Operating Systems: Virtual Memory Part I

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapter 10)

Virtual Memory

- All the memory management strategies discussed so far, required the entire program to be in memory before its executed.
- Entire program rarely used/needed at the same time.
- **Virtual Memory** is a technique that allows the execution of processes that are not completely in memory.

Standard Swapping

- **Standard Swapping:** Moving a process temporarily out of memory to a backing store, and then bring it back into memory for continued execution
- **Backing store is a fast secondary storage/disk space**
  - large enough
  - provides direct access
- Main issues with standard swapping is the time taken to switch data related to a process is high
- Not common in modern OS.

Swapping with paging

- **Swapping with paging**: pages of a process—rather than an entire process —are swapped.
- Less expensive than standard swapping.
- **Swapper** - swaps processes from disk to memory.
- **Pager** – swaps pages of processes from disk to memory.
- Most systems, including Linux and Windows.
- **Page out** operation moves a page from main memory to the backing store
- **Page in** operation moves a page from backing store to main memory.

Demand Paging

- Virtual memory is commonly implemented via **Demand paging.**
- **Demand paging**:
  - When a process is swapped in (from the disk), all its pages are not swapped in all at once.
  - Instead, the pager **guesses** which pages will be used before the program is swapped out again and brings those pages in.
- Programs tend to have *locality of reference and* guessing of the pager is based on it to improve performance.

Locality

- Processes reference pages in **localized patterns**
- **Temporal Locality** - Locations referenced recently likely to be referenced again
- **Spatial Locality** - Locations near recently referenced locations are likely to be referenced soon.
- Processes usually exhibit both kinds of locality.

