INFORMATICA

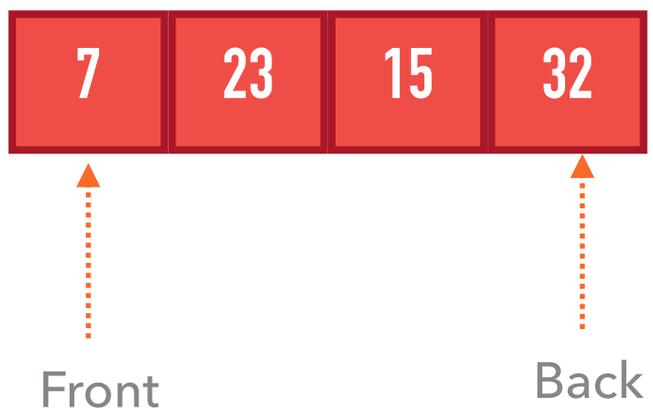
CODE

- Struttura dati astratta con le seguenti operazioni:
  - Enqueue. Inserisce in elemento in fondo alla coda
  - Dequeue. Rimuove l'elemento nella coda che è stato inserito più indietro nel tempo
- Dette anche FIFO (First in - First out)
- Altre possibili operazioni:
  - Empty. Per chiedere se la coda è vuota

CODA

Operazioni da eseguire:

```
enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()
```

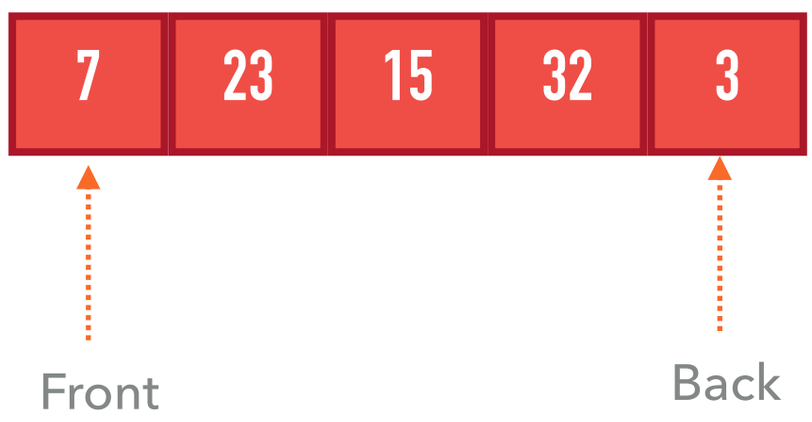


Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseriti

CODA

Operazioni da eseguire:

```
enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()
```

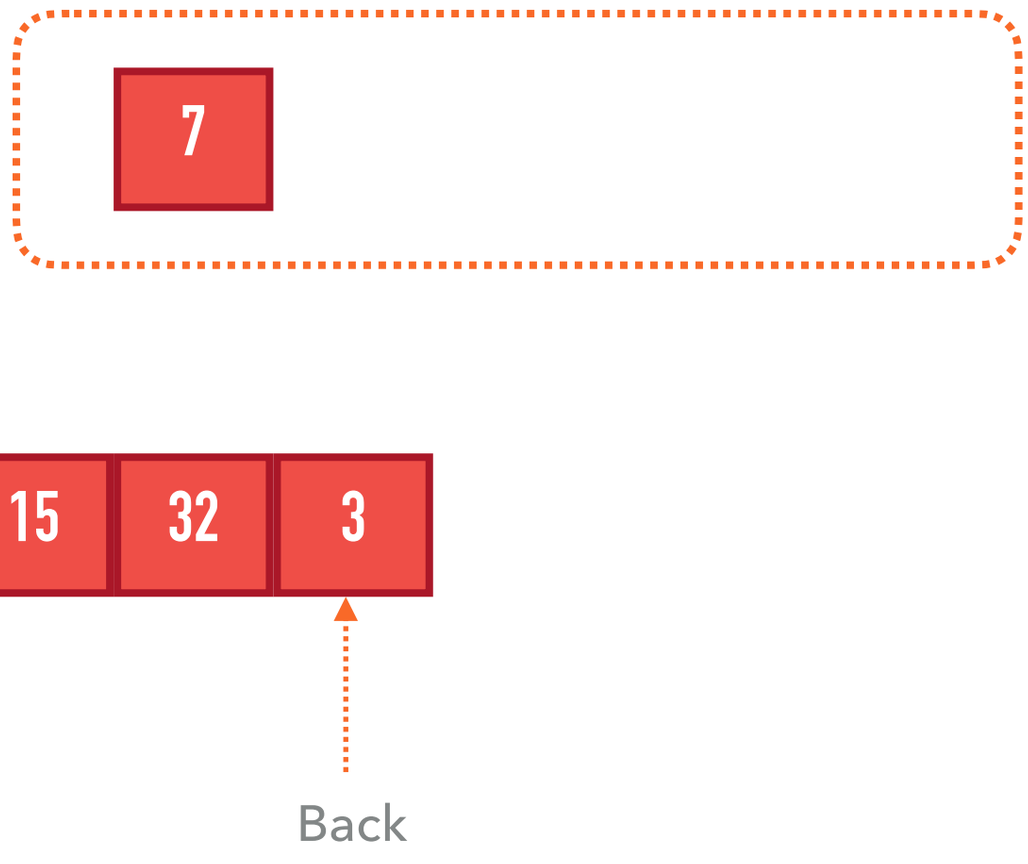


Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseriti

CODA

Operazioni da eseguire:

enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()

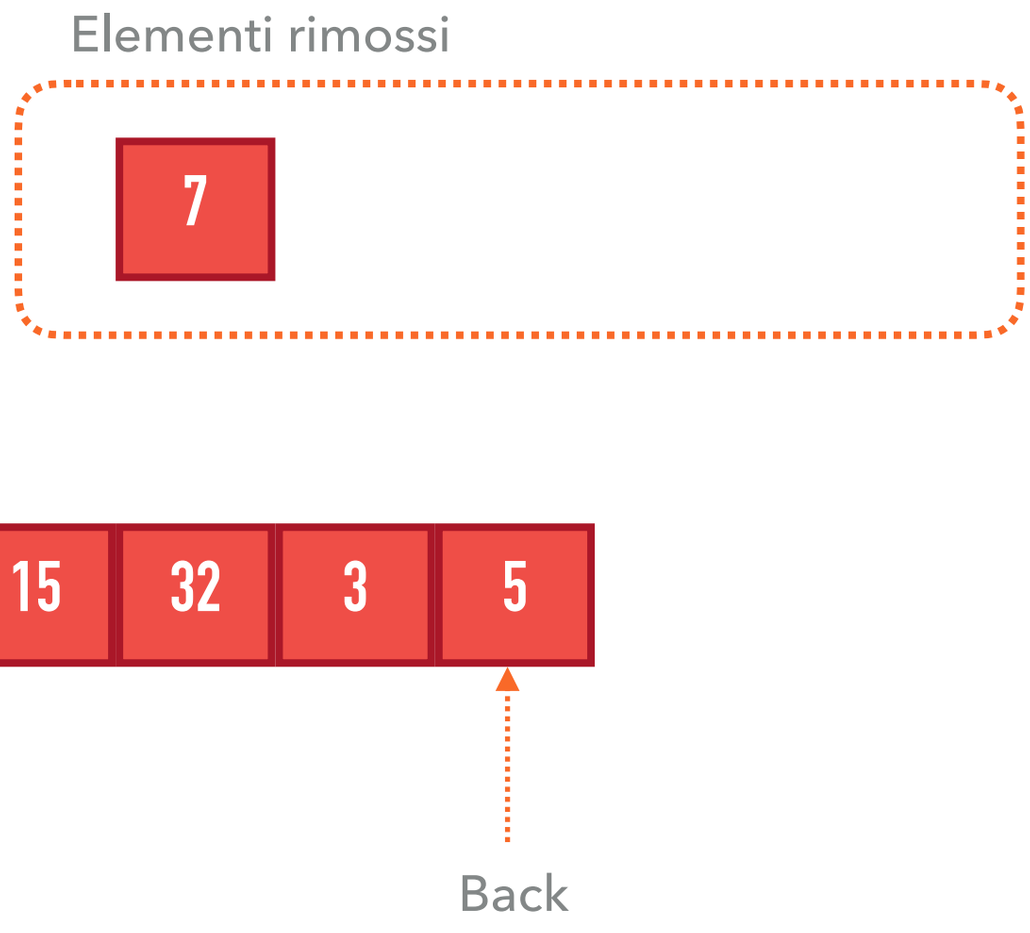


Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseriti

CODA

Operazioni da eseguire:

```
enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()
```

