第七章：运行时存储空间管理(1)

运行时存储空间管理

顾名思义，运行时存储空间的管理，是指目标程序运行时存储空间是如何管理的

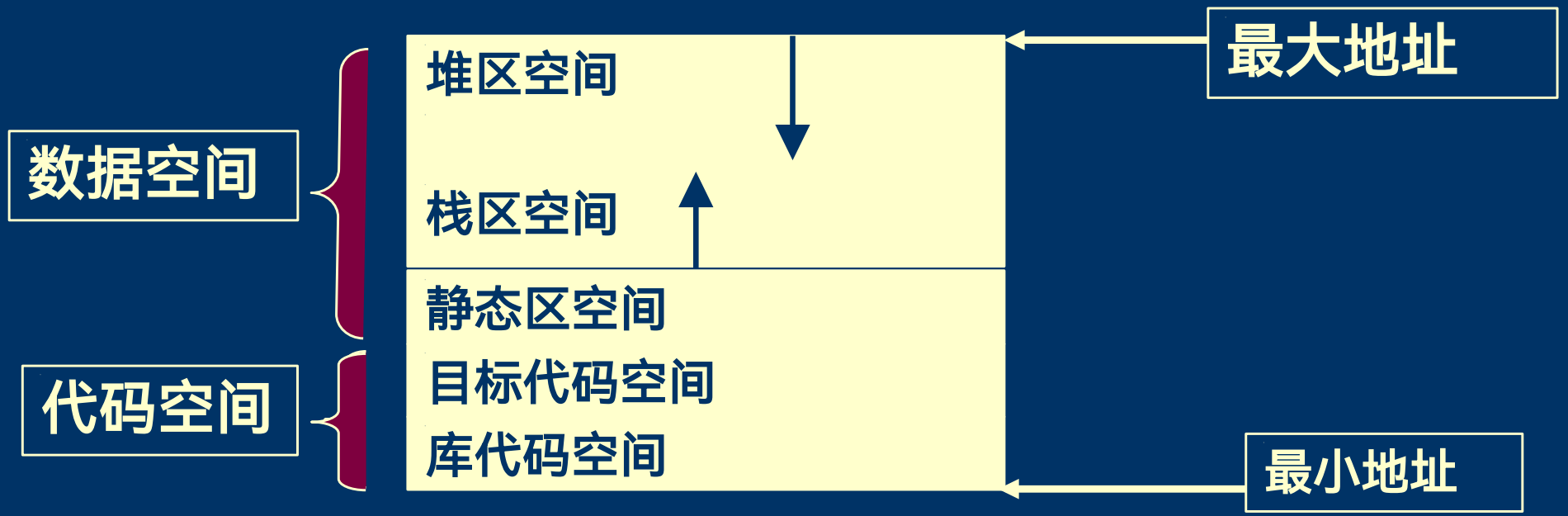
- 本章中要解决两个问题：

运行时空间是怎么管理的

存储空间是如何访问的

目标程序运行时内存的划分

目标程序占用的存储空间如图所示：



目标程序运行时内存的划分

- 库代码区（libaray space）：用于存放标准库函数的目标代码，例如pascal语言中用到的sin、cos等等。高级函数，如C语言中通过#include包含进来的库代码；
- 目标代码区(code space):用以存放编译生成的目标程序。实际上一个目标程序变成可执行文件需要连接-装配的过程，连接装配之后存放在目标代码区。

目标程序运行时内存的划分

- 静态区(static space) :有些数据对象所占用的空间也能在编译时确定，其地址可以编译进目标代码中，这些数据对象通常存放在静态区中，如静态变量和全程变量
- 栈区（stack space）：存放过程活动记录，该存储区被称作一个栈，一个元素是一个过程活动记录，调用函数时压栈，函数结束时退栈

目标程序运行时内存的划分

- 堆区（heap space）：堆不是一个连续分配的模式，可以进行动态分配的空间管理，主要用于存放动态申请的数据对象 （如C，pascal ，Lisp等语言的malloc,calloc,free,new,delete）

