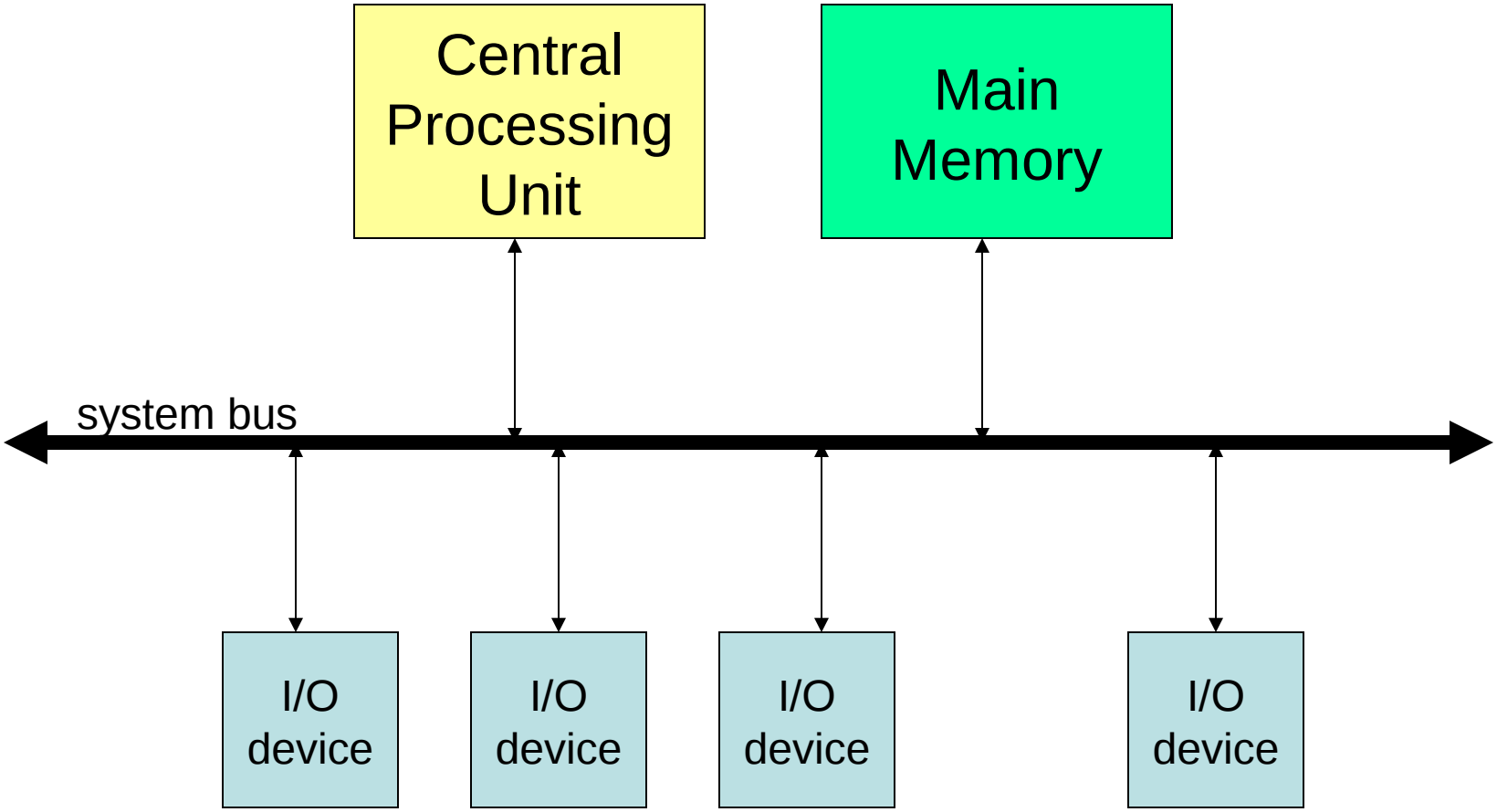
