COMP 3511
Operating Systems

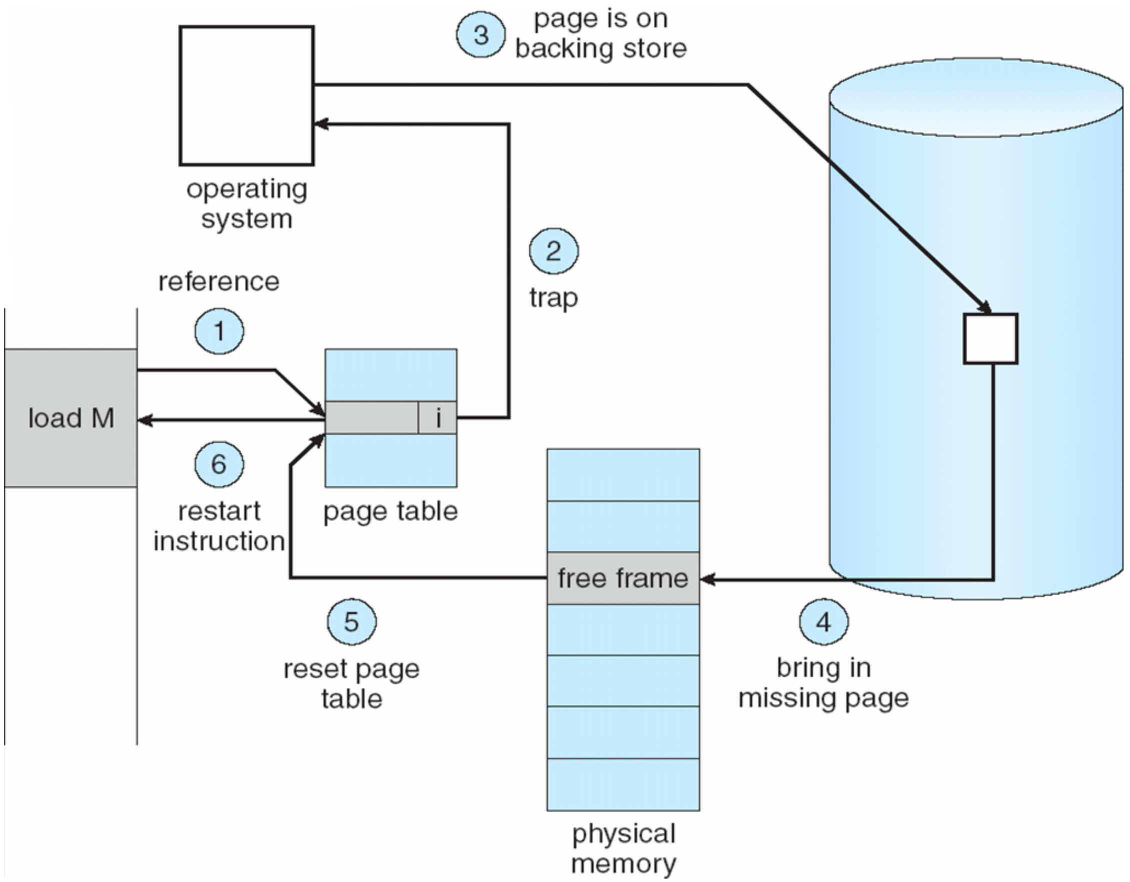
Lab 07

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed → reference to it
  - Invalid reference → abort
  - Not-in-memory → bring to memory

Steps in Handling a Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system: page fault



An example of demand-paged memory

- Assume we have a demand-paged memory
  - The page table is held in registers
  - It takes 8 milliseconds to serve a page fault if an empty page is available or the replaced page is not modified
  - It takes 20 milliseconds if the replaced page is modified
  - Memory access time is 100 nanoseconds

An example of demand-paged memory

- Assume that the page to be replaced is modified 70 percent of the time.

- *What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds ?*

An example of demand-paged memory

$$0.2\mu\text{sec} = (1 - P) \times 0.1\mu\text{sec} + (0.3P) \times 8 \text{ millisec} + (0.7P) \times 20 \text{ millisec}$$

$$0.1 = -0.1P + 2400\,P + 14000\,P$$

$$0.1 \approx 16{,}400\,P$$

$$P \approx 0.000006$$

Hardware support for demand paging

- For every memory access operation, the page table needs to be consulted:
  - check whether the corresponding page is resident or not
  - check whether the program has read or write privileges for accessing the page.

Hardware support for demand paging

- These checks would have to be performed in hardware.
- For example, a **TLB** could serve as a cache and improve the performance of the lookup operation.

