

CYB 510 OPERATING SYSTEMS

Prepared by Prof. Dr. Lamiaa Fattouh

Lecture 9



CYB 510

Virtual Memory



جامعة زايد
EVERYONE'S SMART UNIVERSITY

Chapter 10: Virtual Memory

- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Operating-System Examples



Objectives

- Define virtual memory and describe its benefits.
- Illustrate how pages are loaded into memory using demand paging.
- Apply the FIFO, optimal, and LRU page-replacement algorithms.
- Describe the working set of a process and explain how it is related to program locality.
- Describe how Windows 10 manage virtual memory.



Background

- Code needs to be in memory to execute, but entire program rarely used
 - Error code, unusual routines, large data structures
- Entire program code not needed at same time
- Consider ability to execute partially-loaded program
 - Program no longer constrained by limits of physical memory
 - Each program takes less memory while running -> more programs run at the same time
 - ▶ Increased CPU utilization and throughput with no increase in response time or turnaround time
 - Less I/O needed to load or swap programs into memory -> each user program runs faster



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
- More programs running concurrently
- Less I/O needed to load or swap processes



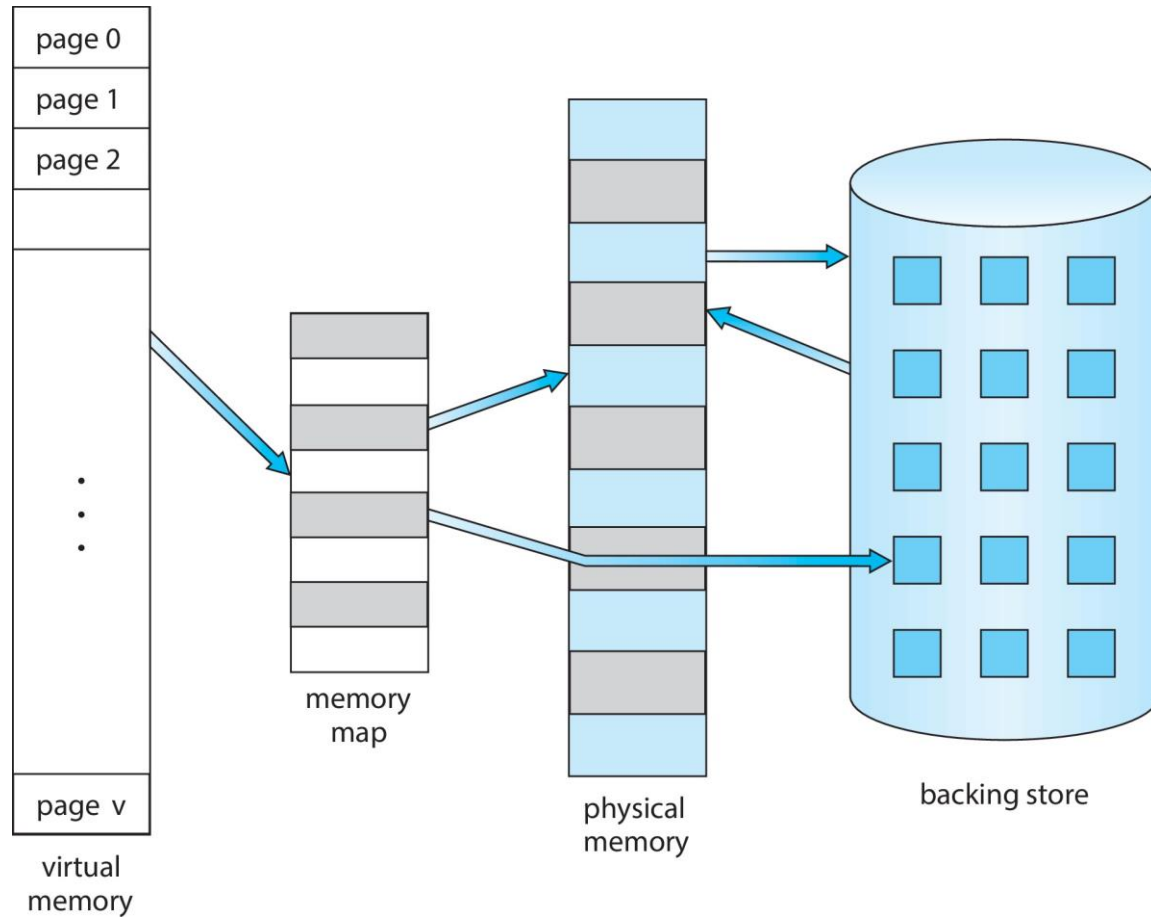
Virtual memory (Cont.)

- **Virtual address space** – logical view of how process is stored in memory
 - Usually start at address 0, contiguous addresses until end of space
 - Meanwhile, physical memory organized in page frames
 - MMU must map logical to physical
- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation



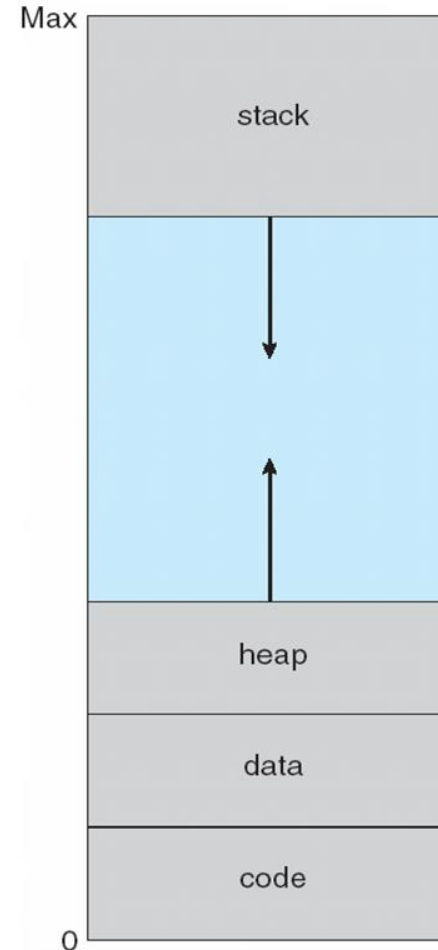
Lecture 9

Virtual Memory That is Larger Than Physical Memory

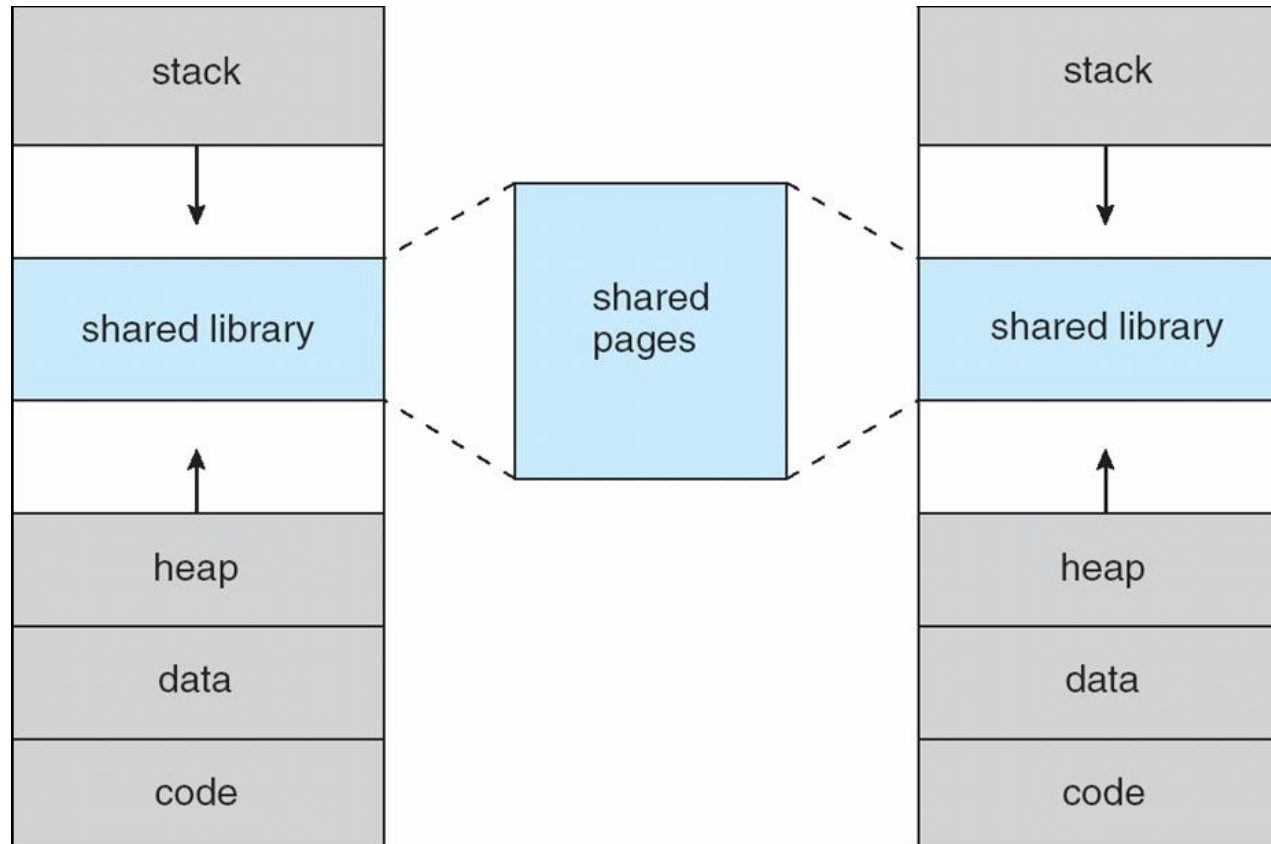


Virtual-address Space

- Usually design logical address space for stack to start at Max logical address and grow “down” while heap grows “up”
 - Maximizes address space use
 - Unused address space between the two is hole
 - ▶ No physical memory needed until heap or stack grows to a given new page
- Enables **sparse** address spaces with holes left for growth, dynamically linked libraries, etc.
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during `fork()`, speeding process creation



Shared Library Using Virtual Memory



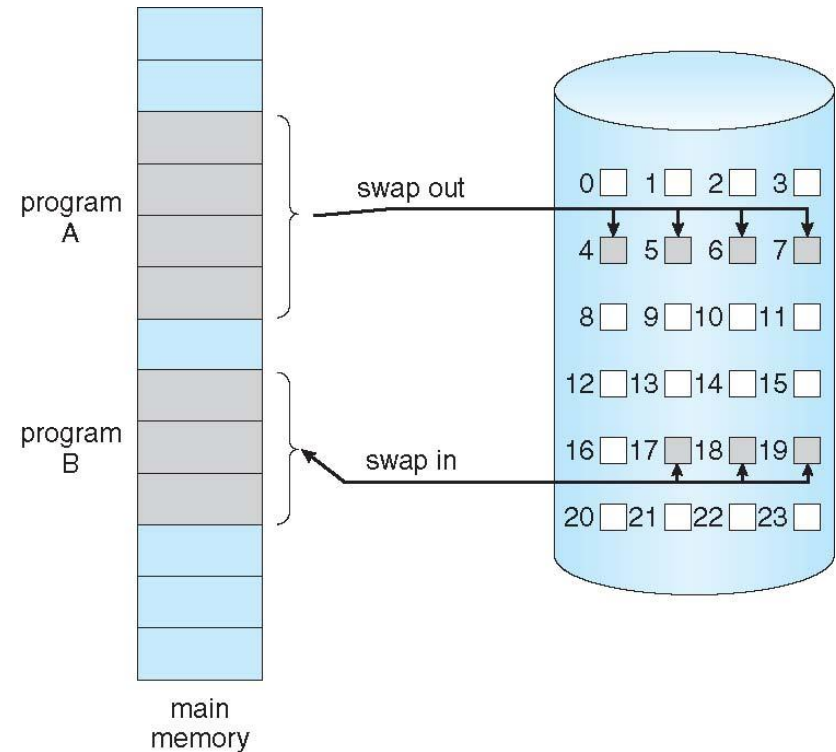
Demand Paging

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
 - Less I/O needed, no unnecessary I/O
 - Less memory needed
 - Faster response
 - More users
- Similar to paging system with swapping (diagram on right)
- Page is needed $\Rightarrow$ reference to it
 - invalid reference $\Rightarrow$ abort
 - not-in-memory $\Rightarrow$ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
 - Swapper that deals with pages is a **pager**



Demand Paging

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
 - Less I/O needed, no unnecessary I/O
 - Less memory needed
 - Faster response
 - More users
- Similar to paging system with swapping (diagram on right)



Basic Concepts

- With swapping, pager guesses which pages will be used before swapping out again
- Instead, pager brings in only those pages into memory
- How to determine that set of pages?
 - Need new MMU functionality to implement demand paging
- If pages needed are already **memory resident**
 - No difference from non demand-paging
- If page needed and not memory resident
 - Need to detect and load the page into memory from storage
 - ▶ Without changing program behavior
 - ▶ Without programmer needing to change code



Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** $\Rightarrow$ in-memory – **memory resident**, **i** $\Rightarrow$ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:

Frame #	valid-invalid bit
	v
	v
	v
	i
...	
	i
	i

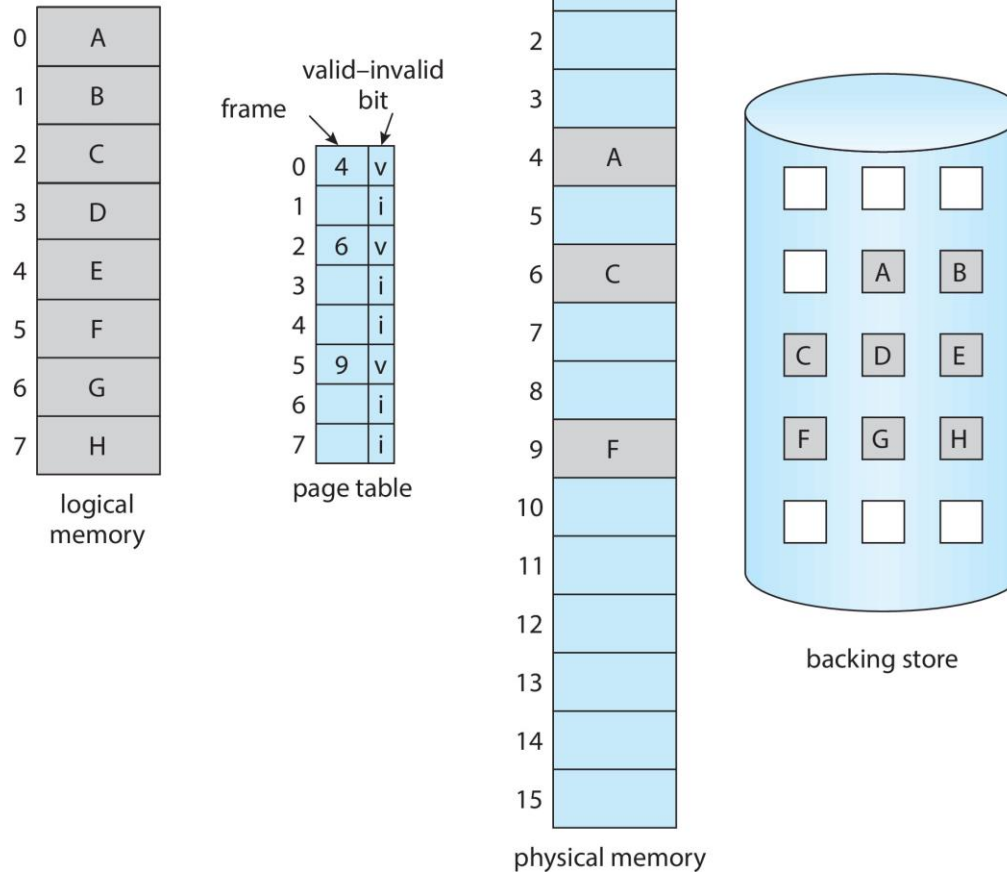
page table

- During MMU address translation, if valid–invalid bit in page table entry is **i** $\Rightarrow$ page fault



Lecture 9

Page Table When Some Pages Are Not in Main Memory



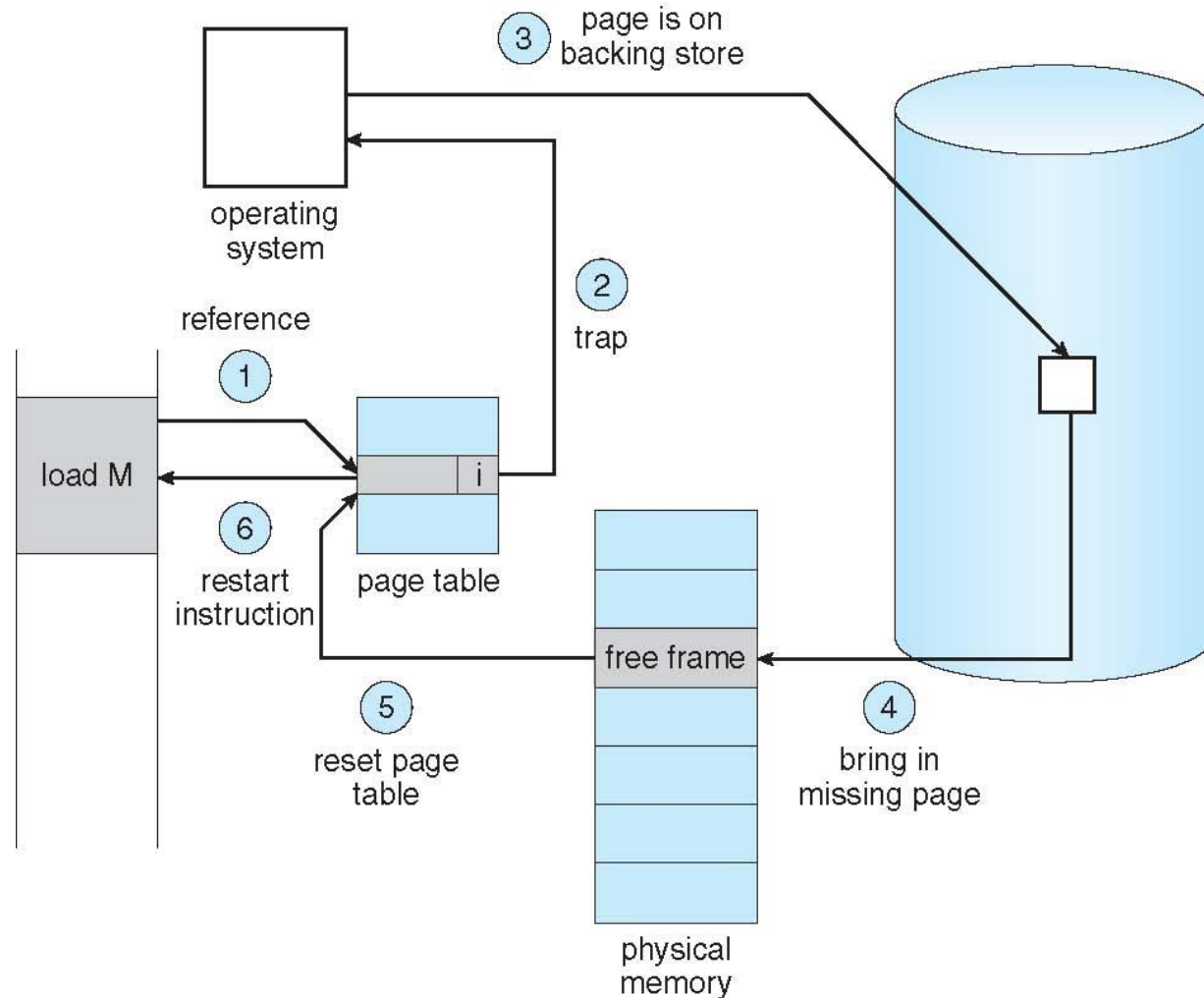
Steps in Handling Page Fault

1. If there is a reference to a page, first reference to that page will trap to operating system
 - Page fault
2. Operating system looks at another table to decide:
 - Invalid reference $\Rightarrow$ abort
 - Just not in memory
3. Find free frame
4. Swap page into frame via scheduled disk operation
5. Reset tables to indicate page now in memory
Set validation bit = **v**
6. Restart the instruction that caused the page fault



Lecture 9

Steps in Handling a Page Fault (Cont.)



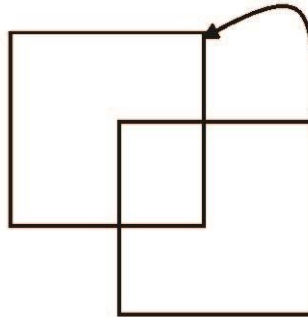
Aspects of Demand Paging

- Extreme case – start process with *no* pages in memory
 - OS sets instruction pointer to first instruction of process, non-memory-resident -> page fault
 - And for every other process pages on first access
 - **Pure demand paging**
- Actually, a given instruction could access multiple pages -> multiple page faults
 - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
 - Pain decreased because of **locality of reference**
- Hardware support needed for demand paging
 - Page table with valid / invalid bit
 - Secondary memory (swap device with **swap space**)
 - Instruction restart



Instruction Restart

- Consider an instruction that could access several different locations
 - Block move

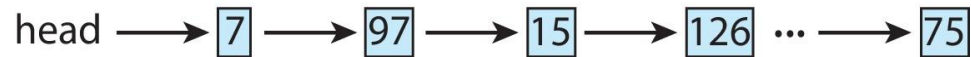


- Auto increment/decrement location
- Restart the whole operation?
 - ▶ What if source and destination overlap?



Free-Frame List

- When a page fault occurs, the operating system must bring the desired page from secondary storage into main memory.
- Most operating systems maintain a **free-frame list** -- a pool of free frames for satisfying such requests.



- Operating system typically allocate free frames using a technique known as **zero-fill-on-demand** -- the content of the frames zeroed-out before being allocated.
- When a system starts up, all available memory is placed on the free-frame list.



Stages in Demand Paging – Worse Case

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
 - a) Wait in a queue for this device until the read request is serviced
 - b) Wait for the device seek and/or latency time
 - c) Begin the transfer of the page to a free frame



Stages in Demand Paging (Cont.)

6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction



Copy-on-Write

- **Copy-on-Write** (COW) allows both parent and child processes to initially **share** the same pages in memory
 - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- In general, free pages are allocated from a **pool** of **zero-fill-on-demand** pages
 - Pool should always have free frames for fast demand page execution



What Happens if There is no Free Frame?

- Page replacement – find some page in memory, but not really in use, page it out
 - Algorithm – terminate? swap out? replace the page?
 - Performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

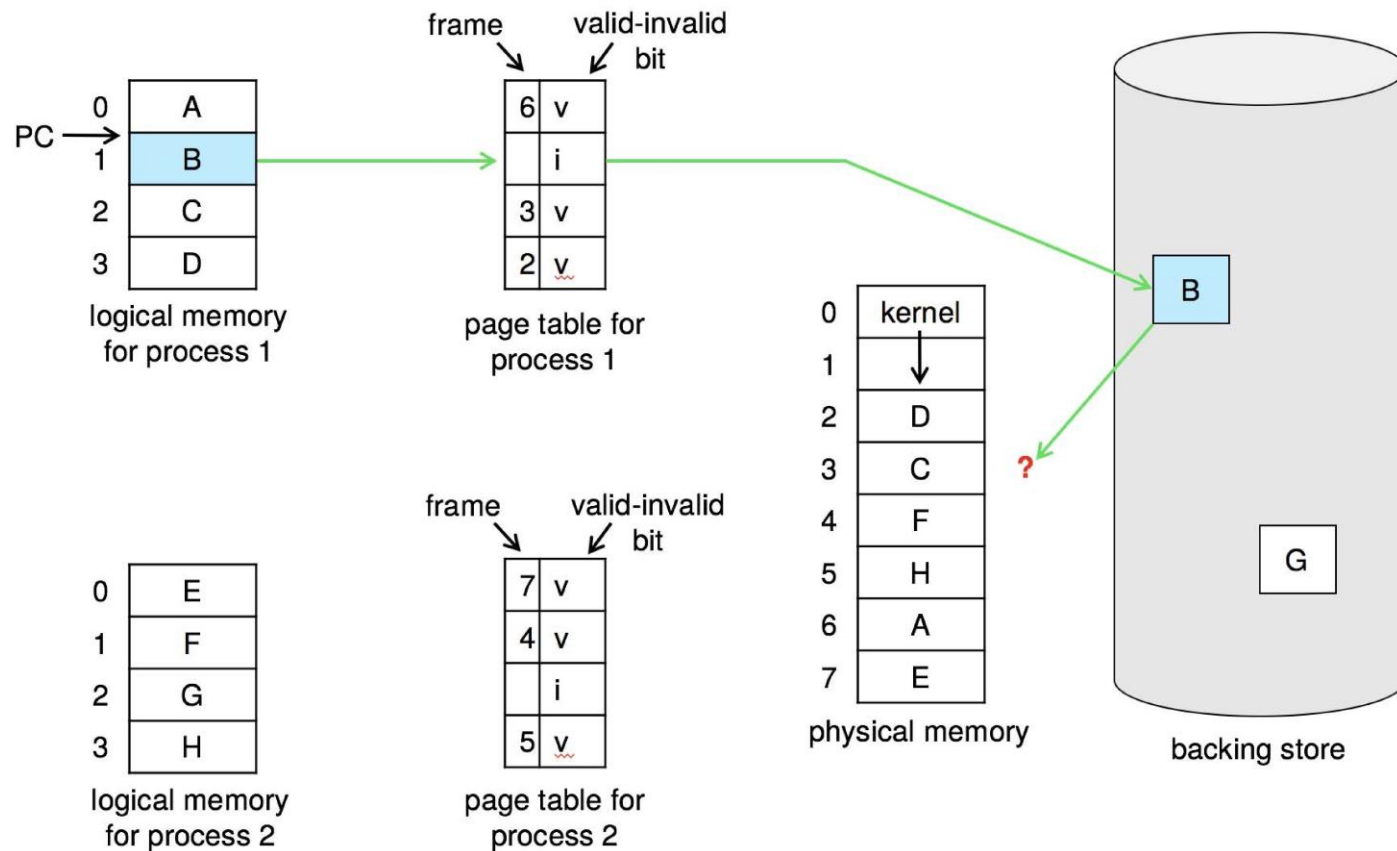


Page Replacement

- 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



Need For Page Replacement



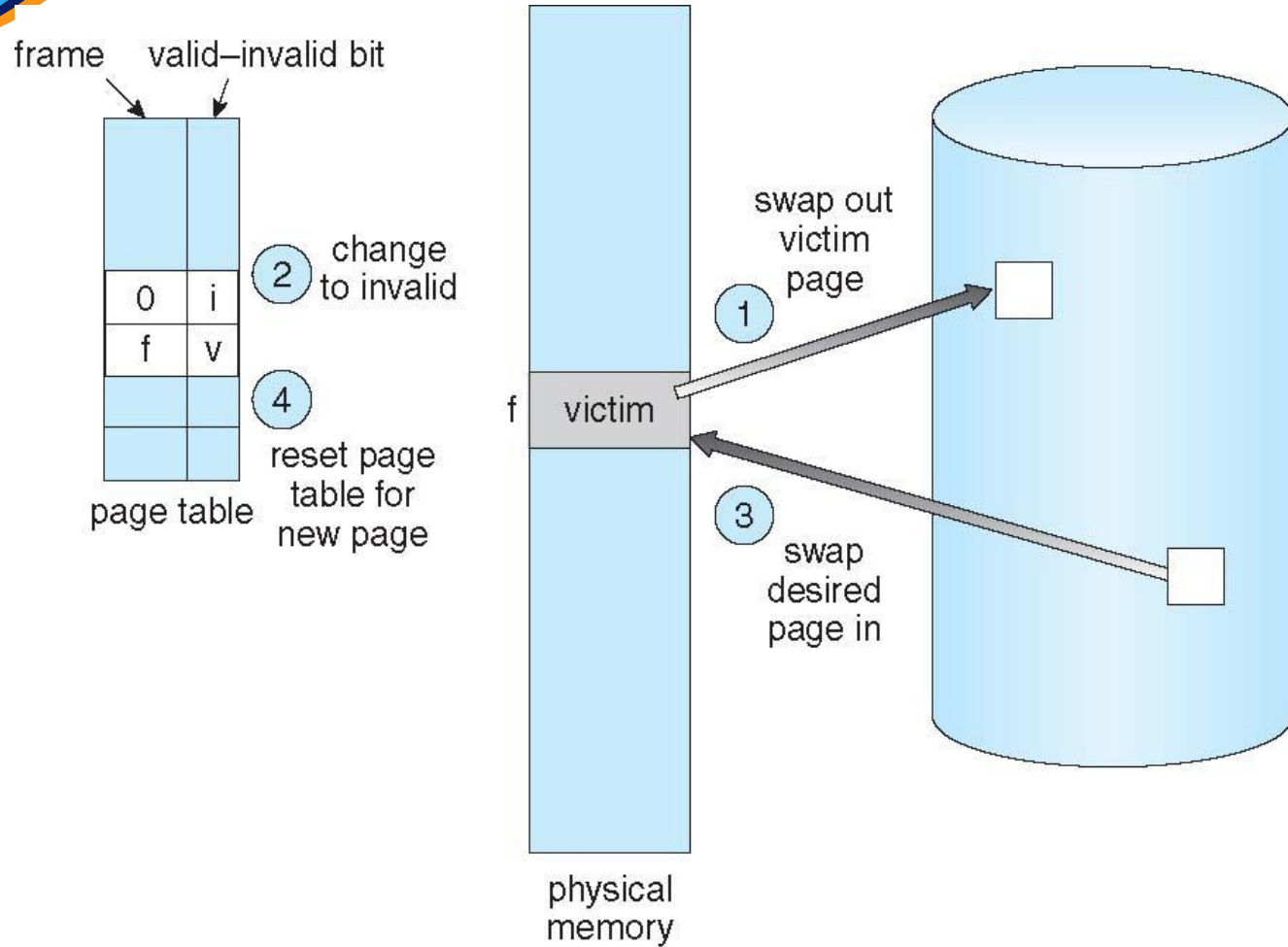
Basic Page Replacement

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**
 - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT



Page Replacement



Page and Frame Replacement Algorithms

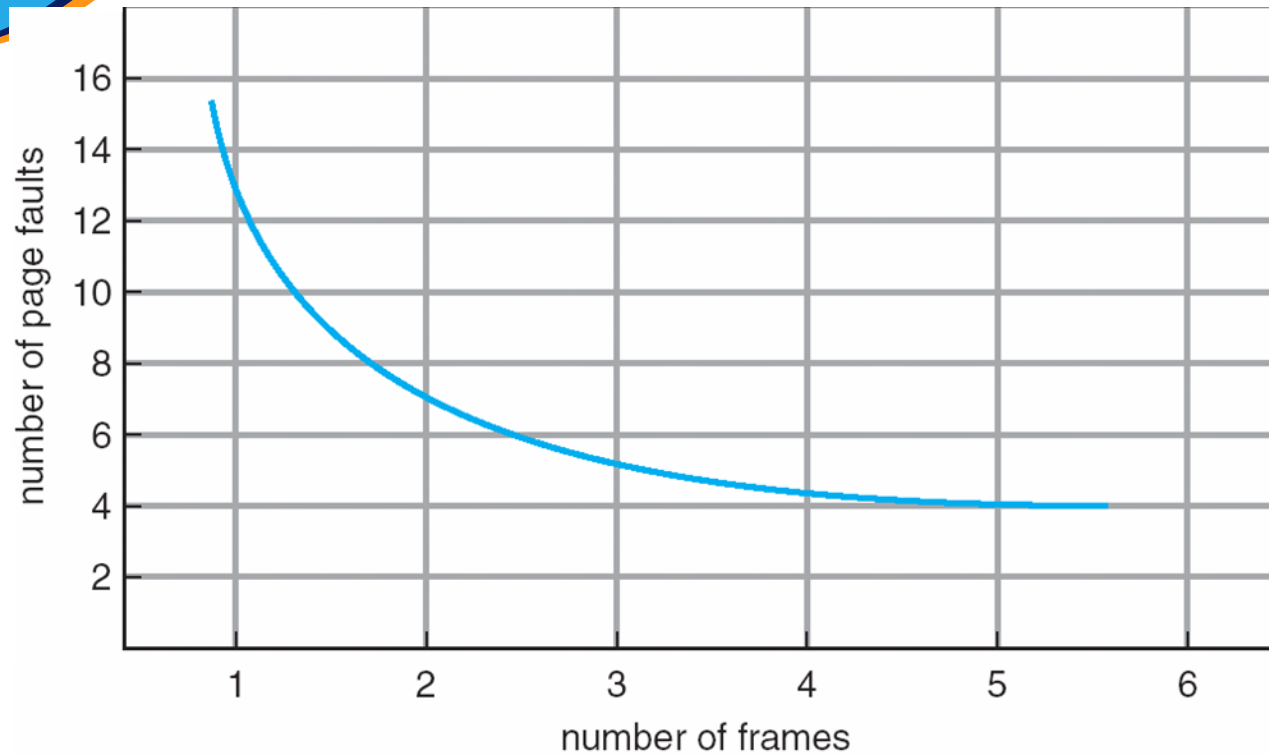
- **Frame-allocation algorithm** determines
 - How many frames to give each process
 - Which frames to replace
- **Page-replacement algorithm**
 - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
 - String is just page numbers, not full addresses
 - Repeated access to the same page does not cause a page fault
 - Results depend on number of frames available
- In all our 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



Lecture 9

Graph of Page Faults Versus the Number of Frames



First-In-First-Out (FIFO) Algorithm

- Reference string:
7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)
- Adding more frames can cause more page faults!
 - ▶ **Belady's Anomaly**
- How to track ages of pages?
 - Just use a FIFO queue



FIFO Example (3 frames)

Reference String

Frames of RAM

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



FIFO Example (3 frames)

* ← Page fault

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



FIFO Example (3 frames)

* *

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



FIFO Example (3 frames)

* * *

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

Now: Frames are all full. On next page fault, we will need to select victim frame

FIFO Example (3 frames)

Circle the victim frame

* * * *

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



FIFO Example (3 frames)

* * * *

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															
	0	0	0	0															
		1	1	1															



FIFO Example (3 frames)

* * * * *

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	2														
	0	0	0	0	3														
		1	1	1	1														



FIFO Example (3 frames)

* * * * *

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	2	2													
	0	0	0	0	3	3													
		1	1	1	1	0													



FIFO Example (3 frames)

* * * * * * *

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



FIFO Example (3 frames)

* * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * * * * * *

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



FIFO Example (3 frames)

* * * * * * * * * * * * * *

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



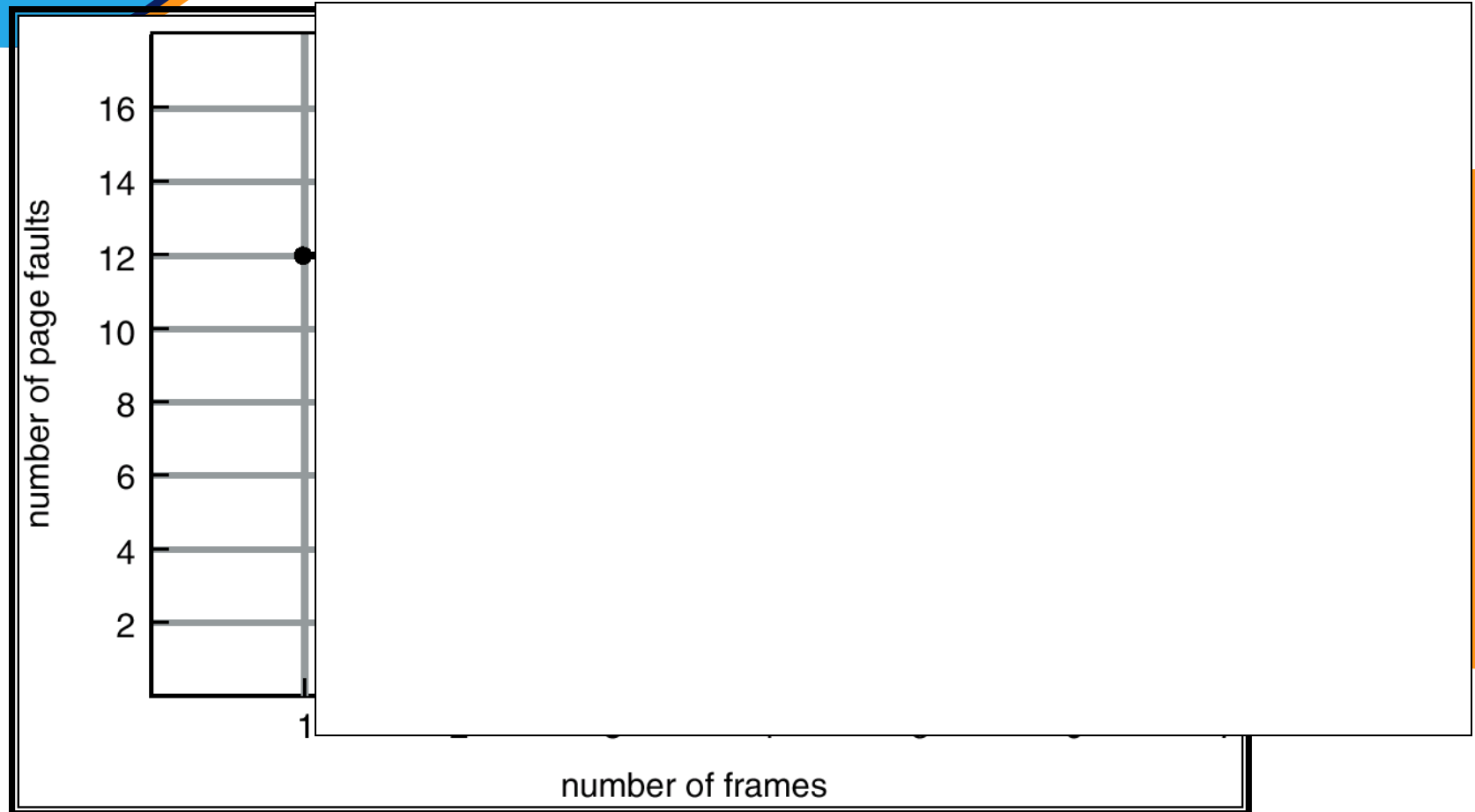
FIFO Example (3 frames)

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

So, 15 page faults for this reference string, FIFO, & 3 frames



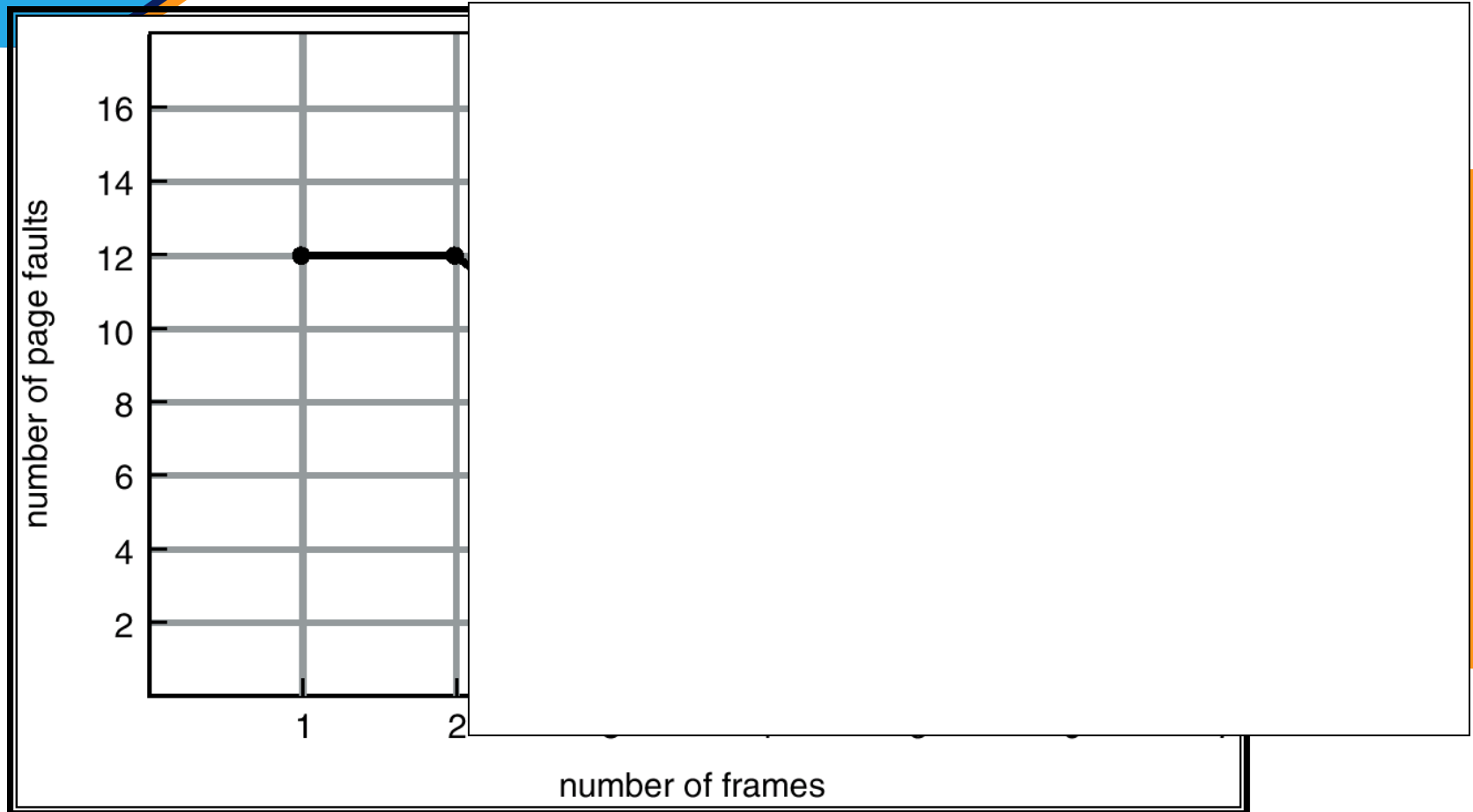
Belady's Anomaly



Graph gives performance of FIFO on reference string:

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

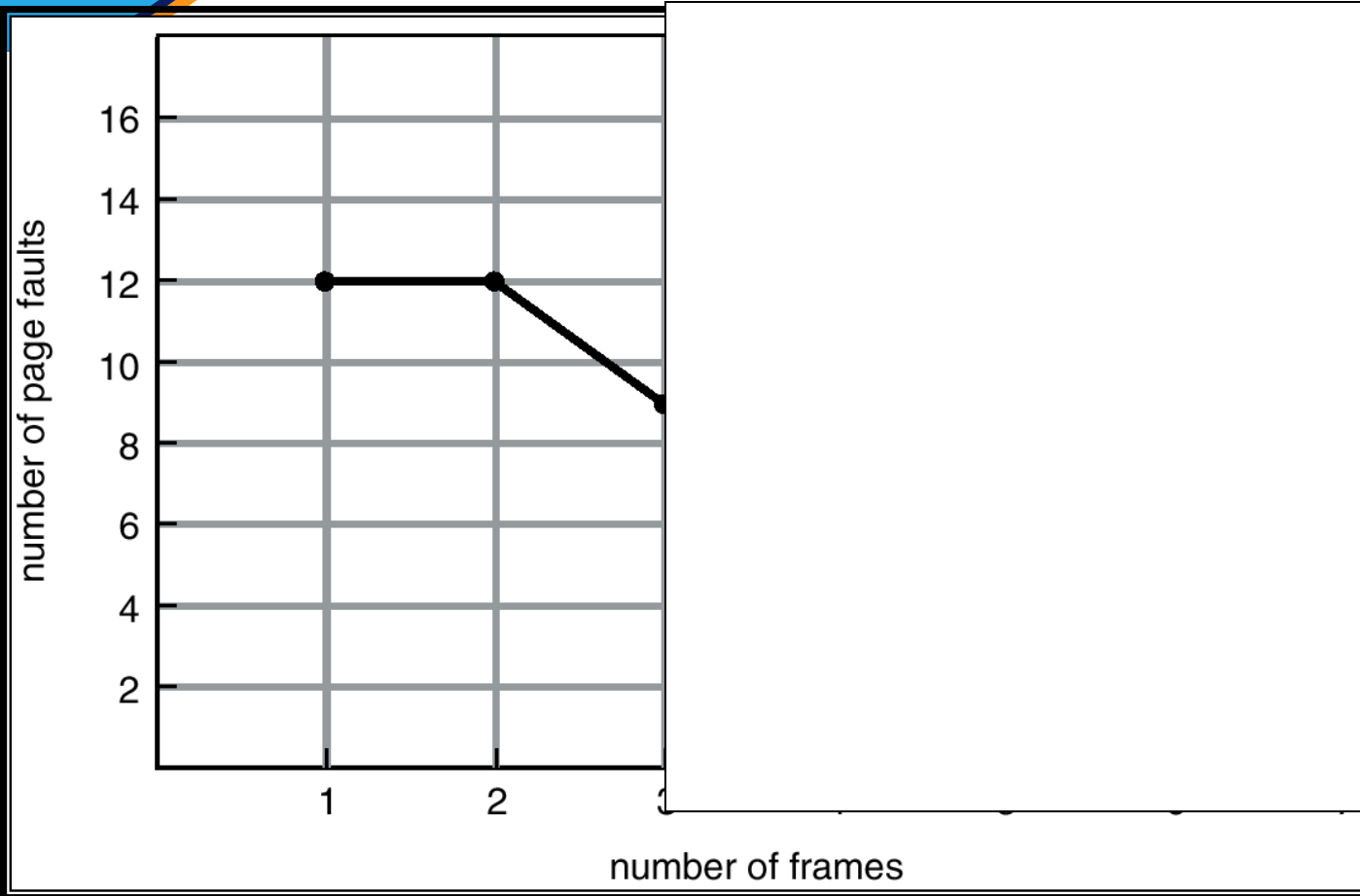
Belady's Anomaly



Graph gives performance of FIFO on reference string:

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

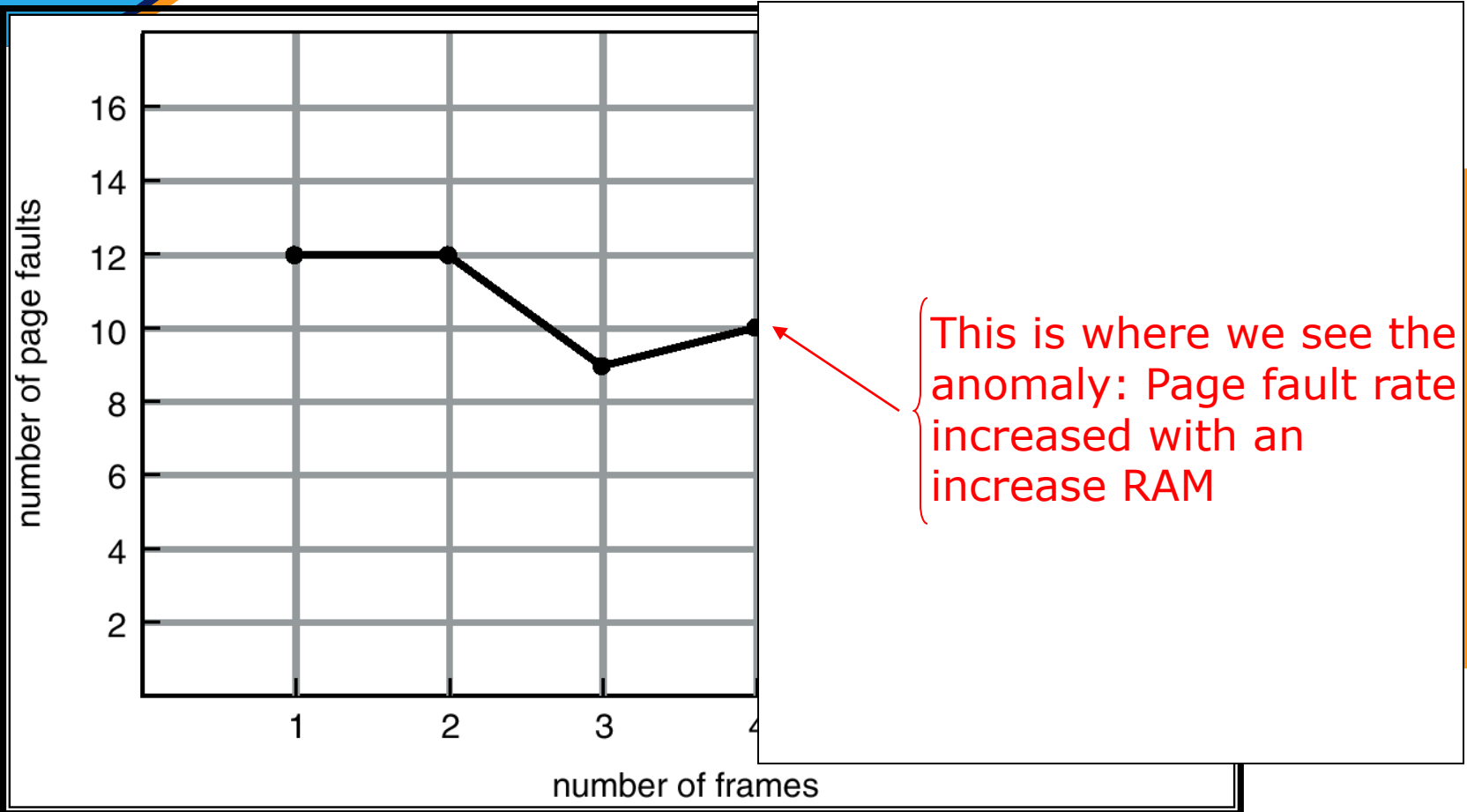
Belady's Anomaly



Graph gives performance of FIFO on reference string:

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

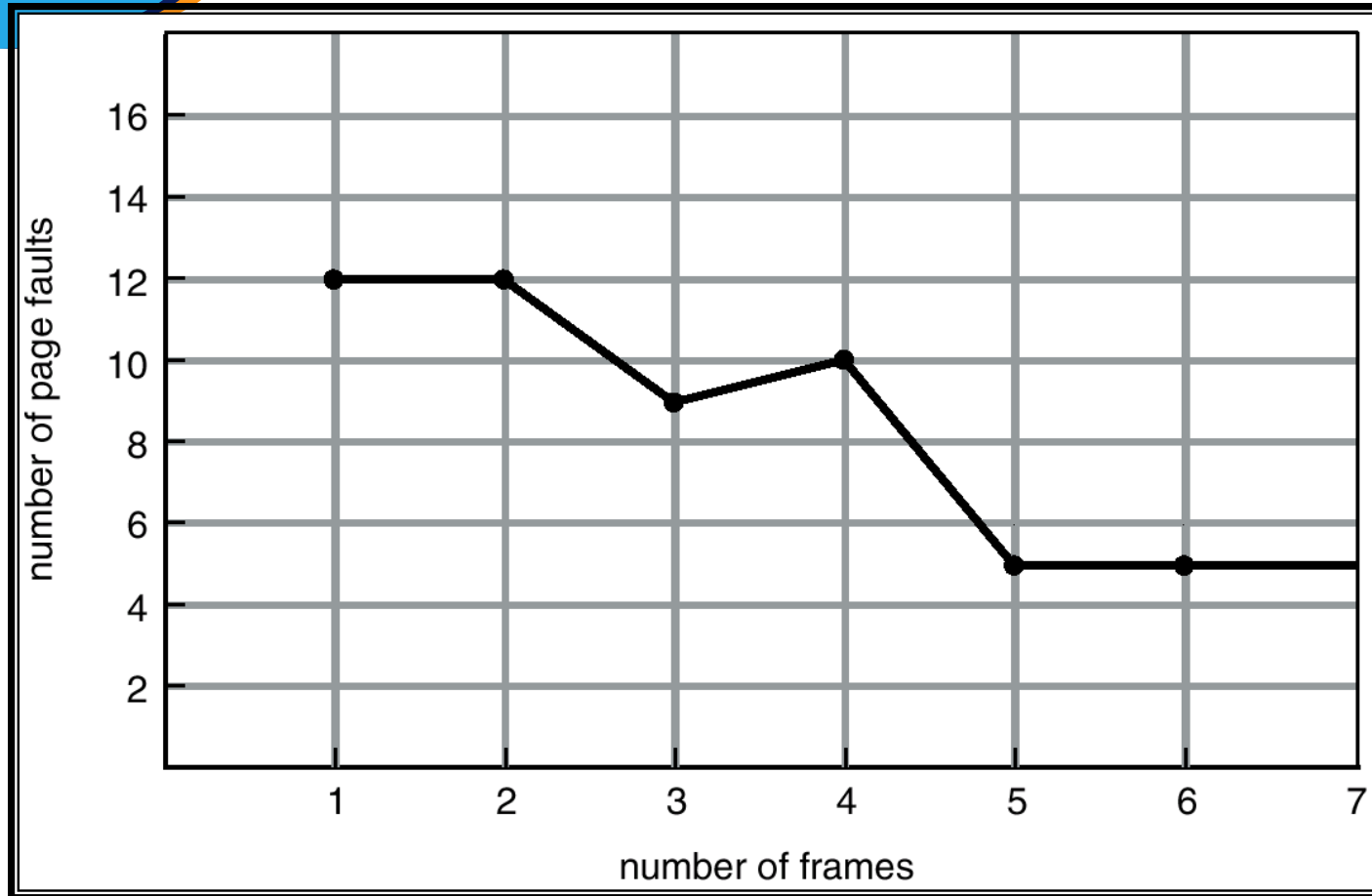
Belady's Anomaly



Graph gives performance of FIFO on reference string:

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

Belady's Anomaly



Graph gives performance of FIFO on reference string:

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
 - 9 is optimal for the example
- How do you know this?
 - Can't read the future
- Used for measuring how well your algorithm performs

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

page frames



OPT Example (3 frames)

Reference String

Frames of RAM

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



OPT Example (3 frames)

*

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



OPT Example (3 frames)

* *

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



OPT Example (3 frames)

* * *

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



OPT Example (3 frames)

* * * *

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

0 and 1 are used before 7 in the future, so replace 7

OPT Example (3 frames)

* * * *

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															
	0	0	0	0															
		1	1	1															



OPT Example (3 frames)

* * * * *

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	2														
	0	0	0	0	0														
		1	1	1	3														



OPT Example (3 frames)

* * * * *

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	2	2													
	0	0	0	0	0	0													
		1	1	1	3	3													



OPT Example (3 frames)

* * * * *

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



OPT Example (3 frames)

* * * * *

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



OPT Example (3 frames)

* * * * *

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



OPT Example (3 frames)

* * * * *

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



OPT Example (3 frames)

* * * * *

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



OPT Example (3 frames)

* * * * *

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



OPT Example (3 frames)

*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
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	2	2	2	2	2	2	2	2	2						
	0	0	0	0	0	0	4	4	4	0	0	0	0						
		1	1	1	3	3	3	3	3	3	3	3	1						



OPT Example (3 frames)

* * * * * * *

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



OPT Example (3 frames)

* * * * * * *

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



OPT Example (3 frames)

*	*	*	*		*	*			*			*							
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	2	2	2	2	2	2	2	2	2	2	2	2			
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0			
		1	1	1	3	3	3	3	3	3	3	3	3	1	1	1	1		



OPT Example (3 frames)

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



OPT Example (3 frames)

* * * * * * *

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



Least Recently Used (LRU) Algorithm

- Well, OPT is not possible... we don't know the sequence of future pages required by a process
- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



LRU Example (3 frames)

Reference String

Frames of RAM

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



LRU Example (3 frames)

*

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



LRU Example (3 frames)

* *

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



LRU Example (3 frames)

* * *

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



LRU Example (3 frames)

* * * *

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

Page 7 has not been used for the longest period of time, so replace it

LRU Example (3 frames)

* * * *

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															
	0	0	0	0															
		1	1	1															



Lecture 9

LRU Example (3 frames)

* * * * *

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	2														
	0	0	0	0	0														
		1	1	1	3														



LRU Example (3 frames)

* * * * *

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	2	2													
	0	0	0	0	0	0													
		1	1	1	3	3													



Lecture 9

LRU Example (3 frames)

* * * * *

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



Lecture 9

LRU Example (3 frames)

* * * * * * *

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



Lecture 9

LRU Example (3 frames)

* * * * * * * *

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



Lecture 9

LRU Example (3 frames)

* * * * * * * *

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



Lecture 9

LRU Example (3 frames)

* * * * * * * *

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



Lecture 9

LRU Example (3 frames)

* * * * * * * *

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



Lecture 9

LRU Example (3 frames)

* * * * * * * * *

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



LRU Example (3 frames)

* * * * * * * * *

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



LRU Example (3 frames)

* * * * * * * * * *

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



LRU Example (3 frames)

*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
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	2	2	4	4	4	0	0	0	1	1	1	1			
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0			
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2			
																			



LRU Example (3 frames)

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



LRU Example (3 frames)

* * * * * * * * * * * *

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



Lecture 9

LRU Example (3 frames)

* * * * * * * * * * * *

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



Comparison (On Example)

Page replacement algorithm	Number of faults (including first three)	Number of faults (not including first three)
FIFO	15	12
OPT	9	6
LRU	12	9



Allocation of Frames

- Each process needs **minimum** number of frames
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
 - instruction is 6 bytes, might span 2 pages
 - 2 pages to handle *from*
 - 2 pages to handle *to*
- **Maximum** of course is total frames in the system
- Two major allocation schemes
 - fixed allocation
 - priority allocation
- Many variations



Fixed Allocation

- Equal allocation – For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
 - Keep some as free frame buffer pool
- Proportional allocation – Allocate according to the size of process
 - Dynamic as degree of multiprogramming, process sizes change

– s_i = size of process p_i

– $S = \sum s_i$

– m = total number of frames

– a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \cdot 64 \gg 4$$

$$a_2 = \frac{127}{137} \cdot 64 \gg 57$$



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
 - But then process execution time can vary greatly
 - But greater throughput so more common
- **Local replacement** – each process selects from only its own set of allocated frames
 - More consistent per-process performance
 - But possibly underutilized memory



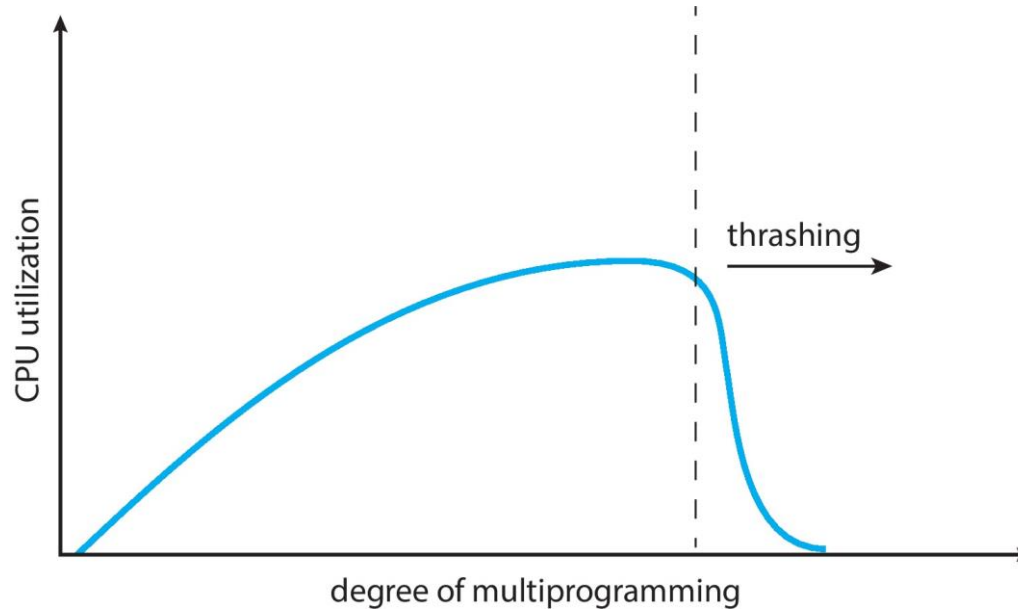
Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
 - Page fault to get page
 - Replace existing frame
 - But quickly need replaced frame back
 - This leads to:
 - ▶ Low CPU utilization
 - ▶ Operating system thinking that it needs to increase the degree of multiprogramming
 - ▶ Another process added to the system



Thrashing (Cont.)

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



Demand Paging and Thrashing

- Why does demand 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
- Limit effects by using local or priority page replacement



Operating System Examples Windows

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page
- Processes are assigned **working set minimum** and **working set maximum**
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory
- Working set trimming removes pages from processes that have pages in excess of their working set minimum



End of Lecture 9

