操作系统 Operating Systems

L20 内存使用与分段

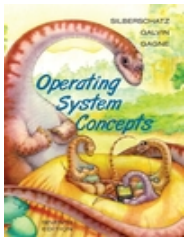
Memory and Segmentation

授课教师：李治军

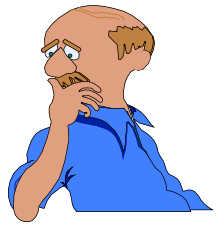
lizhijun_os@hit.edu.cn

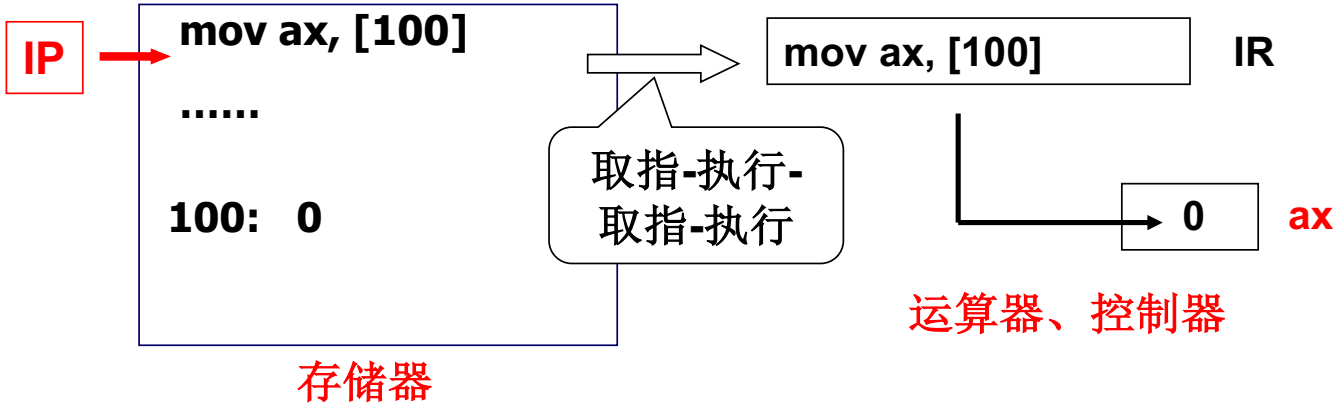
综合楼411室

如何让内存用起来?

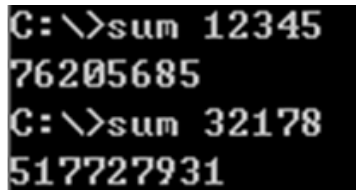


仍然从计算机如何工作开始…

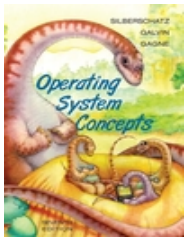





```
int main(int argc, char* argv[])
{   int  i , to, sum = 0;
    to = atoi(argv[1]);
  ...
```