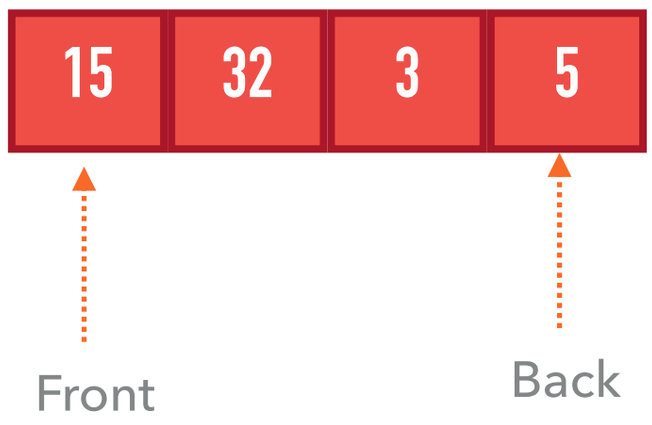


Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseriti

CODA

Operazioni da eseguire:

```
enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()
```

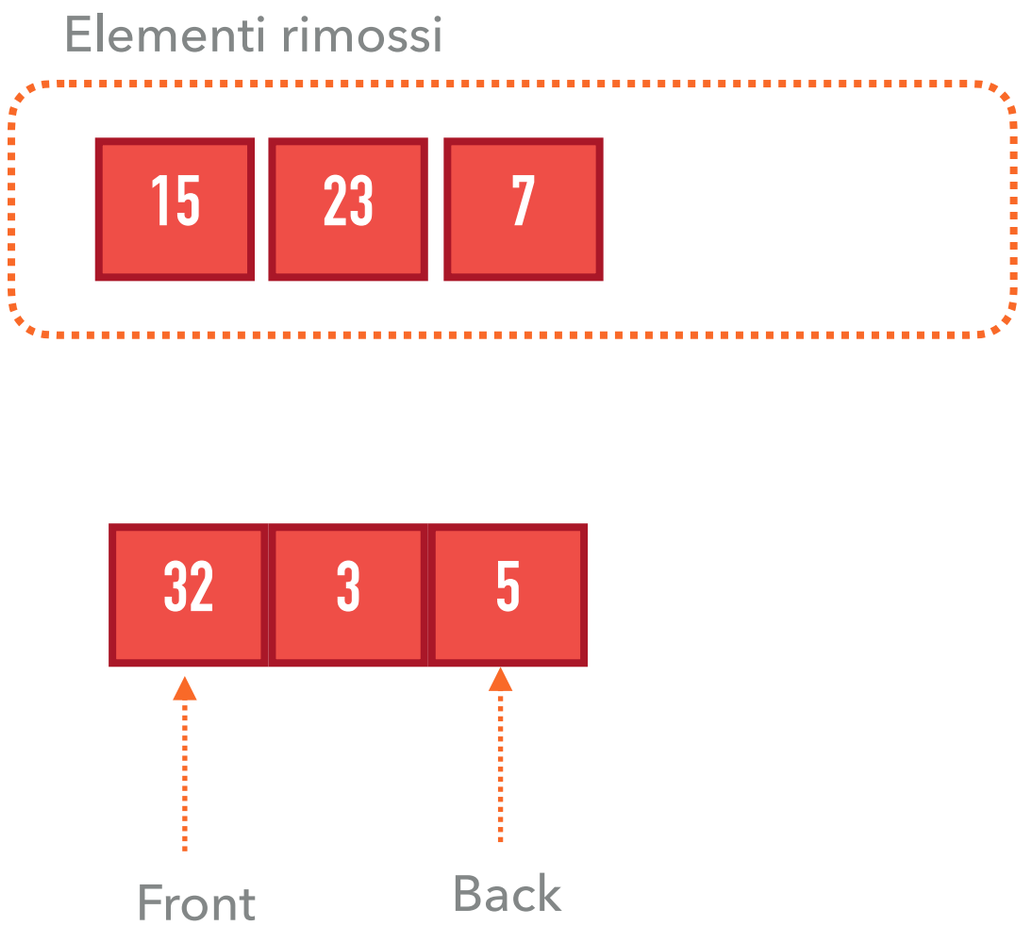


Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseriti

CODA

Operazioni da eseguire:

```
enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()
```



Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseriti

CODE: A COSA SERVONO

- Gestione di dati in arrivo: di solito vogliamo siano processati nello stesso ordine con cui li riceviamo
- All'interno di altri algoritmi: ricerca di un percorso in un grafo, etc.
- Coda dei processi da eseguire, etc.
- Forse una delle strutture più comuni nell'informatica

QUIZ: CHE STRUTTURA DATI?

In una struttura dati inseriamo in ordine i valori:

4 5 9 3

Rimuovendoli li otteniamo in ordine:

3 9 5 4

Quale di queste strutture dati astratte potrebbe essere?

