OPERATING SYSTEMS

Virtual Memory & Demand Paging

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

中山大學
SUN YAT-SEN UNIVERSITY

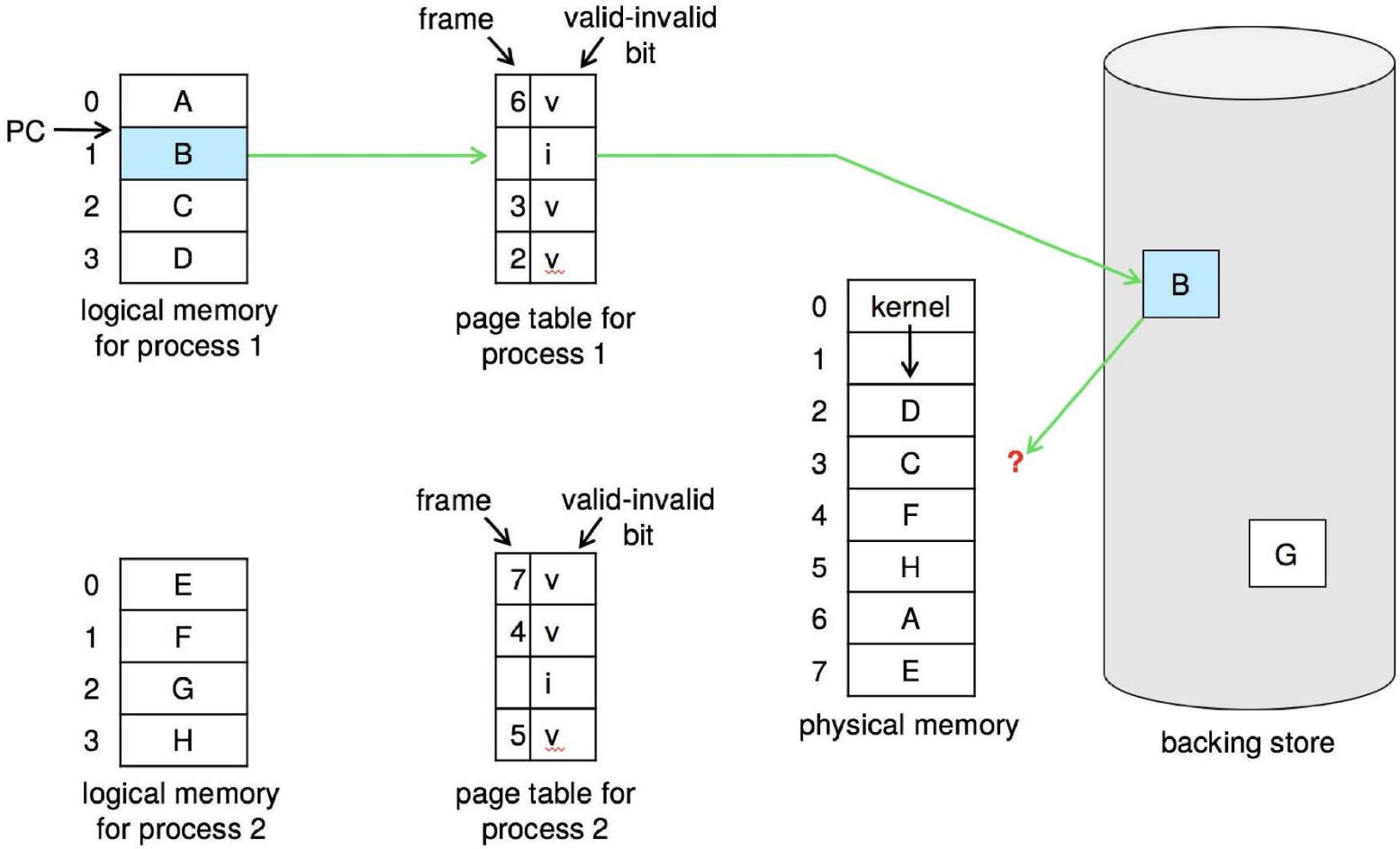
Contents

Page Replacement

- Page replacement (页面置换) finds some page in memory, but not really in use, and pages it out.
  - Prevent *over-allocation* of memory by modifying page-fault service routine to include page replacement.
  - Use *modify bit* (dirty bit) to reduce overhead of page transfers – only modified pages are written back to disk.
- Page replacement completes *separation* between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

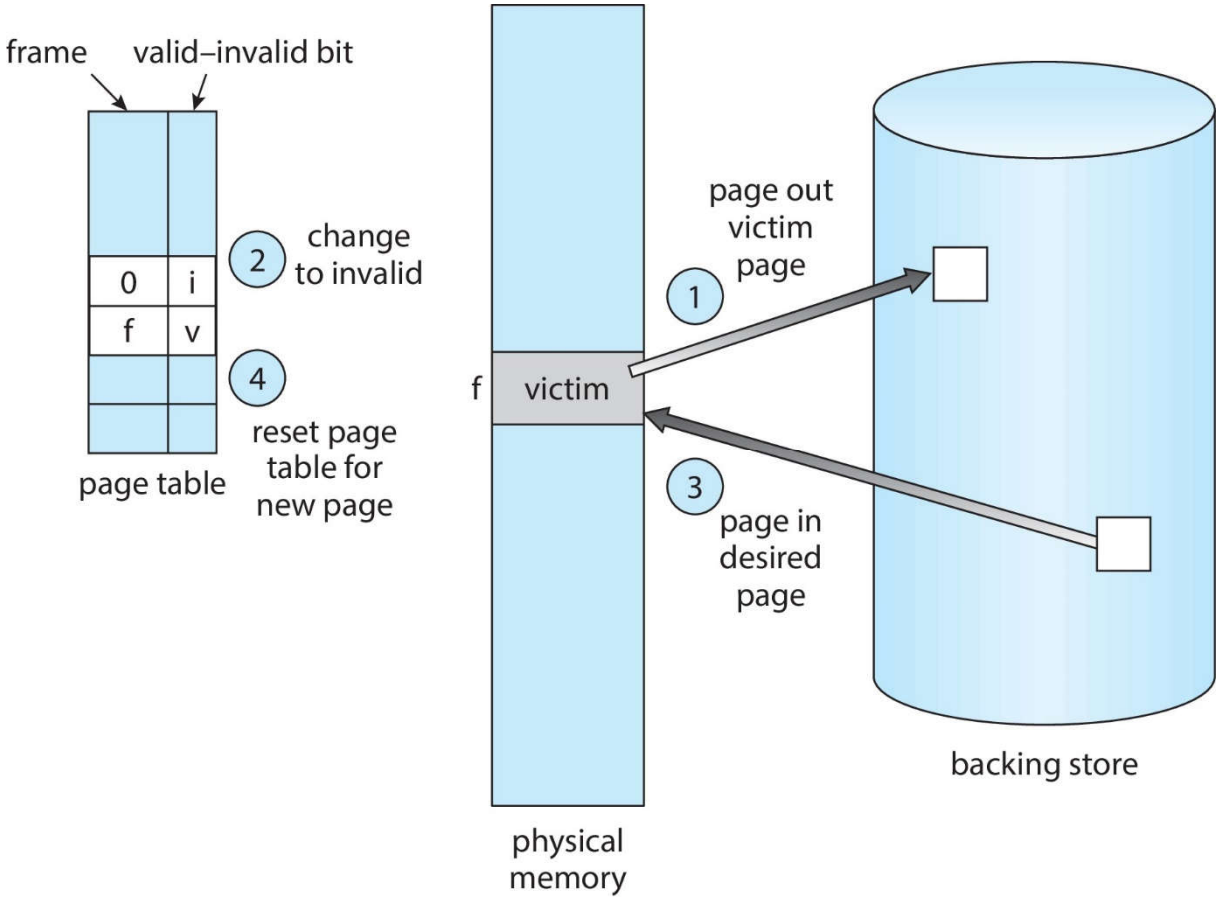
Page Replacement

- Need For Page Replacement.



