UNIVERSITÀ DEGLI STUDI DI TRIESTE

MODULO 2: Coda o Queue

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

Trieste, 23 aprile 2026

CODE

- **Struttura dati astratta** con le seguenti operazioni:
  - **Enqueue**. Inserisce un elemento in fondo alla coda
  - **Dequeue**. Rimuove l’elemento della 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: ESEMPIO (1/6)

Operazioni da eseguire:

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

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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: ESEMPIO (2/6)

Operazioni da eseguire:

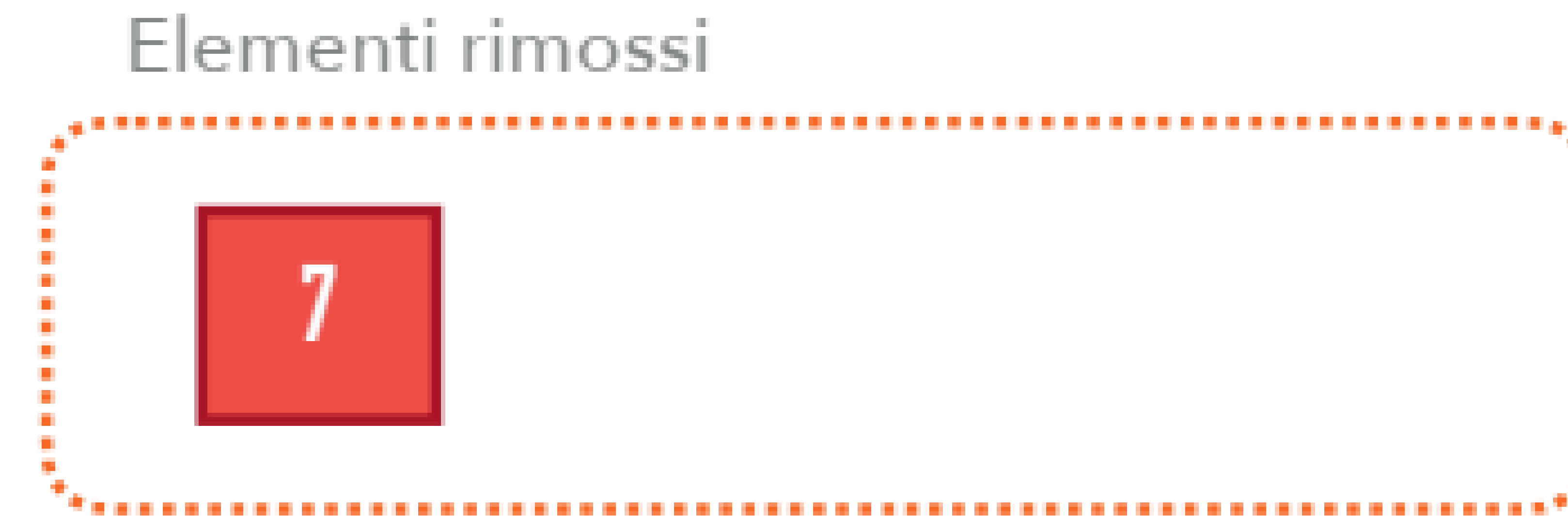
```
enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()
```

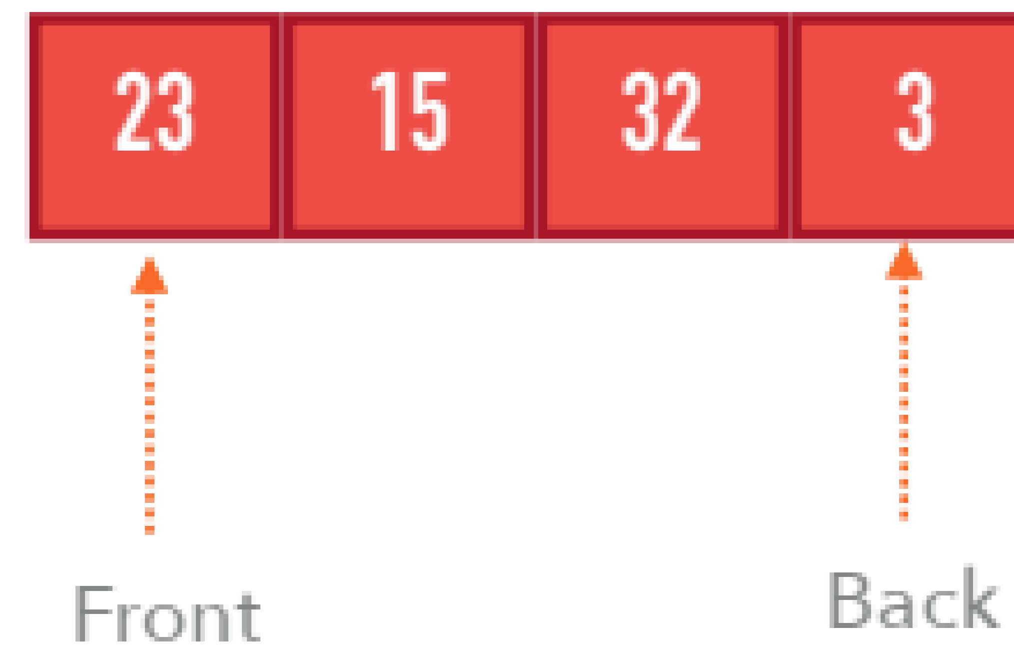
Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseritiav

CODA: ESEMPIO (3/6)

Operazioni da eseguire:

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





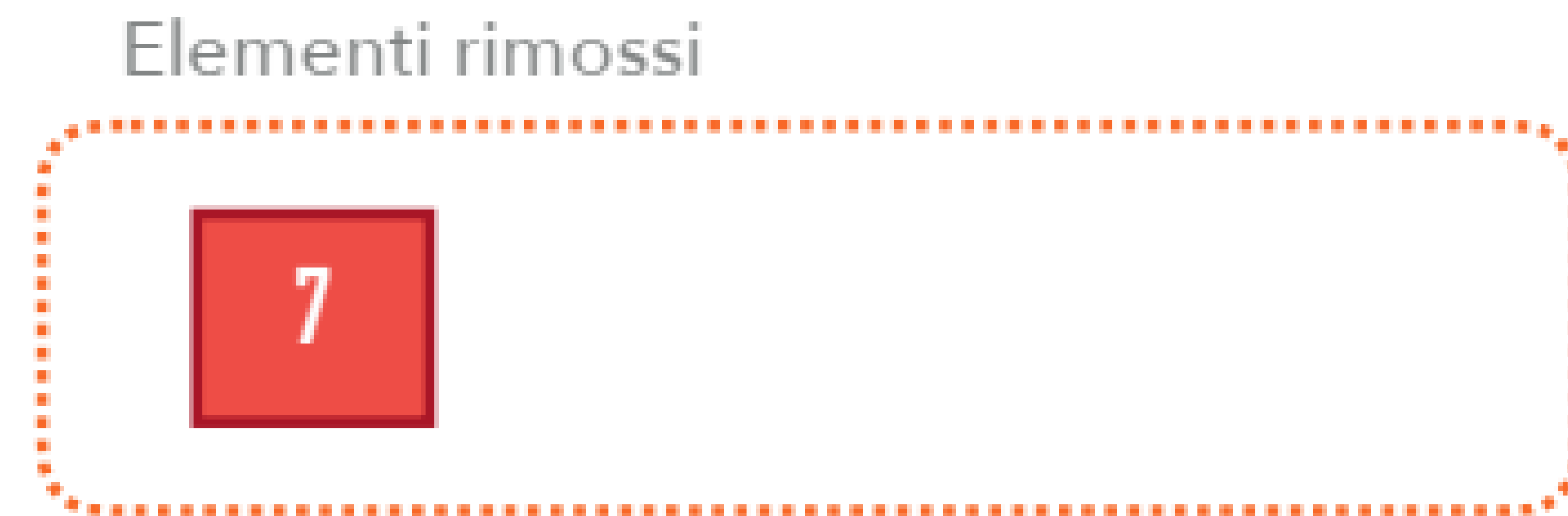
Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseritiav

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: ESEMPIO (4/6)

Operazioni da eseguire:

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



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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: ESEMPIO (5/6)

Operazioni da eseguire:

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

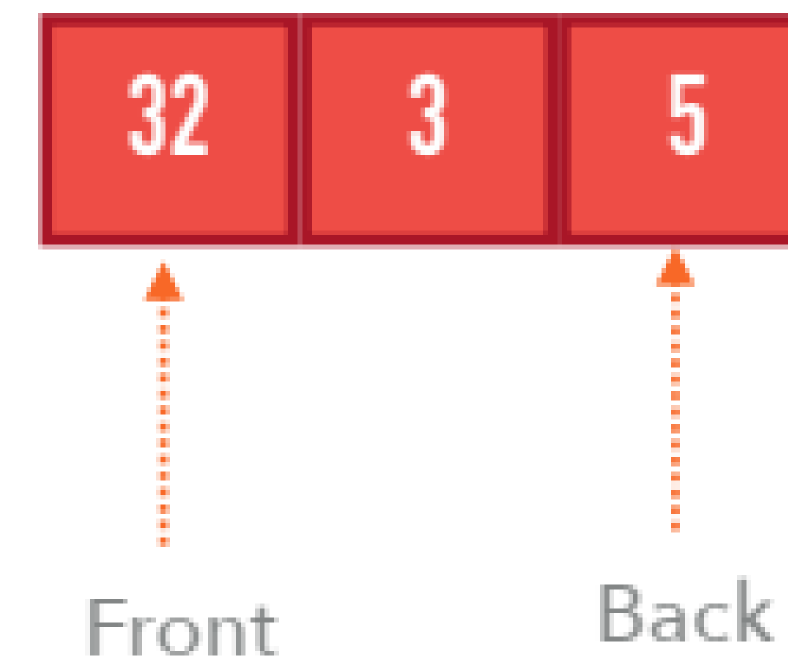
Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseritiav

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: ESEMPIO (6/6)

Operazioni da eseguire:

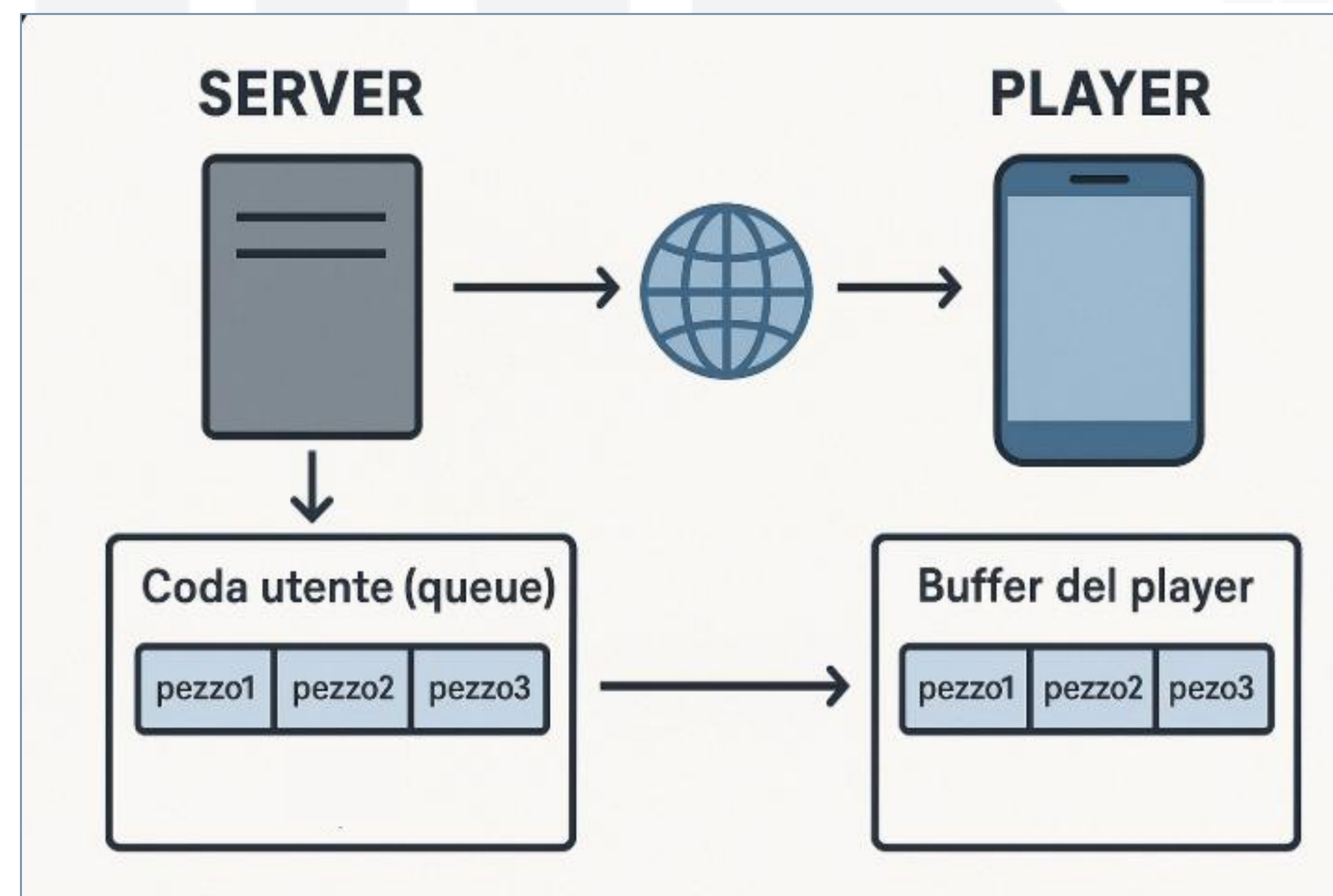
```
enqueue(3)
dequeue()
enqueue(5)
dequeue()
dequeue()
```



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

CODE: UTILITA'

- **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, ***streaming video***, etc.
- Forse una delle strutture più comuni nell'informatica



Il programma («player») riproduce il video (esempio: **VLC**, **YouTube app**, **Netflix app**, **Twitch player**...). **Il buffer** è **una piccola memoria temporanea dentro** il player.

