STRUTTURE DATI
LISTE CONCATENATE

INFORMATICA

COSA È UNA STRUTTURA DATI

- Fino ad ora abbiamo visto algoritmi che lavorano su dei dati (e.g., un elenco di valori)
- Ma memorizzavamo i dati in una singola struttura dati: l'array
- Quali sono i vantaggi e gli svantaggi degli array?
- Possiamo organizzare i dati in modo diverso in base alle operazioni che dobbiamo eseguire?

IT IS BETTER TO HAVE 100 FUNCTIONS OPERATE ON ONE DATA STRUCTURE THAN 10 FUNCTIONS ON 10 DATA STRUCTURES.

Alan Perlis

L'ARRAY

- Possiamo pensare a raccogliere in una tabella le operazioni che di solito facciamo con gli array e vedere quanto tempo ci mettiamo
- Esempi di operazioni:
  - Accedere ad un elemento dato l'indice
  - Cercare se un elemento
  - Rimuovere o aggiungere un elemento in una posizione arbitraria

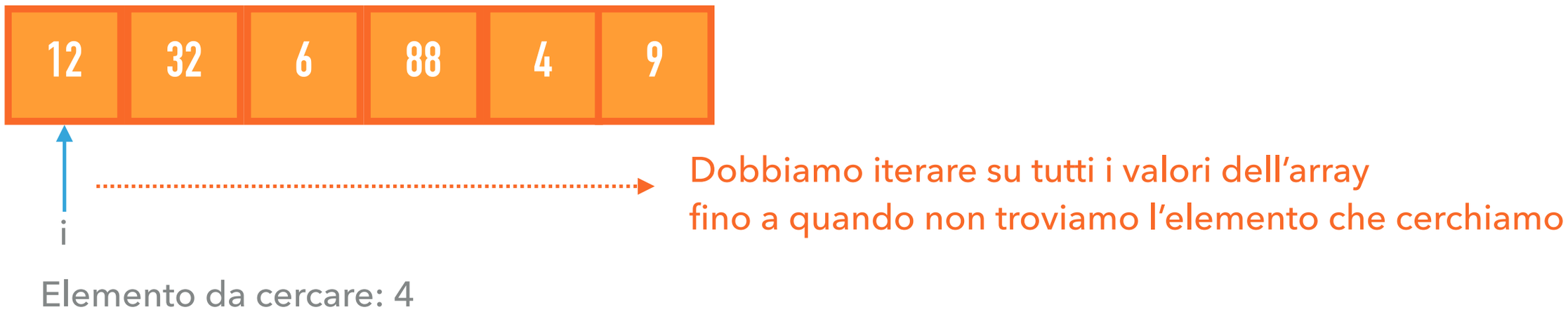
L'ARRAY (NON ORDINATO)

Espresso come funzione del numero di elementi contenuti

Operazione	Tempo
Accedere ad un elemento	$O(1)$
Ricerca	
Inserimento	
Rimozione	

Come facciamo a verificare se un elemento esiste nell'array?

L'ARRAY (NON ORDINATO): LA RICERCA



Nel caso peggiore dobbiamo scorrere tutto l'array.
Dato che l'array contiene n elementi, il tempo richiesto per la ricerca è lineare in n: $O(n)$

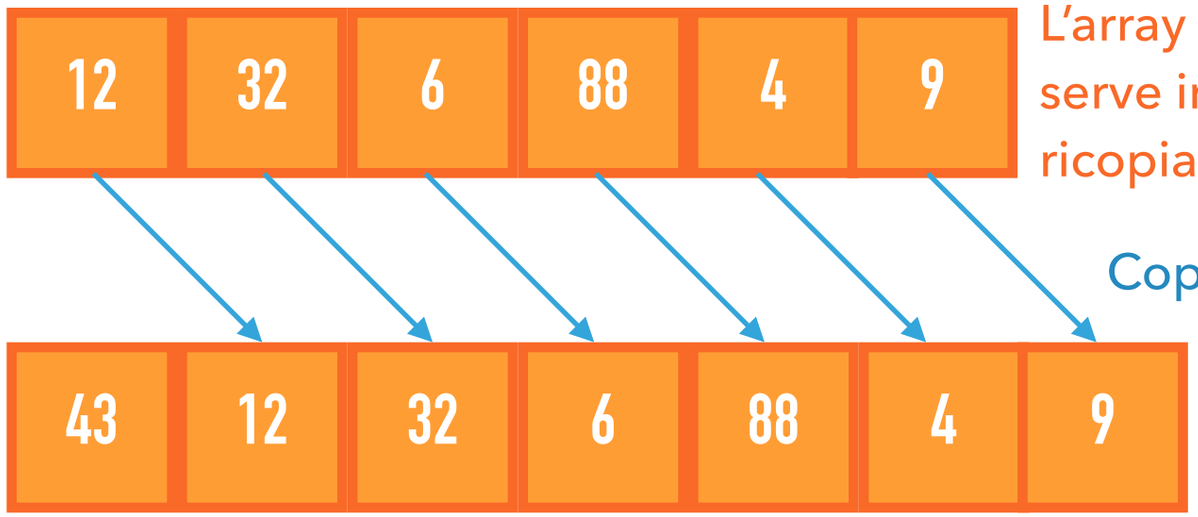
L'ARRAY (NON ORDINATO)

Cosa succede quando dobbiamo inserire un elemento (senza tener conto dell'ordine)?

Operazione	Tempo
Accedere ad un elemento	$O(1)$
Ricerca	$O(n)$
Inserimento	
Rimozione	

L'ARRAY (NON ORDINATO): INSERIMENTO

Elemento da inserire: 43



L'array ha un numero fissato di elementi, se serve inserire un nuovo elemento dobbiamo ricopiare tutto!

Copia di tutti gli elementi

Dato che dobbiamo ricopiare tutti gli elementi, aggiungere un elemento richiede tempo $O(n)$, a meno di non conoscere il numero di elementi che ci servono e preallocare tutto lo spazio necessario.

