

Memory and Device Management

Main Memory

2

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging

Objectives

3

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
 - **Logical address** – generated by the CPU; also referred to as **virtual address**
 - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address
- Many methods possible, covered in the rest of this chapter
- To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
 - Base register now called **relocation register**
 - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
 - Execution-time binding occurs when reference is made to location in memory

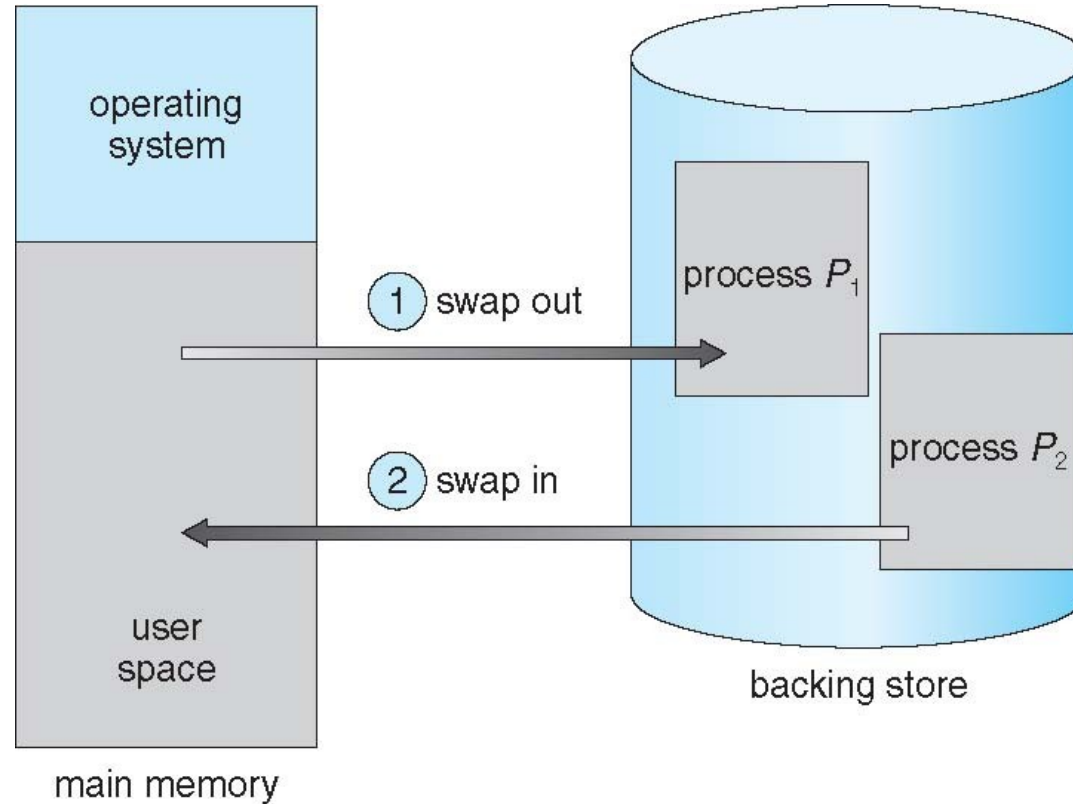
Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
 - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory

Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
 - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
 - Swapping normally disabled
 - Started if more than threshold amount of memory allocated
 - Disabled again once memory demand reduced below threshold

Schematic View of Swapping



Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
 - Resident operating system, usually held in low memory with interrupt vector
 - User processes then held in high memory
 - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
 - Base register contains value of smallest physical address
 - Limit register contains range of logical addresses – each logical address must be less than the limit register
 - MMU maps logical address *dynamically*
 - Can then allow actions such as kernel code being **transient** and kernel changing size

Dynamic Storage-Allocation Problem

How to satisfy a request of size n from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
 - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
 - Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- First fit analysis reveals that given N blocks allocated, $0.5 N$ blocks lost to fragmentation
 - $1/3$ may be unusable -> **50-percent rule**

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
 - Shuffle memory contents to place all free memory together in one large block
 - Compaction is possible *only* if relocation is dynamic, and is done at execution time
 - I/O problem
 - Latch job in memory while it is involved in I/O
 - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

Segmentation

Memory-management scheme that supports user view of memory

A program is a collection of segments

A segment is a logical unit such as:

