操作系统原理与设计

第9章 VM3（虚存3）

陈香兰

中国科学技术大学计算机学院

2009年09月01日

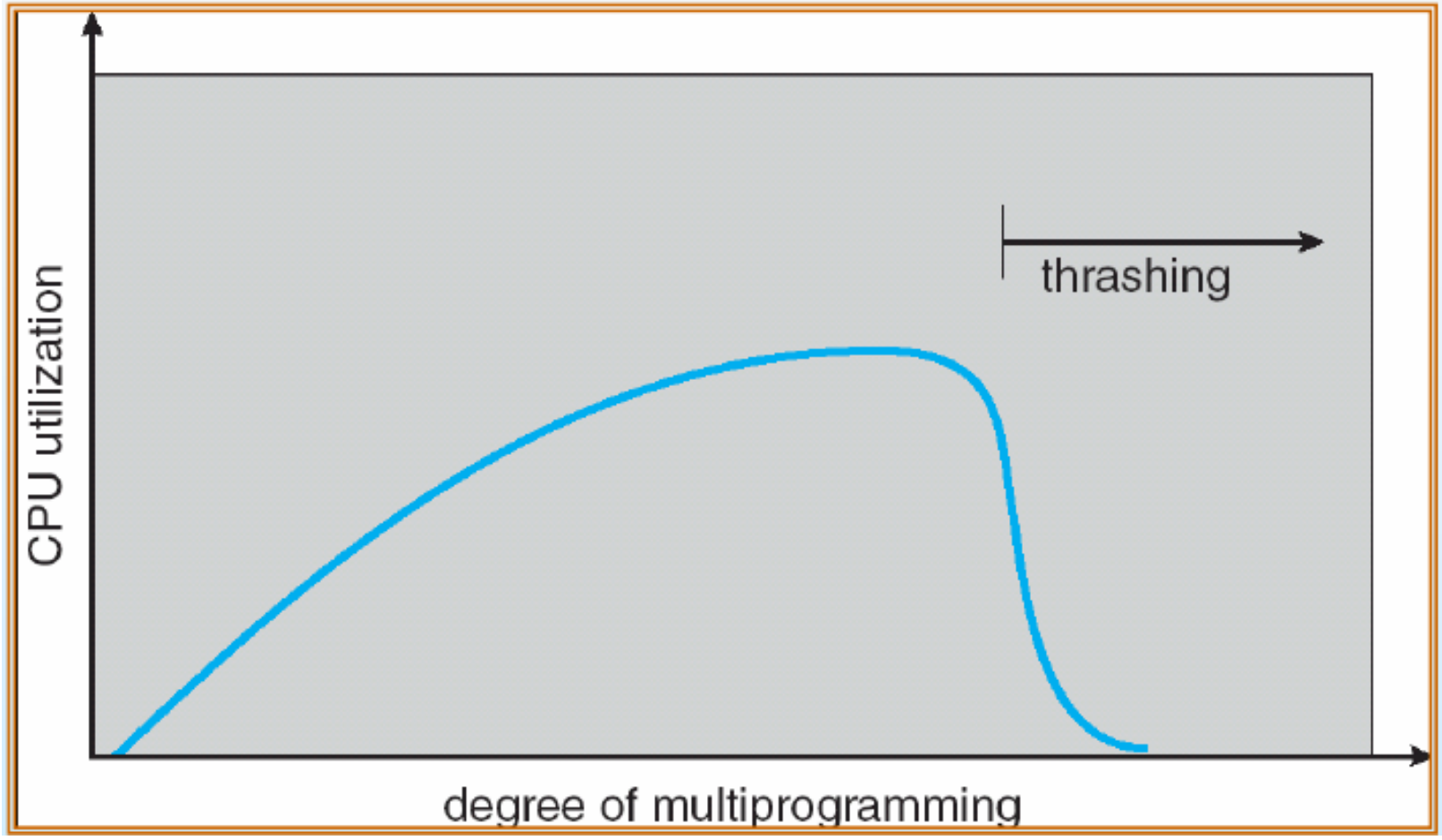
提纲

Outline

Thrashing I

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

Thrashing II

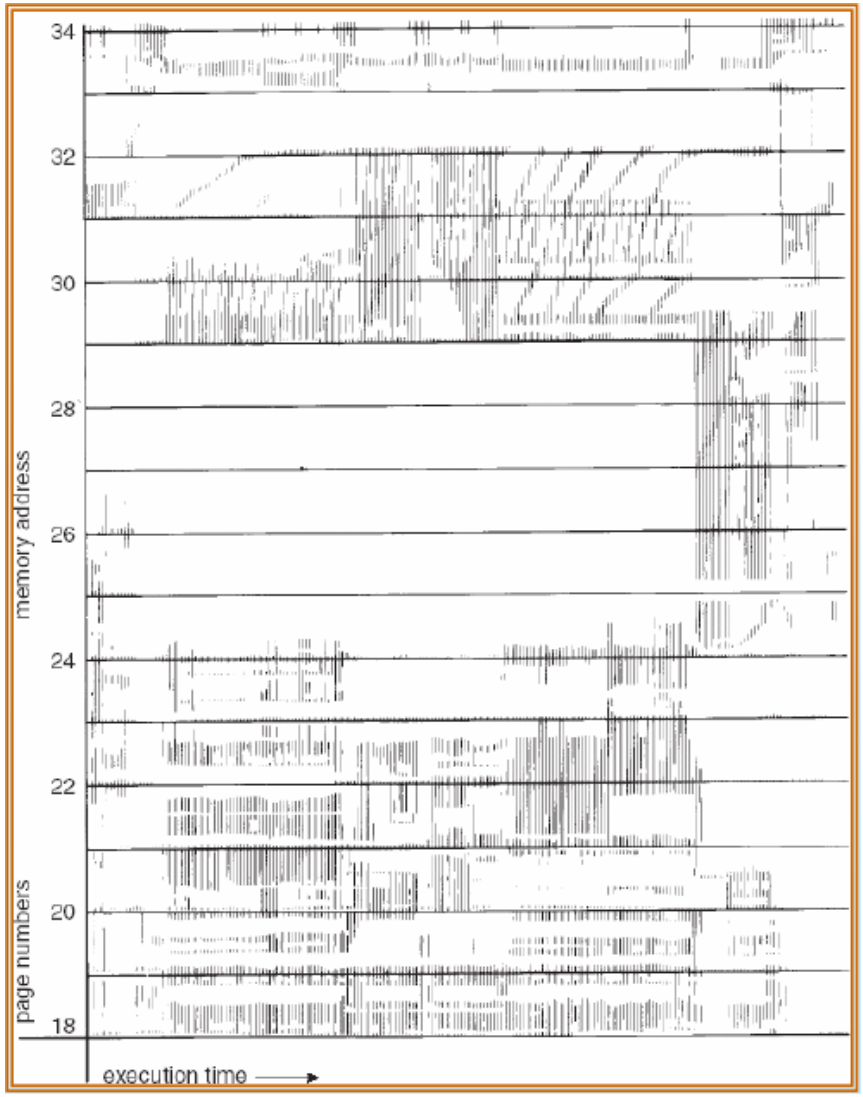


- Cause of trashing: unreasonable degree of multiprogramming

Thrashing III

- How to limit the effects of thrashing
  - local replacement algorithm? not entirely sloved.
  - we must provide a process with as many frames as it needs–locality
  - How do we know how many frames is needed?
    - working-set strategy $\Leftarrow$Locality model
- Locality model: This is the reason why demand paging works?
  - 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 (figure)

Thrashing IV



Outline

Working-Set Model(工作集模型) I

- the **working-set model** is based on the assumption of **locality**.

- let

