Virtual Memory

Chapter 8

Characteristics of Paging and Segmentation

- Memory references are dynamically translated into physical addresses at run-time
  - a process may be swapped in and out of main memory, so it might occupy different regions at different times
- A process can be broken up into pieces that do not need to be located contiguously in main memory
  - No need to load all pieces of a process in main memory during execution at any given time

Execution of a Program

- Operating system brings into main memory a few pieces of the program
- *Resident set* - portion of process that is in main memory
- A *page fault* interrupt is generated when referencing an address that is not in main memory
- Operating system places the process in a blocked state

Execution of a Program

- Piece of process memory that contains the referenced logical address is brought into main memory as follows:
  - OS issues a disk I/O Read request
  - another process is dispatched to run while the disk I/O takes place
  - an interrupt is issued by the disk when it completes the I/O; this tells the OS to place the affected process (the one that caused the page fault) in the Ready state

Advantages of Breaking up Process

- More processes can be maintained in main memory
  - can load only some of the pieces of each process
- With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- It is possible for a process to be larger than all the main memory

Advantages of Breaking up Processes

- ✓ Programmer is dealing with memory the size of a portion of a hard disk
- ✓ It would be wasteful to load in many pieces of the process when only a few pieces might be used
- ✓ Time can be saved because unused pieces are not swapped in and out of memory

Types of Memory

- ✓ Real memory
  - main memory
- ✓ Virtual memory
  - memory on disk

Thrashing

- Swapping out a piece of a process just before that piece is needed again
- The processor spends most of its time swapping pieces rather than executing user instructions

Principle of Locality

- ✓ Program and data references within a process tend to cluster
- ✓ 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

Support Needed for Virtual Memory

- ✓ Hardware must support paging and/or segmentation
- ✓ Operating system must be able to manage the movement of pages and/or segments between secondary memory and main memory

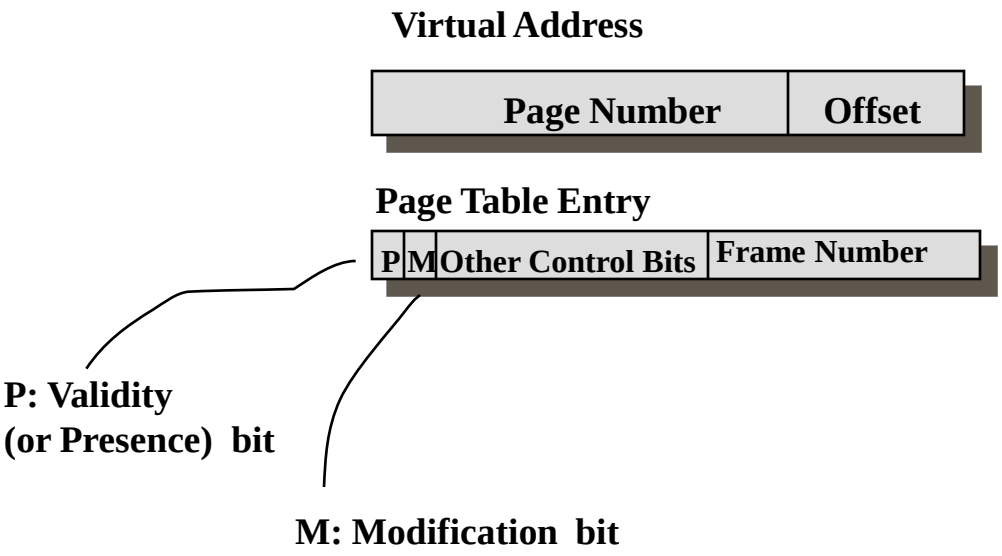
Paging

- Each process has its own page table
- Each page table entry contains the frame number of the corresponding page in main memory
- A bit is needed to indicate whether the page is in main memory or not

