Operating Systems
Chapter 5

Marie Roch

contains slides from:

Tanenbaum 2001, 2008

Silbershatz, Galvin, and Gagne 2003

Basic ideas

- OS view – How to interface with device
- Major types
  - block device
  - character device
  - other, e.g. clock?
- Examples of each?

Communication

- Interface to device via controller
  - e.g. Oxford 912 IEEE 1394 controller
  - Controller interfaces to bus

- How do we communicate with controller?

I/O Port Space

- IN/OUT instructions
- Port types
  - control
  - data
- Sample syntax

```
In Reg, Port
Out Reg, Port
```

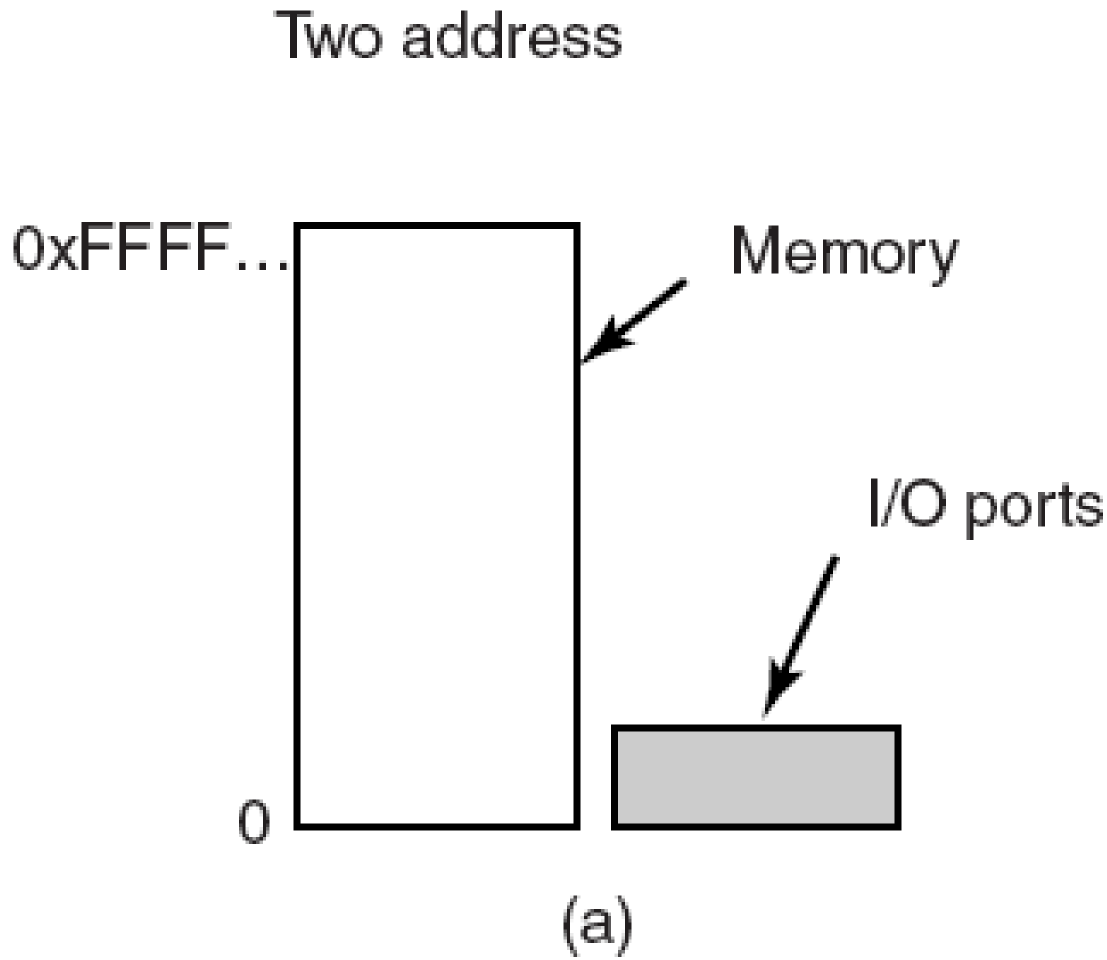
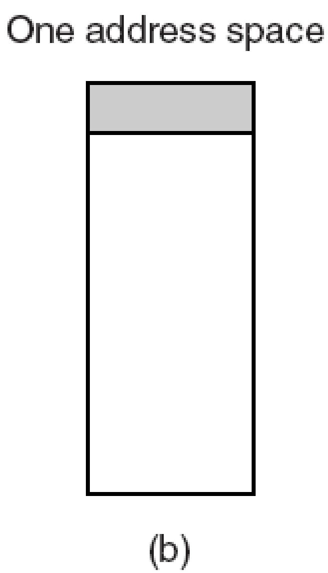


