Virtual Memory

KROPP, YOUSSFI, STALLINGS

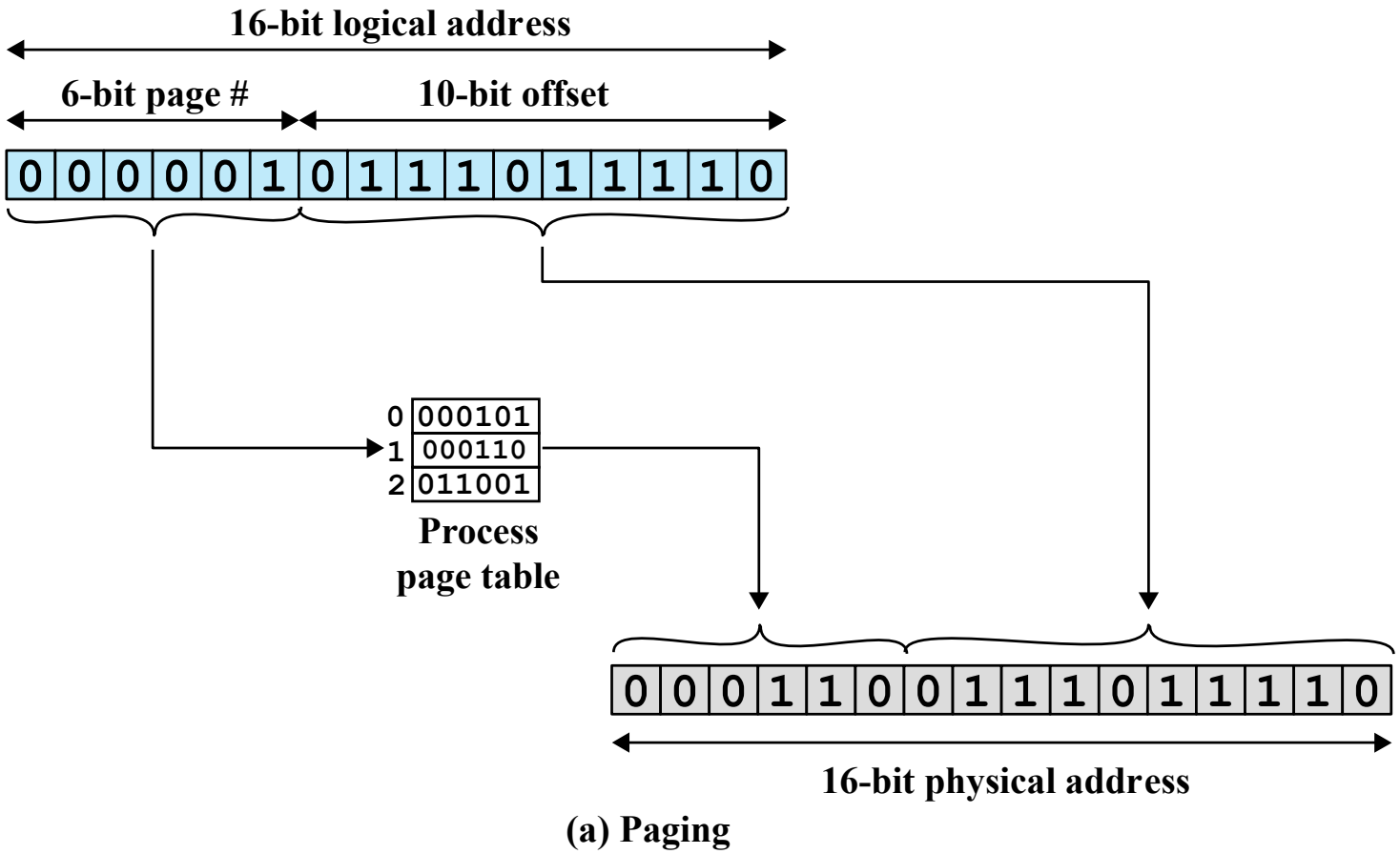
Virtual Memory Objectives

- Define virtual memory
- Hardware to support virtual memory
- OS structures and mechanisms to implement virtual memory

