UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

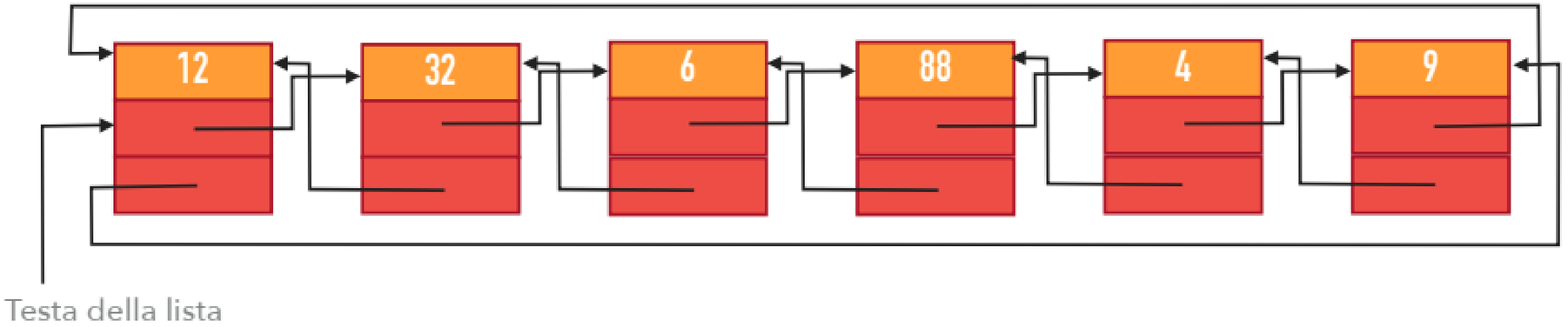
MODULO 2: Pila o Stack

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

Trieste, 23 aprile 2026

RECAP



Lista concatenata doppia e circolare

OPERAZIONE	ARRAY	LISTA CONC. SINGOLA	LISTA CONC. DOPPIA
ACCEDERE AD UN ELEMENTO	$O(1)$	$O(n)$	$O(n)$
RICERCA	$O(n)$	$O(n)$	$O(n)$
INSERIMENTO	$O(n)$	$O(n)$ ma $O(1)$ se in testa	$O(n)$ ma $O(1)$ se in testa
RIMOZIONE	$O(n)$	$O(n)$ ma $O(1)$ se in testa	$O(n)$ ma $O(1)$ se in testa o se abbiamo già il puntatore al nodo da cancellare

STRUTTURE DATI CONCRETE E ASTRATTE

- Fino ad ora abbiamo visto come organizzare in memoria i dati e come eseguire delle operazioni su di essi
- Abbiamo quindi visto strutture dati **concrete**
- Possiamo focalizzarci invece sulle operazioni che dobbiamo consentire (definendo quindi strutture dati **astratte**)
- E poi chiederci quali strutture dati concrete possiamo utilizzare per implementarle

CONCRETO vs ASTRATTO

ASTRATTO

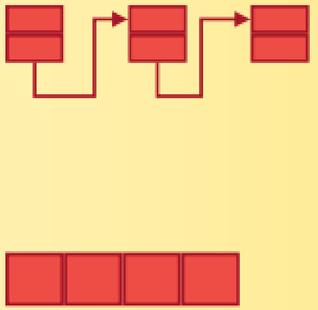
Definisce il comportamento

```
inserisci_in_testa()
rimuovi_testa()
```

```
inserisci_in_testa()
rimuovi_coda()
```

CONCRETO

Specifica come sono organizzati i dati



Ma la distinzione non è sempre netta, è un continuo!

LO STACK, ovvero LA PILA (1/2)

- Lo stack (o pila) è una struttura dati astratta con due principali operazioni:
  - **Push**. Inserisce un elemento nello stack.
  - **Pop**. Rimuove l'ultimo elemento inserito (e non ancora rimosso)

- Chiamato anche **LIFO (Last In, First Out)**

- Le operazioni sono le stesse che si possono fare su una pila di piatti

Stack (pila) di piatti

LO STACK, ovvero LA PILA (2/2)

- La **cima (o top)** dello stack è l'ultimo elemento inserito e non ancora rimosso

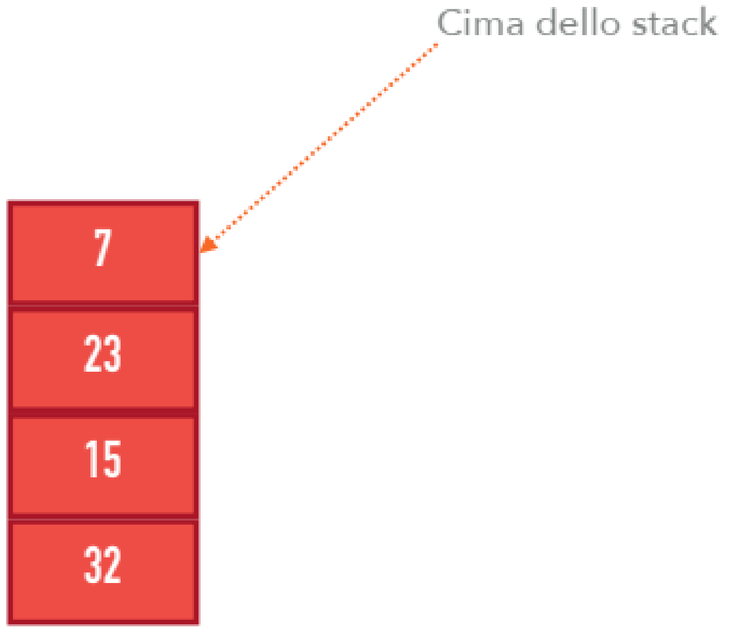
- Altre operazioni possibili (ma non essenziali):
  - **Peek**. Per vedere il valore sulla cima dello stack senza rimuoverlo.
  - **Empty**. Per chiedere se lo stack è vuoto

