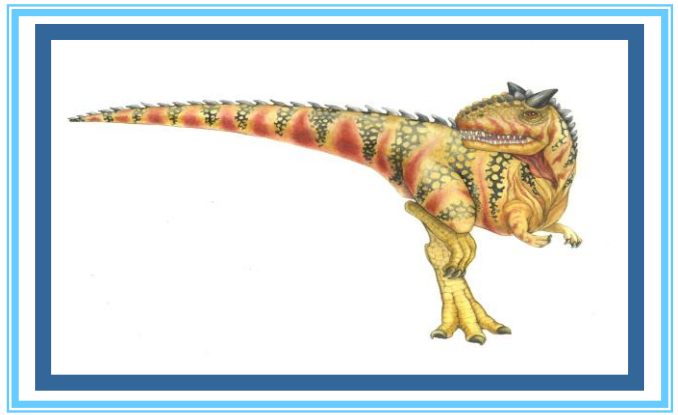
第五章 虚拟存储器



主要内容

- 虚拟存储器概述
- 请求分页存储管理方式
- 页面置换算法

5.1 虚拟存储器概述

- 虚拟存储器的引入
- 虚拟存储器的基本概念

虚拟存储器的引入

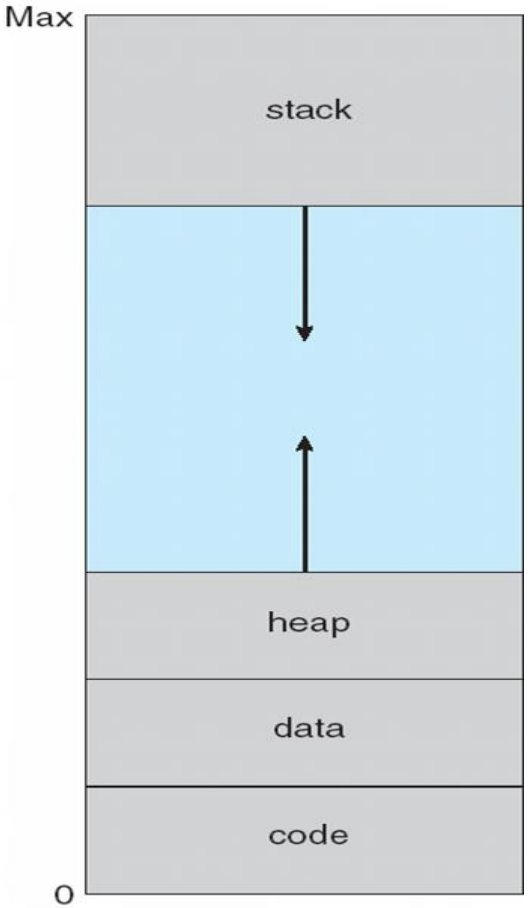
- 离散分配存储管理
  - 一次性
    - 将一个作业全部装入内存
  - 驻留性
    - 作业一旦装入内存，便一直驻留内存直到运行结束
- 问题
  - 很多作业运行时，并非用到了全部，所以造成内存空间的浪费
  - 作业很大，所要求的内存容量超过了内存总量，无法运行

虚拟存储器的引入

- 许多情况下，并不需要将整个程序装入内存
  - 异常处理的代码
  - 某些不常被调用的函数
- 程序部分装入内存的好处
  - 程序不受物理内存空间的限制
  - 更多的程序并发执行

虚拟存储器的引入

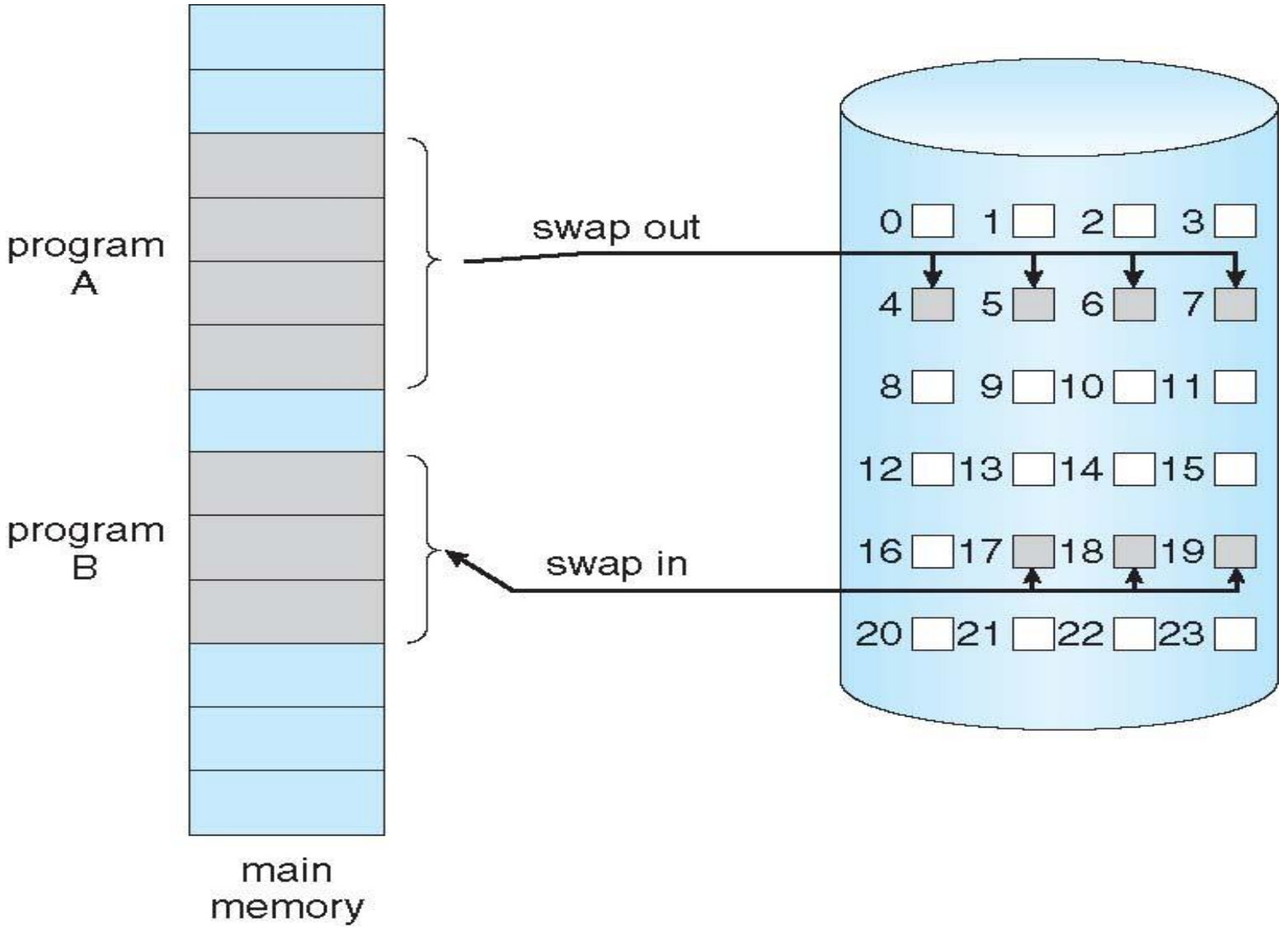
- “局部性”原理
  - 是指程序在执行的某一时刻，并不是均匀地访问它的整个地址空间，而往往是集中于某一小部分区域
    - 除分支和调用指令，程序的执行都是顺序的
    - 大多数循环结构都由较少的几条指令重复若干次



