[APC]

1. Reverse Singly LL

① Lay down your Assumptions & Edge cases

1. Not creating a new LL

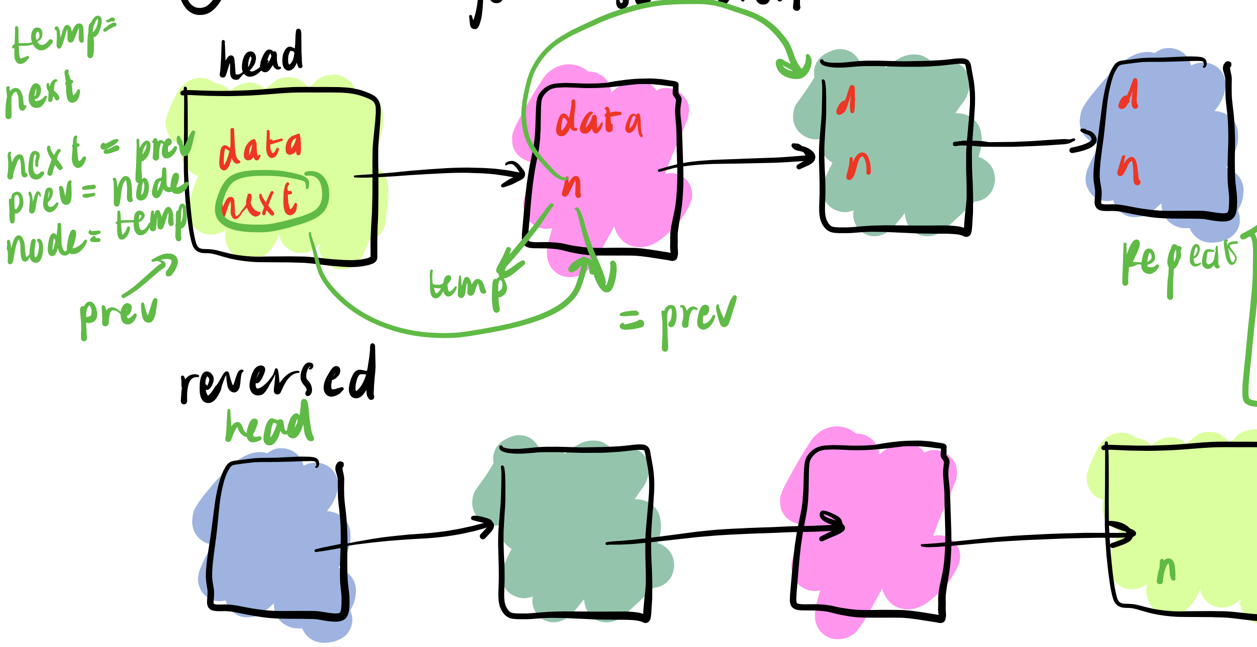
~~② A list is at least 2 nodes~~

2. We can't convert it to DLL

3. LL is empty / 1 node

var dec
→ prev = None
temp = None

② Plan your solution



① store node.next = temp

② node.next = prev

③ prev = node

④ node = temp none

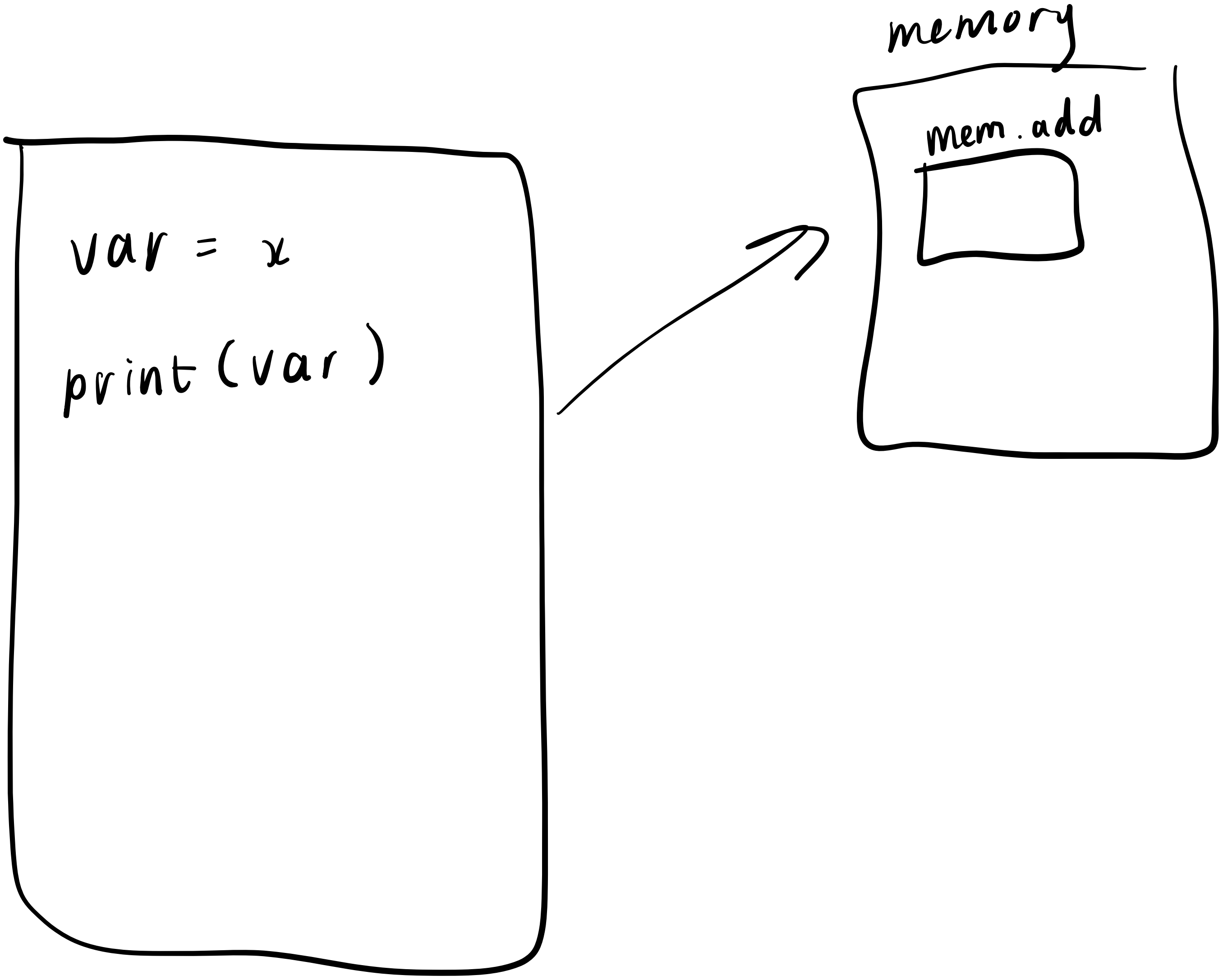
③ Translating plan to code

