ALOCAÇÃO DE MEMÓRIA

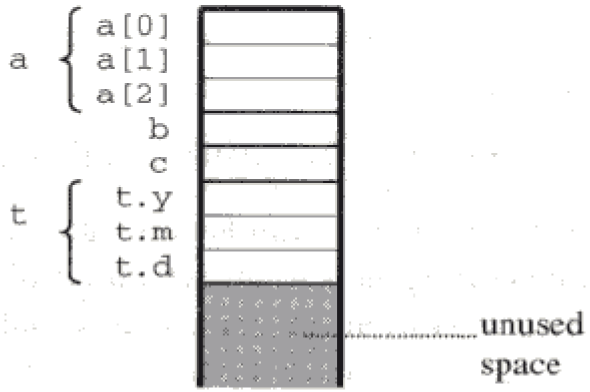
Baseado no Capítulo 6 de Programming Language Processors in Java, de Watt & Brown

Última modificação em 10/05/2024 09:10:15

Alocação Estática

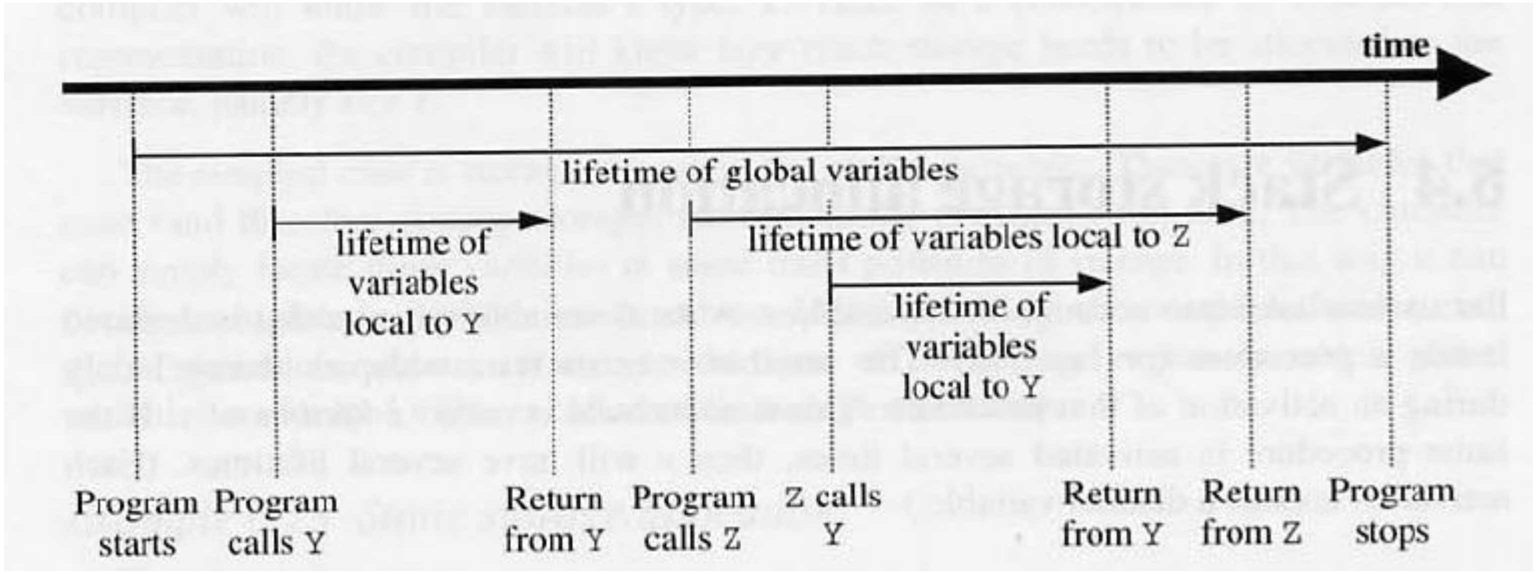
```
let
   type Date = record
                    y: Integer,
                    m: Integer,
                    d: Integer
                 end;
   var a: array 3 of Integer;
   var b: Boolean;
   var c: Char;
   var t: Date
in
   ...
```