- main program

- procedure

- function

- method

- object

- local variables, global variables

- common block

- stack

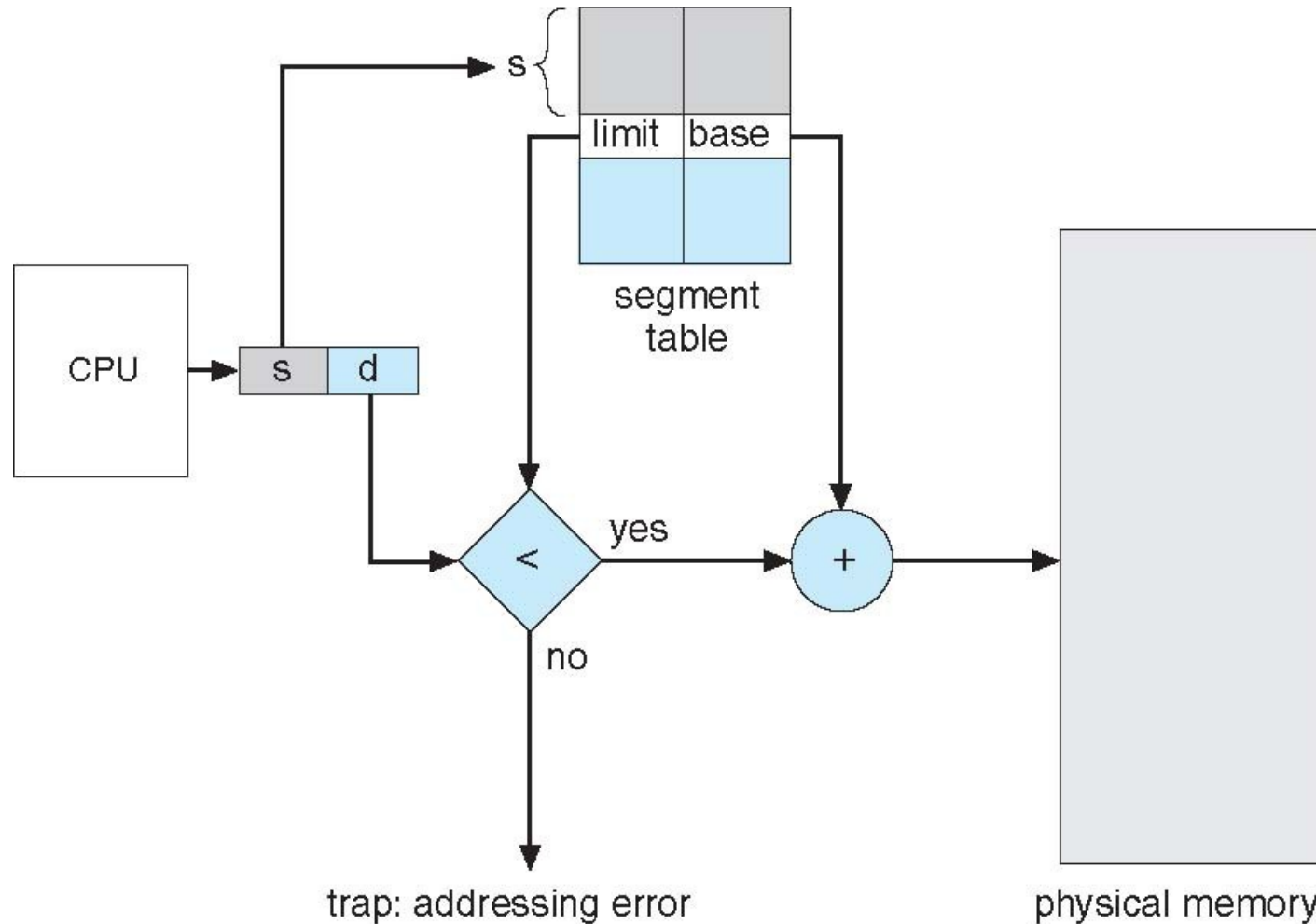
- symbol table

- arrays

Segmentation Architecture (Cont.)

- Protection
 - With each entry in segment table associate:
 - validation bit = 0 $\Rightarrow$ illegal segment
 - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

