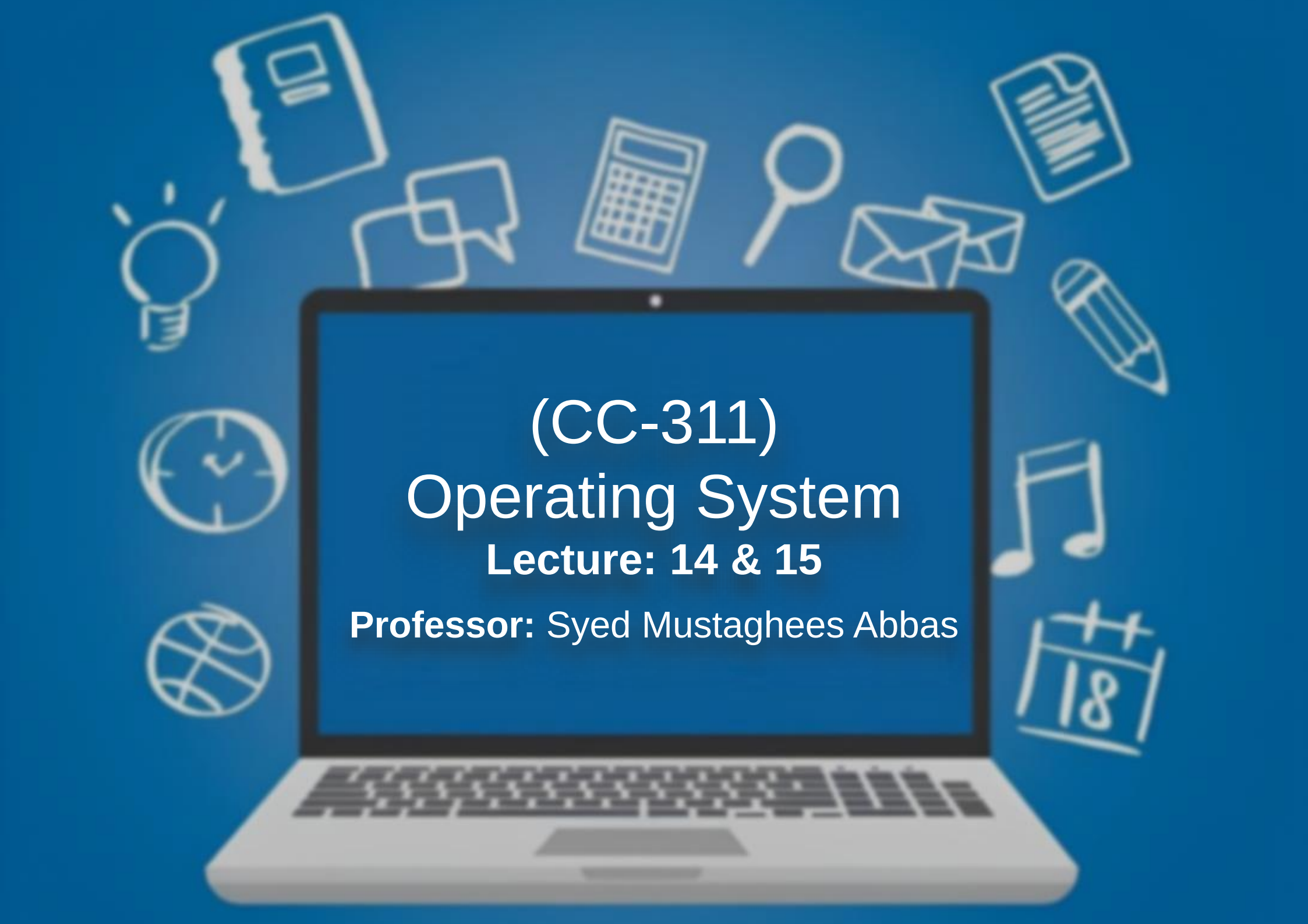

(CC-311)
Operating System
Lecture: 14 & 15
Professor: Syed Mustaghees Abbas

Virtual Memory

Virtual Memory

- Virtual memory – separation of user logical memory from physical memory.

- Only part of the program needs to be in memory for execution.

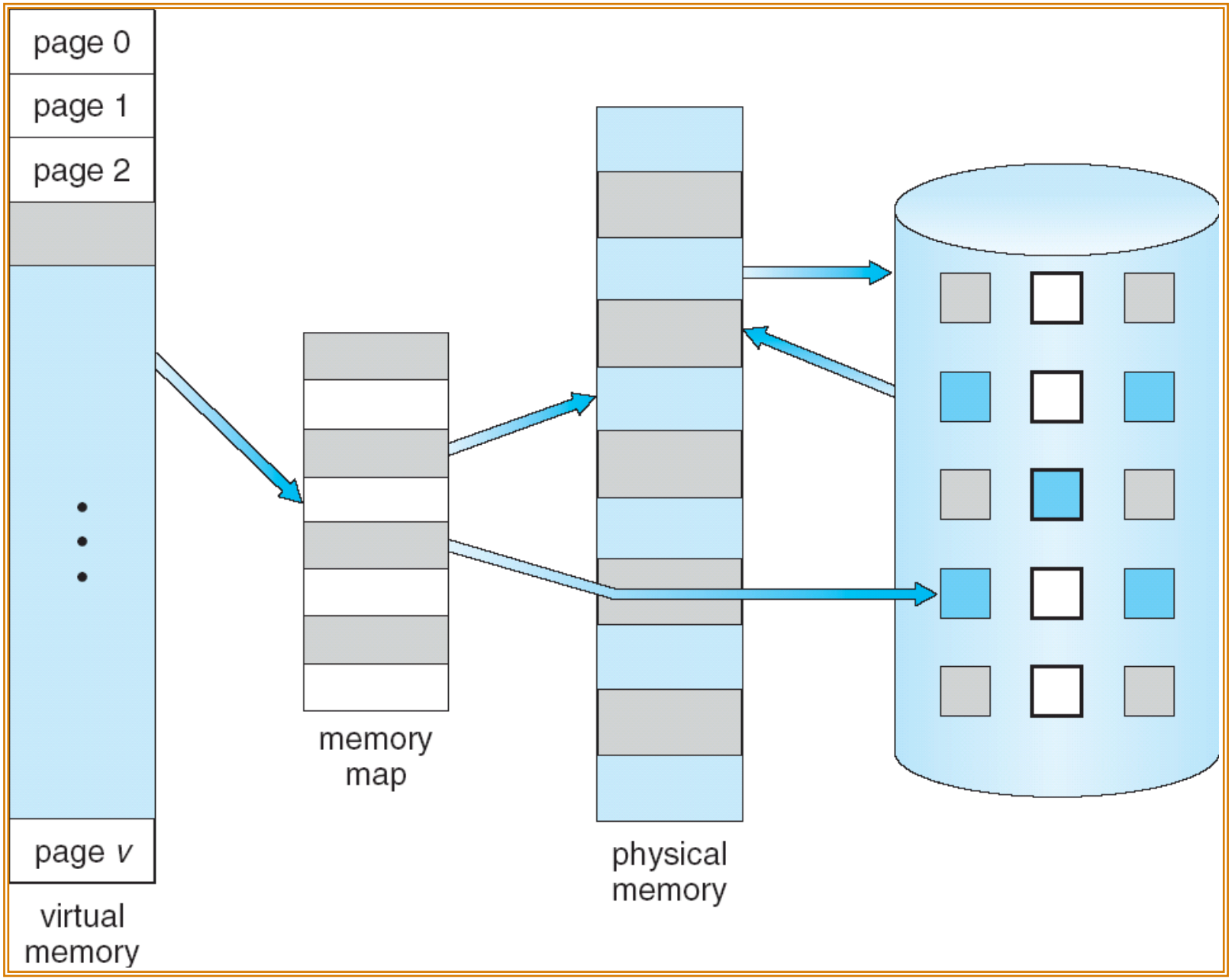
- Logical address space can therefore be much larger than physical address space. (*Virtual address space* of a process refers to the logical (or virtual) view of how a process is stored in memory.)

