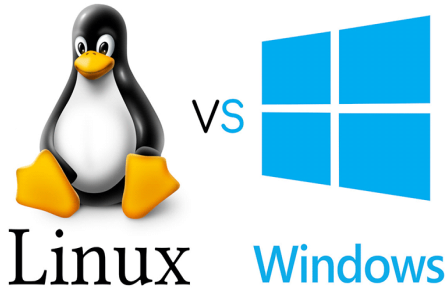
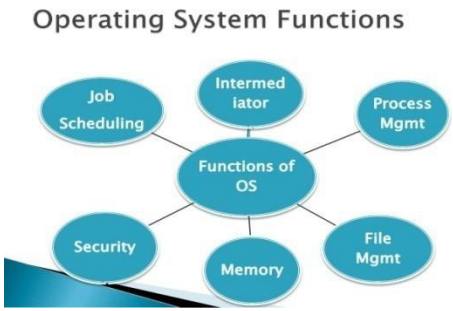
CIT210 OP. SYS. MANAG.

CSUN
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Week6Day2 : Virtual Memory Management, Paging and Page Tables, Virtual-address Space, Virtual Memory Mapping, Fault Page, Valid and Invalid bits, Unix, Linux and Windows Memory Management





Virtual Memory Management

- Background
- Demand Paging
- Demand Segmentation
- Paging Considerations
- Page Replacement Algorithms
- Virtual Memory Policies

Tools of memory management

- Base and limit registers
- Swapping
- Paging (and page tables and TLBs)
- Segmentation (and segment tables)
- Page/segment fault handling => Virtual memory
- The policies that govern the use of these mechanisms

Today's desktop and server systems

- The basic abstraction that the OS provides for memory management is virtual memory (VM)
  - VM enables programs to execute without requiring their entire address space to be resident in physical memory
    - program can also execute on machines with less RAM than it "needs"
  - many programs don't need all of their code or data at once (or ever)
    - e.g., branches they never take, or data they never read/write
    - no need to allocate memory for it, OS should adjust amount allocated based on run-time behavior
  - virtual memory isolates processes from each other
    - one process cannot name addresses visible to others; each process has its own isolated address space
  - Virtual memory requires hardware and OS support
    - MMU's, TLB's, page tables, page fault handling, …
  - Typically accompanied by swapping, and at least limited segmentation

Hardware and Control Structures

- **Two characteristics fundamental to memory management:**
  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 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**

Terminology

Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.
Virtual address	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	The 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.

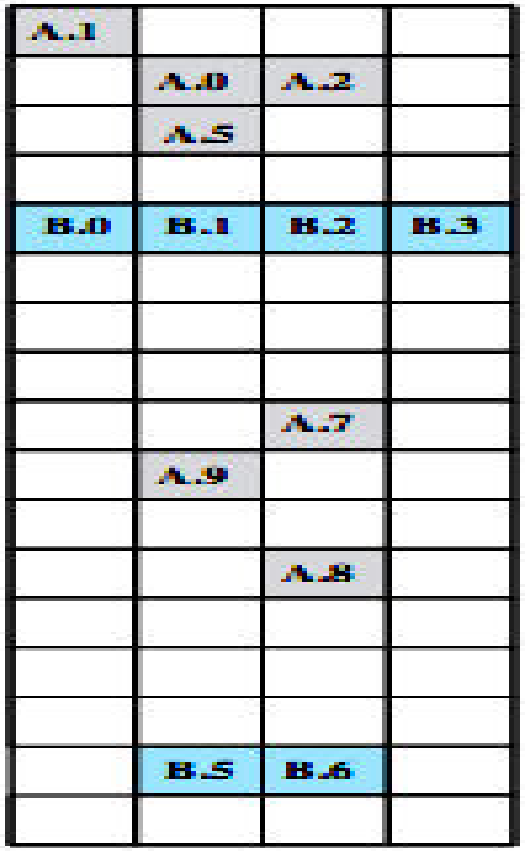
Background (1)

- Code needs to be in memory to execute, but entire program rarely used:
  - Error code, unusual routines, large data structures.
- Entire program code not needed at same time.
- Consider ability to execute partially-loaded program:
  - Program no longer constrained by limits of physical memory.
  - Each program takes less memory while running -> more programs run at the same time:
    - Increased CPU utilization and throughput with no increase in response time or turnaround time.
  - Less I/O needed to load or swap programs into memory -> each user program runs faster.

Background (2)

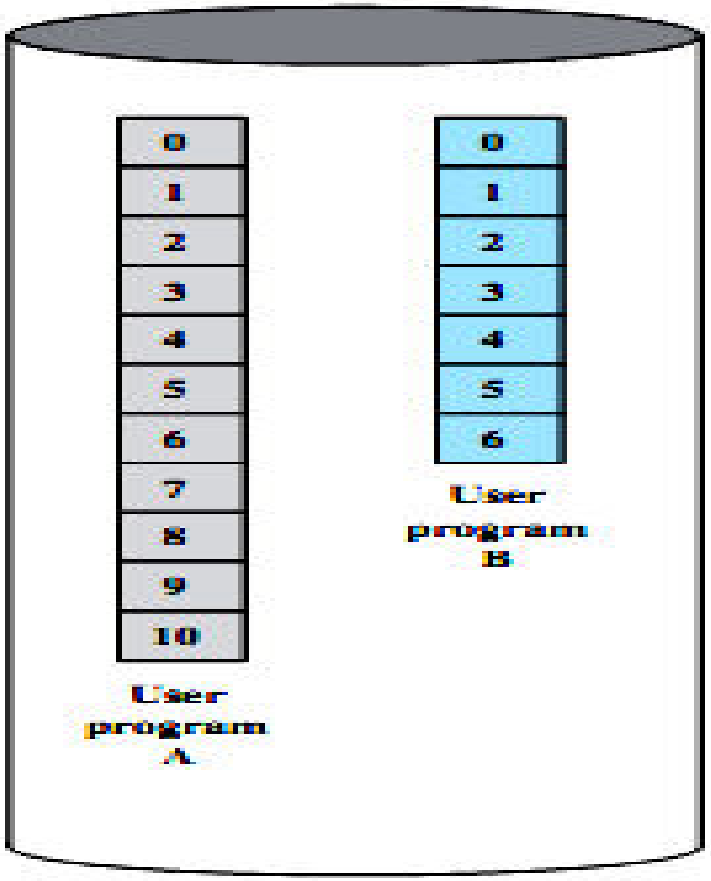
- Virtual memory – separation of user logical memory from physical memory:
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
  - More programs running concurrently.
  - Less I/O needed to load or swap processes.

Virtual Memory Components



Main Memory

Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.



Disk

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Background (3)

- Virtual address space – logical view of how process is stored in memory:
  - Usually start at address 0, contiguous addresses until end of space.
  - Meanwhile, physical memory organized in page frames.
  - MMU must map logical to physical.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Background (4)

- Based on Paging/Segmentation, a process may be broken up into pieces (pages or segments) that do not need to be located contiguously in main memory.
- Based on the Locality Principle, 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.

Multiprogramming Requirements

- **Multiprogramming**
  - **multiple processes/jobs in memory at once**
    - to overlap I/O and computation
  - **memory management requirements:**
    - **protection**: restrict which addresses processes can use, so they can't stomp on each other
    - **fast translation**: memory lookups must be fast, in spite of the protection scheme
    - **fast context switching**: when switching between jobs, updating memory hardware (protection and translation) must be quick

Virtual addresses for multiprogramming

- **To make it easier to manage memory of multiple processes, make processes use virtual addresses**
  - **virtual addresses are independent of location in physical memory (RAM) where referenced data lives**
    - OS determines location in physical memory
  - **instructions issued by CPU reference virtual addresses**
    - e.g., pointers, arguments to load/store instructions, PC …
  - **virtual addresses are translated by hardware into physical addresses (with some setup from OS)**

