BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

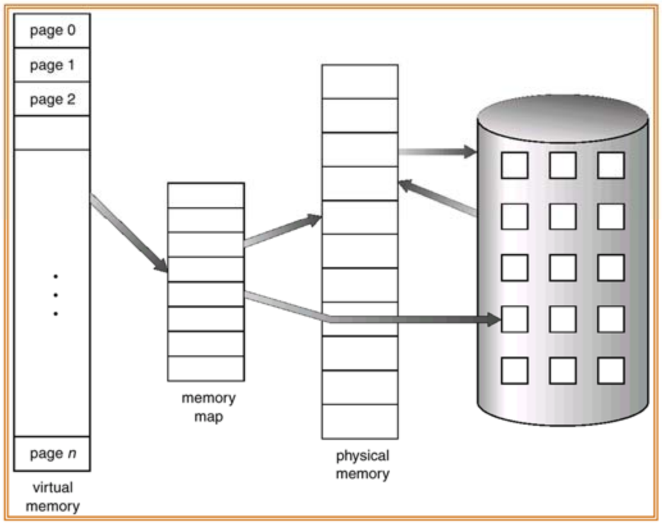
Virtual Memory

Notice: The slides for this lecture have been largely based on those accompanying an earlier edition of the course text *Operating Systems Concepts with Java*, by Silberschatz, Galvin, and Gagne. Many, if not all, of the illustrations contained in this presentation come from this source.

Virtual Memory

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory
Larger than Physical Memory

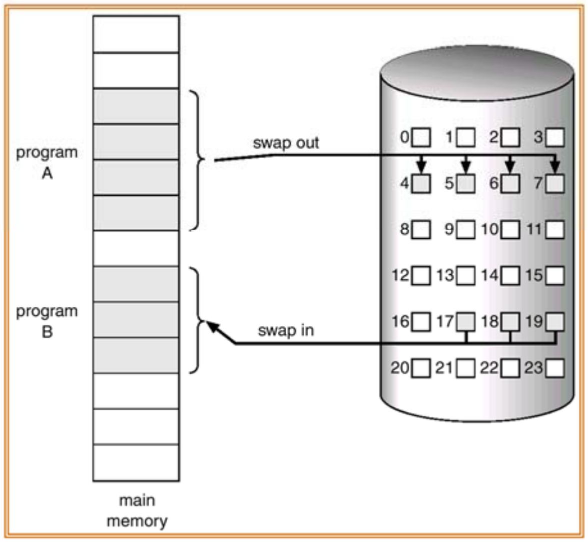


Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed.
  - Less memory needed.
  - Faster response.
  - More users.

- Page is needed (there is a reference to it):
  - invalid reference $\Rightarrow$ abort.
  - not-in-memory $\Rightarrow$ bring to memory.

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

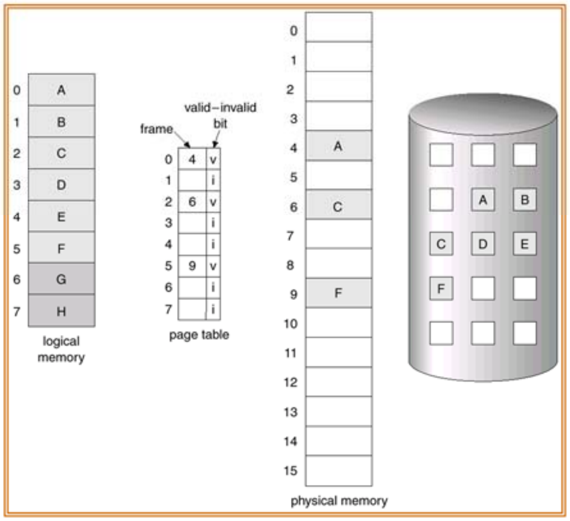
- With each page table entry a valid–invalid bit is associated
  (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot.

