Chapter 5: I/O Systems

Input/Output

- Principles of I/O hardware
- Principles of I/O software
- I/O software layers
- Disks
- Clocks
- Character-oriented terminals
- Graphical user interfaces
- Network terminals
- Power management

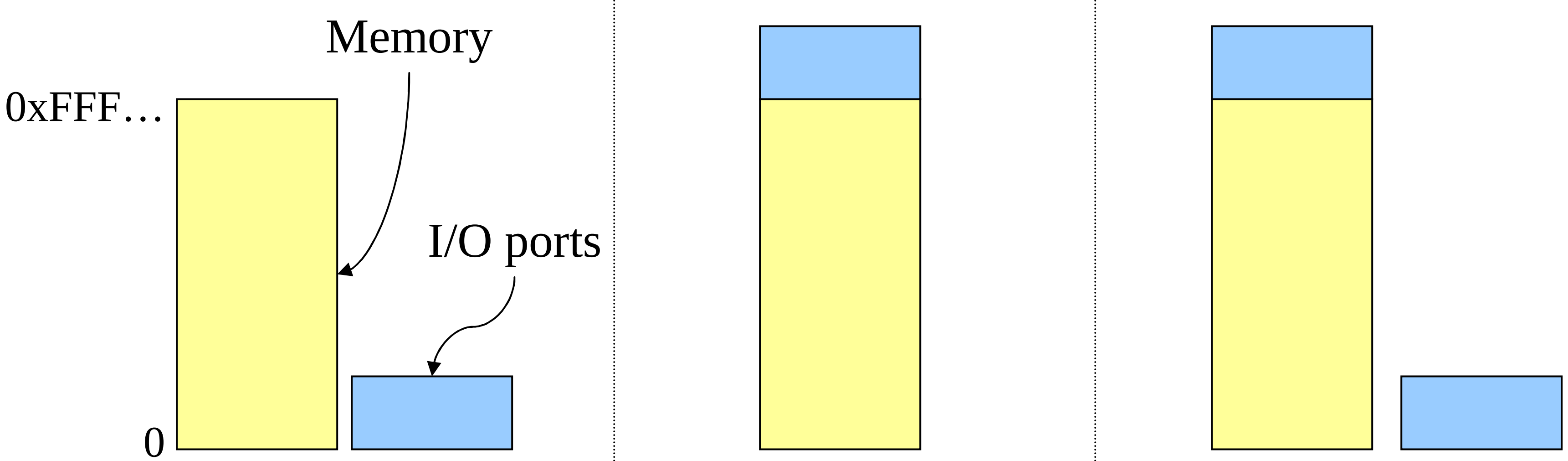
How fast is I/O hardware?

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Printer / scanner	200 KB/sec
USB	1.5 MB/sec
Digital camcorder	4 MB/sec
Fast Ethernet	12.5 MB/sec
Hard drive	20 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA monitor	60 MB/sec
PCI bus	500 MB/sec

Device controllers

- I/O devices have components
  - Mechanical component
  - Electronic component
- Electronic component controls the device
  - May be able to handle multiple devices
  - May be more than one controller per mechanical component (example: hard drive)
- Controller's tasks
  - Convert serial bit stream to block of bytes
  - Perform error correction as necessary
  - Make available to main memory

Memory-Mapped I/O



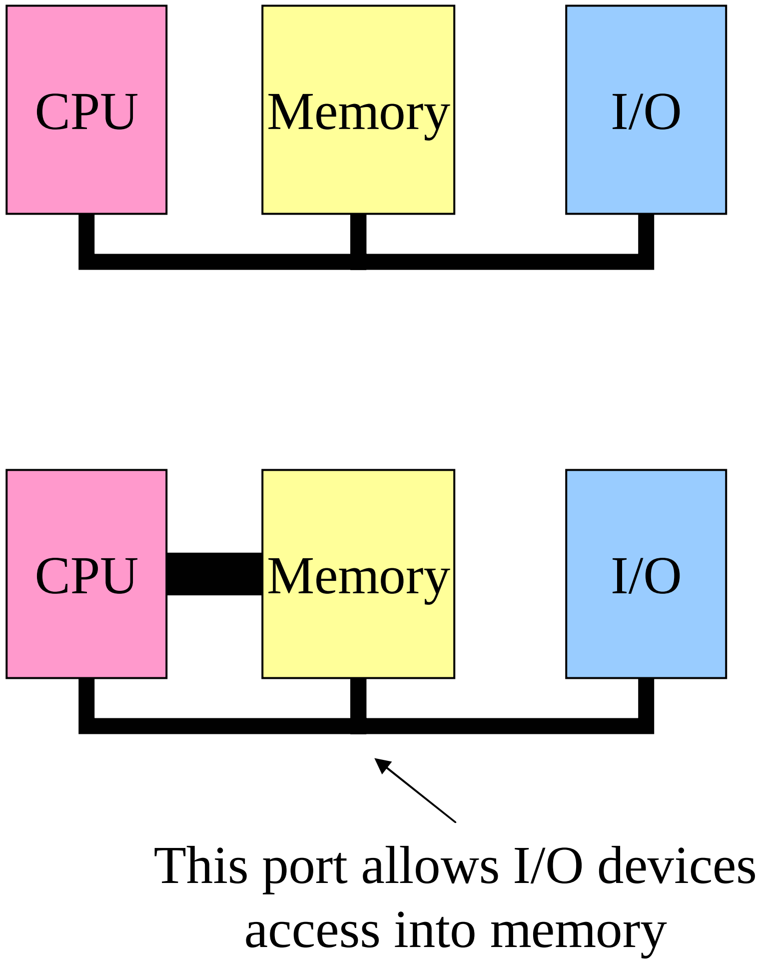
Separate I/O & memory space

Memory-mapped I/O

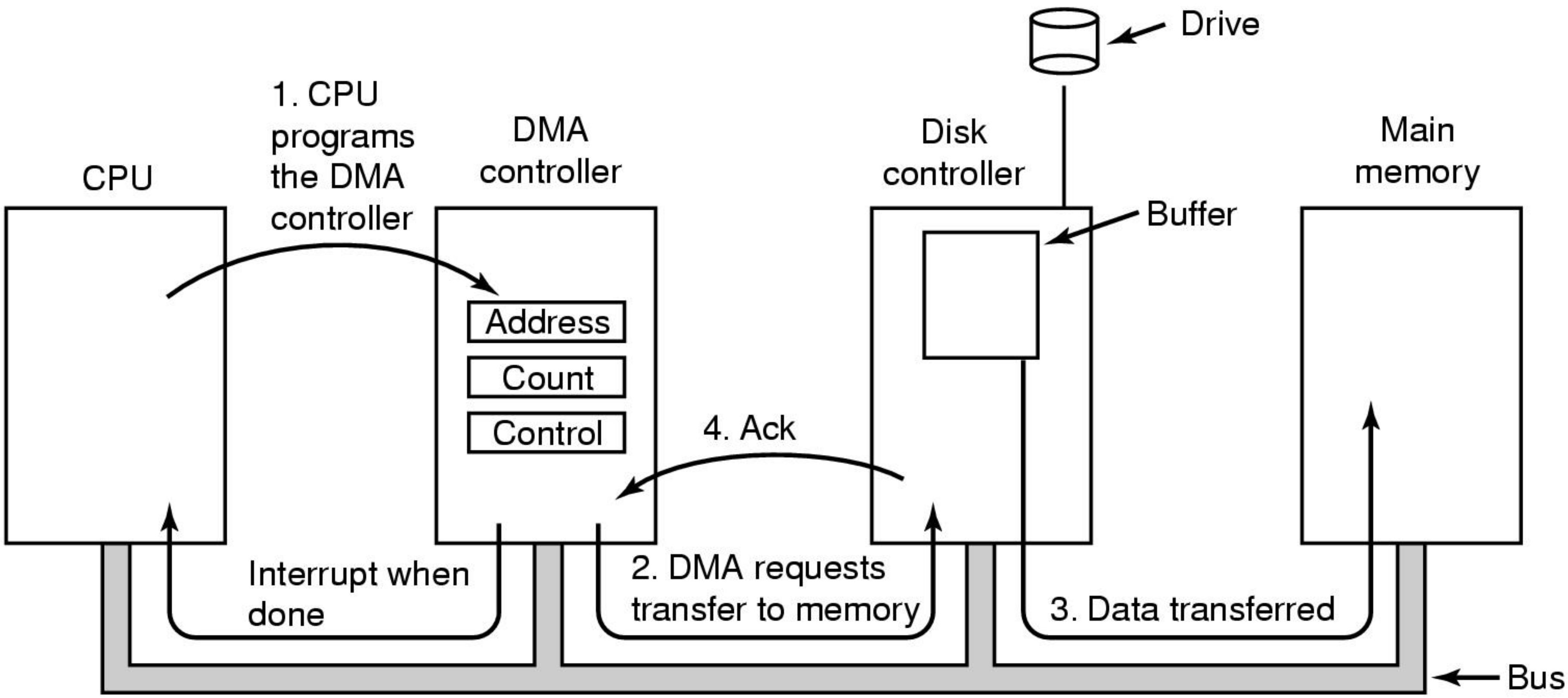
Hybrid: both memory-mapped & separate spaces

How is memory-mapped I/O done?

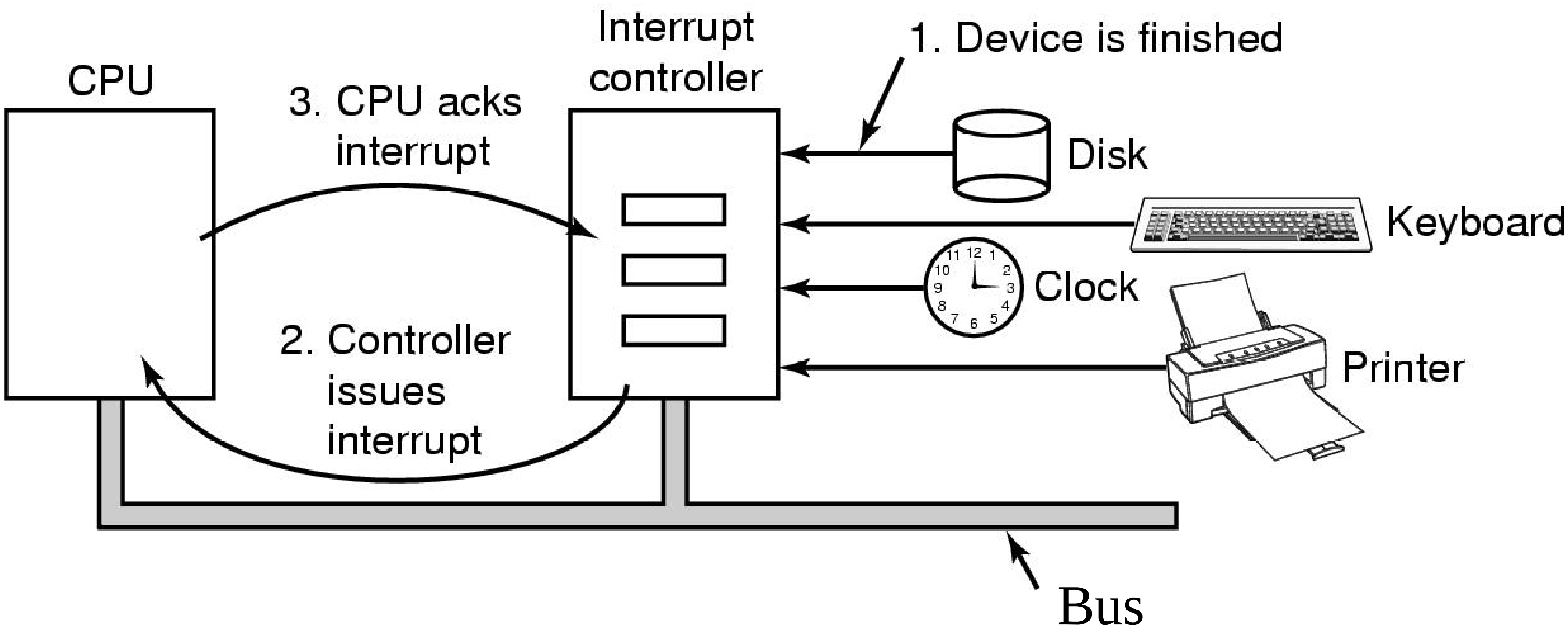
- Single-bus
  - All memory accesses go over a shared bus
  - I/O and RAM accesses compete for bandwidth
- Dual-bus
  - RAM access over high-speed bus
  - I/O access over lower-speed bus
  - Less competition
  - More hardware (more expensive…)



