Allocazione

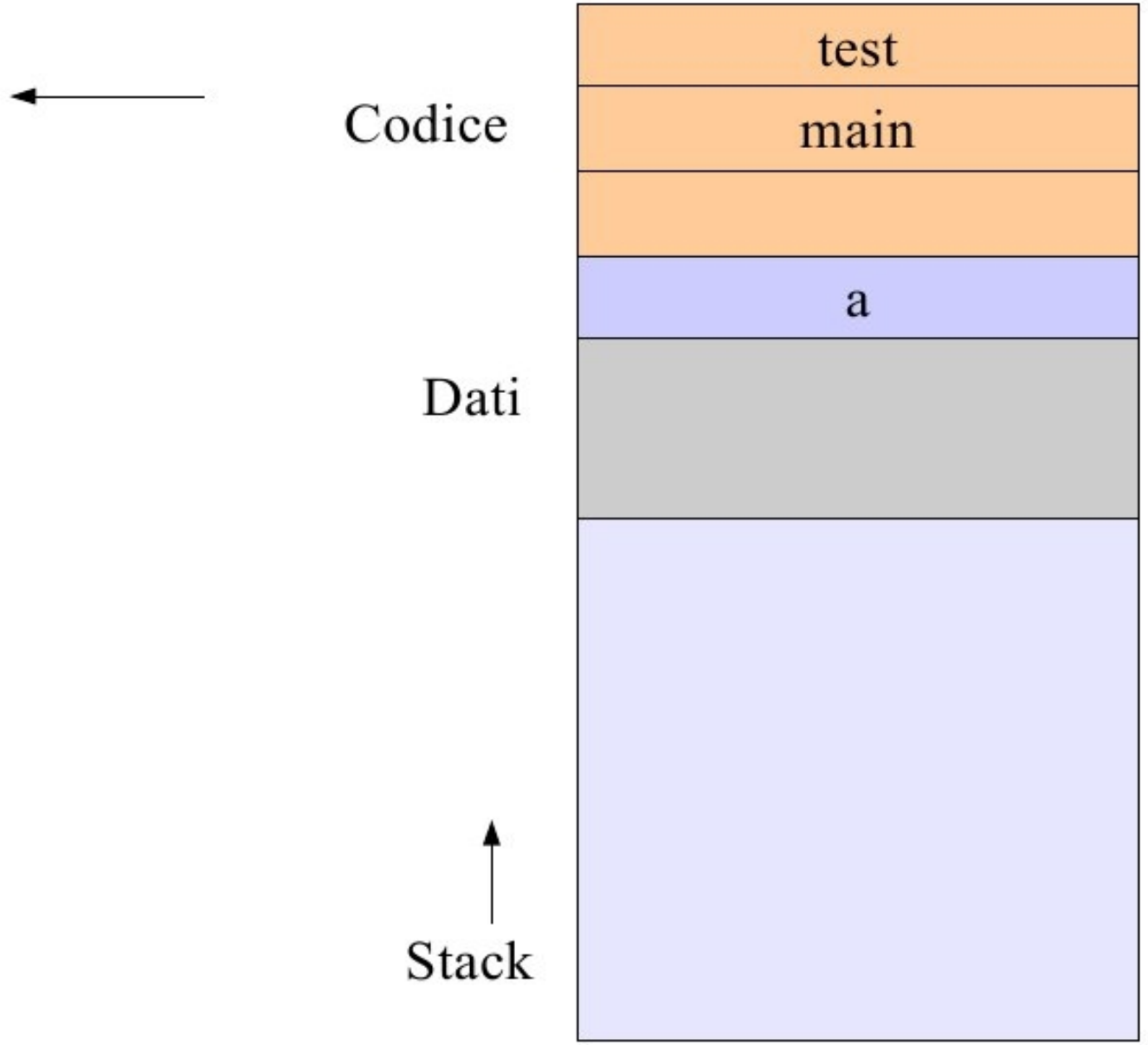
L'allocazione statica: e' quella con cui sono memorizzate le variabili globali e le variabili statiche, queste vengono allocate all'avvio nel segmento dati da exec, con le distizioni che abbiamo visto.

L'allocazione automatica: argomenti di una funzione e variabili locali esistono solo per la durata della funzione. Lo spazio per queste variabili viene allocato nello stack quando viene eseguita la funzione e liberato quando si esce dalla medesima.