address[[a]] = 0
address[[b]] = 3
address[[c]] = 4
address[[t]] = 5

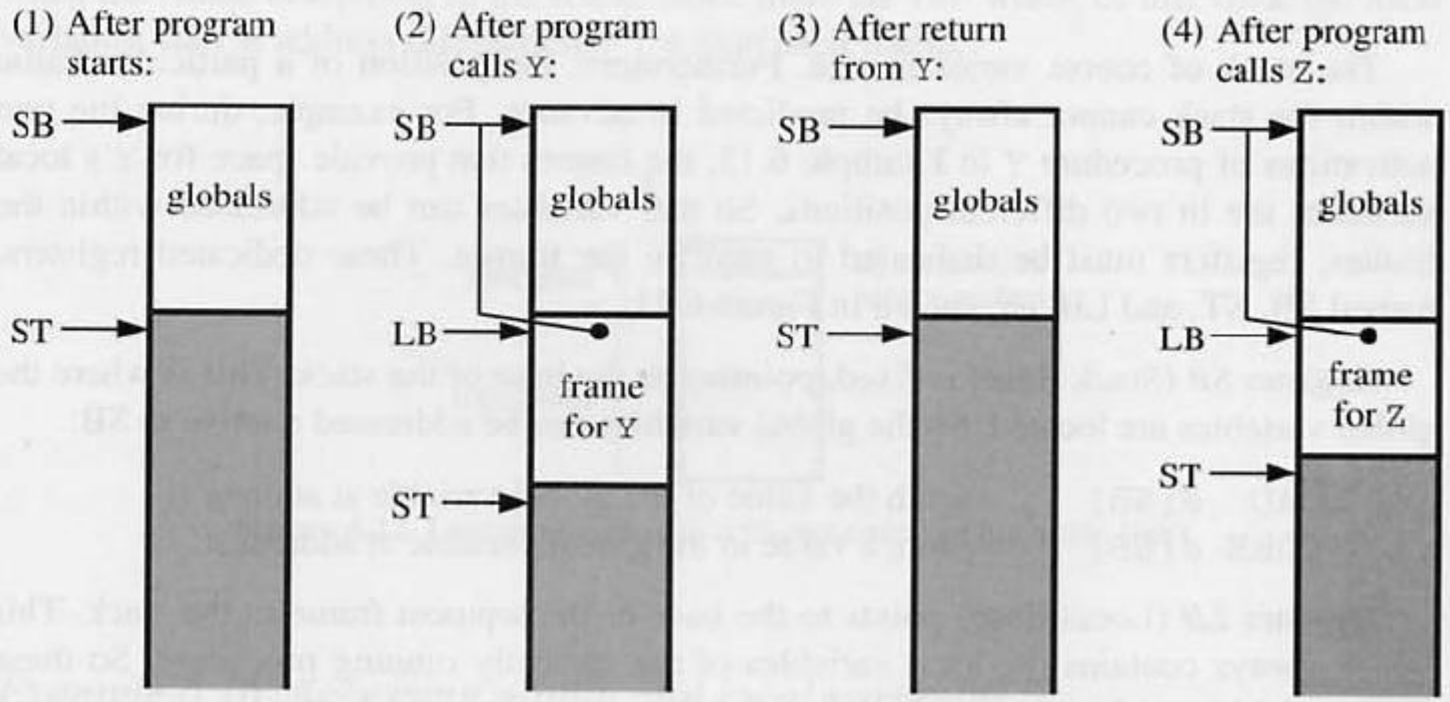


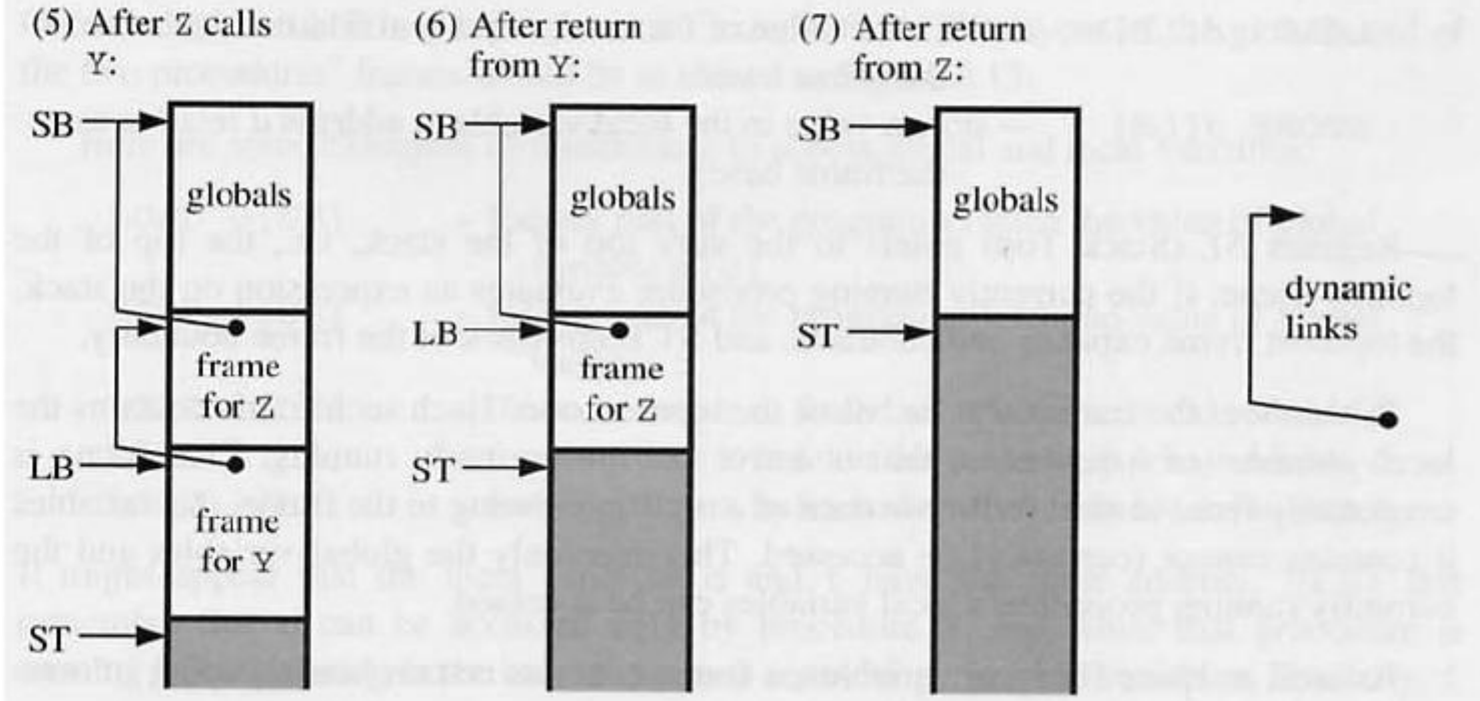
Alocação Automática (stack)

```
let
   var a: array 3 of Integer;
   var b: Boolean;
   var c: Char;
   proc Y () ~
      let
         var d: Integer;
         var e: record c: Char, n: Integer end
      in
         ...;
   proc Z () ~
      let
         var f: Integer
      in
         begin ...; Y(); ... end
in
   begin ...; Y(); ...; Z(); ... end
```


time
lifetime of global variables
lifetime of variables local to Y
lifetime of variables local to Z
lifetime of variables local to Y
Program starts
Program calls Y
Return from Y
Program calls Z
Z calls Y
Return from Y
Return from Z
Program stops

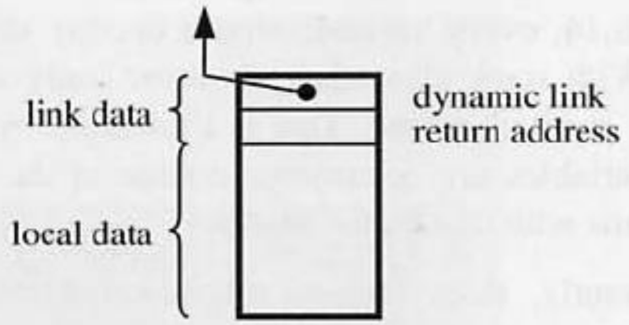
Variáveis globais
e
locais


(1) After program starts:
SB
globals
ST
(2) After program calls Y:
SB
globals
LB
frame for Y
ST
(3) After return from Y:
SB
globals
ST
(4) After program calls Z:
SB
globals
LB
frame for Z
ST

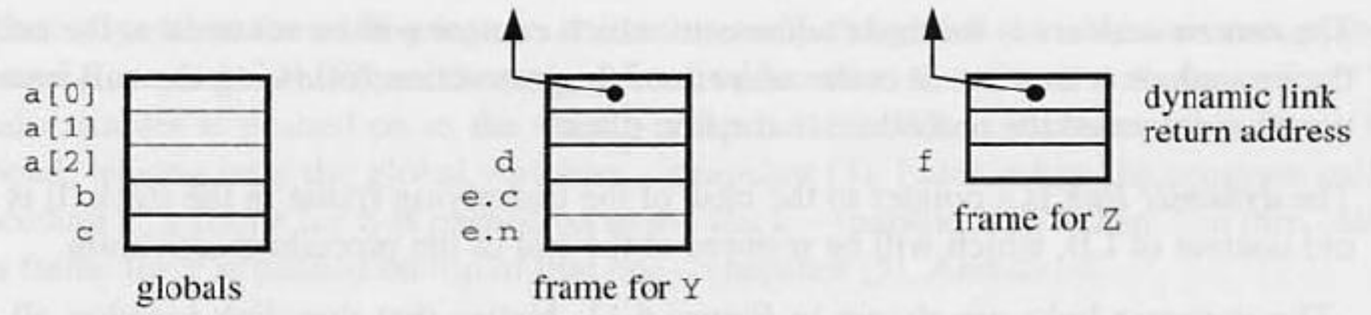

(5) After Z calls Y:
SB
globals
frame for Z
LB
frame for Y
ST
(6) After return from Y:
SB
globals
LB
frame for Z
ST
(7) After return from Z:
SB
globals
ST
dynamic links

`LOAD` *d*`[SB]`	– fetch the value of the global variable at address *d*.
`STORE` *d*`[SB]`	– store a value in the global variable at address *d*.