Direct Memory Access (DMA) operation



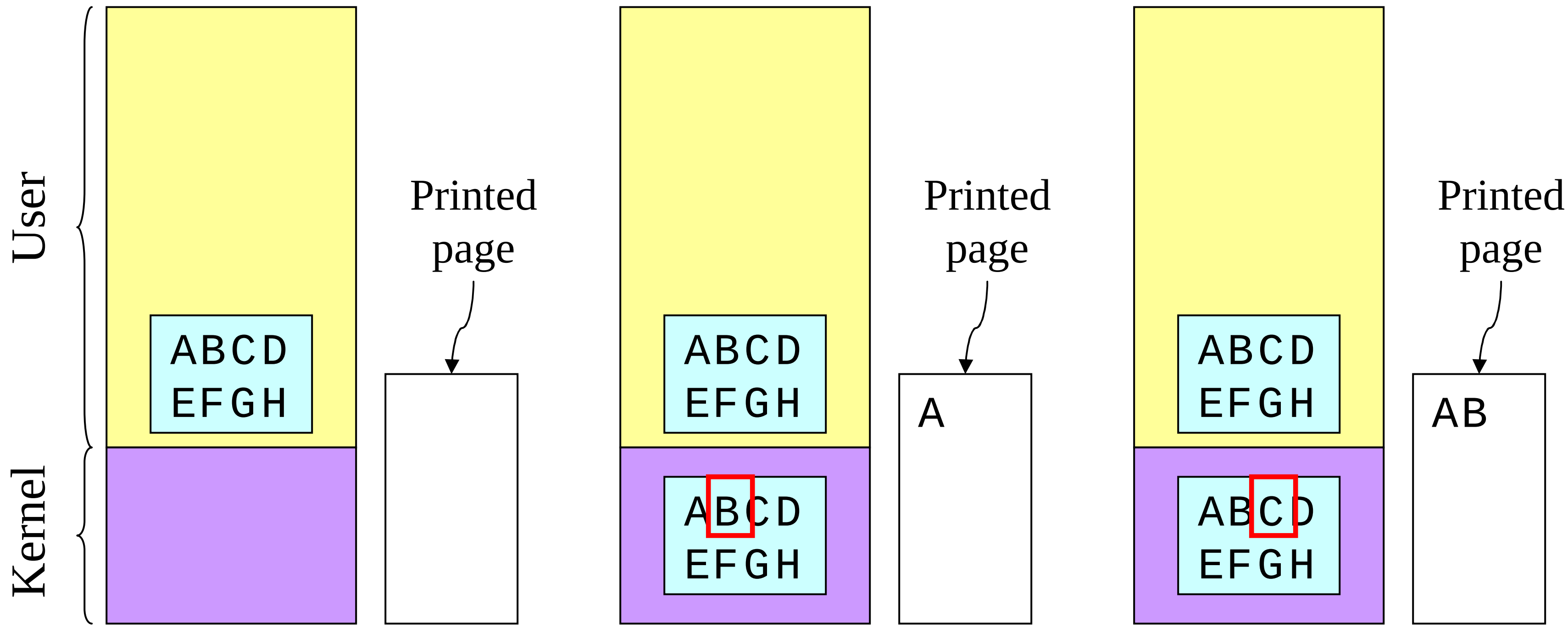
Hardware's view of interrupts



I/O software: goals

- Device independence
  - Programs can access any I/O device
  - No need to specify device in advance
- Uniform naming
  - Name of a file or device is a string or an integer
  - Doesn’t depend on the machine (underlying hardware)
- Error handling
  - Done as close to the hardware as possible
  - Isolate higher-level software
- 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

Programmed I/O: printing a page



Code for programmed I/O

```
copy_from_user (buffer, p, count);  // copy into kernel buffer
for (j = 0; j < count; j++) {  // loop for each char
  while (*printer_status_reg != READY)
    ;                          // wait for printer to be ready
  *printer_data_reg = p[j];    // output a single character
}
return_to_user();
```

Interrupt-driven I/O

```
copy_from_user (buffer, p, count);
j = 0;
enable_interrupts();
while (*printer_status_reg != READY)
  ;
*printer_data_reg = p[0];
scheduler(); // and block user
```

Code run by system call

```
if (count == 0) {
  unblock_user();
} else {
  *printer_data_reg = p[j];
  count--;
  j++;
}
acknowledge_interrupt();
return_from_interrupt();
```

Code run at interrupt time

I/O using DMA

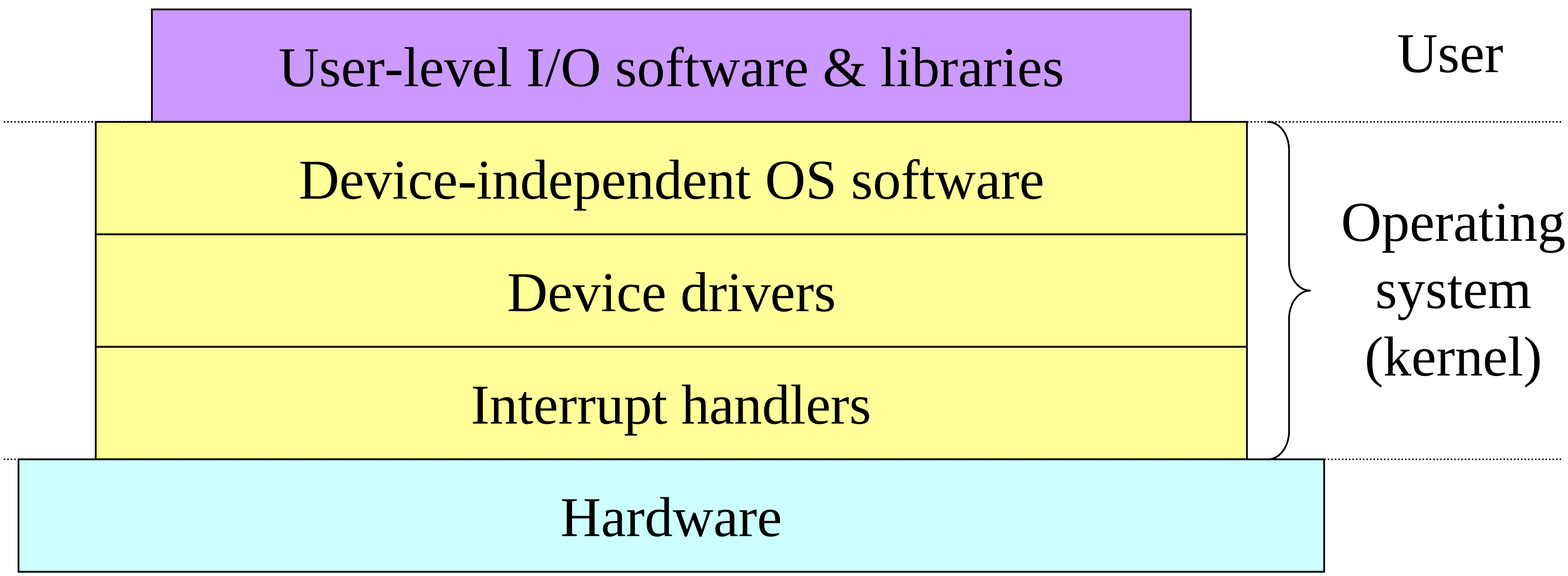
```
copy_from_user (buffer, p, count);
set_up_DMA_controller();
scheduler(); // and block user
```

Code run by system call

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

Code run at interrupt time

Layers of I/O software



Interrupt handlers

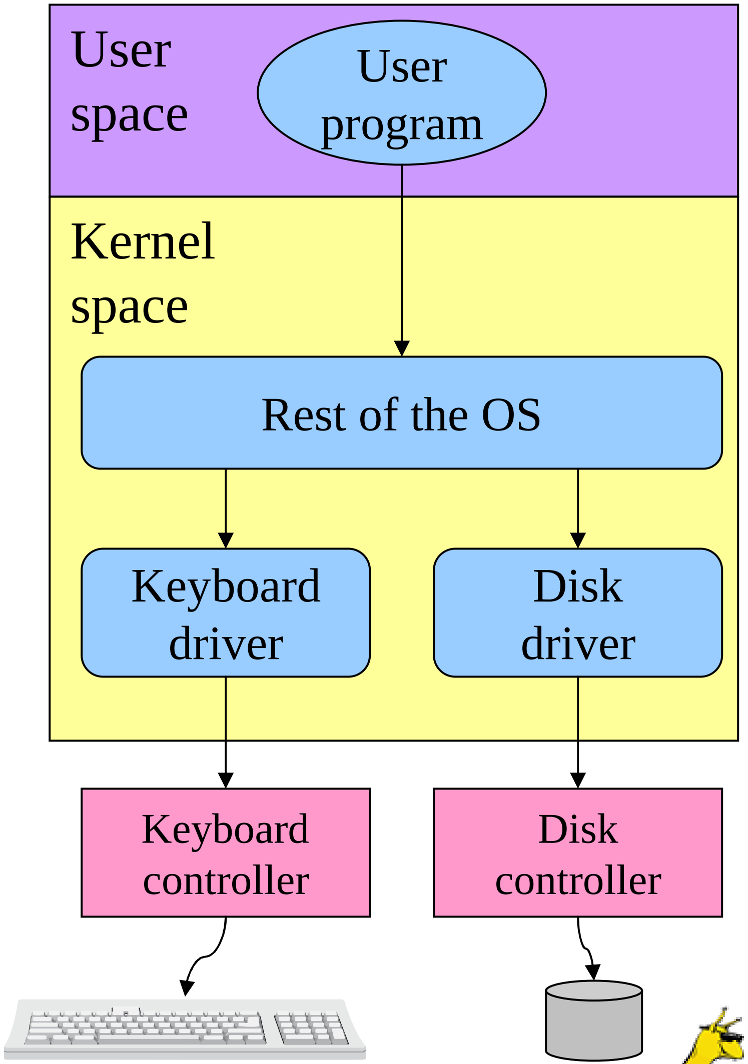
- Interrupt handlers are best hidden
  - Driver starts an I/O operation and blocks
  - Interrupt notifies of completion
- Interrupt procedure does its task
  - Then unblocks driver that started it
  - Perform minimal actions at interrupt time
    - Some of the functionality can be done by the driver after it is unblocked
- Interrupt handler must
  - Save regs not already saved by interrupt hardware
  - Set up context for interrupt service procedure
  - DLXOS: intrhandler (in dlxos.s)

What happens on an interrupt

- Set up stack for interrupt service procedure
- Ack interrupt controller, reenable interrupts
- Copy registers from where saved
- Run service procedure
- (optional) Pick a new process to run next
- Set up MMU context for process to run next
- Load new process' registers
- Start running the new process

Device drivers

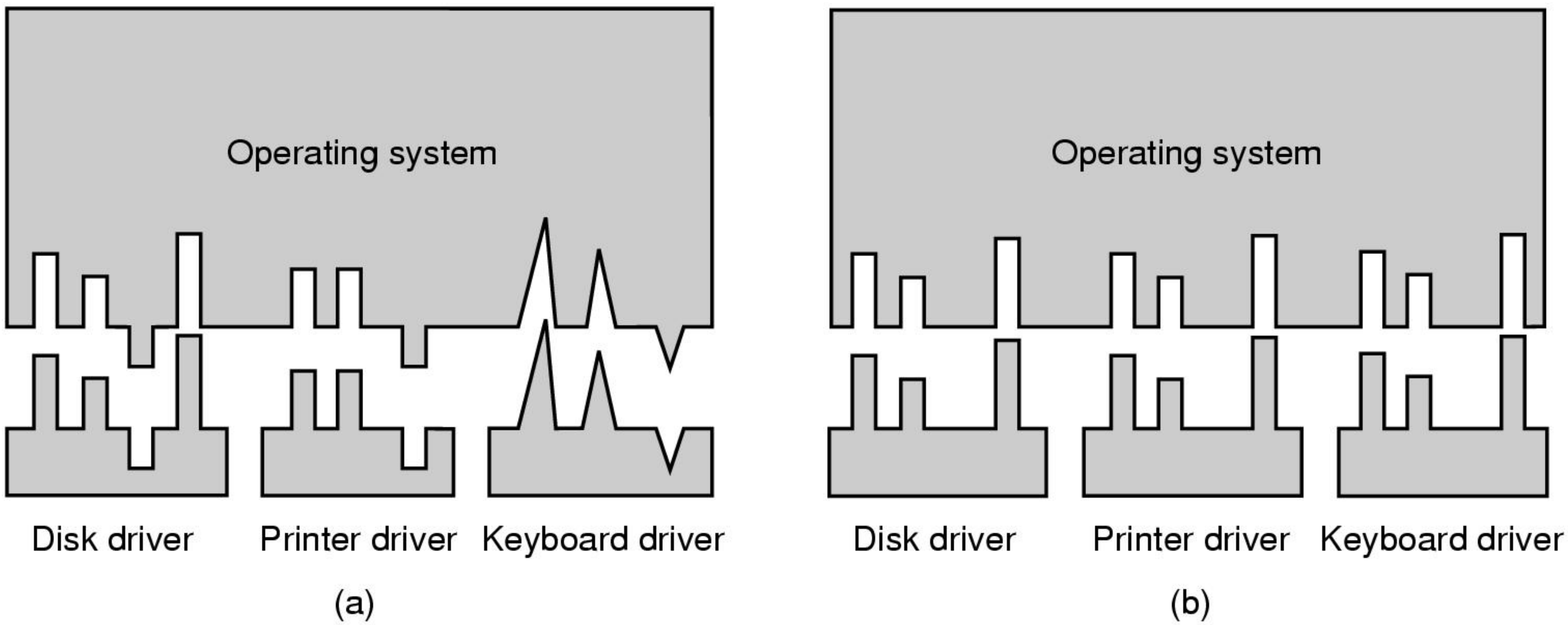
- Device drivers go between device controllers and rest of OS
  - Drivers standardize interface to widely varied devices
- Device drivers communicate with controllers over bus
  - Controllers communicate with devices themselves



Device-independent I/O software

- Device-independent I/O software provides common “library” routines for I/O software
- Helps drivers maintain a standard appearance to the rest of the OS
- Uniform interface for many device drivers for
  - Buffering
  - Error reporting
  - Allocating and releasing dedicated devices
  - Suspending and resuming processes
- Common resource pool
  - Device-independent block size (keep track of blocks)
  - Other device driver resources

