CS354: Machine Organization and Programming

Lecture 26

Monday the November 02nd 2015

Section 2

Instructor: Leo Arulraj

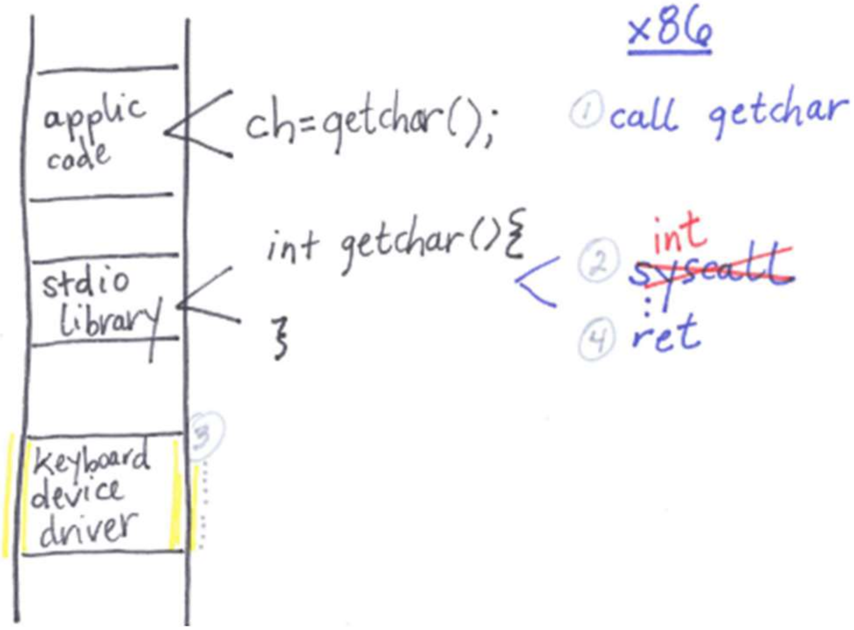
Class Announcements

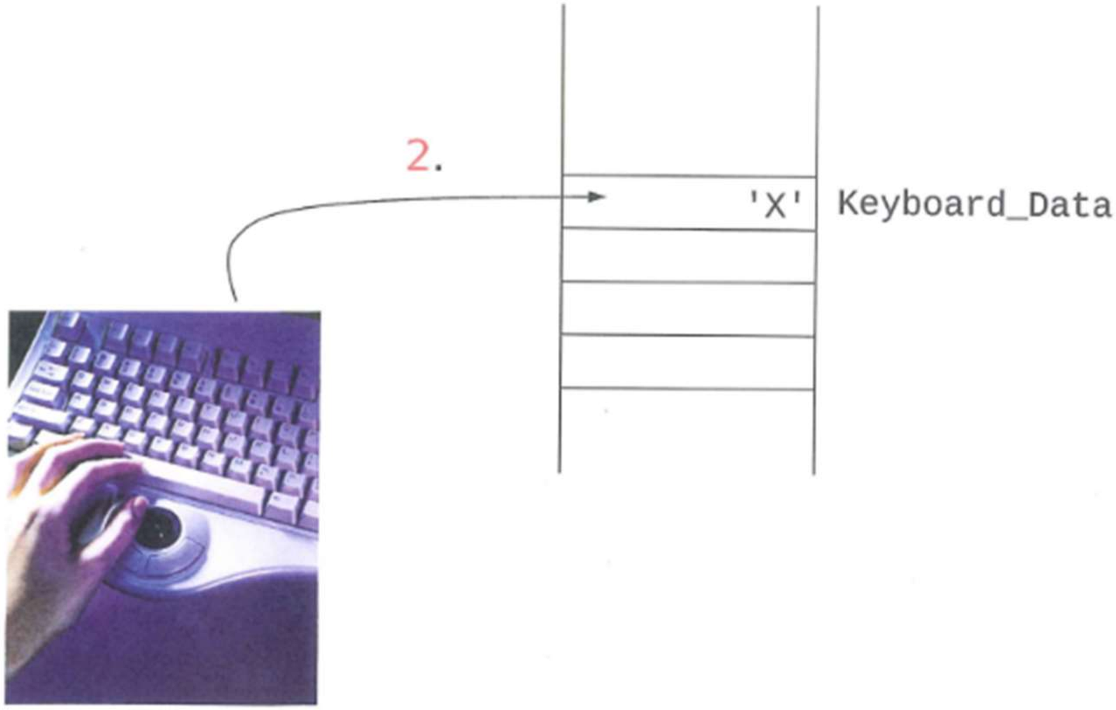
1. Programming Assignment 3 is due by 9 AM day after tomorrow (11/04 – Wednesday). As usual, you can submit it upto 48 hours after the deadline with penalties.

Lecture Overview

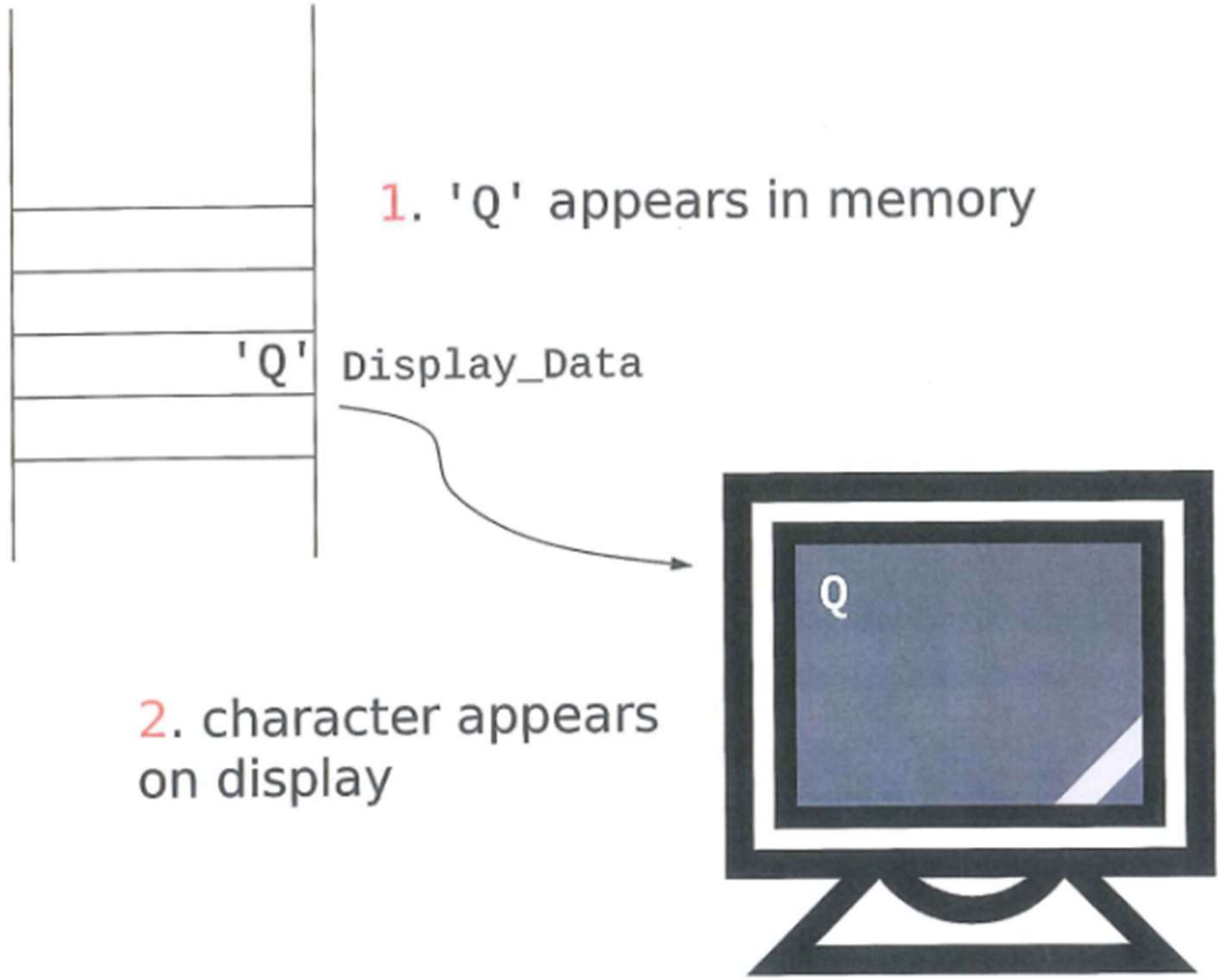
1. Interrupts and Exception
2. Intro to Processes

