第8章 虚拟存储器

- **硬件和控制结构**
  - 虚拟页式存储管理
  - 虚拟段式存储管理
  - 虚拟段页式存储管理
- **操作系统软件**
  - 调页、放置和替换策略
  - 驻留集和工作集管理
  - 清除（回写）策略
  - 加载（并发度）控制
- **虚拟存储管理实例**

8.1 硬件和控制结构

- 虚拟存储的关键基础
- 局部性与虚拟存储
- 页式虚拟存储
- 段式虚拟存储
- 段式和页式结合的虚拟存储
- 共享与保护

虚拟存储的关键基础

- **动态地址转换（重定位）**

进程中的逻辑地址在运行时动态转换成物理地址，从而进程可被交换出/入内存，前后所占内存位置可以不同

- **不连续分配**

进程可分成几块(页/段)，且这些块可分别存储到内存的不连续区域里

虚拟存储的关键基础

- **部分加载**

运行时进程的所有页/段不必都在内存里，只要下一条要执行的指令和下一个要访问的数据在内存里即可

部分加载方式与程序的执行

- 操作系统仅把程序起始的一个或几个块装进内存
- 页表/段表表项中有二进制位指示块是否在内存中，进程中驻留在内存的部分称为**驻留集**
- 若逻辑地址访问不在内存中的块，则产生一个访问内存错误的中断

部分加载方式与程序的执行

- 操作系统响应中断

① 将进程置于阻塞态

② 发出一个磁盘I/O请求，将逻辑地址访问的块读入内存

③ 在执行I/O操作期间，分派另外一个进程运行

④ 磁盘I/O完成时产生一个中断，操作系统把受影响进程置于就绪队列

部分加载与虚拟存储

- 采用**部分加载**，内存中可同时容纳更多的进程
  - 每个进程都只加载一部分，更多进程中应该也会有更多的就绪进程，从而提高CPU利用率

部分加载与虚拟存储

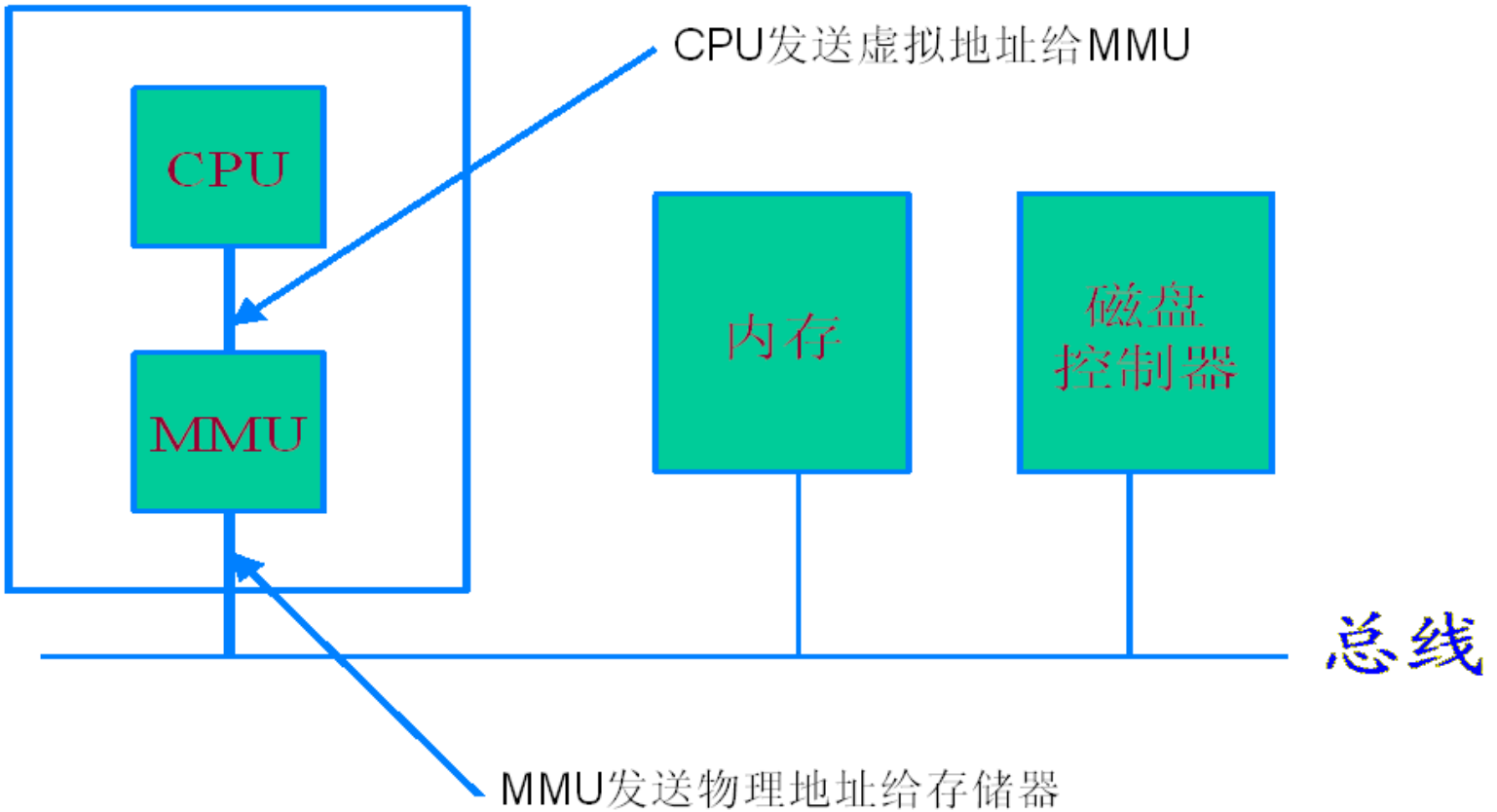
- 采用**部分加载**，进程可以比内存大，实现了**虚拟存储**
  - 用户程序可以使用的独立于物理内存的逻辑地址单元组成存储空间(**虚拟存储**)
  - 逻辑地址空间可以比物理地址空间大，例如，设物理内存64KB，1KB/页，则物理地址需要16位，而逻辑地址可以是28位！
  - 虚拟存储由内存和外存结合实现

虚拟存储技术的特征

- 不连续性
  - 物理内存分配的不连续
- 部分交换
  - 与交换技术相比较，虚拟存储的调入和调出是对部分虚拟地址空间进行的
- 大空间
  - 总容量不超过物理内存和外存交换区容量之和

虚拟存储(virtual memory)

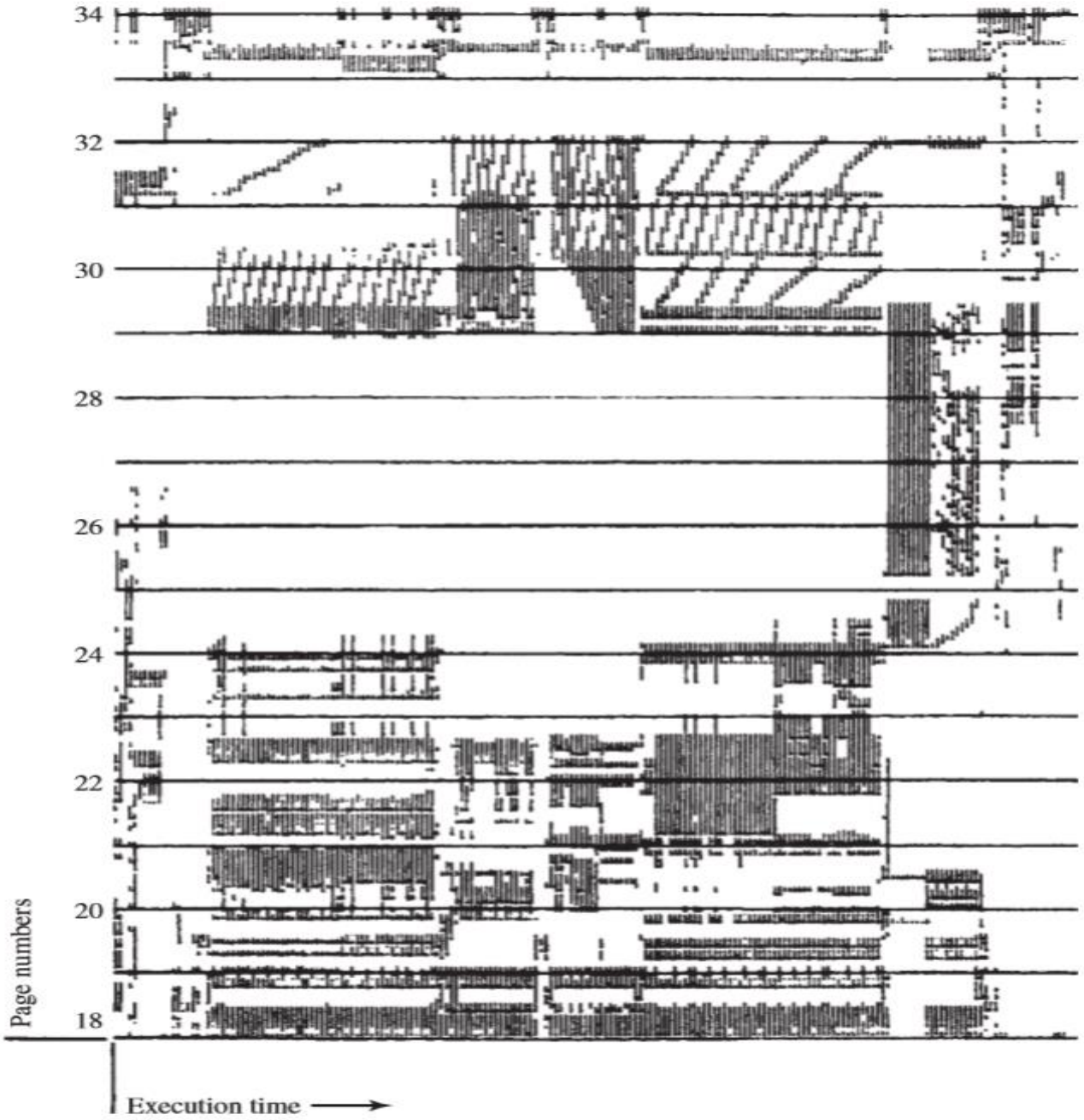
- 虚拟存储的实现模型
  - MMU = Memory Management Unit， 存储管理单元，位于CPU内



8.1.1 程序局部性与虚拟存储

- **局部性原理（principle of locality）**
  - 一段时间内进程的运行往往呈现出高度的局部性，表现为只运行某一段程序，只访问某一块数据区
- **空间局部性**
  - 在执行期间的短时间段内，访问的地址集合聚集在程序的某个局部区域中
- **时间局部性**
  - 模块在程序运行期间只在某些很短的时间段被调用

程序局部性



虚拟存储的要求与抖动问题

- **虚拟存储管理要求**
  - 为了容纳更多进程，每个进程只有一小部分在内存中
  - 如果内存满时操作系统若要调入新的块，则必须要把内存中的某一块换出去
  - 操作系统必须在新的块要使用到之前就换入它
- **抖动（thrashing）问题**

虚拟存储的要求与抖动问题

- **抖动（thrashing）问题**
  - 交换操作过于频繁，导致CPU忙于处理“交换”或等待，而非执行用户程序（页交换涉及外存读写，比内存访问慢）
- 局部性原理保证虚拟存储系统的可行性和效率性：
  - 内存可容纳更多进程
  - 算法优化交换效率，避免抖动，如根据最近的历史记录猜测哪些块最近最不可能使用到

虚拟存储必要的支持

- 硬件支持
  - 内存管理硬件必须要支持分页/分段所需的动态地址转换等，如早期Unix因运行平台的处理器不支持分页/分段而不支持虚存
  - 除一些老式计算机操作系统（如MS-DOS）和专用系统外，当前主流操作系统均支持虚存

虚拟存储必要的支持

- 软件支持
  - 操作系统必须管理内存与外存之间的页/段/段&页的交换
  - 调页策略、放置策略、替换策略
  - 驻留集和工作集管理
  - 清除（回写）策略、加载（并发度）控制

