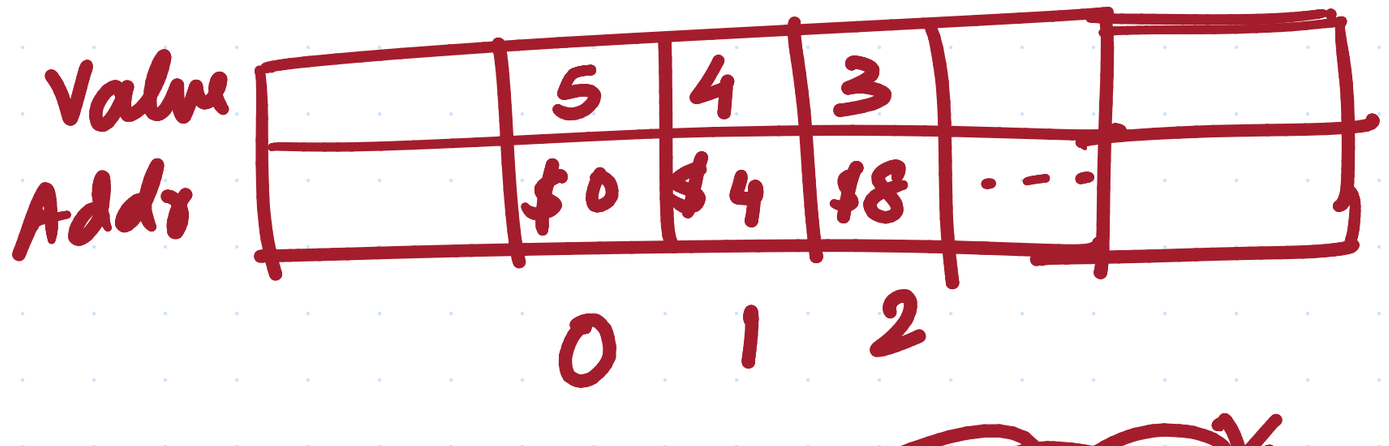
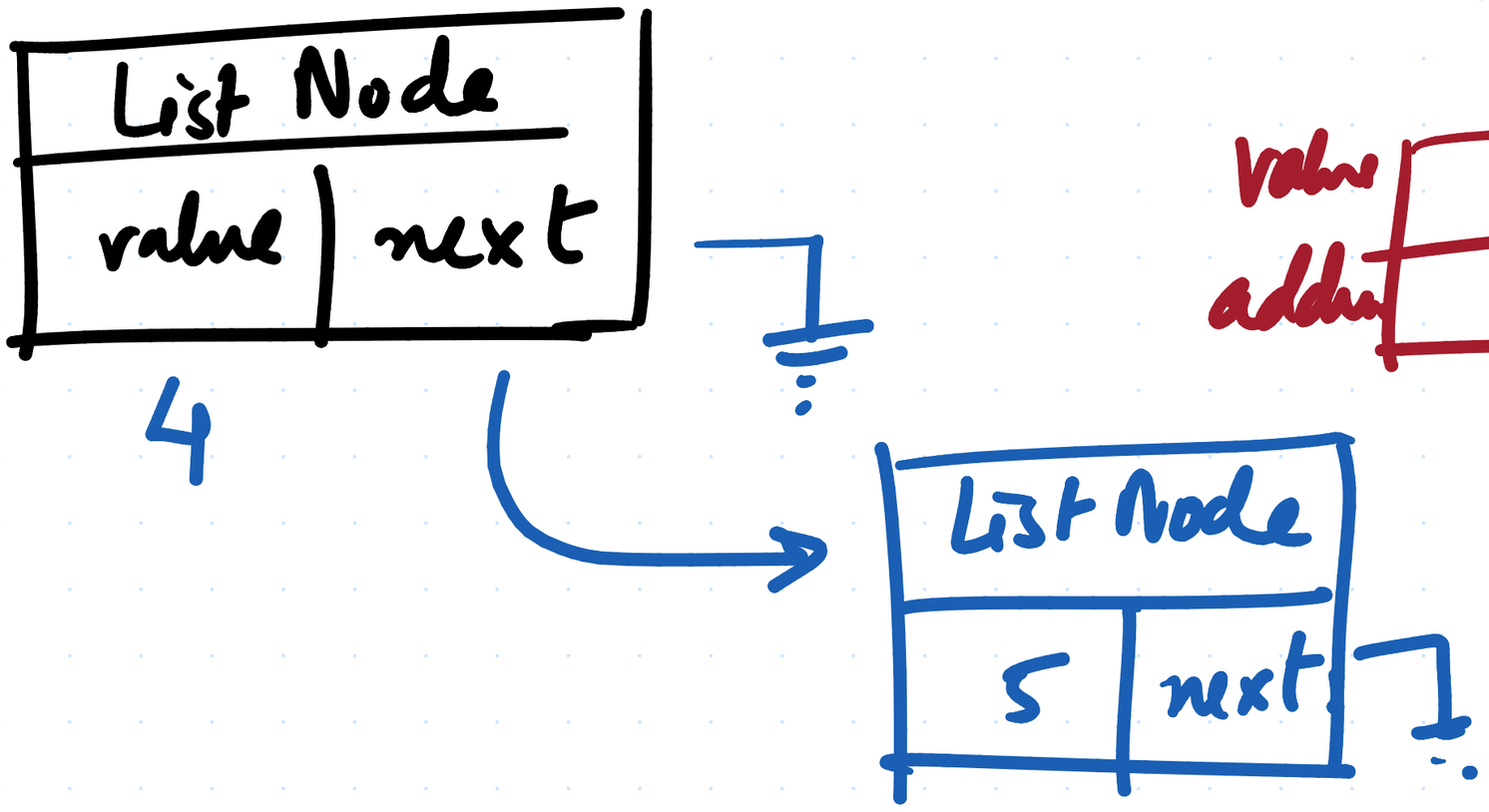
Linked List.

