Chapter 5

Input / Output

The Spectrum of I/O Devices

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

Device Controllers

The Device vs. its Controller

Some duties of a device controller:

- Interface between CPU and the Device
- Start/Stop device activity
- Convert serial bit stream to a block of bytes
- Deal with errors
  - Detection / Correction
- Move data to/from main memory

Some controllers may handle several (similar) devices

Approaches to I/O

Each port has a separate number.

CPU has special commands

```
in    r4,3
out   3,r4
```

The I/O Port Number

Port numbers form an "address space"
...separate from main memory

Contrast with

```
load    r4,3
store   3,r4
```

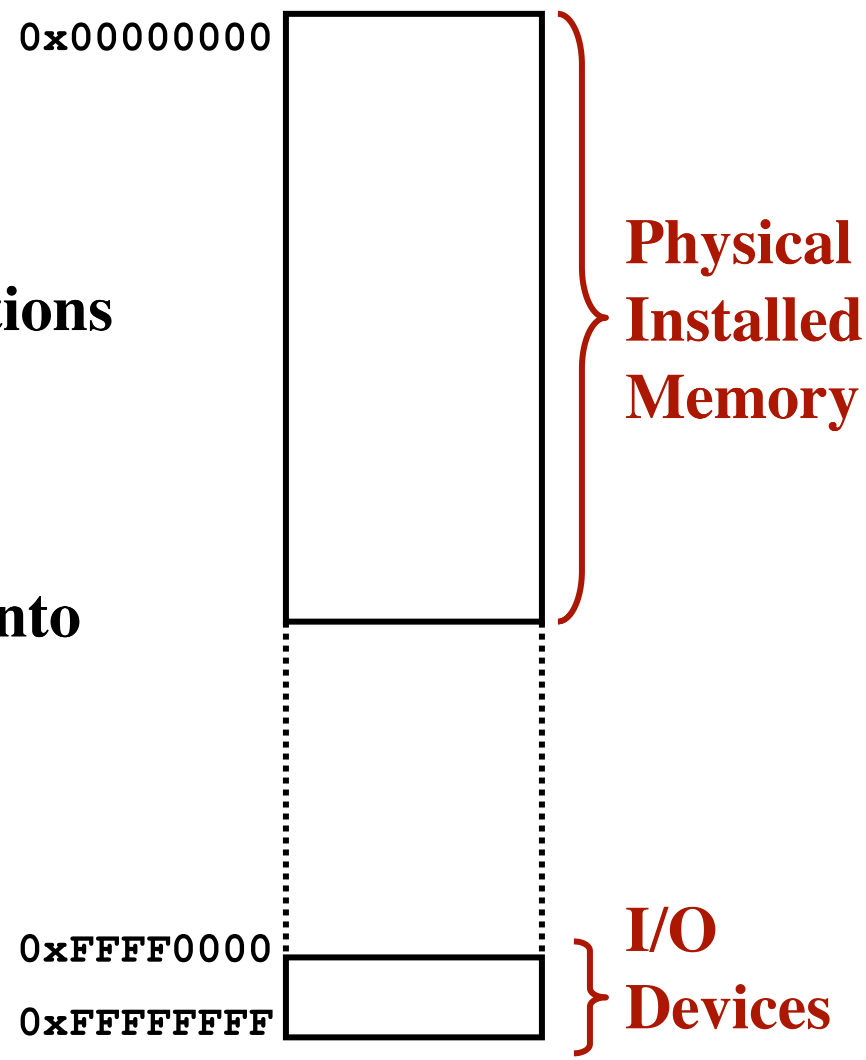
Memory-Mapped I/O

One address space for
- main memory
- I/O devices

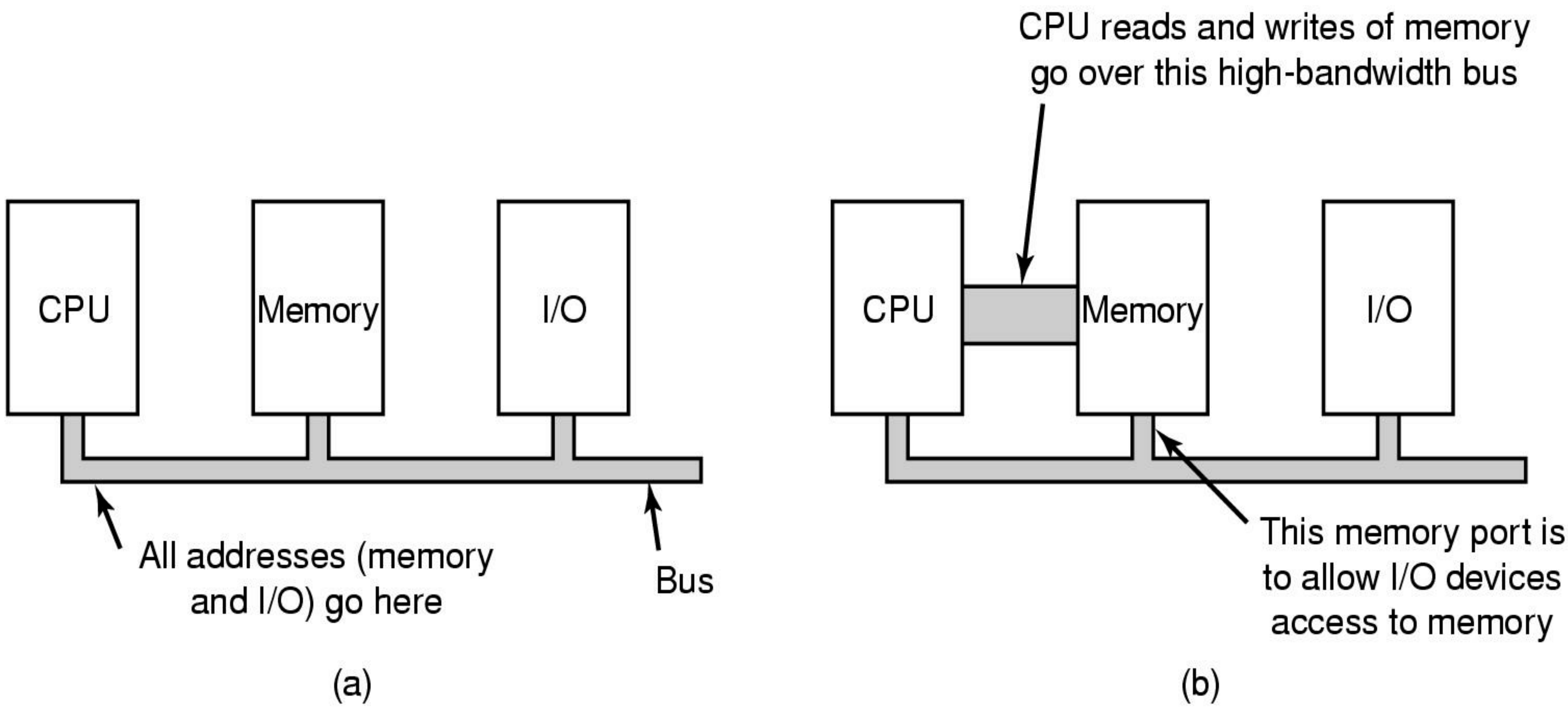
CPU has no special instructions

```
load     r4,3
store    3,r4
```

I/O devices are "mapped" into very high addresses



Single vs. Dual Bus



Direct Memory Access (DMA)

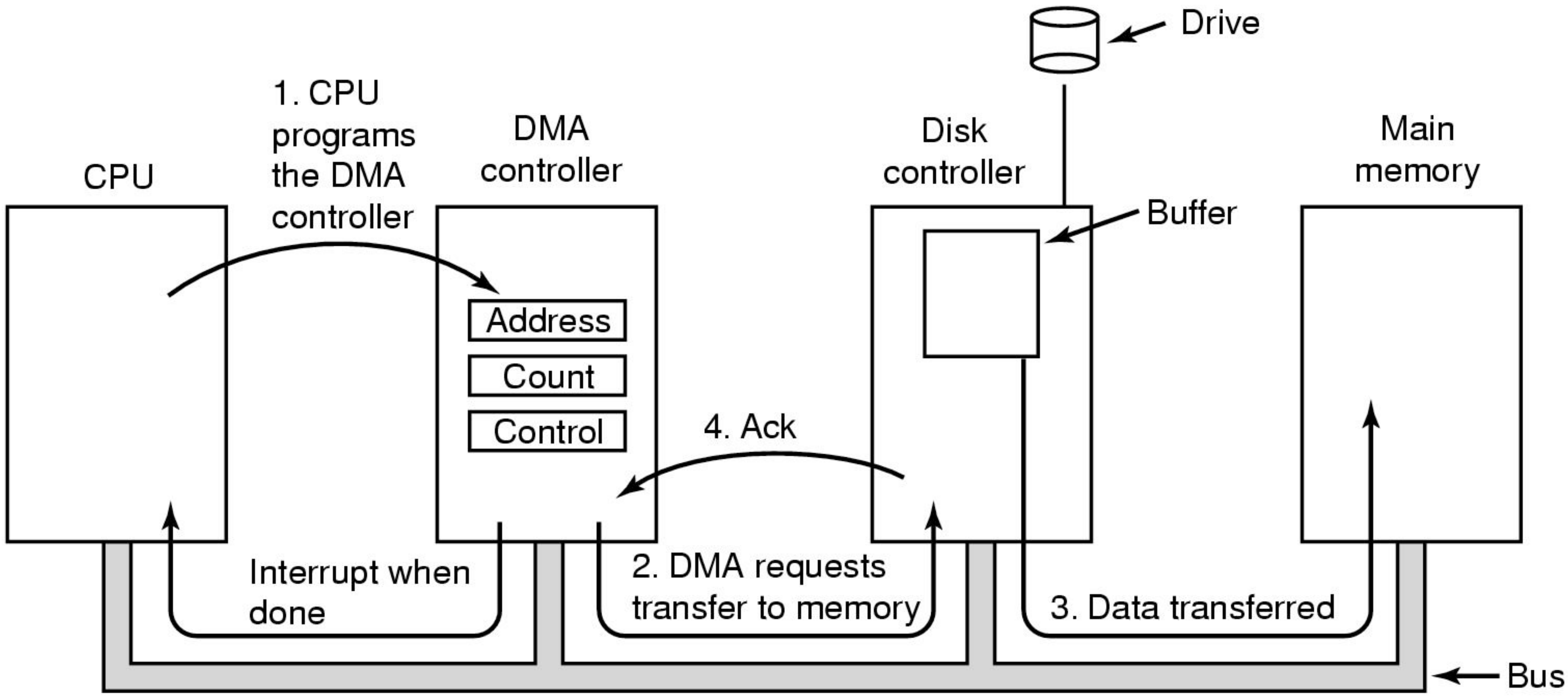
Data transferred from device straight to/from memory.

CPU not involved.

The DMA controller:

- **Does the work of moving the data**
- **CPU sets up the DMA controller ("programs it")**
- **CPU continues**
- **The DMA controller moves the bytes**

DMA


Drive
1. CPU programs the DMA controller
CPU
DMA controller
Disk controller
Main memory
Buffer
Address
Count
Control
4. Ack
Interrupt when done
2. DMA requests transfer to memory
3. Data transferred
Bus

Direct Memory Access (DMA)

Cycle Stealing

DMA Controller acquires control of bus
Transfers a single byte (or word)
Releases the bus
The CPU is slowed down

Burst Mode

DMA Controller acquires control of bus
Transfers all the data
Releases the bus
The CPU operation is suspended

Direct Memory Access (DMA)

Cycle Stealing

DMA Controller acquires control of bus

Transfers a single byte (or word)

Releases the bus

The CPU is slowed down

Not as efficient

Burst Mode

DMA Controller acquires control of bus

Transfers all the data

Releases the bus

The CPU operation is suspended

The CPU may not service interrupts in a timely way

Principles of I/O Software

Device Independence

Programs can access any I/O device

Hard Drive, CD-ROM, Floppy,...

... without specifying the device in advance

Uniform Naming

Devices / Files are named with simple strings

Names should not depend on the device

Error Handling

...should be as close to the hardware as possible

Principles of I/O Software

Synchronous vs. Asynchronous Transfers
- Process is blocked vs. Interrupt-driven approach

Buffering
- Data comes off a device
- Can’t know the final destination of the data
  - e.g., a network packet... Where to put it???

