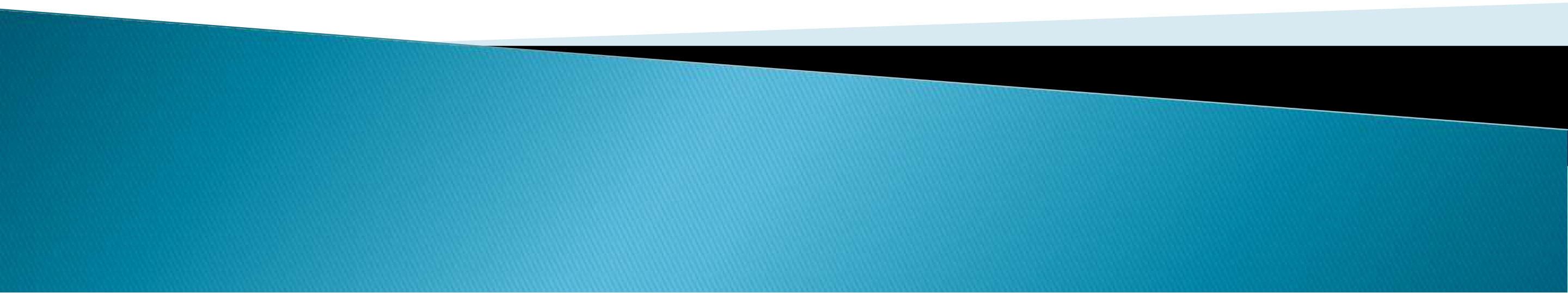



Virtual Memory

Course Instructor: Nausheen Shoaib



Definition of Virtual Memory [1 / 2]

- ▶ **Virtual memory** is a **memory** management capability of an OS
 - ▶ uses hardware and software to allow a computer to compensate for **physical memory** shortages by temporarily transferring data from random access **memory** (RAM) to disk storage.
- 

Definition of Virtual Memory [2/2]

- ▶ Virtual memory is not RAM (Random access memory) on memory chips.
- ▶ Virtual memory is an area of hard drive or other storage space, which OS uses as swap space (overflow) for the physical RAM.

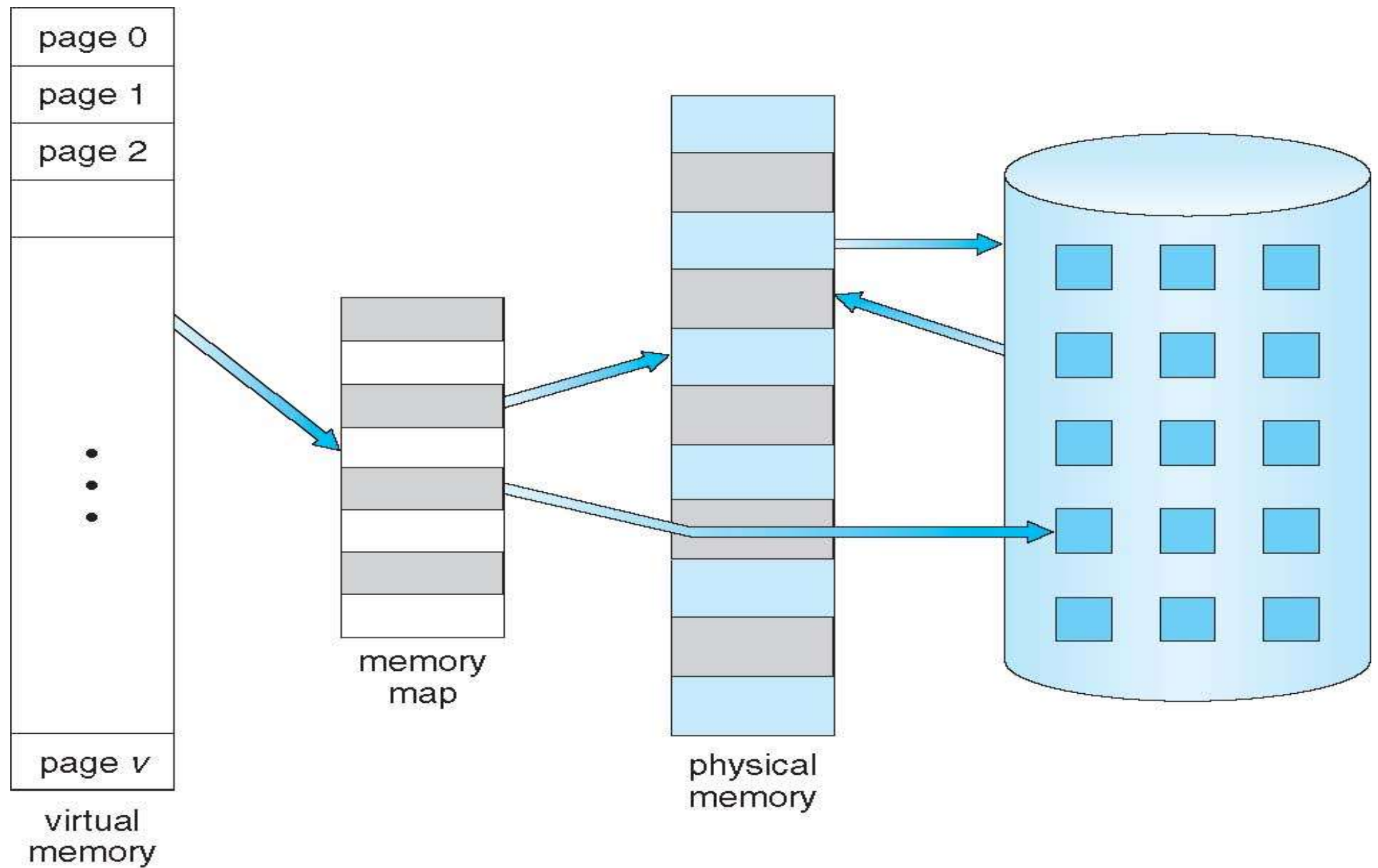
When the demands of the software and data exceed the physical RAM size, the operating system copies to hard disk blocks of data from RAM that have been seldom used, and releases that RAM for use by the current process.