Page Replacement

- If there is no free frame
- Page replacement – find some page in memory, but not really in use, swap it out
  - Replacement algorithm
  - Performance requirement – want an algorithm which will result in minimum number of page faults

Page Replacement

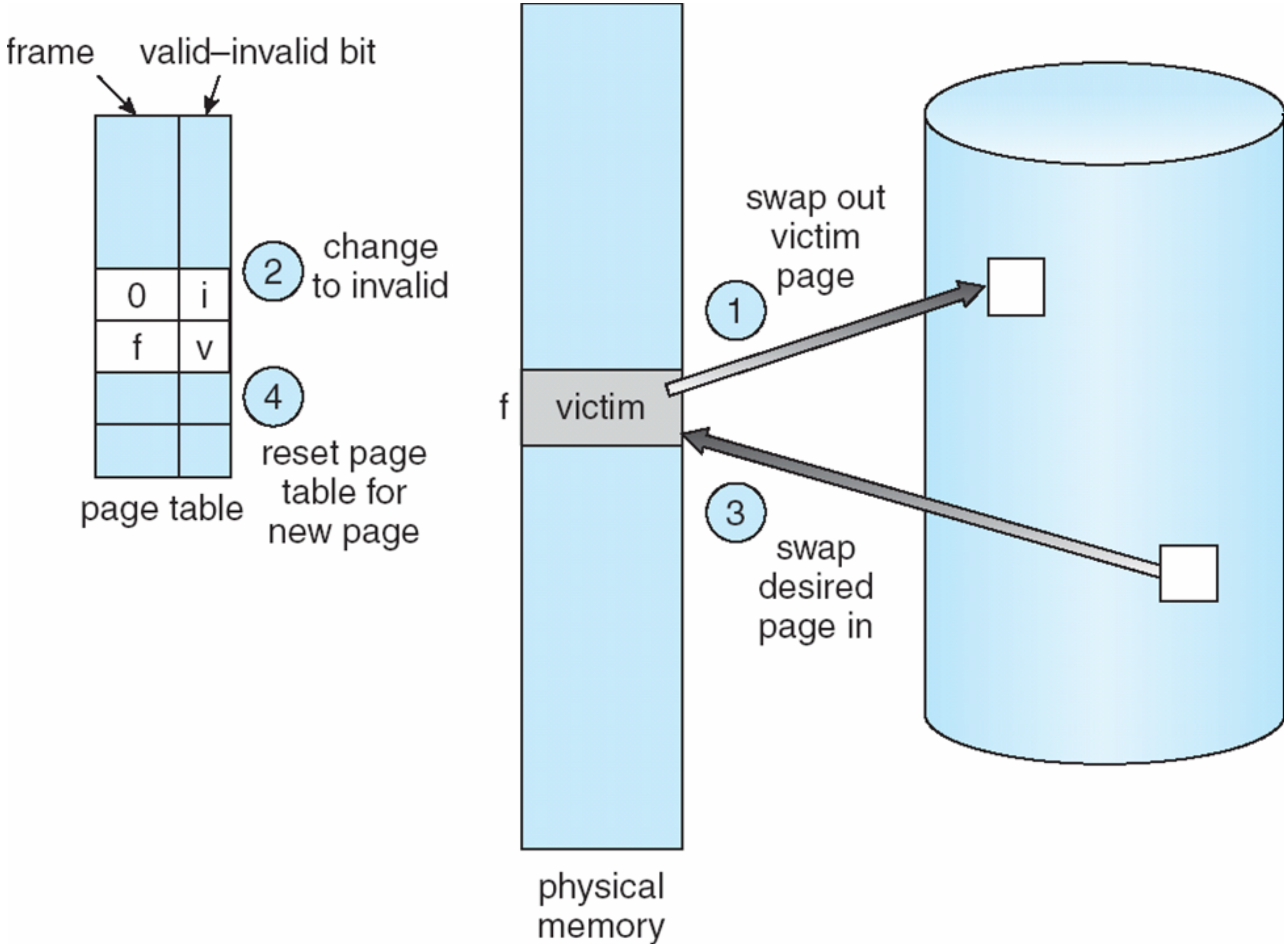
1. Find the location of the desired page on disk

2. Find a free frame inside the memory:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame

3. Read the desired page into the (newly) free frame. Update the page and frame tables.

4. Restart the process

Page Replacement



First-In-First-Out (FIFO) Algorithm

- 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)

