Chapter 9: Virtual Memory

Chapter 9: Virtual Memory

- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Memory-Mapped Files
- Allocating Kernel Memory
- Other Considerations
- Operating-System Examples

Objectives

- To describe the benefits of a virtual memory system
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames
- To discuss the principle of the working-set model

Background

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

虚拟内存

两个特点：

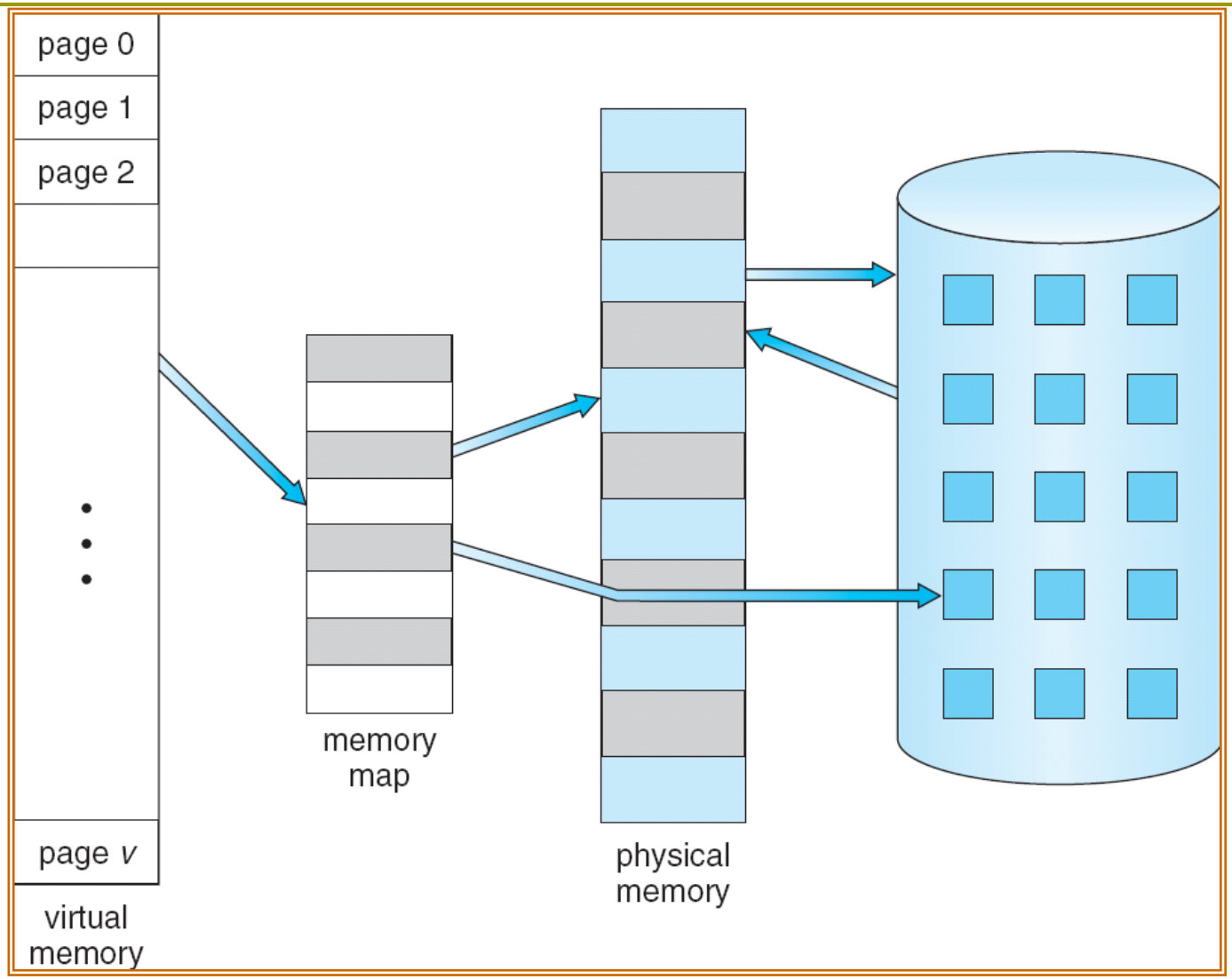
- 进程中所有存储器访问都是逻辑地址，这些逻辑地址在运行时被转换为物理地址；
- 一个进程可以划分为许多块，在执行过程中，这些块不需要连续地位于主存中。

虚拟内存

两个效果：

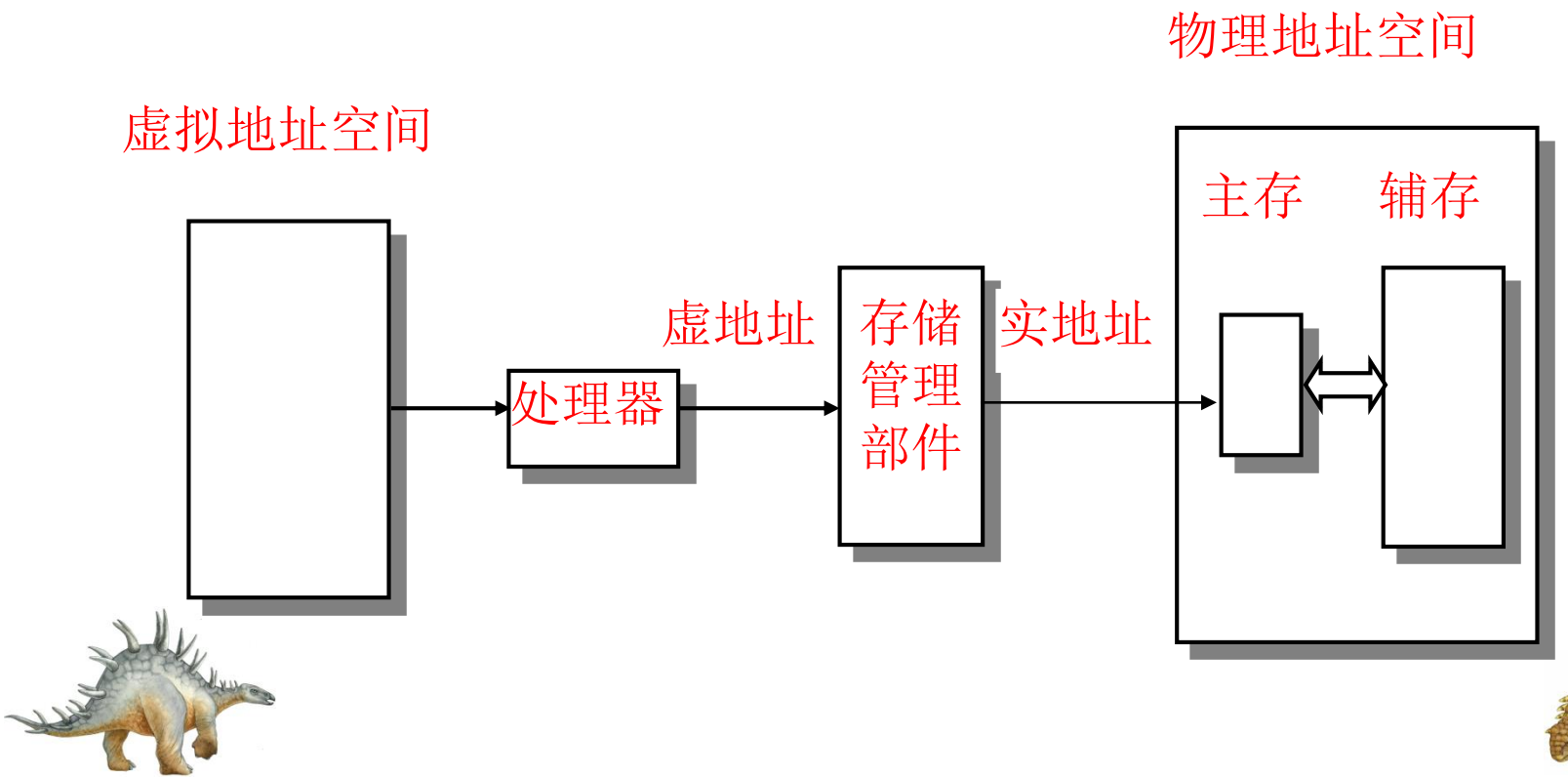
- 在主存中保留多个进程：
  - 由于对任何特定的进程都仅仅装入它的某些块，因此就有足够的空间放置更多进程。
- 进程可以比主存的全部空间还大
  - 通过分段或分页的虚拟内存，程序员可以获得巨大的主存，大小与磁盘储存器相关。