class Node
→ data
next

class LL
→ head

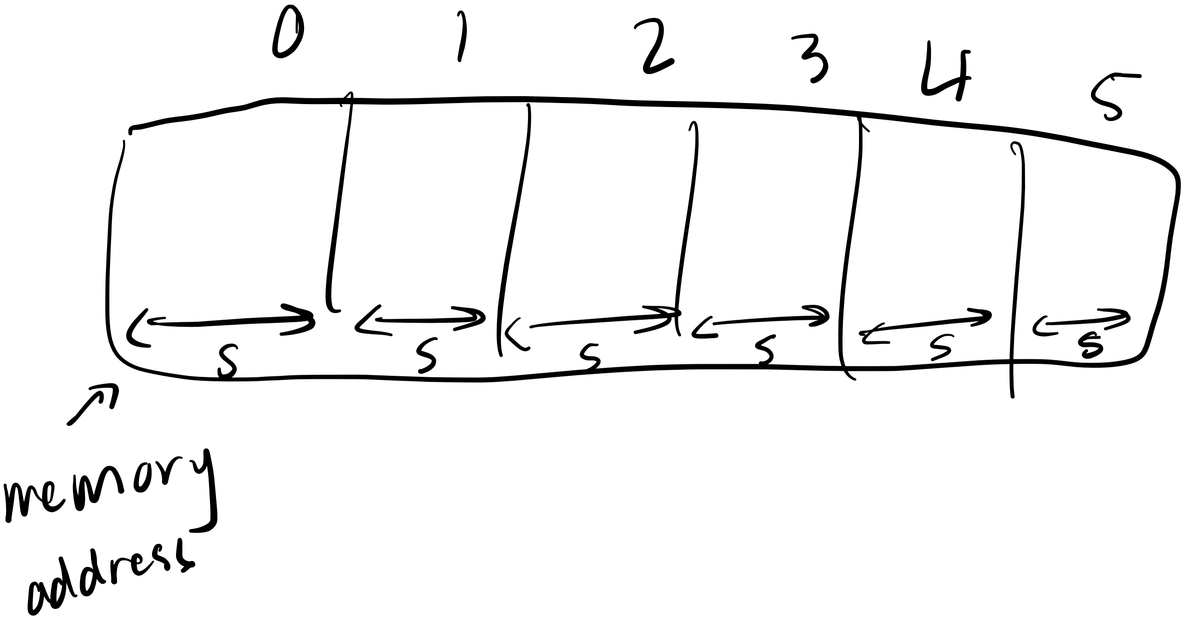
list empty

```
def  reverseSLL ( ll ) :
    node  =  ll. head
    prev  =  None
    temp  =  None
        (! ll.head      )
    if (ll. head == None) :
        return ll
    while (node. next != None):
      temp = node.next
      node. next = prev
      prev = node
      node = temp
     node .next  =  prev
     ll. head = node
```