A cosa serve il buffer?
Il **buffer** si riempie con **un po' di dati video** prima che il player inizi a mostrarli a schermo. Serve per compensare:
- **Ritardi della rete** (lag)
- **Piccole interruzioni**
- **Variazioni della velocità di internet**

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODE: IMPLEMENTAZIONE

- Vediamo due modi:
  - Coda implementata **tramite liste concatenate singole**
  - Coda implementata **tramite array**

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODE: 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

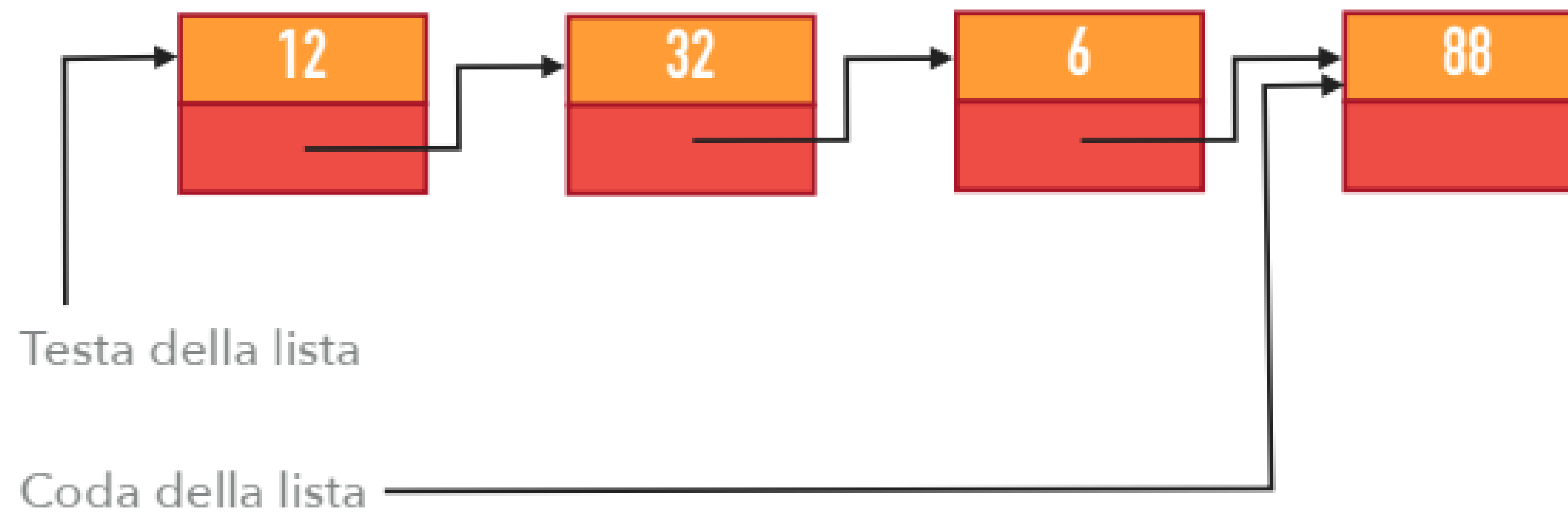
- La rimozione del primo elemento è rapida…
- *…ma l'aggiunta in coda alla lista richiede di scorrere tutta la lista*

- Possiamo eliminare il problema in almeno due modi:
  - Lista con riferimento alla coda (***tail***)
  - **Lista circolare** con riferimento all'ultimo elemento (invece che al primo)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON LISTE CONCATENATE

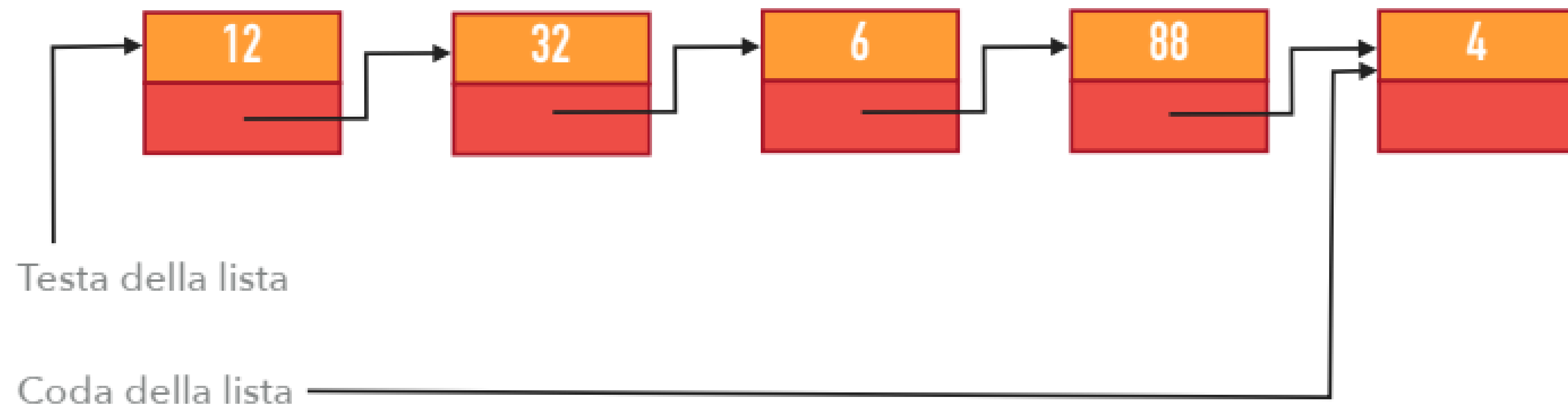
```
enqueue(4)
dequeue()
```



UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON LISTE CONCATENATE

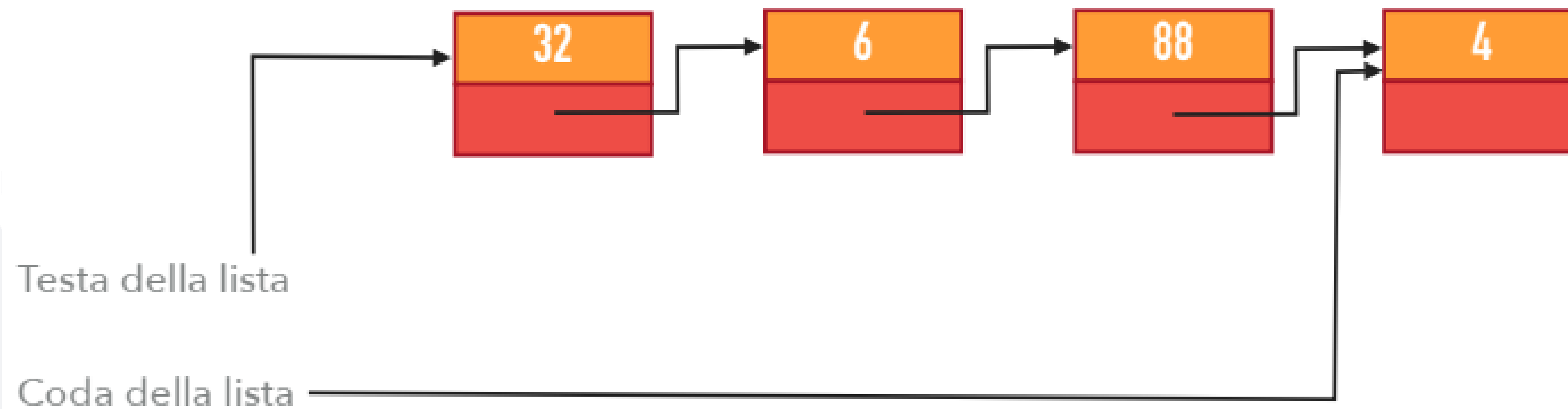
```
enqueue(4)
dequeue()
```



UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON LISTE CONCATENATE

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



Valore ritornato: 12

CODA: IMPLEMENTAZIONE CON LISTE CONCATENATE

- La **rimozione** in testa (“dequeue”) richiede tempo ***O*(1)**
- **L’aggiunta** in coda («enqueue») 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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON **ARRAY** (1/2)

