Newton School of Technology

19

Linked List – Part II

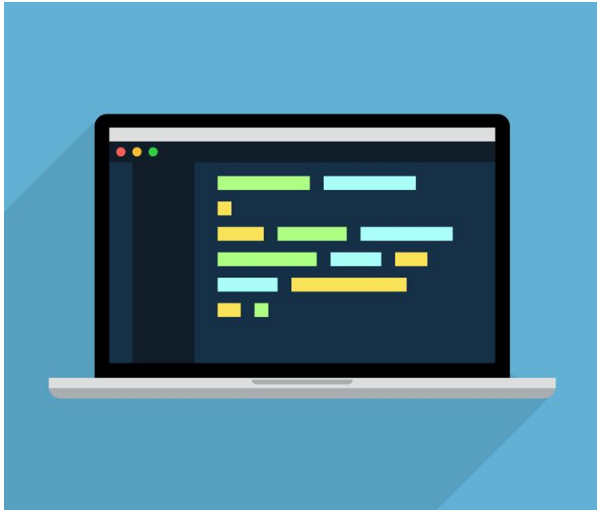
by Gladden Rumao

CSA 221 : DSA

Newton School of Technology

Quick Recap :

- Structure of Linked List
- Insert Node at Beginning
- Insert Node at End
- Delete First Node
- Delete Last Node

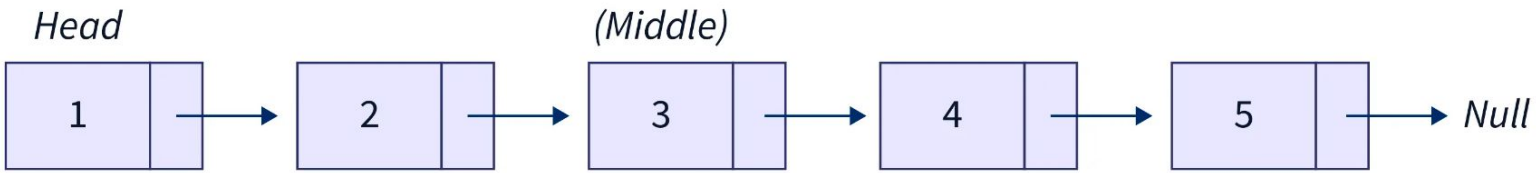


Newton School of Technology

Find Middle in Linked List

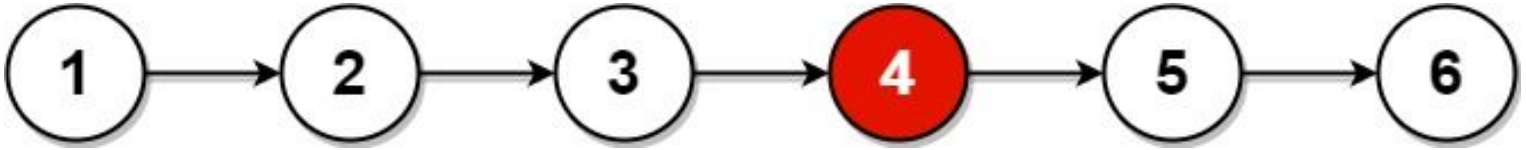
Newton School of Technology

Middle of Linked List :



Newton School of Technology

Middle of Linked List :



Return second middle , if there are two middle nodes.

Newton School of Technology

Brute Force Method :

Newton School of Technology

Brute Force Method :

1. **Traverse the List:** Iterate through the linked list to count the total number of nodes (`n`).

2. **Find Middle Index:** Compute `mid = n / 2` (integer division).

3. **Traverse Again:** Start from the head and move `mid` steps to reach the middle node.

4. **Return Middle Node:** The node at the `mid` position is the middle node.

Newton School of Technology

Middle of Linked List :

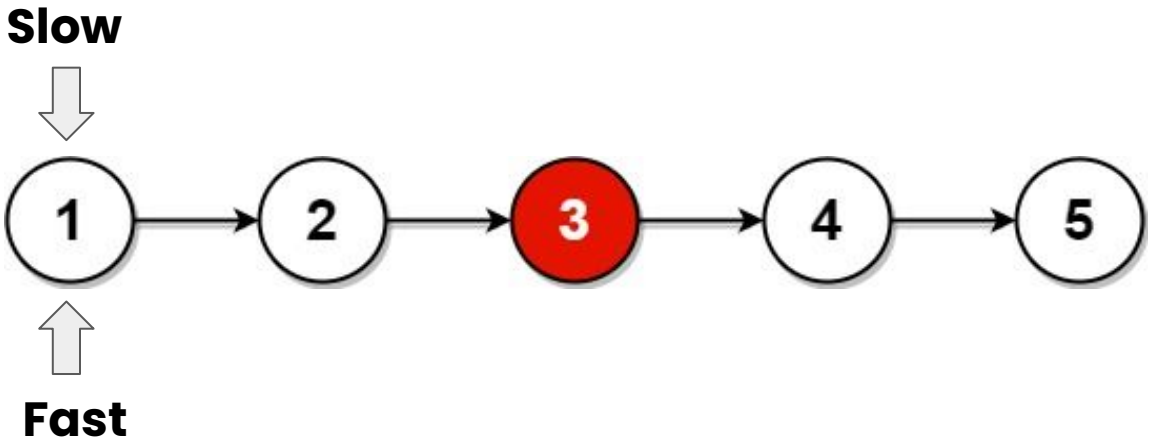
- Can we find the middle in a single pass ?
- Is it possible to find the middle node without computing the length ?