L'allocazione dinamica: non e' di per se prevista direttamente dal linguaggio C, ma la libc mette a disposizione alcune funzioni utilizzabili allo scopo. Di certo tutti conoscono malloc, calloc, realloc e free. Funzioni usate per allocare dinamicamente memoria (heap).

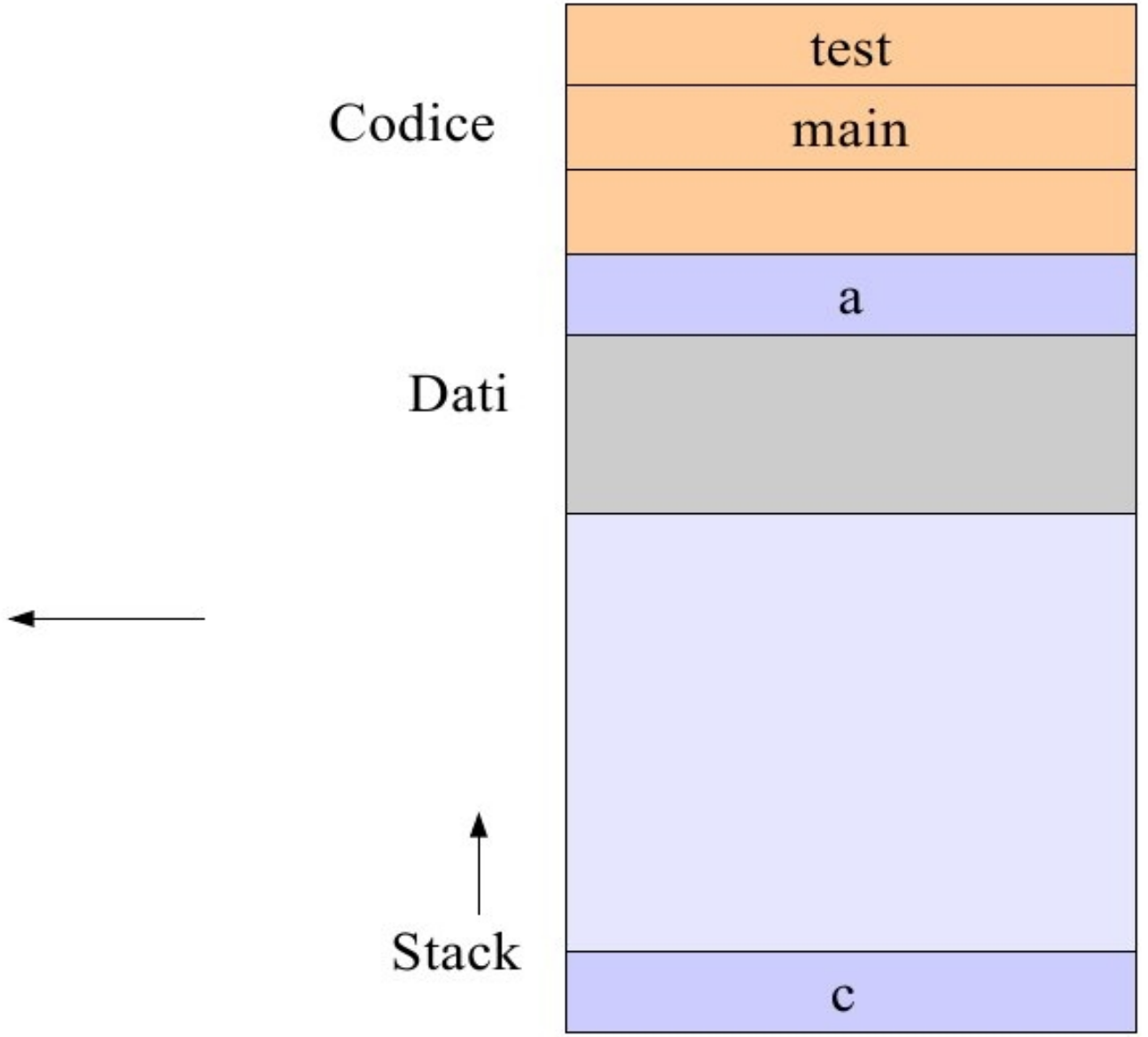
Variabili

```
int a=3;
void test( )
{
  int b=2;
}
int main( )
{
  int c=1;
  test();
}
```



