

# **Esercitazione 07**

2026/10/16 – Puntatori

L. Alessandrini

Credits: A. Montenegro per gli esercizi

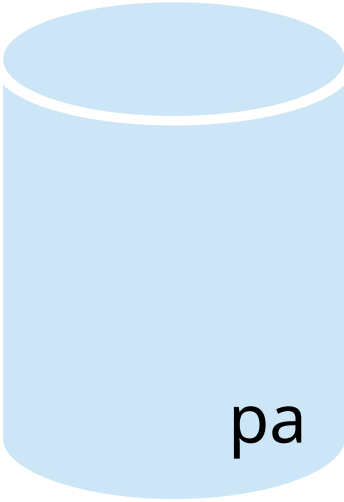
# **Puntatori**

Funzionamento Base

# Puntatori

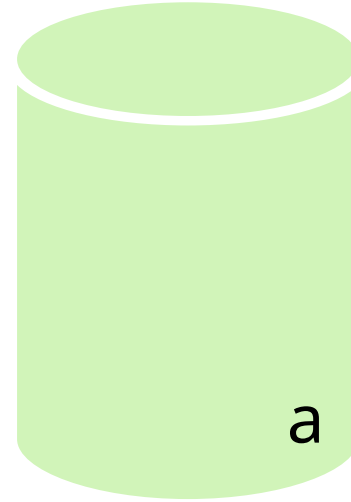
## Funzionamento Base

`int *pa;`



`0x7ff7bfeff310`

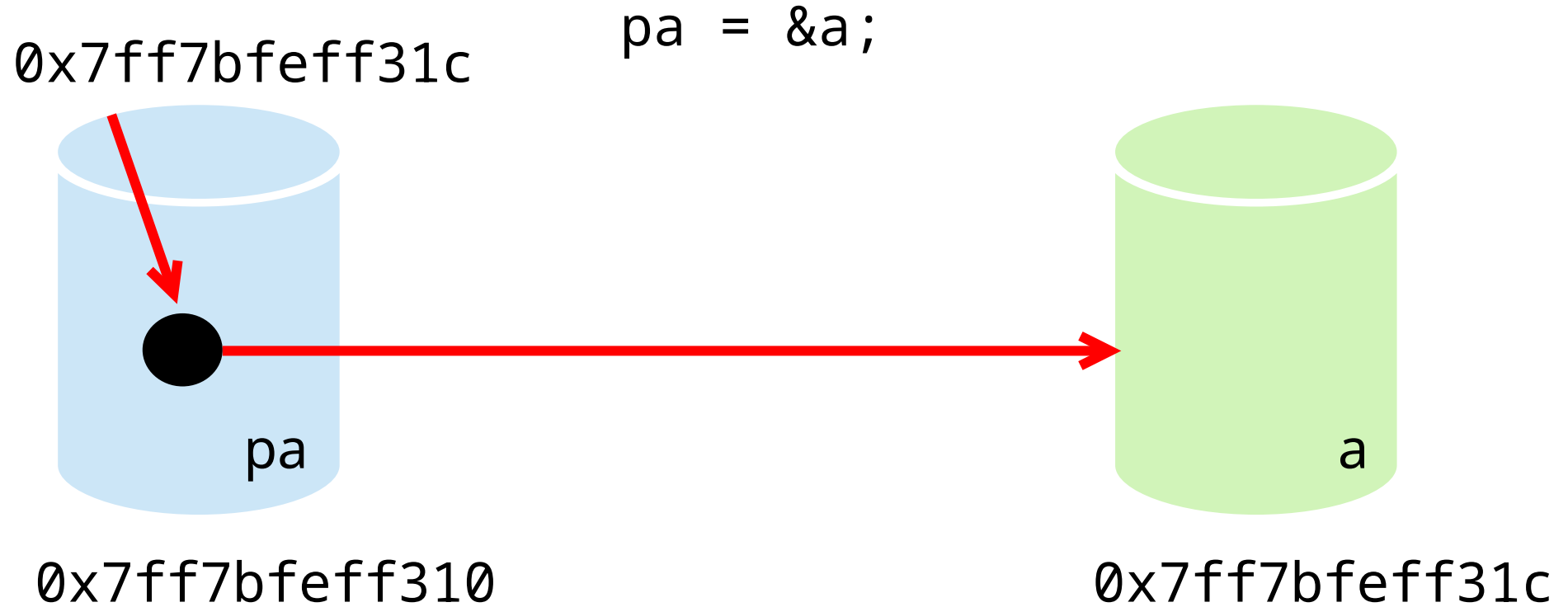
`int a;`



`0x7ff7bfeff31c`

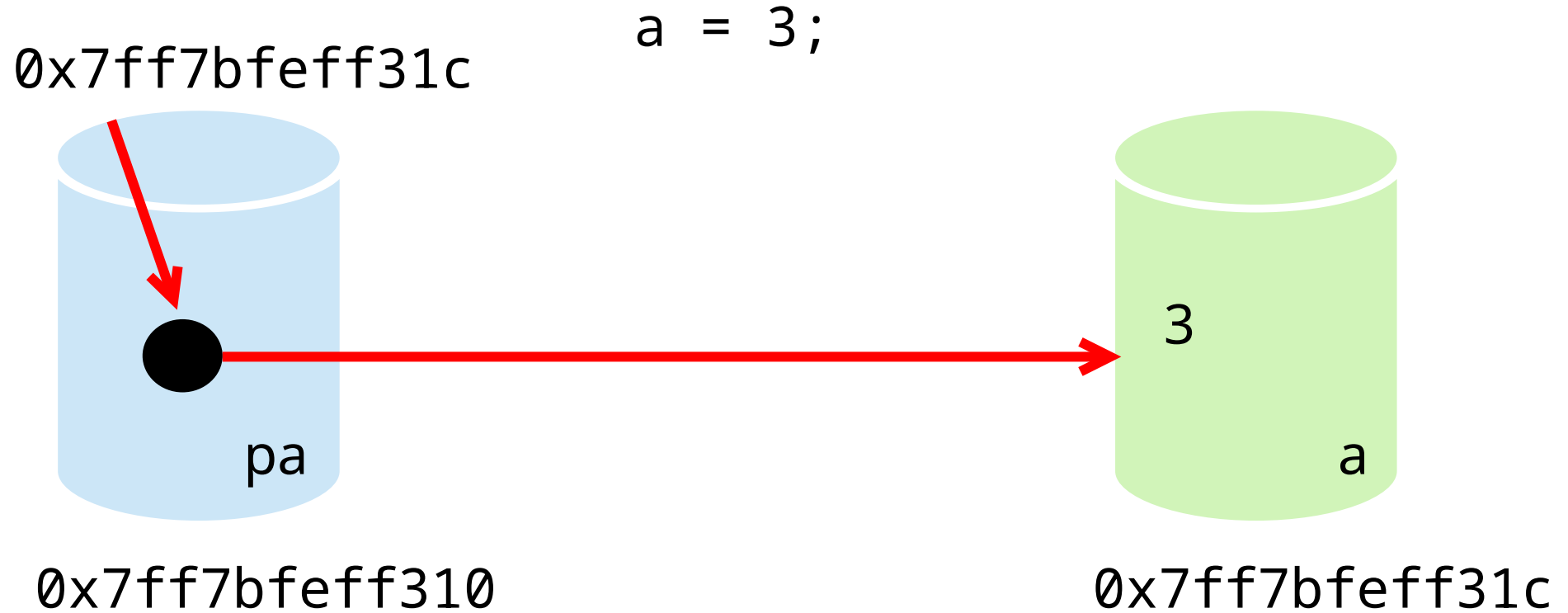
# Puntatori

## Funzionamento Base



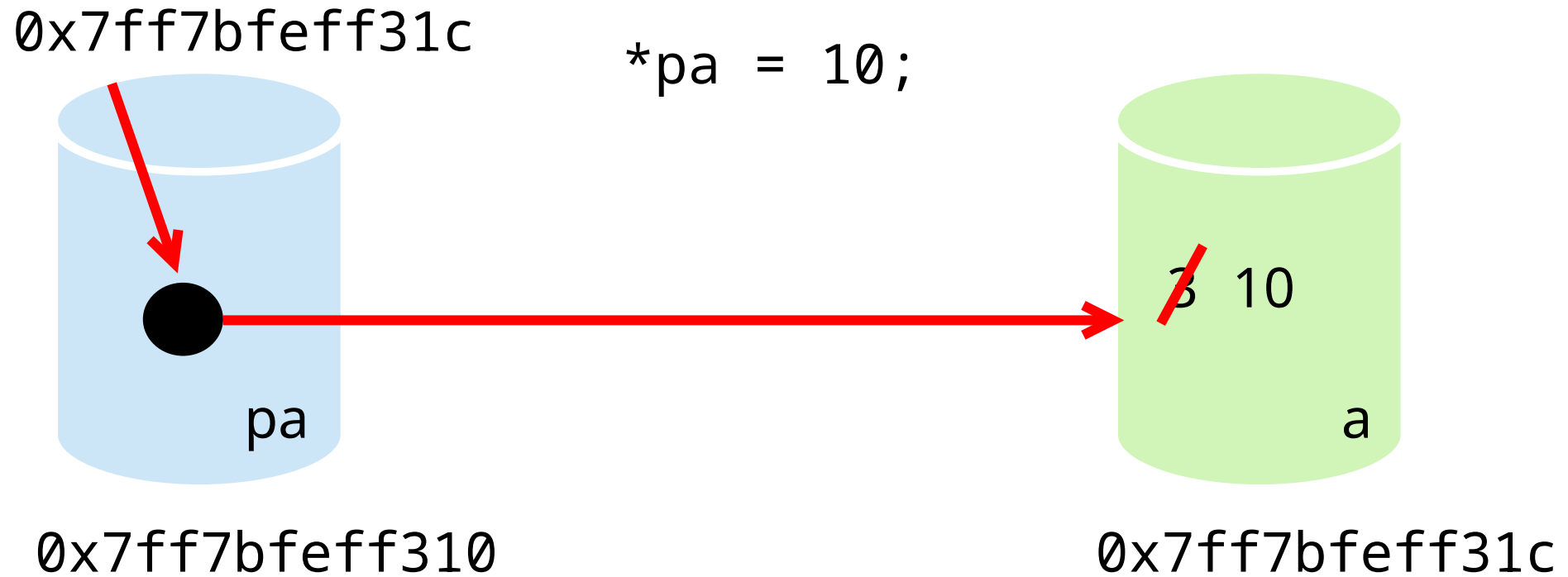
# Puntatori

## Funzionamento Base



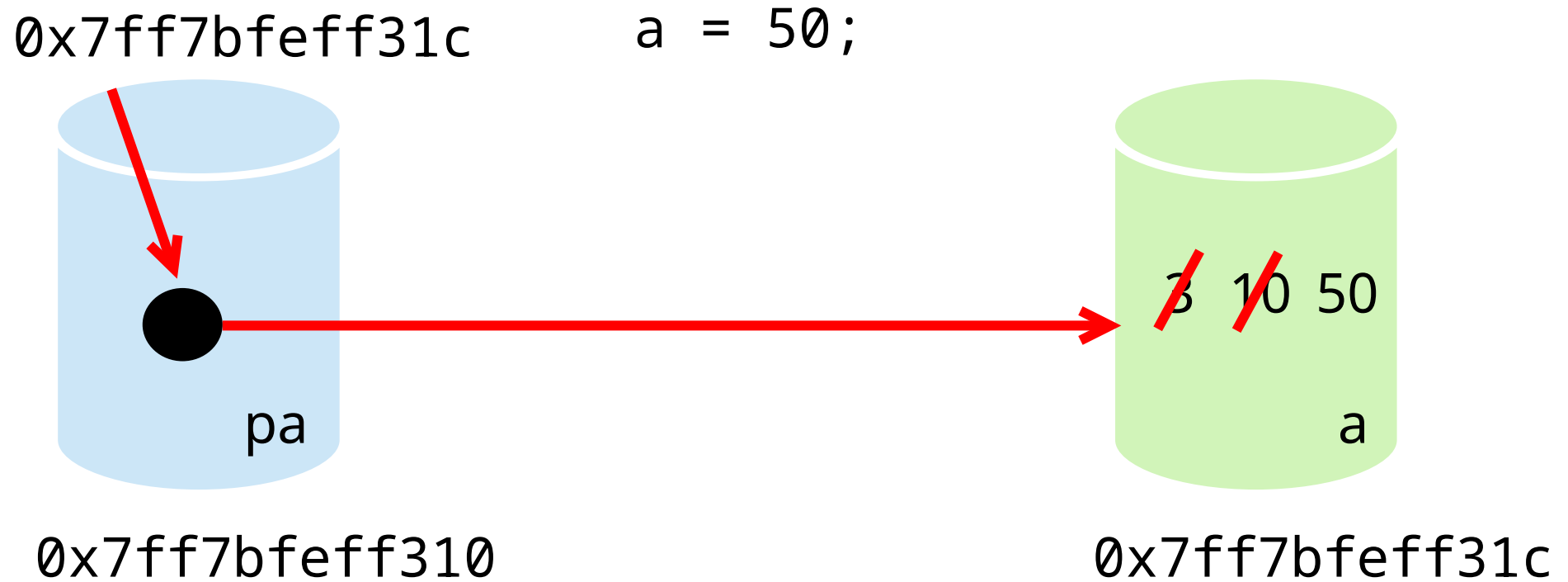
# Puntatori

## Funzionamento Base



# Puntatori

## Funzionamento Base



# **Puntatori**

Esercizi – Parte 1

# Esercizio 01

## Utilizzo Base dei puntatori

### Testo

Scrivi un programma che utilizza un puntatore per leggere dallo standard input un numero intero inserito dall'utente. Il programma raddoppia il valore inserito dall'utente utilizzando il puntatore. Stampare, prima e dopo il raddoppio del valore, le seguenti informazioni:

1. Indirizzo della variabile puntata dal puntatore
2. Indirizzo della variabile a cui punta il puntatore
3. Valore della variabile
4. Valore della variabile a cui punta il puntatore

# Stampare indirizzo di memoria

**%p**

Per stampare l'indirizzo di memoria è sufficiente usare il modificatore %p, come nella prossima slide<sup>1</sup>

1. Credits, anche per il codice della prossima slide: [https://www.w3schools.com/c/c\\_pointers.php](https://www.w3schools.com/c/c_pointers.php)

```
#include <stdio.h>

int main() {
    int myAge = 43;    // An int variable
    int* ptr = &myAge; // A pointer variable, with the name ptr,
                        // that stores the address of myAge

    // Output the value of myAge (43)
    printf("%d\n", myAge);

    // Output the memory address of myAge (0x7ffe5367e044)
    printf("%p\n", &myAge);

    // Output the memory address of myAge with the pointer
    // (0x7ffe5367e044)
    printf("%p\n", ptr);

    return 0;
}
```

```
#include <stdio.h>

int main() {
    int myAge = 43;    // An int variable
    int* ptr = &myAge; // A pointer variable, with the name ptr,
                        // that stores the address of myAge

    // Output the value of myAge (43)
    printf("%d\n", myAge);

    // Output the memory address of myAge (0x7ffe5367e044)
    printf("%p\n", &myAge);

    // Output the memory address of myAge with the pointer
    // (0x7ffe5367e044)
    printf("%p\n", ptr);

    return 0;
}
```

Il %p ci permette di avere in output l'indirizzo!

# Esercizio 02

## Swap

### Testo

Scrivi un programma che implementa il meccanismo di **swap** tra due puntatori, che scambia i valori di due variabili intere usando i puntatori. Il programma dovrà:

1. Leggere due interi da input.
2. Usare il meccanismo di **swap** per scambiare i valori.
3. Stampare i valori prima e dopo lo scambio.

