Lecture 13: 运行时刻环境-I

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301

存储组织

过程的声明/定义

- **过程：函数、过程、方法、子例程的统称**
- **过程定义是一个声明，它的最简单形式是将一个名字和一个语句联系起来。**
  - 该名字是过程名，而这个语句是过程体。在大多数语言中，返回值的过程叫做函数，完整的程序也可以看作一个过程。

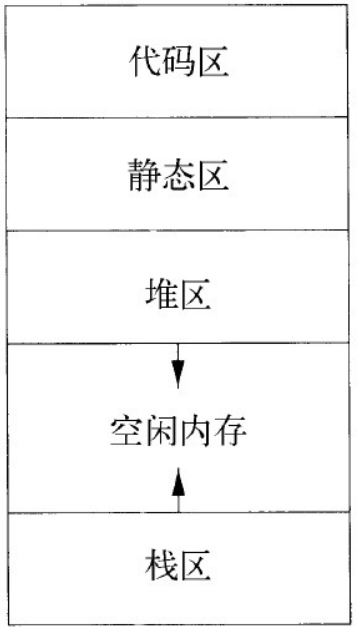
过程的执行

- **当过程名出现在调用语句中时，就说这个过程在该点被调用**
  - 过程调用就是执行被调用过程的过程体
  - 过程调用也可以出现在表达式中，这时也叫做函数调用
- **运行时过程的一次执行称为过程的一次活动**
  - 过程的活动需要可执行代码和存放所需信息的存储空间，后者通常用一块连续的存储区来管理，称为活动记录

存储分配的典型方式

- 目标程序在它自己的逻辑地址空间内运行，
  OS将逻辑地址映射为物理地址(内存空间实际编址)
- 四个字节构成一个机器字，
  多字节数据对象存储在一段连续的字节中，
  并把第一个字节作为它的地址

目标程序的代码放置在代码区



静态区、堆区、栈区分别放置不同类型生命期的数据值

静态和动态存储分配

- **运行时刻环境**
  - 为数据分配安排存储位置---关键任务
  - 确定访问变量时使用的机制
  - 过程之间的连接
  - 参数传递
  - 和操作系统、输入输出设备相关的其它接口
- **主题**
  - 存储管理：栈分配、堆管理、垃圾回收
  - 对变量、数据的访问

静态和动态存储分配

- **程序中同一个名字在运行时指向不同存储位置**
  - 静态（编译时刻）：编译器只需要通过scan source code，就可以完成存储分配
  - 动态（运行时刻）：无法在编译器决定，运行时才决定

静态分配

- 名字在程序被编译时绑定到存储单元，不需要运行时的任何支持
- 绑定的生存期是程序的整个运行期间

静态分配给语言带来限制

- 递归过程不被允许
- 数据对象的长度和它在内存中位置的限制，必须是在编译时可以知道的
- 数据结构不能动态建立

静态分配给语言带来限制

- **例　C程序的外部变量、静态局部变量以及程序中出现的常量都可以静态分配**
- **声明在函数外面**
  - 外部变量 —— 静态分配
  - 静态外部变量 —— 静态分配
- **声明在函数里面**
  - 静态局部变量 —— 也是静态分配
  - 自动变量 —— 不能静态分配

动态存储分配

- **动态分配**
  - 栈式存储：和过程的调用/返回同步进行分配和回收，值的生命期和过程生命期相同
  - 堆存储：数据对象比创建它的过程调用更长寿
    - 手工进行回收
    - 垃圾回收机制

空间的栈式分配

调用栈分配

```
int a[11];
void readArray() { /* 将9个整数读入到a[1],....a[9]中。*/
    int i;
    ...
}
int partition(int m, int n) {
    /* 选择一个分割值 v，划分 a[m..n]，
       使得 a[m..p-1] 小于 v, a[p] = v，
       并且 a[p+1..n] 大于等于 v，返回 p. */
    ...
}
void quicksort(int m, int n) {
    int i;
    if (n > m) {
        i = partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
    }
}
main() {
    readArray();
    a[0] = -9999;
    a[10] = 9999;
    quicksort(1,9);
}
```

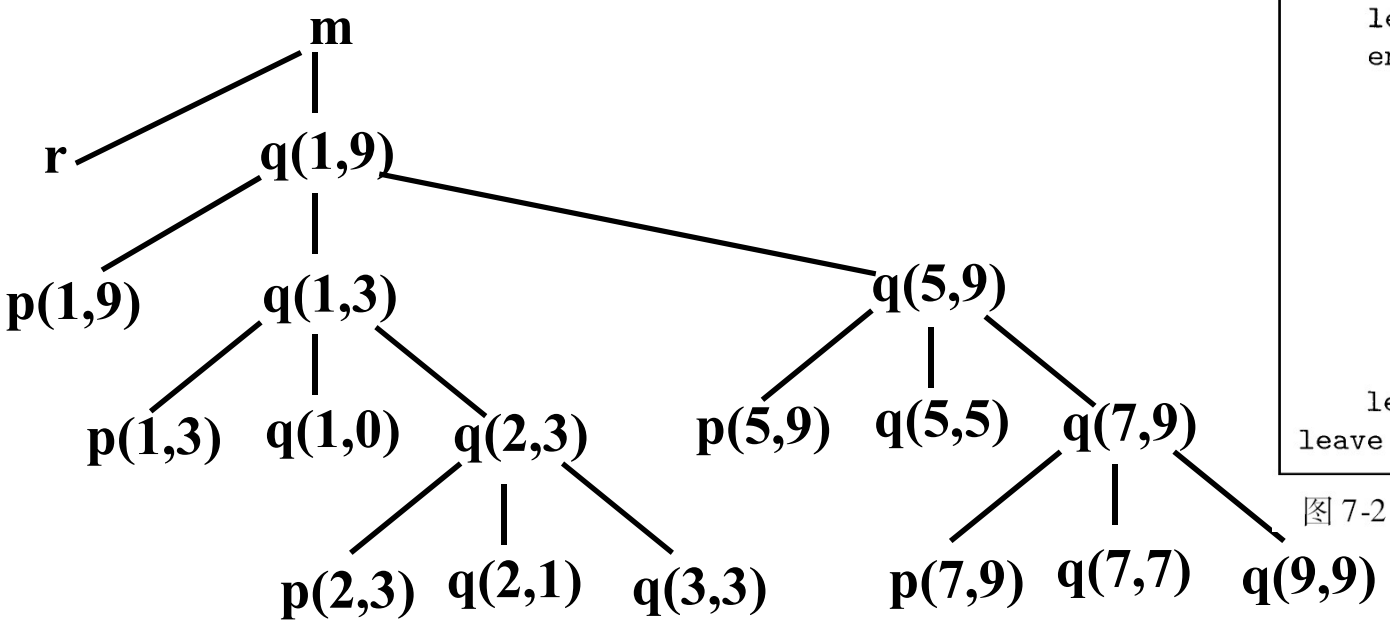
图 7-2　一个快速排序程序的概要

活动树

- **过程调用（过程活动）在时间上总是嵌套的**
  - 后调用的先返回
  - 因此用栈式分配来分配过程活动所需内存空间
- **程序运行的所有过程活动可以用树表示**
  - 每个结点对应于一个过程活动
  - 根结点对应于main过程的活动
  - 过程p的某次活动对应的结点的所有子结点：此次活动所调用的各个过程活动（从左向右，表示调用的先后顺序）

活动树

- 用树来描绘控制进入和离开活动的方式




```
enter main()
    enter readArray()
    leave readArray()
    enter quicksort(1,9)
        enter partition(1,9)
        leave partition(1,9)
        enter quicksort(1,3)
            ...
        leave quicksort(1,3)
        enter quicksort(5,9)
            ...
        leave quicksort(5,9)
    leave quicksort(1,9)
leave main()
```