A) coda

B) stack

C) reference

D) nessuna
delle precedenti

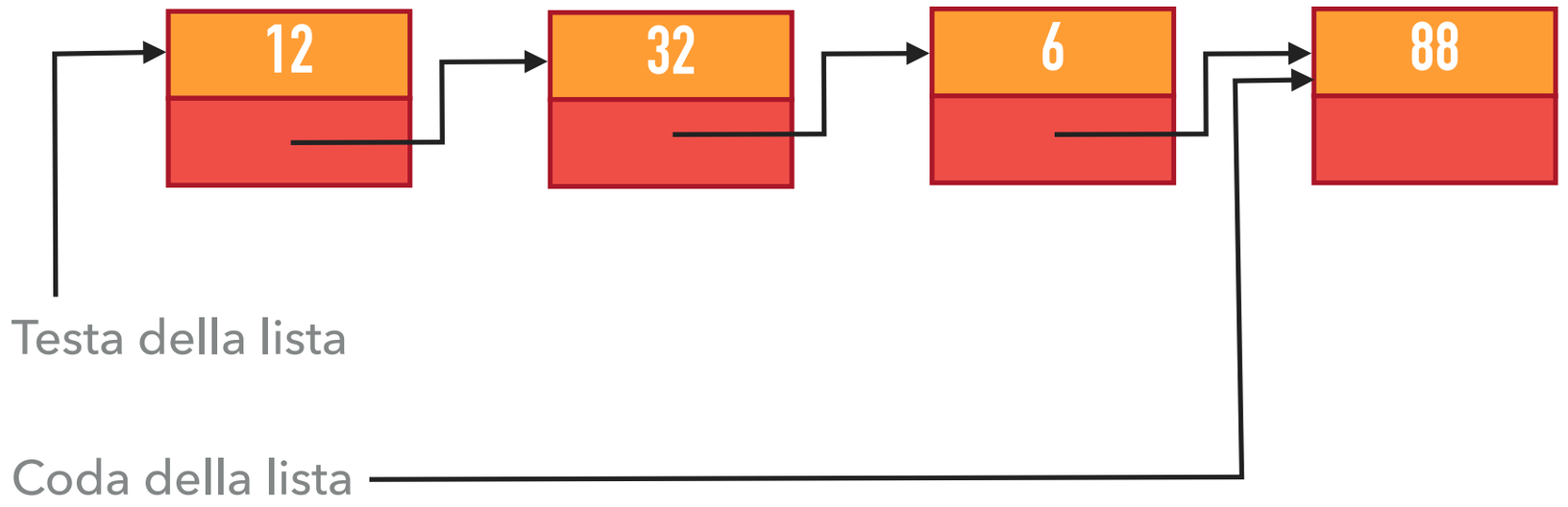
IMPLEMENTAZIONE CON LISTE CONCATENATE

- Possiamo usare una lista concatenata singola in cui la coda inizia alla testa della lista e termina con l'ultimo elemento
- La rimozione del primo elemento è rapida…
- …ma l'aggiunta in coda alla lista richiede di scorrere tutta la lista
- Possiamo eliminare il problema in almeno due modi:
  - Lista con riferimento alla coda
  - Lista circolare con riferimento all'ultimo elemento (invece che al primo)

IMPLEMENTAZIONE CON LISTE CONCATENATE

enqueue(4)

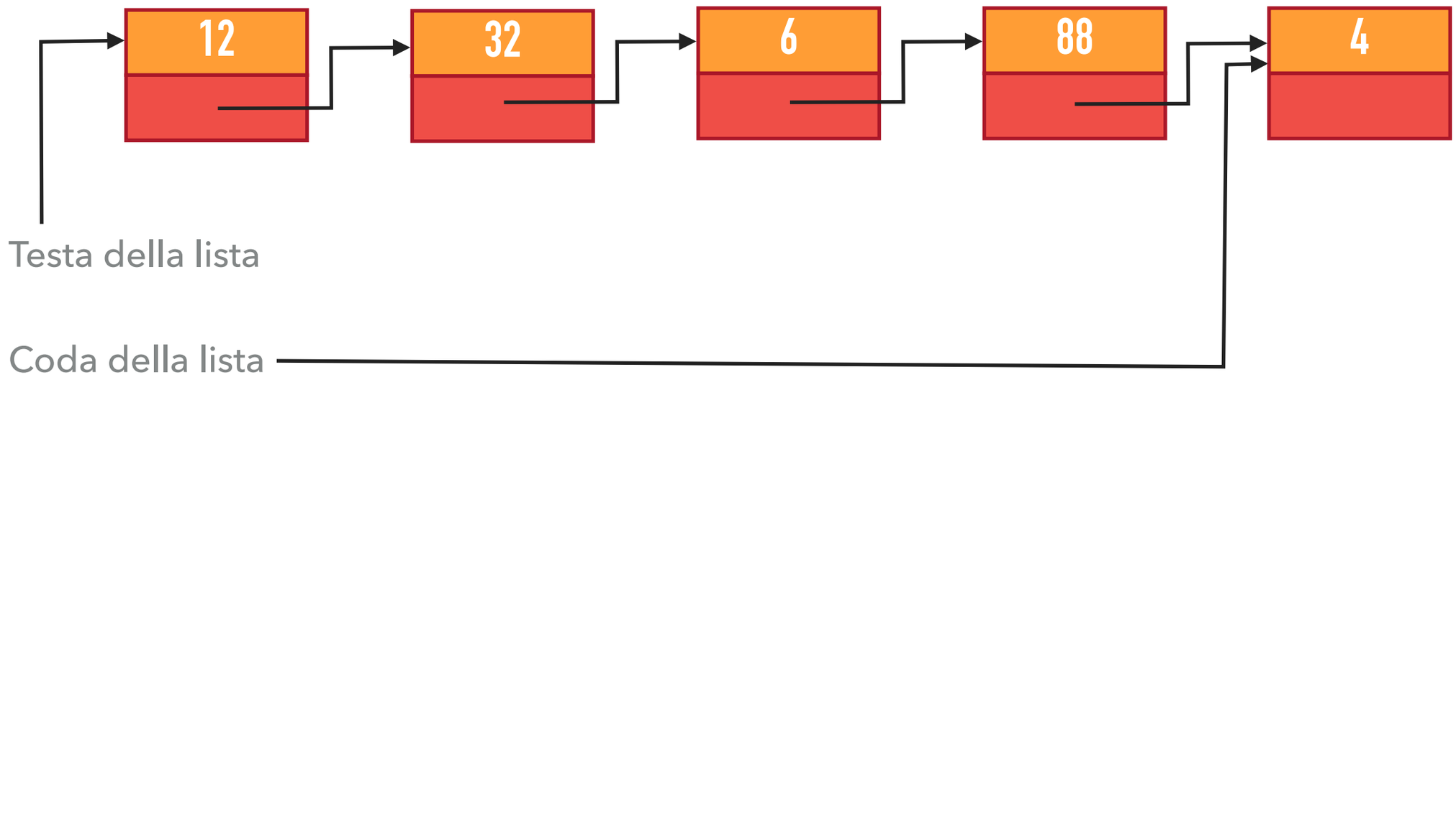
dequeue()



IMPLEMENTAZIONE CON LISTE CONCATENATE

enqueue(4)

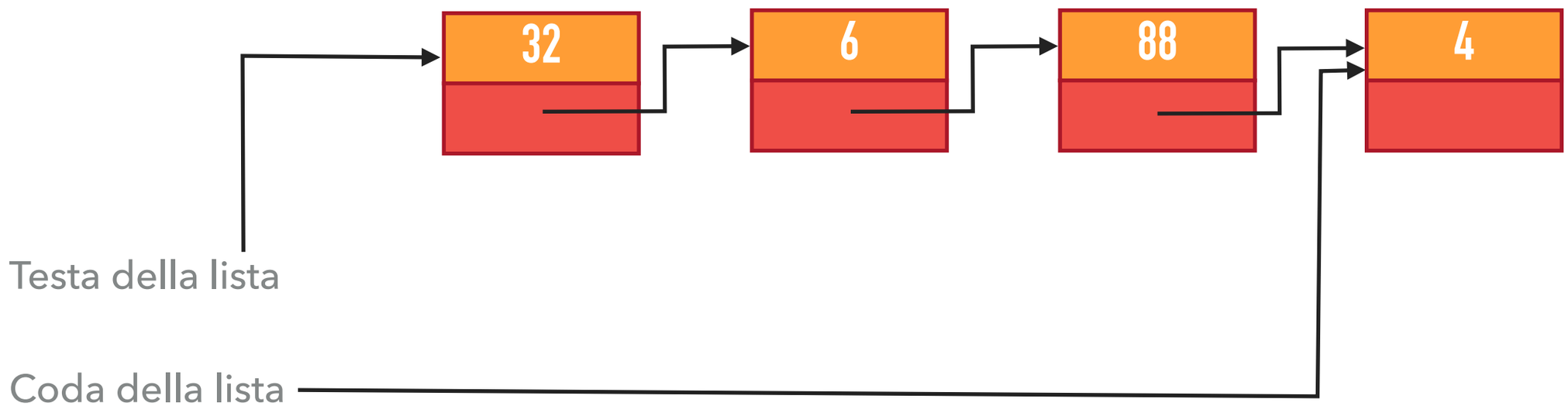
dequeue()



IMPLEMENTAZIONE CON LISTE CONCATENATE

enqueue(4)

dequeue()