Review

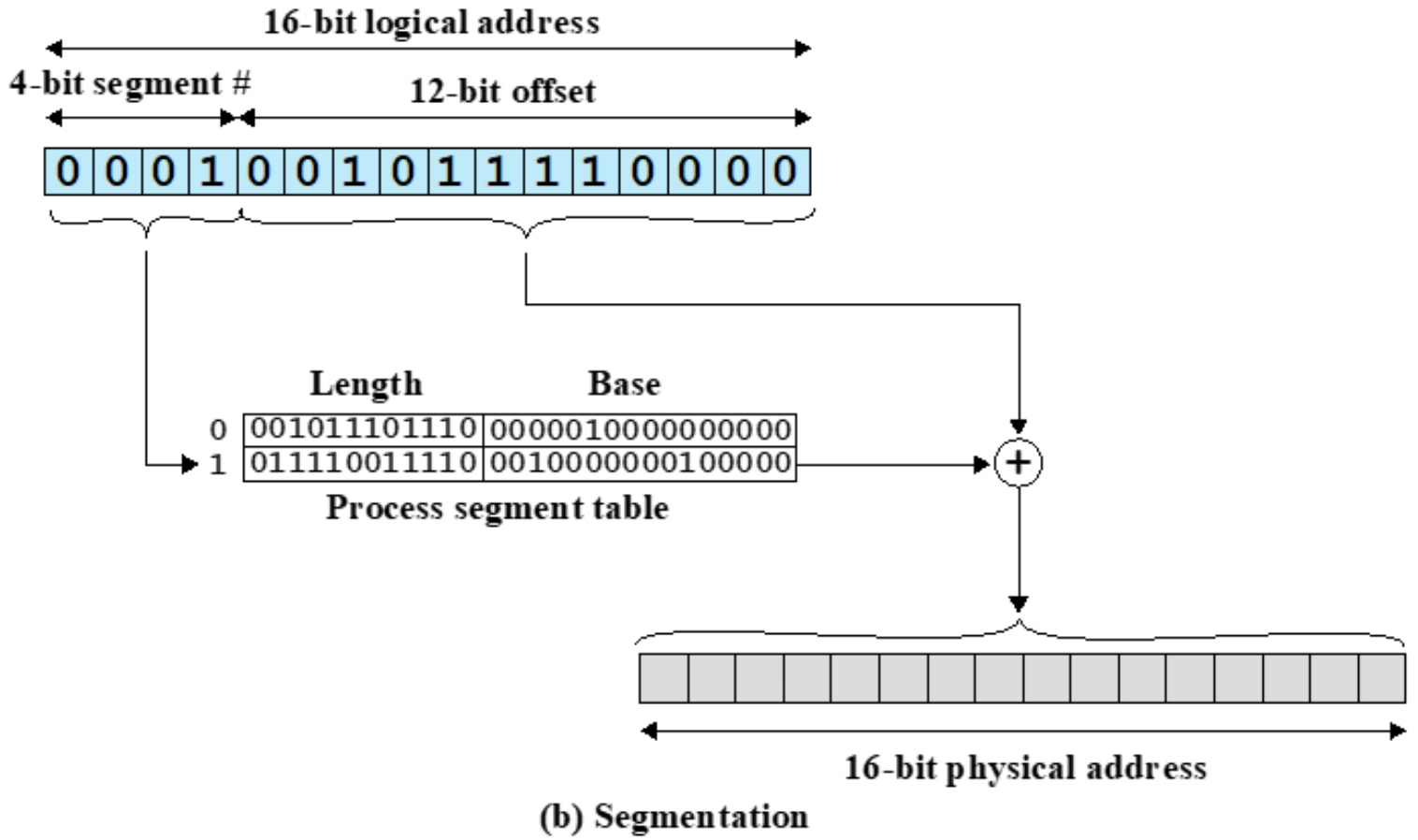
- Last time, we found that there are two approaches to improving simple static and dynamic partitioning schemes
- Paging breaks memory into, small, fixed-width frames to store process memory non-contiguously
- Segmentation breaks memory into variable sized segments stored non-contiguously in memory

Paging Translation Review



(a) Paging

Segmentation Translation Review



(b) Segmentation

Segmentation Translation Example

- Consider a simple segmentation system that has the following table:

Segment number	Starting Address	Length (bytes)
0	660	248
1	1752	422
2	222	198
3	996	604

- For each of the following logical addresses (segment number, offset) determine the physical address or indicate if a segment fault occurs
  - 0, 198
  - 2, 156
  - 1, 530
  - 3, 444
  - 0, 222

Beyond Paging and Segmentation

- Principles of paging and segmentation
  1. All memory references are logical addresses that are dynamically translated into physical addresses at run time
  2. A process may be broken up into several pieces that don’t need to be contiguous in main memory during execution
- Next logical step: if these two characteristics are present, it is not necessary that all the pages or segments of a process be in main memory during execution

Beyond Paging and Segmentation

- This system is *virtual memory*, which combines:
  - Paging
  - Segmentation
  - Splitting between main & secondary memory
- Opposed to *real memory* which exclusively refers to primary memory
- Following slides: how can we accomplish that?

How does this work in practice?

1. Operating system brings into main memory a few *pieces* of the program at a time (called a resident set)
2. Process executes. If memory outside of resident set is needed, jump to step 3
3. Interrupt generated and OS places the process in a blocked state
4. Load piece of data
5. Interrupt again to put process back in ready state
6. Jump to step 2