1	1	4	5
2	2	1	3
3	3	2	4

9 page faults

- 4 frames

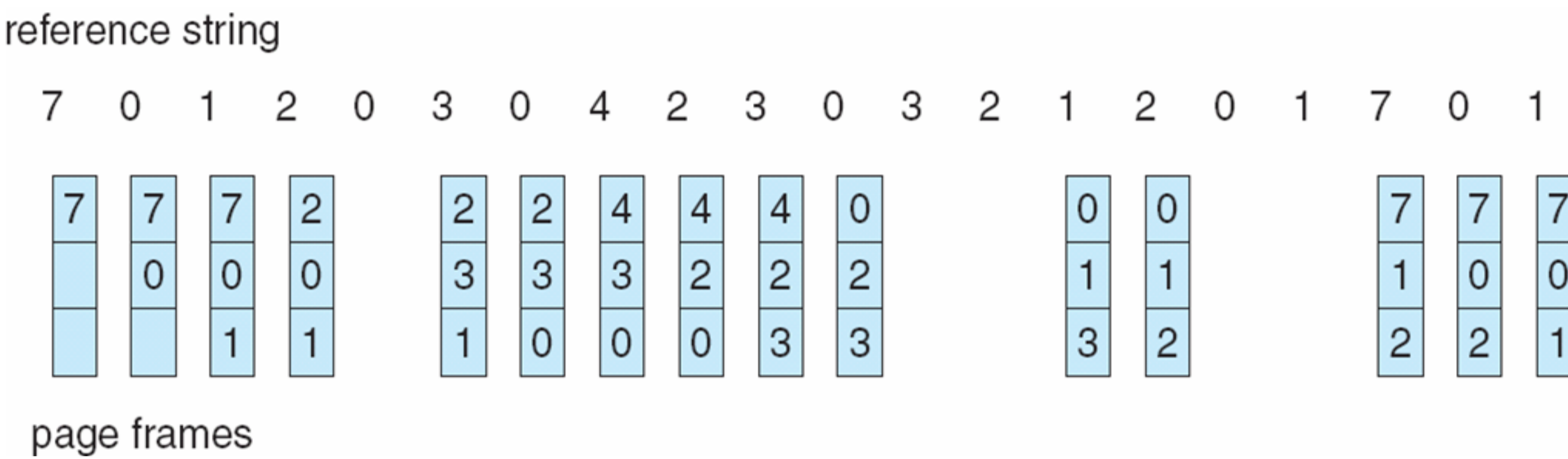
1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page faults

Belady's Anomaly:

more frames →
more page faults

FIFO Page Replacement



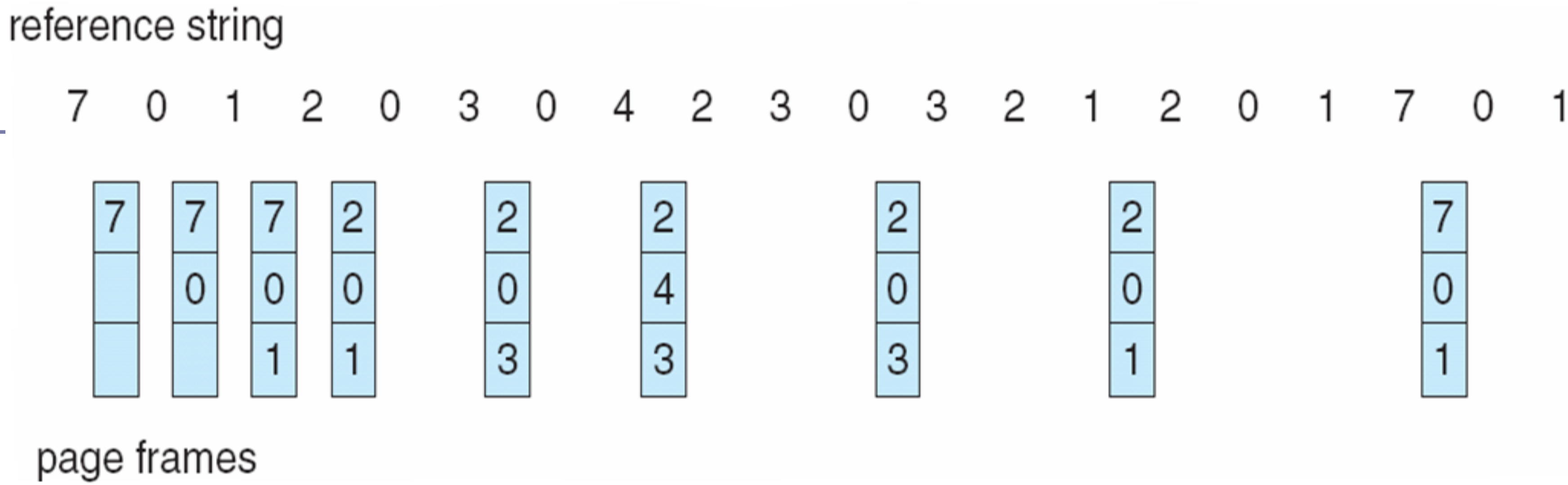
Optimal page replacement algorithm

- Replace page that will not be used for longest period of time
- Problem
  - Can’t read the future
- Used for measuring how well your algorithm performs

