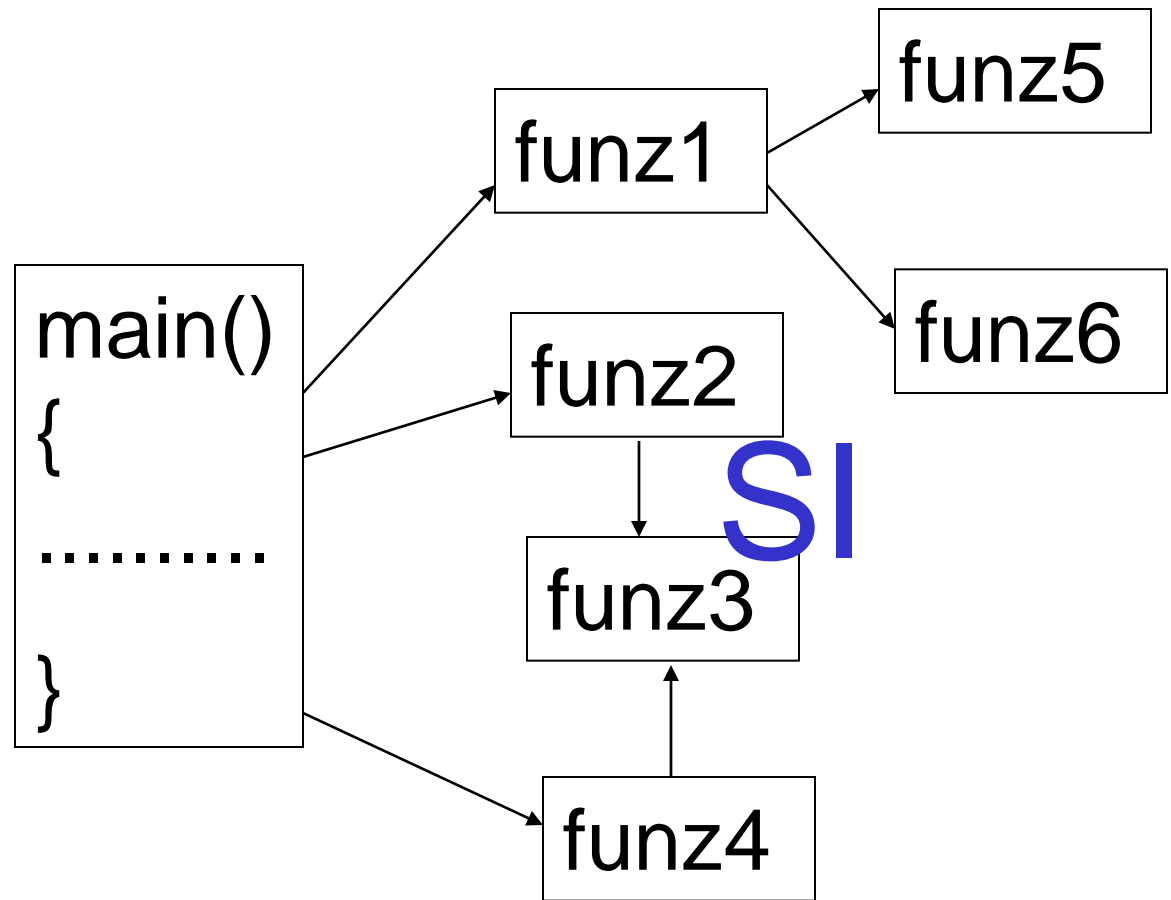


FUNZIONI

cap. 7 del testo

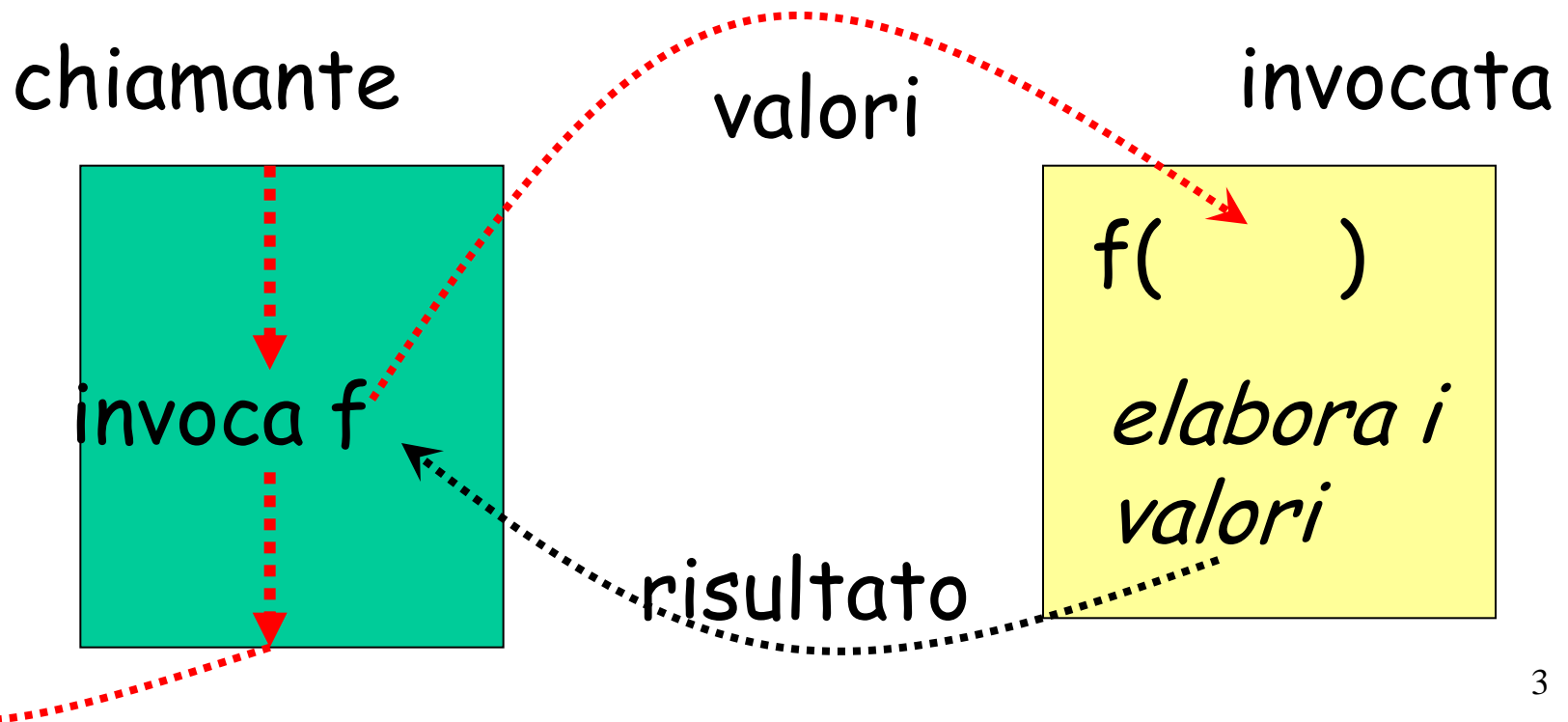
necessità di strutturare i programmi

```
main()
{.....
.....
.....N.....
.....O.....
.....
.....
.....
.....
.....
.....
.....}
```



Funzioni

Una **funzione** è un pezzo di programma con un nome. Essa viene eseguita tramite **l'invocazione** del suo nome.



il più grande divisore di un valore dato:

```
int divisore(int x)
{
    int y=x/2;
    while (x % y != 0)
        y--;
    return y;
}
```

variabile locale

```
int divisore(int x)
{
    int y=x/2;
    while (x % y != 0)
        y--;
    return y;
}
```