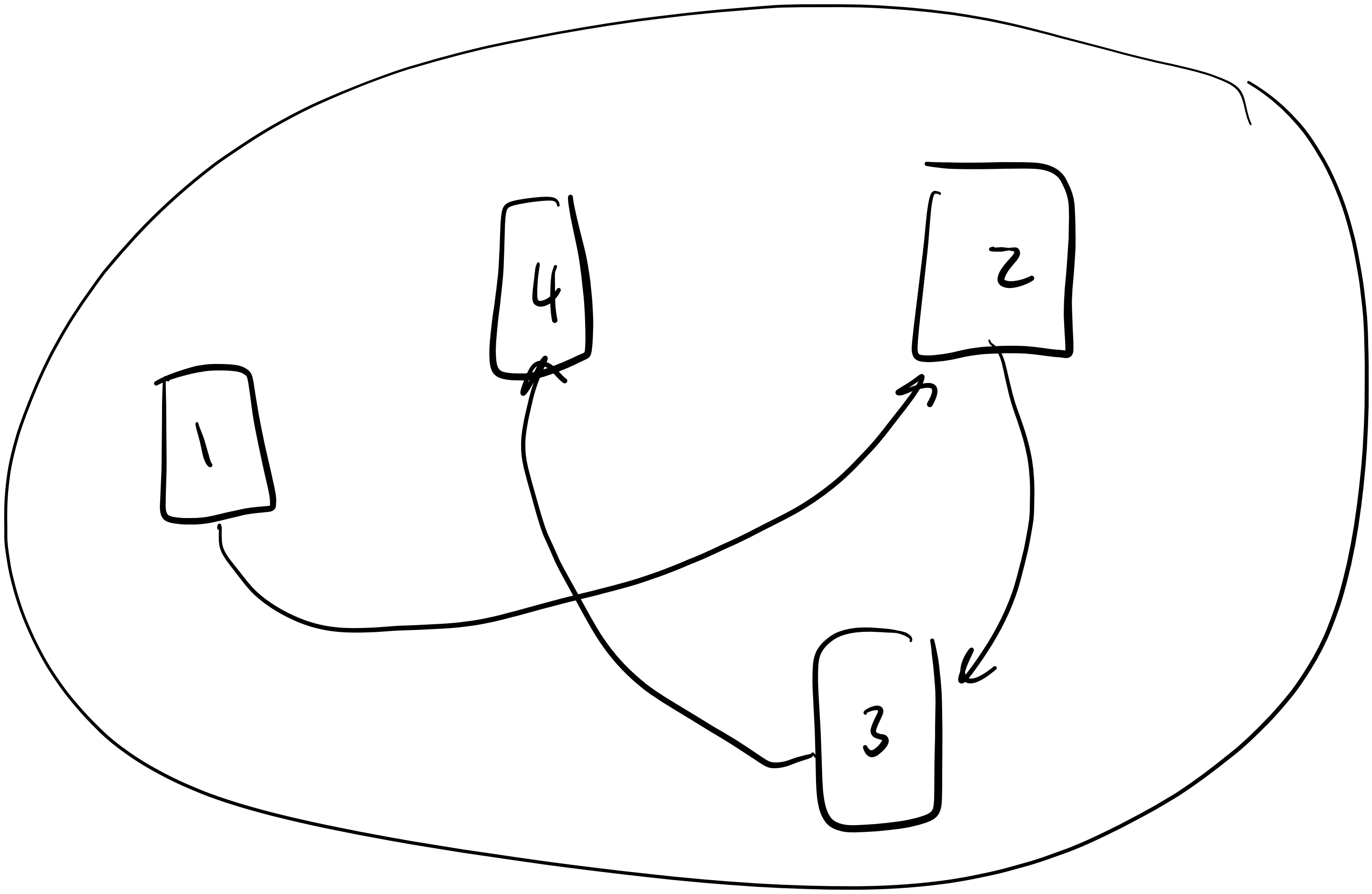


```
var = x

print(var)
```



```
array[2]
```