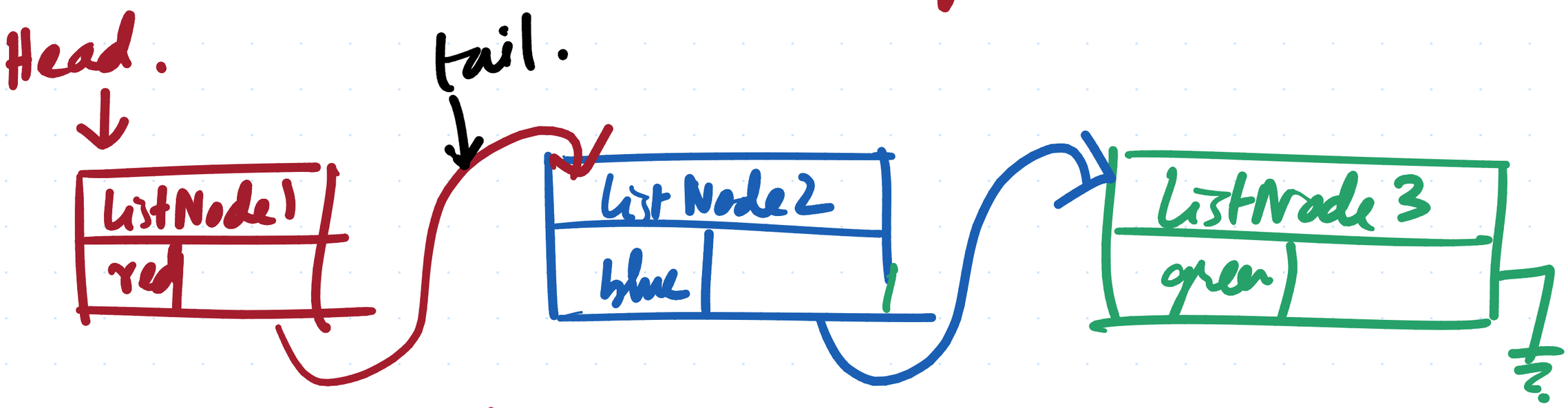
30/8/25

Node





The order in the actual memory will be inconsistent with the references in the linked list.



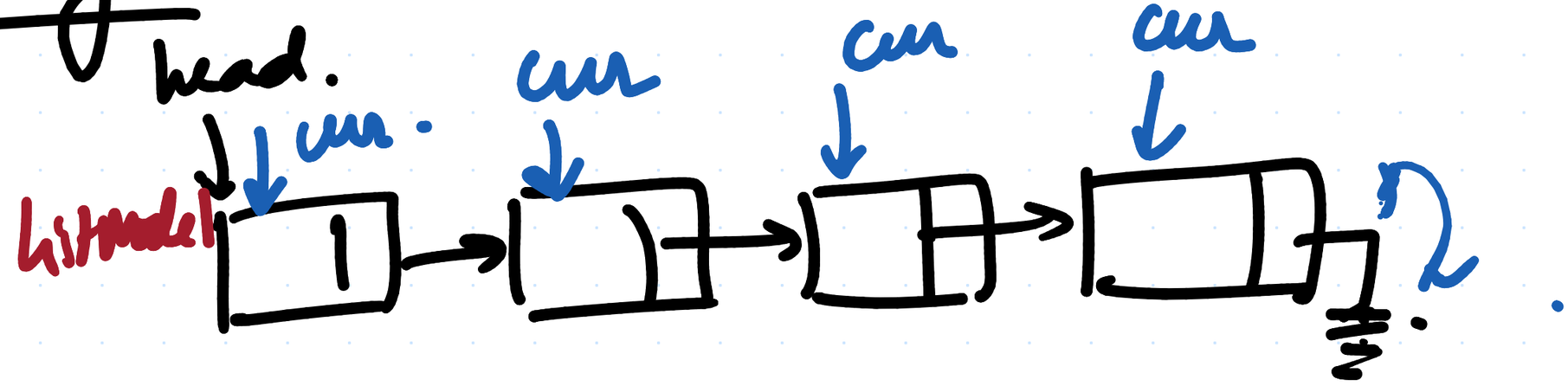
ListNode1.next = ListNode2

ListNode2.next = ListNode3

ListNode3.next = None.

Tail - references the object of ListNode type.

Arrays are stored in the same way as they are indexed (in the memory); But linked lists are not stored in the same way in the memory.



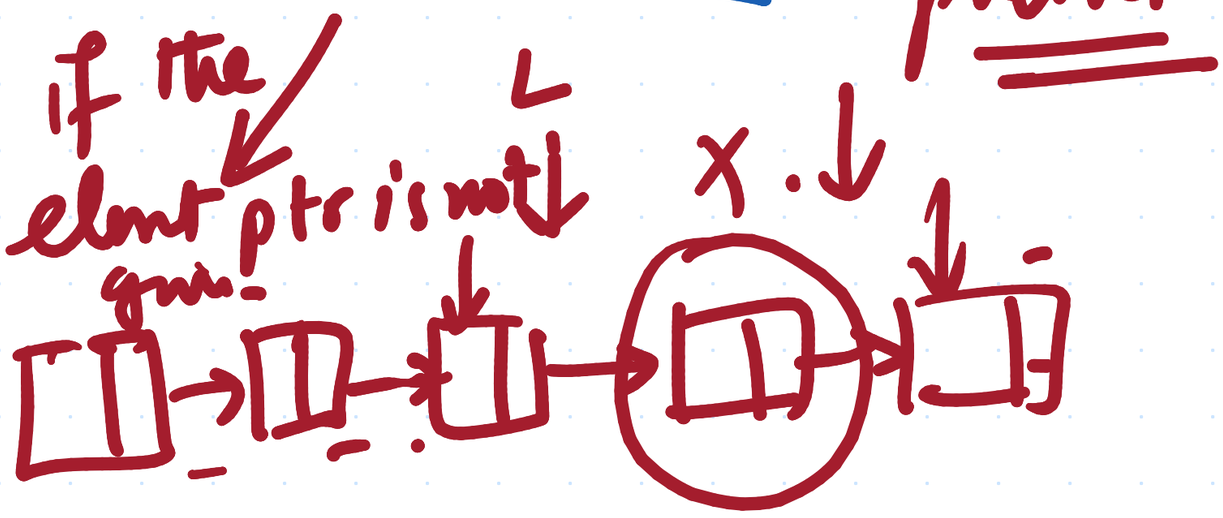
We want to start at the beginning of the linked list and loop to the end of the linked list.

```
cur = listNode1
while cur != null:
    cur = cur.next
```

What is the final value of cur pointer?