Sharable vs. Dedicated Devices
- Disk should be sharable
- Keyboard, Screen dedicated to one process

Programmed I/O

Example:

Writing a string to a serial output

Printing a string on the printer

```
CopyFromUser(kernelBuffer, virtAddr, byteCount)
for i = 0 to byteCount-1
    while *serialStatusReg != READY
    endWhile
    *serialDataReg = kernelBuffer[i]
endFor
return
```

"Busy Waiting"

"Polling"

Interrupt-Driven I/O

Getting the I/O started:

```
CopyFromUser(kernelBuffer, virtAddr, byteCount)
EnableInterrupts()
while *serialStatusReg != READY
endWhile
*serialDataReg = kernelBuffer[0]
Sleep ()
```

The Interrupt Handler:

```
if i == byteCount
  Wake up the user process
else
  *serialDataReg = kernelBuffer[i]
  i = i + 1
endIf
Return from interrupt
```

Sending data to a device using DMA

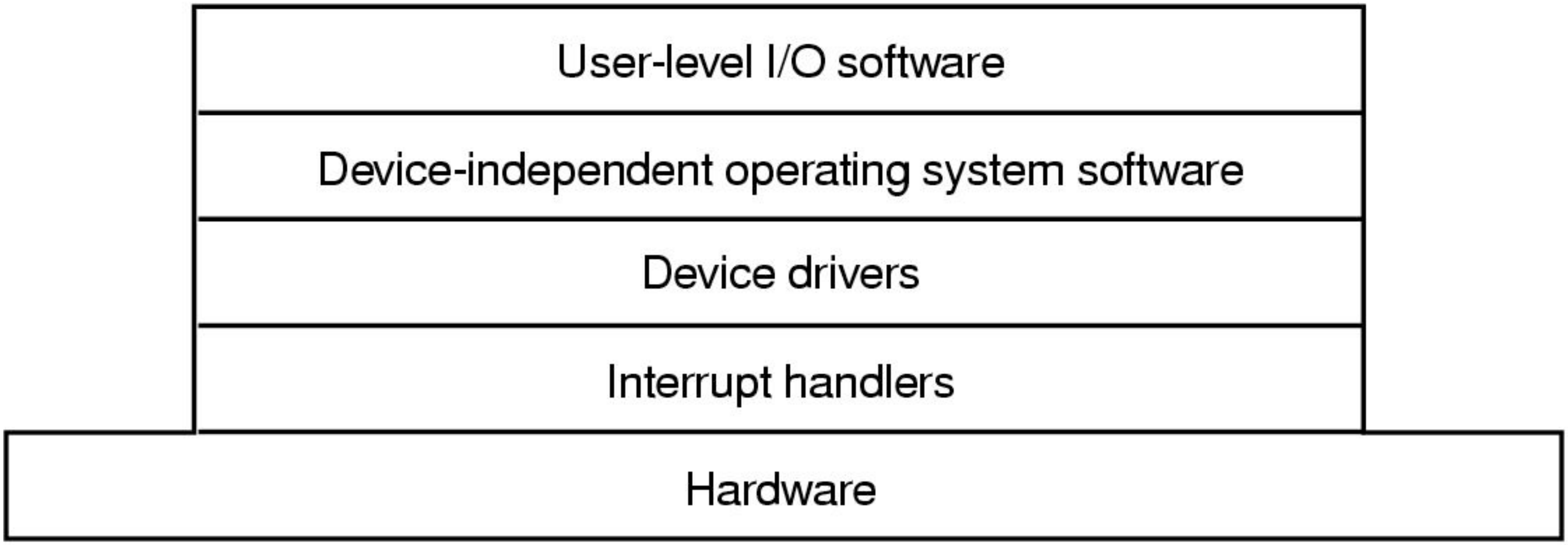
Getting the I/O started:

```
CopyFromUser(kernelBuffer, virtAddr, byteCount)
Set up DMA controller
Sleep ()
```

The Interrupt Handler:

```
Acknowledge interrupt
Wake up the user process
Return from interrupt
```

Layers of the I/O Software System



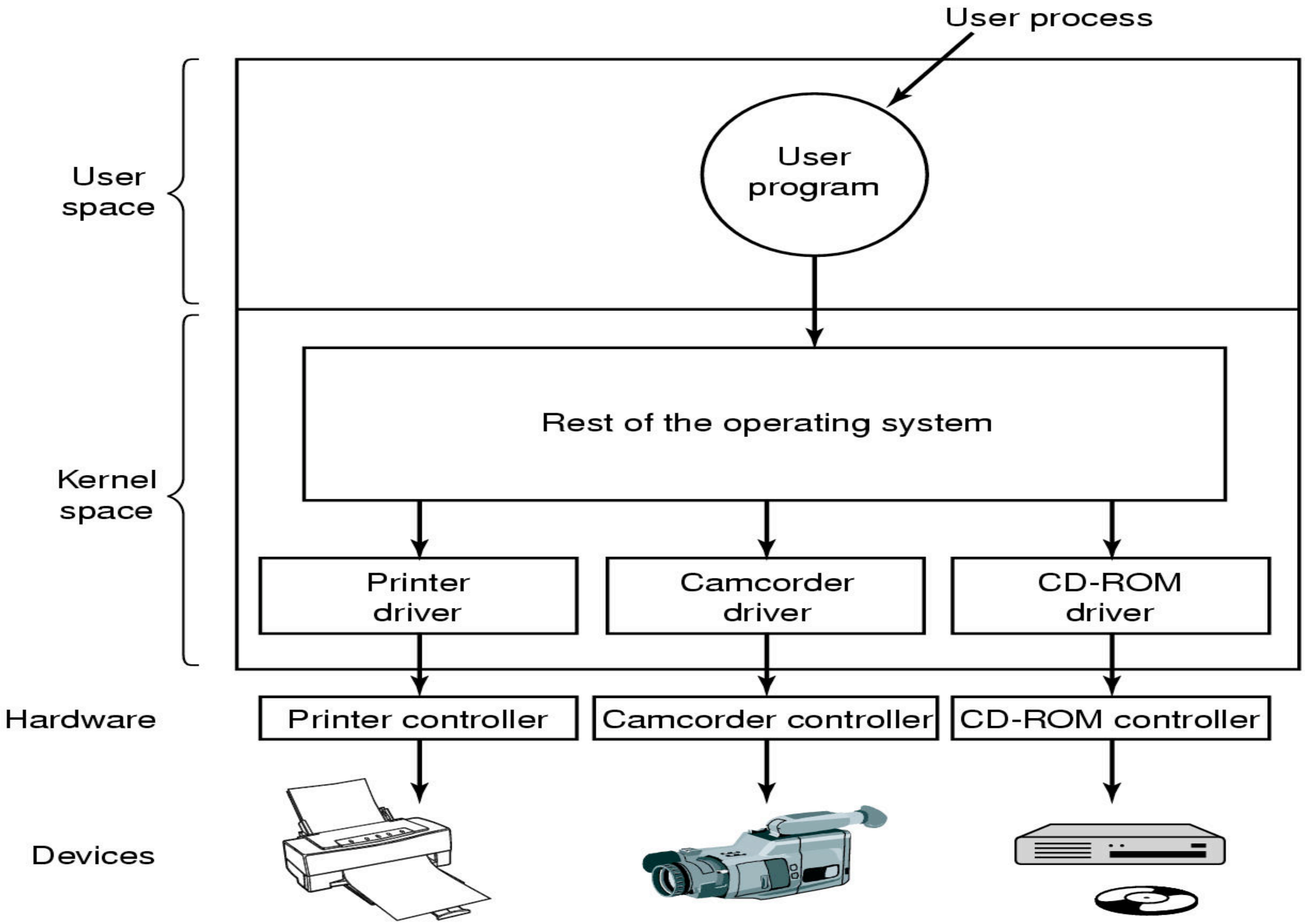
Interrupt Handling

Interrupt handlers are best hidden.
I/O Driver starts the operation
- Then blocks until an interrupt occurs
- Then it wakes up, finishes, & returns

The Interrupt Handler
- Does whatever is immediately necessary
- Then unblocks the driver

Example: The BLITZ "DiskDriver"
- Start I/O and block (waits on semaphore)
- Interrupt routine signals the semaphore & returns

Device Drivers

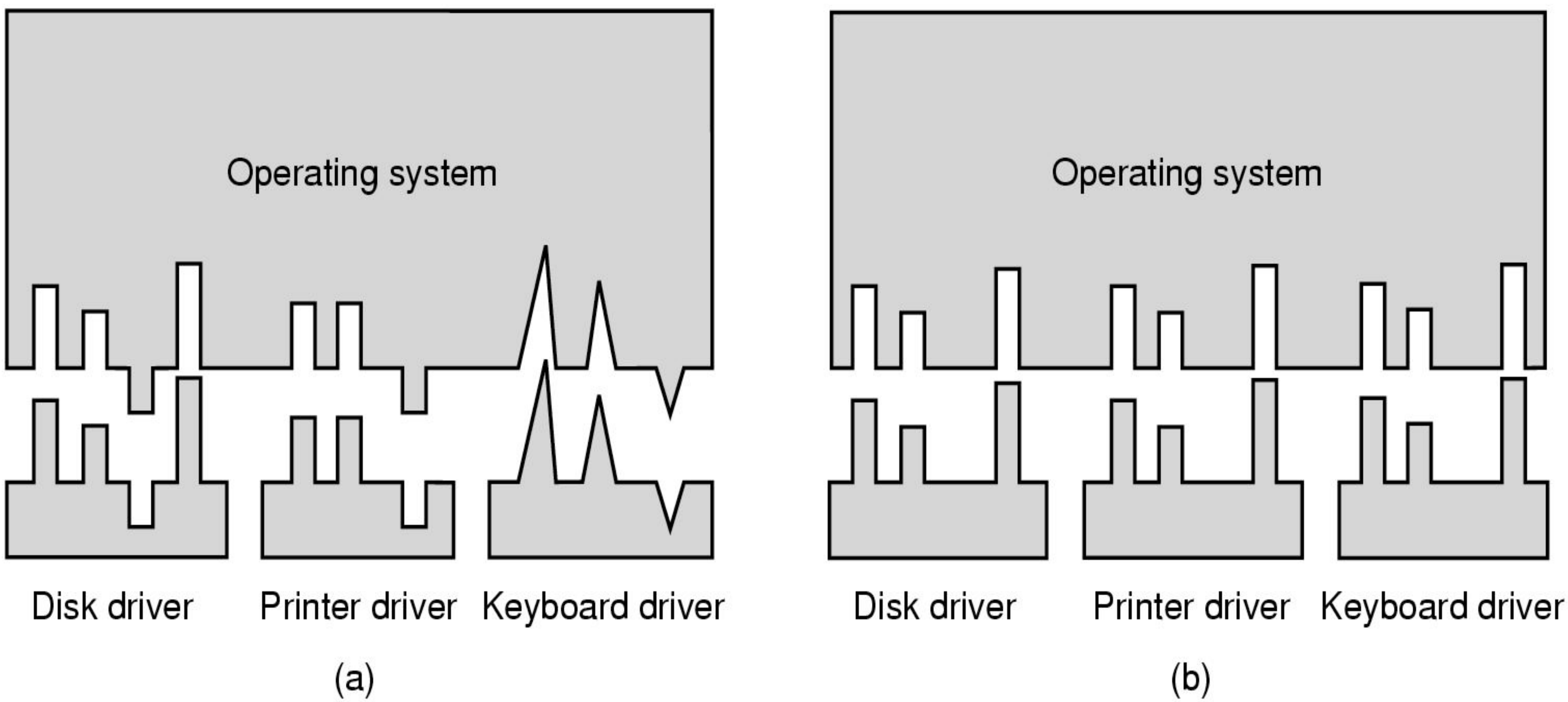


Device-Independent I/O Software

Functions and responsibilities:

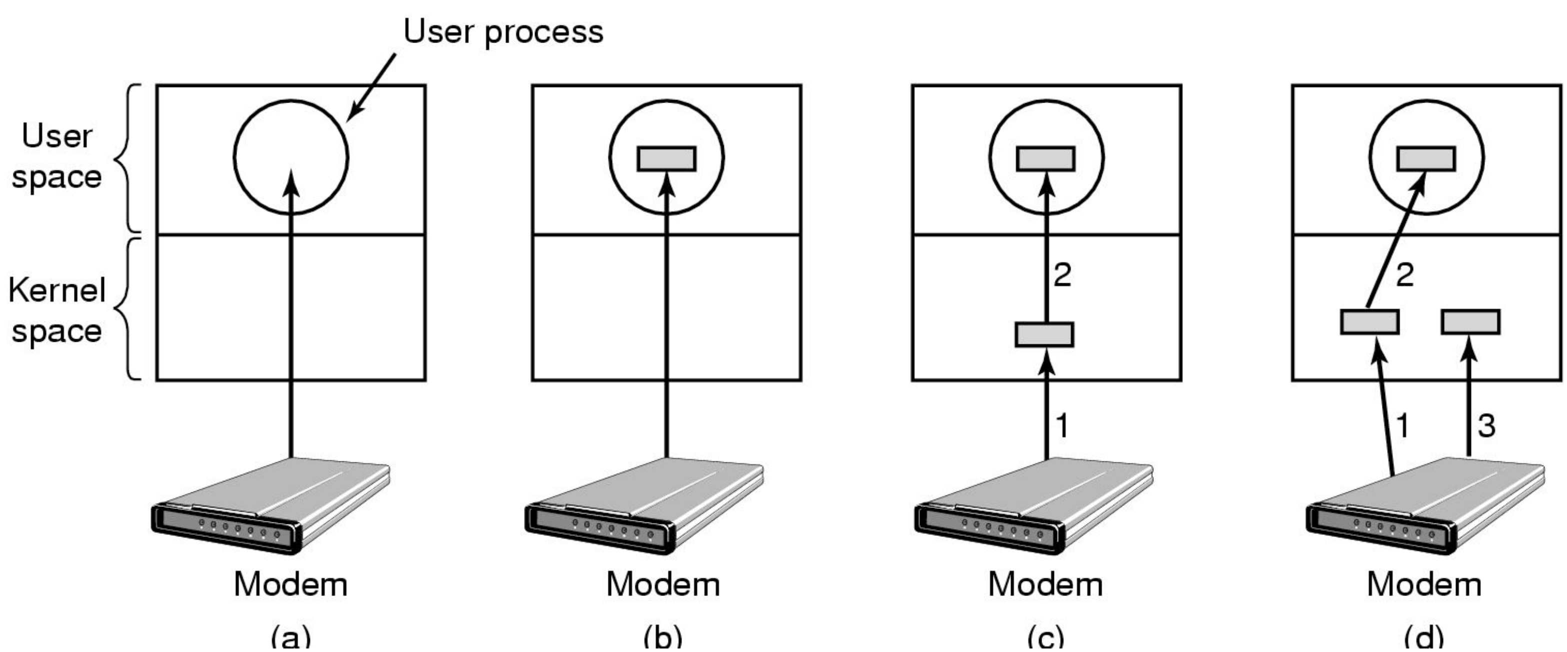
- **Uniform interfacing for device drivers**
- **Buffering**
- **Error reporting**
- **Allocating and releasing dedicated devices**
- **Providing a device-independent block size**

