页式、段式、段页式

基本内存管理方案2

一个进程进入内存中若干
不连续的区域

下面我们介绍另外三种基本的内存管理方案。

页式存储管理方案

- 设计思想
- 用户进程地址空间被划分为大小相等的部分，称为页（page）或页面，从0开始编号
- 内存空间按同样大小划分为大小相等的区域，称为页框（page frame），从0开始编号；也称为物理页面，页帧，内存块
- 内存分配（规则）

以页为单位进行分配，并按进程需要的页数来分配；逻辑上相邻的页，物理上不一定相邻

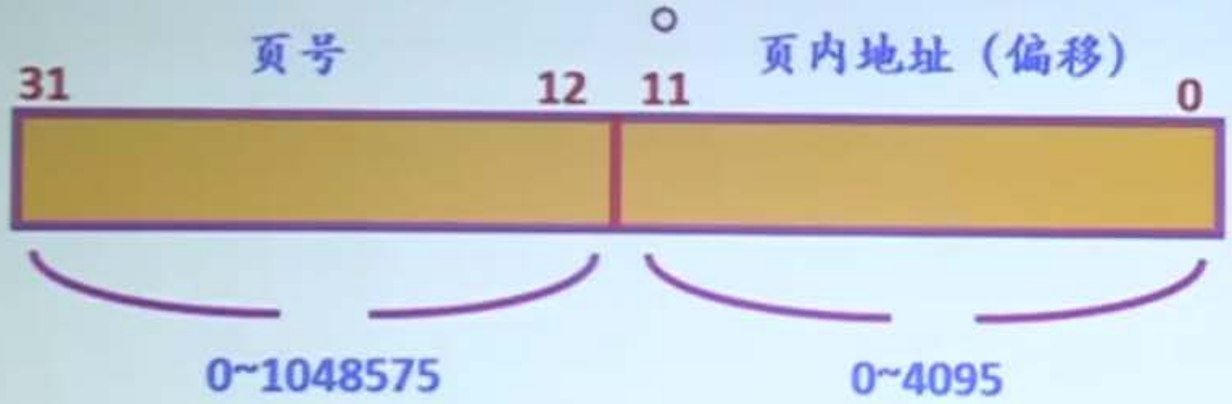
