PROGRAMMAZIONE 2
20. Funzioni e scope dinamico

Funzioni come valori

- Nei *linguaggi funzionali* le funzioni tipicamente sono valori esprimibili (possono essere risultato della valutazione di espressioni)
- Funzioni di ordine superiore: prendono come parametro una funzione o restituiscono una funzione
- Questo porta a dei problemi sia a livello semantico che di implementazione
- Consideriamo i seguenti due casi
  - funzione passata come parametro attuale (semplice)
  - funzione restituita come risultato di un'altra funzione: può essere utilizzata nel seguito della computazione (più complicato)

Parametri funzionali

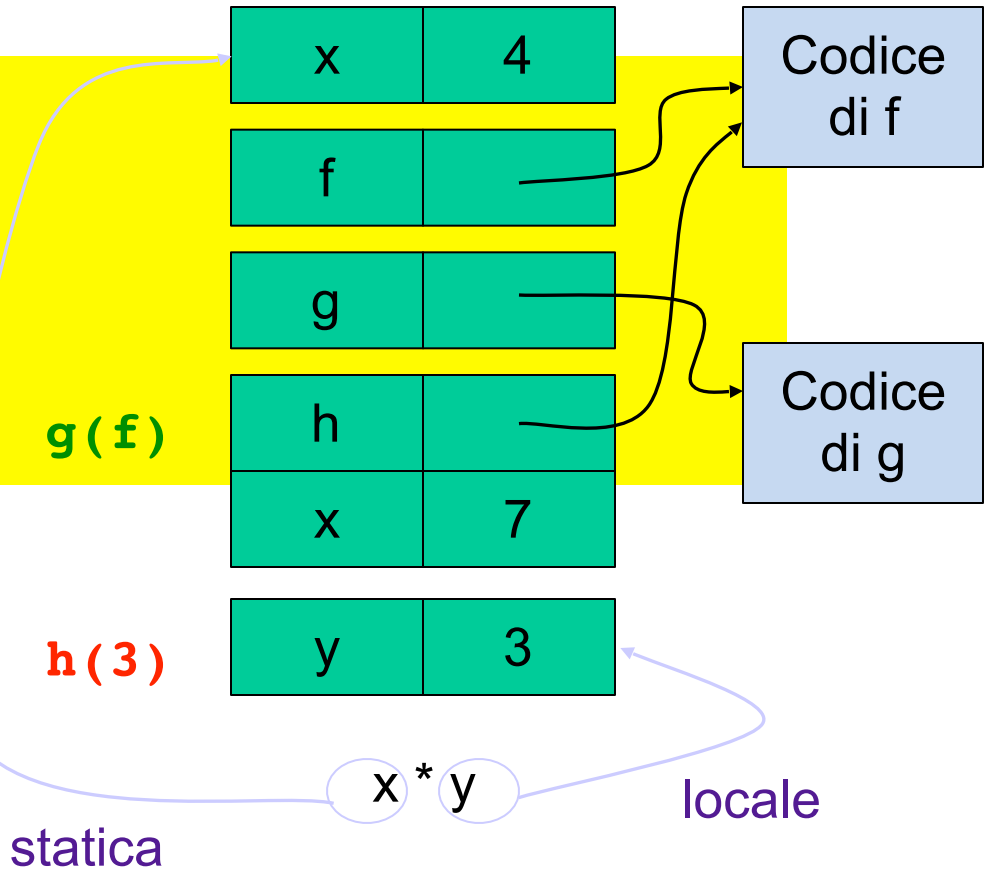
Haskell

```
int x = 4;
   fun f(y) = x*y;
      fun g(h) = let
                    int x = 7
                 in
                    h(3) + x;
      g(f);
```

Due dichiarazioni per la variabile **x**
Quale deve essere usata nella chiamata **g(f)**?

Parametri funzionali: scope statico

```
int x = 4;
    fun f(y) = x*y;
        fun g(h) = let
          int x = 7
                  in
              h(3) + x;
        g(f);
```



Cosa calcola?