Device-Independent I/O Software



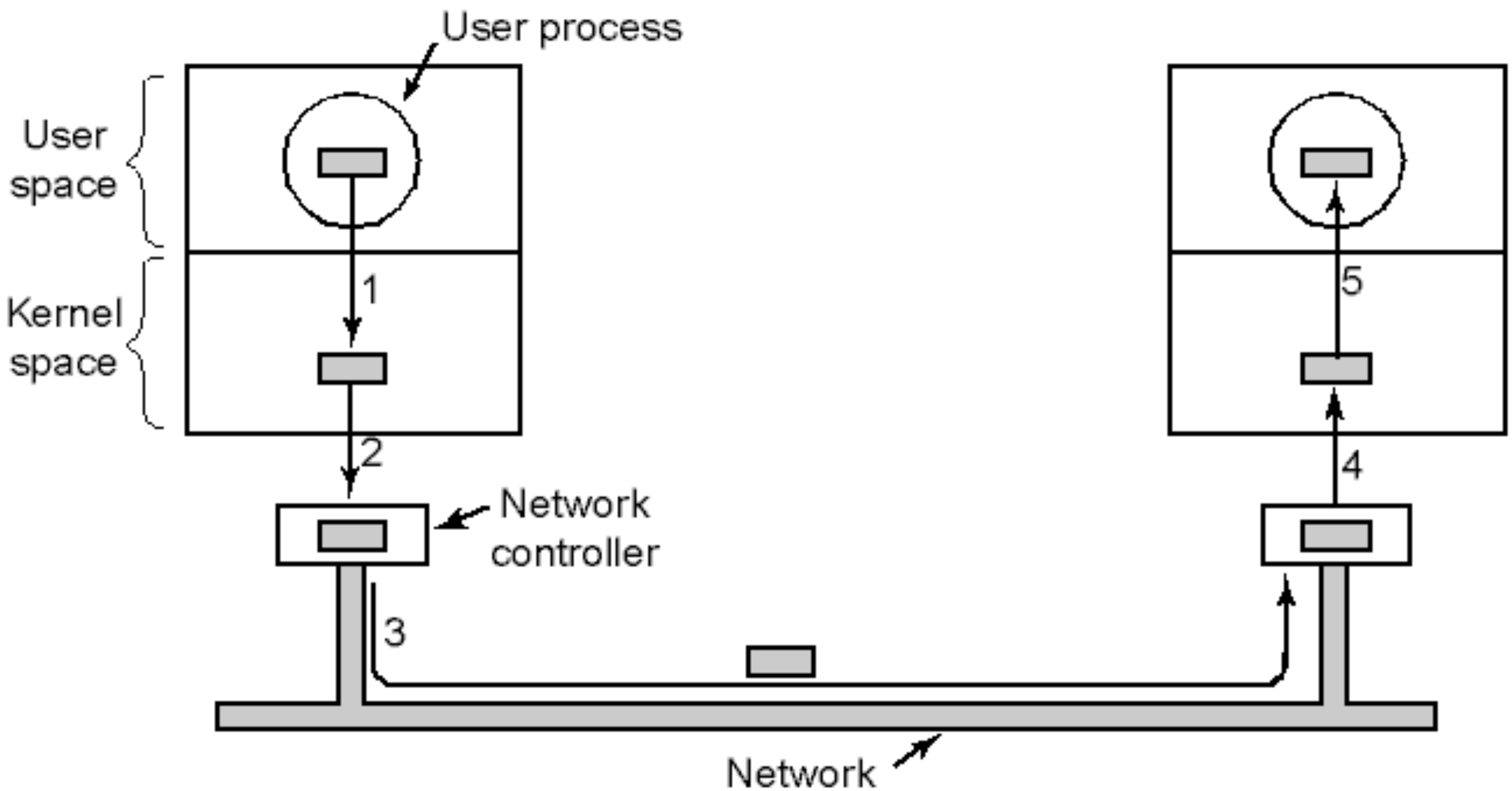
Issues with Buffers

(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

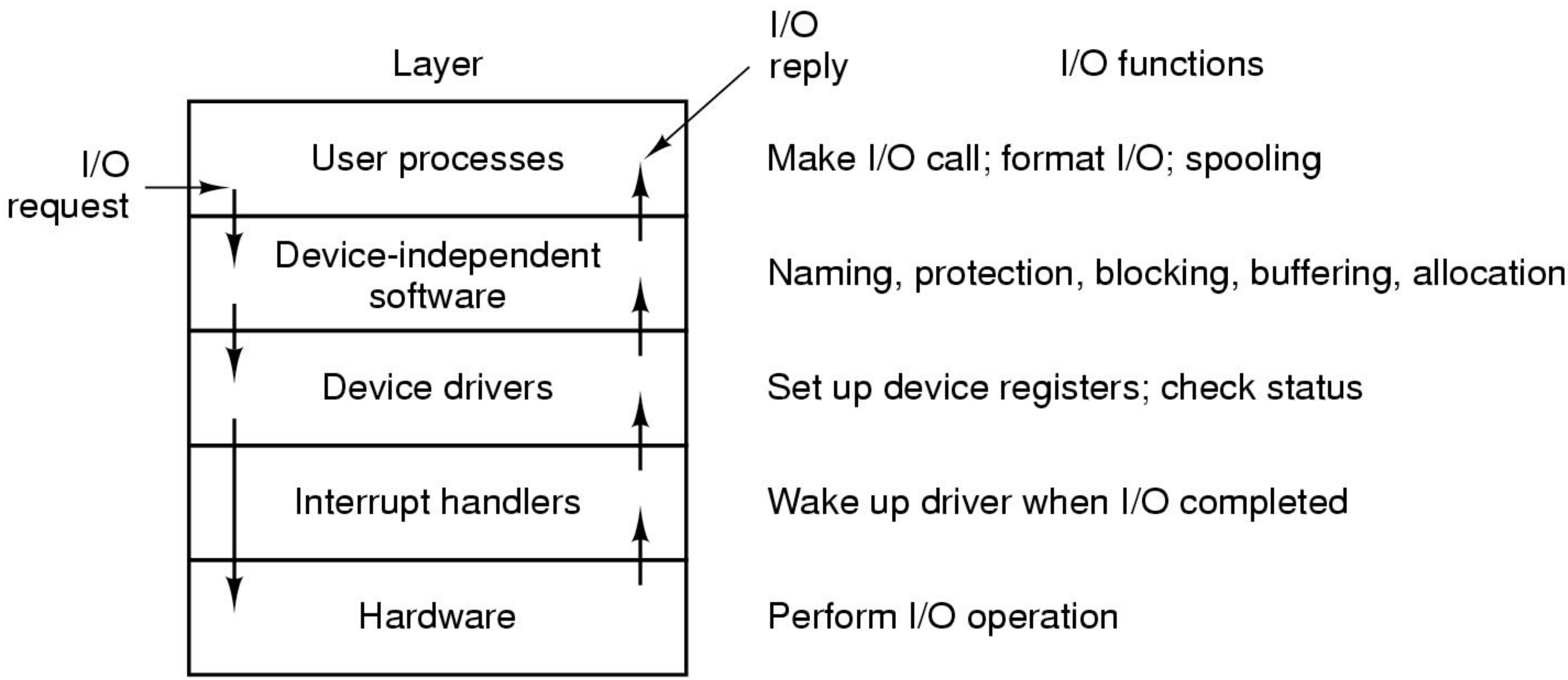


Issues with Buffers

Networking often involves lots of copying!



Layers within the I/O Subsystem



User-Space I/O Software

In user's (C) program

```
count = write (fd, buffer, nbytes);
printf ("The value of %s is %d\n", str, i);
```

Linked with library routines.

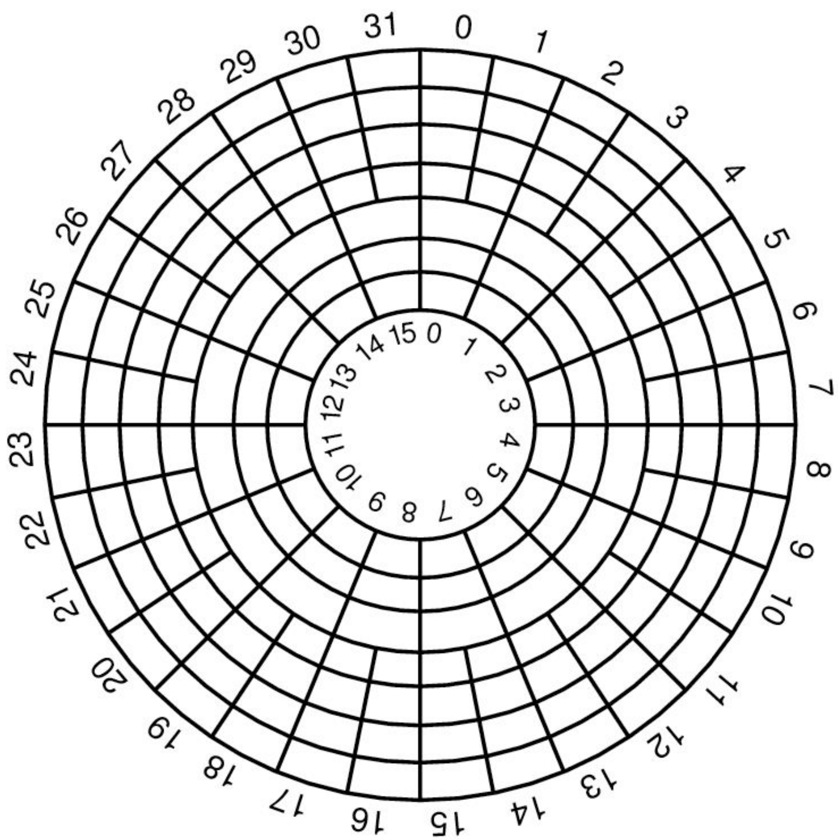
The library routines contain:

- **Lots of code**
  Buffering
- **The syscall to the kernel**

Comparison of Disk Technology

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 Zones



- Constant rotation speed
- Want constant bit density

Inner tracks:
 Fewer sectors per track
Outer tracks:
 More sectors per track

Disk Geometry

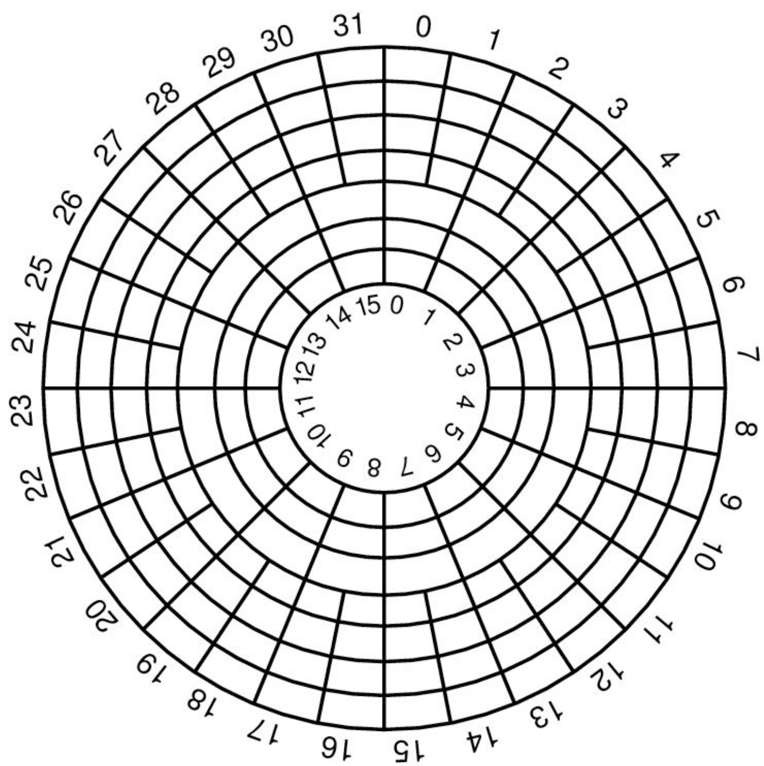
Physical Geometry

- **The actual layout of sectors on the disk**
- **May be complicated**

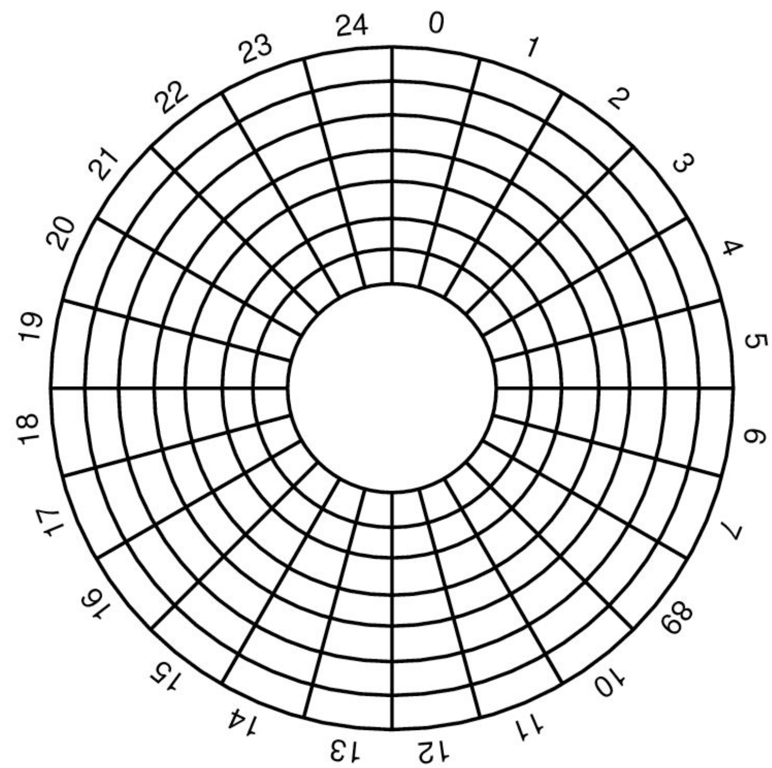
The controller does the translation

The CPU sees a “virtual geometry”.