Newton School of Technology

Two Pointers Intuition :

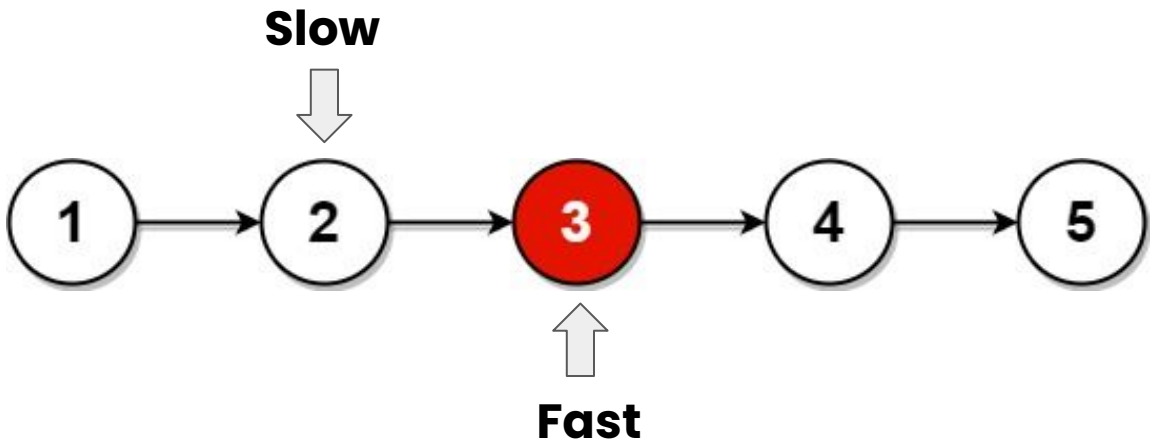
Newton School of Technology

For Odd number of Nodes :



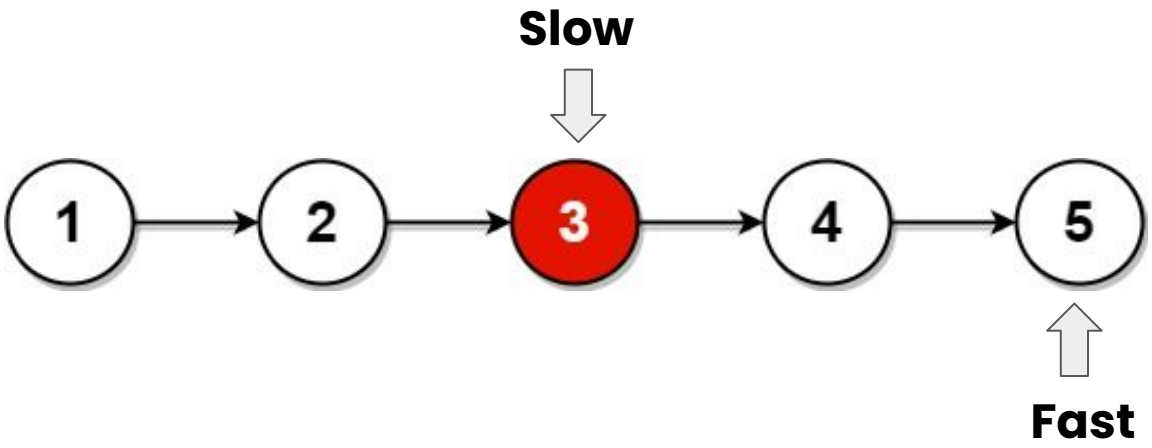
Newton School of Technology

Two Pointers Method (Slow and Fast):



Newton School of Technology

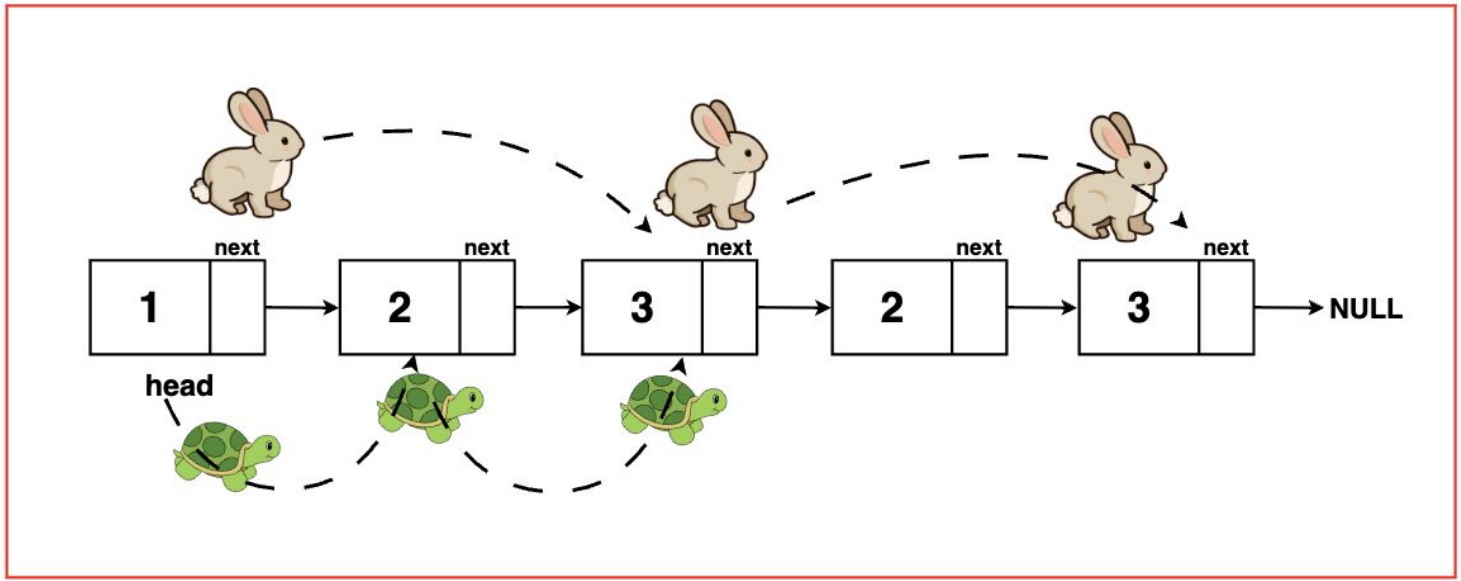
Two Pointers Method (Slow and Fast):



Stop when Fast Pointer reaches last node.

