# liste

# ARRAY

DIMENSIONE MASSIMA PREFISSATA

POSSIAMO ACCEDERE A OGNI ELEMENTO IN TEMPO COSTANTE $O(1)$

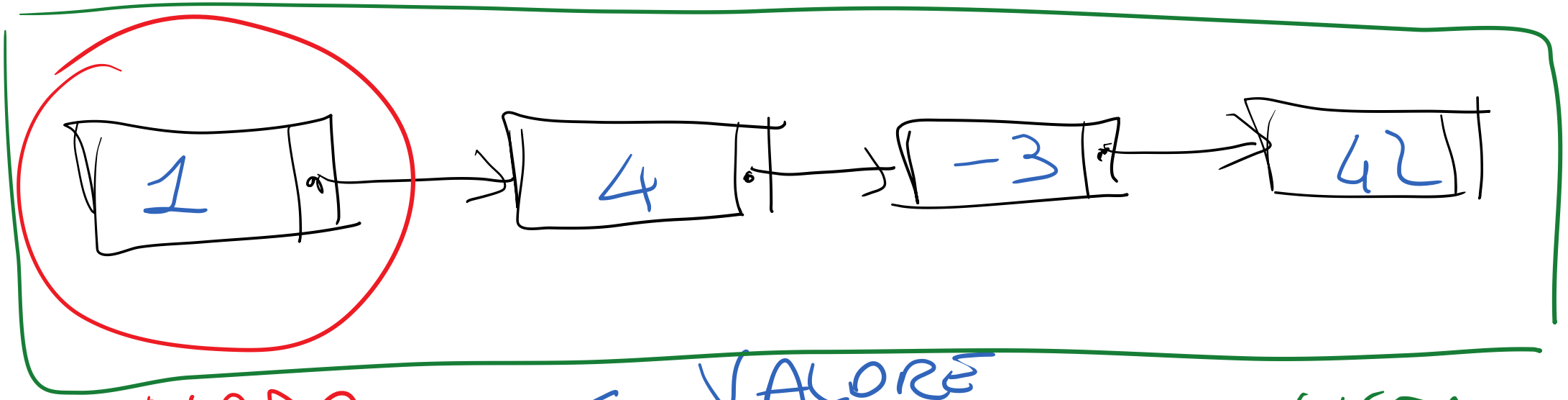
# LISTE

- SEQUENZA DI ELEMENTI
- COLLEGAMENTO TRA ELEMENTI

+ DIMENSIONE NON È PREFISSATA

# LISTE

LINKED LIST



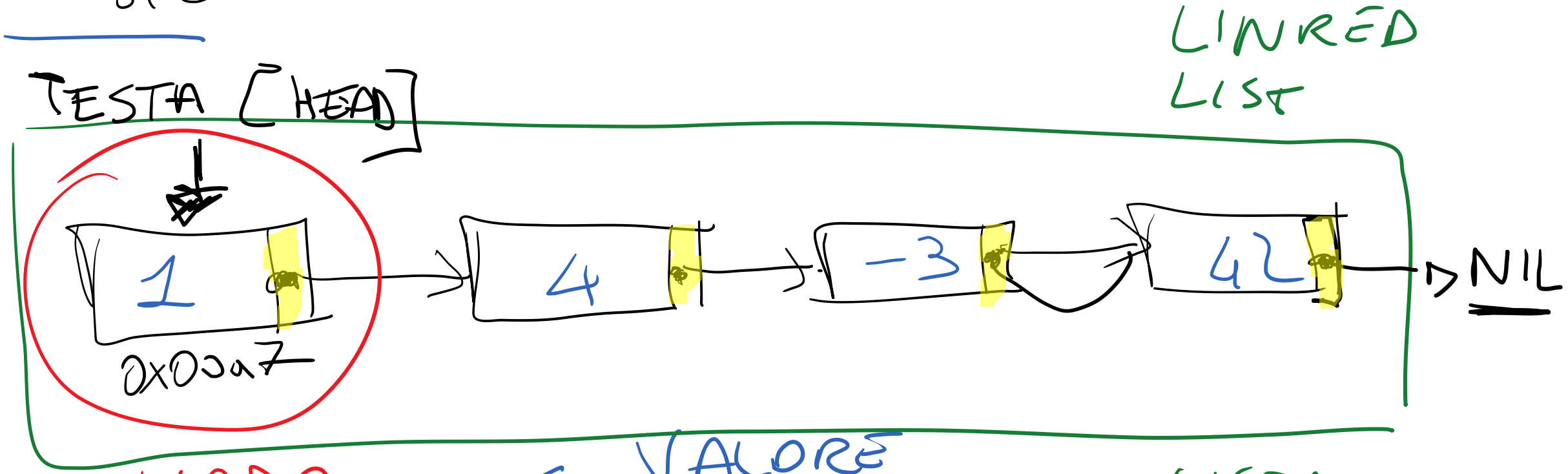

NODO
[NODE]

VALORE

PUNTATORE (LINK)

LISTA LINKATA

# LISTE


TESTA [HEAD]
1
0x0047
4
-3
42
NIL
LINKED LIST
NODO [NODE]
VALORE
PUNTATORE (LINK)
LISTA LINKATA

# LISTE LINEARI SEMPLICI

1. SEQUENZA DI ELEMENTI
2. ~~ELEMENTI COLLEGATI TRA LORO~~
   OGNI ELEMENTO È COLLEGATO AL SUCCESSIVO
3. DIMENSIONE NON NOTA A PRIORI
4. ACCESSIBILE TRAMITE UN PUNTATORE AL PRIMO ELEMENTO
5. TERMINATA DA UN PUNTATORE A NIL

# LISTE LINKATE SEMPLICI

- HEAD : Nodo *
- Tipo di dato

## NODO

- VALORE : Tipo di dato
- SUCCESSIVO : NODO*

- Scrivere una classe template che implementi la lista
- Scrivere una classe template che implementi il nodo

