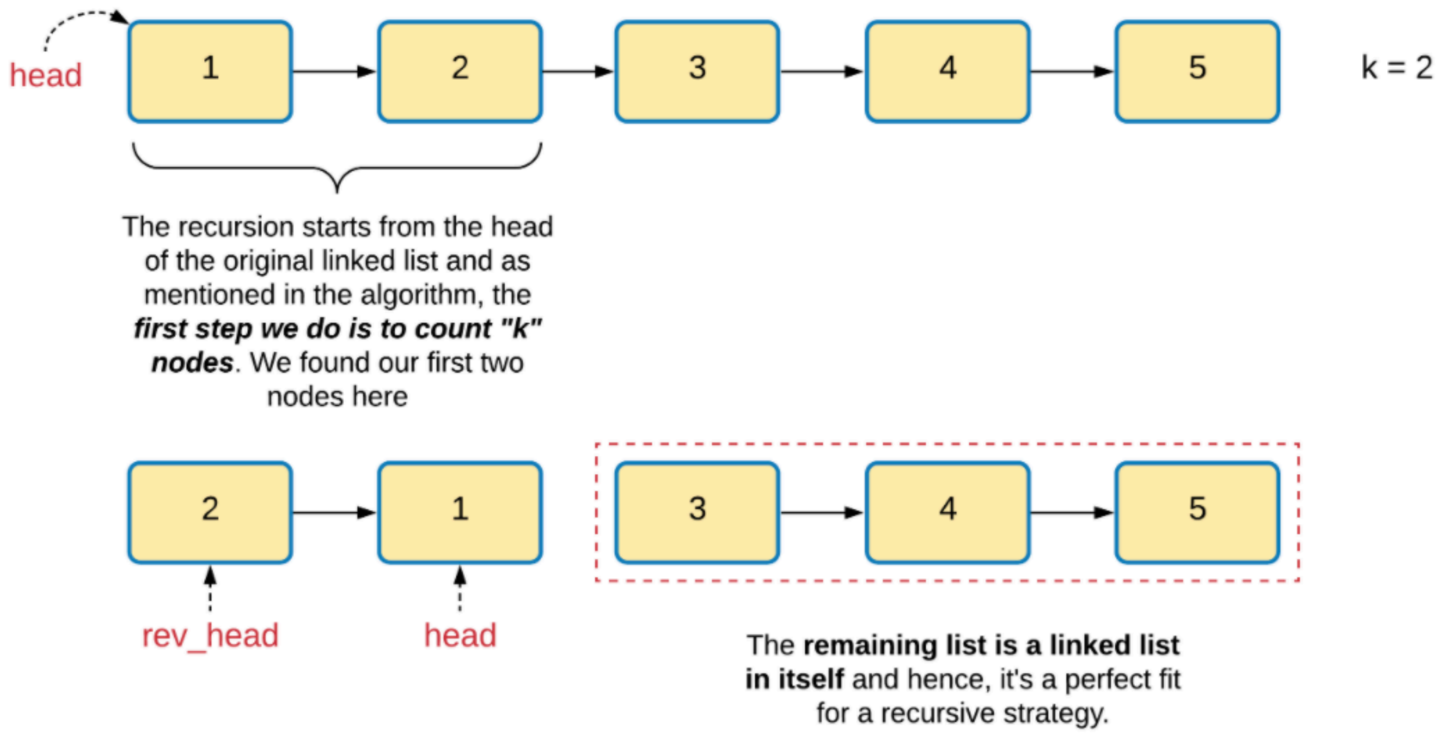
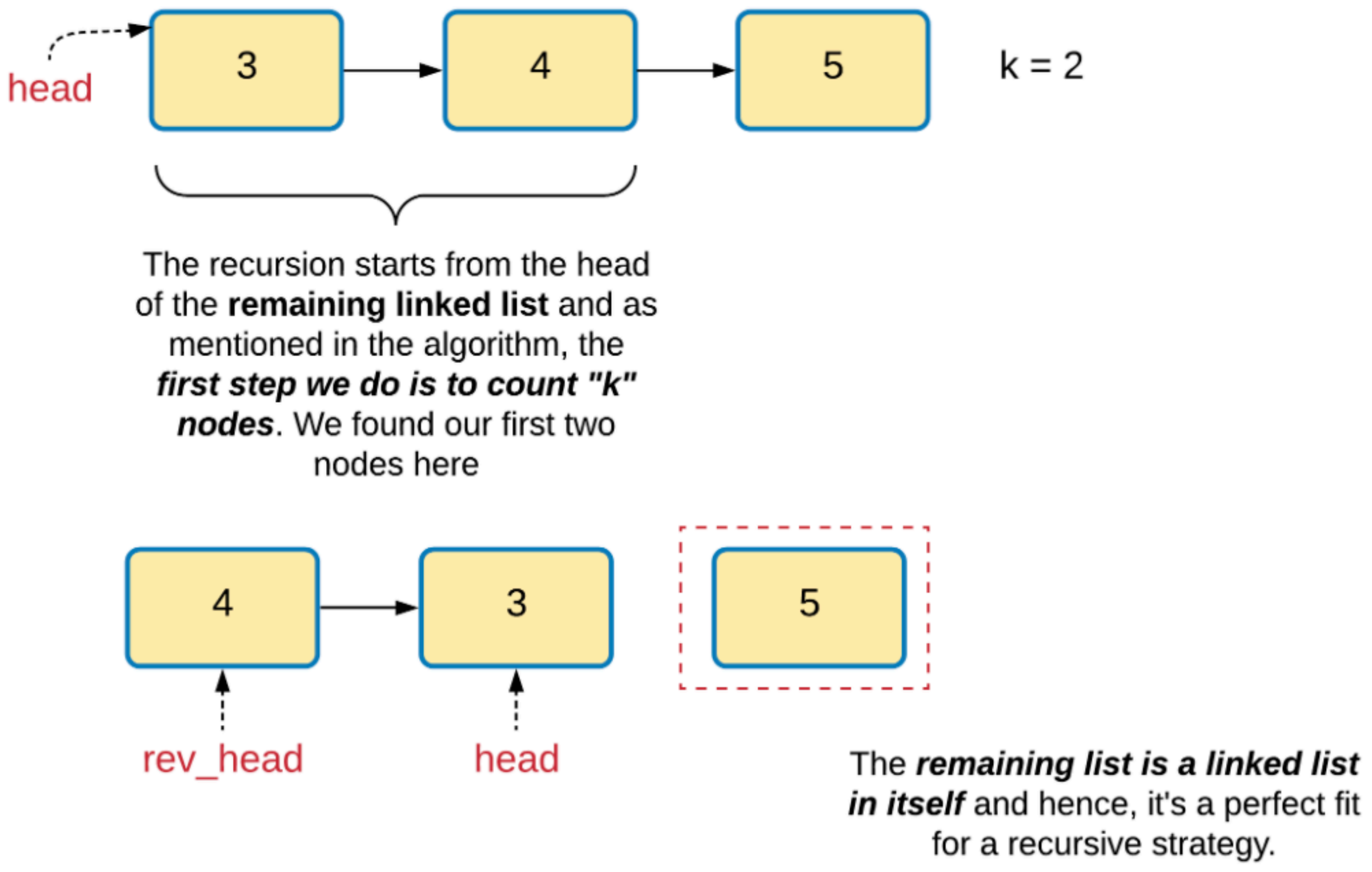
Algorithm

1. Assuming we have a `reverse()` function already defined for a linked list. This function would take the head of the linked list and also an integer value representing `k`. We don't have to reverse till the end of the linked list. Only `k` nodes are to be touched at a time.
2. In every recursive call, we first count the number of nodes in the linked list. As soon as the count reaches `k`, we break.
3. If there are less than `k` nodes left in the list, we return the head of the list.
4. However, if there are at least `k` nodes in the list, then we reverse these nodes by calling our `reverse()` function defined in the first step.
5. Our recursion function needs to return the head of the reversed linked list. This would simply be the k^th nodes in the list passed to the recursion function because after reversing the first `k` nodes, the k^th node will become the new head and so on.
6. So, in every recursive call, we first reverse `k` nodes, then recurse on the rest of the linked list. When recursion returns, we establish the proper connections.

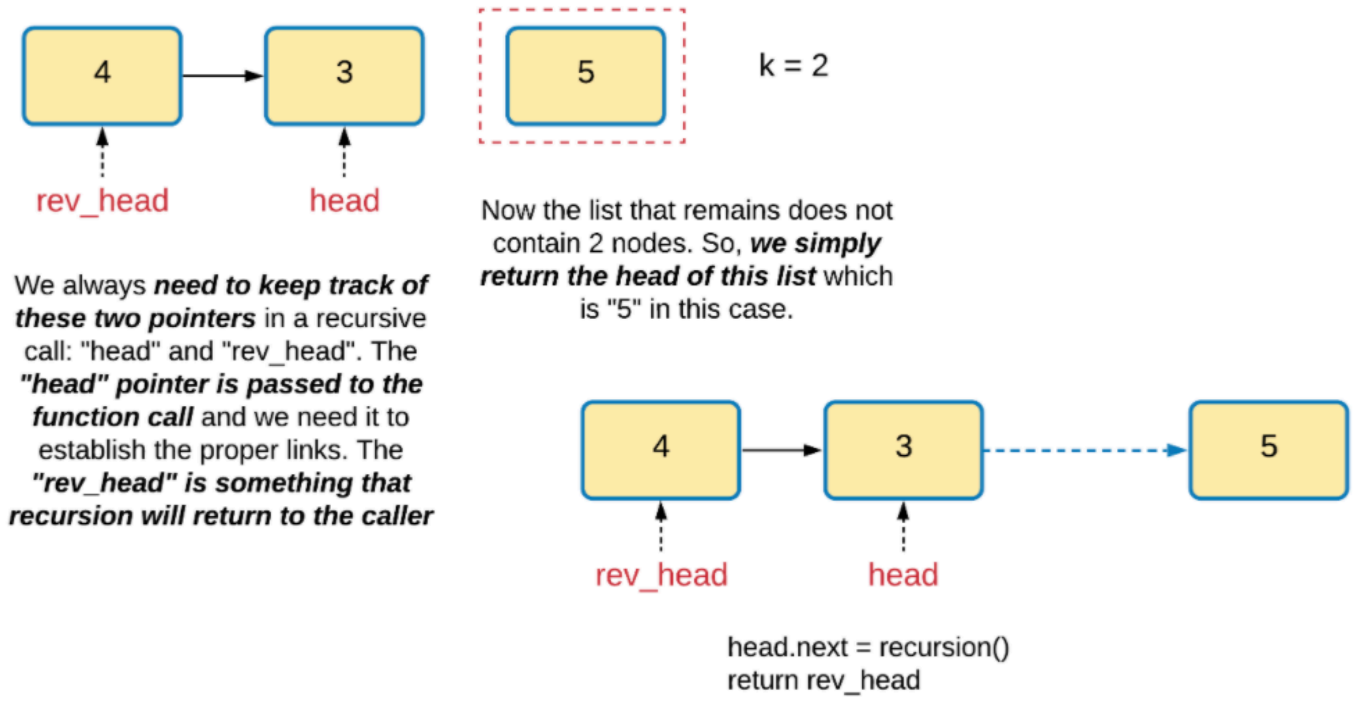
Let's look at a quick example of the algorithm's dry run. So, in the first recursive step, we process the first two nodes of the list and then make a recursive call.


head
1
2
3
4
5
k = 2
The recursion starts from the head of the original linked list and as mentioned in the algorithm, the ***first step we do is to count "k" nodes***. We found our first two nodes here
2
1
3
4
5
rev_head
head
The **remaining list is a linked list in itself** and hence, it's a perfect fit for a recursive strategy.

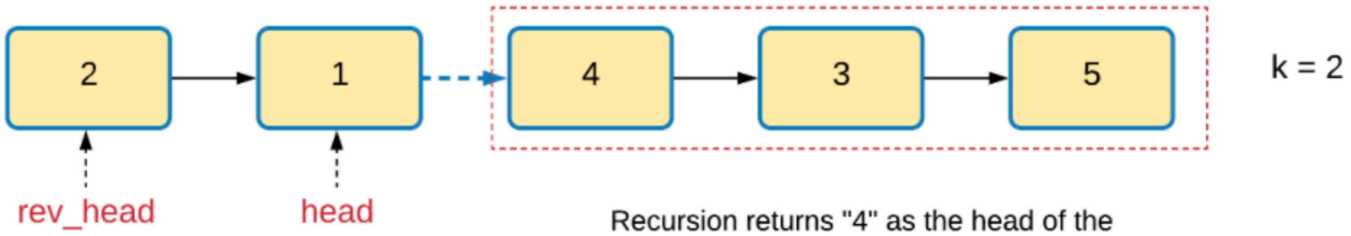
Here again, we process the two nodes and then make the final recursive call for this example linked list.


head
3
4
5
k = 2
The recursion starts from the head of the **remaining linked list** and as mentioned in the algorithm, the ***first step we do is to count "k" nodes***. We found our first two nodes here
4
3
5
rev_head
head
The ***remaining list is a linked list in itself*** and hence, it's a perfect fit for a recursive strategy.

Now here we don't have enough nodes to reverse. So, in the recursive call we simply return the only remaining node here which is "5". Once that node is returned from the recursive call, we need to establish the proper connections i.e. from 3->5.


4
3
5
k = 2
rev_head
head
Now the list that remains does not contain 2 nodes. So, ***we simply return the head of this list*** which is "5" in this case.
We always ***need to keep track of these two pointers*** in a recursive call: "head" and "rev_head". The ***"head" pointer is passed to the function call*** and we need it to establish the proper links. The ***"rev_head" is something that recursion will return to the caller***
4
3
5
rev_head
head
head.next = recursion()
return rev_head

Similarly, recursion would return `4` as the new head node of the modified list ahead. We need to establish the connection from `1` to `4` and then return `2` as the head of the modified list.



Recursion returns "4" as the head of the modified list. The best part about recursion is that ***when a recursive call returns, we know that all the hard work has already been done*** and we just need to do some minor stick work and return