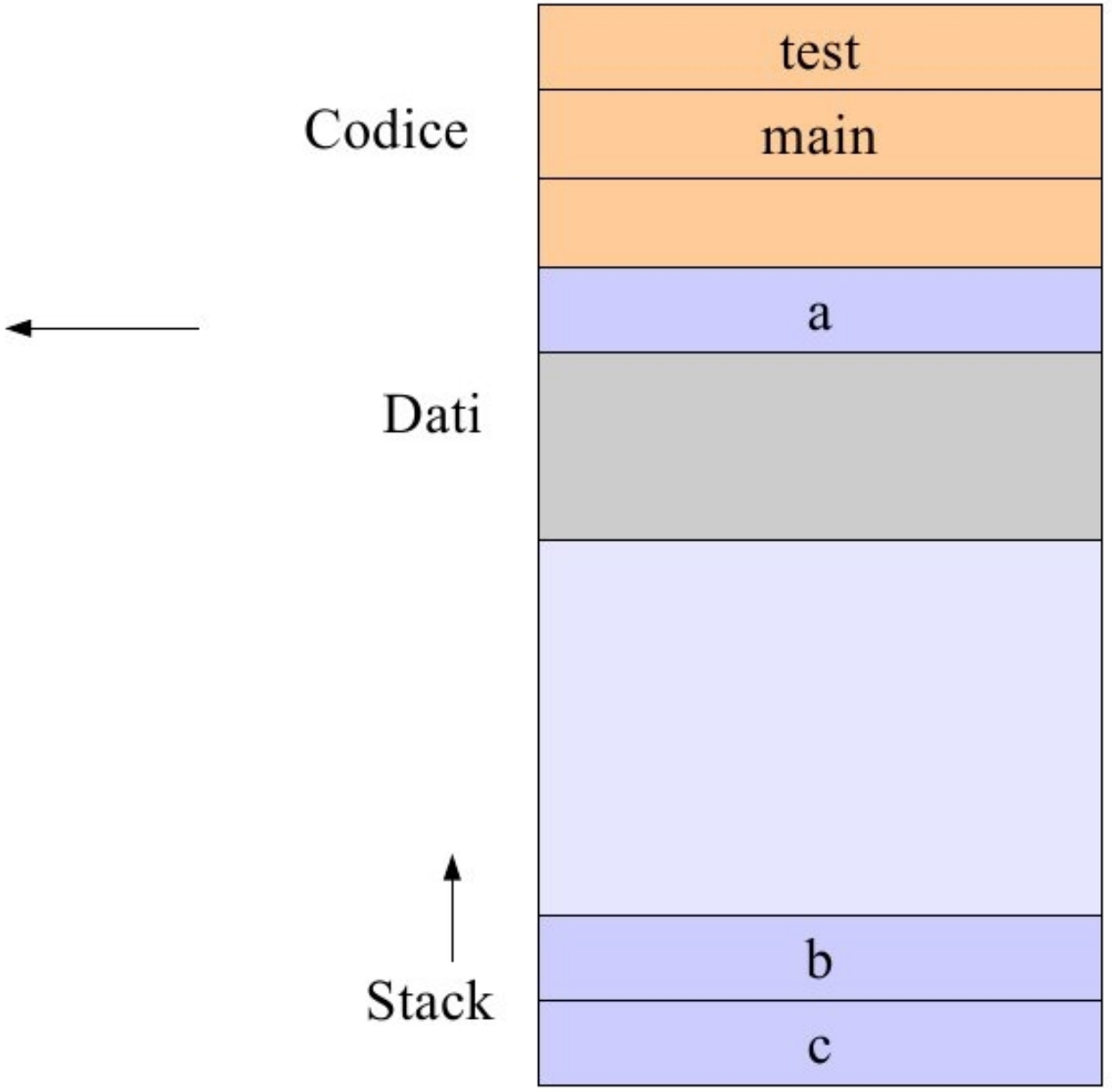
Variabili

```
int a=3;
void test( )
{
  int b=2;
}
int main( )
{
  int c=1;
  test();
}
```