To control access to individual devices, we place code that communicates with the devices into the **O.S.** (this portion of the O.S. is also called the **Kernel**) and in special routines called **device drivers**.





1. user types 'X'



1. 'Q' appears in memory

2. character appears on display

The Keyboard driver code:

movl Keyboard-Data, %eax
retfrom ~~syscall~~ int

Assume this is where the ASCII char is supposed to be after ~~syscall~~ int completes.

The display driver code:

movl %eax, Display-Data
retfrom ~~syscall~~ int

What happens if user has not typed a char on the keyboard?

We want blocking input.

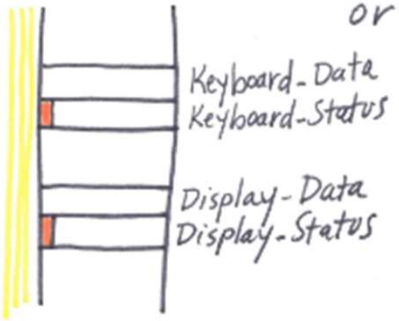
(no retfromsyscall until there is a char)

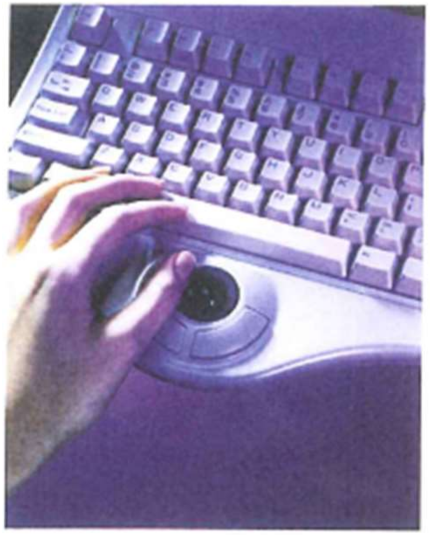
We need a status bit

1 ready

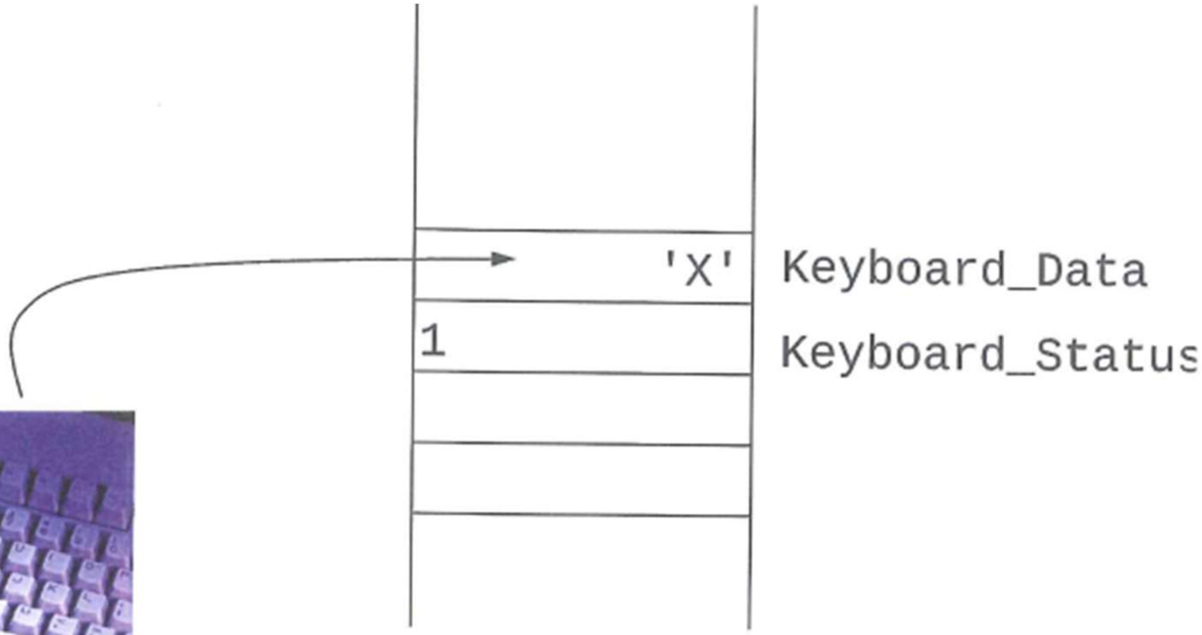
0 not ready (busy)

Place this bit into its own memory mapped word & make it the msb, so code can test for ≥ 0 or < 0.





1. user types 'X'



