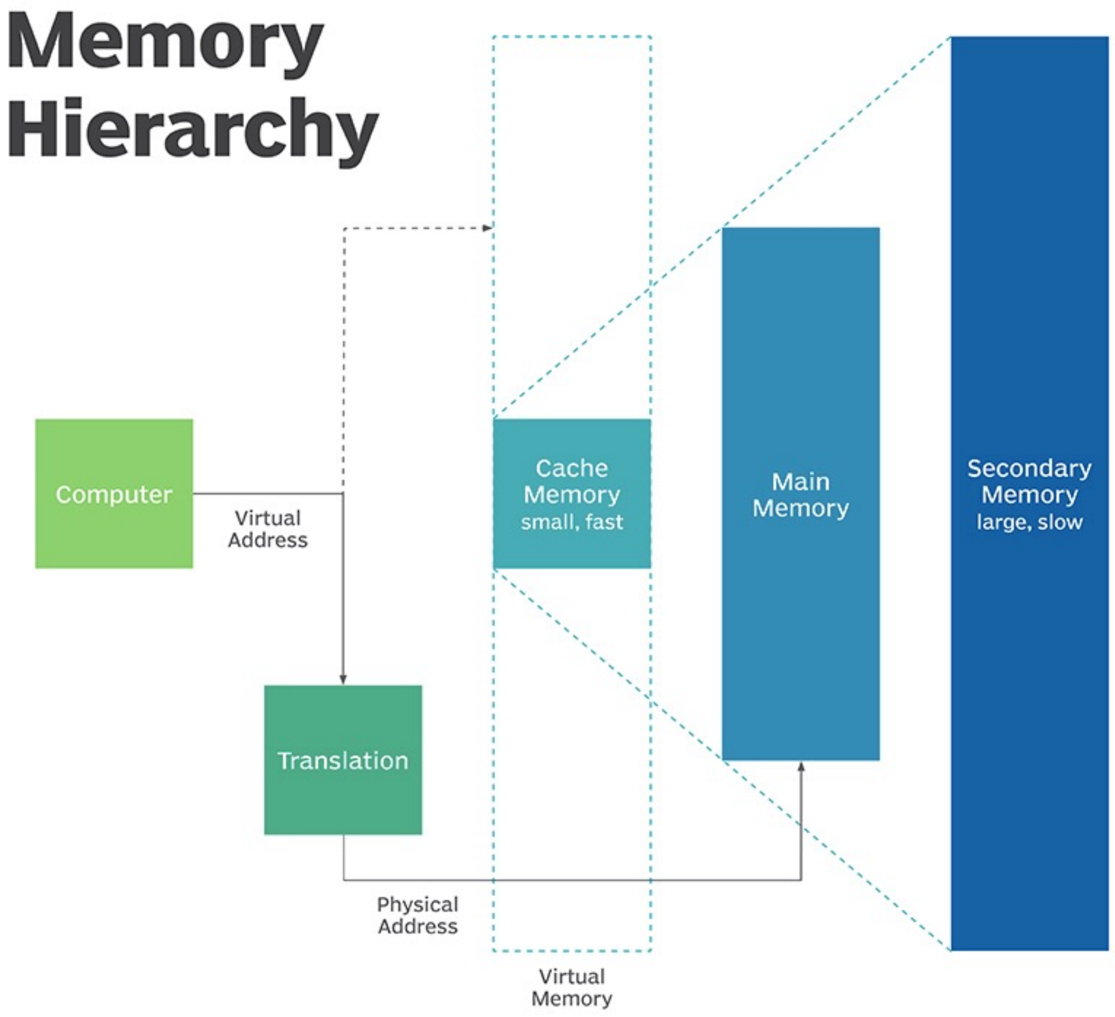
Memory Management (Part 2)