List

- head: Node<T>*

Node

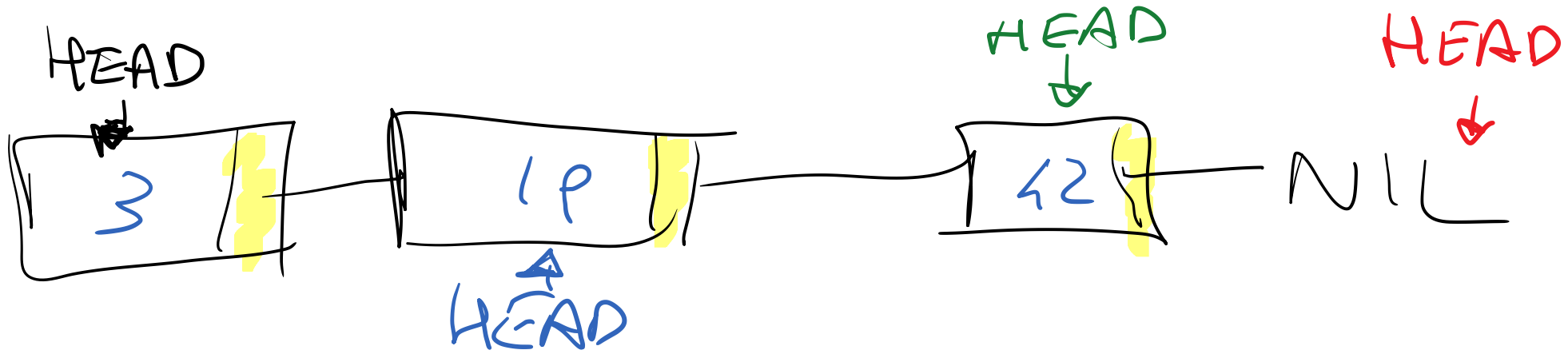
-val : T

- Next : Node<T>*

# OPERAZIONI SULLE LISTE

1. INSERIMENTO 2.
2. ACCESSO
3. RICERCA 4.
4. CANCELLAZIONE 3.
5. ORDINAMENTO
6. COPIA
7. CONTROLLO LISTA VUOTA 1.

# CONTROLLO LISTA VUOTA



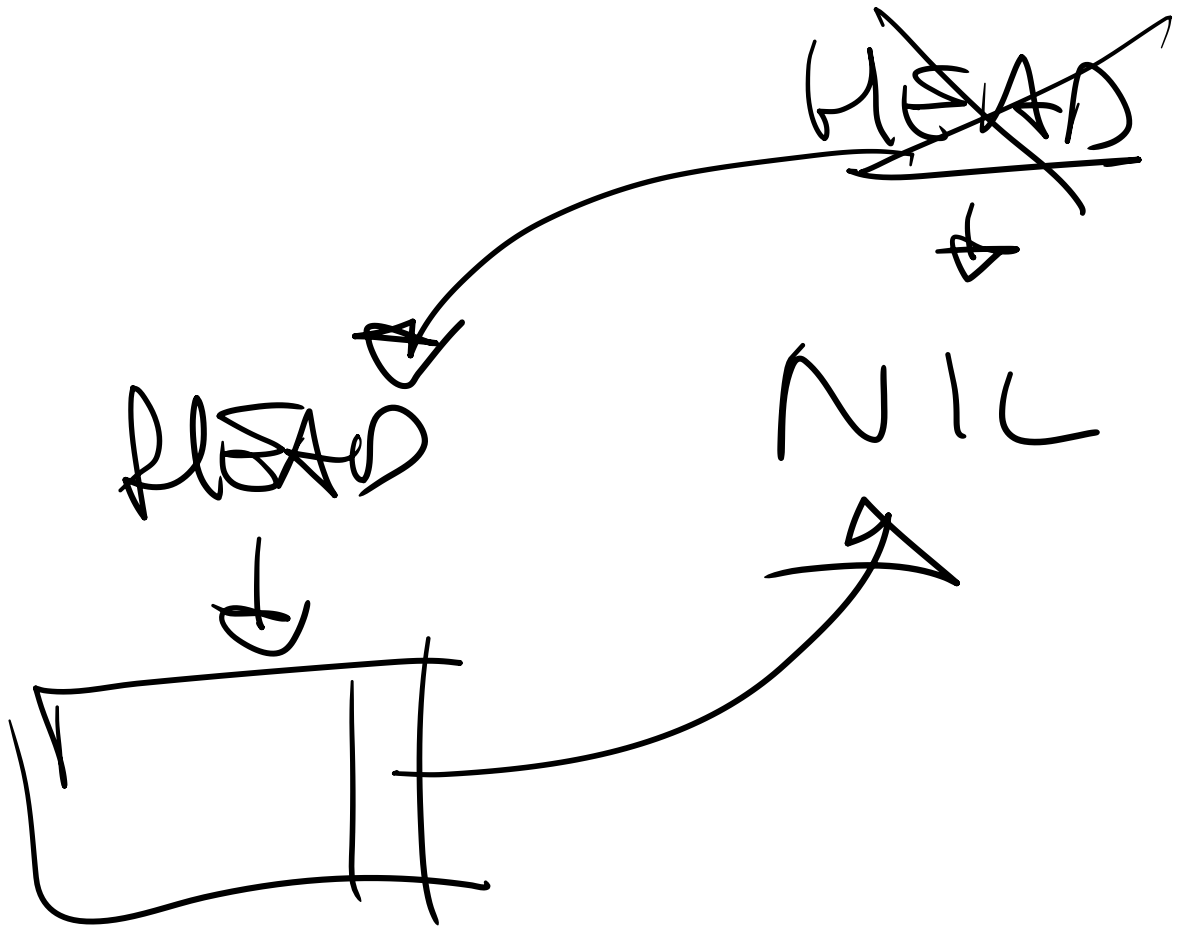

