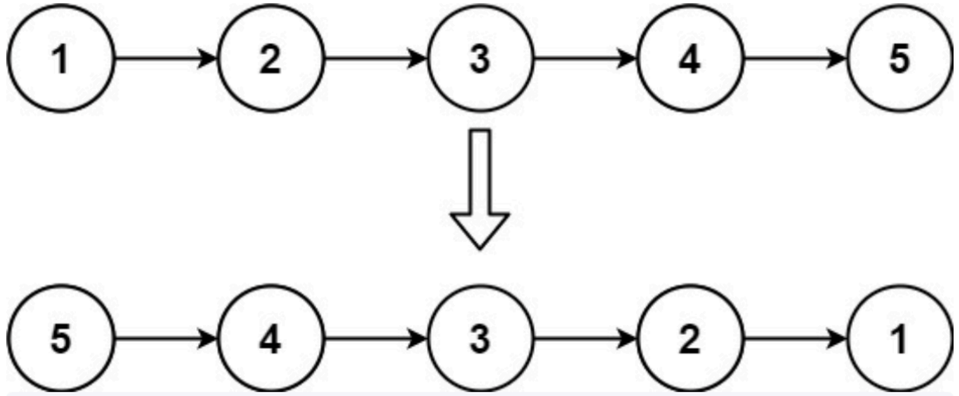
206. Reverse Linked List

Easy 11871 206 Add to List Share

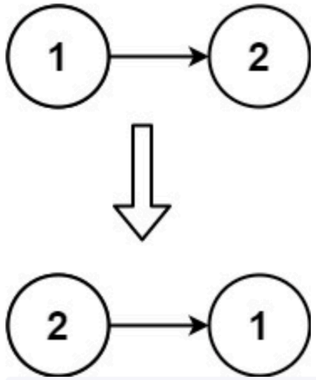
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:




```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

Example 2:



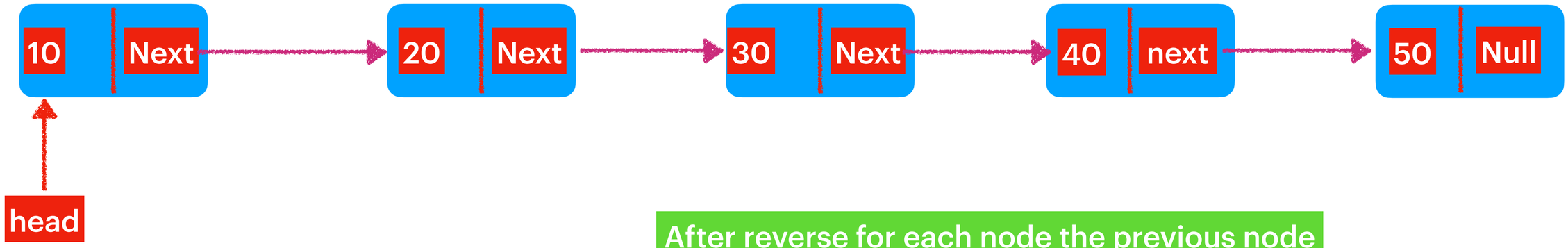

```
Input: head = [1,2]
Output: [2,1]
```

Example 3:

```
Input: head = []
Output: []
```

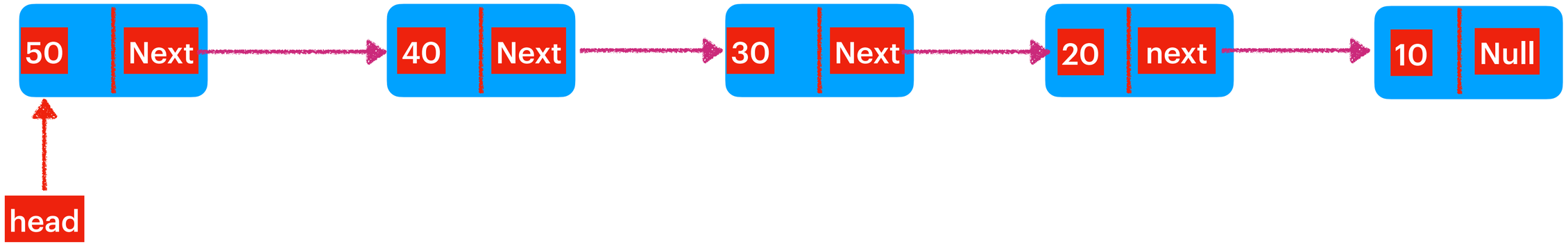
Constraints:

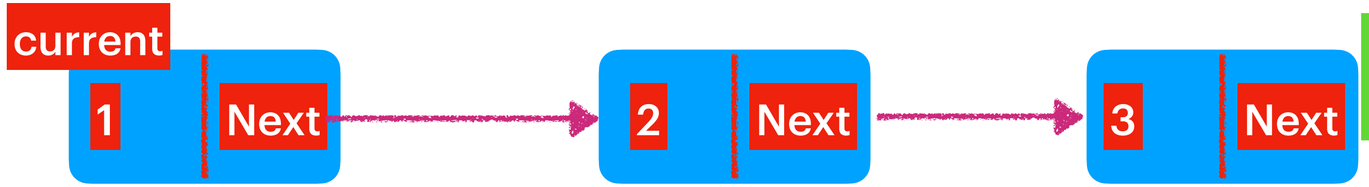
- The number of nodes in the list is the range `[0, 5000]`.
- `-5000 <= Node.val <= 5000`



After reverse for each node the previous node becomes next.

After Reverse





As we know that after reverse for each node the previous node becomes next.

Step1

prev = null;
current = 1 -> 2 -> 3 -> null

ListNode tempNext = current.next;
//tempNext = 2->3->null

current.next = prev ;
//current = 1 -> null

prev = current ;
//prev = 1->null

current = tempNext;
//current = 2->3->null

Step2

prev = 1->null
current = 2->3->null

ListNode tempNext = current.next;
//tempNext = 3->null

current.next = prev;
// current =2 -> 1 -> null

prev = current;
// prev = 2 -> 1 -> null

current = tempNext;
//current = 3->null

Step3

prev = 2 -> 1 -> null
current = 3->null

ListNode tempNext = current.next ;
//tempNext = null

current.next = prev;
//current = 3 -> 2 -> 1 -> null

prev = current ;
//prev = 3 -> 2 -> 1 -> null

current = tempNext;
//current = null

Step4

prev = 3 -> 2 -> 1 -> null
current = null;
As the current is null stop the flow
then return prev:
3 -> 2 -> 1 -> null

141. Linked List Cycle

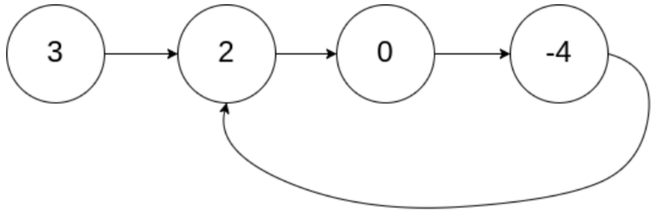
Easy 8109 790 Add to List Share

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

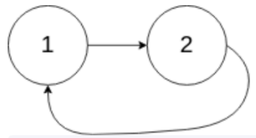
Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:




```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to
the 1st node (0-indexed).
```

Example 2:



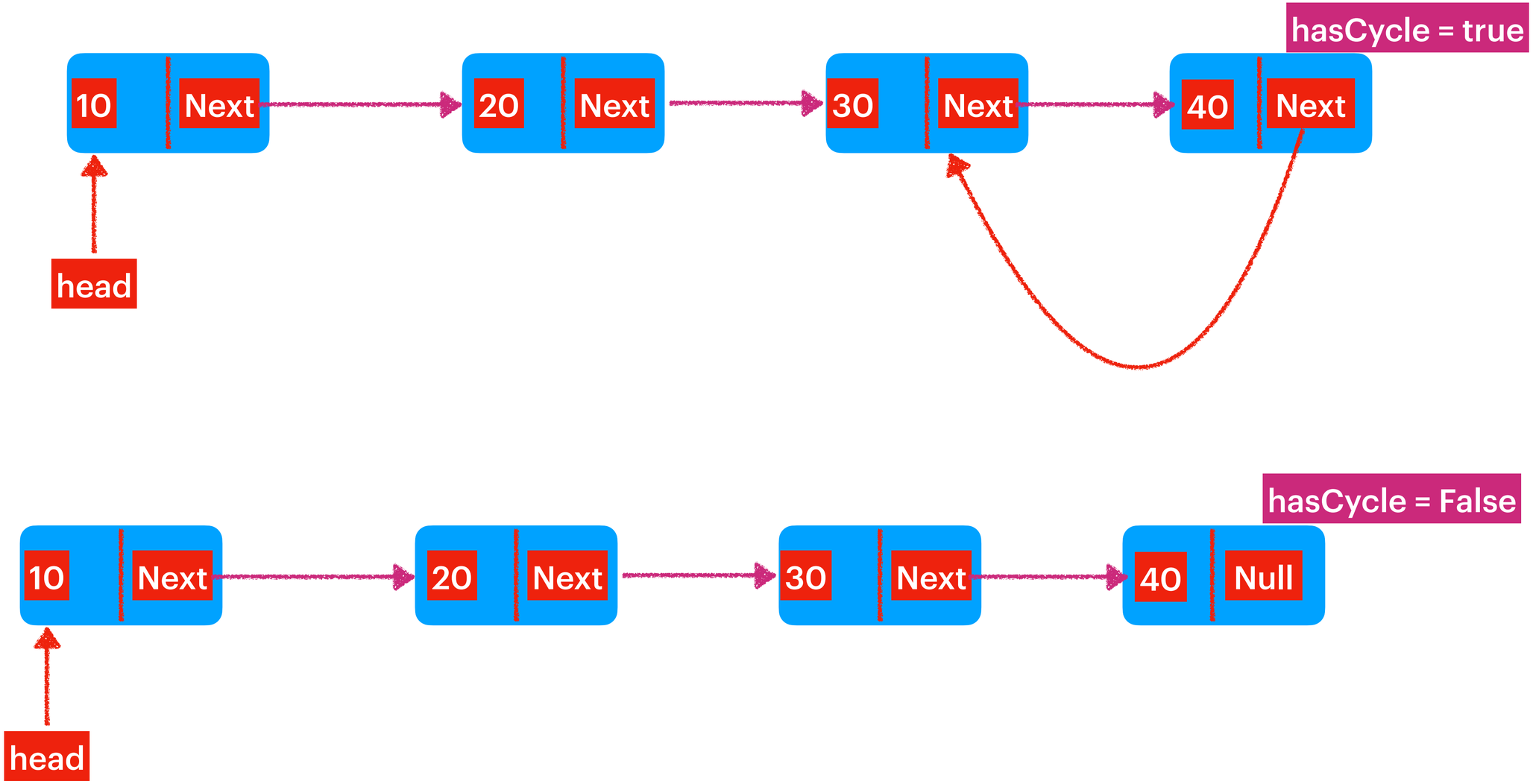

