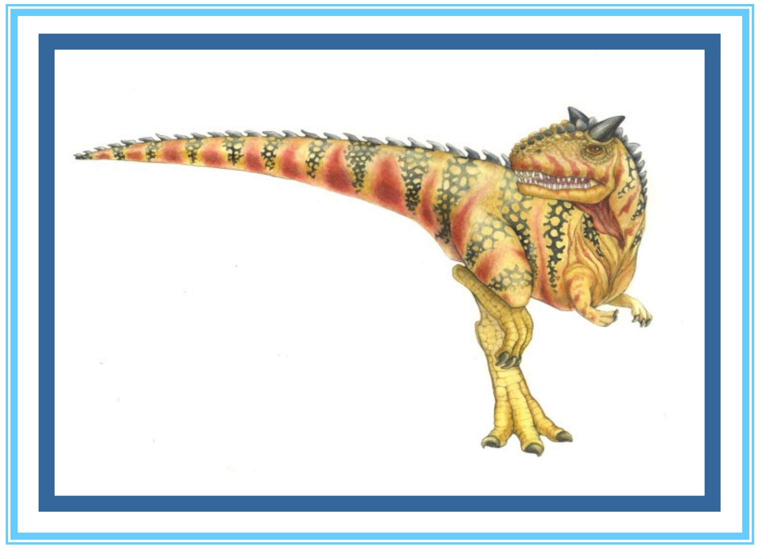
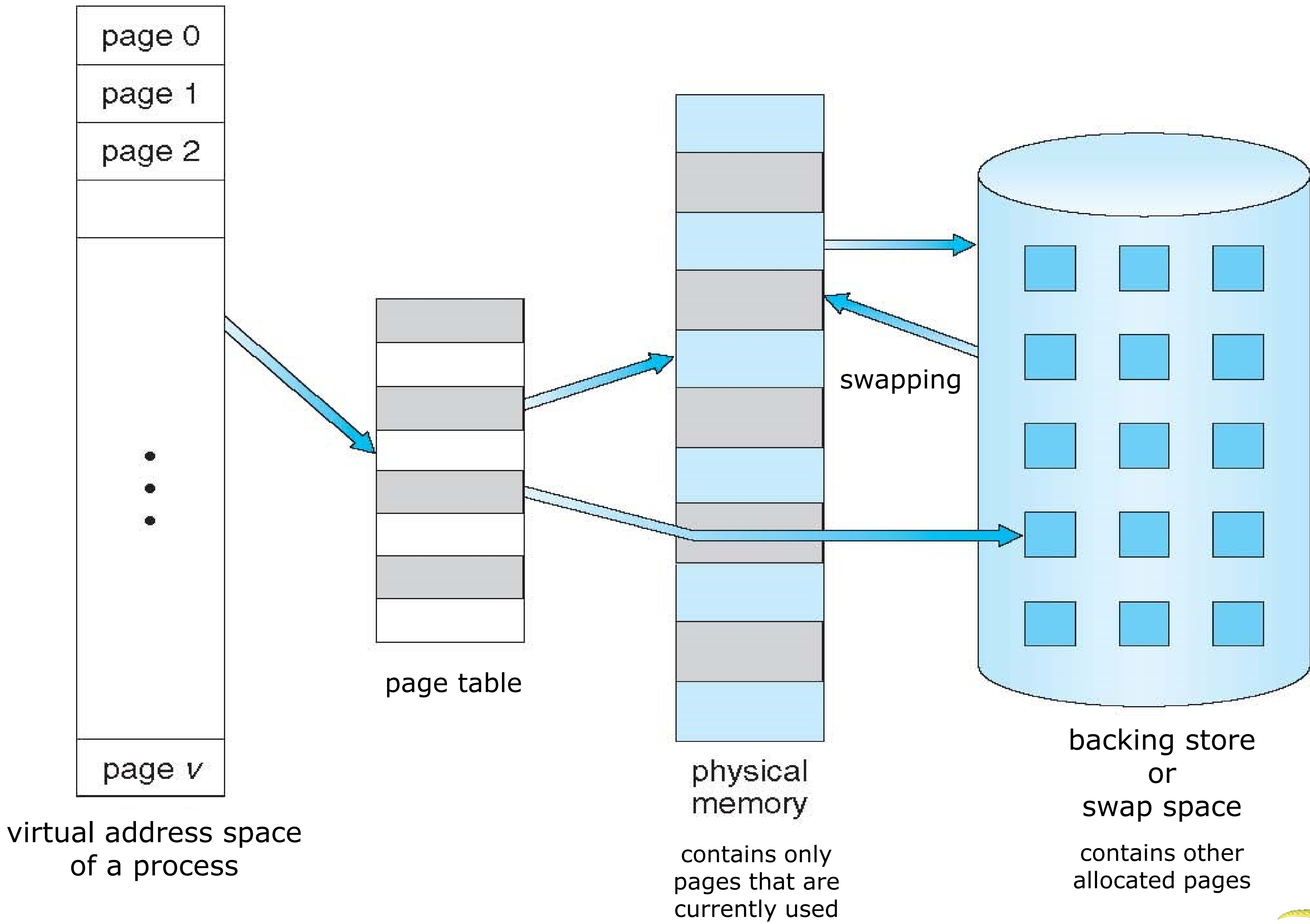
Chapter 9: Virtual-Memory Management



Virtual Memory

- Goal
  - Provide memory space for all processes that can be much larger than physical memory.
- How
  - Only some part of program code and data need to be in physical memory at a time.
    - principle of locality
  - The rest of program code and data can be stored in secondary storage (hard disk)
    - "swap partition" or "swap file"
    - extend memory hierarchy
  - Implement "Demand Paging" mechanism in OS
    - similar to "cache miss"
- Other benefits
  - Address spaces can be shared by several processes
    - shared program code, shared libraries, shared memory (IPC)
  - fast process creation (allocate memory only when needed)
  - memory-mapped file, memory-mapped I/O