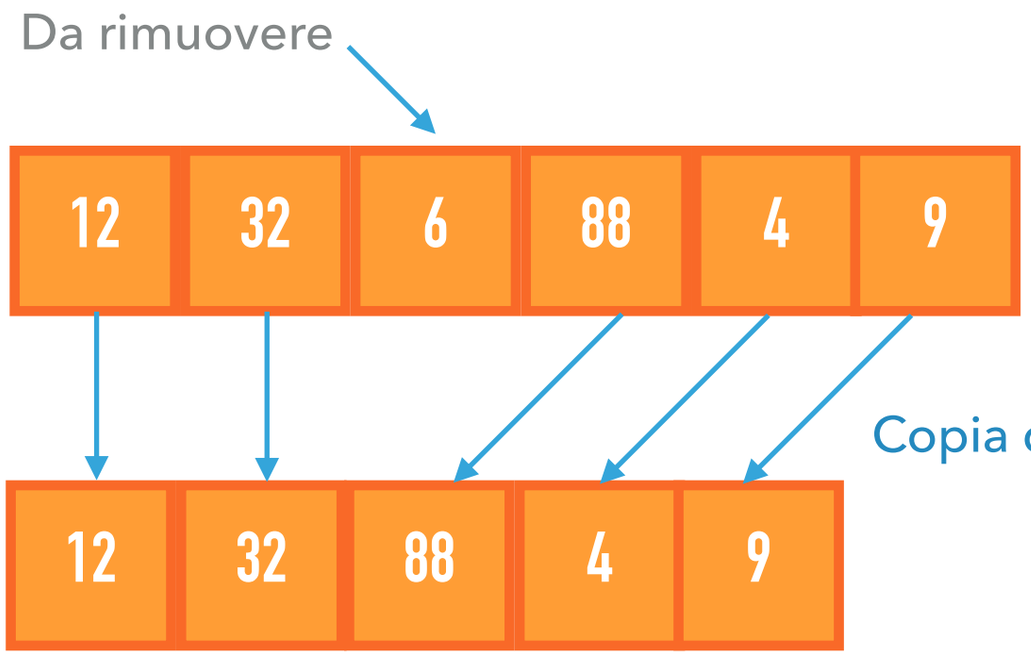
Gli array in python sono implementati in modo da richiedere tempo costante "nella maggior parte dei casi", ma il caso peggiore per una singola operazione rimane lineare

L'ARRAY (NON ORDINATO)

Cosa succede quando dobbiamo rimuovere un elemento (dato l'indice)?

Operazione	Tempo
Accedere ad un elemento	$O(1)$
Ricerca	$O(n)$
Inserimento	$O(n)$
Rimozione	

L'ARRAY (NON ORDINATO): INSERIMENTO



Anche questa volta dobbiamo copiare tutti gli elementi e spostare quelli che seguono l'elemento da rimuovere

Copia di tutti gli elementi tranne quello da rimuovere

Dato che dobbiamo ricopiare tutti gli elementi tranne uno, rimuovere un elemento richiede tempo $O(n)$

L'ARRAY (NON ORDINATO)

Adesso abbiamo una visione più completa di quanto costano le diverse operazioni su un array non ordinato

Cambiando struttura dati (il "contenitore" dei nostri dati) possiamo migliorare alcuni aspetti (magari sacrificandone altri)?

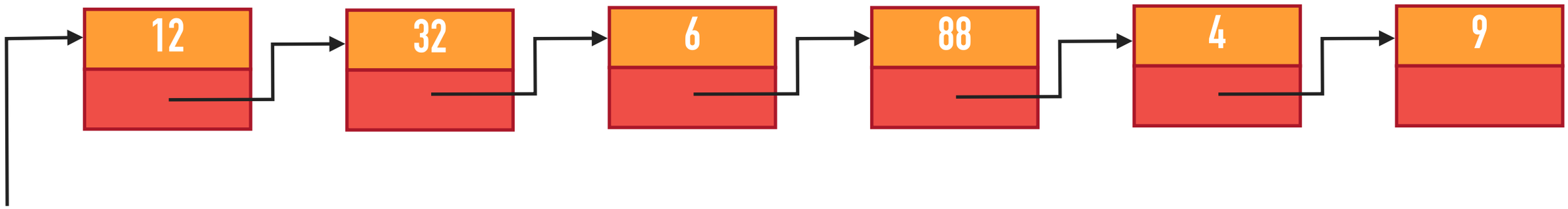
Operazione	Tempo
Accedere ad un elemento	$O(1)$
Ricerca	$O(n)$
Inserimento	$O(n)$
Rimozione	$O(n)$

LA LISTA CONCATENATA

- Una delle strutture dati più semplici è la **lista concatenata singola**
- Possiamo pensare ad una lista concatenata come ad una serie di nodi ognuno dei quali indica il successivo nell'elenco
- Si sacrifica il tempo di accesso ad un elemento per velocizzare le operazioni di inserimento e rimozione
- Al contrario di un array la struttura può continuare a crescere senza necessitare di essere copiata

LA LISTA CONCATENATA

Graficamente possiamo rappresentare una lista concatenata in questo modo:



Testa della lista

Ma cosa sono quelle frecce? Perché la lista concatenata è diversa da un array?