8.1.2 虚拟分页

虚拟地址
Virtual address

页号　　偏移量

页表项
Page table entry

其他控制位　　帧号

P = present bit 存在位
M = modified bit 修改位

页表

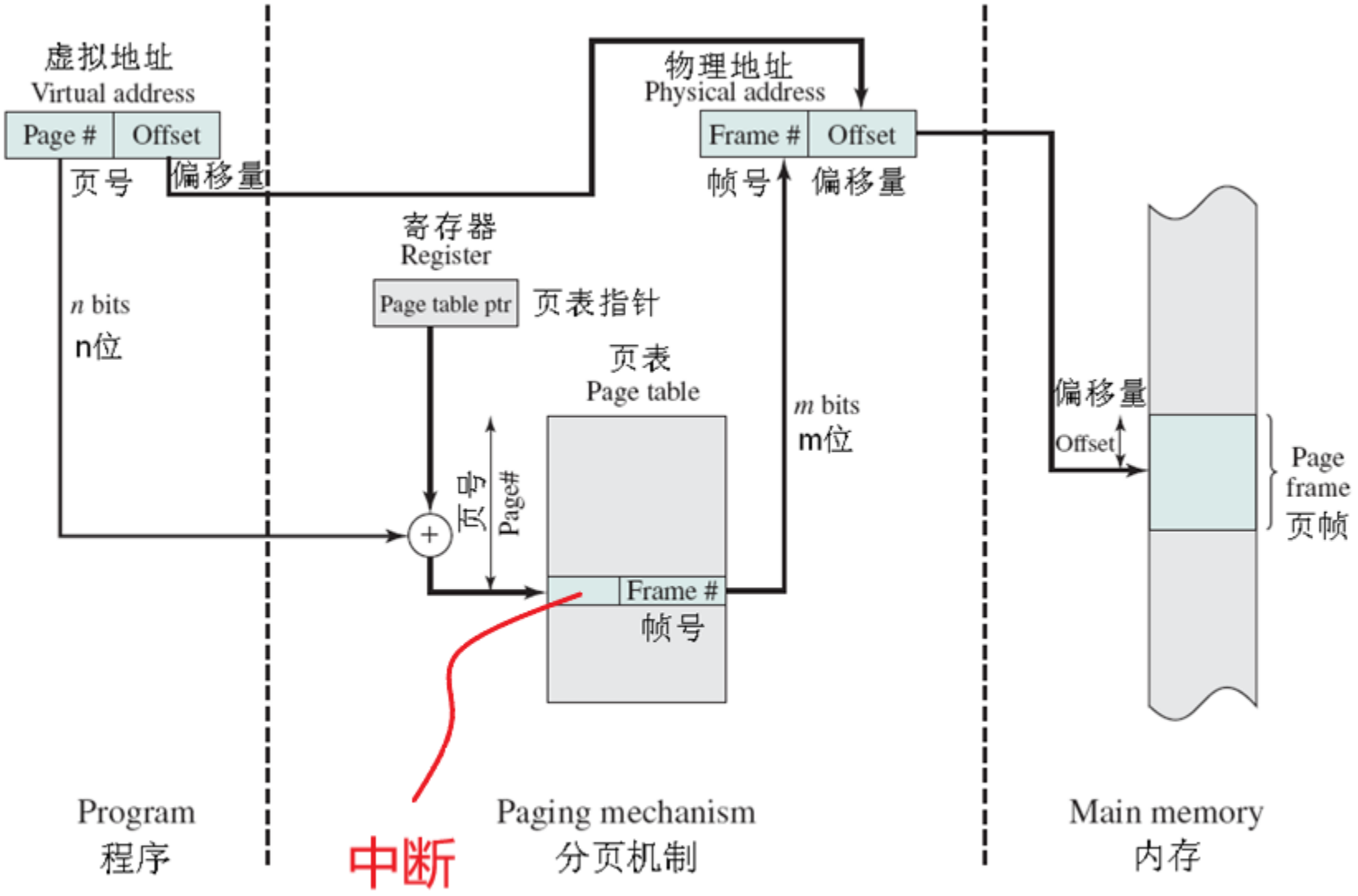
- 页表项（Page Table Entry，简称为PTE）包括：
  - Present：在/不在内存
  - Modified：有没有被修改
  - Protection：保护码，1位或多位(rwe：读/写/执行)
  - Referenced：有没有被访问
  - Cache：是否禁止缓存

页表

- 页表长度不定，取决于进程大小
  - 不适合用寄存器存储页表，而是存放在内存
- 页表起始地址保存在一个CPU专用寄存器

（Intel CPU的为CR3）

页式虚拟存储的地址转换



Address Translation in a Paging System

分页系统中的地址转换

页表组织方法

- 页表可能会非常大，从而占用内存也非常大
  - 例：设逻辑地址为32位，每页(2^{12}=) 4KB，则一个进程的页表就可能有2^{20}个表项
- 通常也对页表进行分页
  - 进程运行时，部分页表必须在内存里（包括正在使用页面所对应的表项）

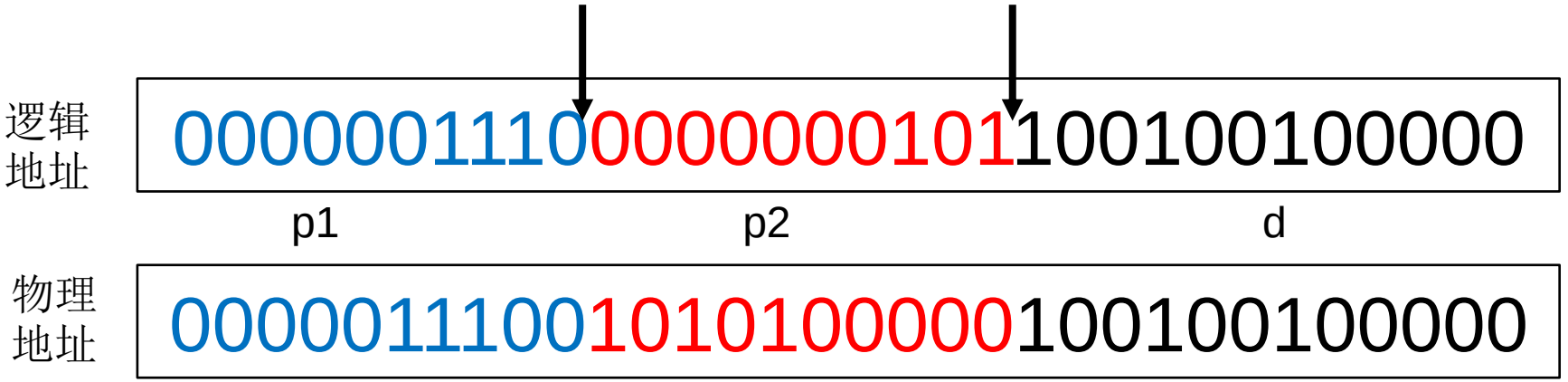
页表组织方法

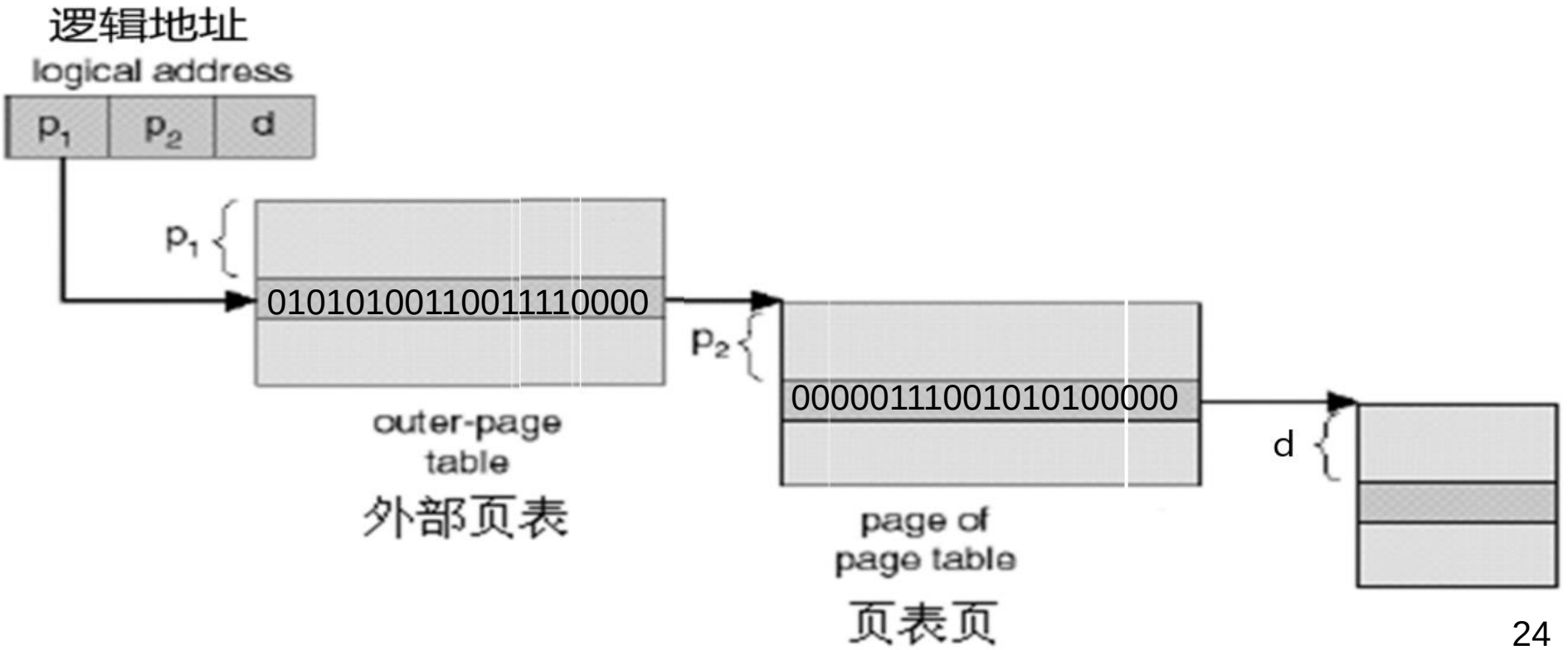
- 多级页表
  - 既然一张页表通常需要几个页框来存储，一种页表的组织结构是多级层次，采用多级层次组织的页表称多级页表
- 反向页表
  - 如PowerPC、SPARC、IA-64的反向页表

