



第九章 运行时存储空间组织



问题

- 程序的静态文本和程序运行时活动联系起来
- 程序运行需要记录和保存哪些信息？
- 源代码中的变量、常量等如何存放、如何访问？




内容线索

- **目标程序运行时的活动**
- **运行时存储器的划分**
- **静态存储分配**
- **简单的栈式存储分配**
- **嵌套过程语言的栈式实现**




过程的活动

- **过程定义：过程名 + 过程体**
- **过程调用：过程名出现在可执行语句中。**
- **过程的活动：过程的一次执行。**
- **活动的生存期：执行过程体第一步操作到最后一步操作之间的操作序列，包括该过程调用其它过程花费的时间。**



```
(1)  program sort(input, output)
(2)    var a: array[0..10] of integer;
(3)    procedure readarray;
(4)      var i: integer;
(5)    begin
(6)      for i:=1 to 9 do read(a[i])
(7)    end;
(8)    function partition(y, z:integer):integer;
(9)      var i:integer;
(10)   begin .....
(11)   end;
```

```
program sort
  procedure readarray
  function partition
  procedure quicksort
```



```
(12)  procedure quicksort(m, n:integer);
(13)      var i:integer;
(14)      begin
(15)          if (n>m) then begin
(16)              i:=partition(m, n );
(17)              quicksort(m, i-1 );
(18)              quicksort(i+1, n )
(19)          end;
(20)      end;
(21)  begin
(22)      a[0]:=-9999; a[10]:=9999;
(23)      readarray;
(24)      quicksort(1, 9 )
(25)  end.
```

program sort

procedure readarray

function partition

procedure quicksort



过程的活动

- 每次控制从过程P进入过程Q后，如果没有错误，最后都返回到过程P。
 - 对于非递归的两个过程，如果a和b都是过程的活动，那么它们的生存期或者是不重叠的，或者是嵌套的。
- 如果一个过程是递归的，那么在某一时刻可能有这个过程的几个活动活跃着。
- 编译程序组织存储空间时，要考虑激活一个过程的新活动时或从过程的活动返回时，对局部名的处理，及是否允许静态或动态存储分配等。



过程调用

- 过程（函数）是结构化程序设计的主要手段，同时也是节省程序代码和扩充语言能力的主要途径。

- 过程定义：

```
procedure add(x,y:integer; var z:integer)
begin
    z: =x+y;
end;
```

- 过程调用

```
add(a,b,c);
```




参数传递

- 调用和被调用过程之间的信息是通过**参数**（或全局量）来传递的。称之为**形式参数**与**实在参数**。

- 传地址
- 得结果
- 传值
- 传名



传地址

- 把**实在参数的地址**传递给相应的**形式参数**，形式参数在过程段有相应的形式单元存放相应的实在参数的地址。
- 方法：
 - 调用段预先把实在参数的地址传递到被调用段可以拿到的地方；
 - 程序控制转入被调用段之后，被调用段首先把实在参数的地址抄进自己相应的形式单元中；
 - 过程体对形式参数的引用或赋值被处理成对形式单元的间接访问。

例. 对如下程序

```
procedure P(x,y,z)
begin
```

```
    y:=y+1;
```

```
    z:=z+x;
```

```
end
```

```
program M(input, output)
```

```
begin
```

```
    A:=2;B:=3;
```

```
    P(A+B,A,A);
```

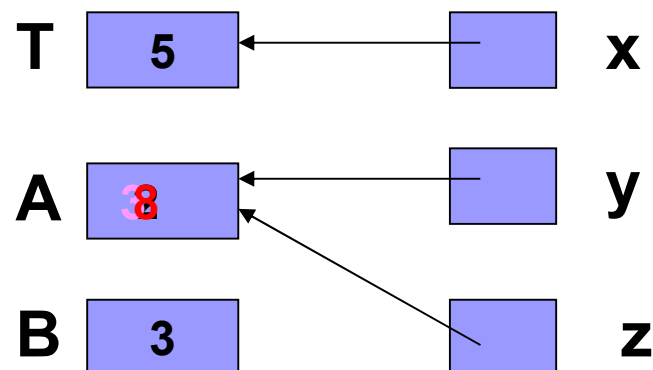
```
    print A;
```

```
end
```

传地址方式下的程序输出结果。

实参单元

形参单元



结果输出A=8



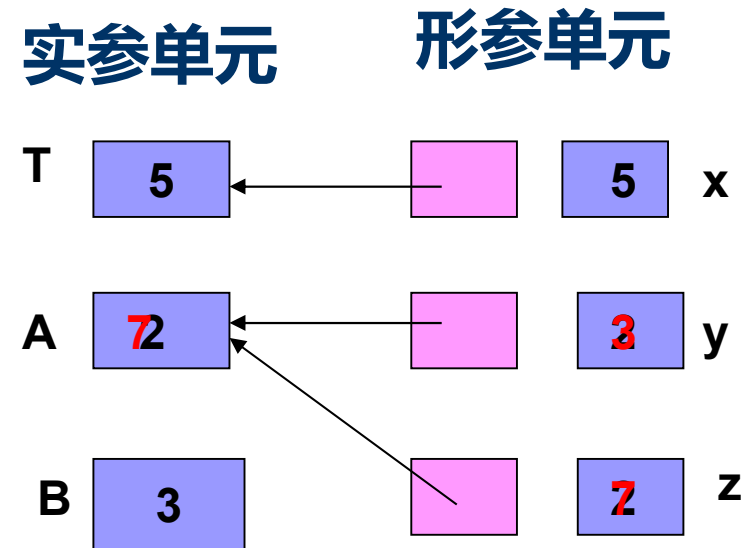
得结果

- 传地址的一种变形，但不等价
- 方法：
 - 每个形参对应**两个形式单元**，第一个形式单元存放实参地址，第二个单元存放实参的值。
 - 在过程体中对形式参数的任何引用或赋值都看作对它的第二个单元的直接访问。
 - 过程完成返回前把第二个单元的内容存放到第一个单元所指的实参单元中。

例：对如下程序

```
procedure P(x,y,z)
begin
    y:=y+1;
    z:=z+x;
end
program M(input,
output)
begin
    A:=2;B:=3;
    P(A+B,A,A);
    print A;
end
```

得结果方式下的程序输出结果。



结果输出A=7



传 值

- 把实在参数的值传递给相应的形式参数
- 方法：
 - 调用段预先把实在参数的的值计算出来并放在被调用段可以拿到的地方；
 - 被调用段开始工作时，首先把实参的值抄入形式参数相应的单元；
 - 被调用段中，象引用局部数据一样引用形式单元。

例. 对如下程序

```
procedure P(x,y,z)
begin
    y:=y+1;
    z:=z+x;
end
program M(input,
output)
begin
    A:=2;B:=3;
    P(A+B,A,A);
    print A;
end
```

传值方式下的程序输出结果。

实参单元

形参单元

T

5

5

x

A

2

3

y

B

3

7

z

结果输出A=2



传名

- 过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，但把其中任一出现的形式参数都替换成相应的实参。
- 如果在替换时发现过程体中的局部名和实在参数中的名字使用相同的标识符,则必须用不同标识符表示这些局部名
- 方法：
 - 在进入被调用段的之前不对实在参数预先进行计值，而是让过程体中每当使用到相应的形式参数时才逐次对它实行计值（或计算地址）。因此，通常把实在参数处理成一个子程序（称为参数子程序），每当过程体中使用到相应的形式参数时就调用这个子程序。



例：对如下程序

procedure P(x,y,z)

begin

y:=y+1;

z:=z+x;

end

program M(input, output)

begin

A:=2;B:=3;

P(A+B,A,A);

print A;

end

传名方式下的程序输出结果。

主程序替换为：

beginA:=2; B=3;

A:=A+1;

A:=A+(A+B);

print A;

end

结果输出A=9

PROGRAM EX

...

var A:integer;

PROCEDURE P(B:integer)

...

var A:integer;

BEGIN

A:=0;

B:=B+1;

A:=A+B;

END;

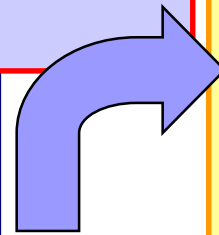
BEGIN

A:=2;

P(A);

write(A);

END



BEGIN

A :=2;

TA:=0;

A:= A +1;

TA:=TA+ A;

write(A);

END



例.