entrata
valore su
cui
lavorare

x
parametro
formale

stesso
tipo

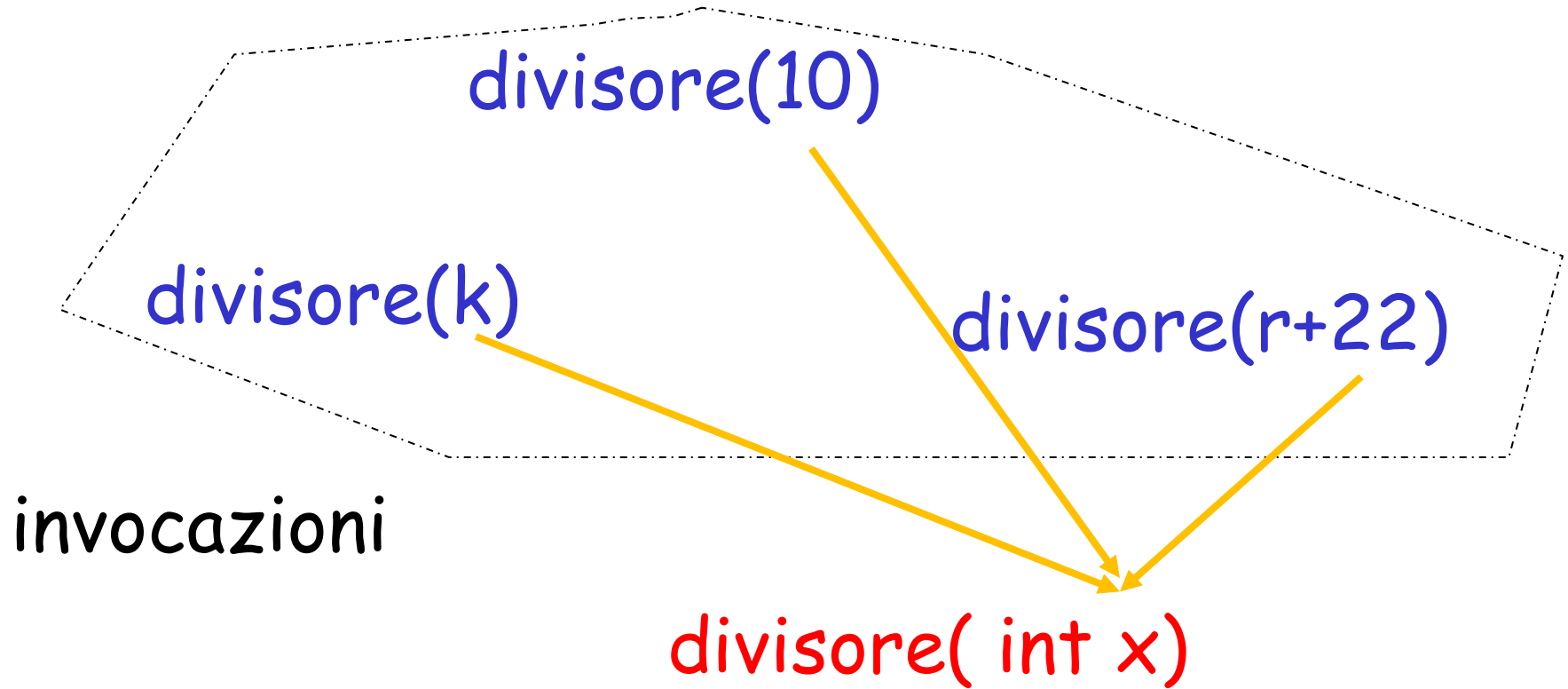
valore restituito

trova il massimo numero primo più piccolo o uguale a z dato

```
int primo(int z)
{
    int k=z;
    while(k>1 && ! (divisore(k) == 1))
        k--;
    return k;
}
```