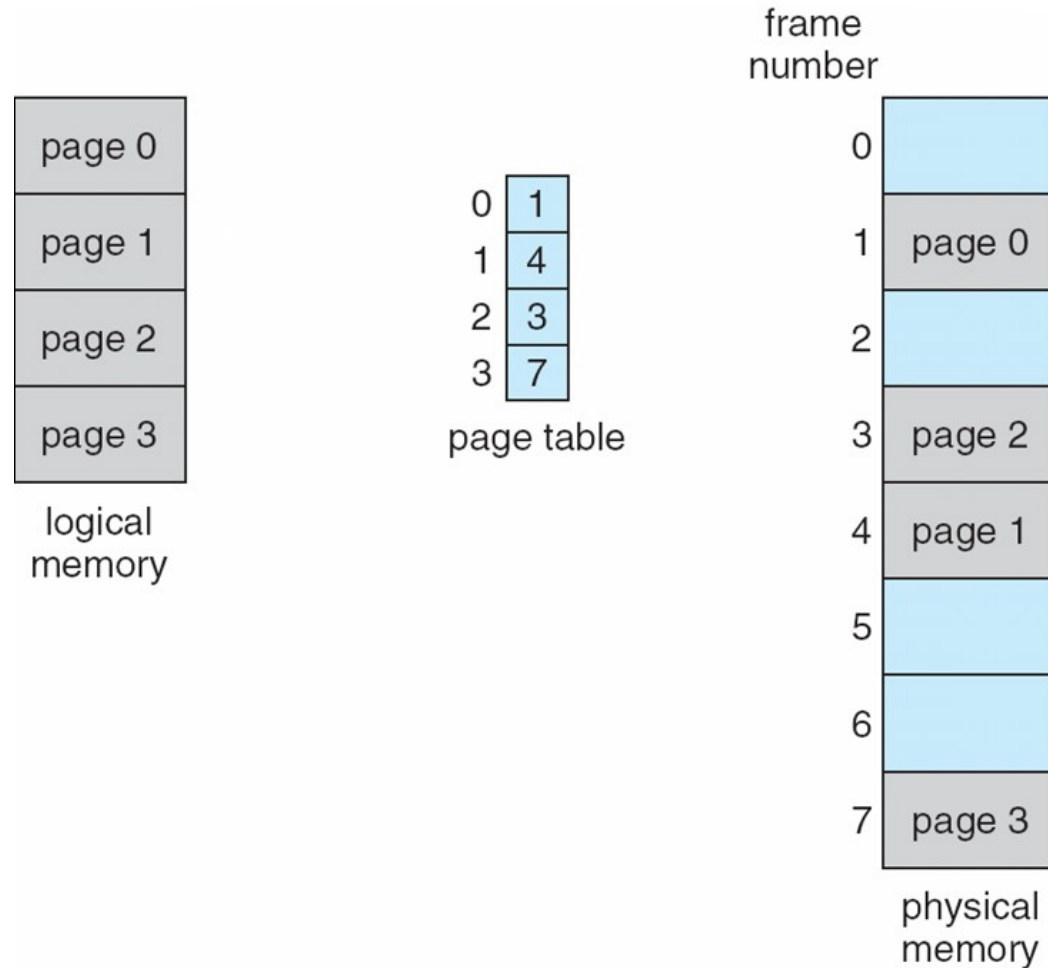
Segmentation Hardware



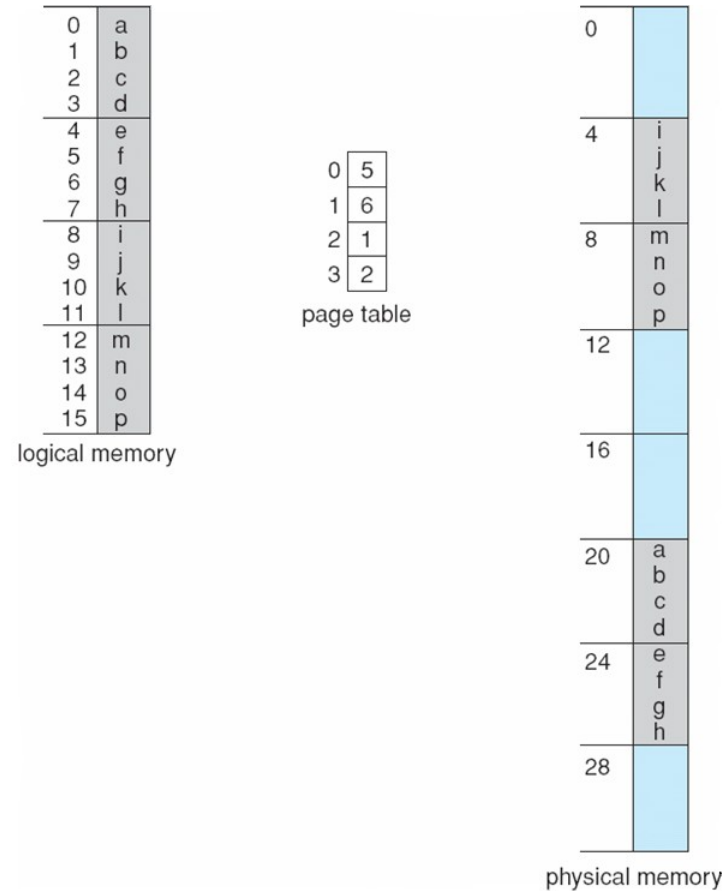
Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
 - Avoids external fragmentation
 - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
 - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have internal fragmentation

Paging Model of Logical and Physical Memory



Paging Example



$n=2$ and $m=4$ 32-byte memory and 4-byte pages

Paging (Cont.)

Calculating internal fragmentation

Page size = 2,048 bytes

Process size = 72,766 bytes

35 pages + 1,086 bytes

Internal fragmentation of $2,048 - 1,086 = 962$ bytes

Worst case fragmentation = 1 frame - 1 byte

On average fragmentation = $1 / 2$ frame size

So small frame sizes desirable?

But each page table entry takes memory to track

Page sizes growing over time

Solaris supports two page sizes - 8 KB and 4 MB

Process view and physical memory now very different

By implementation process can only access its own memory

Structure of the Page Table

- Memory structures for paging can get huge using straight-forward methods
 - Consider a 32-bit logical address space as on modern computers
 - Page size of 4 KB (2^{12})
 - Page table would have 1 million entries ($2^{32} / 2^{12}$)
 - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
 - That amount of memory used to cost a lot
 - Don't want to allocate that contiguously in main memory
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Objectives

- To describe the benefits of a virtual memory system
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames
- To discuss the principle of the working-set model
- To examine the relationship between shared memory and memory-mapped files

Virtual Memory

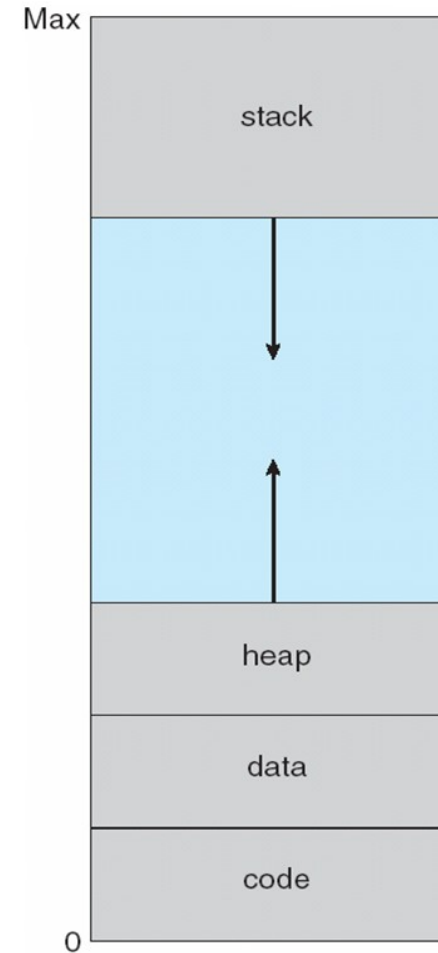
- **Virtual memory** – separation of user logical memory from physical memory
 - Only part of the program needs to be in memory for execution
 - Logical address space can therefore be much larger than physical address space
 - Allows address spaces to be shared by several processes
 - Allows for more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

Virtual Memory (Cont.)