虚拟存储器的基本概念

- 虚拟存储的思想
  - 是一种扩大内存容量的软件设计技术
  - 操作系统把当前需要使用的部分程序、数据等内容读入内存，其他部分保存在磁盘上，必要时由操作系统实施内存和磁盘之间的交换
- 特点
  - 部分装入
    - 程序可以比内存大
    - 进程不必完全装入内存，避免浪费
  - 交换

交换


program
A
program
B
main
memory
swap out
swap in
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

虚拟存储器的实现方式

- 请求分页存储管理
- 请求分段存储管理
- 请求段页式存储管理

5.2 请求分页存储管理方式

- 请求分页的页表机制
- 缺页中断
- 页面置换
- 请求分页系统中的地址变换

请求分页存储管理

- 基本思想
  - 程序部分装入内存
  - 在程序执行过程中，发生缺页，则将相应的页调入到内存，继续执行
  - 如果内存已满，发生置换
- 请求分页系统
  - 在分页系统的基础上，增加了请求调页功能、页面置换功能

请求分页存储管理

- 特点
  - 只装入若干页的程序或数据
  - 之后再通过请求调页功能和页面置换功能，把将要运行的页面调入内存，同时，把暂时不运行的页面换出到外存
  - 置换时以页面为单位

请求分页存储管理

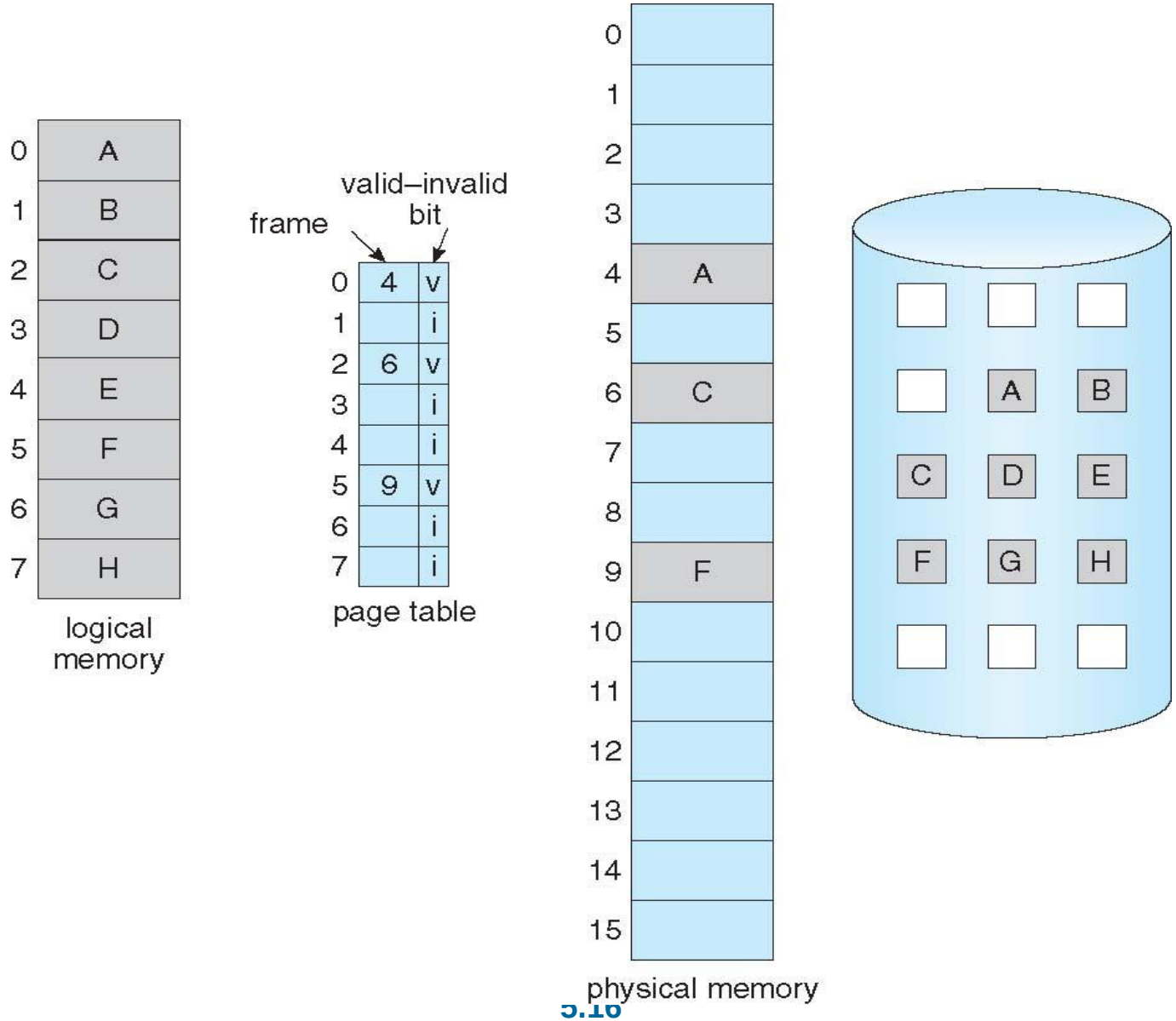
- 硬件支持
  - 请求分页的页表机制
  - 缺页中断机构
  - 地址变换机构

