# Funzioni in C

# Modello client/server

Server:

- Un qualunque ente computazionale capace di nascondere la propria organizzazione interna
- **presentando ai client una precisa *interfaccia*** per lo scambio di informazioni

Client:

- qualunque ente in grado di **invocare uno o più server** per svolgere il proprio compito

# Interfaccia di una funzione

- L'**interfaccia** (o prototipo) di una funzione comprende
  - *nome della funzione*
  - *lista dei parametri*
  - *tipo del valore da essa denotato*
- *Esplicita il contratto di servizio* fra client e server
- Client e server comunicano quindi mediante
  - i *parametri* trasmessi dal cliente al server all'atto della chiamata
  - il *valore restituito* dal server al client

# Funzioni: esempio C

```
int max (int x, int y )
{
    if (x>y) return x;
    else return y;
}
```

- Il simbolo **max** denota il nome della funzione
- Le variabili intere **x** e **y** sono i parametri della funzione
- Il valore restituito è un intero **int**
- Se la funzione non restituisce niente il tipo e' indicato come **void**

# Comunicazione client/server

- Il cliente passa informazioni al servitore mediante una serie di ***parametri***

**Parametri formali**:

- sono specificati nella *dichiarazione* del server
- esplicitano *il contratto* fra server e client
- indicano *che cosa il server si aspetta dal client*

**Parametri attuali**:

- sono *trasmessi dal cliente* all'atto della chiamata
- devono corrispondere ai parametri formali in *numero*, *posizione* e *tipo*

# Funzioni: esempio parametri

Parametri formali

```
int max (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

SERVER

**Definizione** della funzione

```
main() {
     int z = 8;
     int m;

     m = max (z, 4);
}
```

CLIENT

**Chiamata** della funzione

Parametri attuali

# Comunicazione cliente/servitore

Legame tra parametri attuali e parametri formali:

- effettuato *al momento della chiamata*, in modo dinamico

Tale legame:

- vale solo per l'invocazione corrente
- vale solo per la durata della funzione

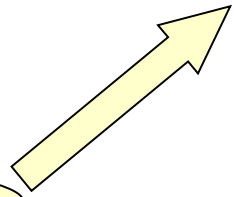
# Esempio

Parametri formali

```
int max (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

```
main() {
     int z = 8;
     int m1, m2;

     m1 = max (z, 4);
     m2 = max (5, z);
}
```



All'atto di questa chiamata della funzione, si effettua un legame tra:

**x** e **z**

**y** e **4**

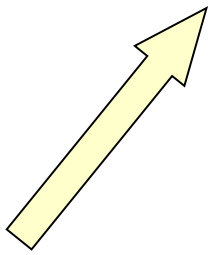
# Esempio

Parametri formali

```
int max (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

```
main() {
     int z = 8;
     int m1, m2;