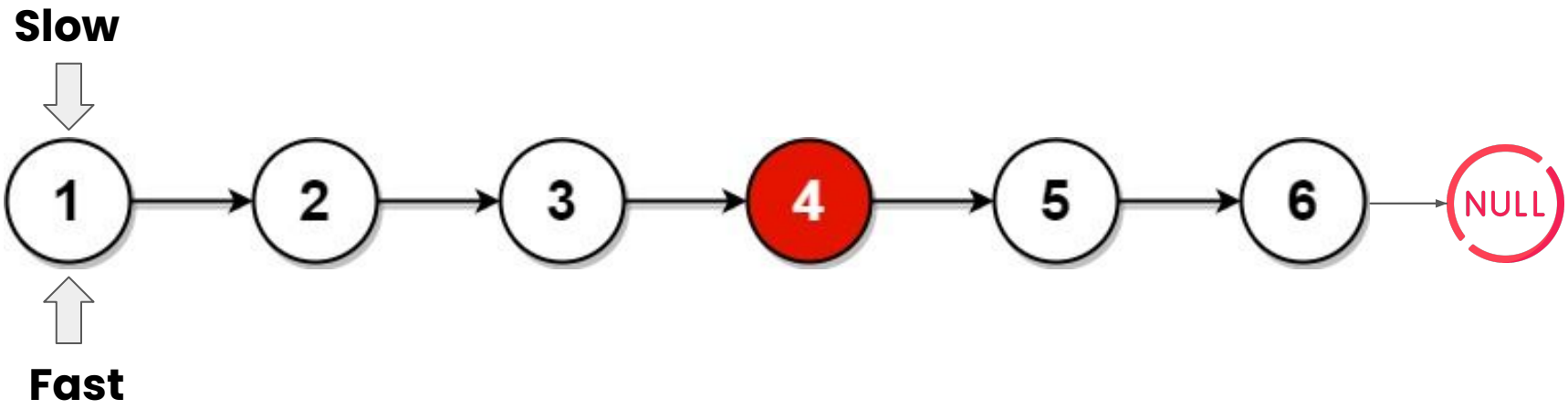
Newton School of Technology

Called as Hare and Tortoise Method :



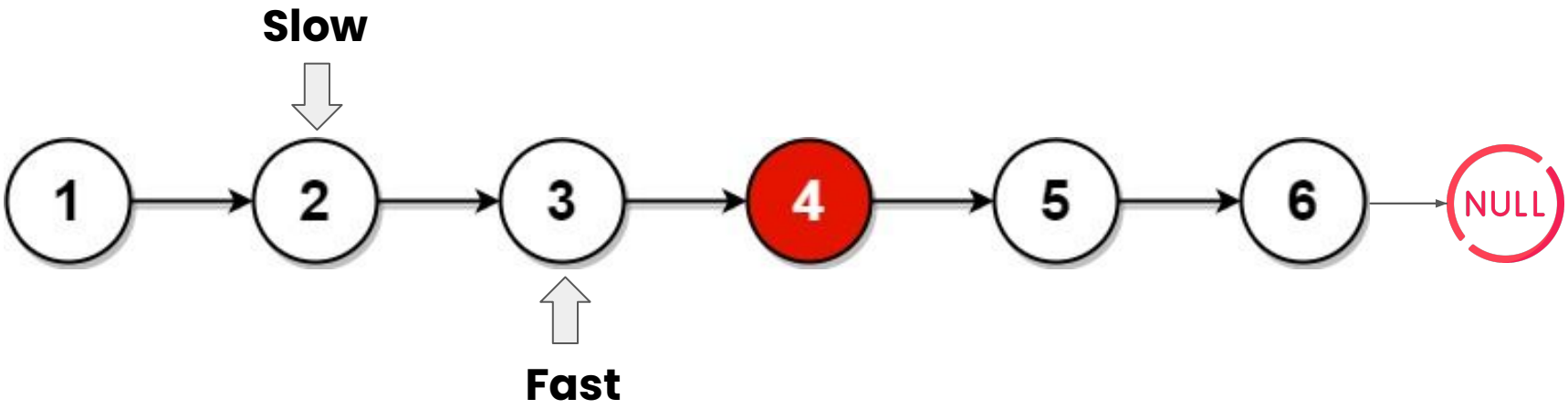
Newton School of Technology

For Even number of Nodes :



