Linked List

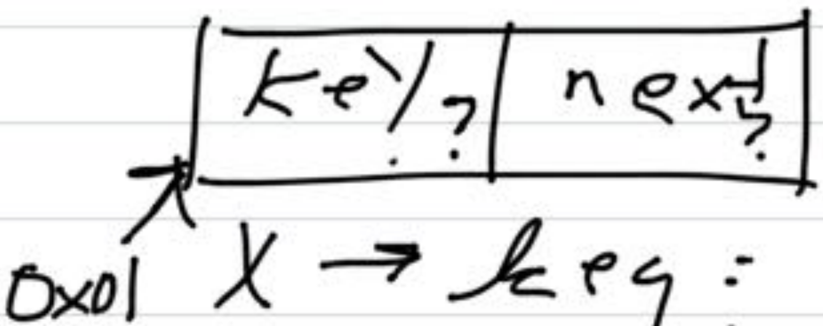
Insert
Delete
Search

Most important skill:
Iterating through list

Once you understand
that, actions to insert,
etc are easier to understand

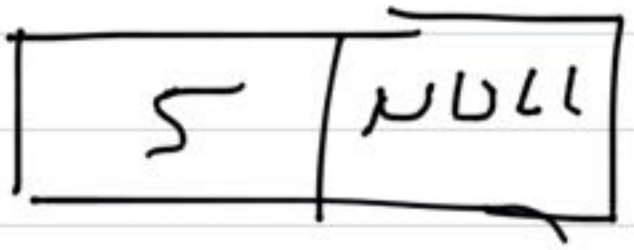
```
struct node{
    int key
    node *next
}
```

node *X = new node

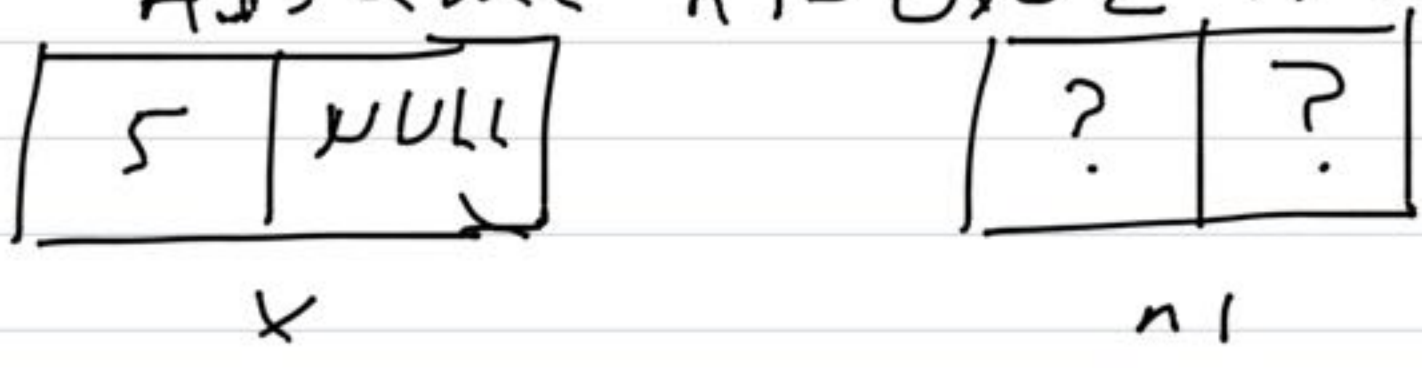


Assume X = 0x01 is address of X

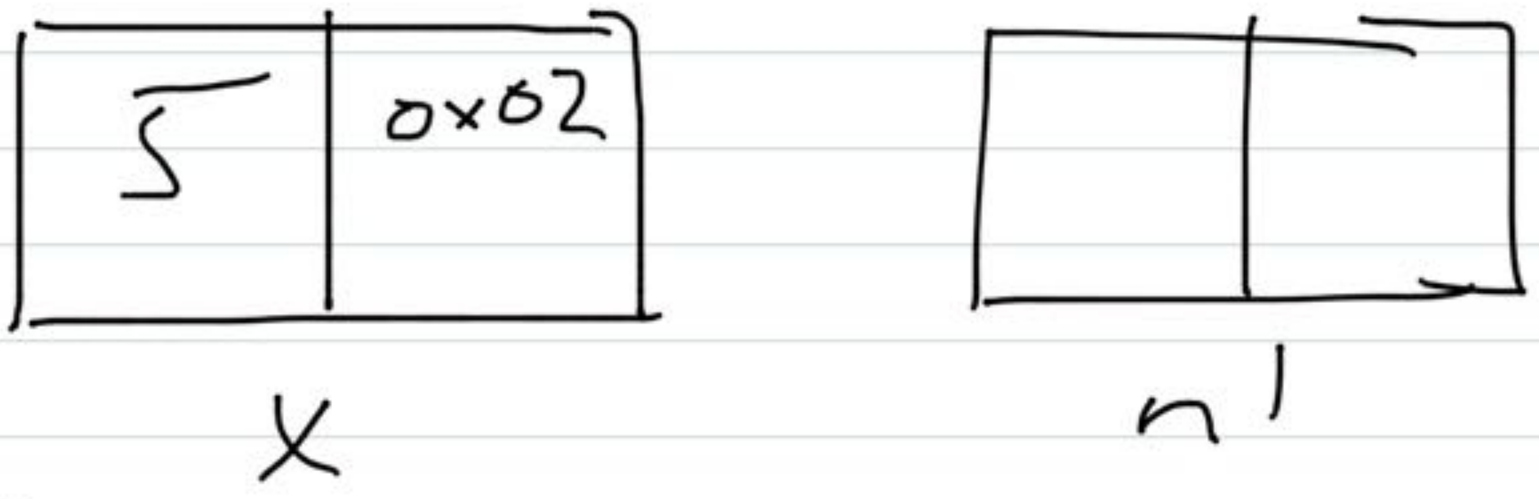
X → key = 5
X → next = NULL



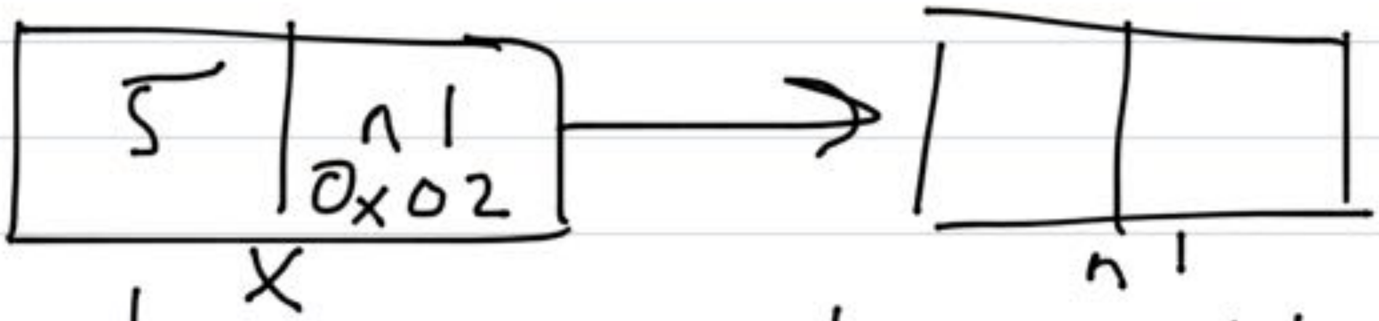
node *n1 = new node
Assume n1 = 0x02 is address of n1