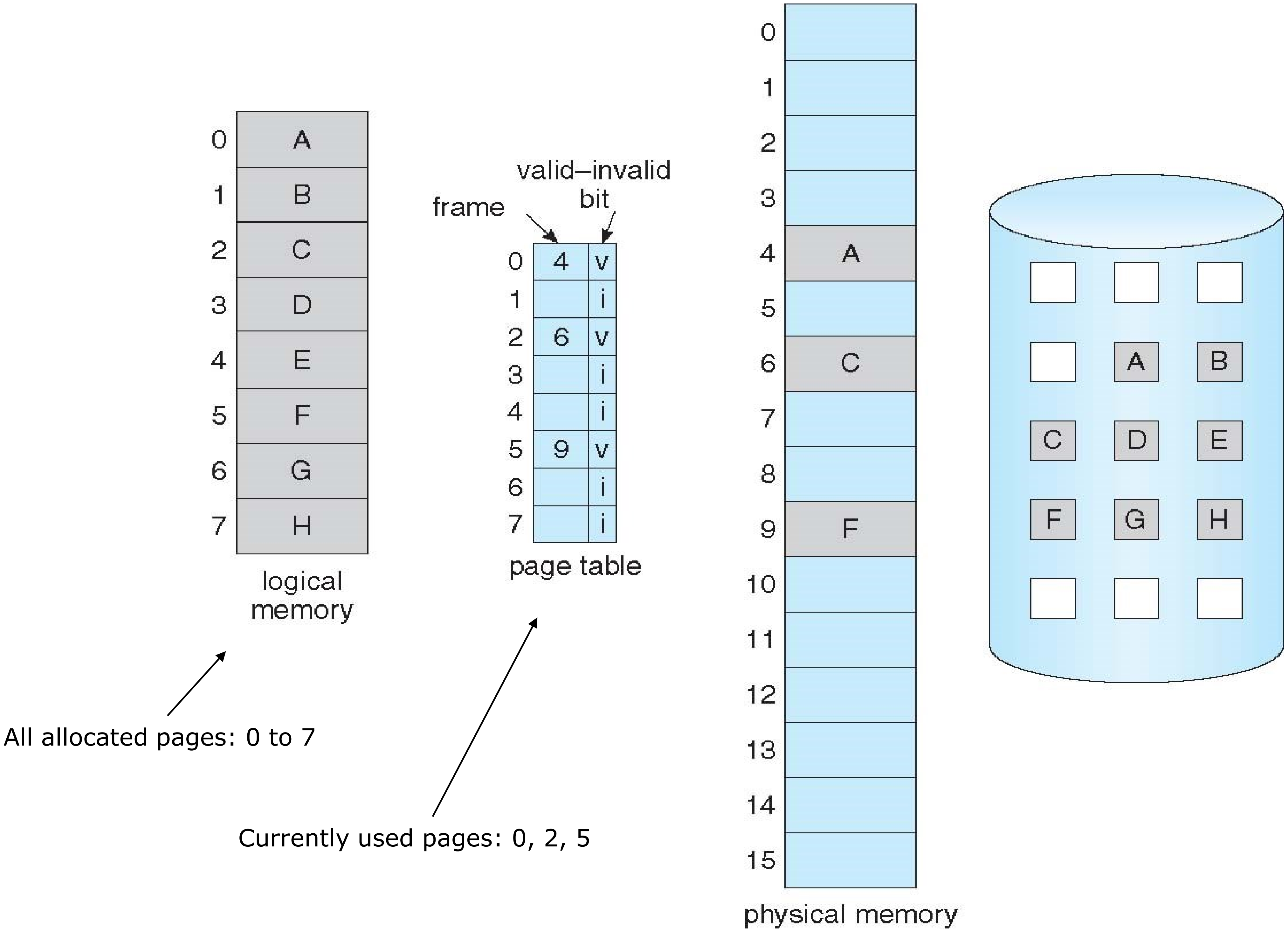
Virtual Memory



Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More processes
- CPU instruction references to anywhere in a page $\Rightarrow$ the page needs to be in physical memory
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory
  - no-free-frame $\Rightarrow$ swap