- Possiamo utilizzare un array con *n* posizioni per memorizzare una coda in grado di contenere *n-1* elementi

- Si utilizza un **buffer o array 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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON ARRAY (2/2)

- 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

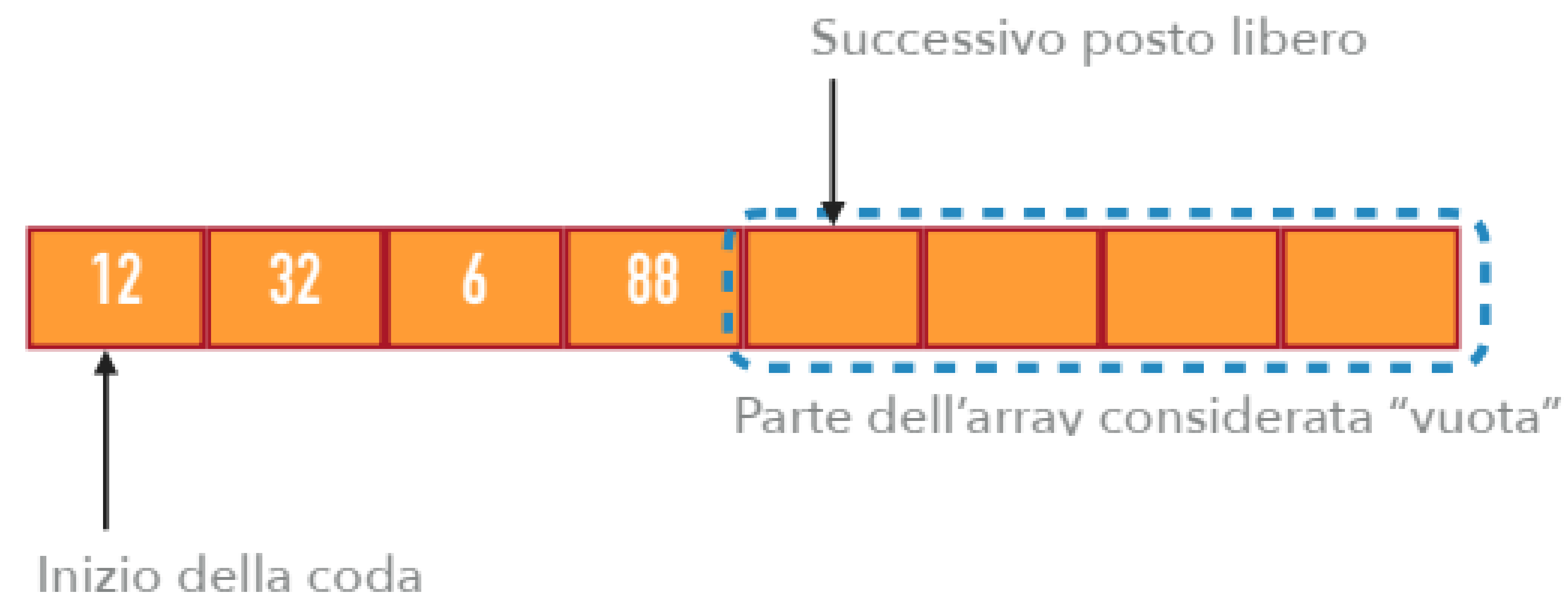
  Enqueue: spostamento in avanti del secondo indice

- Gli indici sono considerati «modulo *n*» *(una volta che arrivano a n-1, si torna a 0)*