# Quanto occupa un puntatore?

## Dimensioni

Considerate il seguente codice

```
#include <stdio.h>
```

```
// STAMPO DIMENSIONI DI  
DIVERSI PUNTATORI
```

```
int main() {  
    int a, *pInt;  
    char b, *pChar;  
    double c, *pDouble;  
    float d, *pFloat;  
  
    typedef struct {  
        int a;  
        char b;  
        double c;  
    } MyStruct;  
  
    MyStruct s, *pStruct;
```

# puntatore?

```
s.a=10;  
s.b='z';  
s.c=3.15;
```

```
// inizializzazione  
// puntatori  
pInt = &a;  
pChar = &b;  
pDouble = &c;  
pFloat = &d;  
pStruct = &s;
```

```
// sizeof restituisce la dimensione in byte del tipo di dato
// source: https://learn.microsoft.com/it-it/cpp/c-language/sizeof-operator-c?view=msvc-170
```

```
printf("Size of int pointer: %d bytes\n", sizeof(pInt));
printf("Size of char pointer: %d bytes\n", sizeof(pChar));
printf("Size of double pointer: %d bytes\n",
sizeof(pDouble));
printf("Size of float pointer: %d bytes\n", sizeof(pFloat));
printf("Size of MyStruct pointer: %d bytes\n",
sizeof(pStruct));
```

```
int a;
char b;
double c;
} MyStruct;
```

```
MyStruct s, *pStruct;
```

```
pChar = &b;
pDouble = &c;
pFloat = &c;
pStruct = &s;
```

Cosa ci  
aspettiamo?

```
// sizeof restituisce la dimensione in byte del tipo di dato
// source: https://learn.microsoft.com/it-it/cpp/c-language/sizeof-operator-c?view=msvc-170
```

```
printf("Size of int pointer: %d bytes\n", sizeof(pInt));
printf("Size of char pointer: %d bytes\n", sizeof(pChar));
printf("Size of double pointer: %d bytes\n",
sizeof(pDouble));
printf("Size of float pointer: %d bytes\n", sizeof(pFloat));
printf("Size of MyStruct pointer: %d bytes\n",
sizeof(pStruct));
```

```
int a;
```

```
char b;
```

```
double
```

```
} MyStruct
```

```
MyStruct
```

```
Size of int pointer: 8 bytes
```

```
Size of char pointer: 8 bytes
```

```
Size of double pointer: 8 bytes
```

```
Size of float pointer: 8 bytes
```

```
Size of MyStruct pointer: 8 bytes
```

```
pChar = &b
```

```
pDouble = &
```

```
c;
```

```
&s;
```

Cosa ci  
aspettiamo?

```
// sizeof restituisce la dimensione in byte del tipo di dato
// source: https://learn.microsoft.com/it-it/cpp/c-language/sizeof-operator-c?view=msvc-170
```

```
printf("Size of int pointer: %d bytes\n", sizeof(pInt));
printf("Size of char pointer: %d bytes\n", sizeof(pChar));
printf("Size of double pointer: %d bytes\n",
sizeof(pDouble));
printf("Size of float pointer: %d bytes\n", sizeof(pFloat));
printf("Size of MyStruct pointer: %d bytes\n",
sizeof(pStruct));
```

```
int a;
```

```
char b;
```

```
double
```

```
pChar = &b
```

```
pDouble = &
```

```
} MyStruct
MyStruct
Size of int pointer: 8 bytes
Size of char pointer: 8 bytes
Size of double pointer: 8 bytes
Size of float pointer: 8 bytes
Size of MyStruct pointer: 8 bytes
```

Cosa ci  
aspettiamo?

N.B.: architettura  
a 64 bit

```
#include <stdio.h>
```

```
// STAMPO DIMENSIONI DI  
DIVERSI PUNTATORI
```

# puntatore?

Cosa ci  
aspettiamo?

```
// Dimensione dei puntati  
printf("Size of int: %d bytes\n", sizeof(*pInt));  
printf("Size of char: %d bytes\n", sizeof(*pChar));  
printf("Size of double: %d bytes\n", sizeof(*pDouble));  
printf("Size of float: %d bytes\n", sizeof(*pFloat));  
printf("Size of MyStruct: %d bytes\n", sizeof(*pStruct));
```

```
return 0;
```

```
}
```

```
double c;  
} MyStruct;
```

```
MyStruct s, *pStruct;
```

```
pFloat = &d;  
pStruct = &s;
```

```
#include <stdio.h>
```

```
// STAMPO DIMENSIONI DI  
DIVERSI PUNTATORI
```

# puntatore?

Cosa ci  
aspettiamo?

```
// Dimensione dei puntati  
printf("Size of int: %d bytes\n", sizeof(*pInt));  
printf("Size of char: %d bytes\n", sizeof(*pChar));  
printf("Size of double: %d bytes\n", sizeof(*pDouble));  
printf("Size of float: %d bytes\n", sizeof(*pFloat));  
printf("Size of MyStruct: %d bytes\n", sizeof(*pStruct));
```

```
return 0;
```

```
}
```

```
double c;  
} MyStruct;
```

```
MyStruct s,
```

```
Size of int: 4 bytes  
Size of char: 1 bytes  
Size of double: 8 bytes  
Size of float: 4 bytes  
Size of MyStruct: 16 bytes
```

```
&d;  
&s;
```

# Se incrementassimo i puntatori?

Considerate il seguente codice

```
#include <stdio.h>
int main() {
    int var = 20;    // variabile intera
    int *p;          // puntatore a intero che verrà usato per l'incremento
    int *p_start;    // puntatore a intero per memorizzare l'indirizzo iniziale
    char *pc;        // puntatore a char che verrà usato per l'incremento
    char *pc_start;  // puntatore a char per memorizzare l'indirizzo iniziale
    char cvar = 'A'; // variabile char

    printf("Dimensioni: sizeof(int)=%d bytes, sizeof(char)=%d bytes\n",
sizeof(int), sizeof(char));

    p = &var;        // assegnazione indirizzo di var al puntatore p
    p_start = &var;  // memorizzazione dell'indirizzo iniziale
    pc = &cvar;      // assegnazione indirizzo di cvar al puntatore pc
    pc_start = &cvar; // memorizzazione dell'indirizzo iniziale
```

```
printf("##### CASO INTERO #####\n");
printf("Indirizzo iniziale del puntatore a intero: %p\n", p); //
0x7fffffffffe1b8, i primi sono a zero
p++; // incremento del puntatore a intero
printf("Indirizzo del puntatore a intero dopo incremento: %p\n", p);
// devo moltiplicare per sizeof int, altrimenti ritorna il numero di
posizioni incrementate
// La dimostrazione ne è la stampa delle posizioni
printf("Stampo la differenza come un indirizzo: %p \n", (p -
p_start)*sizeof(int));
printf("Stampo la differenza come un intero (num. posizioni): %d \n", p -
p_start);
```

Cosa ci  
aspettiamo?

```
printf("##### CASO INTERO #####\n");
printf("Indirizzo iniziale del puntatore a intero: %p\n", p); //
0x7fffffffffe1b8, i primi sono a zero
p++; // incremento del puntatore a intero
printf("Indirizzo del puntatore a intero dopo incremento: %p\n", p);
// devo moltiplicare per sizeof int, altrimenti ritorna il numero di
posizioni incrementate
// La dimostrazione ne è la stampa delle posizioni
printf("Stampo la differenza come un indirizzo: %p \n", (p -
p_start)*sizeof(int));
printf("Stampo la differenza come un intero (num. posizioni): %d \n", p -
p_start);
```

Cosa ci

Dimensioni: sizeof(int)=4 bytes, sizeof(char)=1 bytes

##### CASO INTERO #####

Indirizzo iniziale del puntatore a intero: 0x7fffffffffe1b4

Indirizzo del puntatore a intero dopo incremento: 0x7fffffffffe1b8

Stampo la differenza come un indirizzo: 0x4

Stampo la differenza come un intero (num. posizioni): 1

```

printf("##### CASO INTERO #####\n");
printf("Indirizzo iniziale del puntatore a intero: %p\n", p); //
0x7fffffffffe1b8, i primi sono a zero
p++; // incremento del puntatore a intero
printf("Indirizzo del puntatore a intero dopo incremento: %p\n", p);
// devo moltiplicare per sizeof int, altrimenti ritorna il numero di
posizioni incrementate
// La dimensione di un intero è 4 bytes, la stampa delle posizioni
printf("Stampo la differenza come un indirizzo: %p \n", (p -
p_start)*sizeof(int));
printf("Stampo la differenza come un intero (num. posizioni): %d \n", p -
p_start);

```

Dimensione  
intero: 4 bytes

Cosa ci

Dimensioni: sizeof(int)=4 bytes, sizeof(char)=1 bytes

##### CASO INTERO #####

Indirizzo iniziale del puntatore a intero: 0x7fffffffffe1b4

Indirizzo del puntatore a intero dopo incremento: 0x7fffffffffe1b8

Stampo la differenza come un indirizzo: 0x4

Stampo la differenza come un intero (num. posizioni): 1

```

printf("##### CASO INTERO #####\n");
printf("Indirizzo iniziale del puntatore a intero: %p\n", p); //
0x7fffffffffe1b8, i primi sono a zero
p++; // incremento del puntatore a intero
printf("Indirizzo del puntatore a intero dopo incremento: %p\n", p);
// devo moltiplicare per sizeof int, per ottenere il numero di
posizioni incrementate
// La dimostrazione ne è la stampa della differenza
printf("Stampo la differenza come un indirizzo: %p \n", (p -
p_start)*sizeof(int));
printf("Stampo la differenza come un intero (num. posizioni): %d \n", p -
p_start);

```

L'indirizzo, stampato  
con %p, è  
incrementato di 4  
bytes!

Cosa ci

Dimensioni: sizeof(int)=4 bytes, sizeof(char)=1 bytes

##### CASO INTERO #####

Indirizzo iniziale del puntatore a intero: 0x7fffffffffe1b4

Indirizzo del puntatore a intero dopo incremento: 0x7fffffffffe1b8

Stampo la differenza come un indirizzo: 0x4

Stampo la differenza come un intero (num. posizioni): 1

```

printf("##### CASO INTERO #####\n")
printf("Indirizzo iniziale del puntatore a intero: 0x7fffffffefe1b4, i primi sono a zero\n")
p++; // incremento del puntatore a intero
printf("Indirizzo del puntatore a intero dopo incremento: 0x7fffffffefe1b8\n")
// devo moltiplicare per sizeof(int) per ottenere il numero di posizioni incrementate
// La dimostrazione ne è la stampa delle posizioni
printf("Stampo la differenza come un indirizzo: %p \n", (p - p_start)*sizeof(int));
printf("Stampo la differenza come un intero (num. posizioni): %d \n", p - p_start);

```

Questa ritorna il numero di posizioni incrementate!

Cosa ci

Dimensioni: sizeof(int)=4 bytes, sizeof(char)=1 bytes

##### CASO INTERO #####

Indirizzo iniziale del puntatore a intero: 0x7fffffffefe1b4

Indirizzo del puntatore a intero dopo incremento: 0x7fffffffefe1b8

Stampo la differenza come un indirizzo: 0x4

Stampo la differenza come un intero (num. posizioni): 1

```

printf("##### CASO INTERO #####\n")
printf("Indirizzo iniziale del puntatore a intero: 0x7fffffffefe1b4, i primi sono a zero\n")
p++; // incremento del puntatore a intero
printf("Indirizzo del puntatore a intero dopo incremento: 0x7fffffffefe1b8\n")
// devo moltiplicare per sizeof(int) per ottenere il numero di
posizioni incrementate
// la dimostrazione ne è la stampa delle posizioni

```

Questa ritorna il numero di posizioni incrementate!

**Infatti, per ottenere i 4 bytes, devo moltiplicare per la dimensione dell'intero**

```

printf("Stampo la differenza come un indirizzo: %p \n", (p - p_start)*sizeof(int));

```

```

printf("Stampo la differenza come un intero (num. posizioni): %d \n", p - p_start);

```

Cosa ci

Dimensioni: sizeof(int)=4 bytes, sizeof(char)=1 bytes

##### CASO INTERO #####

Indirizzo iniziale del puntatore a intero: 0x7fffffffefe1b4

Indirizzo del puntatore a intero dopo incremento: 0x7fffffffefe1b8

Stampo la differenza come un indirizzo: 0x4

Stampo la differenza come un intero (num. posizioni): 1

S

C

```
printf("\n##### CASO CHAR #####\n");
printf("Indirizzo iniziale del puntatore a char: %p\n", pc);
pc++; // incremento del puntatore a char
printf("Indirizzo del puntatore a char dopo incremento: %p\n", pc);
printf("Stampo la differenza come un indirizzo: %p \n", (pc -
pc_start)*sizeof(char));
printf("Stampo la differenza come un intero (num. posizioni): %d \n",
pc - pc_start);
return 0;
}
```

Cosa ci aspettiamo?

Ricorda: sizeof(char)=1 bytes

S

C

```
printf("\n##### CASO CHAR #####\n");
printf("Indirizzo iniziale del puntatore a char: %p\n", pc);
pc++; // incremento del puntatore a char
printf("Indirizzo del puntatore a char dopo incremento: %p\n", pc);
printf("Stampo la differenza come un indirizzo: %p \n", (pc -
pc_start)*sizeof(char));
printf("Stampo la differenza come un intero (num. posizioni): %d \n",
pc - pc_start);
return 0;
}
```

Cosa ci aspettiamo?

Ricorda: sizeof(char)=1 bytes

##### CASO CHAR #####

Indirizzo iniziale del puntatore a char: 0x7fffffffef1b3

Indirizzo del puntatore a char dopo incremento: 0x7fffffffef1b4

Stampo la differenza come un indirizzo: 0x1

Stampo la differenza come un intero: 1

S

C

```
printf("\n##### CASO CHAR #####\n");
printf("Indirizzo iniziale del puntatore a char: %p\n", pc);
pc++; // incremento del puntatore a char
printf("Indirizzo del puntatore a char dopo incremento: %p\n", pc);
printf("Stampo la differenza come un indirizzo: %p \n", (pc -
pc_start)*sizeof(char));
printf("Stampo la differenza come un intero (num. posizioni): %d \n",
pc - pc_start);
return 0;
}
```