2a. 'X' into `Keyboard_Data`

2b. 1 into `Keyboard_Status` msb

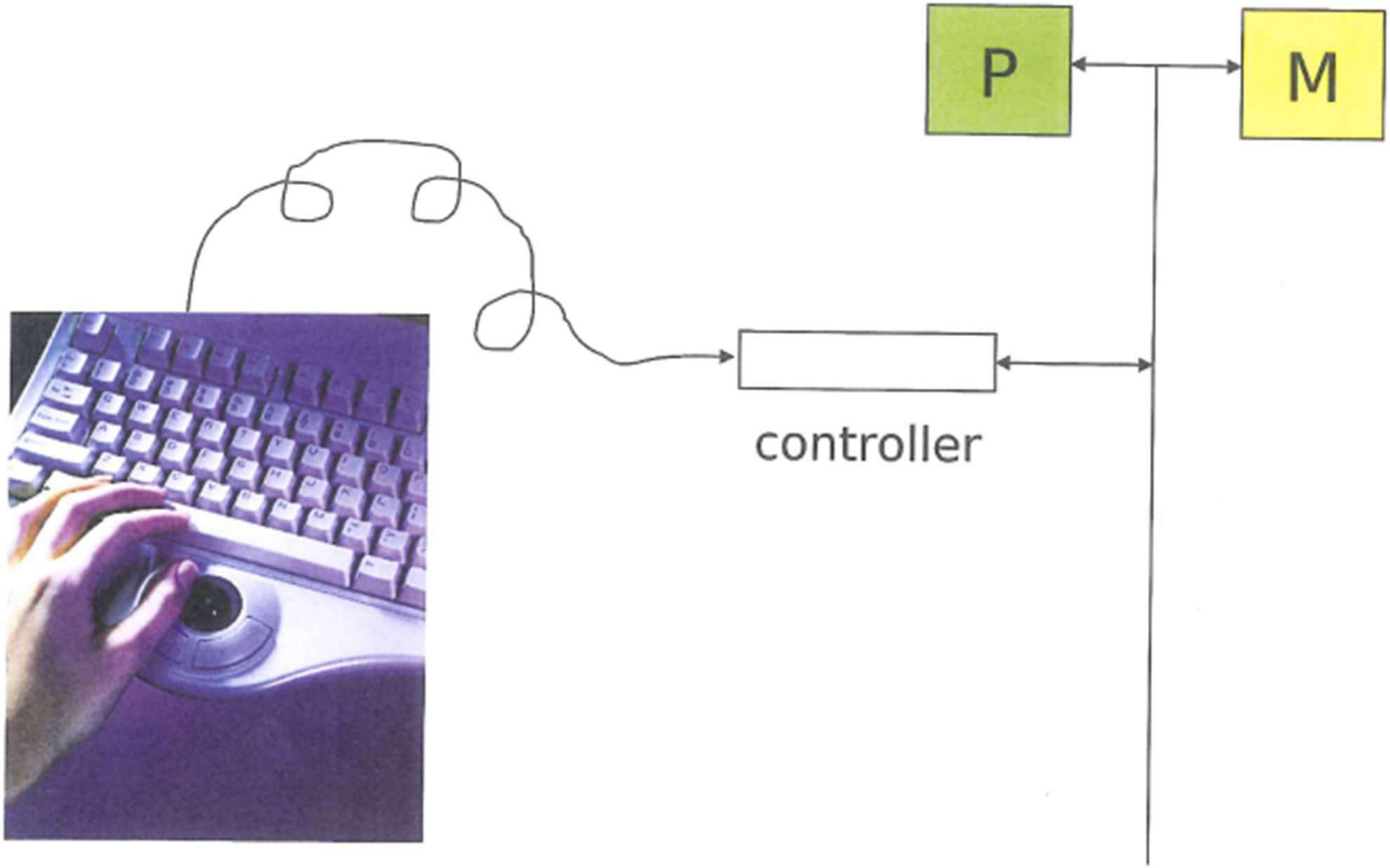
Now, driver code uses a

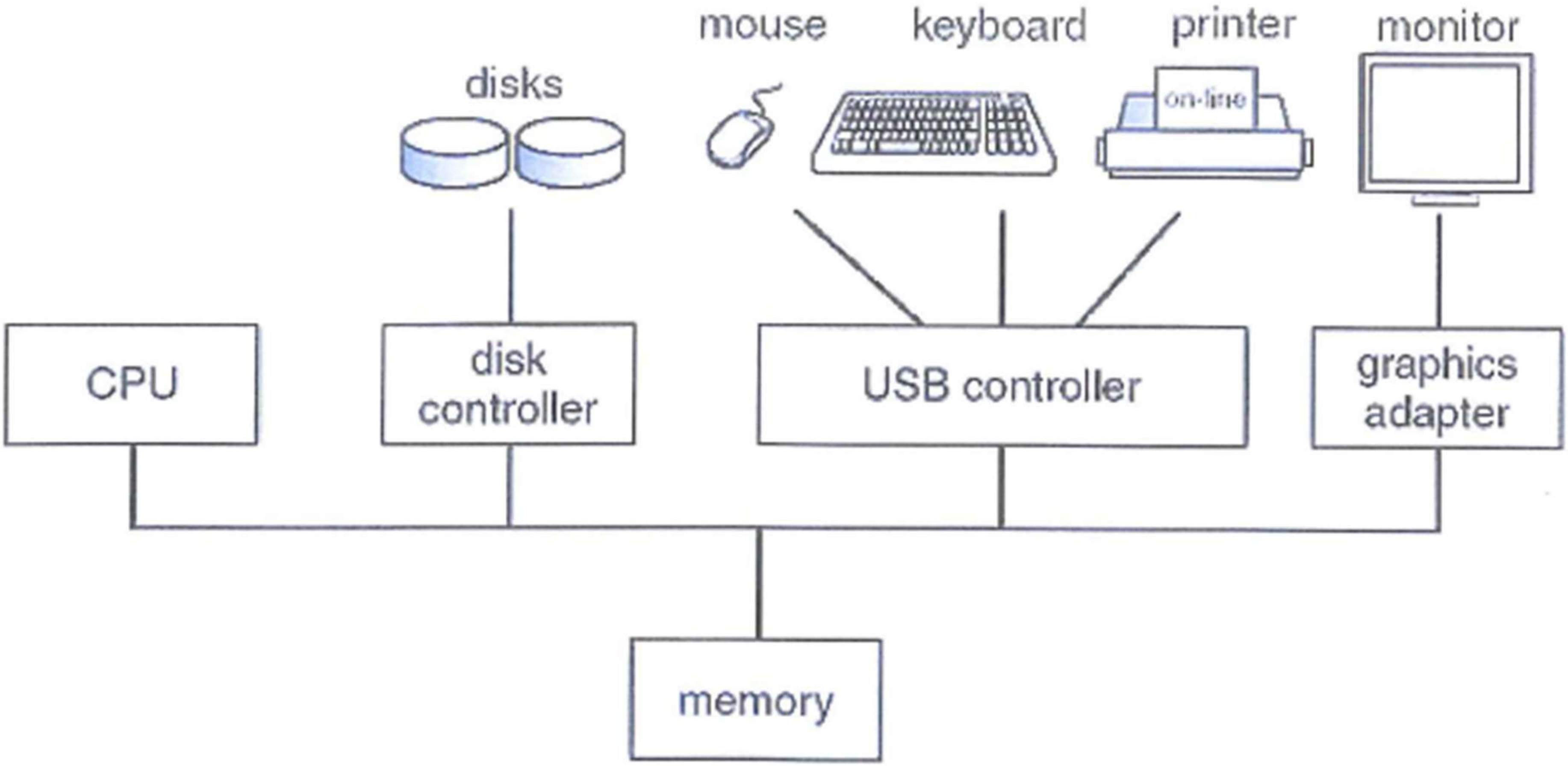
spin wait loop

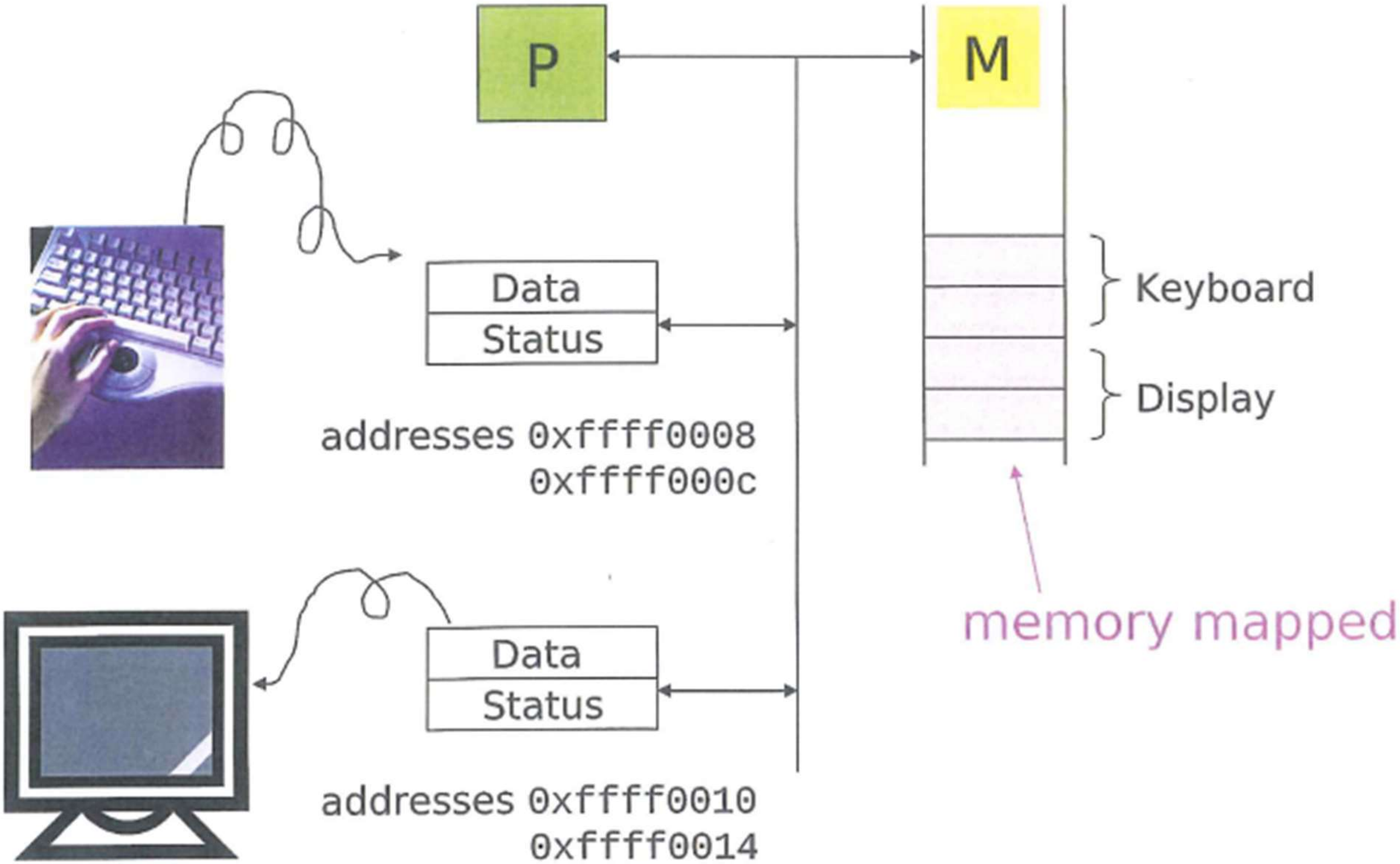
(to implement blocking I/O)

```
kb_spin:  testl  Keyboard_Status, Keyboard-Status
          jz     kb-spin
          movl   Keyboard-Data, %eax
          retfrom~~syscall~~int

disp-spin: testl  Display-Status, Display-Status
           jz     disp-spin
           movl   %eax, Display.Data
           retfrom~~syscall~~int
