Modulo: Strutture dati dinamiche lineari [P2_08]

Unità didattica: Strutture dati dinamiche lineari (1) [3-T]

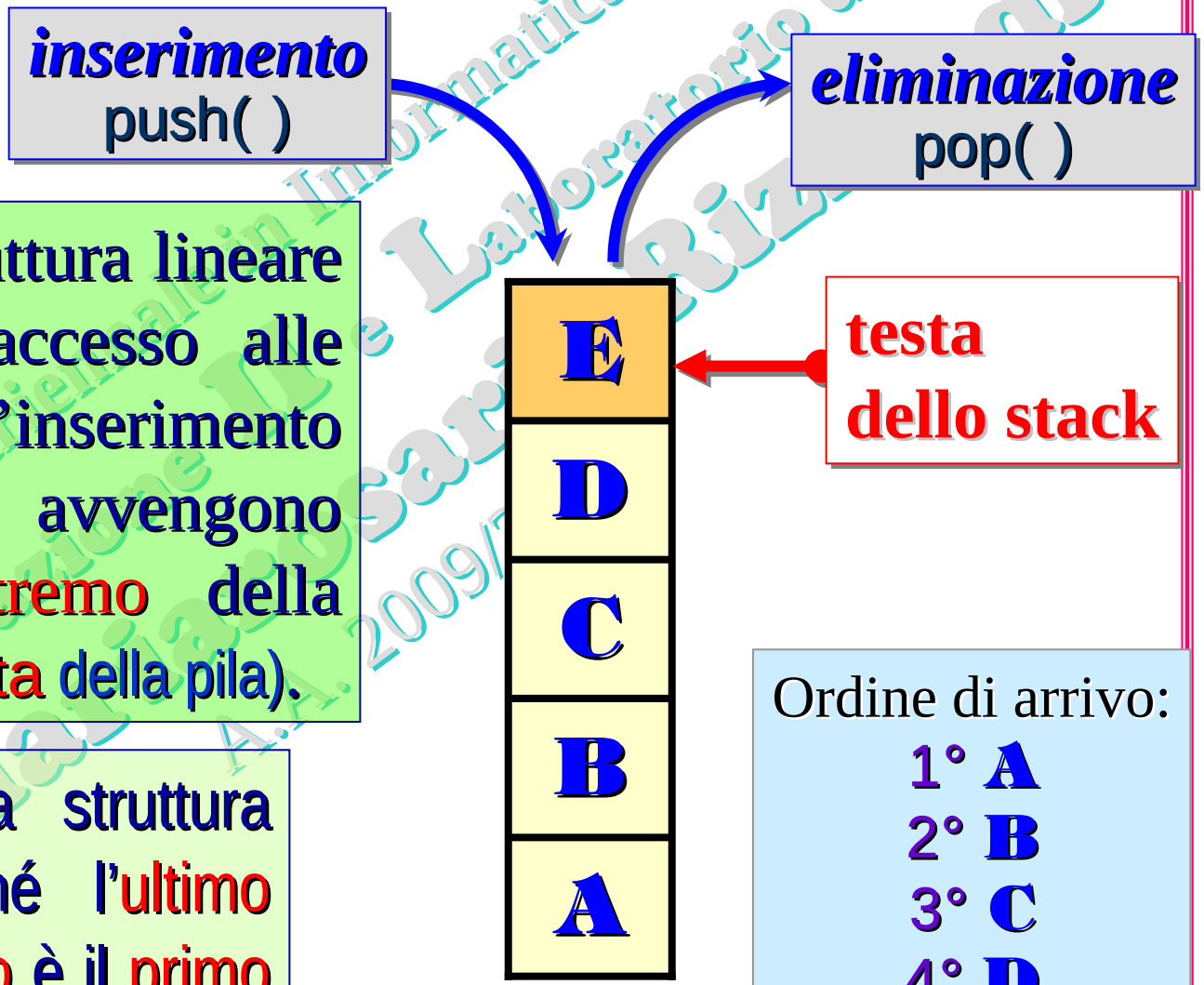
Titolo: Principali strutture dinamiche lineari: pila (stack), coda (queue)

Argomenti trattati:

- ✓ Struttura LIFO: la pila (stack)
- ✓ Inserimento (push), eliminazione (pop) su pila
- ✓ Simulazione di una pila mediante array
- ✓ Struttura FIFO: la coda (queue)
- ✓ Inserimento (enqueue), eliminazione (dequeue) su coda
- ✓ Simulazione di una coda mediante array

Prerequisiti richiesti: fondamenti della programmazione C, array, generalità sulle strutture dati dinamiche

Tipo di dato astratto (Abstract Data Type): **Pila (Stack)**



La pila è una struttura lineare aperta in cui l'accesso alle componenti per l'inserimento e l'eliminazione avvengono solo ad un estremo della struttura (detta testa della pila).

La pila è una struttura **L.I.F.O.** perché l'ultimo elemento inserito è il primo ad essere eliminato.

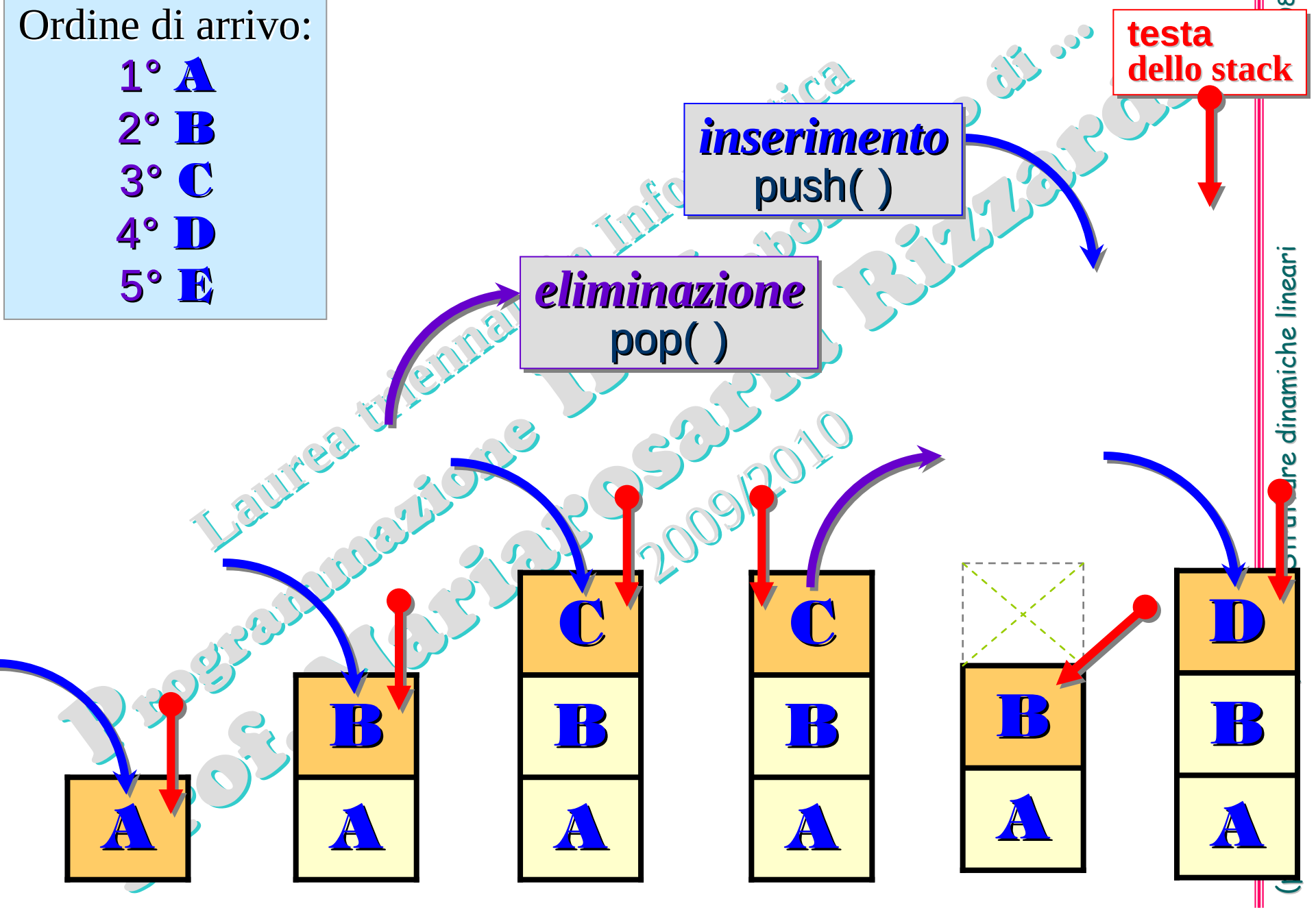
Ordine di arrivo:

1° **A**
2° **B**
3° **C**
4° **D**
5° **E**

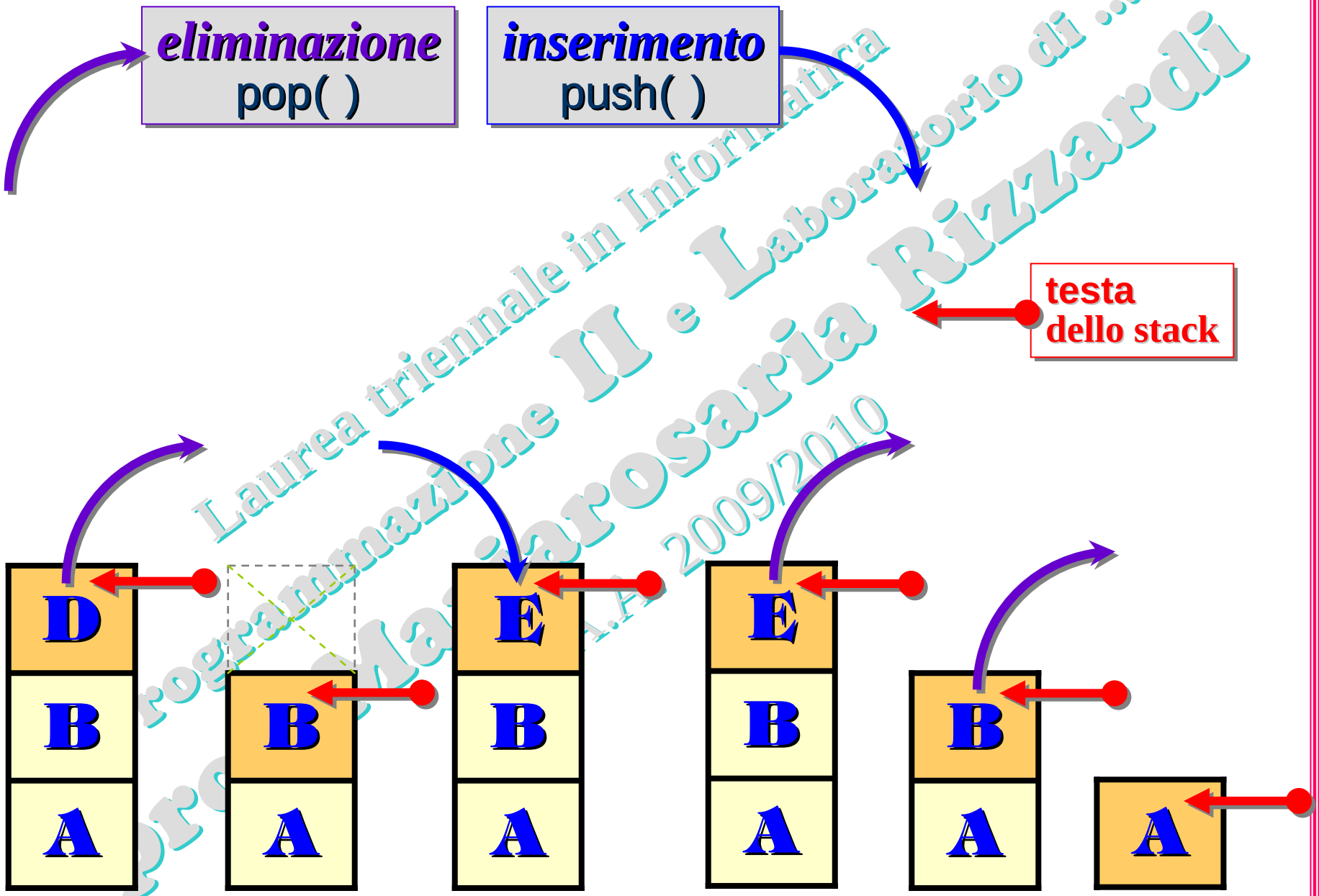
Pila (Stack)

Ordine di arrivo:

1° **A**
2° **B**
3° **C**
4° **D**
5° **E**

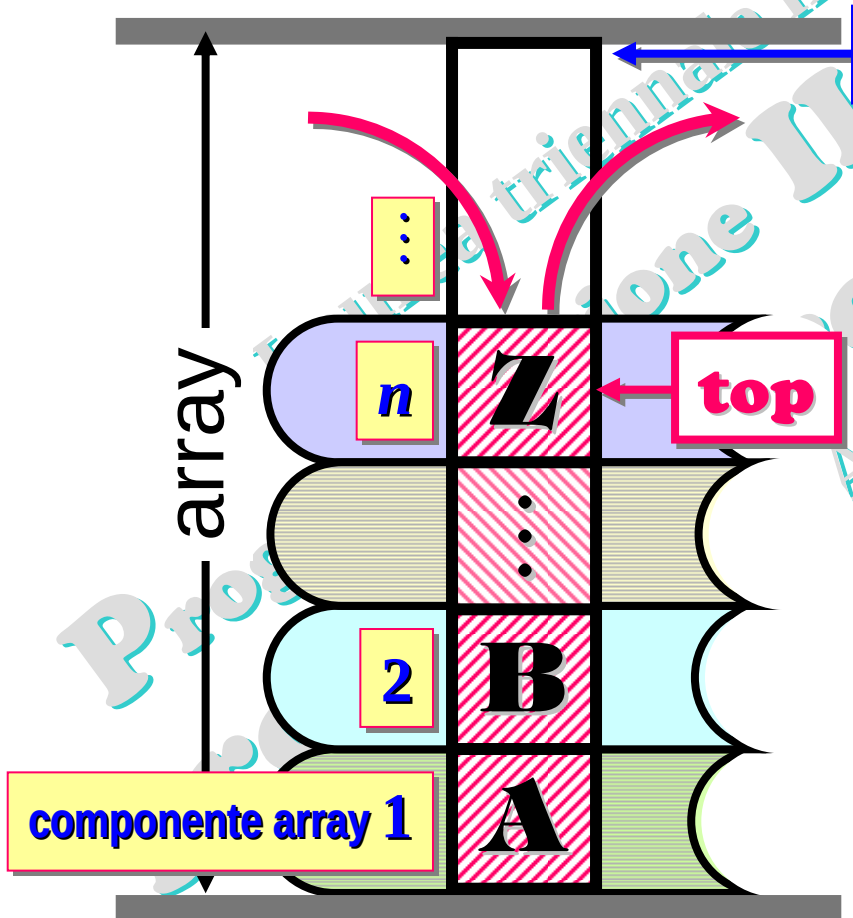


Pila (Stack)



1 - Il tipo pila (stack) tramite array

Si può realizzare una pila (stack) in due modi: nel primo, statico, si usa un array e si stabilisce una dimensione massima di riempimento dello stack (***Max_stack_size***); nel secondo, dinamico, si usa una lista lineare.



La pila si presta ad essere rappresentata come un "array" convenendo di immaginarlo **rovesciato**: l'inserimento e l'eliminazione, che avvengono alla testa dello stack, in realtà si verificano sull'ultima componente usata dell'array.

Esempio 1a: invertire l'ordine delle componenti di un array mediante uno stack – versione array statico


```
#include <stdio.h>
#include <stdlib.h>
#define MAX_STACK_SIZE 100
void inverte_array(char [],short);
void push_s(char ,char [],short *);
void pop_s(char *,char [],short *);
void main()
{char
a[]={'A','B','C','D','E','F','G','H','I','L'};
 short len_a=10, i;
    puts("array prima");
    for (i=0;i<len_a;i++)
         printf("a[%d]=%c\n",i,a[i]);
    inverte_array(a,len_a);
    puts("array dopo");
    for (i=0;i<len_a;i++)
         printf("a[%d]=%c\n",i,a[i]);
}
      …
```

```
      …
void inverte_array(char a[],short len_a)
{char temp[MAX_STACK_SIZE];
 short i,head;
    head=-1; /* indica stack vuoto */
    for (i=0;i<len_a;i++) push_s(a[i],temp,&head);
    for (i=0;i<len_a;i++) pop_s(a+i,temp,&head);
}

void push_s(char elem,char p_stack[],short *head)
{
/* (*head)++ ⇔ *head=*head+1;*/
 *(p_stack+ ++*head)=elem;
}

void pop_s(char *elem,char p_stack[],short *head)
{
 *elem=*(p_stack+(*head)--);
/* (*head)-- ⇔ *head=*head-1;*/
}
```

notazione prefissa

notazione postfissa

Attenzione: si deve controllare il riempimento dello stack !!!

Attenzione: … allo svuotamento dello stack !!!