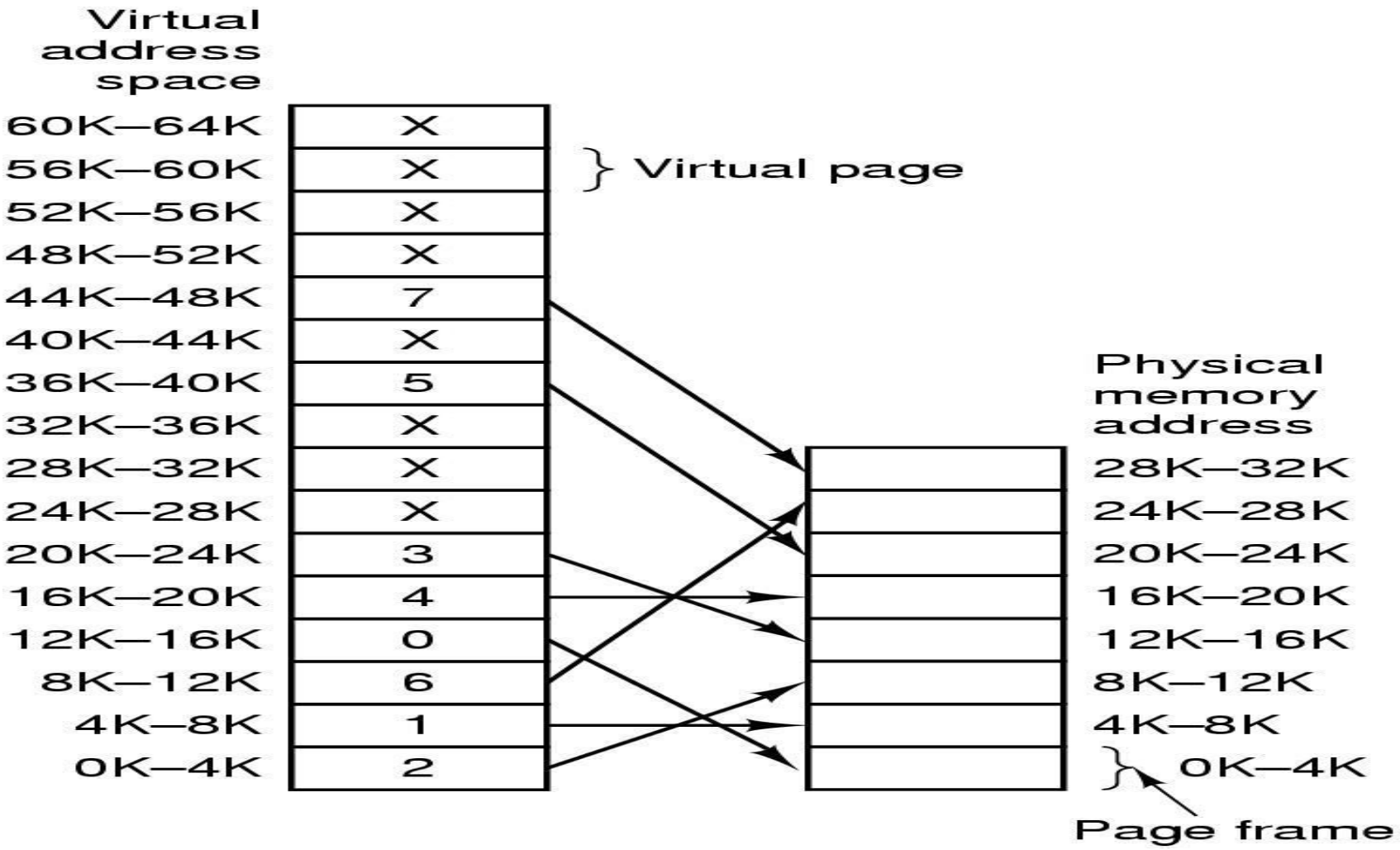
Support Needed for Virtual Memory

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

Virtual memory

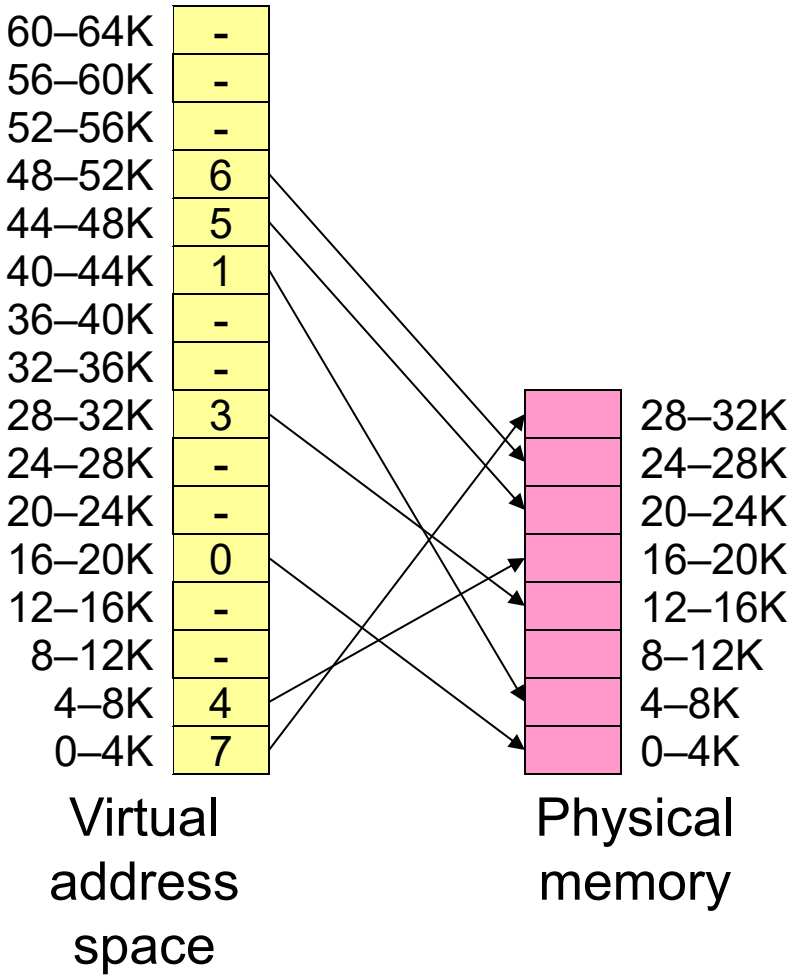
- Basic idea: allow the OS to hand out more memory than exists on the system
- Keep recently used stuff in physical memory
- Move less recently used stuff to disk
- Keep all of this hidden from processes
  - Processes still see an address space from 0 – max address
  - Movement of information to and from disk handled by the OS without process help
- Virtual memory (VM) especially helpful in multiprogrammed system
  - CPU schedules process B while process A waits for its memory to be retrieved from disk

Virtual Memory Example



Paging and Page tables

- Virtual addresses mapped to physical addresses
  - Unit of mapping is called a *page*
  - All addresses in the same virtual page are in the same physical page
  - *Page table entry* (PTE) contains translation for a single page
- Table translates virtual page number to physical page number
  - Not all virtual memory has a physical page
  - Not every physical page need be used
- Example:
  - 64 KB virtual memory
  - 32 KB physical memory



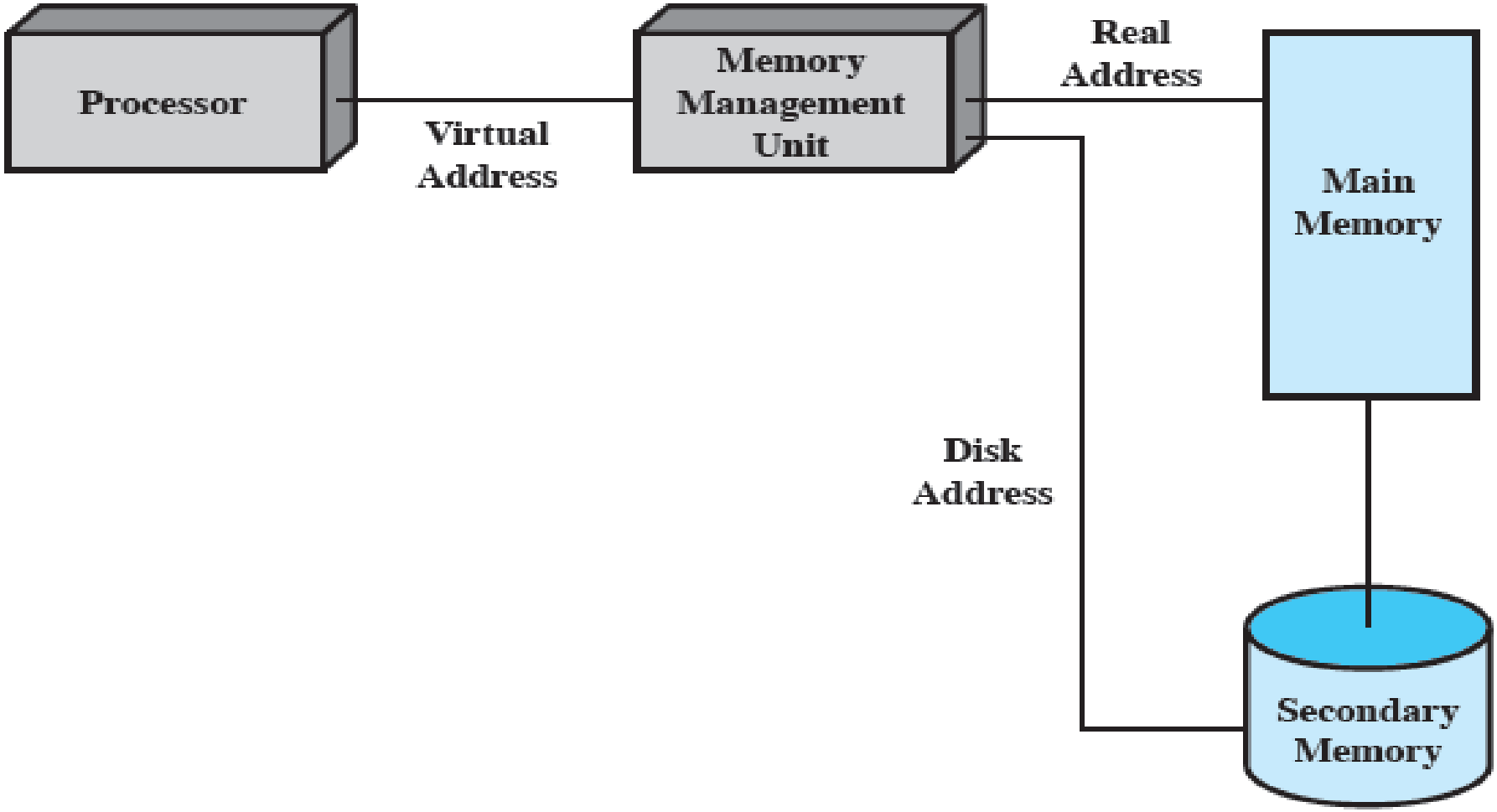
Advantages of Partial Loading

- More processes can be maintained in main memory:
  - only load in some of the pieces of each process.
  - with more processes in main memory, it is more likely that a process will be in the Ready state at any given time.
- A process can now execute even if it is larger than the main memory size:
  - it is even possible to use more bits for logical addresses than the bits needed for addressing the physical memory.

Support needed for Virtual Memory

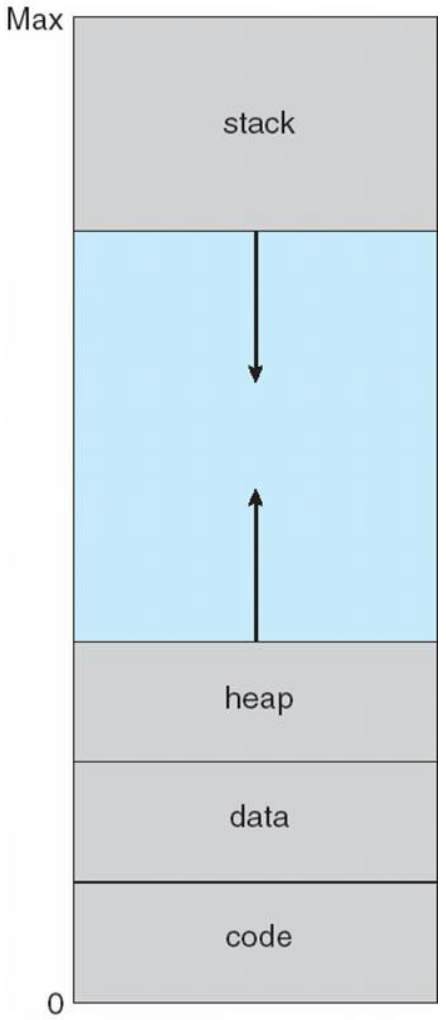
- Memory management hardware must support paging and/or segmentation.
- OS must be able to manage the movement of pages and/or segments between external memory and main memory, including placement and replacement of pages/segments.

