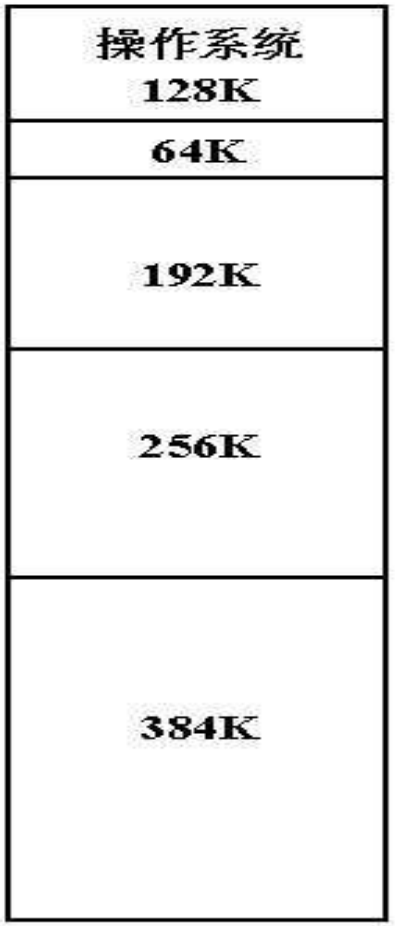
Lecture 15: Virtual Memory II

简单分区（Partitioning）

- 主存分配：
  - 操作系统：固定
  - 用户区：分区
- 简单分区方案：使用长度不等的固定长分区(fixed-size partition)。
- 当一个进程调入主存时，分配给它一个能容纳它的最小的分区。