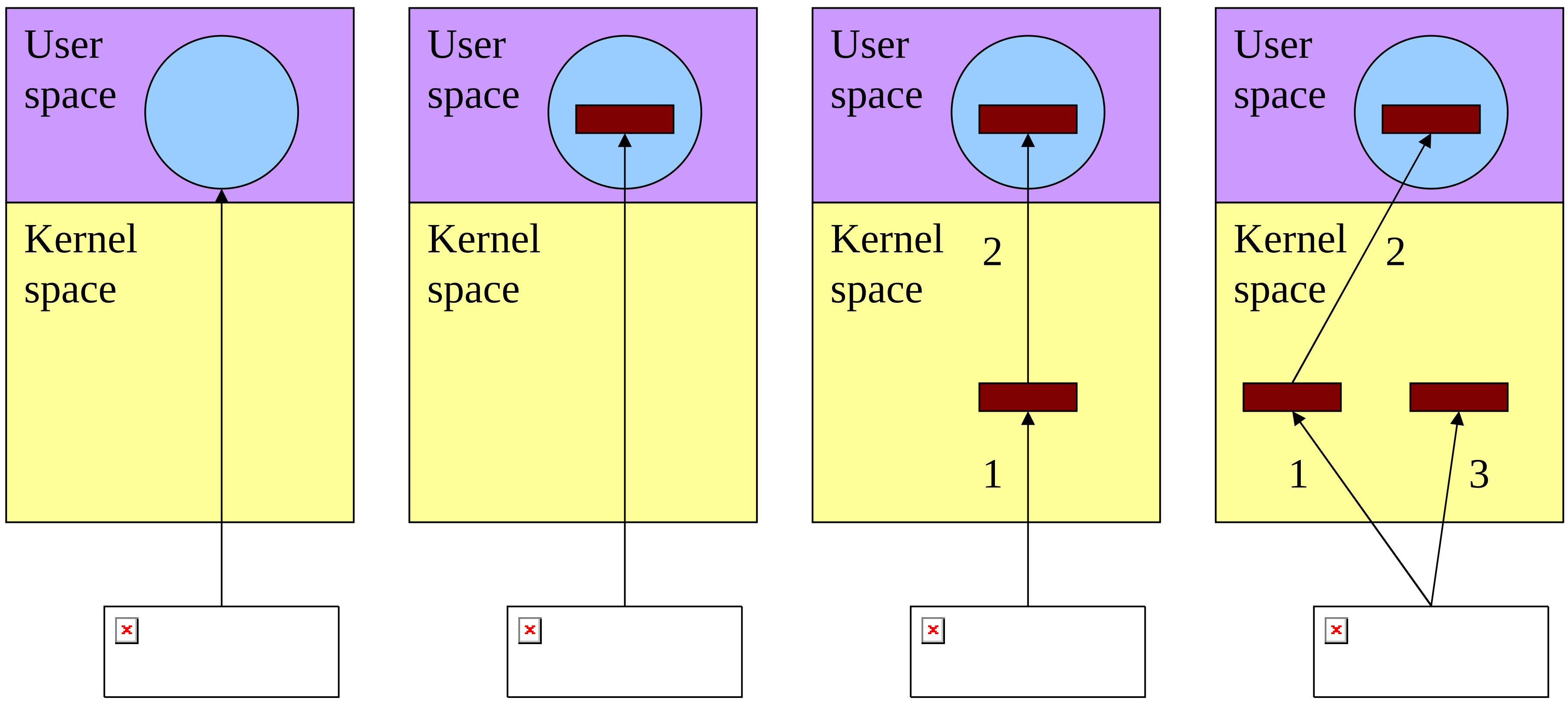
Why a standard driver interface?



Non-standard driver interfaces

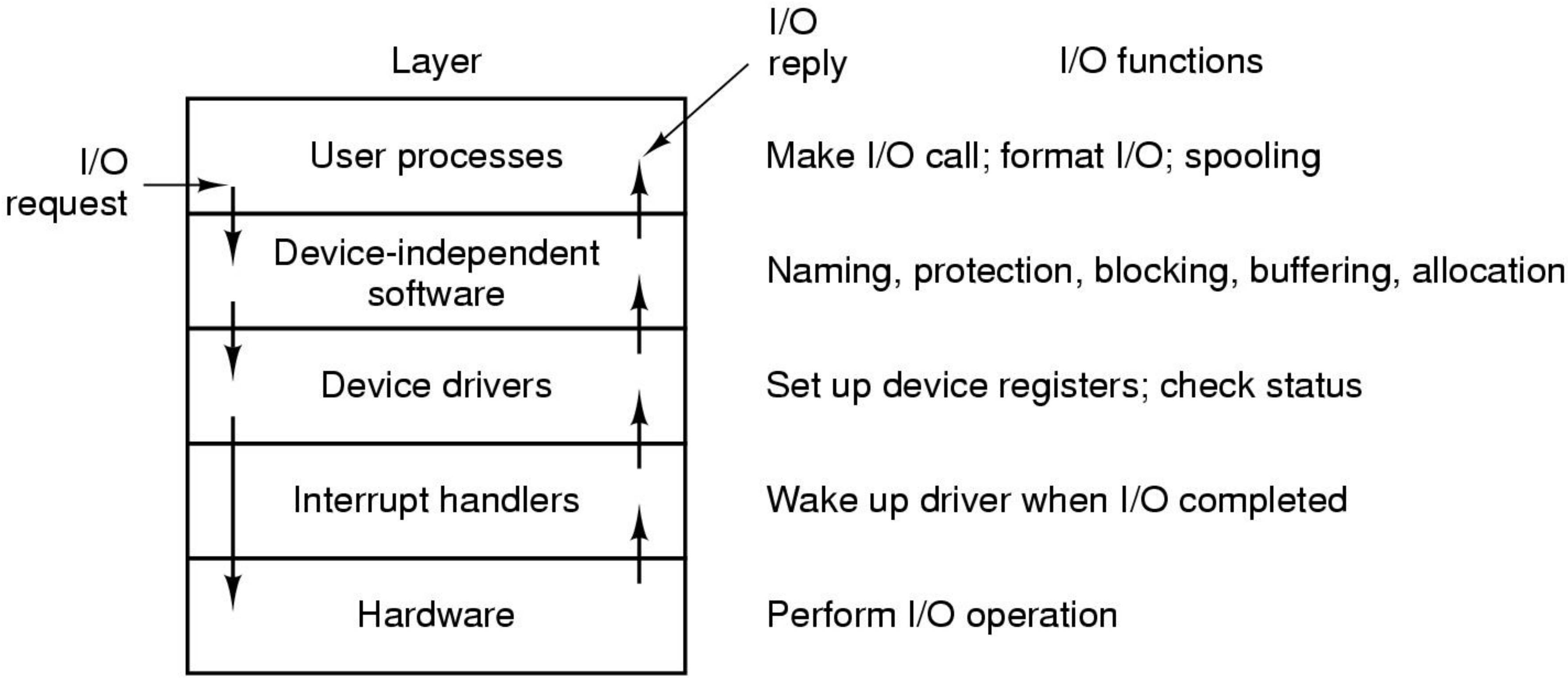
Standard driver interfaces

Buffering device input



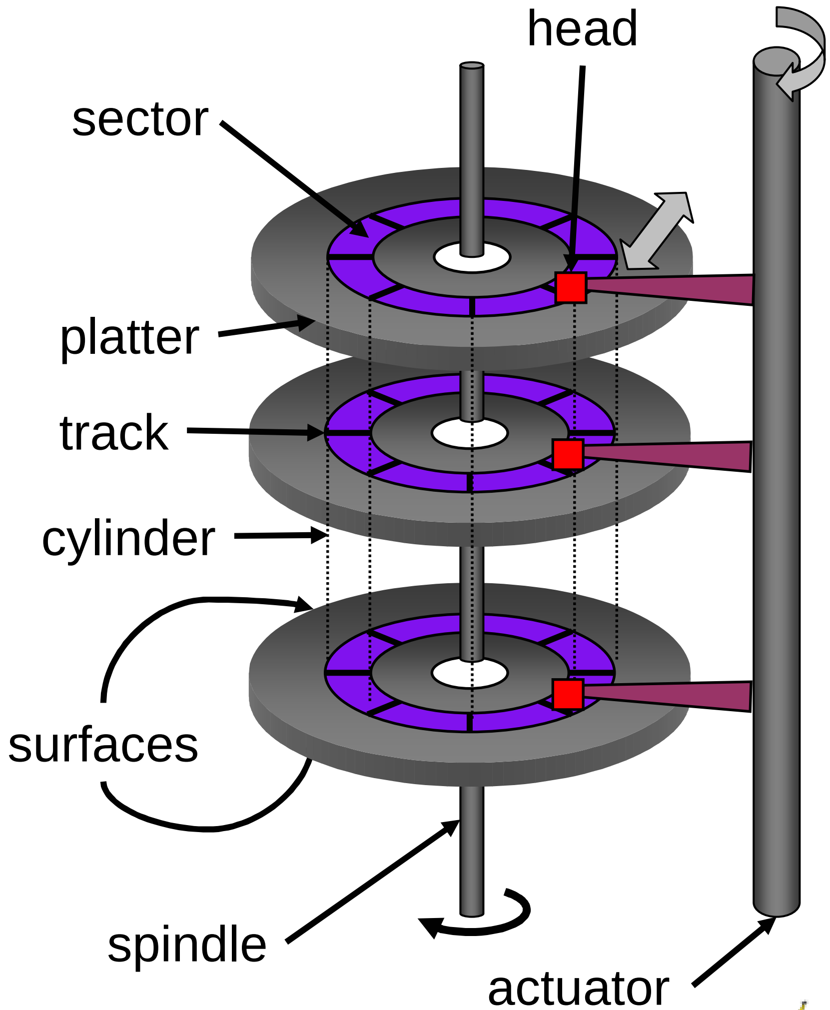
Unbuffered input | Buffering in user space | Buffer in kernel Copy to user space | Double buffer in kernel

Anatomy of an I/O request



Disk drive structure

- Data stored on surfaces
  - Up to two surfaces per platter
  - One or more platters per disk
- Data in concentric tracks
  - Tracks broken into sectors
    - 256B-1KB per sector
  - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
  - Actuator moves heads
  - Heads move in unison

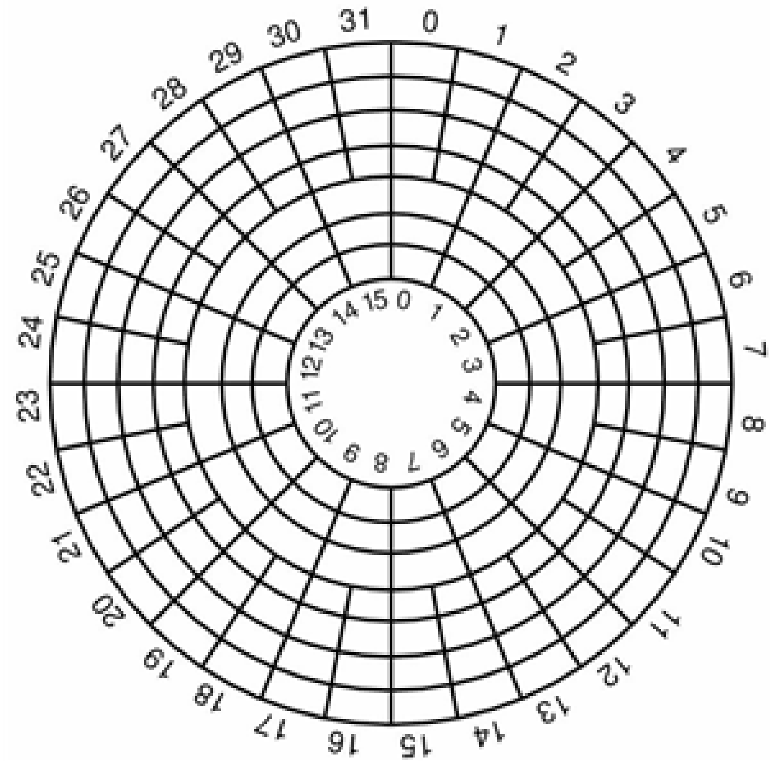


Disk drive specifics

	IBM 360KB floppy	WD 18GB HD
Cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (average)
Sectors per disk	720	35742000
Bytes per sector	512	512
Capacity	360 KB	18.3 GB
Seek time (minimum)	6 ms	0.8 ms
Seek time (average)	77 ms	6.9 ms
Rotation time	200 ms	8.33 ms
Spinup time	250 ms	20 sec
Sector transfer time	22 ms	17 μsec

Disk “zones”

- Outside tracks are longer than inside tracks
- Two options
  - Bits are “bigger”
  - More bits (transfer faster)
- Modern hard drives use second option
  - More data on outer tracks
- Disk divided into “zones”
  - Constant sectors per track in each zone
  - 8–20 (or more) zones on a disk



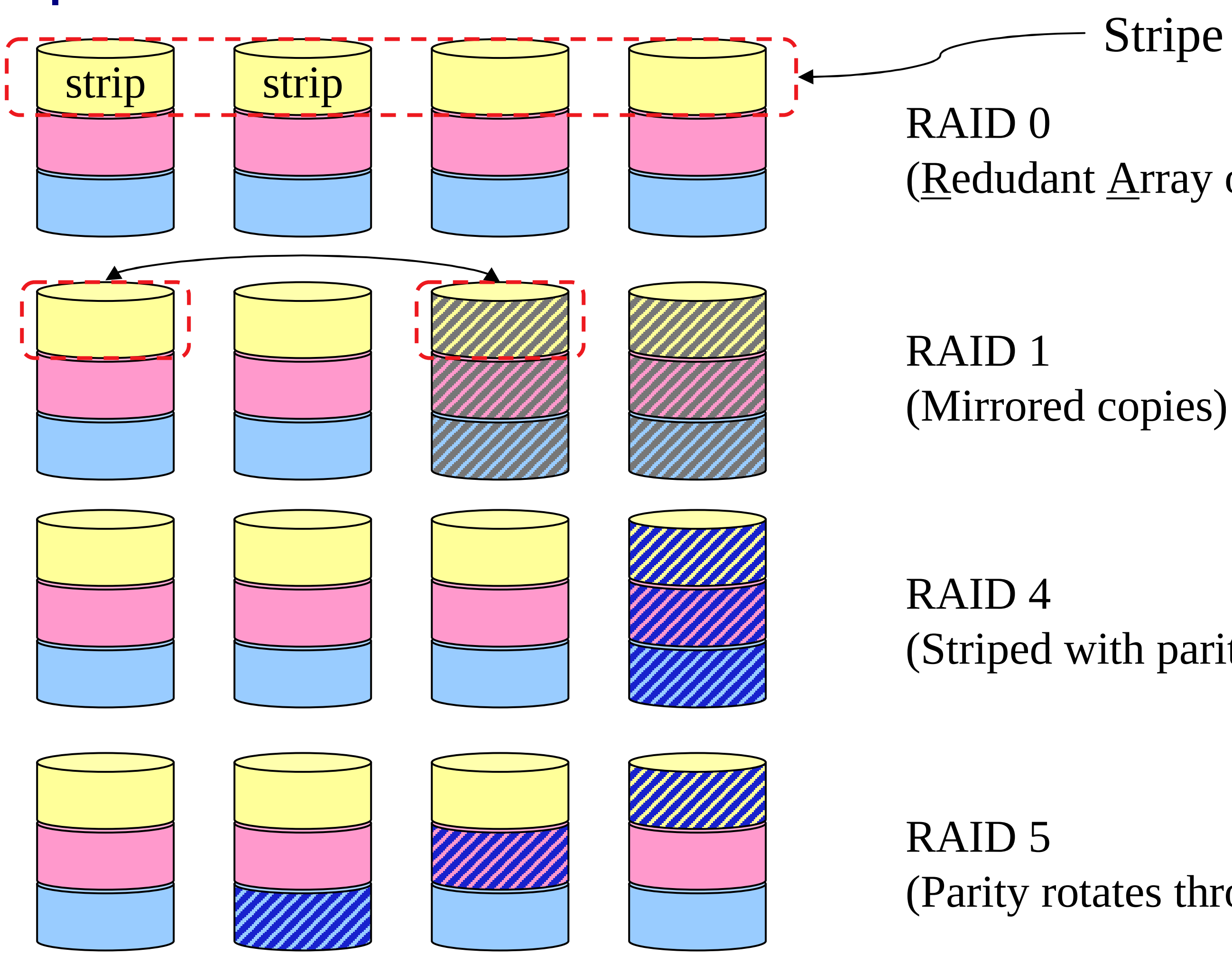
Disk "addressing"

