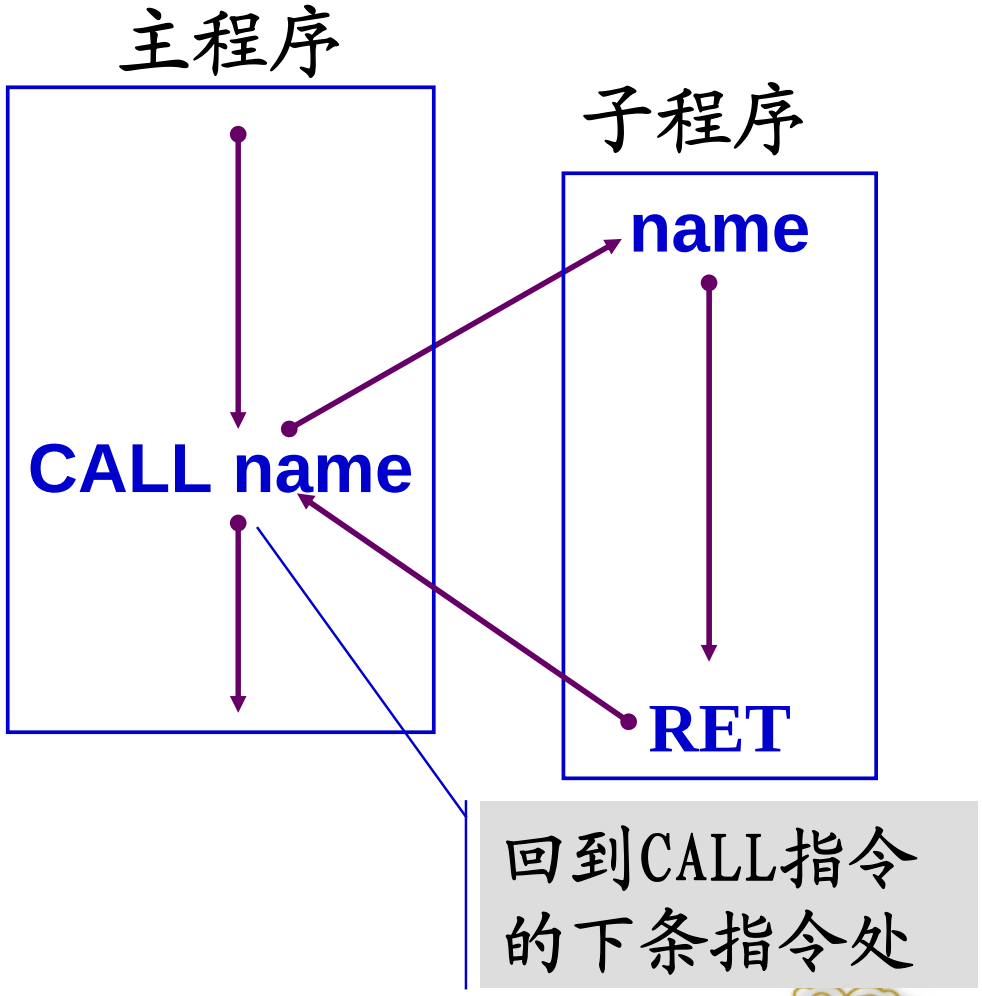
CALL和RET指令

子程序指令

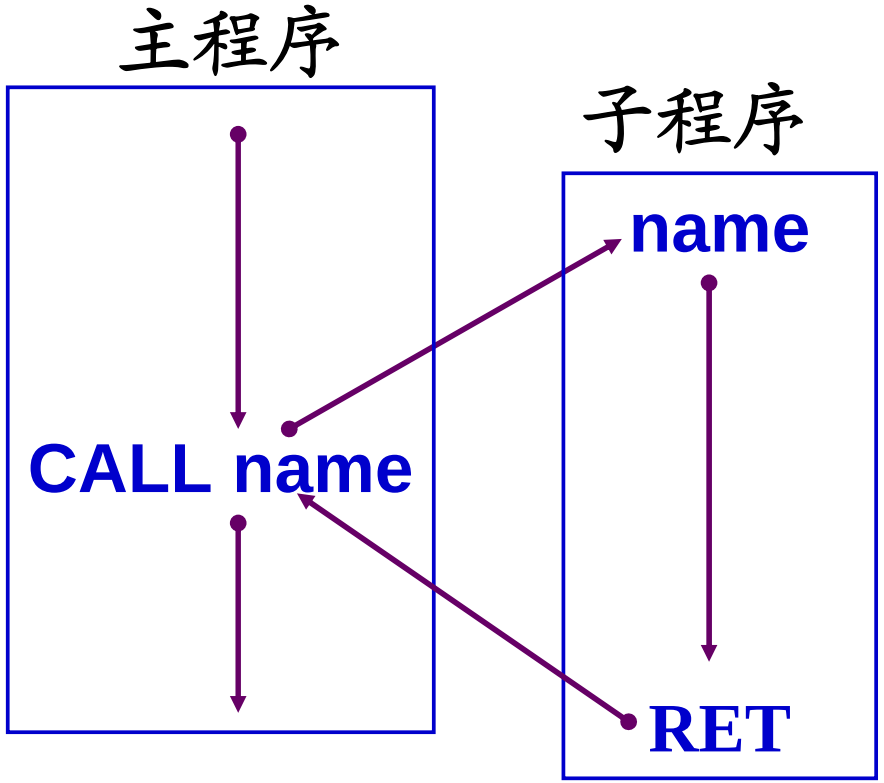
- 主程序(调用程序)
  - 执行调用指令CALL
    调用子程序
- 子程序(被调用程序)
  - 执行返回指令RET
    返回主程序