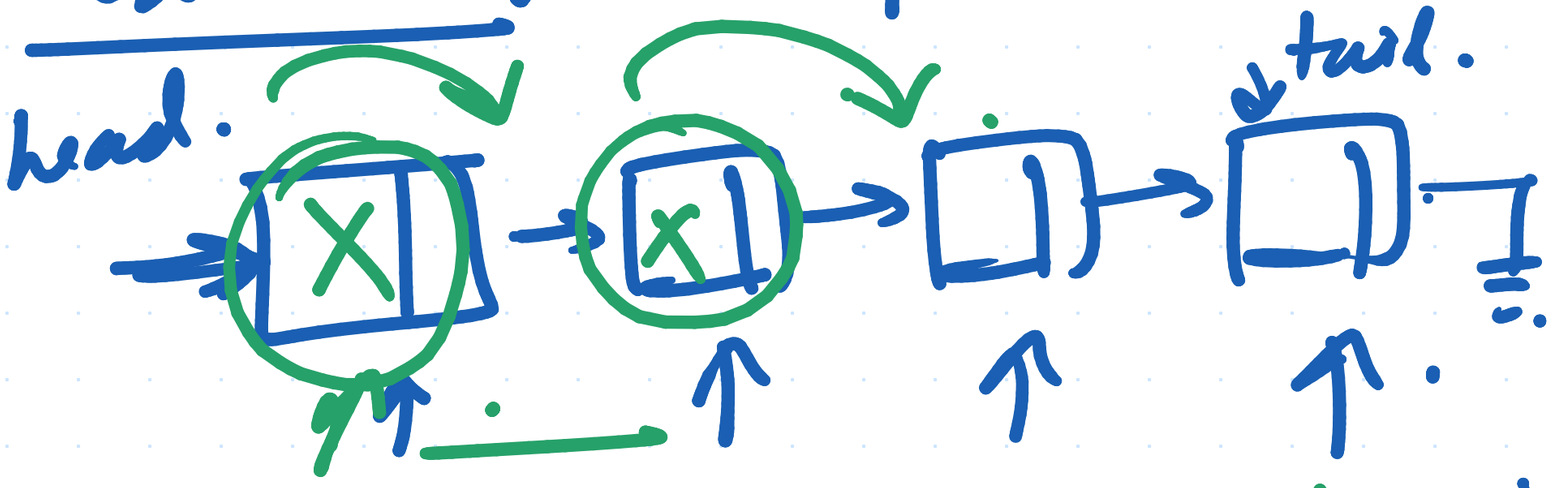
null.

Operation	Arrays Big-O-time	Linked list. Big-O-time.
Access the ith element	O(1)	O(n)
Insert/ Remove end	O(1)	O(n) → if tail is not given. O(1) → if tail is given.
Insert/ remove middle.	O(n)	O(1) → when the pointer is present. O(n)

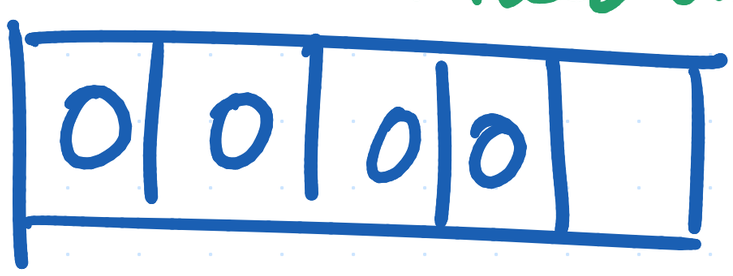


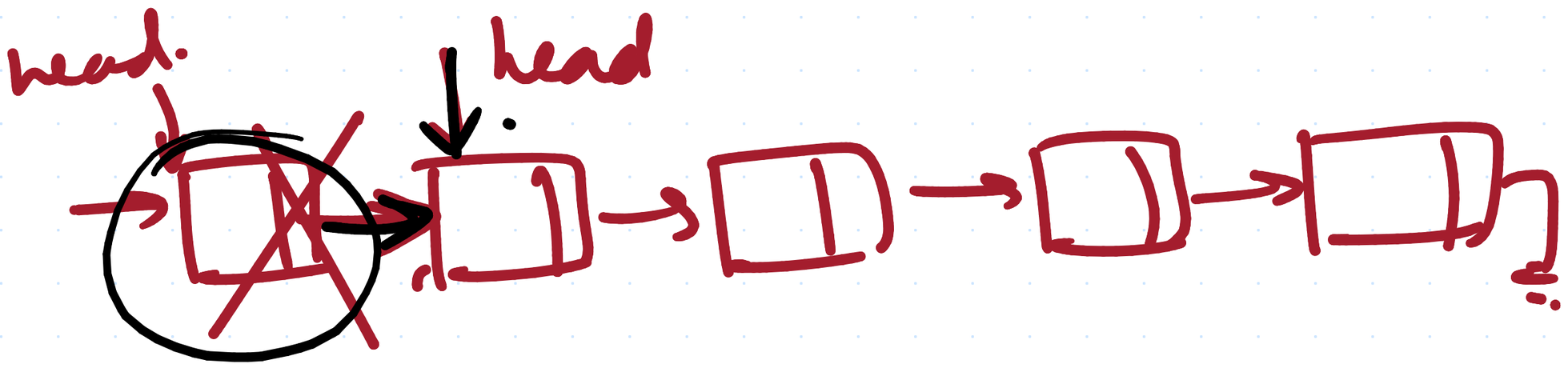
Queues.:

First in First Out.