- 典型页面尺寸：4K 或 4M

当然啦，这个尺寸呢是和具体的硬件相关，计算机体系结构相关的。

页式存储管理方案

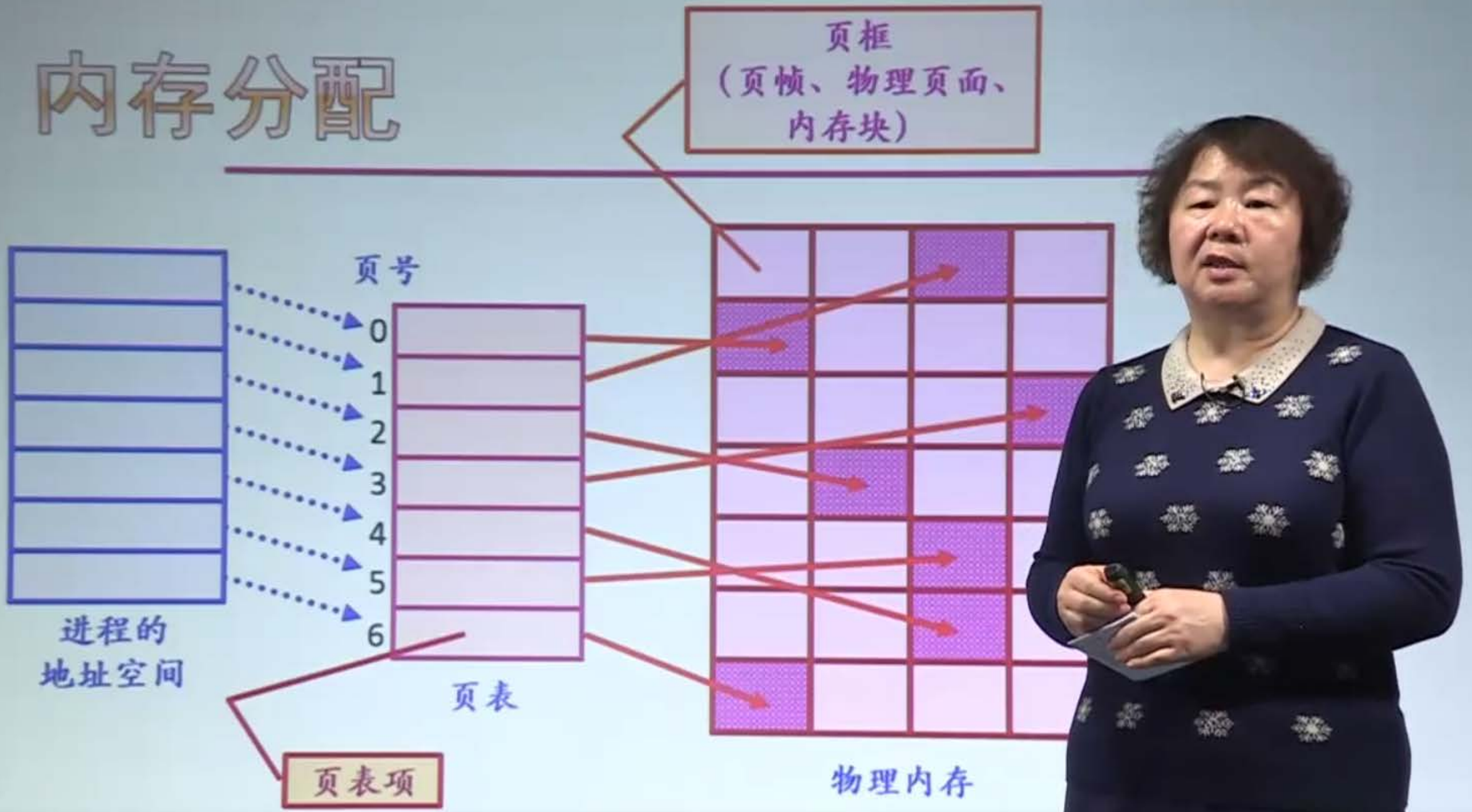
划分是由系统自动完成的，对用户是透明的

逻辑地址



对用户来讲实际上是透明的，所以这是页式存储管理方案的一个特点。

内存分配



相关数据结构及地址转换

- 页表
  - 页表项：记录了逻辑页号与页框号的对应关系
  - 每个进程一个页表，存放在内存
  - 页表起始地址保存在何处？

内碎片