Virtual Memory Addressing

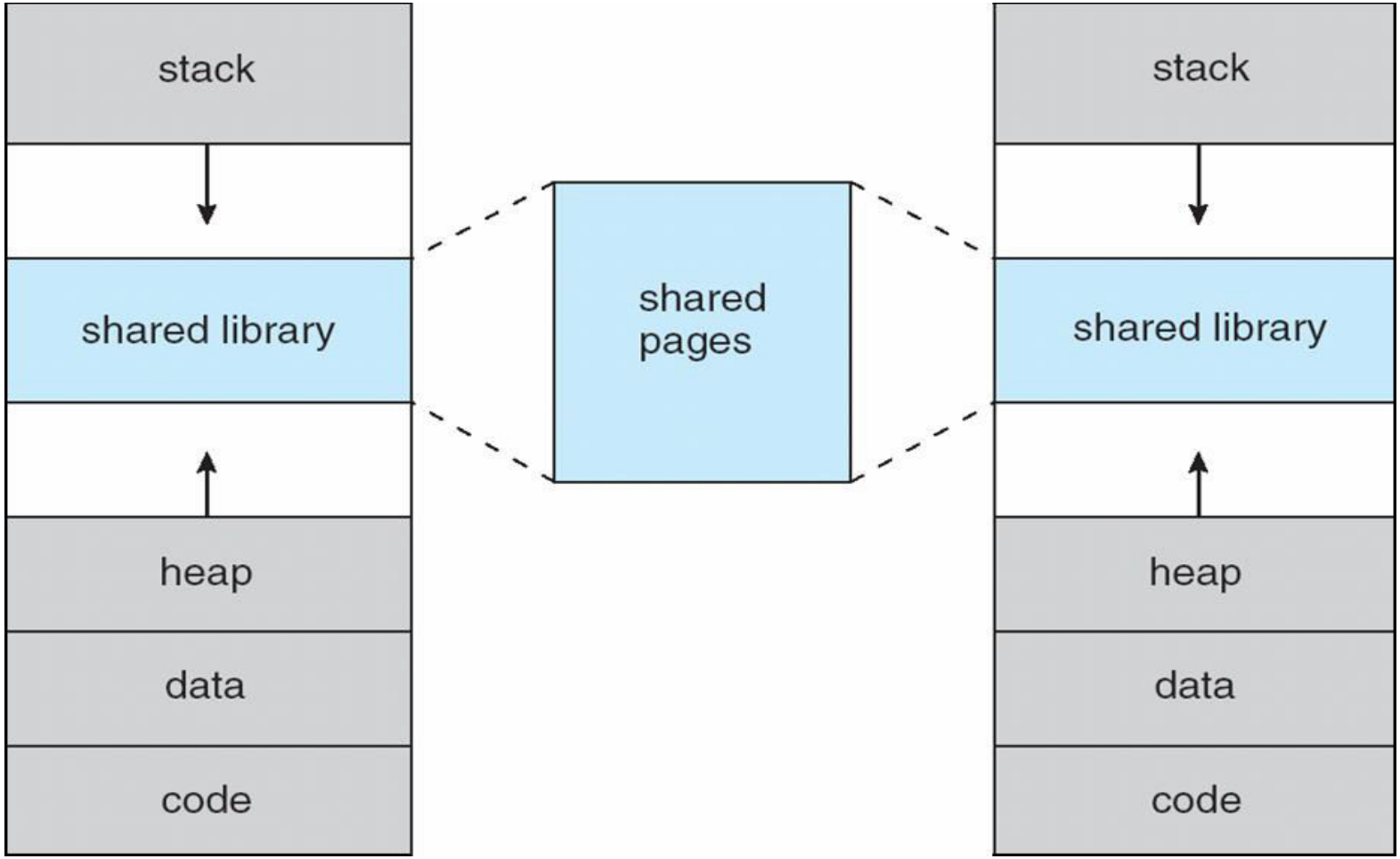


Virtual-address Space

- Usually design logical address space for stack to start at Max logical address and grow “down” while heap grows “up”:
  - Maximizes address space use.
  - Unused address space between the two is hole.
  - No physical memory needed until heap or stack grows to a given new page.
- Enables sparse address spaces with holes left for growth, dynamically linked libraries, etc.
- System libraries shared via mapping into virtual address space.
- Shared memory by mapping pages read-write into virtual address space.
- Pages can be shared during `fork()`, speeding process creation.



Shared Library using Virtual Memory



Process Execution (1)

- The OS brings into main memory only a 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.

Process Execution (2)

- 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.

Demand Paging

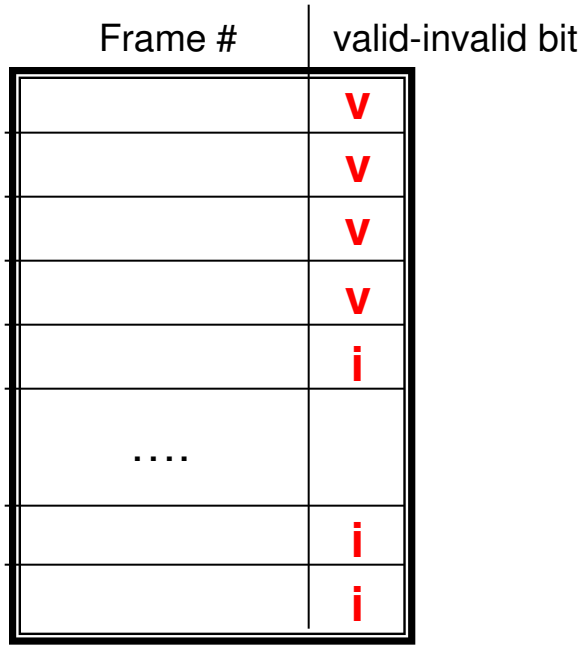
- Bring a page into memory only when it is needed:
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it:
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- Similar to paging system with swapping.
- Lazy swapper – never swaps a page into memory unless page will be needed; Swapper that deals with pages is a pager.

Basic Concepts

- With swapping, pager guesses which pages will be used before swapping out again.
- Instead, pager brings in only those pages into memory.
- How to determine that set of pages?
  - Need new MMU functionality to implement demand paging.
- If pages needed are already memory resident:
  - No difference from non demand-paging.
- If page needed and not memory resident:
  - Need to detect and load the page into memory from storage:
    - Without changing program behavior.
    - Without programmer needing to change code.

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 snapshot:

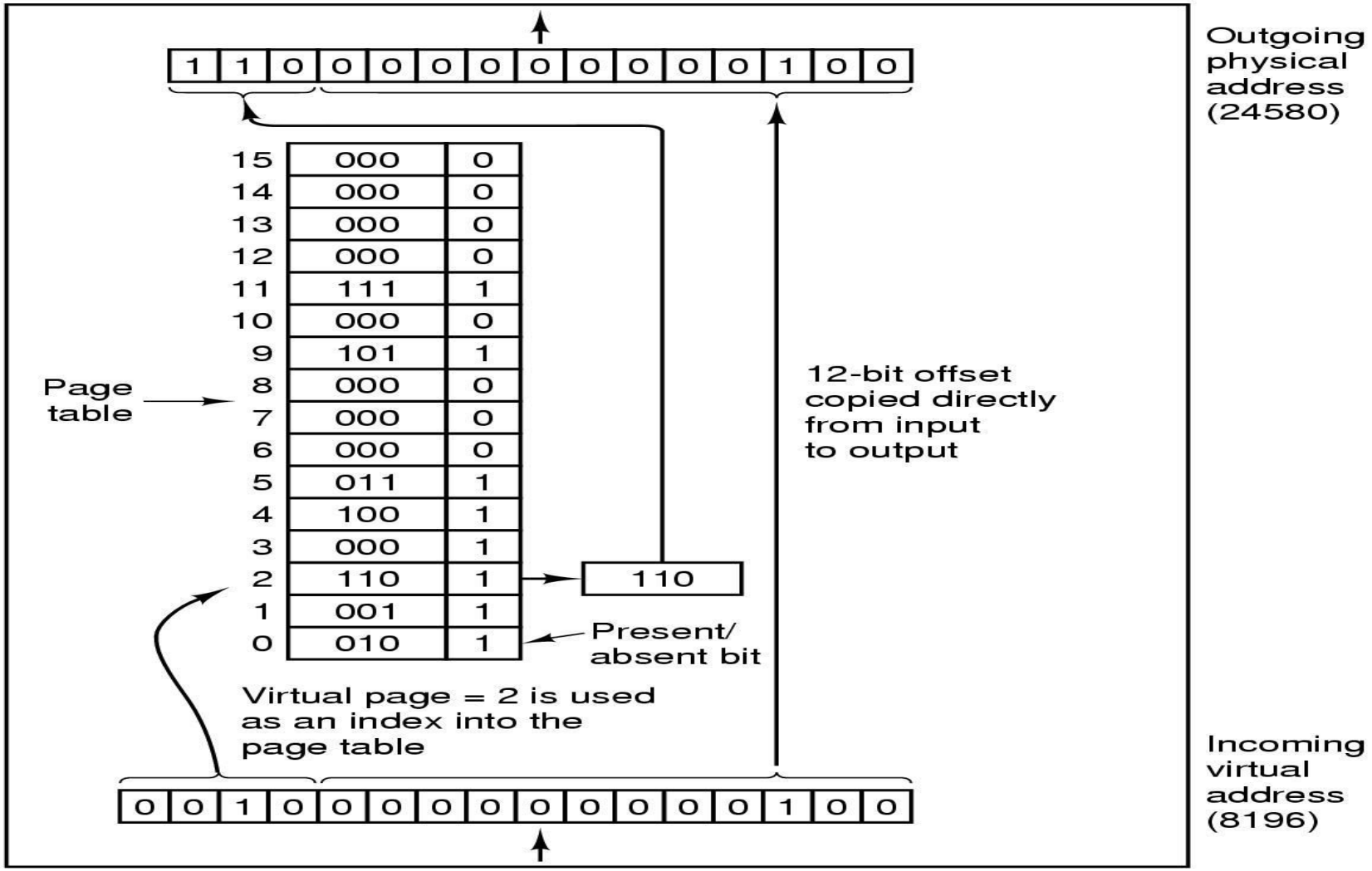


During address translation, if valid–invalid bit in page table entry is i ⇒ page fault.

