Chapter 3
Memory Management

- Fetch Strategies
- Page Replacement
  - -Optimal
    -FIFO、Second-chance、Clock
    -NRU、LRU、NFU、Aging
    -Working Set、WSClock
- Design Issues for paging systems
- Implementation Issues for paging systems
- Segmentation
- Segmentation with paging

Paging Policies

- Fetch Strategies
  - When should a page be brought into primary (main) memory from secondary (disk) storage.
- Replacement Strategies
  - Which page now in primary storage is to be removed from primary storage when some other page or segment is to be brought in and there is not enough room.
- Clean Strategies
  - When a page in memory is brought out?

Fetch Strategies: Demand Fetching

- Algorithm Never bring a page into primary memory until it's needed.
  1. Page fault occurs.
  2. Check if a valid virtual memory address. Kill job if not.
  3. If valid reference, check if it's cached in memory already (perhaps for some other process.) If so, skip to 7.
  4. Find a free page frame.
  5. Map address into disk block and fetch disk block into page frame. Suspend user process.
  6. When disk read finished, add vm mapping for page frame.
  7. If necessary, restart process.

Fetch Strategies: Prepaging

- Algorithm bring a page into primary in advance.
- When page-fault happens, the page needed and adjacent pages will be brought into memory.
  - Advantage: improve the I/O efficiency when fetch page.
  - Disadvantage: based on prediction, if the fetched page is rare referenced, low efficiency.
  - Used when loading page.

Page Replacement

1. Find location of page on disk
2. Find a free page frame
   ① If there is a free page frame then use it
   ② Otherwise, **select a victim frame** using the page replacement algorithm
   ③ Write the selected page to the disk if necessary and update any necessary tables
3. Read the requested page from the disk.
4. Restart the user process.

Page Replacement Objectives

- Achieve a low page fault rate (Main Objective)
  - Insure that heavily used pages stay in memory.
  - The replaced page should not be needed for some time.
- Reduce latency of a page fault
  - Replace pages that do not need to be written out

Page Replacement Algorithms

- Reference String
  - Reference string is the sequence of pages being referenced.
  - Example: If user has the following sequence of addresses:
    - 123, 215, 600, 1234, 76, 96
    - If the page size is 100, then the reference string is: 1, 2, 6, 12, 0, 0
- Usage of Reference String
  - Evaluate a page replacement algorithm by running it on a particular string of memory references (reference string).
  - Compute the number of page faults on that string.

Page Replacement Algorithms

- **The Optimal Algorithm (OPT or MIN)**
  - Replace the page that will not be used again the farthest time in the future.
- **Random page replacement Algorithm**
  - Choose a page randomly
- **FIFO (First-in First-Out) Algorithm**
  - Replace the page that has been in primary memory the longest
- **Second Chance Algorithm**
  - Variant of FIFO
- **Clock Algorithm**
  - Better implementation of second chance
- **NRU (Not Recently Used) Algorithm**
  - Enhanced second chance

Page Replacement Algorithms

- **LRU (Least Recently Used) Algorithm**
  - Replace the page that has not been used for the longest time
- **NFU (Not Frequently Used) Algorithm**
  - Replace the page that is used least often, simulated LRU
- **Aging Algorithm**
  - Modified NFU
- **Working Set Algorithm**
  - Keep in memory those pages that the process is actively using
- **WS Clock Algorithm**
  - The modified working set algorithm based on clock algorithm

The Optimal Algorithm

- Description:
  - Assume that each page can be labeled with the number of instructions that will be executed before that page is first references, i.e., we would know the future reference string for a program.
  - Then the optimal page algorithm would choose the page with the highest label to be removed from the memory.
- It can be shown that this algorithm results in the fewest number of page faults.
- Impractical because it needs future references
- If future references are known
  - should not use demand fetching
  - should use pre paging to allow paging to be overlapped with computation.
- This algorithm provides a basis for comparison with other schemes.

Optimal Example

12 references,
7 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

FIFO Page Replacement Algorithm

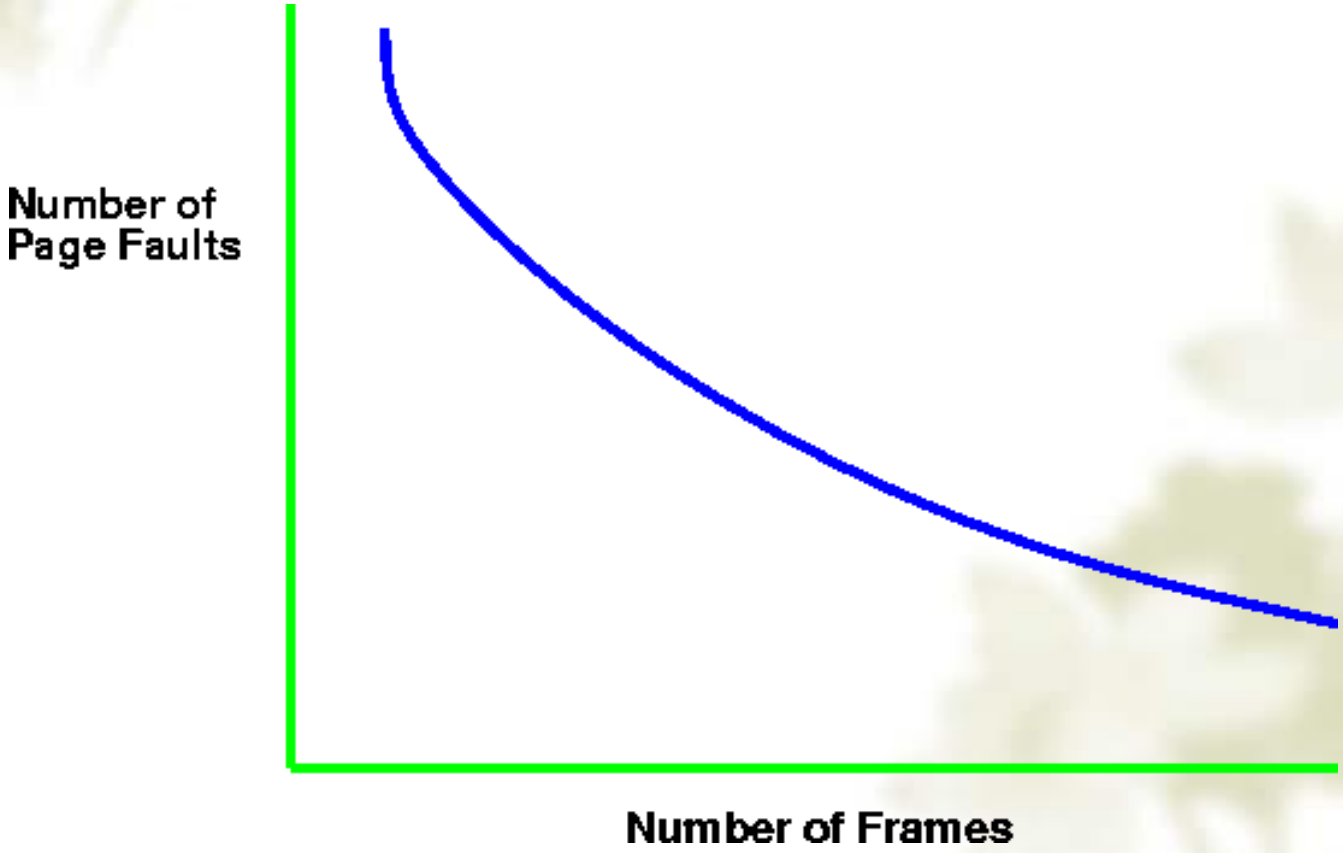
- Maintain a linked list of all pages
  - in order they came into memory with the oldest page at the front of the list.
- Page at beginning of list replaced
- Advantage: easy to implement (FIFO queue)
- Disadvantage
  - page in memory the longest (perhaps often used) may be evicted

FIFO

12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Paging Behavior with Increasing Number of Page Frames



Belady's anomaly

- Belady discovered more page frames might not always have fewer page faults. This is called **Belady's anomaly**.
- A paging system can be characterized by three items:
  - The reference string of the executing process.
  - The page replacement algorithm.
  - The number of page frames available in memory.

Belady's Anomaly (for FIFO)

As the number of page frames increase, so does the fault rate.

12 references,

10 faults

Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

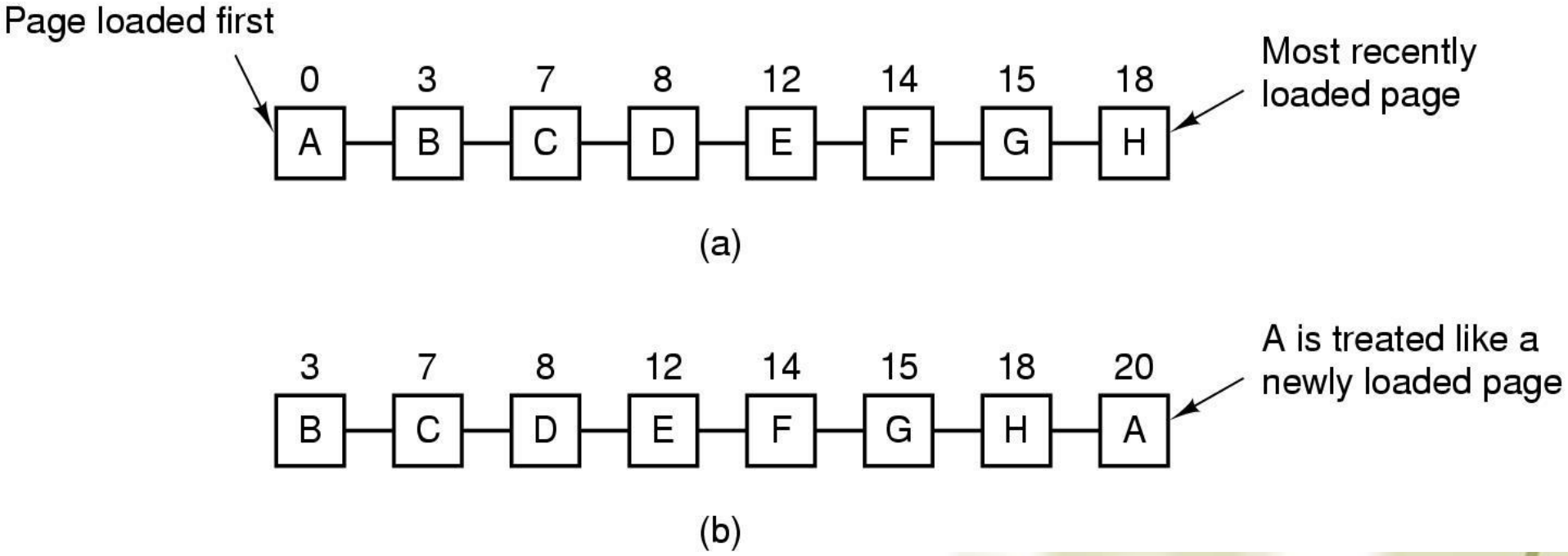
Second Chance Page Replacement Algorithm

- A simple modification of FIFO
  - Inspect R bit

    if R = 0 ➔ evict the page

    if R = 1 ➔ set R = 0 and put page at end (back) of list. The page is treated like a newly loaded page.

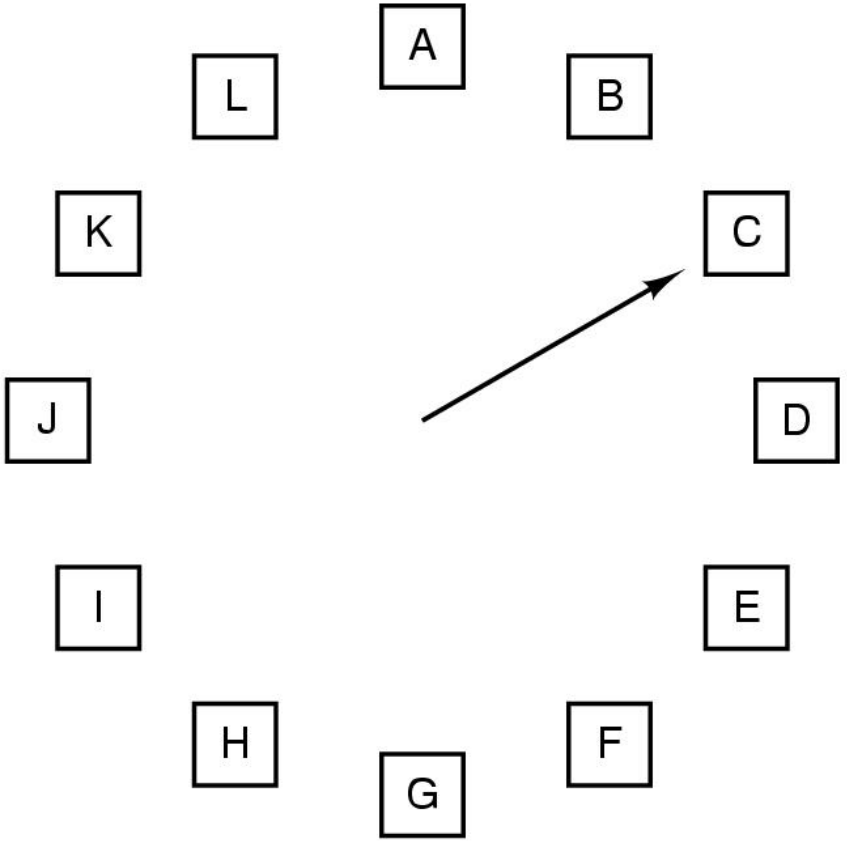
Second Chance Page Replacement Algorithm



