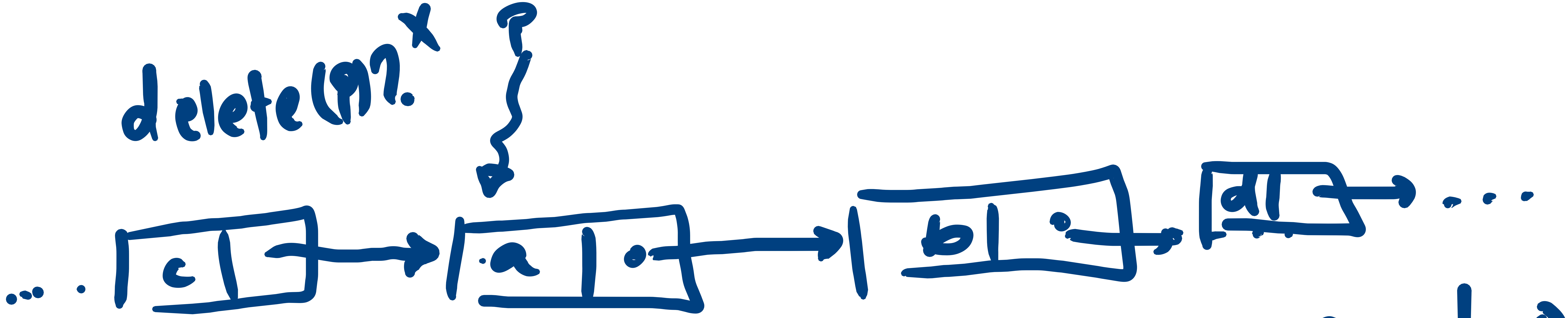
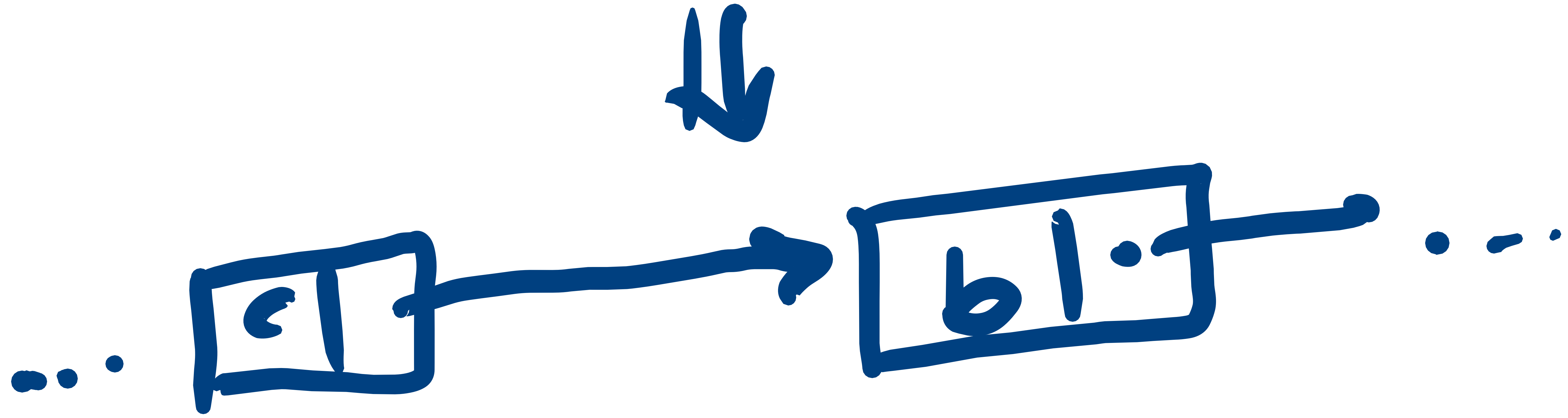
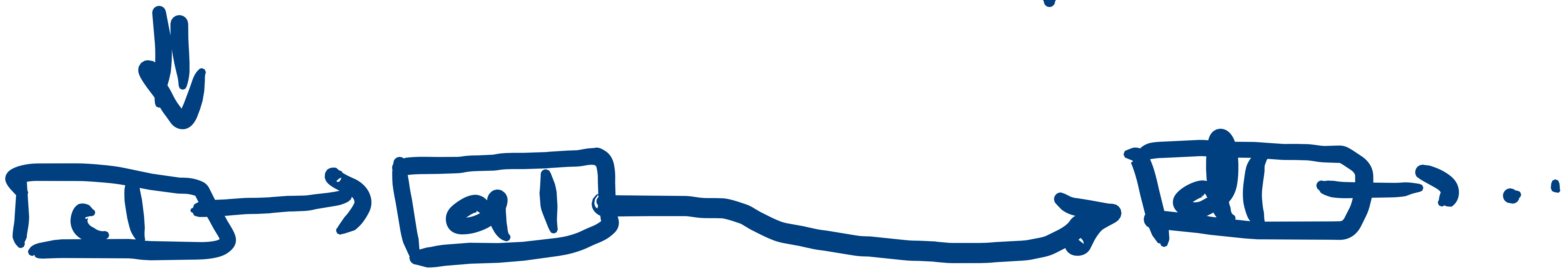
delete(p)? ✗