Esempio 1b: invertire l'ordine delle componenti di un array mediante uno stack – versione array dinamico

stesso main()

```
#include <stdio.h>
#include <stdlib.h>
void inverte_array(char [],short);
void push_s(char ,char *,short *);
void pop_s(char *,char *,short *);

void main()
{char  a[]={'A','B','C','D','E','F','G','H','I','L'};
 short len_a=10,i;
    puts("array prima");
    for (i=0;i<len_a;i++) printf("a[%d]=%c\n",i,a[i]);
    inverte_array(a,len_a);
    puts("array dopo");
    for (i=0;i<len_a;i++) printf("a[%d]=%c\n",i,a[i]);
}
      ...
```

```
      …
void inverte_array(char a[],short len_a)
{char *p_temp; short i,head;
    head=-1; /* indica stack vuoto */
    p_temp = calloc(len_a, sizeof a[1]);
    for (i=0;i<len_a;i++) push_s(a[i],p_temp,&head);
    for (i=0;i<len_a;i++)  pop_s(a+i, p_temp,&head);
    free(p_temp);
}
void push_s(char elem,char *p_stack,short *head)
{
/* (*head)++; *head=*head+1; */
 *(p_stack+ ++*head)=elem;
}

void pop_s(char *elem,char *p_stack,short *head)
{
 *elem=*(p_stack+(*head)--);
/* (*head)--; *head=*head-1;  */
}
```

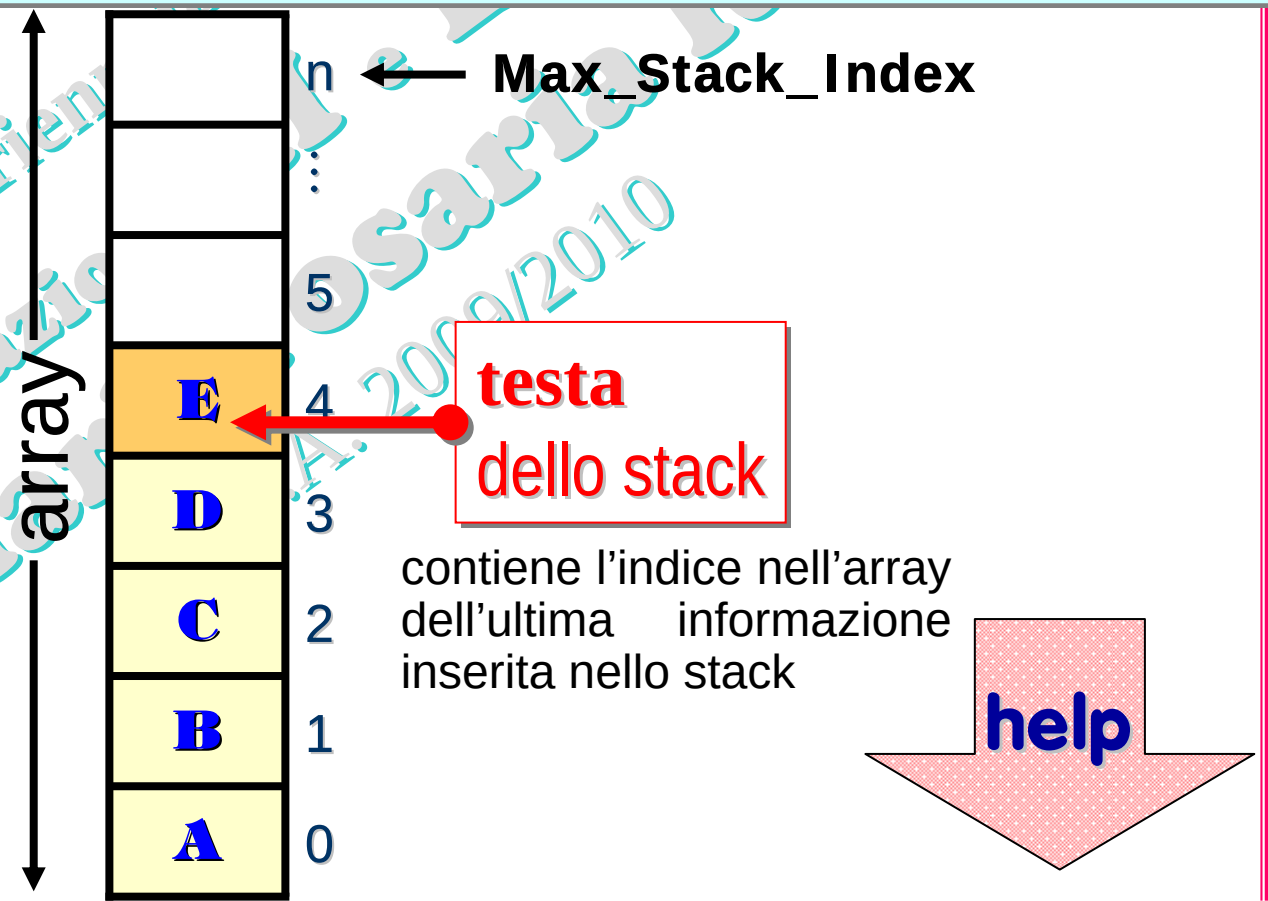
che differenza c'è?

Attenzione: … al riempimento dello stack !!!

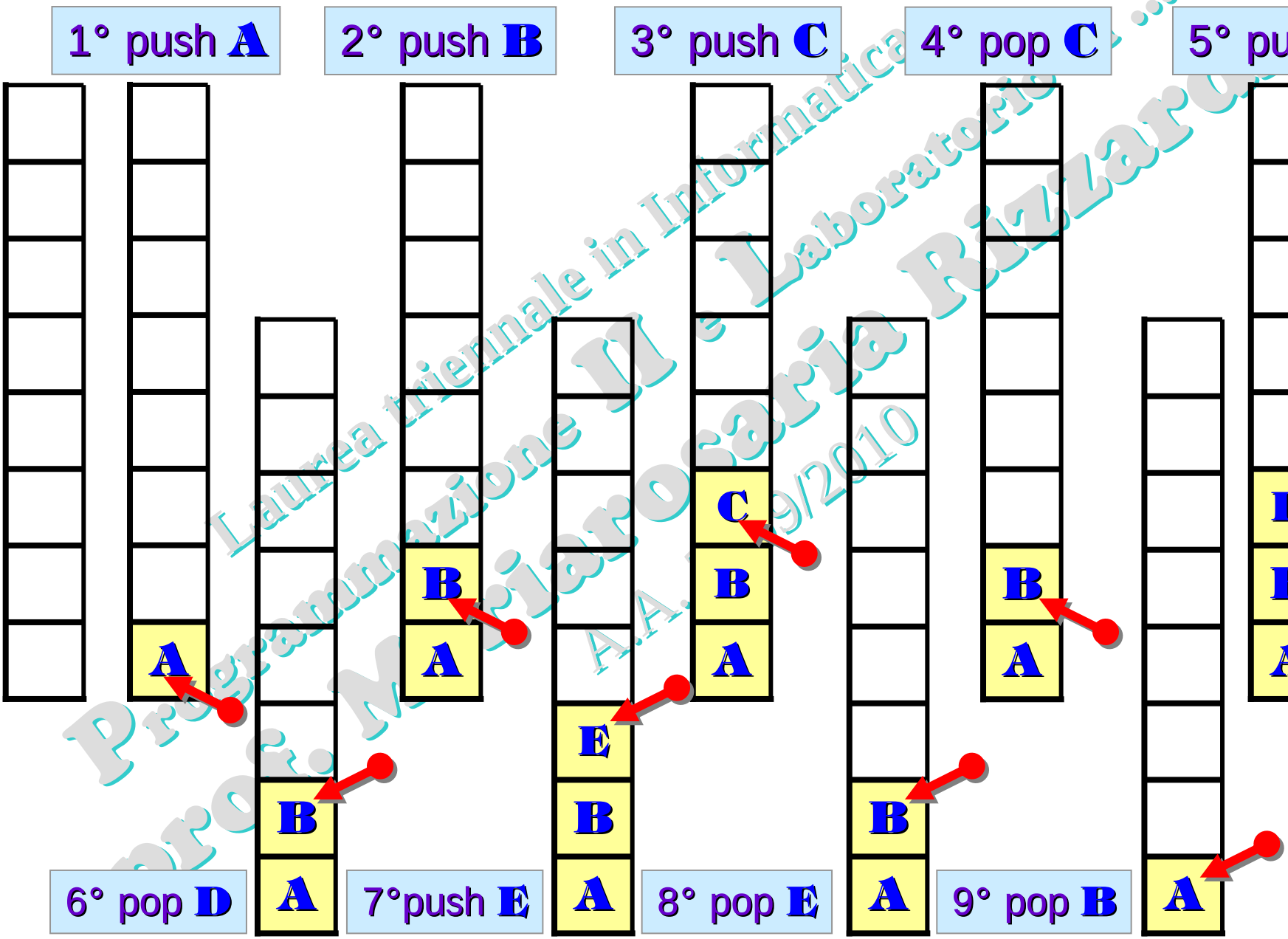
Attenzione: … allo svuotamento dello stack !!!

Laboratorio:

Simulare in *C* la gestione di una ***pila*** (***stack***) tramite array statico (può essere anche un array di struct) creando le funzioni di manipolazione **push( )** [inserimento] e **pop( )** [eliminazione]. Il programma deve prevedere un menù che consenta di scegliere l'operazione da eseguire.