- Allows address spaces to be shared by several processes.

- Allows for more efficient process creation.

- Virtual memory can be implemented via:
  1. Demand paging
  2. Demand segmentation

Virtual Memory That is Larger Than Physical Memory



Demand paging

- Consider how an executable program might be loaded from disk into memory.

- One option is to load the entire program in physical memory at program execution time. However, a problem with this approach, is that we may not initially need the entire program in memory.

- Alternatively we only load pages as they are needed. This technique is known as *demand paging* and is commonly used in virtual memory systems.

- With demand-paged virtual memory, pages are only loaded when they are demanded during program execution.

Demand paging(cntd)

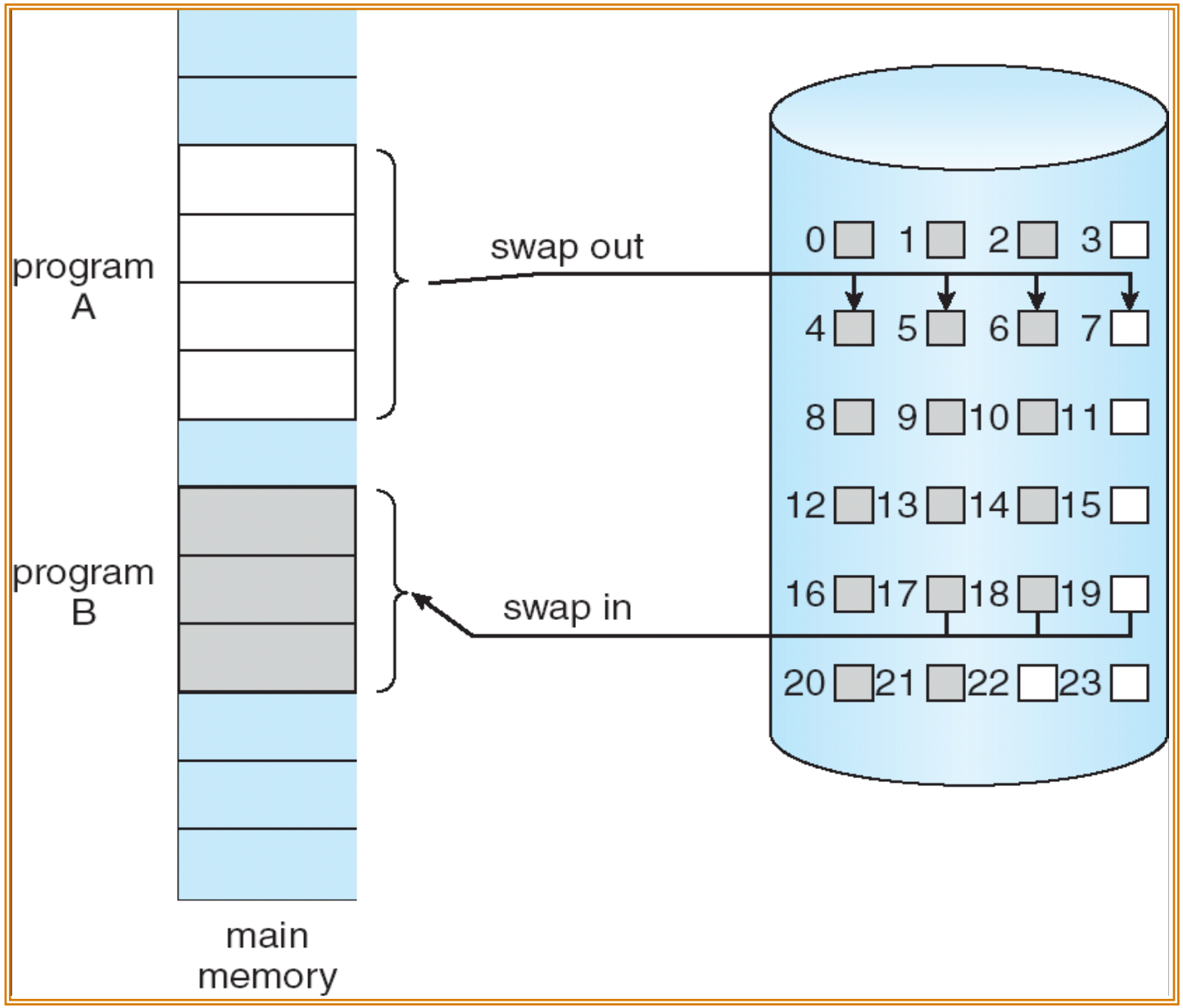
- A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk).

- When we want to execute a process, we swap it into memory.

- For swapping a **lazy swapper** is used. A lazy swapper never swaps a page into memory unless that page will be needed.

- A swapper manipulates entire processes, whereas a **pager** is concerned with the individual pages of a process. We thus use pager, rather than swapper, in connection with demand paging.

Transfer of a Paged Memory to Contiguous Disk Space



Demand paging(cntd)

- For demand paging, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk.

