ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Funzioni in C

Funzioni

Spesso è utile poter riutilizzare un algoritmo

Qualche esempio:

- Calcolare un fattoriale
- Simulare un sistema dinamico
- Determinare il massimo tra due numeri

Non è un concetto completamente nuovo

Abbiamo già utilizzato funzioni in passato

- E.g. `sqrt, pow, exp`...

Funzioni

Dal punto di vista matematico un algoritmo calcola una funzione

- Riceve in ingresso dei dati (parametri)
- Restituisce dei risultati

Il C fornisce il concetto di funzione per incapsulare algoritmi

Una funzione in C è un costrutto che:

- Associa un nome ad un algoritmo
- Permette di utilizzarlo come una espressione
  - E.g. sqrt(exp(3 * x + 2))

Funzioni in C

La sintassi per definire una funzione in C è:

```
<funz.> ::= <tipo rit> <nome>(<param. formali>) <blocco>
<param. formali> ::= [<tipo> <nome> {, <tipo> <nome>}]
```

- `<nome>` è il nome della funzione
  - È un identificatore (stesse convenzioni dei nomi di variabile)
- `<tipo rit>` indica il tipo del risultato
- `<blocco>` è il codice (corpo) della funzione
- Per ognuno dei parametri, vanno indicati tipo e nome
- Se ci sono più parametri, vanno separati da virgole

Funzioni in C: Esempio di Definizione

Un esempio:

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}
```

Dalla prima riga capiamo che:

- Il nome della funzione è max
- Riceve in ingresso un parametro di tipo intero (a)
- ...Ed un secondo parametro di tipo intero (b)
- La funzione restituisce un intero

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Esempio di Definizione

Un esempio:

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}
```

Si chiama interfaccia di una funzione l'insieme di:

- Il nome della funzione
- Il suo tipo di ritorno
- La sequenza dei tipi dei parametri (i nomi non sono necessari)

È definita interamente nella prima riga

Funzioni in C: Esempio di Definizione

Un esempio:

```
int  max(int a, int b)  {
    if (a >= b) return a;
    else return b;
}
```

Il corpo della funzione specifica il suo comportamento

- È una porzione di codice (blocco)
- Viene eseguito ogni volta che la funzione viene chiamata

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Esempio di Definizione

Un esempio:

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}
```

L'istruzione return ha la sintassi:

```
return <expr>
```

- Causa la terminazione immediata della funzione
- Restituisce il valore denotato da <expr>

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Definizione e Chiamata

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}

int main() {
   int x = 3, y = 5, z;
   z = max(x, y);
}
```

Una volta definita `max`, la si può chiamare normalmente

- Si usa anche il termine invocazione
- Si usa la stessa sintassi che per le funzioni predefinite

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Definizione e Chiamata

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}

int main() {
   int x = 3, y = 5, z;
   z = max(x, y);
}
```

Di norma la definizione deve precedere la chiamata

- Il compilatore deve sapere che "max" esiste per chiamarla
- Più avanti vedremo una eccezione a questa regola

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Definizione e Chiamata

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}

int main() {
   int x = 3, y = 5, z;
   z = max(x, y);
}
```

Una chiamata a funzione ha tre passi fondamentali:

- Passaggio di parametri
- Esecuzione del corpo
- Restituzione del risultato

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Definizione e Chiamata

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}

int main() {
   int x = 3, y = 5, z;
   z = max(2*x, y);
}
```

Due classi di parametri:

- Parametri formali: specificati nella definizione
- Parametri attuali: specificati nella chiamata

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Definizione e Chiamata

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}

int main() {
   int x = 3, y = 5, z;
   z = max(2*x, y);
}
```

I **parametri formali**:

- Sono delle variabili (per questo hanno tipo e un nome)
- Sono locali alla funzione

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Definizione e Chiamata

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}

int main() {
   int x = 3, y = 5, z;
   z = max(2*x, y);
}
```

I **parametri attuali**:

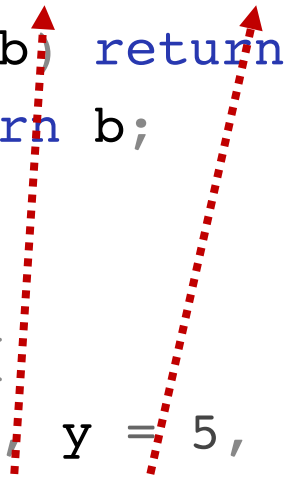
- Sono delle espressioni
- Vengono valutate prima di poter effettuare la chiamata

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzioni in C: Definizione e Chiamata

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}

int main() {
   int x = 3, y = 5, z;
   z = max(2*x, y);
}
```



Il passaggio dei parametri è posizionale:

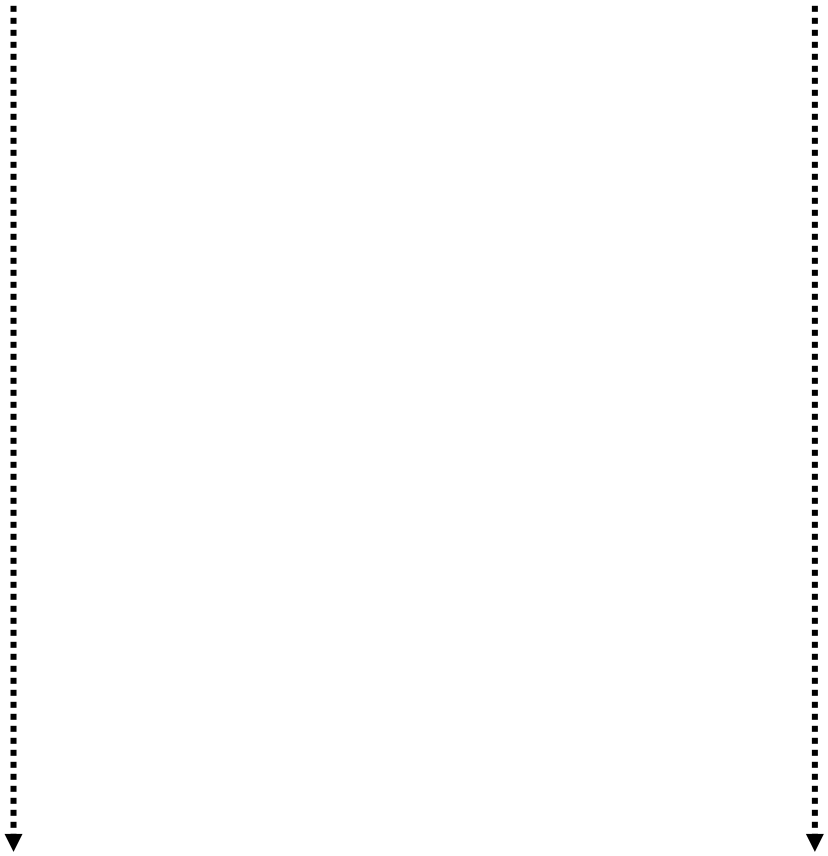
- Il risultato dell'i-mo parametro attuale...
- ...Viene usato per riempire l'i-mo parametro formale

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int  max(int a, int b)  {
    if (a >= b) return a;
    else return b;
}
```

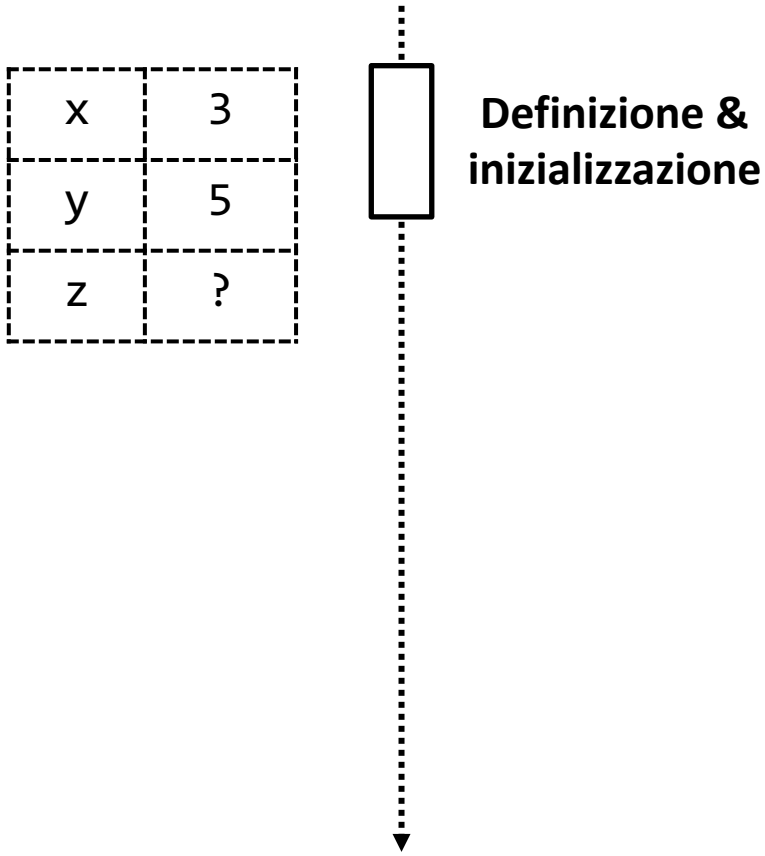


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int  max(int a, int b)  {
    if (a >= b) return a;
    else return b;
}
```

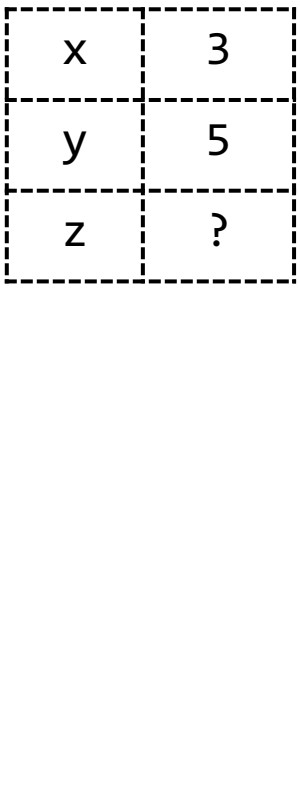


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int  max(int a, int b)  {
    if (a >= b) return a;
    else return b;
}
```



Valutazione dei parametri attuali

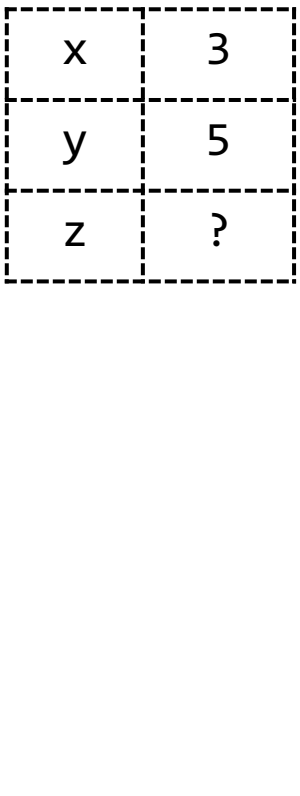
`2*x` → `6`, `y` → `5`

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int  max(int a, int b)  {
    if (a >= b) return a;
    else return b;
}
```



Parametri attuali

6, 5

Predisposizione di memoria per la funzione

a	?
b	?

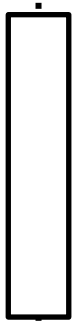
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int  max(int a, int b)  {
    if (a >= b) return a;
    else return b;
}
```

x	3
y	5
z	?



Trasferimento del flusso di controllo e dei parametri
(i.e. 6, 5)

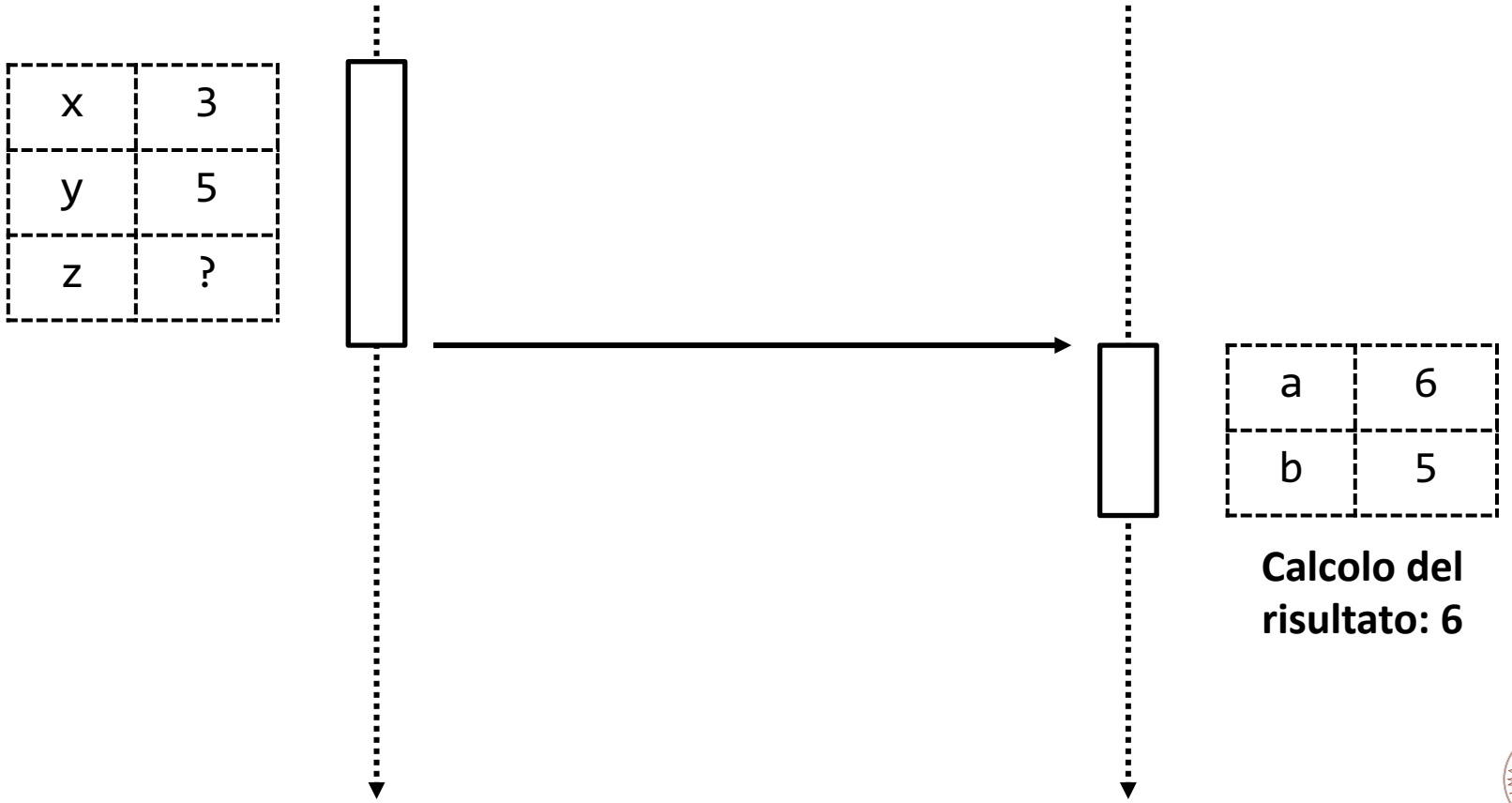
a	6
b	5

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int max(int a, int b) {
    if (a >= b) return a;
    else return b;
}
```

