

# Parte 4

## Liste



P. Picasso – Guernica, 1937

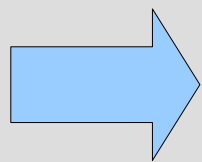
# Strutture dati complesse

- Per la risoluzione di alcuni problemi vi è la necessità di utilizzare ***strutture dati dinamiche e complesse***
- Si consideri ad esempio un problema che operi su una ***sequenza ordinata di valori, il cui numero non è noto a priori, effettuando inserimenti ed estrazioni frequenti***
- Finora per memorizzare **elementi omogenei** abbiamo utilizzato degli array

# Limiti degli array (1)

## *Occupazione di memoria*

- La dimensione dell'array deve essere definita nella parte dichiarativa del programma o comunque al momento dell'allocazione (se dinamico)
- Se il numero di valori non è noto a priori è necessario ***sovrastimarlo***

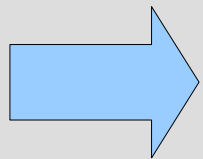


Occupazione di memoria potenzialmente molto maggiore del necessario

# Limiti degli array (2)

## *Velocità di esecuzione*

- Caso di **inserimento**:
  - Ricerca del punto d'inserimento
  - Shift di un posto verso il basso di tutti gli elementi successivi
- Caso di **estrazione**:
  - Shift di un posto verso l'alto di tutti gli elementi successivi



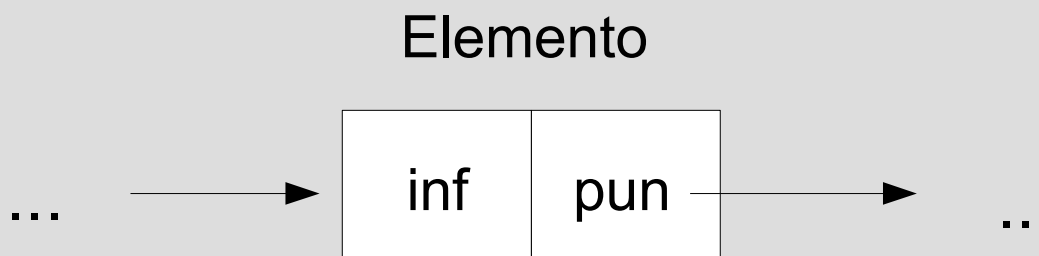
**Inefficiente per frequenti inserimenti/estrazioni**

# Soluzioni?

- I limiti degli array sono dovuti alla **scarsa flessibilità** della loro struttura
  - Elementi successivi dell'array devono essere collocati in locazioni contigue di memoria
- *E se ogni volta che dovessimo allocare un elemento nuovo, lo potessimo allocare in memoria da solo?*
- *E quando lo dovessimo estrarre, lo deallocassimo di nuovo da solo?*
- *Ma come **colleghiamo** i vari elementi?*

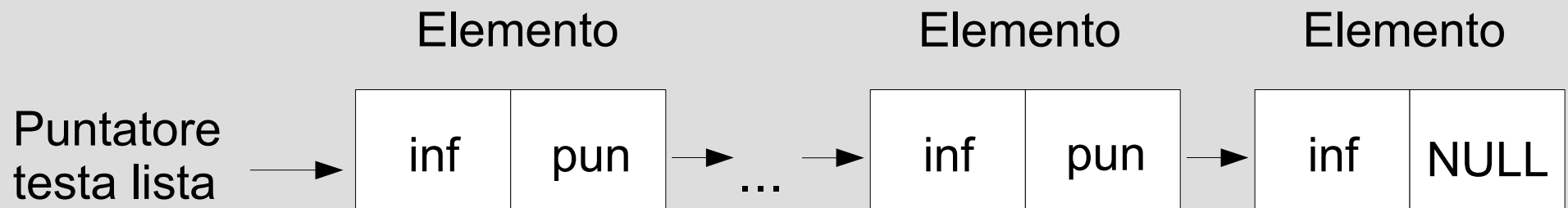
# Lista

- Successione di elementi omogenei ognuno dei quali occupa in memoria una posizione indipendente dagli altri
- Gli elementi sono collegati uno all'altro da un puntatore
- Ciascun elemento contiene un ***campo informazione*** (di qualunque tipo) ed un ***campo puntatore*** all'elemento successivo



# Testa e coda della lista

- L'accesso alla lista avviene attraverso il **puntatore alla testa della lista**
  - Puntatore al primo elemento della lista
  - Deve essere mantenuto in una variabile (globale o locale)
- Il **puntatore dell'ultimo elemento** della lista non fa riferimento a nessun elemento (valore **NULL**)



# Terminologia

- **Lista semplice o singolarmente concatenata**: ogni elemento contiene un unico puntatore che lo collega all'elemento successivo
- **Lista doppia o doppiamente concatenata**: ogni elemento due puntatori, uno all'elemento precedente e uno al successivo
- **Testa** (o **head**) della lista = primo elemento della lista
- **Coda** (o **tail**) della lista = ultimo elemento della lista
- **Lista vuota** = lista senza elementi identificata da un puntatore alla testa della lista avente valore **NULL**



# Caratteristiche di una lista semplice

- Diversamente dagli array, gli elementi in memoria occupano ***posizioni non sequenziali***
- Diversamente dagli array, in cui l'ordine è determinato dagli indici, l'***ordine è determinato da un puntatore in ogni elemento***
- Per determinare la fine della lista è ***necessario il segnale di fine lista NULL***
- Non esiste modo di risalire da un elemento al suo antecedente → ***scansione possibile solo da un elemento verso il successivo***

# Struttura dati in C++

**Elemento** implementato attraverso una **struct**:

```
struct elem {  
    int inf;        // o qualsiasi tipo semplice  
    elem* pun;     // definizione ricorsiva  
};
```

```
elem *testa; // puntatore alla testa
```

```
// oppure:
```

```
typedef elem* lista;  
lista testa;
```

# Osservazioni (1)

- Definito il tipo:

```
struct elem {  
  int inf;  
  elem* pun;  
};
```

- La seguente istruzione:

```
elem *p;
```

crea oppure no un oggetto dinamico di tipo *elem* ?

*No, crea solo il puntatore p*

# Osservazioni (2)

Dunque l'istruzione

```
elem *p;
```

deve essere seguita da un'allocazione

```
p = new elem;
```

(che alloca un elemento dinamico) prima di poter accedere ai campi dell'elemento puntato da *pun*:

```
p->inf
```

```
p->pun
```

***Prima di dereferenziare un puntatore assicurarsi che punti ad un oggetto dinamico correttamente allocato!***

# Stampa della lista

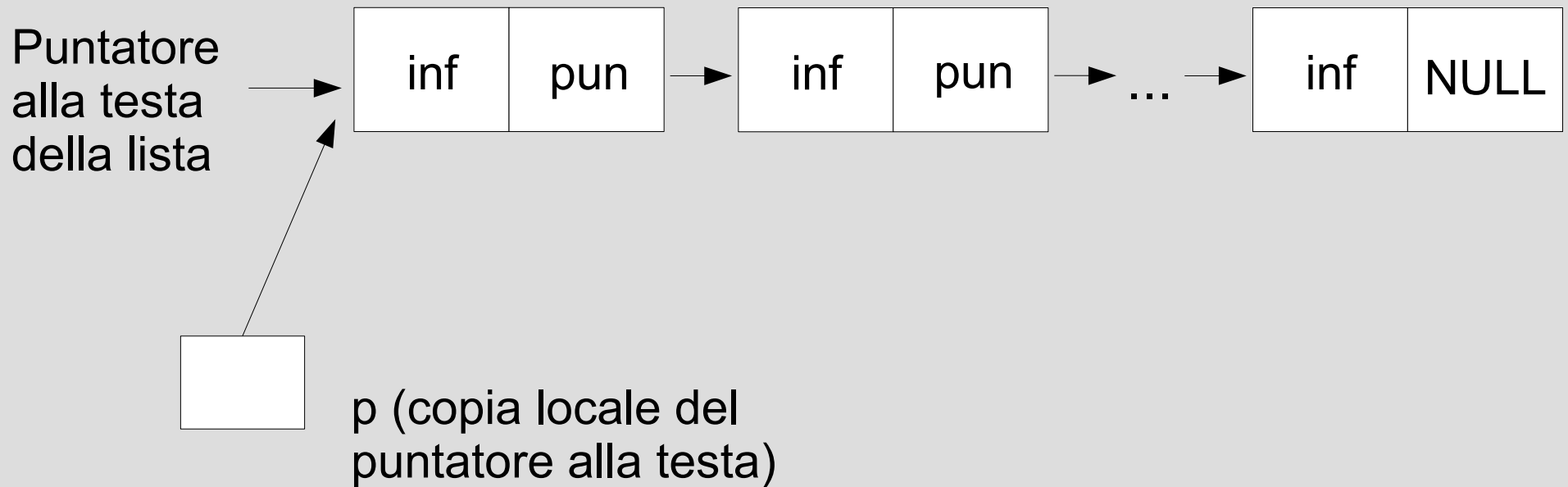
Problema da risolvere: **scandire una lista esistente fino alla coda**

```
void stampalista(lista p)
{ while (p != NULL) {
  cout << p->inf << " " ; // stampa valore
  p = p->pun ;           // spostamento sul
                          // prossimo elemento
}
cout << endl ;
}
```

*stampalista() prende in ingresso il puntatore alla testa della lista*

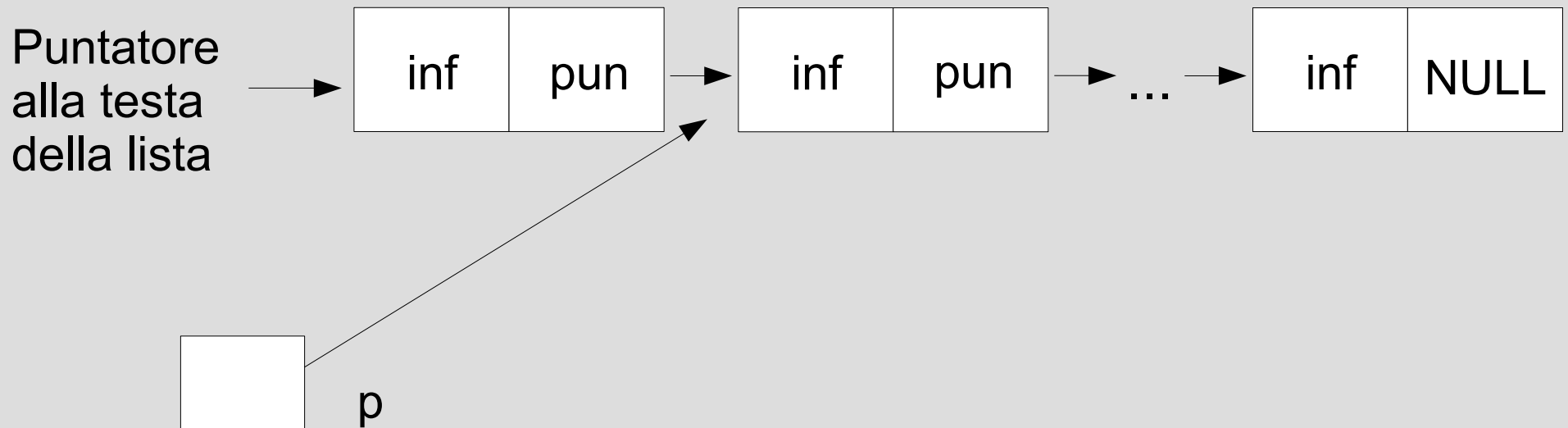
# Iterazioni di stampa

- Prima iterazione



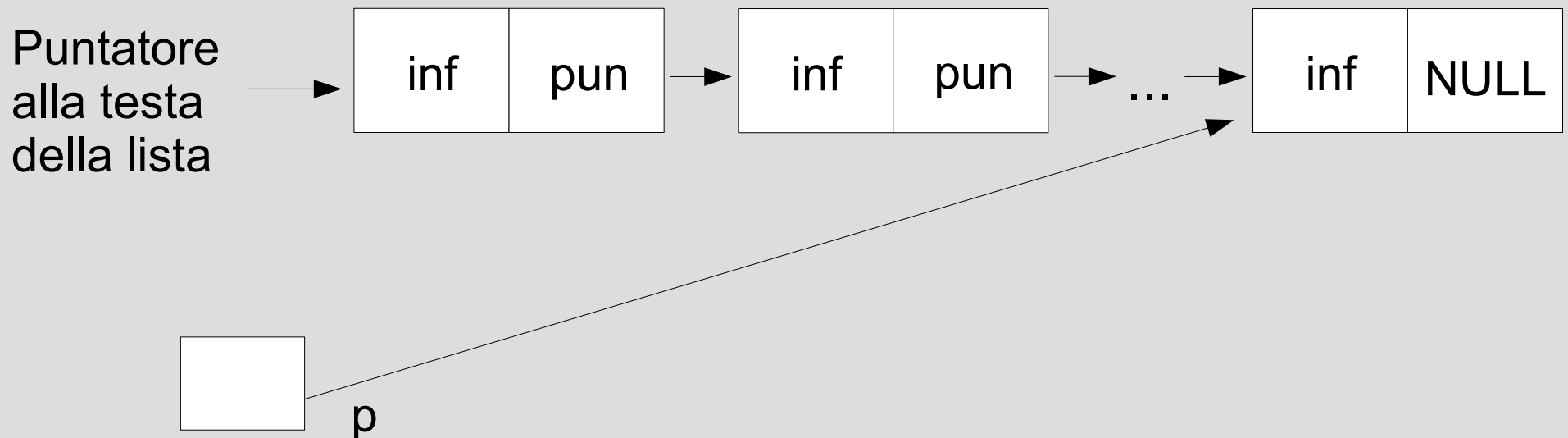
# Iterazioni di stampa

- Seconda iterazione



# Iterazioni di stampa

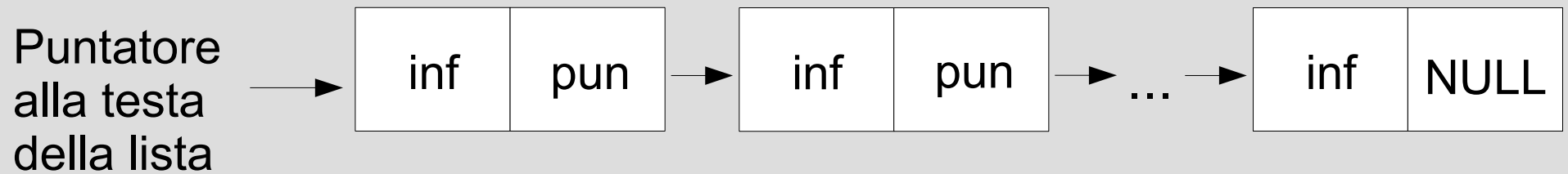
- Ultima iterazione





# Iterazioni di stampa

- Dopo l'ultima iterazione



NULL p

# Esercizio

Scrivere un programma in cui:

- sia definita (a tempo di scrittura del programma stesso) una lista formata da due elementi, contenenti i valori 3 e 7
- si stampi il contenuto della lista mediante la funzione **stampalista**
- Programma *stampa\_elem2.cc*

# Soluzione

```
main() {  
    lista testa = new elem;  
  
    testa->inf = 3;  
    elem * p = new elem; // creo l'elemento  
    p->inf = 7;  
    p->pun = NULL;  
    testa->pun = p; //aggancio l'elemento  
    stampalista(testa);  
}
```

Programma *stampa\_elem2\_sol.cc*

# Creazione di una lista

- **Algoritmo per la creazione di una lista semplice di N elementi**
- Gli elementi devono apparire nella lista in **ordine inverso** rispetto a quello con cui sono inseriti da utente (***inserimento in testa***)
- Cominciamo dal creare una lista di due elementi con valori inseriti pari a 2 e 5  
→ la lista finale dovrà contenere elementi di valore 5 e 2 nell'ordine

# Struttura dati iniziale

?

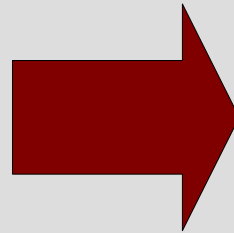
Puntatore alla  
testa della lista

NULL

Puntatore alla  
testa della lista

?

Puntatore  
all'ultimo  
elemento  
creato



*Creazione  
lista vuota*

?

Puntatore  
all'ultimo  
elemento  
creato

# Creazione primo elemento



# Inizializzazione primo elemento



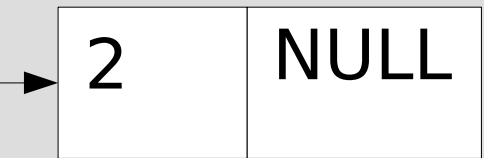
# Inizializzazione primo elemento





# Aggancio primo elemento

Puntatore alla testa della lista

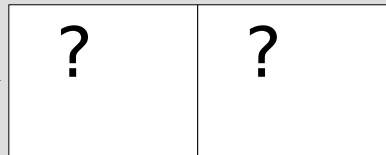
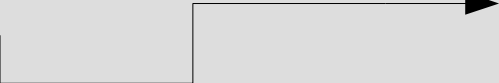
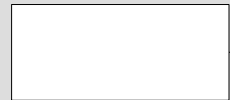
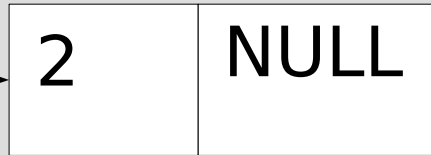


Puntatore all'ultimo elemento creato

Aggiornamento del puntatore alla testa della lista

# Creazione secondo elemento

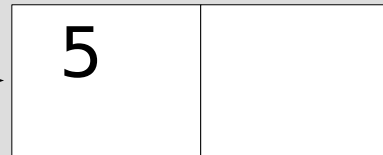
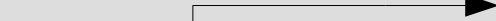
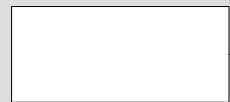
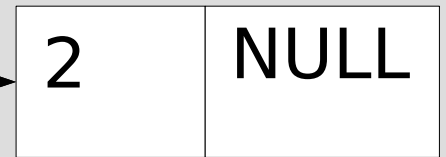
Puntatore alla testa della lista



Puntatore all'ultimo elemento creato

# Inizializzazione secondo elemento

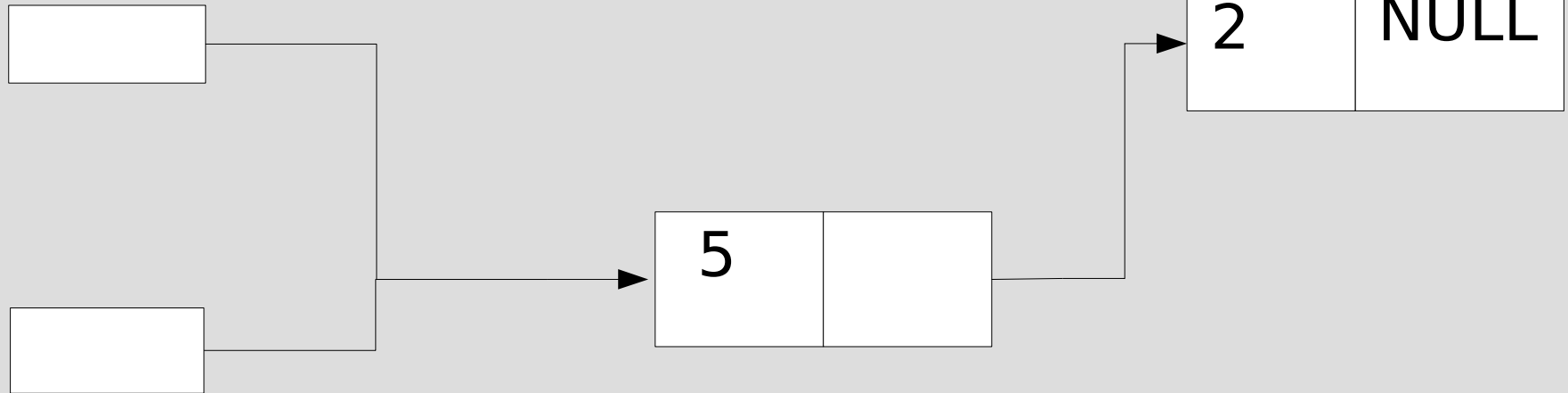
Puntatore alla  
testa della lista



Puntatore  
all'ultimo  
elemento  
creato

# Aggiornamento testa lista

Puntatore alla testa della lista



Puntatore all'ultimo elemento creato

# Algoritmo

- Creazione di una lista vuota
  - Definizione del puntatore alla testa e sua inizializzazione a NULL
- Ciclo
  - Creazione di un elemento
  - Inizializzazione del campo informazione
  - Aggancio dell'elemento alla lista
    - Inizializzazione del campo puntatore con l'indirizzo della testa
    - Aggiornamento del puntatore alla testa della lista

# Aggiunta di un elemento

- 2 casi possibili
- Aggiunta di un elemento ad una lista vuota (primo elemento)
  - il campo ***pun*** assume il valore **NULL**
- Aggiunta di un elemento ad una lista non vuota
  - il campo ***pun*** assume l'indirizzo della (precedente) testa della lista
- E' necessario distinguere i due casi nell'algoritmo?

# Programma

- *crea\_stamp\_lista.cc*
- Si scriva la funzione **crealista**, che prende in ingresso un numero intero  $n$  e crea una lista di  $n$  elementi, inizializzati con valori inseriti dall'utente. Gli elementi devono apparire nella lista nell'ordine inverso in cui sono stati inseriti (***inserimento in testa***).

Inoltre, la funzione deve ritornare l'indirizzo della testa della lista creata (puntatore alla testa)

# Eliminazione di una lista

- Completare *crea\_stamp\_elim\_lista.cc* scrivendo la funzione **eliminalista**, che riceve come parametro il puntatore alla testa di una lista (passato attraverso un riferimento!) ed elimina la lista stessa
  - Ispirarsi alla stampalista per quanto riguarda la scansione
  - Attenzione ai puntatori pendenti!!!



# Uso di delete (1)

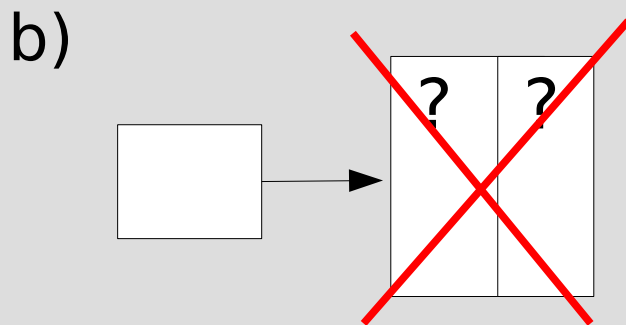
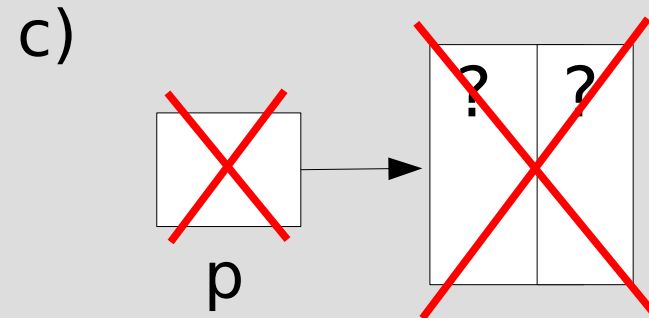
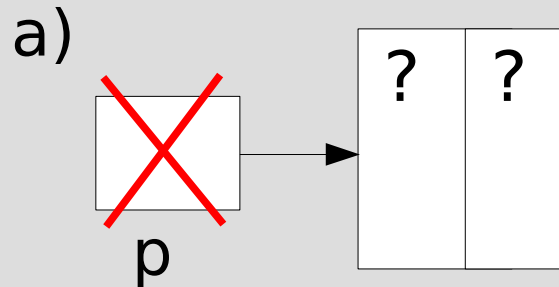
- La seguente sequenza di istruzioni è corretta?

```
elem *p = new elem;  
delete p;
```

***Sì, dealloca correttamente un oggetto allocato dinamicamente in memoria***

# Uso di delete (2)

- Cosa succede in memoria a seguito delle istruzioni precedenti?



***Si verifica il caso b)  
Il puntatore non  
viene deallocato***

# Uso di delete (3)

- Nel caso:

```
int * p = new int ;
```

```
delete p ;
```

cosa viene in realtà passato alla delete?

***La delete si aspetta un indirizzo: alla delete viene passato il valore memorizzato dentro p, che viene interpretato come l'indirizzo dell'oggetto da deallocare dalla memoria dinamica***

# Inserimento in testa

- Inserimento di un nuovo elemento in testa ad una lista già esistente
- Metodo già usato nella creazione di una lista
- La lista di partenza può essere vuota oppure non vuota e correttamente costruita (dobbiamo gestire entrambi i casi)
- Dati di partenza:
  - Puntatore alla testa della lista
  - Elementi della lista (se non vuota)

# Algoritmo di inserimento in testa

- Creazione nuovo elemento
- Inizializzazione del campo informazione
- Aggancio dell'elemento alla lista
  - Inizializzazione del campo puntatore con l'indirizzo della testa
  - Aggiornamento del puntatore alla testa della lista

# Programma

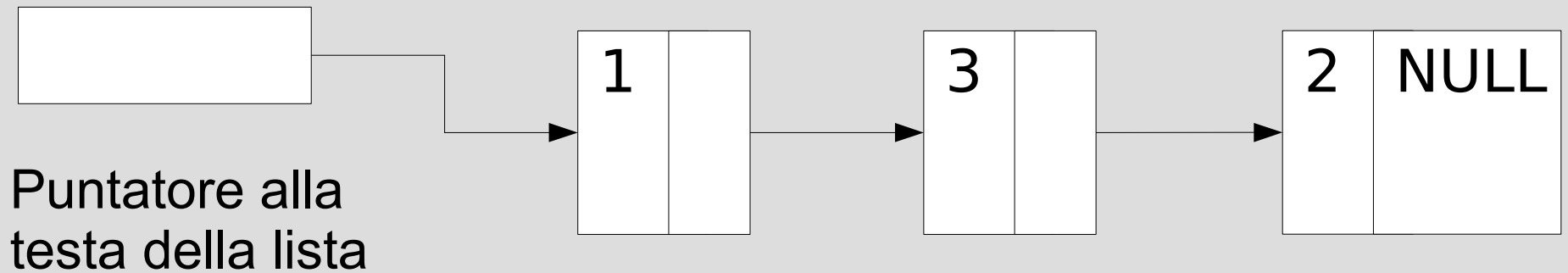
- *lista.cc*
- *Gestione di una lista semplice che contiene un messaggio di testo nel campo informazione*
- Presenti solo le funzioni stampalista, eliminalista ed ***inserisci\_in\_testa***
- Si suppone che `inserisci_in_testa` abbia valore di ritorno void

# Estrazione dalla testa

- Si provi ora ad effettuare l'**estrazione di un elemento dalla testa della lista**
- *Osservazione preliminare: cosa bisogna fare se la lista è vuota?*
  - Niente
- Vediamo il caso di lista non vuota