对于需196K的进程可分配256K的分区。

- 简单分区方式的缺点：
  - 因为是固定长度的分区，故可能会浪费主存空间。多数情况下，进程对分区大小的需求不可能和提供的分区大小一样。



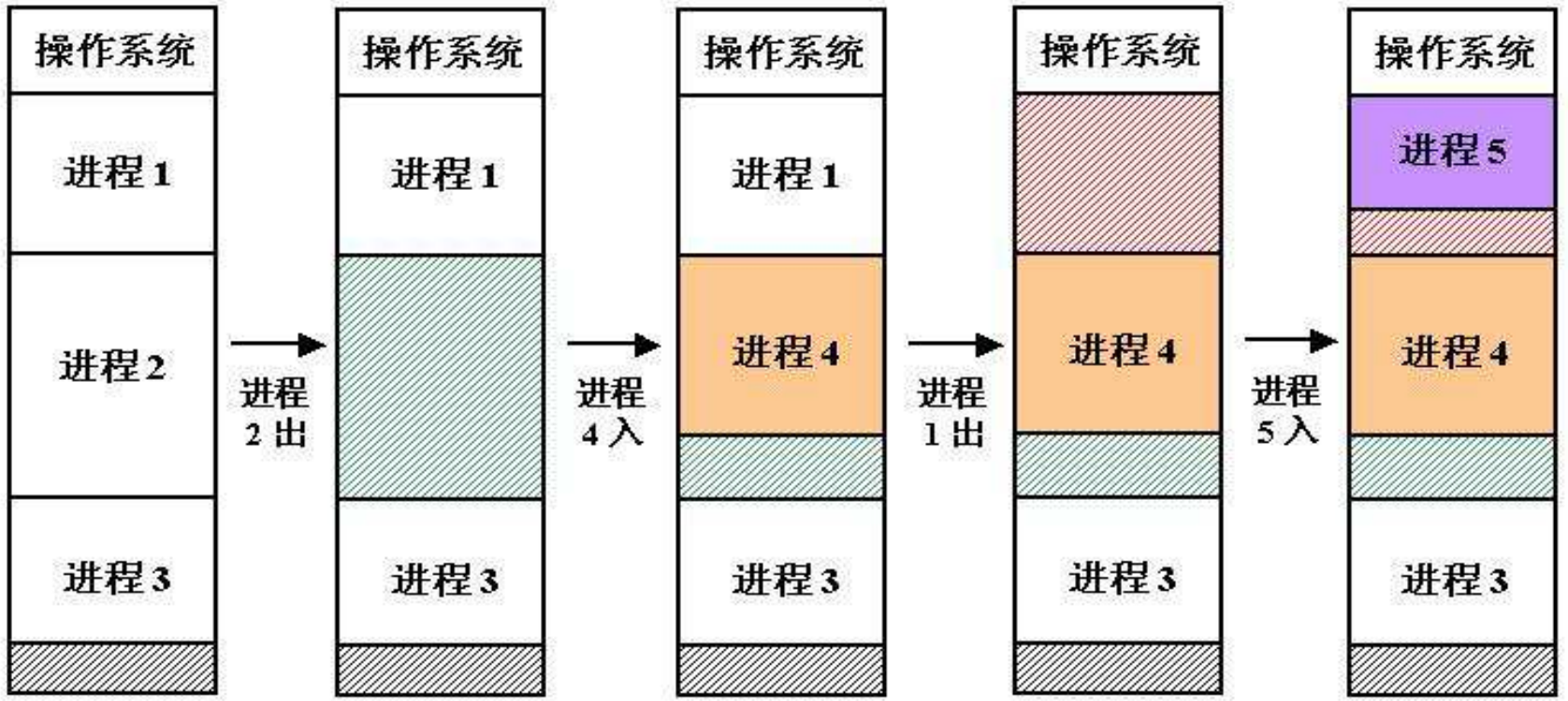
固定分区举例

问题：如何生成物理地址？

可以采用更有效的可变长分区的方式！

可变长分区（variable-length Partitioning）

- 分配的分区大小与进程所需大小一样。
- 特点：开始较好，但到最后在存储器中会有许多小空块出现。时间越长，存储器中的碎片就会越来越多，因而存储器的利用率下降。



分区的效果　更有效的方式是分页！

分页（Paging）

- 基本思想：
  - 内存被分成固定长且比较小的存储块（页框、实页、物理页）
  - 每个进程也被划分成固定长的程序块（页、虚页、逻辑页）
  - 程序块可装到存储器中可用的存储块中
  - 无需用连续页框来存放一个进程
  - 操作系统为每个进程生成一个页表
  - 通过页表(page table)实现逻辑地址向物理地址转换（Address Mapping ）
- 逻辑地址（Logical Address）：
  - 程序中的指令所用的地址，也称为虚拟地址
- 物理地址（physical或Memory Address）：
  - 存放指令或数据的实际内存地址，也称为实地址、主存地址