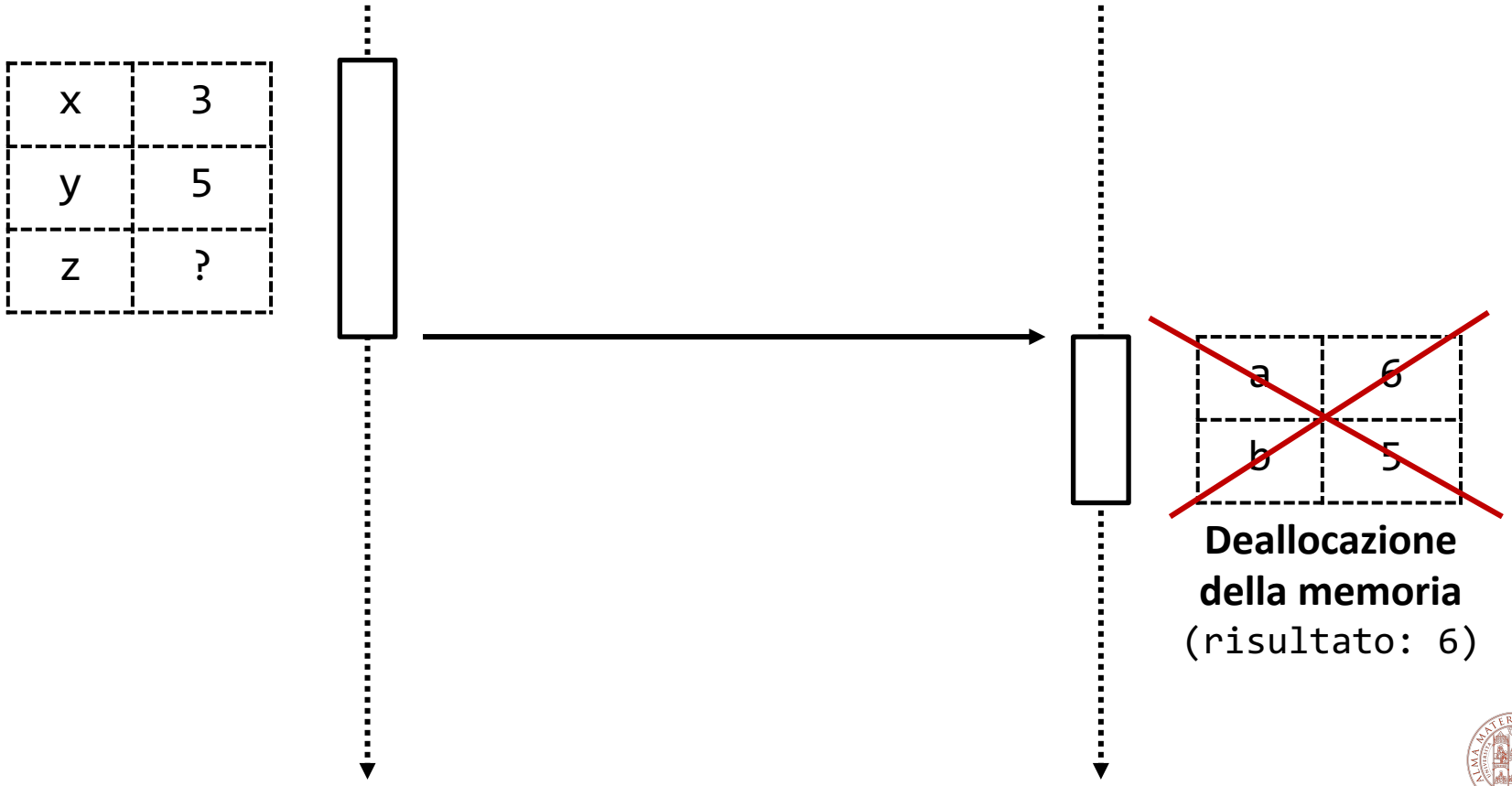


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

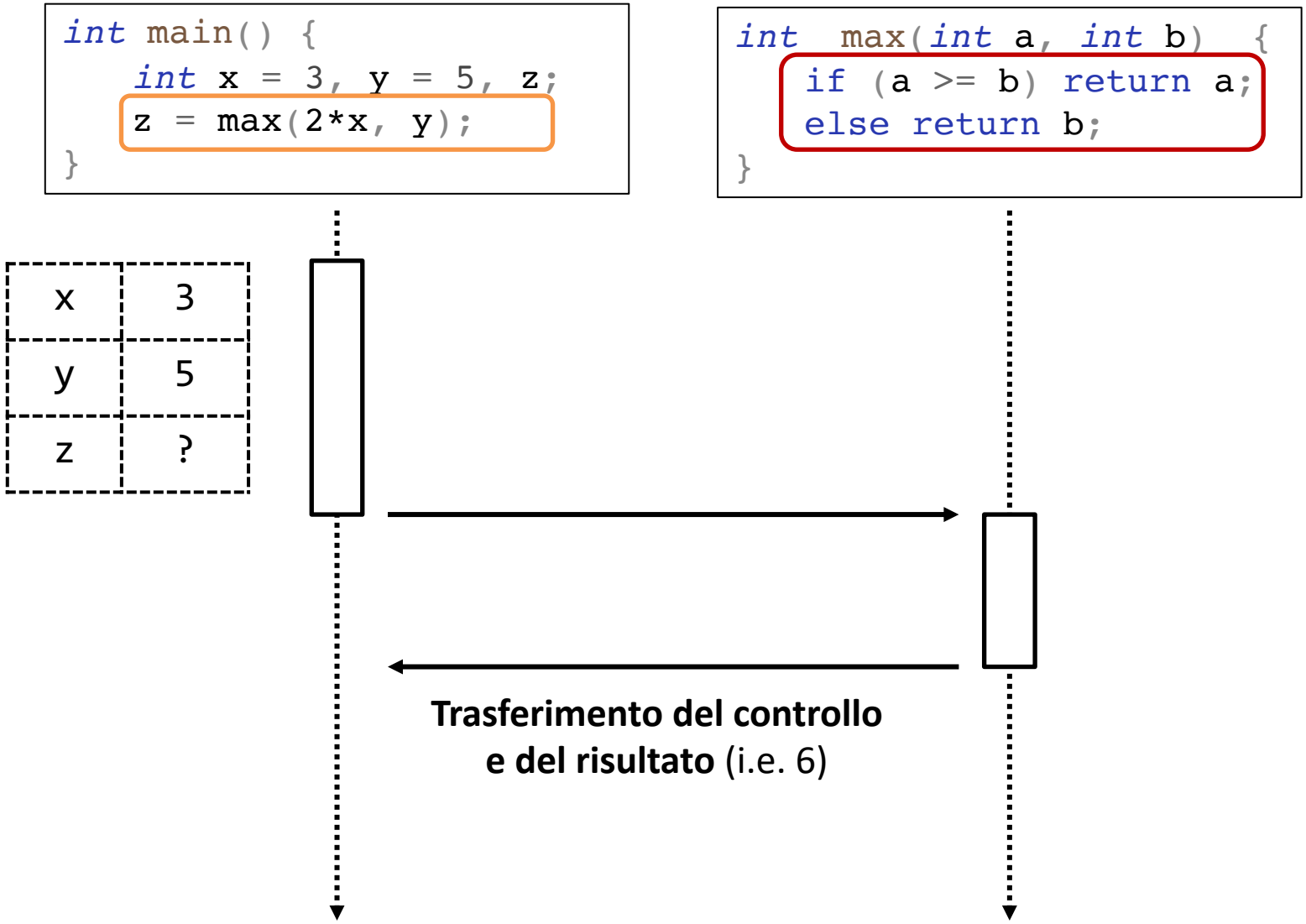
```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int max(int a, int b) {
    if (a >= b) return a;
    else return b;
}
```



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

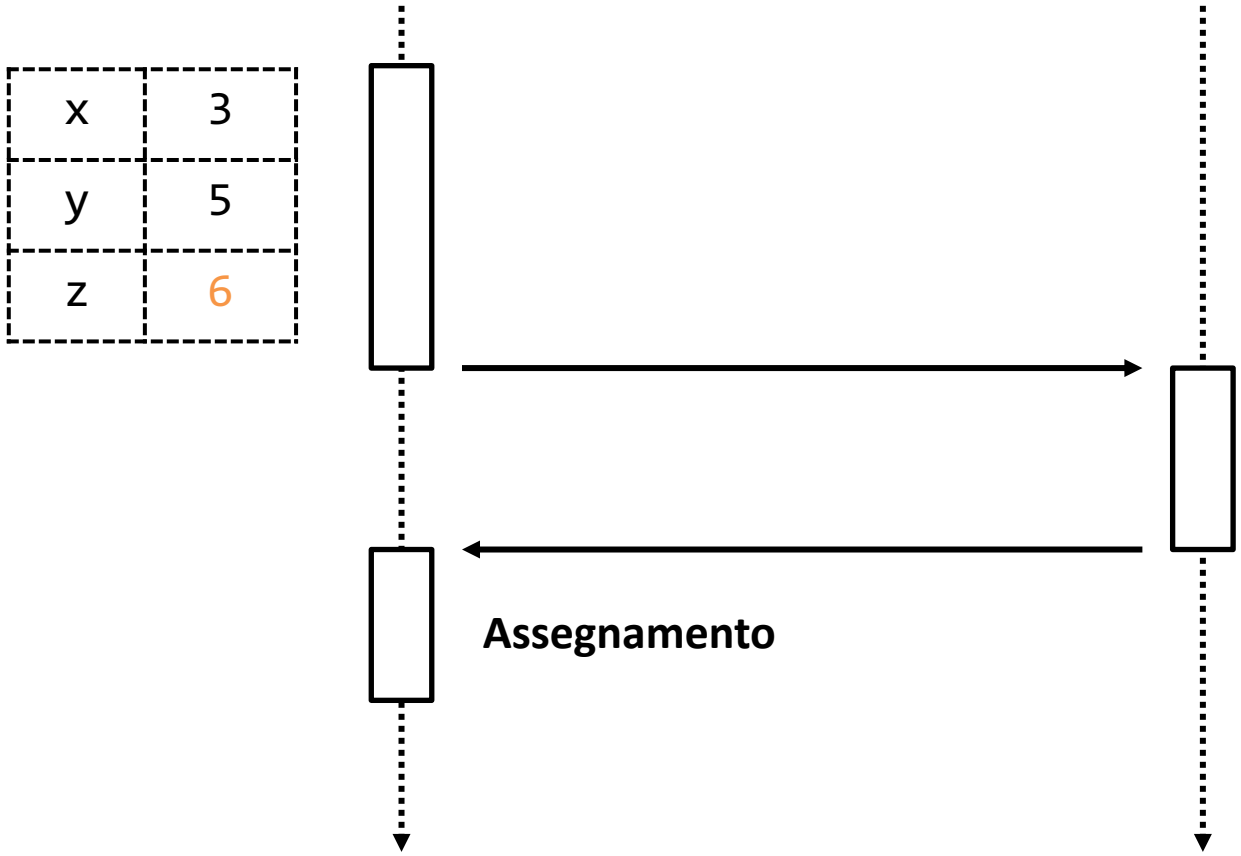


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dinamica di una Chiamata a funzione

```
int main() {
    int x = 3, y = 5, z;
    z = max(2*x, y);
}
```

```
int  max(int a, int b)  {
   if (a >= b) return a;
   else return b;
}
```



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Chiamata a Funzione

Questo meccanismo è comune a tutte le chiamate a funzione

- Valutazione dei parametri attuali
- Predisposizione di memoria per i parametri formali
- Copia dei parametri attuali in quelli formali
  - I parametri formali sono delle variabili!
  - Caratteristica: vengono inizializzate automaticamente
- Trasferimento del flusso di controllo alla funzione
- Esecuzione del codice della funzione
- Valutazione del risultato
- Trasferimento del risultato e del controllo al chiamante

Casi Particolari

Una funzione <u>può chiamare altre funzioni</u>

```
#include <math.h>
int dist(float x1, float y1, float x2, float y2) {
   int dx = x1 - x2, dy = y1 – y2;
   return sqrt(dx1*dx1 + dx2*dx2);
}
int main() {
   float ax = 2, ay = 1, bx = 0, by = 3;
   float z = dist(ax, ay, bx, by);
}
```

Il main chiama dist, che a sua volta chiama sqrt

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Casi Particolari

Una funzione può non avere alcun parametro:

- E.g. `rand()`, che restiuisce un numero casuale
- Un altro esempio:

```
float pigreco() {
    return 3.14159;
}
```

Casi Particolari

Se si termina senza return...

...Il valore restituito viene lasciato non-inizializzato!

```
#include <stdio.h>

float quadrato(float x) {
    float y = x*x;
}

int main() {
    printf("2^2 = %f\n", quadrato(2));
}
```

- È una situazione da evitare (con una eccezione...)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Casi Particolari

Il main è una funzione!

```
int main() {
    ...
}
```

- Viene automaticamente invocata all'inizio del programma
- Ha una interfaccia flessibile (qualche esempio nel tempo)
- Nella variante che usiamo:
  - Non riceve parametri
  - Restituisce un intero (codice di terminazione)
- Se manca `return`: restituisce 0 (è questa l'eccezione!)