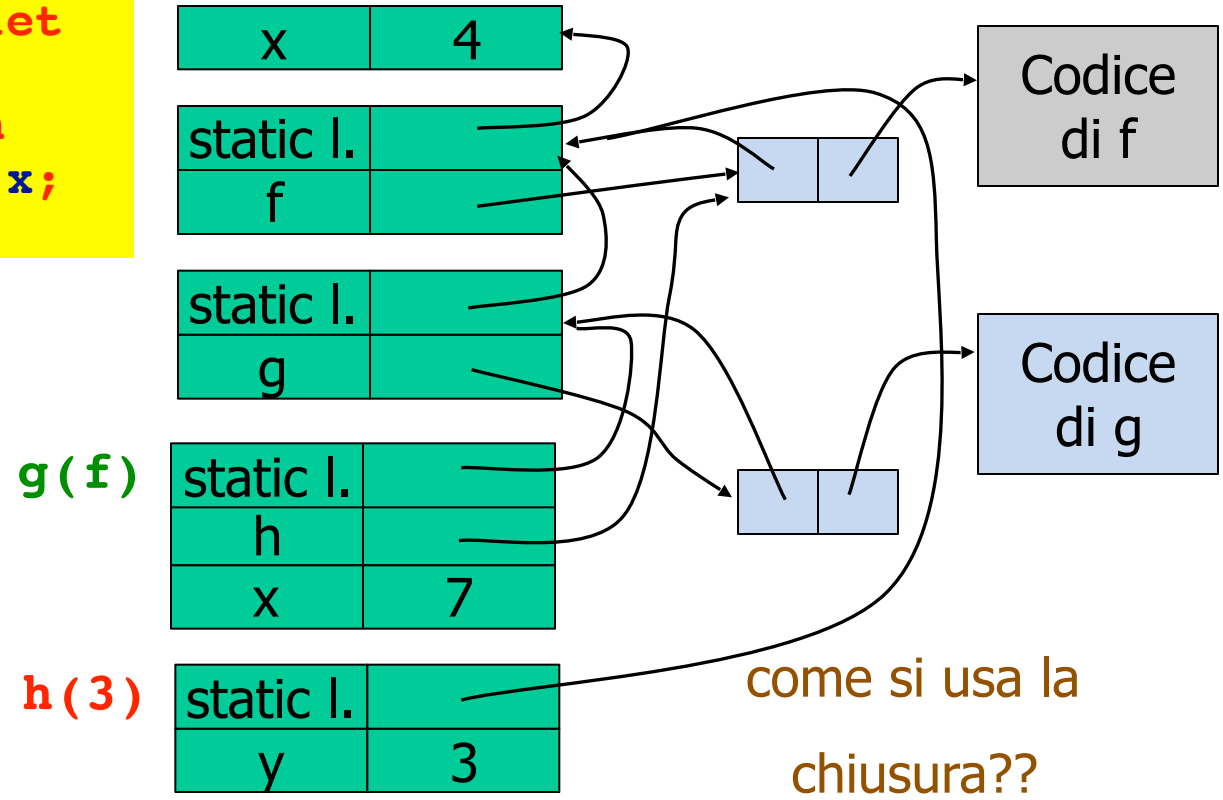
Come si determina?

Chiusure

- Il valore di una funzione trasmessa come parametro è una coppia denominata chiusura
  - *closure* = ⟨*env_dichiarazione*, *codice_funzione*⟩
- Quando il parametro formale (funzionale) viene invocato
  - si alloca sullo stack l'AR della funzione
  - si mette come valore del *puntatore di catena statica* il puntatore a *env_dichiarazione*

