第8章 虚拟内存

- 主要内容
  - 8.1 硬件和控制结构 ★★★★★
  - 8.2 操作系统软件 ★★★★★
  - 8.3 UNIX和Solaris内存管理（自学）
  - 8.4 Linux内存管理（自学）
  - 8.5 Windows内存管理（自学）

8.1 硬件和控制结构 ★★★★★

- 实存储器（实存）：内存
- 虚存储器（虚存）：磁盘

8.1.1 局部性和虚拟内存 ★★★★★

1、程序的局部性原理

- 指程序在执行过程中的一个较短时间内，所执行的指令地址或操作数地址分别局限于一定的存储区域中。又可细分时间局部性和空间局部性。
  - 时间局部性：最近访问过的程序代码和数据很快又被访问。
  - 空间局部性：某存储单元被使用之后，其相邻的存储单元也很快被使用。

2、虚拟内存

- 虚拟内存（virtual memory）：允许进程的执行不必完全在内存中，程序可以比物理内存大。
- 在许多情况下不需要将整个程序放到内存中：
  - 处理异常错误条件的代码（几乎不执行）
  - 数组、链表和表通常分配了比实际所需更多的内存
  - 程序的某些选项或特点可能很少使用
- 能够执行只有部分在内存中的程序的好处
  - 程序不再受现有的物理内存空间限制
  - 更多程序可同时执行，CPU利用率相应增加
  - 用户程序会运行的更快
- 系统抖动：处理器大部分时间用于交换块而非执行指令

虚拟存储管理实现技术

- 使用分页实现虚存
- 使用分段实现虚存
- 使用段页式实现虚存

8.1.2 分页 ★★★★★

- 分页式虚存不把作业信息(程序和数据)全部装入主存，仅装入立即使用的页面，在执行过程中访问到不在主存的页面时，产生缺页中断，再从磁盘动态地装入 。
- 怎样才能发现页面不在主存中呢?怎样处理这种情况呢?
  - 采用的办法是：扩充页表的内容,增加驻留标志位和页面辅存的地址等信息。

