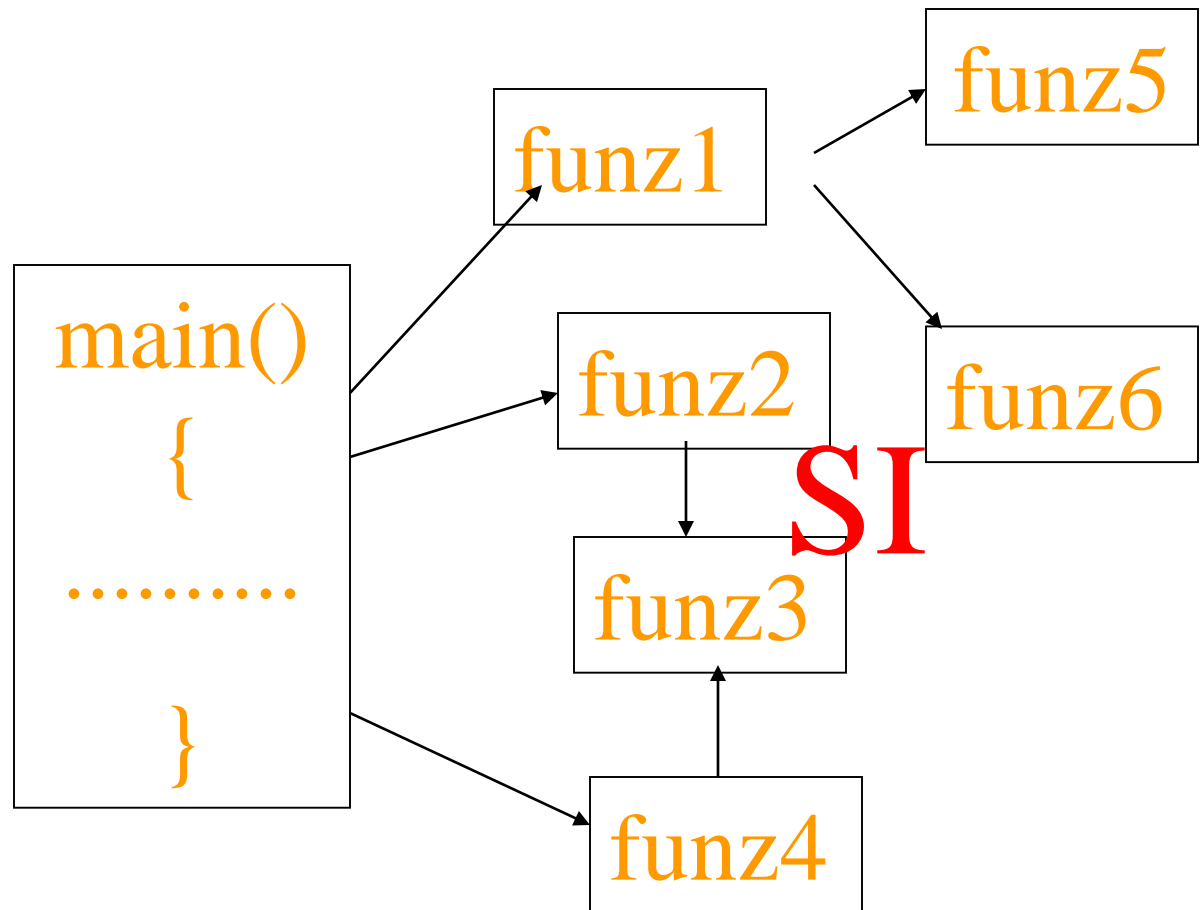


# FUNZIONI

cap. 7 del testo

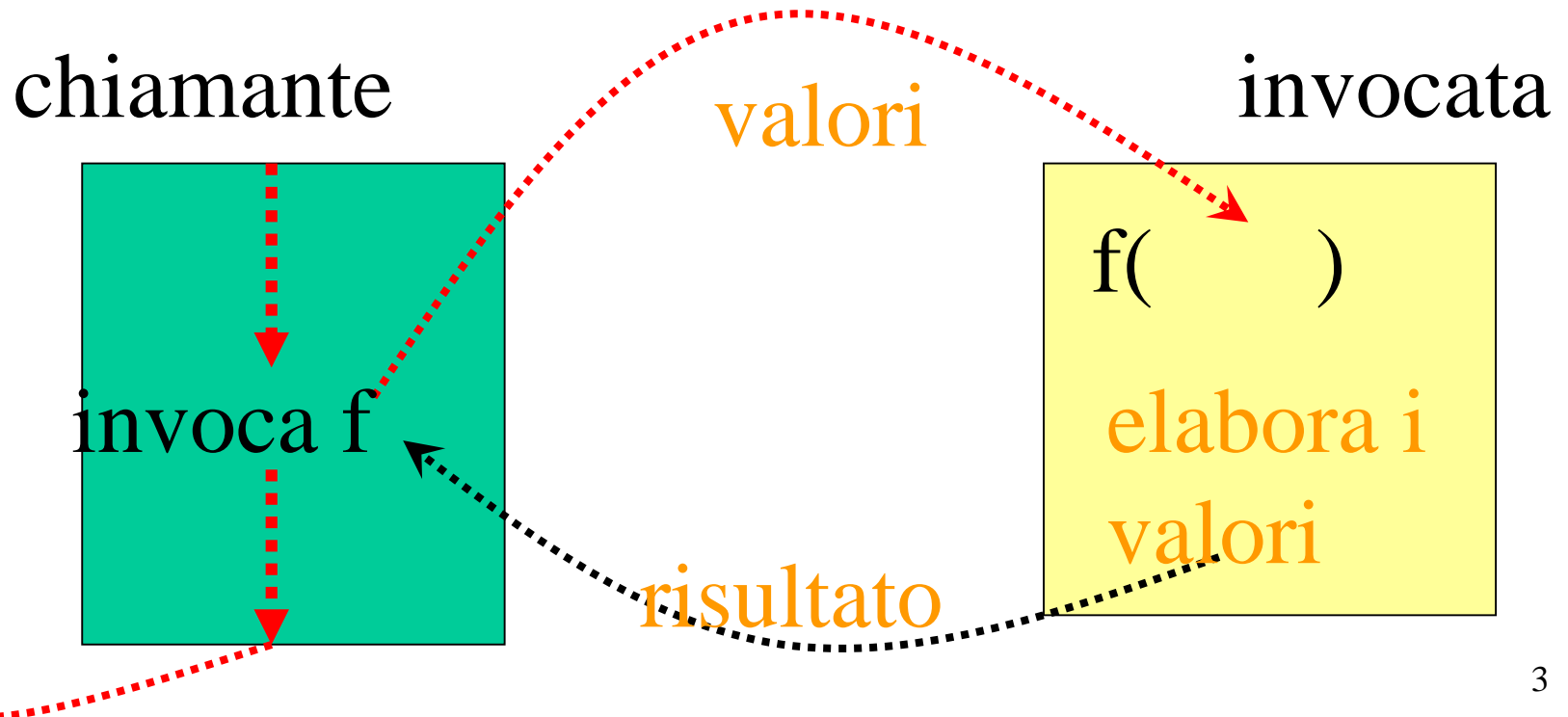
# necessità di strutturare i programmi

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



# Funzioni

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



funzione che calcola il più grande divisore  
di un naturale  $x$  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;
}
```

entra il valore  
su cui  
lavorare

x è 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

parametri

definizione:

$T \quad F( T1 \ x1, T2 \ x2, \dots Tn \ xn) \quad \text{formali}$

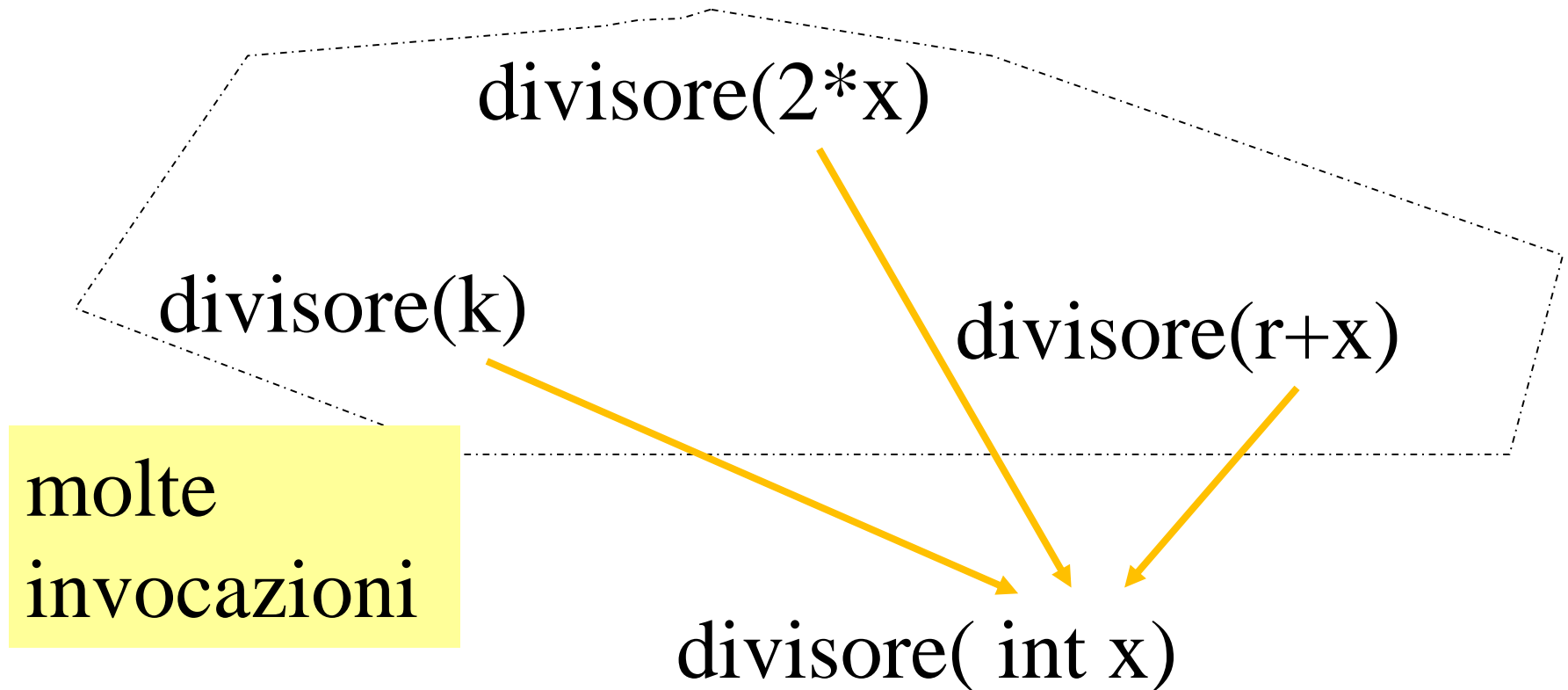
