Chapter 5

Input/Output

sun@hit.edu.cn

INPUT/OUTPUT

- OS
  - control all the computer's I/O devices
  - issue commands to the devices, catch interrupts, and handle errors
  - provide an interface between the devices and the rest of the system that is simple and easy to use
    - the interface should be device independence

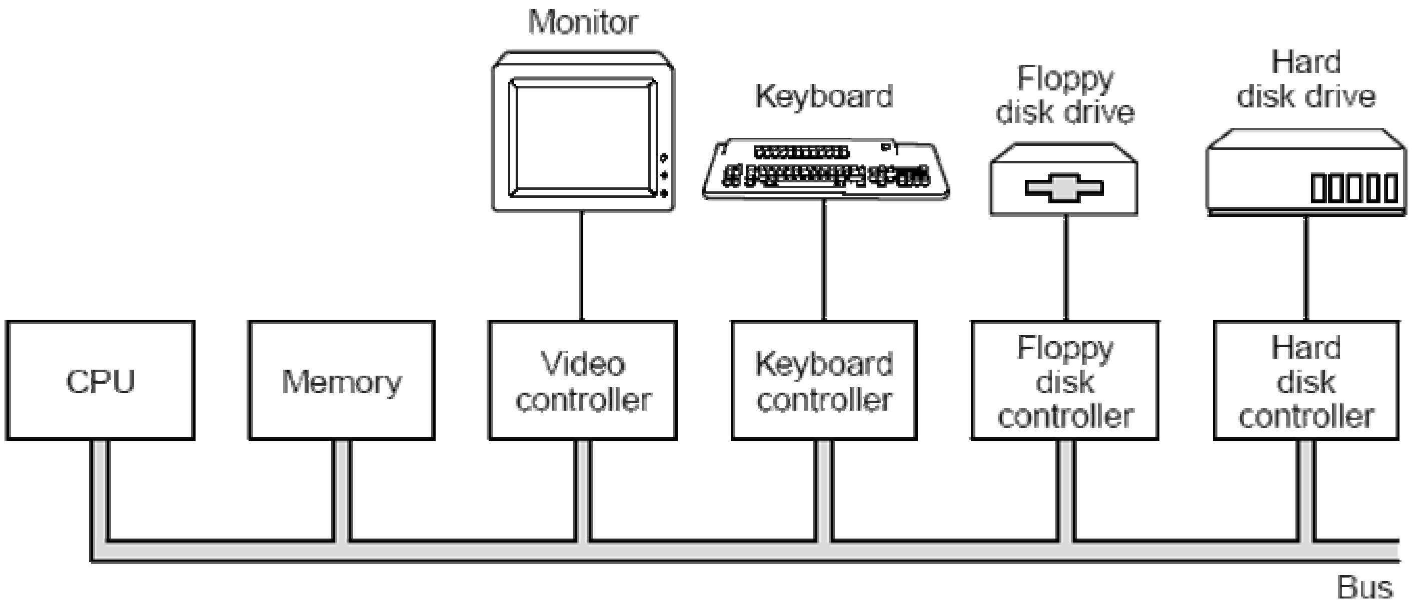
Principles of I/O Hardware

Some typical device, network, and bus data rates

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Device Controllers

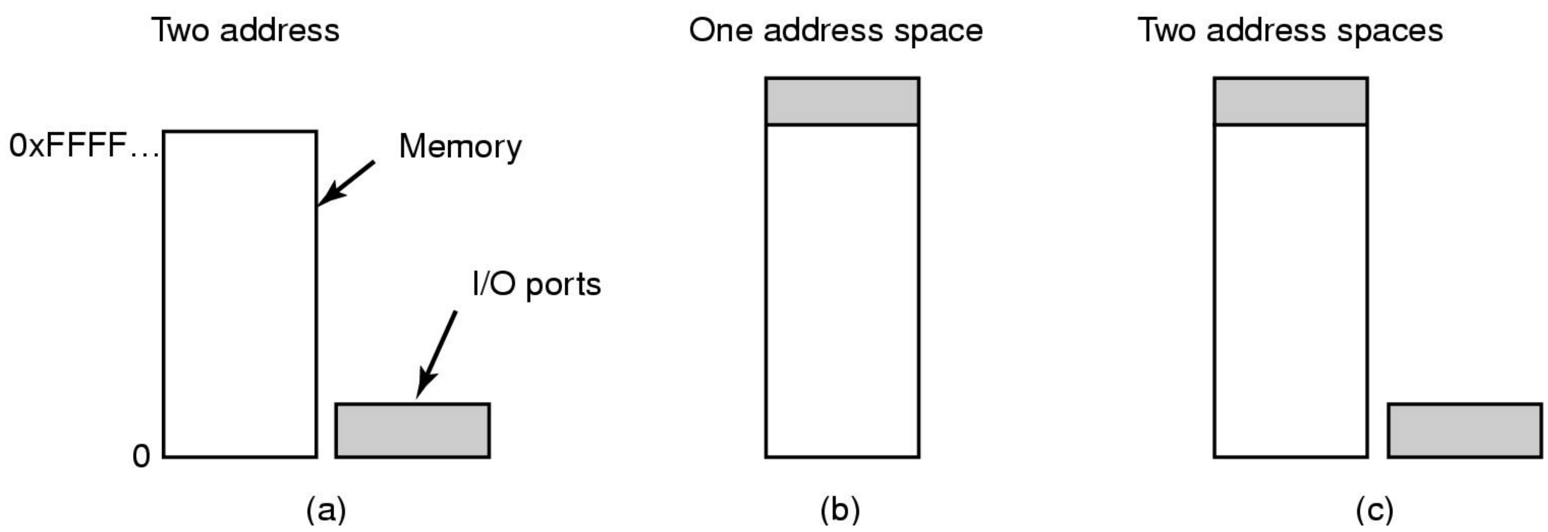
- I/O devices have components:
  - mechanical component
  - electronic component
- The electronic component is the device controller
  - may be able to handle multiple devices



Device Controller's Tasks

- Convert serial bit stream to block of bytes of character
- Perform error correction as necessary
  - checksum, parity and etc.
- Make data available to main memory
  - CPU reads and writes the controller's registers to know states and deliver commands
  - data buffer is used to transfer data

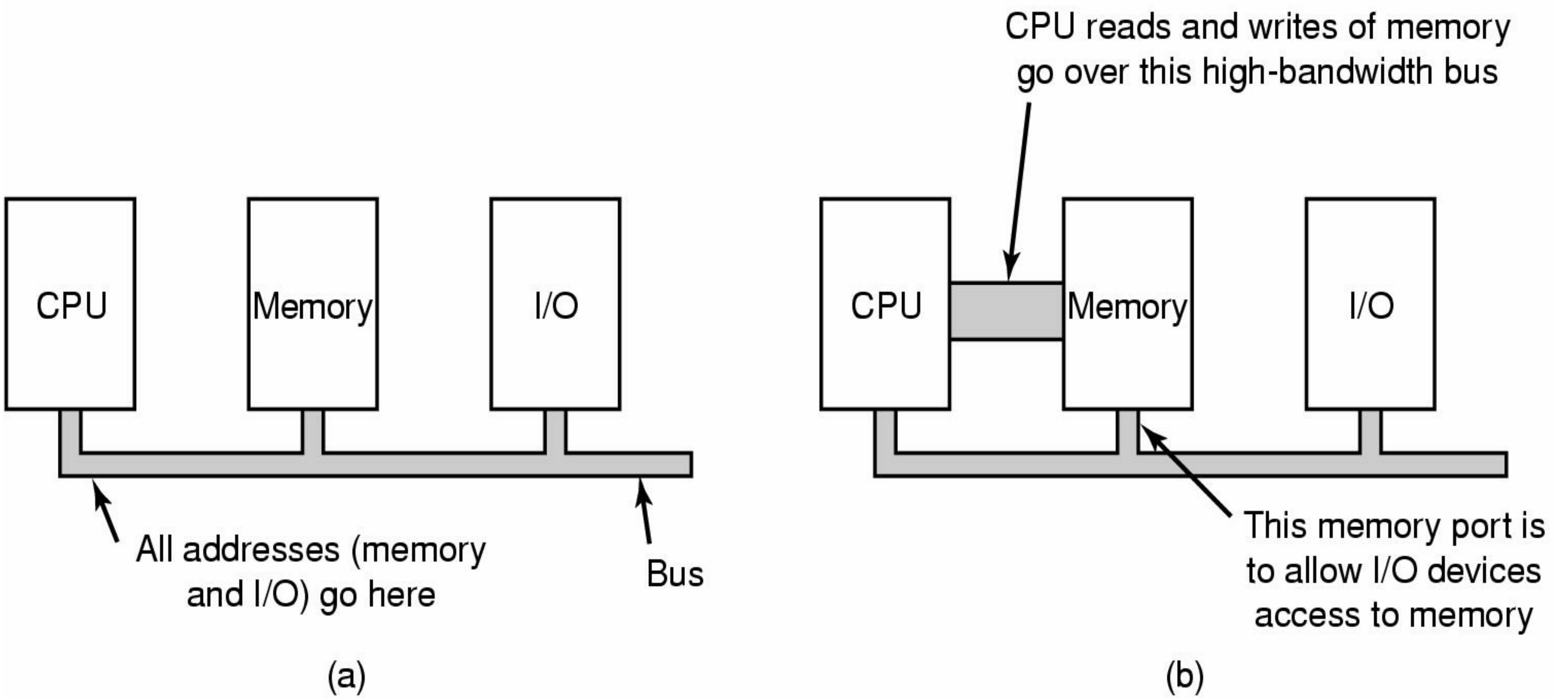
Memory-Mapped I/O (1)



- **Separate I/O and memory space**
- **Memory-mapped I/O**
- **Hybrid**

sun@hit.edu.cn

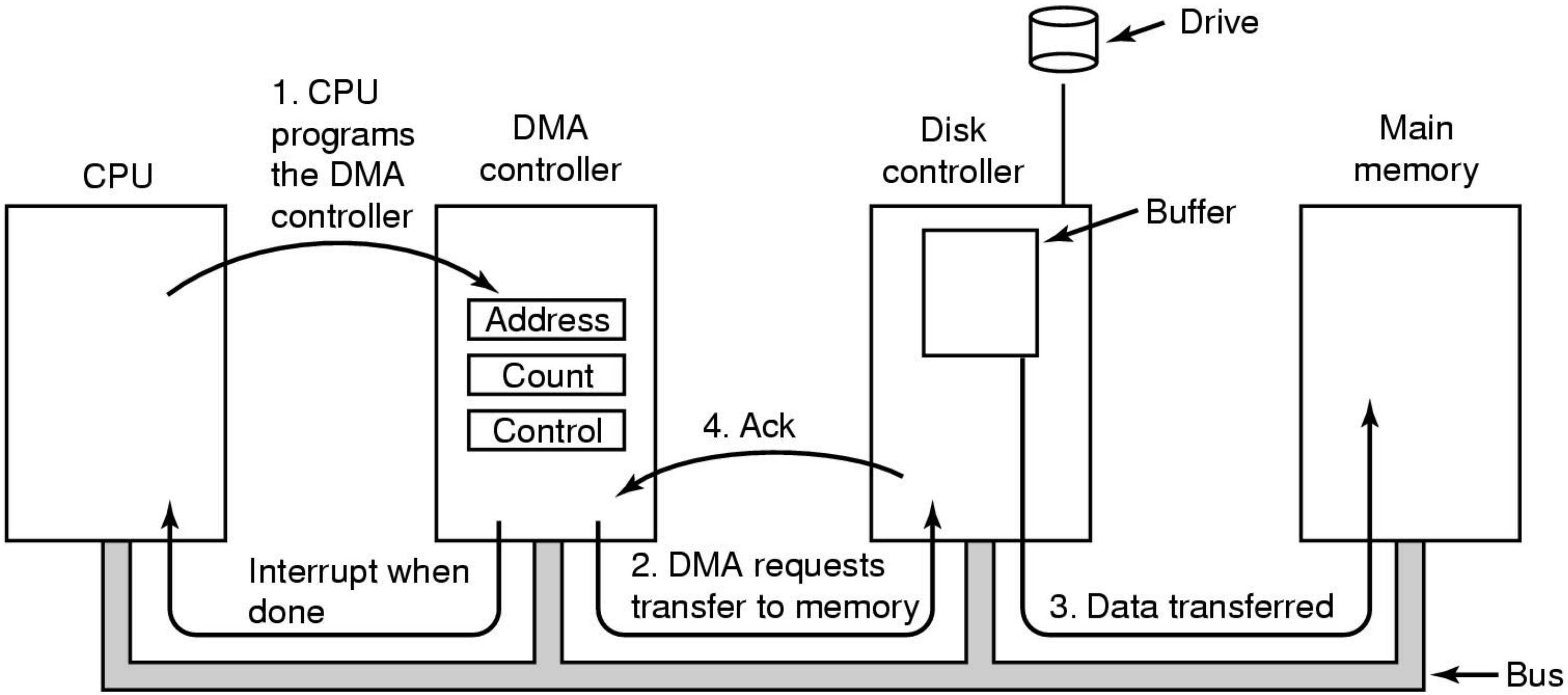
Memory-Mapped I/O (2)



(a) A single-bus architecture

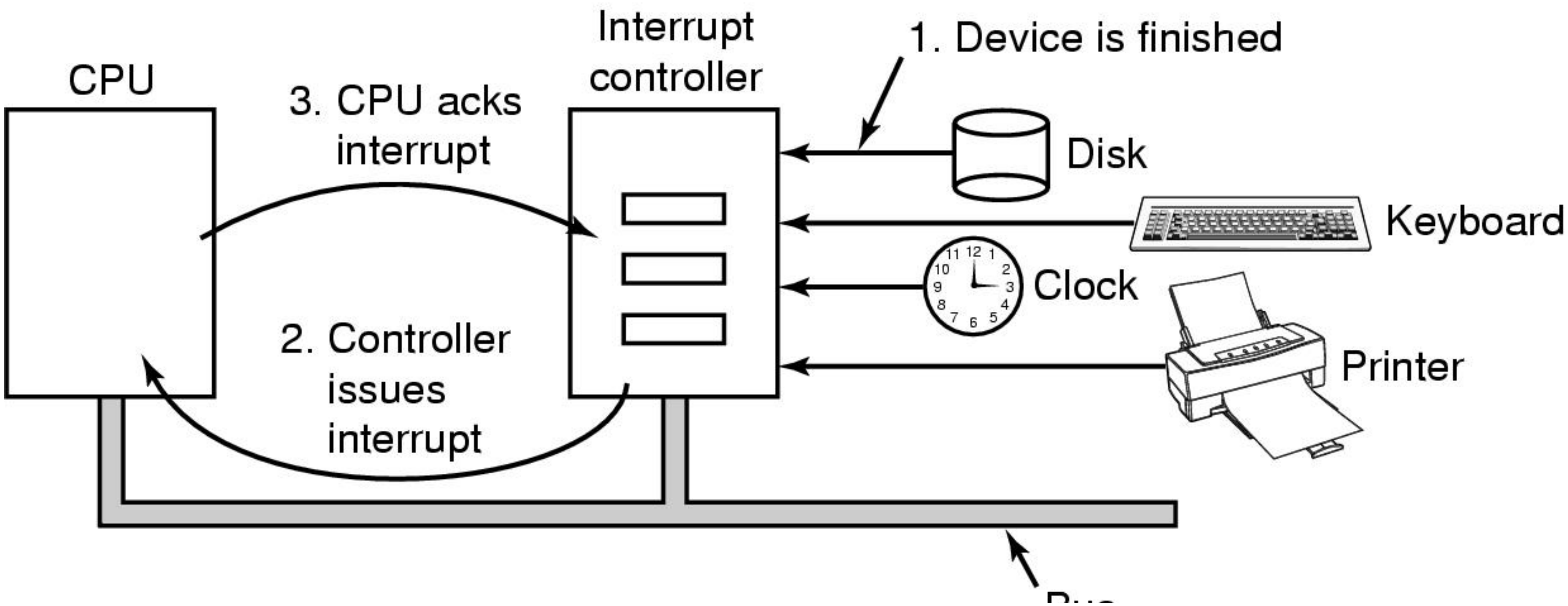
(b) A dual-bus memory architecture

Direct Memory Access (DMA)



