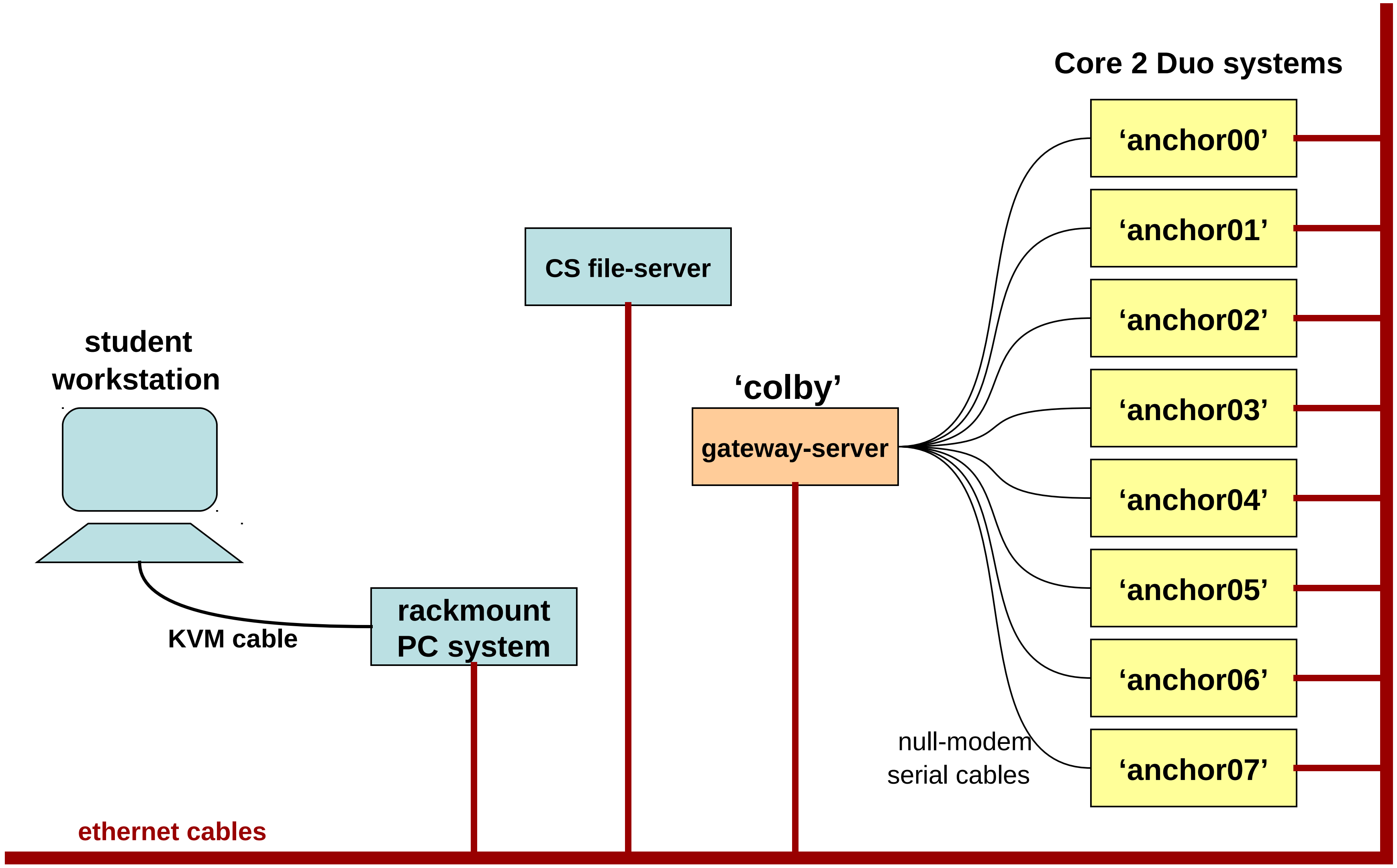
Our first 64-bit ventures

How to remotely access our new Core-2 Duo platforms for some exploration and programming

Recall our system setup



You can login via the LAN

- From a classroom or CS Lab machine, you can use Linux ‘ssh’ command; e.g.:

```
$  ssh  anchor07
```

- All of your files will be there, remotely mounted via the Network File System
- You can use the customary editors and compilers or assemblers, and you can execute your programs or shell-scripts

remotely ‘rebooting’

- If you ever need to ‘reboot’ one of these new Core-2 Duo machines, you can do that from our classroom or the CS Lab
- But you will need to connect via the alternate ‘gateway’ machine to watch screen-output during the reboot stage
- Then you can select GRUB menu-items that let you ‘boot’ alternative systems