Virtual Memory That is Larger Than Physical Memory

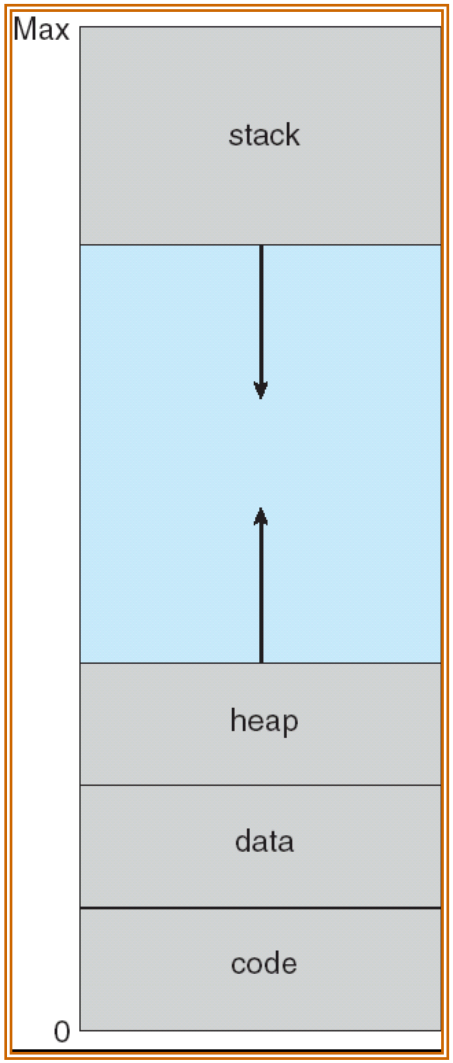


虚拟内存

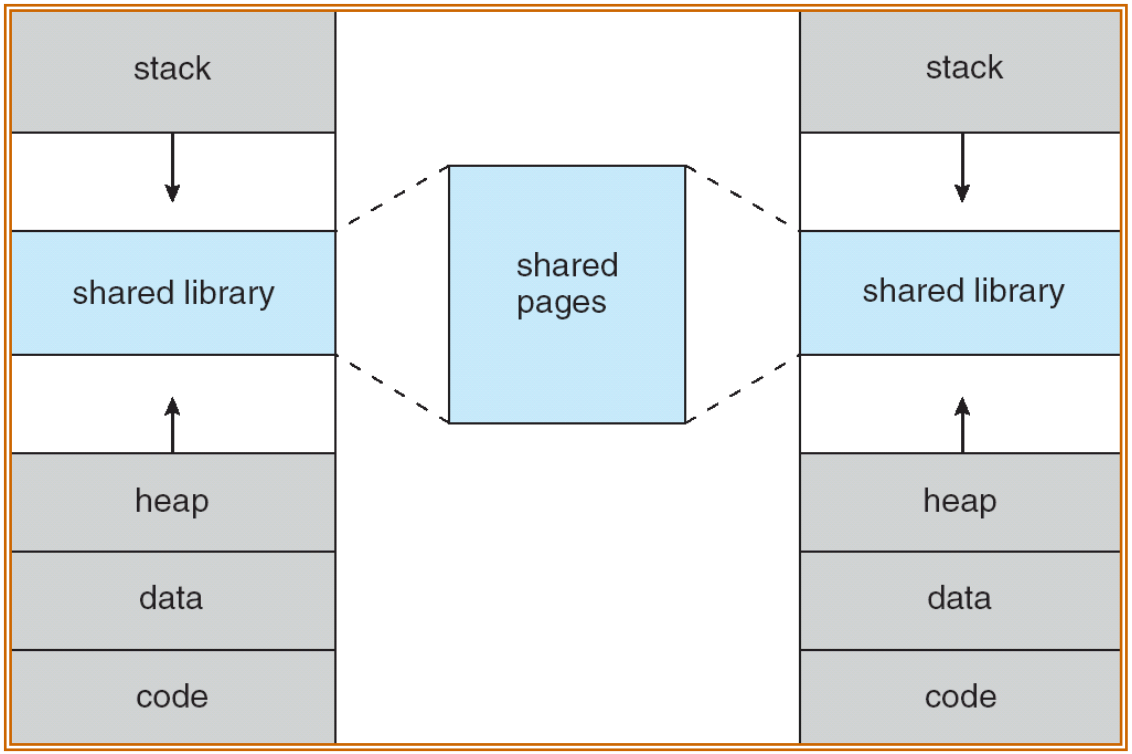
在具有层次结构存储器的计算机系统中，采用自动实现部分装入和部分对换功能，为用户提供一个比物理主存容量大得多的，可寻址的一种“主存储器”。