Variabili

```
int a=3;
void test( )
{
  int b=2;
}
int main( )
{
  int c=1;
  test();
}
```



extern

E' possibile dichiarare una variabile globale in qualsiasi parte del programma, anche in un file separato. Si distingue in questo caso tra:

- dichiarazione reale della varibile globale

- dichiarazione con extern, rende noto al compilatore che quella variabile e' definita altrove (la sintassi e' semplice: **extern int a;**)

static

- Le variabili locali **static** a differenza di quelle locali automatiche “vivono” per tutta la durata del processo. Tuttavia e' possibile fare riferimento ad esse solo quando l'esecuzione e' nel blocco dove sono definite

- Le variabili globali static sono visibili solo nel file dove sono state dichiarate

register

La parola-chiave **register** suggerisce al compilatore che meglio sarebbe mantenere quella variabile in un registo del processore.

Chiaramente il numero di variabili che possono essere dichiare come register e' limitato. E comunque posso dichiarare come register solo variabili automatiche o parametri formali di una funzione.

```
register int i;
```

Puntatori

Scrivere un programma stampi le dimensioni in byte di puntatori: double*, int*, void*, char*. (che risultato vi aspettate ?)

```
$ gcc -o puntatori puntatori.c
$ ./puntatori
(void *)          8 (char *)          8 (int *)          8
(double *)          8
```

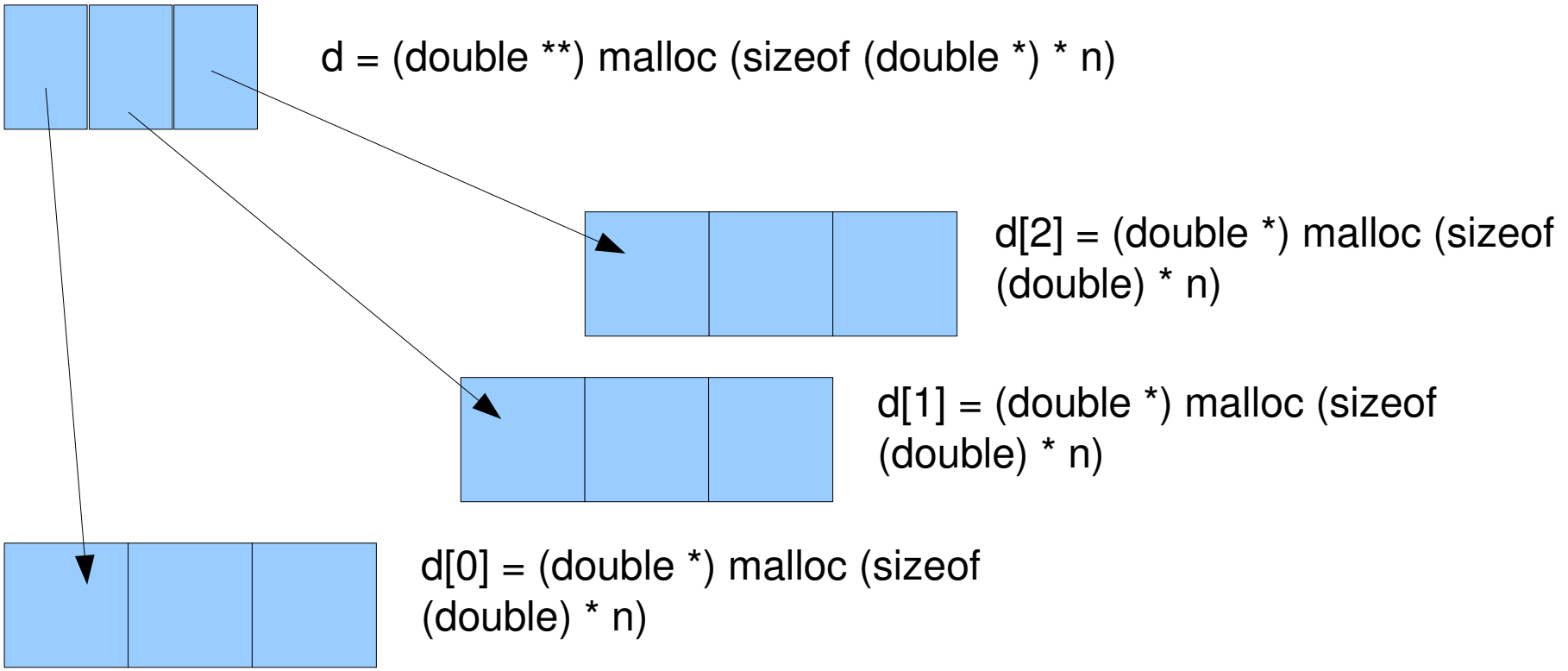
Il risultato e' ovvio visto che la dimensione in byte di un puntatore dipende dall'architettura dell'elaboratore che si sta usando.

