OPERATING SYSTEMS

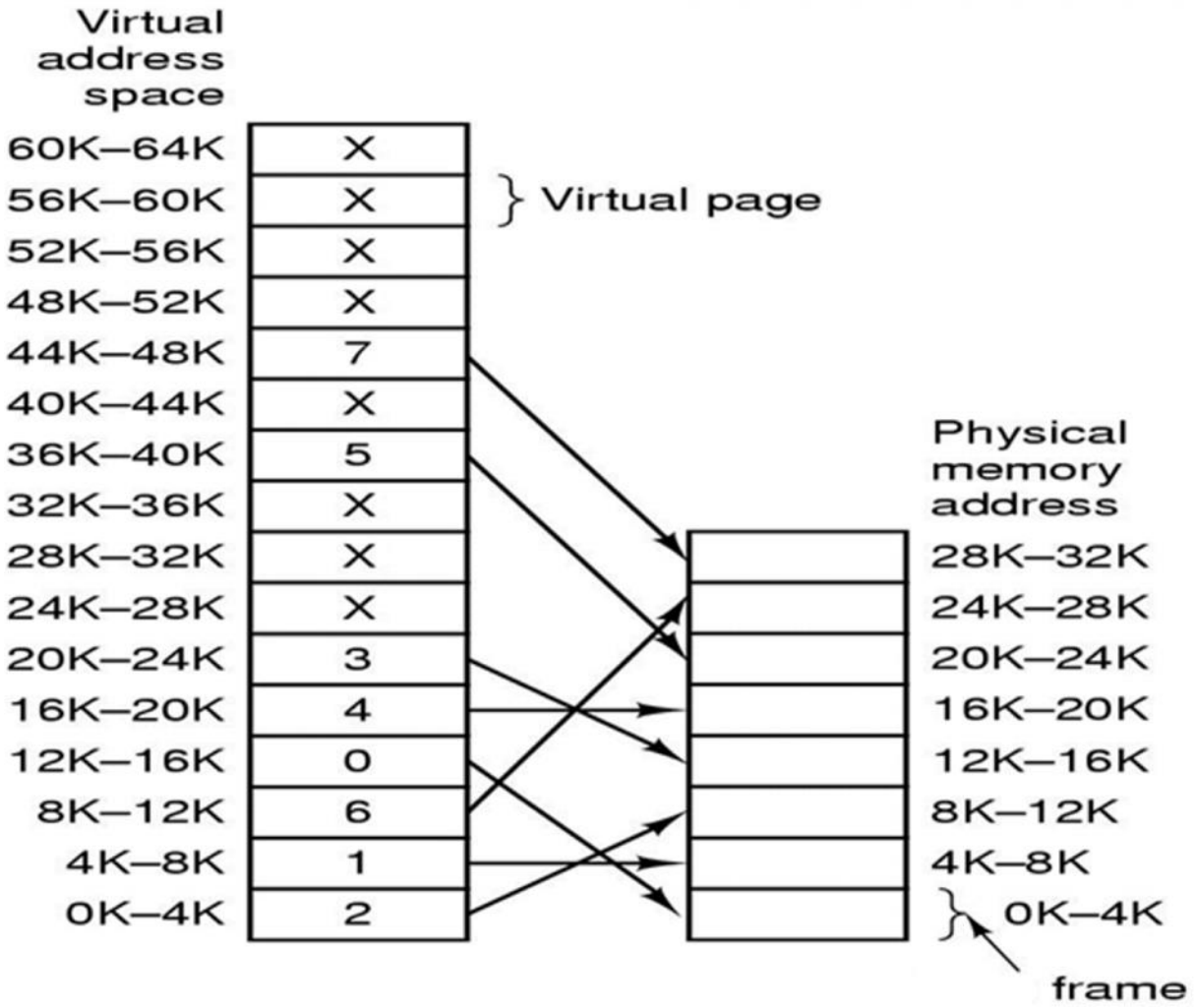
Virtual Memory Management

- Introduction
- Demand Paging
  - Introduction
  - Demand Paging mechanism
  - TLB usage & advantages

Introduction

- All pieces of a process do not need to be loaded in main memory during execution; all addresses are virtual.
- The memory referenced by a virtual address is called virtual memory:
  - It is mainly maintained on secondary memory (disk).
  - pieces are brought into main memory only when needed.

Virtual Memory Example



Advantages of Partial Loading

- More processes can be maintained in main memory
  - only load some pieces of each process
  - with more processes in main memory, more processes will be in the Ready state
- A process can now execute even if it is larger than the main memory size
  - More bits for logical addresses can be used than the bits needed for addressing the physical memory.

Support needed for Virtual Memory

- Memory management hardware must support paging and/or segmentation
- OS must be able to manage the movement of pages and/or segments between external memory and main memory
- Includes placement and replacement of pages/segments.

Process Execution

- The OS brings into main memory only few pieces of the program (including its starting point)
- Each page/segment table entry has a valid-invalid bit that is set only if the corresponding piece is in main memory.
- The resident set is the portion of the process that is in main memory at some stage.
- An interrupt (memory fault) is generated when the memory reference is on a piece that is not present in main memory.
- OS places the process in a Blocking state.
- OS issues a disk I/O Read request to bring into main memory the piece referenced to.
- Another process is dispatched to run while the disk I/O takes place.
- An interrupt is issued when disk I/O completes; this causes the OS to place the affected process back in the Ready state.

Idea of 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.
- 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 ⇒ reference to it:
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- Lazy swapper – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a pager.

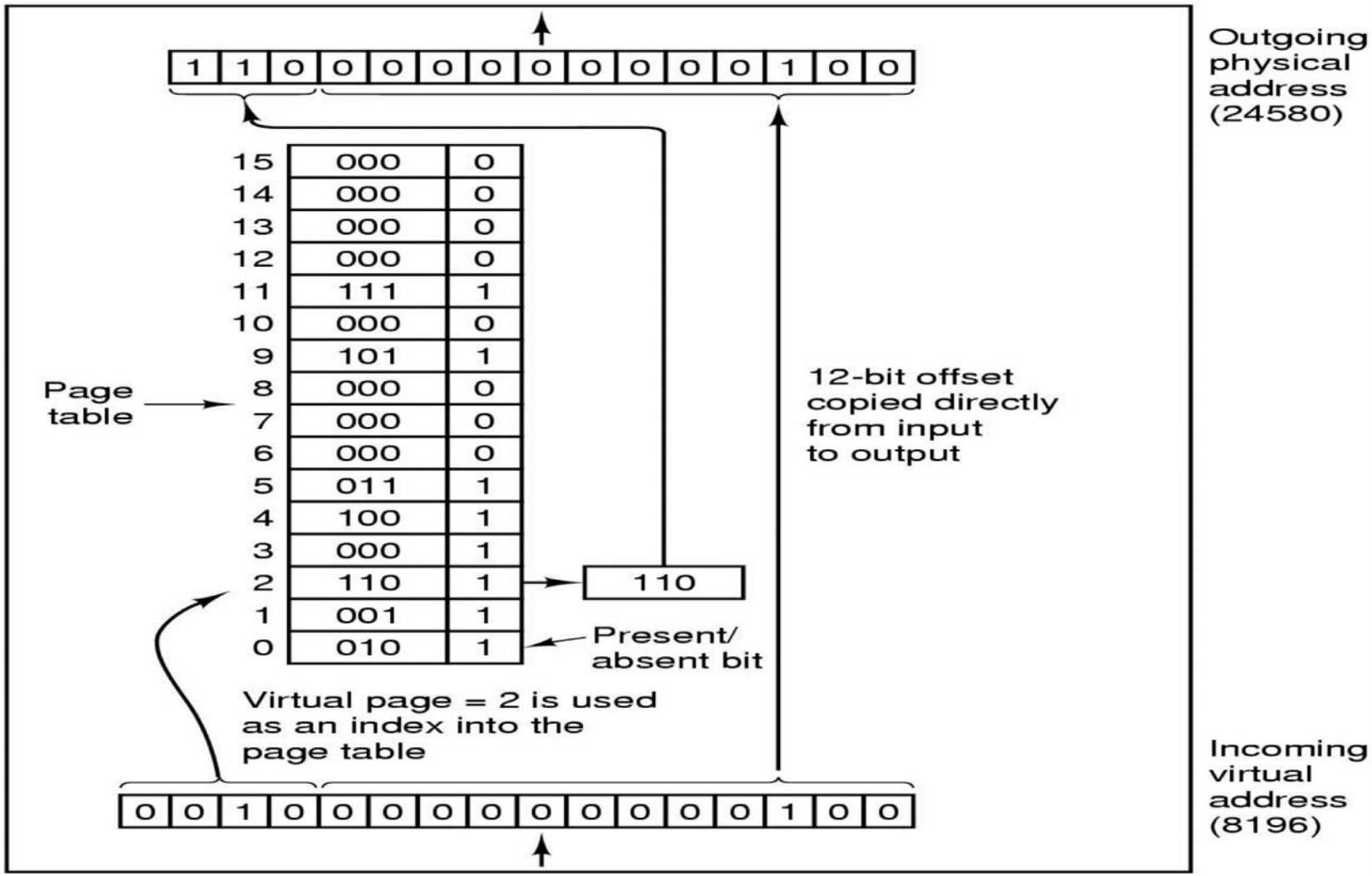
Valid-Invalid Bit

- With each page table entry, a valid–invalid (present-absent) bit is associated
  (**v** ⇒ in-memory, **i** ⇒ not-in-memory).
- Initially valid–invalid bit is set to **i** on all entries.
- Example of a page table:

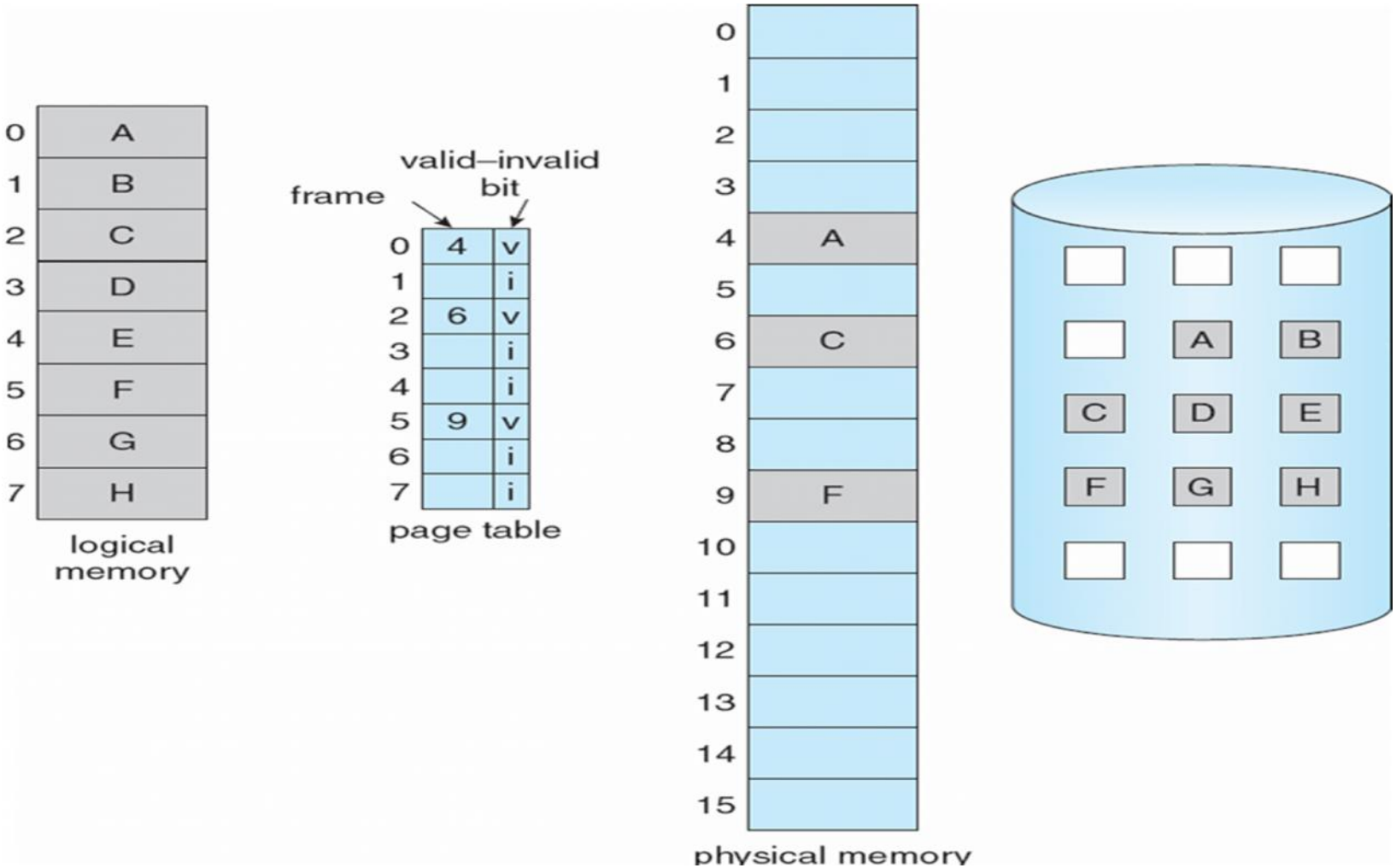
Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

- During address translation, if valid–invalid bit in page table entry is **i** ⇒ page fault.