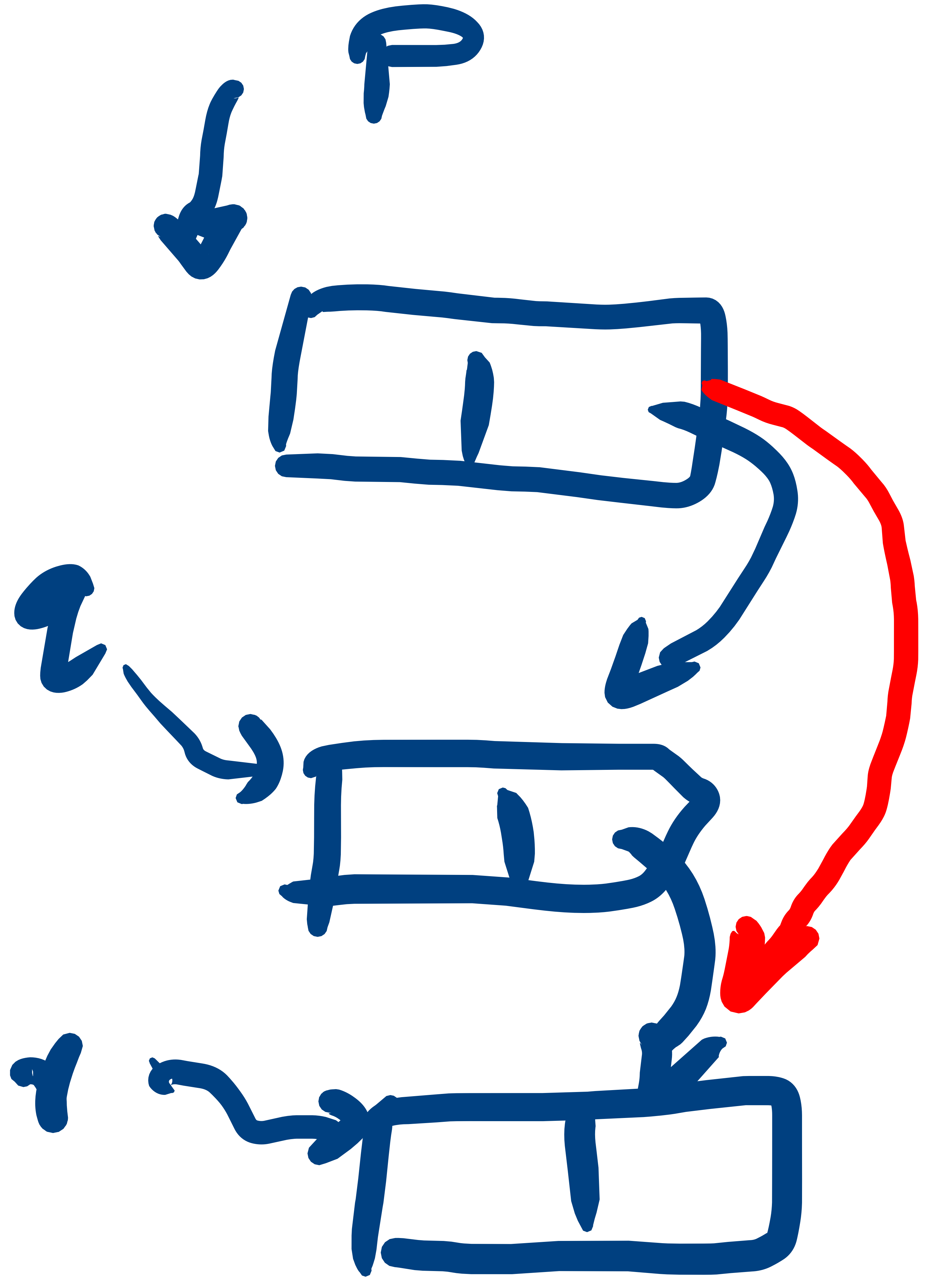


Not a good option.
Need access to c.

delete_next(p) ← delete the node following p from the list



```
delete_next(P)
    q = P.next
    r = q.next
    P.next = r
    delete q
    deallocate
      memory for q.
```



O(1)-time

What happens if P is last elt?

Two Solutions

1. Require P is not last elt.