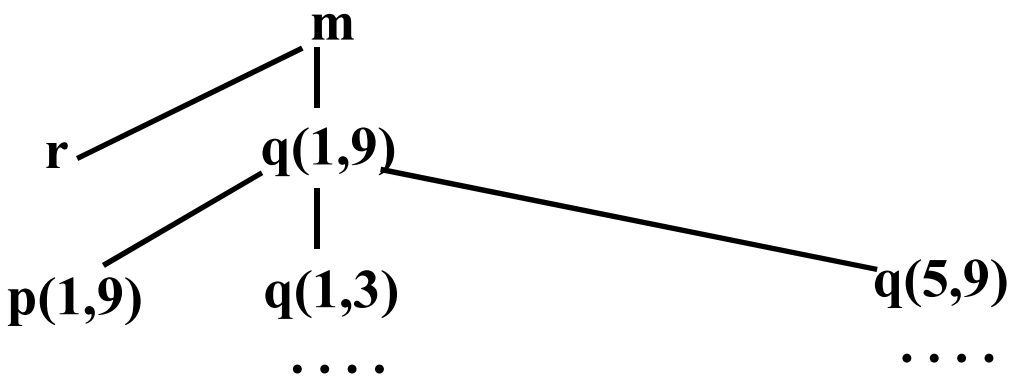
图 7-2 中程序的可能的活动序列

活动树的特点

- 每个结点代表某过程的一个活动
- 根结点代表主程序的活动
- 结点*a*是结点*b*的父结点，当且仅当控制流从*a*的活动进入*b*的活动
- 结点*a*处于结点*b*的左边，当且仅当*a*的生存期先于*b*的生存期