Operation of a DMA transfer

Interrupts Revisited



How interrupts happens. Connections between devices and interrupt controller actually use interrupt lines on the bus rather than dedicated wires

Interrupts Revisited

- Check if an interrupt is pending after each instruction's execution
  - store program counter and PSW, then jump to interrupt vector
- A pipeline or superscalar CPU's PC doesn't mean the next instruction to execute but to fetch

Interrupts Revisited

- Precise interrupt
  - The PC 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
  - The execution state of the instruction pointed to by the PC is known
- Imprecise interrupt

Principles of I/O Software

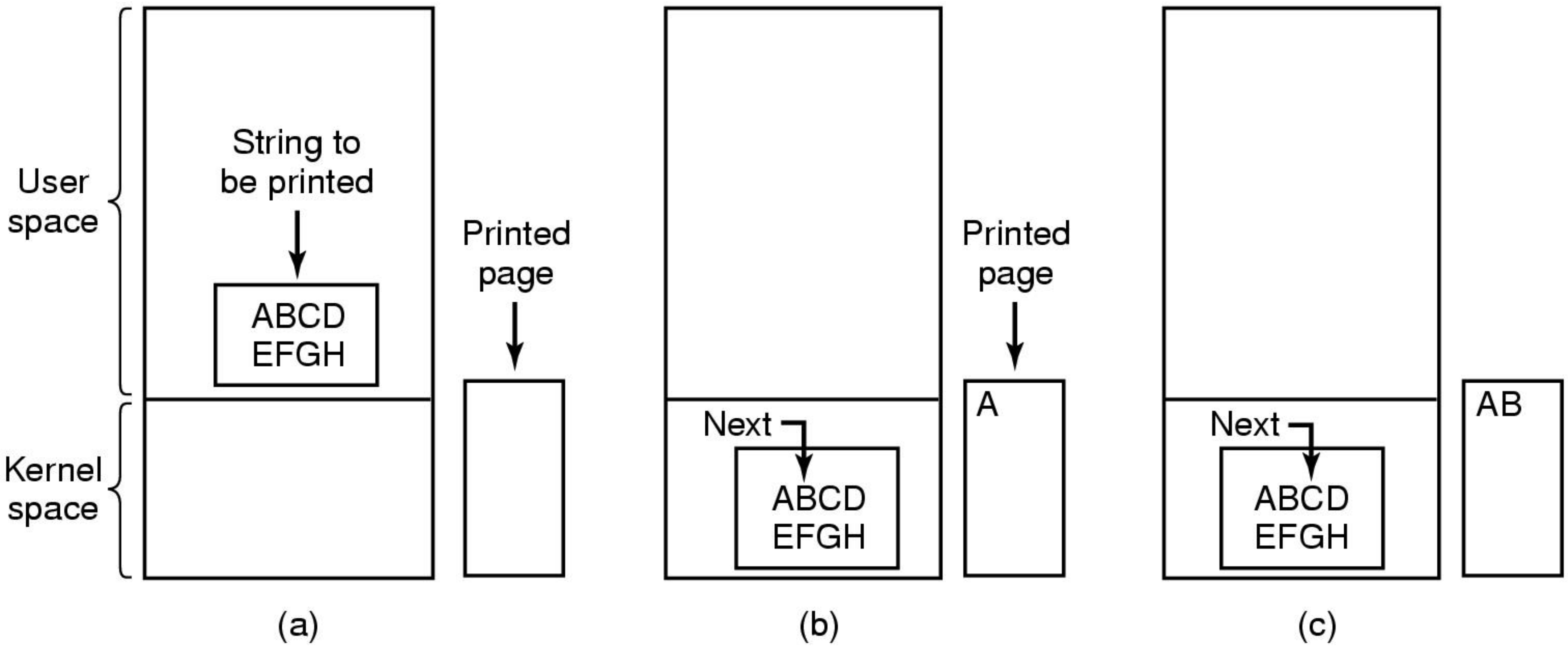
Goals of I/O Software (1)

- Device independence
  - programs can access any I/O device without specifying device in advance
    - (floppy, hard drive, or CD-ROM)
- Uniform naming
  - name of a file or device is a string or an integer
  - not depending on which device
- Error handling
  - handle as close to the hardware as possible
    - controller, driver, application

Goals of I/O Software (2)

- Synchronous vs. asynchronous transfers
  - blocked transfers vs. interrupt-driven
- Buffering
  - data coming off a device cannot be stored in final destination
- Sharable vs. dedicated devices
  - disks are sharable
  - tape drives would not be

Programmed I/O (1)



Steps in printing a string

Programmed I/O (2)

```
copy_from_user(buffer, p, count);               /* p is the kernel bufer */
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( );
```

Writing a string to the printer using programmed I/O

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
  - Code executed when print system call is made
  - Interrupt service procedure

I/O Using DMA

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

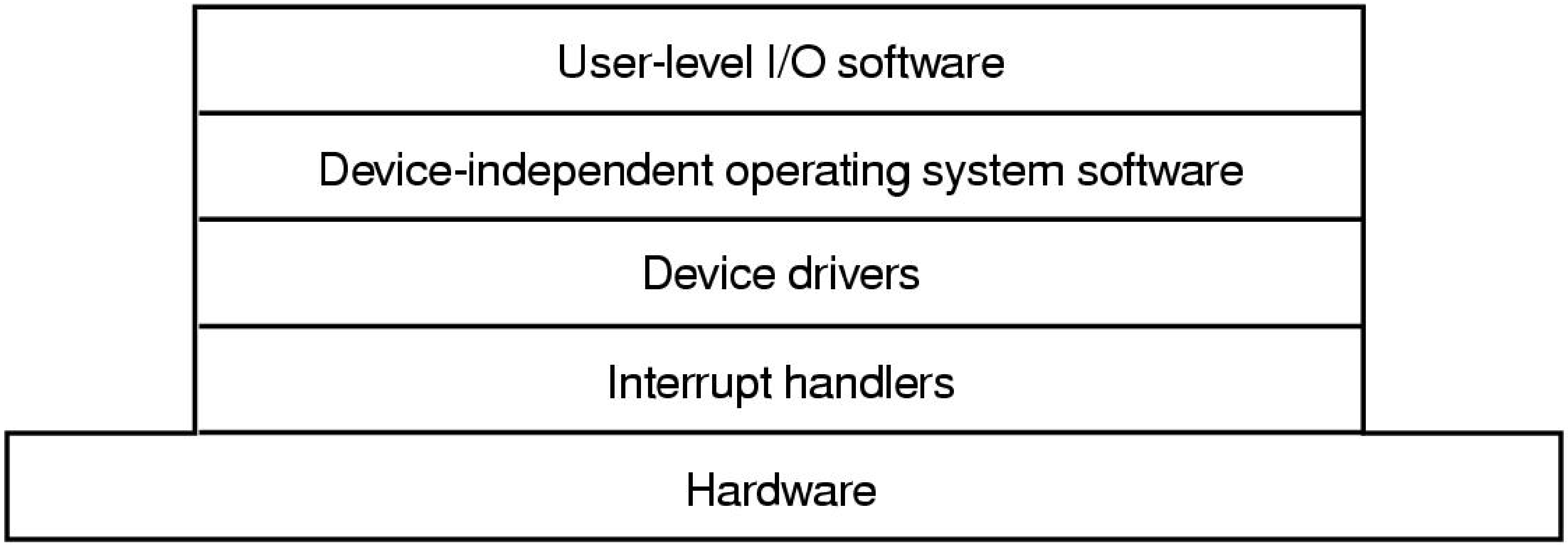
(a)

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

(b)

- **Printing a string using DMA**
  - **code executed when the print system call is made**
  - **interrupt service procedure**

I/O Software Layers



Layers of the I/O Software System

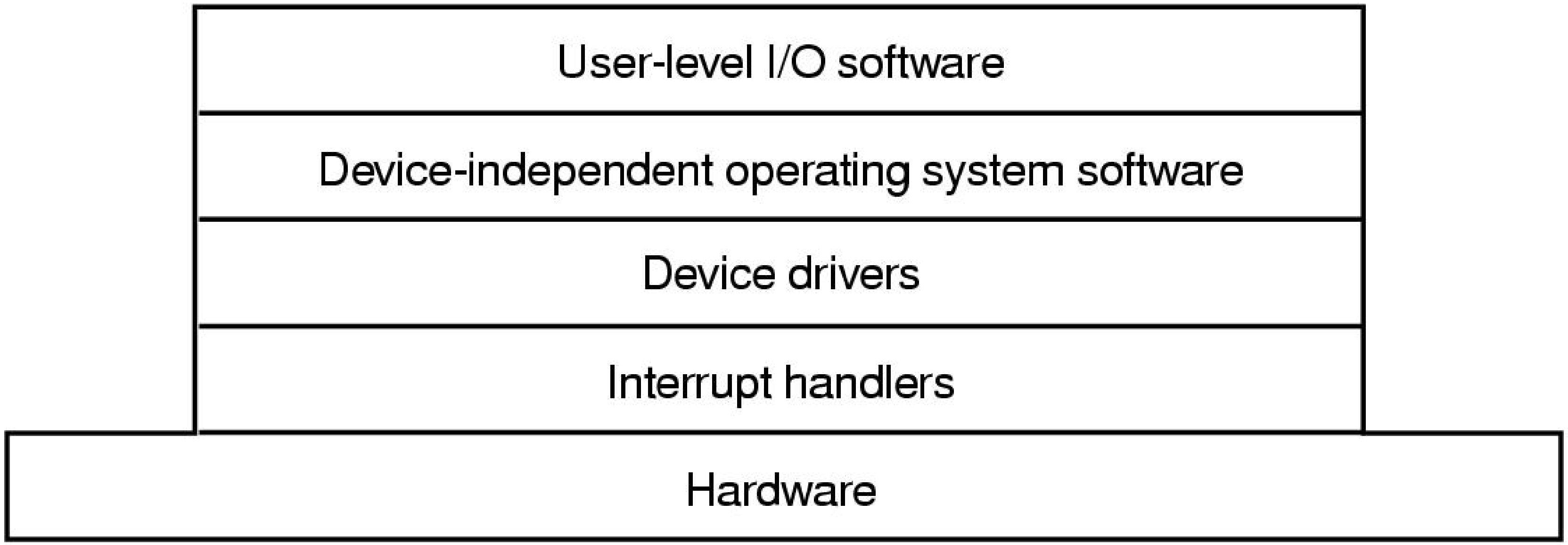
Interrupt Handlers (1)

- Interrupt handlers are best hidden
  - block the driver starting an I/O operation until interrupt notifies of completion
- Interrupt procedure does its task
  - then unblocks driver that started it
- Steps must be performed in software after interrupt completed

Interrupt Handlers (2)

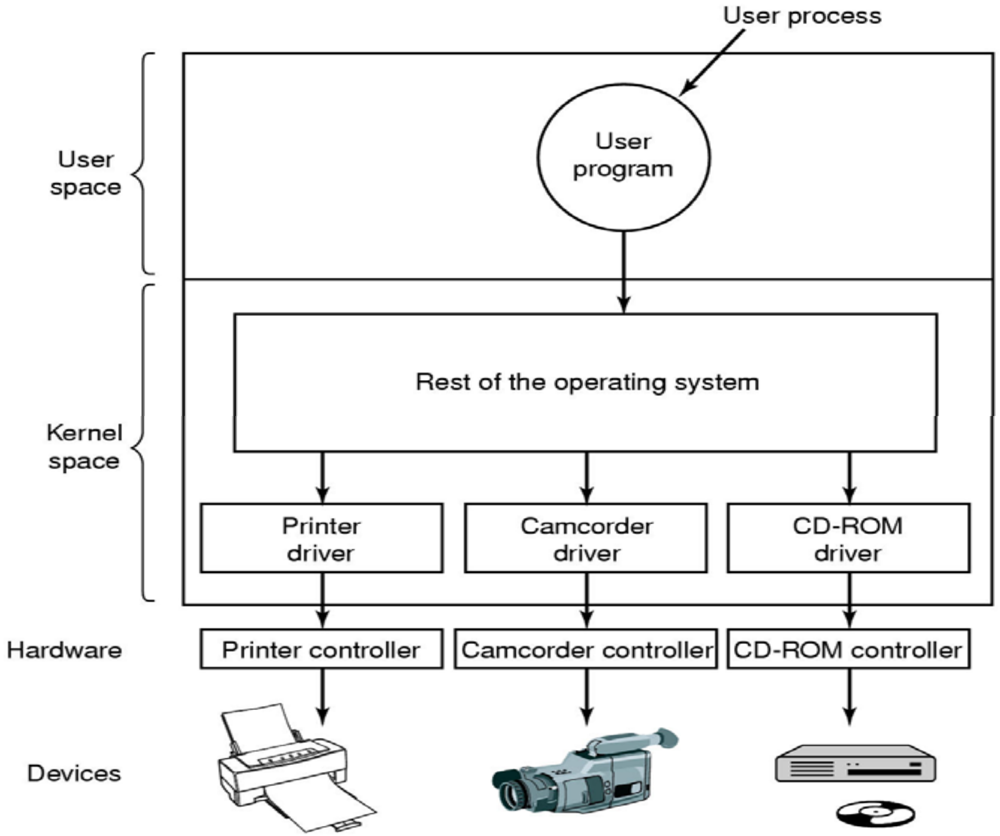
1. Save registers not already saved by interrupt hardware
2. Set up context for interrupt service procedure
3. Set up stack for interrupt service procedure
4. Ack interrupt controller, reenable interrupts
5. Copy registers from where they saved to process table
6. Run service procedure
7. Set up MMU context for process to run next
8. Load new process' registers
9. Start running the new process