Page Replacement

- Basic Page Replacement.



Page Replacement

- Basic Page Replacement
  (1) Find the location of the desired page on secondary storage.
  (2) Find a free frame:
    (a) If there is a free frame, use it.
    (b) If there is no free frame, use a page-replacement algorithm to select a victim frame.
    (c) Write the victim frame to secondary storage (if dirty); change the page and frame tables accordingly.
  (3) Read the desired page into the newly freed frame; change the page and frame tables.
  (4) Continue the process from where the page fault occurred.

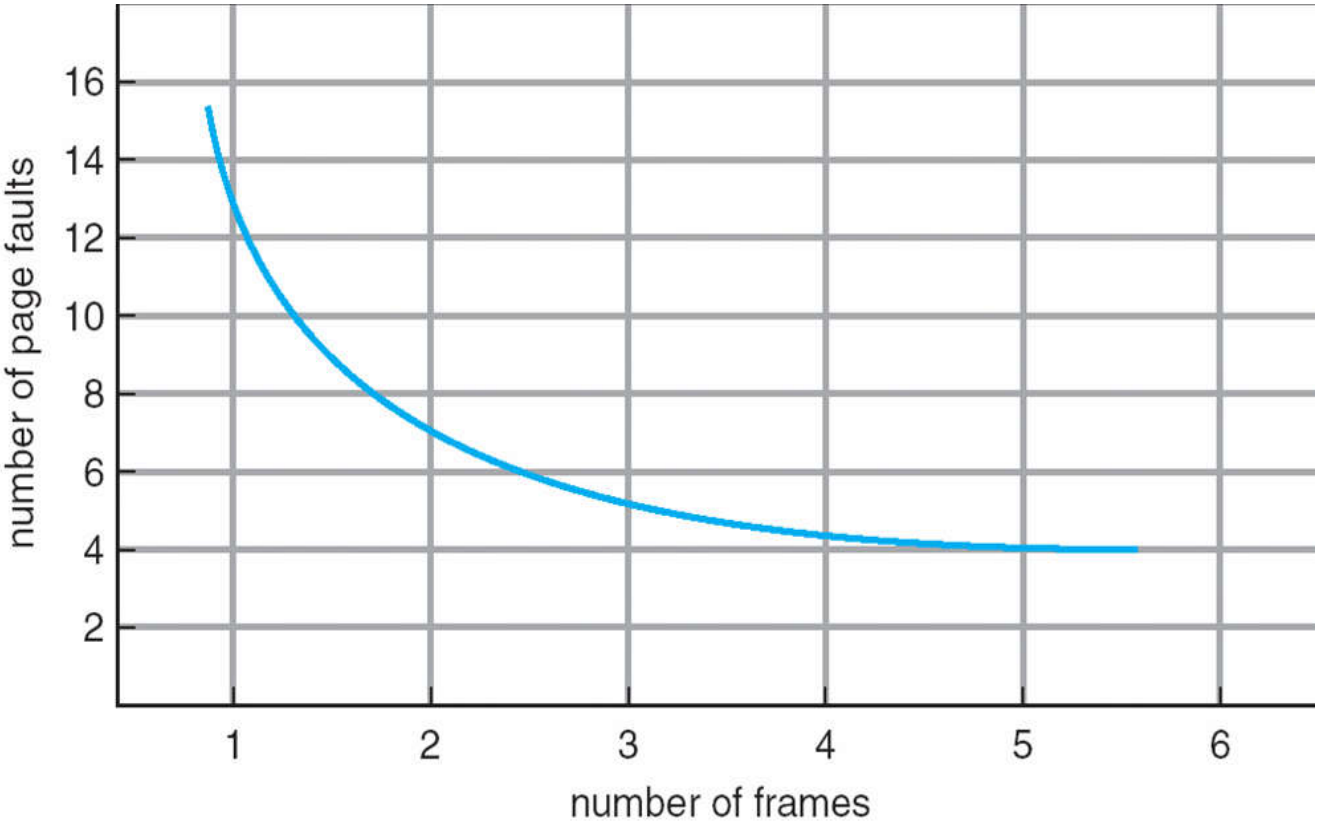
- Note now potentially 2 page transfers for page fault – increasing EAT

Page Replacement

- *Frame-allocation algorithm* determines
  - how many frames to give to each process
  - which frames to replace.
- *Page-replacement algorithm*
  - want lowest page-fault rate on both first access and re-access
- A page replacement algorithm is evaluated by running it on a particular string of memory references (a reference string) and computing the number of page faults on that string.
  - The string contains just page numbers, not full addresses.
  - Repeated access to the same page does not cause a page fault.
  - Results depend on number of frames available.
- In all our examples, we use a few recurring reference strings.

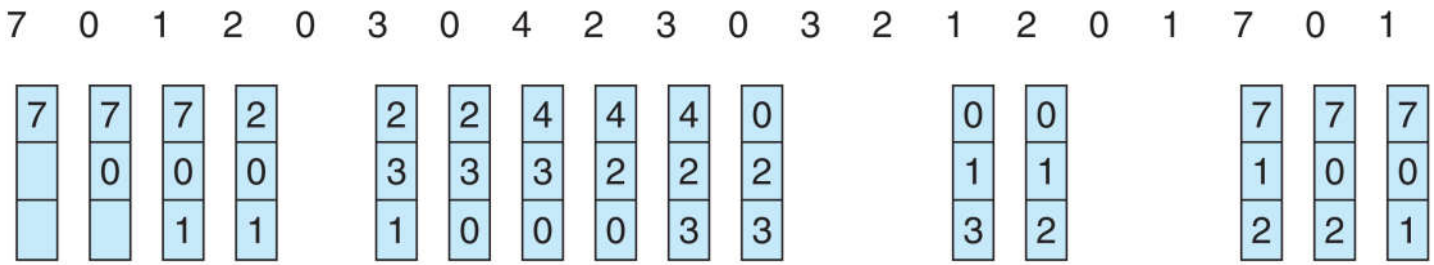
Page Replacement Algorithms

- Number of Page Faults vs. Number of Frames.
  - An expecting curve: as the number of frames increases, the number of page faults drops to some minimal level.