Figure 5-2, Tanenbaum, 3rd ed., p 333

Memory-Mapped I/O



- Control and data registers are assigned addresses

Figure 5-2, Tanenbaum, 3^{rd} ed., p 333

Hybrid I/O

- Combination
  - memory-mapped
  - port space

- Modern Intel architectures support both port and memory-mapped I/O

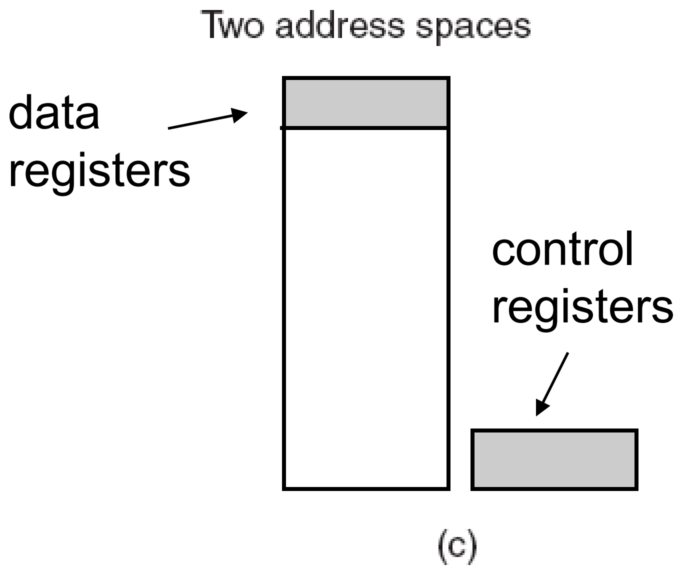


Figure 5-2, Tanenbaum, 3rd ed., p 333

I/O space implementation & consequences

- Port space
  - Extra address line
  - Forces use of assembler IN/OUT
- Memory mapped
  - Conceptually easier
  - Cache issues
  - Bus issues