Queue is similar / simulated using linked list.





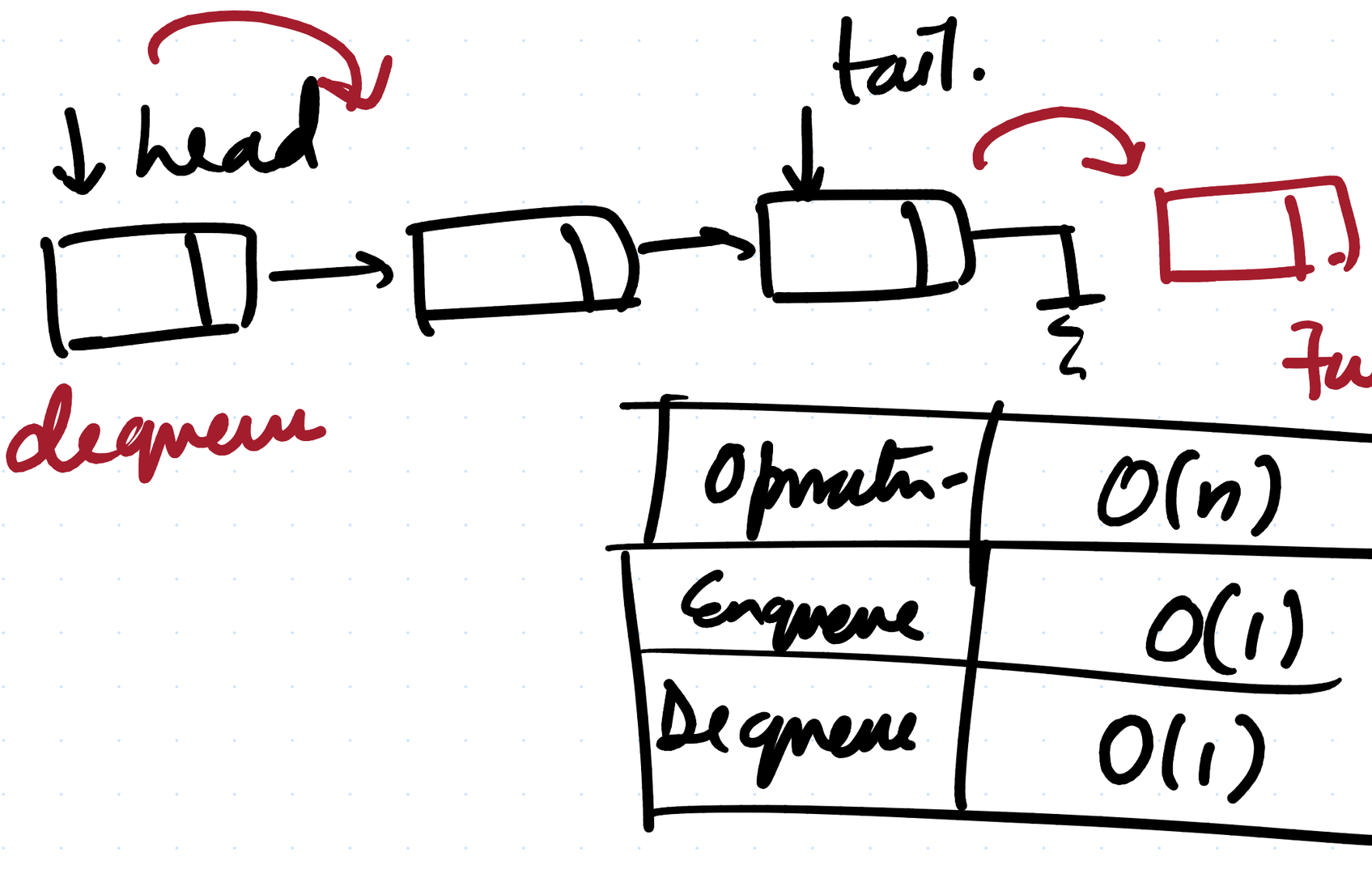
OS Garbage collector / programming lang garbage collector.

will take care of these issues.

Enqueue : add the element.

Dequeue : remove the element.

The elements are to be removed in the same order that they were added.