First-In-First-Out (FIFO) Algorithm

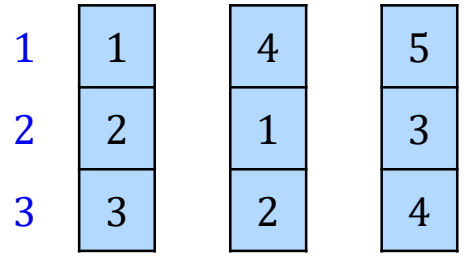
- The FIFO Policy
  - Treats page frames allocated to a process as a circular buffer.
    - When the buffer is full, the oldest page is replaced. Hence first-in, first-out.
      - A frequently used page is often the oldest, so it will be repeatedly paged out by First-In-First-Out (FIFO).
    - Simple to implement
      - requires only a pointer that circles through the page frames of the process.
  - 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



15 Page Faults

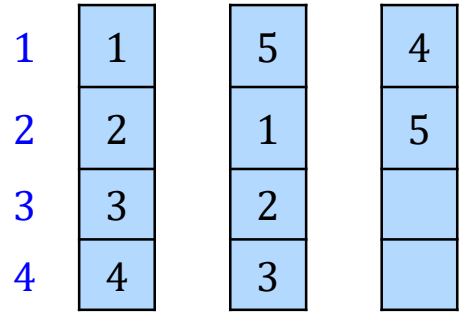
First-In-First-Out (FIFO) Algorithm

- Example of *Belady*'s Anomaly
  - 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):



9 page faults

  - 4 frames:

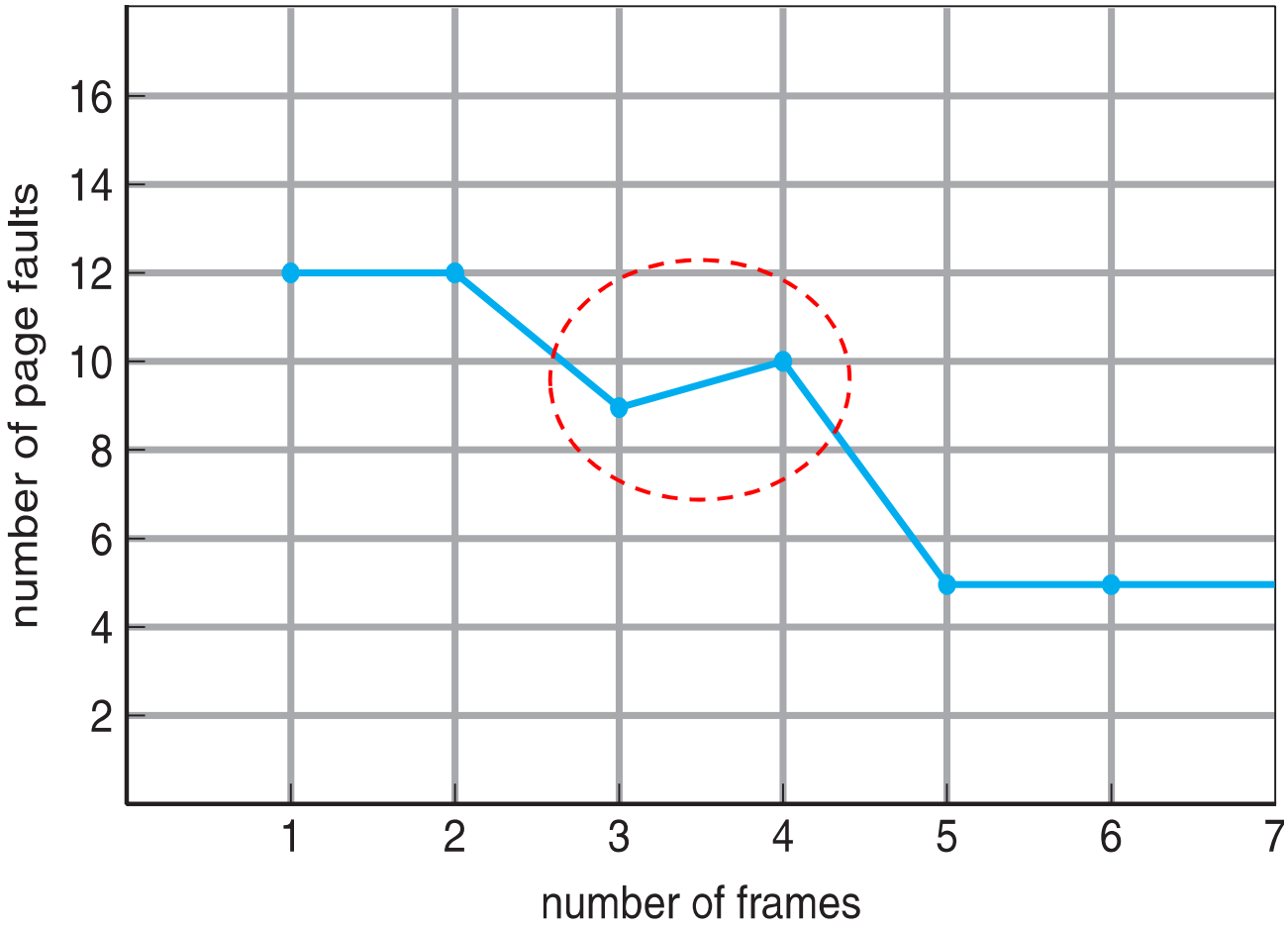


10 page faults

  - FIFO Replacement manifests *Belady*'s Anomaly: for some page-replacement algorithms, the page-fault rate may *increase* as the number of allocated frames increases. (*László Bélády*, 1969)

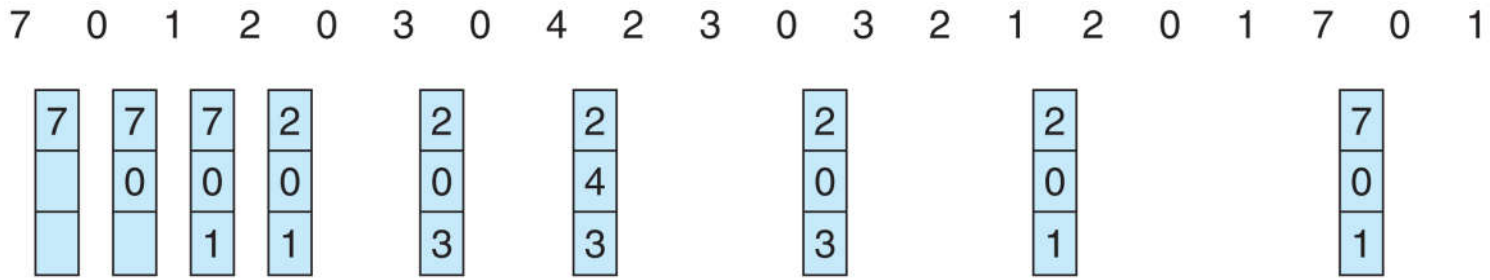
First-In-First-Out (FIFO) Algorithm

- FIFO Illustrating *Belady*'s Anomaly.



Optimal Algorithm

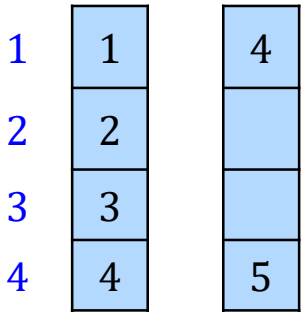
- Optimal Page Replacement
  - The Optimal policy selects for replacement the page that will not be used for longest period of time.
    - impossible to implement (need to know the future) but serves as a standard to compare with the other algorithms we shall study.
  - Example
    - Reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
    - 3 frames
    - 9 page faults is optimal for the demo reference string.