Esercizio

Scrivere un programma che allochi spazio necessario per contenere una matrice N x N di double. Tale programma deve inizializzare anche la matrice con numeri random. (N opzione da linea di comando) (soluzione2.c)

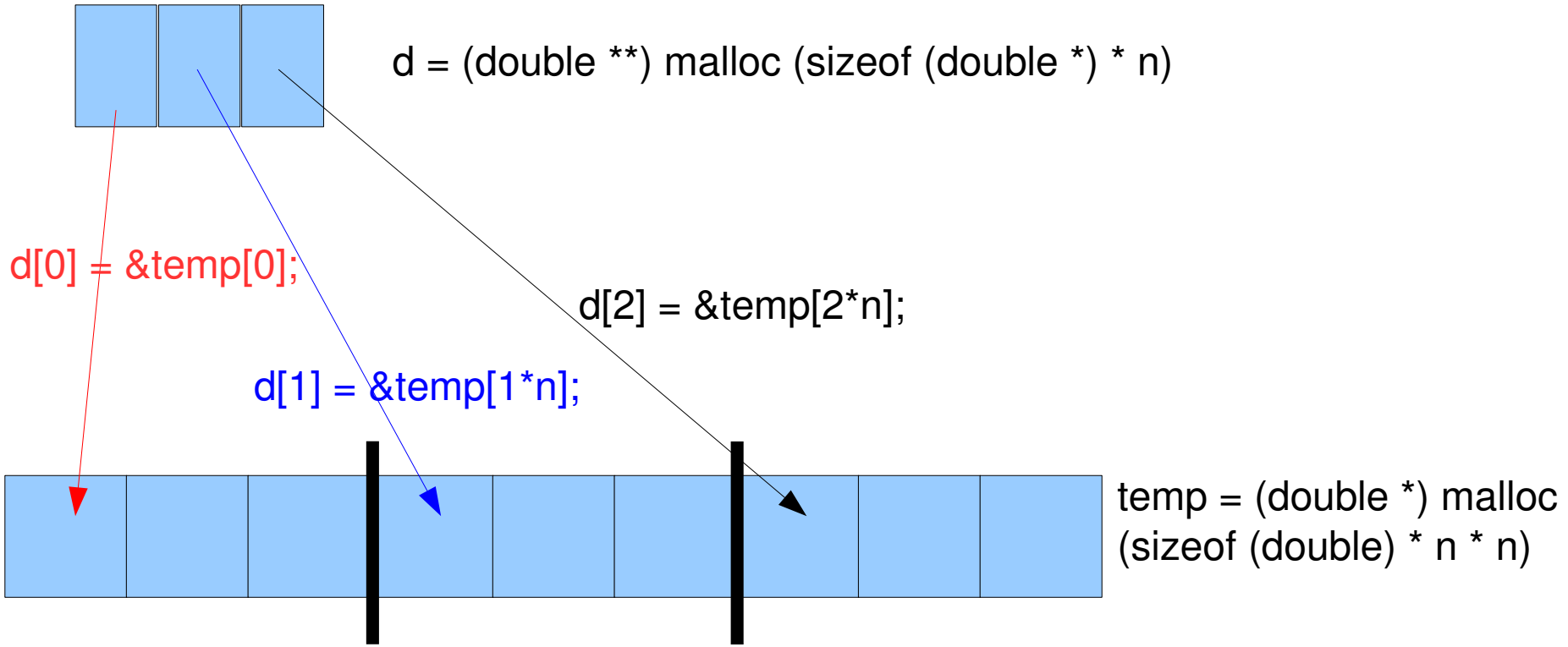
Allocazione dinamica della memoria

Vedi soluzione2.c (primo)