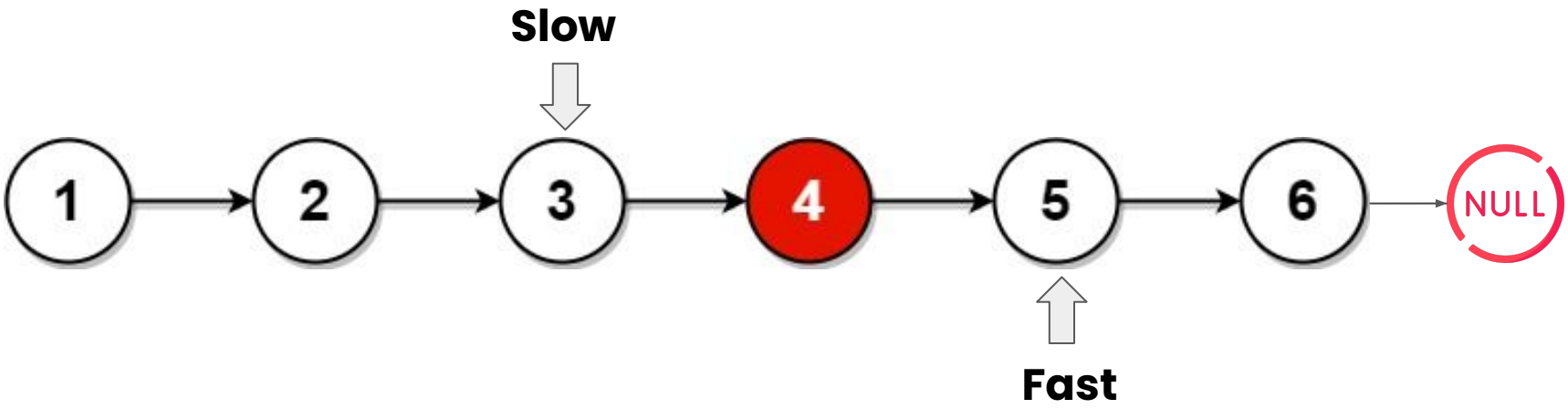
Newton School of Technology

Two Pointers Method (Slow and Fast):



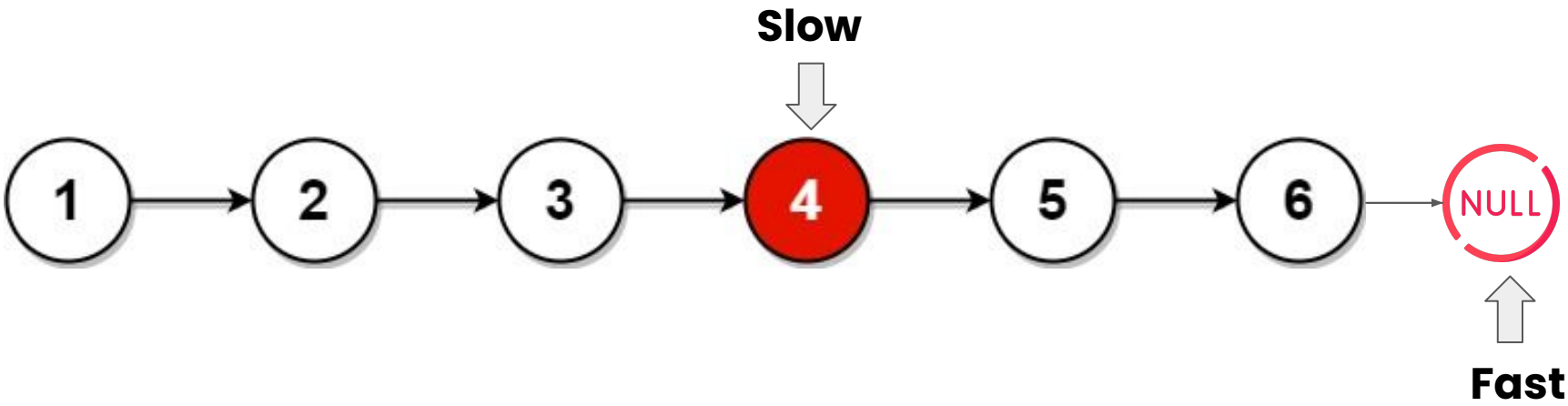
Newton School of Technology

Two Pointers Method (Slow and Fast):



Newton School of Technology

Two Pointers Method (Slow and Fast):



Stop when Fast Pointer reaches None.

Newton School of Technology

Two Pointers Method (Slow and Fast):

1. **Use Two Pointers:** Initialize `slow` and `fast` pointers at the head.
2. **Move Pointers:**
   a. `slow` moves **one step** at a time.
   b. `fast` moves **two steps** at a time.
3. **Stop When Fast Reaches End:**
   a. If `fast` or `fast.next` becomes `NULL`, `slow` is at the middle.
4. **Return Middle Node:** The `slow` pointer now points to the middle node.

Newton School of Technology

Code Implementation :

```
# Optimal Approach - Two Pointer Method
def find_middle(head):
    slow = fast = head

    while fast and fast.next:
        slow = slow.next          # Moves one step
        fast = fast.next.next      # Moves two steps

    return slow  # Middle node
```

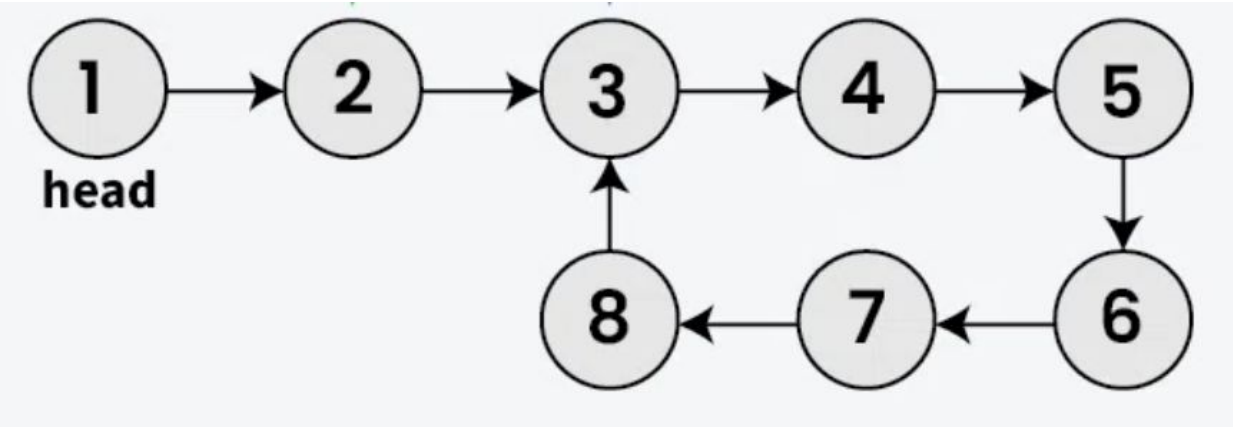
Newton School of Technology

Floyd's Cycle Detection

Newton School of Technology

Problem Statement :