- Millions of sectors on the disk must be labeled
- Two possibilities
  - Cylinder/track/sector
  - Sequential numbering
- Modern drives use sequential numbers
  - Disks map sequential numbers into specific location
  - Mapping may be modified by the disk
    - Remap bad sectors
    - Optimize performance
  - Hide the exact geometry, making life simpler for the OS

Building a better "disk"

- Problem: CPU performance has been increasing exponentially, but disk performance hasn't
  - Disks are limited by mechanics
- Problem: disks aren't all that reliable
- Solution: distribute data across disks, and use some of the space to improve reliability
  - Data transferred in parallel
  - Data stored across drives (*striping*)
  - Parity on an "extra" drive for reliability

RAIDs, RAIDs, and more RAIDs



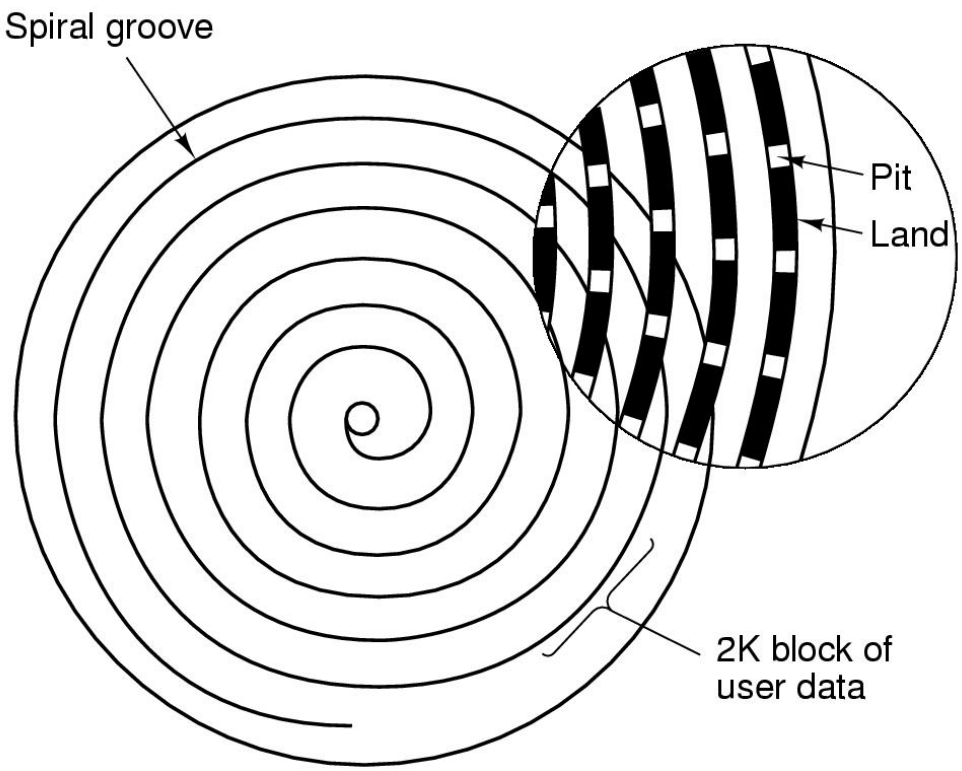
RAID 0
(Redudant Array of Inexpensive Disks

RAID 1
(Mirrored copies)

RAID 4
(Striped with parity)

RAID 5
(Parity rotates through disks)

CD-ROM recording



- CD-ROM has data in a spiral
  - Hard drives have concentric circles of data
- One continuous track: just like vinyl records!
- Pits & lands "simulated" with heat-sensitive material on CD-Rs and CD-RWs

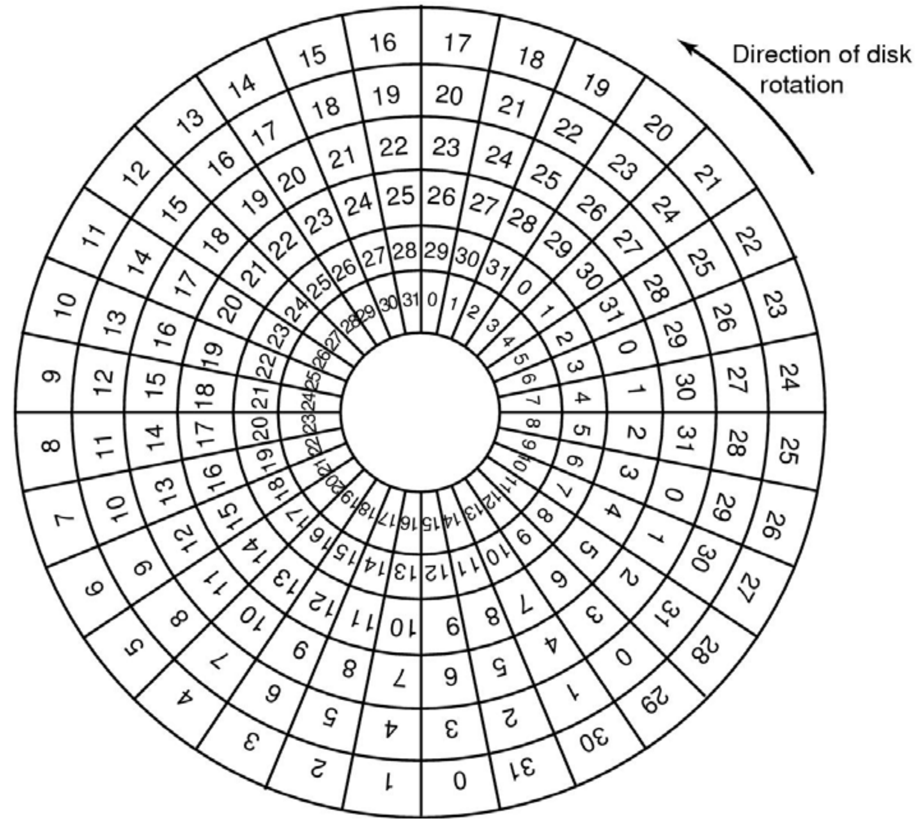
Structure of a disk sector

Preamble	Data	ECC

- Preamble contains information about the sector
  - Sector number & location information
- Data is usually 256, 512, or 1024 bytes
- ECC (Error Correcting Code) is used to detect & correct minor errors in the data

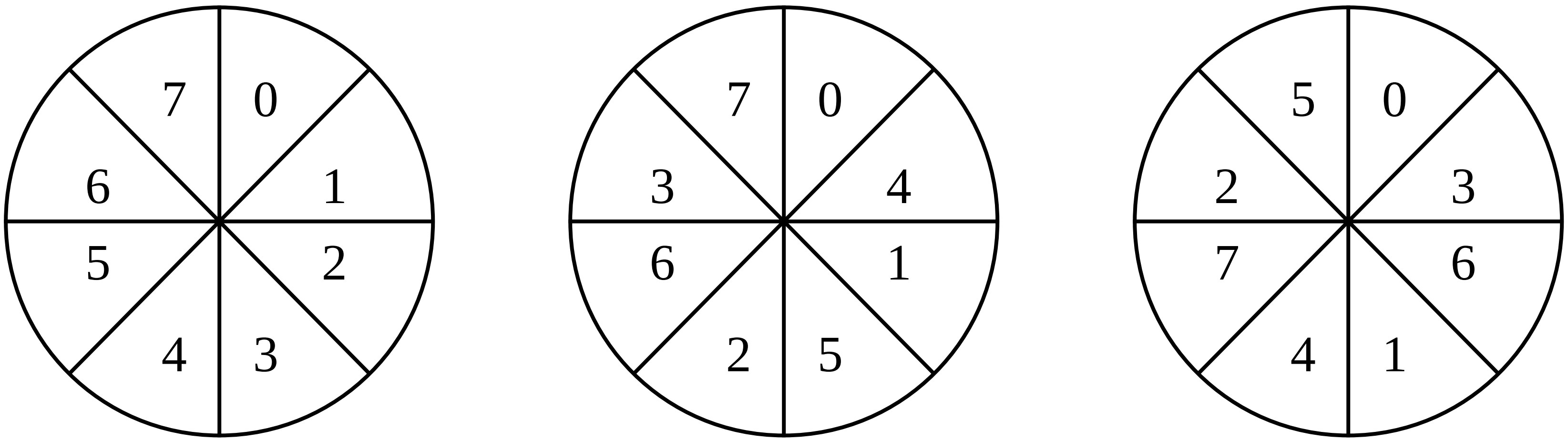
Sector layout on disk

- Sectors numbered sequentially on each track
- Numbering starts in different place on each track: *sector skew*
  - Allows time for switching head from track to track
- All done to minimize delay in sequential transfers



Sector interleaving

- On older systems, the CPU was slow => time elapsed between reading consecutive sectors
- Solution: leave space between consecutively numbered sectors
- This isn't done much these days…



No interleaving Skipping 1 sector Skipping 2 sectors

What's in a disk request?

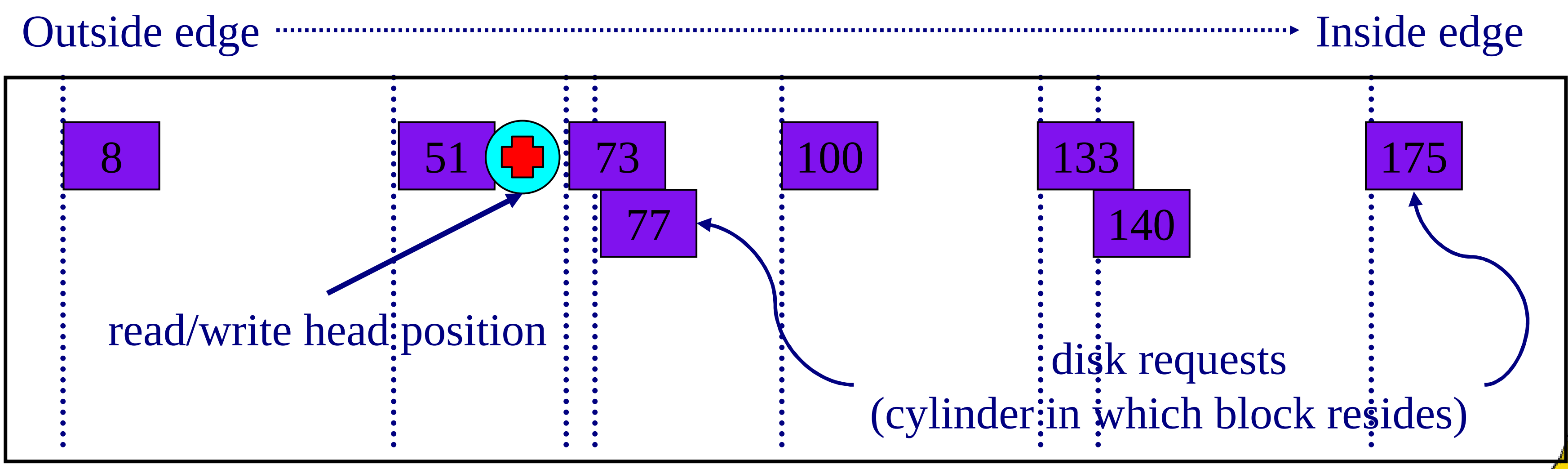
- Time required to read or write a disk block determined by 3 factors
  - Seek time
  - Rotational delay
    - Average delay = 1/2 rotation time
    - Example: rotate in 10ms, average rotation delay = 5ms
  - Actual transfer time
    - Transfer time = time to rotate over sector
    - Example: rotate in 10ms, 200 sectors/track => 10/200 ms = 0.05ms transfer time per sector
- Seek time dominates, with rotation time close
- Error checking is done by controllers

Disk request scheduling

- Goal: use disk hardware efficiently
  - Bandwidth as high as possible
  - Disk transferring as often as possible (and not seeking)
- We want to
  - Minimize disk seek time (moving from track to track)
  - Minimize rotational latency (waiting for disk to rotate the desired sector under the read/write head)
- Calculate disk bandwidth by
  - Total bytes transferred / time to service request
  - Seek time & rotational latency are overhead (no data is transferred), and reduce disk bandwidth
- Minimize seek time & rotational latency by
  - Using algorithms to find a good sequence for servicing requests
  - Placing blocks of a given file "near" each other

