CGS 3763: Operating System Concepts Spring 2006

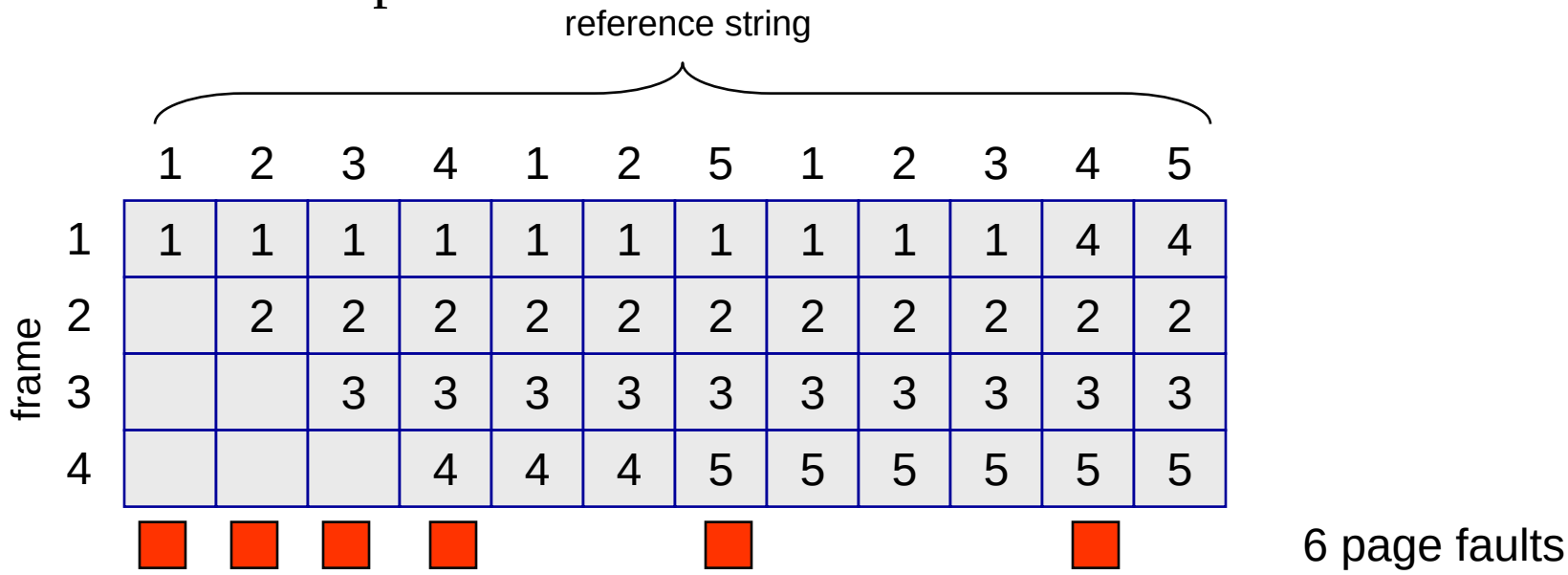
Memory Management – Part 5

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cgs3763/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida

Optimal Algorithm

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

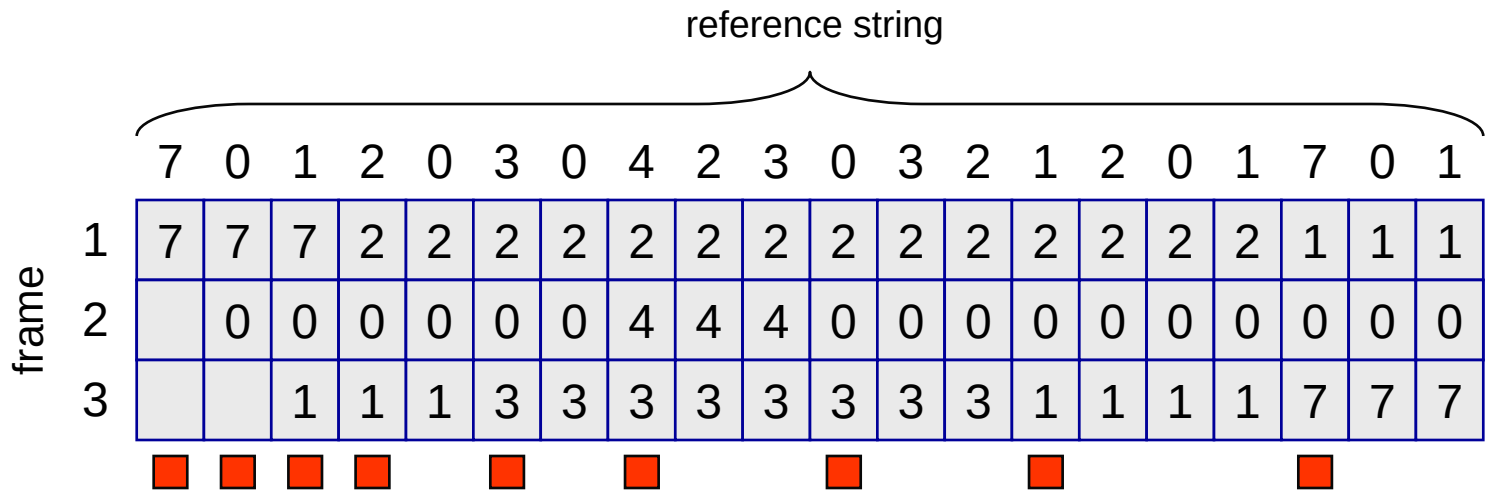


How do you know this?

- Used for measuring how well your algorithm performs

Optimal Algorithm – Another Example

- Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)



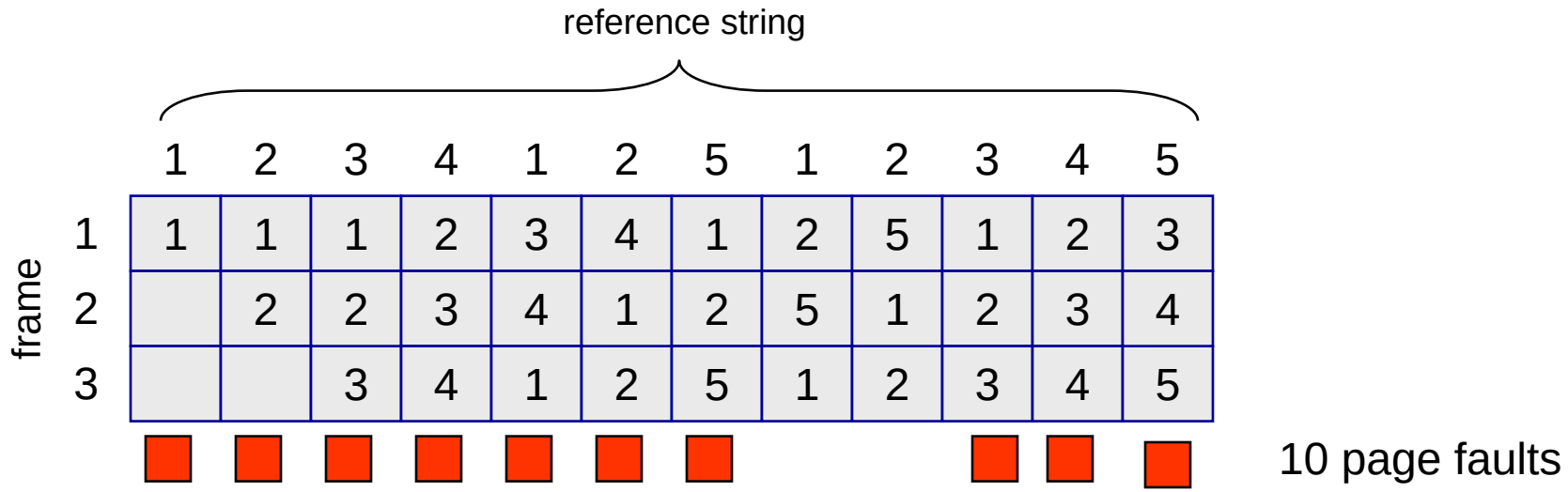
10 page faults

LRU Algorithm (cont.)

