Ambiente
Run-time & Run-time Simulation

Semantica eseguibile

$$\frac{env \triangleright e1 \Rightarrow v1 \qquad v1 = Funval(Fun(x,e), env1) \qquad env \triangleright e2 \Rightarrow v2 \qquad env1[v2/x] \triangleright e \Rightarrow v}{env \triangleright Apply(e1,e2) \Rightarrow v}$$

```
let rec sem ((e: exp), (r: eval env)) =
   match e with
     | ...
     | Fun(x, a) -> Funval(e, r)
     | Apply(e1, e2) -> match sem(e1, r) with
       | Funval(Fun(x, a), r1) ->
              sem(a, bind(r1, x, sem(e2, r)))
       | _ -> failwith("no funct in apply")
```

ricorsione

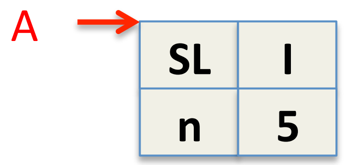
```
Rec(i, e) -> makefunrec(i, e, r)
```

```
makefunrec(f, Fun(args, body), (r: eval env)) =
    let functional(rr: eval env) =
            bind(r, f, Funval(Fun(args, body), rr)) in
              let rec (rfix: string -> eval) =
             function x -> (functional rfix) x in
              Funval(Fun(args, body), rfix)
```

Un esempio

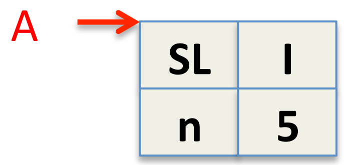
```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack



```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

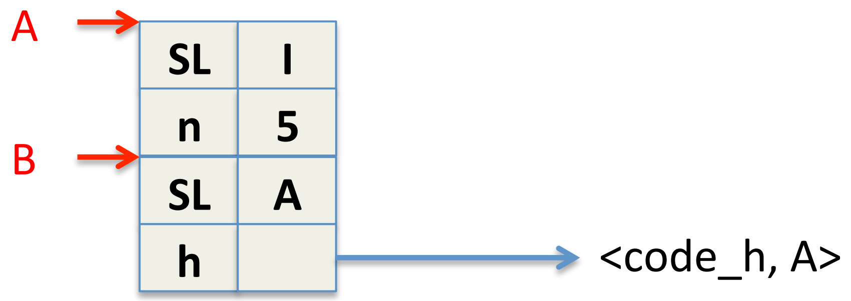
Run-time Stack: simulation



Env_A(n) = 5
Env_A(m) = unbond
for all m != n

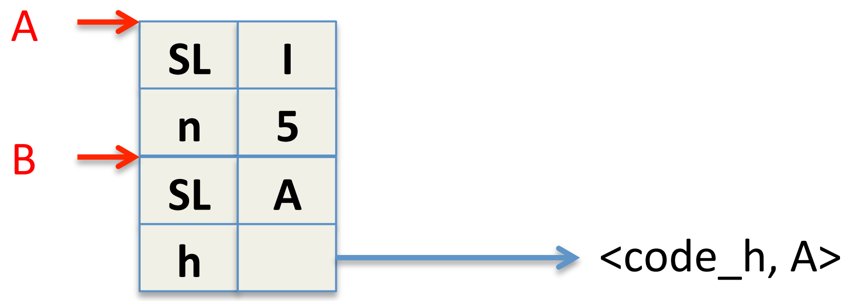
```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack



```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack: simulation

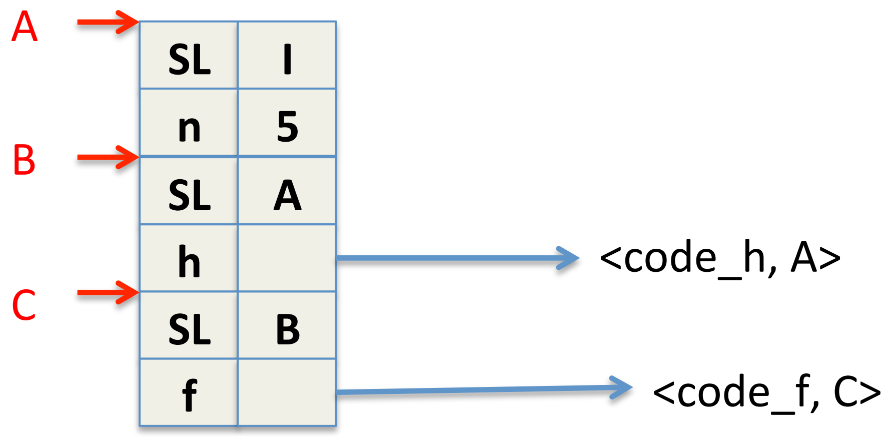


Env_A(n) = 5
Env_A(m) = unbond
for all m != n

Env_B (n) = 5
Env_B(h) = <code_h, Env_A>

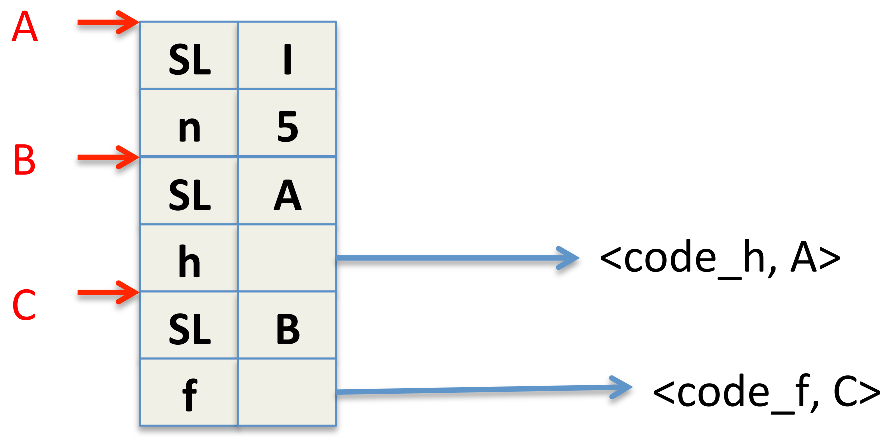
```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack



```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack: simulation



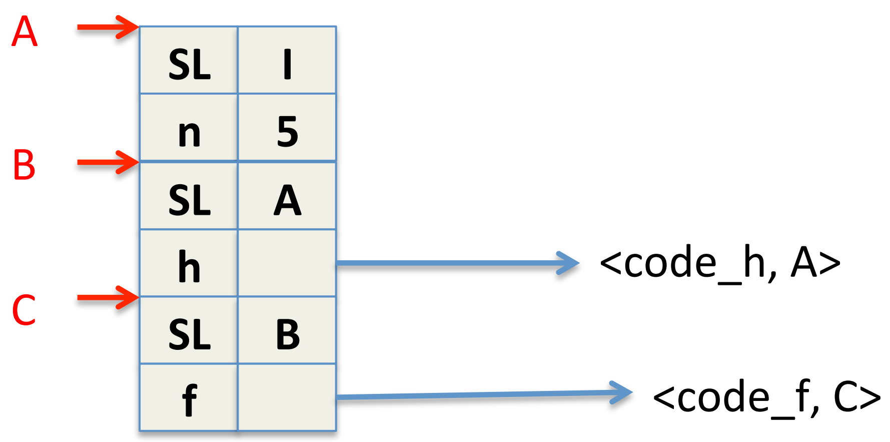
Env_A(n) = 5
Env_A(m) = unbond
for all m != n