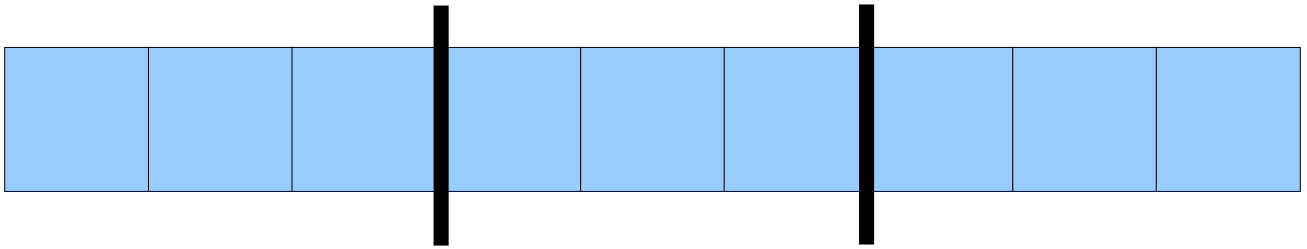
Allocazione dinamica della memoria

Vedi soluzione2.c (secondo)



Allocazione dinamica della memoria

Vedi soluzione2.c (terzo)



d = (double *) malloc (sizeof (double) * n * n)

puntatori

Un po' di algebra dei puntatori. Visto quello che e' detto sopra perche' esistono tipi diversi di puntatori ? (vediamo main.c)

```
double * d, * t;
...
for (t=d, i=0; i<n; i++, t++) {
  fprintf (stdout, "%u - %u [%u]\n", &d[i], t, d+i);
}
```

stampa:

```
26488848 - 26488848 [26488848]
26488856 - 26488856 [26488856]
26488864 - 26488864 [26488864]
26488872 - 26488872 [26488872]
26488880 - 26488880 [26488880]
```

Memory leak

`valgrind ed i memory leak.`

```
$ valgrind --leak-check=full main
...cut
==9341== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 1)
==9341== malloc/free: in use at exit: 60 bytes in 2 blocks.
==9341== malloc/free: 2 allocs, 0 frees, 60 bytes allocated.
==9341== For counts of detected errors, rerun with: -v
==9341== searching for pointers to 2 not-freed blocks.
==9341== checked 64,872 bytes.
==9341==
==9341== 20 bytes in 1 blocks are definitely lost in loss record 1 of 2
==9341==    at 0x4A0739E: malloc (vg_replace_malloc.c:207)
==9341==    by 0x400705: main (in /home/redo/Lezioni/Inform/lab_gen_inf/2008/slides/4/main)
==9341==
==9341==
==9341== 40 bytes in 1 blocks are definitely lost in loss record 2 of 2
==9341==    at 0x4A0739E: malloc (vg_replace_malloc.c:207)
==9341==    by 0x400603: main (in /home/redo/Lezioni/Inform/lab_gen_inf/2008/slides/4/main)
==9341==
==9341== LEAK SUMMARY:
==9341==    definitely lost: 60 bytes in 2 blocks.
==9341==      possibly lost: 0 bytes in 0 blocks.
==9341==    still reachable: 0 bytes in 0 blocks.
==9341==         suppressed: 0 bytes in 0 blocks.
```

Alloca - (Esercizio)

Allocare "dinamicamente" nello stack, la funzione "alloca" (non e' POSIX vedi "man alloca"). Lo spazio allocato e' liberato ovviamente all'uscita dalla routine.

Scrivere un programma che allochi spazio necessario per contenere una vettore di double lungo N nello stack. Tale programma deve inizializzare anche il vettore con numeri random. (N opzione da linea di comando) (soluzione3.c)

Alloca

```
$ ./soluzione3 10000000
Segmentation fault

A cosa e' dovuto l'errore ?

$ ulimit -s
10240
$ bc
bc 1.06
Copyright  1991-1994,  1997,  1998,  2000  Free  Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
10000000*8
80000000
80000000/1024
78125
```

Alloca

```
$ ulimit -s 78125
$ ./soluzione3 10000000
```

Aumentanto il limite dello “stack size” l'errore chiaramente “scompare”.

Esercizio

Concludiamo con una moltiplicazione matrice matrice. Un programma che allocate due matrici quadrate di dimensione N, le riempia con numeri random e calcoli il prodotto tra le due. (soluzione4.c)