CSCI315 – Operating Systems Design

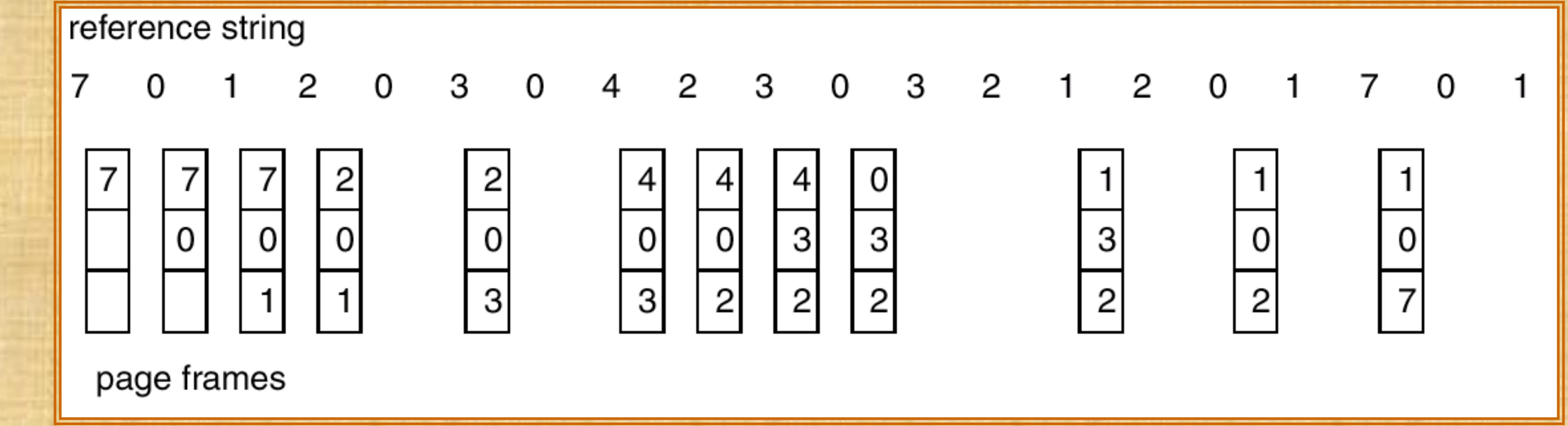
Department of Computer Science
Bucknell University

Page Replacement Algorithms – 2

Ch 10.4-10.5

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

LRU Page Replacement



reference string

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

page frames

Optimal: 9 page faults

LRU: 12 page faults

LRU Algorithm Implementation

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - Bottom one is to be replaced first.
  - No search for replacement.

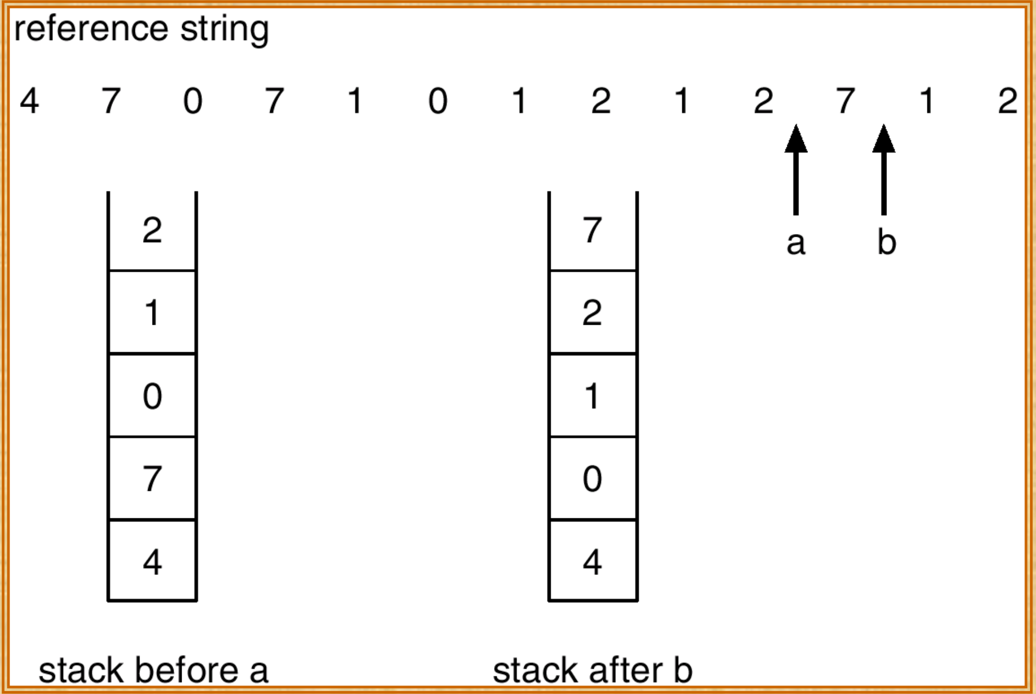
LRU and Belady’s Anomaly

- LRU does not suffer from Belady’s Anomaly (OPT doesn’t either).

- It has been shown that algorithms in a class called **stack algorithms** can never exhibit Belady’s Anomaly.

- A **stack algorithm** is one for which the set of pages in memory for n frames is a subset of the pages that could be in memory for $n+1$ frames.

