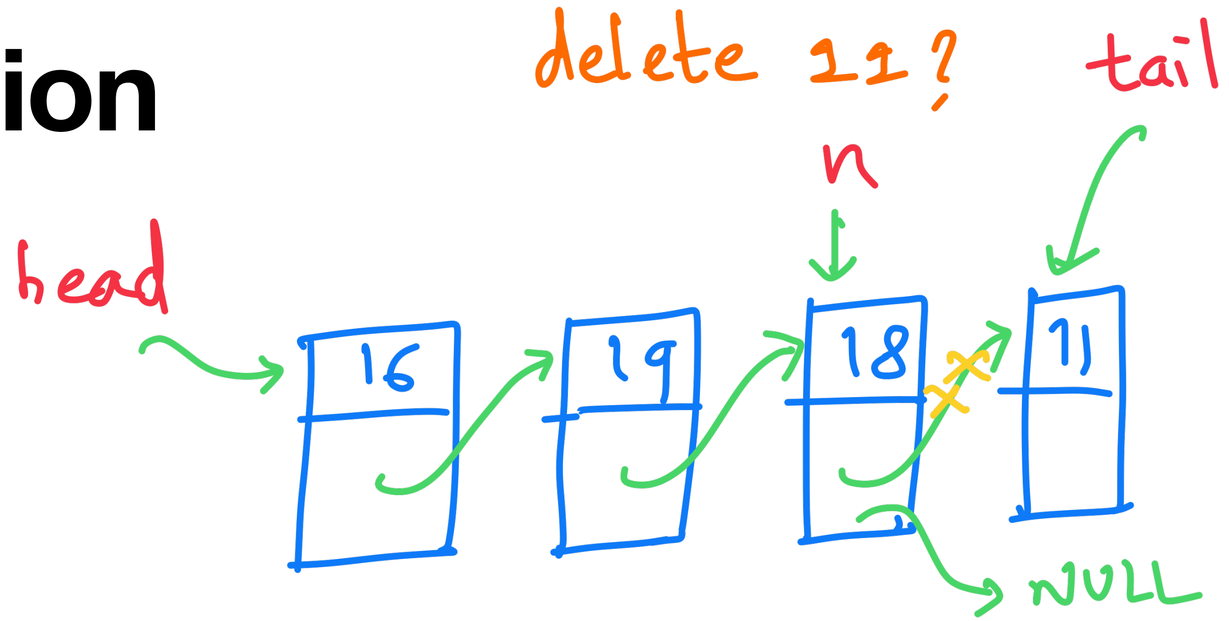
but we need node 18

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

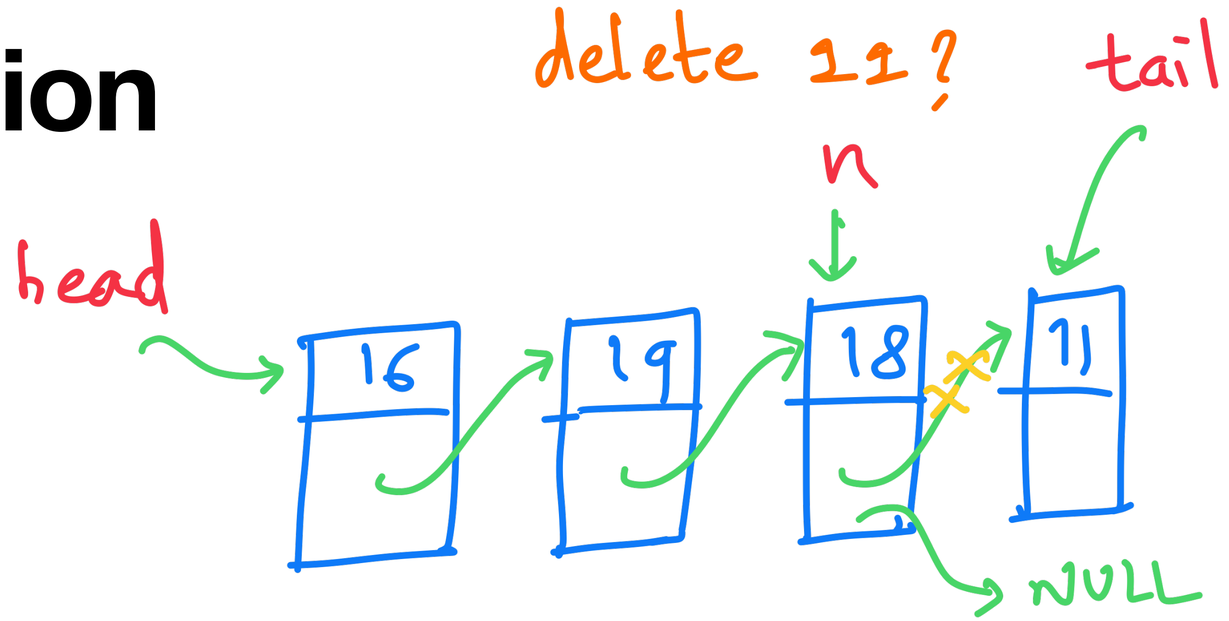
but we need node "18"

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

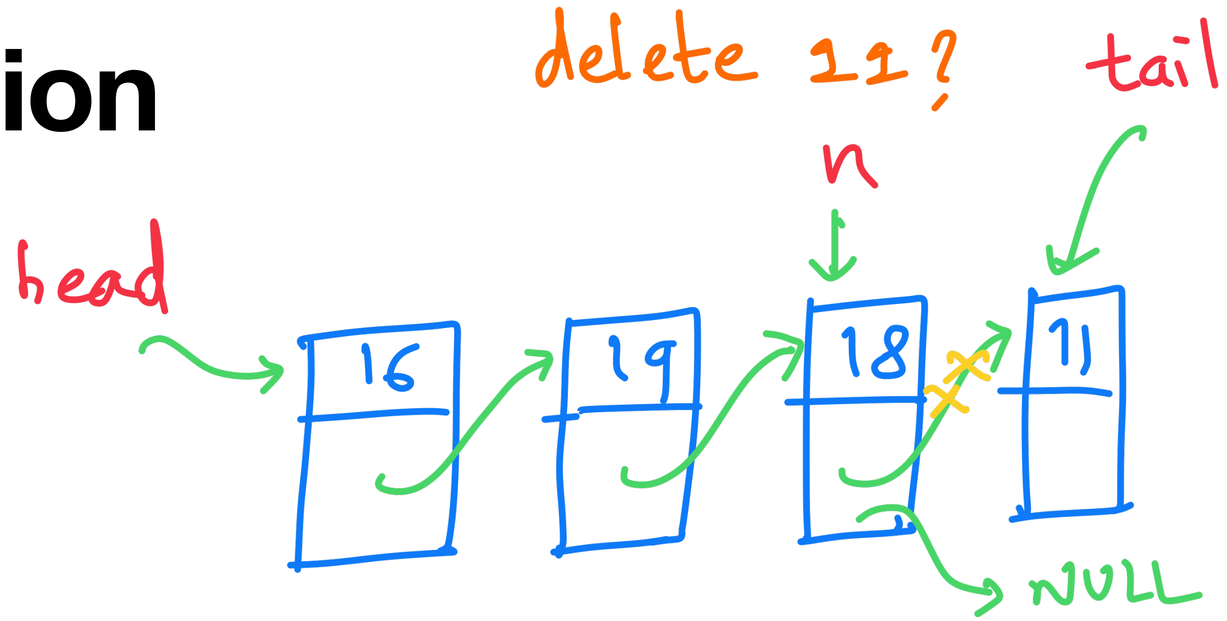
but we need node "18"