invoca la funzione divisore

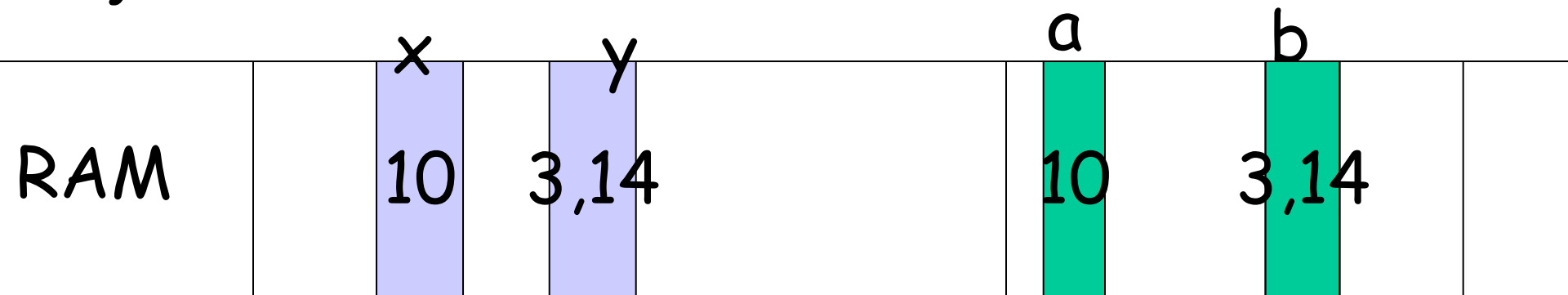
passaggio dei parametri: attuali → formali



il parametro attuale è un valore che diventa l'R-valore di x

passaggio dei parametri per valore

```
int F(int a, double b) {.....return v;}  
main()  
{  
  int x=10; double y=3,14;  
  int z=F(x,y); //invocazione  
}
```



x e a hanno L-valori diversi e lo stesso vale per y e b

passaggio dei parametri

PER VALORE

i parametri attuali sono espressioni

$f(e_1, e_2, \dots, e_m)$

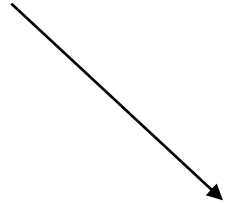
il valore di e_i
diventa l'R-
valore di x_i

e i tipi?

T

$f(T_1 x_1, T_2 x_2, \dots, T_m x_m)$

$f(\dots ei \dots)$



$T f(\dots Ti xi \dots)$

- se il tipo del valore di ei è Ti , facile
- se è diverso ?

conversione automatica (vedi testo
nelle sezioni 2.3 e 9.4)

quando una funzione non restituisce alcun risultato, dobbiamo dichiarare che il suo tipo di ritorno è

void

non esistono valori di tipo void

```
void f(.....);
```

le funzioni viste finora hanno un limite

non è possibile realizzare una funzione
`void f(int x, double y)` tale che :

`int x = 10; double y=3,14;`

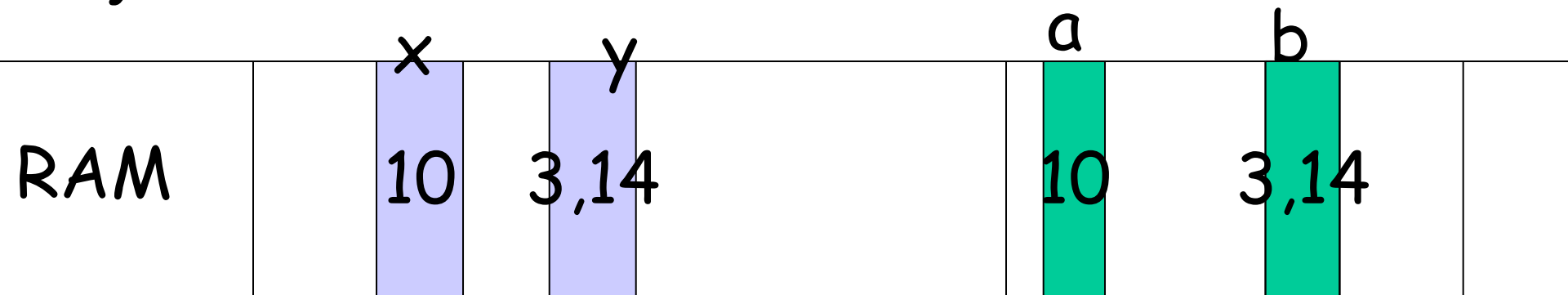
`f(x,y);`

e dopo l'invocazione `x` e/o `y` è cambiato

si chiama *side-effect*

passaggio dei parametri per valore

```
int F(int a, double b) {a=a*b; return v;}  
main()  
{  
  int x=10; double y=3,14;  
  int z=F(x,y); //invocazione  
}
```