8.1.2 分页

存在位

修改位

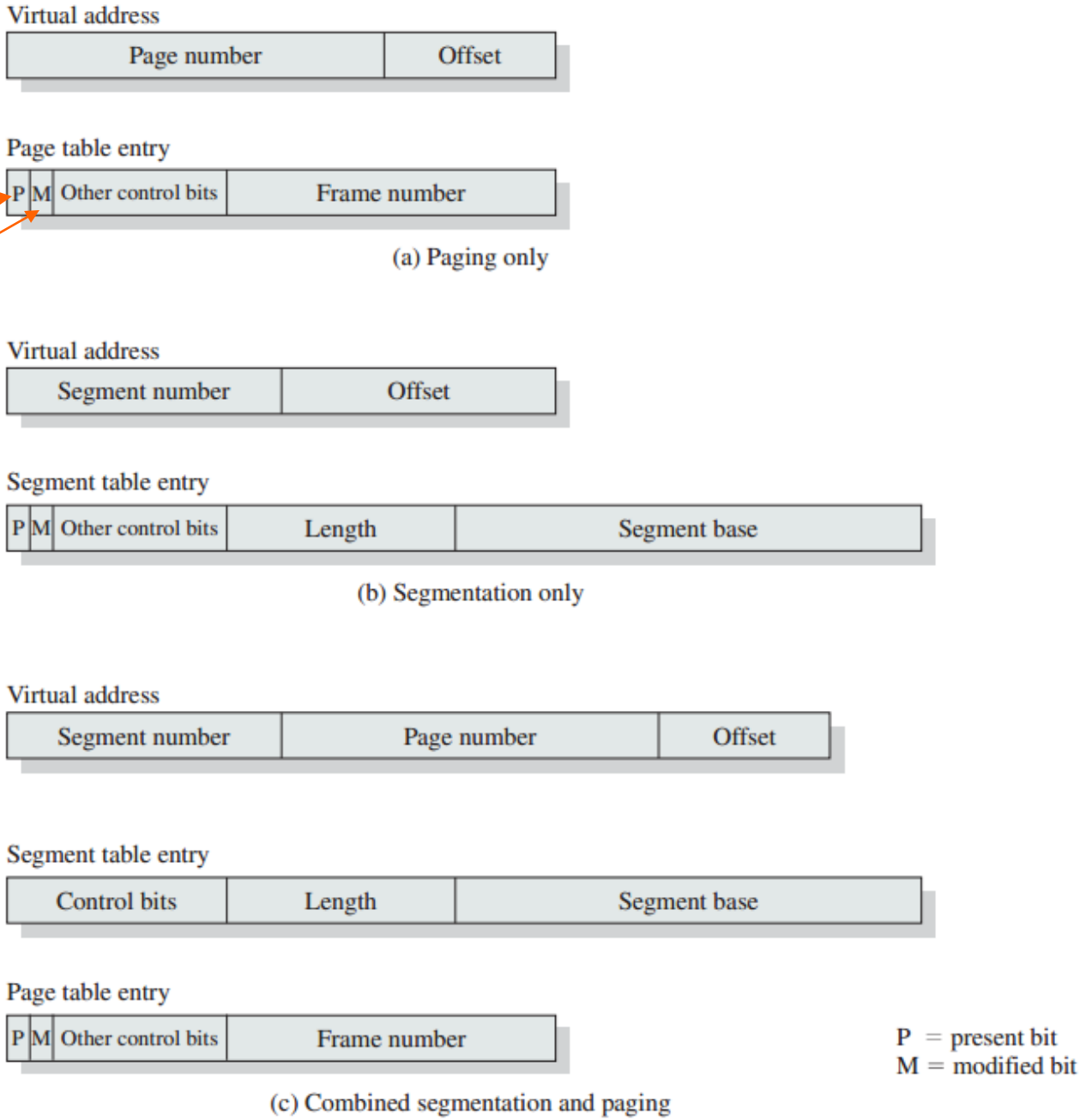


Figure 8.2 Typical Memory Management Formats

8.1.2 分页 ★★★★★

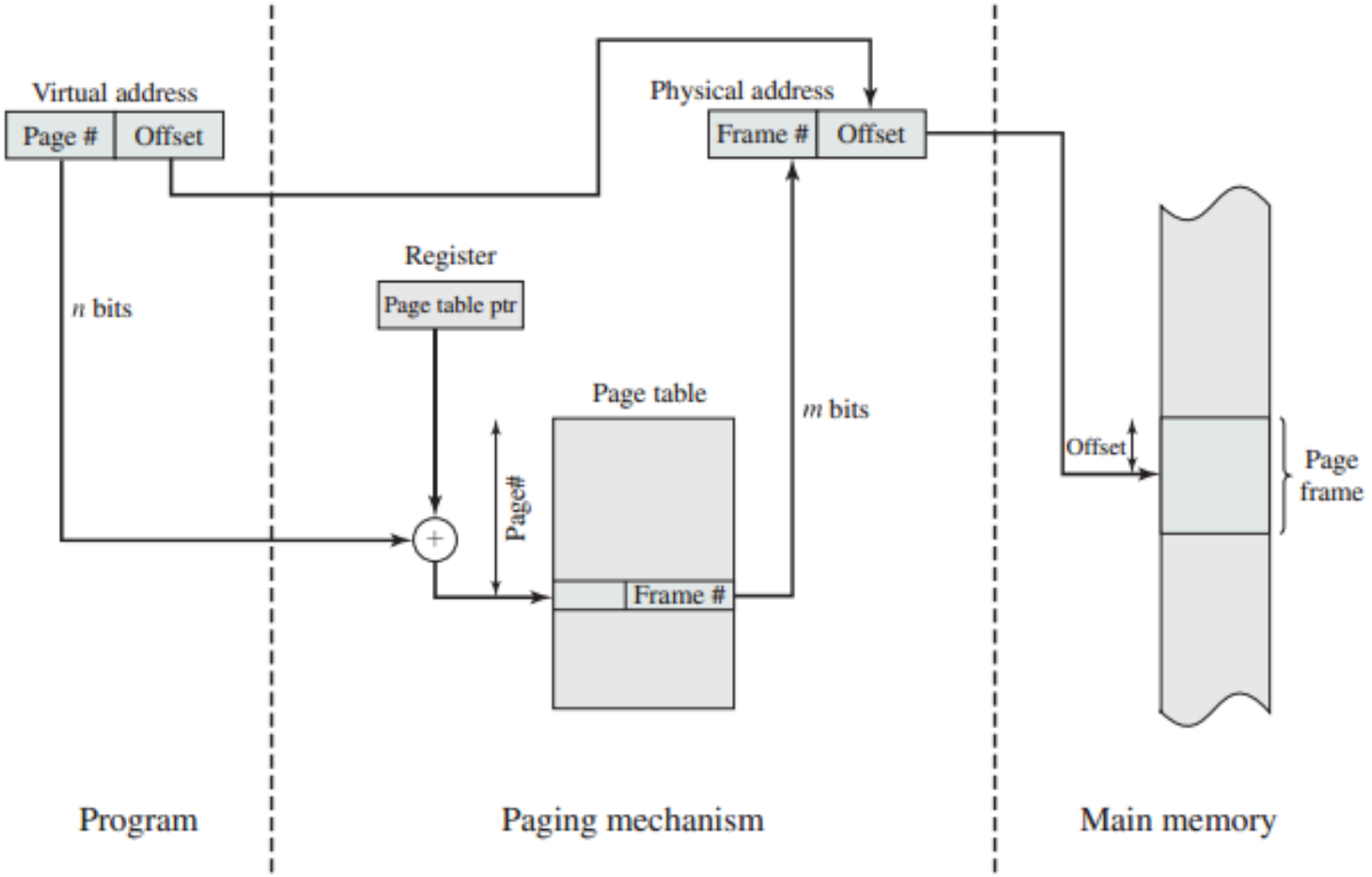


Figure 8.3 Address Translation in a Paging System

多级页表

- 系统为每个进程建一张页目录表，它的每个表项对应一个页表页，而页表页的每个表项给出了页面和页框的对应关系，页目录表是一级页表，页表页是二级页表。

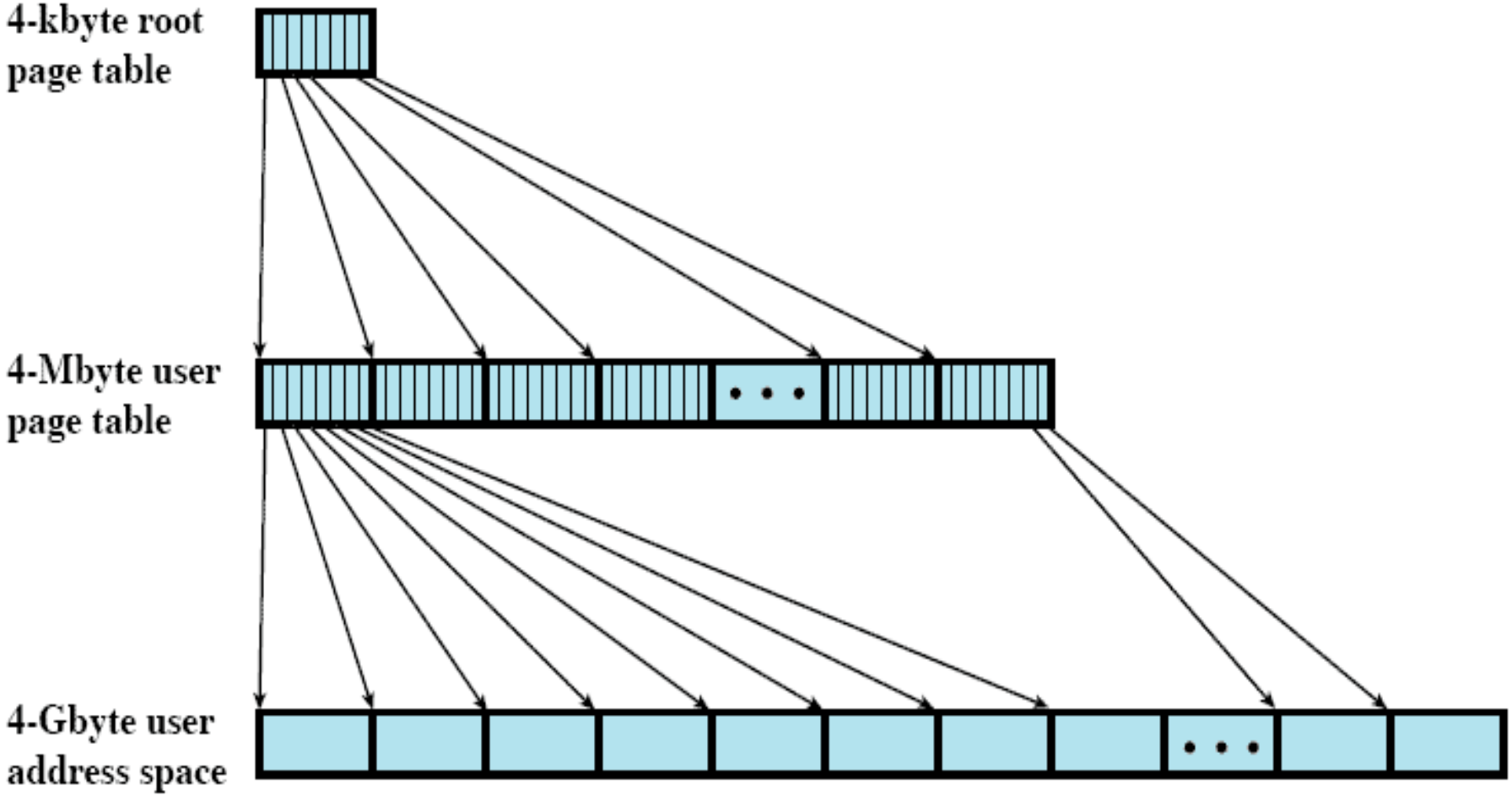


Figure 8.4 A Two-Level Hierarchical Page Table