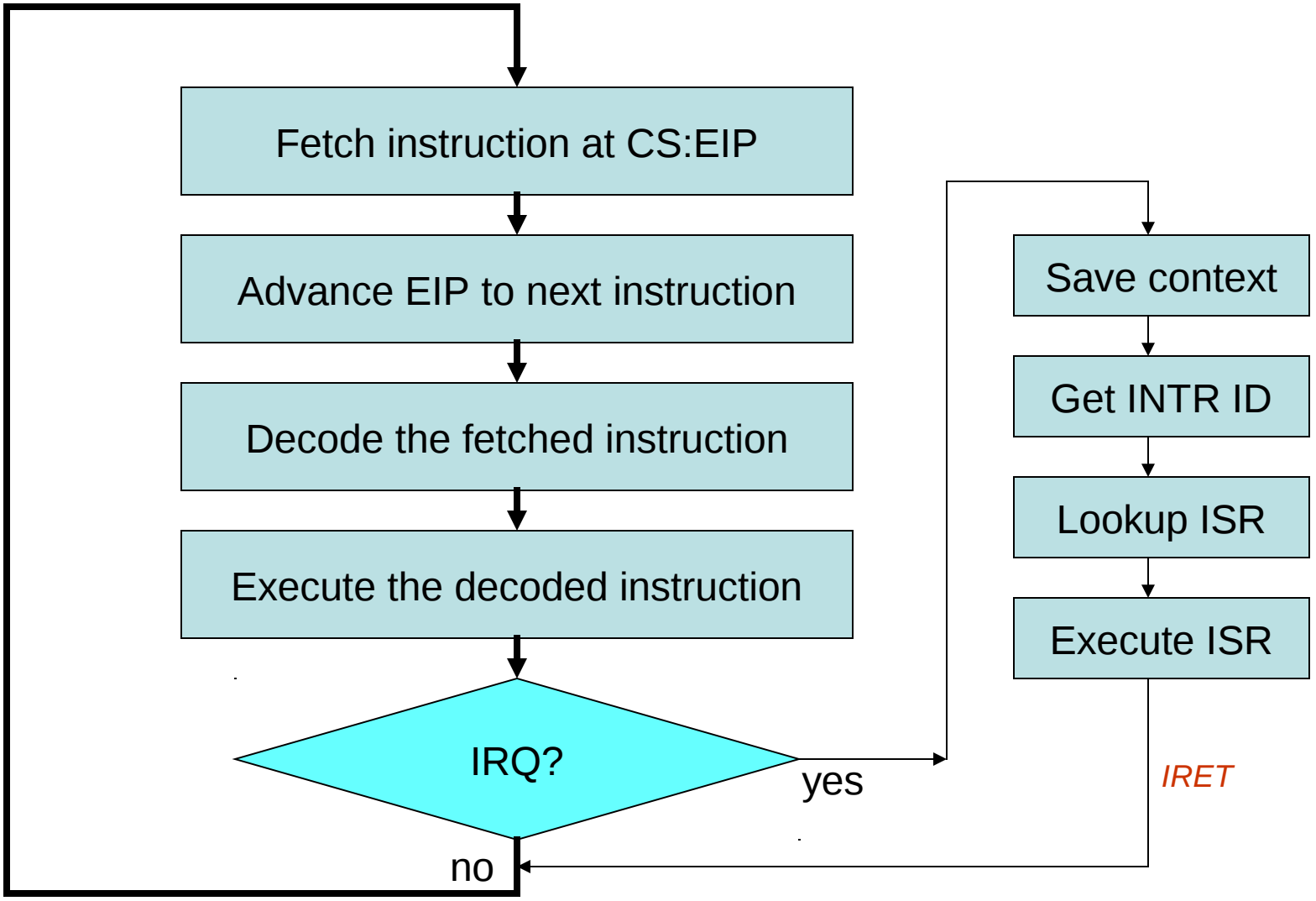
Timeout for some demos