- Operation of a second chance
  a) Pages sorted in FIFO order
  b) Page list if fault occurs at time 20, A has *R* bit set (numbers above pages are loading times)

The Clock Page Replacement Algorithm

- Clock Replacement Algorithm: a different implementation of second chance

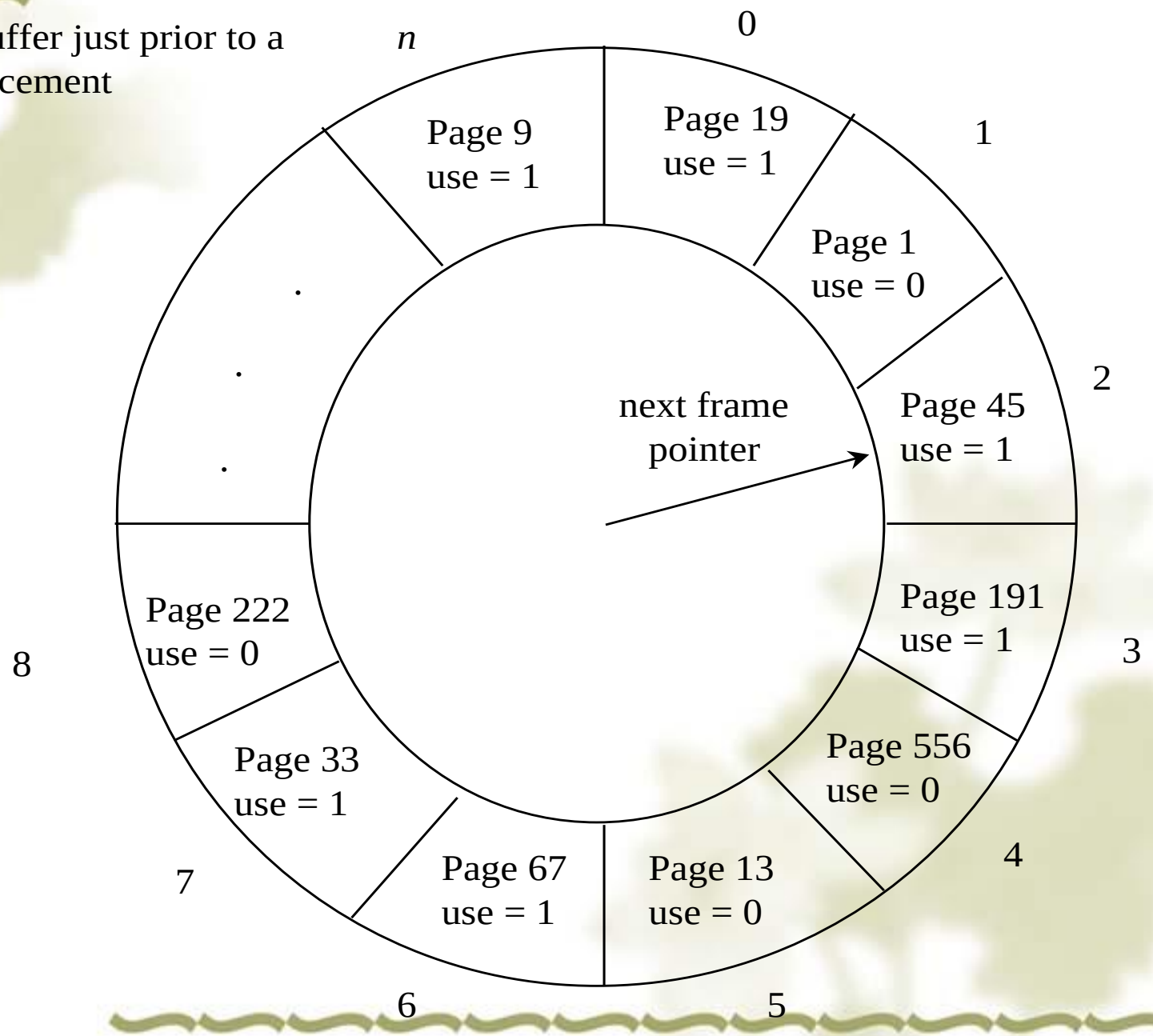


When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

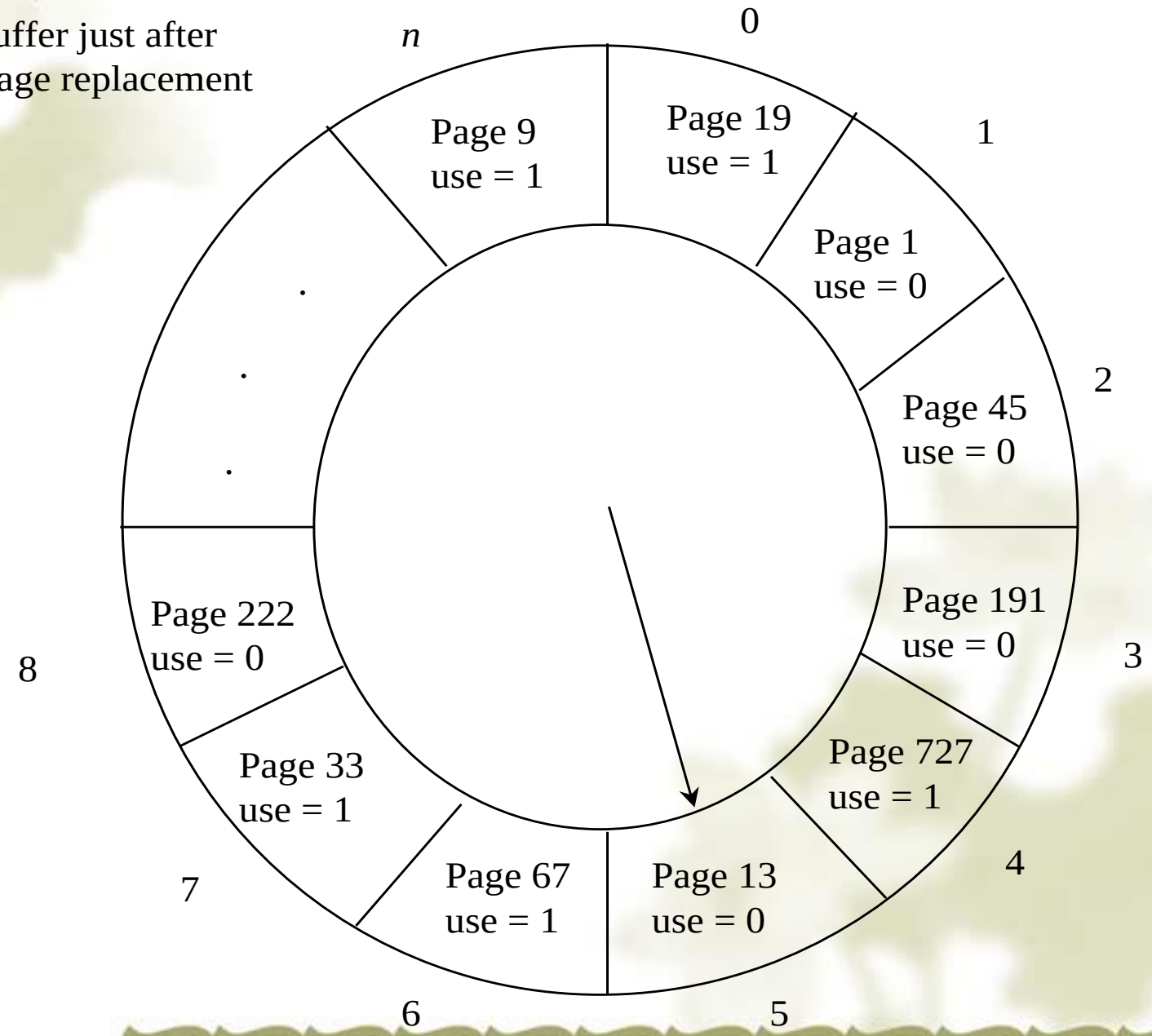
R = 0: Evict the page

R = 1: Clear R and advance hand

State of buffer just prior to a page replacement



State of buffer just after
the next page replacement



Not Recently Used Page Replacement Algorithm (NRU)

- Each page has Reference bit(R) and Modified bit(M).
  - bits are set when page is referenced (read or written recently), modified (written to)
  - when a process starts, both bits R and M are set to 0 for all pages.
  - periodically, (on each clock interval (20msec)), the R bit is cleared. (i.e. R=0).
- Pages are classified

Class 0: not referenced, not modified

Class 1: not referenced, modified

Class 2: referenced, not modified

Class 3: referenced, modified

- NRU removes page at random
  - from lowest numbered non-empty class

Least Recently Used (LRU)

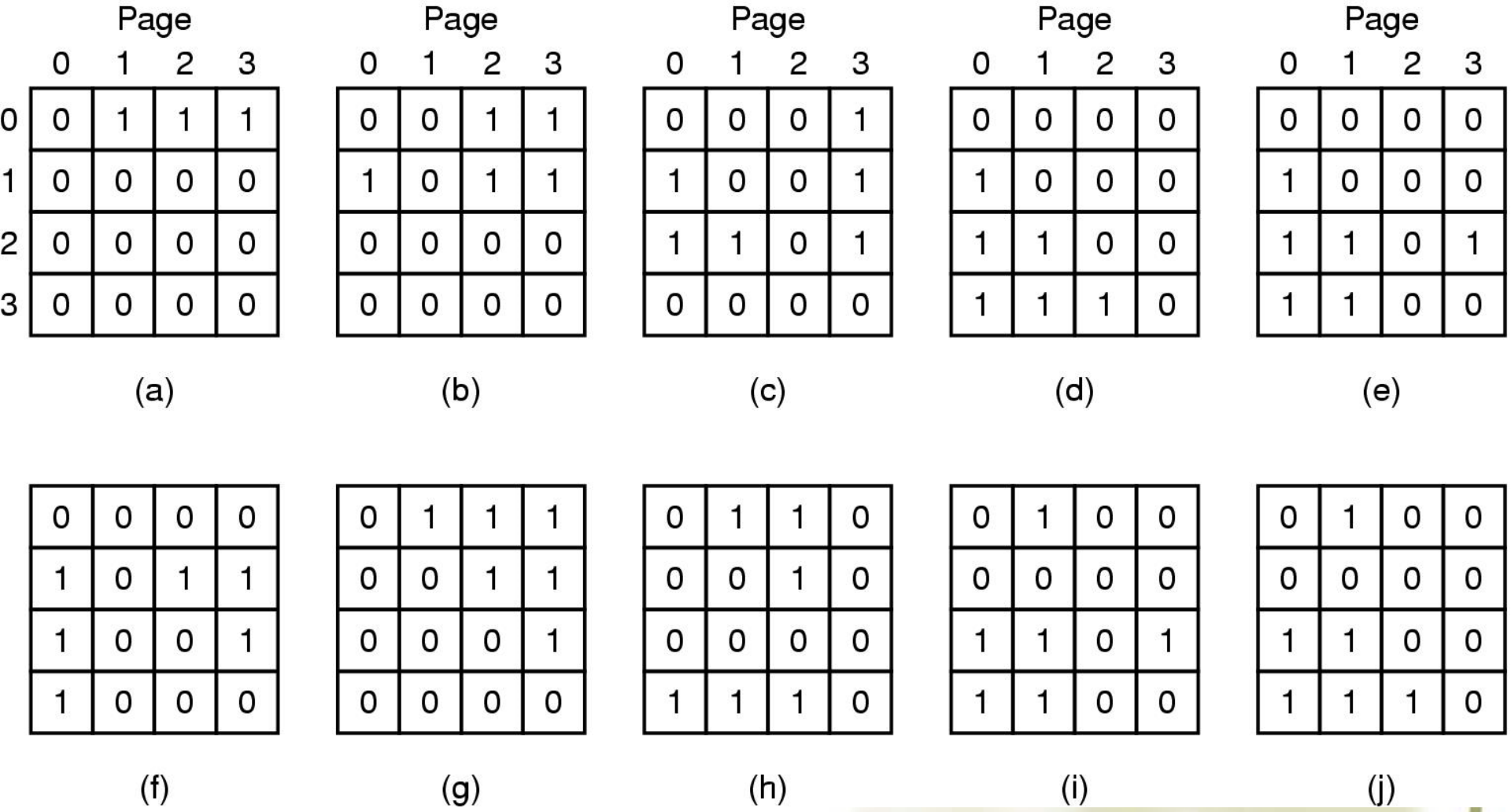
- Assume pages used recently will used again soon
  - throw out page that has been unused for longest time
- Software Solution: Must keep a linked list of pages
  - most recently used at front, least at rear
  - update this list every memory reference ➔ Too expensive!

- Hardware solution 1: Equip hardware with a 64 bit counter
  - That is incrementing after each instruction.
  - After memory reference, the counter value is stored in the page table entry of the page that was just referenced.
  - choose page with lowest value counter
  - periodically zero the counter

Least Recently Used (LRU)

- Hardware solution 2:
  - Maintain a matrix of *n* x *n* bits for a machine with *n* page frames.
    - When page frame *K* is referenced:
      (i) Set row *K* to all 1s.
      (ii) Set column *K* to all 0s.
    - The row whose binary value is smallest is the LRU page.