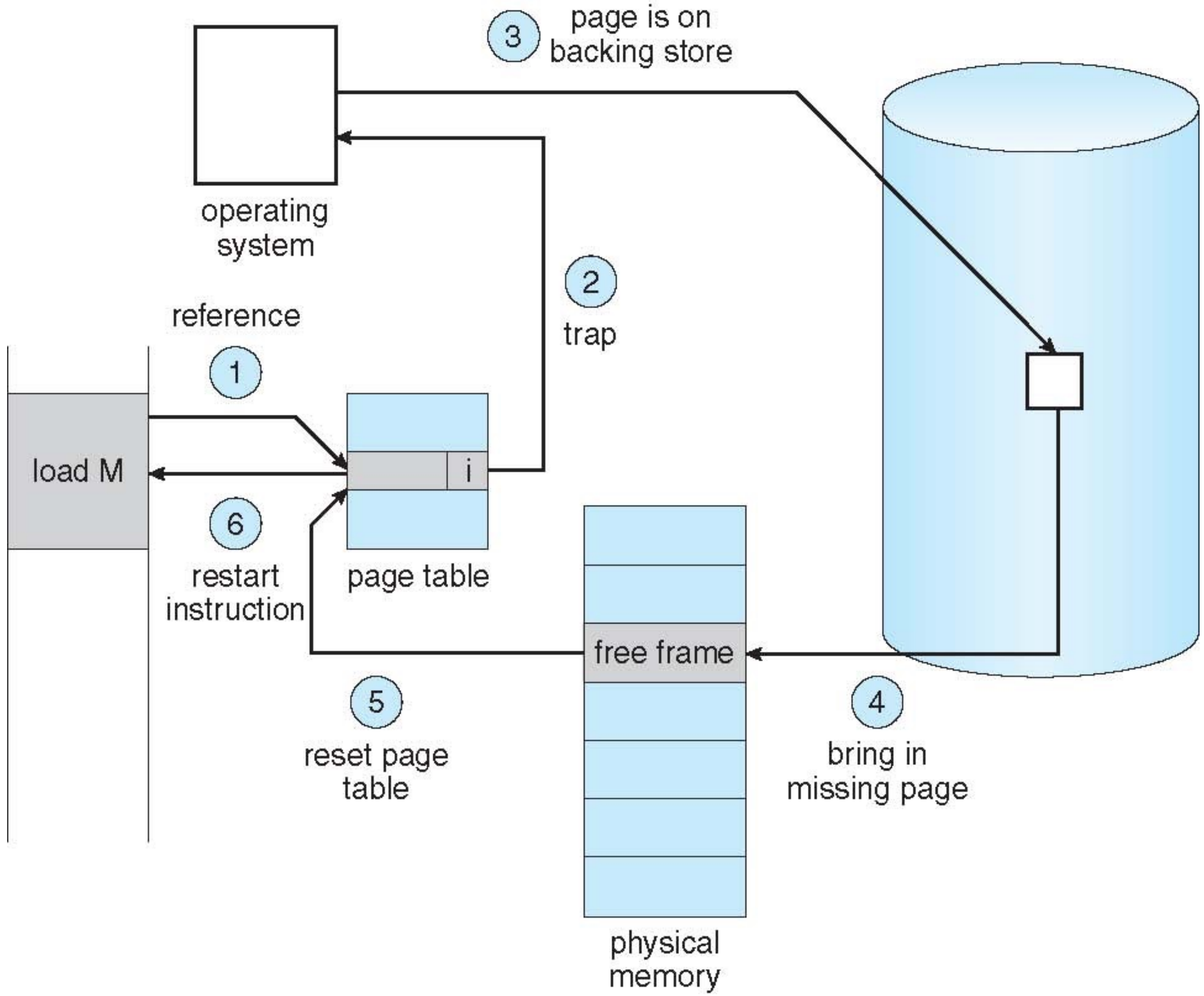
Demand Paging Cont…

- **Pure demand paging:** Process is started with ***no*** pages in memory.
  - OS sets instruction pointer to first instruction of process.
  - Since it is non-memory-resident, page fault occurs.
  - And for every other process pages on first access page fault occurs.
- Demand paging needs hardware support to distinguish between pages in memory and those on disk.
  - Use the **valid-invalid bit scheme**
  - **Valid bit** – indicates the page is legal and in memory
  - **Invalid bit** – indicates the page is either illegal or is legal but not in memory.
- If a process tries to access a page with an invalid bit, it will cause a **page fault**.

Procedure to handle Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system; that is, result in a **page fault**
- Operating system looks up an internal table (in PCB) to decide:
  - Invalid reference $\Rightarrow$ abort
  - Just not in memory
- If not in memory, OS finds a free frame
- Swap page into frame from backing store.
- Reset tables to indicate page now in memory
  - Set validation bit = **v**
- Restart the instruction that caused the page fault

Procedure to handle Page Fault



Page Fault Cont…

What happens if a page fault occurs and no free frames in memory?

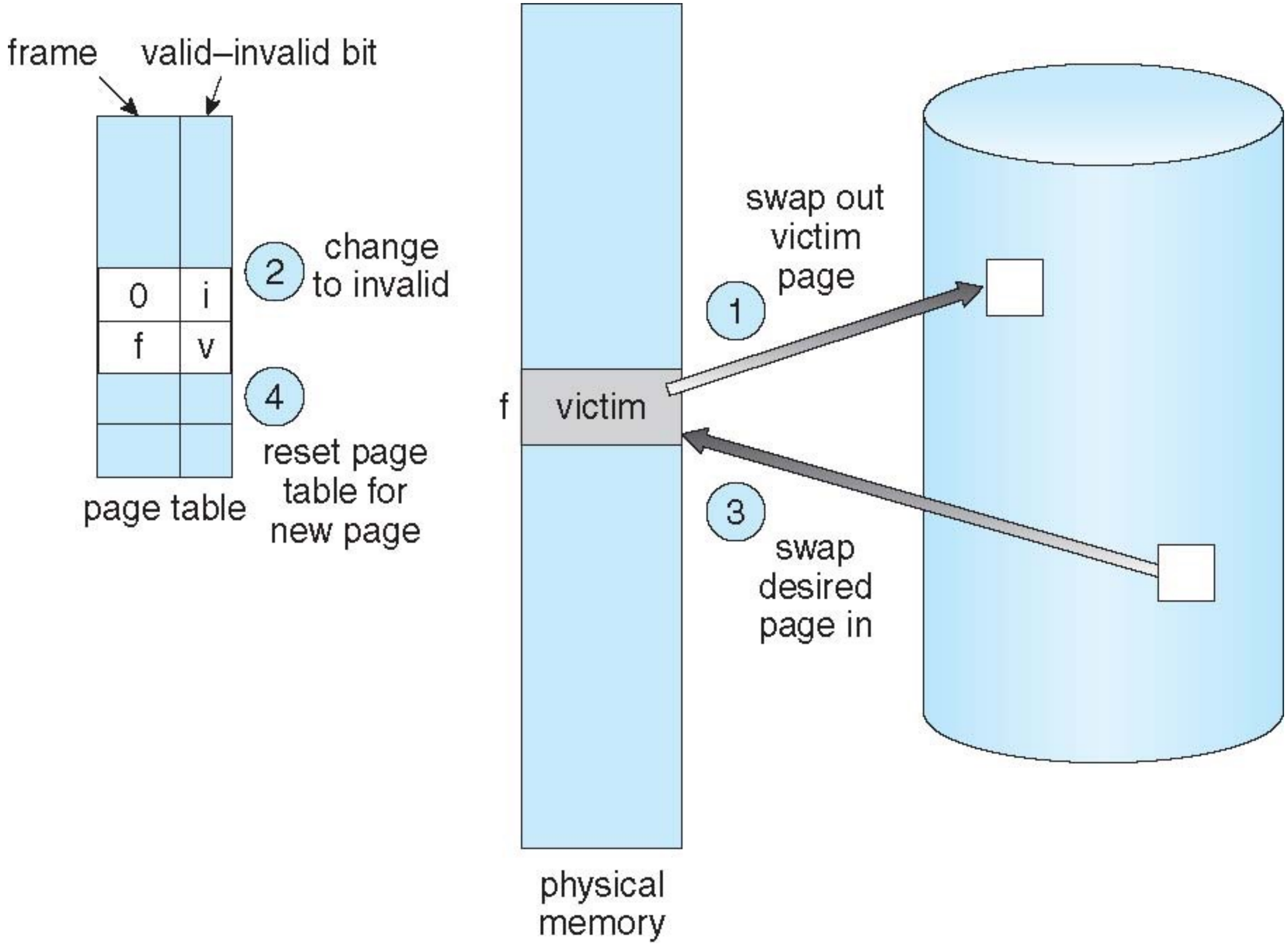
Replace an existing page with the new page!