- ▶ When the block (or Page) is required again, the OS makes room in RAM by copying another block to hard drive and copying in the data from the first block.

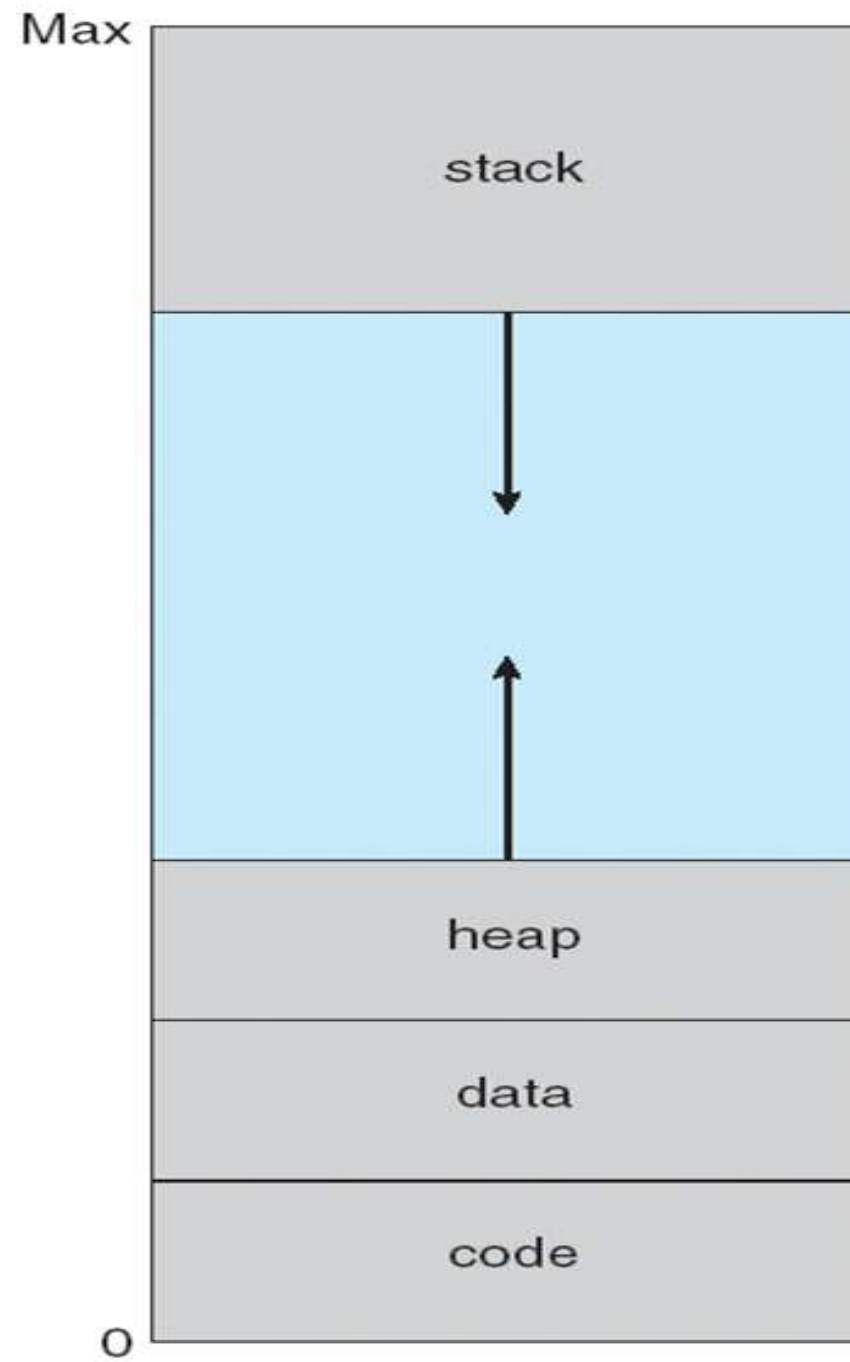
Background

- ▶ **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 can be implemented via:
 - Demand paging