Operating Systems

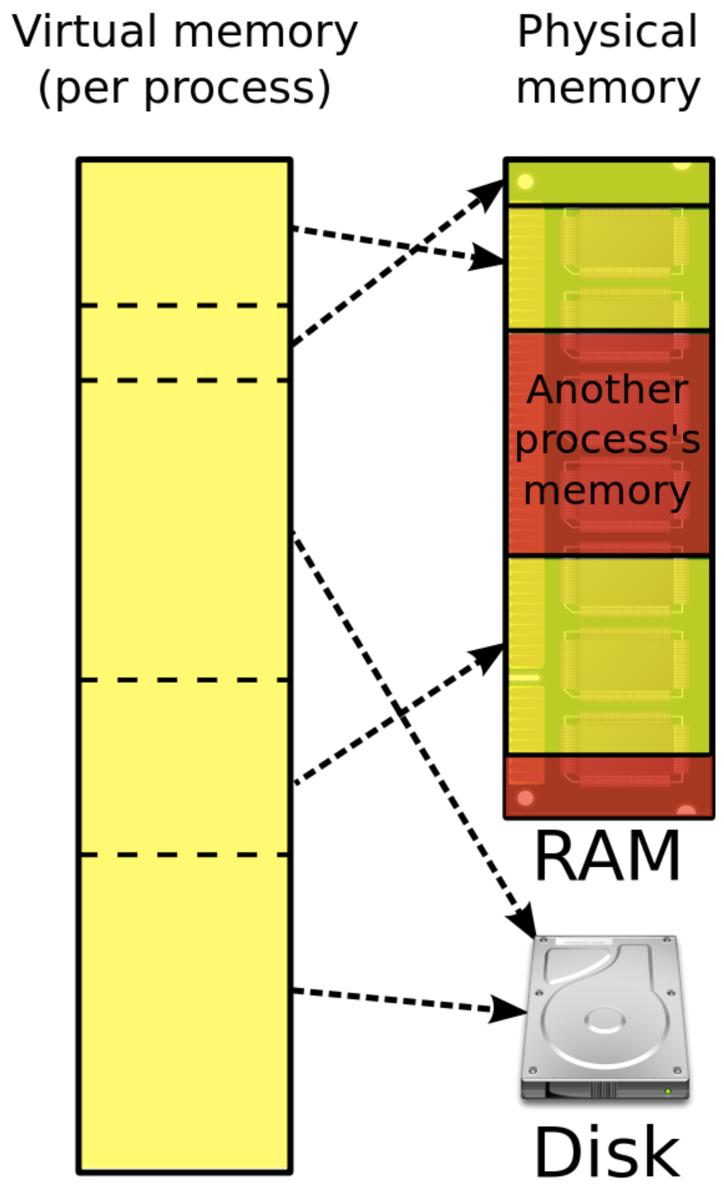
Assoc. Prof. Milos Jovanovik, PhD

Memory Hierarchy



Memory Needs

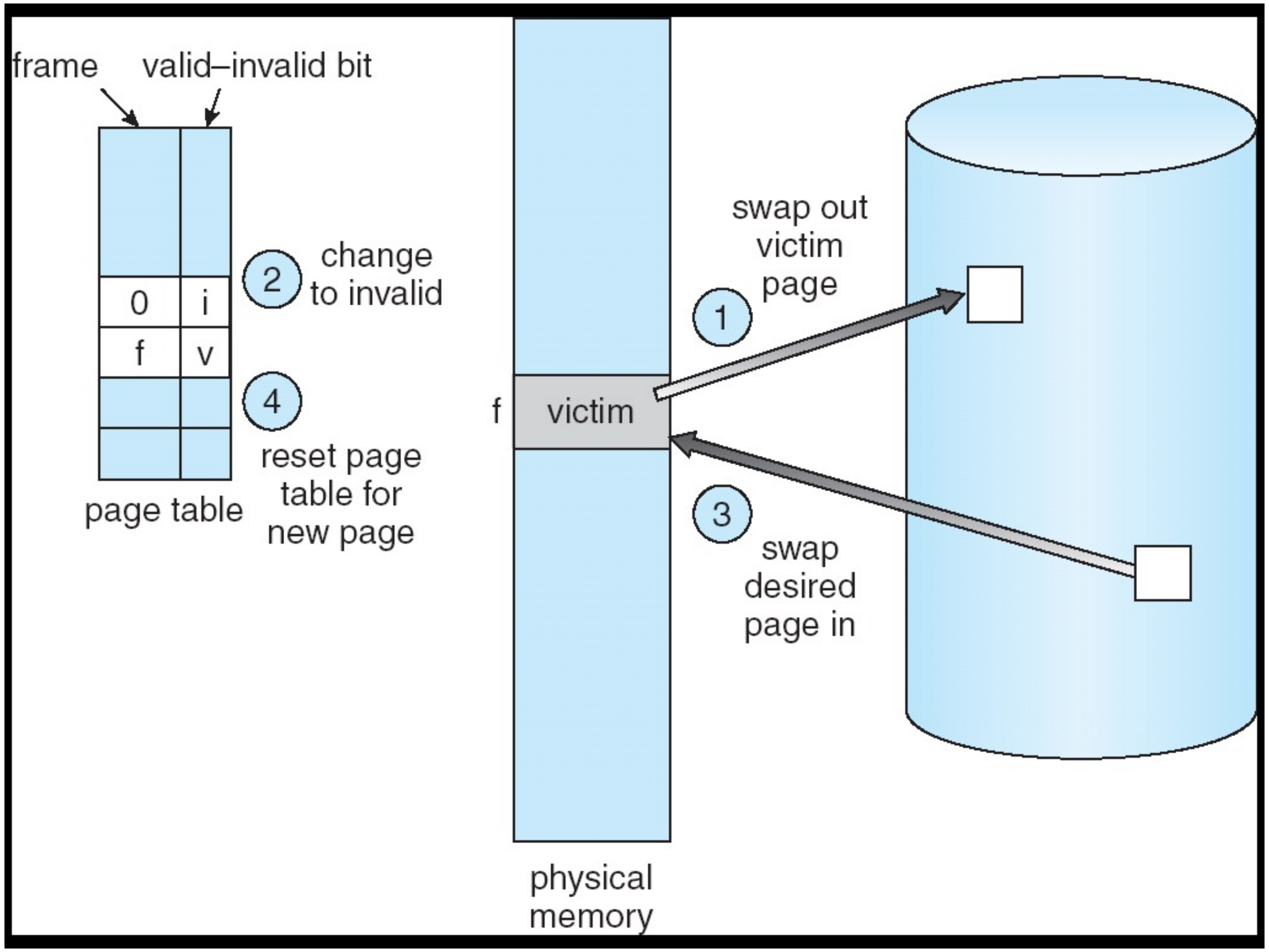
- A need for memory space larger than the physical memory



Page Replacement Algorithm

1. Find the location of the needed page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is not, find a **victim** frame using the page replacement algorithms
3. Read the page in the new, free frame. Update the page table
4. Continue the process

Page Replacement



Page Replacement Algorithms

- When there is a Page Fault, a victim page should be chosen in order to make space for the new page
- Modified pages must be saved to disk
- An algorithm for optimal page replacement should choose the page that will be needed in the most distant future
    - It is optimal, but impossible;

Page Replacement Algorithms

- Not recently used page replacement
- First-In, First-Out page replacement
- Second chance page replacement
- Clock page replacement
- Least recently used page replacement
- Working set page replacement
- WSClock page replacement

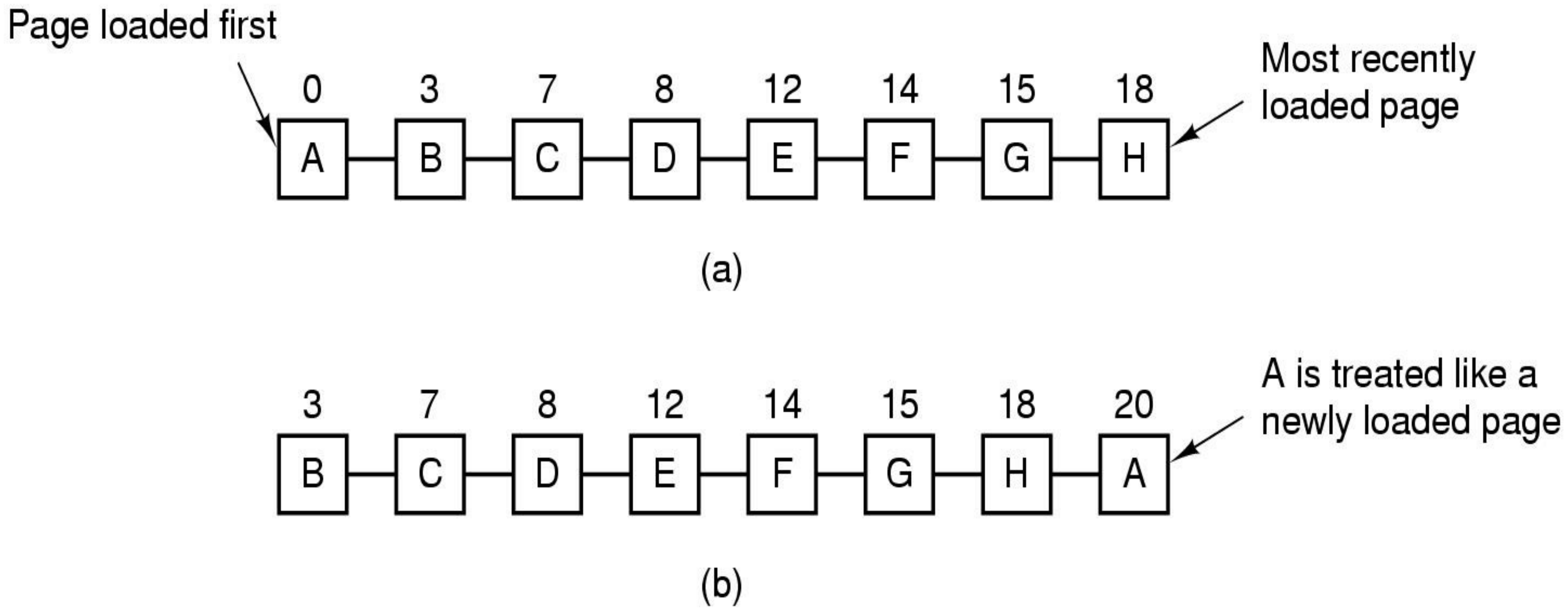
Page Replacement Algorithms: Not Recently Used (NRU)

- For each page there is a Referenced bit and a Modified bit
  - They are set when the page is accessed or modified
  - Periodically, the Referenced bits are erased (reset)
- Pages are classified in several classes:
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
- NRU replaces the pages randomly, starting from the lowest not-empty class

