

# Pile e code (*stacks and queues*)

**Liceo G.B. Brocchi - Bassano del Grappa (VI)**  
**Liceo Scientifico - opzione scienze applicate**  
Giovanni Mazzocchin

# Stack - implementazione *array-based*

- Realizziamo un **ADT** (**A**bstract **D**ata **T**ype) che rappresenta uno **stack** (pila)
- La nostra prima implementazione sarà basata su array
- Uno stack è una struttura dati **LIFO** (**L**ast-**I**n, **F**irst-**O**ut, *l'ultimo ad entrare è il primo ad uscire*), la cui interfaccia è costituita da 2 operazioni:
  - **push(item)**: aggiunge item in cima alla pila
  - **pop()**: rimuove l'elemento dalla cima della pila
- Per realizzare uno stack è necessario tenere traccia dell'indice della cima (**top**, spesso chiamato **stack pointer**). Nell'implementazione *array-based*, **top** sarà semplicemente un indice intero. Analizziamo graficamente il comportamento delle operazioni push e pop

# Stack - implementazione *array-based*



top

**Attenzione:**  
in queste implementazioni, le  
operazioni di inserimento e  
cancellazione non allocano/  
deallocono memoria, ma  
riassegnano soltanto degli indici

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 0 (EMPTY)**

**la pila ha memoria allocata per 8  
elementi, ma finora nessun  
elemento è stato aggiunto**

# Stack - implementazione *array-based*



push(9)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 1**

**la *size* aumenta ad ogni push  
andata a buon fine**

# Stack - implementazione *array-based*



push(15)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 2**

**la *size* aumenta ad ogni push  
andata a buon fine**

# Stack - implementazione *array-based*



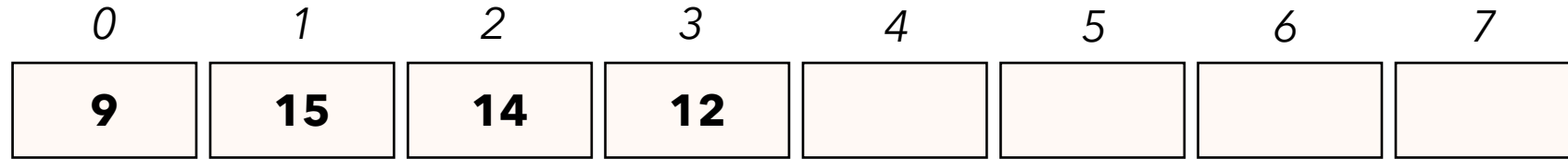
top

push(14)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 3**

**la *size* aumenta ad ogni push  
andata a buon fine**

# Stack - implementazione *array-based*



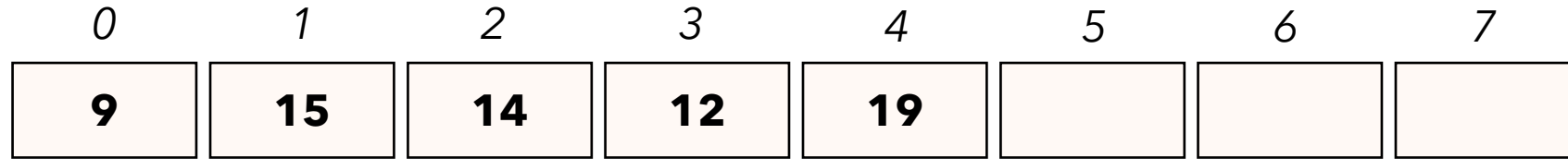
top

push(12)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 4**

**la *size* aumenta ad ogni push  
andata a buon fine**

# Stack - implementazione *array-based*



top

push(19)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 5**

**la *size* aumenta ad ogni push  
andata a buon fine**



# Stack - implementazione *array-based*

0	1	2	3	4	5	6	7
9	15	14	12	19	23		

top

push(23)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 6**

**la *size* aumenta ad ogni push  
andata a buon fine**

# Stack - implementazione *array-based*

0	1	2	3	4	5	6	7
9	15	14	12	19	23	27	

top

push(27)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 7**

**la *size* aumenta ad ogni push  
andata a buon fine**

# Stack - implementazione *array-based*

0	1	2	3	4	5	6	7
9	15	14	12	19	23	27	24

top

push(24)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 8 (FULL)**

**la pila è in stato FULL quando  
STACK\_SIZE è uguale a  
STACK\_CAPACITY**

# Stack - implementazione *array-based*

0	1	2	3	4	5	6	7
9	15	14	12	19	23	27	24

top

push(30)