- 栈区和堆区之间没有事先划好的界线，当目标代码运行时，栈区指针和堆区指针不断地变化，并朝着对方方向不断增长。如果这两个区相交，则表示出现了内存溢出。

语言影响空间分配策略

- 语言中是否允许函数、过程的递归调用

语言如果允许递归调用，则函数的形参、局部量可能对应是一串存储单元，不能采用静态的存储分配方式；如语言不允许有递归出现，则一个函数最多只能分配一个活动记录大小的空间。

语言影响空间分配策略

- 当一个函数结束的时局部变量是否需要保存。

一般来说，一个函数的局部变量在函数调用结束的时候所占用的存储空间都要被释放掉，但是有一类语言的特殊语句，允许函数中的某个局部变量保存着，下一次再调用这个函数的时候，这个局部变量是已经具有上一次语言的值。

语言影响空间分配策略

- 是否可以访问非局部的变量

比方说外层的变量，还有其他的变量是否可以访问，不同的语言也有不同的约定

- 函数参数的传递方式

常用的参数传递方式是值引用和地址引用，这就决定了地址引用型的实参只能是一个变量而不能是一个常数，形参的值引用和地址引用的空间分配的方式是不一样的。

语言影响空间分配策略

- 函数是否可以作为参数进行传递
- 函数的结果是否可以为函数
- 动态的申请存储块
- 是否显示的释放存储空间

存储管理模式的分类

- 静态存储分配策略
- 动态存储分配策略
  - 栈式动态存储分配策略
  - 堆式动态存储分配策略

静态分配策略在编译时为数据对象分配固定的存储空间，且存储对象的存储位置在程序的整个生命周期是固定的。

静态存储分配实例

临时变指量
地址分配

```
int sum = 0;
int square(int i)
{
    int j;
    j = i * i;
    return j;
}
void main()
{
   int j;
    j =square(n);
    sum=sum + j;
}
```

(assign, 0, _ , sum)

(ENTRY, L_{sqr}, 3+off_1, 1)

(*,i,i,t_1)

(assign, t_1,_, j)

(RETURN, _, _, j)

(ENDFUN, _, _, _)

(ENTRY, L_{main}, 3+off_2, 1)

(VALACT, n, off_i, 1)

(call, square, true,t_2)

(assign, t_2, 1, j)

(+, sum, j, t_3)

(assign, t_3, 1, sum)

(ENDFUN, _, _, _)

目标代码
sum
常量信息表等...
square的控制信息
...
i
j
t_1
main的控制信息
.....
j
t_2
t_3

静态存储分配

- 完全采用静态分配策略的语言必须满足以下约束条件:

1. 不允许递归过程。
2. 不允许可变体积的数据，即数据对象的长度和它在内存中位置的限制，必须是在编译时可知。
3. 不允许动态建立的数据结构（如动态数组、指针等），因为没有运行时的存储分配机制。

