操作系统

第8章 虚拟内存
Virtual Memory

孙承杰
哈工大计算学部

E-mail： sunchengjie@hit.edu.cn
2025年秋季学期

Learning Objectives

- Define virtual memory
- Describe the hardware and control structures that support virtual memory

Outline

- **Locality（局部性）and Virtual Memory**
- **Paging**
  - Page table structure
  - Translation lookaside buffer
  - Page size
- **Segmentation**
- **Combined Paging and Segmentation**
- **Protection and Sharing**
- **Replacement Policy**

You're gonna need a **bigger** boat.

-- *Steven Spielberg ,*
JAWS, 1975

Memory Management

- **Virtual memory(虚拟内存)**
  - **secondary memory** can be **addressed**
  - The **addresses a program** may use to reference memory are **distinguished from** the **addresses the memory system** uses to identify physical storage sites
  - The size of virtual storage is limited
    - by the addressing scheme of the **computer system**
    - by the **amount of secondary memory** available
    - **not by** the actual number of main storage locations
- **Virtual address(虚拟地址)**
  - The address assigned to a location in **virtual memory** to allow that location to be accessed as though it were part of main memory
- **Virtual address space**
  - **virtual storage** assigned to a process
- **Address space**
  - The **range of memory addresses** available to a process
- **Real address**
  - The address of a storage location in **main memory**

Hardware and Control Structures

- **Two characteristics fundamental to memory management:**
  - all memory references are **logical addresses** that are **dynamically translated** into **physical addresses** at run time
  - a process may be broken up into **a number of pieces** that **don't** need to be **contiguously located** in main memory during execution
- **If these two characteristics are present**
  - it is not necessary that all of the pages or segments of a process be in **main memory** **during execution**

Relocation Execution of a Process 1/2

- **OS brings into main memory a few pieces of the program**
  - the term **piece** to refer to either **page** or **segment**
- **Resident set**
  - **portion of process** that is in main memory
- **An interrupt is generated when an address is needed that is not in main memory**
- **OS places the process in a blocking state**

Relocation Execution of a Process 2/2

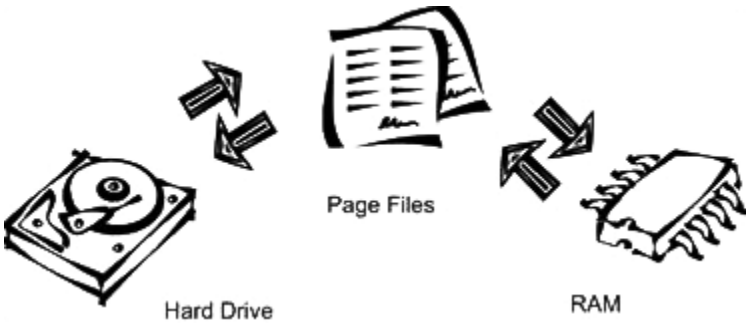
- **Piece of process that contains the logical address is brought into main memory**
  - OS **issues** a disk I/O Read **request**
  - **another process is dispatched to run** while the disk I/O takes place
  - **an interrupt** is issued when disk I/O is **complete**, which causes the OS to place the affected process in the **Ready state**

Two Implications

- **More processes may be maintained in main memory**
  - only load in 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
- **A process may be larger than all of main memory**
  - **Without** the scheme **a programmer** must be **acutely aware** of **how much memory is available**
  - If the program being written is **too large**, the programmer must devise ways to **structure the program into pieces** that can be loaded separately in some sort of overlay strategy
  - OS **automatically loads** pieces of a process into **main memory** as required

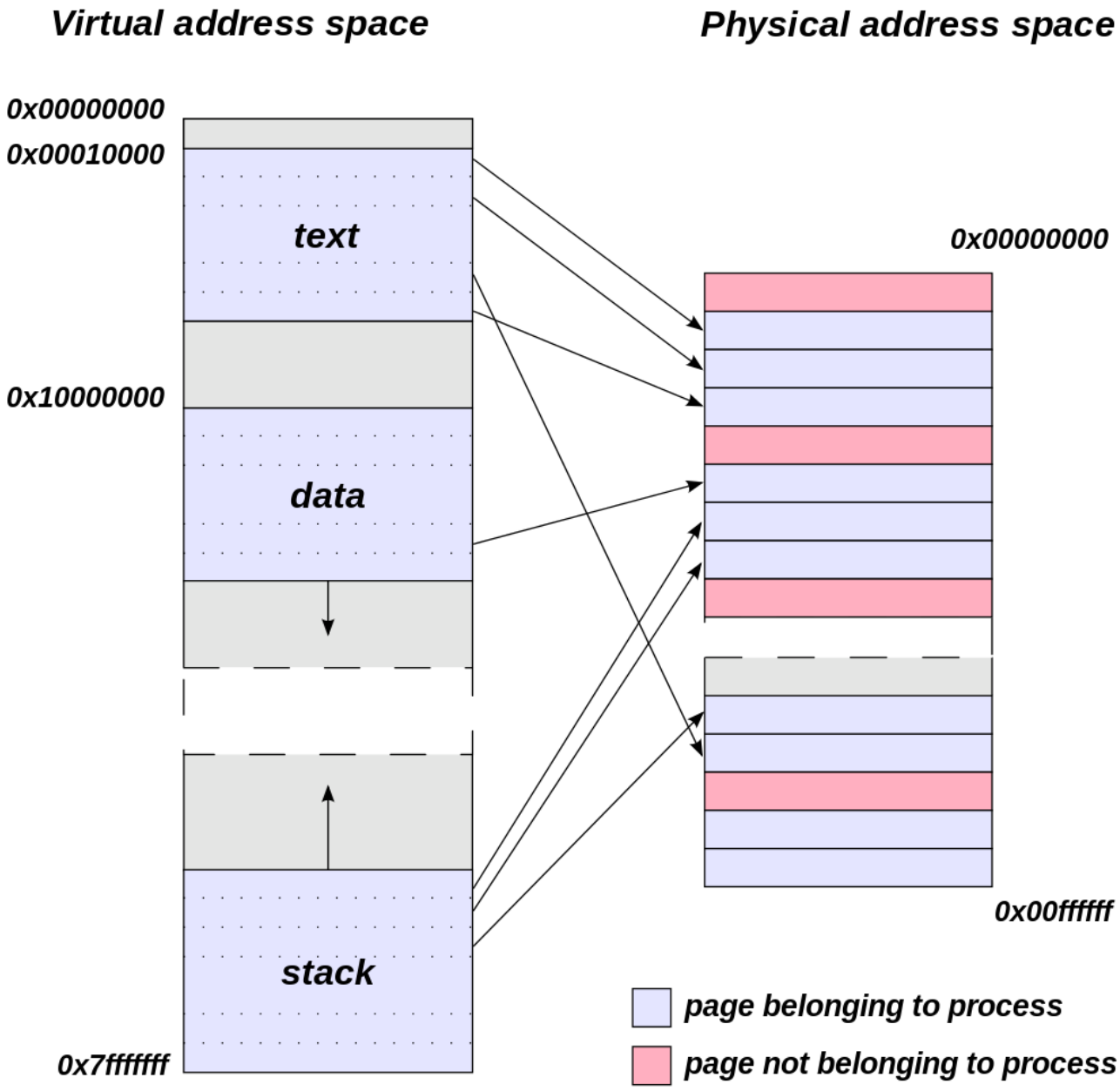
Real and Virtual Memory

- **Real memory**
  - **main** memory
  - the **actual** RAM
  -



- **Virtual memory**
  - memory on **disk**
  - allows for effective multiprogramming
  - relieves the user of tight constraints of main memory

Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
No external fragmentation	No external fragmentation	External fragmentation	External fragmentation
Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a segment table for each process showing the load address and length of each segment	Operating system must maintain a segment table for each process showing the load address and length of each segment
Operating system must maintain a free frame list	Operating system must maintain a free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All the pages of a process must be in main memory for process to run, unless overlays are used	**Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed**	**All the segments of a process must be in main memory for process to run, unless overlays are used**	**Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed**
	Reading a page into main memory may require writing a page out to disk		**Reading a segment into main memory may require writing one or more segments out to disk**