- 内存使用：将程序放到内存中，PC指向开始地址

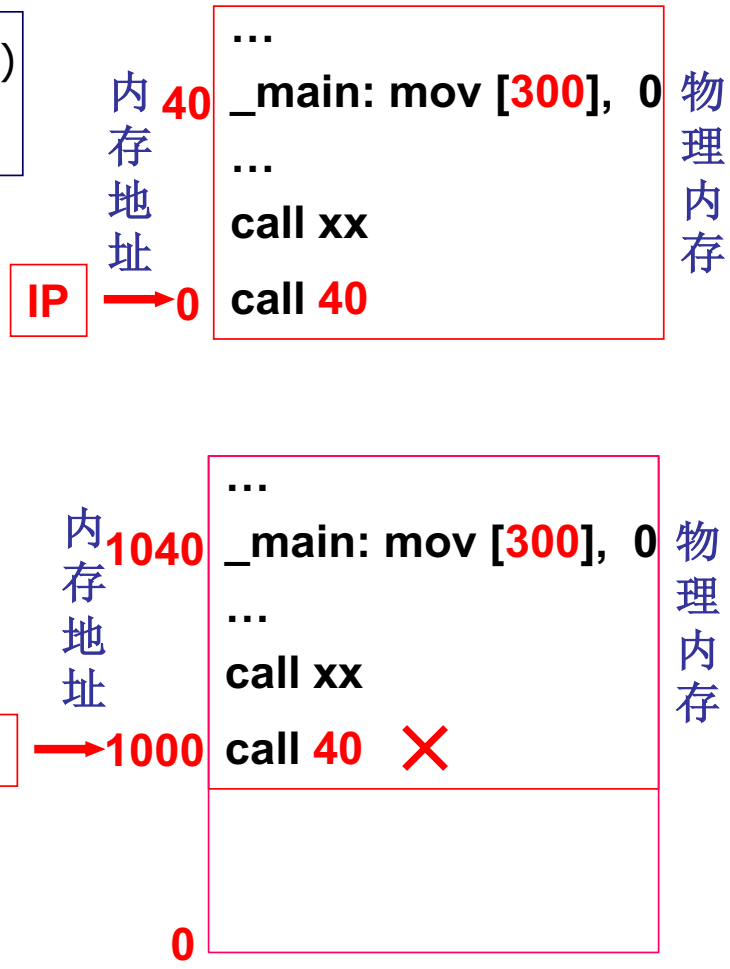


那就让首先程序进入内存

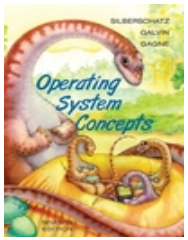
```
int main(int argc, char* argv[])
{ …
```

```
.text
_entry: //入口地址
    call _main
    call _exit
_main:
    …
    ret
```

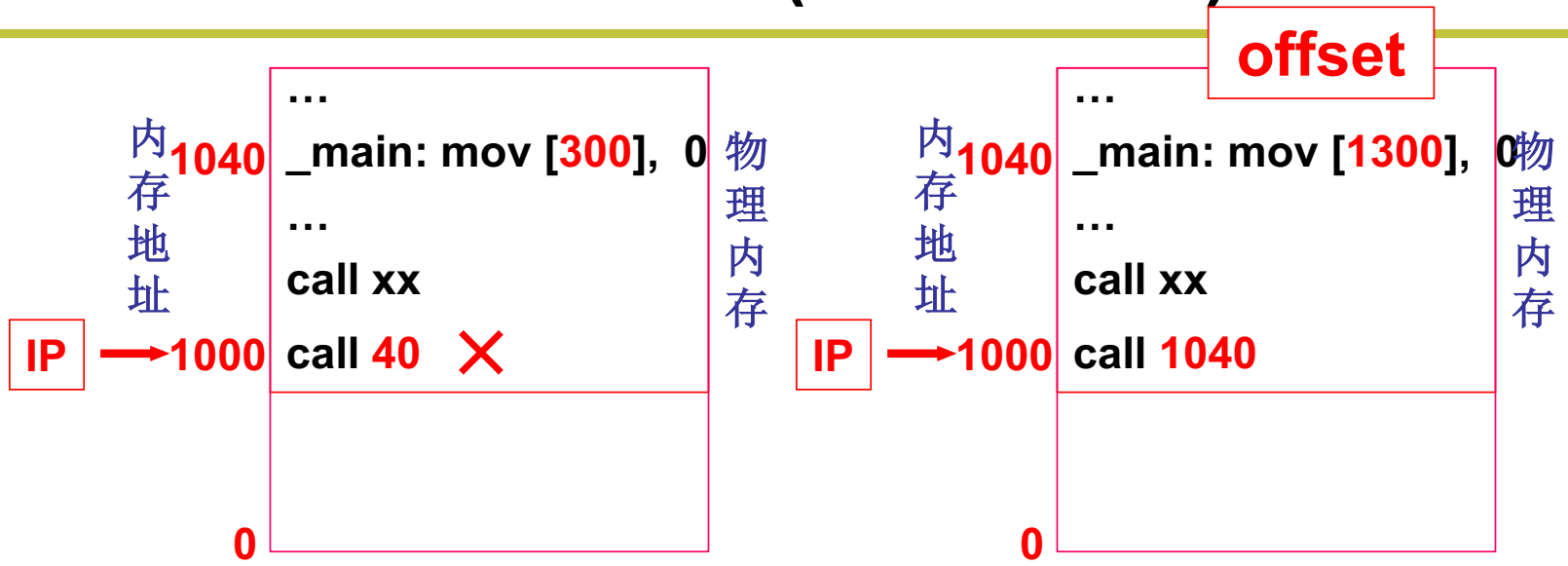
```
_entry: //入口地址
    call 40
    call xx
_main: //偏移是40
    …
```



问题：现在内存是可以使用了，但是…



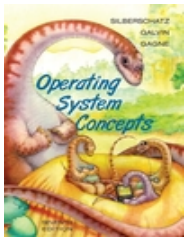
重定位: 修改程序中的地址(是相对地址)