LRU

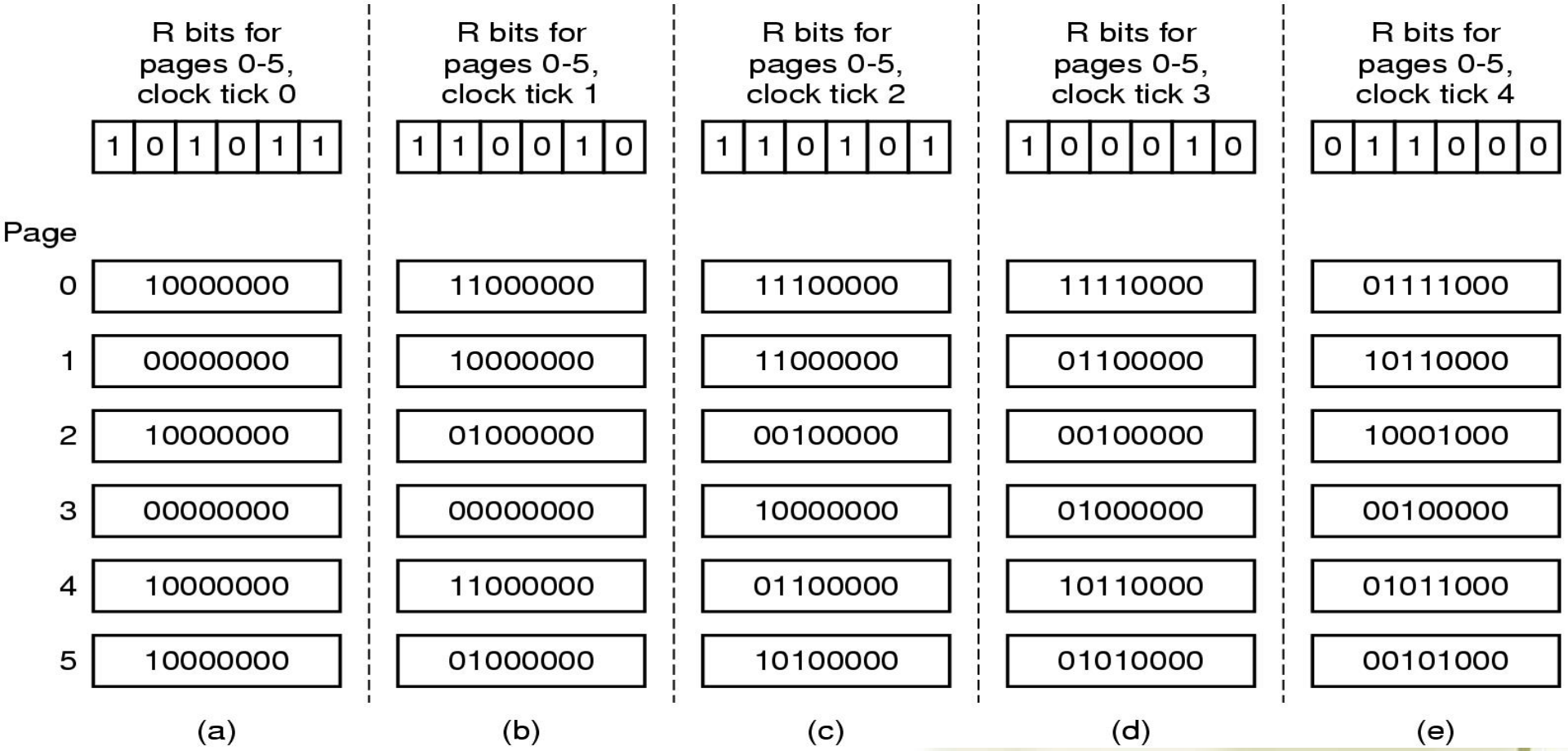


LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3

Simulating LRU in Software

- LRU hardware is not usually available.
- NFU (Not Frequently Used) is implemented in software.
  - At each clock interrupt, the R bit is added to the counter associated with each page. When a page fault occurs, the page with the lowest counter is replaced.
  - Problem: NFU never forgets, so a page referenced frequency long ago may have the highest counter.
- Modified NFU = NFU with Aging - at each clock interrupt:
  - the counters are shifted right one bit, and
  - the R bits are added to the leftmost bit.
  - In this way, we can give higher priority to recent R values.

Simulating LRU in Software



- The aging algorithm simulates LRU in software
- Note 6 pages for 5 clock ticks, (a)~(e)

LRU

12 references,
10 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	A	E	B
B	no	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

LRU and Anomalies

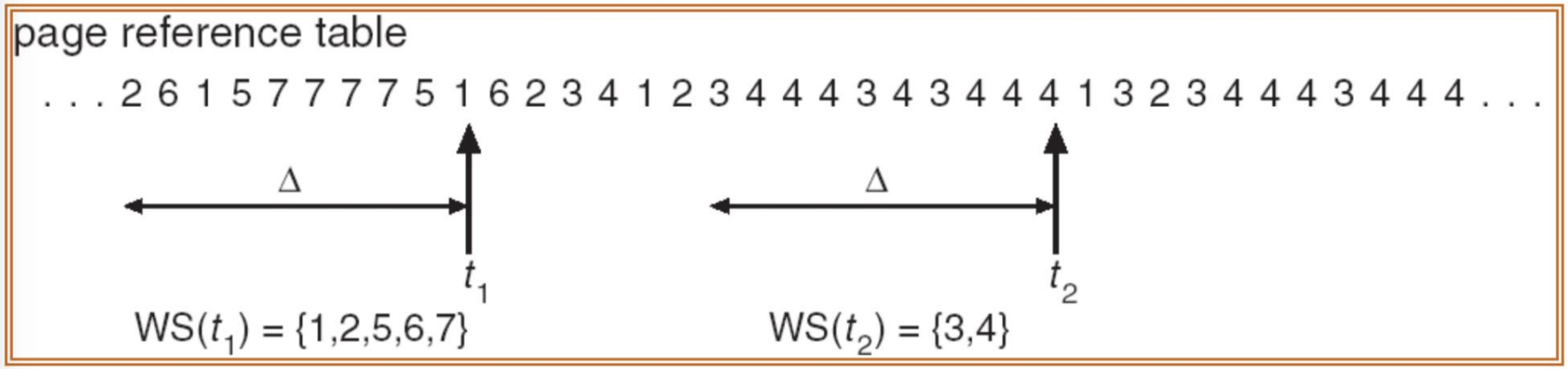
Anomalies cannot occur.

12 references, 8 faults

Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	A	D	C	B
B	no	B	A	D	C
E	yes	E	B	A	D
A	no	A	E	B	D
B	no	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

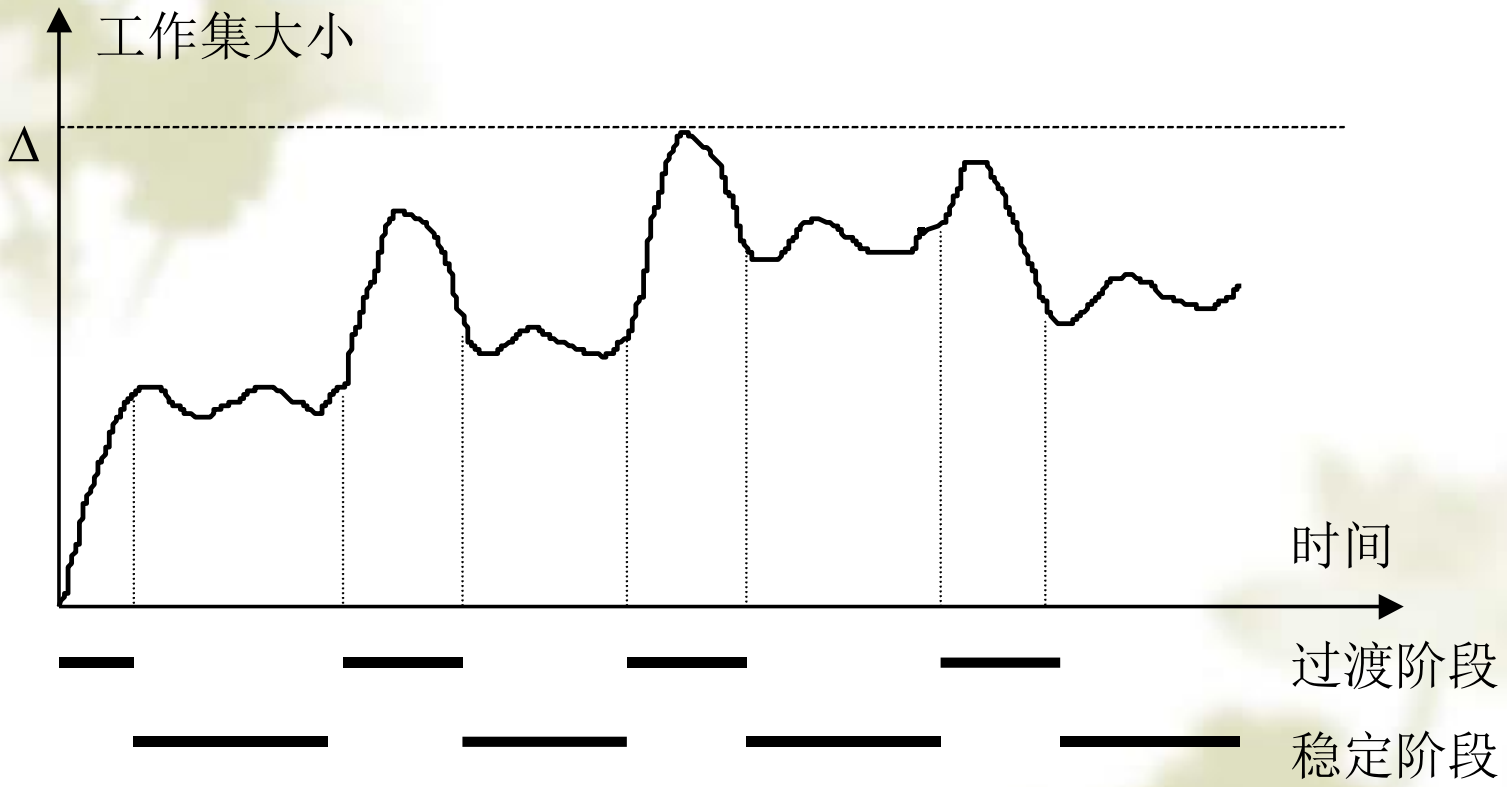
The Working Set Page Replacement Algorithm

- The working set is the set of pages used by the *k* most recent memory references
- w(*k*,*t*) is the size of the working set at time *t*
- Example: *k* or Δ =10



The Working Set Page Replacement Algorithm

- Working set algorithm
  - When a page fault occurs, find a page that is not in the working set and evict it.
- Approximation
  - Drop the idea of counting back *k* memory references and use execution time.
  - The working set of a process is the set of pages it has referenced during the past τ seconds of virtual time.



进程开始执行后，随着访问新页面逐步建立较稳定的工作集。
当内存访问的局部性区域的位置大致稳定时，工作集大小也大致稳定；
局部性区域的位置改变时，工作集快速扩张和收缩过渡到下一个稳定值。

The Working Set Page Replacement Algorithm

- On each clock tick, clear all R bits
- When looking for eviction candidates, scan all pages of process in physical memory
  - If R = 1

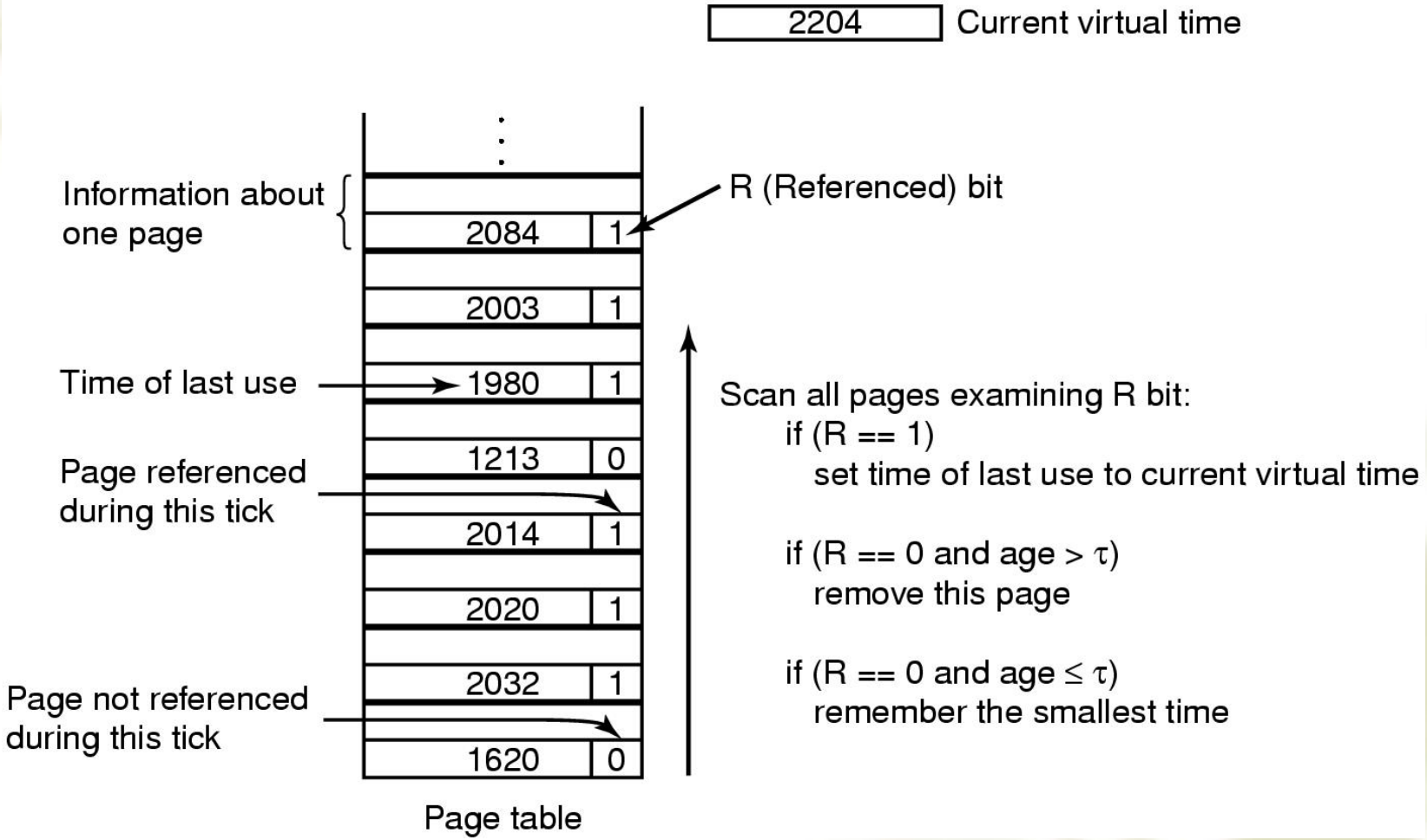
    Store *t* in LTU (last time used) of PTE
  - If R = 0

    If (t – LTU) > τ, evict the page (because it is not in working set)

    If (t – LTU) <=τ, record the page of greatest age.

    age = current virtual time – last time used

The Working Set Page Replacement Algorithm



The working set algorithm

The Working Set Page Replacement Algorithm

- What happens if the entire table is scanned without finding a candidate to evict?
  - If one or more pages with R=0 were found, the one with the greatest age is evicted.
  - The worst case: no page with R=0, choose one page at random (preferably a clean page).