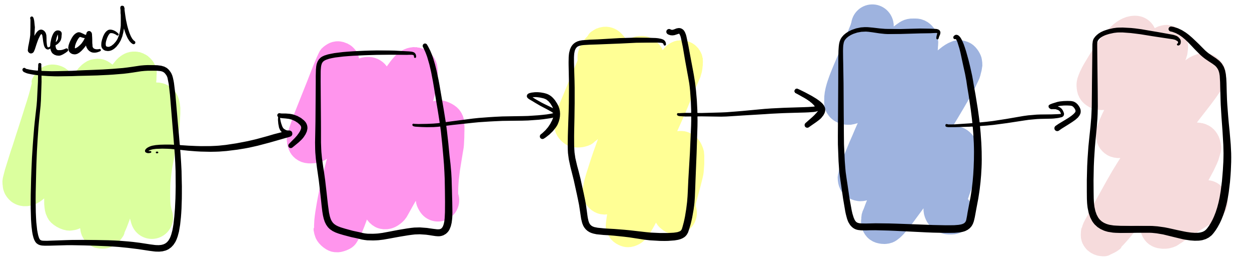
2. finding the middle LL

① Lay down your assumptions

1. LL could be very large
2. Singly LL
3. Edge cases: Empty LL , LL with 1 node
4. If our list is even
   ↳ return 2 nodes → [list] (tuple)

   If our list is odd
   ↳ return 1 node

② Plan — order o(n) $o(n + \frac{1}{2}n) = o(n)$

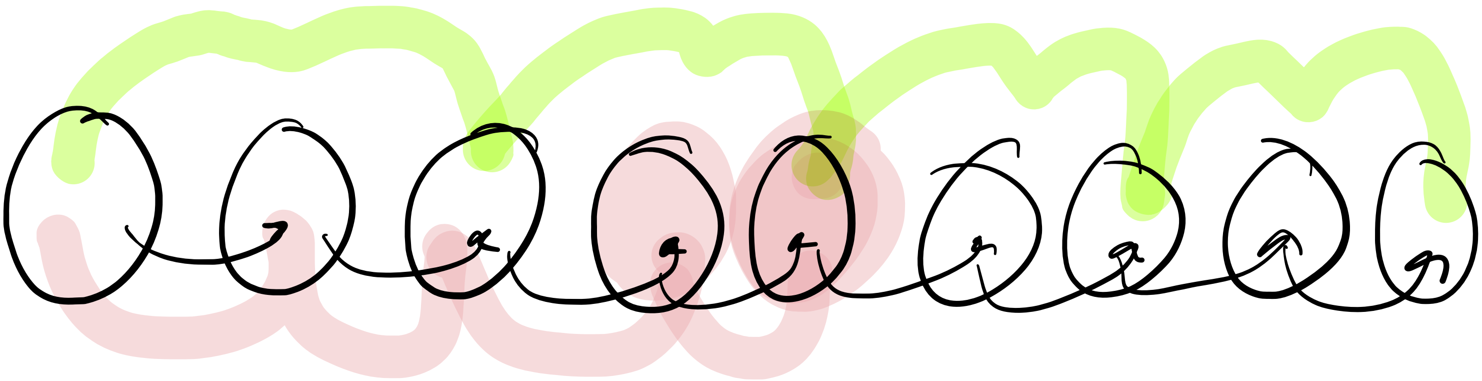


① Time complexity issue waste time traversing

② Space complexity issue waste memory storing a copy of the LL

③ · Not create a new list

· Not traverse twice



1. $\overset{\text{check}}{\underline{\text{fnode}}}$.next == None   return (snode)

2. $\overset{\text{check}}{\underline{\text{fnode}}}$.next.next == None.   return (snode, snode.next)

3. snode = snode.next

fnode = fnode.next.next

③ Code

```
def middleFinder(ll):
    slow = ll.head
    fast = ll.head
    if (ll.head == None):
        return None
    if (ll.head.next == None):
        return ll.head
    while (fast.next != None) and
          (fast.next.next != None)
        fast = fast.next.next
        slow = slow.next
    if (fast.next == None):
        return slow
    return [slow, slow.next]
```