- Different page replacement algorithms exist.
  - For example: FIFO, LRU, and Optimal.
- Note that in this case **two page transfers take place**,
  - One to copy the victim page back to the swap space, and
  - The other to bring in the new page to memory
- Therefore, demand paging increases the effective access time (EAT)!

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
    1. If there is a free frame, use it
    2. If there is no free frame, use a page replacement algorithm to select a **victim page**
    3. Write victim page to disk if dirty (modified)
3. Bring the desired page into the (newly) freed frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Page Replacement

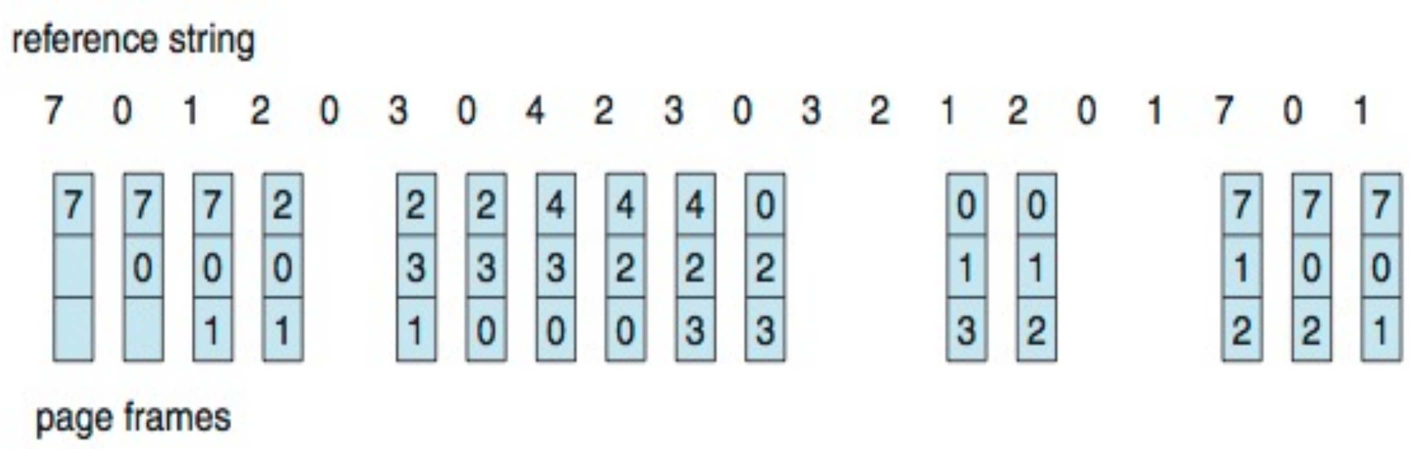


Page Replacement Algorithms

- We will study 3 different page replacement algorithms
  - FIFO Algorithm
  - Optimal Algorithm
  - LRU Algorithm
    - Second Chance (Clock Algorithm)
- In all the examples, the reference string of referenced page numbers is:
- **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

