PROGRAMMAZIONE 2
20. Funzioni e procedure

Determinare la catena statica a runtime

- Quali operazioni deve effettuare il supporto a tempo di esecuzione per determinare il link statico del chiamato?
  - è il chiamante a determinare il link statico del chiamato
- Info a disposizione del chiamante
  - annidamento statico dei blocchi (determinata dal compilatore staticamente)
  - proprio AR

Catena statica a runtime

- Il metodo chiamante C sa se se la definizione del metodo chiamato ***P*** è
  - immediatamente inclusa in C (k=0);
  - in un blocco esterno k passi fuori C
  - nessun altro caso possibile (perché)?

•Se k=0

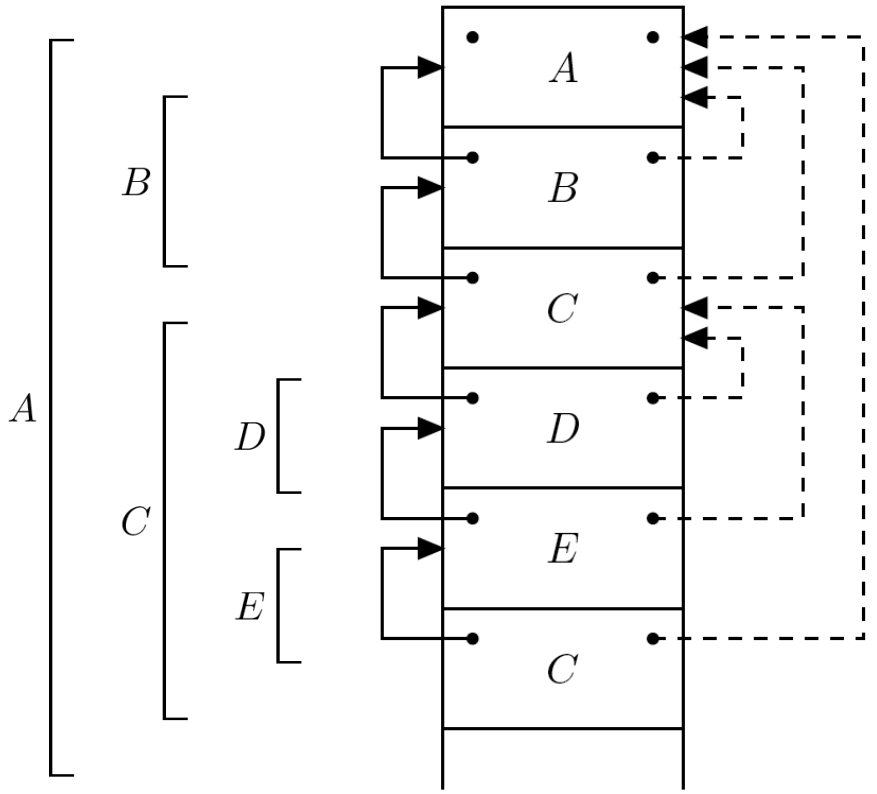
- C passa a ***P*** un puntatore al proprio AR

•Se k>0

- C risale la propria catena statica di **k** passi e passa a ***P*** il puntatore all' AR così determinato

Esempio: catena statica a runtime

– nel caso a destra
  • chiamate: **A, B, C, D, E, C**
– con i dati di catena statica
  • **A; (B,0); (C,1); (D,0); (E,1); (C,2)**