9 Page Faults

Optimal Algorithm

- Example
  - Reference string: 1 2 3 4 1 2 5 1 2 3 4 5
  - 4 frames:

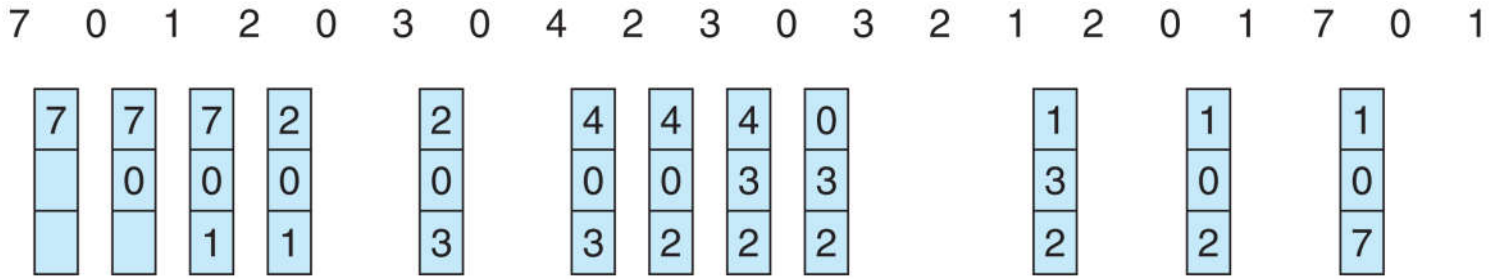


6 page faults

  - How do you know future use? You don't!
  - Used for measuring how well your algorithm performs.

Least Recently Used (LRU) Algorithm

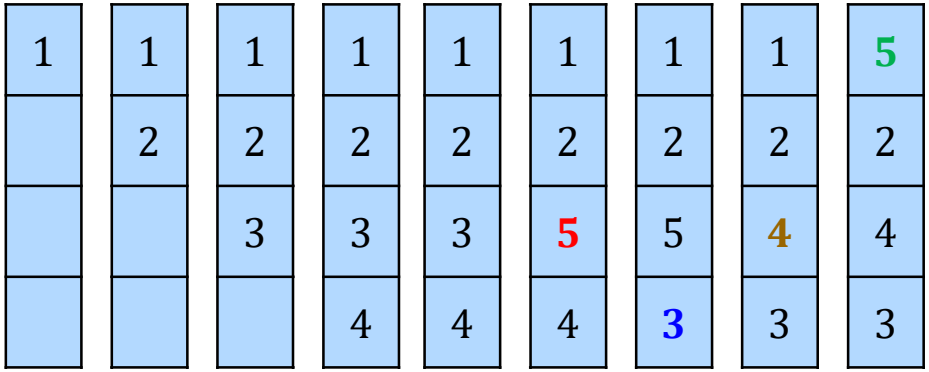
- The LRU Policy
  - Replaces the page that has not been referenced for the longest time.
    - Use past knowledge rather than future
    - Associate time of last use with each page
    - By the principle of locality, this should be the page least likely to be referenced in the near future.
    - performs nearly as well as the optimal policy.
  - 12 page faults for the demo – better than FIFO, worse than OPT
  - Generally good algorithm and frequently used



12 Page Faults

Least Recently Used (LRU) Algorithm

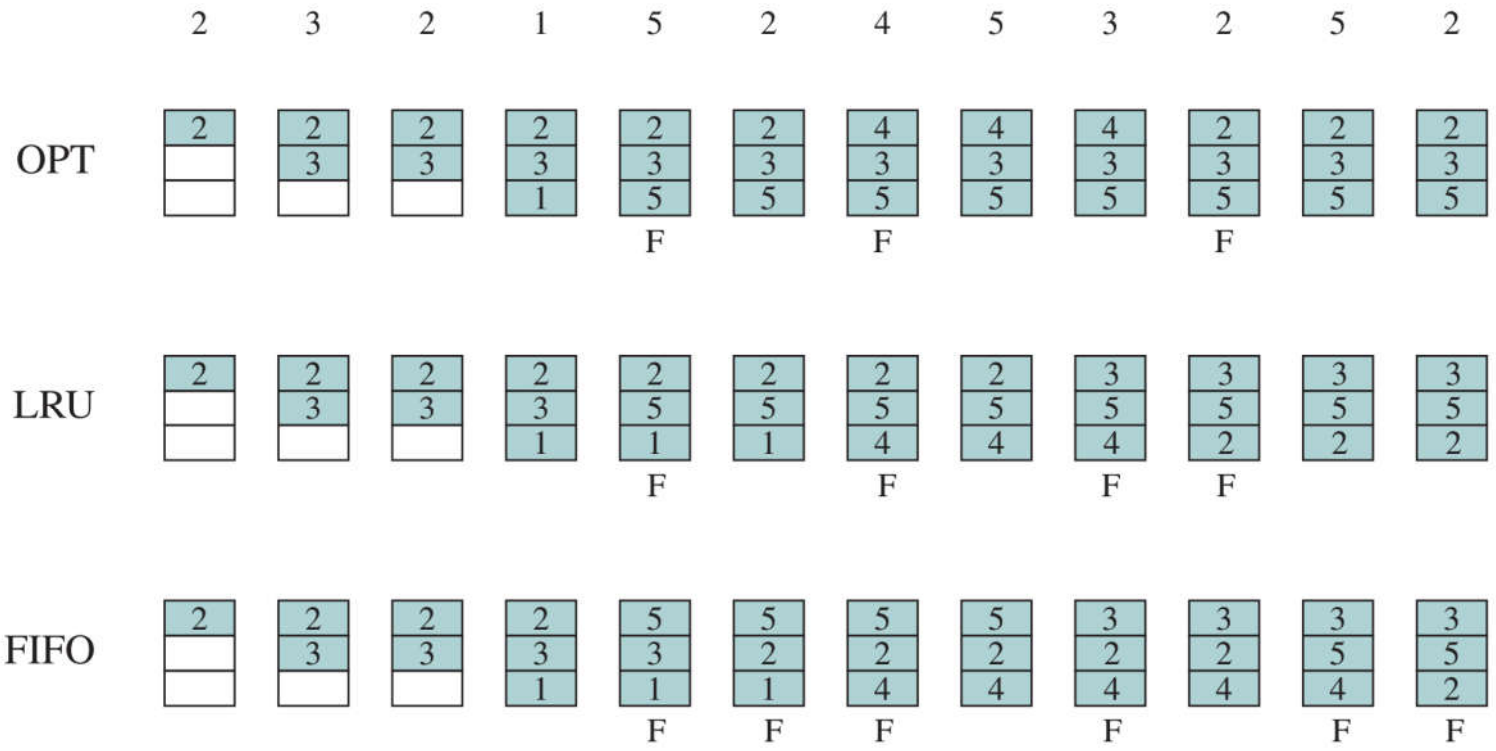
- Example
  - Reference string: 1 2 3 4 1 2 5 1 2 3 4 5
  - 4 frames:



8 page faults

Least Recently Used (LRU) Algorithm

- Comparison of OPT, LRU and FIFO
  - Example: A process of 5 pages with an OS that fixes the resident set size to 3.
  - Reference string: 2 3 2 1 5 2 4 5 3 2 5 2