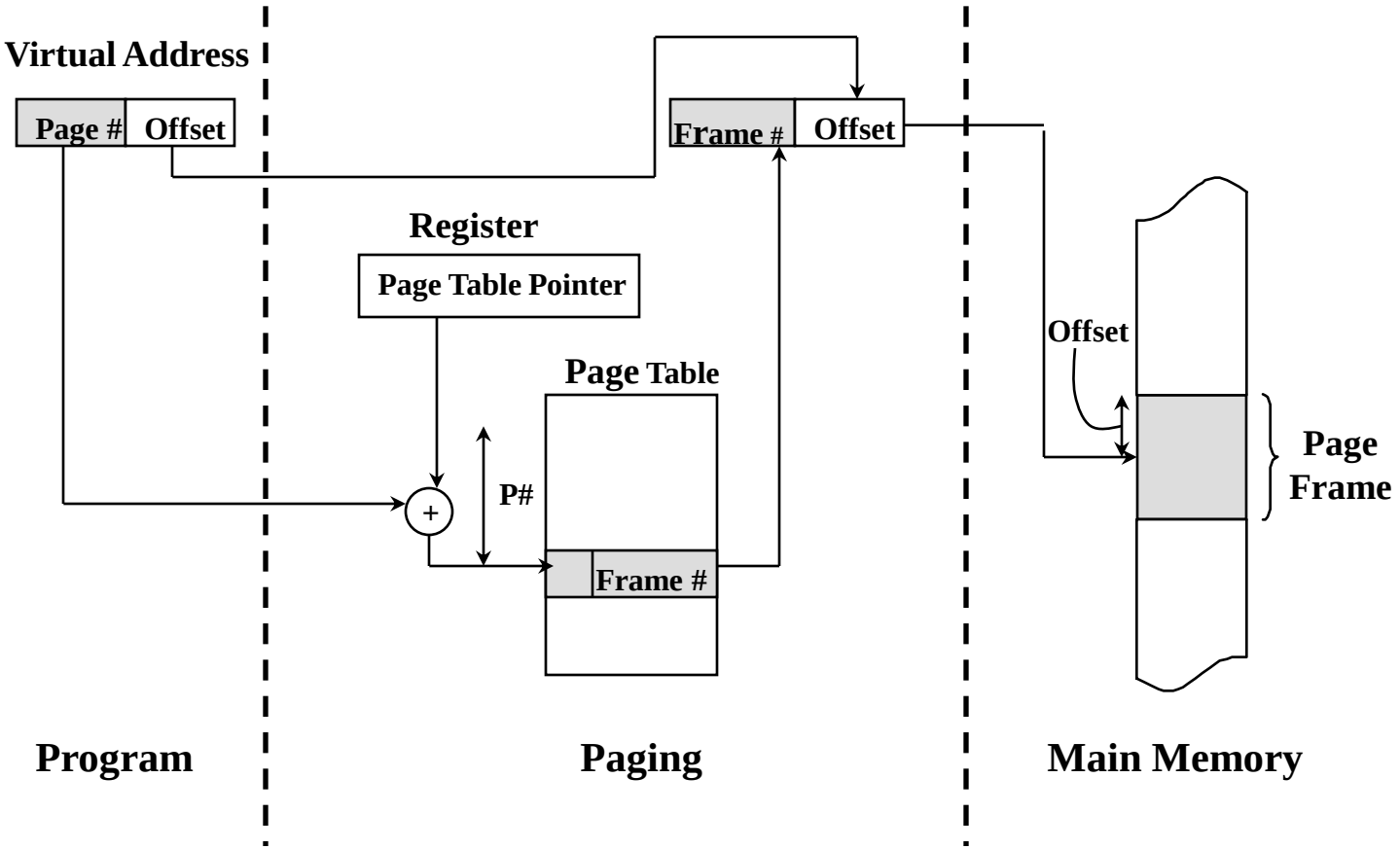
Modification Bit in Page Table

- A *modification bit* is needed to indicate if the page has been altered since it was last loaded into main memory
  - If no change has been made (the mod bit is *off*), the page does not have to be written to the disk when it needs to be swapped out;
  - if the bit is *on* -- the page must be first saved on disk before being swapped out.

Paging


Virtual Address
Page Number
Offset
Page Table Entry
P
M
Other Control Bits
Frame Number
P: Validity
(or Presence) bit
M: Modification bit

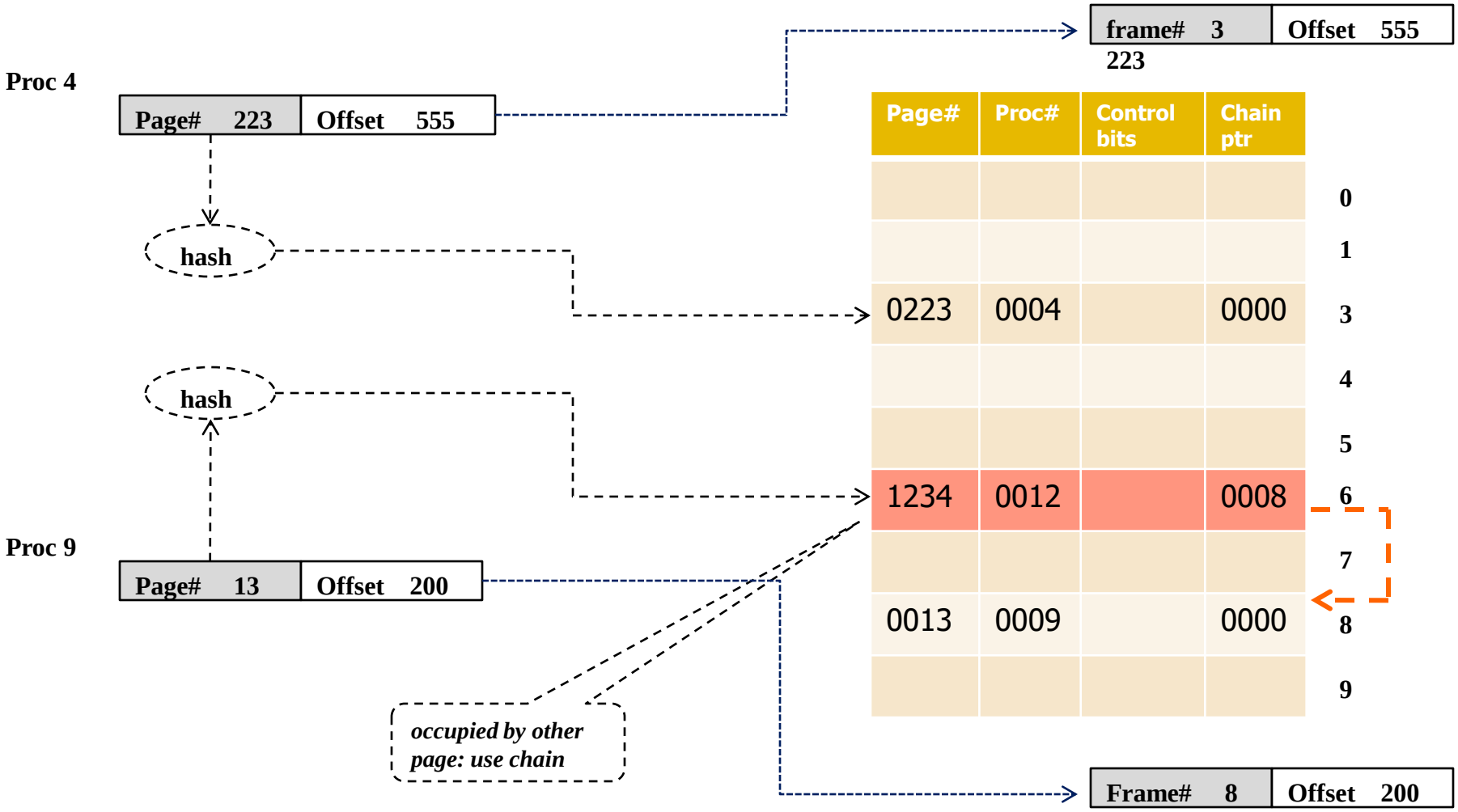
Address Translation in a Paging System



Page Tables

- The entire page table may take up too much main memory
- Page tables are also stored in virtual memory (of the kernel)
- When a process is running, part of its page table is in main memory
- Another method (especially popular with 64 bit OS): inverted tables - next

Inverted Page Tables



Inverted Page Tables

- ✓ Only one table for all processes, not per process
  - Size: the number of frames
- ✓ Chains – a disadvantage
  - But usually their size is 0-2.