BACK

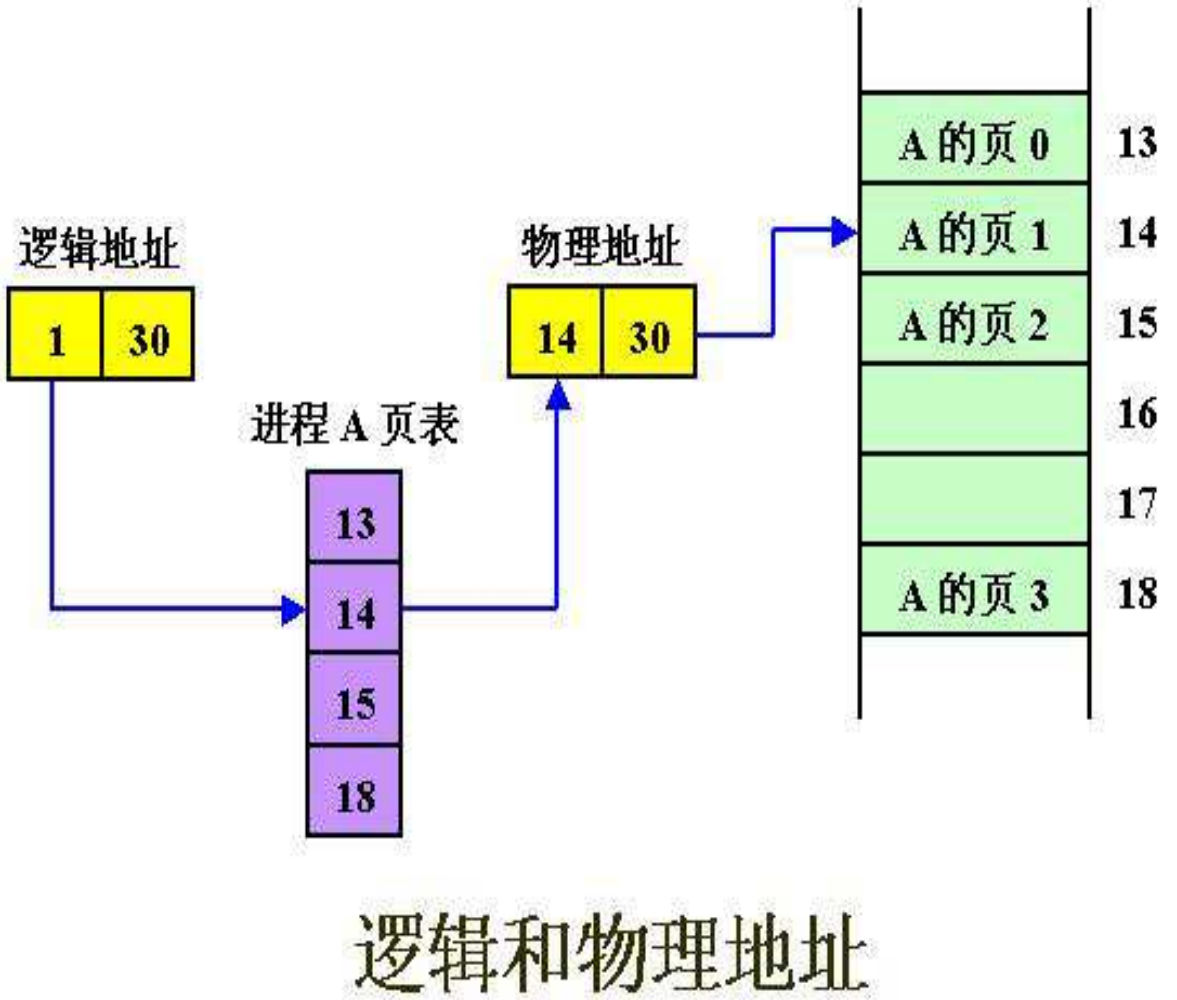
分页（Paging）

问题：是否需要将一个进程的全部都装入内存？

根据程序访问局部性可知：可把当前活跃的页面调入主存，其余留在磁盘上！

采用“按需调页 Demand Paging”方式分配主存！

浪费的空间最多是最后一页的部分！



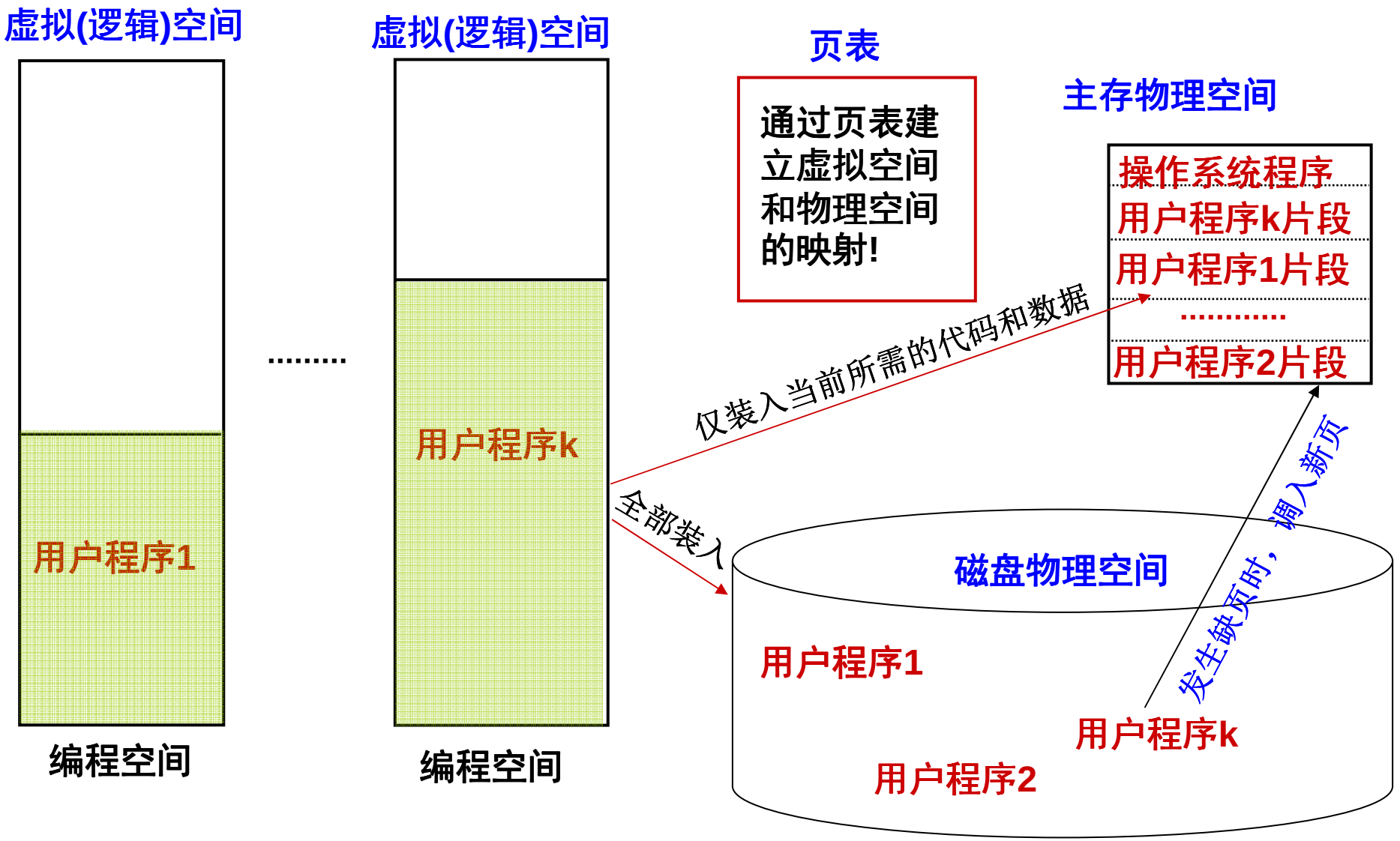
逻辑和物理地址

BACK

虚拟存储系统的基本概念

- 虚拟存储技术的引入用来解决一对矛盾
  - 一方面，由于技术和成本等原因，主存容量受到限制
  - 另一方面，系统程序和应用程序要求主存容量越来越大
- 虚拟存储技术的实质
  - 程序员在比实际主存空间大得多的逻辑地址空间中编写程序
  - 程序执行时，把当前需要的程序段和相应的数据块调入主存，其他暂不用的部分存放在磁盘上
  - 指令执行时，通过硬件将逻辑地址（也称虚拟地址或虚地址）转化为物理地址（也称主存地址或实地址）
  - 在发生程序或数据访问失效时，由操作系统进行主存和磁盘之间的信息交换
- 虚拟存储器机制由硬件与操作系统共同协作实现，涉及到操作系统中的许多概念，如进程、进程的上下文切换、存储器分配、虚拟地址空间、缺页处理等。