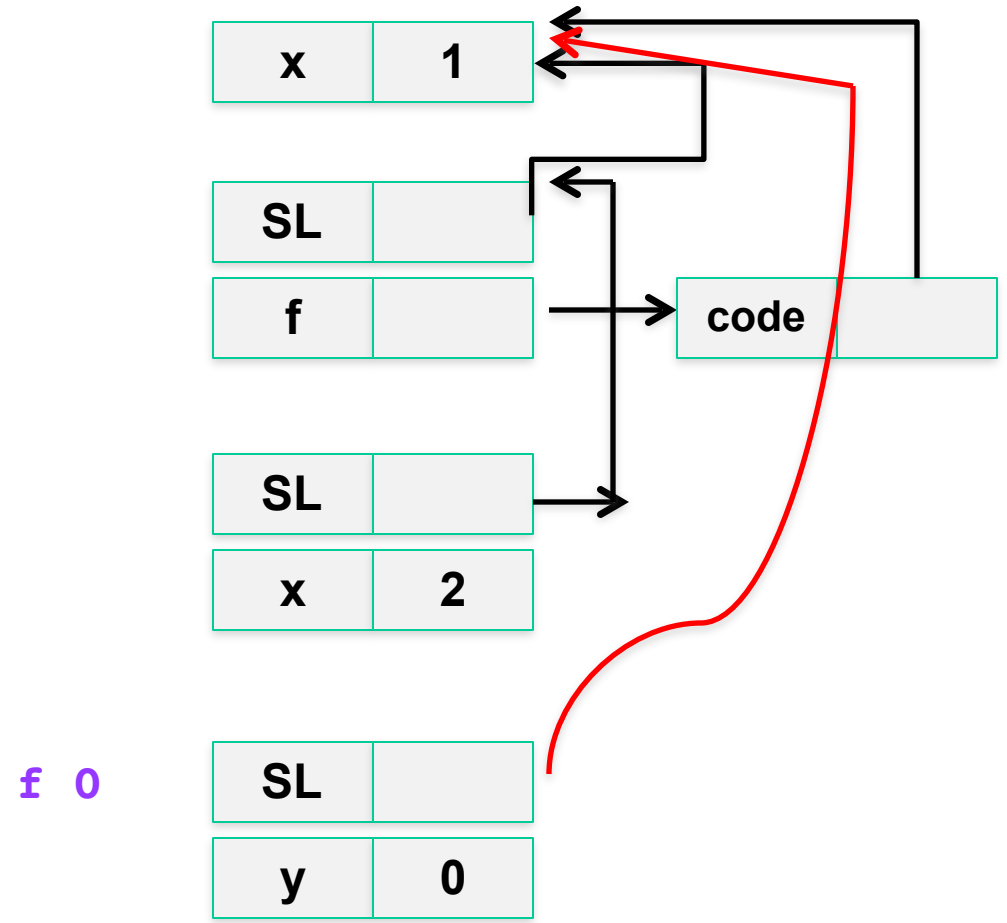
Struttura del run time

```
int x = 4;
    fun f(y) = x*y;
        fun g(h) = let
         int x = 7
                  in
              h(3) + x;
        g(f);
```



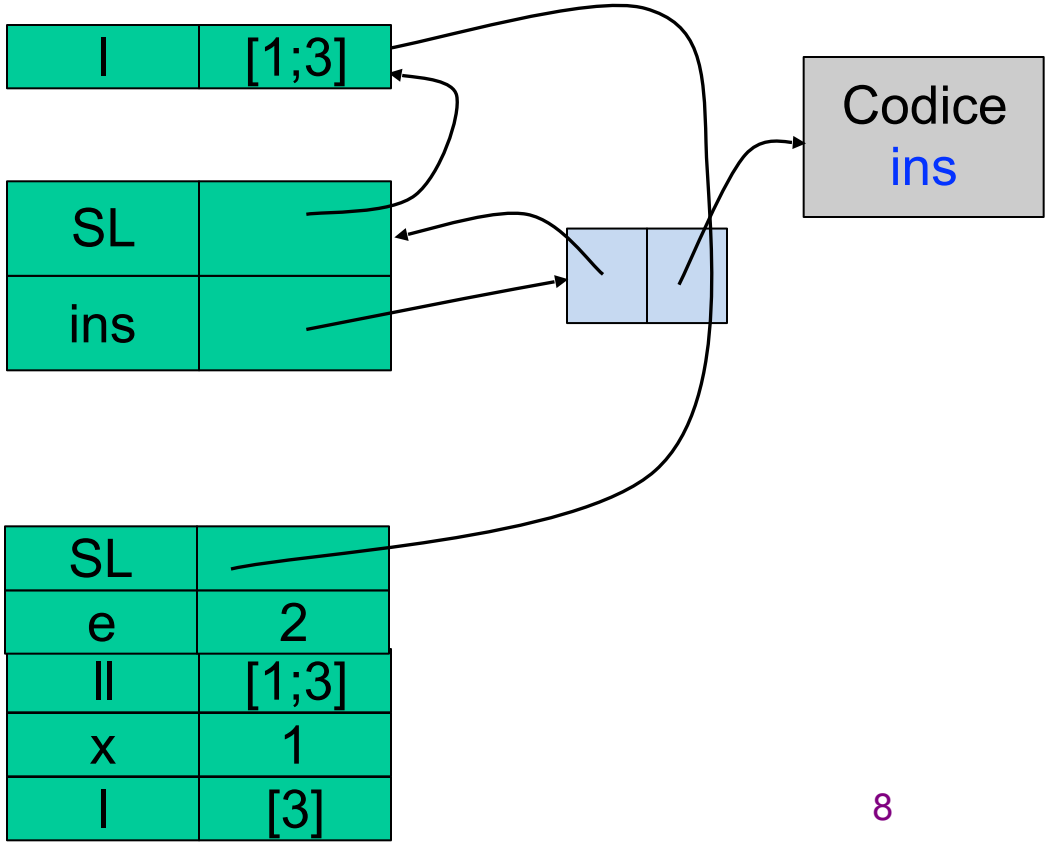
OCaML: funzioni e chiusure

```
let x = 1 ;;
let f y = y + x;;
let x = 2;;
let z = f 0;;
```