pieces: pages or segments

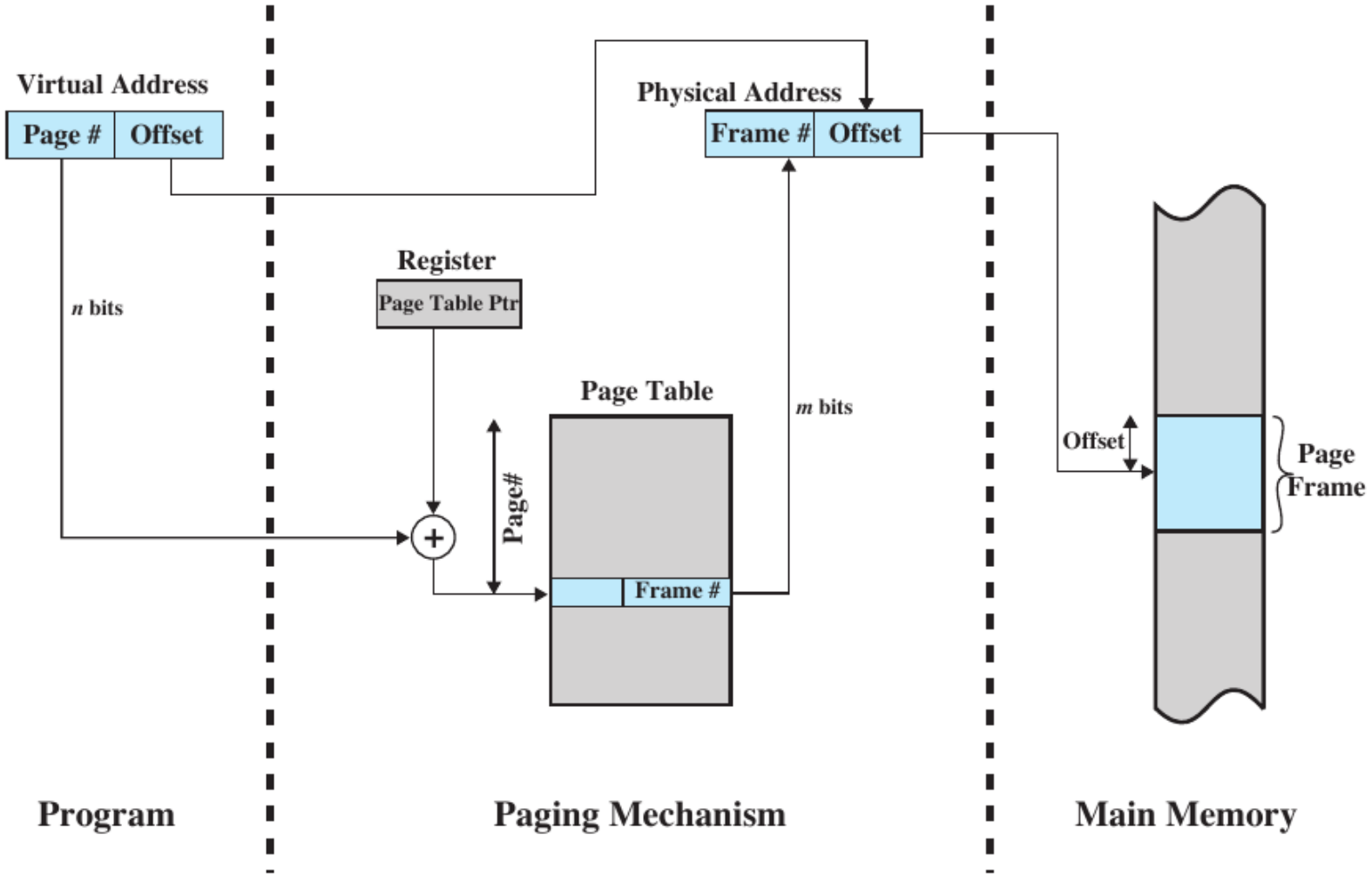
Implication of Virtual Memory

- For now, let's assume the cost of the swapping routine is negligible
- Positive implications:
  - More processes may be maintained in memory
    - Load only pieces of memory that are needed
    - With more processes in memory, more likely one of them is in ready state
  - A process may be larger than all of memory

Support for Virtual Memory

- For virtual memory to be practical and effective, we need:
  - Hardware for paging and segmentation
  - Operating system needs to have efficient algorithms to manage movement of pages and segments into and out of main memory

Access Translation in a Paging System

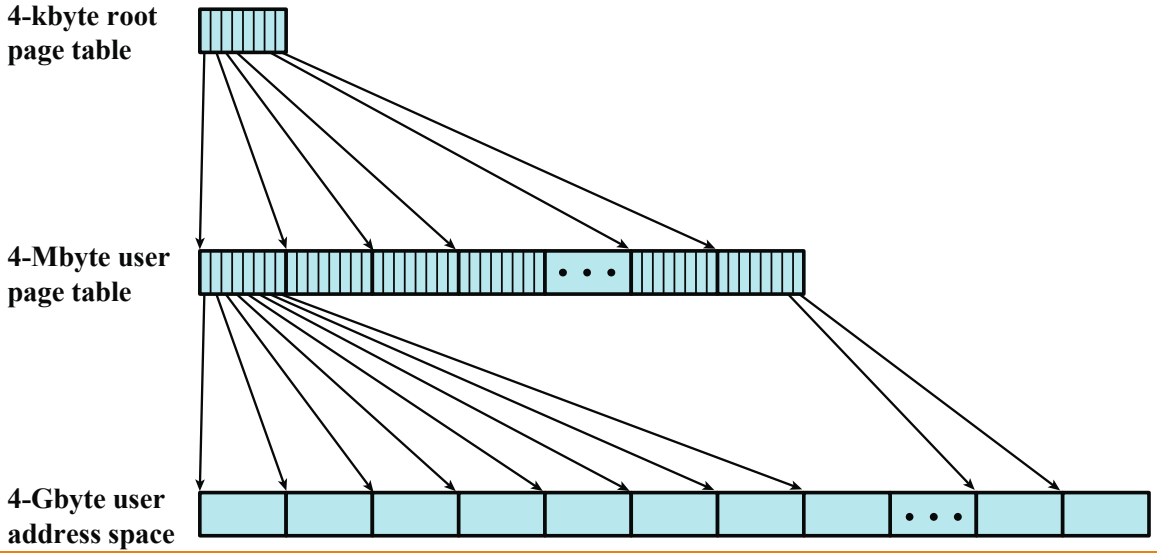


Where do you keep the page table?