Disk Geometry



physical geometry



virtual geometry

(192 sectors in each view)

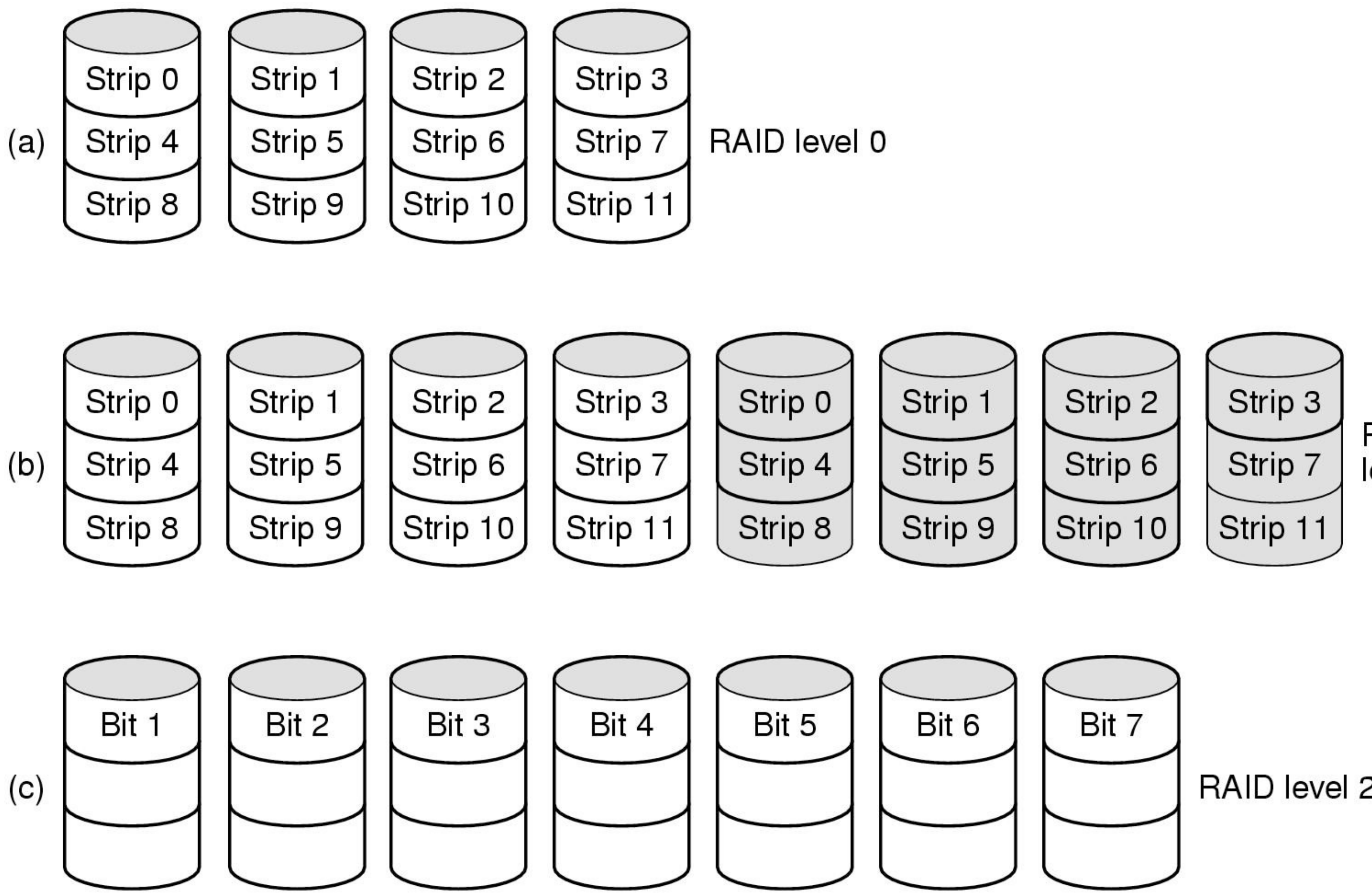
RAID

Redundant Array of Independent Disks
Redundant Array of Inexpensive Disks

Goals:

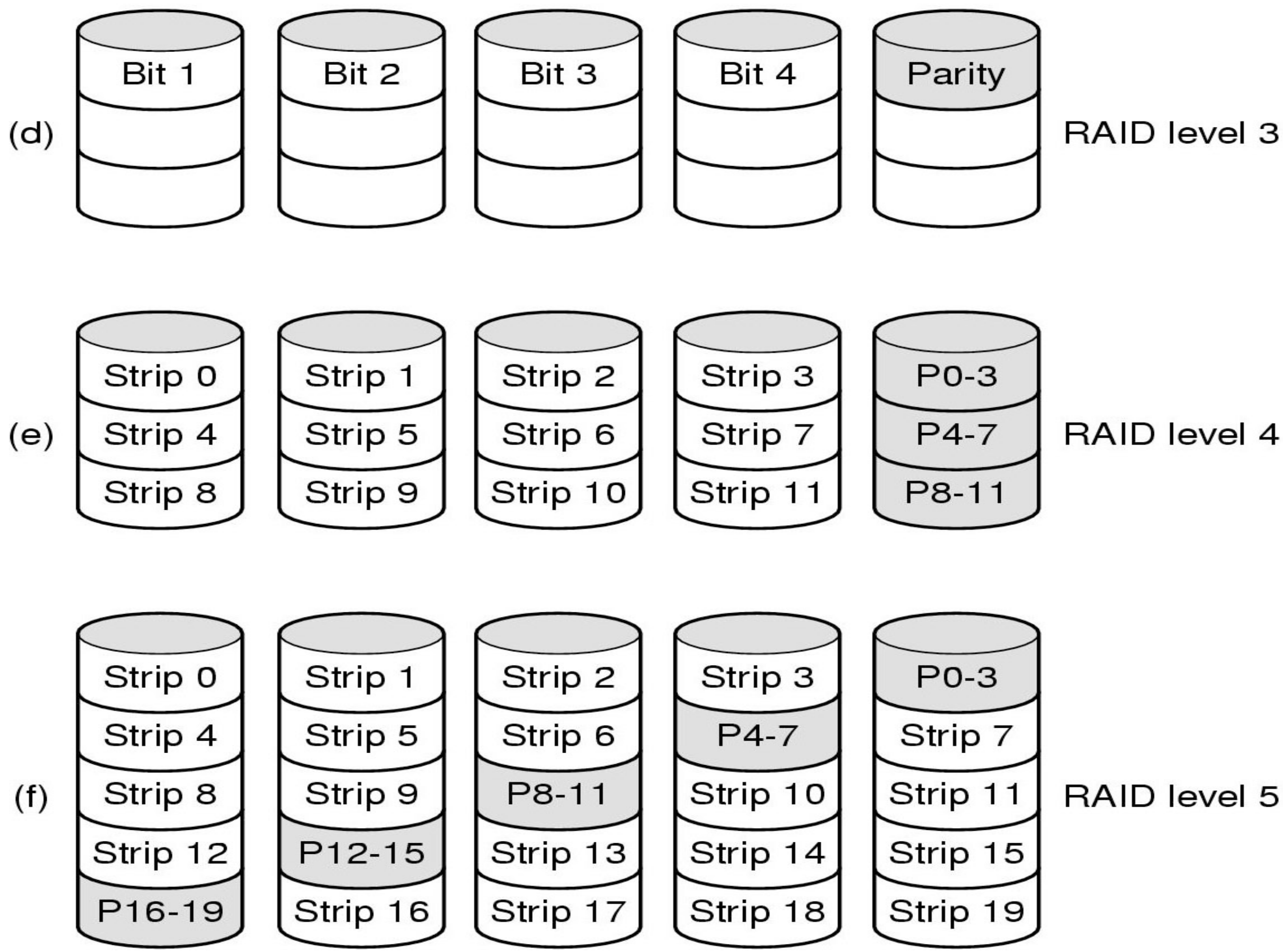
- **Increased reliability**
- **Increased performance**

RAID



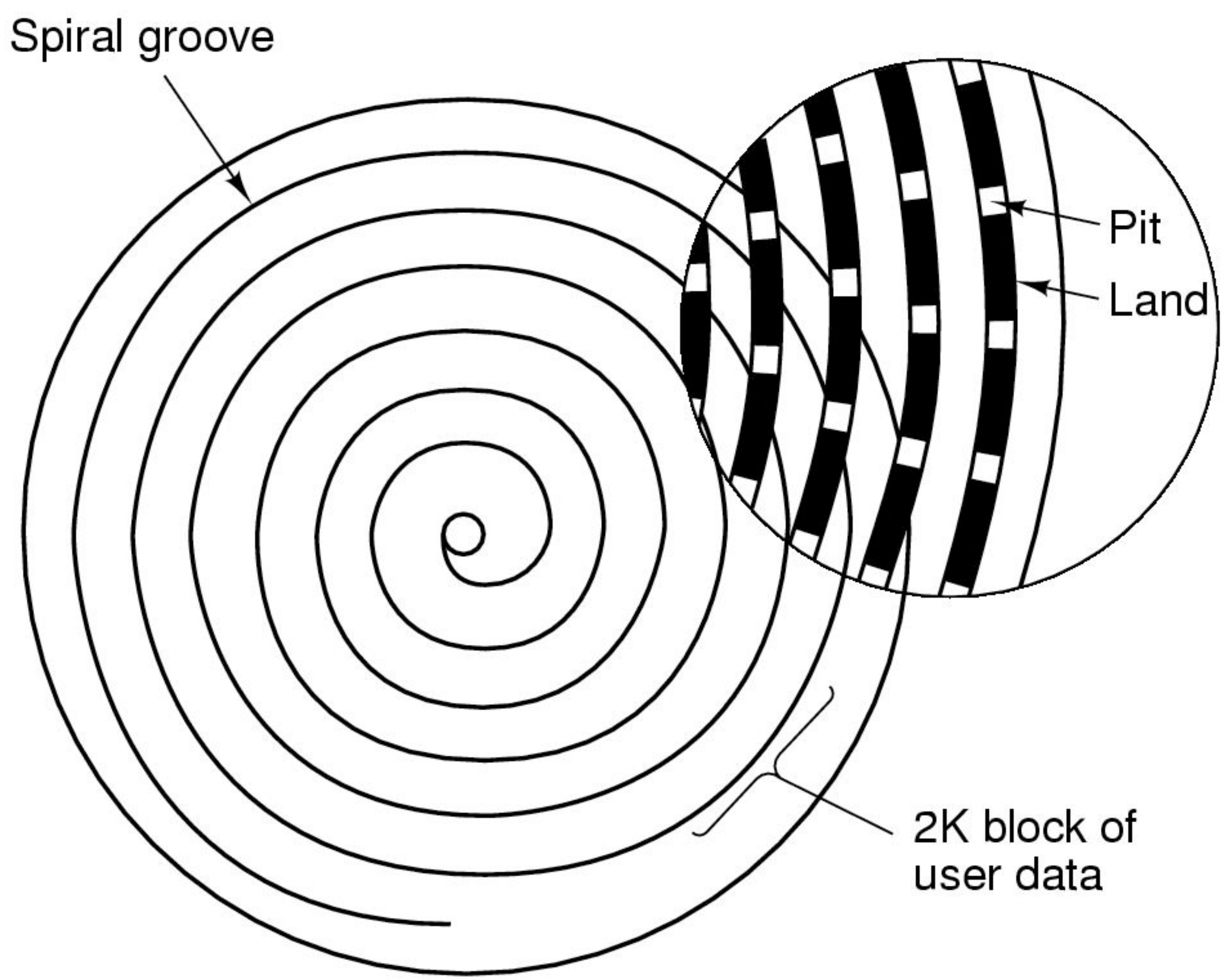
Strip 0
Strip 1
Strip 2
Strip 3
(a)
Strip 4
Strip 5
Strip 6
Strip 7
RAID level 0
Strip 8
Strip 9
Strip 10
Strip 11
Strip 0
Strip 1
Strip 2
Strip 3
Strip 0
Strip 1
Strip 2
Strip 3
RAID level 1
(b)
Strip 4
Strip 5
Strip 6
Strip 7
Strip 4
Strip 5
Strip 6
Strip 7
Strip 8
Strip 9
Strip 10
Strip 11
Strip 8
Strip 9
Strip 10
Strip 11
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
(c)
RAID level 2

RAID



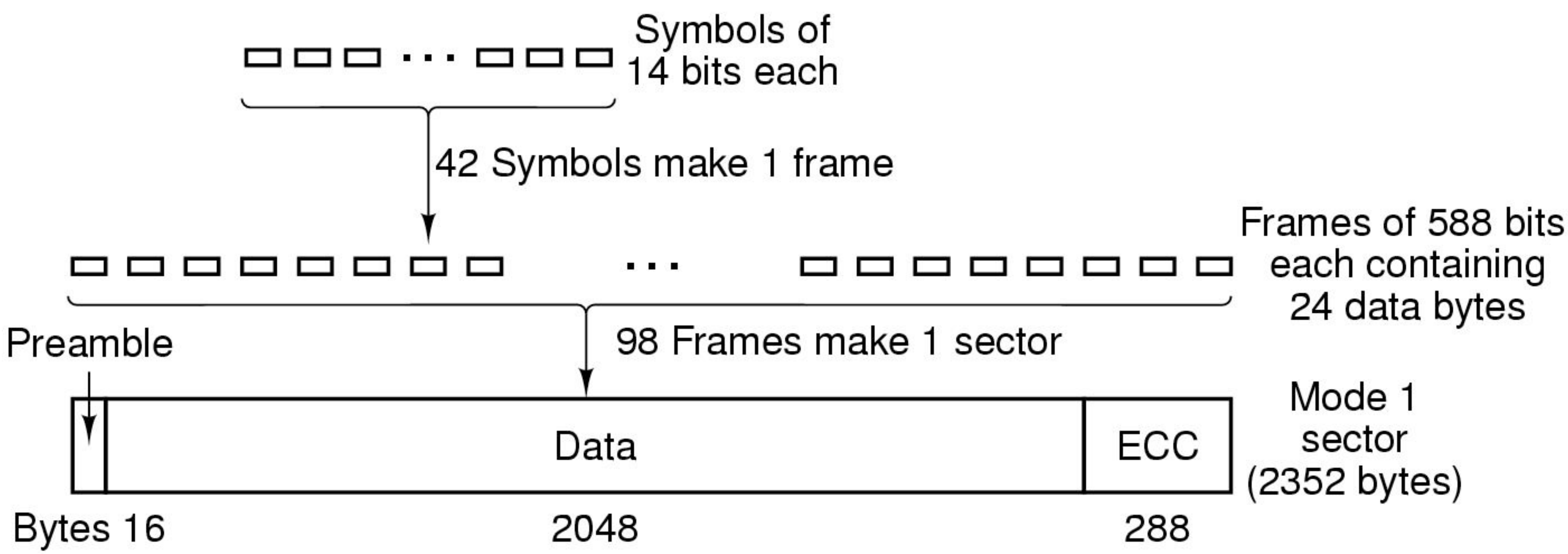
Bit 1
Bit 2
Bit 3
Bit 4
Parity
(d)
RAID level 3
Strip 0
Strip 1
Strip 2
Strip 3
P0-3
Strip 4
Strip 5
Strip 6
Strip 7
P4-7
Strip 8
Strip 9
Strip 10
Strip 11
P8-11
(e)
RAID level 4
Strip 0
Strip 1
Strip 2
Strip 3
P0-3
Strip 4
Strip 5
Strip 6
P4-7
Strip 7
Strip 8
Strip 9
P8-11
Strip 10
Strip 11
Strip 12
P12-15
Strip 13
Strip 14
Strip 15
P16-19
Strip 16
Strip 17
Strip 18
Strip 19
(f)
RAID level 5