Nota l'incremento di un byte

##### CASO CHAR #####

Indirizzo iniziale del puntatore a char: 0x7fffffffefe1b3

Indirizzo del puntatore a char dopo incremento: 0x7fffffffefe1b4

Stampo la differenza come un indirizzo: 0x1

Stampo la differenza come un intero: 1

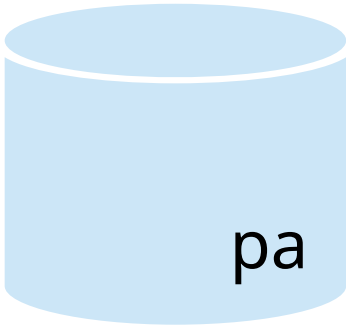
# **Puntatori e Array**

Recap

# Puntatori

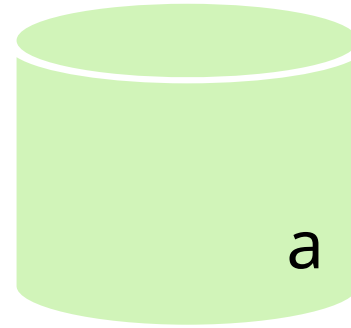
## Puntatori e Array

`int *pa;`



`0x7ff7bfeff310`

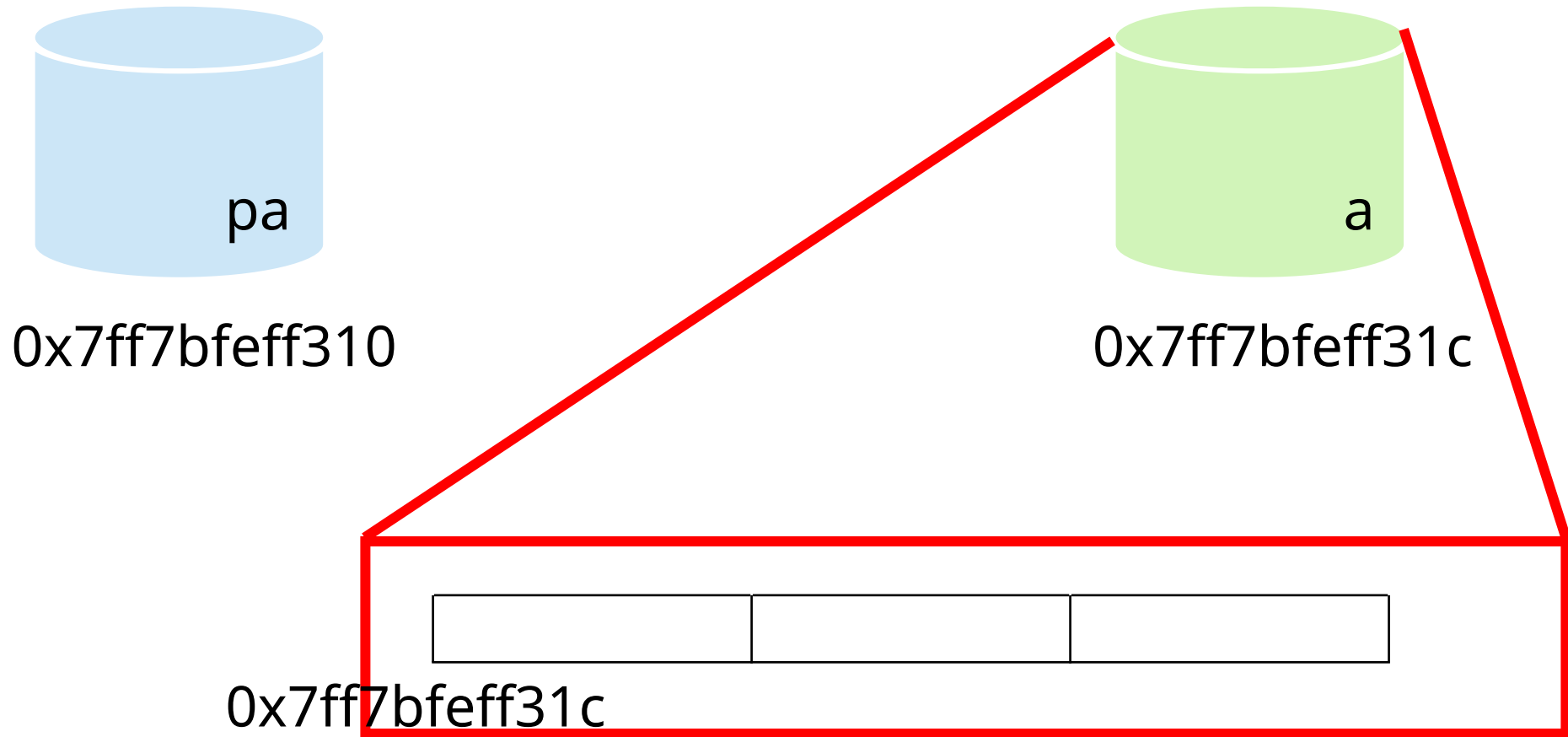
`int a[3];`



`0x7ff7bfeff31c`

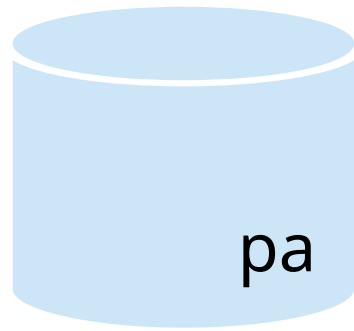
# Puntatori

## Puntatori e Array



# Puntatori

## Puntatori e Array



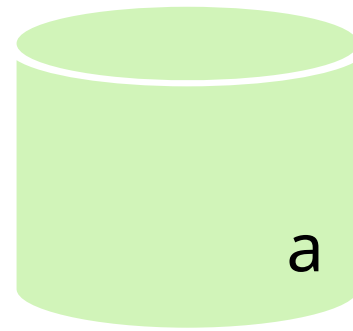
pa

0x7ff7bfeff310

Il valore in cui "a"  
decade si comporta  
come un puntatore

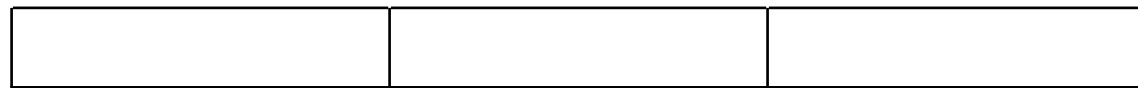
**costante**

(Non gli si può  
cambiare valore, ma  
si possono cambiare  
gli elementi di a)



a

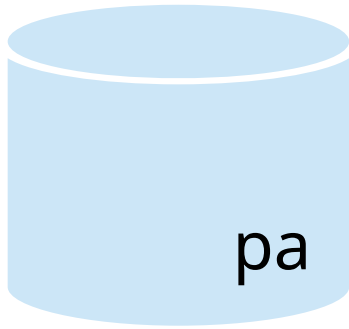
0x7ff7bfeff31c



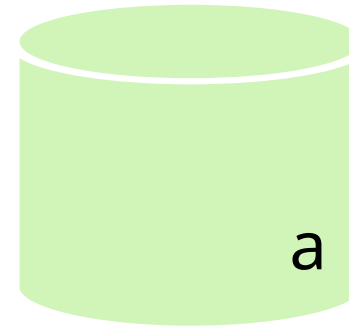
0x7ff7bfeff31c

# Puntatori

## Puntatori e Array



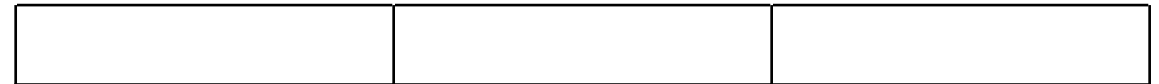
0x7ff7bfeff310



0x7ff7bfeff31c

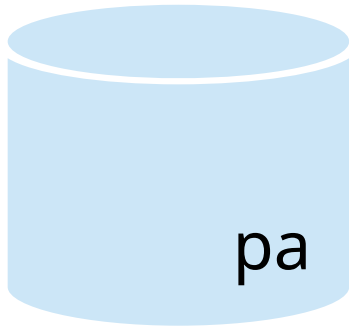
“a” è l'array, e quando si usa “a” in un'espressione, *decade* (il compilatore lo sostituisce con) nell'indirizzo della prima cella

0x7ff7bfeff31c

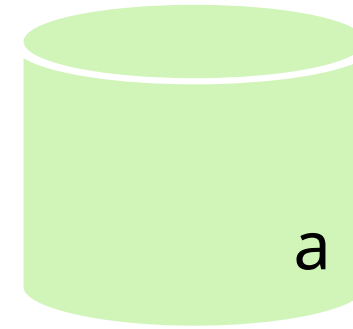


# Puntatori

## Puntatori e Array



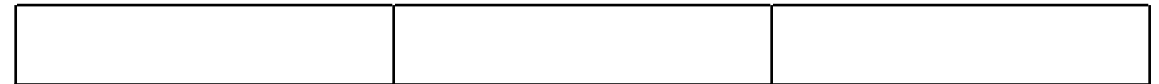
0x7ff7bfeff310



0x7ff7bfeff31c

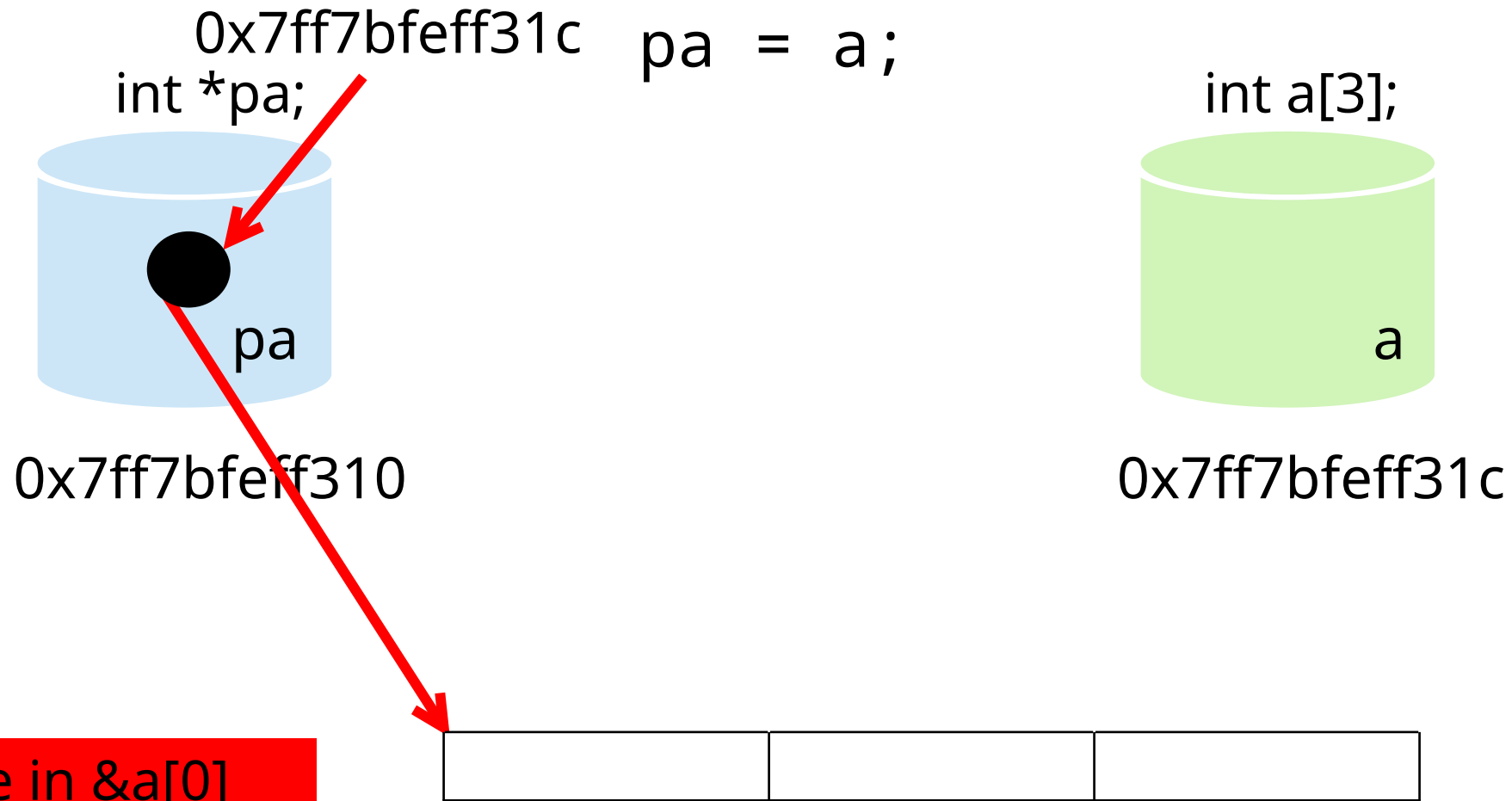
“a” è l'array, e quando si usa “a” in un'espressione, *decade* (il compilatore lo sostituisce con) nell'indirizzo della prima cella