Un Esempio: Valore Assoluto

```c
#include <stdio.h>

float val_ass(float x) {
    if (x >= 0) return x;
    else return -x;
}

int main() {
    int a = 10, b = -5;
    printf("|a| = %f\n", val_ass(a));
    printf("|b| = %f\n", val_ass(b));
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Un Esempio: Zeta di Riemann

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

float zeta(float s) {
    float z = 0, zo;
    for(int n = 1; n <= 1000; ++n) {
        zo = z;
        z += 1.0 / pow(n, s);
        if (z - zo < 1e-6) break;
    }
    return z;
}

int main() {
    float s = 1.5;
    printf("zeta(%f) = %f\n", s, zeta(s));
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Crescita Logistica

Il modello di crescita logistica è una successione matematica

è definita dalla ricorsione:

$$x^{(t)} = rx^{(t-1)}\left(1 - \frac{x^{(t-1)}}{N}\right)$$

Il modello descrive l'evoluzione di una popolazione

- x(0) è la popolazione iniziale
- r è il tasso di crescita della popolazione
- N è la massima popolazione sostenibile
- La successione permette di ottenere $x^{(1)}$ da $x^{(0)}$, $x^{(2)}$ da $x^{(1)}$ etc.

Si implementi il modello in C

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Crescita Logistica

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

int f(float x, float r, float N) { // funzione di transizione
    return r * x * (1 - x / N);
}

int main() {
    int n = 10;
    float r = 1.2, N = 1000, x = 300;

    printf("Popolazione iniziale: %f\n", x);
    for(int t = 1; t <= n; ++t) {
        x = f(x, r, N);
        printf("Popolazione all passo %d: %f\n", t, x);
    }
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Modello di Beverton-Holt

Il modello di Beverton-Holt è una successione matematica

è definita dalla ricorsione:

$$x^{(t)} = \mathrm{r}\frac{x^{(t-1)}}{1 + \frac{x^{(t-1)}}{N}}$$

Il modello descrive l'evoluzione di una popolazione

- x(0) è la popolazione iniziale
- r è il tasso di crescita della popolazione
- N è la massima popolazione sostenibile
- La successione permette di ottenere $x^{(1)}$ da $x^{(0)}$, $x^{(2)}$ da $x^{(1)}$ etc.

Si implementi il modello in C

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Modello di Beverton-Holt

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

int f(float x, float r, float N) { // cambia solo questa!
    return r * x / (1 + x / N);
}

int main() {
    int n = 10;
    float r = 1.2, N = 1000, x = 300;

    printf("Popolazione iniziale: %f\n", x);
    for(int t = 1; t <= n; ++t) {
        x = f(x, r, N);
        printf("Popolazione all passo %d: %f\n", t, x);
    }
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Funzioni e Variabili

Record di Attivazione

Quando si chiama una funzione

Viene predisposta per essa un'area di memoria

- Questa si dice record di attivazione
- Ospita tutte le variabili definite dalla funzione
- ...Inclusi i parametri formali
- Al termine della funzione viene distrutto

Qualche conseguenza importante:

- Il tempo di vita delle variabili definite da una funzione...
- ...È lo stesso del record di attivazione (i.e. la chiamata)

Variabili Locali

Le variabili definite in una funzione sono locali ad essa

- Il loro tempo di vita coincide con quello di una chiamata
- Sono accessibili solo dall'interno di una funzione

```
int f(int x) {
   int y = 2*x;
   return y;
} // "y" è distrutta al termine della chiamata

int main() {
   int a = 10;
   int z = f(a);
} // "a" è distrutta al termine del main
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Variabili Locali

Le variabili definite in una funzione sono locali ad essa

- Il loro tempo di vita coincide con quello di una chiamata
- Sono accessibili solo dall'interno di una funzione

```
int f(int x) {
   int y = 2*a; // "a" non è accessibile in "f"
   return y;
}

int main() {
   int a = 10; // "a" è locale al main
   int z = f(a);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Variabili Locali

Le variabili definite in una funzione sono locali ad essa

- Il loro tempo di vita coincide con quello di una chiamata
- Sono accessibili solo dall'interno di una funzione

```
int f(int x) { // questa "x" è locale a "f"
   int y = 2*x;
   return y;
}

int main() {
   int a = 10, x = 5; // questa "x" è locale al main
   int z = f(a);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Variabili Locali

Le variabili definite in una funzione sono locali ad essa

- Il loro tempo di vita coincide con quello di una chiamata
- Sono accessibili solo dall'interno di una funzione

```
int f(int x) {
   int y = 2*x; // qui "x" vale "10"!
   return y;
}

int main() {
   int a = 10, x = 5;
   int z = f(a); // viene passato "10"!
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Variabili Locali

Le variabili definite in una funzione sono locali ad essa

- Il loro tempo di vita coincide con quello di una chiamata
- Sono accessibili solo dall'interno di una funzione

Ogni funzione ha un ambiente separato

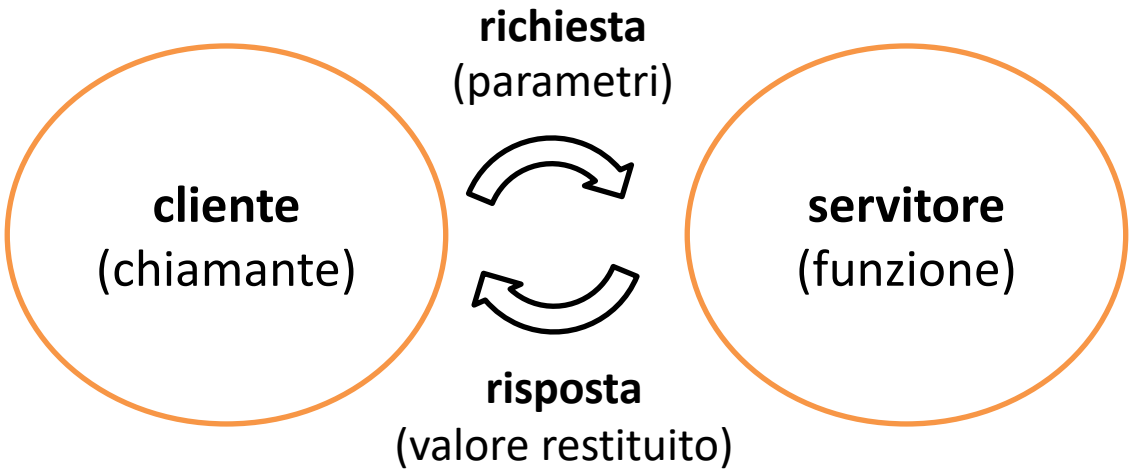
- Eventuali variabili con lo stesso nome...
- ...Sono in realtà variabili distinte

L'unica comunicazione possibile è quella definita dall'interfaccia

- Dal chiamante alla funzione chiamata: valori dei parametri
- Dalla funzione chiamata al chiamante: valore restituito

Modello Cliente/Servitore (Client/Server)

Le funzioni in C seguono il modello cliente-servitore

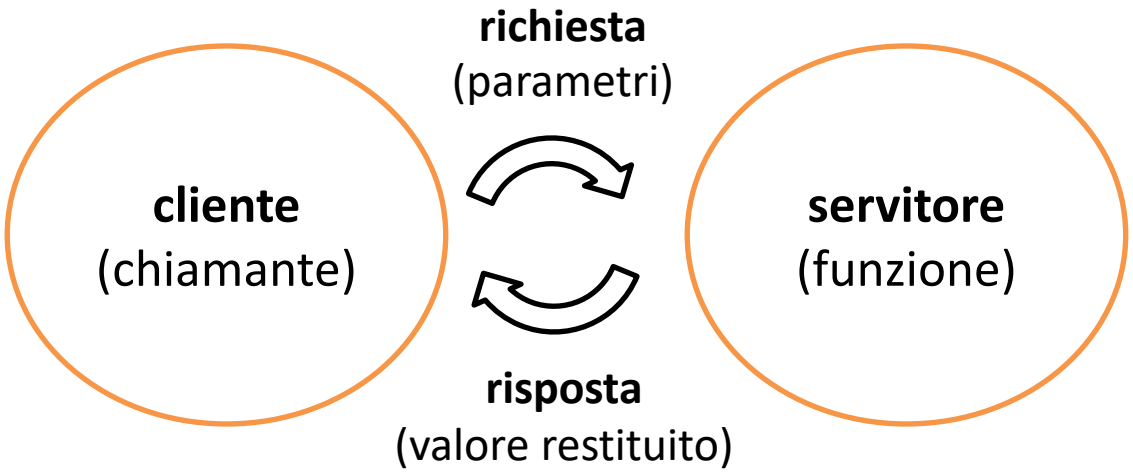


- **Servitore** = un ente computazionale che può nascondere la propria struttura interna
- **Cliente =** un ente computazionale in grado di invocare uno o più servitori

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Modello Cliente/Servitore (Client/Server)

Le funzioni in C seguono il modello cliente-servitore

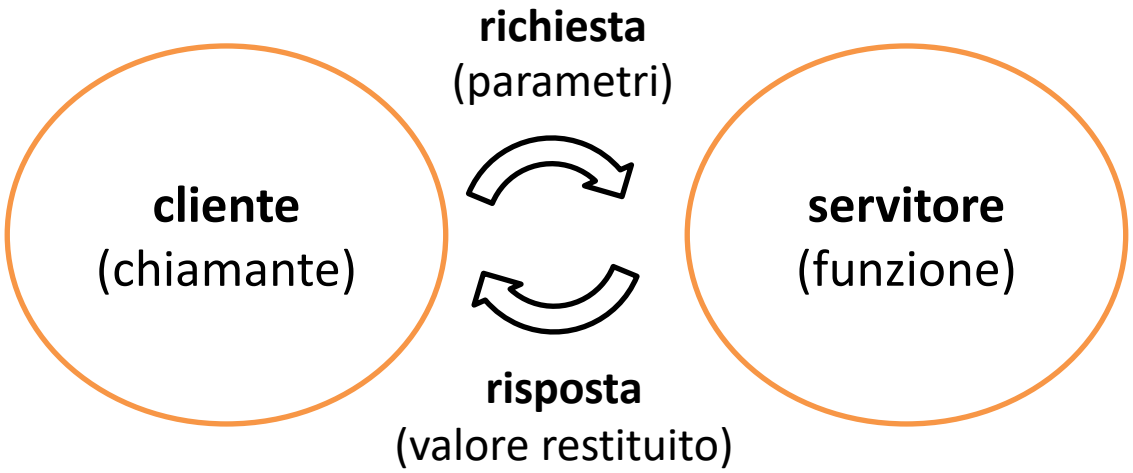


Il modello cliente/servitore:

- È uno dei meccanismi di base per ottenere astrazione
- Un servitore può essere usato senza conoscerne i dettagli!
- È così che finora avete usato printf, exp, sqrt ecc.

Modello Cliente/Servitore (Client/Server)

Le funzioni in C seguono il modello cliente-servitore



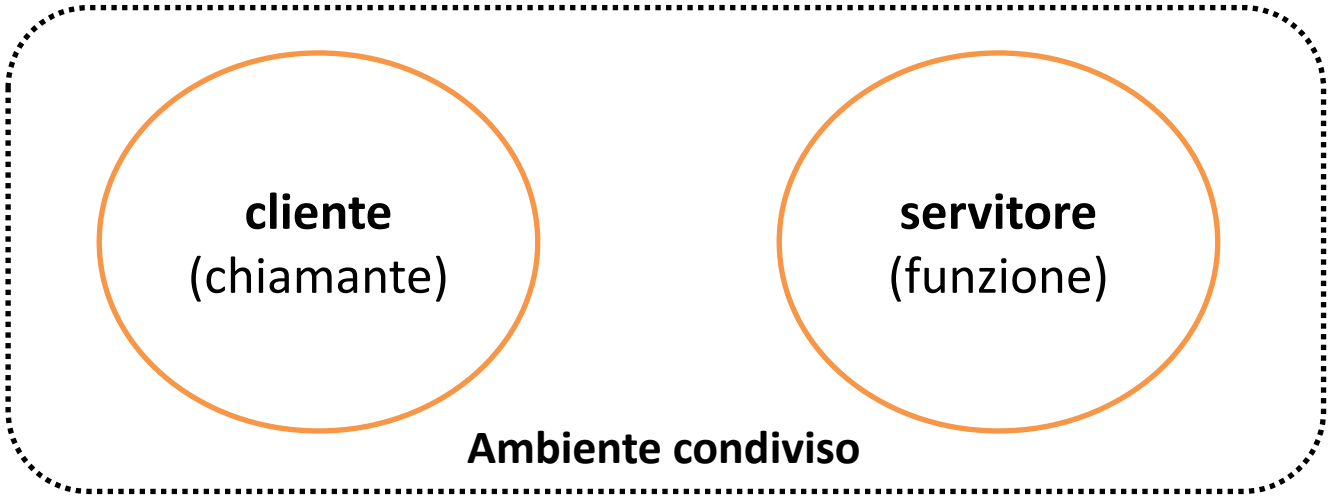
Il tipo di comunicazione vista finora

- Si dice "a scambio di messaggi"
- Il cliente invia un messaggio (invocazione + parametri)
- Il servitore risponde con un messaggio (risultato)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Modello Cliente/Servitore (Client/Server)

Le funzioni in C seguono il modello cliente-servitore



Esiste un metodo di comunicazione alternativo:

- Si dice "ad ambiente condiviso"
- Cliente e servitore possono accedere agli stessi dati
- In C lo si può realizzare mediante le variabili globali

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Variabili Globali

In C, una variabile può essere definita fuori da una funzione:

- In questo caso la variabile si dice globale

Un esempio:

```
#include <stdio.h>

int x = 10;

int main() {
   printf("x vale %d\n", x);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Variabili Globali

Una variabile globale

- Ha lo stesso tempo di vita del programma
- È accessibile nell'intero file in cui è definita

```c
#include <stdio.h>

int x = 10;

int main() {
   printf("x vale %d\n", x); // "x" è accessibile!
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Variabili Globali: un Esempio

L'uso di variabili globali in generale va evitato

- È difficile tenere traccia di quali funzioni le modifichino
- Quando il codice diventa complesso è facile fare errori

```
int s = 1;
int fattoriale(int n) {
   for(int i = 0; i  < n; ++i)  s *= i;
   return s;
}

int main() {
   int z = fattoriale(4); // risultato corretto
   int z2 = fattoriale(5); // errore! "s" non resettato
}
```

Variabili Globali: un Esempio

Possono essere comode per memorizzare uno stato globale

```c
#include <stdio.h>
int cnt = 0; // contatore di invocazioni di "f"

int f (int x) {
   cnt++;
   return x;
}

int main() {
   f(2); // "cnt" vale 1 dopo l'invocazione
   f(7); // "cnt" vale 2
   f(-2); // "cnt" vale 3
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Procedure e Puntatori

Procedure

Non sempre ci interessa stampare un valore

E.g. una funzione che stampi il quadrato di un valore

```
int stampa_quad(int x) {
   printf("%d", x*x);
   return 0;
}
```

- Il valore restituito è superfluo
- Sarebbe più chiaro indicare che non è restituito alcun valore

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedure

Non sempre ci interessa stampare un valore

E.g. una funzione che stampi il quadrato di un valore

```
void stampa_quad(int x) { // Uso void invece di int
   printf("%d", x*x);
   return; // Niente valore di ritorno
}
```

Quando non viene restituito alcun valore:

- Per il tipo di ritorno, si usa la parola chiava `void`
- Non c'è bisogno di specificare una istruzione in `return`

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedure

Non sempre ci interessa stampare un valore

E.g. una funzione che stampi il quadrato di un valore

```
void stampa_quad(int x) { // Uso void invece di int
   printf("%d", x*x);
}
```

Quando non viene restituito alcun valore:

- Per il tipo di ritorno, si usa la parola chiava `void`
- Non c'è bisogno di specificare una istruzione in `return`
- Arrivare a fine funzione senza `return` non causa problemi

Procedure

Non sempre ci interessa stampare un valore

E.g. una funzione che stampi il quadrato di un valore

```
void stampa_quad(int x) { // Uso void invece di int
   printf("%d", x*x);
}
```

Quando non viene restituito alcun valore:

- Si parla di procedura, anzichè di funzione
- Una funzione permette di dare un nome ad una espressione
- Una procedura permette di dare un nume ad una istruzione

Scambio di Variabili

A cosa può servire una procedura?