OCaML: ricorsione

```
let l = [1;3];;
let rec ins e ll =
    match ll with
      [] -> [e]
    | x :: l ->
        if e < x then
          el :: x :: l
            else
            x :: ins e l;;
let l1 = ins 2 l
```



Argomenti funzionali

- Si usano le chiusure per mantenere l'informazione sull'ambiente presente al momento della dichiarazione
- Si usa la chiusure per determinare il puntatore di catena statica

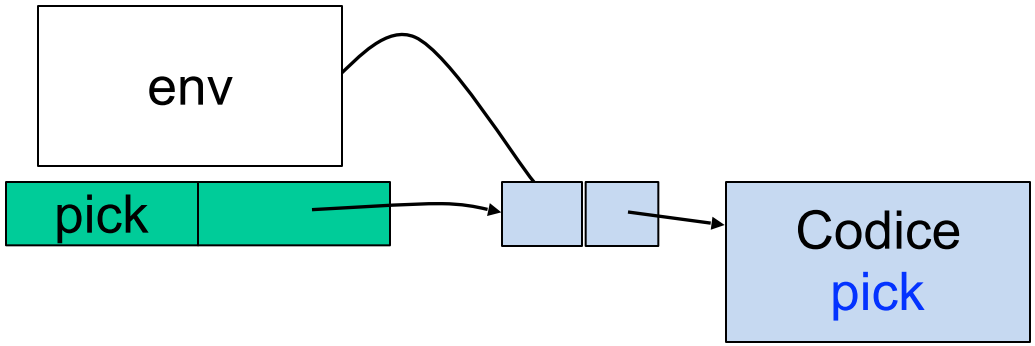
Funzioni come risultato

- Funzione che restituisce come valore una nuova funzione
  - bisogna congelare l'ambiente dove la funzione è "dichiarata"
- Esempio

```
function compose(f, g)
 {return  function(x) { return g(f(x)) } };
```

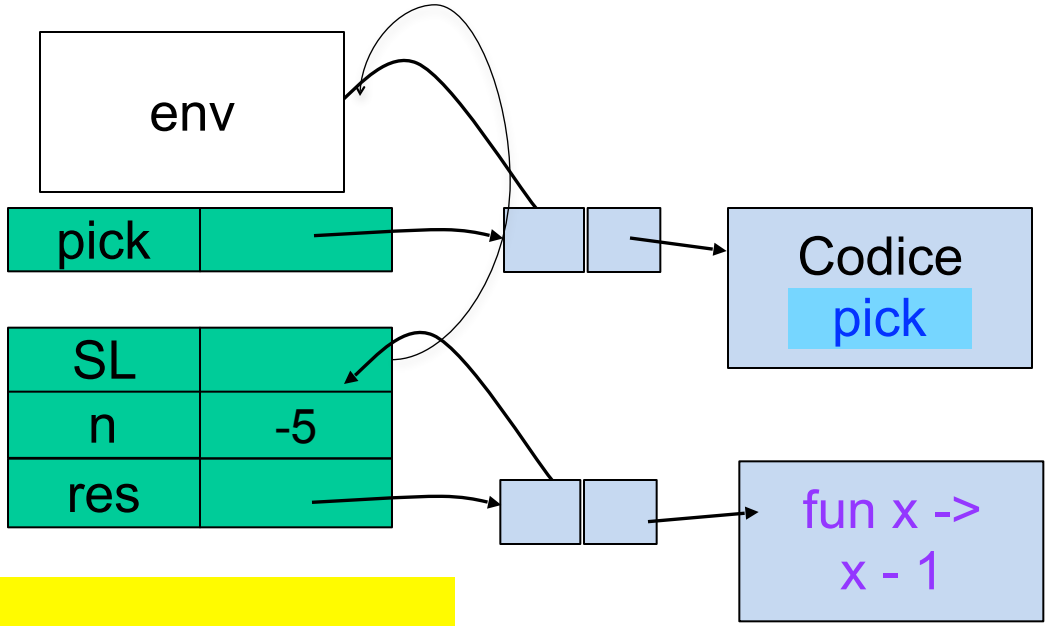
- Funzione "dichiarata" dinamicamente
  - la funzione può avere variabili non locali
  - valore restituito è una chiusura ⟨**env, code**⟩
  - **attenzione:** l'AR cui punta **env** non può essere distrutto finché la funzione può essere usata (**retention**)