Page Replacement Algorithms: FIFO

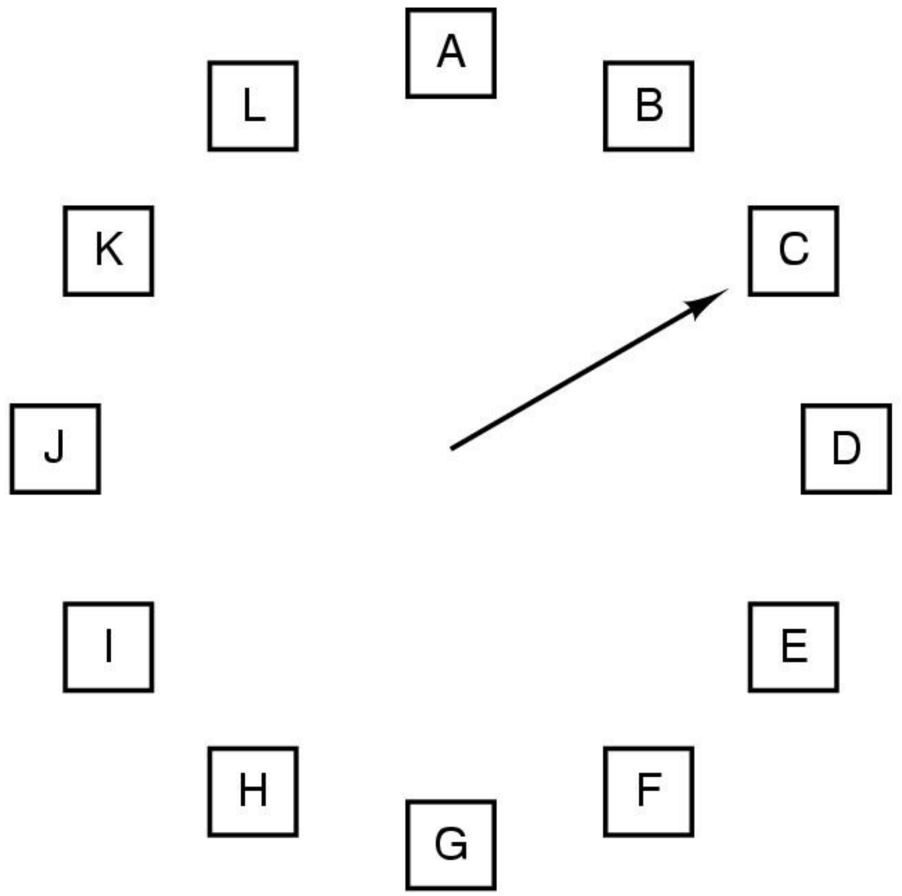
- The OS keeps a linked list of all pages
  - They are sorted according to the time at which they come into the memory
  - Tail – most recent page
  - Head – least recent page
- The algorithm replaces the page that is in the beginning of the list
- Disadvantage
  - The page that is the oldest (in memory) could be the most used page, too

Page Replacement Algorithms: Second Chance



- Pages are sorted according FIFO
  - When there is a page fault, the algorithm checks if the R bit is set. If it is, the page is put at the end and the algorithm tries to replace the next page with R=0

Page Replacement Algorithms: Clock



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page
R = 1: Clear R and advance hand

Page Replacement Algorithms: Least Recently Used (LRU)

- The assumption is that the pages that have been recently used, will be used again
  - Pages that have not been used for the longest time, are replaced
- Solution 1
  - Keep a sorted list of pages based on the number of accesses
  - The list should be updated at each memory access
- Solution 2
  - Each page has a counter, which is incremented on each page access
  - Periodically the counter is reset

Solution 3: LRU using Bit Matrices

Page

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{array}$$

(a)

Page

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}$$

(b)

Page

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}$$

(c)

Page

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{array}$$

(d)

Page

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{array}$$

(e)

$$\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{array}$$

(f)

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}$$

(g)

$$\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{array}$$

(h)

$$\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{array}$$

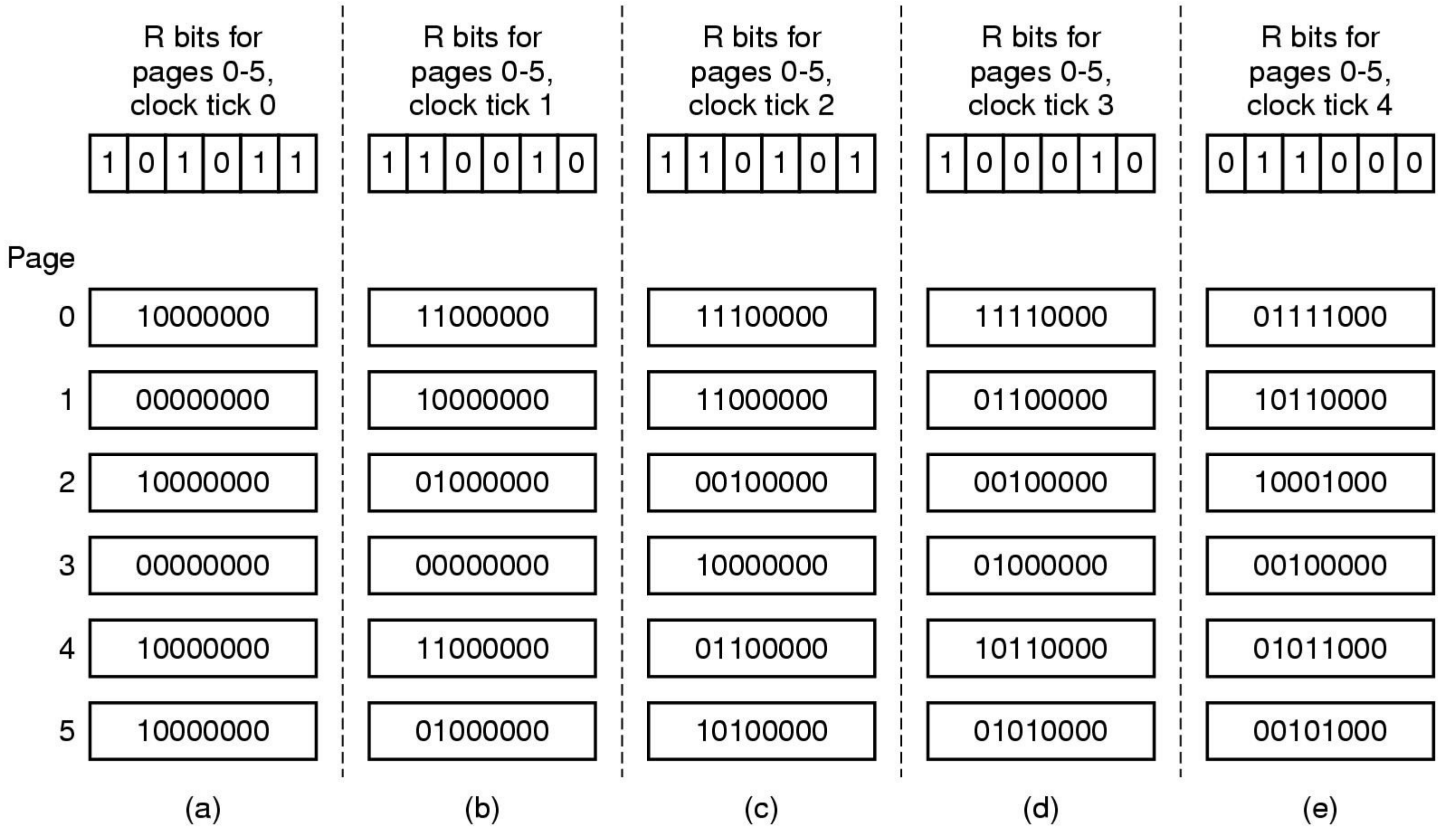
(i)

$$\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{array}$$

(j)

- Pages are referenced in the following order: 0,1,2,3,2,1,0,3,2,3

Page Replacement Algorithms: Simulating LRU – Aging



R bits for pages 0-5, clock tick 0
1 0 1 0 1 1
R bits for pages 0-5, clock tick 1
1 1 0 0 1 0
R bits for pages 0-5, clock tick 2
1 1 0 1 0 1
R bits for pages 0-5, clock tick 3
1 0 0 0 1 0
R bits for pages 0-5, clock tick 4
0 1 1 0 0 0
Page
0
1
2
3
4
5
(a)
(b)
(c)
(d)
(e)