- 空闲内存管理

- 地址转换（硬件支持）

CPU取到逻辑地址，自动划分为页号和页内地址；用页号查页表，得到页框号，再与页内偏移拼接成为物理地址

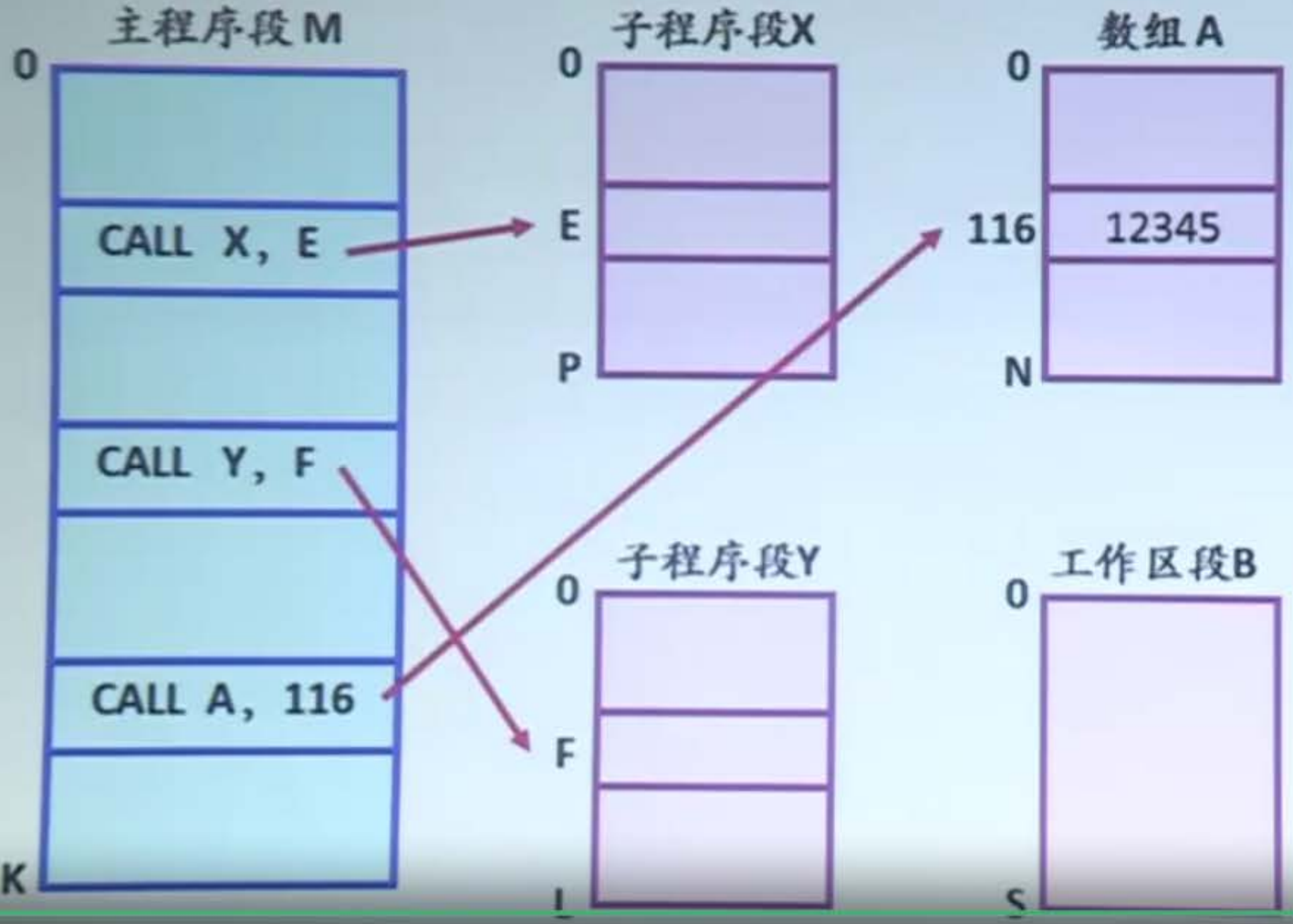
段式存储管理方案

- 设计思想
  - 用户进程地址空间：按程序自身的逻辑关系划分为若干个程序段，每个程序段都有一个段名
  - 内存空间被动态划分为若干长度不相同的区域，称为物理段，每个物理段由起始地址和长度确定
  - 内存分配（规则）：以段为单位进行分配，每段在内存中占据连续空间，但各段之间可以不相