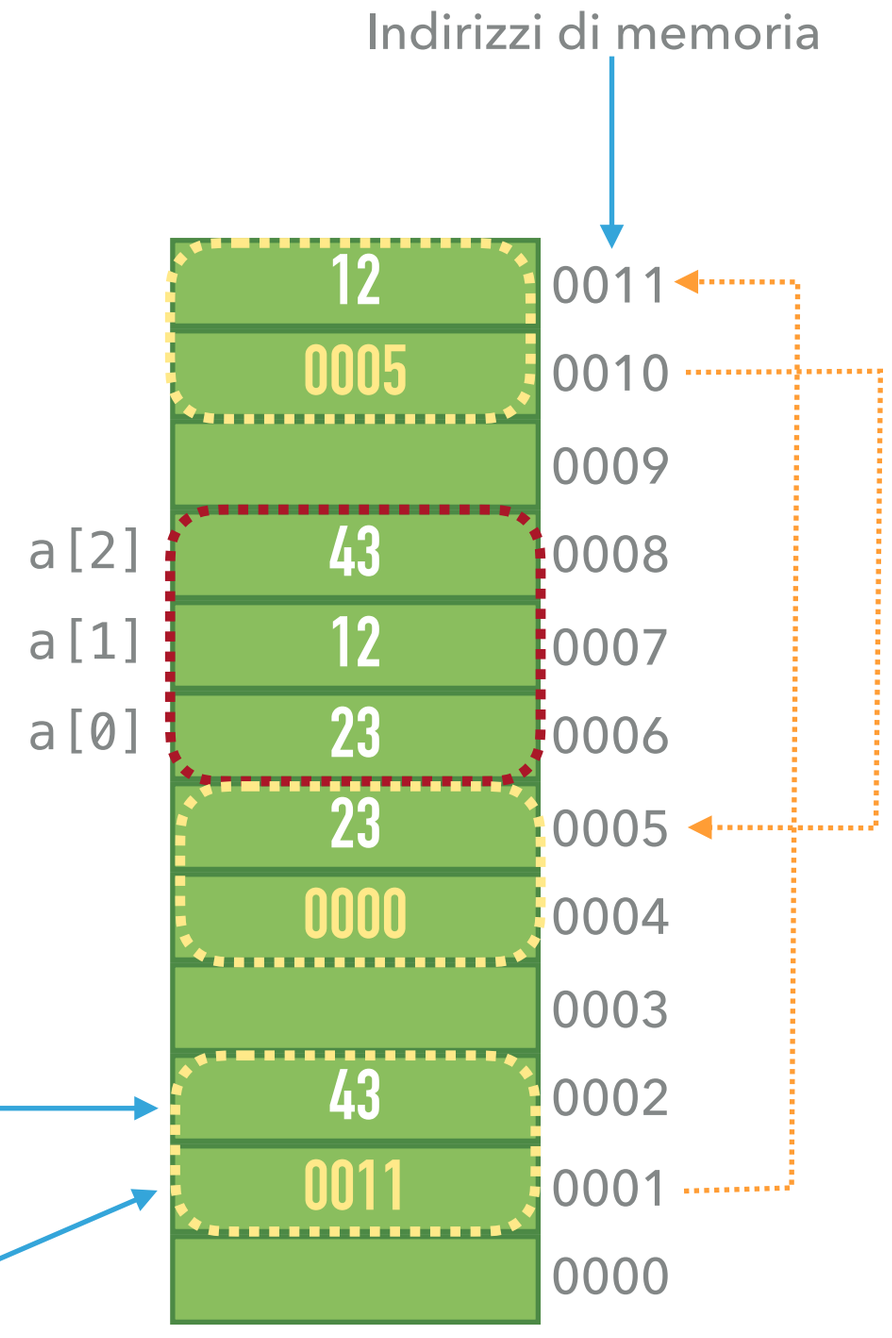
Per questo dobbiamo tornare a vedere come sono disposti in memoria gli array e cosa sono i riferimenti

DISPOSIZIONE IN MEMORIA

Array
Elementi allocati in locazioni di memoria consecutive
e.g., array di tre elementi

Nodo della lista concatenata
Contiene il valore memorizzato
e l'indirizzo di memoria del nodo successivo

Ok, ma come rappresentiamo
questi indirizzi in Python?



CLASSI E REFERENCES

```
class NodoLista:

    def __init__(self, value, next):
        self.value = value
        self.next = next
```

Valore salvato nel nodo della lista

Qui salviamo un **riferimento** ad un oggetto che sarà il nodo successivo nella lista.

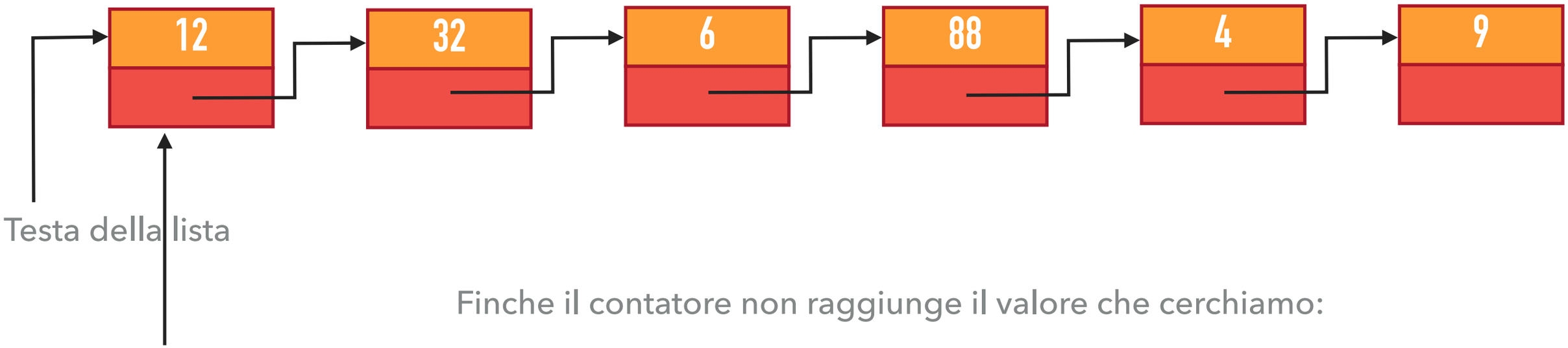
Usiamo **None** per indicare che il riferimento non porta da nessuna parte

LA LISTA CONCATENATA: OPERAZIONI

- Vediamo ora come effettuare alcune operazioni sulle liste concatenate singole:
  - Accedere ad un elemento
  - Ricerca all'interno della lista
  - Inserimento in testa alla lista
  - Inserimento in una posizione arbitraria
  - Rimozione di un elemento

LA LISTA CONCATENATA: ACCESSO AD UN ELEMENTO

Supponiamo di voler accedere all'elemento di posizione 3



Finche il contatore non raggiunge il valore che cerchiamo:

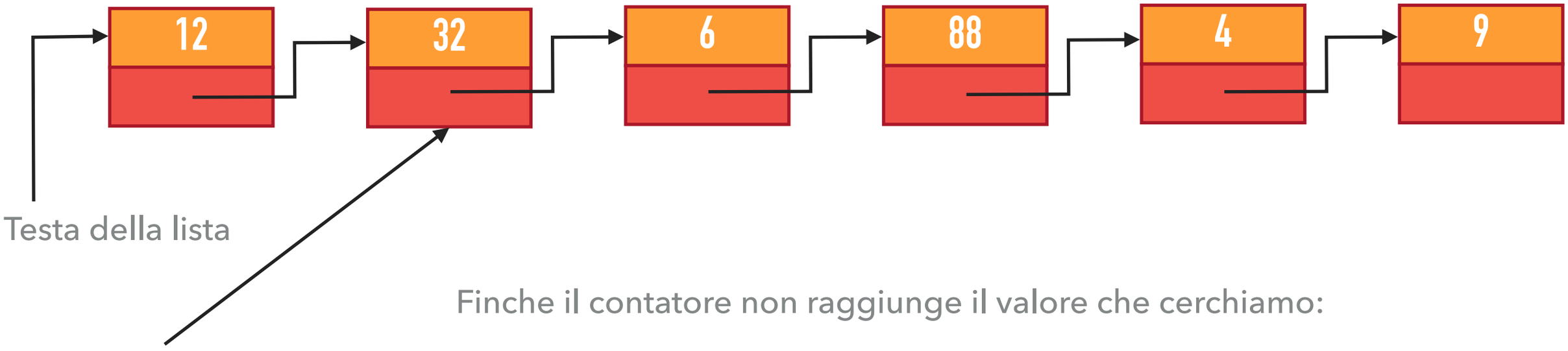
Passiamo an nodo successivo

Incrementiamo il contatore

PERCHÉ NON POSSIAMO ACCEDERE DIRETTAMENTE ALL'I-ESIMO ELEMENTO?