虚拟存储技术的实质



BACK

虚拟地址空间

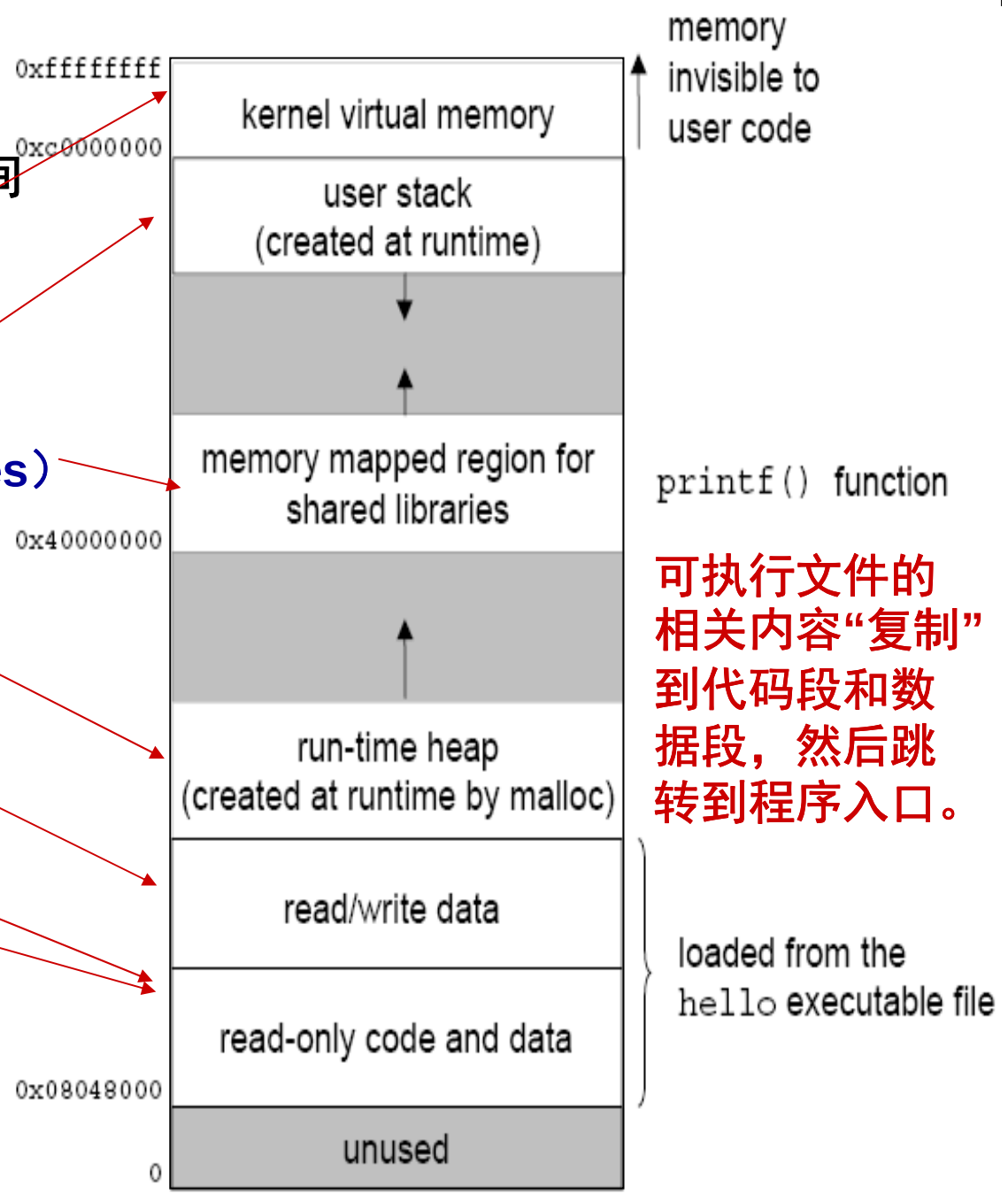
- Linux操作系统的虚拟地址空间

  （其他Unix系统的设计类此）

  - 内核（Kernel）
  - 用户栈（User Stack）
  - 共享库（Shared Libraries）
  - 堆（heap）
  - 可读写数据（Read/Write Data）
  - 只读数据（Read-only Data）
  - 代码（Code）