An instructive look at how the
Linux Operating System
sets up system data-structures

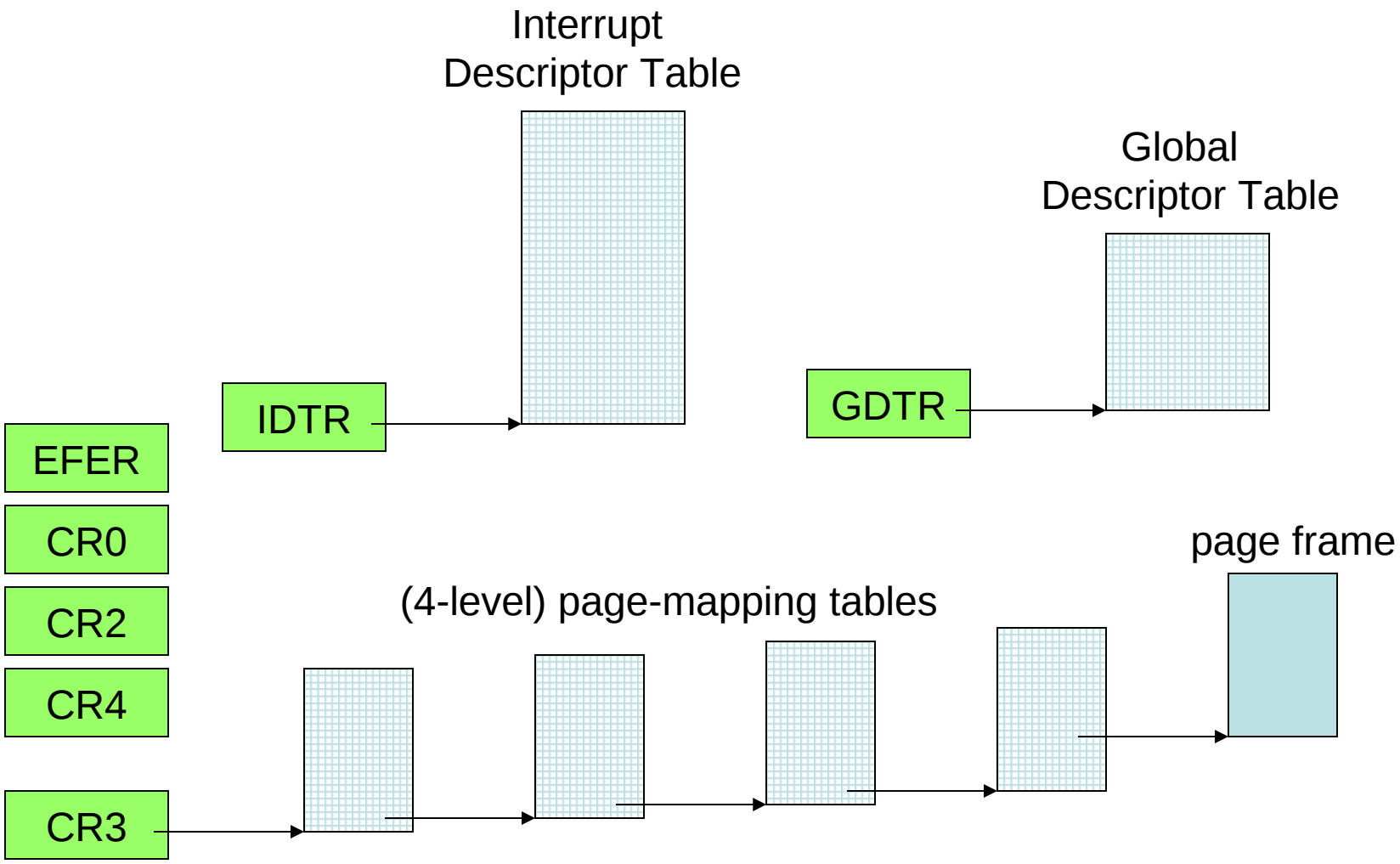
Simplified Block Diagram



CPU's 'fetch-execute' cycle



System registers and tables



‘activity.s’

- This is a ‘boot-time’ application program
- You assemble, link, install, and reboot
- It runs entirely in x86 ‘real-mode’
- It maintains an array of counters
- It ‘hooks’ all of the 256 interrupt-vectors
- Every interrupt-occurrence gets counted
- The counters are continuously displayed

‘activity.cpp’