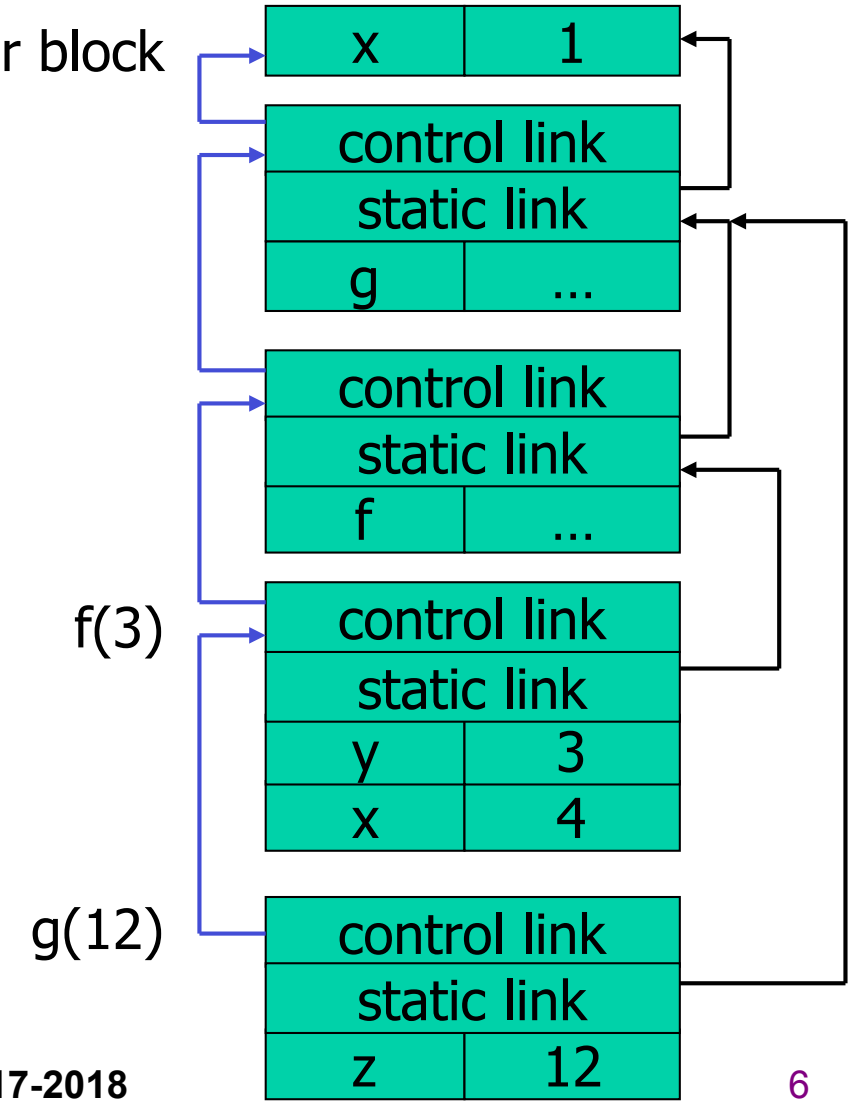


Chiamato esterno al chiamante

- Le regole dello scoping statico assicurano che affinché il chiamato sia visibile si deve trovare in un blocco esterno che includa il blocco del chiamante: *il chiamato deve essere dichiarato prima del chiamante*.
- Questo implica che l'AR che contiene la dichiarazione del chiamato è già presente sullo stack
- Assumiano che
  - **SD(Chiamante) = n**
  - **SD(Chiamato) = m**
  - distanza statica tra chiamante e chiamato **n-m**
  - il chiamante deve fare **n-m** passi lungo la sua catena statica per definire il valore del puntatore della catena statica del chiamato

Static scope: catena statica

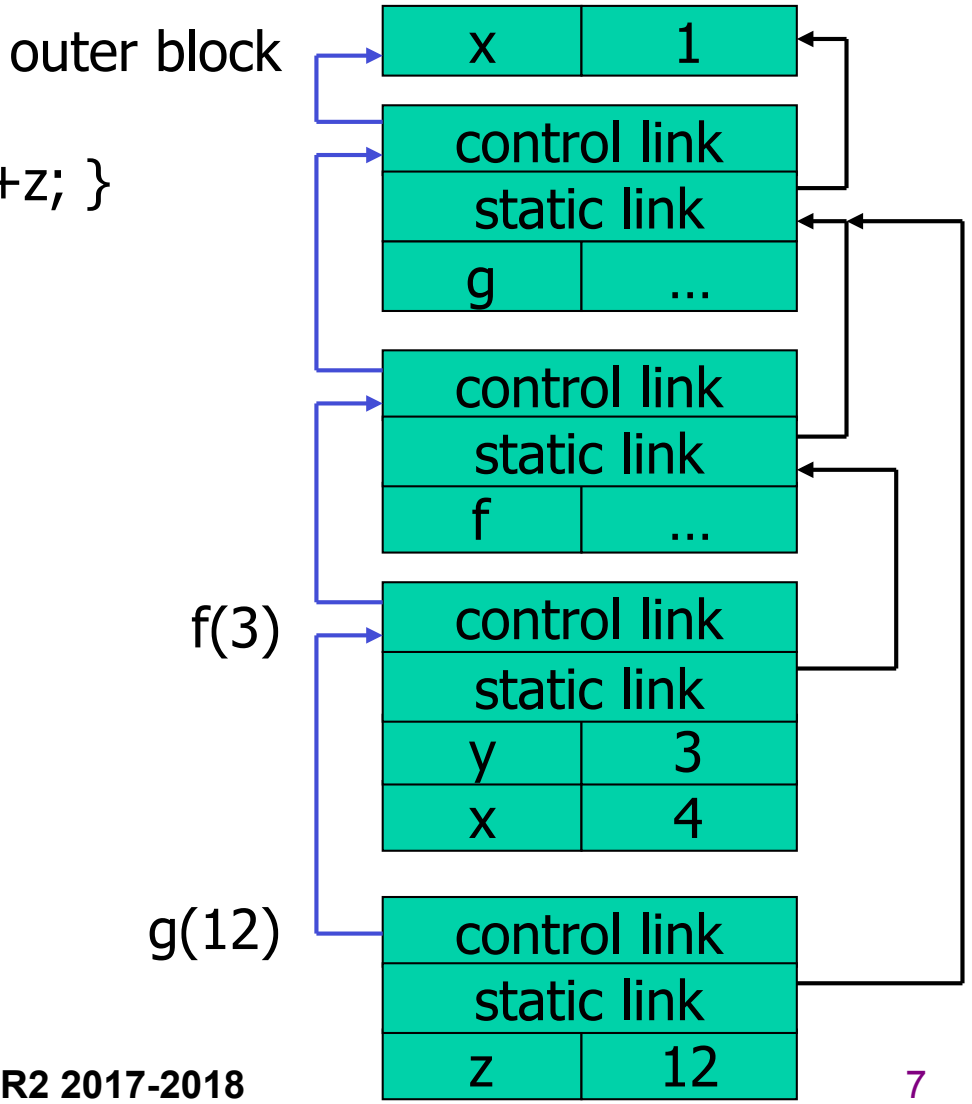
```
var x = 1;
function g(z) = { return x+z; }
    function f(y) =
        { var x = y+1;
          return g(y*x); }
    f(3);
```