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 8 (FULL)**

**una push su un pila piena è una  
*no-operation*, i.e. non succede  
niente**

# Stack - implementazione *array-based*

0	1	2	3	4	5	6	7
9	15	14	12	19	23	27	

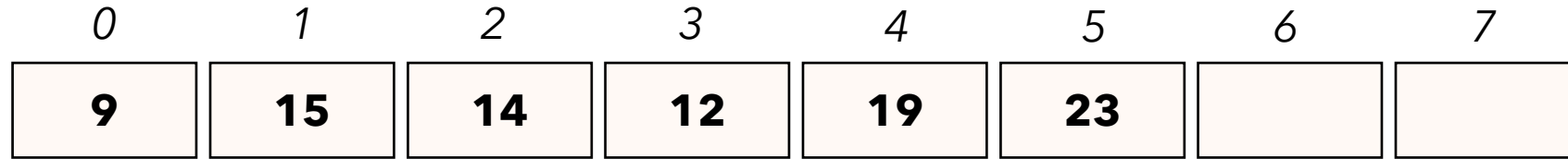
top

pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 7**

**la *size* decresce ad ogni pop  
andata a buon fine**

# Stack - implementazione *array-based*



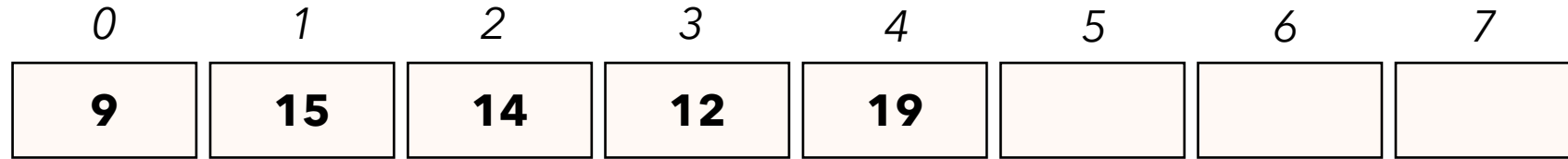
top

pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 6**

**la *size* decresce ad ogni pop  
andata a buon fine**

# Stack - implementazione *array-based*



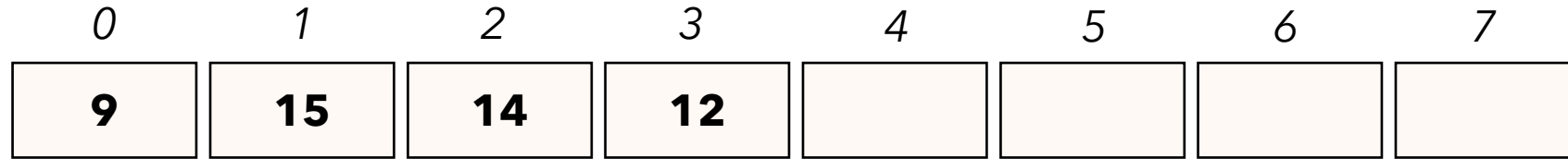
top

pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 5**

**la *size* decresce ad ogni pop  
andata a buon fine**

# Stack - implementazione *array-based*



top

pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 4**

**la *size* decresce ad ogni pop  
andata a buon fine**



# Stack - implementazione *array-based*



top

pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 3**

**la *size* decresce ad ogni pop  
andata a buon fine**

# Stack - implementazione *array-based*



top

pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 2**

**la *size* decresce ad ogni pop  
andata a buon fine**

# Stack - implementazione *array-based*



pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 1**

**la *size* decresce ad ogni pop  
andata a buon fine**

# Stack - implementazione *array-based*



top

pop()

**STACK\_CAPACITY: 8**  
**STACK\_SIZE: 0 (EMPTY)**

**una pop su un pila vuota è una  
*no-operation*, i.e. non succede  
niente**

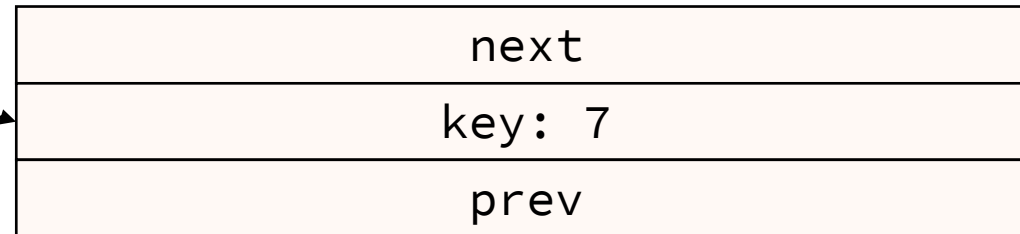
# Stack - implementazione *linked-list-based*