Virtual Memory Mapping Example

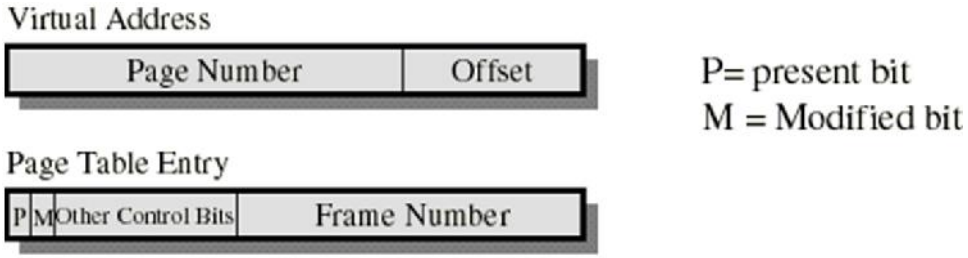


Page Table when some Pages are not in Main Memory



Demand Paging properties

- Typically, each process has its own page table.

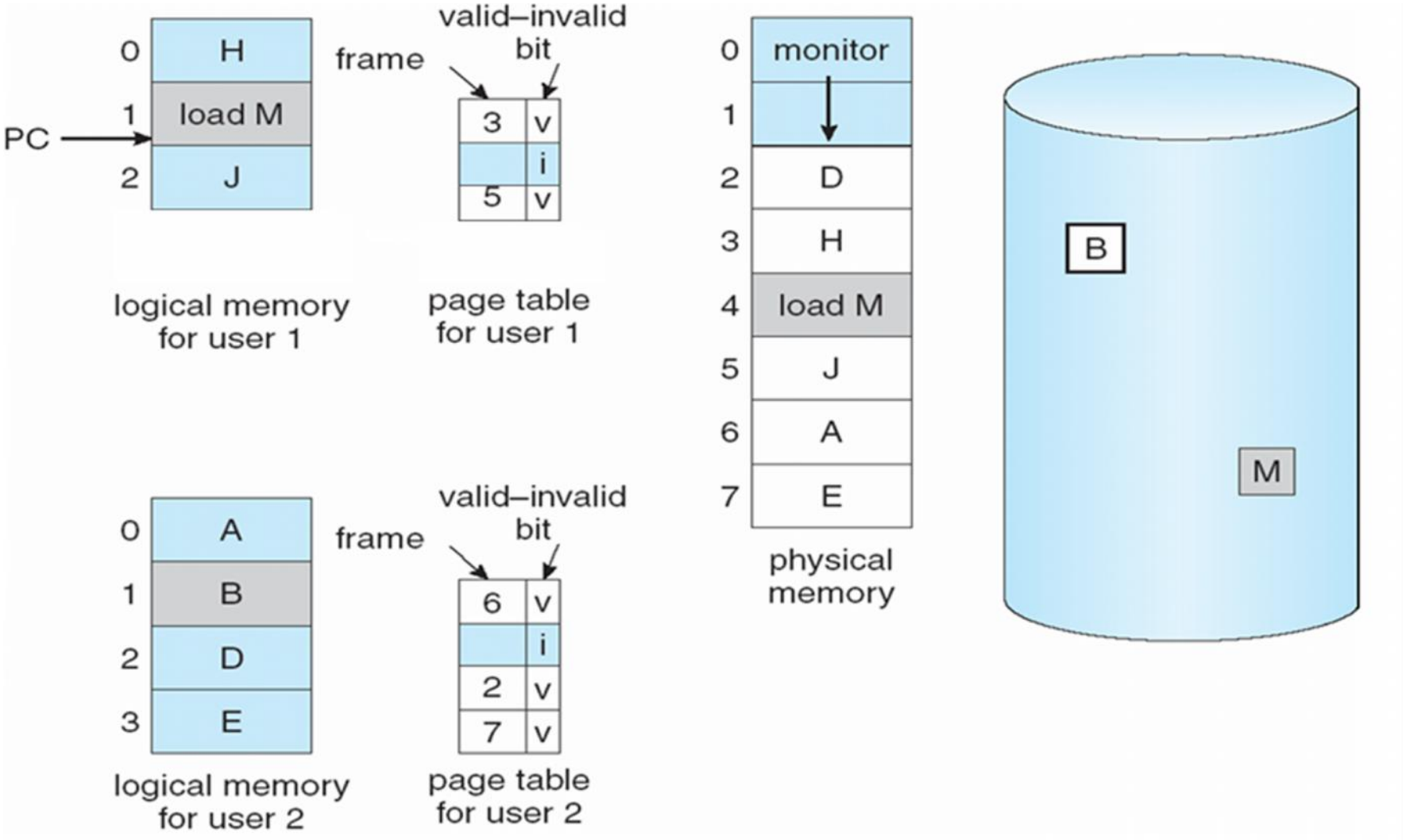


- Each page table entry contains a present bit to indicate whether the page is in main memory or not.
  - If it is in main memory, the entry contains the frame number of the corresponding page in main memory.
  - If it is not in main memory, the entry may contain the address of that page on disk
  - or the page number may be used to index another table (often in the PCB) to obtain the address of that page on disk.

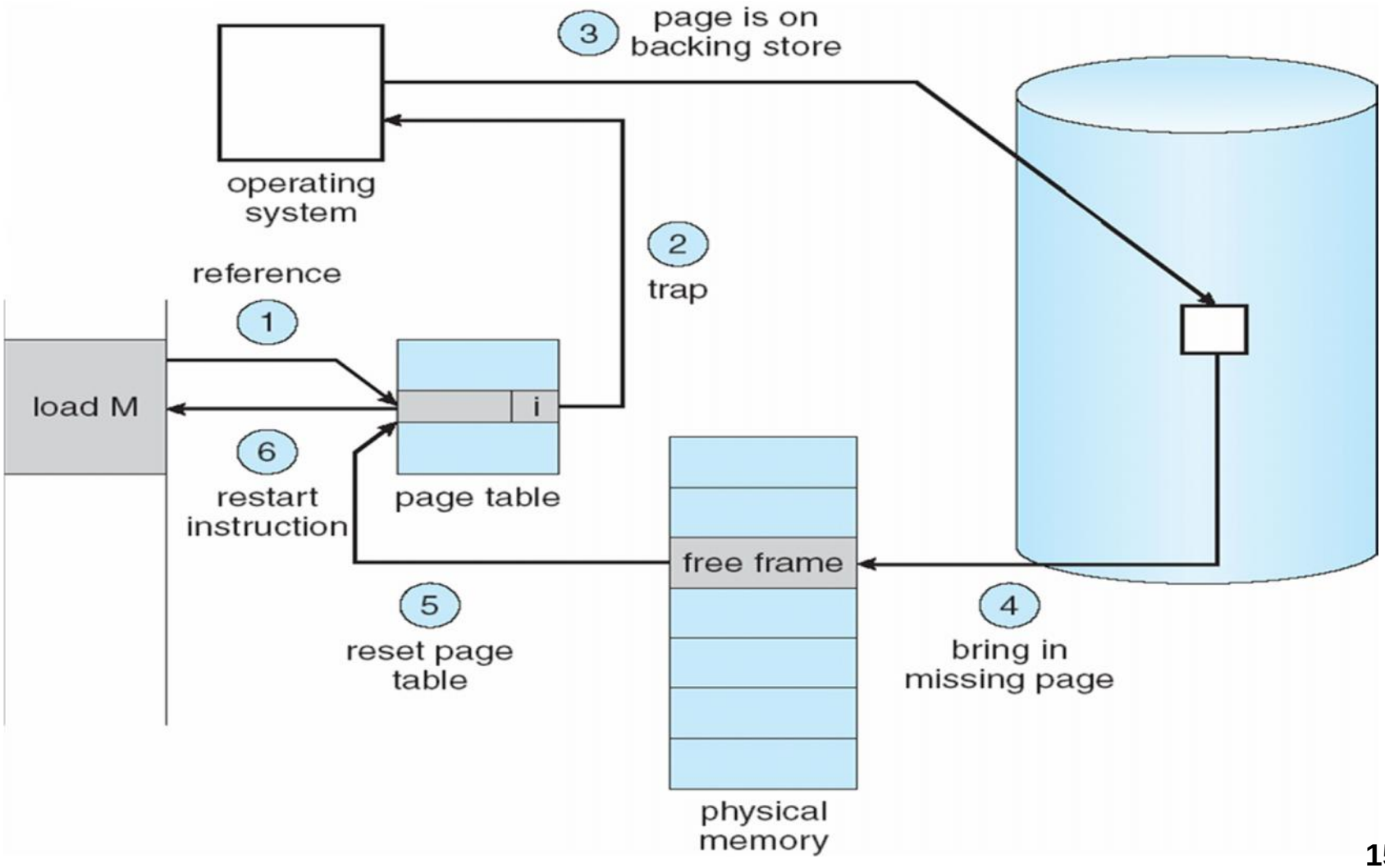
Demand Paging properties

- A modified bit indicates if the page has been altered since it was last loaded into main memory:
  - If no change has been made, page does not have to be written to the disk when it needs to be swapped out.
- Other control bits may be present if protection is managed at the page level:
  - Reference: page referenced or not
  - Protection: read-only/read-write bit.
  - Protection level bit: kernel page or user page
  - Caching bit: Enabled or not

Need For Page Fault/Replacement



Steps in handling a Page Fault



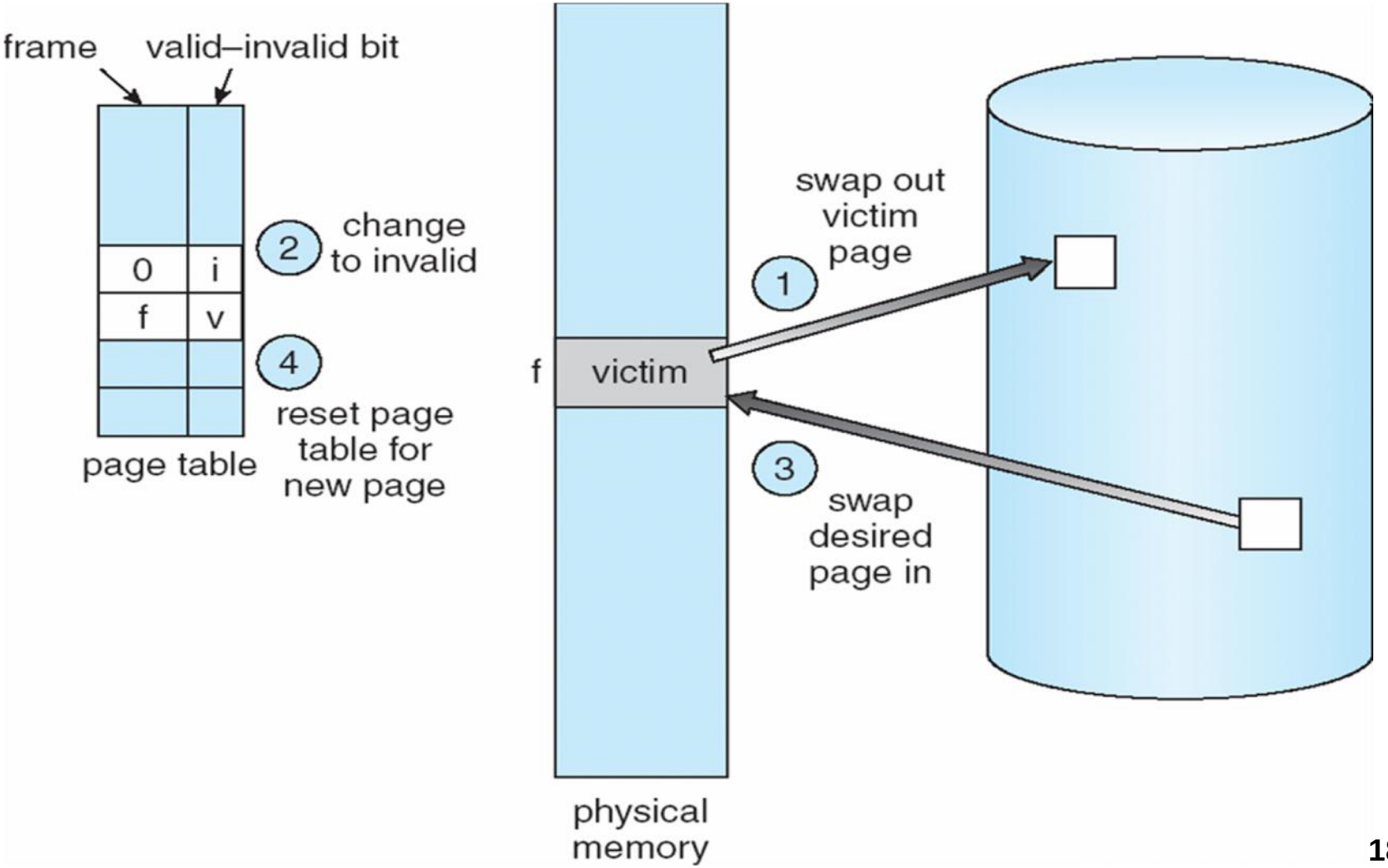
Steps in handling a Page Fault

1. If there is ever a reference to a page not in memory, first reference will cause page fault.
2. Page fault is handled by the appropriate OS service routines.
3. Locate needed page on disk (in file or in backing store).
4. Swap page into free frame (assume available).
5. Reset page tables – valid-invalid bit = 1.
6. Restart instruction.