多级页表

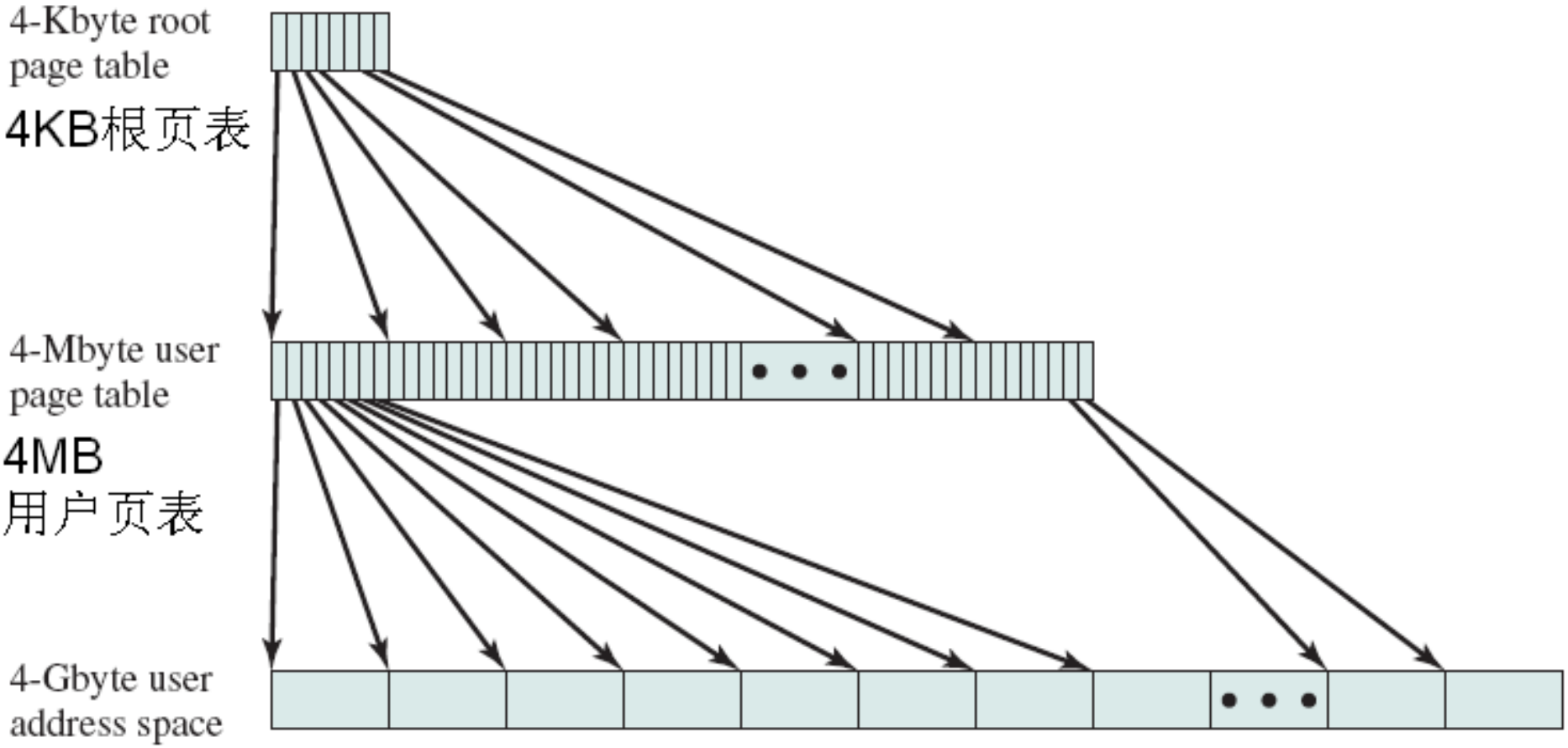
- 多级页表是一种多级层次组织结构
  - 若采用二级页表（如32位的x86 CPU），则页号被划分成两个域：PT1和PT2
  - 顶级页表（内存中）以PT1为索引，其表项指向二级页表，二级页表以PT2为索引
  - 除顶级页表外的其他页表可以在内外存间交换
  - 对64位处理器，一般采用三级页表
    - Linux为了通用，采用支持64位处理器的三级页表结构，对32位CPU，设中间页表的表项个数为1解决

多级页表


逻辑地址
000000111000000000101100100100000
p1
p2
d
物理地址
000001110010101000001100100100000


逻辑地址
logical address
p1
p2
d
p1
01010100110011110000
outer-page table
外部页表
p2
00000111001010100000
page of page table
页表页
d

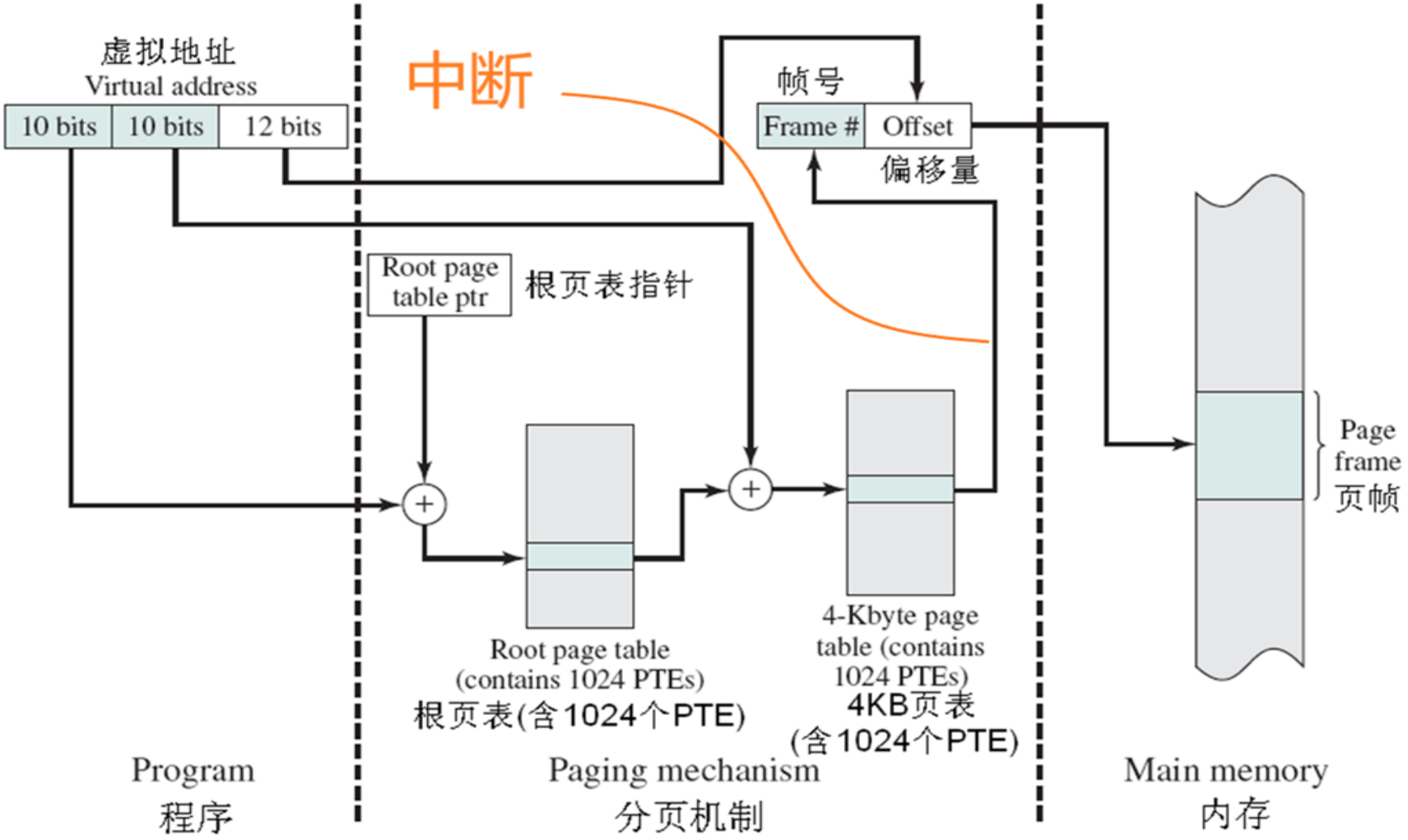
二级页表



A Two-Level Hierarchical Page Table

二级层次页表

二级页表的地址转换



Address Translation in a Two-Level Paging System

二级分页系统中的地址转换

64位处理器与反向页表

- 对64位CPU，若页面大小为4KB，则页表有2^{52}个表项，如果每个表项占8字节，则整个页表需占$8*2^{52}$ B=32PB=32768TB存储空间！
- 当物理内存远小于虚拟内存（CPU的可寻址空间）时，在内存中创建存储虚拟存储空间中的页所对应的所有页表项，显然是不明智的

反向页表

inverted page table

反向页表

- **实际内存**的每个页框对应一个页表项（不是每个虚拟内存的页有一个页表项）
- 页表项的内容为（进程ID，页号）= (n, p)，记录定位于该占用页框的进程号和页号

控	制	位	页号	Pid
1	0		101	273
1	0		103	273
0	0		47	136
1	0		59	136
0	0		71	136
			…	

反向页表

- 优点：当物理内存较小时，反向页表可大量节省空间
- 缺点：从虚拟地址转换到物理地址变得非常困难（不能使用CPU所提供的页框号映射机制，需搜索整个反向页表，查找对应于页表项(n, p)的页框号）

控	制	位	页号	Pid
1	0		101	273
1	0		103	273
0	0		47	136
1	0		59	136
0	0		71	136
			…	

TLB和散列表

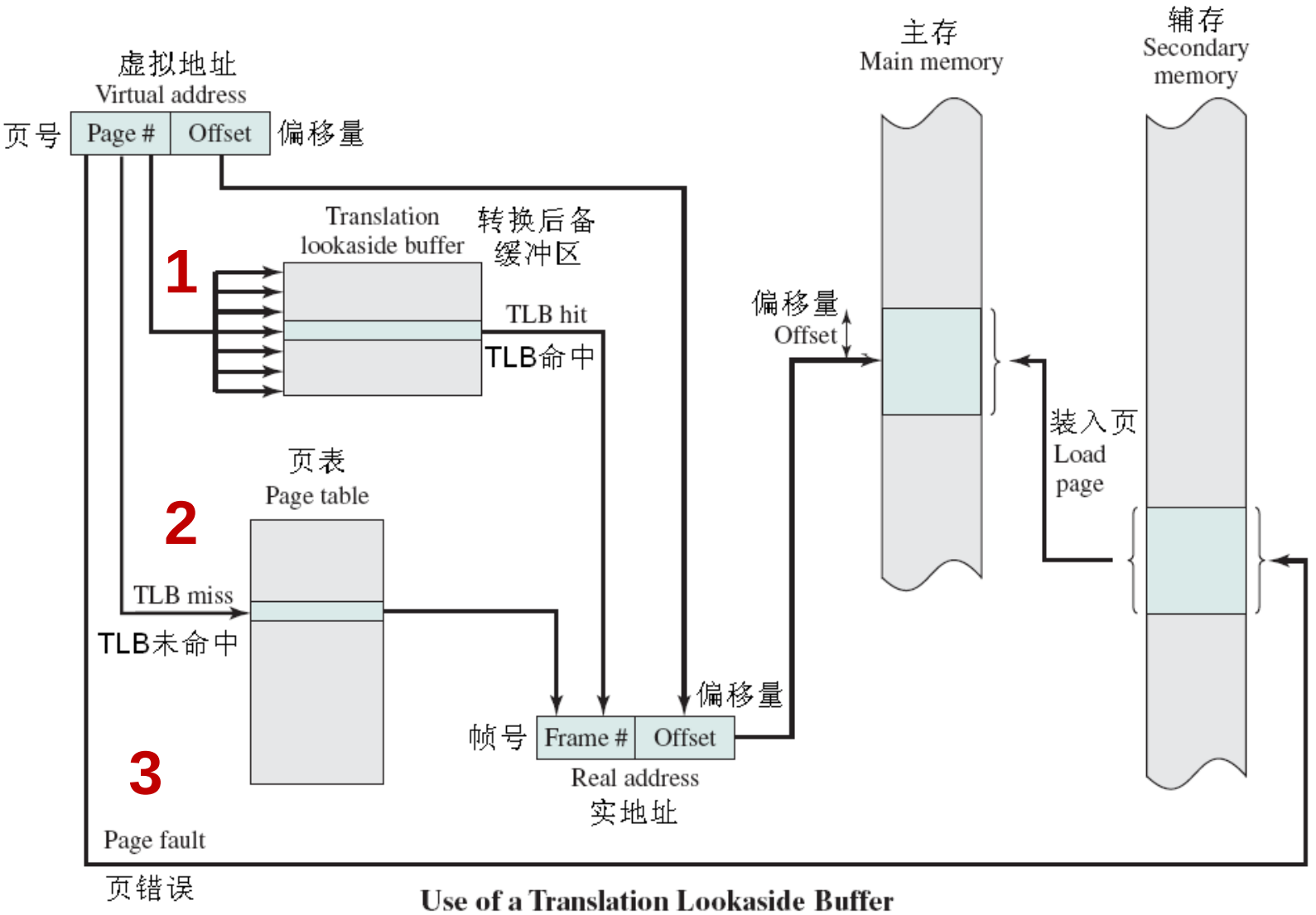
反向页表详细流程

来源： https://www.jianshu.com/p/8a86033dfcb0

转换后备缓冲区TLB

- TLB = Translation Lookaside Buffer（查表缓冲区/转换缓冲区，缓冲页表）
- 页表存储在内存，每次内存引用至少要访问两次内存，大大影响效率
- 设置特殊的硬件装置——TLB缓存页表（联想存储器）
  - 存储少量最近最常用的页表表项
- TLB表项内容：有效位、页号、修改位、保护码、页框号等