F cambia a, ma questo non ha effetto su x

potremmo anche fare;

```
int F(int a, double b) {a=a*b; return a;}  
main()  
{  
    int x=10; double y=3,14;  
    x=F(x,y);  
}
```

funziona solo con 1 variabile

per avere side-effect: passiamo (per valore) **puntatori**

anziché passare alla funzione F **per valore** l'R-valore di x e y, passiamo **per valore** il **puntatore a x e quello a y**

quindi avremo: `void F(int * a, double* b);`

se un parametro formale è `int * x` allora il corrispondente parametro attuale deve fornire l'indirizzo di un intero

cioè `un'espressione` che ha un `valore di tipo`
`int *`

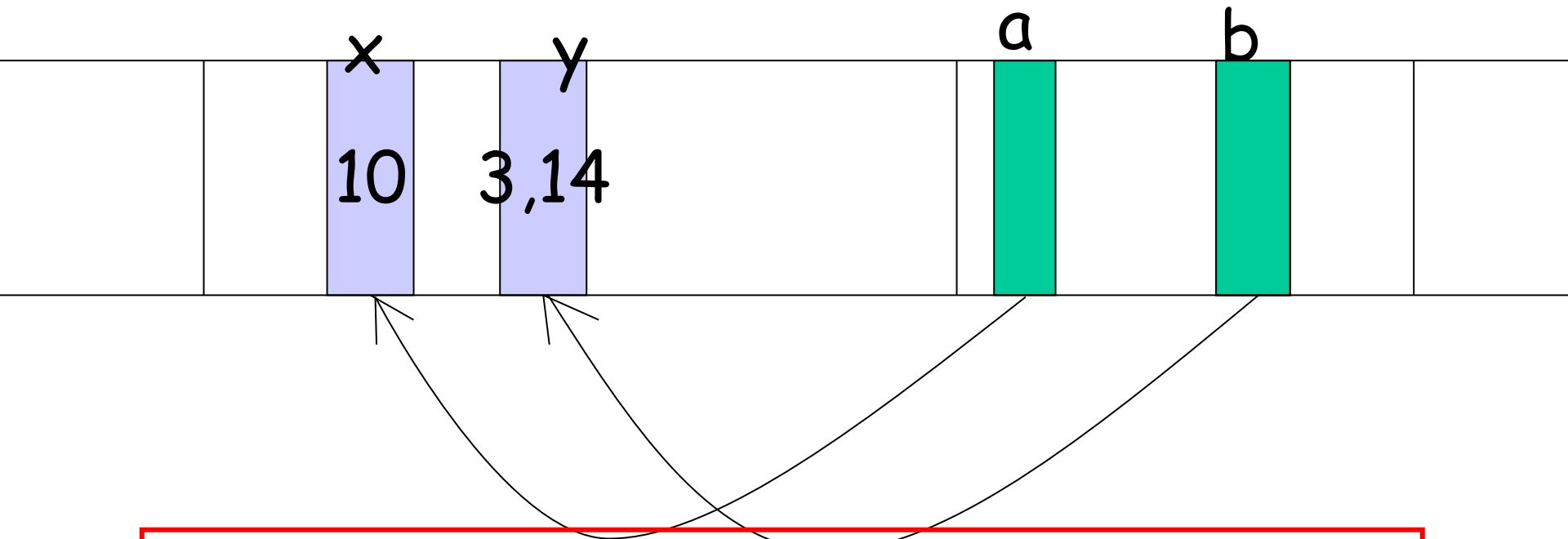
e lo stesso per `double*`


```
void F(int * a, double* b)
{
  *a=(*a)*(*b);
}
```

come invocarla
per cambiare la variabile x ?

