

CSE 421/521 - Operating Systems Spring 2018

LECTURE - XV

VIRTUAL MEMORY - I

Tevfik Koşar

University at Buffalo
March 29th, 2018

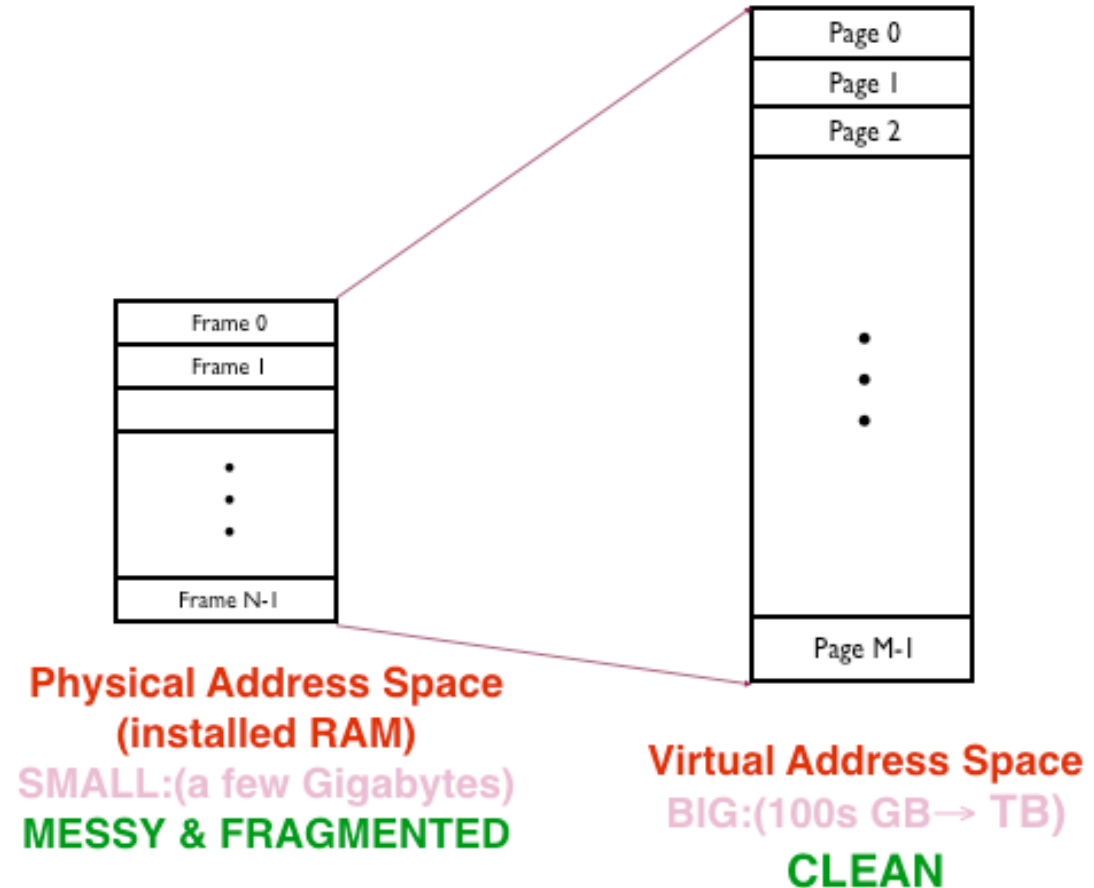
Roadmap

- Virtual Memory
 - Demand Paging
 - Page Faults
 - Page Replacement
 - Page Replacement Algorithms
 - FIFO
 - Optimal Algorithm
 - LRU & LRU Approximations
 - Counting Algorithms



Virtual Memory

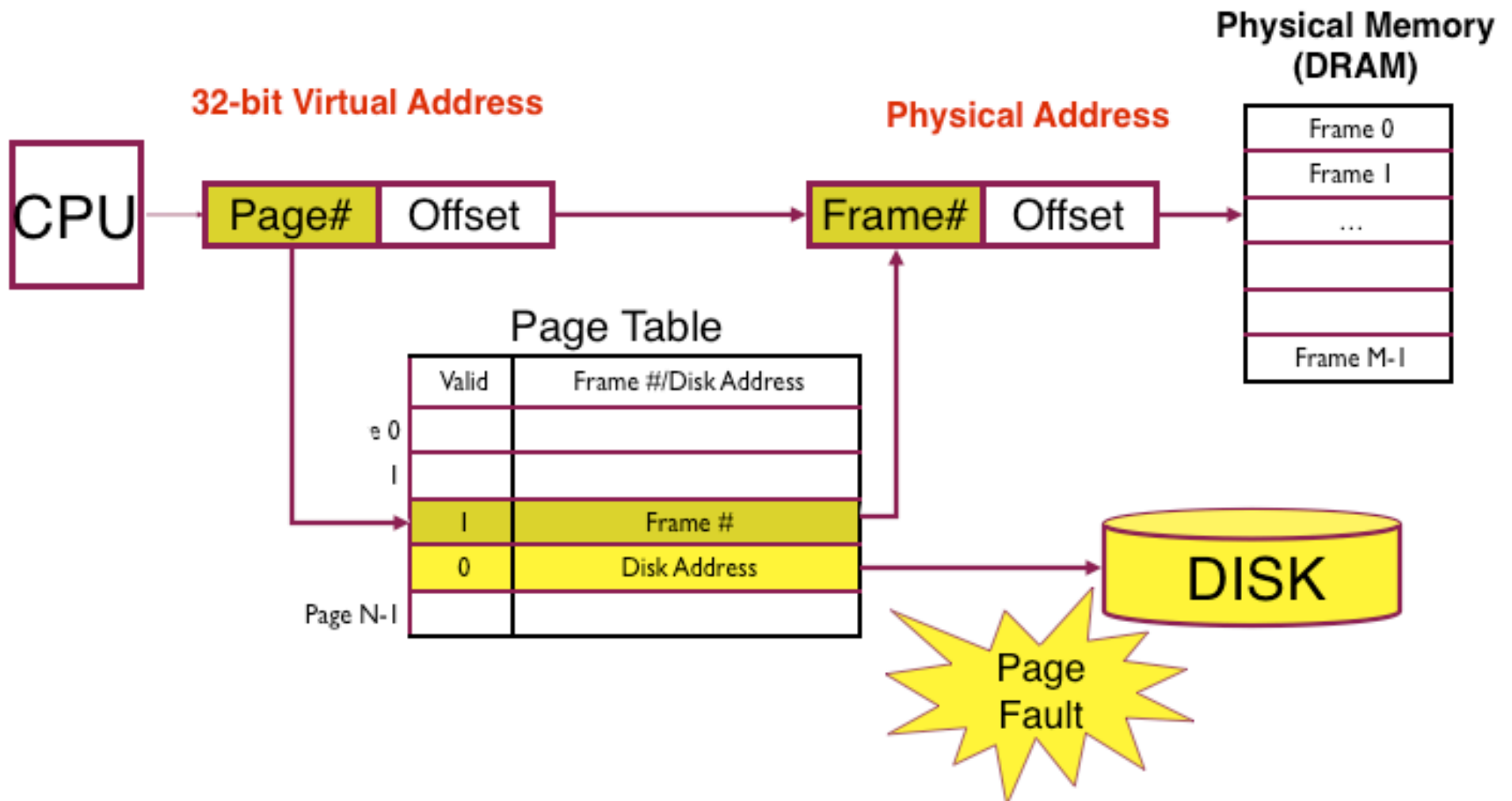
- separation of user logical memory from physical memory.
 - Only part of the program needs to be in memory for execution.
 - Logical address space can therefore be much larger than physical address space.
 - Allows address spaces to be shared by several processes.
 - Allows for more efficient process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