Need of Page replacement

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

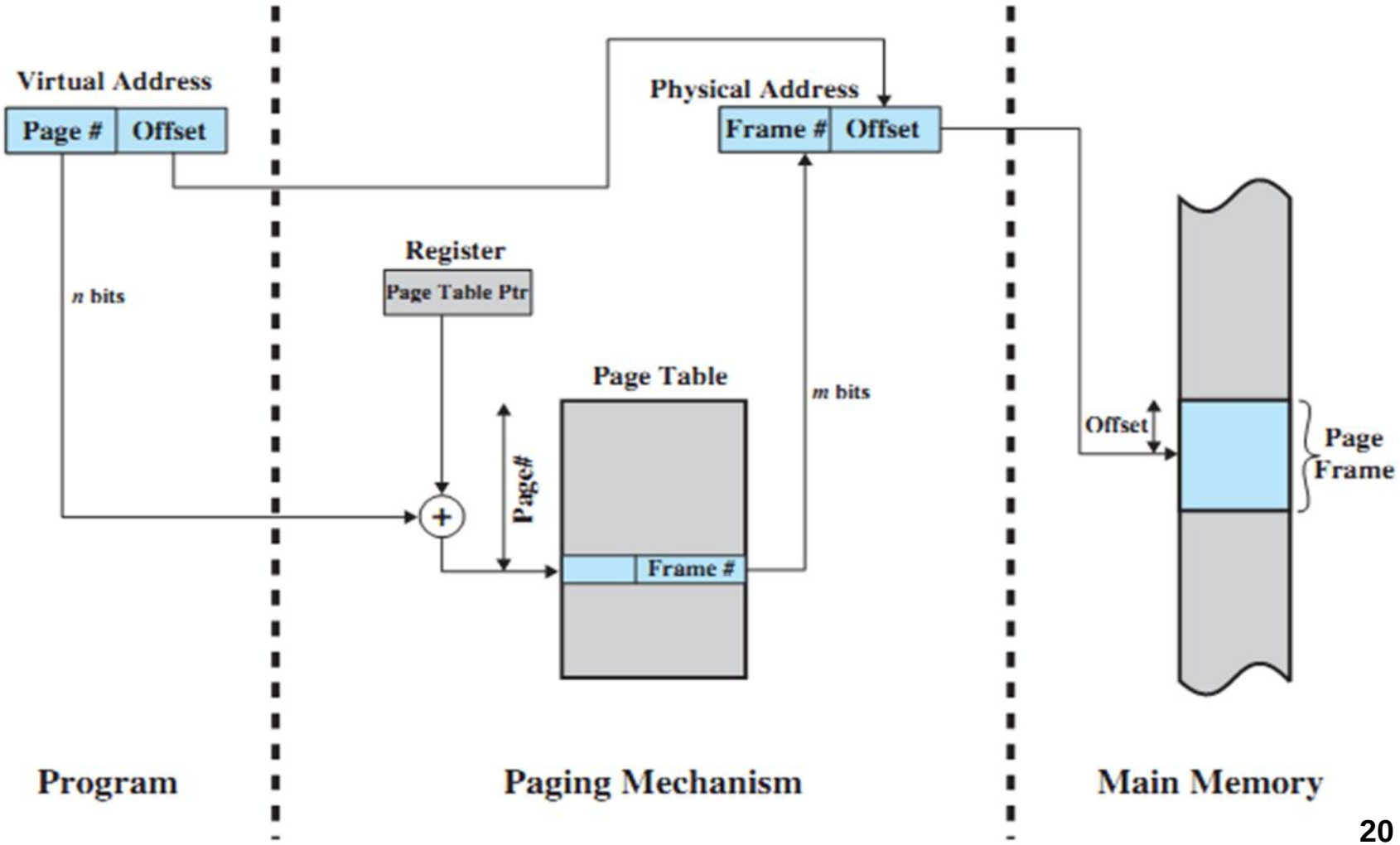
Steps in handling a Page Replacement



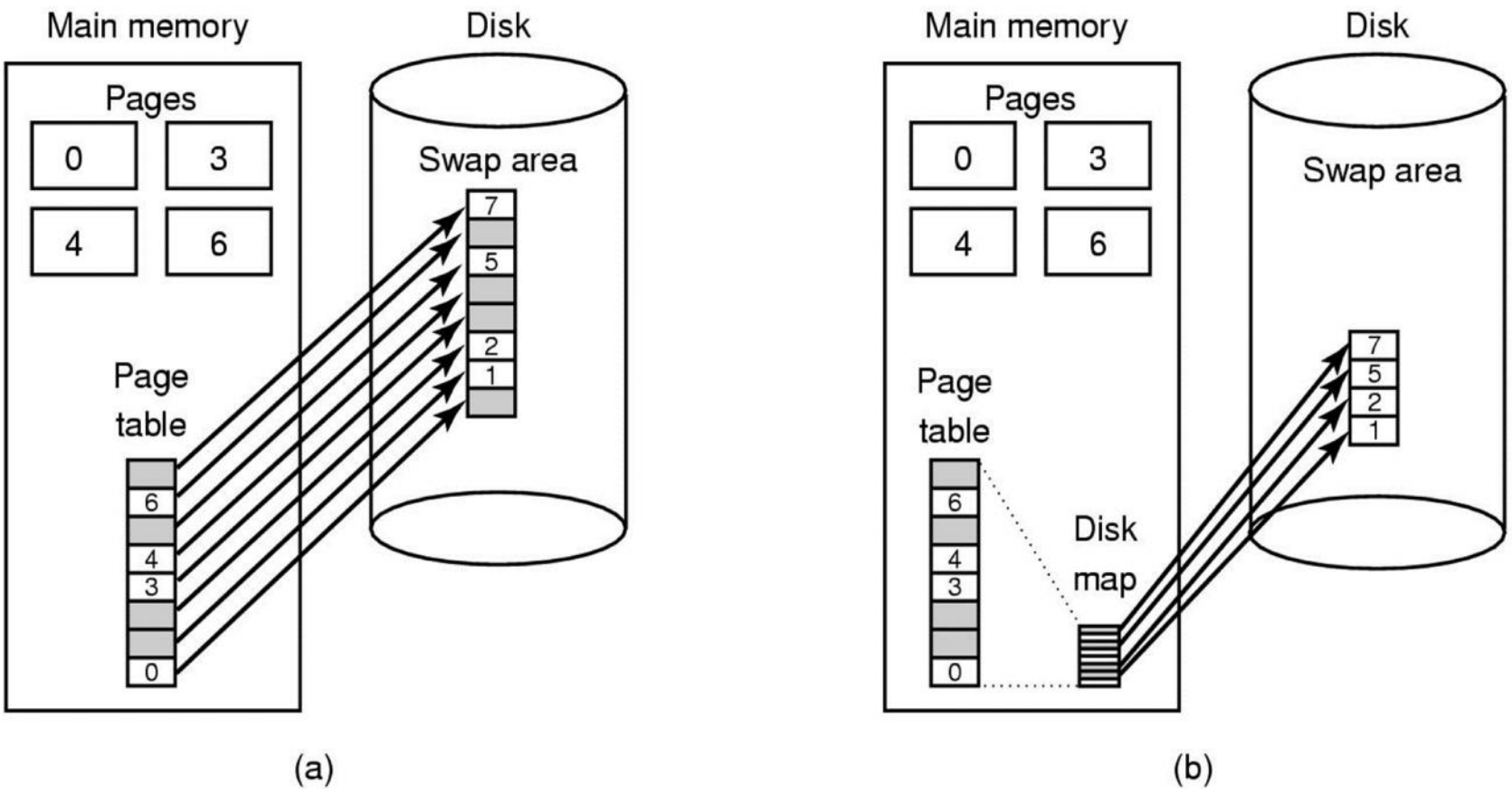
Steps in handling a Page Replacement

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

Address Translation in a Paging System



Backing/Swap Store



(a) Paging to static swap area. (b) Backing up pages dynamically.

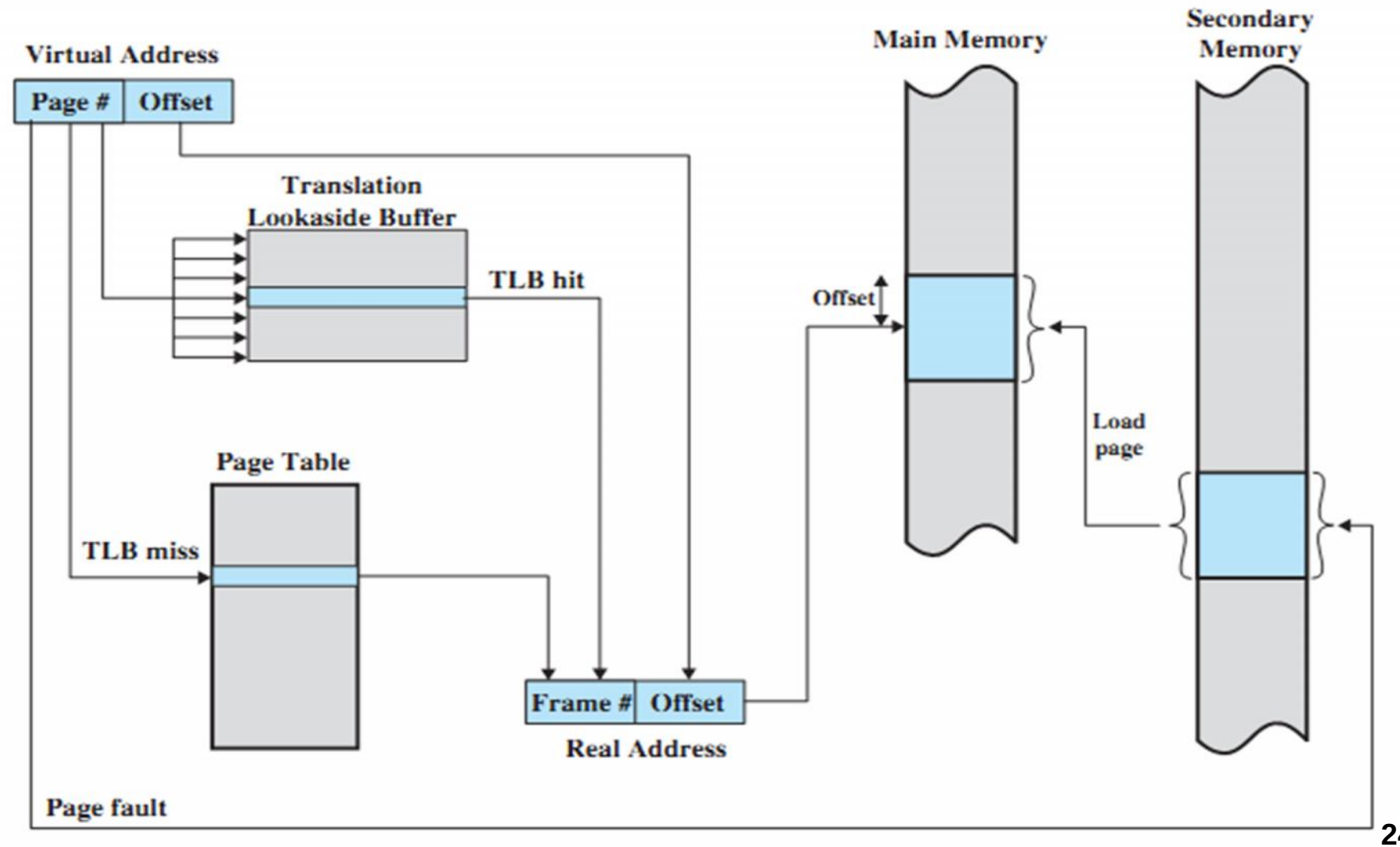
Need for TLB

- Because the page table is in main memory, each virtual memory reference causes at least two physical memory accesses:
  - one to fetch the page table entry.
  - one to fetch the data.
- To overcome this problem a special cache is set up for page table entries, called the TLB (Translation Look-aside Buffer):
  - Contains page table entries that have been most recently used.
  - Works similar to main memory cache.