- Visualizzare/scrivere informazioni
- Modificare dati

Supponiamo di voler scambiare due variabili

Un approccio naïve:

```
void scambia(int a, int b) {
   int t = a; // salvo li valore di a
   a = b; // sovrascrivo a con b
   b = t; // scrivo il vecchio valore di a in b
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scambio di Variabili

Proviamo ad usare la nostra procedura:

```
void scambia(int a, int b) {
   int t = a; // salvo li valore di a
   a = b; // sovrascrivo a con b
   b = t; // scrivo il vecchio valore di a in b
}
int main() {
   int x = 10, y = 7;
   scambia(x, y);
   printf("x = %d, y = %d\n", x, y); // x= 10, y = 7
}
```

Non funziona! Perchè?

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scambio di Variabili

Proviamo ad usare la nostra procedura:

```
void scambia(int a, int b) {
   int t = a; // salvo li valore di a
   a = b; // sovrascrivo a con b
   b = t; // scrivo il vecchio valore di a in b
}
int main() {
   int x = 10, y = 7;
   scambia(x, y);
   printf("x = %d, y = %d\n", x, y); // x= 10, y = 7
}
```

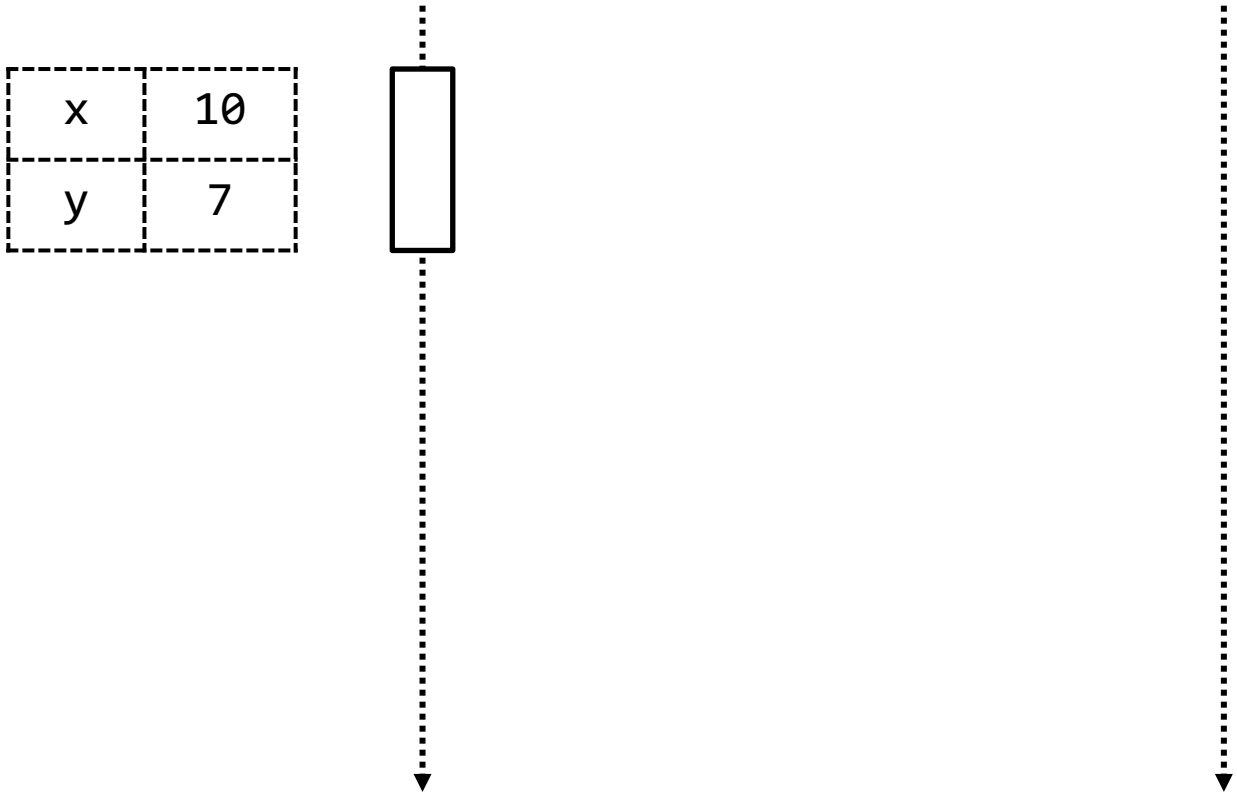
- "a" e "b" vengono effettivamente scambiate
- ...Ma a e b sono locali alla funzione!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedura Scambia: Esecuzione

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```

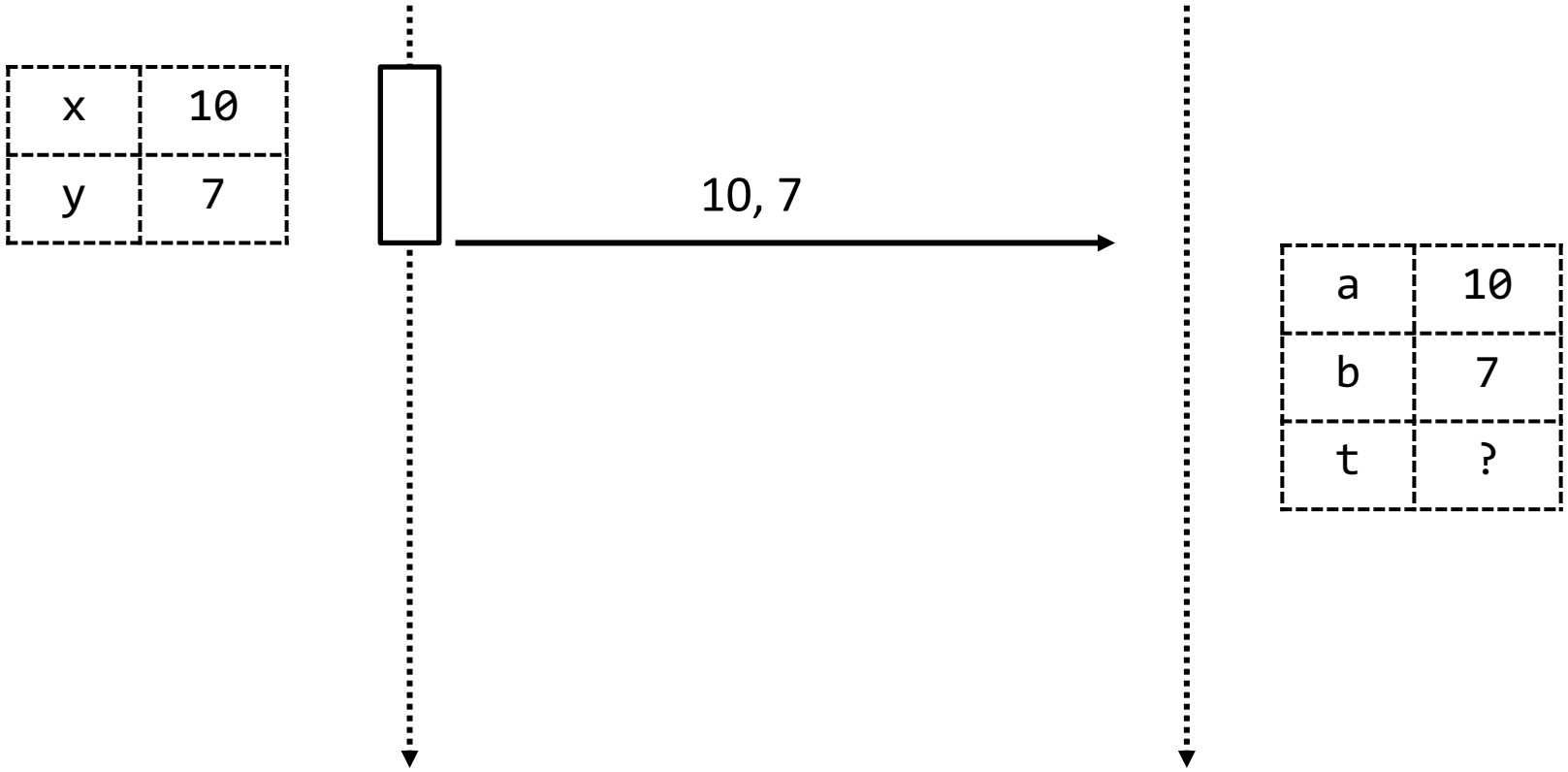


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedura Scambia: Esecuzione

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```

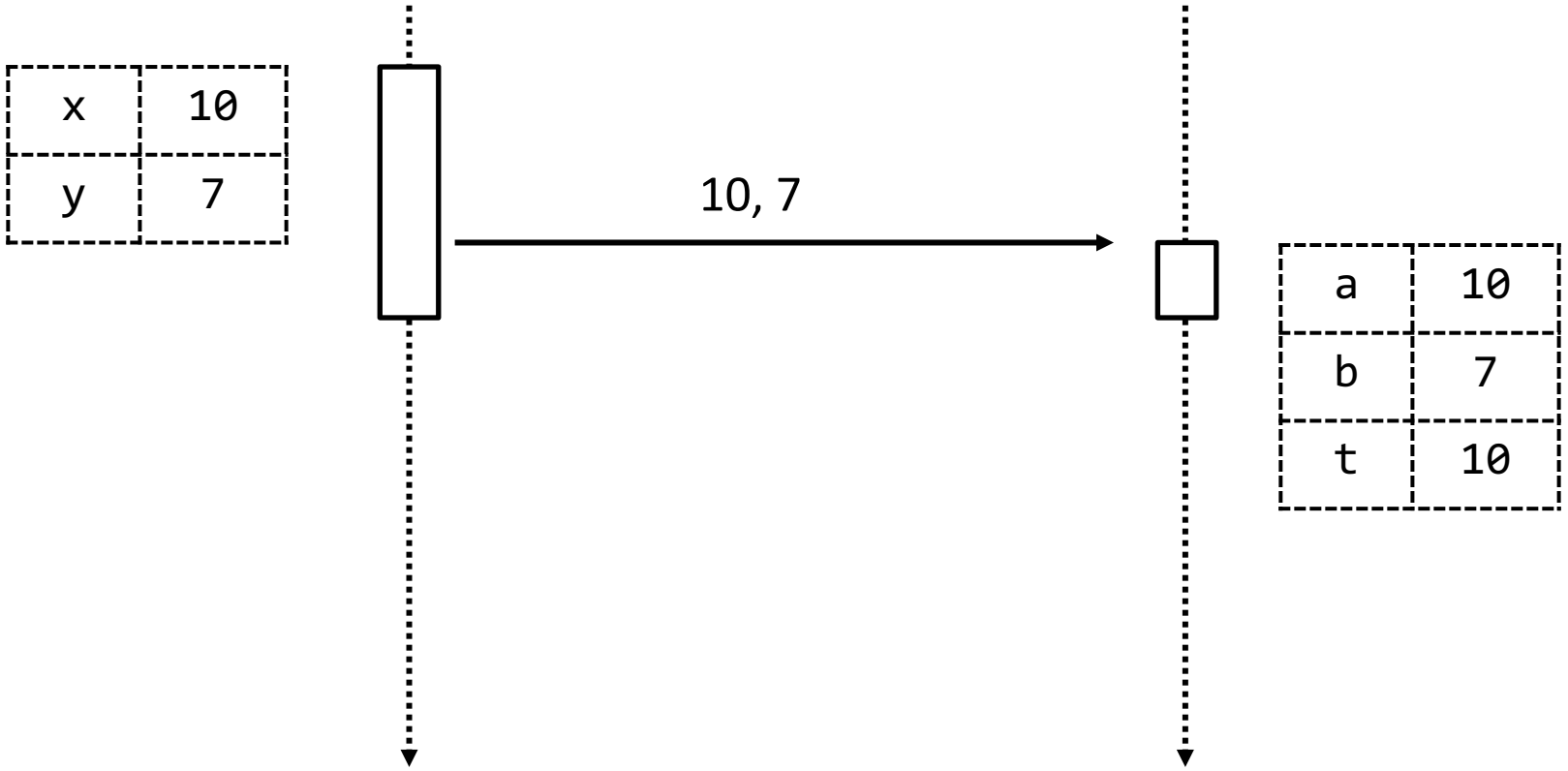


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedura Scambia: Esecuzione

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```

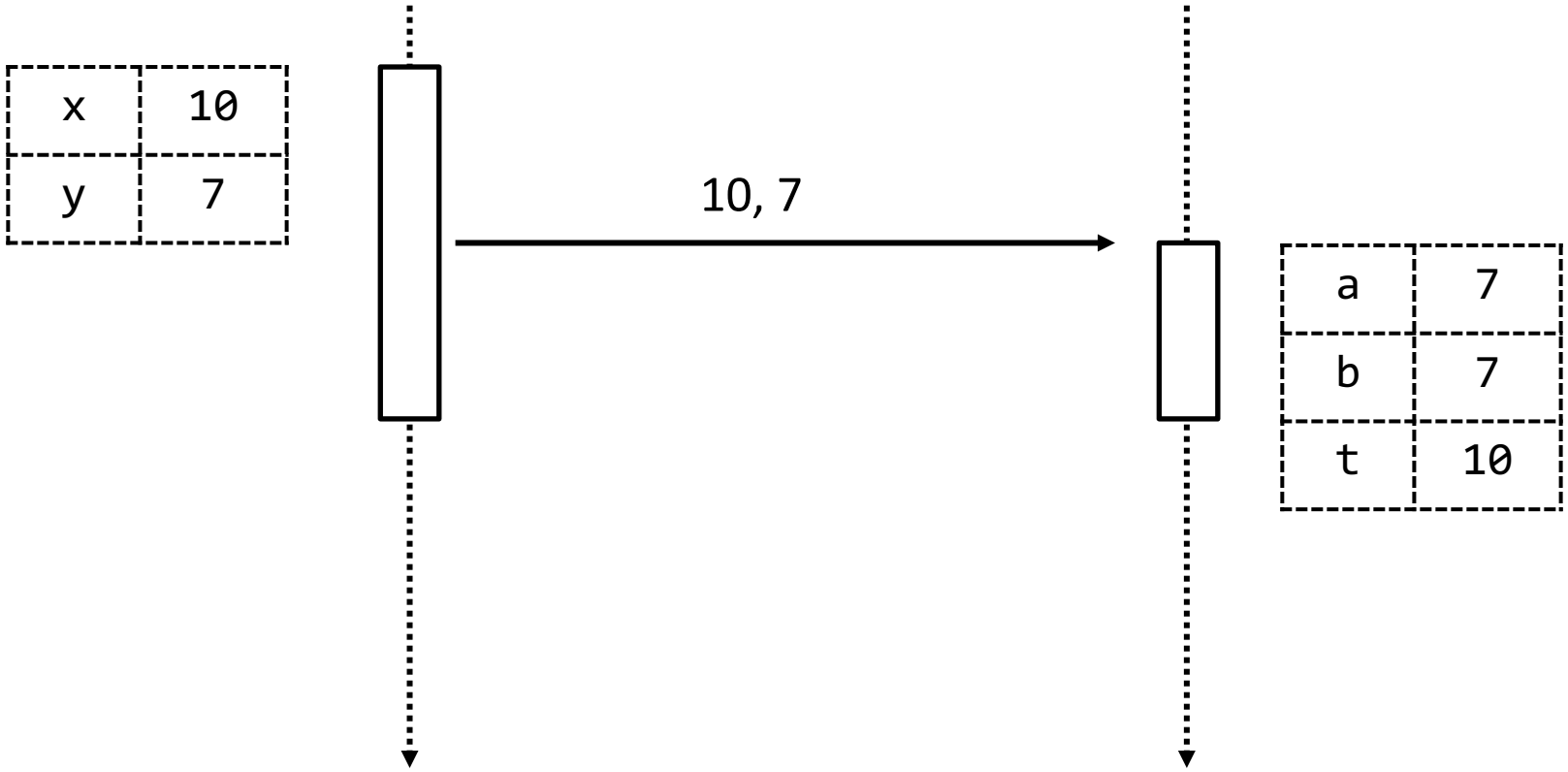


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedura Scambia: Esecuzione

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```

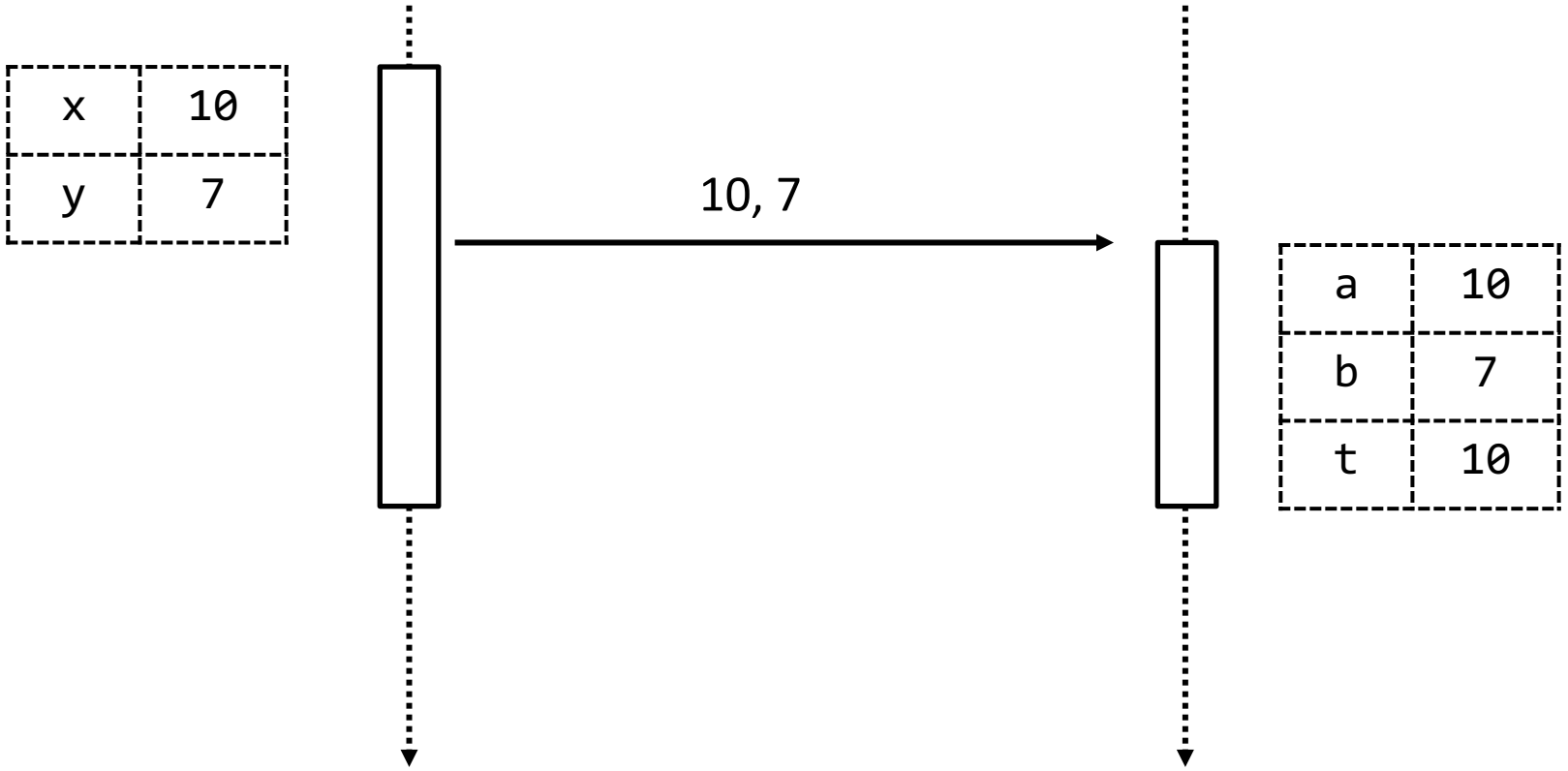


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedura Scambia: Esecuzione

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```

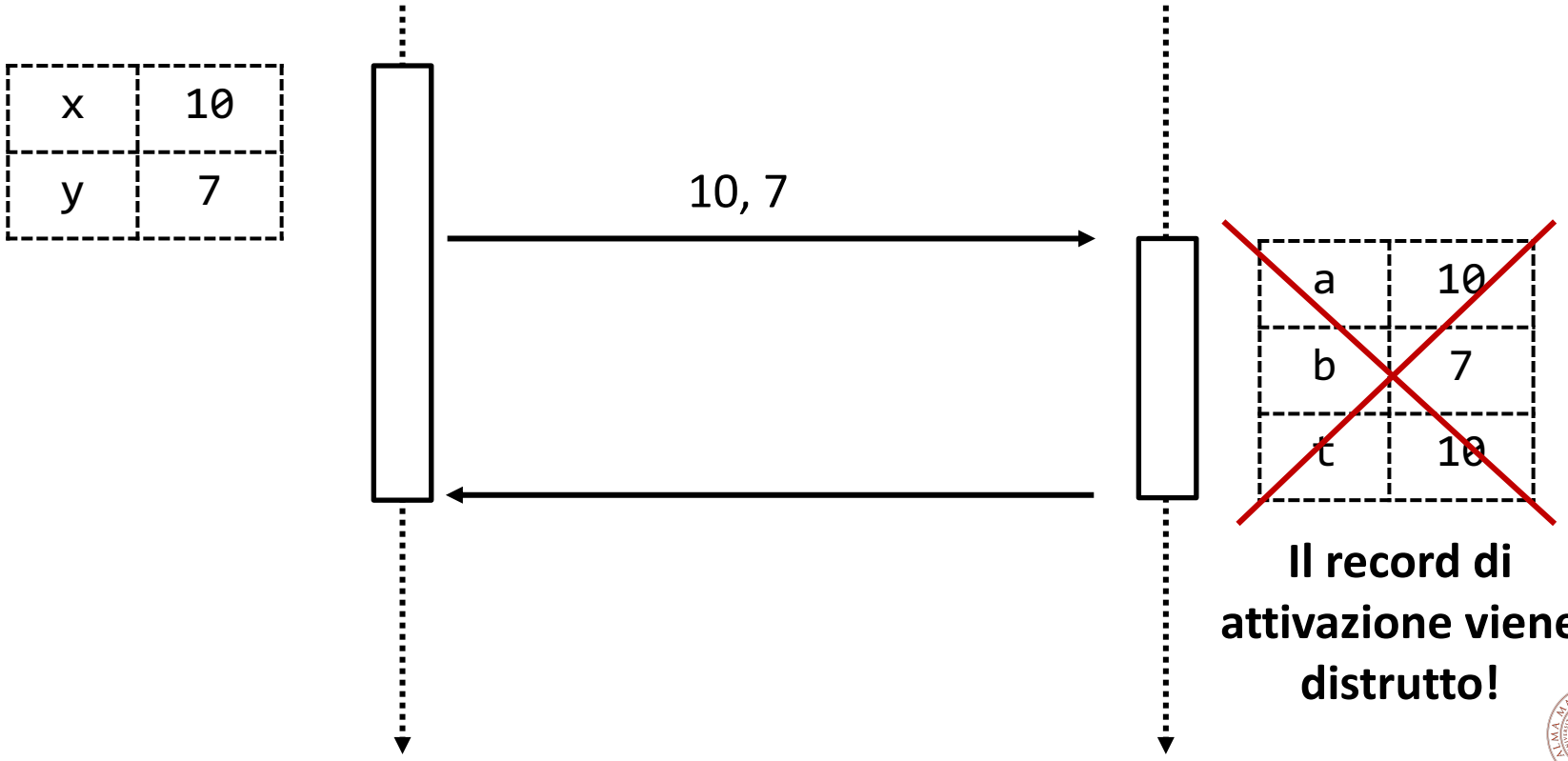


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedura Scambia: Esecuzione

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scambio di Variabili

Proviamo ad usare la nostra procedura:

```
void scambia(int a, int b) {
   int t = a; // salvo li valore di a
   a = b; // sovrascrivo a con b
   b = t; // scrivo il vecchio valore di a in b
}
int main() {
   int x = 10, y = 7;
   scambia(x, y);
   printf("x = %d, y = %d\n", x, y); // x= 10, y = 7
}
```

- Effettuare lo scambio è impossibile...
- ...Fintanto che i due ambienti sono separati

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scambio di Variabili

Come aggirare il problema?

Possibilità 1:

- Usare due variabili globali
- ...Ma facendo così possiamo scambiare solo quelle 😕

Possibilità 2:

- Fornire a alla funzione scambia...
- ....La possibiilitò di accedere a specifiche variabili...
- ...Nell'ambiente del chiamante

Si chiama passaggio per riferimento

Passaggio per Valore e per Riferimento

Due tipologie di passaggio di parametri

Passaggio per valore:

- Si passa il valore di una espressione
- È quello che abbiamo usato finora

Passaggio per riferimento:

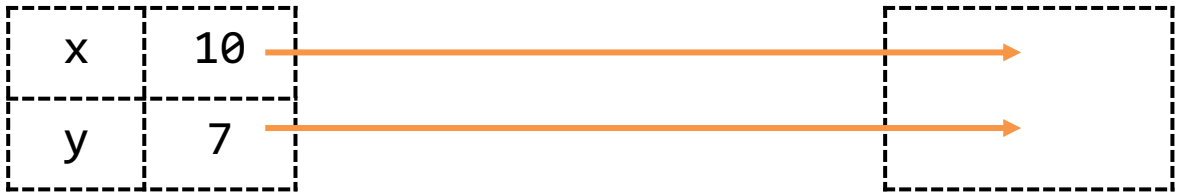
- Si passa l'indirizzo di una variabile
- In questo modo il servitore può modificarla!

Passaggio per Valore e per Riferimento

Nel passaggio per valore si trasferisce il contenuto della memoria

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```



record di attivazione del main

record di attivazione di "scambia"

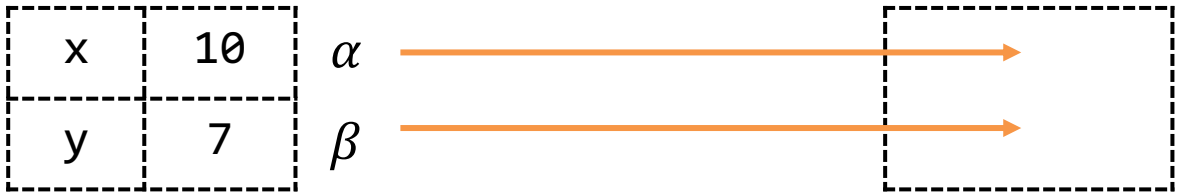
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Passaggio per Valore e per Riferimento

Nel passaggio per valore si trasferisce il contenuto della memoria

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```



indirizzi delle due variabili

record di attivazione di "scambia"

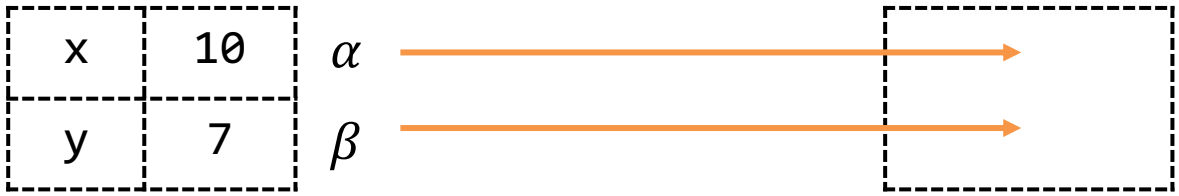
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Passaggio per Valore e per Riferimento

Nel passaggio per valore si trasferisce il contenuto della memoria

```
int main() {
    int x = 10, y = 7;
    scambia(x, y);
}
```

```
void scambia(int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```



- In questo caso a, diventa un alias per x (e b per y)
- Lo scambio avviene nell'ambiente del chiamante

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Passaggio per Riferimento in C

Il passaggio per riferimento:

- È disponibile nativamente in diversi linguaggi (e.g. C++, Java)
- ...Ma non in C!