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

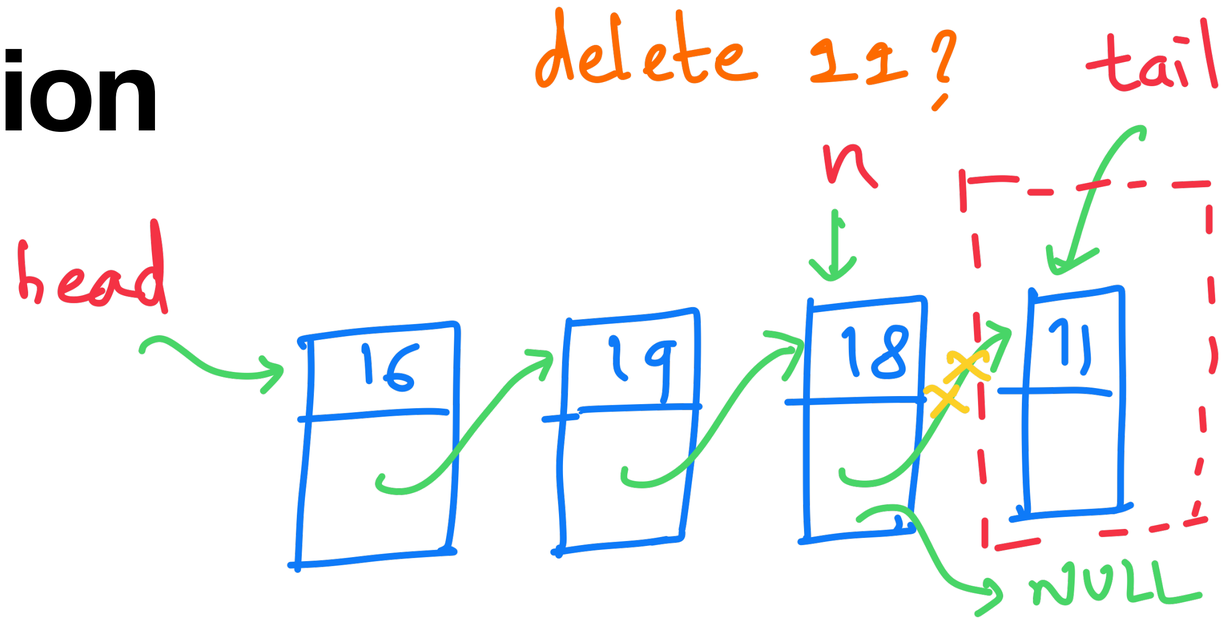
② rewire
   but we need node '18'
   how to get it? - Traversal!

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

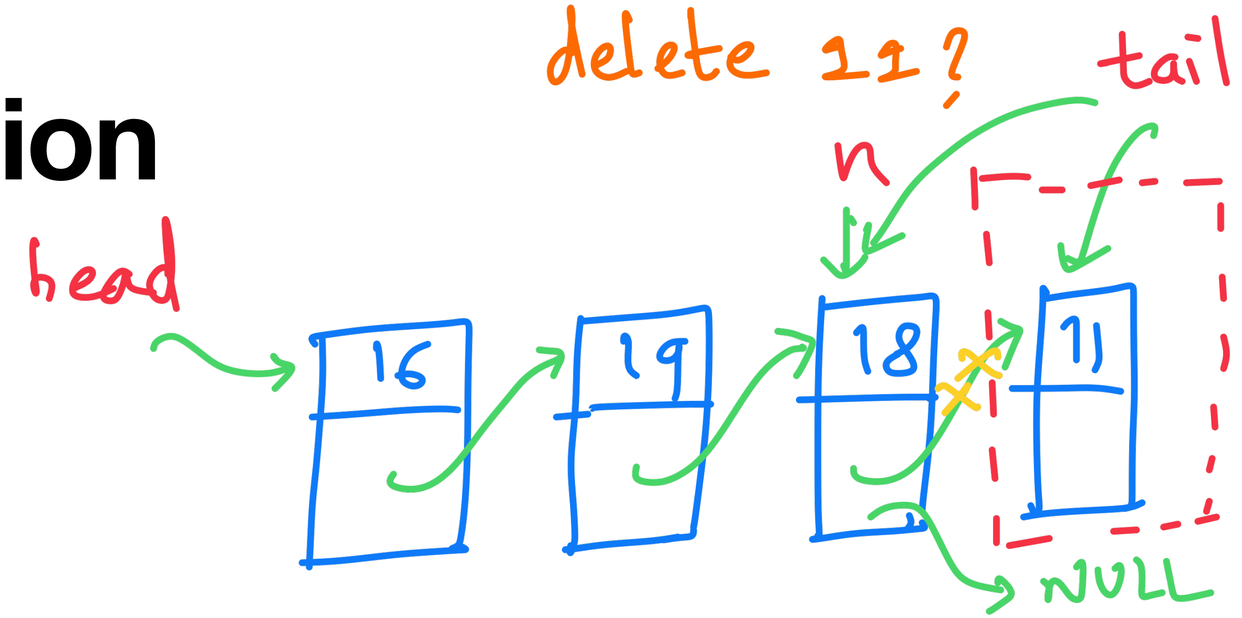
how to get it? - Traversal!

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.

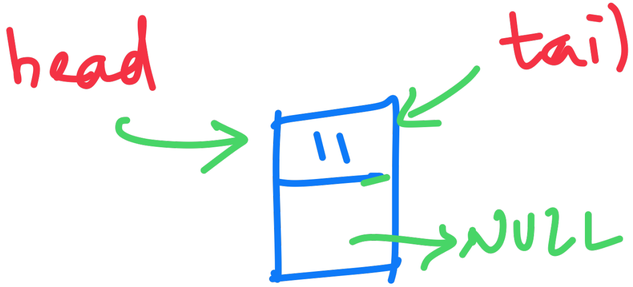
Linked Lists : Deletion

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?

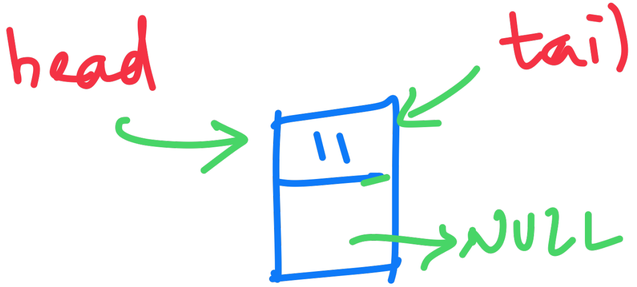
Linked Lists : Deletion

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.