F = page fault occurring after the frame allocation is initially filled

Least Recently Used (LRU) Algorithm

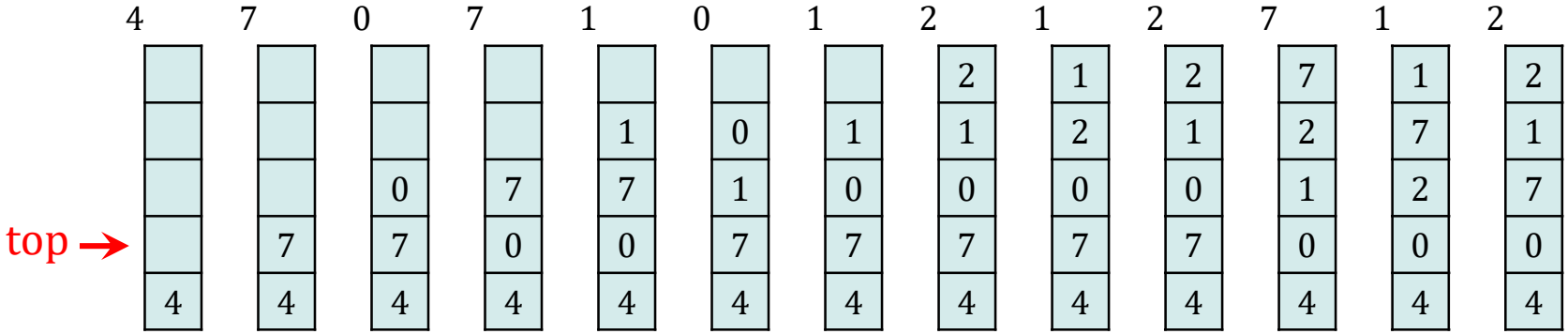
- Implementation of the LRU Policy
  - Each page could be *tagged* (in the page table entry) with the time at each memory reference.
  - The LRU page is the one with the smallest time value (needs to be *searched* at each page fault).
  - This would require expensive hardware and a great deal of overhead.
    - Consequently very few computer systems provide sufficient hardware support for true LRU replacement policy.
    - Other algorithms are used instead.

Least Recently Used (LRU) Algorithm

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

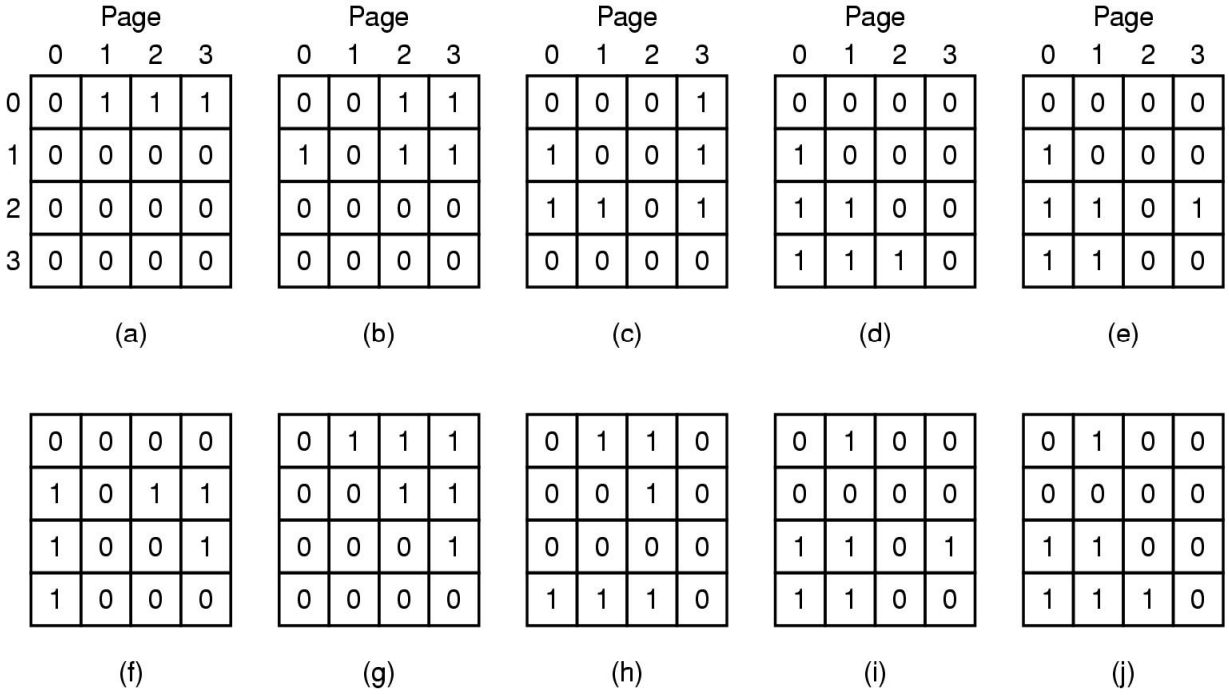
Least Recently Used (LRU) Algorithm

- LRU Implementations
  - Stack implementation
    - keep a stack of page numbers in a double link form.
    - When a page is referenced
      - move it to the top
      - requires 6 pointers to be changed
    - No search for replacement.
    - LRU and OPT are cases of *stack algorithms* that don't have *Belady*'s Anomaly.
    - Example
      - Reference string: 4 7 0 7 1 0 1 2 1 2 7 1 2



Least Recently Used (LRU) Algorithm

- LRU Implementations
  - Hardware Matrix LRU Implementation
    - Pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.
    - When page #i is referenced,
      (1) Set elements of row i to 1;
      (2) Set elements of volume i to 0.
    - The page with the smallest row summary is to be replaced.