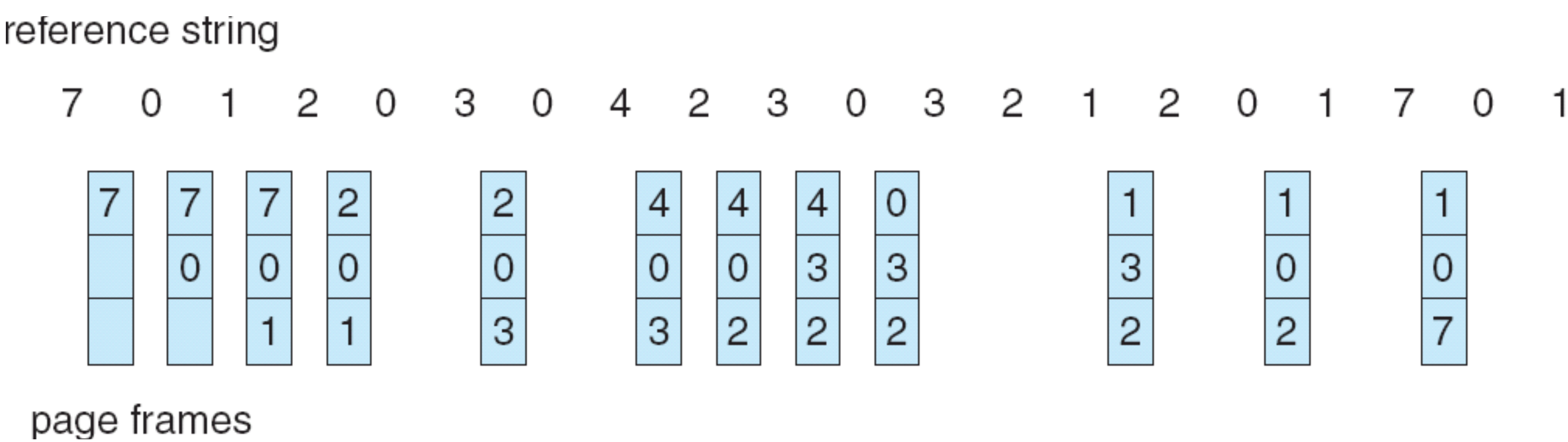
Algorithms for approximating optimal page replacement

- **LRU** (Least Recently Used) algorithm
  - Use the recent past as an approximation of the near future
    - Replace the page that has not been used for the longest period of time
  - Considered to be good, but hard to implement
    - Few computer systems provide sufficient hardware support for true LRU
    - LRU-approximation: Reference bits, Second chance

- Optimal page replacement (9 page faults)



- LRU page replacement (12 page faults)