Page	(a)	(b)	(c)	(d)	(e)
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

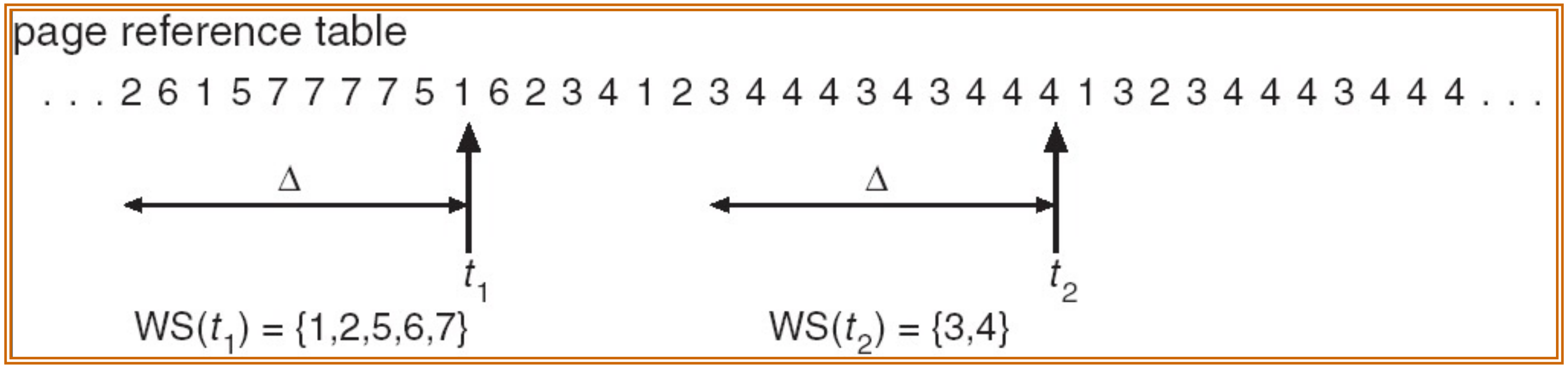
- It measures the age of the pages
- It can be implemented in the software

Working Set

- Processes are started with none of their pages in the memory --> there is a page fault for the first page (instructions, global variables, stack, etc.)
  - Demand paging strategy
- Locality of reference
  - In each moment, the process uses a small number of pages
- Working set is a set of pages which the process can use in a given time
  - If the entire working set is in memory, the process will run without causing many faults, until it moves into another execution phase

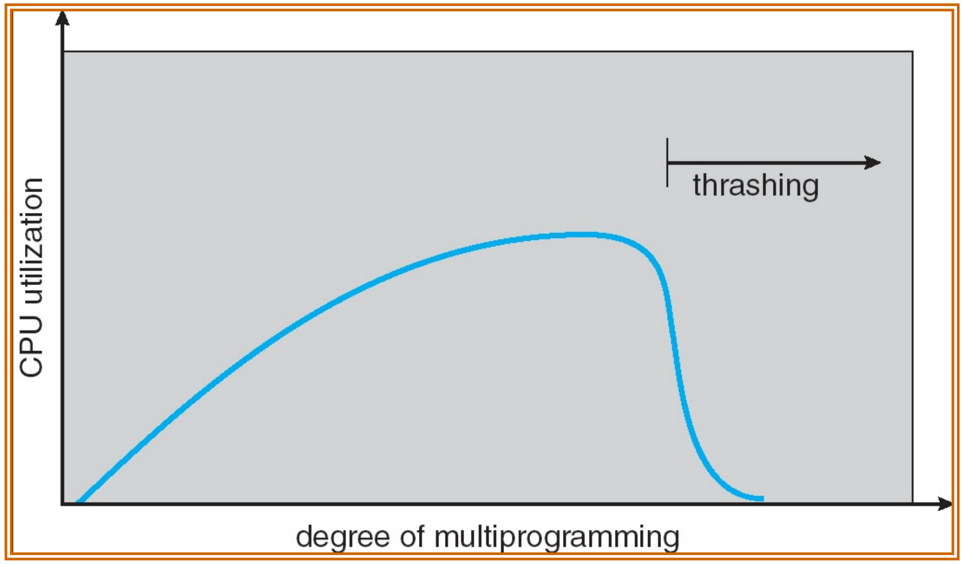
Working Set Model

- $\Delta \equiv$ working set window $\equiv$ fixed number of page references
- Example: Δ=10 references
  - The working sets changes in time depending on the phase of the program



Thrashing

- If the available memory is too small to hold the entire working set, the process will cause many page faults and run slowly
- A program causing page faults every few instructions is said to be **thrashing**



Demand Paging and Thrashing

- Why does demand paging work?
  - Because of the locality model
  - The process migrates from one locality (pages that are currently requested by the process) to another

- Why does thrashing happen?
  - Σ Size of the locality $>$ Size of the memory

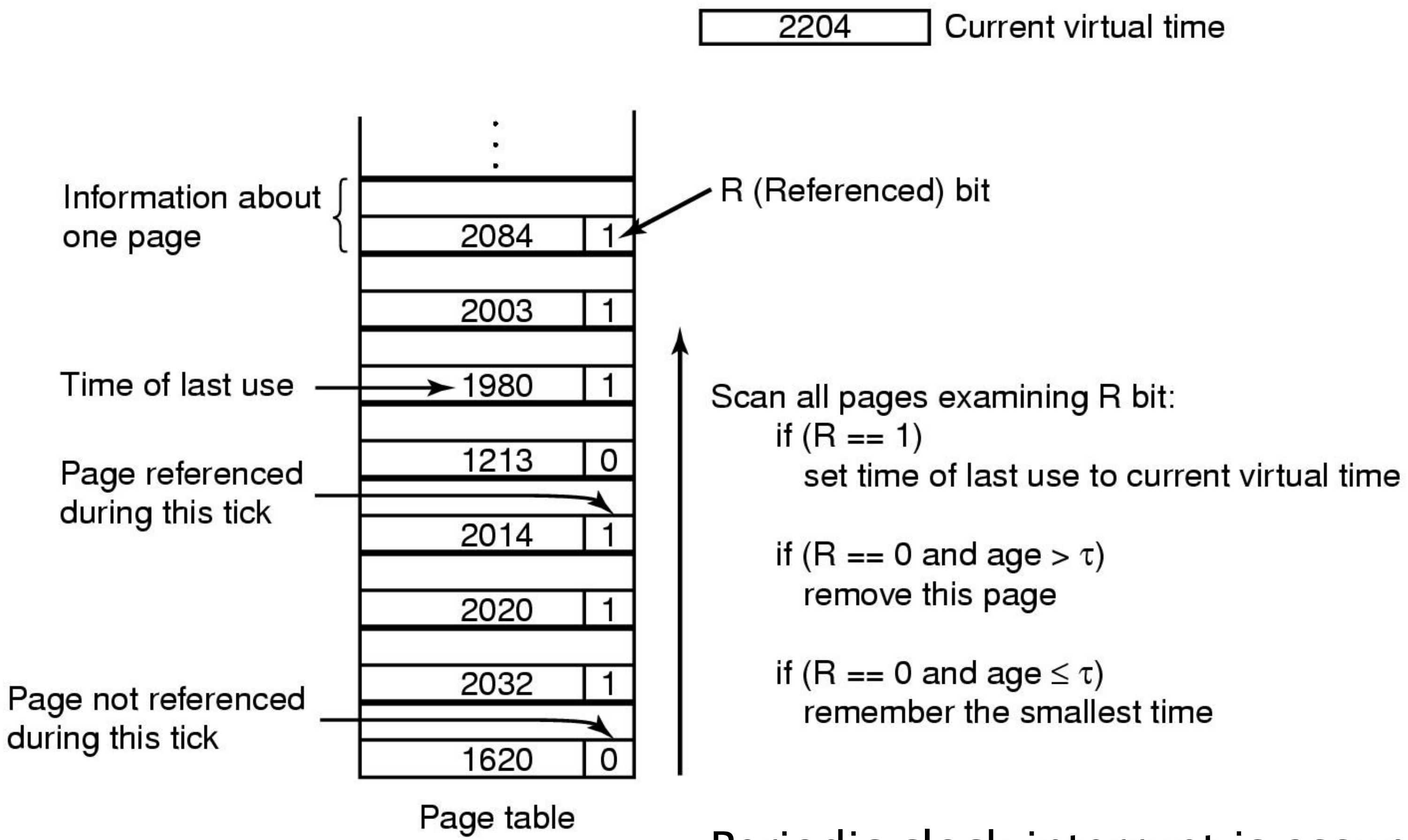
Working Set Model

- WSS_i (Working Set Size of Process P_i) = total number of pages referenced in the last window Δ (it changes)
  - If Δ is too small it does not keep the whole locality
  - If Δ is too big it will keep more localities
  - If $\Delta = \infty \Rightarrow$ it will contain the entire program
- $D = \Sigma\ WSS_i \equiv$ Total number of demanded frames
- If $D >$ m (total number of available frames) $\Rightarrow$ Thrashing
  - Policy: if $D >$ m, suspend one process