Static scope: catena statica

```
var x = 1;
function g(z) = { return x+z; }
    function f(y) =
        { var x = y+1;
          return g(y*x); }
    f(3);
```

SD(g) = 1
SD(f(3)) = 3



Funzioni come valori

- Nei **linguaggi funzionali** le funzioni tipicamente sono **valori esprimibili** (possono essere risultato della valutazione di espressioni)
- 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);
```

Pseudo-JavaScript

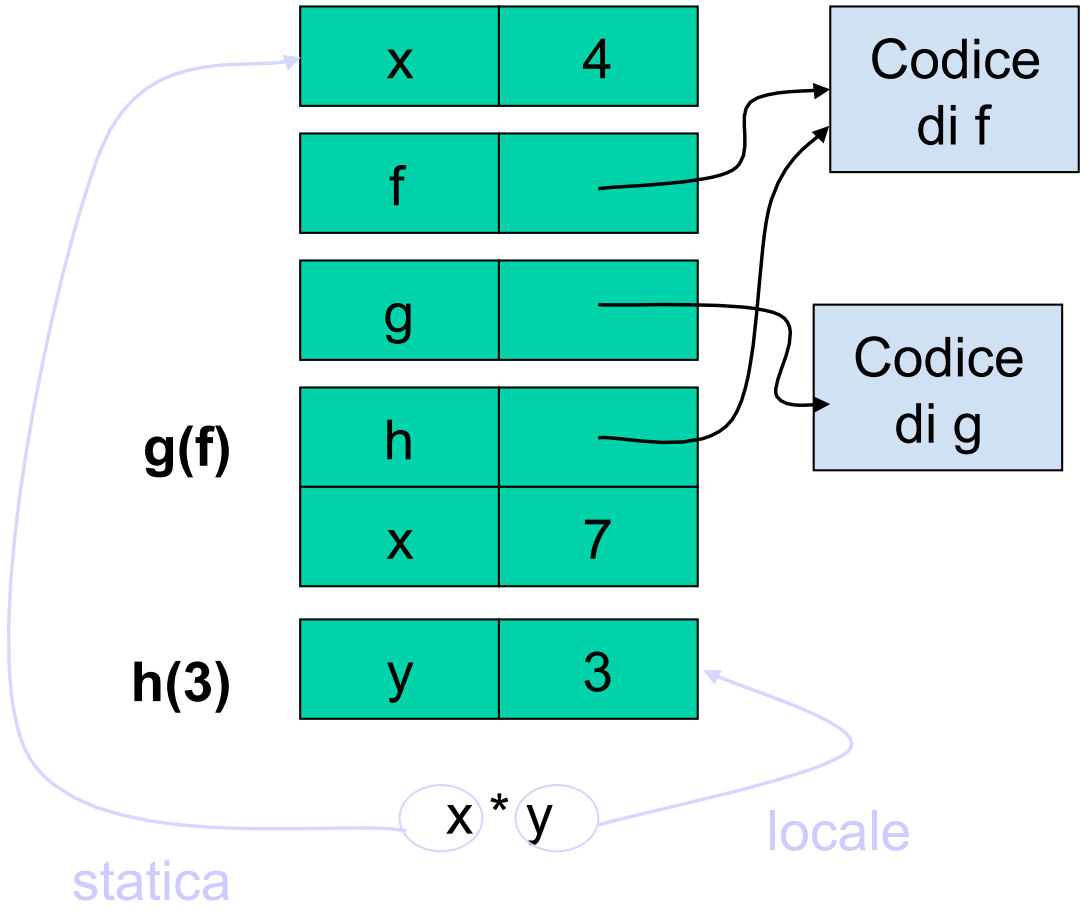
```
{ var x = 4;