CDs & CD-ROMs

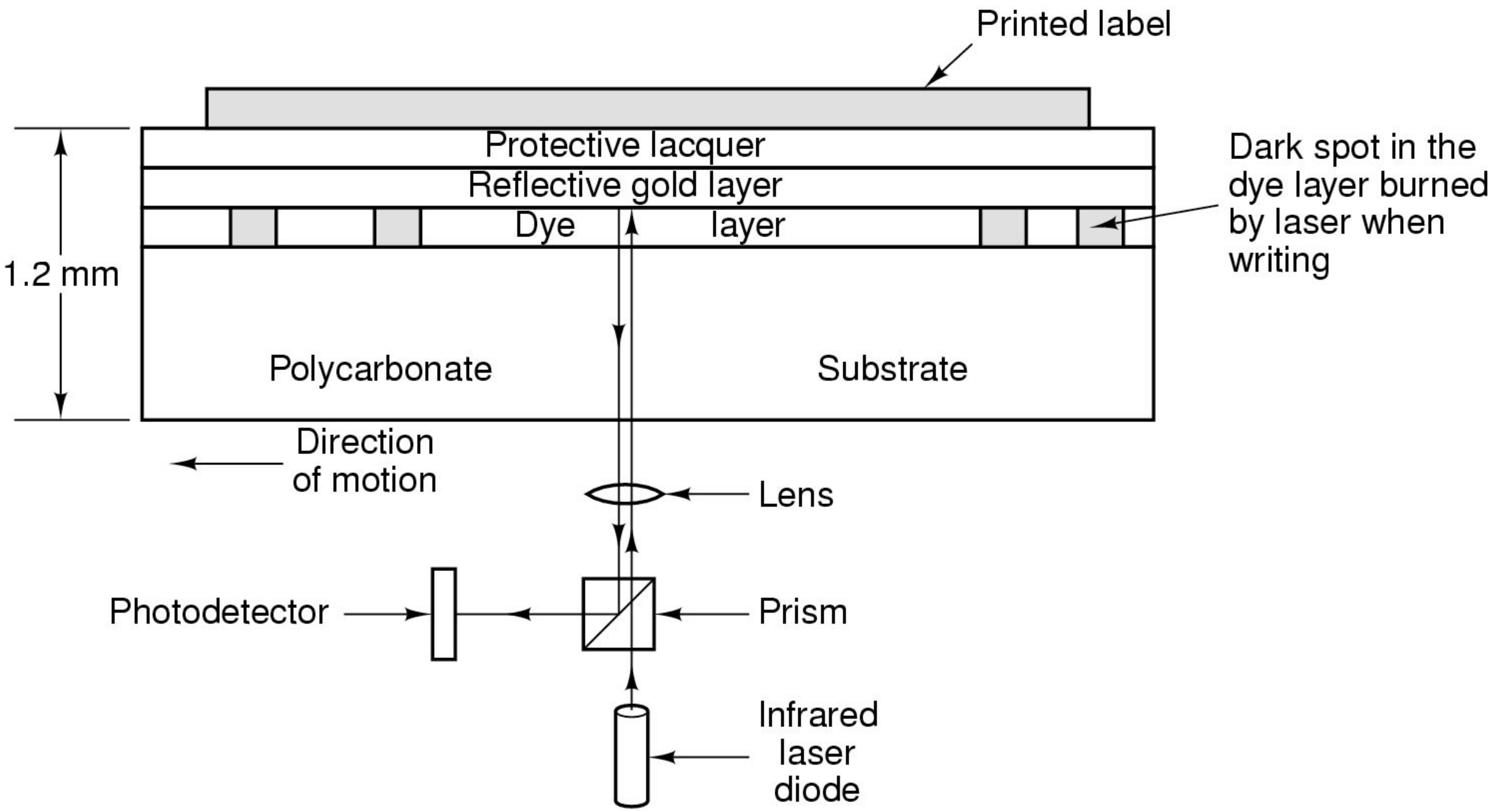


CD-ROMs

7203 bytes to encode 2048 (2K) data bytes
32x CD-ROM = 5,000,000 Bytes/Sec
SCSI-2 is twice as fast.



CD-R (CD-Recordable)



Updating Write-Once Media

VTOC = Volume Table of Contents
When writing, an entire track is written at once.
Each track has its own VTOC.

Updating Write-Once Media

VTOC = Volume Table of Contents
When writing, an entire track is written at once.
Each track has its own VTOC.
Upon inserting a CD-R,
- Find the last track
- Obtain the most recent VTOC
  - This can refer to data in earlier tracks
- This tells which files are on the disk
- Each VTOC supercedes the previous VTOC

Updating Write-Once Media

VTOC = Volume Table of Contents
When writing, an entire track is written at once.
Each track has its own VTOC.
Upon inserting a CD-R,

- **Find the last track**
- **Obtain the most recent VTOC**
  - **This can refer to data in earlier tracks**
- **This tells which files are on the disk**
- **Each VTOC supercedes the previous VTOC**

Deleting files?

- **Just leave out of VTOC for next write**

CD-RW

Uses a special alloy.

Alloy has two states, with different reflectivities

- **Crystalline (highly reflective) - Looks like “land”**
- **Amorphous (low reflectivity) - Looks like a “pit”**

Laser has 3 powers

- **Low power: Sense the state without changing it**
- **High power: Change to amorphous state**
- **Medium power: Change to crystalline state**

DVDs

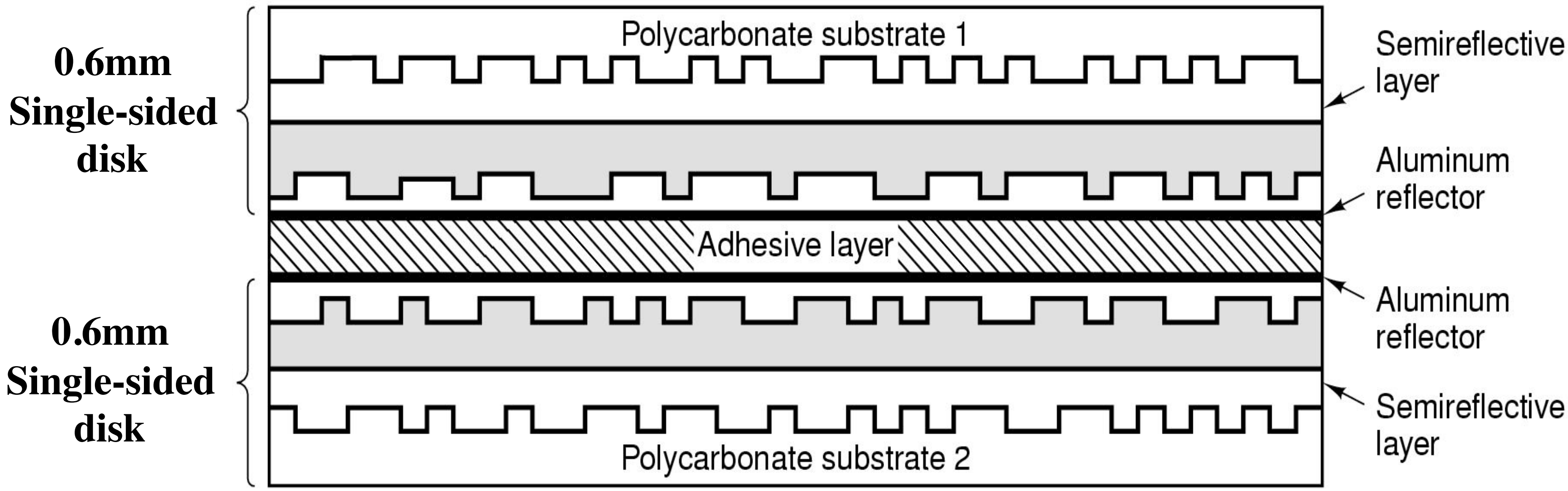
"Digital Versatile Disk"
- **Smaller Pits**
- **Tighter Spiral**
- **Laser with different frequency**

Transfer speed
- **1X = 1.4MB/sec (about 10 times faster than CD)**

Capacity

4.7 GB	**Single-sided, single-layer (7 times a CD-ROM)**
8.5 GB	**Single-sided, double-layer**
9.4 GB	**Double-sided, single-layer**
17 GB	**Double-sided, double-layer**

DVDs



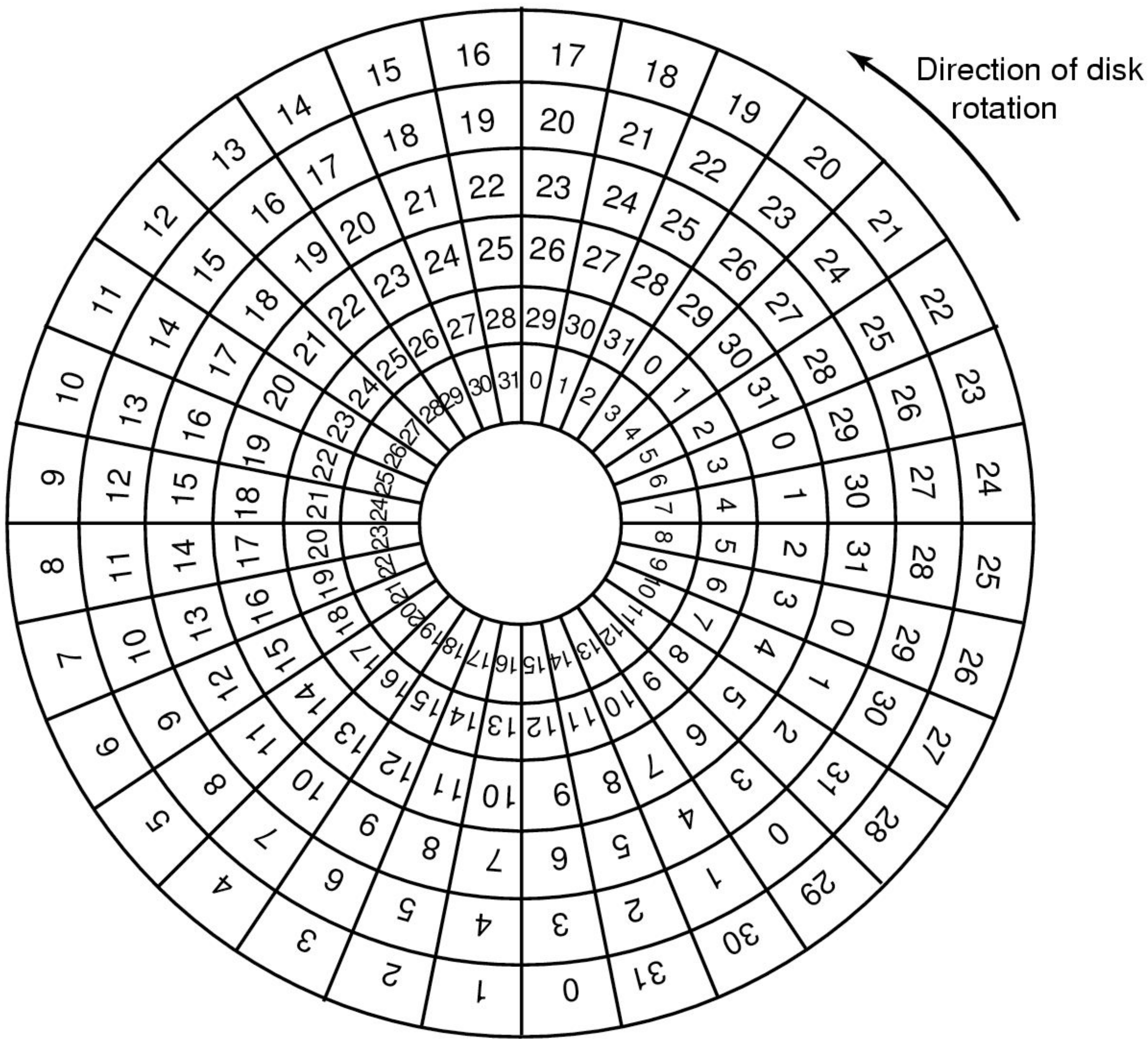
0.6mm
Single-sided
disk
0.6mm
Single-sided
disk
Polycarbonate substrate 1
Adhesive layer
Polycarbonate substrate 2
Semireflective
layer
Aluminum
reflector
Aluminum
reflector
Semireflective
layer

Disk Formatting

A disk sector

Typically
- **512 bytes / sector**
- **ECC = 16 bytes**

Cylinder Skew



Disk Capacity

For communication...

1 Kbps = 1,000 bits per second (10^3)
1 Mbps = 1,000,000 bits per second (10^6)
1 Gbps = 1,000,000,000 bits per second (10^9)

K	**kilo**	**$10^3 = 1000$**	
M	**mega**	**$10^6 = 1000 * 1000$**	**= 1,000,000**
G	**giga**	**$10^9 = 1000^3$**	**= 1,000,000,000**

Disk Capacity

For communication...

1 Kbps = 1,000 bits per second (10^3)
1 Mbps = 1,000,000 bits per second (10^6)
1 Gbps = 1,000,000,000 bits per second (10^9)

For disks and memories...

K	**kilo**	$2^{10} = 1024$	
M	**mega**	$2^{20} = 1024 * 1024$	**= 1,048,576**
G	**giga**	$2^{30} = 1024^3$	**= 1,073,741,824**

Disk Capacity

For communication...