请求分页的页表机制

- 扩充页表
  - 判断访问的页面是否在内存
  - 有效-无效位（valid-invalid bit）
    - 1：表示该页在内存
    - 0：表示该页不在内存

页号	页框号	有效-无效位
0		1
1		1
2		0
…		
9		1

请求分页的页表机制


0 A
1 B
2 C
3 D
4 E
5 F
6 G
7 H
logical memory
valid–invalid bit
frame
0 4 v
1 i
2 6 v
3 i
4 i
5 9 v
6 i
7 i
page table
0
1
2
3
4 A
5
6 C
7
8
9 F
10
11
12
13
14
15
physical memory
A B
C D E
F G H

请求分页的页表机制

- 请求页式的页表项结构

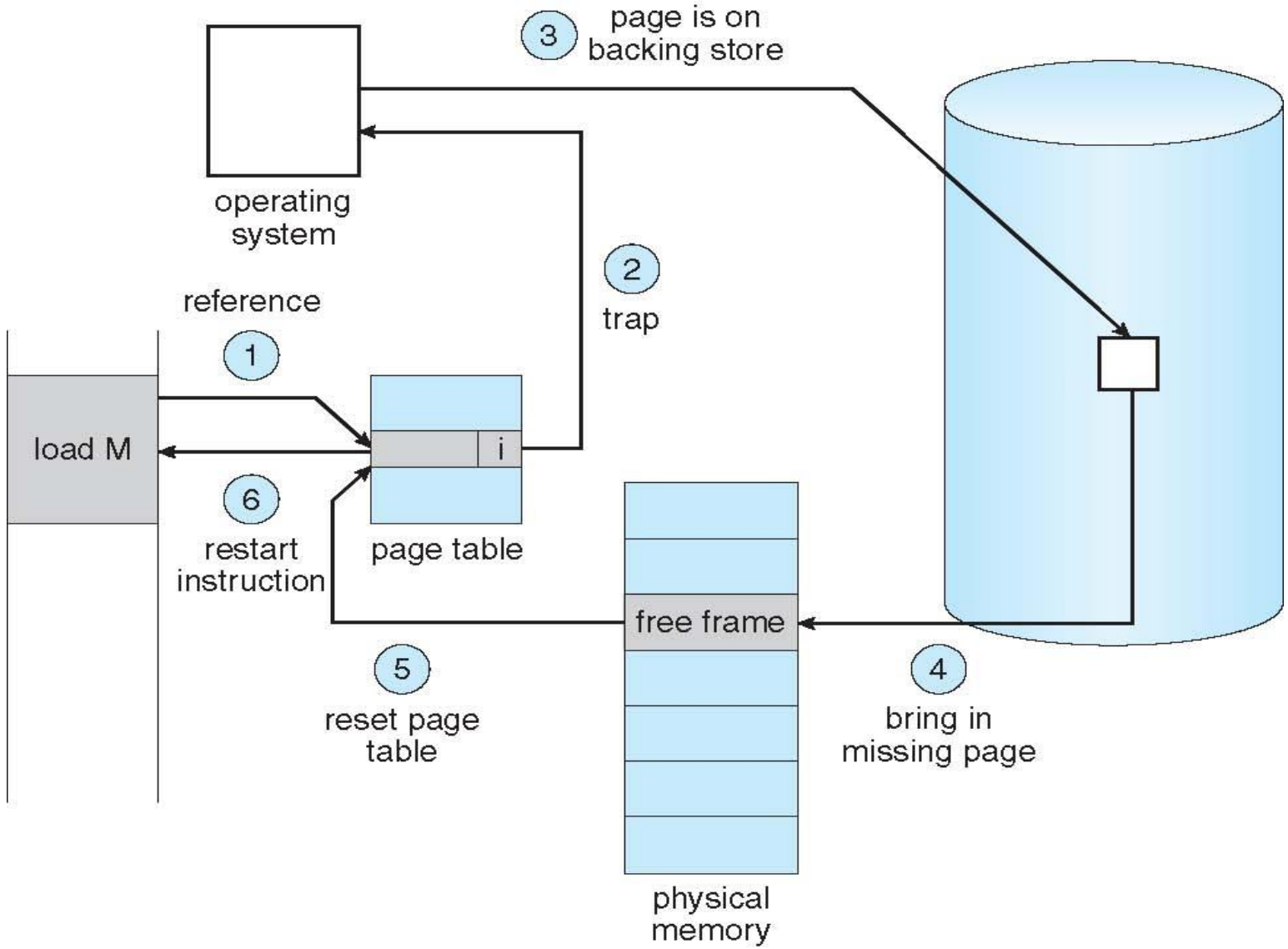
页号	物理块号	状态位P	访问字段A	修改位M	外存地址

  - 状态位/有效位：用于指示该页是否已调入内存
  - 访问字段：记录页面的访问情况
    - 记录本页在一段时间内被访问的次数