- 是么时候完成重定位? 编译时　载入时

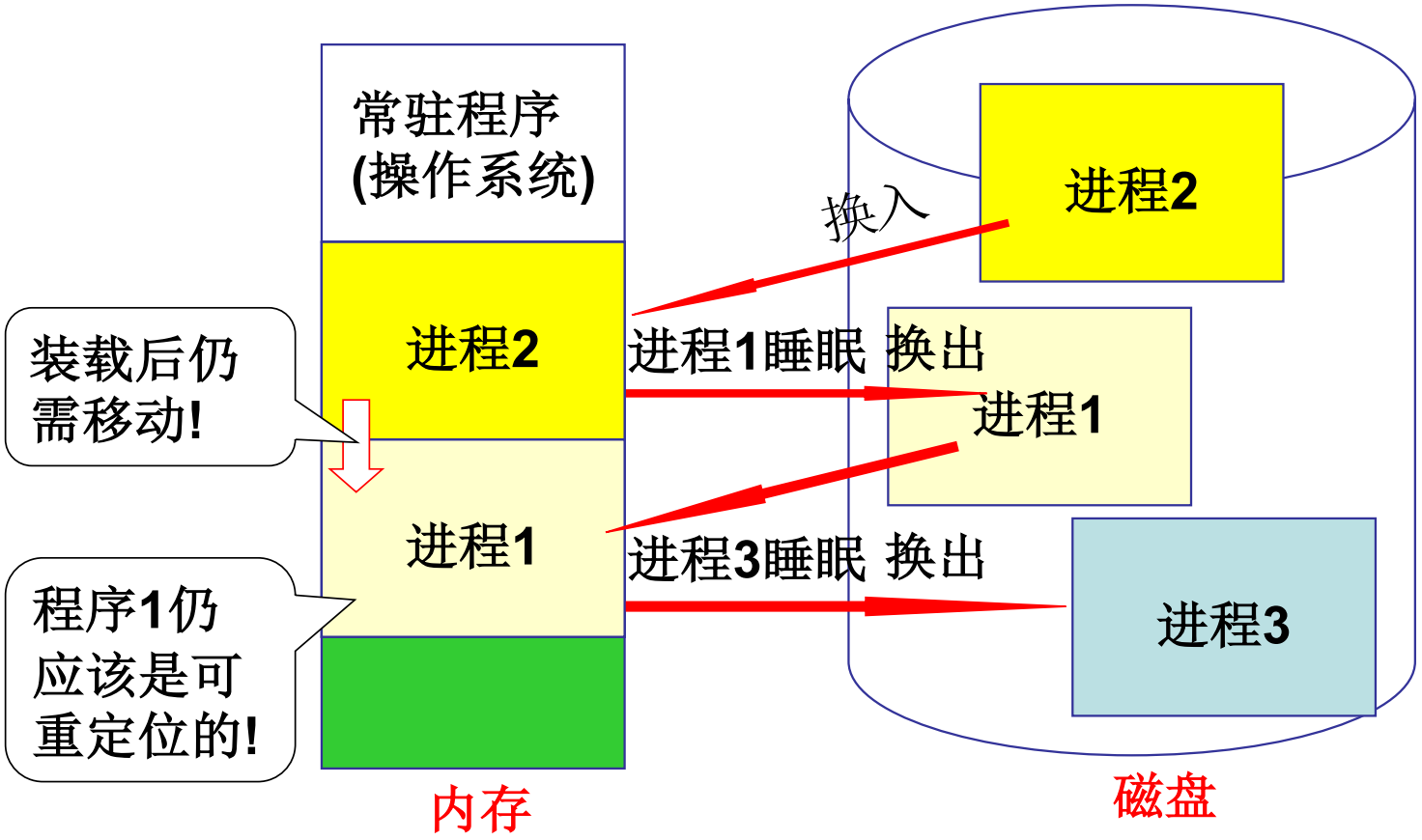
  问题：两种定义各自有什么特点?

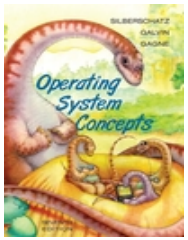
  - 编译时重定位的程序只能放在内存固定位置
  - 载入时重定位的程序一旦载入内存就不能动了



程序载入后还需要移动...