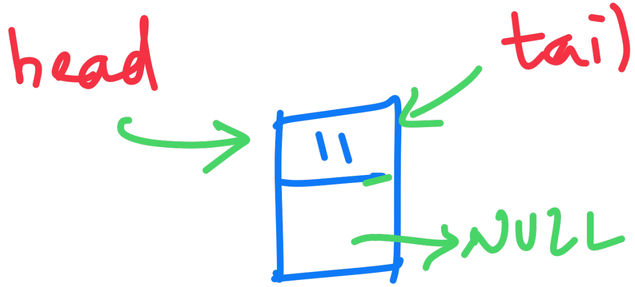
Linked Lists : Deletion

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

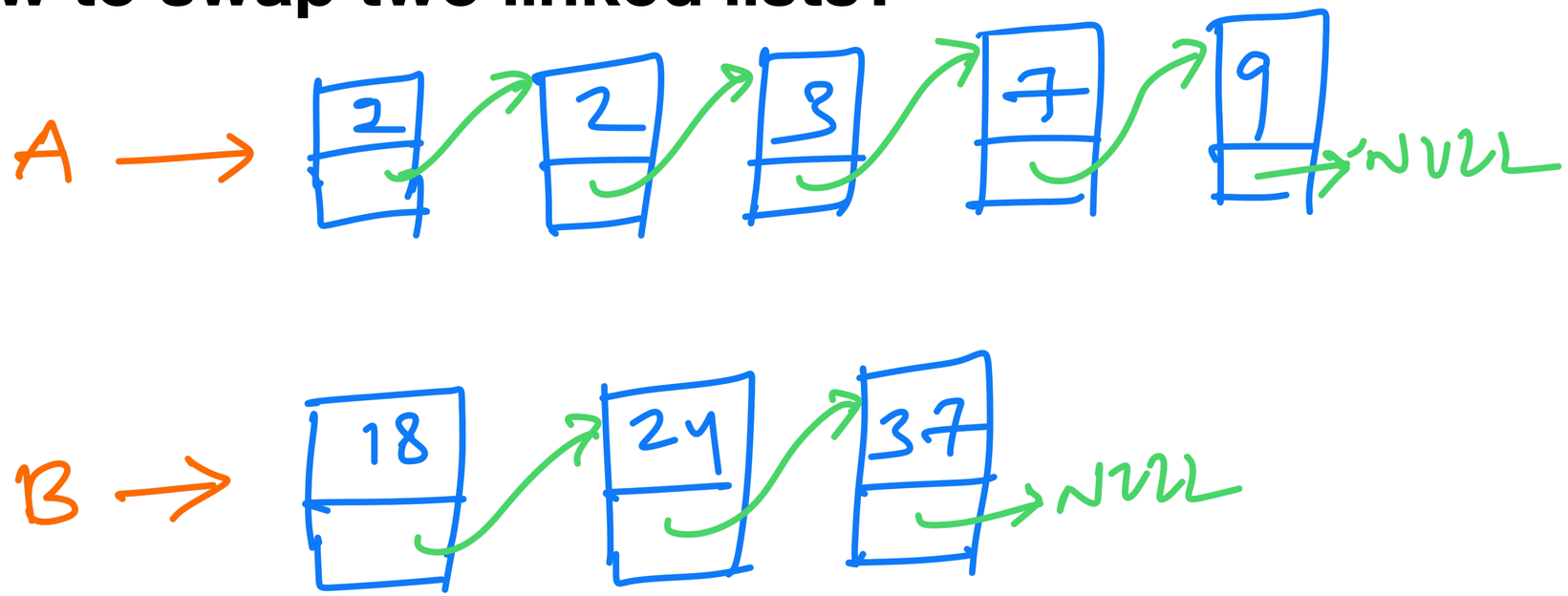
Delete last element

Full code →

```
→ empty list
if (head == NULL) return;
if (head == tail) {          → one element case
    delete tail;
    tail = NULL;
    head = NULL;
    return; }
else {                       → multiple element case.
    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;
}
```

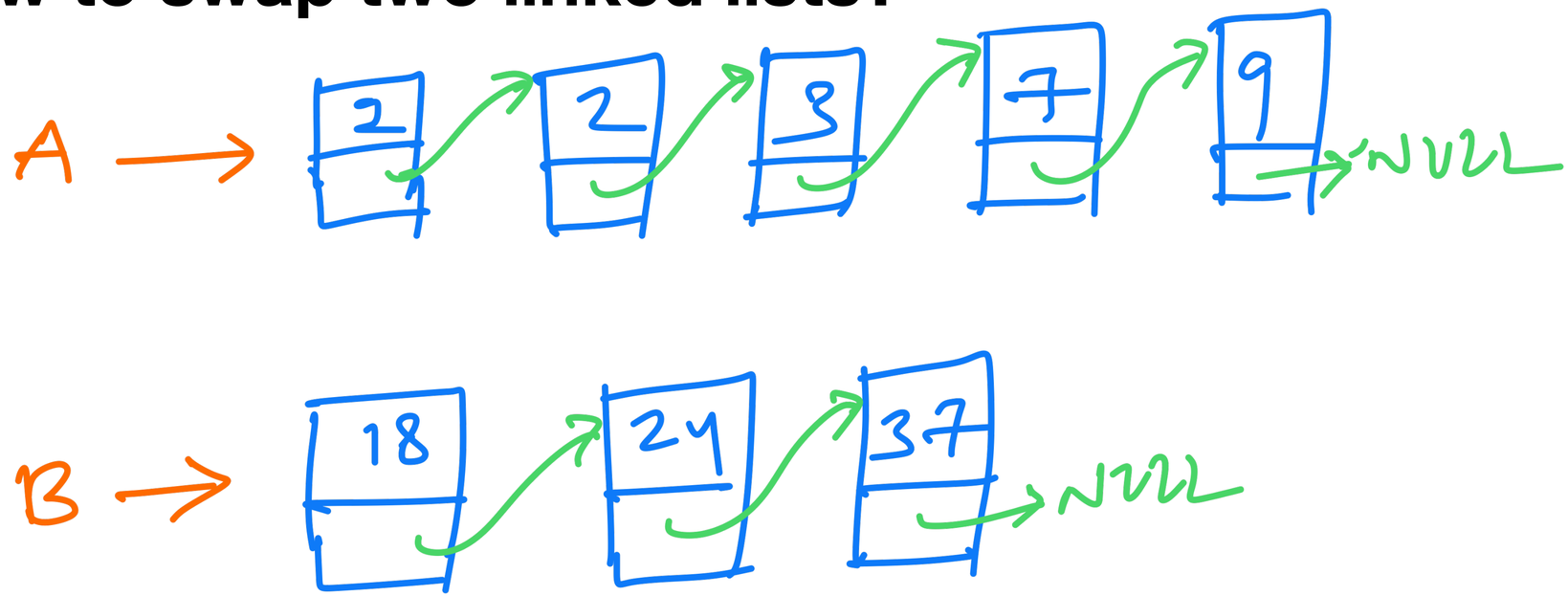
Linked Lists : Problems

How to swap two linked lists?