    - 最近已有多长时间未被访问
  - 修改位：表示该页在调入内存后是否被修改过
  - 外存地址

缺页中断

- 当要访问的页面不在内存时，要产生一次缺页中断，请求OS将所缺之页调入内存

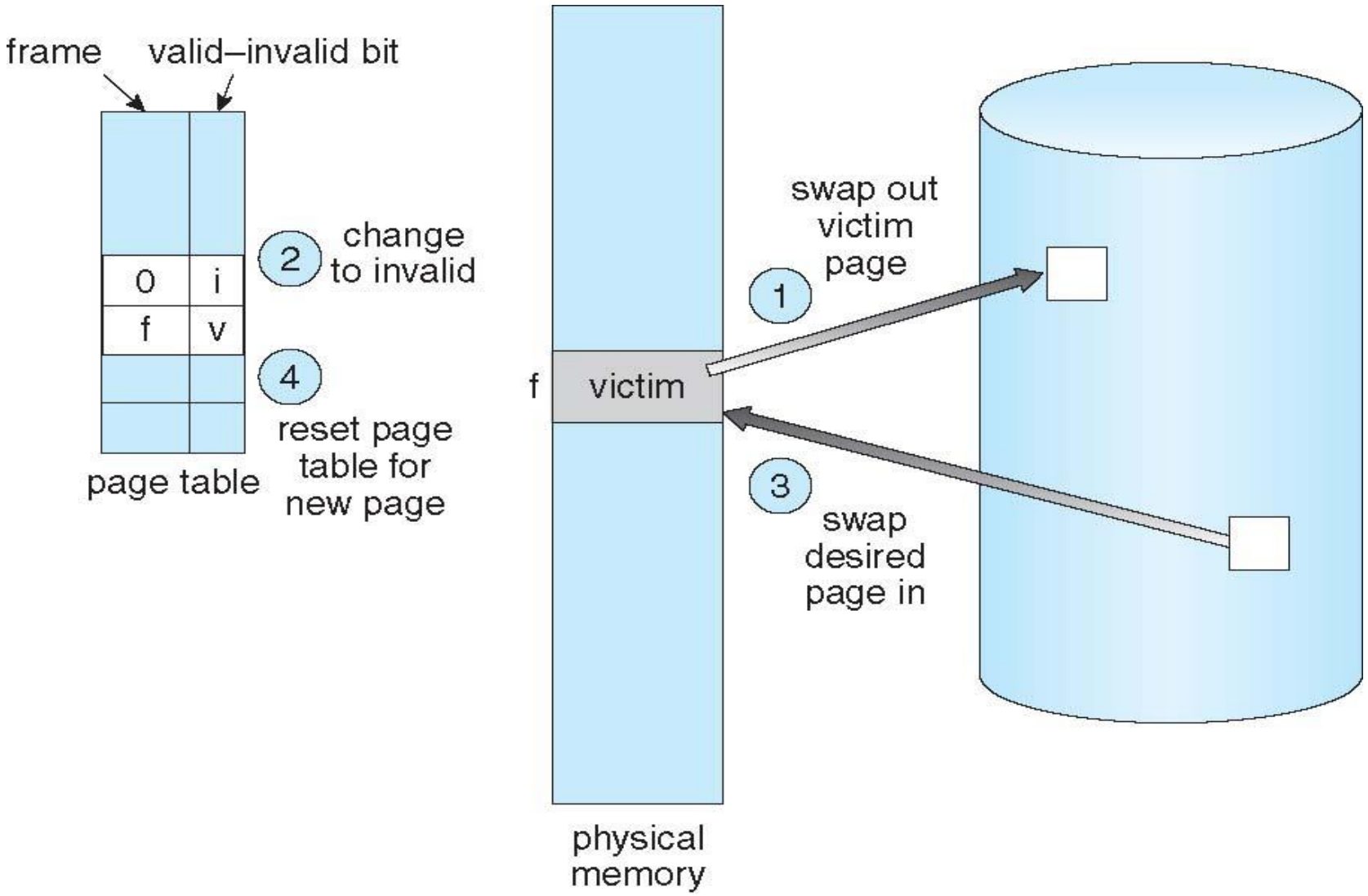
缺页中断处理过程


page is on backing store
3
operating system
2
trap
reference
1
load M
i
6
restart instruction
page table
free frame
5
reset page table
4
bring in missing page
physical memory

页面置换

- 页面置换的过程
  - 查找所需页在磁盘上的位置
  - 查找一空闲帧
    - 如果有空闲帧，那么就使用它
    - 如果没有空闲帧，那么就选择一个“牺牲”帧（victim frame）
    - 将“牺牲”帧的内容写到磁盘上，修改页表
  - 将所需页读入内存，修改页表
  - 重启用户进程