     m1 = max (z, 4);
     m2 = max (5, z);
}
```



All'atto di questa chiamata della funzione, si effettua un legame tra:

**x** e **5**

**y** e **z**

# Information hiding

- La *struttura interna* (corpo) di una funzione è *completamente inaccessibile dall'esterno*
- Così facendo si garantisce *protezione dell'informazione* (*information hiding*)
- Una funzione è accessibile **solo** attraverso la sua interfaccia
- NB: potrei modificare l'implementazione interna mantendo uguale l'interfaccia!

# Riassumendo

## All’atto dell’invocazione di una funzione:

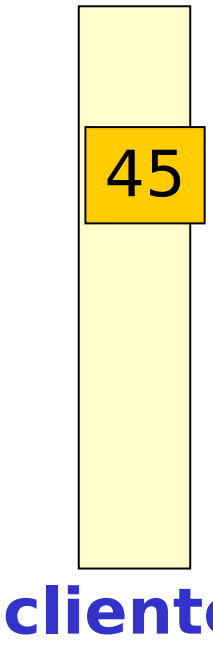
- si crea una ***nuova attivazione (istanza)*** *del server*
- si alloca la ***memoria per i parametri*** (e le eventuali variabili locali)
- si trasferiscono i parametri al server
- si trasferisce il controllo al server