Linked Lists : Problems

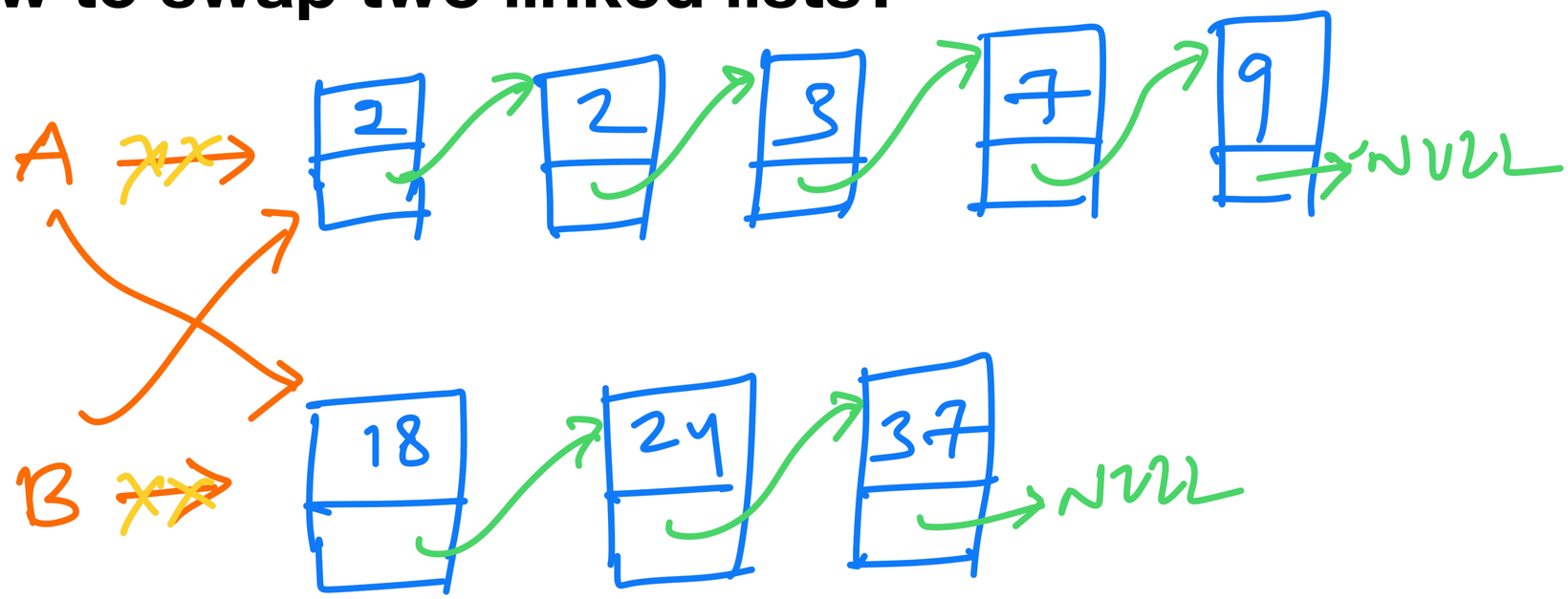
How to swap two linked lists?



Just swap the pointers !!

Linked Lists : Problems

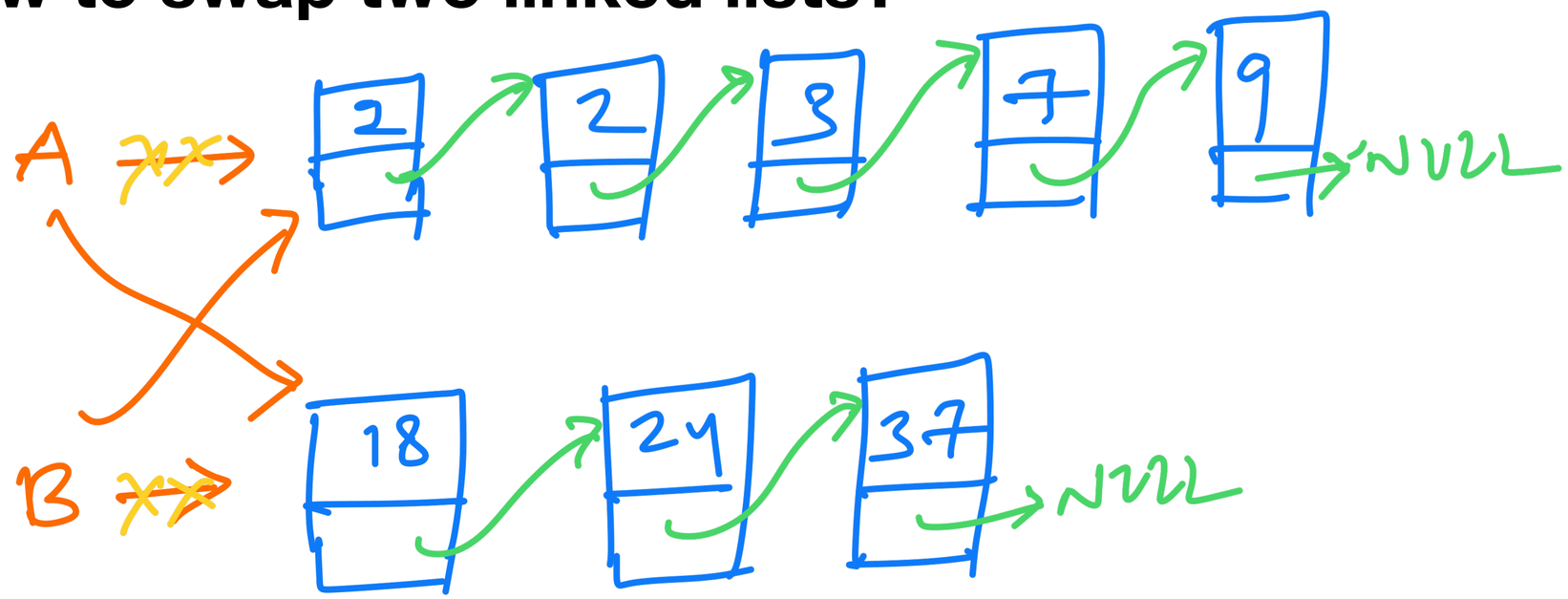
How to swap two linked lists?



Just swap the pointers !!

Linked Lists : Problems

How to swap two linked lists?

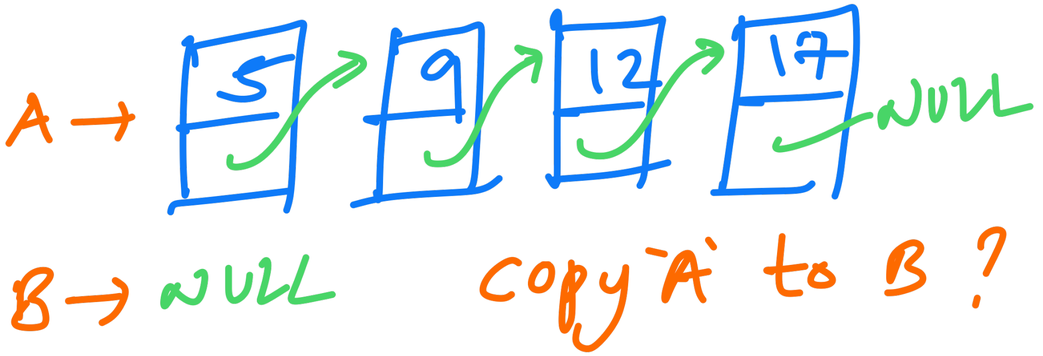



```
Node* tmp = A;
A = B;
B = tmp;
```

Just swap the pointers !!

Linked Lists : Problems

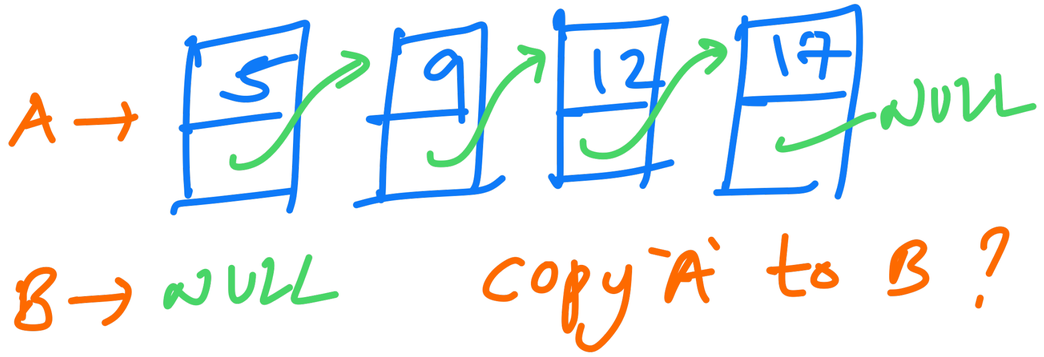
Copy a linked list



B → NULL Copy 'A' to B ?