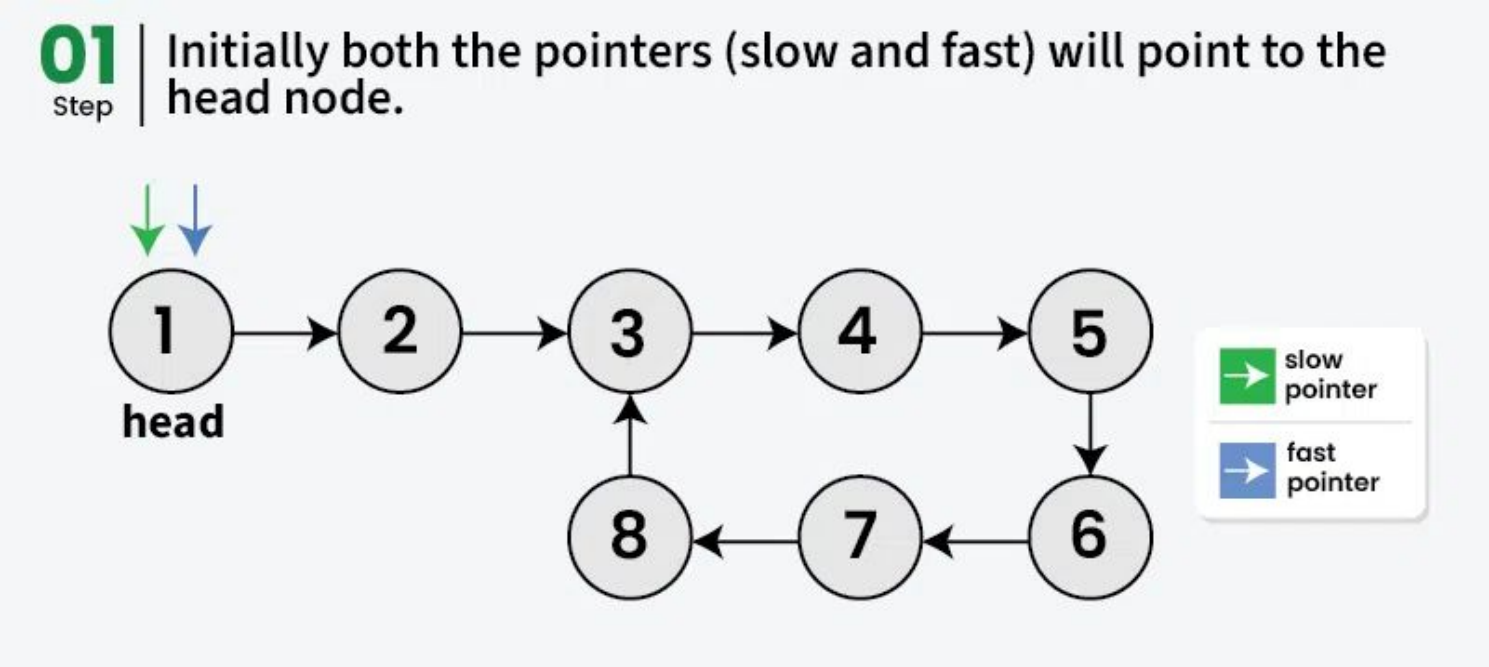
Given a singly linked list, check if the linked list has a loop (cycle) or not. A loop means that the last node of the linked list is connected back to a node in the same list.