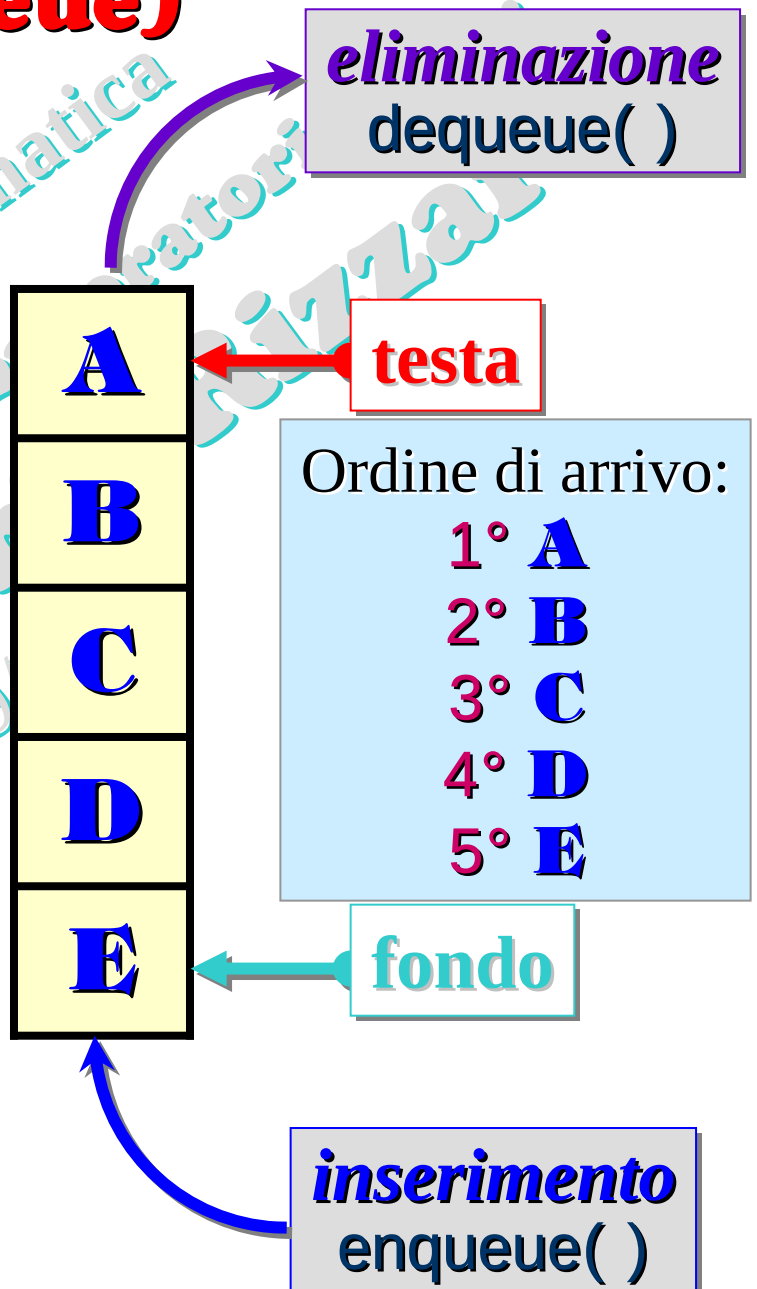
esempio



Tipo di dato astratto (Abstract Data Type): Coda (Queue)

La coda è una struttura lineare aperta in cui l'accesso alle componenti avviene solo ai due estremi:
l'eliminazione avviene solo all'inizio della struttura (testa);
l'inserimento avviene solo alla fine (fondo).

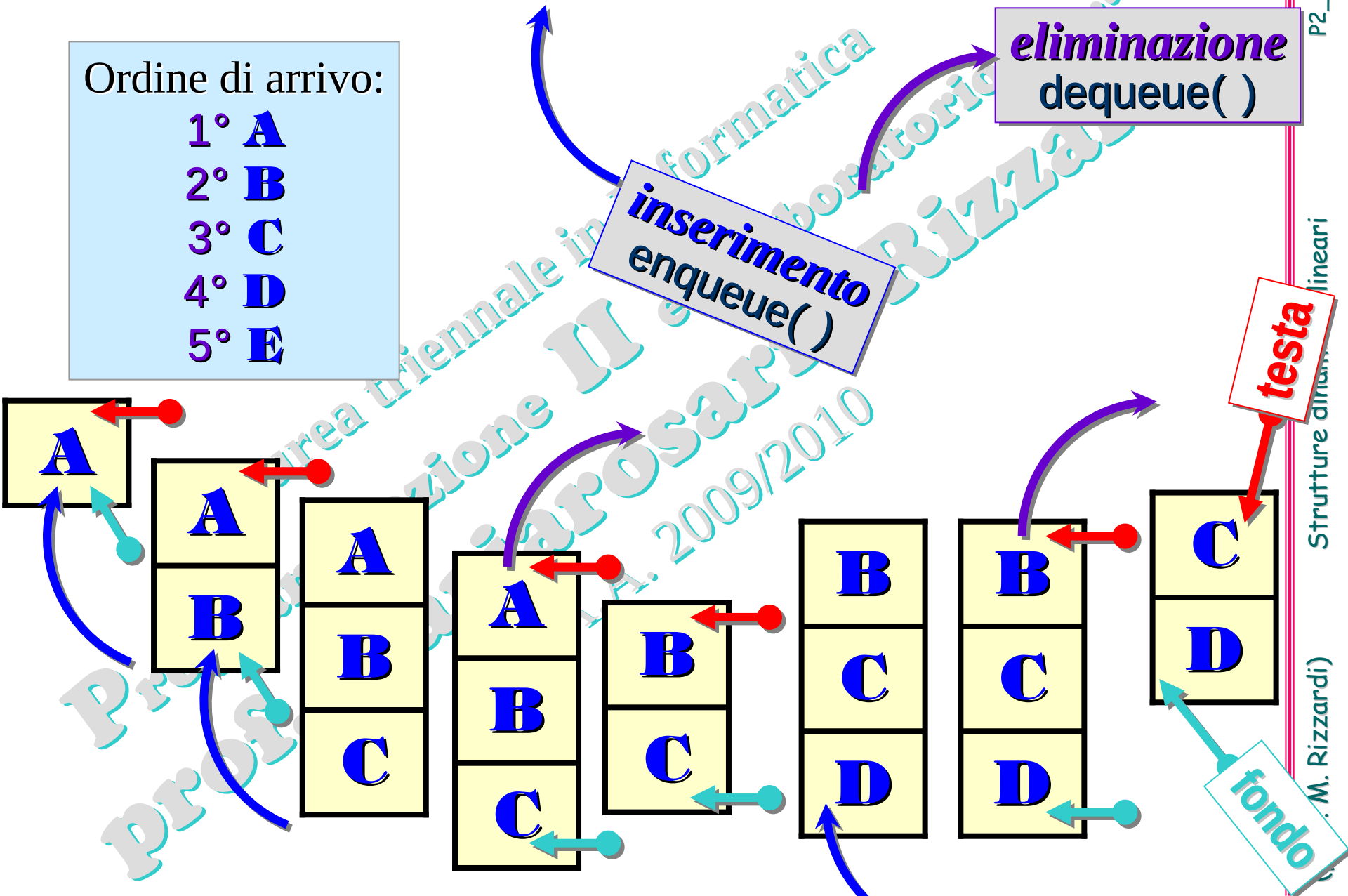
La coda è una struttura **F.I.F.O.** perché il primo elemento inserito è il primo ad essere eliminato.



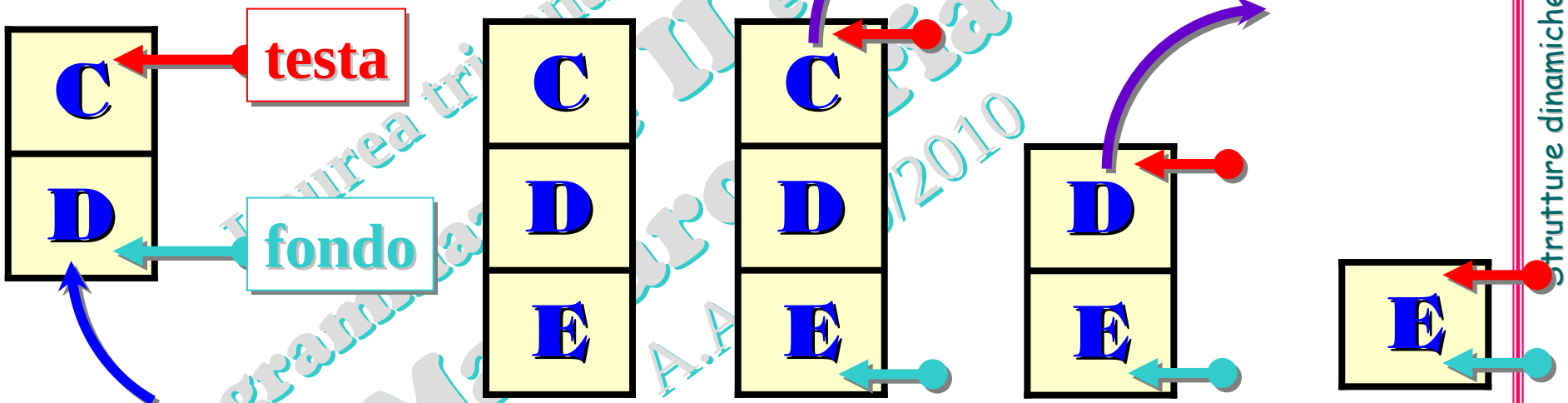
Coda (Queue)