LISTA NON VUOTA

LISTA NON VUOTA

LISTA NON VUOTA

LISTA VUOTA

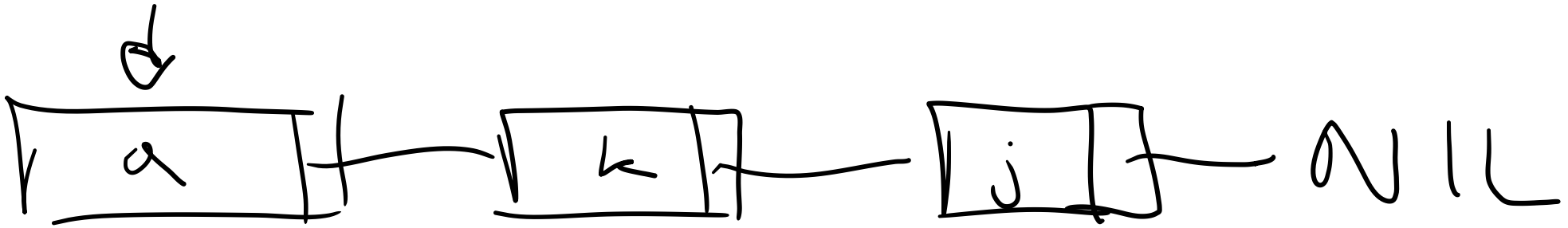
# INSERIMENTO



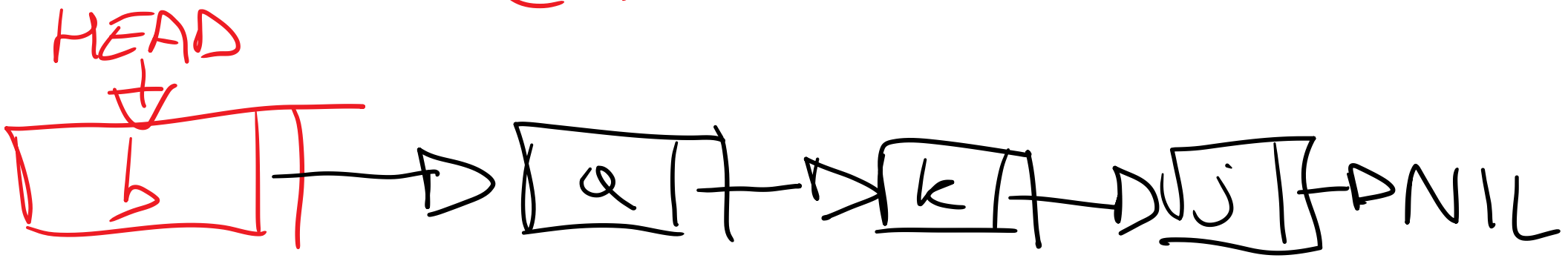

IN LISTA VUOTA

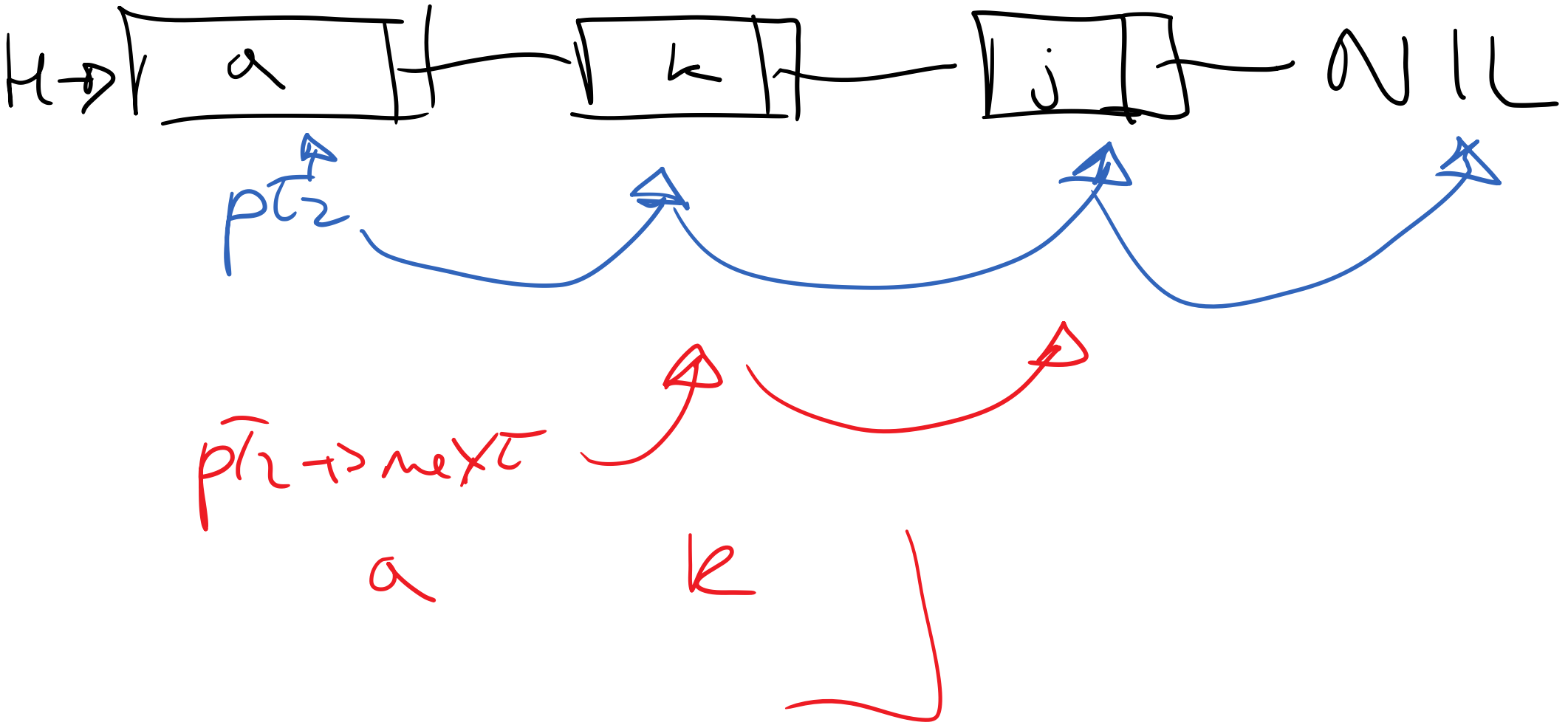
# INSERIMENTO

HEAD

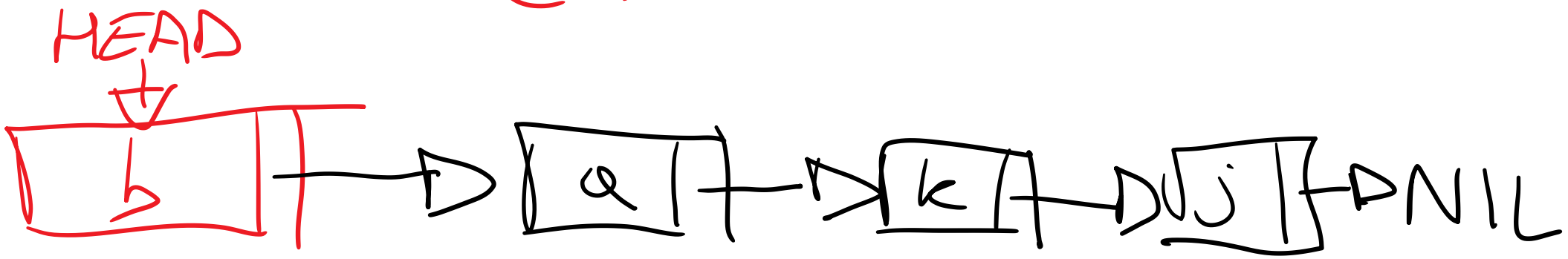



IN TESTA (b)