Use Stack to Record The Most Recent Page References



reference string

4 7 0 7 1 0 1 2 1 2 7 1 2

stack before a

stack after b

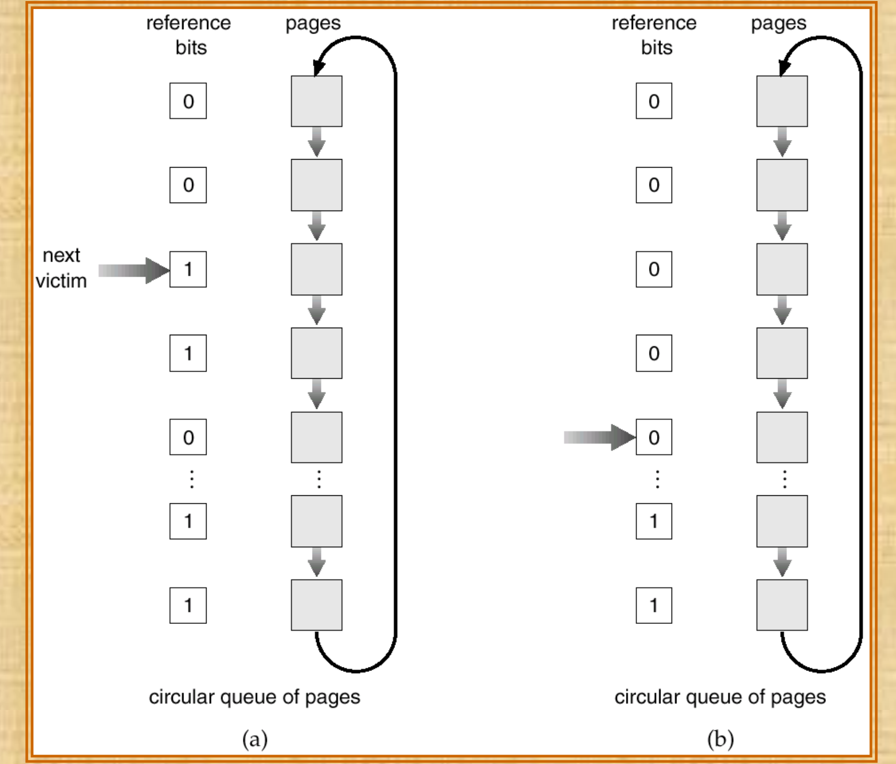
LRU Approximation Algorithms

- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- **Additional reference bits (e.g., 8 bits)**
  - Every time a page is referenced
    - Shift the reference bits to the right by 1
    - Place the reference bit (1 if being visited, 0 otherwise) into the high order bit of the reference bits
    - The page with the lowest reference bits value is the one that is Least Recently Used, thus to be replaced
  - E.g., the page with ref bits 11000100 is more recently used than the page with ref bits 01110111

LRU Approximation Algorithms

- **Second Chance**
  - If we consider the number of reference history bits to be zero, only using the reference bit itself, we have the *Second Chance* (a.k.a. *Clock*) algorithm
  - Need a pointer (clock handle) to point the next victim.
  - At each clock interruption, we check the reference bit for the victim.
  - If the victim page has reference bit = 1, then:
    - set reference bit 0.
    - leave this page in memory.
  - Else if the page reference bit is 0, this page can be replaced.

Second-Chance (Clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page.

- **LFU Algorithm:** replaces page with smallest count.

- **MFU Algorithm:** based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- Each process needs a **minimum** number of pages.

- There are two major allocation schemes:
  - **fixed allocation**
  - **priority allocation**

Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process.

$$
\begin{aligned}
s_i &= \text{size of process } p_i \\
S &= \sum s_i \\
m &= \text{total number of frames} \\
a_i &= \text{allocation for } p_i = \frac{s_i}{S} \times m
\end{aligned}
\left\{
\begin{aligned}
m &= 64 \\
s_i &= 10 \\
s_2 &= 127 \\
a_1 &= \frac{10}{137} \times 64 \approx 5 \\
a_2 &= \frac{127}{137} \times 64 \approx 59
\end{aligned}
\right.
$$

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.

- If process P_i generates a page fault,
  - select for replacement one of its frames.
  - select for replacement a frame from a process with lower priority number.

Global vs. Local Allocation

- **Global** replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.

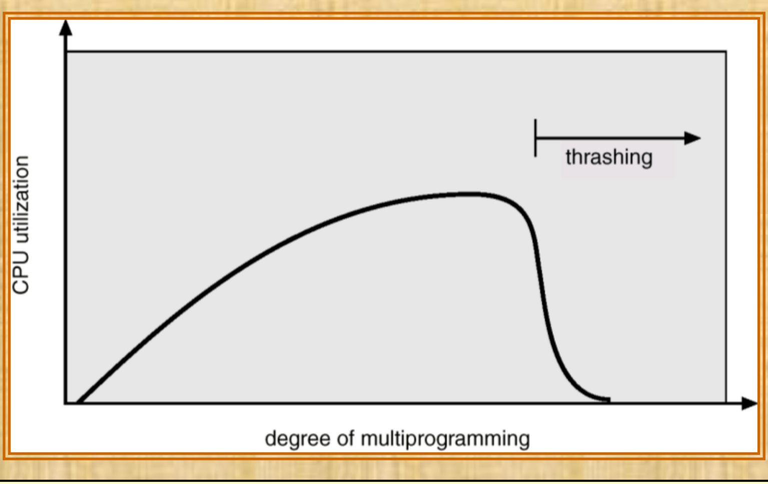
- **Local** replacement – each process selects from only its own set of allocated frames.

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - **Low CPU utilization**.
  - Operating system thinks that it needs to increase the degree of multiprogramming.
  - Another process added to the system.

- **Thrashing** ≡ a process is busy swapping pages in and out.

Thrashing



- Why does paging work?
  Locality model
  - Process migrates from one locality to another.
  - Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size