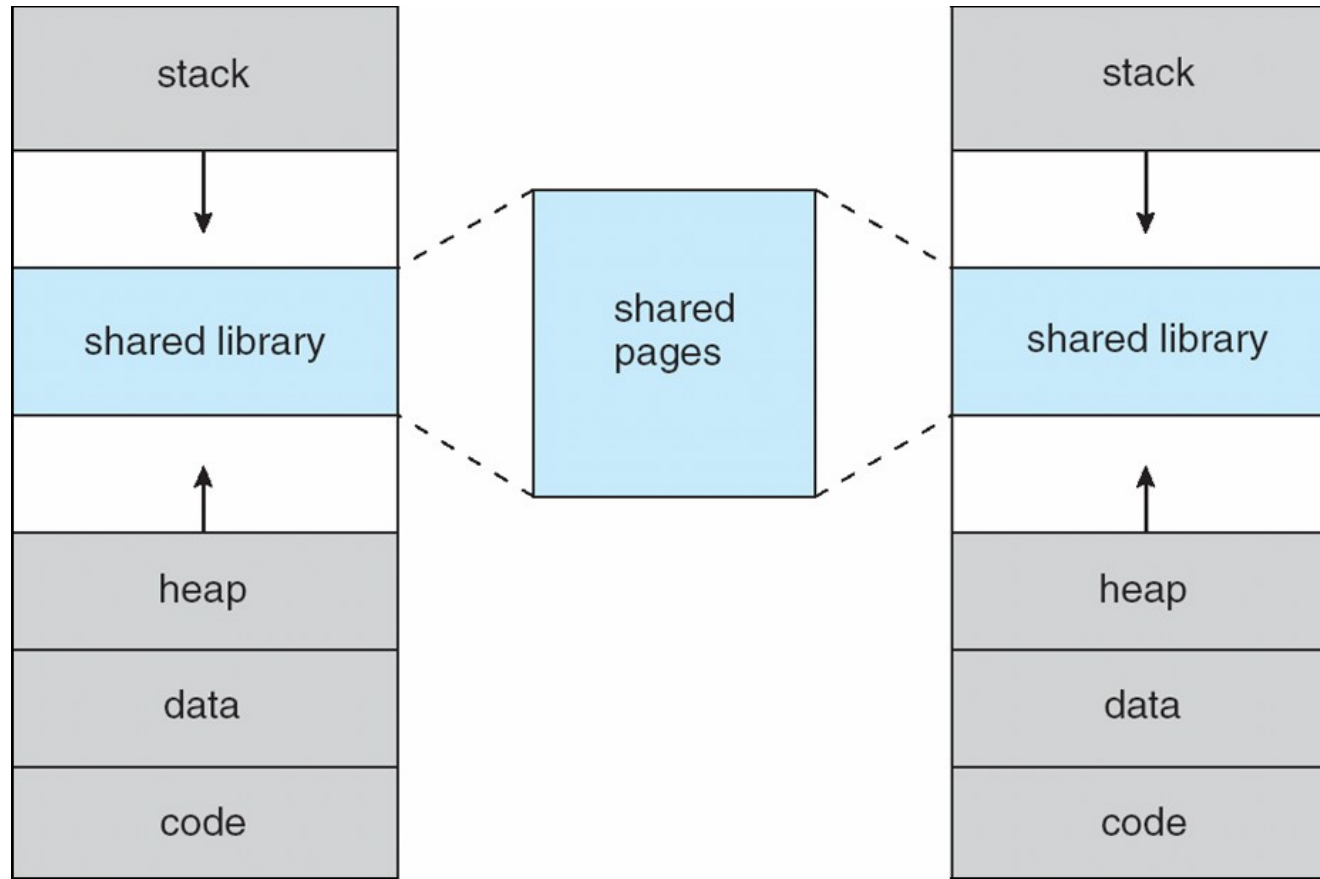
- **Virtual address space** – logical view of how process is stored in memory
 - Usually start at address 0, contiguous addresses until end of space
 - Meanwhile, physical memory organized in page frames
 - MMU must map logical to physical
- Virtual memory can be implemented via:
 - Demand paging
 -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