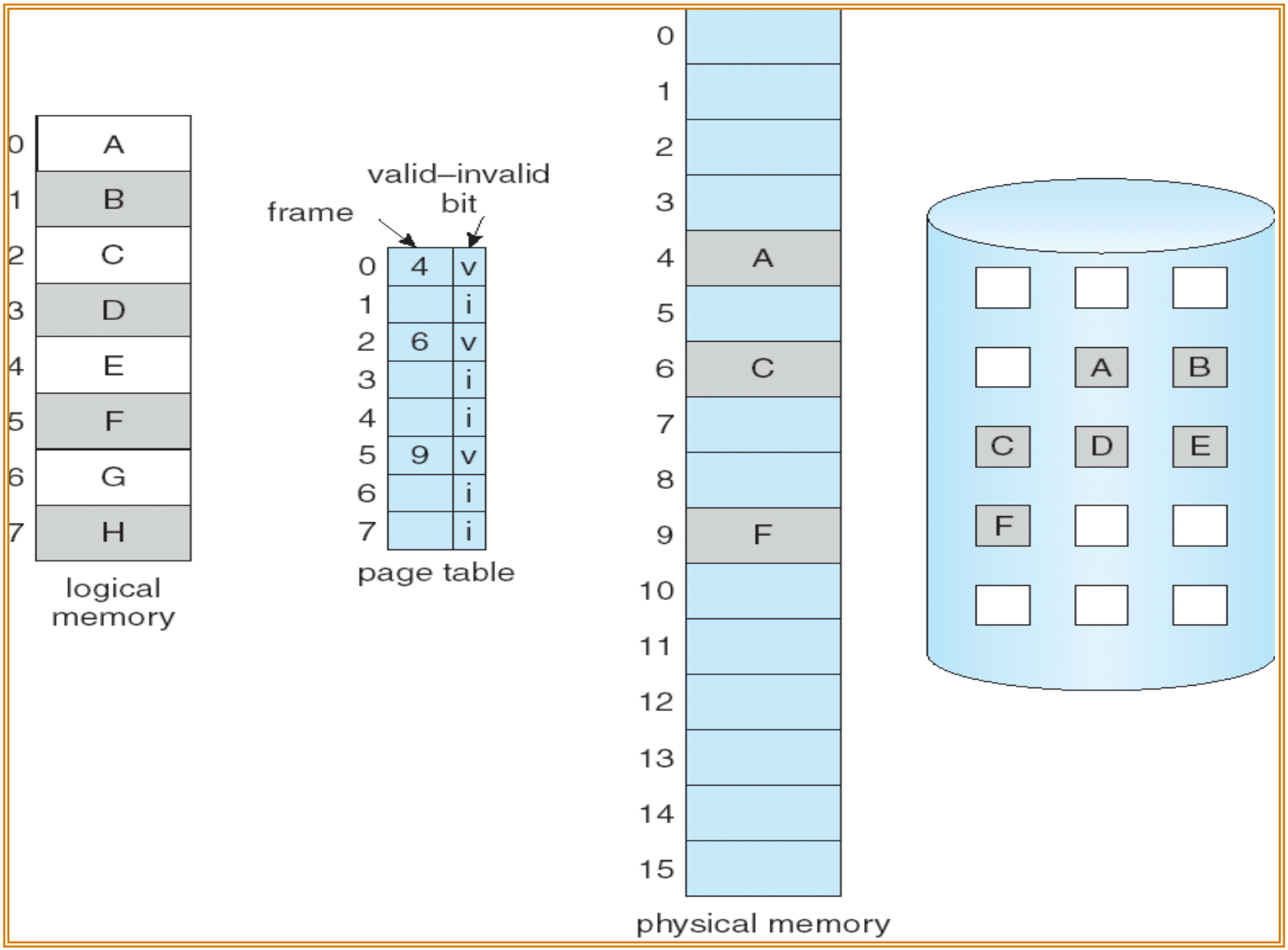
- The ***valid-invalid bit*** scheme can be used for this purpose.

- Now when this bit is set to "valid" the associated page is both legal and in memory.

- If the bit is set to "invalid," the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk.

- If we guess right and page in all and only those pages that are actually needed, the process will run exactly as though we had brought in all pages.

Page Table When Some Pages Are Not in Main Memory



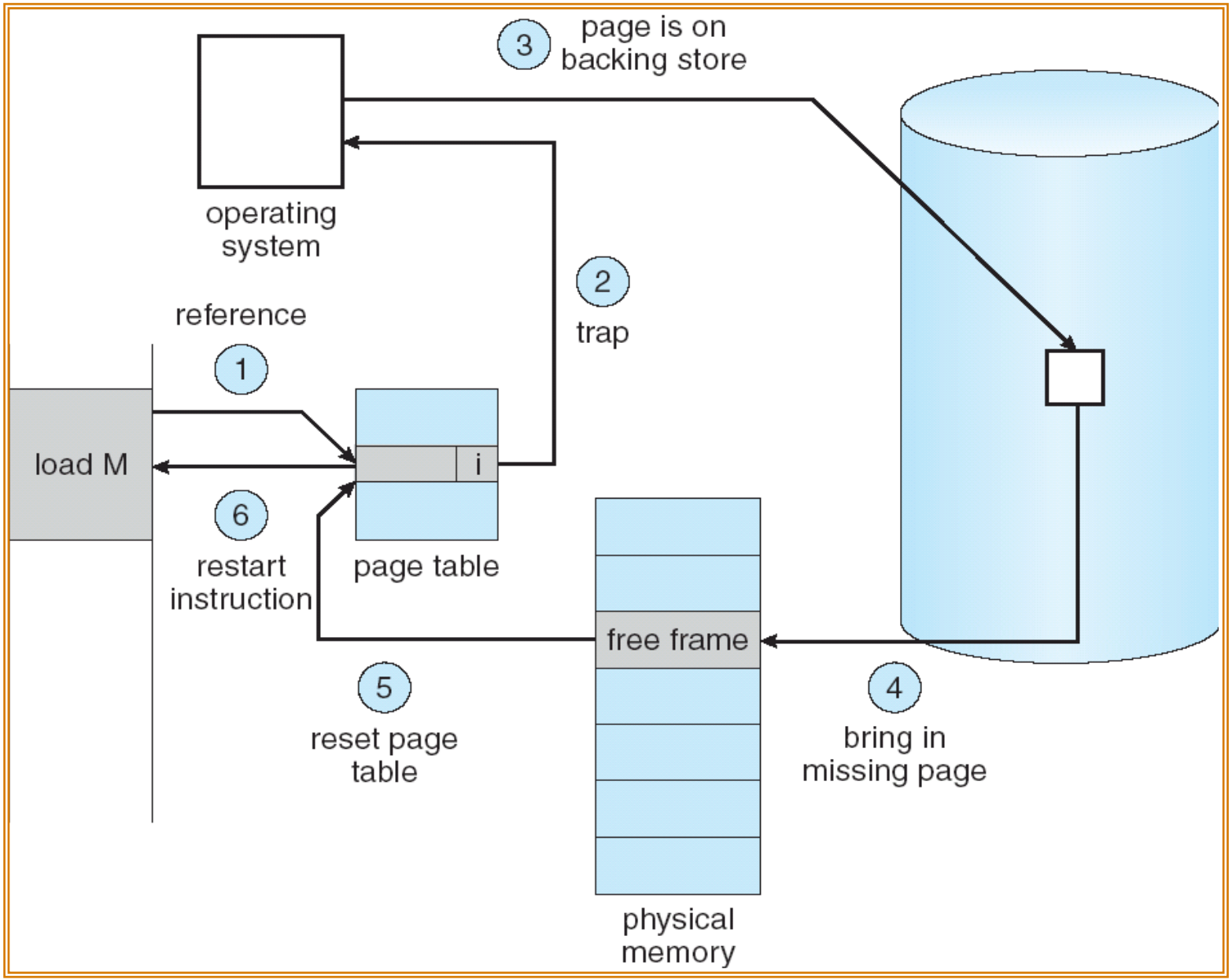
Page Fault

- If the process tries to access a page that was not brought into memory causes a page-fault trap.

- The procedure for handling this page fault is straightforward

  - 1. We check whether the reference was a valid or an invalid memory access.
  - 2. If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
  - 3. We find a free frame (by taking one from the free-frame list, for example).
  - 4. We schedule a disk operation to read the desired page into the newly allocated frame.
  - 5. When the disk read is complete, we modify the page table to indicate that the page is now in memory.
  - 6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

Steps in Handling a Page Fault



Pure Demand Paging

- In pure demand paging we start executing a process with no pages in memory.
- When the first instruction is executed, the process immediately faults for the page.
- After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory.
- This scheme is pure demand paging: Never bring a page into memory until it is required.

Copy-on-Write (COW)

- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory.