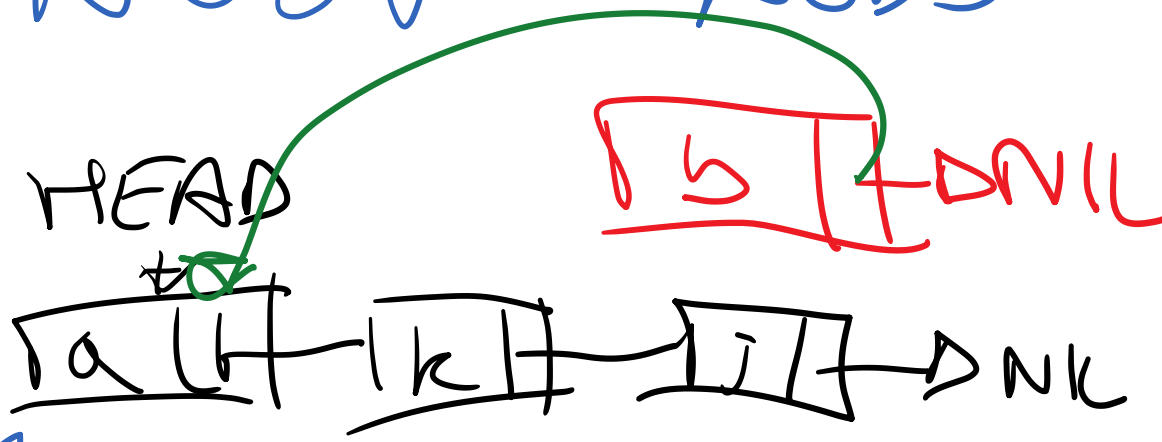

H
a
k
j
NIL
ptr
ptr->next
a
k

# IN TESTA (b)




1. CREARE IL NUOVO NODO

2. IL SUCCESSIVO DEL NUOVO NODO DEVE ESSERE LA TESTA DELLA VECCHIA LISTA

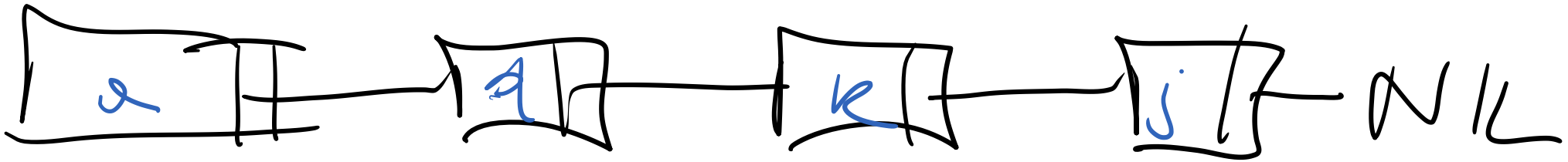


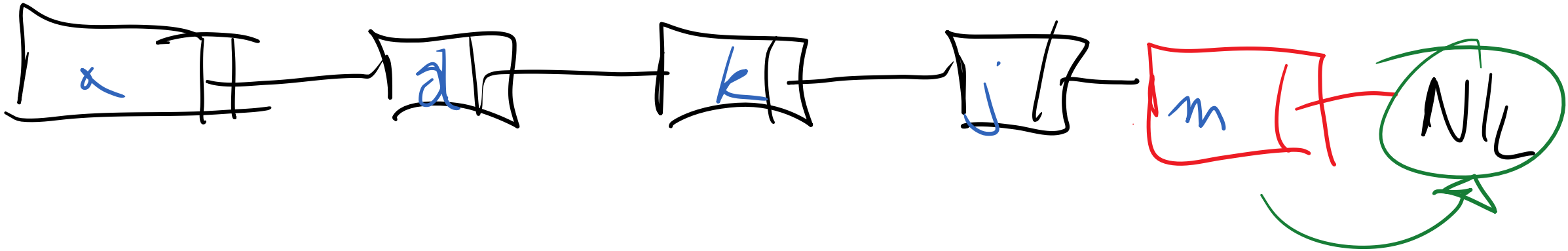

3. AGGIORNARE LA TESTA

# INSERIMENTO IN TESTA

1. temp ← Nodo(val)
2. temp.next ← head
3. head ← temp

# INSERIMENTO IN CODA



# INSERIMENTO IN CODA

1. SE LA LISTA È VUOTA, INSERISCO IN TESTA
2. SE LA LISTA NON È VUOTA
   a. SCORRERE LA LISTA FINO ALL'ULTIMO ELEMENTO
   b. INSERIRE L'ULTIMO ELEMENTO
      CREARE UN NUOVO NODO
      FAR PUNTARE IL VECCHIO NEXT AL NUOVO NODO

OUTLINE 11/05/2022

CREARE UNA LISTA

VERIFICARE SE È VUOTA

INSERIRE IN TESTA

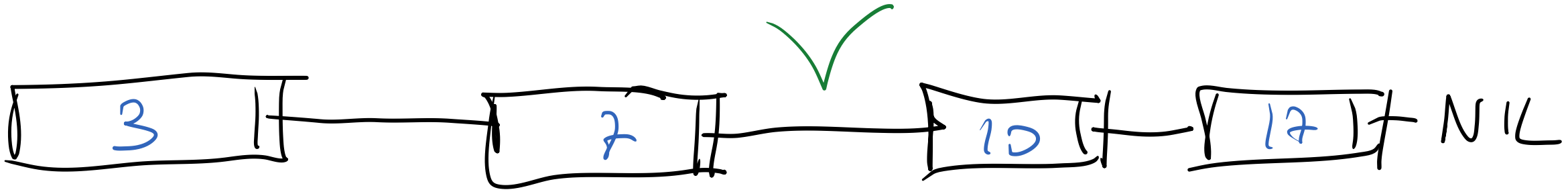
INSERIRE IN CODA

INSERIRE IN POSIZIONI INTERMEDIE

(INSER. ORDINATO)

CANCELLAZIONE

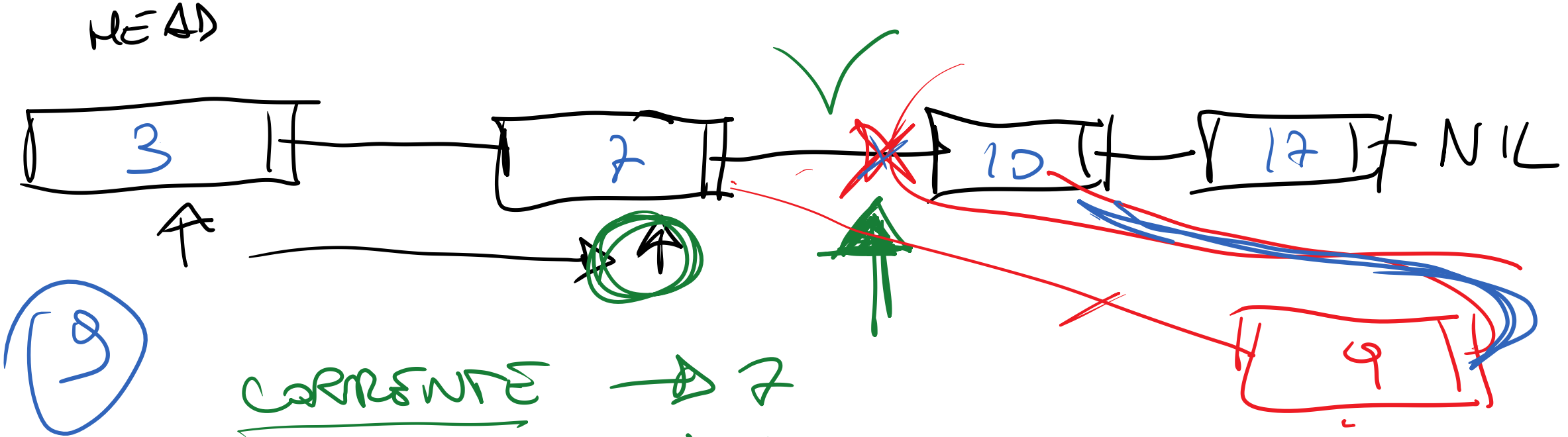
# INSERIMENTO ORDINATO




9

- SCORRERE LA LISTA PER TROVARE LA POSIZIONE CORRETTA
- "STACCARE" I COLLEGAMENTI
- SOSTITUIRE I COLLEGAMENTI CON IL NUOVO NODO