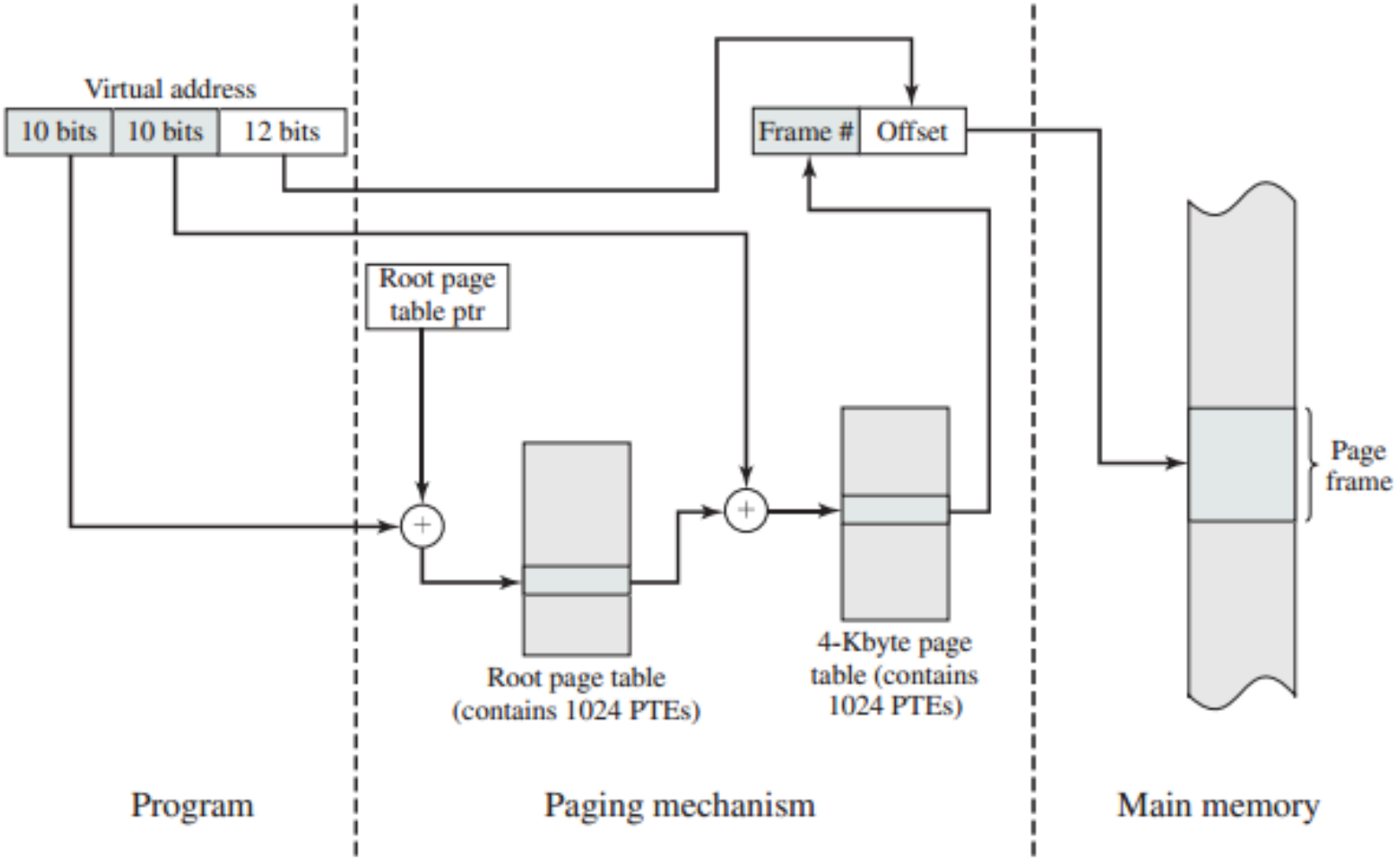


Figure 8.5 Address Translation in a Two-Level Paging System

虚拟地址 32位

页大小 4KB

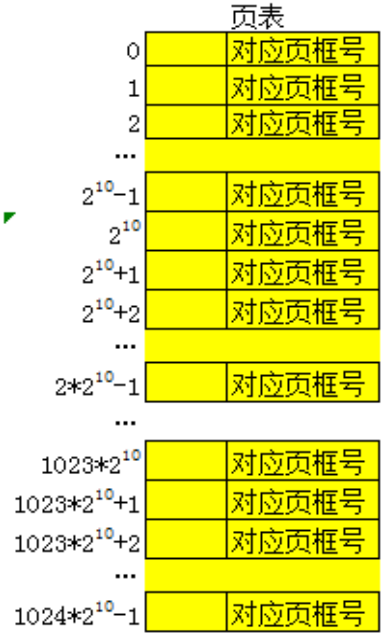
页表项大小 4B

虚拟地址空间 $2^{32}*B=4GB$ （按字节寻址）

单级分页的情况：

页表的页表项数 $4GB/4KB=2^{20}$

页表大小 $4B*2^{20}=4MB$

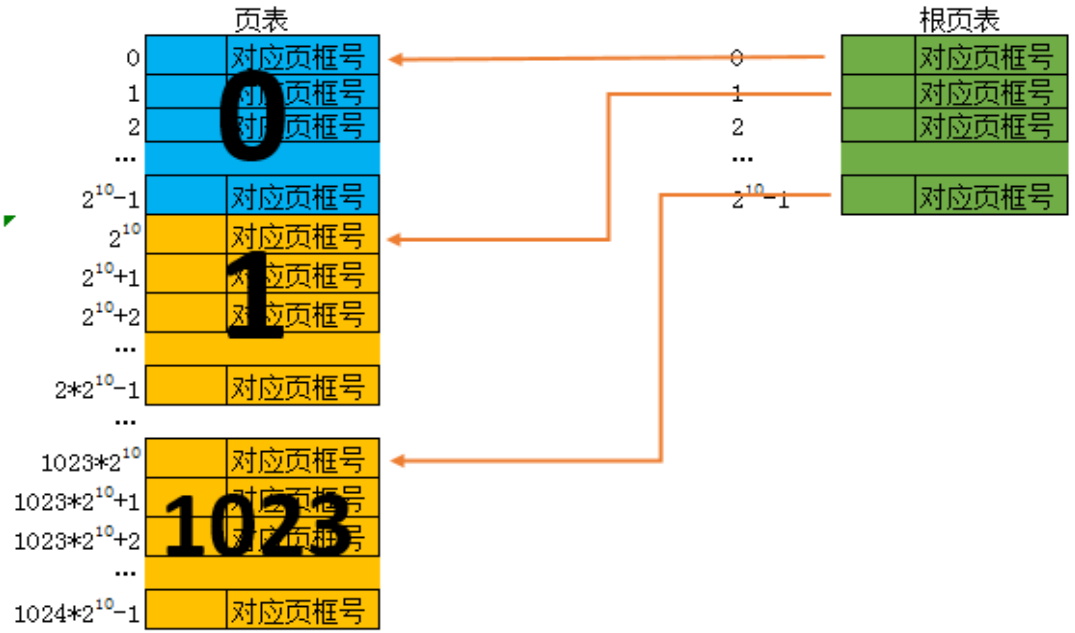


两级分页的情况：

将上述的页表放在页中，

4MB的页表需要的页数 $4MB/4KB=2^{10}$