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

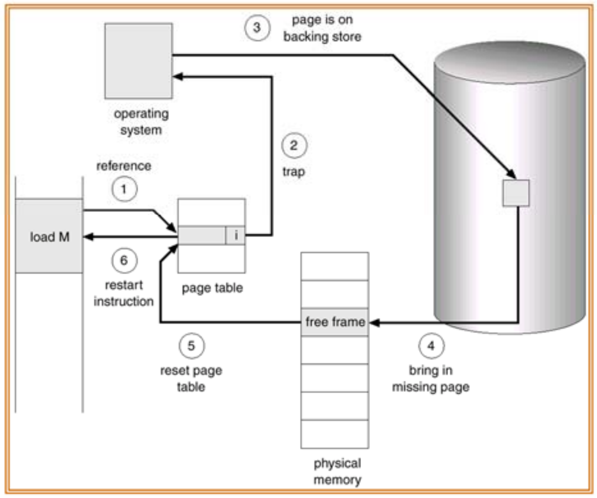
Page Table when some pages are not in Main Memory



Page Fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault.
- OS looks at page table to decide:
  - If it was an invalid reference ⇒ abort.
  - If it was a reference to a page that is not in memory, continue.
- Get an empty frame.
- Swap page into frame.
- Correct the page table and make validation bit = 1.
- Restart the instruction that caused the page fault.

Steps in Handling a Page Fault



No free frame: now what?

- **Page replacement:** Are all those pages in memory being referenced? Choose one to swap back out to disk and make room to load a new page.
  - **Algorithm:** How you choose a victim.
  - **Performance:** Want an algorithm that will result in **minimum** number of page faults.
- Side effect: The same page may be brought in and out of memory several times.

Performance of Demand Paging

- **Page Fault Rate:** $0 \le p \le 1.0$
  - if $p = 0$ no page faults.
  - if $p = 1$, every reference is a fault.
- **Effective Access Time (EAT):**

  EAT = [(1 – *p*) (memory access)] + [*p* (page fault overhead)]

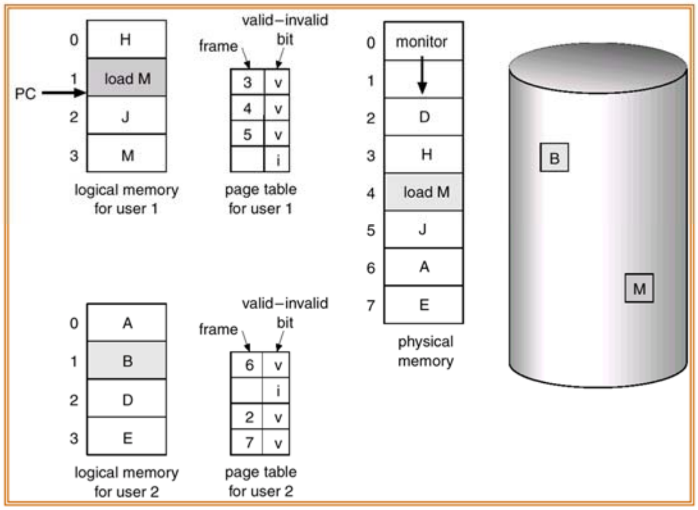
where:

page fault overhead = [swap page out] + [swap page in] + [restart overhead]

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use ***modify (dirty) bit*** to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Need For Page Replacement



Basic Page Replacement

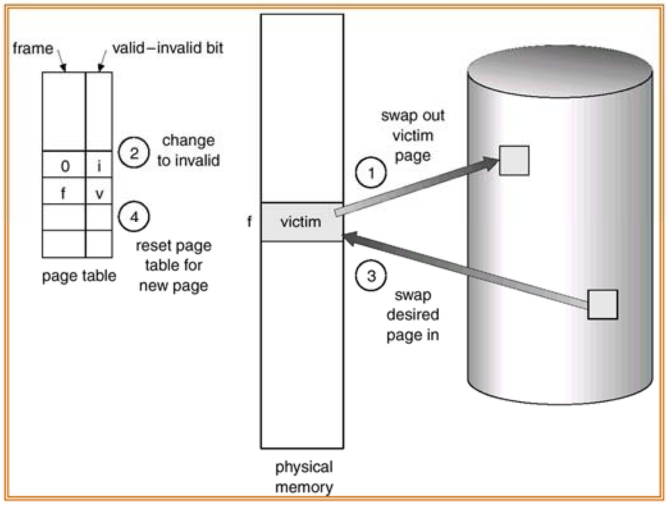
1. Find the location of the desired page on disk.