- While the optimal algorithm is not feasible for implementation, an approximation of it is possible.
- The main difference between the FIFO and Optimal algorithms (other than looking backward versus forward in time) is that the FIFO algorithm uses the time when a page was brought into memory, whereas the Optimal algorithm uses the time when a page is to be used.
- If we use the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of time.
- The Least Recently Used (LRU) algorithm is an approximation of the optimal algorithm.

LRU Example

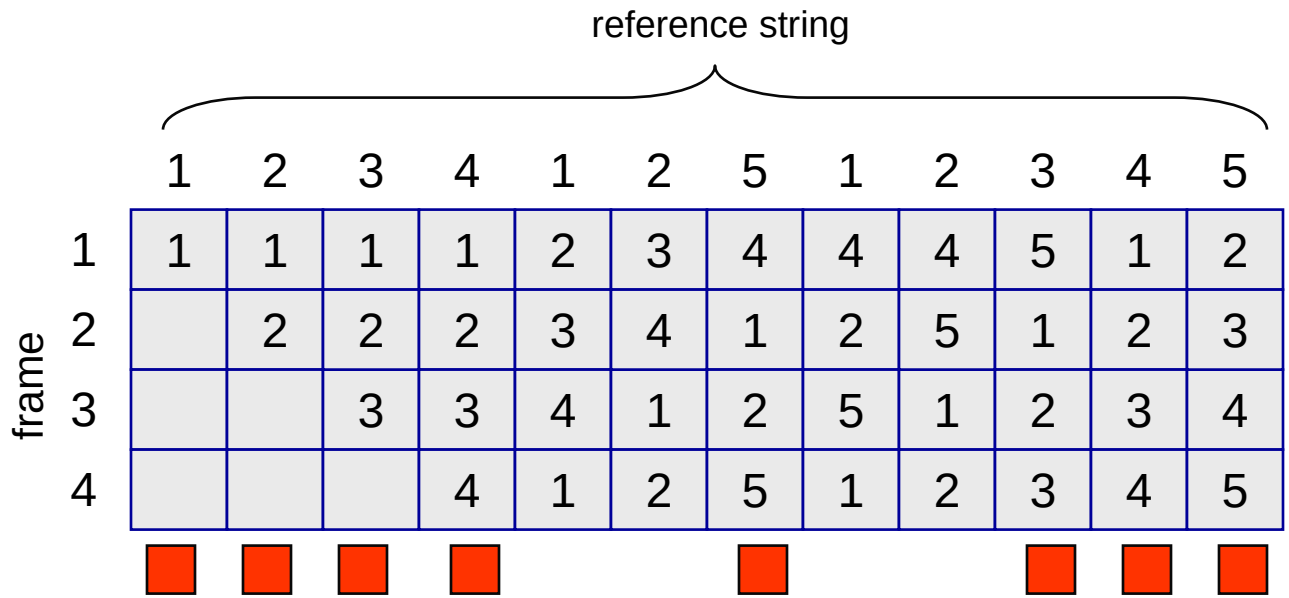
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



10 page faults

Another LRU Example

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



8 page faults

LRU Algorithm (cont.)

- The LRU algorithm can be viewed as the optimal page replacement looking backward in time rather than forward.
- Strangely, if we let S^R be the reverse of the reference string S, then the page fault rate for the Optimal algorithm on S is the same as the page-fault rate for the Optimal algorithm on S^R.
- Similarly, the page-fault rate for the LRU algorithm on S is the same as the page-fault rate for the LRU algorithm on S^R.
- The examples on the next two pages illustrate this phenomenon.