Working Set Model

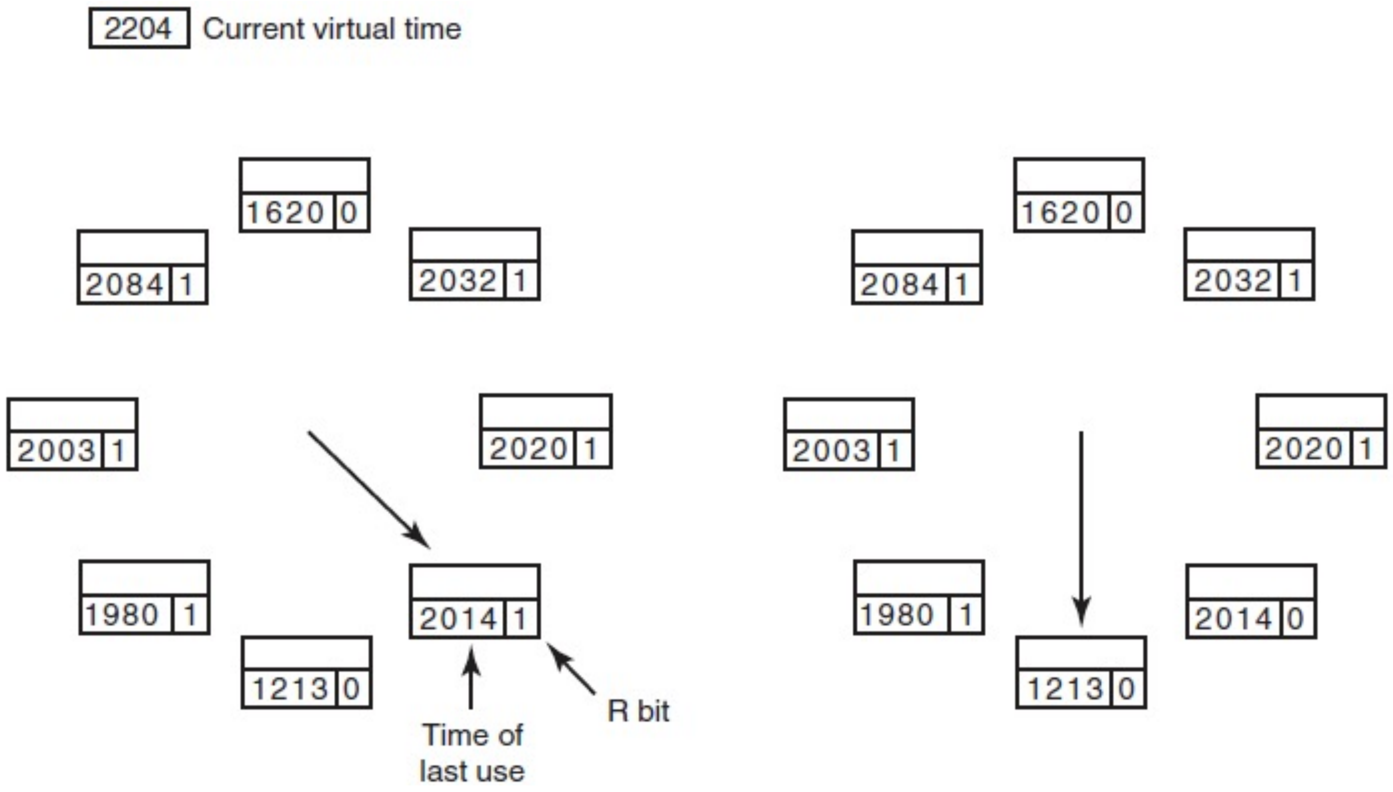
- It is necessary for the operating system to keep track of which pages are in the working set
- It is expensive to maintain the list of the working set pages on every page reference
- Approximation: the working set is defined as a set of pages used during a certain execution interval
- The amount of CPU time a process has actually used since it started, is used as the current virtual time
- Replacement algorithm: find a page that is not in the working set and evict it

Page Replacement Algorithm: Working Set


2204
Current virtual time
Information about one page
2084
1
R (Referenced) bit
2003
1
Time of last use
1980
1
1213
0
Page referenced during this tick
2014
1
2020
1
2032
1
Page not referenced during this tick
1620
0
Page table
Scan all pages examining R bit:
if (R == 1)
set time of last use to current virtual time
if (R == 0 and age > τ)
remove this page
if (R == 0 and age ≤ τ)
remember the smallest time

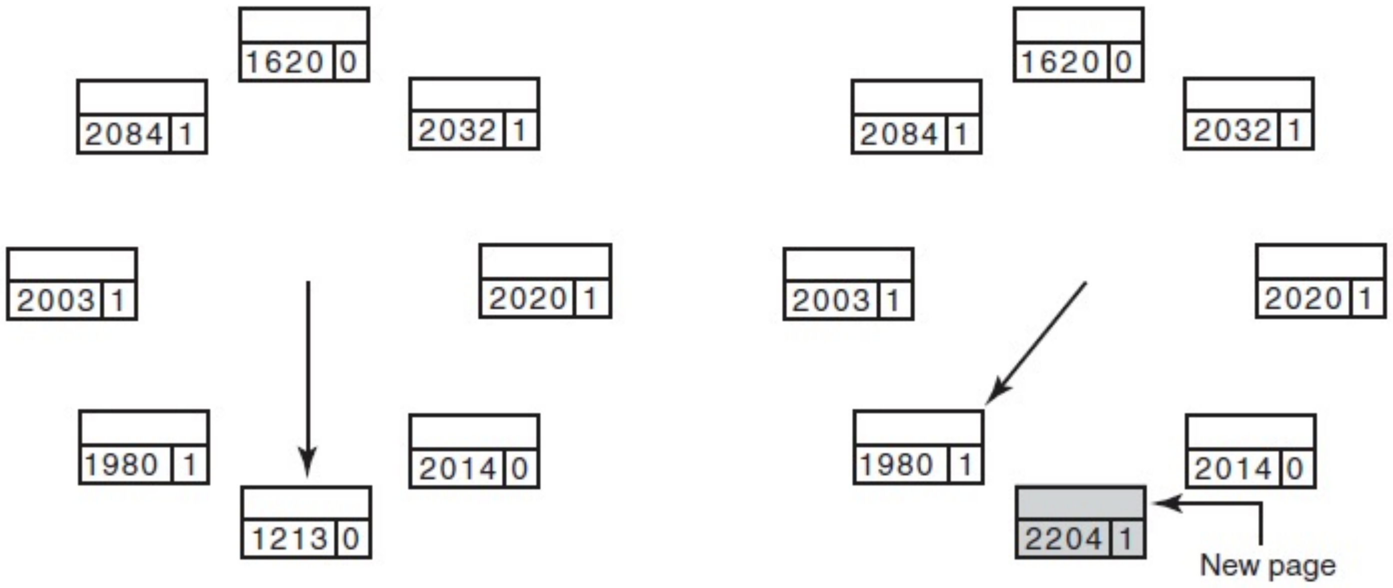
- Periodic clock interrupt is assumed to cause clearing of the R bit on every clock tick.
- On each page fault, the algorithm is executed
- τ is multiple clock ticks
- If all pages are in the working set, the one with the greatest age is evicted.
- If all pages have R = 1 a page is randomly chosen

Page Replacement Algorithm: WSClock



- The first page that satisfies the aging criterion is replaced

Page Replacement Algorithm: WSClock



- The first page that satisfies the aging criterion is replaced

Comparison of the Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Design Issues for Paging Systems