Page table that supports demand paging

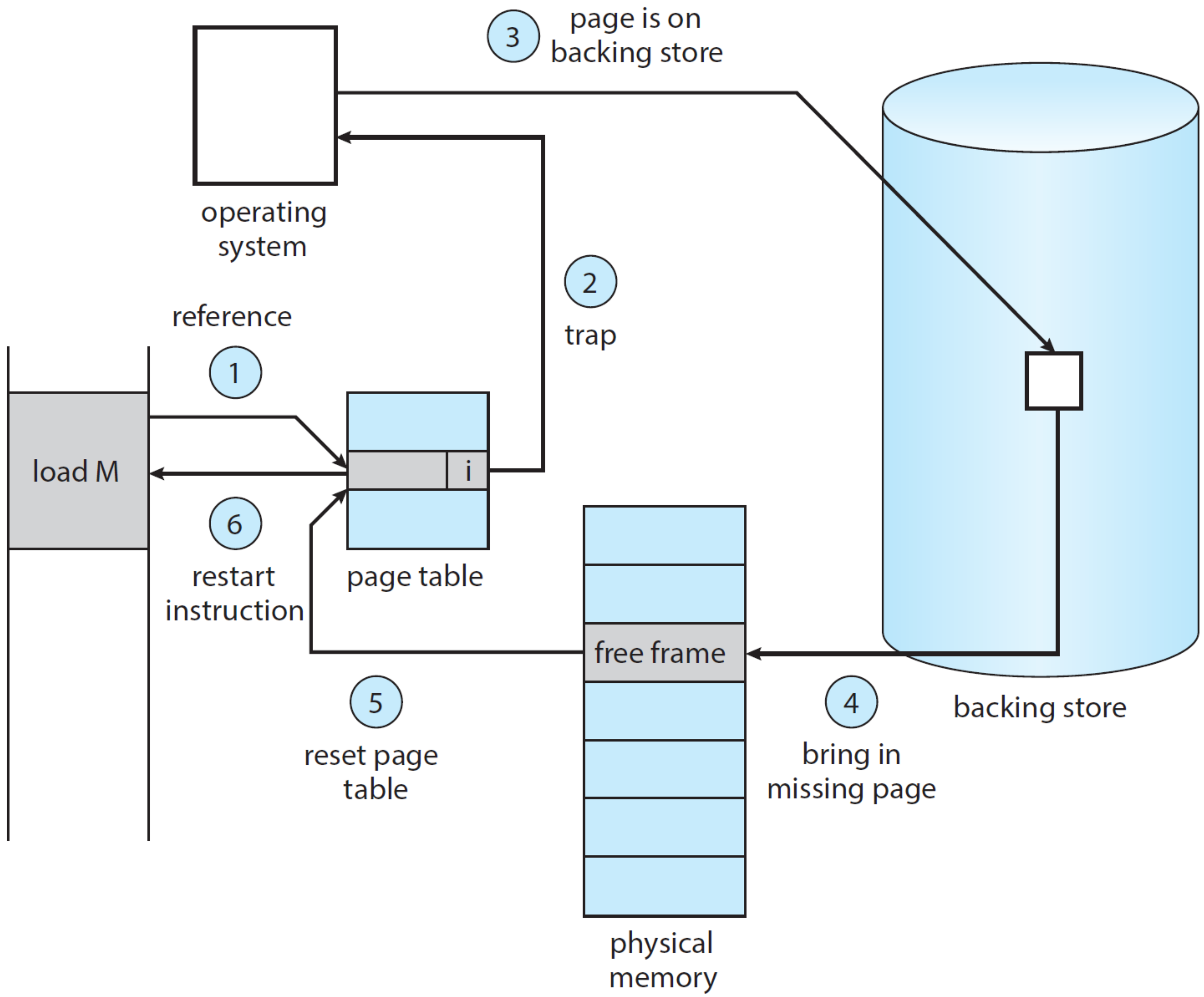


Page Fault

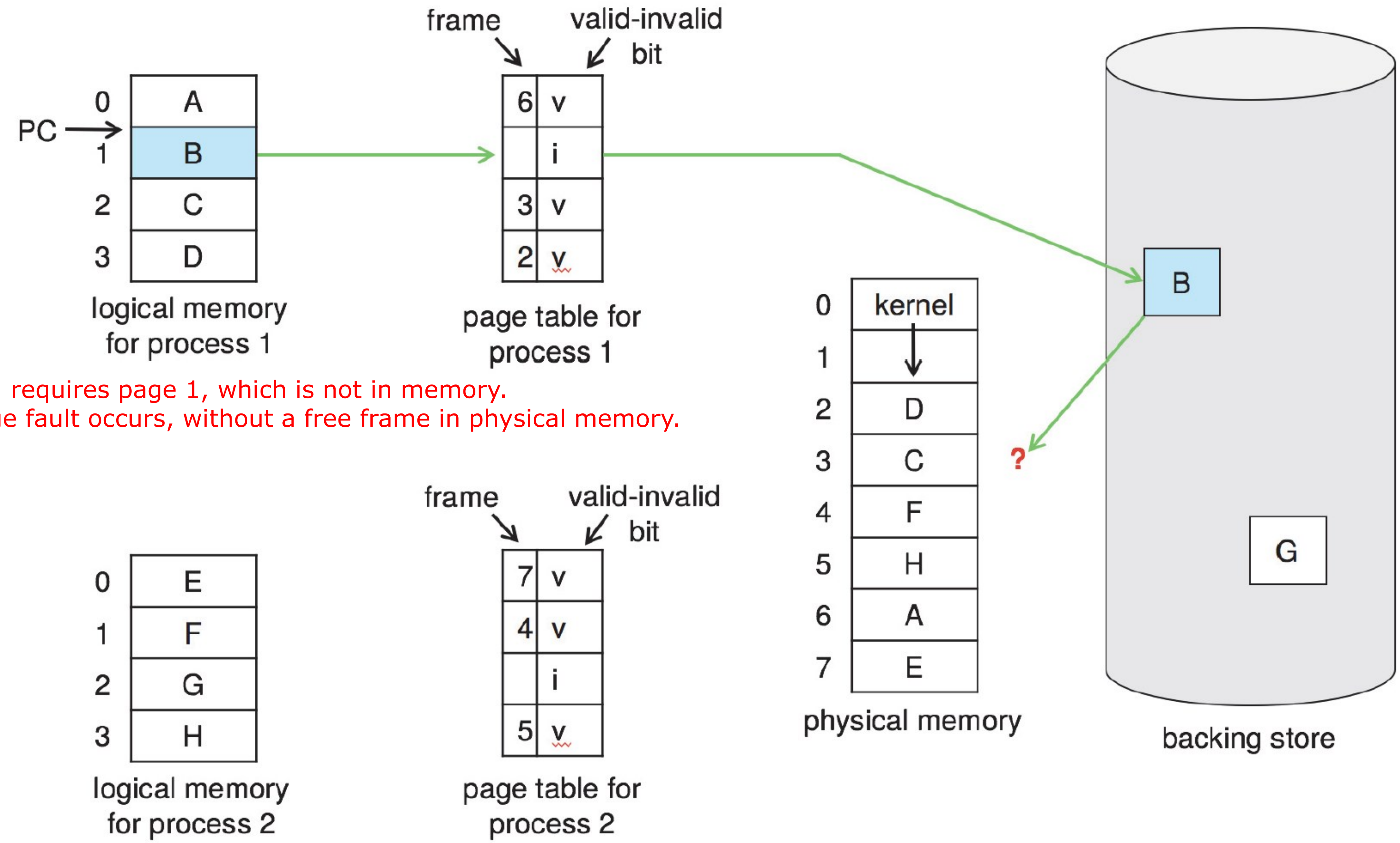
- Page fault is a mechanism to implement Demand Paging
  - Similar to cache miss
- A reference to an invalid page (e.g. first reference to that page) will cause an interrupt to operating system:
  - page fault interrupt (generated by MMU)
  - page fault handler (code in OS kernel)