Virtual-address Space



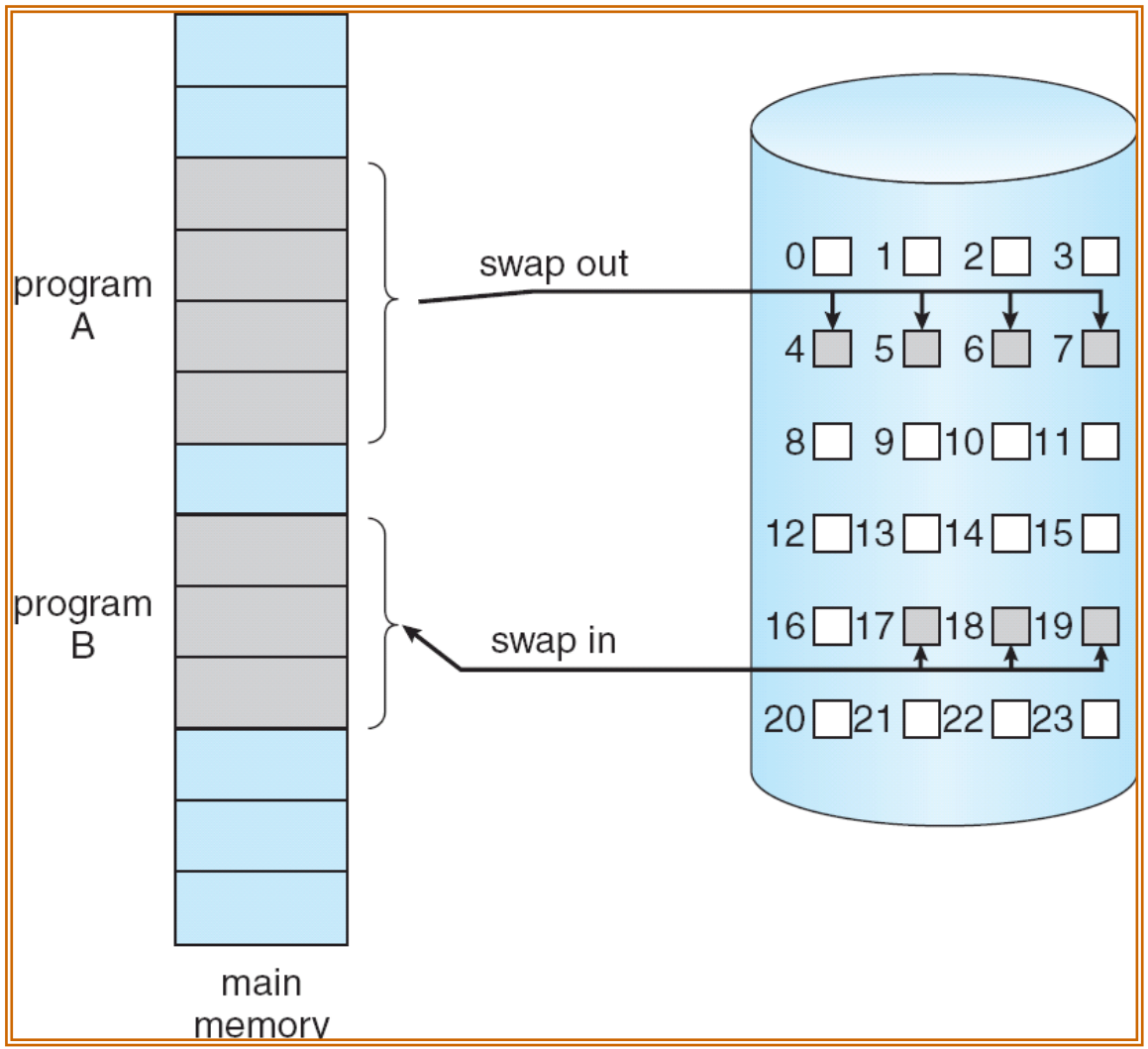
Shared Library Using Virtual Memory



Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

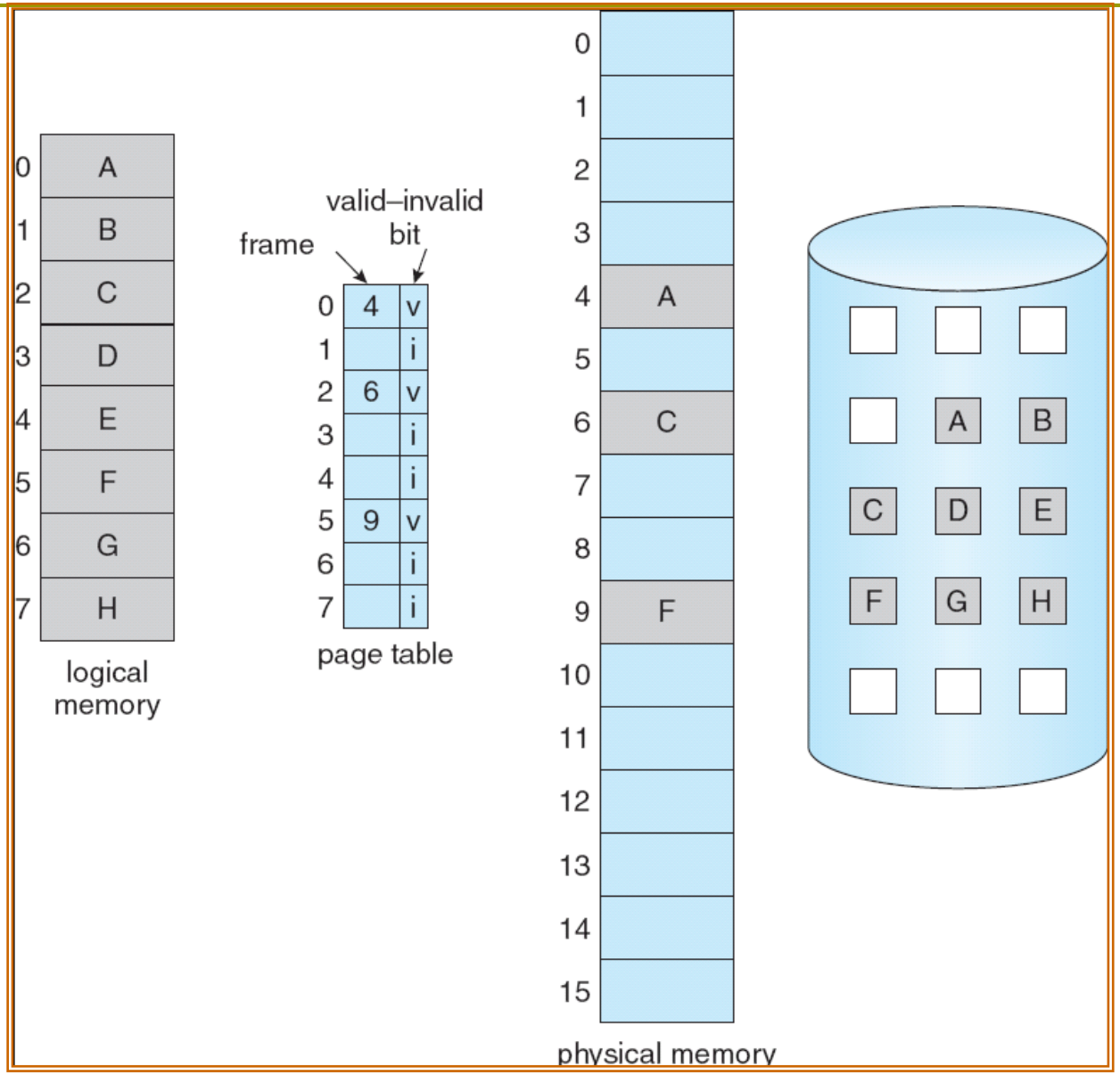
- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:

Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

page table

- During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory



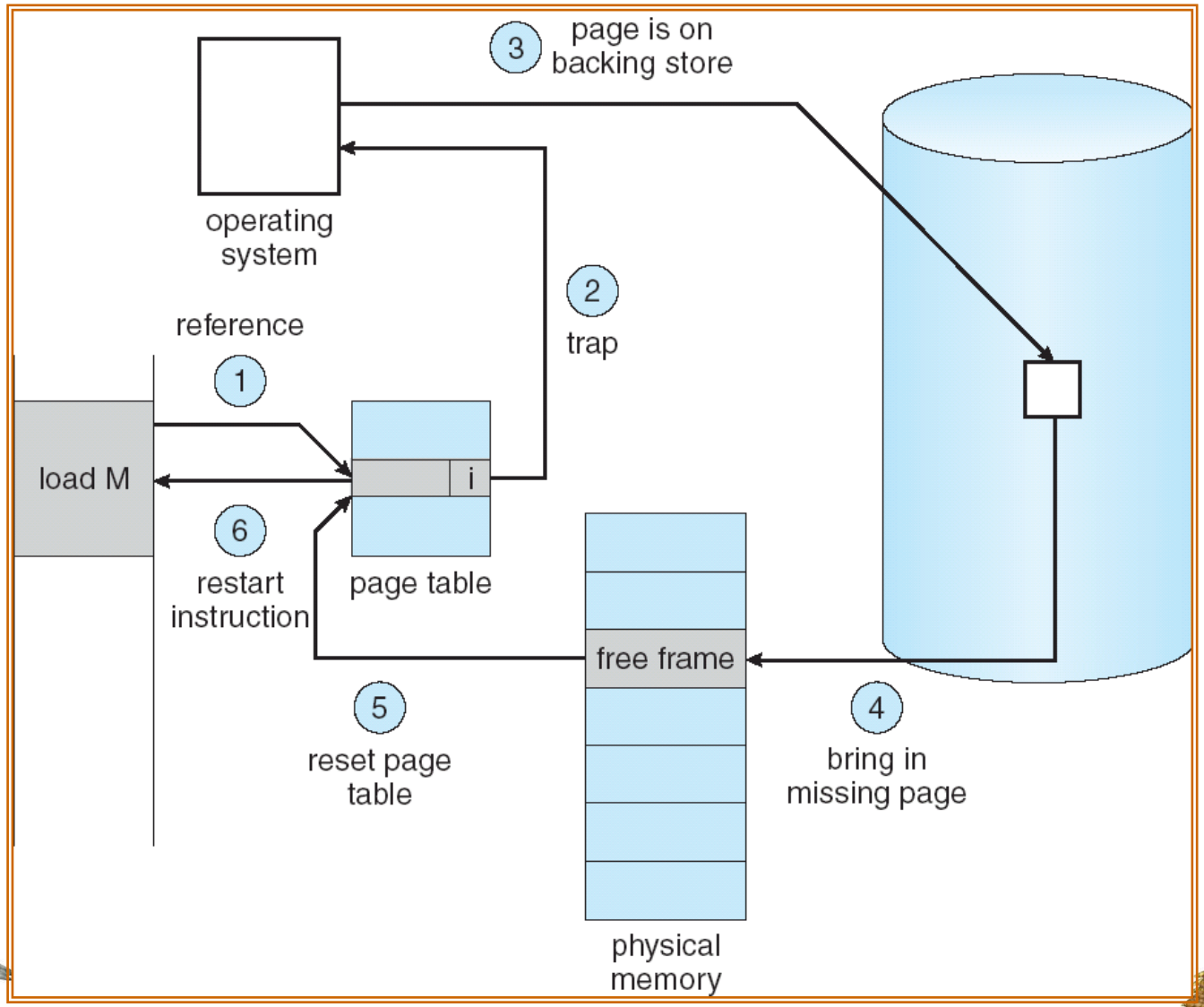
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

$$EAT = (1 - p) \times \text{memory access} + p\ (\text{page fault overhead} + \text{swap page out} + \text{swap page in} + \text{restart overhead})$$

Demand Paging Example

- **Memory access time** = 200 nanoseconds
- **Average page-fault service time** = 8 milliseconds
- **EAT** = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p x 200 + p x 8,000,000

  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then EAT = 8.2 microseconds.

Process Creation

- Virtual memory allows other benefits during process creation:
  - Copy-on-Write
  - Memory-Mapped Files (later)

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory
  If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- Free pages are allocated from a **pool** of zeroed-out pages

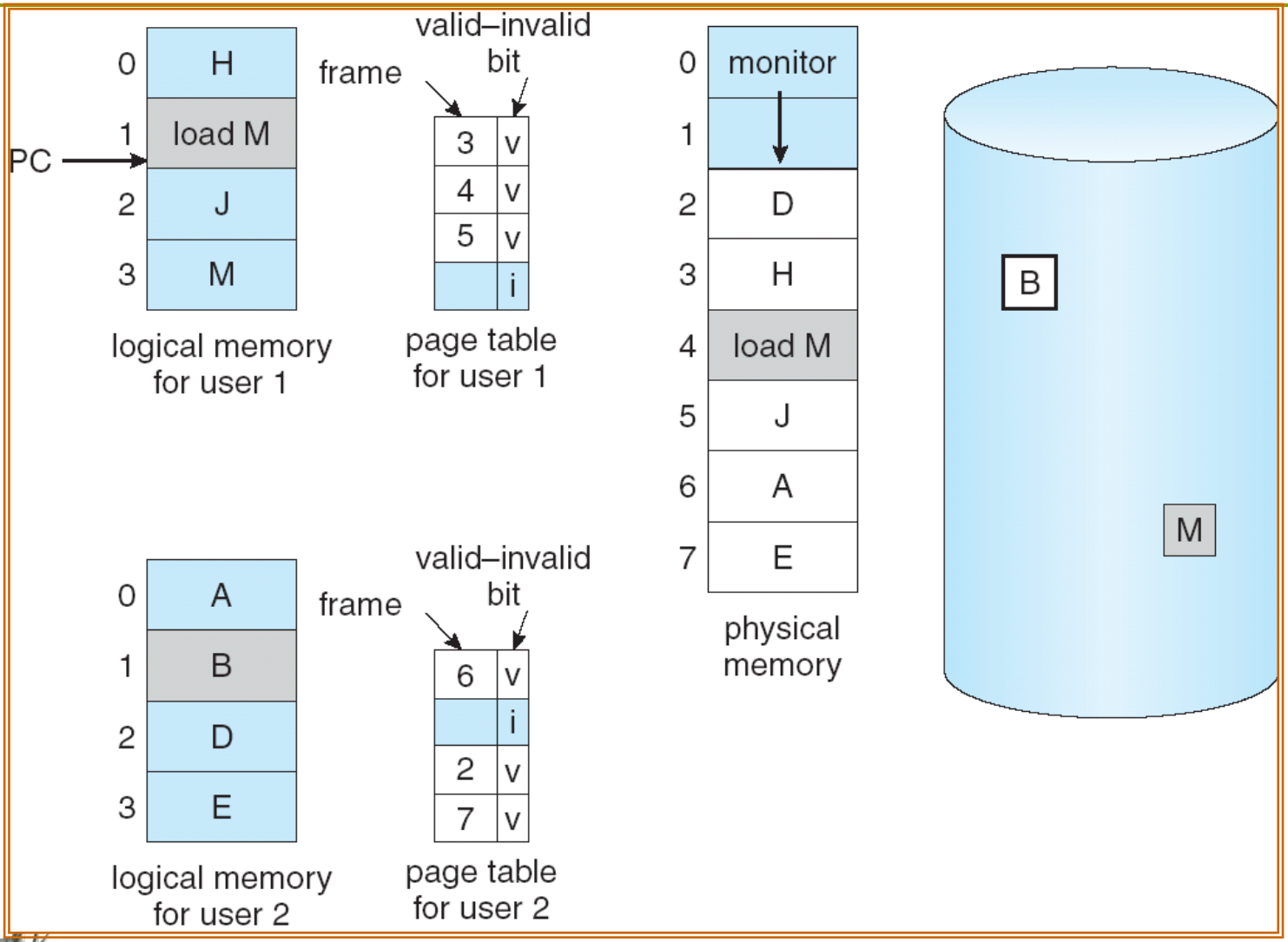
What happens if there is no free frame?