子程序Subroutine
= 函数Function
= 过程Procedure

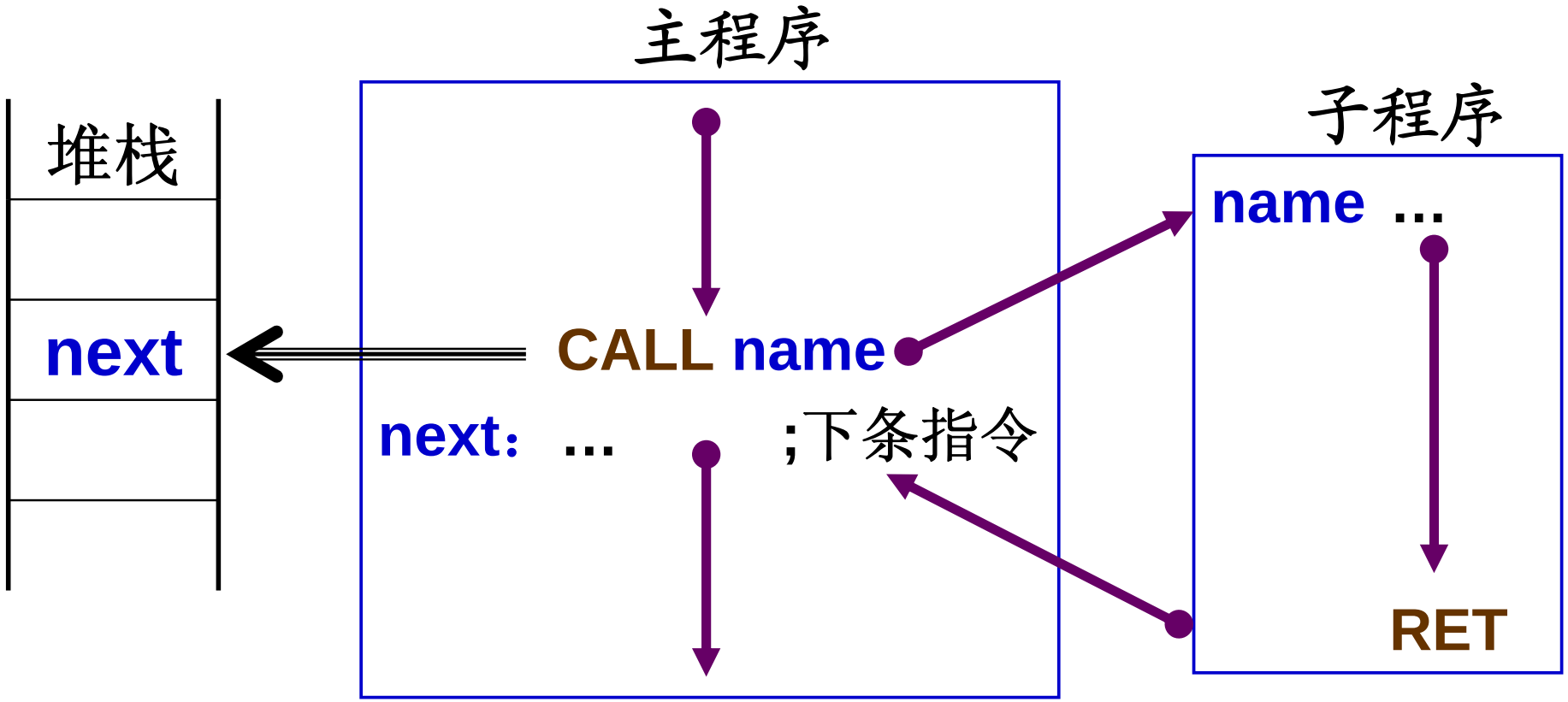


CALL和RET指令的功能

- CALL指令用在主程序中
  实现子程序的调用
  - 功能1：入栈返回地址
  - 功能2：转移到目标地址
- RET指令用在子程序结束
  实现返回主程序
  - 功能1：弹出返回地址
  - 功能2：转移到返回地址



