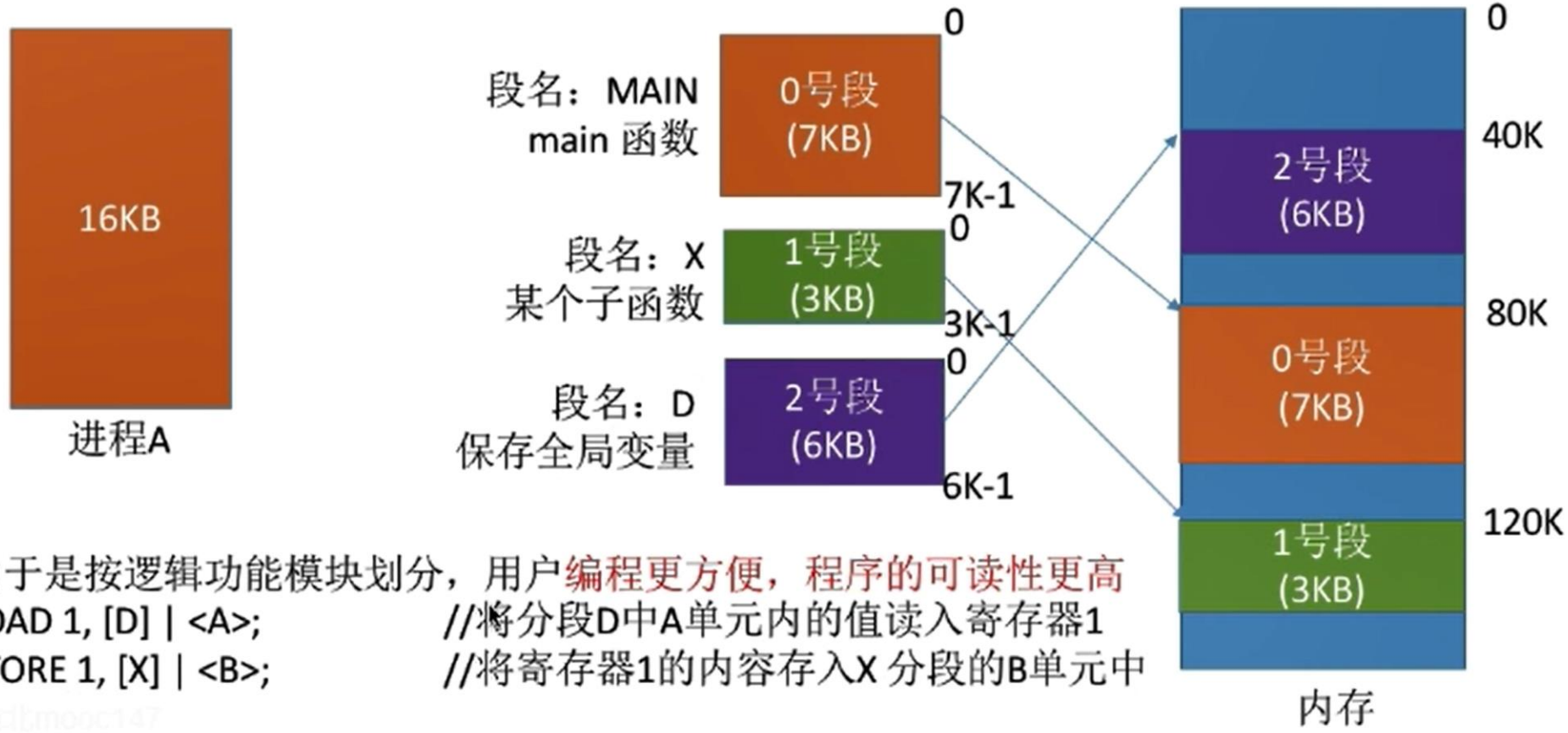
分段

进程的地址空间：按照程序自身的逻辑关系划分为若干个段，每个段都有一个段名（在低级语言中，程序员使用段名来编程），每段从0开始编址

内存分配规则：以段为单位进行分配，每个段在内存中占据连续空间，但各段之间可以不相邻。



由于是按逻辑功能模块划分，用户编程更方便，程序的可读性更高

```
LOAD 1, [D] | <A>;          //将分段D中A单元内的值读入寄存器1
STORE 1, [X] | <B>;         //将寄存器1的内容存入X 分段的B单元中
```

分段

分段系统的逻辑地址结构由段号（段名）和段内地址（段内偏移量）所组成。如：

31 16	15 0
段号	段内地址

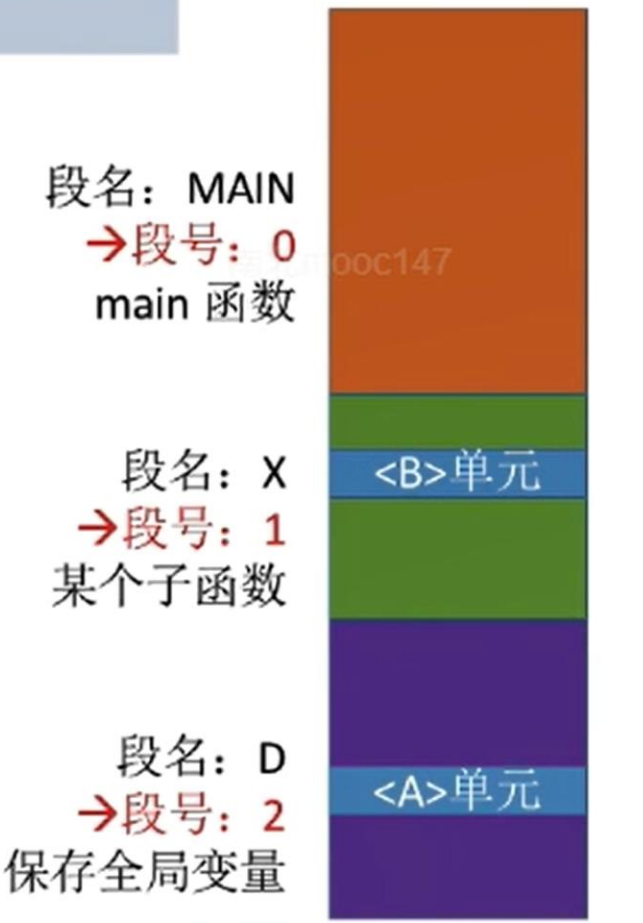
段号的位数决定了每个进程最多可以分几个段
段内地址位数决定了每个段的最大长度是多少