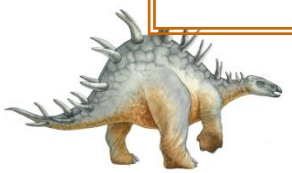
- Page replacement – find some page in memory, but not really in use, swap it out
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Need For Page Replacement

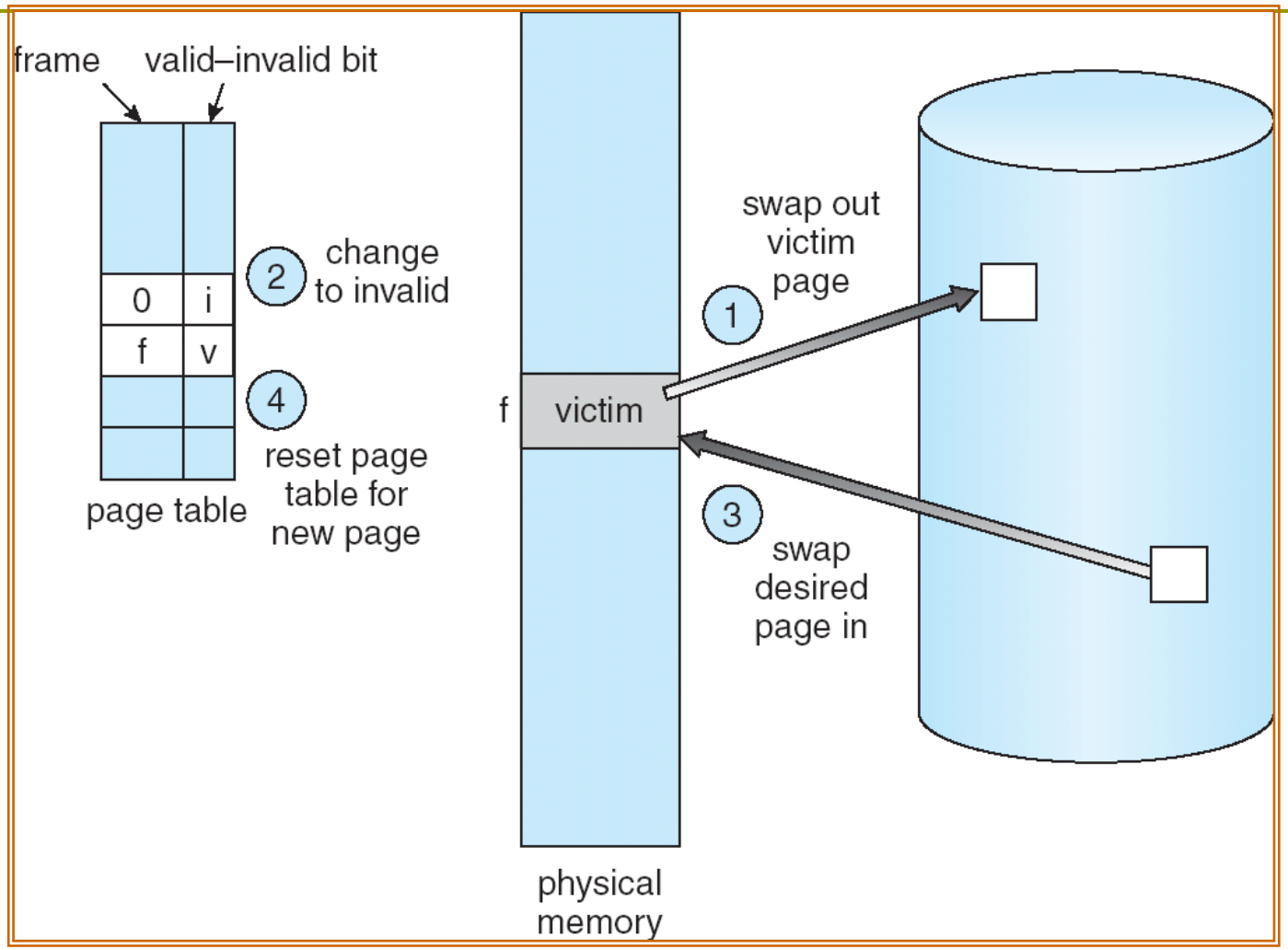




Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Restart the process

Page Replacement

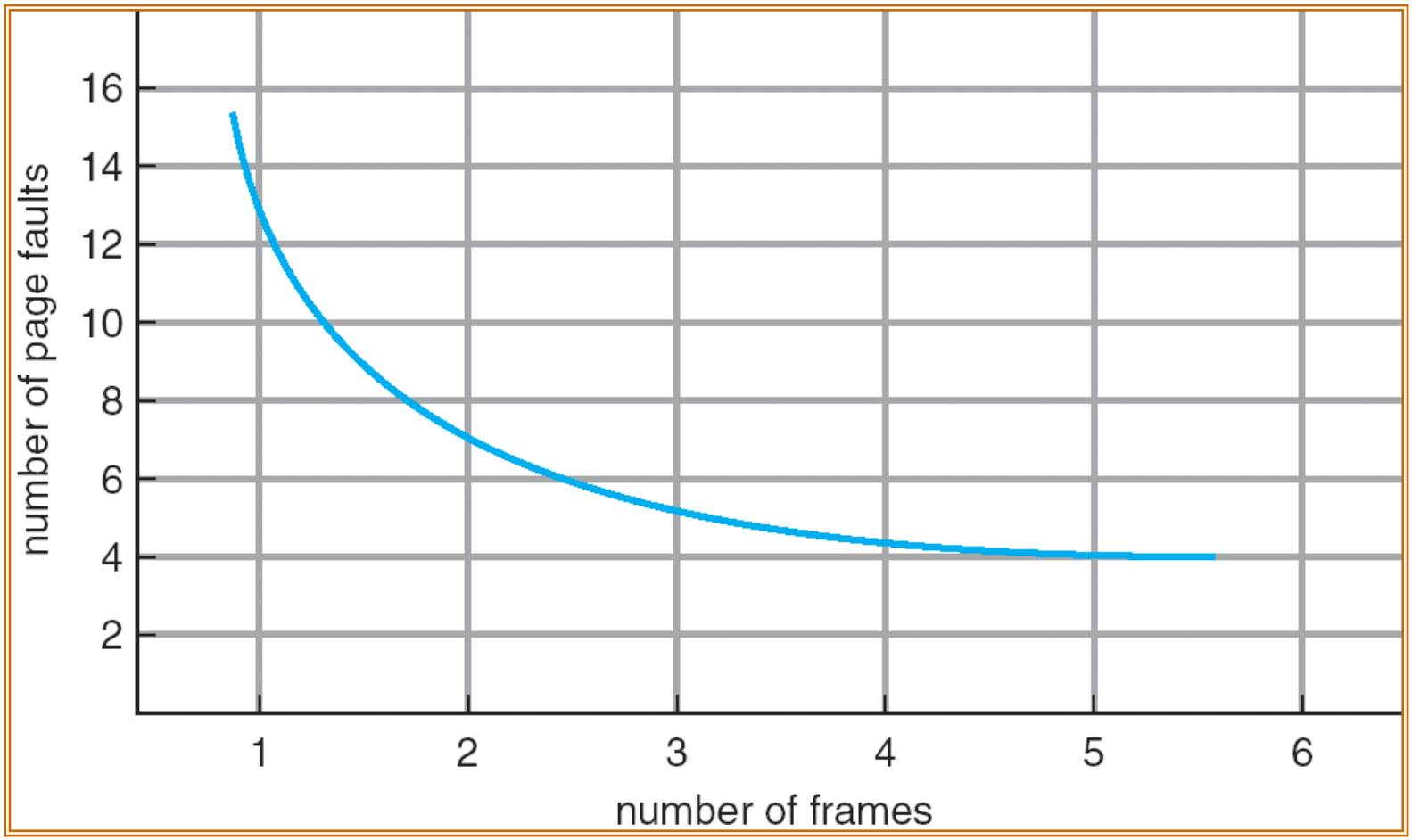


Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- In all our examples, the reference string is

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Graph of Page Faults Versus The Number of Frames



First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)
- 4 frames

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- Belady's Anomaly: more frames $\Rightarrow$ more page faults

