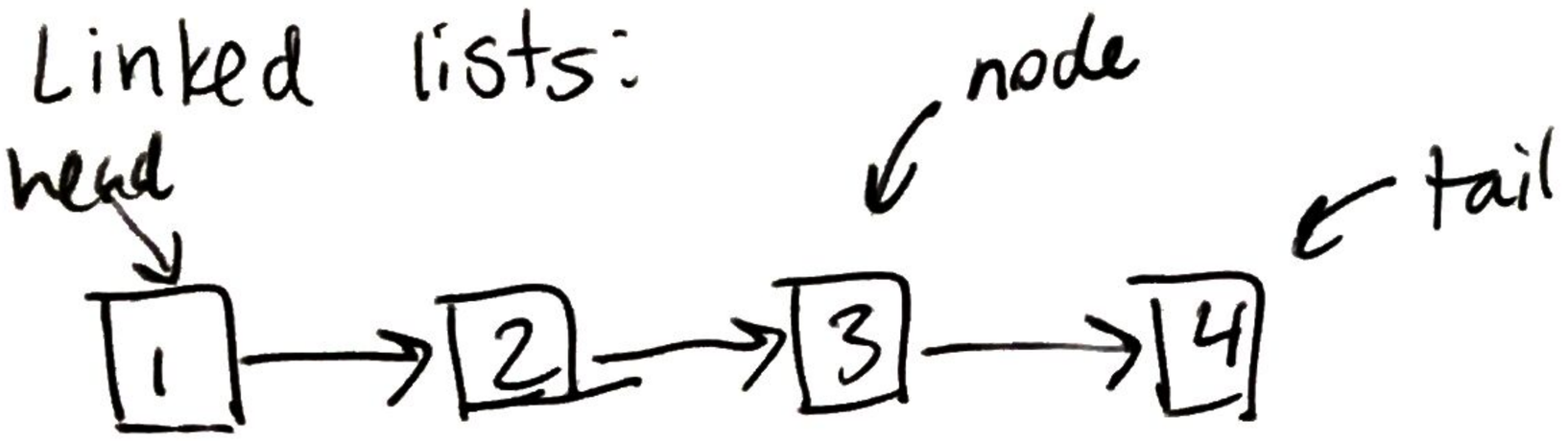
Linked lists:



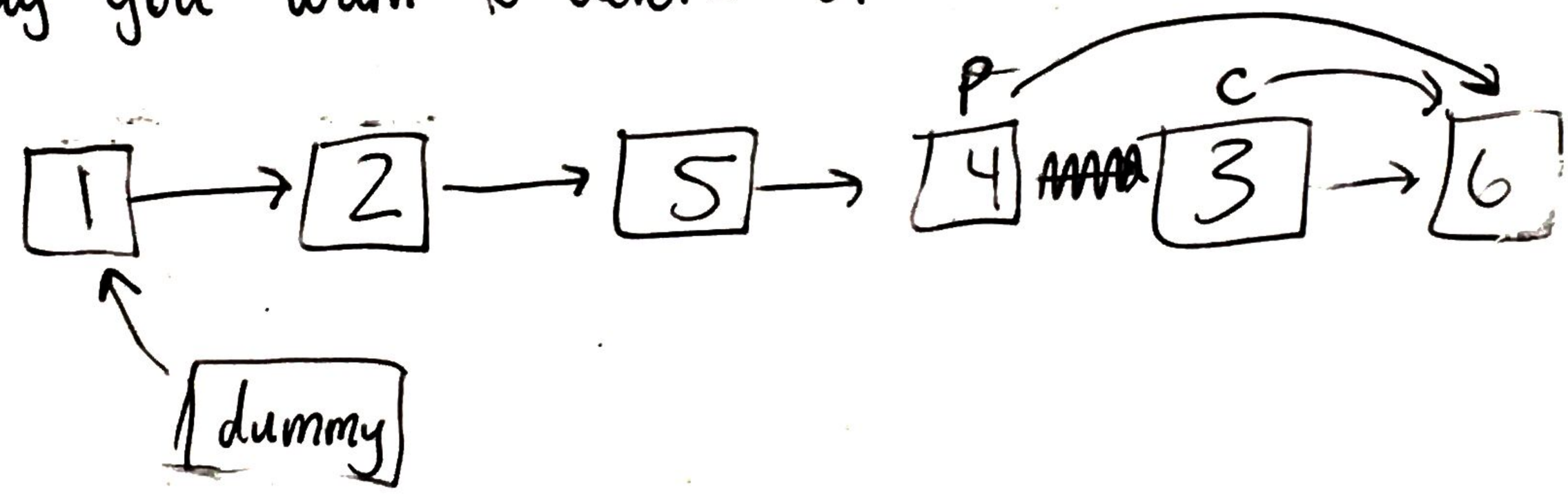
Best case:
- access: O(1)
- insertion: O(1)
- deletion: O(1)