# Estrazione dalla testa

- Lista prima dell'estrazione dalla testa

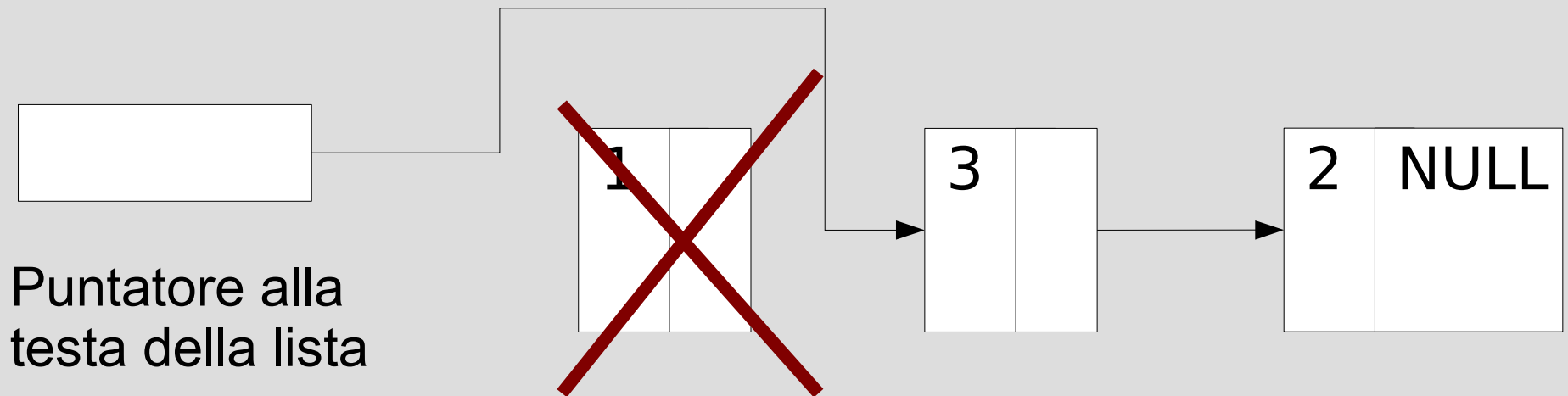


1. Controllo che la lista non sia vuota



# Estrazione dalla testa

- 2. Sgancio il primo elemento dalla lista (unlink)



3. Deallocazione dell'elemento

# Lista con un solo elemento

- Che succede se si estraee dalla testa di una lista contenente un solo elemento?
  - *Il puntatore alla testa della lista dovrà assumere valore NULL*
- Si tratta di un caso che dobbiamo controllare e gestire a parte?

**NO**

# Algoritmo

- ***Algoritmo di estrazione dalla testa:***
  - Controllo lista vuota
  - Sgancio dell'elemento
  - Deallocazione dell'elemento
- **ATTENZIONE A NON “PERDERE” L'ELEMENTO (MEMORY LEAK) PRIMA DI AVERLO DEALLOCATO!**

# Programma

- Realizzare anche la funzione ***estrai\_testa*** in *lista.cc*
  - Supporre che la ***estrai\_testa*** abbia **valore di ritorno bool** e ritorni **true** solo se l'estrazione ha successo, **false** se la lista è vuota
  - La funzione deve anche **“ritornare”** il valore del campo informazione dell'elemento estratto
  - Alla funzione viene passato il puntatore alla testa della lista
    - Come lo devo passare? Ne devo modificare il valore in modo permanente?

# Caratteristiche di `estrai_testa`

- Necessità di cambiare il valore del puntatore alla testa passato come parametro
  - **Utilizzo del riferimento (&)** nella definizione dei parametri della funzione
- Uso di un **puntatore ausiliario** per deallocare l'elemento dopo lo sgancio dalla lista
- Uso di un **parametro buf passato attraverso un puntatore** per consentire alla funzione di ritornare il valore del campo informazione dell'elemento estratto dopo averlo deallocato

# Domanda

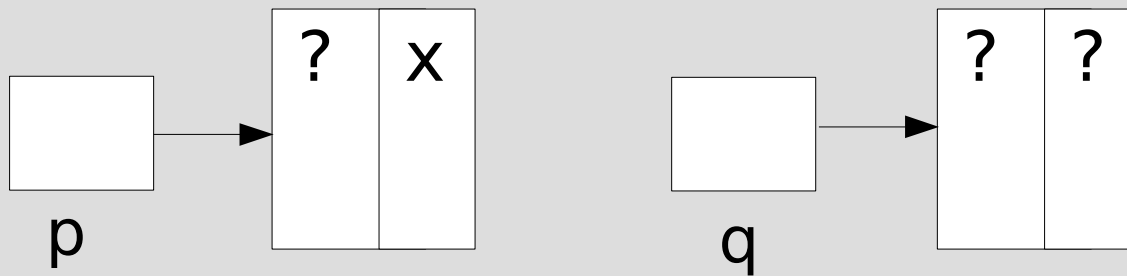
- Nel seguente frammento di codice

```
{    ...  
    elem *p = new elem;  
    ... // lavoro sull'elemento puntato da p  
    elem *q = new elem;  
    ... // lavoro sull'elemento puntato da q  
    // voglio agganciare l'elemento puntato  
    // da q dietro quello puntato da p  
    q = p->pun;  
}
```

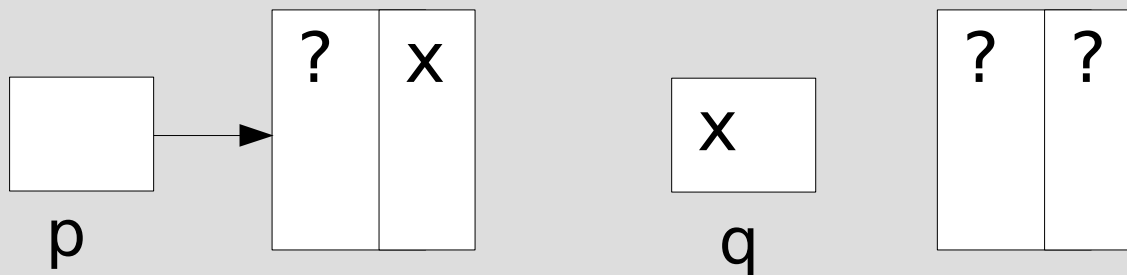
- L'ultima istruzione fa quanto riportato nel commento che la precede?

# Risposta

- No, in memoria accade semplicemente



dopo le prime due istruzioni, mentre, dopo l'ultima istruzione:

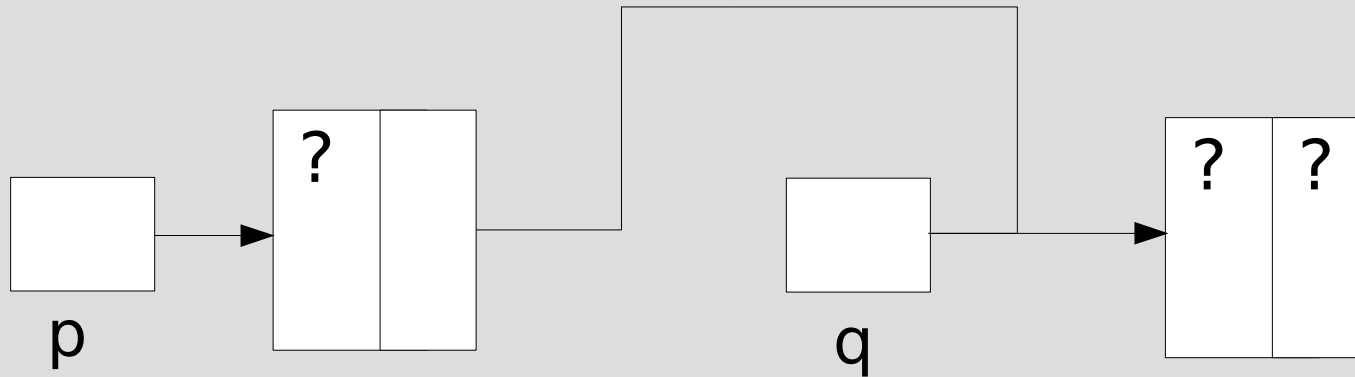


***Perdo anche il riferimento all'elemento puntato da q: memory leak!***

# Istruzione corretta

- L'istruzione corretta è:

```
p->pun=q;
```





# Inserimento in coda

Algoritmo che inserisce un elemento in fondo alla lista

1) Ricerca della coda della lista

- **Necessari puntatori ausiliari? Quanti?**

2) Creazione del nuovo elemento

3) Inizializzazione dei campi dell'elemento  
(informazione e puntatore)

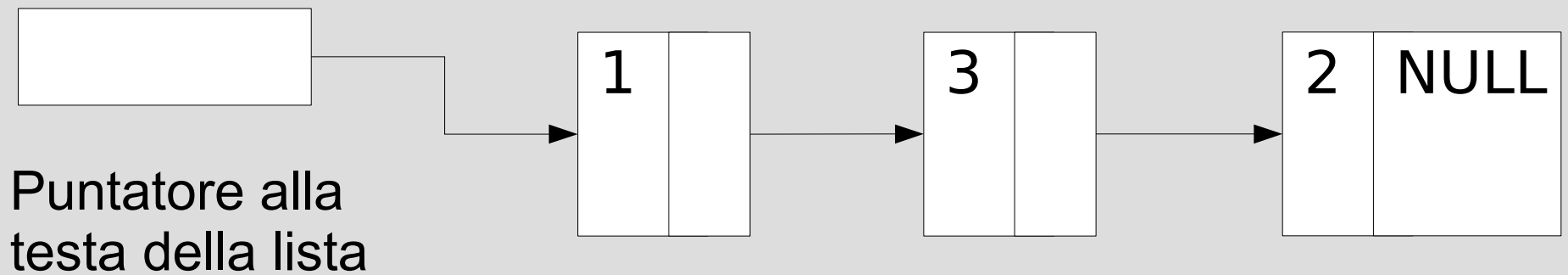
4) Aggancio del nuovo elemento

- **Lista vuota** → aggiornamento di puntatore a testa
- **Lista non vuota** → aggiornamento del campo puntatore dell'elemento che **era** l'ultimo nella lista

*Devo considerare i due casi separatamente?*

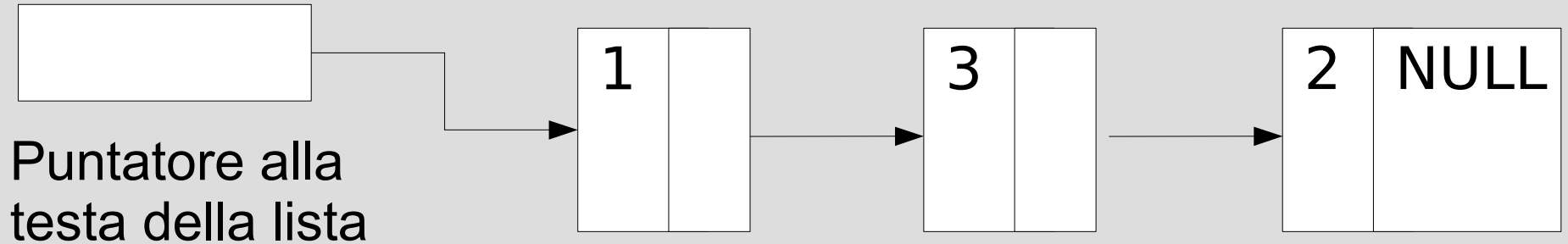
# Inserimento in coda


- Lista di partenza:



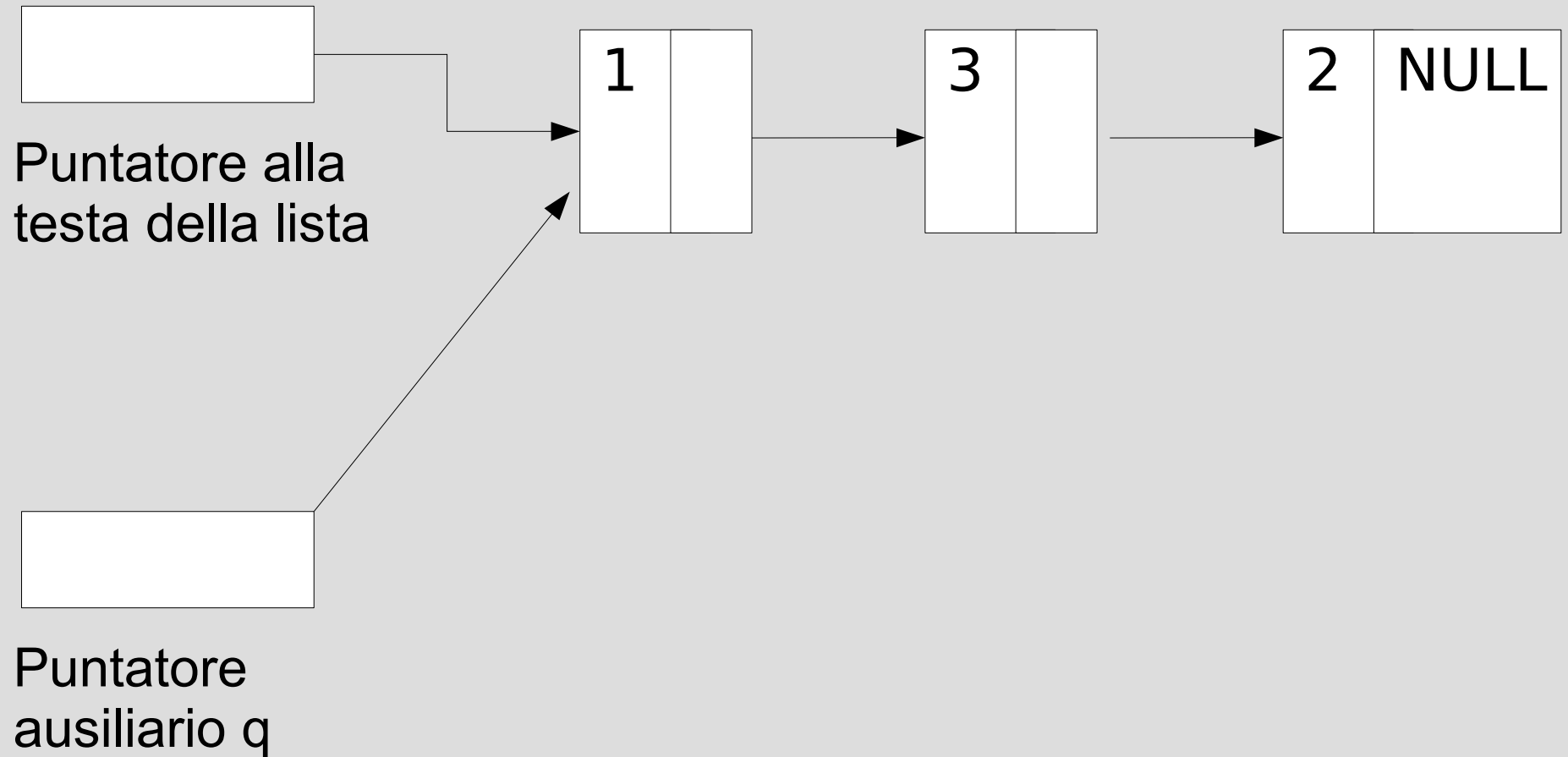
- Bisogna scorrere la lista: quanti puntatori usare?
  - Proviamo con l'uso di un solo puntatore