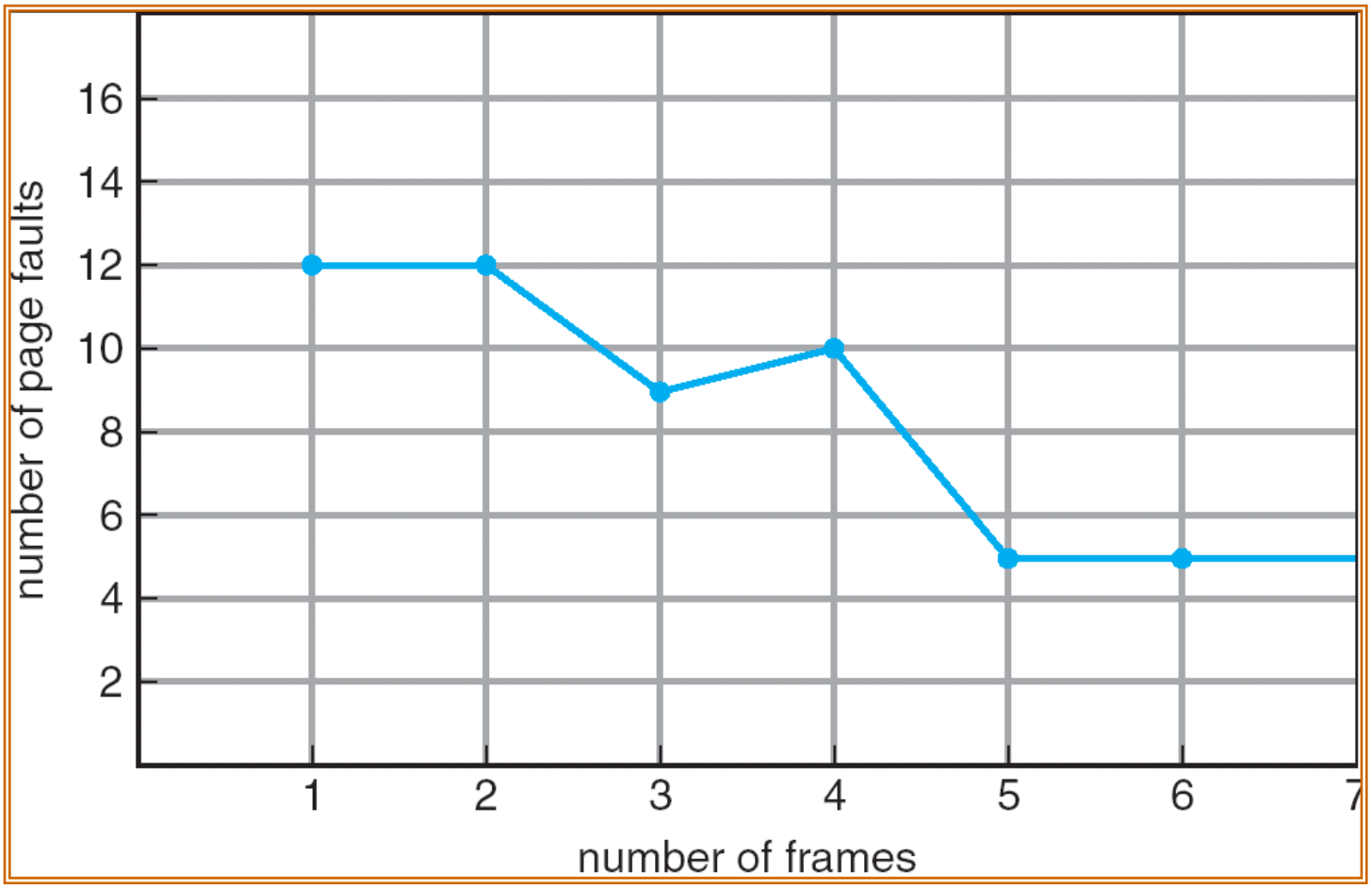
替换策略-先进先出（FIFO）

- 算法：总是淘汰最先调入主存的那一页，或者说在主存中驻留时间最长的那一页（常驻的除外）。
- 理由：最早调入内存的页面，其不再被访问的可能性最大。

替换策略-先进先出（FIFO）

- 特点：实现简单、适合线性访问、对其他情况效率不高
- 异常情况：在某些情况下会出现分配给进程的页面数增多，缺页次数反而增加的奇怪现象。

FIFO Illustrating Belady's Anomaly



替换策略-先进先出（FIFO）

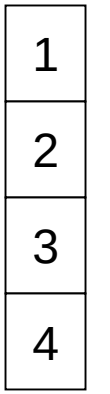
时刻	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
M	7	0 7	1 0 7	2 1 0	2 1 0	3 2 1	0 3 2	4 0 3	2 4 0	3 2 4	0 3 2	0 3 2	0 3 2	1 0 3	2 1 0	2 1 0	2 1 0	7 2 1	0 7 2	1 0 7
F	f	f	f	f	v	f	f	f	f	f	f	v	v	f	f	v	v	f	f	f

缺页率 σ =15/20=75%

Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



4

5

6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs

替换策略-最佳替换算法（OPT）

- **算法：**调入一页而必须淘汰一个旧页时，所淘汰的页应该是以后不再访问的页或距现在最长时间后再访问的页。
- **特点：**不是实际可行的算法，可用来作为衡量各种具体算法的标准，具有理论意义。