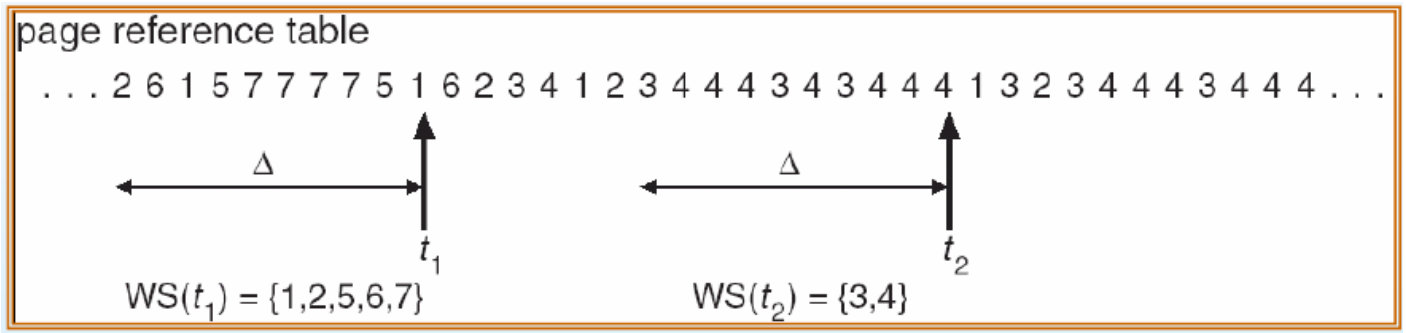
$$\begin{aligned} \Delta &\equiv \textit{working} - \textit{set window} \\ &\equiv \textit{a fixed number of page references} \end{aligned}$$

  Example:10,000 instruction

- **working set** (工作集): the set of pages in the most recent Δ page references.
  - an approximation of the program's locality.

Working-Set Model(工作集模型) II

- Example:
  $\Delta = 10$



- working set size: *WSS_i(working set of Process P_i) = total number of pages referenced in the most recent Δ*
  - veries in time, depend on the selection of Δ
    - 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

Working-Set Model(工作集模型) III

- For all processes in the system, currently

$$D = \Sigma WSS_i \equiv \textit{total demand frames}$$

  - if $D > m \Rightarrow$ Thrashing
  - Policy
    - if $D > m$, then suspend one of the processes