动态存储分配

- 动态存储分配，可以分为：
  - 栈式。是一种最常见的模式，在有函数调用的时候动态的分配存储空间；

若程序运行过程中有动态进行申请和释放程序空间，用栈式的就不行了,没有办法静态的在活动记录里给分配空间

  - 栈堆混合式，活动记录的大小确定的存在栈区中，动态申请的空间在堆区中申请然后释放，同时增加一部分对堆区的管理程序

栈式管理中的过程活动记录

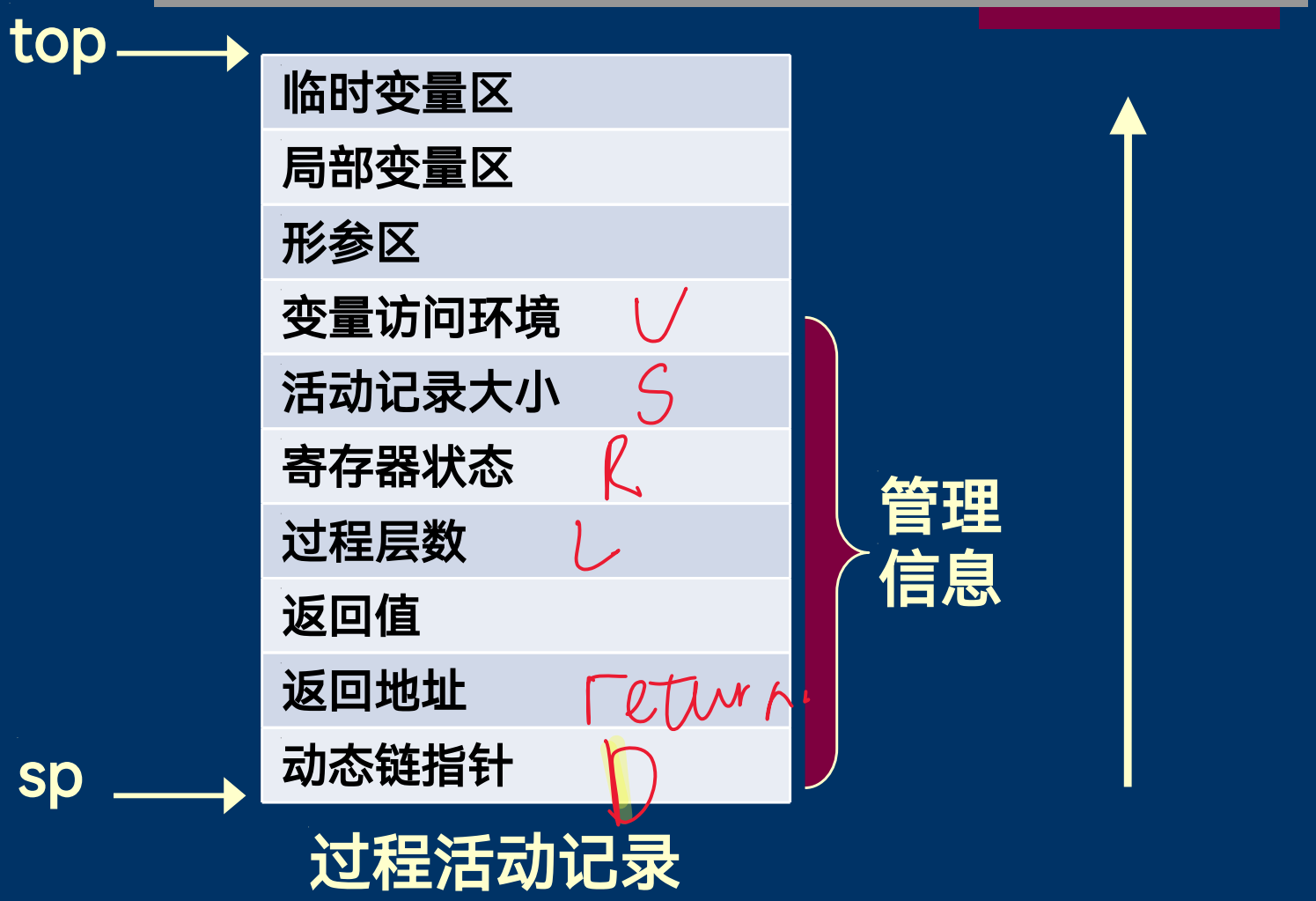
- 过程活动记录的是栈式管理中最重要的内容
- 栈区中通常需要设立两个指针：
  - sp 指向当前活动记录的起始位置
  - top指向第一个可用的存储单元
  - 通常存在寄存器中

栈式管理中的过程活动记录

- 过程的活动记录：

  为管理过程、函数的一次活动所需要的信息，目标程序要在栈区中给被调过程分配一段连续的存储空间，以便存放该过程的局部变量值、控制信息和寄存器内容等，称这段连续的存储空间为过程的活动记录，简称活动记录（Activation Record），并记为AR。