CALL和RET利用堆栈保存返回地址



子程序调用指令CALL

- CALL指令用在主程序中，实现子程序的调用
  - 功能1：将下条指令的地址压入堆栈（顶部）
  - 功能2：转移到目标地址

```
CALL label                    ;调用标号指定的子程序
CALL reg32/reg16              ;调用寄存器指定地址的子程序
CALL mem48/mem32/mem16        ;调用存储单元指定地址的子程序
```

CALL分成段内调用（近调用）和段间调用（远调用）

目标地址支持相对寻址、直接寻址或间接寻址

子程序返回指令RET

- RET指令用在子程序结束，实现返回主程序
  - 功能1：从当前堆栈顶部弹出内容作为返回地址
  - 功能2：转移到返回地址

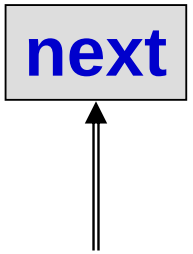
```
RET          ;无参数返回：出栈返回地址
RET i16      ;有参数返回：出栈返回地址，ESP←ESP＋i16
```

调用、返回有段内near和段间far的区别

但是，CALL和RET指令助记符没有区别

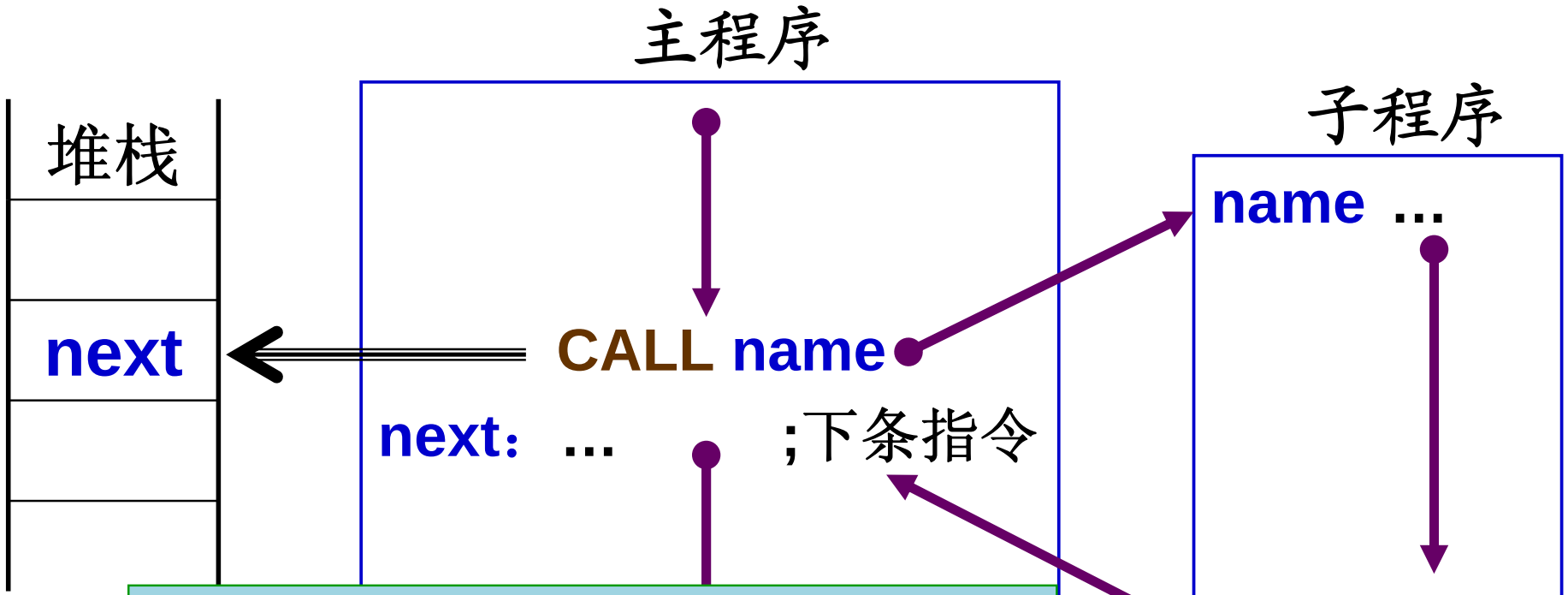
段内CALL和RET指令

```
        CALL label
next:   …
  ;相当于如下两条指令功能
  push next       ;(1) 入栈返回地址：ESP=ESP-4，SS:[ESP]=EIP
  jmp label       ;(2) 转移目标地址：EIP=EIP+偏移量
```




```
        RET       ;栈顶数据出栈到指令指针寄存器EIP
                  ;(1)EIP=SS:[ESP]，ESP=ESP+4
                  ;(2)数据进入EIP，就作为下条要执行指令的地址
```