Esempio




```
::
let pick n =
if n > 0 then (fun x -> x + 1)
         else (fun x -> x - 1)
let g = (pick -5);;
g 6;;
```

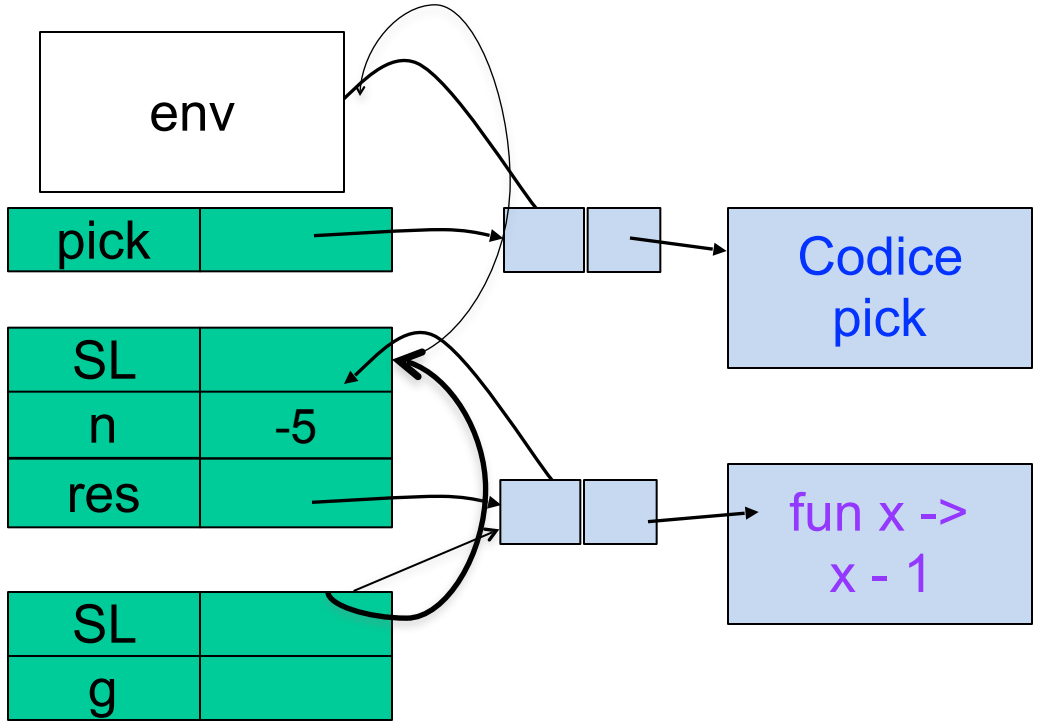
Esempio




```
::
let pick n =
if n > 0 then (fun x -> x + 1)
         else (fun x -> x - 1)
let g = (pick -5);;
g 6;;
```

