# Input/Output

Operating Systems

Assoc. Prof. Milos Jovanovik, PhD

# I/O System

- Principles of I/O hardware
- Principles of I/O software
- I/O software levels

# I/O Devices

- **Block devices**: HD, CD-ROM, USB memory.
  - Transfer of blocks with fixed size (512B-64KB), each block with its own address, used independently;
- **Character devices**: printers, network interface, mouse.
  - Deliver or accept a stream of characters, without regard to any block structure;
  - Non-addressable and no seek operation;

# Some Typical Data Rates

| Device | Data rate |
|---|---|
| Keyboard | 10 bytes/sec |
| Mouse | 100 bytes/sec |
| 56K modem | 7 KB/sec |
| Scanner at 300 dpi | 1 MB/sec |
| Digital camcorder | 3.5 MB/sec |
| 4x Blu-ray disc | 18 MB/sec |
| 802.11n Wireless | 37.5 MB/sec |
| USB 2.0 | 60 MB/sec |
| FireWire 800 | 100 MB/sec |
| Gigabit Ethernet | 125 MB/sec |
| SATA 3 disk drive | 600 MB/sec |
| USB 3.0 | 625 MB/sec |
| SCSI Ultra 5 bus | 640 MB/sec |
| Single-lane PCIe 3.0 bus | 985 MB/sec |
| Thunderbolt 2 bus | 2.5 GB/sec |
| SONET OC-768 network | 5 GB/sec |

# Transfer Rates

| | | | |
|---|---|---|---|
| 100 Gigabit Ethernet (100GBASE-X) 10×/4× | 100 Gbit/s | 12.5 GB/s | 2010/2018 |
| Infiniband EDR 12× | 300 Gbit/s | 37.5 GB/s | 2014 |
| IEEE 802.11ad (max theoretical speed) | 7.14–7.2 Gbit/s | 892.5–900 MB/s | 2011 |
| PCI Express 3.0 (×8 link) | 64 Gbit/s | 7.877 GB/s | 2011 |
| PCI Express 1.0 (×32 link) | 80 Gbit/s | 8 GB/s | 2001 |
| HyperTransport 3.1 (3.2 GHz, 32-pair) | 409.6 Gbit/s | 51.2 GB/s | 2008 |
| NVLink | 640 Gbit/s | 80 GB/s | 2016 |
| SATA revision 3.2 – SATA Express | 16000 Mbit/s | 2000 MB/s | 2013 |
| Fibre Channel 32GFC (28.05 GHz) | 26424 Mbit/s | 3303 MB/s | 2016 |
| USB 3.1 SuperSpeed+ | 10 Gbit/s | 1250 MB/s | 2013 |
| Thunderbolt 3 | 40 Gbit/s | 5000 MB/s | 2015 |
| PC3-21300 DDR3 SDRAM | 170.4 Gbit/s | 21.3 GB/s | |
| PC3-24000 DDR3 SDRAM | 192 Gbit/s | 24 GB/s | |
| PC4-25600 DDR4 SDRAM | 204.8 Gbit/s | 25.6 GB/s | |
| DisplayPort 1.3 (4-lane High Bit Rate 3) | 32.4 Gbit/s | 4.05 GB/s | 2014 |
| superMHL | 36 Gbit/s | 4.5 GB/s | 2015 |
| Thunderbolt 3 | 40 Gbit/s | 5 GB/s | 2015 |
| HDMI 2.1 | 48 Gbit/s | 6 GB/s | 2017 |

# Device Controllers

- I/O devices consist of:
  - Mechanical components
  - Electronic components
- The electronic component (adapter) is the device controller
  - It can manage multiple devices as well
  - IDE, SATA, SCSI, USB interface
- Functions of the controller
  - Serial-parallel data conversion
  - Error detection and correction

# Control Registers

- The controllers have control registers for communication with the CPU.
- Typically there are 4 registers:
  - Status (busy? data ready? error?)
  - Control (command for execution)
  - Data in (data sent from the device to the CPU)
  - Data out (data sent from the CPU to the device)

# Access Modes

- CPU can access the control registers by use of special I/O instructions (IN/OUT in Intel assembler) or by memory mapping
  - Memory mapping is performed by the memory controller and is faster than explicit I/O instructions
- In the controller interaction, the CPU can wait for response by polling the status registers, i.e., by periodical checking whether the status of the device has changed

# Memory Mapped I/O

- The control registers are mapped in the memory address space
- Every control register is assigned a single memory address which is not assigned to the memory
  - Usually, these addresses are at the beginning of the memory address space
  - Hybrid schemes have memory mapped I/O data buffers and special I/O ports for control registers

# Memory Mapped I/O

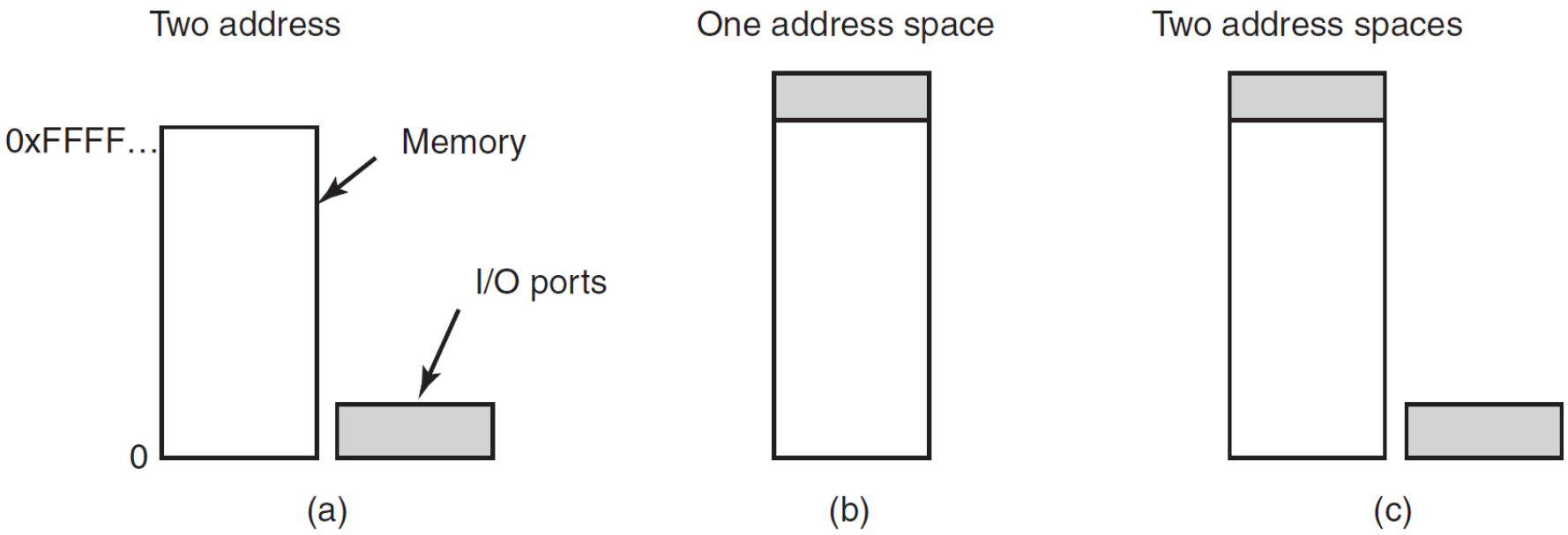



**Figure 5-2.** (a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.

# Advantages

- Drivers can be written in C
  - It is not necessary to use assembly language
- Protection mechanism
  - The assigned memory space of the control registers will never be assigned to any virtual address space
- The instructions used for memory access, can also be used in this case (e.g. TEST)
  - Reduced number of instructions

# Disadvantages

- Caching problems: is the device ready?
  - The problem is suppressed by selective disabling of caching --> Additional complexity
- A problem in determining whether a memory reference is aimed for the memory, or for a memory mapped I/O