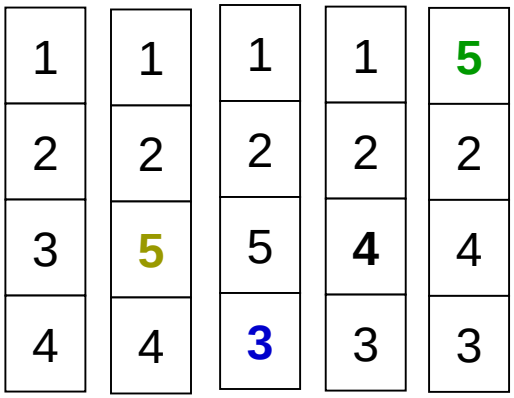
替换策略-最佳替换（OPT）

时刻	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
M	7	0 7	1 0 7	2 1 0	2 1 0	2 3 0	2 3 0	2 3 4	2 3 4	2 3 4	2 3 0	2 3 0	2 3 0	2 1 0	2 1 0	2 1 0	2 1 0	7 1 0	7 1 0	7 1 0
F	f	f	f	f	v	f	v	f	V	V	f	V	V	f	v	V	V	f	v	v

缺页率σ =9/20=45%

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**



- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

替换策略-最近最少使用（LRU）

- 算法：淘汰的页面是在最近一段时间里较久未被访问的那页。
- 原理：根据程序局部性原理，那些刚被使用过的页面，可能马上还要被使用，而在较长时间里未被使用的页面，可能不会马上使用到。

替换策略-最近最少使用算法（LRU）

时刻	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
M	7	0 7	1 0 7	2 1 0	2 1 0	2 3 0	2 3 0	4 3 0	4 2 0	4 2 3	0 2 3	0 2 3	0 2 3	1 2 3	1 2 3	1 2 0	1 2 0	1 7 0	1 7 0	1 7 0
F	f	f	f	f	v	f	v	f	f	f	f	v	v	f	v	f	v	f	v	v

缺页率 σ =12/20=60%

替换策略-时钟策略（Clock）

- **时钟策略**
  - 每一个帧关联一个附加位，称为使用位；
  - 页第一次调入内存，使用位设为1；
  - 当该页被随后访问到时，使用位也被置为1；
  - 当置换页发生时，查找使用位为0的一帧；
  - 当置换检索过程中，遇到的使用位为1的帧，操作系统将其置为0。

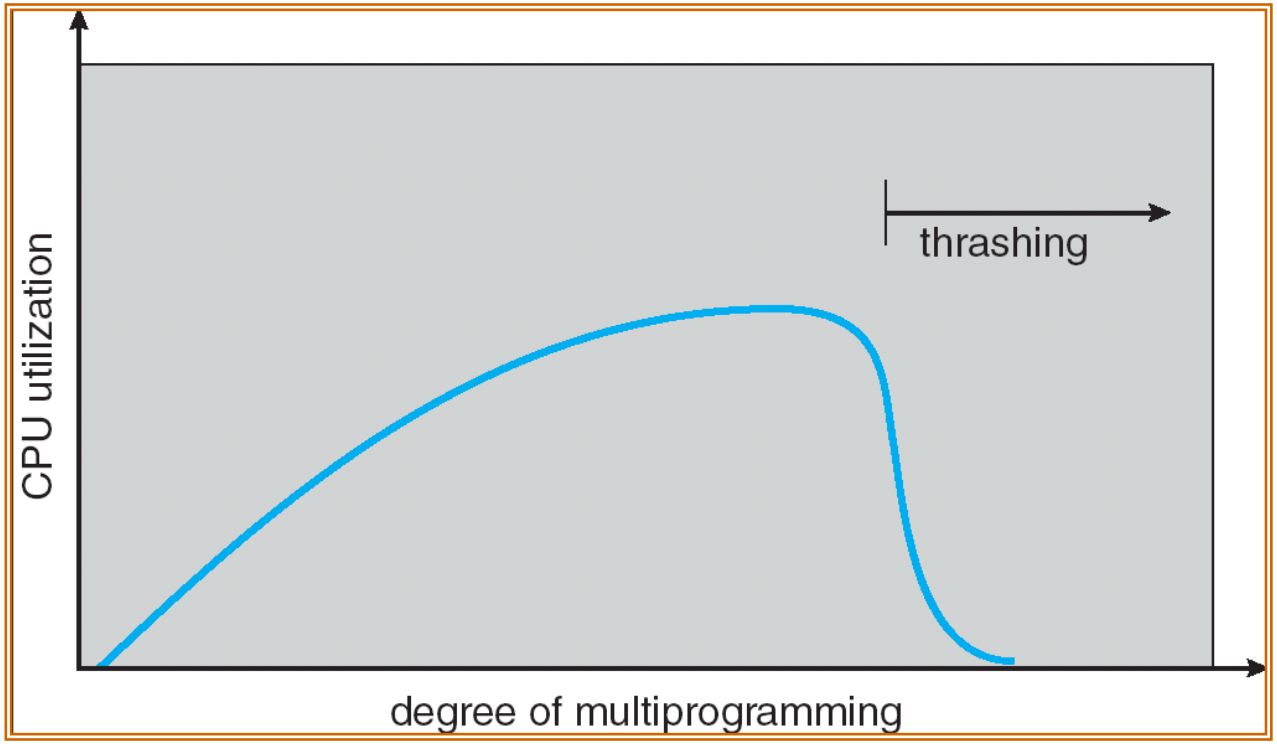
Global vs. Local Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** – each process selects from only its own set of allocated frames

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system
- **Thrashing** ≡ a process is busy swapping pages in and out