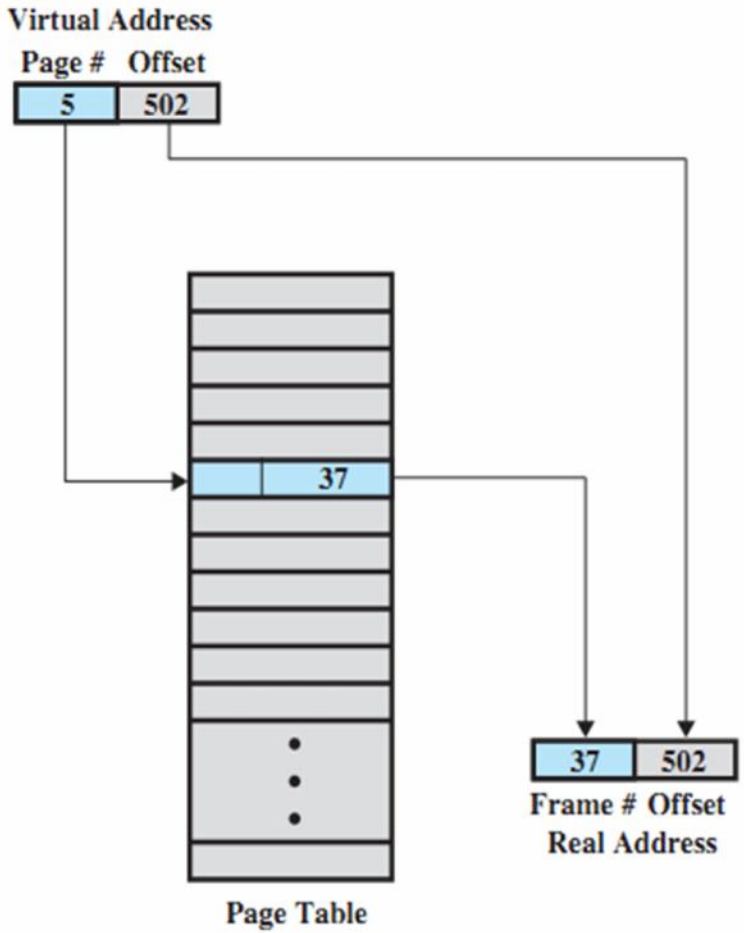
TLB properties

- Given a logical address, the processor examines the TLB.
- If page table entry is present (a hit), the frame number is retrieved and the real (physical) address is formed.
- If page table entry is not found in the TLB (a miss), the page number is used to index the process page table:
  - if valid bit is set, then the corresponding frame is accessed.
  - if not, a page fault is issued to bring in the referenced page in main memory.
- The TLB is updated to include the new page entry.

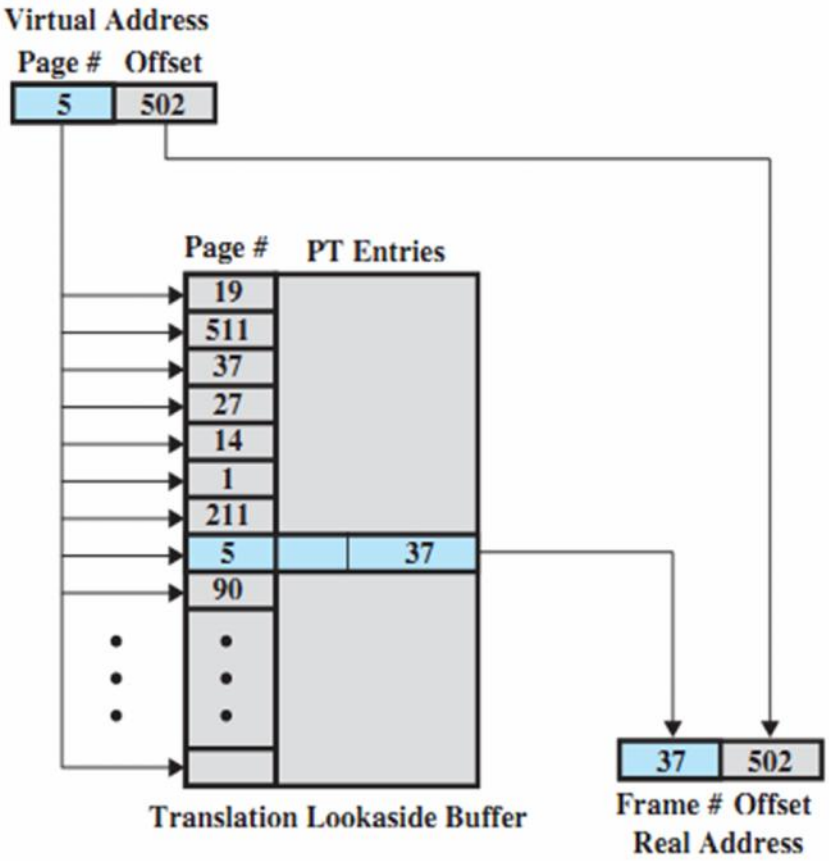
Use of a Translation Look-aside Buffer



Direct vs. Associative Lookup for Page Table Entries



(a) Direct mapping



(b) Associative mapping

TLB characteristics

- TLB use associative mapping hardware to simultaneously interrogates all TLB entries to find a match on page number.
- The TLB must be flushed each time a new process enters the Running state.
- The CPU uses two levels of cache on each virtual memory reference:
  - first the TLB: to convert the logical address to the physical address.
  - once the physical address is formed, the CPU then looks in the regular cache for the referenced word.

Questions

OPERATING SYSTEMS

Virtual Memory Management

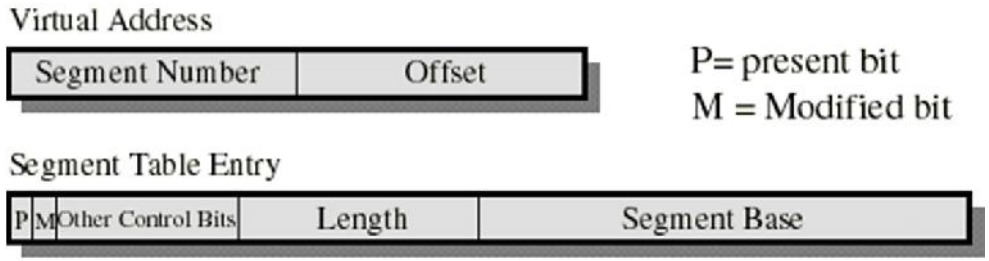
Demand Segmentation

Paging with Segmentation

Paging considerations

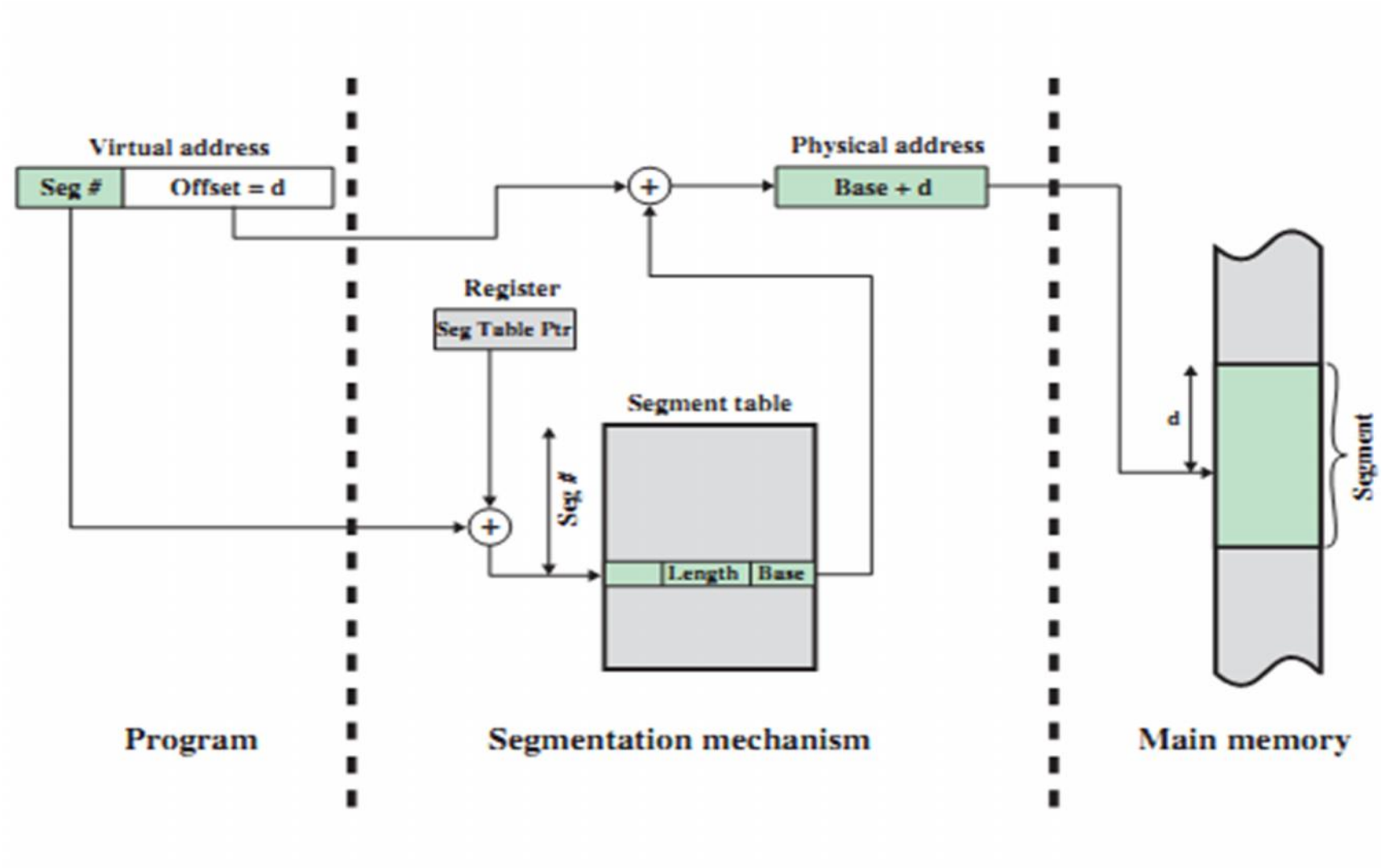
Dynamics of Segmentation

- Typically, each process has its own segment table.



- Similarly to paging, each segment table entry contains a present (valid-invalid) bit and a modified bit.
- If the segment is in main memory, the entry contains the starting address and the length of that segment.
- Other control bits may be present if protection and sharing is managed at the segment level.
- Logical to physical address translation is similar to paging except that the offset is added to the starting address (instead of appended).