First-In-First-Out (FIFO) Algorithm

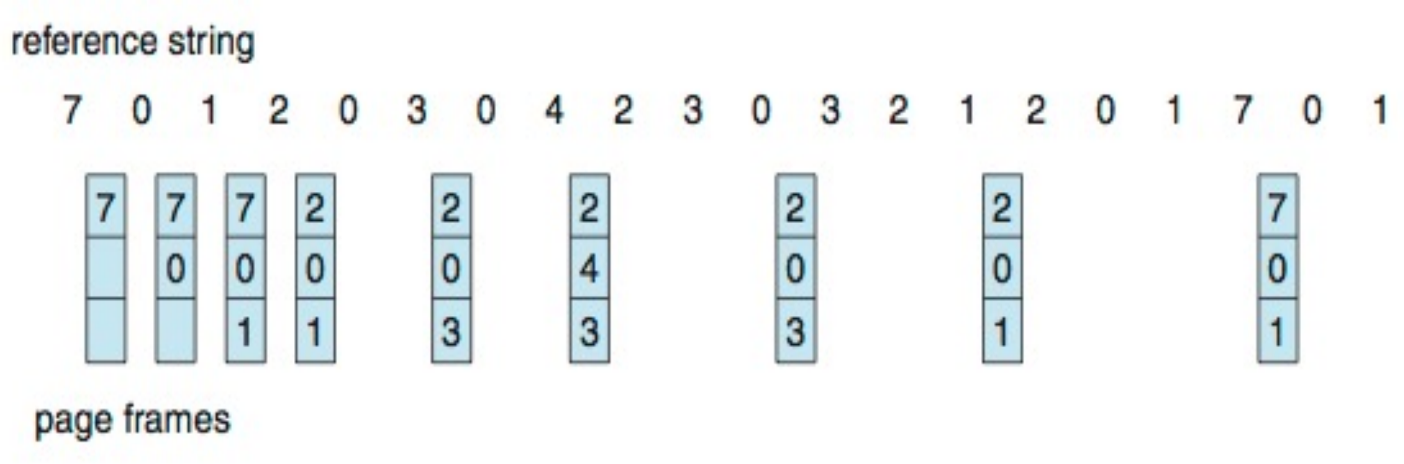
- Replace the **oldest page in memory**
- 3 frames (3 pages can be in memory at a time per process)



- **Note that when page 3 is referenced for the first time, page 0 is replaced, although it was recently referenced.**
- 15 page faults
- Page faults can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5 (check this as an exercise with 3, 4 frames)
  - Adding more frames can cause more page faults!
    - **Belady's Anomaly**

Optimal Algorithm

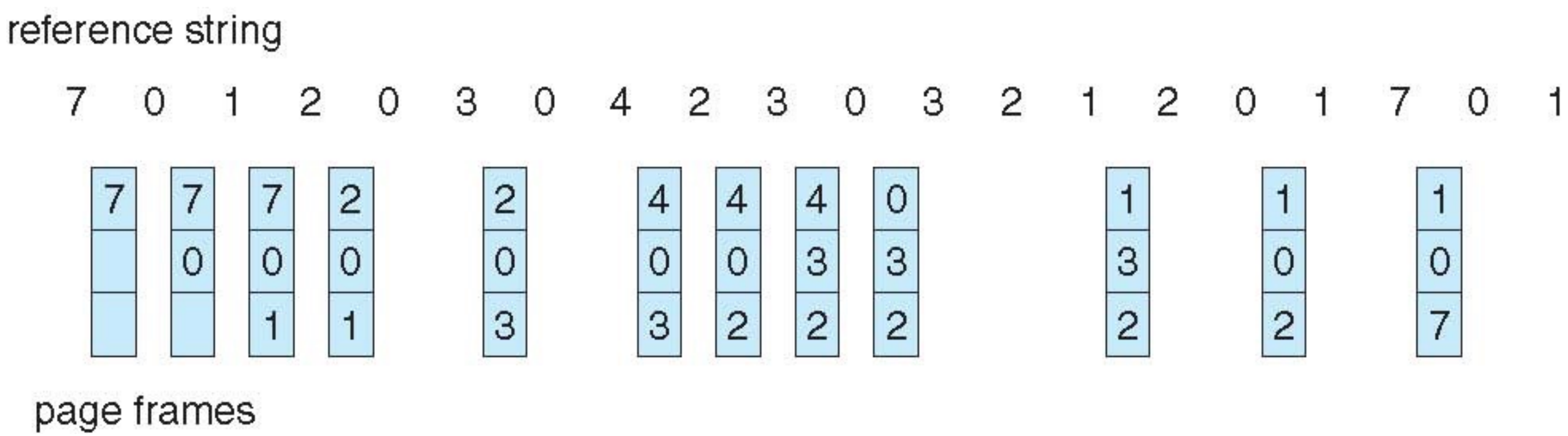
- Replace page that will **not be used for longest period of time in the future**
- 9 page faults is optimal for the example



- Not practical, as it is difficult to predict page requests in future.
- Therefore, used for measuring how well an algorithm performs.

Least Recently Used (LRU) Algorithm

- Replace page that has **not** been used for the **longest period of time**



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used