- Per implementare uno stack **dinamico**, la cui dimensione può variare in base alle necessità del programma a runtime, possiamo utilizzare una lista concatenata
- Su questa lista sarà possibile inserire e rimuovere solo dalla coda
- Nelle liste concatenate viste nelle scorse lezioni era invece possibile aggiungere e rimuovere un nodo in qualsiasi posizione
- La lista dovrà essere **doppiamente concatenata (doubly-linked list)**
- È necessario tenere traccia della cima dello stack con un puntatore, detto **stack pointer**

# Stack - implementazione *linked-list-based*

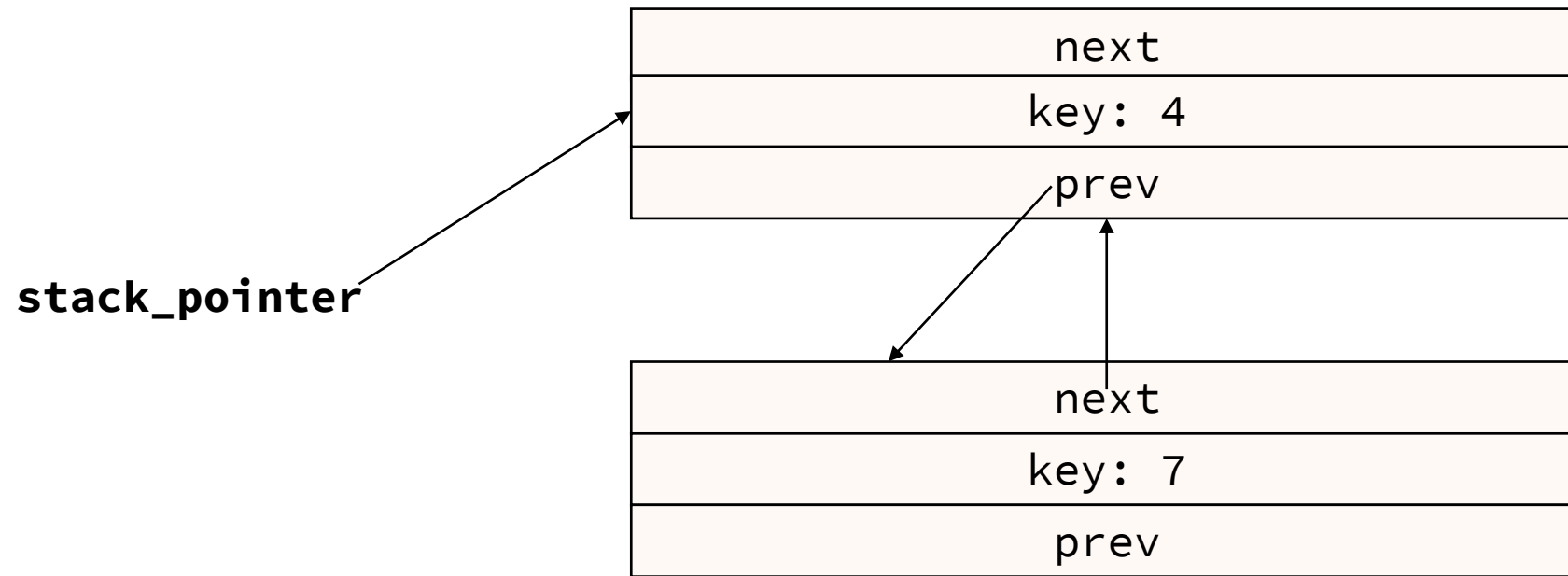
**push(7)**

**stack\_pointer**



# Stack - implementazione *linked-list-based*

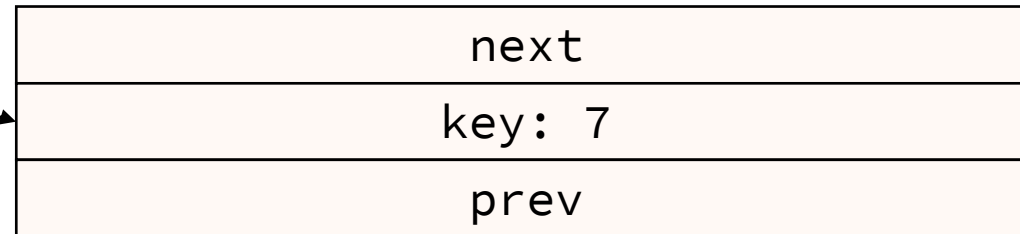
**push(4)**



# Stack - implementazione *linked-list-based*

**pop()**

**stack\_pointer**





# Queue - implementazione *array-based*

- La **queue** (coda) è un Abstract Data Type caratterizzato da due operazioni:
  - **enqueue(item)**: permette di aggiungere un elemento in fondo alla coda
  - **dequeue()**: permette di rimuovere l'elemento in testa alla coda