Stack (pila) di piatti

STACK: RAPPRESENTAZIONE GRAFICA

Operazioni da eseguire:

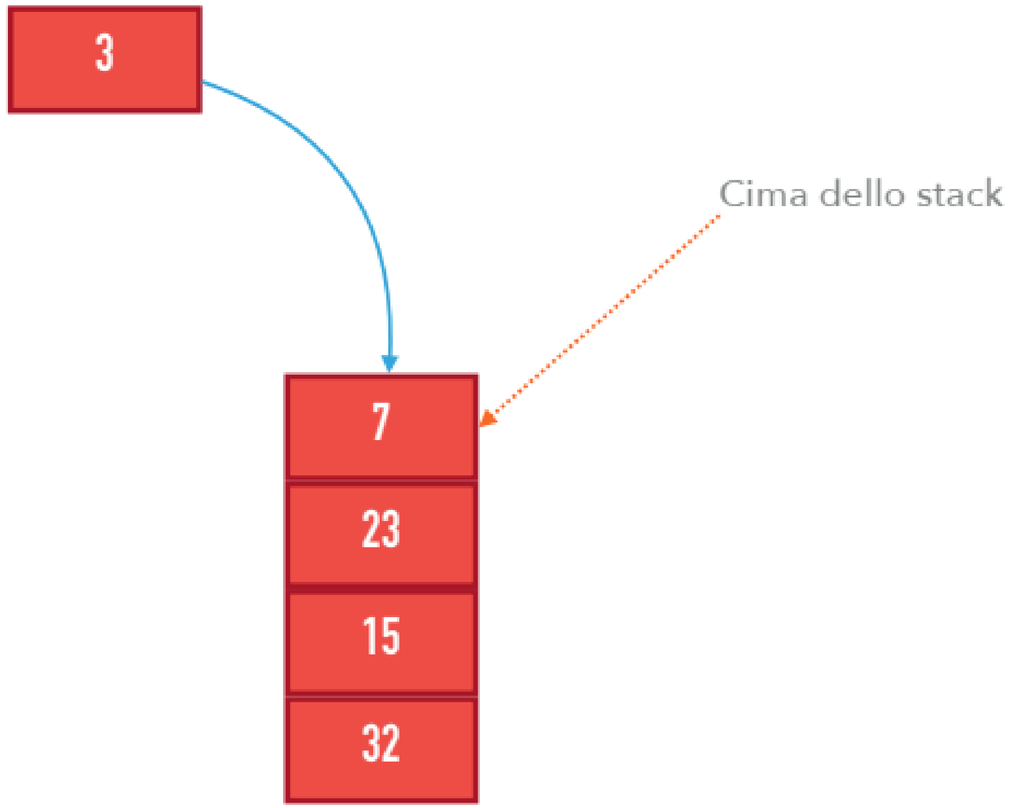
```
push(3)
pop()
push(5)
pop()
pop()
```



STACK: RAPPRESENTAZIONE GRAFICA

Operazioni da eseguire:

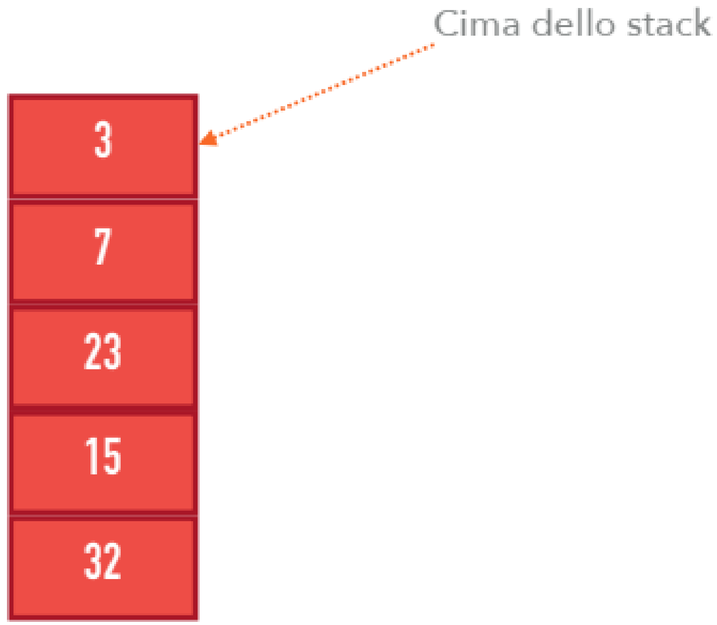
```
push(3)
pop()
push(5)
pop()
pop()
```



STACK: RAPPRESENTAZIONE GRAFICA

Operazioni da eseguire:

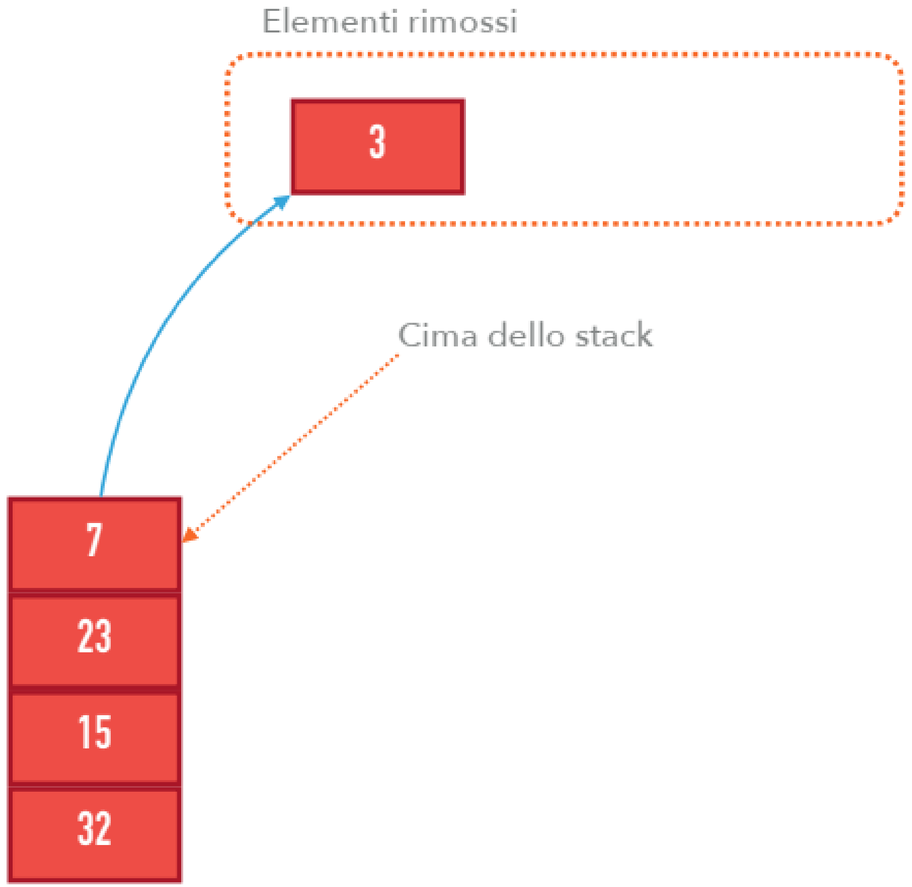
```
push(3)
pop()
push(5)
pop()
pop()
```