Optimal Algorithm On S and S^R

reference string = S

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
frame 1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1
frame 2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
frame 3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	7	7	7

10 page faults

reference string = S^R

	1	0	7	1	0	2	1	2	3	0	3	2	4	0	3	0	2	1	0	7
frame 1	1	1	1	1	1	1	1	1	3	3	3	3	4	4	3	3	3	1	1	1
frame 2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
frame 3			7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7

10 page faults

LRU Algorithm On S and S^R

reference string = S

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
frame 1	7	7	7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7
frame 2		0	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0
frame 3			1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

12 page faults

reference string = S^R

	1	0	7	1	0	2	1	2	3	0	3	2	4	0	3	0	2	1	0	7
frame 1	1	1	1	0	7	1	0	0	1	2	2	0	3	2	4	4	3	0	2	1
frame 2		0	0	7	1	0	2	1	2	3	0	3	2	4	0	3	0	2	1	0
frame 3			7	1	0	2	1	2	3	0	3	2	4	0	3	0	2	1	0	7

12 page faults

LRU Algorithm (cont.)

- The LRU algorithm is often used as a page replacement protocol and is considered to be a reasonably good technique.
- The major problem is how to implement LRU replacement.
- An LRU page-replacement algorithm may require substantial hardware assistance.
- The problem is to determine an order for the frames as defined by their last time of use.
- Two implementations are feasible:

  1. Counters
  2. Stack

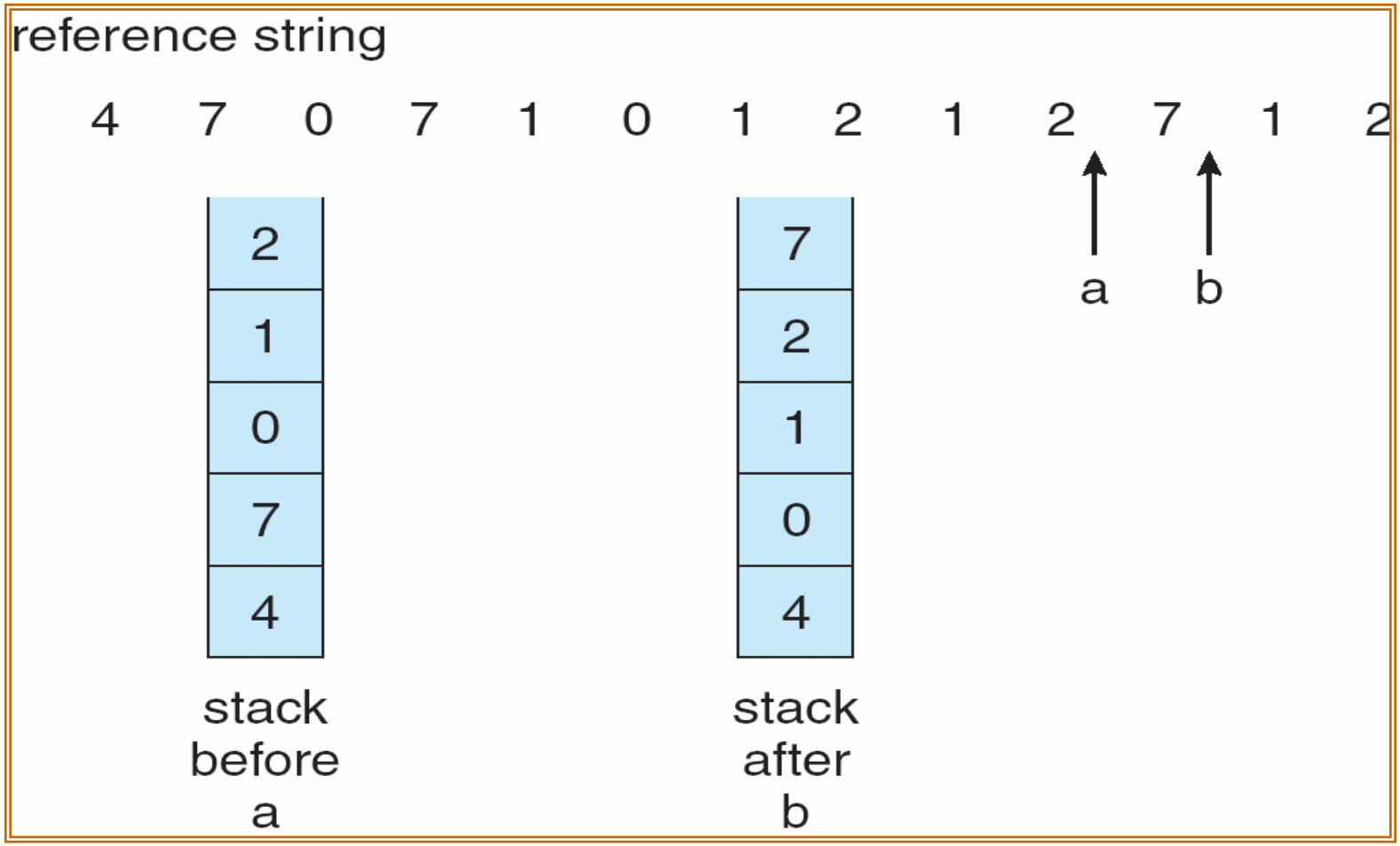
LRU Algorithm – Counter Implementation

- In the simplest case, each page-table entry has an associated time-of-use field and the CPU must include a logical clock or counter.
- The clock is incremented for every memory reference.
- Whenever a reference to a page is made, the content of the clock register are copied into the time-of-use field in the page-table entry for that page. This records the "time" of the last reference to that page.
- The page which is selected as a "victim" is the page with the smallest time value (the oldest page).
- This scheme requires a search of the page table to find the LRU page and a write to memory for each memory access. The times must also be maintained when page tables are changed due to CPU scheduling.

LRU Algorithm – Stack Implementation

- Another approach to implementing LRU replacement is to keep a stack of page numbers.
- Whenever a page is referenced, it is removed from the stack and put on the top. In this manner, the most recently used page is always on the top of the stack and the least recently used page is always on the bottom.
- Since entries are removed from the middle of the stack, the stack is typically implemented as a doubly linked list with head and tail pointers.
- In this fashion, removing a page and putting it on the top of the stack requires changing six pointer values in the worst case.
- Each update is more expensive (a bit) but there is no search required to find the LRU page.

Use Of A Stack to Record The Most Recent Page References



More On Belady's Anomaly

- Did you notice in the examples for the Optimal algorithm and the LRU algorithm that increasing the number of page frames allocated to a process did not increase the number of page faults as was the case with the FIFO algorithm?
- It turns out that the Optimal algorithm and the LRU algorithm belong to a class of algorithms known as stack algorithms.
- A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for *n* frames is always a subset of the pages that would be in memory with *n+1* page frames.
- Consider the LRU algorithm. The set of pages that would be in memory with an *n* frame allocation would be the *n* most recently used pages. If the number of page frames is increased, then these *n* pages will still be the most recently referenced and so would still be in memory.
- Stack algorithms do not suffer from Belady's anomaly.

LRU Approximation Algorithms

- Unfortunately, few computer systems provide sufficient hardware support for true LRU page replacement. Some systems provide no hardware support and must rely on page replacement algorithms such as FIFO.
- Many systems however, do provide some support in the form of a reference bit.
- The reference bit for a page is set by the hardware whenever that page is referenced (either by a read or a write to any byte in the page).
- Reference bits are associated with each entry in the page table.
- Initially, all bits are cleared (set to 0) by the OS. As a user process executed, the bit associated with each page referenced is set (to 1) by the hardware. After some period of time, we can determine which pages have been used and which have not been used by examining the reference bits, although we do not know the order of their use.
- This information is the basis for many page-replacement algorithms that approximate the behavior of the LRU algorithm.