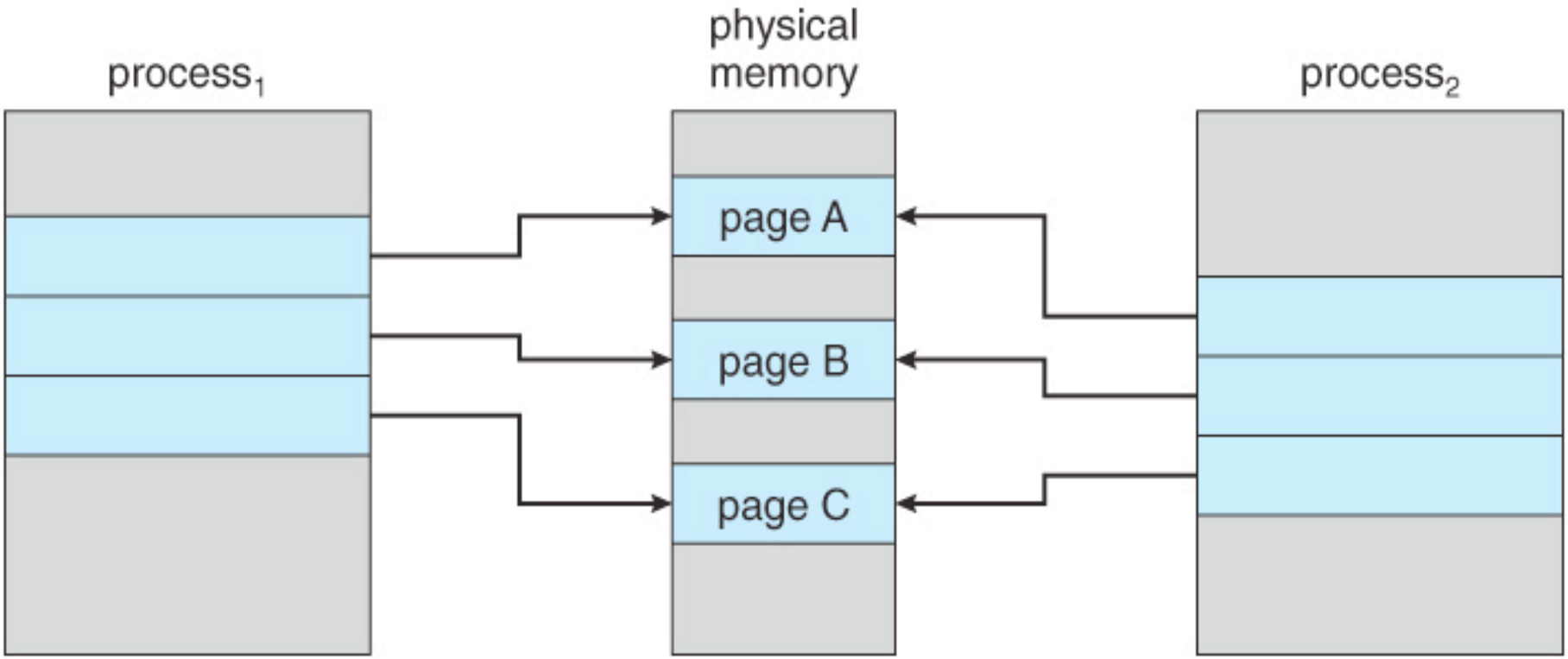
- Recall that the fork() system call creates a child process as a duplicate of its parent. Duplicating the pages belonging to the parent.

- However most child processes invoke the exec() system call immediately after creation, thus copying of the parent's address space may be unnecessary.

- Alternatively, we can use copy-on-write, which works by allowing the parent and child processes initially to share the same pages.

- These shared pages are marked as copy-on-write pages, meaning that if either process writes to a shared page, a copy of the shared page is created.

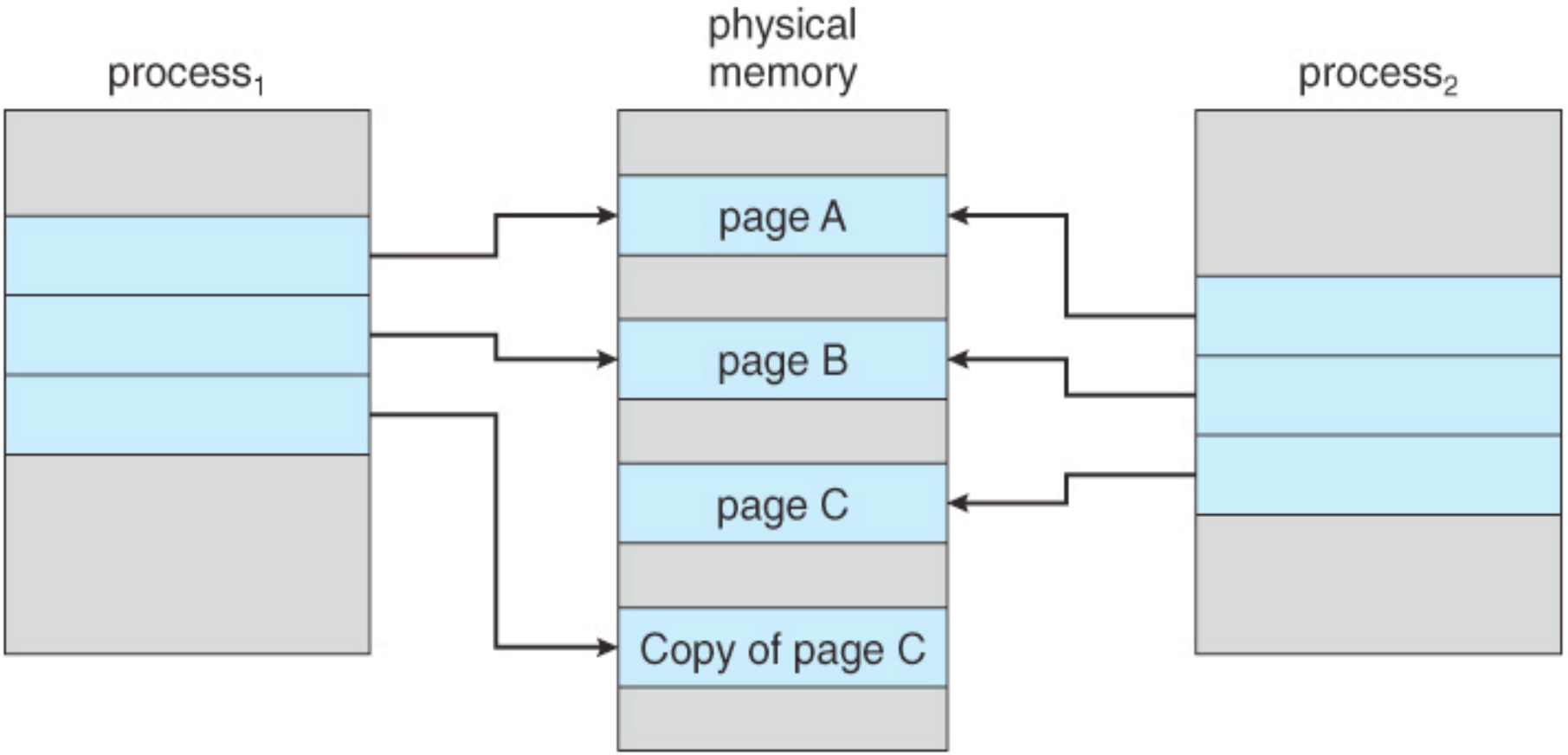
Copy-on-Write (COW)



Copy-on-Write (cntd)

- For example, assume that the child process attempts to modify a page, with the pages set to be copy-on-write.
- The operating system will then create a copy of this page, mapping it to the address space of the child process.
- The child process will then modify its copied page and not the page belonging to the parent process. All unmodified pages can be shared by the parent and child processes.
- COW allows more efficient process creation as only modified pages are copied

After process 1 modifies page C.