Common method of using paging to create a virtual address space

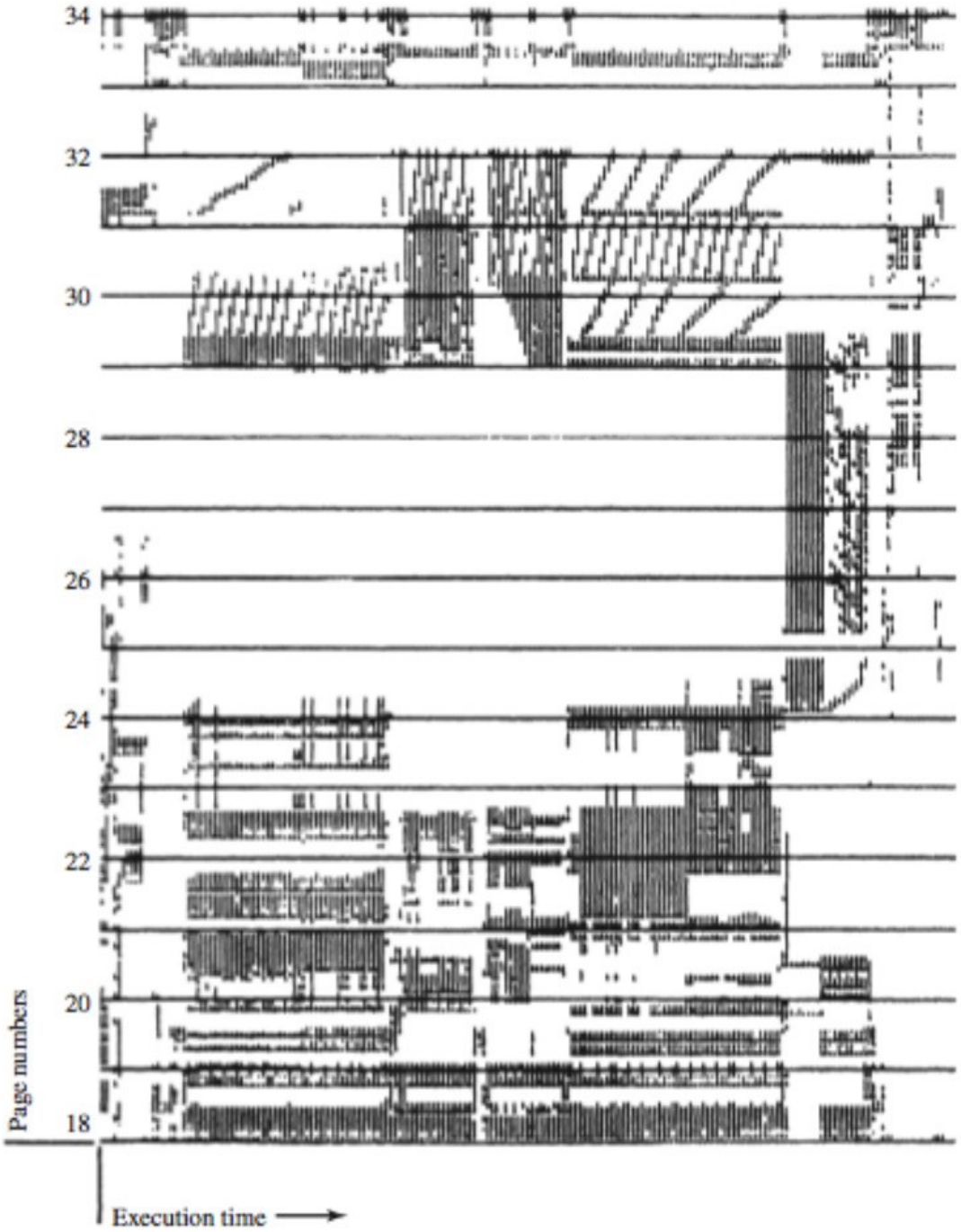
Thrashing

- A state in which the system spends most of its time swapping process pieces rather than executing instructions
- To avoid this, the operating system tries to guess, based on recent history, which pieces are least likely to be used in the near future

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
- Therefore it is possible to make intelligent guesses about which pieces will be needed in the future
- Avoids thrashing

During the lifetime of the process, references are confined to a subset of pages.



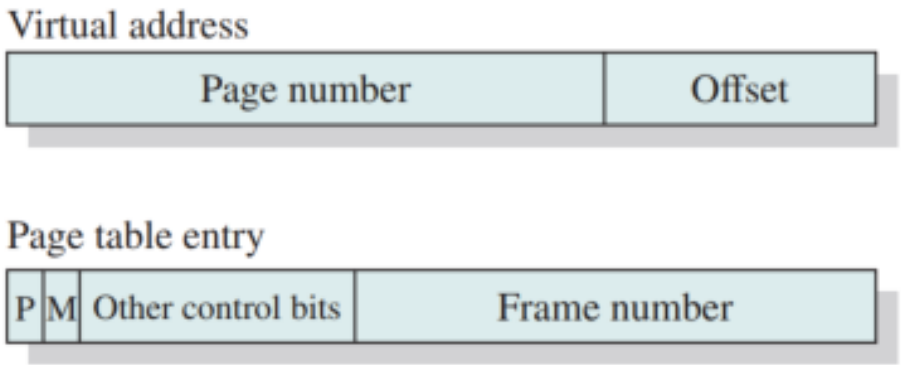
Paging Behavior

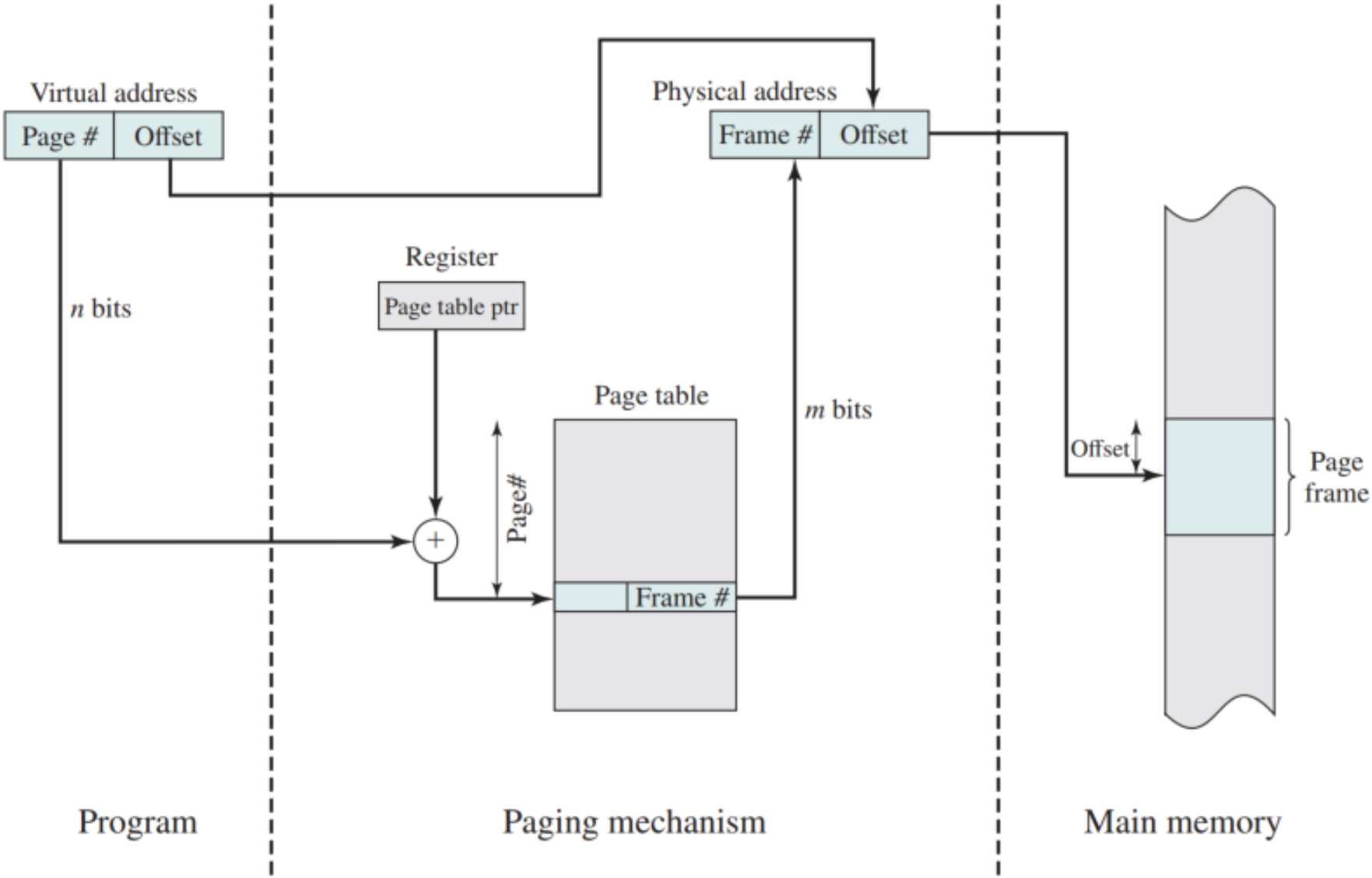
Support Needed for Virtual Memory

- **For virtual memory to be practical and effective**
  - hardware must support paging and segmentation
  - operating system must include software for managing the movement of pages and/or segments between secondary memory and main memory

Paging

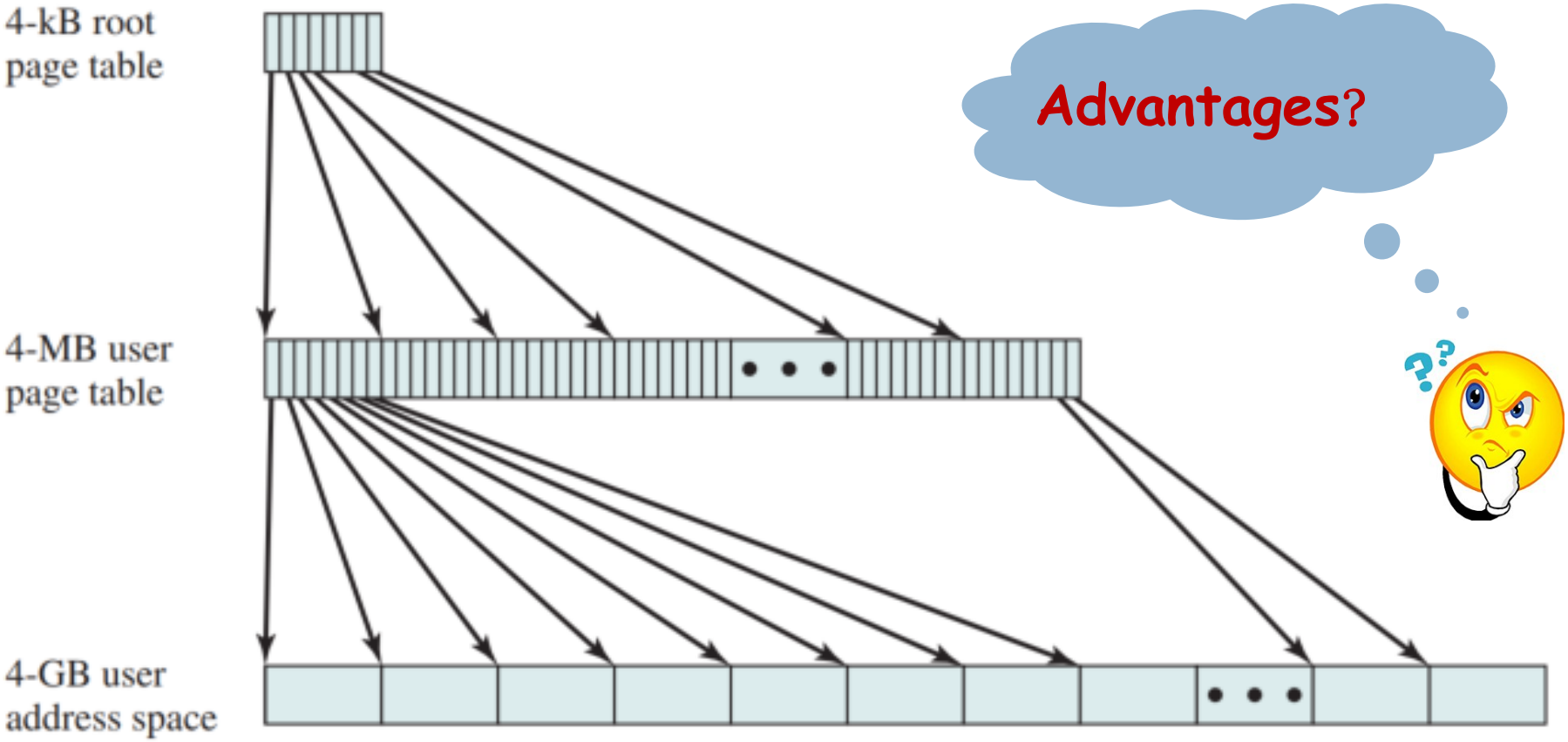
- The term virtual memory is usually associated with systems that employ paging
- Use of paging to achieve virtual memory was first reported for the Atlas computer
- Each process has its own page table