Additional-Reference-Bits Algorithm

- Additional information about the order in which pages are referenced can be obtained by recording the reference bits at regular intervals.
- A single byte (8-bits) is maintained for each page in a table in memory.
- At regular intervals (say 100 msec), a timer interrupt transfers control to the OS. The OS shifts the reference bit for each page into the high-order bit of the its byte, shifting the other bits in the byte, one bit to the right and discarding the low-order bit.
- These 8-bit shift registers contain the history of page use for the last eight time periods.
- If the shift register for a page contains 00000000, then that page has not been referenced for eight time periods.

Additional-Reference-Bits Algorithm (cont.)

- Similarly, a page that has been referenced at least once during each of the last eight time periods would have a history register value of 11111111.
- A page with a history register value of 11000100 has been used more recently than a page with a register value of 01110111.
- If the history register values are interpreted as unsigned integers, the page with the lowest number is the LRU page.
- Notice, that the numbers may not be unique, so it is possible to page out all of the pages with the smallest value, or use the FIFO method to choose amongst them.
- The number of bits of history can be varied and is typically selected, depending on the hardware available, to make the updating as fast as possible.

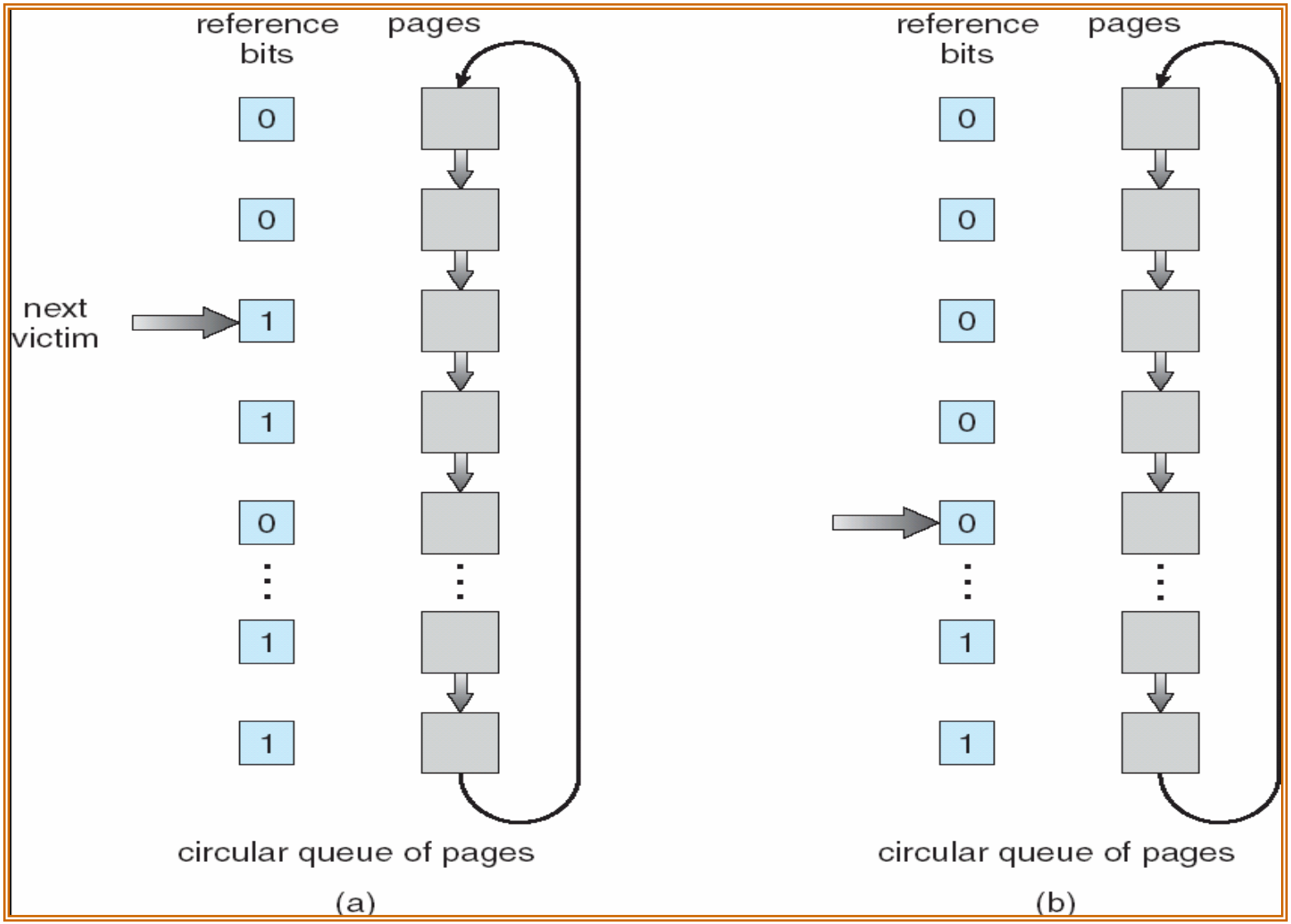
Second-Chance Algorithm

- In the extreme case, the number of bits in the history register is reduced to 0 (i.e., there is no history register, only the reference bit on the page).
- In this case, the page-replacement algorithm is called the second-chance page-replacement algorithm.
- The basic algorithm of second-chance replacement is FIFO, however, when a page has been selected, its reference bit is checked. If the value is 0, the page is replaced; but if the value is 1, we give the page a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chances).
- Additionally, if a page is used often enough to keep its reference bit set to 1, it will never be replaced.

Clock Algorithm

- One way to implement the second-chance algorithm is via a circular queue. This implementation is referred to as the clock algorithm.