2. Find a free frame:
   - 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

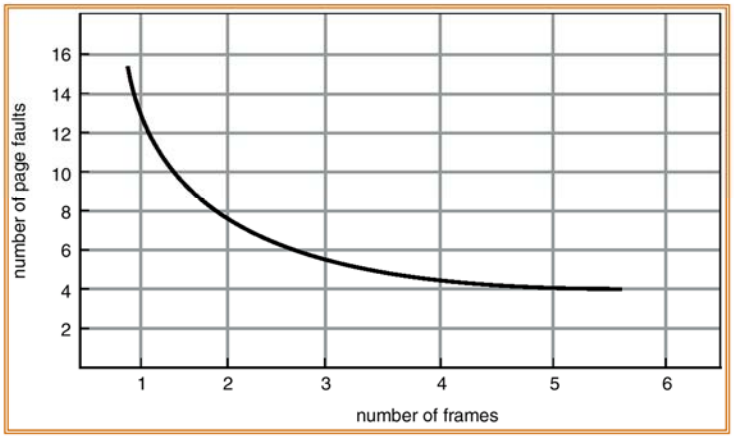


Page Replacement Algorithms

- **Goal:** Produce a low page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (***reference string***) and computing the number of page faults on that string.
- The reference string is produced by tracing a real program or by some stochastic model. We look at every address produced and strip off the page offset, leaving only the page number. For instance:

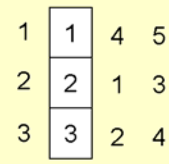
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Graph of Page Faults Versus The Number of Frames



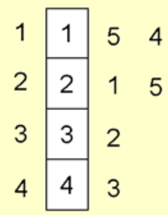
FIFO Page Replacement

- 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

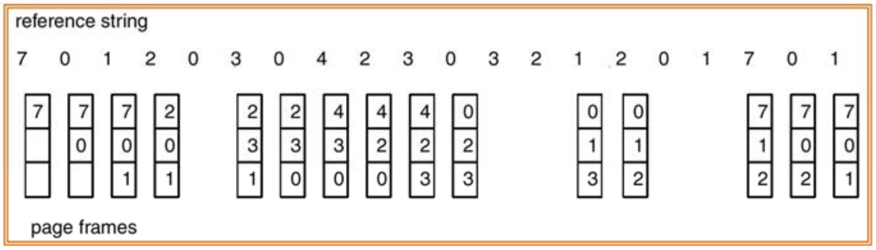
- 4 frames



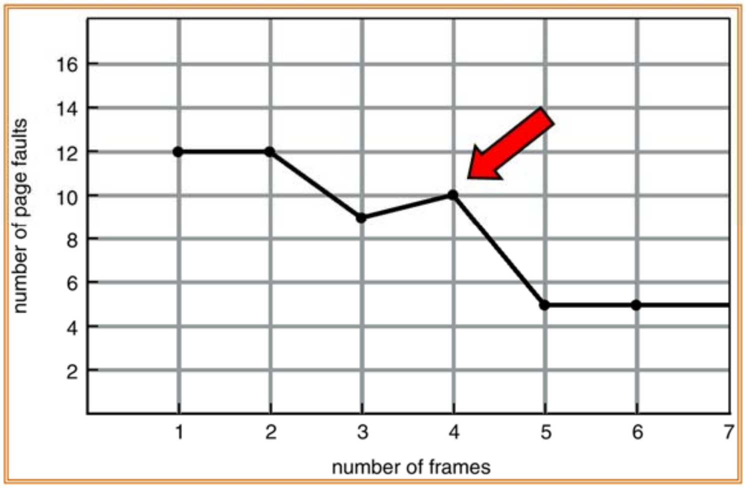
10 page faults

- FIFO Replacement ⇒ **Belady's Anomaly:** more frames, *more* page faults.

FIFO Page Replacement

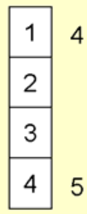


FIFO (Belady's Anomaly)



Optimal Algorithm

- Replace the page that will not be used for longest period of time. (How can you know what the future references will be?)
- 4 frames example: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**



6 page faults

- Used for measuring how well your algorithm performs.

Optimal Page Replacement