0x7ff7bfeff31c



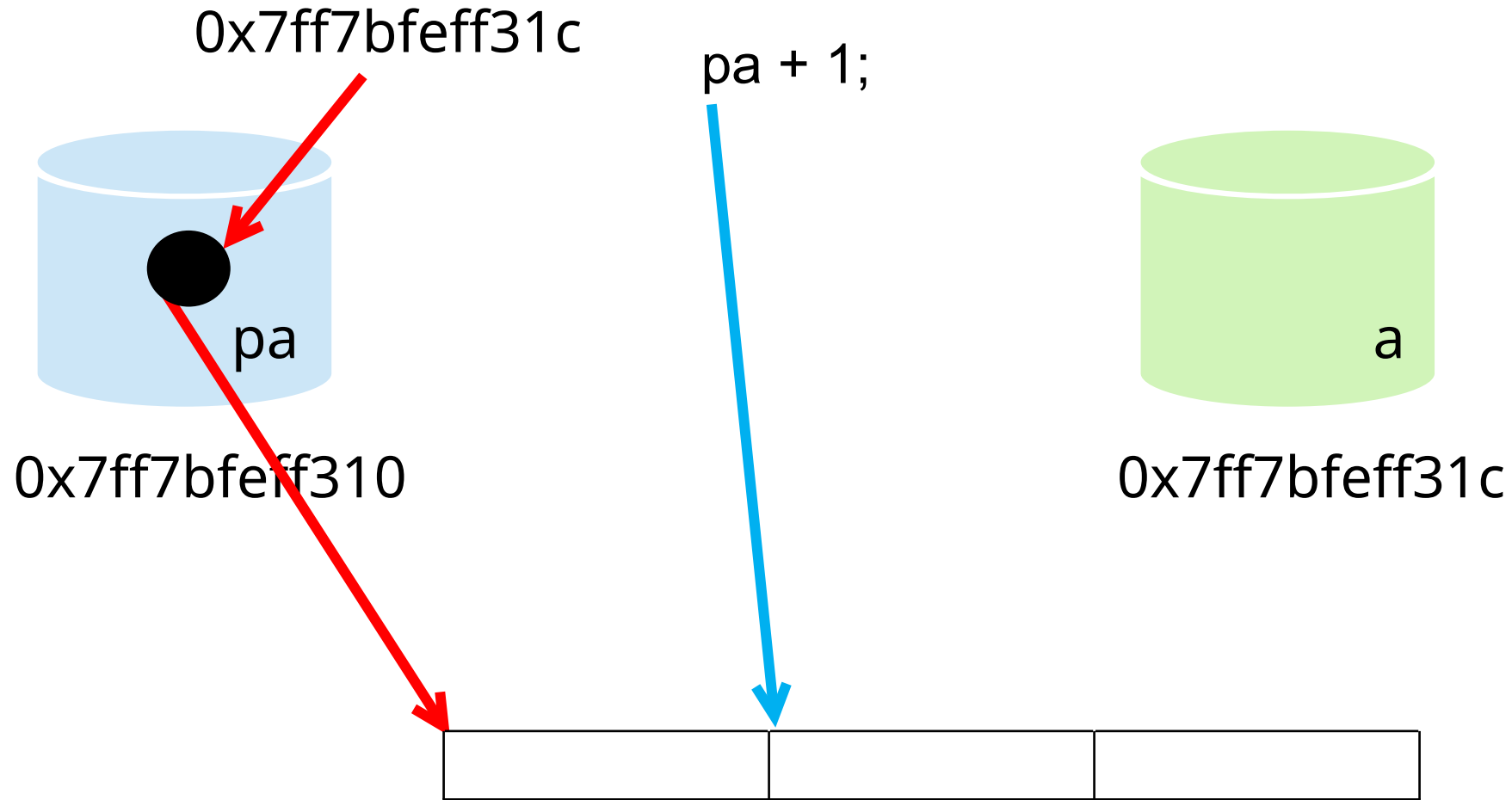
# Puntatori

## Puntatori e Array



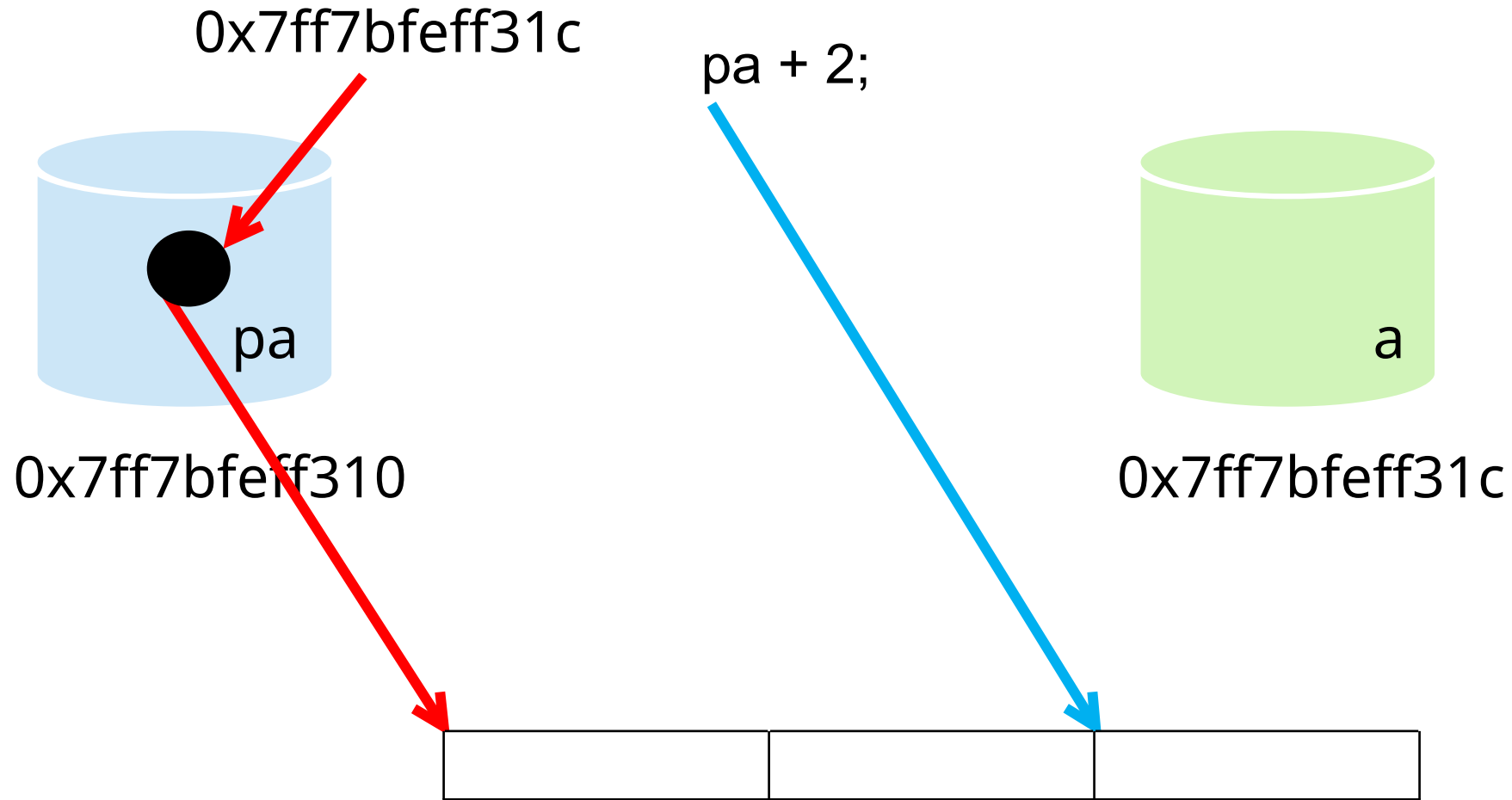
# Puntatori

## Puntatori e Array



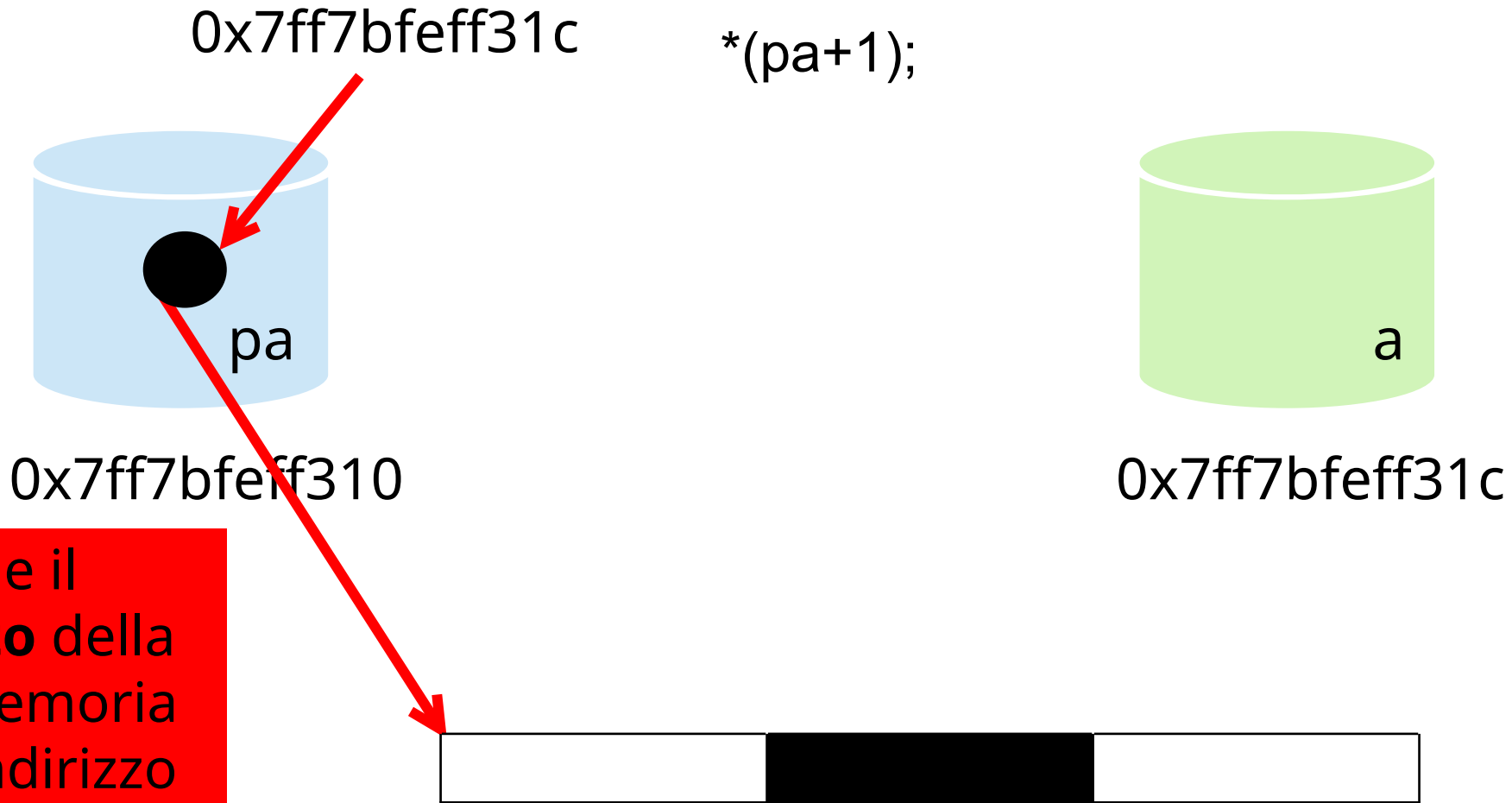
# Puntatori

## Puntatori e Array



# Puntatori

## Puntatori e Array



Prende il  
**contenuto** della  
cella di memoria  
che ha l'indirizzo  
pa+1  
Come a[1]

# **Puntatori**

Esercizi – Parte 2

# Esercizio 03

## Inserimento, Somma, Min e Max di Array

### Testo

Scrivere un programma che dichiari un array di interi di 5 elementi e che usa un puntatore per:

1. Leggere i valori degli elementi dell'array dallo standard input
2. Sommare tutti gli elementi dell'array
3. Trovare il minimo valore dell'array
4. Trovare il massimo valore dell'array
5. Stampare la somma, il minimo e il massimo

# Esercizio 04

## Lunghezza Stringa

### Testo

Scrivi un programma che chiede all'utente di inserire una stringa e poi:

1. Usa un puntatore per contare il numero di caratteri nella stringa (senza usare strlen()).
2. Usa il puntatore per stampare la stringa in ordine inverso

# Esercizio 05

## Inversione di un array

### Testo

Scrivi un programma che chiede all'utente di inserire un array di 10 interi positivi, sfruttando **solo** l'aritmetica dei puntatori.

Il programma deve stamparne quindi l'inverso, di nuovo usando **solo** l'aritmetica dei puntatori.

# Esercizio 06

## Doppio di un array

### Testo

Si scriva un programma che consenta all'utente di inserire un array di 5 interi positivi. Il programma deve salvare in un secondo array di stessa dimensione il doppio di ogni elemento.

Si usi **solo** l'aritmetica dei puntatori

# Esercizio 07

## Pari in pari e Dispari in Dispari

### Testo

Si scriva un programma sfruttando solo l'aritmetica dei puntatori che, in un array di 10 interi positivi acquisito da input, conti quanti elementi pari ci sono in posizione pari, e quanti dispari in posizione dispari (contare la posizione "0" come pari).

### Esempio:

[5,1,3,4,14,16,3,80,191,11]

Pari in pari: 1 (il 14)

Dispari in dispari: 2 (3 e 11)

# Esercizio 07bis

## Pari in pari e Dispari in Dispari

### Testo

Dopo il programma precedente, salvare i “pari in pari” ed i “dispari in dispari” in due nuovi array

### Esempio:

[5,1,3,4,14,16,3,80,191,11]

Pari in pari: 1 (il 14) -> pari\_in\_pari = contiene il 14 in posizione 0

Dispari in dispari: 2 (3 e 11) -> dispari\_in\_dispari = contiene 3 e 11 in posizione 0 e 1

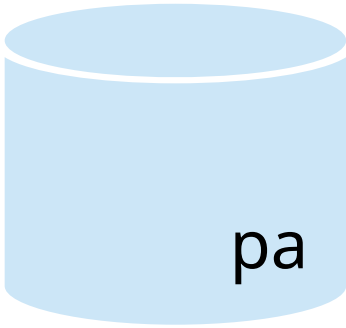
# **Puntatori e Matrici**

Recap

# Puntatori

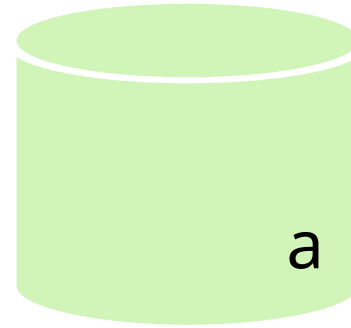
## Puntatori e Matrici

`int *pa;`



0x7ff7bfeff310

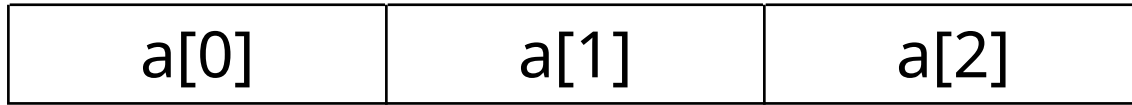
`int a[3][3];`



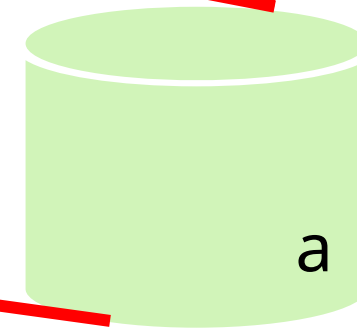
0x7ff7bfeff31c

# Puntatori

## Puntatori e Matrici



`int a[3][3];`

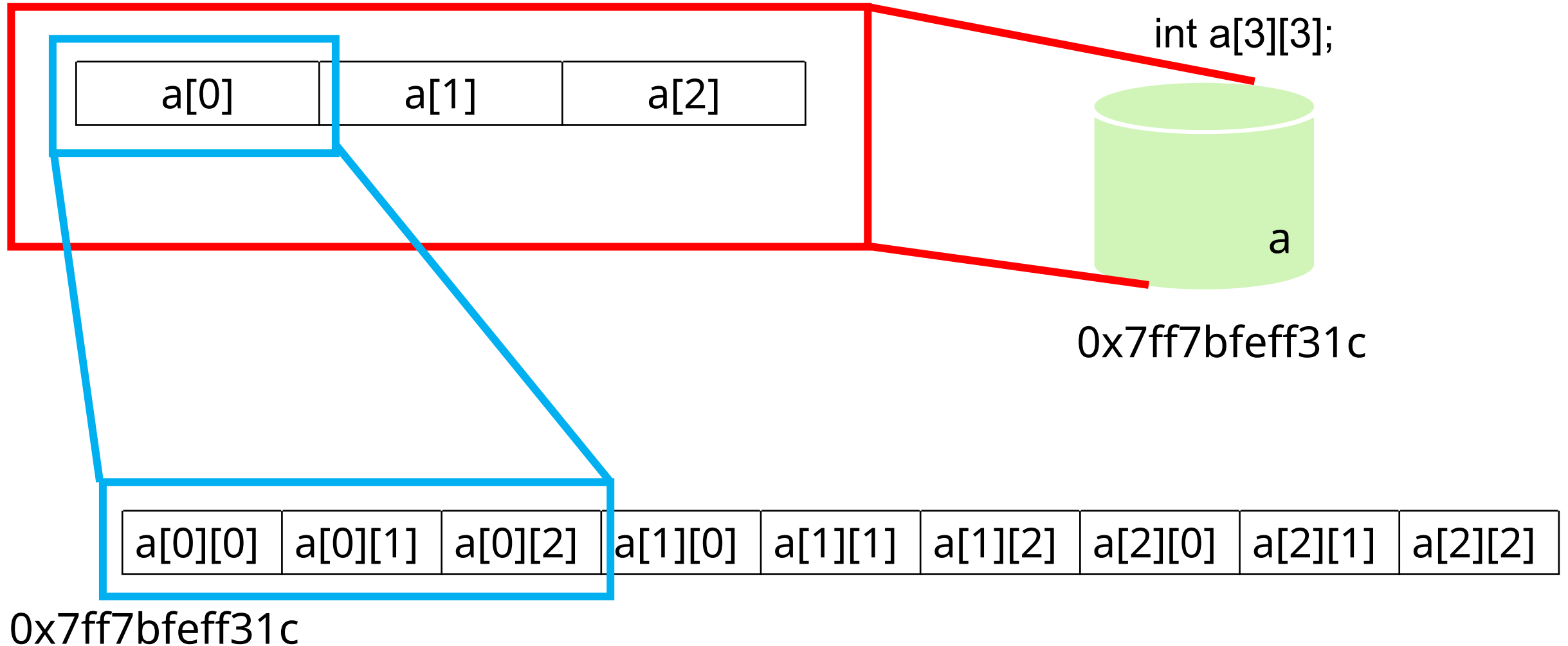


`0x7ff7bfeff31c`

La matrice è un array di array, cioè un blocco contiguo di 9 elementi `int` per come è stato dichiarato, posto all'indirizzo `0x7ff7bfeff31c`