Goals

- Make programmers job easier
 - Can write code without knowing how much DRAM is there
 - Only need to know general memory architecture
 - (e.g., 32-bit address space)
- Enable Multiprogramming
 - Keep several programs running concurrently
 - Together, these programs may need more DRAM than we have.
 - Keep just the actively used pages in DRAM.
 - Share when possible
 - When one program does I/O switch CPU to another.

How it works?



Implementation

- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation

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 \Rightarrow reference to it
 - invalid reference \Rightarrow abort
 - not-in-memory \Rightarrow bring to memory

Valid-Invalid Bit

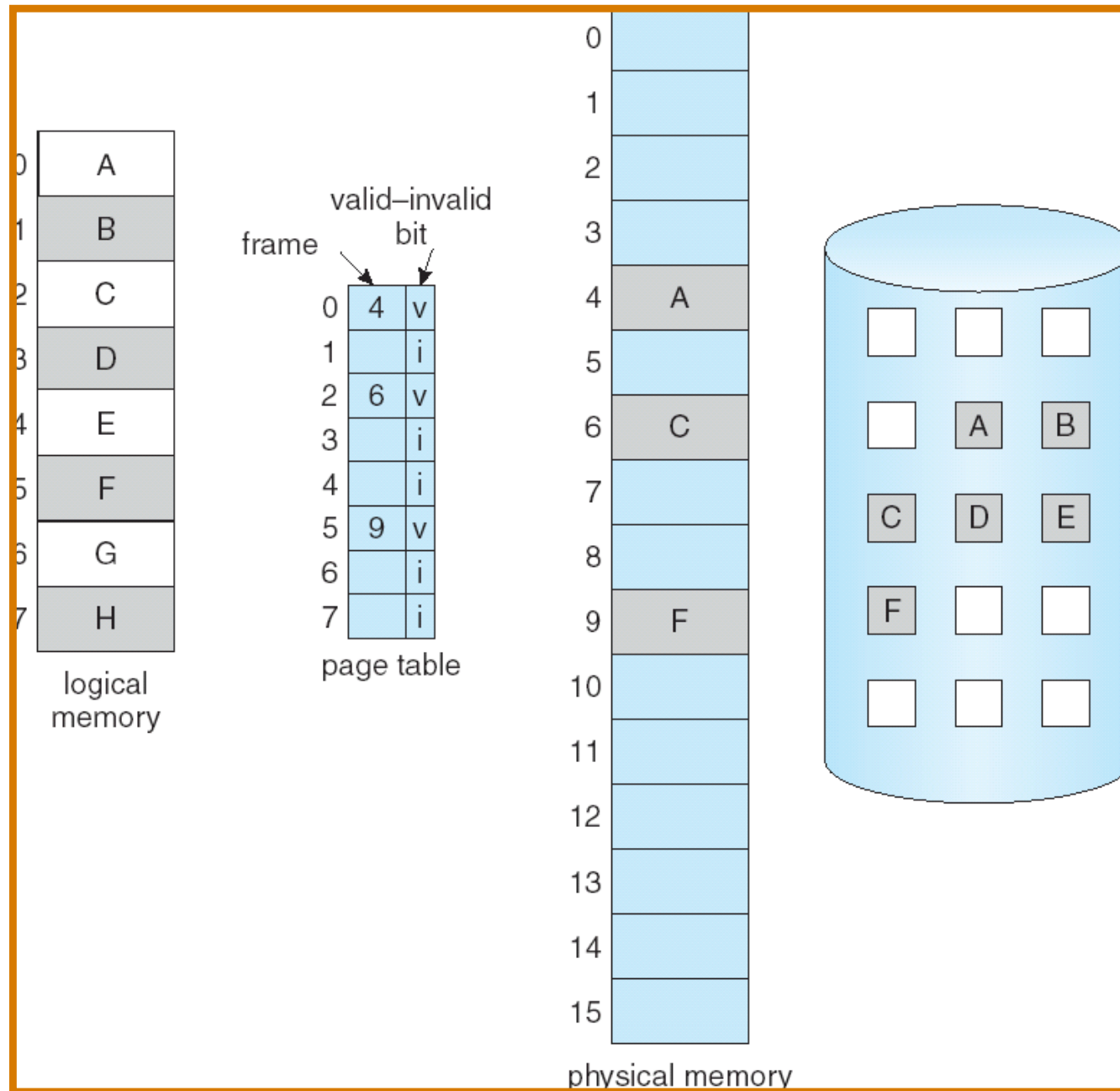
- With each page table entry a valid-invalid bit is associated (1 \Rightarrow in-memory and legal, 0 \Rightarrow not-in-memory or invalid)
- Initially valid-invalid bit is set to 0 on all entries
- Example of a page table snapshot:

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

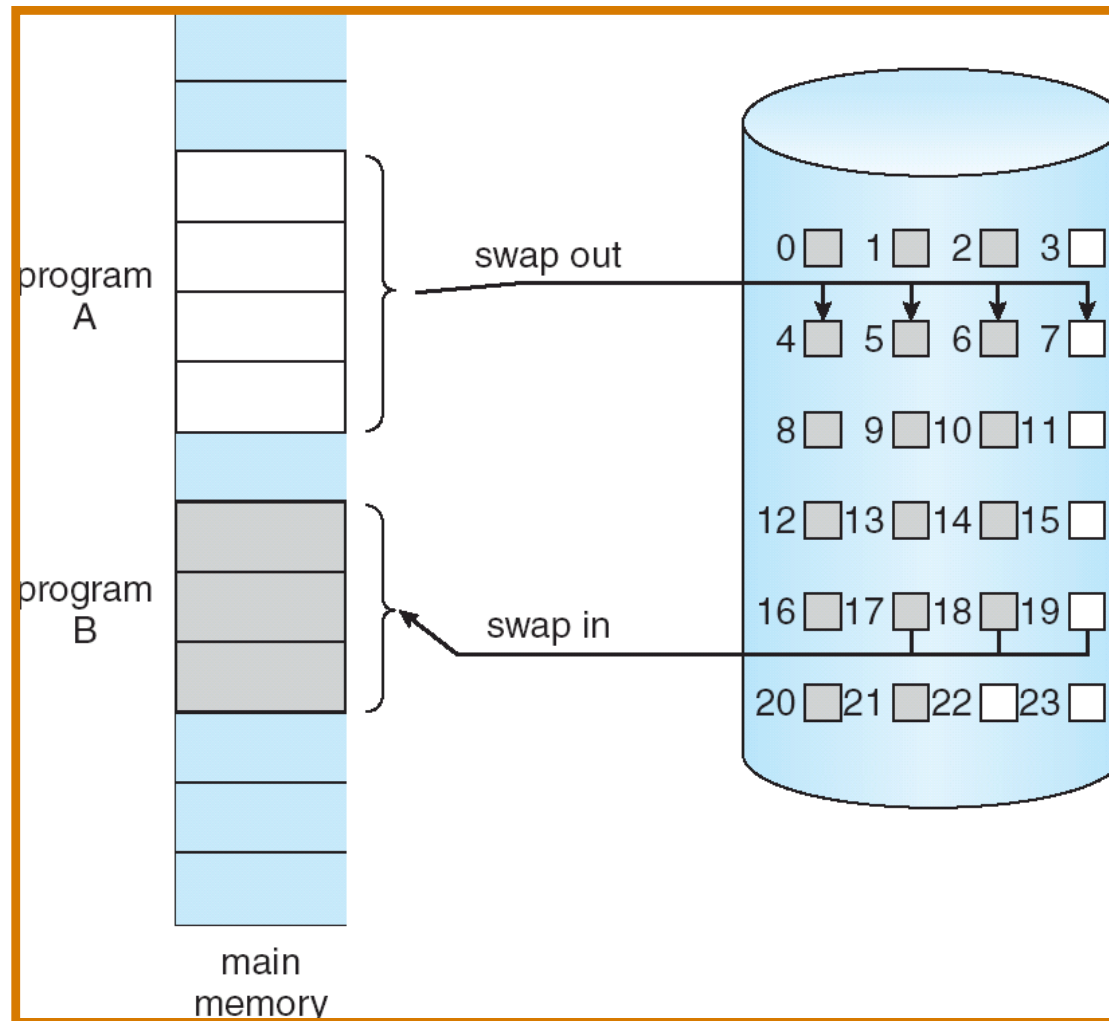
page table

- During address translation, if valid-invalid bit in page table entry is 0 \Rightarrow page fault