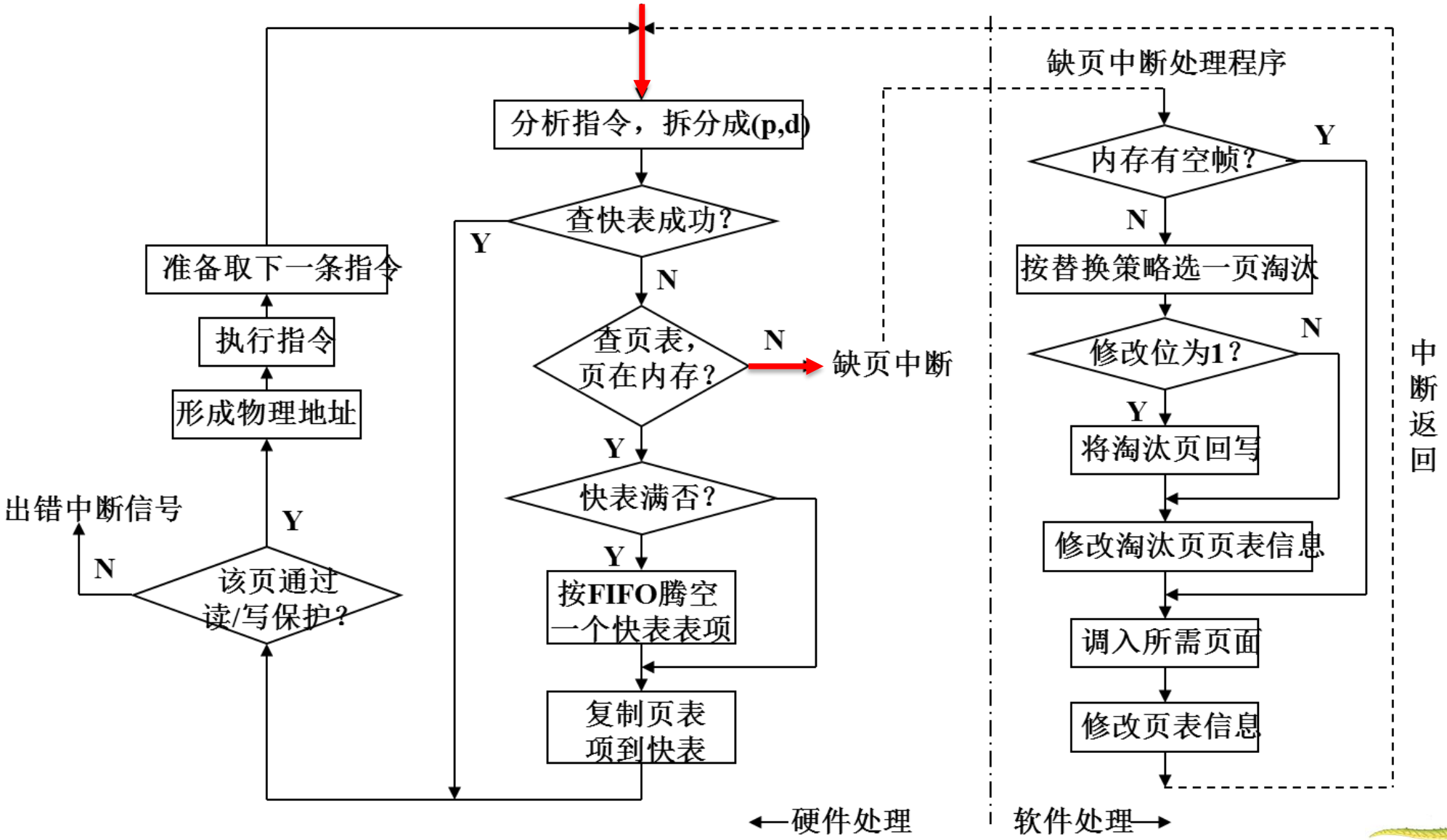
页面置换


frame
valid–invalid bit
0 i
f v
page table
2
change to invalid
4
reset page table for new page
f
victim
1
swap out victim page
3
swap desired page in
physical memory

页面置换

- 说明
  - 使用修改位（modify bit）来降低页传输的开销
    - 只有被修改过的页才写回至磁盘
  - 为实现按需调页，需解决的问题
    - 页面置换算法（ Page-replacement algorithm ）

请求分页系统中的地址变换过程



按需调页的性能

- 有效访问时间（EAT ,Effective Access Time）

$$有效访问时间 = (1-p) \times ma + p \times 页错误时间$$

  - p：缺页中断的概率（缺页率）(0≤p≤1)
    - 如果p等于0，则不存在缺页
    - 如果p等于1，则每次访问都存在缺页
  - ma：访问内存的时间，一般为10-200ns
  - 页错误时间
    - 缺页中断
    - 重新启动进程

按需调页的性能

- 例：设平均页错误处理时间为8ms，内存访问时间为200ns，则有效访问时间为：

  EAT = (1 – p) x 200 + p x 8,000,000

  = 200 + p x 7,999,800

- 说明
  - 有效访问时间与缺页率有关
    - 当缺页率为1/1000时，EAT=8199.8ns=8.2us（微秒）
    - 如果需要性能降低不超过10%，则需要：

      200 + p x 7,999,800 < 220

      p x 7,999,800 < 20

      p < 0.0000025=25/10,000,000=1/4,000,000

小结

- 为什么引入虚拟内存
- 虚拟内存的实现
  - 部分装入
  - 置换
- 请求分页存储管理
  - 扩充页表
  - 缺页中断
  - 地址变换

5.3 页面置换算法

- 先进先出置换算法
- 最优置换算法
- 最近最少使用置换算法
- 最少使用置换算法
- Clock置换算法
- 页面缓冲算法

页面置换算法

- 引入
  - 在内存中没有空闲页面时被调用，如何选出一个被淘汰的页面
- 目的
  - 降低缺页率

页面置换算法

- 常用的置换算法
  - 先进先出置换算法-FIFO
  - 最优置换算法-OPT
  - 最近最少使用置换算法-LRU
  - 最少使用置换算法-LFU
  - Clock置换算法
  - 页面缓冲算法-Page Buffering