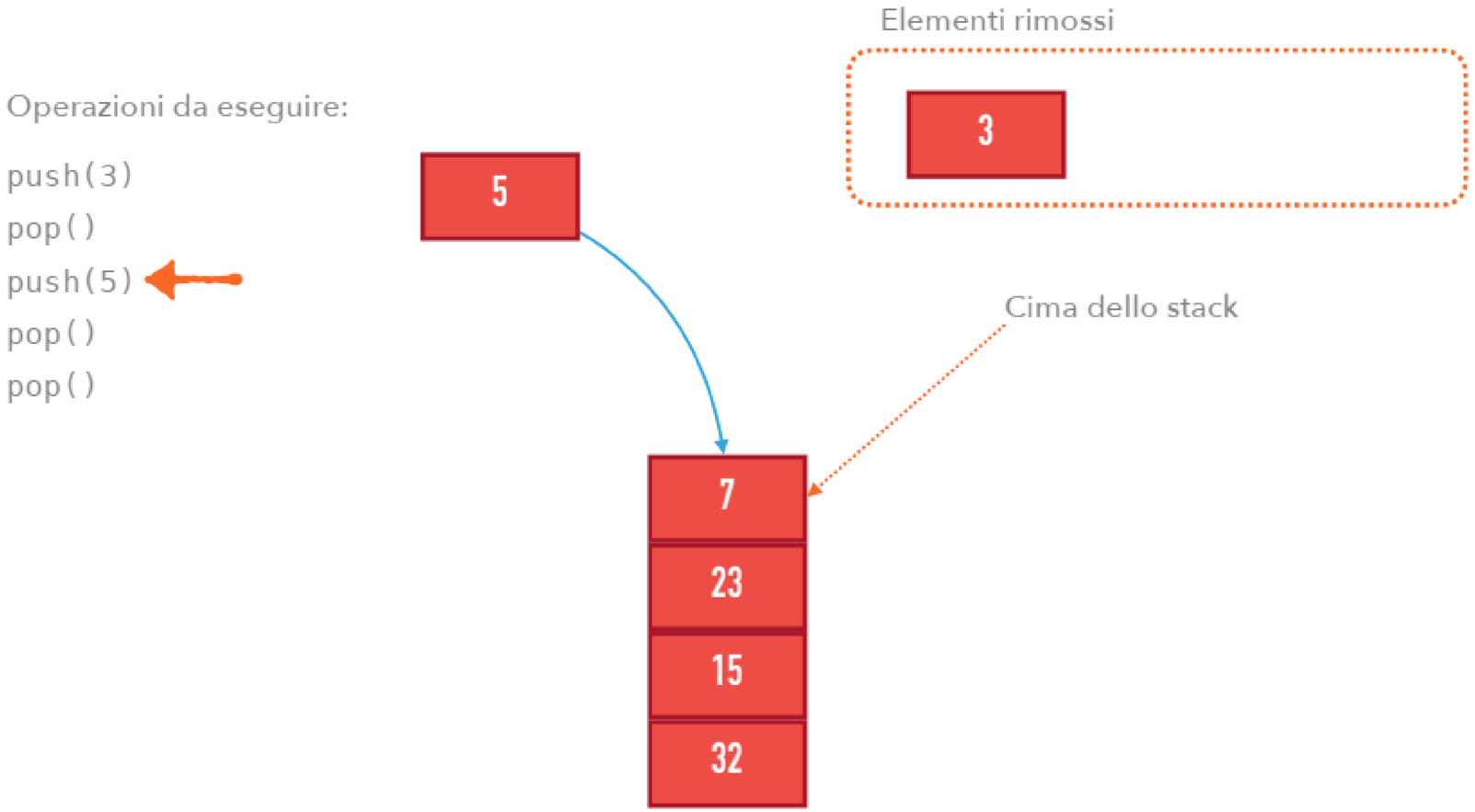
STACK: RAPPRESENTAZIONE GRAFICA

Operazioni da eseguire:

```
push(3)
pop()
push(5)
pop()
pop()
```



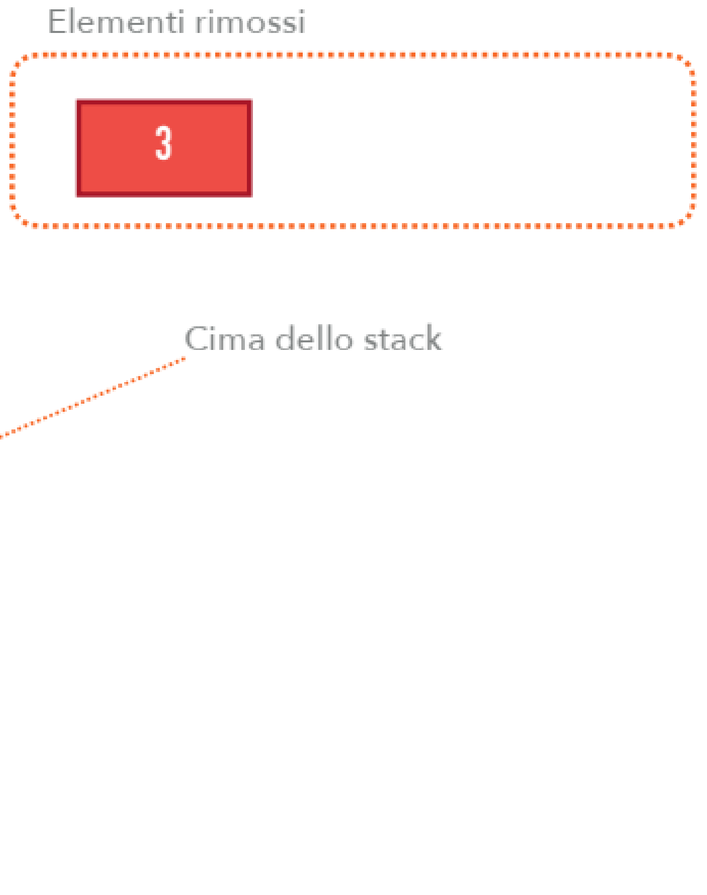
STACK: RAPPRESENTAZIONE GRAFICA



STACK: RAPPRESENTAZIONE GRAFICA

Operazioni da eseguire:

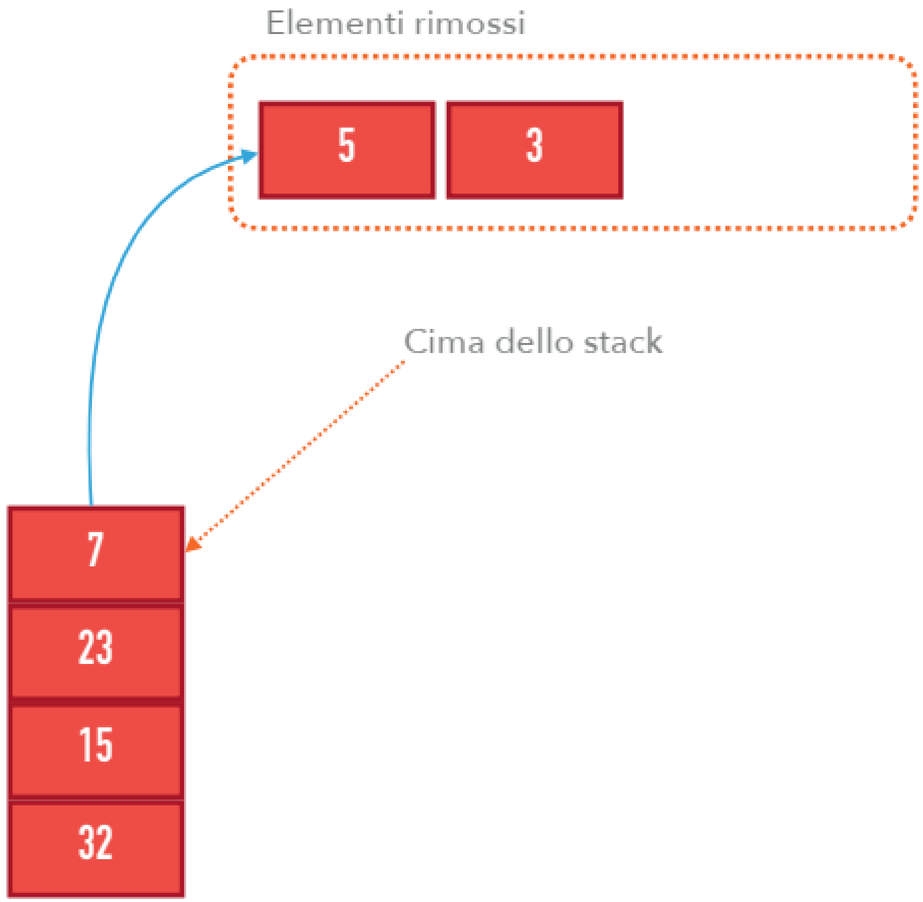
```
push(3)
pop()
push(5)
pop()
pop()
```