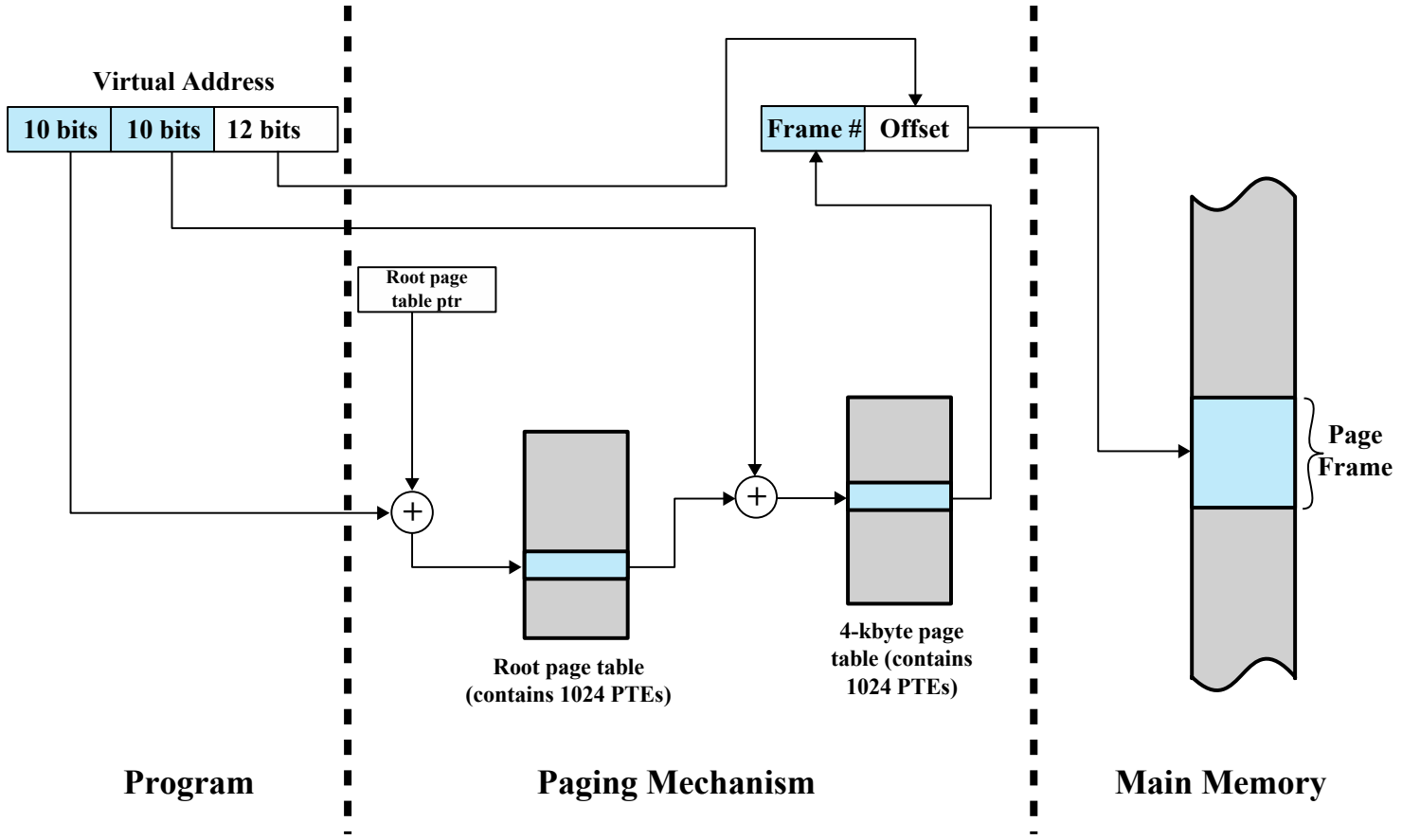
- Do we store our page table in real or virtual memory?
- Let's use the an example to illustrate the point
- The Virtual Address Extension (VAX) is one real-world implementation of virtual memory
  - Each process has a max of 2^{31} (2GB) of virtual memory
  - Break down each 2GB into 2^{9} 512-byte pages
  - Each process would need max 2^{22} (4MB) page table entries
  - This would take up a lot of real memory!
- So we typically store the page table in virtual memory as well

Who pages the page table?

- If process page tables are themselves in virtual memory, then how do we look up their location?
- Common to have a hierarchical scheme
  - Upper portion in real memory referencing the pages storing the page table