...

```
procedure P(w,x,y,z);  
begin  
    y := y*w;  
    z := z+x;  
end  
begin  
    a := 5;  
    b := 3;  
    P(a+b,a-b,a,a);  
    write(a);  
end
```

传值: 5

传地址: 42

得结果: 7

传名: 77



C/C++ 中的参数传递

- 值传递
- 地址传递
- 引用传递
- inline函数



名字的作用范围

- 在许多程序语言中,名字都有一个确定的作用范围.
- 两种程序体结构
 - 单层（并列）结构，如FORTRAN
 - 一个FORTRAN程序由一个主程序段和若干个辅程序段组成
 - 多层（嵌套）结构，如PASCAL，ADA
 - 过程可以嵌套和递归



作用域

- **作用域**：一个名字能被使用的区域范围称作这个名字的作用域。
- 允许同一个标识符在不同的过程中代表不同的名字。
- 名字作用域规则——“最近嵌套原则”
 - 一个在子程序B1中说明的名字X只在B1中有效（局部于B1）；
 - 如果B2是B1的一个内层子程序且B2中对标识符X没有新的说明，则原来的名字X在B2中仍然有效。如果B2对X重新作了说明，那么，B2对X的任何引用都是指重新说明过的这个X。

```

program main
  var A, B : real;
  ...
  procedure P1
    var B:boolean;
    ...
  begin
    ...
  end
  procedure P2
    var A:integer;
    ...
  begin
    ...
  end
begin
  ...
end

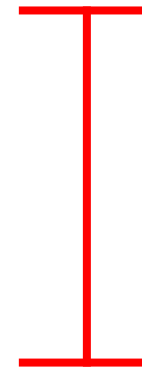
```

A(real)

B(real)

B(bool)

A(integr)





小结

- 编译程序为了组织存储空间，必须考虑下面几个问题：
 - 过程是否允许递归？
 - 当控制从一个过程的活动返回时，对局部名称的值如何处理？
 - 过程是否允许引用非局部名称？
 - 过程调用时如何传递参数；过程是否可以做为参数被传递和做为结果被返回？
 - 存储空间可否在程序控制下进行动态分配？
 - 存储空间是否必须显式地释放？



内容线索

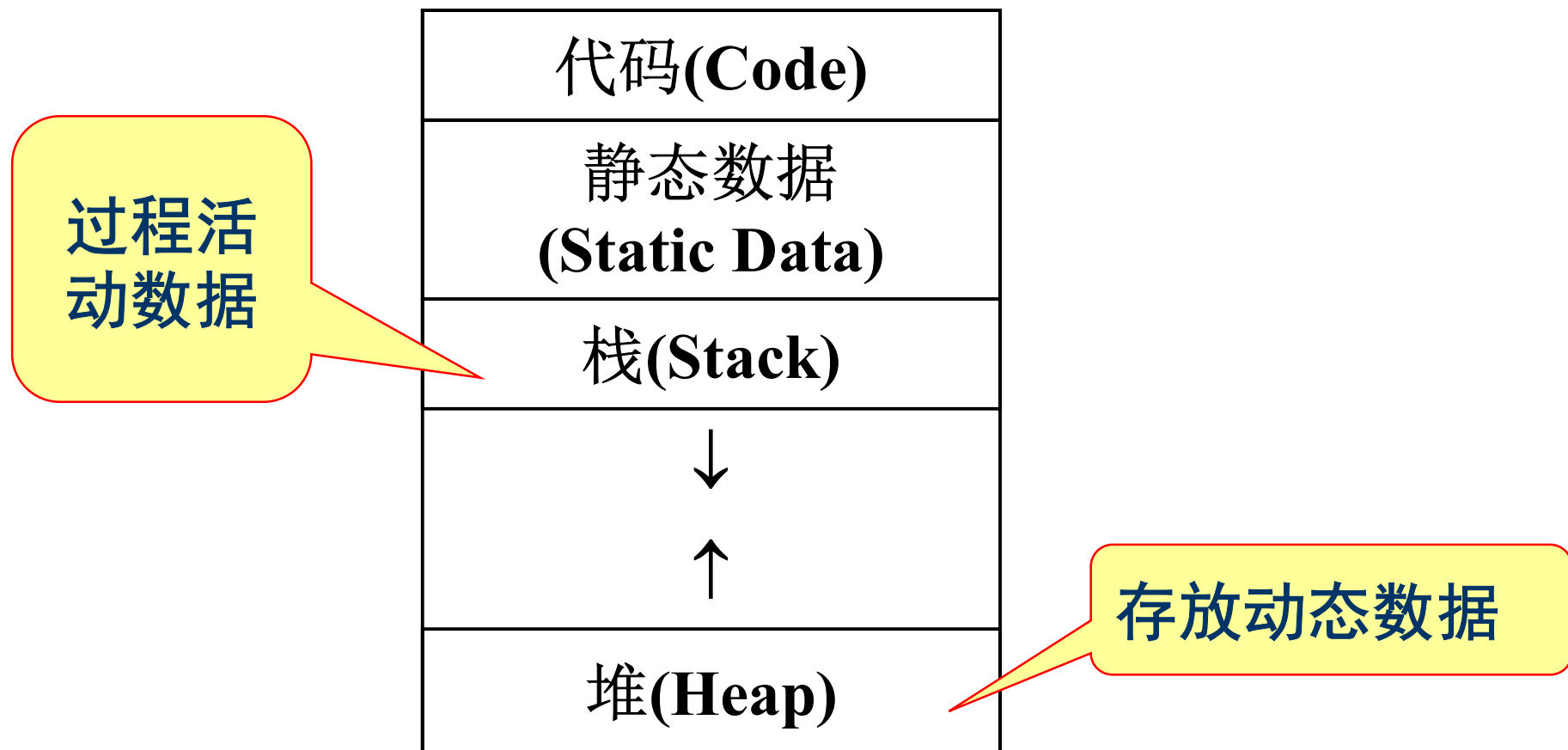
- ✓ 目标程序运行时的活动
- 运行时存储器的划分
- 静态存储分配
- 简单的栈式存储分配
- 嵌套过程语言的栈式实现



存储空间需求

- **一个目标程序运行所需的存储空间包括：**
 - **存放目标代码的空间**
 - **存放数据项目的空间**
 - **存放程序运行的控制或连接数据所需单元（控制栈）**

■ 一个程序在运行时刻的内存划分:





活动记录

- **活动记录：**为了管理一个过程活动所需的信息，使用一个连续的存储块存放这些信息，这个连续存储块称为**活动记录**。
- **运行时，**每当进入一个过程就有一个相应的活动记录累筑于栈顶。此记录含有连接数据、形式单元、局部变量、局部数组的内情向量和临时工作单元等。