invocazione:

$\dots\dots\dots F(b1, b2, \dots, bn) \quad \text{attuali}$

b1 dovrebbe avere tipo T1, b2 tipo T2, ..., bn  
tipo Tn

altrimenti ..... conversioni

passaggio dei parametri: attuali  $\rightarrow$  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
```

```
}
```



# passaggio dei parametri PER VALORE

i parametri attuali sono  
espressioni

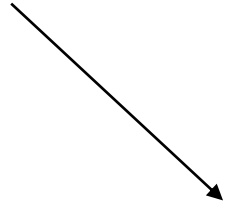
il valore di  $b_i$   
diventa l'R-  
valore di  $x_i$

$F(b_1, b_2, \dots, b_n)$

e i tipi?

$T \quad F(T_1 x_1, T_2 x_2, \dots, T_m x_m)$

$F(\dots bi \dots)$



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

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

conversione automatica e non ci sono scelte

PERICOLO

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(.....);
```

se definiamo F(...) senza specificare il tipo di ritorno, allora è implicitamente int

```
//PRE=(a e b >0 e a>=b)
```

```
int MCD(int a, int b)
```

```
{
```

```
int r=a % b;
```

```
while(r>0)
```

```
{ a=b; b=r; r=a%b;}
```

```
return b;
```

```
}
```

```
int mcm(int a, int b)
```

```
return (a*b) / MCD(a,b);
```

2 funzioni

le funzioni viste finora hanno una caratteristica

se abbiamo

```
void f(int x, double y) {.....}
```

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