1. Operating system looks at another table to decide:
   - Invalid reference → abort
   - Just not in memory → continue
2. Get empty frame
3. Swap page into frame
4. Update tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



Need For Page Replacement



Process 1 requires page 1, which is not in memory.
So, a page fault occurs, without a free frame in physical memory.

Basic Page Replacement

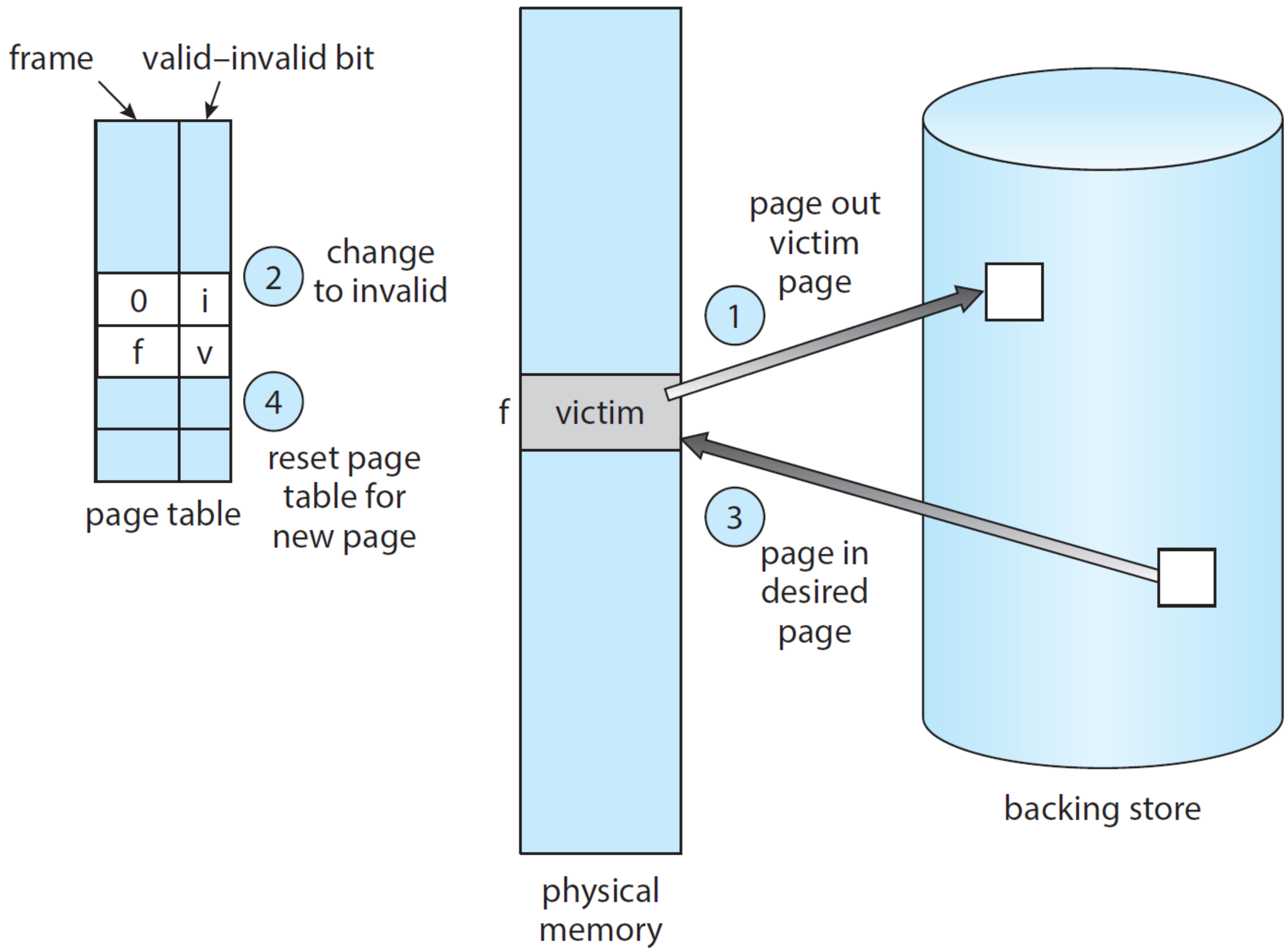
1. Find the location of the desired page on disk.

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.**
   - Write victim frame to disk if dirty.