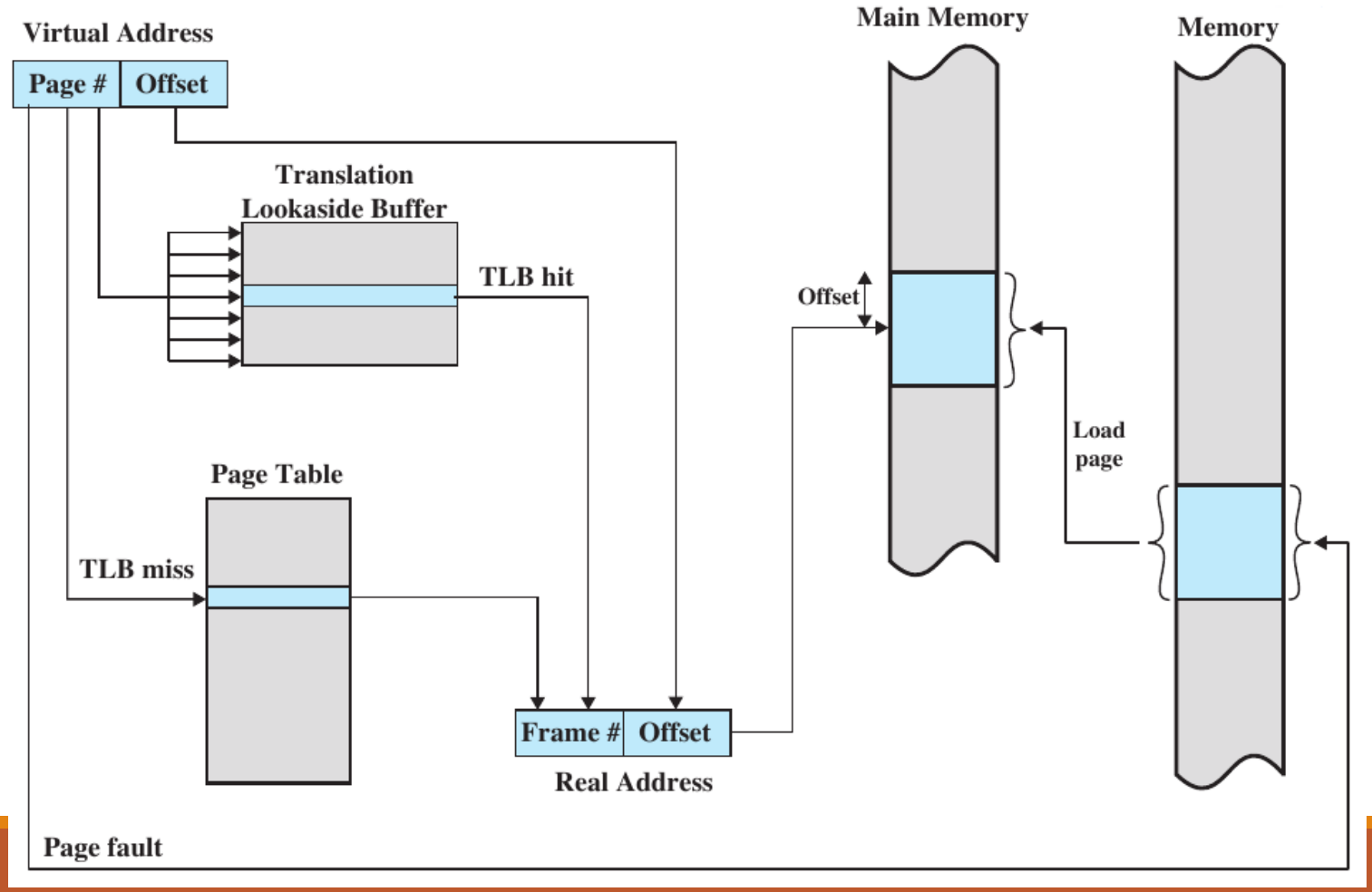
Translation in a two-level paging scheme



Translation Lookaside Buffer (TLB)

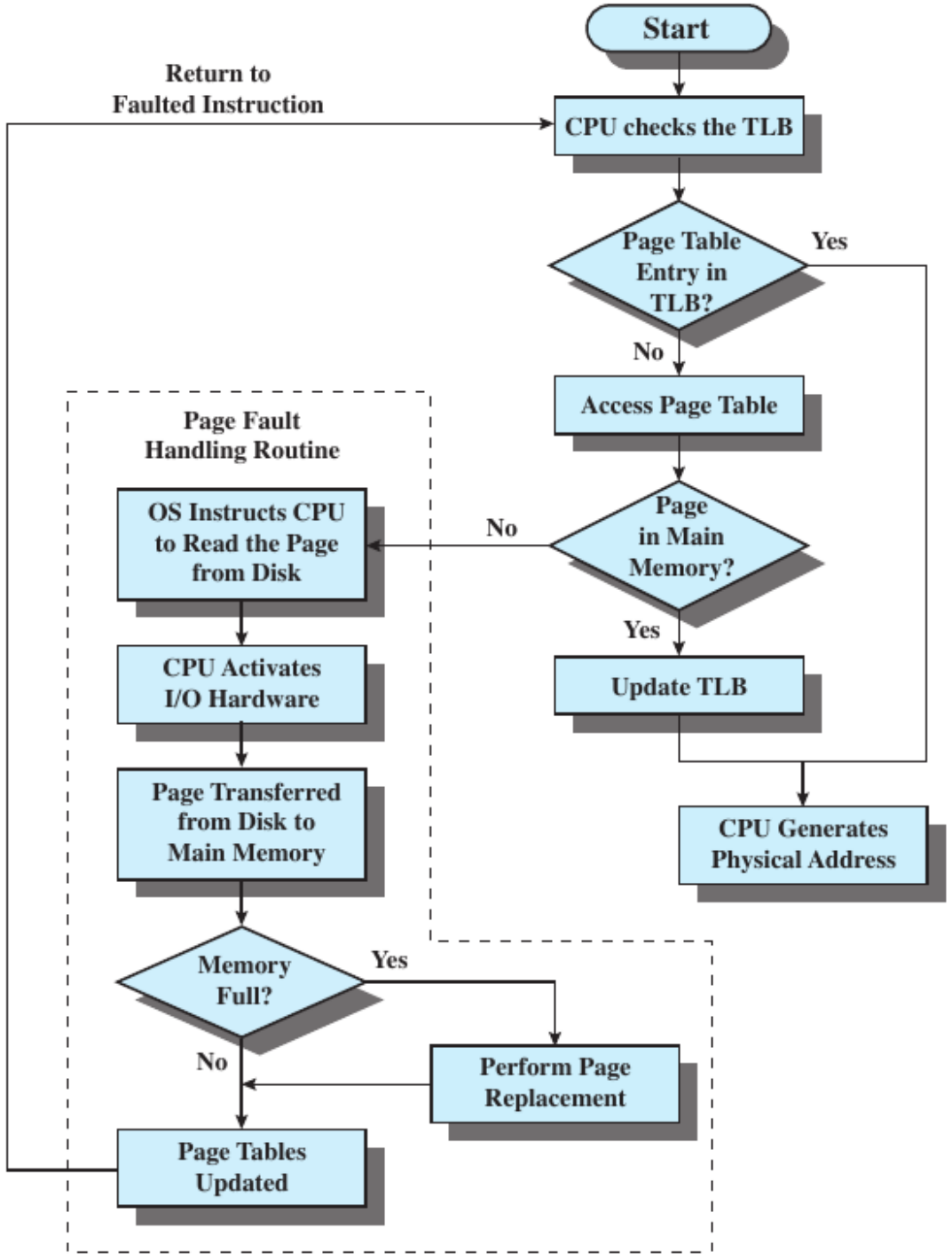
- Each virtual memory reference can cause two physical memory accesses:
  - one to fetch the page table entry
  - one to fetch the data
- To overcome the effect of doubling the memory access time, most virtual memory schemes make use of a special high-speed cache called a *translation lookaside buffer (TLB)*