- Non sono ammesse altre operazioni: è una struttura dati **FIFO** (***F**irst-**I**n, **F**irst-**O**ut - il primo che entra è il primo ad uscire*)
- Facciamo alcuni esempi su una coda di interi

# Queue - implementazione *array-based*

**enqueue(8)**



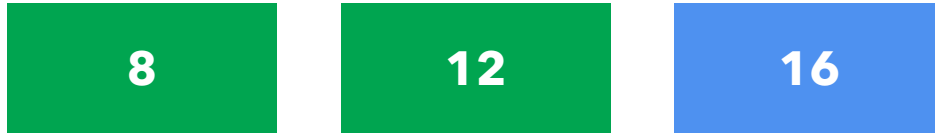
# Queue - implementazione *array-based*

**enqueue(12)**



# Queue - implementazione *array-based*

**enqueue(16)**



# Queue - implementazione *array-based*

**enqueue(11)**



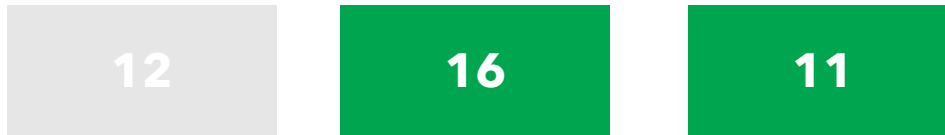
# Queue - implementazione *array-based*

**dequeue()**



# Queue - implementazione *array-based*

**dequeue()**



# Queue - implementazione *array-based*



tail

head

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 0 (EMPTY)**

**la coda ha memoria allocata per 8  
elementi, ma finora nessun  
elemento è stato aggiunto**



# Queue - implementazione *array-based*



head

tail

enqueue(9)

**QUEUE\_CAPACITY: 8**

**QUEUE\_SIZE: 1**

**la *size* aumenta ad ogni enqueue  
andata a buon fine**

# Queue - implementazione *array-based*



head      tail

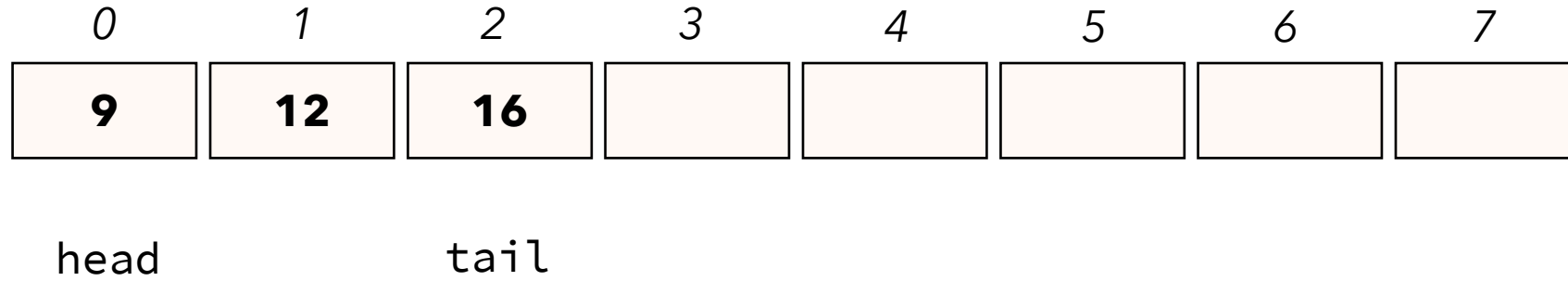
enqueue(12)