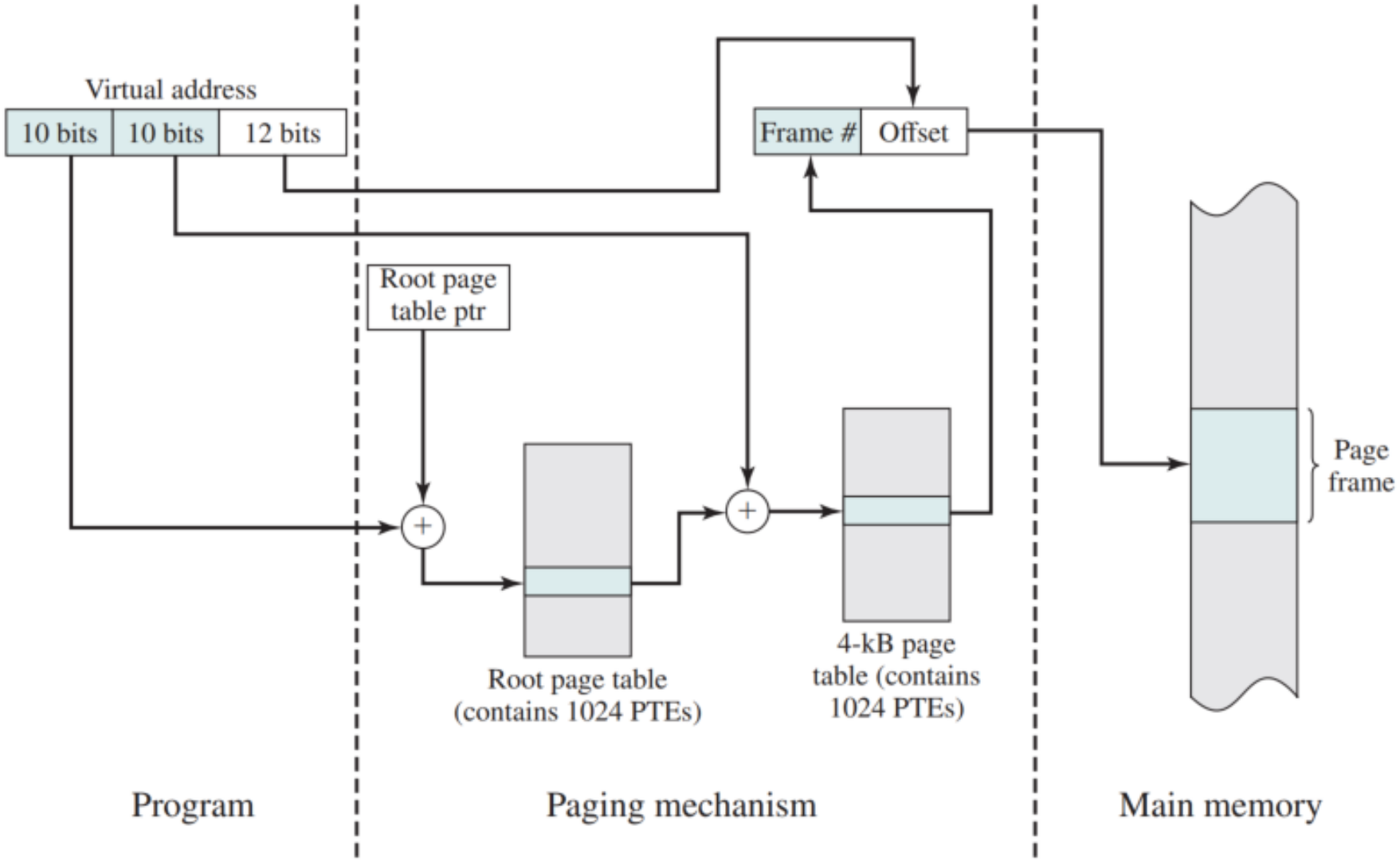




Address Translation in a Paging System

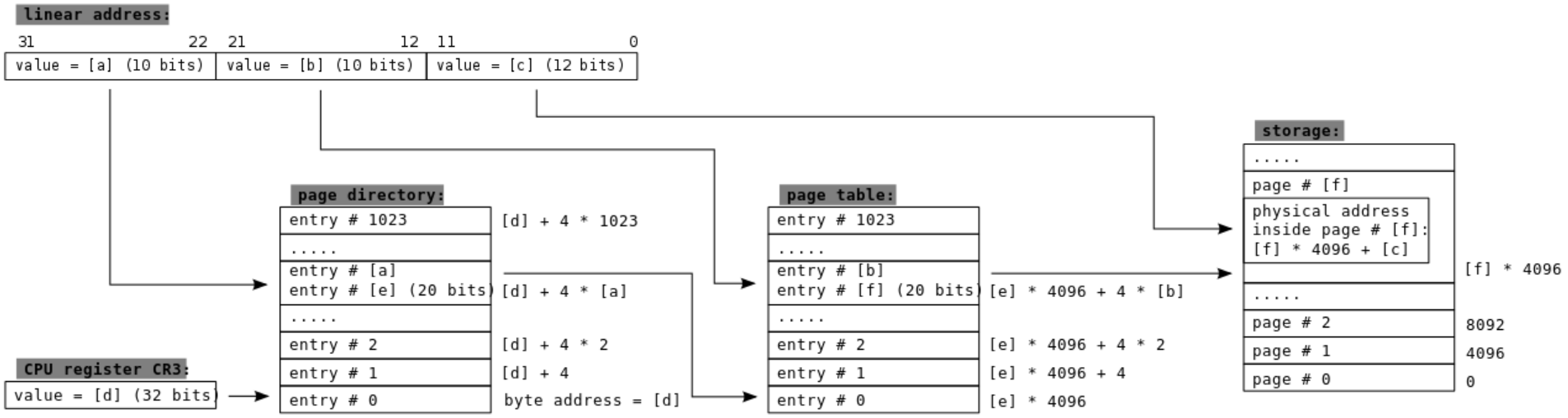
A Two-Level Hierarchical Page Table





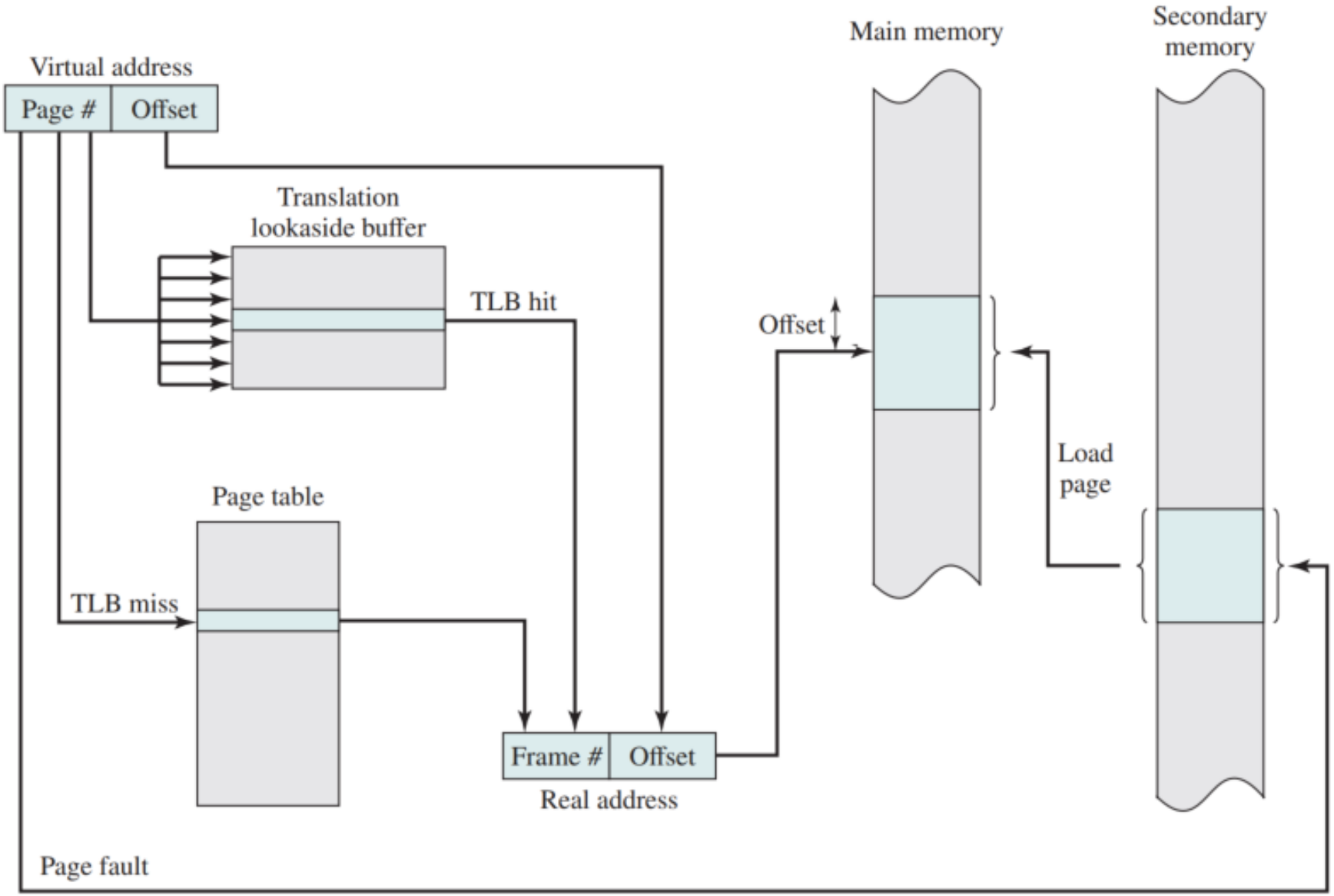
Address Translation in a Two-Level Paging System