2. Check and do nothing if P is last.

1 is faster

2 is safer

Stacks & Queues using linked list

Stack

$$\left.\begin{array}{l}\text{Push}\\ \text{Pop}\\ \text{is\_empty}\\ \text{empty\_stack()}\end{array}\right\} O(1)\text{-time}$$

How is empty stack represented ?. NULL

S= empty()

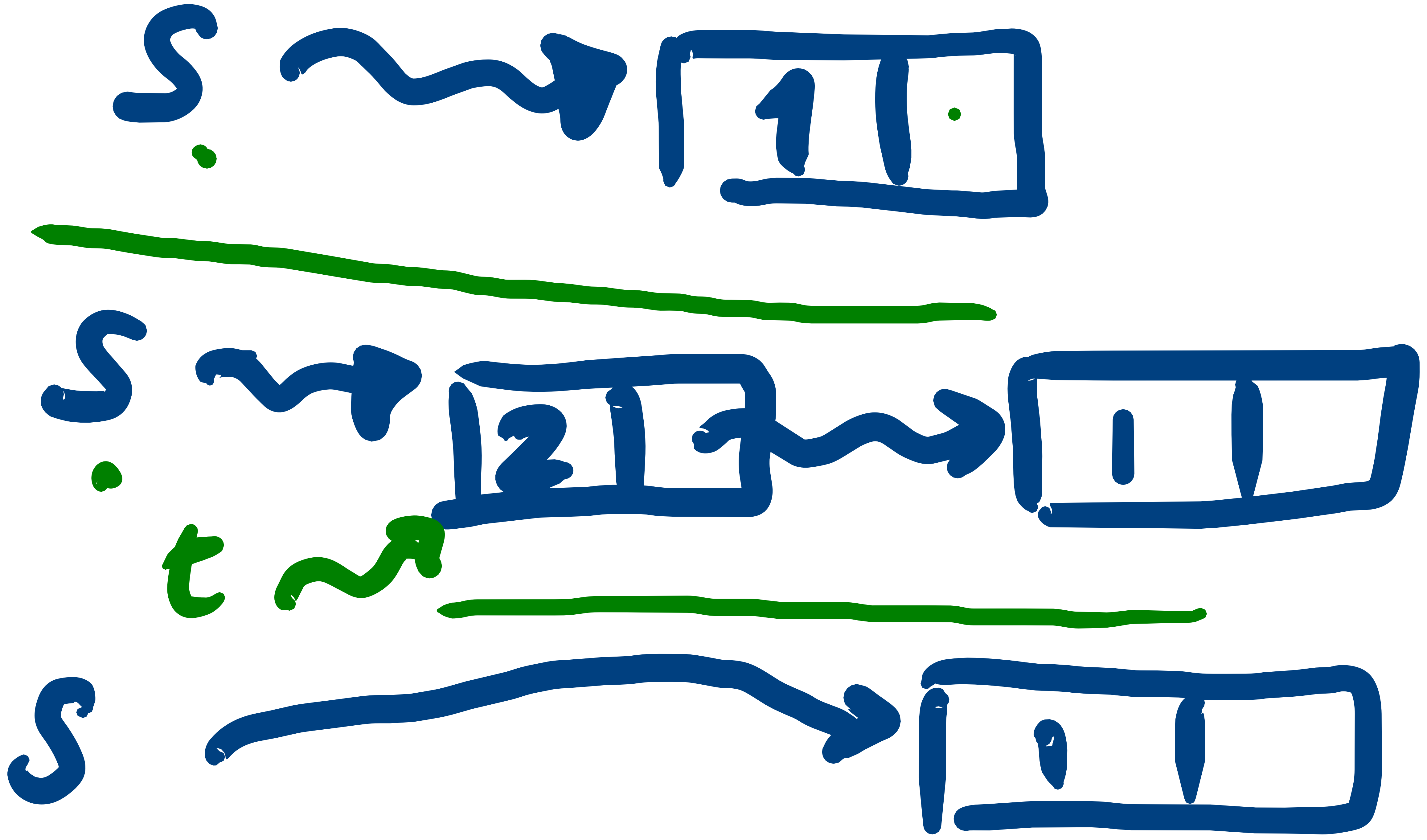
Push(S,1)