Page Table When Some Pages Are Not in Main Memory



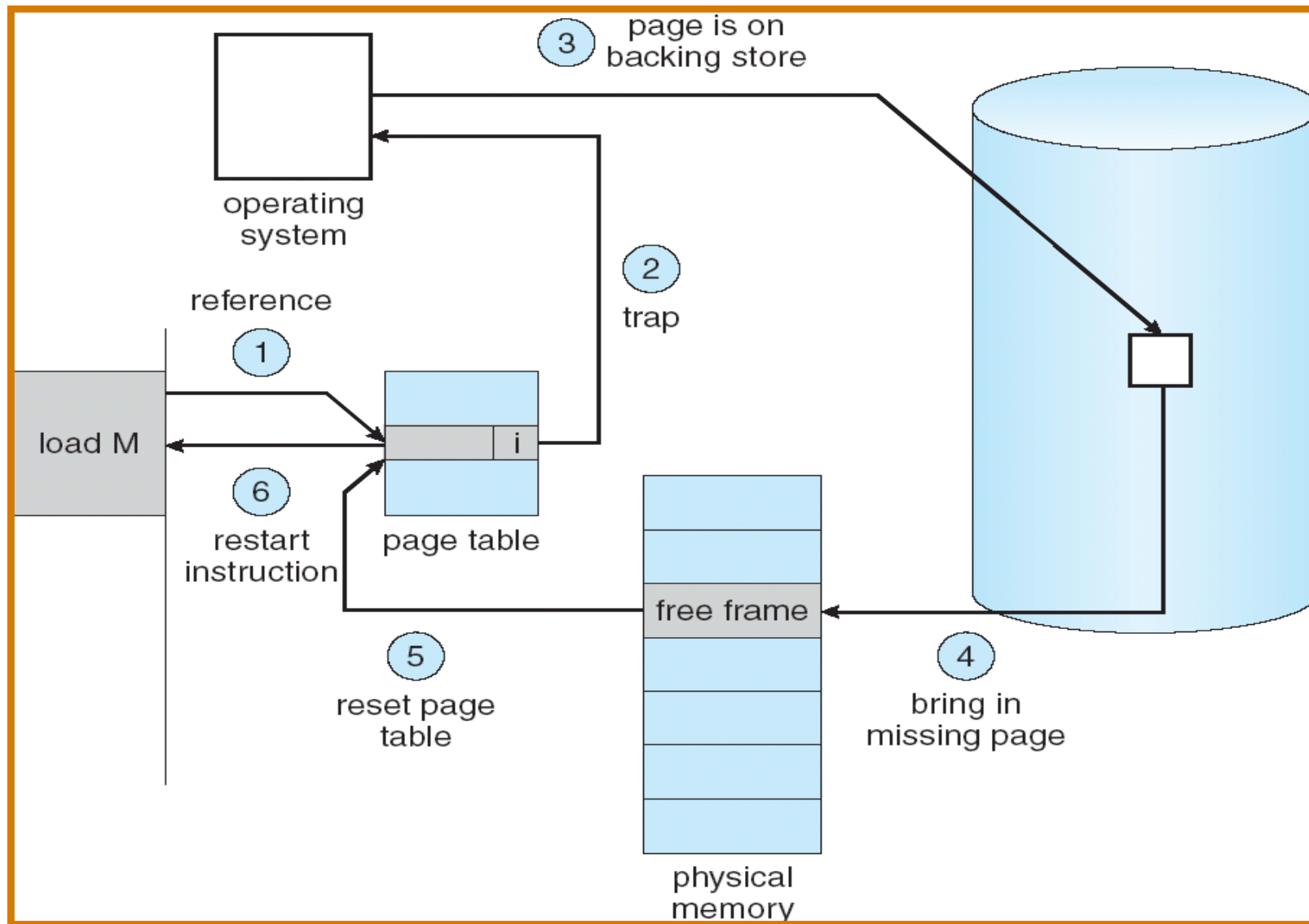
Transfer of a Paged Memory to Contiguous Disk Space

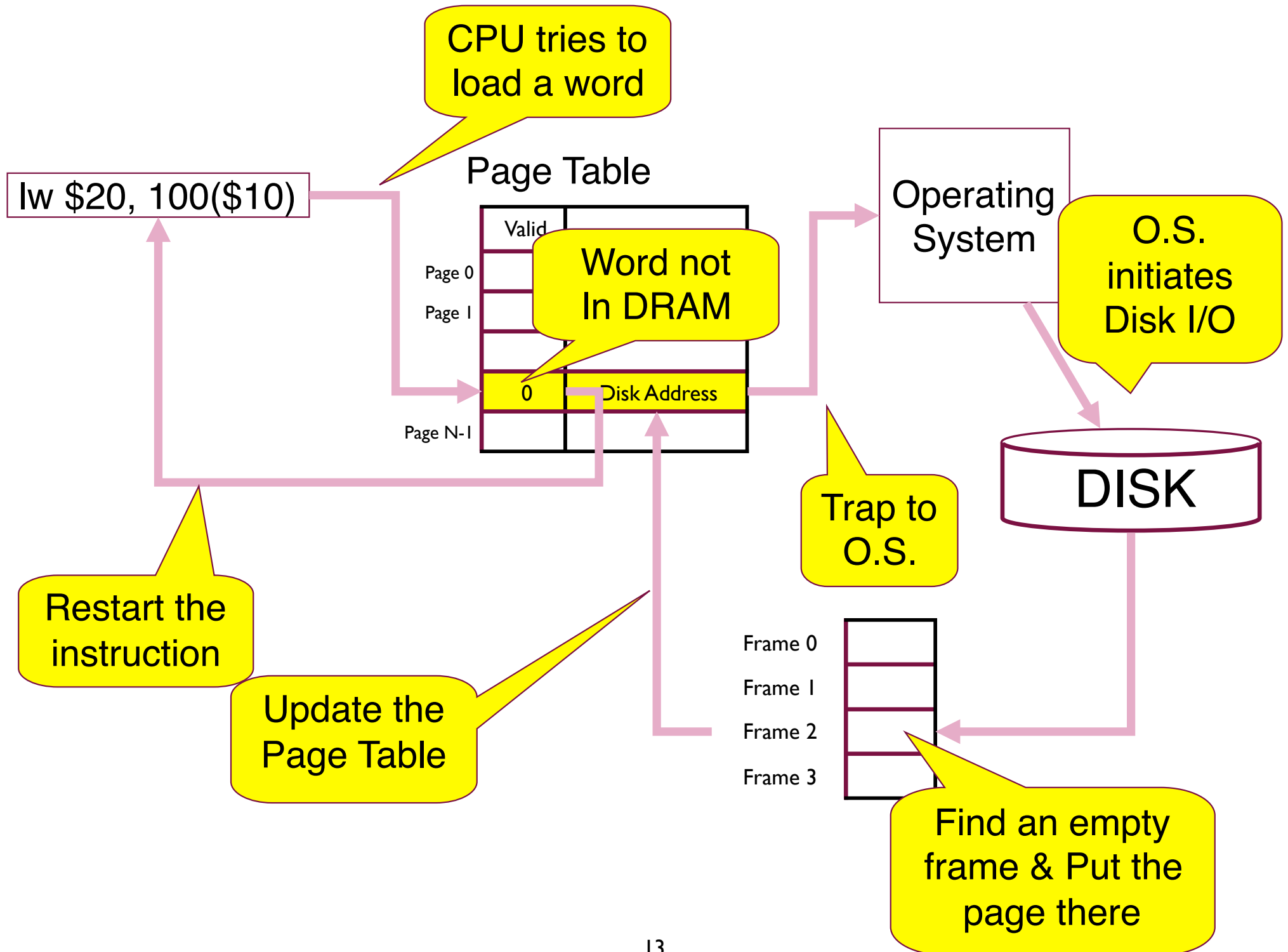


Page Fault

- If there is ever a reference to a page not in memory, first reference will trap to OS \Rightarrow page fault
- OS looks at another table (in PCB) to decide:
 - Invalid reference \Rightarrow abort.
 - Just not in memory. \Rightarrow page-in
- Get an empty frame.
- Swap (read) page into the new frame.
- Set validation bit = 1.
- Restart instruction

Steps in Handling a Page Fault





What happens if there is no free frame?