Ordine di arrivo:

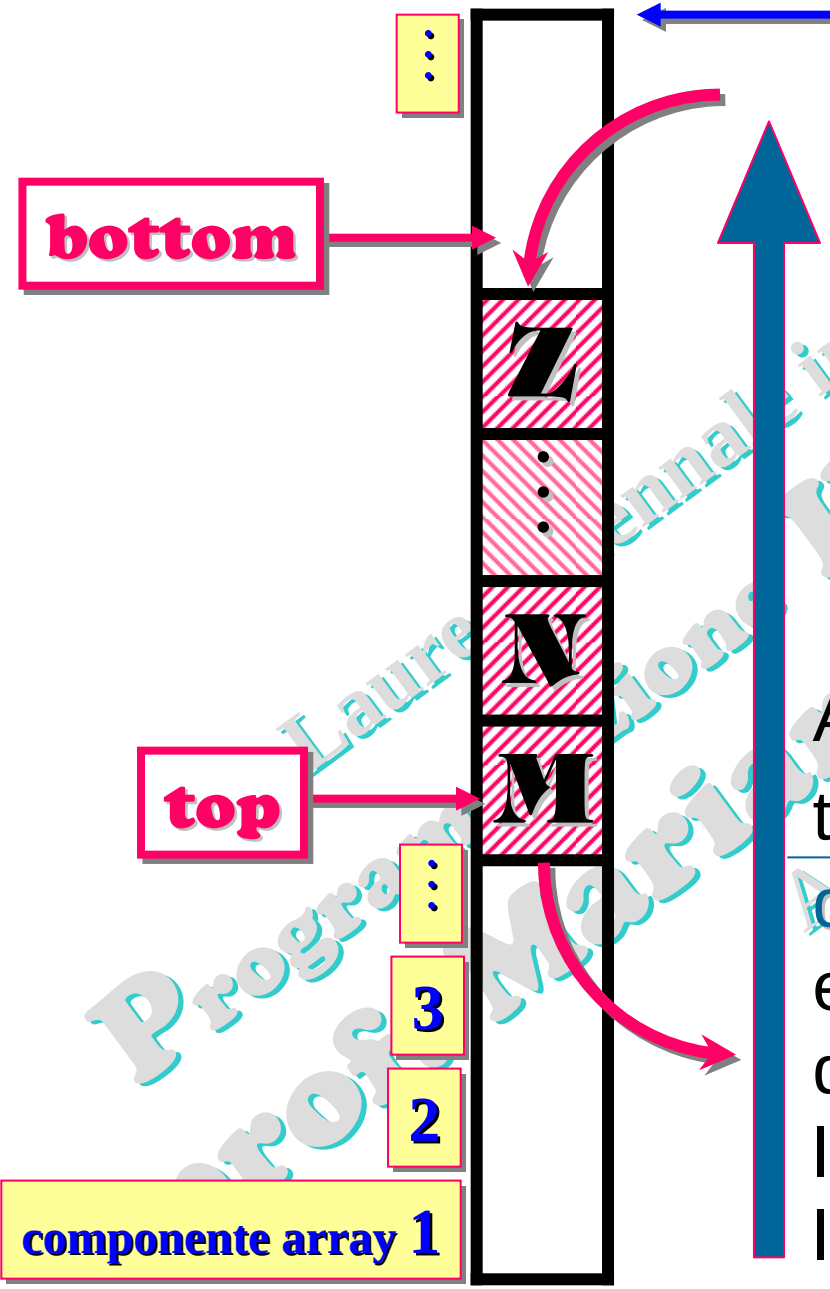
1° **A**
2° **B**
3° **C**
4° **D**
5° **E**



Coda (Queue)