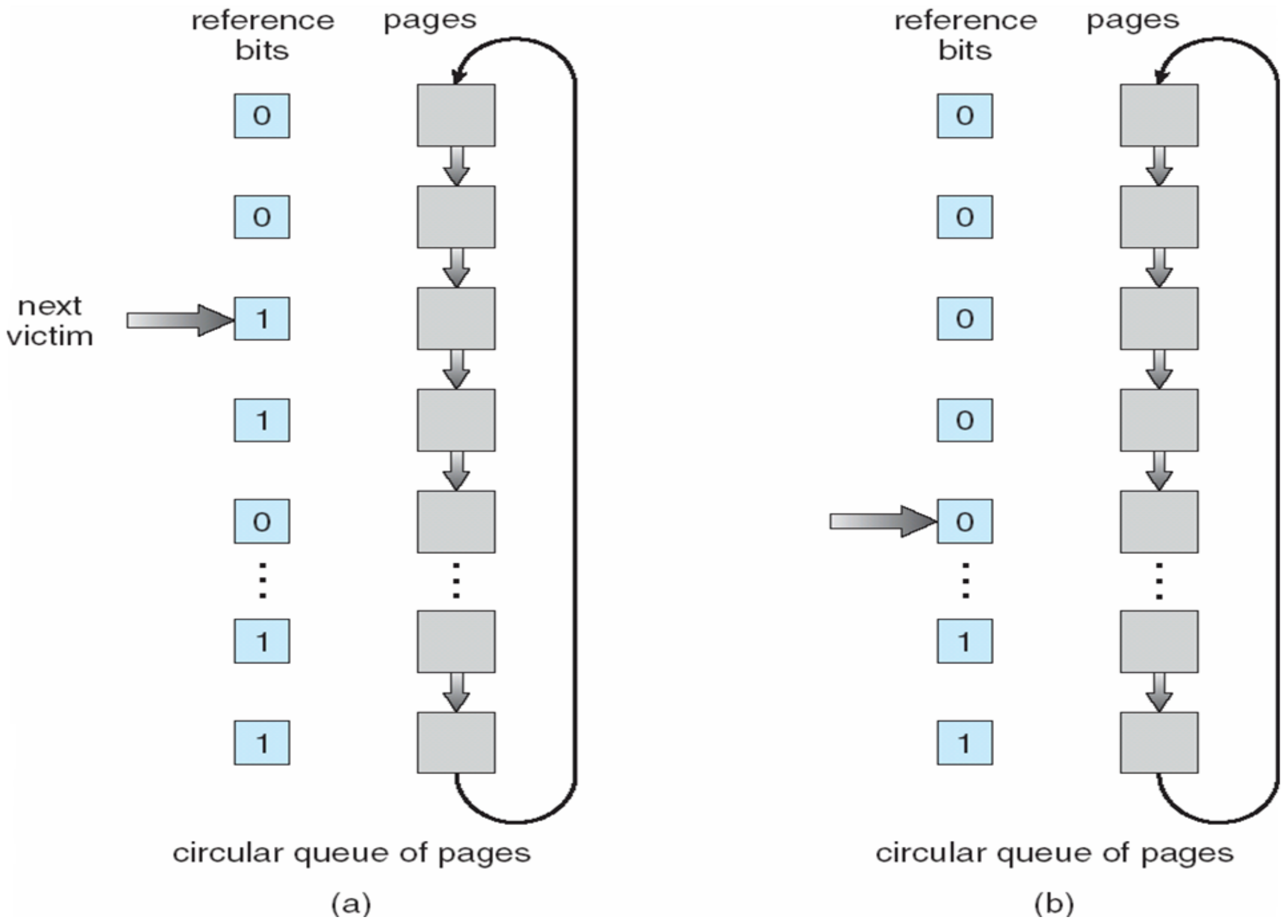


LRU Approximation Algorithms

- Reference bit
  - Each page is associate a reference bit, initially = 0.
  - When page is referenced, the reference bit set to 1.
  - We can determine which pages have been used or not used by examining the reference bits and replace the page whose reference bit is 0 (if one exists).
  - However, we do not know the *order* of use.

- Second chance
  - When a page is selected, check the reference bit.
  - If the value is 0, this page is replaced.
  - If the value is 1, set the bit to 0, then move on to select another page (FIFO). This page gets a second chance.

Second chance algorithm



Algorithms for approximating optimal page replacement

- Keep a counter of the number of references that have been made to each page
  - **LFU** (Least Frequently Used) algorithm
    - an actively used page should have a large reference count
    - replaces page with smallest count
  - **MFU** (Most Frequently Used) algorithm
    - the page with the smallest count was probably just brought in and has yet to be used
  - The implementation of these algorithms is expensive, and they do not approximate OPT replacement well

An example of page-replacement algorithm

***MFU**: most frequently used page-replacement algorithm*

VS.

***LRU**: least recently used page replacement algorithm*

- (a) Under which situations, the **MFU** generates fewer page faults than the **LRU** ? Please give an example.
- (b) Under what circumstance does the opposite holds ?

Example 1: MFU & LRU

- (a) Consider the sequence in a system that holds four pages in memory: ***1 2 3 4 4 4 5 1***
  - the **MFU** evicts page 4 while fetching page 5
  - the **LRU** evicts page 1 while fetching page 5, then another page fault for fetching page 1 again
- (b) For the sequence "***1 2 3 4 4 4 5 4***," the **LRU** algorithm makes the right decision

Example 2: LFU & LRU

- Consider the following sequence of memory accesses in a system that can hold four pages in memory: *1 1 2 3 4 5 1*

- Which of the following page replacement algorithms generates fewer page faults?

  - a) LRU: least recently used page replacement
  - b) LFU: least frequently used page replacement

Example 2: LFU & LRU

- memory accesses: *1 1 2 3 4 5 1*
- four pages in memory
- The LRU evicts page 1 while fetching page 5
- The LFU evicts a page other than 1 while fetching page 5
  - No page fault when page 1 is accessed again
  - fewer page faults