- Page replacement - find some page in memory, but not really in use, swap it out
 - Algorithms (FIFO, LRU ..)
 - performance - want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

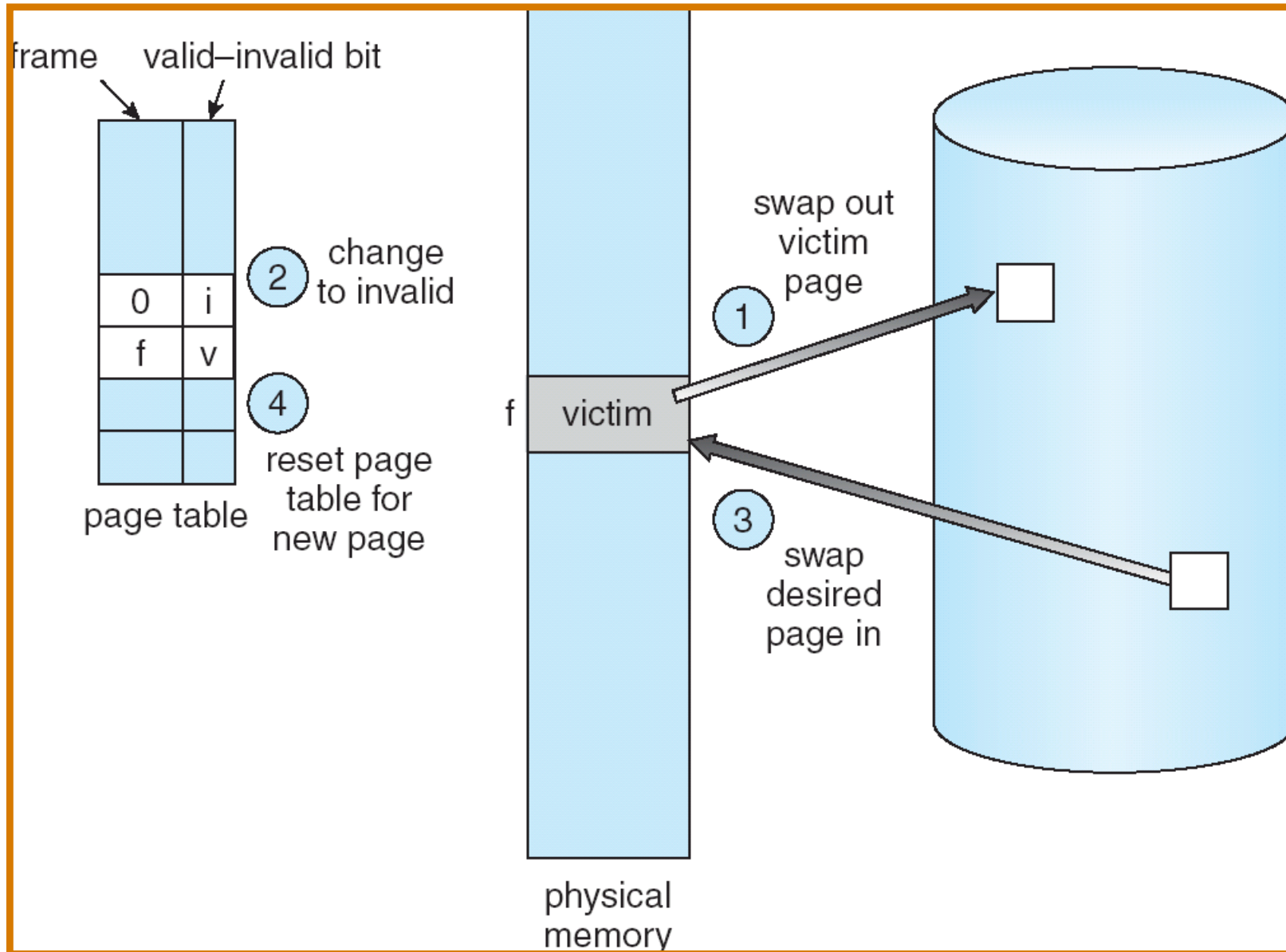
Page Replacement

- 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

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

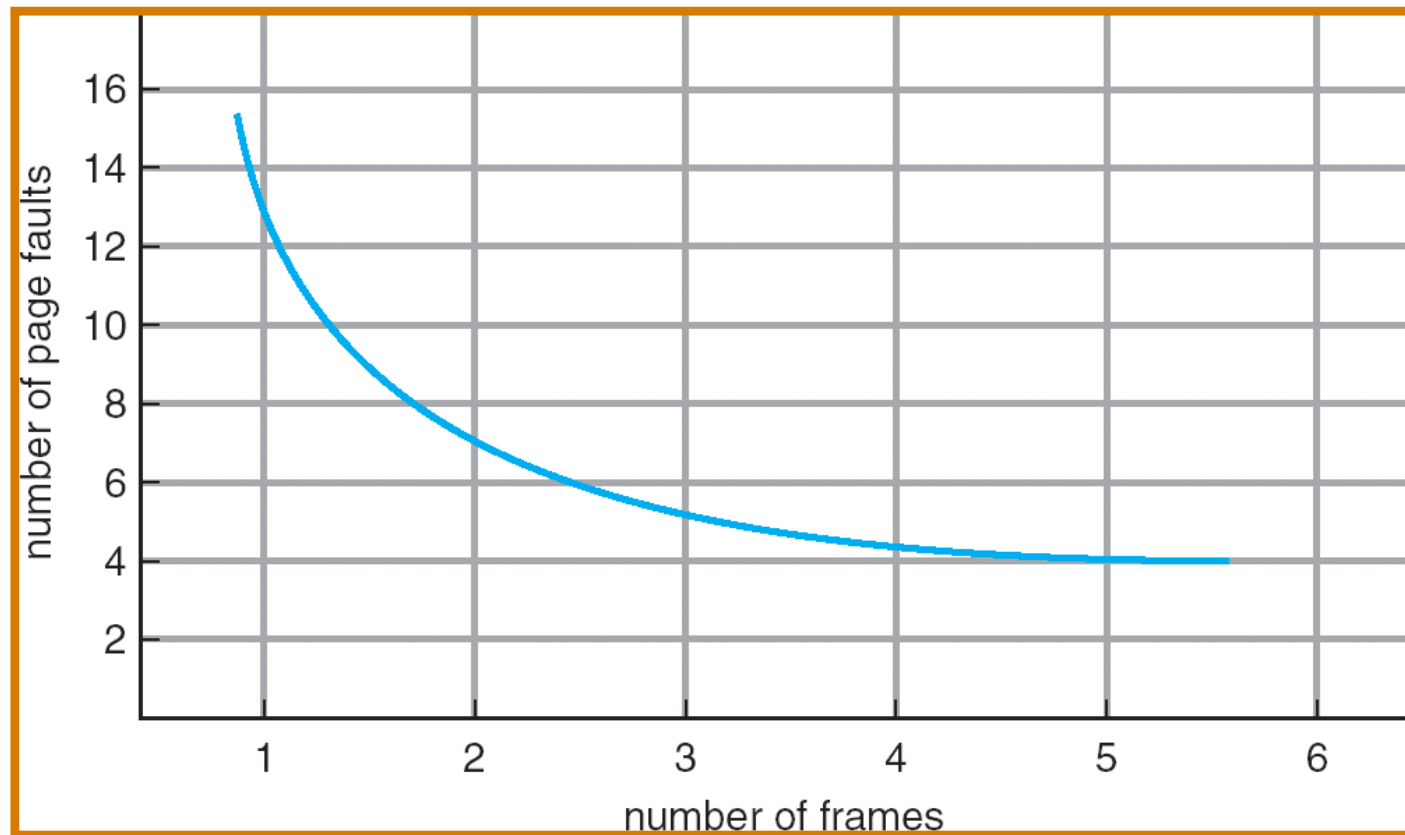
Page Replacement



Page Replacement Algorithms

- Want lowest 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
- In all our examples, the reference string is