I/O Software Layers



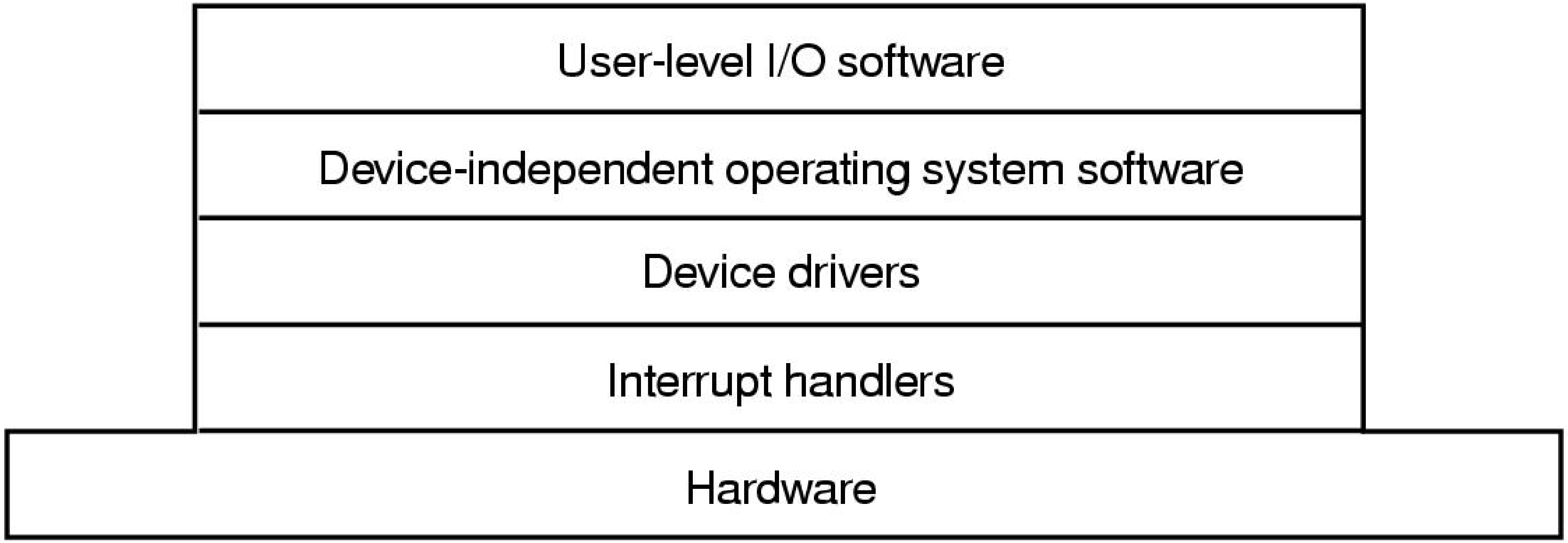
Layers of the I/O Software System

Device Drivers



- **Logical position of device drivers is shown here**
- **Communications between drivers and device controllers goes over the bus**

I/O Software Layers



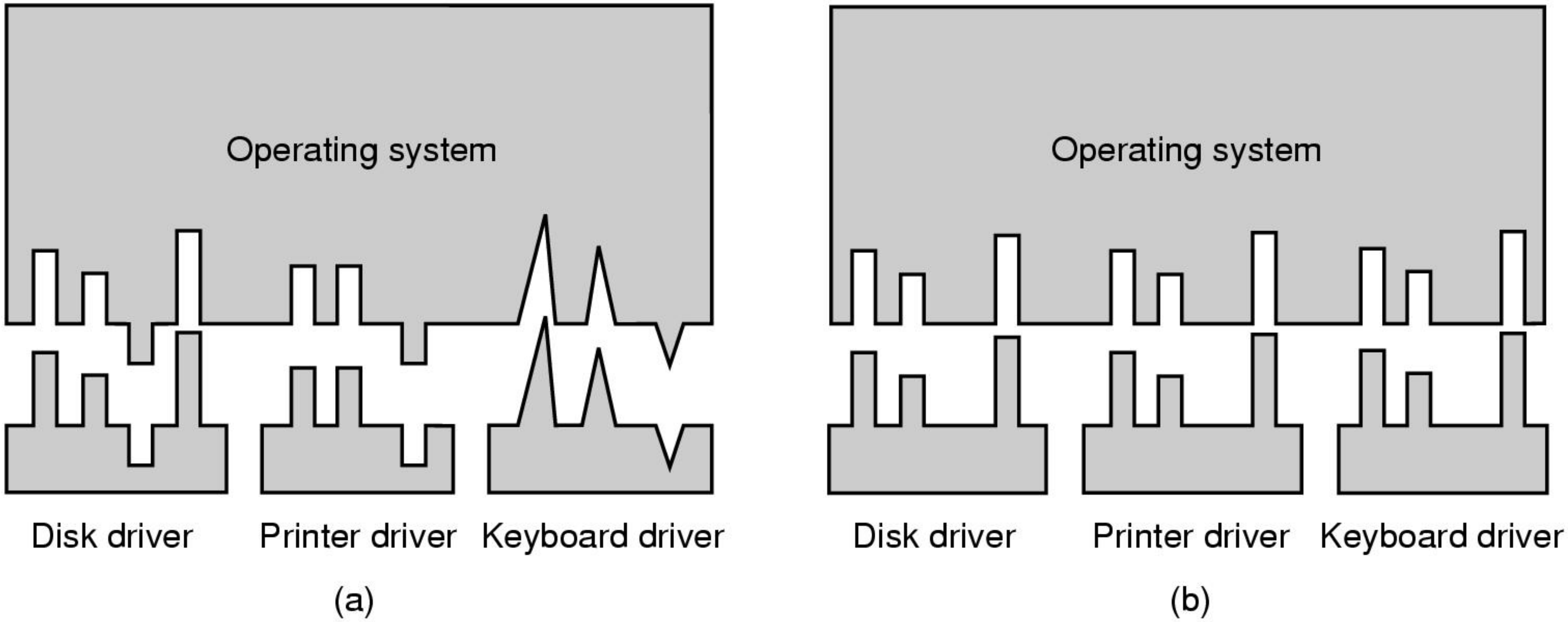
Layers of the I/O Software System

Device-Independent I/O Software

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

Functions of the device-independent I/O software

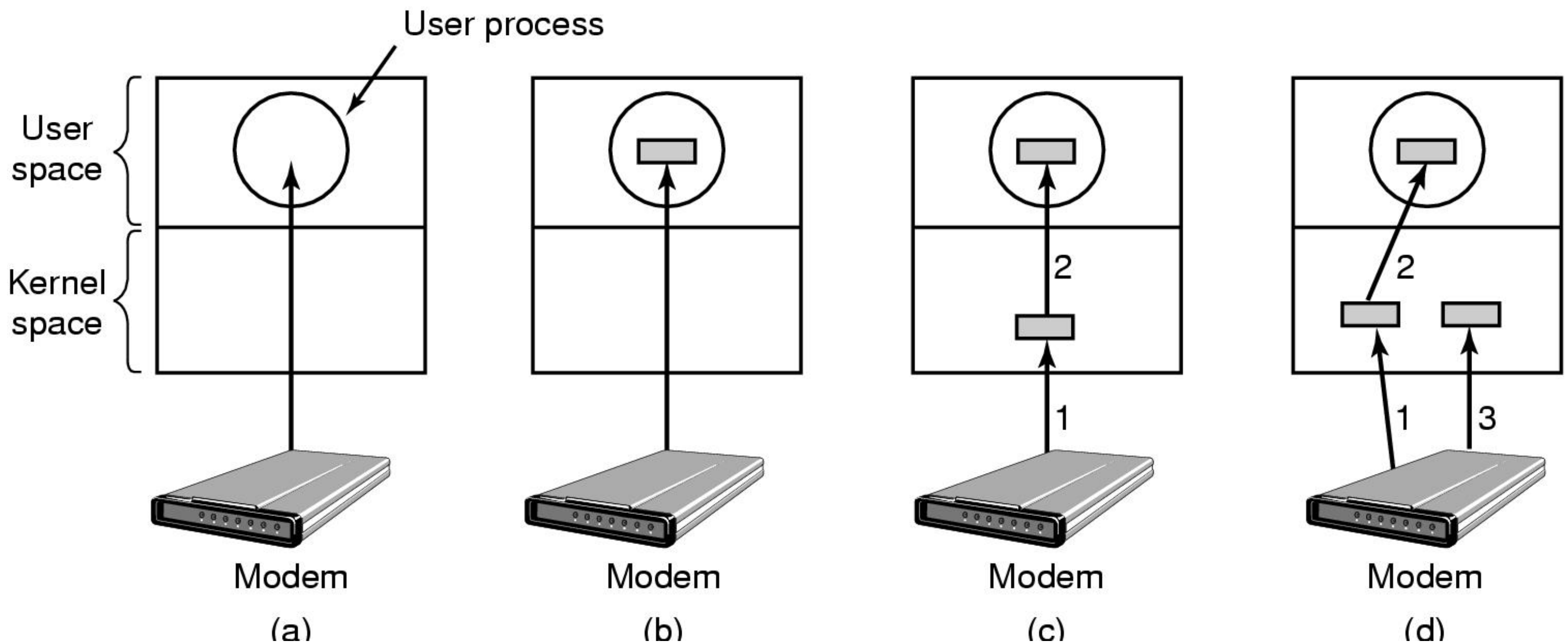
Uniform Interfacing for Device Drivers



(a) Without a standard driver interface

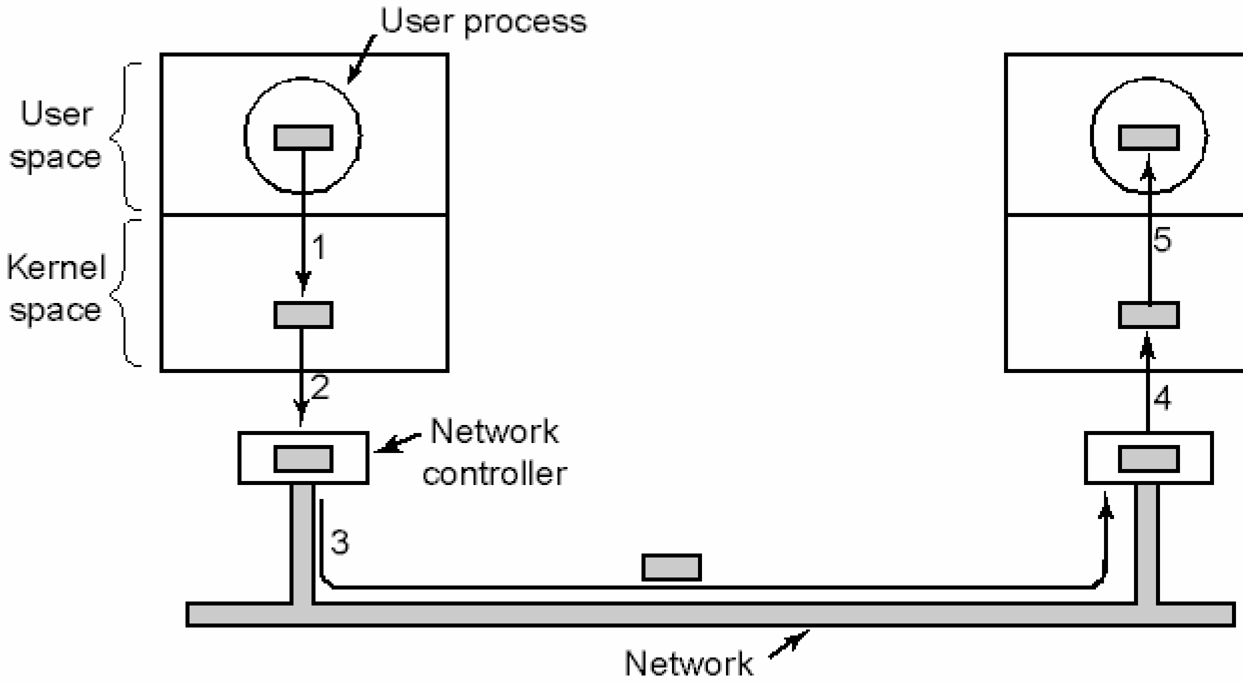
(b) With a standard driver interface

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

Buffering



Networking may involve many copies, performance suffers

Error Reporting

- Errors are far more common in I/O than others
  - Programming errors
    - ask for something impossible
    - invalid device, buffer address or other parameter
  - I/O errors
- When error occurs
  - retry, ignore or quit
  - have the system call fail with an error code
  - terminate the system (blue screen)

Other Functions

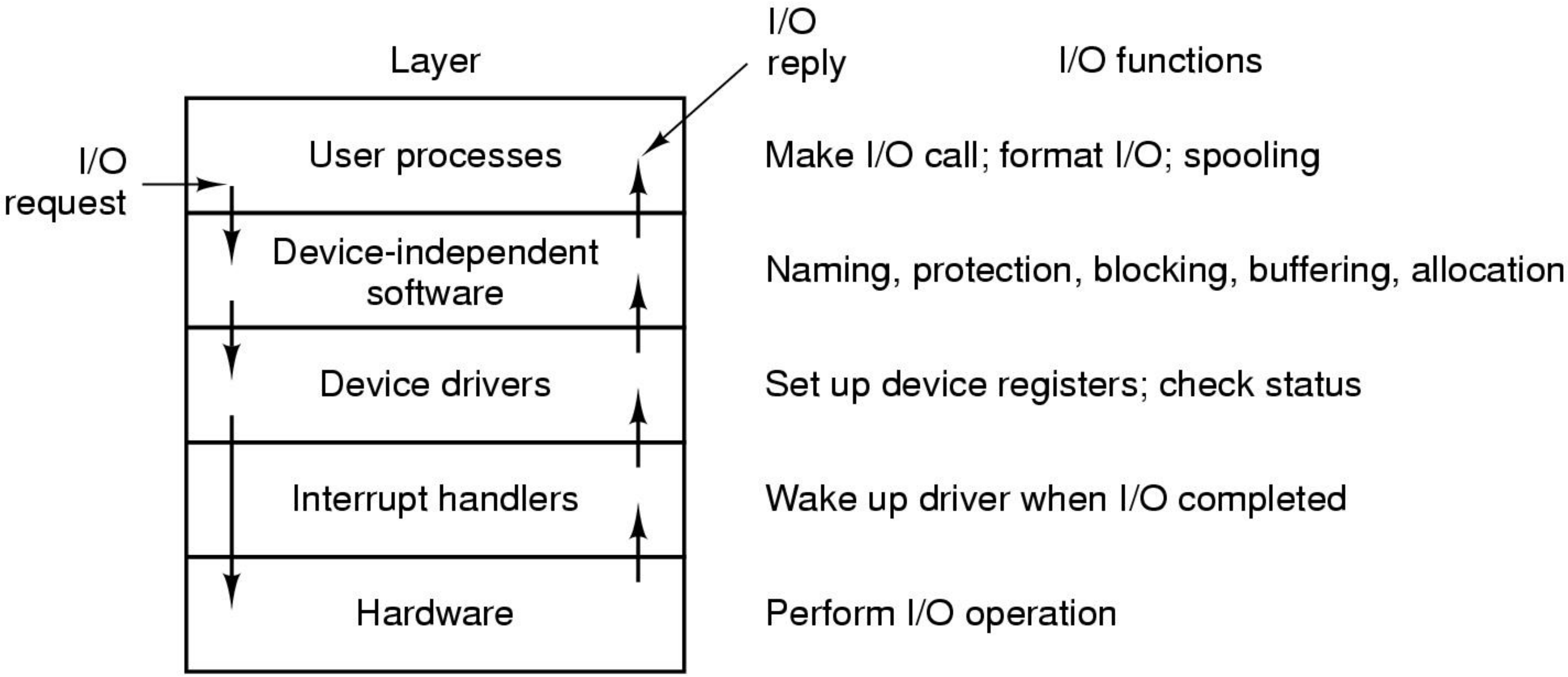
- Allocating and releasing dedicated devices
  - open, close
  - queue
- Device-independent block size
  - different disks may have different sector sizes
  - provide a uniform block size to higher layers