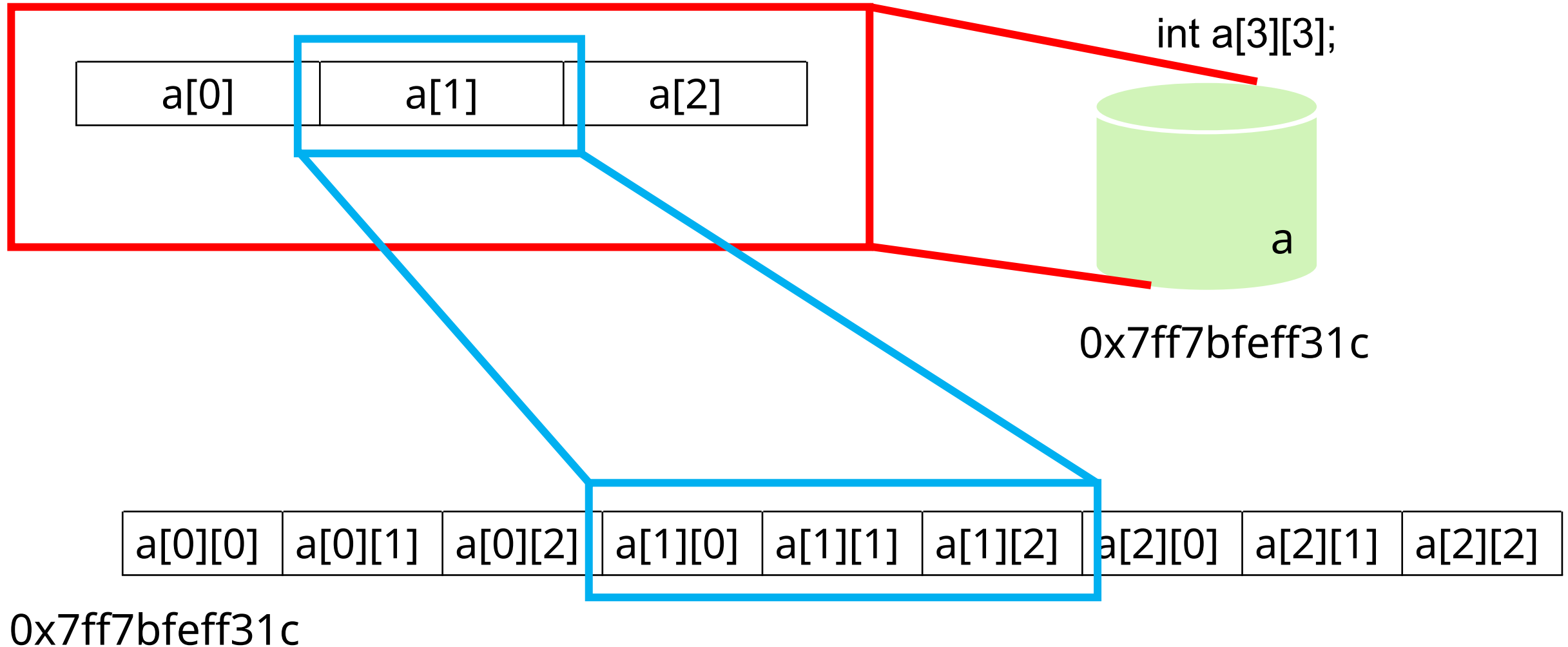
# Puntatori

## Puntatori e Matrici



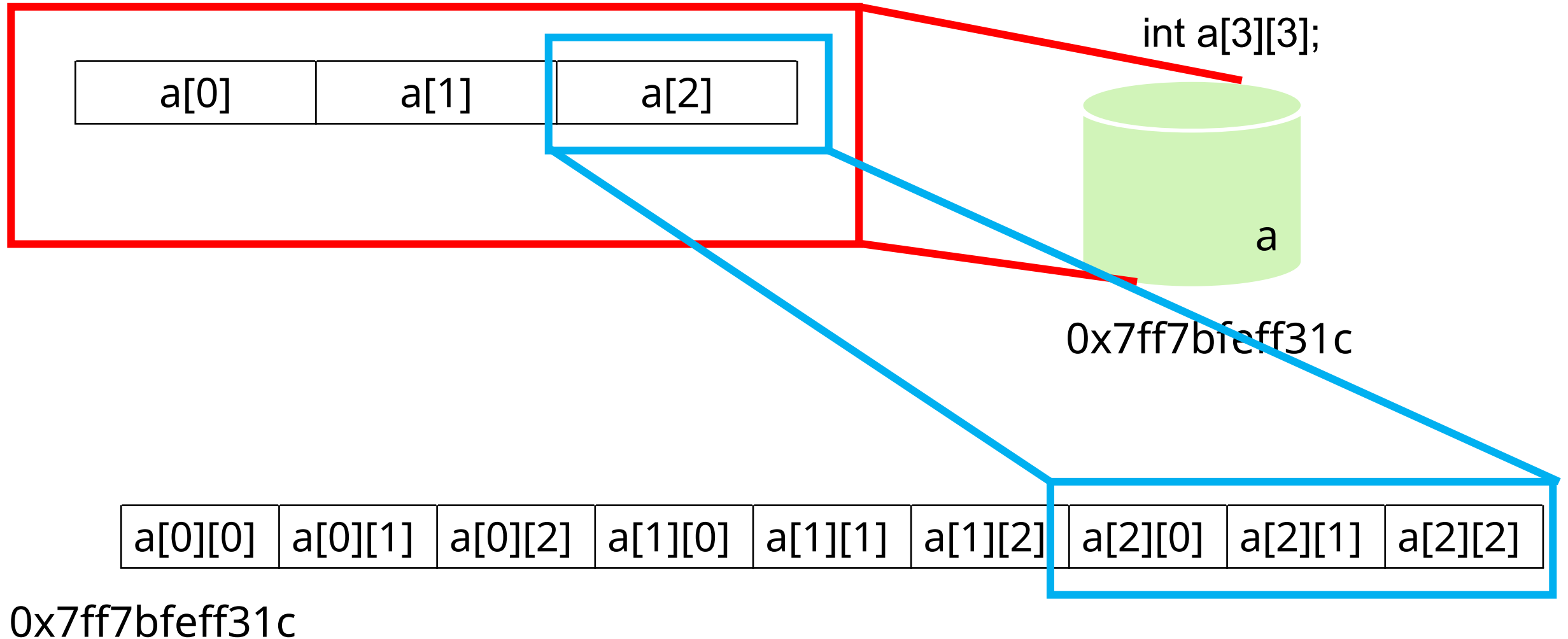
# Puntatori

## Puntatori e Matrici



# Puntatori

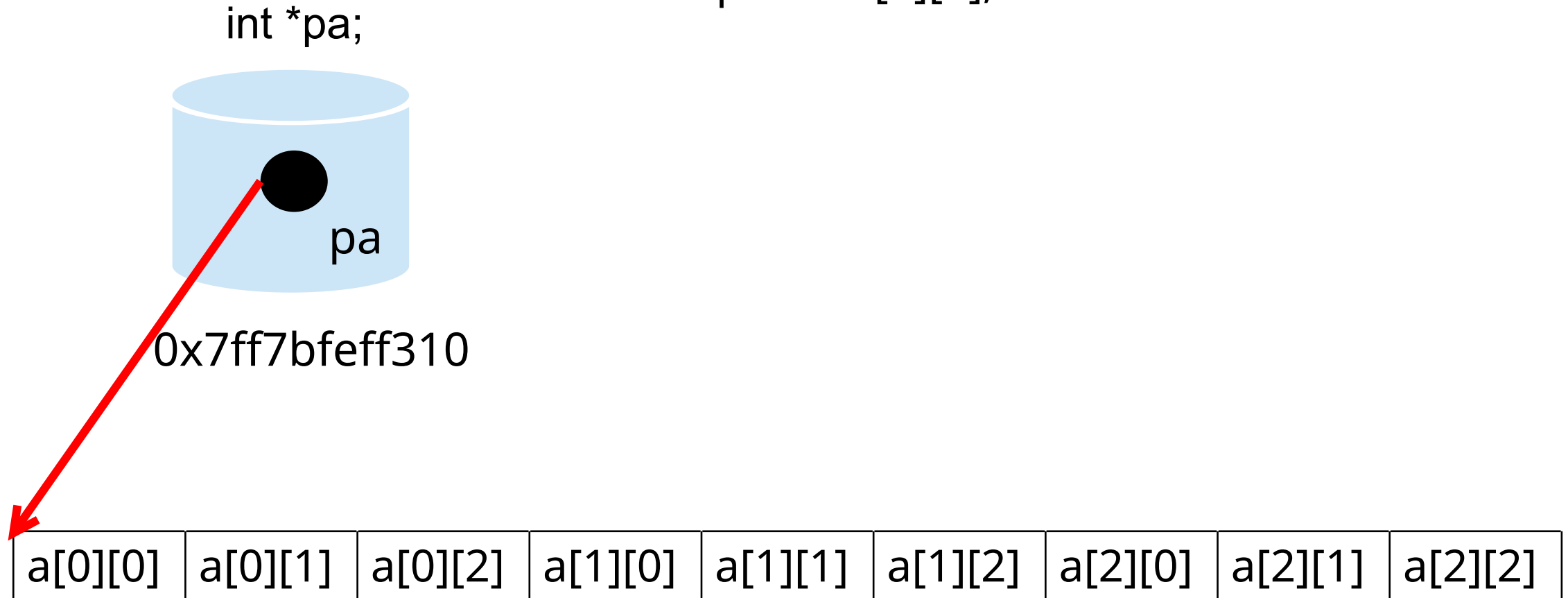
## Puntatori e Matrici



# Puntatori

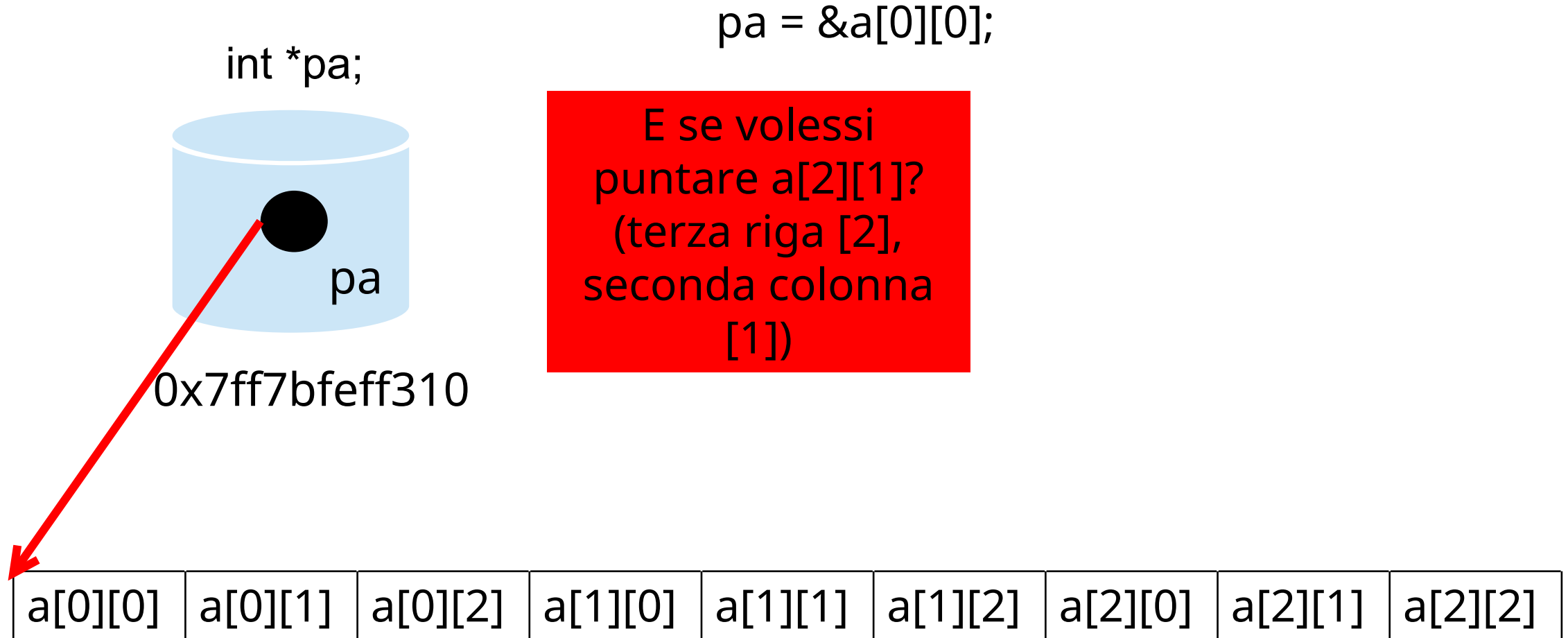
## Puntatori e Matrici

```
pa = &a[0][0];
```