The WSClock Page Replacement Algorithm

- Previous WS algorithm requires entire page table be scanned at each page fault.
- WSClock Algorithm: an improved algorithm
  - Simple, efficient, widely used
  - Based on the clock algorithm
  - Also uses the working set information
  - Uses circular list of page frames

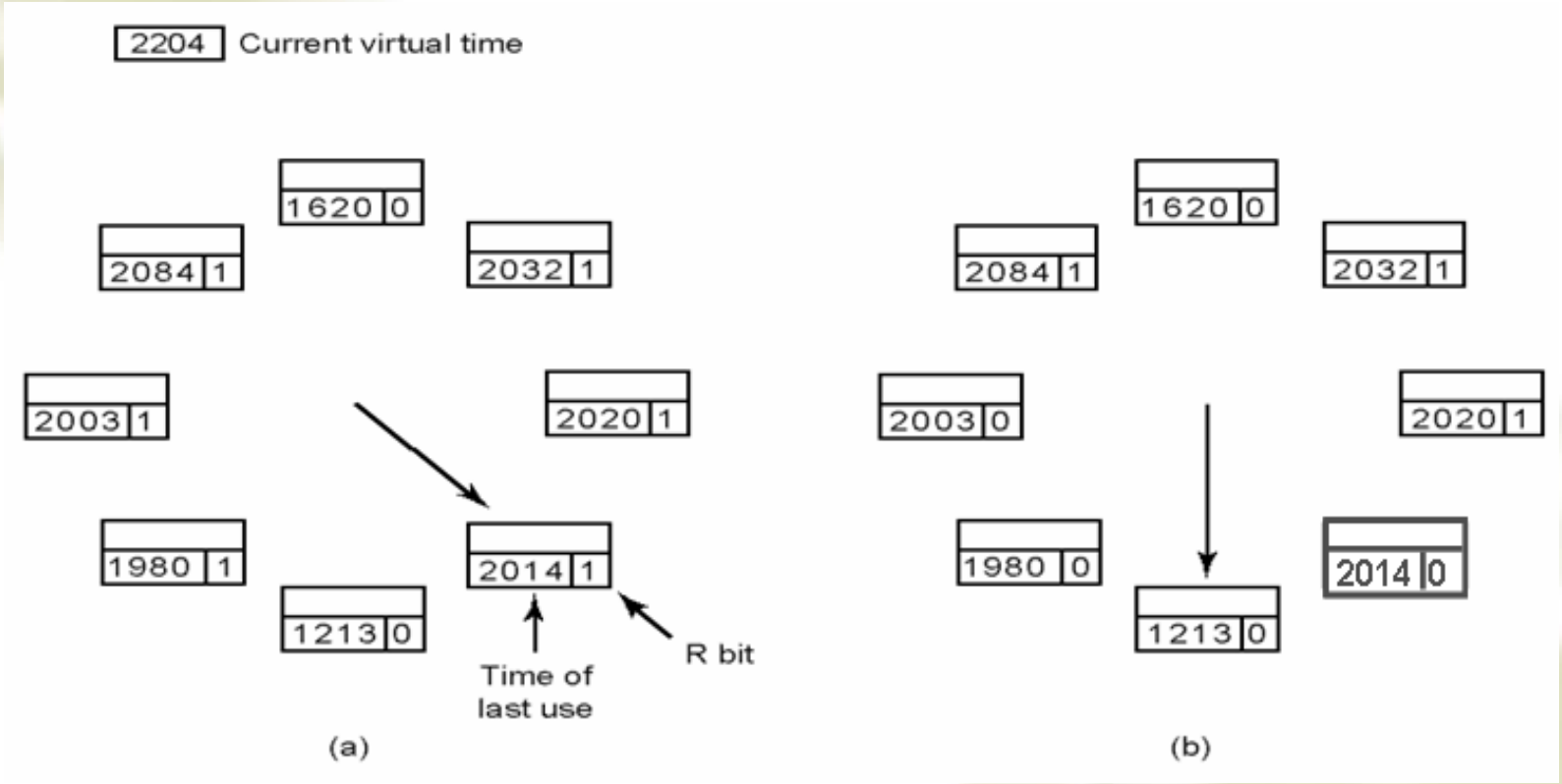
The WSClock Page Replacement Algorithm

- Upon a page fault
  - If R = 1, set R=0 and advance the hand and keep looking for.
  - If R = 0

    If age ≤ τ, advance the hand and keep looking for.
  - If age > τ, and if clean, then claim it.
  - if age > τ, and if dirty, then schedule a disk write.

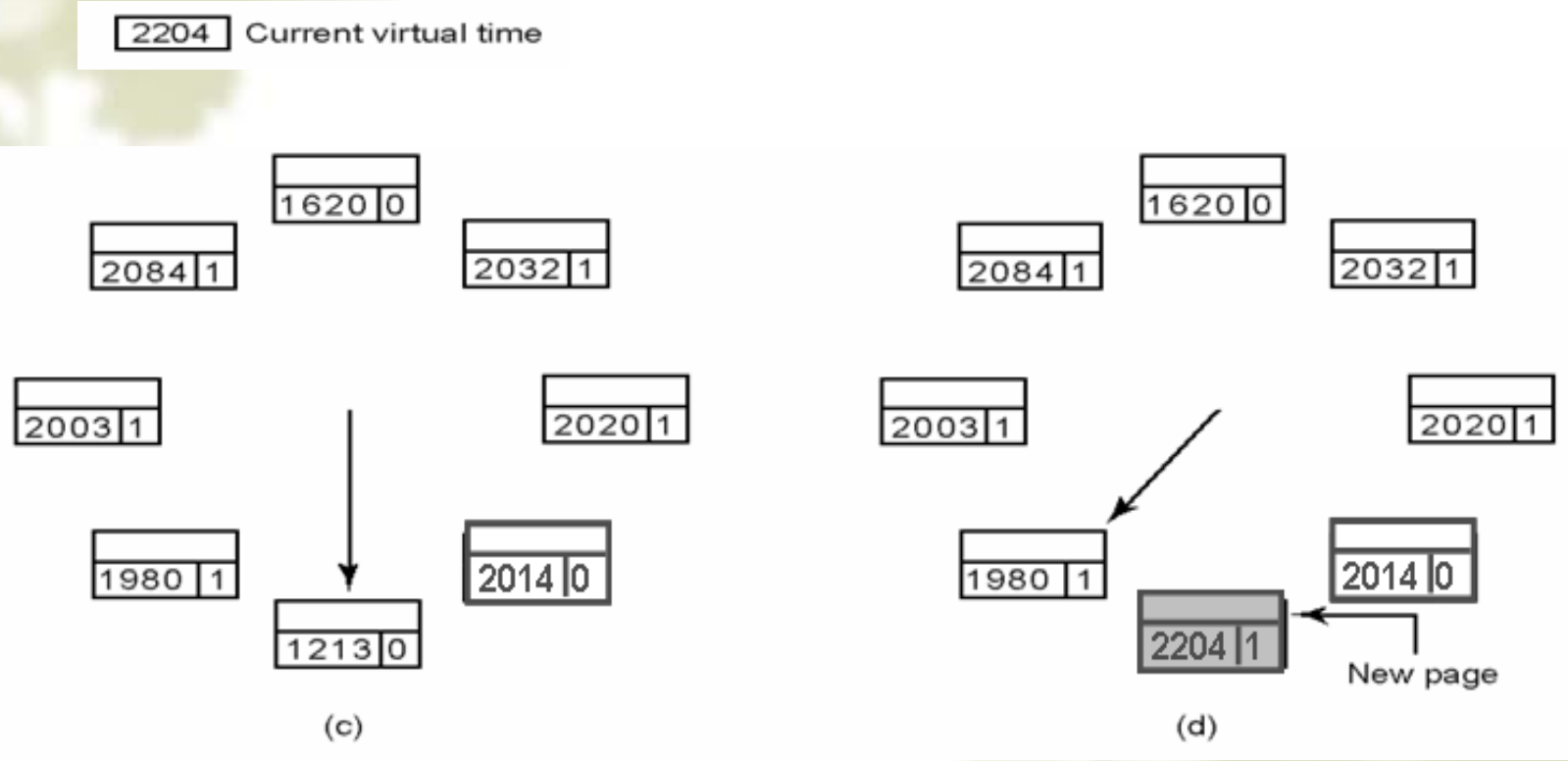
    Advance the hand and keep looking for.

The WSClock Page Replacement Algorithm



Operation of the WSClock algorithm.
(a) and (b) give an example of what happens when R = 1.

The WSClock Page Replacement Algorithm



Operation of the WSClock algorithm.
(c) and (d) give an example of R = 0.

The WSClock Page Replacement Algorithm

- What happens if the hand comes all the way around to its starting point?
  - Case1: At least one write has been scheduled.
    - The first clean page encountered is evicted
  - Case 2: No writes have been scheduled (all pages are in working set)
    - Claim any clean page and use it.
    - If no clean pages exist, the current page is chosen and written back to disk.

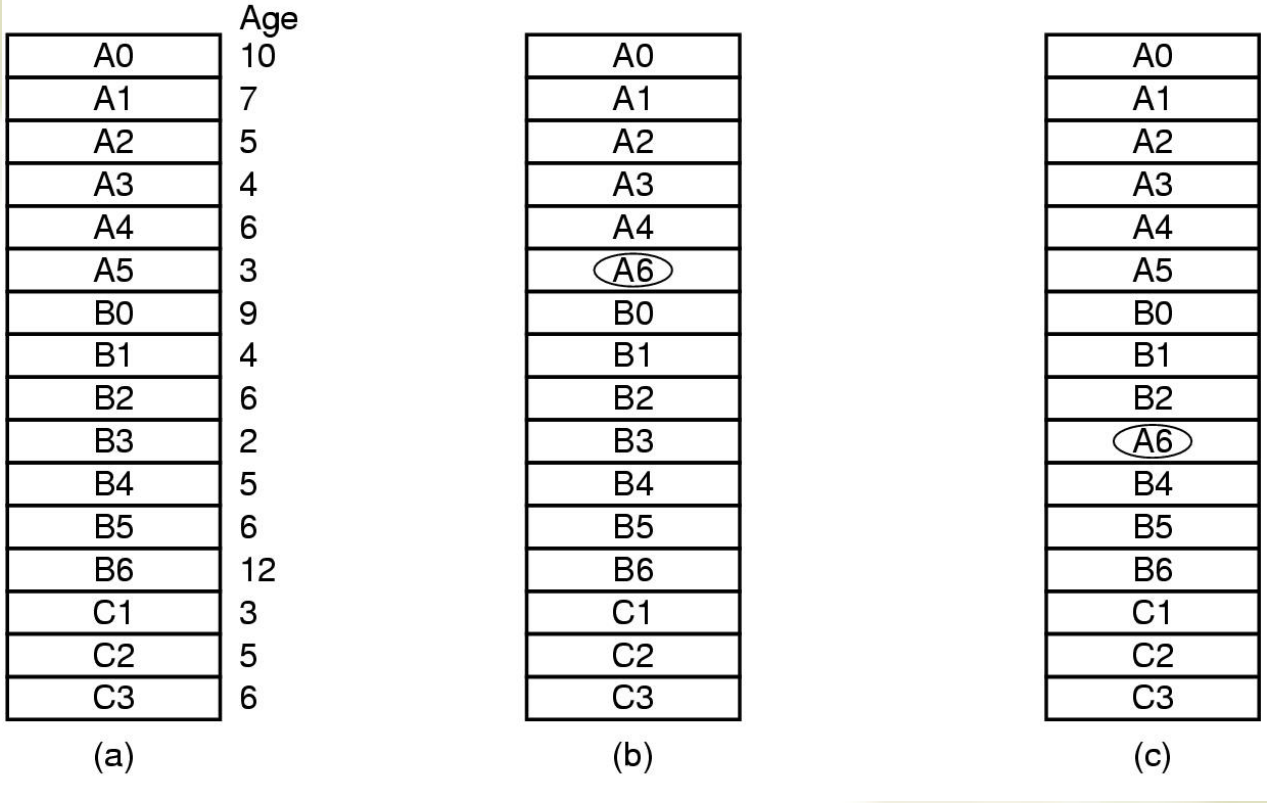
Review of Page Replacement Algorithms

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

- Two Best Algorithms: Aging and WSClock

Design Issues for Paging Systems

Local versus Global Allocation Policies



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

(a)