1 Kbps = 1,000 bits per second (10^3)
1 Mbps = 1,000,000 bits per second (10^6)
1 Gbps = 1,000,000,000 bits per second (10^9)

For disks and memories...

~~K~~	~~kilo~~	Ki	kibi	2^{10} = 1024	
~~M~~	~~mega~~	Mi	mibi	2^{20} = 1024 * 1024	= 1,048,576
~~G~~	~~giga~~	Gi	gibi	2^{30} = 1024^3	= 1,073,741,824

Disk Capacity

For communication...

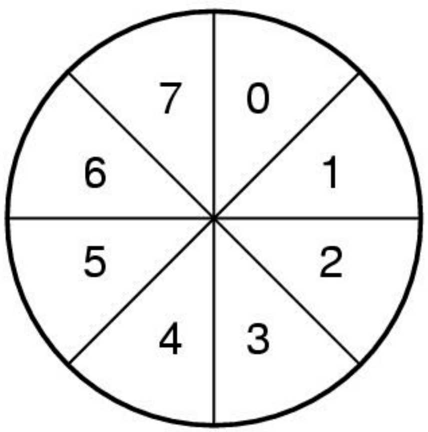
1 Kbps = 1,000 bits per second (10^3)
1 Mbps = 1,000,000 bits per second (10^6)
1 Gbps = 1,000,000,000 bits per second (10^9)

For disks and memories...

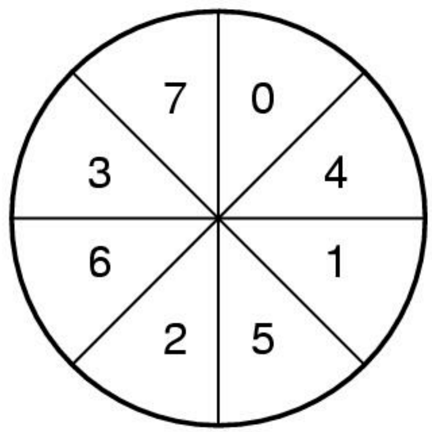
~~K~~	~~kilo~~	Ki	kibi	$2^{10} = 1024$	
~~M~~	~~mega~~	Mi	mibi	$2^{20} = 1024 * 1024$	= 1,048,576
~~G~~	~~giga~~	Gi	gibi	$2^{30} = 1024^3$	= 1,073,741,824

Examples: 30 KiB, 40 MiB, 50 GiB

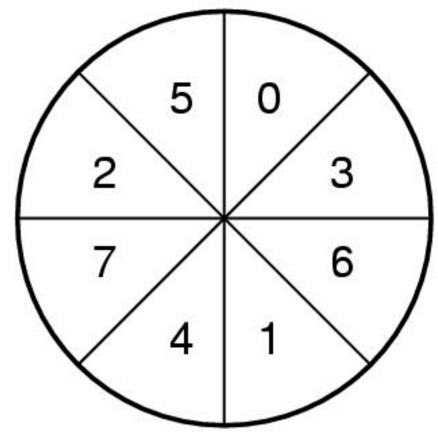
Sector Interleaving



No Interleaving



Single Interleaving



Double Interleaving

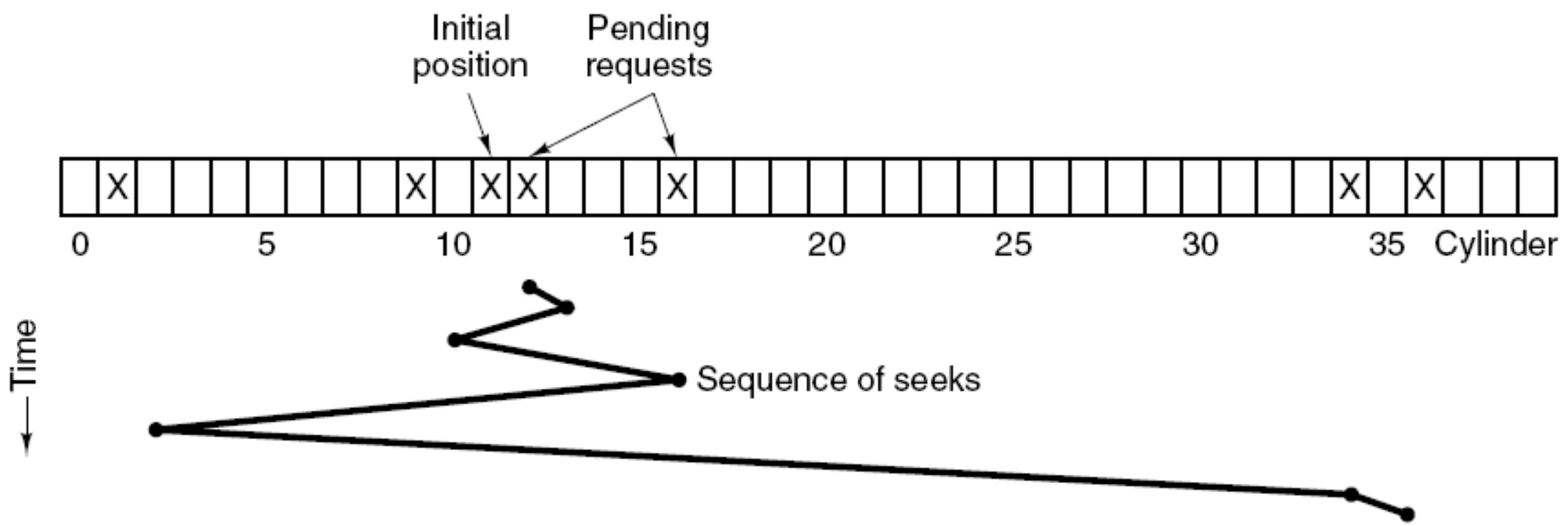
Disk Arm Scheduling Algorithms

- Seek Time
- Rotational Delay
- Transfer Time

Seek time dominates
Want to "schedule" disk reads & writes to minimize it

Scheduling Algorithms:
  FCFS: First come, first served
  SSF: Shortest seek first
  Elevator: keep moving in one direction.

Shortest Seek First (SSF)



Shortest Seek First (SSF)

Cuts arm motion in half.
Fatal problem:
Starvation is possible!

The Elevator Algorithm

One bit: which direction the arm is moving.

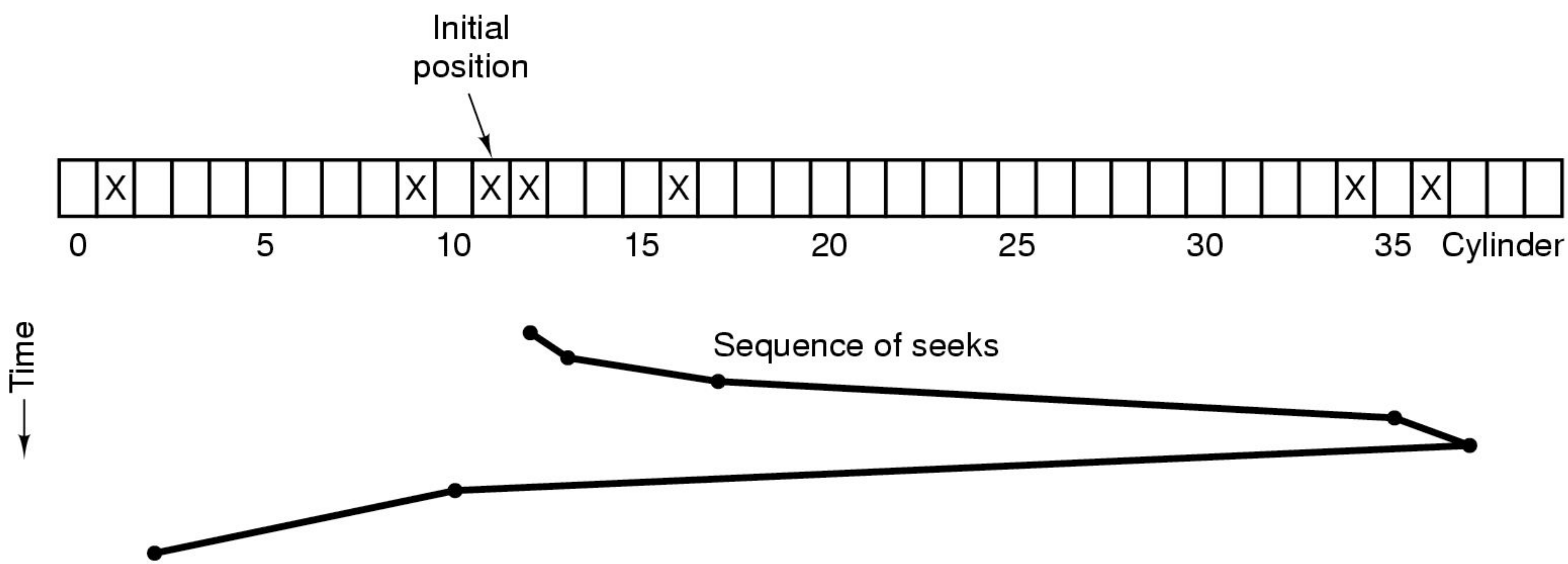
- Up
- Down

Keep moving in that direction.

Service the next pending request in that direction

When there are no more requests in the current direction, reverse direction.

The Elevator Algorithm



Errors on Disks

Transient errors v. Hard errors

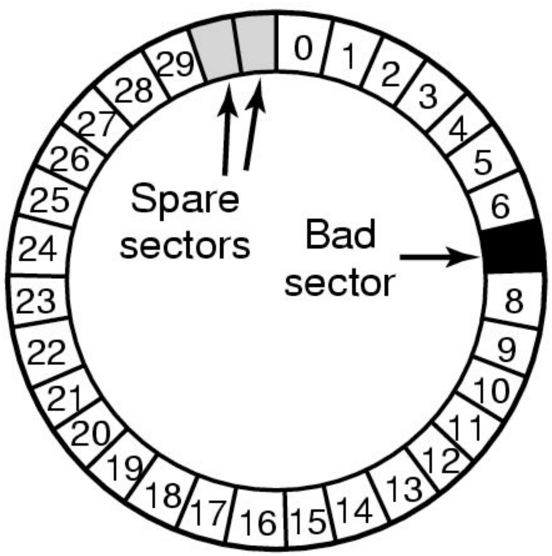
Manufacturing defects are unavoidable
- **Some will be masked with the ECC in each sector**

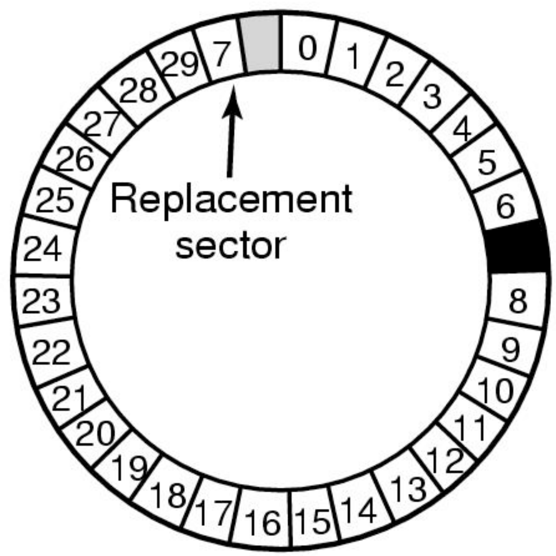
Dealing with bad sectors
- **Allocate several spare sectors per track**

At the factory, some sectors are remapped to spares
- **Errors may occur during the disk lifetime**

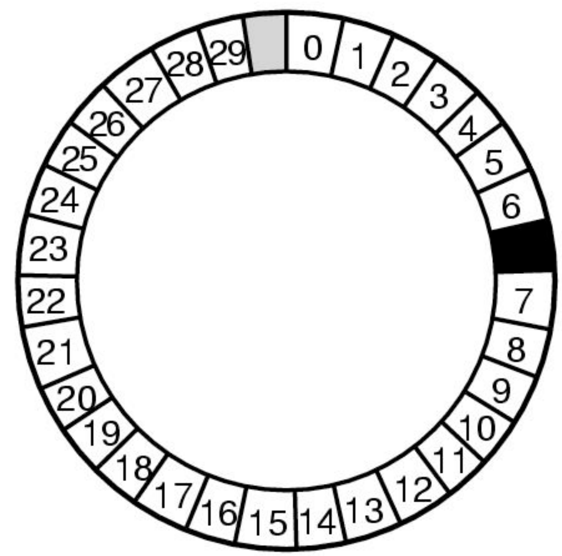
The sector must be remapped to a spare
- **By the OS**
- **By the device controller**

Using Spare Sectors





Substituting a new sector



Shifting sectors

Handling Bad Sectors in the OS

- **Add all bad sectors to a special file.**
  - **Hidden; not in file system**
  - **Users will never see the bad sectors**
    - **Never an attempt to access the file**

- **Backups**
  - **Some backup programs copy entire <u>tracks</u> at a time**
    - **Efficient**
  - **Problem:**
    - **May try to copy every sector**
    - **Must be aware of bad sectors**

Stable Storage

The model of possible errors:

The unit of I/O is a "disk block".

- The write operation writes incorrect bits

  ... but it will be detected upon reading the block

  Probability of an error being missed?

  Assume 16 bytes of ECC code

  8 × 16 bits (= 128 bits) of ECC

  $1 / 2^{128}$ chance ECC just happens to be right
- Disk blocks can go bad spontaneously

  ... but subsequent reads will detect the error
- Computer can fail (Failure model: hardware just stops)

  ... disk writes in progress are detectable errors
- Highly unlikely to loose the same block on two disks

  ... on the same day

Stable Storage

Use two disks for redundancy.

Each write is done twice.
Each disk has N blocks.
Each disk contains exactly the same data.

To read the data,
... you can read from either disk

To perform a write...
must update the same block on both disks.

If one disk goes bad...
You can recover from the other disk.

Stable Storage

Stable Write

Write block on disk # 1.
Read back to verify.
If problems...
  Try again several times to get the block written.
  Then declare the sector bad and remap the sector.
  Repeat until the write to disk #1 succeeds.
Write same data to corresponding block on disk #2
  Read back to verify
  Retry until it also succeeds

Stable Storage

Stable Read

Read the block from disk # 1
If problems...
  Try again several times to get the block
If the block can not be read from disk #1...
  Read the corresponding block from disk #2

Our Assumption:
The same block will not simultaneously go bad on both disks.