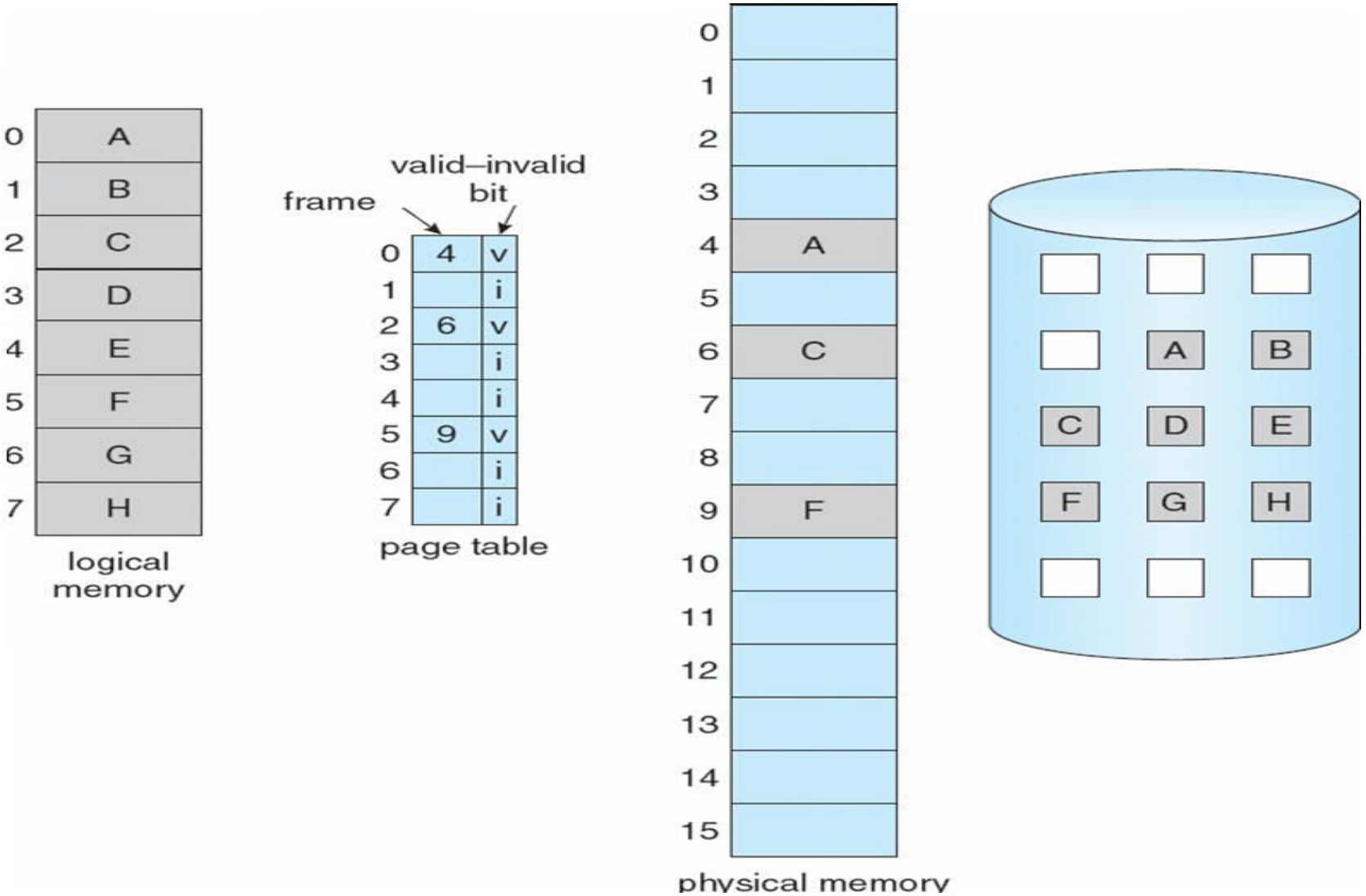
Page Fault

- A **page fault** is a type of exception raised by computer hardware when a running program accesses a memory **page** that is not currently mapped by the memory management unit (MMU) into the virtual address space of a process.
- A **page fault** occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The **fault** notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.

Virtual Memory Mapping Example

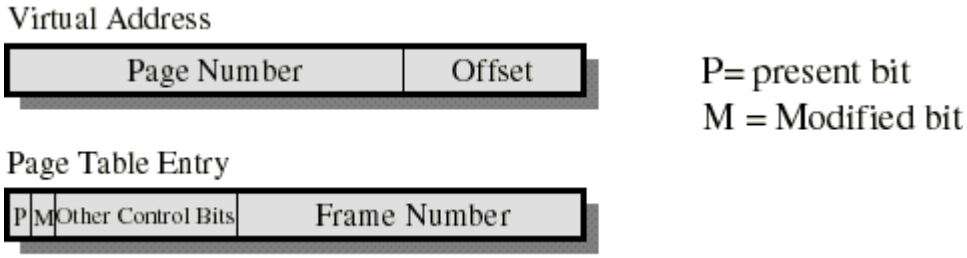


Page Table when some Pages are not in Main Memory



Dynamics of Demand Paging (1)

- Typically, each process has its own page table.

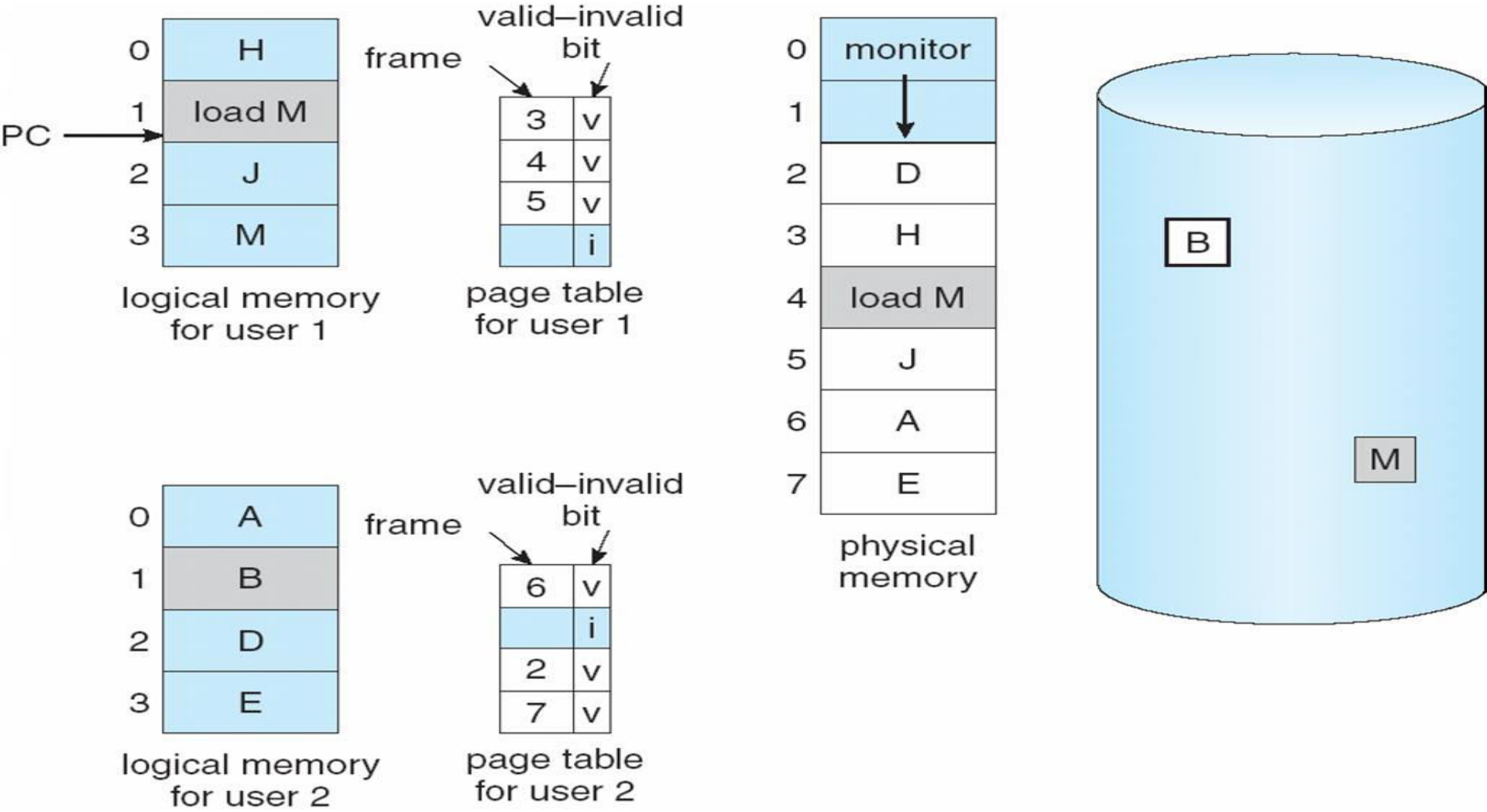


- Each page table entry contains a present (valid-invalid) bit to indicate whether the page is in main memory or not.
  - If it is in main memory, the entry contains the frame number of the corresponding page in main memory.
  - If it is not in main memory, the entry may contain the address of that page 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.

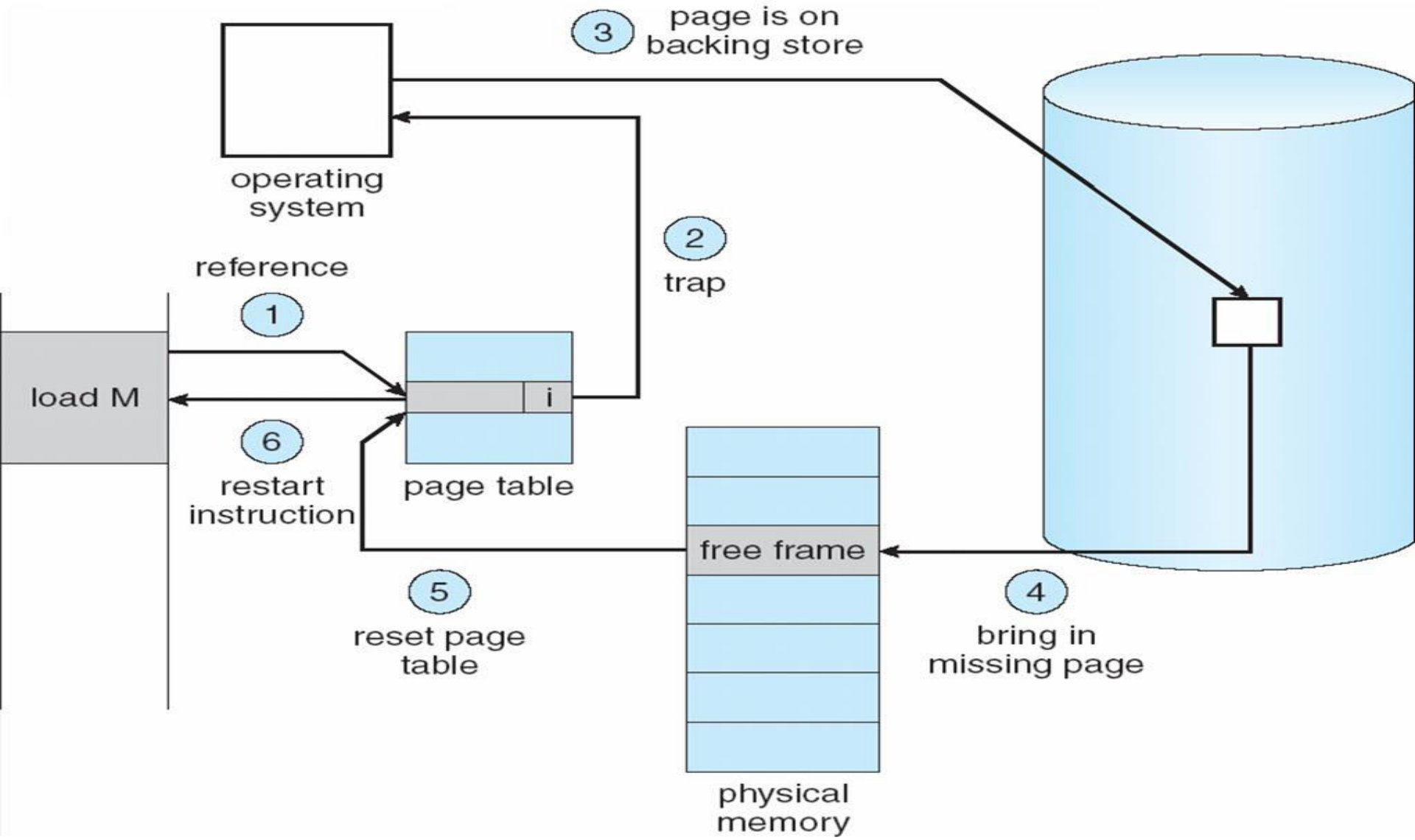
Dynamics of Demand Paging (2)