Translation Look-aside Buffer

- Each virtual memory reference can cause two physical memory accesses
  - one to fetch the page table
  - 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*

Translation Look-aside Buffer

- ✓ Contains page table entries that have been most recently used
- ✓ Works similarly to main memory cache

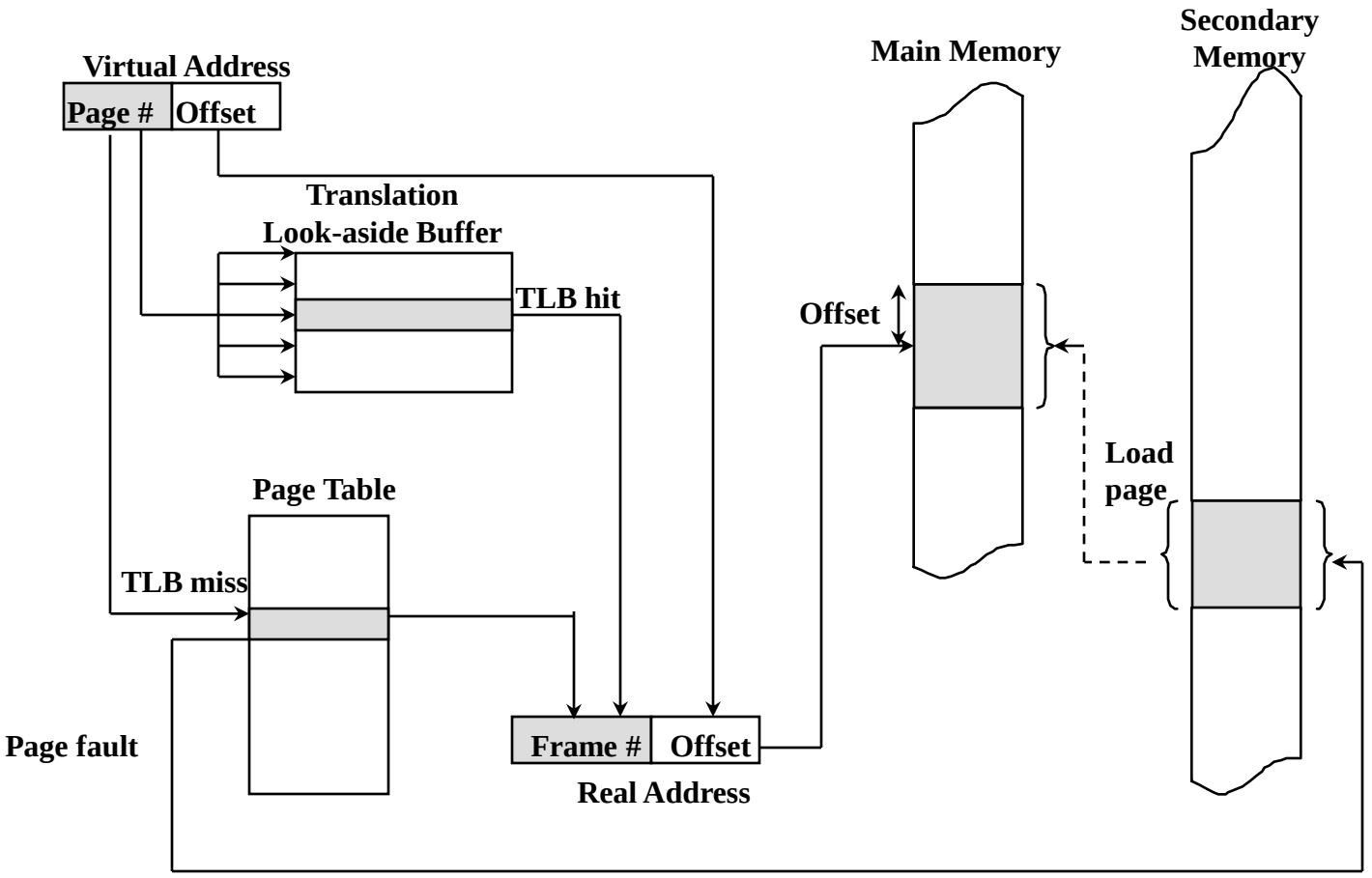
Translation Look-aside Buffer

- Given a virtual address, processor examines the TLB
- If page table entry is present (a hit), the frame number is retrieved and the real 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