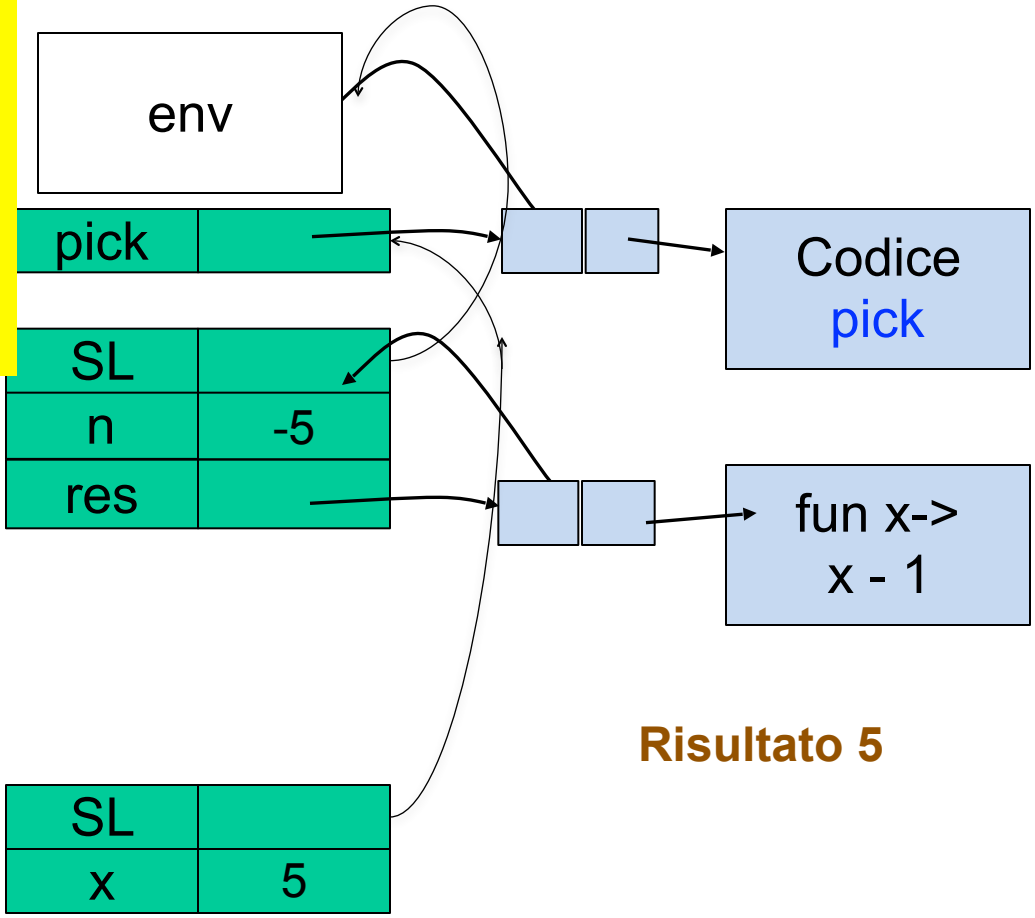
Esempio

```
let pick n =
if n > 0 then
(fun x -> x + 1)
  else (fun x -> x - 1)
let g = (pick -5);;
g 6;;
```



Esempio

```
let pick n =
if n > 0 then
(fun x -> x + 1)
  else (fun x -> x - 1)
let g = (pick -5);;
g 6;;
```