F(&x, &y);

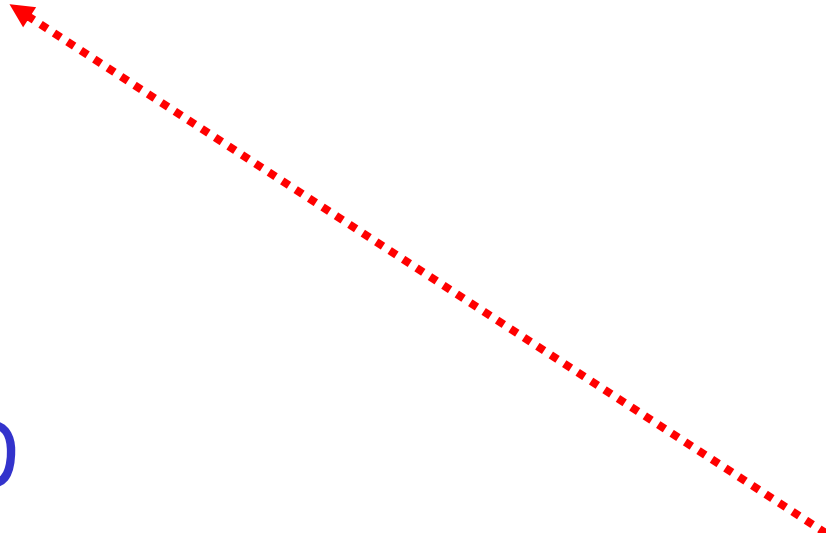
passiamo l'L-valore di x e y



```
void F(int * a, double* b)
{
  *a=(*a)*(*b); *b=(*b)+4,2;
}
```

per avere side-effect: **passaggio dei
parametri per riferimento**

```
void f(int & x) {x=x*2;}  
main()  
{  
  int A=10;  
  f(A);  
} // qui A=20
```



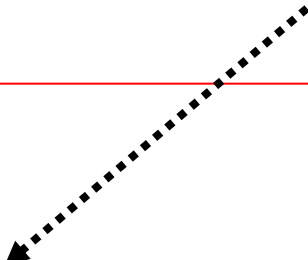
x è passato per
riferimento => x è
un alias di A

```
void g(int x, int & y)
{ x=y+1; }
main()
{
int A=10;
g(A,A);
} // valore di A ?
```

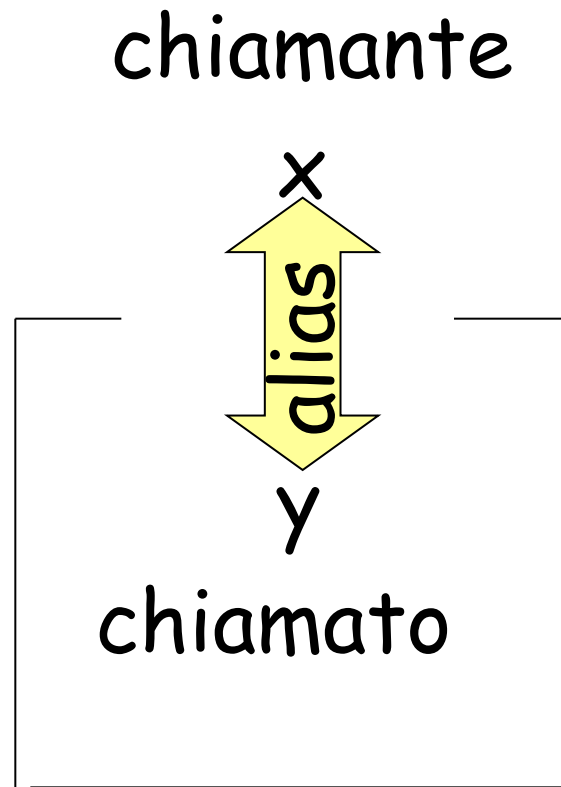
e con & ?

e se ?

```
void g(int x, int & y)
{ x++; y++; }
```



i parametri passati per riferimento
mettono in comune una variabile tra
chiamante e chiamato



che può
servire in
entrambe
le
direzioni
o solo in
una

è diverso per i puntatori passati per valore ?

e ha senso passare un puntatore per riferimento?

e...

esercizio: scambiare gli R-valori di 2 variabili:

```
char x='a',y='b';
```

```
F(x,y);
```

```
// qui x=='b' e y=='a'
```

come deve essere ? f(?, ?)

esercizi:

scambiare 2 array

scambiare 2 parti dello stesso array