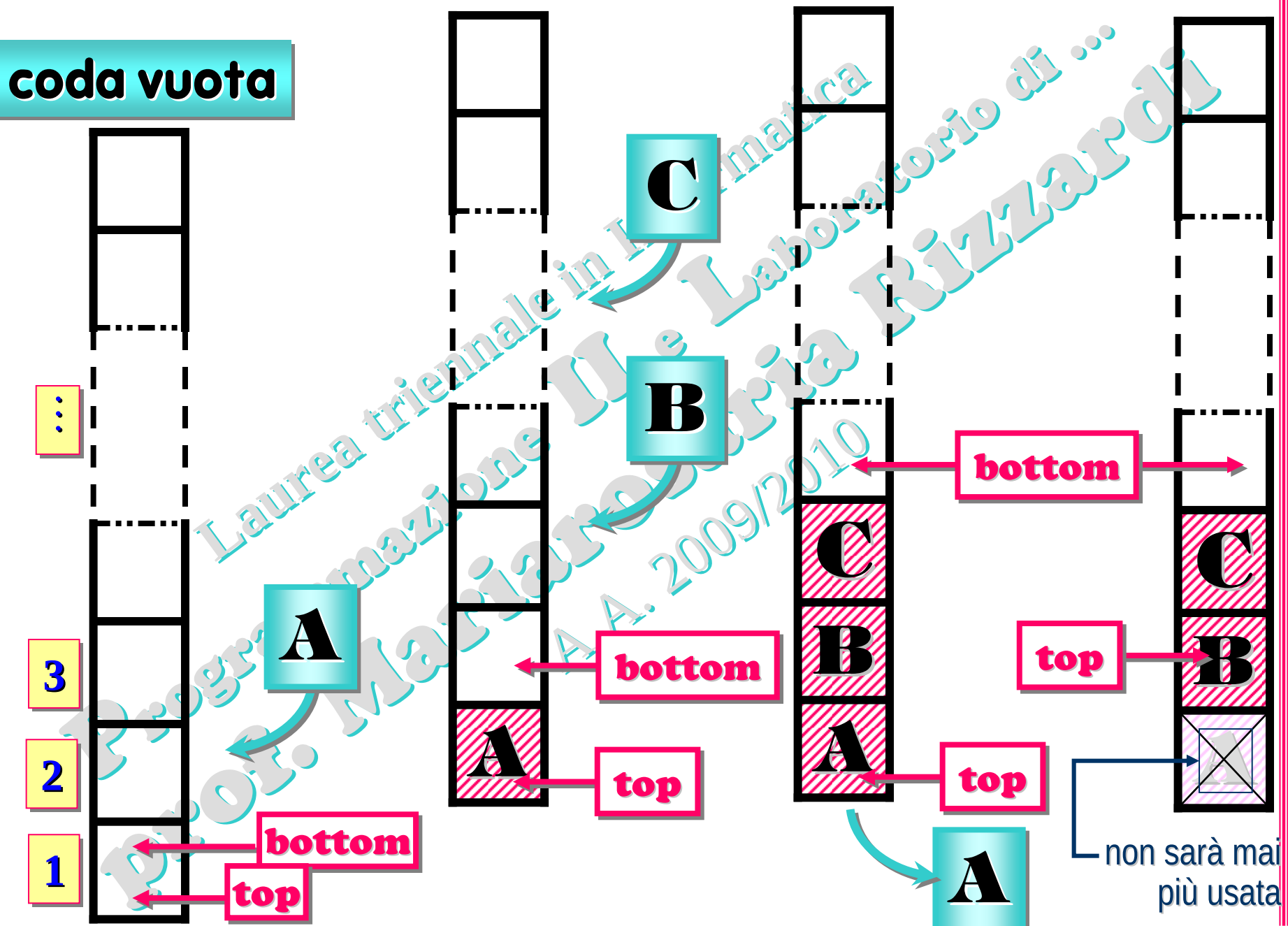
2 - Il tipo queue tramite array



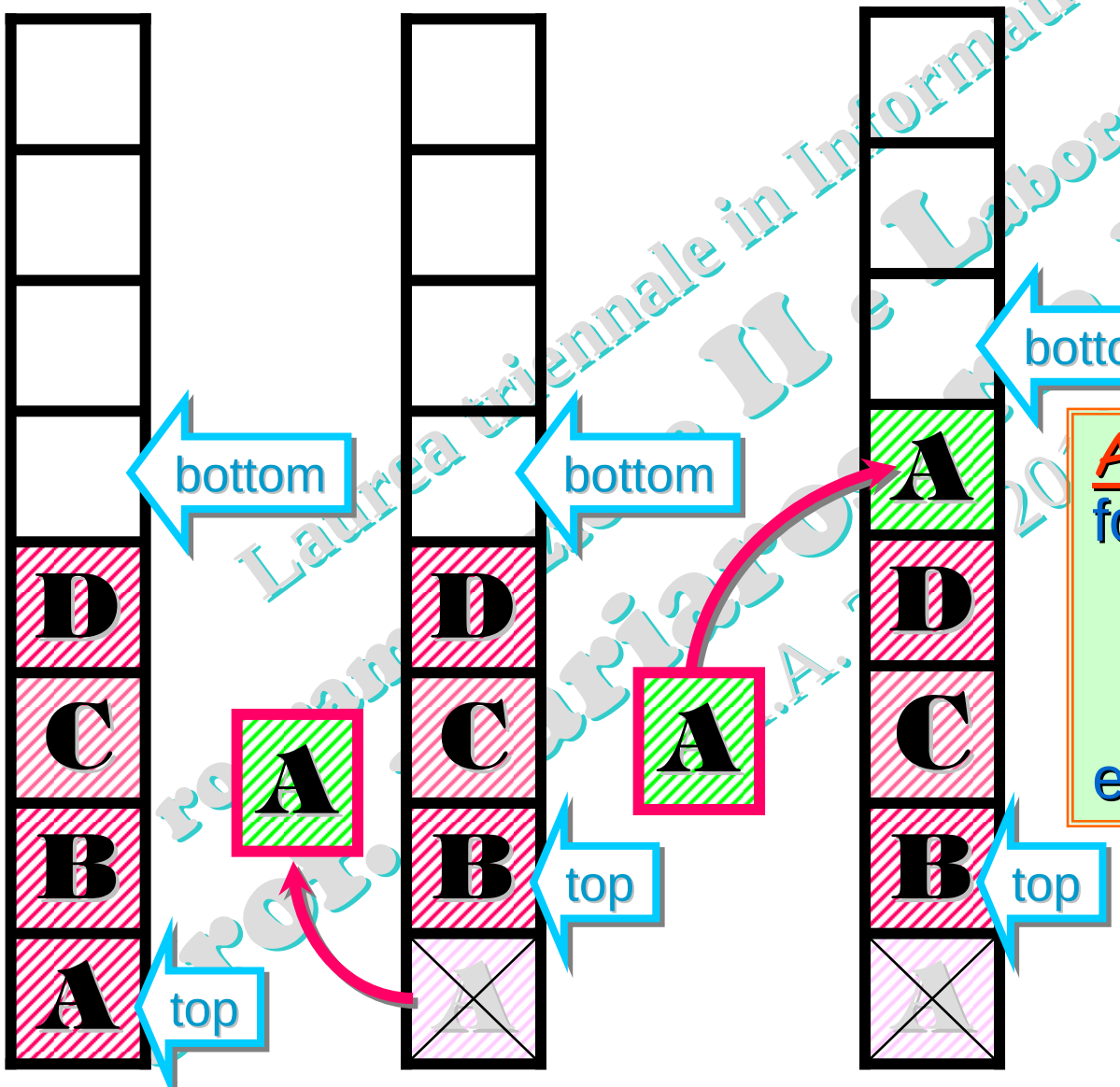
Analogamente a quanto visto per la pila, anche una coda (queue) si può simulare tramite array, ma non è conveniente in generale.

A differenza dello stack, la coda tende a **slittare** verso la fine dell'array in quanto l'inserimento e l'eliminazione avvengono ai due estremi opposti: l'inserimento alla fine dell'array e l'eliminazione all'inizio.