Paging with page size of 4K -- Intel 80386



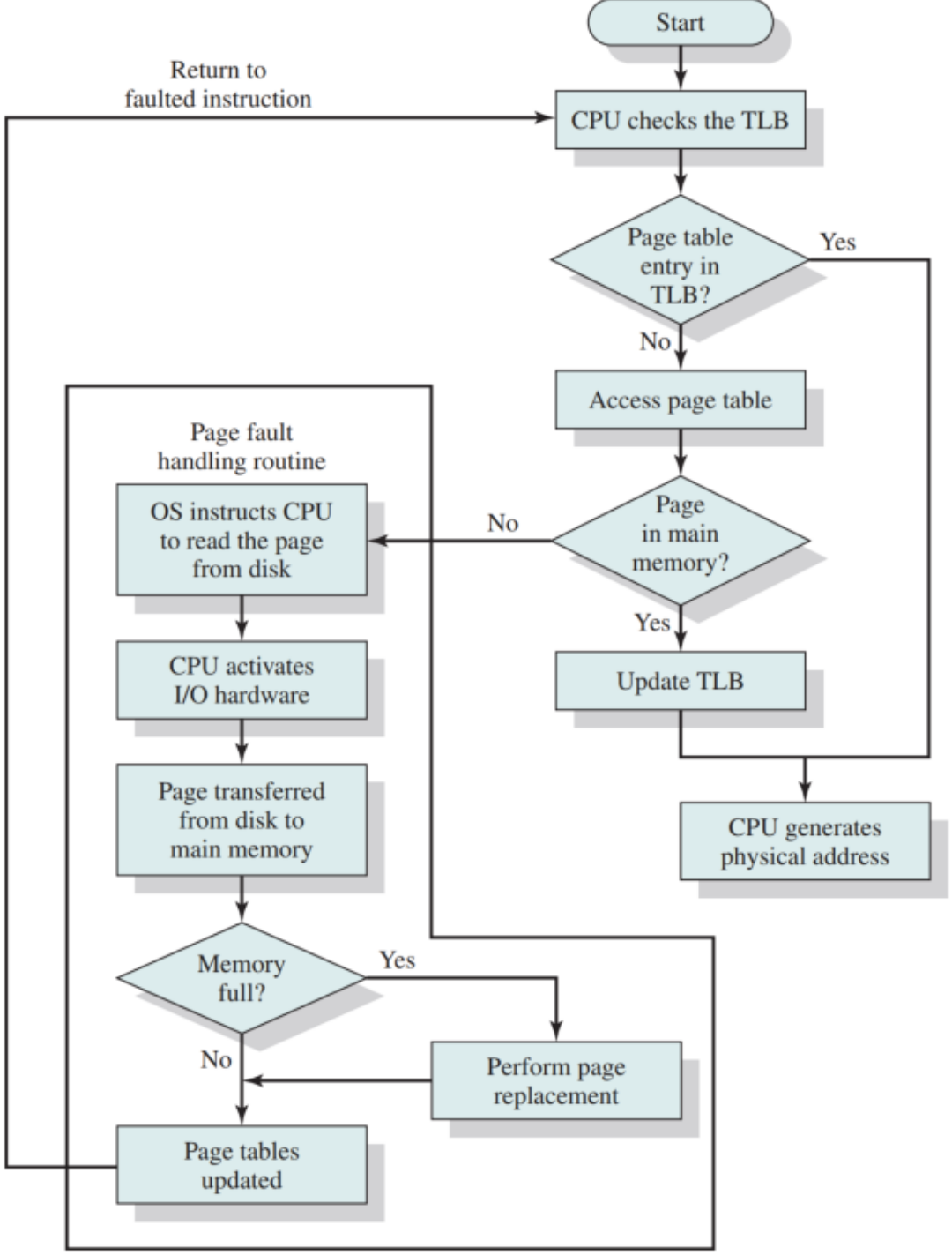
Translation Lookaside Buffer (TLB)

- **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**



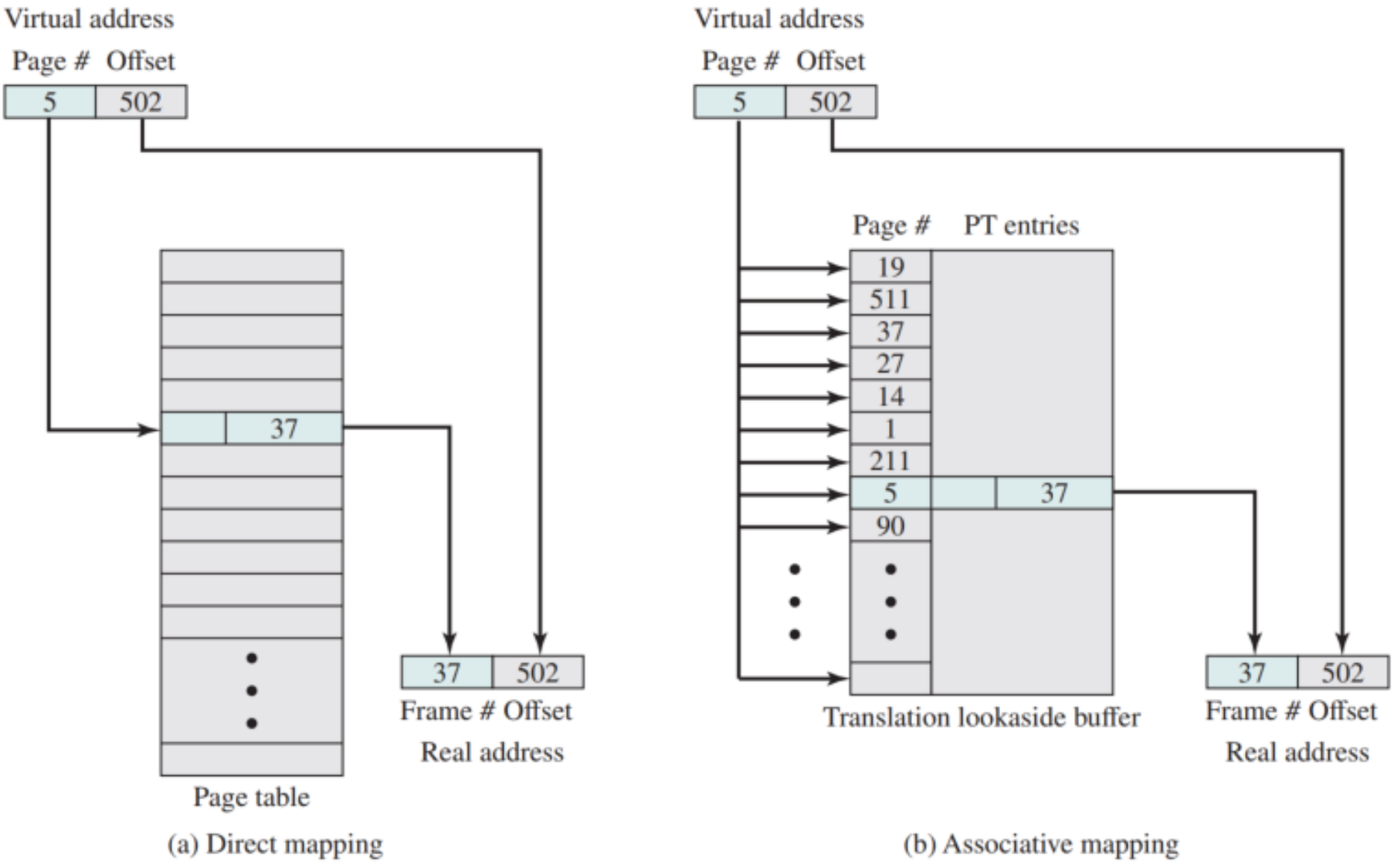
Use of a Translation Lookaside Buffer

Operation of Paging and Translation Lookaside Buffer (TLB)