Valore ritornato: 12

IMPLEMENTAZIONE CON LISTE CONCATENATE

- La rimozione in testa per "dequeue" richiede tempo $O(1)$
- L'aggiunta in coda richiede anch'essa tempo $O(1)$ ma solo perché abbiamo un reference all'ultimo elemento della lista
- Anche in questo caso potevamo usare una lista concatenata doppia (con un reference alla coda)...
- ...ma non avremmo avuto vantaggi in termini di costo computazionale

IMPLEMENTAZIONE CON ARRAY

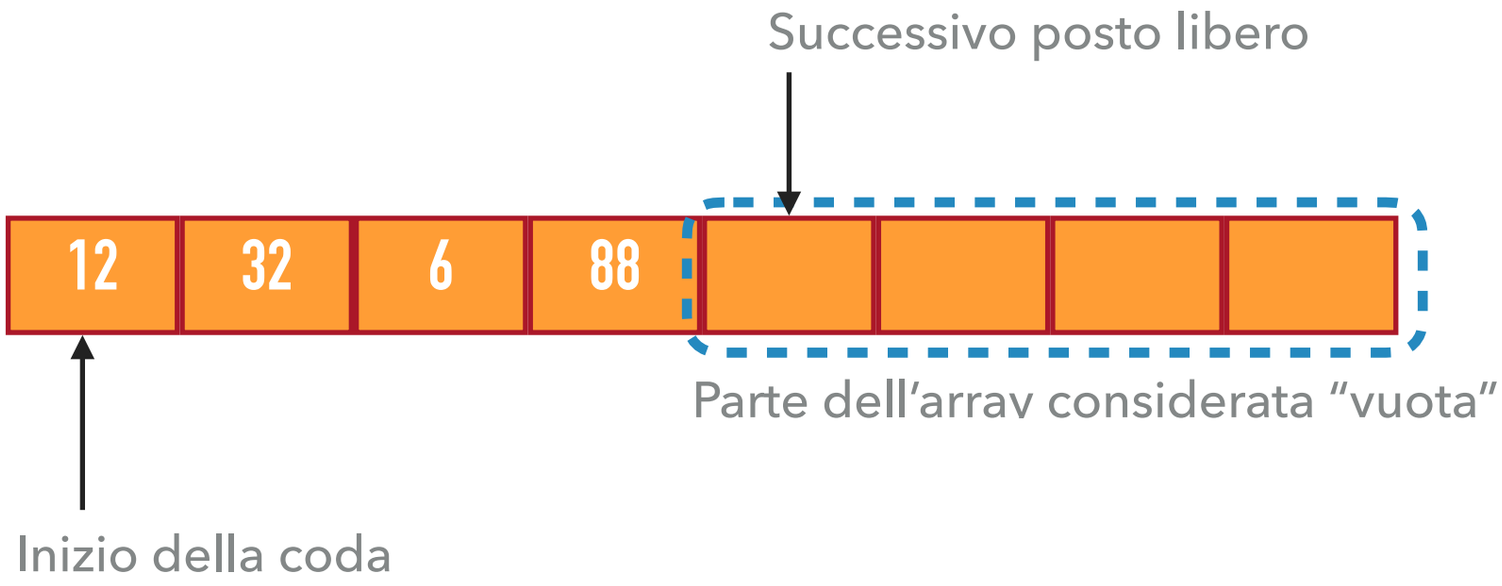
- Possiamo utilizzare un array con n posizioni per memorizzare una coda in grado di contenere $n - 1$ elementi
- Si utilizza un **buffer circolare**, che, finché non superiamo $n - 1$ elementi nella coda, permette di fare inserimenti e rimozioni in tempo costante
- Teniamo due indici:
  - Dove inizia la coda
  - La prima posizione libera dopo la fine della coda

IMPLEMENTAZIONE CON ARRAY

- Il primo indice denota dove si trova il primo elemento della coda
- Il secondo indice denota dove verrà inserito il prossimo elemento
- Dequeue: spostamento in avanti del primo indice
- Enqueue: spostamento in avanti del secondo indice
- Gli indici sono considerati modulo n

IMPLEMENTAZIONE CON ARRAY

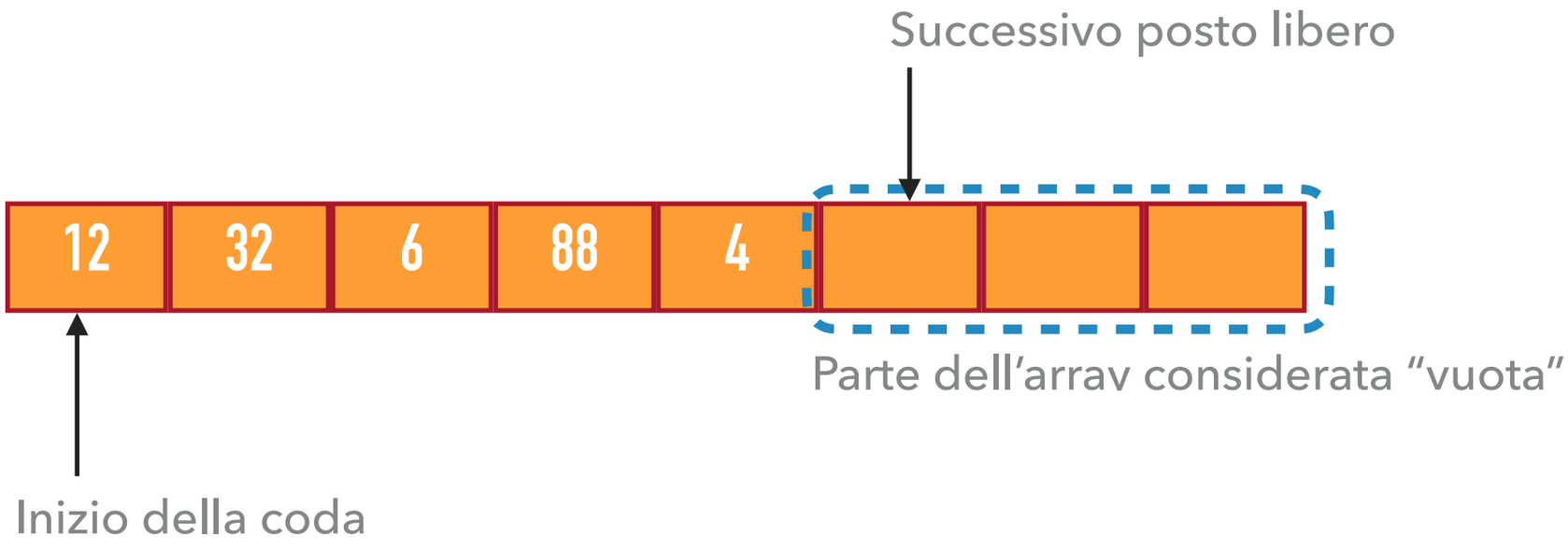
```
enqueue(4)
dequeue()
```



IMPLEMENTAZIONE CON ARRAY

enqueue(4)