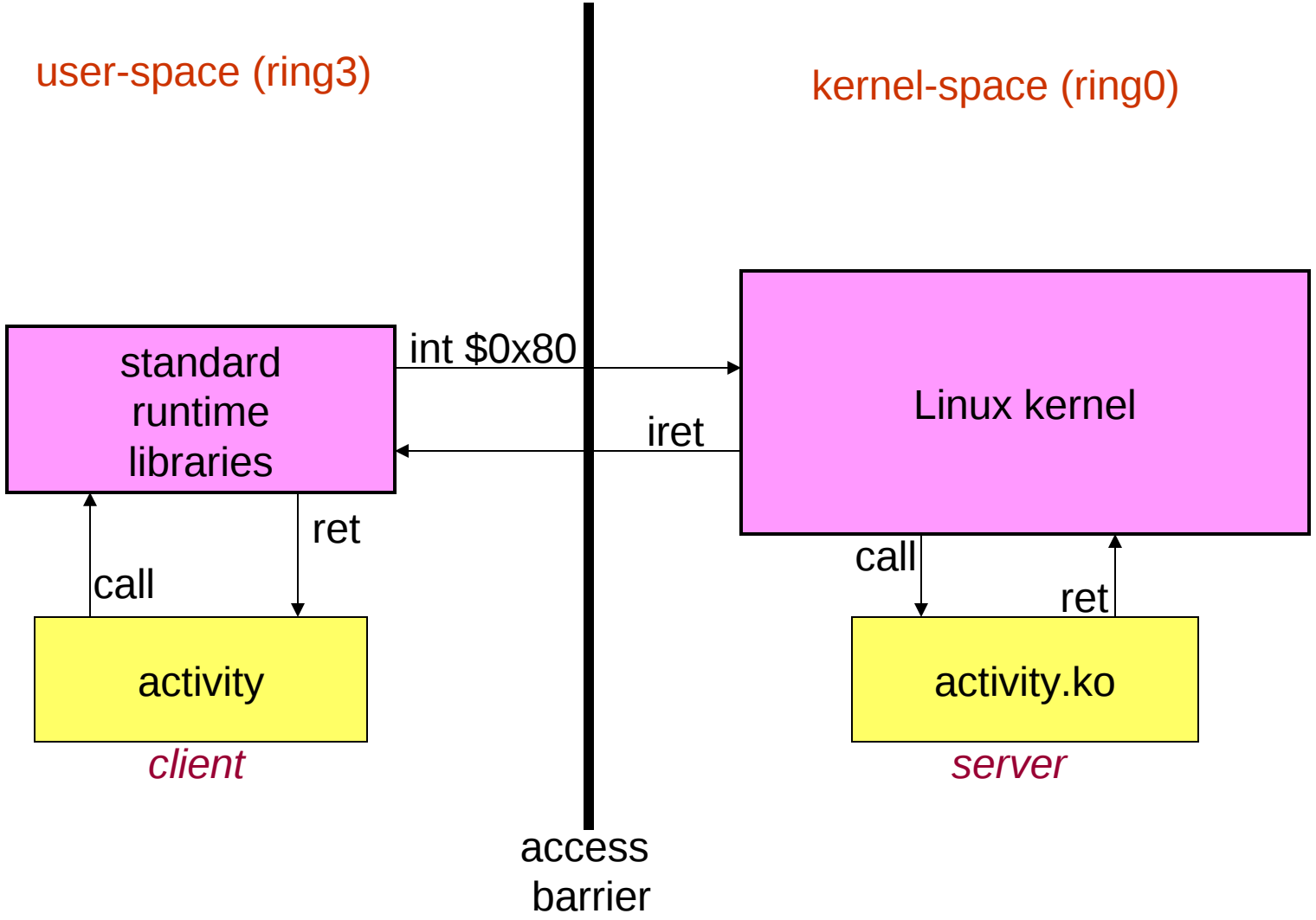
- This is a Linux application program
- It’s similar to our previous demo ‘activity.s’
- It shows us all the interrupts which occur
- But it runs in ‘protected-mode’ at ring3
- So it can’t modify the IDT directly
- Instead it uses the services of an LKM
- The LKM source-file is called ‘activity.c’

‘mmake.cpp’

- This program makes it easy to compile an LKM for version 2.6 of the Linux kernel
- If your LKM is named ‘myLKM.c’, then you compile it using: $ mmake myLKM
- The result is a kernel object (‘myLKM.ko’)
- You install the kernel object with ‘insmod’, and you remove it with ‘rmmod’, like this:

```
$  /sbin/insmod  myLKM.ko
$  /sbin/rmmod  myLKM.ko
```

Overview of the LKM



Device-driver LKM

- Our ‘dram.c’ module provides services
- Lets applications ‘read’ physical memory
- Lets applications find the memory’s size
- Works with 32-bit Linux or 64-bit Linux
- SysAdmin must setup device-file node

‘showgdt.cpp’

- This application shows the Linux GDT
- It uses services of the ‘dram.ko’ driver
- You can try it in Kudlick classroom
- You can try it on your ‘anchor’ machine