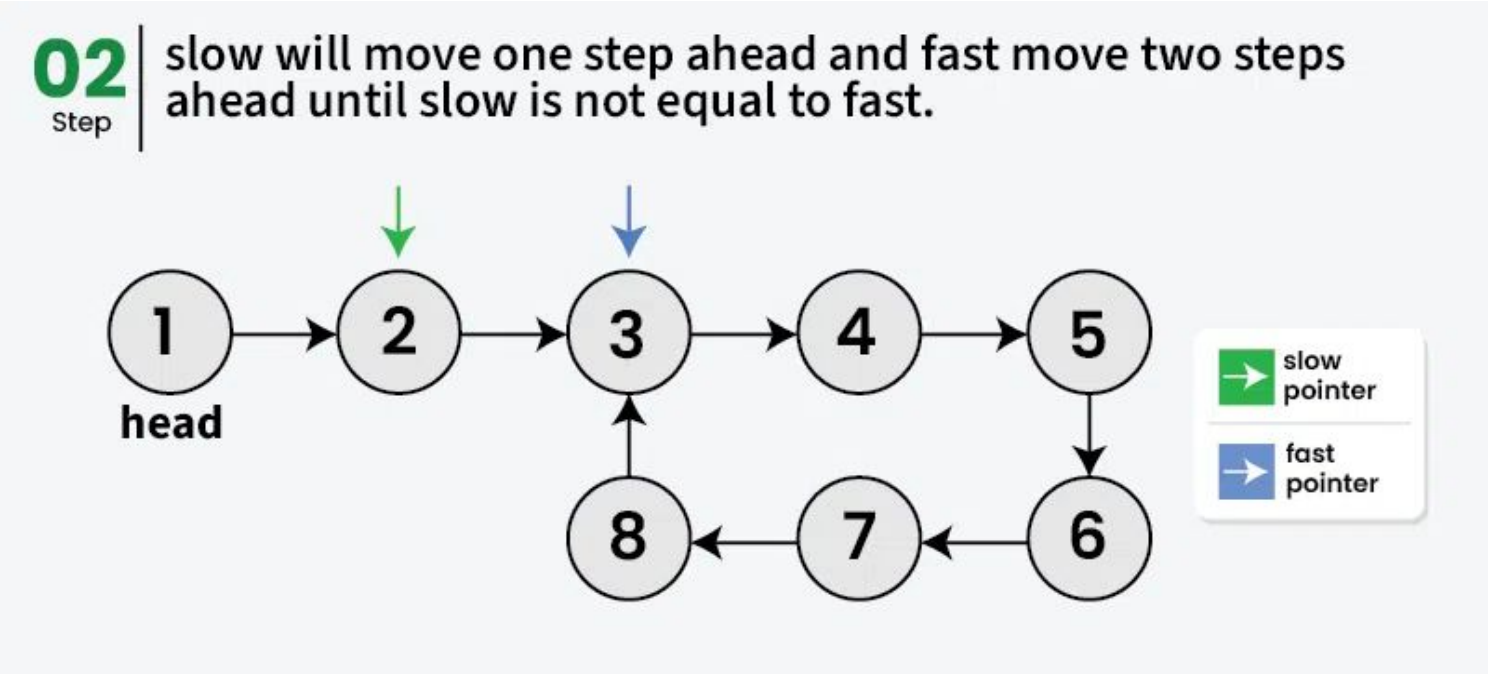
Newton School of Technology

How will you solve the problem ?

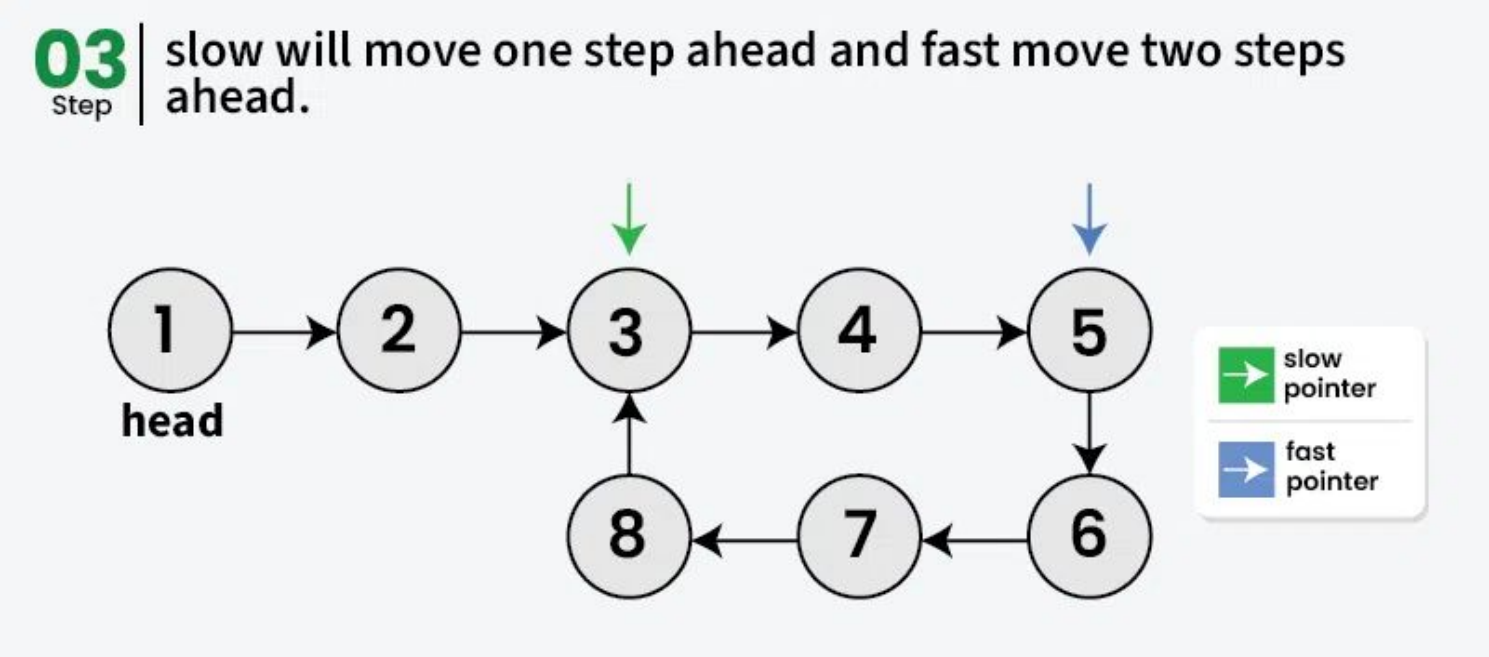
Detect Cycle in Linked List :