1、先进先出置换算法

- 算法思想
  - 选择最早进入内存的页，或者说在内存中停留时间最久的页面，作为替换的对象

1、先进先出置换算法

- 例：分配给作业的页帧数m=3，其页面访问顺序为： 7、0、1、2、0、3、0、4、2、3、0、3、2、1、2、0、1、7、0、1，计算访问过程中的缺页次数和缺页率

页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1#																				
2#																				
3#																				
缺页																				

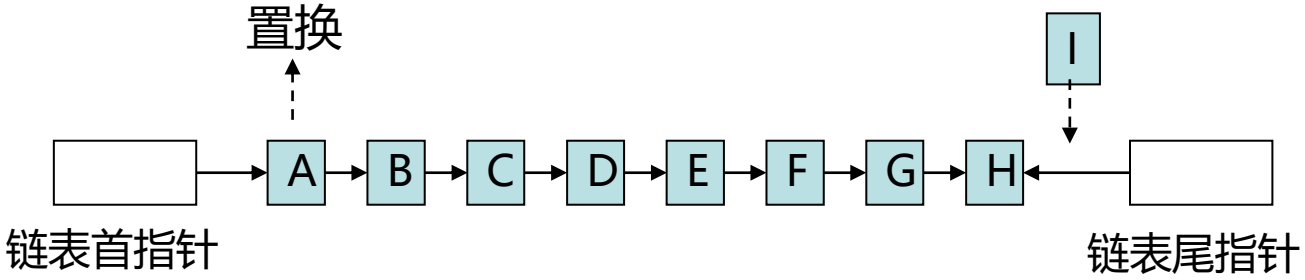
1、先进先出置换算法

- 缺页次数：12
- 缺页率：12/20=60%

页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1#	7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
2#		0	0	0		3	3	3	2	2	2			1	1			1	0	0
3#			1	1		1	0	0	0	3	3			3	2			2	2	1
缺页				*		*	*	*	*	*	*			*	*			*	*	*

1、先进先出置换算法

- 算法实现
  - 把进程已调入内存的页面，按进入内存的先后次序链接成一个FIFO队列
  - 置换时，将队首指针所指页面依次换出，换入的页链接在队列队尾

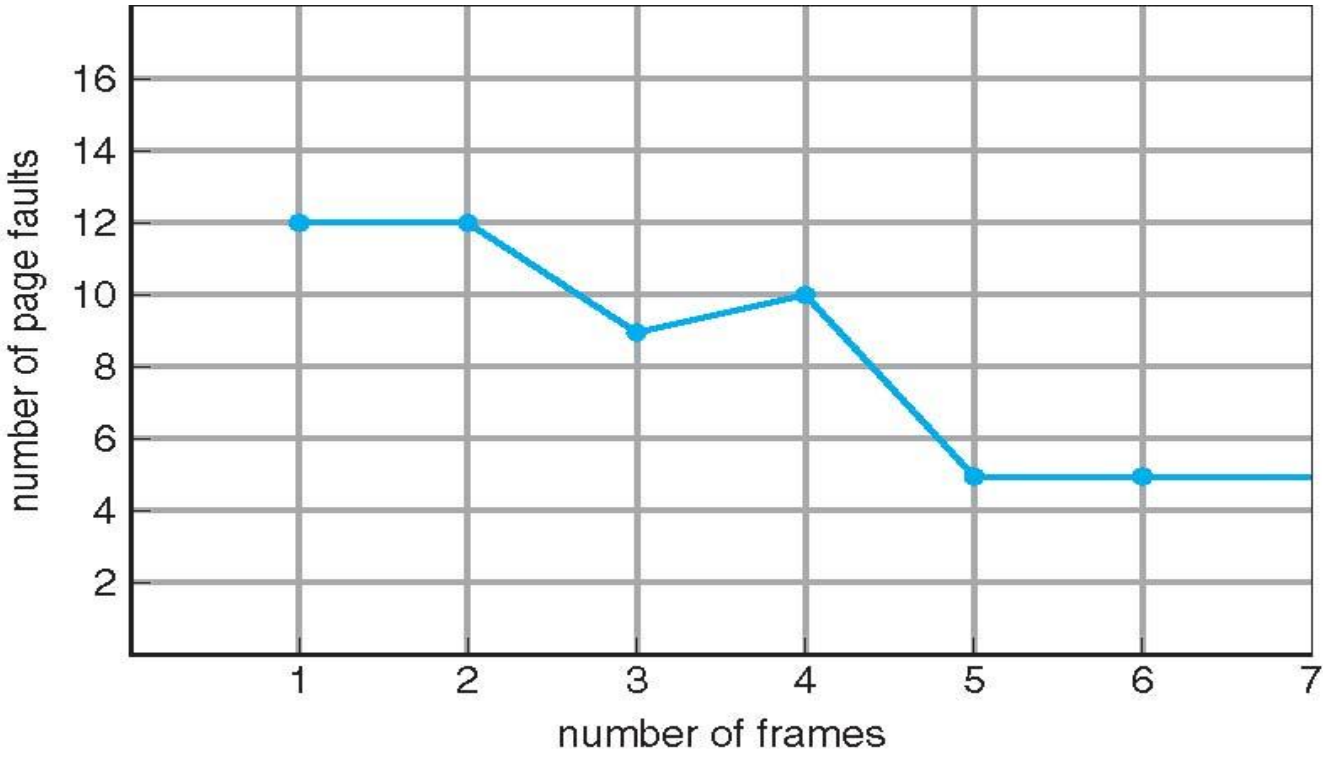


1、先进先出置换算法

- 优点
  - 算法简单，容易实现
  - 按线性顺序访问的地址空间的情况下，该算法理想
- 缺点
  - 如果较早调入的页恰恰是经常被访问的页，则缺页率高
  - FIFO策略的异常现象