Copy-on-Write (cntd)

- Only pages that can be modified need be marked as copy-on-write. Pages that cannot be modified (pages containing executable code) can be shared by the parent and child.

- Many operating systems provide a pool of free pages for such requests.

- Operating systems typically allocate these pages using a technique known as **zero-fill-on-demand**.

- Zero-fill-on-demand pages have been zeroed-out before being allocated, thus erasing the previous contents.

Page Replacement

- If a process of ten pages actually uses only half of them, then demand paging saves the I/O necessary to load the five pages that are never used.
- We could also increase our degree of multiprogramming by running twice as many processes.
- If we had forty frames, we could run eight processes, rather than the four that could run if each required ten frames (five of which were never used).
- But if we increase our degree of multiprogramming, we are over-allocating memory.
- It is possible, however, that each of these processes suddenly try to use all ten of its pages, resulting in a need for eighty frames when only forty are available.

Page Replacement (cntd)

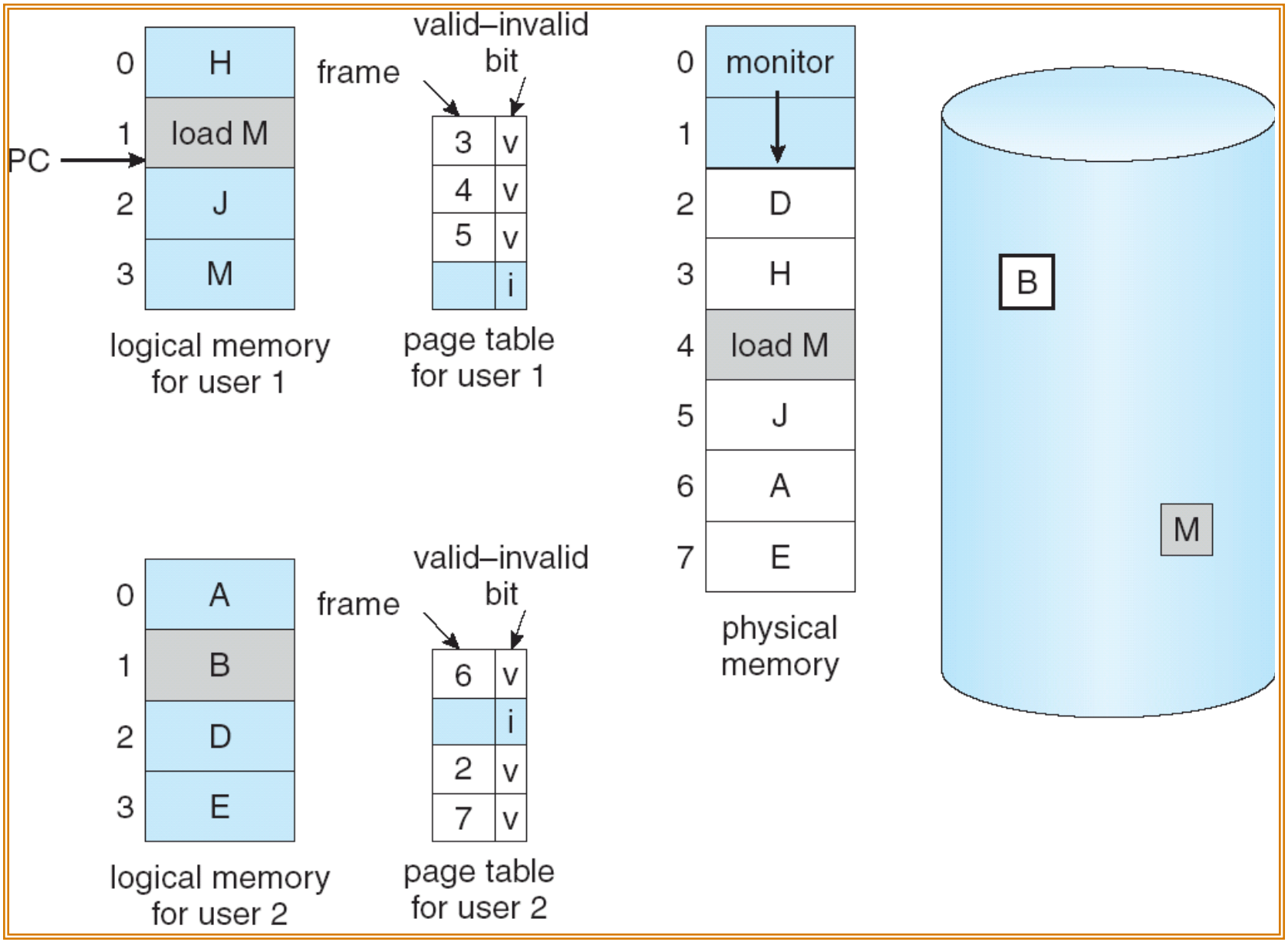
- Over-allocation of memory manifests itself as follows.

- While a user process is executing, a page fault occurs. The operating system determines where the desired page is residing on the disk but then finds that *there are no free frames*; all memory is in use.

- The operating system could terminate the user process (not the best choice).

- The operating system could instead swap out a process, freeing all its frames and reducing the level of multiprogramming. This is known as **page replacement**, and is the most common solution.

Need For Page Replacement



Basic Page Replacement

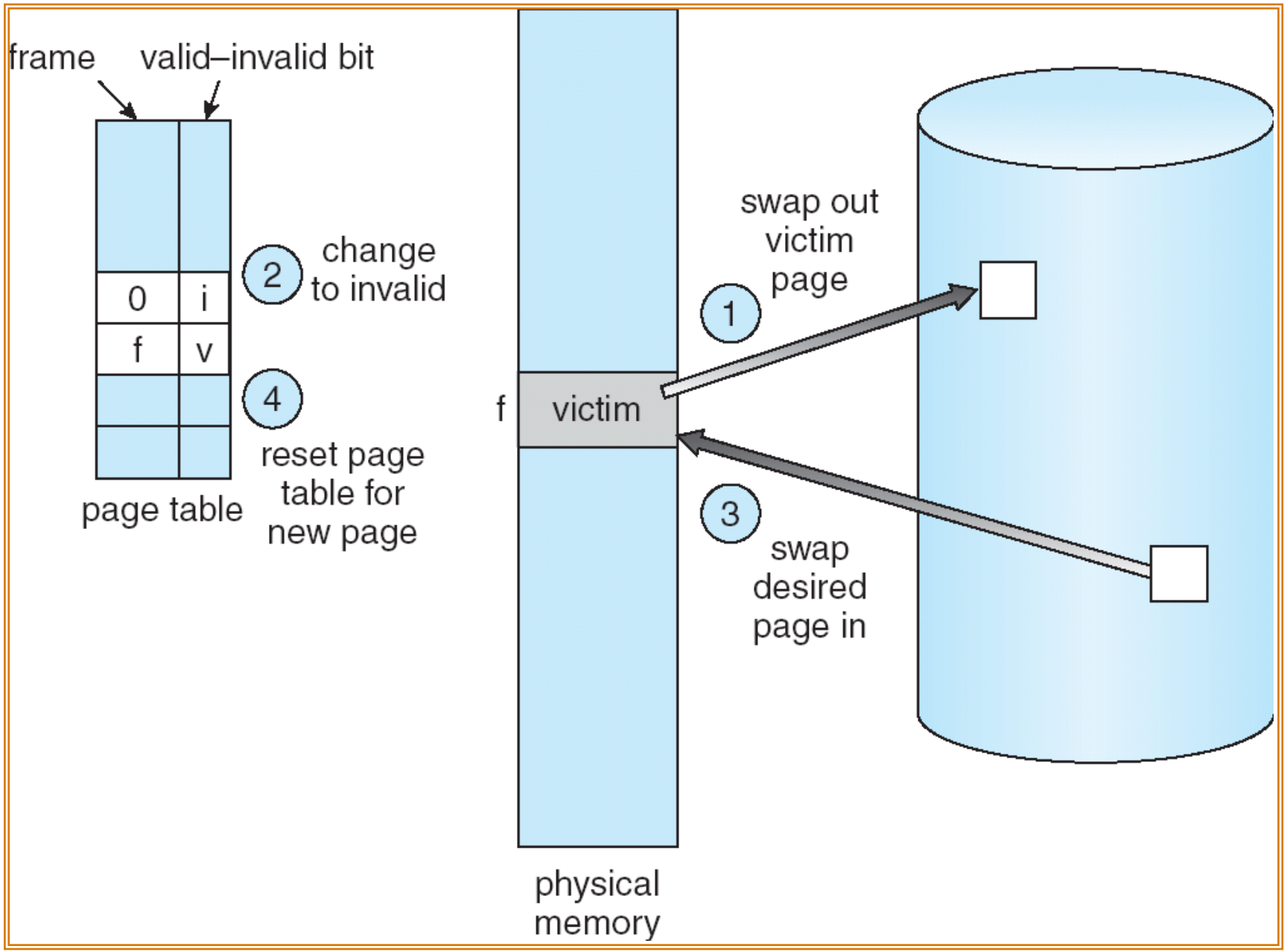
Page replacement takes the following approach.