  { function f(y) {return x*y};
    { function g(h) {
        var x = 7;
        return h(3) + x;
      };
      g(f);
} } }
```

Due dichiarazioni per la variabile **x**

Quale deve essere usata nella chiamata **g(f)**?

Parametri funzionali: static scope

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



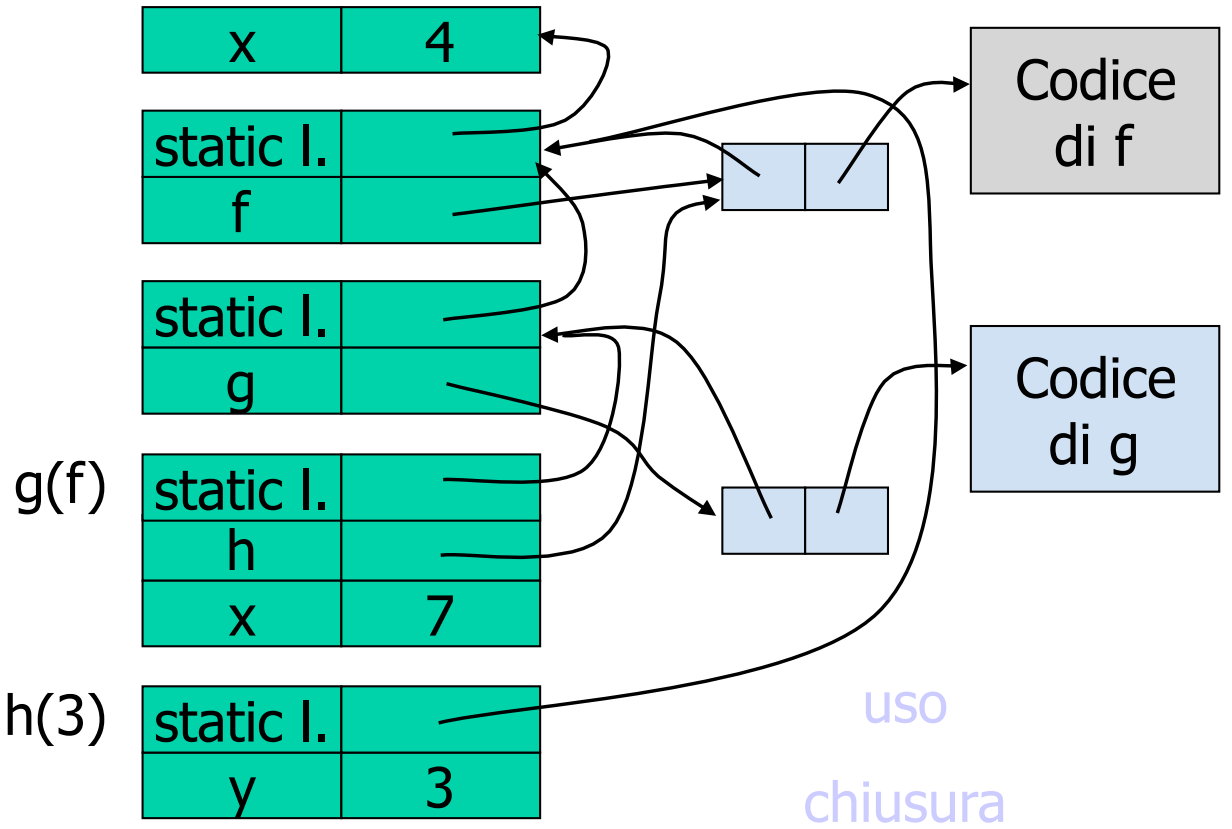
Come si determina?

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*

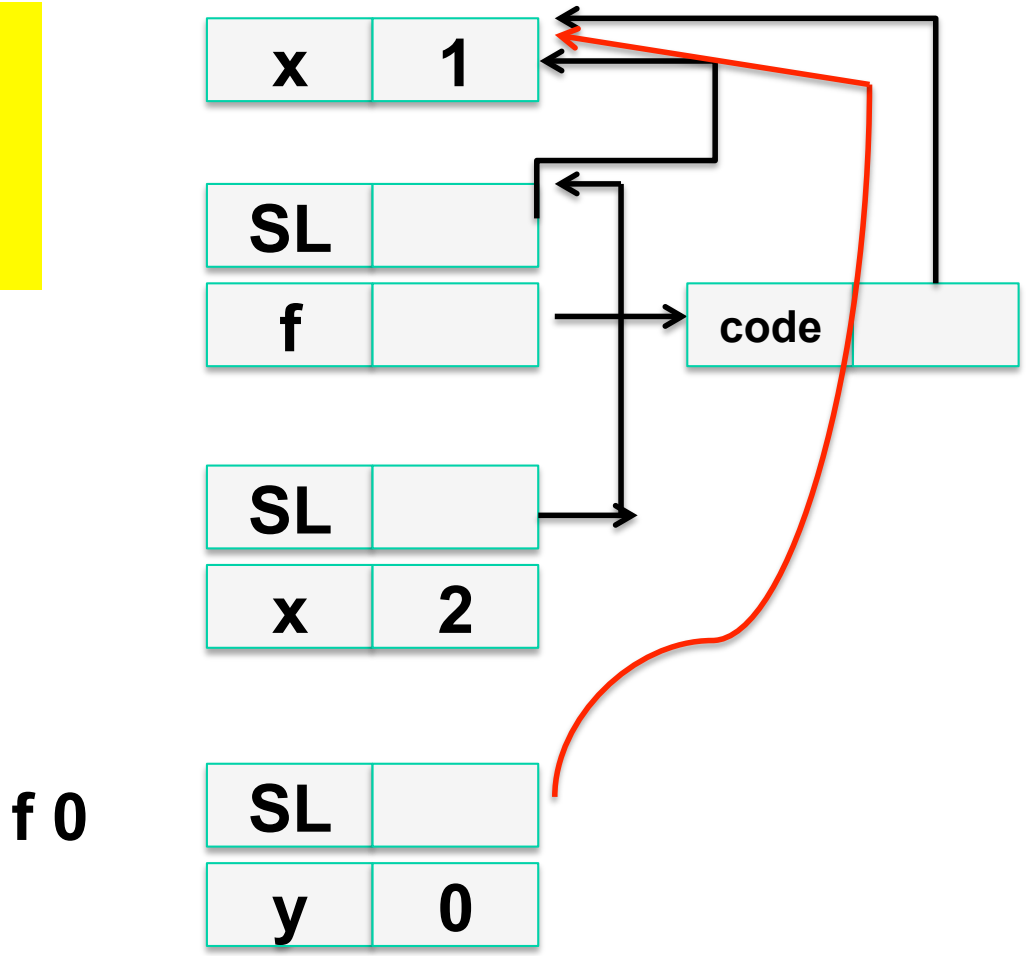
Struttura del run time