la coda tende a **slittare** verso la fine dell'array ...infatti

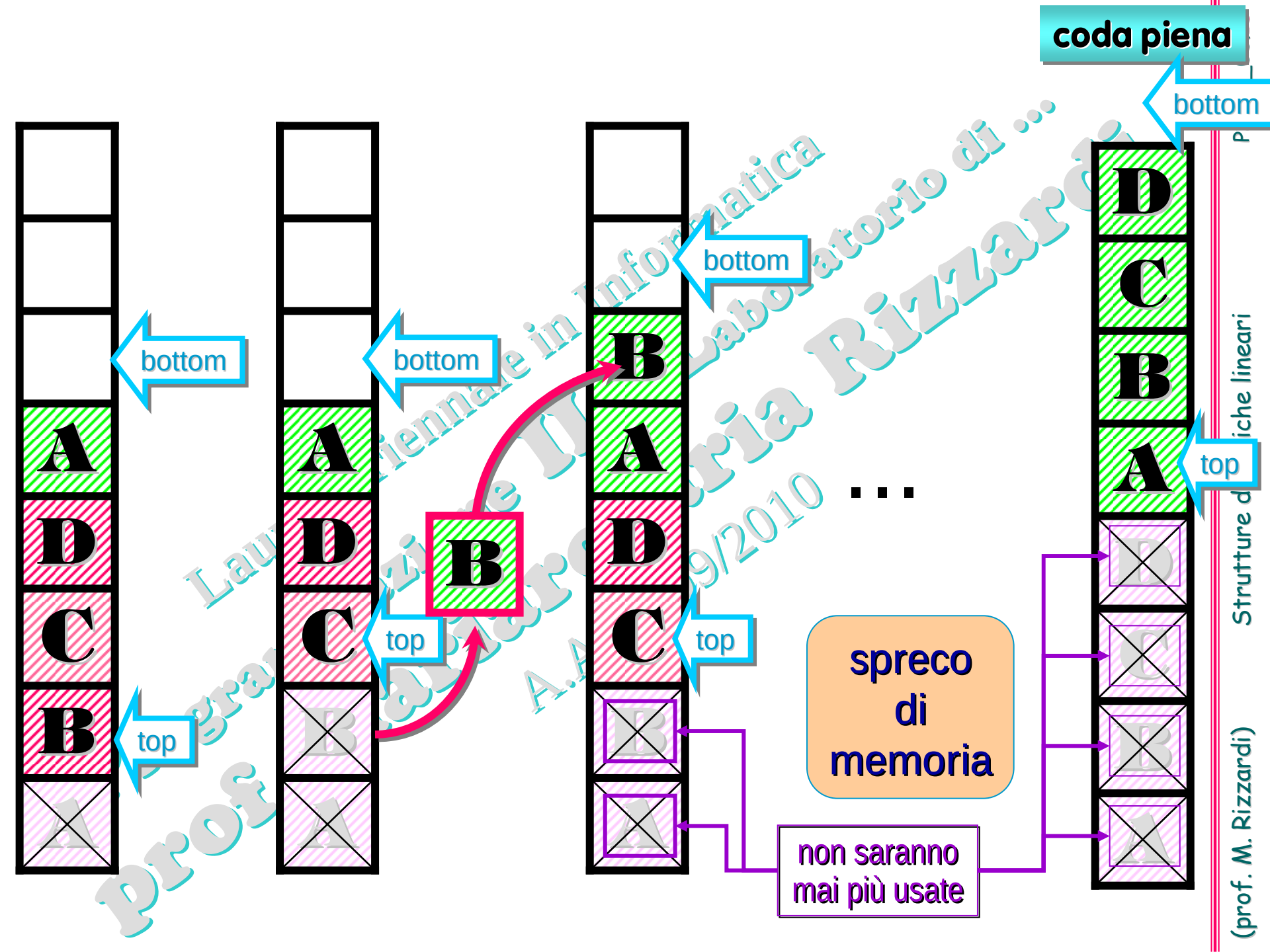


Esempio: visita sequenziale su una coda simulata mediante array



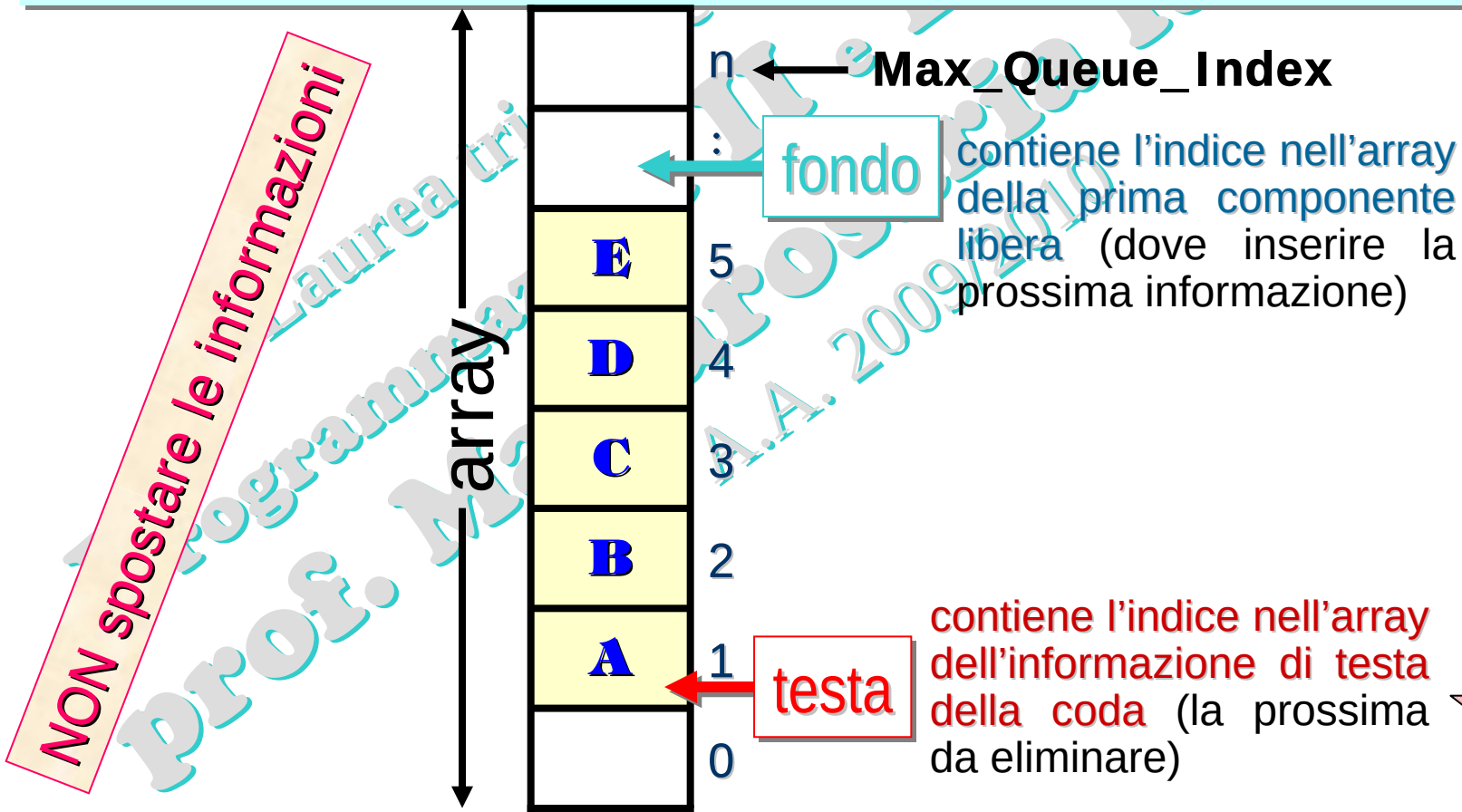
Algoritmo

```
for k=1 to NumElem
    dequeue(coda,Elem)
    ....
    enqueue(Elem,coda)
end
```

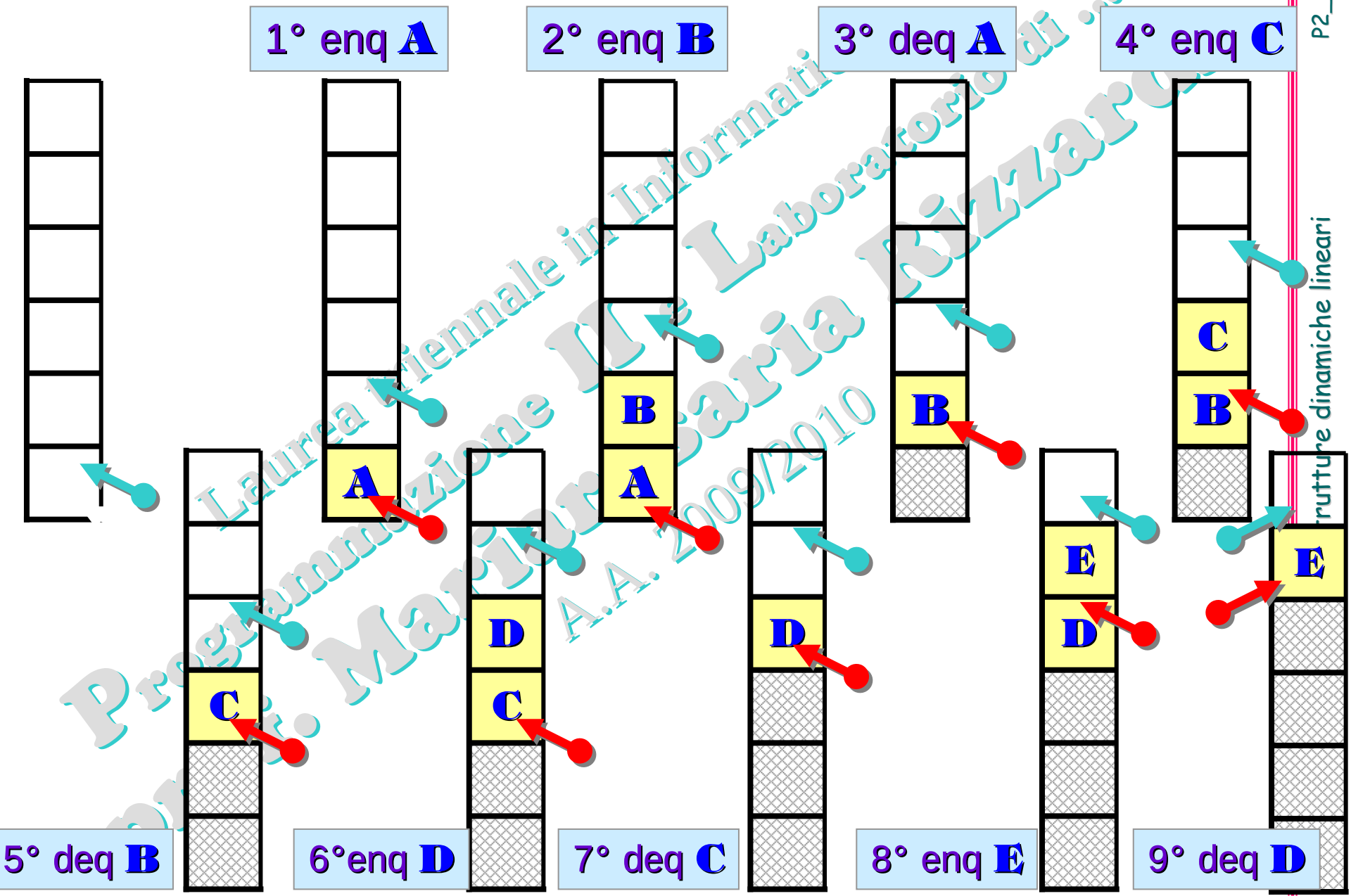


Laboratorio:

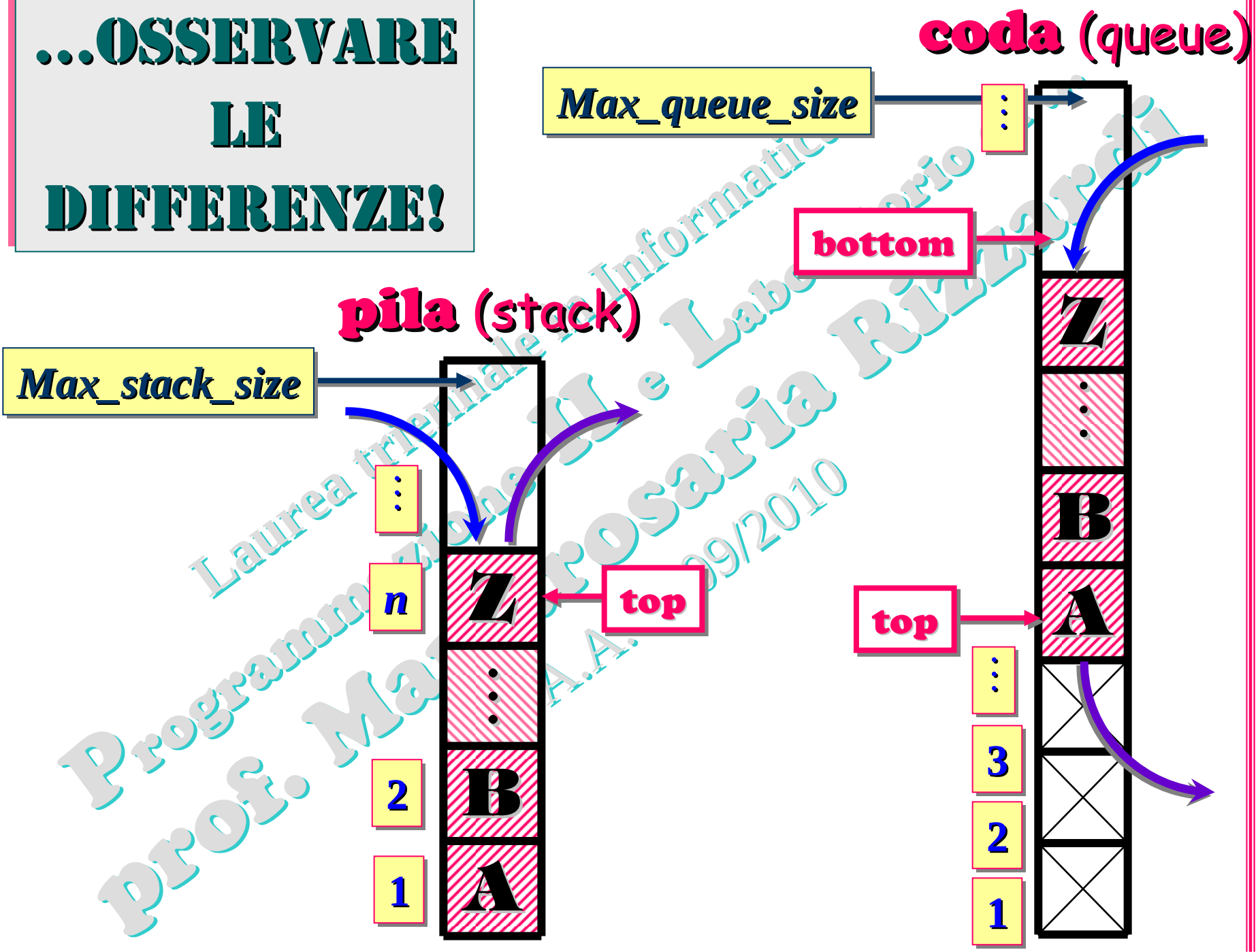
Simulare in *C* la gestione di una ***coda*** (***queue***) tramite array statico (può essere anche un array di struct) creando le funzioni di manipolazione **enqueue( )** [inserimento] e **dequeue( )** [eliminazione]. Il programma deve prevedere un menù che consenta di scegliere l'operazione da eseguire.