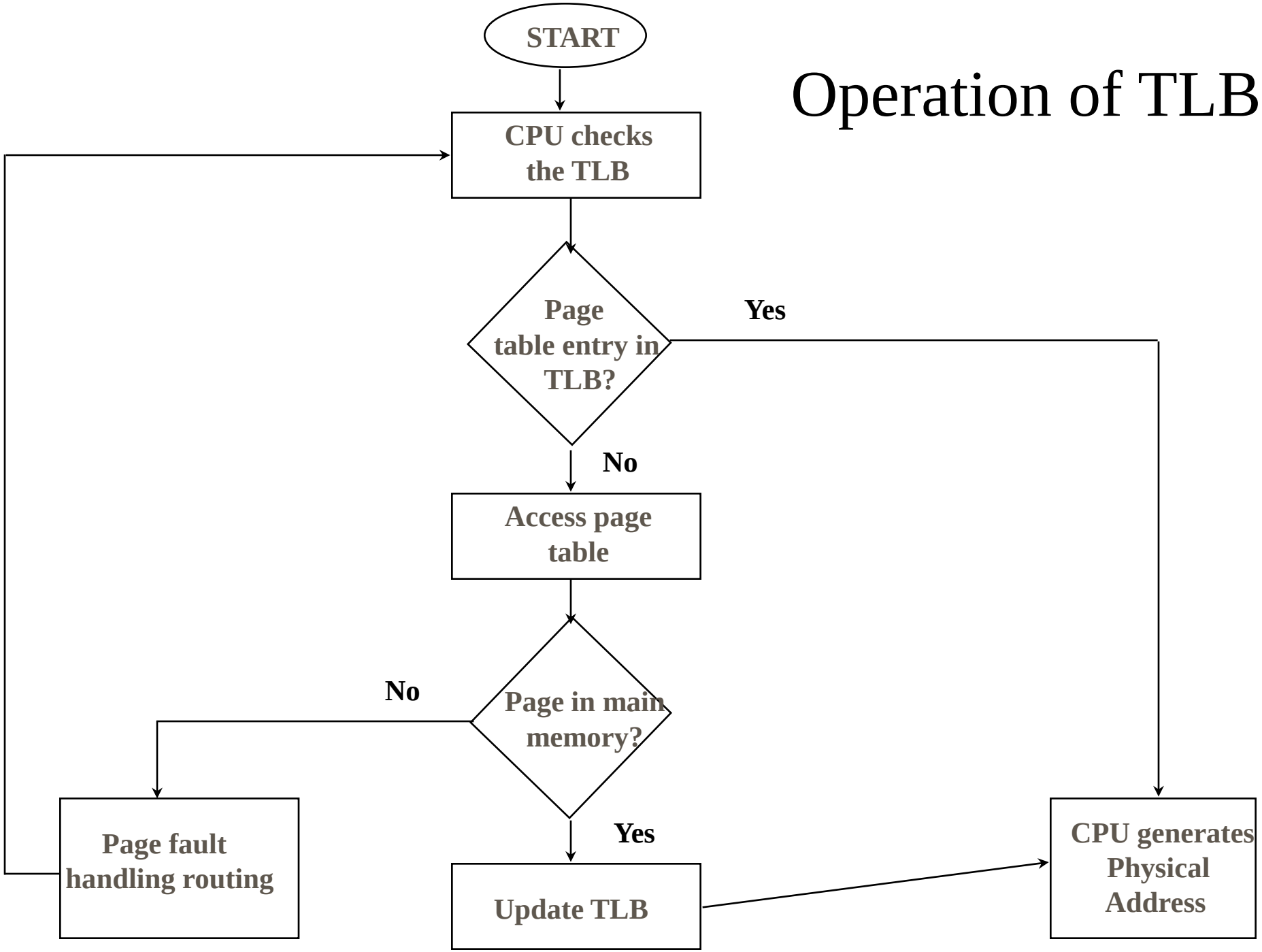
Translation Look-aside Buffer

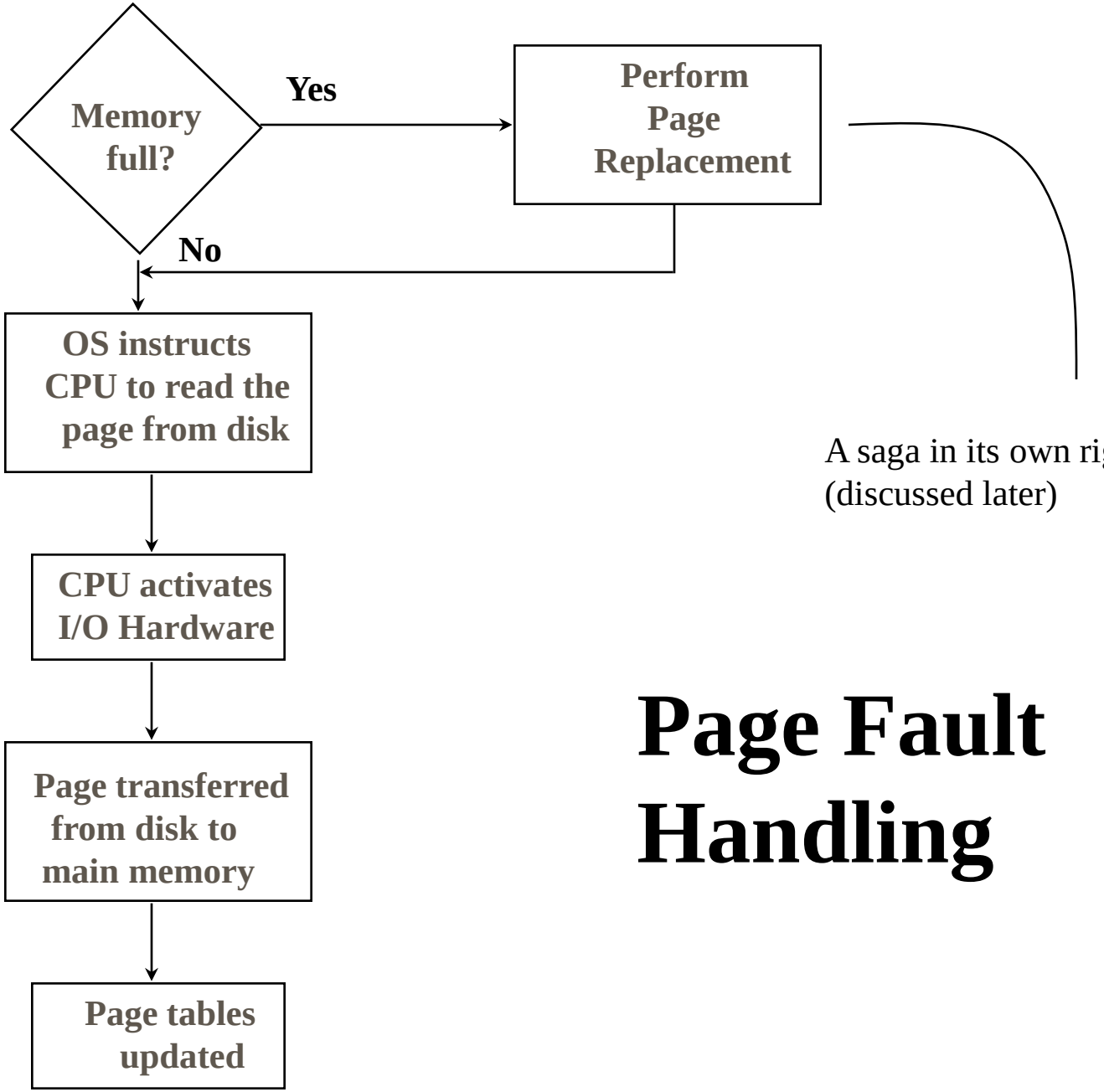
- First checks if page is already in main memory
  - if not in main memory a *page fault* is issued
- The TLB is updated to include the new page entry

Use of a Translation Look-aside Buffer



Operation of TLB





A saga in its own right!
(discussed later)

Page Fault Handling

Page Size Trade-offs

- ✓ Smaller page size, less amount of internal fragmentation (why?)
- ✓ Smaller page size, more pages required per process
- ✓ More pages per process means larger page tables
- ✓ Larger page tables means large portion of page tables in virtual memory
- ✓ Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

Page Size Trade-offs

- Small page size, large number of pages will be found in main memory
- As time goes on during execution, the pages in memory will all contain portions of the process memory situated near the recent references. Page fault rate gets low.
- Increased page size causes pages to contain locations further from any recent reference. Page faults might rise.