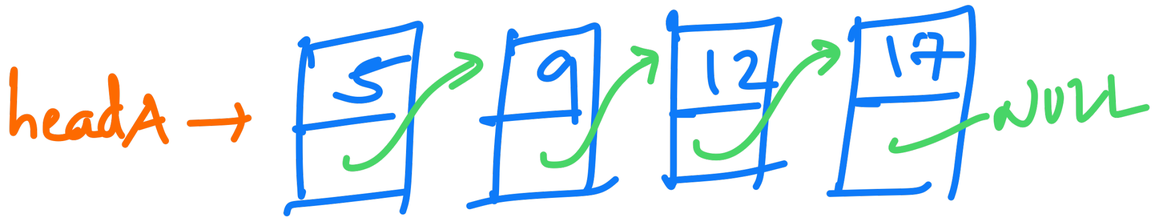
Linked Lists : Problems

Copy a linked list



① Go through each element of A

Linked Lists : Problems



headB → NULL

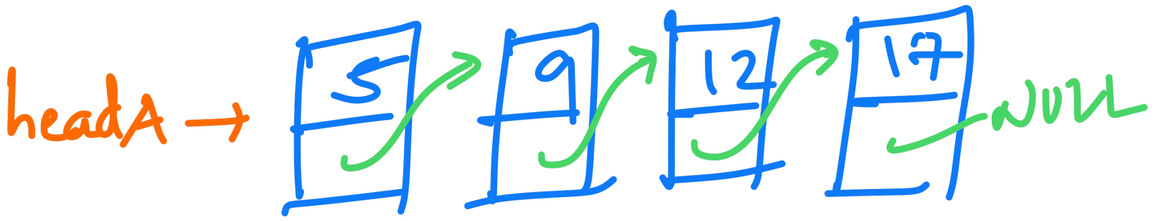
copy 'A' to B ?

Copy a linked list

① Go through each element of A
→ Traversal

```
Node* tmpA = headA;

while(tmpA != NULL){

tmpA = tmpA->next;

}
```

Linked Lists : Problems



headB → NULL

copy 'A' to B ?

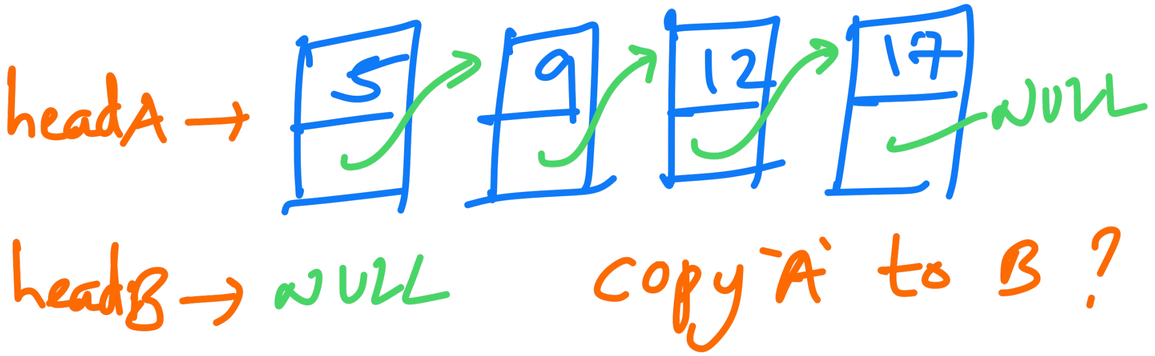
Copy a linked list

```
Node* tmpA = headA;

while(tmpA != NULL){

    tmpA = tmpA->next;

}
```

① Go through each element of A → Traversal

② create copy of the element

Linked Lists : Problems



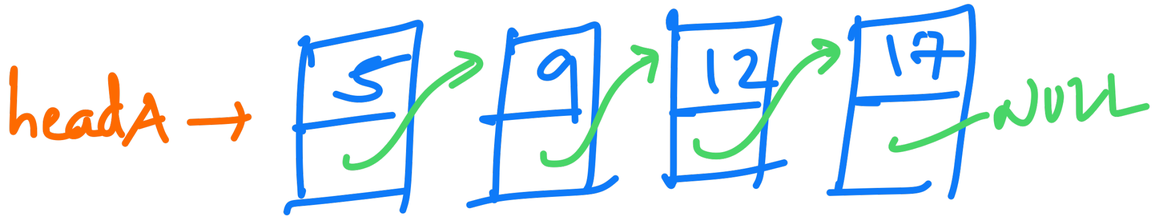
Copy a linked list

```
Node* tmpA = headA;

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

    tmpA = tmpA->next;

}
```

① Go through each element of A
→ Traversal

② create copy of the element

Linked Lists : Problems



headB → NULL copy 'A' to B ?

Copy a linked list

```
Node* tmpA = headA;

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

    tmpA = tmpA->next;
}
```

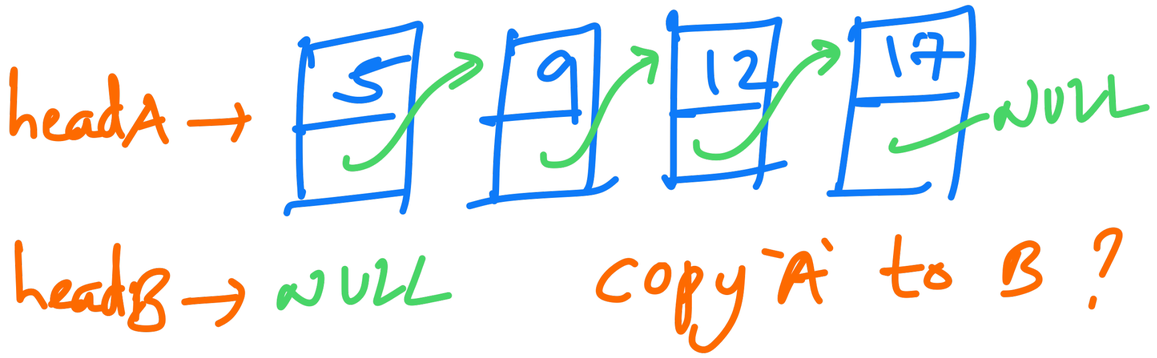
① Go through each element of A
→ Traversal

② create copy of the element

③ Add copy to 'B' list.