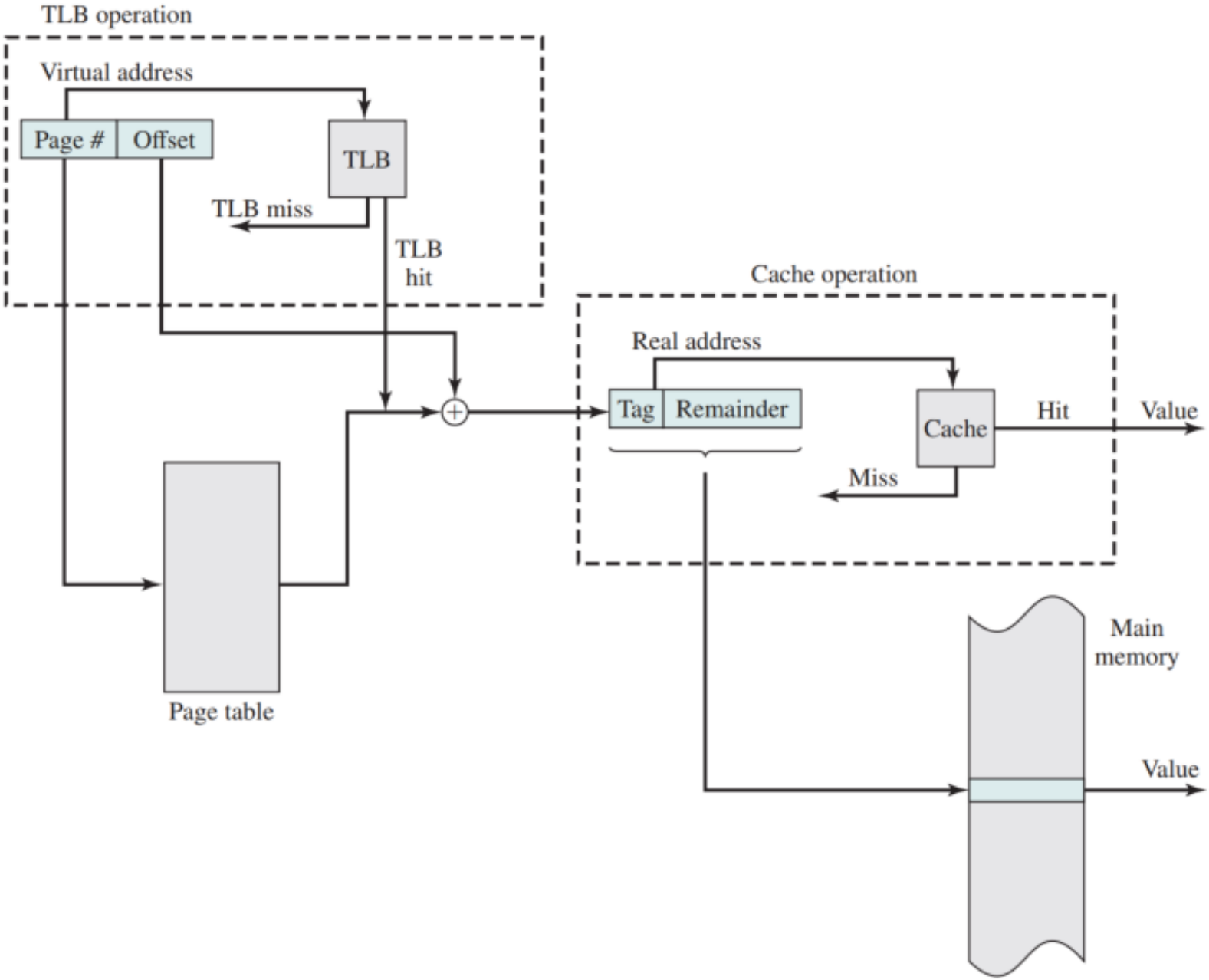


Associative Mapping

- **The TLB only contains some of the page table entries so we cannot simply index into the TLB based on page number**
  - each TLB entry must include the page number as well as the complete page table entry
- **The processor is equipped with hardware that allows it to interrogate simultaneously a number of TLB entries to determine if there is a match on page number**



Direct Versus Associative Lookup for Page Table Entries

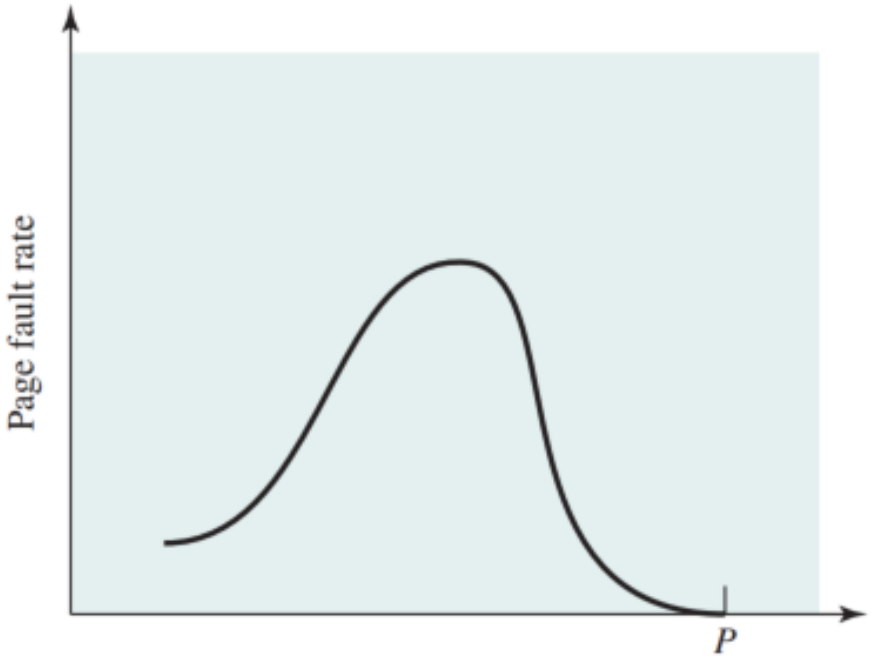


Translation Lookaside Buffer and Cache Operation

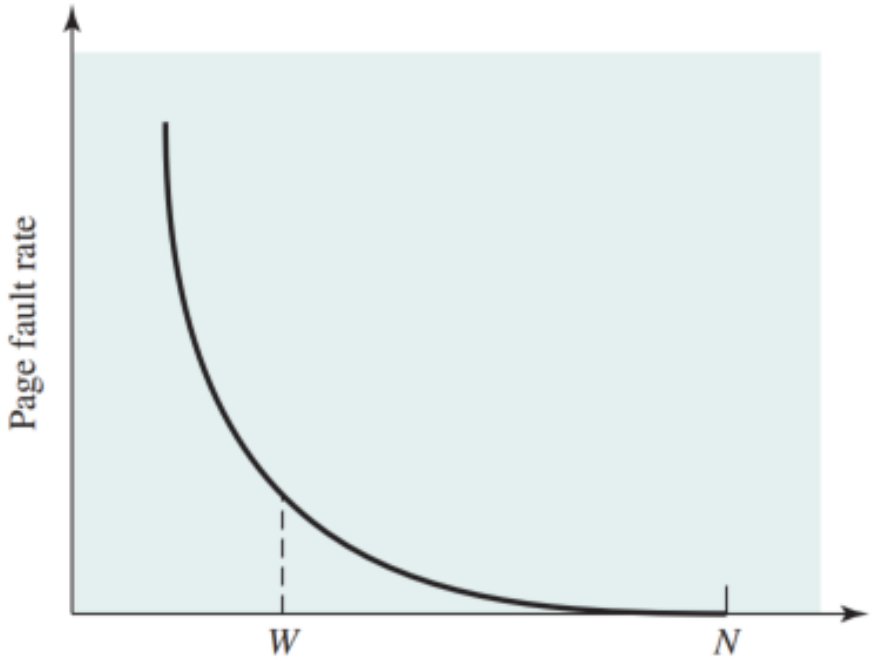
Page Size

- **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 some portion of the page tables of active processes must be in virtual memory instead of main memory
  - the physical characteristics of most secondary-memory devices favor a larger page size for more efficient block transfer of data

Paging Behavior of a Program



(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

Page Size

- The design issue of page size is related to the size of physical main memory and program size
  - main memory is getting larger, the address space used by applications is also growing
  - most obvious on personal computers and workstations, where applications are becoming increasingly complex.
  - contemporary programming techniques used in large programs tend to decrease the locality of references within a process

Example: Page Sizes

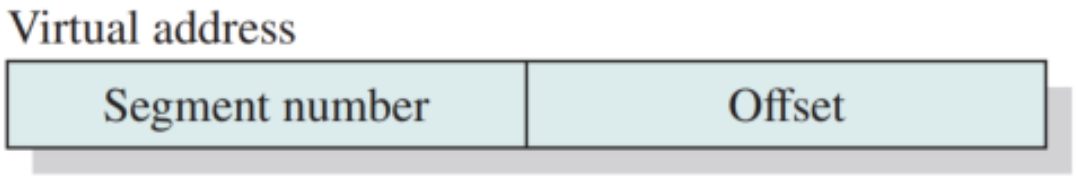
Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1,024 36-bit words
IBM 370/XA and 370/ESA	4 kB
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 kB
MIPS	4 kB to 16 MB
UltraSPARC	8 kB to 4 MB
Pentium	4 kB or 4 MB
Intel Itanium	4 kB to 256 MB
Intel core i7	4 kB to 1 GB

Segmentation

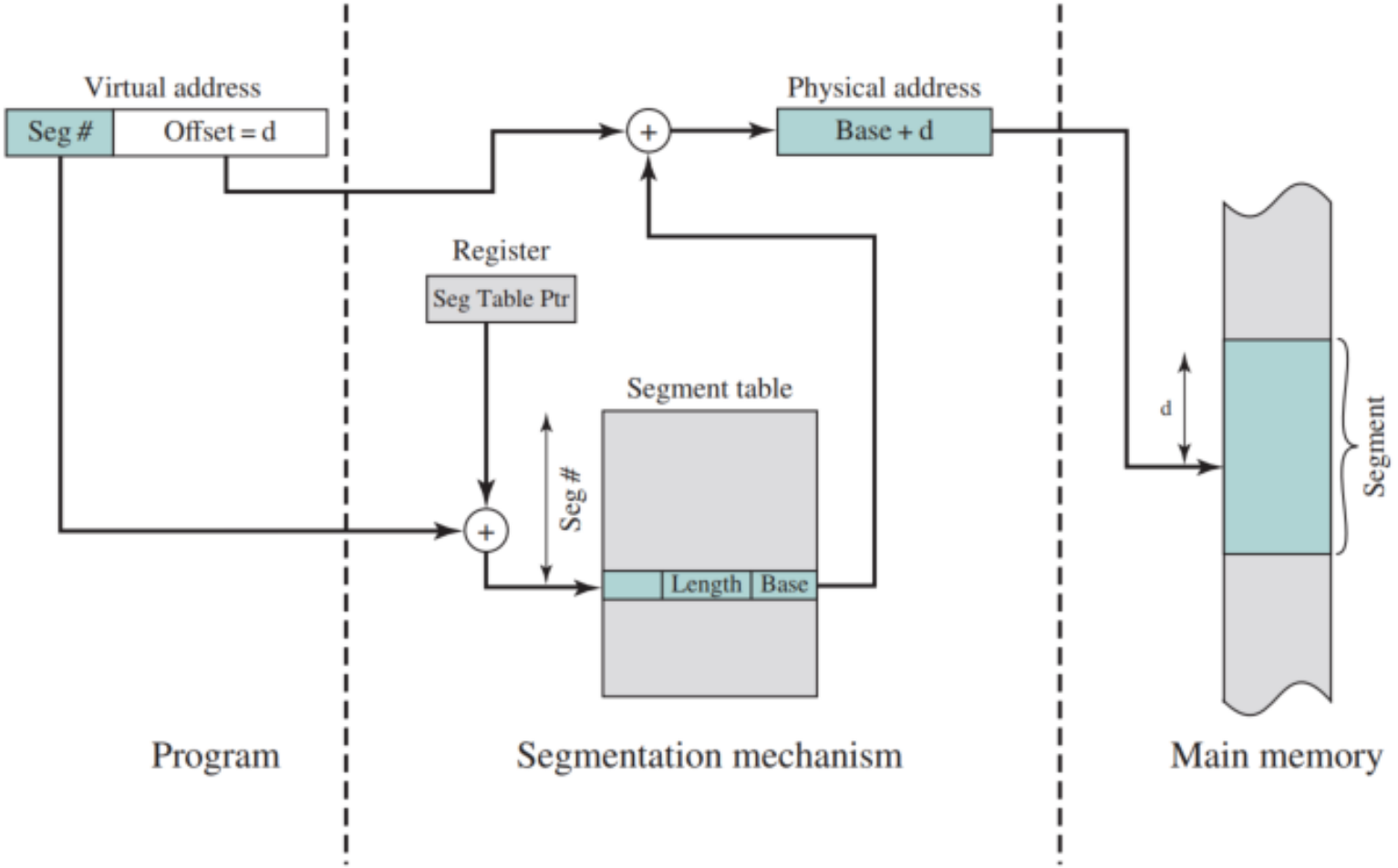
- **Segmentation**
  - allows the **programmer** to view memory as consisting of **multiple address** spaces or segments
  - Segments may be of **unequal**, indeed dynamic, **size**
- **Advantages**
  - simplifies handling of **growing** data structures
  - allows programs to be altered and **recompiled independently**
  - lends itself to sharing data among processes
  - lends itself to protection