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(b)

A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

(c)

a) Original configuration
b) Local page replacement
c) Global page replacement

Load Control

- Despite good designs, system may still thrash
- Some processes need more memory
  but no processes need less
- Solution :
  Reduce a number of processes competing for memory
  - swap one or more to disk, divide up pages they held
  - reconsider degree of multiprogramming

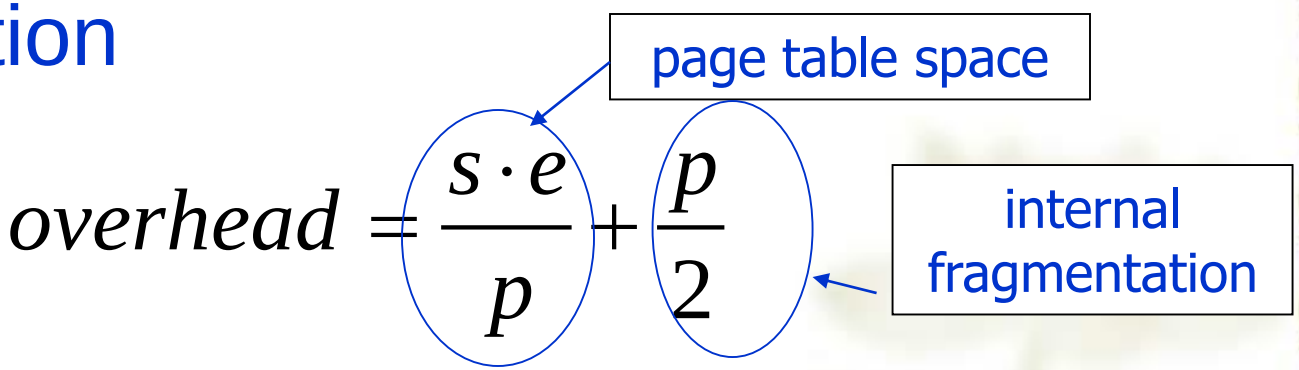
Page Size (1)

Small page size

- Advantages
  - less internal fragmentation
  - less unused program in memory
- Disadvantages
  - programs need many pages, larger page tables

Page Size (2)

- Overhead due to page table and internal fragmentation



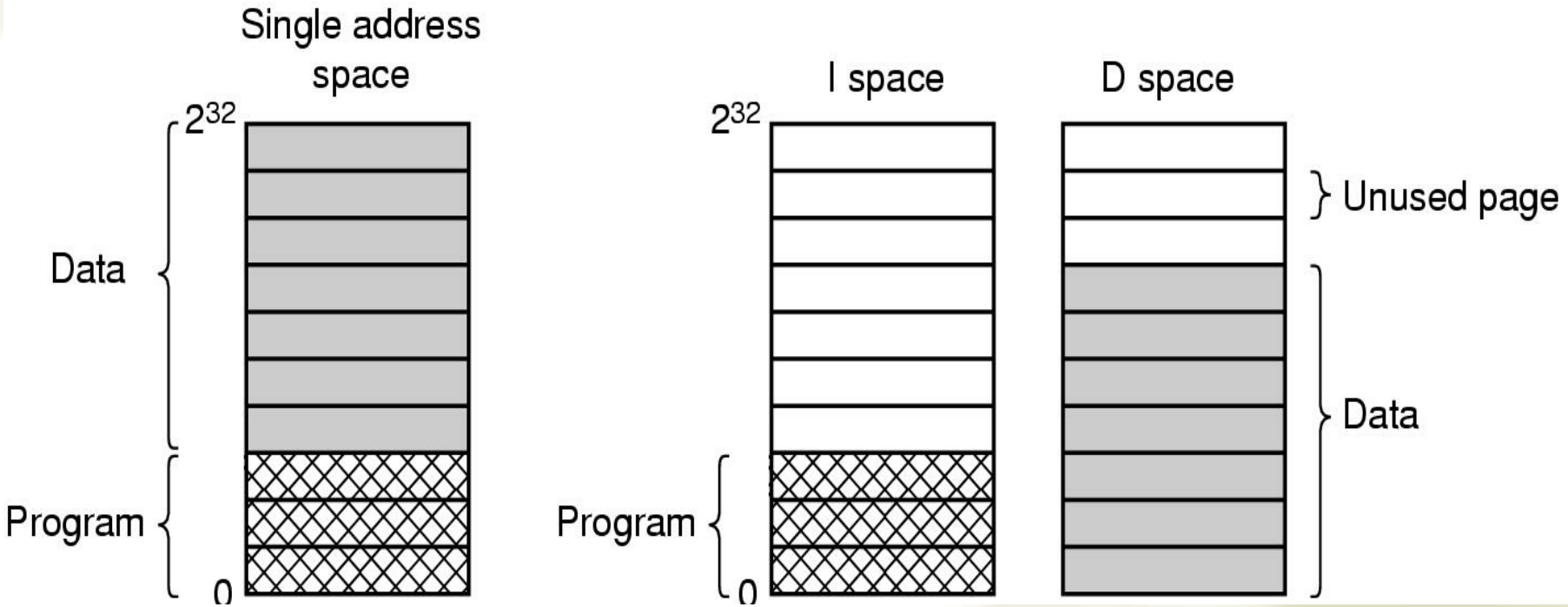
- Where
  - s = average process size in bytes
  - p = page size in bytes
  - e = page entry size in bytes

Optimized when

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

S=1MB, e=8, p=4KB

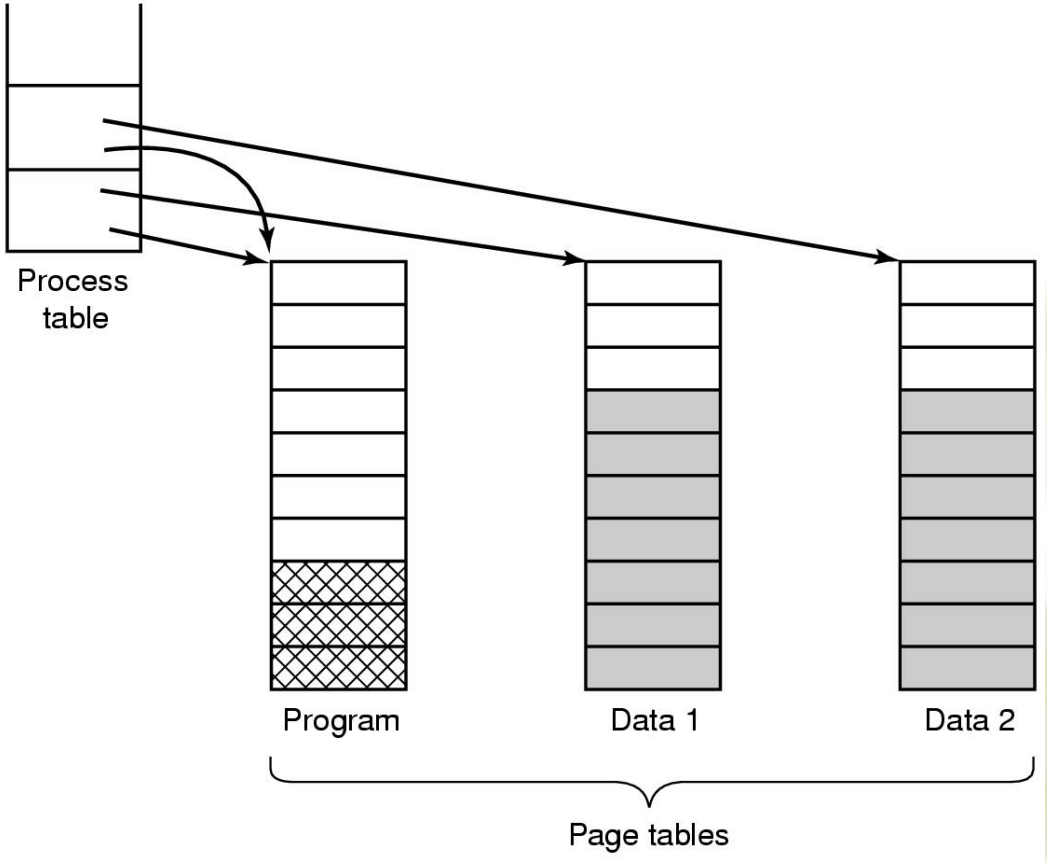
Separate Instruction and Data Spaces



- One address space
- Separate I and D spaces

Shared Pages

Separate instruction and data space, easy to share



Two processes sharing same program sharing its page table

Cleaning Policy

- Need for a background process, paging daemon
  - periodically inspects state of memory
- When too few frames are free
  - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
  - as regular page replacement algorithm but with diff ptr (front hand & back hand)

Implementation Issues

Operating System Involvement with Paging

Four times when OS involved with paging

1. Process creation
   - determine program 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)

1. Hardware traps to kernel, saving the PC on the stack
2. General registers saved
3. OS determines which virtual page needed
4. OS checks validity of address, seeks page frame
5. If selected frame is dirty, write it to disk, suspend the process

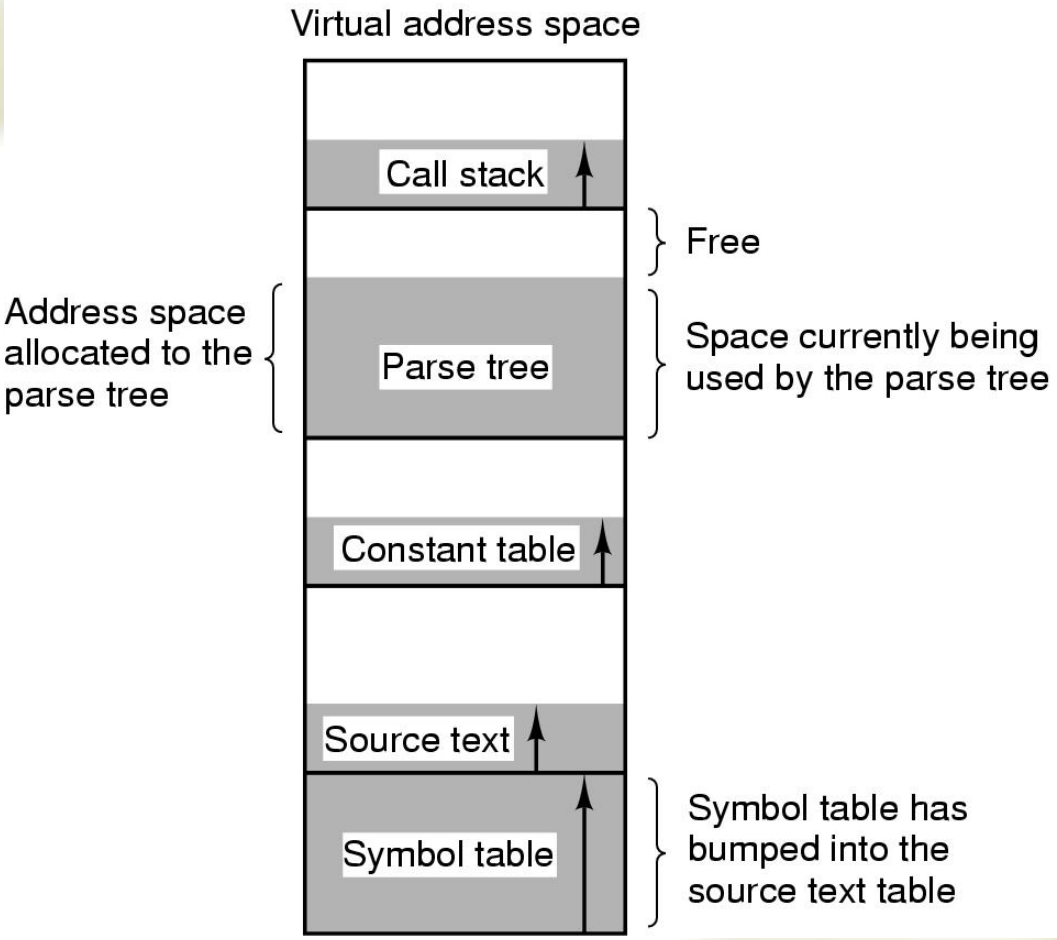
Page Fault Handling (2)

6. OS brings schedules new page in from disk. While page loading, process still suspend
7. Disk interrupts, page tables updated
8. Faulting instruction backed up to when it began
9. Faulting process scheduled
10. Registers restored
11. Program continues

Segmentation

- The virtual memory solution so far is paging (one-dimensional)
- For some applications, having two or more virtual address spaces may be much better than having only one.
  - For example, a compiler has many tables that are built up as compilation proceeds, such as source text, symbol table, constant table, parse tree and stack.
  - What happens if a program has a much larger number of variables but a normal amount of everything else?