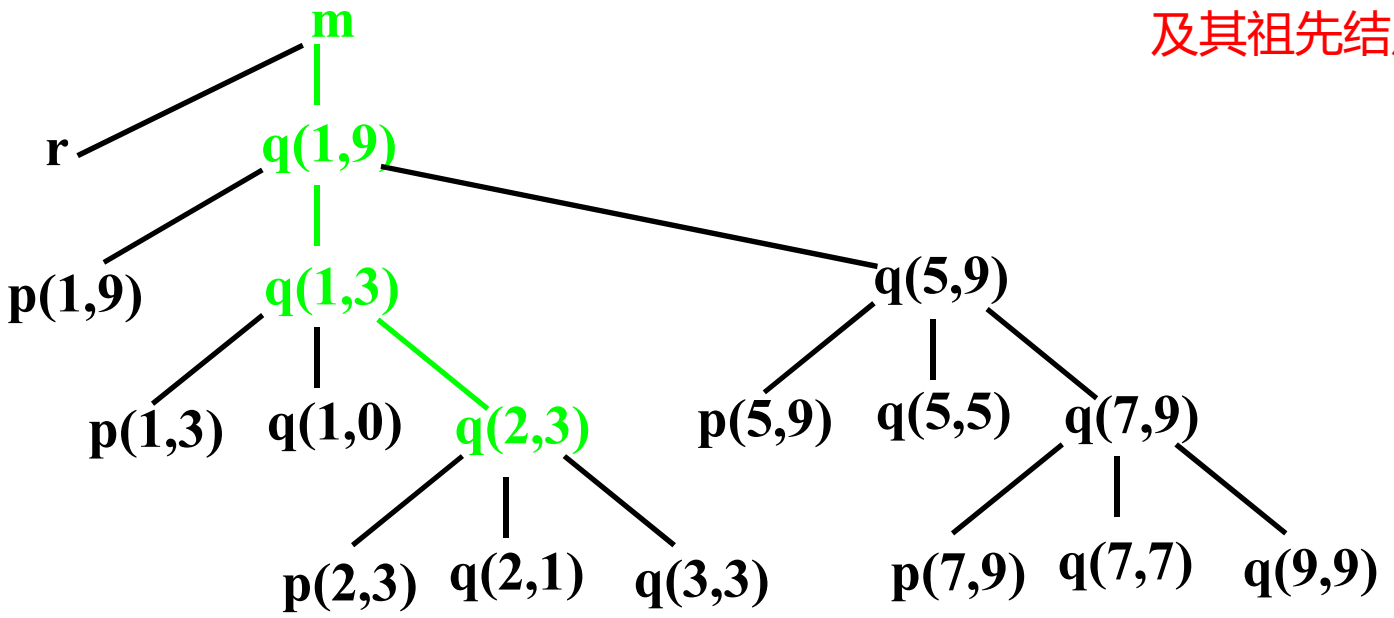
过程调用（返回）序列和活动树的前序（后序）遍历对应

假定当前活动对应结点N，那么所有尚未结束的活动对应于N及其祖先结点。



活动树和栈的关系

- **当前活跃着的过程活动可以保存在一个栈中**
  - **控制栈的内容：m, q (1, 9), q (1, 3), q (2, 3)**



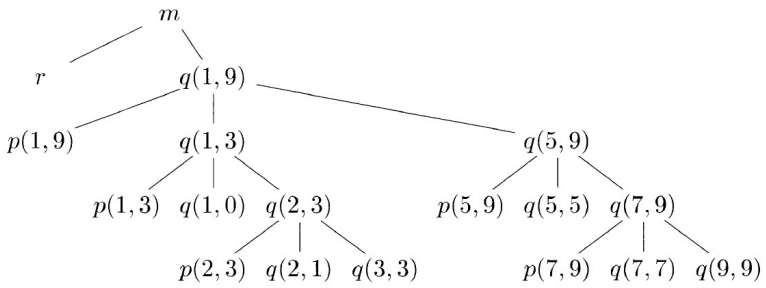
假定当前活动对应结点N，那么所有尚未结束的活动对应于N及其祖先结点。

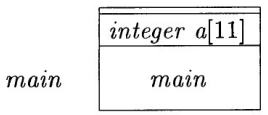
活动记录

- 过程调用和返回由控制栈进行管理
- 每个活跃的活动对应于栈中的一个活动记录
- 活动记录按照活动的开始时间，从栈底到栈顶排列

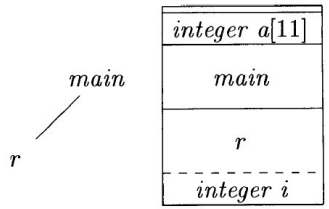
运行时刻栈的例子

- a[11]为全局变量
- main没有局部变量
- r有局部变量i
- q的局部变量i，和参数m,r

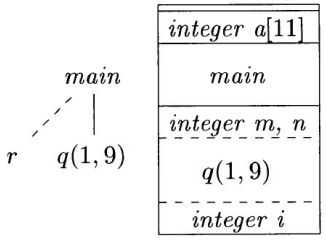




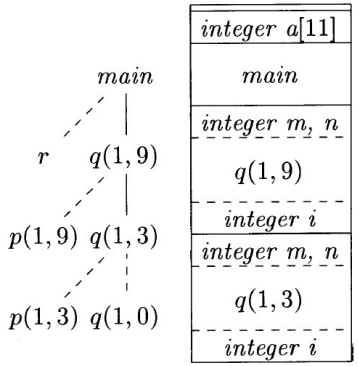
a) γ被弹出栈



b) γ被激活



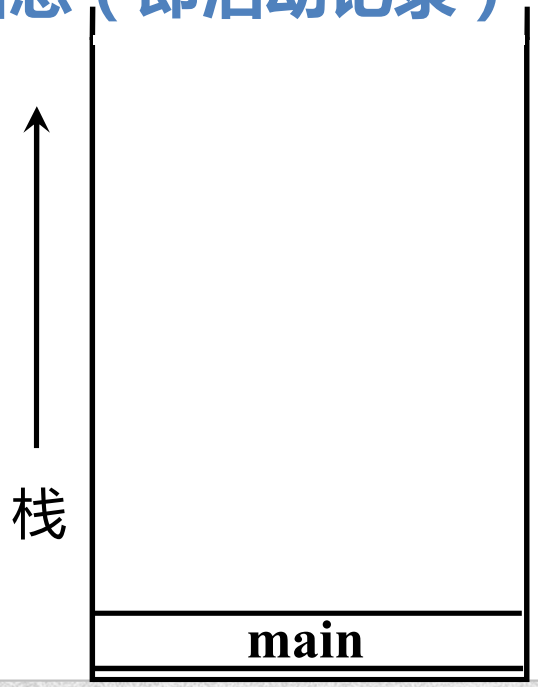
c) γ被弹出栈，$q(1,9)$ 被压栈



d) 控制返回到 $q(1,3)$

运行时刻栈的例子

运行栈：把控制栈中的信息拓广到包括过程活动所需的所有局部信息（即活动记录）