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Expected Graph of Page Faults vs The Number of Frames



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)



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	4	5	9 page faults
2	1	3	
3	2	4	

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	4	5	
2	1	3	9 page faults
3	2	4	

- 4 frames

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)

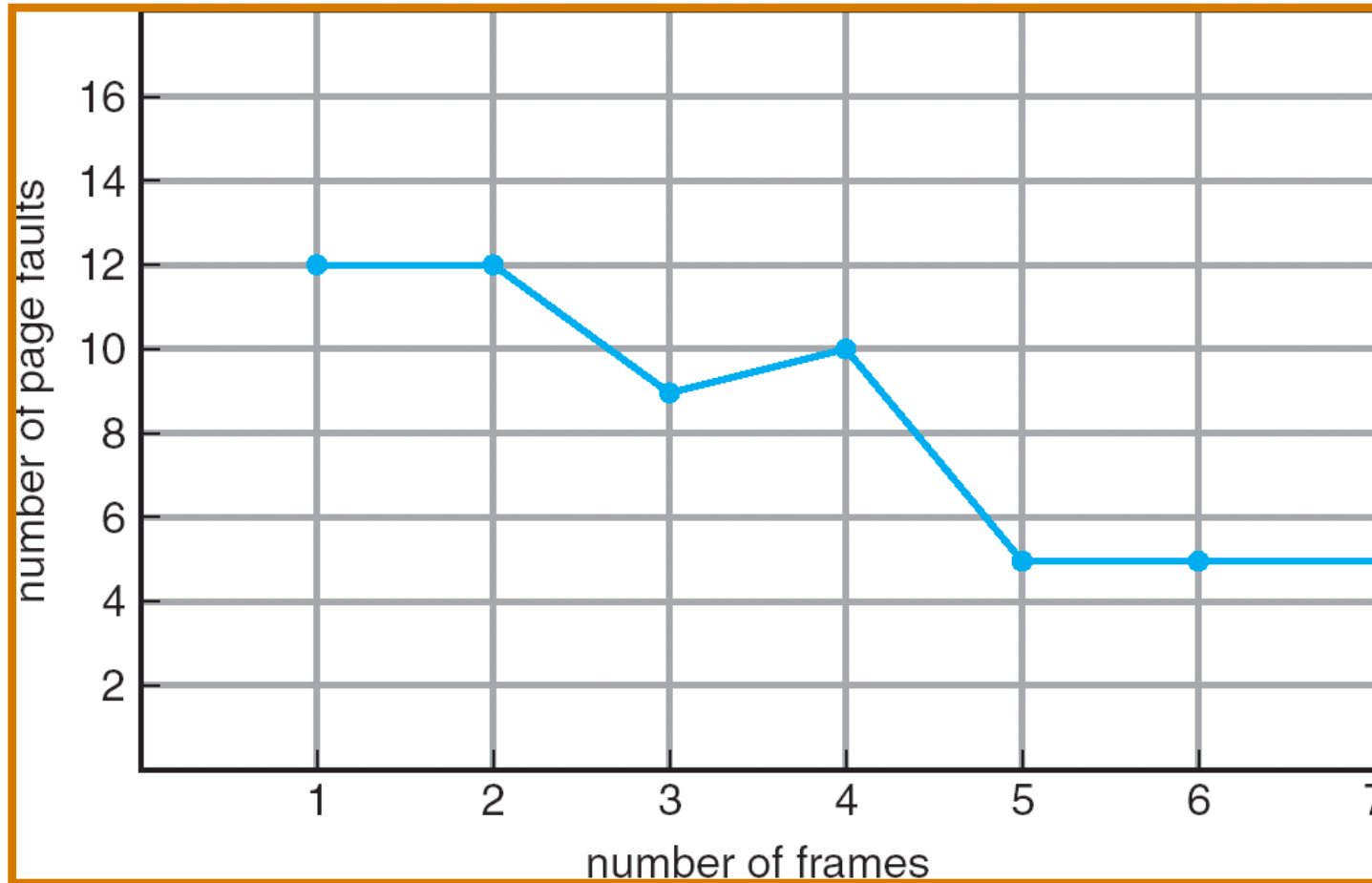
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- FIFO Replacement - **Belady's Anomaly**
 - more frames \Rightarrow more page faults