Using the translation lookaside buffer



Hit: The page you're looking for is in the TLB
Miss: The data you're looking for is not in the TLB

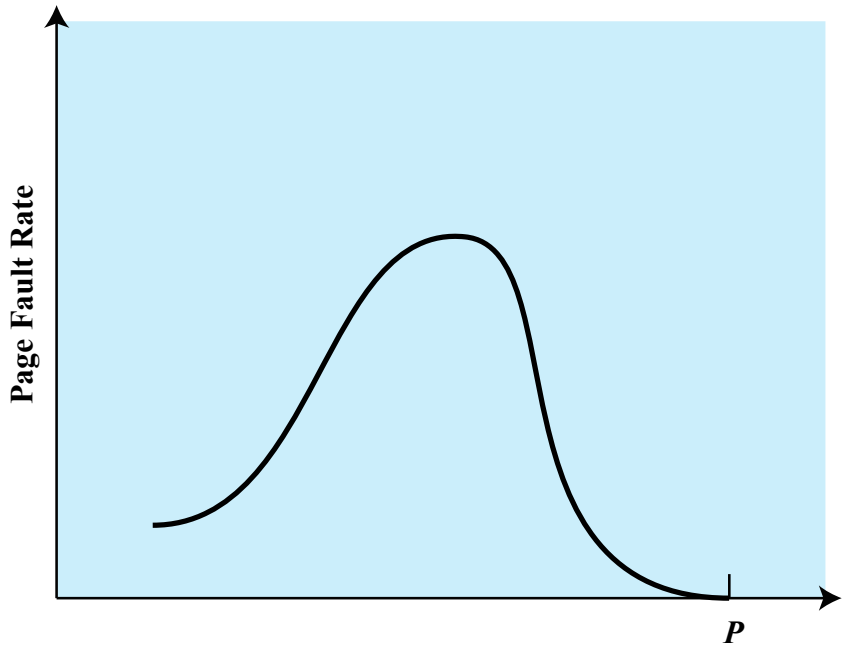
- Operation of the paging and TLB
- Note that if the page faults, the process is put into a blocked state and another process maybe dispatched.
- Most virtual memory references will be in locations in the recently used pages → page table entries in the cache.



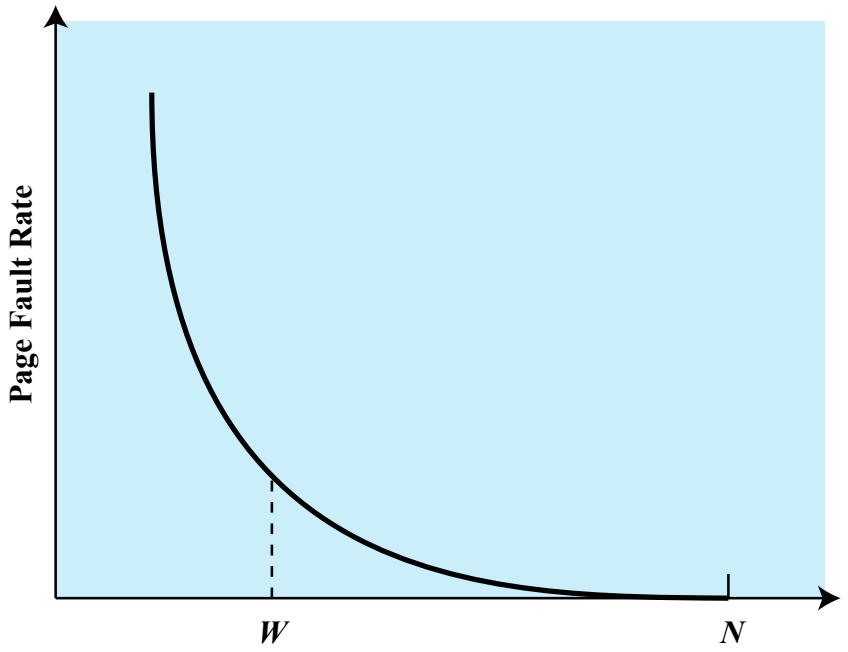
Page Size

- The smaller the page size, the lesser the amount of internal fragmentation
  - However, more pages are required per process
  - More pages per process means larger page tables
  - For large programs in a heavily multiprogrammed environment, this means that some portion of the page tables of active processes must be in virtual memory instead of main memory
    - Double page faults :(
  - The physical characteristics of most secondary memory devices favor a larger page size for more efficient block transfer of data

Paging Behavior



(a) Page Size



(b) Number of Page Frames Allocated

P **= size of entire process**
W **= working set size**
N **= total number of pages in process**

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit words
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Current issues of Page size Design