X → next = n1



We put address of n1 at X→next, and we think of it as

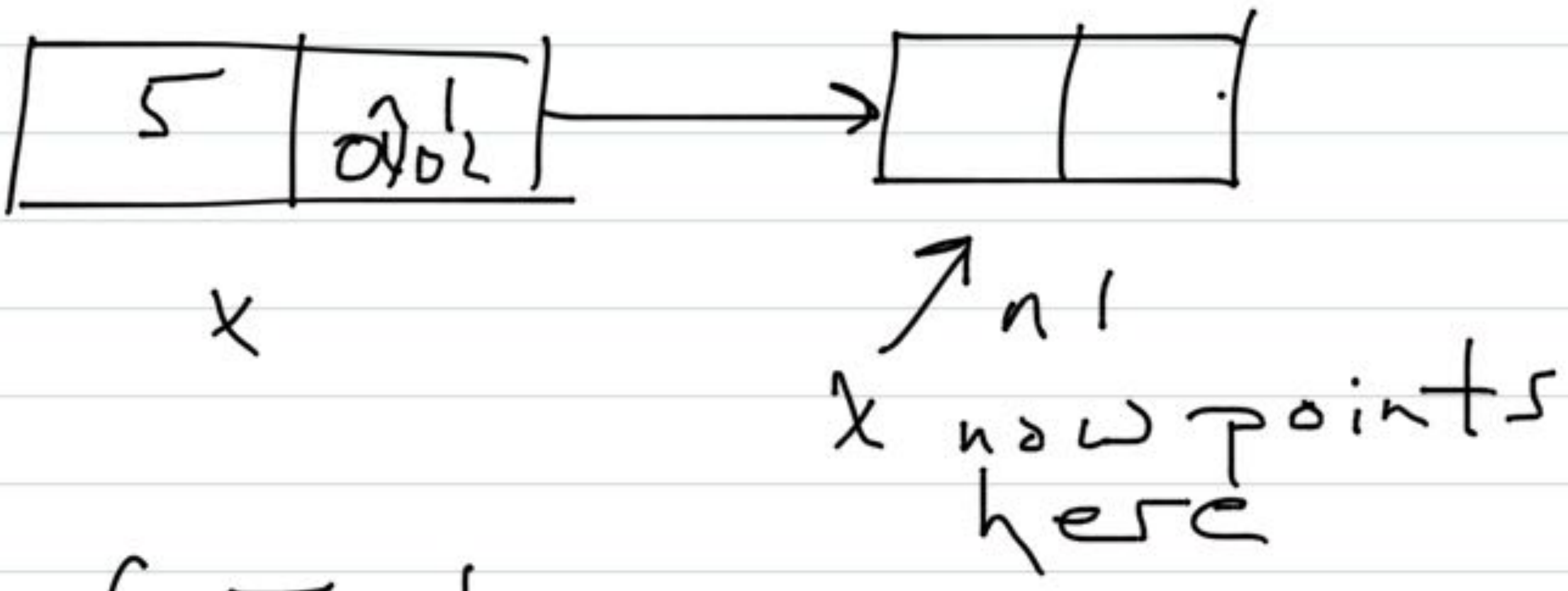


The → doesn't have meaning, its just representation. Shows possible movement direction. X knows n1.
X is the address of the node, so we can change it to point to something else