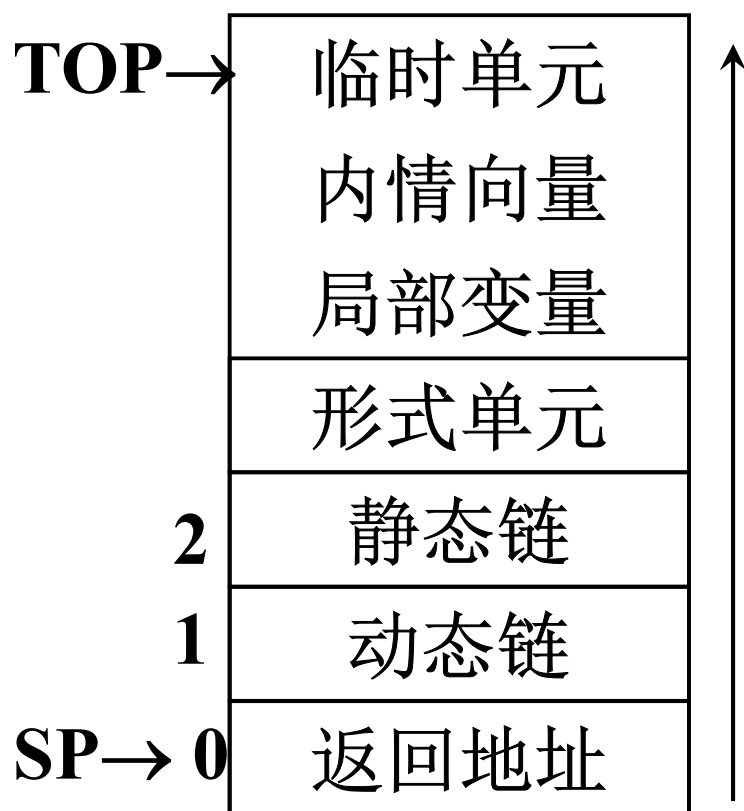
活动记录内容



■ 连接数据

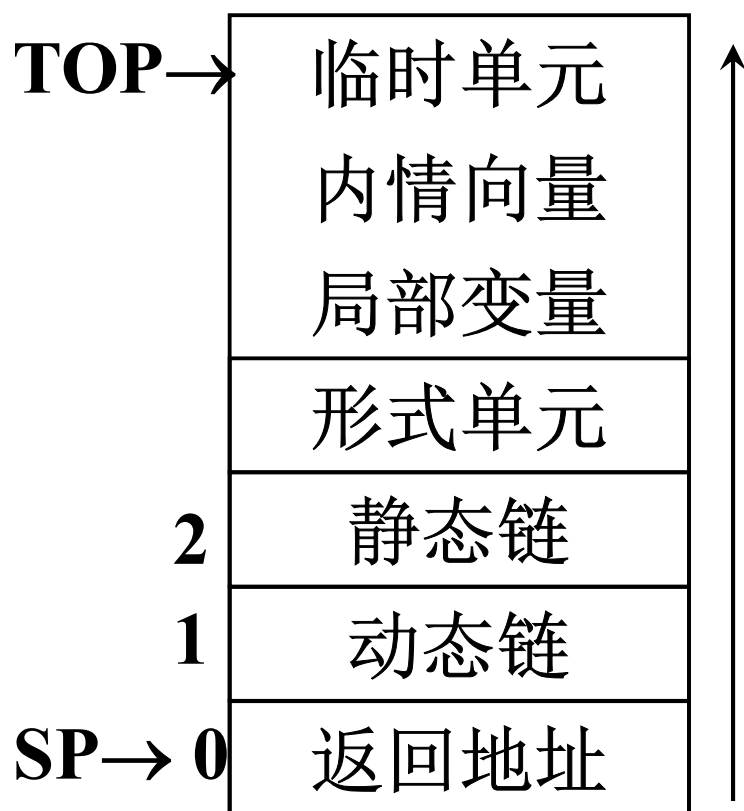
- 返回地址
- 动态链：指向调用该过程前的最新活动记录地址的指针。
- 静态链：指向静态直接外层最新活动记录地址的指针，用来访问非局部数据。

活动记录内容



- **形式单元：**存放相应的实在参数的地址或值。
- **局部数据区：**局部变量、内情向量、临时工作单元（如存放对表达式求值的结果）。

活动记录内容



**对任何局部变量X的引用
可表示为变址访问:**

dx[SP]

dx: 变量X相对于活动记录起点的地址, 在编译时可确定。



存储分配策略

- 静态分配策略(FORTRAN): 如果在编译时能确定数据空间的大小, 则可采用静态分配方法。在编译时刻为每个数据对象分配固定的存储单元, 且在运行时始终保持不变。
- 动态分配策略(PASCAL): 如果在编译时不能确定运行时数据空间的大小, 则必须采用动态分配方法。允许递归过程和动态申请释放内存。
 - 栈式动态分配
 - 堆式动态分配



内容线索

- ✓ 目标程序运行时的活动
- ✓ 运行时存储器的划分
- 静态存储分配
- 简单的栈式存储分配
- 嵌套过程语言的栈式实现



FORTRAN的存储分配

■ FORTRAN语言特点

- 不允许递归过程;
- 不允许可变数组;
- 每个数据所需空间是常数;
- 数据名性质确定;

■ 整个程序所需空间在编译时可确定, 可采用静态分配策略

■ 难点:

- COMMON 语句的处理
- EQUIVALENCE 语句的处理

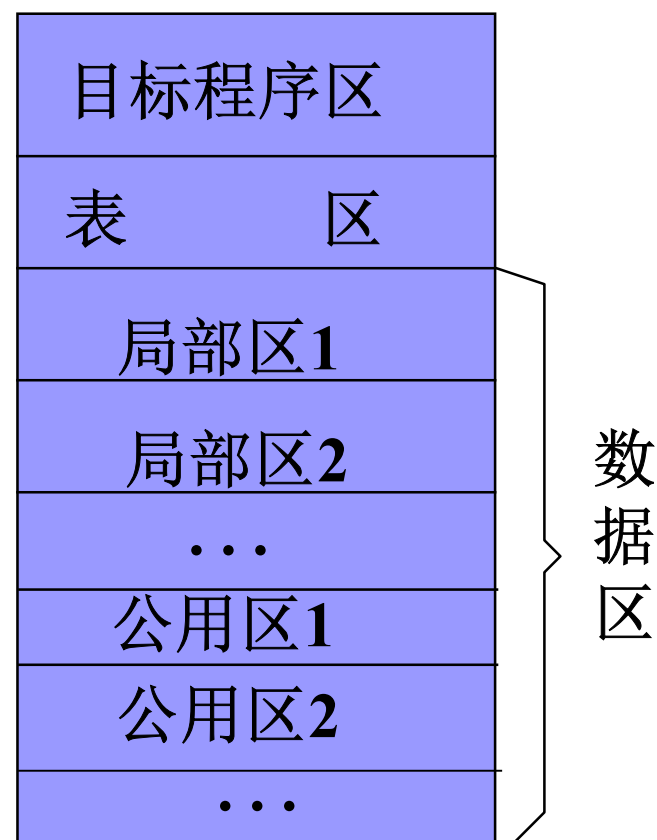
存储结构

■ 数据区

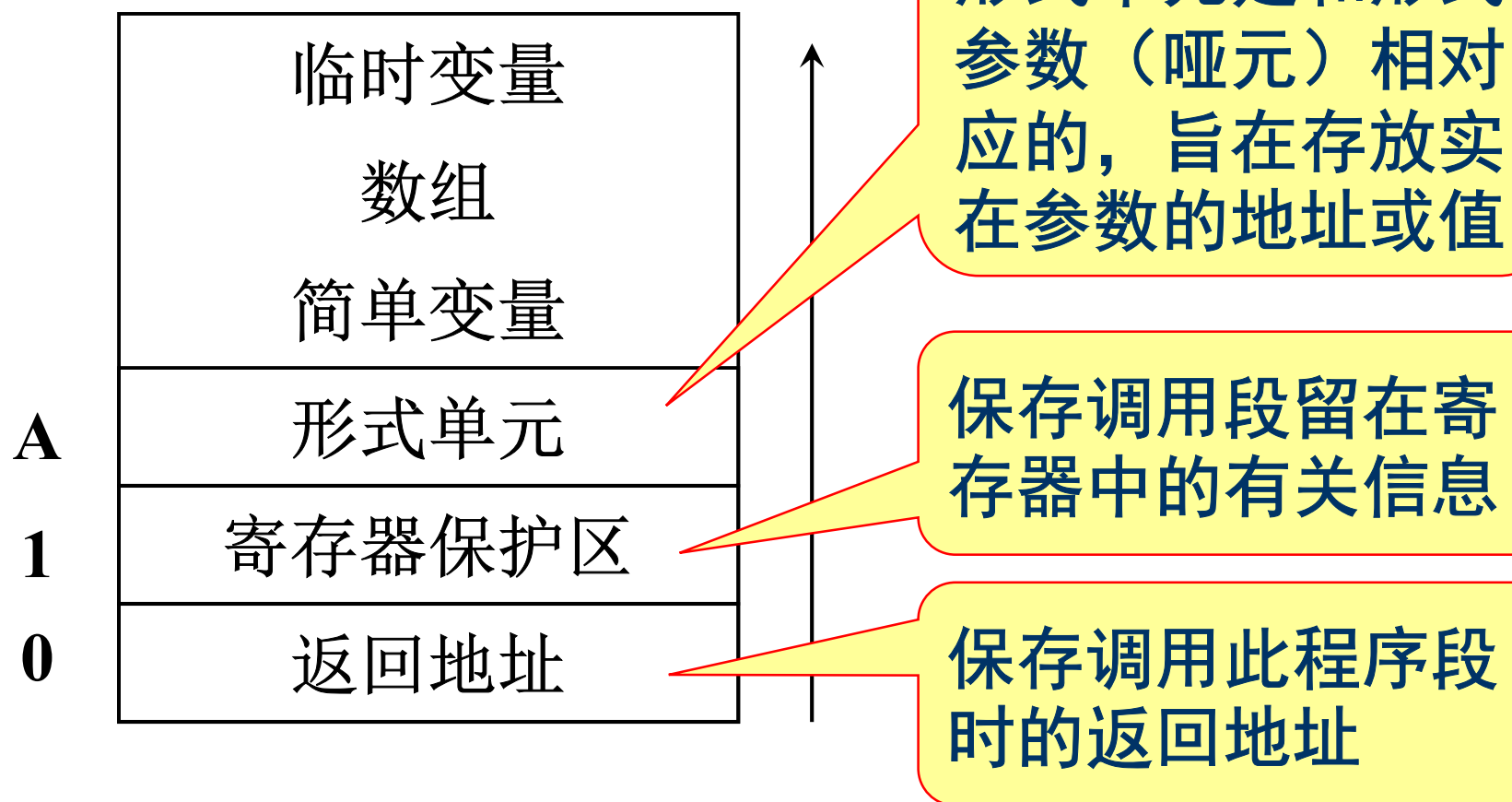
- 局部区: 每个程序段一个, (编号 <128) 存放局部变量的值。
- 公用区: 每个公用块一个, (编号 >128) 存放公用块中变量的值。

■ 存储映象:描述数据区中名字与地址的对应关系。

- 在符号表中, 对每个数据名登记其所属数据区编号及在该区中的相对位置。



局部数据区的内容及存放次序



■ 考虑子程序段：
 SUBROUTINE SWAP(A,B)
 T=A
 A=B
 B=T
 RETURN
 END

名字	性质	地址	
NAME	ATTRIBUTE	DA	ADDR
SWAP	子程序，二目		
A	哑，实变量	k	a
B	哑，实变量	k	a+2
T	实变量	k	a+4



临时变量的地址分配

- 特点: 放中间结果, 定值一次, 引用一次
- 方法: 用栈实现作用域的层次嵌套

例. $X := A * B - C * D + E * F$

四元式

$(*, A, B, T_1)$

$(*, C, D, T_2)$

$(-, T_1, T_2, T_3)$

$(*, E, F, T_4)$

$(+, T_3, T_4, T_5)$

$(:=, T_5, _, X)$

临时变量的地址分配

四元式

(*, A, B, T₁)
(*, C, D, T₂)
(-, T₁, T₂, T₃)
(*, E, F, T₄)
(+, T₃, T₄, T₅)
(:=, T₅, _, X)



临时变量名	地址
-------	----

T1	a
T2	a+1
T3	a
T4	a+1
T5	a

代真后的四元式

(*, A, B, \$a)
(*, C, D, \$(a+1))
(-, \$a, \$(a+1), \$a)
(*, E, F, \$(a+1))
(+, \$a, \$(a+1), \$a)
(:=, \$a, _, X)