STACK: RAPPRESENTAZIONE GRAFICA

Operazioni da eseguire:

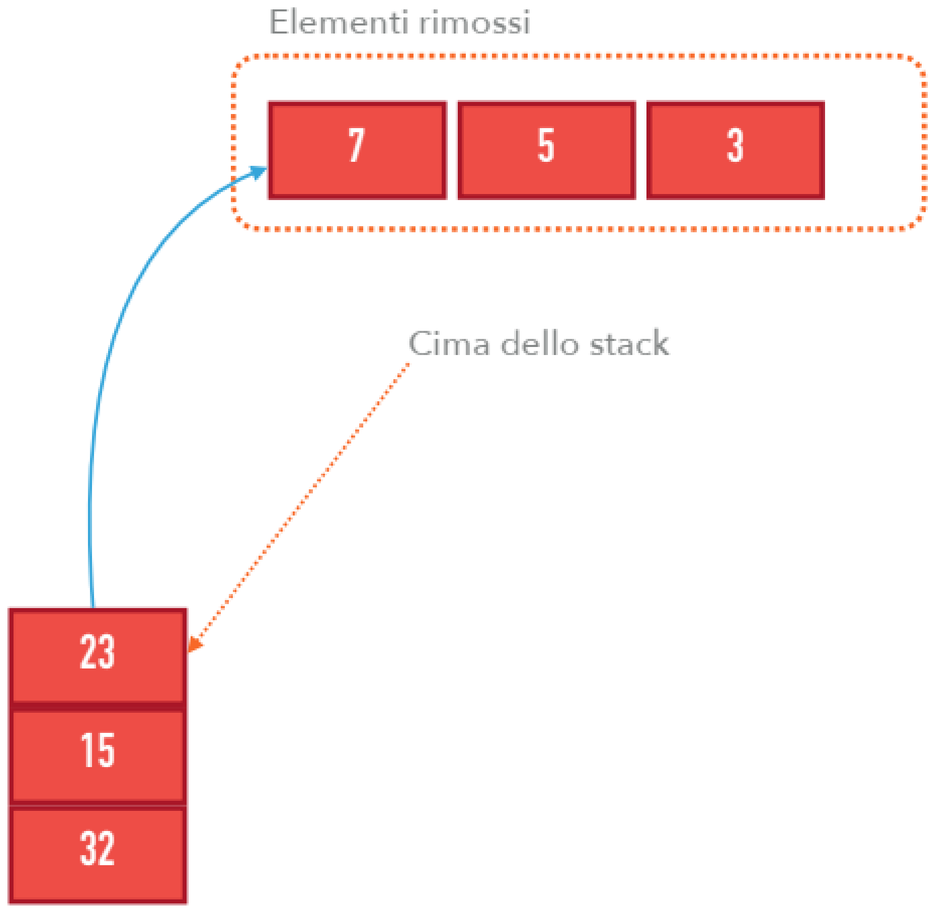
push(3)
pop()
push(5)
pop()
pop()



STACK: RAPPRESENTAZIONE GRAFICA

Operazioni da eseguire:

```
push(3)
pop()
push(5)
pop()
pop()
```



QUIZ

Cosa è contenuto in uno stack dopo la seguente sequenza di operazioni?

```
push(4)
push(5)
pop()
push(6)
push(7)
pop()
```

A) 4 6

B) 6 7

C) 5 7

D) 4 5 6 7

UTILITA' DELLO STACK

- Diversi altri algoritmi ne fanno uso al loro interno:
  - Chiamate a funzioni/procedure
  - Trovare l'uscita in un labirinto (esempio di ricerca in un grafo)

- Possono essere usati per valutare espressioni in RPN (reverse Polish notation — notazione polacca inversa)

ESEMPIO (1/9)

Notazione usata in alcune calcolatrici scientifiche

(2 + 3) × 4 diventa 2 3 + 4 ×

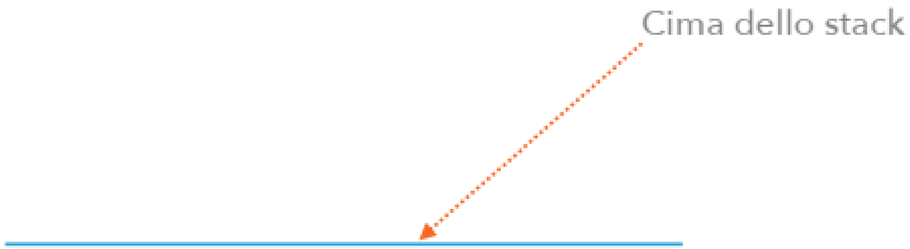
L’operazione si mette dopo gli operandi.