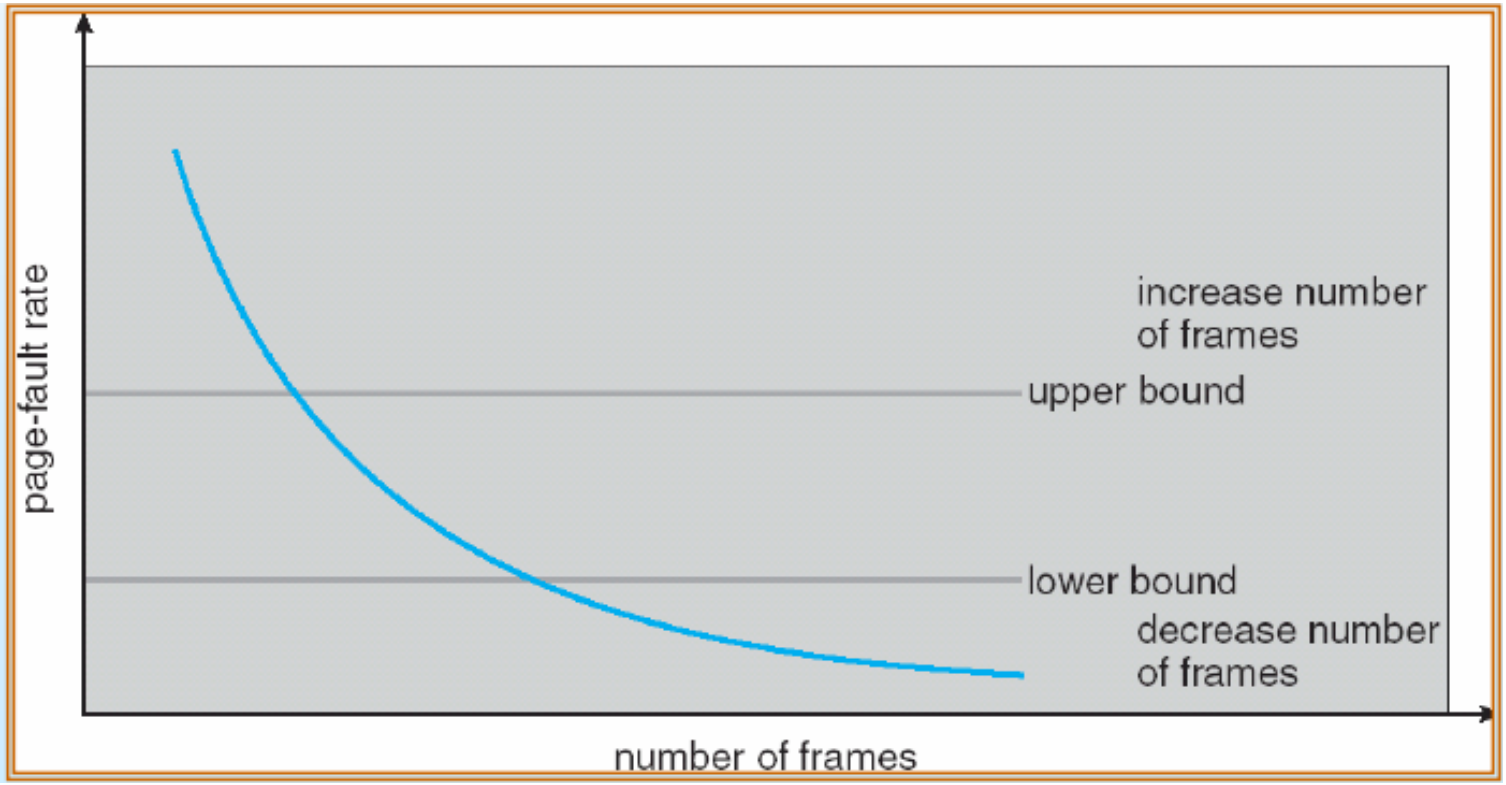
Keeping Track of the Working Set

- Approximate with interval timer + reference bits
- Example: $\Delta = 10{,}000$
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts, copy and sets the values of all reference bits to 0
  - If one of the bits in memory $= 1 \Rightarrow$ page in working set
- Why is this not completely **accurate**?
  - IN!! But where?
- Improvement:
  - 10 bits and interrupt every 1000 time units