LA LISTA CONCATENATA: ACCESSO AD UN ELEMENTO

Supponiamo di voler accedere all'elemento di posizione 3



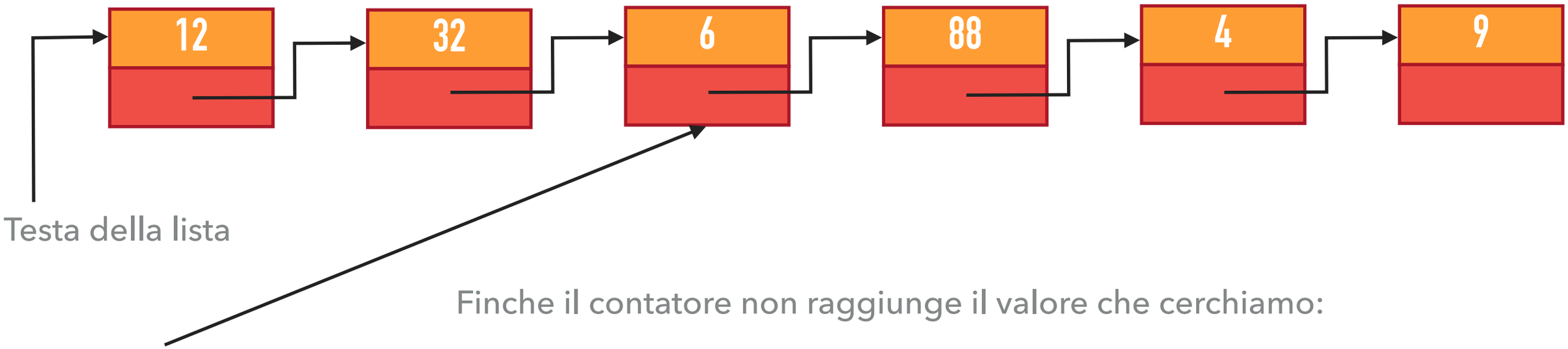
Finche il contatore non raggiunge il valore che cerchiamo:

Passiamo an nodo successivo

Incrementiamo il contatore

LA LISTA CONCATENATA: ACCESSO AD UN ELEMENTO

Supponiamo di voler accedere all'elemento di posizione 3



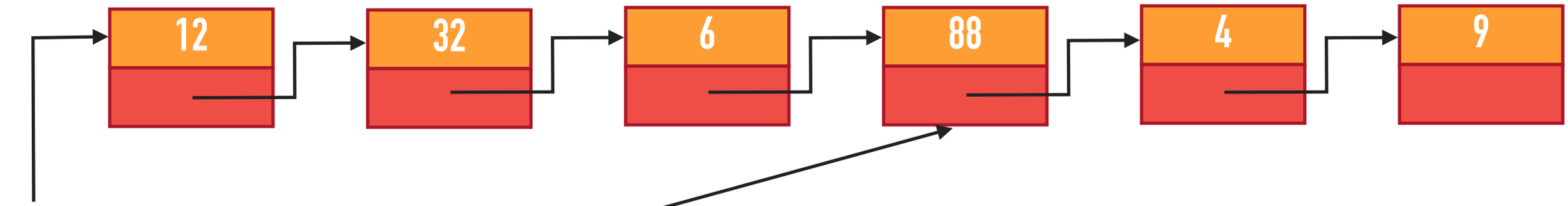
Finche il contatore non raggiunge il valore che cerchiamo:

Passiamo an nodo successivo

Incrementiamo il contatore

LA LISTA CONCATENATA: ACCESSO AD UN ELEMENTO

Supponiamo di voler accedere all'elemento di posizione 3



Nodo corrente

i = 3

Finche il contatore non raggiunge il valore che cerchiamo:

Passiamo an nodo successivo

Incrementiamo il contatore

LA LISTA CONCATENATA: ACCESSO AD UN ELEMENTO

```
Parametri: testa (riferimento al primo elemento della lista), i (indice)
nodo_corrente = testa
n_elementi = 0
while nodo_corrente ≠ None and n_elementi < i
    nodo_corrente = nodo_corrente.next
    n_elementi = n_elementi + 1
return nodo_corrente
```

Dobbiamo fare questo controllo perché potremmo finire la lista se questa contiene meno di *i+1* elementi

Ritorneremo un riferimento al nodo nella posizione i-esima oppure None nel caso la lista abbia meno elementi

LA LISTA CONCATENATA: ACCESSO AD UN ELEMENTO

- Quale è la complessità di trovare accedere ad un elemento in una data posizione?
- Dobbiamo scorrere la lista fino ad arrivare all'elemento in posizione i-esima. Passare da un elemento successivo costa solo un numero costante di operazioni
- Nel caso peggiore dobbiamo scorrere tutta la lista di n elementi. Quindi il tempo necessario è $O(n)$
- Meno efficiente dell'accesso in un array!