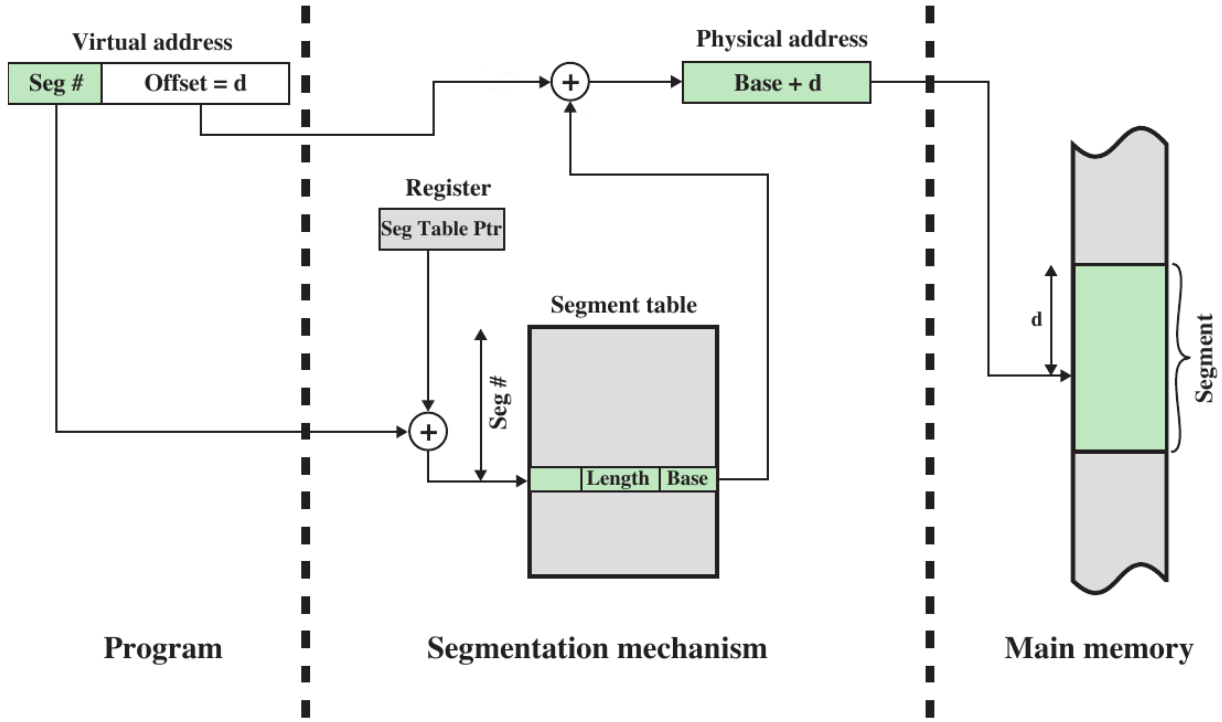
- The design issue of page size is related to the size of physical main memory and program size
- Main memory is getting larger and address space used by applications is also growing → effect on TLB and page table
- Contemporary programming techniques used in large programs tend to decrease the locality of references within a process

Segmentation

- Virtual Memory Implementation
  - Segments of varying size, unlike pages
  - Memory reference consists of: segment number + offset
  - Advantages
    - simplifies handling of growing data structures – no internal fragmentation
    - allows programs to be altered and recompiled independently
    - lends itself to sharing and protection of specific data structures among processes

Segment Organization

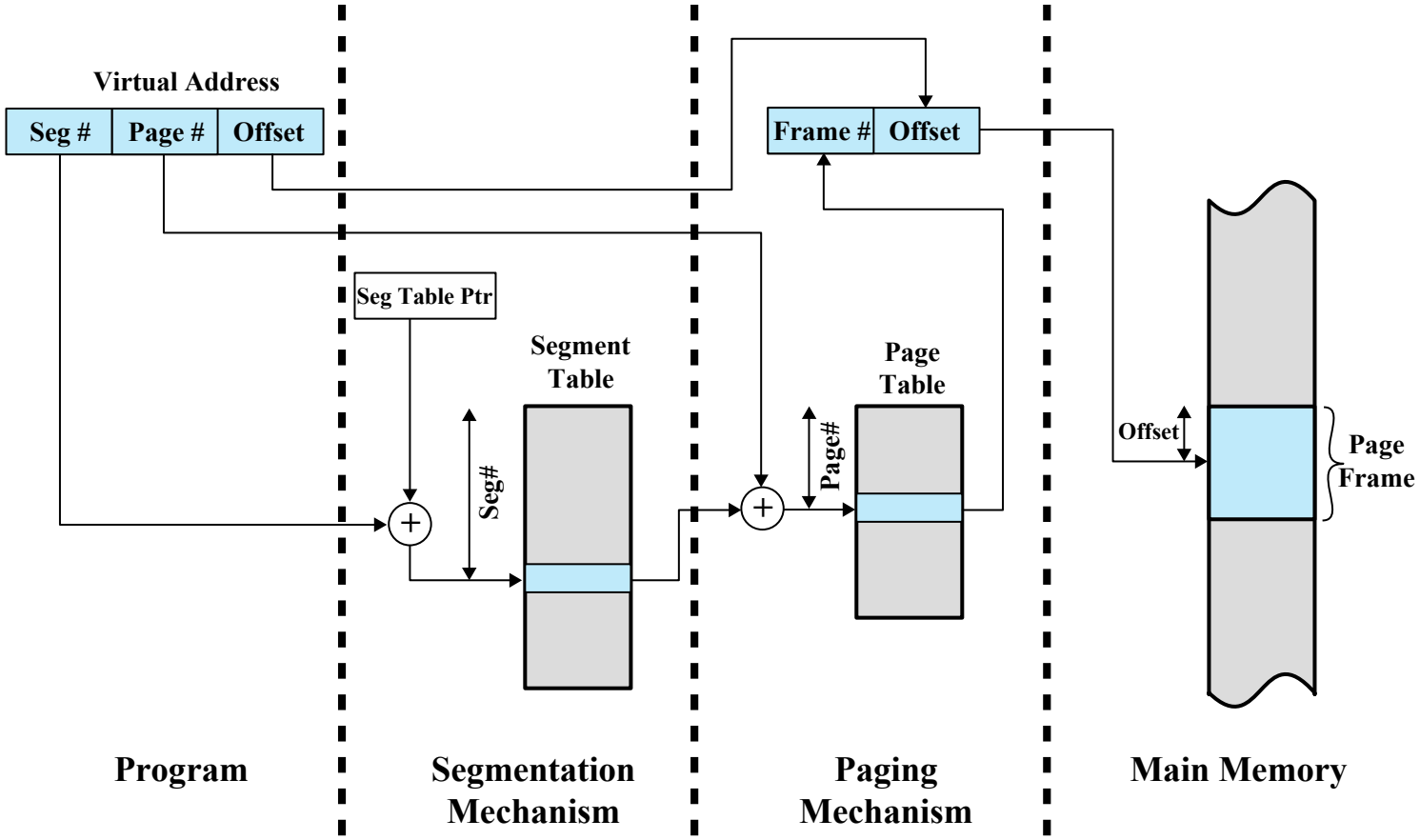
- Segment table entry = starting address of segment + length
- Very similar to paging except segment entry has length limit
- Bits are needed for presence in main memory and modified status