# Architectures

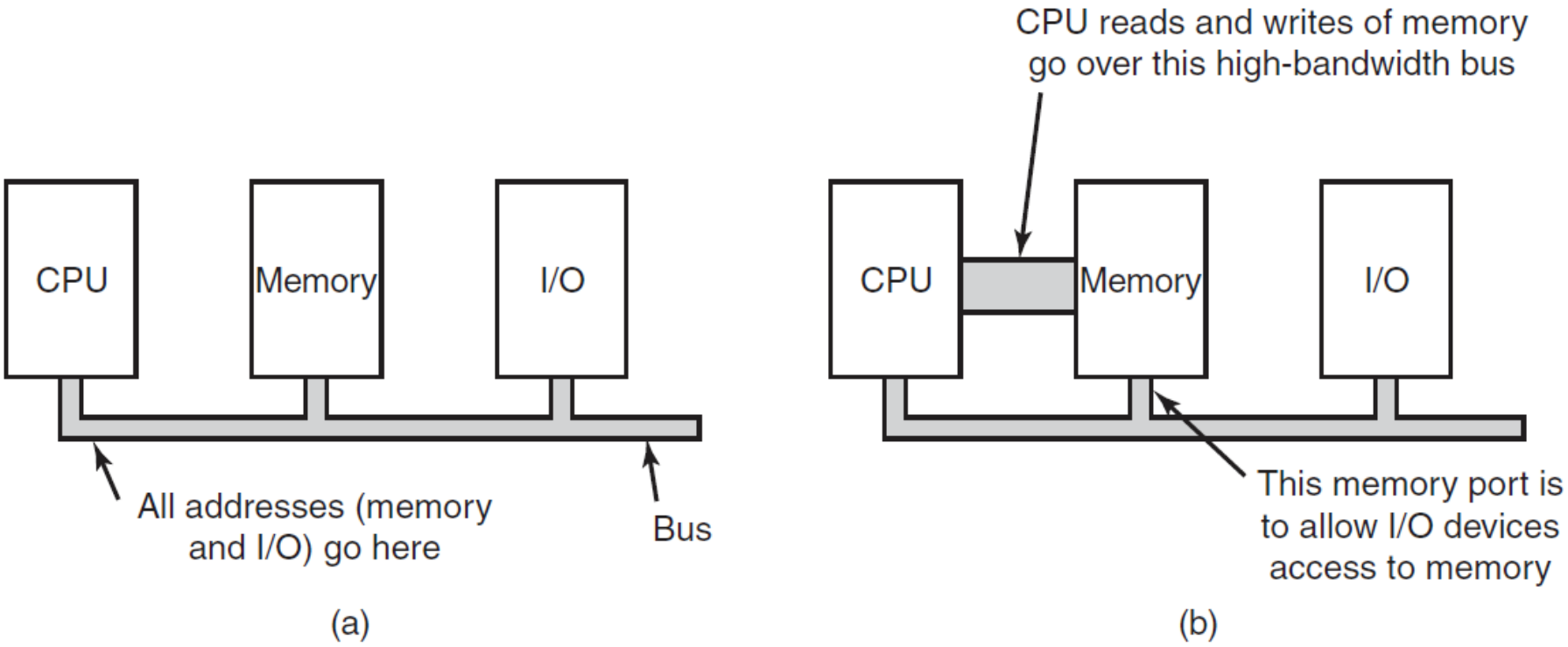
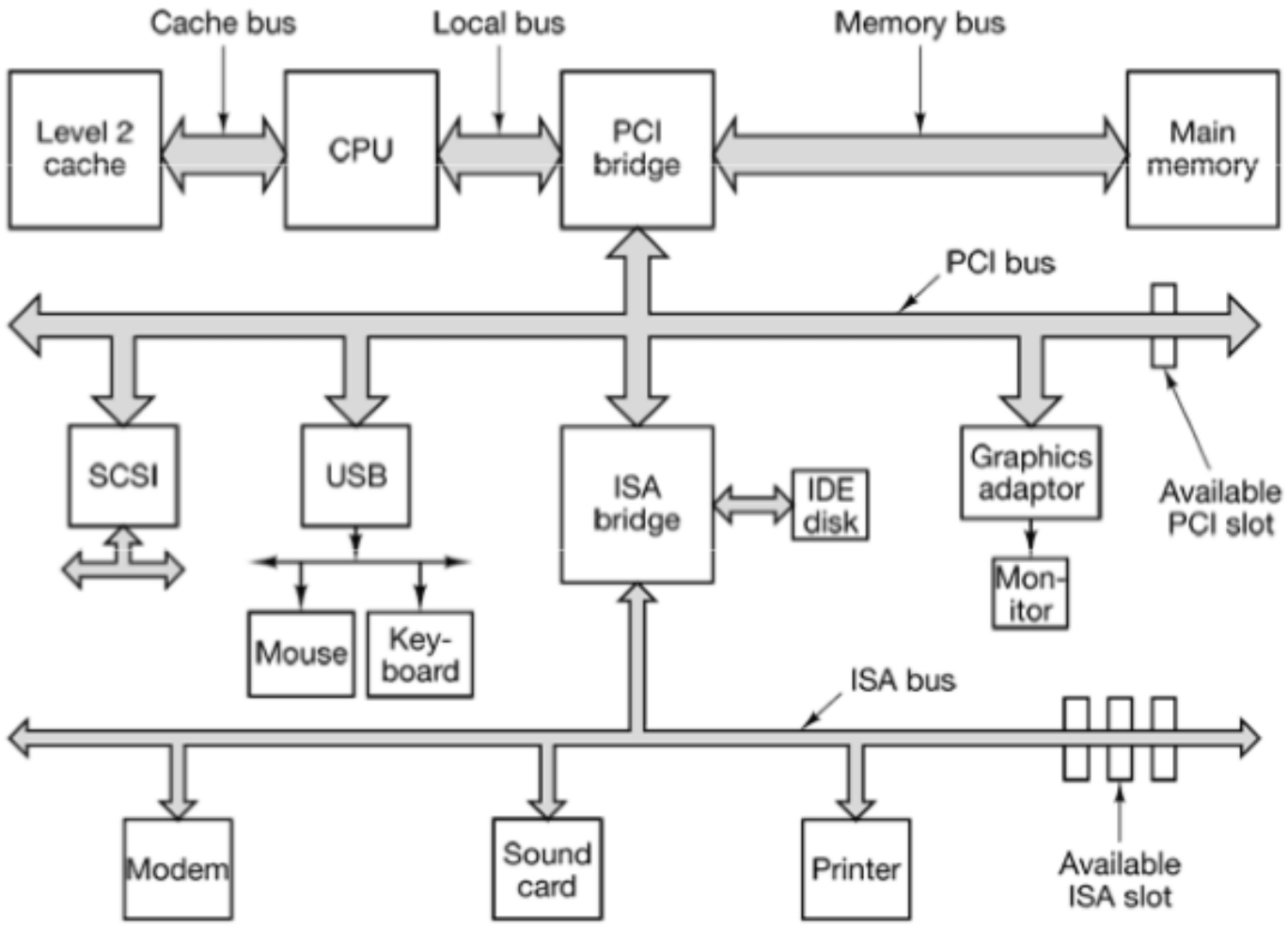
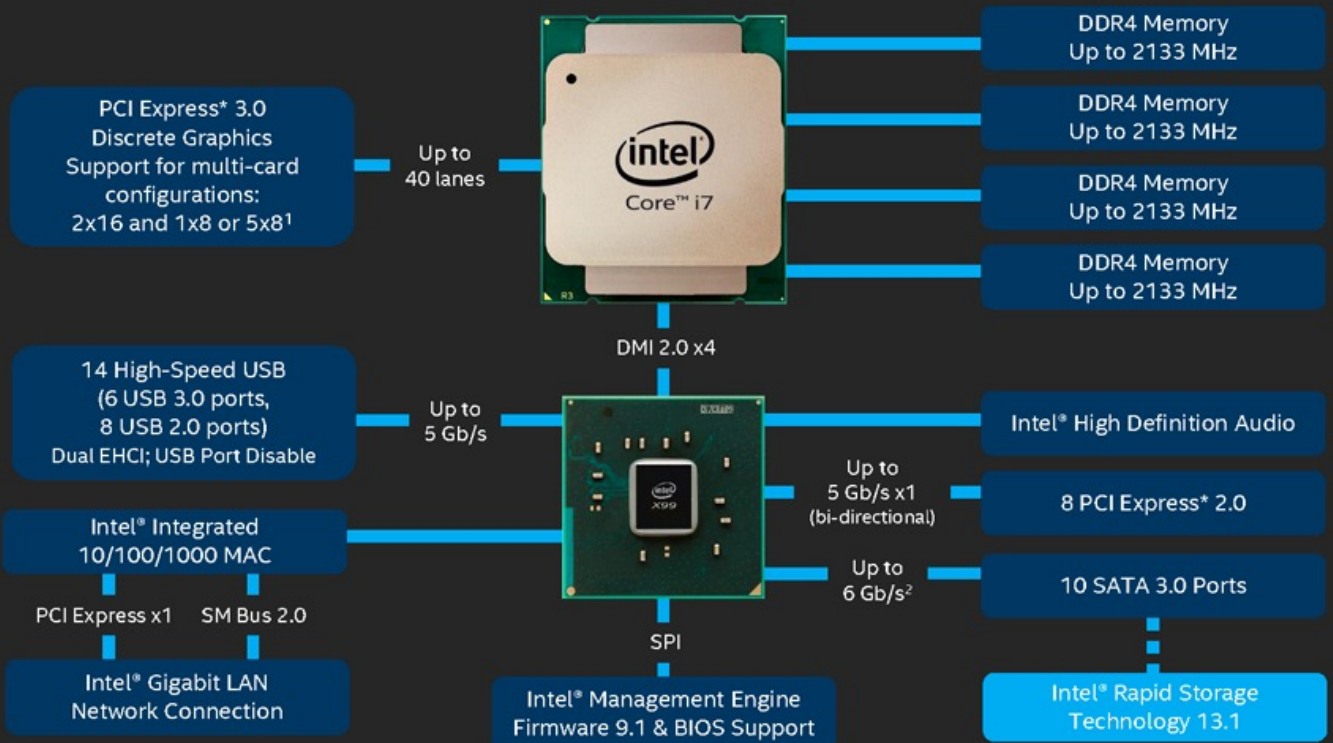



