CS 3650 Computer Systems – Spring 2023

Introduction to Computer Systems

Week 1

* Acknowledgements: created based on Christo Wilson, Ferdinand Vesely, and Alden Jackson's lecture slides for the same course.

Pre-Class Warmup

- Take a moment and introduce yourself to someone next to you. They are going to be your colleagues for the next 14 weeks!

  "e.g. What is your name? What is the worst bug you have ever encountered?"

- Will your classmate(s) and you be the next:
  - Jobs-Woz
  - Gates-Allen
  - Frances Allen
  - Turing-Church
  - Radhia and Patrick Cousot







Course Instructor

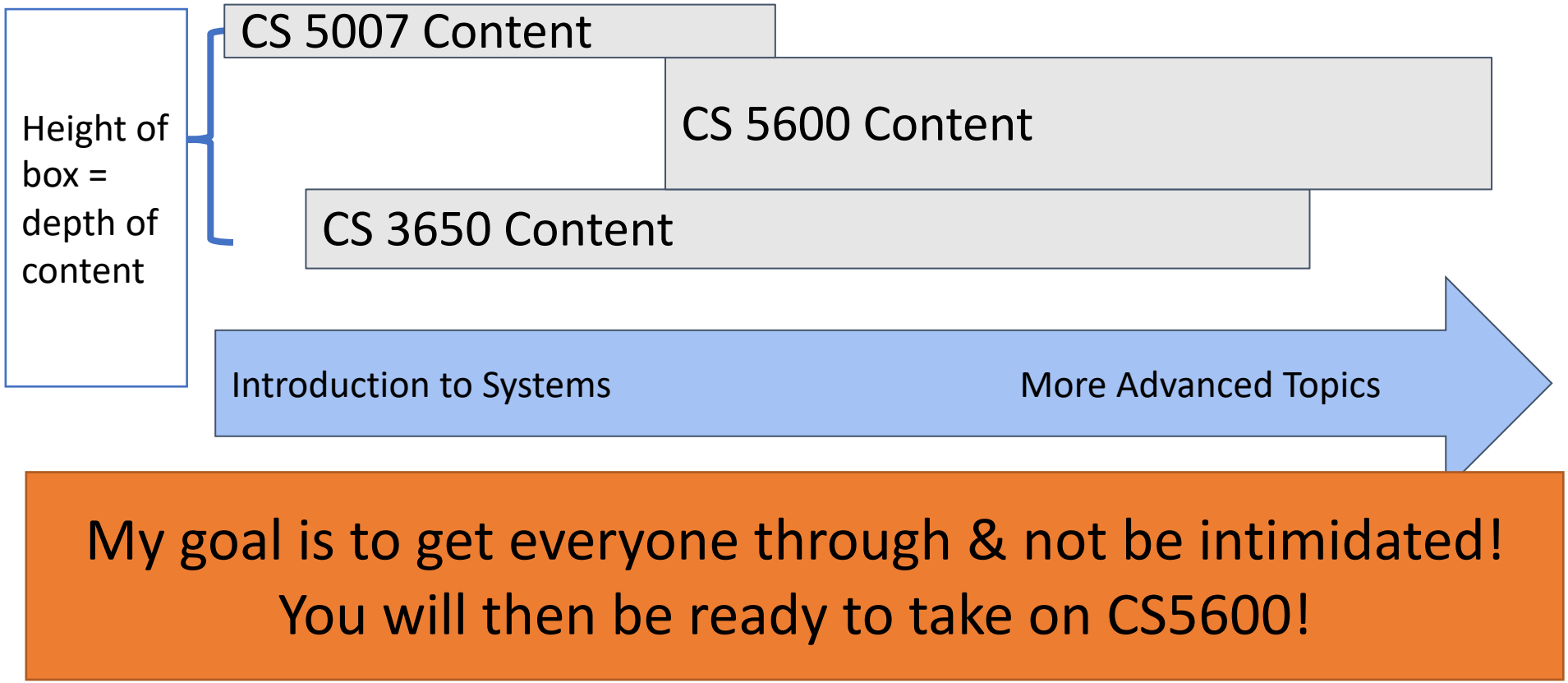
- Professor Ji-Yong Shin
- Email: j.shin@northeastern.edu

- Works on systems and formal methods
- I grew up in South Korea
- Moved to the U.S. to pursue Ph.D. degree in CS at Cornell Univ.
  - Wireless data center networks
  - Cloud storage systems
  - Distributed systems
- Associate Research Scientist at Yale University
  - Formal verification of distributed systems
  - Some work done at Yale is implemented in Facebook's system
- Many internship experience
  - Microsoft Research (Asia, Silicon Valley, Redmond)
  - IBM TJ Watson Research
  - Google

What is this course about?