Address Translation in a Segmentation System



Segmentation properties

- In each segment table entry, starting address (base) and length of the segment are available
- The segment can thus dynamically grow or shrink as needed.
- But variable length segments introduce external fragmentation and are more difficult to swap in and out.
- It is natural to provide protection and sharing at the segment level since segments are visible to the programmer (pages are not).
- Useful protection bits in segment table entry:
  - read-only/read-write bit
  - Kernel/User bit

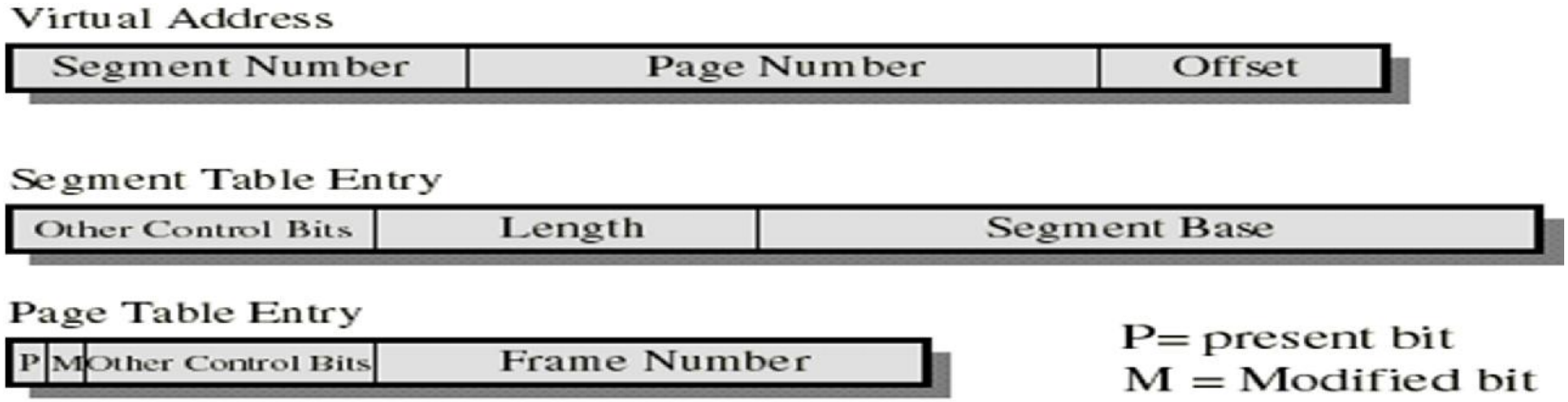
Comparison of Paging and Segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Combined Segmentation and Paging

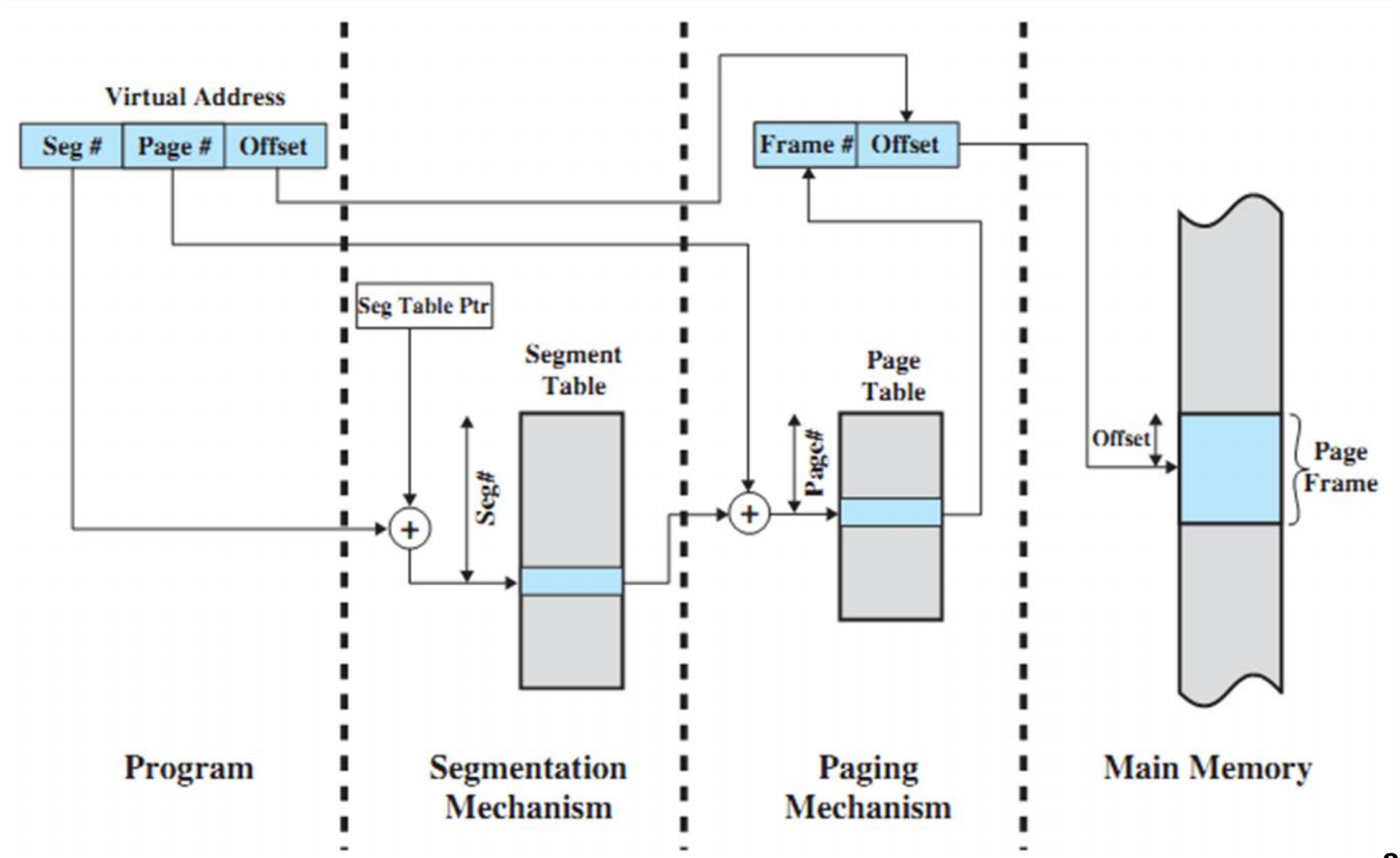
- To combine their advantages, some OSs page the segments.
- Several combinations exist – assume each process has:
  - one segment table.
  - several page tables: one page table per segment.
- The virtual address consists of:
  - a segment number: used to index the segment table who's entry gives the starting address of the page table for that segment.
  - a page number: used to index that page table to obtain the corresponding frame number.
  - an offset: used to locate the word within the frame.

Simple Combined Segmentation and Paging



- The Segment Base is the physical address of the page table of that segment.
- Present/modified bits are present only in page table entry.
- Protection and sharing info most naturally resides in segment table entry.
  - Ex: a read-only/read-write bit, a kernel/user bit.

Address Translation in combined Segmentation/Paging



Paging Considerations

- Locality, VM and Thrashing
- Pre-paging
- Page size issue
- TLB reach
- Program structure
- I/O interlock
- Copy-on-Write
- Memory-Mapped Files

Locality and Virtual Memory

- Principle of locality of references: memory references within a process tend to cluster.
- Hence: only a few pieces of a process will be needed over a short period of time.
- Possible to make intelligent guesses about which pieces will be needed in the future.
- This suggests that virtual memory may work efficiently (i.e., thrashing should not occur too often).

Possibility of Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization.
  - operating system thinks that it needs to increase the degree of multiprogramming.
  - another process added to the system.
  - This just increases the load on physical memory.
- **Thrashing** = a process is busy swapping pages in and out.

Possibility of Thrashing

- To accommodate as many processes as possible, only a few pieces of each process are maintained in main memory.
- But main memory may be full: when the OS brings one piece in, it must swap one piece out.
- The OS must not swap out a piece of a process just before that piece is needed.
- If it does this too often this leads to thrashing:
  - The processor spends most of its time swapping pieces rather than executing user instructions.

Pre-paging

- Anticipatory Paging
- It can help to reduce the large number of page faults that occurs at process startup or resumption.
- Pre-page all or some of the pages a process will need, before they are referenced.
- But if pre-paged pages are unused, I/O and memory was wasted.

The Page Size Issue

- Page size is always a power of 2 for more efficient logical to physical address translation
  - Large page size is good since for a small page size, more pages are required per process
  - More pages per process means larger page tables. Hence, a larger portion of page tables in virtual memory.
  - Large page size is good since disks are designed to efficiently transfer large blocks of data.
  - Larger page sizes means less pages in main memory; this increases the TLB hit ratio.

The Page Size Issue

- Small page size is good to minimize internal fragmentation.
- Lower page faults
- Increased page size causes each page to contain more code that is not used. Page faults rise.

- Page fault rate can determined by the number of frames allocated per process or according to size of the pages for process(es).
- Page sizes from 1KB to 4KB are most commonly used.
- Some systems supports multiple page sizes

TLB Reach

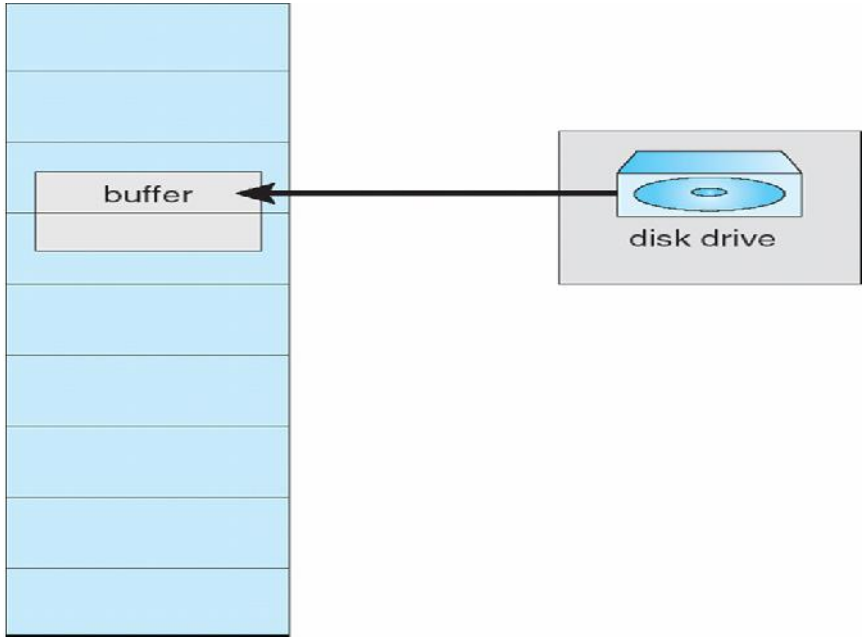
- The amount of memory accessible from the TLB.
- TLB Reach = (TLB Size) x (Page Size)
- Ideally, working set of each process is stored in TLB
  - Otherwise there is a high degree of page faults.
- Increase the size of TLB:
  - Much expensive
- Increase the Page Size:
  - This may lead to an increase in internal fragmentation as not all applications require a large page size.
- Provide Multiple Page Sizes:
  - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.

Program Structure

- The structure of program is necessary to know
- It is very difficult to share & protect pages without knowing the structure
- Segmentation along with paging can resolve this issue

I/O Interlock

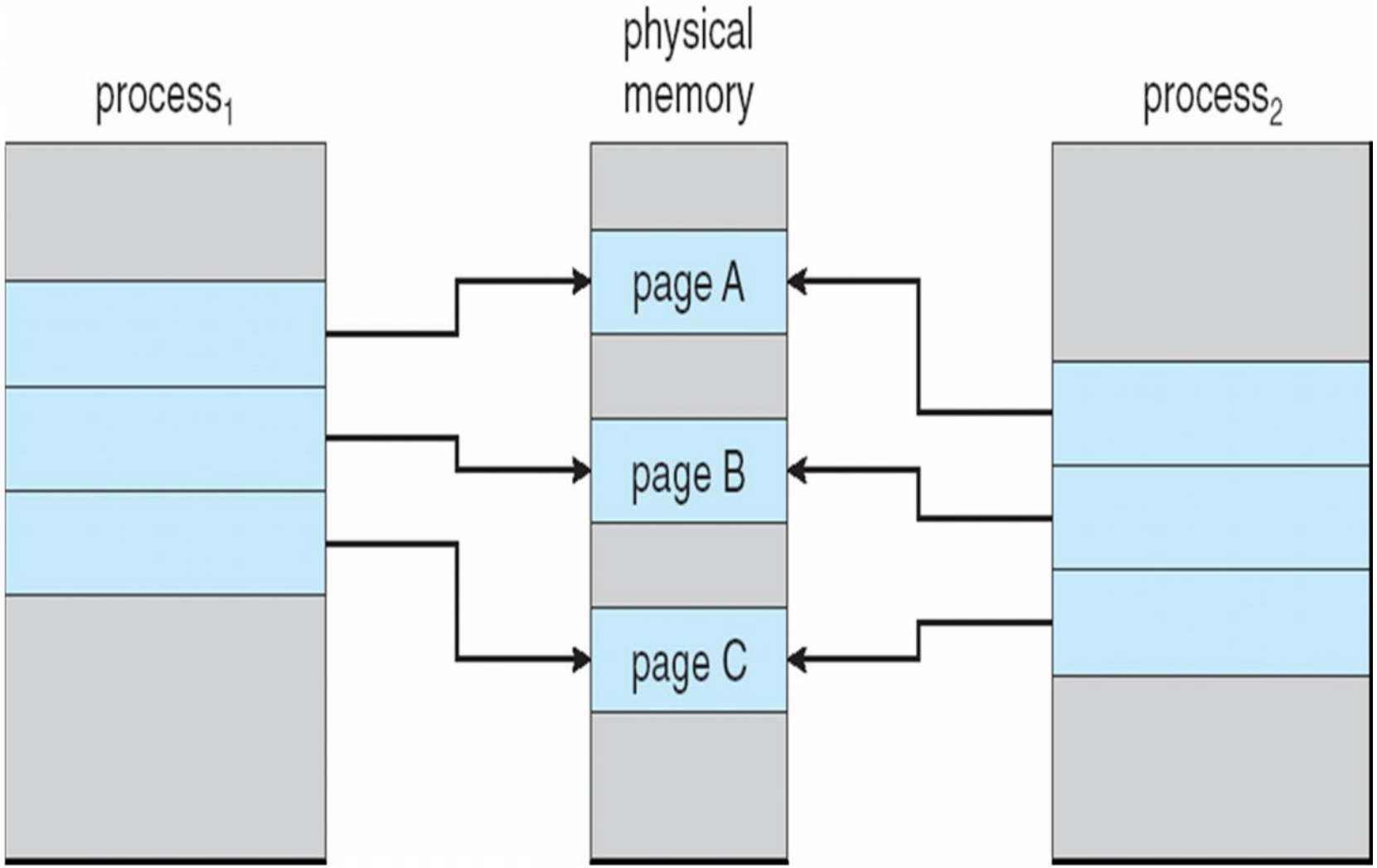
- **I/O Interlock** – Pages must sometimes be locked into memory.
- Pages that are used for copying a file from a device must be locked from being selected as victim page