- 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:
  - a read-only/read-write bit.
  - protection level bit: kernel page or user page (more bits are used when the processor supports more than 2 protection levels).

Need For Page Fault/Replacement



Steps in handling a Page Fault (1)



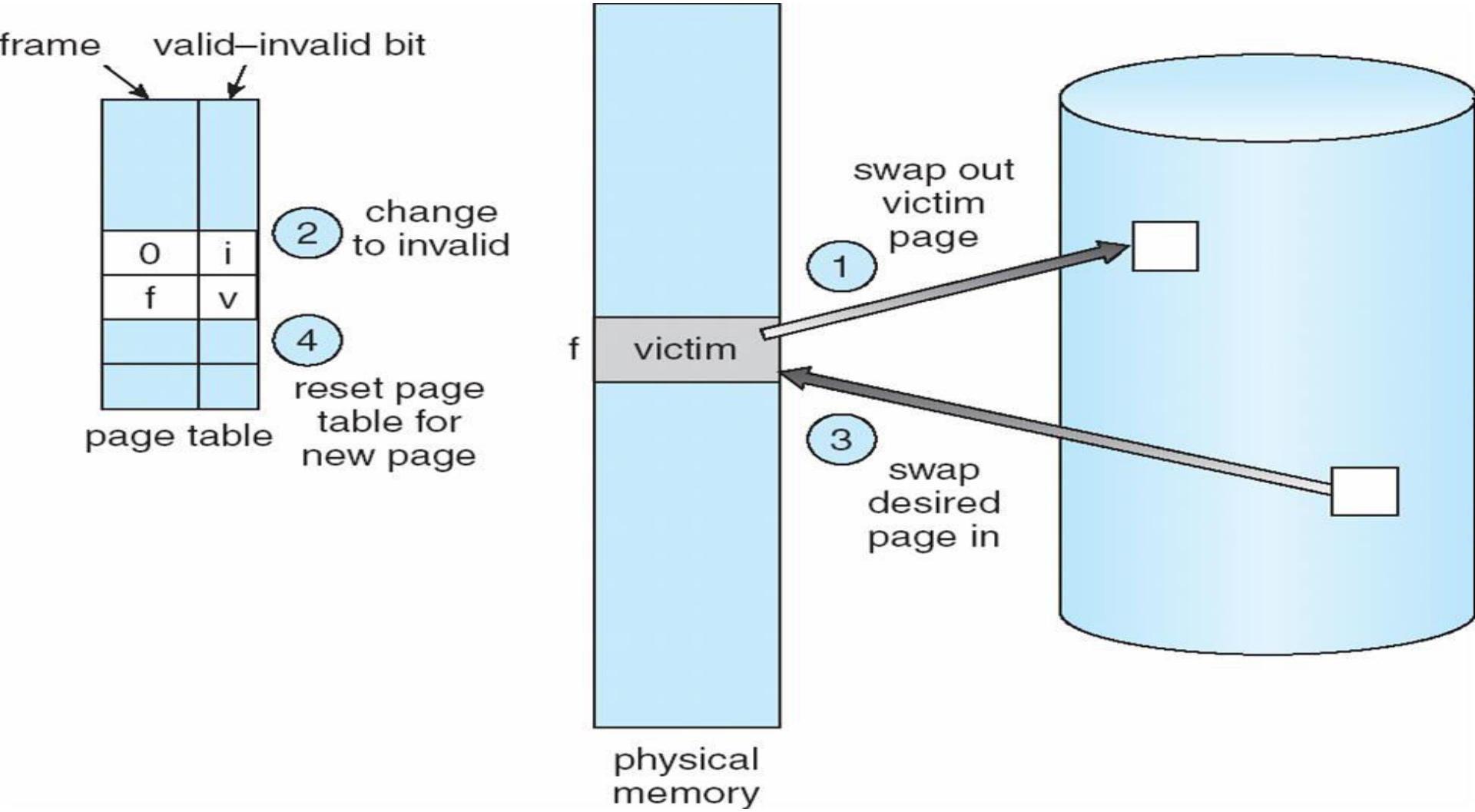
Steps in handling a Page Fault (2)

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 = v.
6. Restart the instruction that caused the page fault.

What happens if there is no free frame?

- 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 (1)



Steps in handling a Page Replacement (2)

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.

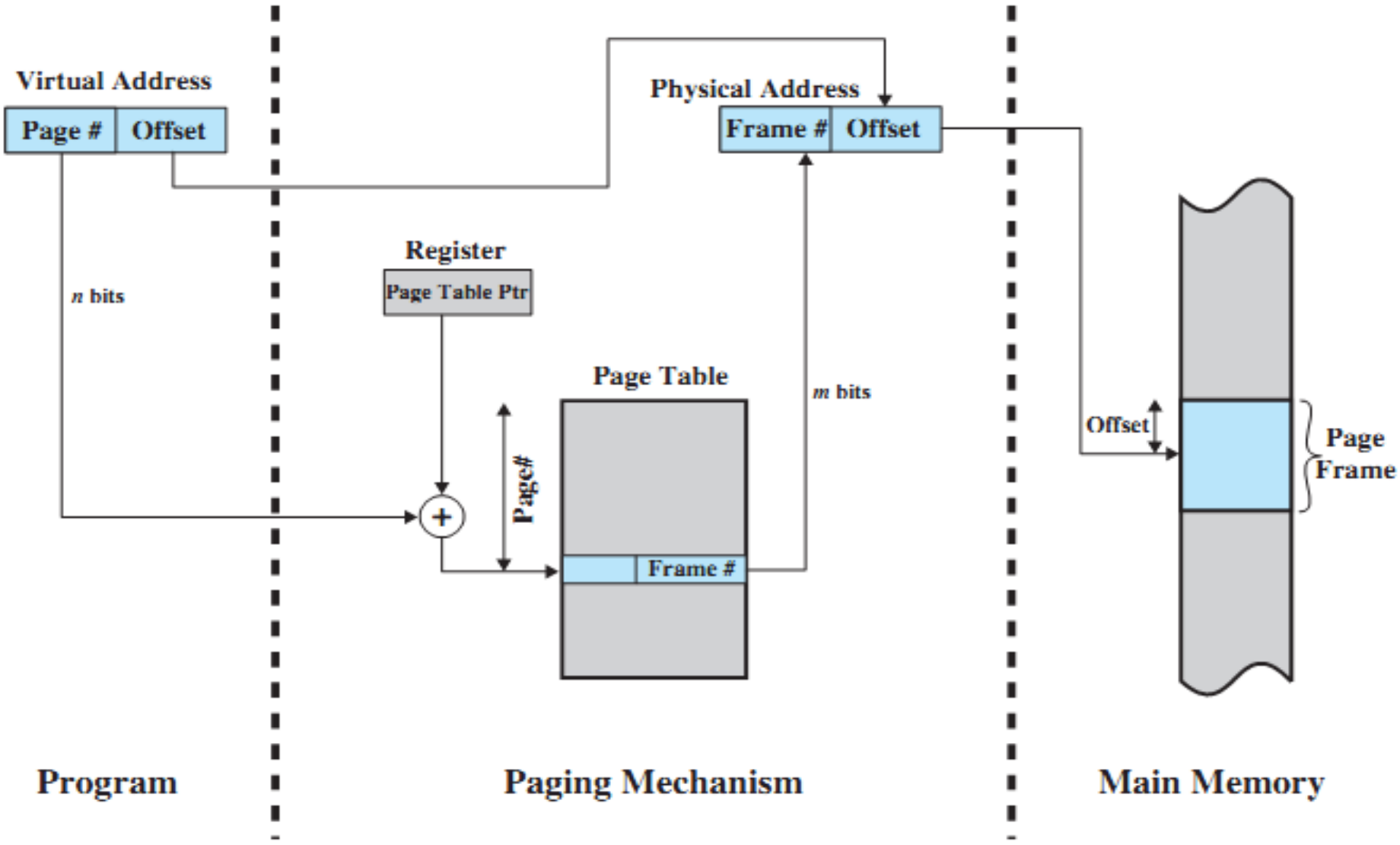
Comments on Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