Page Size Trade-offs

- ✓ Multiple page sizes provide the flexibility needed to effectively use a TLB
- ✓ Large pages can be used for program instructions
- ✓ Small pages can be used for thread stack
  - Good idea in theory; not used in practice

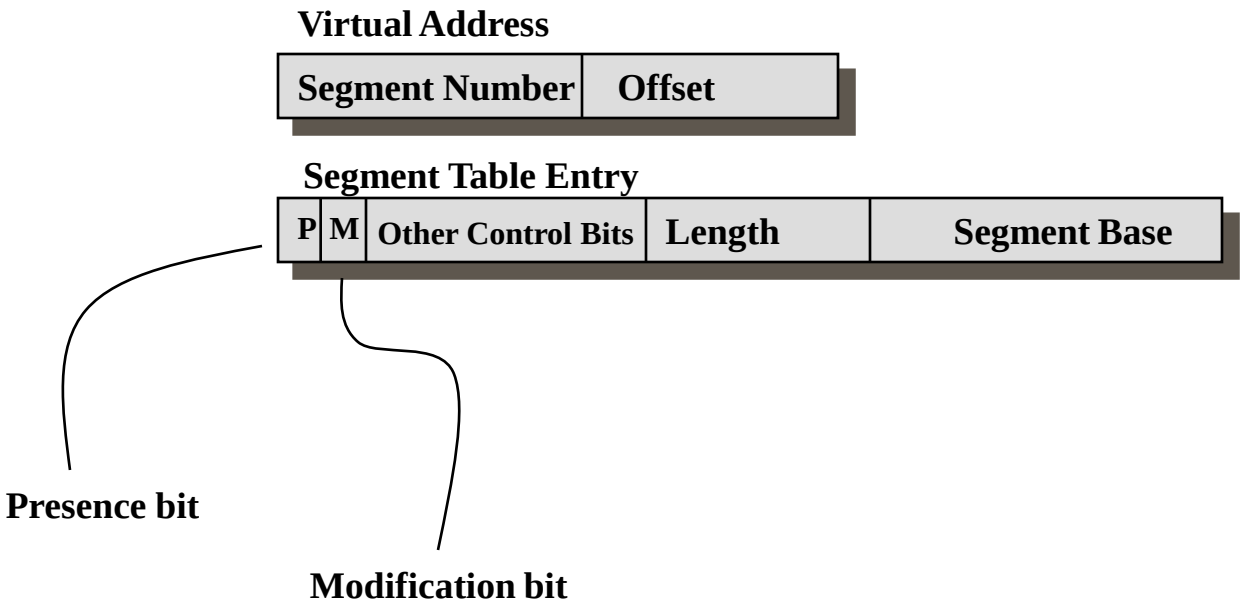
Segmentation

- ✓ Can be dynamic
- ✓ Simplifies handling of growing data structures (why?)
- ✓ Allows programs to be altered and recompiled independently (how?)
- ✓ Used for sharing data among processes
- ✓ Lends itself to protection (how?)

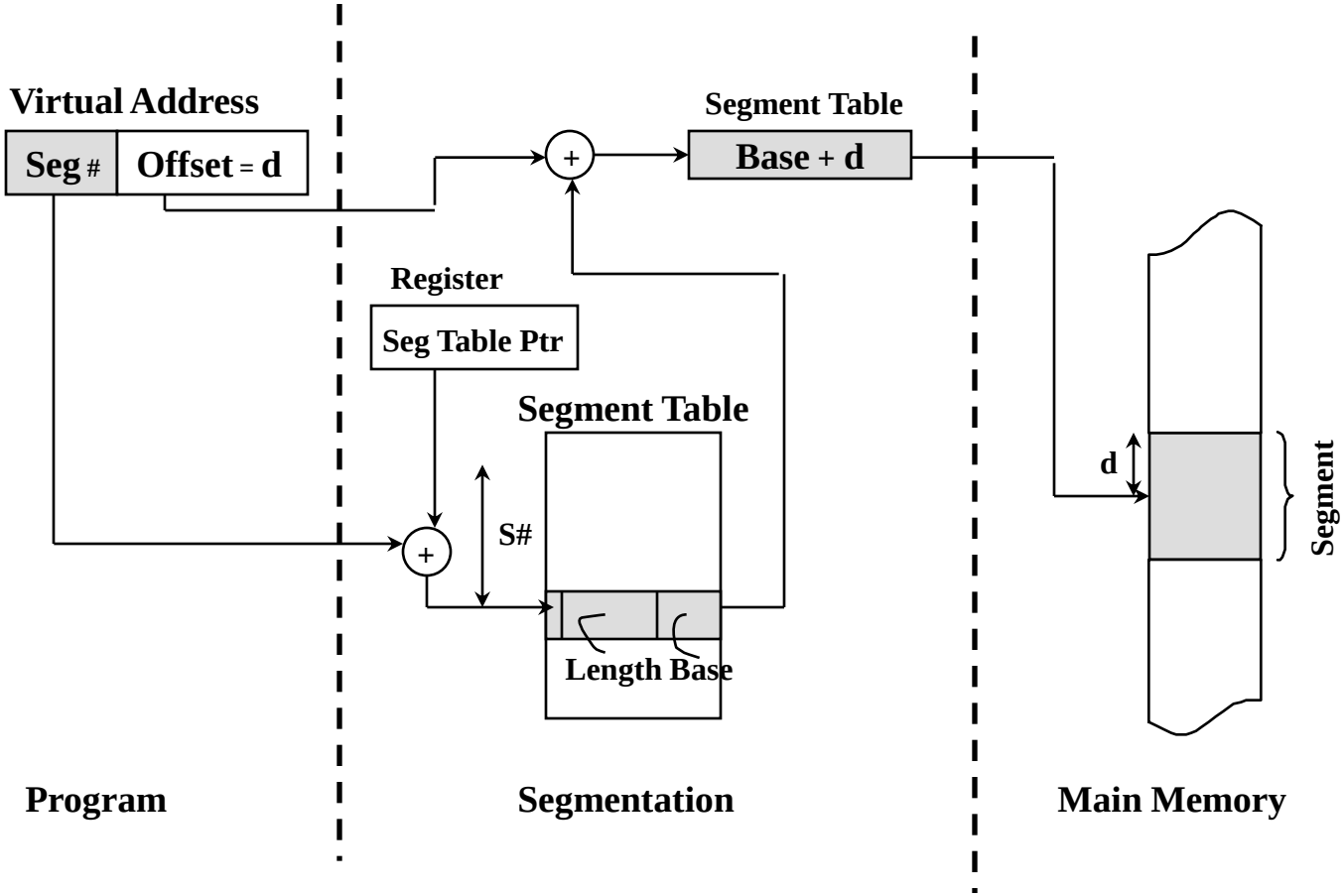
Segment Tables

- Each entry contains the starting address of the corresponding segment in main memory
- Each entry contains the length of the segment
- A bit is needed to determine if segment is already present in main memory
- A bit is needed to determine if the segment has been modified since it was loaded in main memory