# Strutture dati

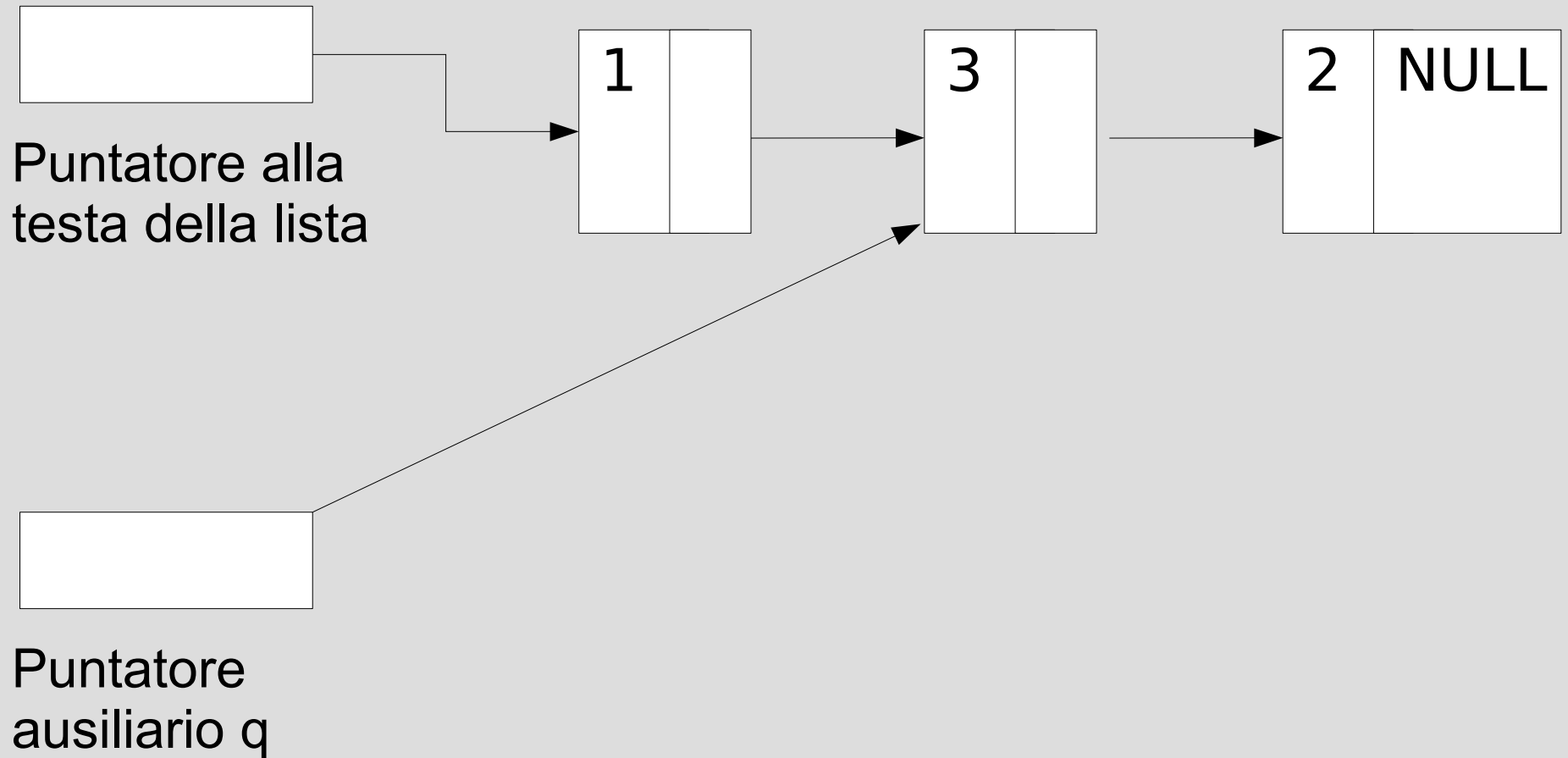


  
Puntatore ausiliario q

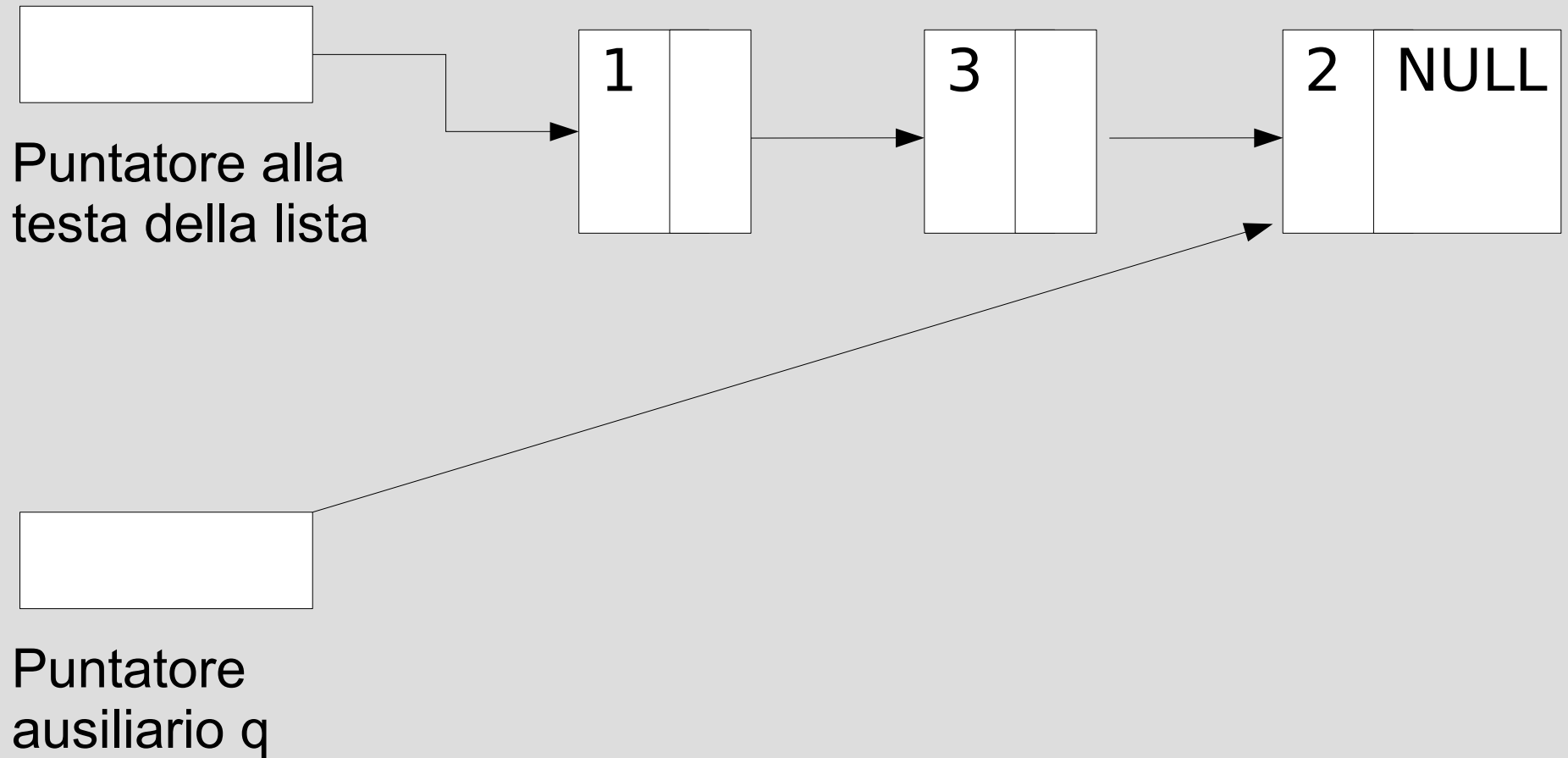
# Ricerca della coda



# Ricerca della coda



# Ricerca della coda



# Ricerca della coda

Quale condizione dobbiamo controllare per far procedere il ciclo di ricerca fino a far fermare *q* sull'ultimo elemento?

```
q->pun != NULL
```

Disponendo di un puntatore **testa** alla testa della lista, l'intestazione del ciclo sarebbe:

```
for (q = testa; q->pun != 0; q = q->pun)
```

E' un *ciclo sempre sicuro*?

# Osservazione

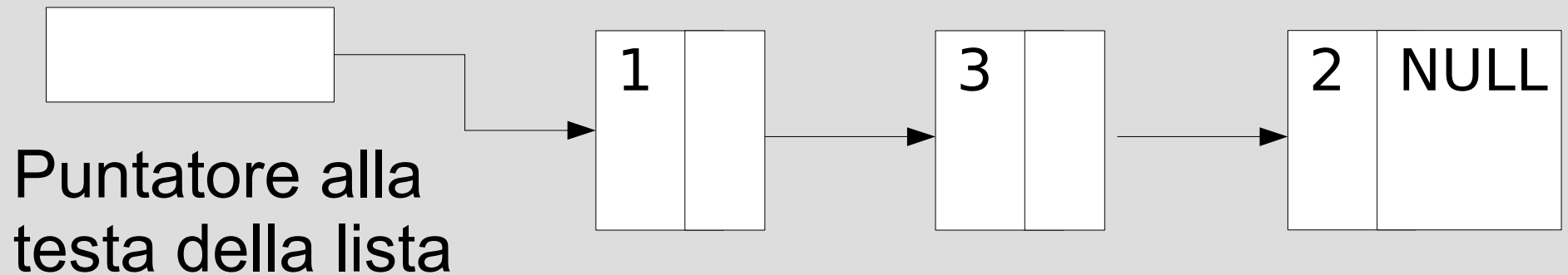
```
for (q = testa; q->pun != 0; q = q->pun)
```

- **Il ciclo precedente non è sempre sicuro**
  - Nel caso di lista inizialmente vuota si dereferenzia un puntatore nullo!
- Si rende necessario controllare che la lista non sia vuota prima di entrare nel ciclo



# Uso di due puntatori ausiliari

- Una soluzione è di usare **due puntatori ausiliari**
  - Un puntatore resta 'un passo indietro' rispetto all'altro