Segment Organization

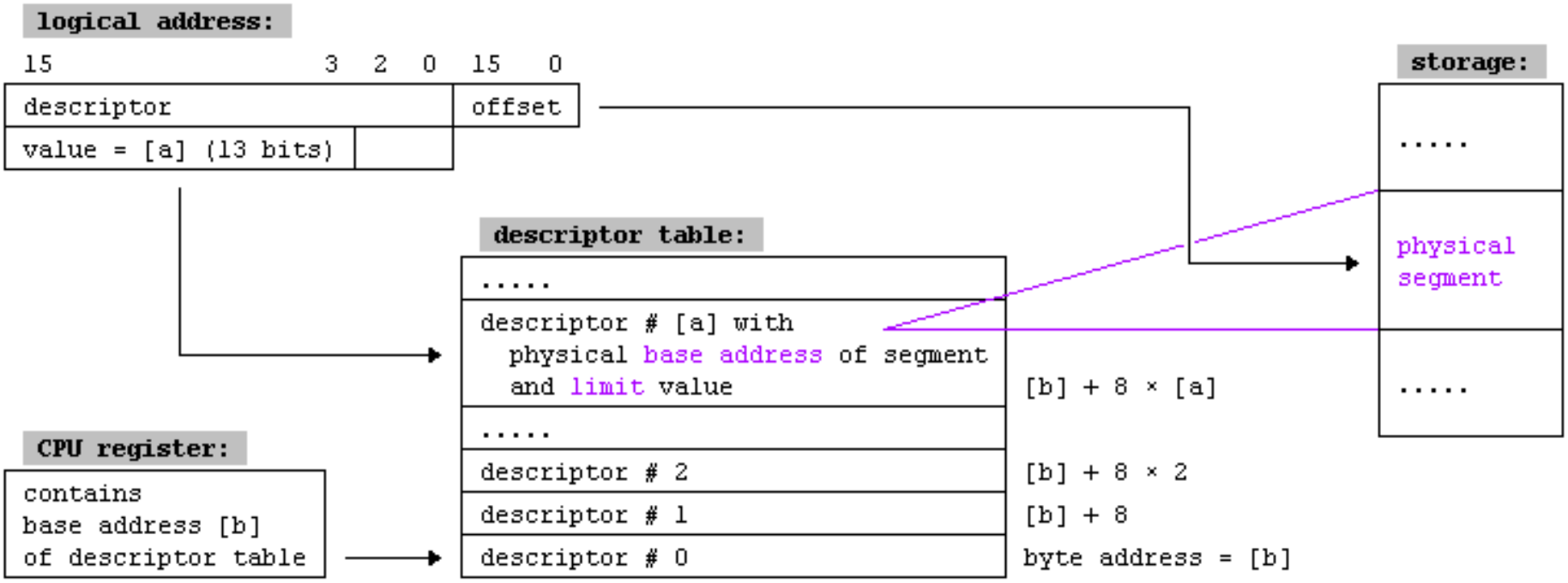
- **Each segment table entry contains**
  - the **starting address** of the corresponding segment in main memory
  - the **length** of the segment



Address Translation in a Segmentation System



Virtual segments of 80286



Combined Paging and Segmentation

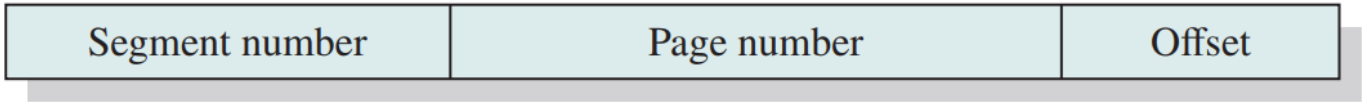
In a combined paging/segmentation system, a user's address space is broken up into a number of segments. Each segment is broken up into a number of fixed-size pages, which are equal in length to a main memory frame.