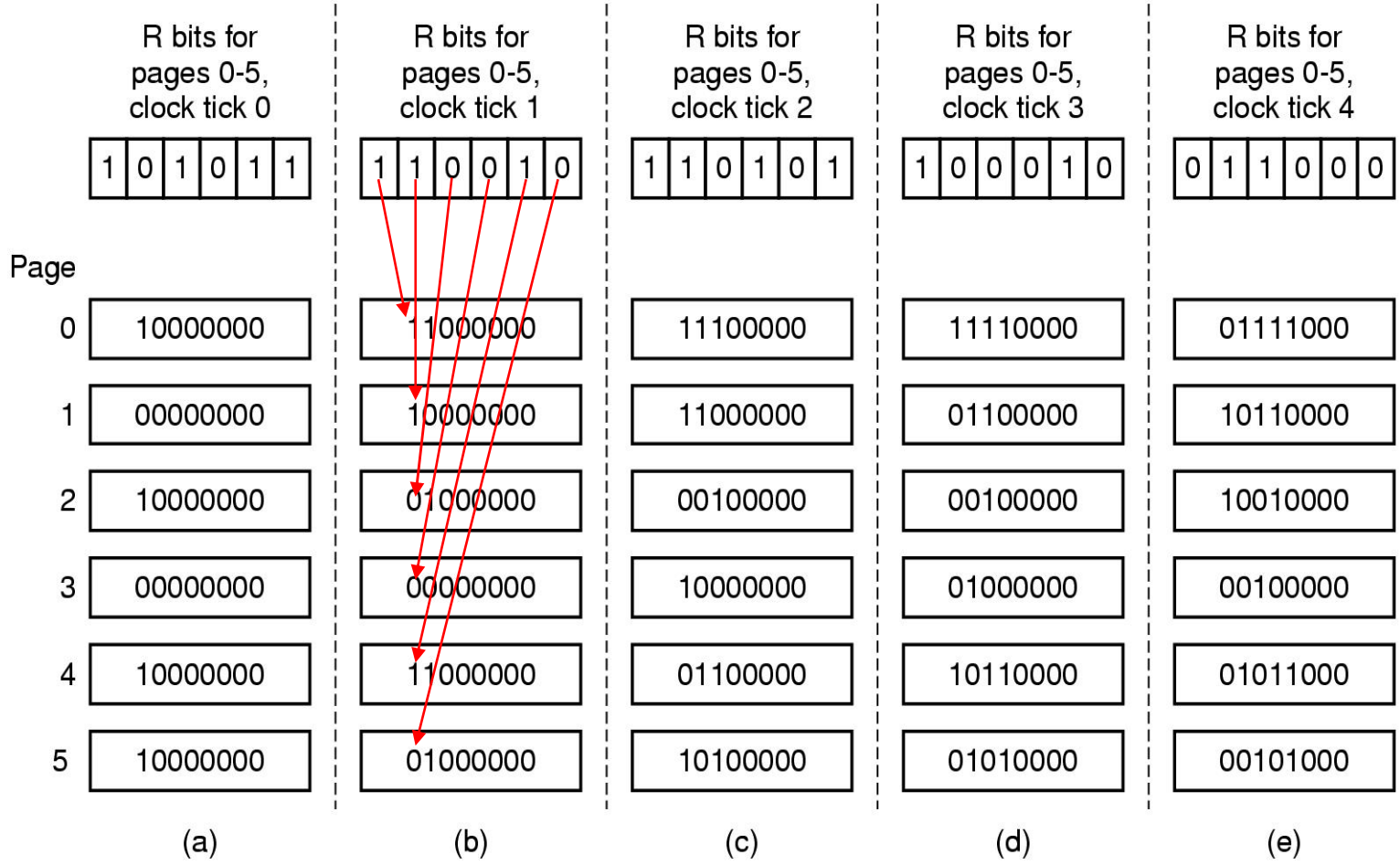


LRU Approximation Algorithms

- LRU needs special hardware and still slow.
- Reference Bit
  - with each page associate a bit, initially = 0.
  - When page is referenced, bit is set to 1.
  - Replace the page whose reference bit is 0 (if one exists)
    - we do not know the real order of use, however.
- Reference Byte
  - Record reference bits at regular intervals; Keep a byte of reference bits for each page.
  - At regular intervals (say, every 20ms), a timer interrupt transfers control to the operating system. The operating system shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifting the other bits right by 1 bit and discarding the low-order bit.
  - Each reference byte keeps the history of the page use (aging) for the last eight time intervals.
  - If we interpret the reference byte as an unsigned integer, the page with the lowest number is the LRU page.

LRU Approximation Algorithms

- Reference Byte
  - Example.

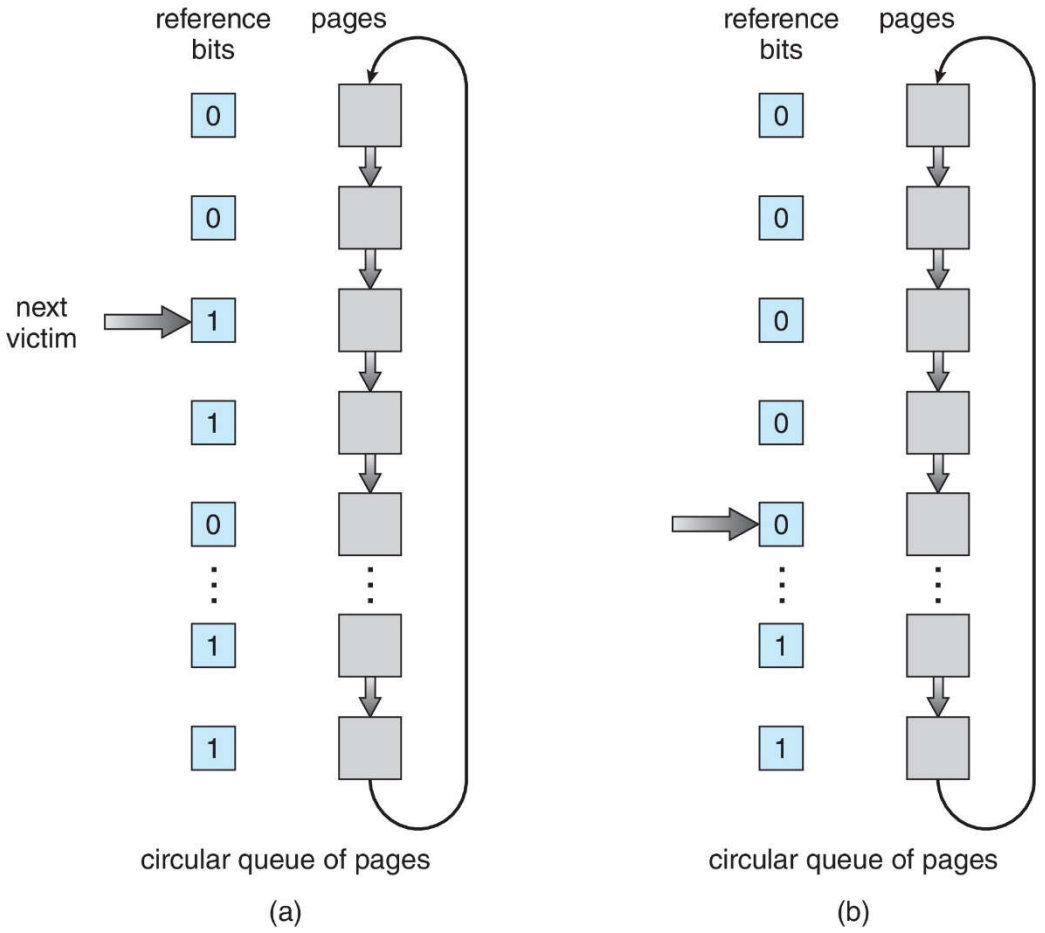


LRU Approximation Algorithms

- Second-Chance Algorithm
  - The set of frames candidate for replacement is considered as a circular buffer (sometimes the alg. is referred to as CLOCK alg.).
  - A hardware reference bit for each frame is set to 1 whenever:
    - a page is first loaded into the frame, or
    - the corresponding page is referenced.
  - The basic algorithm of second-chance replacement is FIFO. When examining a page, its reference bit is inspected.
  - The Second-Chance Policy as:
    - If the reference bit is 0, this page is selected to be replaced. The next frame will be examined in next replacement.
    - Otherwise the bit is cleared and the arrival time of the page is reset to the current time (i.e., the page is given a second chance). We move on to examine the next FIFO page. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chances).
    - In addition, if a page is used often enough to keep its reference bit set, it will never be replaced.