```
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;;
```



Argomenti funzionali

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

Funzioni come risultato

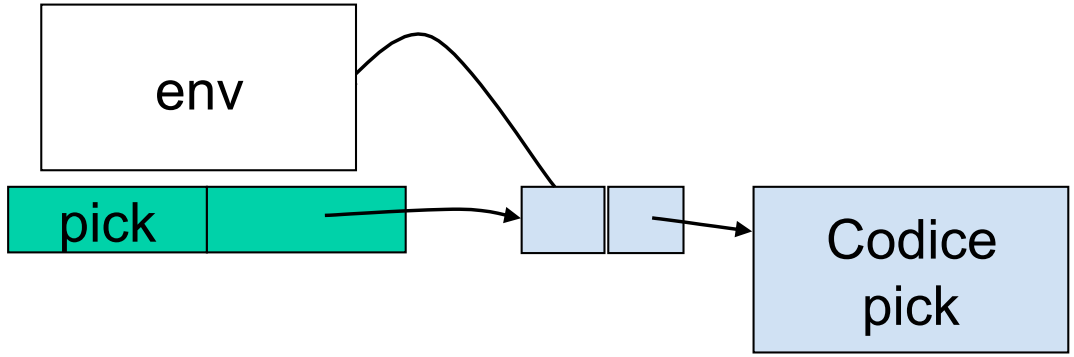
- Funzione che restituisce come valore una nuova funzione
  - bisogna congelare l'ambiente dove la funzione è "dichiarata"
- Esempio

```
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**)