1、先进先出置换算法

- FIFO策略的异常现象
  - 一般情况下，分给作业的页帧数增多，发生缺页的可能性就下降
  - 但是，有时增加页帧数，缺页次数反而会上升



2、最优置换算法

- 算法思想
  - 在需要进行页面替换时，置换最长时间不会使用的页

页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1#																				
2#																				
3#																				
缺页																				

2、最优置换算法

- 缺页次数：6
- 缺页率：6/20=30%

页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1#	7	7	7	2		2		2			2			2				7		
2#		0	0	0		0		4			0			0				0		
3#			1	1		3		3			3			1				1		
缺页				*		*		*			*			*				*		

2、最优置换算法

- 优点
  - 缺页率最低
  - 无异常现象
- 缺点
  - 该算法难以实现，实际执行中无法预知页面的访问情况
  - 主要用来评判其他算法的性能

3、最近最久未用(LRU)页面置换算法

- 算法思想
  - 根据页面调入内存后的使用情况，用“最近的过去”作为“最近的将来”的近似，置换内存中最长时间没有使用的页

页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1#																				
2#																				
3#																				
缺页																				

3、最近最久未用(LRU)页面置换算法

- 缺页次数：9
- 缺页率：9/20=45%

页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1#	7	7	7	2		2		4	4	4	0			1		1		1		
2#		0	0	0		0		0	0	3	3			3		0		0		
3#			1	1		3		3	2	2	2			2		2		7		
缺页				*		*		*	*	*	*			*		*		*		

3、LRU页面置换算法

- 优点
  - 性能接近最佳算法
- 缺点
  - 需要记录页面使用时间的先后关系
  - 算法实现需要一定的硬件支持
    - 计数器
    - 堆栈

3、最近最久未用（LRU）置换算法

- 计数器实现
  - 为每一个内存的页面设立移位寄存器
  - 当页面被访问时，寄存器最高位置1，定期右移，并且最高位补0
  - 寄存器值最小的页面就是最近最少使用的页面

页号	移位寄存器的内容							
	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

3、最近最久未用（LRU）置换算法

- 堆栈实现
  - 设立一个特殊的栈，在堆栈中保存当前使用的页面的页号
  - 当某个页面被访问时，将页号移到栈顶
  - 这样，堆栈顶部总是最近使用的页，堆栈底部就是最久未使用页面

4	7	0	7	1	0	1	2	1	2	6
				1	0	1	1	2	1	2
		0	7	7	1	0	0	0	0	1
	7	7	0	0	7	7	7	7	7	0
4	4	4	4	4	4	4	4	4	4	7

习题4

- 在一个请求分页系统中，分配给作业的页帧数m=3，其页面访问顺序为：4，3，2，1，4，3，5，4，3，2，1，5，分别采用以下算法画出置换图，并计算缺页次数和缺页率
  - 最佳置换算法
  - FIFO置换算法
  - LRU置换算法

OPT	4	3	2	1	4	3	5	4	3	2	1	5
M=3												
缺页												

4、最少使用（LFU）置换算法

- 算法思想
  - 选择最近一段时间内使用频率最小的页面置换
  - 当有不止一个页面满足被替换的条件时，就在满足条件的页面里随便选一个淘汰

页面	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
1#	0	0	0	0			0	0		0		3	3	3	3	3
2#		1	1	1			1	3		1		1	1	1	1	1
3#			2	3			2	2		2		2	4	5	6	7
缺页				*			*	*		*		*	*	*	*	*