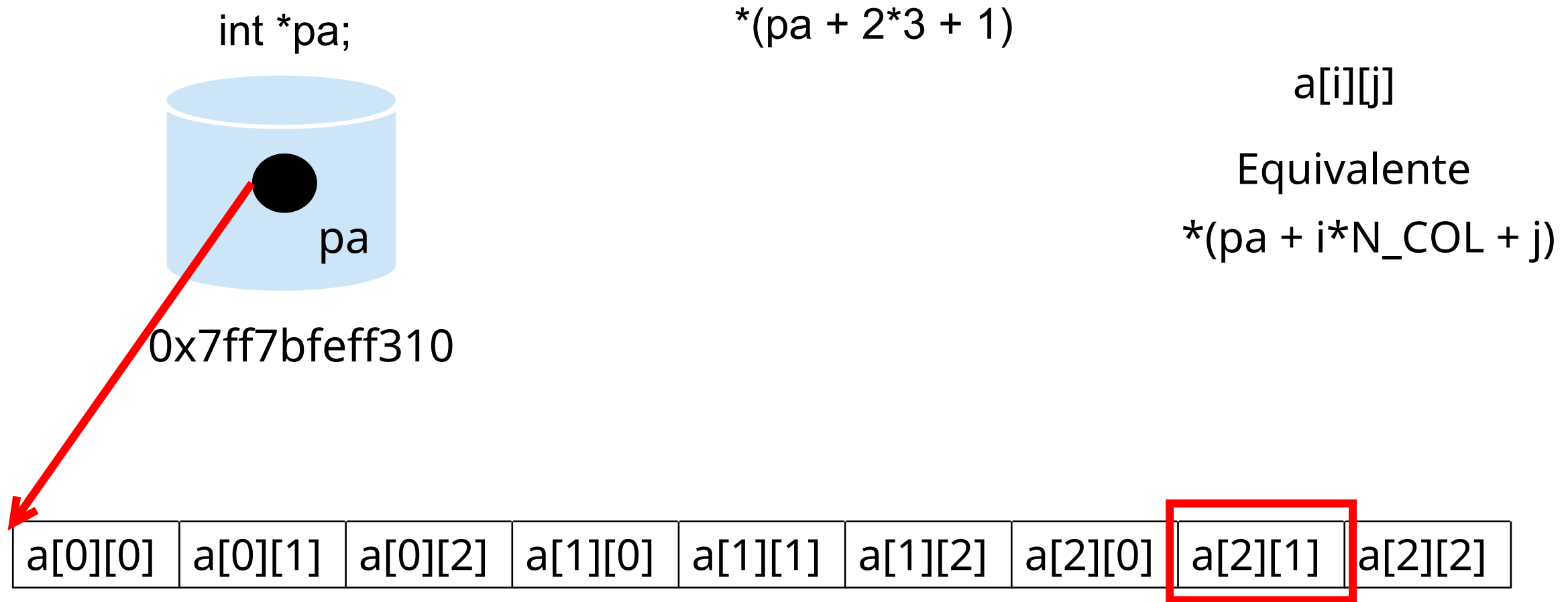
# Puntatori

## Puntatori e Matrici



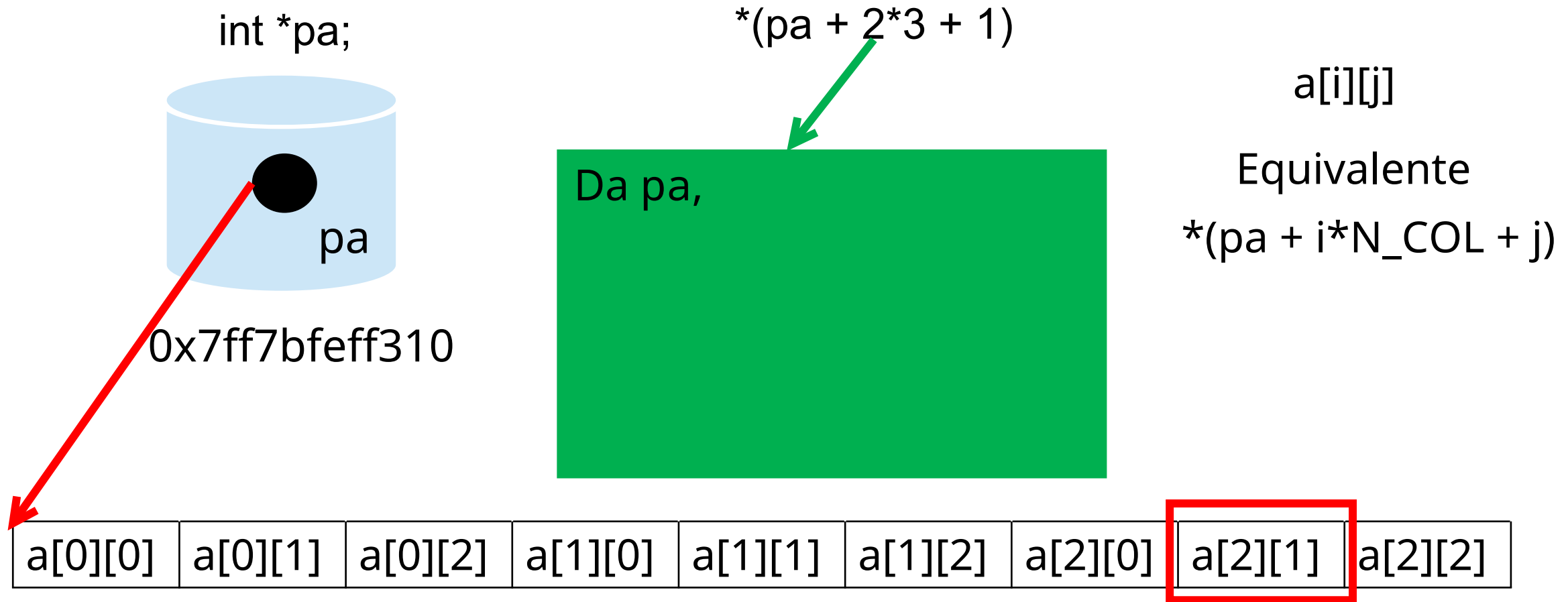
# Puntatori

## Puntatori e Matrici



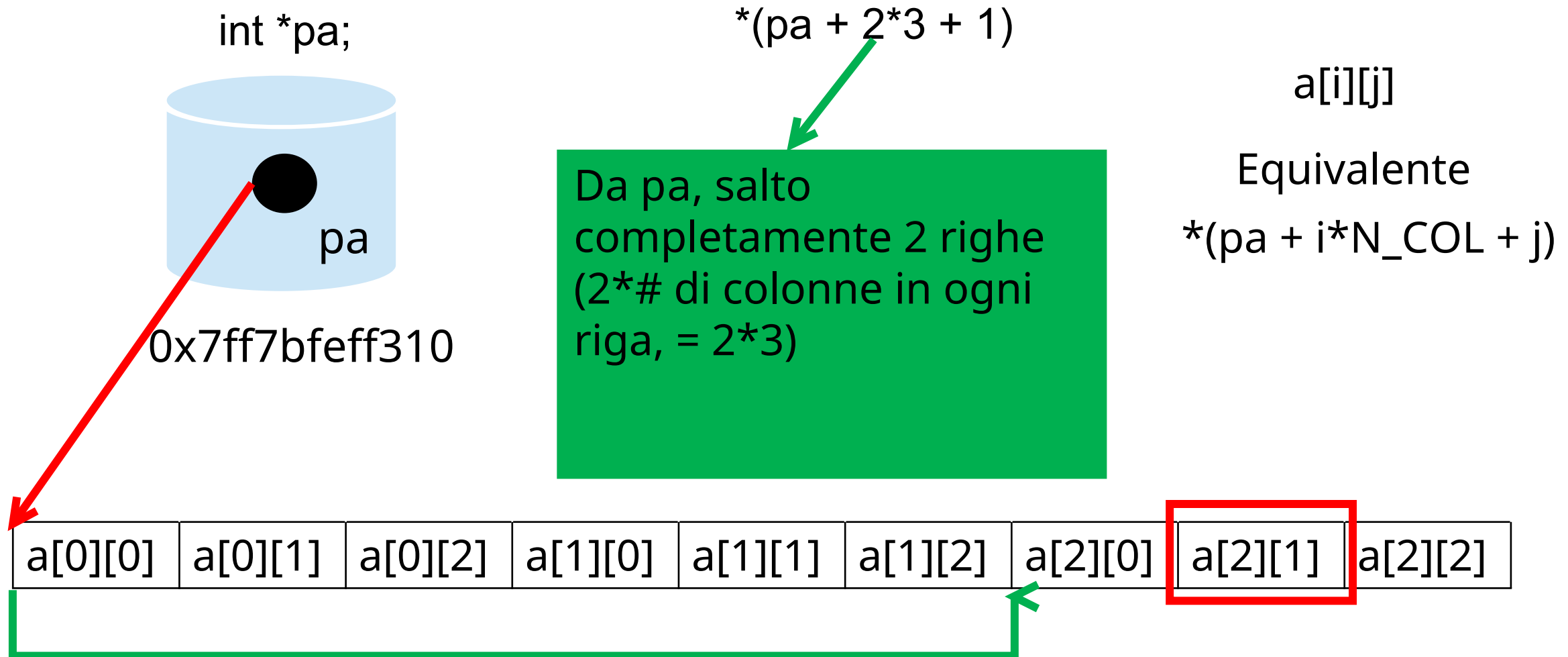
# Puntatori

## Puntatori e Matrici



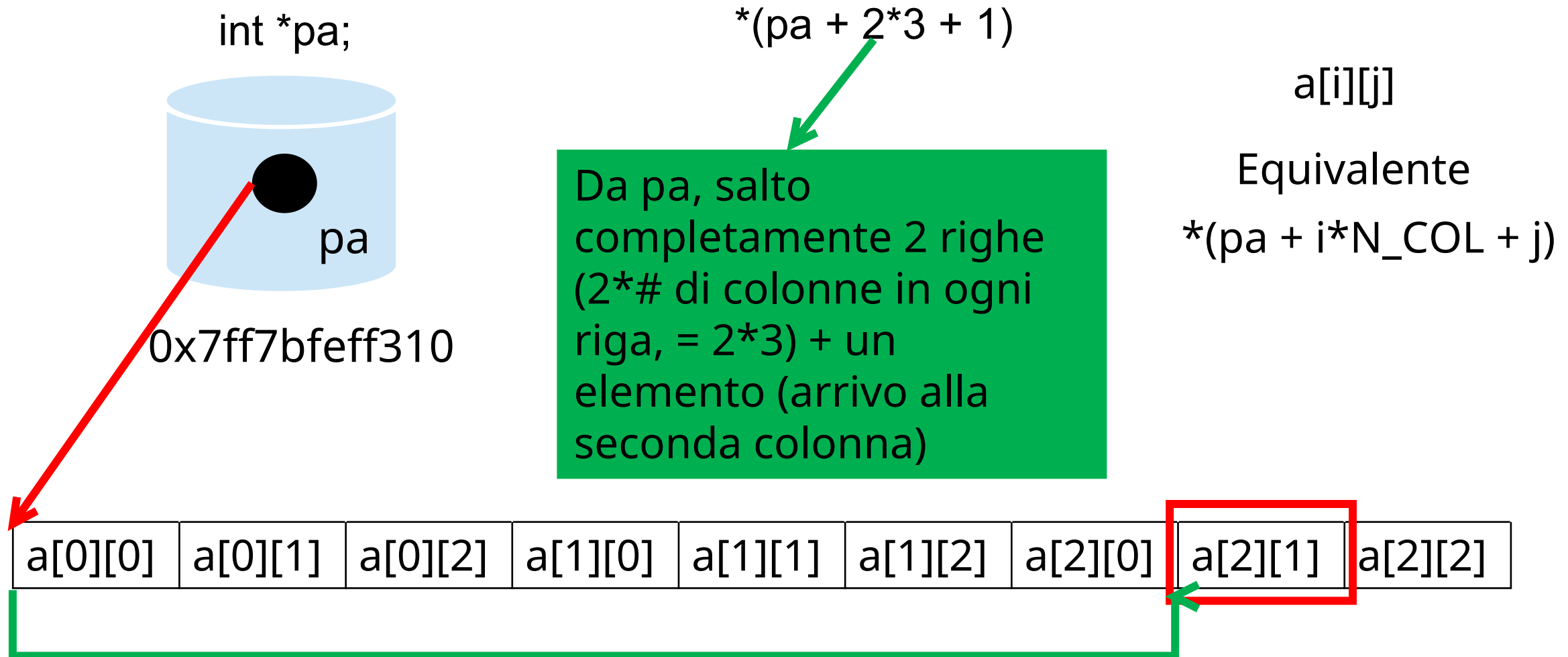
# Puntatori

## Puntatori e Matrici



# Puntatori

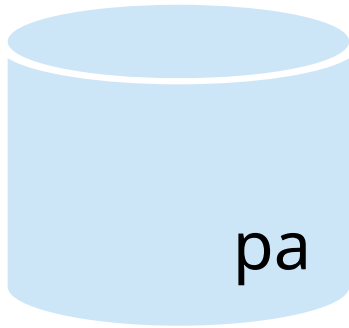
## Puntatori e Matrici



# Puntatori

## Puntatori e Matrici

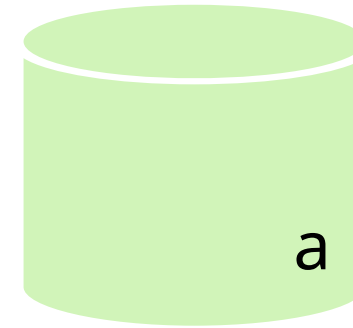
`int *pa[3];`



0x7ff7bfeff310

In questo caso  
vogliamo invece  
definire pa come  
un array  
contenente  
N\_RIGHE puntatori  
a intero