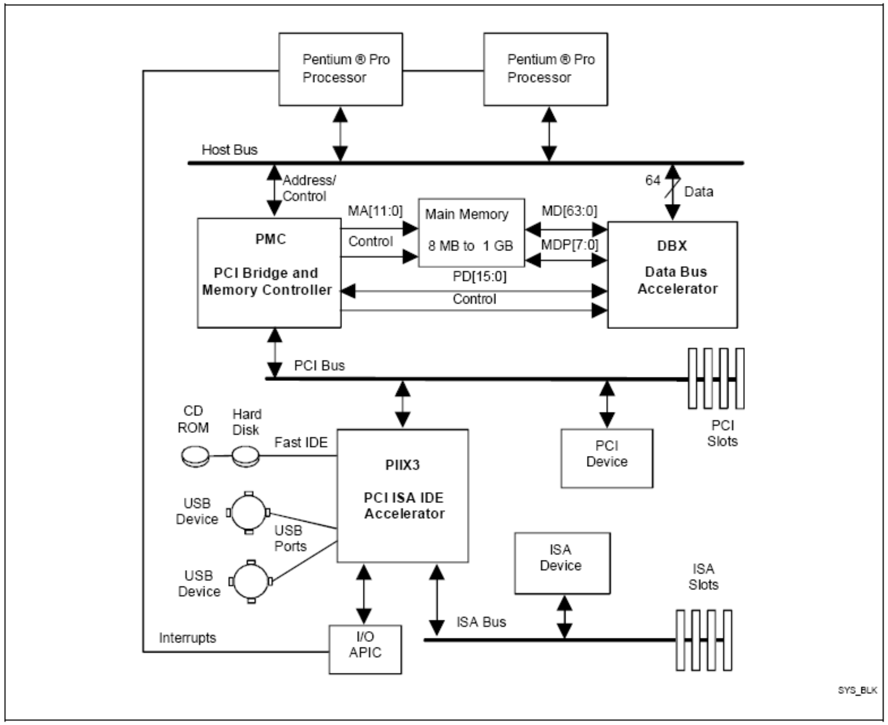


Figure 1. 440FX PCIset System Block Diagram

Intel

Sample memory controller for Pentium Pro
Intel 440 FX Memory controller

Memory mapped example

```
/*
 * Kernel mode routine to write to
 *   a fictitious device
 * CmdReg - Pointer to command register
 * StatusReg - Pointer to status register
 * DeviceBuf - Pointer to device buffer
 * ToXfr - data buffer to be transferred
 * BlockSz - Block size
 * BlockN - Number of blocks
 */


bool block_xfr_out(REG *CmdReg,
    REG *StatusReg, char *DeviceBuf,
    char *ToXfr, int BlockSz, int BlockN)
    {
```

```
/* write blocks one at a time */
  for (int b=0; b < BlockN; b++) {
    /* Wait for device to be ready */
    while ! (*StatusReg & READYBIT)
      ;
    /* Fill buffer */
    for (int i=0; i < BlockSz; i++) {
      /* Copy next byte to device */
      *(DeviceBuf + i) = *ToXfr++;
    }

    /* Write block */
    *CmdReg = (*CmdReg | WRITEBIT);
  }

  /* wait for final write */
  while ! (*StatusReg & READYBIT)
    ;
}
```

What type of I/O is this?

Device I/O

Memory mapped

```
// byte register
// mapped to location 0x1200

uint8 *char = 0x1200;
uint8 value;

// Read register
value = *(char);
// Assign register
*(char) = 0x7;
```

Port mapped

```
// x86 assembler
// byte register mapped
// to I/O port 0x400

// Read to byte (AL register)
//
IN AL, 0x400

// Write to register
MOV AL, 0x7
OUT 0x400, AL
```