```


P
M
controller


mouse
keyboard
printer
monitor
disks
on-line
CPU
disk controller
USB controller
graphics adapter
memory



P

M

Data

Status

addresses `0xffff0008`
`0xffff000c`

Data

Status

addresses `0xffff0010`
`0xffff0014`

Keyboard

Display

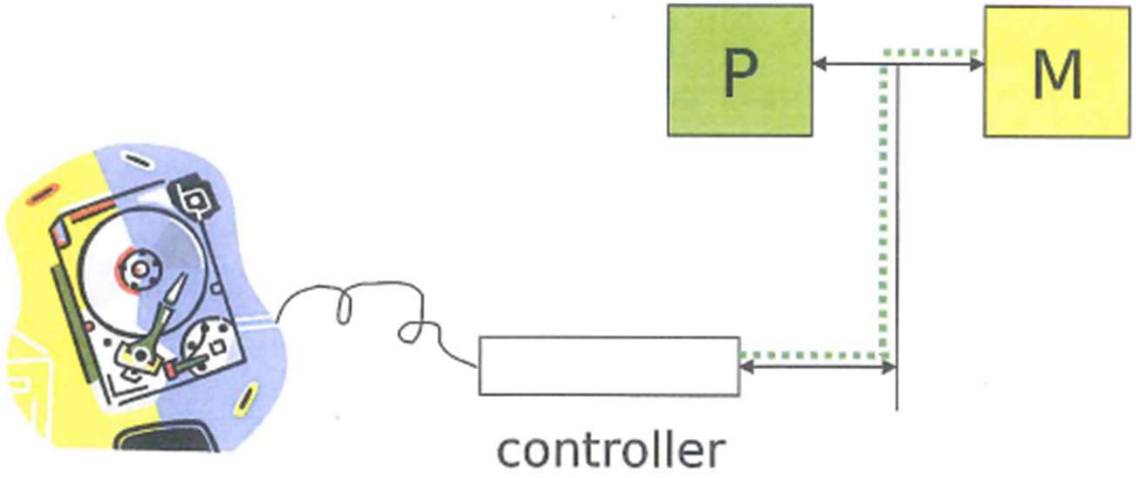
memory mapped

Byte transfers are OK,

But, what about faster devices that like to transfer more than a byte ?

the solution: DMA

Direct Memory Access



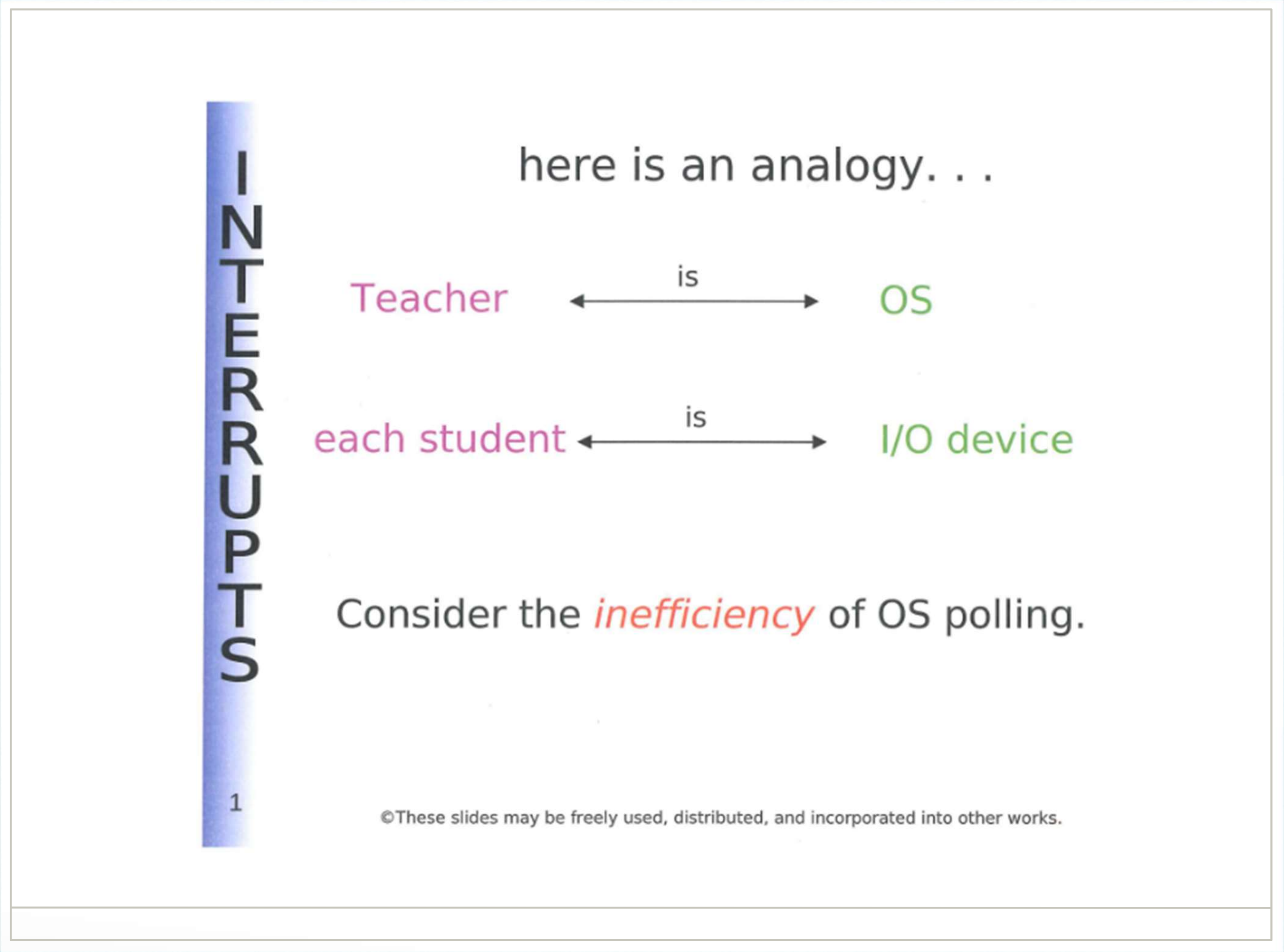
Issue for spin wait loop implementations:

One byte *only* in `_Data` has the potential for an incorrect result.

For example, if the user types 2 characters on the keyboard before `getchar()` is called.

The needed fix introduces a kernel-maintained queue for each device.

Then, the kernel polls to check status bits and handle any ready devices.



INTERRUPTS

here is an analogy. . .

Teacher ← is → OS

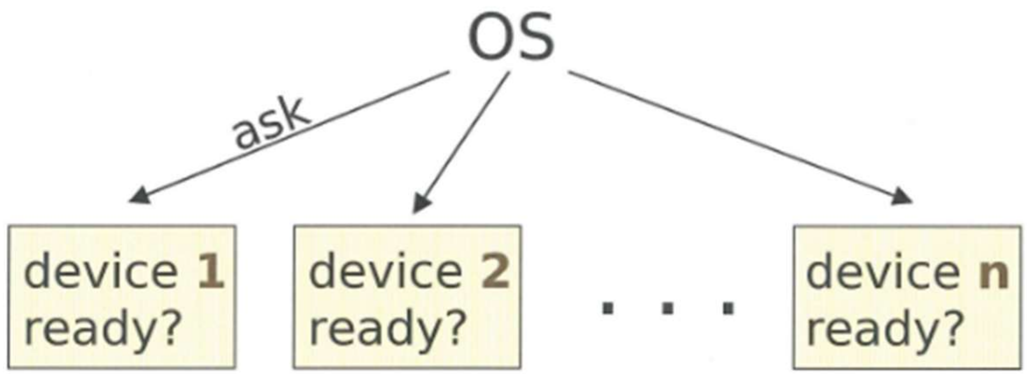
each student ← is → I/O device

Consider the *inefficiency* of OS polling.

INTERRUPTS

Because polling is *so inefficient*,

instead of



OS

ask

device **1** ready?

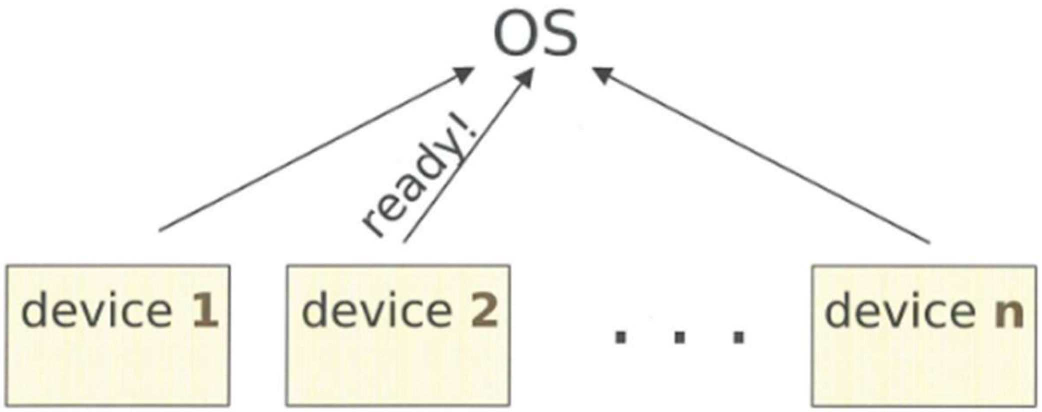
device **2** ready?

. . .

device **n** ready?

INTERRUPTS

Turn the situation upside down



OS

ready!

device **1**

device **2**

. . .

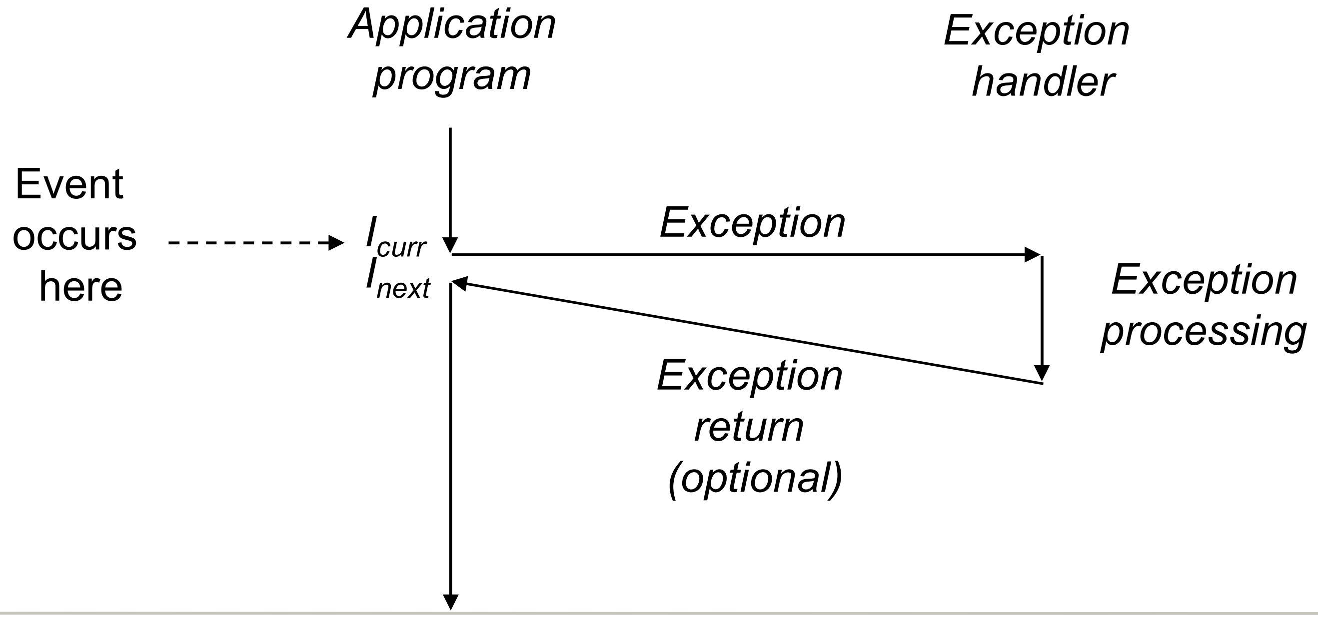
device **n**

INTERRUPTS

Anatomy of an exception

An exceptions is an abrupt change in control flow.