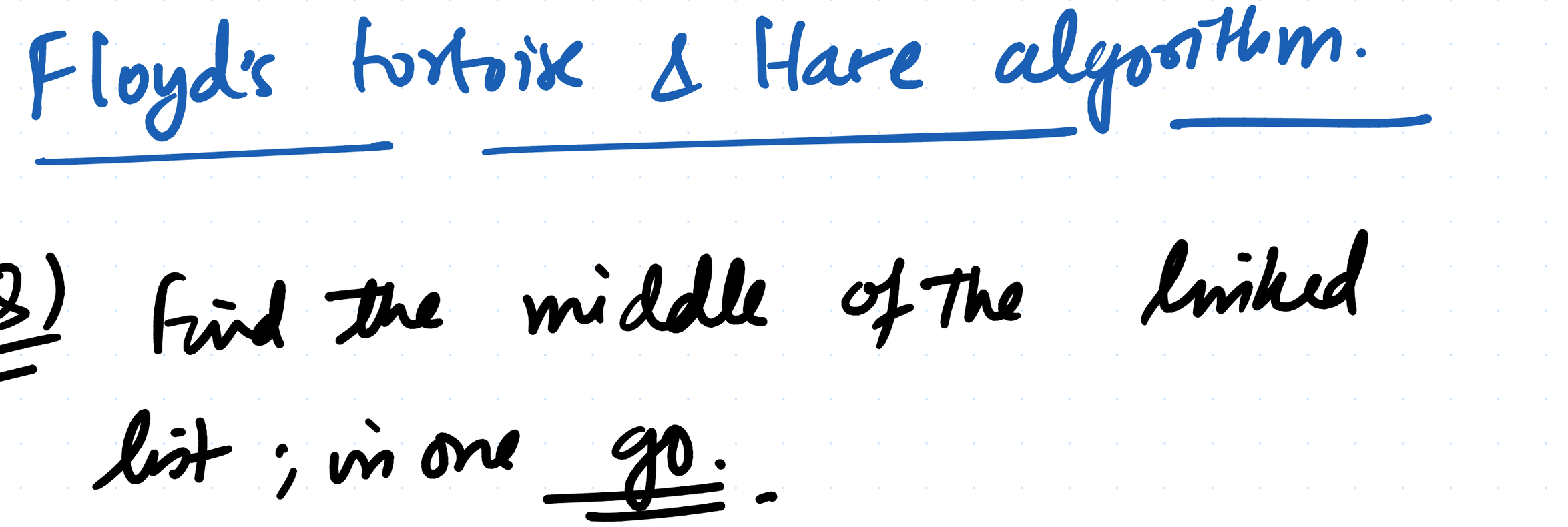


Operation	O(n)
Enqueue	O(1)
Dequeue	O(1)

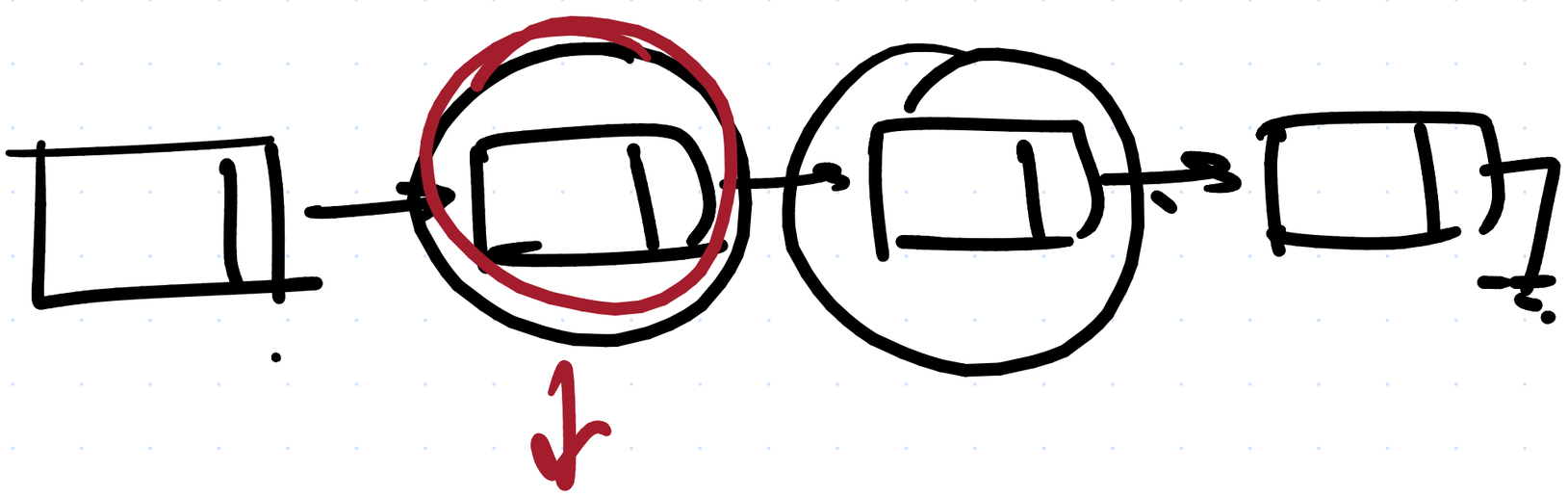
So, we can implement queues with arrays, but it can get a lot more complicated ; (we have to shift & add & there are fixed no. of elements in the array) hence we use linked list for Queues.

Floyd's tortoise & Hare algorithm.

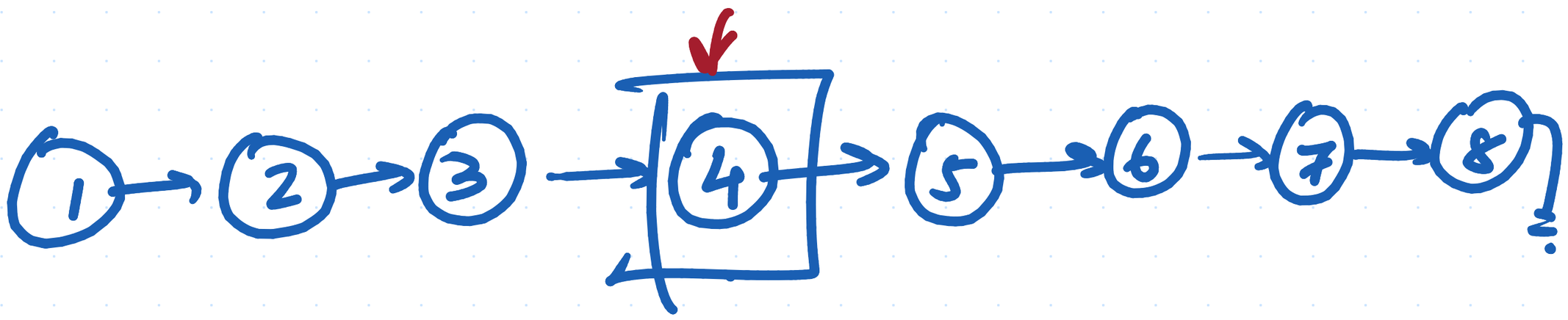
Q) Find the middle of the linked list ; in one go.



odd no. of elements.



first one is the middle.