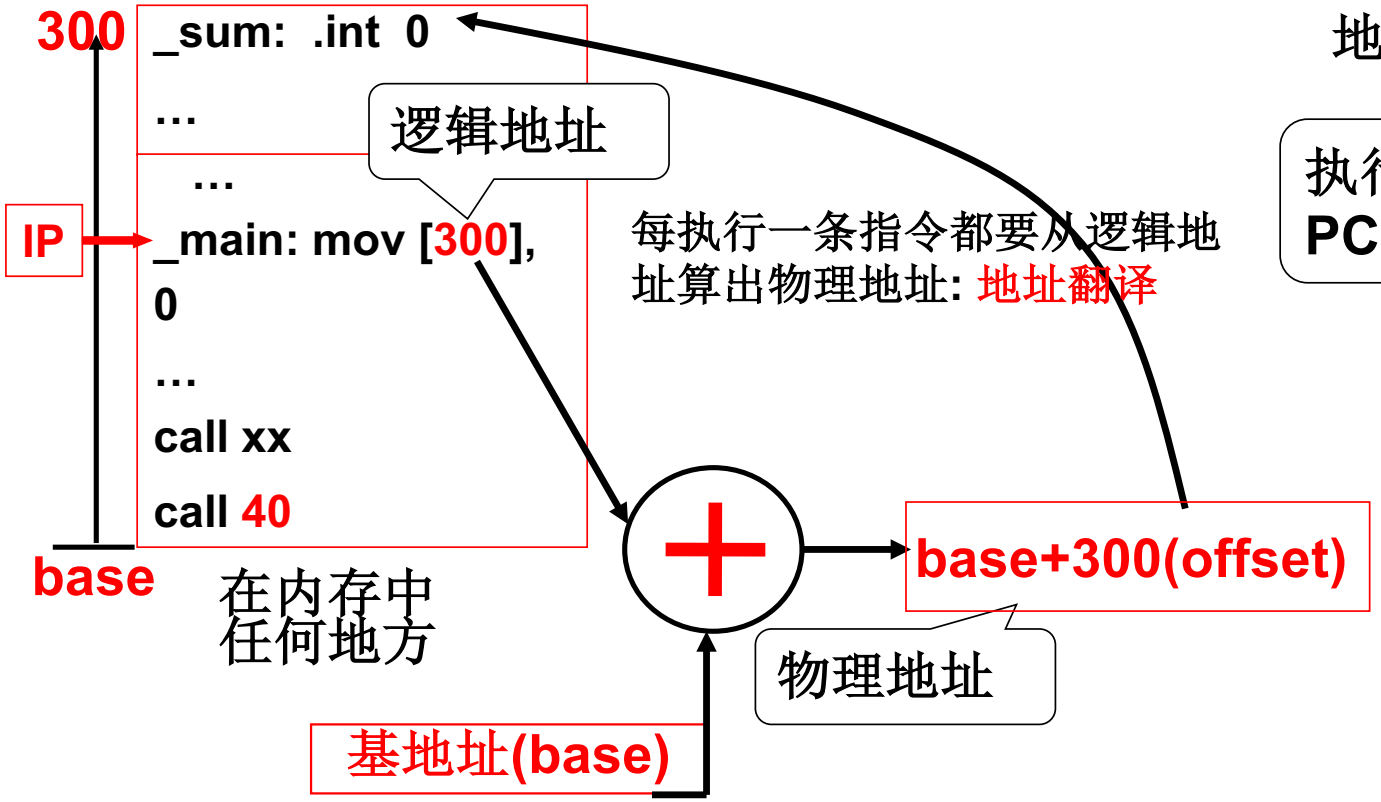
- 一个重要概念: 交换(swap)　充分利用内存





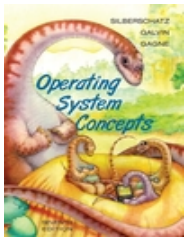
重定位最合适的时机 – 运行时重定位

- 在运行每条指令时才完成重定位

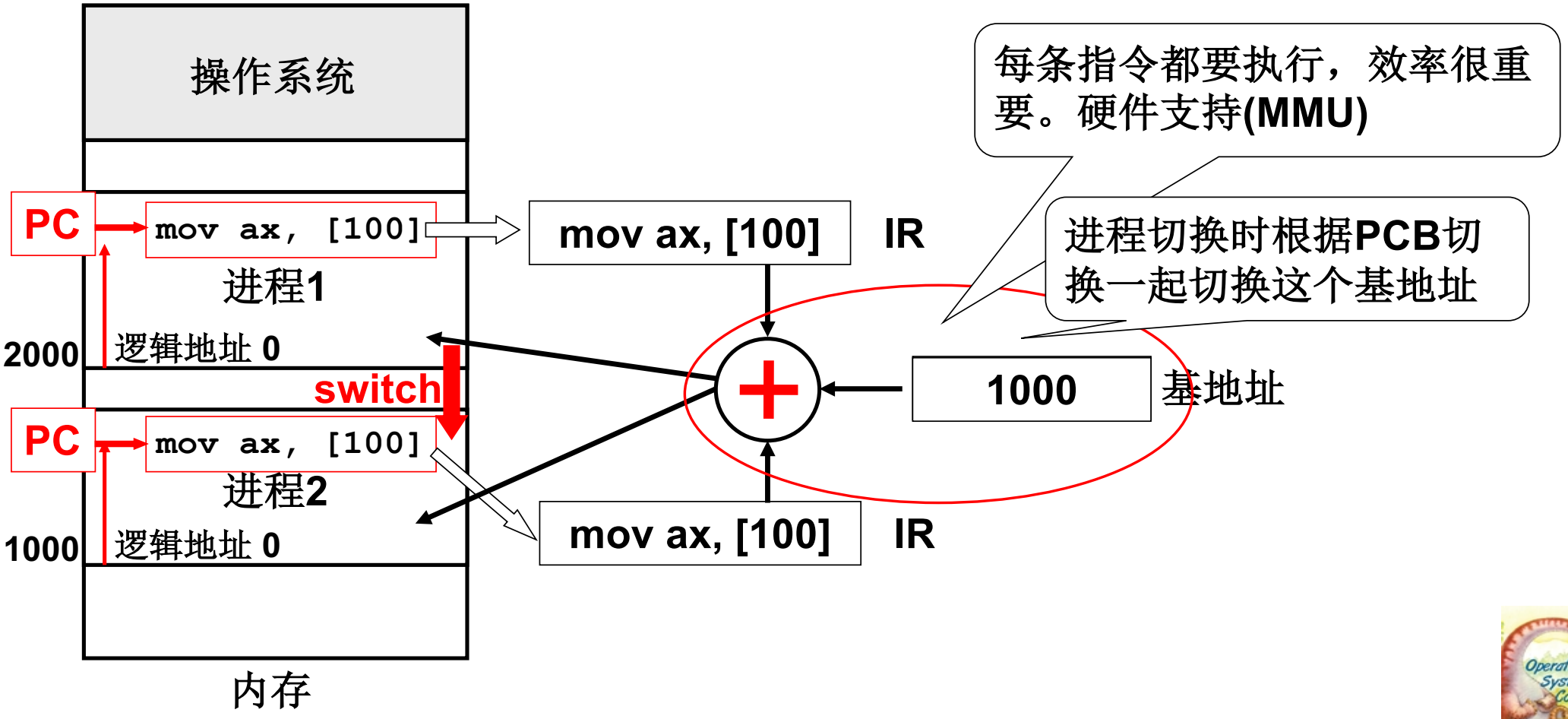


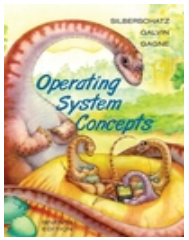
- 每个进程有各自的基地址，放在哪里？ PCB

执行指令时第一步先从PCB中取出这个基地址

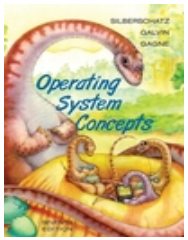


整理一下思路...


操作系统
PC
mov ax, [100]
进程1
2000
逻辑地址 0
switch
PC
mov ax, [100]
进程2
1000
逻辑地址 0
内存
mov ax, [100]
IR
mov ax, [100]
IR
1000
基地址
每条指令都要执行，效率很重要。硬件支持(MMU)
进程切换时根据PCB切换一起切换这个基地址