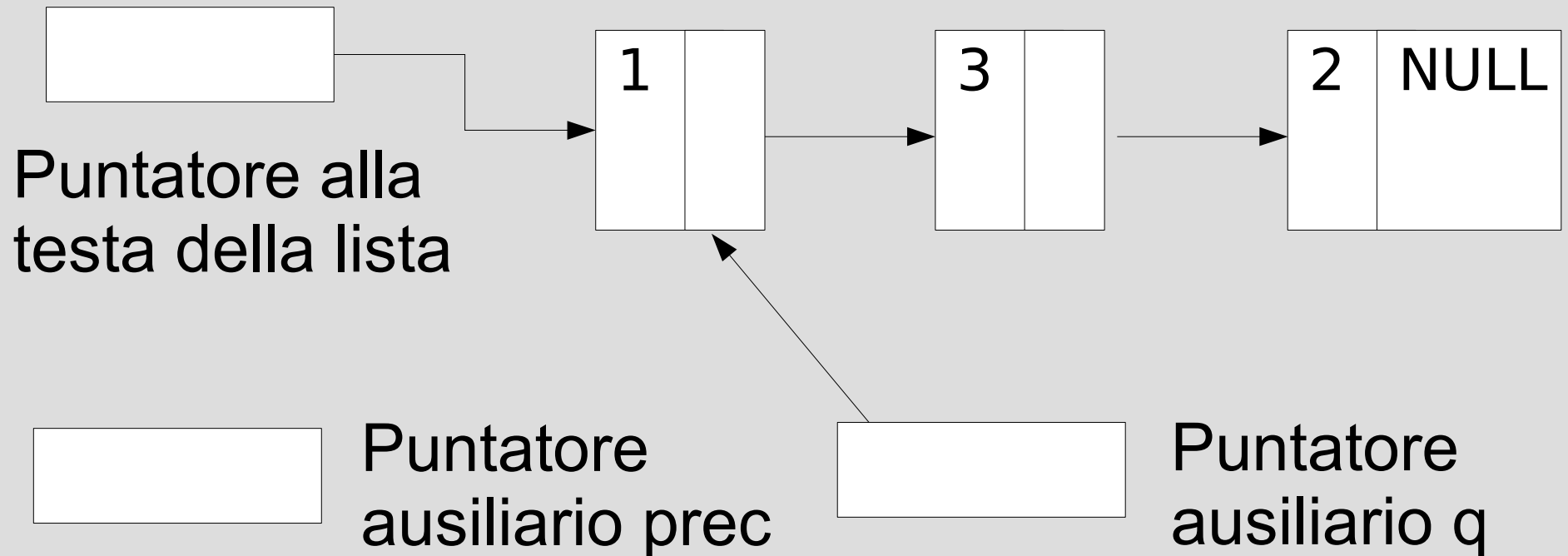


Puntatore ausiliario prec

Puntatore ausiliario q

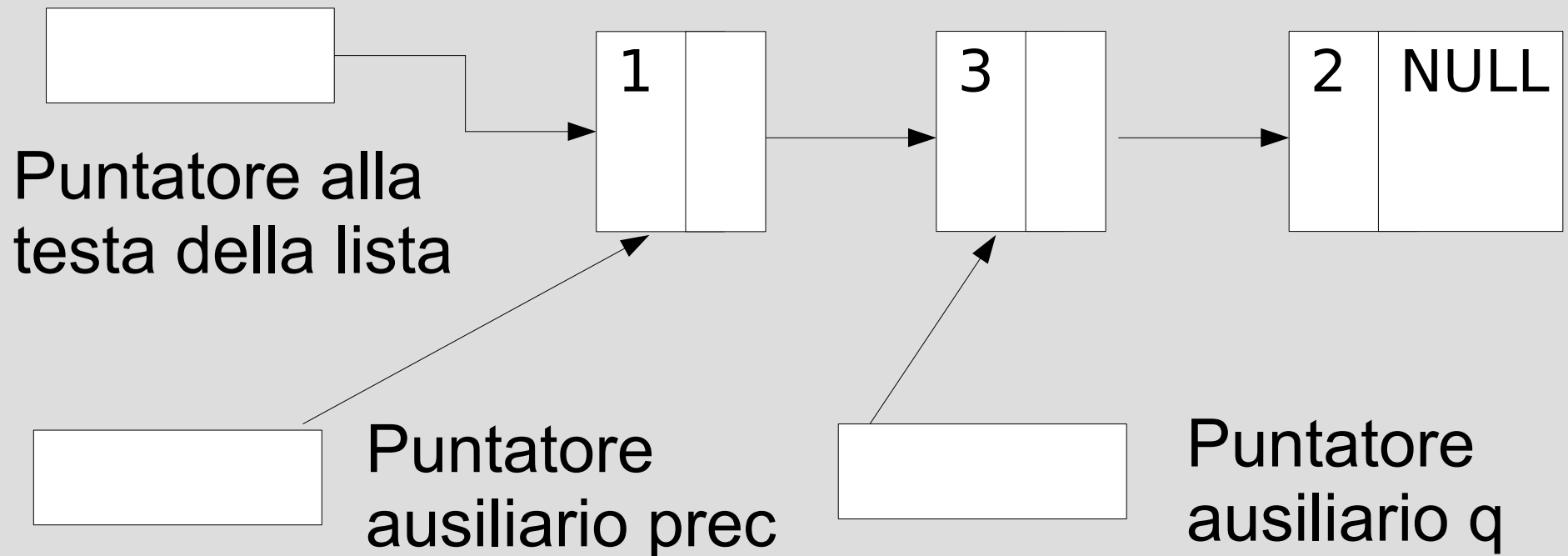
# Inserimento in coda

- Situazione iniziale: **q** punta alla testa della lista, **prec** non ancora inizializzato



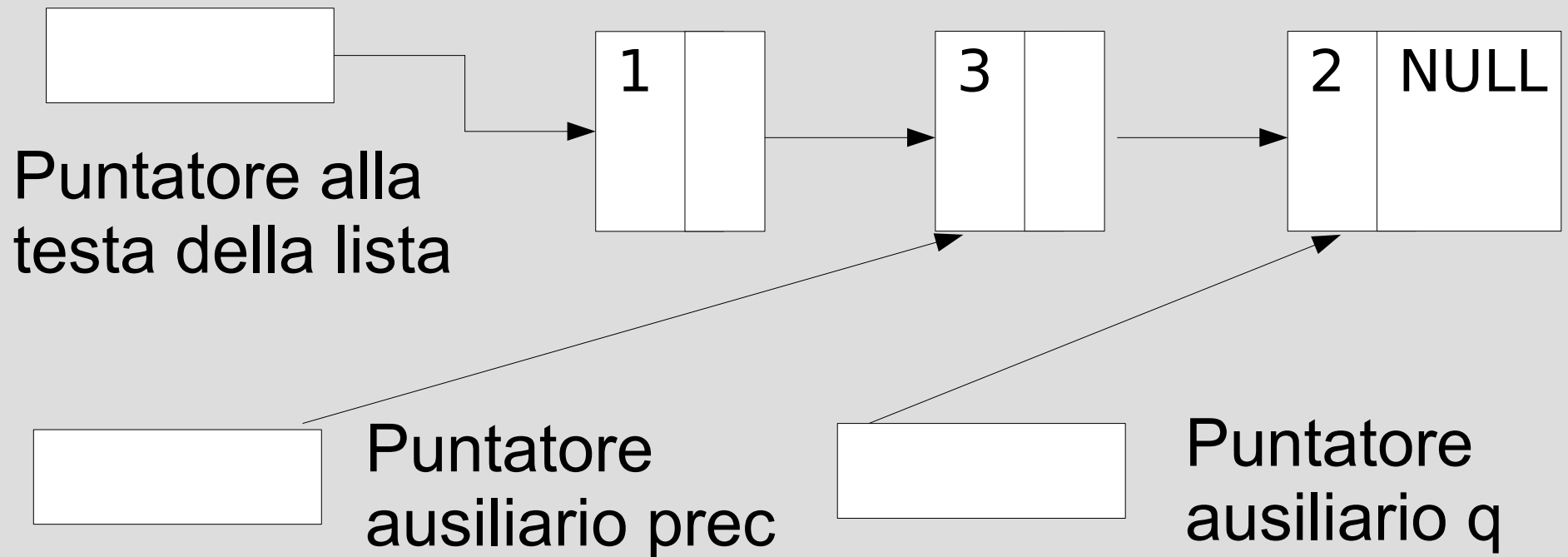
# Inserimento in coda

- **Se la lista non è vuota ( $q \neq NULL$ ),  $prec$  assume il valore di  $q$  e  $q$  viene fatto avanzare**



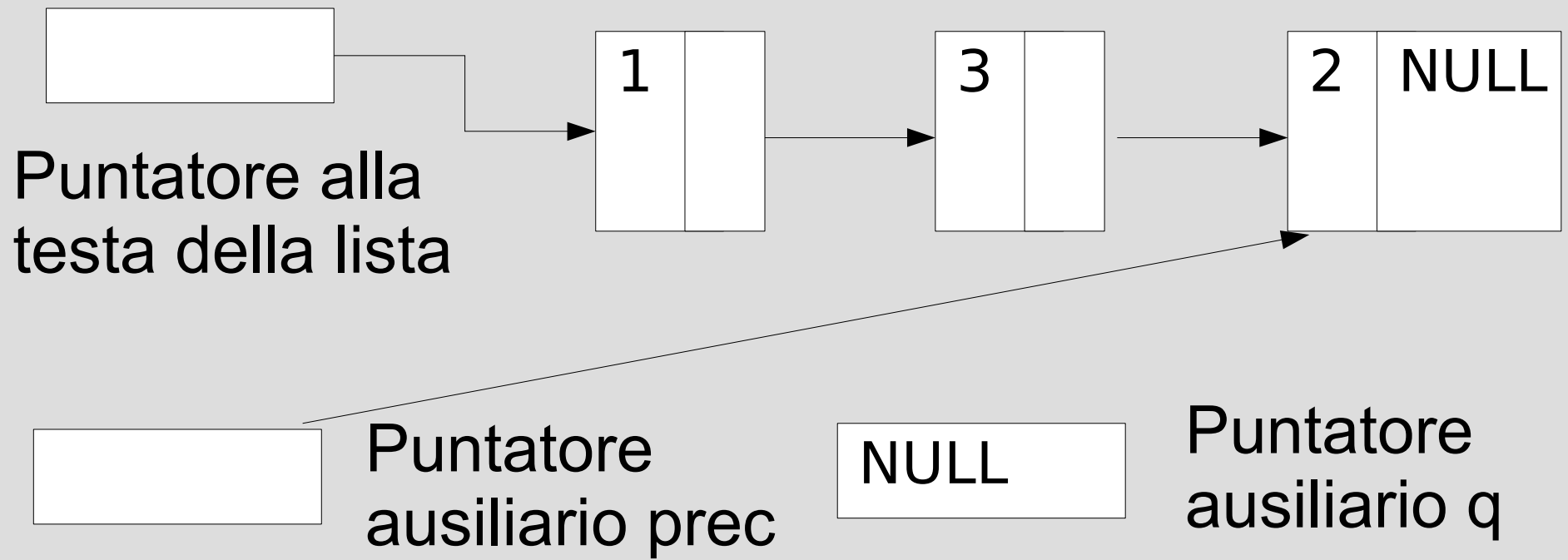
# Inserimento in coda

- *Step successivo*: se non si è finiti fuori dalla lista ( $q \neq \text{NULL}$ ), avanzano entrambi i puntatori



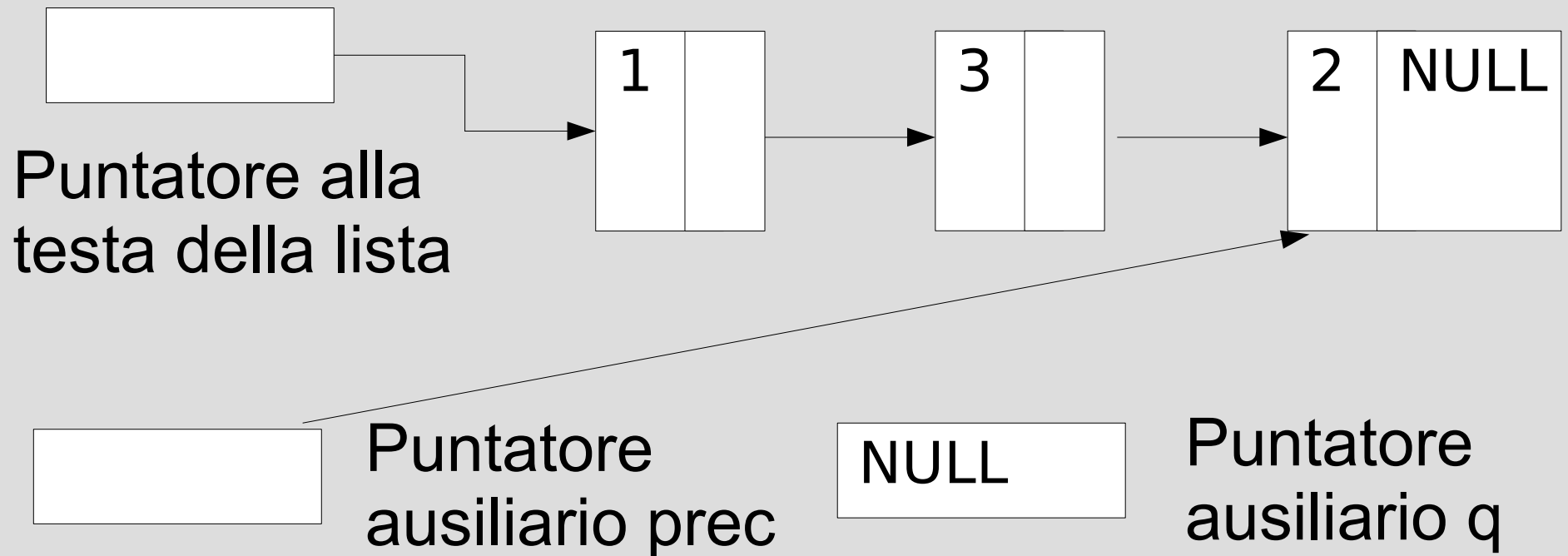
# Inserimento in coda

- *Ultimo step: q* avanza di nuovo e stavolta si trova fuori dalla lista (valore **NULL**), mentre *prec* si ferma sull'ultimo elemento



# Inserimento in coda

- A questo punto si esce dal ciclo senza far avanzare **prec**, che viene usato per inserire l'elemento in coda



# Condizione di uscita dal ciclo

Condizione da controllare per uscire dal ciclo:

`q != NULL`

Possono sorgere problemi di accesso alla memoria nel valutare questa condizione?