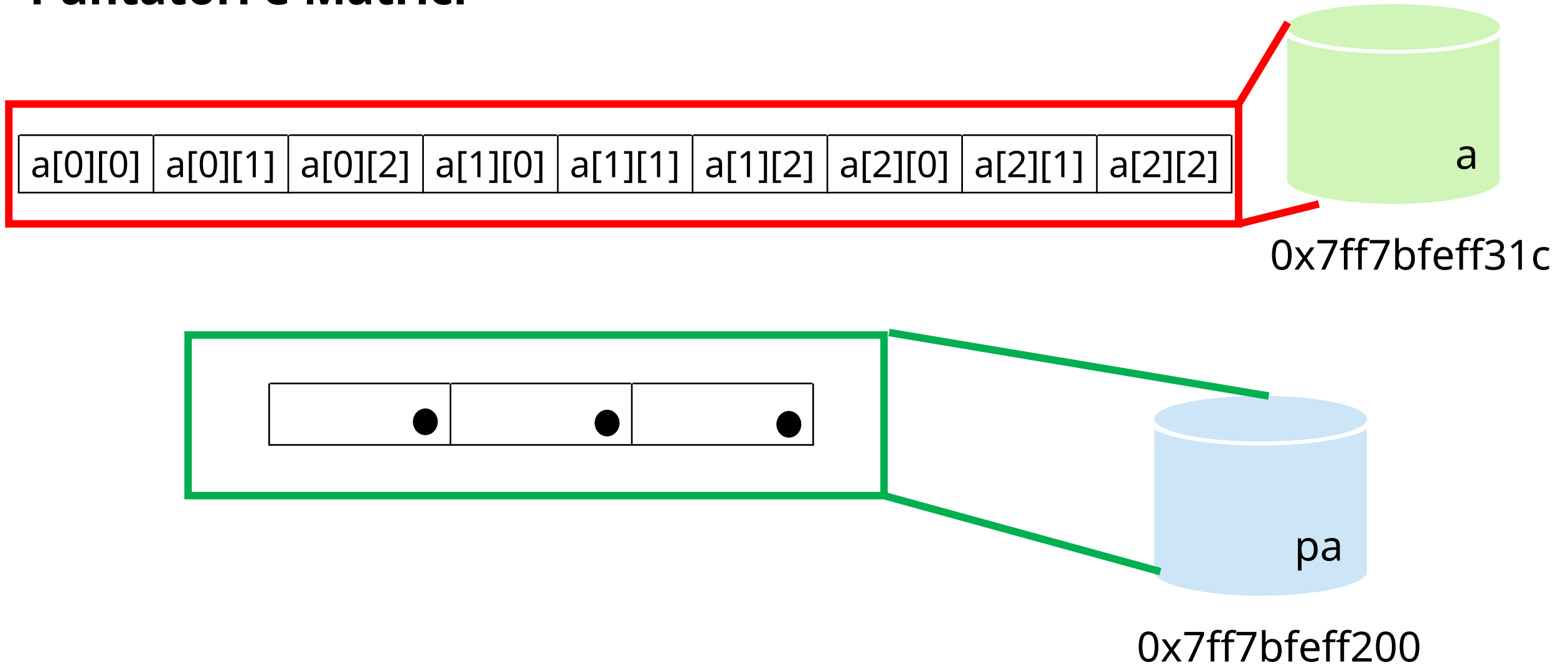
`int a[3][3];`



0x7ff7bfeff31c

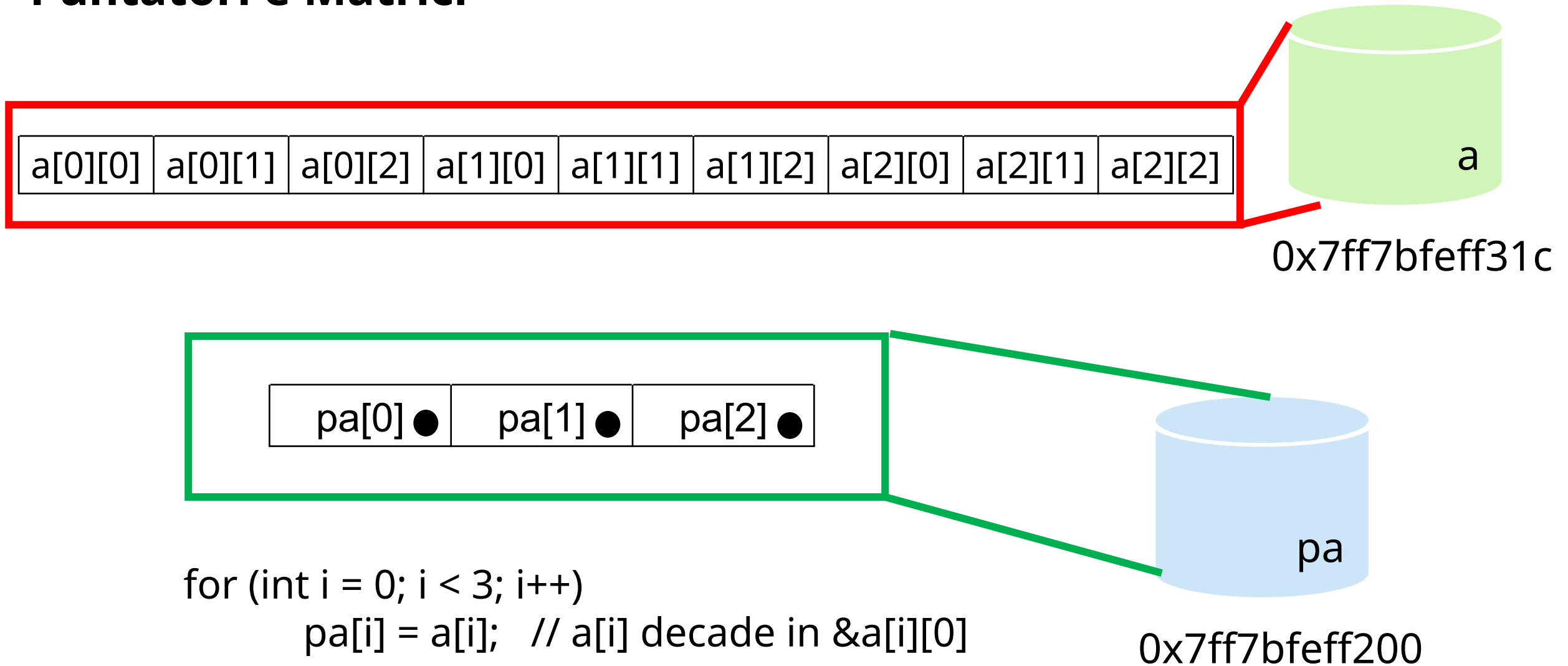
# Puntatori

## Puntatori e Matrici



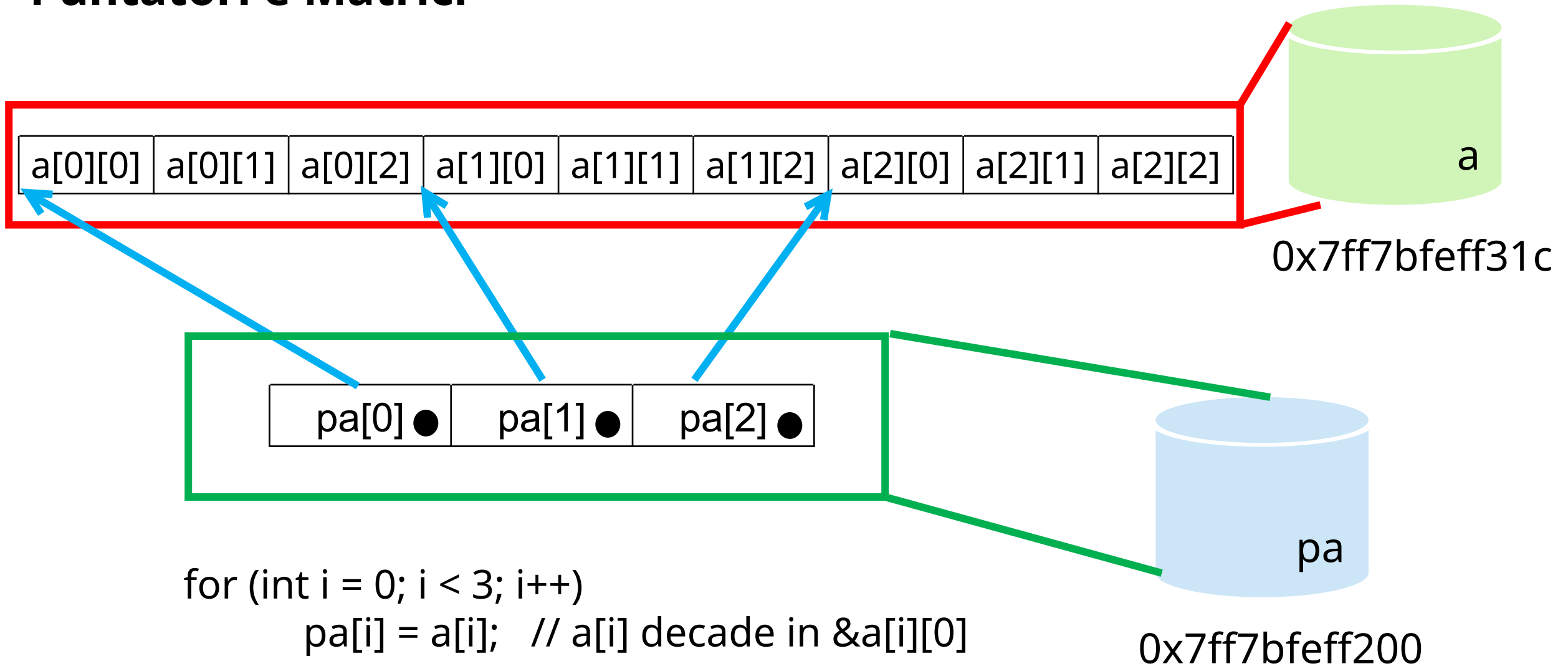
# Puntatori

## Puntatori e Matrici



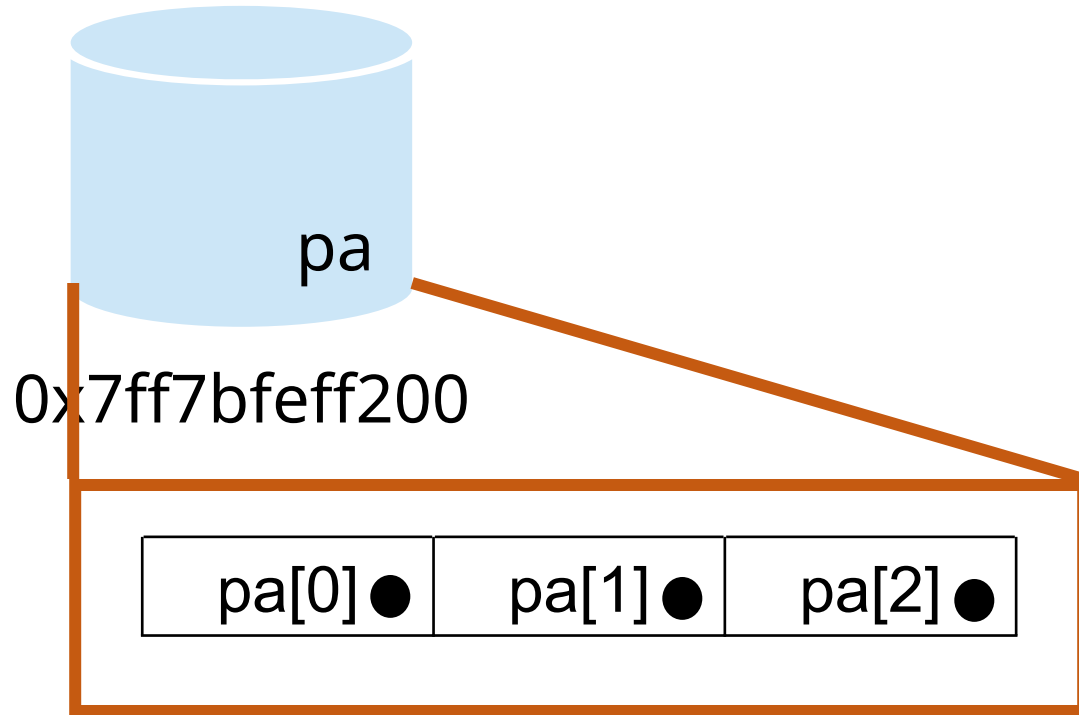
# Puntatori

## Puntatori e Matrici



# Puntatori

## Puntatori e Matrici



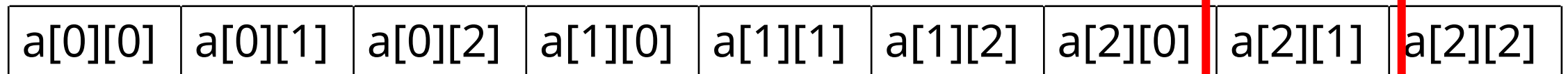
$*(*(pa + 2) + 1)$

$a[i][j]$

Equivalente

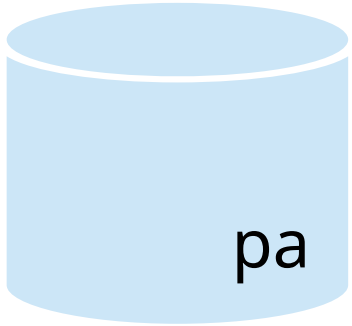
$*(*(pa + i) + j)$

Ognuno di questi  
puntatori punta ad  
una riga di a



# Puntatori

## Puntatori e Matrici



0x7ff7bfeff200

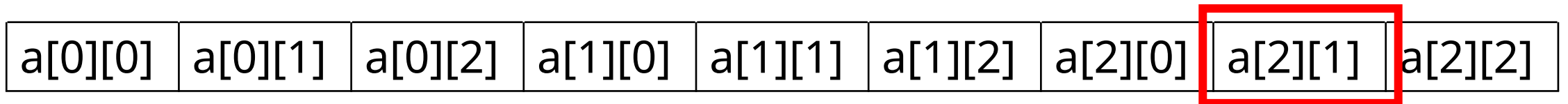


$*(*(pa + 2) + 1)$

$a[i][j]$

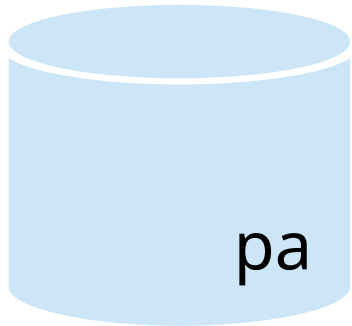
Equivalente

$*(*(pa + i) + j)$



# Puntatori

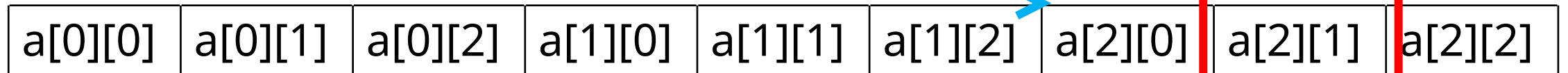
## Puntatori e Matrici



$*(*(pa + 2) + 1)$

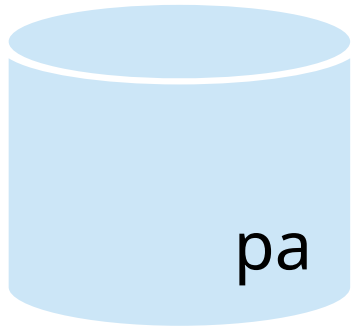
$a[i][j]$   
Equivalente  
 $*(*(pa + i) + j)$

0x7ff7bfeff200



# Puntatori

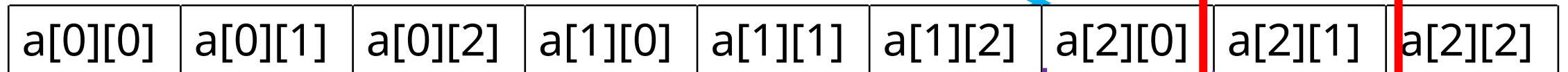
## Puntatori e Matrici



$*(*(pa + 2) + 1)$

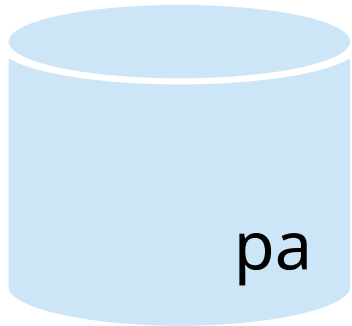
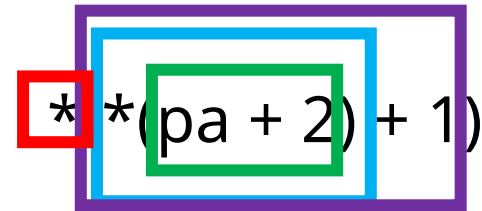
$a[i][j]$   
Equivalente  
 $*(*(pa + i) + j)$

0x7ff7bfeff200



# Puntatori

## Puntatori e Matrici

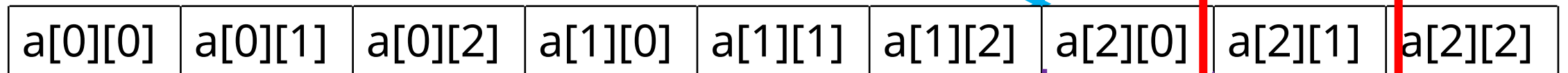


$a[i][j]$   
Equivalente  
 $*(*(pa + i) + j)$

0x7ff7bfeff200



E col puntato ne  
prendo il valore



# **Puntatori**

Esercizi – Parte 3

# Esercizio 08

## Somma di Matrici con Puntatori

### Testo

Scrivi un programma che esegue la somma di due matrici di interi di dimensione 3x3. Il programma dovrà:

1. Leggere le due matrici da input.
2. Usare puntatori per sommare gli elementi corrispondenti delle due matrici.
3. Stampare la matrice risultante dalla somma.

# **Puntatori**

Es. Pag. 58 di teoria

# **Array e Puntatori**

## **Modifica dei puntati**

Si consideri il seguente programma:

```
#include<stdio.h>
#define L 5

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(p + i) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

# **Array e Puntatori**

## **Modifica dei puntati**

Ora ne estraiamo i punti fondamentali per analizzarlo meglio

# Array e Puntatori

## Modifica dei puntati

Si consideri il seguente programma:

Copio nel puntatore  
l'indirizzo di `vet[0]`.

**Coincide con**

`p = &vet[0]`

```
[...]
int vet[L] = {-1, -1, -1, -1, -1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
    *(p + i) = 2*i;
    // [print i, vet[i], *p, p]
}

// [print valori di vet +
return]
```

# Array e Puntatori

## Modifica dei puntati

Si consideri il seguente

p

Vado a scrivere nella cella vet[i]:

```
[...]
int vet[L] = {-1, -1, -1, -1, -1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++) {
    *(p + i) = 2*i;
    // [print i, vet[i], *p, p]
}

// [print valori di vet +
return]
```

# Array e Puntatori

## Modifica dei puntati

Si consideri il seguente

p

Vado a scrivere nella cella vet[i]:

- p è come vet (vet[0], dove 0 è l'offset partendo dall'indirizzo vet),

```
[...]
int vet[L] = {-1,-1,-1,-1,-1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++) {
    *(p + i) = 2*i;
    // [print i, vet[i], *p, p]
}

// [print valori di vet +
return]
```

# Array e Puntatori

## Modifica dei puntati

Si consideri il seguente

p

Vado a scrivere nella cella `vet[i]`:

- `p` è come `vet` (`vet[0]`, dove `0` è l'offset partendo dall'indirizzo `vet`),
- `i` è l'offset partendo da `p` (`vet`)

```
[...]
int vet[L] = {-1, -1, -1, -1, -1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++) {
    *(p + i) = 2*i;
    // [print i, vet[i], *p, p]
}

// [print valori di vet +
return]
```

# Array e Puntatori

## Modifica dei puntati

Si consideri il seguente programma:

**Nota: non sto mai cambiando p!**

```
[...]  
int vet[L] = {-1, -1, -1, -1, -1};  
int *p, i;  
p = vet;  
  
// [print valori di vet]  
  
for(i = 0; i < L; i++){  
    *(p + i) = 2*i;  
    // [print i, vet[i], *p, p]  
}  
  
// [print valori di vet +  
return]
```

# Array e Puntatori

## Modifica dei puntati

Si consideri il seguente programma:

```
[...]  
int vet[L] = {-1, -1, -1, -1, -1};  
int *p, i;  
p = vet;
```

```
// [print valori di vet]
```

```
for(i = 0; i < L; i++){  
    *(p + i) = 2*i;
```

```
// [print i, vet[i], *p, p]
```

```
}
```

```
// [print valori di vet +  
return]
```

**Arr**

**Modi**

Si con  
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffefe1d0)
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffefe1d0)
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffefe1d0)
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffefe1d0)
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffefe1d0)
valori di vet: [0 2 4 6 8 ]
```

```
for(i = 0; i < L; i++){
    *(p + i) = 2*i;
    // [print i, vet[i], *p, p]
}
```

**Arr**

**Modi**

Si con  
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
```

```
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffefe1d0)
```

```
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

```
for(i = 0; i < L; i++){  
    *(p + i) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**Arr**

**Modi**

Si con  
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
```

```
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffefe1d0)
```

```
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

```
[print i, vet[i], *p,  
p] (for)
```

```
for(i = 0; i < L; i++){  
    *(p + i) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**Arr**

**Modi**

Si con  
progr

valori di vet: [-1 -1 -1 -1 -1 ]

i = 0, vet[0] = 0, \*p=0, (p=0x7fffffffefe1d0)

i = 1, vet[1] = 2, \*p=0, (p=0x7fffffffefe1d0)

i = 2, vet[2] = 4, \*p=0, (p=0x7fffffffefe1d0)

i = 3, vet[3] = 6, \*p=0, (p=0x7fffffffefe1d0)

i = 4, vet[4] = 8, \*p=0, (p=0x7fffffffefe1d0)

valori di vet: [0 2 4 6 8 ]

Come vedete, la linea evidenziata  
non modifica affatto p

[pr

[print i, vet[i], \*p,  
p] (for)

```
for(i = 0; i < L; i++){  
    *(p + i) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**Arr**

**Modi**

Si con  
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
```

```
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffefe1d0)
```

```
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

[  
p  
Accede invece a p+i, usando il  
contatore del loop (i) come  
offset dall'indirizzo p  
(p=vet=vet[0])

```
for(i = 0; i < L; i++){  
    *(p + i) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**Arr**

**Modi**

Si con  
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
```

```
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffefe1d0)
```

```
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffefe1d0)
```

```
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

```
[print i, vet[i], *p,  
p] (for)
```

```
[print valori di vet  
+ return]
```

```
for(i = 0; i < L; i++){  
    *(p + i) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

# **Array e Puntatori**

## **Modifica dei puntati**

Modifichiamo il programma ... Cambiamo dove andiamo a scrivere all'interno del loop for

```
#include<stdio.h>
#define L 5

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(p + i) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

```

#include<stdio.h>
#define L 5

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(p++) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nindirizzo di i: %p", &i);
    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}

```

Ovvero cambio da:

$$*(p + i) = 2*i;$$

a:

$$*(p++) = 2*i;$$

Ed aggiungo la print  
dell'indirizzo di i

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

1. Scrivo nella cella puntata da p il valore 2\*i
2. Incremento p

- La prima volta, scriverò in vet[0]=0, poi incremento p
- La seconda, scrivo in vet[1]=2
- [...]