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON ARRAY

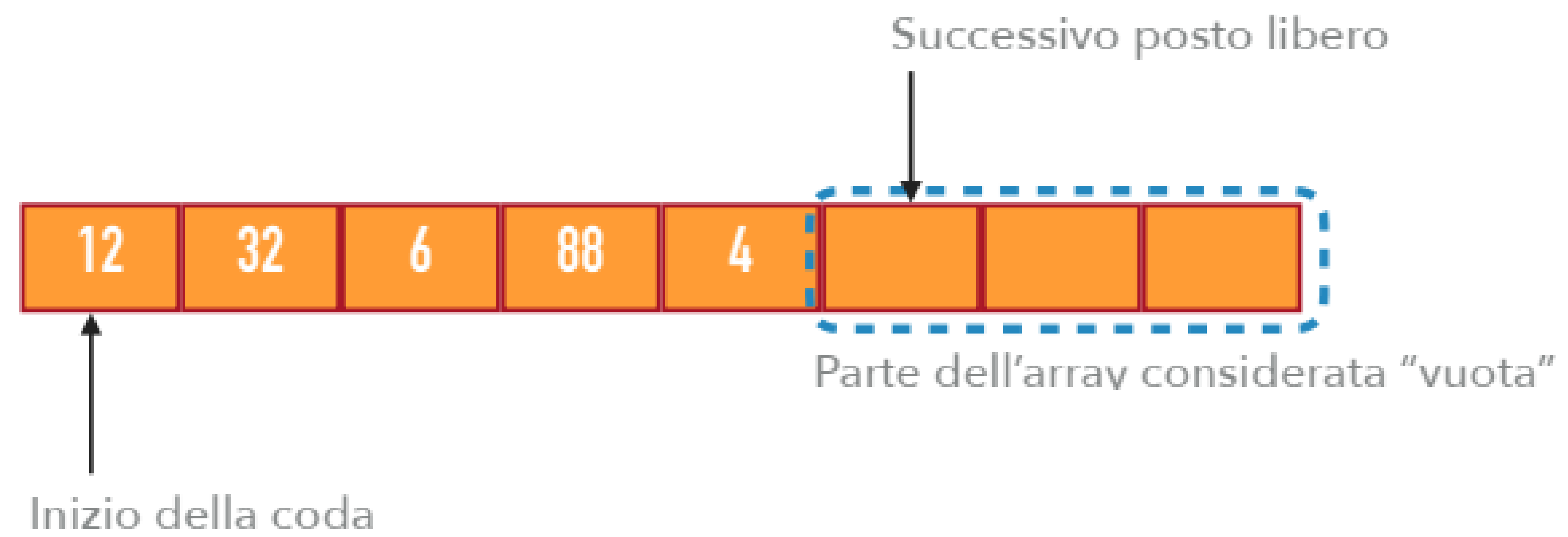
```
enqueue(4)
dequeue()
```



UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON ARRAY

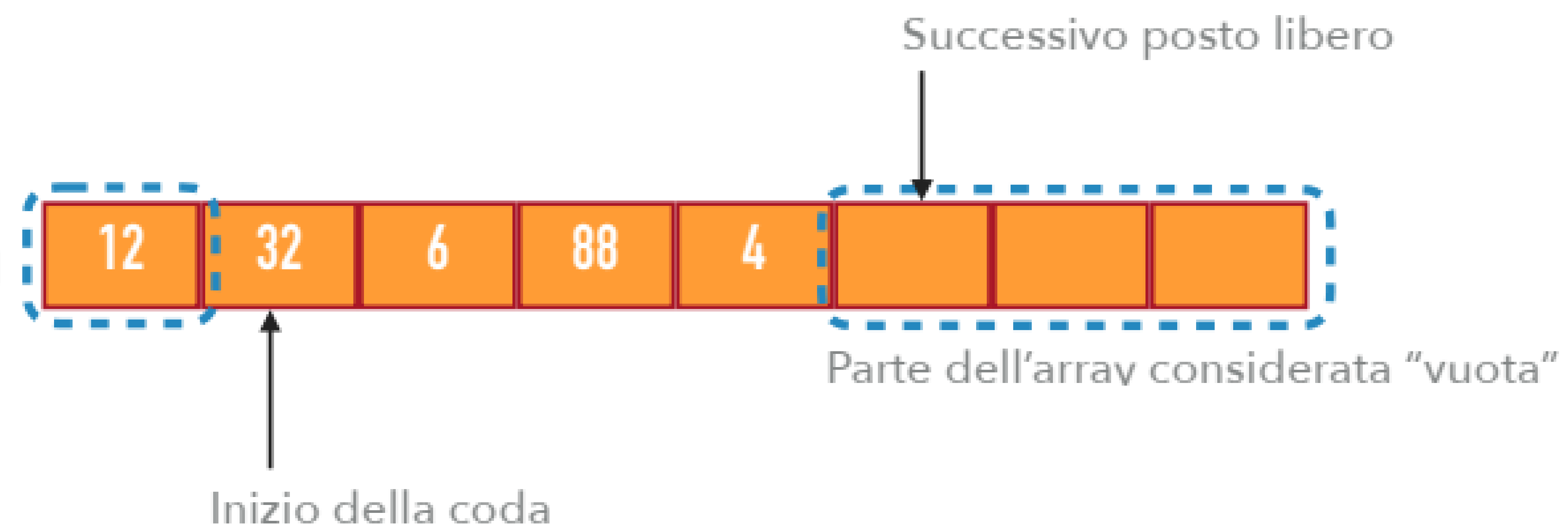
```
enqueue(4)
dequeue()
```



UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON ARRAY

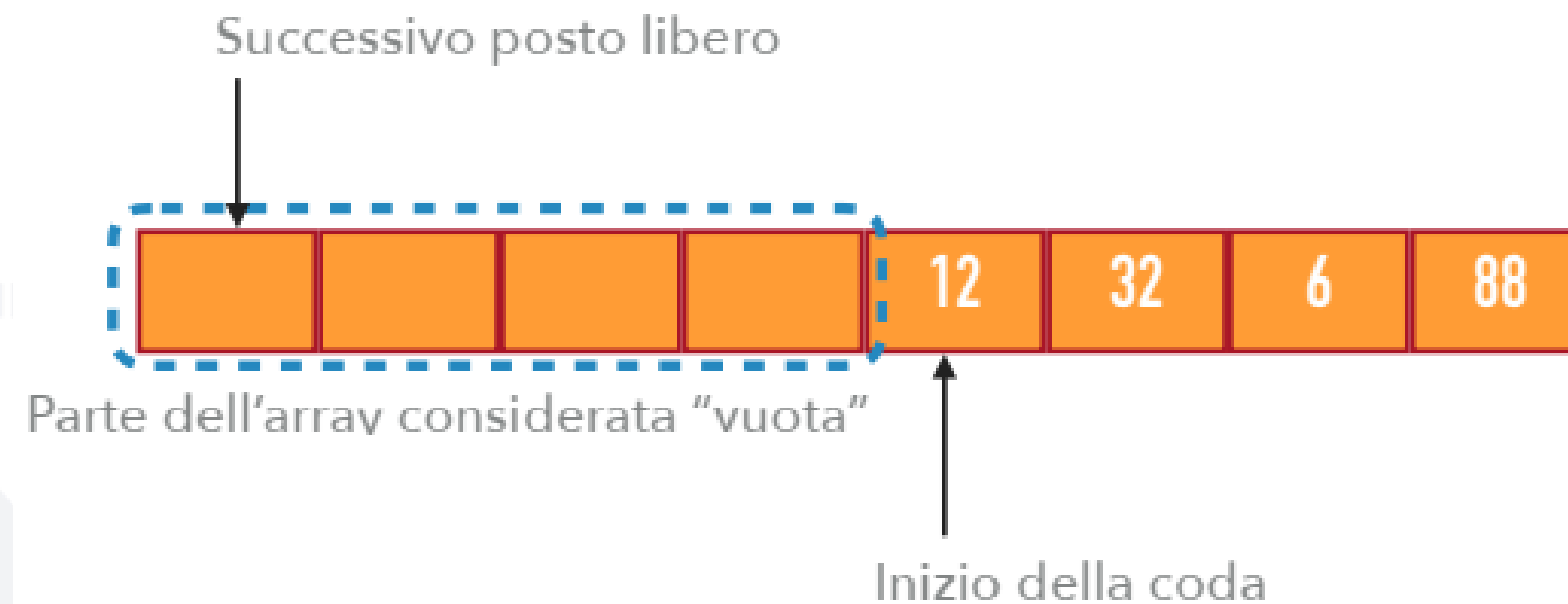
```
enqueue(4)
dequeue()
```



CODA: IMPLEMENTAZIONE CON ARRAY **#2**

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

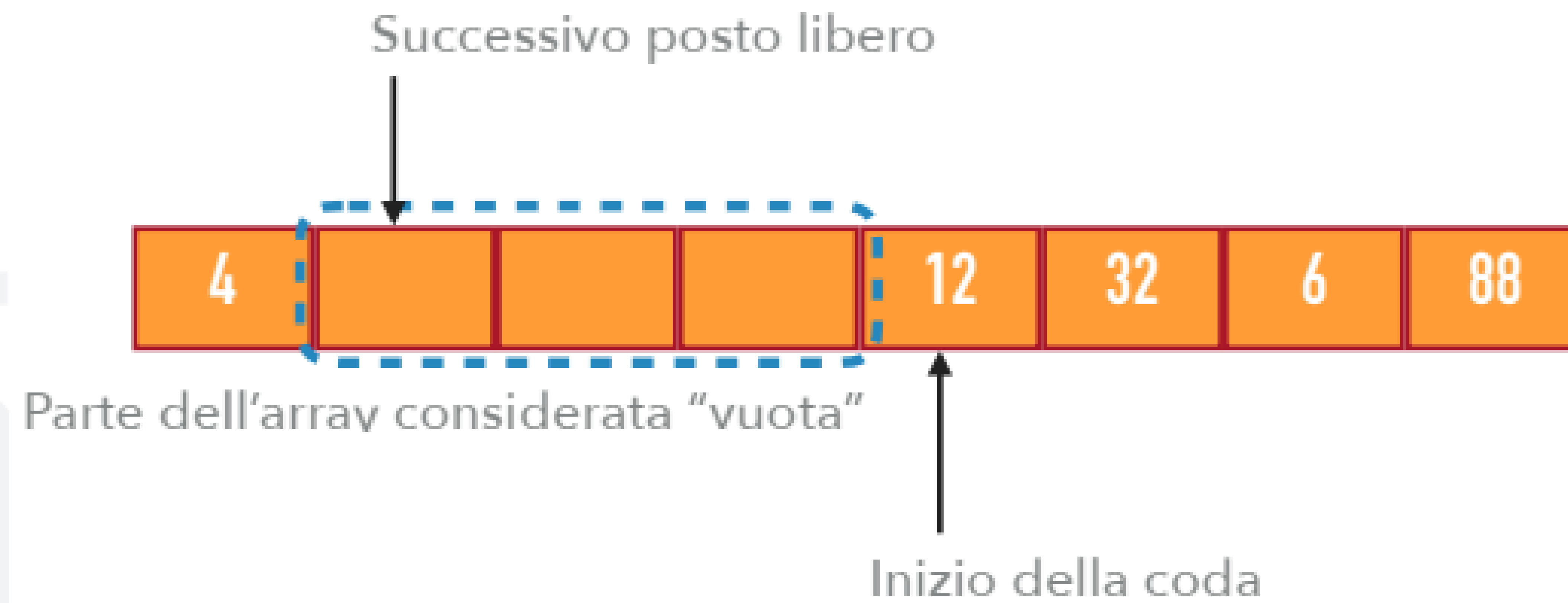
Le posizioni sono sempre considerate modulo *n*, quindi il successore della posizione *n* - 1 è la posizione 0