Il C fornisce costrutti per manipolare direttamente indirizzi

- Tipi di dato che corrispondono ad indirizzi
- Variabili per memorizzare indirizzi
- Operatori che agiscono su indirizzi

È un approccio più complesso, ma più potente

Refenziamento e Dereferenziamento

Il C offre due operatori chiave per lavorare con gli indirizzi

Operatore unario di indirizzamento "&":

- Se applicato ad una variabile ne restituisce l'indirizzo
- Esempio:

```
&a // denota l'indirizzo di a
```

Operatore unario di dereferenziamento "*":

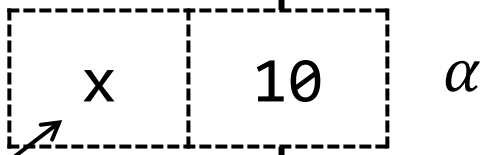
- Se applicato ad un indirizzo di una cella di memoria...
- ...Permette di trattarla come una variabile
- Esempio:

```
*(&a) // è equivalente a scrivere "a"
```

Refenziamento e Dereferenziamento

Livello di astrazione del linguaggio C

Livello di astrazione dell'elaboratore



α

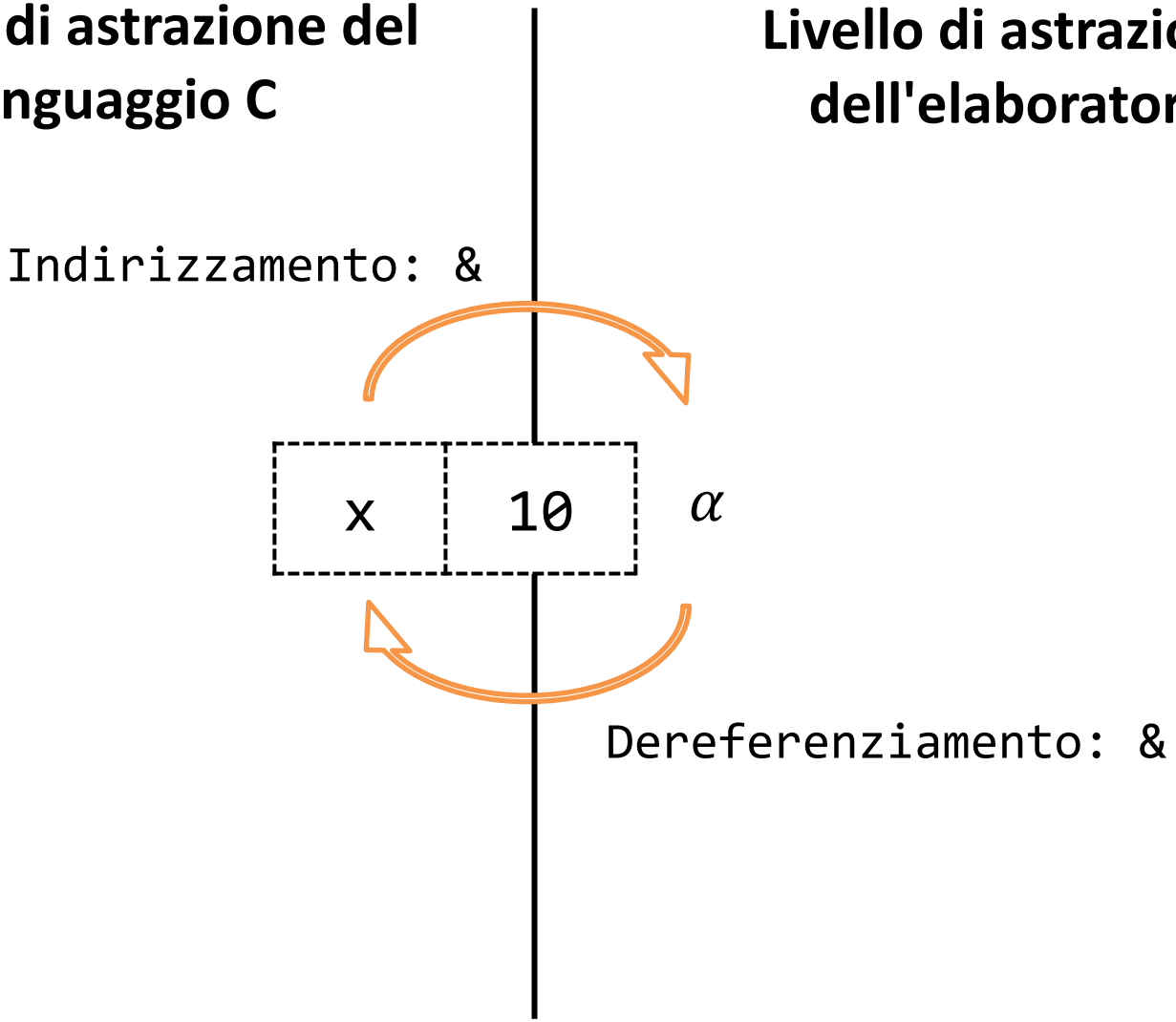
nome della variabile

indrizzo della cella di memoria

Refenziamento e Dereferenziamento

Livello di astrazione del linguaggio C

Livello di astrazione dell'elaboratore



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori

Il C permette di memorizzare indirizzi in speciali variabili

Si chiama puntatore una variabile che memorizza un indirizzo

Sintassi:

```
<tipo> *<nome>;
```

Semantica:

- <nome> è una variabile
- ...che contiene l'indirizzo di un dato
- ...Il cui tipo è <tipo>

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori

Qualche esempio:

```
int *a;
```

- La variabile (puntatore) "a" contiene l'indirizzo di un intero
- Si dice che "a" è di tipo "puntatore a int"

```
float *b;
```

- La variabile (puntatore) "b" contiene l'indirizzo di un float
- Si dice che "b" è di tipo "puntatore a float"

Puntatori

Uno mnemonico per ricordare:

- La sintassi per definire un puntatore
- La differenza tra indirizzamento e dereferenziamento

```
int (*a);
```

- Immaginate che ci siano delle parentesi intorno a "*a"
- Allora potete leggere che:
  - "*a" è un intero
  - "a" è un indirizzo
  - * trasforma un indirizzo in una variabile

Puntatori

Anche il C legge la definizione in questo modo!

Se incontriamo questa definizione

```
int *a, b, *c, d;
```

- "a" è un puntatore
- "b" è un normale intero
- "c" è un puntatore
- "d" è un normale intero

Puntatori

Un esempio di utilizzo

```c
int main() {
   int x = 10;
   int *px; // px è un puntatore ad int
   px = &x; // assegno l'indirizzo di x a px
   printf("x: %d\n", *px); // stampa "x: 10"
   *px += 1; // incremento la cella puntata
   printf("x: %d\n", x); // stampa "x: 11"
}
```

- L'operatore & ottiene un indirizzo a partire da una variabile
- Lo si può usare per riempire un puntatore

Puntatori

Un esempio di utilizzo

```c
int main() {
   int x = 10;
   int *px; // px è un puntatore ad int
   px = &x; // assegno l'indirizzo di x a px
   printf("x: %d\n", *px); // stampa "x: 10"
   *px += 1; // incremento la cella puntata
   printf("x: %d\n", x); // stampa "x: 11"
}
```

- L'operatore * tratta una cella puntata come una variabile
- Possiamo stamparla, usarla per calcoli, modificarla...

Puntatori e Passaggio per Riferimento

I puntatori possono emulare il passaggio per riferimento:

Usiamo dei puntatori come parametri formali:

```
void scambia(int *a, int *b) { // Puntatori
   ...
}
```

In fase di invocazione, passiamo degli indirizzi:

```
int x = 10, y = 5;
scambia(&x, &y); // Usiamo l'operatore &
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

La procedura deve essere modificata:

- Con le variabili normali avevamo:

```
void scambia(int a, int b) {
   int t;
   t = a, a = b, b = t;
}
```

- Con i puntatori:

```
void scambia(int *a, int *b) {
   int t;
   t = *a, *a = *b, *b = t;
}
```

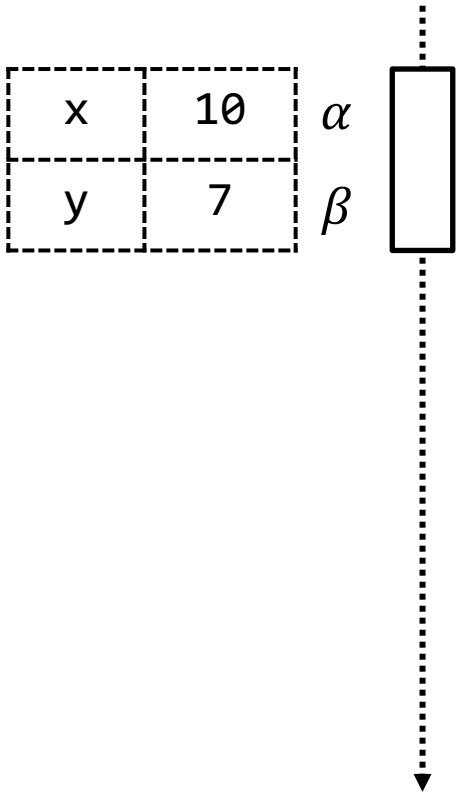
Ricordate: se "a" un puntatore, "*a" è la variabile puntata

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

```
int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
}
```

```
void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

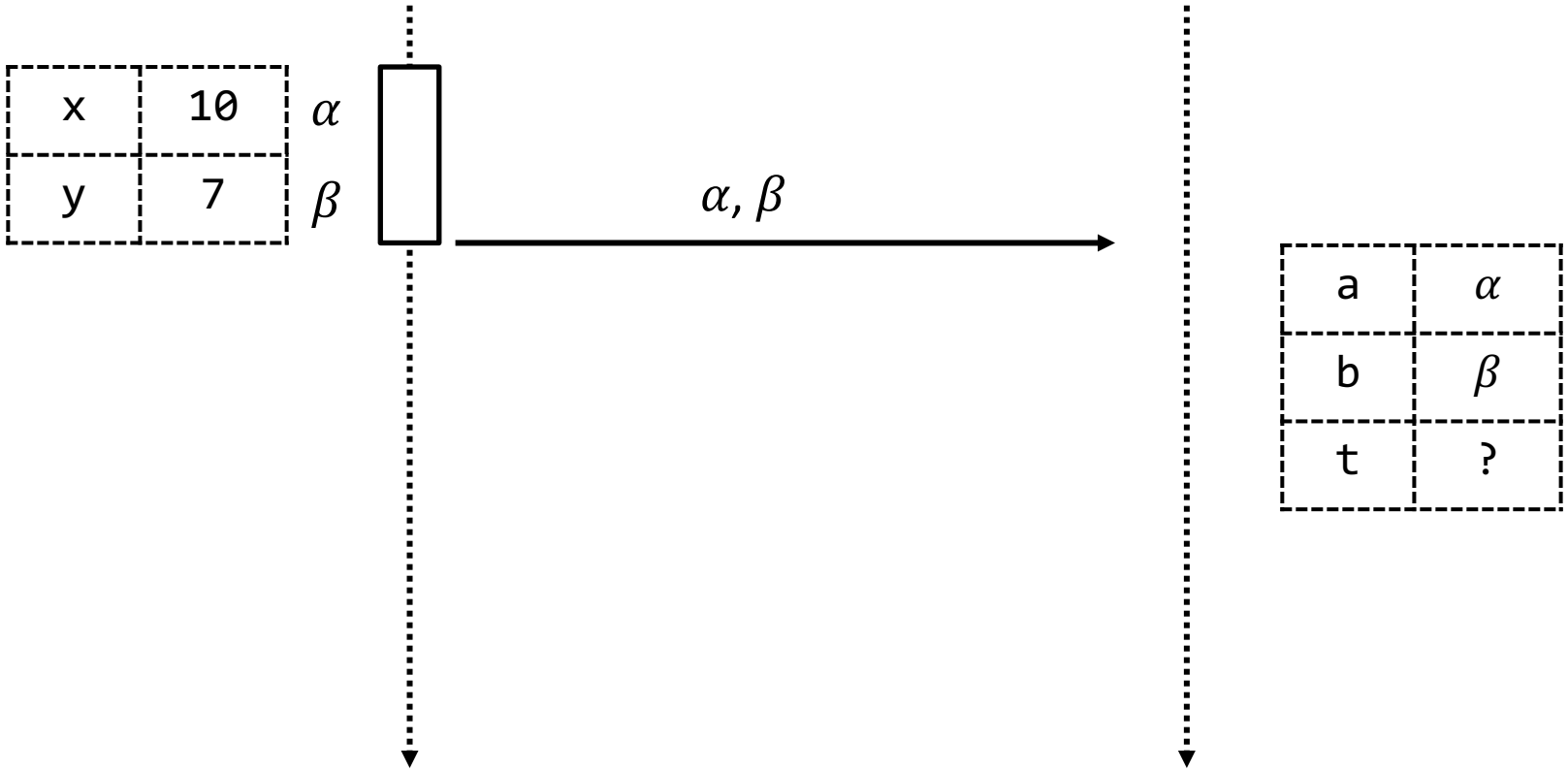


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

```
int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
}
```

```
void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

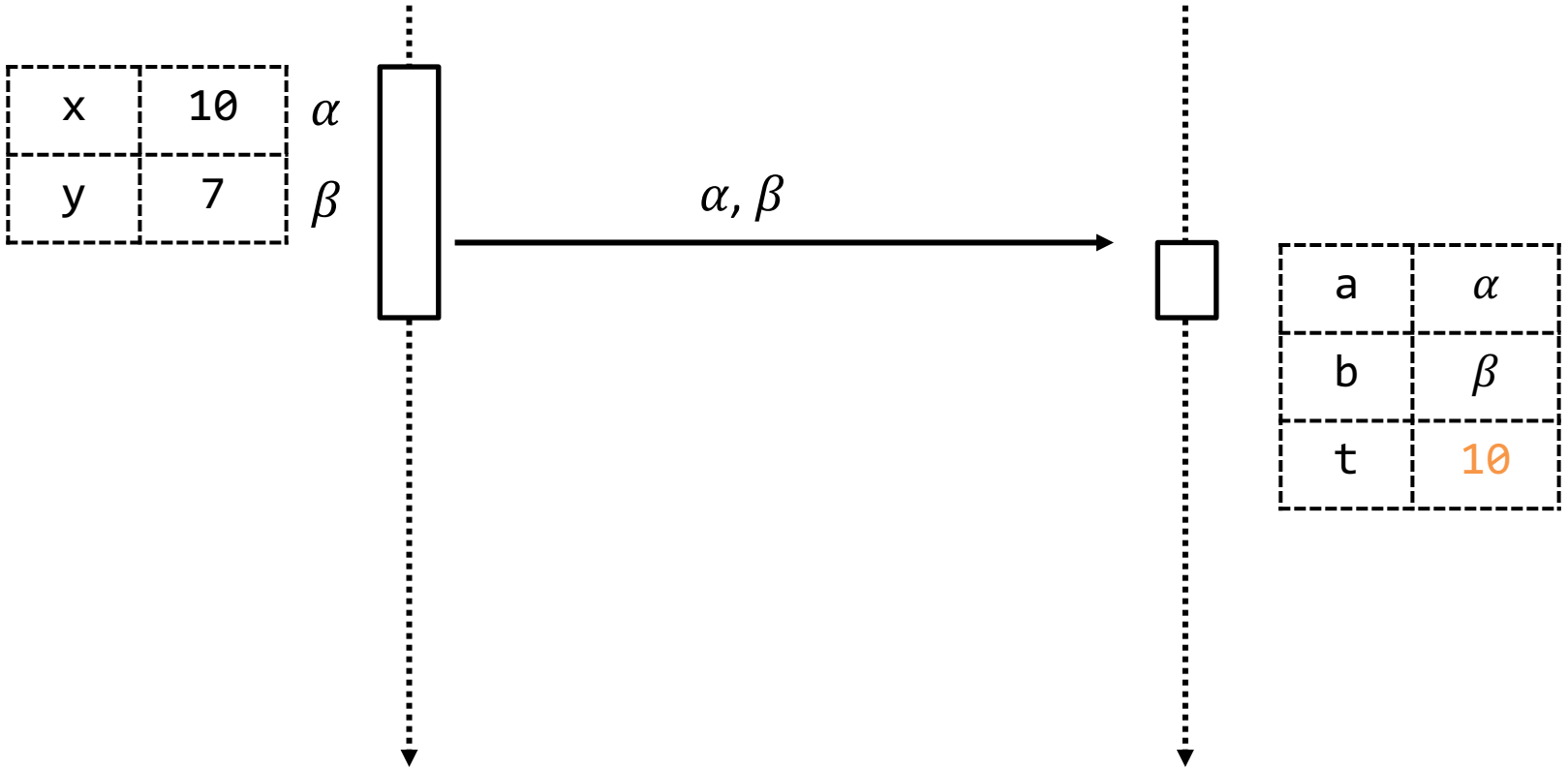


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