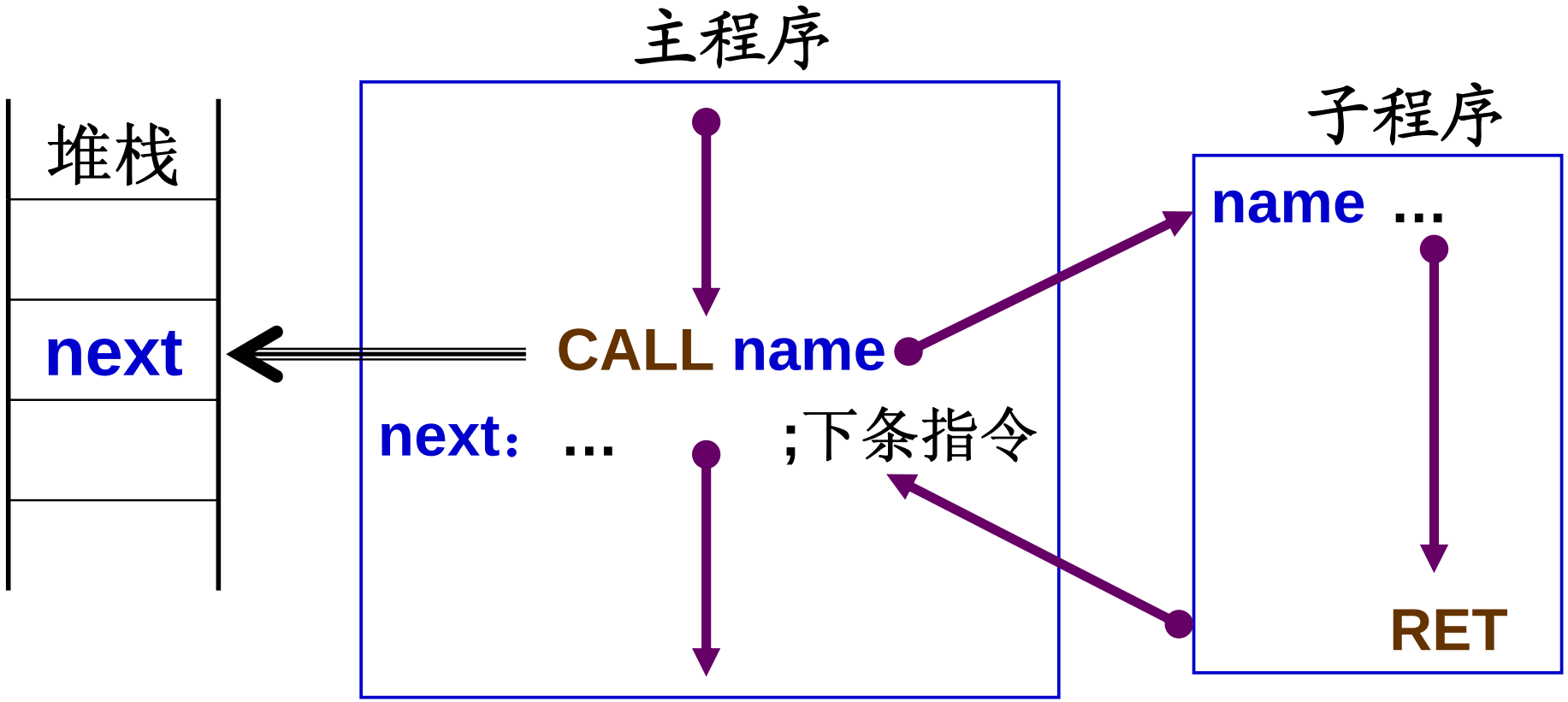
CALL的调用和RET的返回




```
;思考如下代码片段的功能
        call next
next:   pop eax     ;EAX＝？
```