Virtual Memory That is Larger Than Physical Memory



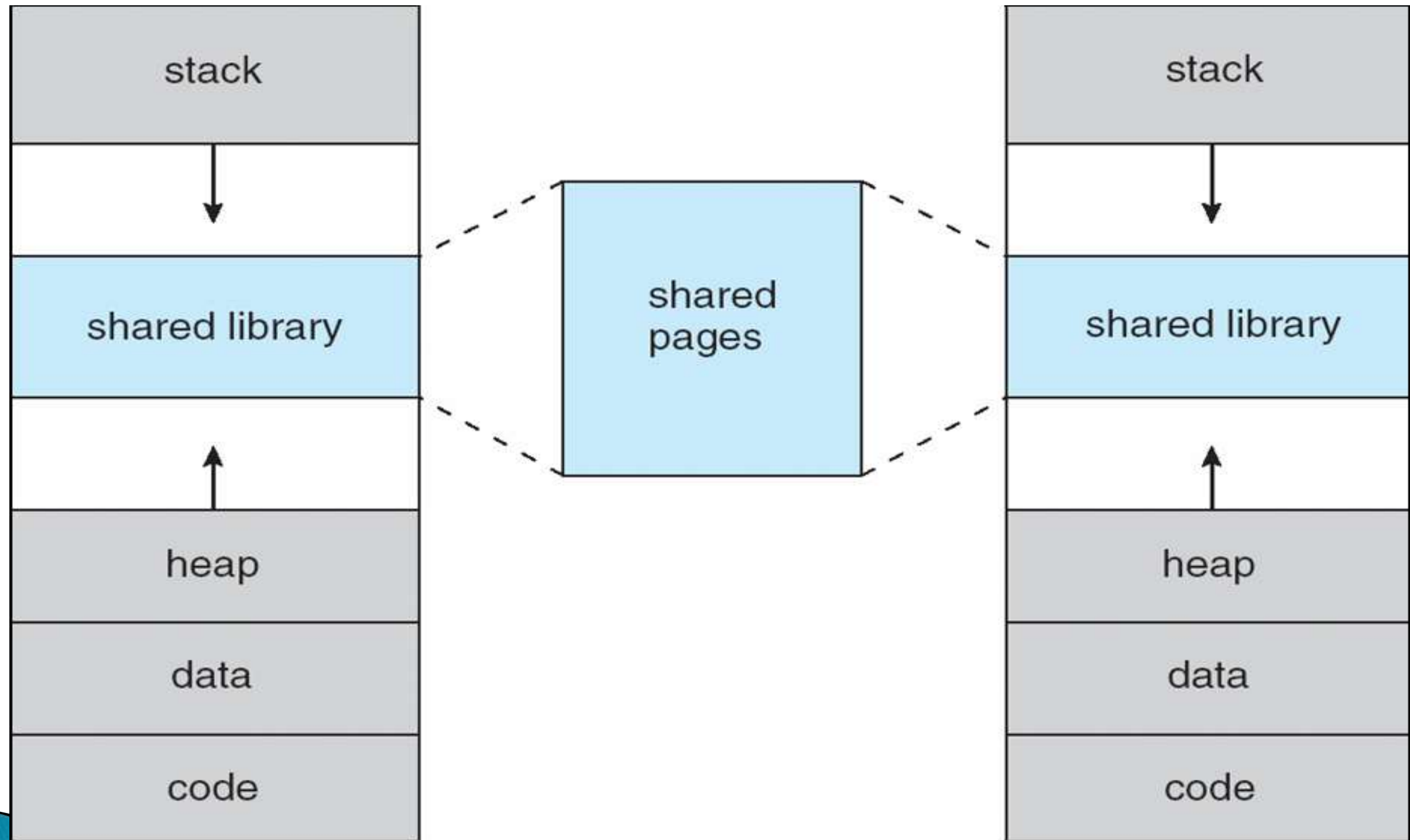
Virtual-address Space



Virtual Address Space

- ▶ Virtual address spaces that include holes are known as **sparse** address spaces.
 - ▶ 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
- 

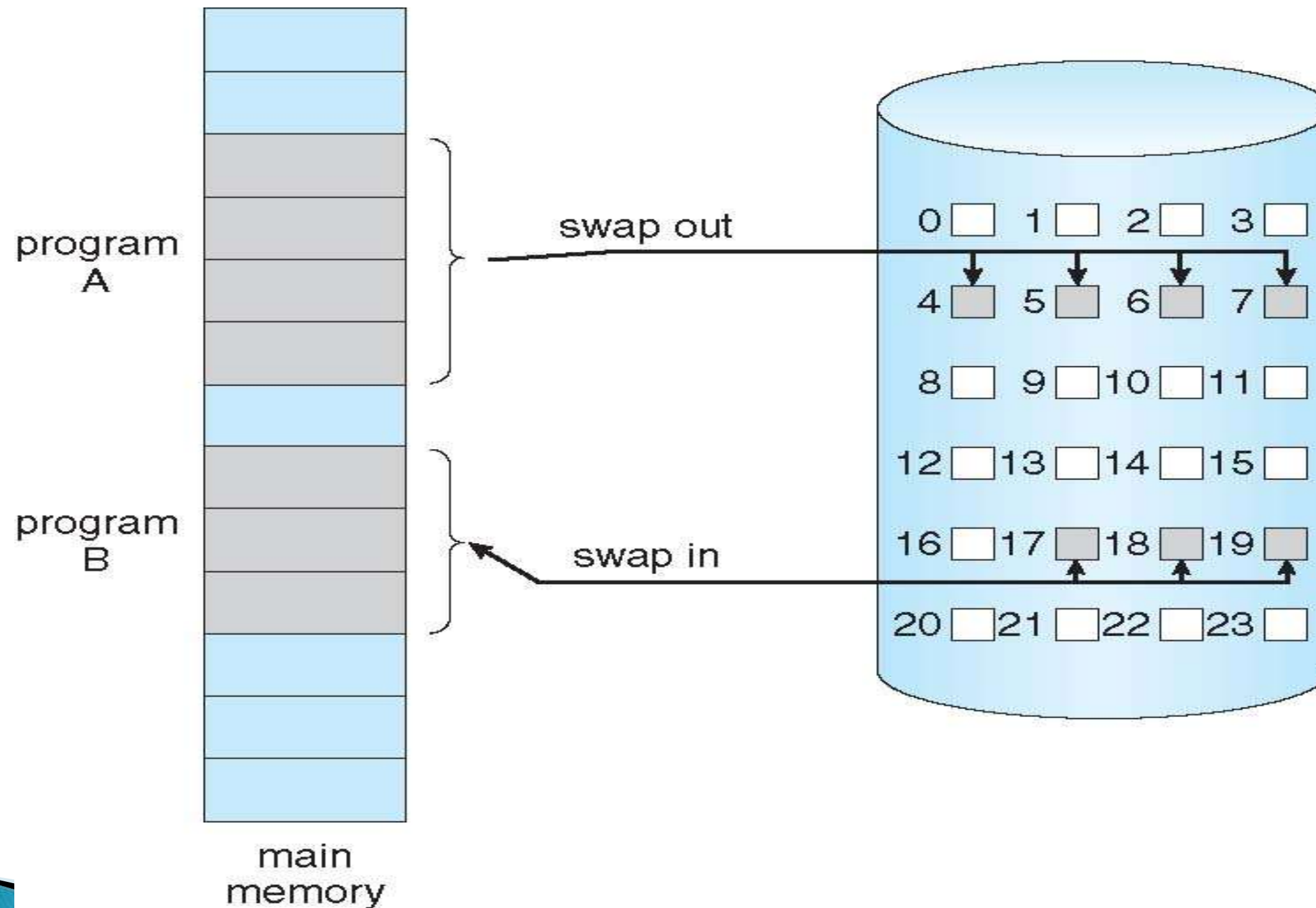
Shared Library Using Virtual Memory



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
- ▶ 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**