Push(S,2)

Pop()

Pop()

S = NULL



S = NULL

```
Push(s, v)
    t = new_node(v)
    t.next = s
return t {  s = t
            return s
```

```
Pop(s) // may be deallocate
    return s.next

empty()
    return NULL

is_empty(s)
    return s==NULL
```

S_1 = copy-stack(S_2)

```
s= empty()
Push(s,1)
t= copy(s)
Push(t,2)
Push(s,3)
Print(s) // s= ( 3   1)
Print(t) // t = ( 2   1)
```

```
Pop(t)
Print(s) //(3 1)
Pop(t)
Print(s) //( 3 1)
```

```
a= 2
b=a
a += 1
b += 2
Print(a)  //  3
Print(b)  //  4
```

Be like water

Cppcon talk
"Be like int"

copy_stack()

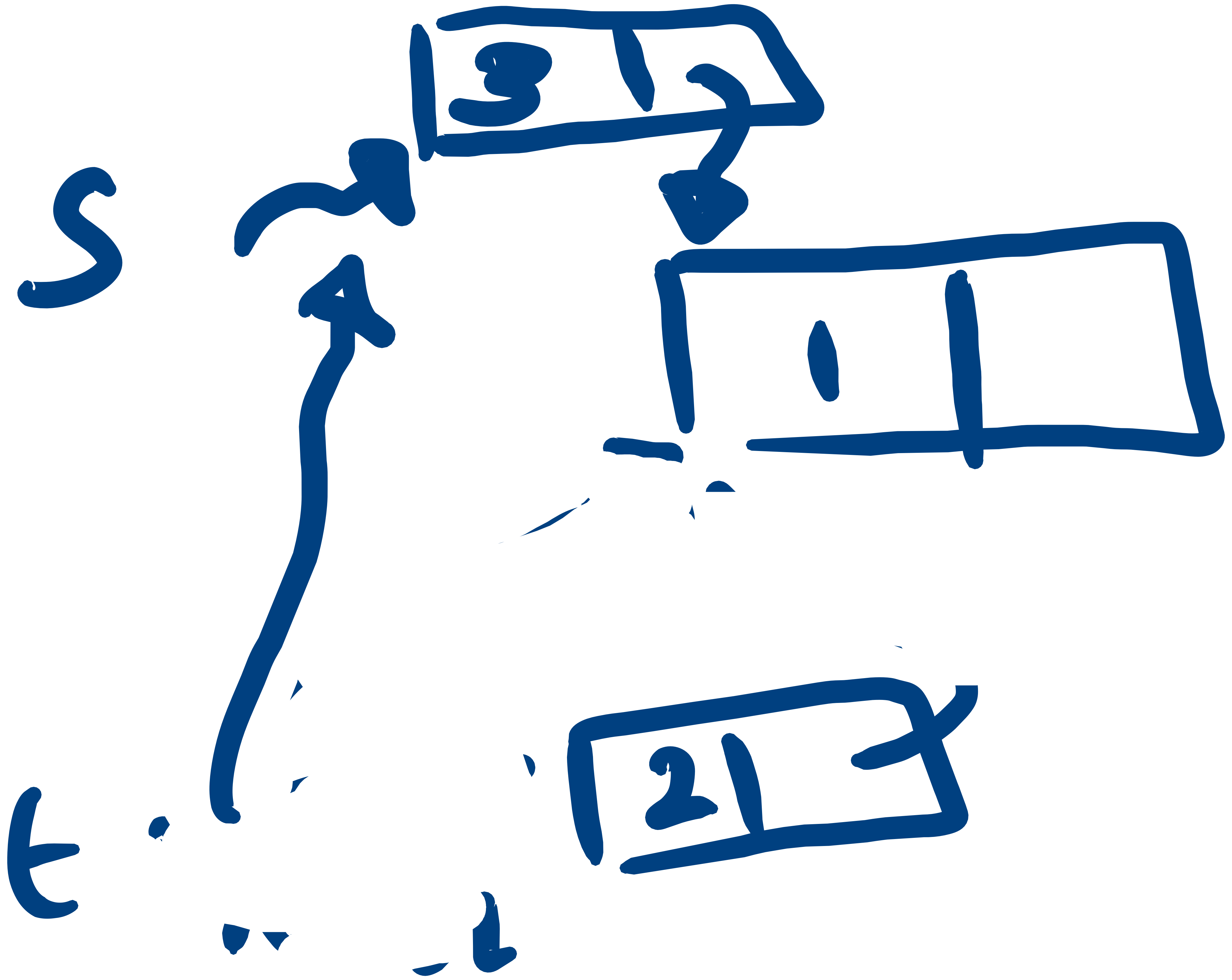
array : $\Omega(n)$

allocate a new array

copy all elts from old array

to new array

linked list?



Push(t, 2)

Push(s, 3)

Pop(t)

Pop(t)

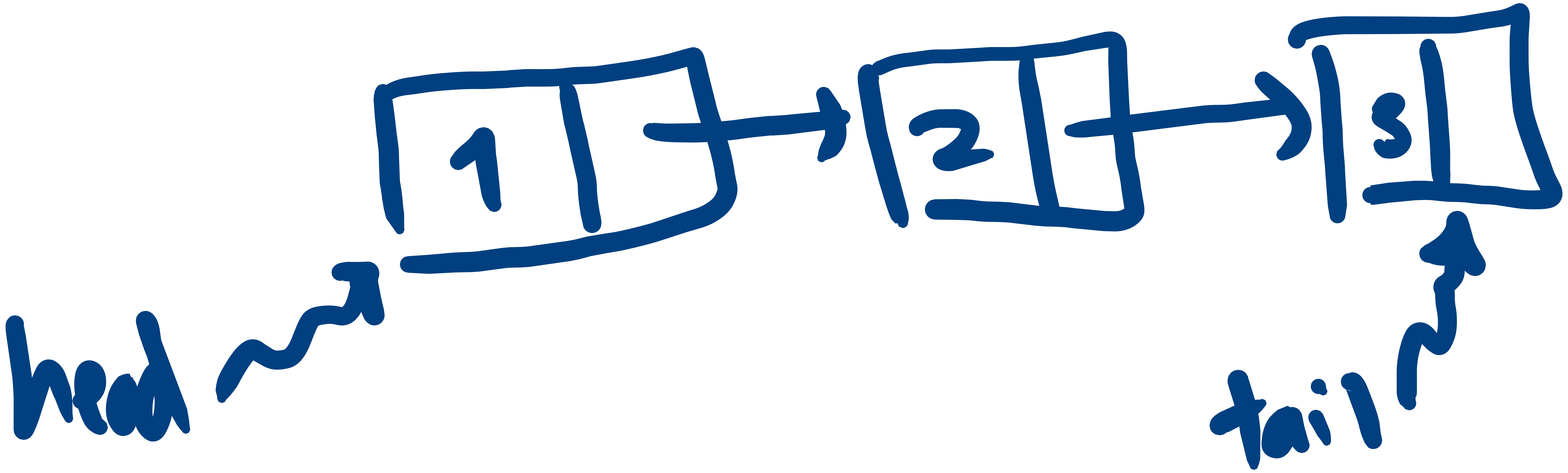
t = copy(s)