If you need ‘boot-time’ access

- From a classroom or CS Lab machine, you can use Linux ‘ssh’ command:

```
$  ssh  colby
```

- Then use the Linux ‘telnet’ command, e.g.:

```
$  telnet  localhost  2007
```

- Now you are connected to ‘anchor07’ via the serial-port null-modem link and can login normally (with username, password)

Name-to-number associations

- The new Core-2 Duo machine-names, and their corresponding ‘telnet’ port-numbers:

‘anchor00’	→	2000
‘anchor01’	→	2001
‘anchor02’	→	2002
‘anchor03’	→	2003
‘anchor04’	→	2004
‘anchor05’	→	2005
‘anchor06’	→	2006
‘anchor07’	→	2007

Ordinary ‘rebooting’

- As long as your ‘anchor’ machine’s OS is working, you can reboot it using this Linux command: `$ sudo reboot`
- But if your ‘anchor’ machine gets ‘hung’ as a result of some unintended program ‘bug’ and you need to reboot it while the Linux operating system is non-responsive, then you can do it from ‘colby’ using ‘telnet’

Emergency ‘rebooting’

- Be sure you are logged into the ‘colby’ gateway-server, and type this command:

```
$  telnet  localhost  2222
```

- When the telnet-program prompts you for a command, type this:

```
$ telnet>  Reboot  8
```

- This reboots ‘anchor07’ (You can adjust the number for other ‘anchor’ machines)

Name-to-number for ‘reboot’

- The new Core-2 Duo machine-names, and their corresponding reboot-numbers:

‘anchor00’	→	Reboot 1
‘anchor01’	→	Reboot 2
‘anchor02’	→	Reboot 3
‘anchor03’	→	Reboot 4
‘anchor04’	→	Reboot 5
‘anchor05’	→	Reboot 6
‘anchor06’	→	Reboot 7
‘anchor07’	→	Reboot 8

‘Quirks’

- When you type the ‘telnet’ command to reboot a machine, you may find that you have to type it more than once
- Whenever you want to disconnect from a serial-port link between ‘colby’ and one of the ‘anchor’ machines, you can do it by typing the key-combination: <CTRL>-’]’
- You can exit from ‘telnet’ by typing ‘quit’

Code-fragments

- Here is an often-needed code-fragment in assembly language ‘systems’ programs:

```
# converts the 32-bit value in EAX to a string of 8 hex numerals at DS:EDI
eax2hex:  .code32
          pushal                                # preserve register-values
          mov       $8, %ecx                    # setup numeral-count in ECX
nxnyb:    rol       $4, %eax                    # rotate next nybble into AL
          mov       %al, %bl                    # copy nybble-pair into BL
          and       $0xF, %ebx                  # mask out all but lowest nybble
          mov       hex(%ebx), %dl              # lookup the nybble’s numeral
          mov       %dl, (%edi)                 # put numeral into output buffer
          inc       %edi                        # and advance the buffer-pointer
          loop      nxnyb                       # go back for another nybble
          popal                                 # restore the saved registers
          ret                                   # return control to the caller
hex:      .ascii    “0123456789ABCDEF”          # array of hex numerals
```

How do we use ‘eax2hex’?

- Here’s how we modify ‘eflags.s’ to show the register-value in hexadecimal format

```
            .section  .data
msg:        .ascii    “\n  EFLAGS=“
buf:        .ascii    “xxxxxxxx  \n”
len:        .int      . - msg

            .section  .text
_start:     pushfl
            pop       %edx

            mov       %edx, %eax
            lea       buf, %edi
            call      eax2hex
#   the remainer of ‘eflags.s’ may be kept unchanged
```

In-class exercise #1

- Try logging onto an ‘anchor’ via the Local Area Network, using the ‘ssh’ command (Your instructor will assign you to one of our new ‘anchor’ machines for your use)
- Compile and execute the ‘typesize.cpp’ demo-program (from our website), and compare its screen output with what you see when you run it on a classroom PC

In-class exercise #2

- Make a copy of the demo-program from our course website named ‘eflags.s’, but use ‘rflags.s’ as your name for the copy
- Insert the directive .code64 at the top
- Now edit ‘rflags.s’ so that it uses 64-bit register-names, memory-addresses and opcode-suffixes, instead of 32-bit ones (e.g., change ‘pushfl’ to ‘pushfq’, and change ‘%edx’ to ‘%rdx’, etc.).

In-class exercise #3

- Modify your ‘rflags.s’ program so that it would display the value in the RFLAGS register as a 16-digit hexadecimal value when executed on 64-bit Linux machines