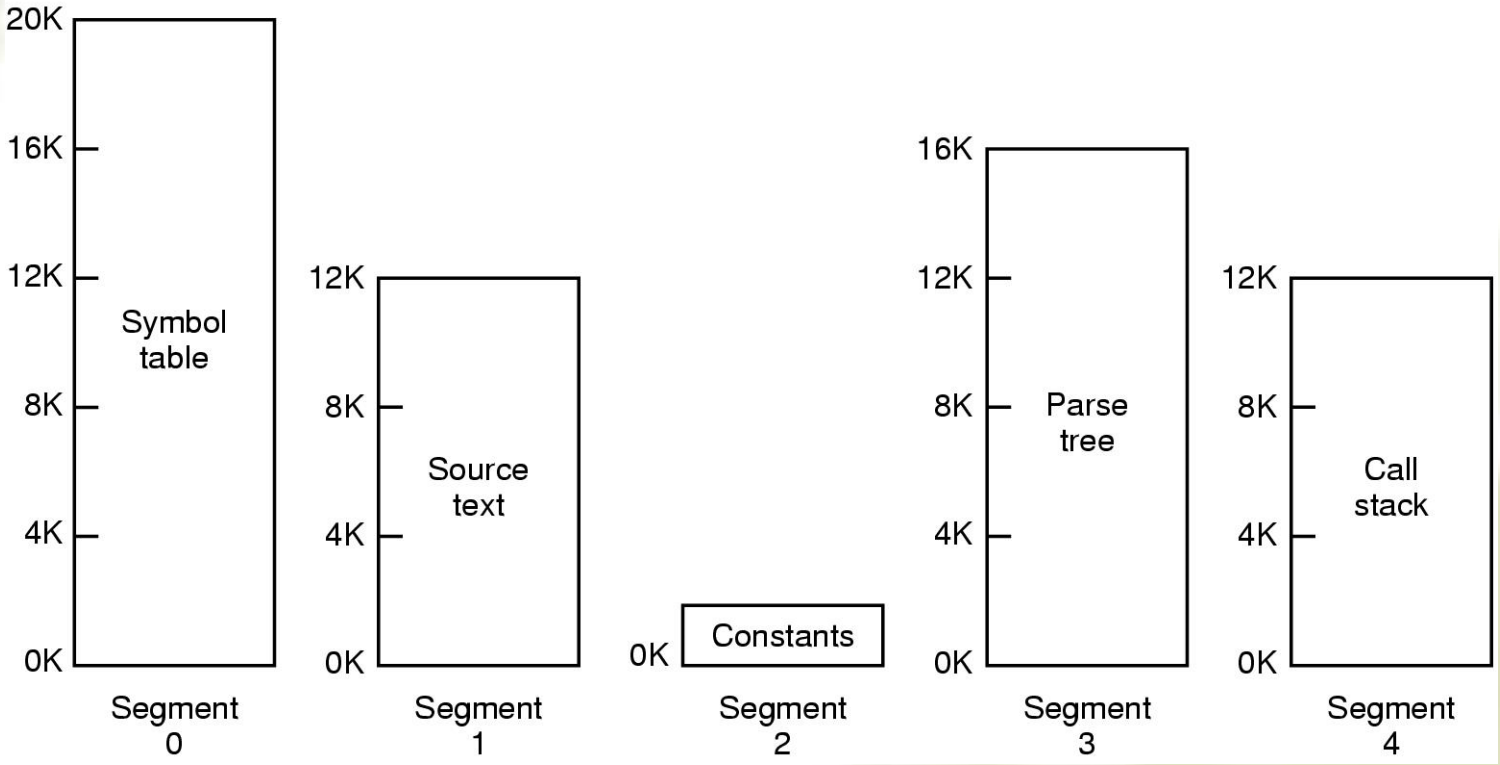
Segmentation (1)



- One-dimensional address space with growing tables
- One table may bump into another

Segmentation (2)

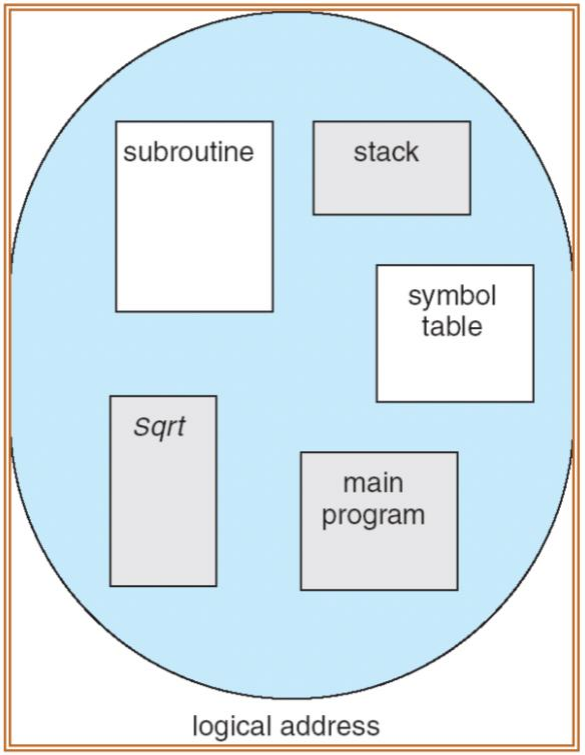
- Solution: To provide the machine with many completely independent address spaces, called segments.



A segmented memory allows each table to grow or shrink, independently

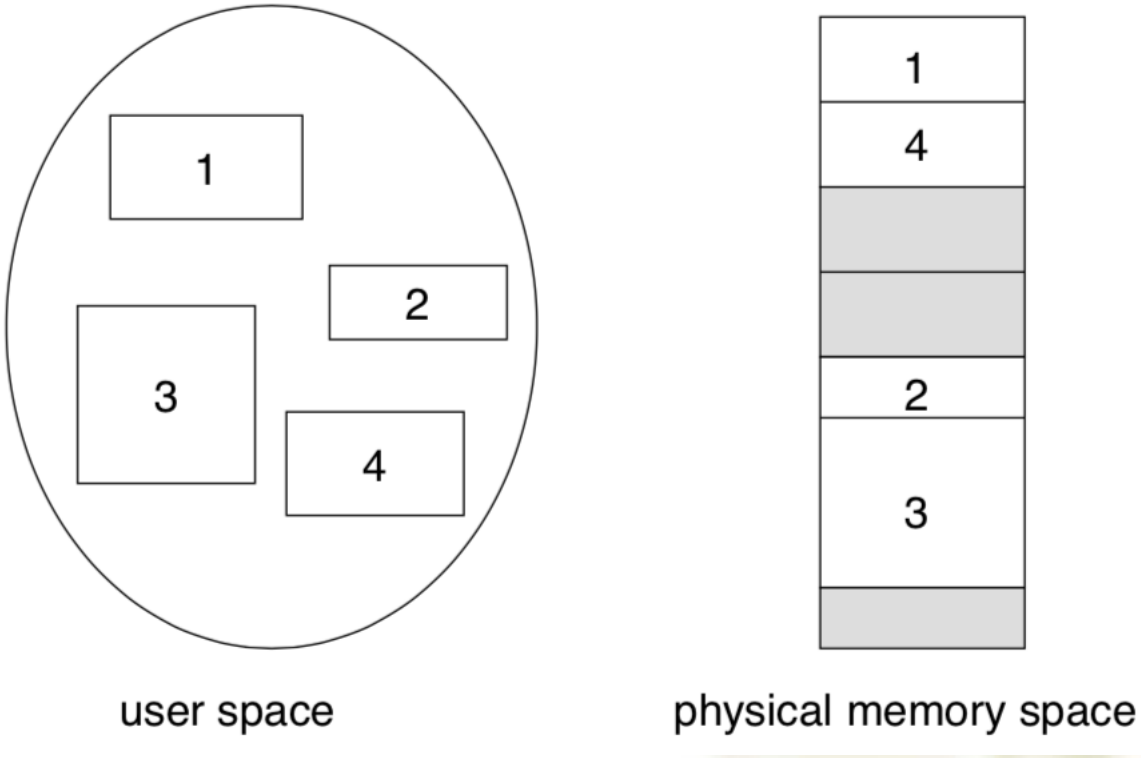
Segmentation (3)

- Segmentation is a memory-management scheme that supports user view of memory.
- User's view of a program
  - A program is a collection of segments. A segment is a logical unit such as
    - Main program
    - Procedure
    - Function
    - Symbol table
    - Stack



Segmentation (4)

- Logical view of segmentation



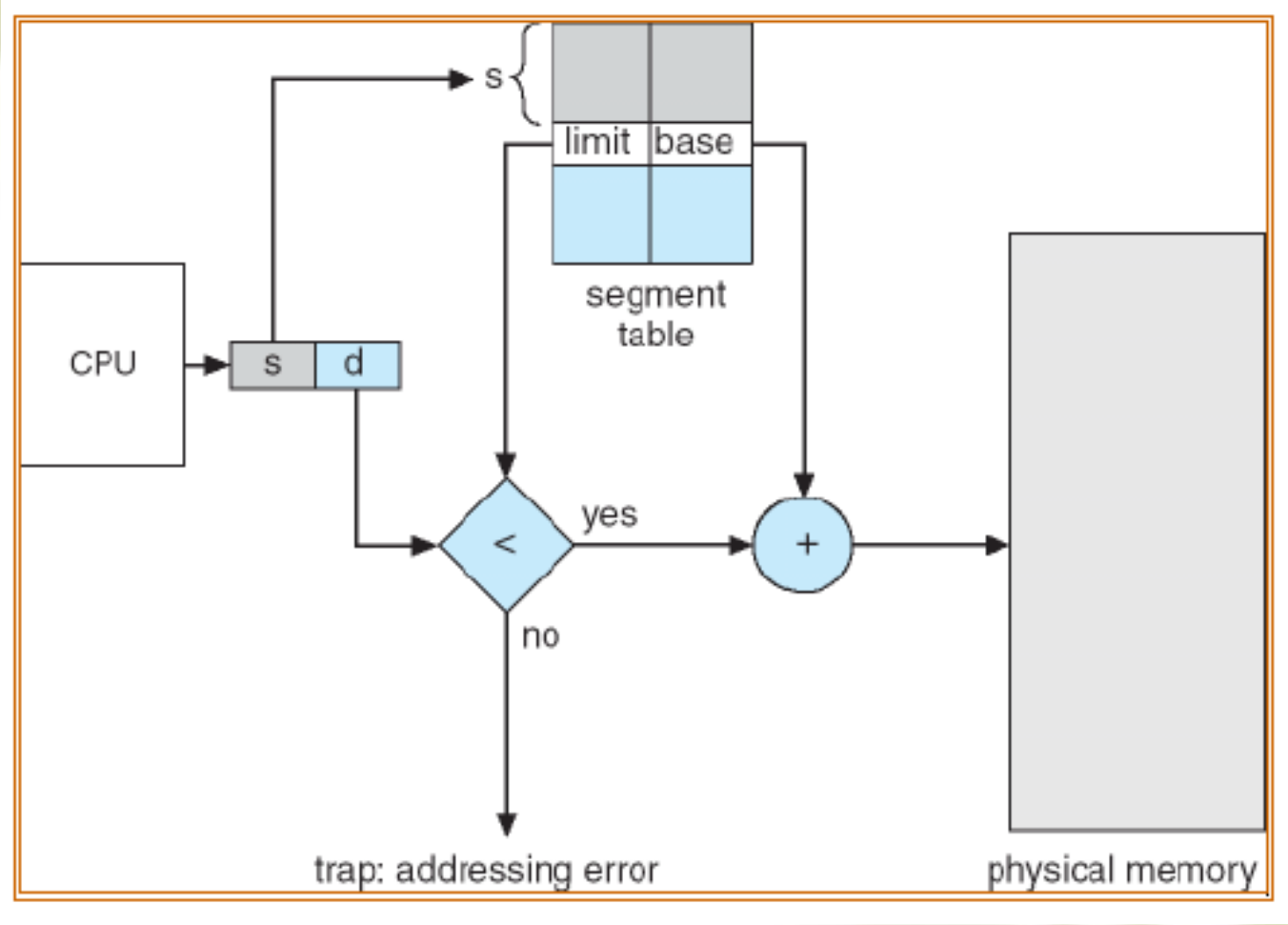
Segmentation Architecture (1)

- Logical address consists of two parts:

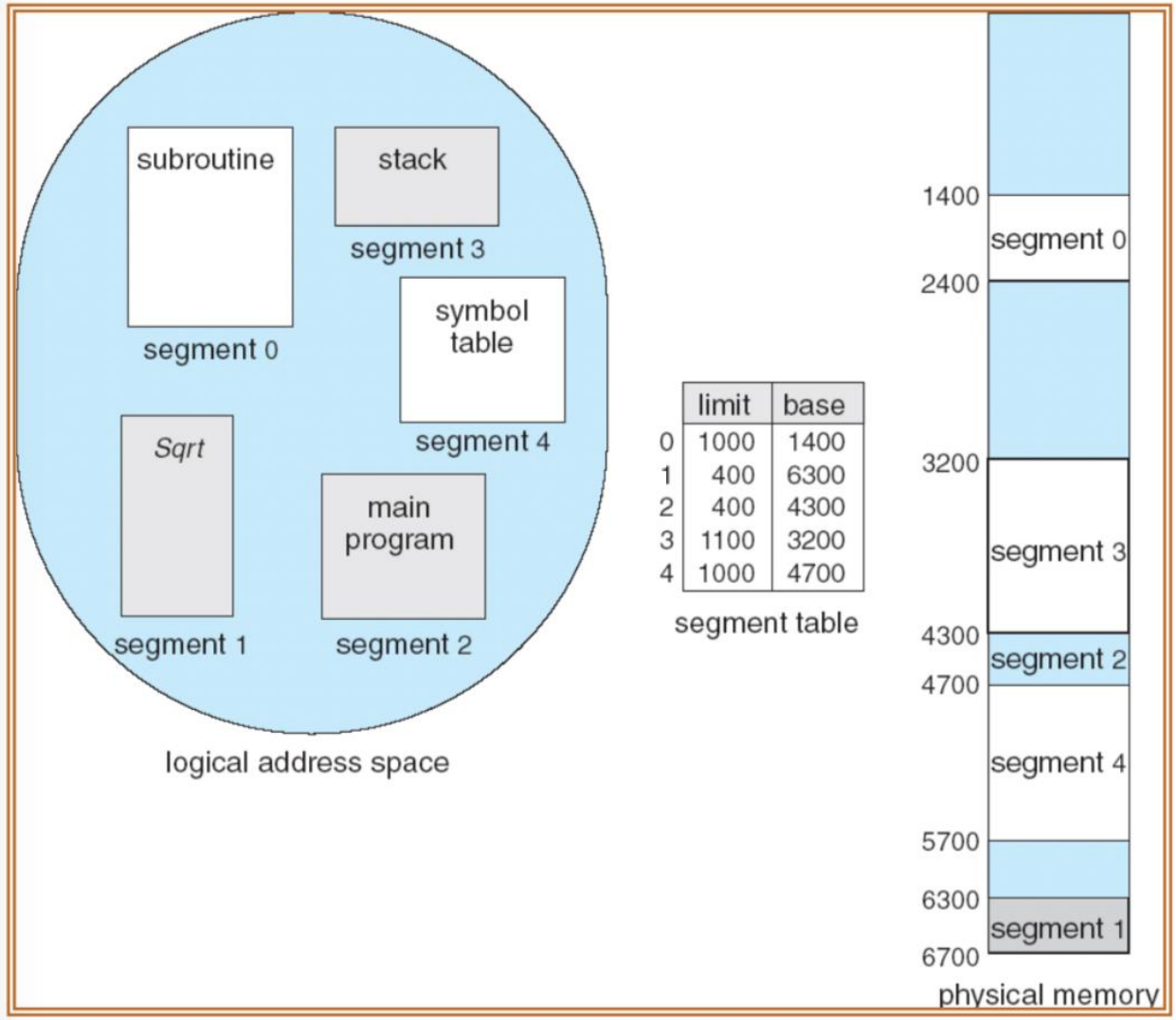
  <virtual segment-number, offset>

- Segment table
  - Maps two-dimensional user-defined addresses into one-dimensional physical addresses
  - The virtual segment number is used as an index to the segment table
  - Each table entry has
    - **Base**: contains the starting physical address where the segments reside in memory
    - **Limit**: specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory

Address Translation



Example of Segmentation



Exercise

- Using the following segment table, compute the physical address for the logical address consisting of segment and offset as follows
  - Segment 2 and offset 247
  - Segment 4 and offset 439

	Base	Limit
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

Solution

- Segment 2 and offset 247
  - From the segment table, the limit of segment 2 is 780 and the base of segment 2 is 2200
  - Since the offset is less than the limit (247<780),
  - Physical address = Segment base + Offset = 2200 + 247 = 2447

	Base	Limit
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

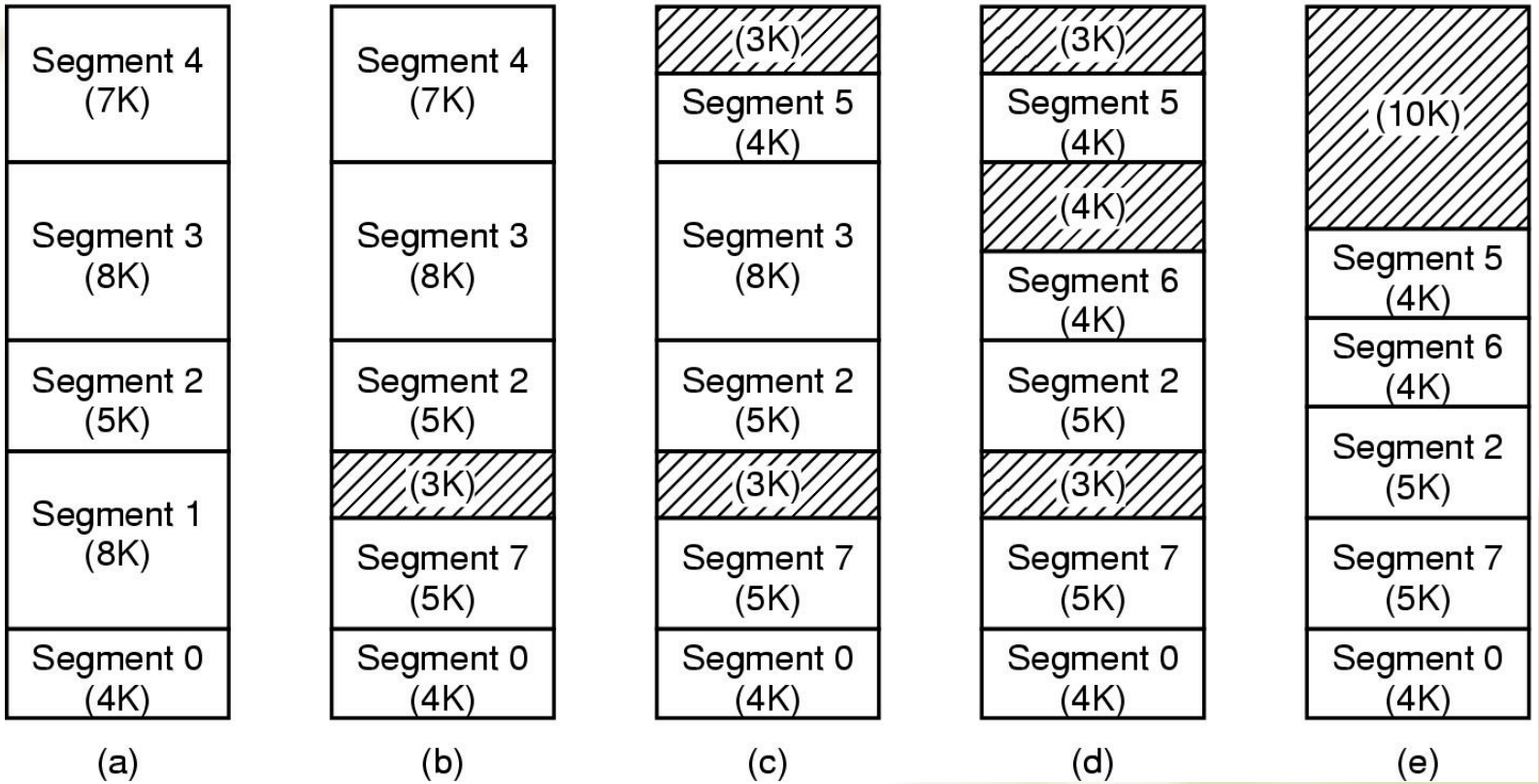
Solution

- Segment 4 and offset 439
  - From the segment table, the limit of segment 4 is 400 and the base of segment 4 is 1650
  - Since the offset is greater than the limit (439 > 400), invalid-address error is generated.

	Base	Limit
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

Implementation of Pure Segmentation

- Since segments vary in length, memory allocation is a dynamic storage-allocation problem



(a)-(d) Development of checkerboarding

(e) Removal of the checkerboarding by compaction

Paging vs. Segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Comparison of paging and segmentation

Advantage of Segmentation

- May be possible to shrink/grow segments
- Each segment can be given its own protection information…this is a much easier protection scheme than trying to protect each page of memory
- Linking programs is a trivial task
- Code can be shared between processes easily. Just load the code segment once. Copies of the same program access the same segment.

Disadvantage of Segmentation

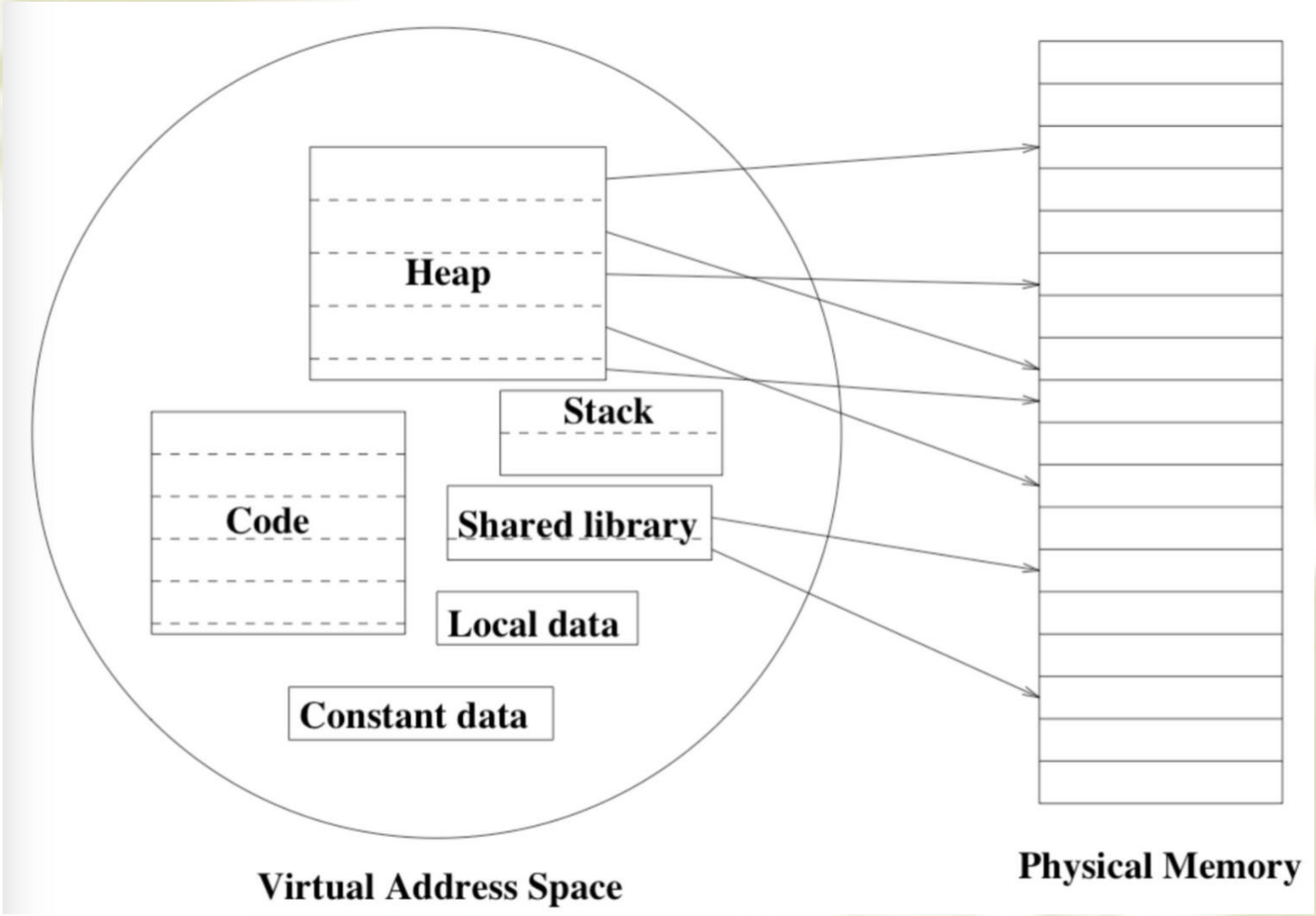
- Programmer must be aware of the memory model in use (at the assembly level, anyway)
- Just like with swapping systems, fragmentation can waste much memory.
- Segments may be too large to fit in physical memory

Segmentation with paging (1)

- Segmentation in virtual memory, paging in physical memory
- A segment is composed of pages
- An address has three components

- The physical memory contains only the demanded pages of a segment, not the full segment