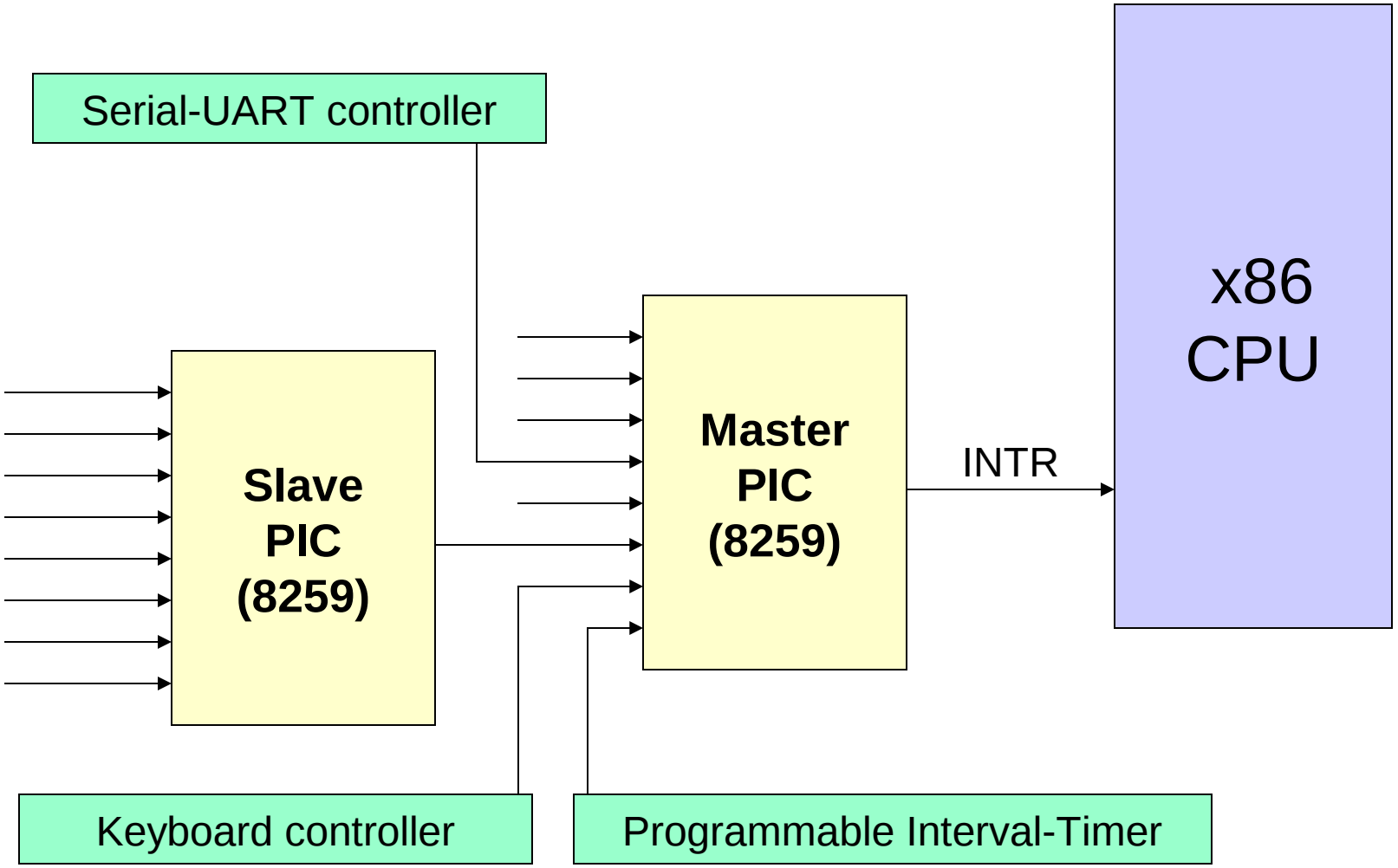
‘showidt.cpp’

- This application shows the Linux IDT
- It uses services of the ‘dram.ko’ driver
- You can try running it on ‘anchor’ machine
- You will see all 256 IDT gate-descriptors!
- (But it’s designed only for 64-bit Linux)

pseudocode

- We wrote an algorithm-description (using pseudocode) for UART’s THRE interrupt when you want to implement ‘redirection’ of console output to a remote terminal via the serial null-modem cable
- We implemented our algorithm in a demo program (named ‘redirect.s’) which runs in 32-bit protected-mode

Two Interrupt-Controllers



A ‘rapid prototype’ tool

- We wrote a ‘newapp64.cpp’
- It creates six pages of ‘boilerplate’ code for a boot-time assembly language application that will execute 64-bit code
- It includes ‘isrGPF’ (for debugging aid)
- We can use it to quickly create future demo programs that explore EM64T

In-class exercise

- Use the ‘newapp64’ tool to create a ‘test’ program that you will install and execute on an ‘anchor’ machine – but first insert into the 64-bit portion of the code some type of invalid instruction which will trigger a General Protection Exception – so you can see what debugging information gets displayed by the ‘isrGPF’ fault-handler