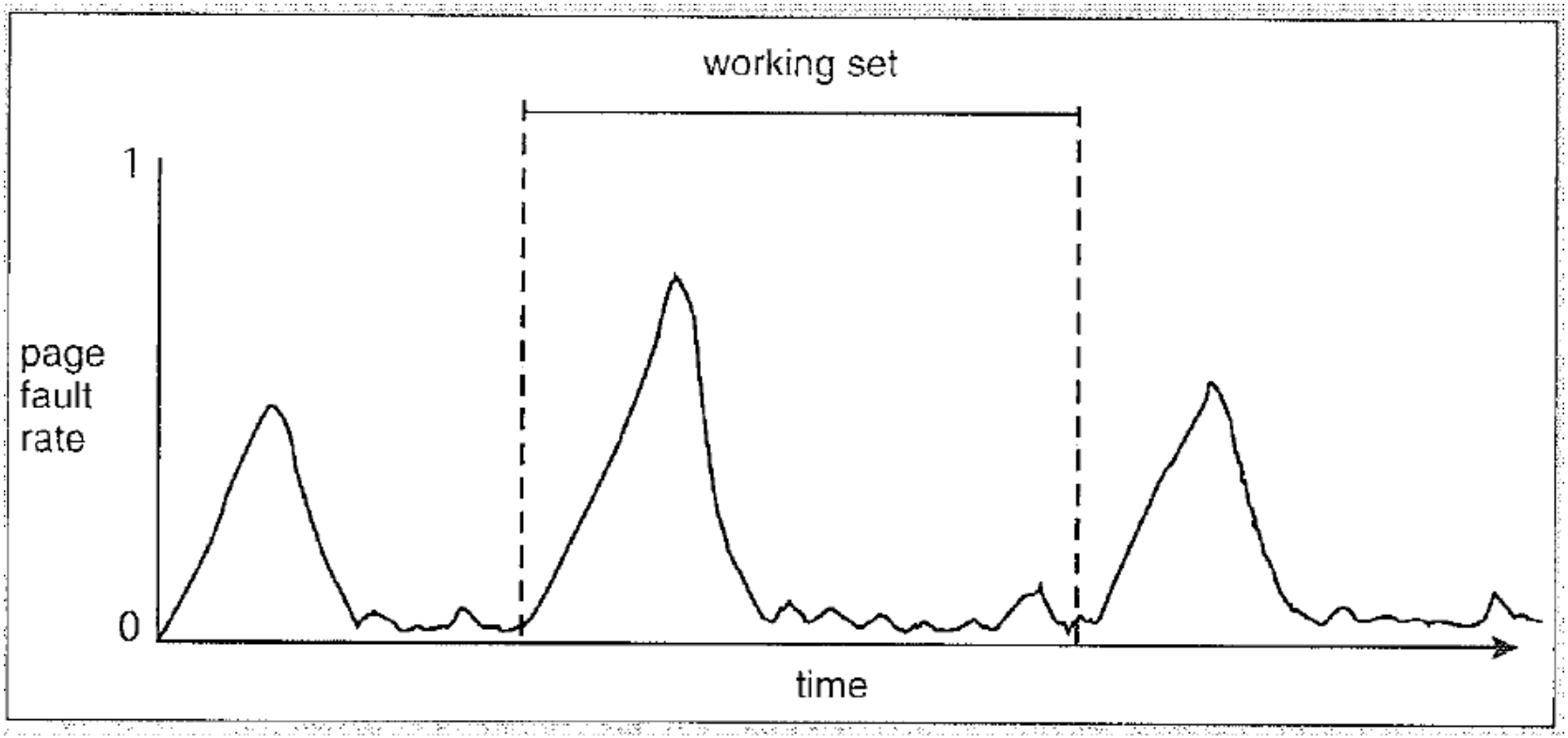
Outline

Page-Fault Frequency Scheme I

- helpful for controling trashing
- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