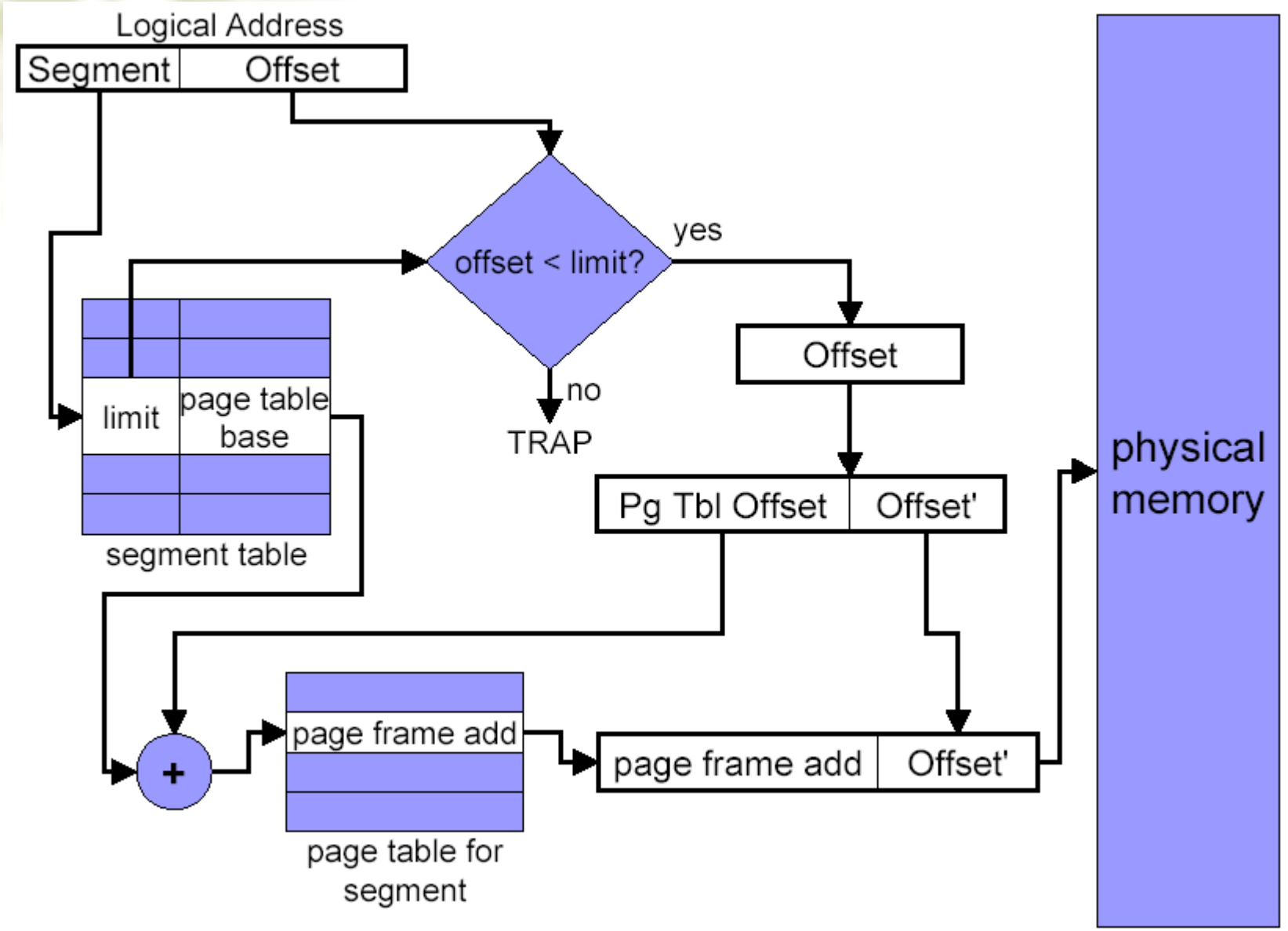
Segmentation with paging (1)



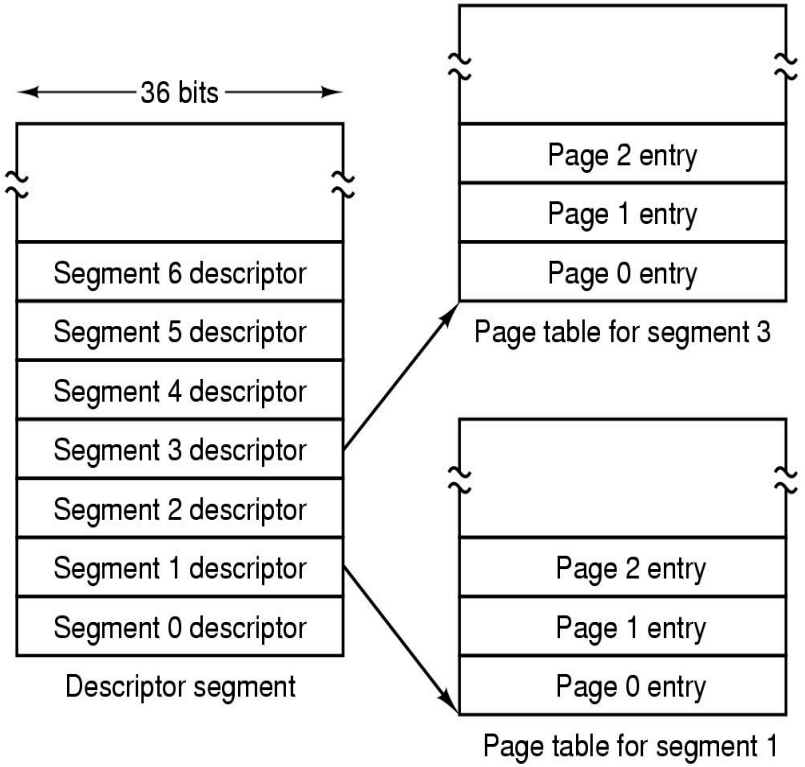
Segmentation with paging (2)

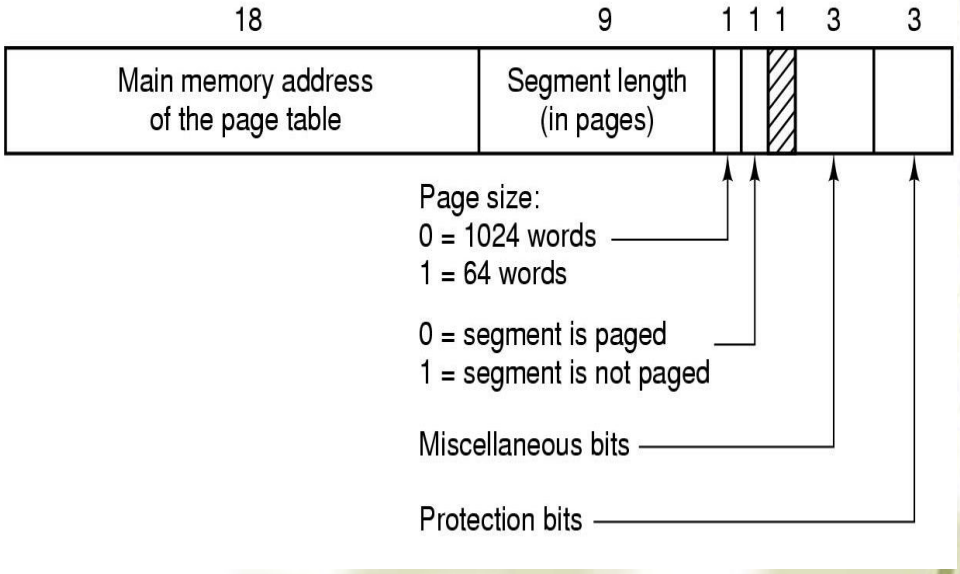
- Required
  - Each process needs a segment table
    - This table may be segmented and paged itself!
  - Each entry in the segment table points to the page table for that segment
    - Like before, this may be a multi-level page table

Address Translation



Segmentation with Paging: MULTICS

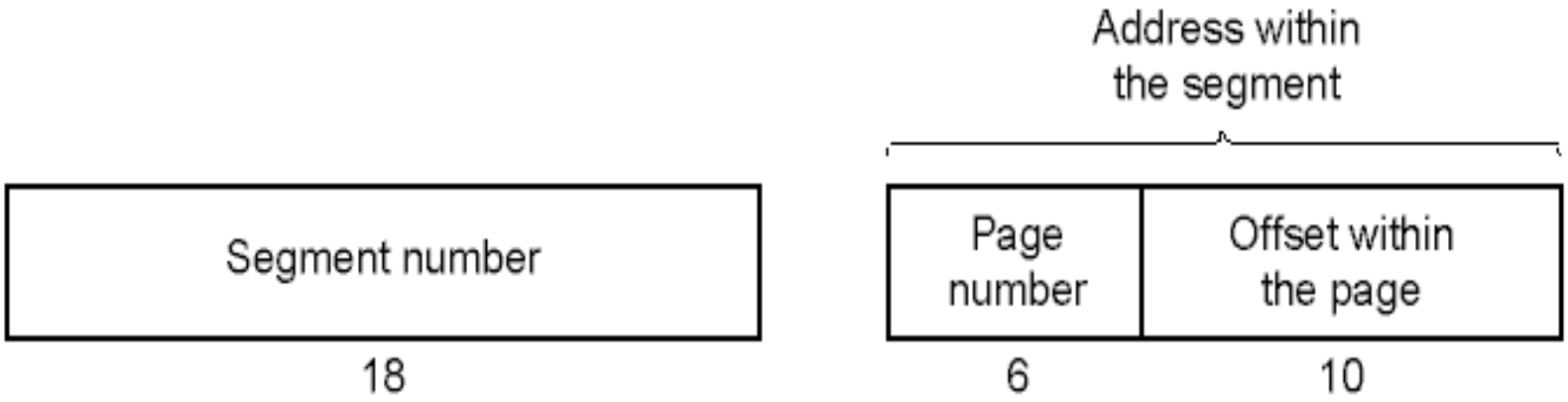




- Descriptor segment points to page tables
- Segment descriptor (numbers are field lengths)

Segmentation with Paging: MULTICS

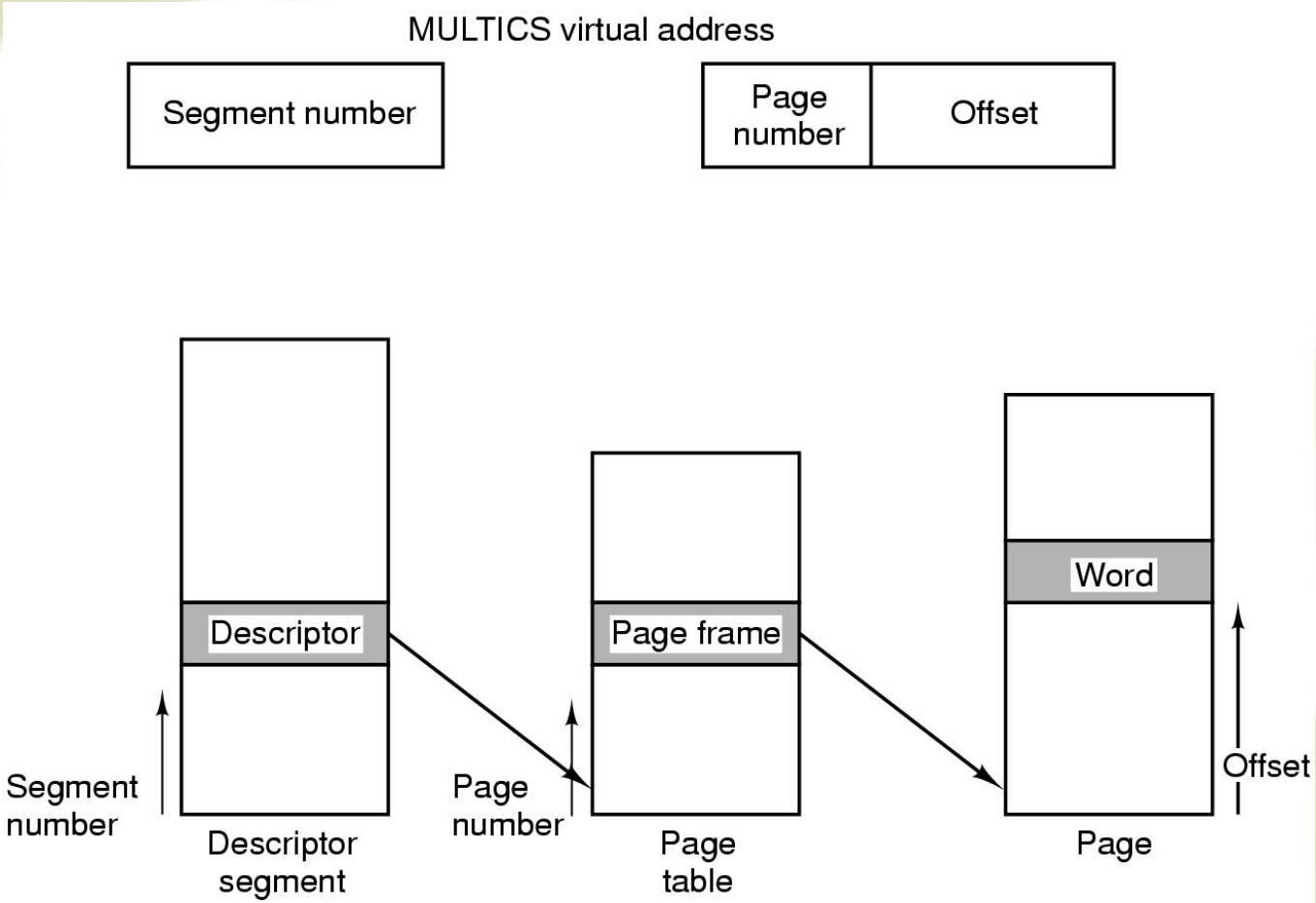
- Virtual address structure



A 34-bit MULTICS virtual address

Segmentation with Paging: MULTICS

- Address translation



Conversion of a 2-part MULTICS address into a main memory address

Segmentation with Paging: MULTICS

Comparison field					Is this entry used?
Segment number	Virtual page	Page frame	Protection	Age	
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

- Simplified version of the MULTICS TLB
- Existence of 2 page sizes makes actual TLB more complicated

Summary

- Fetch Strategies
- Page Replacement
  - Optimal
  - NRU
  - FIFO, Second-chance, Clock
  - LRU, NFU, Aging
  - Working Set, WSClock
- Design Issues for paging systems
- Implementation Issues
- Segmentation
- Segmentation with paging

Homework

- 4、6、7、17、19、25、28、30、33、36、38
- Reading assignment: 10.4