LA LISTA CONCATENATA: RICERCA DI UN ELEMENTO

- La ricerca di un elemento è molto simile all'accesso dato un indice
- Invece di fermarci quando raggiungiamo il numero di elementi desiderato ci fermiamo quando abbiamo trovato il valore che cerchiamo
- La complessità rimane uguale, lineare nel numero di elementi nella lista: $O(n)$

LA LISTA CONCATENATA: RICERCA DI UN ELEMENTO

```
Parametri: testa (riferimento al primo elemento della lista), x (valore)
nodo_corrente = testa
while nodo_corrente ≠ None and nodo_corrente.value ≠ x
    nodo_corrente = nodo_corrente.next
return nodo_corrente
```

Questo controllo va mantenuto perché il valore cercato potrebbe non essere all'interno della lista

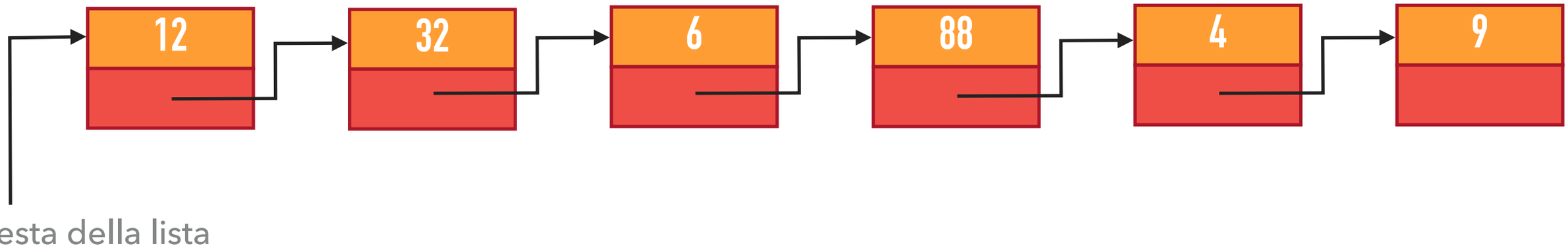
Ritorneremo un riferimento al nodo nella posizione i-esima oppure None nel caso la lista non contenga il valore cercato

LA LISTA CONCATENATA: INSERIMENTO IN TESTA

- Negli array l'inserimento – in particolare l'inserimento all'inizio dell'array – richiede una copia
- Nelle liste è più semplice:
  - Si crea un nuovo nodo col valore da inserire
  - Tutta la lista già esistente si "aggancia" come nodo successivo a quello nuovo

LA LISTA CONCATENATA: INSERIMENTO IN TESTA

Supponiamo di voler aggiungere in testa un nodo avente valore 7



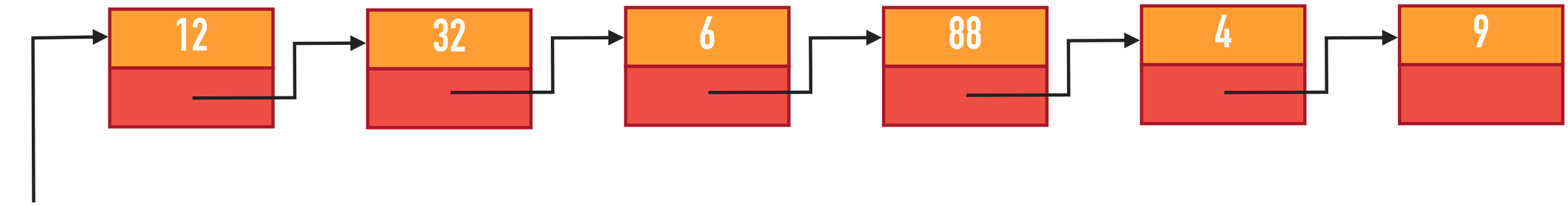
Creare il nuovo nodo

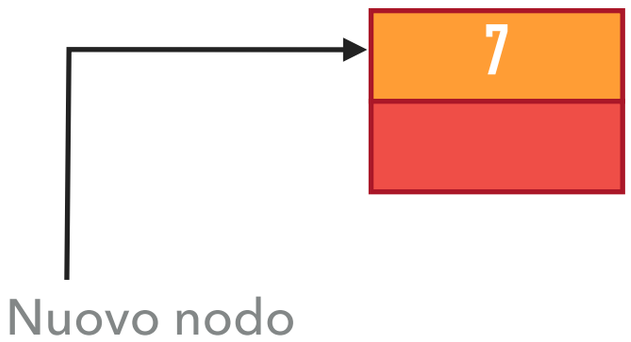
Agganciare il resto della lista al nuovo nodo