# INSERIMENTO ORDINATO




9

CORRENTE → 7

NEXT → 10

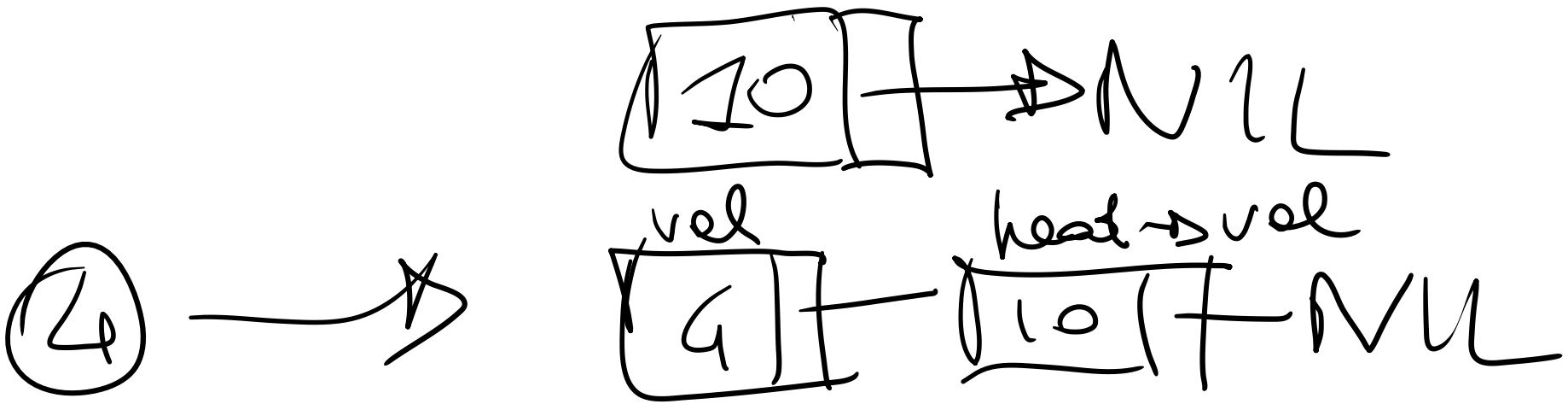
NUOVO → 9

1

- 9. NEXT =
- 7. NEXT
- 7. NEXT = 9

ASS. La lista è ordinata

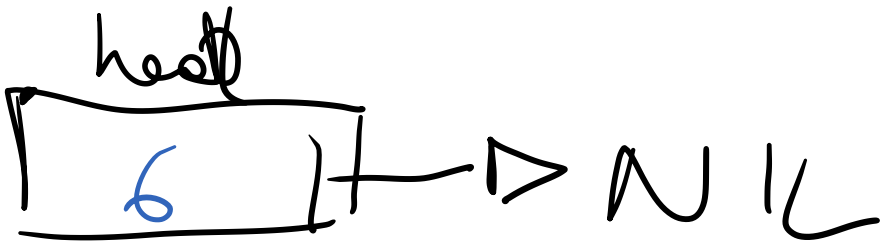
DEF. Una lista vuota è ordinata



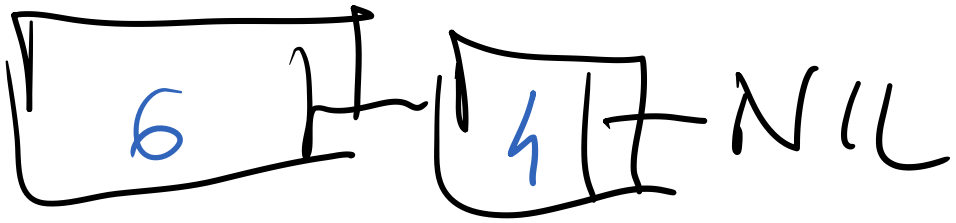
# esercizio

- Modificare il codice dell'inserimento ordinato in modo che sia possibile inserire sia in ordine ascendente (quello che abbiamo fatto adesso) che in ordine discendente
- Suggerimento: scrivere due nuovi metodi e selezionare l'inserimento appropriato attraverso un parametro

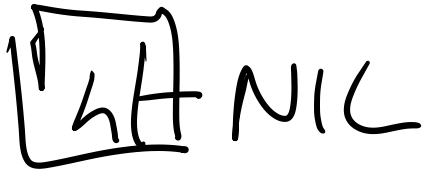
# CANCELLAZIONE

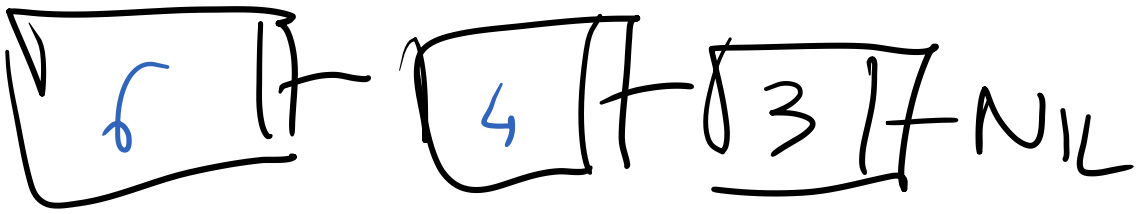



DELETE (6)