User-Space I/O Software

- **Link the I/O system call with the program**
  - `count = write(fd, buffer, nbytes);`
  - `printf("i = %d", i);`

- **Spooling to deal with dedicated devices**
  - **create a special directory—spooling directory**
  - **create a special process—daemon, which is the only process having permission to use printer**
  - **put the files to be printed in the spooling directory**

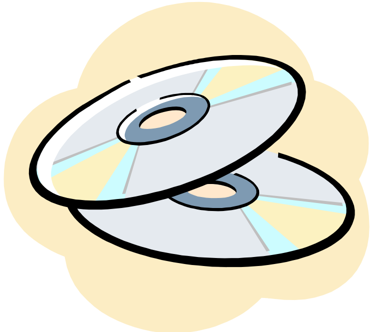
User-Space I/O Software

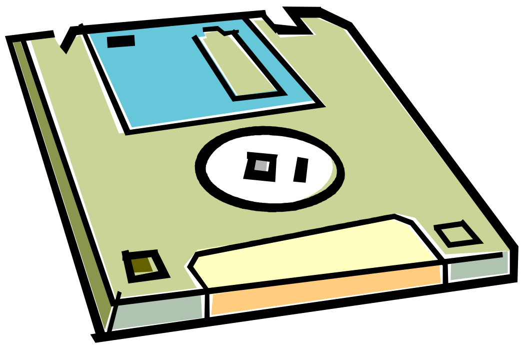


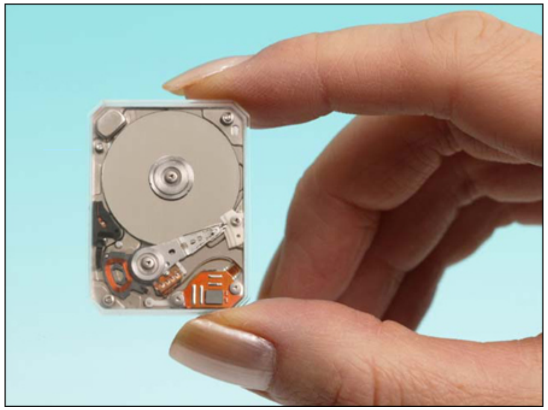
Layers of the I/O system and the main functions of each layer

Disks

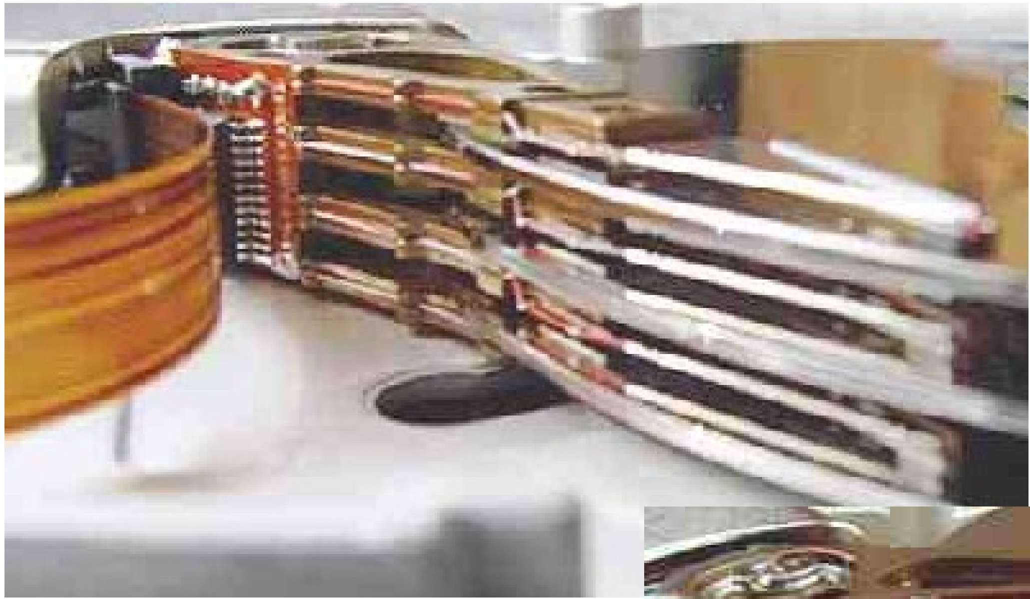
- Magnetic disks
- RAID
- CD-ROMs
  - CD-R, CD-RW, DVD





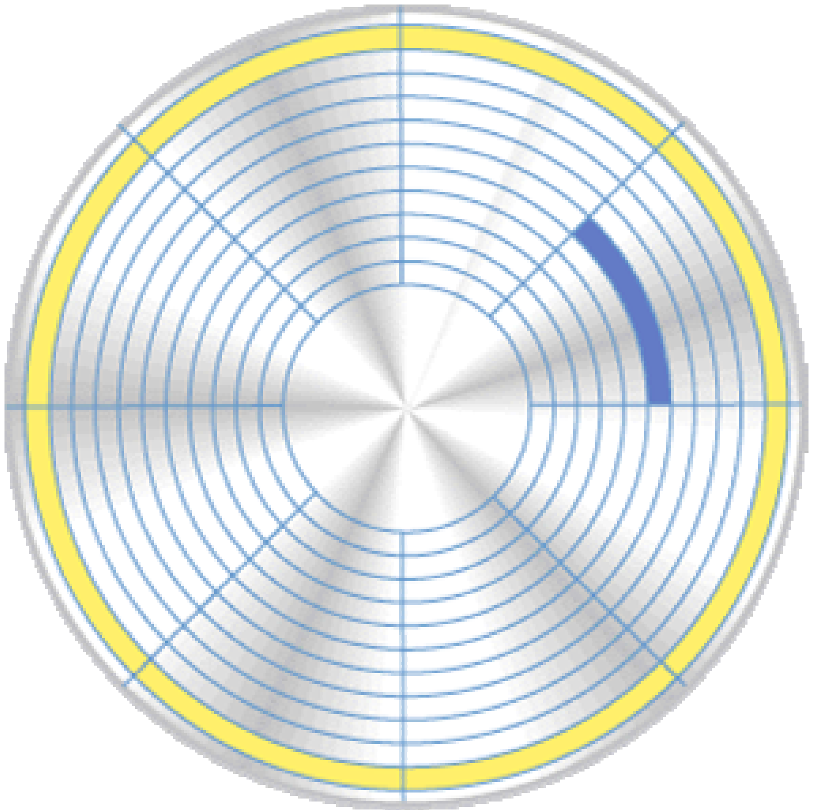


Hard Disks



Hard Disks

- Cylinders (x)
- Tracks (y)
- Sectors (z)
  - usually 512 Bytes
- Total capacity=?

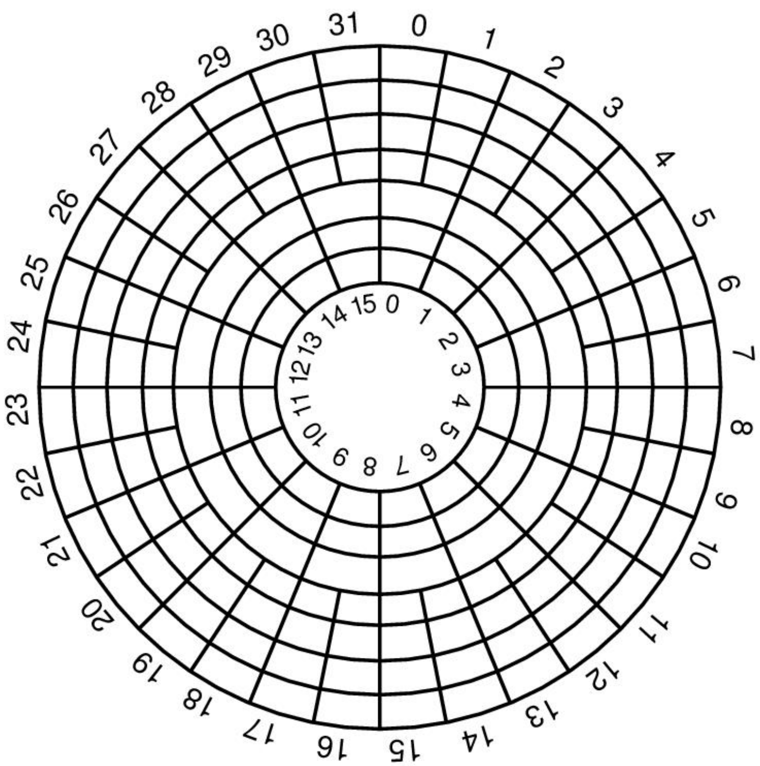


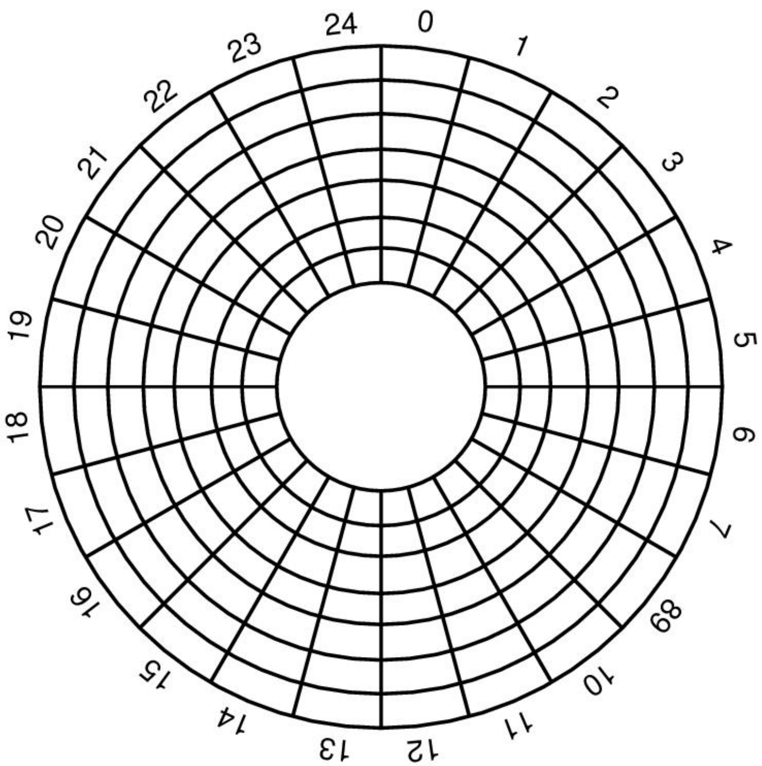
Magnetic Disks

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

Disk parameters for the original IBM PC floppy disk and a Western Digital WD 18300 hard disk

Geometry of a Disk



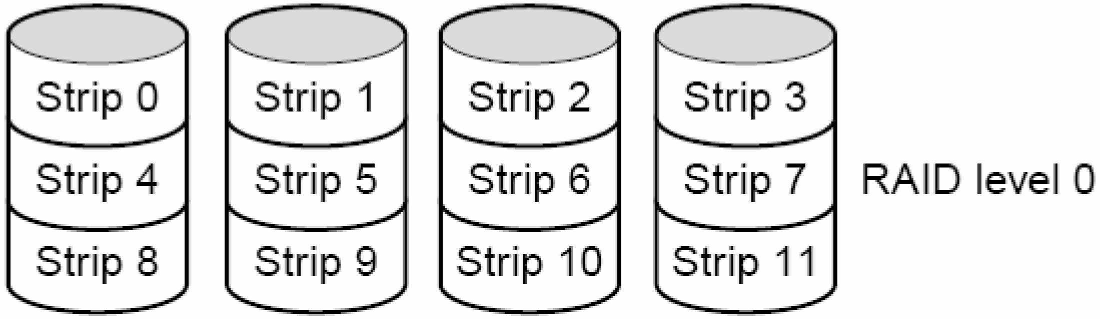


- **Physical geometry of a disk with two zones**
- **A possible virtual geometry for this disk**

RAID

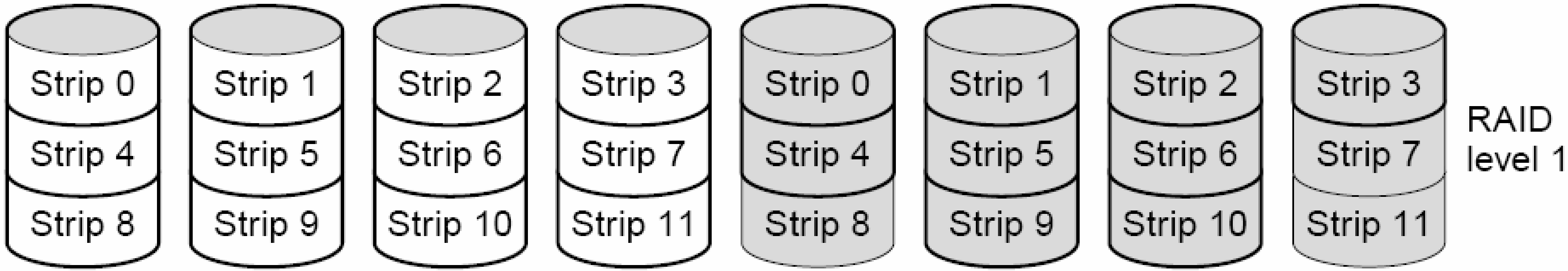
- Parallel I/O might be a good idea to improve performance
- RAID
  - Redundant Array of Inexpensive Disks
  - Redundant Array of Independent Disks
- SLED
  - Single Large Expensive Disk
- Make a RAID to be looked like a SLED to the OS

RAID 0



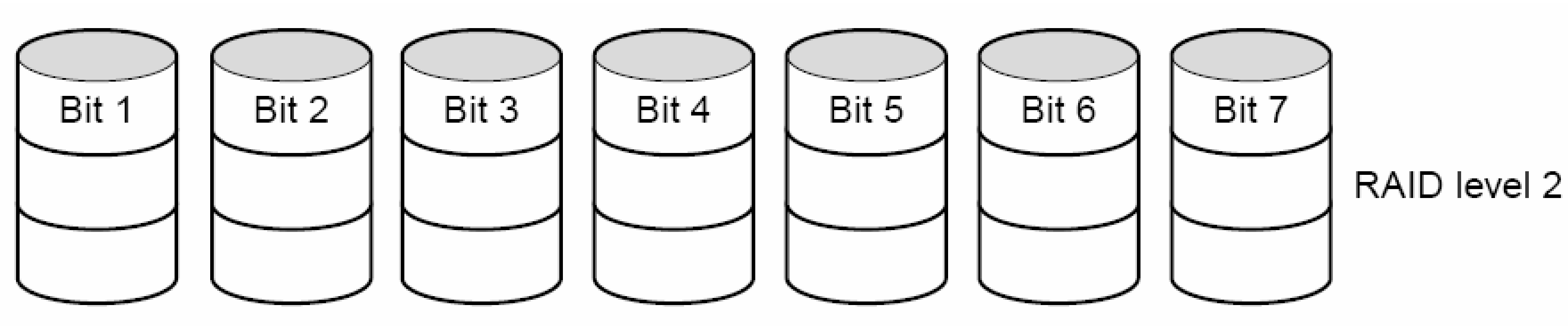
- Every strip has *k* sectors
- Works best with large requests
  - worst with asking for one sector at a time
- Reliability is worse than a SLED
- Uses all space, no redundancy

