

[APC]

1. Reverse Singly LL

① Lay down your Assumptions & Edge cases

1. Not creating a new LL

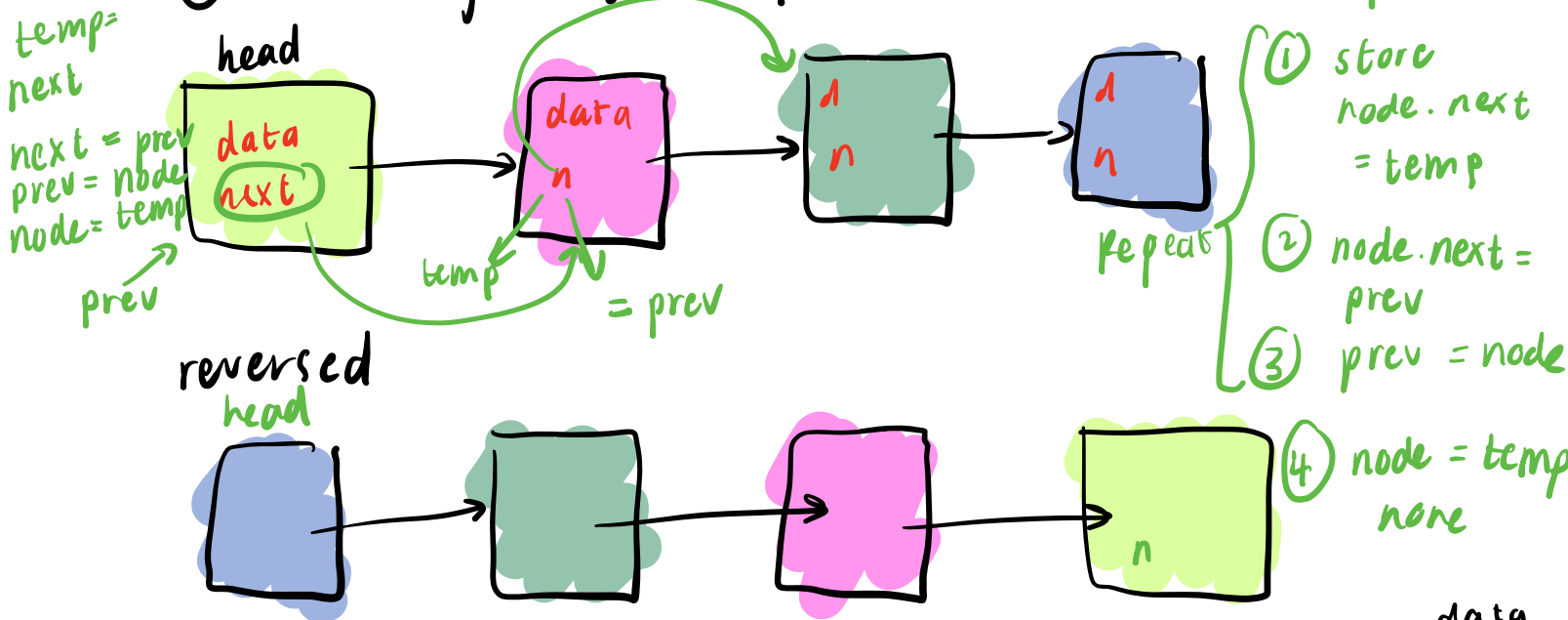
② A list is at least 2 nodes

2. We can't convert it to DLL

3. LL is empty / 1 node

var dec
→ prev = None
temp = None

② Plan your solution



③ Translating plan to code

```
class Node
class LL
    → head
```

```
def reverseSLL (ll) :
```

```
    node = ll.head
```

```
    prev = None
```

```
    temp = None
```

list empty {

```
        (! ll.head )
```

```
    if (ll.head == None) :
```

```
        return ll
```

```
    while (node.next != None) :
```

```
        temp = node.next
```