- A pointer (i.e., a hand on the clock) indicates which page is to be replaced next.
- When a frame is needed, the pointer advances until if finds a page with a 0 reference bit. As it advances, it clears all the reference bits.
- Once a victim page is found, the page is replaced, and the new page is inserted in the circular queue in that position.
- The figure on the next page illustrates this implementation.
- Notice that, in the worst case, when all bits are set, the pointer cycles through the entire queue, giving each page a second chance. It clears all the reference bits before selecting the next page for replacements.
- Second-chance degenerates to FIFO if all the replacement bits are set.

Clock Page-Replacement Algorithm



Enhanced Second-Chance Algorithm

- The second-chance algorithm can be enhanced by considering the reference bit and the modify bit (the bit used to indicate whether any bit on a page has been modified) as an ordered pair.
- With these two bits, there are four possible classes that can be defined:
  1. (0,0) neither recently used nor modified, the best page to replace.
  2. (0,1) not recently used but modified, not quite as good because the page will need to be written out before it can be replaced.
  3. (1,0) recently used but clean – probably will be used again soon.
  4. (1,1) recently used and modified – probably will be used again soon, and the page will need to be written to disk before it can be replaced. Worst case as a victim.
- Each page is in one of these four classes.

Enhanced Second-Chance Algorithm (cont.)

- When page replacement is required, the same basic scheme as the clock algorithm is utilized; but instead of examining whether the page to which the pointer is pointing has the reference bit set to 1, a check is made to determine the class to which that page belongs.
- The page to replace is the first page encountered in the lowest nonempty class.
- Notice that the circular queue may require several scans before a page to replace can be found.
- The major difference between this algorithm and the simpler clock algorithm is that in this case preference is given to those pages that have been modified in order to reduce the number of I/O operations that will be required.