Transfer of a Paged Memory to Contiguous Disk Space



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

Page Table When Some Pages Are Not in Main Memory

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

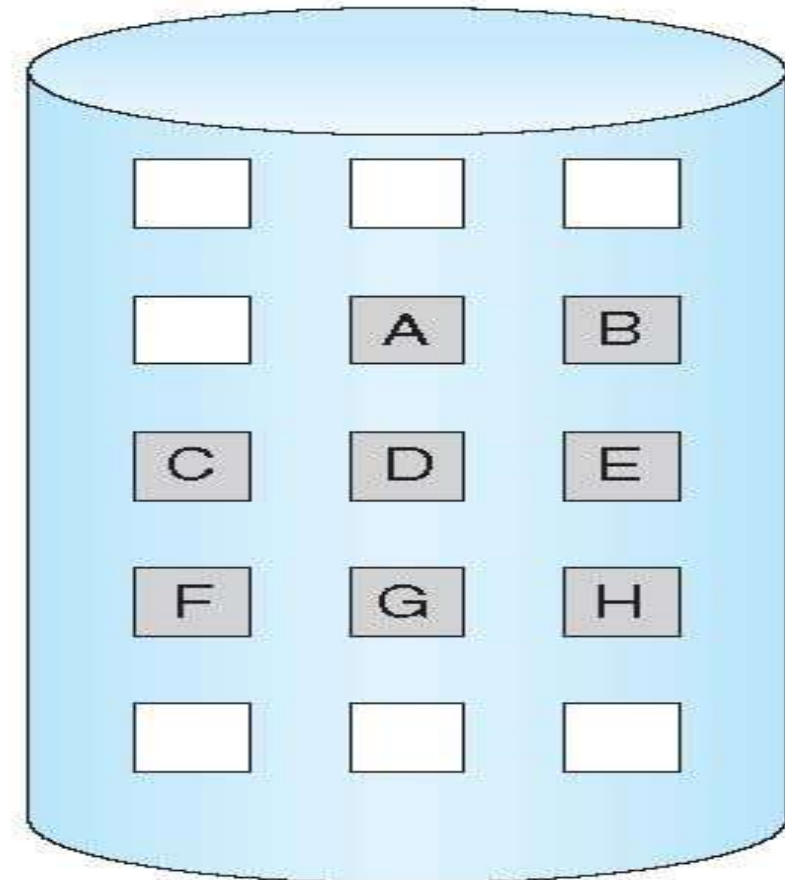
logical
memory

valid-invalid bit		
frame		
0	4	v
1		i
2	6	v
3		i
4		i
5	9	v
6		i
7		i

page table

0	
1	
2	
3	
4	A
5	
6	C
7	
8	
9	F
10	
11	
12	
13	
14	
15	

physical memory



Page Fault

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

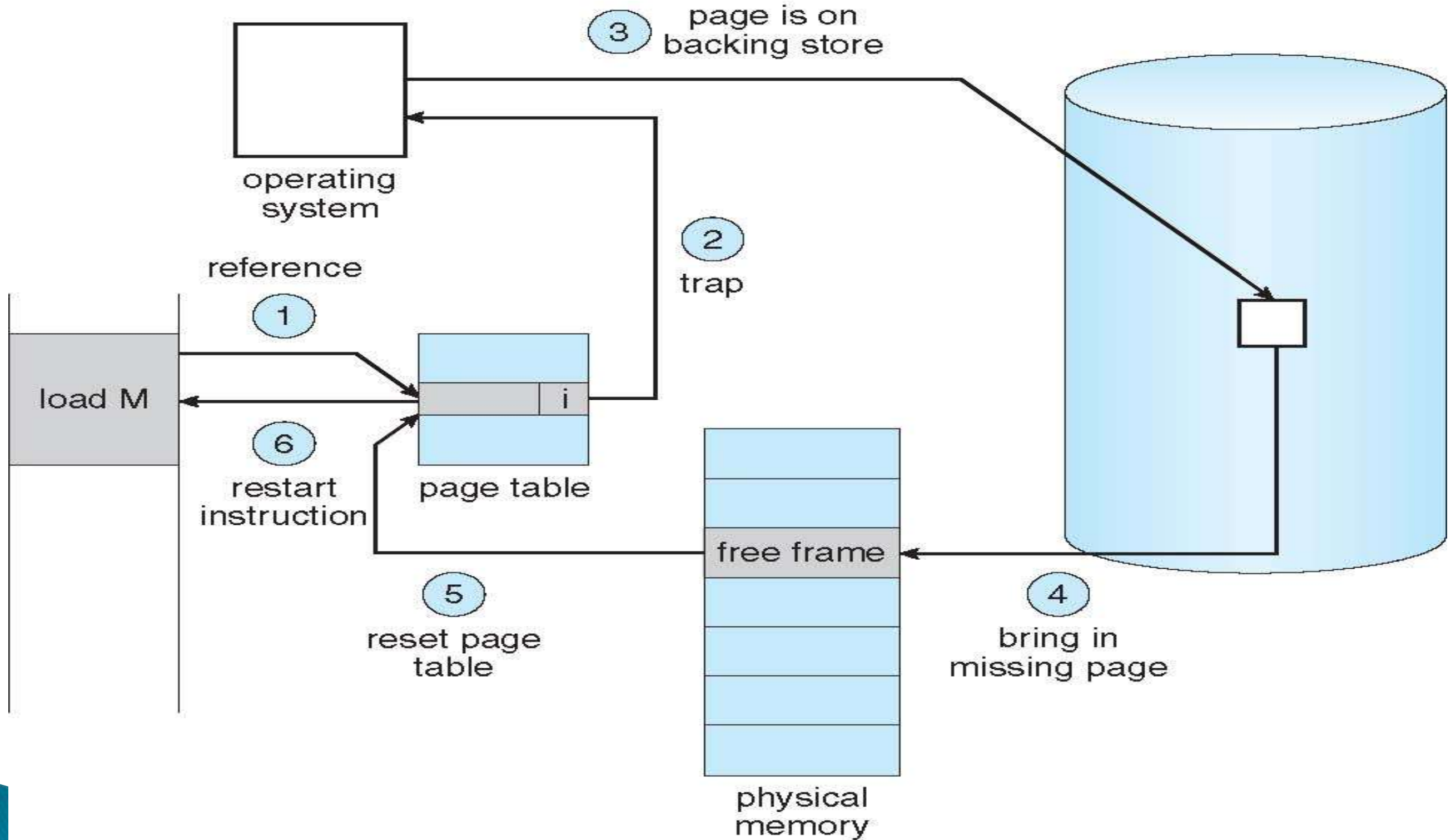
page fault

1. Operating system looks at another table to decide:
 - Invalid reference \Rightarrow abort
 - Just not in memory
2. Get empty frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
Set validation bit = **V**
5. Restart the instruction that caused the page fault

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
 - 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

Steps in Handling a Page Fault



Performance of Demand Paging (Cont.)

- ▶ Page Fault Rate $0 \leq p \leq 1$