子程序调用

子程序调用CALL和返回RET指令



子程序调用程序—1

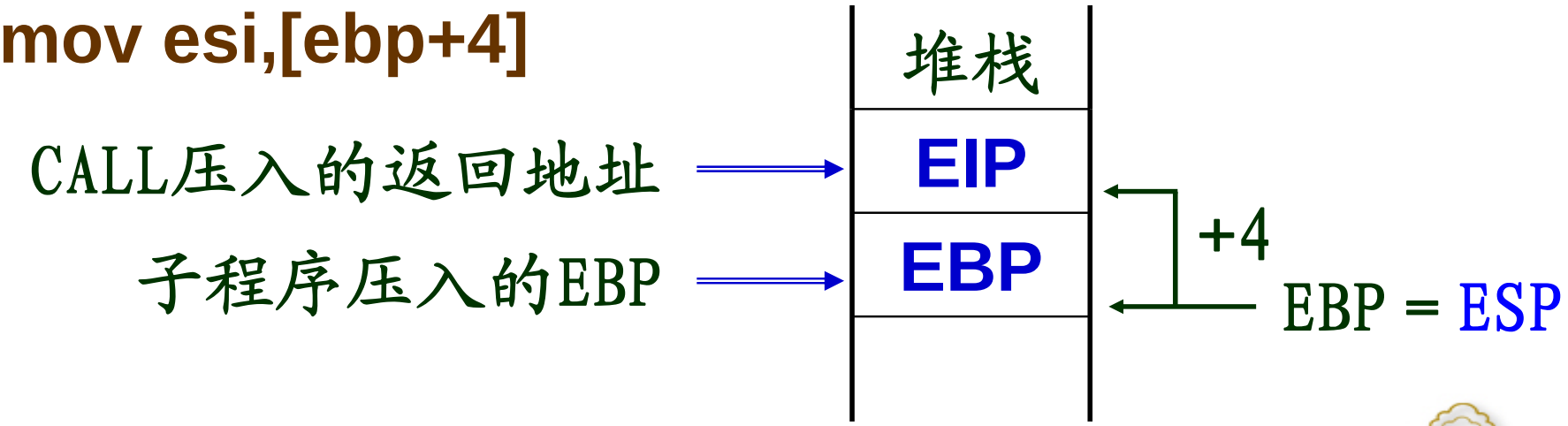
```
;列表文件的地址                    ;代码段，主程序
00000000                          mov eax,1
00000005                          mov ebp,5
0000000A                          call subp          ;子程序调用
0000000F  ⟸          retp1:      mov ecx,3
00000014  ⟸          retp2:      mov edx,4
00000019                          call disprd
```

子程序调用程序—2

```
        ;代码段，子程序
subp    proc            ;过程定义，过程名为subp
        push ebp
        mov ebp,esp
        mov esi,[ebp+4]
        ;ESI＝CALL下条指令(标号RETP1)偏移地址
```

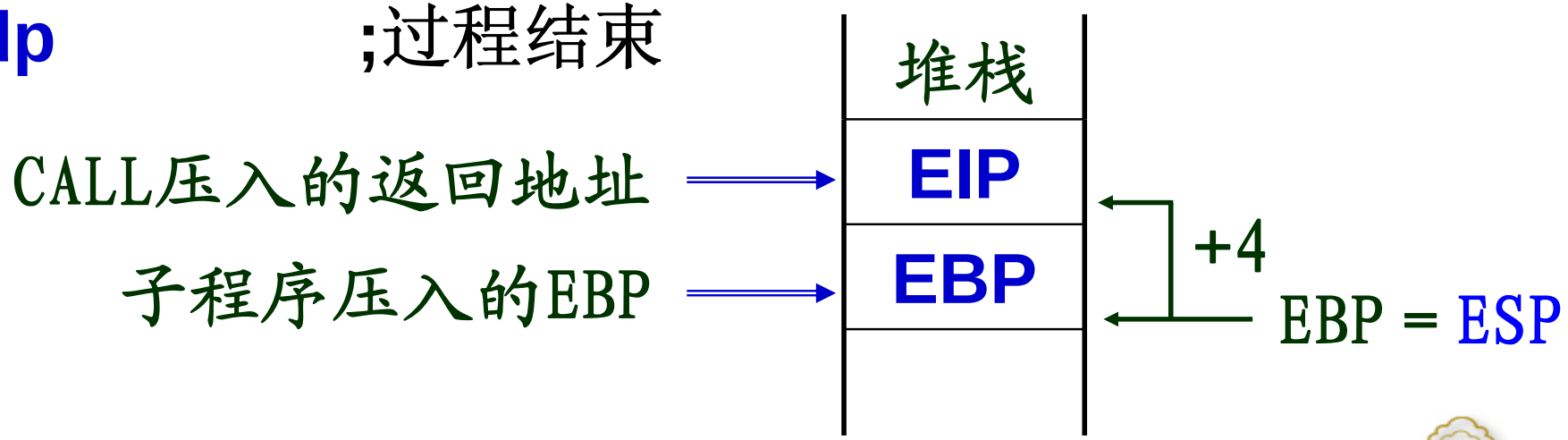
子程序调用程序—2

```
        ;代码段，子程序
subp    proc            ;过程定义，过程名为subp
        push ebp
        mov ebp,esp
        mov esi,[ebp+4]
```



子程序调用程序—3

```
        mov edi,offset retp2
        mov ebx,2
        pop ebp          ;弹出堆栈，保持堆栈平衡
        ret              ;子程序返回
subp endp                ;过程结束
```