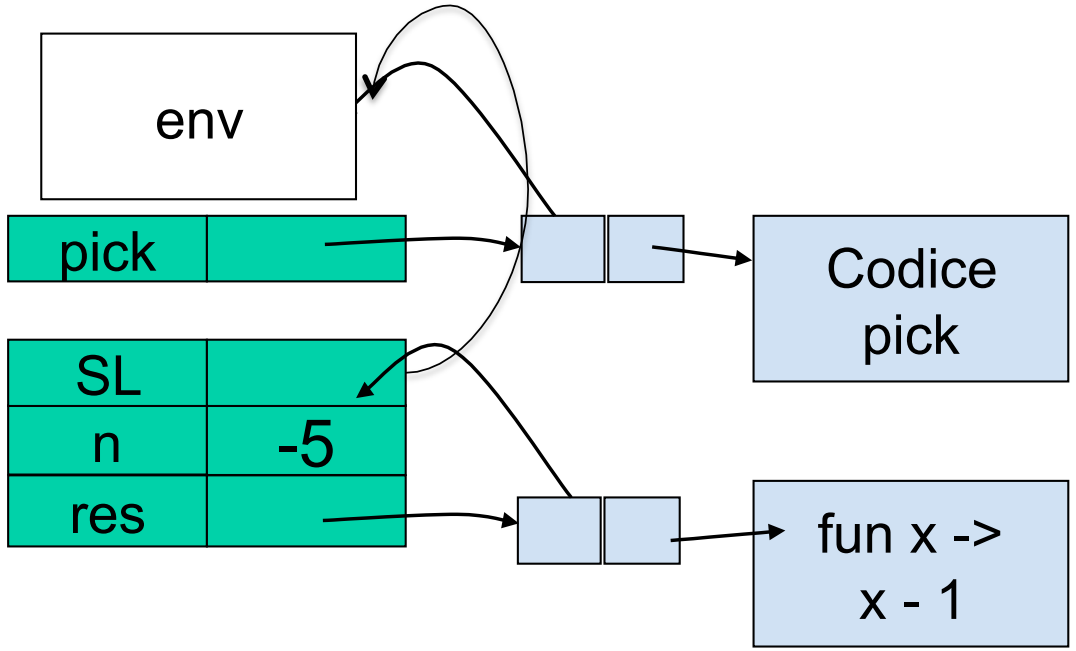
Esempio

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



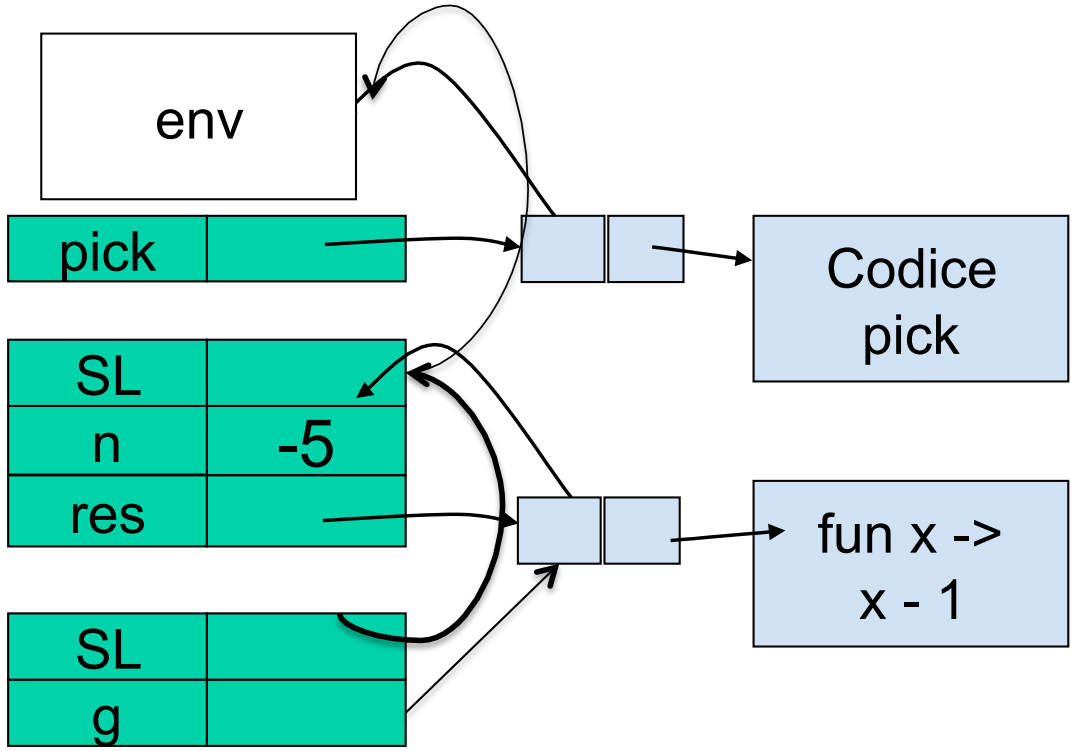
Esempio