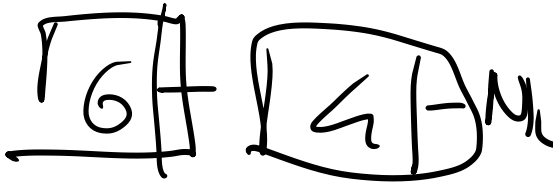



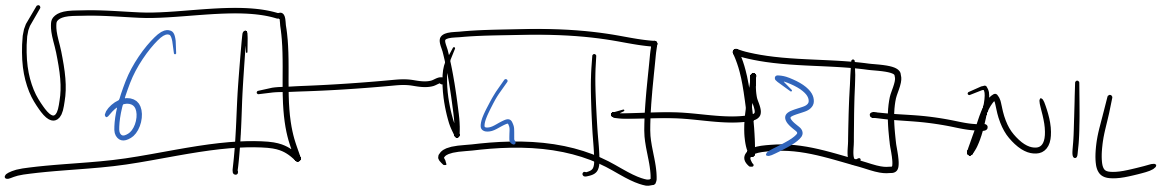

DELETE HEAD



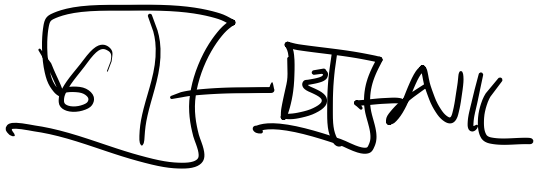



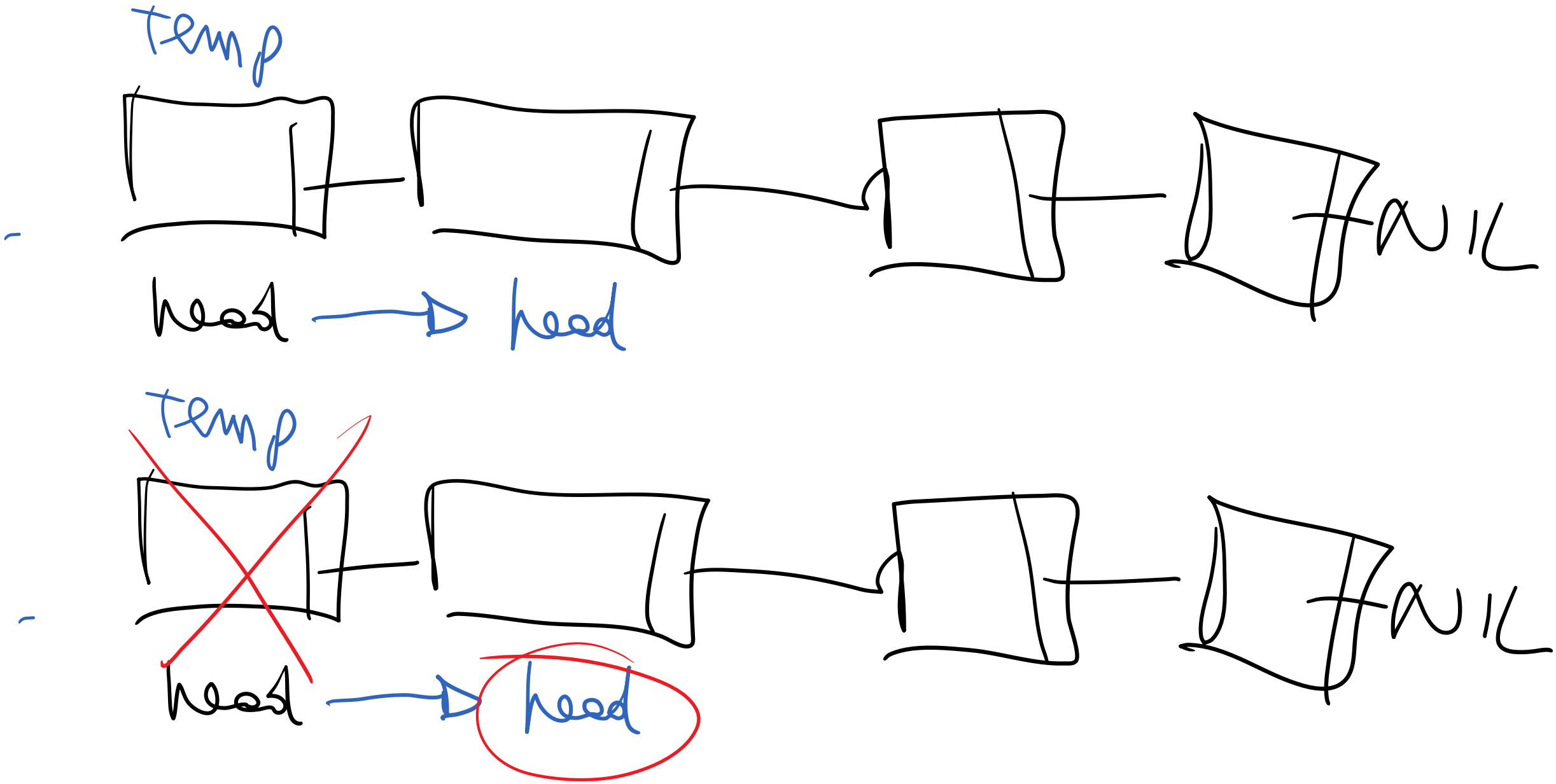

DELETE TAIL

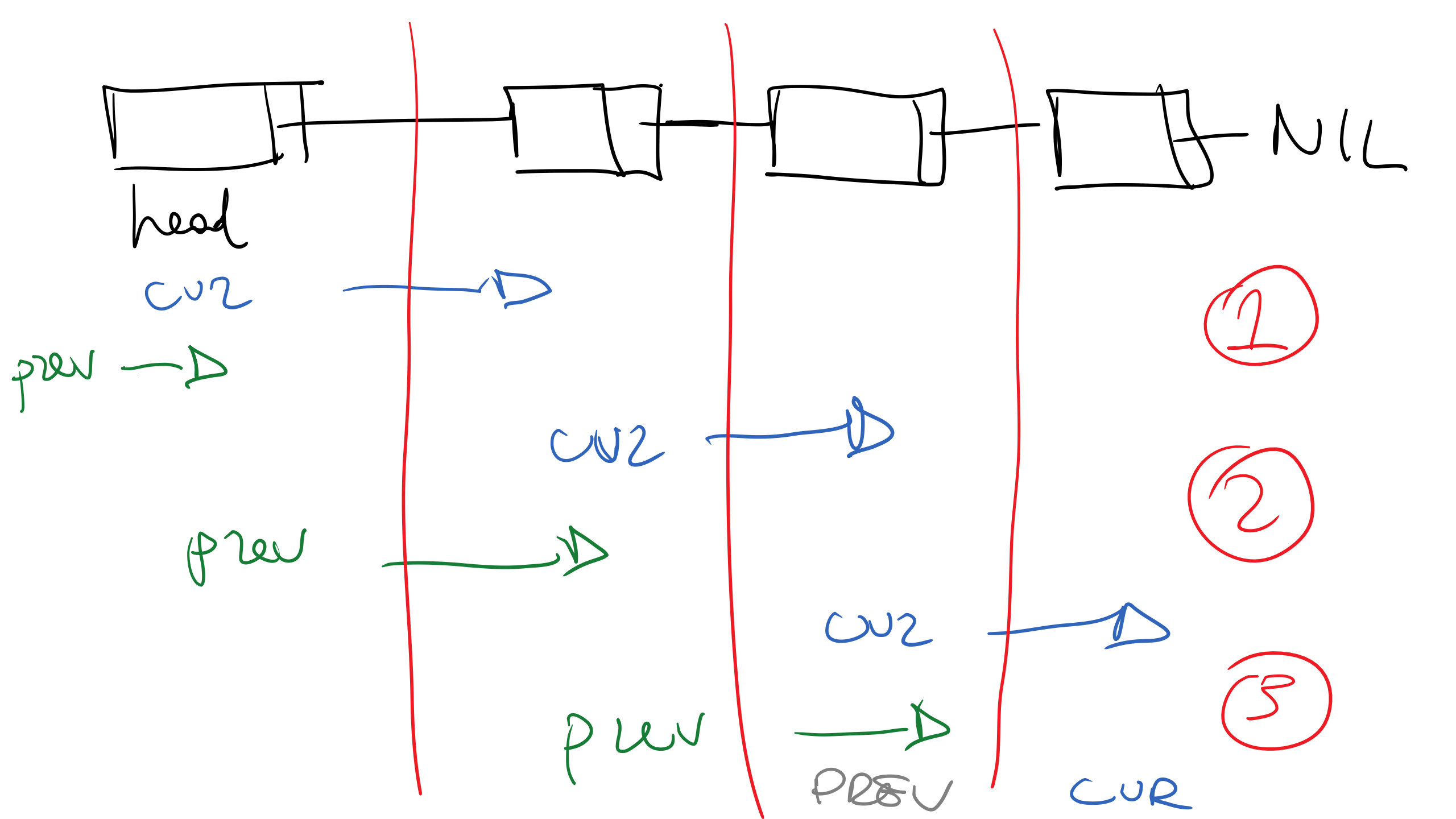



DELETE(4)

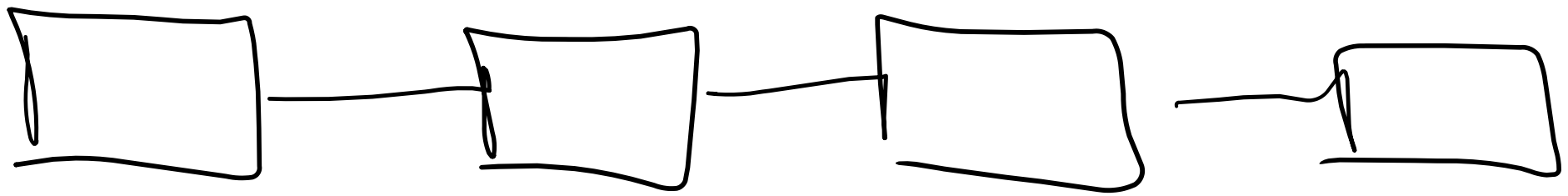



temp
head
head
NIL
temp
head
head
NIL


NIL
head
cur
prev
cur
prev
cur
prev
PREV
CUR
1
2
3

# esercizi

- Aggiungere alla classe lista un contatore che tenga conto del numero di elementi inseriti, e quindi della dimensione della lista
- Aggiungere un puntatore alla coda e modificare tutti i metodi che accedono alla coda
- Implementare l'operatore di accesso ( [] ) sfruttando la conoscenza del numero di elementi inseriti (potrebbe richiedere modifiche al nodo)
- Valutare la complessità computazionale di inserimento e cancellazione sia della lista implementata a lezione che della lista con le modifiche apportate.