1. Find the location of the desired page on the disk.
2. Find a free frame:
    1. If there is a free frame, use it.
    2. If there is no free frame, use a page-replacement algorithm to select a victim frame.
    3. Write the *victim frame* to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.

- Notice that, if no frames are free, two page transfers (one out and one in) are required.

- This situation effectively doubles the page-fault service time.

Page Replacement



Basic Page Replacement (cntd)

- We can reduce this overhead by using a **modify bit (or dirty bit)**.
- When this scheme is used, each page or frame has a modify bit associated with it in the hardware.
- The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.

- When we select a page for replacement,
  - If the bit is set, we know that the page has been modified since it was read in from the disk. In this case, we must write that page to the disk.
  - If the modify bit is not set, however, the page has not been modified since it was read into memory. Hence we need not write the memory page to the disk: It is already there.

- This scheme can significantly reduce the time required to service a page fault, since it reduces I/O time by one-half if the page has not been modified

Basic Page Replacement (cntd)

- There are many different page-replacement algorithms. How do we select a particular page replacement algorithm? In general, we want the one with the lowest page-fault rate.

- We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults.

- The string of memory references is called a **reference string**.

- We can generate reference strings artificially (by using a random-number generator, for example), or we can trace a given system and record the address of each memory reference.

Basic Page Replacement (cntd)

- The latter choice produces a large number of data. To reduce the number of data, we use two facts.

- We need to consider only the page number, rather than the entire address. If we have a reference to a page p, then any immediately following references to page p will never cause a page fault.

- For example, if we trace a particular process, we might record the following address sequence:

0100, 0432, 0101,0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104,0101,0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

- At 100 bytes per page, this sequence is reduced to the following reference string:

1,4,1,6,1,6,1,6,1,6,1

Basic Page Replacement (cntd)

- We next illustrate several page-replacement algorithms. In doing so, we use the reference string for a memory with **three frames**.

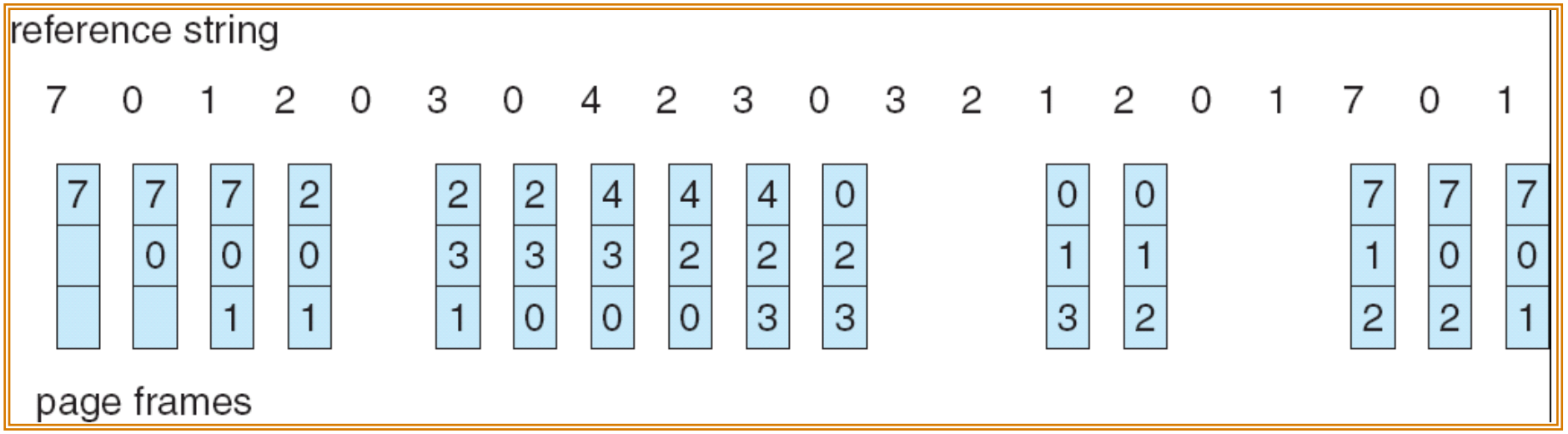
7, 0,1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2,1, 2, 0, 1, 7, 0,1

FIFO Page Replacement

- The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen.
- Although it is easy to understand and program but its performance is not always good. A bad replacement choice increases the page-fault rate and slows process execution.

1. On the one hand, the page replaced may be an initialization module that was used a long time ago and is no longer needed.
2. On the other hand, it could contain a heavily used variable that was initialized early and is in constant use. After we replace an active page with a new one, a fault occurs almost immediately to retrieve the active page.

FIFO Page Replacement



Total 15 page faults!