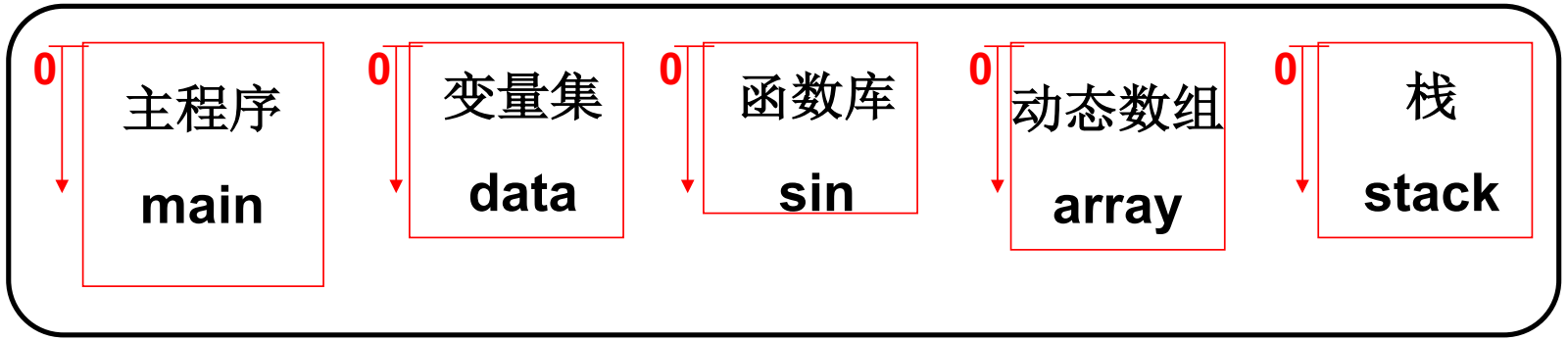

Operating System Concepts

引入分段：是将整个程序一起载入内存中吗？



程序员眼中的程序

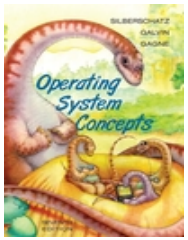
- 由若干部分(段)组成，每个段有各自的特点、用途：代码段只读，代码/数据段不会动态增长…



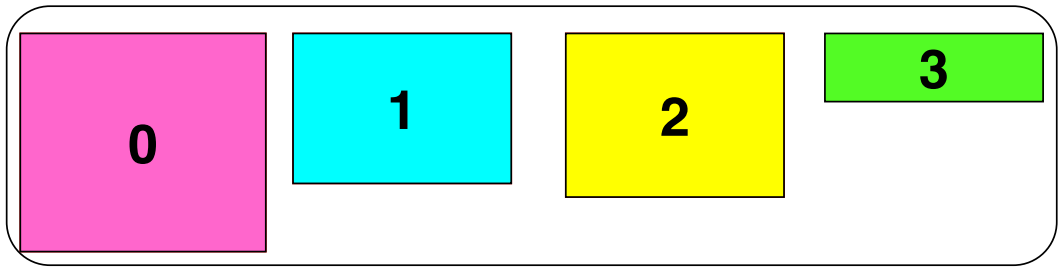
程序员眼中的一个程序

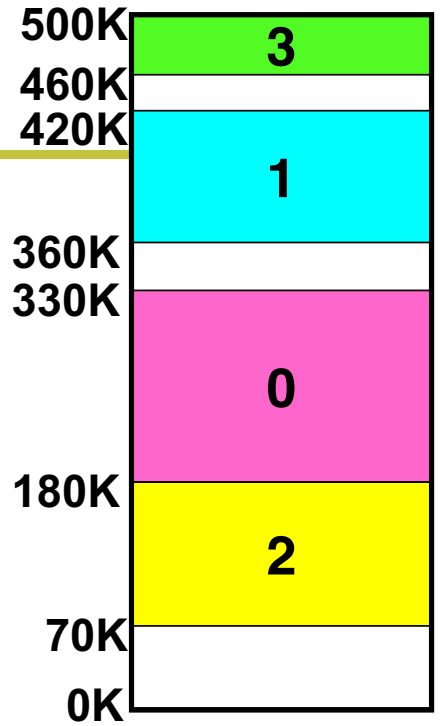
- 符合用户观点: 用户可独立考虑每个段(分治)
- 怎么定位具体指令(数据): <段号, 段内偏移>

如mov [es:bx], ax



不是将整个程序，是将各段分别放入内存





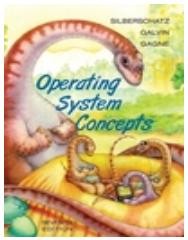
- 再进行运行时重定位会怎么样？

mov [DS:100], %eax　　jmpi 100, CS

进程段表

段号	基址	长度	保护
0	180K	150K	R
1	360K	60K	R/W
2	70K	110K	R/W
3	460K	40K	R

问题：假设DS=1，CS=0，上面两条指令运行时重定位成什么？那么jmp 500K呢？



这个表似曾相识… 真正故事:GDT+LDT