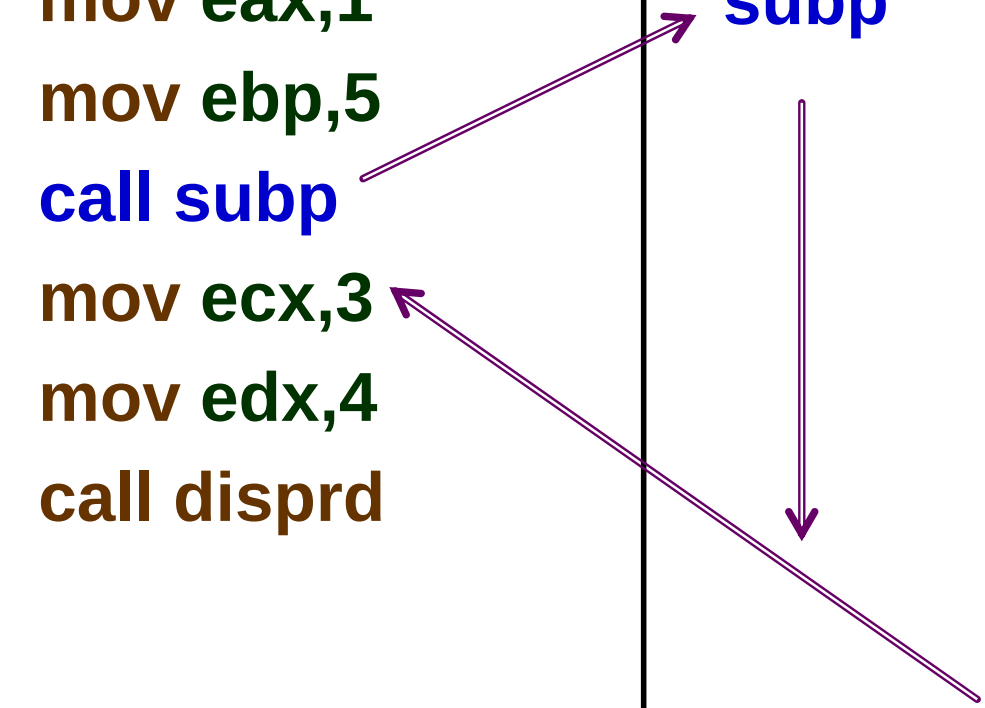


主程序和子程序

```
        ;代码段，主程序
        mov eax,1
        mov ebp,5
        call subp
retp1:  mov ecx,3
retp2:  mov edx,4
        call disprd
```

```
subp    proc
        push ebp
        mov ebp,esp
        mov esi,[ebp+4]
        mov edi,offset retp2
        mov ebx,2
        pop ebp
        ret
subp    endp
```

;代码段，子程序



程序运行结果

```
;列表文件的地址                    ;代码段，主程序
00000000                          mov eax,1
00000005                          mov ebp,5
0000000A                          call subp
0000000F    <==    retp1:         mov ecx,3
00000014    <==    retp2:         mov edx,4
```

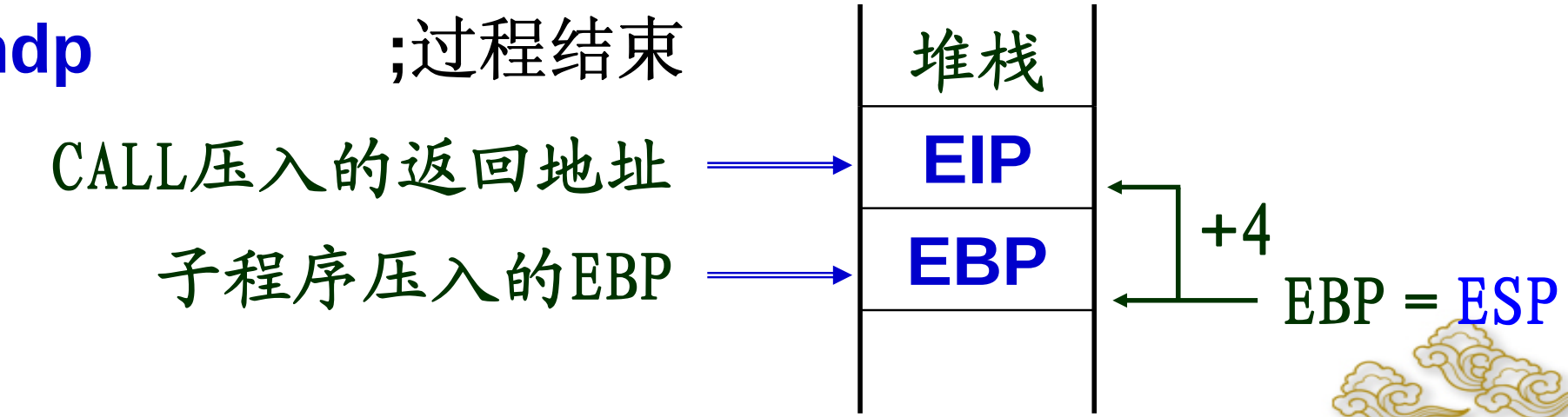
```
D:\MASM>eg0501.exe
EAX=00000001, EBX=00000002, ECX=00000003, EDX=00000004
ESI=0040101F, EDI=00401024, EBP=00000005, ESP=0012FFC4
```

本程序运行时，起始地址：00401010H

子程序调用程序—3a

```
        mov edi,offset retp2
        ; EDI＝标号RETP2的偏移地址
        mov ebx,2
        pop ebp          ;弹出堆栈，保持堆栈平衡
        ret              ;子程序返回
subp endp                ;过程结束
```