```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to
the 0th node.
```

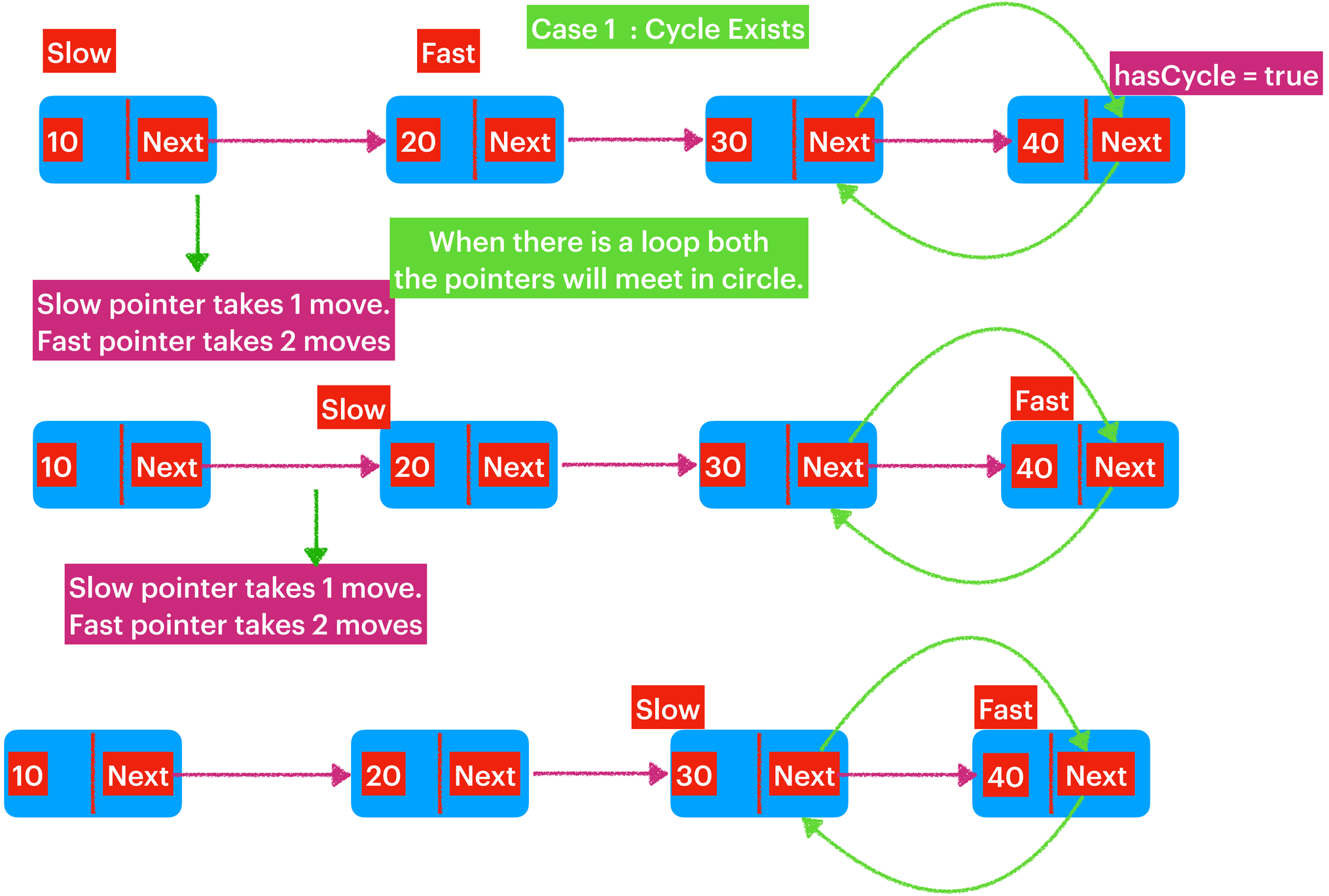
Example 3:

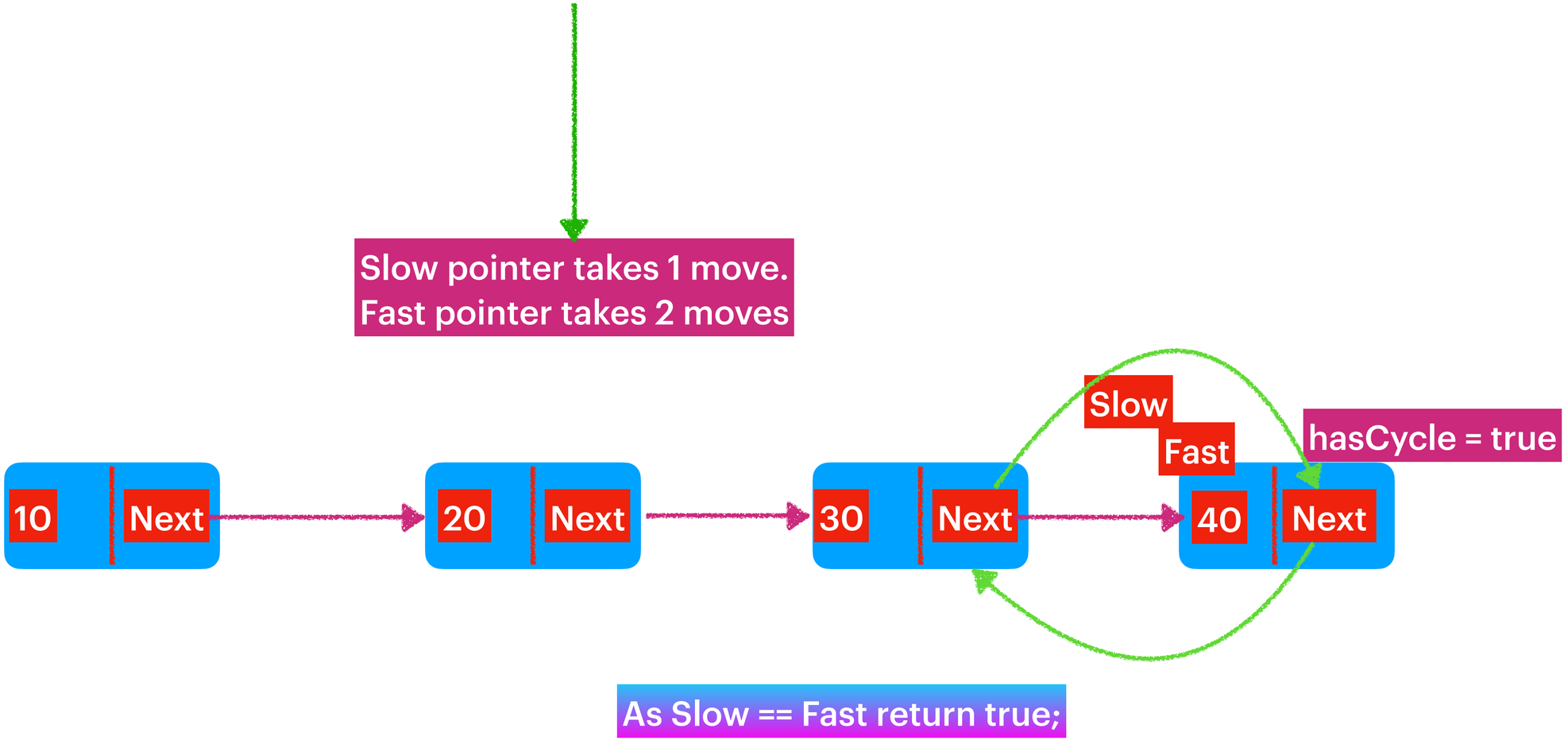