Examples: div by 0, arithmetic overflow, page fault, I/O request completes, Ctrl-C



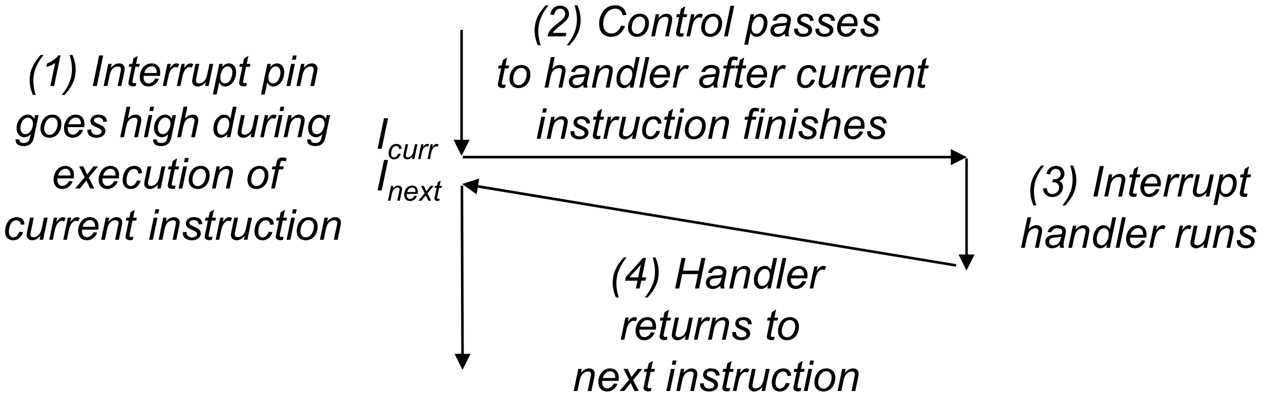
Classes of Exceptions

1. Interrupts (Asynchronous): Always return to next instruction.
2. Traps & System Calls (Synchronous): Always return to next instruction.
3. Faults (Synchronous): Might return to next instruction.
4. Aborts (Synchronous): Never returns

Interrupts

Examples of Interrupts:

- Timer interrupt
- Arrival of a packet from a network
- When a key is pressed on the keyboard
- When the mouse is moved



(1) Interrupt pin goes high during execution of current instruction

I_{curr}

I_{next}

(2) Control passes to handler after current instruction finishes

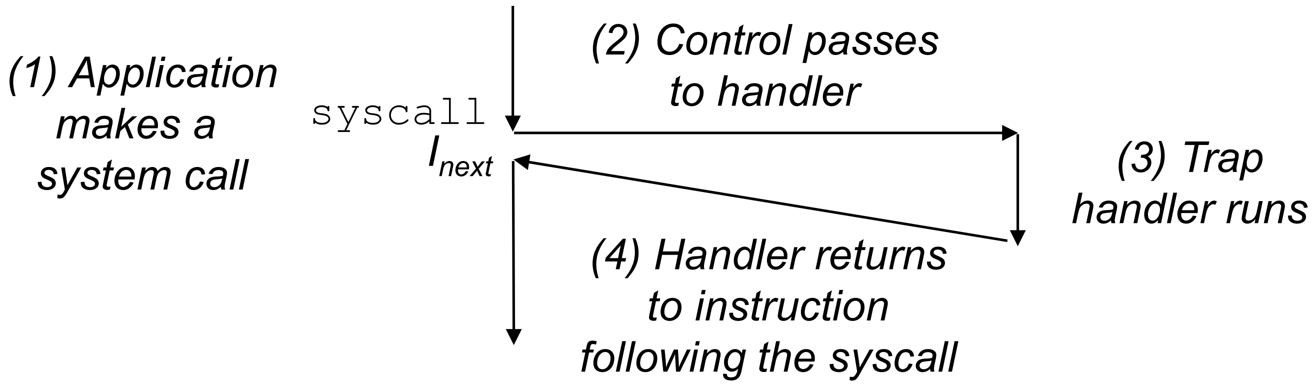
(3) Interrupt handler runs

(4) Handler returns to next instruction

Traps

Traps are intentionally issued by executing an instruction.

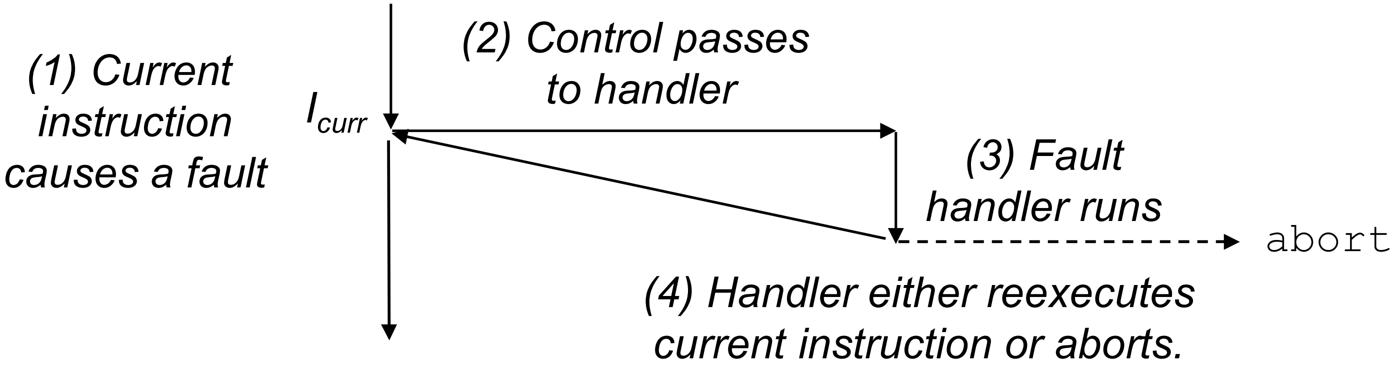
Example: System calls



Faults

Faults result from error conditions that might be correctable.

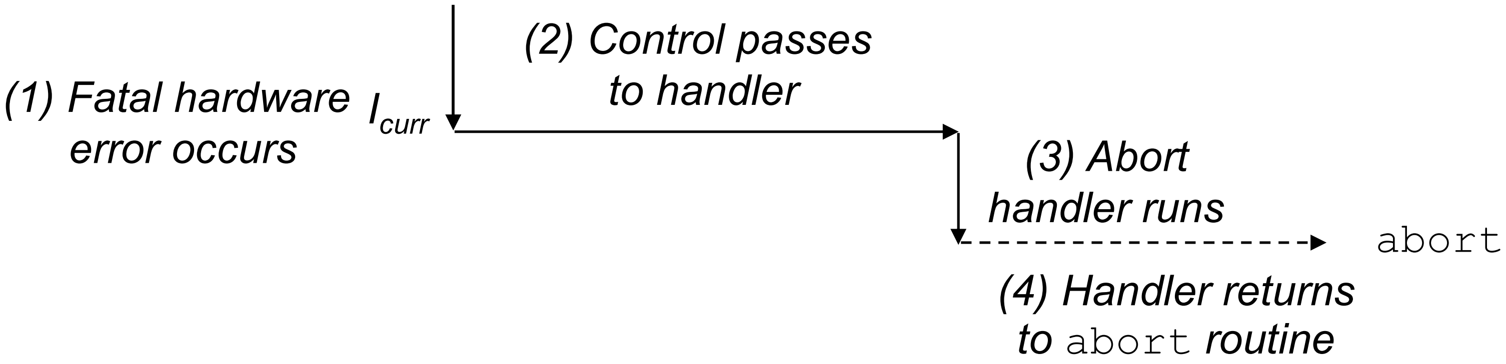
Examples: Page fault, Divide error



Aborts

Aborts result from unrecoverable fatal errors.