3. Bring the desired page into the (newly) free frame; update the page and frame tables.

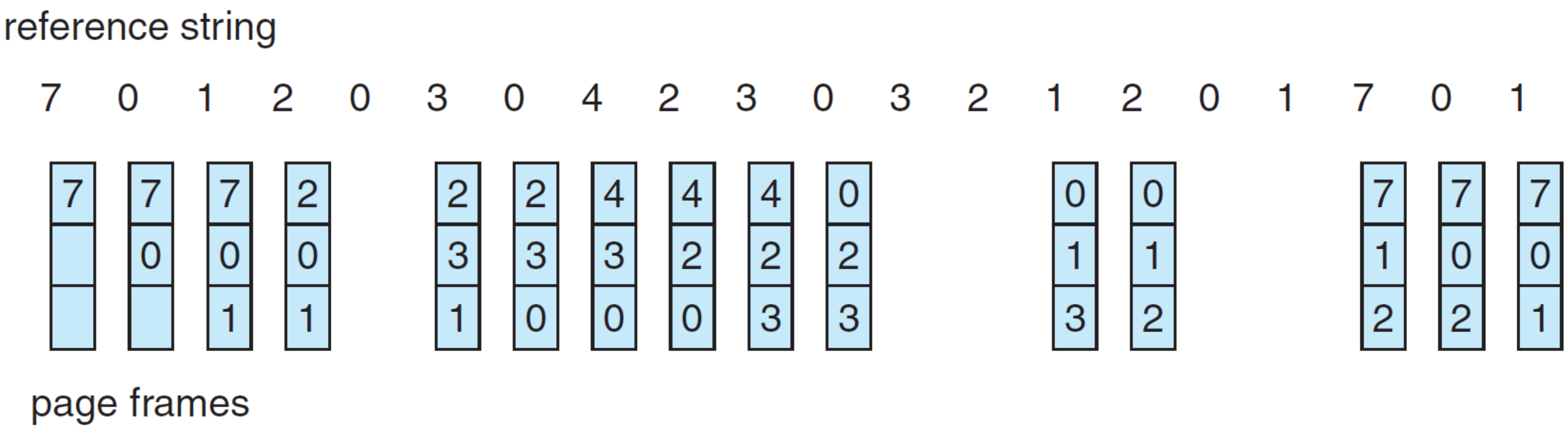
4. Continue the process by restarting the instruction that caused the trap.

Page Replacement



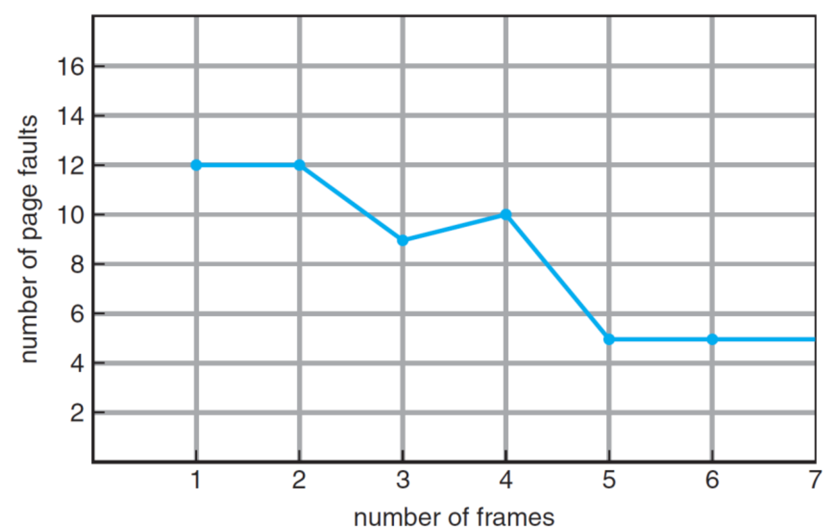
First-In-First-Out (FIFO) Algorithm

- Use a FIFO queue. When a page is brought into memory, append it into the queue.
- When a page must be replaced, choose the oldest page in memory (the first one in the queue).
- Example: Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



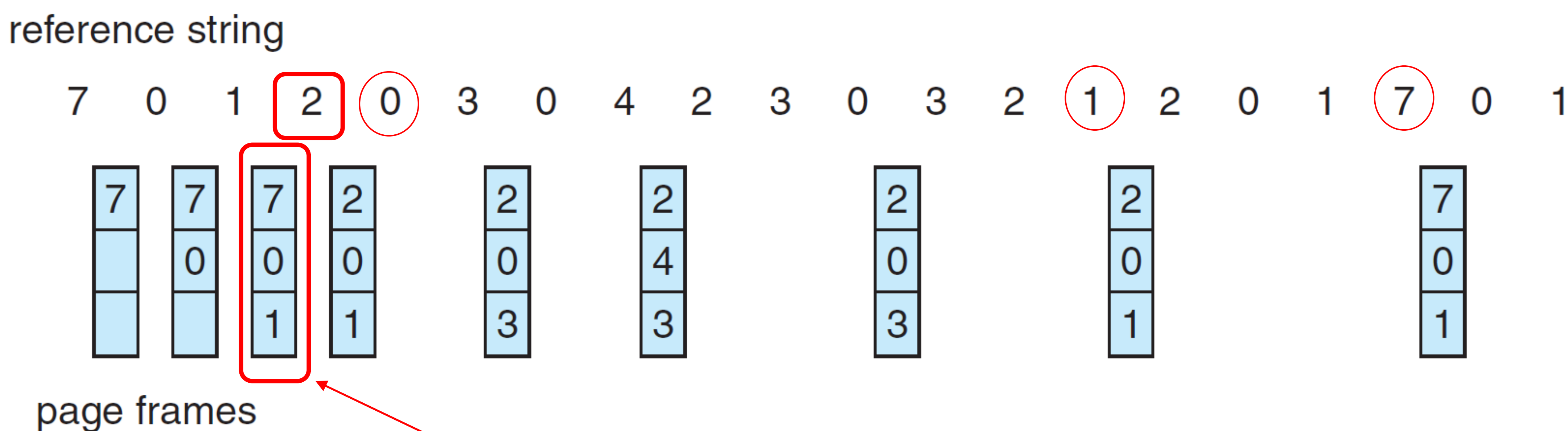
- 15 page faults

- Adding more frames can cause more page faults!
  - **Belady's Anomaly**
  - consider 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
- How do you know this?
    - Can't read the future
- Used for measuring how well your algorithm performs