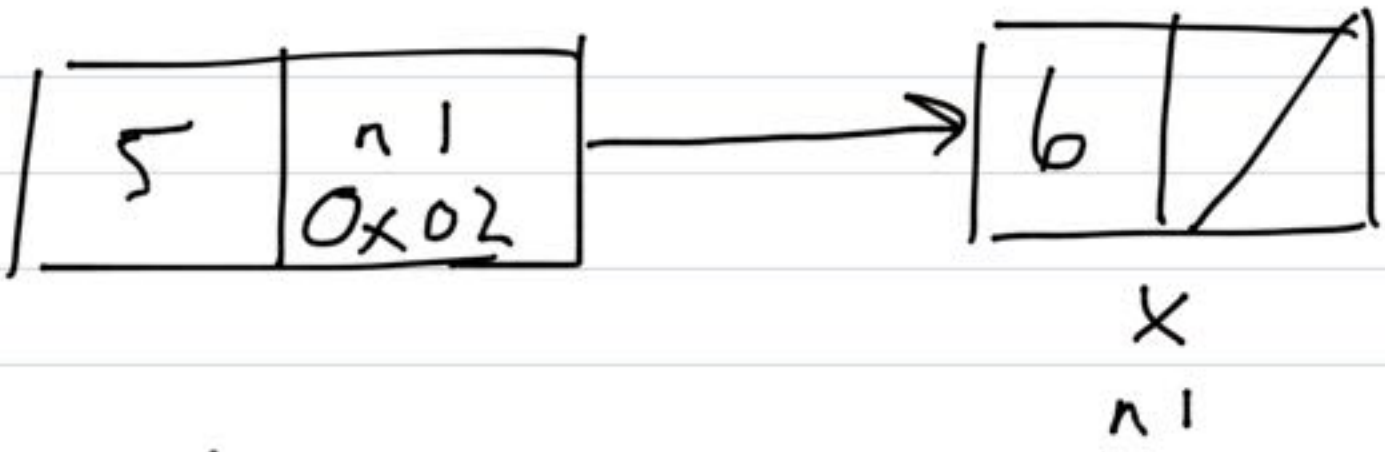
Saying x = X→next

n1 doesn't know X

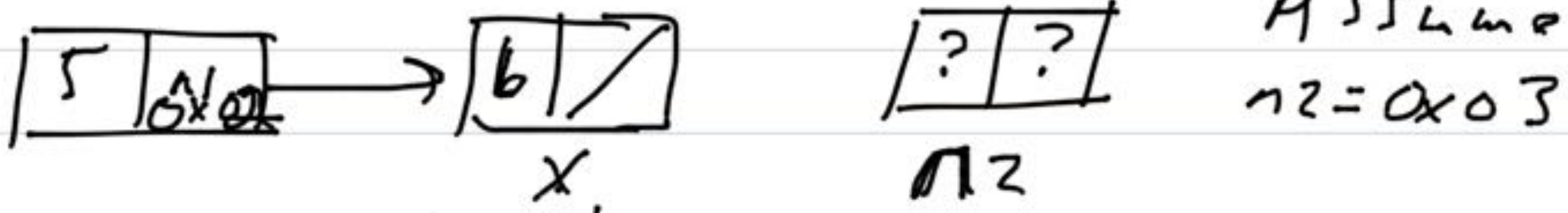
is same as x = n1



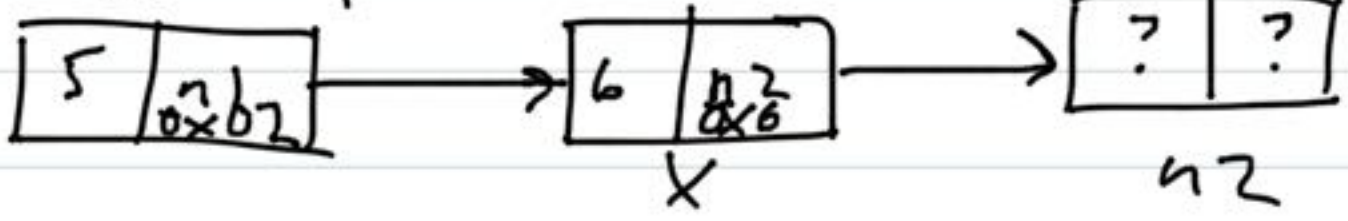
if I do
x → key = 6
x → next = NULL
then the list holds

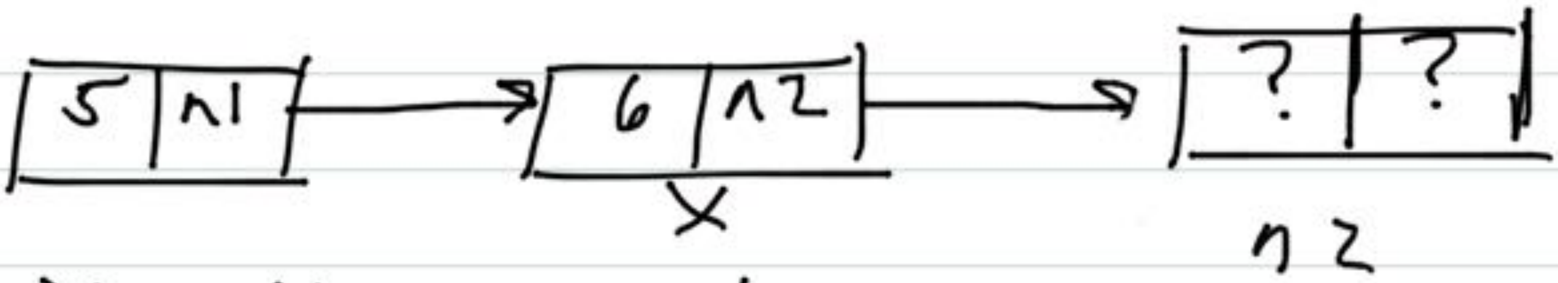


node *n2 = new node

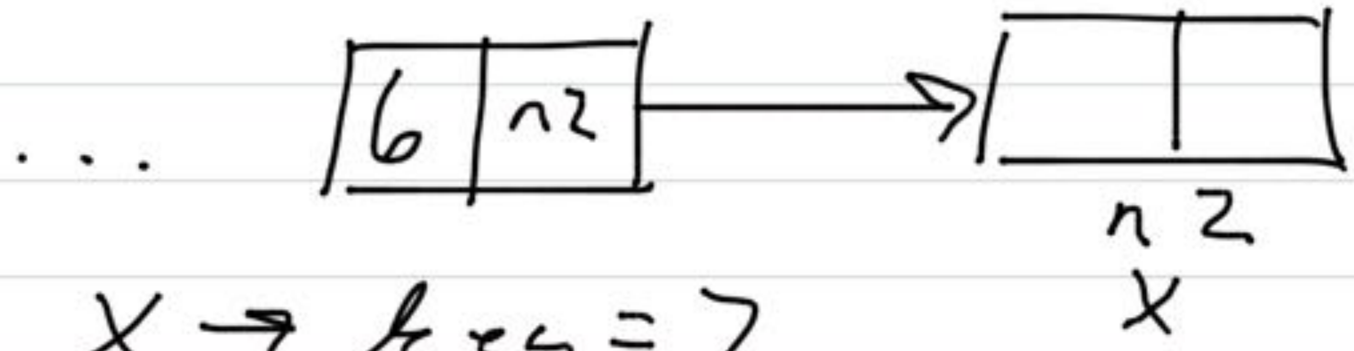


x → next = n2

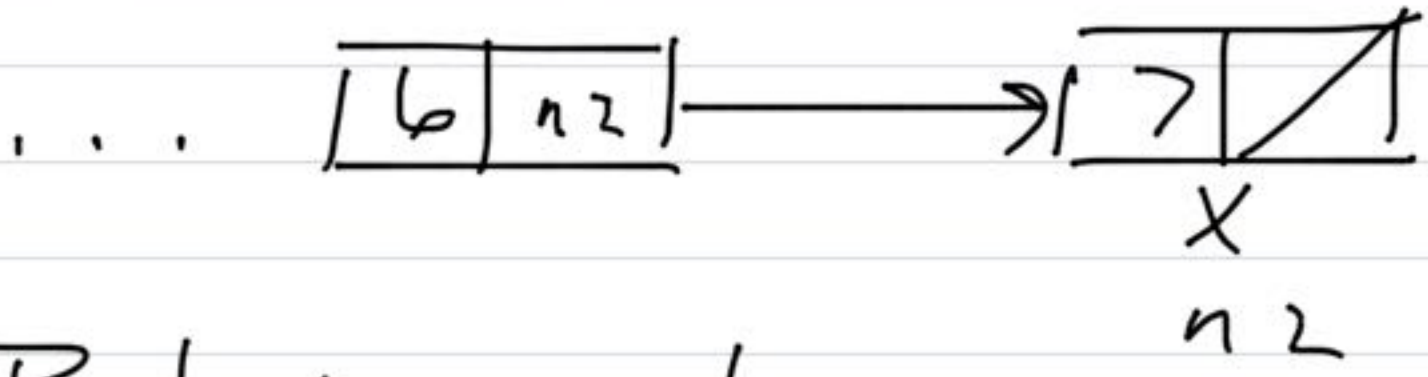




X = X → next
Same as X = n2

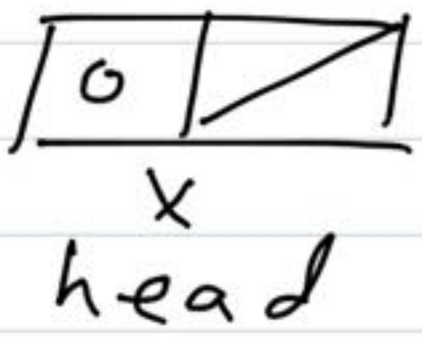


X → key = 7
X → next = NULL



Put In a loop

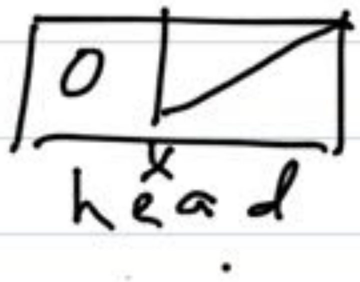
Assume empty list to start
node *X = new node
x → key = 0
x → next = NULL
Store head of list
node *head = X