Clock Hardware

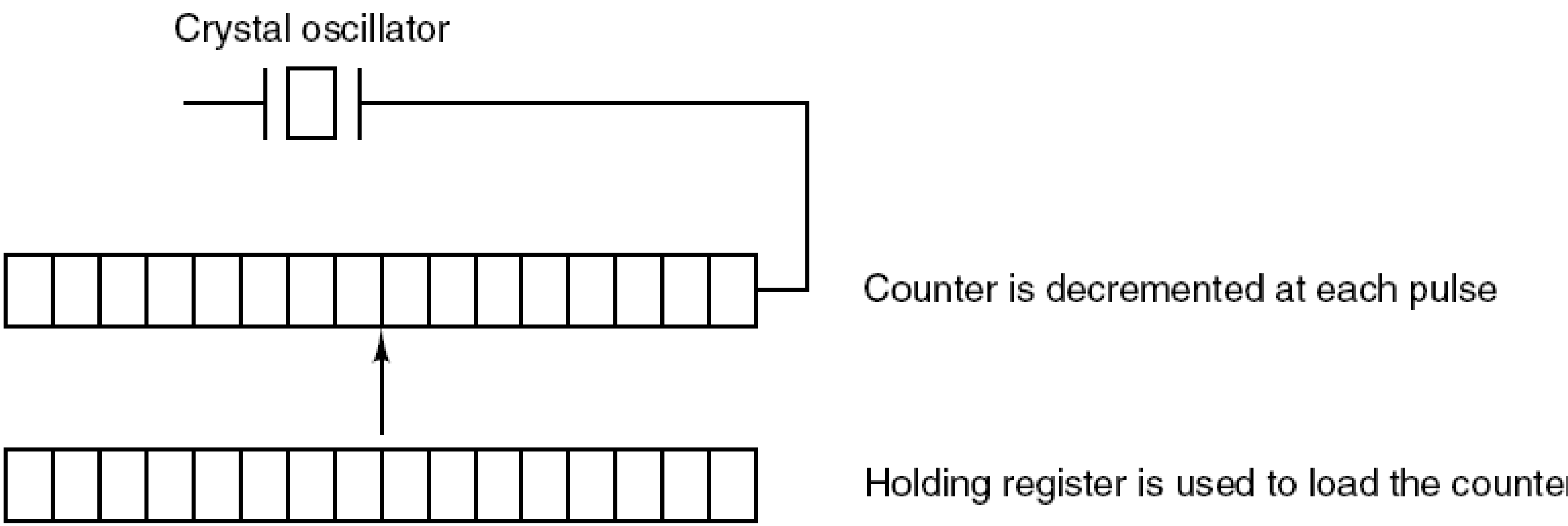


Figure 5-32. A programmable clock.

- Two modes of operation
  - one-shot – Count down then interrupt
  - square-wave – Count down, interrupt, reload & repeat

Clock usage

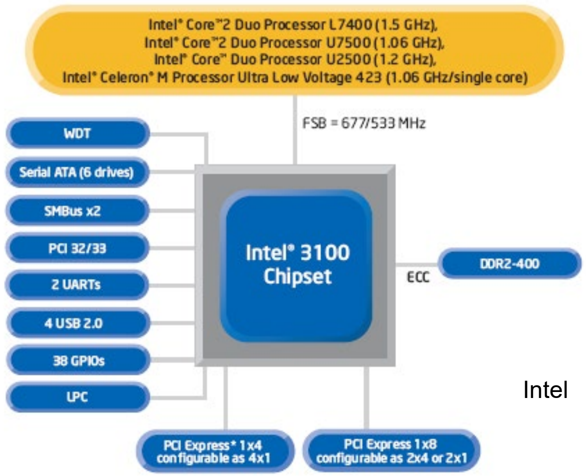
- Many tasks
  - time of day
  - scheduling processes
  - providing timing services to processes
  - profiling and bookkeeping
  - watchdog timers

Clock implementation

- Limited number of clocks
- Possible for many users to request timers
- Solution
  - Maintain min heap of deadlines
  - When a timer expires, reset to the remaining time in the top item of the heap

Direct memory access (DMA)