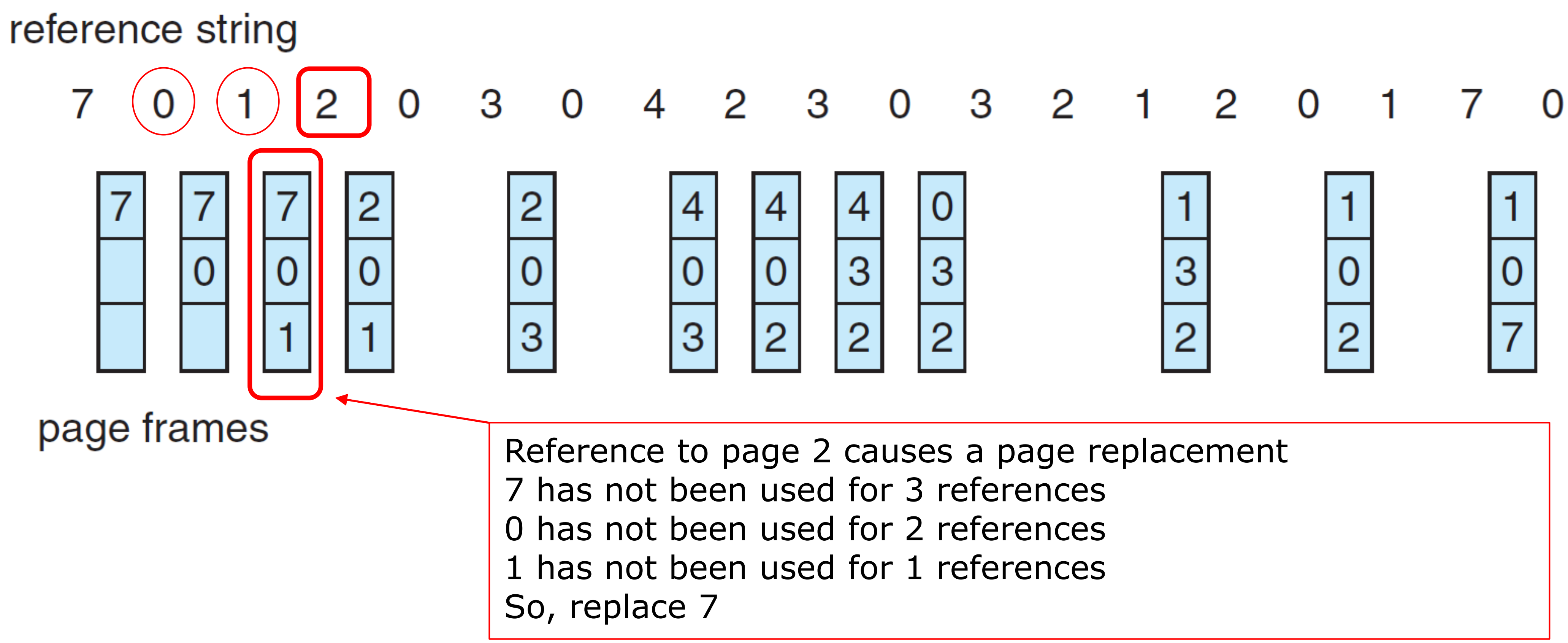


Reference to page 2 causes a page replacement
7 will be used again in 14 references
0 will be used again in 1 reference
1 will be used again in 10 reference
So, replace 7

- 9 page faults

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



- 12 faults – better than FIFO but worse than Optimal Algorithm
- Generally good algorithm and frequently used
- But how to implement?