Computer Systems courses at Khoury College

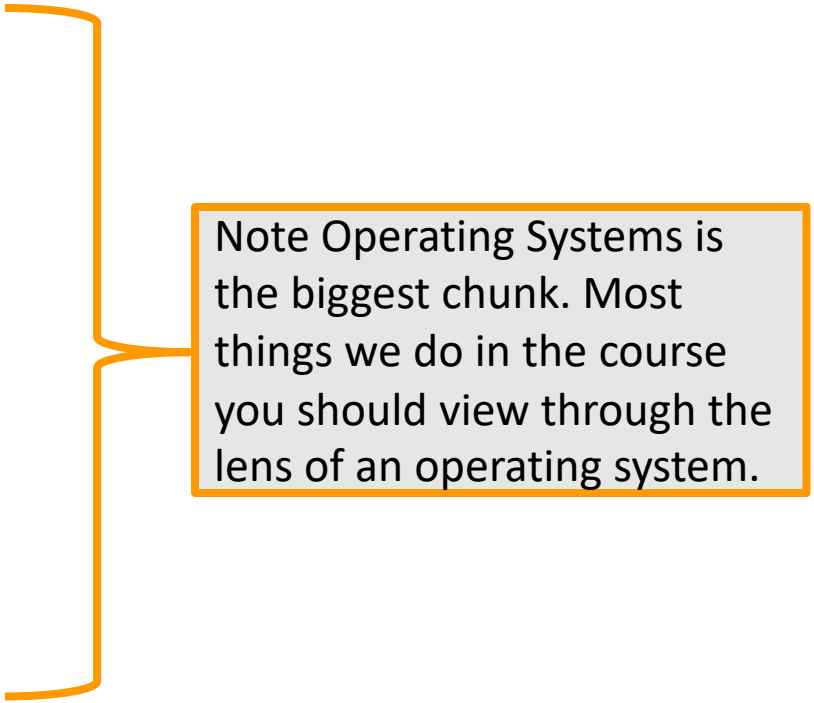
- Three courses with the same name!
- A rough visualization of where the course is in the curriculum



My goal is to get everyone through & not be intimidated!
You will then be ready to take on CS5600!

Roughly Speaking this course has a few ‘modules’

- Computer Systems Fundamentals
  - Terminal, C, Assembly, and Compilers
- Virtualization
  - Processes
- Computer Architecture
  - Memory/Cache/etc
- Concurrency
  - Threads/Locks/Semaphores
  - Parallelism
- Persistence
  - File Systems
  - Storage Devices
- Other Selected Topics
  - Debugging/Instrumentation/Final

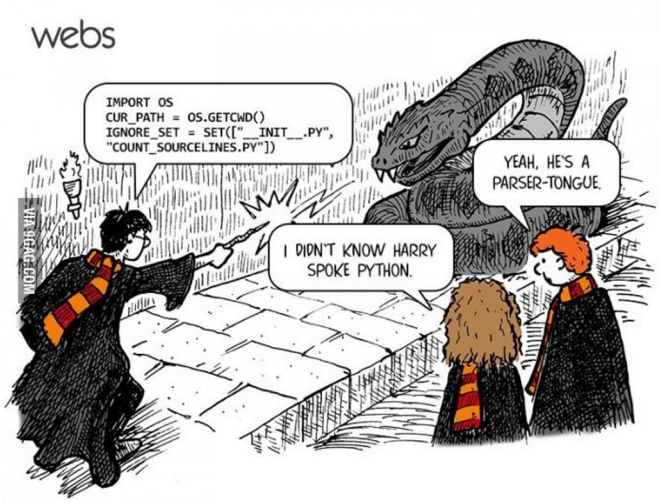


Computer Systems = Magic?

- I hate to break it to you, but there is no magic in computers.
- Computers are just 1's and 0's.
- In this course, we are going to look at 1's and 0's, and how to combine them to create different abstractions.
- That is where the magic comes in however–through the creativity and the art of computer science.
- Computer Science is an art!

“No more magic”

- This is my mantra for all computer systems courses
- We do not have to look at machines any more and think there is magic going on.
- Someone programmed our operating systems, devices, and software
  - And they started off where you are!