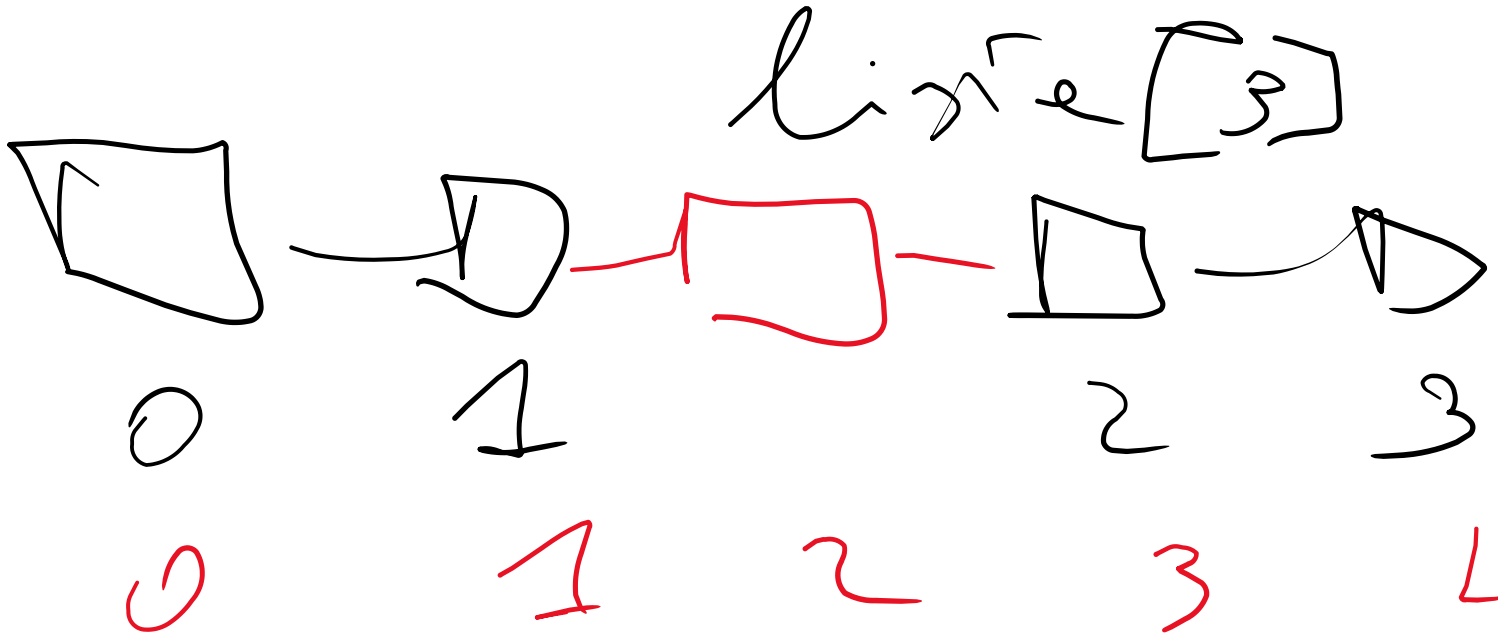


4 nodes

int size = 0

insert → size++;

remove → size--;




Liste semplici n=elementi

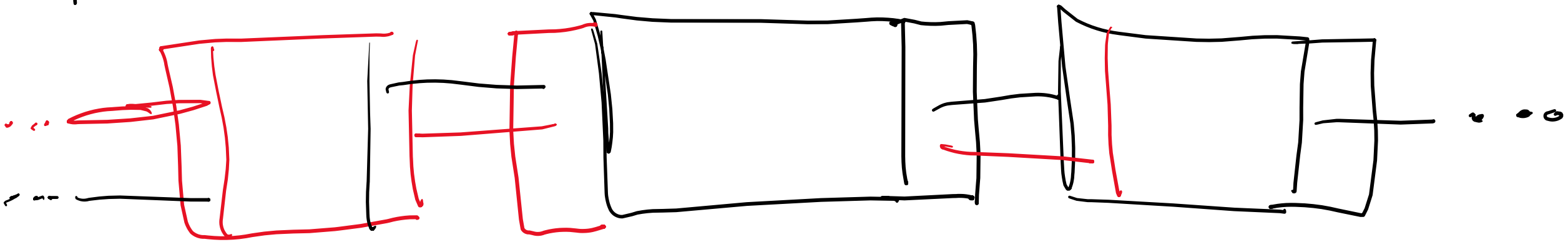
INSERT-READ → O(1)

INSERT-TAIL → O(n)

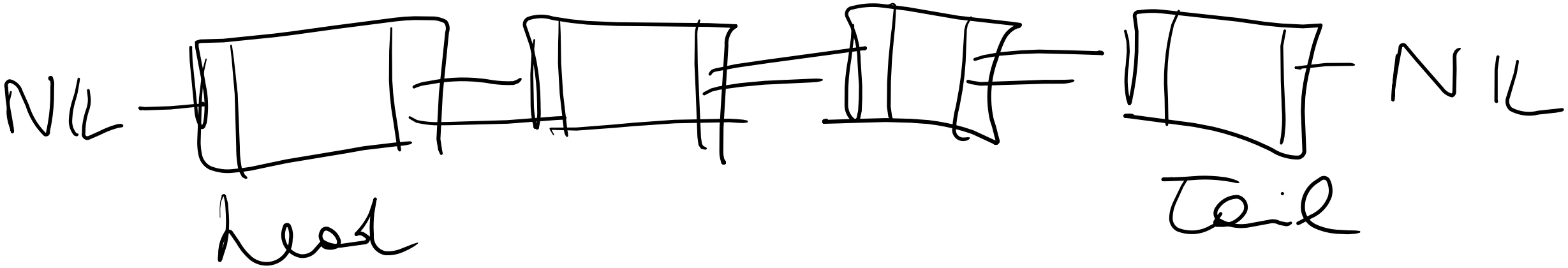
(INSERT-TAIL con PUNT. ALLA CODA → O(1))