RAID 1



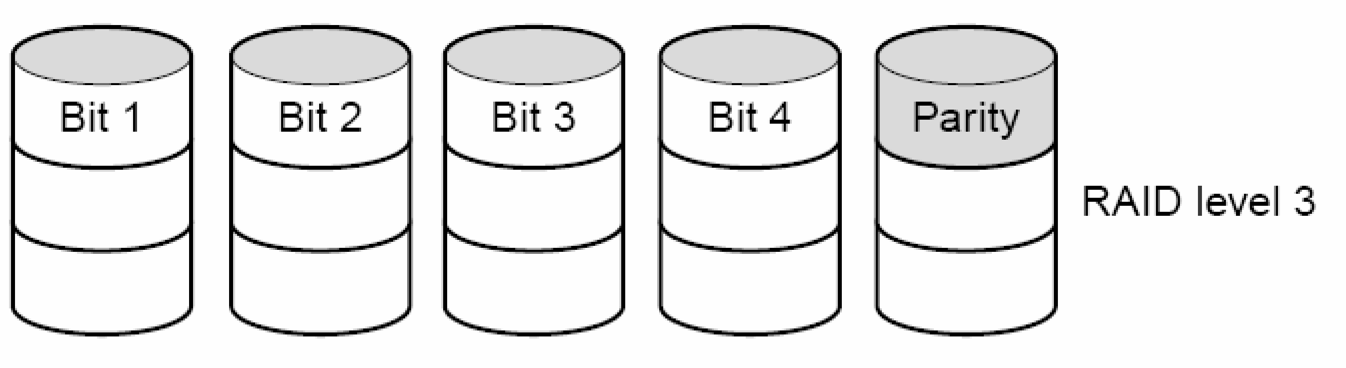
- There are the same number of backup disks as the primary disks
- Write performance is no better, but read performance is up to twice
- Fault tolerance is excellent

RAID 2



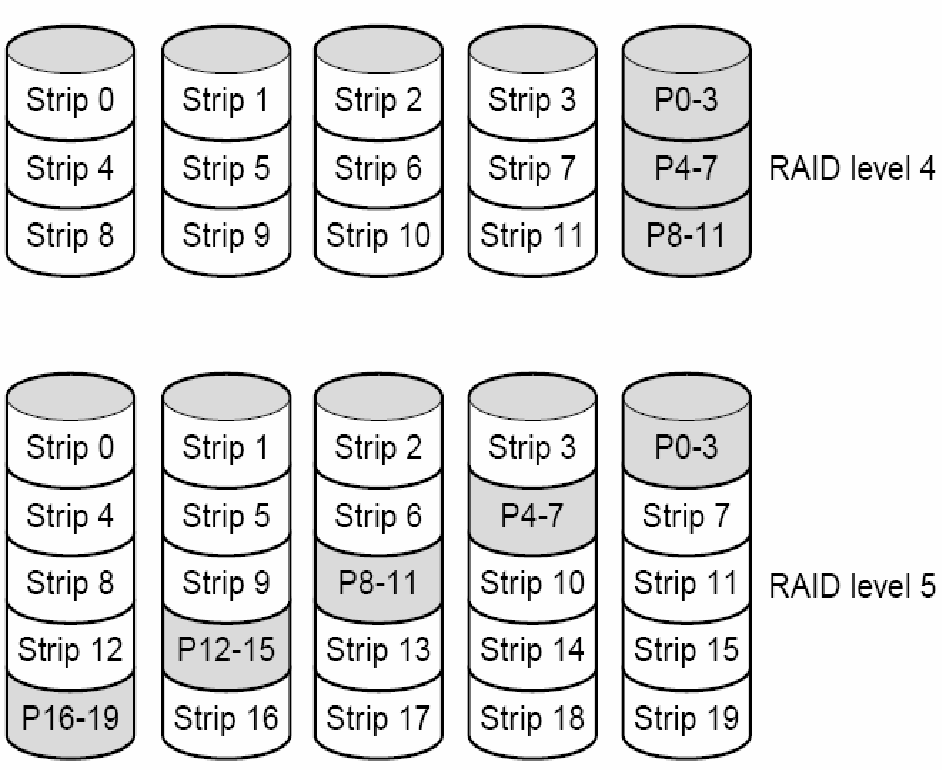
- Splits each byte into a pair of 4-bit nibbles
- Adds a Hamming code to each one to form a 7-bit word
- One bit per drive
- Also can 32-bit word with 6 parity bits to form a 38-bit Hamming word
- Losing one drive did not cause problems

RAID 3



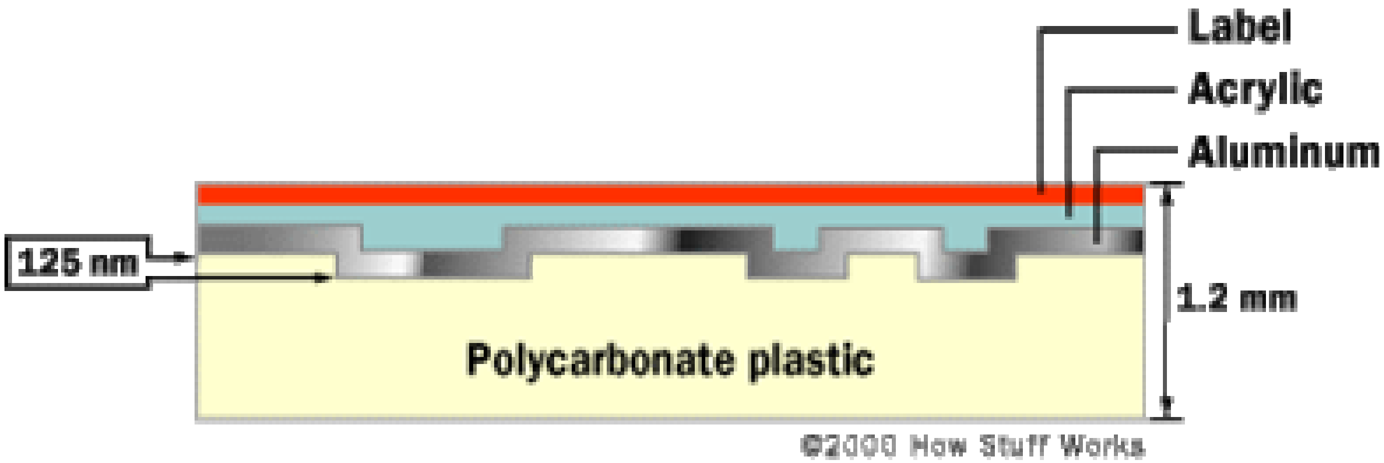
- **A simplified version of RAID level 2**
- **A single parity bit is computed for each data word and written to a parity drive**
- **One drive crashing can be recovered**

RAID 4 & 5

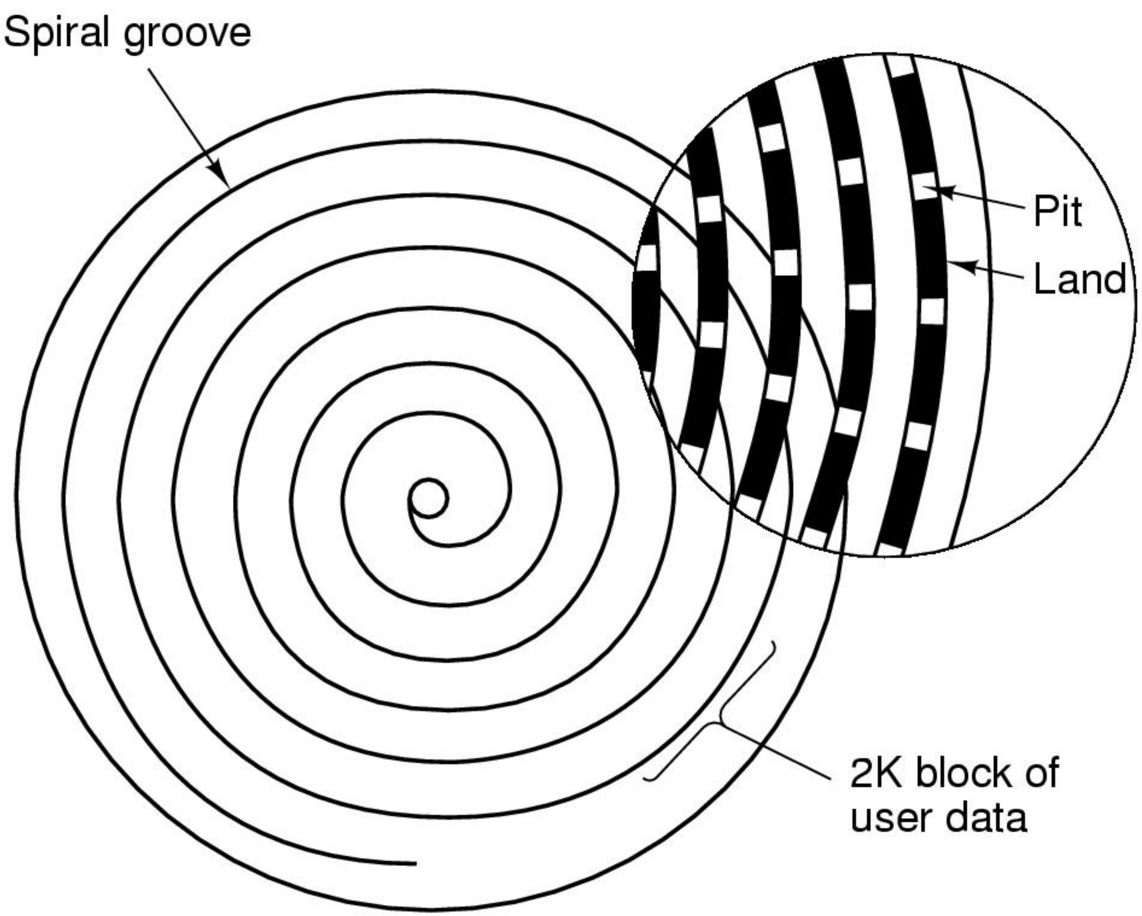


- All the strips are EXCLUSIVE ORed together to a parity strip
- Write performance is bad
- The parity drive in RAID 4 is bottleneck
- RAID 5 distributes the parity bits