使用TLB的地址转换



Use of a Translation Lookaside Buffer

转换后备缓冲区的用法

地址转换

- 使用TLB的地址转换工作流程
  a) 给定一个逻辑地址，CPU首先检查TLB，判断页号是否存在
  b) 若存在（命中，hit），则直接从TLB中提取页框号并形成物理地址
  c) 若不存在（不中/未命中，miss），则按普通访问页表方式工作，形成物理地址，并更新TLB（新页表表项替换一个TLB表项）

地址转换

- 不使用TLB的址转换工作流程
  - 由页号去页表检查该页在不在内存（P位）
  - 若在，则形成物理地址
  - 若不在，则产生页错误(Page Fault)并发出缺页中断，由操作系统将页调入内存并更新页表，进而形成物理地址

TLB的一些细节

- 逻辑地址中的页号与TLB表项的匹配检查由硬件实现，是并行的——关联映射
- TLB中每个表项的页号部分必须包含页号的**所有域**，只有整个页号匹配时才算命中
- TLB应随着进程的切换而刷新：
  - 提供一条清除TLB中有效位的机器指令
  - 扩充TLB使包含一个进程标识域，同时增加一寄存器保存当前进程标识符

页大小问题

- 页大小是一个重要的硬件设计问题
  - 小页有利于减少内碎片总量
  - 大页有利于减小每进程的页表容量
  - 大页有利于实现有效的磁盘数据块传送

最常用的页大小介于 1KB～8KB

页大小问题

- 有些处理器支持多种页大小（也是发展趋势），例如：
  - x86支持4种：4KB、4MB、 2MB、 1GB
  - IA-64支持8种：4KB、8KB、64KB、256KB、1MB、4MB、16MB、256MB
  - R4000（ MIPS 于1991年10月1日发布的64位CPU）支持7种：4KB到16MB

页大小与缺页率

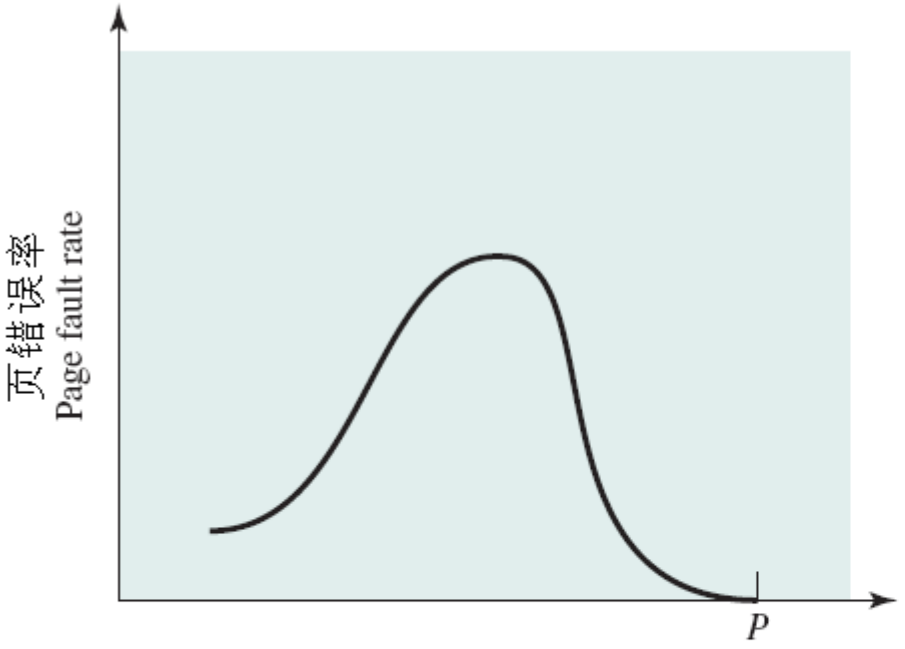
- 页大小会影响缺页率
- 缺页率
  - 缺页次数/内存访问次数
  - 缺页的平均时间间隔
    - 页很小：每个进程的内存页较多，通过调页很快适应局部性原理的要求，缺页率低
    - 页很大：进程使用的大部分地址空间都在内存，缺页率低
    - 页中等大小：局部性区域只占每页的较小部分，缺页率高

页大小与软件策略

- 页大小也受软件策略影响
- 页大小固定时，缺页率与分配给进程的内存页框数目的关系：
  - 分配给进程的内存页框数 可少于进程所需页总数
  - 数目越多，缺页率越低
  - 页框数目的下限是一条指令及其操作数可能涉及页数
    - 足以保证进程的每条指令都能被执行

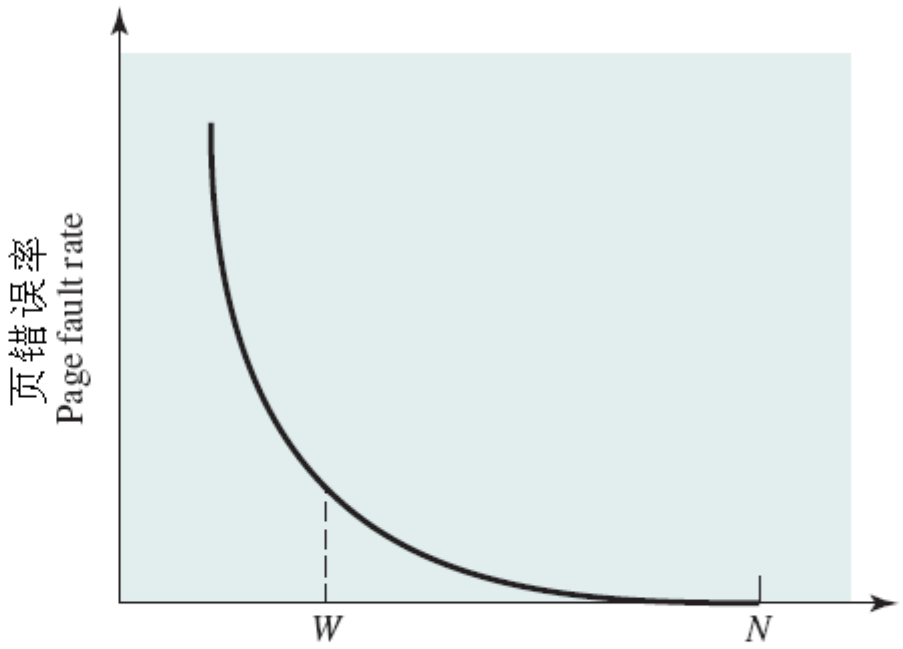
缺页率与页大小、软件策略

页大小与缺页率



(a) Page size
页大小

页框数量与缺页率



(b) Number of page frames allocated
分配的页帧数

P = size of entire process 整个进程的大小
W = working set size 工作集大小
N = total number of pages in process 进程的总页数

Typical Paging Behavior of a Program
程序的典型分页行为

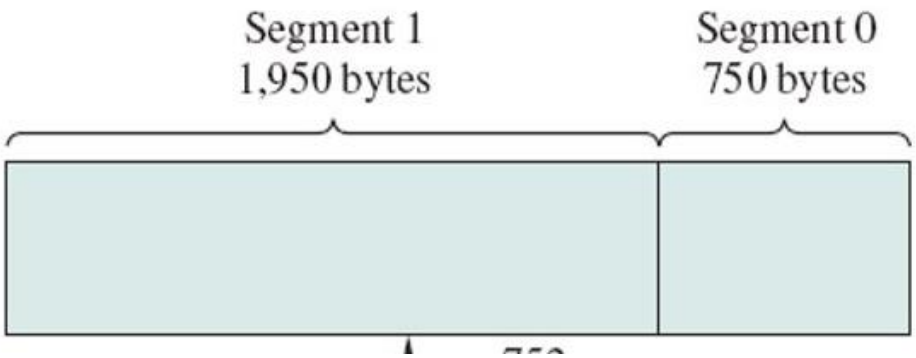
8.1.3 虚拟段式存储管理

- **段式管理的优点**
  - 简化处理动态增长的数据结构
  - 支持模块的独立修改和重编译
  - 更有效的进程共享
  - 更容易实现保护

8.1.3 虚拟段式存储管理

- **虚拟段式存储管理的组织**
  - 段表：由表项组成，每个进程一张
  - 段表长度不定，存放在内存
  - 当前进程的段表起始地址保存在CPU的一个专用寄存器里（如Intel CPU为LDTR/GDTR）
  - 进程中有的段可能不在内存中（位于外存）

段式地址与段表表项



无虚拟内存的分段

虚拟内存
Virtual address

Segment number	Offset

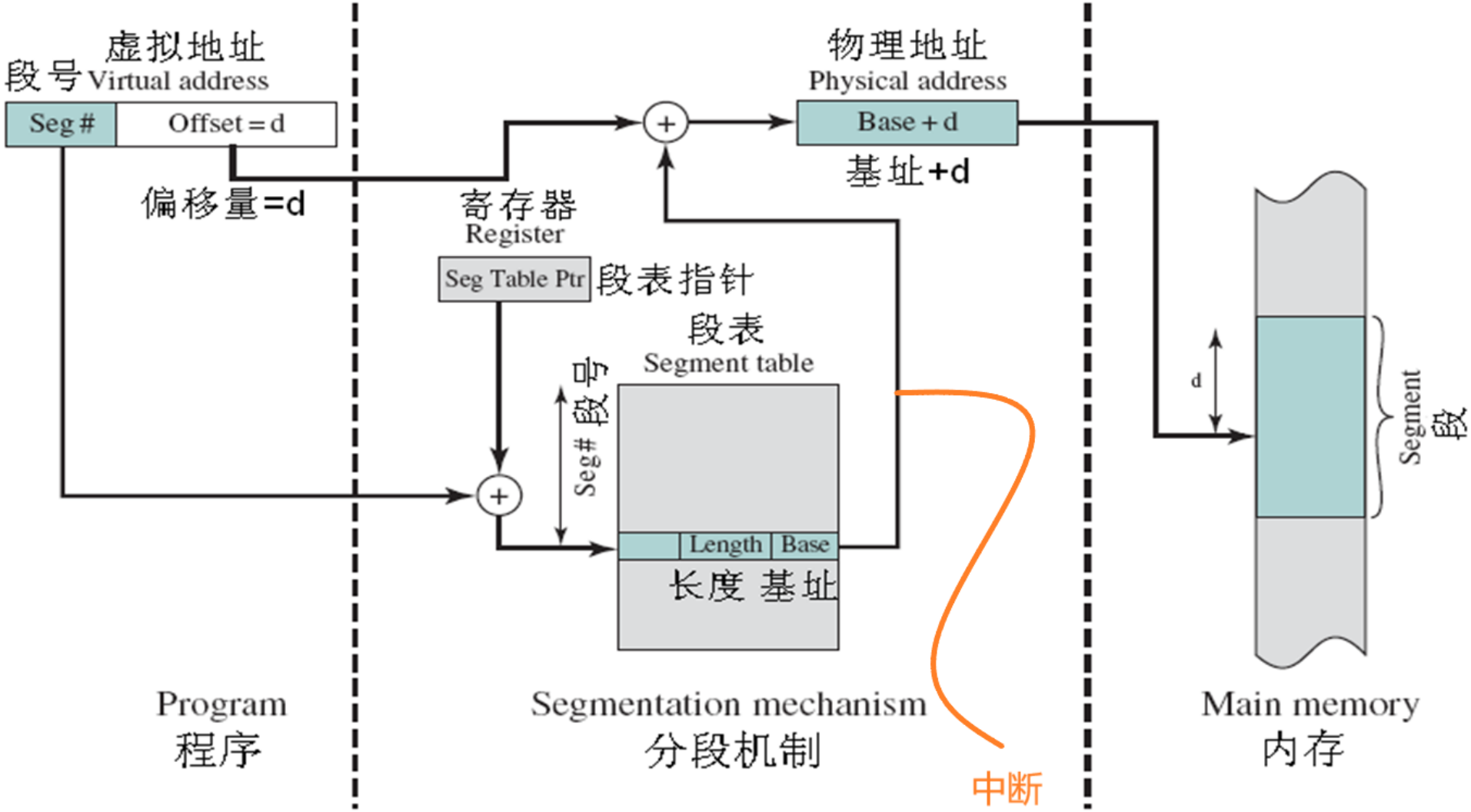
段号　　偏移量

段表项
Segment table entry

P	M	Other control bits	Length	Segment base

其他控制位　　长度　　段基址

虚拟段式存储的地址转换



Address Translation in a Segmentation System

分段系统中的地址转换

8.1.4 分段和分页优缺点对比

	优势	不足
分页	➢ 内存利用率高 ➢ 消除了外部碎片 ➢ 固定分页大小易于存储管理	➢ 程序员透明（不可见） ➢ 不方便按逻辑模块共享与保护*
分段	➢ 程序员可见 ➢ 动态应对数据增长 ➢ 支持共享和保护	➢ 段长过大易导致外部碎片