Overview

- Lectures
  - Tuesdays 11:45 am – 1:25 pm
  - Thursdays 2:50 pm – 4:30 pm

- Course website: https://course.ccs.neu.edu/cs3650sp23
  - General Info
  - Lecture materials
  - Assignments

- Assignment submission
  - Canvas > Gradescope

- Discussions and questions
  - Canvas > Piazza

Course Goals

- let us review the syllabus
  - https://course.ccs.neu.edu/cs3650sp23/syllabus.html

- All course related information is on the webpage

Course Materials

- A laptop is highly recommended

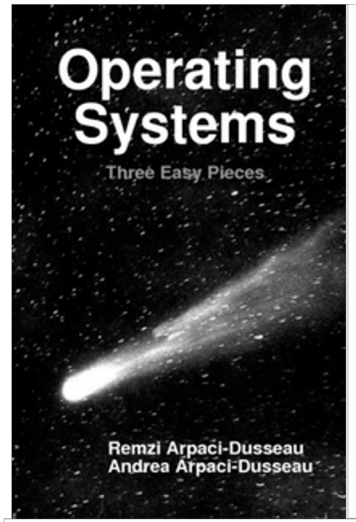
- I do not care what operating system you use on your computer
  - Mac (even with an M1) , Linux (Ubuntu, Debian, etc.), Windows

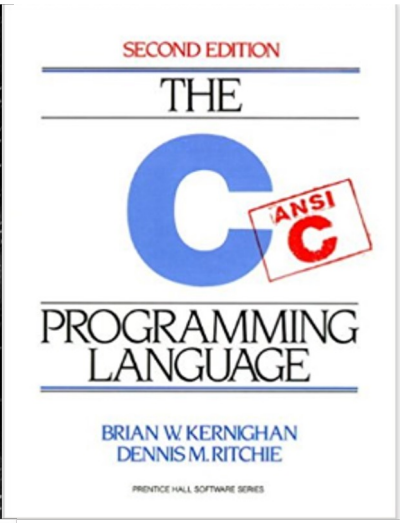
- We will have an online virtualized Linux environment set up for you

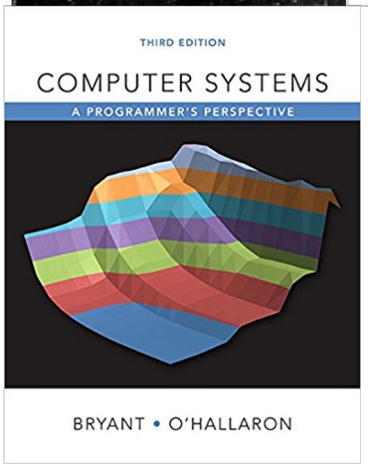


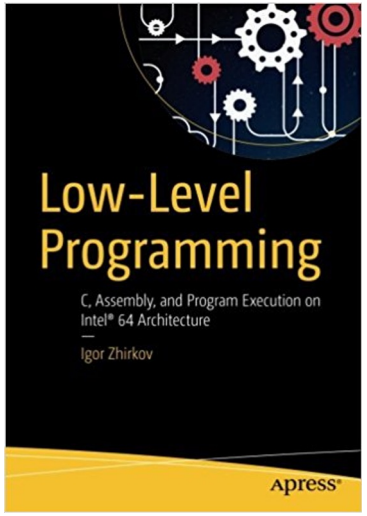
Course Text

- Free main textbooks
  - Dive into Systems
  - Operating Systems: Three Easy Pieces (aka OSTEP)
- Recommended
  - Low-Level Programming: C, Assembly, and Program Execution on Intel® 64 Architecture
  - C Programming Language Book
  - Computer Systems: A Programmer's Perspective
- Inspiration drawn from both of these texts.
- Labs and lectures will have several web resources to check out!









Teaching Style

- Everyone learns differently--optimize as needed along the way
  - There will be lectures (for auditory learners)
  - Many visuals on slides (for visual learners)
  - Labs and assignments (for kinesthetic learners)

- This is a very hands-on class, we will build things

- There will be plenty of opportunity to make mistakes
  Do not be afraid to be wrong
  - The worst-case scenario is we review

- Do ask questions!
  - I try to avoid randomly calling on students--but do participate!

- Come to office hours! Mine or the TAs or both!