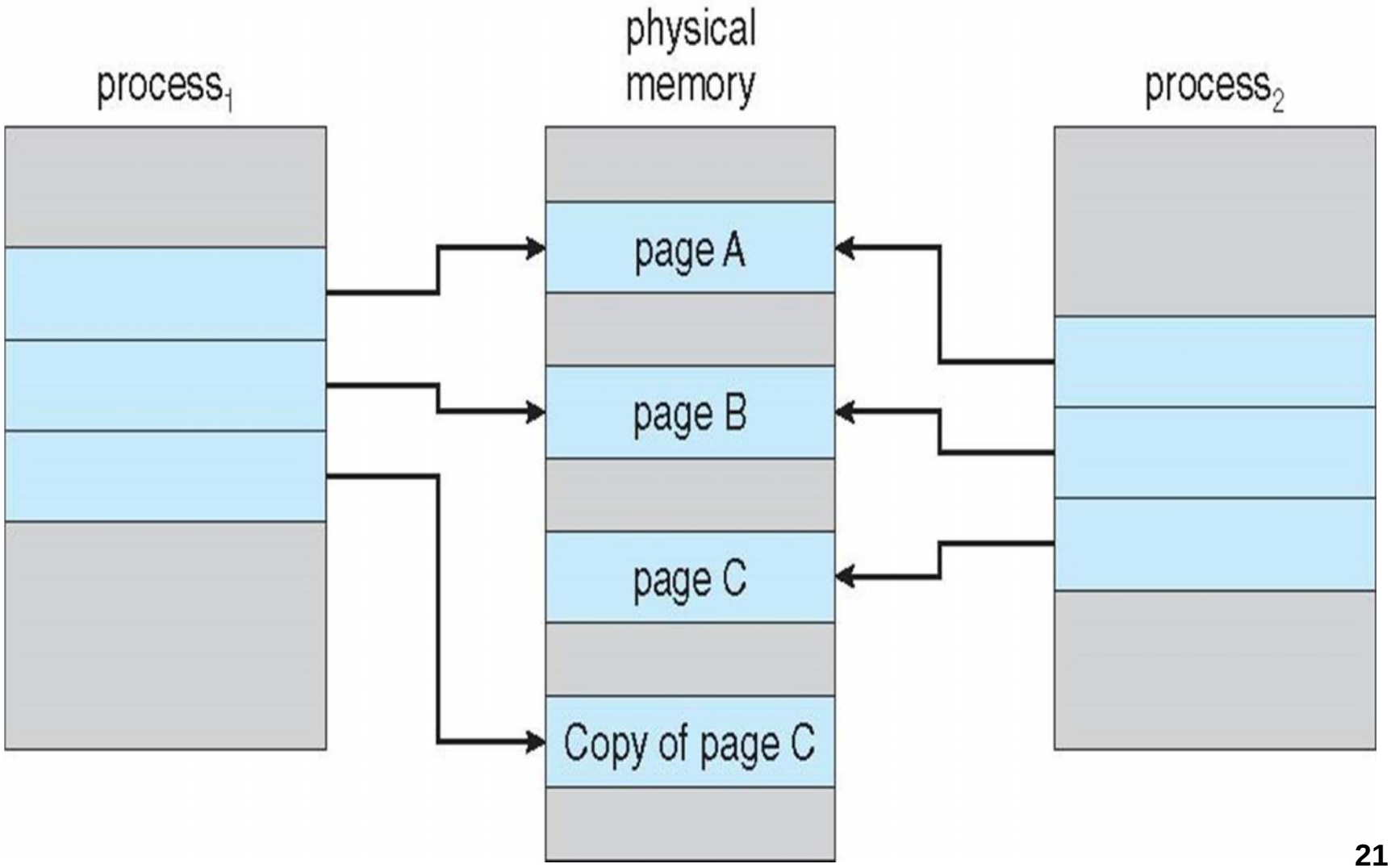
Copy-on-Write

- 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 the page is copied.
- COW allows more efficient process creation as only modified pages are copied.
- Pages are allocated from a pool of zeroed-out frames

Before Process 1 Modifies Page C



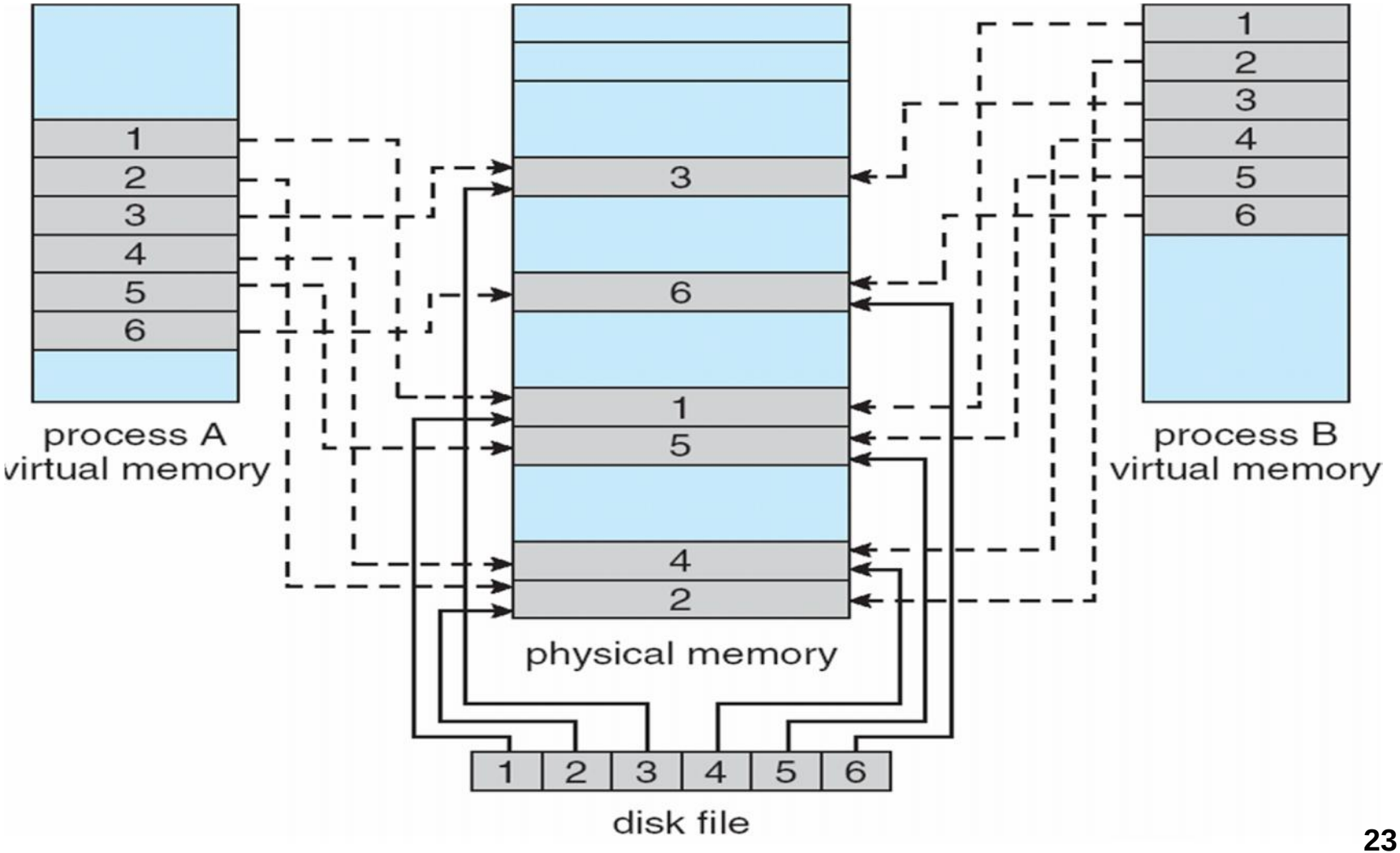
After Process 1 Modifies Page C



Memory-Mapped Files

- It 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 read/write operations from/to file are treated as ordinary memory accesses.
- Also allows several processes to map the same file allowing the pages in memory to be shared.

Layout of Memory Mapped Files



Questions

OPERATING SYSTEMS

Virtual Memory Management

Page Replacement Algorithms

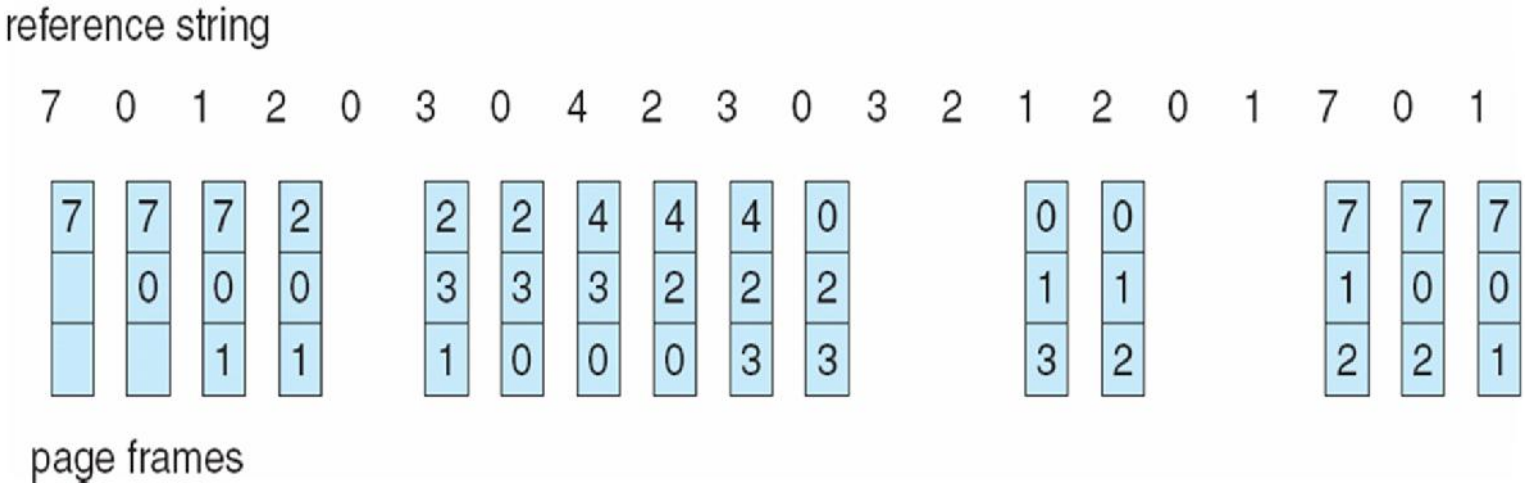
Page Buffering

Cleaning Policy

The FIFO Policy

- Treats page frames allocated to a process as a circular buffer:
  - When the buffer is full, the oldest page is replaced. Hence first-in, first-out:
    - A frequently used page is often the oldest, so it will be repeatedly paged out by FIFO.
  - Simple to implement:
    - requires only a pointer that circles through the page frames of the process.