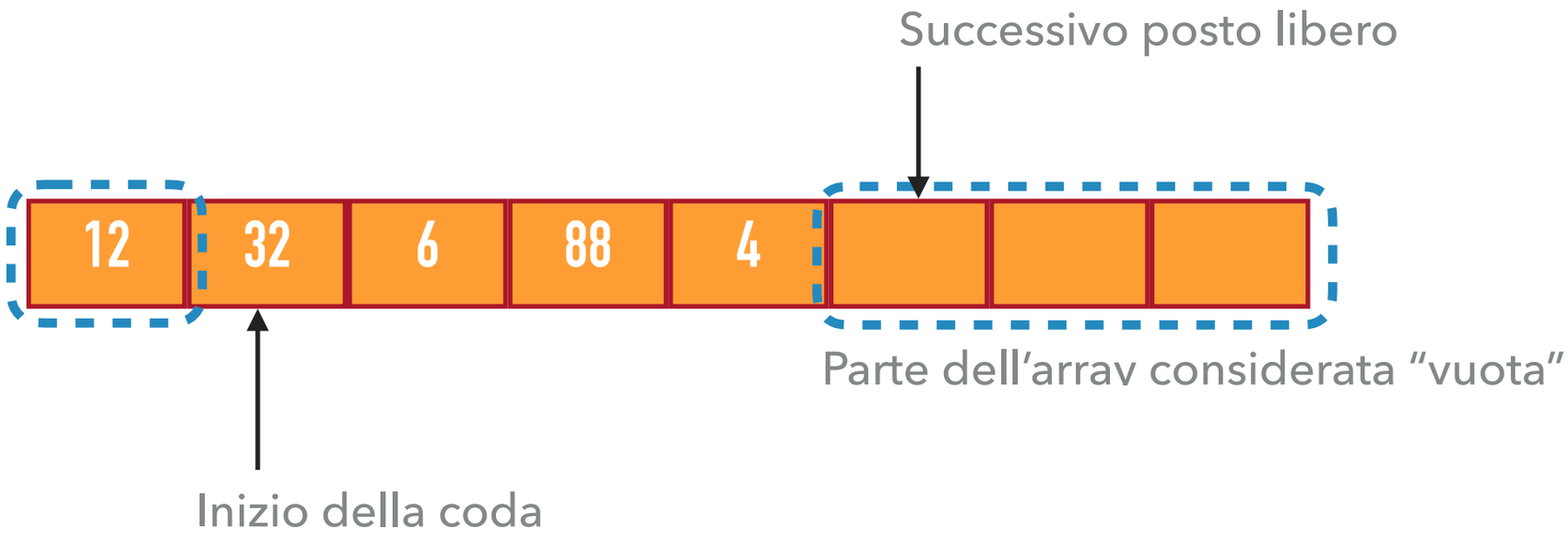
dequeue()



IMPLEMENTAZIONE CON ARRAY

enqueue(4)

dequeue()



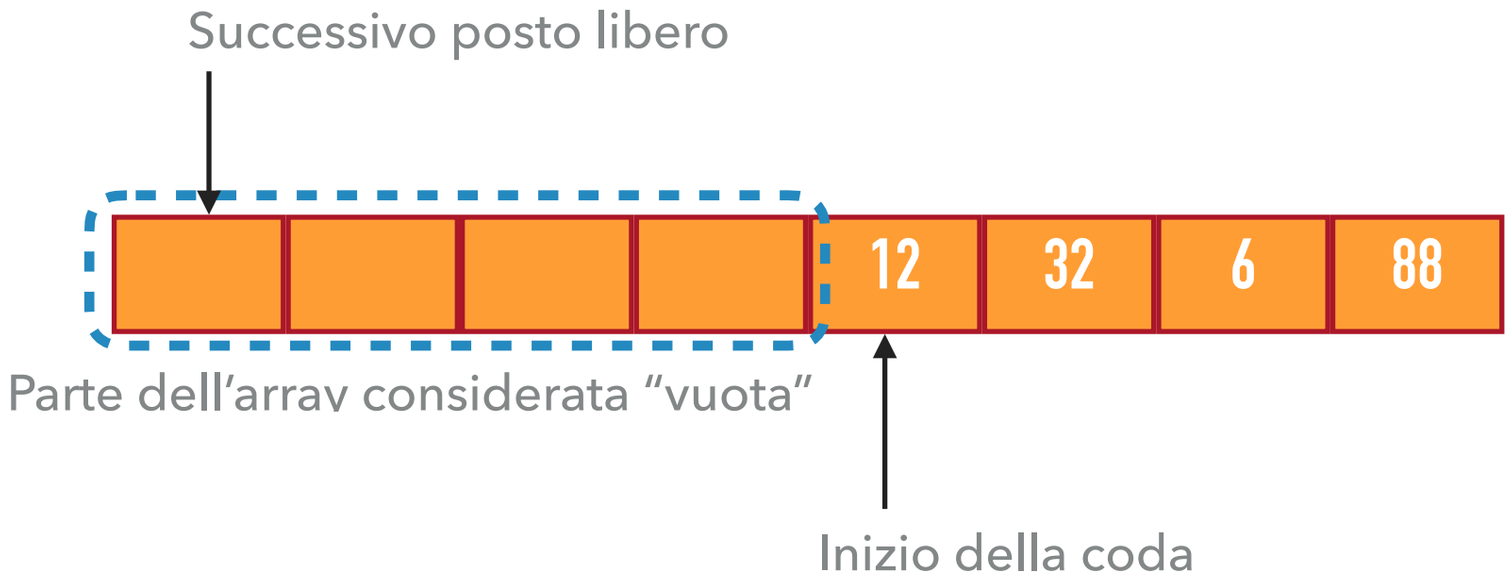
Valore ritornato: 12

IMPLEMENTAZIONE CON ARRAY #2

enqueue(4)

dequeue()

Le posizioni sono sempre considerate modulo n, quindi il successore della posizione $n-1$ è la posizione 0

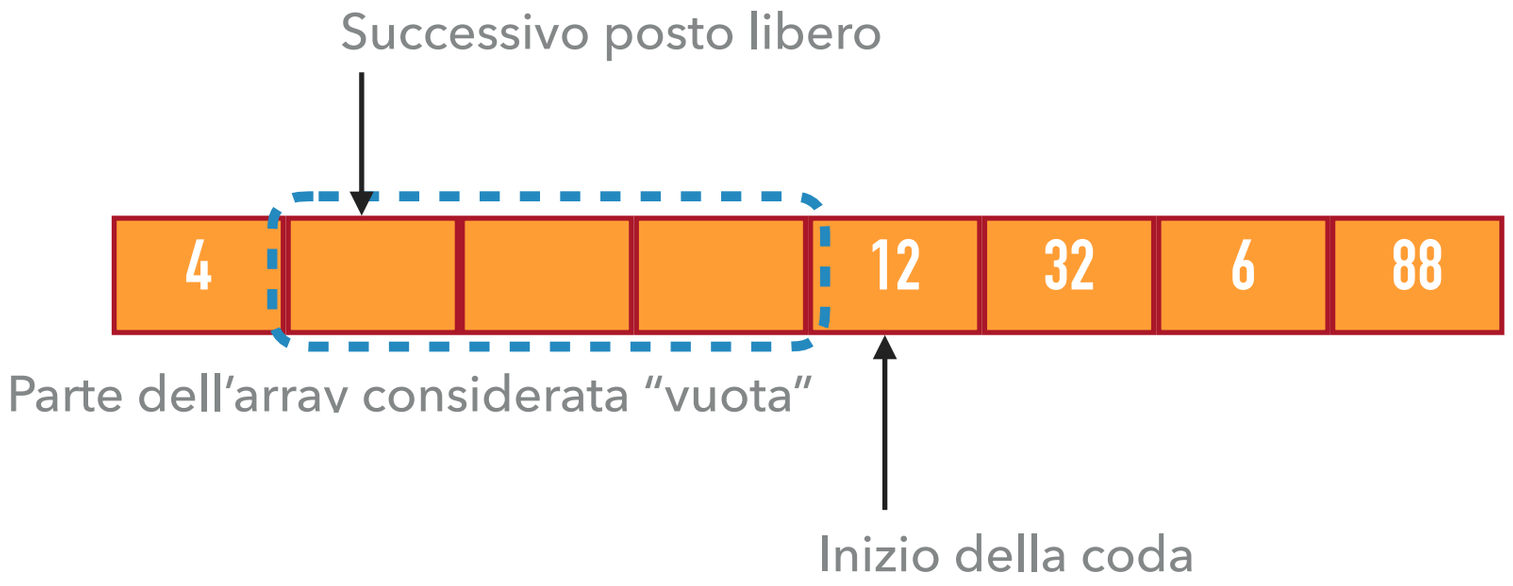


Questa array e quello dell'esempio precedente rappresentano la stessa coda!