4、最少使用（LFU）置换算法

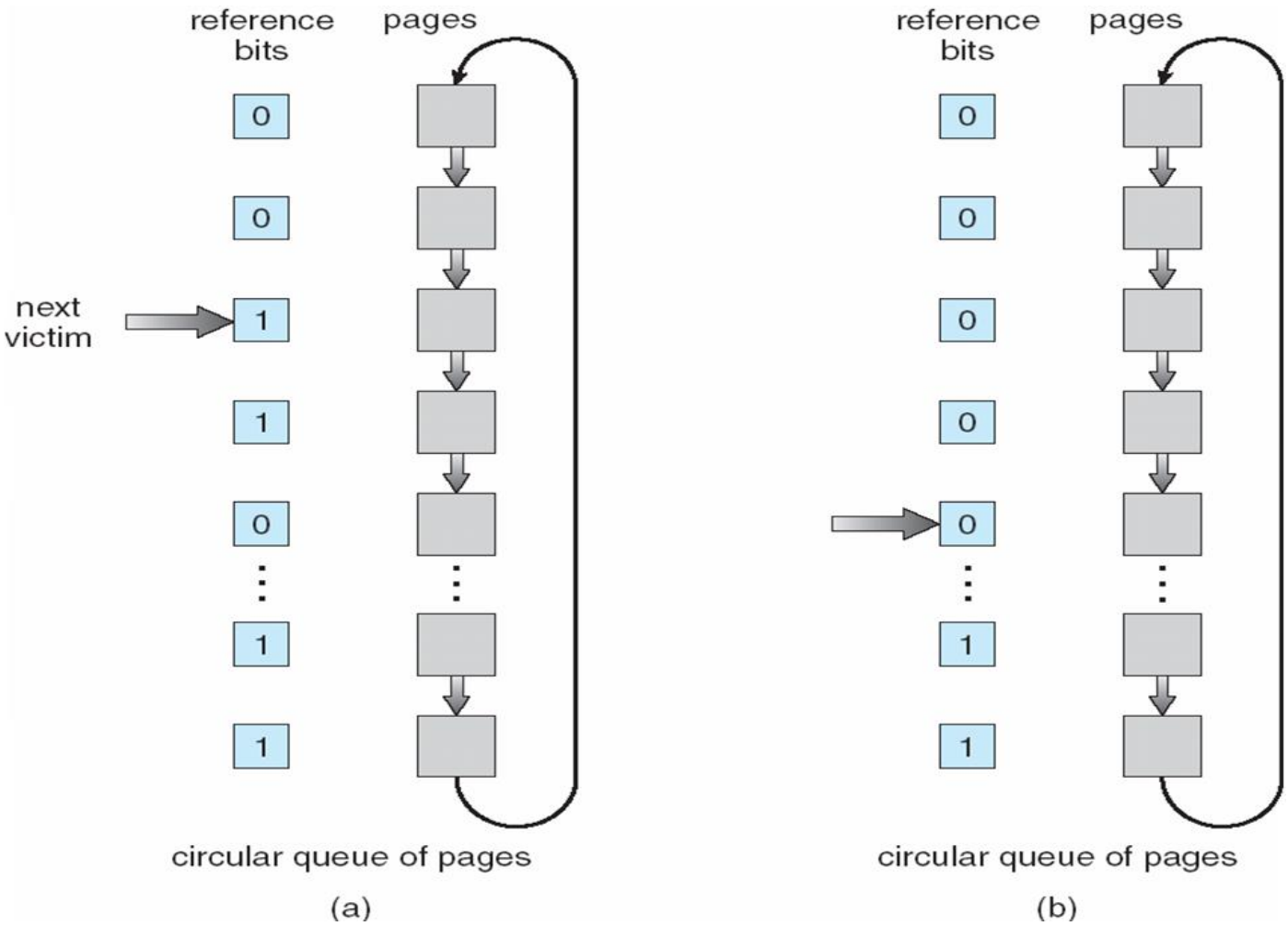
- 算法实现
  - 为每一个内存中的页面设立移位寄存器，用来记录该页面被访问的频率
  - 当页面被访问时，寄存器最高位置1，定期右移，并且最高位补0
  - 一段时间内 $\sum R_i$ 最小的页就是最少使用的页面

页号	移位寄存器							
	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1

5、Clock置换算法

- 算法思想
  - 将内存中所有的页面链成一个循环队列，每页设一个访问位A
    - A=0，表示该页没被访问过
    - A=1，表示该页被访问过
  - 置换时，按顺时针方向检查各个页的访问位
    - 若A=0，则置换该页
    - 若A=1，则将A位清“0”，然后把指针后移。因此，也称为“二次机会算法”

5、Clock置换算法


reference bits
pages
0
0
next victim
1
1
0
1
1
circular queue of pages
(a)
reference bits
pages
0
0
0
0
0
1
1
circular queue of pages
(b)

5、Clock置换算法

- 算法实现
  - 采用循环队列
- 优点
  - 考虑到页面在内存的访问情况
  - Clock算法可以直接使用页表项，无需额外的内存空间和硬件

5、Clock置换算法

- 改进型Clock置换算法
  - 在循环队列中，按照各个页面的访问位和修改位的组合来进行置换

页面类型	访问位A	修改位M	含义
1	0	0	没有被访问 没有被修改 最佳置换页
2	0	1	没有被访问 被修改过
3	1	0	被访问过 没有被修改
4	1	1	被访问 被修改

  - 优点
    - 给修改过的页更高的级别，从而降低I/O次数

6、页面缓冲算法

- 算法思想
  - 置换算法采用FIFO
  - 被置换页面暂时不换出内存，而是放入两个链表中：
    - 空闲页面链表：页面未被修改
    - 已修改页面链表：页面被修改，当已修改页面达到一定数目后，再将它们一起写回外存
- 特点
  - 减少磁盘I/O次数
    - 一旦置换的页面被再次调入，无需访问磁盘

小结

- 为什么引入页面置换算法
- 常用的页面置换算法
  - 先进先出置换算法-FIFO
  - 最优置换算法-OPT
  - 最近最少使用置换算法-LRU
  - 最少使用置换算法-LFU
  - Clock置换算法
  - 页面缓冲算法-Page Buffering

本章小结

连续分配存储管理	☑ 作业全部装入内存 ☑ 装入内存中连续的空间	单一连续分配方式	优点：易于实现 缺点：不支持多道，内存浪费
		固定分区分配方式	优点：支持多道程序并发执行 缺点：分区总数固定 限制进程数目 内碎片
		动态分区分配	优点：无内碎片 缺点：外碎片
		动态重定位分区分配	优点：解决外碎片问题（紧凑）
离散分配存储管理	☑ 作业全部装入内存 ☑ 装入内存中不连续的空间(页/段)	分页存储管理方式	优点：没有外碎片 页内碎片不超过页的大小
		分段存储管理方式	优点：易于实现共享、保护 缺点：有外碎片
		段页式存储管理方式	优点：综合了段式和页式两种方式 缺点：三次访存
虚拟存储管理	☑ 作业部分装入内存 ☑ 装入内存中不连续的空间	请求分页存储管理方式	优点：从逻辑上扩充内存
		请求分段存储管理方式	

End of Chapter 5