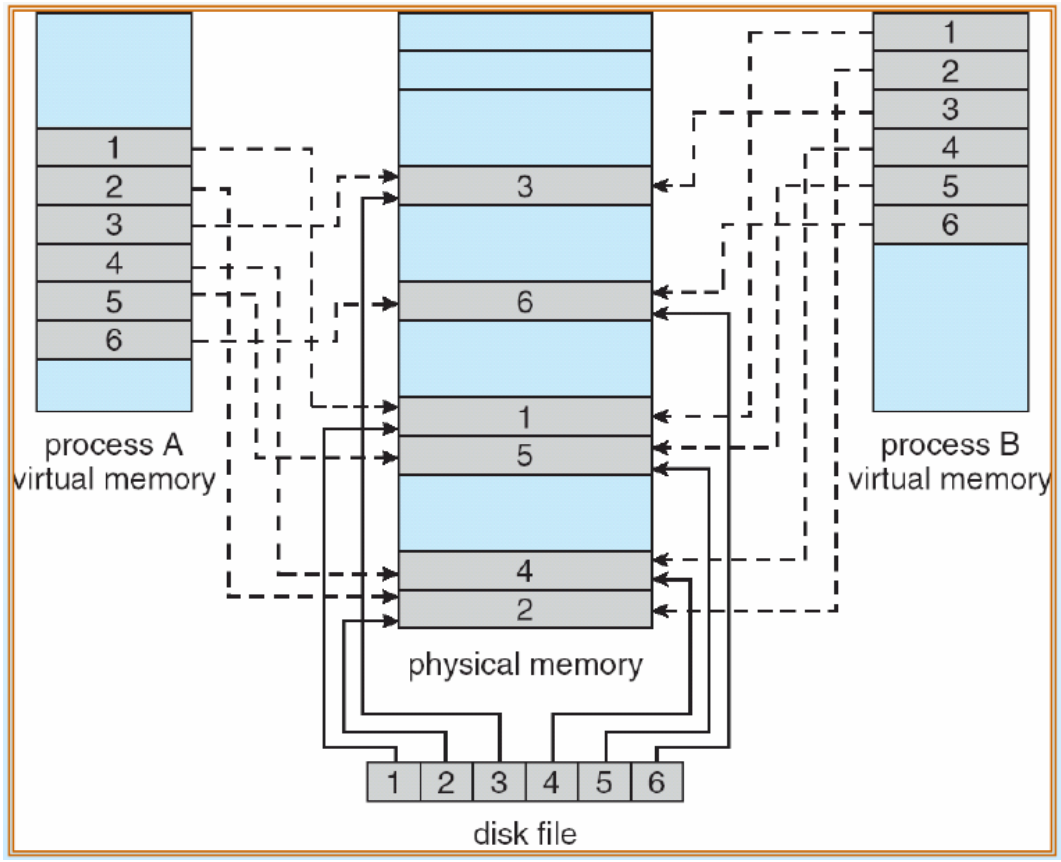
working sets and page fault rates



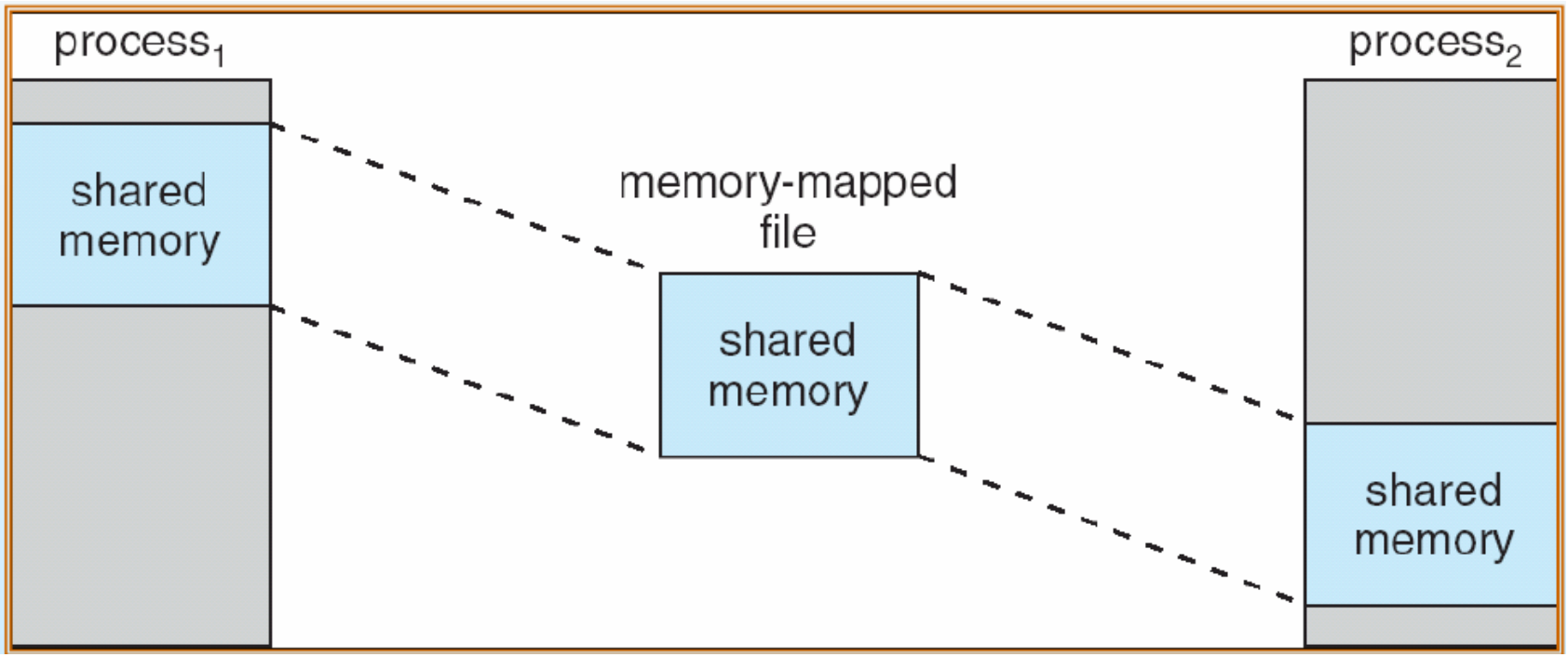
Memory-Mapped Files I

- Memory-mapped file I/O **allows file I/O to be treated as routine memory access** by mapping a disk block to a page in memory
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page.
  Subsequent reads/writes to/from the file are treated as ordinary memory accesses.
- Simplifies file access by treating file I/O through memory rather than read() write() system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared

Memory-Mapped Files II



Memory-Mapped Shared Memory in Windows



- reading the source code 英文351和352页，中文？和？页

Memory—mapped I/O

- many computer archtectures provide memory-mapped I/O
  - ranges of memory addresses are set aside and are mapped to the device registers.
  - directly read/write the mapped range of memory address for transfer data from/to device registers
  - fast response times
  - for example: video controler
    - displaying text on the screen is almost as easy as writing the text into the appropriate memory-mapped locations.