Worst case:
- access: O(N)
- insertion: O(N)
- deletion: O(N)

saves us from dealing with edge cases w.r.t. head operations

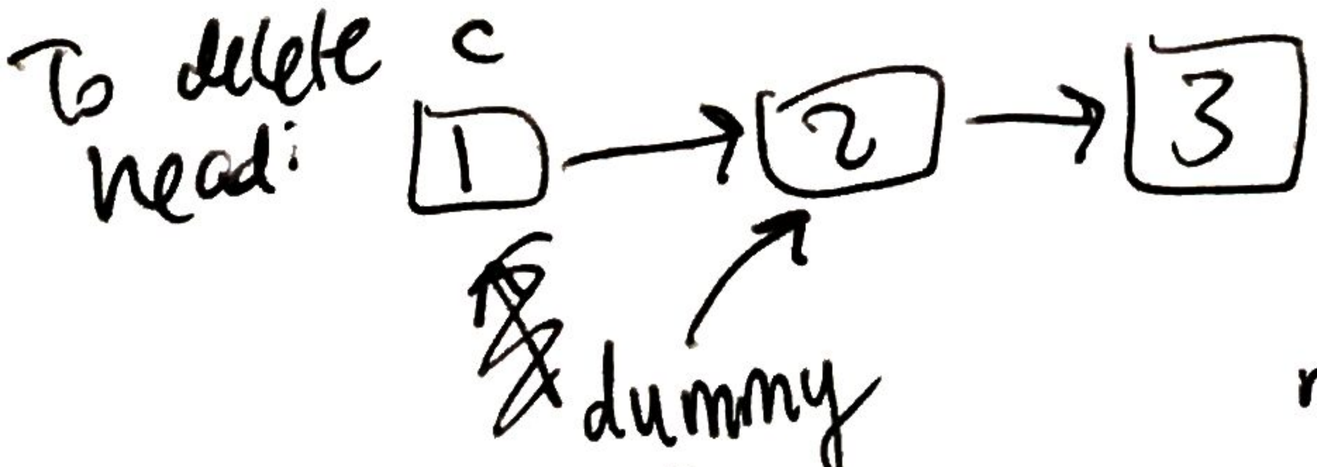
Dummy Head technique: create an extra pointer that will point to the final answer or list you will return

Say you want to delete 3.



return head of list: 1

To delete head:



return head of list: 2

Multiple Pass Technique

e.g. write a program to find the node at which the intersection of 2 singly linked lists begins

- if ∄ intersection, return null
- linked lists must keep original structure at end
- assume no cycles
- code should run in $O(N)$ time w/ $O(1)$ space

2 observations:

- Once a list has intersected w/ another, rest of list is identical.
- A list can only be identical if it has same length.
  - ↳ To make 2 linked lists the same length:
    - get the length of both
    - assign head of longer LL to "long" & head of shorter LL to "short"
    - record longer_length & shorter_len
    - while longer_length > shorter_len:
      - long = long.next
      - longer_length --

2 ways to calculate the length of a linked list:

- recursively: get_length(LL)
  - if LL is null: return 0
  - return get_length(LL.next) + 1
- iteratively: len = 0, len++ every time you move to LL.next