在上述例子中，若系统是按字节寻址的，则
段号占16位，因此在该系统中，每个进程最多有 $2^{16} = 64K$ 个段
段内地址占 16位，因此每个段的最大长度是 $2^{16} = 64KB$。

```
LOAD 1, [D] | <A>;        //将分段D中A单元内的值读入寄存器1
STORE 1, [X] | <B>;       //将寄存器1的内容存入X 分段的B单元中
```

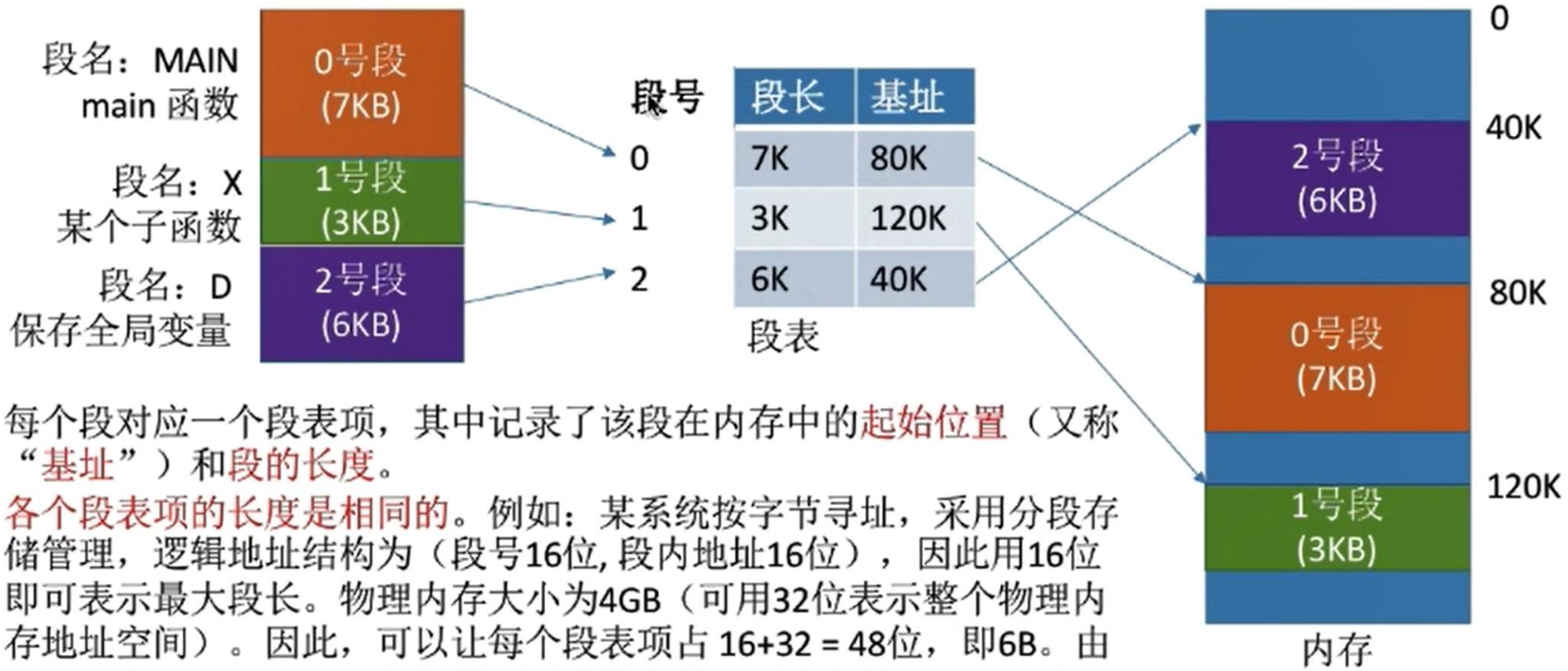
写程序时使用的段名 [D]、[X] 会被编译程序翻译成对应段号

<A>单元、<B>单元会被编译程序翻译成段内地址

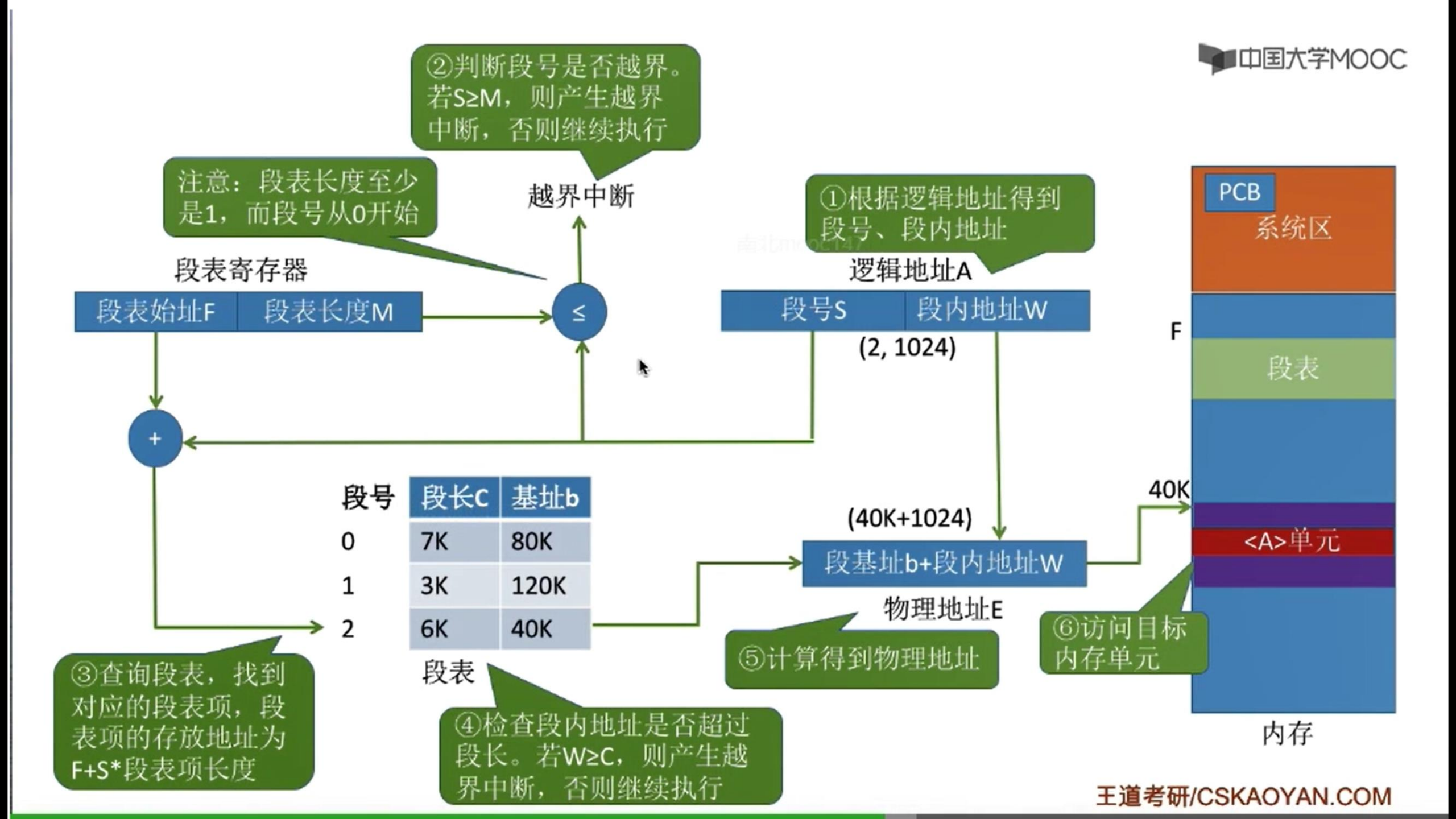


段表

问题：程序分多个段，各段离散地装入内存，为了保证程序能正常运行，就必须能从物理内存中找到各个逻辑段的存放位置。为此，需为每个进程建立一张段映射表，简称“段表”。



1. 每个段对应一个段表项，其中记录了该段在内存中的起始位置（又称“基址”）和段的长度。