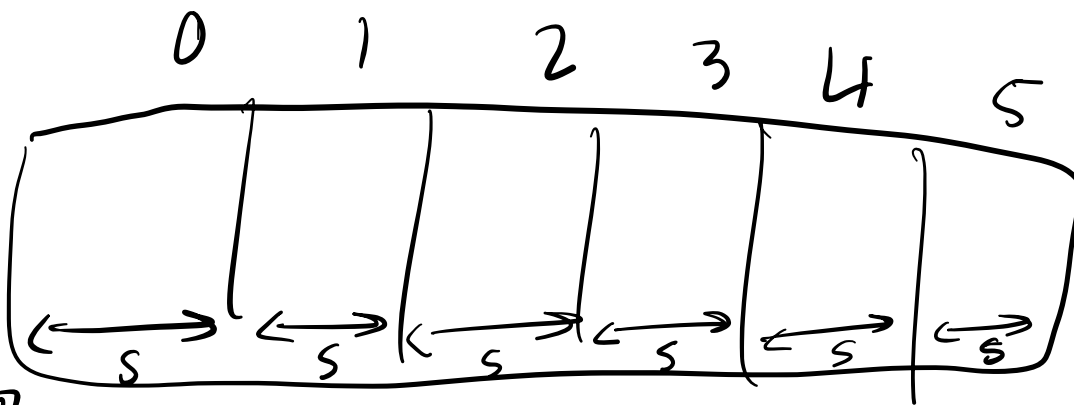
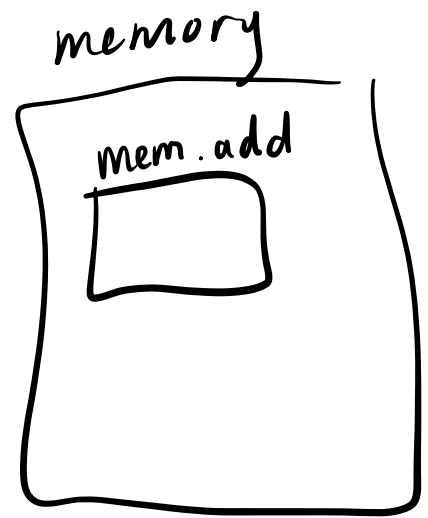
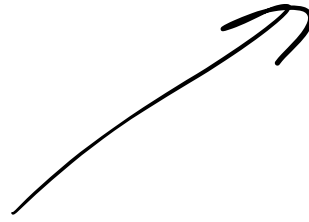
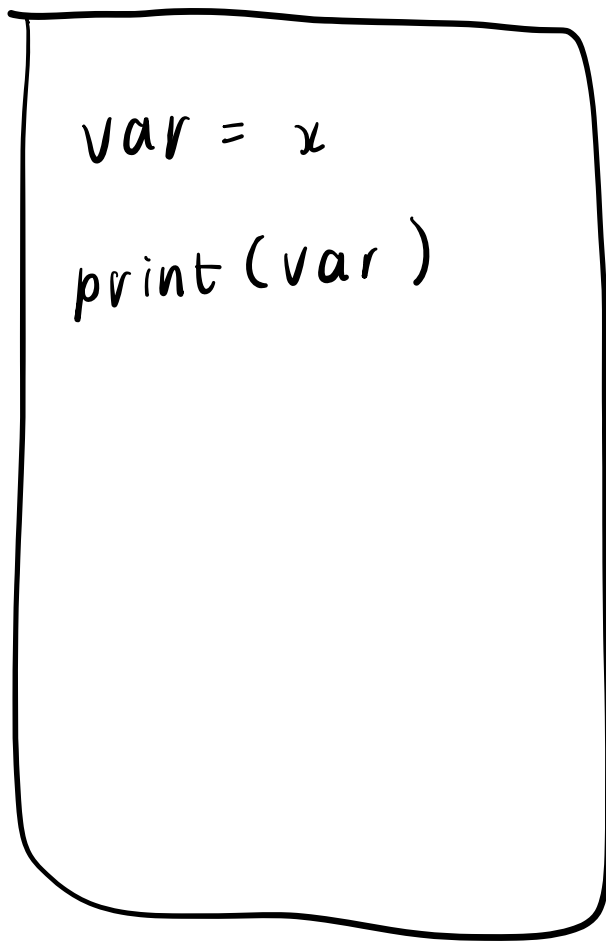
```
        node.next = prev
```

```
        prev = node
```

```
        node = temp
```

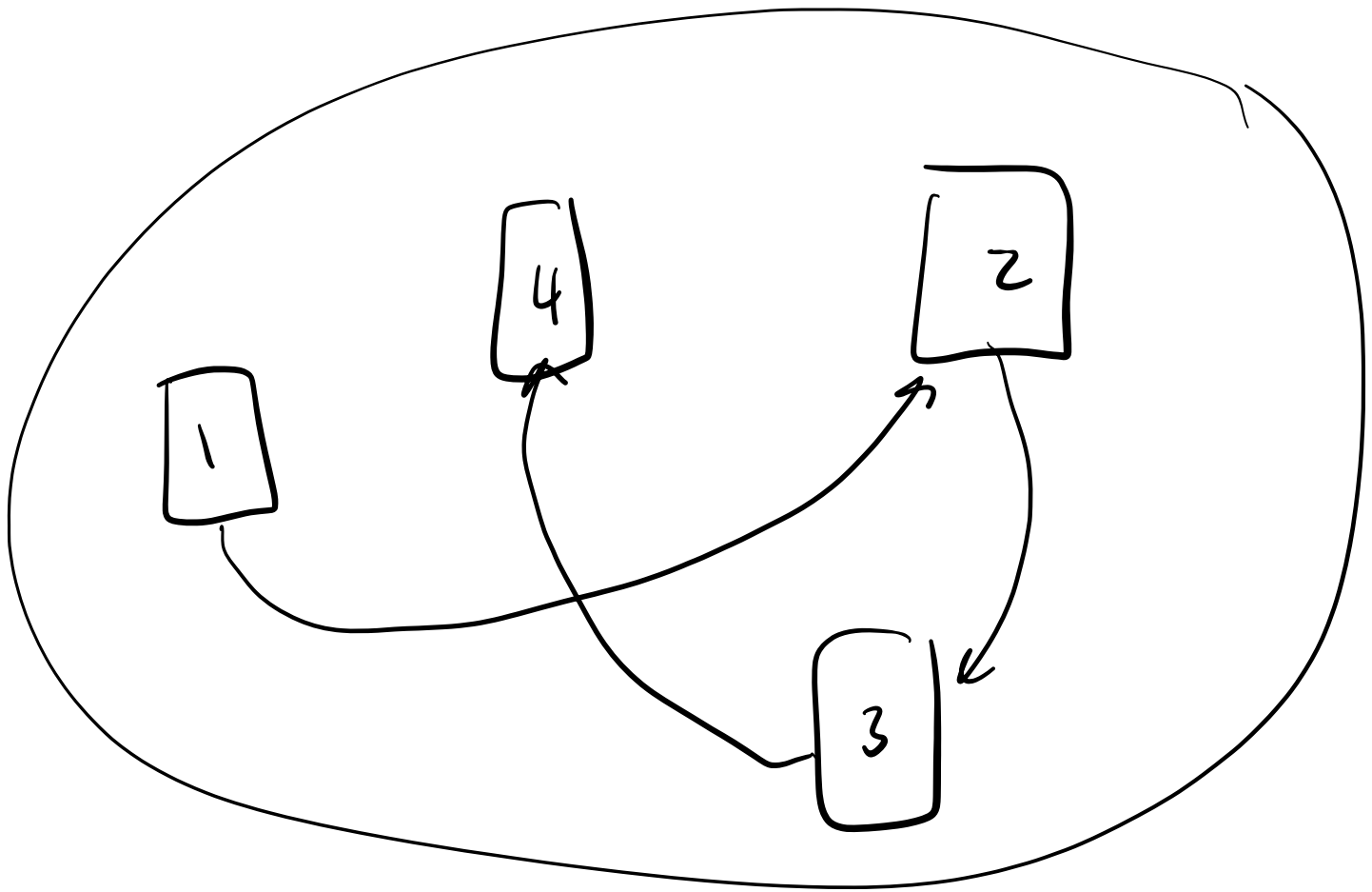
```
    node.next = prev
```

```
    ll.head = node
```



memory
address

array[2]



2. Finding the middle LL

① Lay down your assumptions

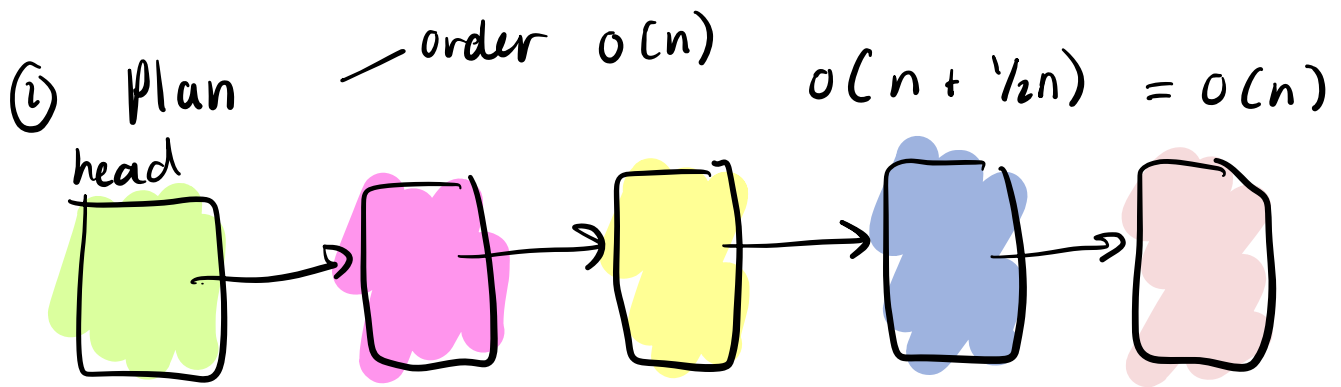
1. LL could be very large
2. Singly LL
3. Edge cases: Empty LL, LL with 1 node
4. If our list is even

↳ return 2 nodes

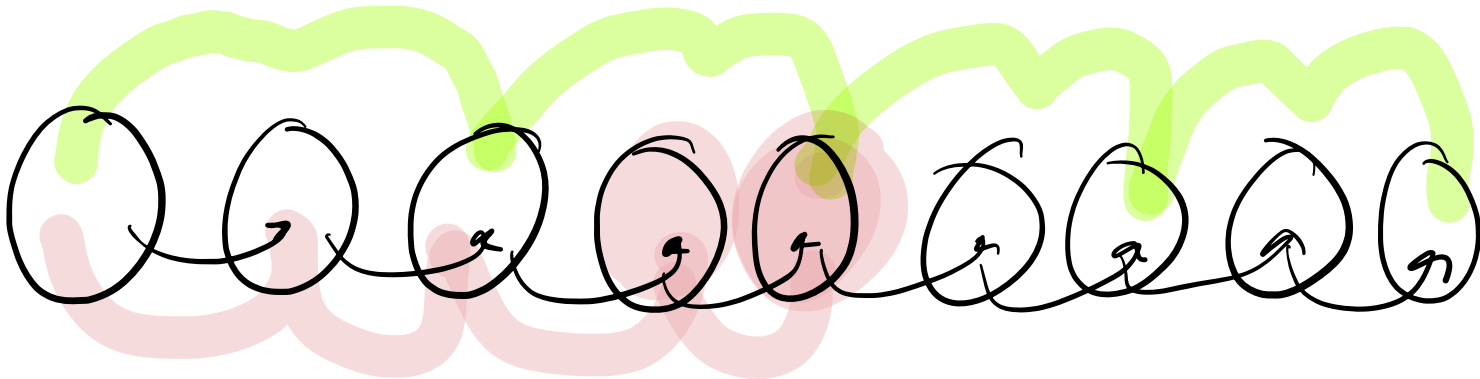
→ [list]
(tuple)

If our list is odd

↳ return 1 node



- 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



- check
1. fnode.next == None return (snode)
 2. 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]
```