***No, perchè il puntatore non è dereferenziato***

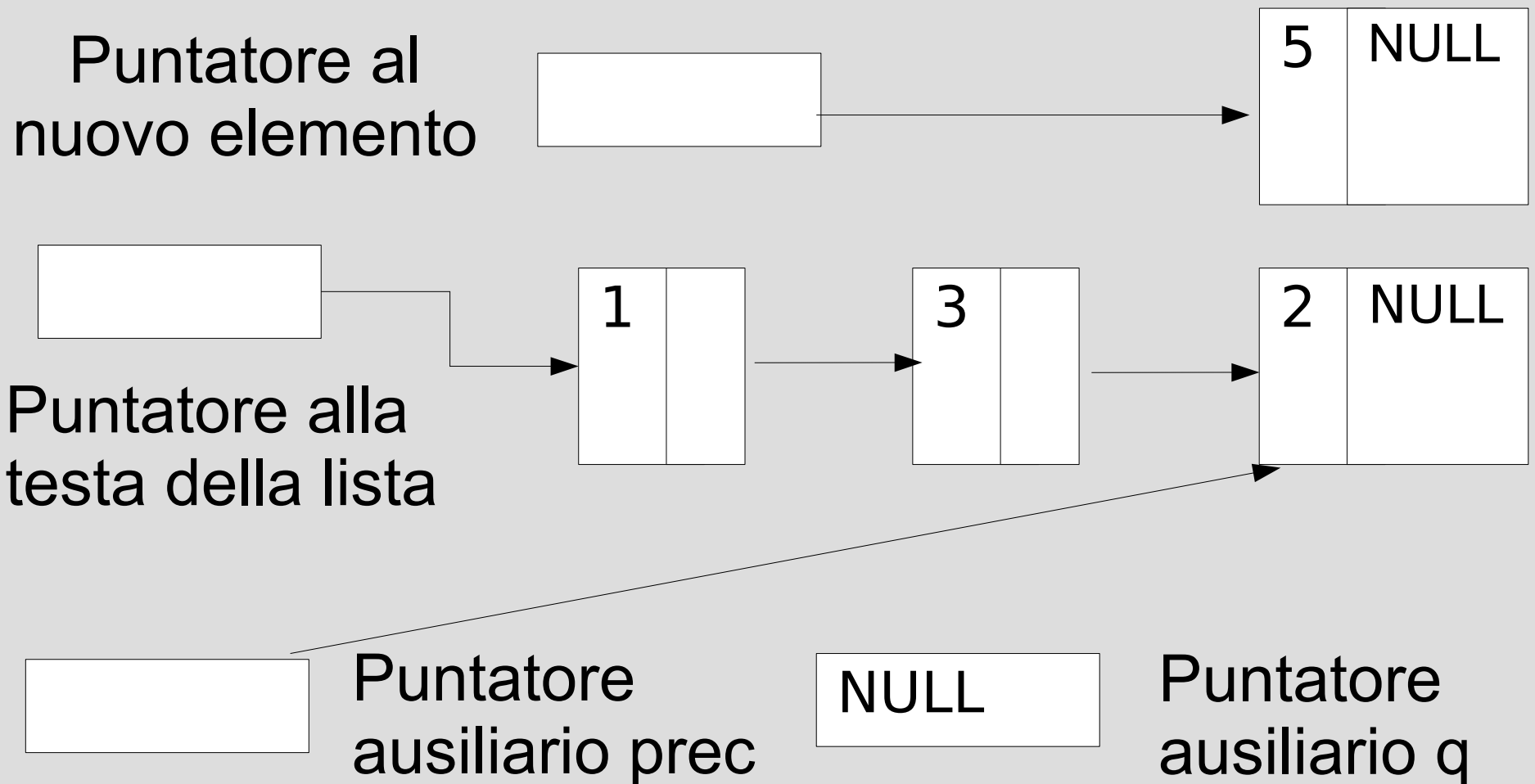
# Riassumendo

- In uscita dal ciclo, **prec** punta all'ultimo elemento della lista
- **prec** verrà utilizzato per agganciare il nuovo elemento in coda
  - Come vedremo, l'uso della ***coppia di puntatori*** è comodo anche per gli inserimenti nel mezzo di una lista ordinata (***inserimento in ordine***)
- Proseguiamo con l'algoritmo...



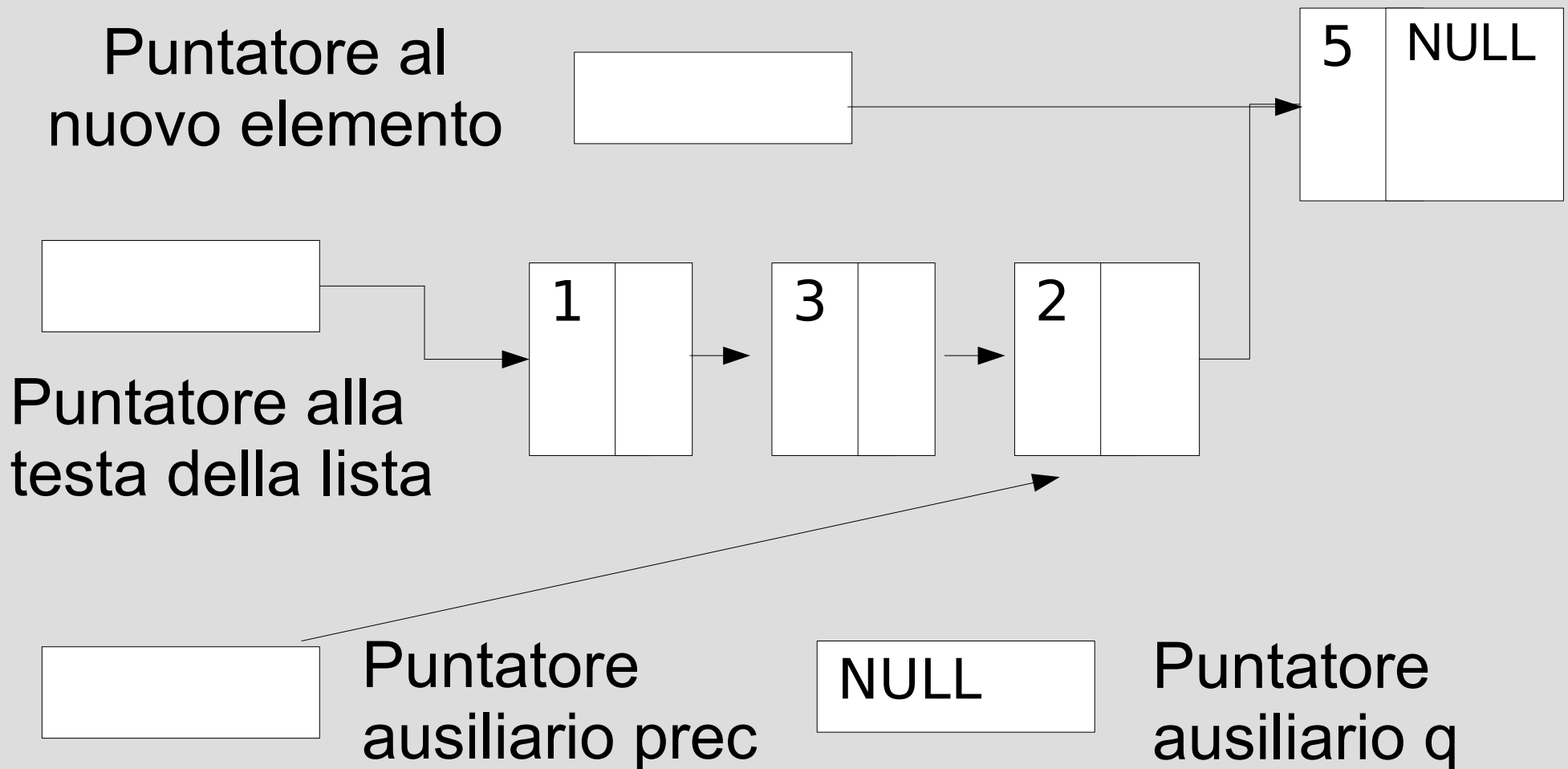
# Inserimento in coda

- Creazione del nuovo elemento



# Inserimento in coda

- Aggancio sfruttando **prec**



# Programma

- Realizzare anche *inserisci\_in\_coda* in *lista.cc*
  - Supporre che la funzione abbia tipo di ritorno ***void*** e che le venga **passato in ingresso il puntatore alla testa della lista**
  - Scegliere ed implementare correttamente **una delle due precedenti soluzioni** per la ricerca dell'ultimo elemento

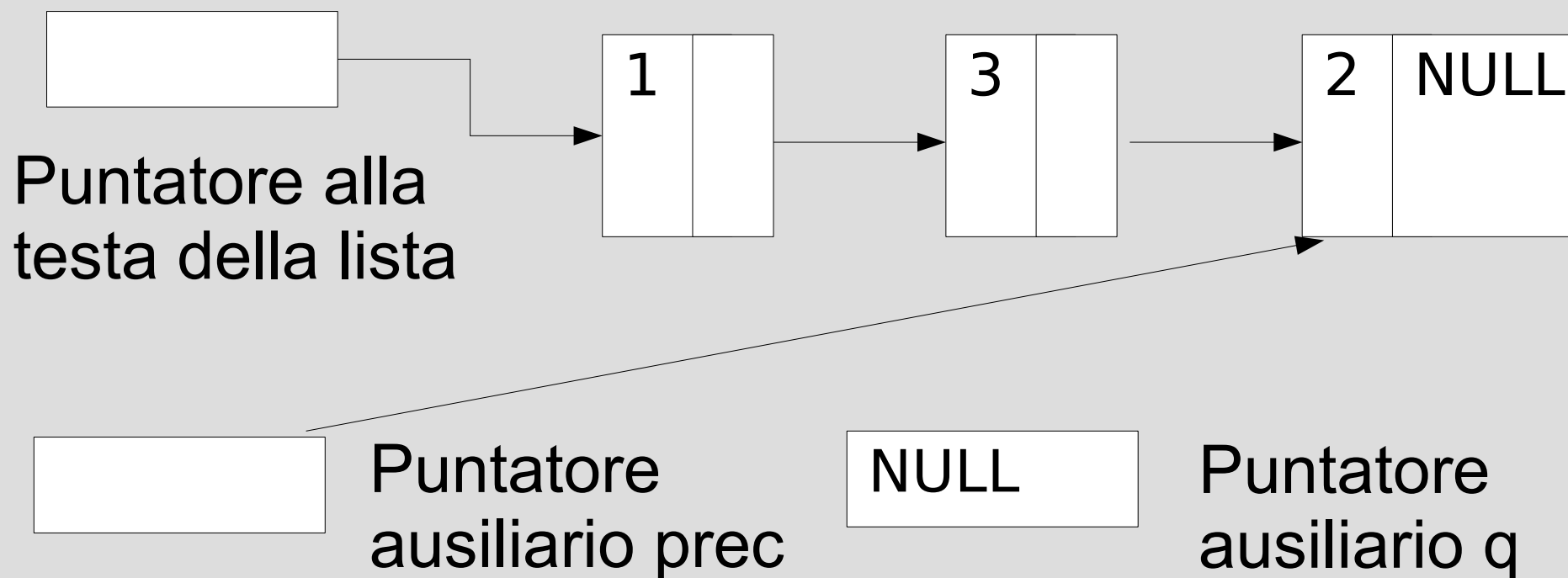
# Estrazione dalla coda

## *Algoritmo*

- 1) Controllo che la lista non sia vuota
  - Necessario, non è possibile l'estrazione
- 2) Ricerca della coda (memorizzazione degli indirizzi necessari per effettuare l'eliminazione)
- 3) Sgancio dell'ultimo elemento
  - Se la lista conteneva un solo elemento: aggiornamento del puntatore alla testa
  - Se la lista conteneva più di un elemento: aggiornamento del campo puntatore del (precedente) penultimo elemento
- 4) Deallocazione dell'elemento estratto

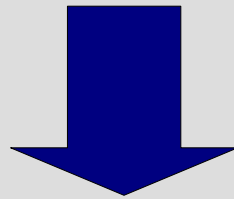
# Soluzioni

- Riprendiamo la coppia di puntatori usata per l'inserimento in coda
- Al termine della ricerca la situazione era:



# Differenza nelle condizioni

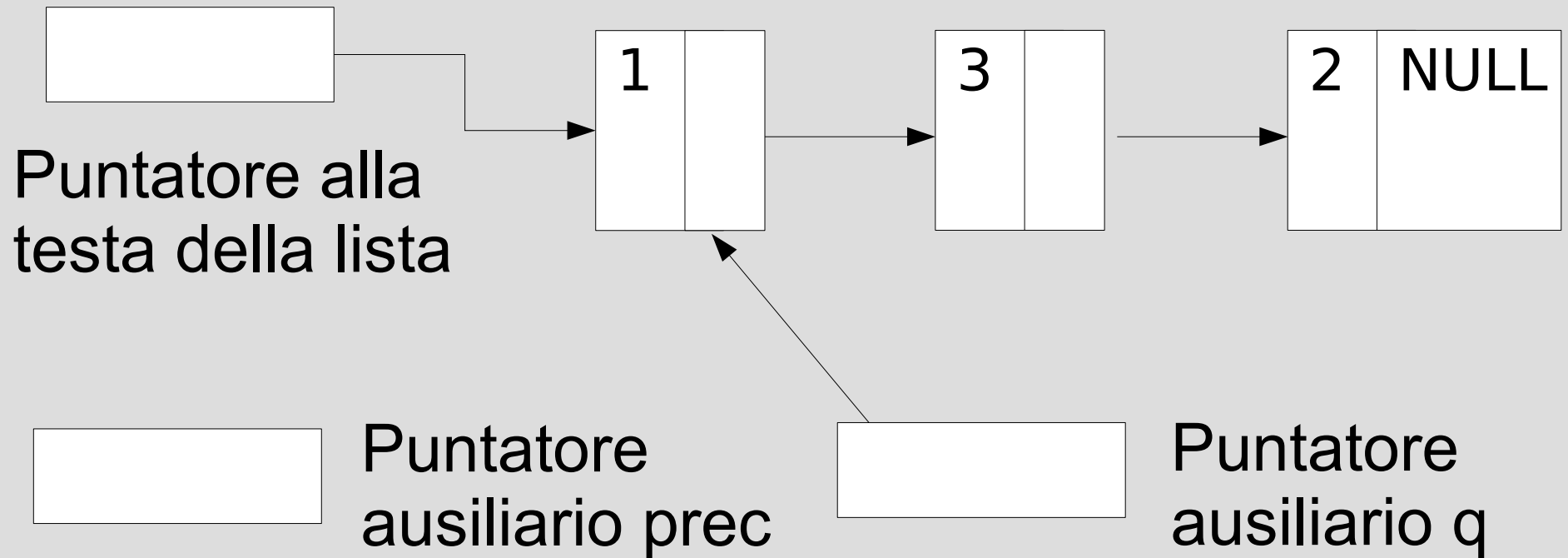
- Ora però serve che **prec** punti al penultimo elemento → va fermato prima
- Quale condizione usare per fermare **prec** al penultimo elemento?