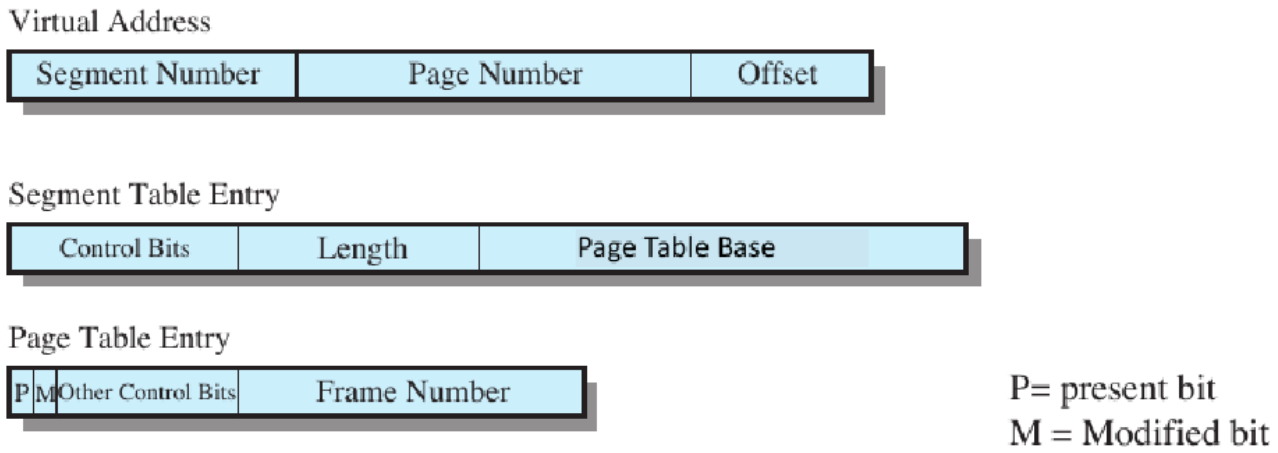
Combined Paging & Segmentation

- Paging
  - Transparent to programmer
  - No external fragmentation
  - Equal size pieces allow for sophisticated management
- Segmentation
  - Simplifies handling of growing data structures
  - No internal fragmentation
- Combined approach
  - Address space is broken into segments, each of which consists of a number of pages
  - Programmer/compiler sees only segments

Address Translation: Segmentation + Paging



Entries Segmentation + Paging



Virtual Memory Software

- Memory management design depends on
  - Whether or not to use virtual memory
  - The use of paging, segmentation, or both
  - Algorithms employed for memory management
- Today,
  - Most if not all operating systems provide virtual memory
  - Pure segmentation is rare
  - Most OS design issues concern paging

Paging Example

- Suppose that the page size is 1,024 bytes.
- Suppose the page table for the process currently executing on the processor looks like the following:

Virtual Page #	Valid Bit	Page frame #
0	1	4
1	1	7
2	0	--
3	1	2
4	0	--
5	1	0

- What physical address, if any, would each of the following virtual addresses correspond to
  - 1,052:
  - 2,221:
  - 5,499:

Paging Example

- Suppose that the page size is 1,024 bytes.
- Suppose the page table for the process currently executing on the processor looks like the following:

Virtual Page #	Valid Bit	Page frame #
0	1	4
1	1	7
2	0	--
3	1	2
4	0	--
5	1	0

- What physical address, if any, would each of the following virtual addresses correspond to
  - 1,052:
    - $1052 = 1024 + 28$ maps to VPN 1 in PFN 7, $(7 \times 1024 + 28 = 7196)$
  - 2,221:
    - $2221 = 2 \times 1024 + 173$ maps to VPN 2, page fault
  - 5,499:
    - $5499 = 5 \times 1024 + 379$ maps to VPN 5 in PFN 0, $(0 \times 1024 + 379 = 379)$