MOV [EBP+4],EDI



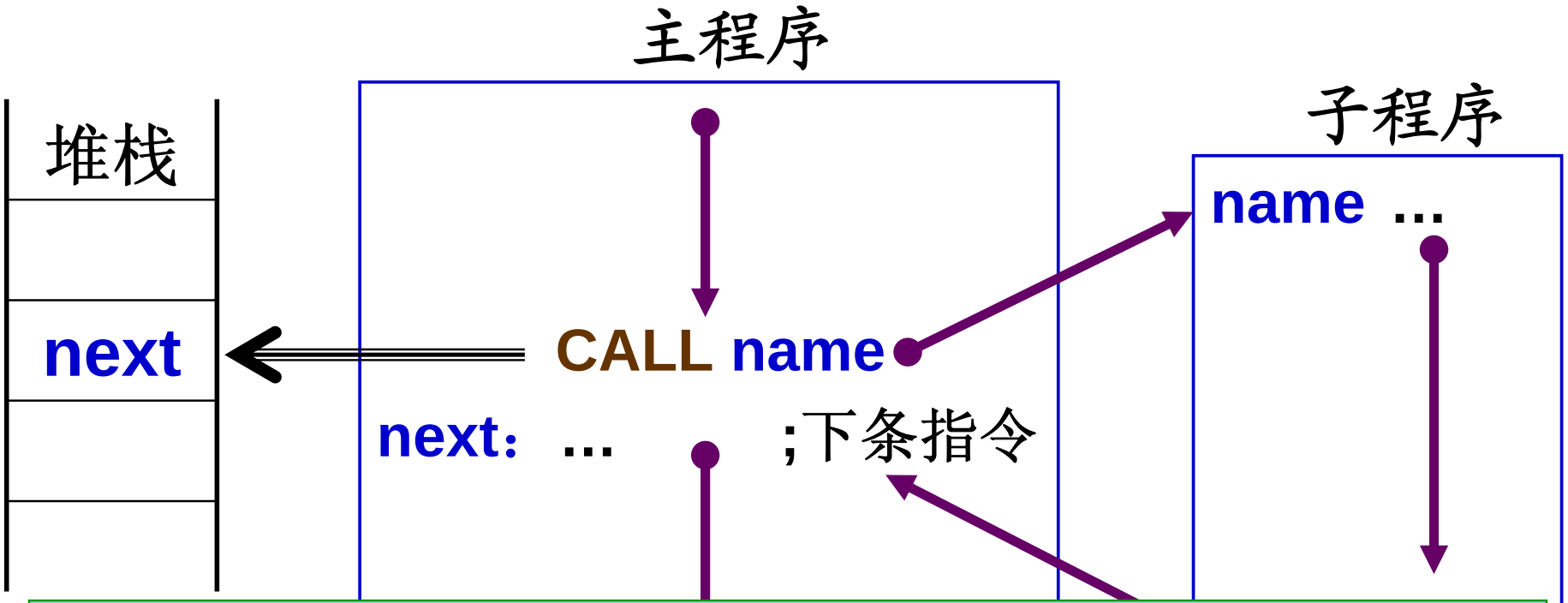
程序运行结果a

```
;列表文件的地址                    ;代码段，主程序
00000000                          mov eax,1
00000005                          mov ebp,5
0000000A                          call subp
0000000F    <==    retp1:         mov ecx,3
00000014    <==    retp2:         mov edx,4
```

```
D:\MASM>eg0501a.exe
EAX=00000001, EBX=00000002, ECX=0012FFB0, EDX=00000004
ESI=0040101F, EDI=00401024, EBP=00000005, ESP=0012FFC4
```

本程序运行时，起始地址：00401010H

CALL和RET利用堆栈保存返回地址




```
        call next
next:   pop eax     ;执行后，EAX获得标号next的地址
                    ;也就是当前指令的地址
```