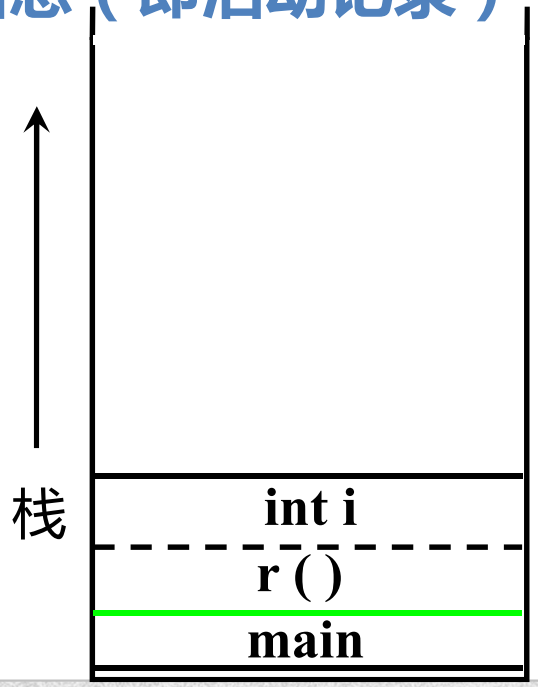


函数调用关系树

main

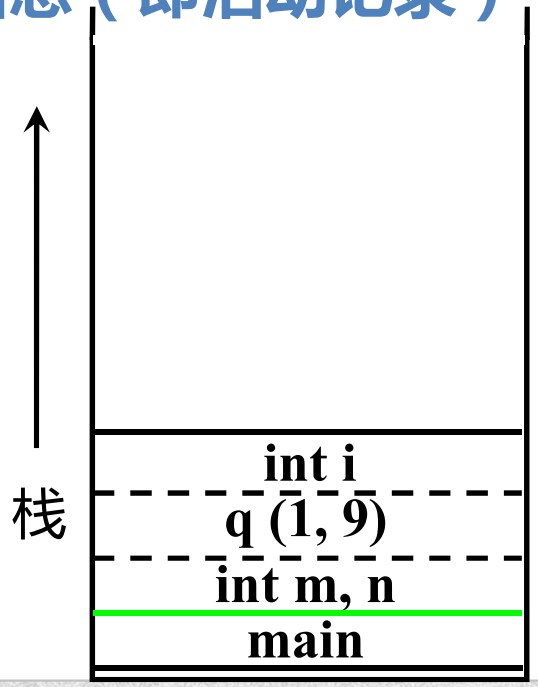
运行时刻栈的例子

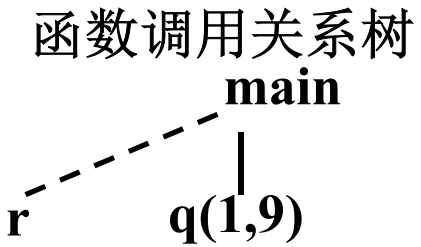
运行栈：把控制栈中的信息拓广到包括过程活动所需的所有局部信息（即活动记录）



运行时刻栈的例子

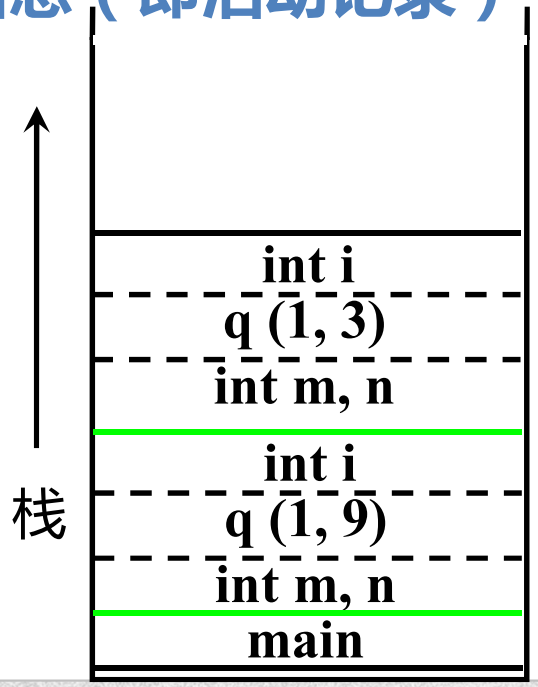
运行栈：把控制栈中的信息拓广到包括过程活动所需的所有局部信息（即活动记录）

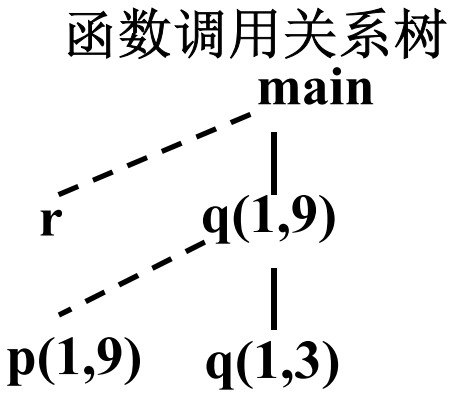




运行时刻栈的例子

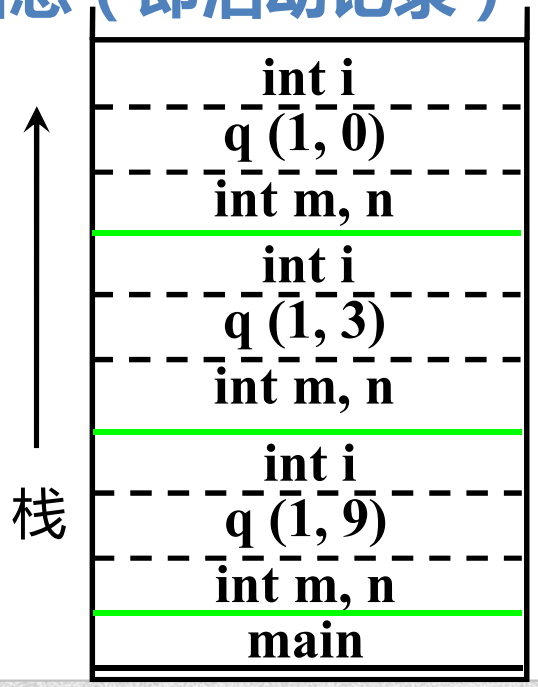
运行栈：把控制栈中的信息拓广到包括过程活动所需的所有局部信息（即活动记录）

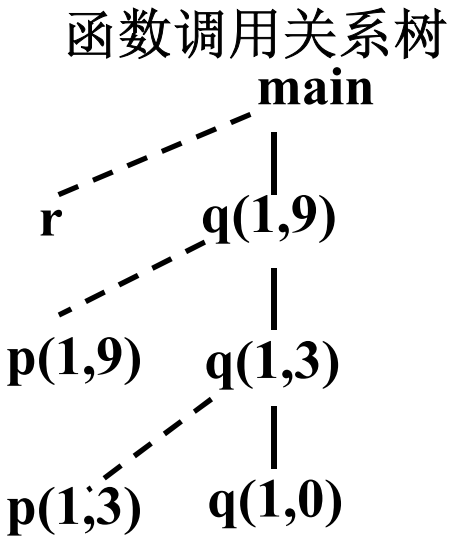




运行时刻栈的例子

运行栈：把控制栈中的信息拓广到包括过程活动所需的所有局部信息（即活动记录）





调用代码序列

- **调用代码序列(calling sequence)为活动记录分配空间，填写记录中的信息**
- **返回代码序列(return sequence)恢复机器状态，使调用者继续运行**
- **调用代码序列会分割到调用者和被调用者中**
  - 根据源语言、目标机器、操作系统的限制，可以有不同的分割方案
  - 把代码尽可能放在被调用者中