**QUEUE\_CAPACITY: 8**

**QUEUE\_SIZE: 2**

**la *size* aumenta ad ogni enqueue  
andata a buon fine**

# Queue - implementazione *array-based*



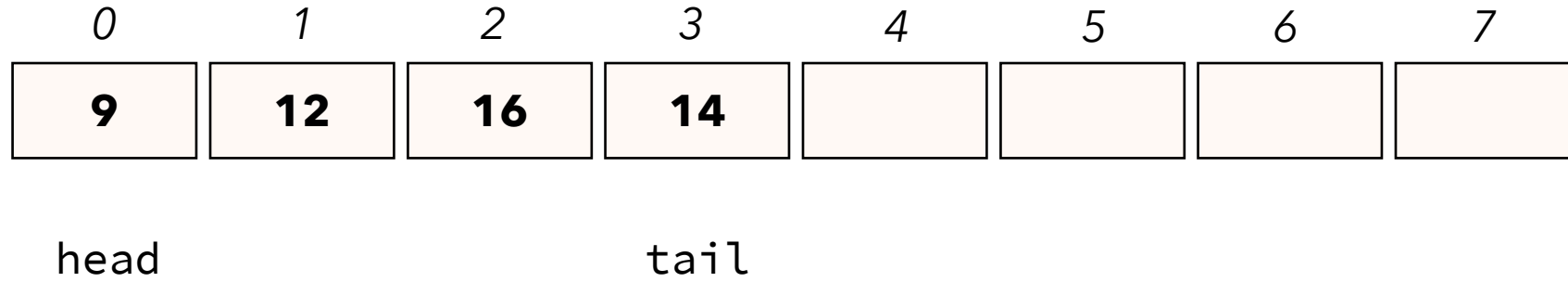
enqueue(16)

**QUEUE\_CAPACITY: 8**

**QUEUE\_SIZE: 3**

**la *size* aumenta ad ogni enqueue  
andata a buon fine**

# Queue - implementazione *array-based*



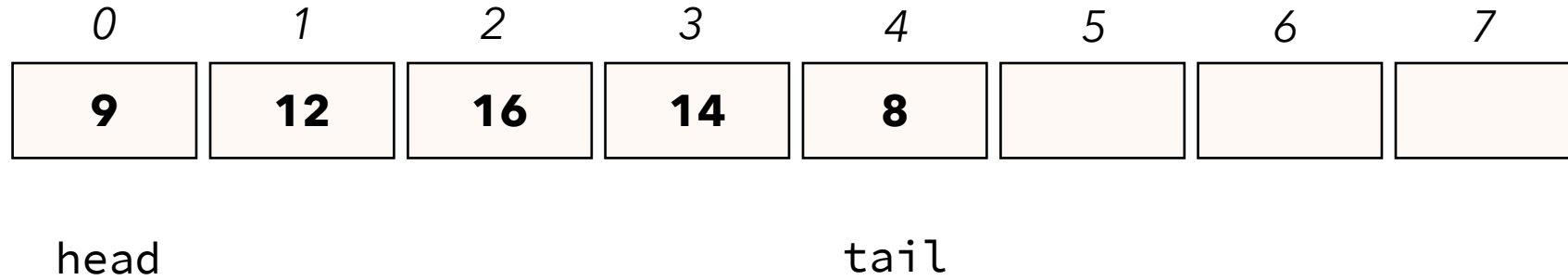
enqueue(14)

**QUEUE\_CAPACITY: 8**

**QUEUE\_SIZE: 4**

**la *size* aumenta ad ogni enqueue  
andata a buon fine**

# Queue - implementazione *array-based*

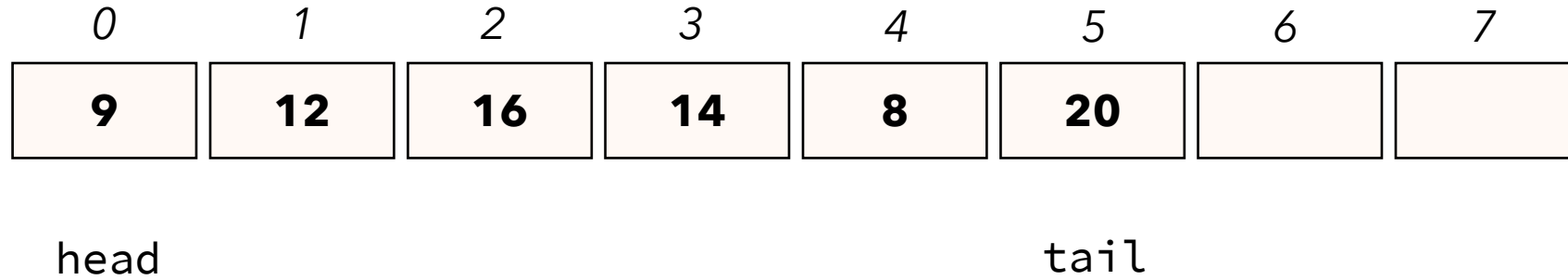


enqueue(8)

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 5**

**la *size* aumenta ad ogni enqueue andata a buon fine**

# Queue - implementazione *array-based*



enqueue(20)