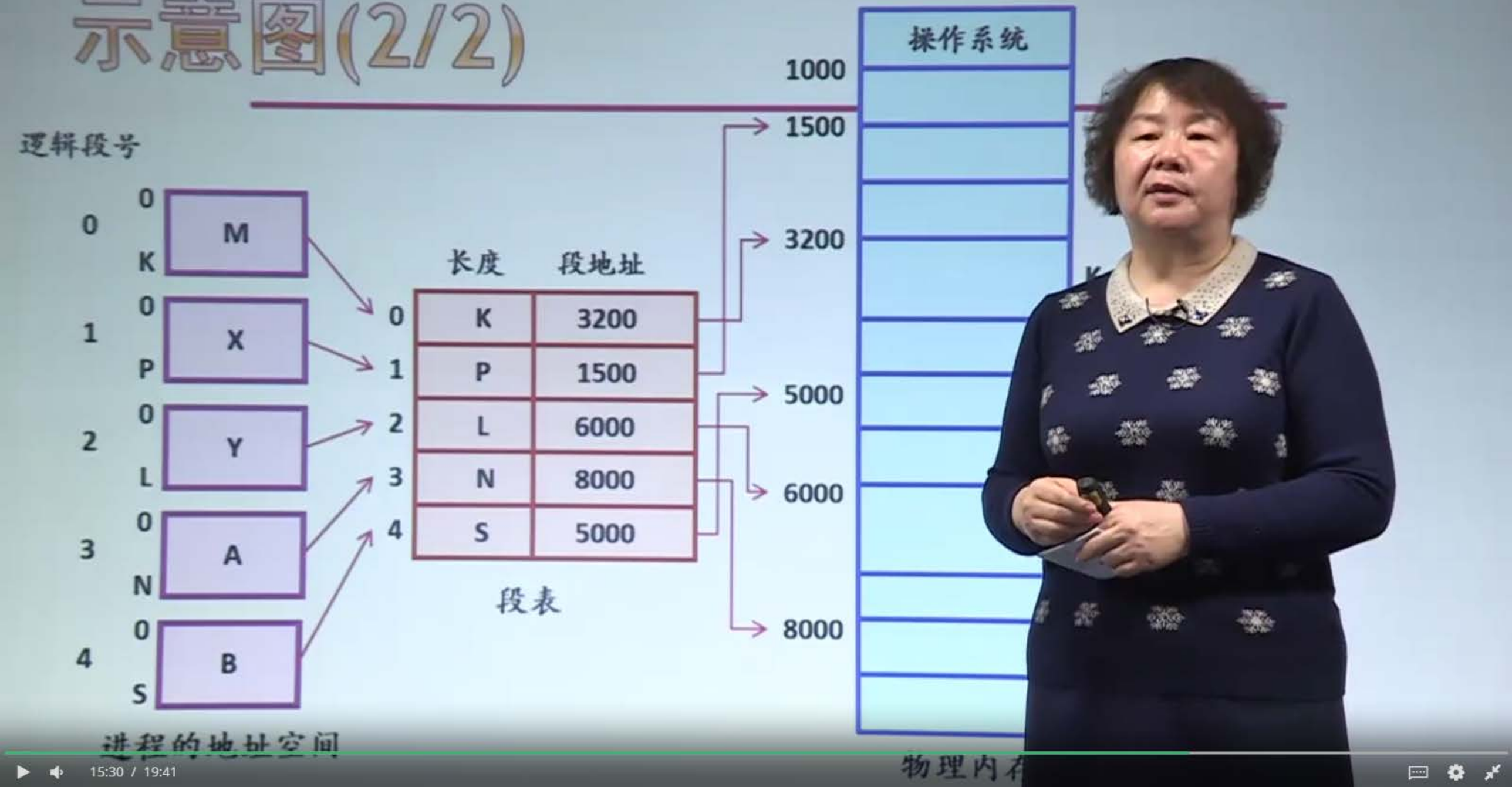
- 逻辑地址

段号	段内地址

示意图(1/2)


主程序段 M
0
CALL X, E
CALL Y, F
CALL A, 116
K
子程序段X
0
E
P
子程序段Y
0
F
L
数组 A
0
116
12345
N
工作区段B
0
S

示意图(2/2)


逻辑段号
0
0
K
M
1
0
P
X
2
0
L
Y
3
0
N
A
4
0
S
B
长度
段地址
0
K
3200
1
P
1500
2
L
6000
3
N
8000
4
S
5000
段表
进程的地址空间
1000
1500
3200
5000
6000
8000
操作系统
物理内存

相关数据结构及地址转换

- 段表
  - 每项记录了段号、段首地址和段长度之间的关系
  - 每个进程一个段表，存放在内存
  - 段表起始地址保存在何处？

- 物理内存管理

- 地址转换（硬件）

CPU取到逻辑地址，用段号查段表，得到该段在内存的起始地址，与段内偏移地址计算出物理地址

段页式存储管理方案

- 产生背景

  综合页式、段式方案的优点，克服二者的缺点

- 设计思想

  用户进程划分：

  先按段划分，每一段再按页面划分

  逻辑地址：

段号	段内地址	
	页号	页内地址

  内存划分：同页式存储管理方案

  内存分配：以页为单位进行分配

段页式存储管理

- 数据结构及有关操作
  - 段表：记录了每一段的页表始址和页表长度
  - 页表：记录了逻辑页号与页框号的对应关系
    每一段有一张页表，一个进程有多个页表

  - 空闲区管理：同页式管理
  - 内存分配、回收：同页式管理

- 地址转换

段号	段内地址	
	页号	页内地址

基本内存管理方案小结

单一连续区	每次只运行一个用户程序，用户程序独占内存，它总是被加载到同一个内存地址上
固定分区	把可分配的内存空间分割成若干个连续区域，每一区域称为分区。每个分区的大小可以相同也可以不同，分区大小固定不变，每个分区装一个且只能装一个进程
可变分区	根据进程的需求，把可分配的内存空间分割出一个分区，分配给该进程
页式	把用户程序地址空间划分成大小相等的部分，称为页。内存空间按页的大小划分为大小相等的区域，称为内存块（物理页面，页框，页帧）。以页为单位进行分配，逻上相邻的页，物理上不一定相邻
段式	用户程序地址空间按进程自身的逻辑关系划分为若干段，内存空间被动态的划分为若干个长度不相同的区域（可变分区）。以段为单位分配内存，每一段在内存中占据连续空间，各段之间可以不连续存放
段页式	用户程序地址空间：段式；内存空间：页式；分配单位：页