调用/返回代码序列的要求

- **数据方面**
  - 能够把参数正确地传递给被调用者
  - 能够把返回值传递给调用者
- **控制方面**
  - 能够正确转到被调用过程的代码开始位置
  - 能够正确转回调用者的调用位置（的下一条指令）
- **调用代码序列和活动记录的布局相关**

活动记录的布局原则

- **即使是同一种语言，过程调用序列、返回序列和活动记录中各域的排放次序，也会因实现而异**
- **设计这些序列和活动记录的一些原则**
  - 以活动记录中间的某个位置作为基地址
  - 长度能较早确定的域放在活动记录的中间

活动记录的布局原则

- **即使是同一种语言，过程调用序列、返回序列和活动记录中各域的排放次序，也会因实现而异**
- **设计这些序列和活动记录的一些原则**
  - 一般把临时数据域放在局部数据域的后面
  - 把参数域和可能有的返回值域放在紧靠调用者活动记录的地方

活动记录的布局原则

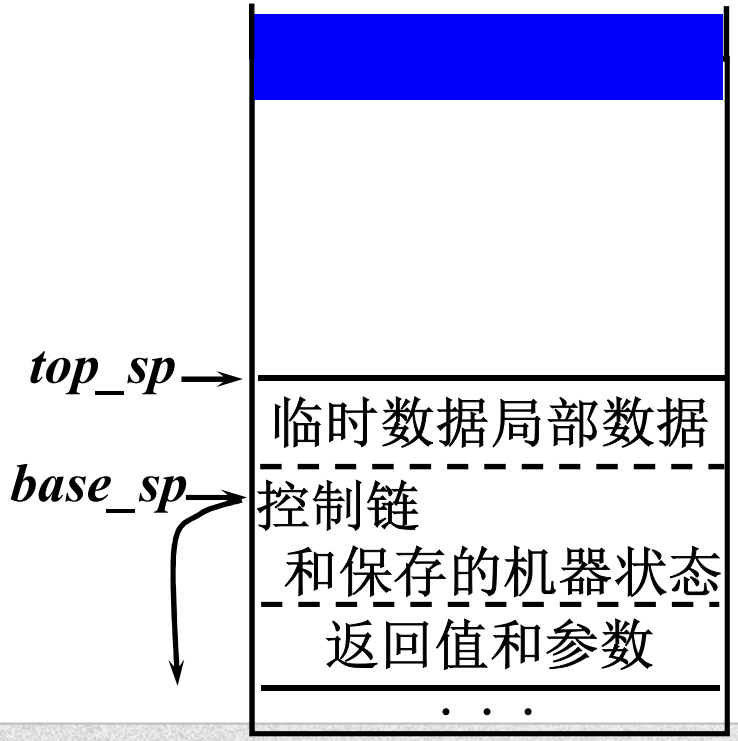
- **即使是同一种语言，过程调用序列、返回序列和活动记录中各域的排放次序，也会因实现而异**
- **设计这些序列和活动记录的一些原则**
  - 用同样的代码来执行各个活动的保存和恢复

调用代码序列的例子

- **Calling sequence**
  - 调用者计算实在参数的值
  - 将返回地址和原top_sp存放到被调用者的活动记录中。调用者增加top_sp的值（越过了局部数据、临时变量、被调用者的参数、机器状态字段）
  - 被调用者保存寄存器值和其他状态字段
  - 被调用者初始化局部数据、开始运行
- **Return sequence**
  - 被调用者将返回值放到和参数相邻的位置
  - 恢复top_sp和寄存器，跳转到返回地址