Time Complexity of this operation

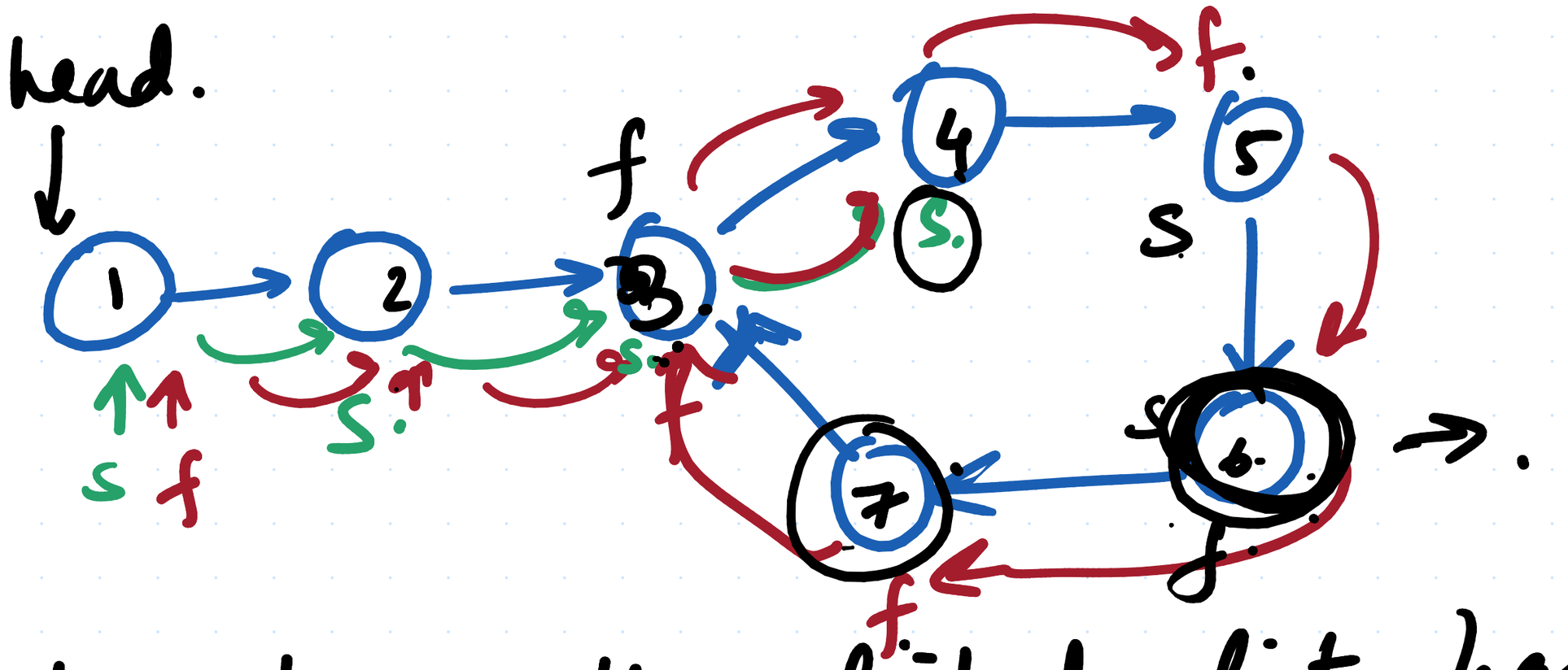
O(n)

Time O(n); Space → O(1)

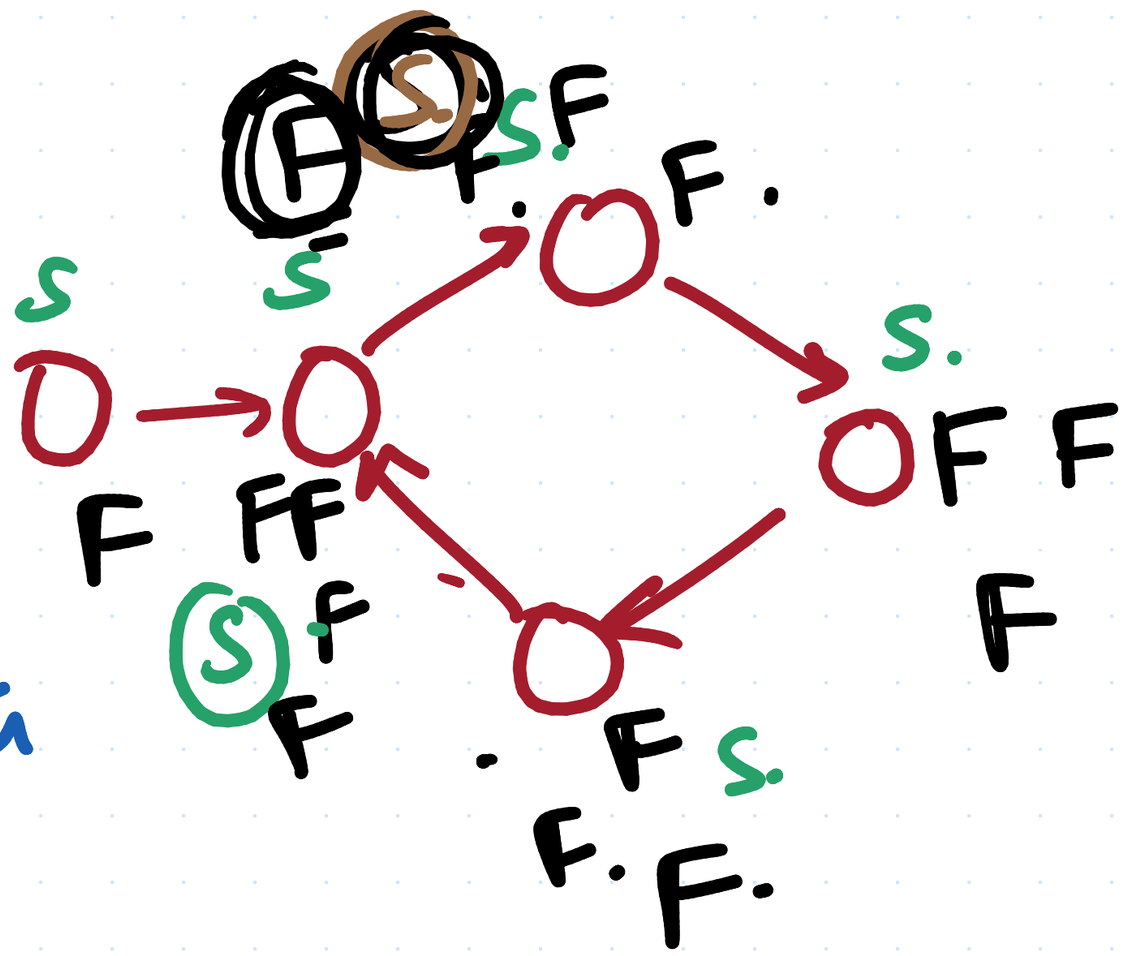
```
def middleOfList(head):

    slow, fast = head, head

    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    return slow. # midpoint.
```