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

CODA: IMPLEMENTAZIONE CON ARRAY #2

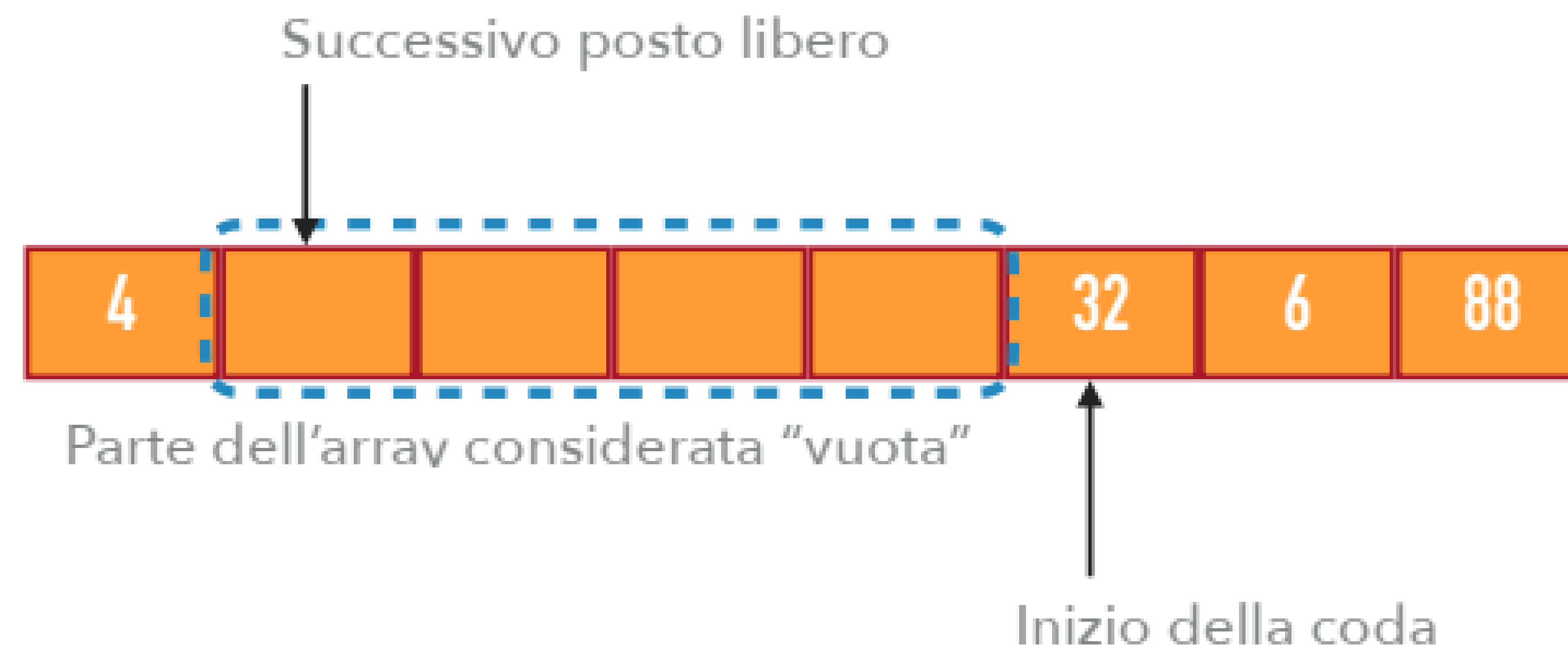
```
enqueue(4)
dequeue()
```



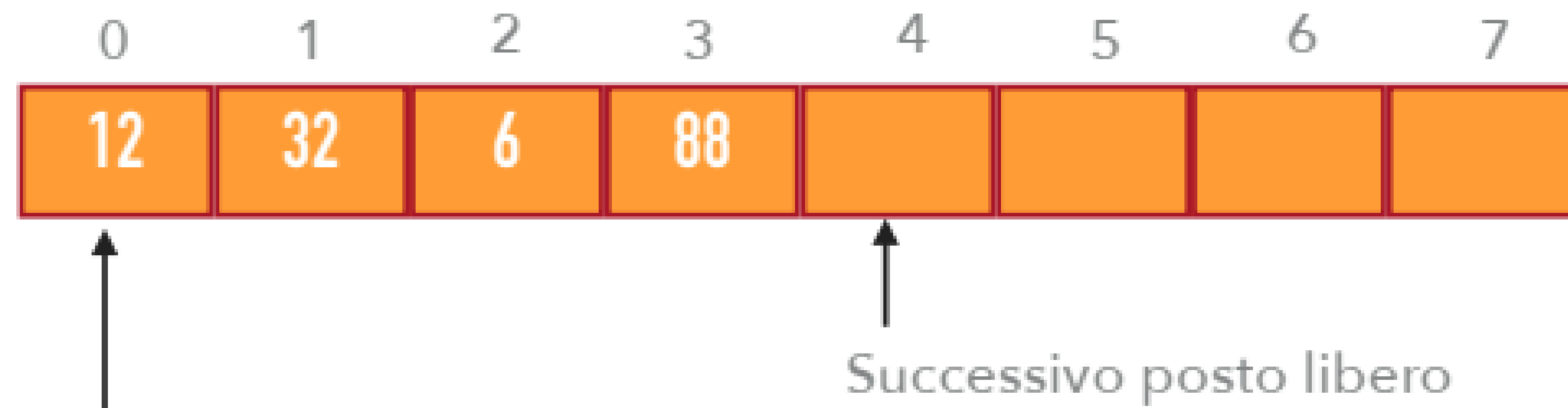
Gli elementi vengono rimossi nello stesso ordine con cui sono stati inseriti.