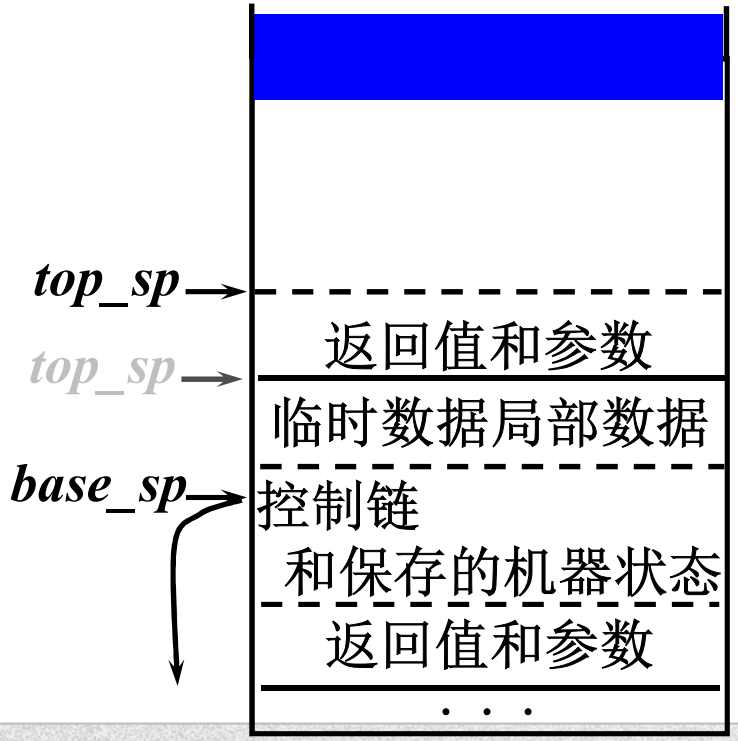
调用代码序列的例子

1、过程p调用过程q的调用序列



调用代码序列的例子

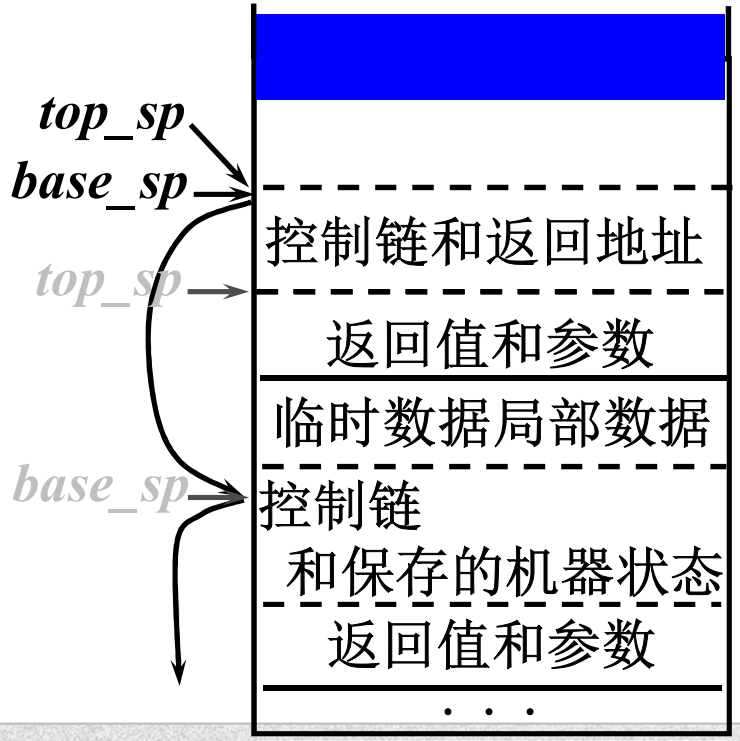
1、过程p调用过程q的调用序列



(1) p计算实参，依次放入栈顶，并在栈顶留出放返回值的空间。*top_sp*的值在此过程中被改变

调用代码序列的例子

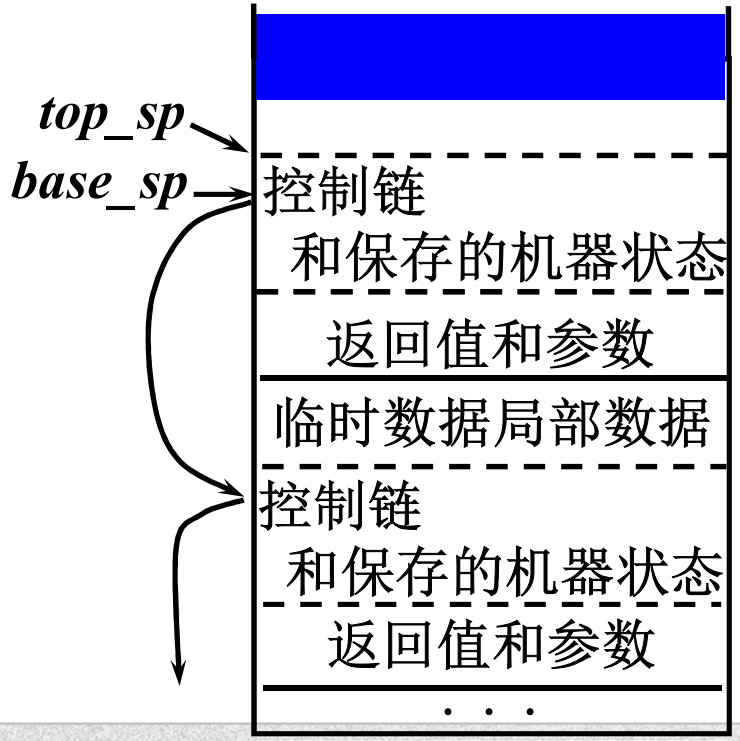
1、过程p调用过程q的调用序列



(2) p把返回地址和当前*base_sp*的值存入q的活动记录中，建立q的访问链，增加*base_sp*的值

调用代码序列的例子

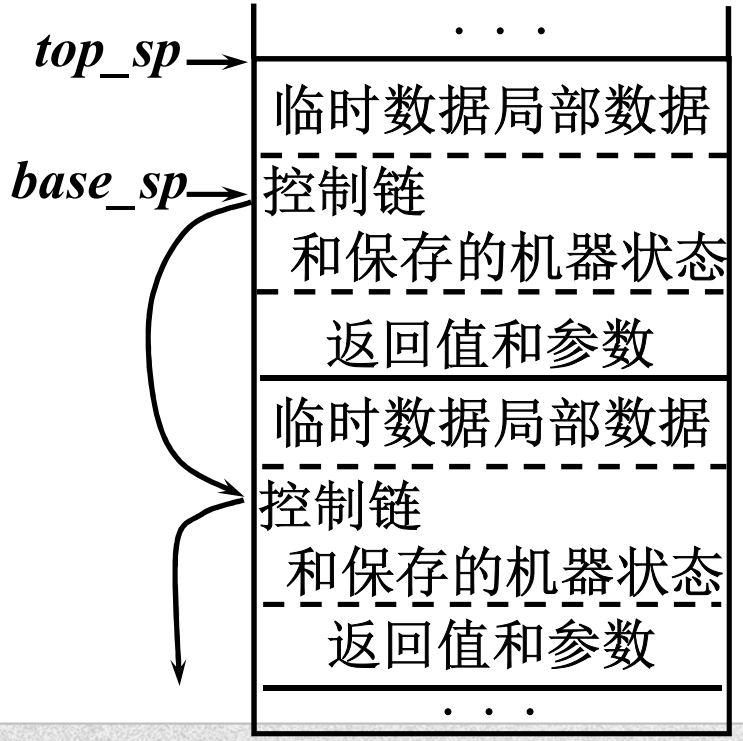
1、过程p调用过程q的调用序列



(3) q保存寄存器的值和其它机器状态信息

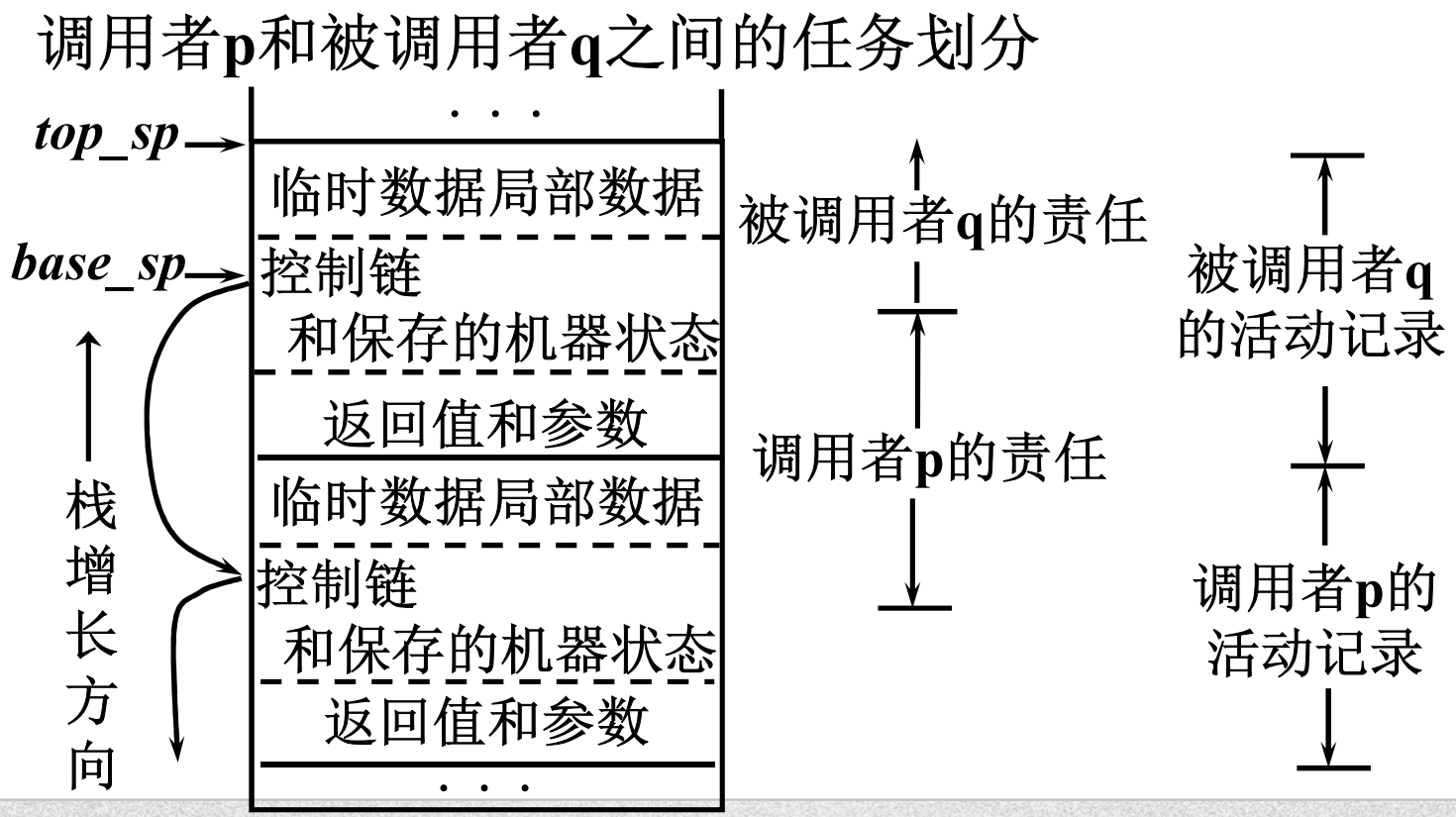
调用代码序列的例子

1、过程p调用过程q的调用序列



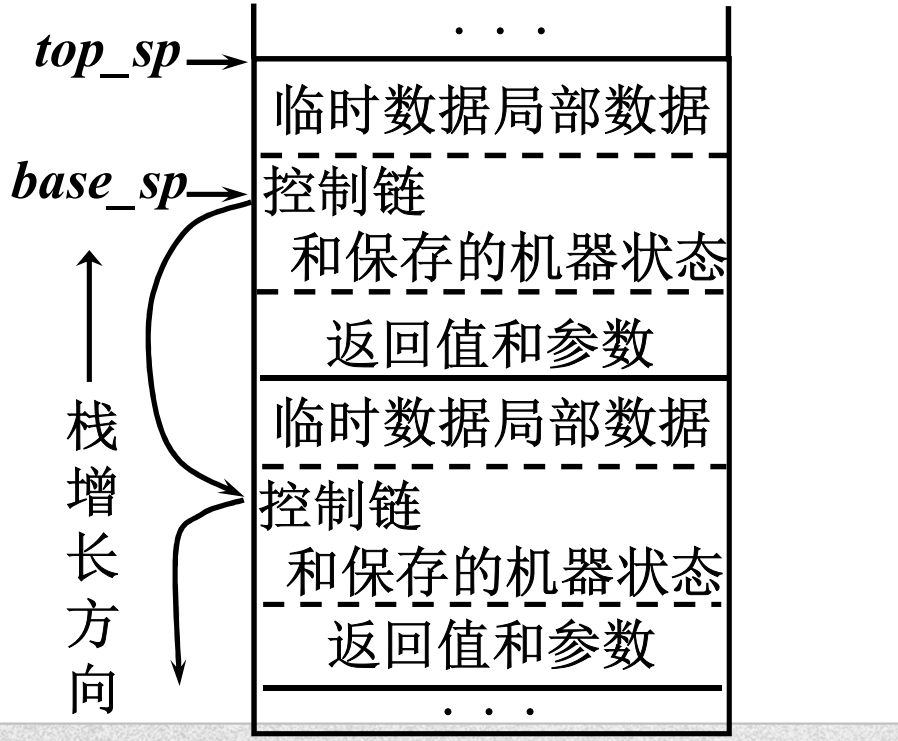
(4) q根据局部数据域和临时数据域的大小增加*top_sp*的值，初始化它的局部数据，并开始执行过程体

调用代码序列的例子


调用者p和被调用者q之间的任务划分
top_sp
base_sp
临时数据局部数据
控制链
和保存的机器状态
返回值和参数
临时数据局部数据
控制链
和保存的机器状态
返回值和参数
栈增长方向
被调用者q的责任
调用者p的责任
被调用者q的活动记录
调用者p的活动记录