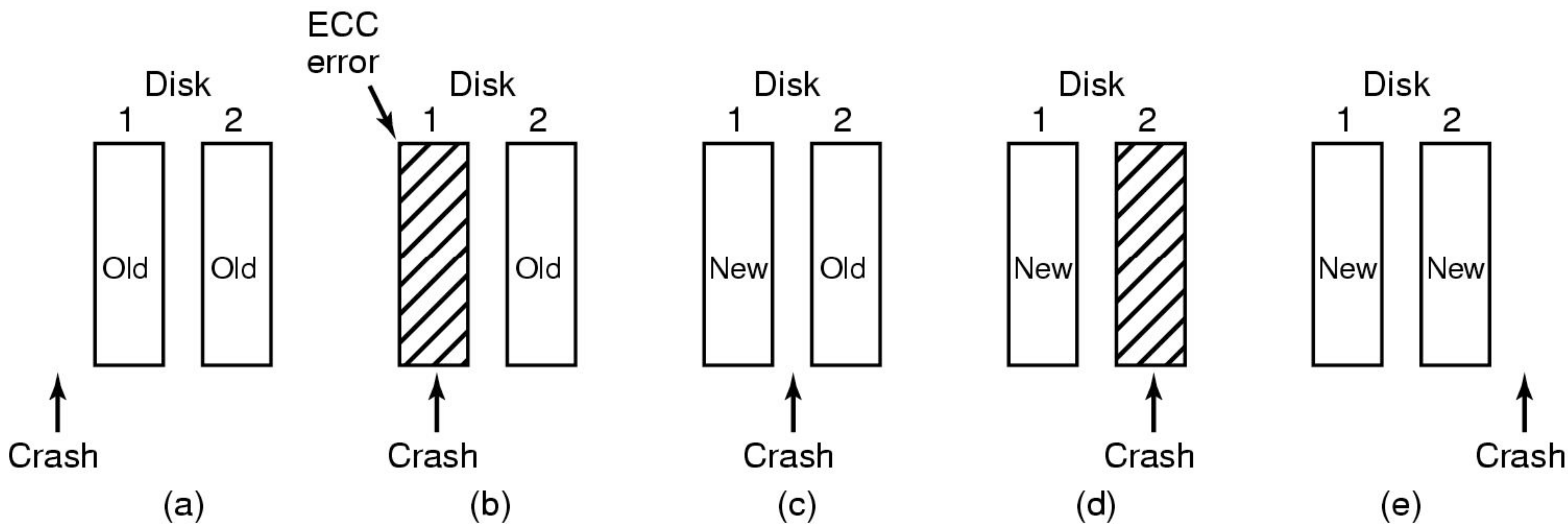
Stable Storage

Crash Recovery

Scan both disks
Compare corresponding blocks
For each pair of blocks...

- **If both are good and have same data...**
  - **Do nothing; go on to next pair of blocks.**
- **If one is bad (failed ECC)...**
  - **Copy the block from the good disk.**
- **If both are good, but contain different data...**
  - **(CPU must have crashed during a "Stable Write")**
  - **Copy the data from disk #1 to disk #2.**

Crashes During a Stable Write



Stable Storage

Disk blocks can spontaneously decay.

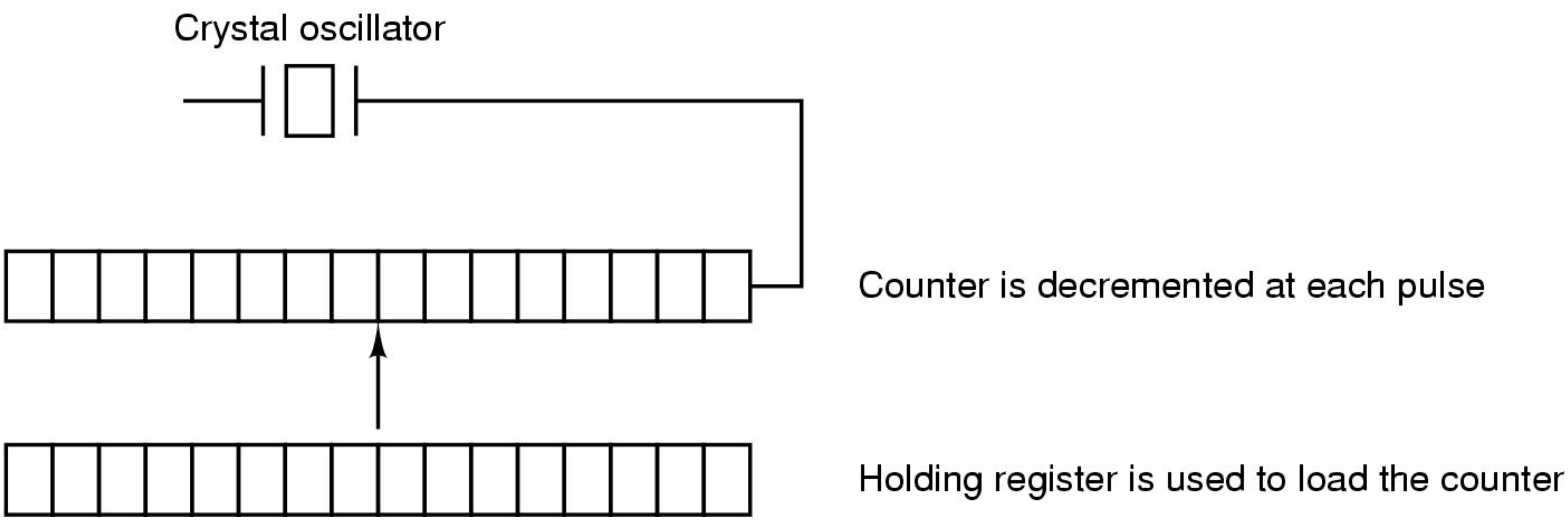
Given enough time...

- The same block on both disks may go bad
  - Data could be lost!
- Must scan both disks to watch for bad blocks
  - (e.g., every day)

Many variants to improve performance

- Goal: avoid scanning entire disk after a crash.
- Goal: improve performance
  - Every Stable Write requires: 2 writes & 2 reads
    - Can do better...

Programmable Clocks



One-shot mode:

Counter decremented until zero
A single interrupt occurs

Square wave mode:

When counter reaches zero, it is reloaded.
Periodic interrupts (called “clock ticks”)

Time

500 MHz Crystal (every 2 nanoseconds)
32 bit register overflows in 8.6 seconds

Backup clock

- **Similar to digital watch**
- **Low-power circuitry, battery-powered**
- **Periodically reset from the internet**
- **UTC: Universal Coordinated Time**
- **Unix: Seconds since Jan. 1, 1970**
- **Windows: Seconds since Jan. 1, 1980**

Goals of Clock Software

- Maintain time of day
  Must update the time-of-day every tick
- Prevent processes from running too long
- Account for CPU usage
  Separate timer for every process
  Charge each tick to the current process
- Handling the "Alarm" syscall
  User programs ask to be sent a signal at a given time
- Providing watchdog timers for the OS itself
  E.g., when to spin down the disk
- Doing profiling, monitoring, and statistics gathering

Software Timers

A process can ask for notification at time T.

- At time T, the OS will signal the process

Processes can "go to sleep until time T".

Several processes can have active timers.

The CPU has only one clock.

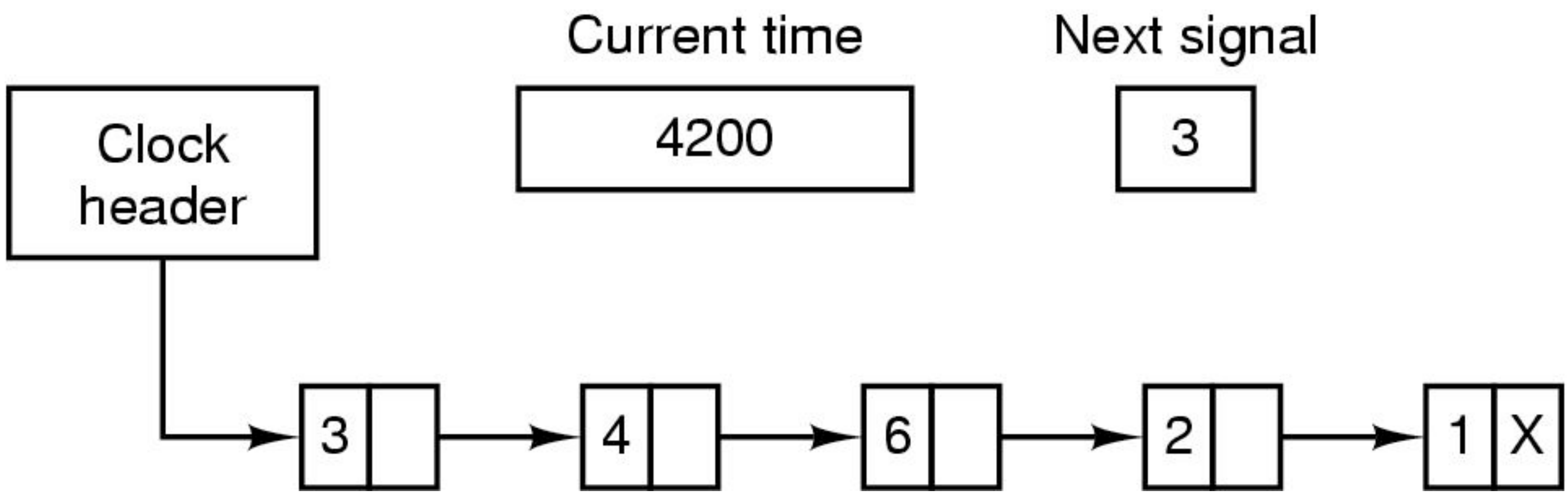
- Must service the alarms in the right order.

Keep a sorted list of all timers.

Each entry tells when the alarm goes off

- and what to do then.

Software Timers



Alarms set for 4203, 4207, 4213, 4215 and 4216.
Each entry tells how many ticks past the previous entry.
On each tick, decrement the "NextSignal".
When it gets to 0, then signal the process.

Watchdog Timers

Scenario:

- **Embedded system**
- **Detect and recover from crashes, infinite loops**

  Example: Space probe, bug, infinite loop

Initialize the timer with a "interval" of time

e.g., 1 second

Software must "feed" the timer

...every second

By writing 0x12345678 to a special device register

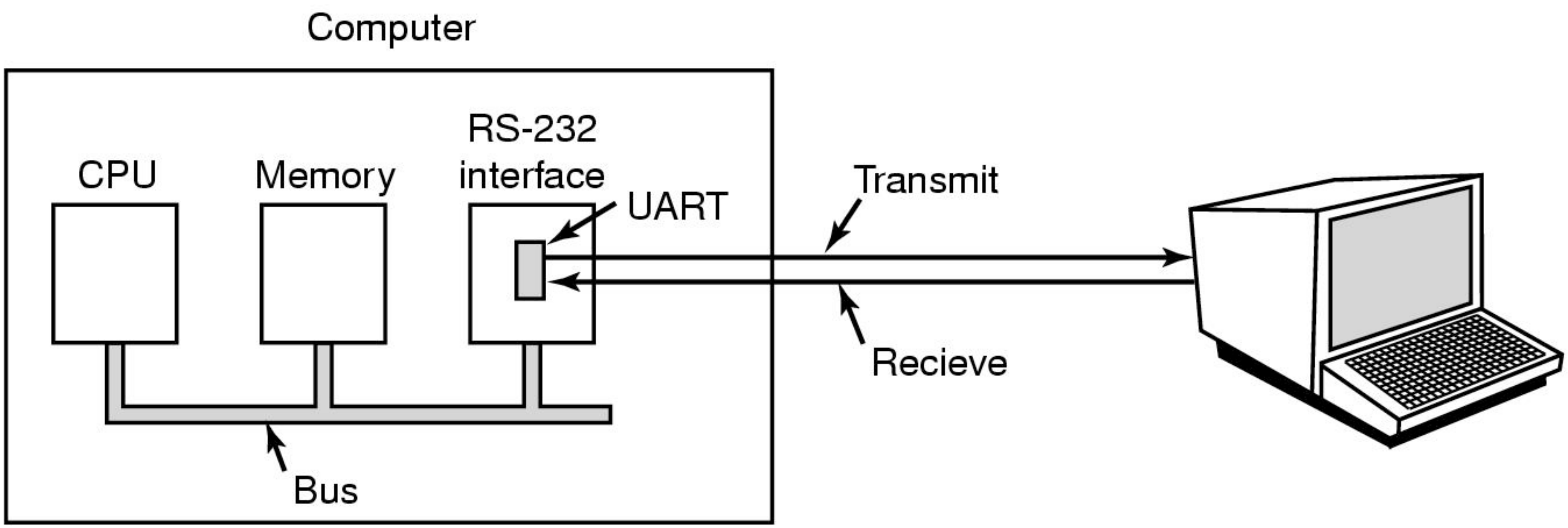
Failure to "feed" the watch dog?

Full "SYSTEM RESET" will be triggered.

Character-Oriented I/O

RS-232 / Serial interface / Modem / Terminals / tty / COM
Bit serial (9- or 25-pin connectors), only 3 wires used
UART: Universal Asynchronous Receiver Transmitter

byte → serialize bits → wire → collect bits → byte



Terminals

56,000 baud = 56,000 bits per second = 8000 bytes / sec
ASCII character codes

Dumb CRTs / teletypes
- **Very few control characters**
  - **newline, return, backspace**

Intelligent CRTs
- **Also accept "escape sequences"**
- **Reposition the cursor, clear the screen, insert lines, etc.**
- **The standard "terminal interface" for computers**
  - **Example programs: vi, emacs**
- **VT-100: The terminal emulator standard**

Input Software

Character Processing

User types "`hella←o`"

Computer echoes as: "`hella←_←o`"

Program will see "`hello`"

Raw Mode

The driver delivers all characters to application

No modifications, no echoes.

vi, emacs, the BLITZ emulator, password entry

Cooked Mode

The driver does echoing and processing of special chars.