Si potrebbe fermare **q** sull'ultimo elemento anziché farlo uscire dalla lista → ***fermare il ciclo quando q punta all'ultimo elemento senza far avanzare prec***

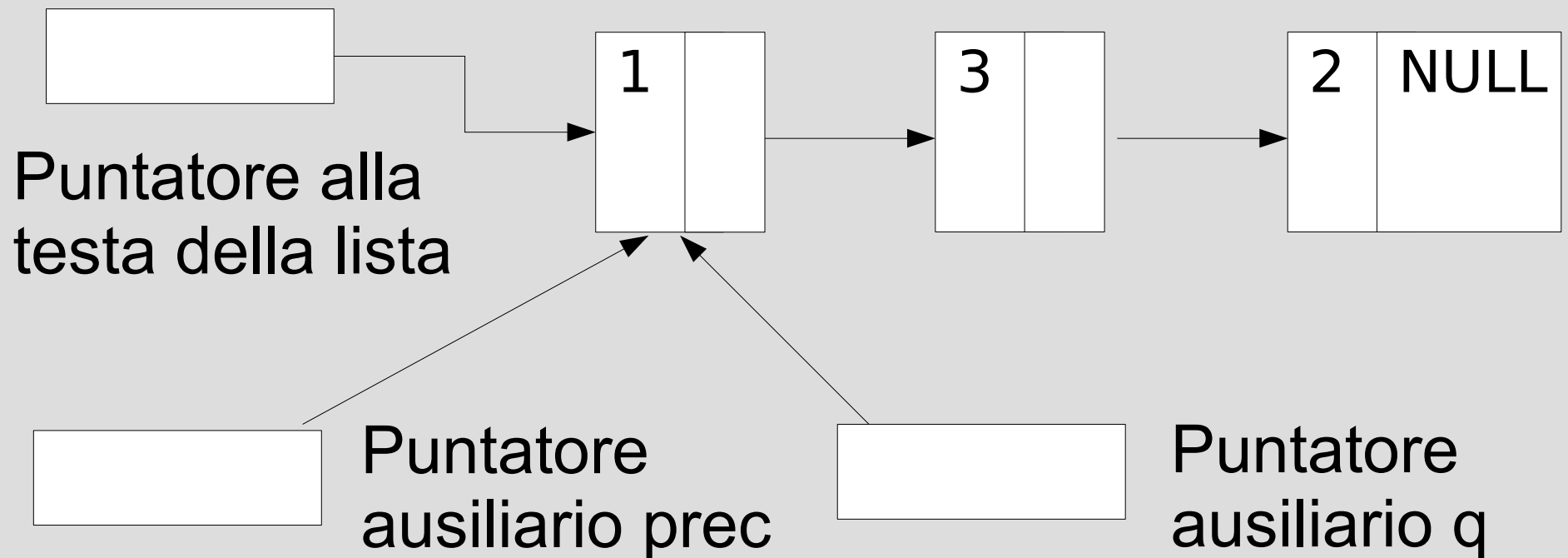
# Estrazione dalla coda

- Situazione iniziale



# Estrazione dalla coda

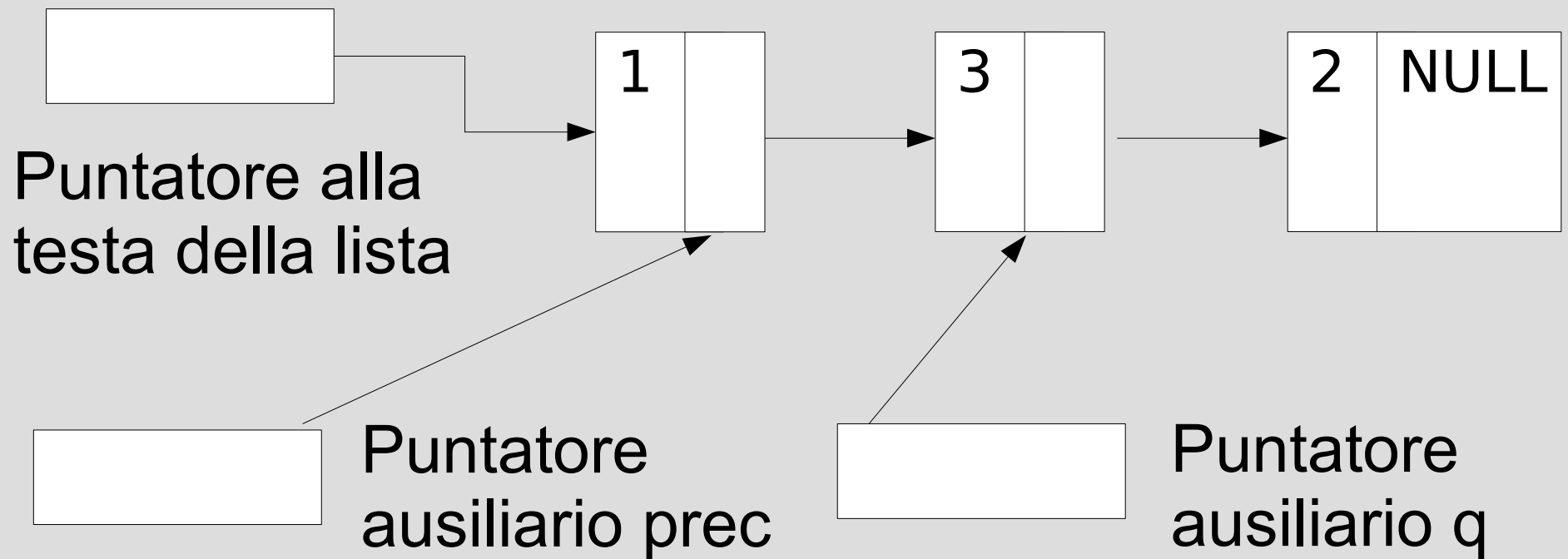
- Se non si è sull'ultimo elemento, il puntatore **prec** prende il valore di **q**...





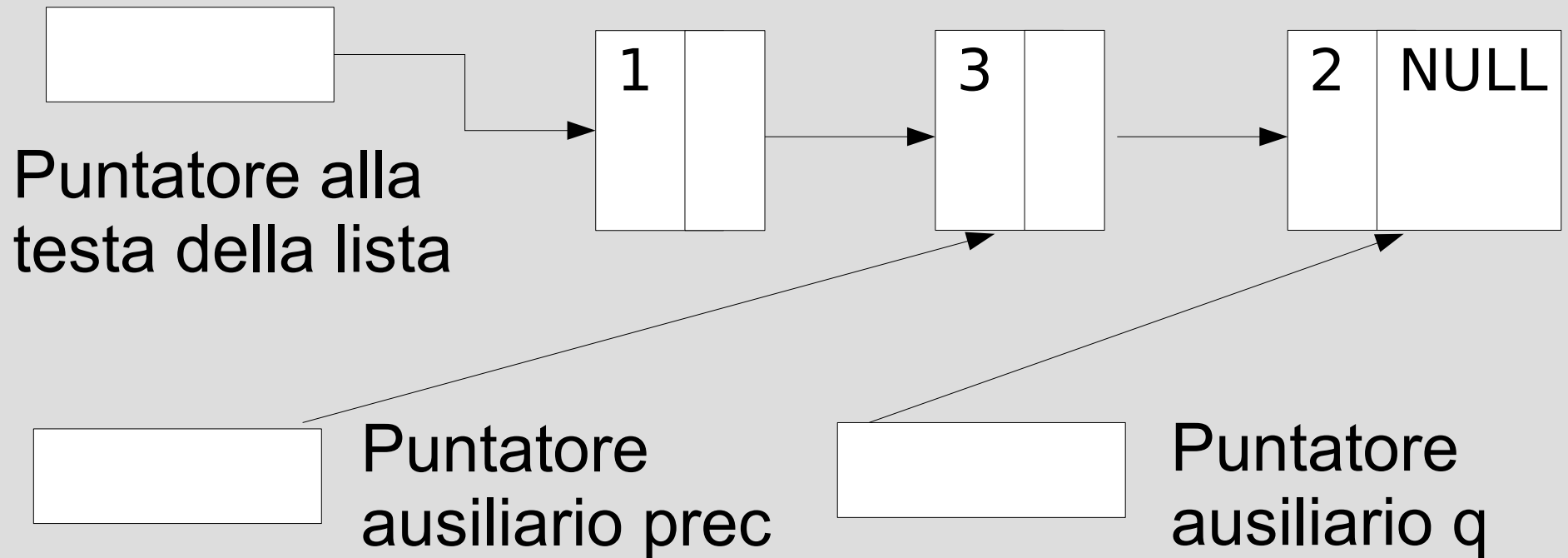
# Estrazione dalla coda

- ... e **q** avanza



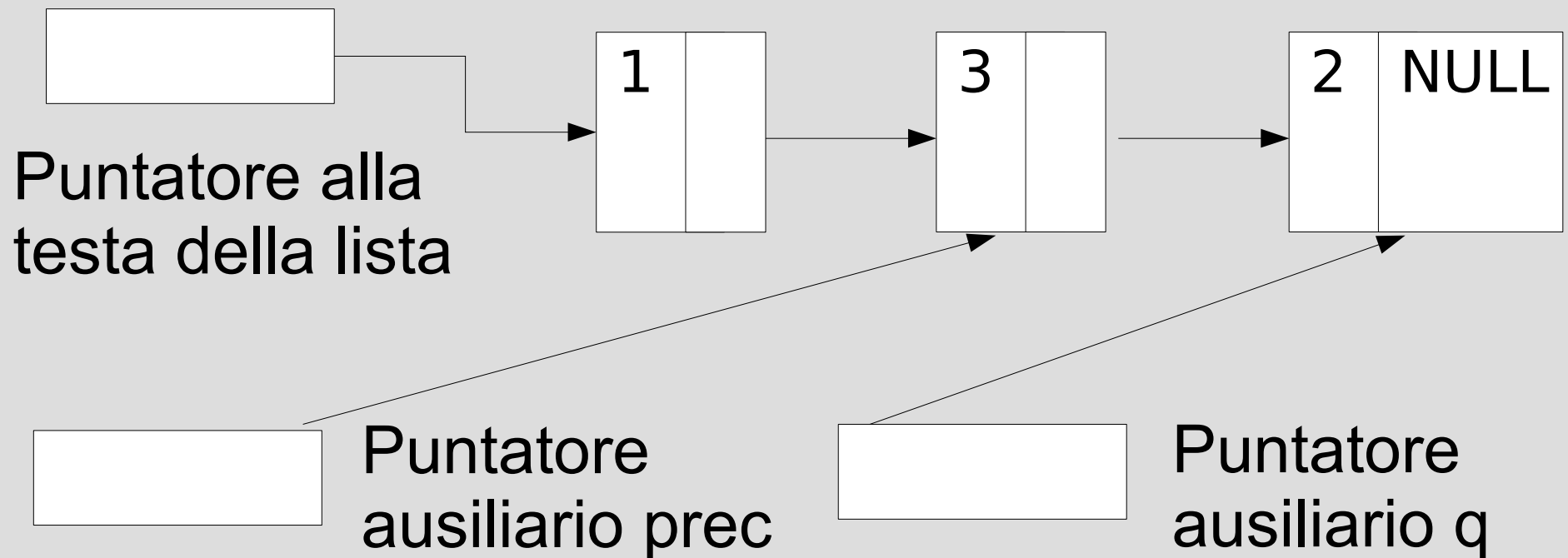
# Estrazione dalla coda

- Ulteriore passo di avanzamento



# Estrazione dalla coda

- A questo punto  $q$  è sull'ultimo elemento
- ***Si esce dal ciclo senza aggiornare prec***



# Condizione per l'uscita dal ciclo

- Qual è la **condizione da controllare** per fermarsi quando q è sull'ultimo elemento?

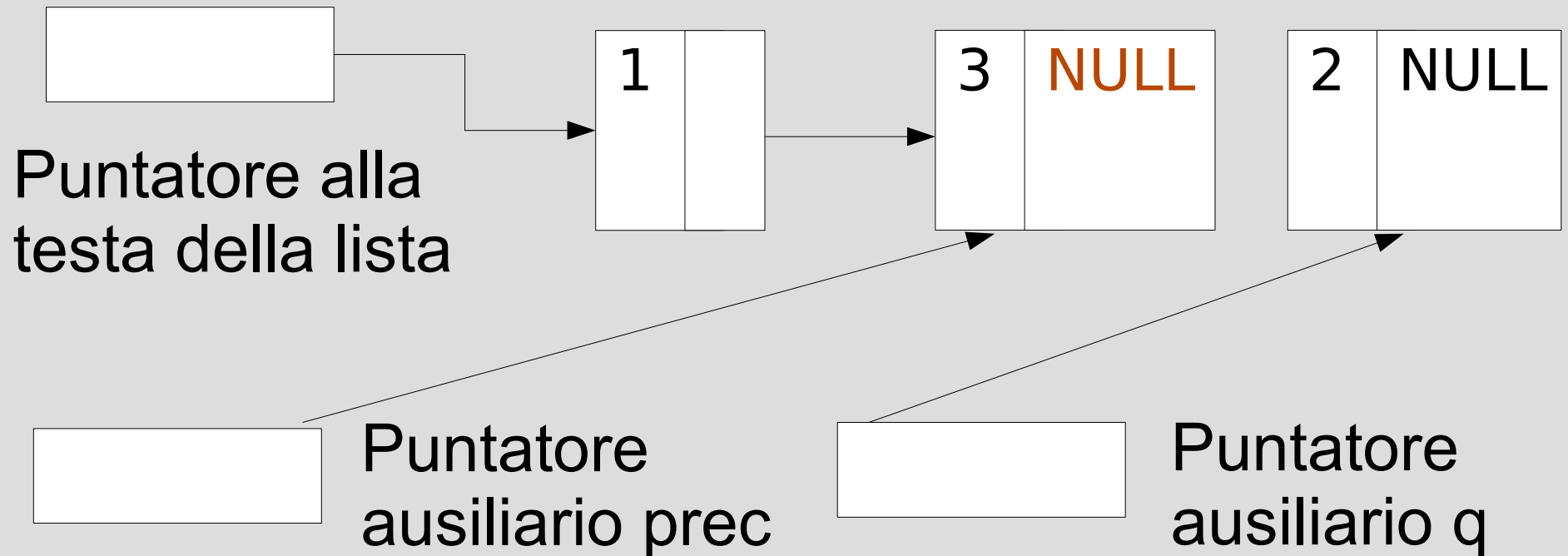
**q->pun != 0**

Possono sorgere problemi nell'accesso alla memoria nel valutare questa condizione?