**QUEUE\_CAPACITY: 8**

**QUEUE\_SIZE: 6**

**la *size* aumenta ad ogni enqueue  
andata a buon fine**

# Queue - implementazione *array-based*

0	1	2	3	4	5	6	7
9	12	16	14	8	20	23	

head

tail

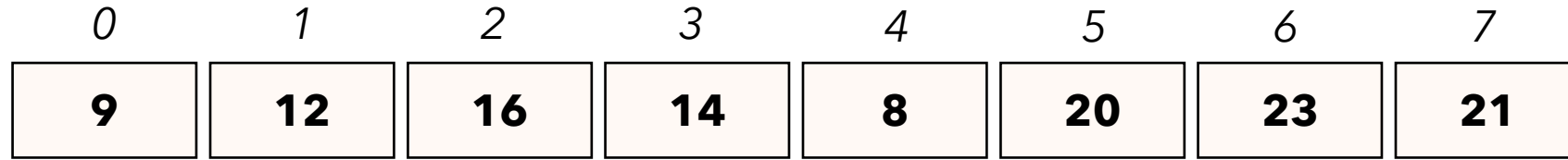
enqueue(23)

**QUEUE\_CAPACITY: 8**

**QUEUE\_SIZE: 7**

**la *size* aumenta ad ogni enqueue  
andata a buon fine**

# Queue - implementazione *array-based*



head

tail

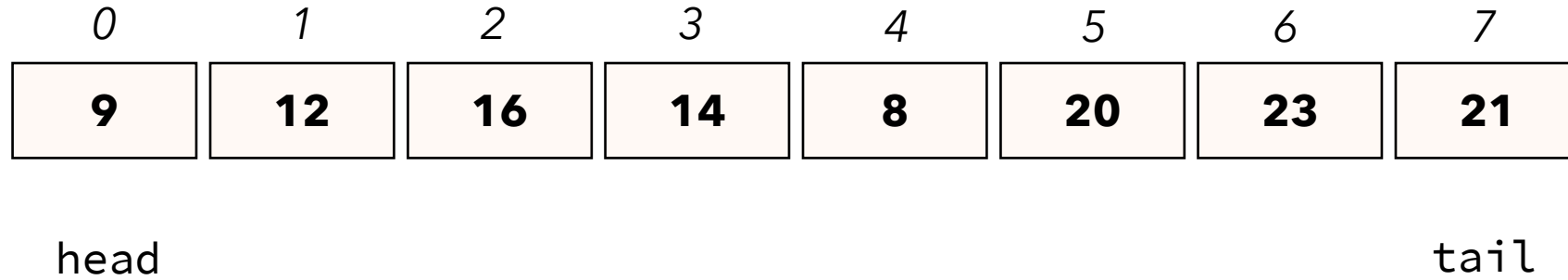
enqueue(21)

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 8 (FULL)**

**la coda è in stato FULL quando**  
**QUEUE\_SIZE è uguale a**  
**QUEUE\_CAPACITY**



# Queue - implementazione *array-based*



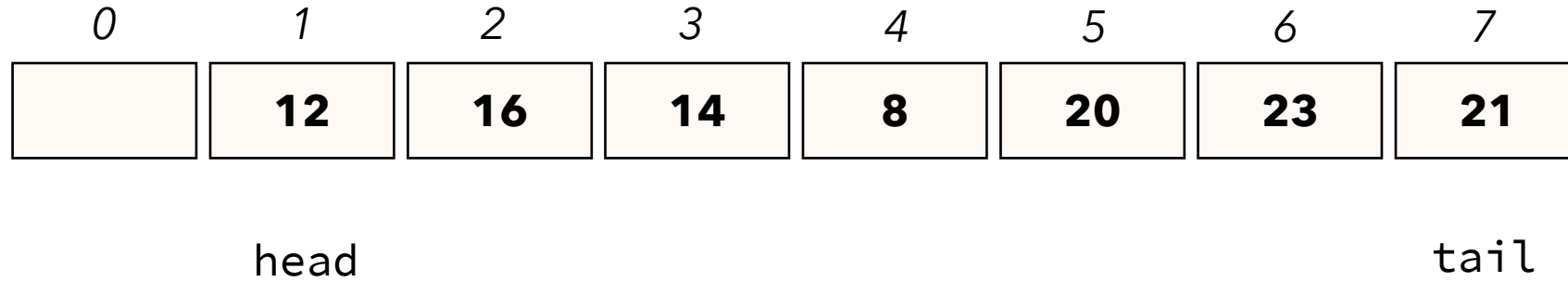
enqueue(19)