CODA: IMPLEMENTAZIONE CON ARRAY #2

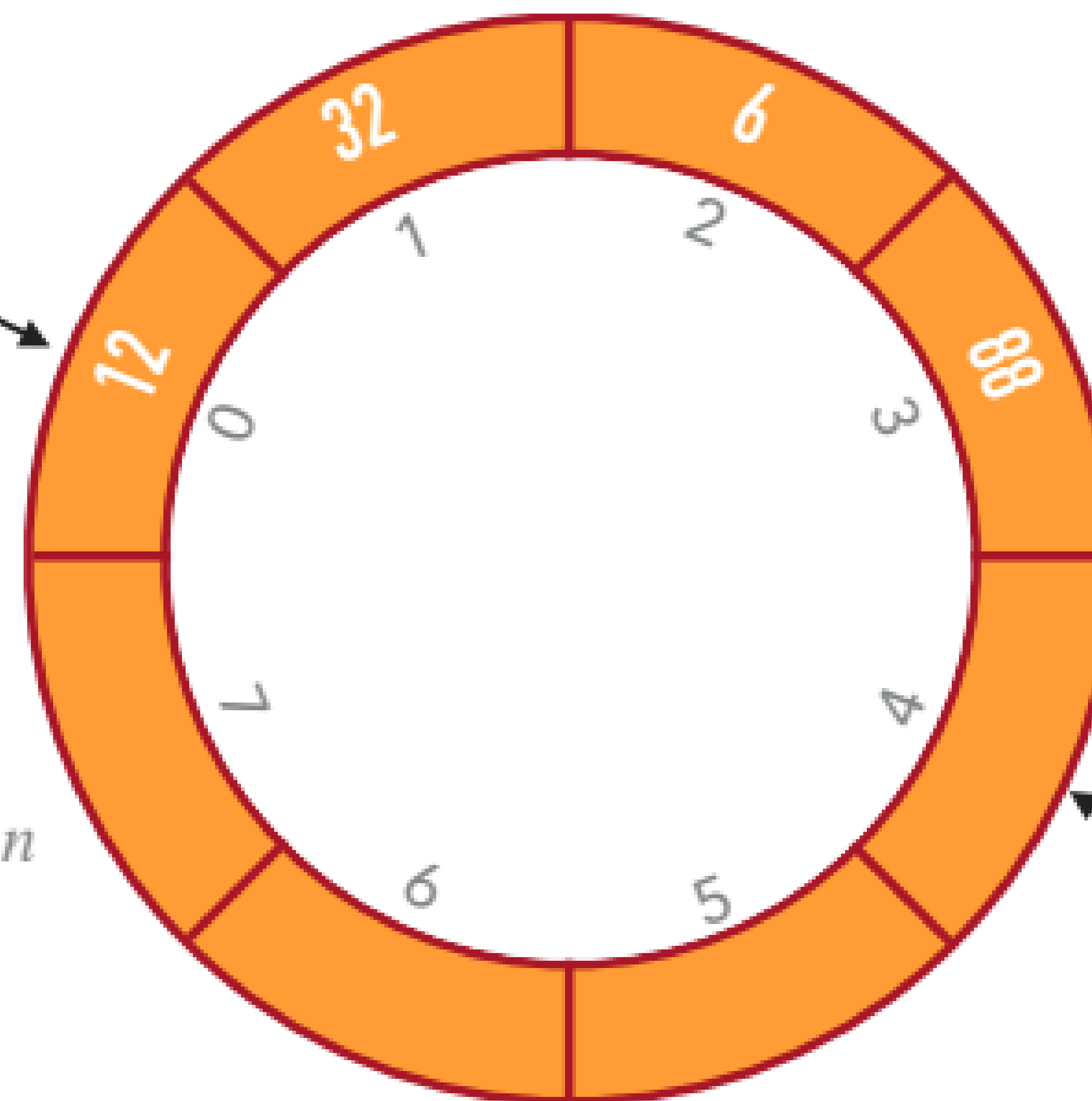
```
enqueue(4)
dequeue()
```



CODA: IMPLEMENTAZIONE CON ARRAY



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



Inizio della coda

Considerando le posizioni modulo *n* la posizione successiva alla 7 è la 0

Successivo posto libero

UNIVERSITÀ DEGLI STUDI DI TRIESTE

IMPLEMENTAZIONE CON ARRAY: PERCHE' *n-1* ELEMENTI?

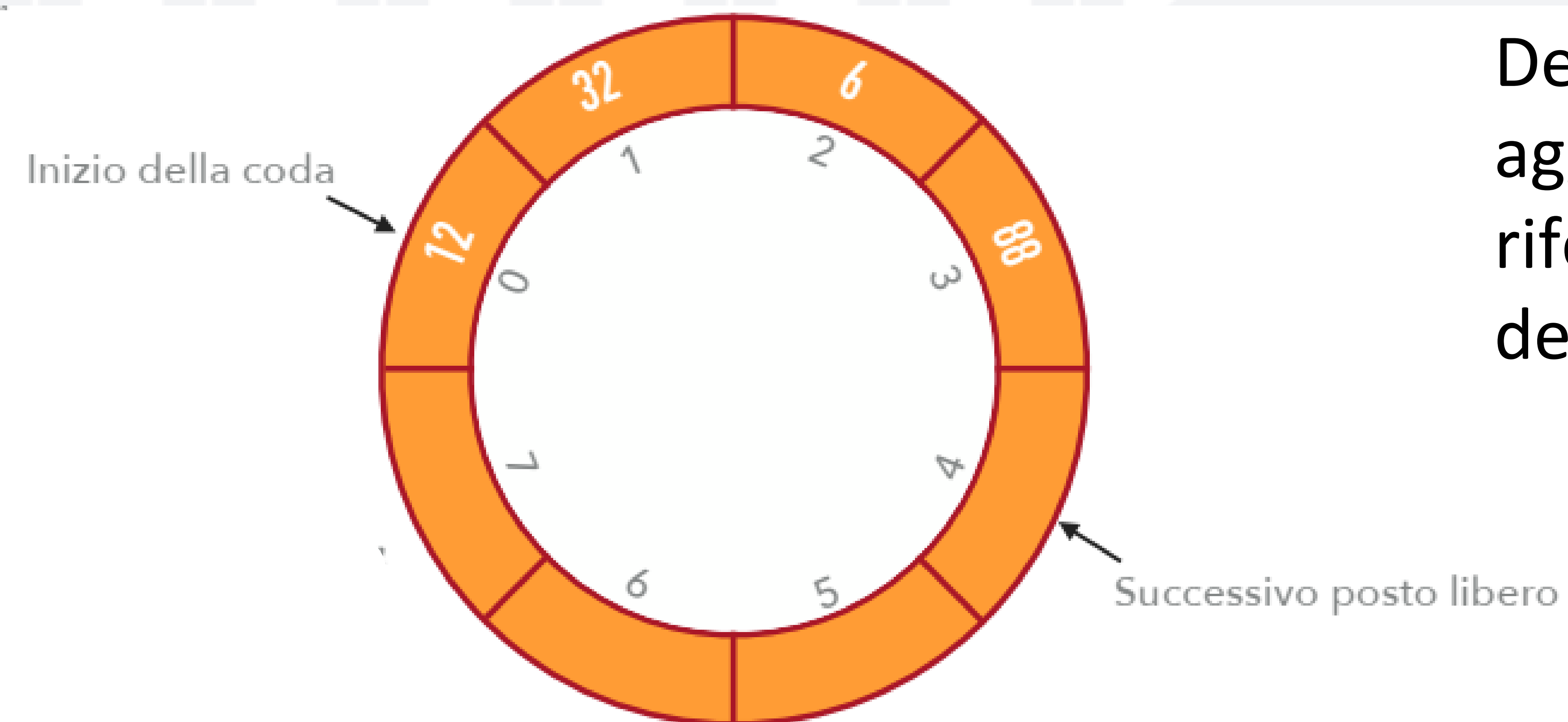
Se consentissimo di memorizzare *n* elementi (invece di *n-1*) non potremmo distinguere tra due casi:

*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 **O(1)**

- Se non avessimo i due indici ma tenessimo sempre l'inizio della coda in posizione 0 dovremmo copiare l'array ad ogni operazione di *dequeue*!