问题：加载时是否真正从磁盘调入信息到主存？

实际上不会从磁盘调入，只是将代码和数据与虚拟空间建立对应关系，称为“映射”。



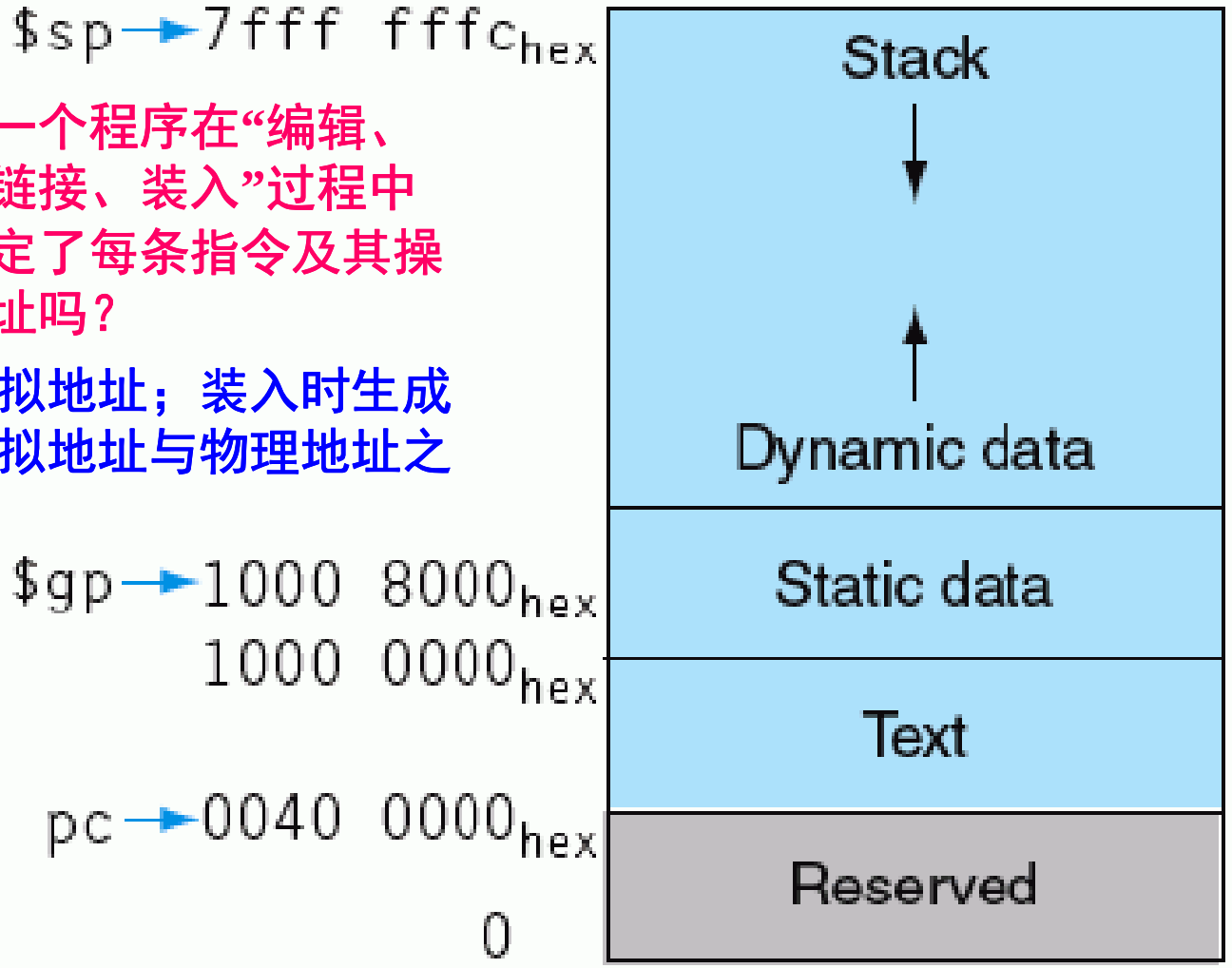
可执行文件的相关内容“复制”到代码段和数据段，然后跳转到程序入口。

MIPS程序和数据的存储器分配

问题：你知道一个程序在“编辑、编译、汇编、链接、装入”过程中的哪个环节确定了每条指令及其操作数的虚拟地址吗？

链接时确定虚拟地址；装入时生成页表以建立虚拟地址与物理地址之间的映射！

请参考《深入理解计算机系统》和有关Linux内核分析方面的资料。



每个用户程序都有相同的虚拟地址空间地址空间！

这就是每个进程的虚拟（逻辑）地址空间！

虚拟存储器管理

- 实现虚拟存储器管理，需考虑：
  - 块大小（在虚拟存储器中“块”被称为“页 / Page”）应多大?
  - 主存与辅存的空间如何分区管理?
  - 程序块 / 存储块之间如何映像?
  - 逻辑地址和物理地址如何转换，转换速度如何提高?
  - 主存与辅存之间如何进行替换（与Cache所用策略相似）?
  - 页表如何实现，页表项中要记录哪些信息?
  - 如何加快访问页表的速度?
  - 如果要找的内容不在主存，怎么办?
  - 如何保护进程各自的存储区不被其他进程访问?