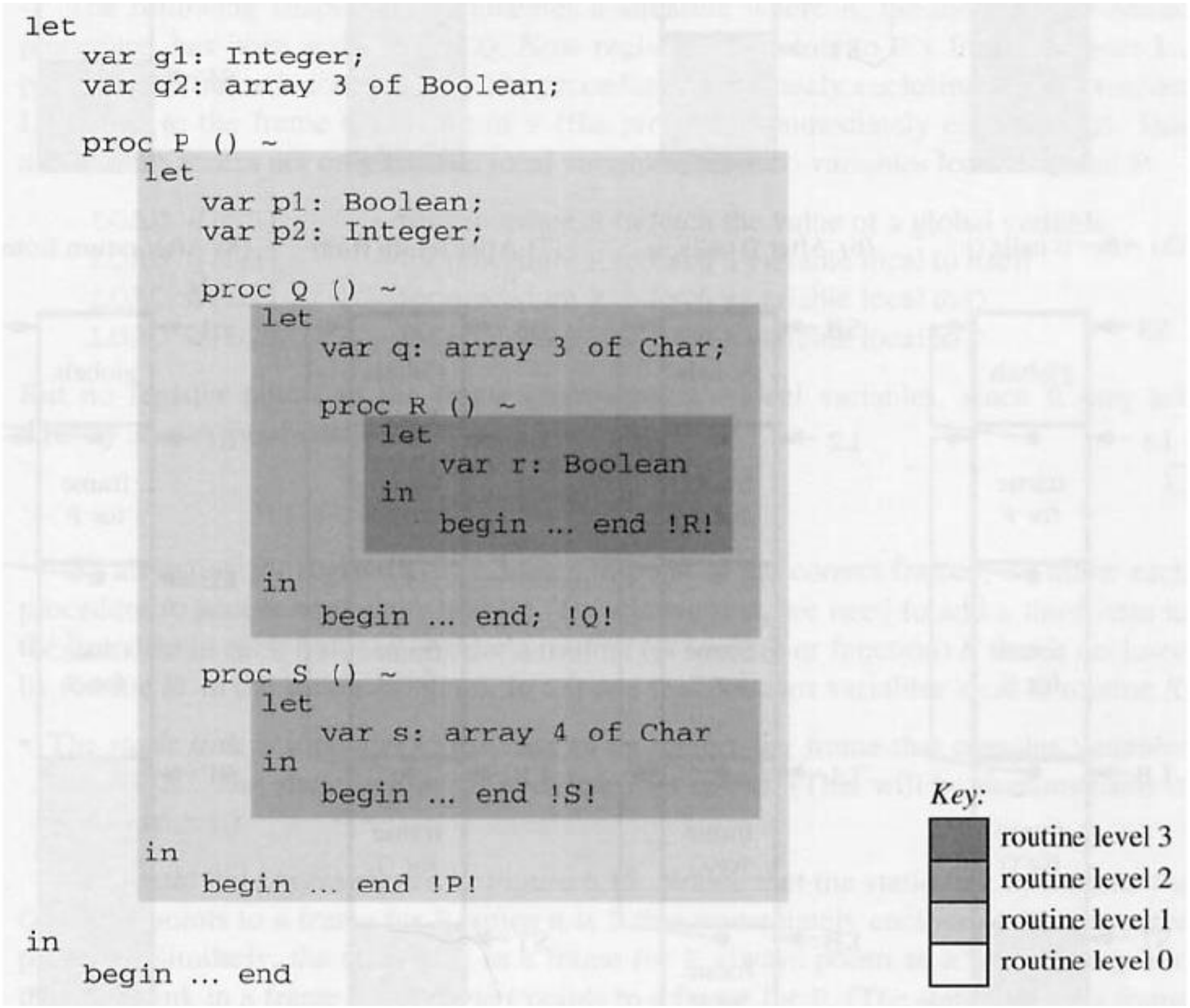
`LOAD` *d*`[LB]`	– fetch the value of the local variable at address *d* relative to the frame base.
`STORE` *d*`[LB]`	– store a value in the local variable at address *d* relative to the frame base.



`LOAD 0[SB]`	– for any part of the program to fetch the value of global variable `a[0]`
`LOAD 4[SB]`	– for any part of the program to fetch the value of global variable `c`
`LOAD 2[LB]`	– for procedure `Y` to fetch the value of its local variable `d`
`LOAD 4[LB]`	– for procedure `Y` to fetch the value of its local variable `e.n`
`LOAD 2[LB]`	– for procedure `Z` to fetch the value of its local variable `f`



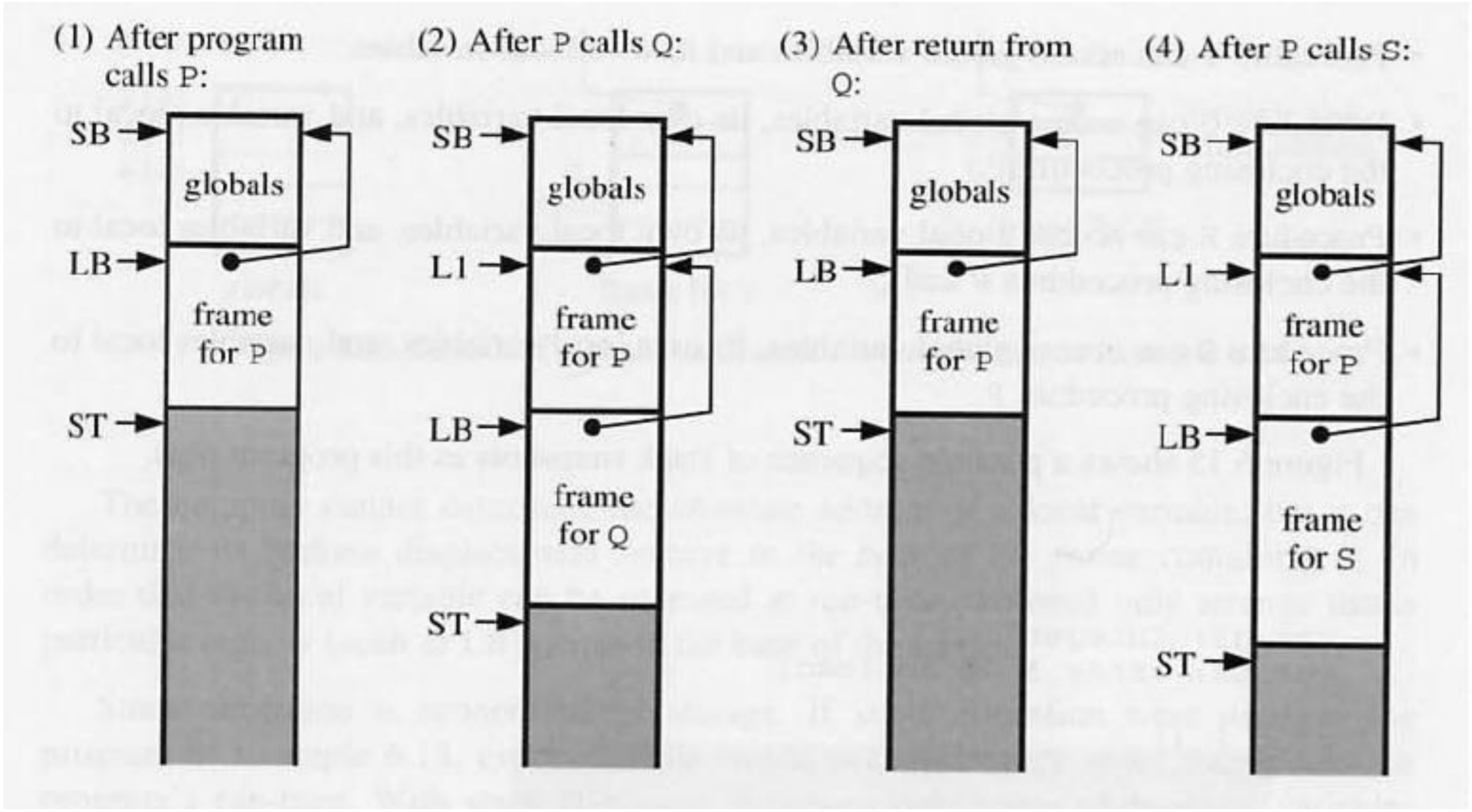
Variáveis não-locais

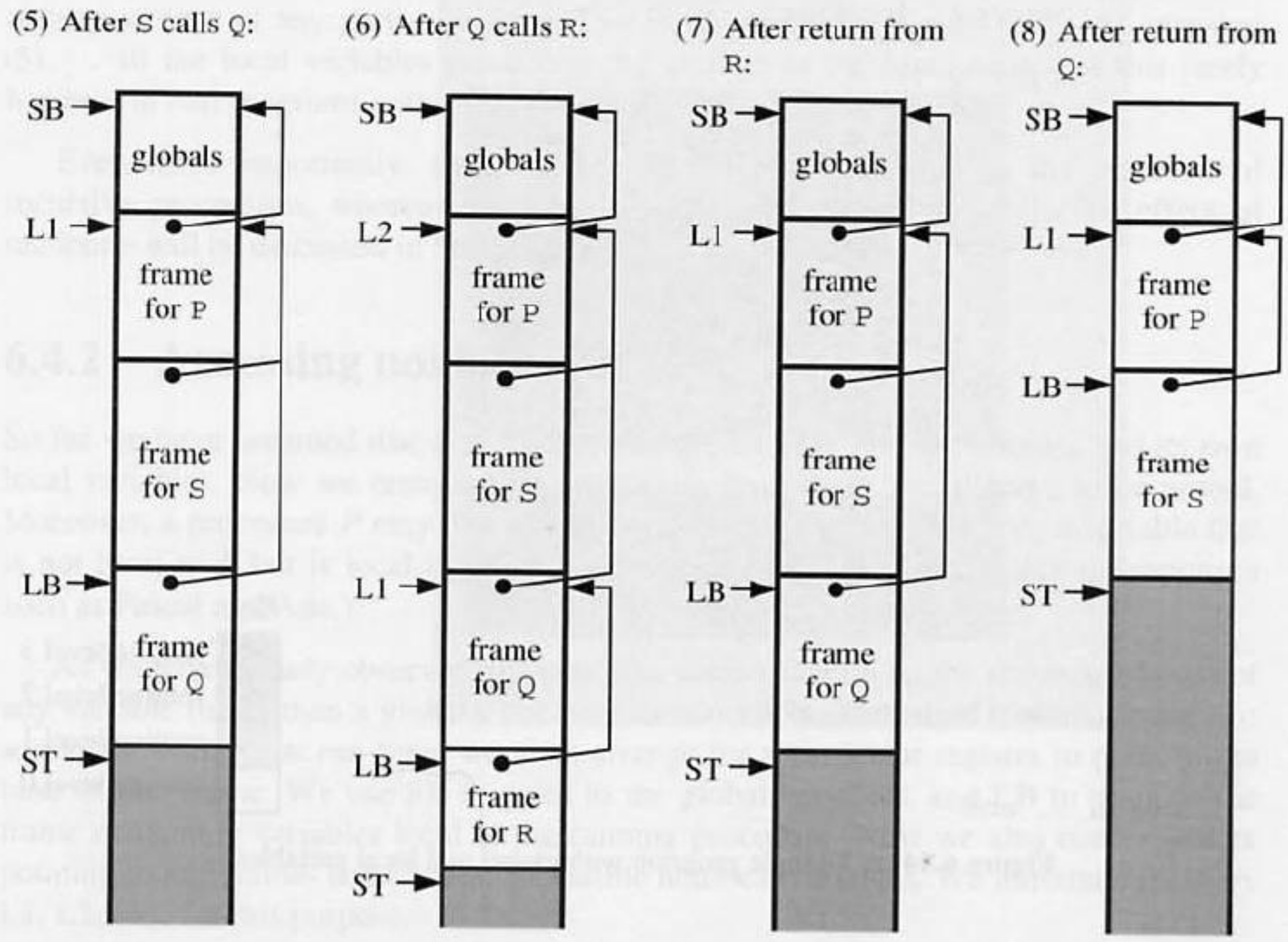


```
let
   var g1: Integer;
   var g2: array 3 of Boolean;

   proc P () ~
      let
         var p1: Boolean;
         var p2: Integer;

         proc Q () ~
            let
               var q: array 3 of Char;

               proc R () ~
                  let
                     var r: Boolean
                  in
                     begin ... end !R!
            in
               begin ... end; !Q!

         proc S () ~
            let
               var s: array 4 of Char
            in
               begin ... end !S!
      in
         begin ... end !P!
in
   begin ... end
```

Key:

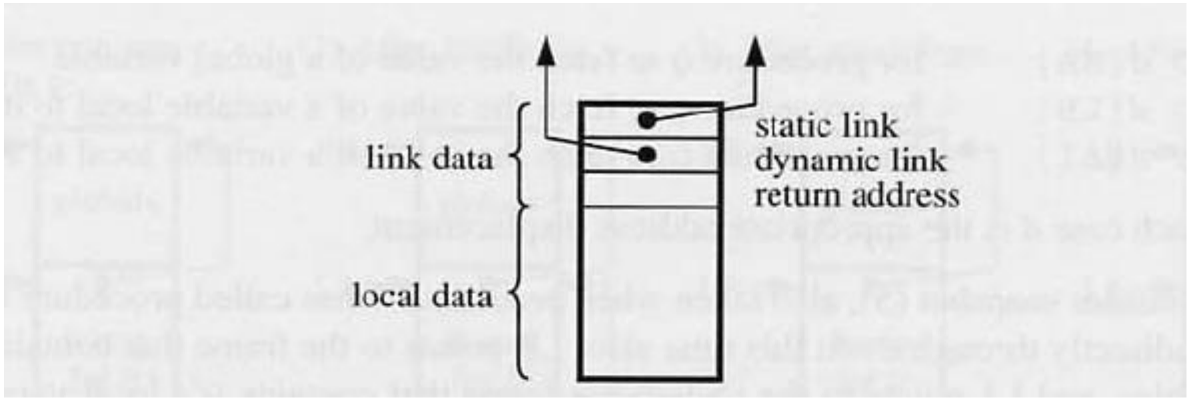
▨	routine level 3
▨	routine level 2
▨	routine level 1
□	routine level 0

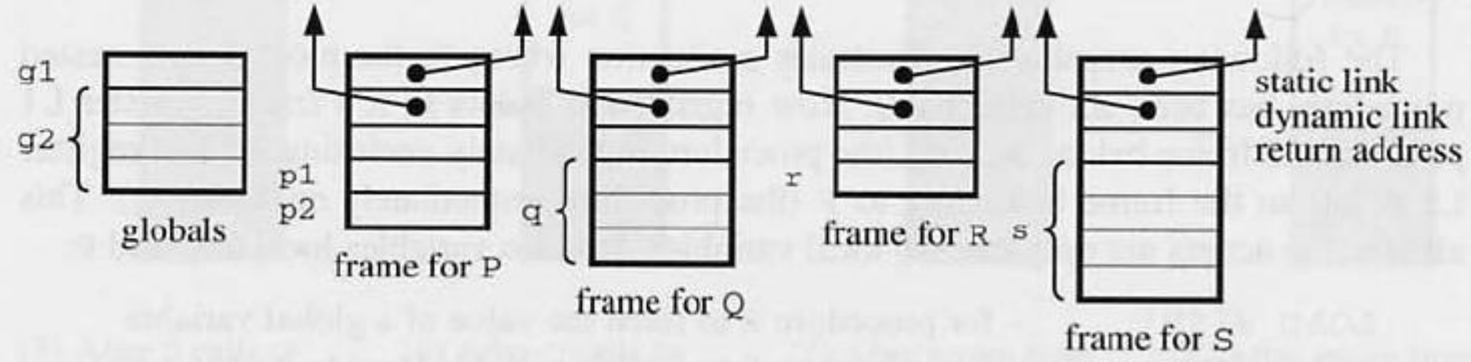

(1) After program calls P:
(2) After P calls Q:
(3) After return from Q:
(4) After P calls S:
SB
LB
L1
ST
globals
frame for P
frame for Q
frame for S


(5) After S calls Q:
(6) After Q calls R:
(7) After return from R:
(8) After return from Q:
SB
L1
L2
LB
ST
globals
frame for P
frame for S
frame for Q
frame for R

`LOAD` *d*`[SB]`	– for procedure `Q` to fetch the value of a global variable
`LOAD` *d*`[LB]`	– for procedure `Q` to fetch the value of a variable local to itself
`LOAD` *d*`[L1]`	– for procedure `Q` to fetch the value of a variable local to `P`

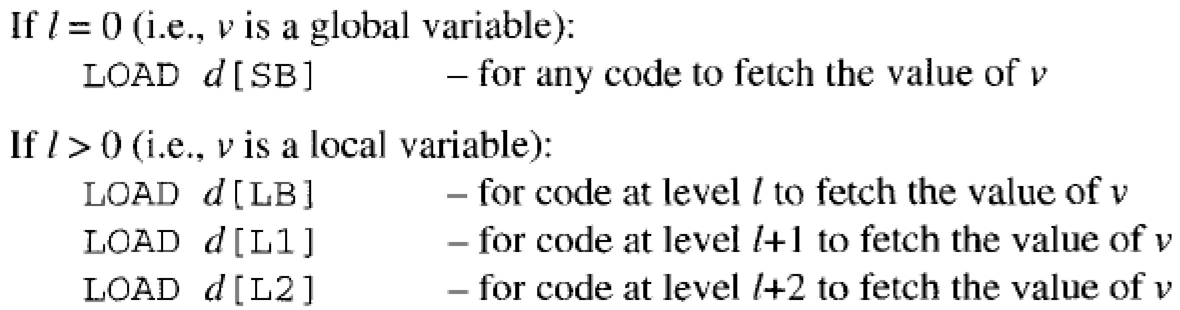
`LOAD` *d*`[SB]`	– for procedure `R` to fetch the value of a global variable
`LOAD` *d*`[LB]`	– for procedure `R` to fetch a variable local to itself
`LOAD` *d*`[L1]`	– for procedure `R` to fetch a variable local to `Q`
`LOAD` *d*`[L2]`	– for procedure `R` to fetch a variable local to `P`





$$\begin{aligned}
\text{L1} &= \mathit{content}(\text{LB}) \\
\text{L2} &= \mathit{content}(\text{L1}) = \mathit{content}(\mathit{content}(\text{LB})) \\
\text{L3} &= \mathit{content}(\text{L2}) = \mathit{content}(\mathit{content}(\mathit{content}(\text{LB})))
\end{aligned}$$

Endereçamento de variáveis:



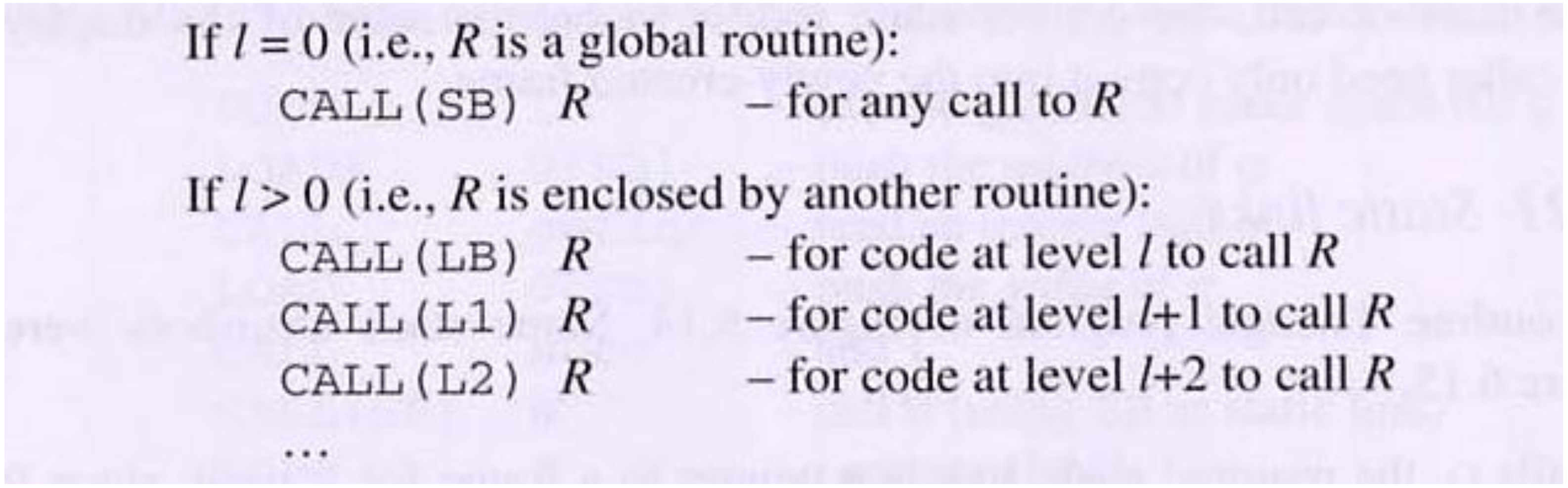
If $l = 0$ (i.e., v is a global variable):

`LOAD` d`[SB]`	– for any code to fetch the value of v

If $l > 0$ (i.e., v is a local variable):

`LOAD` d`[LB]`	– for code at level l to fetch the value of v
`LOAD` d`[L1]`	– for code at level l+1 to fetch the value of v
`LOAD` d`[L2]`	– for code at level l+2 to fetch the value of v

Cálculo do link estático:



If $l = 0$ (i.e., *R* is a global routine):

```
    CALL(SB) R          – for any call to R
```

If $l > 0$ (i.e., *R* is enclosed by another routine):

```
    CALL(LB) R          – for code at level l to call R
    CALL(L1) R          – for code at level l+1 to call R
    CALL(L2) R          – for code at level l+2 to call R
    ...
```

Cálculo do link estático:

Sejam

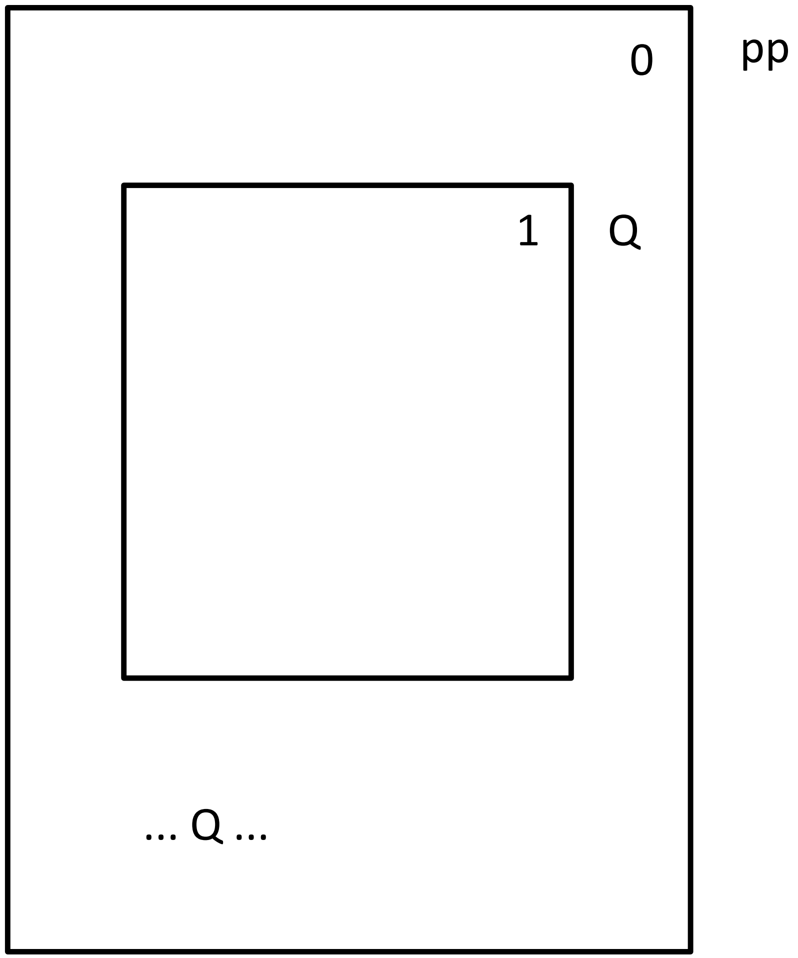
- l_1 nível em que o bloco foi declarado
- l_2 nível em que ocorre a chamado bloco

Então:

Se l_1=0 (bloco declarado no nível global) usar SB *(caso 1)*
Se l_1 >0 (bloco declarado dentro de outro bloco)

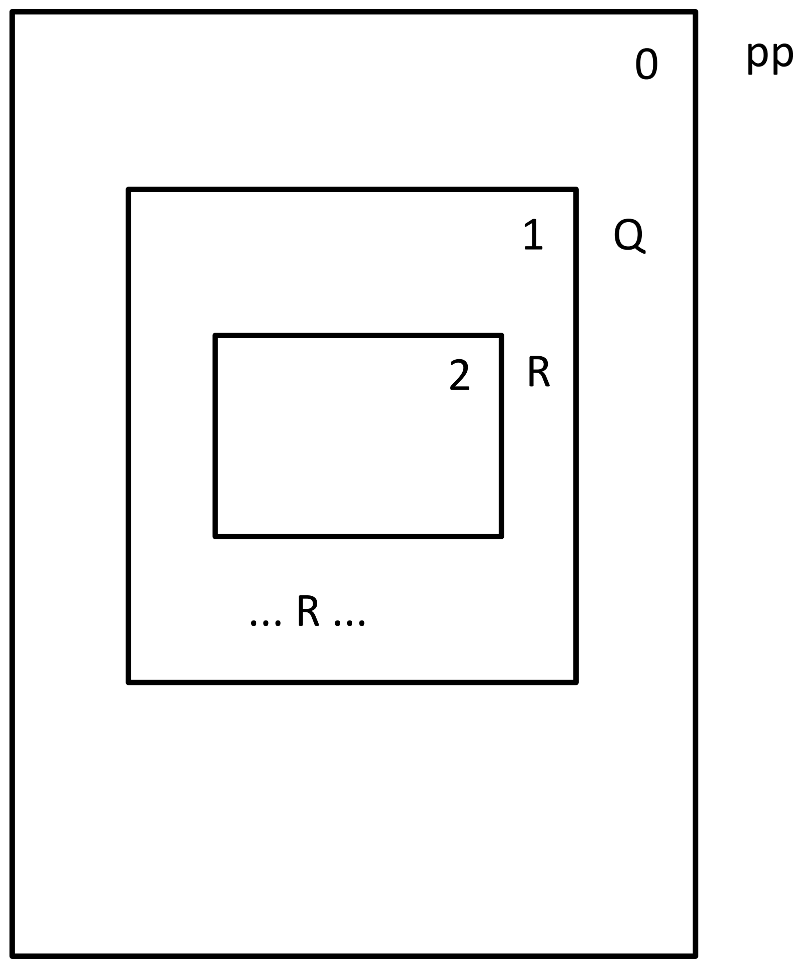
    se l_2-l_1=0 usar LB *(caso 2)*
    se l_2-l_1=1 usar L1 *(caso 3)*
    se l_2-l_1=2 usar L2 *(caso 4)*
    etc

Caso 1:



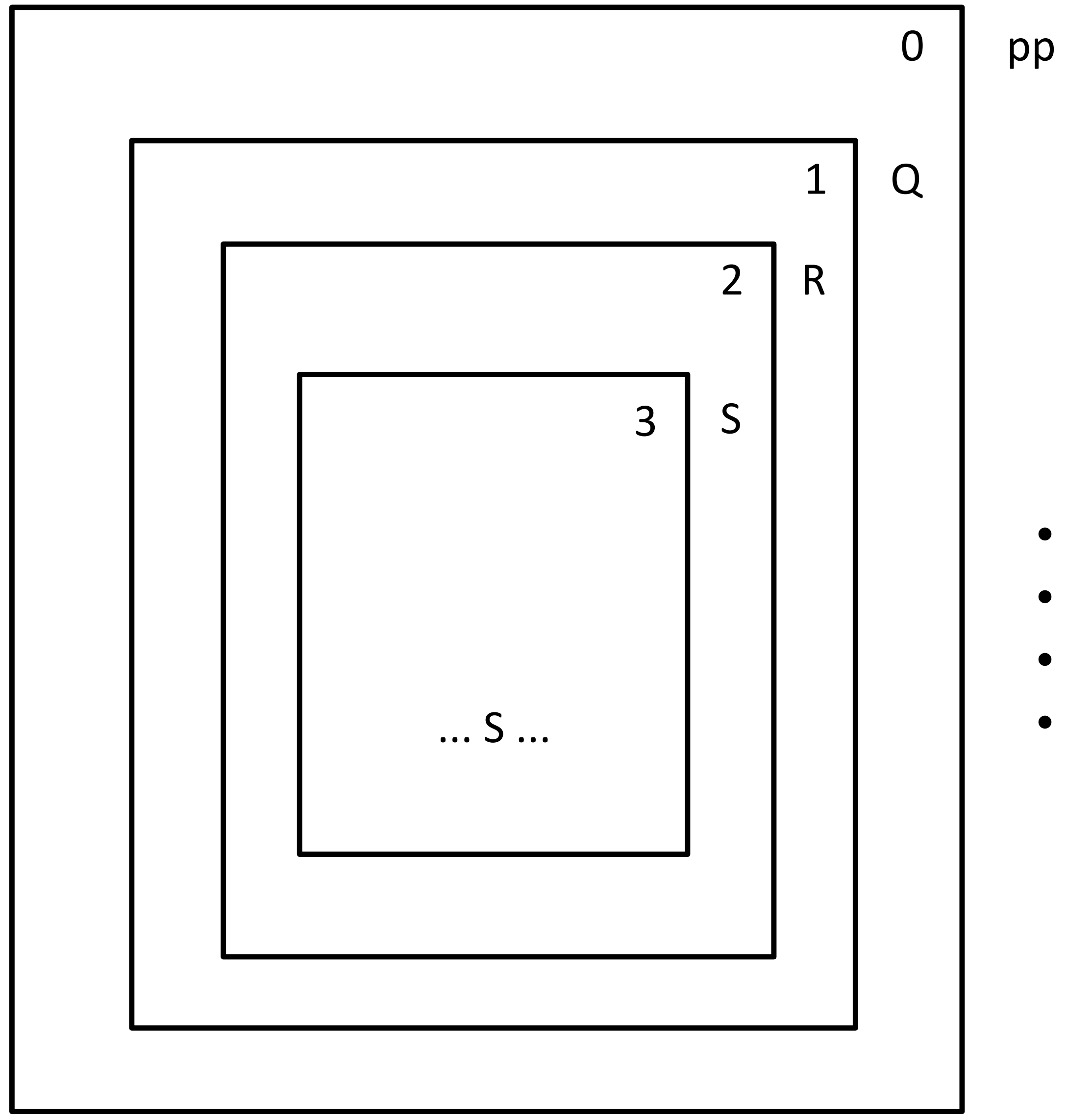
- Q está declarado no nível 0.
- Usar SB como LE

Caso 2:



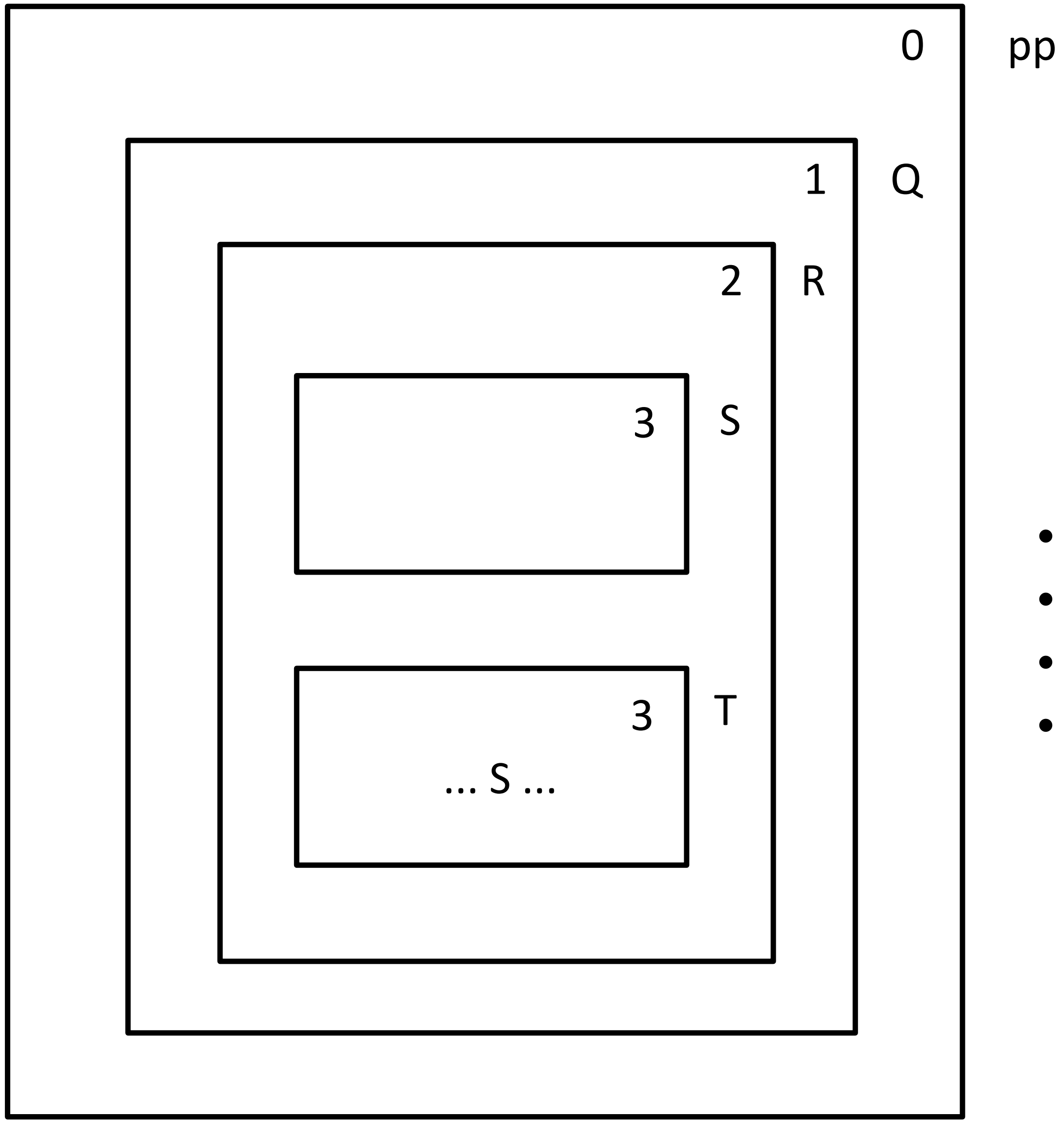
- Q define o nível 1.
- R está declarado no nível 1.
- 1-1=0.
- Usar LB como LE.