- Come si interpreta?
  - Se si incontra un numero questo viene messo in uno stack
  - Se si incontra un’operazione, gli operandi sono rimossi dallo stack, l’operazione viene effettuata ed il risultato inserito nello stack

ESEMPIO (2/9)

Operazioni da eseguire:

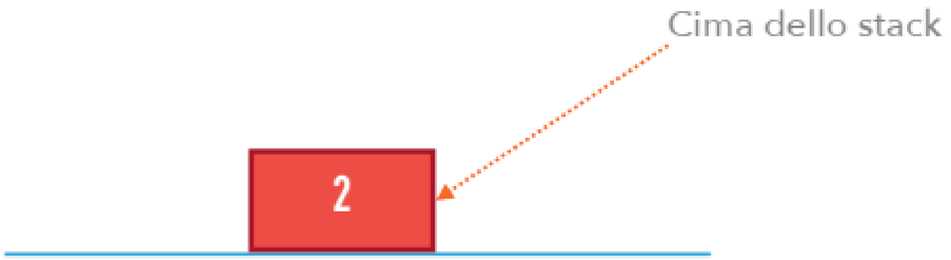
2 3 + 4 ×



ESEMPIO (3/9)

Operazioni da eseguire:

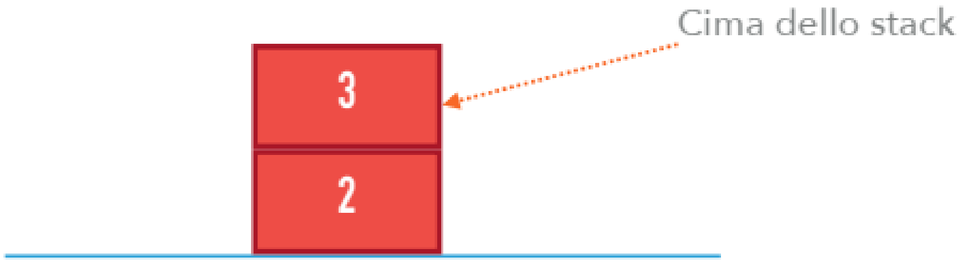
2 3 + 4 ×



ESEMPIO (4/9)

Operazioni da eseguire:

2 3 + 4 ×

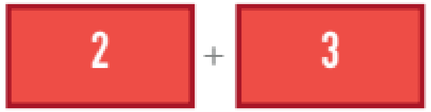


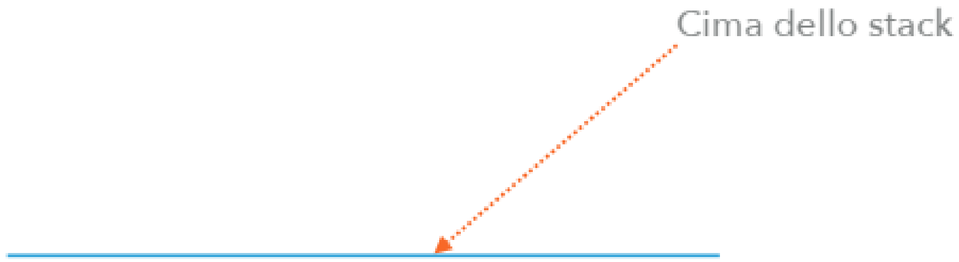
ESEMPIO (5/9)

Operazioni da eseguire:

2 3 + 4 ×

Eseguiamo l'operazione usando come operandi i primi due elementi in cima allo stack



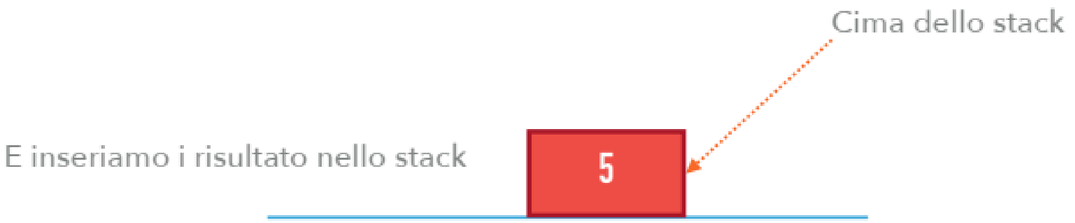


UNIVERSITÀ DEGLI STUDI DI TRIESTE

ESEMPIO (6/9)

Operazioni da eseguire:

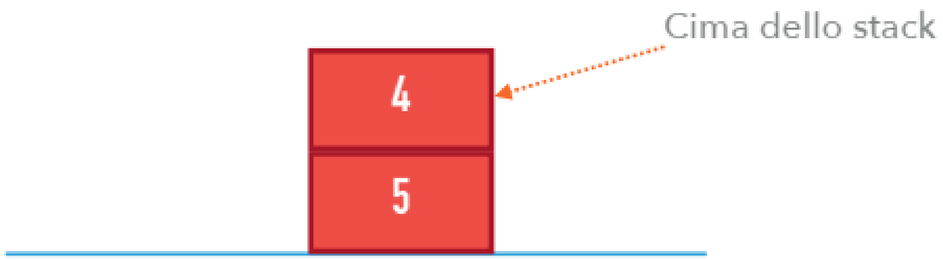
2 3 + 4 ×



ESEMPIO (7/9)

Operazioni da eseguire:

2 3 + 4 ×

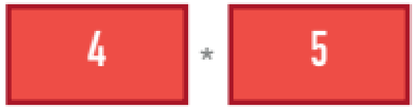


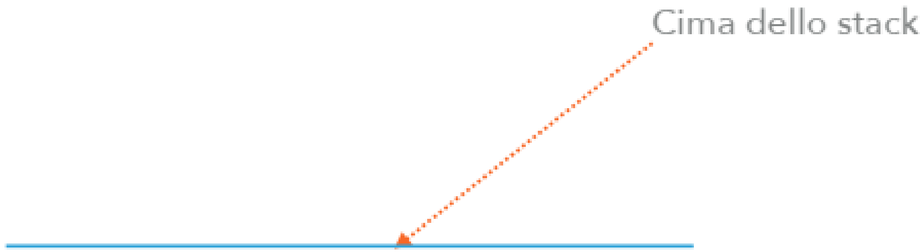
ESEMPIO (8/9)

Operazioni da eseguire:

2 3 + 4 ×

Eseguiamo l'operazione usando come operandi i primi due elementi in cima allo stack

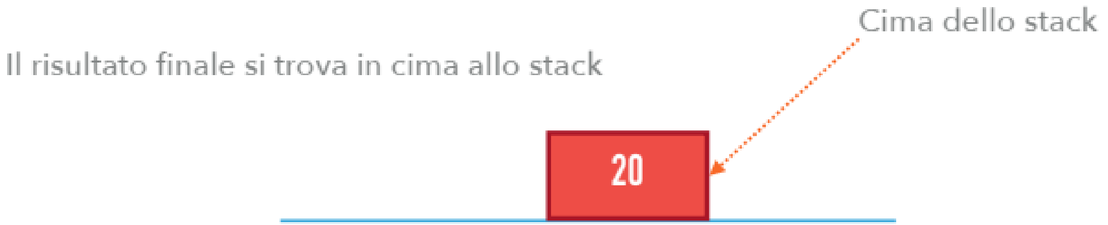




ESEMPIO (9/9)

Operazioni da eseguire:

2 3 + 4 ×



QUIZ: RPN

Qual è il risultato della valutazione della seguente espressione in RPN?

```
3 2 * 3 2 + +
```

A) 13

B) espressione non valida

C) 11

D) asksjdfg

STACK: IMPLEMENTAZIONE

- Dato che uno stack definisce solo le operazioni che si possono effettuare dobbiamo decidere come implementare lo stack

- Vediamo due modi:
  - Stack implementato **tramite liste concatenate singole**
  - Stack implementato **tramite array**

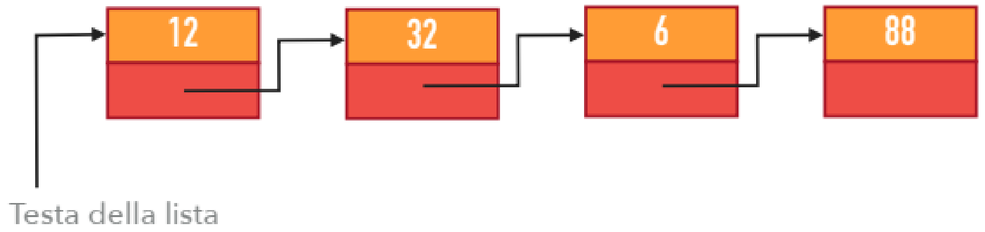
STACK: IMPLEMENTAZIONE CON LISTE CONCATENATE

- L’implementazione con liste concatenate richiede due operazioni:
  - Push: inserimento in testa alla lista
  - Pop: rimozione dalla testa della lista

- Altre operazioni sono comunque facili da implementare:
  - Peek: valore della testa della lista
  - Empty: controllo che la testa non sia None

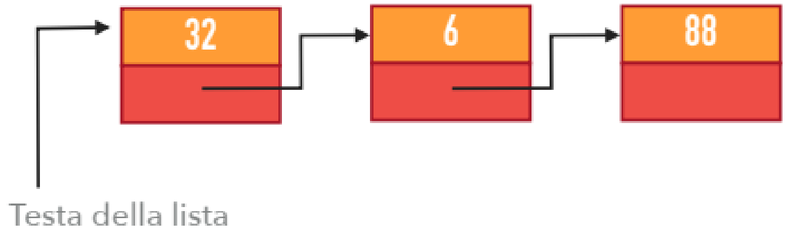
STACK: IMPLEMENTAZIONE CON LISTE CONCATENATE

```
pop()
push(3)
```



STACK: IMPLEMENTAZIONE CON LISTE CONCATENATE

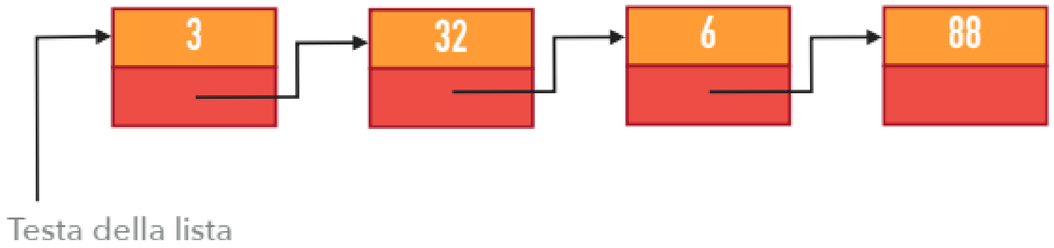
```
pop()
push(3)
```



Valore ritornato: 12

STACK: IMPLEMENTAZIONE CON LISTE CONCATENATE

```
pop()
push(3)
```



STACK: IMPLEMENTAZIONE CON LISTE CONCATENATE

- La **complessità** di inserimento e rimozione è pari a quella di inserimento in testa e rimozione del primo elemento

OPERAZIONE	ARRAY	LISTA CONC. SINGOLA	LISTA CONC. DOPPIA
ACCEDERE AD UN ELEMENTO	$O(1)$	$O(n)$	$O(n)$
RICERCA	$O(n)$	$O(n)$	$O(n)$
INSERIMENTO	$O(n)$	$O(n)$ ma $O(1)$ se in testa	$O(n)$ ma $O(1)$ se in testa
RIMOZIONE	$O(n)$	$O(n)$ ma $O(1)$ se in testa	$O(n)$ ma $O(1)$ se in testa o se abbiamo già il puntatore al nodo da cancellare

Push: *O*(1)

Pop: *O*(1)

- **Nota bene**: la complessità delle operazioni dipende dall'implementazione!
- Non possiamo dire quale sia il costo computazionale senza dire come sono state implementate!