栈式管理中的过程活动记录



地址分配原则回顾

- 值引用的形参按照类型大小分
- 地址引用的形参分配1
- 局部变量按照类型长度分
- 临时变量分1
- 函数和过程作为形参分2，其中1个是实参的入口地址，另一个是先行的display表地址

实例分析

```
proc p()
type at=array[1..100] of
          array[1..10] of integer
var x:real; a:at; i:integer;
proc p1(var a1:at;   a2:at)
   var x:integer; a:real;
    proc p2(n:integer)
          var m:1..50; x:real;
           end
    end
end
```

```
int sum;
int f(int n){
    int k;
    if n>1 return n*f(n-1)
    else return 1;
}
void main(){
    int a[5],i;
    读入a的值;
    for (i=0;i<5;i++){
    输出f(a[i]);
}
```

过程活动记录的申请和释放

- 遇到函数过程调用时申请地址，具体来看在遇到call四元式中间代码时，要生成相应的目标代码。要做的工作有两个：
  - 产生一个新的活动记录，即sp=top，top=top+size
  - 填写过程活动记录的管理信息，返回地址、寄存器内容、动态链指针等等
- 释放一个是遇到了return，一个是遇到了函数的结束要做的主要工作有：
  - 恢复现场，将寄存器里的值恢复
  - 释放当前活动记录即top=sp; sp=动态链指针
  - 根据返回地址创建跳转指令

过程活动记录的申请和释放

调用链：调用链是过程名的序列，序列的头是主程序名M（pascal主程序、C语言的main函数）。具体地说:

- ※（M）是调用链;
- ※若（M，…，R）是调用链，并且R中有S的调用，则（M，…，R，S）也是调用链。

对于任一过程（函数）S，其调用链不是唯一的，每个调用链对应于一个动态的过程调用序列。

用CallChain(S)=(M，…，R，S）表示S的调用链，表示当前正在执行的是S的过程体，而M，…，R则是已经开始执行但被中断了的过程。

过程活动记录的申请和释放

动态链：如果当前正在执行的是S，并且 *CallChain*（S）=（M，N,...，R，S），则栈的当前内容可表示为：

[AR（M),AR（N),...，AR（R),AR（S）]，称它为对应调用链（M，N，...，R，S）的动态链。

表示为：

DynamicChains（S）

=[AR（M),AR（N),...，AR（R),AR（S）]

调用链和动态链的示例

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

主程序P ⇒ 过程 S

⇒ 过程 Q ⇒ 过程 R

⇒ 过程 R

调用链

(P)

主程序P 过程 S (P,S)

主程序P 过程 S 过程 Q (P,S,Q)

主程序P 过程 S 过程 Q 过程 R (P,S,Q,R)

主程序P 过程 S 过程 Q 过程 R 过程 R (P,S,Q,R,R)