Dequeue richiede aggiornamento del riferimento all'inizio della coda

Potremmo 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

CODA: IMPLEMENTAZIONE CON STACK

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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON STACK

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

Rimozioni fatte nello stack A

Stack A

4

Stack B

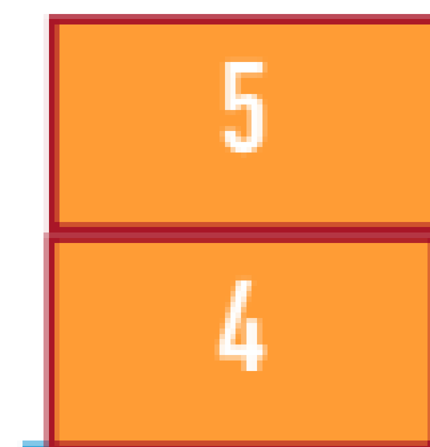
Inserimenti fatti nello stack B

CODA: 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

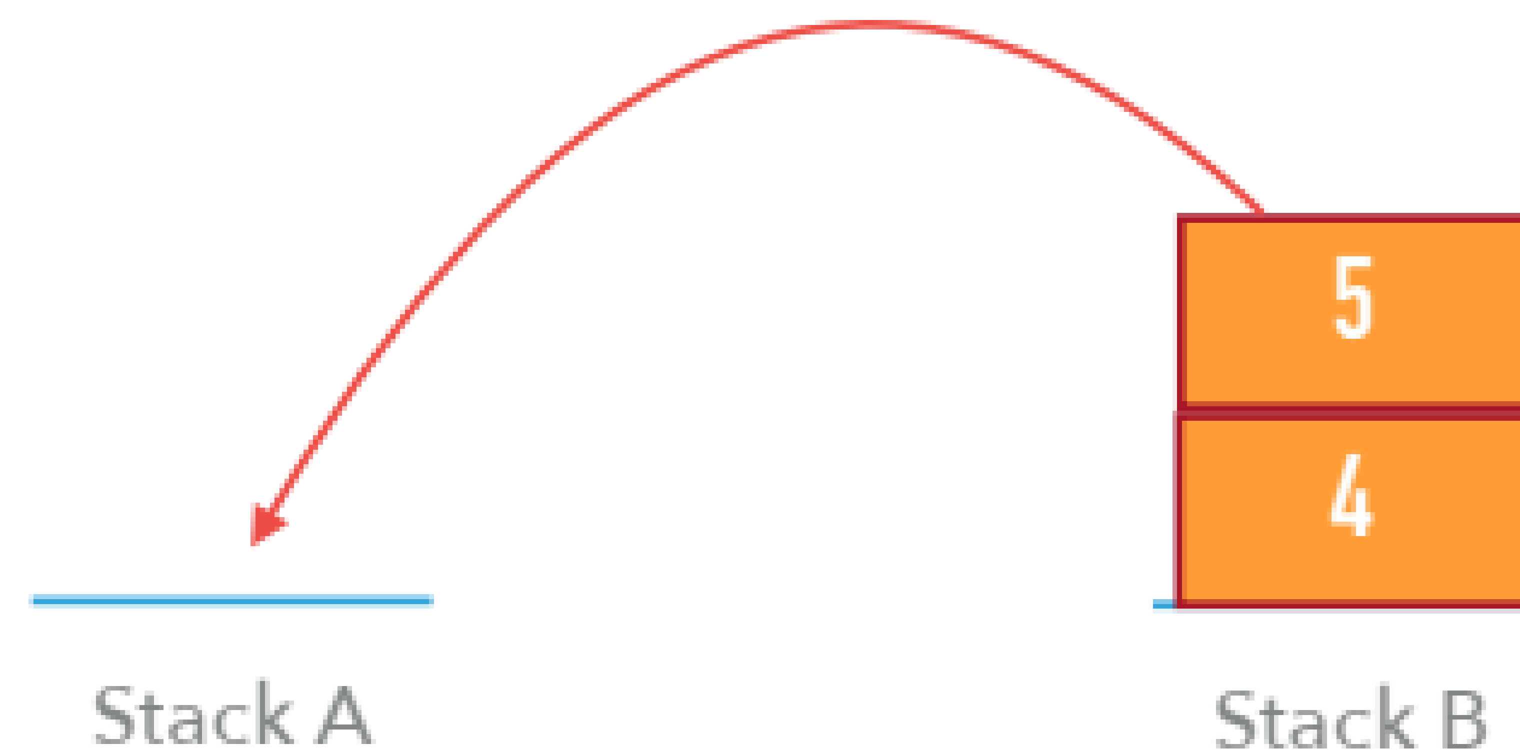
UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON STACK

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

NOTA. Questo «ribaltamento» consiste in una serie di pop() dallo Stack B che sono operazioni native e potenzialmente molto veloci da eseguire.

Lo stack A è vuoto! "ribaltiamo" B in A



Rimozioni fatte nello stack A

Inserimenti fatti nello stack B

CODA: IMPLEMENTAZIONE CON STACK

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

Lo stack A è vuoto! “ribaltiamo” B in A

A.push(B.pop())

Rimozioni fatte nello stack A

Stack A

Stack B

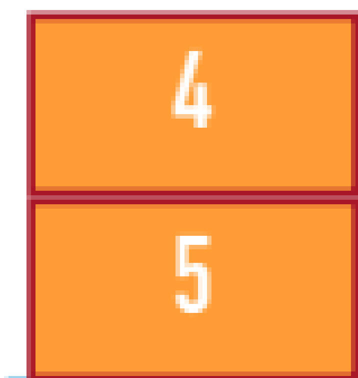
Inserimenti fatti nello stack B

CODA: IMPLEMENTAZIONE CON STACK

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

Ora il top dello stack A contiene

Rimozioni fatte nello stack A



Stack A

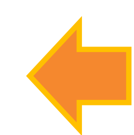
Stack B

Inserimenti fatti nello stack B

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON STACK

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



Valore ritornato: 4

Rimozioni fatte nello stack A

5

Stack A

Stack B

Inserimenti fatti nello stack B

UNIVERSITÀ DEGLI STUDI DI TRIESTE

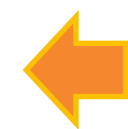
CODA: IMPLEMENTAZIONE CON STACK

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

```
B.push(3)
```

CODA: 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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CODA: IMPLEMENTAZIONE CON STACK

Qual è quindi la complessità per un'operazione di inserimento e rimozione di un elemento in una coda se la implementiamo con due stack?

- Inserimento (enqueue) richiede tempo costante **O(1)**

- Rimozione (dequeue) richiede tempo costante **O(1)** se lo stack A non è vuoto.

Altrimenti, devo prima «ribaltare» lo stack B in A. Caso peggiore: **O(n)**.

Nota. Se però si fanno **tante operazioni di enqueue** (inserimenti) ma **poche dequeue** (rimozioni), allora il costo O(n) sporadico del trasferimento Stack B → Stack A non pesa quasi niente.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

DOUBLE-ENDED QUEUE («DEQUE»)

Ibrido di stack e coda: si possono **inserire e rimuovere da entrambe le estremità**:

- Inserire elementi all'inizio o alla fine
- Rimuovere elementi dall'inizio o dalla fine

Come negli altri casi può essere implementata come lista concatenata oppure con un array utilizzato come buffer circolare.

In questo caso, **potrebbe essere comoda una lista concatenata doppia**:

- ogni nodo punta sia avanti che indietro
- inserire e togliere all'inizio o alla fine diventa **O(1)**

Applicazioni: browser: cronologia avanti/indietro, task scheduling (scegliere se processare subito o mettere in coda).

UNIVERSITÀ DEGLI STUDI DI TRIESTE

QUIZ FINALE

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

Materiale per la lezione

- Cormen et al. **CAP. 3.10**

Prossima lezione: 28 ***aprile, h.14:00, aula 4C***