"tail-sharing"

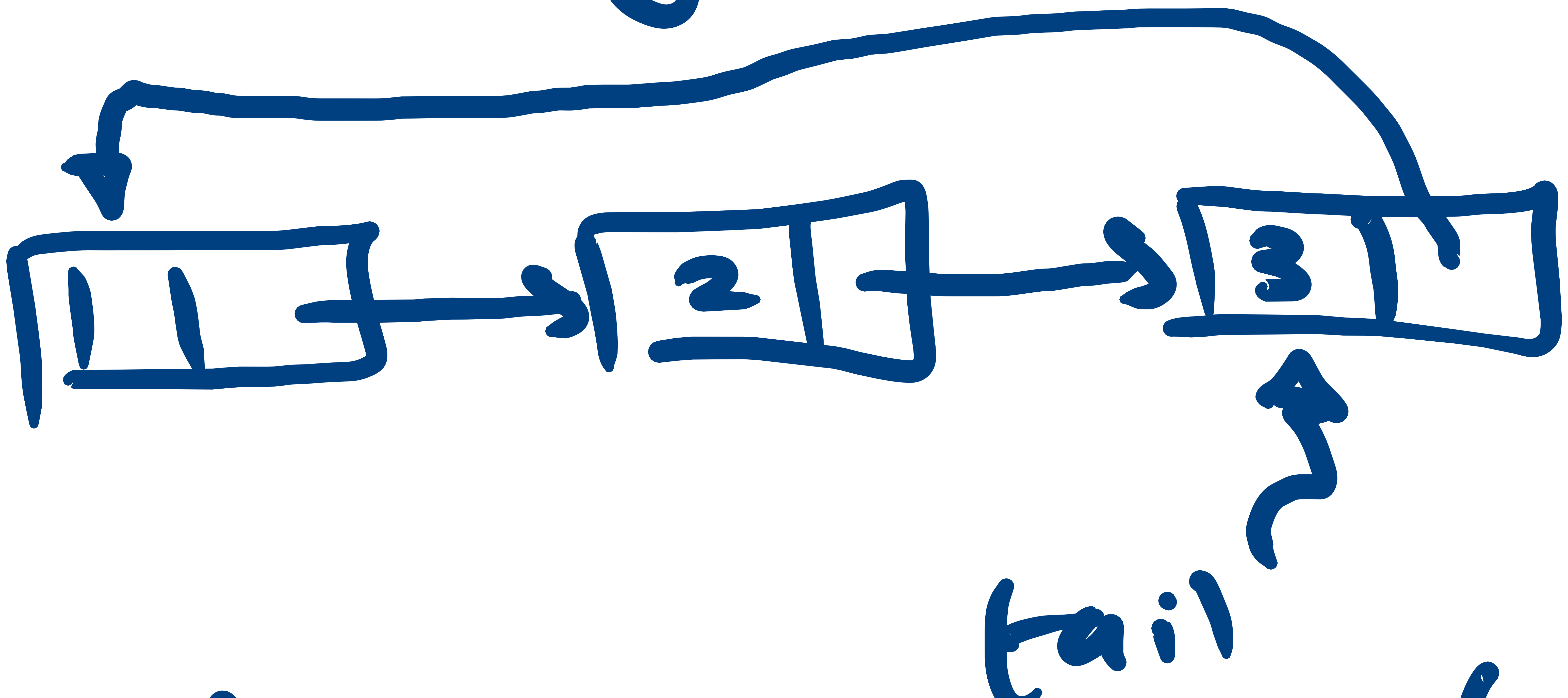
enables copy_stack() in O(1)-time.

Queue

enqueue() } O(1)-time
dequeue()



Circular Singly linked list



Copy_queue() ?

Corner cases