Segmentation



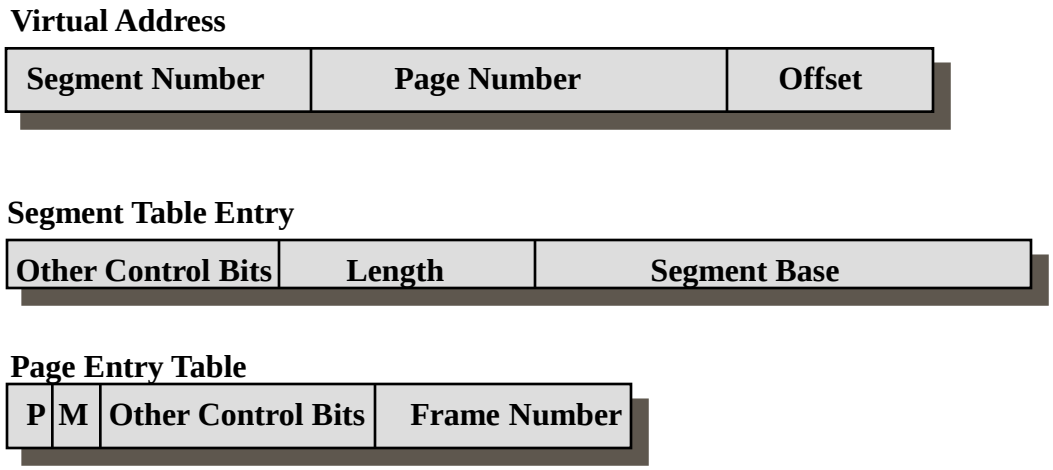
Address Translation in a Segmentation System



Combined Paging and Segmentation

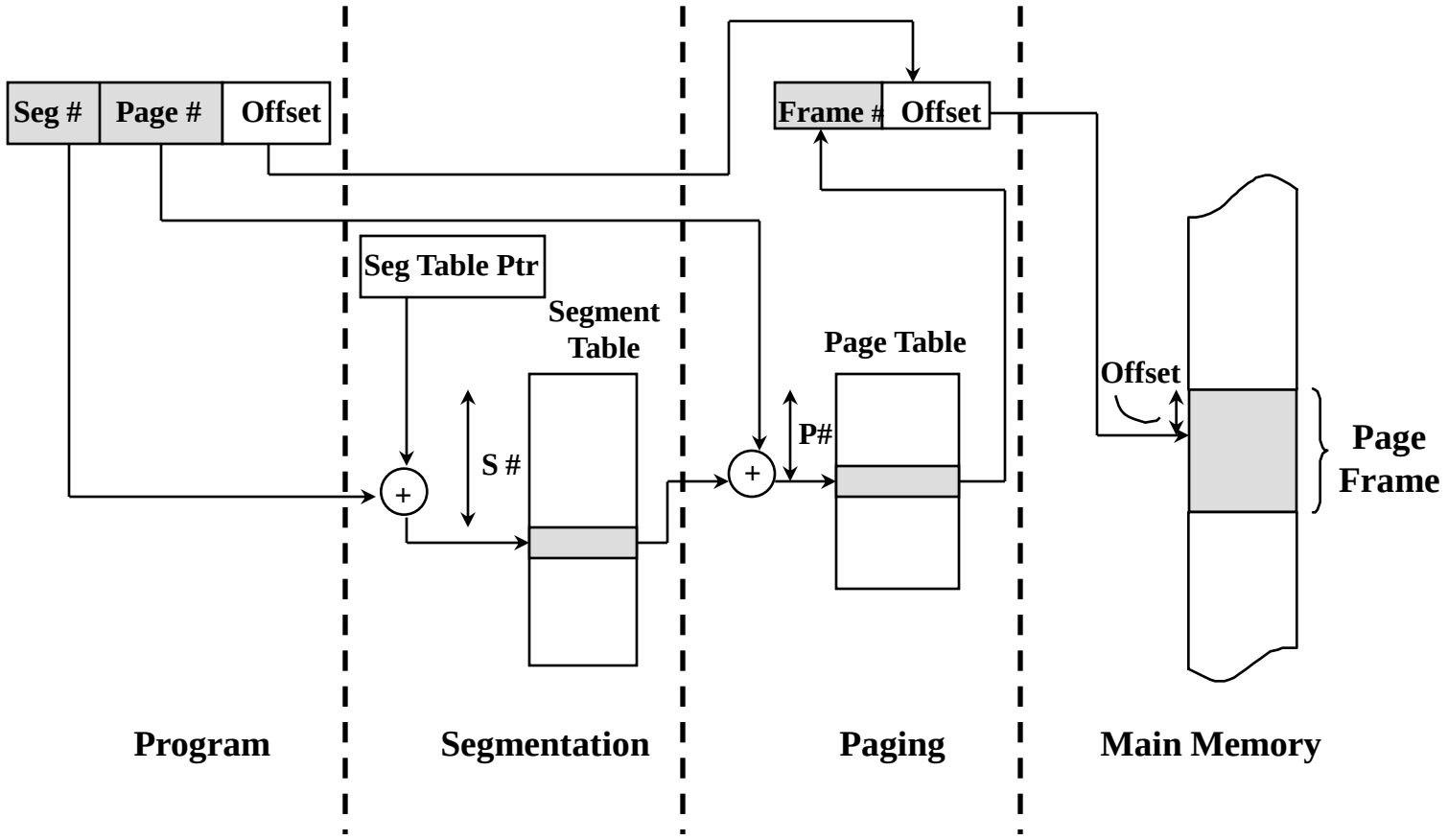
- ✓ Paging is transparent to the programmer
- ✓ Paging eliminates external fragmentation
- ✓ Segmentation is visible to the programmer
- ✓ Segmentation allows for growing data structures, modularity, and support for sharing and protection
- ✓ Each segment is broken into fixed-size pages