子程序设计

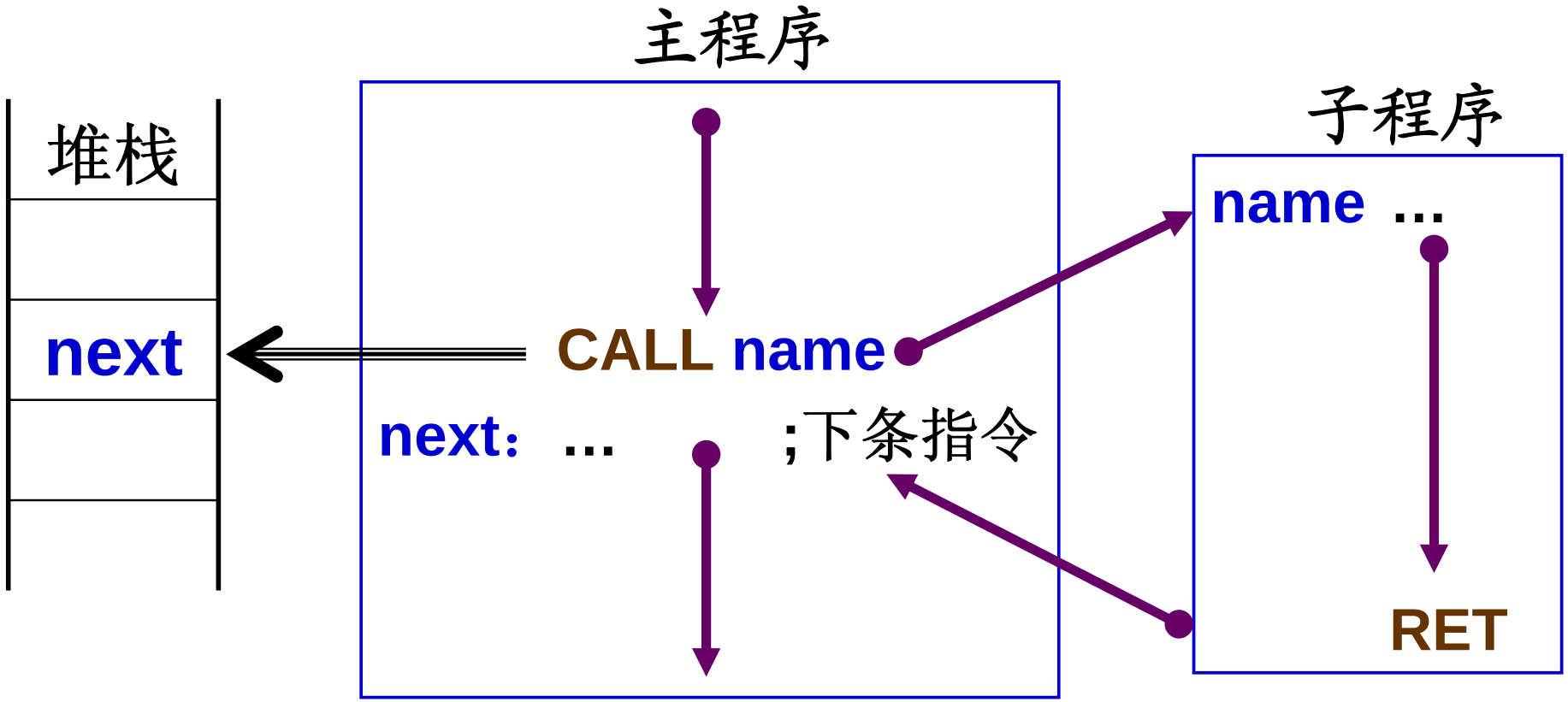
子程序设计

- 子程序的编写方法与主程序一样
- 但需要留意几个问题：
  - ✓RET指令返回主程序，CALL指令调用子程序
  - ✓利用过程定义，获得子程序名和调用属性
  - ✓压入和弹出操作要成对使用，保持堆栈平衡
  - ✓子程序开始保护寄存器，返回前相应恢复
  - ✓安排在代码段的主程序之外
  - ✓子程序允许嵌套和递归

应有完整的注释

难点是参数传递

RET指令返回主程序，CALL指令调用子程序



过程定义伪指令

- MASM利用过程定义伪指令获得子程序信息

```
过程名    PROC
          …        ;过程体
过程名    ENDP
;过程名为符合语法的标识符
```

MASM会根据存储模型等信息确定子程序的远近调用，并相应产生调用、返回指令

子程序框架

```
标识符    proc          ;过程定义（子程序开始）
          push ...1     ;保护寄存器
          push ...2
          ...           ;子程序体
          pop ...2      ;恢复寄存器
          pop ...1
          ret           ;子程序返回
标识符    endp          ;过程（子程序）结束
```

回车换行功能

```
mov al,0dh    ;输出回车字符
call dispc
mov al,0ah    ;输出换行字符
call dispc
;子程序中调用子程序，实现子程序嵌套
```

编写成子程序

回车换行子程序

```
dpcrlf   proc          ;回车换行子程序
         push eax      ;保护寄存器
         mov al,0dh    ;输出回车字符
         call dispc
         mov al,0ah    ;输出换行字符
         call dispc
         pop eax       ;恢复寄存器
         ret           ;子程序返回
dpcrlf   endp          ;子程序结束
```

子程序设计

- 子程序的编写方法与主程序一样
- 但需要留意几个问题：
  - ✓RET指令返回主程序，CALL指令调用子程序
  - ✓利用过程定义，获得子程序名和调用属性
  - ✓压入和弹出操作要成对使用，保持堆栈平衡
  - ✓子程序开始保护寄存器，返回前相应恢复
  - ✓安排在代码段的主程序之外
  - ✓子程序允许嵌套和递归

应有完整的注释

难点是参数传递