Aggiornare la testa della lista

LA LISTA CONCATENATA: INSERIMENTO IN TESTA

Supponiamo di voler aggiungere in testa un nodo avente valore 7





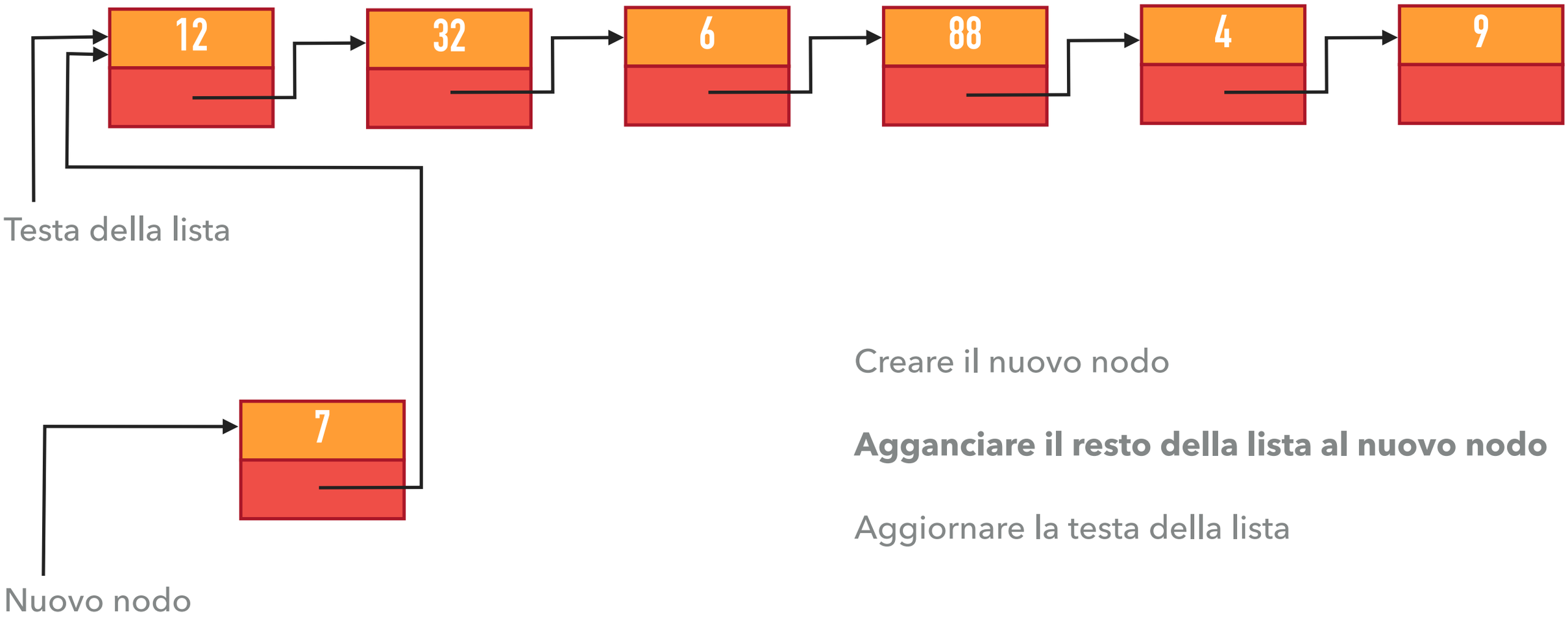
Creare il nuovo nodo

Agganciare il resto della lista al nuovo nodo

Aggiornare la testa della lista

LA LISTA CONCATENATA: INSERIMENTO IN TESTA

Supponiamo di voler aggiungere in testa un nodo avente valore 7



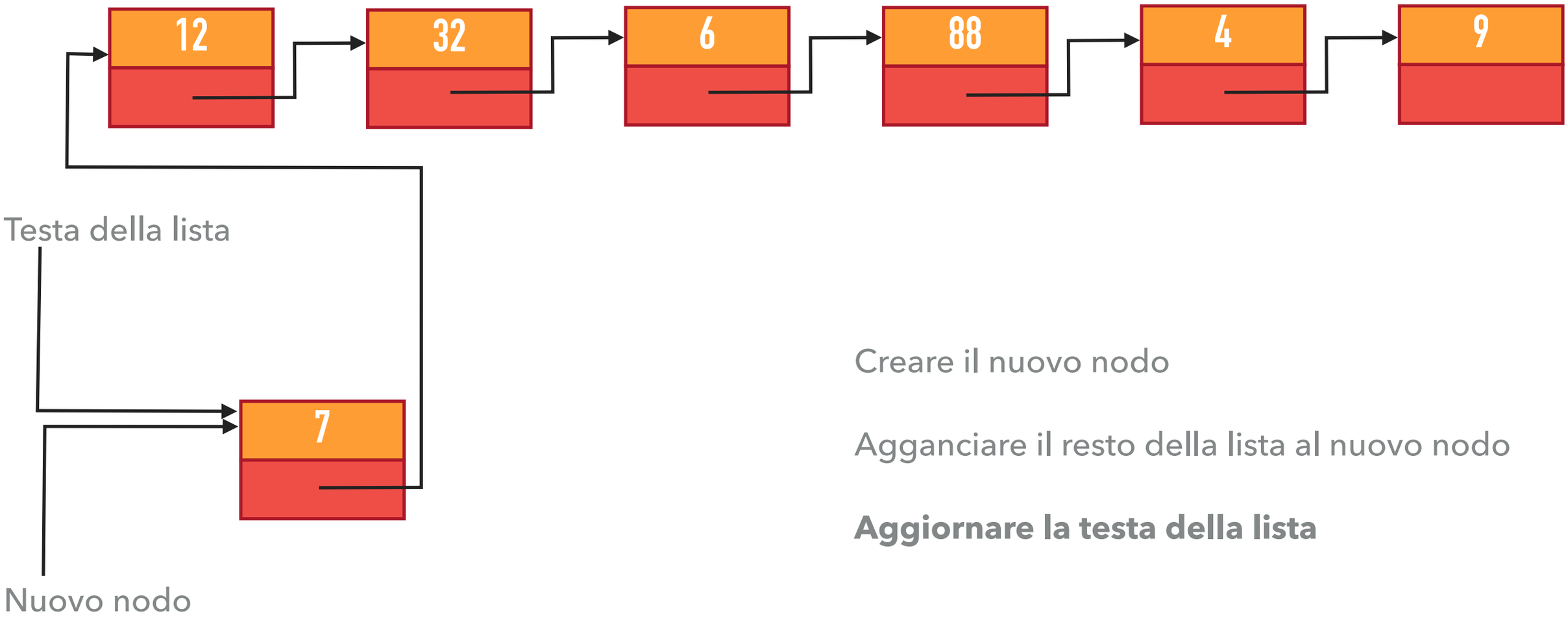
Creare il nuovo nodo

Agganciare il resto della lista al nuovo nodo

Aggiornare la testa della lista

LA LISTA CONCATENATA: INSERIMENTO IN TESTA

Supponiamo di voler aggiungere in testa un nodo avente valore 7



Creare il nuovo nodo

Agganciare il resto della lista al nuovo nodo

Aggiornare la testa della lista

LA LISTA CONCATENATA: INSERIMENTO IN TESTA

```
Parametri: testa (riferimento al primo elemento della lista), x (valore)
nuovo_nodo = crea un nuovo nodo
nuovo_nodo.value = x
nuovo_nodo.next = testa
return nuovo_nodo # il nuovo nodo è ritornato come nuova testa della lista
```

- Notate che non abbiamo nessun ciclo, eseguiamo un numero costante di istruzioni
- Il tempo di inserimento in testa è quindi costante: $\Theta(1)$

LA LISTA CONCATENATA: INSERIMENTO IN UNA POSIZIONE ARBITRARIA

- Inserire un nuovo nodo in una posizione arbitraria può essere diviso in due singole operazioni:
  - Individuare il nodo nella posizione precedente a quella dell'inserimento
  - Effettuare un inserimento in testa aggiornando il riferimento nel nodo precedente
- Possiamo anche darne una definizione ricorsiva

LA LISTA CONCATENATA: INSERIMENTO IN UNA POSIZIONE ARBITRARIA

```
Parametri: testa (riferimento al primo elemento della lista),
x (valore), i (posizione)
if i == 0
     return inserisci_in_testa(testa, x)
else
     testa.next = inserisci(testa.next, x, i - 1) # chiamata ricorsiva
     return testa
```