内容线索

- ✓ 目标程序运行时的活动
- ✓ 运行时存储器的划分
- ✓ 静态存储分配
- 简单的栈式存储分配
- 嵌套过程语言的栈式实现

简单程序语言

■ C语言特点

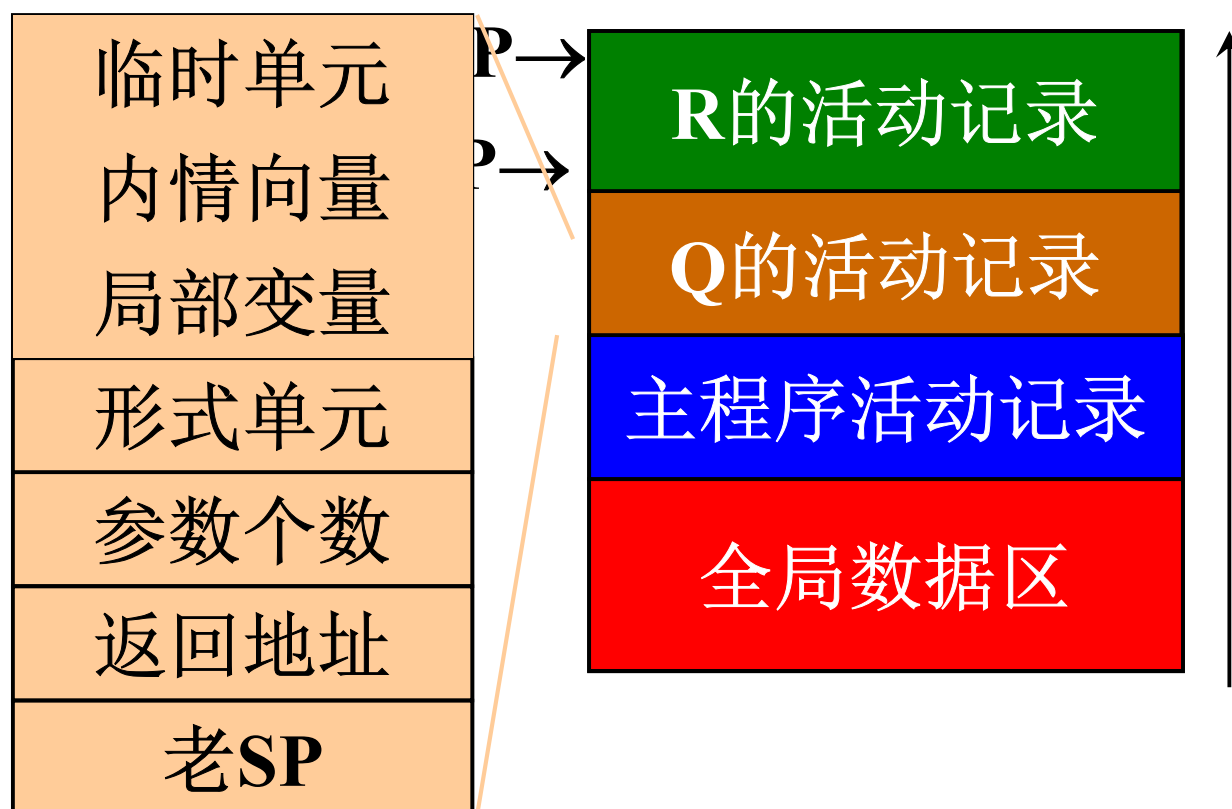
- 没有分程序结构，过程定义不允许嵌套
- 允许过程的递归调用。

C语言的程序结构

```
全局数据说明;  
main( )  
{  
    main中数据说明;  
    ...  
}  
void R( );  
{ R 中数据说明;  
  ... }  
void Q( );  
{ Q 中数据说明;  
  ... }
```

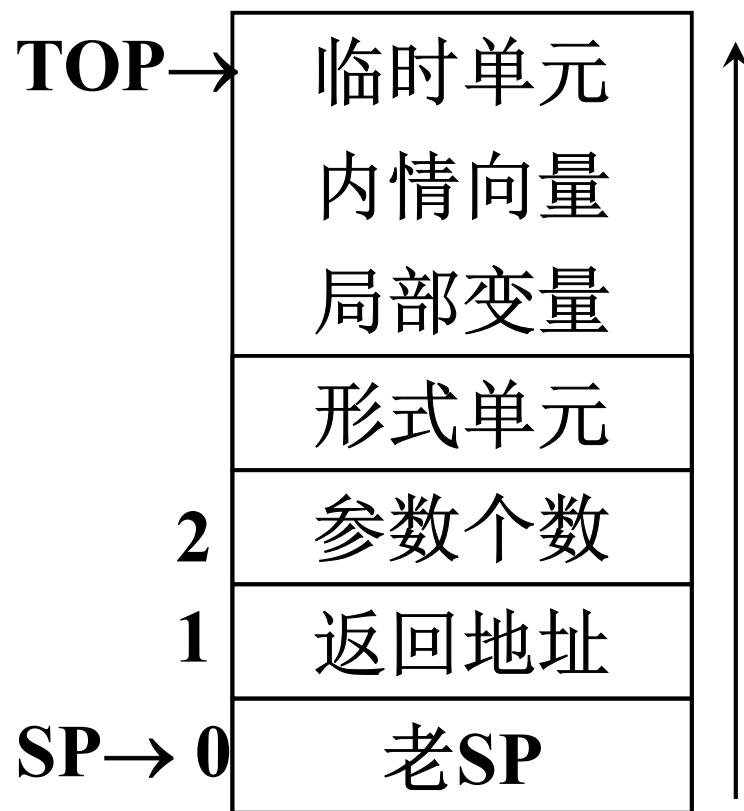
存储组织

主程序 → 过程Q → 过程R



- 非局部数据静态分配于栈底。
- 局部数据分配于活动记录。
- 栈式: 每调用一个过程将其活动记录分配于栈顶, 退出一个过程释放该空间。

C的活动记录



对任何局部变量X的引用可表示为变址访问:

dx[SP]

dx: 变量X相对于活动记录起点的地址, 在编译时可确定。

C 过程调用

■ 过程调用的四元式序列:

par T_1

par T_2

...

par T_n

call P, n

参数传递

形参数目

过程名

C过程调用的翻译过程

1) 每个par $T_i (i=1,2,\dots,n)$ 可直接翻译成如下指令:

$(i+3)[TOP] := T_i$ (传值)

$(i+3)[TOP] := \text{addr}(T_i)$ (传地址)

2) call P, n 被翻译成:

$1[TOP] := SP$ (保护现行SP)

$3[TOP] := n$ (传递参数个数)

JSR P (转子指令)



3) 转进过程P后，首先应执行

下述指令：

$SP := TOP + 1$

(定义新的SP)

$1[SP] := \text{返回地址}$

(保护返回地址)

$TOP := TOP + L$

(新TOP)

L：过程P的活动记录所需单元数，在编译时可确定。

TOP →

临时单元
内情向量
局部变量

形式单元

参数个数

返回地址

SP →

老SP

调用过程的活动记录

4) 过程返回时，应执行下列指令：

TOP:=SP-1

(恢复调用前TOP)

SP:=0[SP]

(恢复调用前SP)

X:=2[TOP]

(把返回地址取到X中)

UJ 0[X]

(按X返回)

TOP→

临时单元
内情向量
局部变量

形式单元

参数个数

返回地址

老SP

SP→
TOP→
SP→

调用过程的活动记录



内容线索

- ✓ 目标程序运行时的活动
- ✓ 运行时存储器的划分
- ✓ 静态存储分配
- ✓ 简单的栈式存储分配
- 嵌套过程语言的栈式实现



Pascal语言

- 语言特点

- ☐ 过程嵌套定义
- ☐ 过程递归调用

- 一个PASCAL过程：

过程头；

说明段（由一系列的说明语句组成）；

begin

执行体（由一系列的执行语句组成）；

end



Pascal过程调用规则

- (1) 任何子程序都不能调用主程序，主程序可直接调用所有第1层的子程序。**
- (2) 任何子程序（包括主程序）可直接调用自己的直接内层子程序，但不能直接调用隔层的内层子程序。**
- (3) 内层子程序可以直接调用自己及它的任何一层外层子程序（不包括主程序）。**



Pascal过程调用规则

- (4) 并列子程序中后说明的可以调用先说明的，而先说明的不能调用后说明的（除非采用向前引用的方式）。**
- (5) 一个子程序可以调用所有与自己的某个外层子程序并列，且其说明先于这个外层子程序的子程序。**
- (6) Pascal语言允许子程序自己调用自己，这种调用方式称为递归调用**

```

program main
  var A, B : real;
  ...
  procedure P1
    var B:boolean;
    ...
  begin
    ...
  end
  procedure P2
    var A:integer;
    ...
  begin
    ...
  end
begin
  ...
end

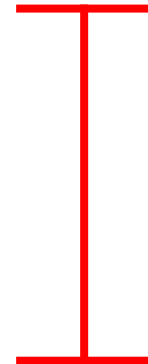
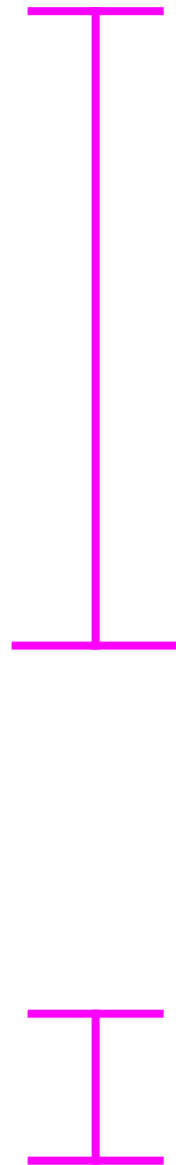
```

A(real)

A(integer)

B(real)

B(bool)





非局部名字的访问的实现

- 主程序的层次为0；在 i 层中定义的过程，其层次为 $i+1$ ；

(层数， 偏移量)

- 过程运行时，必须知道其所有外层过程的当前活动记录的起始地址。
 - 嵌套层次显示表display
 - 静态链和活动记录

```

program P;
var a, x : integer;
procedure Q(b: integer);
    var i: integer;
    procedure R(u: integer; var v:
integer);
        var c, d: integer;
        begin
            if u=1 then R(u+1, v)
            .....
            v:=(a+c)*(b-d);
            .....
        end {R}
    begin
        .....
        R(1,x);
        .....
    end {Q}

```

```

procedure S;
    var c, i:integer;
    begin
        a:=1;
        Q(c);
        .....
    end {S}
begin
    a:=0;
    S;
    .....
end. {P}

```

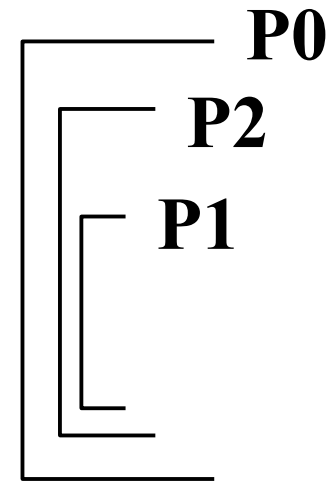
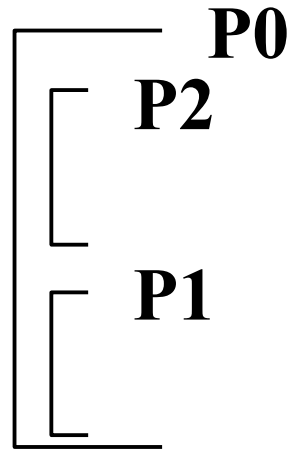
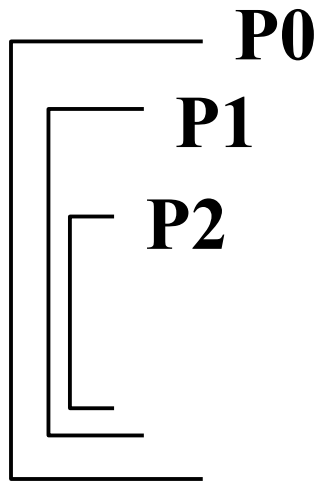
主程序P →过程 S
 →过程 Q →过程 R
 →过程 R

嵌套层次显示表display

- 当进入一个过程后，在建立其活动记录区的同时建立一张嵌套层次显示表display，把display表作为活动记录的一部分。
- 令过程R的外层为Q，Q的外层为主程序P，则过程R运行时的DISPLAY表内容为：

2	R 的现行活动记录的地址(SP 的现值)
1	Q 的最新活动记录的地址
0	P 的活动记录的地址

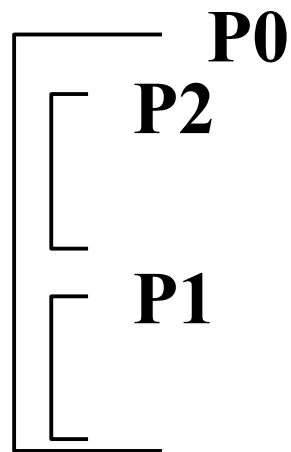
- 问题：当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？



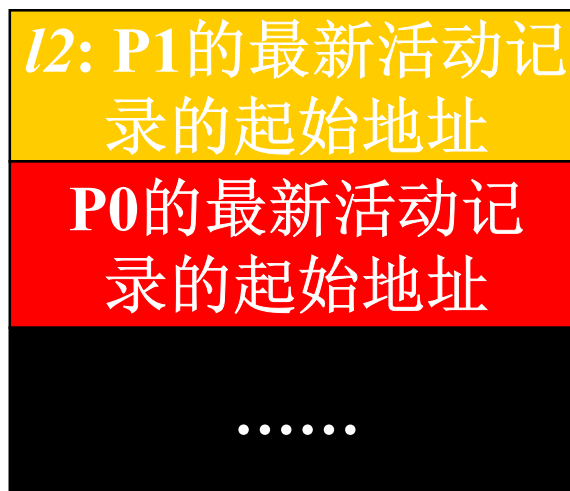
- 问题：当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？



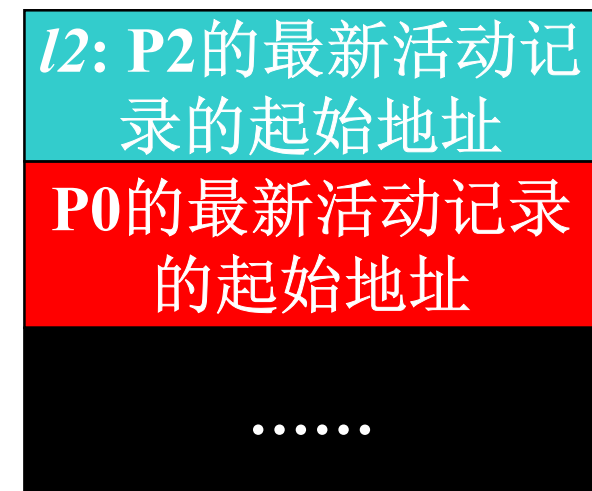
- 问题：当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？



从P1的display表中自底而上地取过 l_2 个单元（ l_2 为P2的层数）再添上进入P2后新建的SP值就构成了P2的display表。

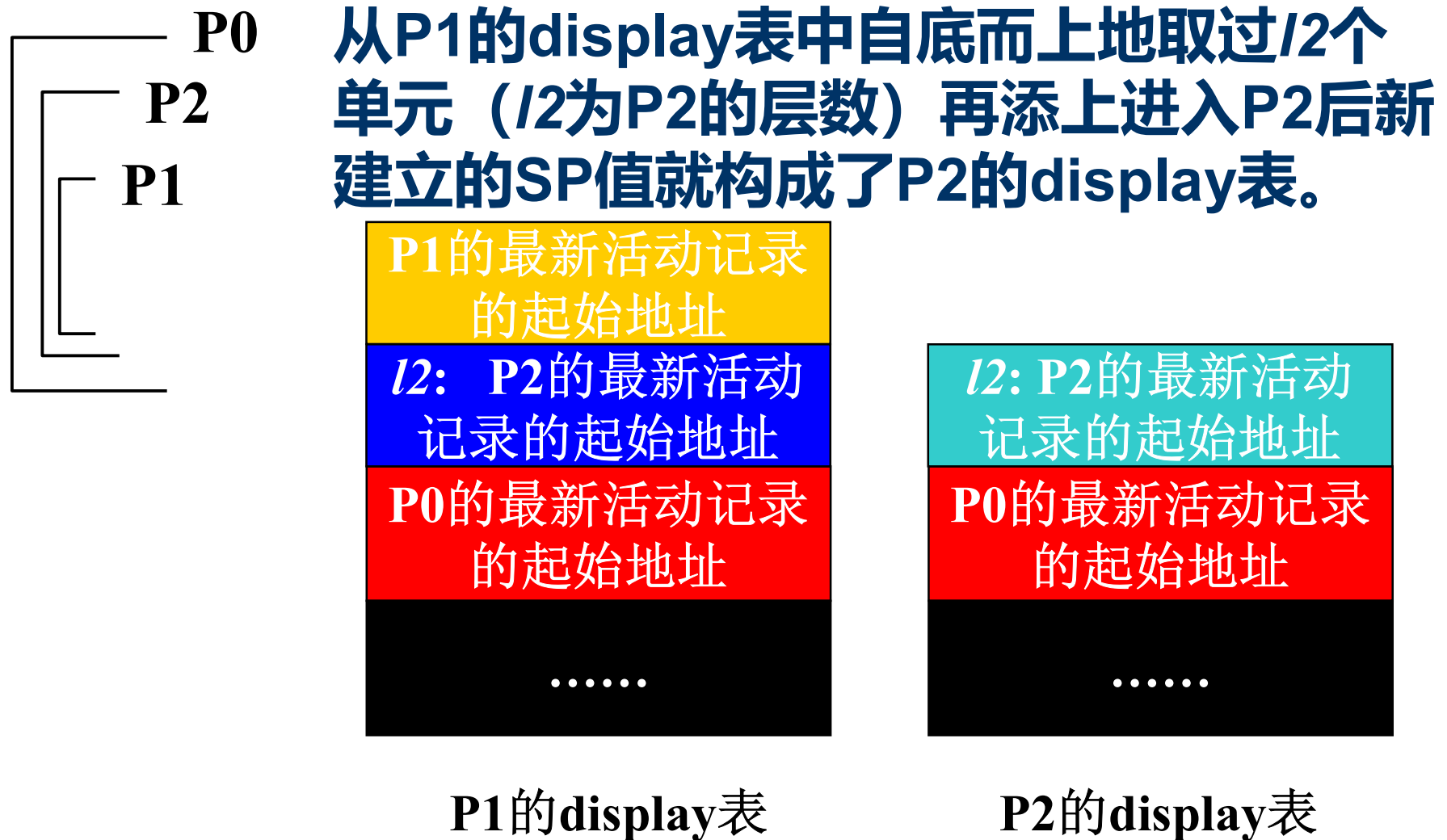


P1的display表

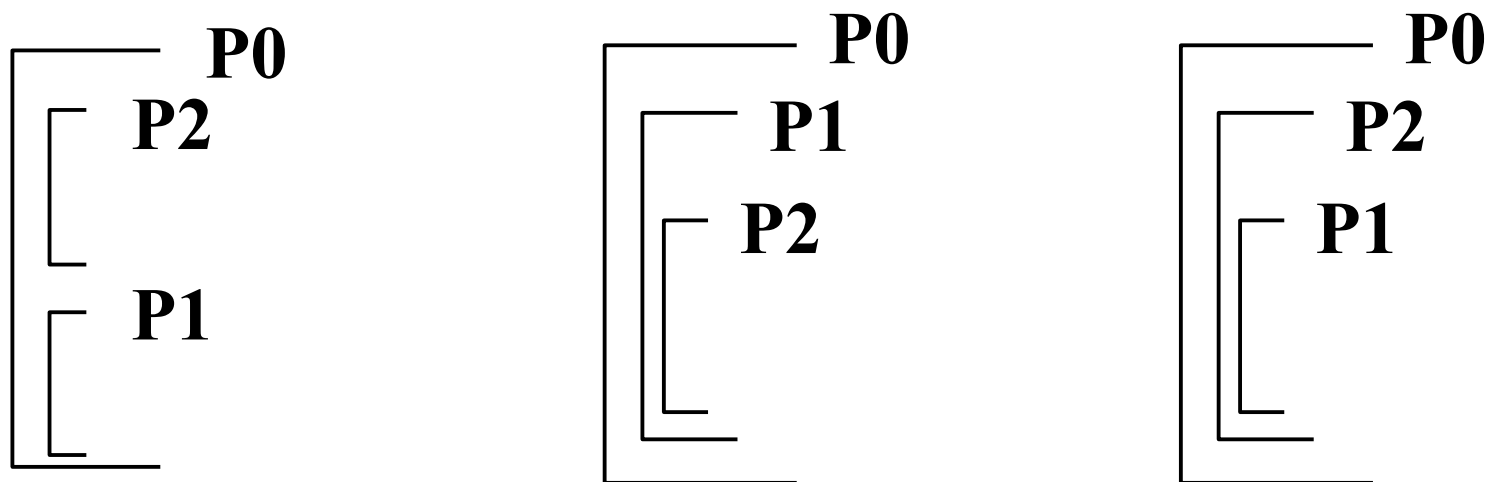


P2的display表

- 问题：当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？



- 问题：当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？



答案：从P1的display表中自底而上地取过 l_2 个单元（ l_2 为P2的层数）再添上进入P2后新建立的SP值就构成了P2的display表。

- 👉 把P1的display表地址作为连接数据之一传送给P2就能够建立P2的display表。

嵌套过程语言活动记录

TOP→	临时单元 内情向量 局部变量
d	Display 表
...	形式单元
3	参数个数
2	全局Diaplay
1	返回地址
SP→ 0	老SP

- diaplay表在活动记录中的相对地址d在编译时能完全确定。
- 假定在现行过程中引用了某层过程(令其层次为k)的X变量, 那么, 可用下面两条指令获得X的地址:
LD R₁ (d+k)[SP]
LD R₂ dx[R₁]

```

program P;
var a, x : integer;
procedure Q(b: integer);
    var i: integer;
    procedure R(u: integer; var v:
integer);
    var c, d: integer;
    begin
        if u=1 then R(u+1, v)
        .....
        v:=(a+c)*(b-d);
        .....
    end {R}
begin
    .....
    R(1,x);
    .....
end {Q}

```

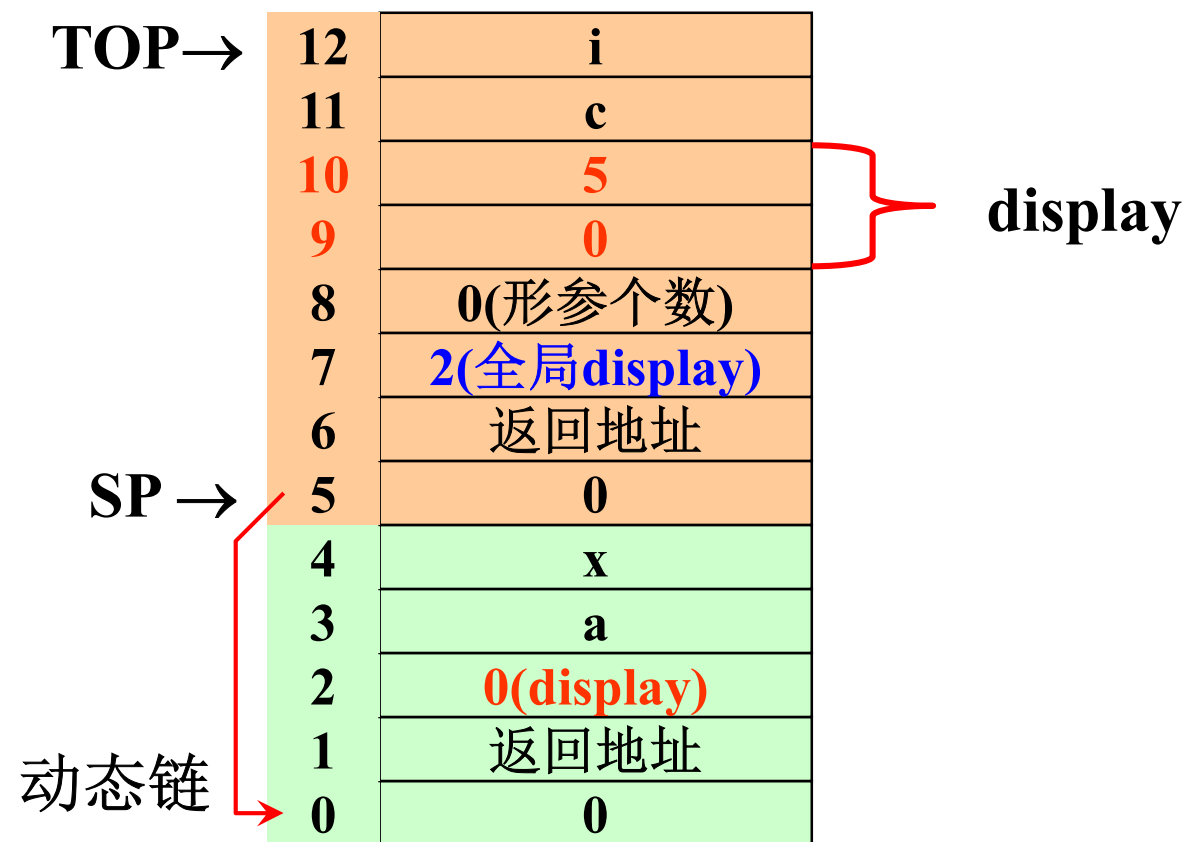
主程序P →过程 S
 →过程 Q →过程 R
 →过程 R

```

procedure S;
    var c, i:integer;
    begin
        a:=1;
        Q(c);
        .....
    end {S}
begin
    a:=0;
    S;
    .....
end. {P}

```

主程序→
过程 S





TOP→

主程序 P→

过程 S →

过程 Q

SP →

动态链

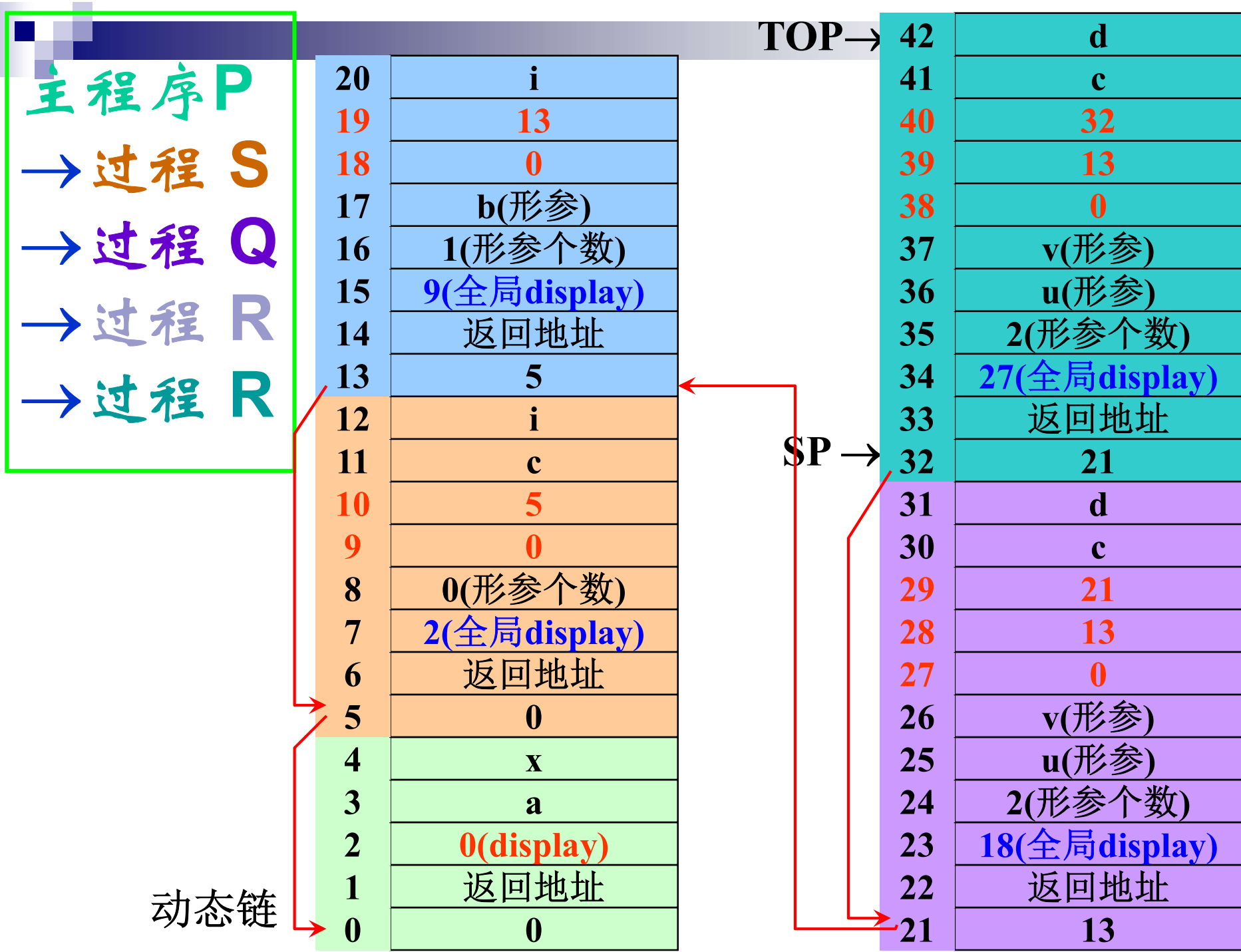
20	i
19	13
18	0
17	b(形参)
16	1(形参个数)
15	9(全局display)
14	返回地址
13	5
12	i
11	c
10	5
9	0
8	0(形参个数)
7	2(全局display)
6	返回地址
5	0
4	x
3	a
2	0(display)
1	返回地址
0	0

主程序 P →
过程 S →
过程 Q →
过程 R

动态链

20	i
19	13
18	0
17	b(形参)
16	1(形参个数)
15	9(全局display)
14	返回地址
13	5
12	i
11	c
10	5
9	0
8	0(形参个数)
7	2(全局display)
6	返回地址
5	0
4	x
3	a
2	0(display)
1	返回地址
0	0

TOP →	31	d
	30	c
	29	21
	28	13
	27	0
	26	v(形参)
	25	u(形参)
	24	2(形参个数)
	23	18(全局display)
	22	返回地址
SP →	21	13



过程调用、过程进入、过程返回

1. 每个par $T_i (i=1,2,\dots,n)$ 可直接翻译成如下指令:

$(i+4)[TOP] := T_i$ (传值)

$(i+4)[TOP] := \text{addr}(T_i)$ (传地址)

TOP →

SP →

临时单元
内情向量
局部变量

Display 表

形式单元

参数个数

全局Diaplay

返回地址

老SP

调用过程的
活动记录

.....

过程调用、过程进入、过程返回

2. call P, n 被翻译成:

1[TOP]:=SP (保护现行SP)

3[TOP]:=i[SP] (传送现行display地址)

4[TOP]:=n (传递参数个数)

JSR (转子指令)

TOP→

SP →

临时单元
内情向量
局部变量

Display 表

形式单元

参数个数

全局Diaplay

返回地址

老SP

调用过程的
活动记录

.....

TOP →

3. 转进过程P后，首先定义新的SP和TOP，保存返回地址。
4. 根据"全局display"建立现行过程的display：从全局display表中自底向上地取/个单元，在添上进入P后新建立的SP值就构成了P的display。

SP →

临时单元
内情向量
局部变量

Display 表

形式单元

参数个数

全局Diaplay

返回地址

老SP

调用过程的
活动记录

.....

TOP→

临时单元
内情向量
局部变量

5. 过程返回时，执行下述指令：

TOP:=SP-1

SP:=0[SP]

X:=2[TOP]

UJ 0[X]

Display 表

形式单元

参数个数

全局Diaplay

返回地址

老SP

SP →
TOP →

调用过程的
活动记录

SP →

.....



作业

■ P269

☐ 4

☐ 5