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



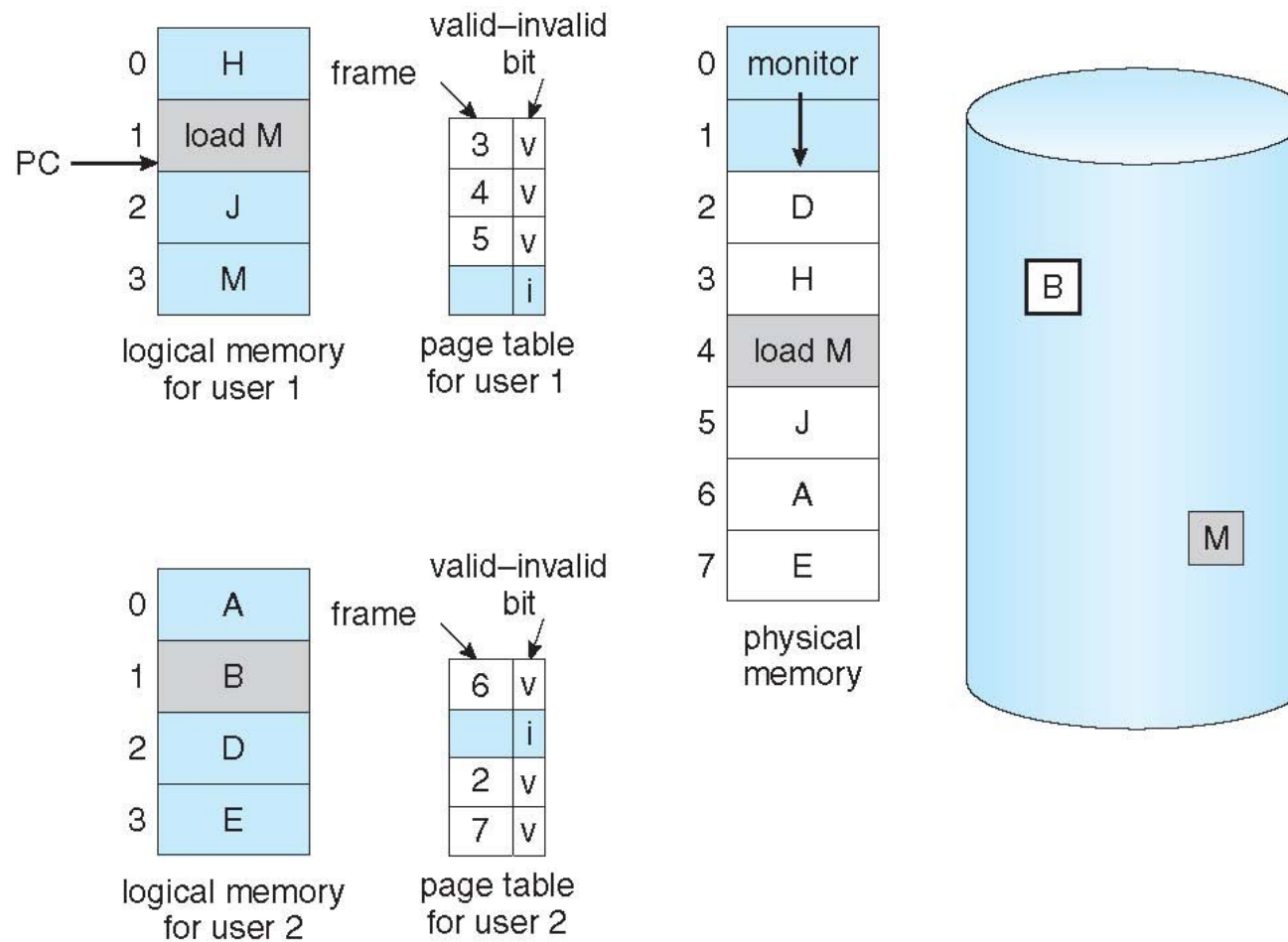
Shared Library Using Virtual Memory



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

Need For Page Replacement

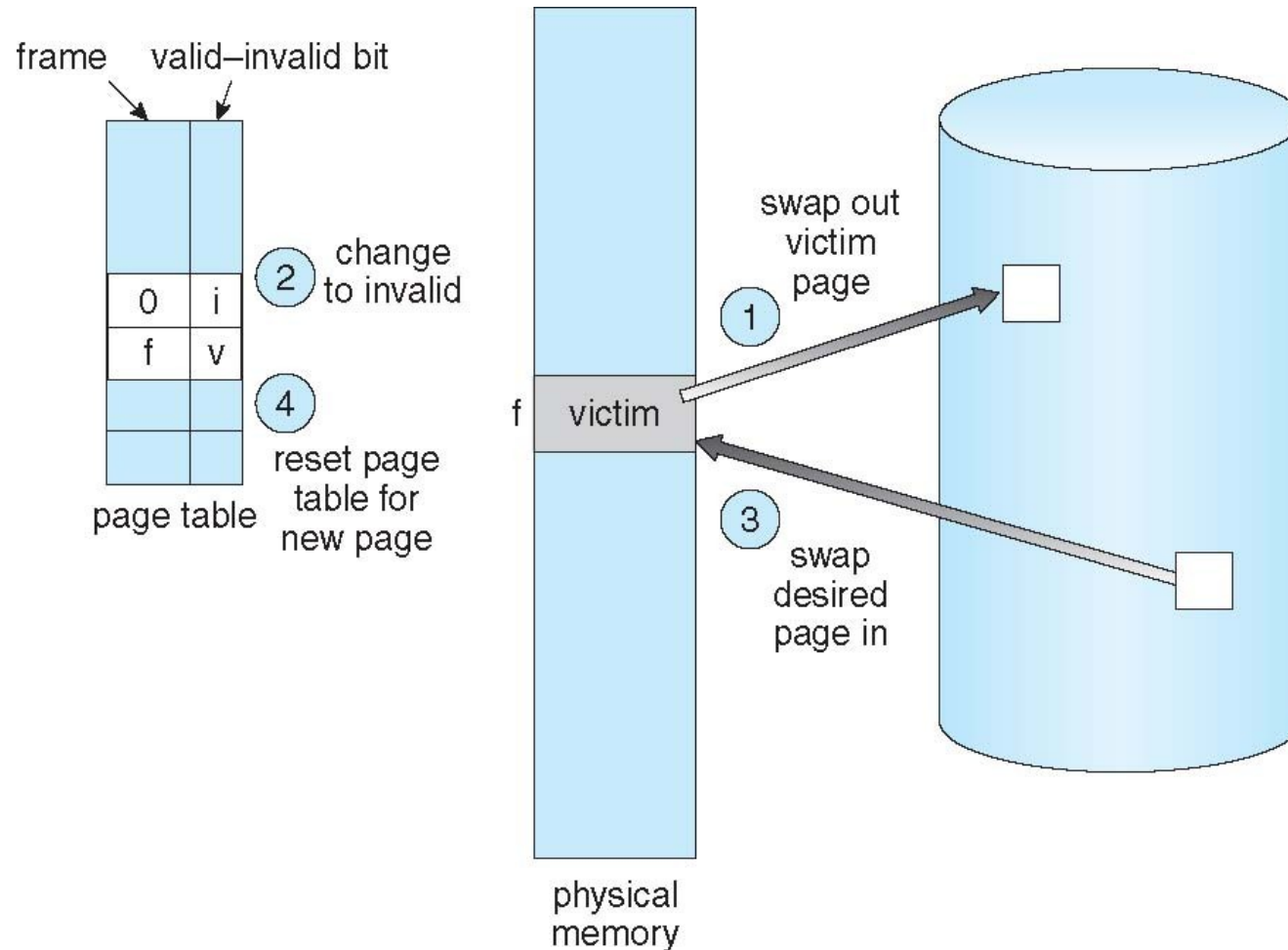


Basic Page Replacement

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 frame**
 - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

