现 代 操 作 系 统

Modern Operating Systems

宋虹 songhong@csu. edu. cn

中南大学计算机学院网络空间安全系

Chapter 5 Input/Output

5.1.1 Overview

- OS controls I/O devices =>
  - Issue commands,
  - handles interrupts,
  - handles errors
- Provide easy to use interface to devices
  - Hopefully device independent
- First look at hardware, then software
  - Emphasize software
  - Software structured in layers
  - Look at disks

I/O Devices

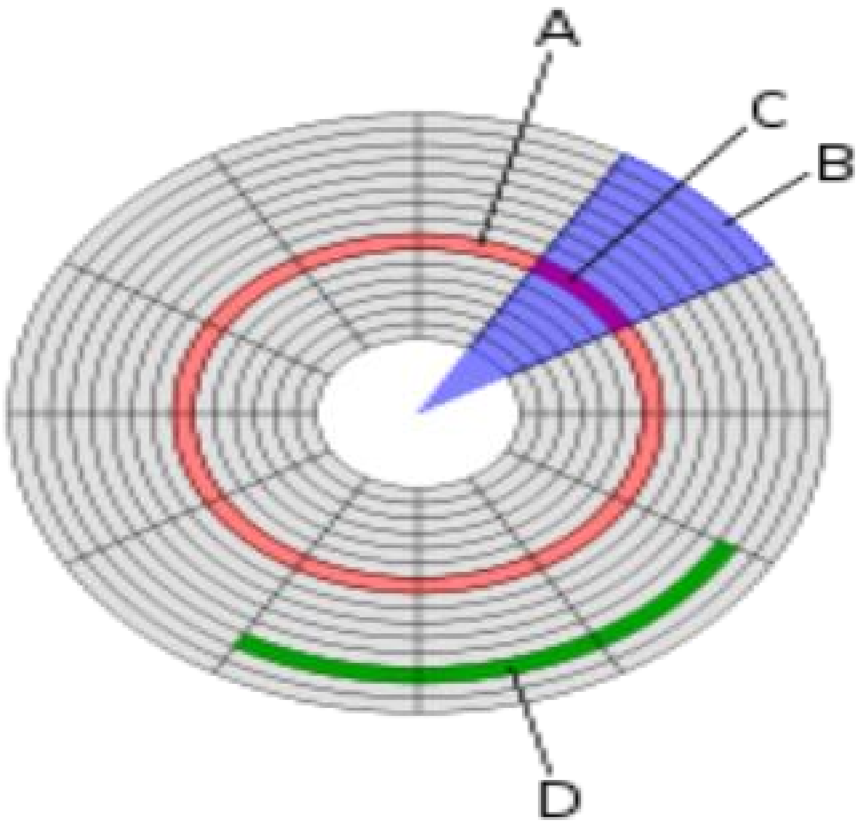
- Some typical device, network, and bus data rates.

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	3.5 MB/sec
802.11g Wireless	6.75 MB/sec
52x CD-ROM	7.8 MB/sec
Fast Ethernet	12.5 MB/sec
Compact flash card	40 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
SATA disk drive	300 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec

5.1.1 Overview

- Two types of I/O devices- block, character
  - Block- can read blocks independently of one another
    - Hard disks, CD-ROMs, USB sticks
    - 512-32,768 bytes
  - Character-accepts characters without regard to block structure
    - Printers, mice, network interfaces
- Not everything fits, e.g. clocks don’t fit
- Division allows for OS to deal with devices in device independent manner
  - File system deals with blocks

Disk geometry



A is track, B is sector, C is geometrical sector, D is cluster

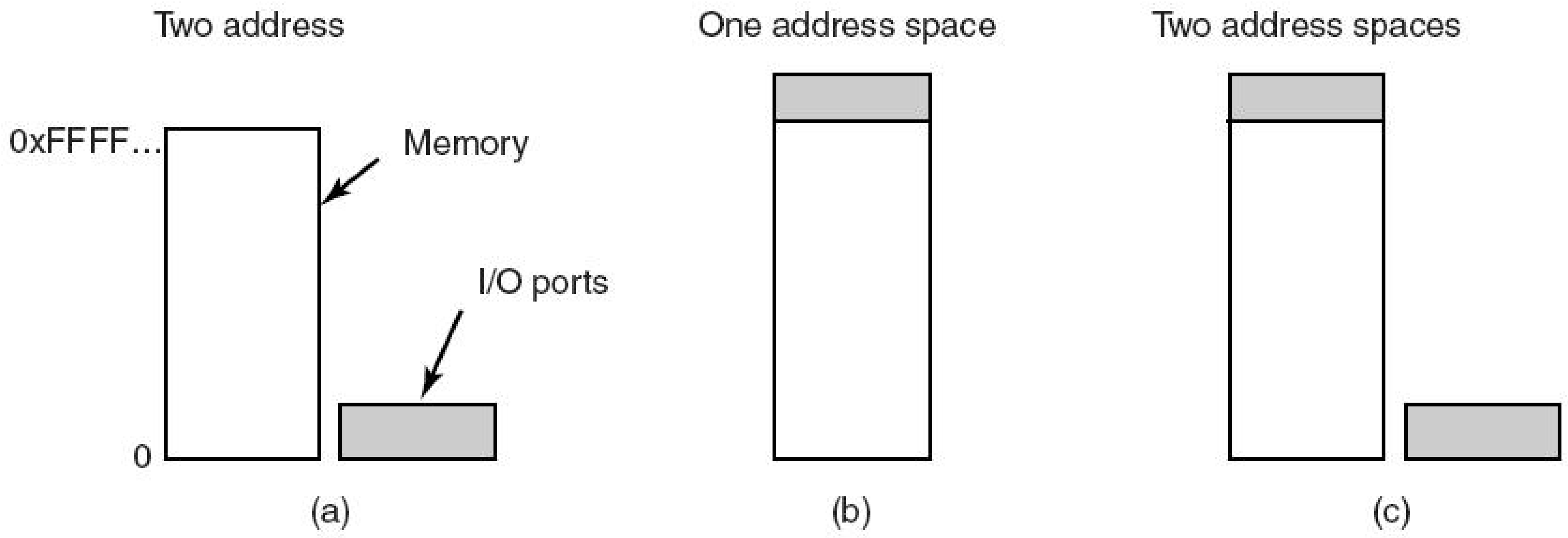
5.1.2 Device Controllers

- I/O unit has 2 components-mechanical, electronic (controller)
- Controller is a chip with a connector which plugs into cables to device
- Disk
  - Disk might have 10,000 sectors of 512 bytes per track
  - Serial bit stream comes off drive
  - Has preamble, 4096 bits/sector, error correcting code
  - Preamble has sector number, cylinder number, sector size….
- Controller assembles block from bit stream, does error correction, puts into buffer in controller
- Blocks are what are sent from a disk

5.1.3 Memory Mapped I/O

- Controller has registers which OS can write and read
- Write-gives command to device
- Read-learn device status……
- Devices have data buffer which OS can read/write (e.g. video RAM, used to display pixels on a screen)
- How does CPU communicate with registers and buffers?

5.1.3 Memory-Mapped I/O



(a) Separate I/O ports and memory space.

(b) Memory-mapped I/O.

(c) Memory mapped data buffers and separate ports (Pentium)

How CPU addresses registers and buffers

- First scheme
  - Puts read on control line
  - Put address on address line
  - Puts I/O space or memory space on signal line to differentiate
  - Read from memory or I/O space
- Memory mapped approach
  - Put address on address line and let memory and I/O devices compare address with the ranges they serve

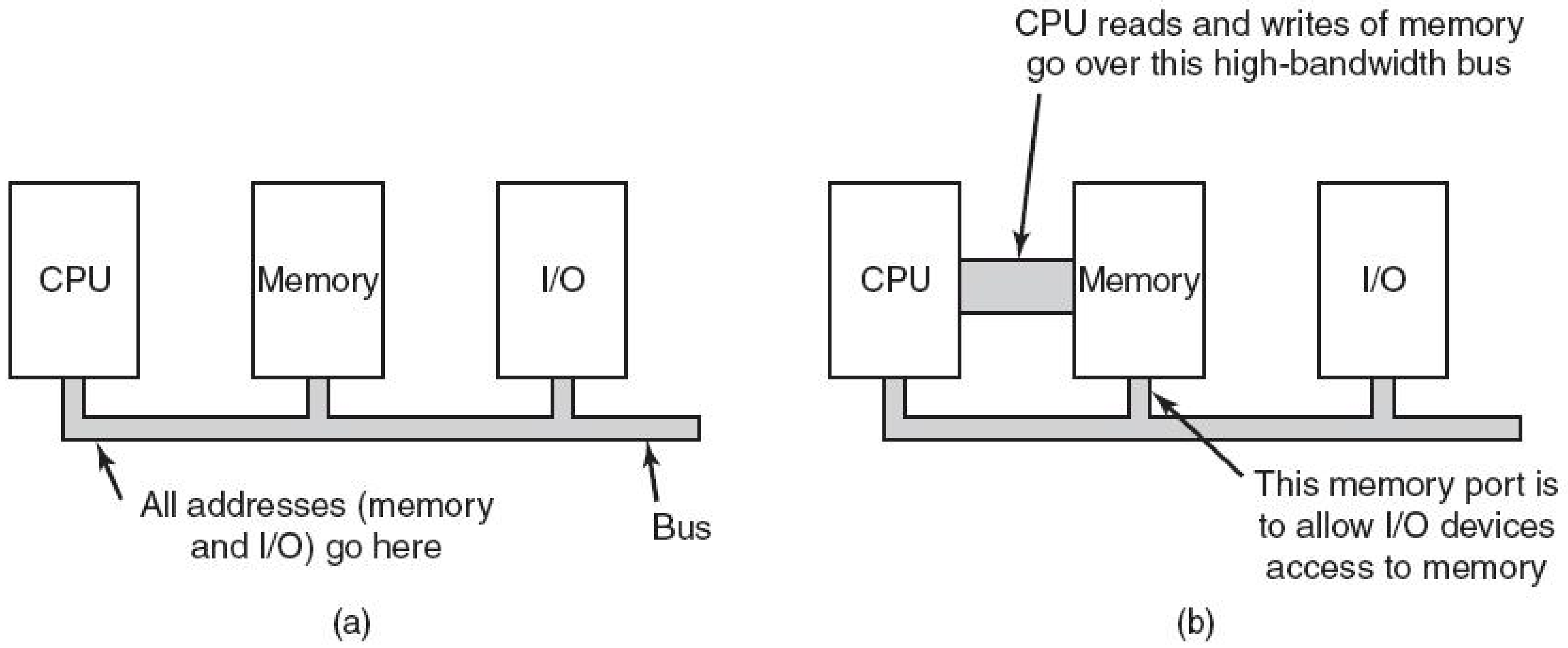
Memory mapped advantages

- Don’t need special instructions to read/write control registers=> can write a device driver in C
- Don’t need special protection to keep users from doing I/O directly. Just don’t put I/O memory in any user space
- An instruction can reference control registers and memory

Memory mapped disadvantage

- Can cache memory words, which means that old memory value could remain in cache
  - => have to be able to disable caching when it is worthwhile
- I/O devices and memory have to respond to memory references
- Works with single bus because both memory and I/O look at address on bus and decide who it is for
- harder with multiple buses because I/O devices can't see their addresses go by any more

Memory-Mapped I/O


CPU reads and writes of memory go over this high-bandwidth bus
CPU
Memory
I/O
All addresses (memory and I/O) go here
Bus
(a)
CPU
Memory
I/O
This memory port is to allow I/O devices access to memory
(b)

(a) A single-bus architecture. (b) A dual-bus memory architecture.

Solutions for multiple buses

- Solution 1 is for CPU to try memory first. If it does not get a response then it tries I/O device
- Solution 2: Other fixes-snooping device, filter addresses
- Solution 3: Main point-have to complicate hardware to make this work

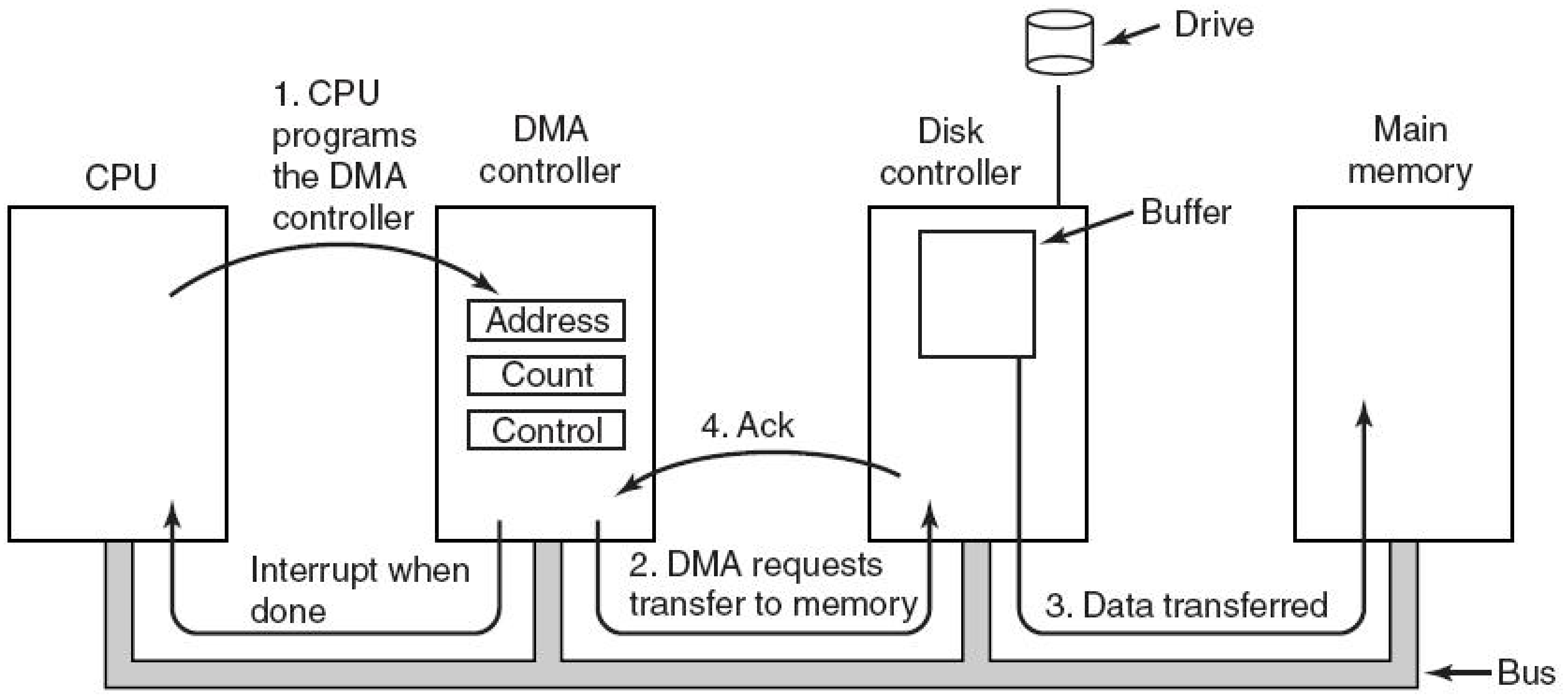
5.1.4 DMA

- CPU COULD request data one byte at a time from I/O controller
- Big waste of time, use DMA
- DMA controller on mother-board; normally one controller for multiple devices
- CPU reads/writes to registers in controller
  - Memory address register
  - Byte count register
  - Control registers-I/O port, direction of transfer, transfer units (byte/word), number of bytes to transfer in a burst

(1) When DMA is not used

- Controller reads a block into its memory
- Computes checksum
- Interrupts OS
- Sends byte at a time to memory

(2) How does DMA work?



Operation of a DMA transfer.

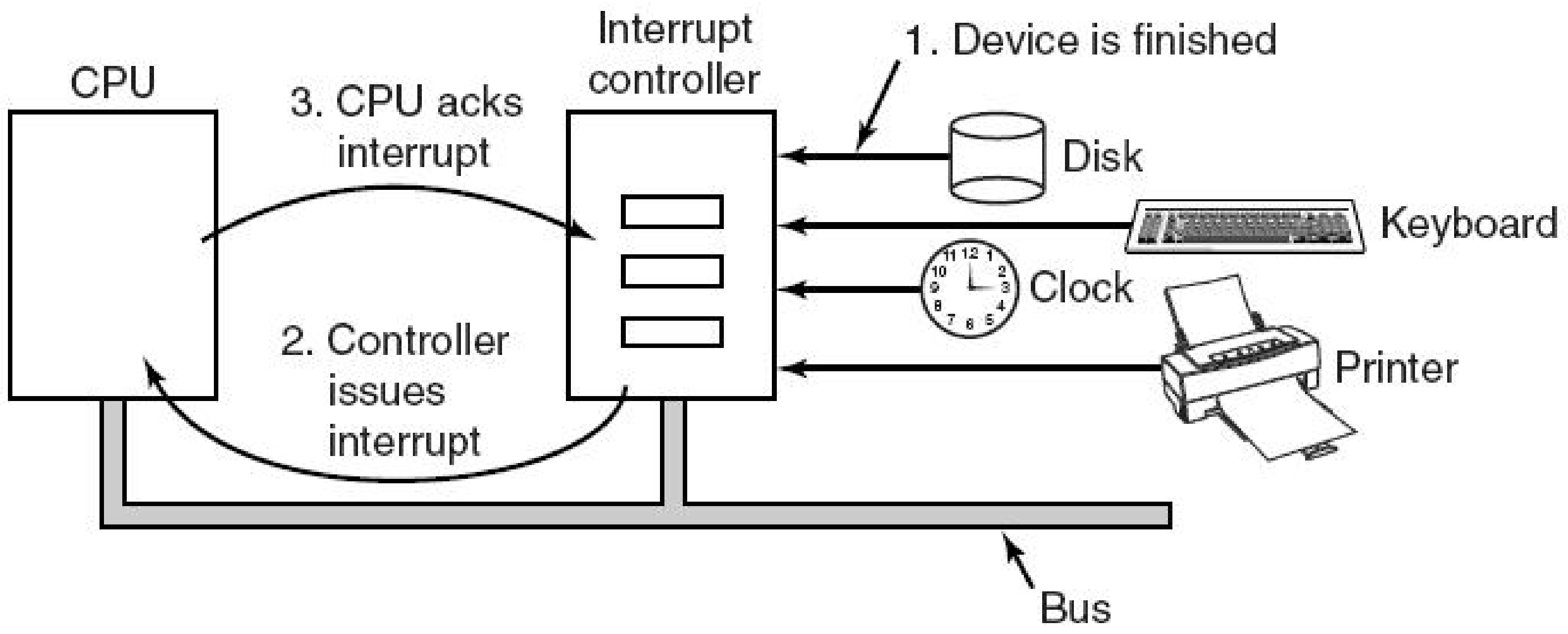
(3) DMA controller modes

- Cycle stealing mode-transfer goes on word at a time, competing with CPU for bus cycles. Idea is that CPU loses the occasional cycle to the DMA controller
- Burst mode-DMA controller grabs bus and sends a block
- Fly by mode-DMA controller tells device controller to send word to it instead of memory. Can be used to transfer data between devices.

Questions

- Why buffer data in controllers?
  - Can do check-sum
  - Bus may be busy-need to store data some place
- Is DMA really worth it? Not if
  - CPU is much faster then DMA controller and can do the job faster
  - don’t have too much data to transfer

5.1.5 PC interrupt structure



The connections between the devices and the interrupt controller use interrupt lines on the bus rather than dedicated wires.

(1) Interrupt processing

- Controller puts number on address line telling CPU which device wants attention and interrupts CPU
- Table (interrupt vector) points to interrupt service routine
- Number on address line acts as index into interrupt vector
- Interrupt vector contains PC which points to start of service routine

(1) Interrupt processing

- Interrupt service routine acks interrupt
- Saves information about interrupted program
- Where to save information
    - User process stack, kernel stack are both possibilities
    - Both have problems

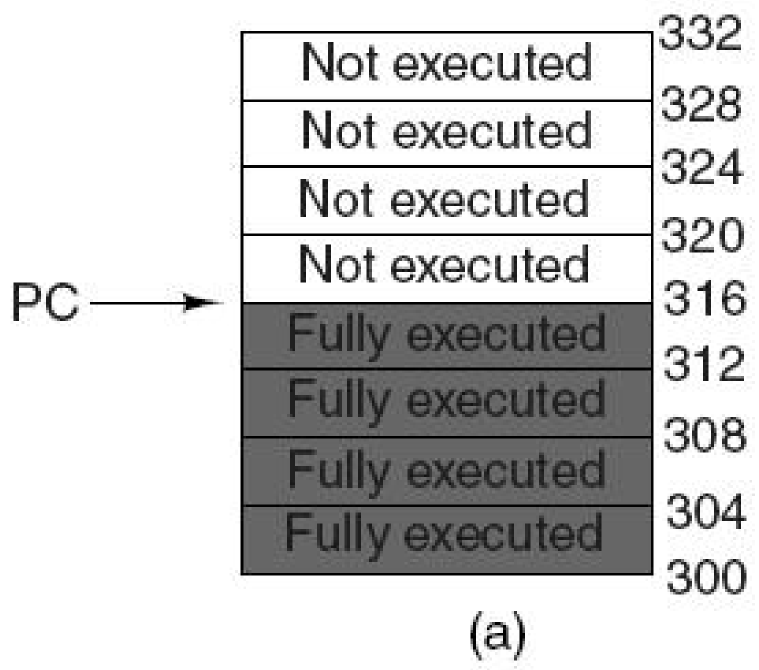
Can we save the PC and PSW?

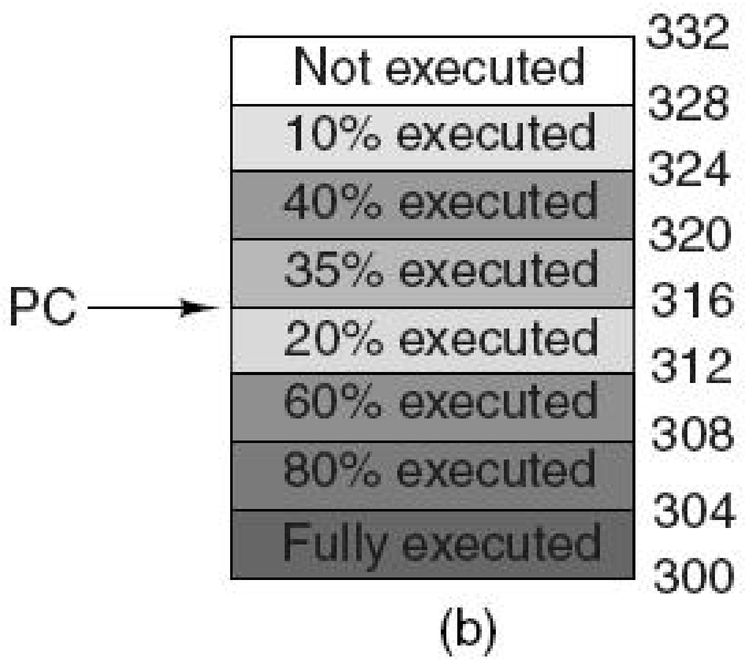
- Sure, if we don’t use pipelined or superscaler CPU’s. But we do use them.
- Can’t assume that all instructions up to and including given instruction have been executed
  - Pipeline-bunch of instructions are partially completed
  - Superscaler-instructions are decomposed and can execute out of order

(2) The precise (ideal) interrupt

- PC (Program Counter) is saved in a known place.
- All instructions before the one pointed to by the PC have fully executed.
- No instruction beyond the one pointed to by the PC has been executed.
- Execution state of the instruction pointed to by the PC is known.

Precise and Imprecise Interrupts





(a) A precise interrupt.

(b) An imprecise interrupt.

(2) How to process an imprecise interrupt

- With great difficulty
- Either Need complicated hardware logic to re-start after interrupt (Pentium)
- Or have complicated processing in the OS

5.2 I/O Software

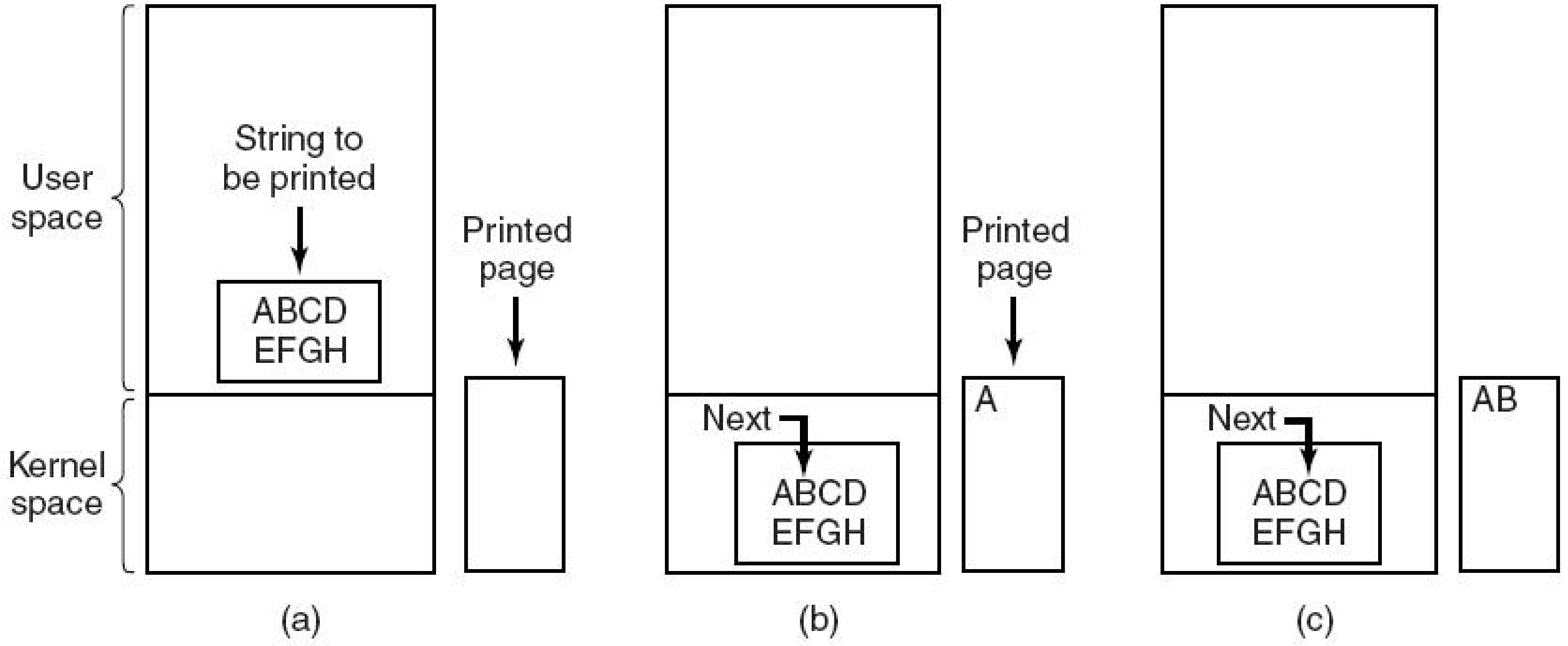
5.2.1 Goals

- Device independence-don’t have to specify the device when accessing the device
- Uniform naming-name should not depend on device type
- Error handling-do it as close to the device as possible (e.g. controller should be the first to fix error, followed by the driver)
- OS needs to make I/O operations blocking (e.g. program blocks until data arrives on a read) because it is easy to write blocking ops

5.2.1 Goals

- Buffering- e.g. when a packet arrives
- Shared devices (disks) and un-shared devices (tapes) must be handled

5.2.2 Programmed I/O



Steps in printing a string.

5.2.2 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();
```

Send data one character at a time

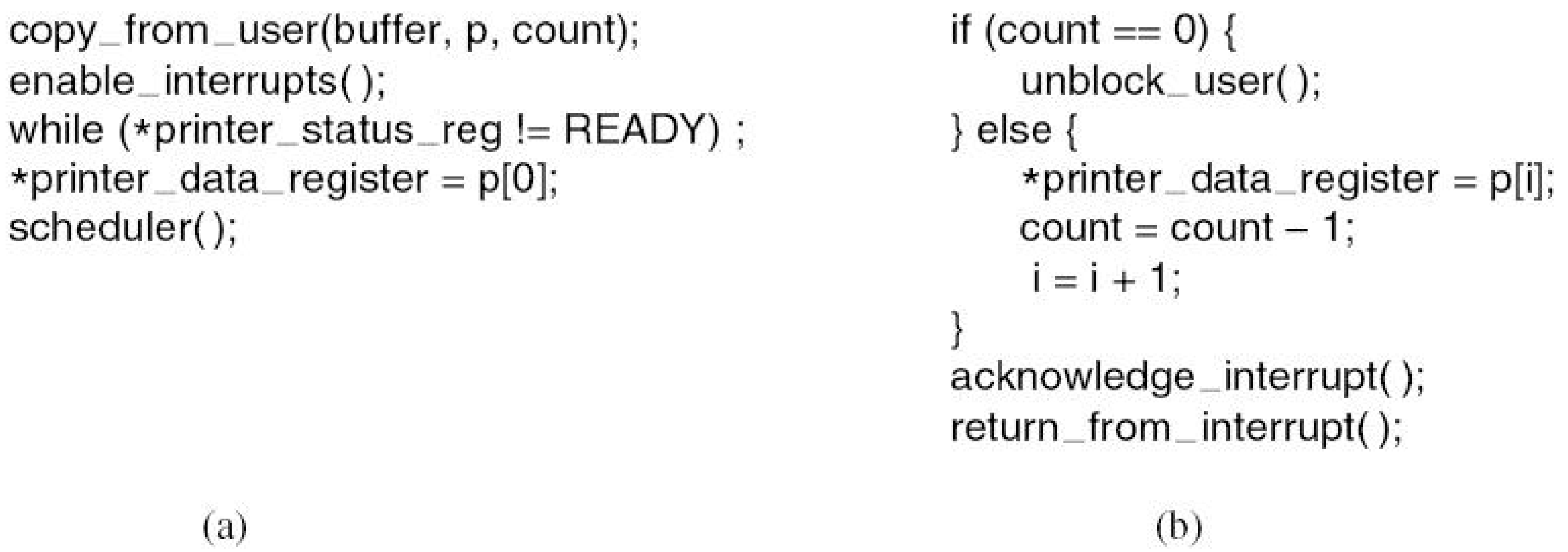
Good vs Bad

- The good: simple idea, OK if the CPU is not bothered too much
- The bad: CPU is bothered too much

5.2.3 Interrupt Driven I/O

- Idea: block process which requests I/O, schedule another process
- Return to calling process when I/O is done
- Printer generates interrupt when a character is printed
- Keeps printing until the end of the string
- Re-instantiate calling process

5.2.3 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)

(a) Code executed at the time the print system call is made.

(b) Interrupt service procedure for the printer.

5.2.4 DMA

- Use DMA controller to send characters to printer instead of using the CPU
- CPU is only interrupted when the buffer is printed instead of when each character is printed
- DMA is worth it if (1) DMA controller can drive the device as fast as the CPU could drive it (2) there is enough data to make it worthwhile

I/O Using DMA

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

(a)

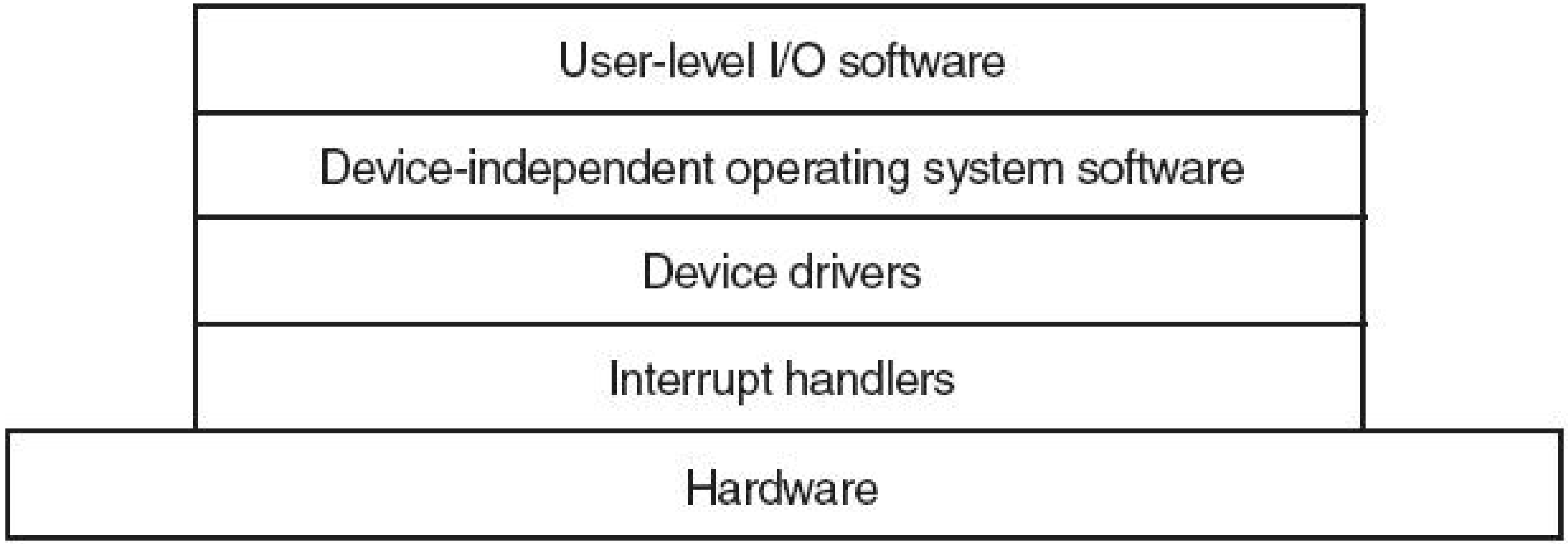
```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

(a) Code executed when the print system call is made.

(b) Interrupt service procedure.

5.3 I/O Software Layers



Layers of the I/O software system.

5.3.1 Interrupt Handlers

- The idea: driver starting the I/O blocks until interrupt happens when I/O finishes
- Handler processes interrupt
- Wakes up driver when processing is finished
- Drivers are kernel processes with their very own
  - Stacks
  - PCs
  - states

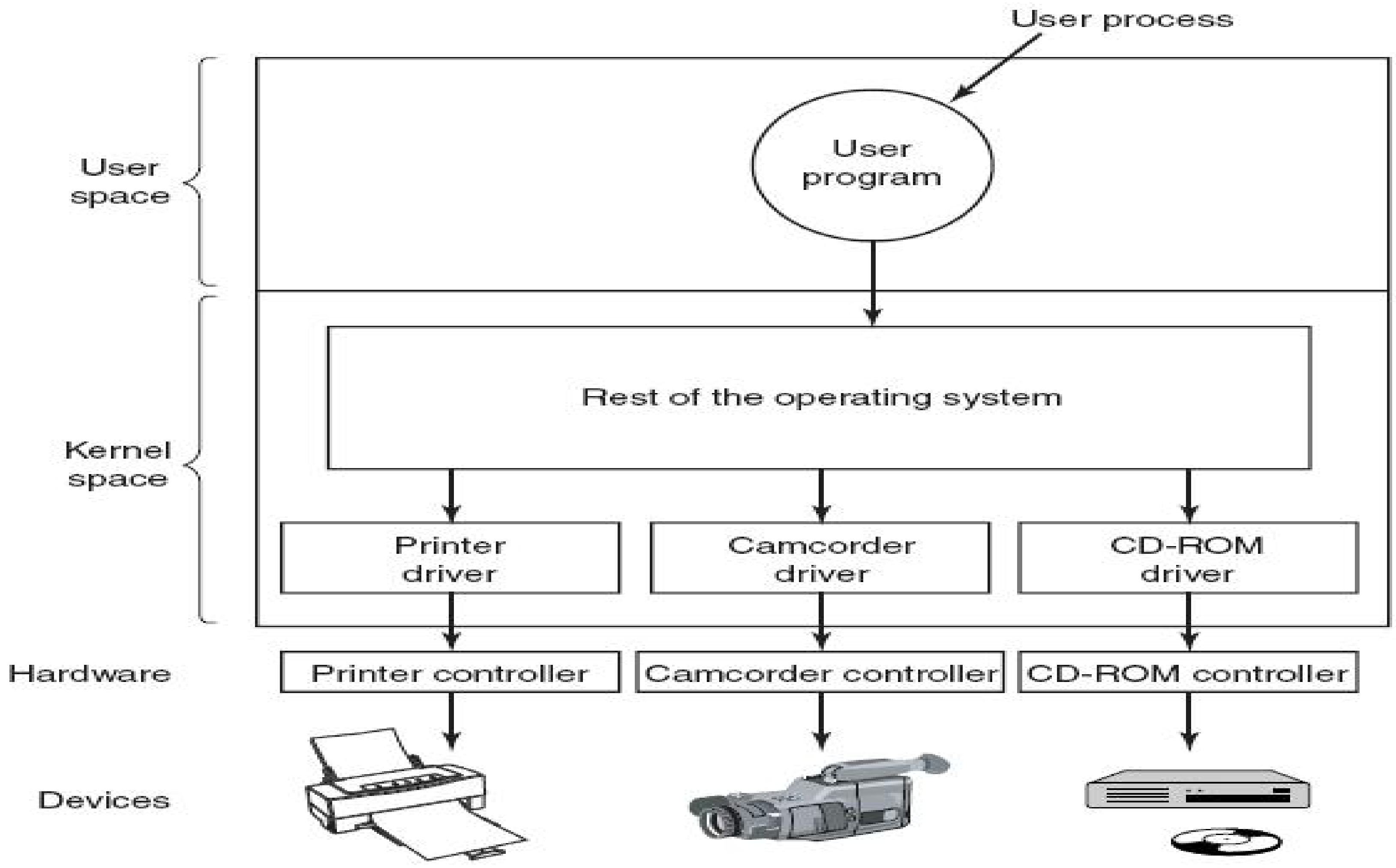
Interrupt processing details

1. Save registers not already saved by 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, re-enable interrupts.
5. Copy the registers from where they were saved to the process table.

Interrupt processing details

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.

5.3.2 Device Drivers



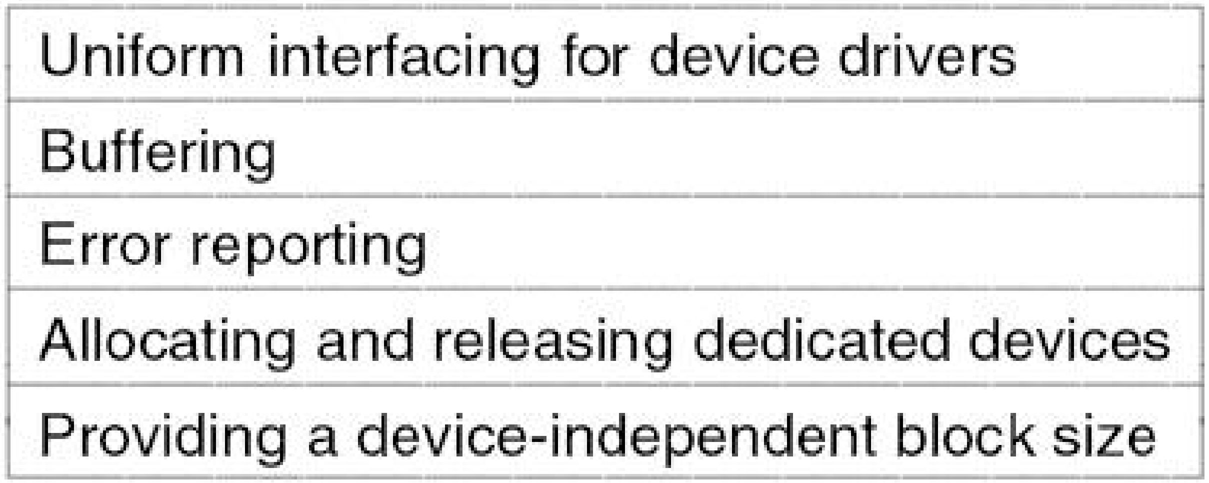
5.3.2 Device Drivers-Act 1

- Driver contains code specific to the device
- Supplied by manufacturer
- Installed in the kernel
- User space might be better place
- Why? Bad driver can mess up kernel
- Need interface to OS
  - block and character interfaces
  - procedures which OS can call to invoke driver (e.g. read a block)

5.3.2 Device drivers Act 2

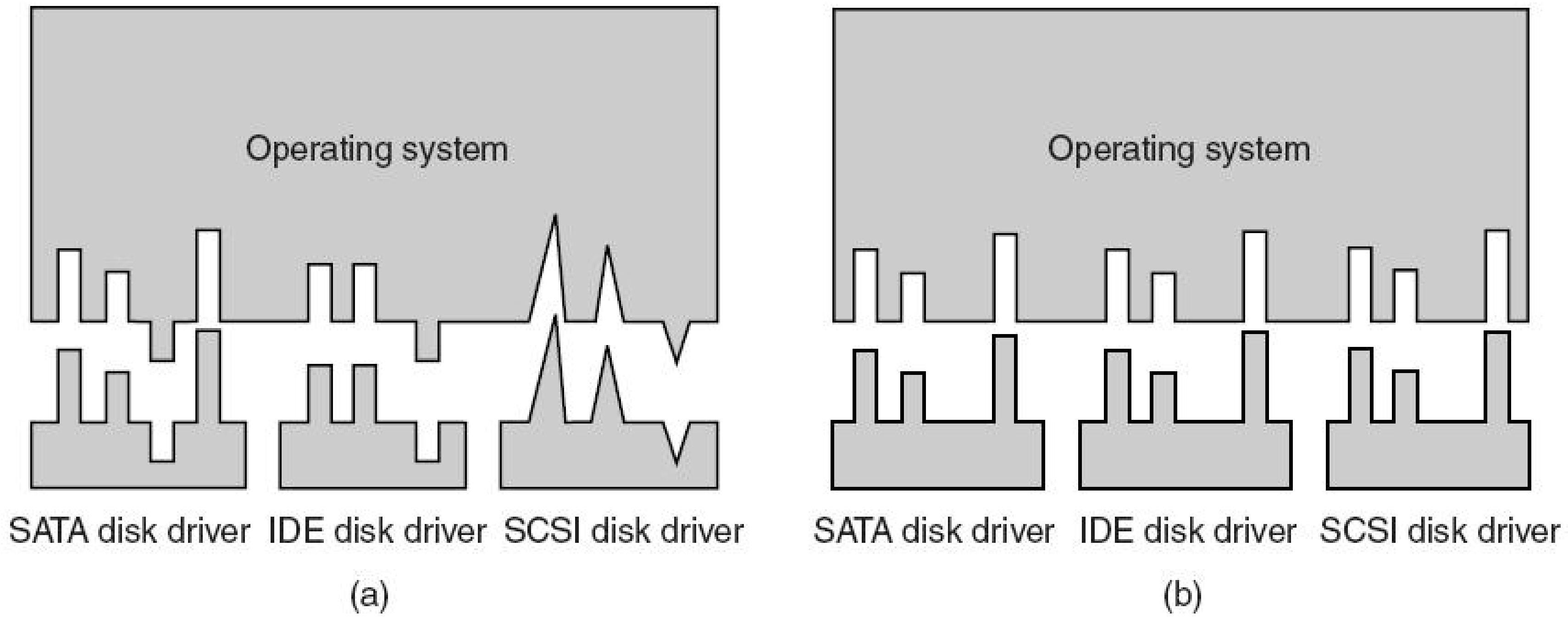
- Checks input parameters for validity
- Abstract to concrete translation (block number to cylinder, head, track, sector)
- Check device status. Might have to start it.
- Puts commands in device controller’s registers
- Driver blocks itself until interrupt arrives
- Might return data to caller
- Does return status information
- The end
- Drivers should be re-entrant
- OS adds devices when system (and therefore driver) is running

5.3.3 Device-Independent I/O Software



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

Why the OS needs a standard interface



- Driver functions differ for different drivers
- Kernel functions which each driver needs are different for different drivers
- Too much work to have new interface for each new device type

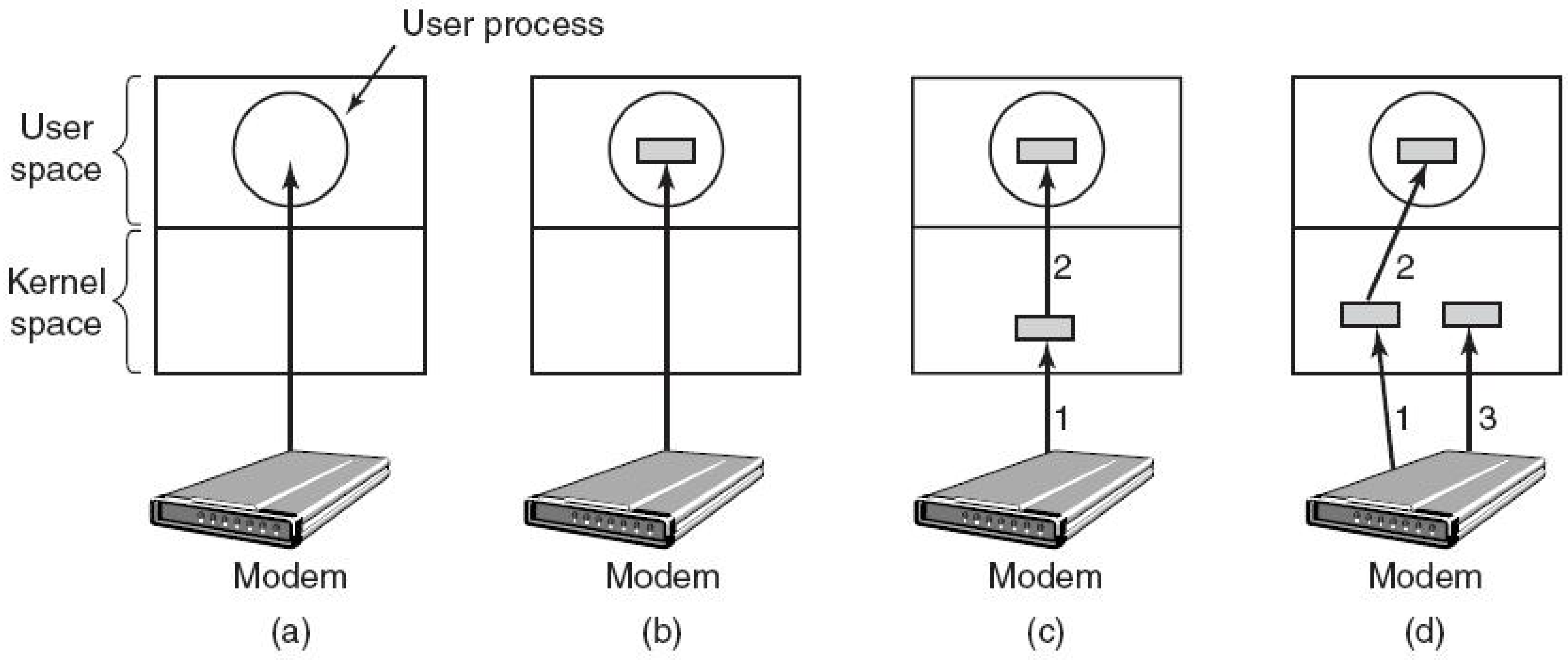
1. Interface:Driver functions

- OS defines functions for each class of devices which it MUST supply, e.g. read, write, turn on, turn off……..
- Driver has a table of pointers to functions
- OS just needs table address to call the functions
- OS maps symbolic device names onto the right driver

1. Interface:Names and protection

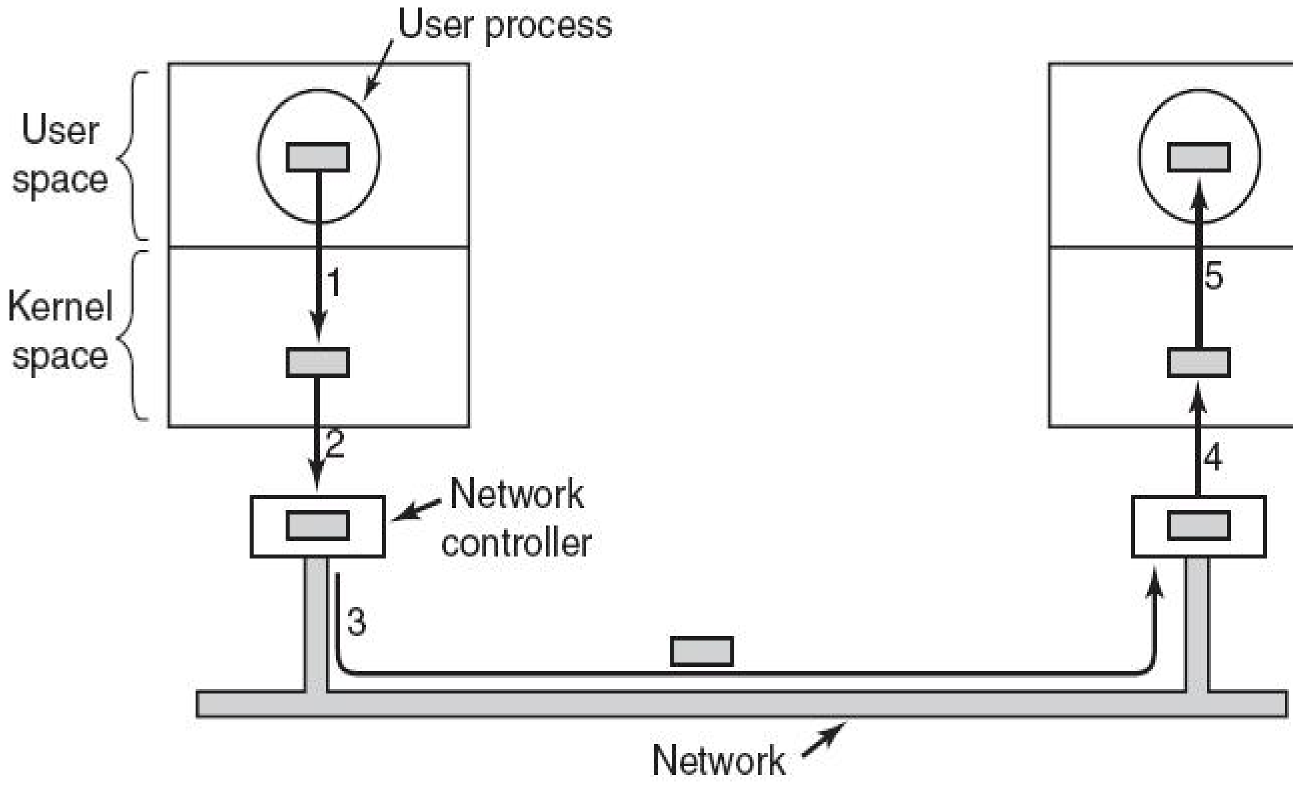
- OS maps symbolic device names onto the right driver
- Unix: /dev/disk0 maps to an i-node which contains the major and minor device numbers for disk0
  - Major device number locates driver, minor device number passes parameters (e.g. disk)
- Protection: In Unix and Windows devices appear as named objects => can use file protection system

2. Buffering



(a) Unbuffered input. (b) Buffering in user space.
(c) Buffering in the kernel followed by copying to user space.
(d) Double buffering in the kernel.

2. Buffering



Networking may involve many copies of a packet.

More Functions of Independent Software

- Error reporting-programming errors (the user asks for the wrong thing), hardware problems (bad disk) are reported if they can’t be solved by the driver
- Allocates and releases devices which can only be used by one user at a time (CD-ROM players)
  - Queues requests or
  - Lets open simply fail
- Device independent block size-OS does not have to know the details of the devices
  - E.g. combine sectors into blocks

5.3.4 User Space I/O Software

- Library routines are involved with I/O- printf,scanf,write for example. These routines makes system calls
- Spooling systems-keep track of device requests made by users.
- Think printing.
  - User generates file, puts it in a spooling directory.
  - Daemon process monitors the directory, printing the user file
- File transfers also use a spooling directory

Example Flow through layers

- User wants to read a block, asks OS
- Device independent software looks for block in cache
- If not there, invokes device driver to request block from disk
- Transfer finishes, interrupt is generated
- User process is awakened and goes back to work

5.4 Disks
5.4.1 some Disk hardware

- Magnetic (hard)
  - Reads and writes are equally fast=> good for storing file systems
  - Disk arrays are used for reliable storage (RAID)
- Optical disks (CD-ROM, CD-Recordables, DVD) used for program distribution

Floppy vs hard disk (20 years apart)

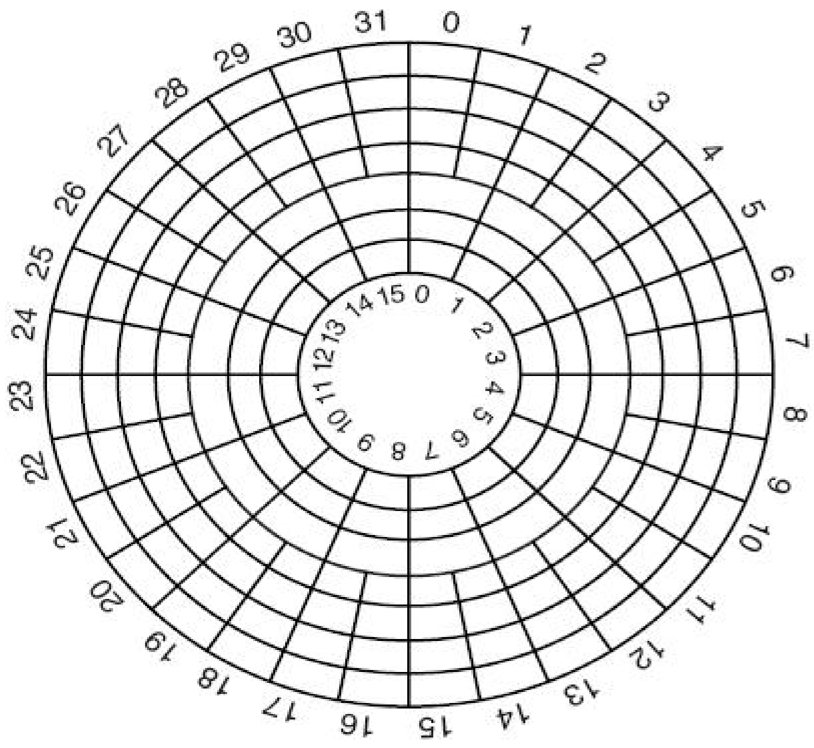
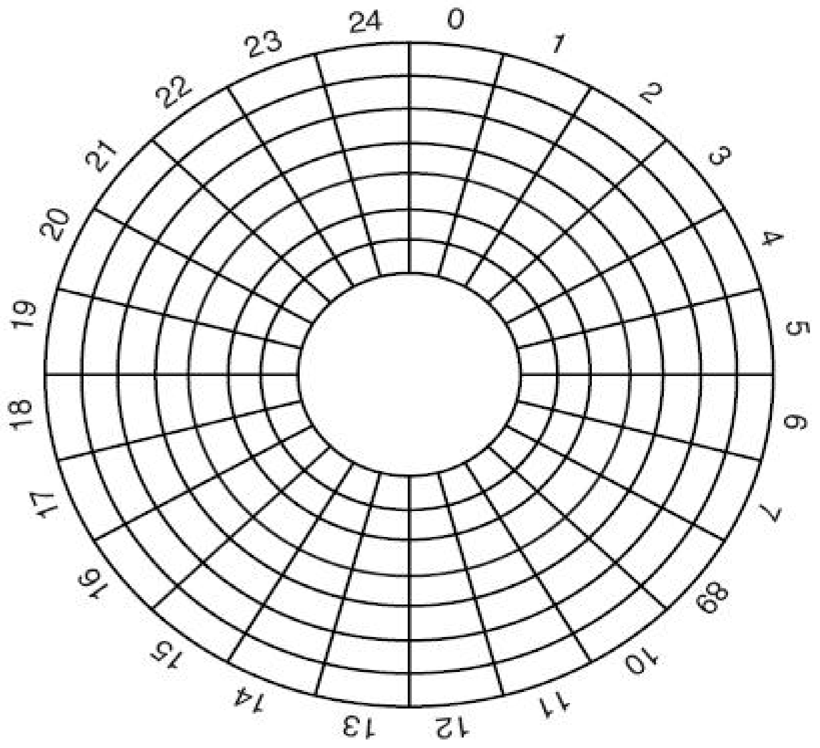
Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

Seek time is 7x better, transfer rate is 1300 x better, capacity is 50,000 x better.

Disks-more stuff

- Some disks have microcontrollers which do bad block re-mapping, track caching
- Some are capable of doing more then one seek at a time, i.e. they can read on one disk while writing on another
- Real disk geometry is different from geometry used by driver => controller has to re-map request for (cylinder, head,sector) onto actual disk
- Disks are divided into zones, with fewer tracks on the inside, gradually progressing to more on the outside

Disk Zones

(a) Physical geometry of a disk with two zones.
(b) A possible virtual geometry for this disk.

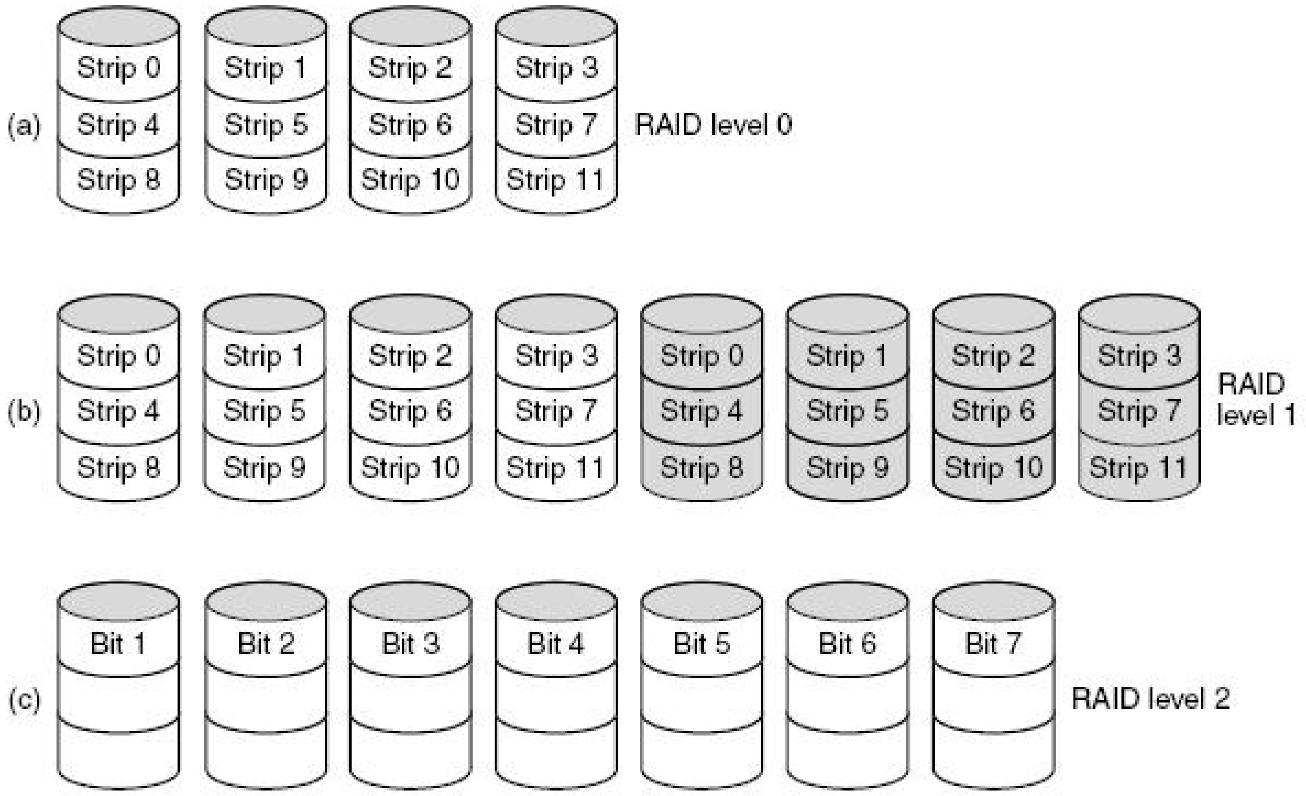
Redundant Array of Inexpensive Disks (RAID)

- Parallel I/O to improve performance and reliability
- vs SLED, Single Large Expensive Disk
- Bunch of disks which appear like a single disk to the OS
- SCSI disks often used-cheap, 7 disks per controller
- SCSI is set of standards to connect CPU to peripherals
- Different architectures-level 0 through level 7

RAID Levels

- Raid level 0 uses strips of k sectors per strip.
  - Consecutive strips are on different disks
  - Write/read on consecutive strips in parallel
  - Good for big enough requests
- Raid level 1 duplicates the disks
  - Writes are done twice, reads can use either disk
  - Improves reliability
- Level 2 works with individual words, spreading word + ecc over disks.
  - Need to synchronize arms to get parallelism

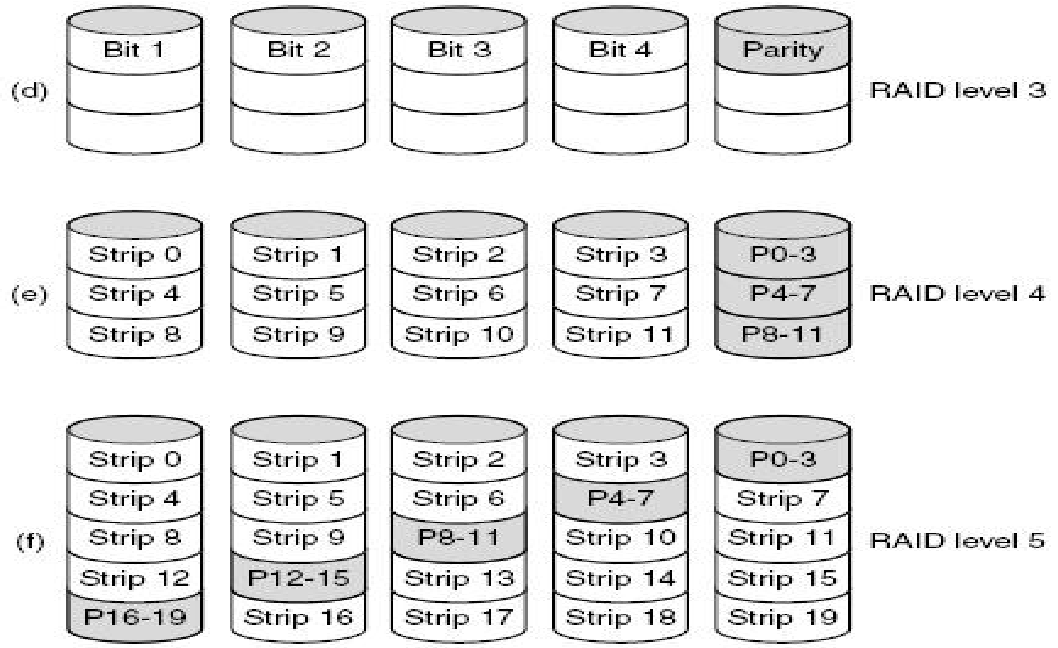
RAID Levels 0,1,2



Backup and parity drives are shown shaded.

RAID Levels 4 and 5

- Raid level 3 works like level 2, except all parity bits go on a single drive
- Raid 4,5 work with strips. Parity bits for strips go on separate drive (level 4) or several drives (level 5)



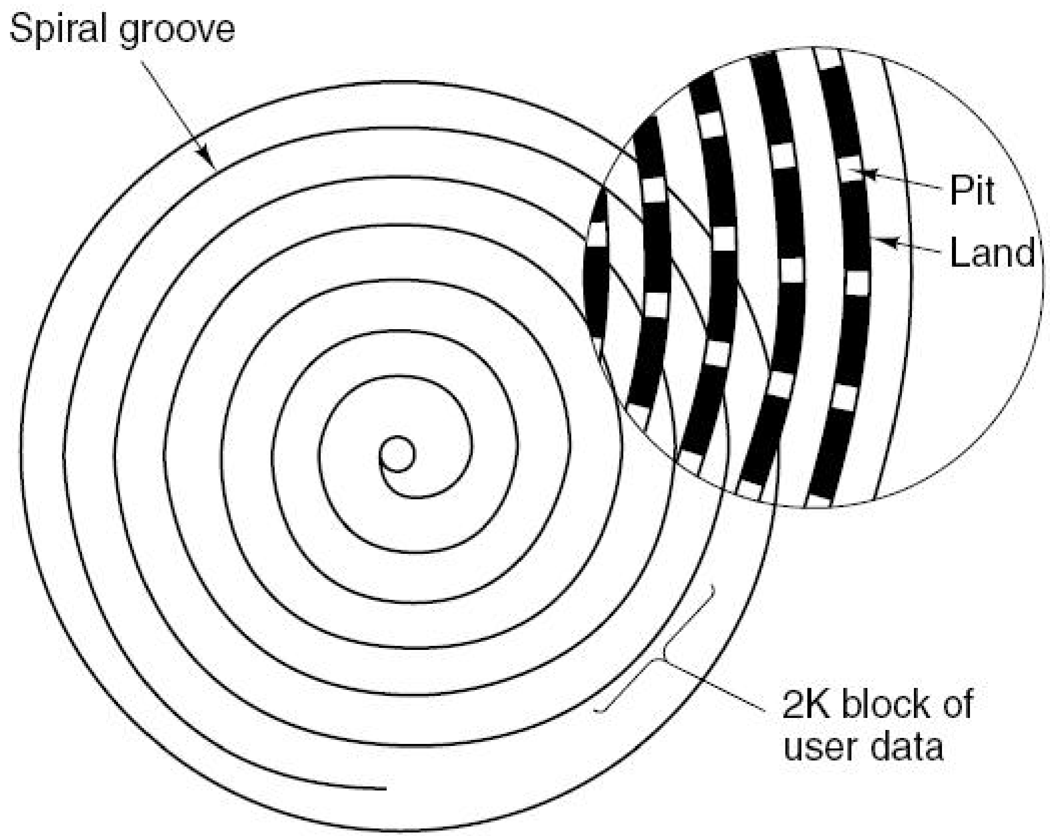
CD

- Optical disks have higher density then mag disks
- Used for distributing commercial software + reference works (books)
- Cheap because of high production volume of music CDs
- First used for playing music digitally

CD

- Laser burns holes on a master (coated glass) disk
- Mold is made with bumps where holes were
- Resin poured into –has same pattern of holes as glass disk
- Aluminum put on top of resin
- Pits (depressions) and lands (unburned area) are arranged in spirals
- Laser is used to read the pits and lands and convert them into bits (0 and 1)

CD

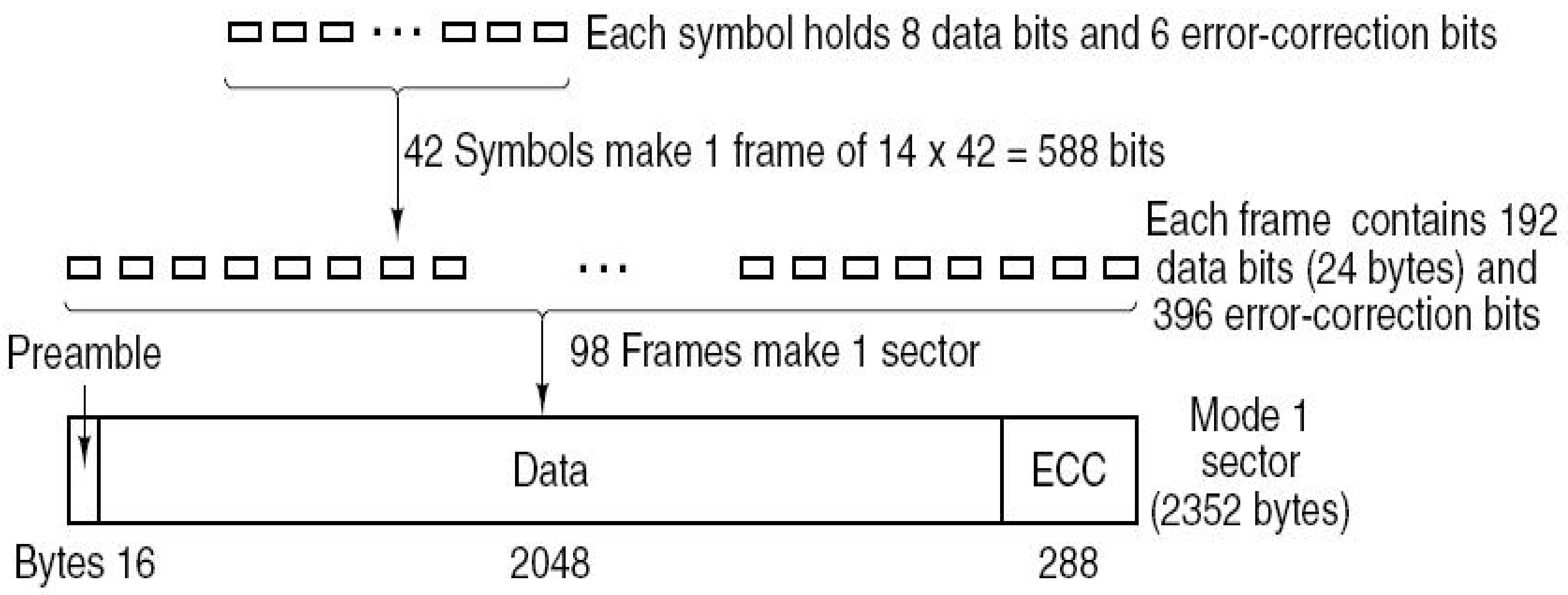


Recording structure of a compact disc or CD-ROM.

CD-ROM

- CD’s can be used to store data as well as audio
- Enter the CD-ROM
- Needed to improve the error-correcting ability of the CD
- Encode each byte (8 bits) in a 14 bit symbol with 6 bits of ECC
- 42 symbols form a frame
- Group 98 frames into a CD-ROM sector
- Extra error-correcting code is attached to sector

CD-ROMs Sector Layout


Each symbol holds 8 data bits and 6 error-correction bits
42 Symbols make 1 frame of 14 x 42 = 588 bits
Each frame contains 192 data bits (24 bytes) and 396 error-correction bits
Preamble
98 Frames make 1 sector
Data
ECC
Mode 1 sector (2352 bytes)
Bytes 16
2048
288

CD-ROM Performance

- 650 MB capacity
- 150 GB SCSI disk capacity
- 150 KB/sec in mode 1,
- up to 5 MB/sec for 32x CD-ROM
- Scsi-2 magnetic disk transfers at 10 MB/sec
- Bottom line: CD drives can’t compare to scsi disk drives

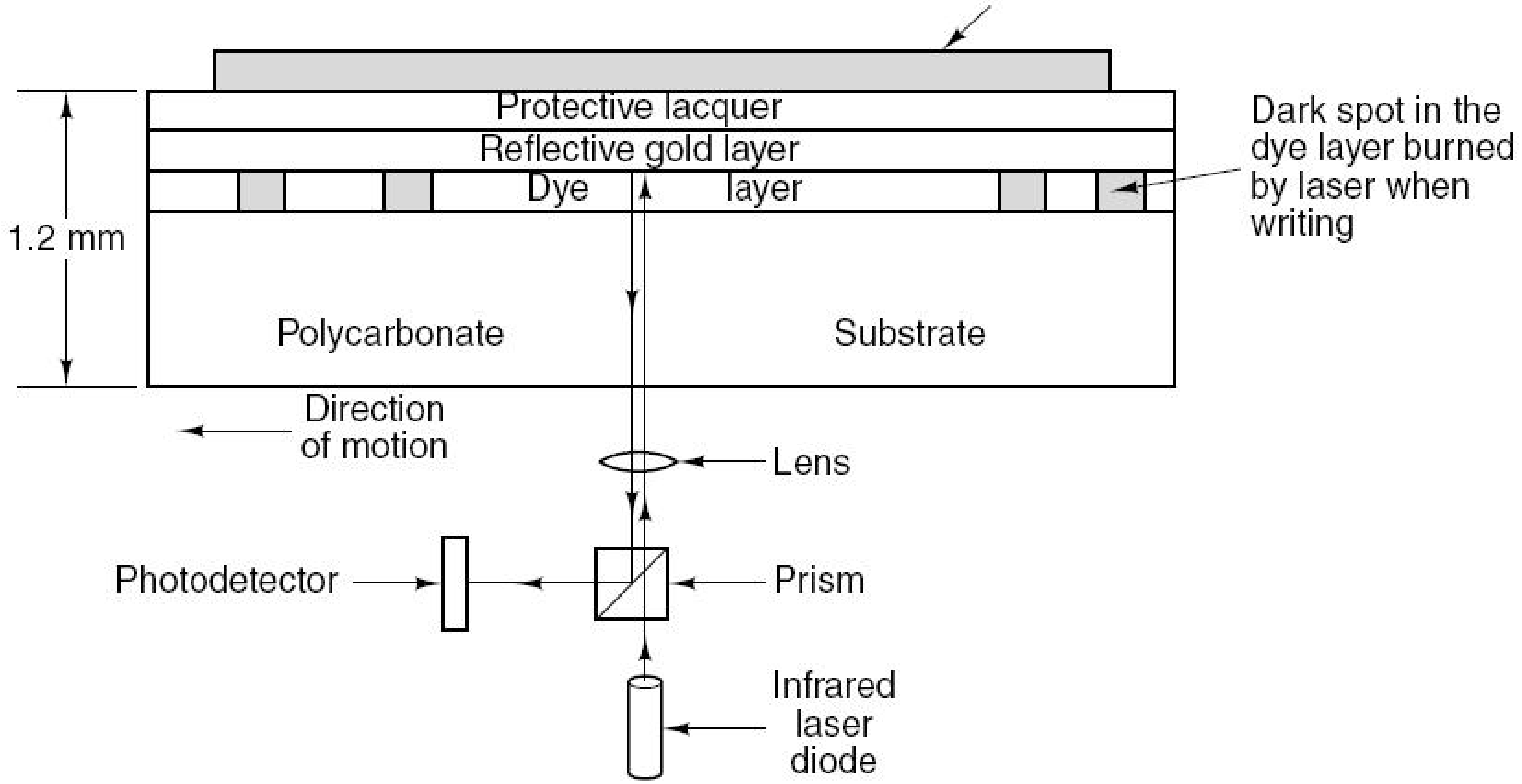
CD-ROM

- Added graphics, video, data
- File system standards agreed upon
- High Sierra for file names of 8 characters
- Rock ridge for longer names and extensions
- CD-ROM’s used for publishing games, movies, commercial software, reference works
- Why? Cheap to manufacture and large capacity

CD-Recordable

- Cheaper manufacturing process led to cheaper CD-ROM (CD-R)
- Used as backup to disk drives
- Small companies can use to make masters which they give to high volume plants to reproduce

CD-Recordables



Cross section of a CD-R disk and laser. A silver CD-ROM has similar structure, except without dye layer and with pitted aluminum layer instead of gold layer.

DVD (Digital Versatile Disk)

DVD use same design as CD with a few improvements

1. Smaller pits
   (0.4 microns versus 0.8 microns for CDs).
2. A tighter spiral
   (0.74 microns between tracks versus 1.6 microns for CDs).
3. A red laser
   (at 0.65 microns versus 0.78 microns for CDs).

DVD (Digital Versatile Disk)

- This led to much bigger capacity ~ 5 Gbyte (seven fold increase in capacity)
- Can put a standard movie on the DVD (133 minutes)
- Hollywood wants more movies on the same disk, so have 4 formats

DVD

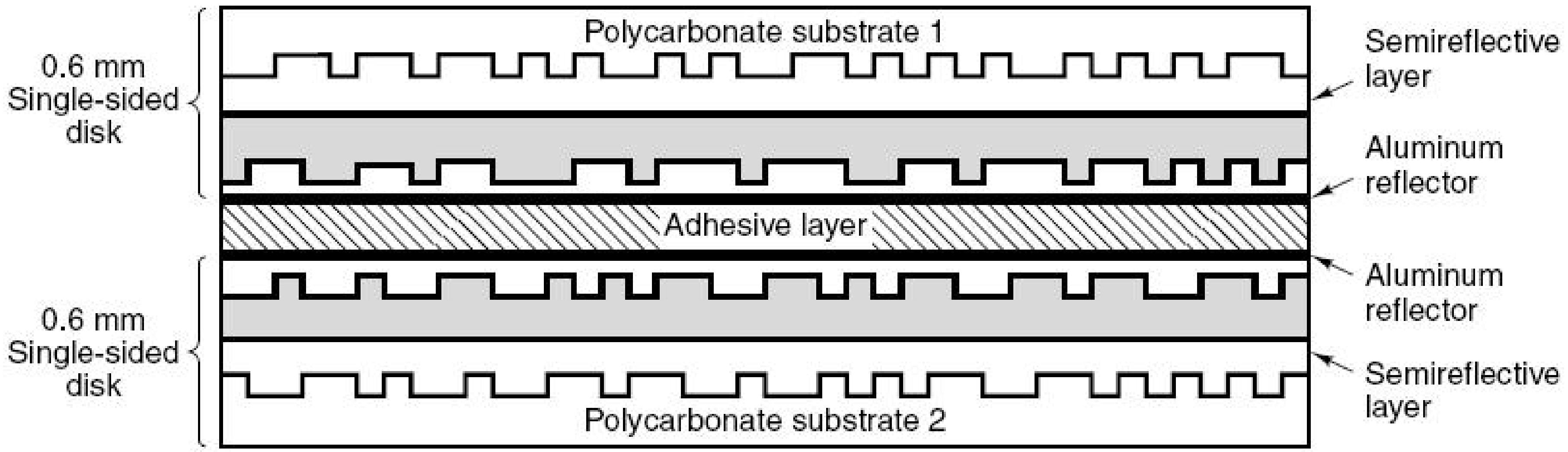
DVD Formats

1. Single-sided, single-layer (4.7 GB).
2. Single-sided, dual-layer (8.5 GB).
3. Double-sided, single-layer (9.4 GB).
4. Double-sided, dual-layer (17 GB).

DVD: next generation

- Blu-ray
- HD
- Computer industry and Hollywood have not agreed on formats yet!!

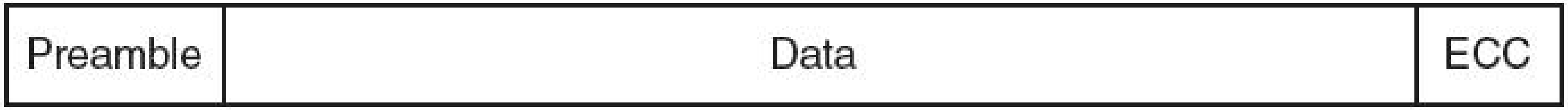
DVD



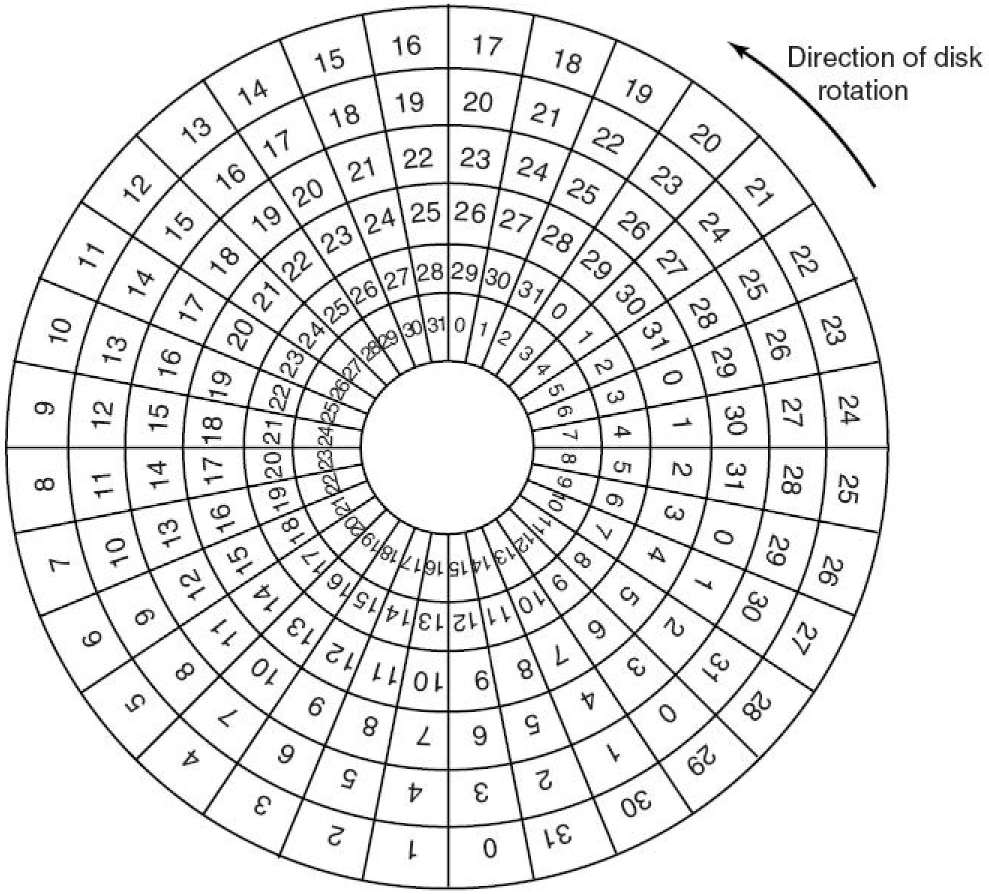
A double-sided, dual-layer DVD disk.

5.4.2 Hard Disk Formatting

- Low level format-software lays down tracks and sectors on empty disk (picture next slide)
- High level format is done next-partitions
- Sector format
  - 512 bit sectors standard
  - Preamble contains address of sector, cylinder number
  - ECC for recovery from errors

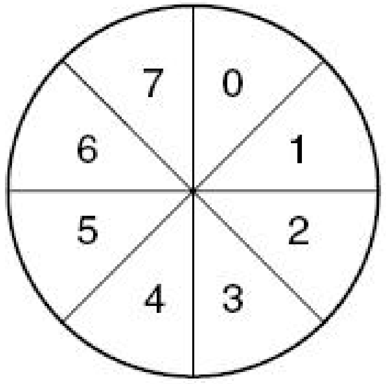


Cylinder Skew

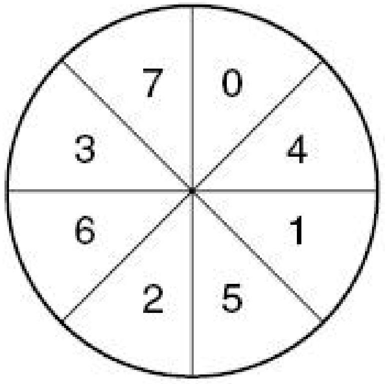


Offset sector from one track to next one in order to get consecutive sectors

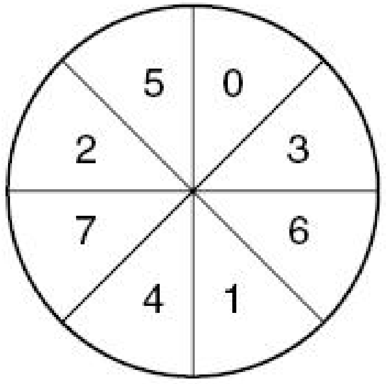
Interleaved sectors



(a)



(b)



(c)

Copying to a buffer takes time; could wait a disk rotation before head reads next sector.
So interleave sectors to avoid this (b,c)

5.4.2 Hard Disk Fomatting

- High level format
  - Partitions-for more then one OS on same disk
  - Pentium-sector 0 has master boot record with partition table and code for boot block
  - Pentium has 4 partitions-can have both Windows and Unix
  - In order to be able to boot, one sector has to be marked as active
  - For each partitions:
    - master boot record in sector 0
    - boot block program
    - free storage admin (bitmap or free list)
    - root directory
    - empty file system
    - indicates which file system is in the partition (in the partition table)

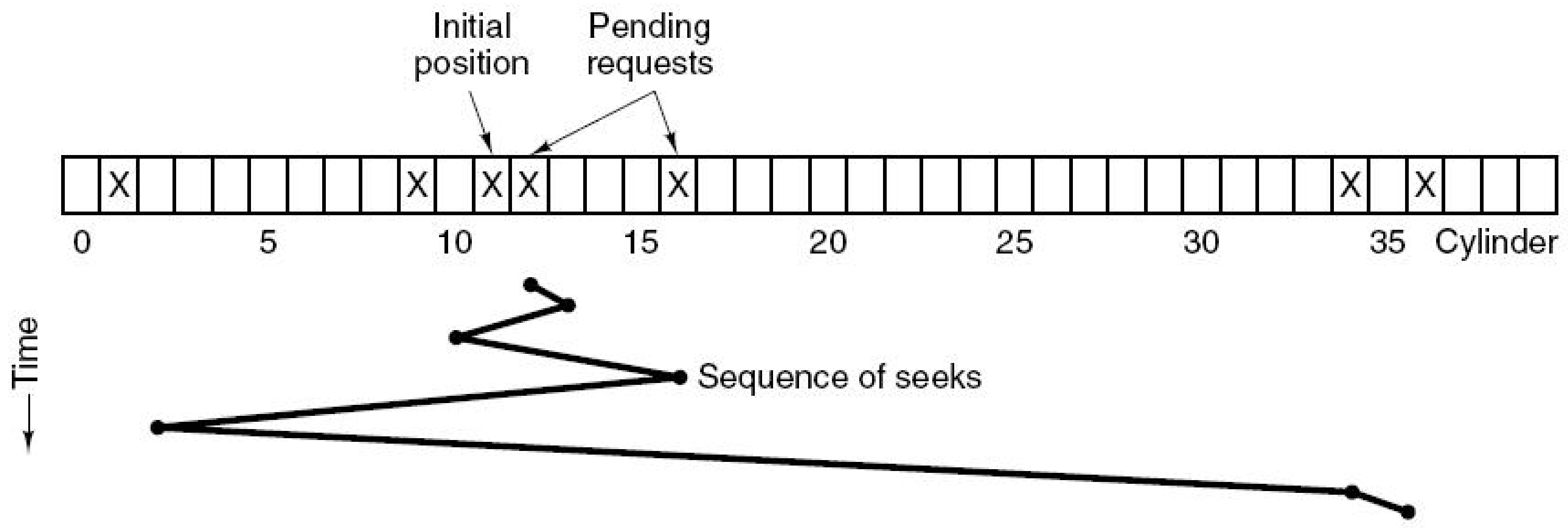
Power goes on

- BIOS reads in master boot record
- Boot program checks which partition is active
- Reads in boot sector from active partition
- Boot sector loads bigger boot program which looks for the OS kernel in the file system
- OS kernel is loaded and executed

5.4.3 Disk Arm Scheduling Algorithms

- Read/write time factors
  - Seek time (the time to move the arm to the proper cylinder).
  - Rotational delay (the time for the proper sector to rotate under the head).
  - Actual data transfer time.
- Driver keeps list of requests (cylinder number, time of request)
- Try to optimize the seek time
- Some disk arm scheduling algorithms
  - FCFS is easy to implement, but optimizes nothing

SSF (Shortest Seek Time First)



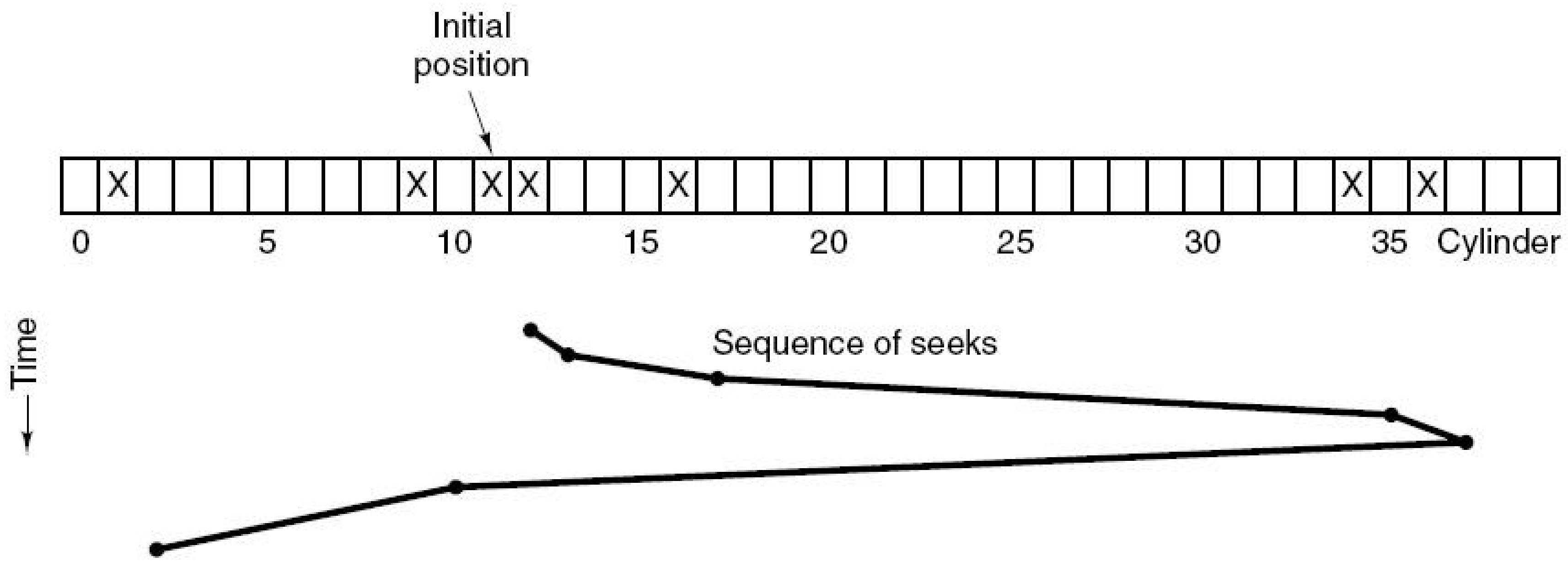
While head is on cylinder 11, requests for 1,36,16,34,9,12 come in

- FCFS would result in 111 cylinders
- SSTF would require 1,3,7,15,33,2 movements for a total of 61 cylinders

Elevator algorithm

- It is a greedy algorithm-the head could get stuck in one part of the disk if the usage was heavy
- Elevator-keep going in one direction until there are no requests in that direction, then reverse direction
- Real elevators sometimes use this algorithm
- Variation on a theme-first go one way, then go the other

The Elevator

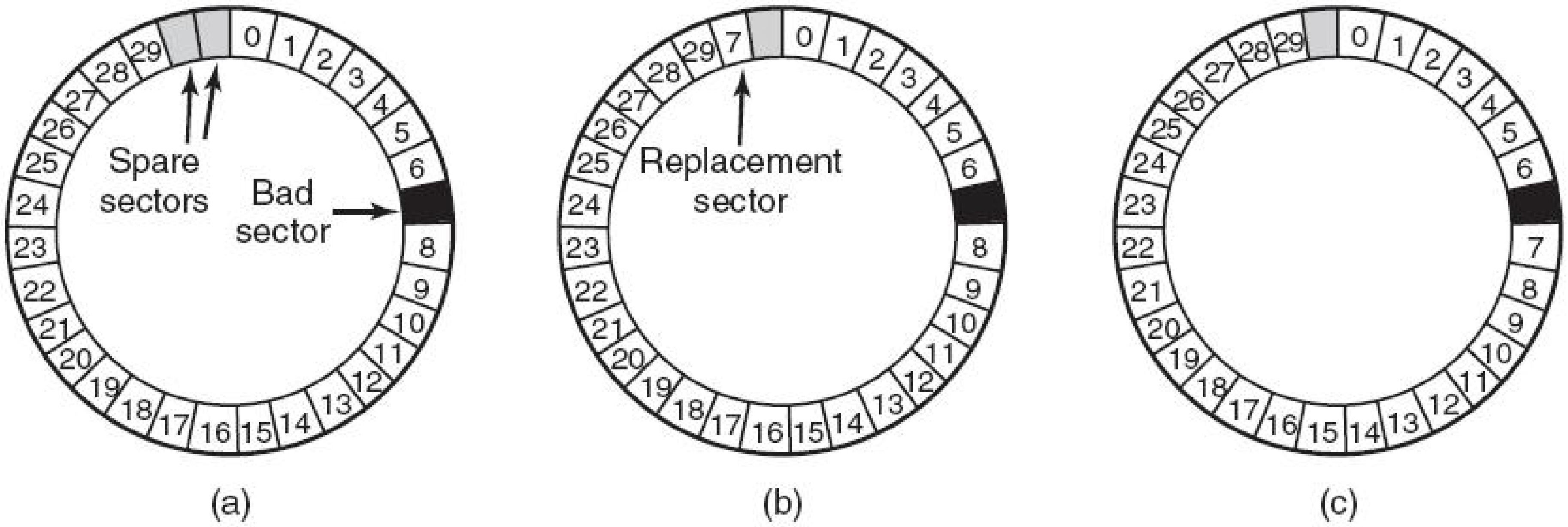


Uses 60 cylinders, a bit better

Disk Controller Cache

- Disk controllers have their own cache
- Cache is separate from the OS cache
- OS caches blocks independently of where they are located on the disk
- Controller caches blocks which were easy to read but which were not necessarily requested

5.4.4 Error Handling



(a) A disk track with a bad sector. (b) Substituting a spare for the bad sector.
(c) Shifting all the sectors to bypass the bad one.

Bad Sectors-the controller approach

- Manufacturing defect-that which was written does not correspond to that which is read (back)
- Controller or OS deals with bad sectors
- If controller deals with them the factory provides a list of bad blocks and controller remaps good spares in place of bad blocks
- Substitution can be done when the disk is in use-controller “notices” that block is bad and substitutes

Bad Sectors-the OS approach

- Gets messy if the OS has to do it
- OS needs lots of information-which blocks are bad or has to test blocks itself

5.4.5 Stable Storage

- RAIDS can protect against sectors going bad
- Can't protect against write operations spitting out garbage or crashes during writes
- Stable storage: either correct data is laid down or old data remains in place
- Necessary for some apps-data can't be lost or go bad

Assumptions

- Can detect a bad write on subsequent reads via ECC
- Probability of having bad data in sector on two different disks is negligible
- If CPU fails, it stops along with any write in progress at the time. Bad data can be detected later via ECC during read op

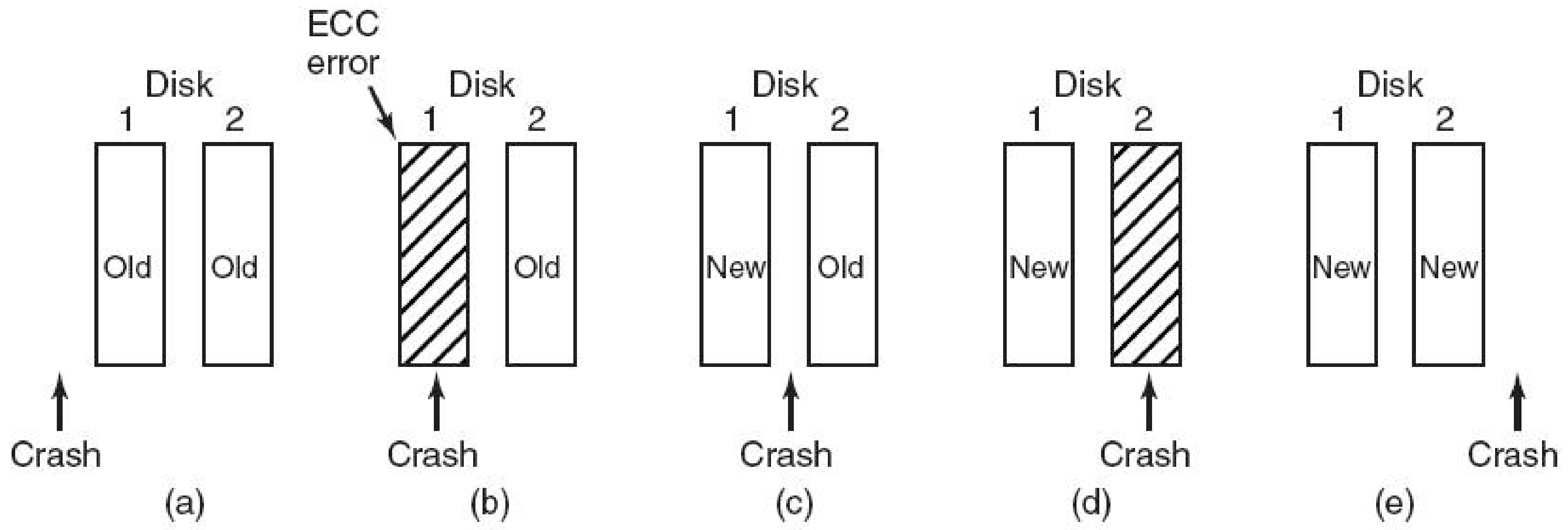
The idea and the operations

- Use 2 identical disks-do the same thing to both disks
- Use 3 operations
- Stable write-first write, then read back and compare. If they are the same write to second disk. If write fails, try up to n times to get it to succeed. After n failures keep using spare sectors until it succeeds. Then go to disk 2.

The idea and the OPS

- Stable read-read from disk 1 n times until get a good ECC, otherwise read from disk 2 (assumption that probability of both sectors being bad is negligible)
- Crash recovery-read both copies of blocks and compare them. If one block has an ECC error, overwrite it with the good block. If both pass the ECC test, then pick either

CPU Crashes



(a)Crash happens before write

(b) crash happens during write to 1

(c)crash happens after 1 but before 2

(d) during 2, after 1

(e) both are the same

5.5 Clock

5.5.1 Clock Hardware

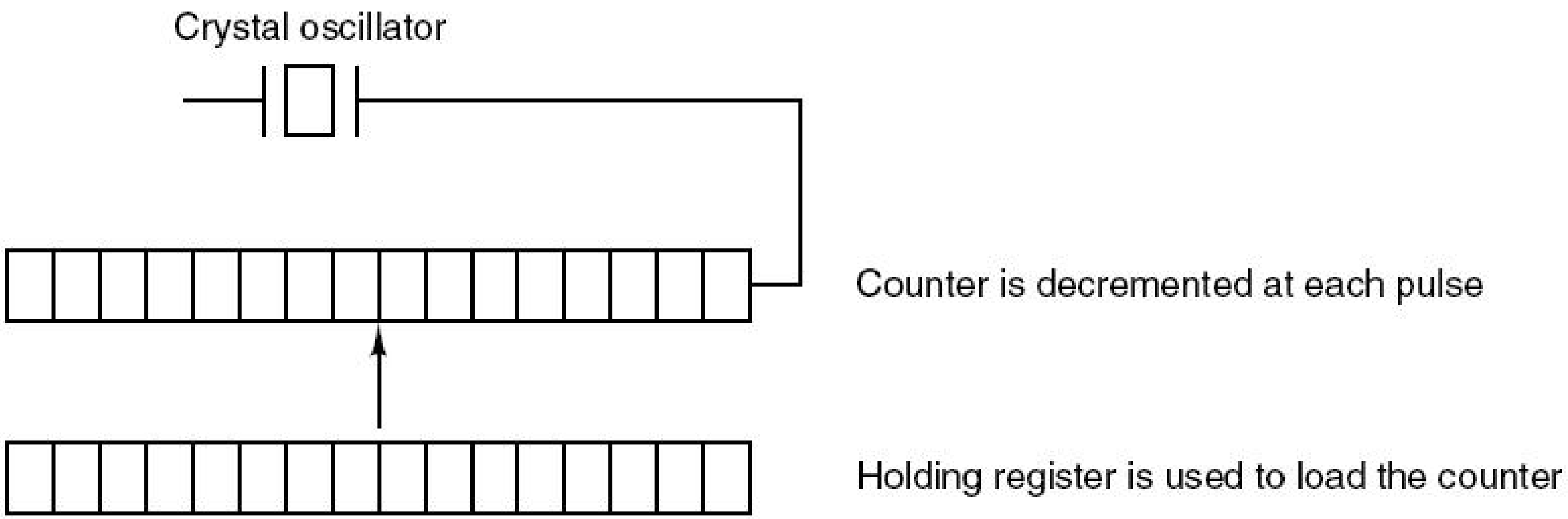

Crystal oscillator
Counter is decremented at each pulse
Holding register is used to load the counter

Figure 5-32. A programmable clock.

5.5.2 Clock Software

- Typical duties of a clock driver
  - Maintaining the time of day.
  - Preventing processes from running longer than they are allowed to.
  - Accounting for CPU usage.
  - Handling alarm system call made by user processes.
  - Providing watchdog timers for parts of the system itself.
  - Doing profiling, monitoring, statistics gathering.

5.5.2 Clock Software

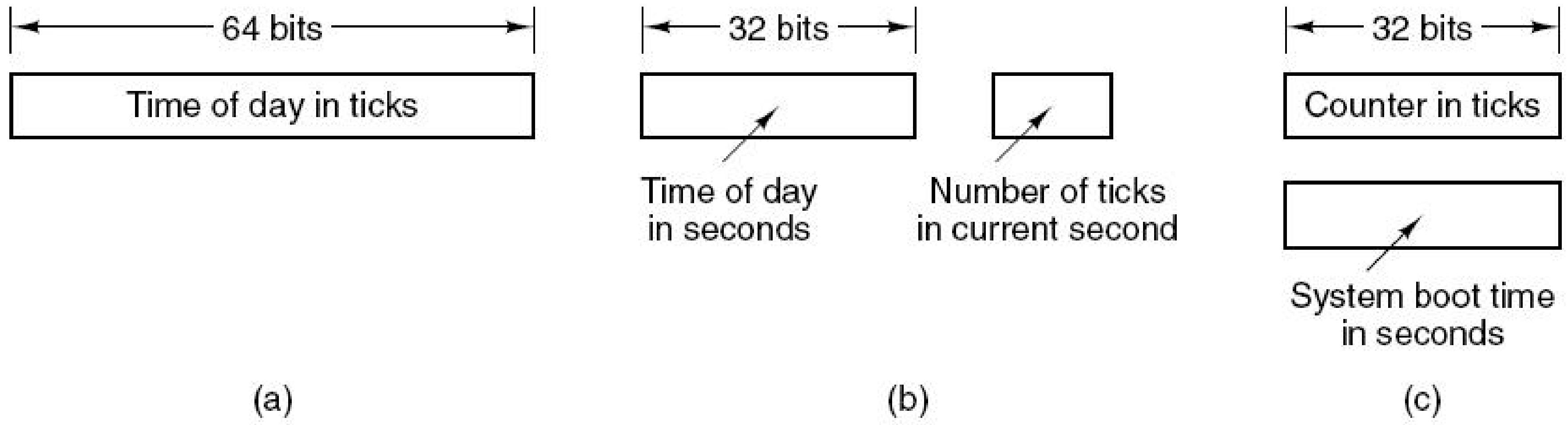


Figure 5-33. Three ways to maintain the time of day.

5.5.2 Clock Software

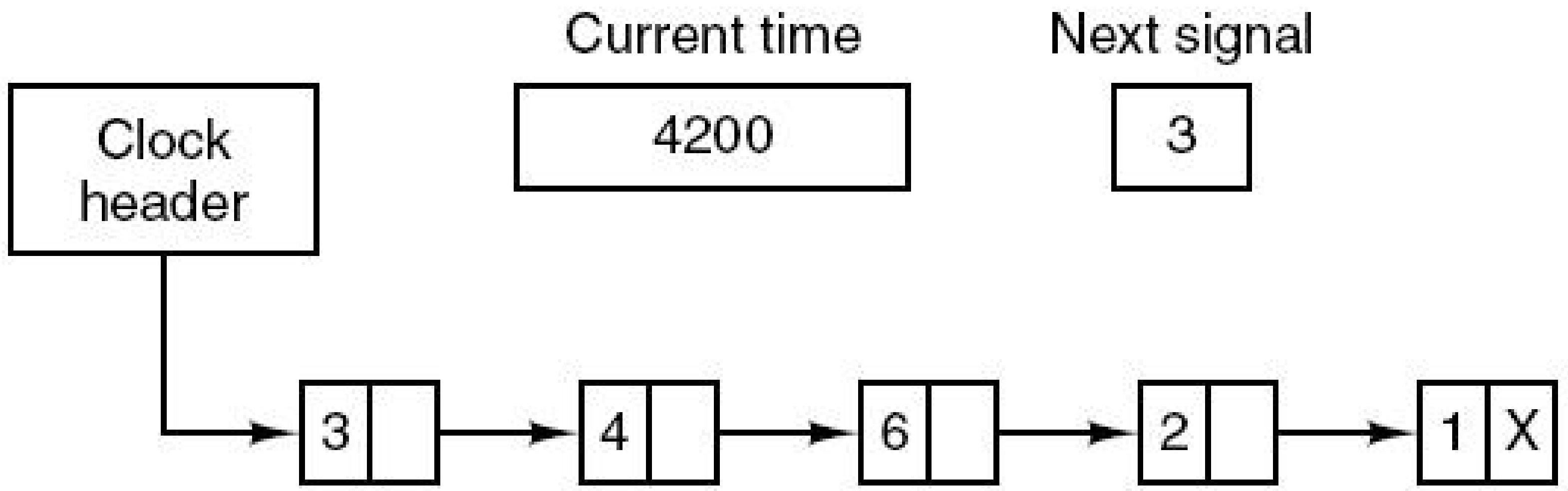
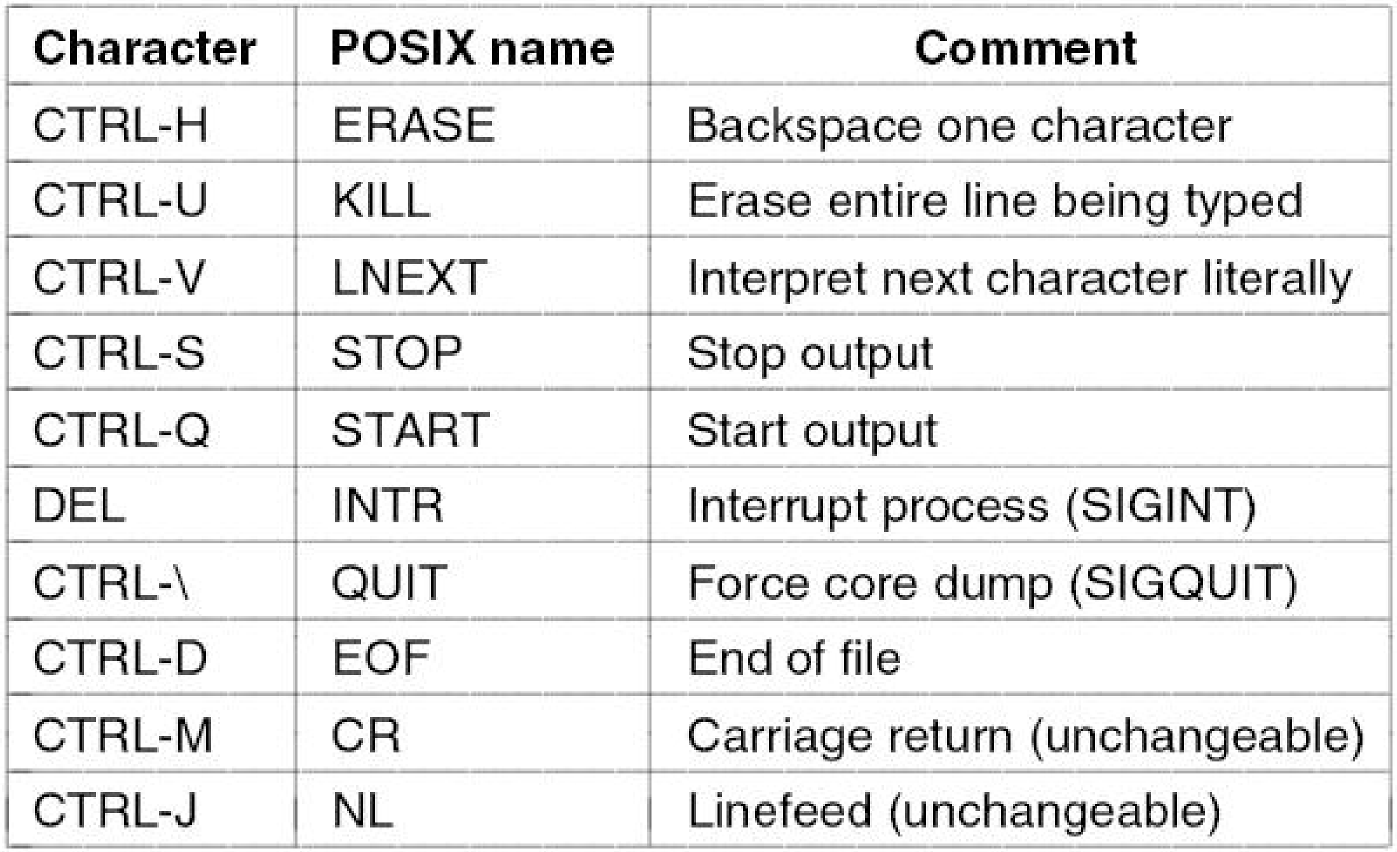


Figure 5-34. Simulating multiple timers with a single clock.

5.5.3 Soft Timers

- Soft timers succeed according to rate at which kernel entries are made because of:
  - System calls.
  - TLB misses.
  - Page faults.
  - I/O interrupts.
  - The CPU going idle.

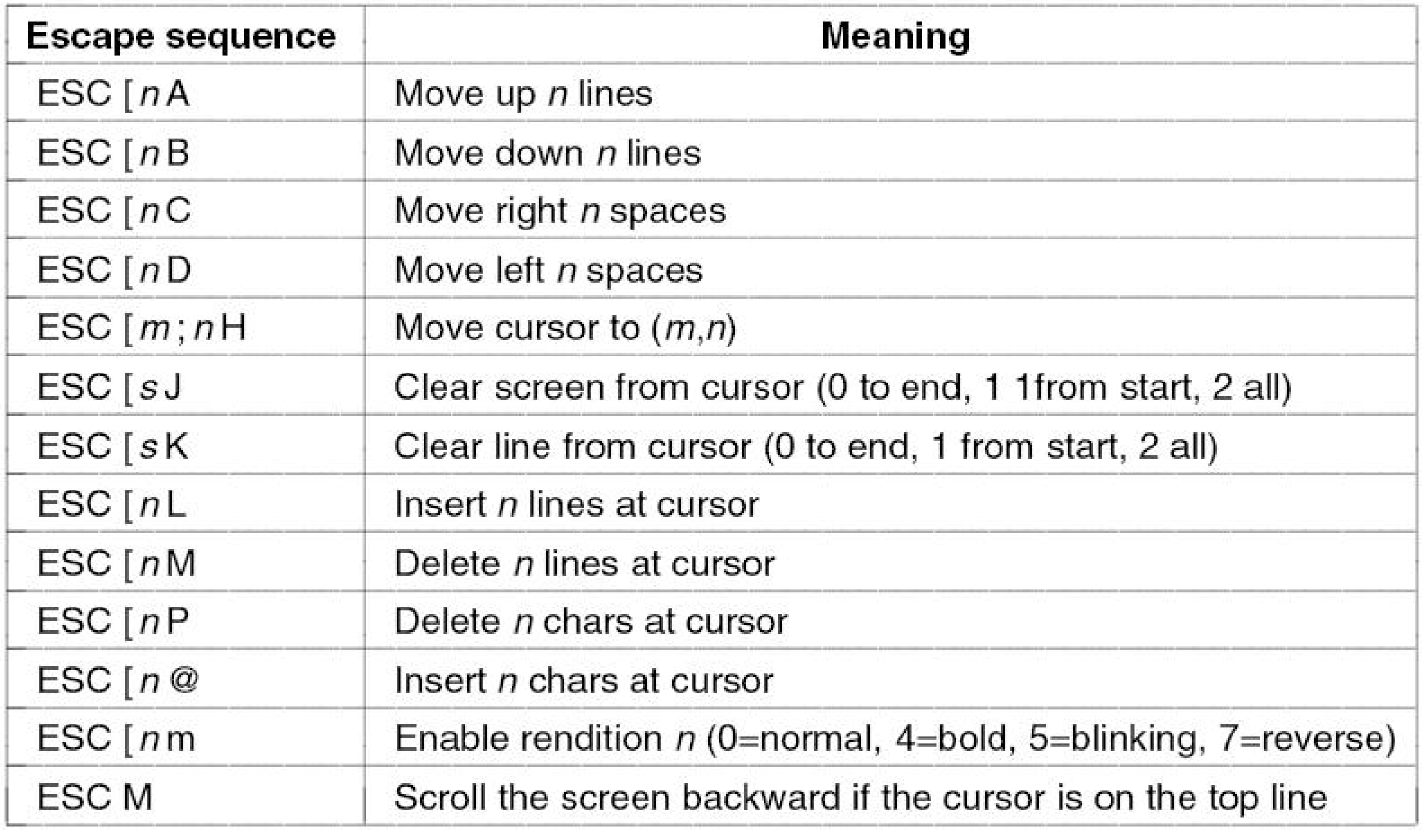
5.6 User Interface
5.6.1 Keyboard Software



Character	POSIX name	Comment
CTRL-H	ERASE	Backspace one character
CTRL-U	KILL	Erase entire line being typed
CTRL-V	LNEXT	Interpret next character literally
CTRL-S	STOP	Stop output
CTRL-Q	START	Start output
DEL	INTR	Interrupt process (SIGINT)
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-D	EOF	End of file
CTRL-M	CR	Carriage return (unchangeable)
CTRL-J	NL	Linefeed (unchangeable)

Figure 5-35. Characters that are handled specially in canonical mode.

5.6.2 The X Window System



Escape sequence	Meaning
ESC [*n* A	Move up *n* lines
ESC [*n* B	Move down *n* lines
ESC [*n* C	Move right *n* spaces
ESC [*n* D	Move left *n* spaces
ESC [*m* ; *n* H	Move cursor to (*m*,*n*)
ESC [*s* J	Clear screen from cursor (0 to end, 1 1from start, 2 all)
ESC [*s* K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n* L	Insert *n* lines at cursor
ESC [*n* M	Delete *n* lines at cursor
ESC [*n* P	Delete *n* chars at cursor
ESC [*n* @	Insert *n* chars at cursor
ESC [*n* m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

Figure 5-36. The ANSI escape sequences accepted by the terminal driver on output. ESC denotes the ASCII escape character (0x1B), and *n*, *m*, and *s* are optional numeric parameters.

5.6.2 The X Window System

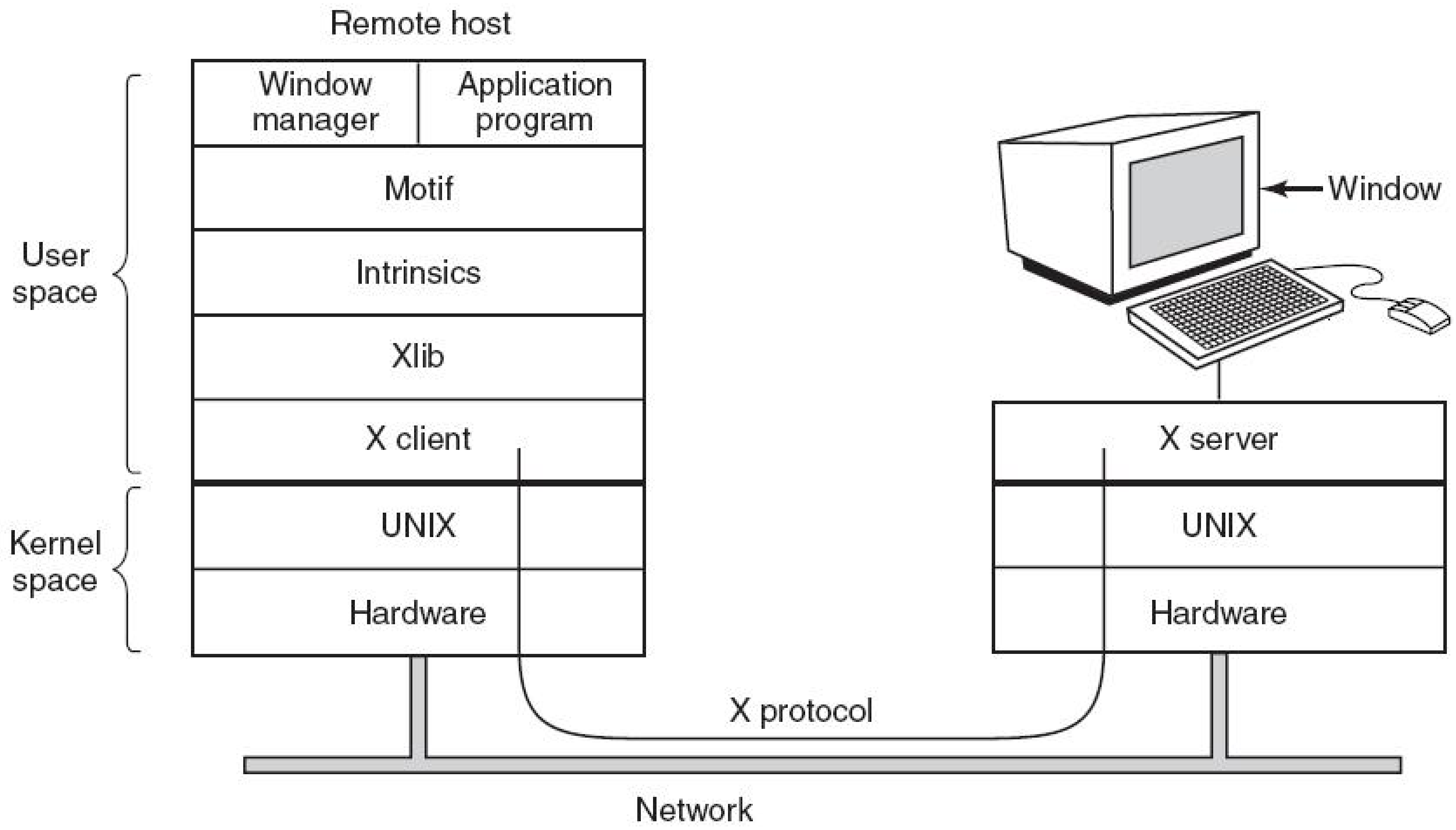
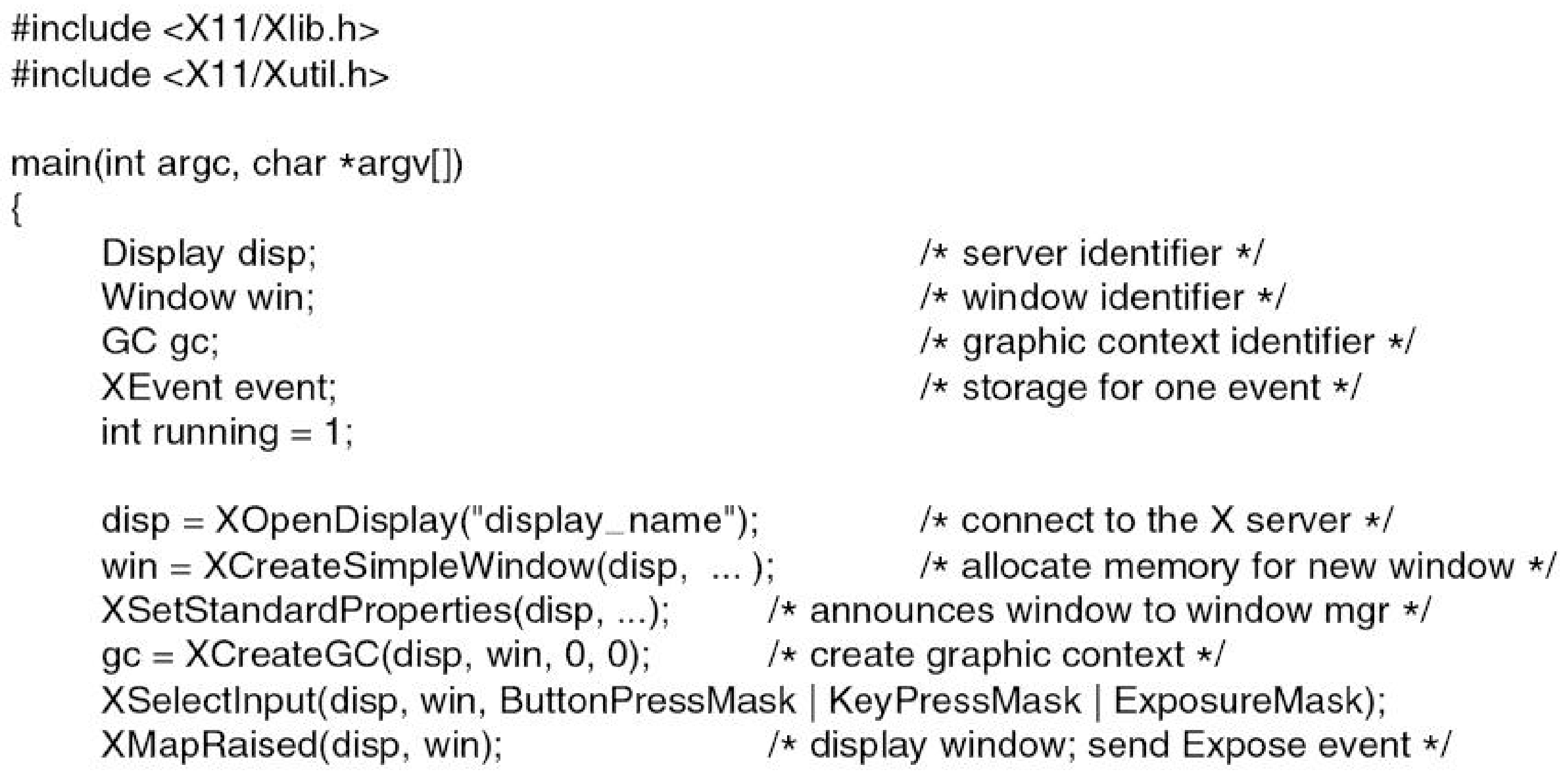


Figure 5-37. Clients and servers in the M.I.T. X Window System.

5.6.2 The X Window System

- Types of messages between client and server:
  - Drawing commands from the program to the workstation.
  - Replies by the workstation to program queries.
  - Keyboard, mouse, and other event announcements.
  - Error messages.

5.6.3 Graphical User Interfaces



```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(int argc, char *argv[])
{
      Display disp;                                        /* server identifier */
      Window win;                                          /* window identifier */
      GC gc;                                               /* graphic context identifier */
      XEvent event;                                        /* storage for one event */
      int running = 1;

      disp = XOpenDisplay("display_name");                 /* connect to the X server */
      win = XCreateSimpleWindow(disp, ... );               /* allocate memory for new window */
      XSetStandardProperties(disp, ...);           /* announces window to window mgr */
      gc = XCreateGC(disp, win, 0, 0);             /* create graphic context */
      XSelectInput(disp, win, ButtonPressMask | KeyPressMask | ExposureMask);
      XMapRaised(disp, win);                       /* display window; send Expose event */
```

Figure 5-38. A skeleton of an X Window application program.

5.6.3 Graphical User Interfaces

```
    while (running) {
        XNextEvent(disp, &event);               /* get next event */
        switch (event.type) {
            case Expose:         ...;   break;        /* repaint window */
            case ButtonPress:    ...;   break;        /* process mouse click */
            case Keypress:       ...;   break;        /* process keyboard input */
        }
    }

    XFreeGC(disp, gc);                          /* release graphic context */
    XDestroyWindow(disp, win);                  /* deallocate window's memory space */
    XCloseDisplay(disp);                        /* tear down network connection */
}
```

Figure 5-38. A skeleton of an X Window application program.

5.6.3 Graphical User Interfaces

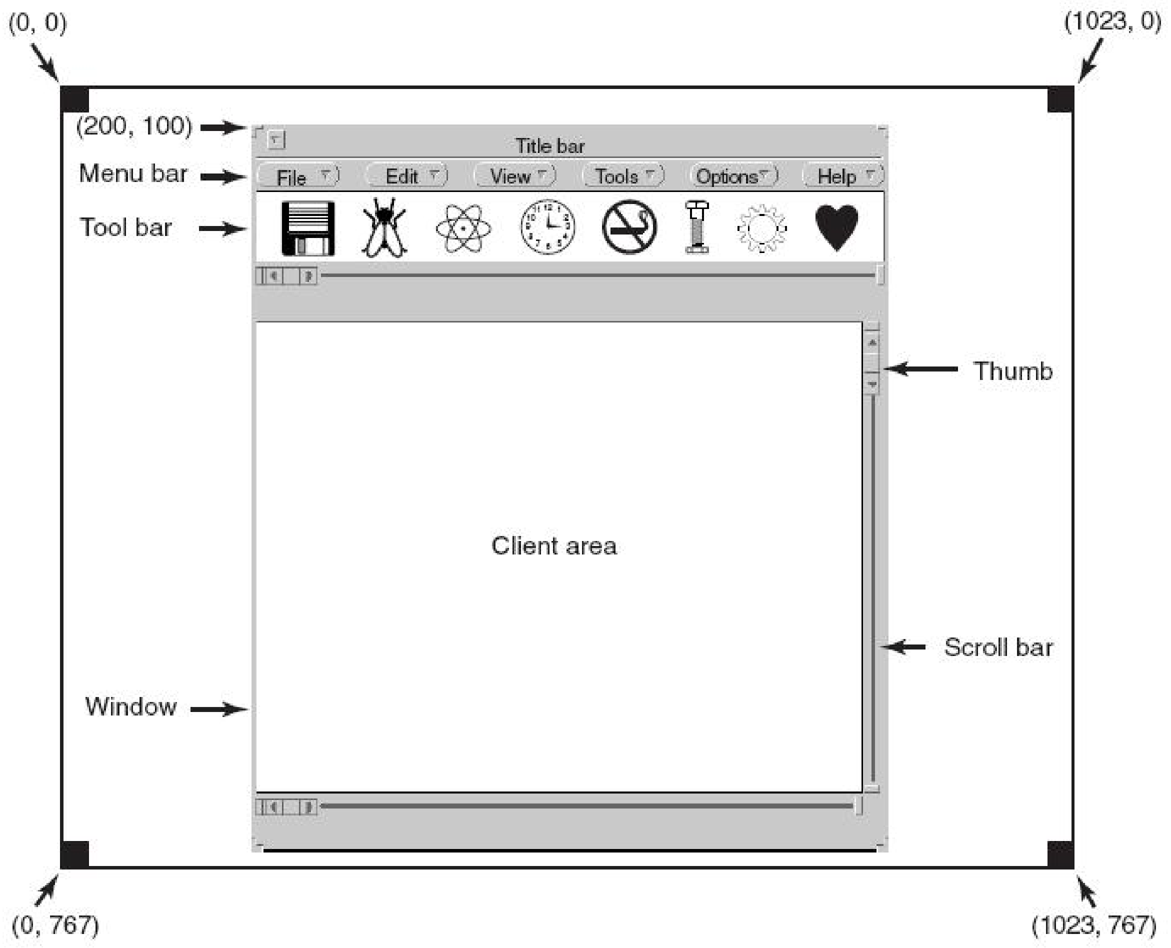


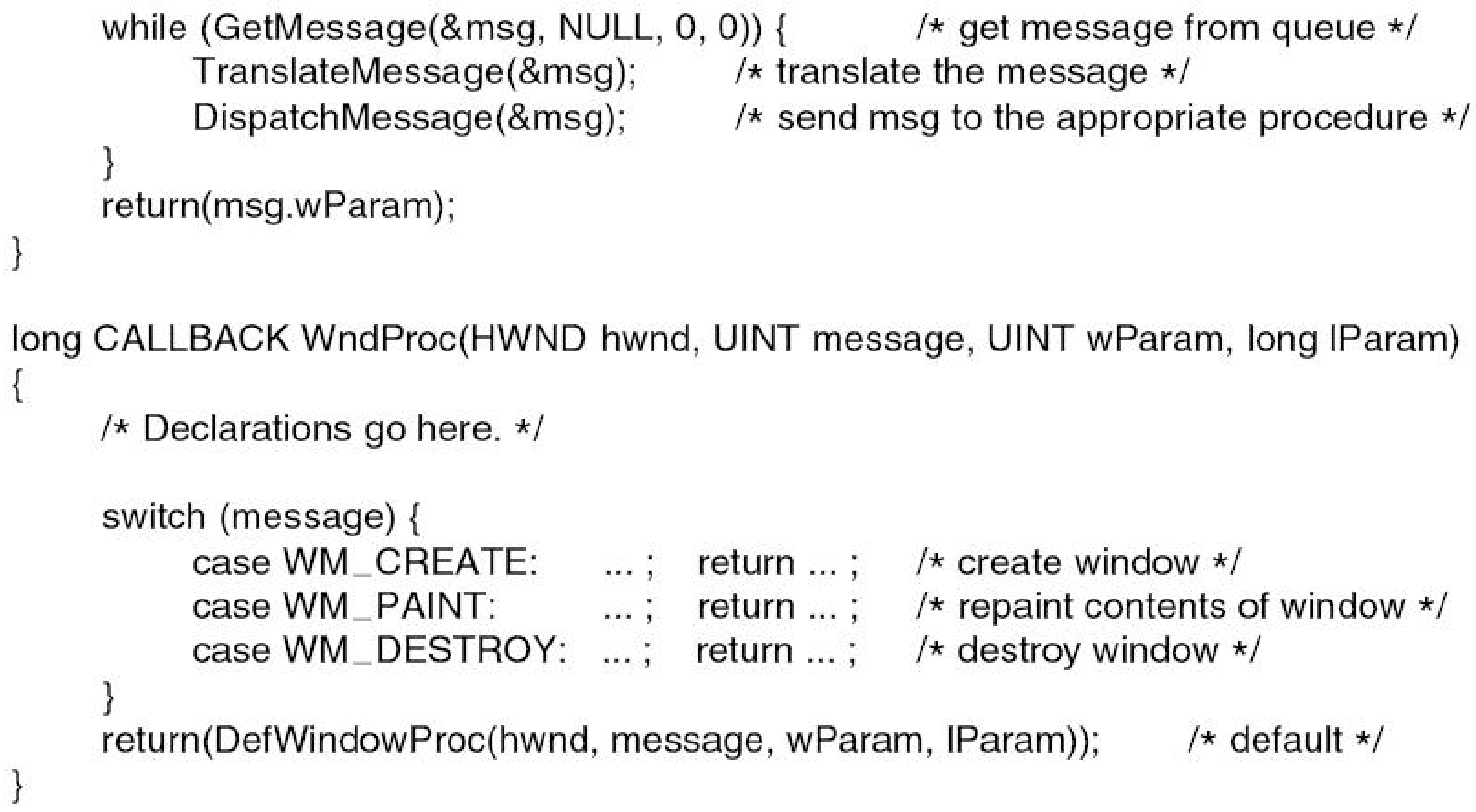
Figure 5-39. A sample window located at (200, 100) on an XGA display.

5.6.3 Graphical User Interfaces

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE h, HINSTANCE, hprev, char *szCmd, int iCmdShow)
{
      WNDCLASS wndclass;                    /* class object for this window */
      MSG msg;                              /* incoming messages are stored here */
      HWND hwnd;                            /* handle (pointer) to the window object */

      /* Initialize wndclass */
      wndclass.lpfnWndProc = WndProc;       /* tells which procedure to call */
      wndclass.lpszClassName = "Program name";    /* Text for title bar */
      wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);    /* load program icon */
      wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);      /* load mouse cursor */

      RegisterClass(&wndclass);             /* tell Windows about wndclass */
...   hwnd = CreateWindow ( ... )           /* allocate storage for the window */
      ShowWindow(hwnd, iCmdShow);           /* display the window on the screen */
      UpdateWindow(hwnd);                   /* tell the window to paint itself */
```

Figure 5-40. A skeleton of a Windows main program.

5.6.3 Graphical User Interfaces



```
        while (GetMessage(&msg, NULL, 0, 0)) {          /* get message from queue */
                TranslateMessage(&msg);         /* translate the message */
                DispatchMessage(&msg);          /* send msg to the appropriate procedure */
        }
        return(msg.wParam);
}

long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)
{
        /* Declarations go here. */

        switch (message) {
                case WM_CREATE:      ... ;   return ... ;    /* create window */
                case WM_PAINT:       ... ;   return ... ;    /* repaint contents of window */
                case WM_DESTROY:     ... ;   return ... ;    /* destroy window */
        }
        return(DefWindowProc(hwnd, message, wParam, lParam));       /* default */
}
```

Figure 5-40. A skeleton of a Windows main program.

5.6.4 Bitmaps

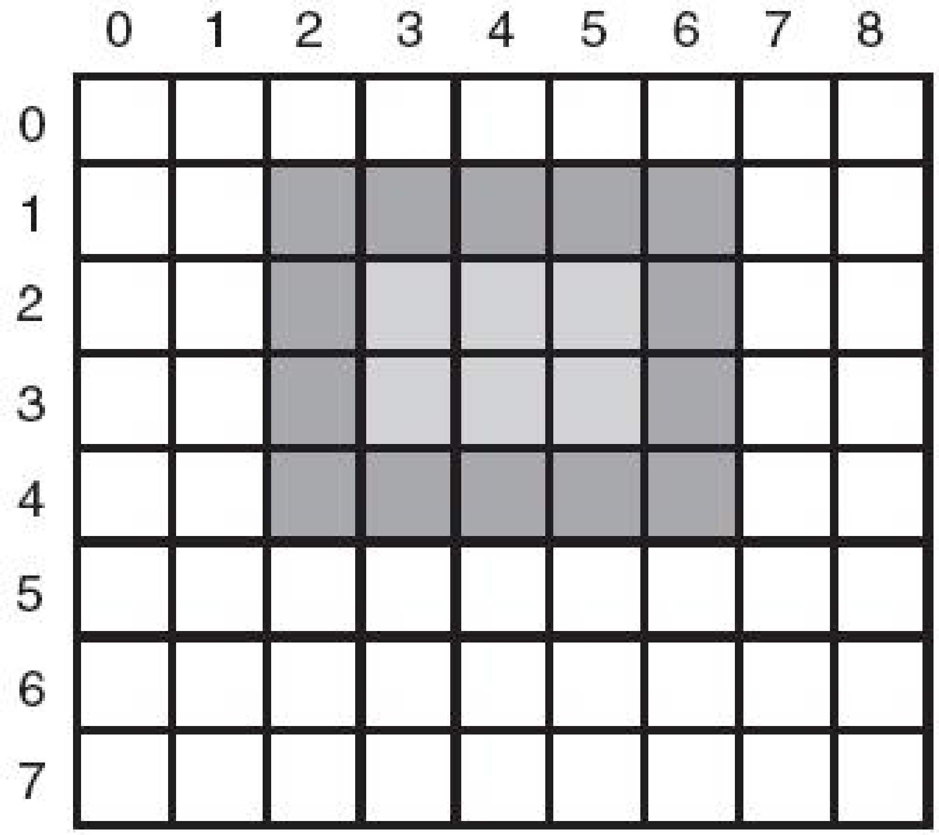


Figure 5-41. An example rectangle drawn using Rectangle. Each box represents one pixel.

5.6.4 Bitmaps

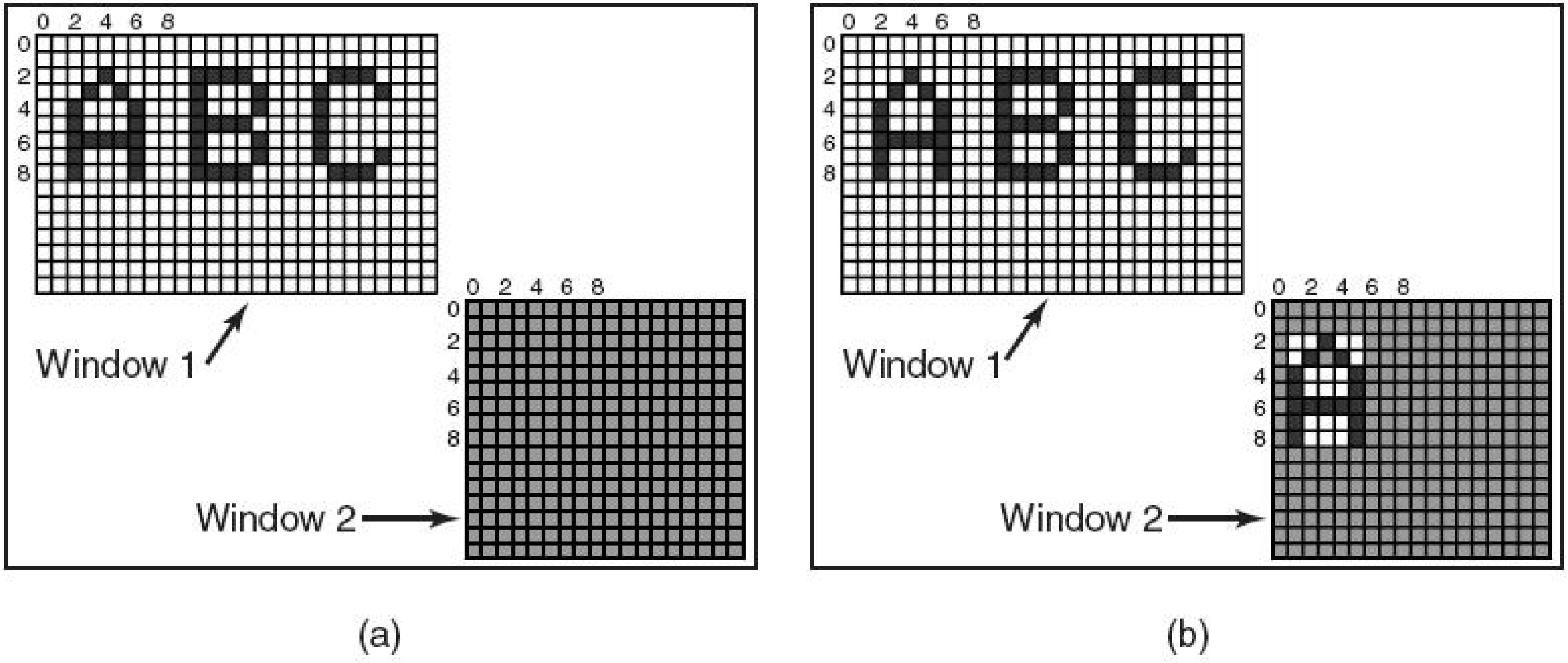


Figure 5-42. Copying bitmaps using *BitBlt*. (a) Before. (b) After.

Fonts

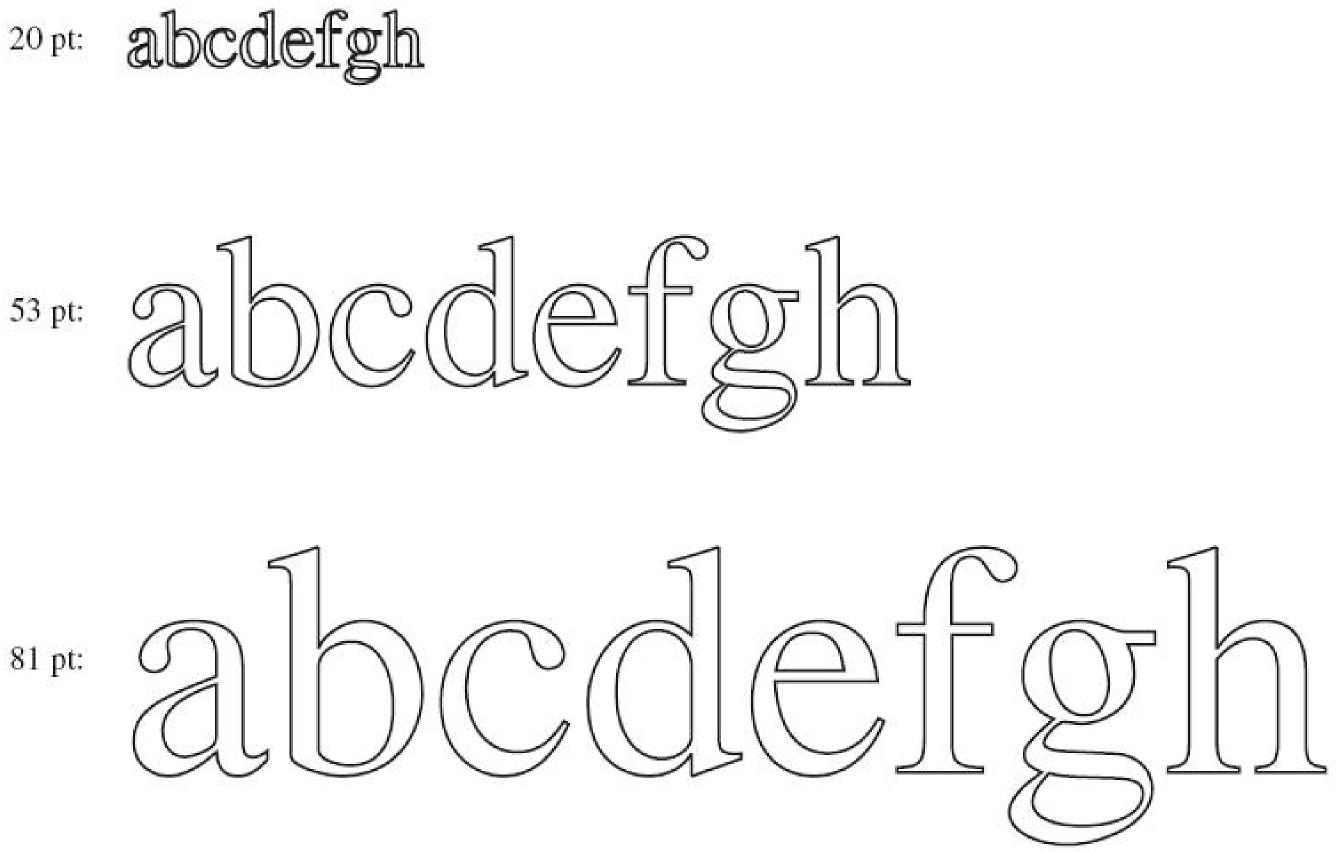


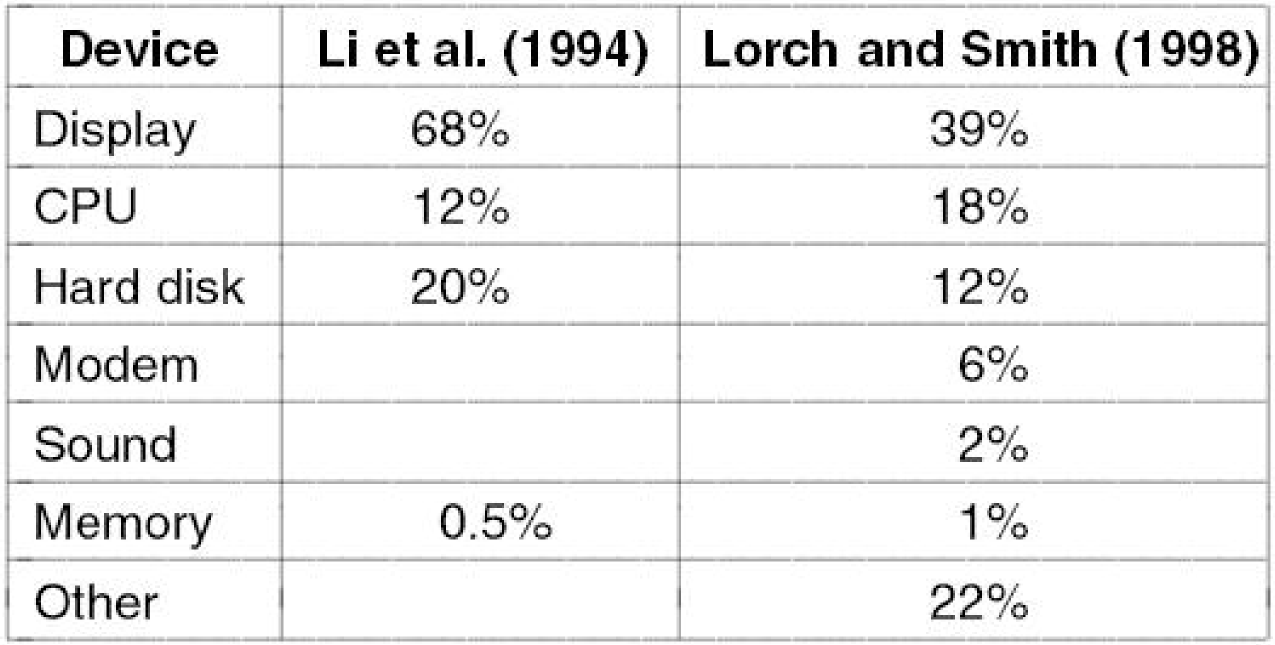
Figure 5-43. Some examples of character outlines at different point sizes.

5.7 Thin Clients

Command	Description
Raw	Display raw pixel data at a given location
Copy	Copy frame buffer area to specified coordinates
Sfill	Fill an area with a given pixel color value
Pfill	Fill an area with a given pixel pattern
Bitmap	Fill a region using a bitmap image

Figure 5-44. The THINC protocol display commands.

5.8 Power Management
5.8.1 Hardware Issues



Device	Li et al. (1994)	Lorch and Smith (1998)
Display	68%	39%
CPU	12%	18%
Hard disk	20%	12%
Modem		6%
Sound		2%
Memory	0.5%	1%
Other		22%

Figure 5-45. Power consumption of various parts of a notebook computer.

5.8.2 The Display

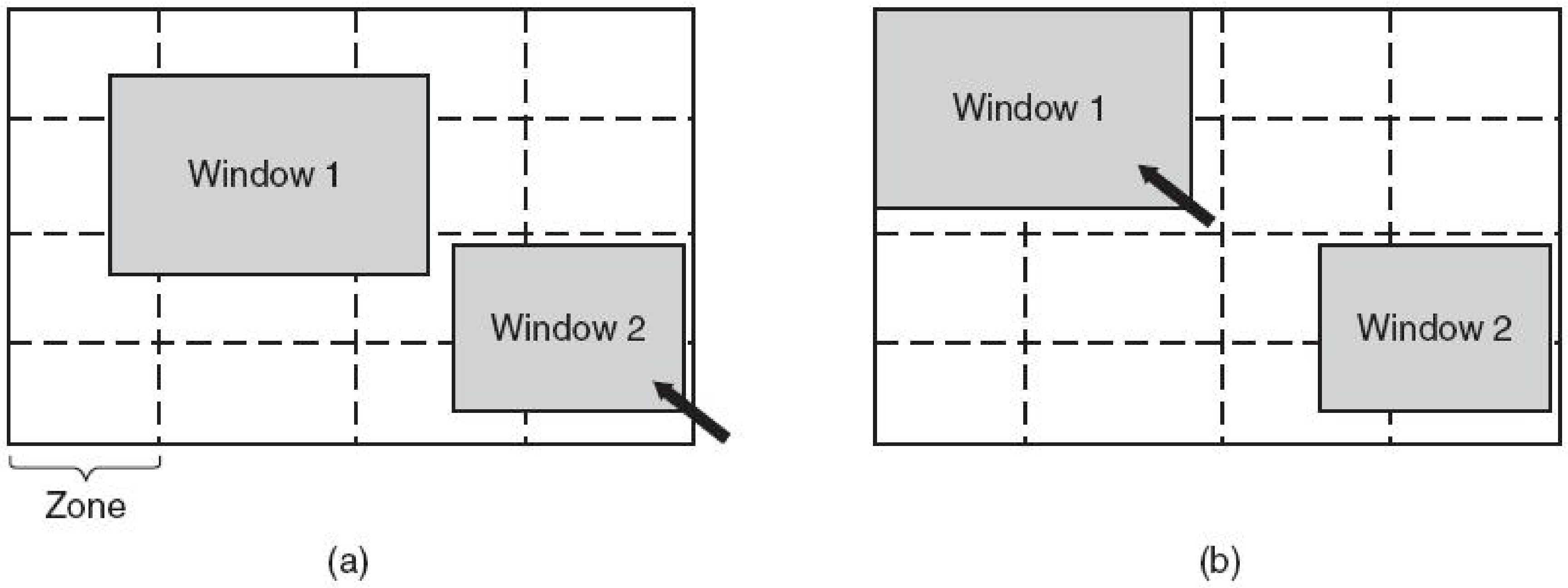
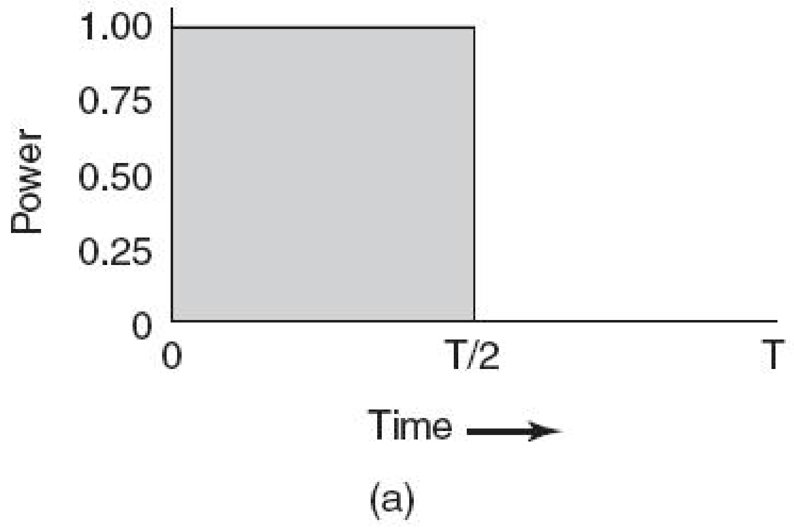


Figure 5-46. The use of zones for backlighting the display.
(a) When window 2 is selected it is not moved.
(b) When window 1 is selected, it moves to reduce the number of zones illuminated.

5.8.3 The CPU



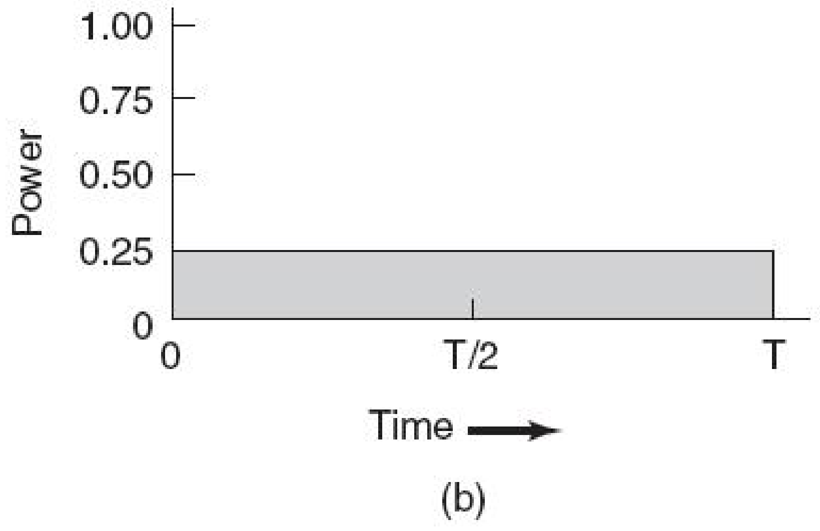


Figure 5-47. (a) Running at full clock speed. (b) Cutting voltage by two cuts clock speed by two and power consumption by four.

Reading Materials

- Disk Scheduling Algorithms
  - Geist and Daniel. A continuum of disk scheduling algorithms. ACM Trans. On Computer Systems. 1987, 7: 80-112
  - Alexander Thomasian. Survey and analysis of disk scheduling methods. ACM SIGARCH Computer Architecture News. 2011,39(2):8-25
  - David Boutcher. Does virtualization make disk scheduling passé? ACM SIGOPS Operating Systems Review 2010, 44(1):20-24
- Storage
  - Scheible. A survey of storage options. Computer. 2002,35:42-46
  - Avinash Lakshman. Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review . 2010 ,44(2):35-40
- Research on Power
  - Stan and Skadron. Power-Aware Computing.Computer, 2003,36:35-38
  - Parthasarathy Ranganathan. Recipe for efficiency: principles of power-aware computing. Communications of the ACM. 2010, 53(4): 60-67

谢 谢！