

■ Middle of a Linked List

Fast and slow pointers — the tortoise and hare trick

● Linked List

■ EASY

■ Intuition First

Two runners on a circular track — one runs at double speed. When the fast runner finishes, the slow one is exactly in the middle. Same idea: fast pointer moves 2 steps at a time, slow moves 1. When fast reaches the end, slow is at the middle!

```
List: 1 → 2 → 3 → 4 → 5
      S              F (start)
          S          F (step 1)
              S      F (step 2: F.next=null → stop)
Slow = node 3 = middle ✓

Even length: 1 → 2 → 3 → 4
      S              F (start)
          S          F (step 1)
              S      F (F.next=null → stop)
Slow = node 3 = second middle ✓
```

■ Java Code

```
1  class Solution {
2  public ListNode middleNode(ListNode head) {
3      ListNode slow = head;
4      ListNode fast = head;
5
6      // Fast moves 2 steps, slow moves 1
7      while (fast != null && fast.next != null) {
8          slow = slow.next;
9          fast = fast.next.next;
10     }
11     return slow; // Slow is at middle
12 }
13 }
```

■ Complexity

Type	Value	Meaning
Time	O(n)	One pass — fast reaches end in n/2 iterations

Space	$O(1)$	Only two pointer variables
-------	--------	----------------------------

■ Fast & Slow Pattern

This 'Floyd's two-pointer' technique is reused in: Detect cycle, Find cycle start, Check palindrome LL, Find nth from end. Master this pattern — it appears everywhere!

■ Quick Revision

✓	slow=head, fast=head
✓	Loop: while fast!=null && fast.next!=null
✓	slow=slow.next, fast=fast.next.next
✓	When loop ends, slow points to middle
✓	Even length: slow at second middle node