IMPLEMENTAZIONE CON ARRAY #2

enqueue(4)

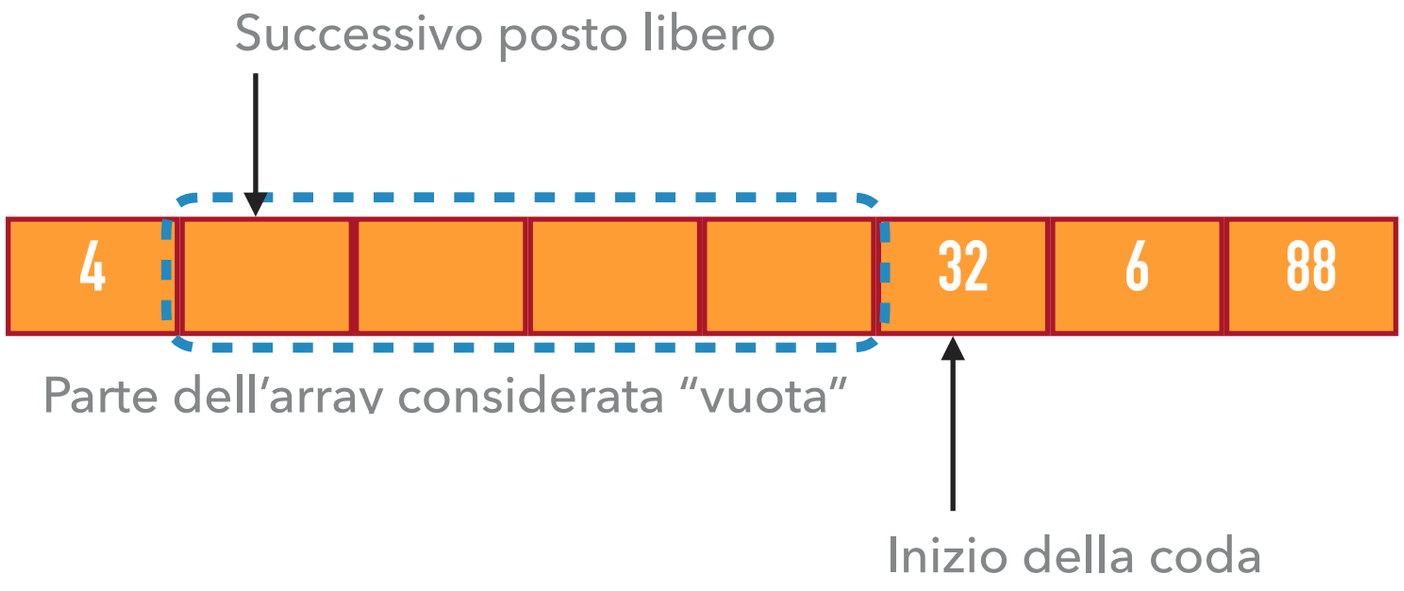
dequeue()



IMPLEMENTAZIONE CON ARRAY #2

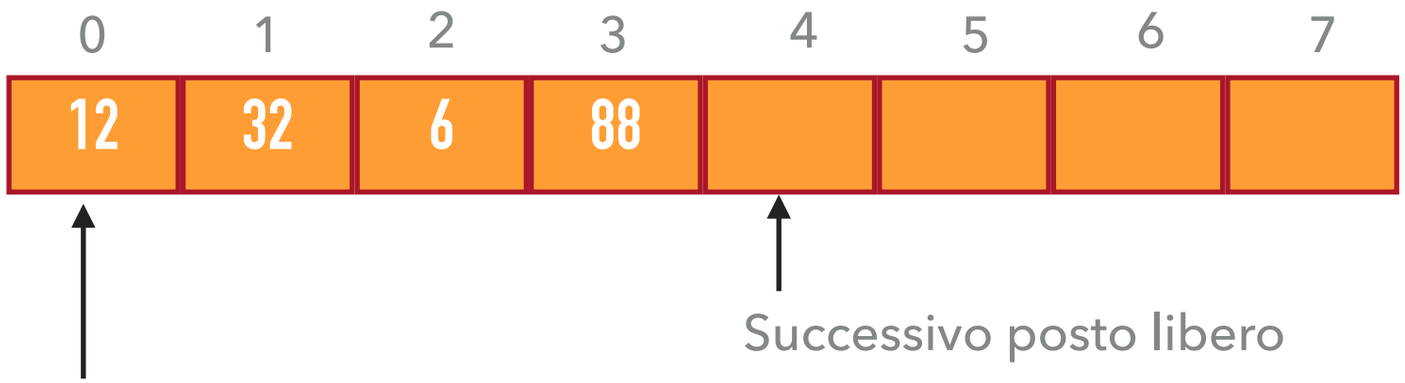
enqueue(4)

dequeue()