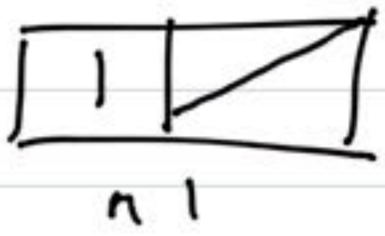
head has no memory of own, points to memory alloc for X.

```
i = 1
while(i<5) :
1     node *n1 = new node
2     n1→key = i
3     n1→next = NULL
4     X→next = n1
5     X = X→next
6     i++
```

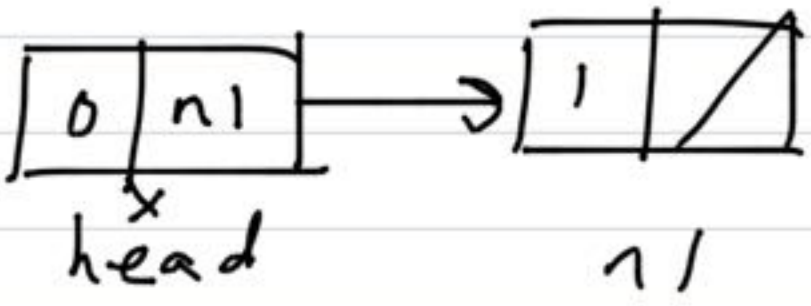
When i = 1



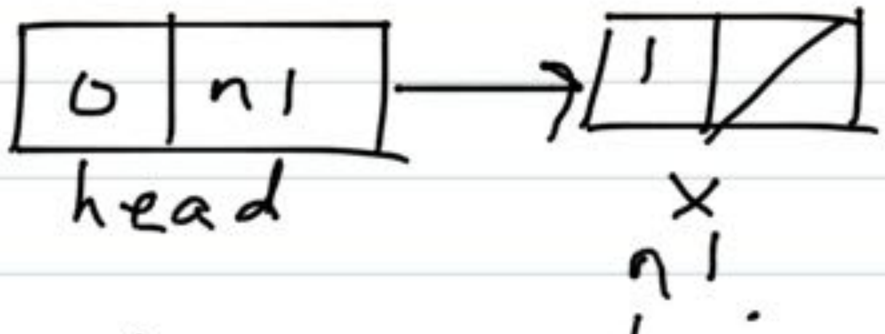
starting condition outside loop



Lines 1-3



Line 4



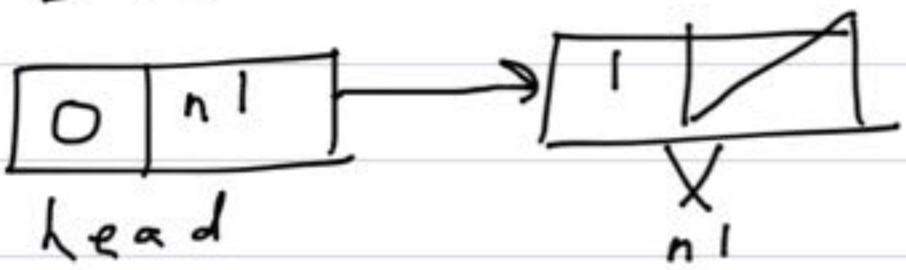
Line 5

increment i, i = 2   Line 6

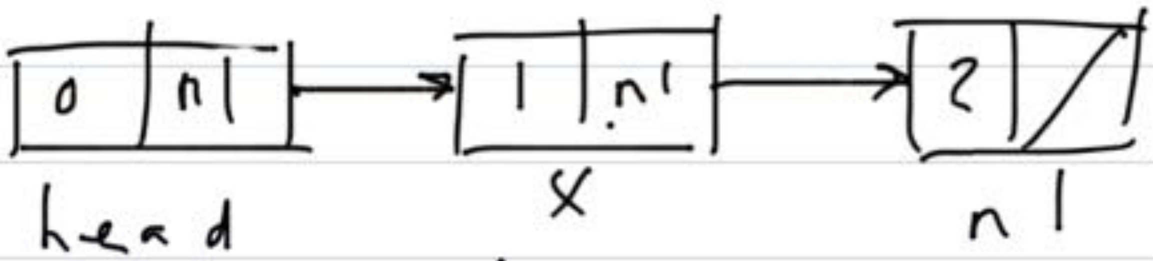
```
  i=1
  while(i<5) :
1     node *n1 = new node
2     n1→key = i
3     n1→next=NULL
4     x→next=n1
5     x = x→next
6     i++
```

Second loop iteration

i = 2

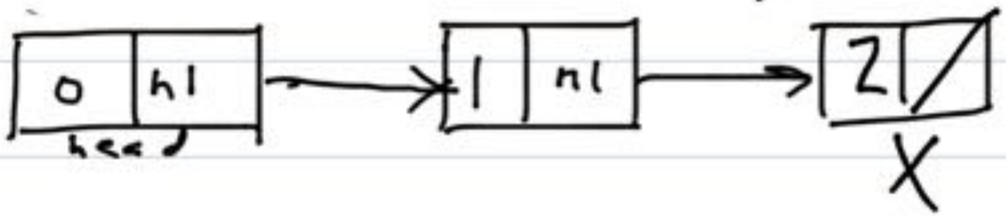


starting condition



Lines 1 - 4

We have two n1 in this list, but they're not the same. Its the address assigned on line 1 in the loop, which changes each iteration