FIFO Page Replacement



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

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

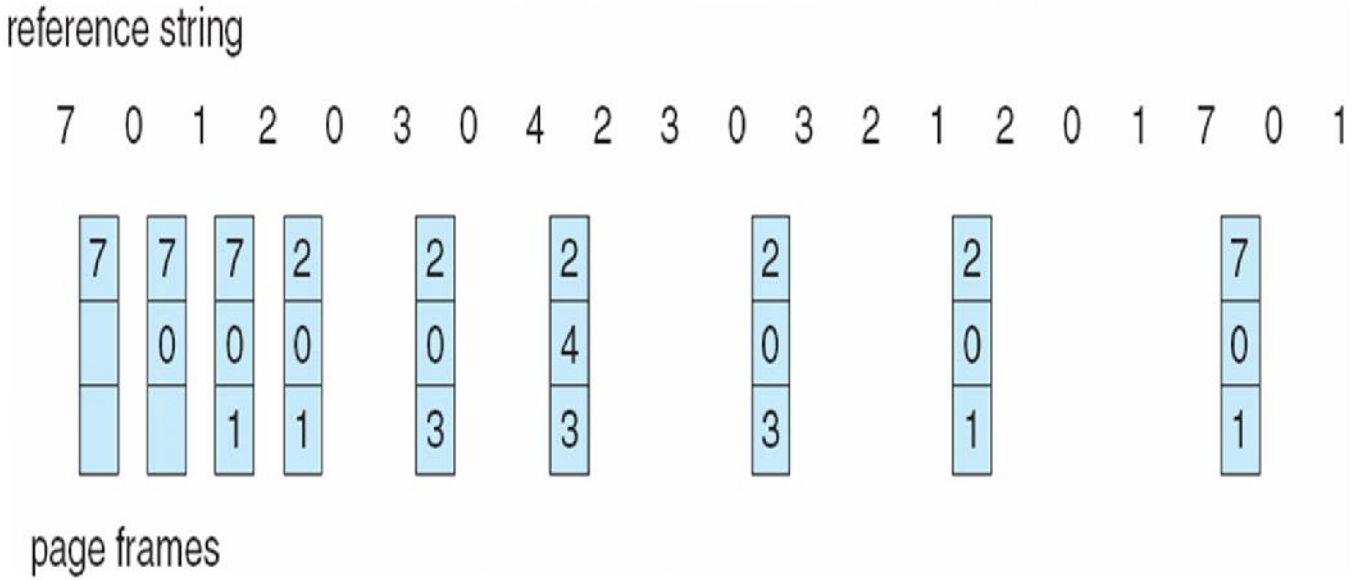
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

FIFO Replacement manifests Belady's Anomaly:

- more frames $\Rightarrow$ more page faults

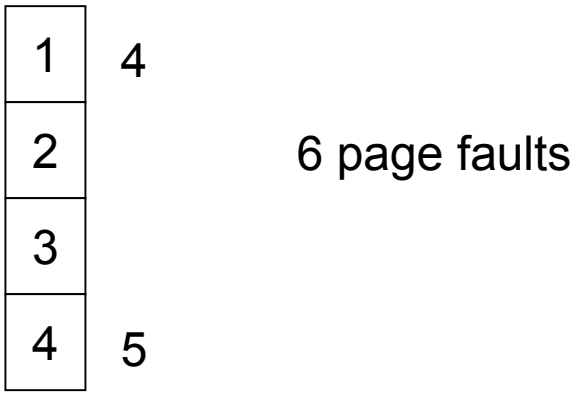
Optimal Page Replacement

- The Optimal policy selects for replacement the page that will not be used for longest period of time.
- Difficult to implement (need to know the future)
- Serves as a standard to compare with the other algorithms.



Optimal Algorithm

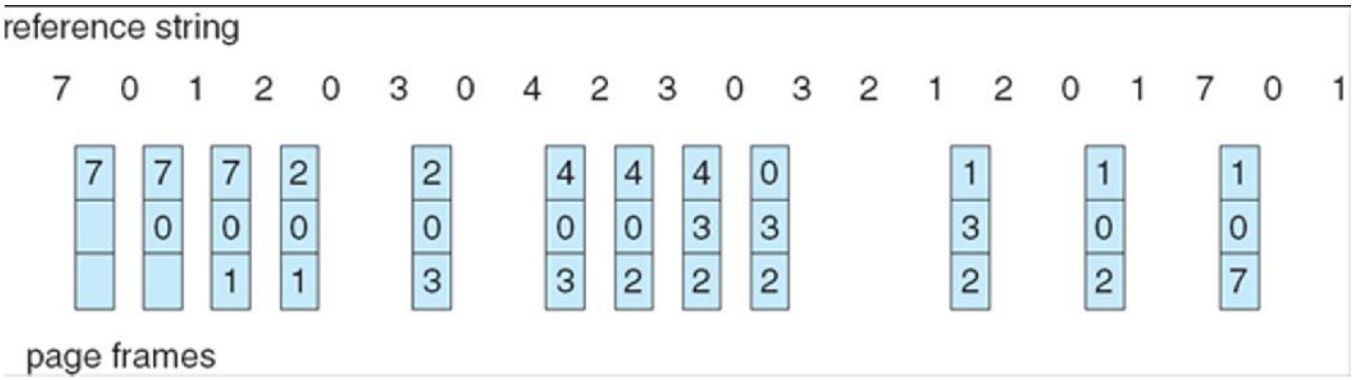
- Reference string : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 4 frames example



- Impossible to know the pages referred in future
- Used for measuring how well your algorithm performs.

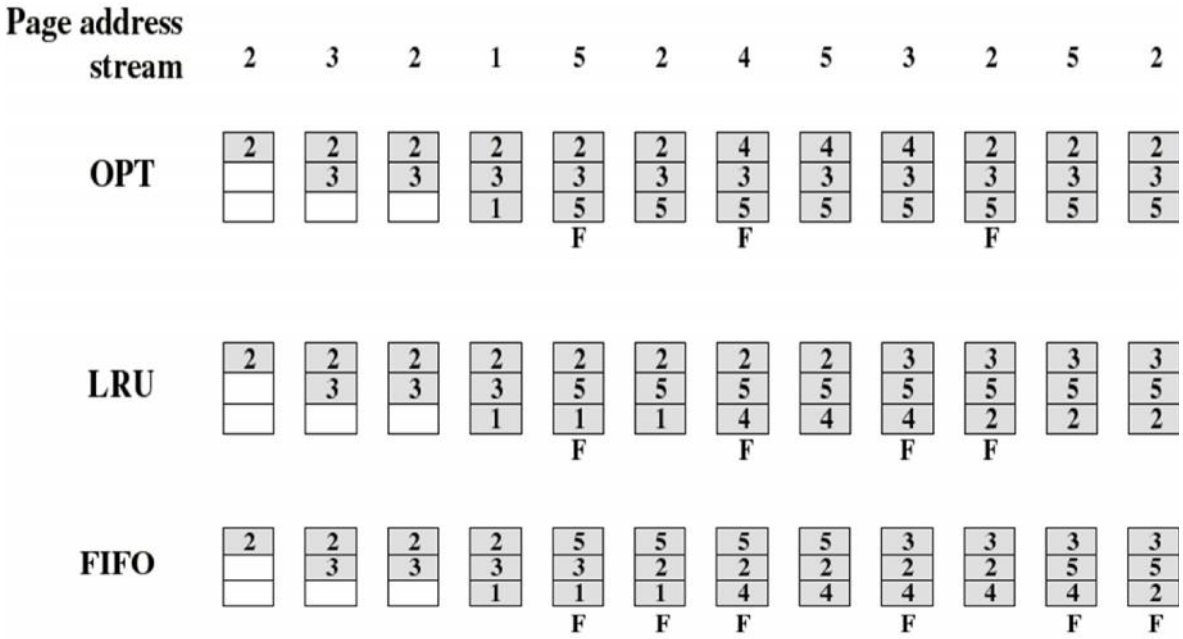
The LRU Policy

- Replaces the page that has not been referenced for the longest time:
  - By the principle of locality, this should be the page least likely to be referenced in the near future.
  - performs nearly as well as the optimal policy.



Comparisons

- Example: A process of 5 pages with an OS that fixes the resident set size to 3.



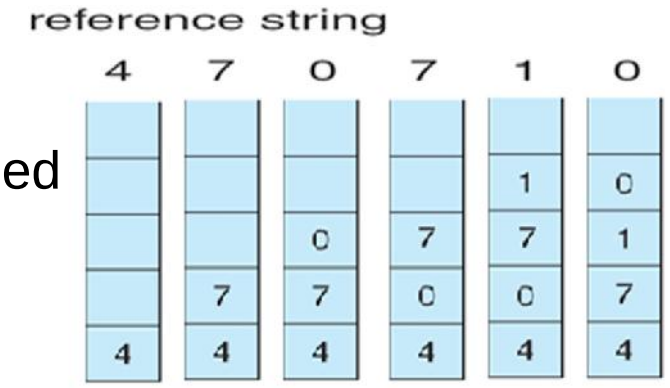
- LRU recognizes that pages 2 and 5 are referenced more frequently than others but FIFO does not.

LRU variants

- LRU is a counting based technique from where 2 more techniques can be derived:
  - NRU (Not Recently Used): Replace the page that is not used for too long time or not recently referred
  - MRU (Most Recently Used): The page with least count can be the page which could be used in near future, so replace the page with highest count .
- Both of these variants can be implemented as LRU
- While
  - MRU seems not to be effective in prediction of future pages in real scenarios, so it is not used practically,

LRU Implementations

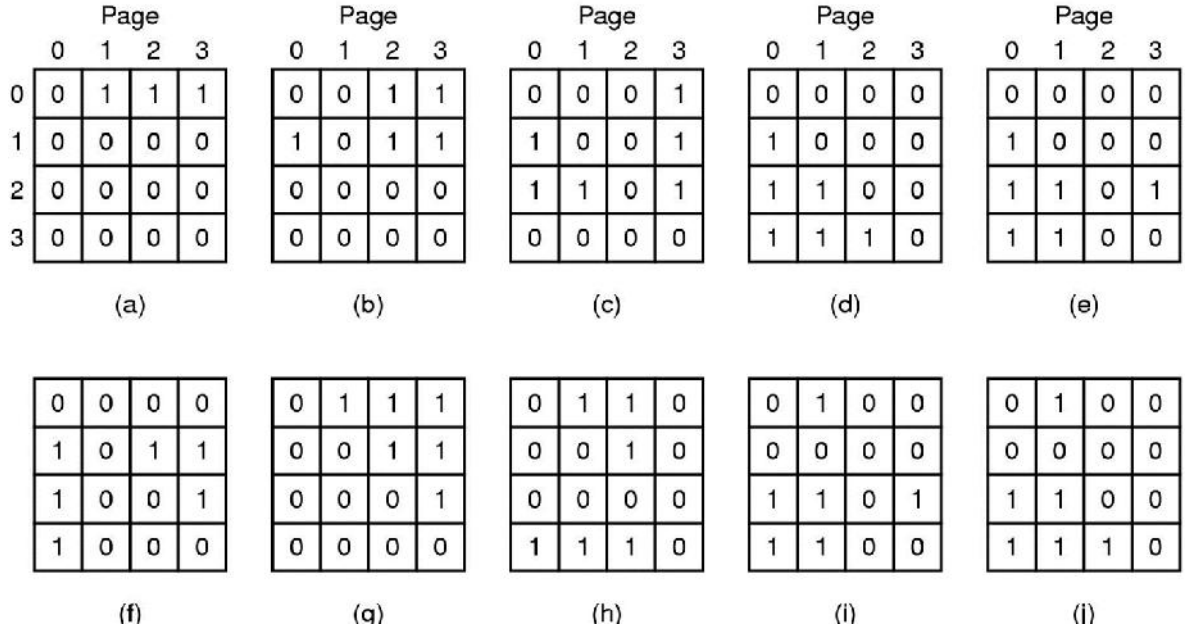
- Counter implementation:
  - Every page entry has a counter; every time a page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be changed, look at the counters to determine lowest value to select victim page.
  - Requires more page table memory & comparison time
- Stack implementation:
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires several pointers to be changed
  - Victim page at bottom of stack
    - No search for replacement.