- si esegue il codice della funzione
- Al termine del blocco associato alla funzione o in corrispondenza di una istruzione return, il controllo torna al client.

# Passaggio dei parametri

- In generale, un parametro può essere trasferito dal cliente al servitore:
  - per **valore** o **copia** (*by value*)
    - si trasferisce ***il valore*** del parametro attuale
  - per **riferimento** (*by reference*)
    - si trasferisce ***un riferimento*** al parametro attuale

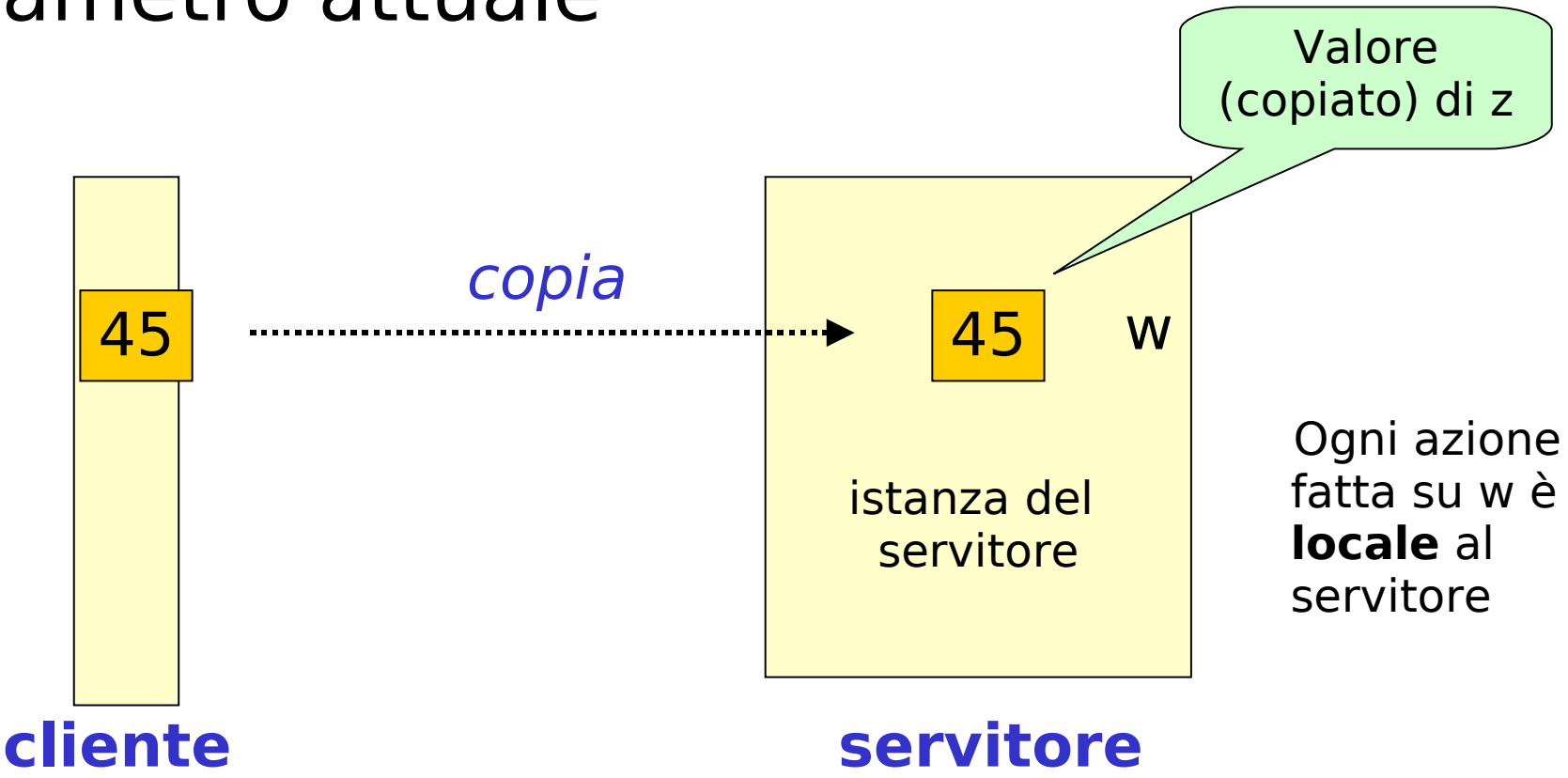
# Passaggio per valore

- Si trasferisce **una copia del valore** del parametro attuale



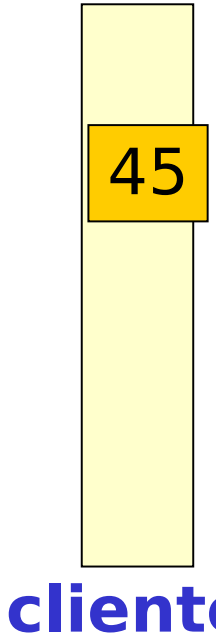
# Passaggio per valore

- Si trasferisce **una copia del valore** del parametro attuale



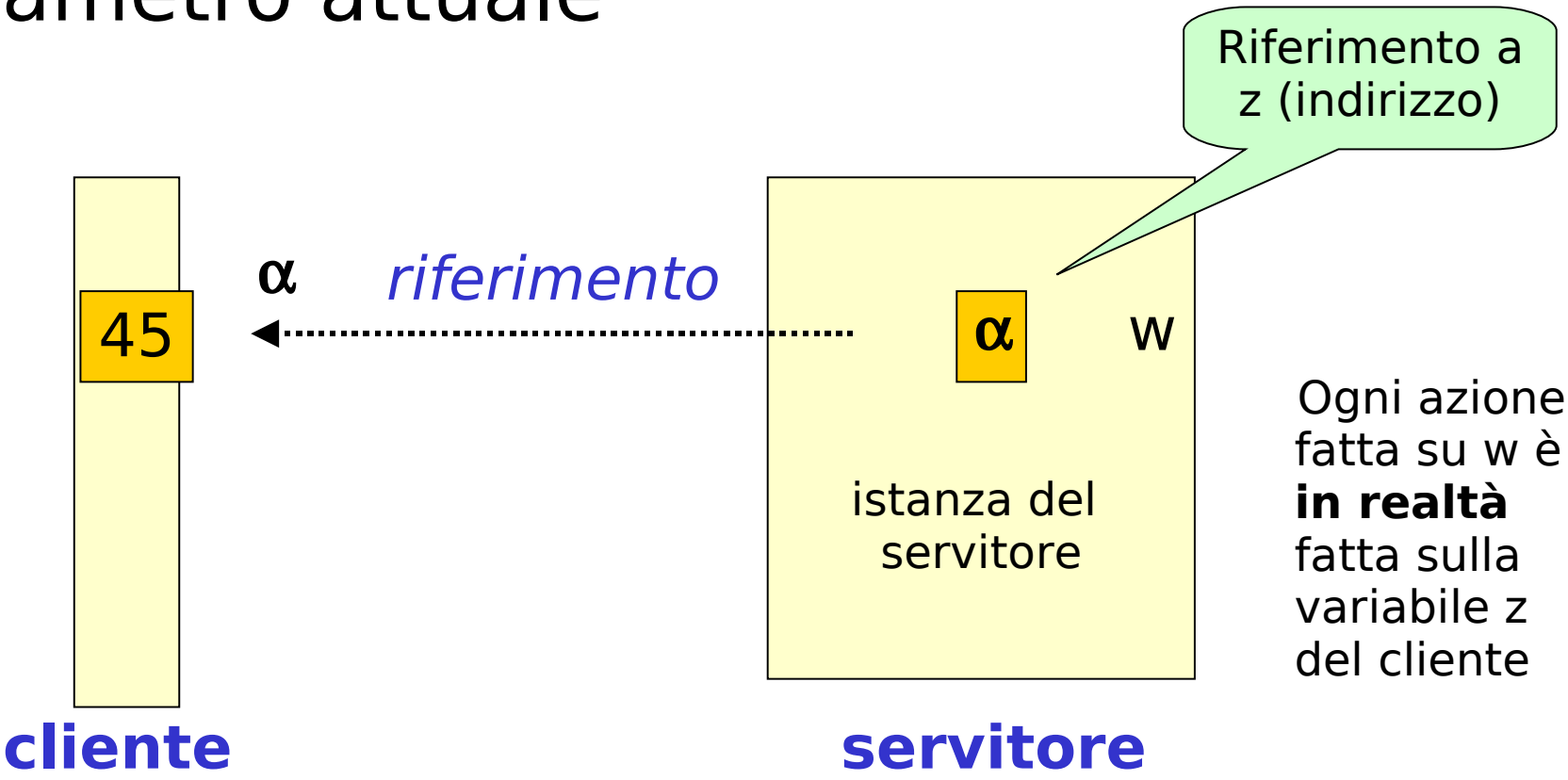
# Passaggio per riferimento

- Si trasferisce **un riferimento** al parametro attuale



# Passaggio per riferimento

- Si trasferisce **un riferimento** al parametro attuale




Ogni azione fatta su w è **in realtà** fatta sulla variabile z del cliente

# Passaggio dei parametri in C

- In C, i parametri sono trasferiti sempre e solo per valore (*by value*)
  - si trasferisce ***una copia*** del parametro attuale, *non l'originale*
  - tale copia è *strettamente* ***privata e locale*** *a quel server*
  - il server potrebbe quindi alterare il valore ricevuto, *senza che ciò abbia alcun impatto sul client*

# Esempio

## *Perché il passaggio per valore non basta?*

*Problema:*

scrivere una procedura che *scambi i valori di due variabili intere*

*Specifica:*

Dette A e B le due variabili, ci si può appoggiare a una *variabile ausiliaria T*, e svolgere lo scambio in *tre fasi*

*Frammento di codice:*