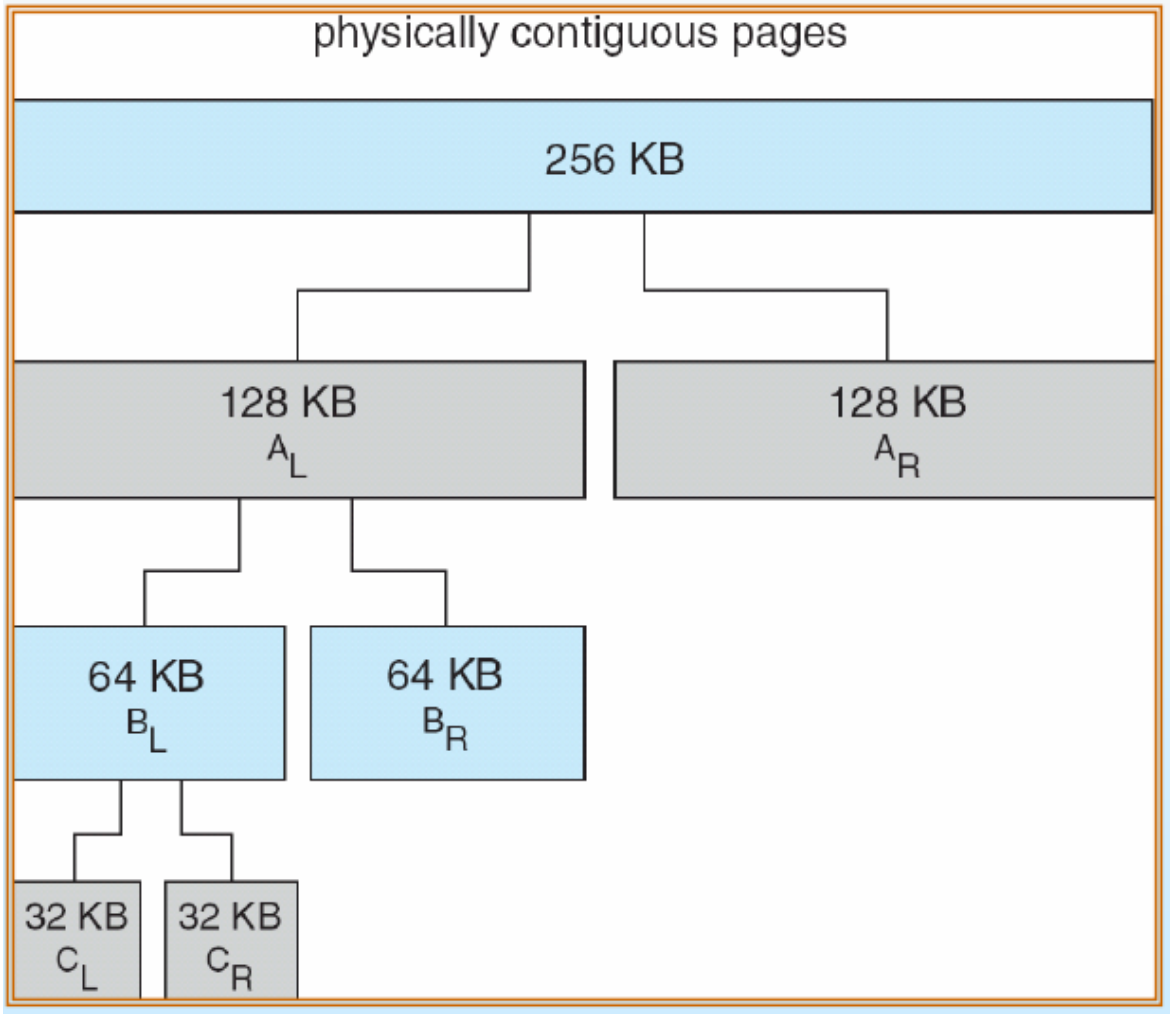
Allocating Kernel Memory

- Treated differently from user memory
- Often allocated from a free-memory pool
  - Kernel requests memory for structures of varying sizes
  - Some kernel memory needs to be contiguous

Buddy System

- Allocates memory from fixed-size segment consisting of physically- contiguous pages
- Memory allocated using power-of-2 allocator
  - Satisfies requests in units sized as power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available

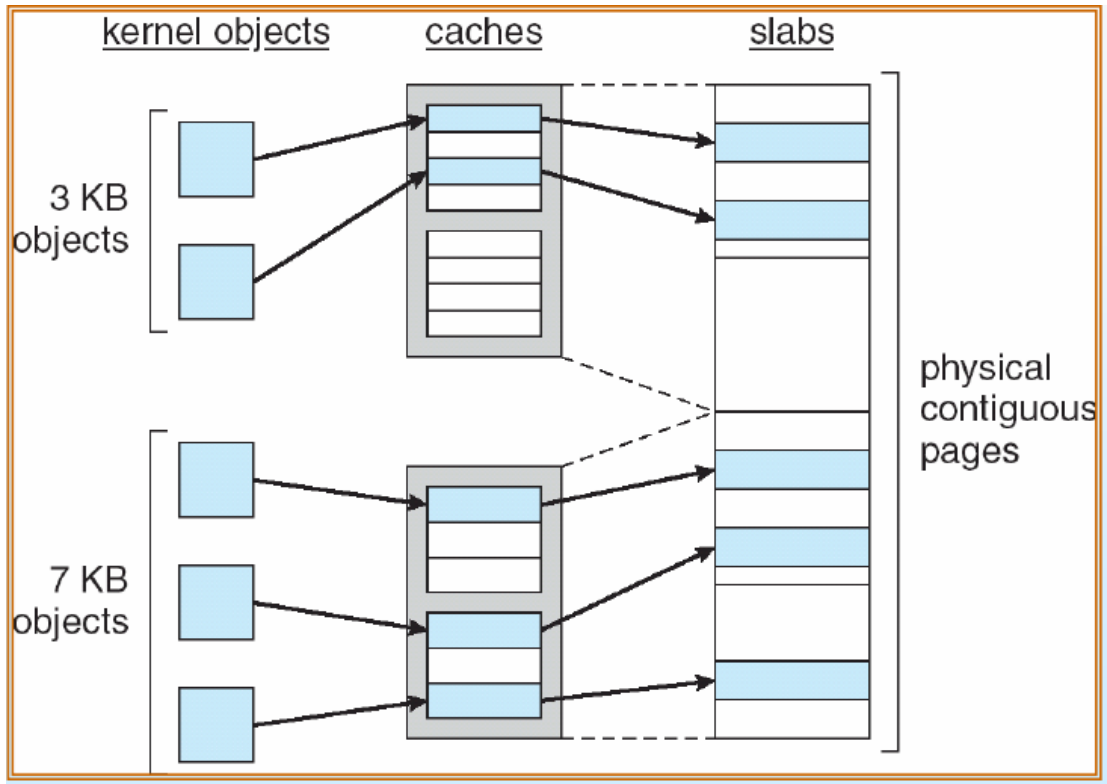
Buddy System Allocator



Slab Allocator I

- Alternate strategy
- Slab is one or more physically contiguous pages
- Cache consists of one or more slabs
- Single cache for each unique kernel data structure
  - Each cache filled with objects – instantiations of the data structure
- When cache created, filled with objects marked as free
- When structures stored, objects marked as used
- If slab is full of used objects, next object allocated from empty slab
  - If no empty slabs, new slab allocated
- Benefits include no fragmentation, fast memory request satisfaction