LRU Implementations

- Matrix LRU Implementation:
  - Mostly implemented by Hardware.
  - A Matrix of n by n, where n is total number of pages available in RAM.
  - If a page is accessed, then set all bits as 1 to its row & then set all bits as 0 to its concerned column.
  - Select the page that have lowest number for replacement

Pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3

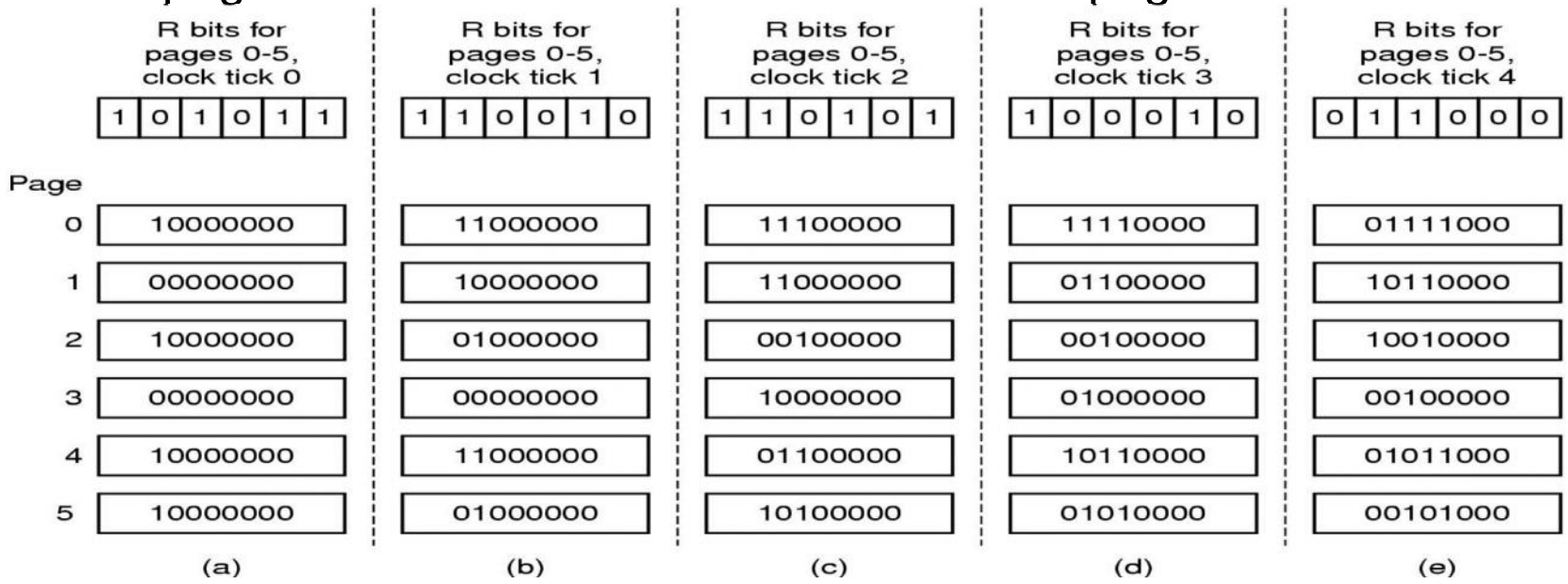


LRU Implementations

- Reference Bit:
  - With each page associate a bit, initially = 0
  - When page is referenced, bit is set to 1.
  - Replace the page which contains 0 in reference bit.
  - Purely NRU technique

LRU Implementations

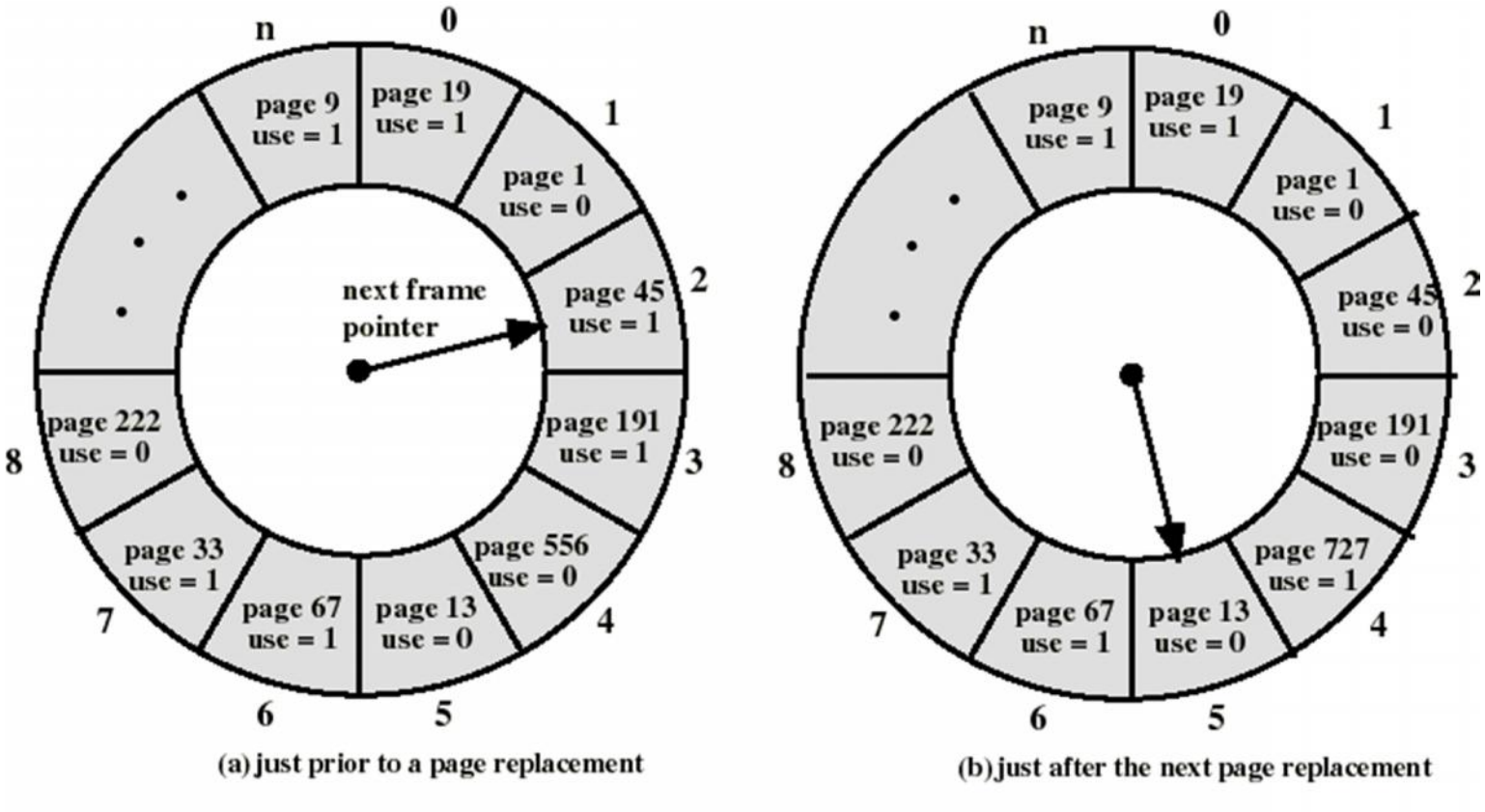
- Reference Byte:
  - Record reference bits at regular intervals; Keep a byte of reference bits for each page.
  - At regular chosen time intervals, left shift the reference bit of each page:
    - If page is referred then shift 1 bit else shift 0 in reference byte
  - Each reference byte keeps the history of the page.
  - The page with the lowest number is the victim page.



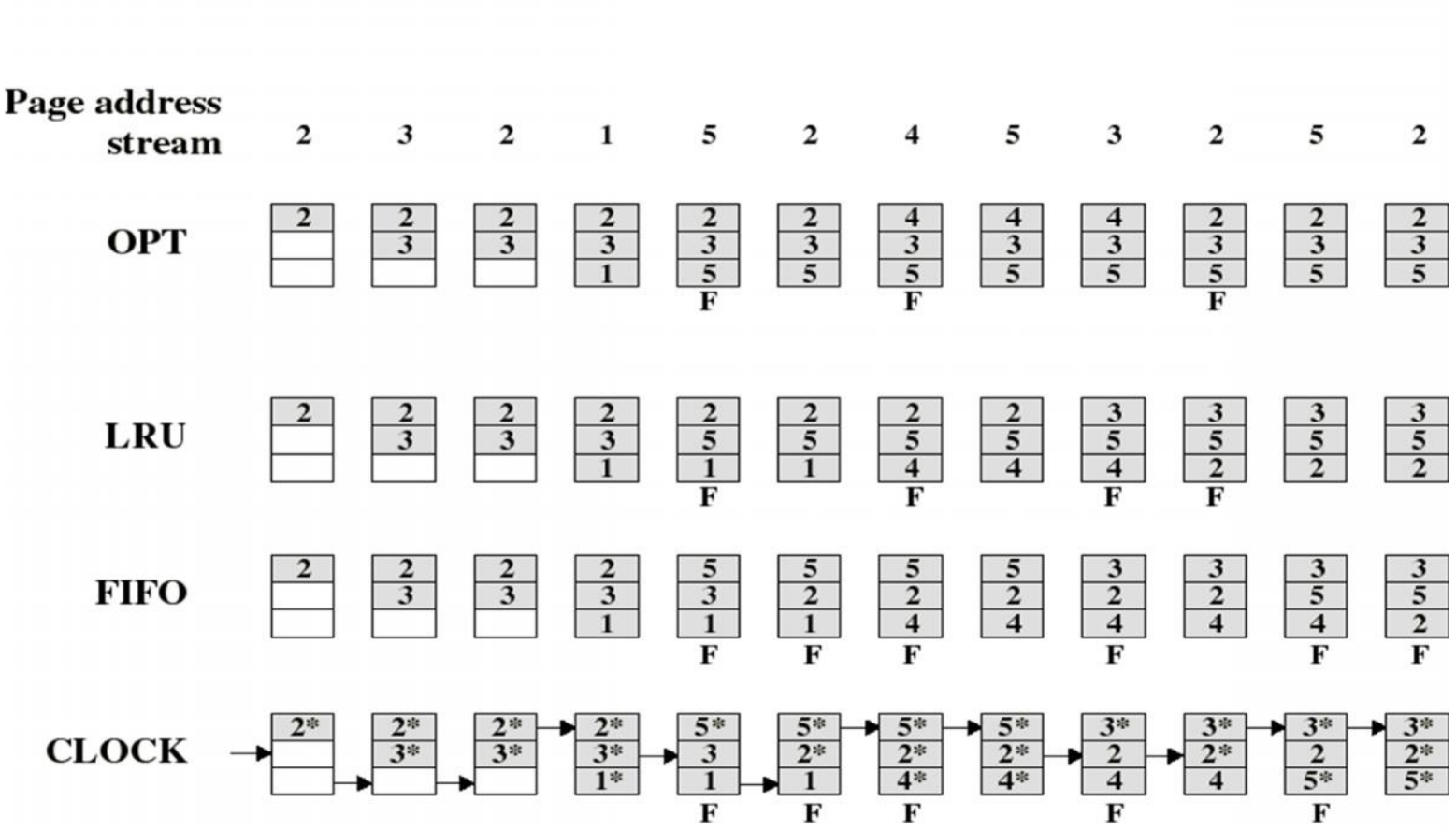
The Clock (Second Chance) Policy

- The variant of FIFO and/or NRU
- Set of pages in RAM are considered as a circular queue.
- When a page is replaced/loaded, a pointer is set to point the next frame in queue.
- A reference bit for each page is set to 1 whenever:
  - a page is first loaded into the frame.
  - the corresponding page is referred again (if reference bit is 0).
- For replacement, it start searching the pages from the frame pointer:
  - If a page is encountered with reference bit set to 1 & it is set to 0 (Gives page a second chance to reside in RAM).
  - If a page with reference bit set to 0 is encountered then it is replaced

The Clock Policy Example



Comparisons



- Asterisk indicates that the corresponding use bit is set to 1.
- The arrow indicates the current position of the pointer.

Page Buffering

- Also known as pooling
- Replacement algorithm is applied before page fault
- Several pages those are selected for replacement are still kept in main memory for a while to improve performance & to help in cleaning process.
- Two lists are maintained:
  - a free page list for frames that have not been modified since brought in (no need to swap out, clean pages).
  - a modified page list for frames that have been modified (need to write them out, dirty pages).
- When a page is selected as victim page, it present bit is cleared in page table & is added in one of above lists; but the page remains in the same memory frame.

Page Buffering

- At each page fault the two lists are examined to see if the needed page is still in main memory:
  - If it is, then set the present bit in the corresponding page table entry and remove its entry form free/modified page list
  - If it is not, then the needed page is brought in and the replaced page entry is removed from free/modified page list.

Cleaning Policy

- Writing back the modified pages
- Demand cleaning:
  - a page is written out only when it's frame has been selected for replacement
    - but a process that suffers a page fault may have to wait for 2 page transfers.
- Pre-cleaning:
  - modified pages are written before their frames are needed so that they can be written out in groups:
    - but majority of them will be modified again before they are replaced.
- Using Page Buffering:
  - pages on the modified list can be periodically written out in batches and moved to the free list
  - not all dirty pages but only those which are chosen for replacement are written out

Questions