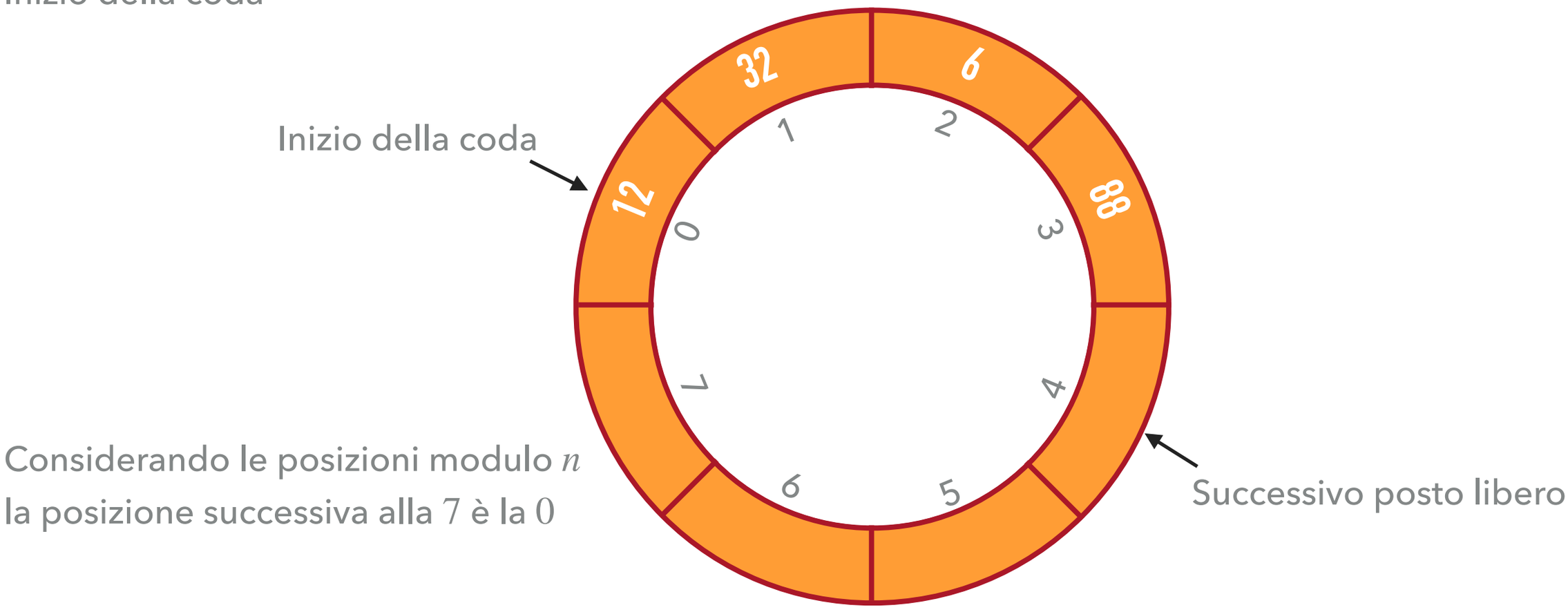


Valore ritornato: 12

IMPLEMENTAZIONE CON ARRAY

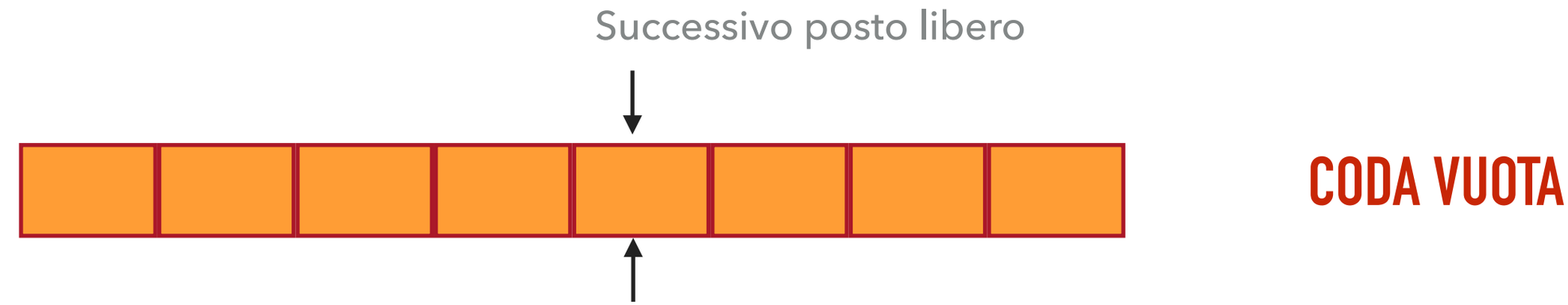


Due modi di vedere lo stesso array:
in modo lineare e considerando
le posizioni modulo n

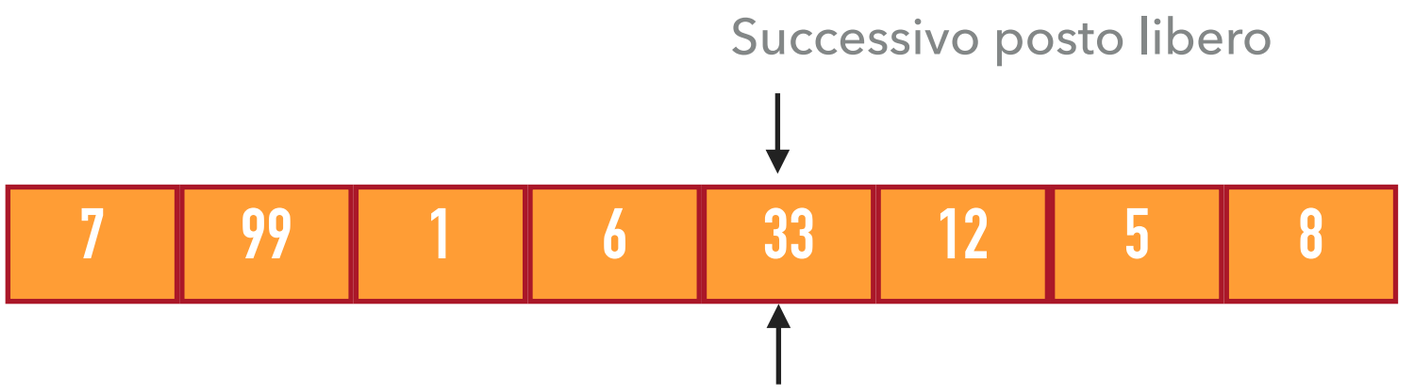


PERCHÉ N-1 ELEMENTI?

Se consentissimo di memorizzare n elementi (invece di $n-1$)
non potremmo distinguere tra due casi



CODA VUOTA



CODA PIENA

*Conflitto potenzialmente risolubile memorizzando informazioni aggiuntive

IMPLEMENTAZIONE CON ARRAY

- Fino a quando il numero di elementi nella coda rimane limitato ogni operazione di inserimento e rimozione richiede tempo costante
- Se non avessimo i due indici ma tenessimo sempre l'inizio della coda in posizione 0 dovremmo copiare ad ogni operazione di dequeue!

IMPLEMENTAZIONE CON STACK

- Possiamo utilizzare degli stack per simulare una coda?
- Possiamo utilizzare due stack:
  - In uno effettueremo le operazioni di enqueue
  - Nell'altro le operazioni di dequeue
  - Dobbiamo – in qualche modo – spostare elementi da uno stack all'altro

IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)
dequeue()
enqueue(3)
dequeue()
```

Rimozioni fatte
nello stack A

Stack A

Stack B

Inserimenti fatti
nello stack B

IMPLEMENTAZIONE CON STACK

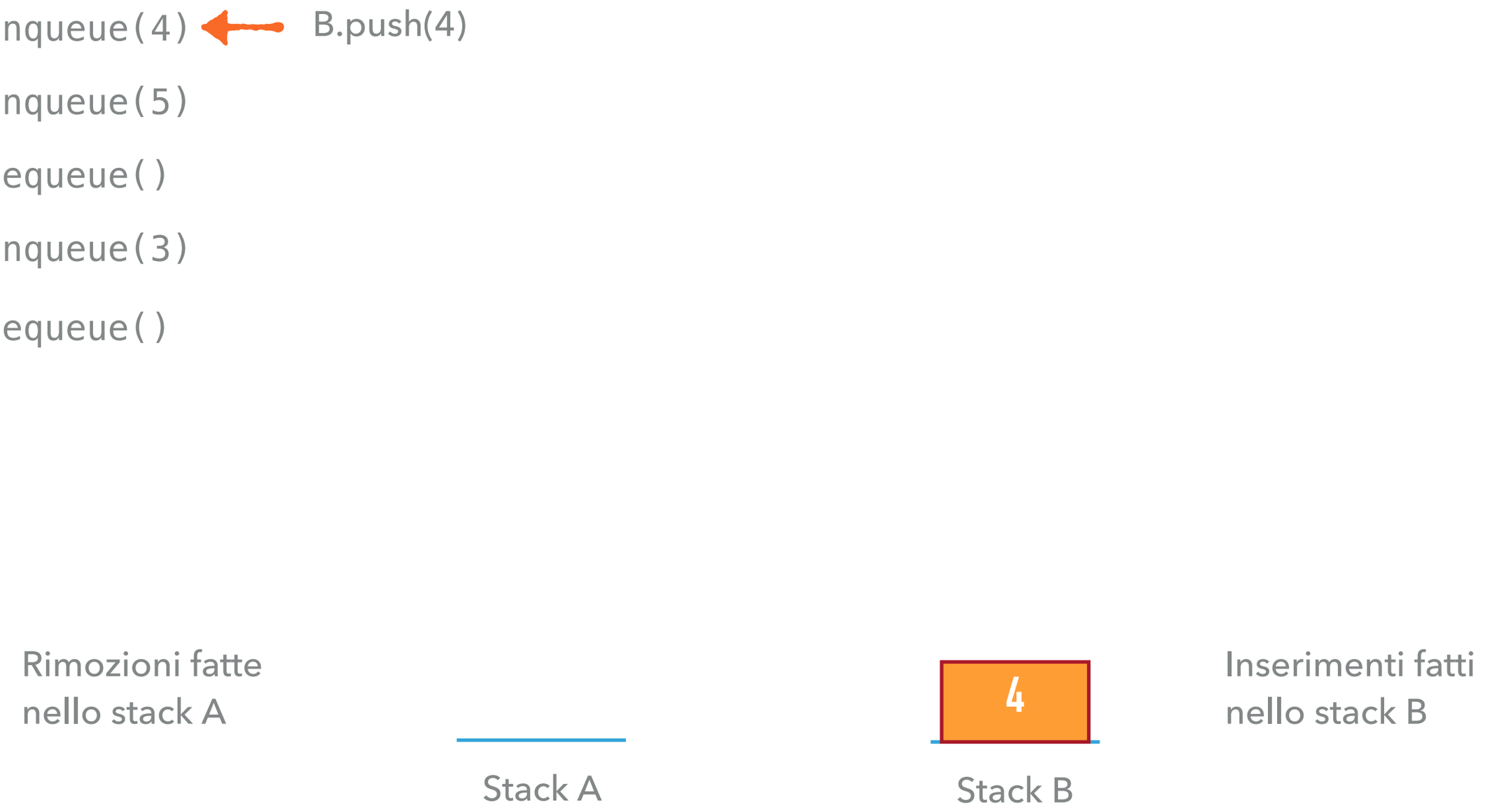
enqueue(4) ← B.push(4)

enqueue(5)

dequeue()

enqueue(3)

dequeue()

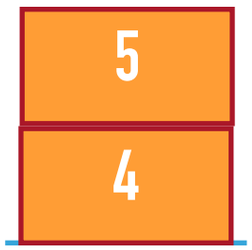


IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)  <-- B.push(5)
dequeue()
enqueue(3)
dequeue()
```