Questo esempio non mostra retention

Risultato 5

Scope dinamico

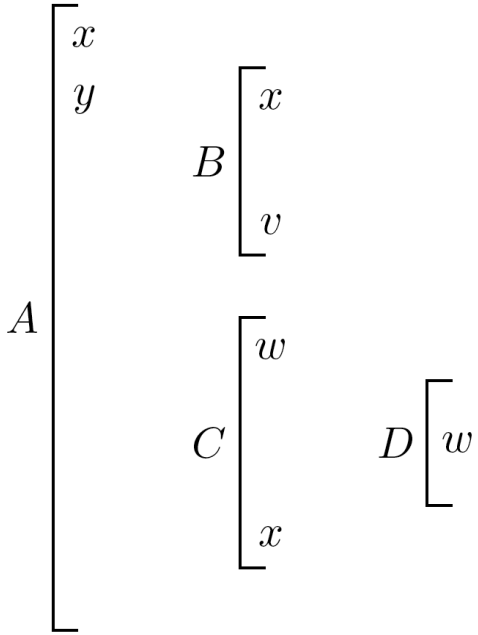
Regole scope dinamico

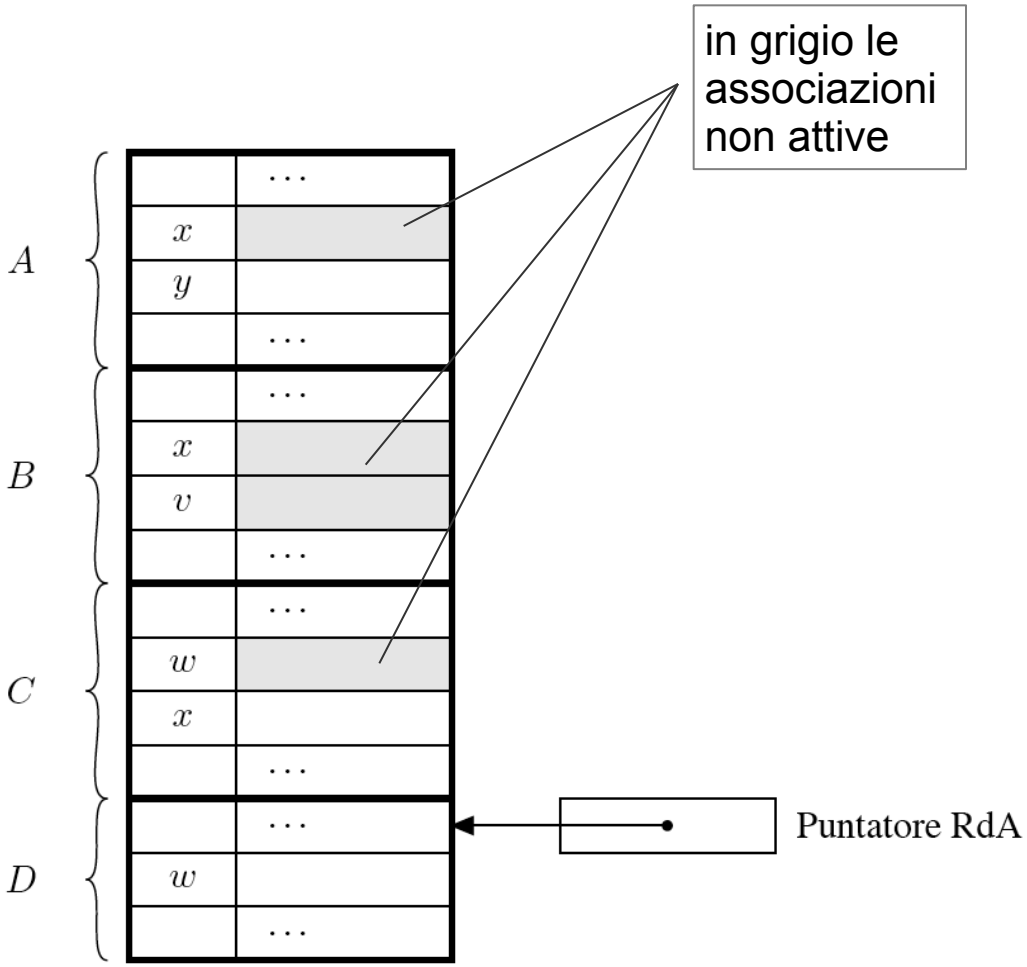
- Con scope dinamico l'associazione nomi-oggetti denotabili dipende
  - dal flusso del controllo a runtime
  - dall'ordine con il quale i sotto-programmi sono chiamati

- La regola generale è semplice: l'associazione corrente per un nome è quella determinata per ultima nell'esecuzione (non ancora distrutta)

Implementazione ovvia

- Ricerca per nome risalendo la pila
- Esempio
  - chiamate A, B, C, D





Discussione

- Lo scope statico permette di determinare tutti gli ambienti di un programma staticamente, osservando la struttura sintattica del programma
  - controlli di correttezza a compile time
  - ottimizzazione del codice a compile time
  - possibile il controllo statico dei tipi
- Gestione a run time articolata
  - gli ambienti non locali di funzioni e procedure evolvono diversamente dal flusso di attivazione e disattivazione dei blocchi (LIFO)

Scope dinamico

- L'associazione valida per un nome in un punto del programma è la *più recente associazione* creata (in senso temporale) ancora attiva secondo il flusso di esecuzione
- Quindi, lo scope dinamico ha una gestione a run time semplice
  - vantaggi: flessibilità nei programmi
  - svantaggi: difficile comprensione delle chiamate delle procedure e controllo statico dei tipi non possibile