 - if $p = 0$ no page faults
 - if $p = 1$, every reference is a fault
- ▶ Effective Access Time (EAT)
effective access time = $(1 - p) \times ma$
+ $p \times$ page fault service time.

Performance of Demand Paging (Cont.)

Memory Access Time = 200 nanoseconds

Average page fault service time = 8 millisecond

If one access out of 1,000 causes a page fault, then
 $p = 1 / 1000 = 0.001$

effective access time = $(1 - p) \times ma + p \times \text{page fault time.}$

$$\text{EAT} = (1 - 0.001) \times 200 + (0.001 \times 8 \times 10^{-6})$$

$$= 8199.8 \text{ nanosecond}$$

$$= 8.199 \text{ microsecond}$$

Millisecond = 10×10^{-3}

Microsecond = 10×10^{-6}

Nanosecond = 10×10^{-9}

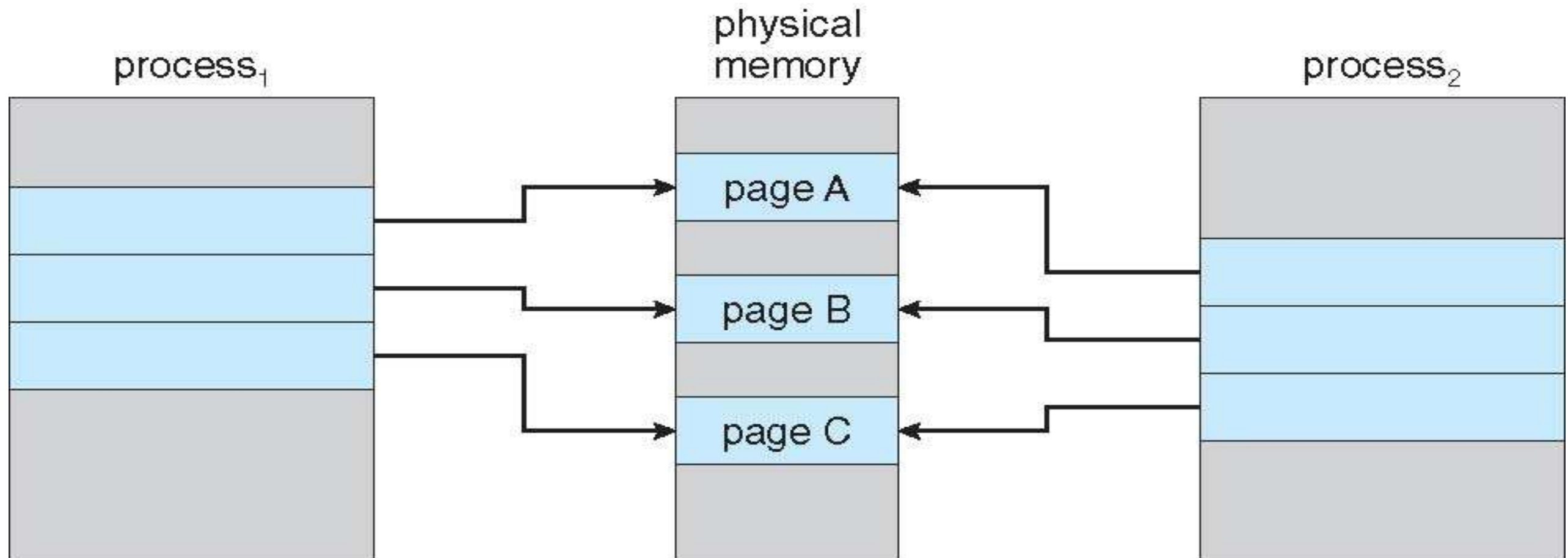
Demand Paging Optimizations

- ▶ Copy entire process image to swap space at process load time
 - Then page in and out of swap space
 - Used in older BSD Unix
- ▶ Demand page in from program binary on disk, but discard rather than paging out when freeing frame
 - Used in Solaris and current BSD