```
int a,b,t;
…
t = a; a = b; b = t;
…
```

# Esempio

- Supponendo di utilizzare, senza preoccuparsi, il passaggio per valore usato finora, la codifica potrebbe essere espressa come segue:

```
void scambia(int a, int b) {
  int t;
  t = a; a = b; b = t;
  return; /* può essere omessa */
}
```

# Esempio

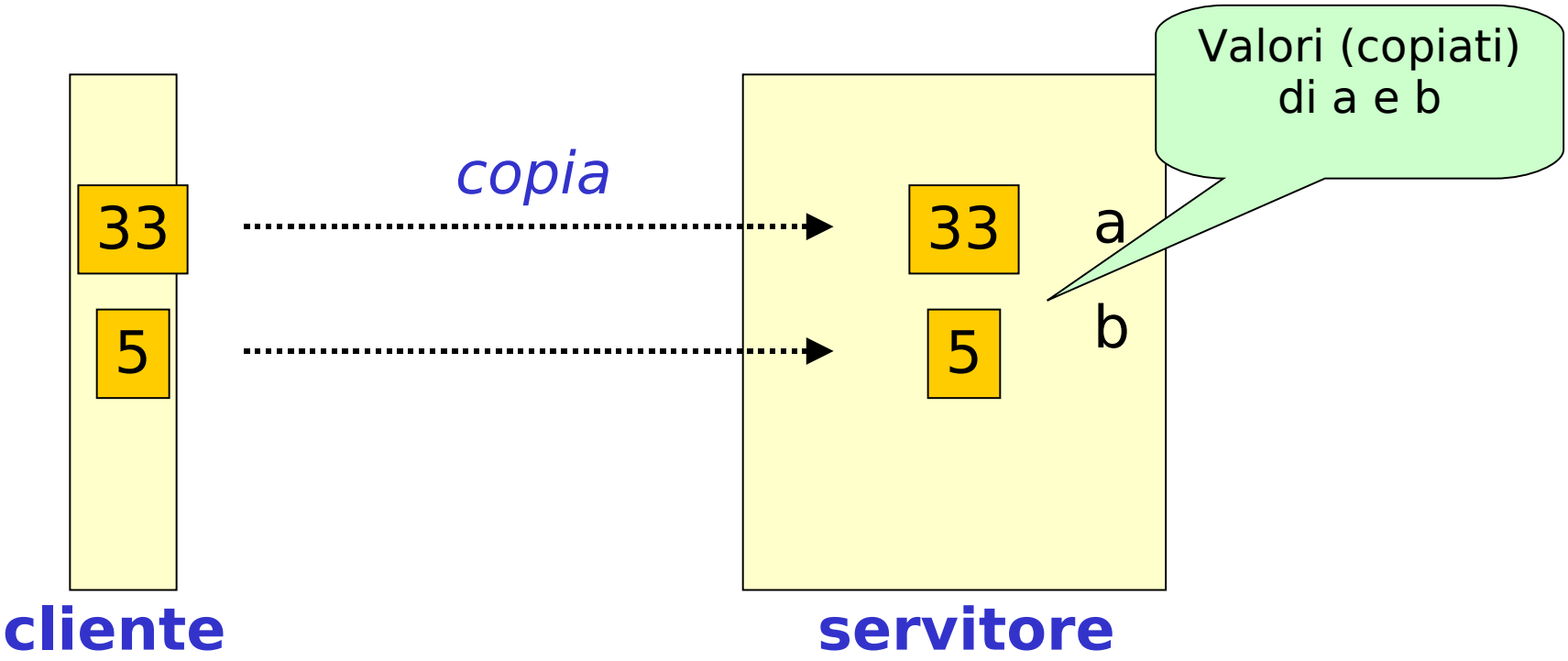
Il client invocherebbe quindi la procedura così:

```
main(){
  int y = 5, x = 33;
  scambia(x, y);
  /* ora dovrebbe essere
     x=5, y=33 ...
      MA NON È VERO
  */
}
```

***Perché non funziona?***

# Esempio: cosa è successo?

- Ogni azione fatta su **a** e **b** è strettamente locale al server. Quindi **a** e **b** vengono scambiati, ma quando l'istanza del server termina, tutto scompare



# Passaggio dei parametri in C

## Il C adotta sempre il passaggio per valore!

- È sicuro: le variabili del cliente e del servitore sono disaccoppiate
- … ma non consente di scrivere componenti software il cui scopo sia diverso dal calcolo di una espressione
- Per superare questo limite occorre il passaggio per riferimento (by reference)

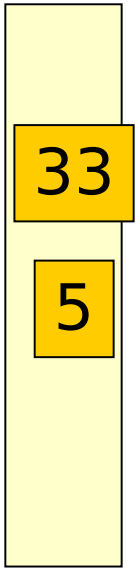
# Passaggio per riferimento

Il passaggio per riferimento (*by reference*):

- non trasferisce una copia del valore del parametro attuale
- *ma un riferimento al parametro,* in modo da dare al server *accesso diretto* al parametro in possesso del client
- il server, quindi, ***accede direttamente*** al dato del client *e* ***può modificarlo***

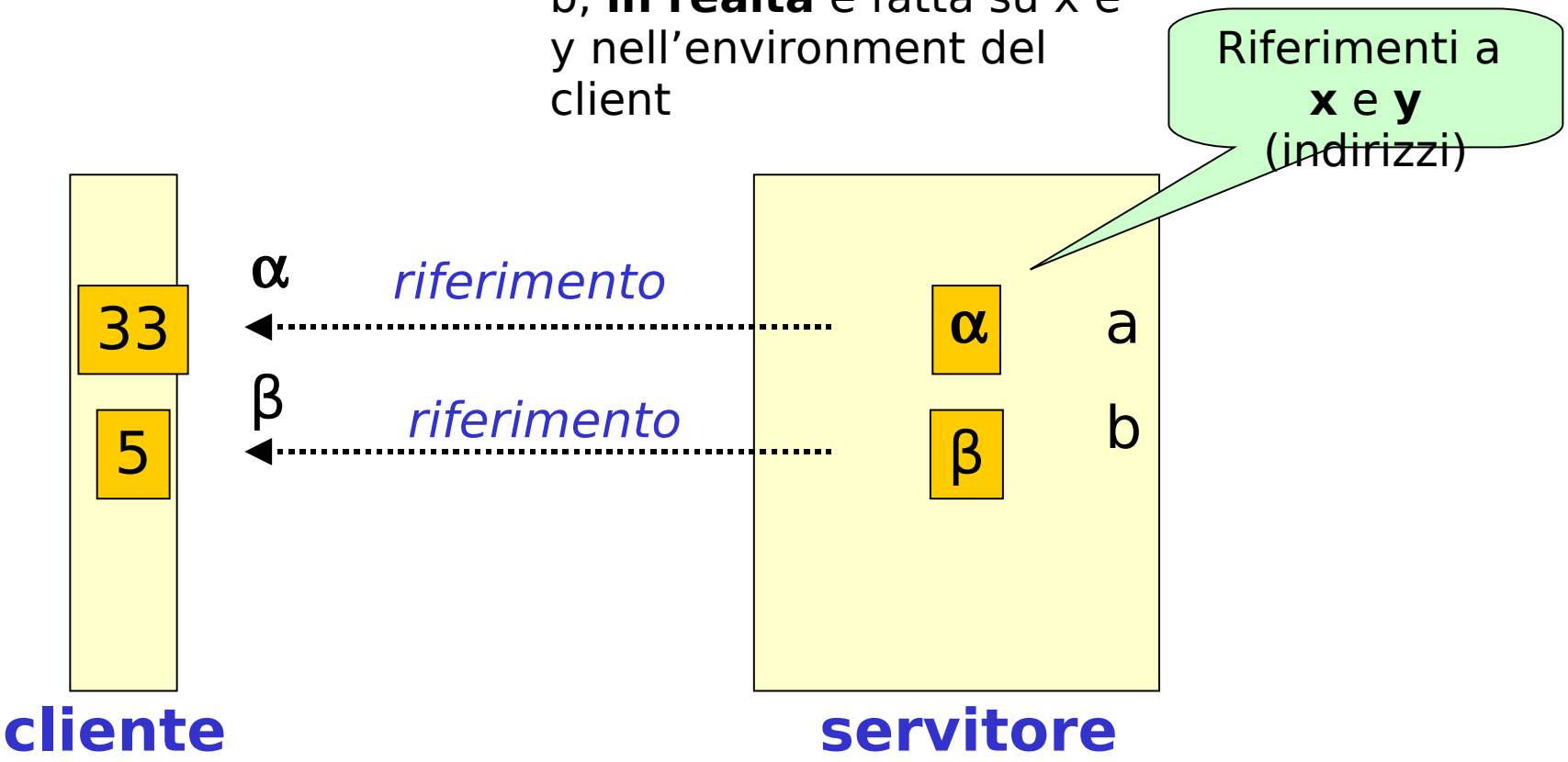
# Passaggio per riferimento

- Si trasferisce un riferimento ai parametri attuali (cioè i loro indirizzi)



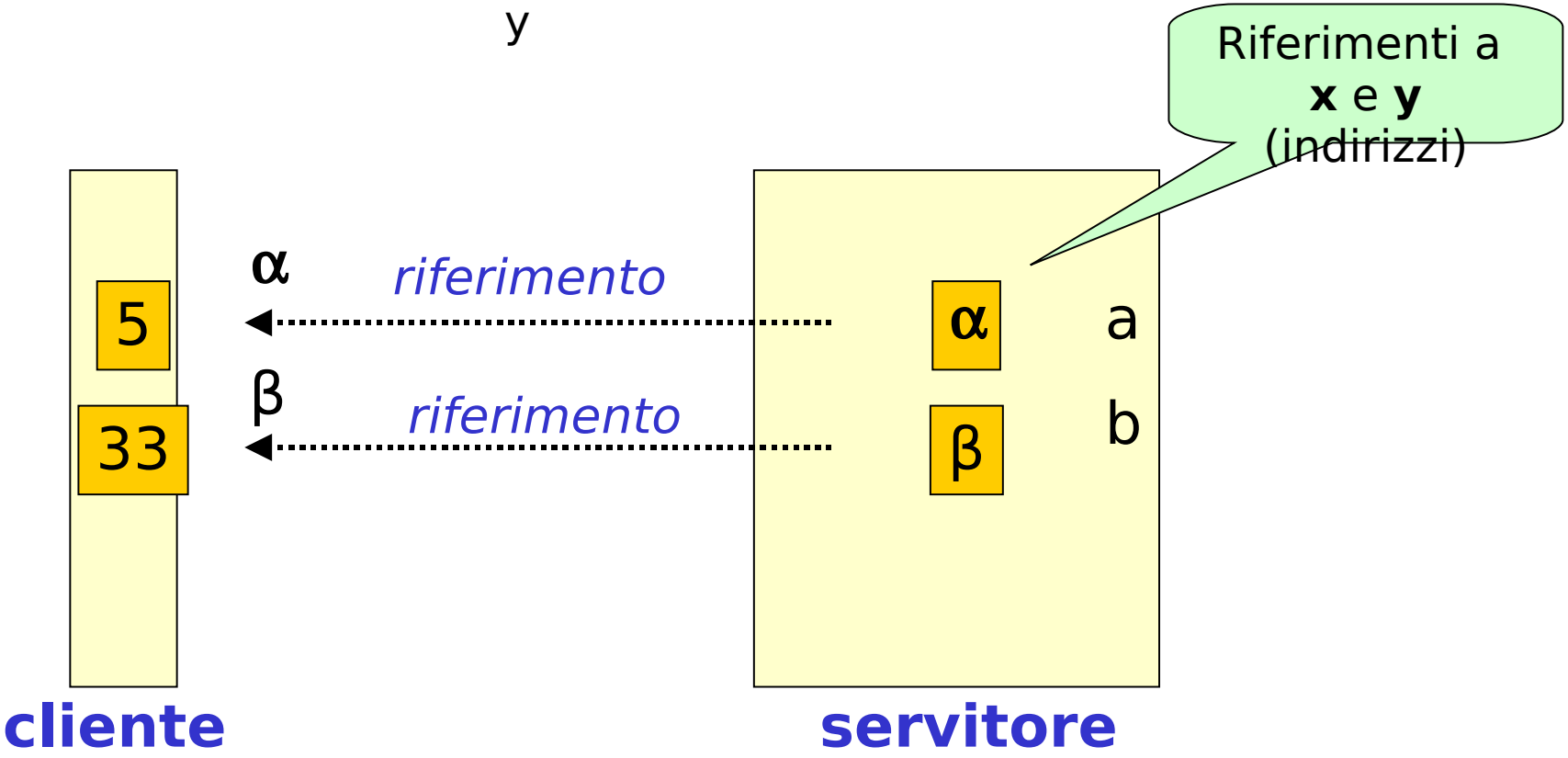
# Passaggio per riferimento

Ogni azione fatta su a e b, **in realtà** è fatta su x e y nell'environment del client



# Passaggio per riferimento

Quindi, scambiando a e b,
**in realtà** si scambiano x e y



# Realizzare il passaggio per riferimento in C

- Il C *non* fornisce *direttamente* un modo per attivare il passaggio per riferimento -> a volte occorre *costruirselo*

**È possibile costruirlo? Come?**

- Poiché passare un parametro per riferimento comporta la capacità di manipolare *indirizzi di variabili* ...
- ... gestire il passaggio per riferimento implica la capacità di *accedere, direttamente o indirettamente, agli indirizzi* delle variabili

# Indirizzamento e dereferencing

- Ricordiamoci che il C offre a tale scopo *due operatori*, che consentono di:

- **ricavare l'indirizzo di una variabile**
  - operatore *estrazione di indirizzo* **&**

- **dereferenziare un indirizzo di variabile**
  - operatore *di dereferenziamento* *****

# Realizzare il passaggio per riferimento in C

- il client deve *passare esplicitamente gli indirizzi*
- il server deve *prevedere esplicitamente dei* ***puntatori*** *come parametri formali*

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

main(){
  int y=5, x=33;
  scambia(&x, &y);
}
```