Example: parity errors due to DRAM bit corruption



(1) Fatal hardware error occurs I_{curr}

(2) Control passes to handler

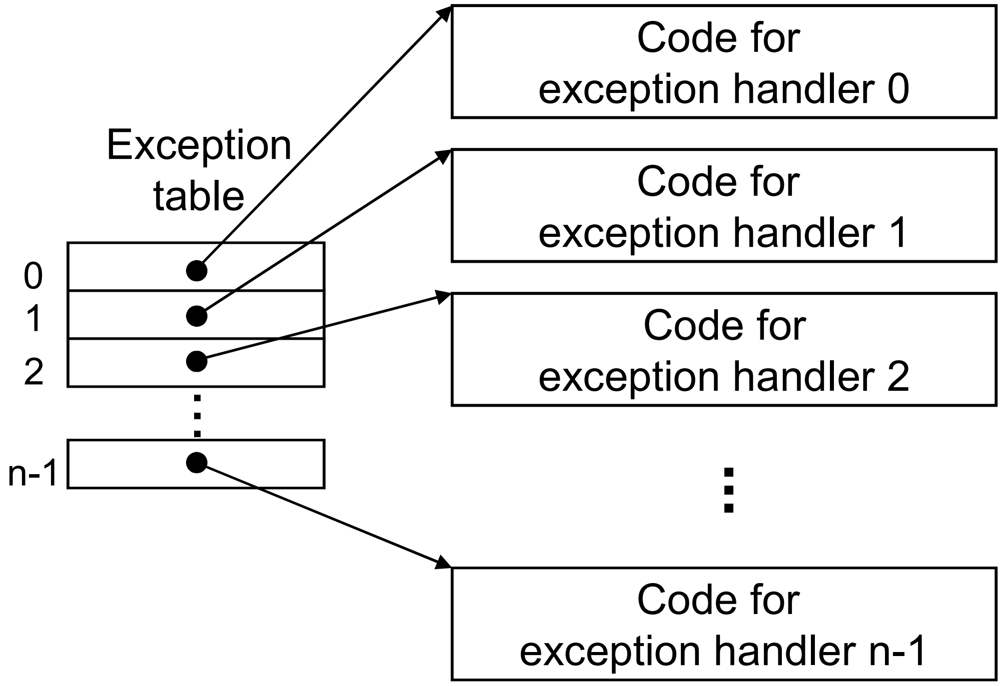
(3) Abort handler runs

`abort`

(4) Handler returns to `abort` *routine*

Exception Table

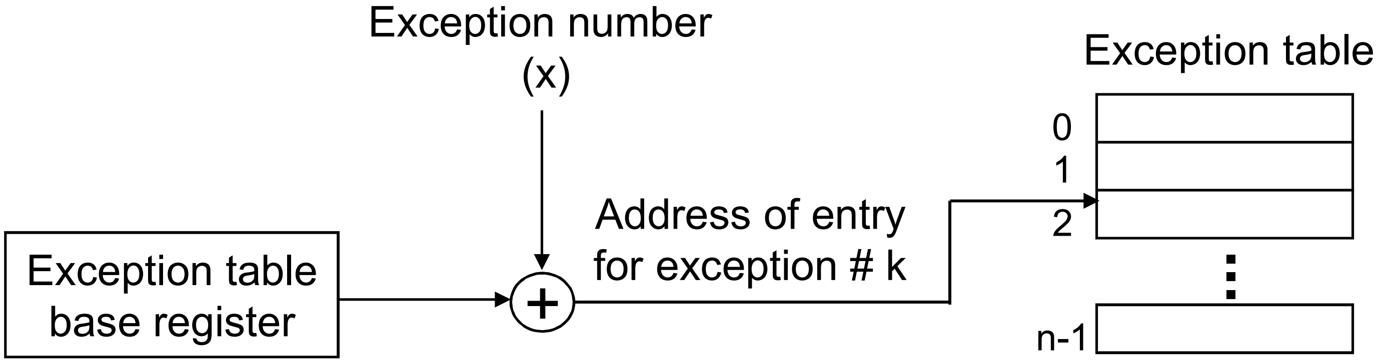
Exception table



Exception Table lookup

Exception is similar to procedure calls except for some important differences:

- Return address is not the next instruction always
- Push EFLAGS register also onto kernel stack
- Run exception handler in kernel mode



IA32 Exception Table

From CSAPP text book:

Exception Number	*Description*	*Exception Class*
0	Divide error	Fault
13	General protection fault	Fault
14	Page fault	Fault
18	Machine check	Abort
32-127	OS-defined	Interrupt or trap
128 (0x80)	System call	Trap
129-255	OS-defined	Interrupt or trap

Rather important, but not covered in textbook:

If running a handler, and a new interrupt request arrives, what should happen?

- ☆ Continue on, complete handling of current interrupt, then, when done, deal with new request?

  (probably) nonreentrant

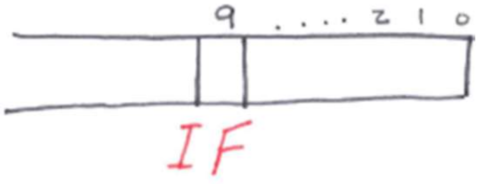
- ☆ Interrupt the handling of this interrupt?

  reentrant

Every architecture has a control bit which identifies whether the F+E cycle is paying attention to IRQs. Called interrupt enable

on x86:

EFLAGS



1 enabled
0 disabled

fetch + execute cycle:

⓪ if IF=1 and interrupt requested, go handle it

① fetch instr
② PC update
③ decode
⋮

Consider the x86 instruction:

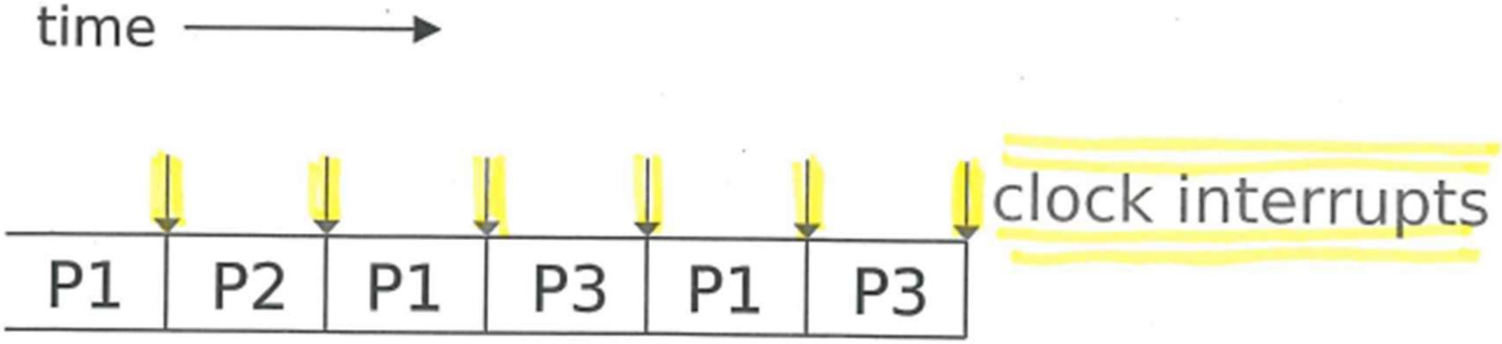
`cli` clear IF

What happens if an application includes this `cli` *instruction?*

Irrelevant (to this discussion) x86 instruction:

`sti` set IF

OS relies on
clock interrupts
to allocate processing time. . .



time

clock interrupts

P1 | P2 | P1 | P3 | P1 | P3

As the clock interrupts, the kernel runs, and it decides which program runs next.