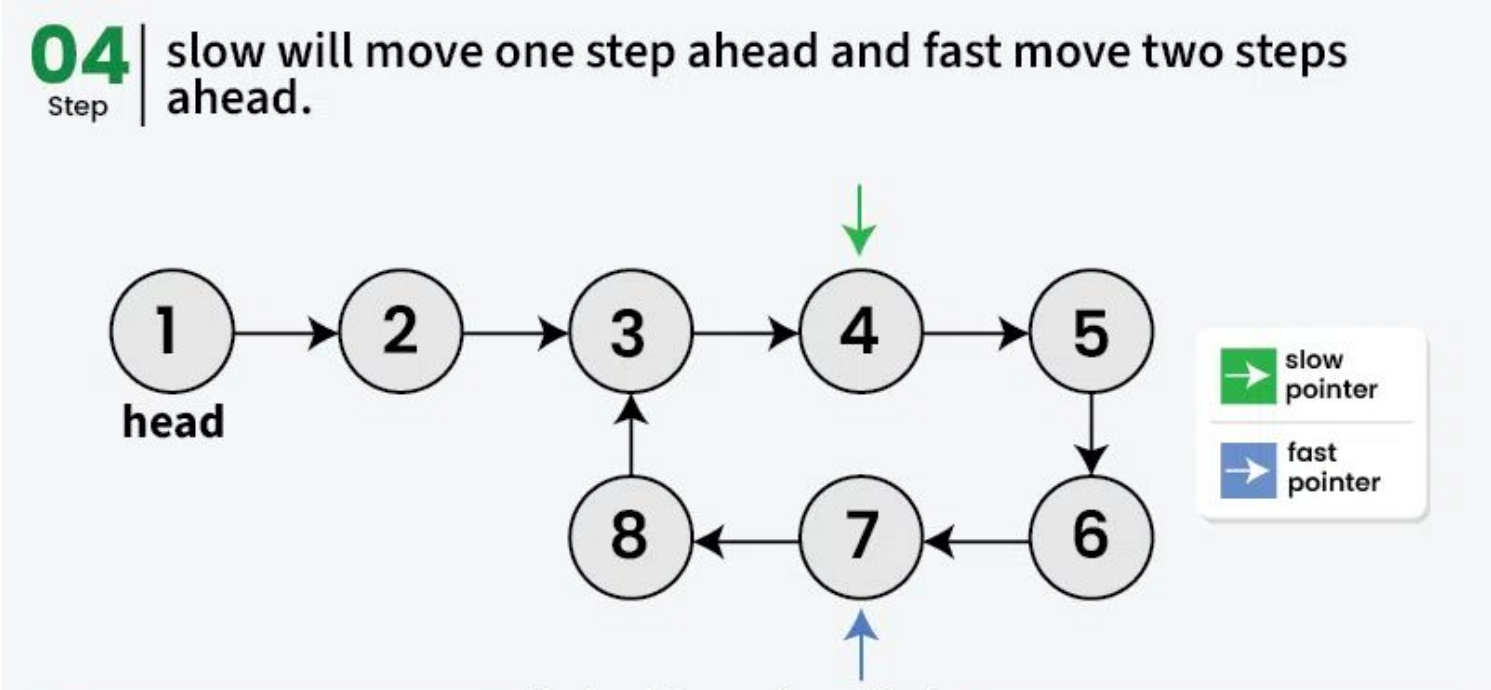
Detect Cycle in Linked List :



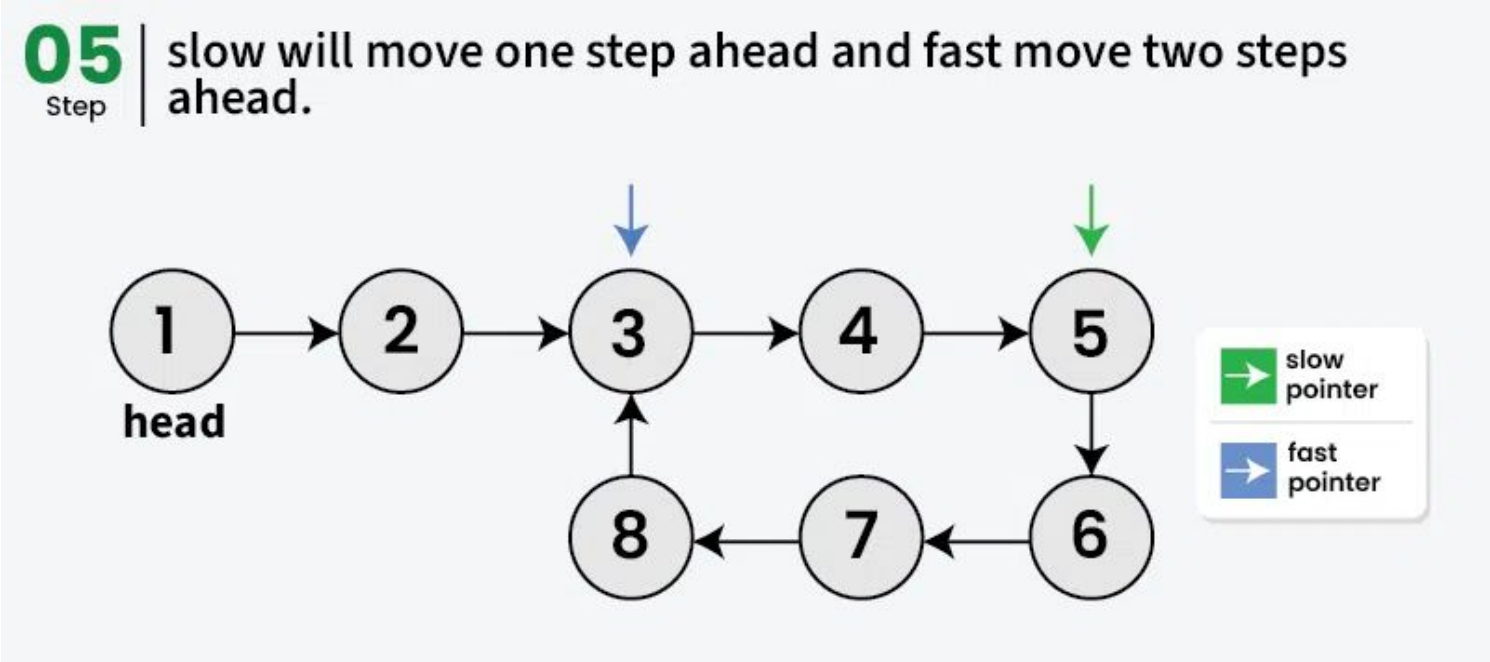
Detect Cycle in Linked List :