- 有三种虚拟存储器实现方式：

  分页式、分段式、段页式

这些问题是由硬件和OS共同协调解决的！

主存--磁盘层次

和CPU（Cache）和主存层次相比：

页大小比Cache中Block大得多！2KB~64KB Why？

采用全相联映射！ Why?

因为缺页的开销比Cache缺失开销（缺失损失）大的多！！缺页时需要访问磁盘（约几百万个时钟周期），而cache缺失时访问主存仅需要几十到几百个时钟周期！因此，页命中率比cache命中率更重要！“大页面”和“全相联”可提高页命中率。

通过软件来处理“缺页”！ Why?

缺页时需要访问磁盘（约几百万个时钟周期），慢！不能用硬件实现。

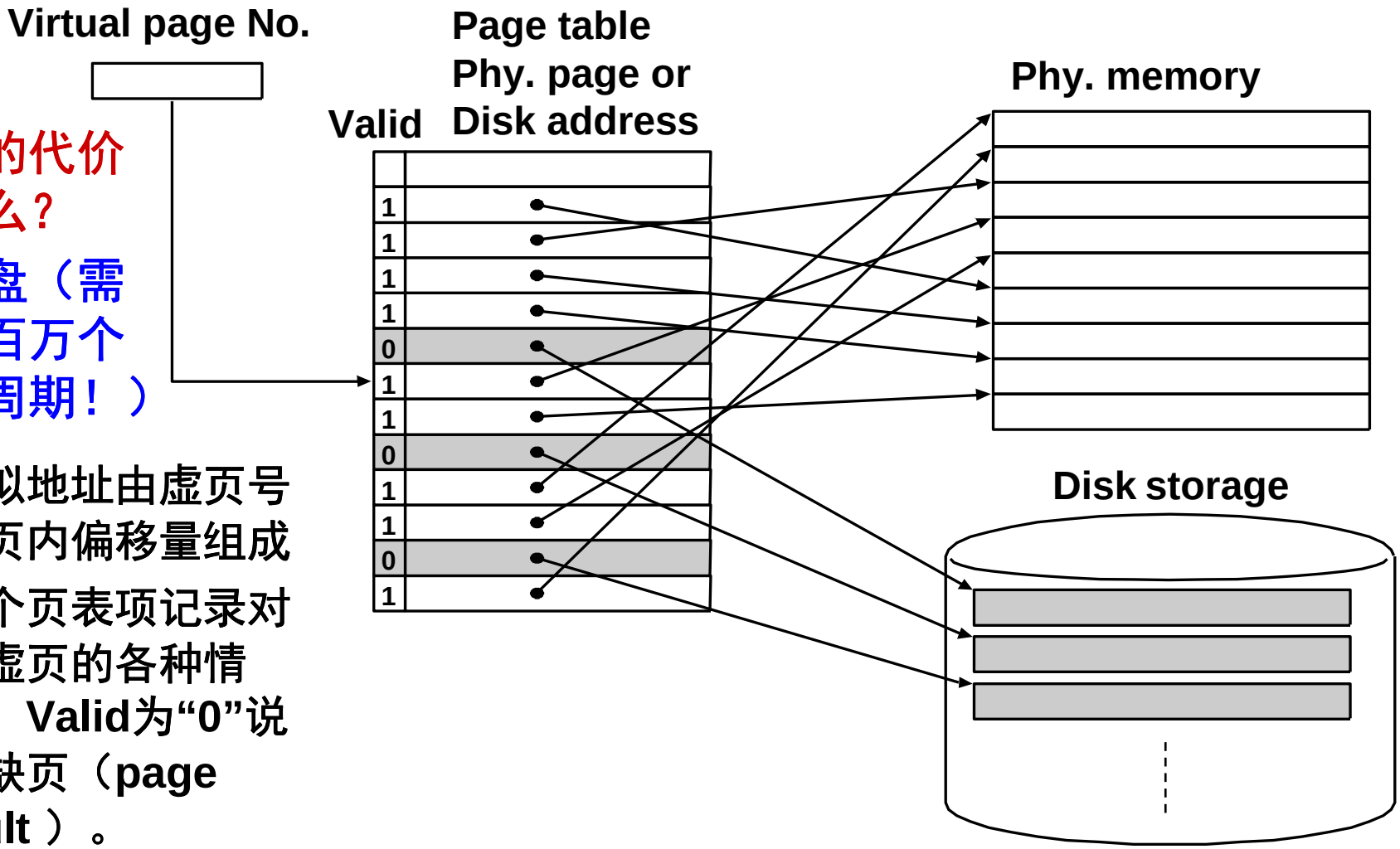
采用Write Back写策略！ Why?