Belady's anomaly

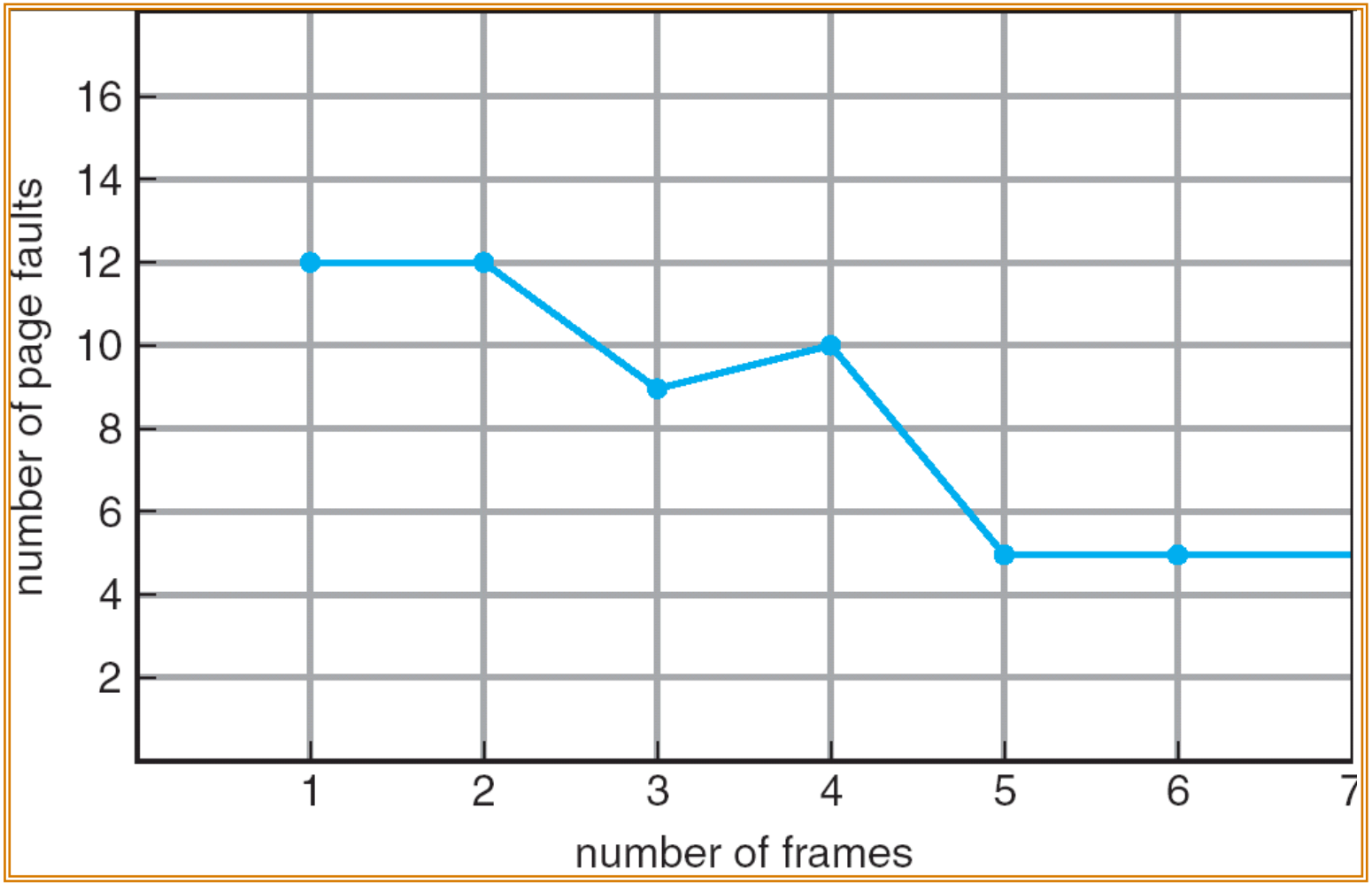
- We would expect that giving more memory to a process would improve its performance but its not true.

- Belady's anomaly: For some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.

- To illustrate the problems that are possible with a FIFO page-replacement algorithm, we consider the following reference string:

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

Belady's anomaly



Notice that the number of faults for four frames (ten) is greater than the number of faults for three frames (nine)!

Optimal Page Replacement

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.

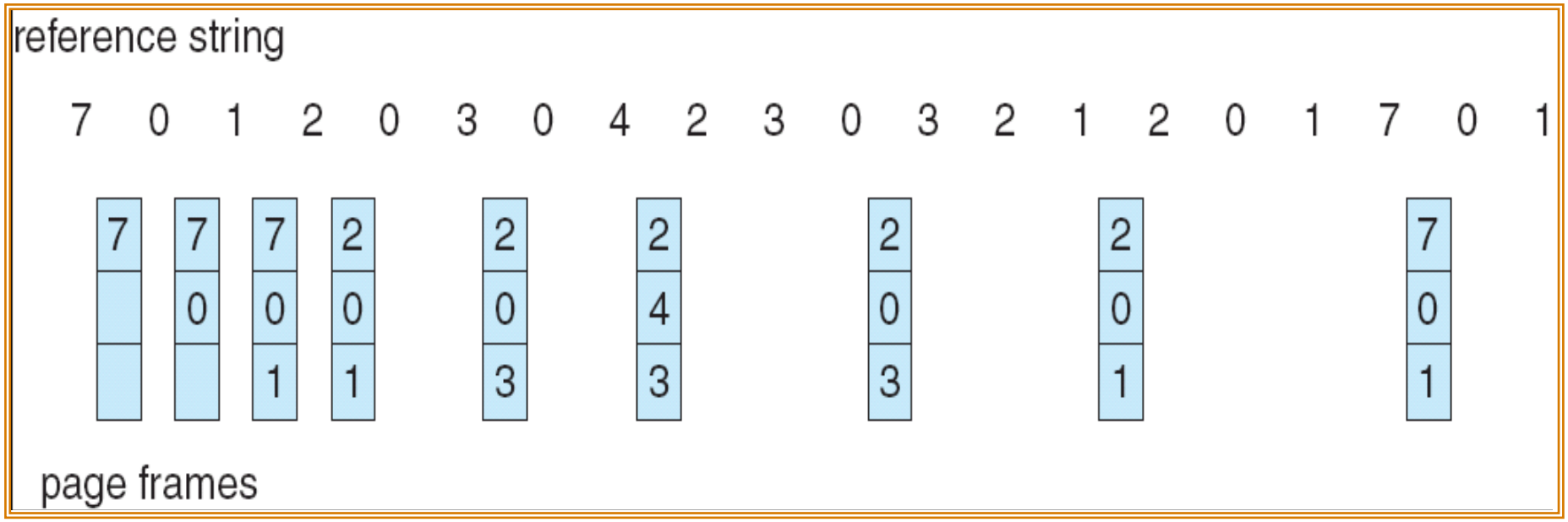
- *Replace the page that will not be used for the longest period of time.*

- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

- Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

- The optimal page replacement algorithm is used mainly for comparison studies.

Optimal Page Replacement



Total 9 page faults!

LRU Page Replacement

- If we replace the page that has not been used for the longest period of time then this approach is called least-recently-used (LRU) algorithm.

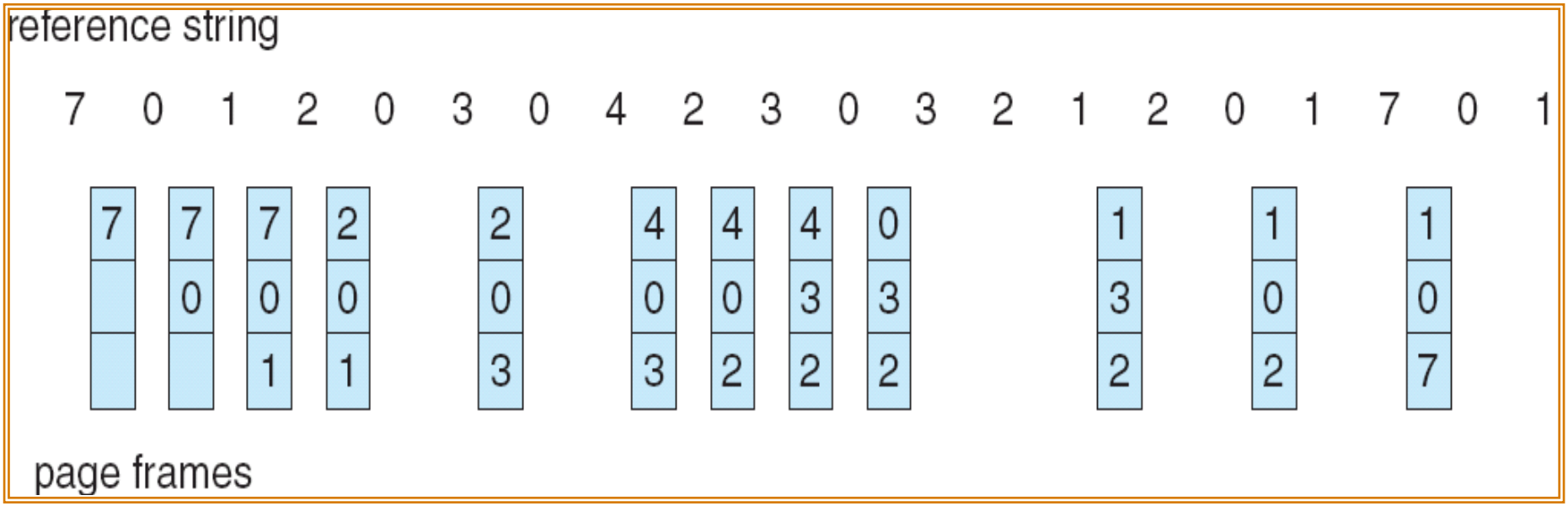
- LRU replacement associates with each page the time of that page's last use.

- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.

- We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.

- The LRU policy is often used as a page-replacement algorithm and is considered to be good. Like optimal replacement, LRU replacement does not suffer from Belady’s anomaly.

LRU Page Replacement



Total 12 page faults!

Allocation of Frames

- How do we allocate the fixed amount of free memory among the various processes?
- If we have 93 free frames and two processes, how many frames does each process get?
- The simplest case is the single-user system. Consider a single-user system with 128 KB of memory composed of pages 1 KB in size. This system has 128 frames.

- The operating system may take 35 KB, leaving 93 frames for the user process. Under pure demand paging, all 93 frames would initially be put on the free-frame list.

- When a user process started execution, it would generate a sequence of page faults. The first 93 page faults would all get free frames from the free-frame list. When the free-frame list was exhausted, a page-replacement algorithm would he used to select one of the 93 in-memory pages to be replaced with the 94th, and so on.

Allocation of Frames (cntd)

- When the process terminated, the 93 frames would once again be placed on the free-frame list.

- We can try to keep three free frames reserved on the free-frame list at all times.

- Thus, when a page fault occurs, there is a free frame available to page into. While the page swap is taking place, a replacement can be selected, which is then written to the disk as the user process continues to execute.

Minimum Number of Frames

- Although we cannot allocate more than the total number of available frames, we must also allocate at least a minimum number of frames.

- One reason for allocating at least a minimum number of frames involves performance.

- Obviously, as the number of frames allocated to each process decreases, the page-fault rate increases, slowing process execution.

- *We must have enough frames to hold all the different pages that any single instruction can reference.*

- The minimum number of frames is defined by the *computer architecture*.

Minimum Number of Frames (cntd)

- The worst-case scenario occurs in computer architectures that allow multiple levels of indirection.

- A simple load instruction could reference an indirect address that could reference an indirect and so on, until every page in virtual memory had been touched.

- To overcome this difficulty, we must place a limit on the levels of indirection (for example, limit an instruction to at most 16 levels of indirection).

- Whereas the minimum number of frames per process is defined by the architecture, the maximum number is defined by the amount of available physical memory.

Allocation Algorithms

- The easiest way to split in frames among n processes is to give everyone an equal share, m/n frames.

- For instance, if there are 93 frames and five processes, each process will get 18 frames. The leftover three frames can be used as a free-frame buffer pool. This scheme is called **equal allocation**.

- An alternative is to recognize that various processes will need differing amounts of memory. Consider a system with a 1-KB frame size. If a small student process of 10 KB and an interactive database of 127 KB are the only two processes running in a system with 62 free frames, it does not make much sense to give each process 31 frames. The student process does not need more than 10 frames, so the other 21 are, strictly speaking, wasted.

- To solve this problem, we can use **proportional allocation**, in which we allocate available memory to each process according to its size.

Allocation Algorithms (cntd)

- In both equal or proportional allocation, a high-priority process is treated the same as a low-priority process.

- By its definition, however, we may want to give the high-priority process more memory to speed its execution, to the detriment of low-priority processes.

- *One solution is to use a proportional allocation scheme wherein the ratio of frames depends on the priorities of processes or on a combination of size and priority.*

Global versus Local Allocation

- We can classify page-replacement algorithms into two broad categories: global replacement and local replacement.

- Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process.

- Local replacement requires that each process select from only its own set of allocated frames.

- In local replacement, the number of frames allocated to a process does not change. With global replacement, a process may take frames from other processes, thus increasing the number of frames allocated to it.

Global versus Local Allocation (cntd)

- One problem with a global replacement algorithm is that a process cannot control its own page-fault rate. The set of pages in memory for a process depends not only on the paging behavior of that process but also on the paging behavior of other processes.

- Local replacement might hinder a process, however, by not making available to it other, less used pages of memory.

- Global replacement generally results in greater system throughput and is therefore the more common method.

Thrashing

- If any process does not have the number of frames it needs to support pages in active use, it will quickly page-fault.

- After replacing some page. it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.

- This high paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.

- Thrashing results in severe performance problems.

- Consider the following scenario. The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system.

Thrashing (cntd)

- A global page-replacement algorithm is used; it replaces pages without regard to the process to which they belong.
- Now suppose that a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames away from other processes.
- These processes need those pages, however, and so they also fault, taking frames from other processes.
- As these faulting processes queue up for the paging device, the ready queue empties and CPU utilization decreases.
- The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result.

Thrashing (cntd)

- The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device.

- Thrashing has occurred, and system throughput plunges.

- The page fault rate increases tremendously. No work is getting done, because the processes are spending all their time paging.

- We can limit the effects of thrashing by using a local replacement algorithm.

- With local replacement, if one process starts thrashing, it cannot steal frames from another process and cause the latter to thrash as well.

- However, the problem is if the processes are thrashing, they will be in the queue for the paging device most of the time.

Thrashing (cntd)

- The average service time for a page fault will increase because of the longer average queue for the paging device.

Memory-Mapped Files

- Normally a sequential read of a file on disk using the standard system calls open(), read(), and write() requires a system call and disk access each time.

- We can use the virtual memory techniques to treat file I/O as routine memory accesses.

- This approach, known as memory mapping a file, allows a part of the virtual address space to be logically associated with the file.

- It is accomplished by mapping a disk block to a page (or pages) in memory.

- A page-sized portion of the file is read from the file system into a physical page . Subsequent reads and writes to the file are handled as routine memory accesses, which simplifies file access and eliminating the overhead of using the read() and write() system calls.

Memory-Mapped Files (cntd)

- Writes to the file mapped in memory are not necessarily immediate writes to the file on disk.
- Some systems may choose to update the physical file when the operating system periodically checks whether the page in memory has been modified.
- When the file is closed, all the memory-mapped data are written back to disk and removed from the virtual memory of the process.
- Multiple processes may be allowed to map the same file concurrently, to allow sharing of data.
- The virtual memory map of each sharing process points to the same page of physical memory—the page that holds a copy of the disk block.

Memory-Mapped Files (cntd)

- The memory-mapping system calls can also support copy-on-write functionality, allowing processes to share a file in read-only mode but to have their own copies of any data they modify.