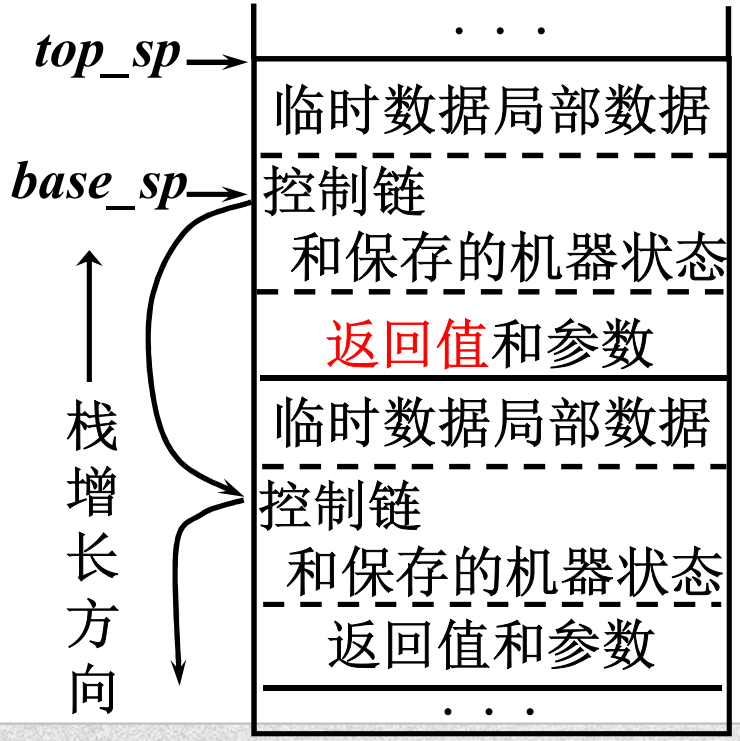
调用代码序列的例子

2、过程p调用过程q的返回序列



调用代码序列的例子

2、过程p调用过程q的返回序列

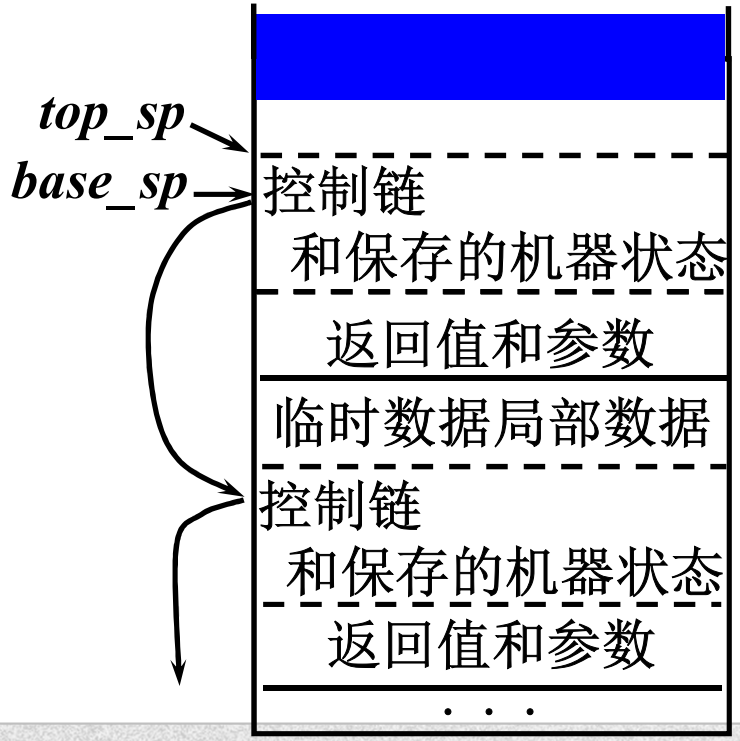


(1) q把返回值置入邻近p的活动记录的地方

参数个数可变场合难以确定存放返回值的位置，因此通常用寄存器传递返回值

调用代码序列的例子

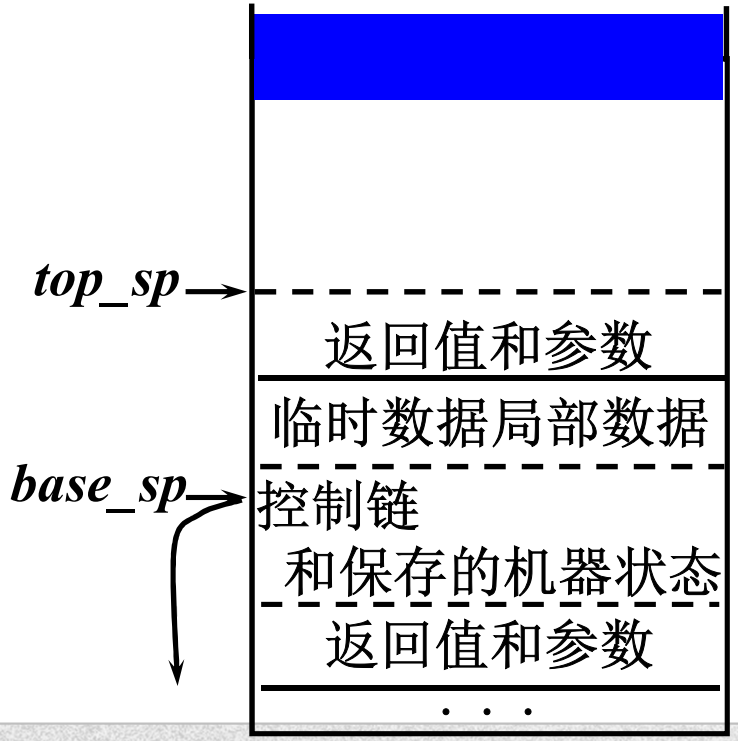
2、过程p调用过程q的返回序列



(2) q对应调用序列的步骤(4)，减小*top_sp*的值

调用代码序列的例子

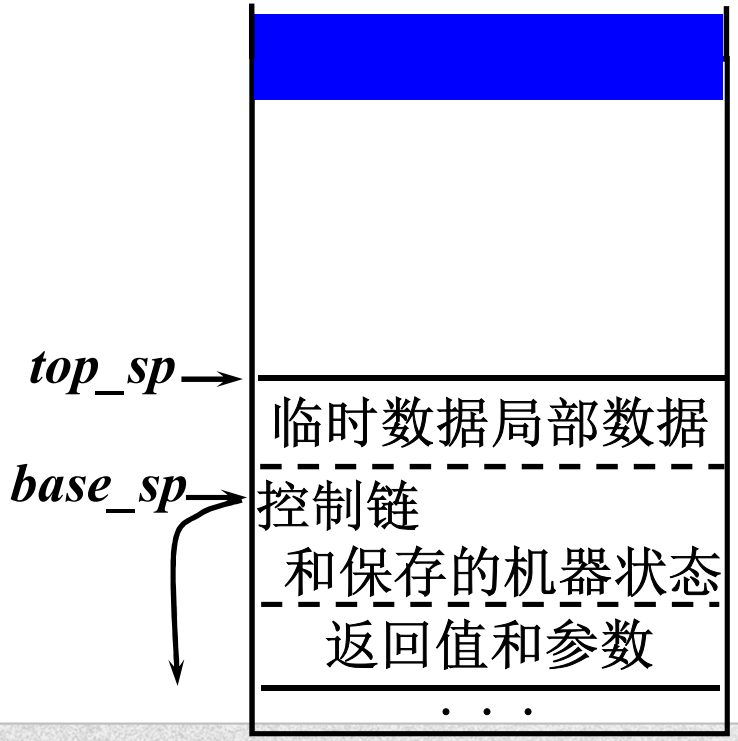
2、过程p调用过程q的返回序列



(3) q恢复寄存器(包括*base_sp*)和机器状态，返回p

调用代码序列的例子

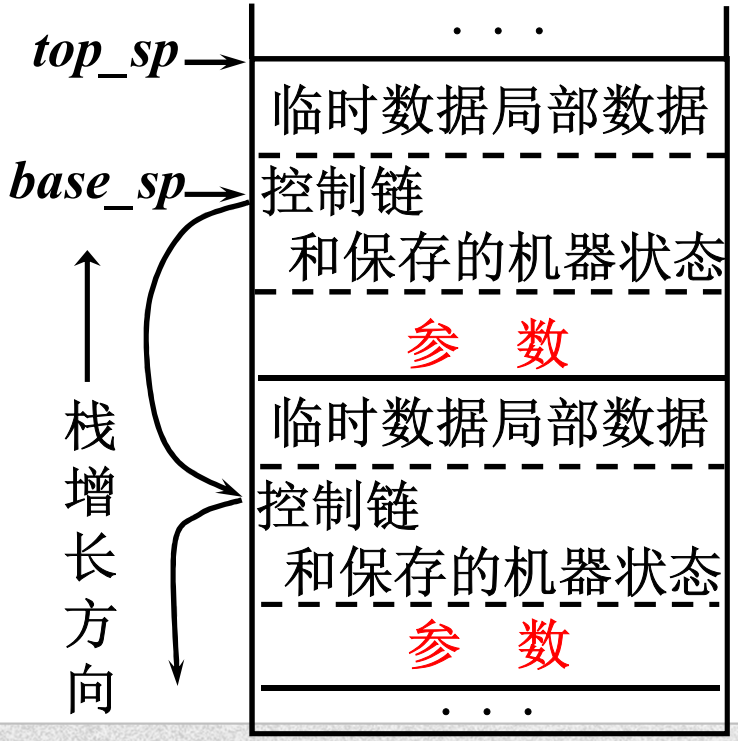
2、过程p调用过程q的返回序列



(4) p根据参数个数与类型和返回值类型调整*top_sp*，然后取出返回值

调用代码序列的例子

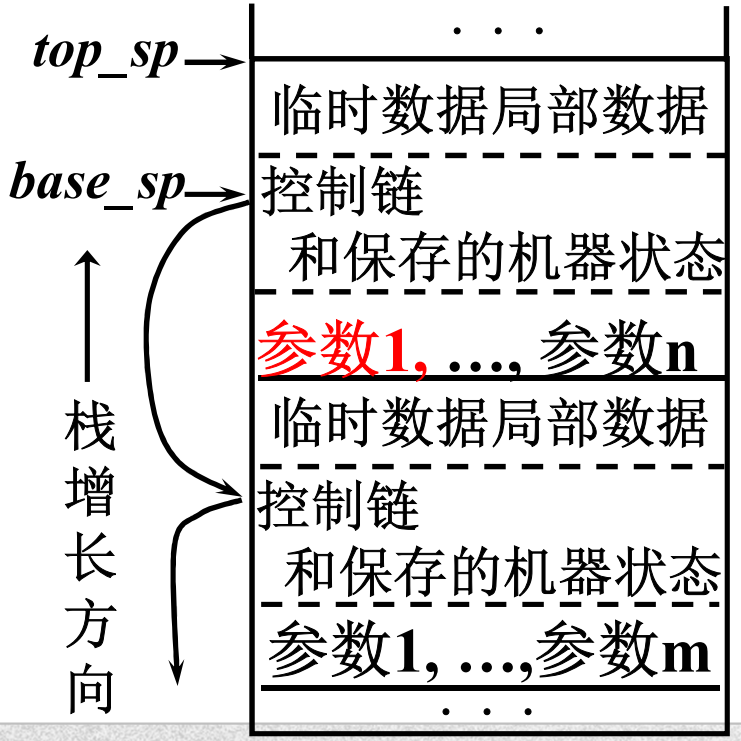