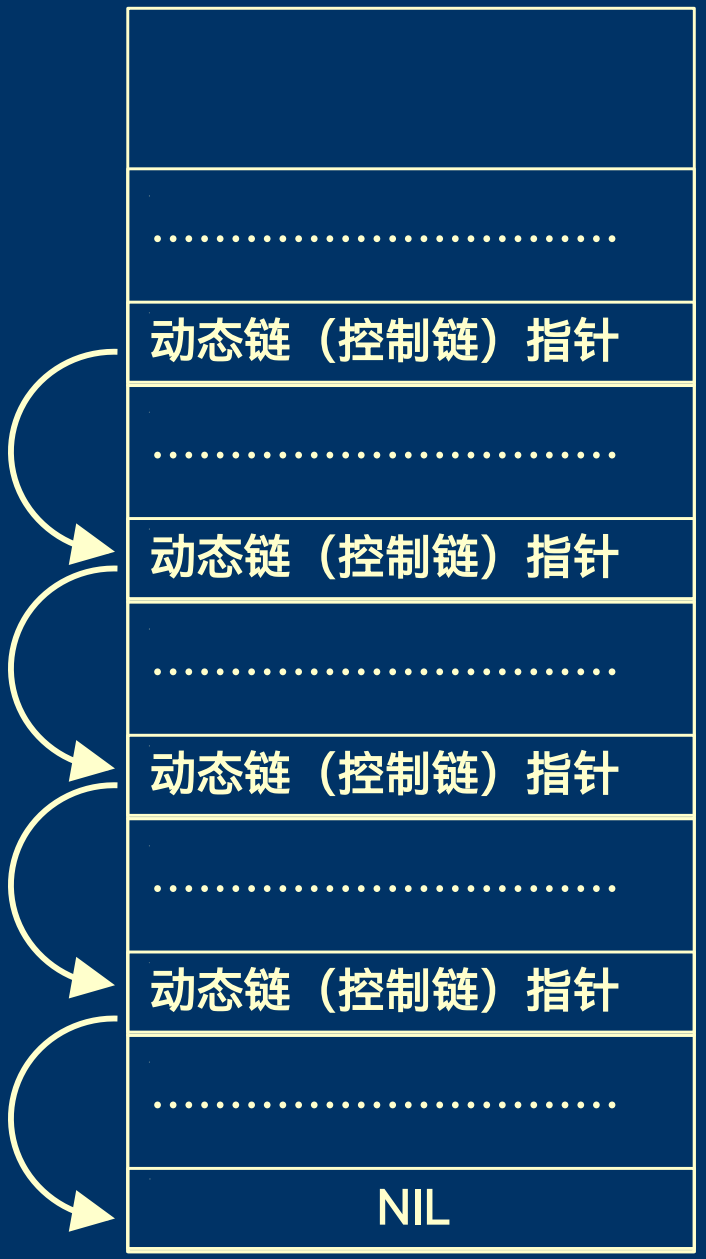
```
procedure S;
    var c, i:integer;
    begin
        a:=1;
        Q(c);
        .....
    end {S}
begin
    a:=0;
    S;
    .....
end. {P}
```

调用链和动态链的示例

	调用链	动态链
主程序P	(P)	[AR(P)]
主程序P 过程 S	(P,S)	[AR(P)， AR(S)]
主程序P 过程 S 过程 Q	(P,S,Q)	[AR(P)， AR(S)， AR(Q)]
主程序P 过程 S 过程 Q 过程 R	(P,S,Q,R)	[AR(P)， AR(S)， AR(Q)，AR(R)]
主程序P 过程 S 过程 Q 过程 R 过程 R	(P,S,Q,R,R)	[AR(P)， AR(S)， AR(Q)，AR(R)，AR(R)]

主程序P ⇒ 过程 S ⇒ 过程 Q ⇒ 过程 R ⇒ 过程 R

空间申请过程



动态链：

[AR(P)， AR(S)， AR(Q)，AR(R)，AR(R)]

[AR(P)， AR(S)， AR(Q)，AR(R)]

[AR(P)， AR(S)， AR(Q)]

[AR(P)， AR(S)]

[AR(P)]

主程序P ⇒ 过程 S ⇒ 过程 Q ⇒ 过程 R ⇒ 过程 R

空间释放过程

执行下述指令:

TOP:=SP

SP:=0[SP]

变量的地址映射

- 并列式语言：

相对简单只有0层和1层，可以用两个指针来解决，sp0指向全局量的首地址，sp指向当前活动记录的首地址，根据变量的层数来确定是sp0+off还是sp+off。

例如int x,y; int f(){int i; y=x*i; return y;}

值得注意的是const变量也在静态区中

变量的地址映射

- 嵌套式语言：相比之下复杂一些

  若抽象地址是L,off,如果L等于当前层，则说明他所对应的变量处于当前的活动记录中，如果L不等于当前层，就要找到他所对应的活动记录的首地址，为此要构造一个display表，把每一个活动记录活跃的静态外层的首地址都存起来，然后找到L所对应的那一层的首地址加上off就可以了