Linked Lists : Problems

Copy a linked list

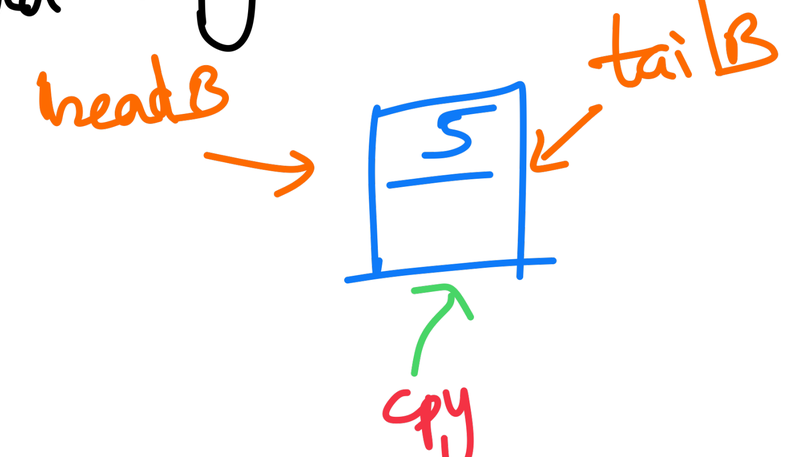



```
Node* tmpA = headA;

while(tmpA != NULL){
    Node* cpy = new Node;
    cpy->m_val = tmpA->m_val;
    if(headB == NULL){
        headB = cpy;
        tailB = cpy; }
    ⋮
```

① Go through each element of A
→ Traversal

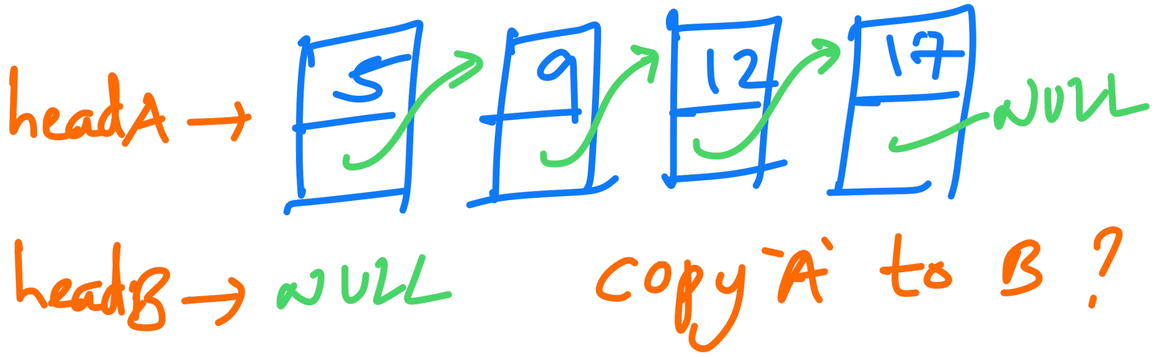
② create copy of the element

③ Add copy to 'B' list.



Linked Lists : Problems

Copy a linked list

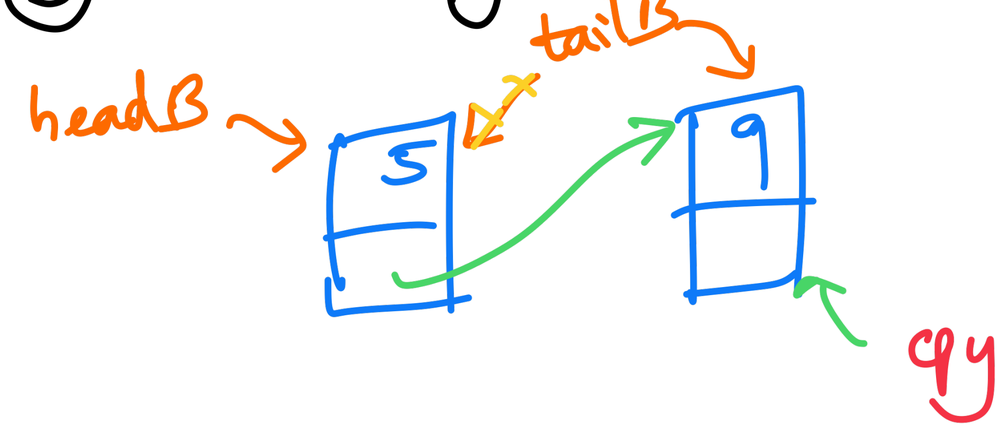



```
Node* tmpA = headA;

while(tmpA != NULL){
  Node* cpy = new Node;
  cpy->m_val = tmpA->m_val;
  if(headB == NULL){
      headB = cpy;
      tailB = cpy; }
  else{
      tailB->next = cpy;
      tailB = cpy;
  }
  tmpA = tmpA->next; }
```

① Go through each element of A → Traversal

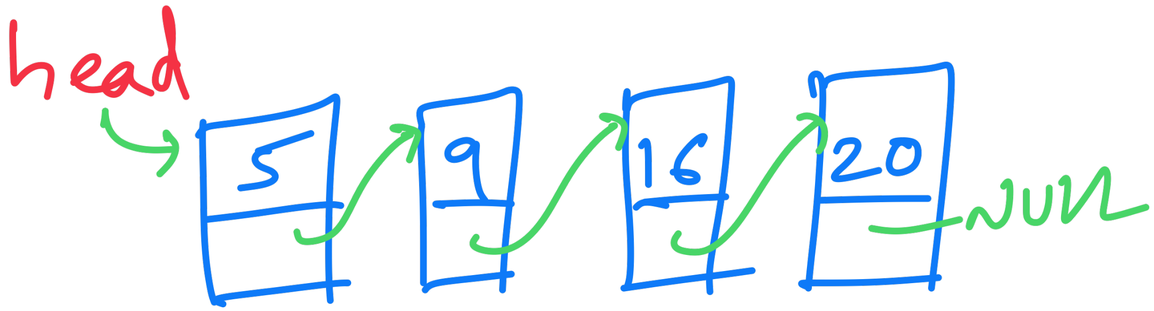
② create copy of the element

③ Add copy to 'B' list.



Linked Lists : Problems

Delete a linked list

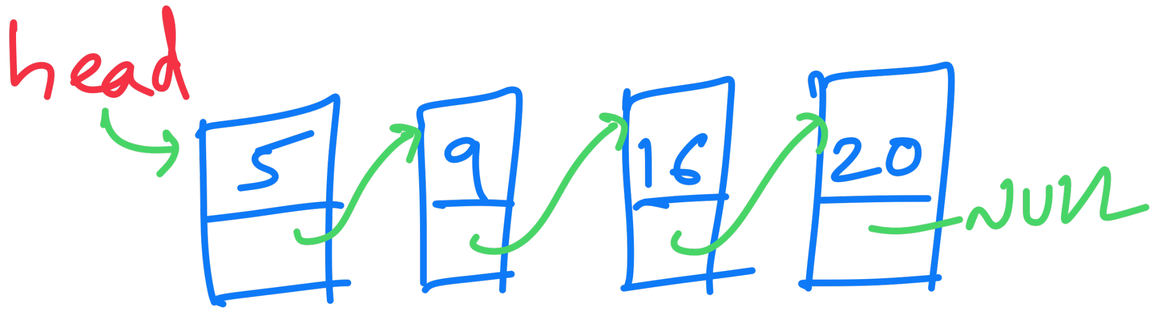


delete all nodes.

Any ideas?

Linked Lists : Problems

Delete a linked list



delete all nodes.

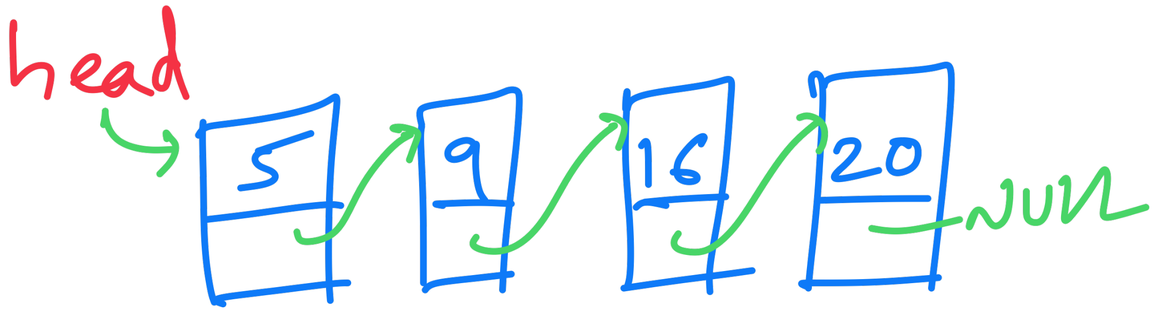
Any ideas?