E l'ultima? Dopo aver incrementato p, a cosa punterà?

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]
```

Notiamo innanzitutto come  
gli indirizzi aumentino di 4  
parole

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefc4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefc8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefcc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefd0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefd4)  
indirizzo di i: 0x7fffffffefd4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

p++ incrementa dopo!  
Prima viene eseguito  
vet[4]=8, poi a p viene  
aggiunto sizeof(int), e...

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffef1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]
```

Dopo l'assegnazione  
nell'ultima iterazione punta  
allo stesso indirizzo di i,  
per questo \*p=4 (=i)!

p++ incrementa dopo!  
Prima viene eseguito  
vet[4]=8, poi a p viene  
aggiunto sizeof(int), e...

```
vet = [-1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)
```

indirizzo di i: 0x7fffffffefe1d4

valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]=-1 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i         |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]=-1 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=0       |
|       |           |
|       | p         |

i = 0

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]=-1 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=0       |
|       |           |
|       | p         |

i = 0

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]=-1 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=0       |
|       |           |
|       | p         |

i = 0

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]=-1 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=0       |
|       |           |
|       | p         |

i = 0

valori di vet: [-1 -1 -1 -1 -1 ]

i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)

i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)

i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)

i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)

i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)

indirizzo di i: 0x7fffffffefe1d4

valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]=-1 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       |           |
|       | p         |

i = 1

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       |           |
|       | p         |

**i = 1**

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 1**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       |           |
|       | p         |

**i = 1**

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
**i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)**  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffffe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffffe1c8)  
**i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffffe1cc)**  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffffe1d0)  
i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffffe1d4)  
indirizzo di i: 0x7fffffffffe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=3       |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=3       |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=3       |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=3       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
**i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)**  
i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 4**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=4       |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 4**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]= 8 |
| fe1d4 | i=4       |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 4**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]= 8 |
| fe1d4 | i=4       |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(p++) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 4**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]= 8 |
| fe1d4 | i=4       |
|       |           |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffefe1d0)  
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0;  
    *(p++)  
    // [pri  
}
```

Come mai non va in segmentation fault?

**i = 4**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]= 8 |
| fe1d4 | i=4       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)  
**i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)**  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0;
    *(p++)
    // [pr i
}
```

Come mai non va in segmentation fault?

Sto accedendo in lettura ad una cella dimensionata correttamente per un intero!

**i = 4**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]= 0 |
| fe1c4 | vet[1]= 2 |
| fe1c8 | vet[2]= 4 |
| fe1cc | vet[3]= 6 |
| fe1d0 | vet[4]= 8 |
| fe1d4 | i=4       |
|       |           |
|       | p         |

valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = 0, \*p=-1, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 2, \*p=-1, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 4, \*p=-1, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 6, \*p=-1, (p=0x7fffffffefe1d0)  
**i = 4, vet[4] = 8, \*p=4, (p=0x7fffffffefe1d4)**  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [0 2 4 6 8 ]

Arra

Modifi

Visuali

È sempre così??

**NO!** Questa era una semplificazione, chi dispone le variabili in memoria è il compilatore.

Possiamo solo limitarci ad osservare cosa succede, ed in più aggiungere una variabile o cambiarne l'ordine può cambiare tutto

4)  
3)  
c)  
0)  
)

valori di vet: [0 2 4 6 8 ]

Arra

Modifi

Visuali

**N.B.:** Anche la rappresentazione dello stack come nelle slide è una semplificazione di ciò che accade realmente.

Abituatevi comunque a schematizzarlo, per entrare meglio nell'ottica dei puntatori

|       |    |
|-------|----|
| fe1c0 | va |
| fe1c4 | va |
| fe1c8 | va |
| fe1cc | va |
| fe1d0 | va |
| fe1d4 |    |
|       |    |
|       |    |

valori di vet: [0 2 4 6 8 ]

# Array e Puntatori

## Modifica dei puntati

Modifichiamo nuovamente il programma ... Cambiamo dove andiamo a scrivere all'interno del loop for: anziché usare `p++`, vediamo cosa succede con `++p`

```
#include<stdio.h>
```

```
#define L 5
```

```
int main(){
```

```
    int vet[L] = {-1,-1,-1,-1,-1};
```

```
    int *p, i;
```

```
    p = vet;
```

```
    printf("\nvalori di vet: [");
```

```
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
```

```
    printf("]");
```

```
    for(i = 0; i < L; i++){
```

```
        *(p++) = 2*i;
```

```
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
```

```
    }
```

```
    printf("\nindirizzo di i: %p", &i);
```

```
    printf("\nvalori di vet: [");
```

```
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
```

```
    printf("]\n");
```

```
    return 0;
```

```
}
```

Ovvero cambio da:

```
*(p++) = 2*i;
```

```
#include<stdio.h>
```

```
#define L 5
```

```
int main(){
```

```
    int vet[L] = {-1,-1,-1,-1,-1};
```

```
    int *p, i;
```

```
    p = vet;
```

```
    printf("\nvalori di vet: [");
```

```
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
```

```
    printf("]");
```

```
    for(i = 0; i < L; i++){
```

```
        *(++p) = 2*i;
```

```
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
```

```
    }
```

```
    printf("\nindirizzo di i: %p", &i);
```

```
    printf("\nvalori di vet: [");
```

```
    for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

```
    printf("]\n");
```

```
    return 0;
```

```
}
```

Ovvero cambio da:

```
        *(p++) = 2*i;
```

a:

```
        *(p++) = 2*i;
```

```
#include<stdio.h>
#define L 5

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(++p) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nindirizzo di i: %p", &i);
    printf("\nvalori di vet: [");
    for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

Stampiamo anche "un po' più in là" per capire meglio cosa succede

# Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

1. Prima incremento p
  2. Scrivo nella cella puntata da p il valore 2\*i
- La prima volta, incremento scriverò in vet[1]=0

# Array e Puntatori

## Modifica dei puntati

Concentri

### Modifica dei puntati

Si consideri il seguente programma:

1. Prima

2. Scrivo

- La pri  
vet[1]

Copio nel puntatore  
l'indirizzo di vet[0].

**Coincide** con

p = &vet[0]

```
for(i = 0; i < L; i++){
```

```
*(p + i) = 2*i;
```

```
int vet[L] = {-1,-1,-1,-1,-1}
```

```
int *p, i;
```

```
p = vet;
```

```
// [print valori di vet]
```

```
for(i = 0; i < L; i++){
```

```
*(p + i) = 2*i;
```

```
// [print i, vet[i], *p, p]
```

```
}
```

[Dallo screenshot della slide in cui  
spiegavamo il programma]  
p era inizializzato con vet,  $\equiv p = \&vet[0]$

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

1. Prima incremento p
  2. Scrivo nella cella puntata da p il valore 2\*i
- La prima volta, incremento scriverò in vet[1]=0
  - La seconda, incremento e scrivo in vet[2]=2
  - [...]
  - E l'ultima? Dopo aver incrementato p, a cosa punterà?

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
}
```

Cerchiamo di analizzare lo stack come prima, per capire meglio (sempre con la nostra semplificazione, per rimanere coerenti con le slides prima)

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -865439461 -7552 32767 ]
```

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 0**

|       |           |  |
|-------|-----------|--|
| fe1c0 | vet[0]=-1 |  |
| fe1c4 | vet[1]=-1 |  |
| fe1c8 | vet[2]=-1 |  |
| fe1cc | vet[3]=-1 |  |
| fe1d0 | vet[4]=-1 |  |
| fe1d4 | i=0       |  |
|       | ...       |  |
|       | p         |  |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 0**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]=-1 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=0       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 0**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=0       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 0**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=0       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 1**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 1**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]=-1 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 1**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 1**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=1       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffefe1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffefe1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffefe1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffefe1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffefe1d4)  
indirizzo di i: 0x7fffffffefe1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(&p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]=-1 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(&p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]= 4 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 2**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]= 4 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=2       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]= 4 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=3       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(&p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]= 4 |
| fe1d0 | vet[4]=-1 |
| fe1d4 | i=3       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(&vet[i]) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |             |
|-------|-------------|
| fe1c0 | vet[0] = -1 |
| fe1c4 | vet[1] = 0  |
| fe1c8 | vet[2] = 2  |
| fe1cc | vet[3] = 4  |
| fe1d0 | vet[4] = 6  |
| fe1d4 | i = 3       |
|       | ...         |
|       | p           |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 3**

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]= 4 |
| fe1d0 | vet[4]= 6 |
| fe1d4 | i=3       |
|       | ...       |
|       | p         |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 4**

|       |             |
|-------|-------------|
| fe1c0 | vet[0] = -1 |
| fe1c4 | vet[1] = 0  |
| fe1c8 | vet[2] = 2  |
| fe1cc | vet[3] = 4  |
| fe1d0 | vet[4] = 6  |
| fe1d4 | i = 4       |
|       | ...         |
|       | p           |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(&vet[i]) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 4**

|       |             |
|-------|-------------|
| fe1c0 | vet[0] = -1 |
| fe1c4 | vet[1] = 0  |
| fe1c8 | vet[2] = 2  |
| fe1cc | vet[3] = 4  |
| fe1d0 | vet[4] = 6  |
| fe1d4 | i = 4       |
|       | ...         |
|       | p           |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(&p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 8**

|       |              |
|-------|--------------|
| fe1c0 | vet[0] = -1  |
| fe1c4 | vet[1] = 0   |
| fe1c8 | vet[2] = 2   |
| fe1cc | vet[3] = 4   |
| fe1d0 | vet[4] = 6   |
| fe1d4 | <b>i = 8</b> |
|       | ...          |
|       | p            |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

**i = 8**

|       |              |
|-------|--------------|
| fe1c0 | vet[0] = -1  |
| fe1c4 | vet[1] = 0   |
| fe1c8 | vet[2] = 2   |
| fe1cc | vet[3] = 4   |
| fe1d0 | vet[4] = 6   |
| fe1d4 | <b>i = 8</b> |
|       | ...          |
|       | p            |

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

i = 8

vet[8]

|       |            |
|-------|------------|
| fe1c0 | vet[0]= -1 |
| fe1c4 | vet[1]= 0  |
| fe1c8 | vet[2]= 2  |
| fe1cc | vet[3]= 4  |
| fe1d0 | vet[4]= 6  |
| fe1d4 | i=8        |
|       | ...        |
|       | p          |



valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, \*p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, \*p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, \*p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, \*p=6, (p=0x7fffffffef1d0)  
**i = 8, vet[8] = -7552, \*p=8, (p=0x7fffffffef1d4)**  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

|       |            |
|-------|------------|
| fe1c0 | vet[0]= -1 |
| fe1c4 | vet[1]= 0  |
| fe1c8 | vet[2]= 2  |
| fe1cc | vet[3]= 4  |
| fe1d0 | vet[4]= 6  |
| fe1d4 | i=8        |
|       | ...        |
|       | p          |

i = 8

p "punta"  
i

ri di vet: [-1 -1 -1 -1 -1 ]

i = 0, vet[0] = -1, \*p=0, (p=0x7fffffffef1c4)

i = 1, vet[1] = 0, \*p=2, (p=0x7fffffffef1c8)

i = 2, vet[2] = 2, \*p=4, (p=0x7fffffffef1cc)

i = 3, vet[3] = 4, \*p=6, (p=0x7fffffffef1d0)

i = 8, vet[8] = -7552, \*p=8, (p=0x7fffffffef1d4)

indirizzo di i: 0x7fffffffef1d4

valori di vet: [-1 0 2 4 6 5 -1620749056 -865439461 -7552 32767 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++) {  
    *(++p) = 2*i;  
    // [print i, vet[i], *p, p]  
}
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]= 4 |
| fe1d0 | vet[4]= 6 |
| fe1d4 | i=8       |
|       | ...       |
|       | p         |

**i = 8**

**Perché allora qui  
otteniamo 5 se  
abbiamo detto che  
sovrascriviamo i??**

valori di vet: [-1 -1 -1 -1 -1]  
i = 0, vet[0] = -1, \*p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, \*p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, \*p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, \*p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, \*p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 **5** -1620749056 -  
865439461 -7552 32767 ]

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack

```
for(i = 0;  
    *(++p)  
    // for i
```

Hint su un pezzo di codice  
che viene dopo il loop  
appena analizzato ...

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

|       |            |
|-------|------------|
| fe1c0 | vet[0]= -1 |
| fe1c4 | vet[1]= 0  |
| fe1c8 | vet[2]= 2  |
| fe1cc | vet[3]= 4  |
| fe1d0 | vet[4]= 6  |
| fe1d4 | i=8        |
|       | ...        |
|       | p          |

i = 8

**Perché allora qui  
otteniamo 5 se  
abbiamo detto che  
sovrascriviamo i??**

```
valori di vet: [-1 -1 -1 -1 -1 ]  
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffef1c4)  
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffef1c8)  
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffef1cc)  
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffef1d0)  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# Array e Puntatori

## Modifica dei puntati

Visualizziamo lo stack

```
for(i = 0;  
    *(++p)  
    // for(i
```

Hint su un pezzo di codice  
che viene dopo il loop  
appena analizzato ...

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

|       |           |
|-------|-----------|
| fe1c0 | vet[0]=-1 |
| fe1c4 | vet[1]= 0 |
| fe1c8 | vet[2]= 2 |
| fe1cc | vet[3]= 4 |
| fe1d0 | vet[4]= 6 |
| fe1d4 | i=8       |
|       | ...       |
|       | p         |

**i = 8**

**È comunque la cella riservata  
ad i ... Anche nel loop dopo!  
Quando i vale 5, vet[5]  
stampa i (quindi 5). Dopo, i  
continuerà a crescere per  
stampare gli altri valori**

```
valori di v  
i = 0, vet[  
i = 1, vet[  
i = 2, vet[  
i = 3, vet[  
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffef1d4)  
indirizzo di i: 0x7fffffffef1d4  
valori di vet: [-1 0 2 4 6 5 -1620749056 -  
865439461 -7552 32767 ]
```

# **Array e Puntatori**

## **Modifica dei puntati**

Come mai non otteniamo segmentation fault?

# Array e Puntatori

## Modifica dei puntati

Come mai non otteniamo segmentation fault?

Per come è stata disposta la memoria (ricorda, aggiungere un'altra variabile o cambiare compilatore potrebbe portare ad un risultato diverso), lì posso accedere, ma **non è garantito!**

# Array e Puntatori

## Modifica dei puntati

Trasformando il ciclo di stampa da:

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

# Array e Puntatori

## Modifica dei puntati

Trasformando il ciclo di stampa da:

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

a:

```
for(i = 0; i < L+10000; i++) printf("%d ", vet[i]);
```

# Array e Puntatori

## Modifica dei puntati

Trasformando il ciclo di stampa da:

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

a:

```
for(i = 0; i < L+10000; i++) printf("%d ", vet[i]);
```

# Array e Puntatori

## Modifica dei puntati

Ho segmentation fault, vado ad accedere a dove non dovrei!

```
25 printf("\nvalori di vet: ");  
26 for(i = 0; i < L+10000; i++) printf("%d ", vet[i]);
```

Exception has occurred. ×  
Segmentation fault

```
27 printf("]\n");  
28 return 0;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
67853929 1633971823 1868771123 1650549870 792358505 796029813 1633  
906540 1651715948 792358505 796029813 1633906540 1768042348 196602  
9422 1647276659 792358505 796029813 795765090 1702127987 191925052  
7 1966029420 1647276659 1982819945 1868852837 1701863282 792358002  
796029813 795765090 1701998435 1919250527 1747925612 795176303 16  
84955500 2020175215 1868770863 1734960750 1685013295 1934962533 17  
31162725 1633841004 1869894508 1701273970 1953064751 778204520 176  
8976227 762605420 1952540771 1650811951 1866688373 1851878765 1380  
909156 1313425225 1482640449 1130317636 1163022933 1147098190 1414  
222661 1211977807 1819439225 6581857 1398096452 1397052255 1313818
```

# Thank You!

Mail: [luca1.alessandrini@polimi.it](mailto:luca1.alessandrini@polimi.it)

Website: <https://alessandriniluca.github.io/>