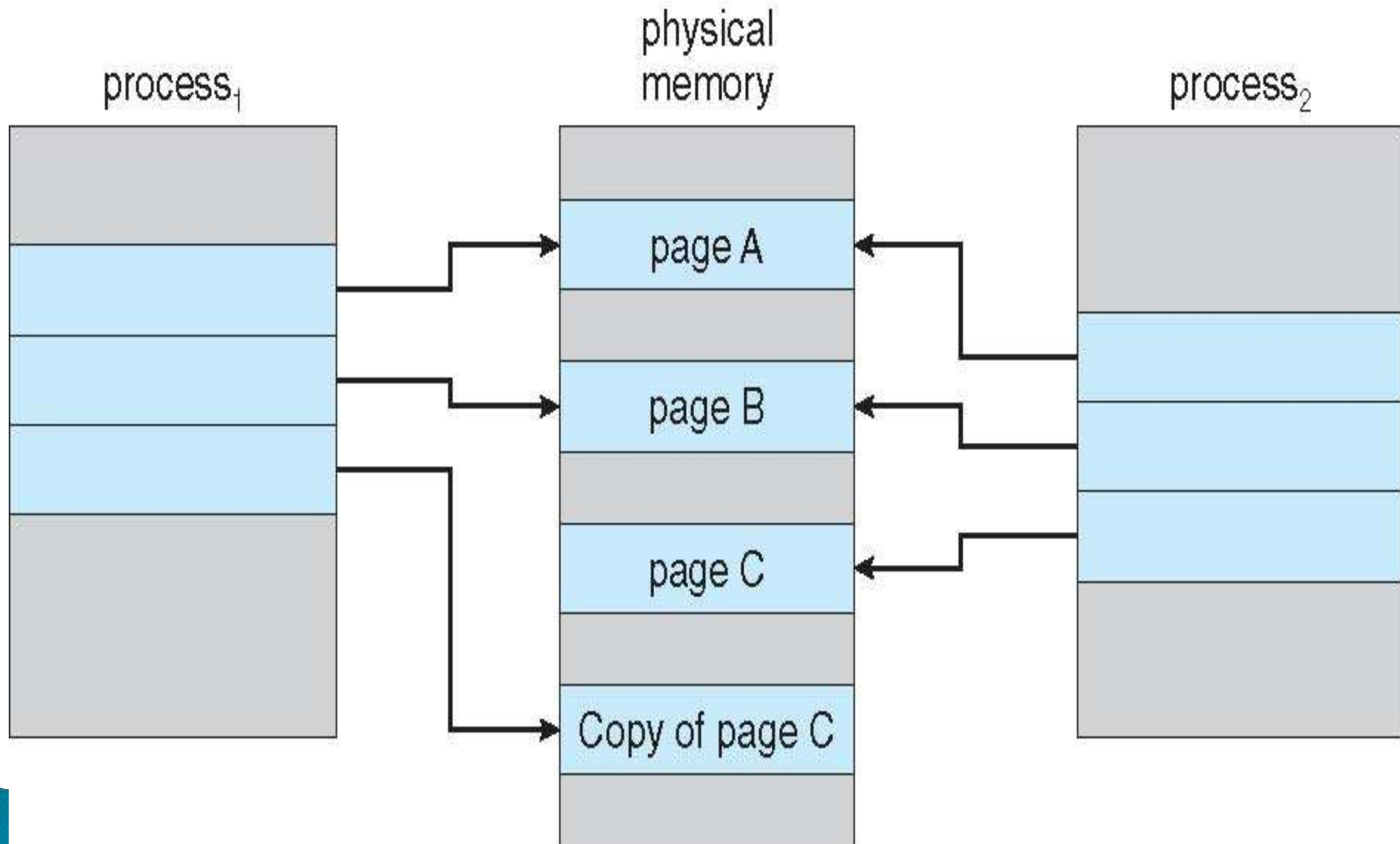
Copy-on-Write

- ▶ **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
- ▶ `vfork()` variation on `fork()` system call has parent suspend and child using copy-on-write address space of parent
 - Designed to have child call `exec()`
 - Very efficient