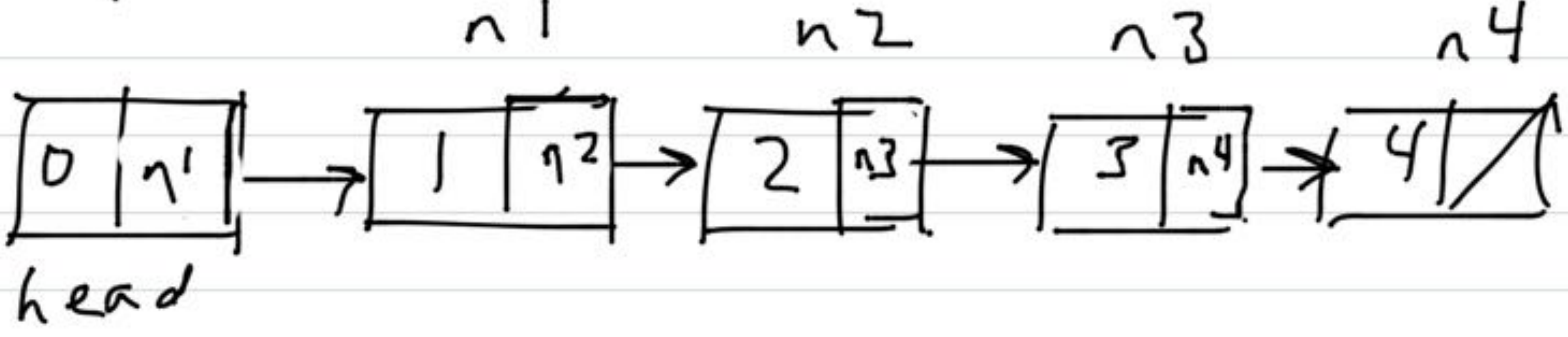
Line 5

increment i , i = 3   Line 6

and so on until i = 5.

Once we have list, how do we search it.

Assume we have this:



The n1, n2, n3, n4 represent the address where the node was created, not variable but addr.

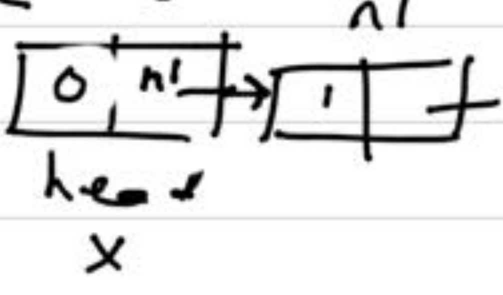
Traverse list to find Key = 2

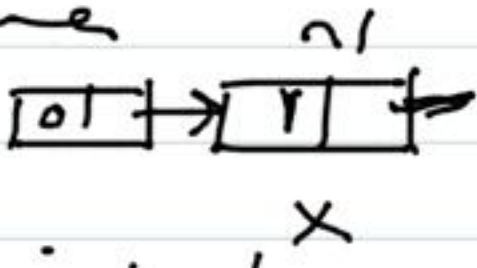
Search

Start at head. same technique for ~~deleting~~

Need a pointer to move between nodes, call it x and declare as

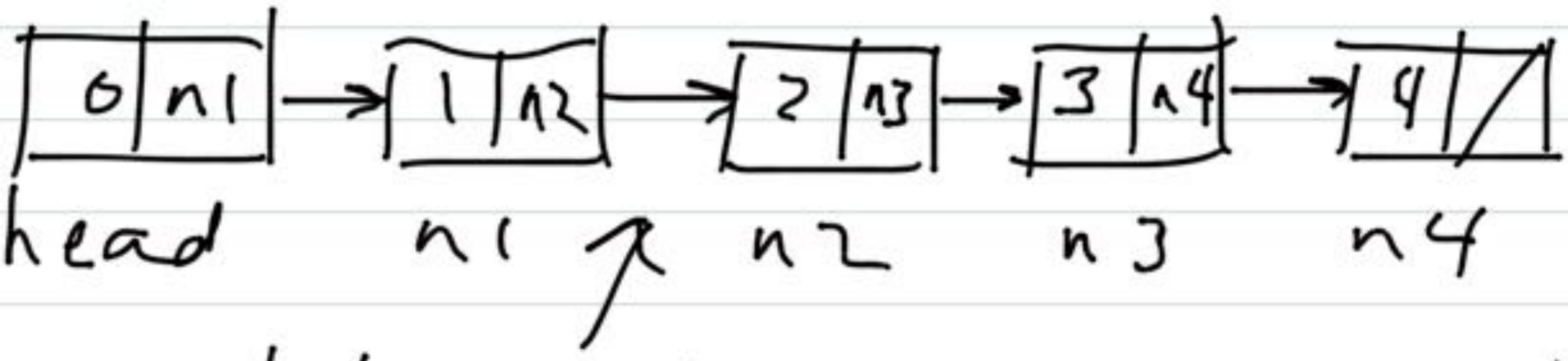
```
node *x = head
bool found = false
while (!found)
    if x→key == 2
        found = true
    else
        x = x→next
```



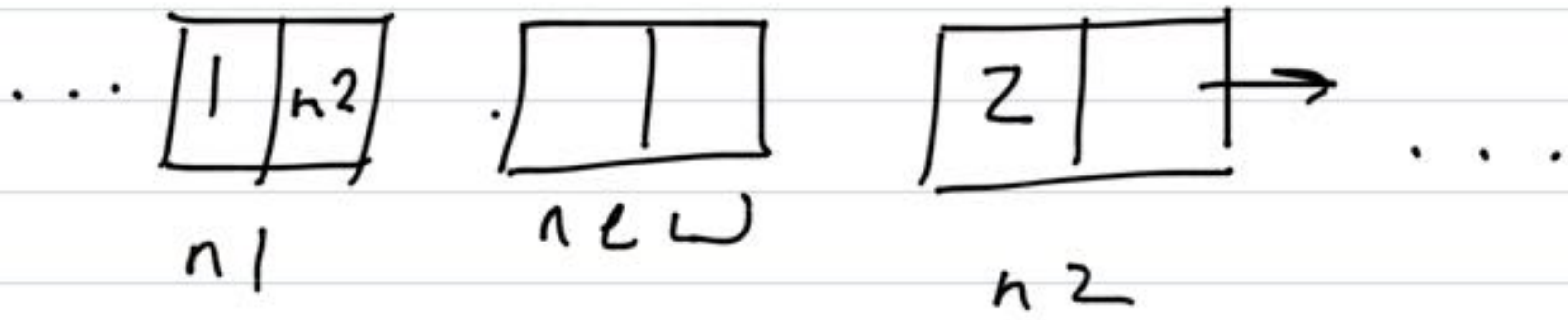


no memory alloc for x, just points to existing

Inserting a node after a node



want to put a new node here



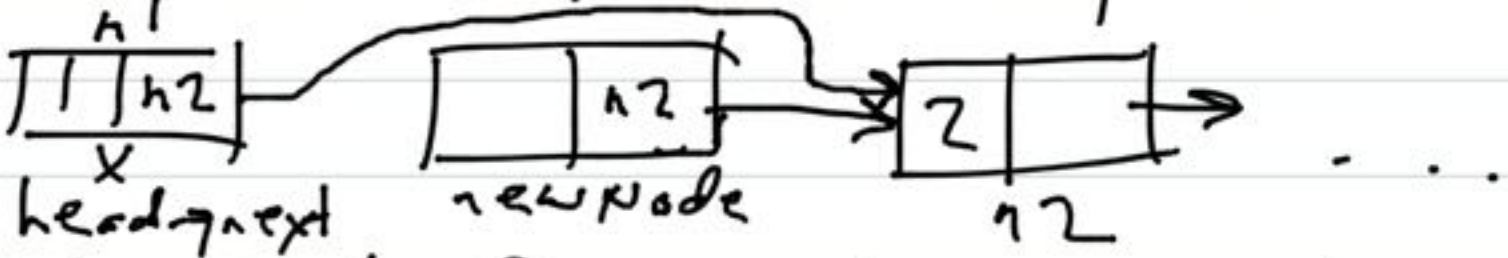
We don't actually use the address reference in code, we use x→next, or some other variable