CD-ROM

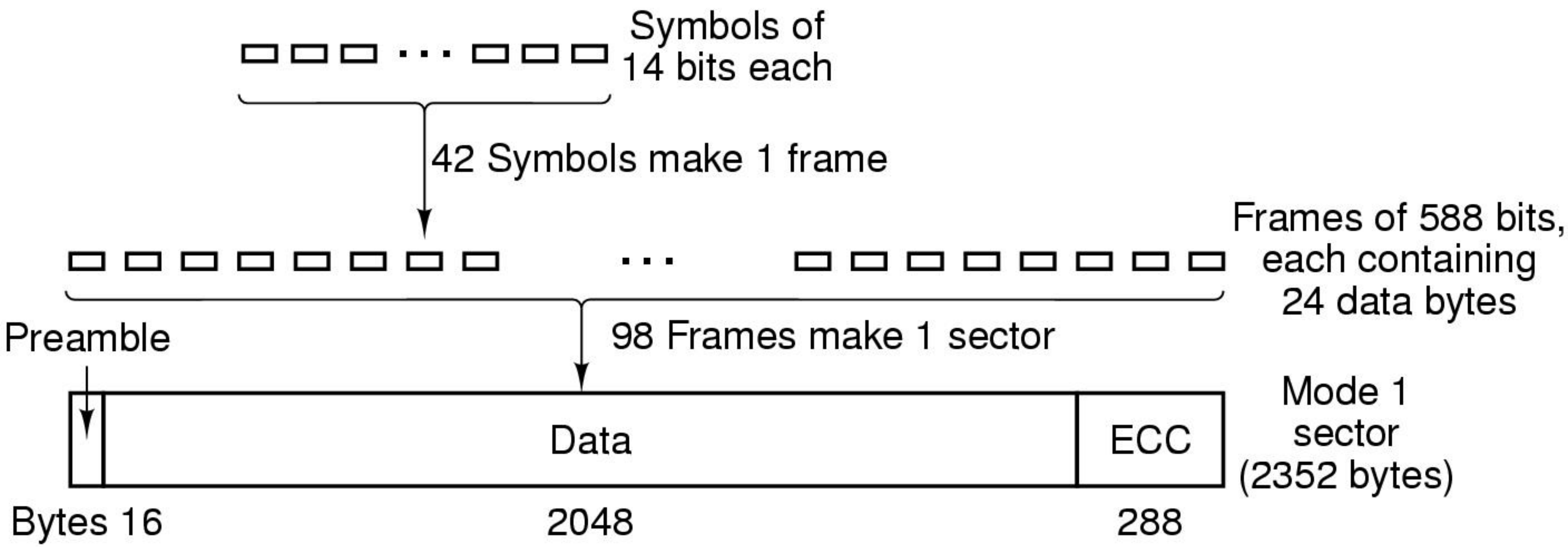

Label
Acrylic
Aluminum
125 nm
1.2 mm
Polycarbonate plastic
©2000 How Stuff Works

CD-ROM



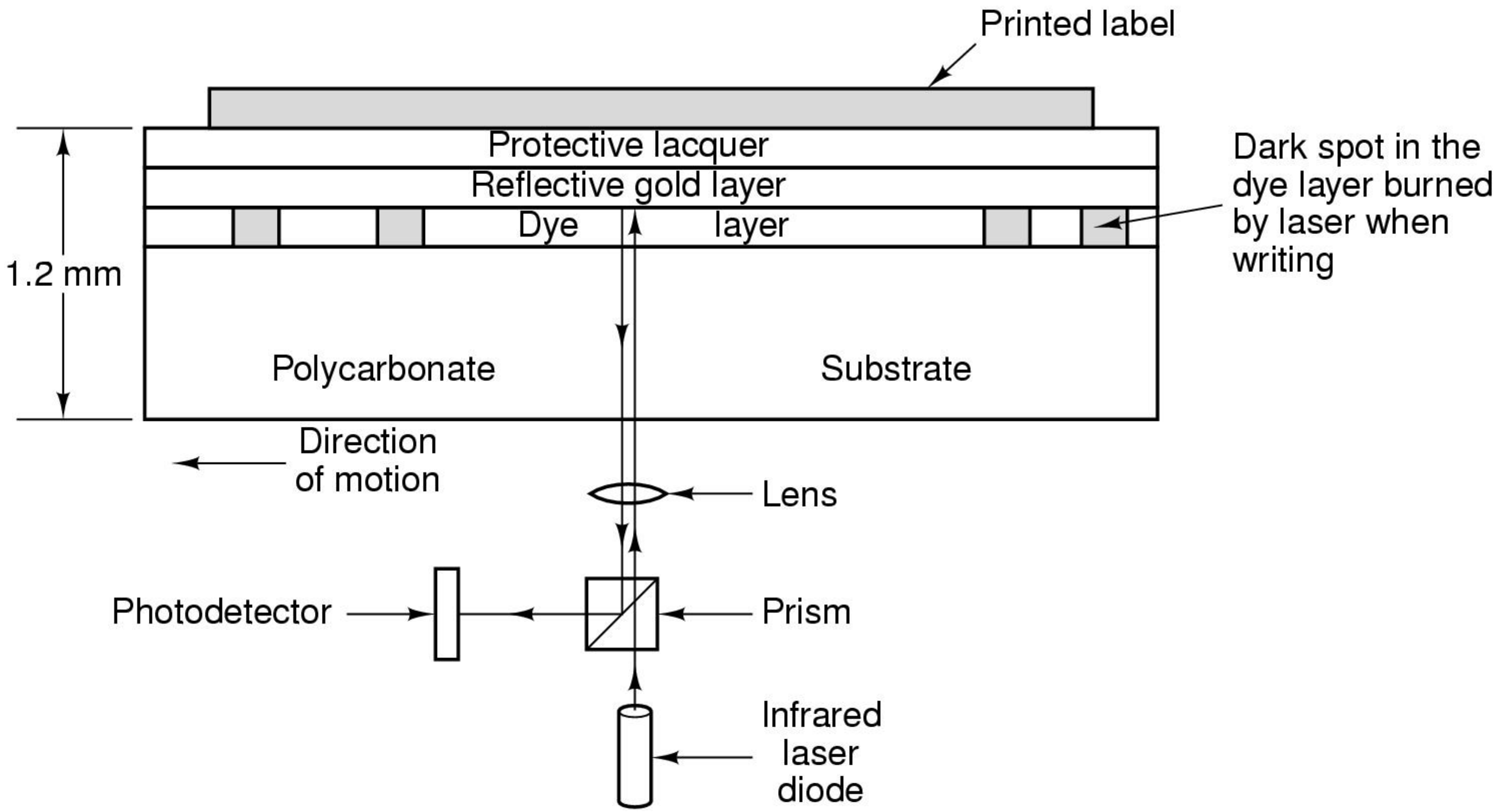
Recording structure of a CD or CD-ROM

CD-ROM

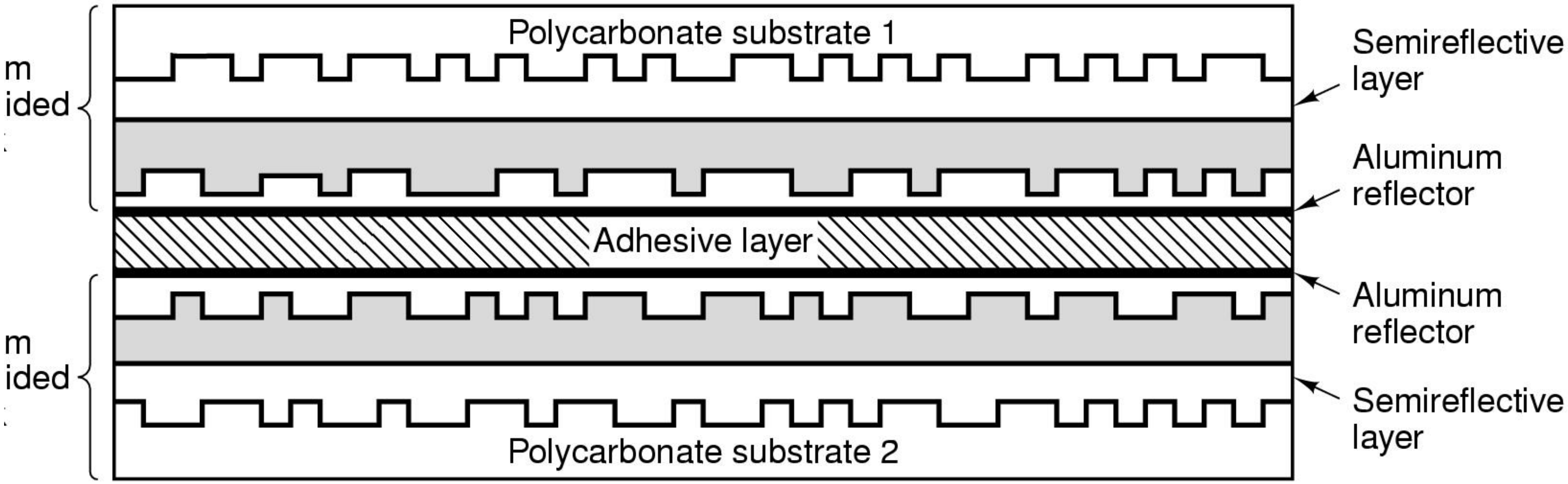


Logical data layout on a CD-ROM

CD-R


Printed label
Protective lacquer
Reflective gold layer
Dye layer
Dark spot in the dye layer burned by laser when writing
1.2 mm
Polycarbonate
Substrate
Direction of motion
Lens
Photodetector
Prism
Infrared laser diode

DVD

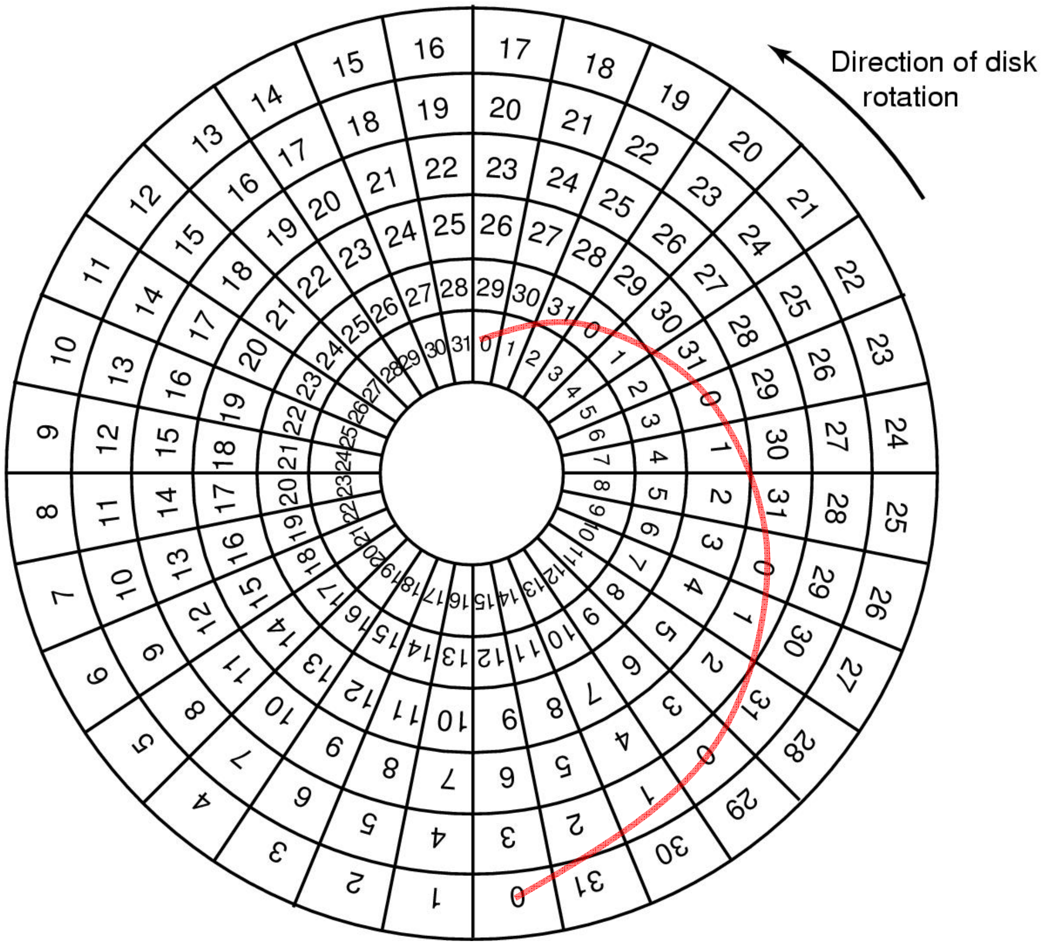


A double sided, dual layer DVD disk

Disk Formatting (1)

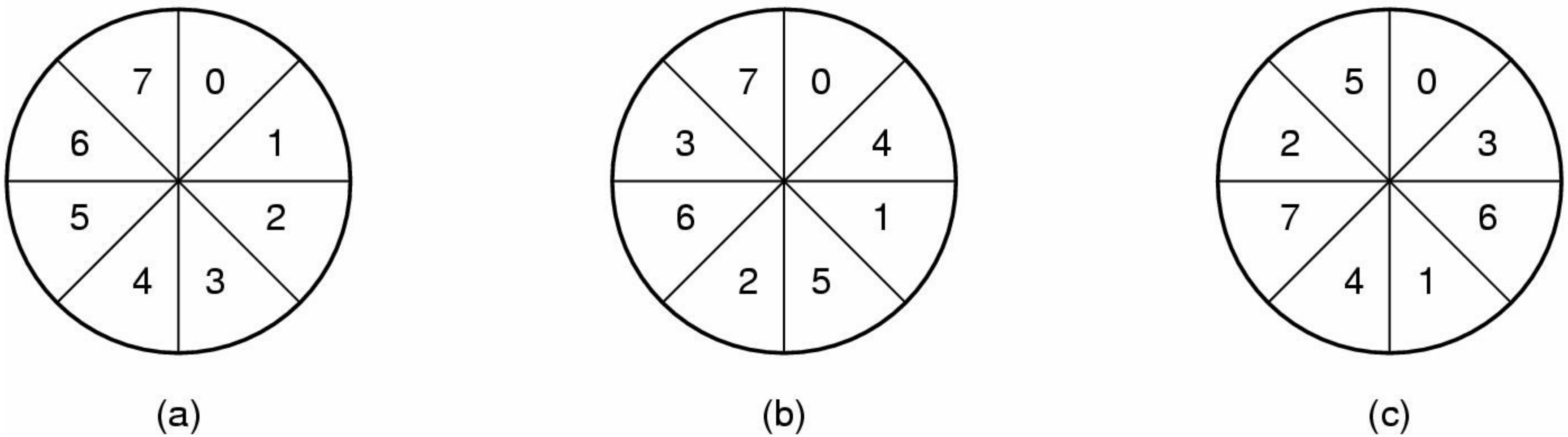
A disk sector

Disk Formatting (2)



An illustration of cylinder skew

Disk Formatting (3)

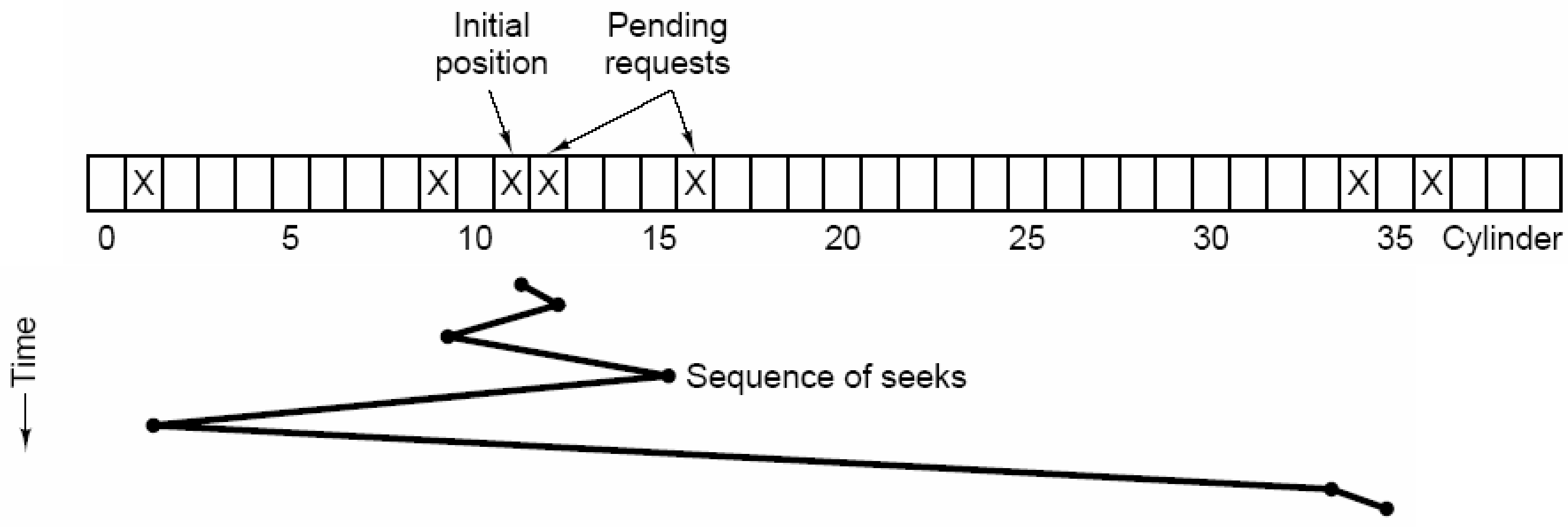


- No interleaving
- Single interleaving
- Double interleaving

Disk Arm Scheduling Algorithms

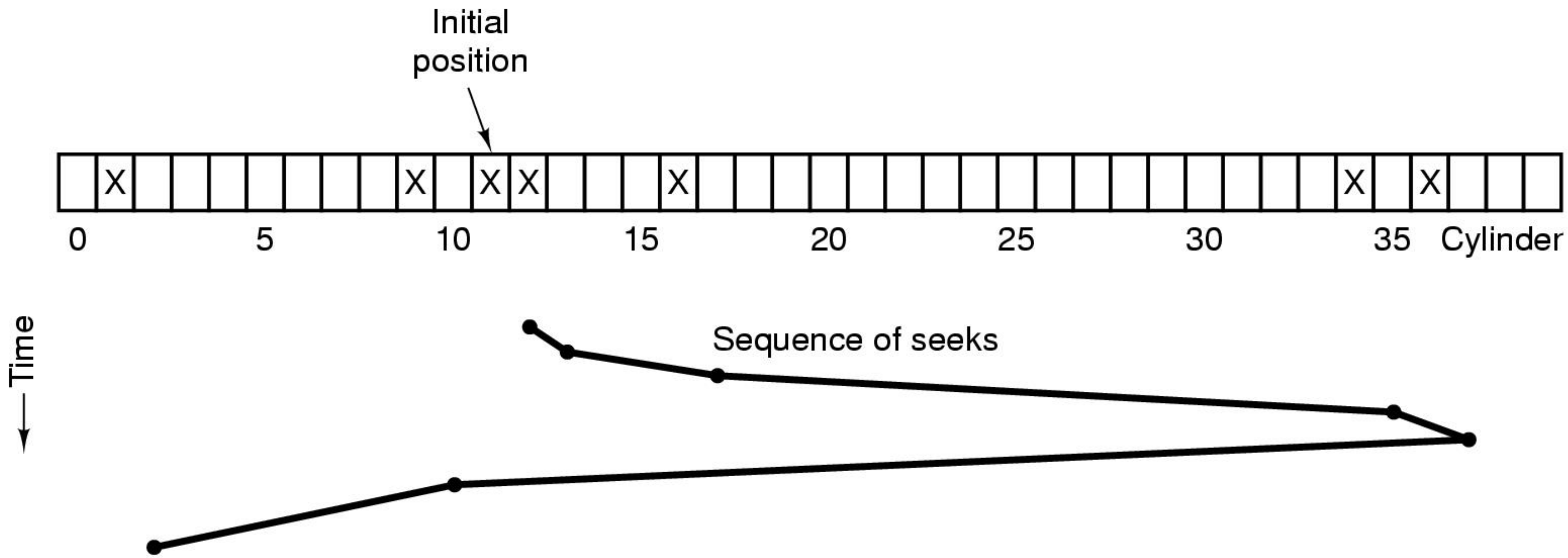
- Time required to read or write a disk block determined by 3 factors
  1. Seek time
  2. Rotational delay
  3. Actual transfer time
- Seek time dominates
- Error checking is done by controllers