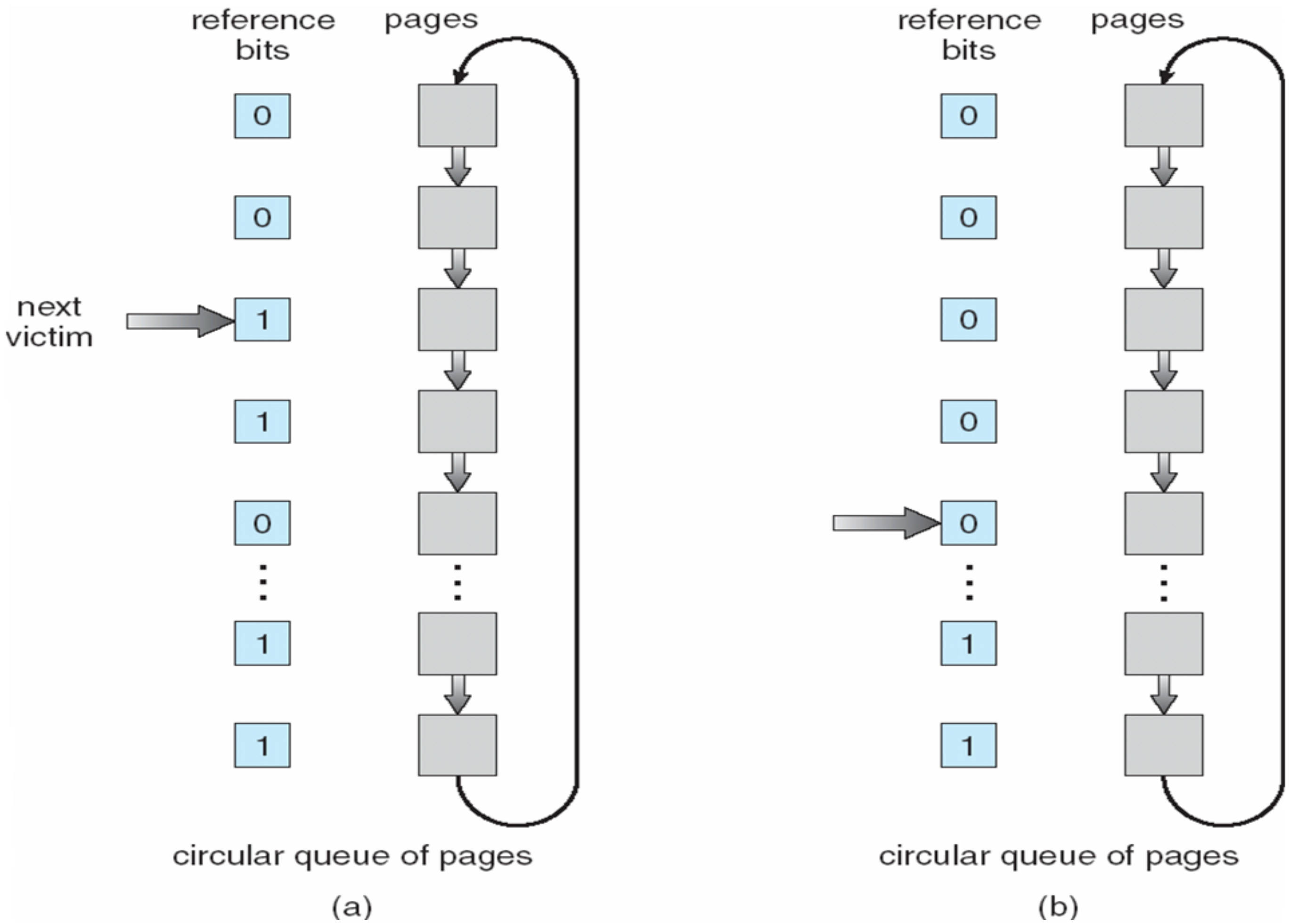
LRU Algorithm (Cont.)

- 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 and OPT are cases of **stack algorithms** that don't have Belady's Anomaly

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 -> replace it
    - reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - replace next page, subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



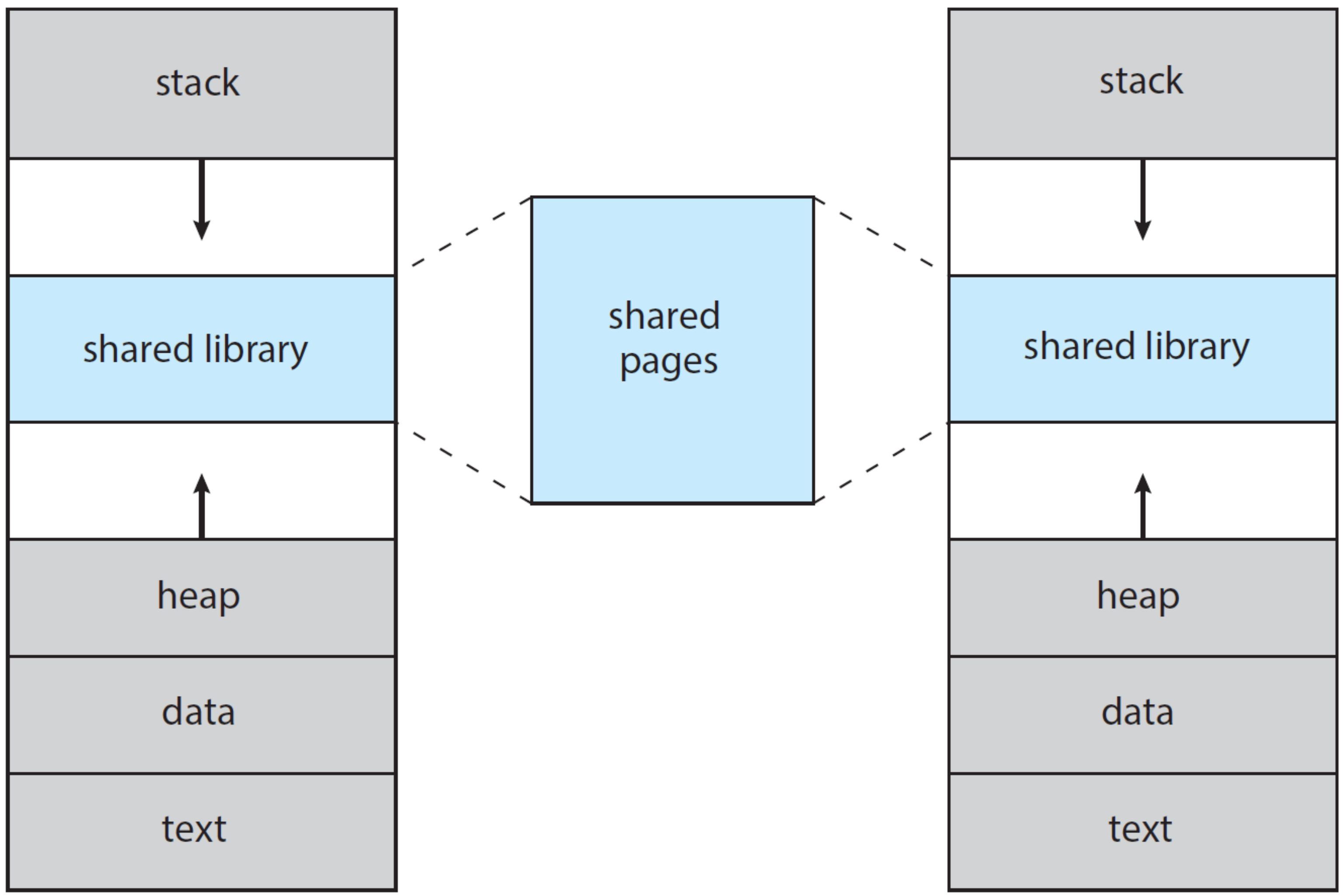
- Use a circular queue to keep pages in FIFO manner.
  - Oldest page in memory is at the head of queue and will be replaced first.
  - But if it has been used, it will be given a second chance.
  - Each page has a reference bit. Initialized to zero and set to one when accessed. Also reset to zero periodically.
- If page to be replaced has
  - Reference bit = 0 → replace it
  - reference bit = 1 then:
    - set reference bit 0, leave page in memory
    - replace next page, subject to same rules