```
f(x,y);
```

dopo l'invocazione x e y restano inalterati

si chiama assenza di side-effect

passaggio dei parametri per valore: no side effect

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

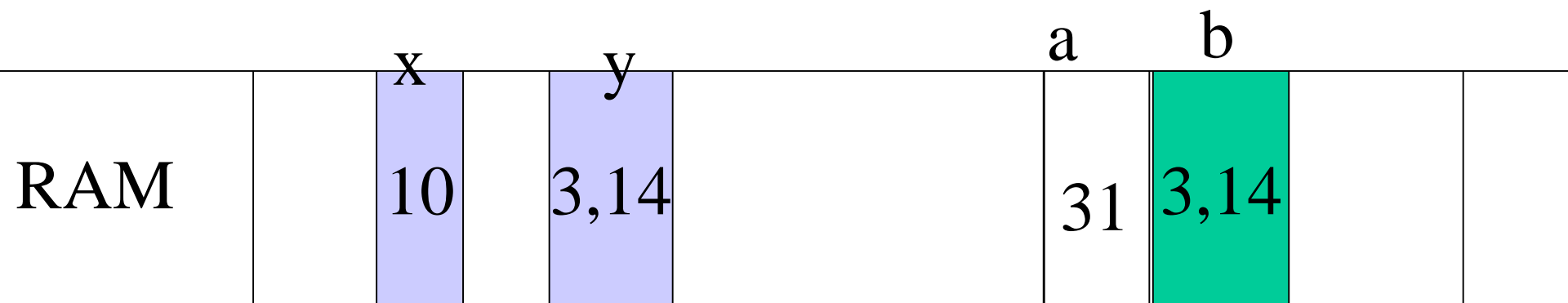
```
main()
```

```
{
```

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

```
  int z=F(x,y); //x e y non cambiano
```

```
}
```



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);  
}
```

ma funziona solo con 1 variabile



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

anziché passare alla funzione F per valore x e y,  
passiamo per valore il puntatore a x e quello a y

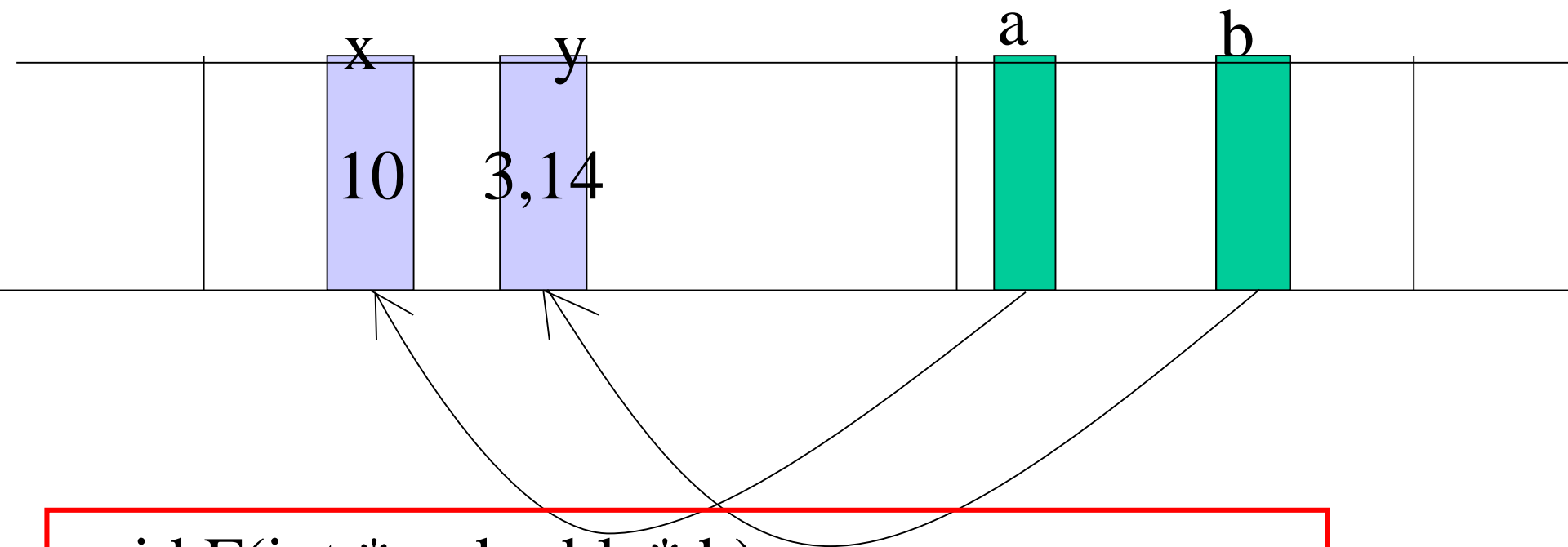
quindi avremo: `void F(int * a, double* b);`