Shortest Seek First (SSF) disk scheduling algorithm



Seek request: 11, 1, 36, 16, 34, 9 and 12

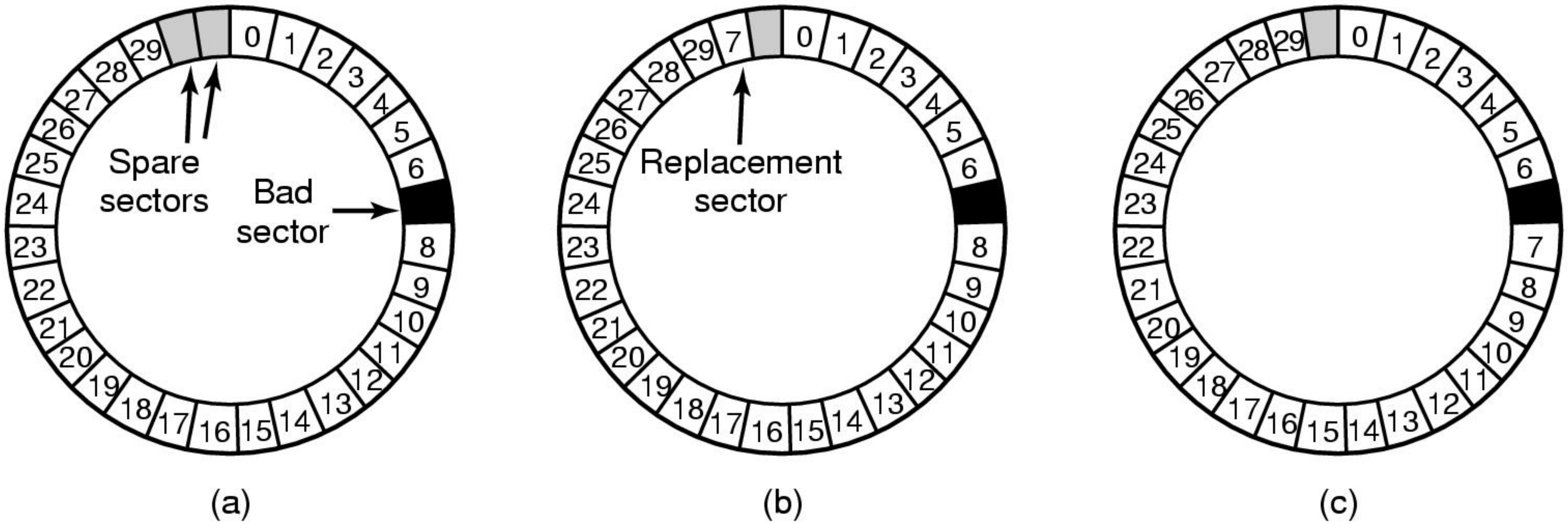
The elevator algorithm for scheduling disk requests



Error Handling

- Small defeat can be corrected by the ECC
- Bigger defeat
  - deal with them in the controller
  - deal with them in the OS

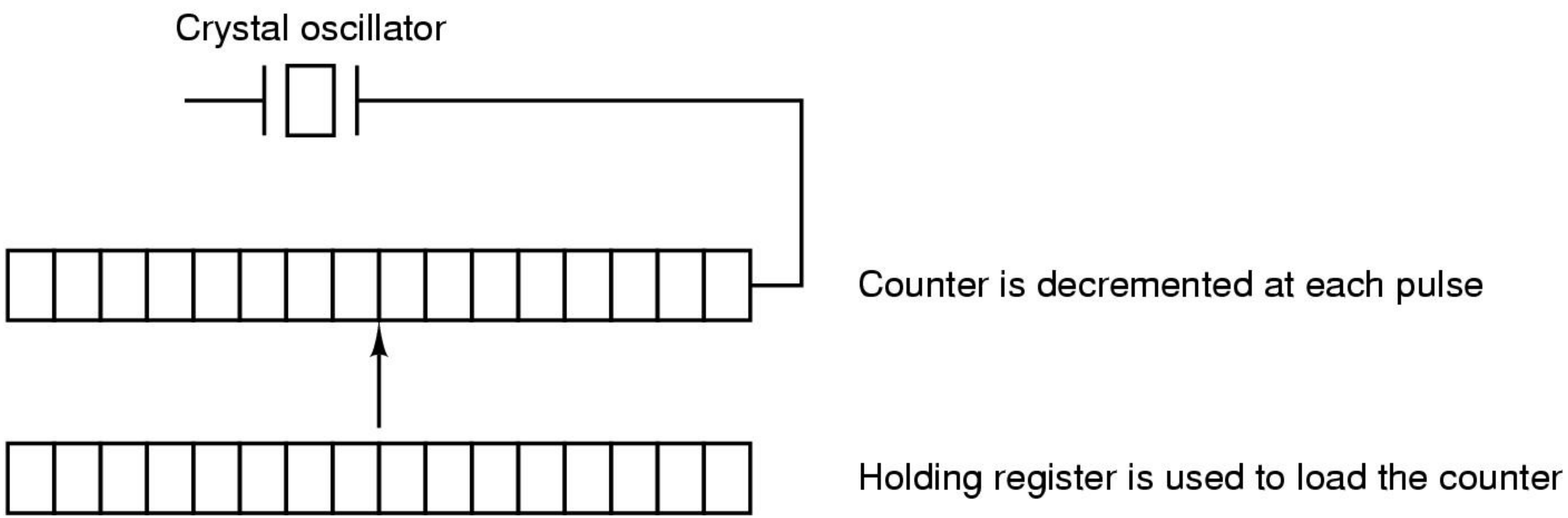
Error Handling in Controller



- A disk track with a bad sector
- Substituting a spare for the bad sector
- Shifting all the sectors to bypass the bad one

Clocks

Clock Hardware

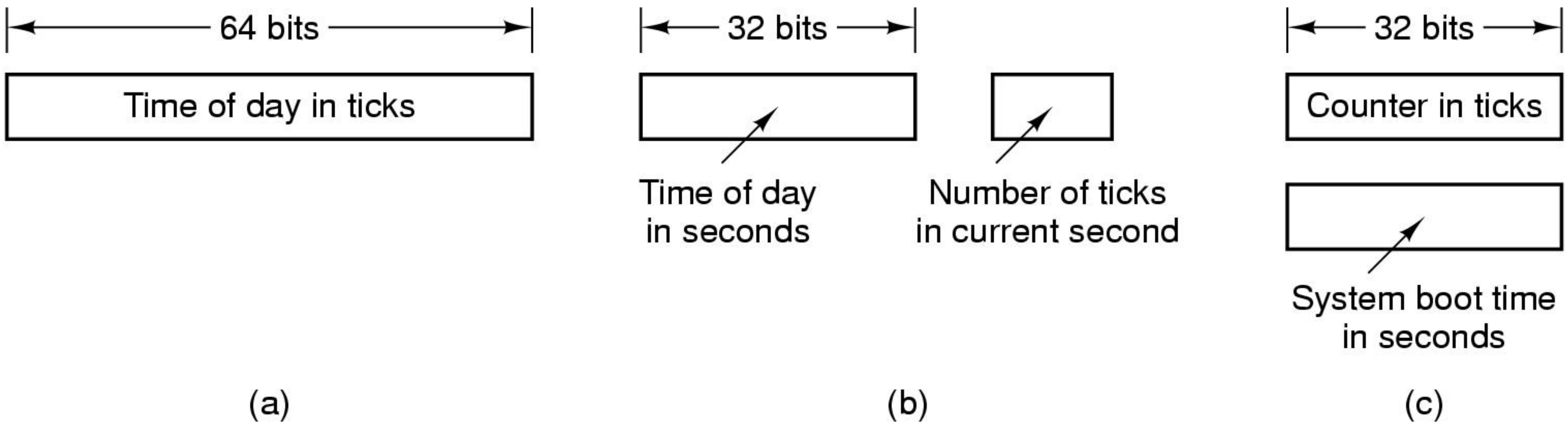


A programmable clock

Clock Software

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

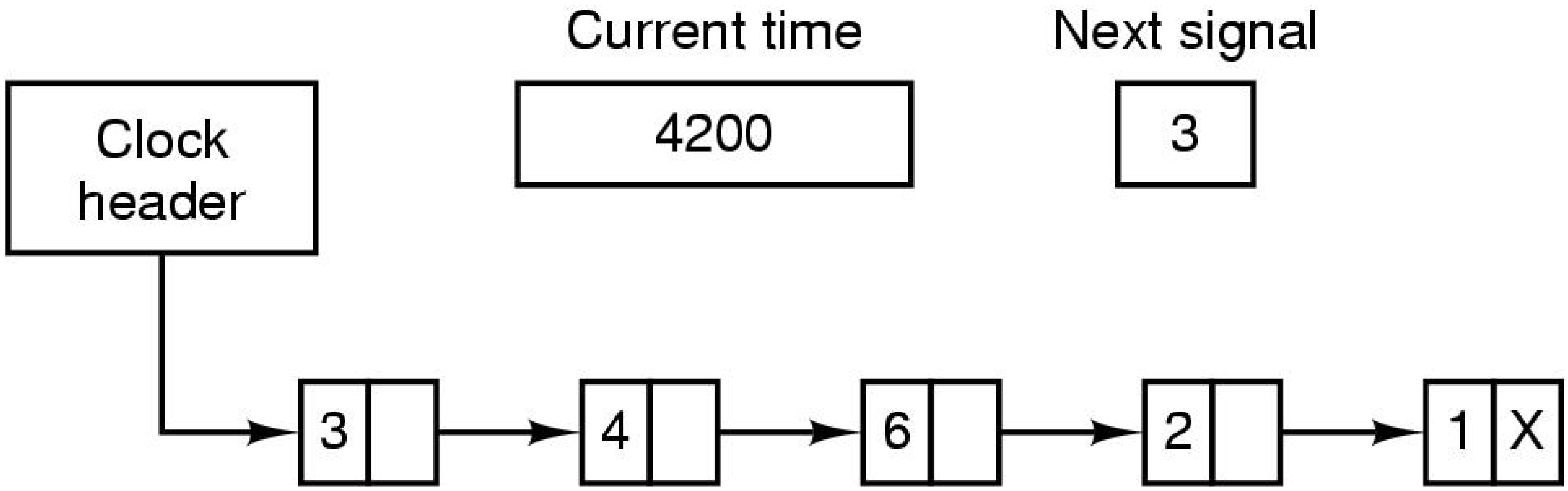
Maintaining the time of day



Clock Software

- **Preventing processes from running longer than they are allowed to**
  - **decrement the quantum counter of running process by 1**
- **Accounting for CPU usage**
  - **use a second timer to count**
  - **count in the PCB**

Handling the alarm system call made by user processes



- **Simulating multiple timers with a single clock**
- **On each tick, *Next signal* is decremented**

Clock Software

- Providing watchdog timers for parts of the system itself
  - for kernel software such as drivers and etc.
- Doing profiling, monitoring, and statistics gathering

Soft Timers

- Soft timers avoid interrupts
  - kernel checks for soft timer expiration before it exits to user mode
  - how well this works depends on rate of kernel entries
- Kernel entries are made for reasons:
  - system calls
  - TLB misses
  - page faults
  - I/O interrupts
  - the CPU going idle

Graphical User Interface (GUI)

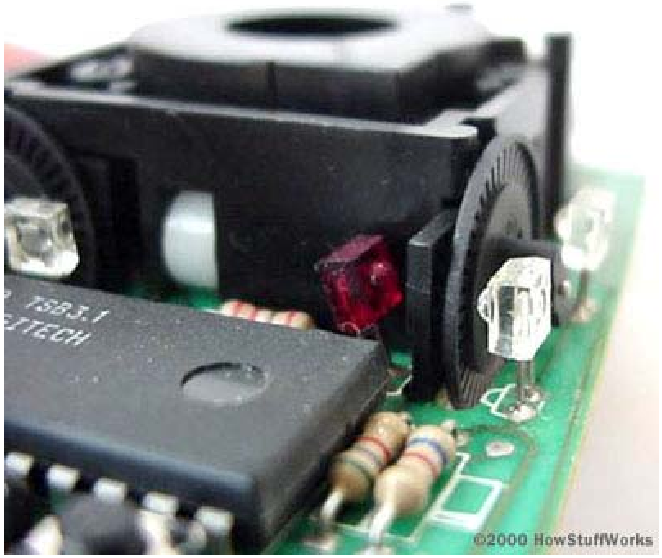
- Invented by Douglas Engelbart and used by Steve Jobs
- WIMP
  - Windows Icons, Menus, and Pointing device
- Can be implemented in either user-level code or in the operating system itself

Keyboard

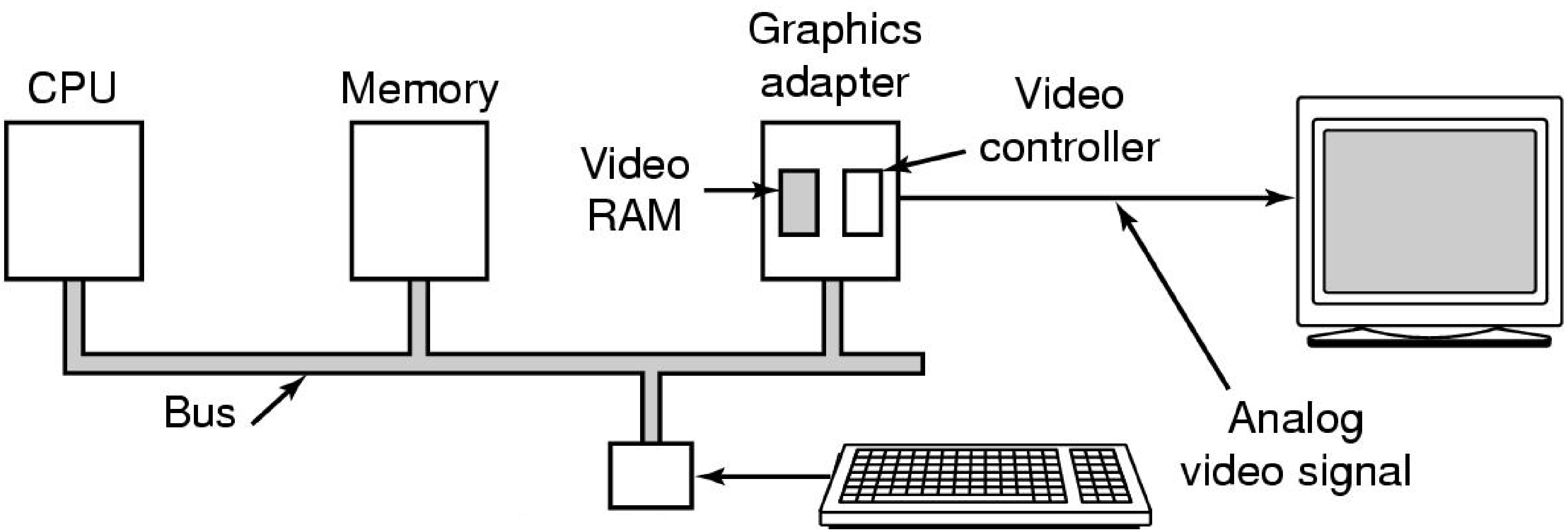
- All that the keyboard hardware provides is the key number, not the ASCII code
- An interrupt is generated whenever a key is struck or released
  - `DEPRESS SHIFT, DEPRESS A, RELEASE A, RELEASE SHIFT`
  - `DEPRESS SHIFT, DEPRESS A, RELEASE SHIFT, RELEASE A`