IMPLEMENTAZIONE CON LISTE CONCATENATE: QUIZ

Perché non abbiamo usato una lista concatenata **doppia** per implementare lo stack?

A) Potevamo usarla, avrebbe migliorato la complessità

B) Potevamo usarla, ma non avrebbe migliorato la complessità

C) Non era utilizzabile

D) Era utilizzabile solo se circolare

STACK: IMPLEMENTAZIONE CON **ARRAY**

- Dato che un array non “cresce”, gli array possono **implementare facilmente solo stack di dimensione limitata**, ma *copiando* l’array nel caso si inserisca un elemento in un array “pieno” è possibile ottenere stack di dimensioni arbitrarie.

Spoiler Lab!!

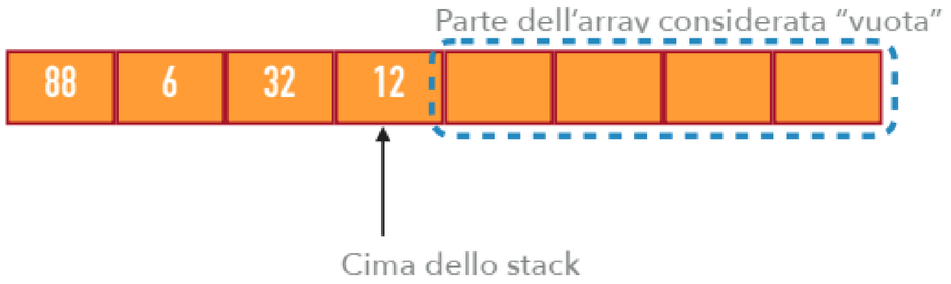
- Uno **stack di *n* elementi** occupa le posizioni da *0* a *n-1* dell’array con la testa dello stack in posizione *n-1*. L’**array ha dimensione K** ≥ n.

- **Inserire un elemento** significa copiarlo in posizione *n* e ricordarsi che la testa dello stack è ora in posizione *n*

- **La rimozione** è simile: ritorniamo l’elemento in posizione *n-1* ricordando che la testa dello stack è ora in posizione *n-2*

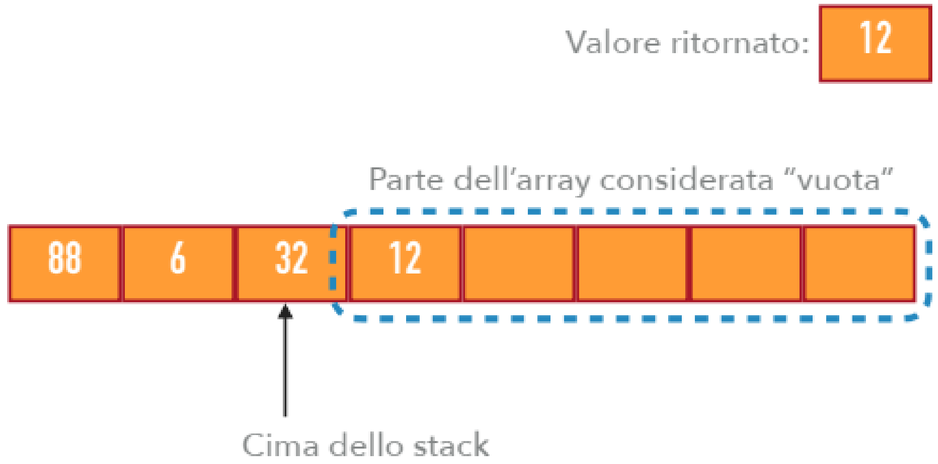
STACK: IMPLEMENTAZIONE CON ARRAY

```
pop()
push(3)
```



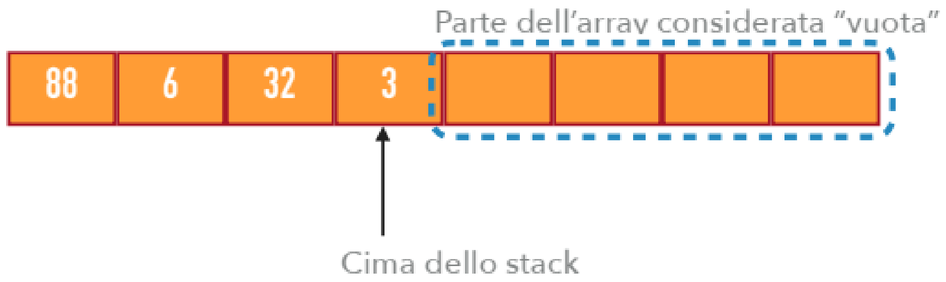
STACK: IMPLEMENTAZIONE CON ARRAY

```
pop()
push(3)
```



STACK: IMPLEMENTAZIONE CON ARRAY

```
pop()
push(3)
```



Se la cima dello stack fosse stata l'ultima posizione dell'array avremmo dovuto copiare tutti in un array più grande prima di inserire il nuovo elemento!

STACK: IMPLEMENTAZIONE CON ARRAY

- La complessità della rimozione è data dal decrementare una variabile e copiare un valore: ***O*(1)**
- L'inserimento, nel caso ci sia spazio disponibile è rapido richiedendo un numero costante di passi…
- … ma nel caso peggiore dobbiamo copiare l'intero array, ottenendo un tempo che è ***O*(*n*)**

Quindi l'implementazione di uno stack tramite array è:

Push: *O*(1)

Pop: *O*(n)

Materiale per la lezione

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