* 程序的页结构和数据对程序员不可见

8.1.4 虚拟段页式存储管理

- 结合分页和分段的优点，克服两者的缺点
- 基本原理
  - 将程序按逻辑结构划分成若干段，每个段进一步划分成若干个页
  - 将内存划分成许多小的页框，页框与页大小相等
  - 操作系统为每个进程建立并维护一个段表，为每个段建立并维护一个页表

8.1.4 虚拟段页式存储管理

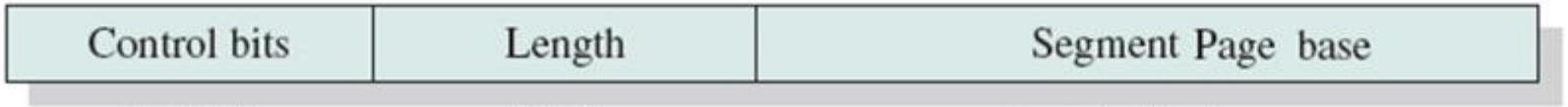
- 段表和页表的表项分别类似段式、虚拟页式
  - 段表表项中的段起始地址是该段的页表起始地址
  - Present位和Modified位只含在页表表项中
  - 保护和共享位通常在段表表项中
  - 只有缺页没有缺段

段表和页表

虚拟地址
Virtual address

段号　　页号　　偏移量

段表项
Segment table entry



控制位　　长度　　段页表基址

页表项
Page table entry

其他控制位　　帧号

段页式中的逻辑地址

- 程序中的逻辑地址包含由两个部分

  (段号, 段内偏移)

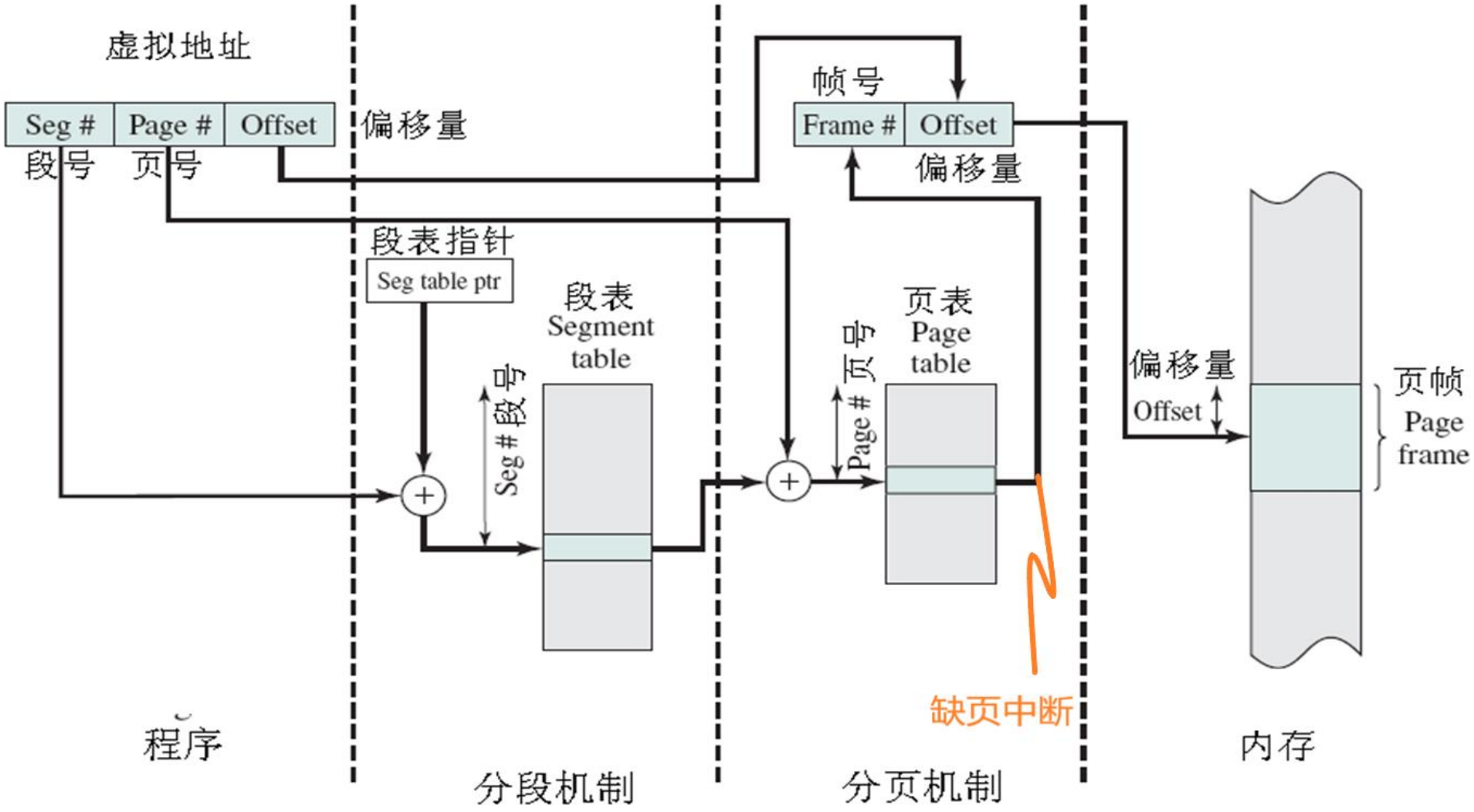
- 操作系统将程序中的逻辑地址分解为三个部分

  (段号, 页号, 页内偏移)

- 逻辑地址(n, p, d)中，程序员可见的是段号 n 以及段内偏移，段内偏移分解为页号 p 及页内偏移 d 对程序员透明（假设页大小是 2 的 m 次幂）

- 地址转换：综合分页和分段

虚拟段页式存储的地址转换



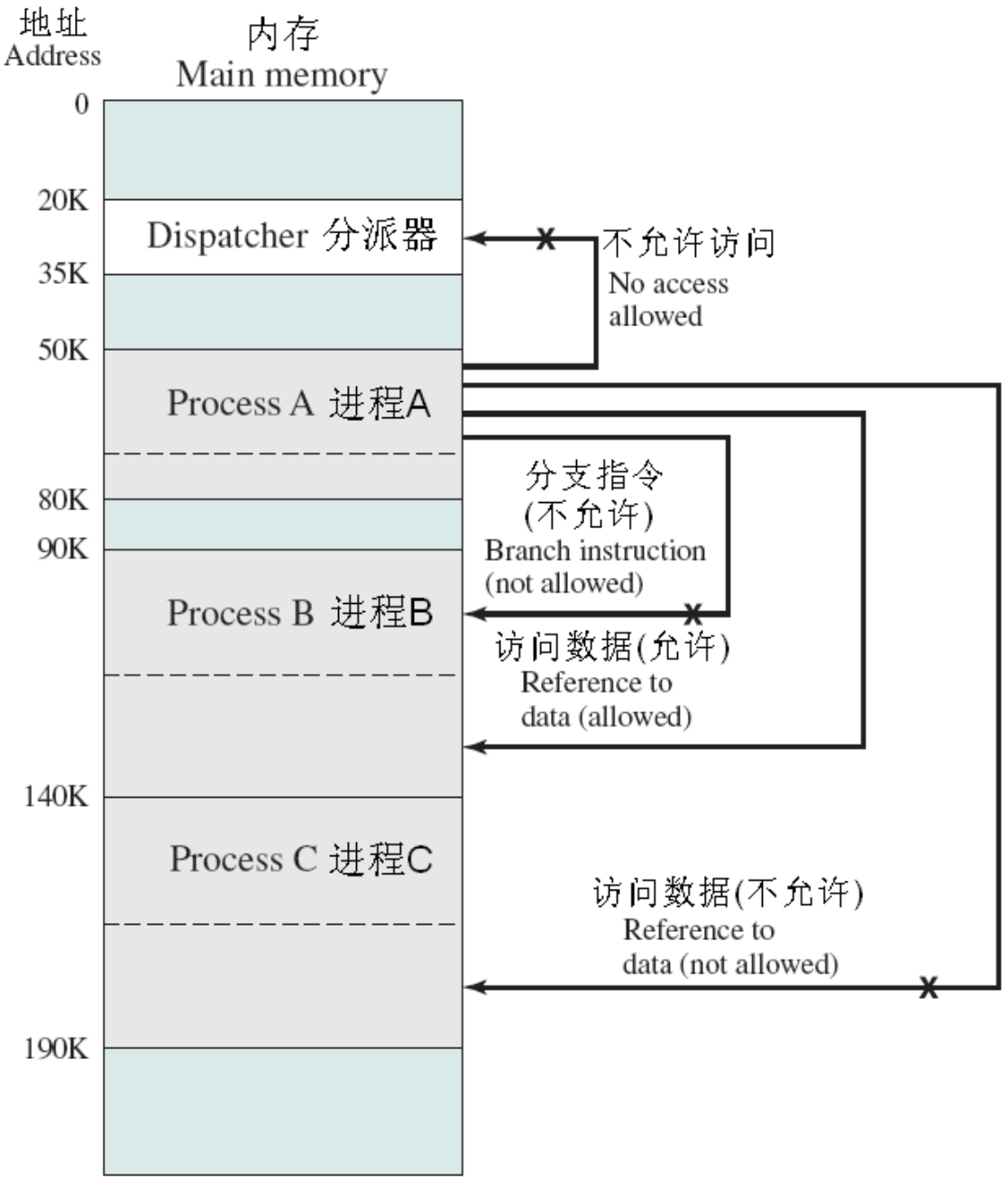
Address Translation in a Segmentation/Paging System

分段/分页系统中的地址转换

共享和保护

- 共享
  - 不同进程间可以共享代码段和数据段
  - 实现：段表表项记录相同的段起始地址
- 保护
  - 越界保护（段基址&段长）
  - 访问方式保护（权限，如读写保护）
  - 环保护（模式&级别）

共享和保护

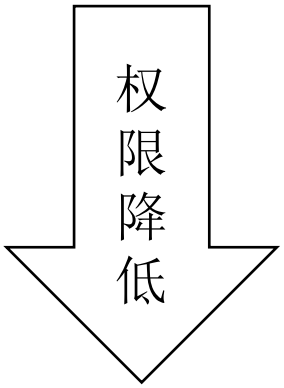


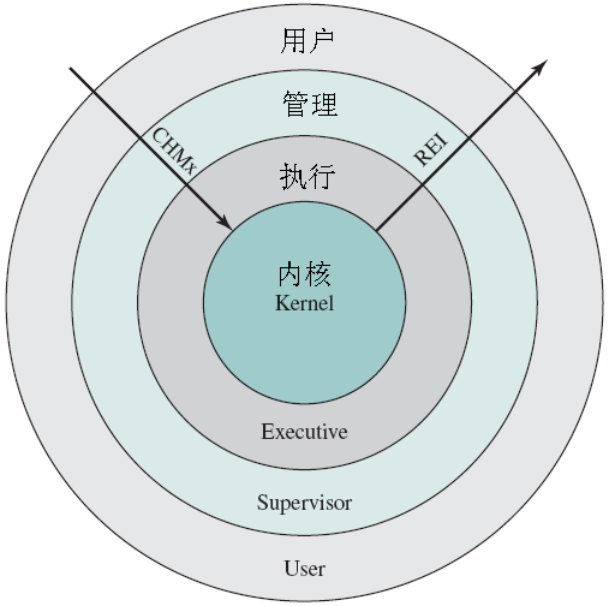
Protection Relationships between Segments

段之间的保护关系

环保护（ring-protection）

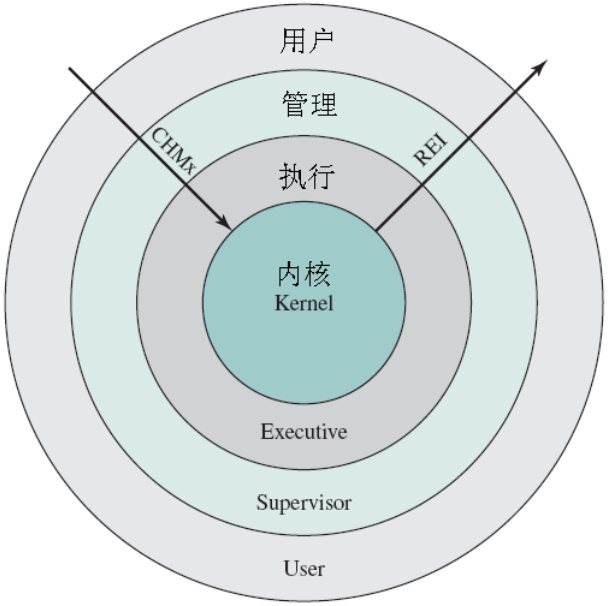
- 分层访问模式：
  - 中心0号环为内核模式
  - 内环1号环为执行模式
  - 中环2号环为管理模式
  - 外环3号环为用户模式





环保护（ring-protection）

- 程序只能访问同层或更外层的数据
- 程序可以调用同层或更内层的服务
- CHM（change-mode，改变模式）
- REI（return from exception or interrupt，从异常或中断返回）



8.2 操作系统软件

- 设计时三个基本选择问题
  - 是否支持虚存技术
  - 是否支持页式/段式/段页式
  - 采用的算法

取决于硬件平台

- 软件设计的主要问题
  - 纯段式系统越来越少，段通常被分页，虚拟存储管理的问题主要就是虚拟页式存储管理的问题

目标：缺页率(Page Fault Rate)最小！

缺页中断产生
巨大开销！

虚拟存储管理软件的各个方面

- 调页策略（Fetch Policy）

无绝对最佳策略！

- 放置策略（Placement Policy）
- 替换策略（Replacement Policy）
- 驻留集和工作集管理（Resident Set and Working Set Management）
- 清除（回写）策略（Cleaning Policy）
- 负载（并发度）控制（Load Control）

8.2.1 调页策略

- 决定何时将页载入内存
- 两种常用策略：
  - 请求调页(demand paging)：只通过响应缺页中断调入需要的页，也**只调入发生缺页时所需的页**
    - 进程开始运行时会有许多缺页，对外存I/O次数多，开销较大（一次I/O操作包括旋转等待时间和读写时间）

8.2.1 调页策略

- 决定何时将页载入内存
- 两种常用策略：
  - 请求调页(demand paging)
  - 预先调页(prepaging)：在发生缺页需要调入某页时，一次调入该页以及相邻的几个页
    - 提高调页的I/O效率
    - 效率不能保证：额外装入的页可能没用

8.2.2 放置策略

- 决定进程各部分驻留在内存的哪个位置
- 纯段式系统（动态分区，涉及外碎片、压缩操作等）
  - 最佳适配、首先适配、……
- 纯页式或段页式系统
  - 地址转换、内存存取原理相通

8.2.3 替换策略

- 需要调入页而内存已满时，决定置换（淘汰）内存中的某些页框
- 经常要进行替换（为提高并发程度，操作系统总是载入尽量多的进程）
- 不是所有内存中的页都可以被替换：

8.2.3 替换策略

- 并非所有内存中的页都可以被替换：
  - 锁定帧(locked)：操作系统内核、关键控制结构、I/O缓冲区等
  - 驻留集策略决定了不同的替换范围：被替换的页框局限在本进程，或允许在其他进程

基本替换算法

- 最优算法（OPT，OPTimal）
- 最近最少使用算法（LRU，Least Recently Used）
- 先进先出算法（FIFO，First-In-First-Out）
- 时钟算法（Clock）

最优算法OPT

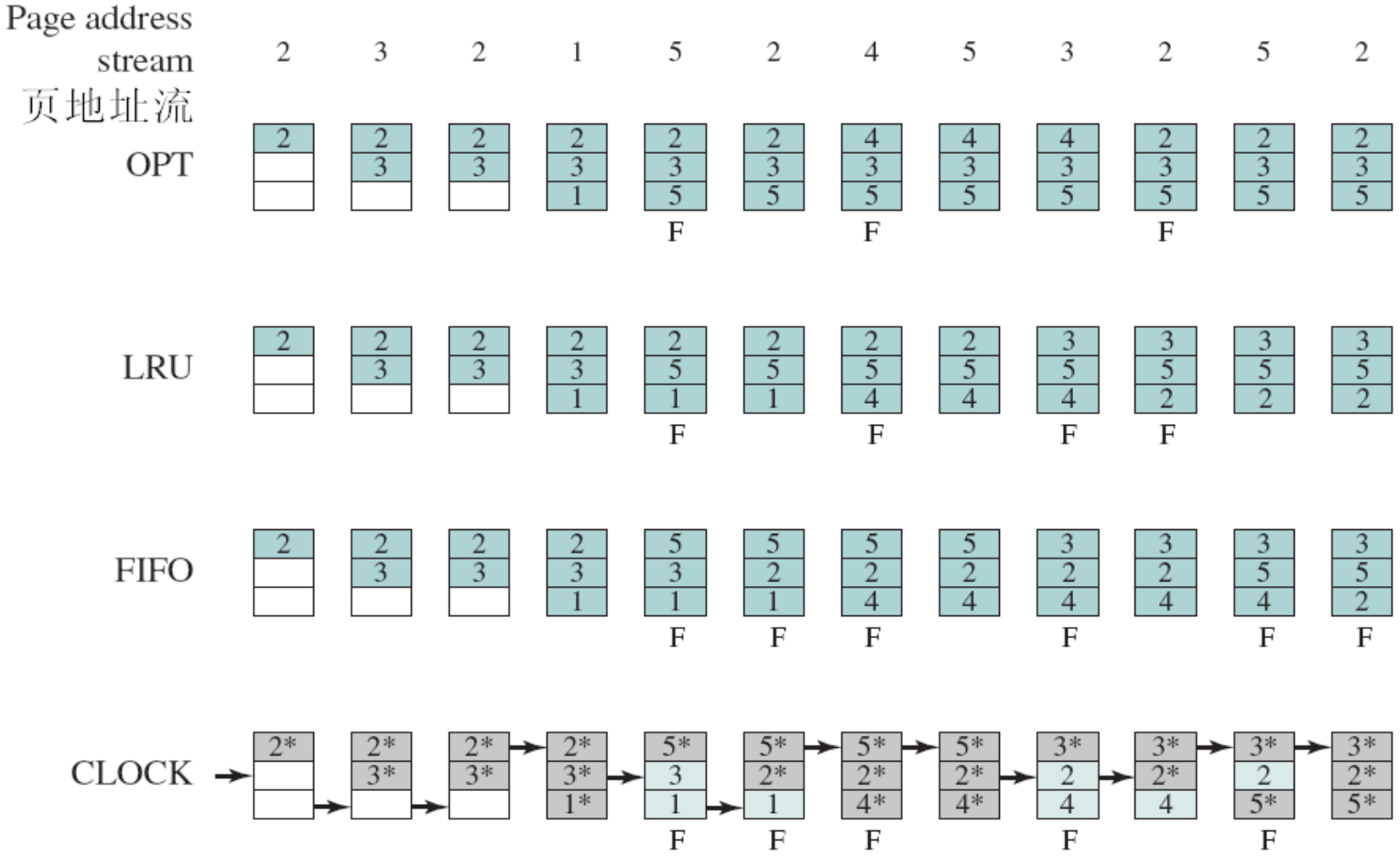
- 淘汰“未来不再使用”或“还要最长时间才会使用”的那个页框
- 效果最佳，**实现困难（实际执行难以预知）**
- 可用作其他算法的性能评价依据

最近最少使用算法 LRU

- 淘汰内存中最近最少使用的页框
- 性能接近最佳算法(局部性原理的合理近似)

示例

- 一个进程有5个页，系统规定进程最多占3个页框



Behavior of Four Page Replacement Algorithms

四种页面替换算法的行为

LRU的实现

- 需要记录页框使用时间的先后关系，开销高昂
  - 链表：每次内存访问后在链表中找到对应的页框，把它移到表头，因此表尾的就是最久未使用的
  - 硬件移位计数器：每个页框关联一个计数器，每次访问该页框，计数器就加1；
  - 每次内存访问后将当前计数器的值写到相应的页表表项里，计数器每隔一段时间右移一位。
  - 缺页时，计数器值最小即为最近最少使用。

先进先出算法 FIFO

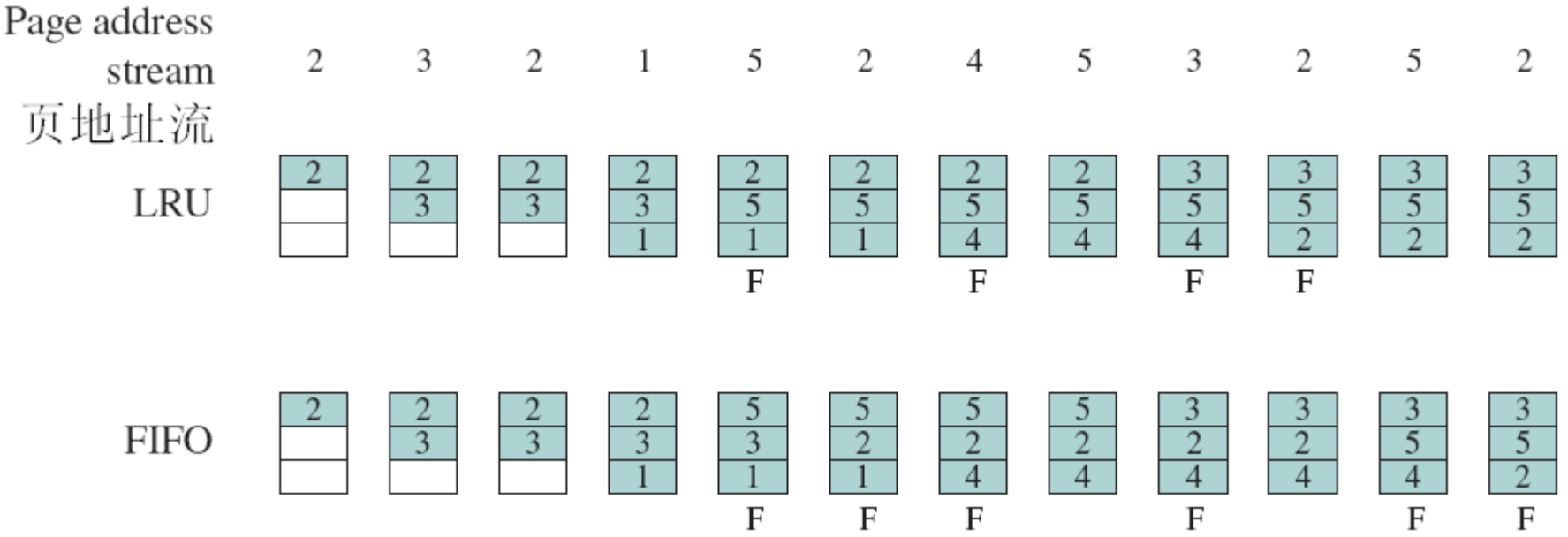
- 淘汰内存中最“早”页
- 实现简单
  - 可通过链表来表示各页的建立时间先后，新页添加到表尾，表头是最“早”页