**Figure 5-3.** (a) A single-bus architecture. (b) A dual-bus memory architecture.

# Structure of the Pentium System



# Intel® Core™ i7 High End Desktop Platform Overview



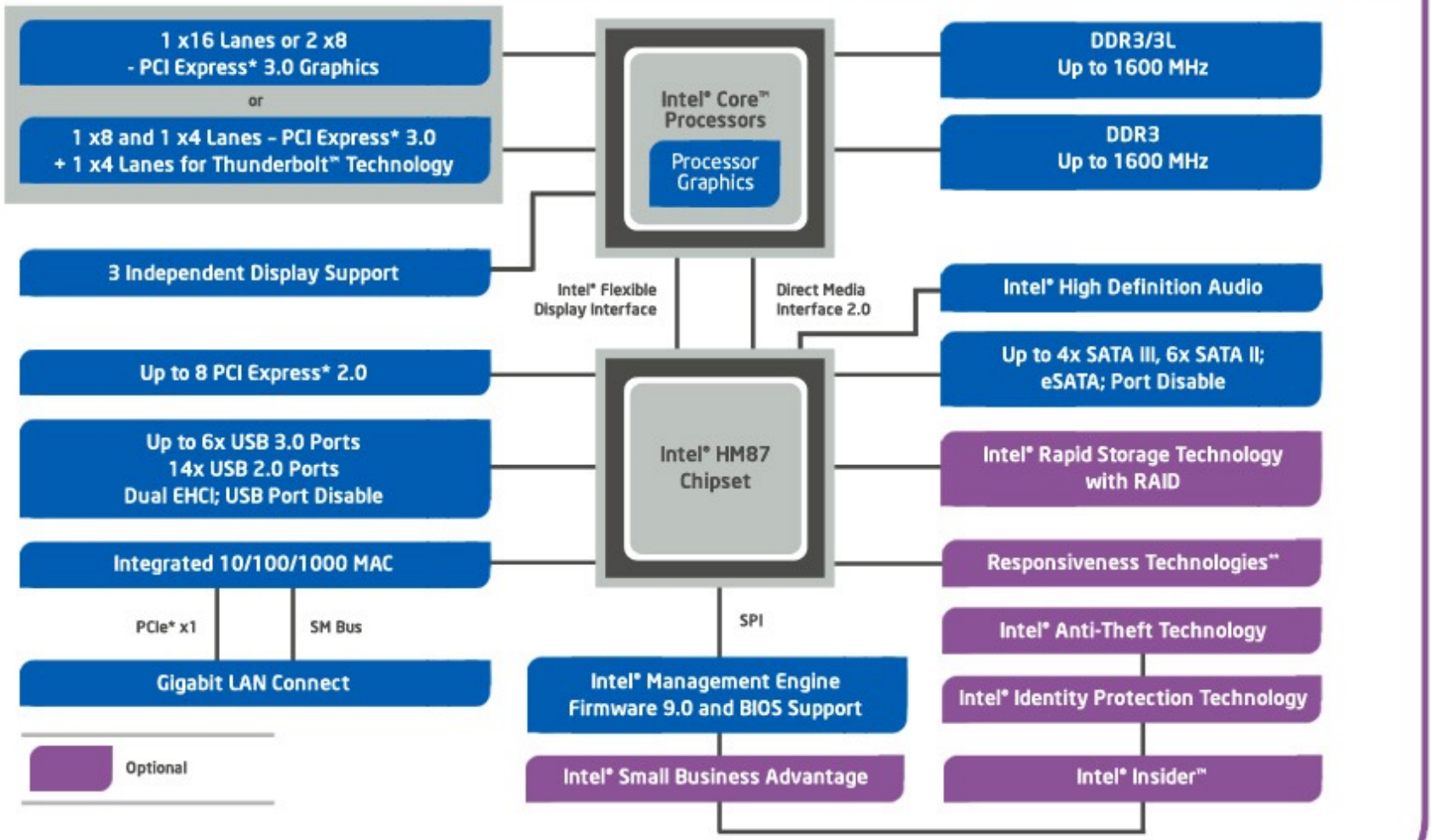

[1] 3 slots available, but need additional logic on board to support more slots. 5x8 configuration requires additional system clocks to be provided by 3rd party components.

[2] All SATA ports capable of 6 Gb/s.

Under Embargo until
**9:00am PST, August 29, 2014**

intel Look Inside.™

# Mobile Intel® HM87 Chipset Block Diagram




**Includes Intel® Smart Response Technology, Intel® Rapid Start Technology, and Intel® Smart Connect Technology.

# Device Communication Types

1. **Programmed I/O**
   - One byte/word per time unit
   - Uses polling --> wastes CPU cycles
   - Good for devices for special use: microprocessor-controlled devices
2. **Interrupt-driven I/O**
   - Many I/O use this approach as a good alternative of polling
3. **Direct Memory Access (DMA)**
   - Used for block transfer
   - Minimal CPU participation: only at the beginning and at the end of the transfer

# Programmed I/O

Polling:

- CPU busy-waits until the controller status becomes idle
- CPU sets the command register (status ready) and the data (in case of an output operation)
- The controller acts on the ready status & sets status busy
- The controller reads the command from the command register and executes it by sending / receiving the data
- Upon completion of the operation, the controller changes the status to idle
- CPU reads the new status and reads the data (in case of a read operation)

The reading of the data has to be very fast, otherwise the data can be lost (modem, keyboard, gaming controller, etc.)

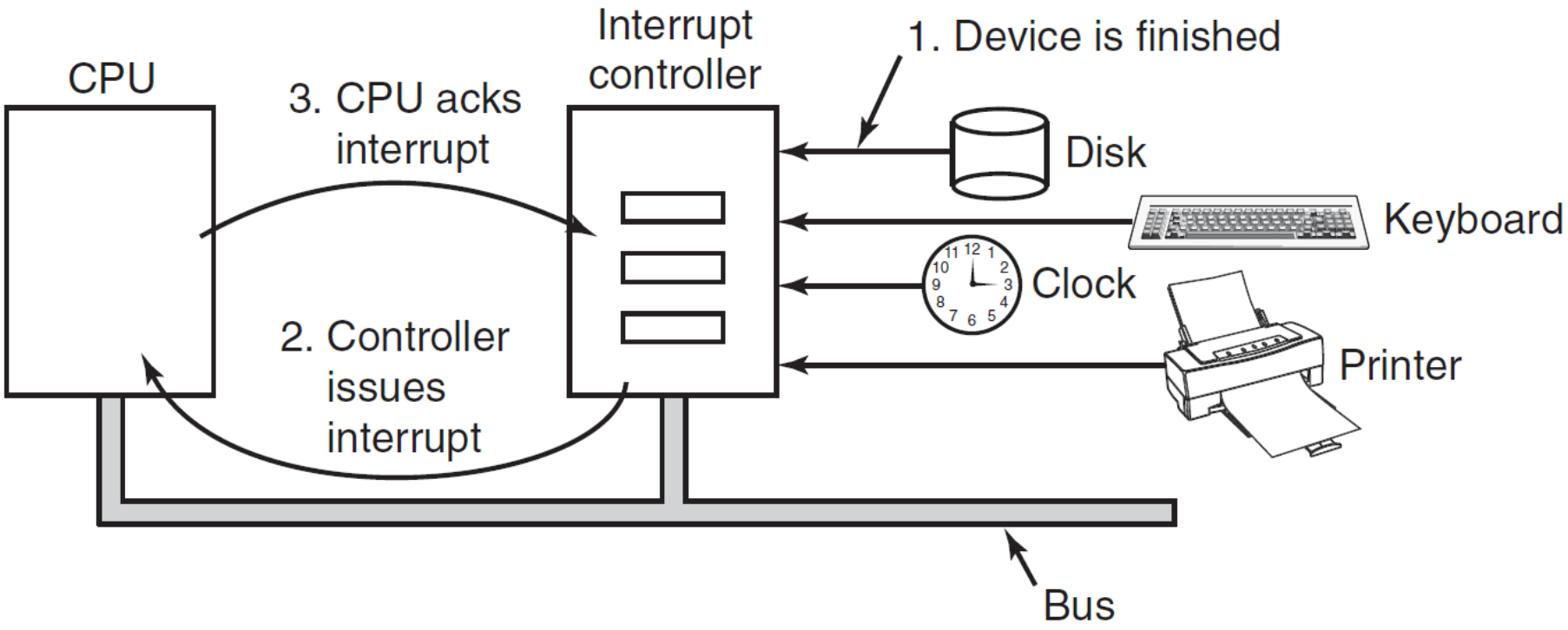
# Interrupt-Driven I/O

Interrupts:

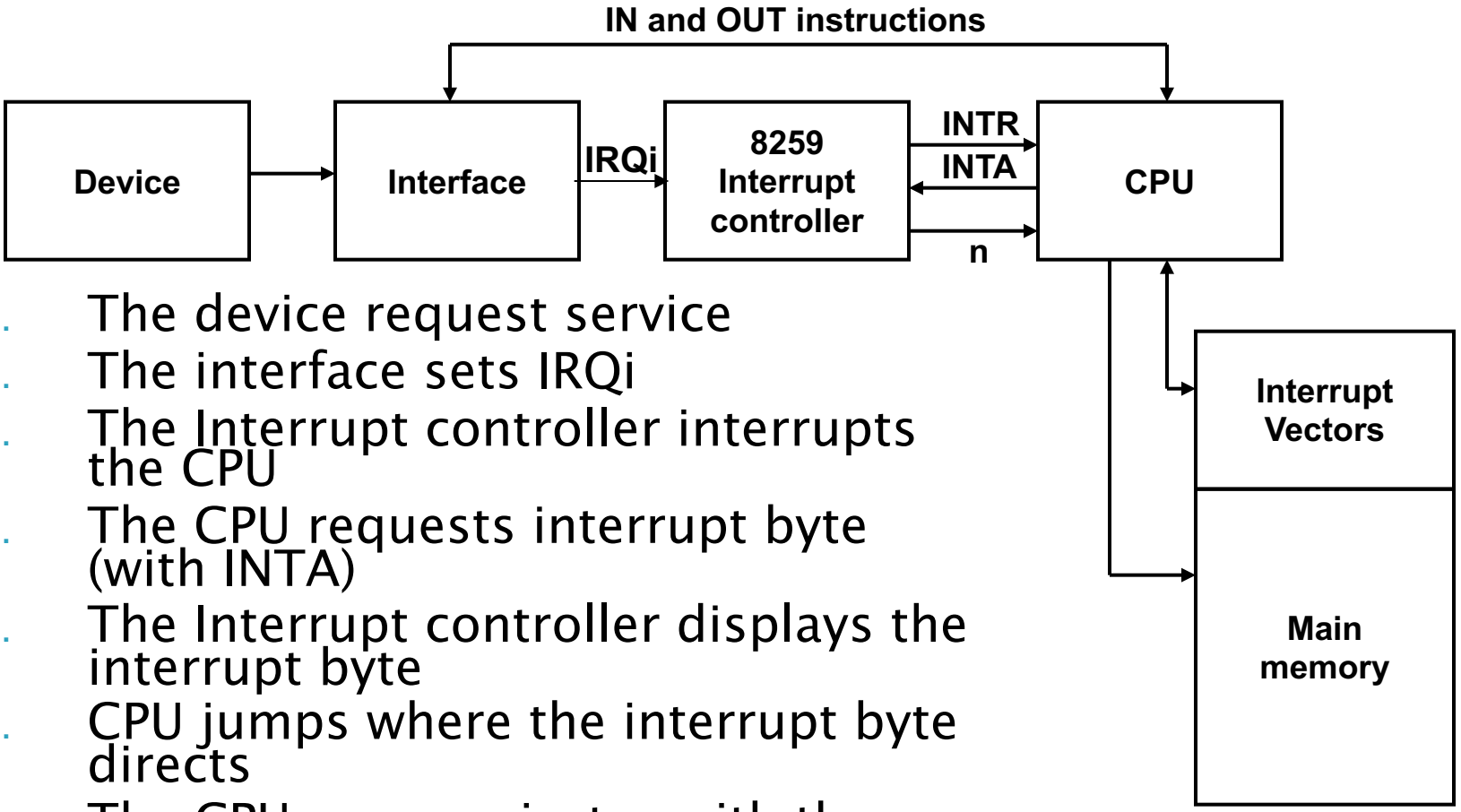
- Instead of busy waiting, the device can interrupt the CPU when it completes the operation

- In case of an I/O interrupt:
  - The CPU detects which device caused the interrupt
  - In case of input operation, it reads the data from the control registers
  - Initiates next operation with the device

# Interrupts


CPU
Interrupt controller
1. Device is finished
3. CPU acks interrupt
Disk
Keyboard
Clock
Printer
2. Controller issues interrupt
Bus

# Interrupt Request




1. The device request service
2. The interface sets IRQi
3. The Interrupt controller interrupts the CPU
4. The CPU requests interrupt byte (with INTA)
5. The Interrupt controller displays the interrupt byte
6. CPU jumps where the interrupt byte directs
7. The CPU communicates with the interface using PIO

# Interrupt Service Routines

- Routine designed for a device to handle an interrupt from another device
  - It has to be invisible to the interrupted program
  - It services a device that uses programmed I/O
  - If an interrupt is caught, it has to be removed before the application is terminated

- Example
  - ISR for a new interface

# Conditions for Entering ISR

- Further interrupts from other devices are disabled
- The IP (instruction pointer), CS (code segment) and FLAGS are saved on the stack
- All other CPU registers are unmodified
- The interrupted program can continue from the point where it was interrupted using a special CPU instruction ‘IRET’
- ‘IRET’ performs POP of the upper three words of the stack and stores them back to IP, CS, FLAGS

# Storing the Program Context

- Continuing from exactly the same CPU state after the interrupt, is of great importance
- The CPU saves a minimal quantity of the program context (FLAGS, CS and IP)
- It is up to the programmer of the ISR to save any other registers that might be needed and modified in the ISR

# General Structure

1. The interrupt operation saves PC and FLAGS
2. Save the environment
   - Push all registers to be used
3. The I/O action
   - Receives the device status
   - Executes the I/O action
4. Restore the environment
   - Pop the previously saved registers from stack
5. IRET

*If an interrupt is caught (e.g. hotkey) the procedure has to know what's on the stack before jumping to the original ISR*

# Precise and Imprecise Interrupts

Features of precise interrupts:

1. The PC (Program Counter) is saved in a known place
2. All instructions before the one pointed to by the PC have completed
3. No instruction beyond the one pointed to by the PC has been executed
4. The execution state of the instruction pointed to by the PC is known

# Precise and Imprecise Interrupts

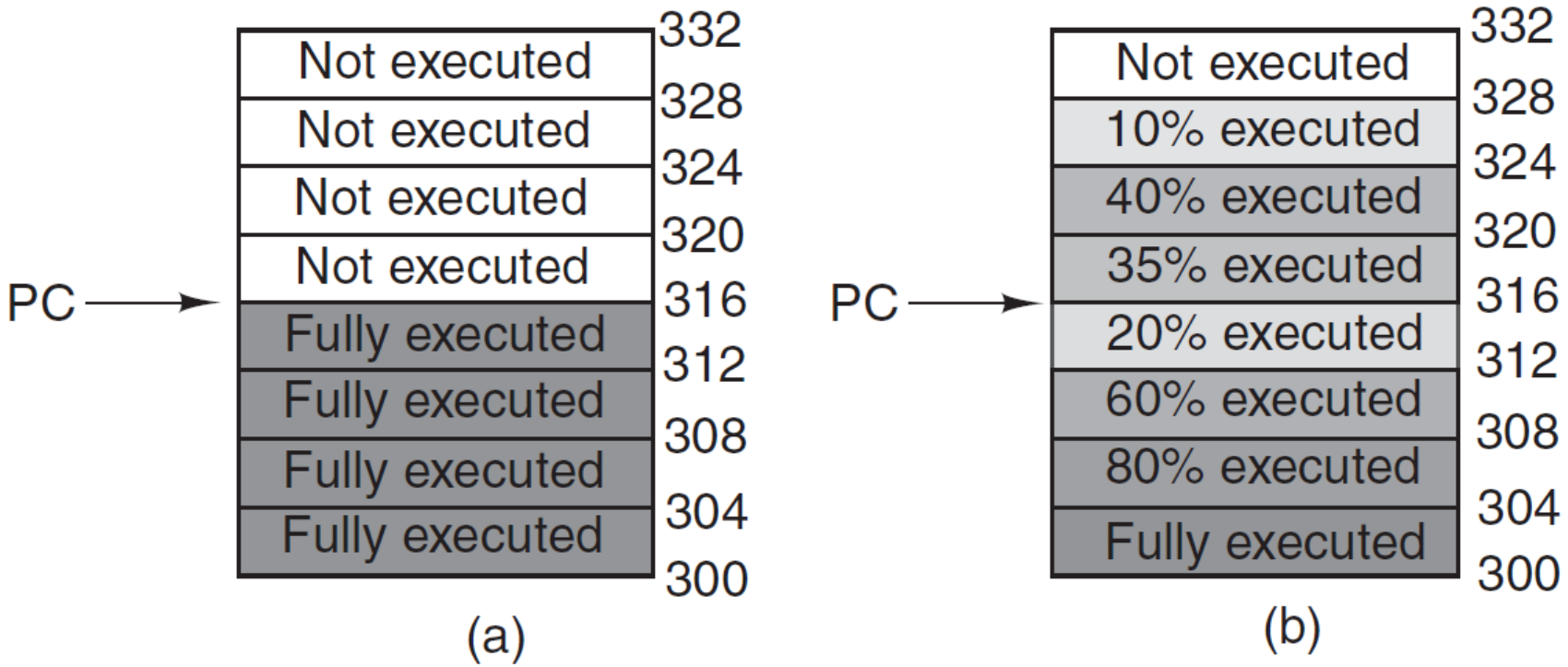



**Figure 5-6.** (a) A precise interrupt. (b) An imprecise interrupt.

# Imprecise Interrupts

- Big amount of state data is saved on the stack
- For compatibility reasons, Pentium works with both precise and imprecise interrupts
  - The size of the chip is increased because of the more complex chip design
- The operating system working with imprecise interrupts is more complex and slower

# Direct Memory Access (DMA)

- DMA is a controller which alleviates the CPU from low-level data transfers
- The DMA controller can directly write data into memory, instead of using the input and output registers of the CPU
- Besides the disk address of the block that should be read, the CPU gives two additional parameters to the DMA controller:
  - The memory address where the block will be written
  - Number of bytes that have to be transferred

# DMA

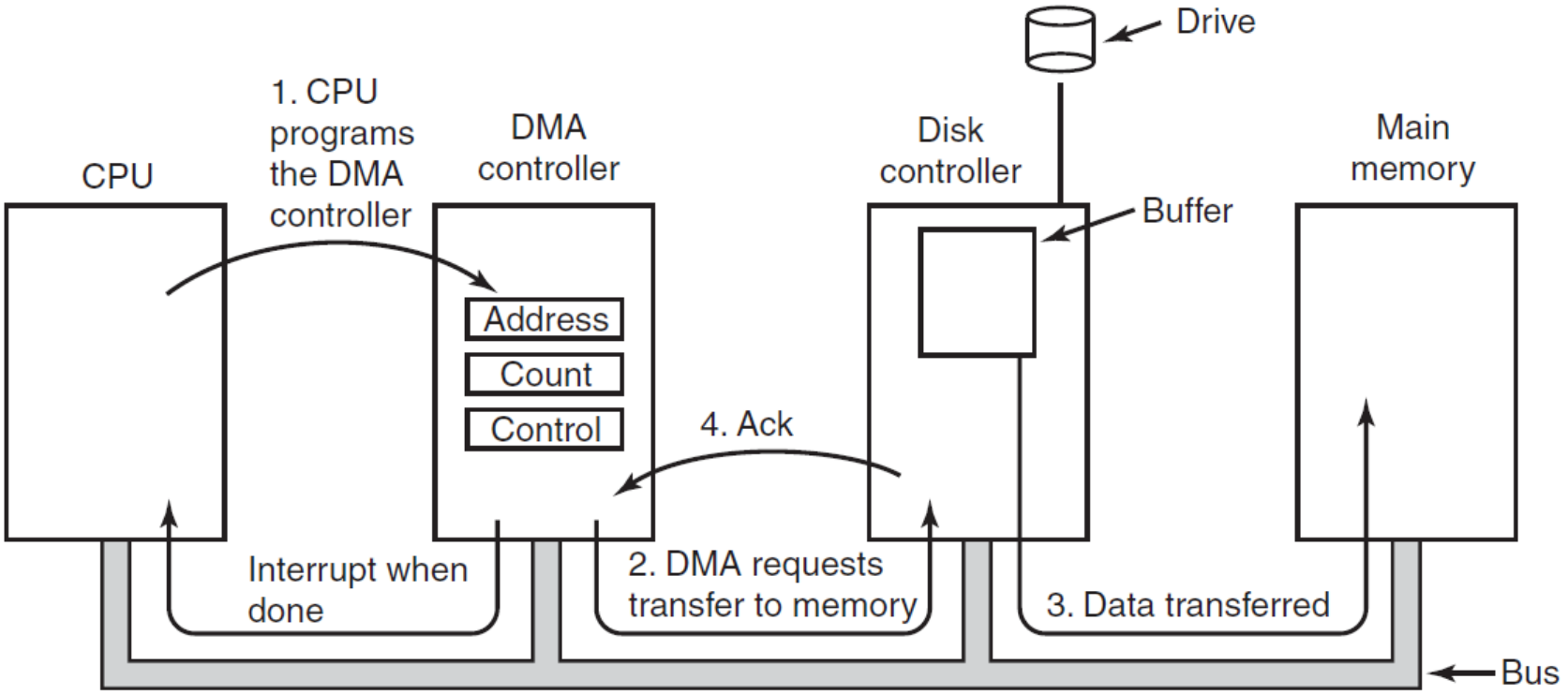



**Figure 5-4.** Operation of a DMA transfer.

# DMA

## Reading from HDD with DMA:

- The controller:
  - Reads the entire block from the device (HDD) into its own buffer and verifies the correctness of the data
  - Copies the first byte or word into the main memory, on an address assigned by the DMA
  - Acknowledges the DMA
  - DMA increments the memory address and decrements the byte count
  - The process is repeated until the byte counter in the DMA reaches 0
  - Causes an interrupt of the CPU

# DMA

- With DMA, the CPU is interrupted only when the transfer is done
- The DMA controller and the CPU compete for the system bus, which partially slows down the CPU, but to a significantly smaller scale compared to the case when the CPU is being continuously interrupted

# DMA Modes

- Cycle Stealing: Used to transfer the data word-at-a-time using the system bus, i.e., the instruction cycle is suspended so that the data can be transferred.

- Burst Mode: The DMA controller instructs the device to acquire the bus, perform the operations (bursts) and, eventually to release the bus.
  - Blocks the CPU and other devices for a substantial period if a long burst is being transferred

# I/O Software: Printing a String

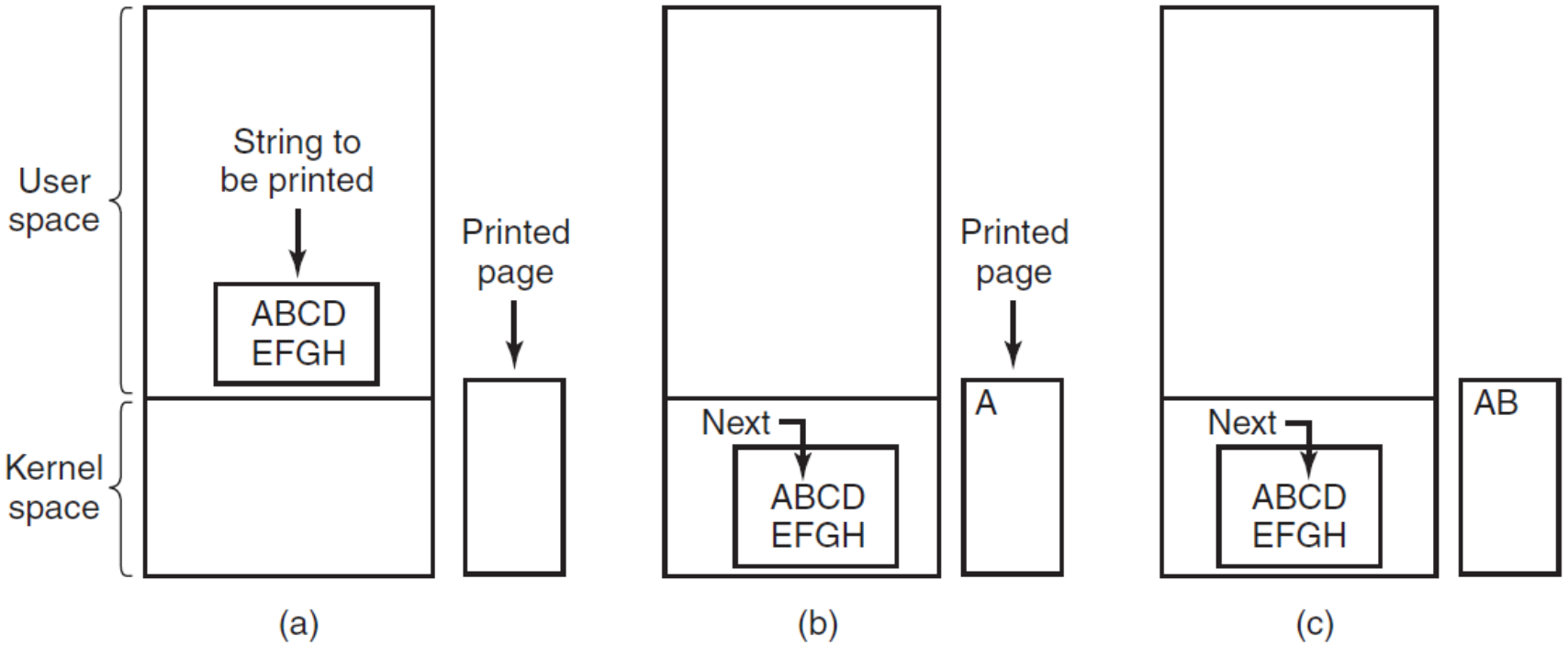



**Figure 5-7.** Steps in printing a string.

# Programmed I/O

```
copy_from_user(buffer, p, count);          /* p is the kernel buffer */
for (i = 0; i < count; i++) {              /* loop on every character */
        while (*printer_status_reg != READY) ;   /* loop until ready */
        *printer_data_register = p[i];           /* output one character */
}
return_to_user( );
```

- Polling / Busy Waiting

# Interrupt-Driven I/O

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

- Writing a string to the printer using interrupt-driven I/O.
  - (a) Code executed at the time the print system call is made.
  - (b) Interrupt service procedure for the printer.

# I/O with DMA

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

- Printing a string using DMA.
  - (a) Code executed when the print system call is made.
  - (b) Interrupt-service procedure.

# Principles of I/O Software

- Device independence
  - Programmers can access any group of I/O devices the same way
    - Example: floppy, hard drive or CD-ROM
    - sort < input > output
- Unified naming
  - Names of devices and files have the same format (string or integer)
  - The name does not depend on the device type
- Error handling
  - It has to be as close as possible to the hardware (where the error occurred)

# Principles of I/O Software

- Synchronous and asynchronous transfer
  - Blocking and interrupt-driven transfer
- Buffering
  - The data that goes to / comes from the devices has to be stored on a temporary location
- Sharable and dedicated devices
  - Hard drives are sharable – multiple users can access them simultaneously
  - The tape is dedicated

# I/O Software Layers

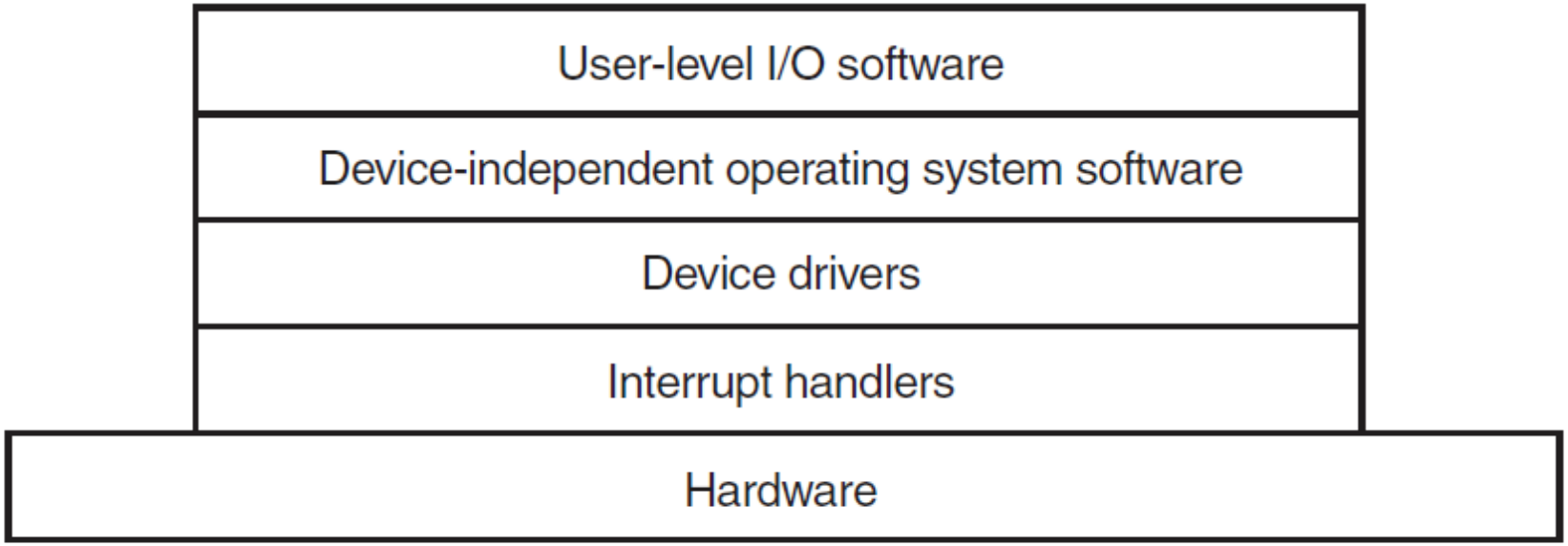



**Figure 5-11.** Layers of the I/O software system.

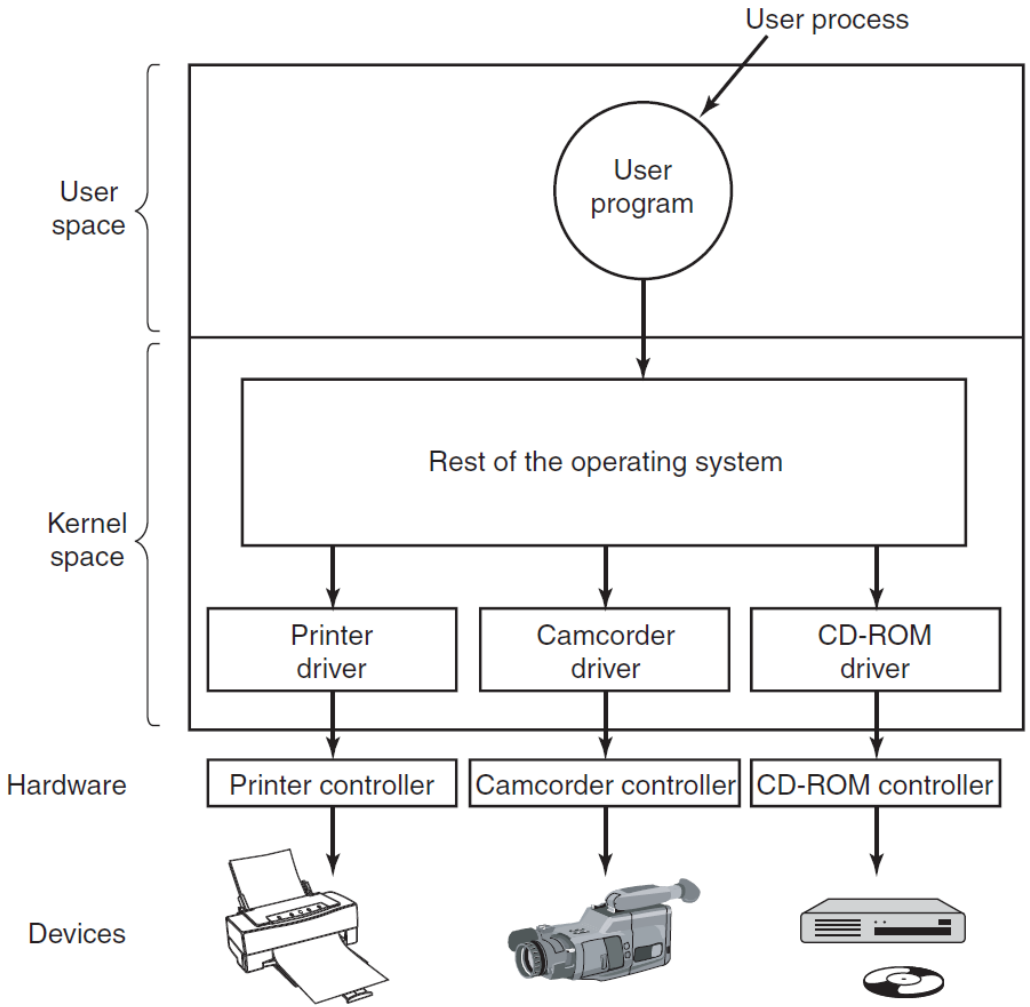
# OS Interrupt Handling

1. Save any registers (including the PSW) that have not already been saved by the interrupt hardware.
2. Set up a context for the interrupt service procedure.
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
5. Copy the registers from where they were saved to the process table.
6. Run the interrupt service procedure.
7. Choose which process to run next.
8. Set up the MMU context for the process to run next.
9. Load the new process' registers, including its PSW.
10. Start running the new process

# Drivers

- Code necessary for managing a device
  - OS specific and distributed by the device manufacturer
- The drivers access the control words for managing the device

# Logical Positioning of Drivers



# Driver Characteristics

- One driver per device
  - Or a group of connected devices (SCSI driver)
- Driver in the kernel space
  - It can be placed in the user space (MINIX 3)
- The operating systems define what has to be supported by the block-oriented and character-oriented drivers

# Driver Functions

- Drivers have to provide a way to:
  - Read data
  - Write data
  - Initialize the device
  - Manage the energy requirements
- They can optionally provide:
  - Error detection
  - Error correction
  - Error report to higher levels

# Driver Structure

- Checking the input parameters
- Conversion of parameters to hardware specific format
  - HDD: head, track, sector
- Check if the device is currently in use
  - If it is, the request will be queued for later processing
- Creating a sequence of commands
- Issuing the commands to the device
- Waiting the completion of the requested action by the device

# Important Problems that Drivers Should Handle

- Initiating a second instance of the driver while the first instance is still executing (reentrant driver)

- Additional problems that occur with hot-pluggable systems (devices that can be plugged and unplugged while the computer is on)

# Device Independent I/O Software

- Uniform interface for device drivers
- Buffering
- Error reporting
- Allocating and releasing of dedicated devices
- Providing a device-independent block size

# Driver Standardization

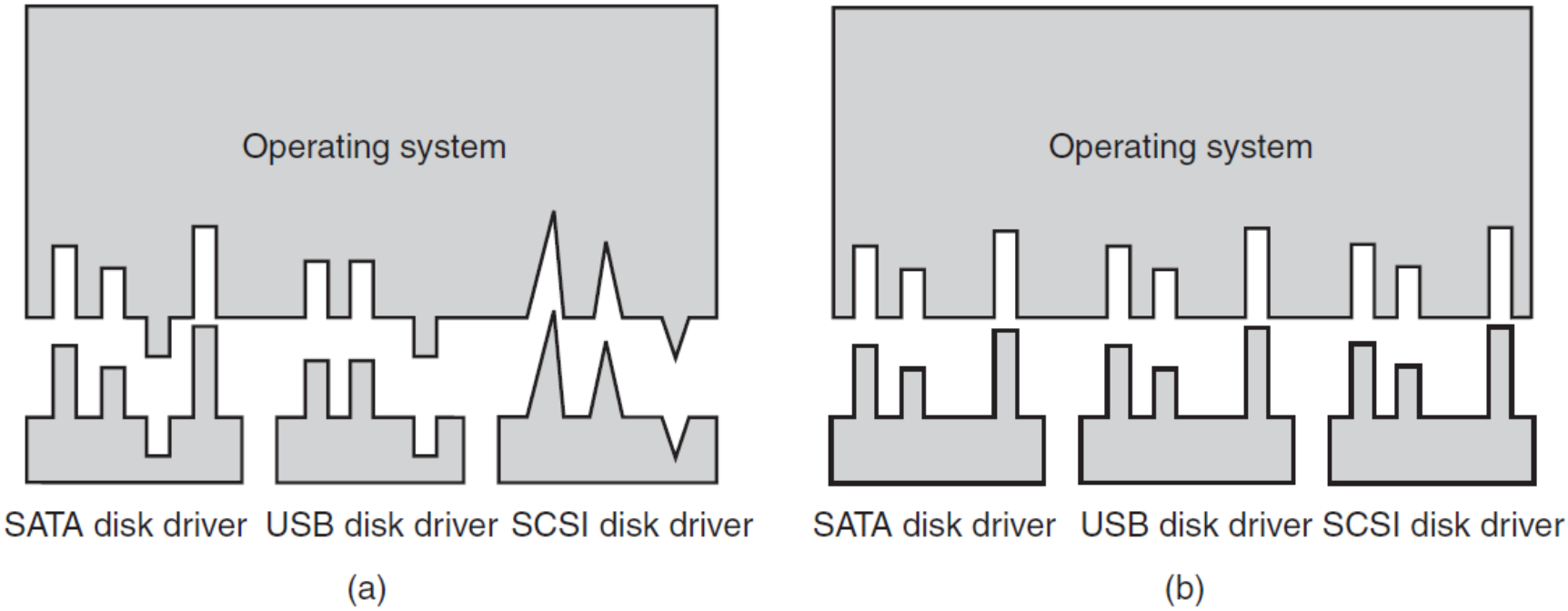



**Figure 5-14.** (a) Without a standard driver interface. (b) With a standard driver interface.

# Driver Standardization

- The OS defines a set of functions for every driver
  - Read, write, turn on, turn off, format (HDD), …
- The driver contains a table of pointers to functions
- When the driver is loaded, the OS saves the address of the function table
- The table is an interface between the OS and the driver
- All drivers belonging to the same class must implement the functions of the interface

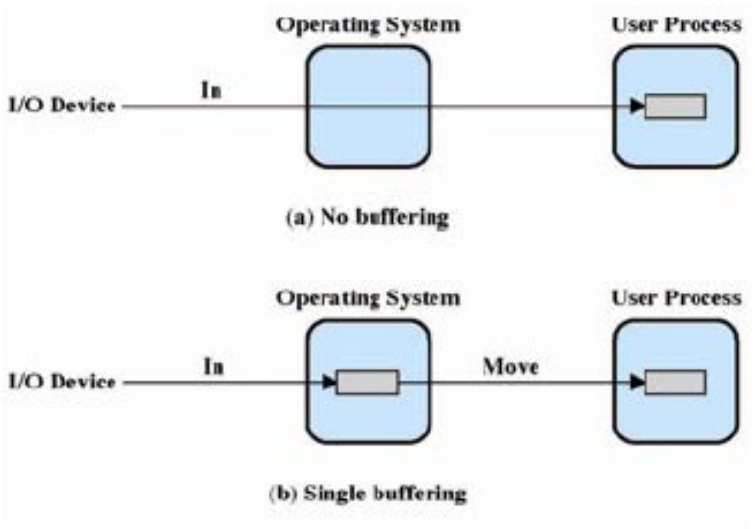
# Mapping

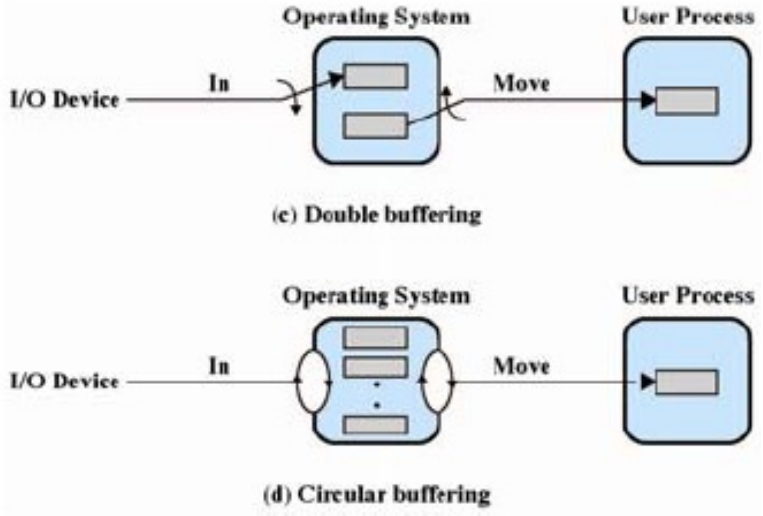
- Mapping of a symbolic name (/dev/disk0) to a corresponding driver (major device number)

```
brw-rw---- 1 root disk 8, 0 May 3 10:38 /dev/sda
```

- The same rules for file protection apply

# I/O Schemes (Input)



# Single Buffering

- The OS assigns a buffer in the main memory for a single I/O request
- Input transfers are stored in the buffer
- A block is transferred in the user space when it is necessary
- The user process can process a single data block while other block is being transferred to the buffer
- **Swapping of a user process is enabled because the input is stored in the kernel space, not in the user space**

# Double and Circular Buffering

- Double buffering
  - Uses 2 system buffers instead of one
  - The process can transfer data from/to one of the buffers, while the OS empties/fills the other buffer
- Circular buffering
  - More than 2 buffers are used
  - Each buffer is a unit of the circular buffer
  - It is used when the I/O operation has to keep up with the processes in execution

# Buffering Problem (Performance)

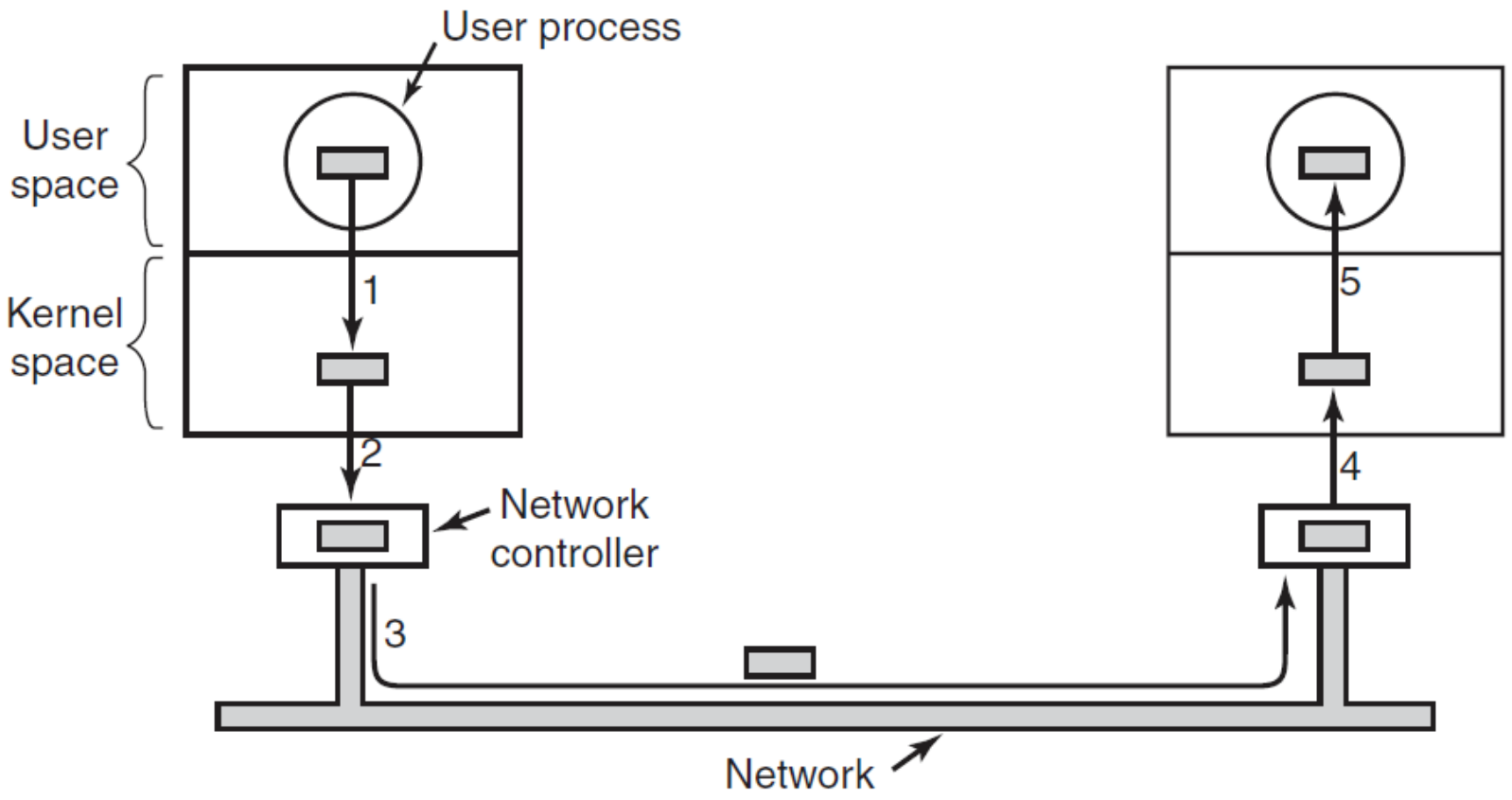



**Figure 5-16.** Networking may involve many copies of a packet.

# Other Functions of Device Independent I/O Software

- Error handling and reporting
  - Program errors: writing data to keyboard, reading from printer, non-existing device, etc.
  - I/O errors: writing on a bad block on the hard drive, reading from a switched-off device, etc.
- Allocation and releasing of non-shareable devices (CD-ROM writer, a process)
  - Direct opening of special device files
  - Blocking mode until the device is released
- Mechanism for device-independent block size

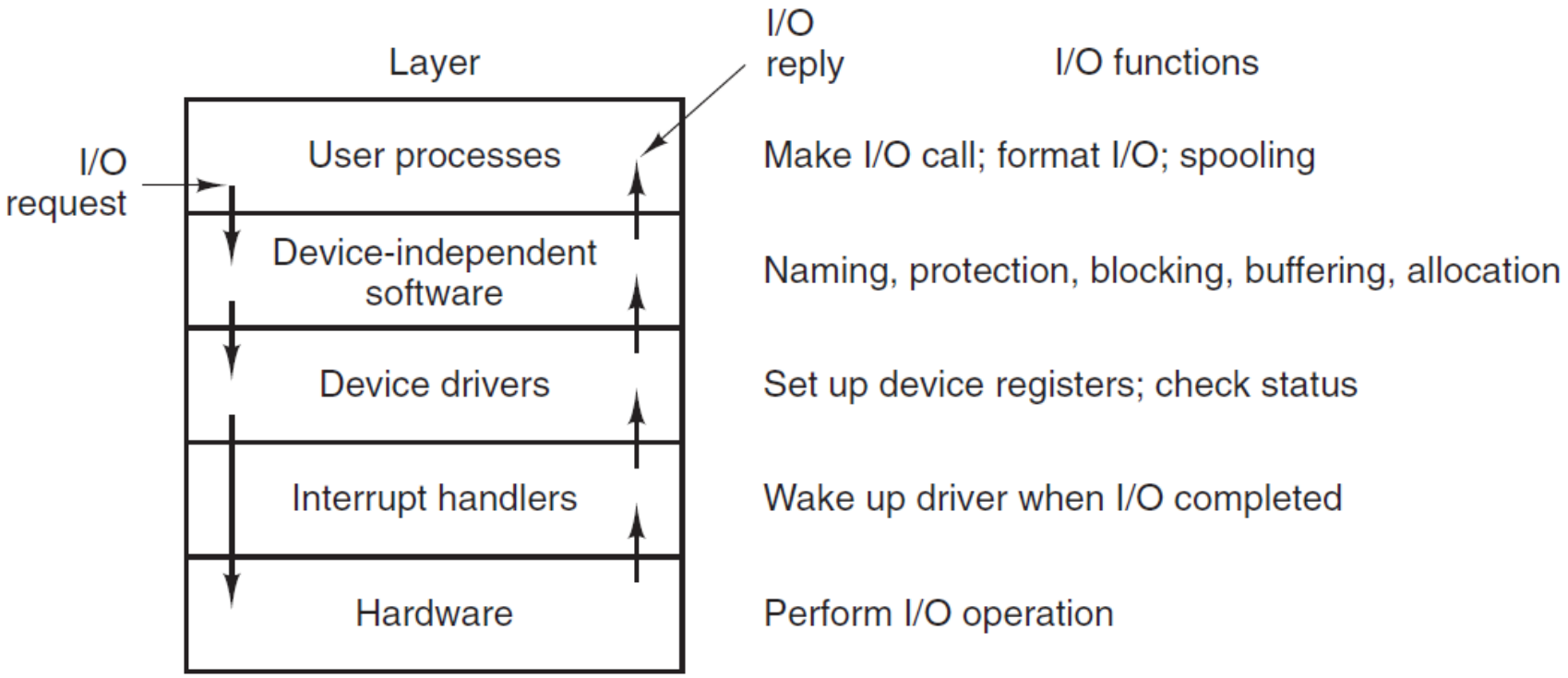
# User Space I/O Software

- Libraries that perform I/O operations

  count = write (fd, buffer, nbytes);

- Spooling process
  - It deals with dedicated devices
  - Typically used for printers
  - Daemon process and spooling directory

# Communication Between I/O System Layers



# Questions?