0:00 下面我们介绍另外三种基本的内存管理方案。 页式，段式和段页式。 这三种方案和前面介绍的三种它区别是在哪里呢？主要是 一个进程它不是整个进入内存的 一片连续的区域，而是进入内存 的若干个区域，而这些区域呢又不连续， 这就是这三种存储管理方案的特点。 我们首先介绍一下页式存储管理方案，这是一个非常重要的 因为现代大多数的计算机系统都是支持这种存储管理方案的， 它的设计思想是这样的，用户进程 他的空间地址空间被划分为 大小相等的部分，那么这个每个部分呢，我们称之为页或者叫页面。 那么一个进程地址空间会划分成多个 页面，那么我们从0开始编号，0页，1页，2页 同样道理，内存地址空间也被按同样的大小划分成大小相等的区域， 那么我们又给它取了一个新的术语，叫做页框或者叫 页帧，物理页面，内存块儿。 那注意就是说如果我们说页框，其实就指的是物理内存的一个页面， 它也是同样从零开始编号儿，好了，这是 用户地址空间，这是内存，这两个空间 那么内存的分配规则是什么样呢？策略是什么样呢？ 怎么样把用户地址空间内容把它送到内存呢？ 我们是按这样一个策略完成的，首先 以页为单位进行分配，每次分配一页。 那么进程需要多少页，那么我们就分给它多少页？但是逻辑上相邻的页物理上不一定相邻，也就是说 我们假设啊一个进程有5页，那么编号是0到4， 那么逻辑上呢这0页1页2页就是相邻的，但是在物理上，在内存里头 那么0页和1页，进程的0页和1页 在物理内存里头可能不是放在连续的两页。 这就是逻辑上相邻物理上不相邻的含义。 那么通常情况下，页面的大小，典型的页面尺寸啊 是2的整数次幂，我们比较常见的尺寸呢是4k， 2的12次方或者是4兆，这个常见的尺寸。 当然啦，这个尺寸呢是和具体的硬件相关，计算机体系结构相关的。 那我们来再看一下 逻辑地址，我们前面讲的要进行地址转换， 那么逻辑地址在页式存储管理方案中是什么样呢？ 逻辑地址实际上由两部分组成，一部分呢叫页号， 一部分呢就是页内地址， 比如说如果我们有一个三十二位的计算机， 那么它的逻辑地址呢三十二位，那我们来看一下， 这三十二位我们假设刚才按照 常规的尺寸页面大小是2的十二次方，4k，因此呢 我们留出来12位，0到11，把它表示页内地址。 而这个也称为叫偏移，页内偏移，也就是从0到4095是页内偏移。 然后高的20位我们就把它 划分为叫页号，啊，叫页号，也就是说2的32次方这么一个地址 那么2的12次方，这里头有12位的页内地址偏移， 剩下20位就是页号，从0一直到十万多，十万多。 而这种划分由于我们的页面大小采用的2的整数次幂，所以这种划分 是系统自动完成的硬件自动完成的。 对用户来讲实际上是透明的，所以这是页式存储管理方案的一个特点。 好，

那么我们要把 用户地址空间的内容把它 加载到内存里头去，我们来看一下内存分配的这个过程。 那么这是一片物理内存，我们分成了很多的 页，页框，这里头就是页框，一个一个页框， 那么这是进程的地址空间， 那么我们来看这个进程 1234 有那么7页，啊，7页。 那我们要把这7页 装载到物理内存，那由于页式存储管理方案是说 逻辑上相邻的页物理上不一定相邻，所以我们简单看一下 我们把第0页装载到这个地方 第1页加载到这个地方， 以此类推，大家可以看一下，一个进程的 这个7个页面在内存中分别放在不同的页框当中， 那这里头假设我们就比较这个 画的就是东一块西一块了，就是让大家看看这个示意，这是不同的 逻辑上相邻的在物理上不相邻这样一个效果。 那这里就有一个问题了，逻辑上相邻，那么 程序在执行的时候呢，我们是按照一个顺序慢慢执行的。 不考虑跳转的情况的话，执行完第一条指令再执行第二条指令， 如果执行完上一页的最后一条指令，要执行下一页的第一条指令 那我怎么样能找到下一页的页框呢，也就是说 这种转换，这种映射关系我怎么能够， 得到？系统怎么能够知道 好，那我们来看看怎么知道呢，我们来看一下 在这个对应关系中一定有一个成分存在 它会把逻辑上的某一页和 物理上的某一页框的对应关系记录下来， 那么这个呢就是我们 非常重要的一个数据结构就是页表， 也就是通过这个页表记录了 这样一个映射关系，也就是说逻辑上的 第0页在我的页表当中呢 第1行，那么这个页表当中呢它就会记录了 这个逻辑上或者说用户空间的第0页， 是放在了物理内存的某个页框，这个页框号是记录在这张表里头的。 那么这里头每一行在页表里头的每一行，我们叫做页表项，叫页表项。 这也是非常重要的一个概念，这就是我们刚才所讲的一个问号。 就是要把这个对应关系 记录下来，这就是数据结构，下面我们介绍页式存储管理方案中相关的数据结构 以及地址转换过程，最重要的一个数据结构就是页表， 页表呢由若干页表项组成， 页表项记录了逻辑页号 到页框号的一个映射关系，也就是通过页表项我们根据逻辑页号就能找到它对应的页框号。 我们可以做地址转换，那么每个进程呢都有一张页表， 那么这个页表呢一般情况下是放在内存 页表放在内存，那么这个页表的起始地址放在哪里呢？ 保存在什么地方呢？根据我们以前的知识，一个进程上CPU 那么计算机系统的若干寄存器都由这个进程来使用， 那么换句话说一个进程上CPU之后 这个进程的页表的起始地址要推送到某一个寄存器里头， 那么由系统来完成相应的工作。 当这个进程不上CPU了，或者说临时处于其他的状 态，就绪呀，等待啊等等，

那么这个进程的页表的起始地址又保存在什么地方呢？ 大家可以想一下我们之前呢我们有很多的介绍的数据结构， 是哪一个数据结构里头要保存页表的起始地址， 后面呢我们来看一下 物理内存，在这种页式存储管理方案中的物理内存的 管理呢其实呢我们可以用 bitmap 位图啊这个数据结构就可以管理物理内存了。 那么地址转换过程是什么样呢？ 首先我们来看一下应该强调的是地址转换过程 通常为了加快这一过程，我们是需要由硬件来支持的， 那么一个逻辑地址我们知道把它分成了两部分。 一部分是页号，一部分是页内地址 那么CPU 渠道的指令啊，或者数据的地址啊，这个逻辑地址呢，是 这两部分组成，而这两部分呢，是系统自动划分的 然后硬件就会把 页号拿来去查页表 通过查页表，得到了这个页号所对应的页框号 然后再将这个页框号，和页内 偏移，或者页内地址拼接成一个物理地址 然后CPU用这个物理地址到内存去取指令，或者是取数据 这就是一个地址转换的一个过程，所以 这个划分啊，对一个逻辑地址的这个划分，实际上是自动完成的 好，那么在页式存储管理方案当中呢 我们可能遇到了一个碎片问题，但这个碎片呢，我们称之为内碎片 举一个例子，假设一个进程需要五页 然后加一条指令，也就是说它需要五页，再加一条指令 可是，尽管只有还剩一条指令，我们要给它分 六页，所以这个进程呢，实际上呢，你要给它分成六页 那么最后一条指令，假设啊是 在这个第六页上，只占了很小很小的 这个空间，整个大部分的页面都是空的 很浪费，但是呢，由于我们的这种存储管理方案的这个模型是这个样子的，所以呢 那么这个 也要分给这个进程六页，因此呢 这就是产生了内碎片，内碎片 那么下面我们来简单介绍一下段式存储管理方案 段式存储管理方案的设计思想呢，是这样的，用户 地址空间呢，是按照程序自身的逻辑关系划分为若干个程序片段 每个程序段都有一个段的名字 好，这是由于程序 自身是有逻辑关系的，所以我们就按这个逻辑关系来划分 就不是按照等长来划分，那么不同的逻辑关系，可能占据的空间大小不一样 而内存空间呢，也被动态的划分成 若干长度不等的区域，其实我们就是我们前面所讲的这种 等不等长的这种划分 那么分配的时候呢，实际上是按照你分的这些程序段 为单位来分配内存，也就是以段为单位来分配内存，同样一个进程可能分成 若干段，然后每一段占据一块内存，而段与段之间呢，其实呢，可能在内存中是不连续存放的 这是段式存储管理方案 好，那么强调的是段啊 那么逻辑地址呢，跟页式应该思路是一样的，那么它是由两部分组成的，一部分是段号 一部分是段内地址，但是这里头呢，和页式不

一样的地方是，段号和段内地址 不是自动划分的，没法自动划分，必须显示给出 我们来一个示意啊，假设某一个进程，我们给它分成这么几段 这一段呢叫主程序段，还有一个 X 段， Y 段 还有一个数组 A 它也占一段，然后还有一些工作区占一段 所以假设啊，这个进程分成了五段 那么这里头呢，在主程序里头去会调用其他的段，比如说它要调用 不同的内容啊，假定是这么一个关系 那么如果我们按照刚才啊 页式存储管理方案的设计思路呢，我们实际上也是 要把这些段都分在内存的不同的区域啊，给它分配不同的区域，让这些段进入内存 我们假设啊，M 就表示主程序段，它呢长度是 K，所以它占据了 这一块区域，占据这一块区域，那么 X 段呢，它的 长度是 P 它占据的是这个区域，是这样一个示意，那么如果我们要把 这个对应关系记录下来，我们就需要一个数据结构，就是段表 当然段表的内容啊，每个段表项呢，那就是包括了 每一段的长度，每一段的段的起始点，起始地址 这叫段表，好，那我们来看看这个映射关系，实际上呢 就可以通过这个段表就能够把这个映射关系记录下来 那么如果我要想找某一段，我就可以去查段表，找到它在内存的位置，然后再去做相应的- 其他工作 这就是段式存储管理方案的一个示意，那么相关数据结构呢 和页式也差不多，要有一个段表 段表呢是由若干段表项组成 每一项记录了段号啊、 段首地址啊，段长度之间的一个啊关系 也是每个进程一个段表放在内存，段表的起始地址呢 也跟页表一样的这样的一个存放的位置，大家去想一下 物理内存管理呢，我们可以用不等长的物理内存 管理方法，我们可以用空闲区表来管理这个物理内存 地址转换过程呢，也是由硬件来支持的 CPU 呢，取到了一个逻辑地址 这个逻辑地址呢，实际上是两部分组成，段号、 段内地址 那么硬件呢，去通过段号呢，来查段表 通过这个查段表呢，得到了这一段在内存中的起始地址 然后再去用这个 起始地址和段内地址进行相应的计算 得到某个地址对应的什么呀？物理地址 然后再用这个物理地址到相应的内存呢，去取数据或者取指令 这就是段式存储管理方案的一个地址转换过程 下面我们介绍段页式存储管理方案 段页式存储管理方案呢，主要是综合了 页式存储管理方案和段式存储管理方案的优点 当然了也克服了前两者的缺点，它的总的设计思想是 用户进程的划分呢，首先先划分成 段，也就是按程序自身的逻辑的关系划分成段 然后呢，在每一段当中呢，再按页面划分，按等长划分 因此一个逻辑地址呢，实际上是由三部分组成 第一部分呢是段号，然后呢原来的段内地址呢 实际上呢，又划分成页号和页内地址，所以由三部分组成 内存的

划分呢，是和页式存储管理方案 是一样的，等长划分，那么内存分配呢，也是按 页为单位来进行分配，这是段页式存储管理方案的简单的思想 段页式存储管理的数据结构呢，就比较复杂，那么它有段表 段表实际上是记录了每一段，所对应的那个页表的 起始地址和页表长度，那么页表 记录了逻辑页号到页框号对应关系 那么一个进程分成若干段 那么有一张段表就把这若干段的信息记录下来 然后每一段呢，又对应了一张页表，所以 一个进程呢，是由一张段表和多个页表组成的 那么空闲区的管理呢，跟页式是相同的内存的分配，回收的算法呢，也同页式相对比较简单 那么地址转换过程呢，也是有硬件的支持，所以硬件的 部件呢，比较复杂，它先拿到了一个逻辑地址 然后呢，这个逻辑地址是由两部分组成，段号和段内地址 那么它用段号去查段表，得到了所对应的页表的起始地址和长度 然后再把段内地址自动划分成两部分 页号页内地址，用页号去查对应的页表，得到 页框号，然后再和页内地址拼接成最后的五个地址 所以整个这个过程比较复杂，成本也比较高 那么这是呢 各种基本内存管理方案的一个小结，一共有六种，大家自己回去呢，看一下