"Canonical mode"

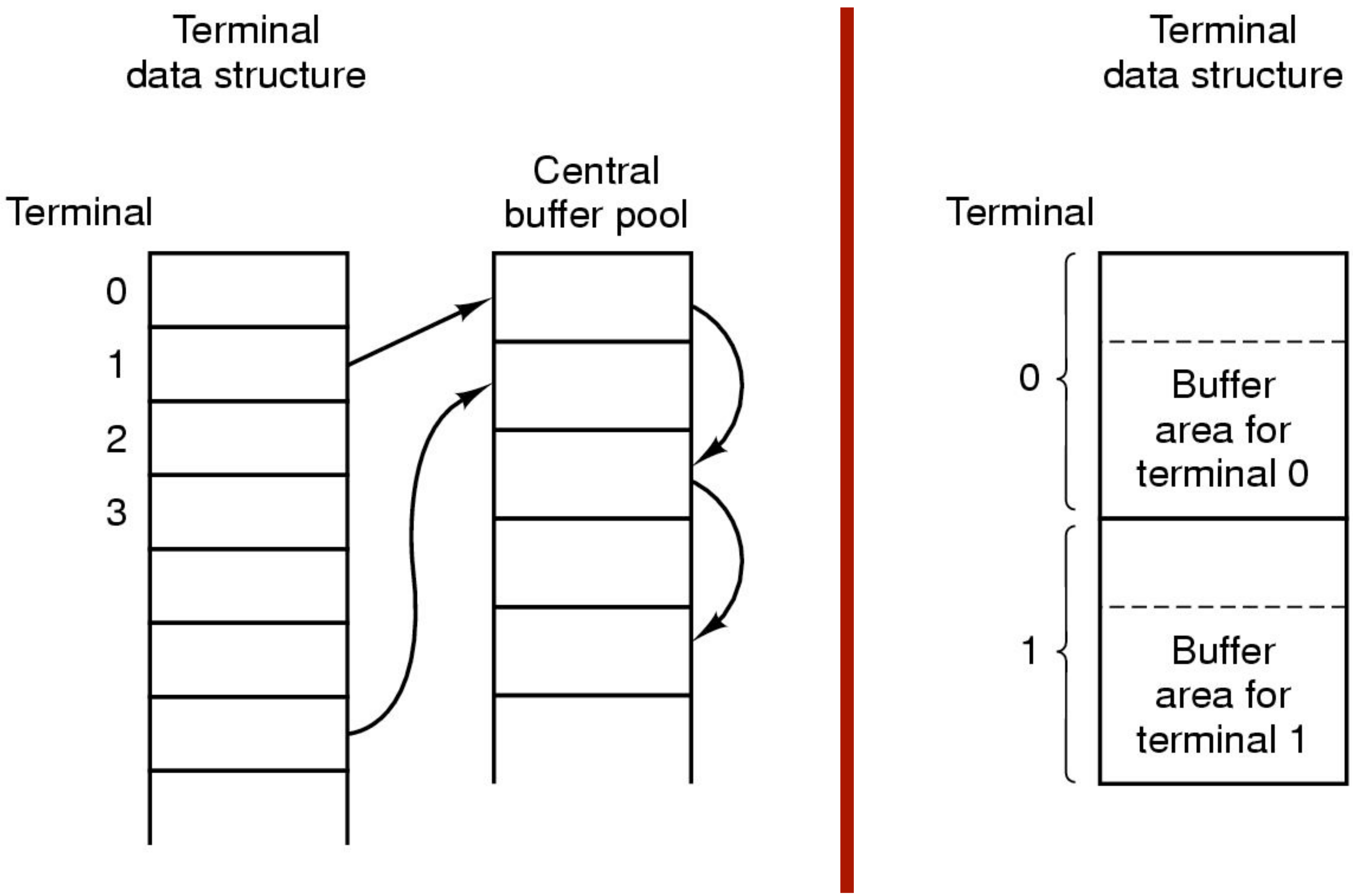
Cooked Mode

The terminal driver must...

- **Buffer an entire line before returning to application**
- **Processes special control characters**
  - **Control-C**
  - **Backspace, line-erase, tabs**
- **Echo the character just typed**
- **Accommodate type-ahead**
  - **Need an internal buffer**

Approach 1 (for computers with many terminals)
Have a pool of buffers to use as necessary
Approach 2 (for single-user computer)
Have one buffer (e.g., 500 bytes) per terminal

Central Buffer Pool vs. Dedicated Buffers



(a) (b)

The End-Of-Line Problem

NL "newline" (ASCII `0x0A`, `\n`)

Move cursor down one line (no horizontal movement)

CR "return" (ASCII `0x0D`, `\r`)

Move cursor to column 1 (no vertical movement)

"ENTER key"

Behavior depends on the terminal specs.

May send CR, may send NL, may send both.

Software must be device independent.

Unix, Macintosh:

Each line (in a file) ends with a NL.

Windows:

Each line (in a file) ends with CR & NL.

Special Control Characters (in "cooked mode")

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)

Control-D: EOF

Typing Control-D ("End of file")
causes the read request to be satisfied immediately

Do not wait for "enter key"
Do not wait for any characters at all
May return 0 characters

Within the user program

```
count = Read (fd, buffer, buffSize)
if count == 0
 -- Assume end-of-file reached...
```

Outputting to a Terminal

The terminal accepts an "escape sequence"
Tells it to do something special

Example:

esc [3 ; 1 H esc [0 K esc [1 M

ESCAPE: 0x1B

- esc [3 ; 1 H — Move to position (3,1) on screen
- esc [0 K — Erase the line
- esc [1 M — Shift following lines up one

Each terminal manufacturer had
a slightly different specifcation.

Makes device independent software difficult

Unix "termcap" file
Database of different terminals and their behaviors.

ANSI Escape Sequence Standard

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

Graphical User Interfaces (GUIs)

Memory-Mapped Displays
- **"bit-mapped graphics"**

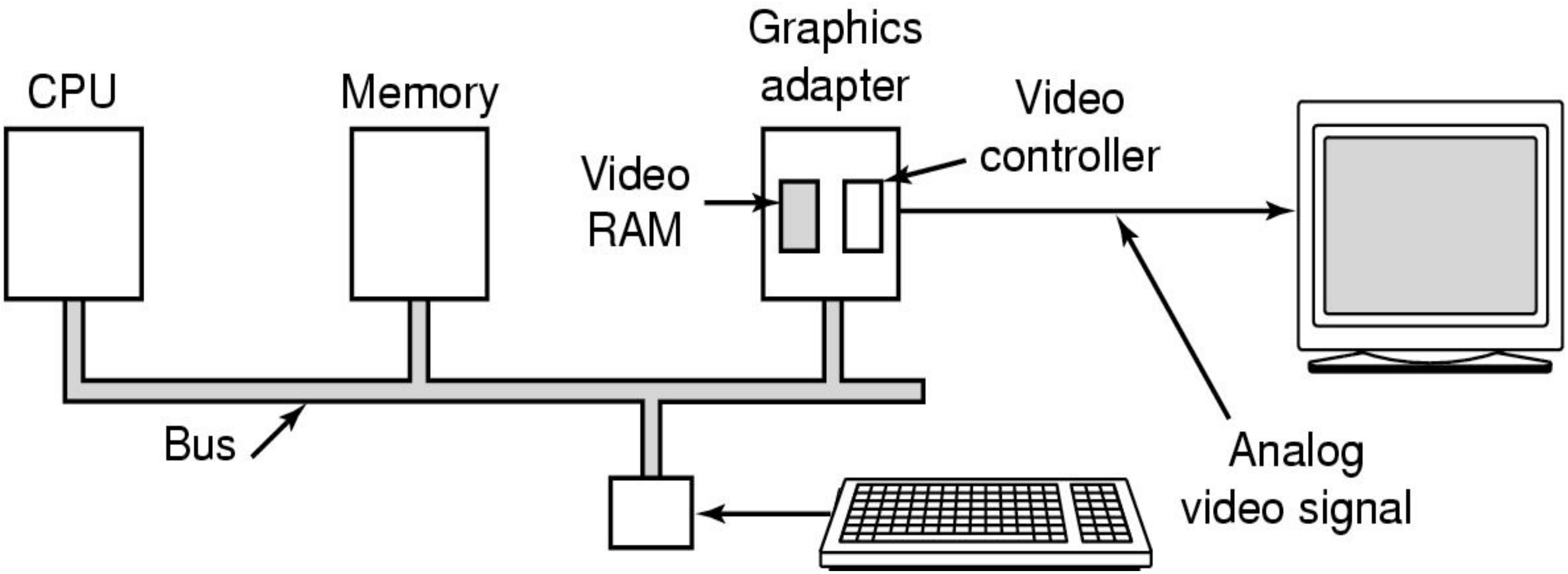
Video driver moves bits into special memory region
- **Changes appear on the screen**
  - **Video controller constantly scans video ram**

Black and white displays
- **1 bit = 1 pixel**

Color
- **24 bits = 3 bytes = 1 pixels**
  - **red (0-255)**
  - **green (0-255)**
  - **blue (0-255)**

1280 * 854 * 3 = 3 MB

Graphical User Interfaces (GUIs)



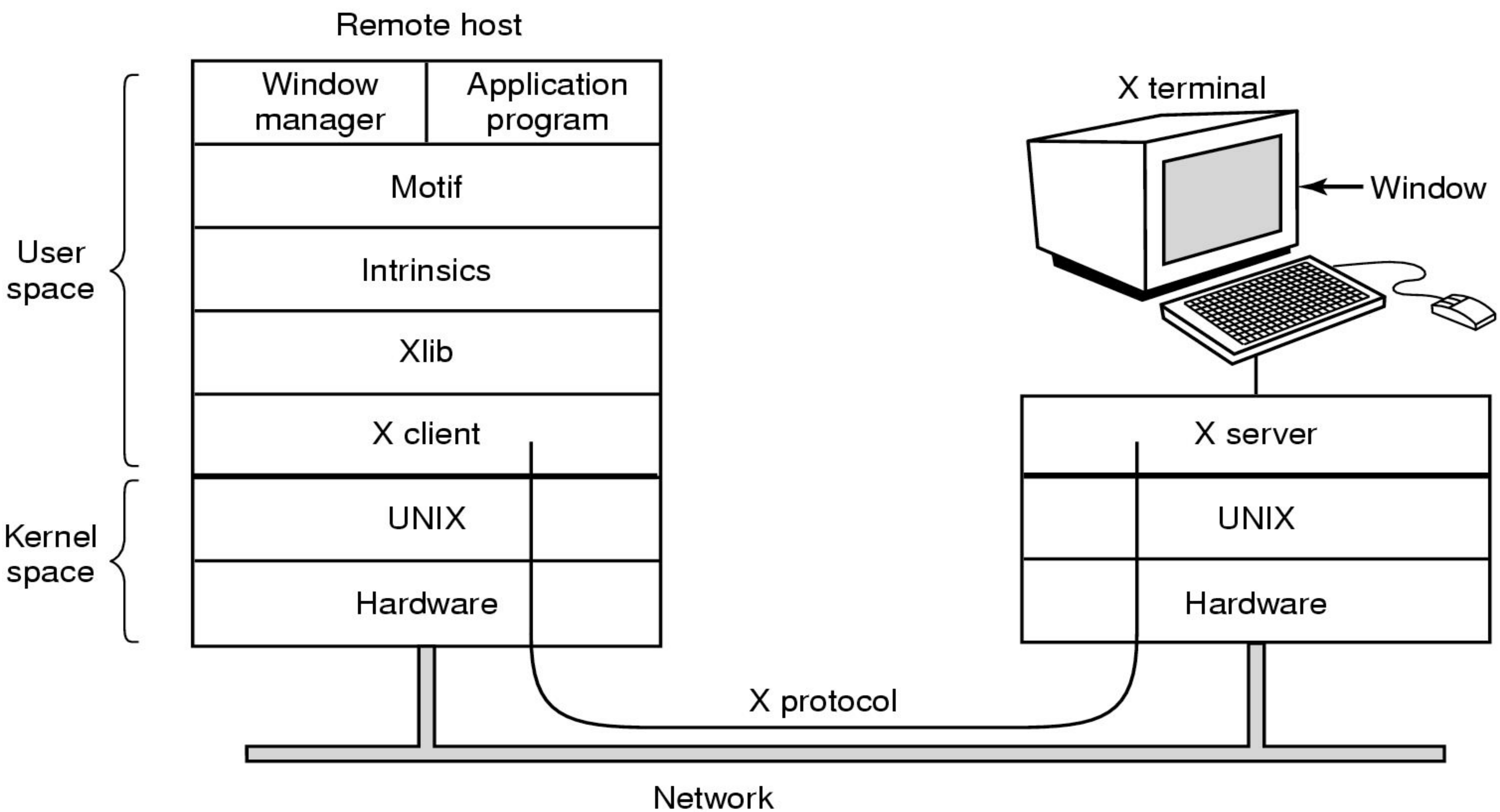
X Window System

Client - Server
Remote Procedure Calls (RPC)
Client makes a call.
Server is awakened; the procedure is executed.

Intelligent terminals ("X terminals")

The display side is the *server*.
The application side is the *client*.
The application (client) makes requests to the display server.
Client and server are separate processes
(May be on the same machine)

X Window System



X Window System

X-Server

Display text and geometric shapes, move bits

Collect mouse and keyboard status

X-Client

Xlib

library procedures; low-level access to X-Server

Intrinsics

provide "*widgets*"

buttons, scroll bars, frames, menus, etc.

Motif

provide a "look-and-feel" / style

Window Manager

Application independent functionality

Create & move windows

The SLIM Network Terminal

Stateless Low-level Interface Machine (SLIM)
Sun Microsystems

Philosophy: ***Keep the terminal-side very simple!***

Back to "dumb" terminals"

Interface to X-Server:
100's of functions

SLIM:
Just a few messages
The host tells which pixels to put where
The host contains all the intelligence

The SLIM Network Terminal

The SLIM Protocol
from application-side (server)
to terminal (the “thin” client)

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

Also in Chapter 5 – But not covered here

Power Management

Graphical User Interfaces

Soft Timers