Order of operations matters

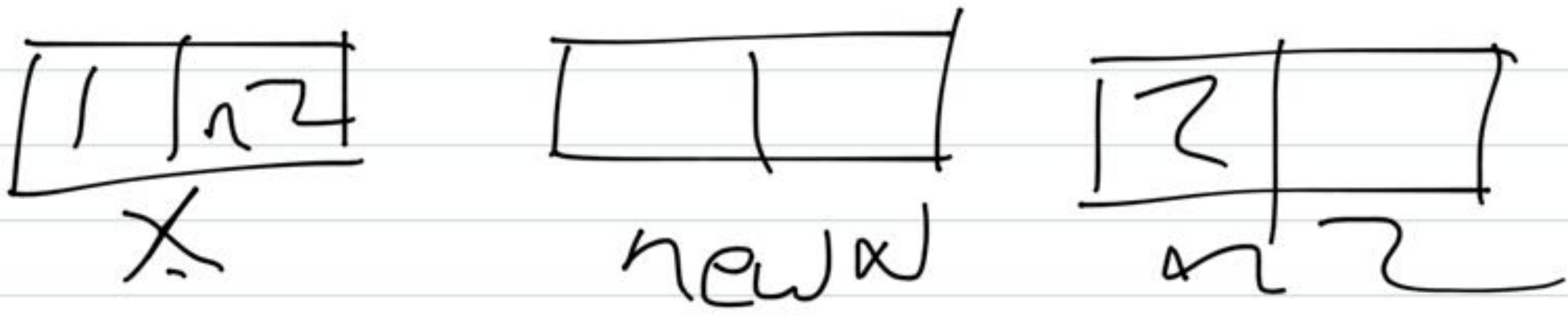
node *newNode = new Node. //alloc

node *x = head→next

newNode → next = X → next

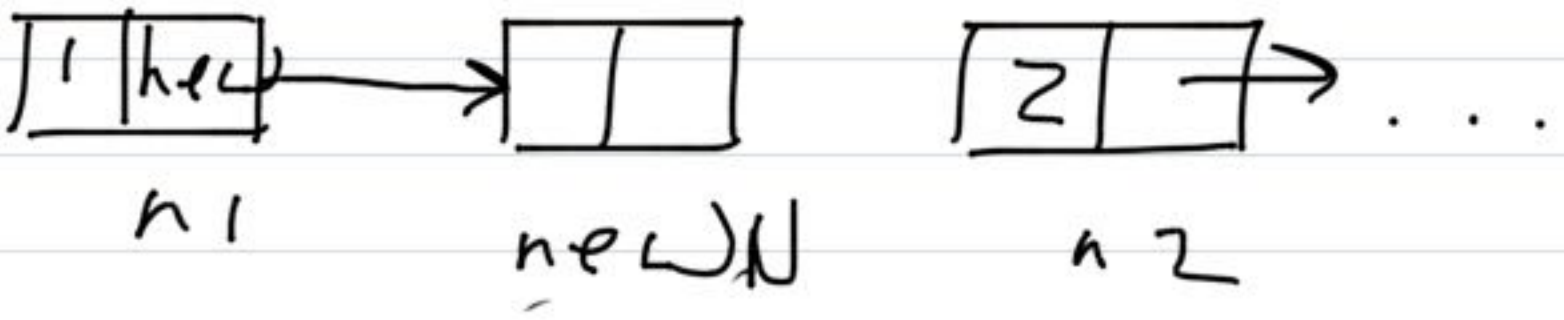


reset next for X : X→next = newNode



Need to set next for new node first, otherwise

if you do n1→next = newN

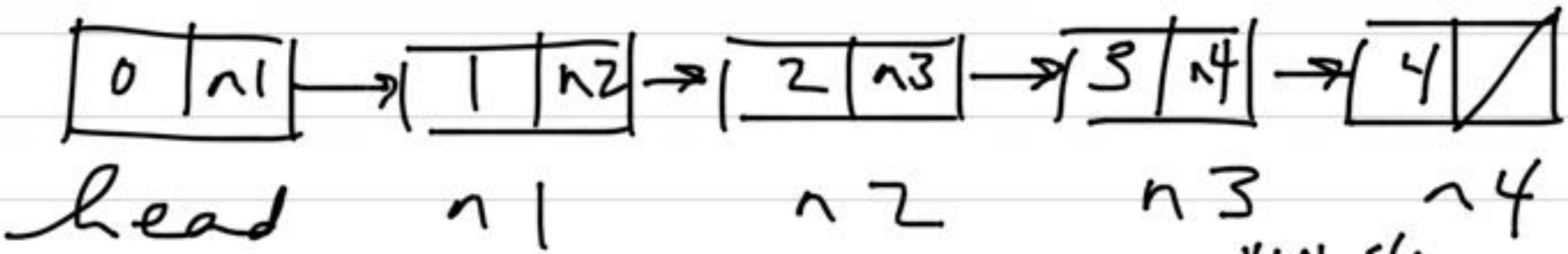


n2 was stored in n1→next, which was just overwritten

No path from newN to n2.