Page Replacement



Page and Frame Replacement Algorithms

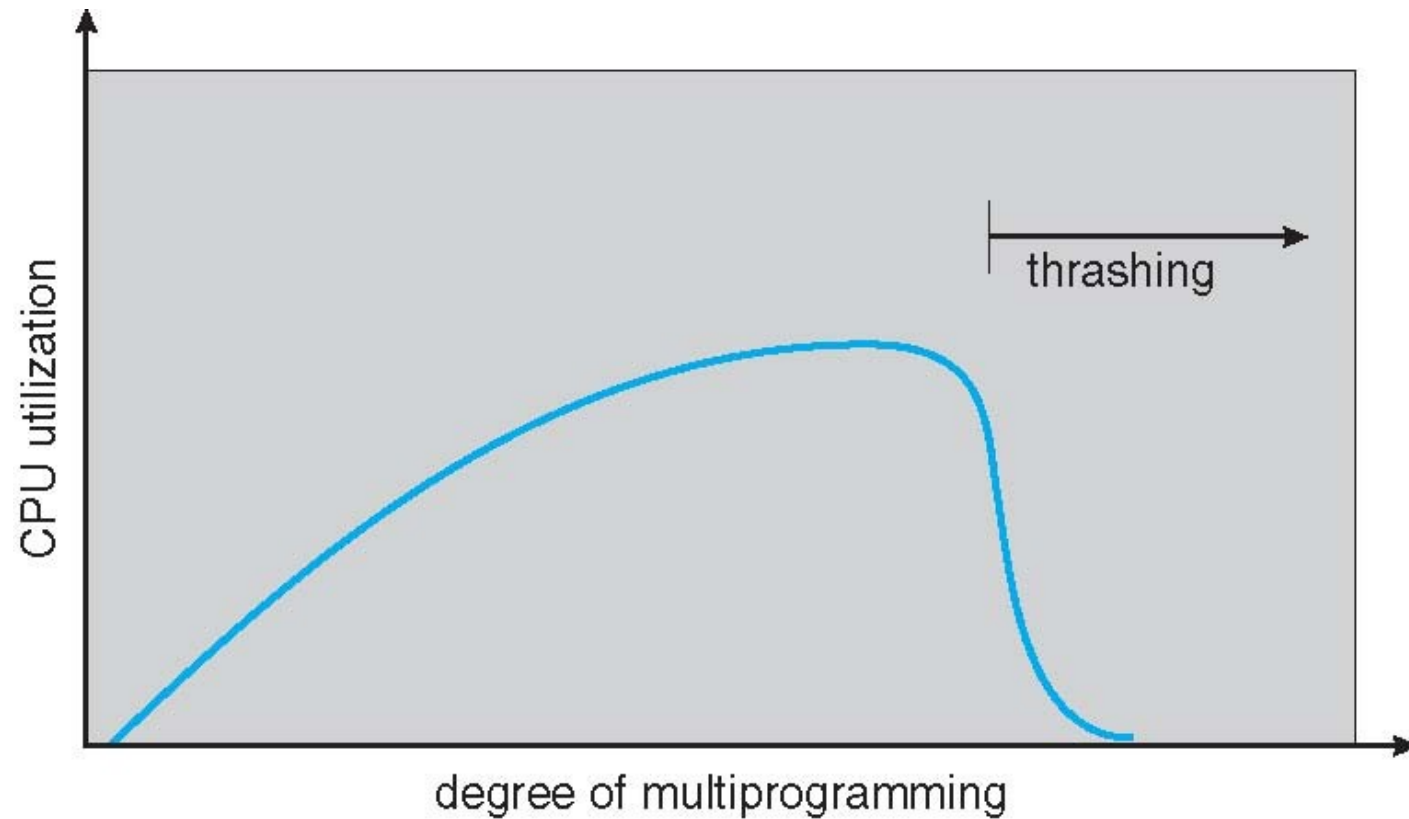
- **Frame-allocation algorithm** determines
 - How many frames to give each process
 - Which frames to replace
- **Page-replacement algorithm**
 - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
 - String is just page numbers, not full addresses
 - Repeated access to the same page does not cause a page fault
 - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0 1

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
 - Page fault to get page
 - Replace existing frame
 - But quickly need replaced frame back
 - This leads to:
 - Low CPU utilization
 - Operating system thinking that it needs to increase the degree of multiprogramming
 - Another process added to the system
- **Thrashing** $\equiv$ a process is busy swapping pages in and out

Thrashing (Cont.)



Operating System Examples



- Windows
- Solaris

Windows

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page
- Processes are assigned **working set minimum** and **working set maximum**
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold, **automatic working set**