```
int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
}
```

```
void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

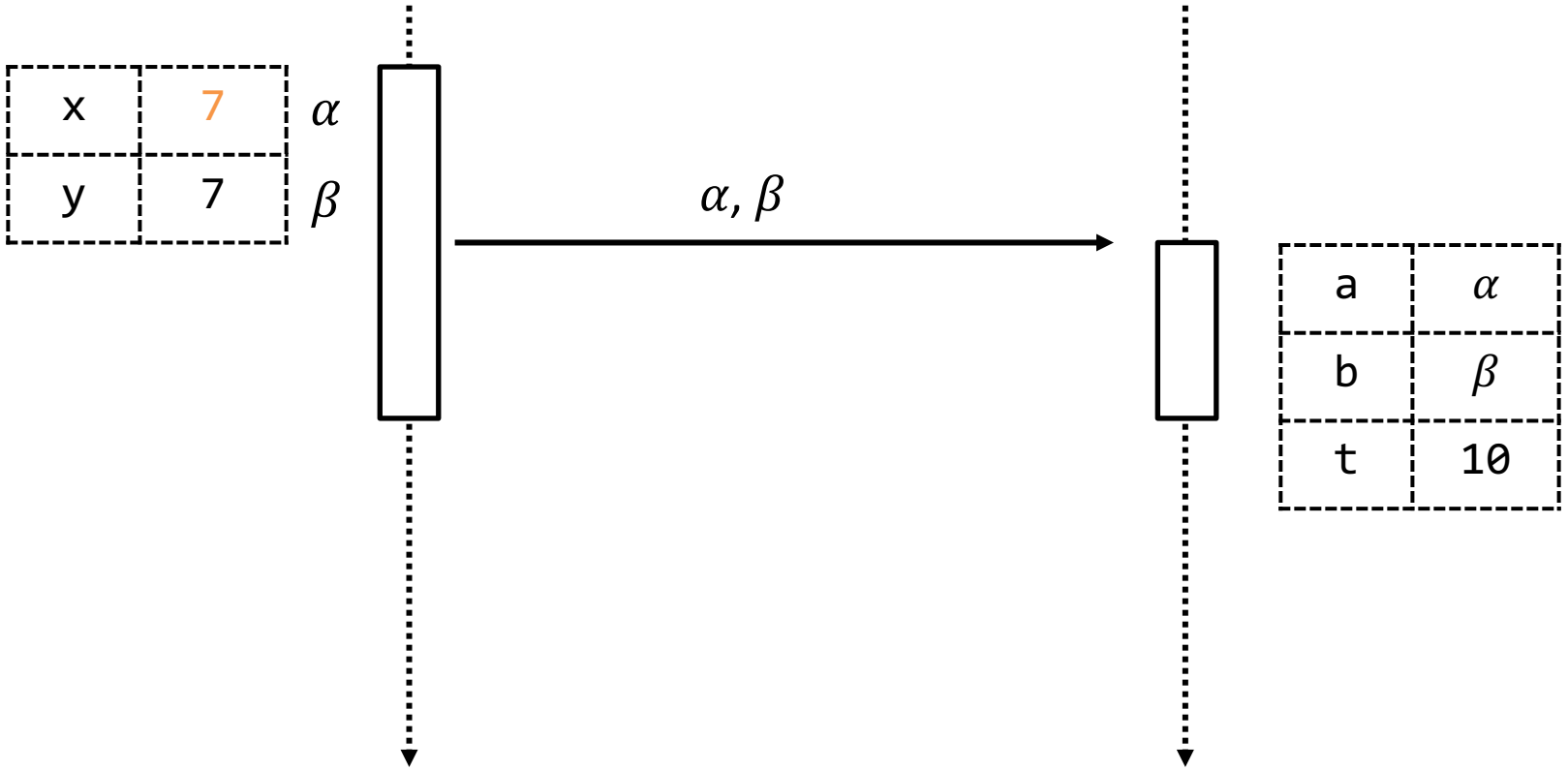


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

```c
int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
}
```

```c
void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

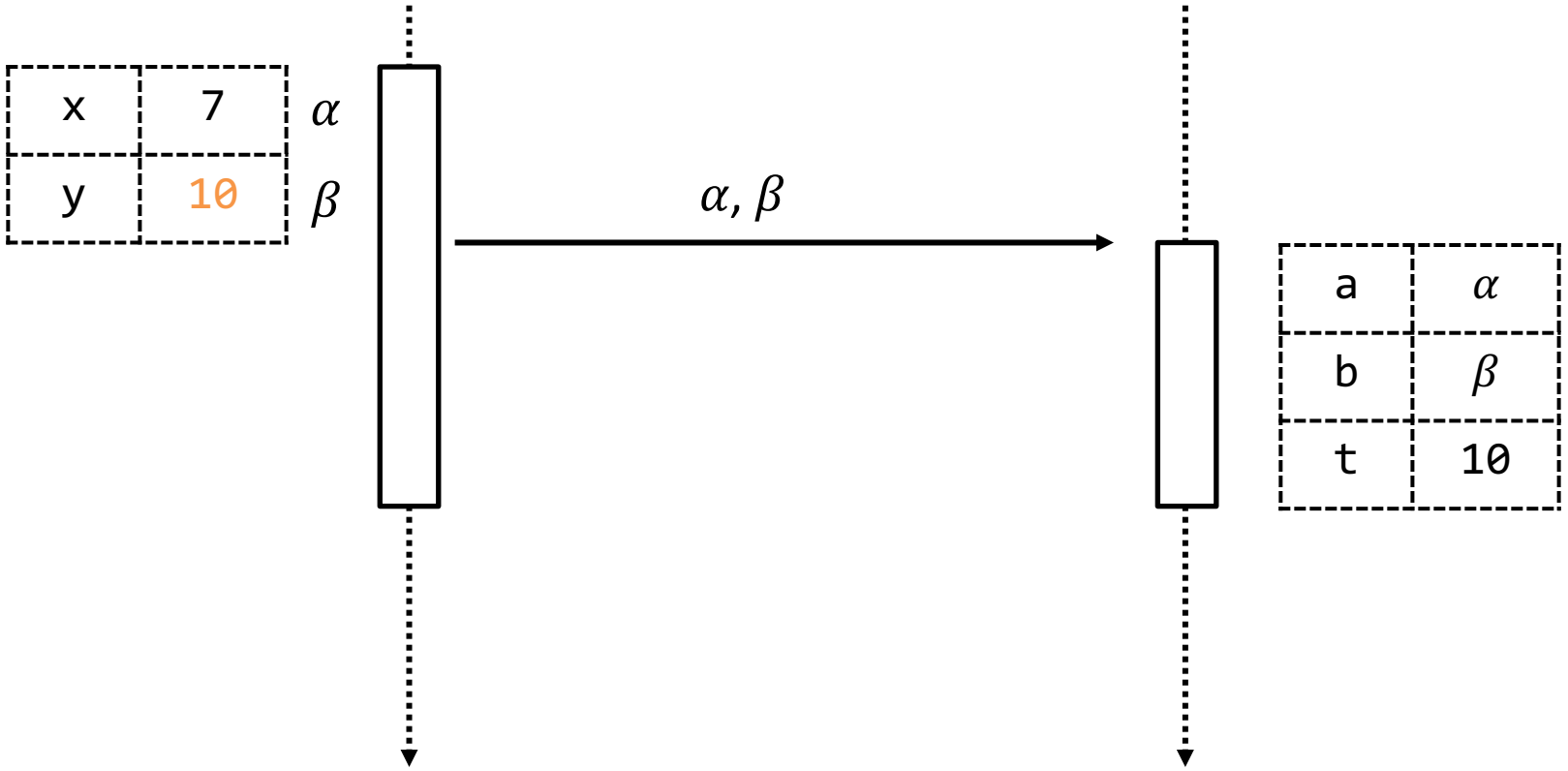


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

```c
int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
}
```

```c
void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

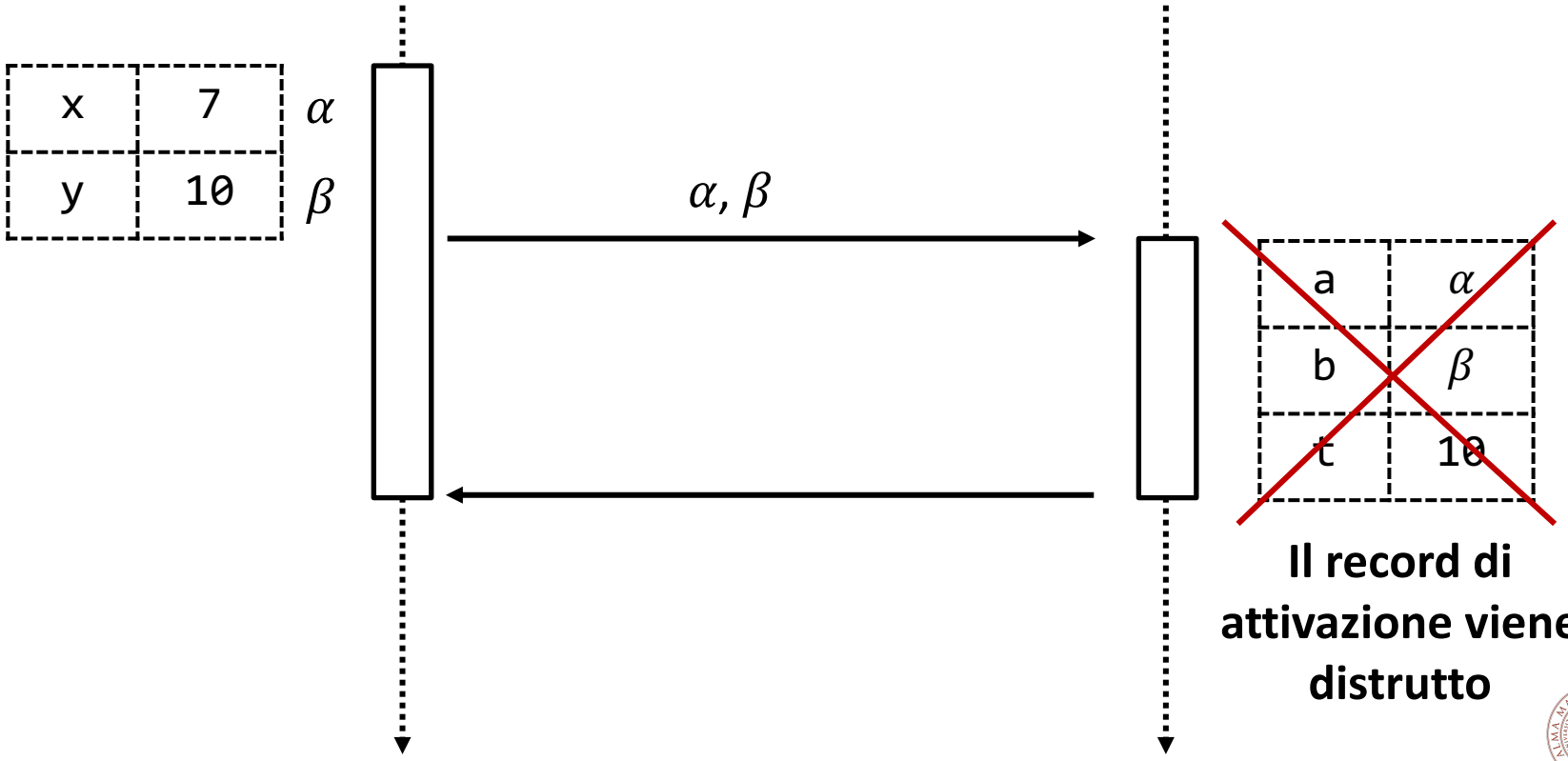


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

```
int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
}
```

```
void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```



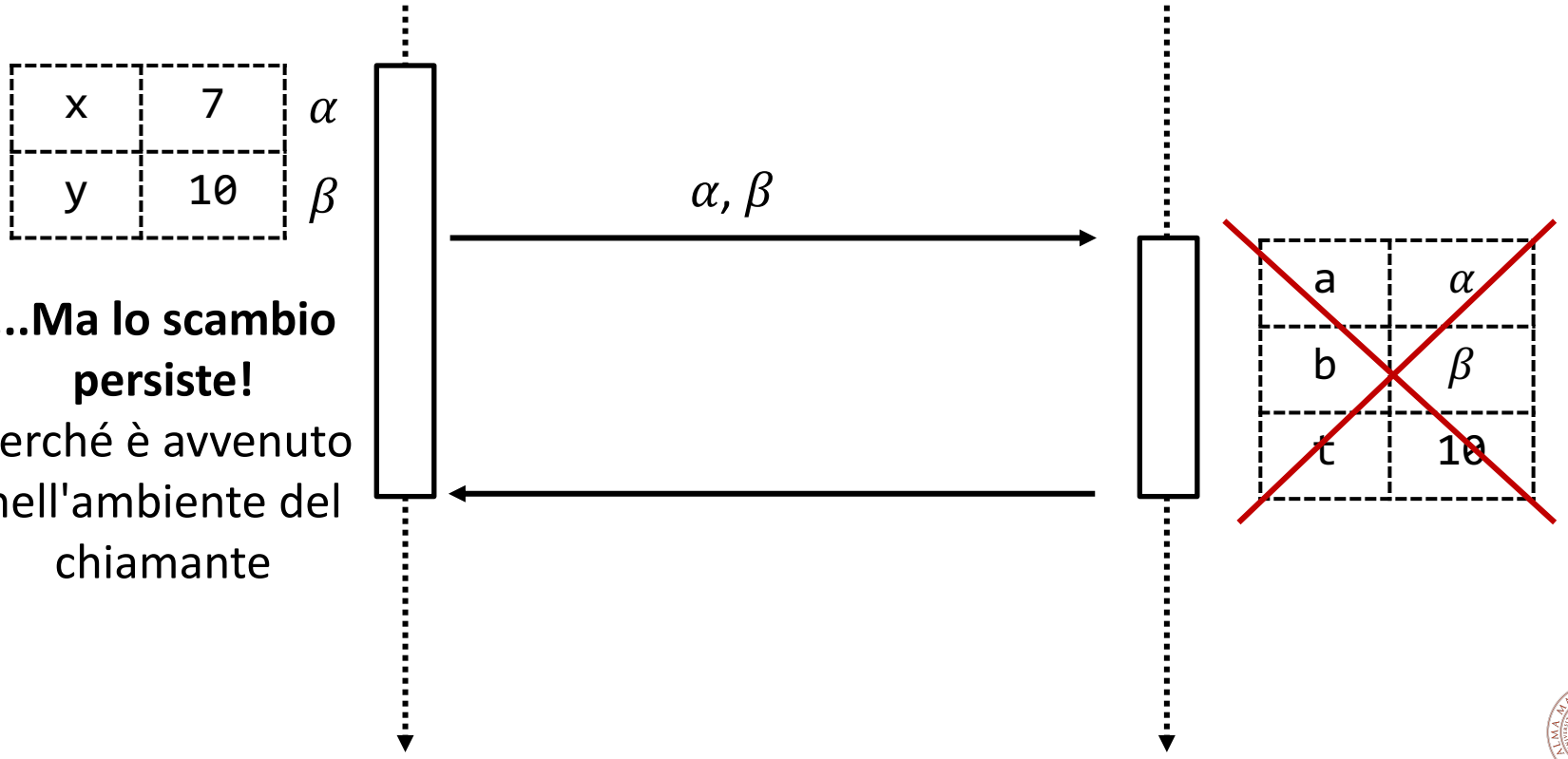
Il record di attivazione viene distrutto

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori e Passaggio per Riferimento

```c
int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
}
```

```c
void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```



...Ma lo scambio persiste!