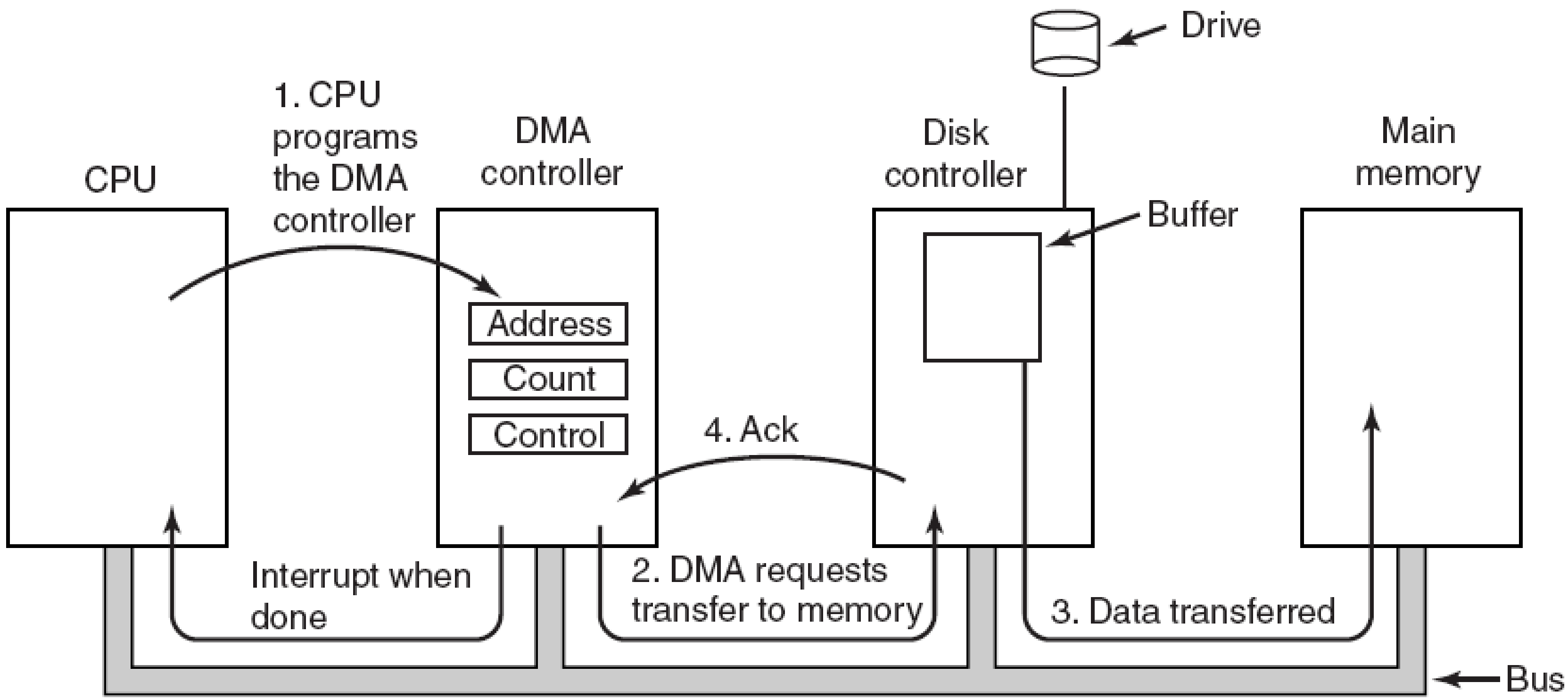
- DMA controller
  – manages transfer between device and RAM
  – integrated with device controller or separate device on motherboard



Intel

nVIDIA nForce4 integrated chipset

Direct memory access (DMA)



Step 3 uses either

- Cycle stealing – Acquire bus and transfer a bus cycle or two

OR

- Burst mode – Acquire bus and complete the transfer

Bus acquisition takes time

Goals of I/O Software

- Device independence
- Uniform naming

Woligroski 2004, *Graphics Beginner’s Guide,*Tom’s Hardware
http://www.tomshardware.com/2006/07/24/graphics_beginners/

- Error handling
  - – handle at lowest layer possible
  - – many errors are transient

Issues for I/O software

- How is transfer done?
  - Buffering
    - where to put data
    - buffer size
    - copying takes time →latency
  - Interrupt driven vs. Programmed I/O
  - Direct memory access (DMA)