FIFO Illustrating Belady's Anomaly



FIFO

- FIFO is obvious, and simple to implement
 - when you page in something, put it on the tail of a list
 - evict page at the head of the list
- Why might this be good?
 - maybe the one brought in longest ago is not being used
- Why might this be bad?
 - then again, maybe it *is* being used
 - have absolutely no information either way
- In fact, FIFO's performance is typically lousy
- In addition, FIFO suffers from **Belady's Anomaly**
 - there are **reference strings** for which the fault rate *increases* when the process is given more physical memory

Optimal Algorithm

- Replace page that **will not be used for the longest time in future**
- 4 frames example

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



Optimal Algorithm

- Replace page that **will not be used for longest period of time**
- 4 frames example

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

1	4
2	
3	
4	5

6 page faults

- How would you know this in advance?

Optimal (Belady's) Algorithm

- **Provably optimal**: lowest fault rate (remember SJF?)
 - evict the page that won't be used for the longest time in future
 - **problem: impossible to predict the future**
- Why is Belady's Optimal algorithm useful?
 - as a yardstick to compare other algorithms to optimal
 - if Belady's isn't much better than yours, yours is pretty good
 - how could you do this comparison?
- Is there a best practical algorithm?
 - no; depends on workload
- Is there a worst algorithm?
 - no, but random replacement does pretty badly
 - there are some other situations where OS's use near-random algorithms quite effectively!

Least Recently Used (LRU)

- Replace the page that has not been used for the longest amount of time in the past
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Least Recently Used (LRU)

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 4 frames example:

1	5	
2		
3	5	4
4	3	

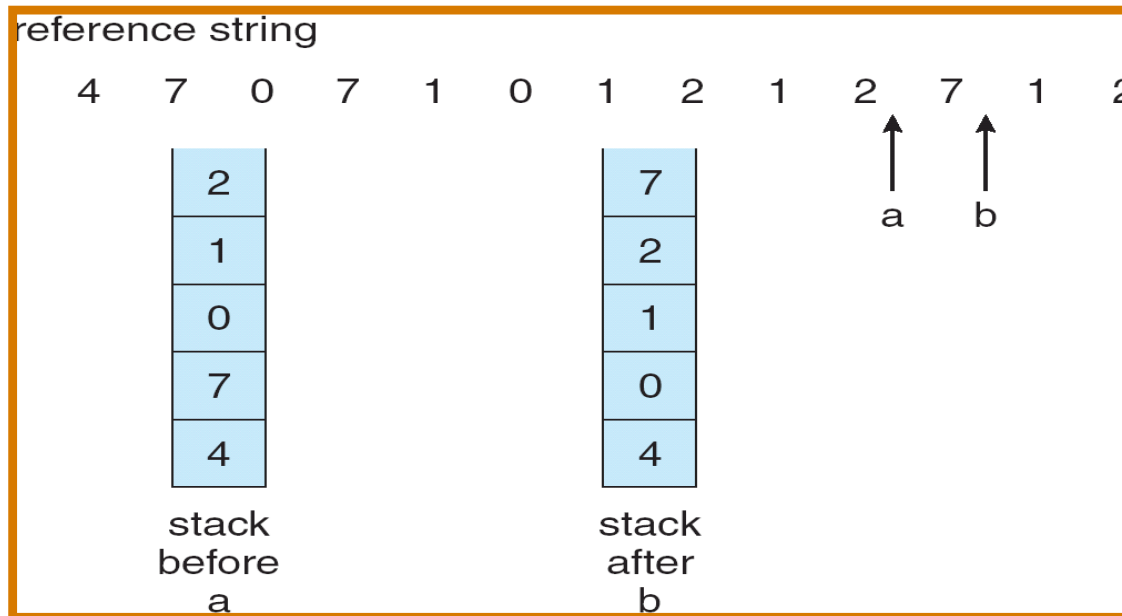
8 page faults

Least Recently Used (LRU)

- LRU uses reference information to make a more informed replacement decision
 - idea: past experience gives us a guess of future behavior
 - on replacement, evict the page that hasn't been used for the longest amount of time
 - LRU looks at the past, Belady's wants to look at future
 - *How is LRU different from FIFO?*
- Implementation
 - to be perfect, must grab a timestamp on every memory reference, then order or search based on the timestamps ...
 - way **too costly** in memory bandwidth, algorithm execution time, etc.
 - so, we need a cheap approximation ...

LRU Implementations

- **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
 - No search for replacement



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.