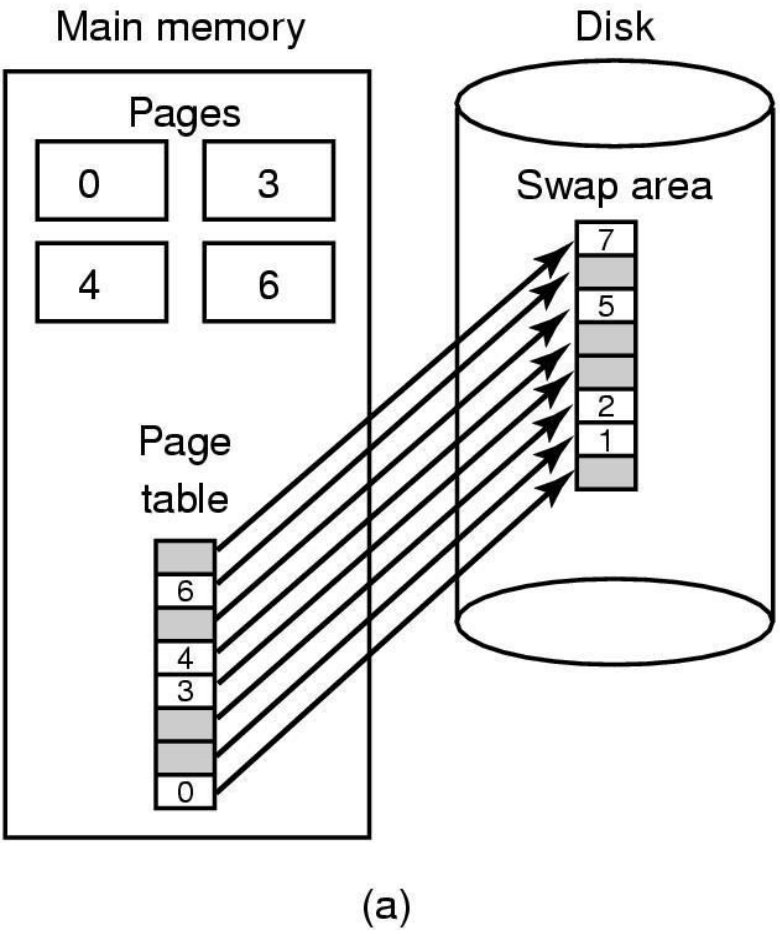
Aspects of Demand Paging

- Extreme case – start process with *no* pages in memory:
  - OS sets instruction pointer to first instruction of process, non-memory-resident -> page fault.
  - And for every other process pages on first access.
  - This is Pure demand paging.
- Actually, a given instruction could access multiple pages -> multiple page faults:
  - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory.
  - Pain decreased because of locality of reference.
- Hardware support needed for demand paging:
  - Page table with valid/invalid bit.
  - Secondary memory (swap device with swap space).
  - Instruction restart.

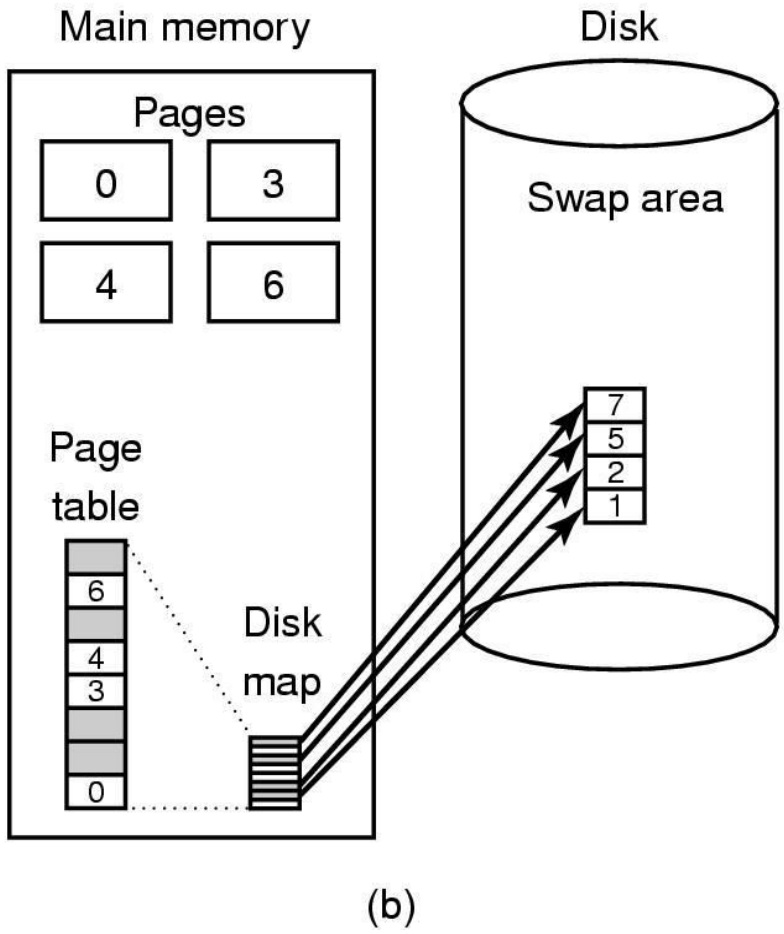
Address Translation in a Paging System



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.