Perché è avvenuto nell'ambiente del chiamante

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Procedura "scambia": Esempio Completo

```c
#include <stdio.h>

void scambia(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int main() {
    int x = 10, y = 7;
    scambia(&x, &y);
    printf("x: %d, y: %d\n", x, y);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Puntatori ed Ambiti

Avevamo detto che:

- Funzioni diverse hanno ambiti diversi
- Variabili in un ambito non sono accessibili da altri ambiti
- ...Ma con i puntatori pare che invece sia possibile!

Accedere ad un altro ambito in C non è fisicamente impossbile

È solo una questione di disponibilità d indirizzi

- Una funzione non ha gli indirizzi di variabili fuori dal suo ambito
- ...Ma se glieli forniamo, allora li può usare!

Puntatori Non-Inizializzati

Una variabile non inzializzata ha un valore non controllabile

Vale anche con i puntatori... ma con consueguenze peggiori!

```
int main() {
   int *p;
   *p = 10; // scriviamo 10 in una locazione ignota
}
```

- Se siamo fortunati, otteniamo un errore a run-time (segfault)
- Se siamo sfortunati, modifichiamo un'altra variabile

Correggere questi errori può essere molto difficile

Costante NULL

I puntatori non-inizializzati vanno evitati a tutti i costi

- Possiamo inizializzarli con la costante NULL
- È definita in diferse librerie (tra cui stdio.h)

```
int main() {
    int *p = NULL;
    *p = 10; // Causa un errore a run-time
}
```

Un accesso ad indirizzo NULL:

- Produce sempre un segfault a run-time
- È bene! Sappiamo che c'è un problema e dove si trova

Puntatori: da Ricordare

I puntatori sono un costrutto molto potente

- Permettono di realizzare procedure
- Permettono di "restituire" valori multipli
- Permettono lo scambio efficiente di collezioni di dati
- Rendono più efficienti alcune implementazioni
- Rendono possibli operazioni di basso livello (e.g. per Sistemi Operativi o Device Driver)

Vanno però usati con molta attenzione

Puntatori: da Ricordare

Esempio: Procedura di Incremento

```c
#include <stdio.h>

void inc(int *x) {
    (*x)++;
}

int main() {
    int x = 10;
    inc(&x);
    printf("x: %d\n", x);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: "Restituire" Valori Multipli

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

void sin_cos(float x, float *s, float *c) {
    *s = sin(x); // riempio *s
    *c = cos(x); // riempio *c
}

int main() {
    float x = 3.14159, ys, yc;
    sin_cos(x, &ys, &yc);
    printf("sin: %f, cos: %f\n", ys, yc);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Modello Preda-Predatore

Consideriamo il seguente modello preda-predatore

Descrive l'evoluzine del numero dell prede (H) e predatori (P):

$$H^{(t)} = \mathrm{r}\left(1 - \frac{H^{(t-1)}}{N}\right) - sH^{(t-1)}\,P^{(t-1)}$$

$$P^{(t)} = \mathrm{u}\,P^{(t-1)} + v\,(sH^{(t-1)}\,P^{(t-1)})$$

- *r* è il tasso di riproduzione naturale delle prede
- N è la quantità di prede sostenibili con le risorse naturali
- *s* è l'efficenza di caccia dei predatori
- *u* è il calo fisiologico dei predatori
- *v* è il "bonus riproduttivo" della predazione

È simile ai modelli già visti, ma con due variabili di stato

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Modello Preda-Predatore

```c
#include <stdio.h>
void f(float *pH, float *pP) {
    float r = 2, N = 1000, s = 0.01, u = 0.5, v = 0.3;
    float H = *pH, P = *pP;
    *nH = r * H * (1 - H/N) - s * H * P;
    *nP = u*P + v * s * H * P;
}
int main() {
    int n = 10;
    float H = 500, P = 30;
    printf("H(0), P(0): \t%f, %f\n", H, P);
    for(int t = 1; t <= n; ++t) {
        f(&H, &P);
        printf("H(%d), P(%d): \t%f, %f\n", t, t, H, P);
    }
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Input da Terminale

Funzione scanf

La libreria stdio consente di fare anche input da terminale

- Ci sono diverse funzioni per farlo
- La più versatile è probabilmente "scanf"

La funzione scanf:

- Legge caratteri dal file speciale stdio (di norma, la tastiera)
- Può convertire i caratteri in altri tipi (e.g. int, float...)
- Può richiedere input con sintassi specifica (e.g. gg/mm/aa)

Funzione scanf

Utilizzo della funzione scanf

- La funzione scanf ha una interfaccia piuttosto complessa
- Utilizza funzionalità che non vedremo nel corso

Utilizzarla è però relativamente semplice:

```
scanf(<str. di formato>, <indirizzo1>, <indirizzo2>, ...)
```

- La stringa di formato specifica cosa leggere da input...
- ...Ed in che formato aspettarselo
- Si utilizzano le stesse specifiche di formato della printf
- Seguono gli indirizzi delle variabili da riempire

Funzione scanf

Vediamo qualche esempio

Lettura di un intero:

```
int a;
scanf("%d", &a);
```

- La scanf tenta di leggere dal buffer di input
- Se lo trova vuoto, interrompe temporaneamente il programma
- ...In attesa che arrivi input da terminale
- Lo possiamo fornire scrivendo qualcosa e premendo "invio"

Funzione scanf

Vediamo qualche esempio

Lettura di due interi, separati da spazi/tab/fine linea:

```
int a, b;
scanf("%d%d", &a, &b);
```

- La scanf non termina finché non ha effettuato tutte le letture
- ...Con due eccezioni:
  - Il file viene chiuso (difficile, nel caso del terminale)
  - Una conversione fallisce (e.g. input 1.2 con %d)
- La scanf restituisce il numero di conversioni effettuate

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Funzione scanf

Vediamo qualche esempio

Lettura di un intero ed un float:

```
int a;
float b;
scanf("%d%f", &a, &b);
```

- Se entrambe le letture hanno successo restituisce 2
- Se la seconda fallisce restituisce 1
- Se entrambe falliscono (e.g. input "A") restituisce 0

Funzione scanf

Vediamo qualche esempio

Lettura di un orari (due interi separati da ";"):

```
int hh, mm;
scanf("%d:%d", &hh, &mm);
```

- Se la stringa di formato contiene caratteri non di formato
- ...Questi devono essere forniti, o la lettura fallisce
- E.g. se scriviamo "12:15" è ok
- E.g. se scriviamo "12 15" la seconda conversione fallisce

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Attenzione a leggere caratteri

Perché i separatori (spazio/tab/fine linea) sono essi stessi caratteri

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

- Se scriviamo "x y z" (e battiamo invio), stampa "x y"!
- Legge i caratteri 'x', ' ' (spazio!) e 'y'

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Attenzione a leggere caratteri

Perché i separatori (spazio/tab/fine linea) sono essi stessi caratteri

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

- Se leggiamo 'x', ' ', e 'y', vuol dire che sono avanzati dei caratteri
- In questo caso: ' ' (spazio), 'z' e '\n' (il fine linea!)

Lettura di Caratteri

Consideriamo questo programma:

```c
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

Avviamo il programma (e.g. "test"):

```
C:\> test
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

Il buffer di input è vuoto: l'esecuzione si interrompe

buffer

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

Digitiamo "x y z" e battiamo "invio"

```
x y z
```

buffer

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

I caratteri vengono trasferiti sul buffer di input

```
x y z
```

buffer

'x'	' '	'y'	' '	'z'	'\n'

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

...Incluso il fine linea (i.e. "invio" stesso!)

```
x y z
```

buffer

'x'	' '	'y'	' '	'z'	'\n'

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

La scanf legge i primi tre caratteri: a='x', b=' ', c='y'

buffer

' '	'z'	'\n'			

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

La printf stampa le tre variabili: a='x', b=' ', c='y'

x y\n ← **a capo**

buffer

' '	'z'	'\n'			

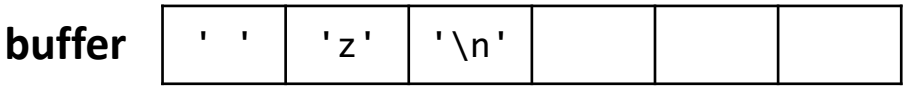
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

Il buffer è pieno: la scanf non si ferma!



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

...E legge tre caratteri: a=' ', b='z', c='\n'

buffer

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

La printf stampa: a=' ', b='z', c='\n'

```
z\n\n
```

buffer

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lettura di Caratteri

Consideriamo questo programma:

```
#include <stdio.h>
int main() {
    char a, b, c;
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
    scanf("%c%c%c", &a, &b, &c);
    printf("%c%c%c\n", a, b, c);
}
```

Output nel programma (per "x y z[invio]" come input):

```
x y\n
 z\n
\n
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Calcolo di un Fattoriale

```c
#include <stdio.h>

int fattoriale(int n) {
    int f = 1;
    for(int i = 2; i <= n; ++i) f *= i;
    return f;
}

int main() {
    int n;
    printf("Inserire un numero: ");
    scanf("%d", &n);
    printf("Fattoriale di %d: %d", n, fattoriale(n));
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Esercizio: Modello SIR

Esercizio: Modello SIR

Consideriamo un modello epidemiologico SIR

Descrive l'andamento di tre "compartimenti" di popolazione:

- S = susceptibles (suscettibili)
- I = infected (infetti)
- R = recovered (guariti – si considerano immunizzati)

I modelli SIR

- Sono tra i modelli epidemilogici più semplici
- Ciononostante, possono essere utilizzati per ottenere previsioni

Prima di inziare, però...

Esercizio: Modello SIR

Non improvvisatevi epidemiologi!

- Usare uno strumento (matematico) richiede un esperto
- ...Un paio di slide non bastano di certo

Esercizio: Modello SIR

Il modello SIR più semplice

...È definito da una equazione differenziale in tre incognite

$$\frac{dS}{dt} = -\beta \frac{1}{N} SI$$

$$\frac{dI}{dt} = \beta \frac{1}{N} SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

- Nel tempo i suscettibili calano
- La velocità di calo è proporzionale al prodotto S*I
- N è la popolazione totale
- β/N è la costante di proporzionalità

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio: Modello SIR

Il modello SIR più semplice

...È definito da una equazione differenziale in tre incognite

$$\frac{dS}{dt} = -\beta \frac{1}{N} SI$$

$$\frac{dI}{dt} = \beta \frac{1}{N} SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

- Nel tempo i guariti crescono
- La velocità di crescita è proporzionale a I (gli infetti)
- La costante di proporzionalità è γ

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio: Modello SIR

Il modello SIR più semplice

...È definito da una equazione differenziale in tre incognite

$$\frac{dS}{dt} = -\beta \frac{1}{N} SI$$

$$\frac{dI}{dt} = \beta \frac{1}{N} SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

- I valori di S, I e R definiscono lo stato del sistema
- L'andamento del modello dipende dal fattore $R_0 = \beta/\gamma$
- Se $R_0 > 1$, allora si verifica una epidemia

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio: Modello SIR

Come risolvere l'equazione differenziale?

Useremo il metodo di Eulero

$$\frac{dS}{dt} = -\beta \frac{1}{N} SI$$

$$\frac{dI}{dt} = \beta \frac{1}{N} SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

- Invece che trattare le derivate in modo diretto...

Esercizio: Modello SIR

Come risolvere l'equazione differenziale?

Useremo il metodo di Eulero

$$\frac{S^{(t)} - S^{(t-1)}}{h} = -\beta \frac{1}{N} S^{(t-1)} I^{(t-1)}$$

$$\frac{I^{(t)} - I^{(t-1)}}{h} = \beta \frac{1}{N} S^{(t-1)} I^{(t-1)} - \gamma I^{(t-1)}$$

$$\frac{R^{(t)} - R^{(t-1)}}{h} = \gamma I^{(t-1)}$$

- Invece che trattare le derivate in modo diretto...
- ...Le approssimiamo come rapporti incrementali
- *h* è la differenza tra il tempo "t" ed il tempo "t-1"

Esercizio: Modello SIR

Come risolvere l'equazione differenziale?

Useremo il metodo di Eulero

$$S^{(t)} = S^{(t-1)} - h\left(\beta \frac{1}{N} S^{(t-1)} I^{(t-1)}\right)$$

$$I^{(t)} = I^{(t-1)} + h\left(\beta \frac{1}{N} S^{(t-1)} I^{(t-1)} - \gamma I^{(t-1)}\right)$$

$$R^{(t)} = R^{(t-1)} + h\left(\gamma I^{(t-1)}\right)$$

- Separando le variabili a "t" e "t-1"...
- ...Otteniamo una formulazione iterativa
- Come quelle che sappiamo già trattare!

Modello SIR

```c
#include <stdio.h>
void sir(float N, float b, float g, float h,
         float *pS, float *pI, float *pR){
    float S = *pS, I = *pI, R = *pR;
    *pS = S - h * b *  S* I / N;
    *pI = I + h * b * S * I / N -g * I;
    *pR = R + h * I;
}
int main() {
    int n = 10, i;
    float N  = 300e3, b = 1.2, g = 1, h = 1;
    float I = 10, S = N-I, R = 0;
    for(i = 0; i < n; ++i) {
        sir(N, b, g, h, &S, &I, &R);
        printf("S: %.0f, I: %.0f, R: %.0f\n", S, I, R);
    }
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Andrea Acquaviva < andrea.acquaviva@unibo.it>
Michele Lombardi <michele.lombardi2@unibo.it
Andrea Borghesi <andrea.borghesi3@unibo.it>
Giuseppe Tagliavini <giuseppe.tagliavini@unibo.it>
Allegra De Filippo < allegra.defilippo@unibo.it>