Before Process 1 Modifies Page C



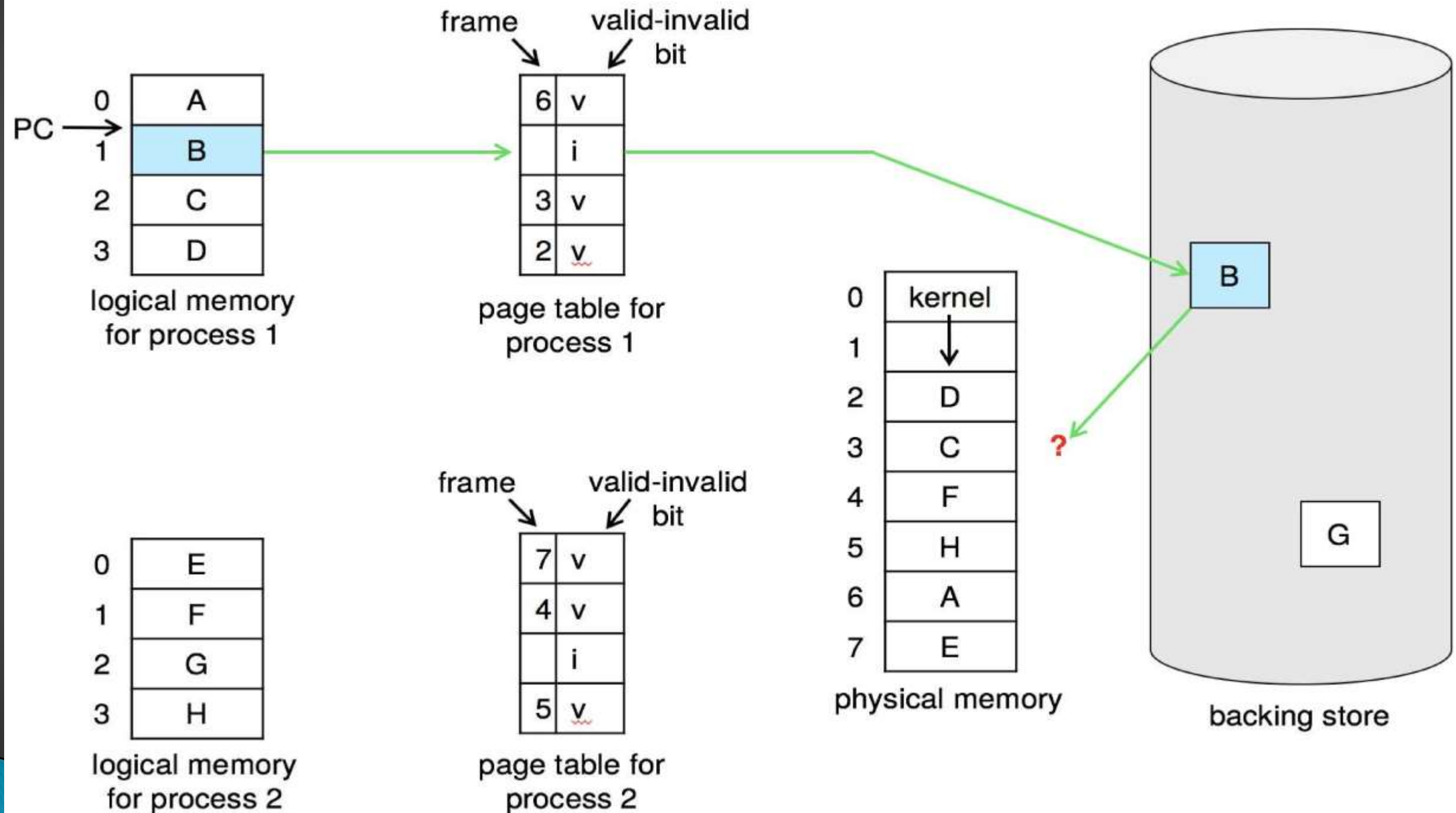
After Process 1 Modifies Page C



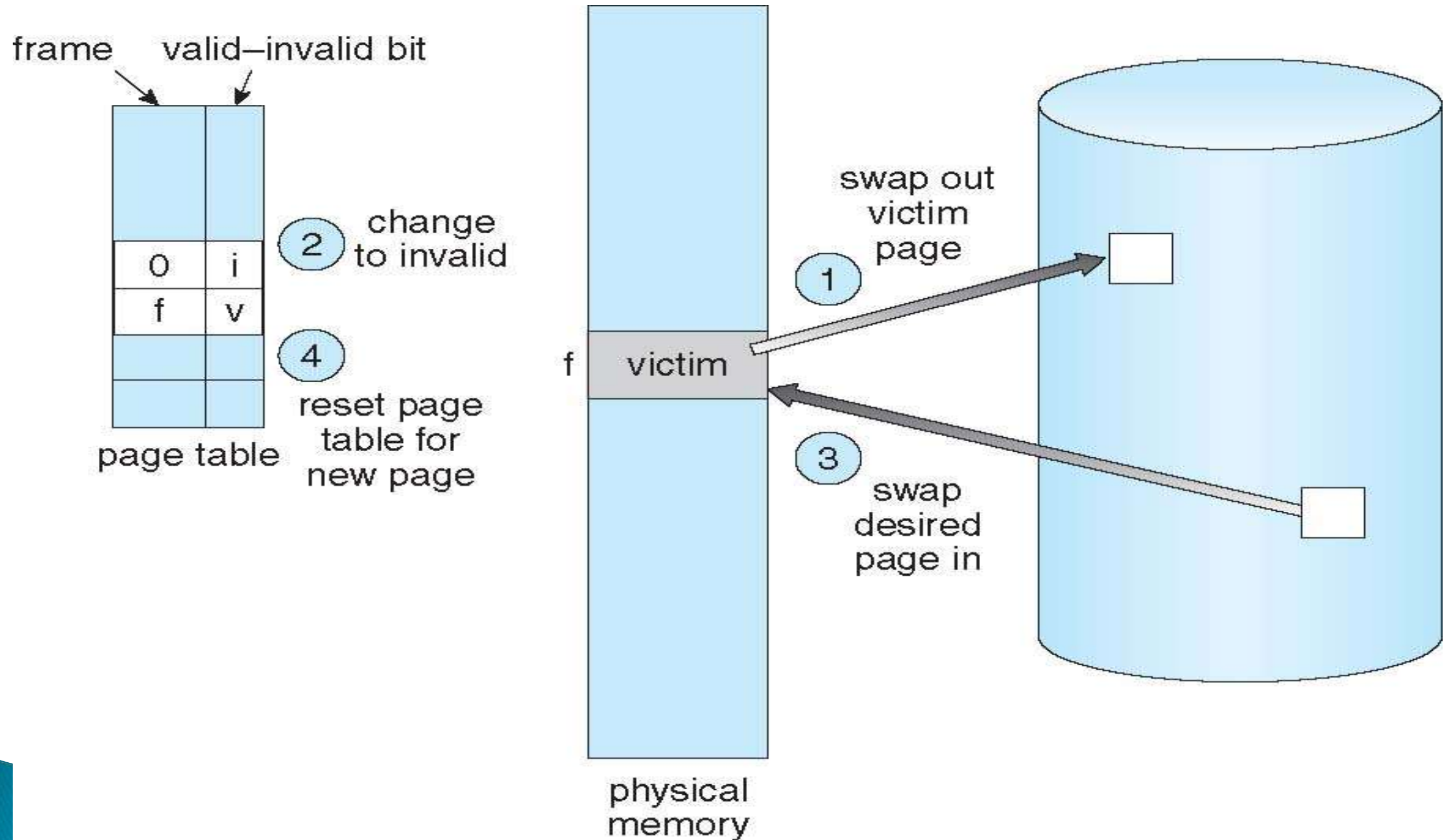
What Happens if There is no Free Frame?