需要一个根页表来指向构成4MB的页（1024页），该根页表需要有1024个页表项，故该根页表的大小为 $1024*4B=4KB$



倒排页表

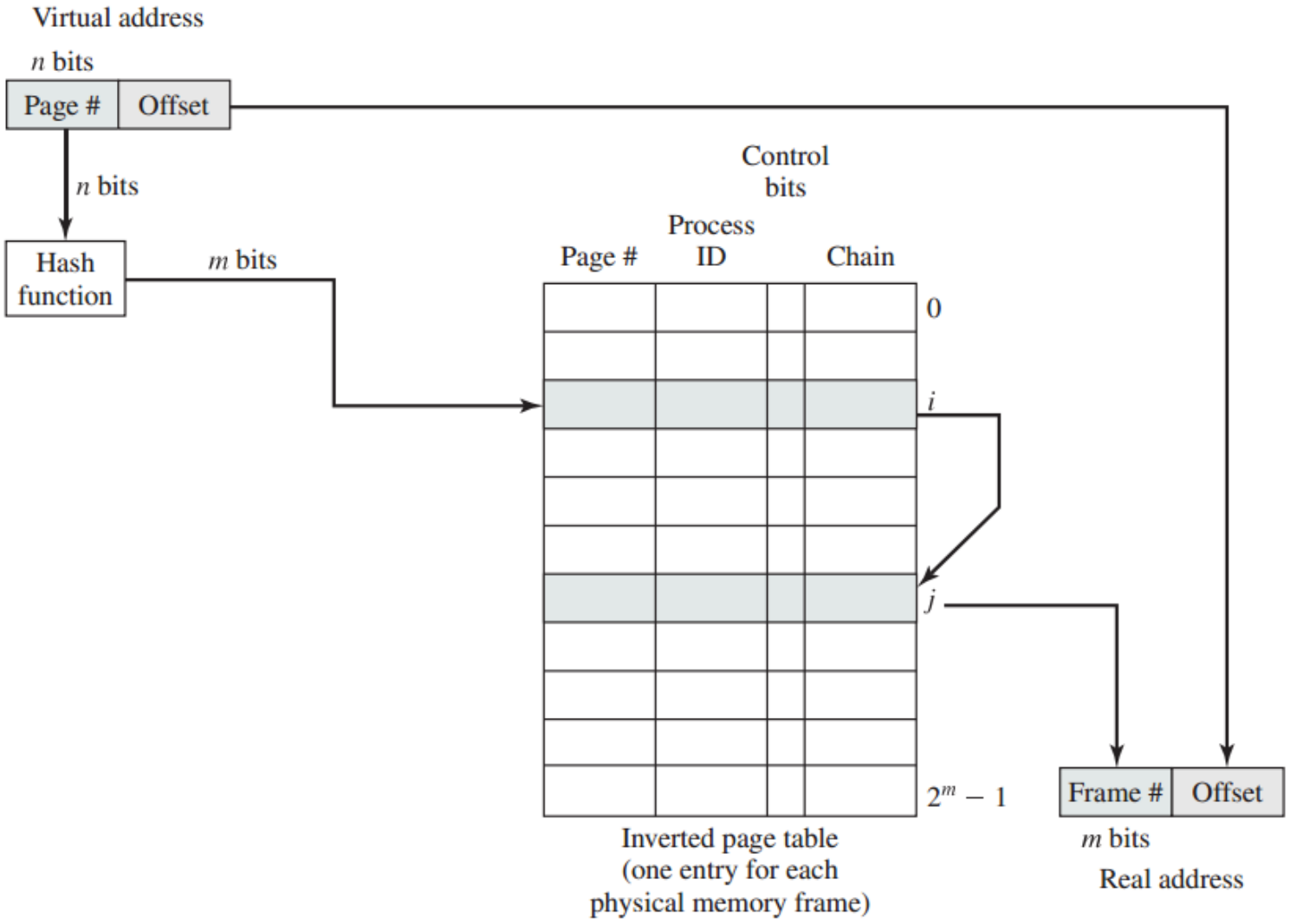


Figure 8.6 **Inverted Page Table Structure**

快表

- 转换检测缓冲区（translation look-aside buffer，TLB），也叫快表。
- TLB与页表一起工作
  - 快表项包含页号及对应的页框号，当把页号交给快表后，它通过并行匹配同时对所有快表项进行比较。
  - 如果找到，立即输出页框号，并形成物理地址；
  - 如果找不到，再查找主存中的页表以形成物理地址，同时将页号和页框号登记到快表中；
  - 当快表已满且要登记新页时，系统需要淘汰旧的快表项。

采用快表的地址转换

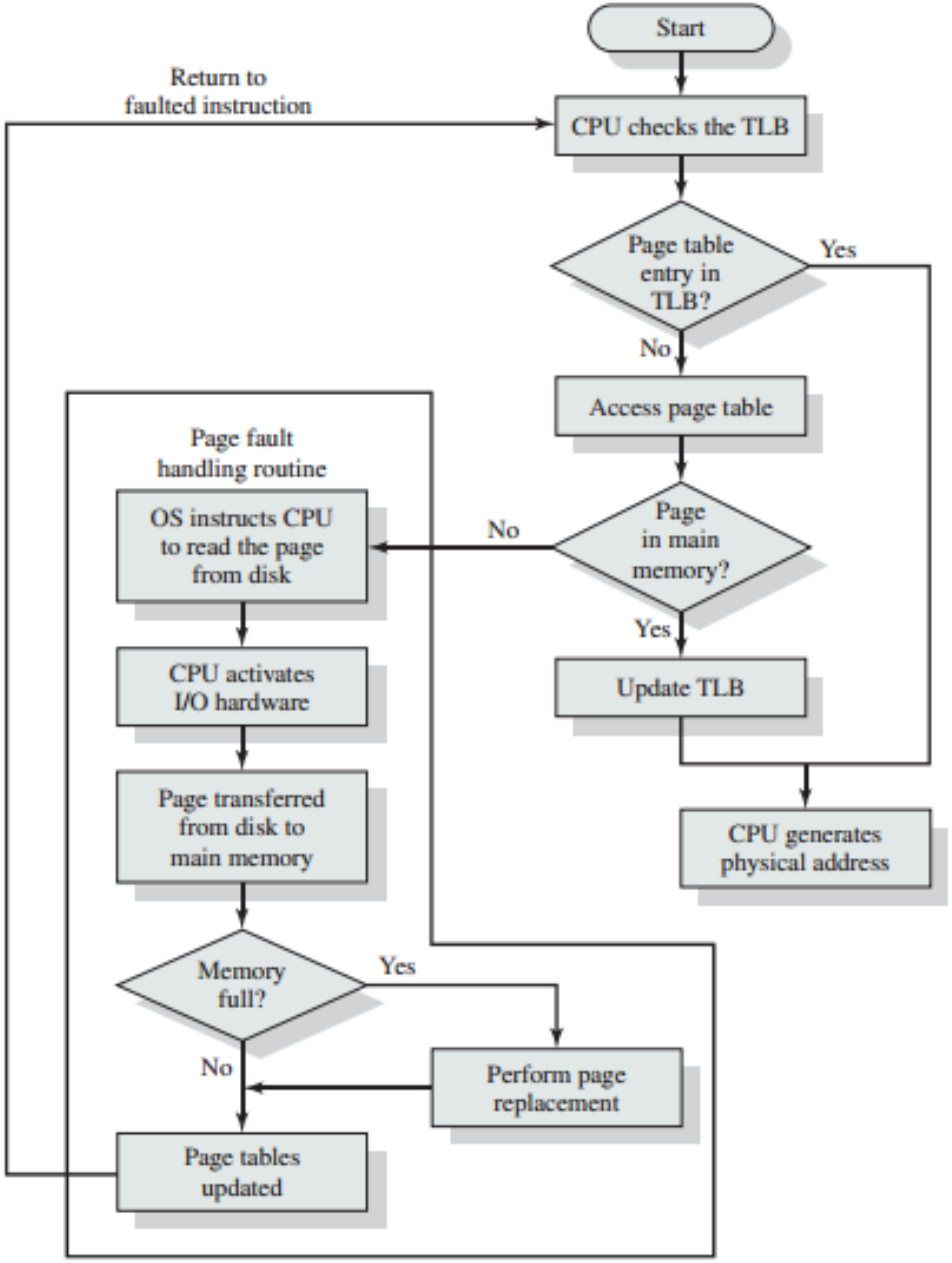


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB)

采用快表的地址转换

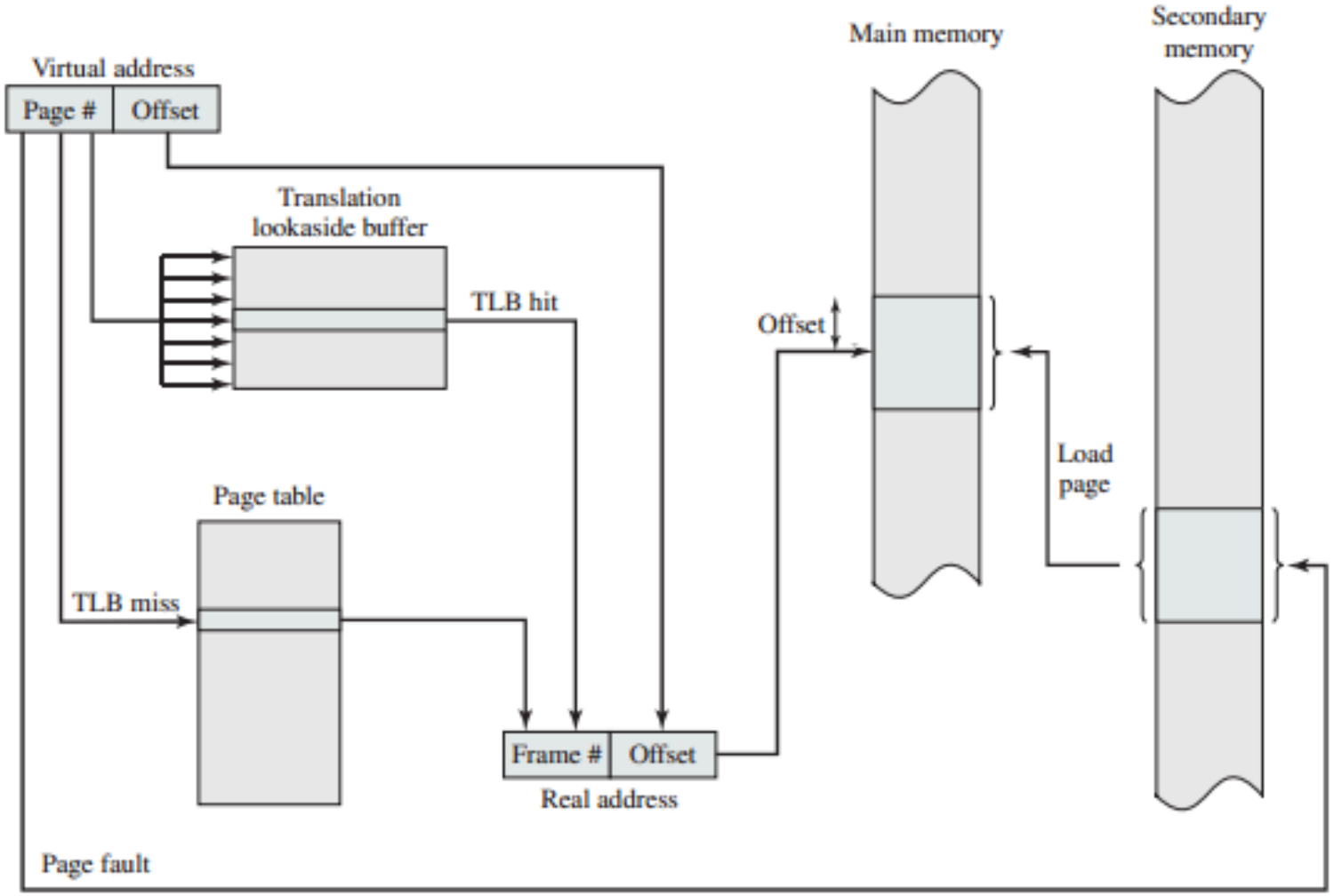


Figure 8.7 Use of a Translation Lookaside Buffer

采用快表的地址转换

- 假定访问主存时间为100毫微秒，访问快表时间为20毫微秒，相联存储器为32个单元时快表命中率可达90%，按逻辑地址存取的平均时间为：

  （100＋20）×90%＋(100+100+20)×(1−90%)＝130

- 比两次访问主存的时间100×2＝200下降了三成多。

页尺寸

- 内存利用（减少内部碎片）—小页
- 页表大小—大页
- 数据传送的效率—大页
- 降低缺页率—大页
- 分析：没有最佳答案，趋向于大页

8.1.3 分段 ★★★★★

- 分段式虚拟存储系统把作业的所有分段的副本都存放在辅助存储器中，当作业被调度投入运行时，首先把当前需要的一段或几段装入主存，在执行过程中访问到不在主存的段时再把它们装入。
- 段表也需要增加一些控制位。

8.1.4 段页式 ★★★★★

- 分页对程序员是透明的，消除了外部碎片，可以更有效地使用内存。此外，移入或移出的块是大小相等的，易于处理。
- 分段对程序员是可见的，具有处理不断增长的数据结构的能力以及支持共享和保护的能力。
- 把两者结合起来就是段页式存储管理。
  - 内存划分成大小小等的页框。
  - 用户的地址空间被程序员划分成许多段，每个段一次划分成许多固定大小的页，页的长度等于内存中的页框大小。

8.2 操作系统软件★★★★★

8.2.1 读取策略★★★★★

- 读取策略确定一个页何时取入内存
  - 请求分页
  - 预先分页

8.2.2 放置策略 ★★★★★

- 放置策略决定一个进程块驻留在内存中什么地方

8.2.3 置换策略★★★★★

- 当内存中所有页框都被占据，并且需要读取一个新页以处理一次缺页中断时，置换策略决定当前在内存中的哪个页将被置换。
- 目标：移出最近最不可能访问的页。
- 方法：基于过去的行为预测将来的行为。
- 基本算法
  - 最佳OPT
  - 最近最少使用LRU
  - 先进先出FIFO
  - 时钟Clock

1）最佳OPT

- 选择置换下次访问距当前时间最长的那些页。
  - 该算法能导致最少的缺页中断，但它要求操作系统必须知道将来的事件，所以不可能实现，可作为一种标准来衡量其它算法的性能。

OPT例

- （可用页框数量3）使用串如下：
  - 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

使用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
帧1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
帧2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
帧3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
置换	F	F	F	R		R		R			R			R				R		

缺页率 *p=9/20=45%*

2) 先进先出FIFO

- 置换驻留在内存中时间最长的页
  - 思想：一个很久以前取入内存的页，到现在可能已经不会再用到了。
  - 问题：经常出现一部分程序或数据在整个程序的生命周期中使用频率都很高的情况。

FIFO例

- 容易理解和实现，性能并不总是很好
- 例（可用页框数量3）使用串如下：
  - 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

使用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
帧 1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
帧 2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
帧 3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
置换	F	F	F	R		R	R	R	R	R	R			R	R			R	R	R

缺页率 *p=15/20=75%*

- Belady异常：页错误率随着所分配的帧数增加而增加的现象，例：

	1	2	3	4	1	2	5	1	2	3	4	5
帧1	1	1	1	4	4	4	5	5	5	5	5	5
帧2		2	2	2	1	1	1	1	1	3	3	3
帧3			3	3	3	2	2	2	2	2	4	4
缺页:	F	F	F	R	R	R	R			R	R	

可用帧数3，
9次缺页

	1	2	3	4	1	2	5	1	2	3	4	5
帧1	1	1	1	1	1	1	5	5	5	5	4	4
帧2		2	2	2	2	2	2	1	1	1	1	5
帧3			3	3	3	3	3	3	2	2	2	2
帧4				4	4	4	4	4	4	3	3	3
缺页:	F	F	F	F			R	R	R	R	R	R

可用帧数4，
10次缺页

3) 最近最少使LRU

- 置换内存中上次使用距当前最远的页。
  - 根据程序局部性原理，在较长时间里未被使用的页面，可能不会马上使用到。
- 性能接近于OPT，但比较难以实现。
  - 为每一页添加一个最后一次访问的时间戳，并且必须在每次访问内存时，都要更新时间戳。
  - 维护一个关于访问页的栈。

➢开销很大。

LRU例

- 思想：置换过去最长时间没有使用的页
- 例（可用页框数量3）使用串如下：
  - 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

使用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
帧1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
帧2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
帧3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
置换	F	F	F	R		R		R	R	R	R			R		R		R		

缺页率 *p=12/20=60%*

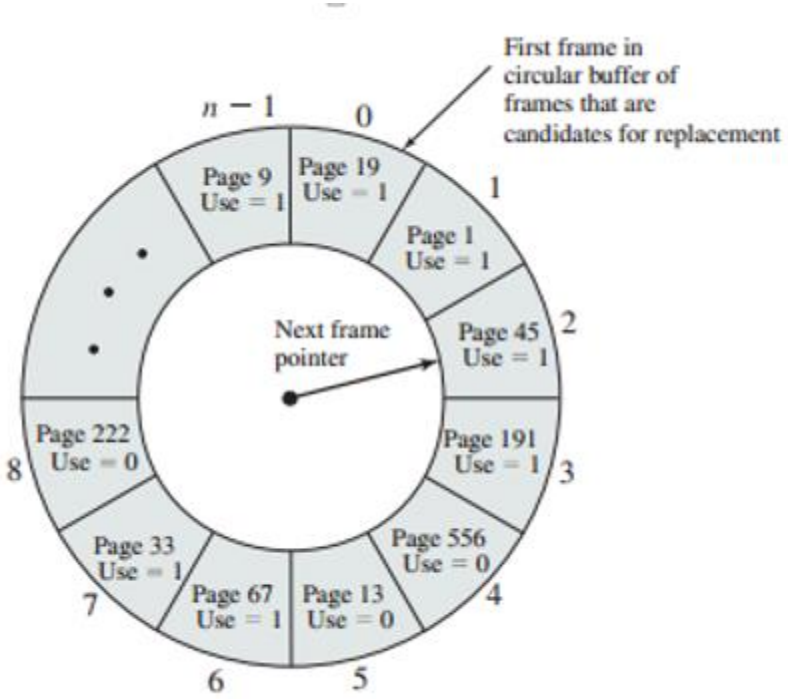
4）时钟Clock

- 以较小的开销接近LRU的性能。

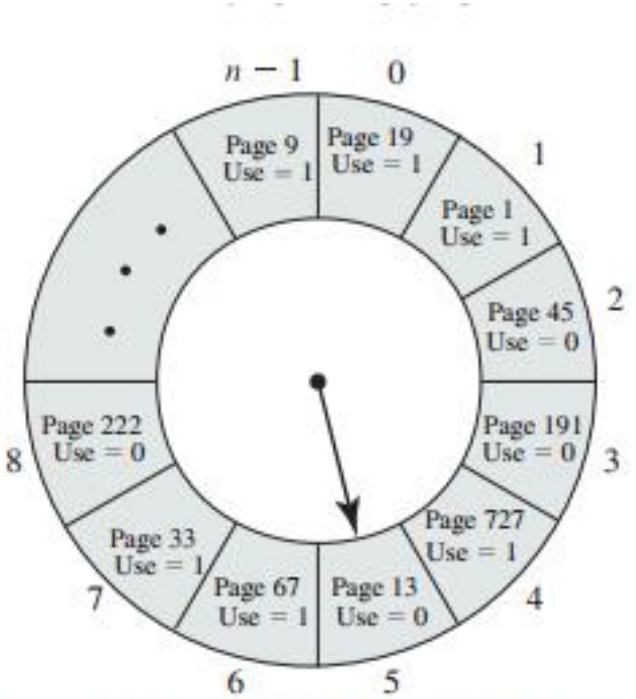
a）简单时钟策略

- 一个页面首次装入内存，其“使用位”置1 。
- 主存中的任何页面被访问时，“使用位”置1。
- 当需要置换一页时，扫描以查找“使用位”为0的页框。查找过程中，把遇到的“使用位”是1的页面的“使用位”清0，跳过这个页面，继续查找。
- 扫描时，如果遇到的所有页面的“使用位”为1，指针就会绕整个循环队列一圈，把碰到的所有页面的“使用位”清0；指针停在起始位置，并置换掉这一页。

a)简单时钟策略

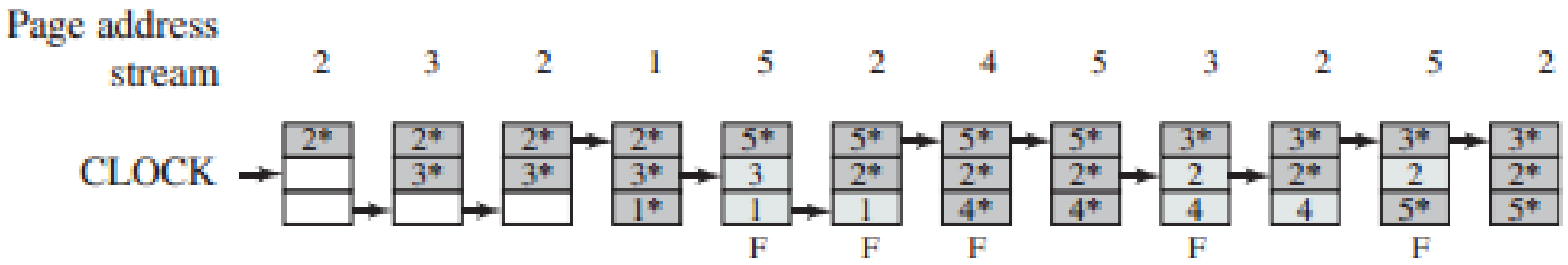


(a) State of buffer just prior to a page replacement



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

a)简单时钟策略



b)时钟策略改进算法

- 把”使用位”和”修改位”结合起来使用，共组合成四种情况：

(1)最近未被使用，也未被修改(u=0，m=0)

(2)最近被使用，没有被修改(u=1，m=0)

(3)最近未被使用，但被修改(u=0，m=1)

(4)最近被使用，也被修改(u=1，m=1)

算法执行过程

1. 从指针当前位置开始，扫描页框缓冲区。扫描过程中不改变”使用位”，把遇到的第一个u=0，m=0的页面作为置换页面。
2. 如果步1失败，则重新扫描，查找u=0，m=1的页面，把遇到的第一个这样的页面作为置换页面，而在扫描过程中把指针所扫过的页面的”使用位”u置0。
3. 如果步2失败，指针再次回到了起始位置，由于此时所有页面的”使用位”u均已为0，再转向步1操作，必要时再做步2操作，这次一定可以挑出一个可置换的页面。

✓页置换算法练习

1. 已知页面走向为1、2、1、3、1、2、4、2、1、3、4，且开始执行时内存中没有页面。若只给该作业分配2个物理块。

1）当采用FIFO、OPT、LRU页置换算法时缺页率为多少？

2）假定现有一种置换算法，该算法淘汰页面的策略为当需要淘汰页面时，就把刚使用过的页面作为淘汰对象，试问就相同的页面走向，其缺页率又为多少？

3）假如分配3个物理块呢？

➢ 2个物理块

1	2	1	3	1	2	4	2	1	3	4	
1	1		3	3	2	2		1	1	4	
	2		2	1	1	4		4	3	3	FIFO 9/11=81.8%
1	1		1		1	4		4	4		
	2		3		2	2		1	3		OPT 7/11=63.6%
1	1		1		1	4		1	1	4	
	2		3		2	2		2	3	3	LRU 8/11=72.7%
1	1		3	1		1	1		3	4	
	2		2	2		4	2		2	2	NEW 8/11=72.7%

➢ 3个物理块

1	2	1	3	1	2	4	2	1	3	4
1	1		1			4		4		
	2		2			2		1		
			3			3		3		
1	1		1			1			3	
	2		2			2			2	
			3			4			4	
1	1		1			1			1	1
	2		2			2			2	4
			3			4			3	3
1	1		1			1	1			1
	2		2			4	2			2
			3			3	3			4

FIFO 5/11=45.5%

OPT 5/11=45.5%

LRU 6/11=54.5%

NEW 6/11=54.5%

8.2.4 驻留集管理 ★★★★★

驻留集大小

- 给特定进程分配多大内存空间，需考虑因素：
  - 分配给一个进程的内存越少，在任何时候驻留在内存中的进程数就越多，从而增加了操作系统至少找到一个就绪进程的可能性，从而减少了由于交换而消耗的处理器时间。
  - 如果一个进程在内存中的页数比较少，尽管有局部性原理，缺页率仍然相对较高。
  - 给特定进程分配的内存空间超过一定的大小后，由于局部性原理，该进程的缺页率没有明显的变化

分配策略

- 固定分配策略
  - 为一个进程在内存中分配固定数目的页框用于执行时使用。
- 可变分配策略
  - 允许分配给一个进程的页框在该进程的生命周期中不断地发生变化。
  - 缺页率高，可增加分配的页框；
  - 缺页率很低，可适当减少分配的页框。

置换范围

- 局部置换策略
  - 仅在产生这次缺页的进程的驻留页中选择
- 全局置换策略
  - 把内存中所有未被锁定的页都作为置换的候选页，不管它们属于哪一个进程。

驻留集管理

	局部置换	全局置换
固定分配	• 分配给一个进程的页框数是固定的 • 从分配给该进程的页框中选择被置换的页	• 无此方案
可变分配	• 分配给一个进程的页框数可以变化 • 从分配给该进程页框中选择被置换的页	• 从内存中所有可用页框中选择被置换的页，这导致进程驻留集大小不断变化

工作集策略

- 进程工作集指“在某一段时间间隔内进程运行所需访问的页面集合”。
- W(t, △)
  - t是虚拟时间
  - △是观察进程的虚拟时间窗口
  - 工作集大小是关于窗口大小的一个非减函数
    - ☒$W(t, \Delta+1) \supseteq W(t, \Delta)$
  - 虚拟时间窗口越大，工作集就越大

Sequence of Page References	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

图8.19 由窗口大小所定义的进程工作集

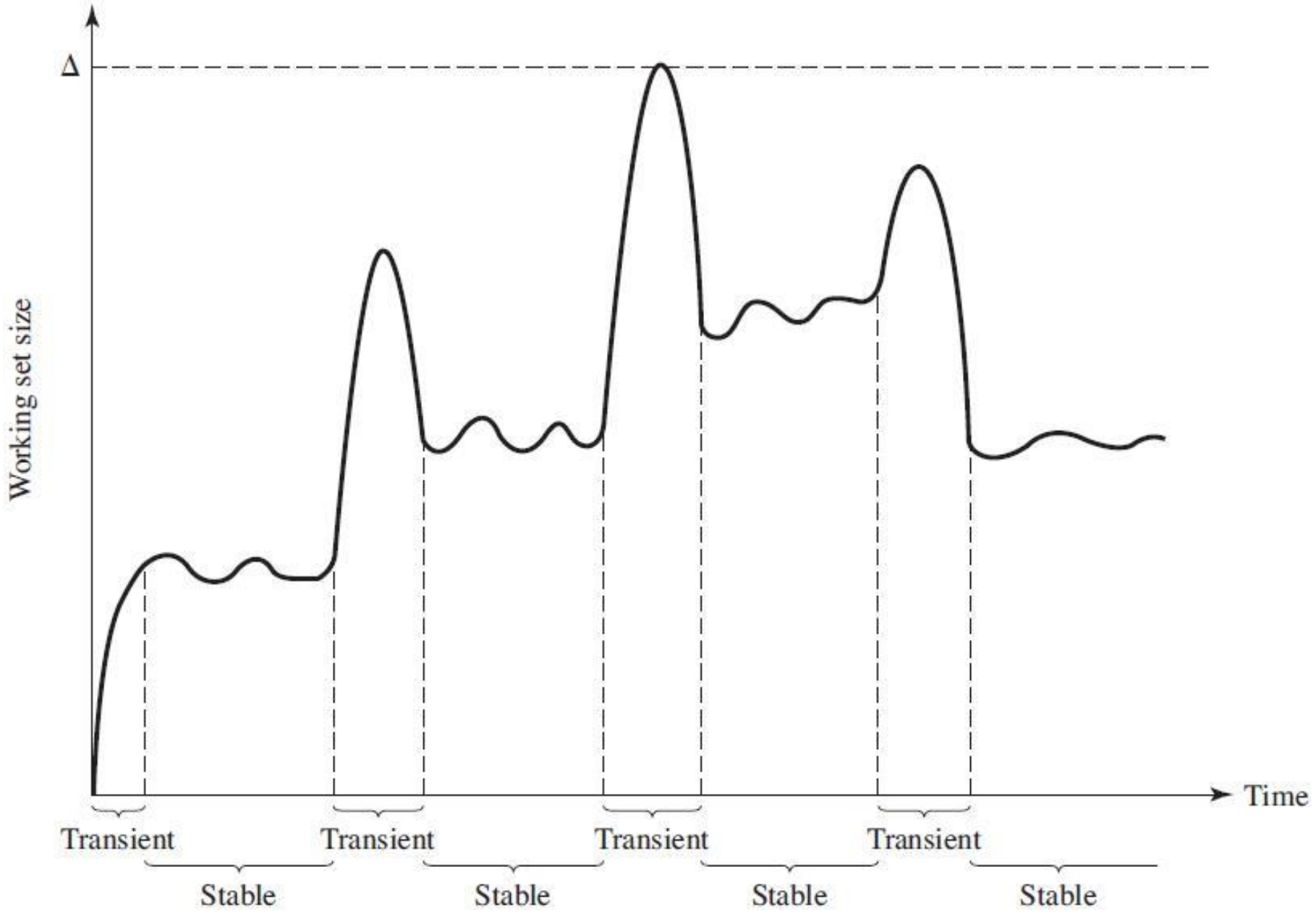


图8.20 关于工作集大小的一个典型示意图[MAEK87]

工作集策略的优缺点

- 优点：可用于指导驻留集大小
  - 监视每个进程的工作集
  - 周期性地从一个进程的驻留集中移去那些不在它的工作集中的页
  - 只有当一个进程的工作集在内存中时，才可以执行
- 缺点
  - 根据过去预测将来的不准确性
  - 为每个进程真实地测量工作集是不实际的。
  - △的最优值是未知的，并且它在任何情况下都会变化

工作集策略的近似算法-缺页中断频率PFF

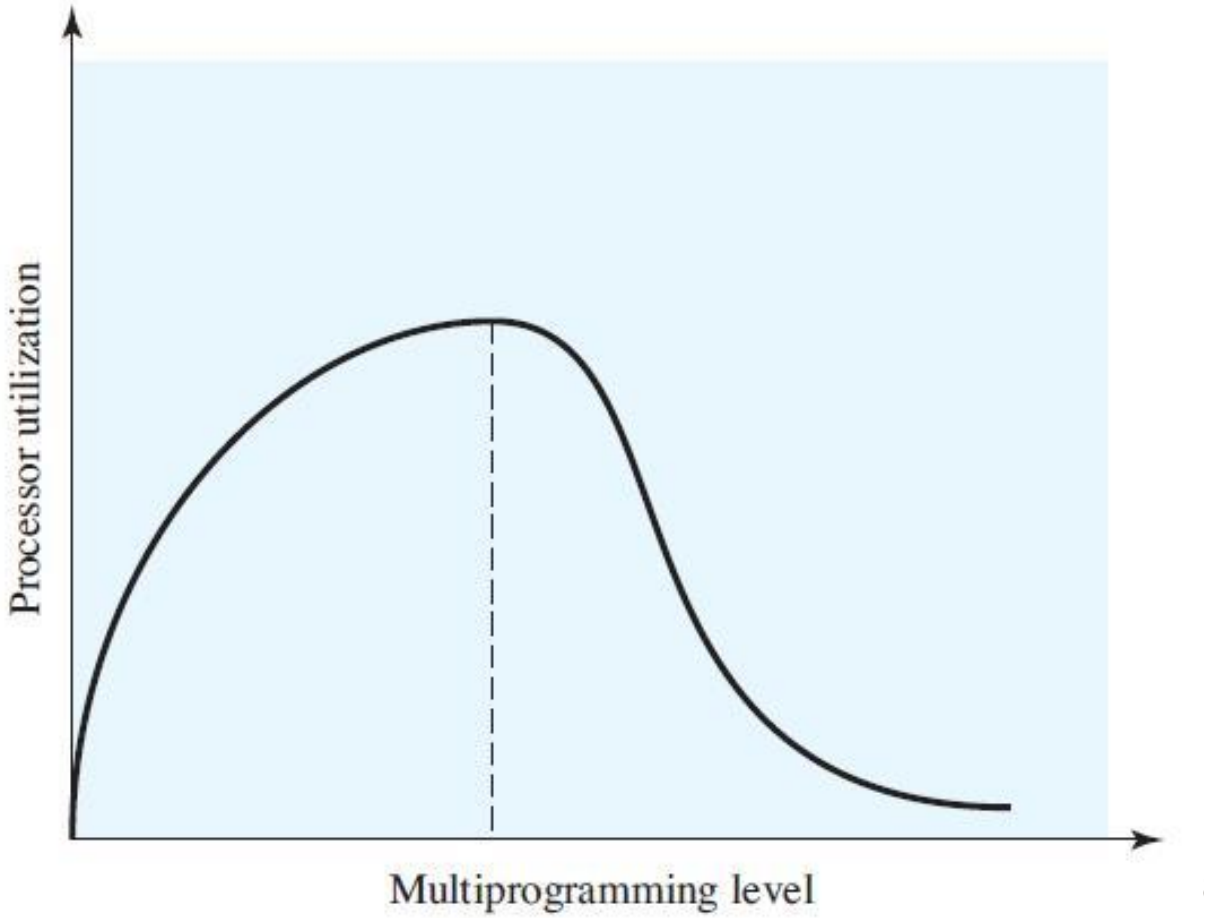
- 根据连续的缺页之间的时间间隔来对缺页频率进行测量，每次缺页时，利用测量时间调整进程工作集尺寸。
- 规则：如果本次缺页与前次缺页之间的时间超过临界值τ，那么，所有在这个时间间隔内没有使用的页面都被移出工作集。

8.2.5 清除策略 ★★★★★

- 确定在何时将一个被修改过的页写回辅存。
- 清除策略
  - 请求式清除：只有当一页被选择用于置换时才被写回
  - 预约式清除：被修改的多个页在需要用到它们所占据的页框之前成批地写回辅存

8.2.6 加载控制 ★★★★★

- 加载控制会影响到驻留在内存中的进程数目，即系统并发度。
- 系统并发度



进程挂起

- 如果系统并发度被减小，一个或多个当前驻留进程必须被挂起。
  - 最低优先级进程
  - 缺页中断进程
  - 最后一个被激活的进程
  - 驻留集最小的进程
  - 最大空间的进程
  - 具有最大剩余执行窗口的进程

8.3-8.5 简述

- UNIX
  - 早期版本：可变分区
  - 现代版本：分页式虚存管理+懒惰伙伴系统（内核）
- Linux
  - 页式虚存管理（三级页表）
  - slab分配方案（类似于伙伴系统，内核）
- Windows
  - 页式虚存管理

作业

- 复习题 8.2， 8.5， 8.11
- 习题 8.1， 8.4， 8.11