→ Traversal !!

```
Node* tmp = head;

while(tmp != NULL){

    tmp = tmp->next;

}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

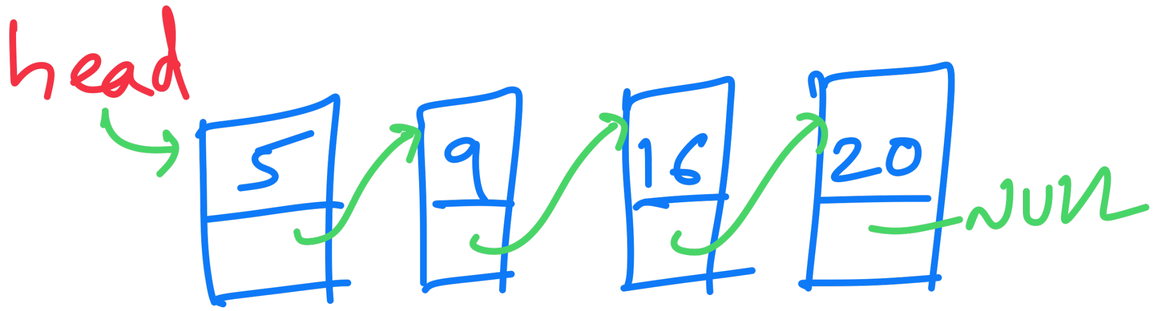
Any ideas?

→ Traversal !!

```
Node* tmp = head;

while(tmp != NULL){

  (1) _ ------

  tmp = tmp->next;

}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

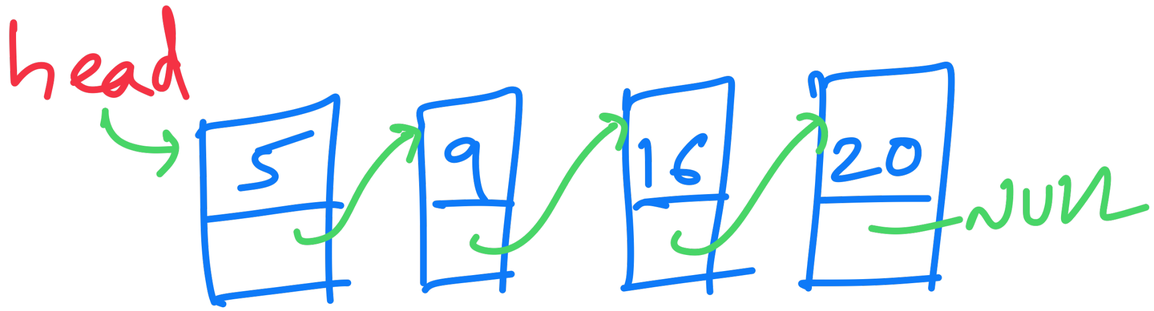
Any ideas?

→ Traversal !!

```
Node* tmp = head;

while(tmp != NULL){

  ① delete tmp ??

  tmp = tmp->next;

}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

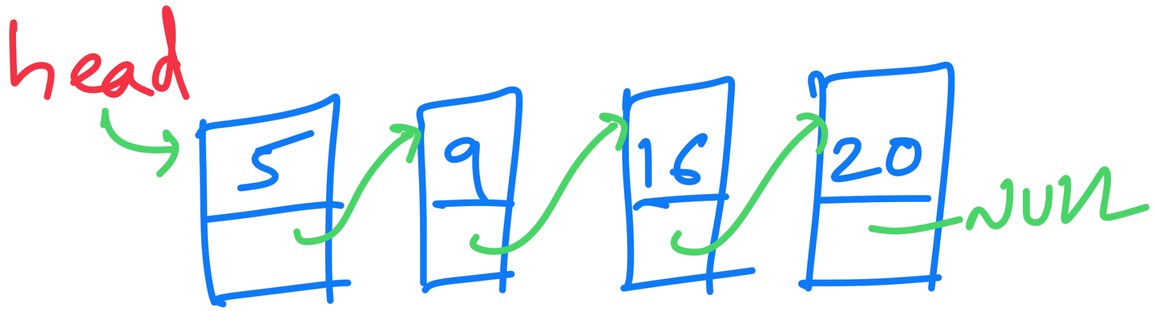
Any ideas?

→ Traversal !!

```
Node* tmp = head;

while(tmp != NULL){

    delete tmp ?? → "NO"

    tmp = tmp→ next;

}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

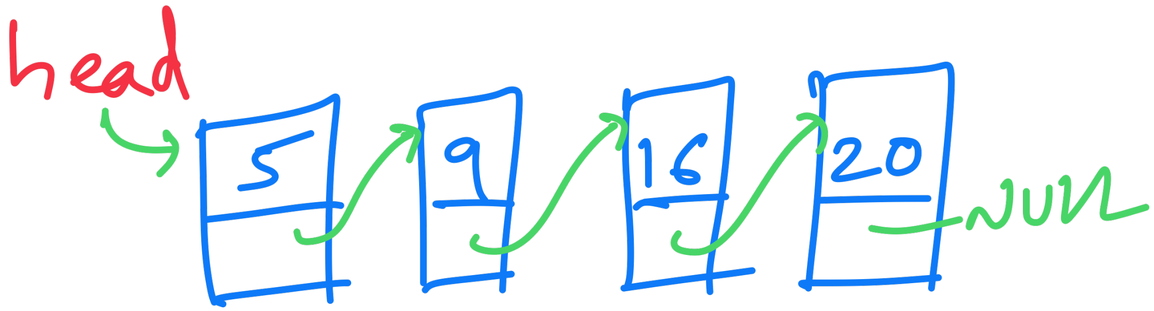
Any ideas?

→ Traversal !!

```
Node* tmp = head;

while(tmp != NULL){

  ① ------------

    tmp = tmp->next;
  ② delete tmp !?
}
```

Linked Lists : Problems

Delete a linked list



delete all nodes.

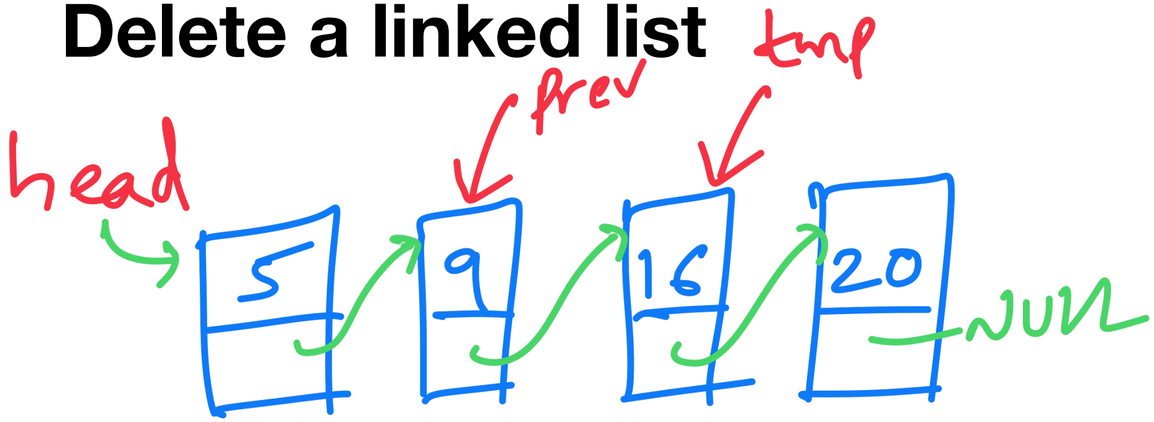
Any ideas?