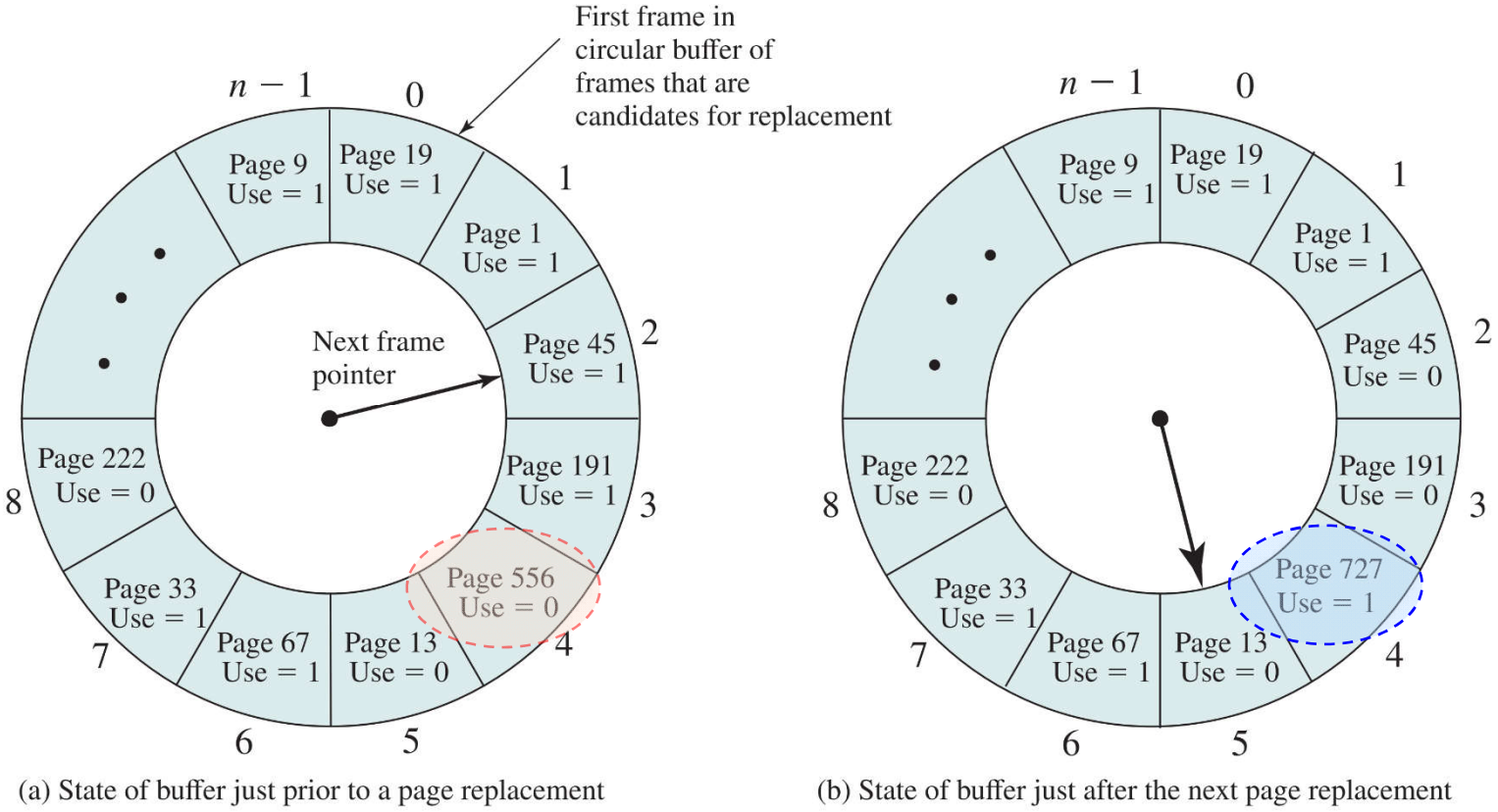
LRU Approximation Algorithms

- Second-Chance Algorithm
  - Example.



LRU Approximation Algorithms

- Second-Chance Algorithm
  - Another example of CLOCK policy operation.



(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

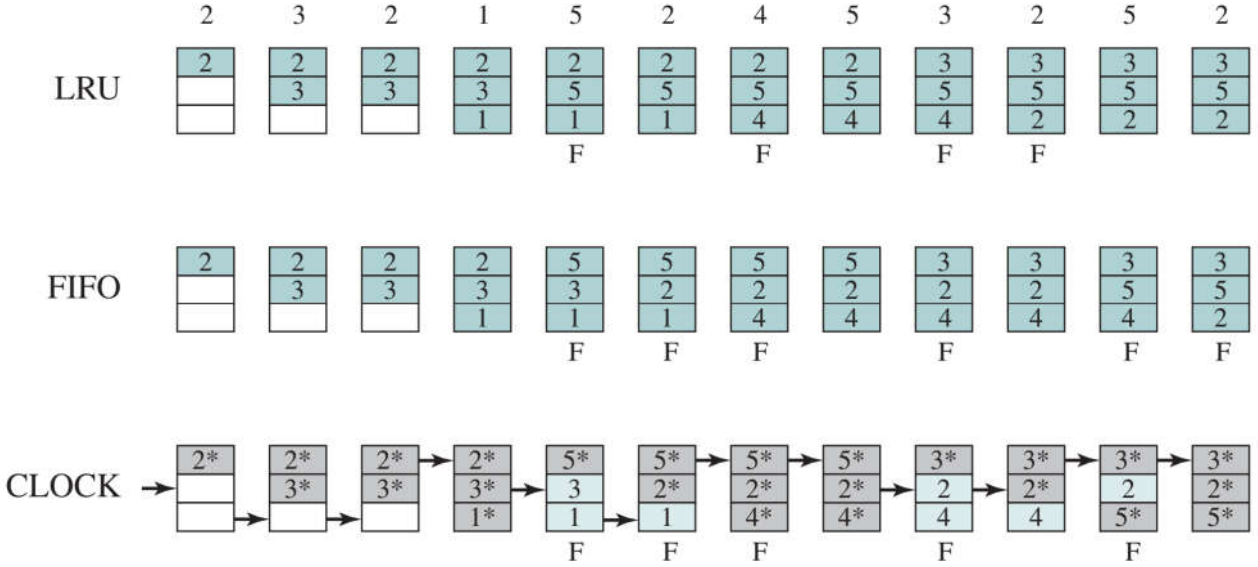
LRU Approximation Algorithms

- Enhanced Second-Chance Algorithm
  - Improve algorithm by considering the reference bit and the modify bit (if available) as an ordered pair

    <reference_bit, modify_bit>
  - We have the following four possible classes:
    - Class 0: <0, 0> neither recently used not modified – best page to replace.
    - Class 1: <0, 1> not recently used but modified – not quite as good, because the page will need to be written out before replacement.
    - Class 2: <1, 0> recently used but clean – probably will be used again soon.
    - Class 3: <1, 1> recently used and modified – probably will be used again soon and need to write out to secondary storage before replacement.
  - When page replacement is called for, use the clock scheme, replace the first page in the lowest non-empty class.
    - Might need to search circular queue several times.