- ▶ Used up by process pages
- ▶ Also in demand from the kernel, I/O buffers, etc
- ▶ How much to allocate to each?
- ▶ 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

Need For Page Replacement



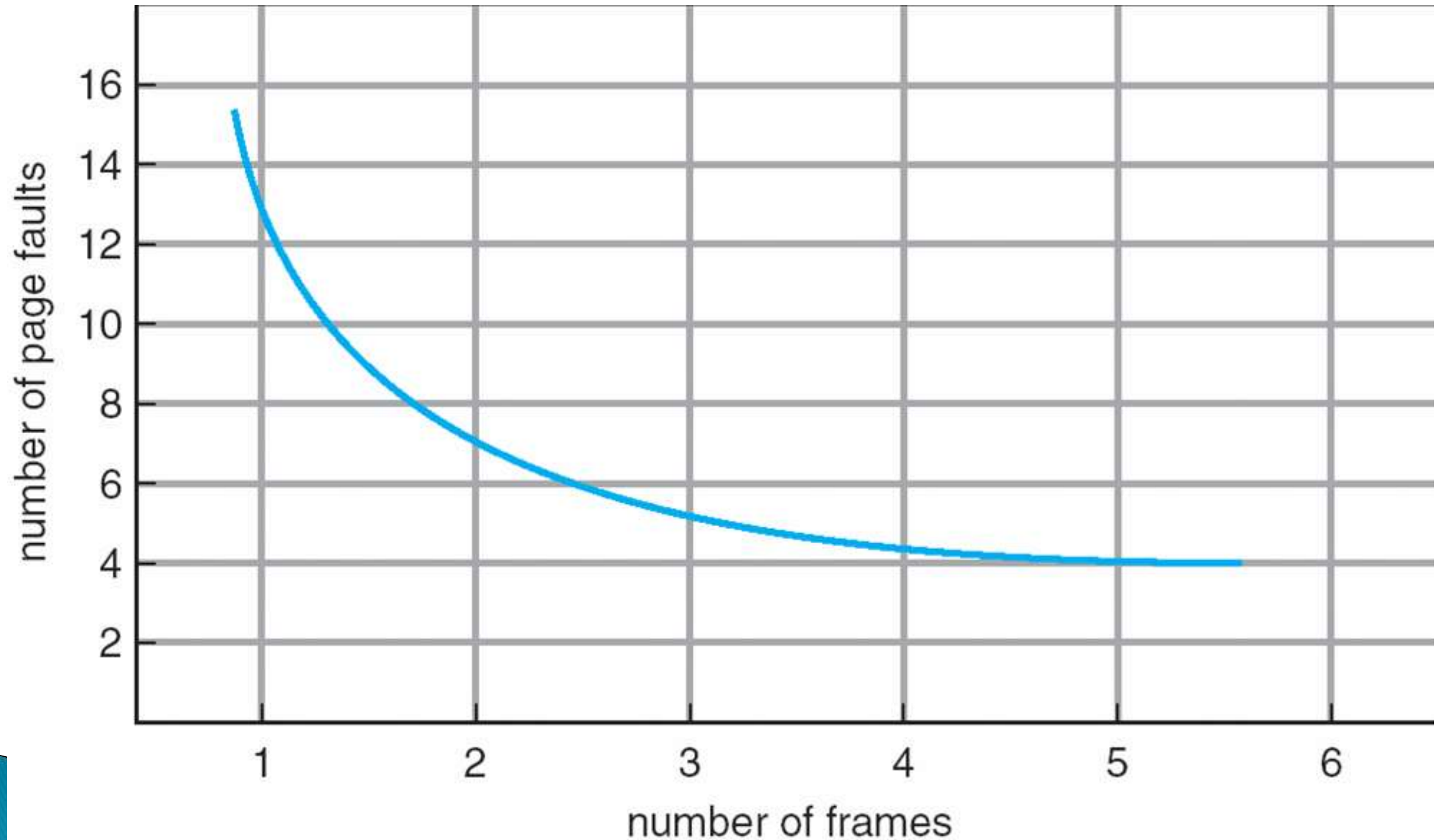
Page Replacement



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

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)

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


15 page faults

- ▶ Can vary by reference string: consider
1,2,3,4,1,2,5,1,2,3,4,5
 - Adding more frames can cause more page faults!
 - **Belady's Anomaly**
- ▶ How to track ages of pages?
 - Just use a FIFO queue

FIFO Page Replacement Example

- ▶ Covered in Class

Least Recently Used (LRU) Algorithm

- ▶ 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 Algorithm (Cont.)

▶ Counter implementation

- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
- When a page needs to be changed, look at the counters to find smallest value
 - Search through table needed

▶ 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
- But each update more expensive
- No search for replacement

LRU Algorithm Example

- ▶ Covered in Class

Optimal Algorithm

- ▶ Replace page that will not be used for longest period of time

Optimal Page Replacement Example

- ▶ Covered in Class

LRU Approximation Algorithms

- ▶ LRU needs special hardware and still slow
- ▶ **Reference bit**
 - With each page associate a bit, initially = 0
 - When page is referenced bit set to 1
 - Replace any with reference bit = 0 (if one exists)
 - We do not know the order, however
- ▶ **Second-chance algorithm**
 - Generally FIFO, plus hardware-provided reference bit
 - Clock replacement
 - If page to be replaced has
 - Reference bit = 0 \rightarrow replace it
 - reference bit = 1 then:
 - set reference bit 0, leave page in memory
 - replace next page, subject to same rules

Second Chance Algorithm

- ▶ Covered in Class