Deleting nodes

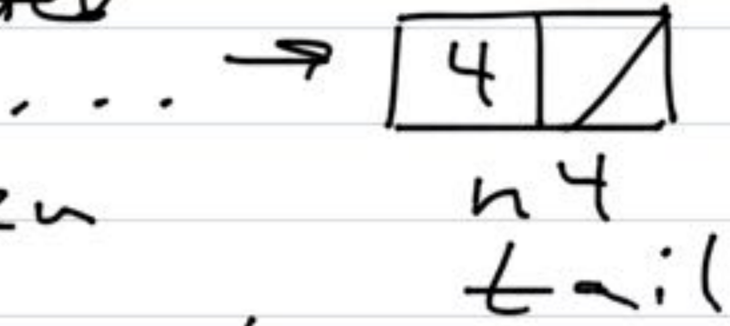
Start with this



Assume you've previously declared a tail pointer:

node *tail = n4

Assume you've iterated to where x→next = NULL, then tail = x

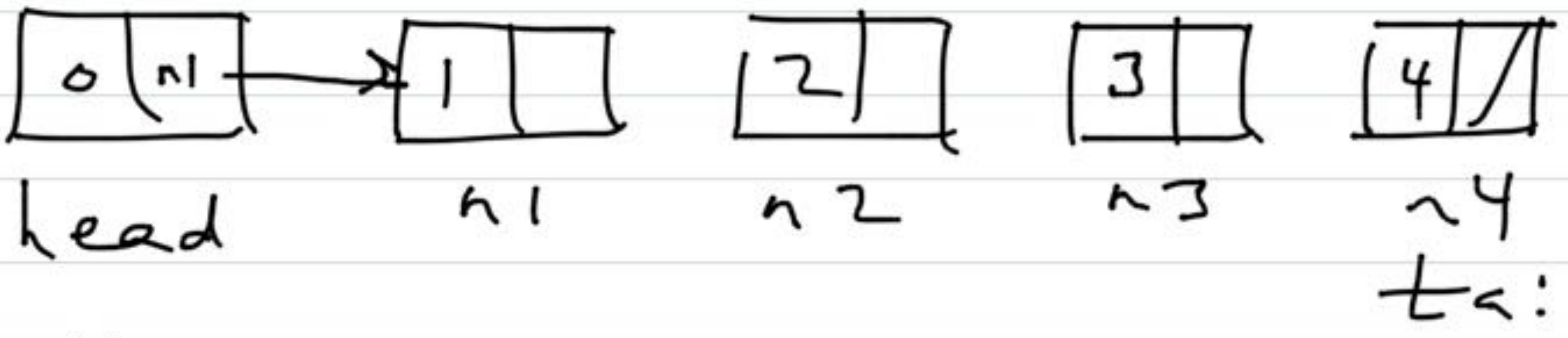


What happens if I do:

delete head ?

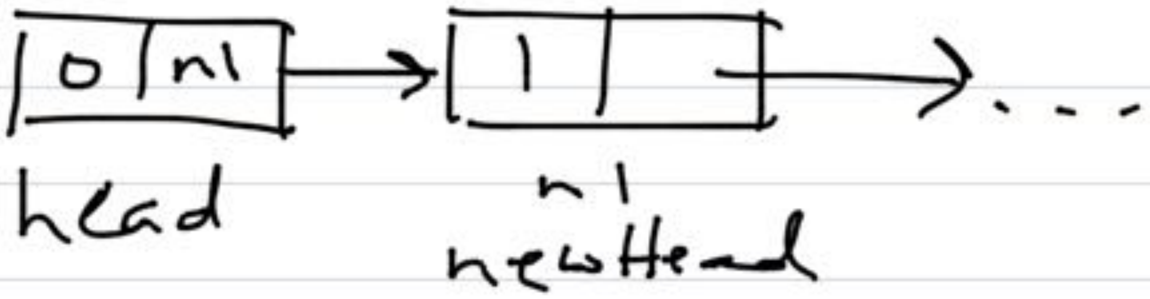
We've lost pointer to start of list, we don't actually know n1, we know head→next. And, all that memory still allocated = memory leak