per un parametro formale con tipo `int *` `x`  
il corrispondente parametro attuale deve fornire  
l'indirizzo di una variabile intera  
cioè un'espressione che ha un valore di tipo `int *`  
e lo stesso per `double*`

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

invocazione

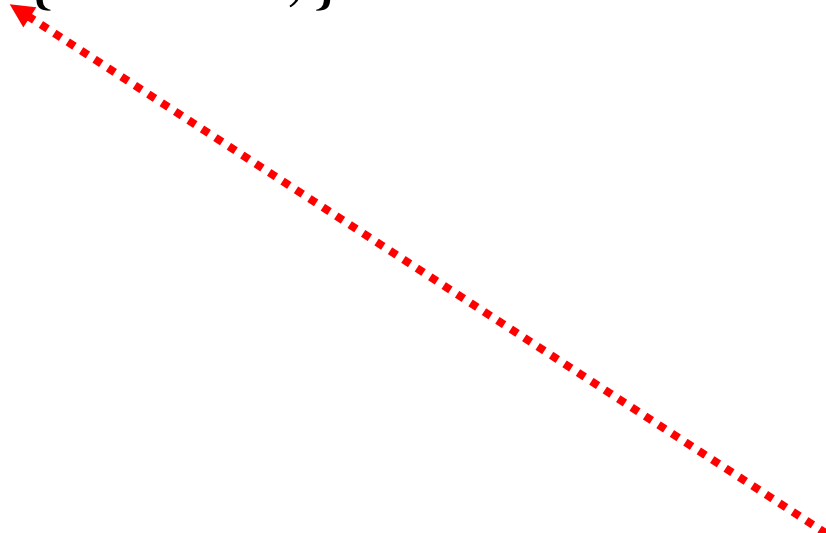
F(&x, &y);



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

## altro modo 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

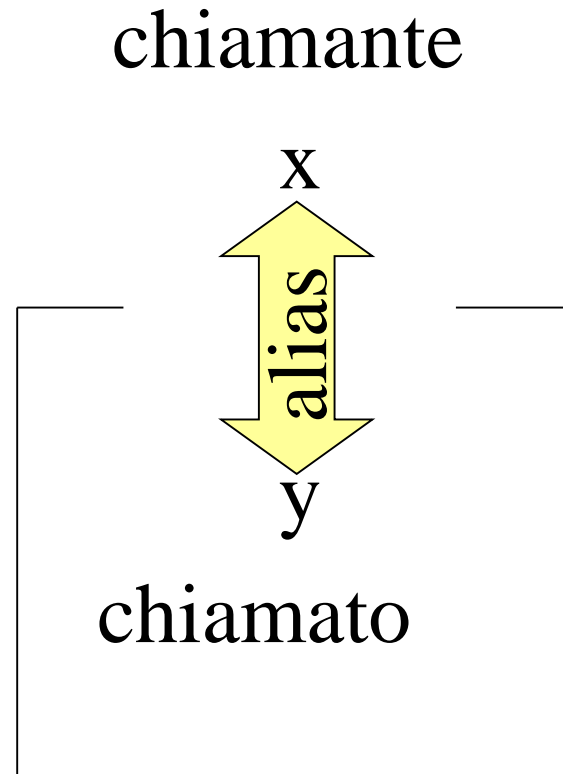
```
void g(int x, int & y)
{ x=y+1; }
main()
{
int A=10;
g(A,A);
} // valore di A ?
```

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



può servire  
in entrambe  
le direzioni

è diverso per i puntatori passati per valore ?

e ha senso passare un puntatore per riferimento?

e...



esercizio:

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

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

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

come deve essere      ? F( ?, ?){??}