# Osservazione

- Quando un puntatore è usato per realizzare il passaggio per riferimento, *la funzione non dovrebbe mai alterare il valore del puntatore*

- Quindi, se **a** e **b** sono due puntatori:

  ***a = *b** **SI**

  **a = b** **NO**

- In generale una funzione può modificare un puntatore, ma *non è opportuno che lo faccia se esso realizza un passaggio per riferimento*

# Comunicazione tramite environment globale

- Una procedura può anche comunicare con il cliente mediante *aree dati globali:* ad esempio, *variabili globali:*
  - sono allocate nell'area dati globale (*fuori da ogni funzione*)
  - esistono *prima* della chiamata del *main*
  - sono *inizializzate automaticamente a 0* salvo diversa indicazione
  - possono essere *nascoste* in una funzione da una variabile locale omonima
- Il loro uso è fortemente **SCONSIGLIATO** poichè:
  - Rendono le funzioni non autosufficienti, in quanto legate a variabili esterne
  - La modifica delle variabili utilizzate dalla funzione è possibile anche da parte di altre funzioni
  - Minore leggibilita'

# Esempio

- **Esempio:** Divisione intera x/y con calcolo di quoziente e resto. Occorre calcolare *due* valori che supponiamo di mettere in due variabili globali

```
int quoziente, int resto;

void dividi(int x, int y) {
   resto = x % y; quoziente = x/y;
}
main(){
  dividi(33, 6);
  printf("%d%d", quoziente,resto);
}
```

Variabili globali **quoziente** e **resto** visibili in tutti i blocchi

Il risultato è disponibile per il cliente nelle variabili globali **quoziente** e **resto**

# Esempio

- **Esempio:** Divisione intera x/y con calcolo di quoziente e resto.
  Con il passaggio dei parametri per indirizzo avremmo il seguente codice.

Non ho più variabili globali

```
void dividi(int x, int y, int *quoziente, int
  *resto) {
   *resto = x % y; *quoziente = x/y;
}
main(){
  int k=33, h=6, quoz, rest;
  int *pq= &quoz, *pr = &rest;
  dividi(k, h, pq, pr);
  printf("%d%d", quoz,rest);
}
```

Il risultato è disponibile per il cliente nelle variabili **quoz** e **rest** di cui ho passato l'indirizzo

# Progetto di una funzione

- *Scegliere un nome significativo per la funzione*

*La funzione deve ricevere qualche dato dalla funzione chiamante?*

  - Se sì, elencare ed identificare tutti i tipi di dato da passare alla funzione (lista dei parametri)
  - Se no, la lista dei parametri è void (indicabile anche non specificando alcun parametro nella lista)

- *La funzione deve restituire un valore alla funzione chiamante?*
  - Se sì, identificare il tipo di dato
  - Se no, il tipo di ritorno della funzione è void
- *La funzione deve restituire più di un valore alla funzione chiamante?*
  - Se sì, bisogna inserire nella lista dei parametri dei parametri passati esplicitamente per riferimento, cioè dei puntatori

# Progetto di una funzione: lista dei parametri

- **Parametri formali:**
  argomenti dichiarati nella definizione di funzione

- Devono essere variabili:
  - Non appena l'ambiente della funzione chiamata viene attivato, i parametri formali vengono dichiarati (come variabili locali all'ambiente della funzione) ed inizializzati al valore del corrispondente parametro attuale
  - La corrispondenza tra parametri formali ed attuali è sia posizionale sia di tipo. Ovvero si presume che la lista dei parametri formali e la lista dei parametri attuali abbia lo stesso numero e tipo di elementi
  - I nomi dei parametri attuali e formali non hanno importanza. *Possono essere gli stessi o diversi*. L'importante è la posizione ed il valore che assume un parametro attuale al momento della chiamata