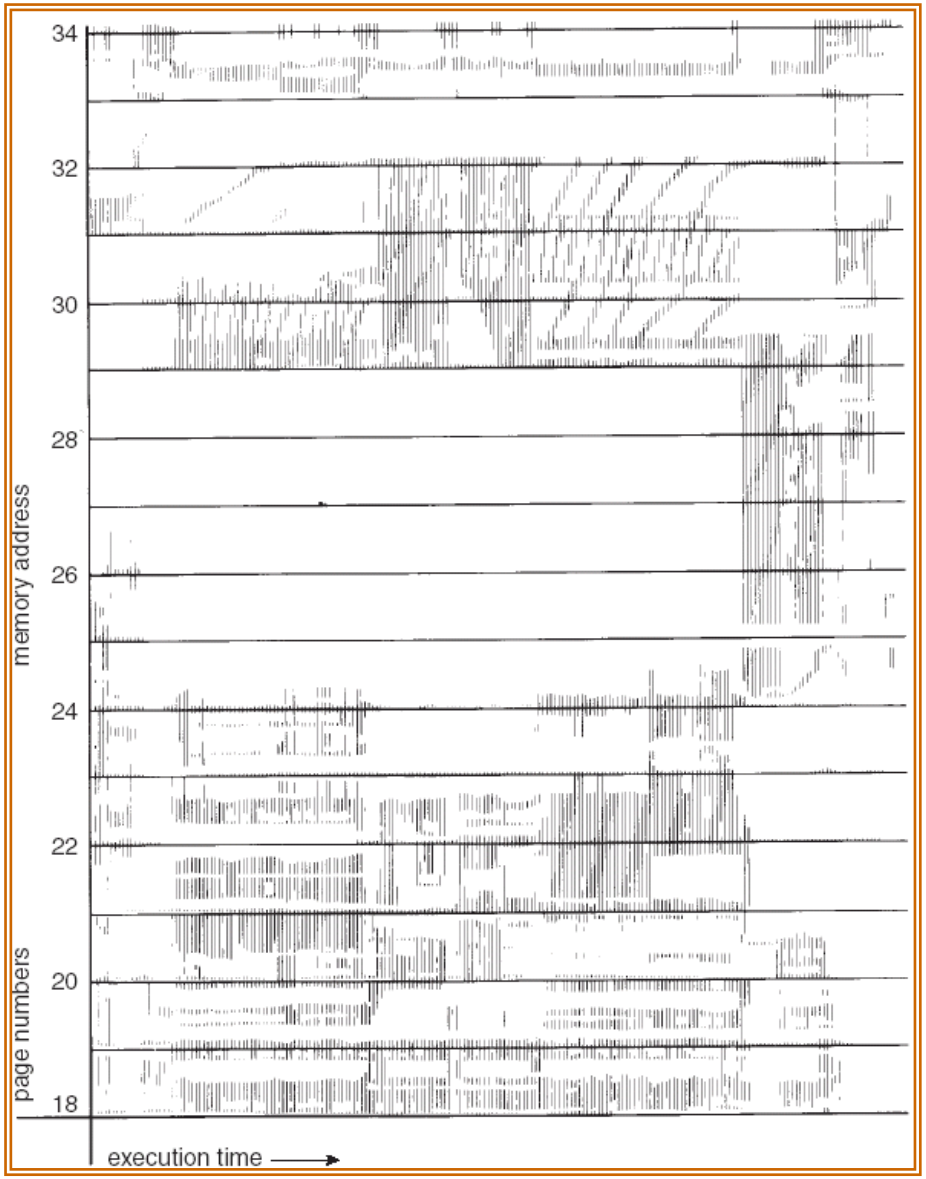
Thrashing (Cont.)



Demand Paging and Thrashing

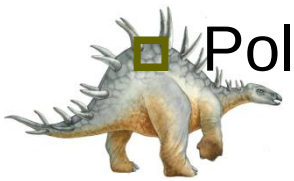
- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?
  Σ size of locality > total memory size

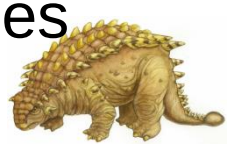
Locality In A Memory-Reference Pattern



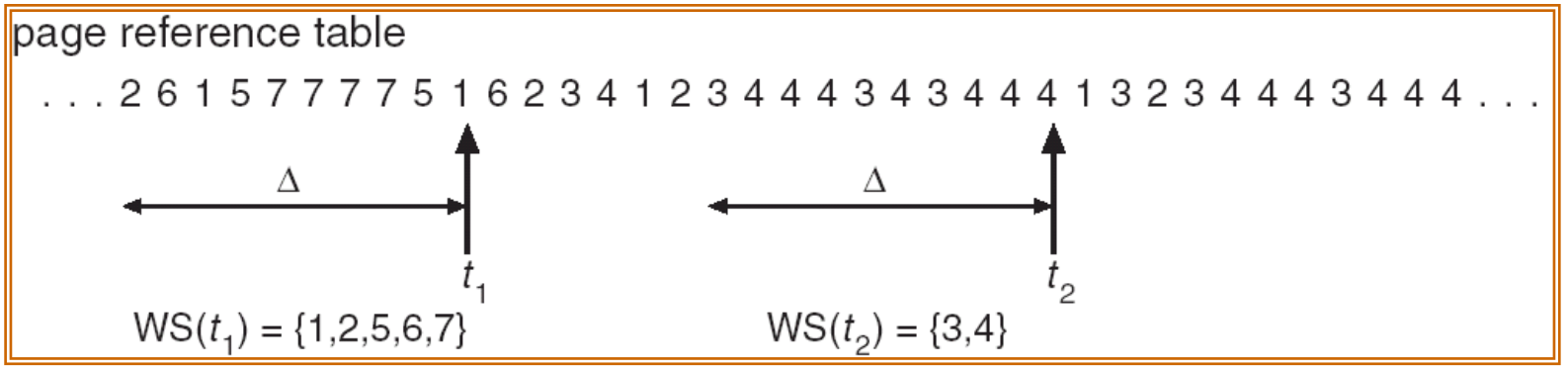
Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSS_i (working set of Process P_i) =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if $D >$ m, then suspend one of the processes



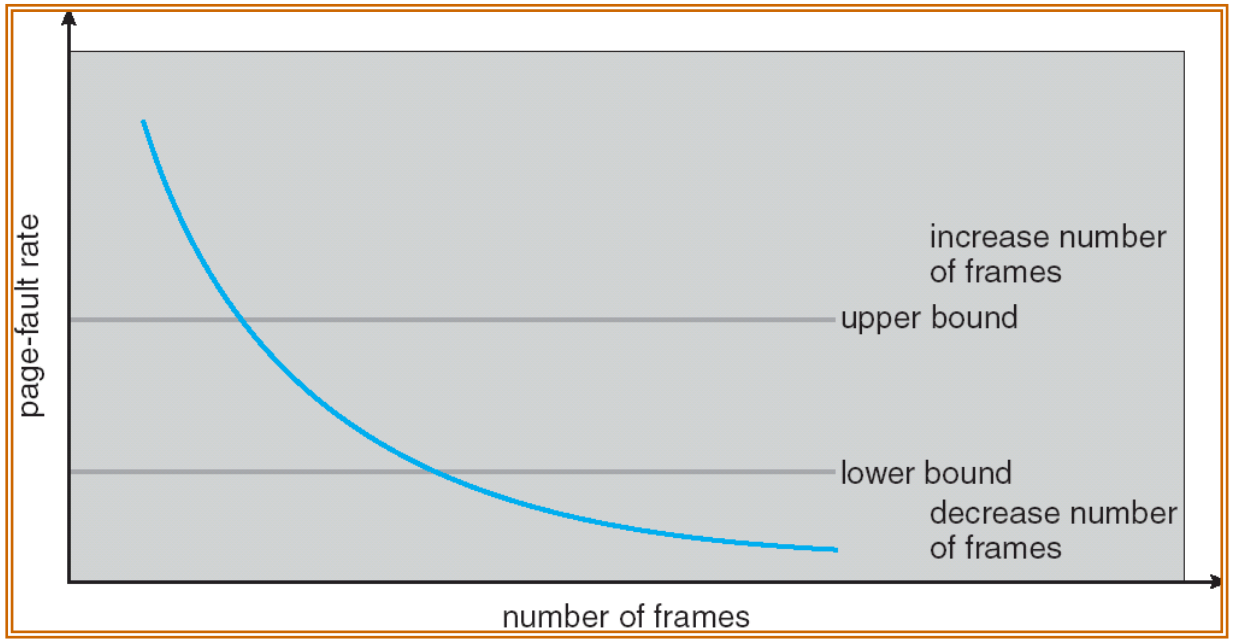


Working-set model



Page-Fault Frequency Scheme

- Establish "acceptable" page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



End of Chapter 9