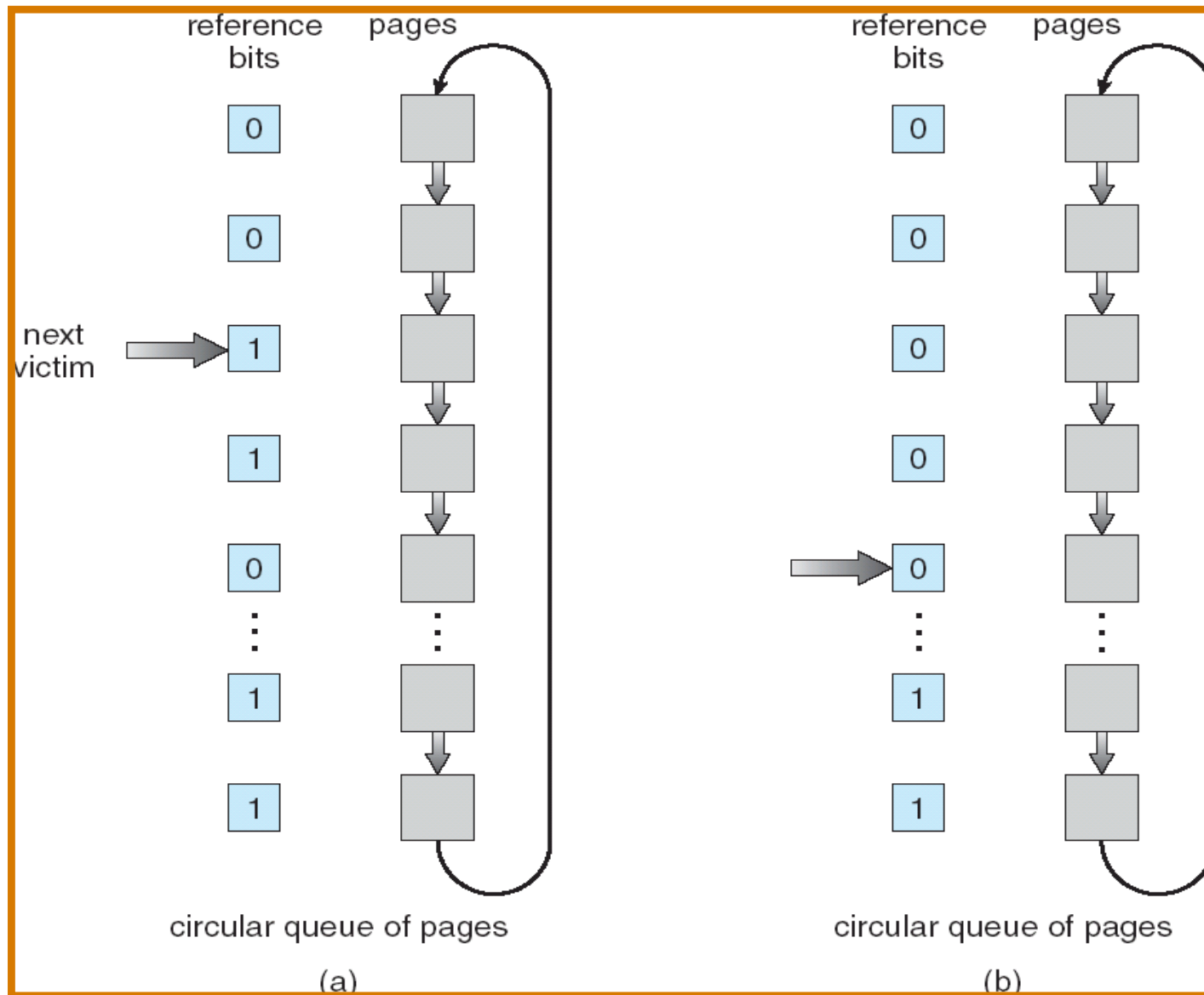
- Additional Reference bits

- 1 byte for each page: eg. 00110011
- Shift right at each time interval

LRU Clock Algorithm

- AKA Not Recently Used (NRU) or **Second Chance**
 - replace page that is “old enough”
 - logically, arrange all physical page frames in a big circle (clock)
 - just a circular linked list
 - a “clock hand” is used to select a good LRU candidate
 - sweep through the pages in circular order like a clock
 - if ref bit is off, it hasn’t been used recently, we have a victim
 - so, what is minimum “age” if ref bit is off?
 - if the ref bit is on, turn it off and go to next page
 - arm moves quickly when pages are needed
 - low overhead if have plenty of memory
 - if memory is large, “accuracy” of information degrades
 - add more hands to fix

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

Exercise

- Consider the following page-reference string:

1, 2, 3, 4, 4, 3, 2, 1, 5, 6, 2, 1, 2, 3, 7, 8, 3, 2, 1, 5

Assuming 4 memory frames and LRU, LFU, or Optimal page replacement algorithms, how many page faults, page hits, and page replacements would occur? Show your page assignments to frames.

1	2	3	4	4	3	2	1	5	6	2	1	2	3	7	8	3	2	1	5

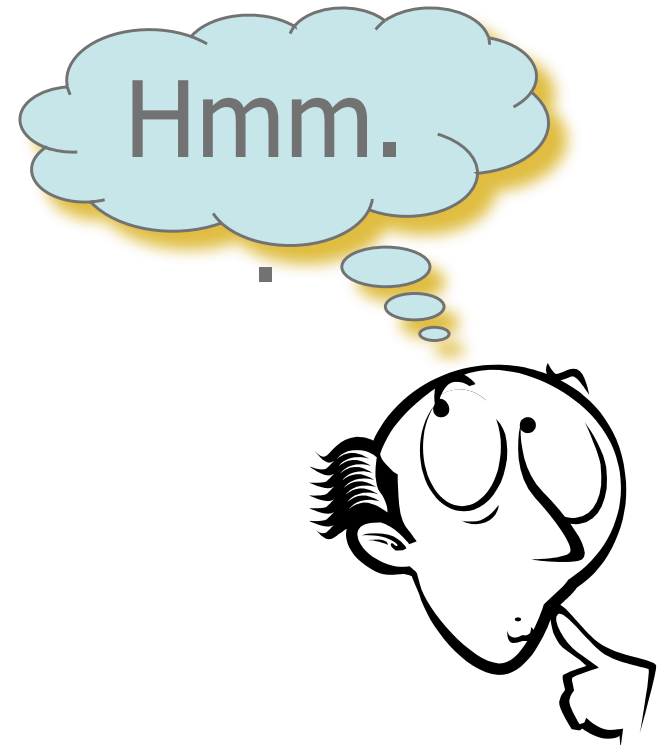
of page faults:

of page hits:

of page replacements:

Summary

- Virtual Memory
 - Demand Paging
 - Page Faults
 - Page Replacement
 - Page Replacement Algorithms
 - FIFO
 - Optimal Algorithm
 - LRU & LRU Approximations
 - Counting Algorithms



- Next Lecture: Virtual Memory - II
- Reading Assignment: Chapter 9 from Silberschatz.

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR