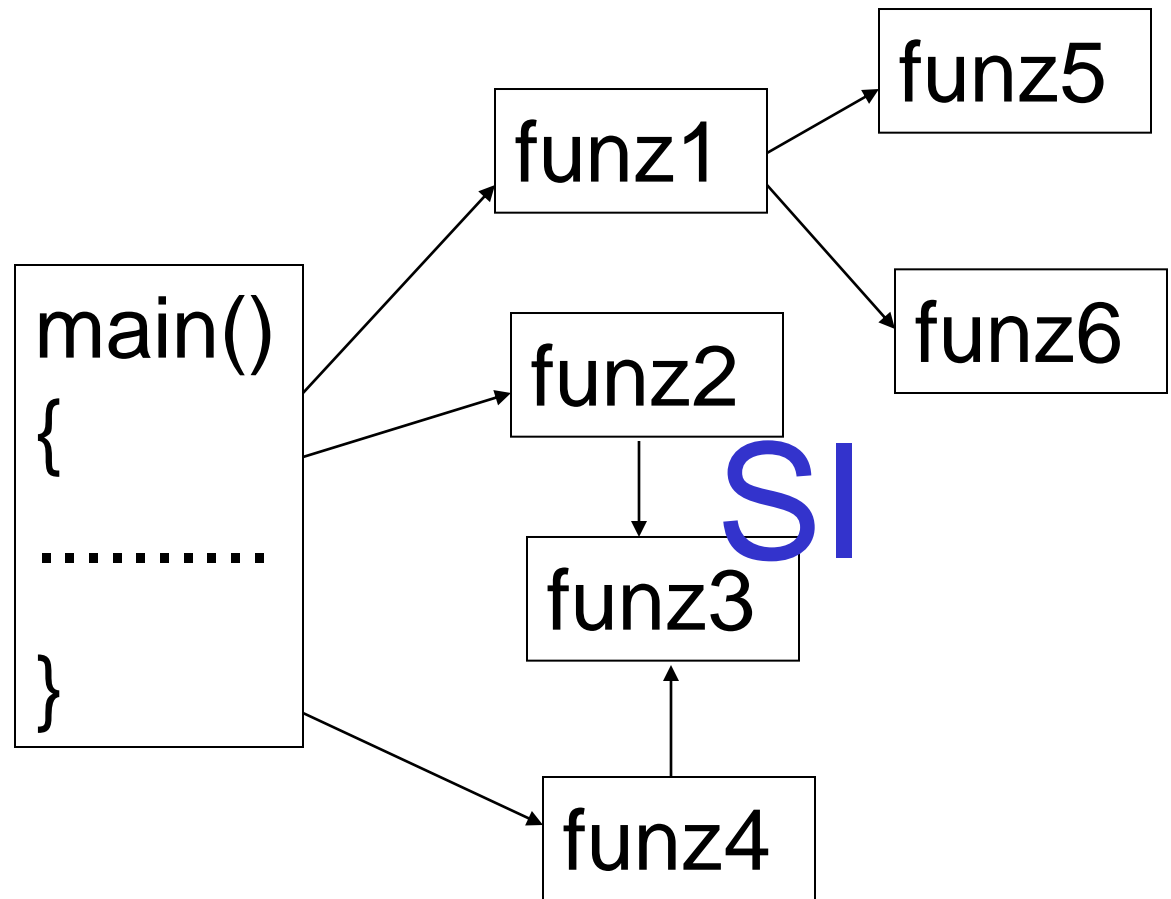


FUNZIONI

cap. 7 del testo

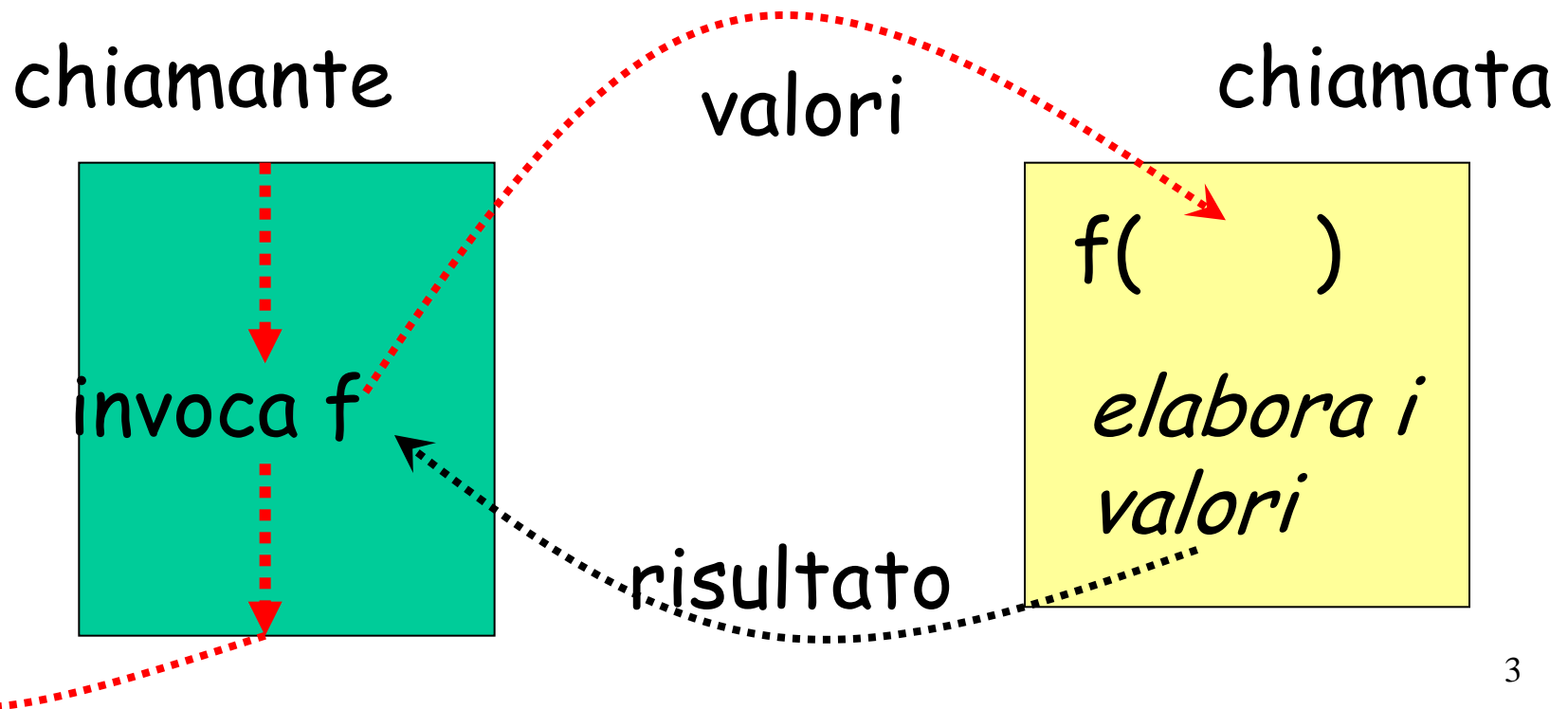
necessità di strutturare i programmi

```
main()
{
.....
.....
NO
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+1;
    while (x % y != 0)
        y--;
    return y;
}
```

entrata
valore su
cui
lavorare

x
parametro
formale

stesso
tipo

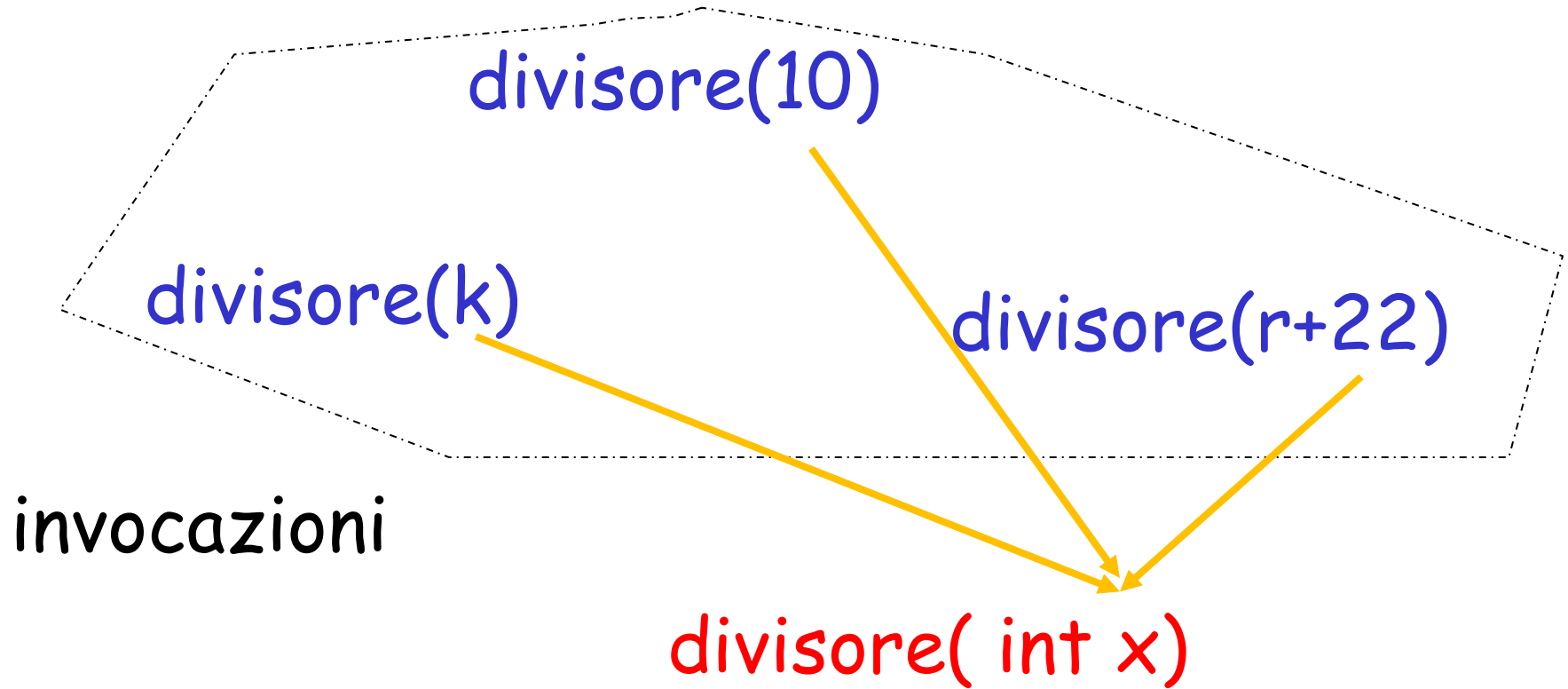
valore restituito

trova il massimo primo più piccolo o uguale di 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

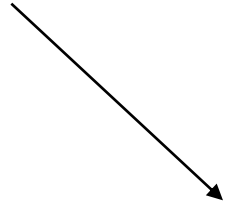
il valore di e_i
diventa l'R-
valore di x_i

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

e_i tipi?

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

$f(\dots ei \dots)$



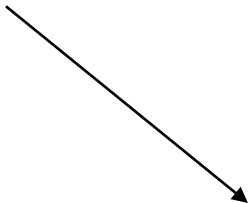
$T f(\dots Ti \ xi \dots)$

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

conversione obbligata come per le
assegnazioni (possibile warning)

IMPORTANTE

$f(\dots y \dots)$



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

l'R-valore di $xi ==$ R-valore di y

ma L-valore di $xi \neq$ L-valore di y

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) tale che :

int A = 10;

f(A);

e qui A ha R-valore 20 (o qualsiasi
valore diverso da 10)

non possiamo realizzare side-effect

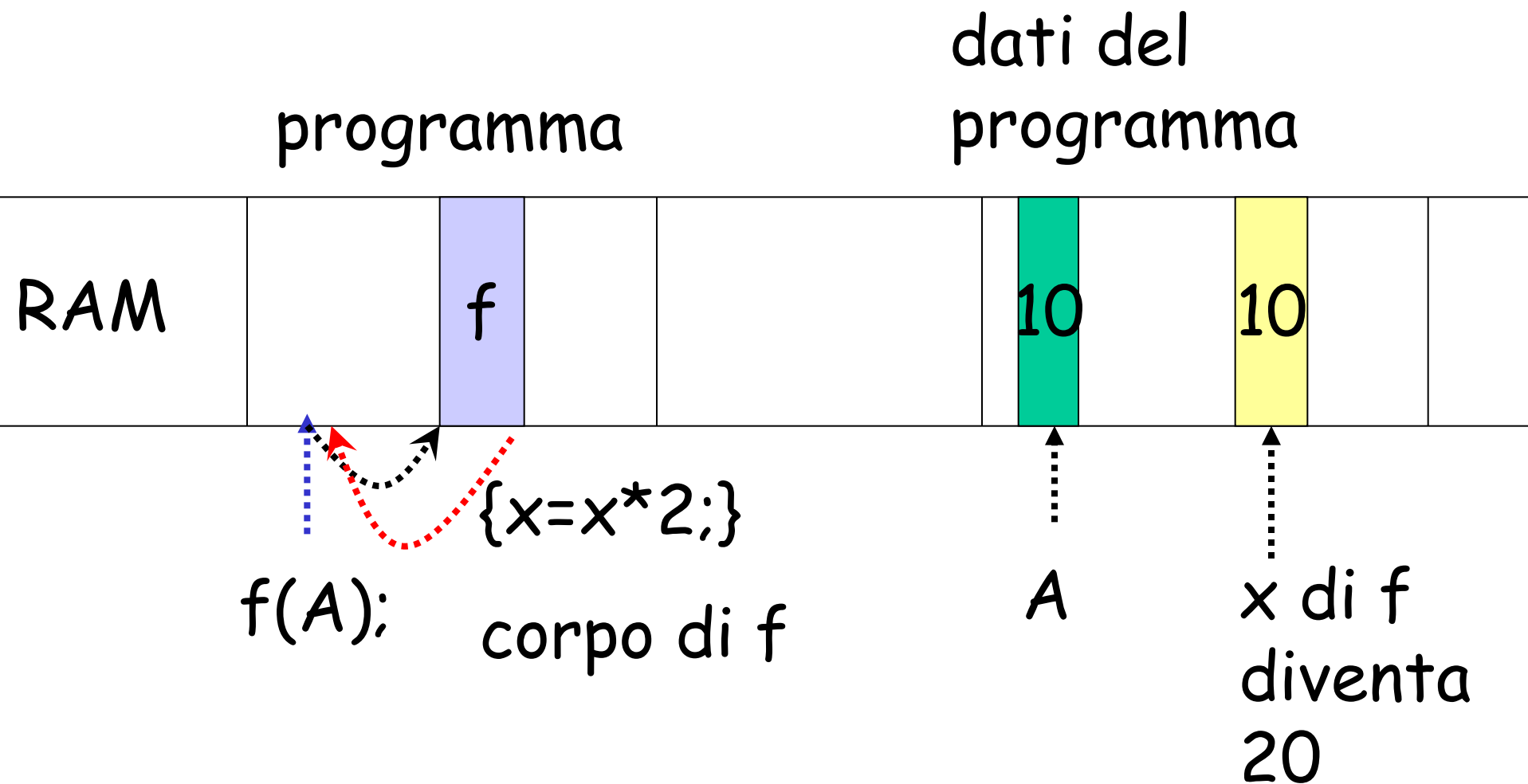
A=10



```
void f (int x)
{ x=x*2;}
```

x e A hanno diversi L-valori e quindi
x=x*2; non ha alcun effetto su A il
cui R-valore resta 10 !!

A non è visibile in f



al ritorno x sparisce e il suo spazio RAM è liberato

ma $x*2$ è l'R-valore che vorremmo dare ad A e quindi potremmo fare:

```
int f(int x)
```

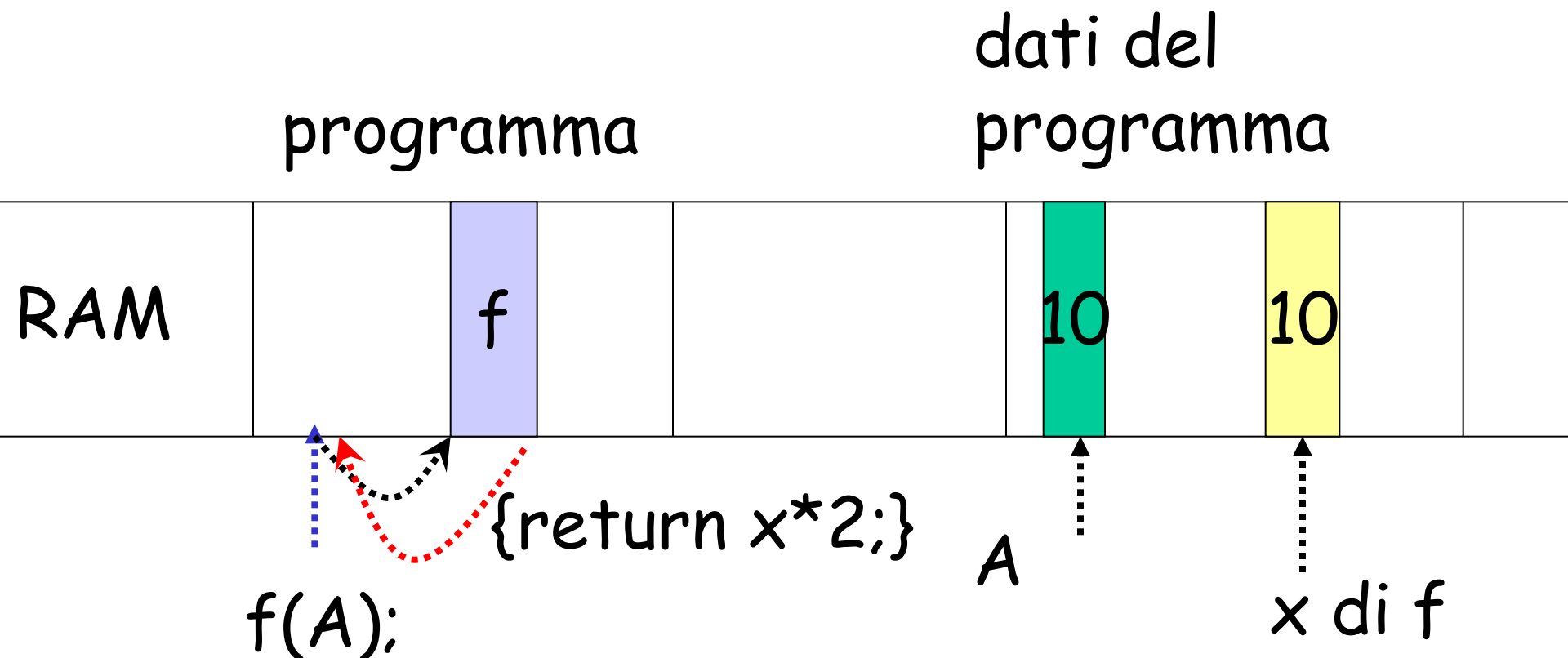
```
{return x*2;}
```

invocata con:

```
int A=10;
```

```
A=f(A);
```

ok per 1 variabile, ma se sono di più?



return $x*2$ ritorna 20 ad A e poi x
sparisce e il suo spazio RAM è liberato

per estendere le funzioni in modo che
possano fare side-effect

possiamo usare i **puntatori**

anziché passare **per valore** a f l'R-valore
della variabile A da modificare, passiamo
sempre per valore il puntatore alla
variabile A