3、过程的参数个数可变的情况



(1) 函数返回值改成用寄存器传递

调用代码序列的例子

3、过程的参数个数可变的情况



(3) 从右向左压入参数，使得被调用函数能准确地知道第一个参数的位置

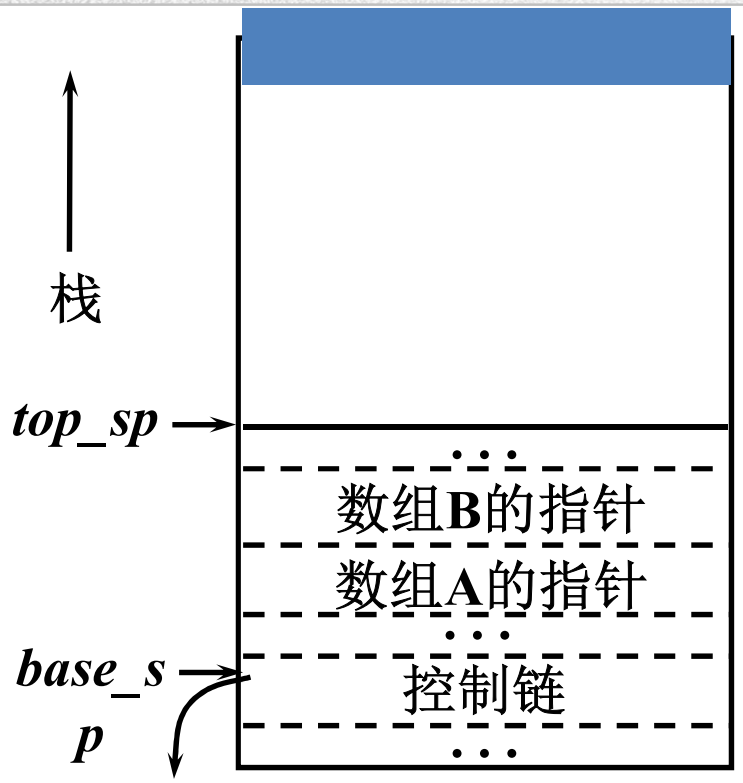
栈上可变长数据

活动记录的长度在编译时不能确定的情况

- 例：局部数组的大小要等到过程激活时才能确定

备注： Java语言的实现是将它们分配在堆上

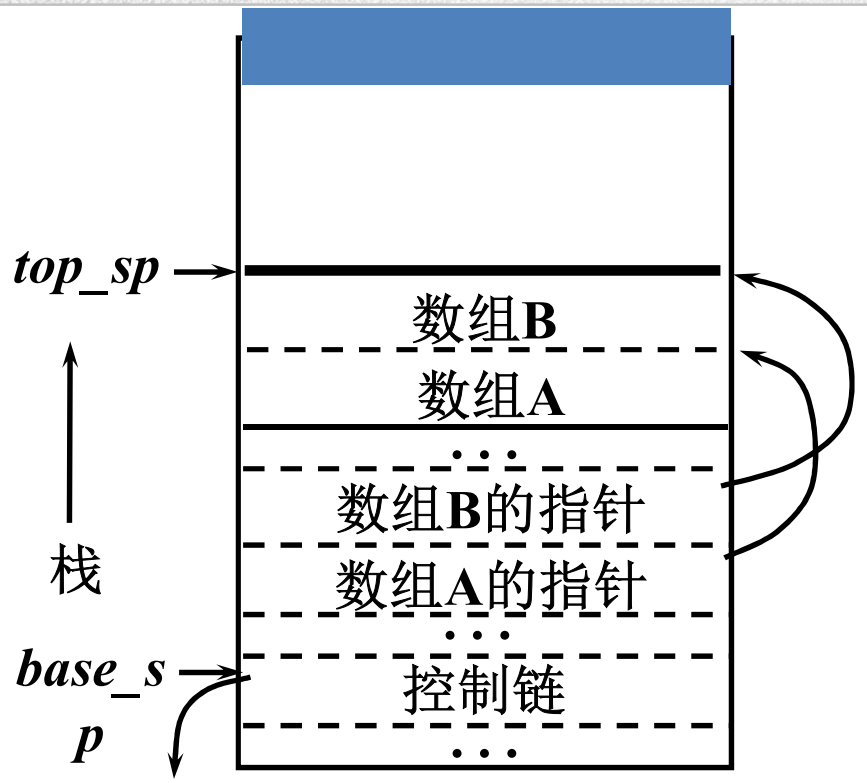
栈上可变长数据



(1) 编译时，在活动记录中为这样的数组分配存放数组指针的单元

访问动态分配的数组

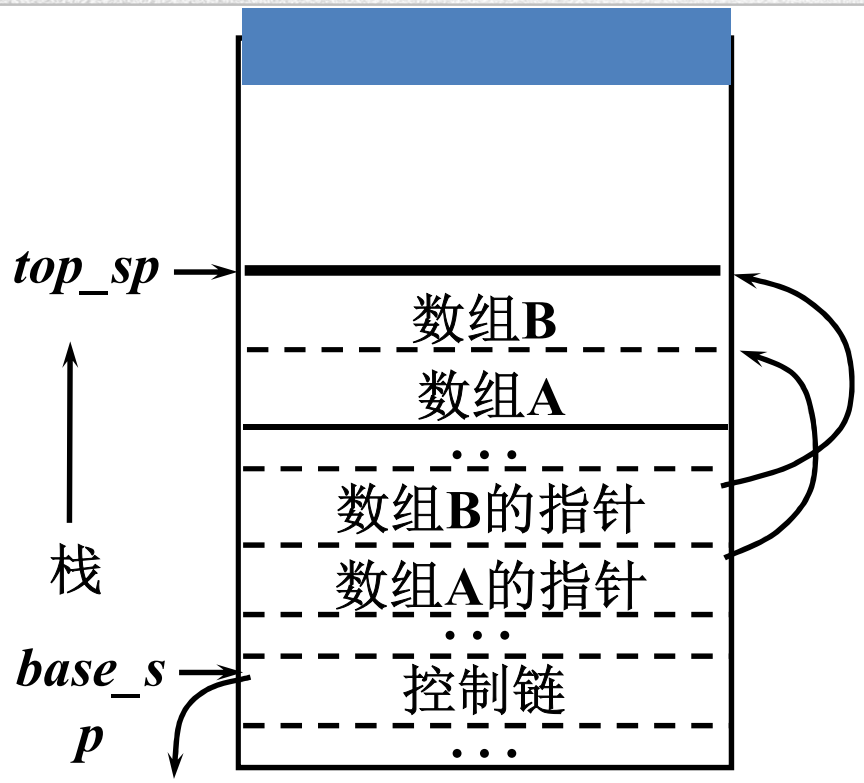
栈上可变长数据



(2) 运行时，
这些指针指向
分配在栈顶的
数组存储空间

访问动态分配的数组

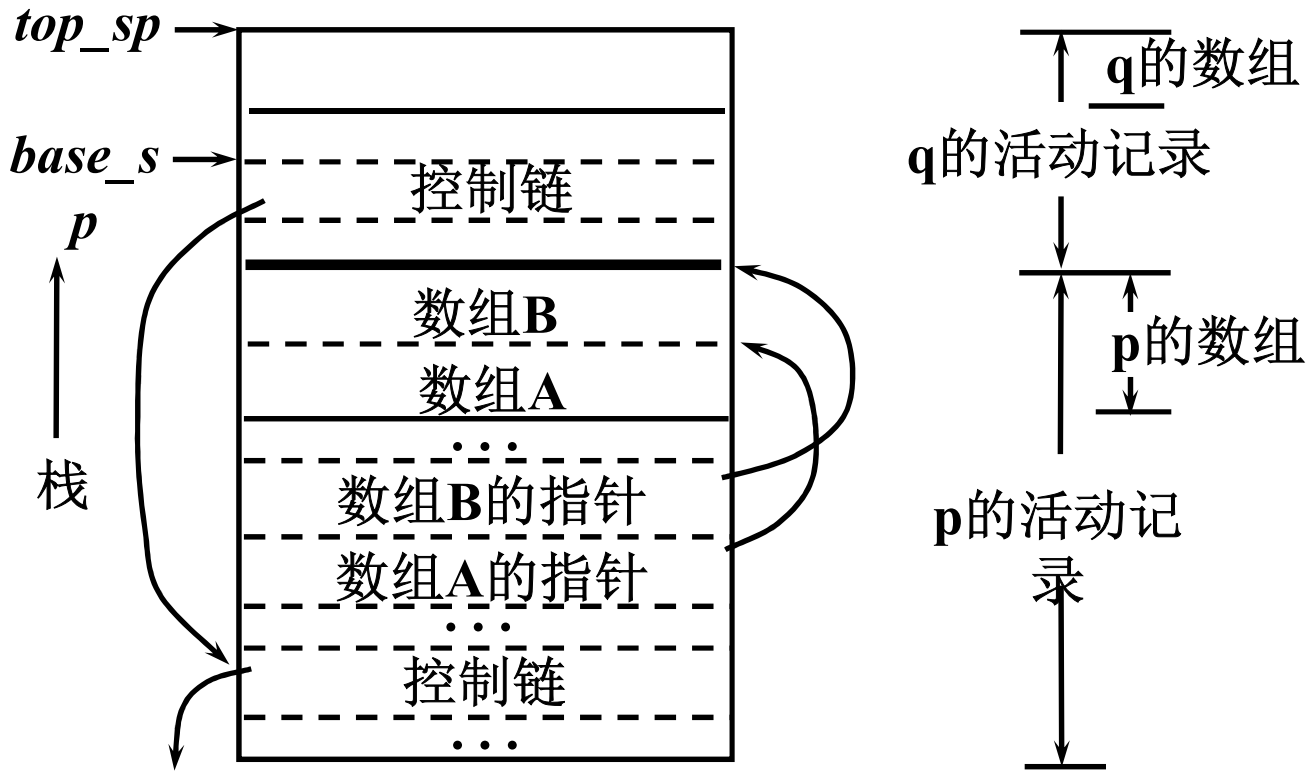
栈上可变长数据



(3) 运行时，对数组A和B的访问都要通过相应指针来间接访问

访问动态分配的数组

栈上可变长数据



访问动态分配的数组

Thank you!