2. 各个段表项的长度是相同的。例如：某系统按字节寻址，采用分段存储管理，逻辑地址结构为（段号16位, 段内地址16位），因此用16位即可表示最大段长。物理内存大小为4GB（可用32位表示整个物理内存地址空间）。因此，可以让每个段表项占 16+32 = 48位，即6B。由于段表项长度相同，因此段号可以是隐含的，不占存储空间。若段表存放的起始地址为 M，则 K号段对应的段表项存放的地址为 M + K*6


②判断段号是否越界。若S≥M，则产生越界中断，否则继续执行
越界中断
注意：段表长度至少是1，而段号从0开始
①根据逻辑地址得到段号、段内地址
段表寄存器
段表始址F
段表长度M
≤
逻辑地址A
段号S
段内地址W
(2, 1024)
PCB
系统区
F
段表
+
段号
段长C
基址b
0
7K
80K
1
3K
120K
2
6K
40K
段表
(40K+1024)
段基址b+段内地址W
物理地址E
40K
<A>单元
内存
③查询段表，找到对应的段表项，段表项的存放地址为F+S*段表项长度
④检查段内地址是否超过段长。若W≥C，则产生越界中断，否则继续执行
⑤计算得到物理地址
⑥访问目标内存单元
中国大学MOOC
王道考研/CSKAOYAN.COM

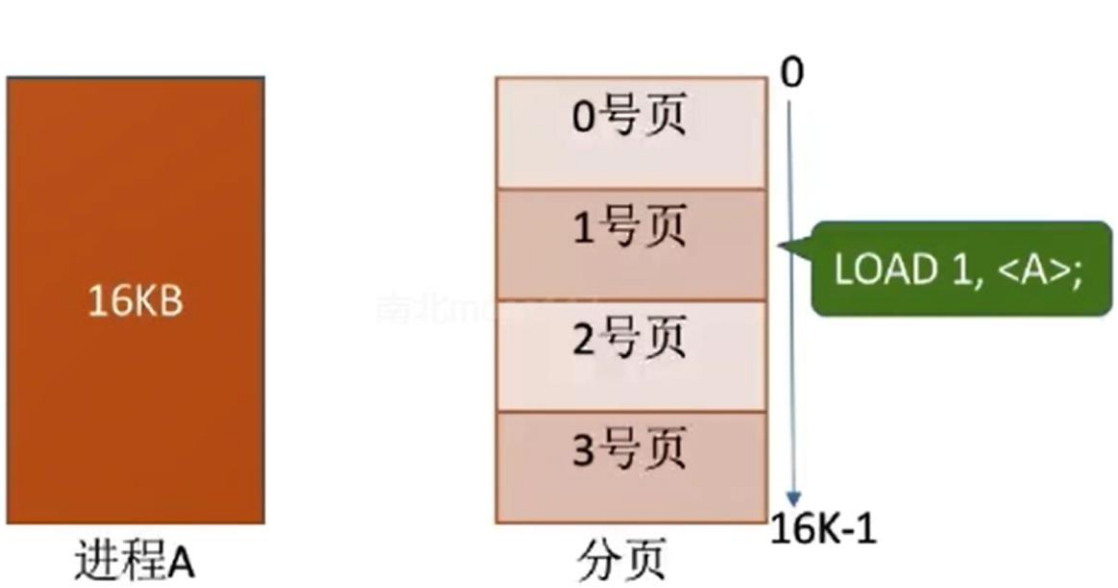
分段、分页管理的对比

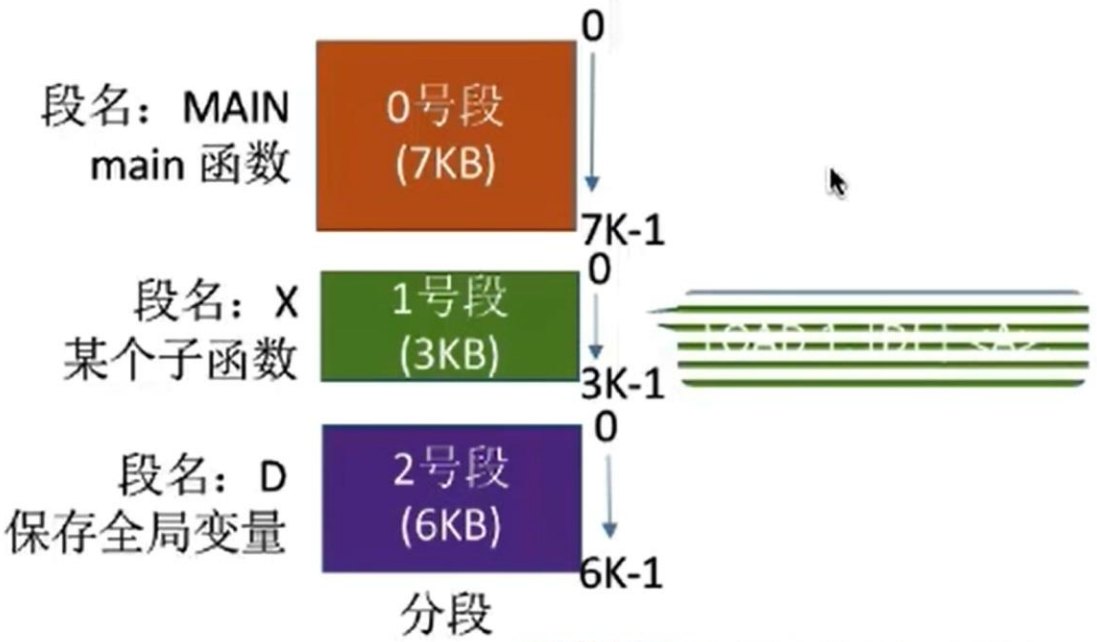
页是信息的物理单位。分页的主要目的是为了实现离散分配，提高内存利用率。分页仅仅是系统管理上的需要，完全是系统行为，对用户是不可见的。

段是信息的逻辑单位。分段的主要目的是更好地满足用户需求。一个段通常包含着一组属于一个逻辑模块的信息。分段对用户是可见的，用户编程时需要显式地给出段名。

页的大小固定且由系统决定。段的长度却不固定，决定于用户编写的程序。

分页的用户进程地址空间是一维的，程序员只需给出一个记忆符即可表示一个地址。
分段的用户进程地址空间是二维的，程序员在标识一个地址时，既要给出段名，也要给出段内地址。

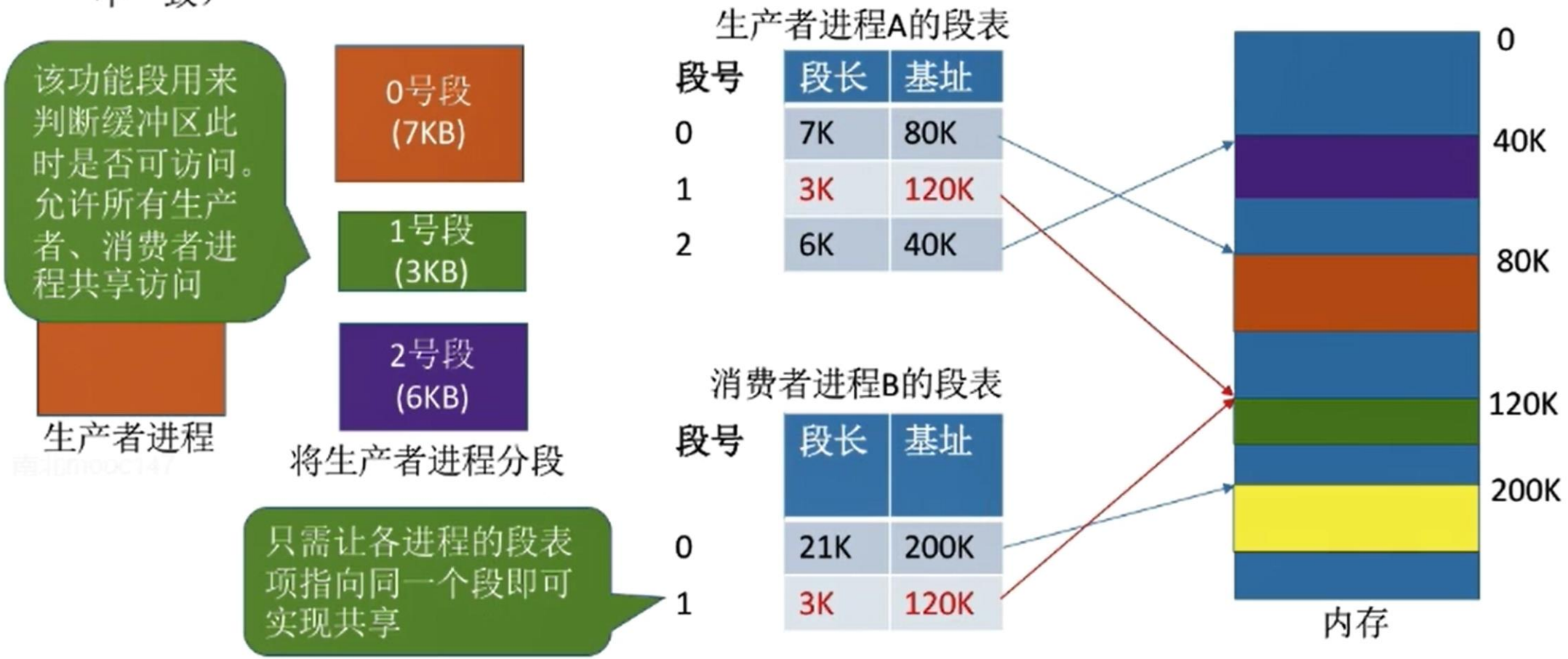




分段、分页管理的对比

分段比分页更容易实现信息的共享和保护。

不能被修改的代码称为纯代码或可重入代码（不属于临界资源），这样的代码是可以共享的。可修改的代码是不能共享的（比如，有一个代码段中有很多变量，各进程并发地同时访问可能造成数据不一致）



分段、分页管理的对比

分段比分页更容易实现信息的共享和保护。

将生产者进程分段

将生产者进程分页

如果让消费者进程的某个页表项指向这个页面，显然不合理，因为这个页面中的橙色部分是不允许共享的，只有绿色部分可以

页面不是按逻辑模块划分的。这就很难实现共享。

生产者进程A的段表

段号	段长	基址	是否允许其他进程访问
0	7K	80K	不允许
1	3K	120K	允许
2	6K	40K	不允许

分段、分页管理的对比

页是信息的物理单位。分页的主要目的是为了实现离散分配，提高内存利用率。分页仅仅是系统管理上的需要，完全是系统行为，对用户是不可见的。

段是信息的逻辑单位。分页的主要目的是更好地满足用户需求。一个段通常包含着一组属于一个逻辑模块的信息。分段对用户是可见的，用户编程时需要显式地给出段名。

页的大小固定且由系统决定。段的长度却不固定，决定于用户编写的程序。

分页的用户进程地址空间是一维的，程序员只需给出一个记忆符即可表示一个地址。

分段的用户进程地址空间是二维的，程序员在标识一个地址时，既要给出段名，也要给出段内地址。

分段比分页更容易实现信息的共享和保护。不能被修改的代码称为纯代码或可重入代码（不属于临界资源），这样的代码是可以共享的。可修改的代码是不能共享的

访问一个逻辑地址需要几次访存？

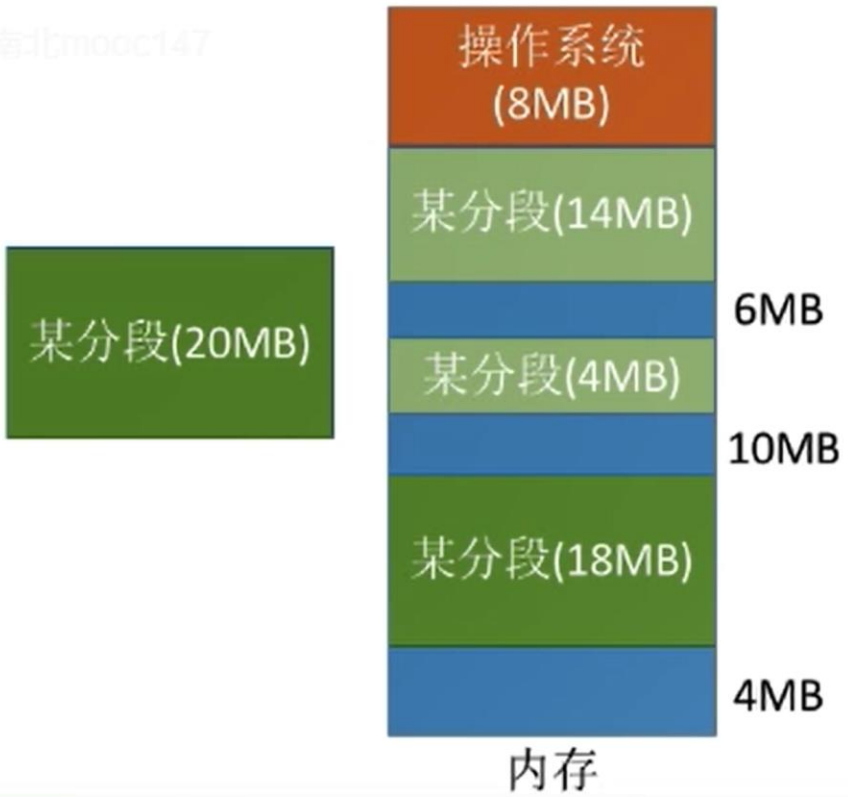
分页（单级页表）：第一次访存——查内存中的页表，第二次访存——访问目标内存单元。总共两次访存

分段：第一次访存——查内存中的段表，第二次访存——访问目标内存单元。总共两次访存

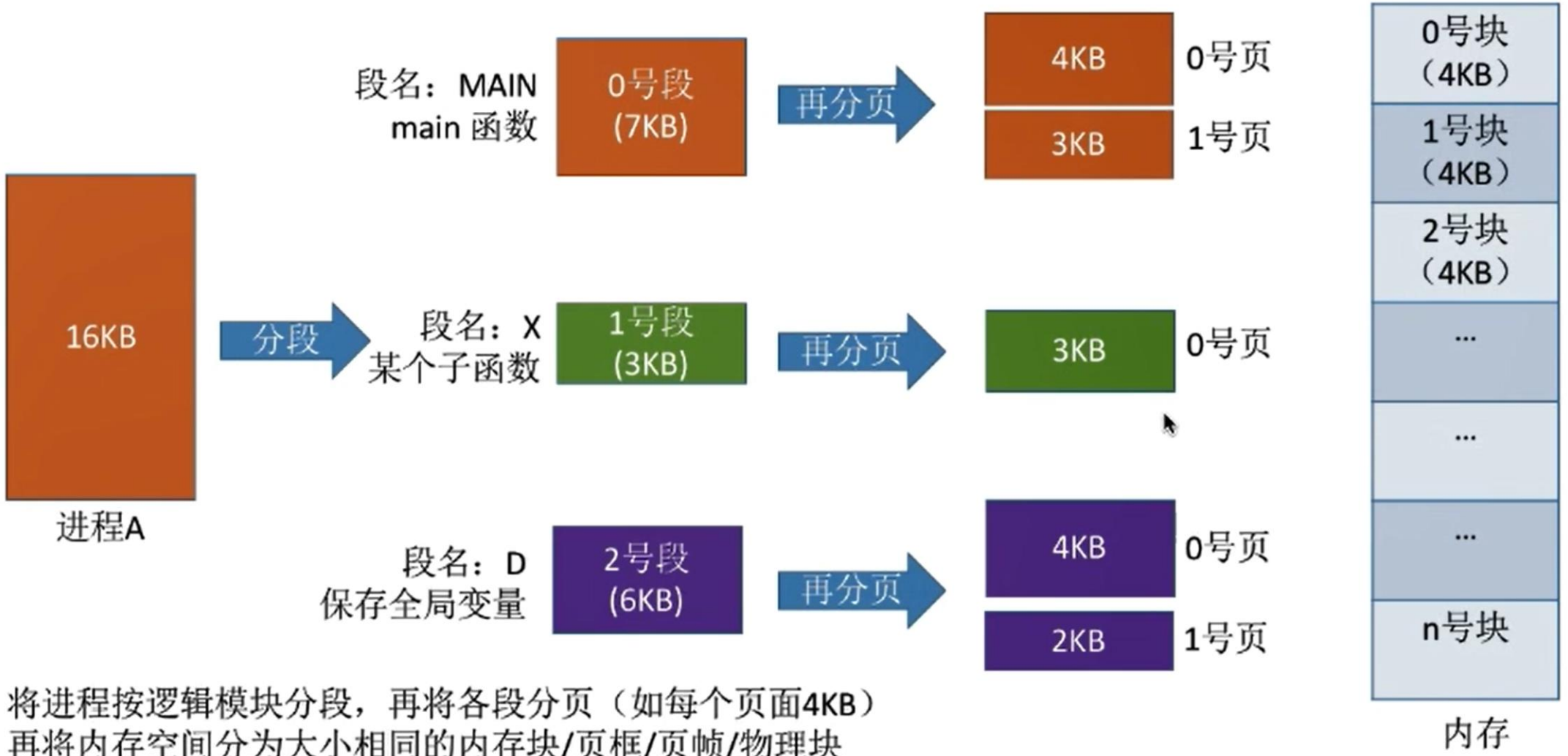
与分页系统类似，分段系统中也可以引入*快表*机构，将近期访问过的段表项放到快表中，这样可以少一次访问，加快地址变换速度。

分页、分段的优缺点分析

	优点	缺点
分页管理	内存空间利用率高，不会产生外部碎片，只会有少量的页内碎片	不方便按照逻辑模块实现信息的共享和保护
分段管理	很方便按照逻辑模块实现信息的共享和保护	如果段长过大，为其分配很大的连续空间会很不方便。另外，段式管理会产生外部碎片



分段+分页=段页式管理



将进程按逻辑模块分段，再将各段分页（如每个页面4KB）
再将内存空间分为大小相同的内存块/页框/页帧/物理块
进程前将各页面分别装入各内存块中

段页式管理的逻辑地址结构

分段系统的逻辑地址结构由段号和段内地址（段内偏移量）组成。如：

31 …… 16	15 …… 0
段号	段内地址

段页式系统的逻辑地址结构由段号、页号、页内地址（页内偏移量）组成。如：

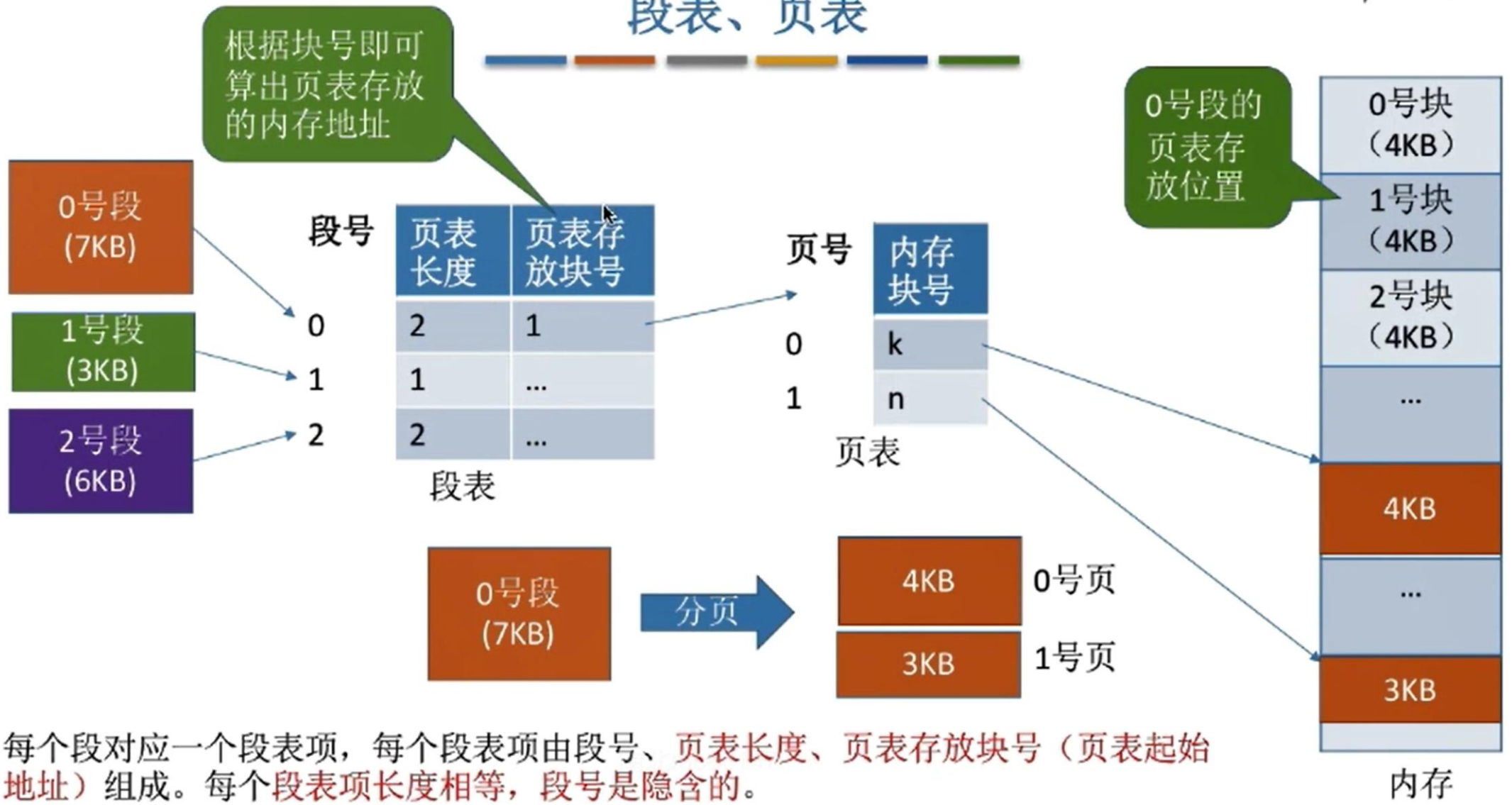
31 …… 16	15 …… 12	11 …… 0
段号	页号	页内偏移量

段号的位数决定了每个进程最多可以分几个段
页号位数决定了每个段最大有多少页
页内偏移量决定了页面大小、内存块大小是多少

在上述例子中，若系统是按字节寻址的，则
段号占16位，因此在该系统中，每个进程最多有 $2^{16} = 64K$ 个段
页号占4位，因此每个段最多有 $2^4 = 16$ 页
页内偏移量占12位，因此每个页面\每个内存块大小为 $2^{12} = 4096 = 4KB$

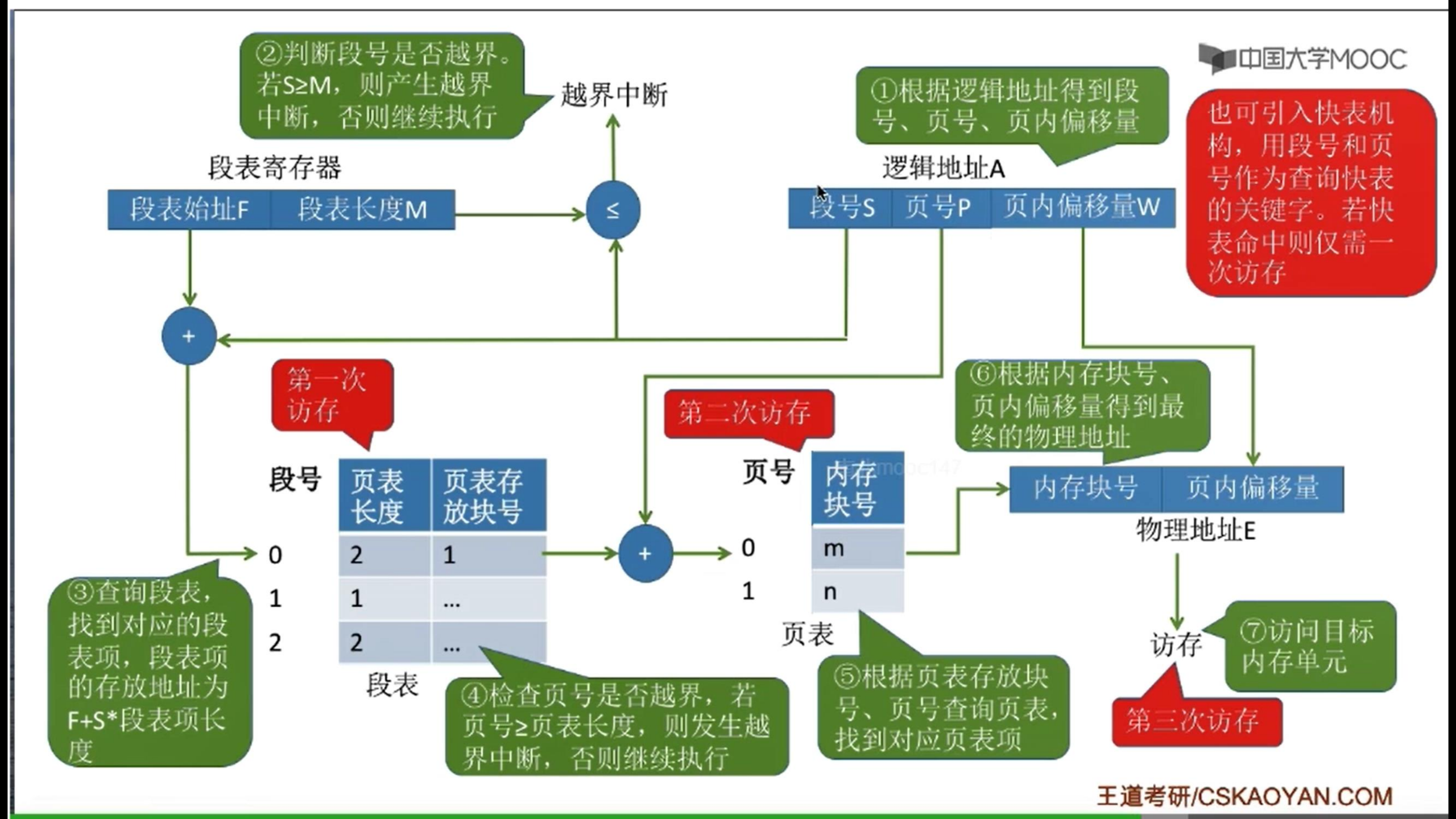
“分段”对用户是可见的，程序员编程时需要显式地给出段号、段内地址。而将各段“分页”对用户是不可见的。系统会根据段内地址自动划分页号和页内偏移量。
因此段页式管理的地址结构是二维的。

段表、页表



每个段对应一个段表项，每个段表项由段号、页表长度、页表存放块号（页表起始地址）组成。每个段表项长度相等，段号是隐含的。

每个页面对应一个页表项，每个页表项由页号、页面存放的内存块号组成。每个页表项长度相等，页号是隐含的。


②判断段号是否越界。若S≥M，则产生越界中断，否则继续执行
越界中断
①根据逻辑地址得到段号、页号、页内偏移量
也可引入快表机构，用段号和页号作为查询快表的关键字。若快表命中则仅需一次访存
段表寄存器
段表始址F
段表长度M
≤
逻辑地址A
段号S
页号P
页内偏移量W
+
第一次访存
第二次访存
⑥根据内存块号、页内偏移量得到最终的物理地址
段号
页表长度
页表存放块号
0
1
2
2
1
1
...
2
...
段表
+
页号
内存块号
0
1
m
n
页表
内存块号
页内偏移量
物理地址E
访存
⑦访问目标内存单元
第三次访存
③查询段表，找到对应的段表项，段表项的存放地址为F+S*段表项长度
④检查页号是否越界，若页号≥页表长度，则发生越界中断，否则继续执行
⑤根据页表存放块号、页号查询页表，找到对应页表项
王道考研/CSKAOYAN.COM

知识回顾与重要考点

段页式管理

- **分段+分页**
  - 将地址空间按照程序自身的逻辑关系划分为若干个段，在将各段分为大小相等的页面
  - 将内存空间分为与页面大小相等的一个个内存块，系统以块为单位为进程分配内存
  - 逻辑地址结构：（段号，页号，页内偏移量）
- **段表、页表**
  - 每个段对应一个段表项。各段表项长度相同，由段号（隐含）、页表长度、页表存放地址 组成
  - 每个页对应一个页表项。各页表项长度相同，由页号（隐含）、页面存放的内存块号 组成
- **地址变换**
  1. 由逻辑地址得到段号、页号、页内偏移量
  2. 段号与段表寄存器中的段长度比较，检查是否越界
  3. 由段表始址、段号找到对应段表项
  4. 根据段表中记录的页表长度，检查页号是否越界
  5. 由段表中的页表地址、页号得到查询页表，找到相应页表项
  6. 由页面存放的内存块号、页内偏移量得到最终的物理地址
  7. 访问目标单元
- **访问一个逻辑地址所需访存次数**
  - 第一次——查段表、第二次——查页表、第三次——访问目标单元
  - 可引入快表机构，以段号和页号为关键字查询快表，即可直接找到最终的目标页面存放位置。引入快表后仅需一次访存