Segmentation is visible to the programmer

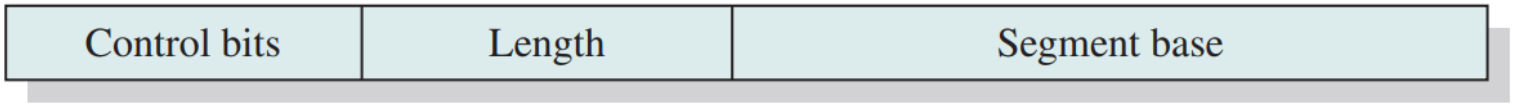
Paging is transparent to the programmer

Combined Segmentation and Paging

Virtual address

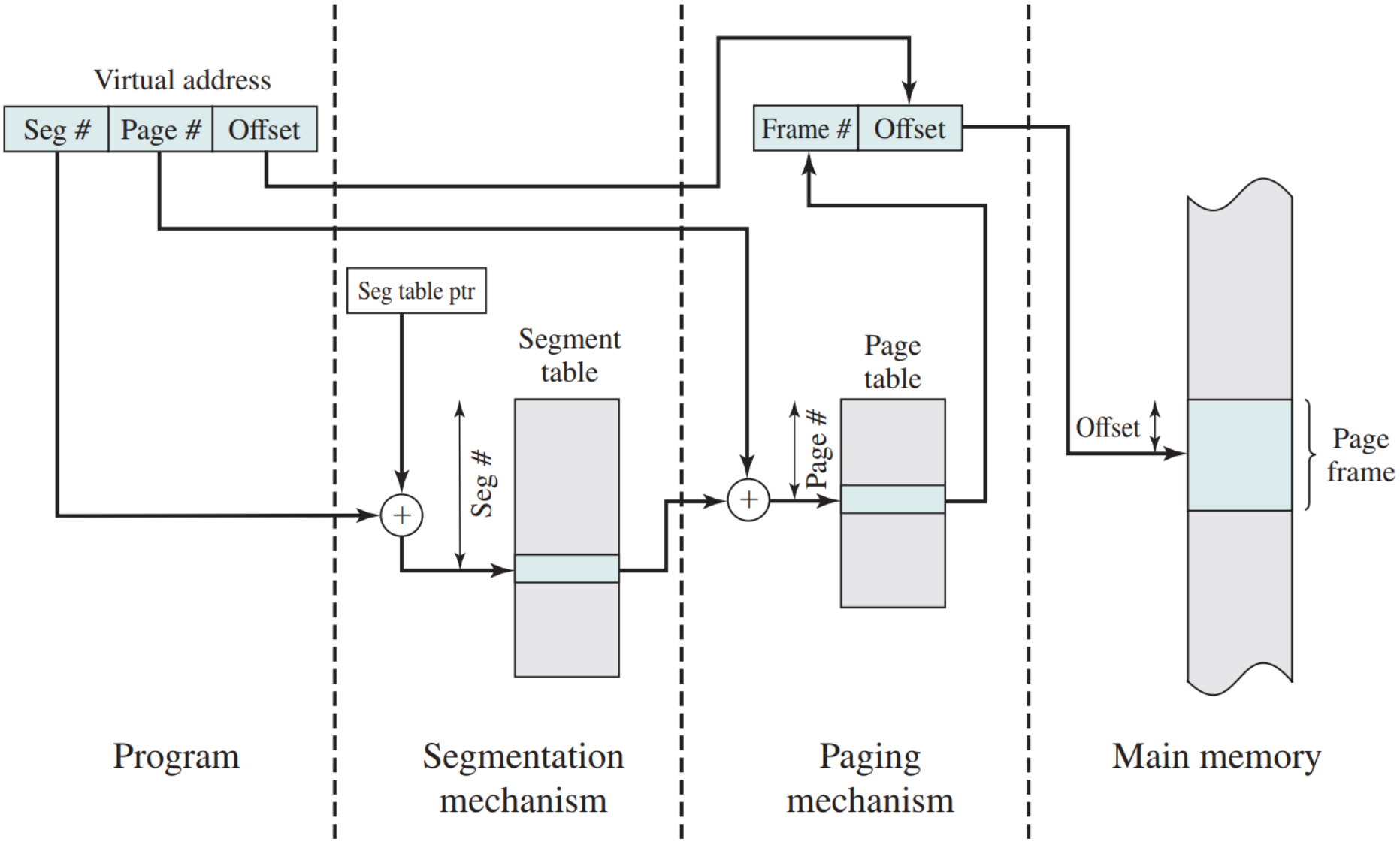


Segment table entry



Page table entry

P = present bit
M = modified bit



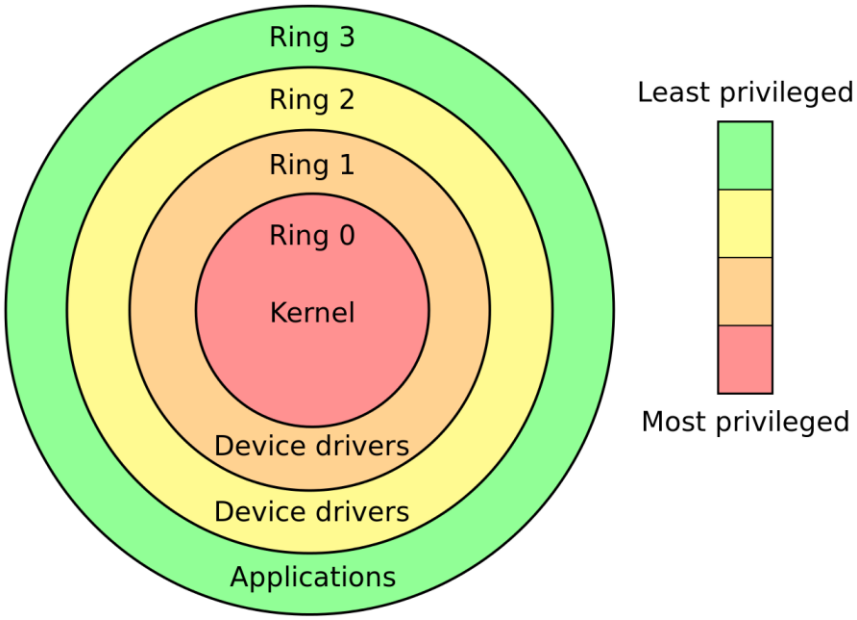
Address Translation in a Segmentation/Paging System

Protection and Sharing

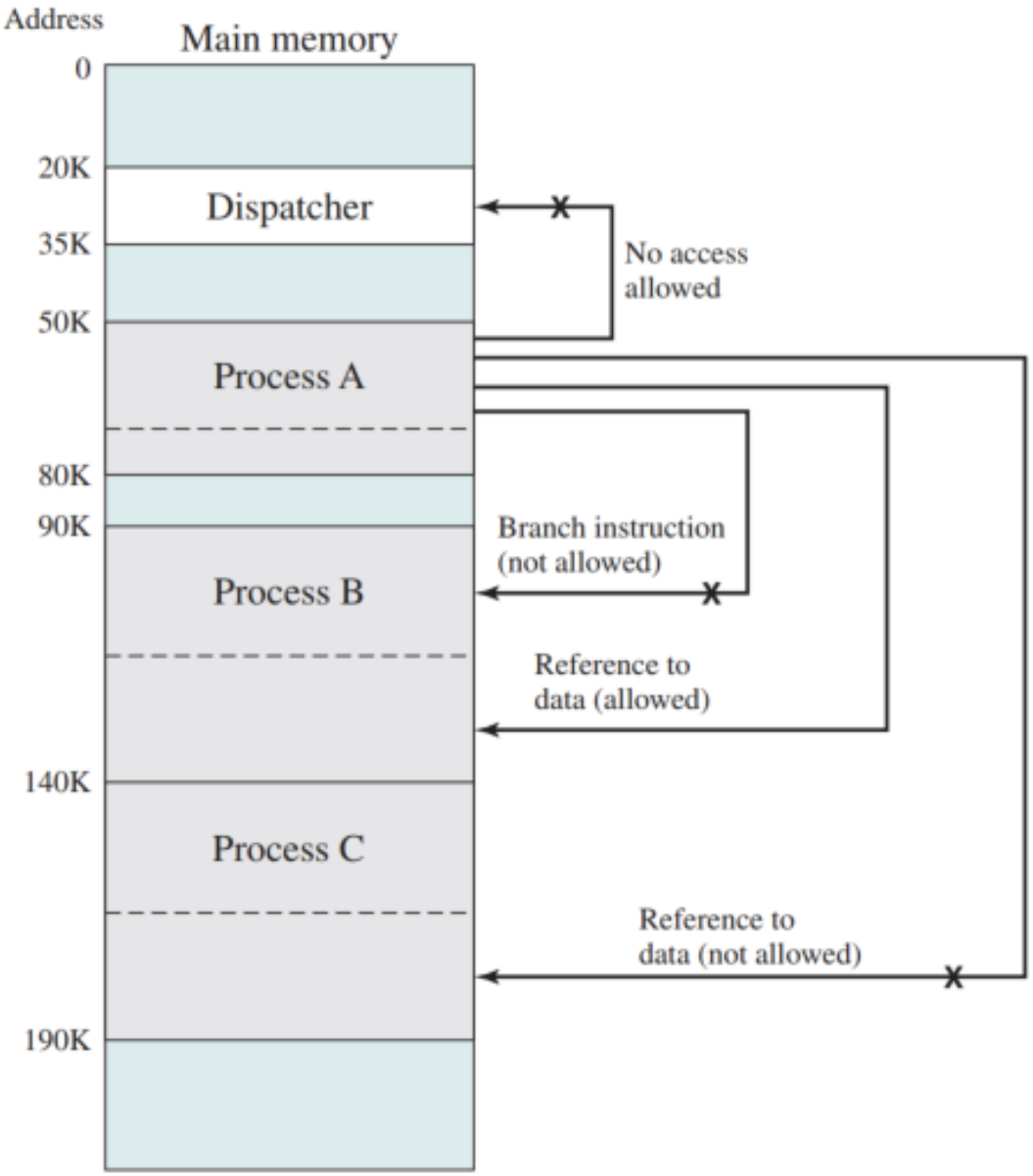
- Segmentation lends itself to the implementation of protection and sharing policies
- Each entry has a base address and length so inadvertent(疏忽的) memory access can be controlled
- Sharing can be achieved by segments referencing multiple processes

Use a ring-protection structure

- **Basic principles of the ring system are**
  - A program may **access only data** that reside on the **same** ring or a **less** privileged ring
  - A program may **call services** residing on the **same** or a **more** privileged ring



Privilege rings for the x86 available in protected mode



Protection Relationships between Segments

Replacement Policy

- Deals with the selection of a page in main memory to be replaced when a new page must be brought in
  - How many page frames are to be allocated to each active process
  - Whether the set of pages to be considered for replacement should be limited to those of the process that caused the page fault or encompass all the page frames in main memory
  - Among the set of pages considered, which particular page should be selected for replacement

Basic Algorithms

- Optimal
- Least recently used (LRU)
- First-in-first-out (FIFO)
- Clock

Optimal

- The optimal policy selects for replacement that page for which the time to the next reference is the longest.
- Results in the fewest number of page faults
- Impossible to implement, because it would require the OS to have perfect knowledge of future events.
- Serve as a standard against which to judge real-world algorithms

Least Recently Used (LRU)

- Replaces the page in memory 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.
- Does nearly as well as the optimal policy.
- Difficulty in implementation.
  - One approach would be to tag each page with the time of its last reference; this would have to be done at each memory reference, both instruction and data. Even if the hardware would support such a scheme, the overhead would be tremendous.
  - Alternatively, one could maintain a stack of page references, again an expensive prospect.

First-in-first-out (FIFO)

- Treats the page frames allocated to a process as a circular buffer, and pages are removed in round-robin style.
  - All that is required is a pointer that circles through the page frames of the process
- One of the simplest page replacement policies to implement
- The logic is that one is replacing the page that has been in memory the longest
  - A page fetched into memory a long time ago may have now fallen out of use
  - This reasoning will often be wrong, because there will often be regions of program or data that are heavily used throughout the life of a program.
  - Those pages will be repeatedly paged in and out by the FIFO algorithm

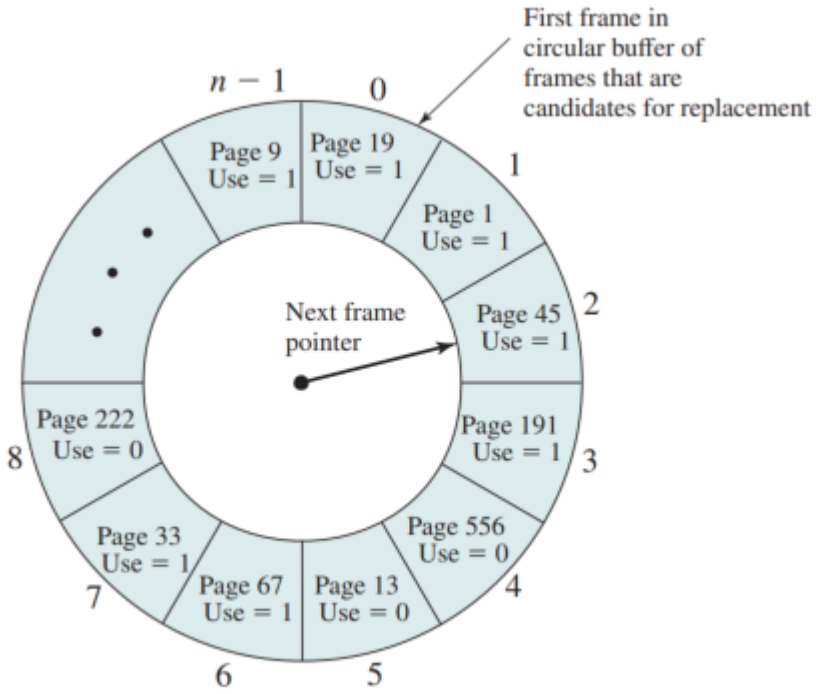
Clock(1)

- Requires the association of an additional bit with each frame, referred to as the use bit.
  - When a page is first loaded into a frame in memory, the use bit for that frame is set to 1.
  - Whenever the page is subsequently referenced (after the reference that generated the page fault), its use bit is set to 1.
- For the page replacement algorithm, the set of frames that are candidates for replacement is considered to be a circular buffer, with which a pointer is associated.
- When a page is replaced, the pointer is set to indicate the next frame in the buffer after the one just updated.
- When it comes time to replace a page, the OS scans the buffer to find a frame with a use bit set to 0. Each time it encounters a frame with a use bit of 1, it resets that bit to 0 and continues on.