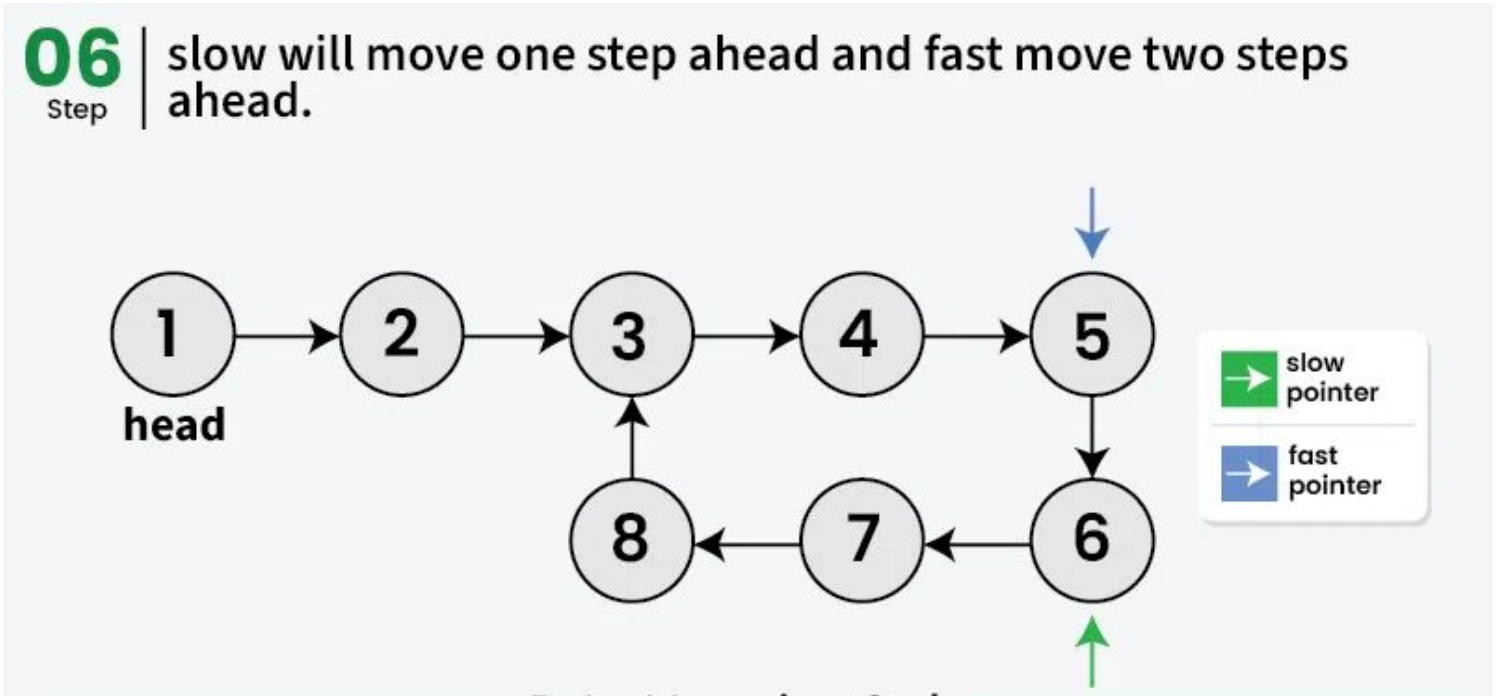
Detect Cycle in Linked List :



Detect Cycle in Linked List :

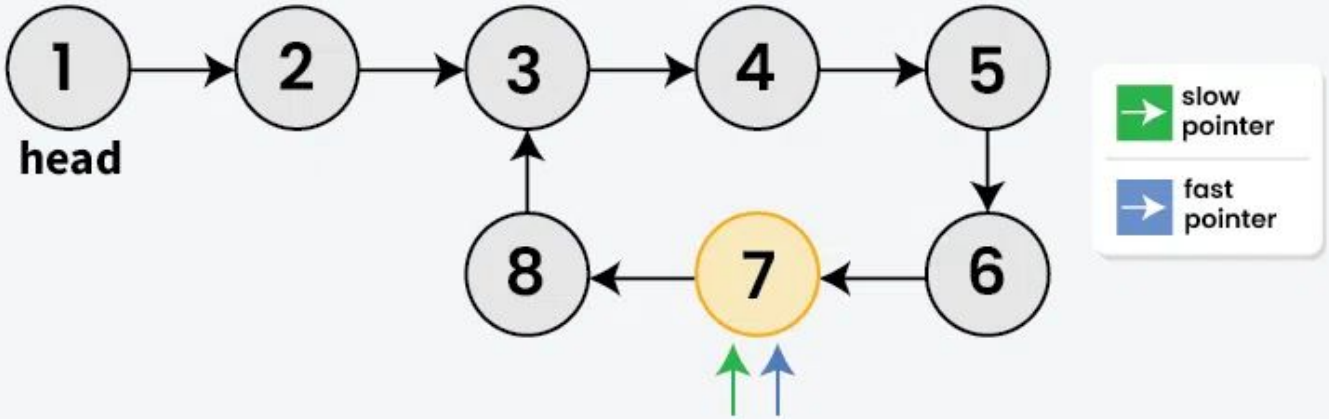


Detect Cycle in Linked List :



Detect Cycle in Linked List :

07 Step | As slow and fast pointer points to the same node, there is a cycle in the Linked List.



Newton School of Technology

Floyd's Cycle Detection Algorithm :

1. **Use Two Pointers:** Initialize ***`slow`*** and ***`fast`*** pointers at the head.
2. **Move Pointers:**
   - ***`slow`*** moves **one step** at a time.
   - ***`fast`*** moves **two steps** at a time.
3. **Check for Cycle:**
   - If **`slow == fast`**, a cycle exists.
4. **Stop Condition:**
   - If **`fast`** or **`fast.next`** becomes **`NULL`**, no cycle exists.

Newton School of Technology

Code Implementation :

```
# Floyd’s Cycle Detection Algorithm
def has_cycle(head):
    slow = fast = head

    while fast and fast.next:
        slow = slow.next          # Moves one step
        fast = fast.next.next      # Moves two steps

        if slow == fast:  # If they meet, cycle exists
            return True

    return False  # No cycle found
```

Newton School of Technology

END