quindi avremo: `void f(int * x);`

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

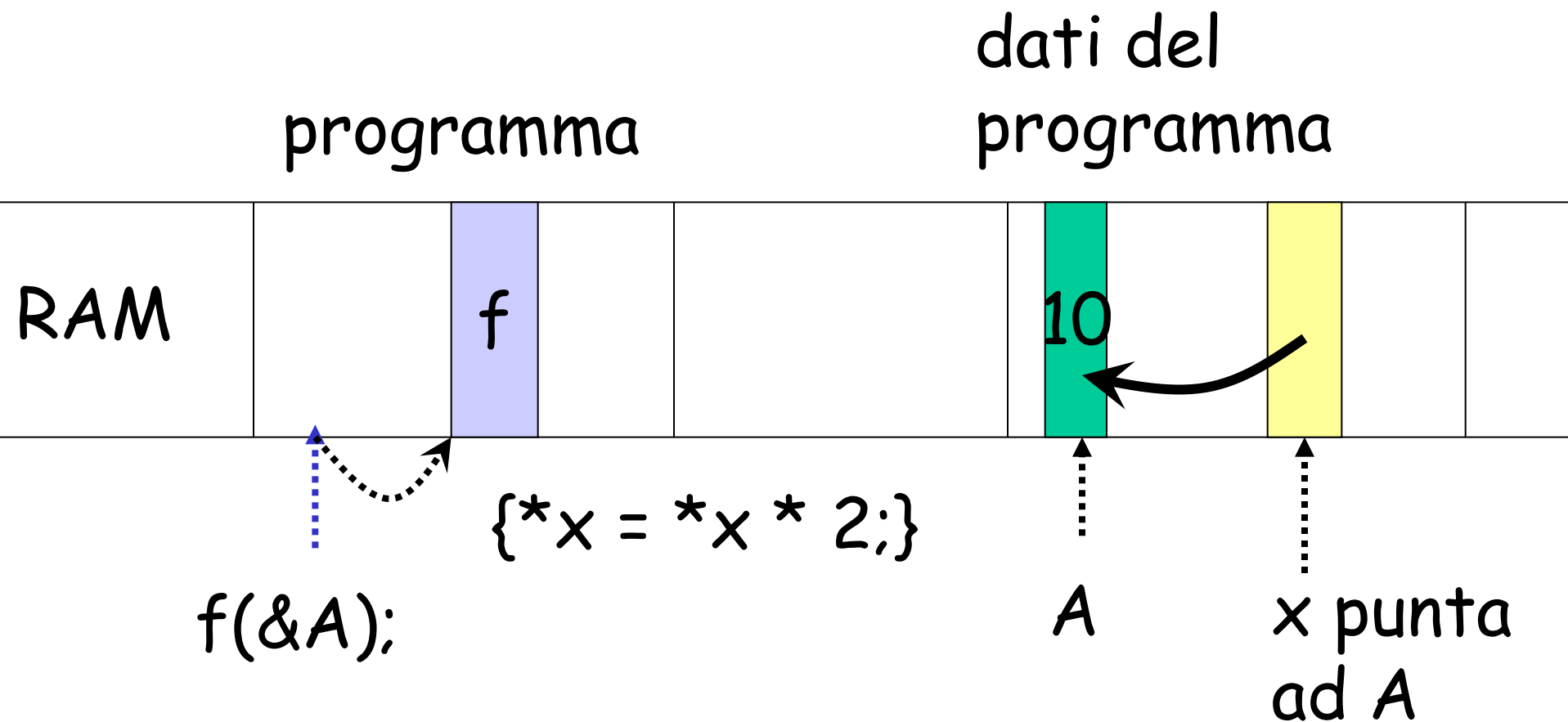
cioè `un'espressione` che ha un `valore di tipo`
`int *`

```
void f(int * x)
{
  *x=*x+2;
}
```

come invocarla
per cambiare la variabile A ?

f(&A);

passiamo l'L-valore di A



$*x$ = oggetto puntato da x cioè A

posso invocare f anche così:

```
int A=10, *p=&A;
```

```
f(p);    // passaggio per valore
```

```
qui A==20
```

RIFERIMENTI

i riferimenti ci permettono di creare **alias** di variabili

```
int x, &y=x;
```

y è un alias di x

cioè ha lo stesso R- e lo stesso L-valore

i riferimenti non esistono in C

sono introdotti nel C++ per facilitare il passaggio dei parametri alle funzioni

```
int x=2, &y=x;
```

```
cout<< x <<' '<< &x <<' '<< y <<' '<< &y;
```

se I indica l'L-valore di x, stampa:

```
2 I 2 I
```

x e y sono variabili con uguale
L-valore e quindi anche uguale
R-valore


```
int x, &y=x;
```

come viene realizzato un alias ?