避免频繁的慢速磁盘访问操作。

地址转换用硬件实现！ Why?

加快指令执行。

分页式系统



缺页的代价是什么？

读磁盘（需花几百万个时钟周期！）

- 虚拟地址由虚页号和页内偏移量组成
- 每个页表项记录对应虚页的各种情况，Valid为“0”说明缺页（page fault）。
- 地址转换由CPU中的MMU实现

有些系统采用双表结构：物理地址和磁盘地址分开存放在不同的页表中

页表结构

- 页表首址记录在页表基址寄存器中

用户程序 A 的页表

- 每个进程有一个页表，有装入位、修改（Dirt）位、替换控制位、访问权限位、禁止缓存位、实页号，页表项数由进程大小决定

页表结构

- 必须解决页表占空间过大的问题

（1）页表可能很大，为了让一个进程具有很大的虚拟编程空间，系统必须允许页表的项数足够多

例如：在VAX系统中，每个进程能拥有高达2^{31}＝2G字节的虚拟存储器，按512字节/页进行分页，则每个进程最多可达2^{22}个页表项。显然，这么大的页表全部放在主存中是不适合的

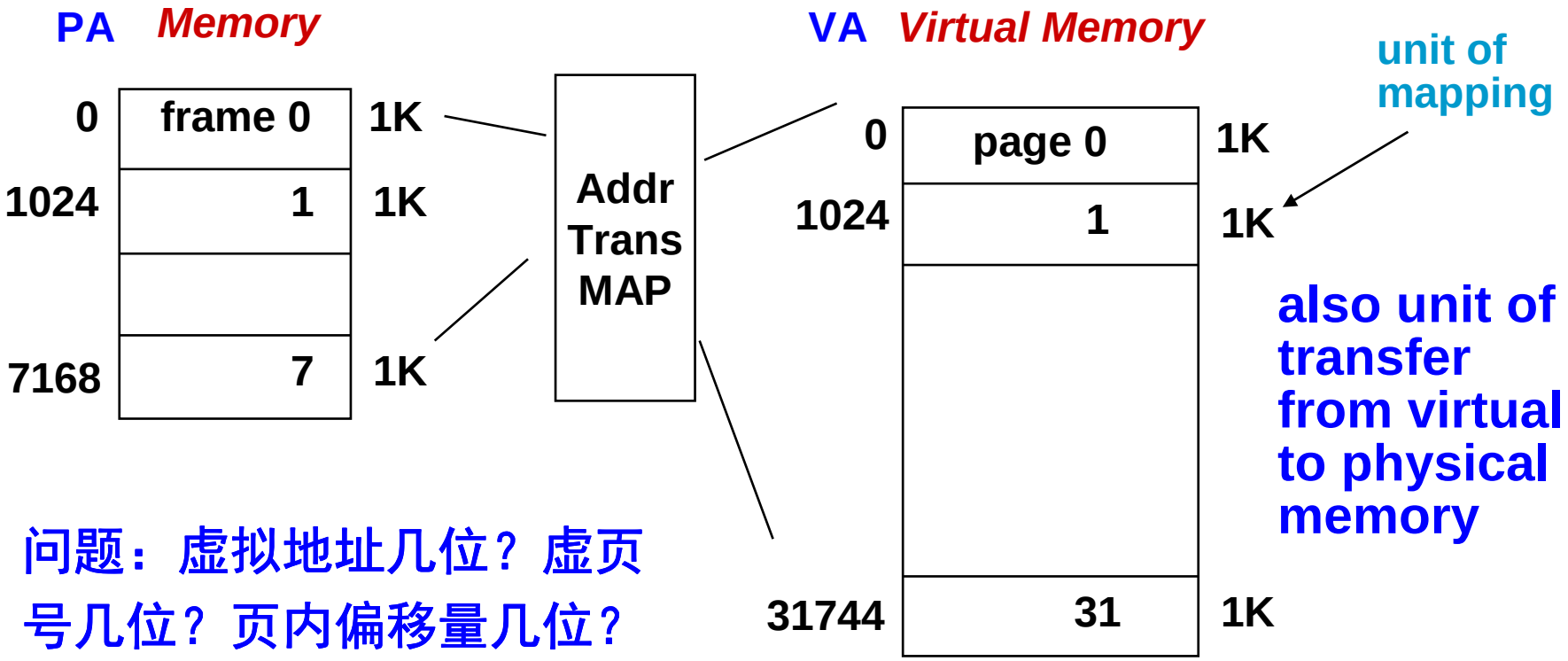
（2）系统中有许多进程，每个进程有一个页表

页表结构

- 解决页表过大的方法
  - 一级页表：动态扩充，限制大小
  - 一级页表：分两个独立的段
  - 二级或多级页表：一级为段、二级为页
  - 将页表分页，当前使用的页的页表项所在页表在内存，其他在外存，页表也要调进调出
  - 反向（倒置）页表