LRU Approximation Algorithms

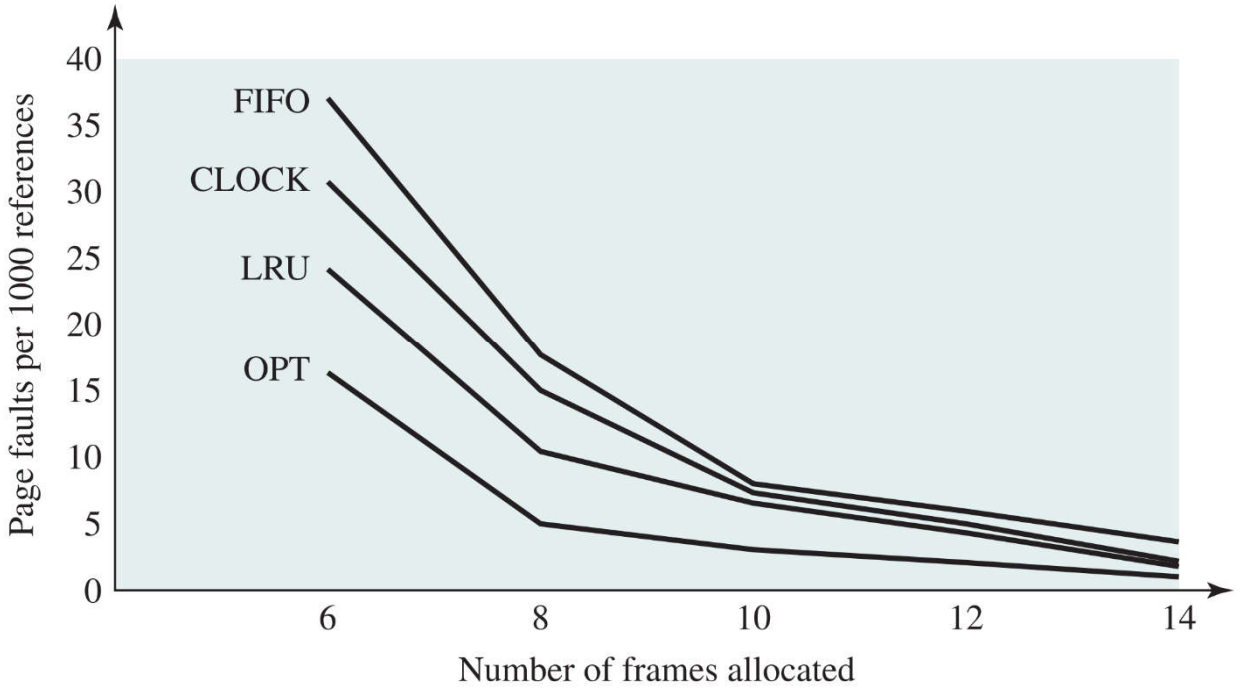
- Comparison of Clock with FIFO and LRU
  - Example: A process of 5 pages with an OS that fixes the resident set size to 3. Reference string: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2.



  - In the Clock scheme
    - Asterisk indicates that the corresponding use bit is set to 1.
    - The arrow indicates the current position of the pointer.
    - Note that the clock policy is adept at (擅长于) protecting frames 2 and 5 from replacement.

LRU Approximation Algorithms

- Comparison of Clock with FIFO and LRU
  - Fixed-Allocation, Local Page Replacement.
    - It is assumed that the number of page frames assigned to a process is fixed.



Counting-based Algorithms

- Let each page keep a counter as the accumulator of the number of references to the page.
- *Least Frequently Used* (LFU)
  - LFU algorithm replaces a page with the smallest count.
    - Others were and will be used more.
- *Most Frequently Used* (MFU)
  - MFU algorithm is based on the argument that the page with the smallest count was probably just brought in the main memory and has yet to be used.
  - Less commonly used.

Page-Buffering Algorithms

- Pages to be replaced are kept in main memory for a while to guard against poorly performing replacement algorithms such as FIFO.
- Two lists of pointers are maintained.
  - A free page list (unmodified list) for frames that have been selected but not been modified since brought in.
    - no need to be swapped out.
  - A modified page list for frames that have been selected and been modified.
    - need to write them out.
- A frame selected by algorithms such as FIFO has a pointer added to the tail of one of the lists (depending on its modify bit) and the present bit in the corresponding page table entry is cleared; but the page remains in the same memory frame.
  - When page fault, the reference page is not necessarily brought in the selected page, but the first frame in the free page list (see the next slide).

Page-Buffering Algorithms

- At each page fault the two lists are *first* examined to see if the needed page is still in main memory.
  - If it is, we just need to set the present bit in the corresponding page table entry, and remove the matching entry in the relevant list (of the free page list or the modified list).
  - If it is not, then the needed page is brought in. It is placed in the frame pointed by the head of the free page list (overwriting the page that was there); the head of the free frame list is moved to the next entry.
  - The frame number in the page table entry could be used to scan the two lists, or each list entry could contain the process ID and page number of the occupied frame.
- The modified page list also serves to write out modified pages in cluster rather than individually and moves the entries to the free page list.

Cleaning Policy

- When should a modified page be written out to disk?
- Demand cleaning
  - A page is written out only when it's frame has been selected for replacement.
    - But a process that suffers a page fault may have to wait for 2 page transfers.
- Pre-cleaning
  - Modified pages are written before their frames are needed so that they can be written out in batches.
    - But it makes little sense to write out so many pages if the majority of them will be modified again before they are replaced.

Cleaning Policy

- A good compromise can be achieved with page buffering.
  - Recall that pages chosen for replacement are maintained either on a free (unmodified) page list or on a modified page list.
  - Pages on the modified page list can be periodically written out in batches and moved to the free page list.
  - A good compromise since:
    - Not all dirty pages are written out but only those chosen for replacement.
    - Writing is done in cluster.

Combined Segmentation and Paging

- Comparison of Paging and Segmentation.

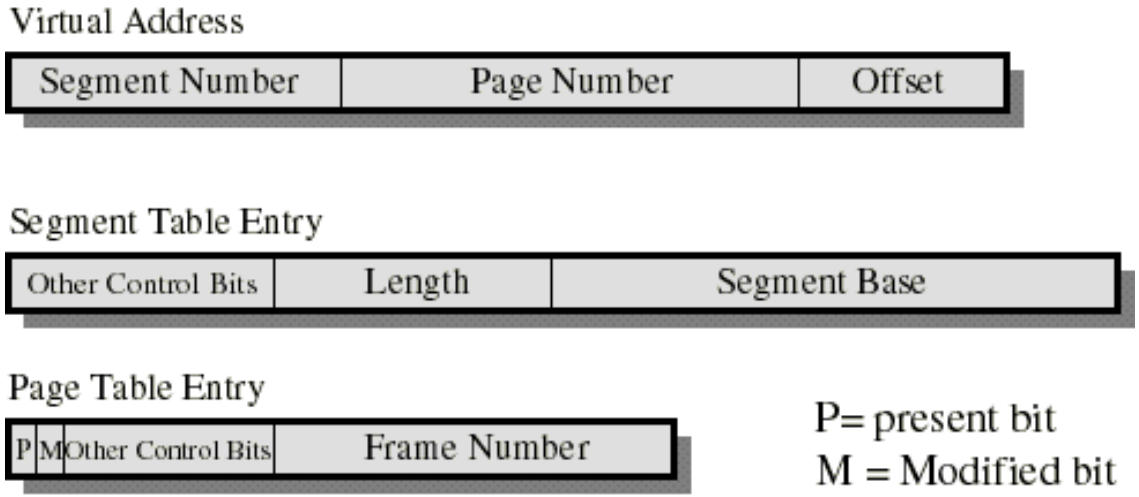
Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can 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

Combined Segmentation and Paging

- To combine their advantages, some OSs page the segments.
- Several combinations exist – assume each process has:
  - one segment table.
  - several page tables: one page table per segment.
- The virtual address consists of:
  - a segment number: used to index the segment table who's entry gives the starting address of the page table for that segment.
  - a page number: used to index that page table to obtain the corresponding frame number.
  - an offset: used to locate the word within the frame.

Simple Combined Segmentation and Paging

- The Segment Base is the physical address of the page table of that segment.
- Present/modified bits are present only in page table entry.
- Protection and sharing info most naturally resides in segment table entry.
  - Ex: a read-only/read-write bit, a kernel/user bit...



Simple Combined Segmentation and Paging

- Address Translation in Combined Segmentation and Paging.