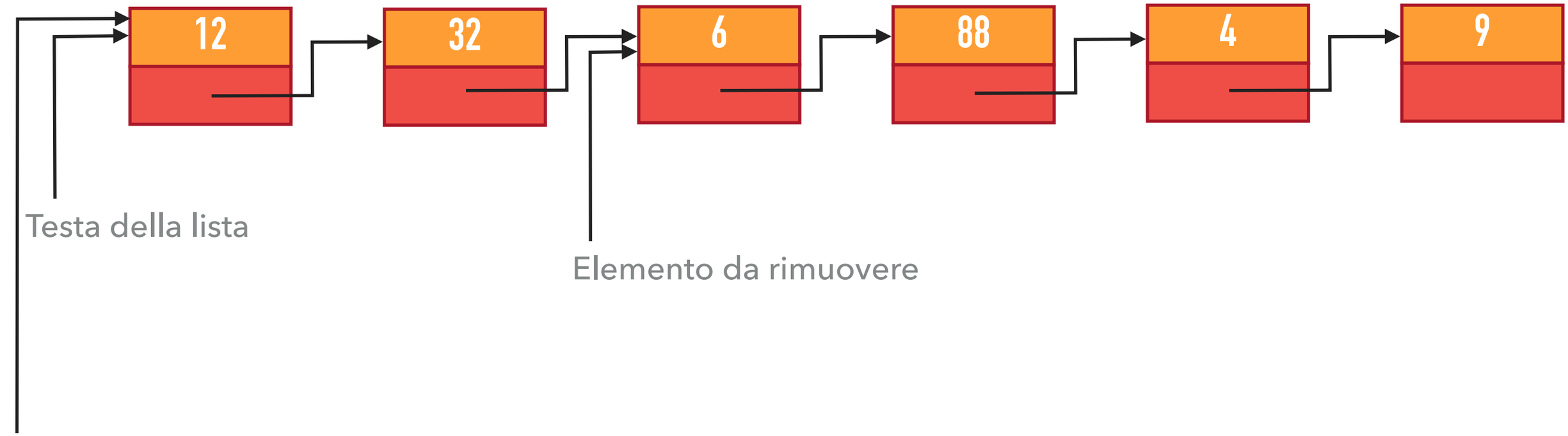
- In entrambe le implementazioni dobbiamo sempre scorrere la lista
- Il tempo richiesto è limitato superiormente dalla lunghezza della lista: $O(n)$

LA LISTA CONCATENATA: RIMOZIONE DI UN ELEMENTO

- Vogliamo rimuovere un elemento di cui ci viene dato il riferimento
- Dovremmo “aggiustare” il riferimento nel nodo che lo precede…
- …ma la lista concatenata singola permette solo di guardare al nodo successore

LA LISTA CONCATENATA: RIMOZIONE DI UN ELEMENTO

Supponiamo di voler aggiungere in testa un nodo avente valore 7

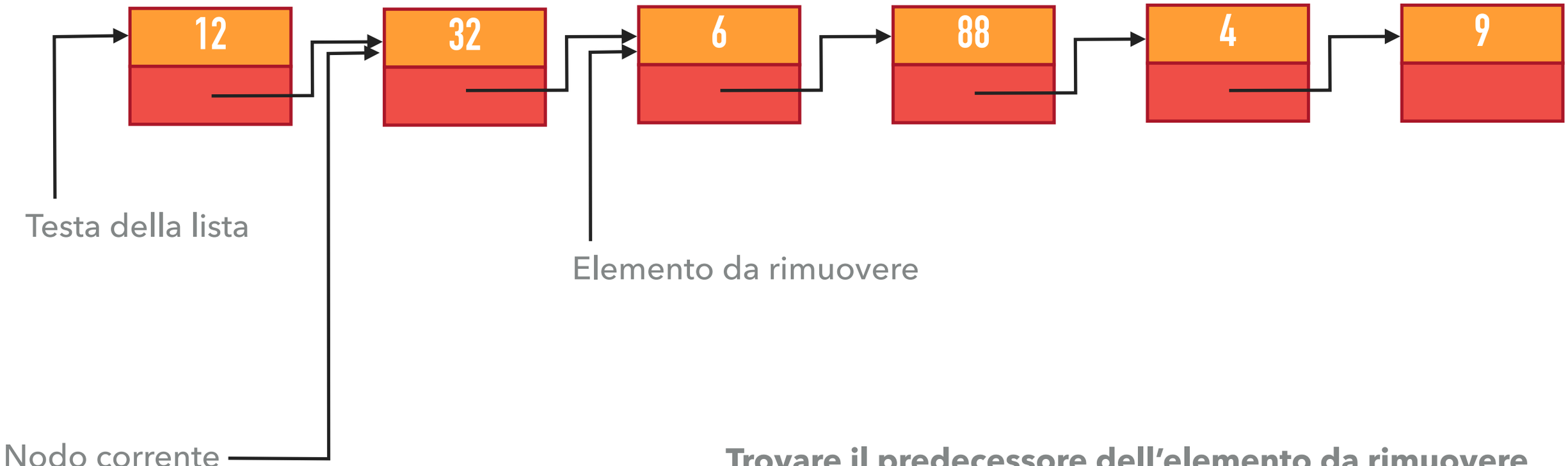


Trovare il predecessore dell'elemento da rimuovere

Impostare il nodo successivo al predecessore uguale al
il node successivo dell'elemento da rimuovere