Caso 3, primeiro exemplo:



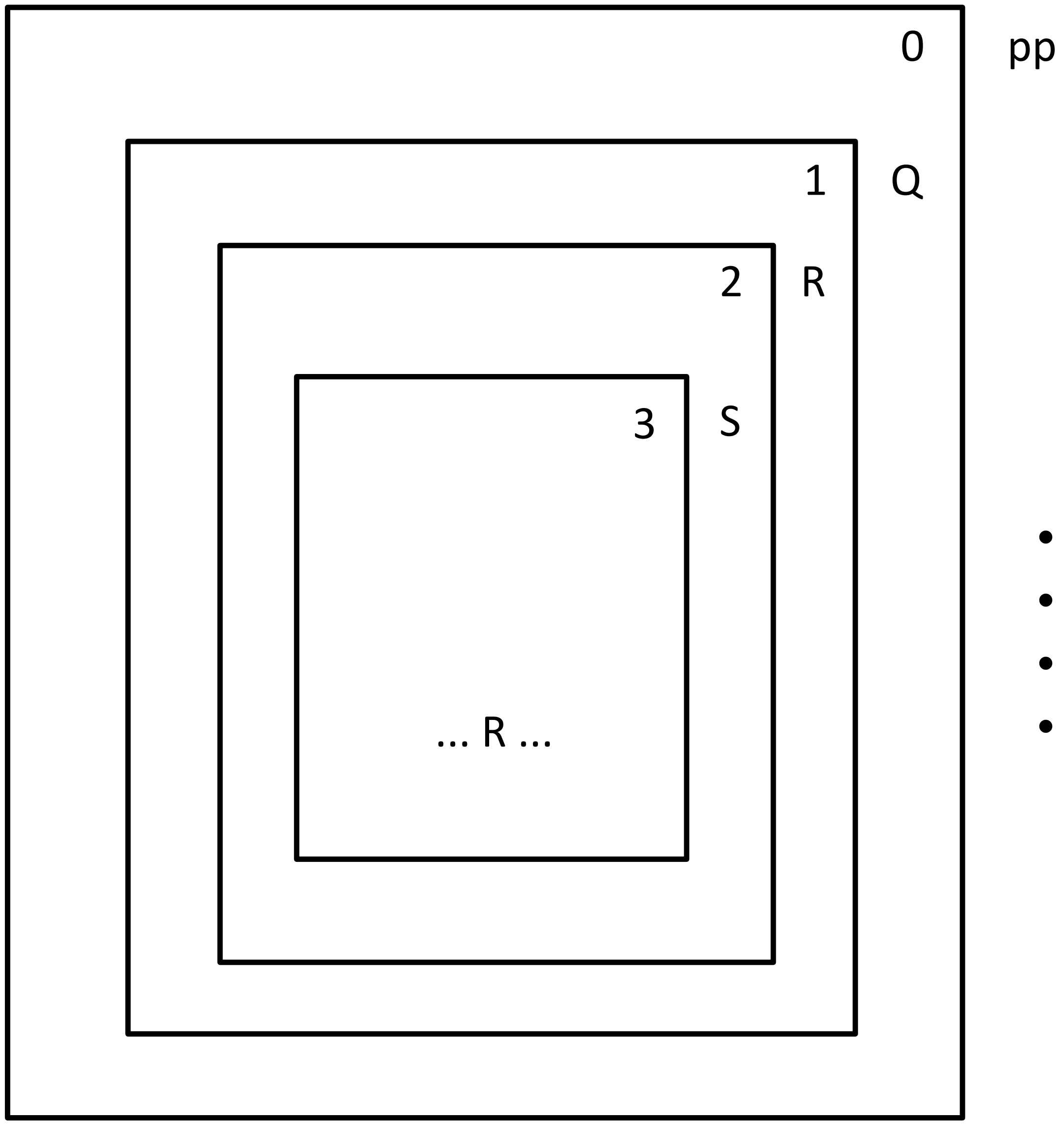
- S define o nível 3.
- S está declarado no nível 2.
- 3-2=1.
- Usar L1 como LE.

Caso 3, segundo exemplo:



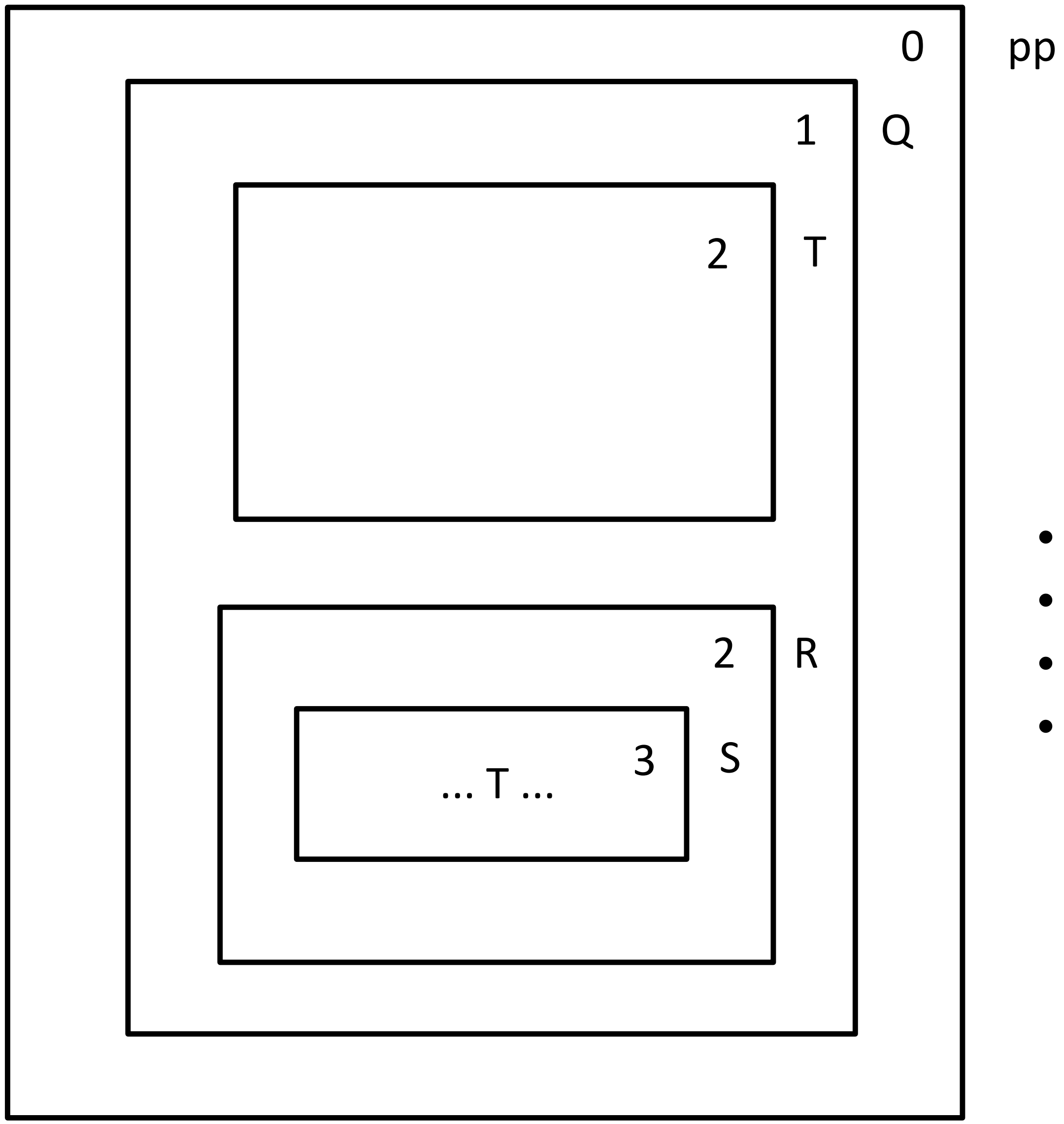
- T define o nível 3.
- S está declarado no nível 2.
- 3-2=1.
- Usar L1 como LE.

Caso 4, primeiro exemplo:



- S define o nível 3.
- R está declarado no nível 1.
- 3-1=2.
- Usar L2 como LE.

Caso 4, segundo exemplo:



- S define o nível 3.
- T está declarado no nível 1.
- 3-1=2.
- Usar L2 como LE.