Mouse

- The messages sent by mice to the computer contains three items:
  - ⊿x, ⊿y, buttons
- Mouse only reports single click



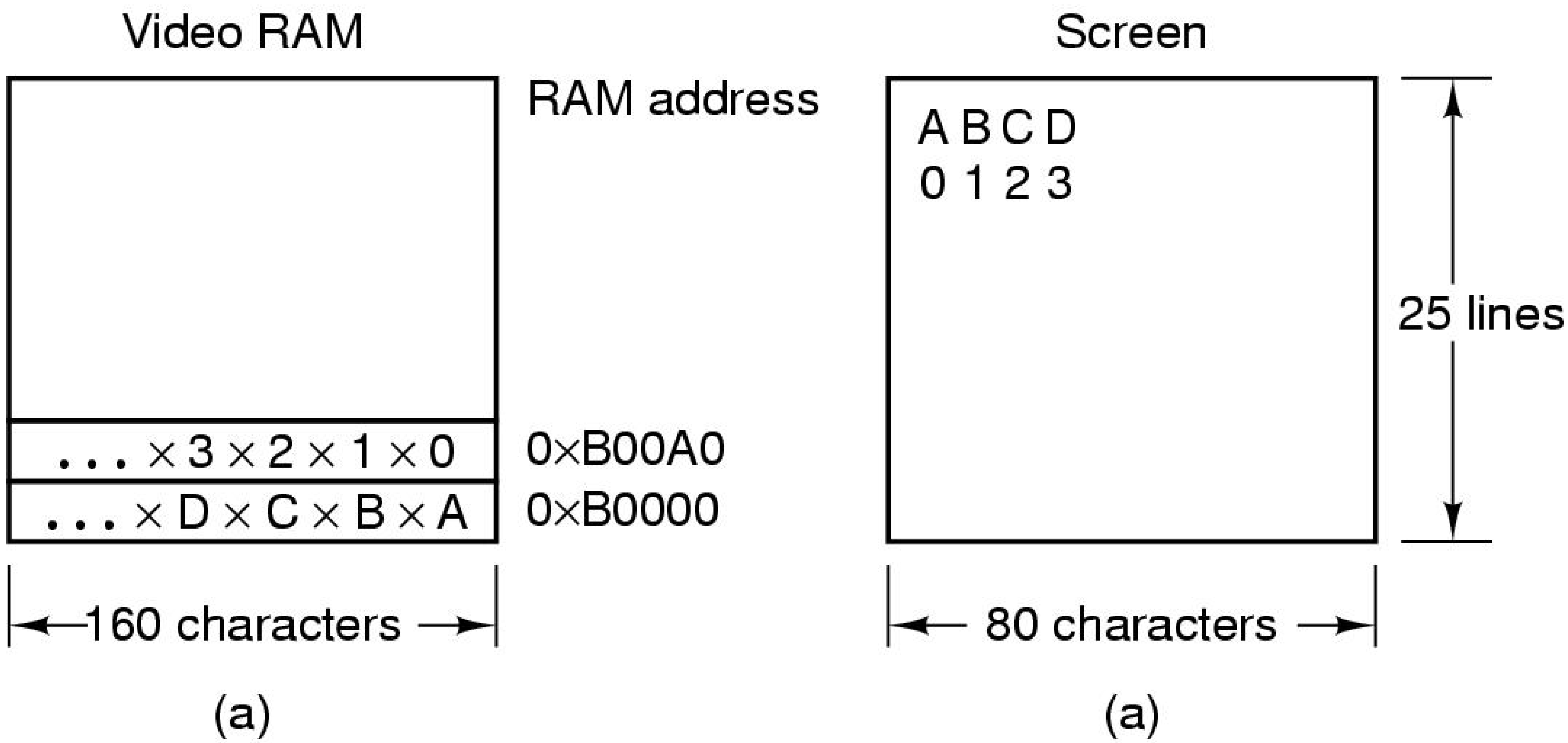
Display Hardware



Memory-mapped displays

- driver writes directly into display's video RAM

Monochrome Display

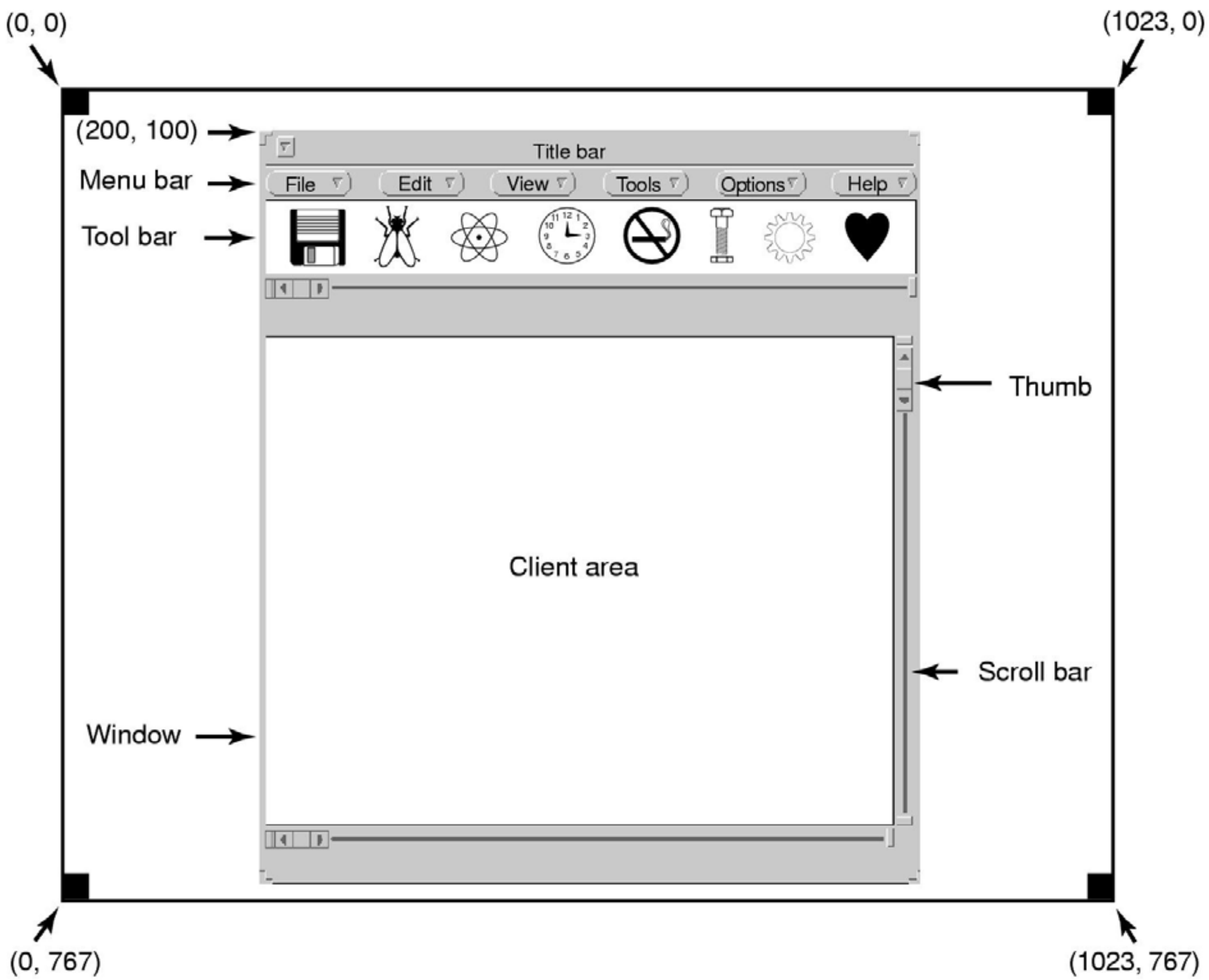


- **A video RAM image**
- **Corresponding screen**

Input Software

- Keyboard driver delivers a number
  - driver converts to characters
  - uses a ASCII table
- Exceptions, adaptations needed for other languages
  - many OS provide for loadable keymaps or code pages

Output Software for Windows



Sample window located at (200,100) on XGA display

Output Software for Windows

```
#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 */
```

Skeleton of a Windows main program (part 1)

Output Software for Windows

```
    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);
}

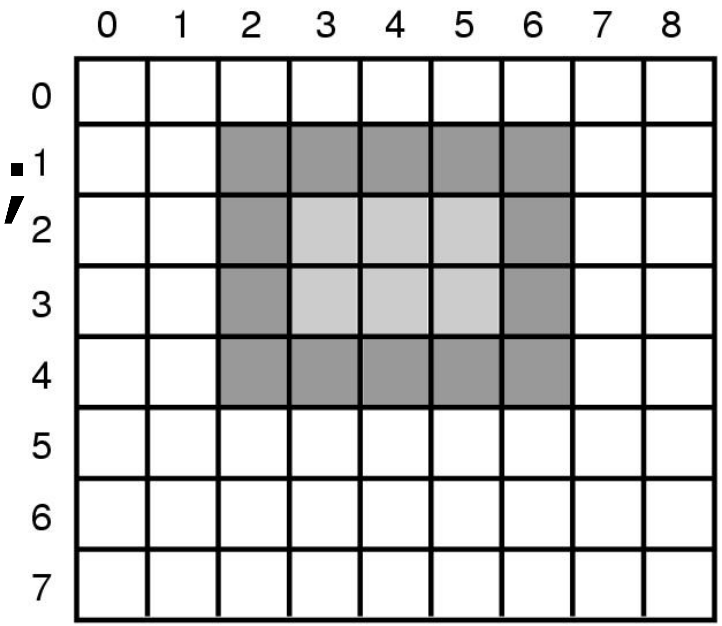
long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)
{
    /* Declarations go here. */

    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 */
}
```

Skeleton of a Windows main program (part 2)

GDI
(Graphical Device Interface)

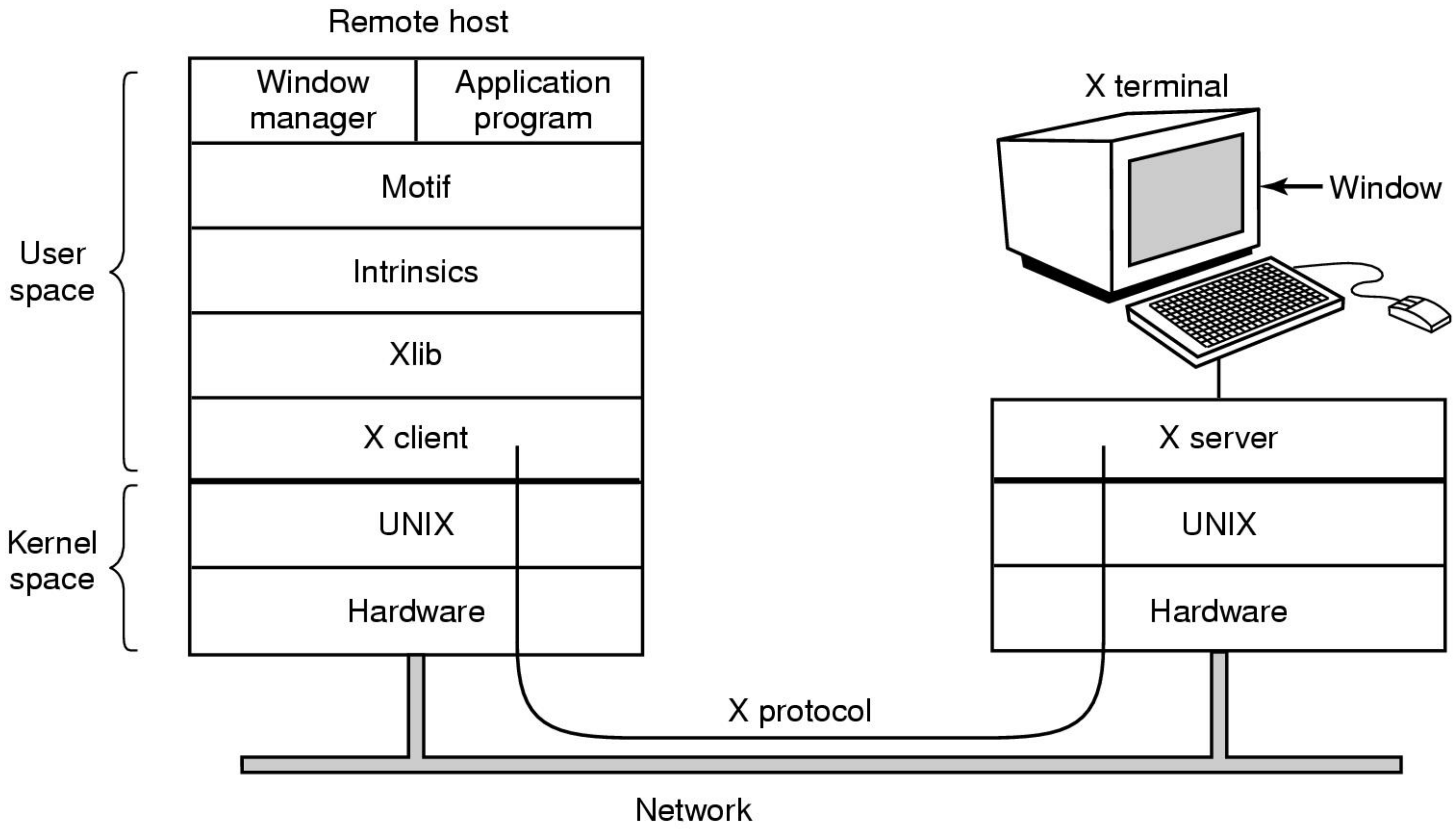
- ```
  hdc = GetDC(hwnd);
  TextOut(
     hdc, x, y,
     psText, iLength);
  ReleaseDC(
     hwnd, hdc);
  ```



- `Rectangle(hdc, 2, 1, 6, 4)`
  ```

Network Terminals

X Windows



Clients and servers in the M.I.T. X Window System

Skeleton of an X Windows Application Program

```
#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 */
```

Skeleton of an X Windows Application Program

```
    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 */
}
```