LA LISTA CONCATENATA: RIMOZIONE DI UN ELEMENTO

Supponiamo di voler aggiungere in testa un nodo avente valore 7

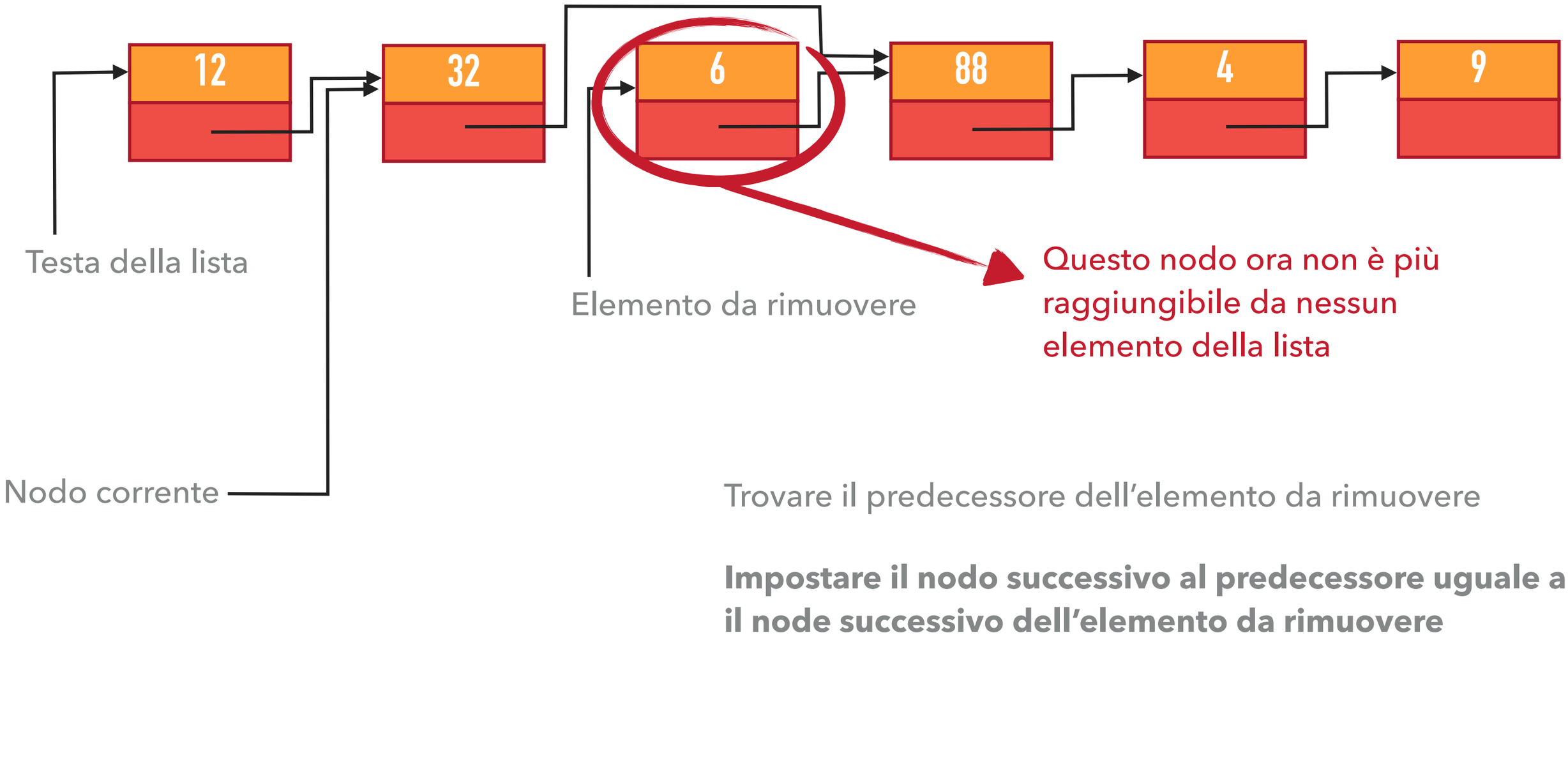


Trovare il predecessore dell'elemento da rimuovere

Impostare il nodo successivo al predecessore uguale al il node successivo dell'elemento da rimuovere

LA LISTA CONCATENATA: RIMOZIONE DI UN ELEMENTO

Supponiamo di voler aggiungere in testa un nodo avente valore 7



Trovare il predecessore dell'elemento da rimuovere

Impostare il nodo successivo al predecessore uguale al il node successivo dell'elemento da rimuovere

LA LISTA CONCATENATA: RIMOZIONE DI UN ELEMENTO

```
Parametri: testa (riferimento al primo elemento della lista),
           x (nodo da rimuovere)
if x == testa
     return testa.next
nodo_corrente = testa
while nodo_corrente.next ≠ x
     nodo_corrente = nodo_corrente.next
nodo_corrente.next = x.next
return testa # ritorniamo comunque la testa della lista
```

LA LISTA CONCATENATA: RIMOZIONE DI UN ELEMENTO

- Abbiamo un ciclo while che scorre la lista
- Nel caso peggiore l'elemento da eliminare è l'ultimo (la **coda**) della lista
- Le altre operazioni hanno costo costante, otteniamo quindi in costo per la rimozione di un elemento che è lineare rispetto alla lunghezza della lista: $O(n)$

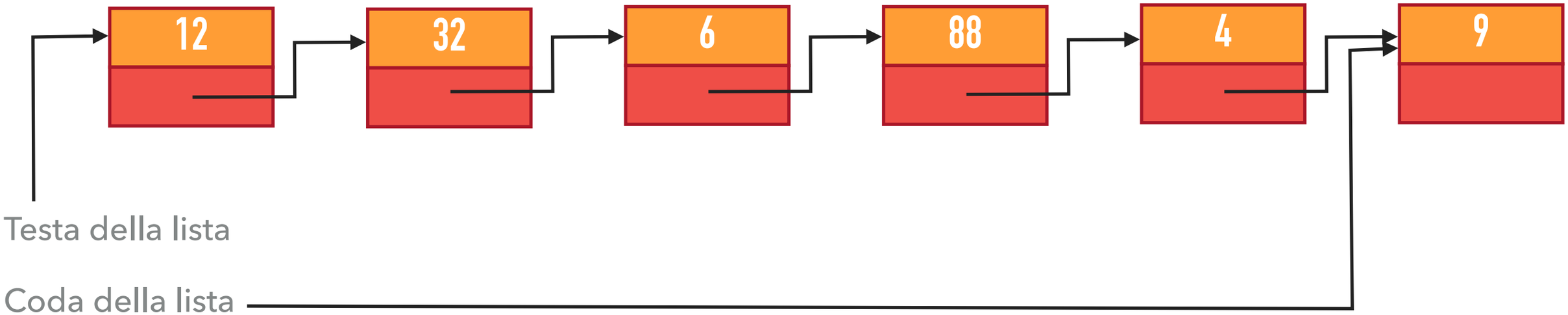
LA LISTA CONCATENATA: DISCUSSIONE

- Abbiamo visto come possiamo memorizzare i dati in modo diverso con differenti compromessi
- Vedremo come possiamo migliorare usando strutture dati più complesse: liste concatenate doppie, alberi, etc.

Operazione	Tempo (array)	Tempo (lista concatenata singola)
Accedere ad un elemento	$O(1)$	$O(n)$
Ricerca	$O(n)$	$O(n)$
Inserimento	$O(n)$	$O(1)$ (se in testa)
Rimozione	$O(n)$	$O(n)$

LA LISTA CONCATENATA: DISCUSSIONE

Le strutture dati possono essere modificate per migliorare il costo computazionale di alcune operazioni frequenti. Per esempio, se volessimo inserire in coda?



Normalmente dovremmo scorrere sempre l'intera lista

Ma è più veloce aggiungere un riferimento alla coda della lista ed aggiungere un elemento in coda in tempo costante

Dobbiamo però ricordarci di sempre di tenere aggiornato correttamente il puntatore alla coda!