Slab Allocator II



Other Issues I

1. Prepaging
   - To reduce the large number of page faults that occurs at process startup
   - Prepage all or some of the pages a process will need, before they are referenced
   - But if prepaged pages are unused, I/O and memory was wasted
   - Assume s pages are prepaged and α of the pages is used
     - Is cost of $s * \alpha$ save pages faults $>$ or $<$ than the cost of prepaging $s * (1 - \alpha)$ unnecessary pages?
     - α near zero $\Rightarrow$ prepaging loses

2. Page Size
   - Page size selection must take into consideration:
     - fragmentation
     - table size
     - I/O overhead
     - locality

Other Issues II

3. TLB Reach - The amount of memory accessible from the TLB
   - TLB Reach = $(TLB\ Size) \times (Page\ Size)$
   - Ideally, the working set of each process is stored in the TLB
     - Otherwise there is a high degree of page faults
   - Increase the Page Size
     - This may lead to an increase in fragmentation as not all applications require a large page size
   - Provide Multiple Page Sizes
     - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation
4. Inverted page tables
5. Program structure
   - Int[128,128] data;
   - Each row is stored in one page

Other Issues III

- Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

  128 x 128 = 16,384 page faults

- Program 2

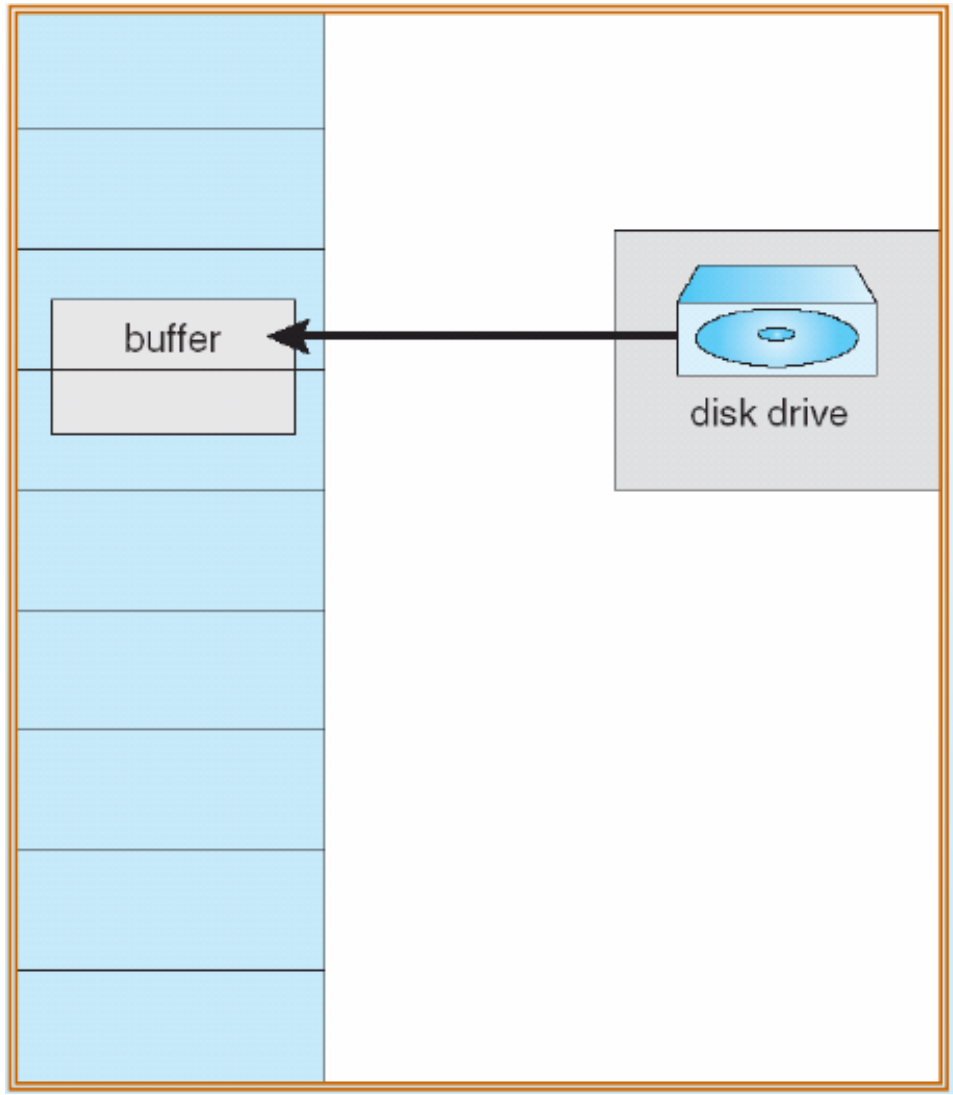
```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

  128 page faults