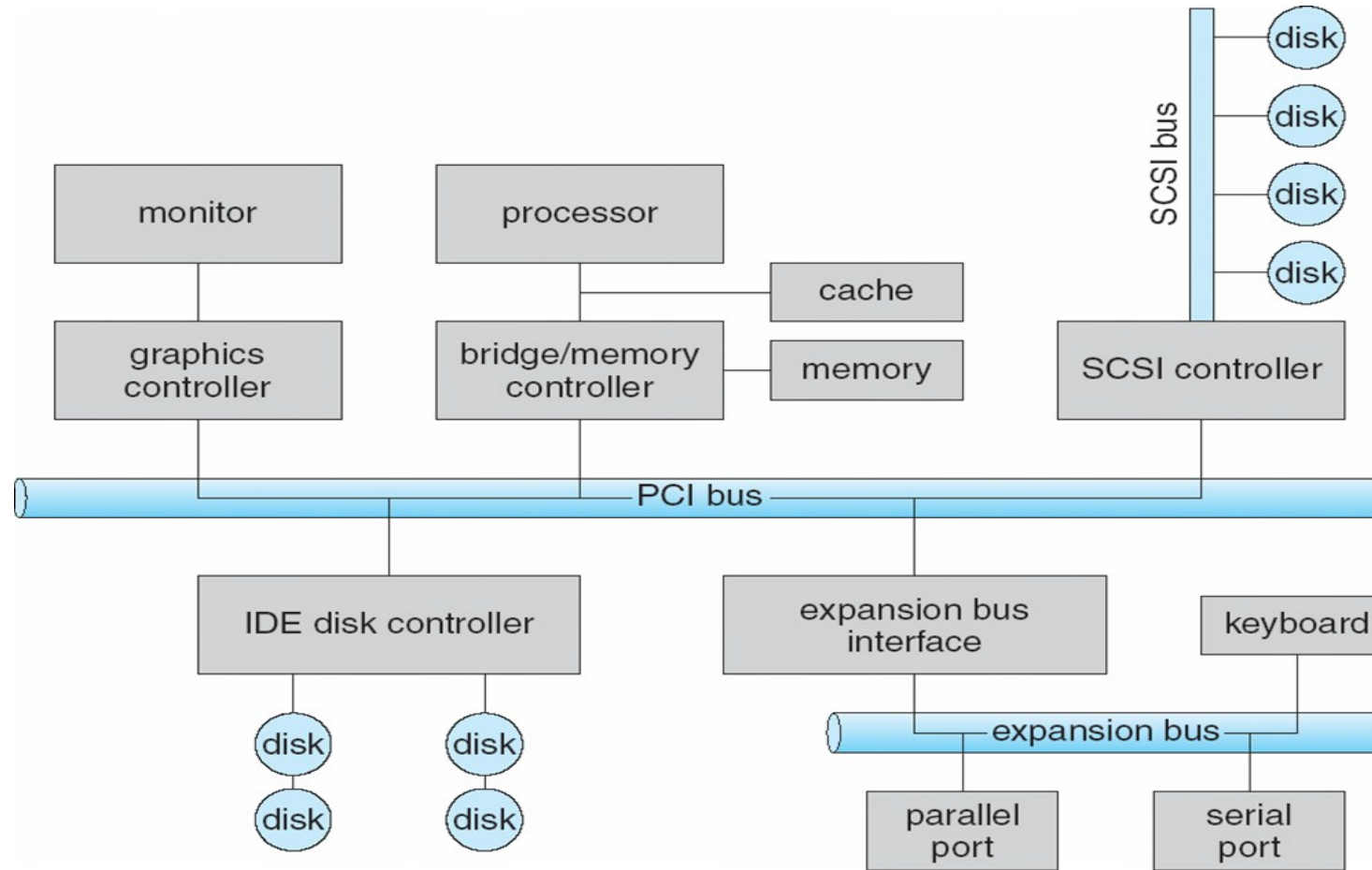
Solaris

- Maintains a list of free pages to assign faulting processes
- **Lotsfree** – threshold parameter (amount of free memory) to begin paging
- **Desfree** – threshold parameter to increasing paging
- **Minfree** – threshold parameter to being swapping
- Paging is performed by **pageout** process
- **Pageout** scans pages using modified clock algorithm
- **Scanrate** is the rate at which pages are scanned. This ranges from **slowscan** to **fastscan**
- **Pageout** is called more frequently depending upon the amount of free memory available
- **Priority paging** gives priority to process code pages

Device Management

- Incredible variety of I/O devices
 - Storage
 - Transmission
 - Human-interface
- Common concepts – signals from I/O devices interface with computer
 - **Port** – connection point for device
 - **Bus - daisy chain** or shared direct access
 - **PCI** bus common in PCs and servers, PCI Express (**PCIe**)
 - **expansion bus** connects relatively slow devices
 - **Controller (host adapter)** – electronics that operate port, bus, device
 - Sometimes integrated
 - Sometimes separate circuit board (host adapter)
 - Contains processor, microcode, private memory, bus controller, etc
 - Some talk to per-device controller with bus controller, microcode, memory, etc

A Typical PC Bus Structure



Device Management (Cont.)

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
 - Data-in register, data-out register, status register, control register
 - Typically 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
 - Direct I/O instructions
 - **Memory-mapped I/O**
 - Device data and command registers mapped to processor address space

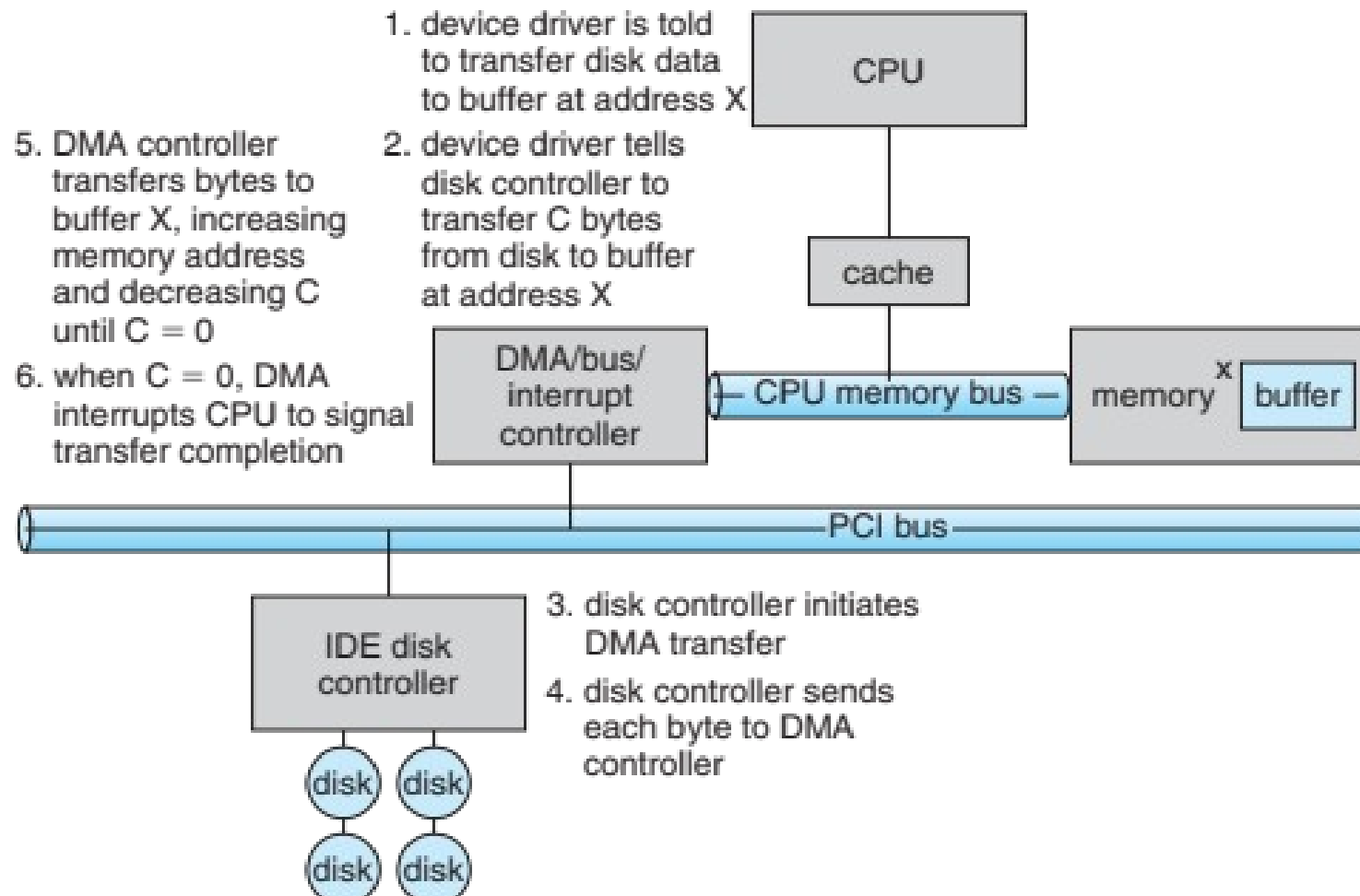
Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Direct Memory Access

- Used to avoid **programmed I/O** (one byte at a time) for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
 - Source and destination addresses
 - Read or write mode
 - Count of bytes
 - Writes location of command block to DMA controller
 - Bus mastering of DMA controller – grabs bus from CPU
 - **Cycle stealing** from CPU but still much more efficient
 - When done, interrupts to signal completion

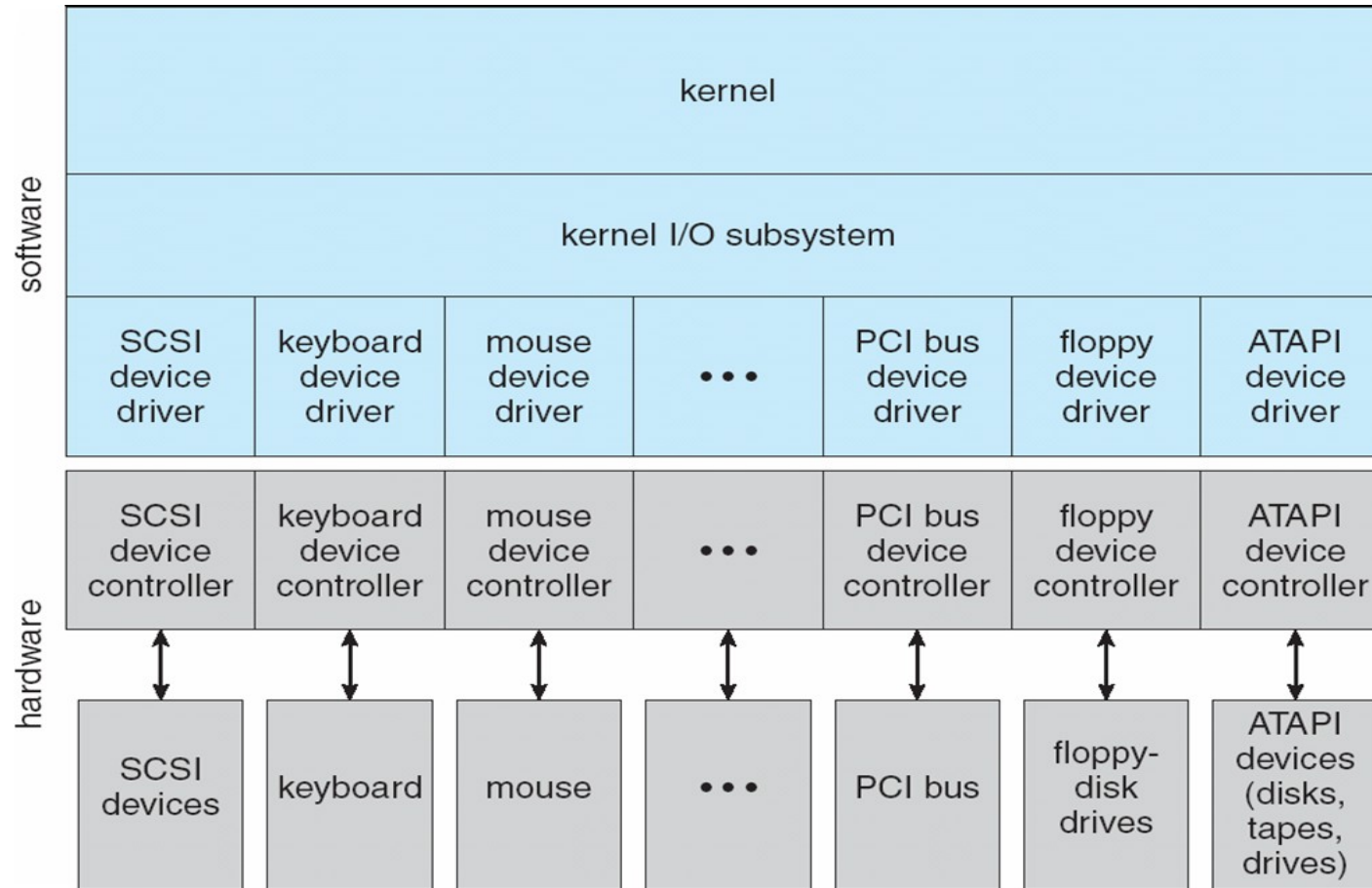
Six Step Process to Perform DMA Transfer



Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
 - **Character-stream** or **block**
 - **Sequential** or **random-access**
 - **Synchronous** or **asynchronous** (or both)
 - **Sharable** or **dedicated**

A Kernel I/O Structure



Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Network Devices

- Varying enough from block and character to have own interface
- Linux, Unix, Windows and many others include **socket** interface
 - Separates network protocol from network operation
 - Includes **select()** functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
 - Retry a read or write, for example
 - Some systems more advanced – Solaris FMA, AIX
 - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
 - All I/O instructions defined to be privileged
 - I/O must be performed via system calls
 - Memory-mapped and I/O port memory locations must be protected too