return slow, slow.next.

This works in empty linked list too.

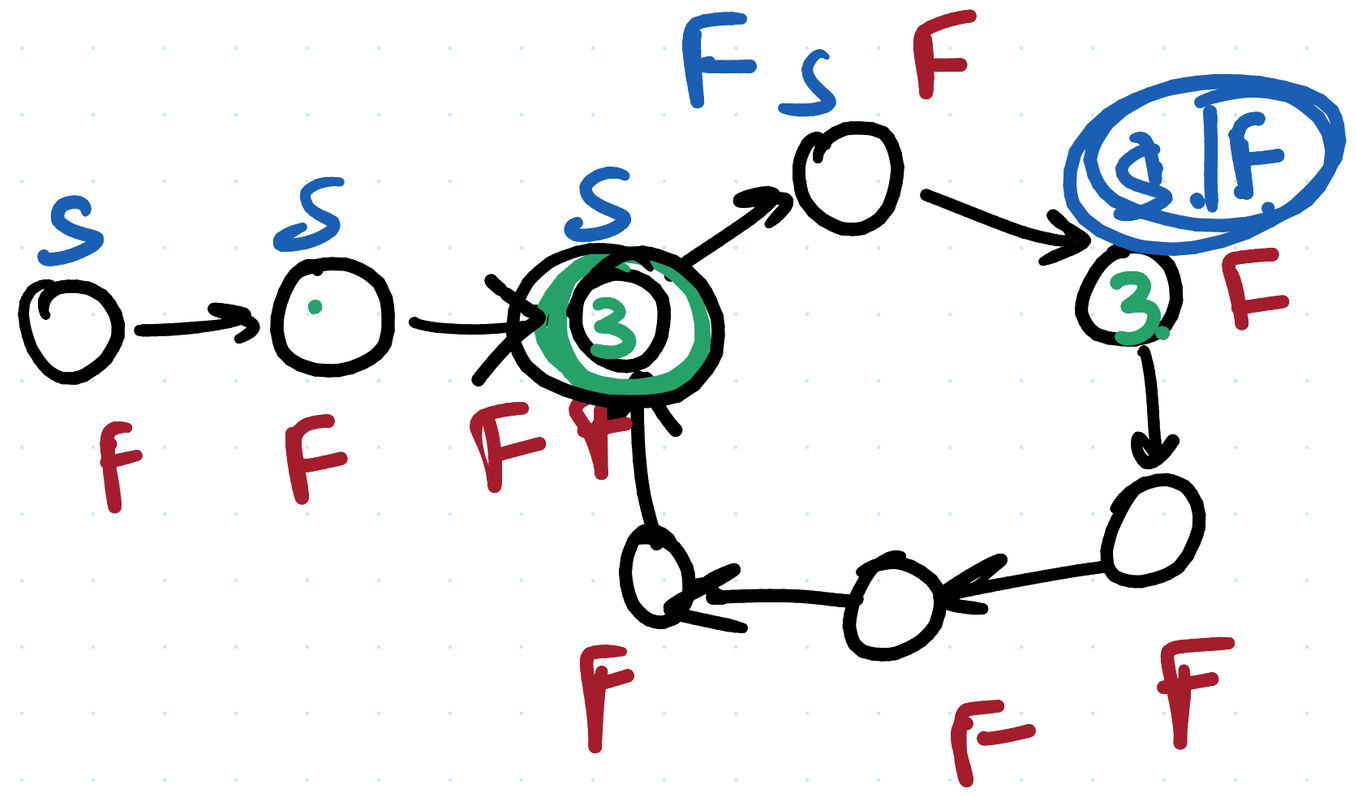


Check whether the linked list has cycle in it?



Downside → O(n)

If there are slow and fast pointers, they will be equal after some time.. & when they are equal we know that there is a cycle.



class LN:

self.val = val

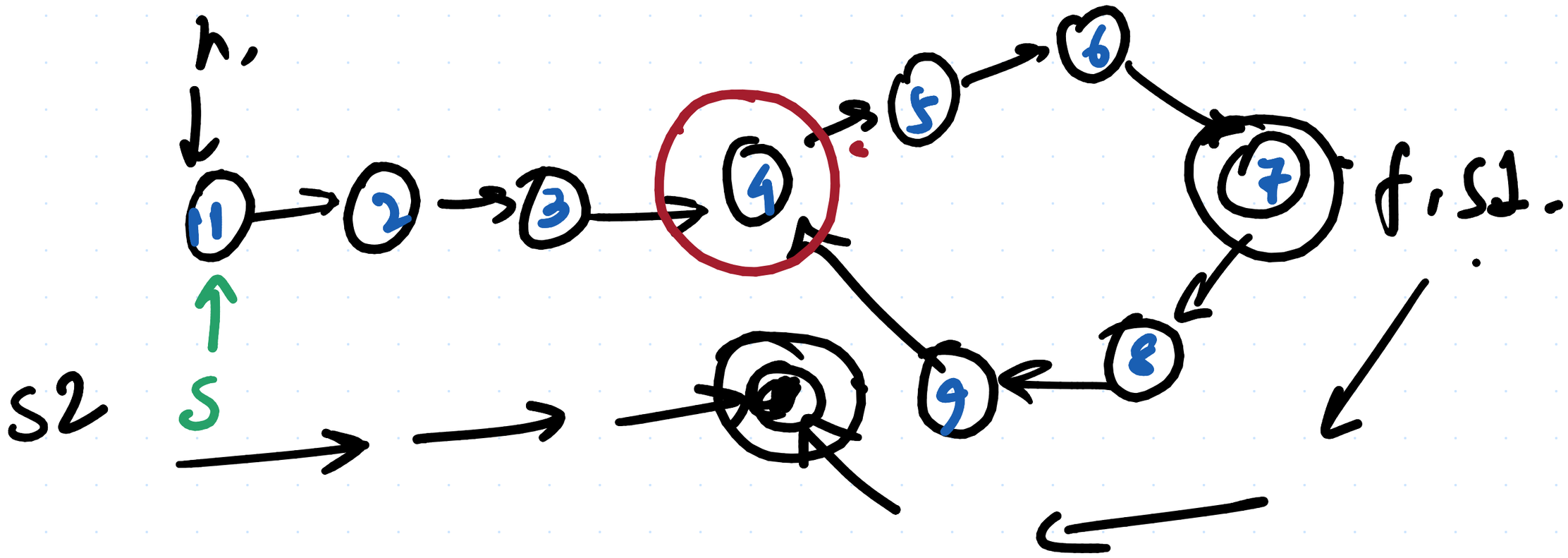
self.next = Null.

{../..}

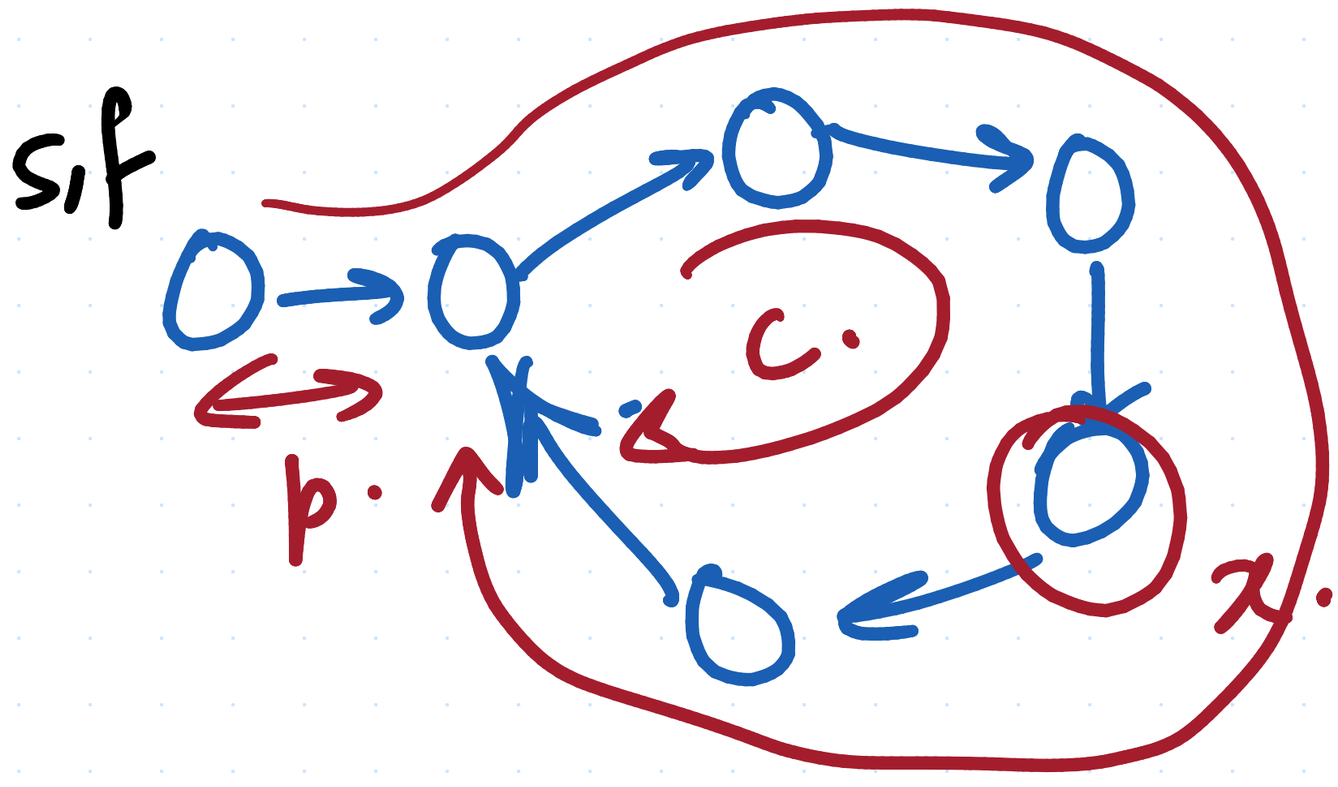
O(n) 2 loop → the fast point.

Code: O(n) time & O(1) space

```
def hasCycle(head):
    slow, fast = head, head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return true.
    return False.
```



show & fast pointer starting from head.
when they intersects; we will
start another show pointer from the
head & we will see where they meet.
will be the begin of the cycle.



cycle of length
c
starting of cycle
to head = p.

$2 \times \text{slow} = \text{fast}$.

$2 \times (p + c - x) = p + (c - x) + c$.

$\therefore p = x$.

O(n) time & O(1) space.

```
def cycleStart(head):
    slow, fast = head, head
    while fast & fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            break
    if not fast or not fast.next:
        return None
    slow2 = head.
```

} detect the loop.

```
while slow != slow2:
    slow = slow.next
    slow2 = slow2.next
return slow
```

Record place to find the beginning of the cycle.