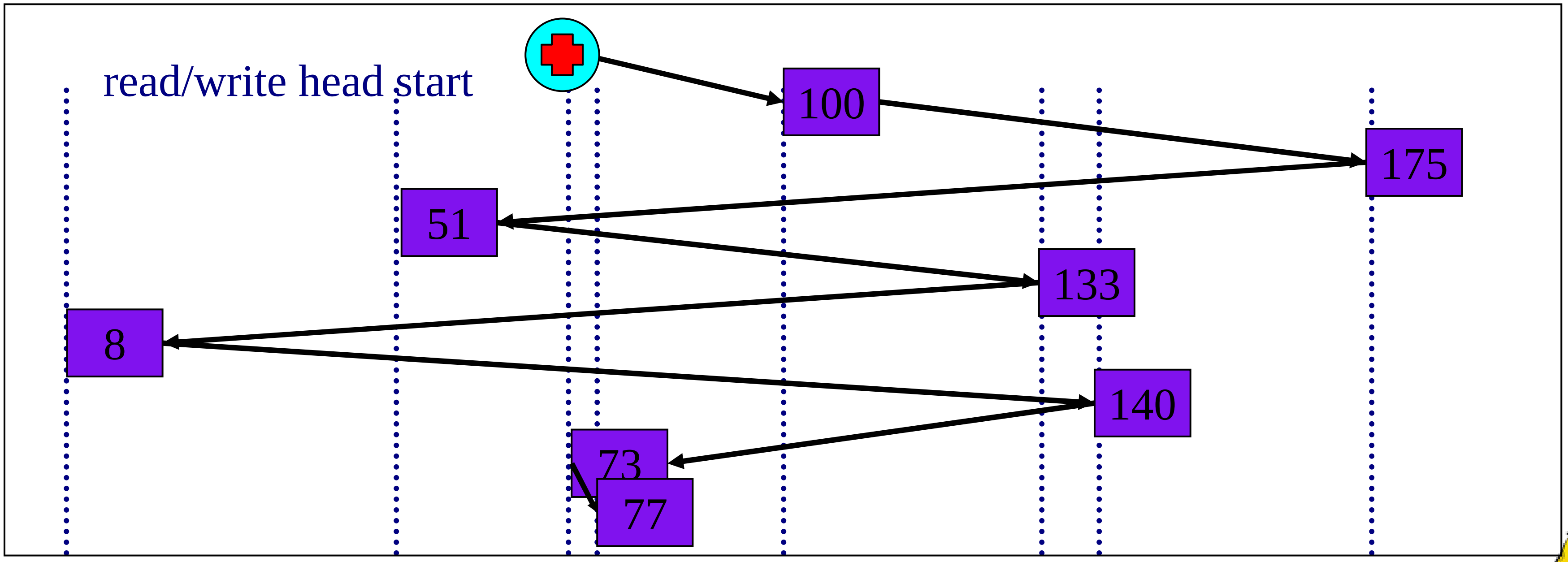
Disk scheduling algorithms

- Schedule disk requests to minimize disk seek time
  - Seek time increases as distance increases (though not linearly)
  - Minimize seek distance -> minimize seek time
- Disk seek algorithm examples assume a request queue & head position (disk has 200 cylinders)
  - Queue = 100, 175, 51, 133, 8, 140, 73, 77
  - Head position = 63



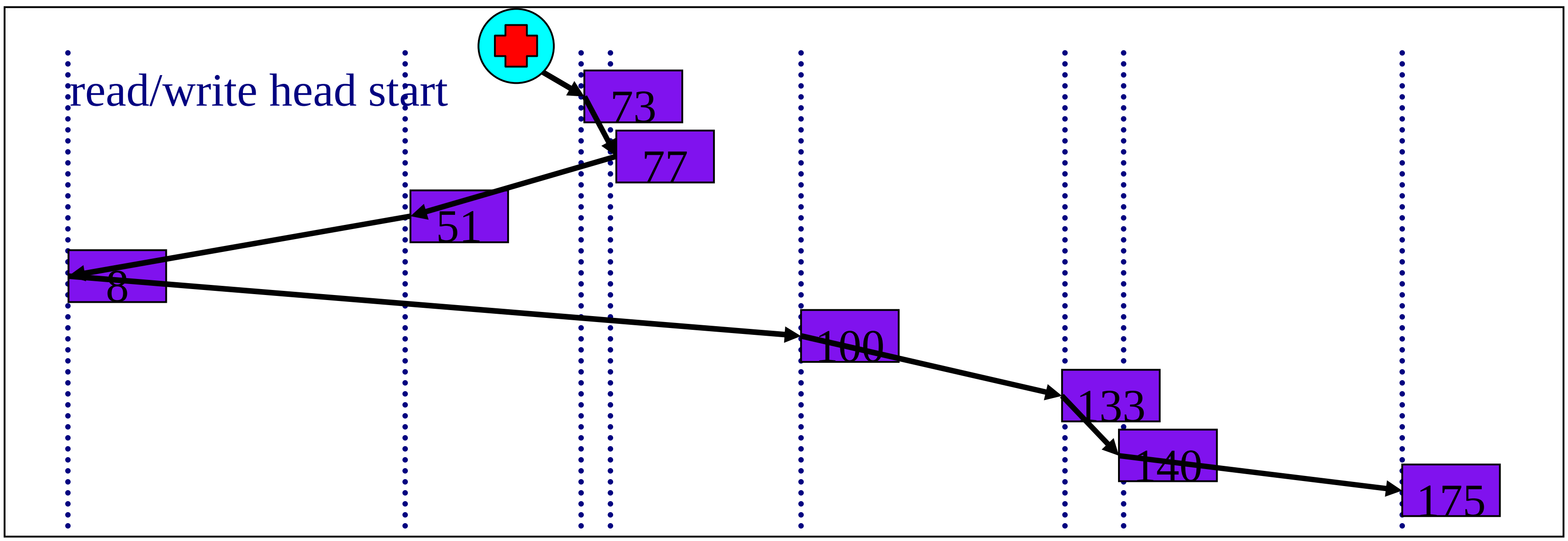
First-Come-First Served (FCFS)

- Requests serviced in the order in which they arrived
  - Easy to implement!
  - May involve lots of unnecessary seek distance
- Seek order = 100, 175, 51, 133, 8, 140, 73, 77
- Seek distance = (100-63) + (175-100) + (175-51) + (133-51) + (133-8) + (140-8) + (140-73) + (77-73) = 646 cylinders



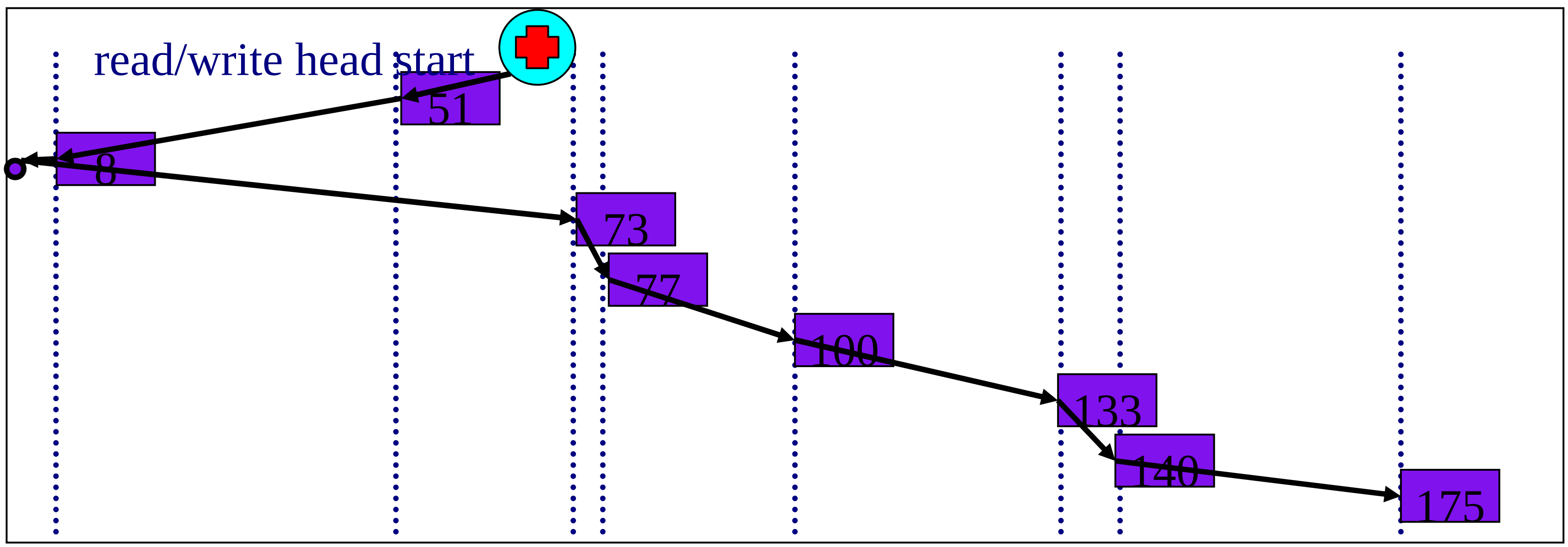
Shortest Seek Time First (SSTF)

- Service the request with the shortest seek time from the current head position
  - Form of SJF scheduling
  - May starve some requests
- Seek order = 73, 77, 51, 8, 100, 133, 140, 175
- Seek distance = 10 + 4 + 26 + 43 + 92 + 33 + 7 + 35 = 250 cylinders



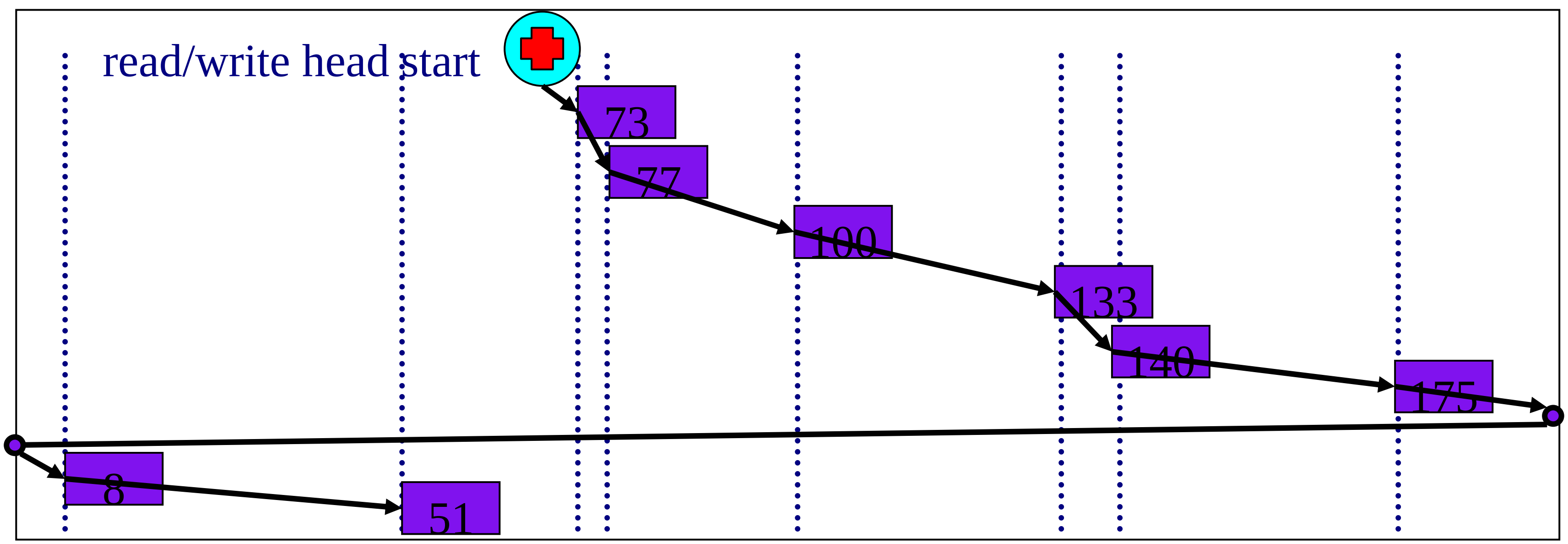
SCAN (elevator algorithm)

- Disk arm starts at one end of the disk and moves towards the other end, servicing requests as it goes
  - Reverses direction when it gets to end of the disk
  - Also known as elevator algorithm
- Seek order = 51, 8, 0 , 73, 77, 100, 133, 140, 175
- Seek distance = 12 + 43 + 8 + 73 + 4 + 23 + 33 + 7 + 35 = 238 cyls



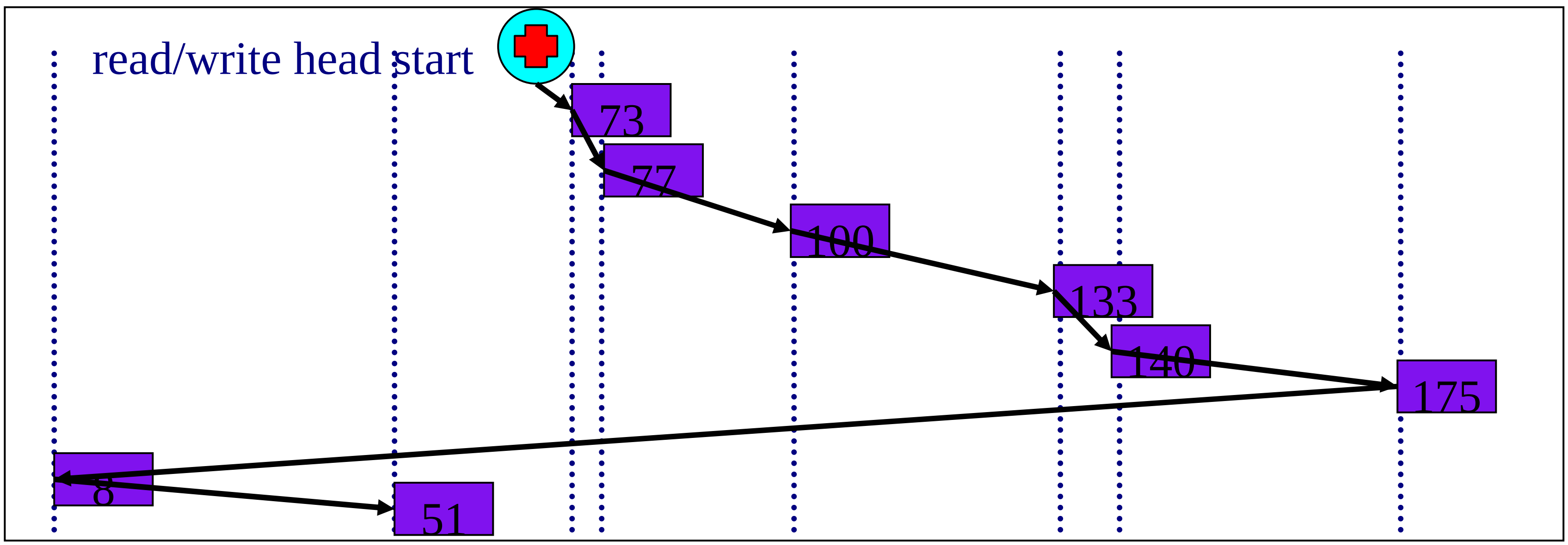
C-SCAN

- Identical to SCAN, except head returns to cylinder 0 when it reaches the end of the disk
  - Treats cylinder list as a circular list that wraps around the disk
  - Waiting time is more uniform for cylinders near the edge of the disk
- Seek order = 73, 77, 100, 133, 140, 175, 199, 0, 8, 51
- Distance = 10 + 4 + 23 + 33 + 7 + 35 + 24 + 199 + 8 + 43 = 386 cyls