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



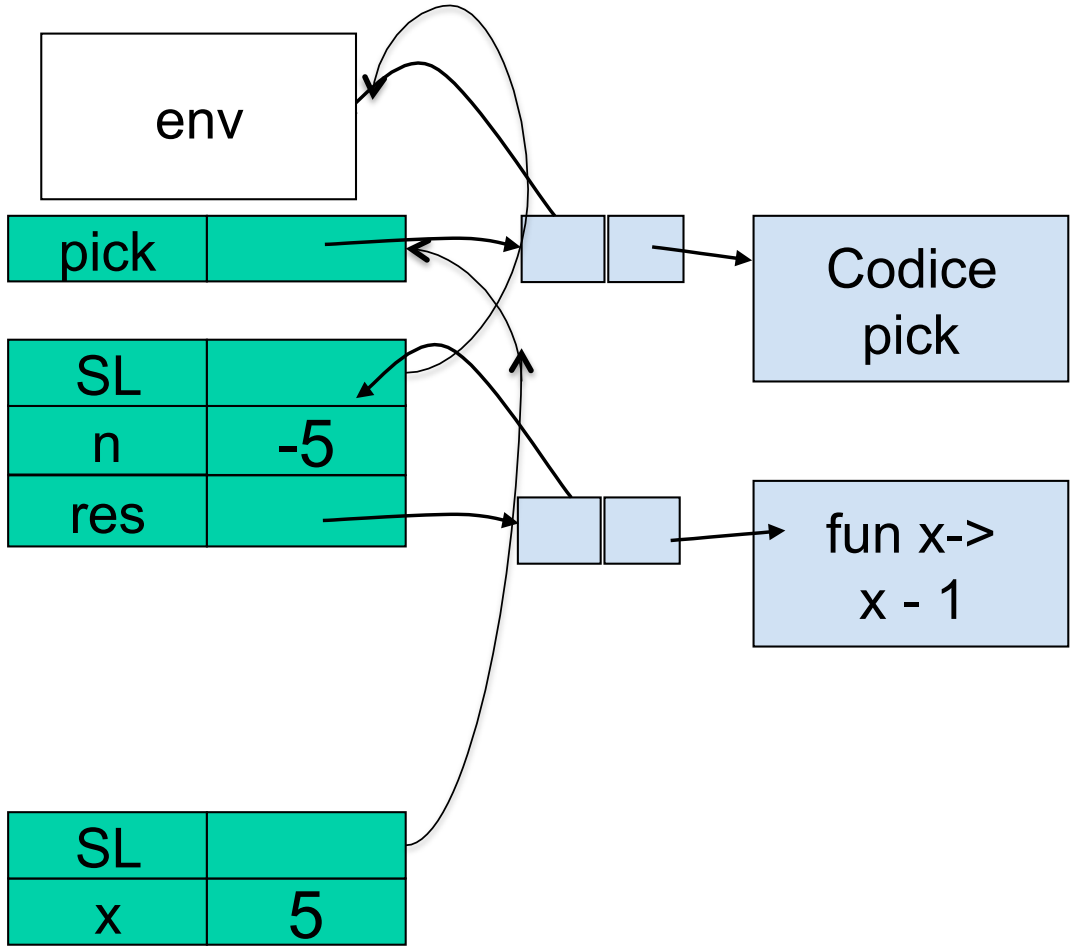
Esempio

```
:::
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;;
```