Clarified instruction:

`cli` clear IF *if CPL is high enough, otherwise trap*

Does `sti` also need to be a privileged instruction?

Keep IF=1 while CPL=00.
(So, applications can always be interrupted

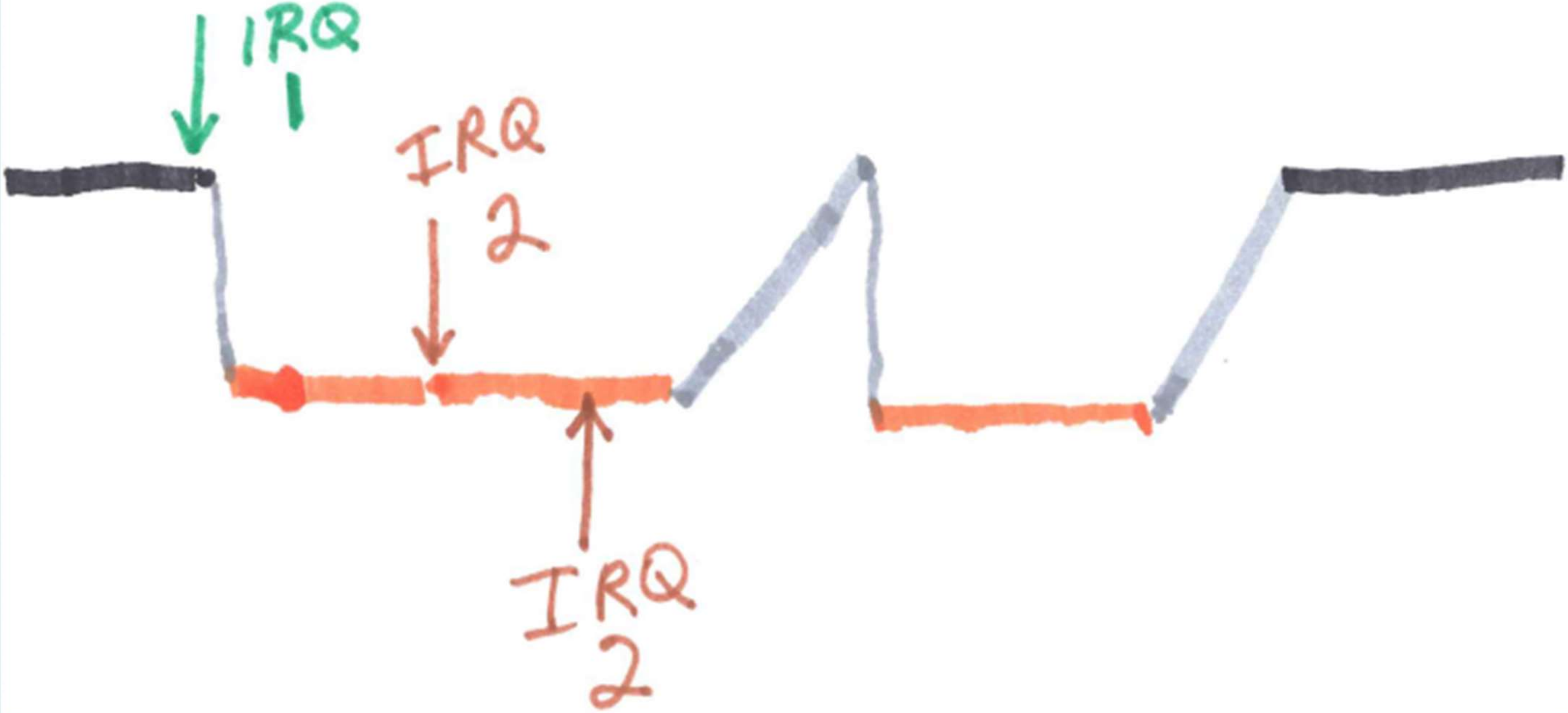
HW must disable interrupts while saving state + at least until first instruction within handler is fetched.

Better Definitions:

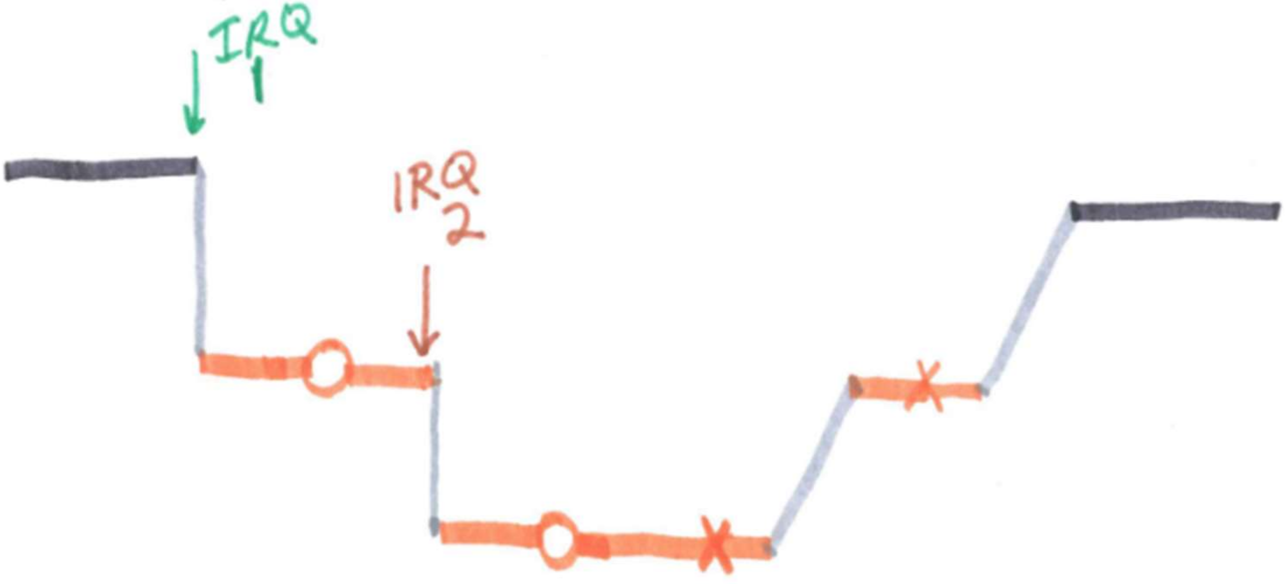
nonreentrant IF=0 the entire time a handler runs

reentrant interrupts may be reenabled while handler runs (usually only for higher priority requests)

Nonreentrant timeline



Reentrant timeline



dev 2 interrupts are higher priority