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 8 (FULL)**

**una enqueue su una coda FULL è  
una *no-operation*, i.e. non  
succede niente**

**dualmente, una dequeue su una  
coda vuota è anch'essa una *no-  
operation***

# Queue - implementazione *array-based*

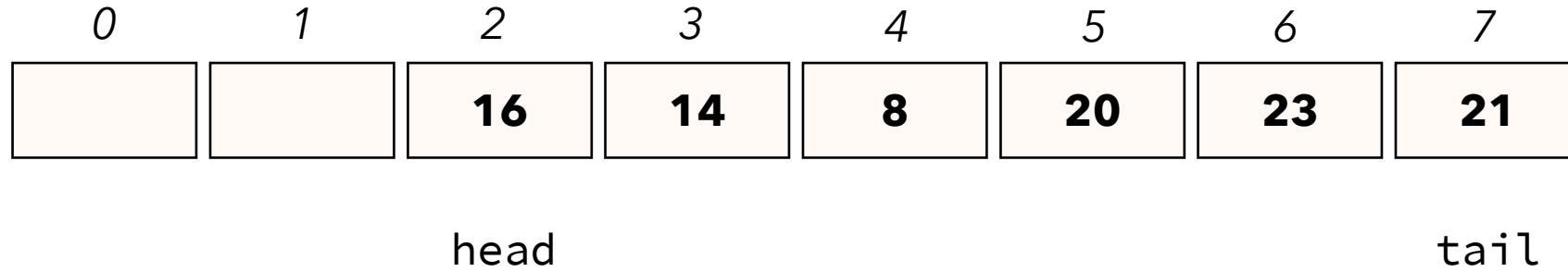


dequeue()

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 7**

**la *size* diminuisce ad ogni  
dequeue andata a buon fine**

# Queue - implementazione *array-based*

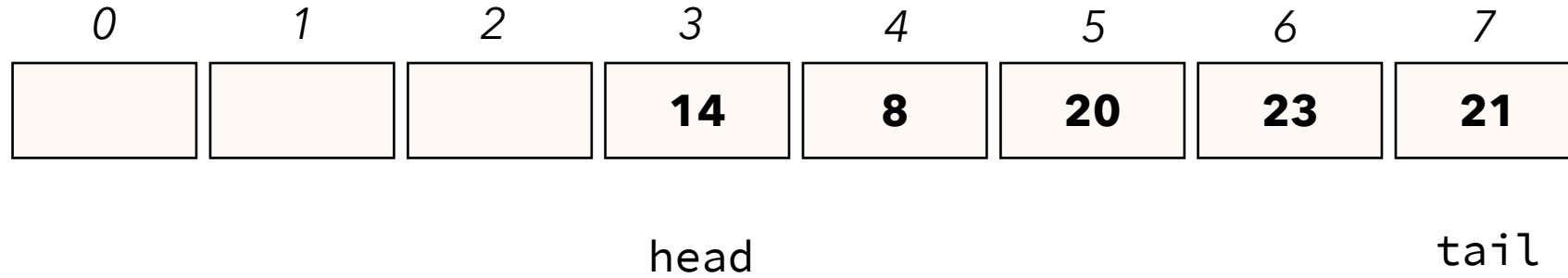


dequeue()