Risultato 5

Scope dinamico

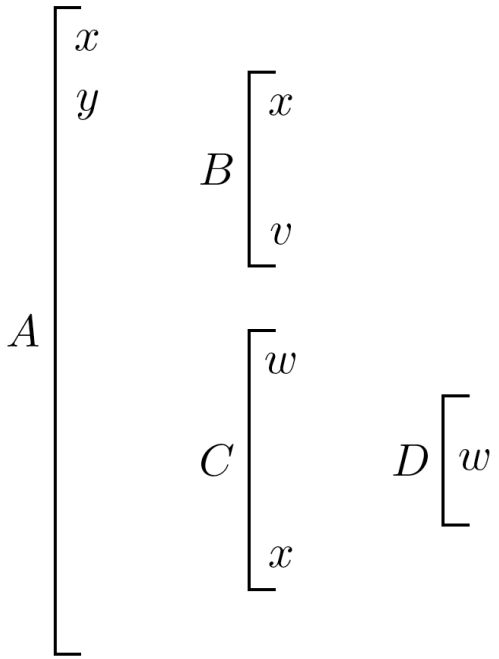
Regole scope dinamico

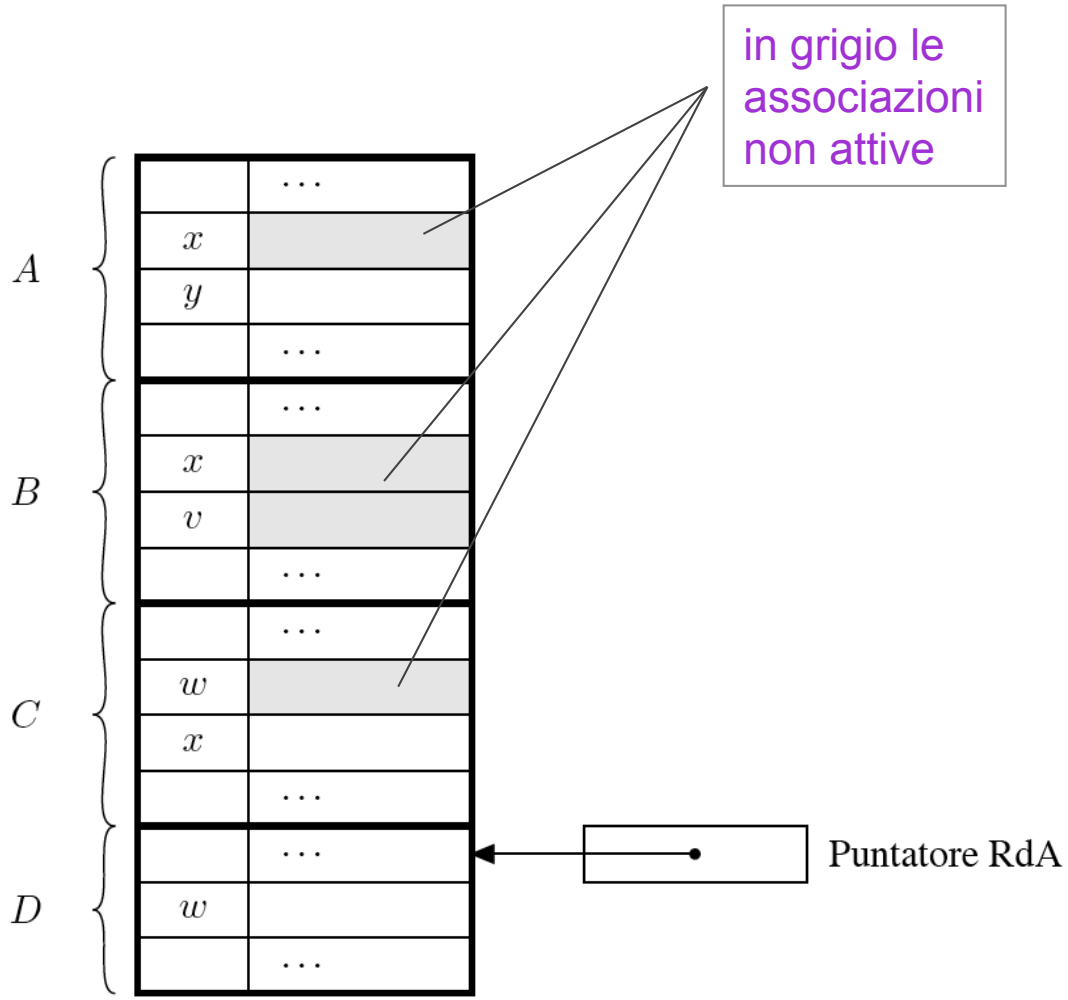
- Con scope dinamico l'associazione nomi-oggetti denotabili dipende
  - dal flusso del controllo a runtime
  - dall'ordine con il quale i sotto-programmi sono chiamati

- 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





Scope statico e analisi statica

In brevissimo

- Ambiente non locale con scope statico
  - il numero di passi che a tempo di esecuzione vanno fatti lungo la catena statica per trovare l'associazione (non locale) per l'identificatore "x" è uguale alla differenza fra le profondità di annidamento del blocco nel quale "x" è dichiarato e quello in cui è usato
- Ogni *riferimento* a un identificatore *ide* nel codice è staticamente tradotto in una coppia (m,n) di interi
  - m è la differenza fra le profondità di nesting dei blocchi (0 se *ide* si trova nell'ambiente locale)
  - n è la posizione relativa – offset – (partendo da 0) della dichiarazione di *ide* fra quelle contenute nel blocco

Valutazione

- Efficienza nella rappresentazione
  - l'accesso diventa efficiente (non c'è più ricerca per nome)
- Si può economizzare nella rappresentazione degli ambienti locali che non necessitano più di memorizzare i nomi