Combined Segmentation and Paging



One page table is used for each segment

Address Translation in Segmentation/Paging System



Operating System Policies for Virtual Memory

- *Fetch Policy*
  - determines when a page should be brought into memory
- *Demand paging* only brings pages into main memory when a reference is made to a location on the page
  - many page faults when process first started
- *Prepaging* brings in more pages than needed
  - more efficient to bring in pages that reside contiguously on the disk

Operating System Policies for Virtual Memory

- ✓ *Replacement Policy*
  - deals with the selection of a page in memory to be replaced when a new page is brought in
  - *frame locking* used for frames that cannot be replaced
    - used for the kernel and key control structures of the operating system
    - I/O buffers (why lock them?)

Operating System Policies for Virtual Memory

- Replacement Policy (contd.)
  - Optimal policy selects for replacement the page for which the time to the next reference is the longest
    - impossible to have perfect knowledge of future events

Operating System Policies for Virtual Memory

- ✓ Replacement Policy - *Least Recently Used* (LRU)
  - 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
  - Each page could be tagged with the time of last reference. This requires a great deal of overhead.

Operating System Policies for Virtual Memory

- ✓ Replacement Policy
  - *First-in, first-out* (FIFO): treats page frames allocated to a process as a circular buffer
    - Pages are removed in round-robin style
    - Simplest replacement policy to implement
    - Page that has been in memory the longest is replaced
    - These pages may be needed again very soon (e.g., in program loops!)

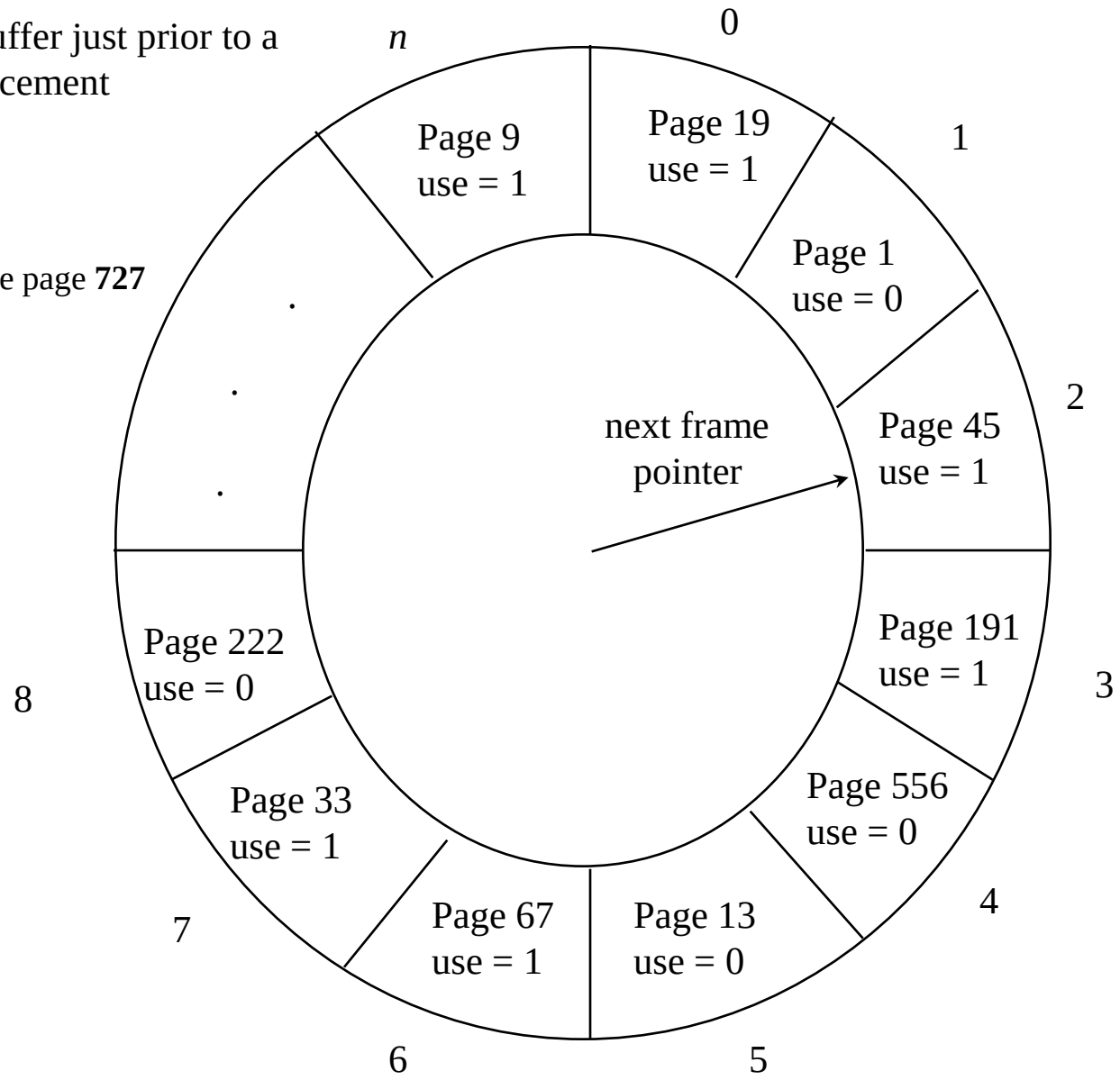
Operating System Policies for Virtual Memory

- ✓ Replacement Policy - *Clock Policy*
  - Additional bit called a use bit
  - When a page is first loaded in memory, the use bit is set to 0
  - When the page is referenced, the use bit is set to 1 (in hardware)
  - When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
  - During the search for replacement, each use bit is changed to 0