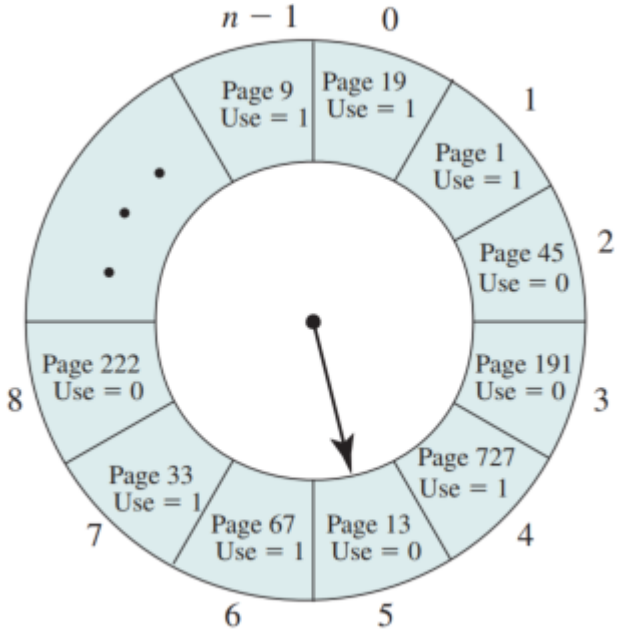
Clock(2)

- If any of the frames in the buffer have a use bit of 0 at the beginning of this process, the first such frame encountered is chosen for replacement.
- If all of the frames have a use bit of 1, then the pointer will make one complete cycle through the buffer, setting all the use bits to 0, and stop at its original position, replacing the page in that frame.
- Similar to FIFO, except that, in the clock policy, any frame with a use bit of 1 is passed over by the algorithm. The policy is referred to as a clock policy because we can visualize the page frames as laid out in a circle.

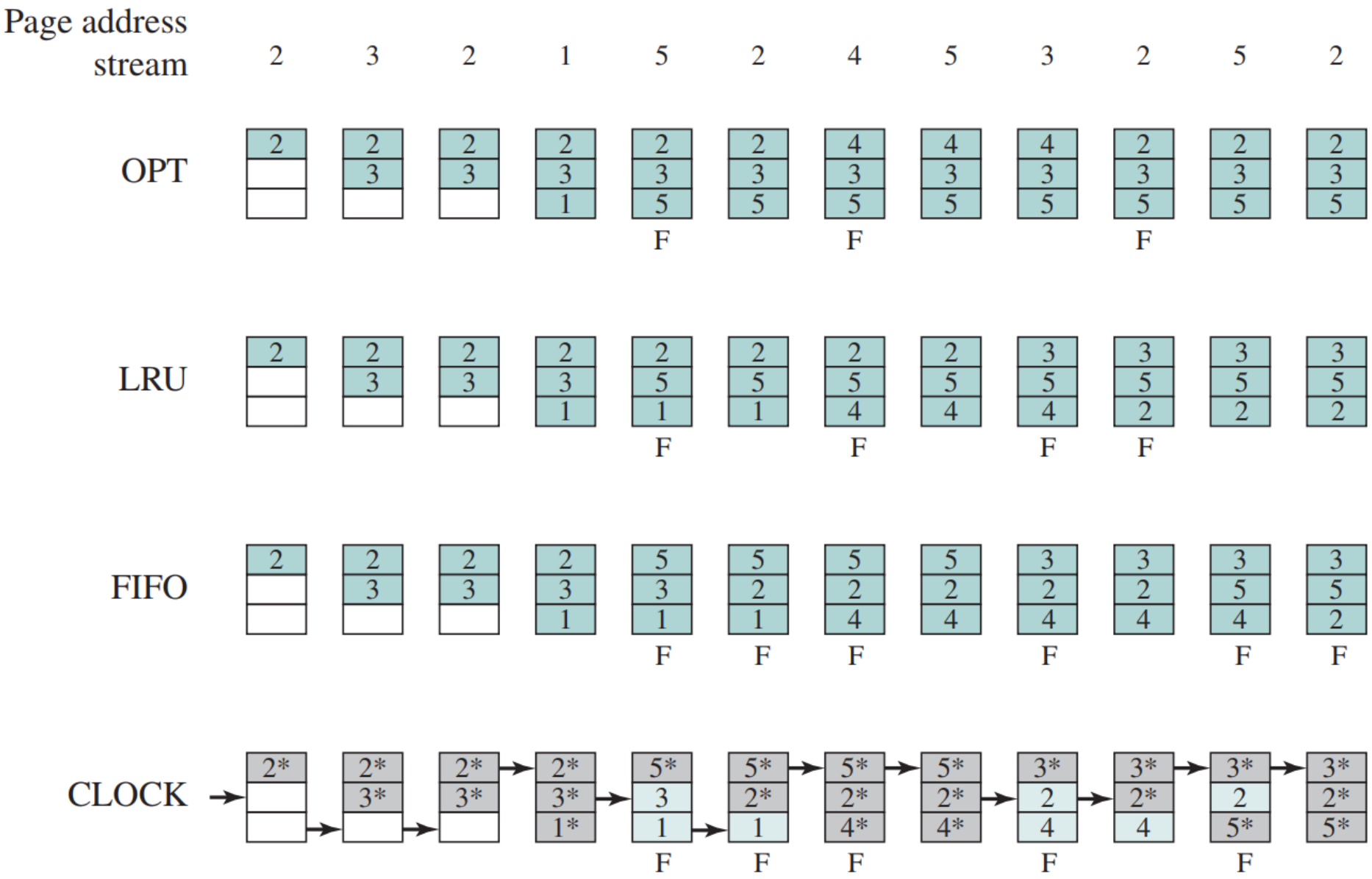
Example of Clock Policy Operation



(a) State of buffer just prior to a page replacement

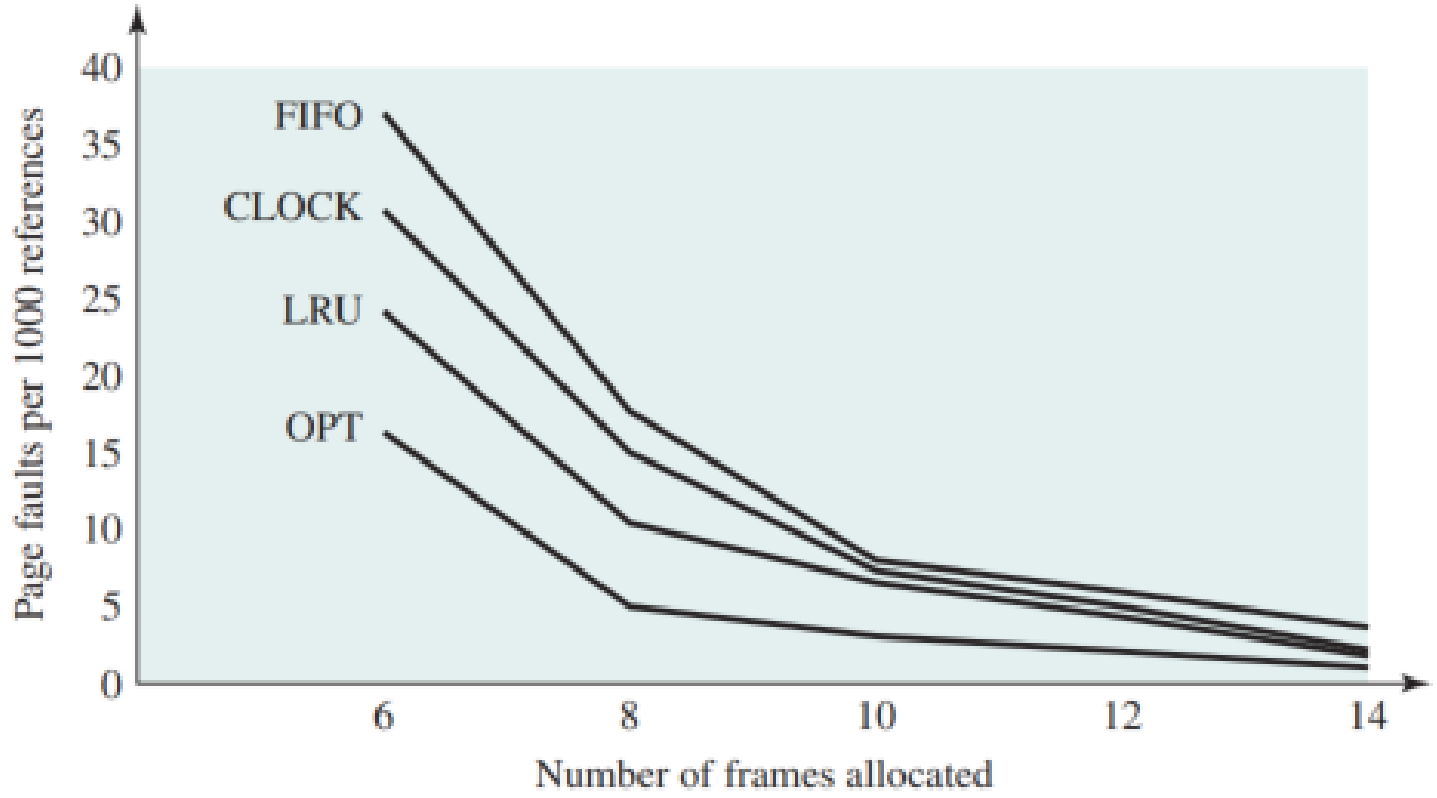


(b) State of buffer just after the next page replacement



Behavior of Four Page Replacement Algorithms

Comparison of Fixed-Allocation, Local Page Replacement Algorithms



Summary

- **Desirable to:**
  - maintain as many processes in main memory as possible
  - free programmers from size restrictions in program development
- **With virtual memory:**
  - all address references are logical references that are translated at run time to real addresses
  - a process can be broken up into pieces
  - two approaches are paging and segmentation
  - management scheme requires both hardware and software support