Env_B (n) = 5
Env_B(h) = <code_h, Env_A>

Env_C(f) = <code_f, Env_C>
Env_C(h) = <code_h, Env_A>
Env_C(n) = 5

```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack: simulation



Env_A(n) = 5
Env_A(m) = unbond
for all m != n

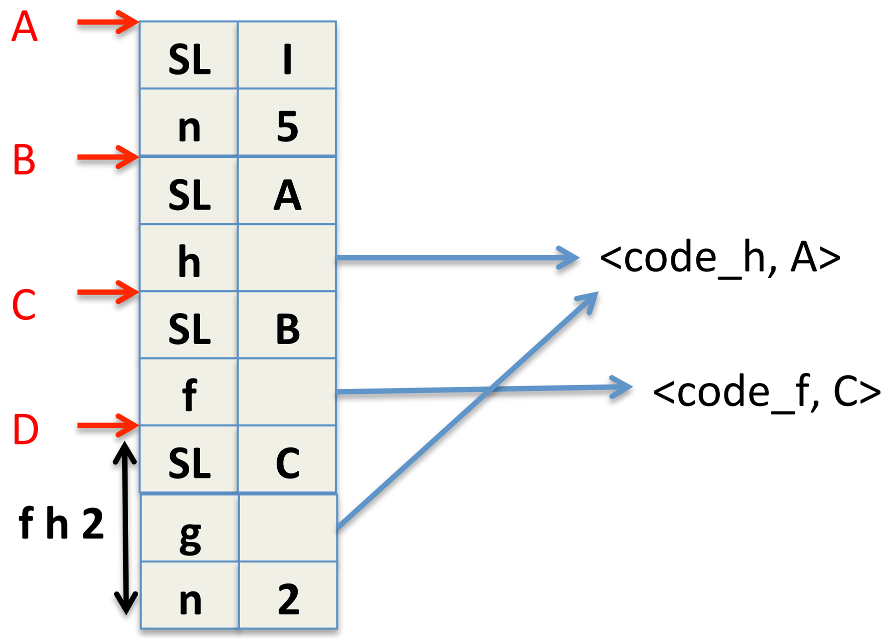
Env_B (n) = 5
Env_B(h) = <code_h, Env_A>

Env_C(f) = <code_f, Env_C>
Env_C(h) = <code_h, Env_A>
Env_C(n) = 5

Definizione ricorsiva:
makefunrec!!!!

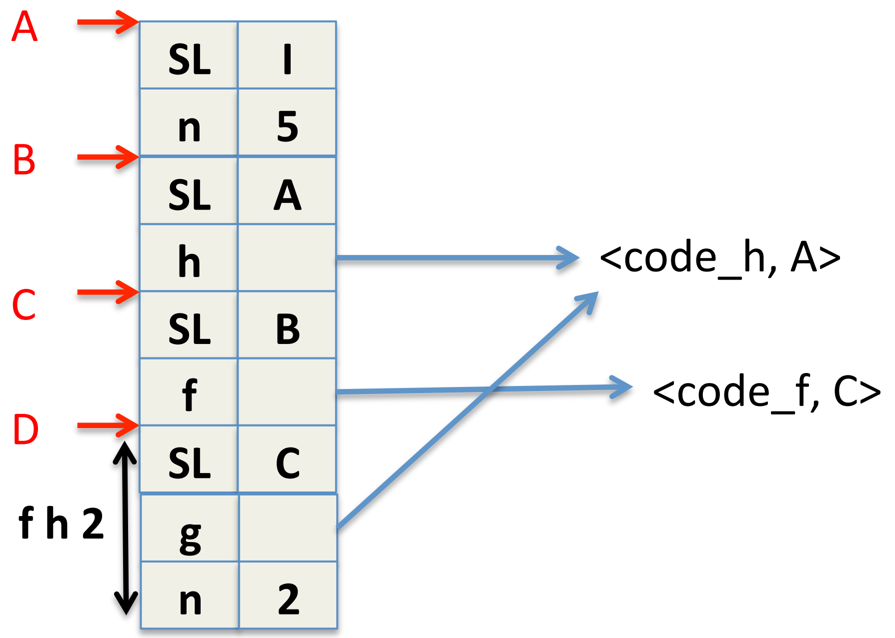
```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack



```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack:simulation



Env_A(n) = 5
Env_A(m) = unbond
for all m != n

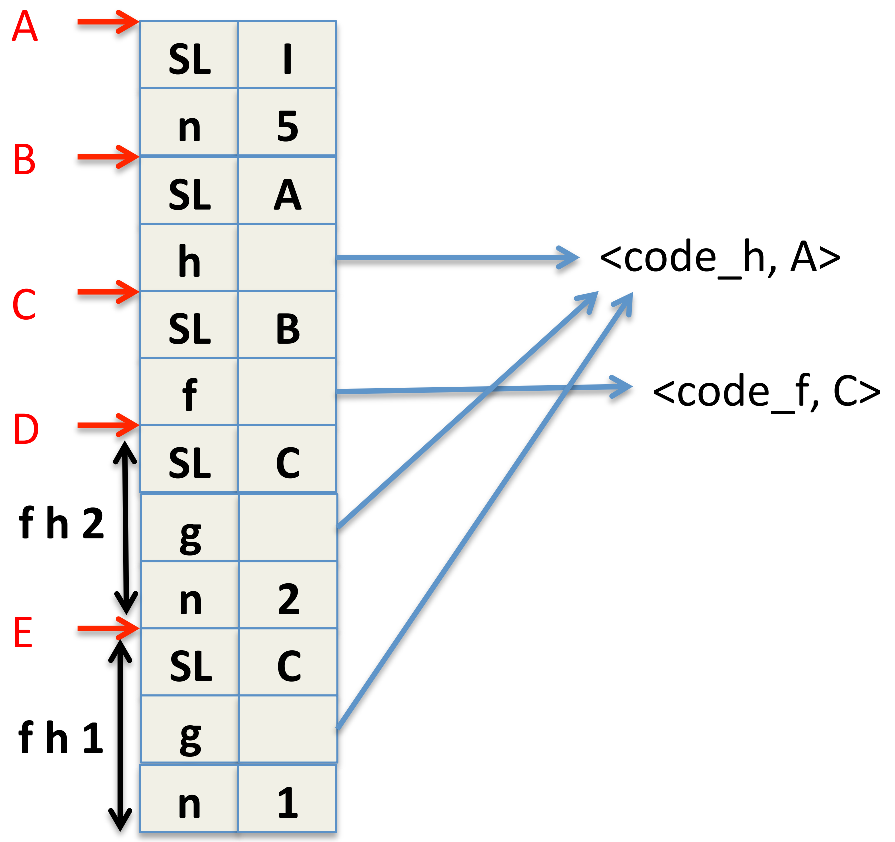
Env_B (n) = 5
Env_B(h) = <code_h, Env_A>

Env_C(f) = <code_f, Env_C>
Env_C(h) <code_h, Env_A>
Env_C(n) = 5

Env_D(g) = <code_h, Env_A>
Env_D(n) = 2
Env_D(f) = <code_f, Env_C>
Env_D(h) <code_h, Env_A>

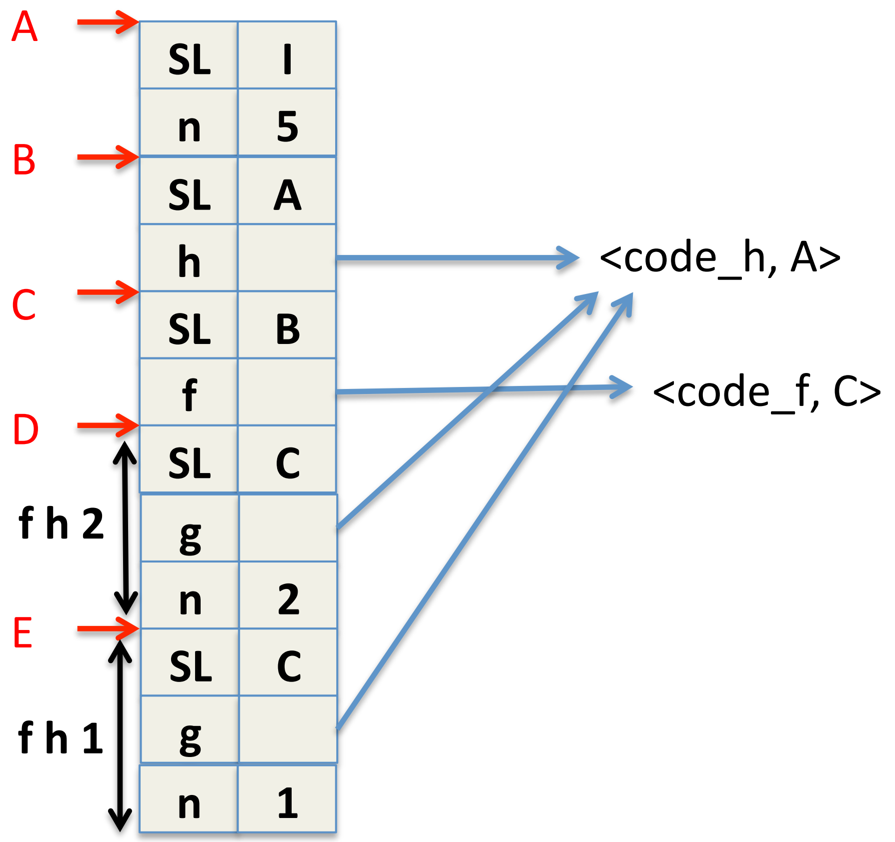
```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack



```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Run-time Stack: simulation