C-LOOK

- Identical to C-SCAN, except head only travels as far as the last request in each direction
  - Saves seek time from last sector to end of disk
- Seek order = 73, 77, 100, 133, 140, 175, 8, 51
- Distance = 10 + 4 + 23 + 33 + 7 + 35 + 167 + 43 = 322 cylinders



How to pick a disk scheduling algorithm

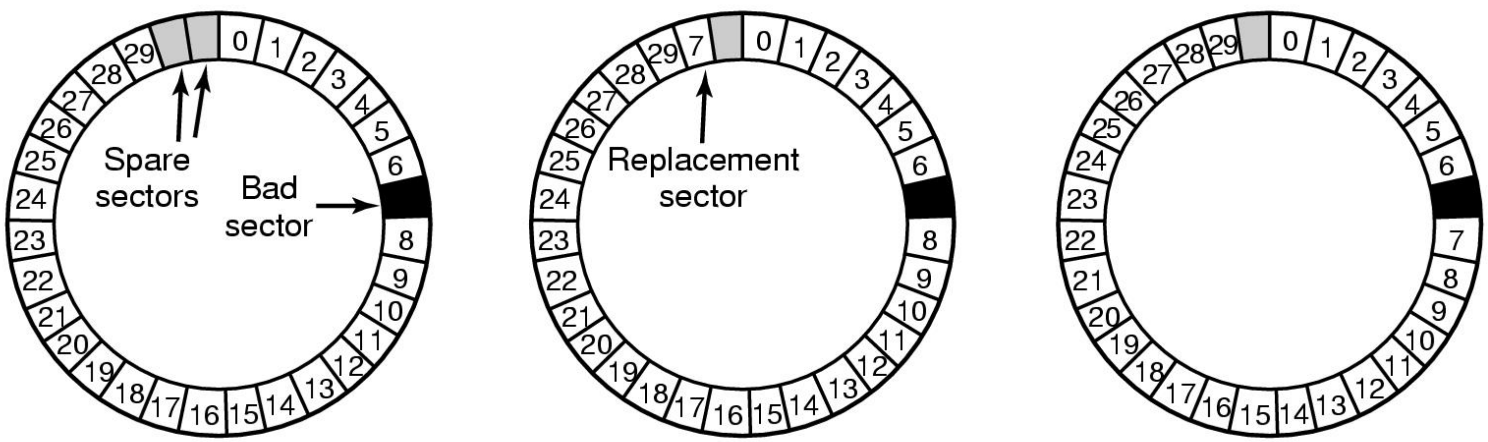
- SSTF is easy to implement and works OK if there aren't too many disk requests in the queue
- SCAN-type algorithms perform better for systems under heavy load
  - More fair than SSTF
  - Use LOOK rather than SCAN algorithms to save time
- Long seeks aren't too expensive, so choose C-LOOK over LOOK to make response time more even
- Disk request scheduling interacts with algorithms for allocating blocks to files
  - Make scheduling algorithm modular: allow it to be changed without changing the file system

⇒ Use SSTF for lightly loaded systems

⇒ Use C-LOOK for heavily loaded systems

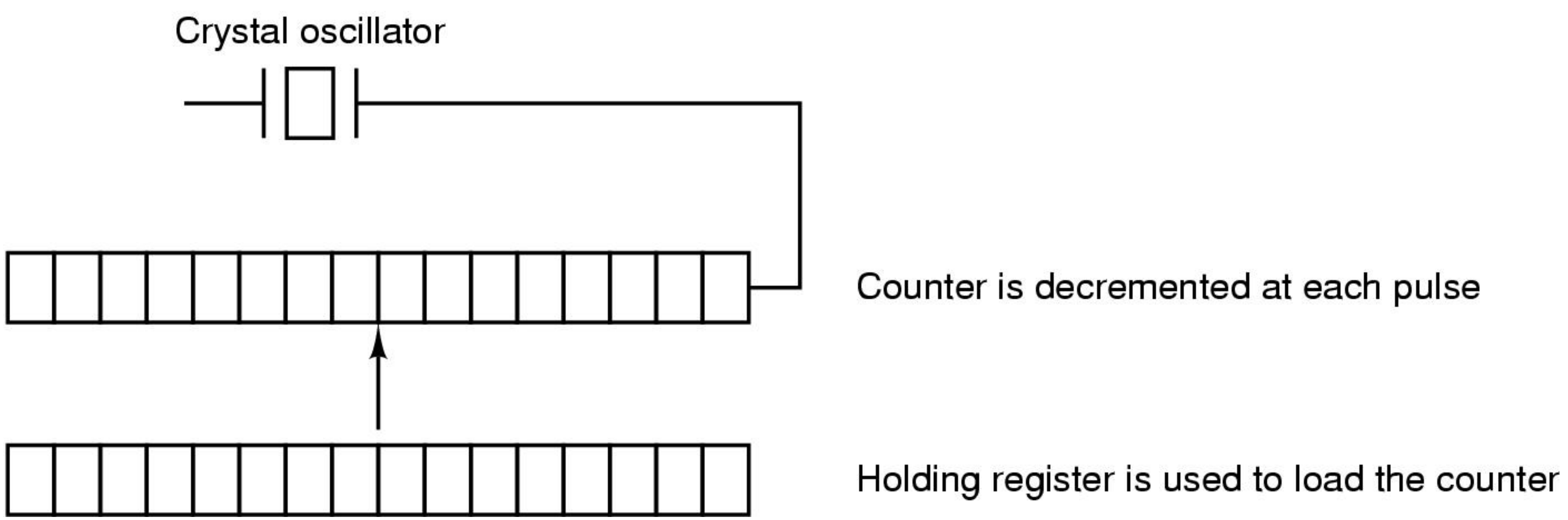
When good disks go bad…

- Disks have defects
  - In 3M+ sectors, this isn't surprising!
- ECC helps with errors, but sometimes this isn't enough
- Disks keep spare sectors (normally unused) and remap bad sectors into these spares
  - If there's time, the whole track could be reordered…

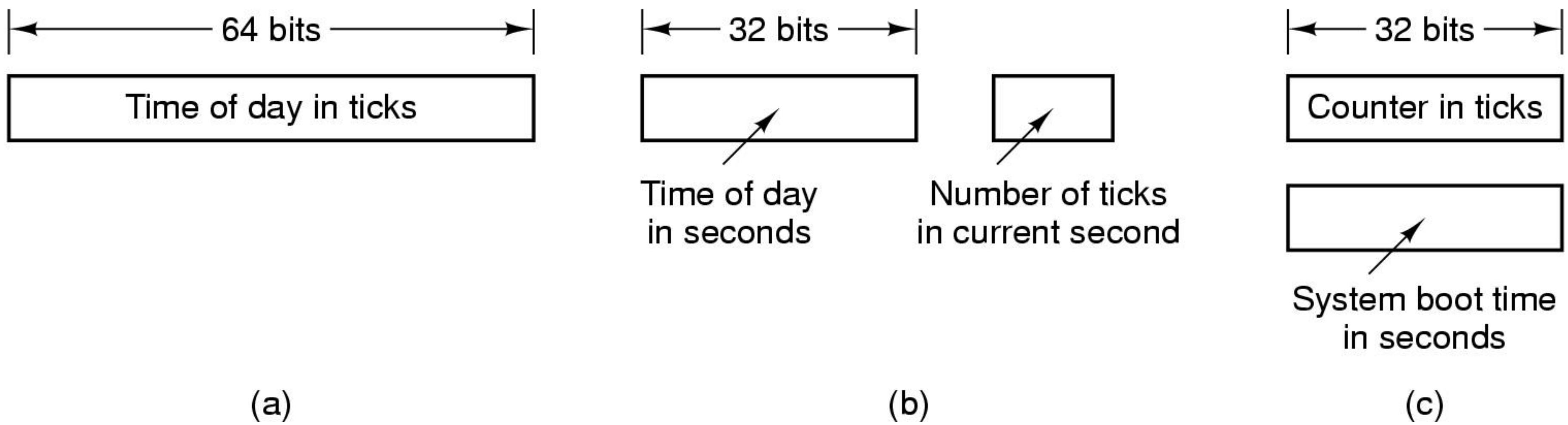


(a) (b) (c)

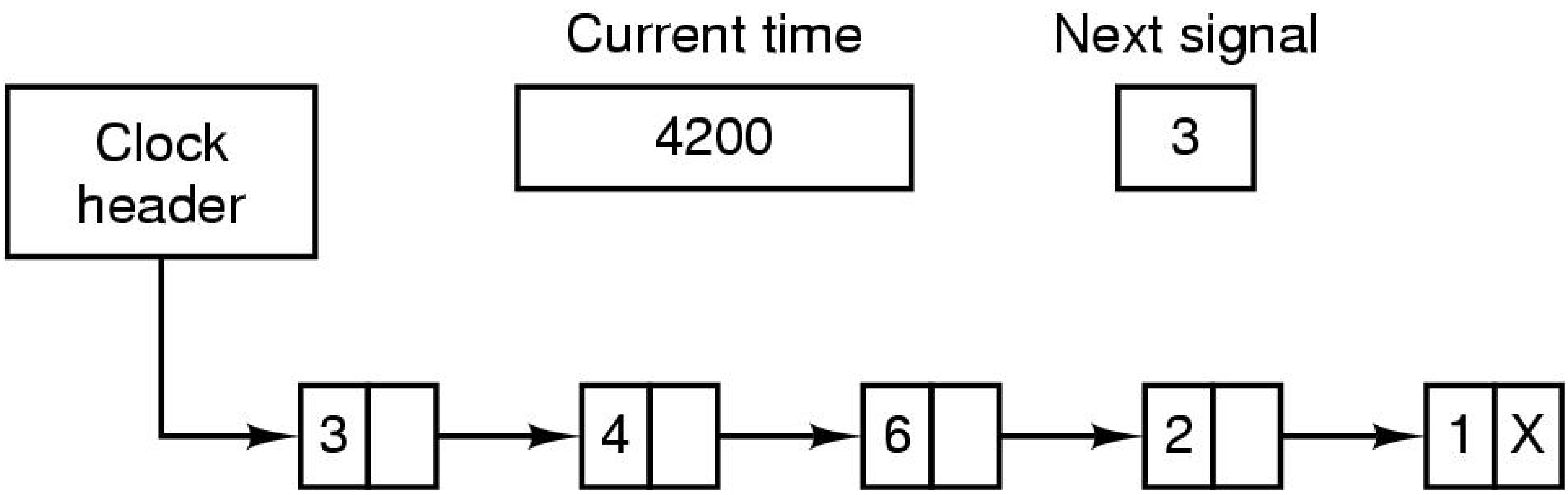
Clock hardware



Maintaining time of day



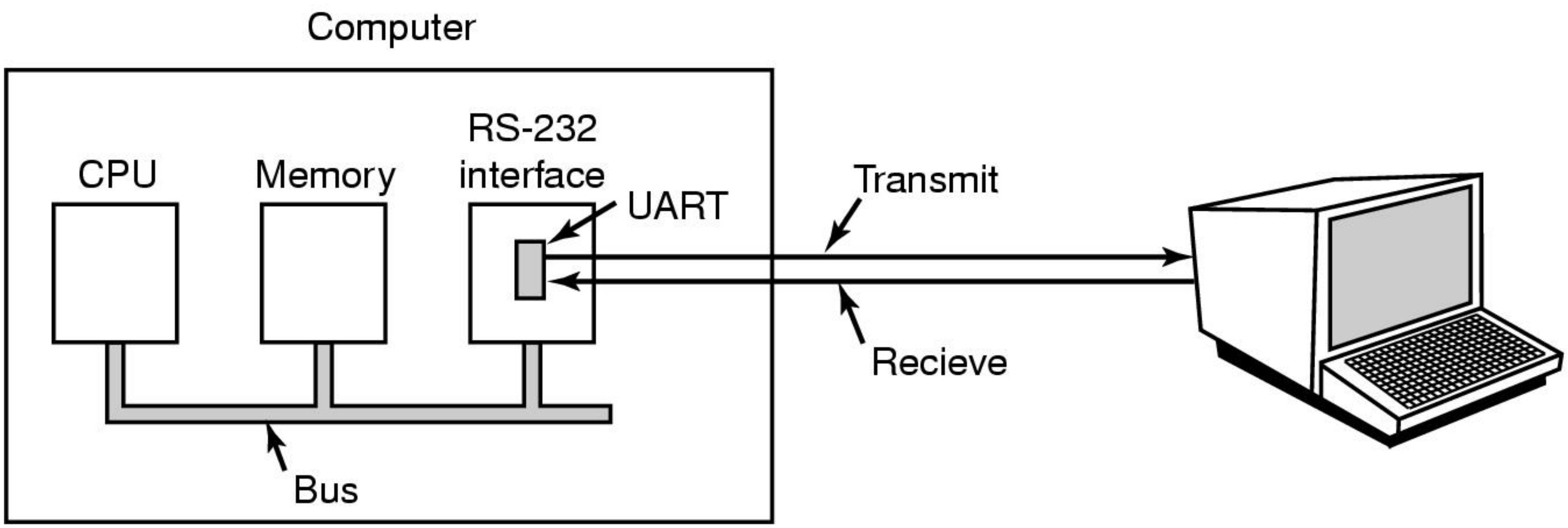
Doing multiple timers with a single clock