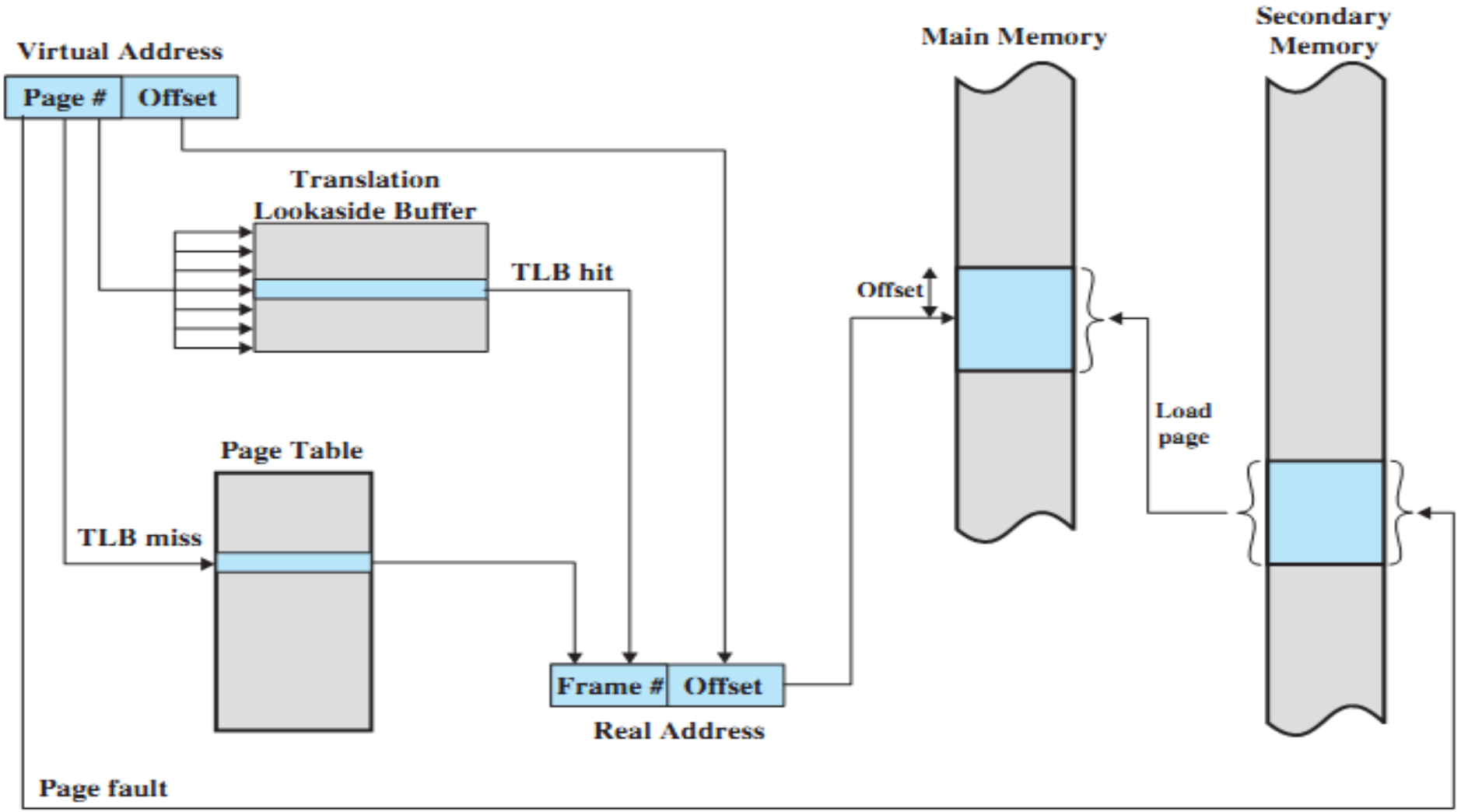
Example TLB

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

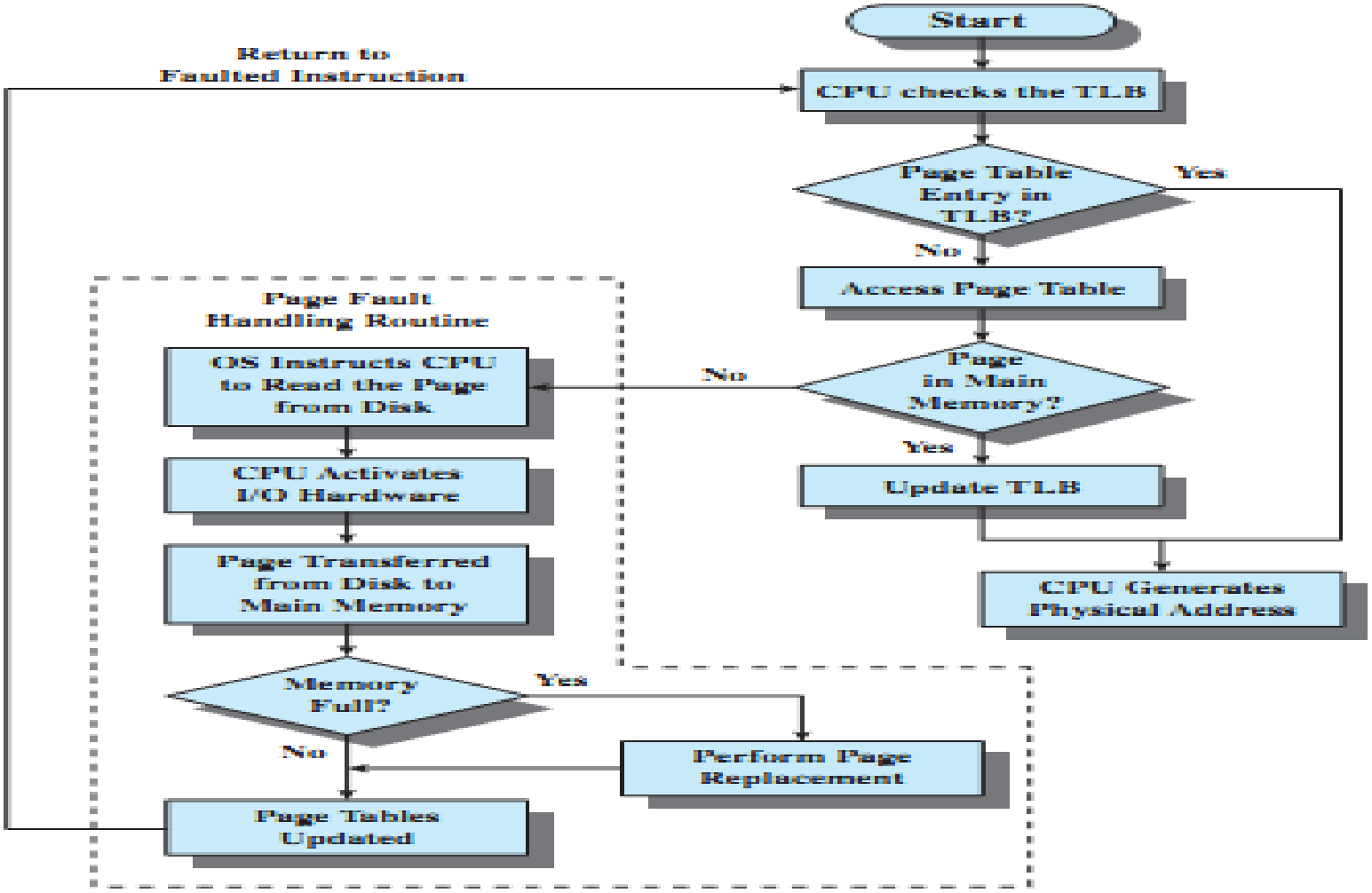
TLB Dynamics

- 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.

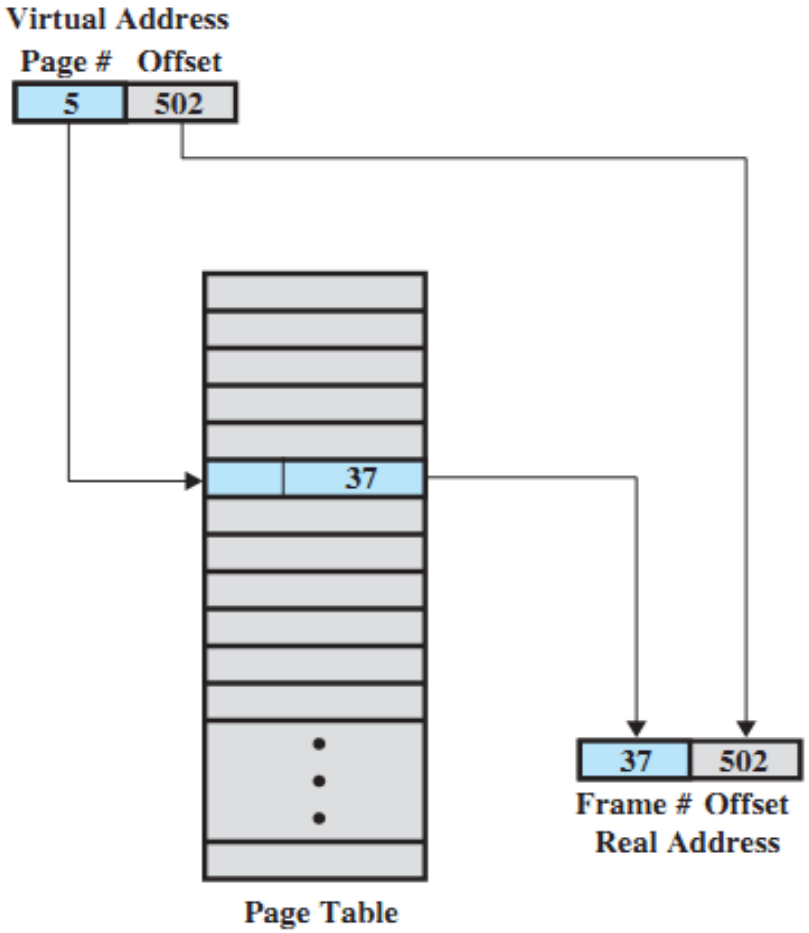
Use of a Translation Look-aside Buffer



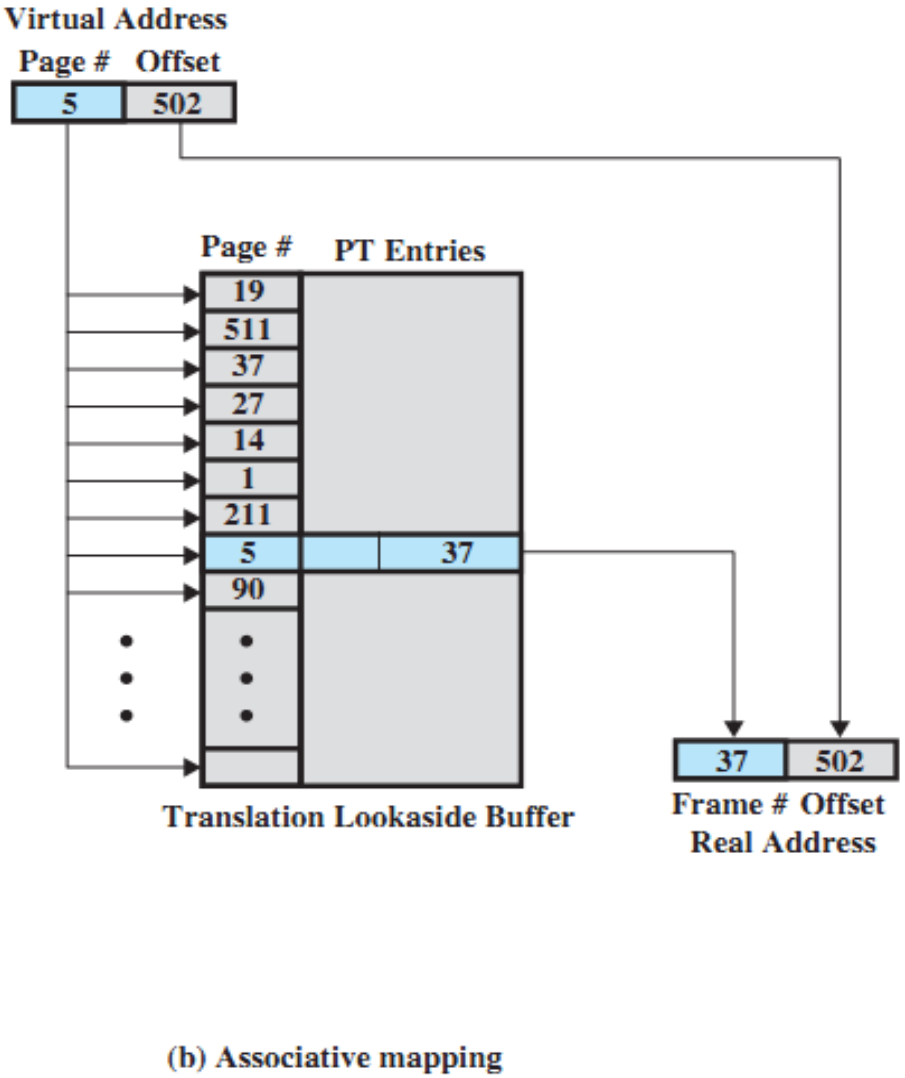
Operation of Paging and TLB



Direct vs. Associative Lookup for Page Table Entries



(a) Direct mapping



(b) Associative mapping

TLB: further comments

- 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.

Stages in Demand Paging (worse case)

1. Trap to the operating system.
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Check that the page reference was legal and determine location of page on the disk.
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced.
    2. Wait for the device seek and/or latency time.
    3. Begin the transfer of the page to a free frame.
6. While waiting, allocate the CPU to some other user.
7. Receive an interrupt from the disk I/O subsystem (I/O completed).
8. Save the registers and process state for the other user.
9. Determine that the interrupt was from the disk.
10. Correct the page table and other tables to show page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction.

Characteristics of Paging and Segmentation

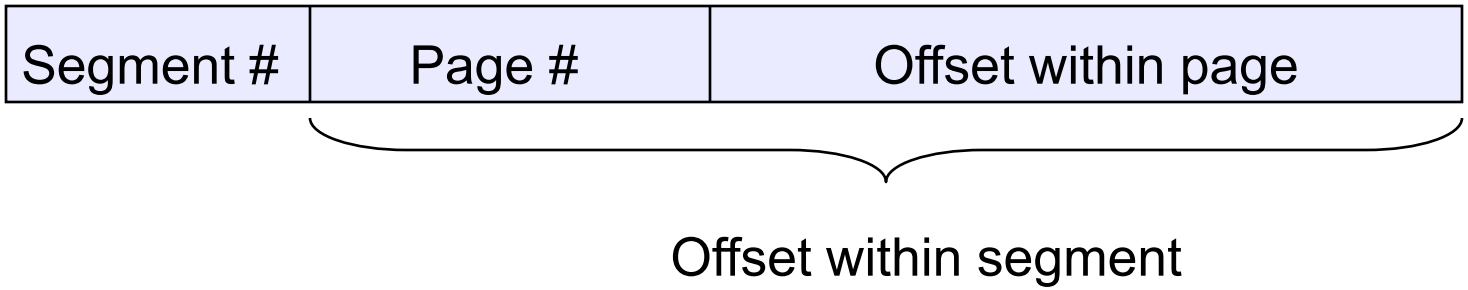
Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Program broken into pages by 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