```
let  n = 5;;
let h = fun x -> n + x ;;
let rec f g n = if n = 1 then g(n) else n * f g (n-1);;
f h 2;;
```

Env_A(n) = 5
Env_A(m) = unbond
for all m != n

Env_B (n) = 5
Env_B(h) = <code_h, Env_A>

Env_C(f) = <code_f, Env_C>
Env_C(h) <code_h, Env_A>
Env_C(n) = 5

Env_D(g) = <code_h, Env_A>
Env_D(n) = 2
Env_D(f) = <code_f, Env_C>
Env_D(h) <code_h, Env_A>

Env_E(g) = <code_h, Env_A>
Env_E(n) = 1
Env_E(f) = <code_f, Env_C>
Env_E(h) <code_h, Env_A>

Ricorsione

Una simulazioen alternativa

Definizione ricorsiva

$$env \triangleright \mathrm{Re}\,c(i,e) \Rightarrow Funval(\mathrm{Re}\,c(i,e),\ env)$$

```
Rec(i, e) -> Funval(Rec(i,e), r))
```

REGOLA APPLICAZIONE

$$\frac{\begin{array}{c} env \triangleright e1 => v1 \quad v1 =< \mathrm{Rec}(f, Fun(x,e)), env' > \\ env \triangleright e2 => v2 \\ env[v2/x][v1/f] \triangleright e => v \end{array}}{env \triangleright Apply(e1,e2) => v}$$

Ocaml

```
App(e1, e2) -> let e1' = sem e1 env
       in (match e1' with
         | Funval(Fun(x, e), env') -> sem e (bindx (eval e2 env) env')
         | Funval(Rec(f, Fun(x,e)), env') ->
                    sem e (bind f e1' (bind x (eval e2 env) env'))
         | _ -> raise (TypeError)
```