Three cases for deleting
Head, Tail, Middle node

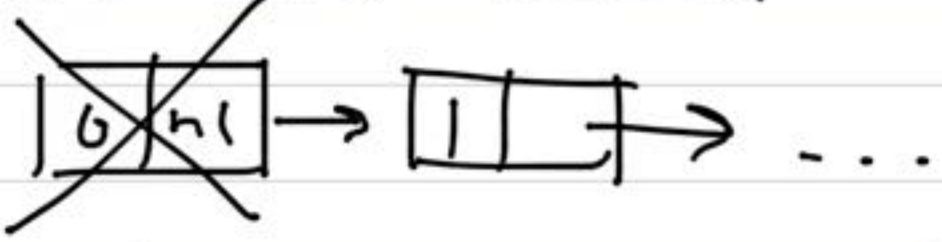


Head node

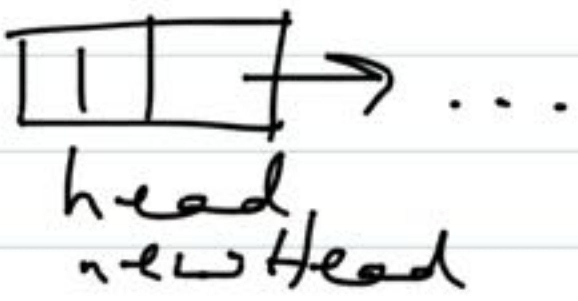
node *newHead = head→next

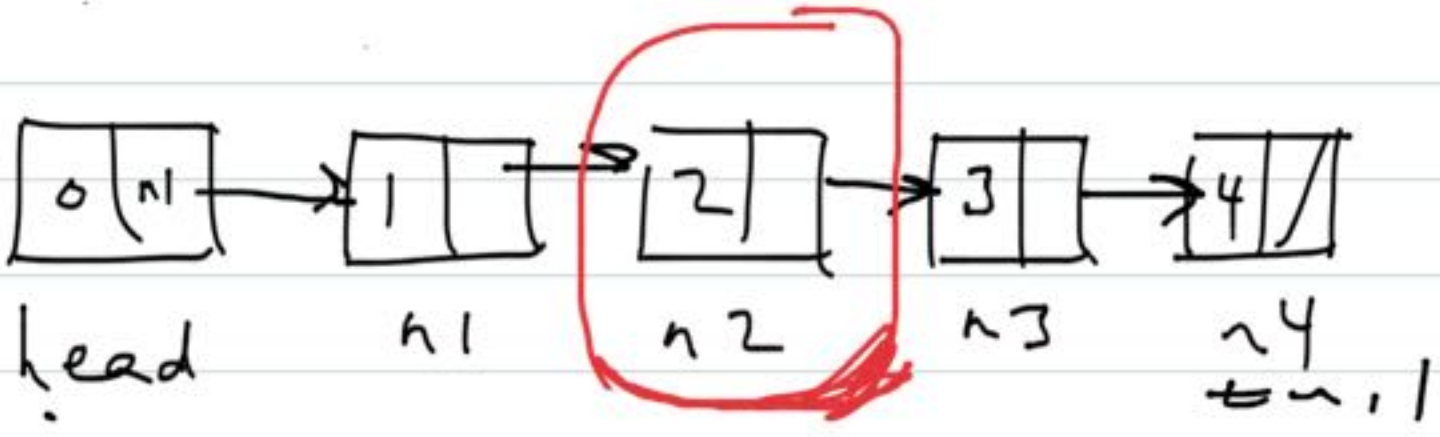


delete head : frees memory,
still have
head ptr



head = newHead





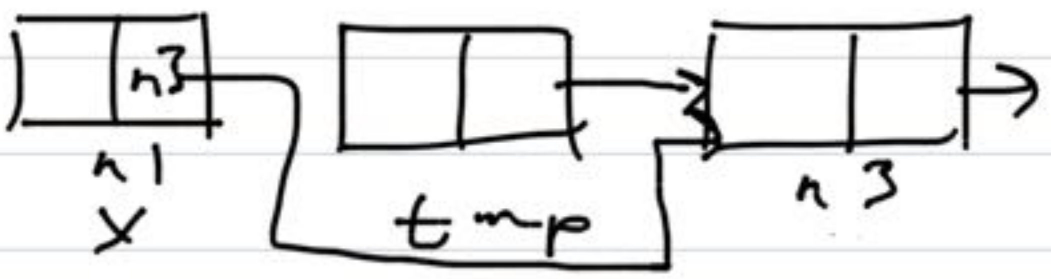
Node between 2 nodes
ex: n2

Need to preserve n2→next

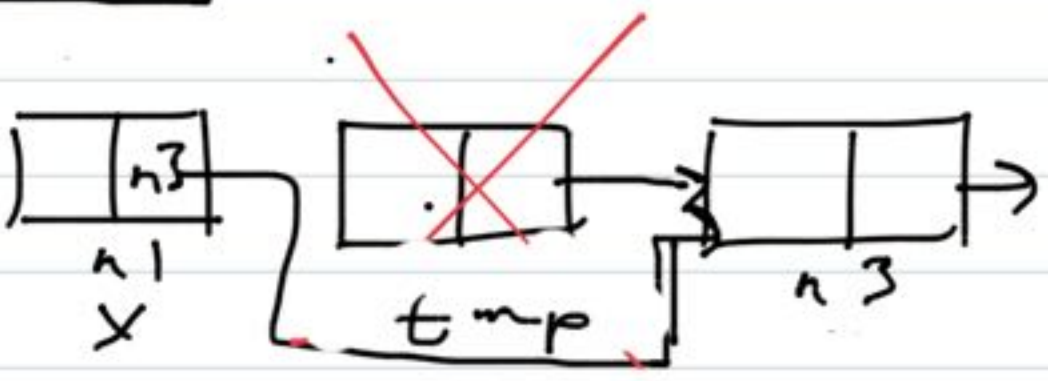
Deletion algorithm
for singly linked list requir
previous node to one being
deleted n2 has no path
to n1, can't go n1 to n3
with only knowledge of n2

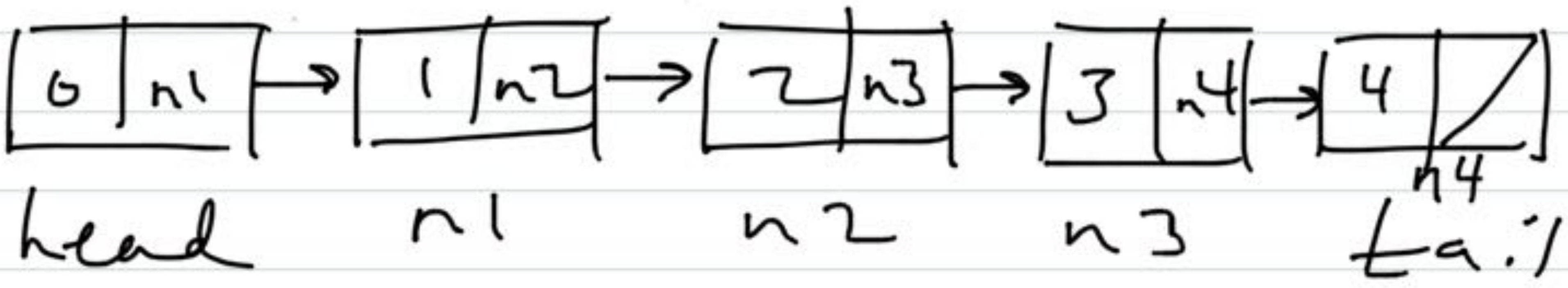
there's
no → in
that
direct

Assume we have x pointing to n1,
want to delete x→next
Store n2
node *tmp = x→next
Update next pointer
x→next = tmp→next



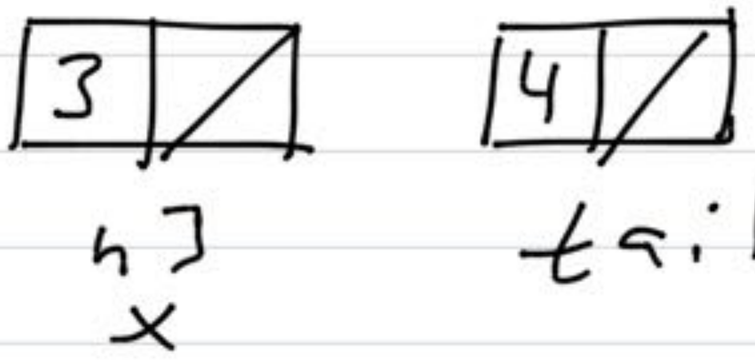
delete tmp



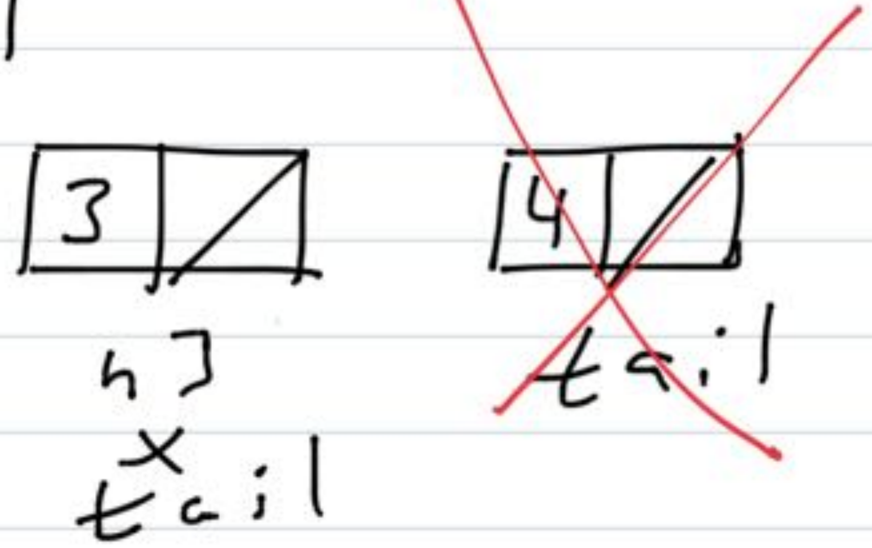


Delete tail

Assume we know previous node, eg. x→next = tail
or x→next→next = NULL
Assume
We have a pointer to n3 called x
Since we have a stored pointer to tail, we can say
x→next = NULL to get



delete tail
tail = x


3
4
n3
x
tail
tail