先进先出算法 FIFO

- 性能较差
  - 较早调入的页可能是经常被访问的页，这些页在FIFO算法下被反复调入和调出
  - Belady现象：在分页式虚拟存储器管理中，缺页置换算法采用FIFO算法时，如果对一个进程未分配它所要求的全部页框，有时就会出现分配的页框数增多但缺页率反而提高的异常现象（L. Belady，1966）

FIFO算法与LRU算法比较

- FIFO性能较差
- LRU能识别出2和5是最常用的页



Belady现象

- 采用FIFO算法时，如果对一个进程未分配它所要求的全部页框，可能存在页框数增多，缺页率反而提高的异常现象

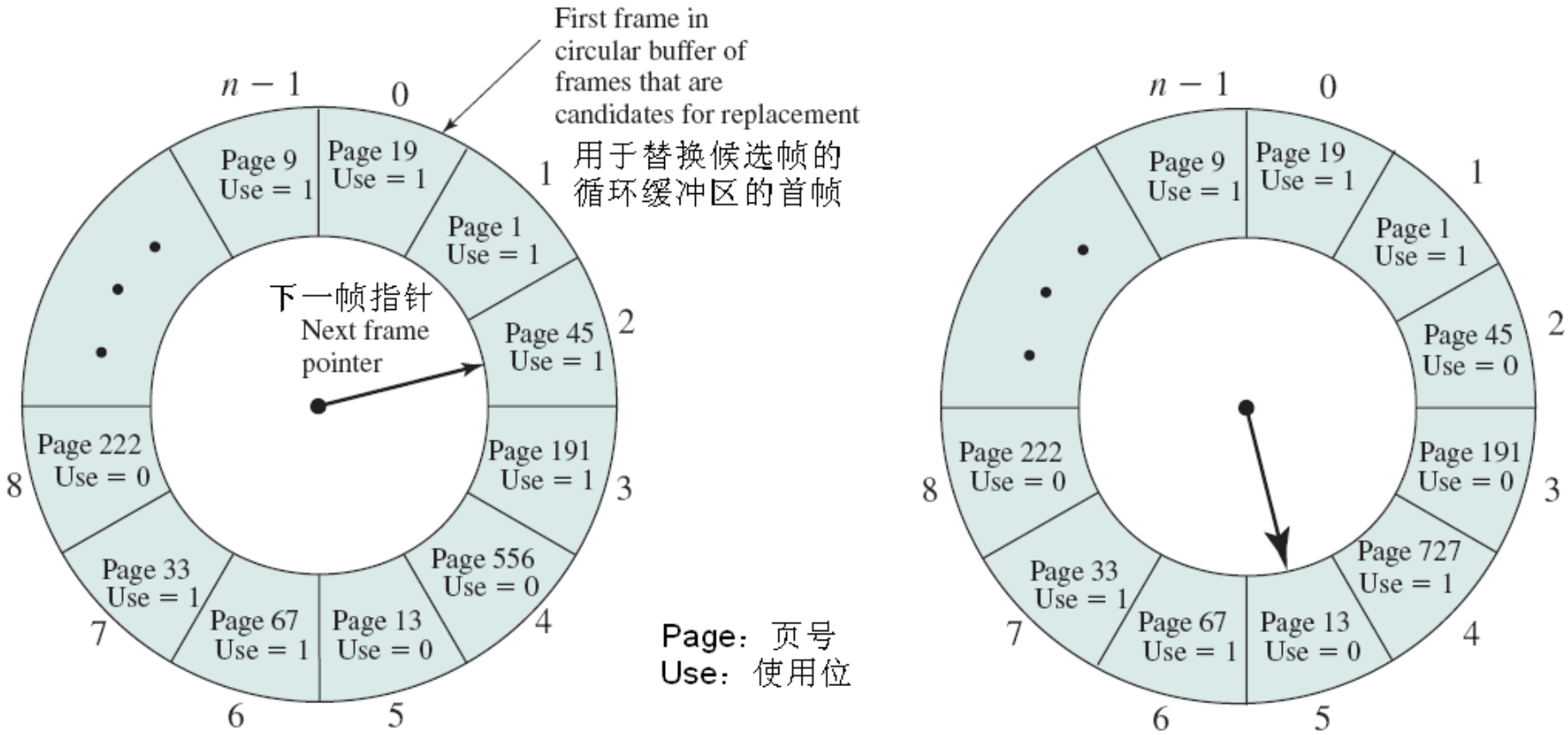
	0	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
最年轻的页	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
最老的页			0	1	2	3	0	0	0	1	4	4
	F	F	F	F	F	F	F			F	F	

	0	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
最年轻的页	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
			0	1	1	1	2	3	4	0	1	2
最老的页				0	0	0	1	2	3	4	0	1
	F	F	F	F			F	F	F	F	F	F

时钟算法（Clock）

- LRU性能近似OPT，但是实现开销大
- Clock性能近似LRU，而且实现开销小
- 环形链表实现算法
  - 环形链表头尾相邻，因此只需要移动一个指针
  - 每页关联一使用位（use bit）R
    - 在页首次被装入时和发生缺页后被访问时，置R为1
    - 在替换算法扫描后，置R为0
  - 当需要置换页时，从指针所在的当前位置开始扫描整个缓冲区，选择遇到的第一个使用位为0的帧进行替换

时钟算法示意图(二次机会算法)



(a) State of buffer just prior to a page replacement

页替换前的缓冲区状态

(b) State of buffer just after the next page replacement

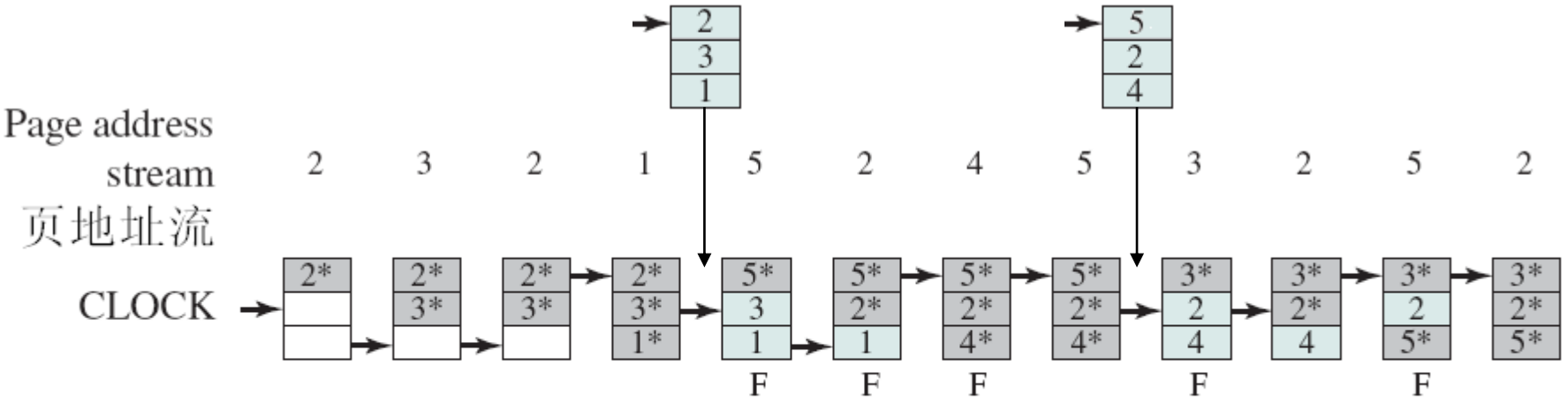
下一页替换后的缓冲区状态

Example of Clock Policy Operation

时钟策略操作例

Clock与LRU、FIFO比较

- *号表示R=1，箭头表示指针
- Clock通过设置R位保护了常用的页



8.2.4 驻留集管理

- 给每个进程分配多少页框，以及如何动态调整各进程的页框数（内存）？
- 驻留集(resident set)指虚拟页式存储管理中给进程分配的物理页框的集合
- 驻留集大小即该集合的页框元素个数

8.2.4 驻留集管理

- 每个进程的驻留集越小，则同时驻留内存的进程就越多，CPU利用率越高
- 进程的驻留集过小，则缺页率高，调页的开销增大
- 进程的驻留集大小达到一定数目之后，再给它分配更多页框，缺页率不再明显下降

管理策略

- 固定分配(fixed-allocation)：在执行过程中进程的驻留集大小固定
  - 各个进程的驻留集大小在进程创建时决定，可根据进程类型，或由程序员、系统管理员决定
  - 替换页面时从各自驻留集中选择

管理策略

- 可变分配(variable-allocation)：在执行过程中进程的驻留集大小可变
  - 可根据缺页率动态调整，性能较好
  - 需要操作系统对活动进程的行为进行评估，增加开销

替换范围

- 缺页发生时从在内存中的哪些页框中选择替换?
- 全局替换(global replacement)：内存中任意非锁定页框均可以被替换
- 局部替换(local replacement)：被替换的页局限在缺页进程的驻留集
  - 容易进行性能分析
  - 性能不一定比全局替换好

驻留集管理组合策略

	全局替换	局部替换
固定分配	无此方案	进程开始前需预先确定分配页框数量：过多影响并发水平，过少使缺页率过高
可变分配	最容易实现的组合，被许多操作系统采用（如Unix） 主要问题：如何决定哪个进程的页将被替换，不利优化（“损人利己”）	试图克服全局替换的问题（具体做法见后） 可能是最佳组合（Windows NT采用）

可变分配+局部替换

- 具体做法：
  - 进程加载进内存时，给它分配一定数目的页框；采用请求调页或预先调页填满这些页框
  - 缺页时，从缺页进程本身的**驻留集**中选择替换一页
  - 定期重新评估进程的**驻留集**大小，并相应增加或减少，以提高系统整体性能
- 比简单的全局替换复杂（基于对进程未来请求的估计改变**驻留集**大小），但性能要好

可变分配 + 局部替换

- 问题：如何调整**驻留集**大小？
  - 工作集策略

工作集（Working Set）

- P. Denning于1968年提出
- 引入工作集，依据进程在过去一段时间内访问的页面调整驻留集大小
- 工作集是一个进程执行过程中某段时间内所访问的页的集合，可用一个二元函数W(t, Δ)表示：

工作集（Working Set）

- 工作集可用一个二元函数W(t, Δ)表示：
  - t 是执行时刻
  - Δ是一个虚拟时间段，称为窗口大小(window size)，它采用“虚拟时间”单位(即实际执行时间，阻塞时不计时)，执行时间来计算可用执行的指令数目或处理器
  - 工作集是在 [t-Δ, t] 虚拟时间段内所访问的页的集合，| W(t, Δ) | 指工作集大小，即页数目

工作集

Sequence of Page References 页访问序列 W	窗口大小 Window Size, Δ 2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