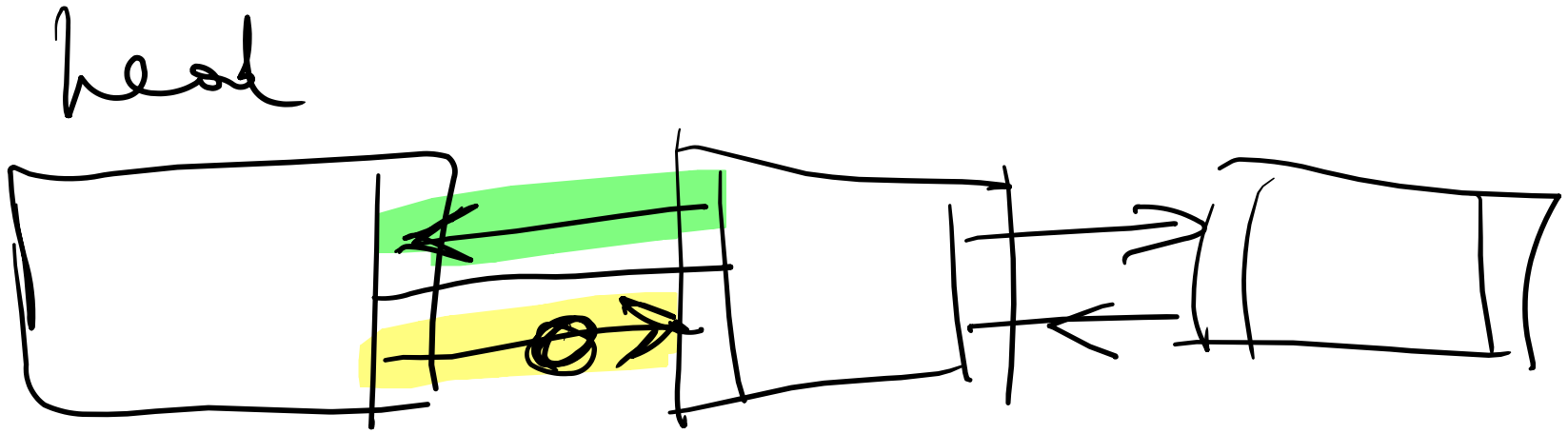
# LISTE DOPPIAMENTE LINKATE

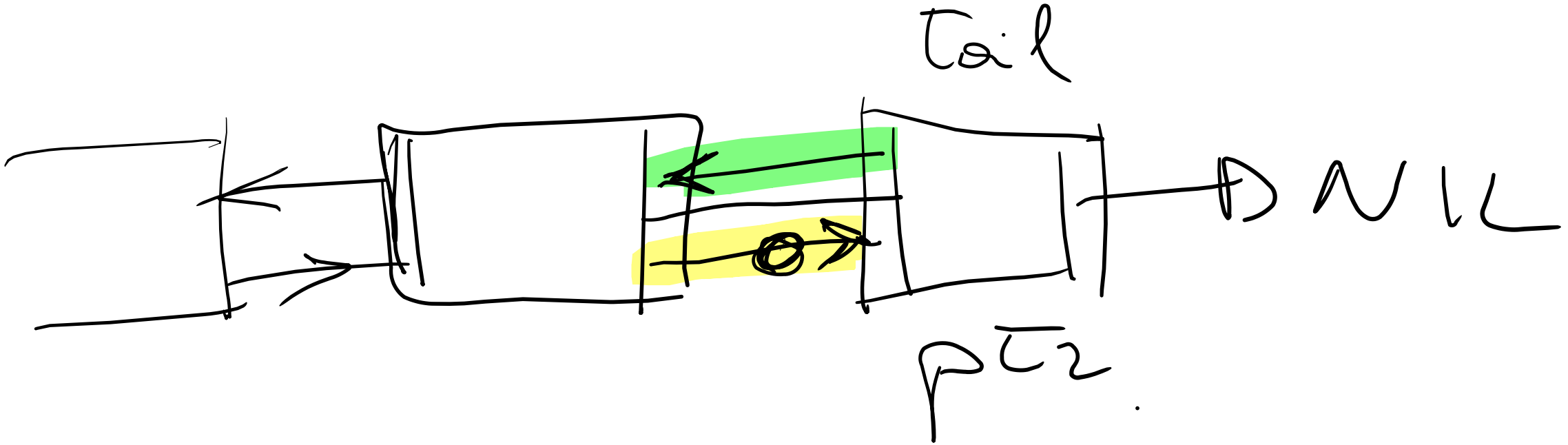
## NODO



## LISTA




head
ptr
ptr->next
ptr->next->prev

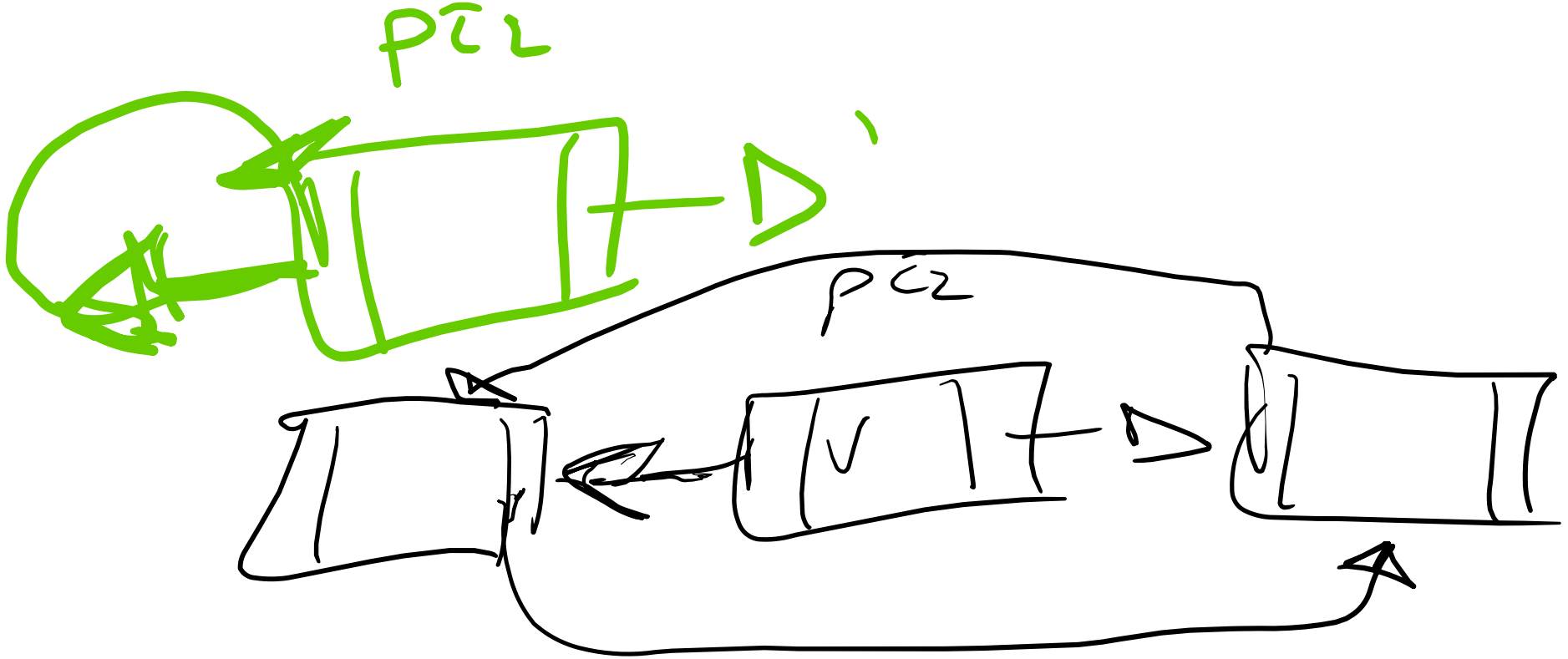


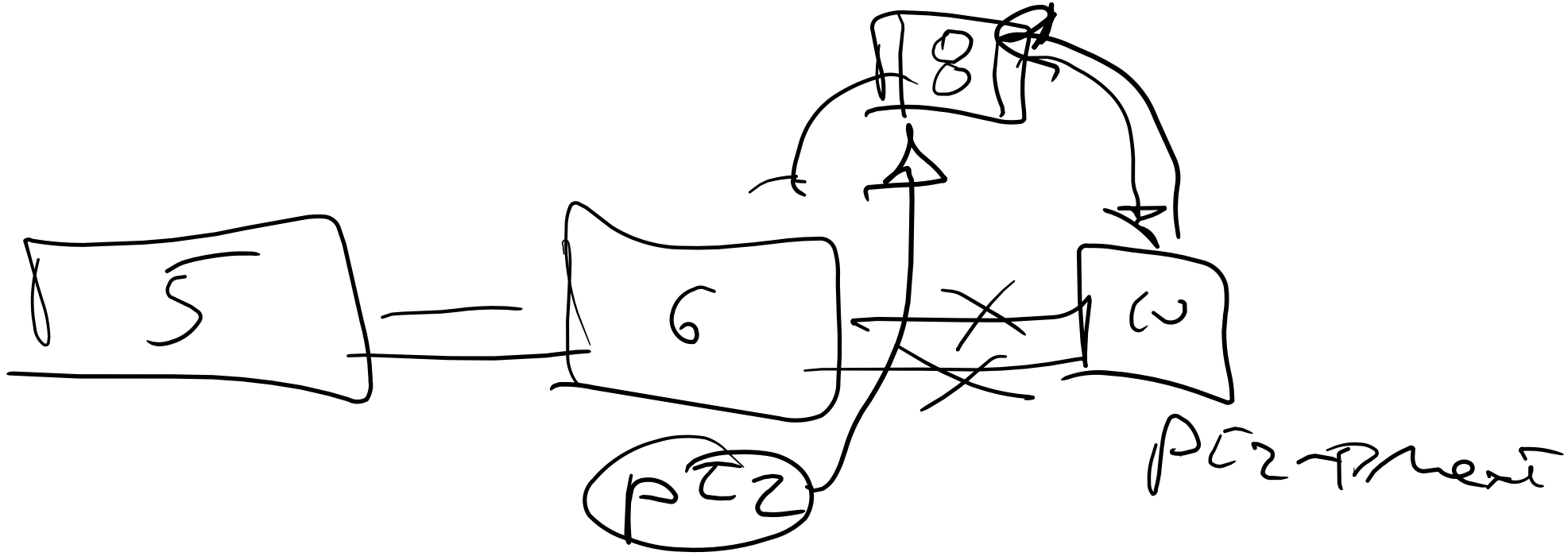

ptr2->prev

ptr->prev->next

# esercizio

- Implementare l’inserimento ordinato e la cancellazione di un elemento dato il valore nelle DLList






pInsert->prev = pz

cInsert->next = pz->next

pz->next->prev = cInsert

pz->next = cInsert