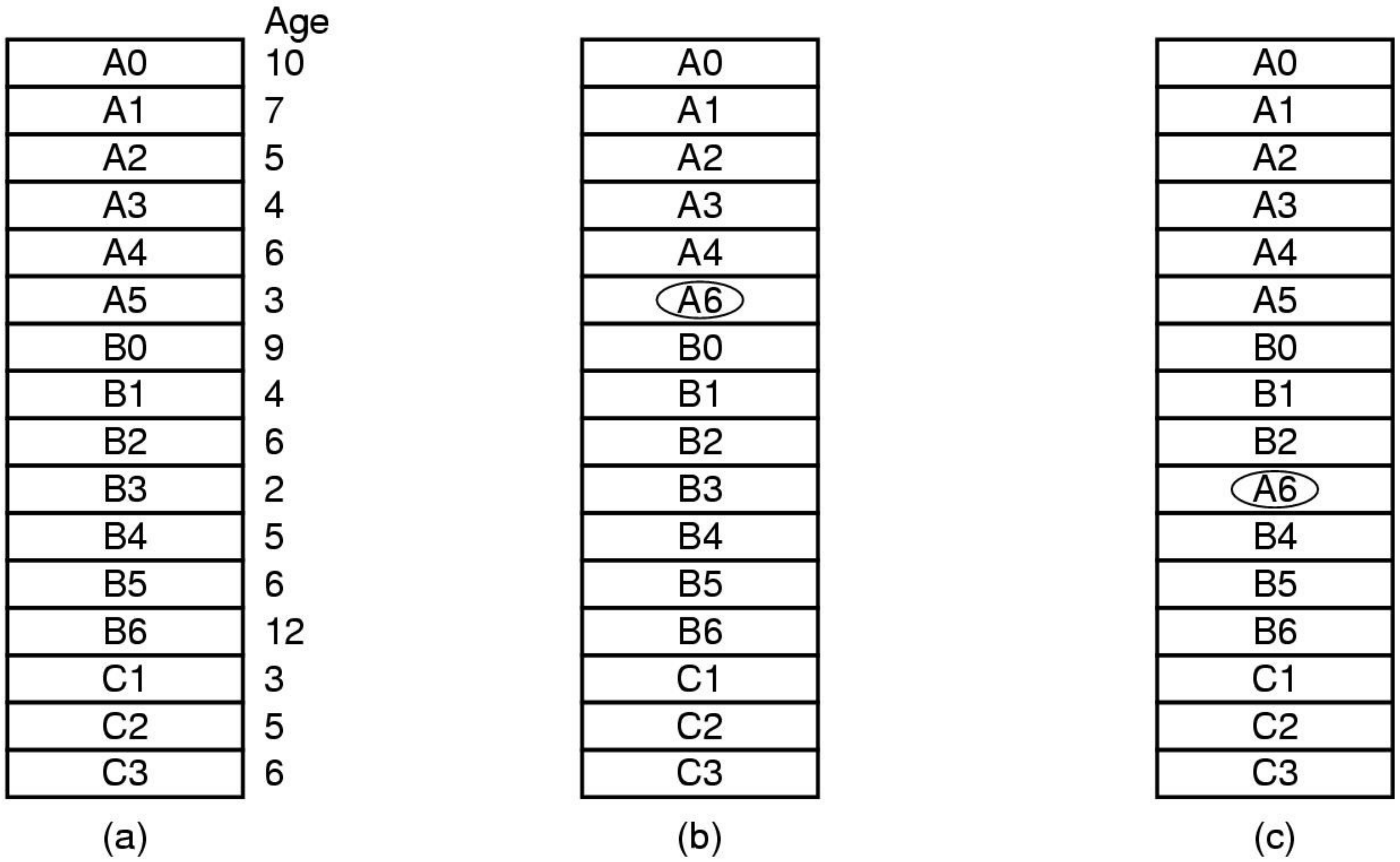
- Global and local replacement
- Load control
- Page size
- Separation of the address spaces for data and programs
- Shared pages
- Shared libraries

Design Elements of the Paging System

- Local replacement
  - Each process is allocated a fixed space in the memory
  - The process can replace only its own pages

- Global replacement
  - Dynamically allocates replacement between all the running processes
  - The number of frames assigned to each process is susceptible to change

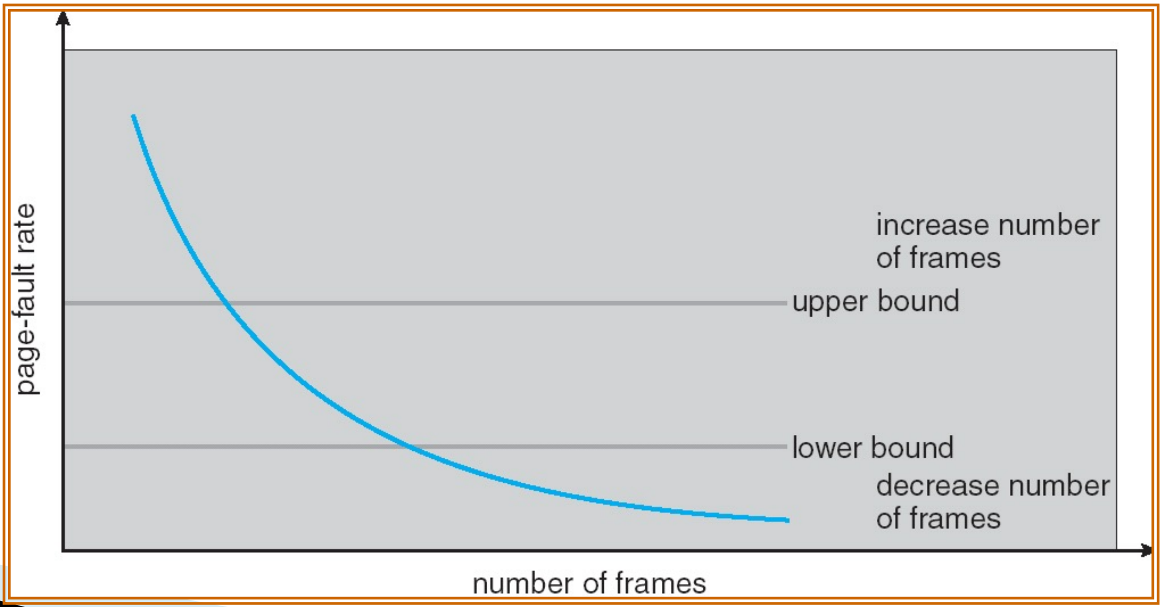
Global and Local Replacement



(a)	Age	(b)	(c)
A0	10	A0	A0
A1	7	A1	A1
A2	5	A2	A2
A3	4	A3	A3
A4	6	A4	A4
A5	3	A6	A5
B0	9	B0	B0
B1	4	B1	B1
B2	6	B2	B2
B3	2	B3	A6
B4	5	B4	B4
B5	6	B5	B5
B6	12	B6	B6
C1	3	C1	C1
C2	5	C2	C2
C3	6	C3	C3

Page Fault Frequency

- Used for global replacement
- Examines the page fault rate for each process
- Each process is assigned an initial number of frames
- The idea is to find the “acceptable” page fault rate
  - If it is too small, it uses too many frames (a frame is taken from the process)
  - If it is high, the process receives a frame



Load Control

- Even though well designed, the systems can get stuck in continuous page replacement (thrashing)
- PFF algorithm shows that:
  - Some processes seek memory
  - No other process needs less memory (can release memory)
- Solution:
  - Decrease the number of processes in memory
  - Swap some of them to the disk and free up all the pages they are holding

Page Size

Small pages

- Advantages
  - Smaller fragmentation
  - Smaller parts of the programs that are not used will be in the memory
- Disadvantages
  - Huge number of pages => big page table

Page Size

- The overhead for one process would be:

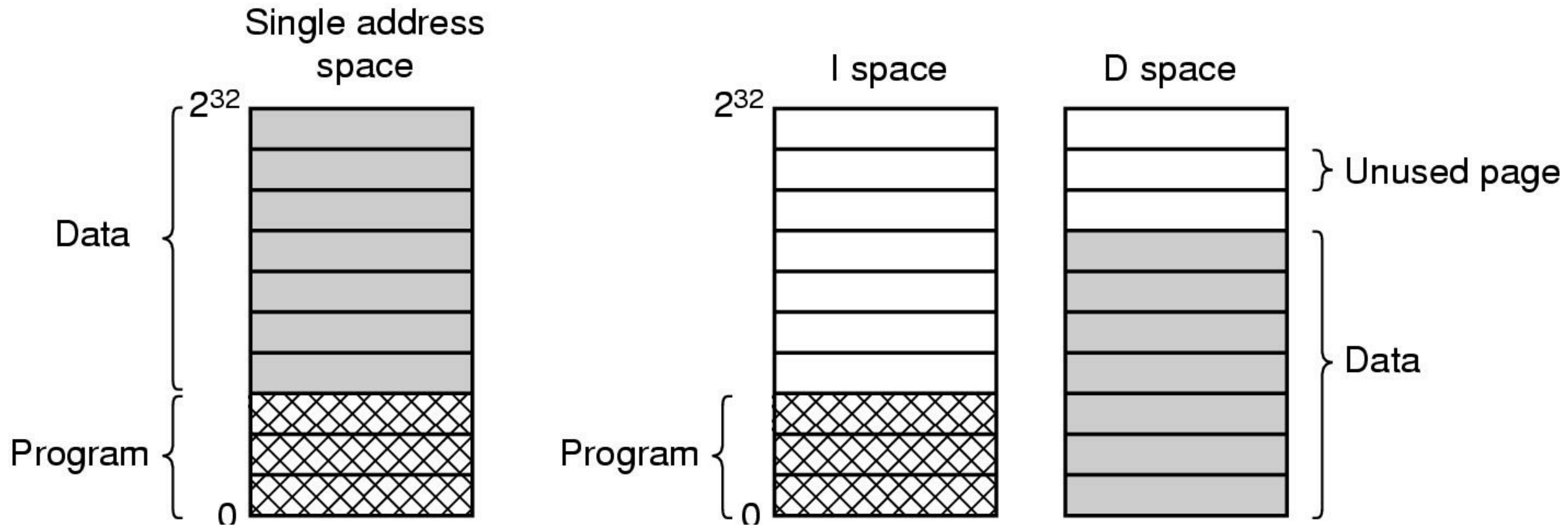
$$overhead = \overbrace{\frac{s}{p}\cdot e}^{\text{page table space}} + \underbrace{\frac{p}{2}}_{\text{internal fragmentation}}$$

- Where
  - s = process size in bytes
  - p = page size in bytes
  - e = size of the entry in the page table in bytes

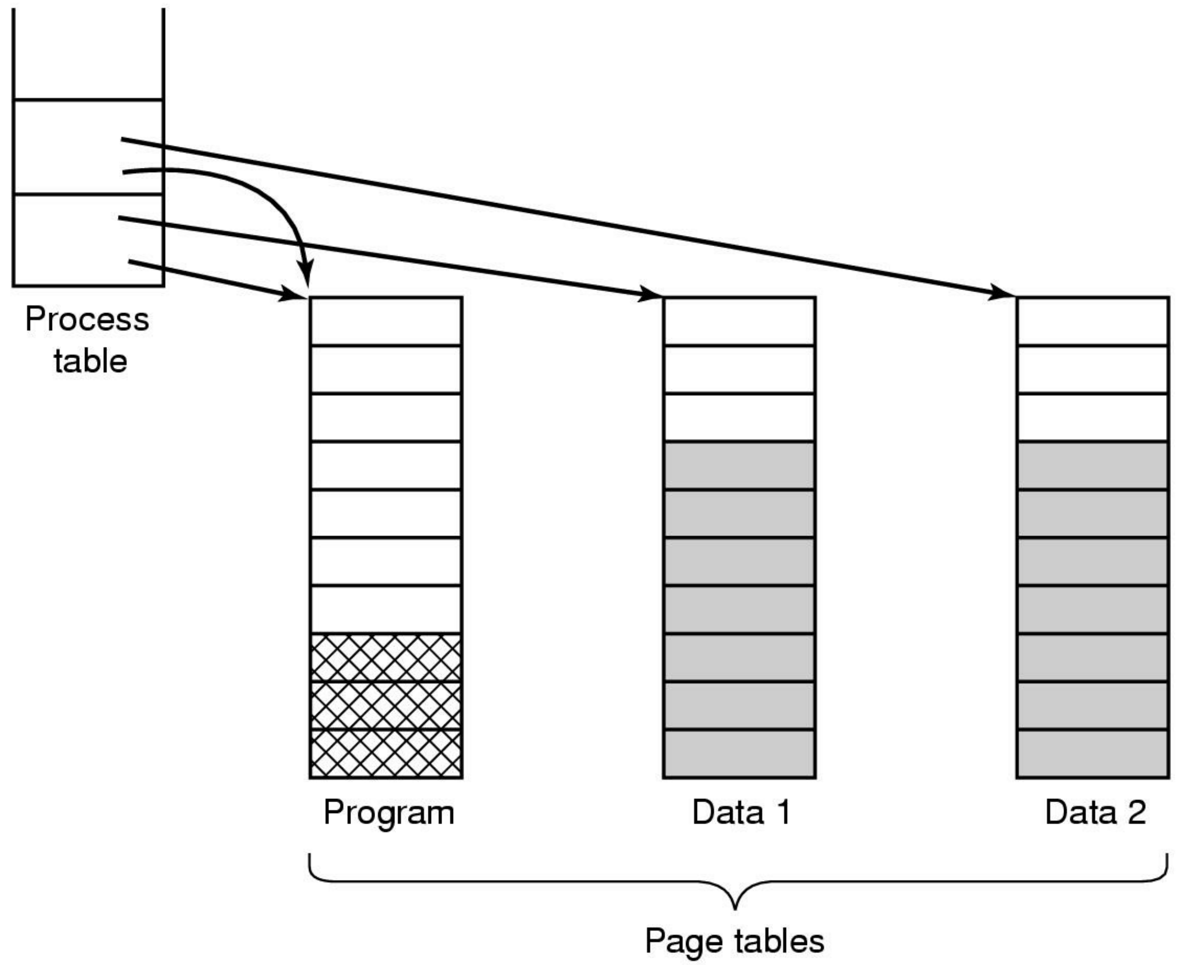
Optimal value:

$$p = \sqrt{2se}$$

Separation of the Address Spaces in Data and Programs



Shared Pages



More processes are using the same pages

Shared Libraries

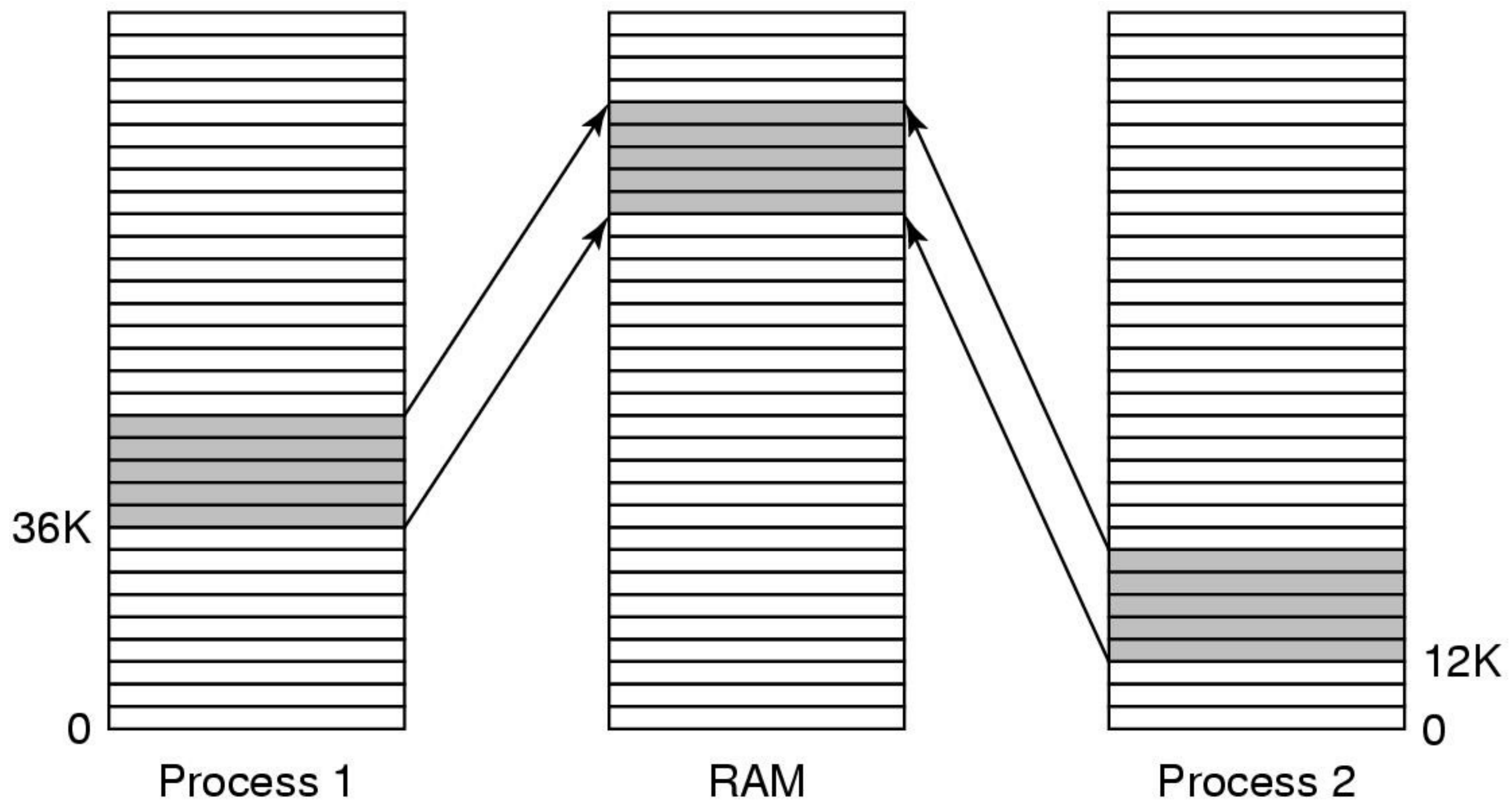


Figure 3-27. A shared library being used by two processes.

Operating System Involvement with Paging

- Moments when paging is used:

1. Process creation
   - determine program and data size
   - create page table
2. Process execution
   - MMU reset for new process
   - TLB flushed
3. Page fault time
   - determine virtual address causing fault
   - swap target page out, needed page in
4. Process termination time
   - release page table, pages

Page Fault Handling (1)

- The hardware traps to the kernel, saving the program counter on the stack.
- An assembly code routine is started to save the general registers and other volatile information.
- The operating system discovers that a page fault has occurred, and tries to discover which virtual page is needed.
- Once the virtual address that caused the fault is known, the system checks to see if this address is valid and the protection consistent with the access. Then, the system checks to see if a page frame is free.

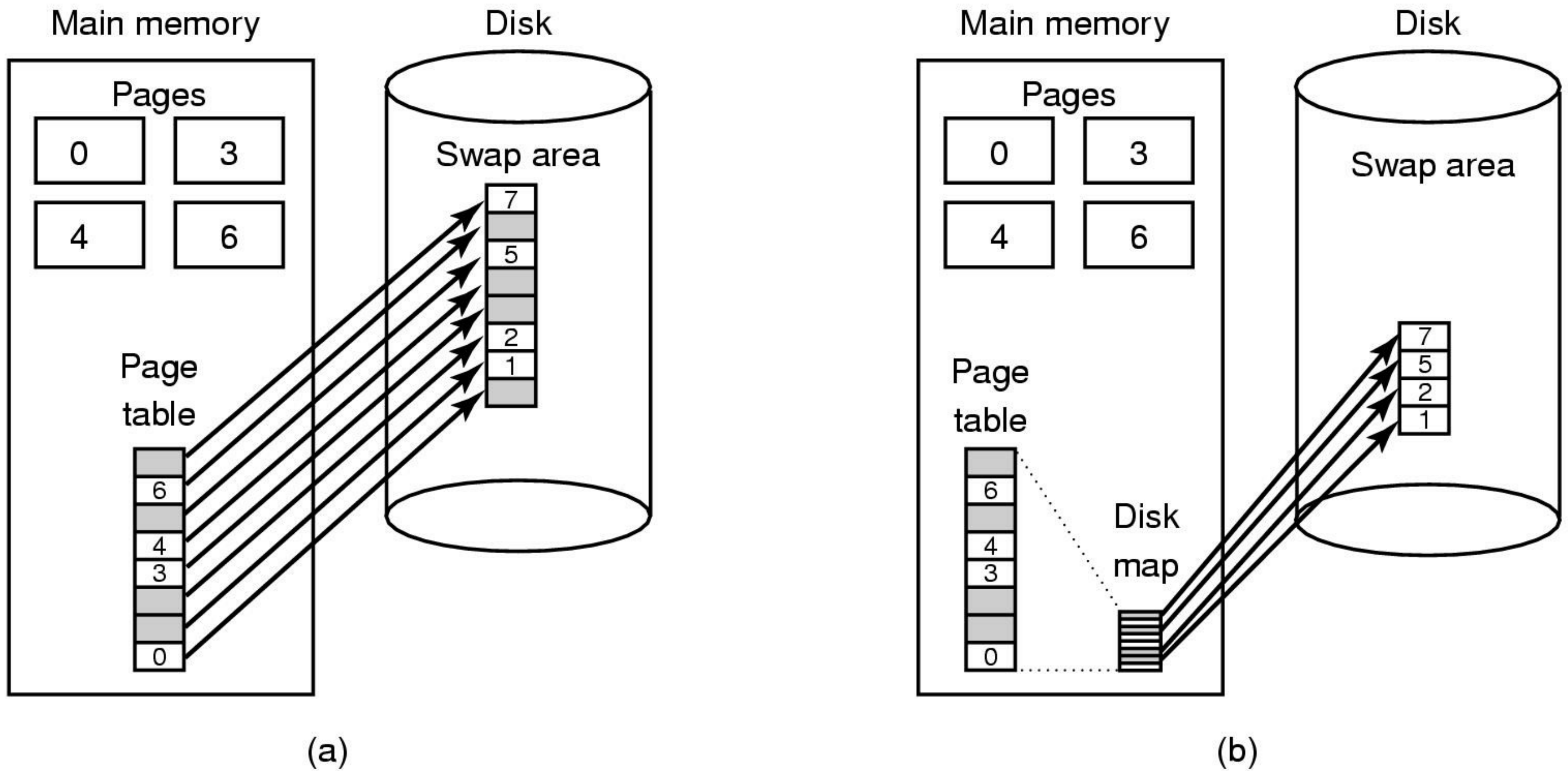
Page Fault Handling (2)

- If the page frame selected is dirty, the page is scheduled for transfer to the disk, and a context switch takes place.
- When a page frame is clean, the operating system looks up the disk address where the needed page is, schedules a disk operation to bring it in.
- When a disk interrupt indicates that the page has arrived, the page table is updated to reflect the position, the frame is marked as being in normal state.

Page Fault Handling (3)

- The faulting instruction is backed up to state it had when it began and the program counter is reset to point to that instruction.
- The faulting process is scheduled for execution, the OS returns to the assembly language routine that called it.
- This routine reloads registers and other state information and returns back to the user space to continue execution, as if no fault had occurred.

Saving Pages to Disk



- (a) Paging to static swap area
- (b) Backing up pages dynamically

Questions?