Rimozioni fatte nello stack A

Stack A



Stack B

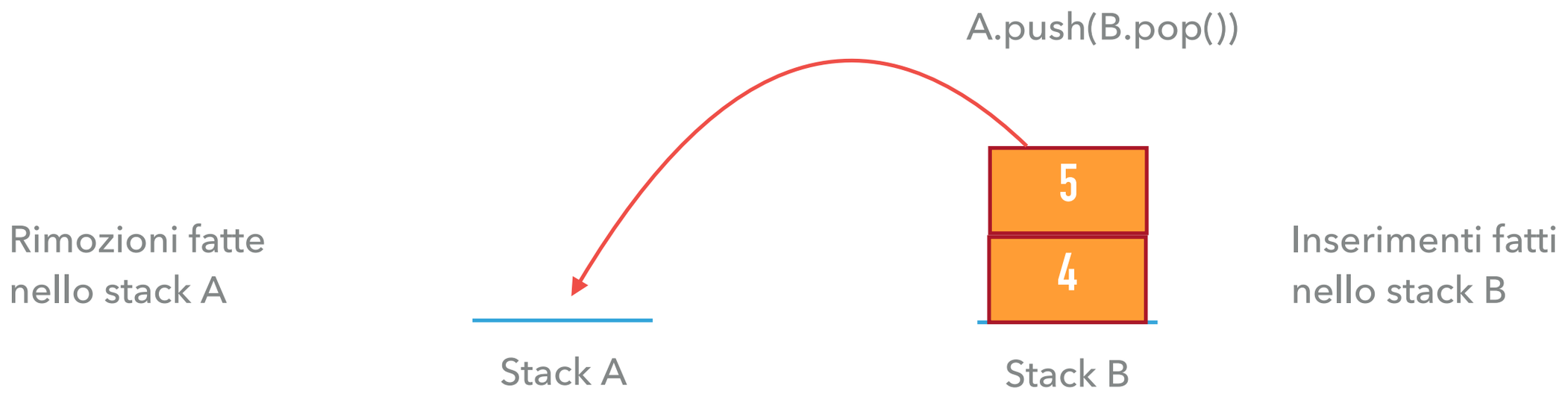
Inserimenti fatti nello stack B

IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)
dequeue()
enqueue(3)
dequeue()
```

Lo stack A è vuoto!

“ribaltiamo” B in A

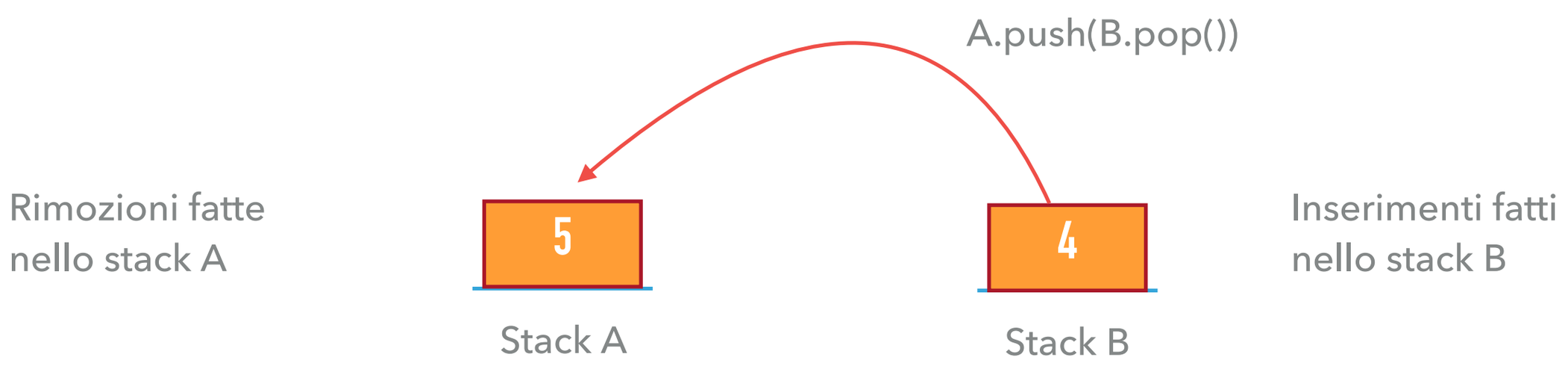


IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)
dequeue()
enqueue(3)
dequeue()
```

Lo stack A è vuoto!

“ribaltiamo” B in A

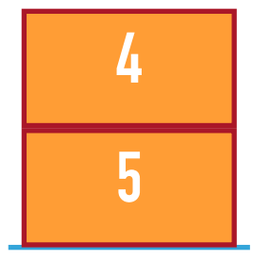


IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)
dequeue()
enqueue(3)
dequeue()
```

Ora il top dello stack A contiene l'elemento da rimuovere

Rimozioni fatte nello stack A



Stack A

Stack B

Inserimenti fatti nello stack B

IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)
dequeue()
enqueue(3)
dequeue()
```

Valore ritornato: 4

Rimozioni fatte nello stack A

Stack A

Stack B

Inserimenti fatti nello stack B

IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)
dequeue()
enqueue(3)
dequeue()
```

B.push(3)

Rimozioni fatte nello stack A

Stack A

Stack B

Inserimenti fatti nello stack B

IMPLEMENTAZIONE CON STACK

```
enqueue(4)
enqueue(5)
dequeue()
enqueue(3)
```

dequeue( ) ← Valore ritornato: **5**

Se lo stack A non è vuoto ci basta fare una operazione di pop

Rimozioni fatte nello stack A

Stack A

3

Stack B

Inserimenti fatti nello stack B

DOUBLE-ENDED QUEUE

- Ibrido di stack e coda. Possiamo inserire e rimuovere da entrambe le estremità:
  - Inserisci all'inizio o alla fine
  - Rimuovi dall'inizio o dalla fine
- Come negli altri casi può essere implementata come lista concatenata oppure con un array utilizzato come buffer circolare