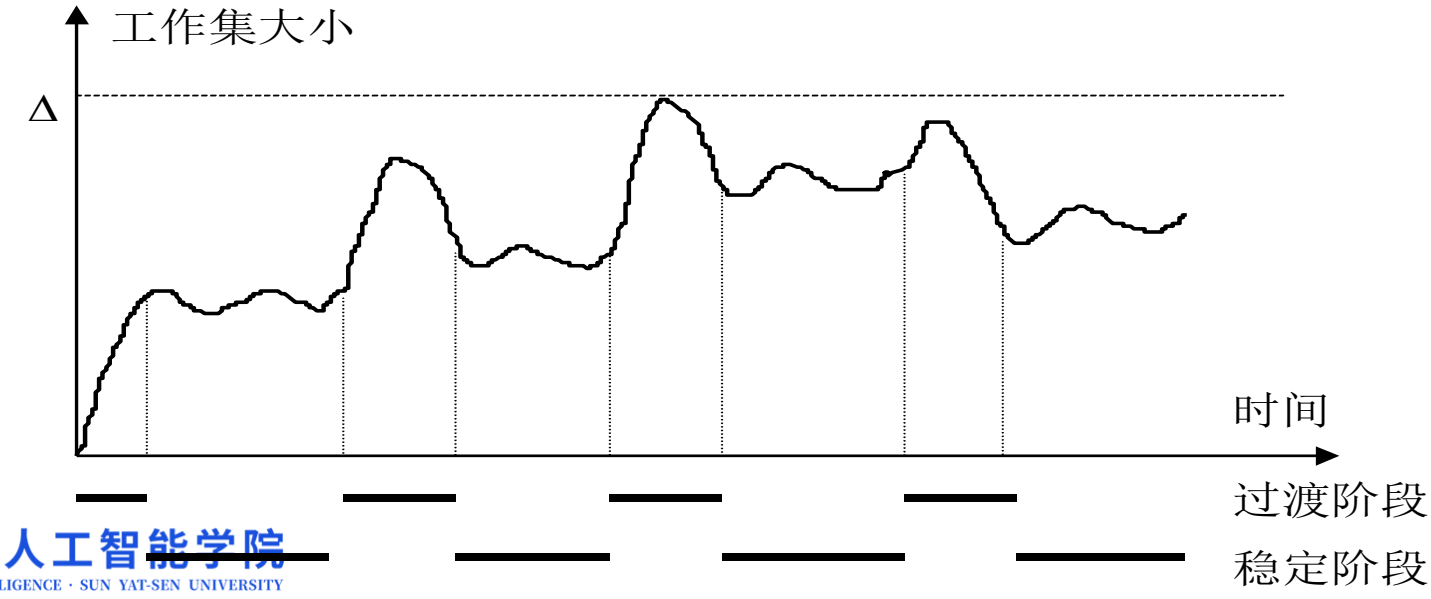
状态不变

Working Set of Process as Defined by Window Size

注：• 表示状态不变 由窗口大小定义的进程工作集

工作集大小的变化

- 进程开始执行时，工作集随着访问新页逐步增大
- 内存访问的局部性区域趋向稳定时，工作集大小也趋于稳定
- 局部性区域的位置改变时，工作集快速扩张和收缩直到趋于下一个稳定值



工作集的性质

- 随Δ单调递增：

  $W(t, \Delta) \subseteq W(t,\Delta+ a)$，其中 $a>0$

- 工作集大小范围：

  $1 \le |W(t,\Delta)| \le \min(\Delta,N)$，

  其中 N 是进程的总页数

工作集策略

- 利用工作集来进行驻留集调整的策略：
  - 记录一个进程的工作集变化
  - 定期删除**驻留集**中不在工作集中的页
  - 总是让**驻留集**包含工作集（不能包含时则增大**驻留集**）

工作集策略

- 存在问题：
  - 工作集的过去变化未必能够预示工作集的将来（大小或组成页均可能会改变）
  - 记录每个进程的工作集变化所要求的开销太大
  - 对工作集窗口大小Δ的最优值难以确定，而且通常该值动态变化

工作集策略的接近策略

- 缺页率 (PFF，Page Fault Frequency) 算法
- 跟踪缺页率而不是工作集的变化！
  - 设定缺页率高/低阈值，缺页率高/低于相应阈值时，增加/减少驻留集大小
  - 主要缺点：在局部性阶段的过渡期间效果不好

工作集策略的接近策略

- VSWS (Variable-interval Sampled Working Set，可变间隔采样工作集)策略
  - 通过增加采样频率来解决PFF算法的缺点
  - 驱动参数：采样区间的最大/最小宽度M/L（为异常条件提供边界保护）、采样实例间允许发生的缺页中断数Q（使能正常激活采样）
  - 策略：
    - 采样间隔达到L时挂起进程并扫描使用位
    - 若在采样间隔<L时发生了Q次缺页中断
      - 采样间隔<M，则一直等待
      - 采样间隔≥M，则扫描使用位

清除策略

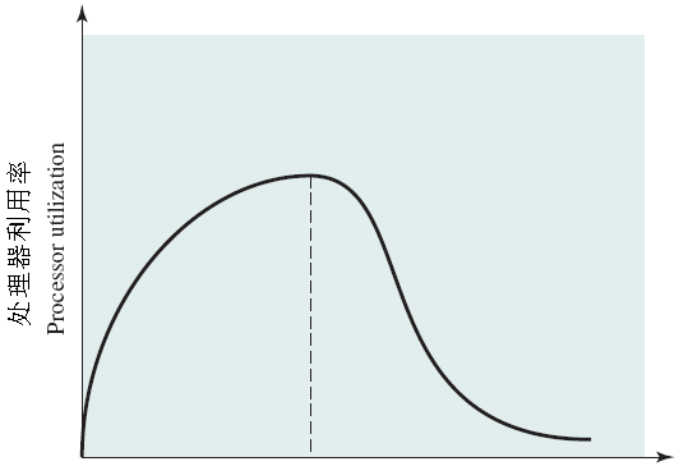
- 决定何时将已修改页调出到外存上
- 有两种常用清除策略：
  - 请求清除(demand cleaning)：该页被置换之前才调出，即把清除推迟到最后一刻
    - 调入所缺页之前还要调出已修改页，缺页进程的等待时间较长
  - 预先清除(precleaning)

清除策略

- 决定何时将已修改页调出到外存上
- 有两种常用清除策略：
  - 请求清除(demand cleaning)：
  - 预先清除(precleaning)：该页被置换之前就调出，因而可以成批调出多个页
    - 若这批调出外存的页中的多数在被置换之前还要被再次修改，则形成不必要的开销

负载控制

- 决定内存中同时驻留的进程数目（即多道程序系统的并发水平）
  - 过少，则通常所有进程可能都处于阻塞状态，从而CPU空闲时间太多
  - 过多，则每个进程的驻留集太小，因此缺页频繁发生，导致“抖动”现象



Multiprogramming Effects
多道程序设计的效果

负载控制策略

- 基于工作集策略的算法（如缺页率PFF等）
  - 它们隐含负载控制策略，只有那些驻留集足够大的进程才能运行，从而实现对负载的自动和动态控制
- “L = S 判据”策略（ P. Denning，1980）
  - 让缺页的平均间隔时间（是指真实时间而不是虚拟时间）等于对每次缺页的处理时间，研究表明这时CPU的利用率达到最大
  - “50%判据”策略：让外存交换设备保持50%利用率，这时CPU也达到最高的利用率

加载控制策略

- 基于Clock替换算法的加载控制策略
  - 定义一个轮转计数，描述轮转的速率（即扫描环形页面链的速率）
  - 当轮转计数小于一定的阈值时，表明缺页较少或存在较多不常使用的页，可提高系统负载
  - 当轮转计数大于某阈值时，表明系统的进程并发水平过高，需降低系统负载

加载控制的实施

- 当系统并发水平过高时（根据前述加载控制策略判定），需要降低系统负载
- 操作系统不能完全控制进程的创建，但可通过进程挂起（中程调度）来减少驻留内存的进程数目。
  - 即需要减少驻留内存的进程数目时，可以将部分进程挂起并全部换出到外存上。如低优先级的、缺页率高的、驻留集最小的、页最多的，等等

总结：虚存中的策略

页何时怎么读入内存?	读取策略
读进来的页放哪?	放置策略
放不进得把谁写回辅存?	置换策略
页何时怎么写回辅存?	清除策略
老进程留页问题	驻留集管理
新进程加载问题	加载控制

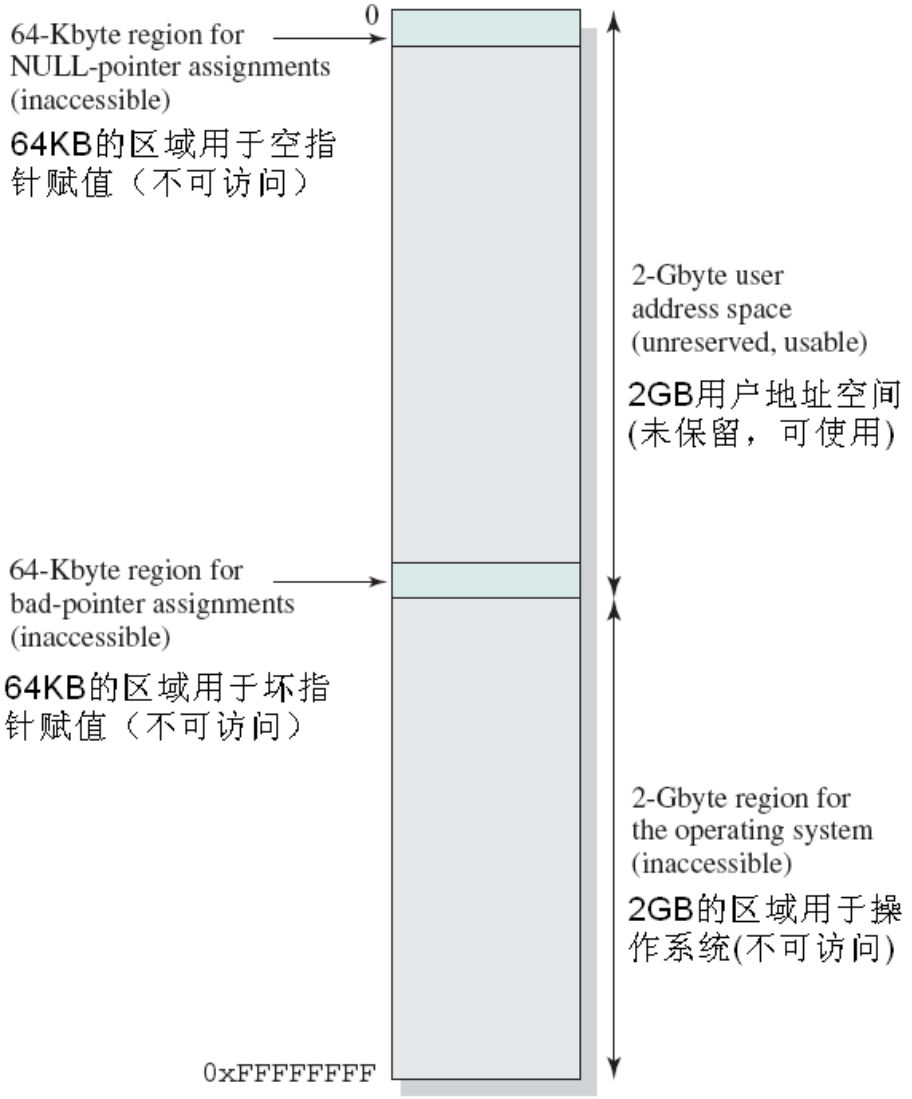
问答

Linux内存管理

- 虚拟内存
  - 虚拟存储采用三级页表
  - 页框分配采用伙伴系统
  - 页置换采用时钟算法
- 内核内存分配
  - 管理物理内存页框的分配和收回
  - 以用户虚存管理的页分配机制为基础，也使用伙伴算法
  - 对小于一页的小块内存分配，采用slab（厚片/平板）分配方案，可分配32、64、128、252、508、2040和4080

Windows存储器管理

- WinNT的存储器管理程序可以在各种平台上运转，使用的页大小可以从4KB到64KB
- 32位WinNT的虚地址空间：4GB



Windows Default 32-Bit Virtual Address Space

Windows默认32位虚拟地空间

Windows存储器管理

- WinNT进程地址空间被划分成固定大小的页，一个页的状态可以是：
  - 可用(Available/Free)：可保留或提交的空闲页面
  - 保留(Reserved)：逻辑页已分配，但物理存储没有分配的页面；可被释放或提交
  - 提交(Committed)：物理存储（在内存或磁盘上）已分配的页；可被回收（变成保留页）
- WinNT的驻留集管理方案是：可变分配+局部替换

笨重老旧的操作系统？

人工智能让操作系统焕发新光彩