Shared Pages

- Virtual memory allows pages to be shared among processes
- Logical memory pages of multiple processes are mapped to the same physical memory frames.
- This technique enables many capabilities:
    - Shared libraries
    - Memory-Mapped files
    - Shared-memory communication

Shared Library Using Virtual Memory

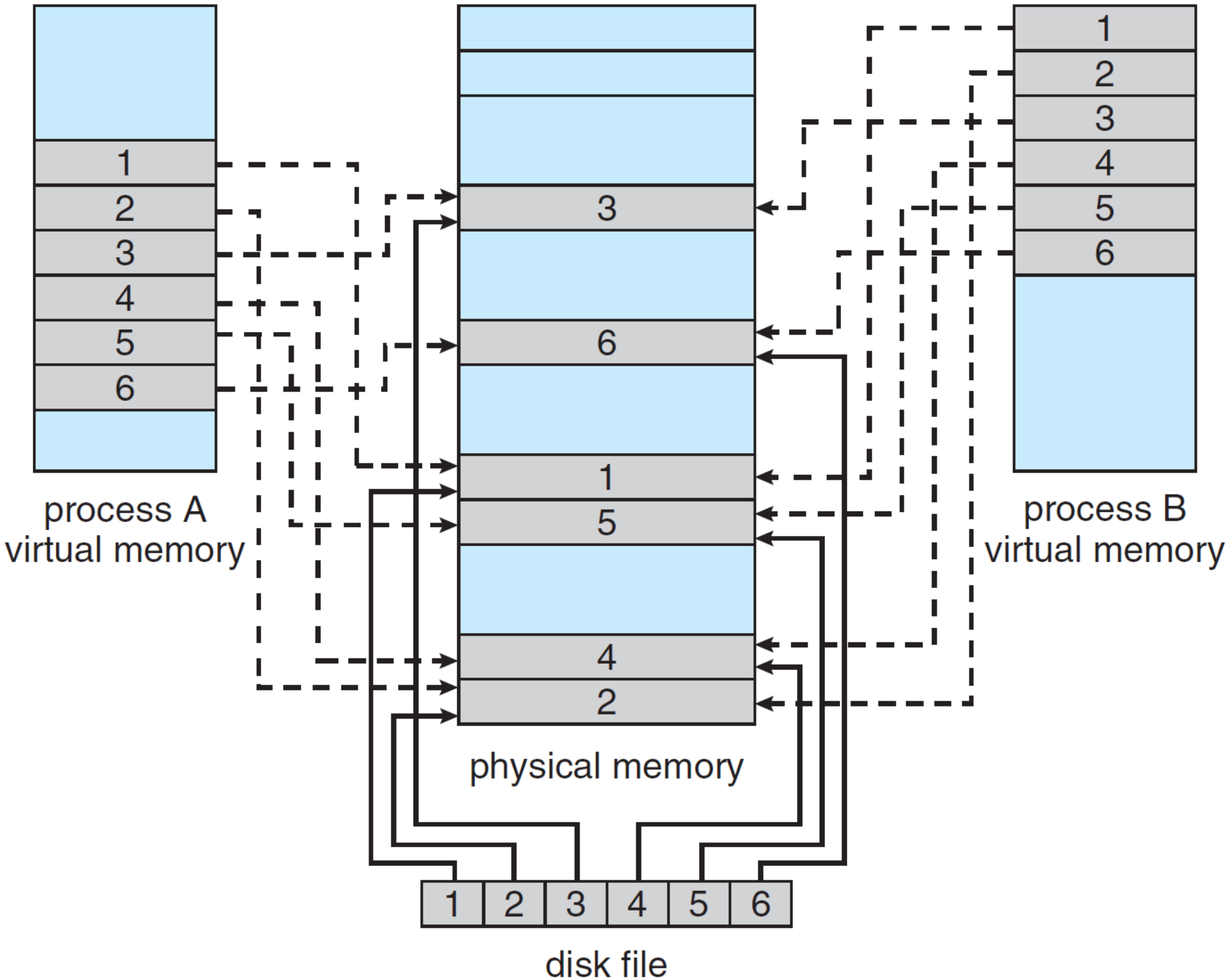
- Code libraries can be shared by several processes.
- Each process considers the library as part of its virtual address space.
- The actual pages where the libraries reside are shared.



Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory
- A file is initially read using demand paging
  - A page-sized portion of the file is read from the file system into a physical page
  - Subsequent reads/writes to/from the file are treated as ordinary memory accesses
- Simplifies and speeds file access by driving file I/O through memory rather than `read()` and `write()` system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared
- But when does written data make it to disk?
  - Periodically and / or at file `close()` time

Memory Mapped Files



Memory-Mapped Shared Memory

- Processes can communicate through shared memory.
- A process can create a region of memory that it share with another process.
- Processes sharing this region consider it part of their virtual address space, yet the actual physical pages of memory are shared.