    - 操作系统课程会详细介绍！

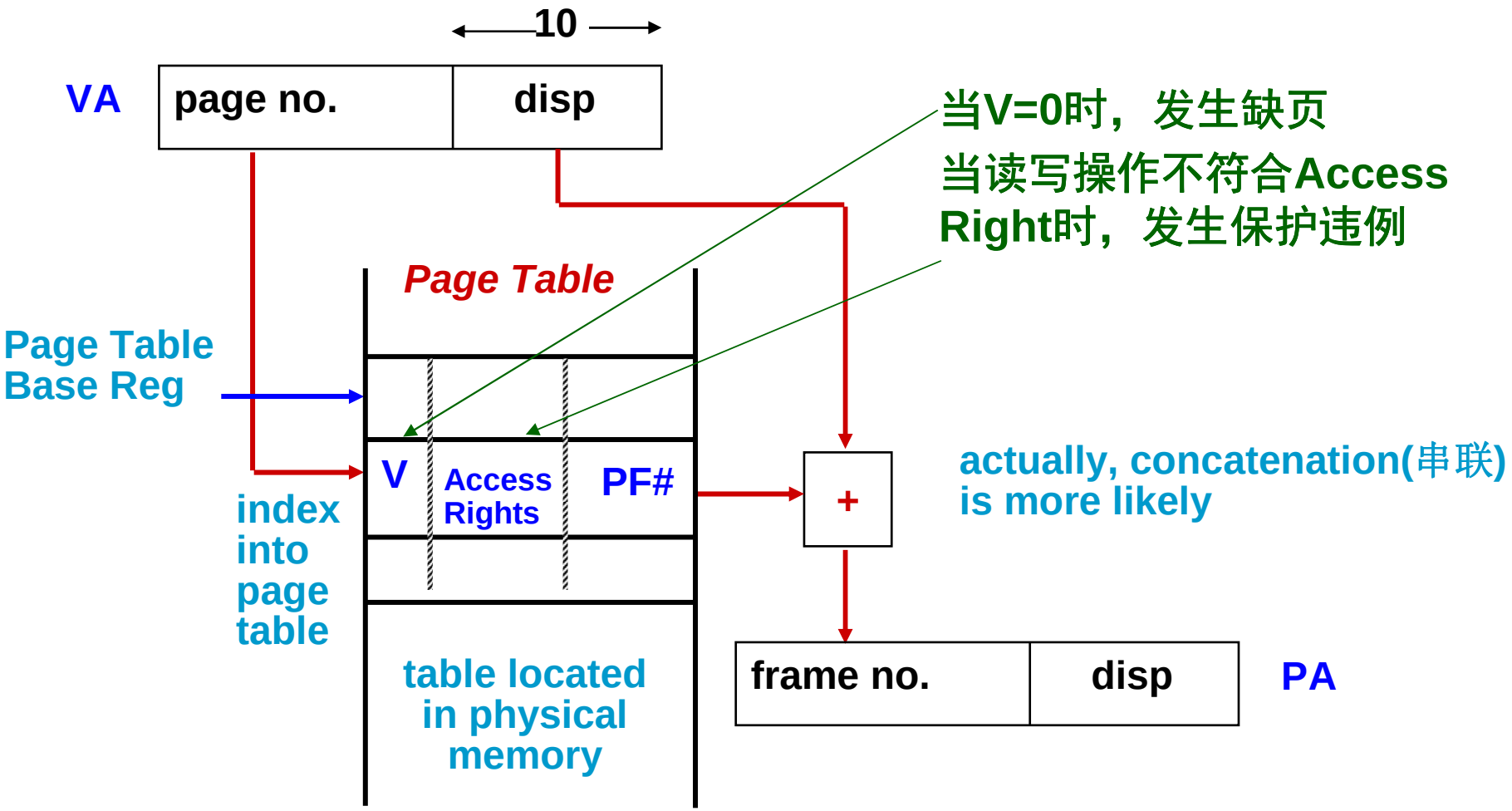
逻辑地址转换为物理地址的过程



问题：虚拟地址几位？虚页号几位？页内偏移量几位？

问题：物理地址几位？页框号几位？

逻辑地址转换为物理地址的过程



信息访问中可能出现的异常情况

可能有两种异常情况：

1）缺页（ page fault）

产生条件：当Valid（有效位 / 装入位）为 0 时

相应处理：从磁盘中读信息到内存，若内存没有空间，则还要从内存选择一页替换到磁盘上，替换算法类似于Cache，采用回写法，页面淘汰时，根据“dirty”位确定是否要写磁盘

异常处理结束后：当前指令的执行被阻塞，当前进程被挂起，处理结束后回到原指令继续执行

2）保护违例（ protection_violation_fault ）

产生条件： 当Access Rights (存取权限)与所指定的具体操作不相符时

相应处理：在屏幕上显示“内存保护错”信息

异常处理结束后：当前指令的执行被阻塞，当前进程被终止

Access Rights (存取权限)可能的取值有哪些？

R = Read-only，R/W = read/write，X = execute only