I/O Software layers

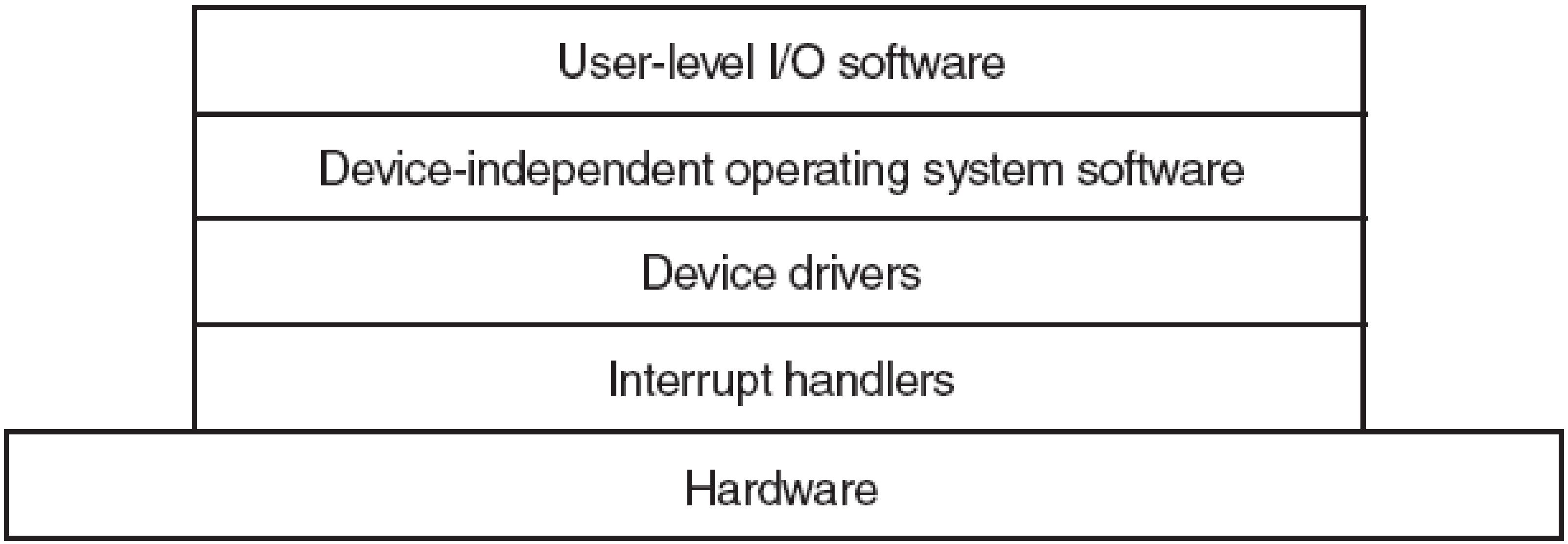