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 6**

**la *size* diminuisce ad ogni  
dequeue andata a buon fine**

# Queue - implementazione *array-based*

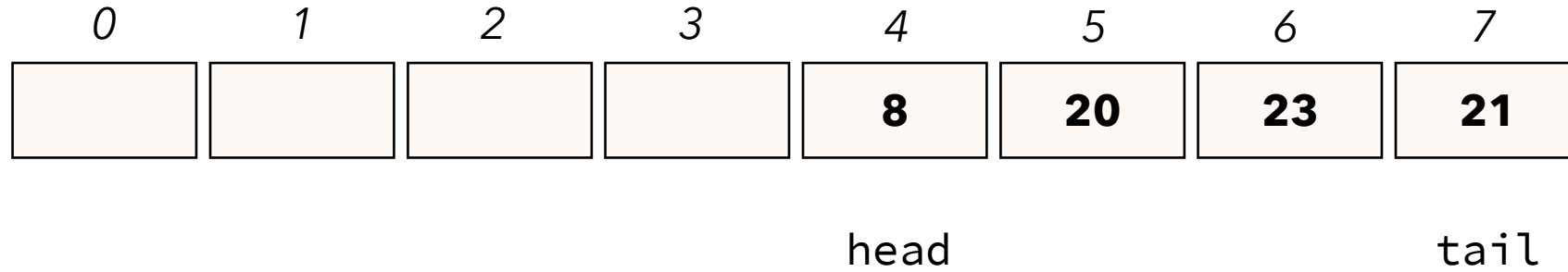


## dequeue()

QUEUE\_CAPACITY: 8  
QUEUE\_SIZE: 5

**la *size* diminuisce ad ogni dequeue andata a buon fine**

# Queue - implementazione *array-based*

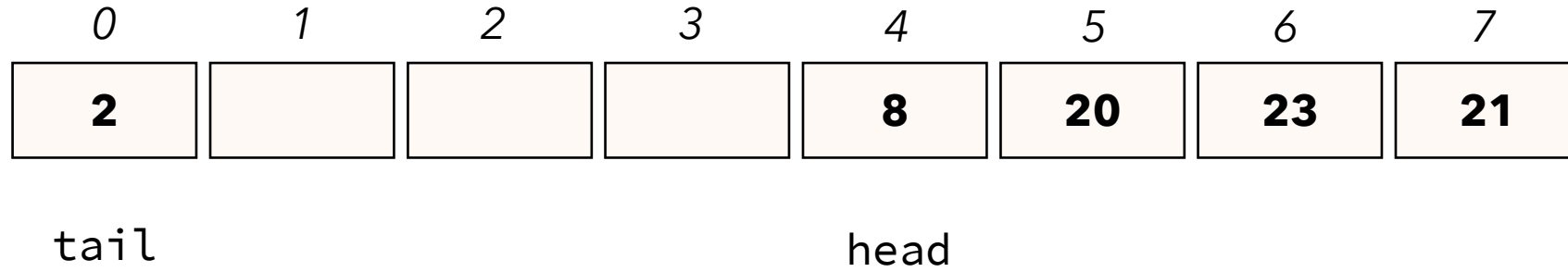


dequeue()

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 4**

**la *size* diminuisce ad ogni  
dequeue andata a buon fine**

# Queue - implementazione *array-based*



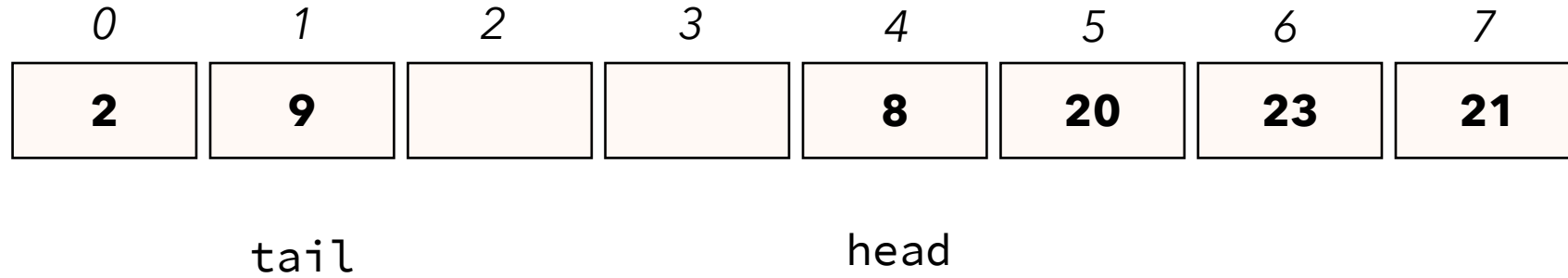
enqueue(2)

QUEUE\_CAPACITY: 8  
QUEUE\_SIZE: 5

**la *size* aumenta ad ogni enqueue andata a buon fine**

**si noti che l'array è gestito  
circularmente**

# Queue - implementazione *array-based*



enqueue(9)

**QUEUE\_CAPACITY: 8**  
**QUEUE\_SIZE: 6**

**la *size* aumenta ad ogni enqueue  
andata a buon fine**

# Queue - implementazione *linked-list-based*

- Si tratta di implementare una lista concatenata semplice per la quale sono ammesse solo le operazioni di inserimento in coda e rimozione dalla testa
- Bisogna mantenere 2 puntatori, uno al nodo in testa e un altro al nodo in coda
- L'operazione enqueue aggiornerà il puntatore al nodo in coda
- L'operazione dequeue aggiornerà il puntatore al nodo in testa



# Codice sorgente

- <https://github.com/Cyofanni/high-school-cs-class/tree/main/cplusplus/oop/stack>
- <https://github.com/Cyofanni/high-school-cs-class/tree/main/cplusplus/oop/queue>