Soft timers

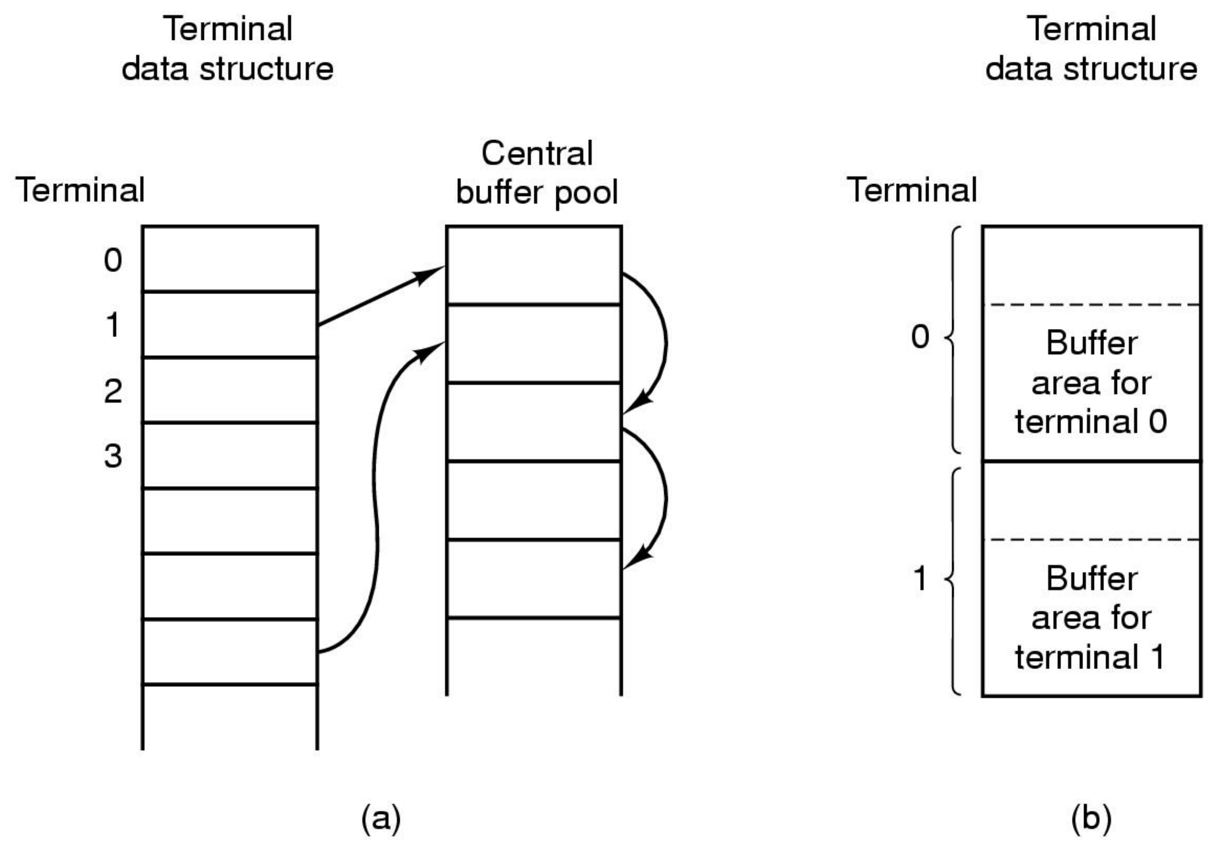
- A second clock may be available for timer interrupts
  - Specified by applications
  - No problems if interrupt frequency is low
- Soft timers avoid interrupts
  - Kernel checks for soft timer expiration before it exits to user mode
  - How well this works depends on rate of kernel entries

Character-oriented terminals



- An RS-232 terminal communicates with computer 1 bit at a time
- Called a serial line – bits go out in series, 1 bit at a time
- Windows uses COM1 and COM2 ports, first to serial lines
- Computer and terminal are completely independent

Buffering for input



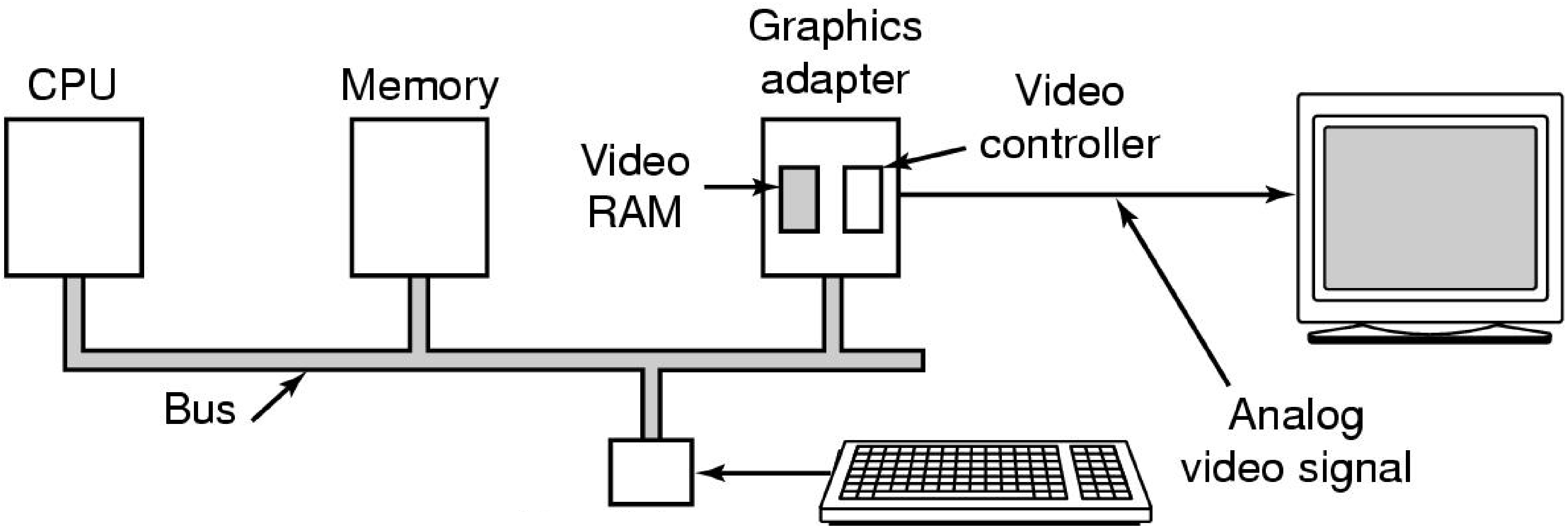
Special terminal characters

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)

Special output characters

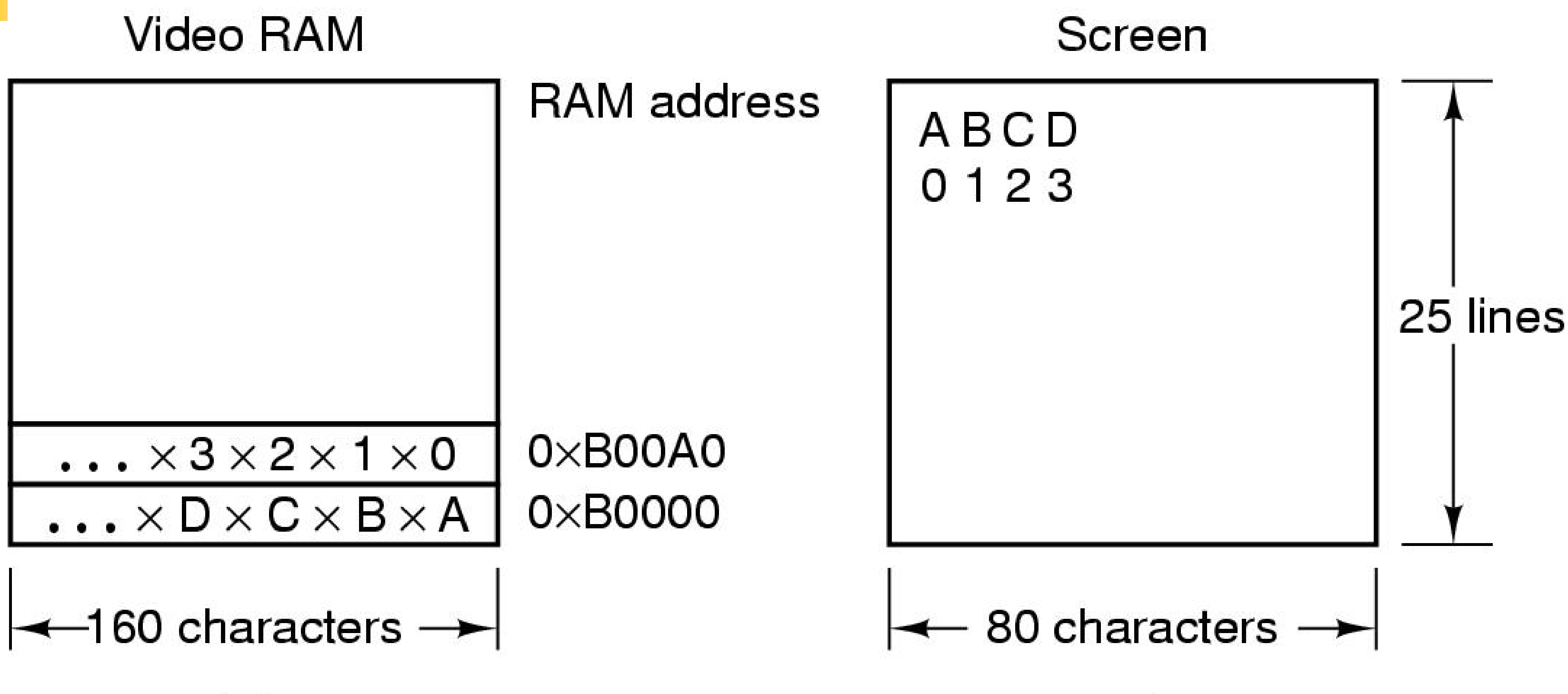
Escape sequence	Meaning
ESC [*n*A	Move up *n* lines
ESC [*n*B	Move down *n* lines
ESC [*n*C	Move right *n* spaces
ESC [*n*D	Move left *n* spaces
ESC [*m* ; *n*H	Move cursor to (*m*,*n*)
ESC [*s*J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [*s*K	Clear line from cursor (0 to end, 1 from 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

Memory-mapped display



Driver writes directly into display's video RAM

How characters are displayed

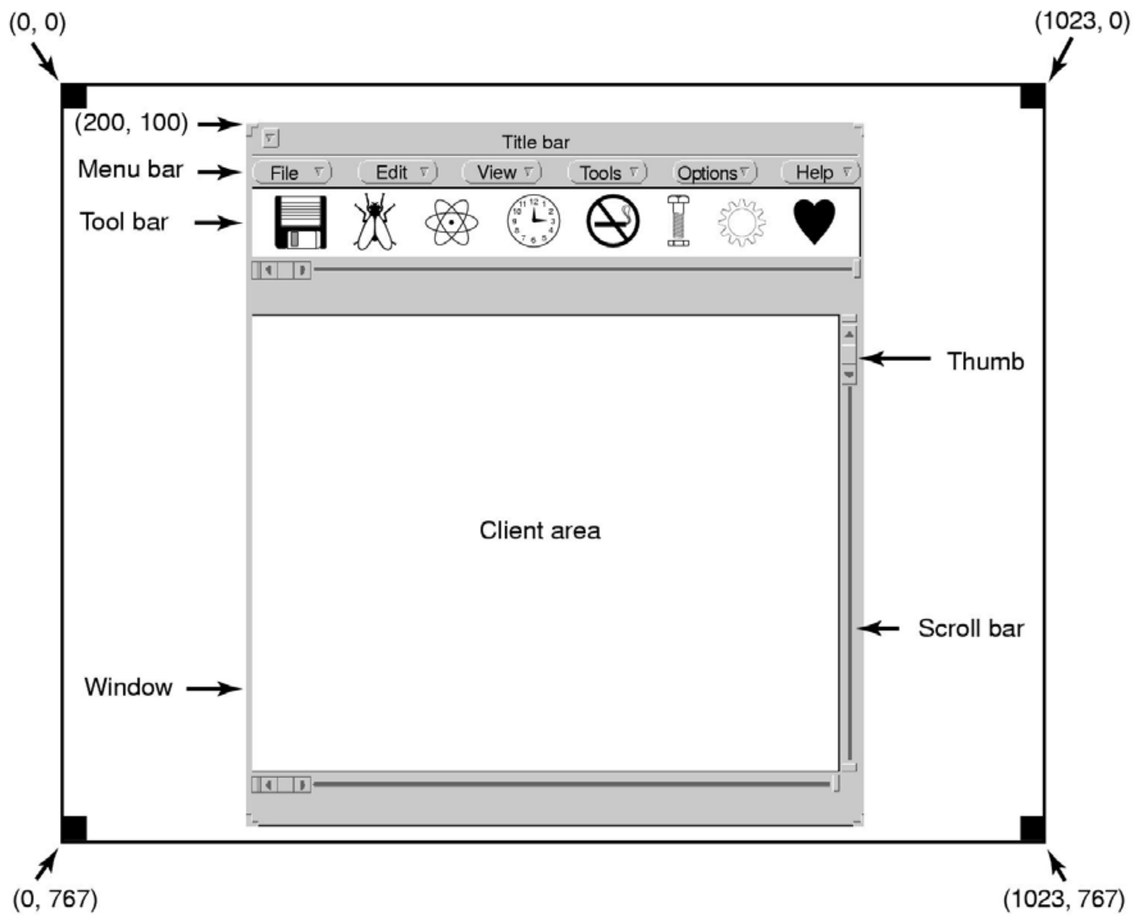


- A video RAM image
  - simple monochrome display
  - character mode
- Corresponding screen
  - the Xs are attribute bytes