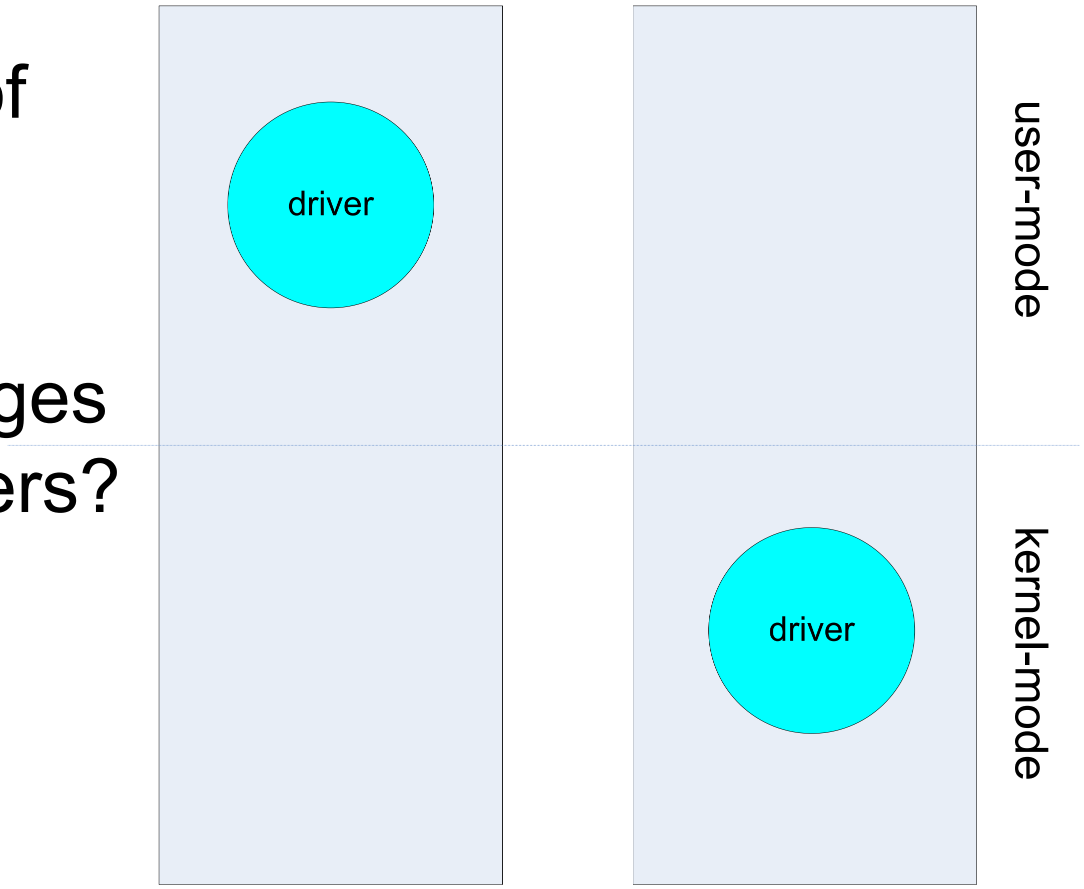