6. I/O Interlock – Pages must sometimes be locked into memory
   - Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm

Other Issues IV

- Reason Why Frames Used For I/O Must Be In Memory



Operating System Examples

- Windows XP
- Solaris

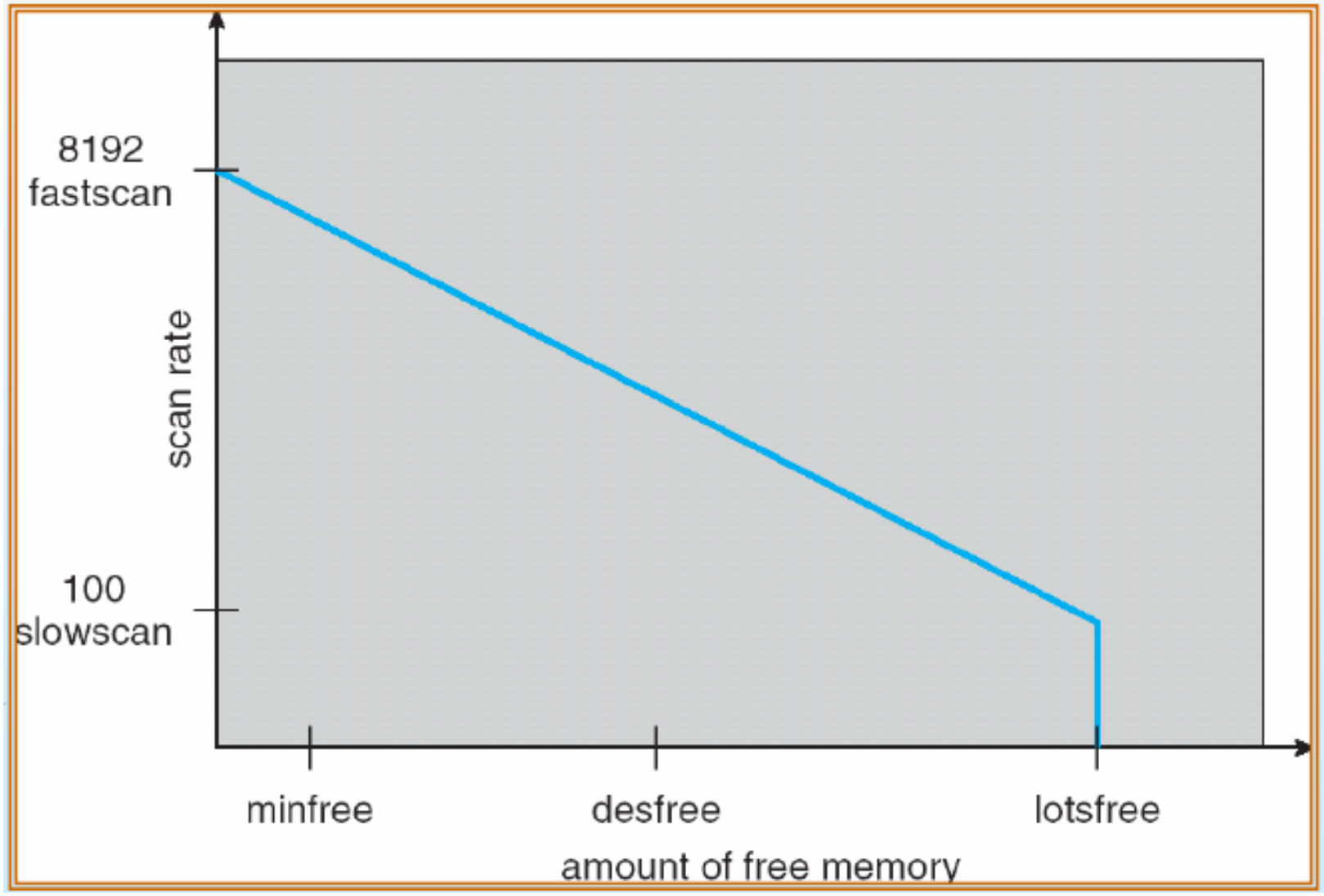
Windows XP

- Uses demand paging with clustering. Clustering brings in pages surrounding the faulting page.
- Processes are assigned working set minimum and working set maximum
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold, automatic working set trimming is performed to restore the amount of free memory
- Working set trimming removes pages from processes that have pages in excess of their working set minimum

Solaris

- Maintains a list of free pages to assign faulting processes
- Lotsfree – threshold parameter (amount of free memory) to begin paging
- Desfree – threshold parameter to increasing paging
- Minfree – threshold parameter to being swapping
- Paging is performed by pageout process
- Pageout scans pages using modified clock algorithm
- Scanrate is the rate at which pages are scanned. This ranges from slowscan to fastscan
- Pageout is called more frequently depending upon the amount of free memory available

Solaris 2 Page Scanner



小结

- Thrashing
  - Cause of trashing
  - Working-Set Model
  - Page-Fault Frequency
- Memory-Mapped Files
- Allocating Kernel Memory
- Other Issues
- Operating System Examples
- 小结和作业

阅读

- Reading 英文9.10，9.11.

作业

- 华夏班：9.2, 9.4, 9.10, 9.15
- 非华夏班：10.3, 10.4, 10.10, 10.20

谢谢！