esempio

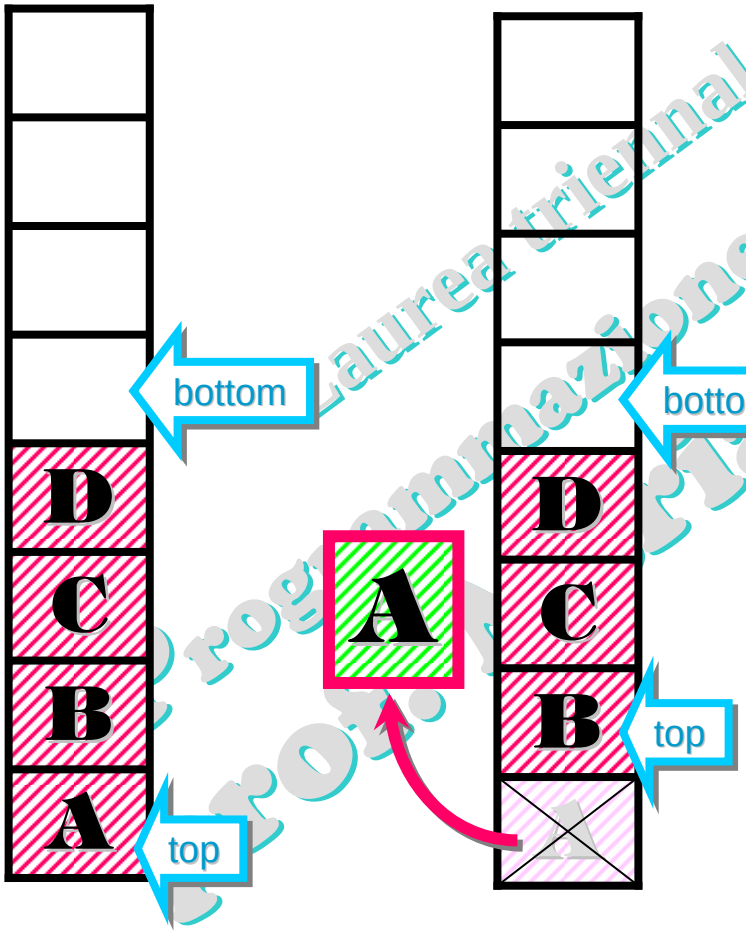


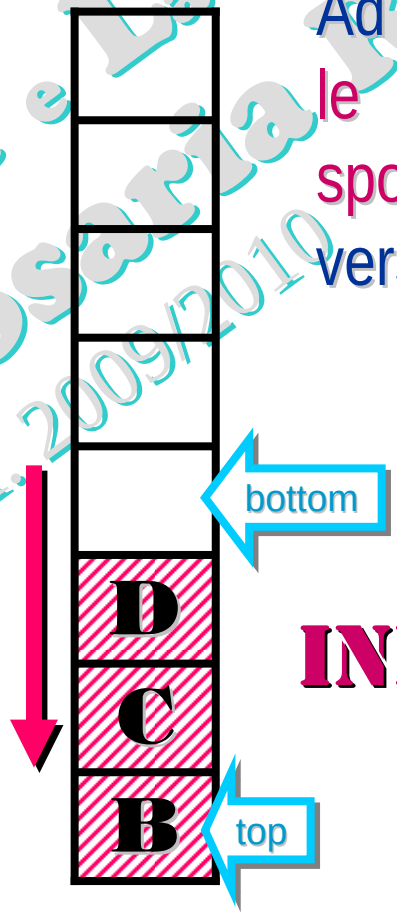
...OSSERVARE LE DIFFERENZE!



come evitare lo slittamento della coda verso la fine dell'array?

compattando verso l'inizio dell'array





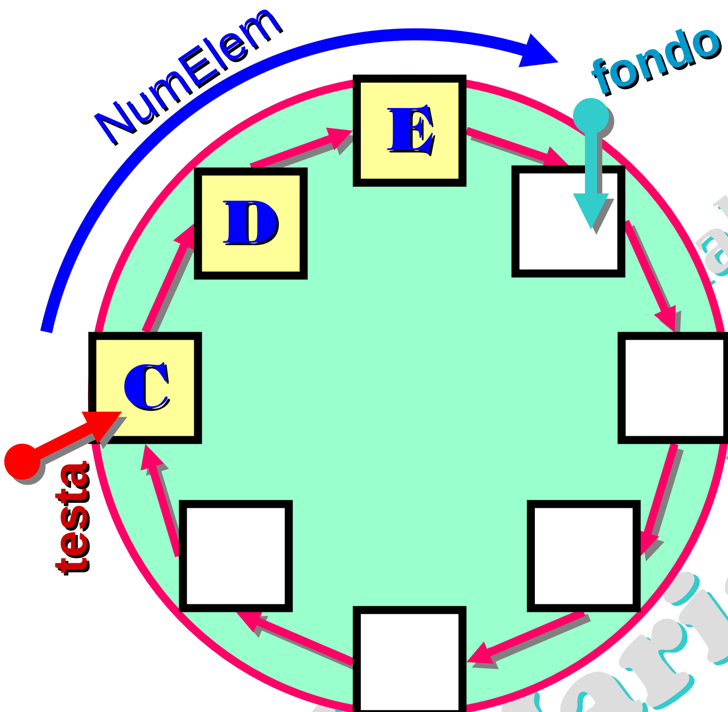
Ad ogni eliminazione le informazioni sono spostate di un posto verso l'inizio dell'array.

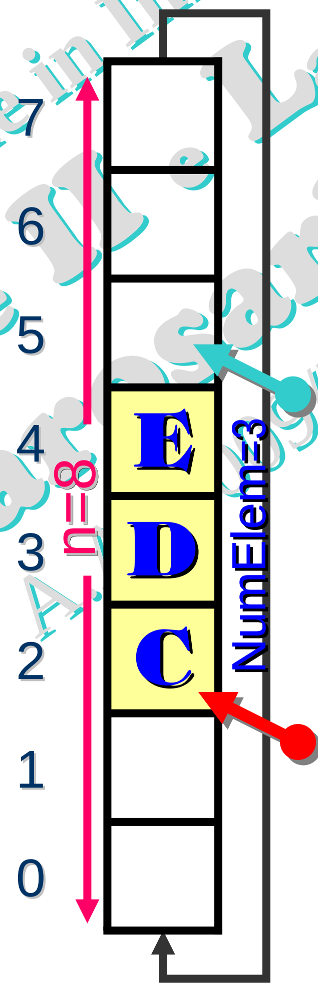
INEFFICIENTE!

come evitare lo slittamento della coda verso la fine dell'array?

considerando l'array circolare

$\forall k \in \mathbb{N} \; k_{mod 8} \in \{0,1,2,\dots,7\}$





inserimento

if (NumElem < n)
  fondo := testa + NumElem;
  fondo := $[fondo]_{mod\ n}$;
  NumElem := NumElem+1;
else
  {Overflow di array}

eliminazione

if (NumElem > 0)
  testa := testa + 1;
  testa := $[testa]_{mod\ n}$;
  NumElem := NumElem-1;
else
  {coda vuota}

inizializzazione

NumElem := 0; {coda vuota}
testa := 0;

Esercizio:

3

Simulare in ***C*** la gestione di una ***coda*** (***queue***) tramite array circolare statico (può essere anche un array di struct) creando le funzioni di manipolazione **enqueue( )** [inserimento] e **dequeue( )** [eliminazione]. Il programma deve prevedere un menù che consenta di scegliere l'operazione da eseguire. [liv. 3]