Figure 5-11. Layers of the I/O software system.

Where do device drivers live?

- Traditionally part of kernel
- Are there advantages to user-mode drivers?
- Disadvantages?



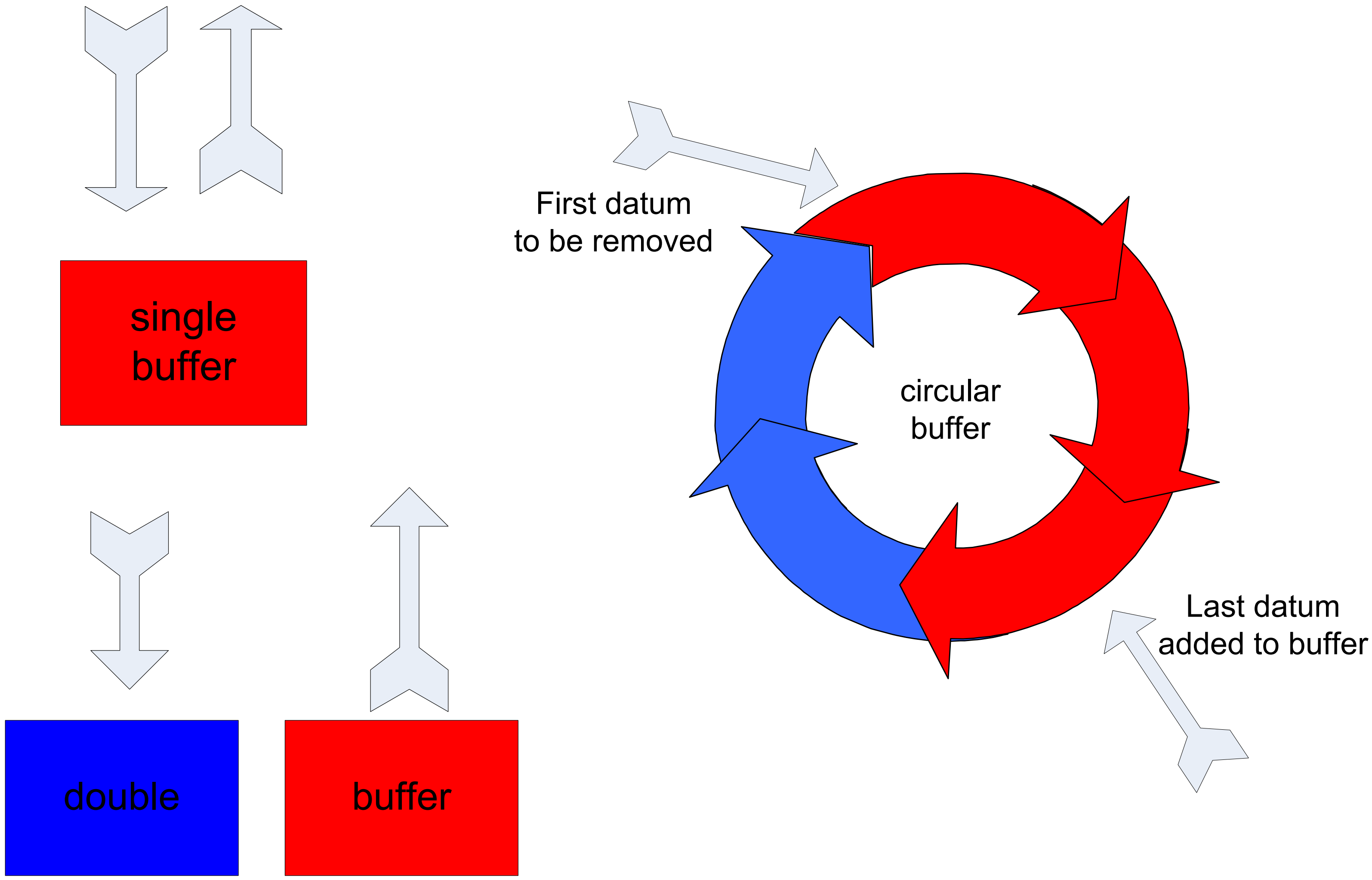
Device drivers

- Typically loaded as needed
- Implements device-independent kernel interface
- Must be robust
  - check for valid parameters
  - be able to handle an interrupt that occurs during an interrupt
- May need to support
  - hot plug
  - suspend/hibernate

The responsible OS designer and device drivers

- Provide clean abstractions
  - driver API should be as generic as possible
  - provide uniform manner to name devices
- Provide security
  - Who is allowed to access each device?
  - What permissions might they have?

Buffering schemes



Error reporting

- User error
  - invalid buffer
  - bad parameter

- Device error
  - is another try likely to fix the problem?
  - if unable to fix, report error to OS

User-space I/O Software

- Libraries
  - Provide abstraction of system calls
  - Examples: scanf/printf/cout
- Spoolers
  - Prioritize and execute requests to access devices
  - Specialized user processes called daemons handle this.

www.freebsd.org

Character based devices

- Keyboards
  - Most modern keyboard have ≤ 128 keys
    → 7 bits sufficient to encode a key
  - Each key produces a scan code, with the high order bit indicating if the key has been depressed or released
    - e.g. depress k, release k, depress i, release i, depress s, release s, depress s, release s
      Device driver maps this to kiss

Keyboards

- Consider
  - Shift depress, k depress, k release, shift release
    or
  - Shift depress, k depress, shift release, k release

    Both are what we might think of as capital K

– Drivers

- handle keycode conversions to a coding system such as ASCII or unicode
- deliver:
  - non-canonical (raw): character by character input
  - canonical (cooked): gives a line at a time

Terminals

- Character oriented output
  - Frequently need to echo keystrokes
  - Control codes: tab, newline, etc. can vary between devices

Pointing devices

- Mice
  - Device provides
    - delta x and delta y changes
    - Buttons
    - Delta wheel changes
  - Device driver
    - Determines double clicks
    - Pointer speed