→ 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.

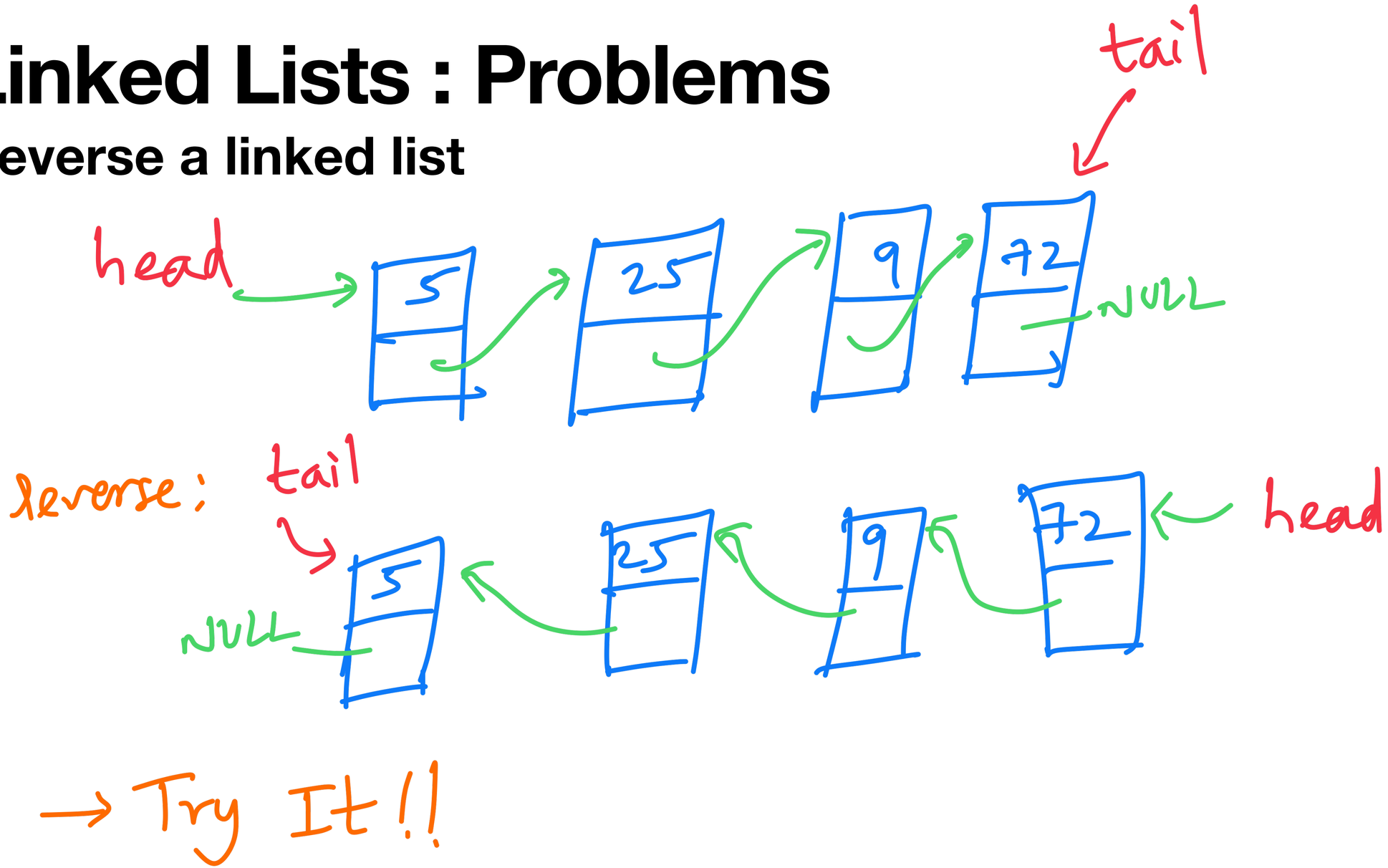
Any ideas?

→ Traversal !!

```
Node* tmp = head;

while(tmp != NULL){

  ① Node* prev = tmp;

  tmp = tmp->next;

  ② delete prev;

}
```

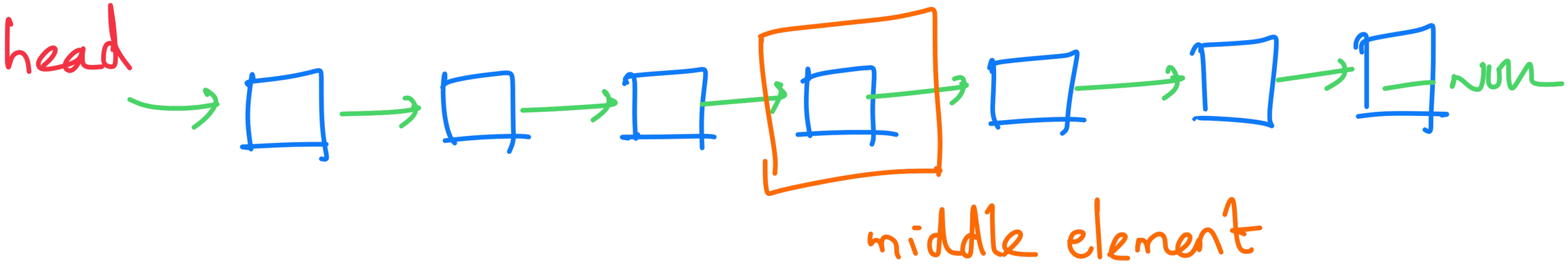
Linked Lists : Problems

Reverse a linked list



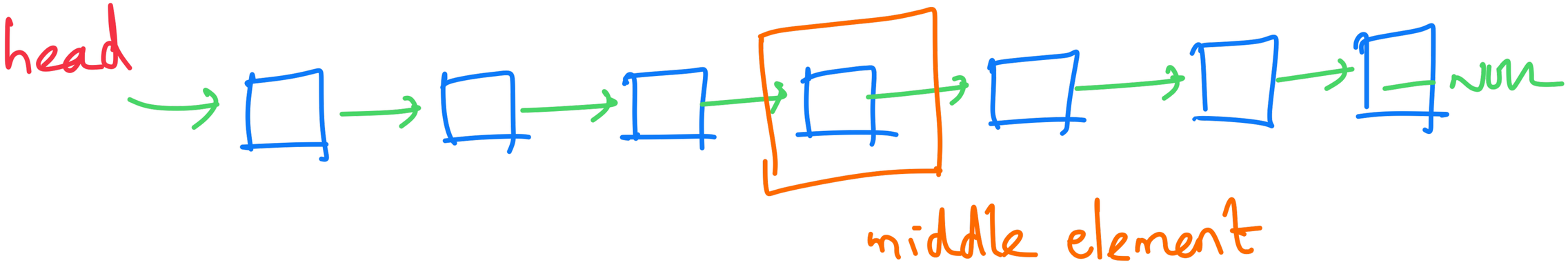
Linked Lists : Problems

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