Clock Policy

State of buffer just prior to a page replacement

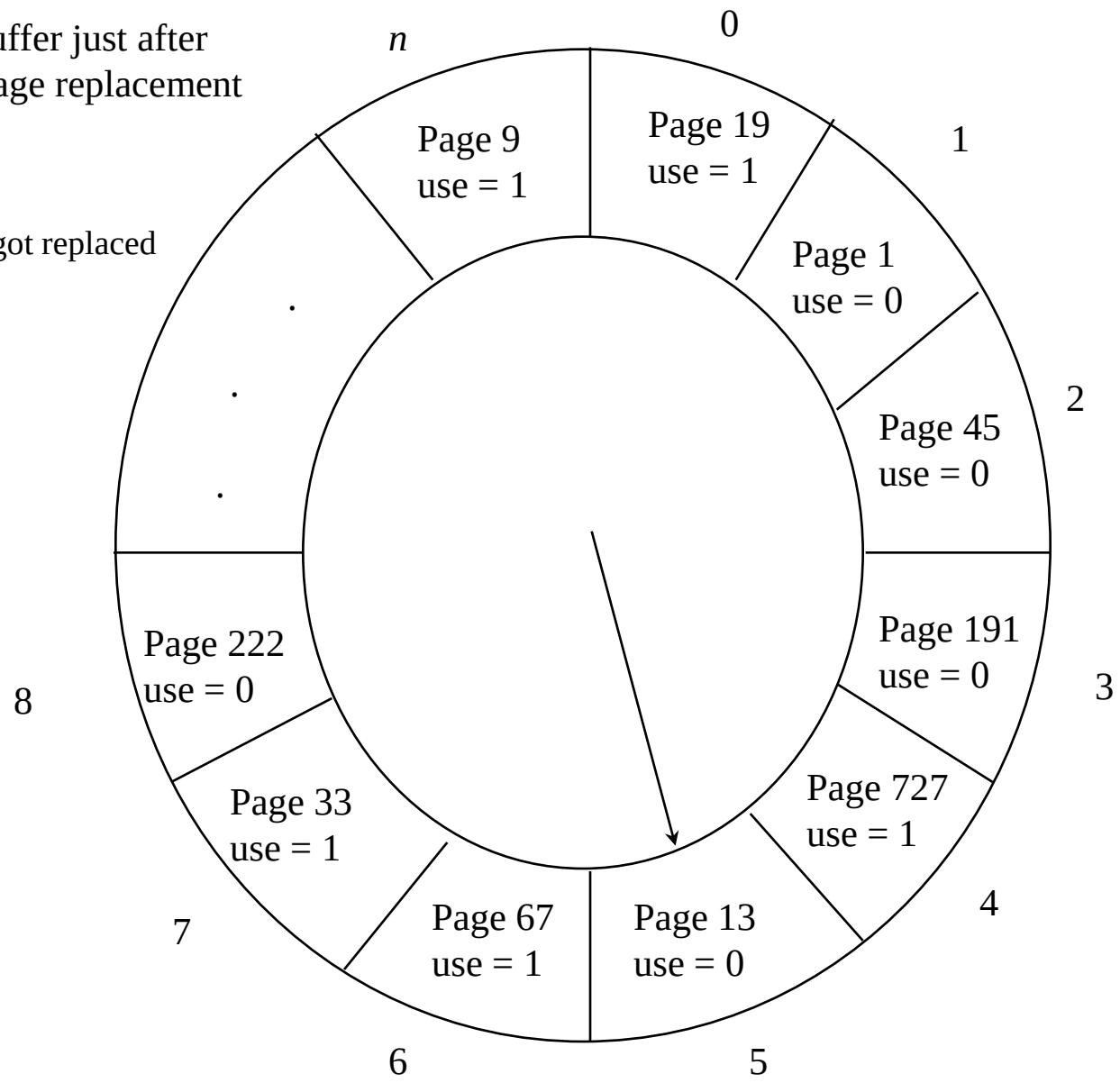
Need to place page **727**



Clock Policy

State of buffer just after
the next page replacement

Page **556** got replaced



Resident Set Size

- ***Fixed-allocation***
  - gives a process a fixed number of pages within which to execute
  - when a page fault occurs, one of the pages of that process must be replaced
- ***Variable-allocation***
  - number of pages allocated to a process varies over the lifetime of the process

Cleaning Policy

- ✓ *Demand cleaning*
  - a page is written out only when it has been selected for replacement
- ✓ *Precleaning*
  - pages are written out in batches when the swap I/O device idles

Cleaning Policy

- Best approach uses page buffering
  - replaceable pages are placed in two lists
    - modified and unmodified
  - Pages in the modified list are periodically written out in batches
  - Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page

Load Control

- Determines the number of processes that will be resident in main memory
- Too few processes, many occasions when all processes will be blocked and processor will be idle
- Too many processes will lead to *thrashing*

Process Suspension

- Needed when there are too many page faults because too many processes are active & their page allotments get small due to memory contention; suspended process is swapped out
- Policies:
  - Lowest priority process
  - Faulting process
    - this process does not have its working set in main memory so it will be blocked anyway
  - Last process activated
    - this process is least likely to have its working set resident

Process Suspension

- Process with smallest resident set
  - this process requires the least future effort to reload
- Largest process
  - obtains the most free frames
- Process with the largest remaining execution window

UNIX and Solaris Memory Management

- ✓ Process memory: Paging system
- ✓ Kernel memory: special memory allocator

UNIX and Solaris Memory Management

- ✓ Data Structures
  - Page table - one per process
  - Disk block descriptor - describes the disk copy of the virtual page (where in the swap device the virtual page lives)
  - Page frame data table - describes each frame of real memory (free/occupied, info needed to swap page out of the frame)
  - Swap-use table - one for each swap device (# of page table entries that point to swap device)

UNIX and Solaris Memory Management

- ✓ Page Replacement
  - refinement of the clock policy known as the *two-handed clock algorithm* (Read!)
- ✓ Kernel Memory Allocator
  - most blocks are smaller than a typical page size, hence paging would be inefficient for the frequently used tables in the kernel

Windows Memory Management

- ✓ Each process sees a separate 2 Gbyte of user space
- ✓ All processes share the same system (OS) space
- ✓ A page can have different states:
  - *available:* pages not currently used
  - *reserved:* for a process but does not count against the process's memory quota until actually used; no space reserved on the swap device
  - *committed:* pages actually used by the process and space committed on swap device