Intel's Memory Architecture

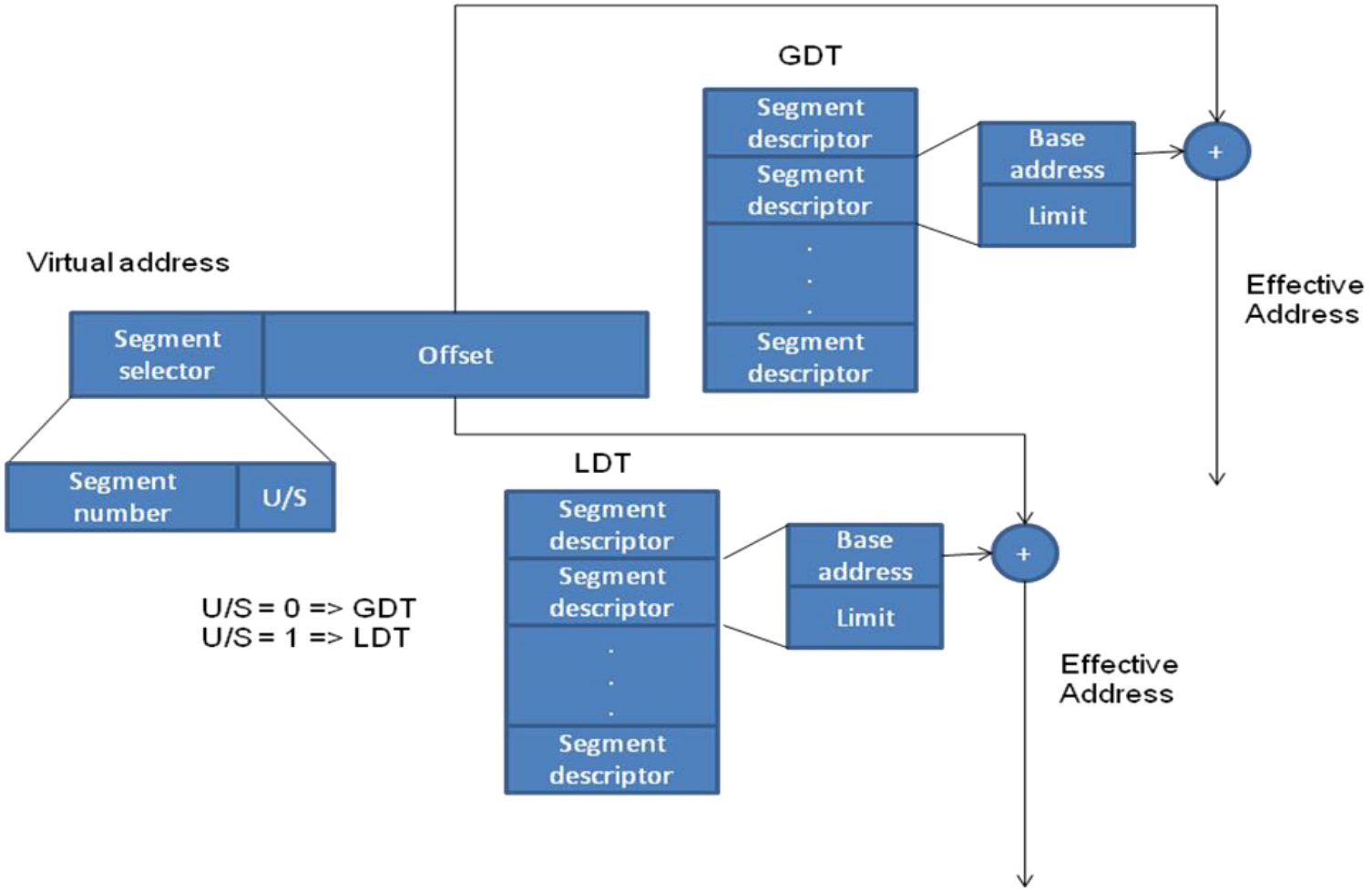
- Intel Pentium line of processors uses paged-segmentation.
- Approximately, a virtual address is a segment selector plus an offset
- Total segment space divided into two halves
  - *System* segments are common to all processes and are used by OS
  - *User* segments are unique to each process.
- Two descriptor tables
  - *Global Descriptor Table (GDT)* common to all processes
  - *Local Descriptor Table (LDT) for each process*
- A bit in the segment selector identifies whether the segment being named by the virtual address is a system or a user segment.
- Segment descriptor for the selected segment contains the details for translating the offset specified in the virtual address to a physical address.
- Choice
  - Use simple segmentation w/o any paging (compatible with earlier processors)
  - Use paged-segmentation.

Intel x86: Combining segmentation and paging

- **Use segments to manage logical units**
- **Use pages to partition segments into fixed-size chunks**
  - **Each segment has its own page table**
    - There is a page table per segment, rather than per user address space
  - **Memory allocation becomes easy once again**
    - no contiguous allocation, no external fragmentation



Intel's Memory Architecture



Unix

- Intended to be machine independent so its memory management schemes will vary
  - early Unix: variable partitioning with no virtual memory scheme
  - current implementations of UNIX make use of paged virtual memory

Unix

Unix use two separate schemes:

- paging system
- kernel memory allocator

Paging System and Kernel Memory Allocator

Paging system

provides a virtual memory capability that allocates page frames in main memory to processes

allocates page frames to disk block buffers

Kernel Memory Allocator

allocates memory for the kernel

UNIX Memory Management Parameters (page 1 of 2)

Page Table Entry

Page frame number
Refers to frame in real memory.

Age
Indicates how long the page has been in memory without being referenced. The length and contents of this field are processor dependent.

Copy on write
Set when more than one process shares a page. If one of the processes writes into the page, a separate copy of the page must first be made for all other processes that share the page. This feature allows the copy operation to be deferred until necessary and avoided in cases where it turns out not to be necessary.

Modify
Indicates page has been modified.

Reference
Indicates page has been referenced. This bit is set to 0 when the page is first loaded and may be periodically reset by the page replacement algorithm.

Valid
Indicates page is in main memory.

Protect
Indicates whether write operation is allowed.

Disk Block Descriptor

Swap device number
Logical device number of the secondary device that holds the corresponding page. This allows more than one device to be used for swapping.

Device block number
Block location of page on swap device.

Type of storage
Storage may be swap unit or executable file. In the latter case, there is an indication as to whether the virtual memory to be allocated should be cleared first.

UNIX Memory Management Parameters (page 2 of 2)

Page Frame Data Table Entry

Page state
Indicates whether this frame is available or has an associated page. In the latter case, the status of the page is specified: on swap device, in executable file, or DMA in progress.

Reference count
Number of processes that reference the page.

Logical device
Logical device that contains a copy of the page.

Block number
Block location of the page copy on the logical device.

Pfdata pointer
Pointer to other pfdata table entries on a list of free pages and on a hash queue of pages.

Swap-Use Table Entry

Reference count
Number of page table entries that point to a page on the swap device.

Page/storage unit number
Page identifier on storage unit.

Page Replacement

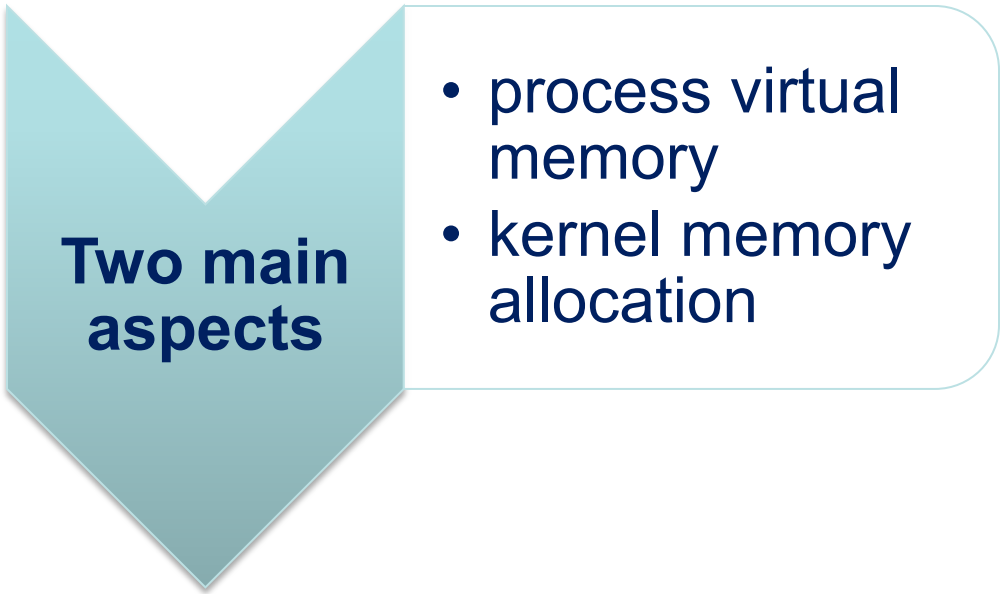
- The page frame data table is used for page replacement
- Pointers are used to create lists within the table
  - all available frames are linked together in a list of free frames available for bringing in pages
  - when the number of available frames drops below a certain threshold, the kernel will steal a number of frames to compensate

Kernel Memory Allocator

- The kernel generates and destroys small tables and buffers frequently during the course of execution, each of which requires dynamic memory allocation.
- Most of these blocks are significantly smaller than typical pages (therefore paging would be inefficient)
- Allocations and free operations must be made as fast as possible

Linux Memory Management

- **Shares many characteristics with Unix**



Linux Virtual Memory

- **Three level page table structure:**

Page directory	Page middle directory	Page table
process has a single page directory	may span multiple pages	may also span multiple pages
each entry points to one page of the page middle directory	each entry points to one page in the page table	each entry refers to one virtual page of the process
must be in main memory for an active process		

Kernel Memory Allocation

- **Kernel memory capability manages physical main memory page frames**
  - **primary function is to allocate and deallocate frames for particular uses**

Possible owners of a frame include:
• user-space processes • dynamically allocated kernel data • static kernel code • page cache

- A buddy algorithm is used so that memory for the kernel can be allocated and deallocated in units of one or more pages
- Page allocator alone would be inefficient because the kernel requires small short-term memory chunks in odd sizes
- Slab allocation
  - used by Linux to accommodate small chunks

Windows Memory Management

- Virtual memory manager controls how memory is allocated and how paging is performed
- Designed to operate over a variety of platforms
- Uses page sizes ranging from 4 Kbytes to 64 Kbytes

Windows Virtual Address Map

- **On 32 bit platforms each user process sees a separate 32 bit address space allowing 4 Gbytes of virtual memory per process**
  - by default half is reserved for the OS
- Large memory intensive applications run more effectively using 64-bit Windows
- Most modern PCs use the AMD64 processor architecture which is capable of running as either a 32-bit or 64-bit system

Windows Paging

- On creation, a process can make use of the entire user space of almost 2 Gbytes
- This space is divided into fixed-size pages managed in contiguous regions allocated on 64 Kbyte boundaries
- Regions may be in one of three states:

Resident Set Management System

- Windows uses variable allocation, local scope
- When activated, a process is assigned a data structure to manage its working set
- Working sets of active processes are adjusted depending on the availability of main memory

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

Summary

Scheme	Hardware Support	Still in Use?
User/Kernel Separation	Fence register	No
Fixed Partition	Bounds registers	Not in any production operating system
Variable-sized Partition	Base and limit registers	Not in any production operating system
Paged Virtual Memory	Page table and page table base register	Yes, in most modern operating system
Segmented Virtual Memory	Segment table, and segment table base register	Segmentation in this pure form not supported in any commercially popular processors
Paged-segmented Virtual Memory	Combination of the hardware for paging and segmentation	Yes, most modern operating systems based on Intel x86 use this scheme[1]