con un puntatore !

in realtà `&y=x;` definisce un puntatore
`int *z = &x;` e ogni volta che scriviamo
`y` nel programma il compilatore lo
traduce in `*z`

tecnica usata in Java dove tutti
puntatori sono nascosti da riferimenti

regole dei riferimenti:

1) va inizializzato subito nella dichiarazione

```
int x, int & y; // NON VA !!!
```

```
y=x;
```

```
int x;
```

```
.....
```

```
int & y=x; // OK
```

2) non si possono definire puntatori a riferimenti:

```
int & * x; // NON è C++
```

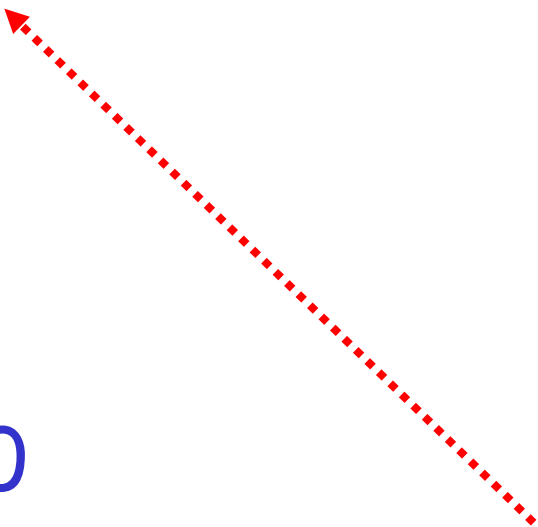
```
int x=1, &y=x; // da questo punto x e y  
              //sono la stessa  
              //variabile con 2 nomi
```

```
x++; y++;
```

```
cout << x << y; // stampa 3 3
```

passaggio dei parametri per riferimento

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



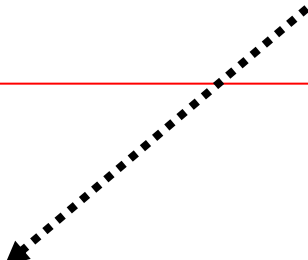
x è passato per
riferimento => x è
un alias di A

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

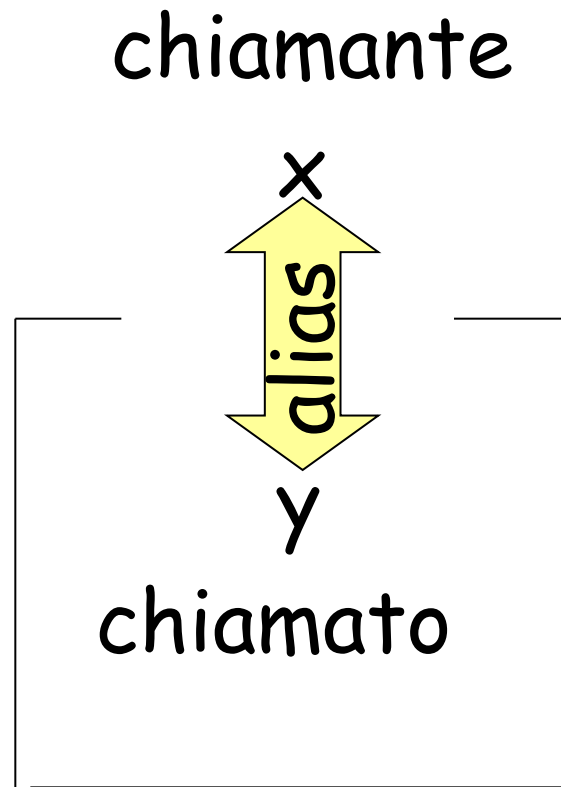
e con & ?

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(?, ?)

i parametri passati per riferimento sono realizzati con puntatori

```
int x;  
f(x) → f(&x)
```

```
void f(int &x) → void f(int *x)
```

i riferimenti sono introdotti in C++
perché