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
  - Example: characters such as ç

Output software for Windows



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

Skeleton of a Windows program

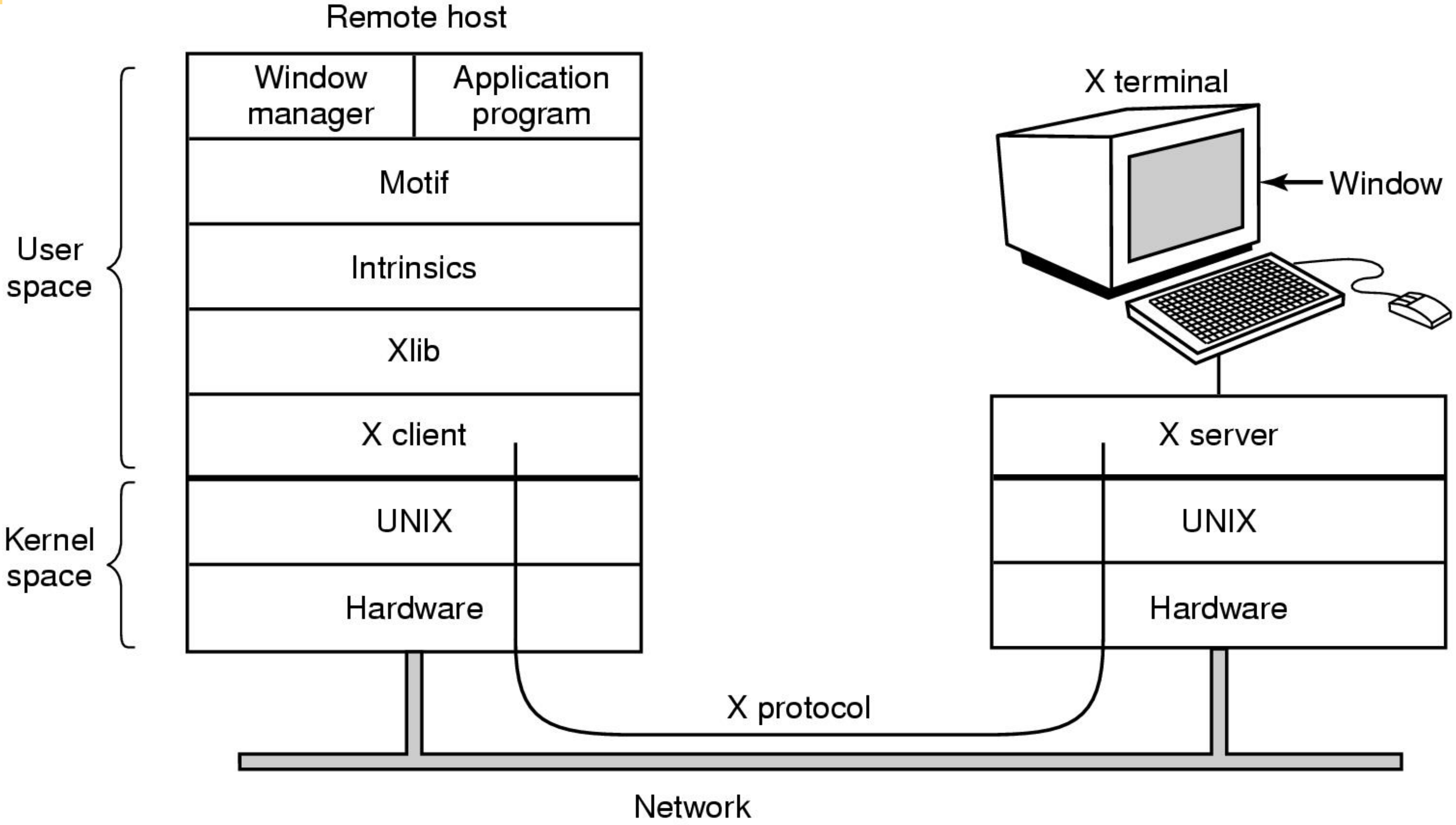
```
#include <windows.h>

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 program (cont'd)

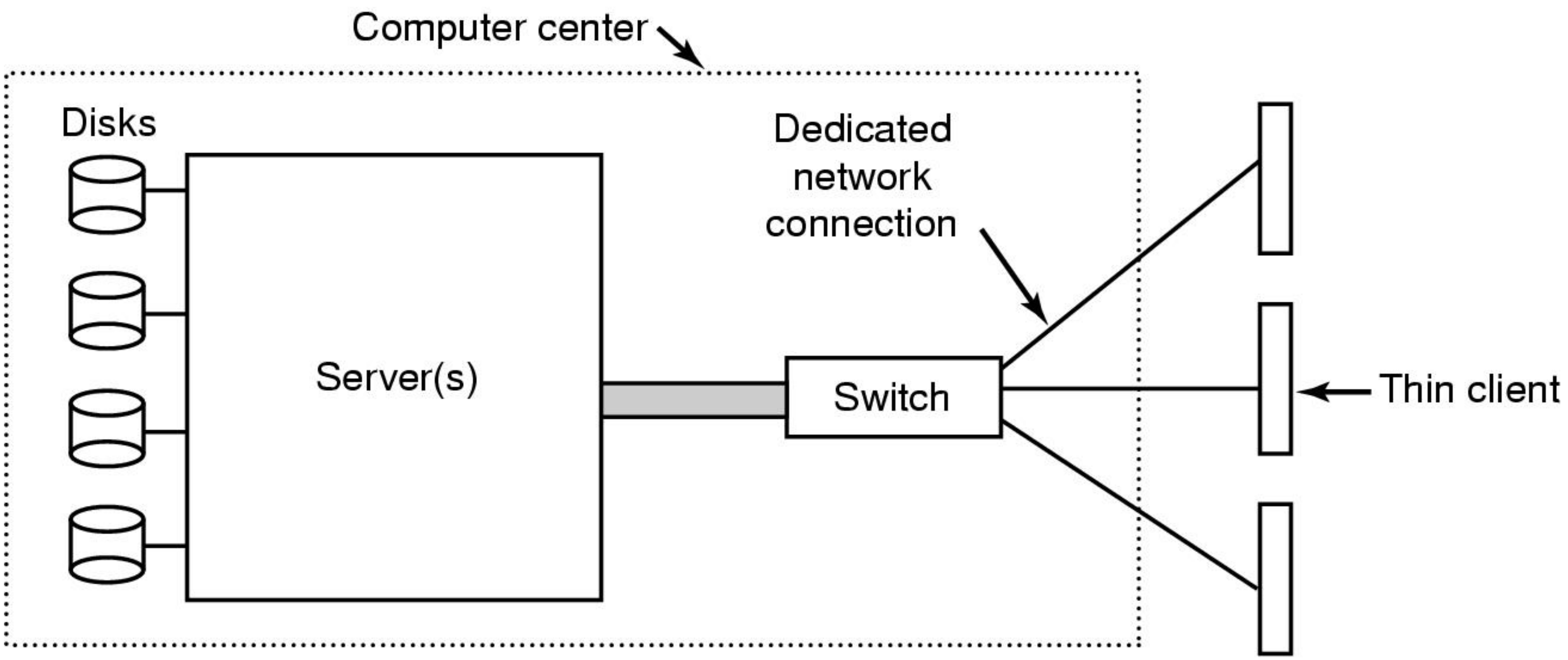
```
    while (GetMessage(&msg, NULL, 0, 0)) {    /* get message from queue */
        TranslateMessage(&msg);    /* translate the message */
        DispatchMessage(&msg);     /* send msg to the appropriate procedure */
    }
    return(msg.wParam);
}

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 */
}
```

Character outlines at different point sizes

X Windows


Remote host
User space
Kernel space
Window manager
Application program
Motif
Intrinsics
Xlib
X client
UNIX
Hardware
X terminal
Window
X server
UNIX
Hardware
X protocol
Network

Architecture of the SLIM terminal system



The SLIM Network Terminal

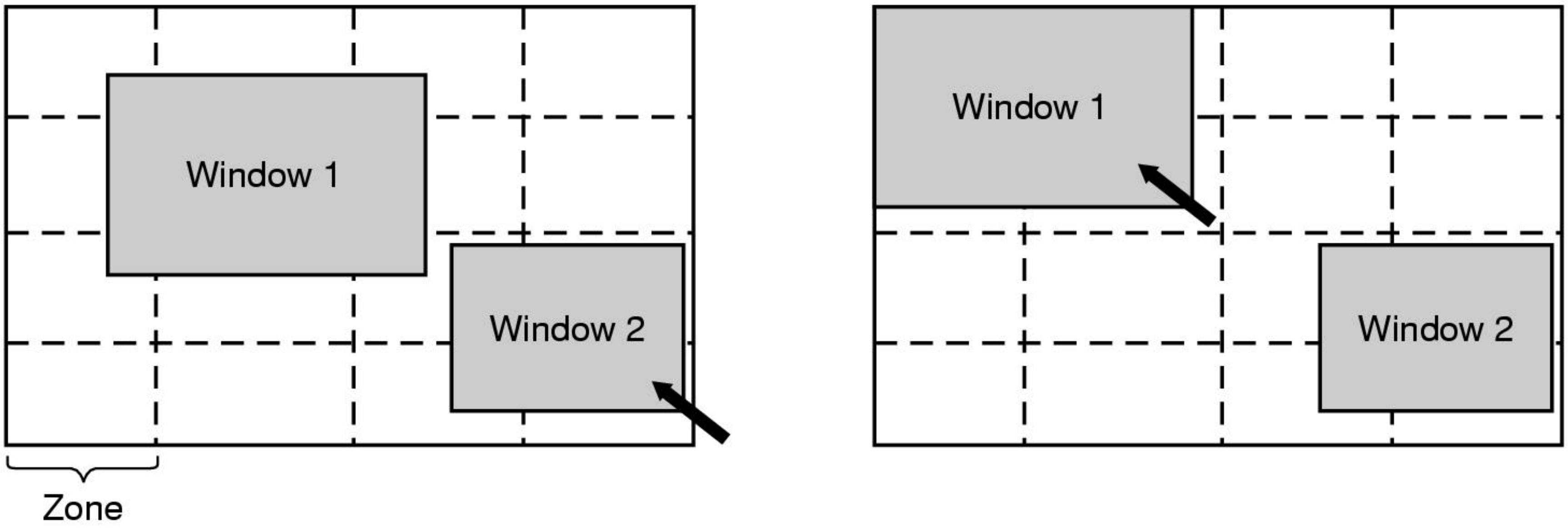
Message	Meaning
SET	Update a rectangle with new pixels
FILL	Fill a rectangle with one pixel value
BITMAP	Expand a bitmap to fill a rectangle
COPY	Copy a rectangle from one part of the frame buffer to another
CSCS	Convert a rectangle from television color (YUV) to RGB

Power Management (1)

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%

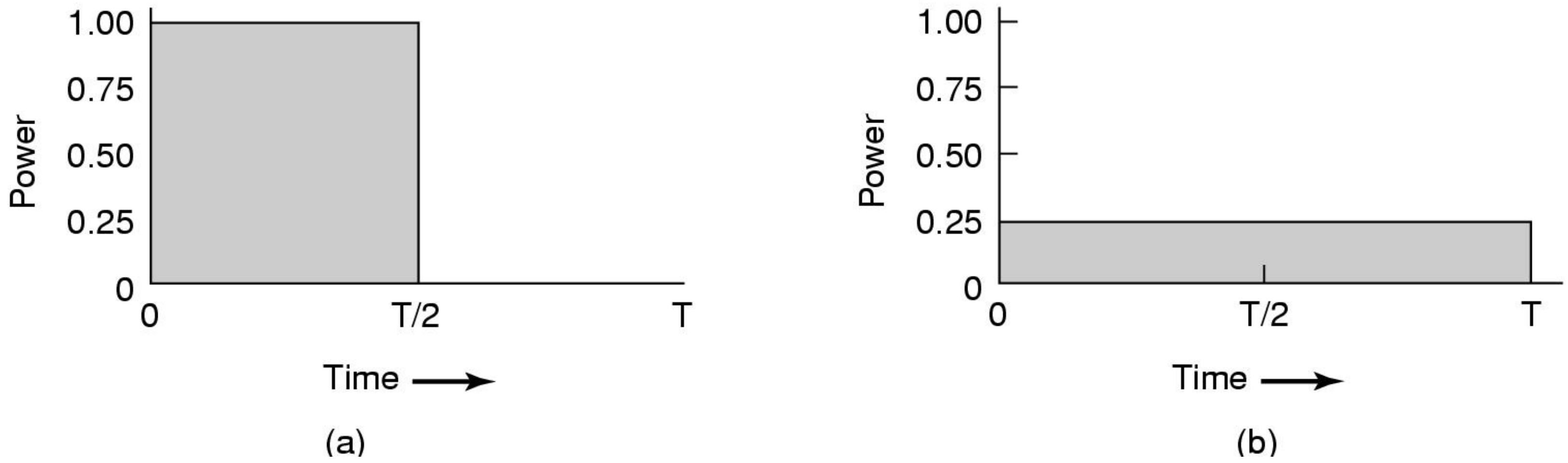
Power consumption of various parts of a laptop computer

Power management (2)



The use of zones for backlighting the display

Power Management (3)



- Running at full clock speed
- Cutting voltage by two
  - cuts clock speed by two,
  - cuts power by four

Power Management (4)

- Telling the programs to use less energy
  - may mean poorer user experience
- Examples
  - change from color output to black and white
  - speech recognition reduces vocabulary
  - less resolution or detail in an image