***No, perché il caso di lista vuota lo abbiamo già controllato nel punto 1 dell'algoritmo - in questo caso è un controllo necessario***

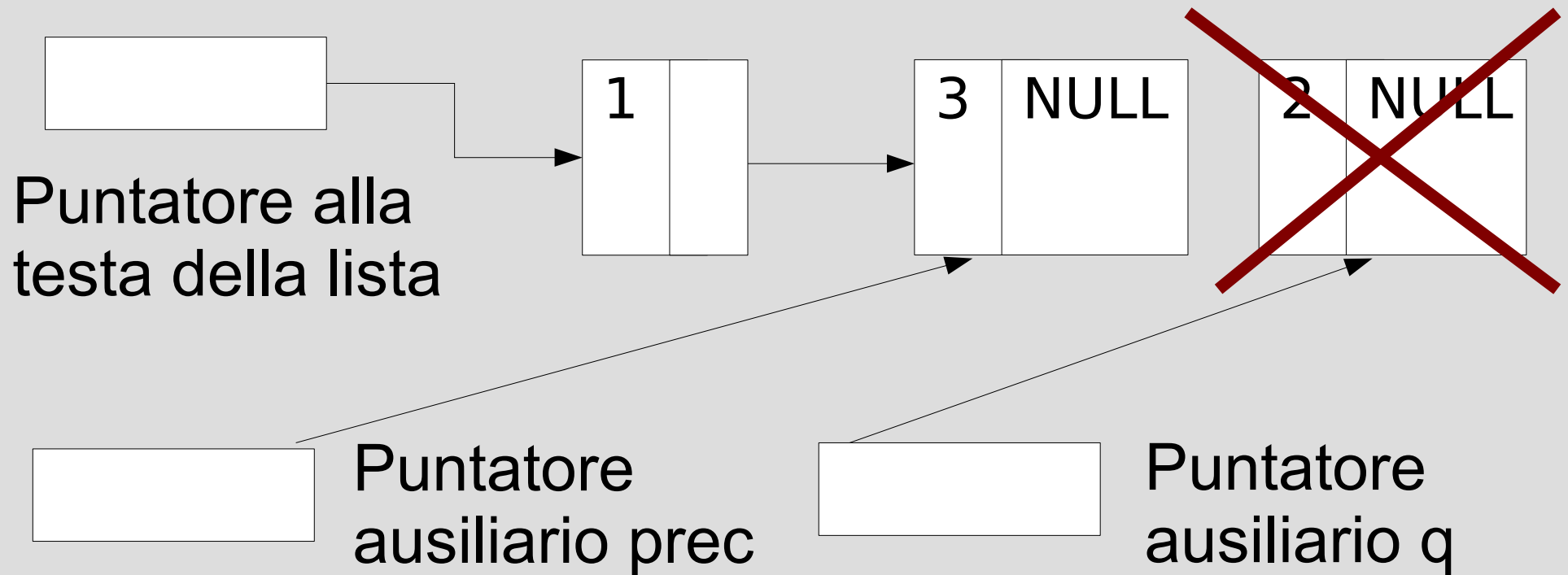
# Estrazione dalla coda

- **Sgancio** dell'ultimo elemento



# Estrazione dalla coda

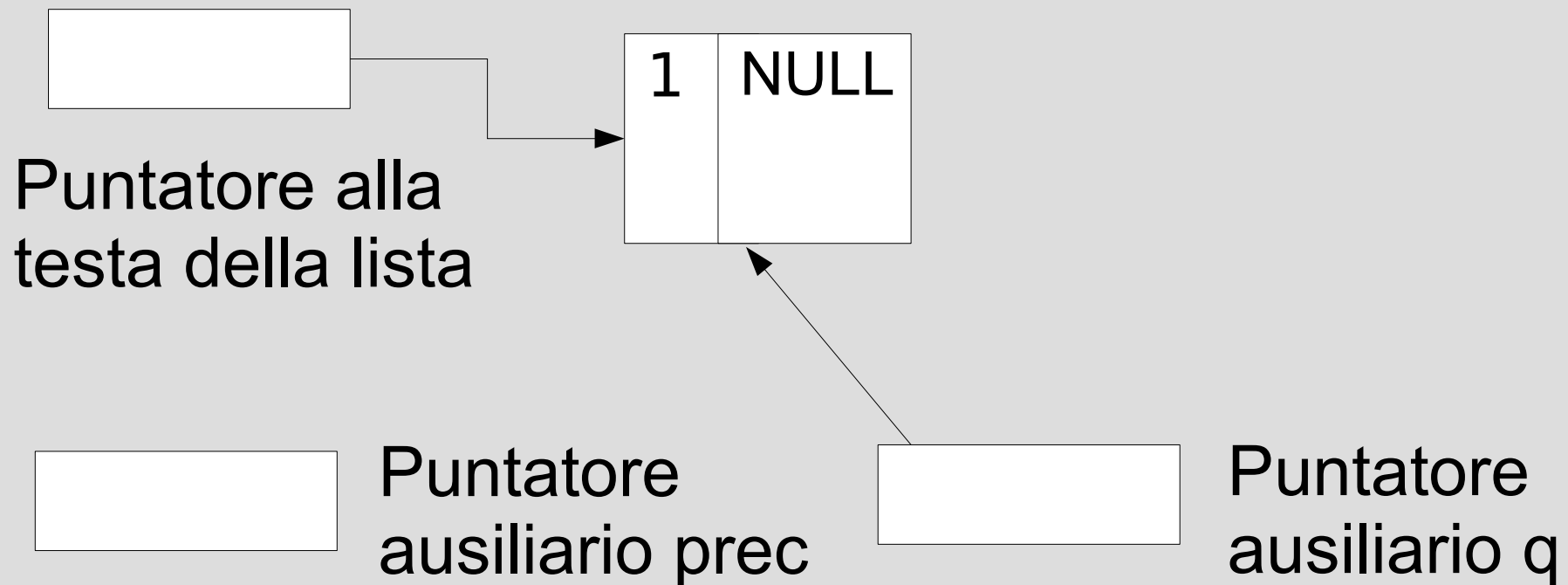
- Deallocazione dell'elemento



# Lista con un solo elemento

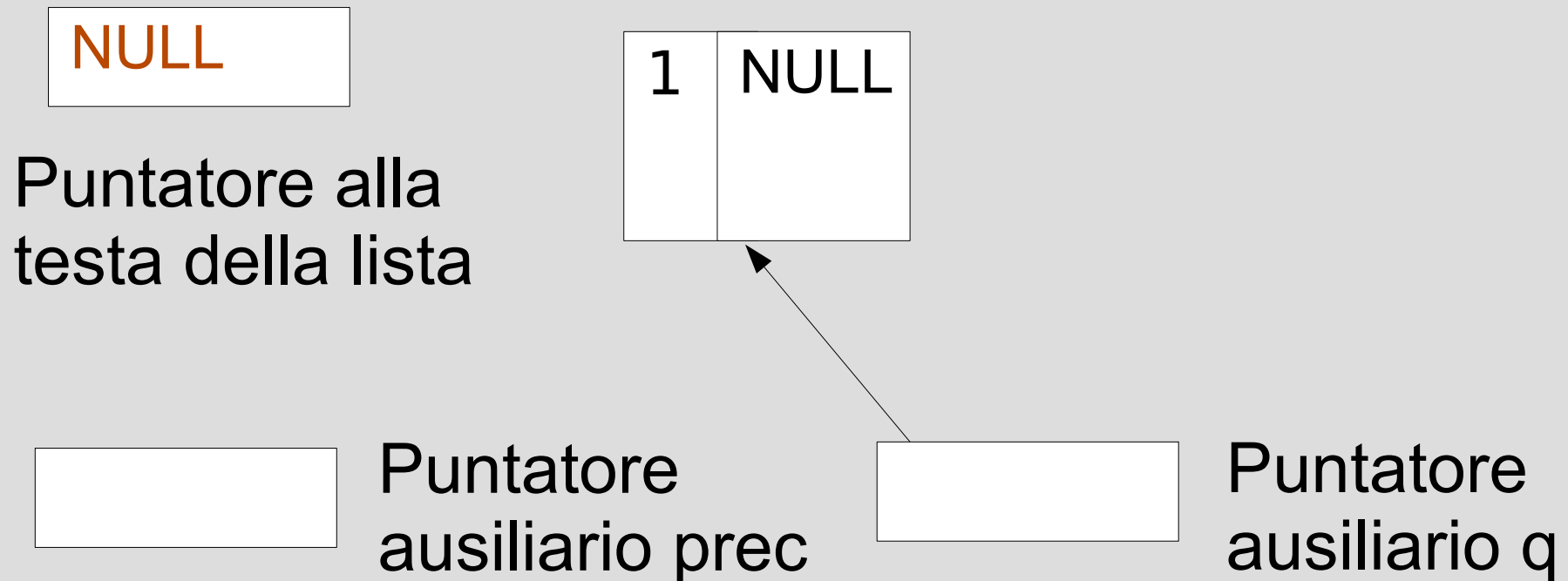
## Caso di lista con un elemento solo

- q punta all'unico elemento... e lì rimane
- Cosa fare?



# Lista con un solo elemento

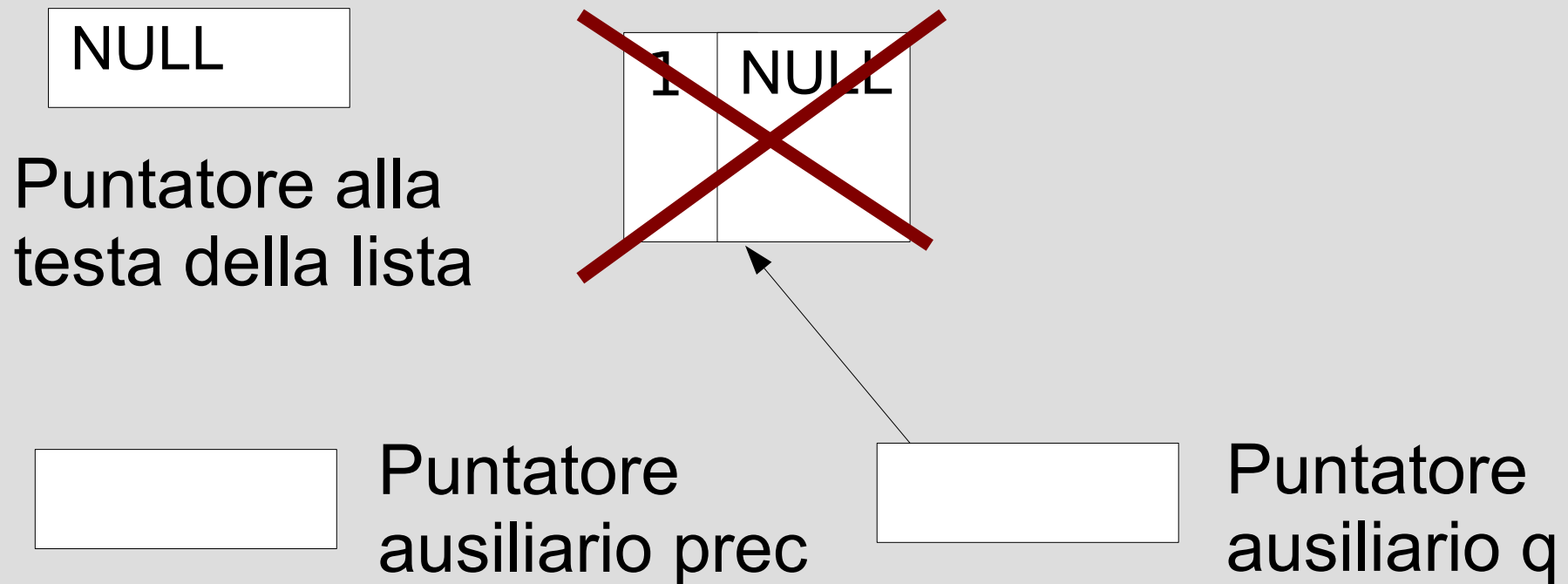
- In base alla **condizione precedente**, si aggiorna il puntatore alla testa della lista
- **Sgancio**





# Lista con un solo elemento

- Deallocazione dell'elemento



**Si tratta di un caso che devo trattare separatamente?**

# Programma

- Realizzare anche la funzione *estrai\_dal\_fondo* in *lista.cc*
  - Supporre che la funzione abbia tipo di ritorno **bool** e ritorni **true** solo se l'estrazione ha successo (lista non vuota)
  - La funzione deve ritornare anche il valore del campo informazione dell'elemento estratto
  - Alla funzione viene passato il puntatore alla testa della lista