```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

Constraints:

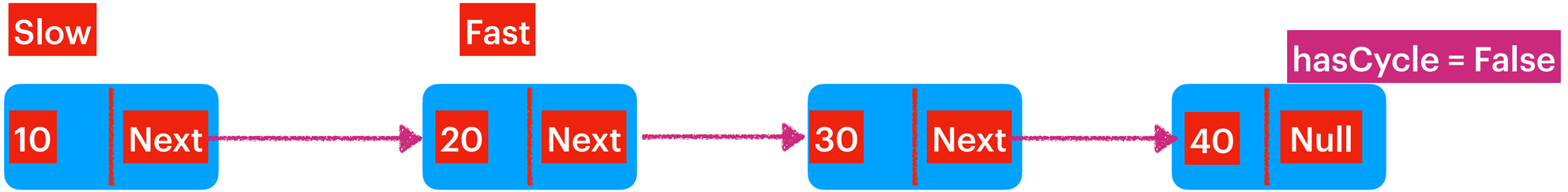
- The number of the nodes in the list is in the range `[0, 10^4]`.
- `-10^5 <= Node.val <= 10^5`
- `pos` is `-1` or a **valid index** in the linked-list.


hasCycle = true
10
Next
20
Next
30
Next
40
Next
head
hasCycle = False
10
Next
20
Next
30
Next
40
Null
head

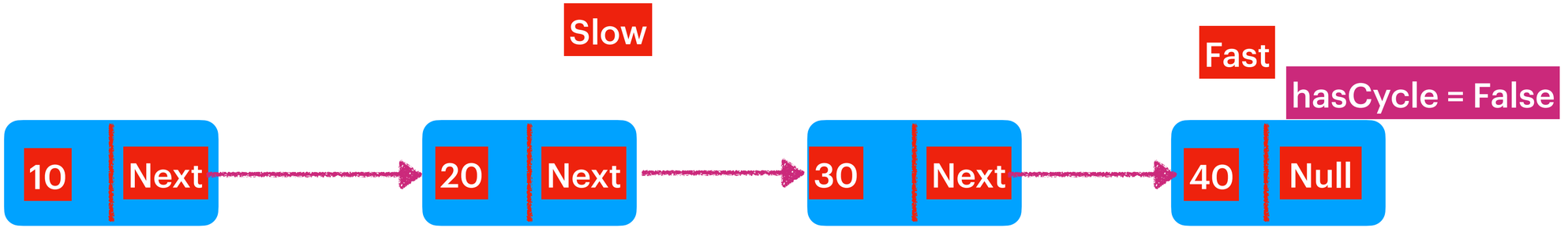

Case 1 : Cycle Exists
Slow
Fast
10
Next
20
Next
30
Next
40
Next
hasCycle = true
When there is a loop both
the pointers will meet in circle.
Slow pointer takes 1 move.
Fast pointer takes 2 moves
Slow
Fast
Slow pointer takes 1 move.
Fast pointer takes 2 moves
Slow
Fast


Slow pointer takes 1 move.
Fast pointer takes 2 moves
10
Next
20
Next
30
Next
40
Next
Slow
Fast
hasCycle = true
As Slow == Fast return true;

Case 2 : No Cycle



Take one Step From slow pointer
then take two steps from fast pointer.



Base Check : Fast == null || Fast.next == null —-> return False