Teaching Assistants

- (Will be) Listed on the General tab of the course website
  - We have 17 TAs (for three sections)
  - Welcome them!
  - TA Office Hours: TBD
  - Via Khoury Office Hours

E-mail: don’t use it!

- Post on Piazza general questions to minimize e-mail
  - If not already a member, register through the link on Canvas
- Come to office hours to minimize e-mail

How to ask questions

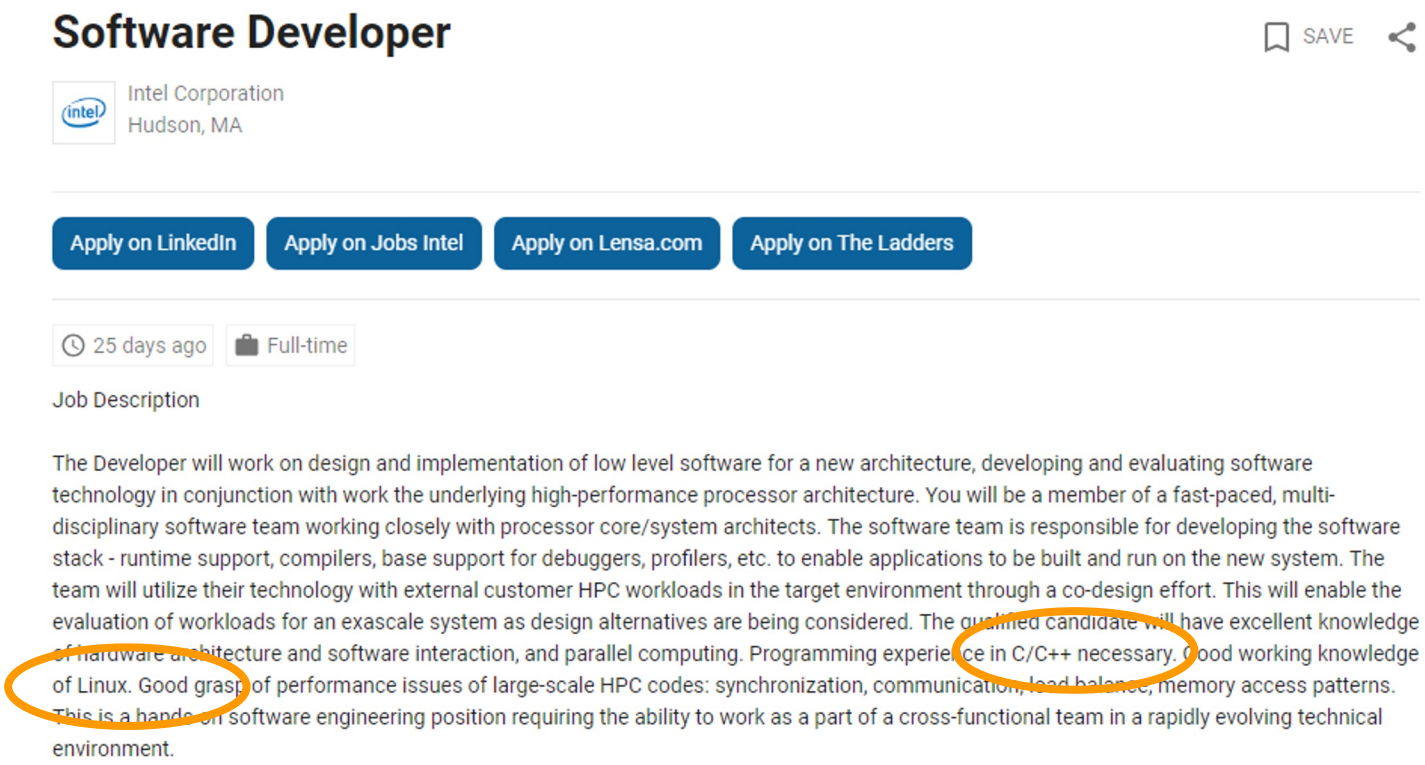
- Ask specific questions
  - My code doesn't work/compile (bad)
  - I tried to do A and A doesn't work in the following ways B (error msg), C (debug info), D (certain behavior), etc. (good)
  - To solve this issue with A, I tried E, F, G but did not work (good)
- But do not reveal solutions

Expectations

- You have taken some ‘programming’ related class.
  - In the instance that you have not--you can still perform well.
    - i.e. Make sure you do the readings
- You know at least one programming language well
- In this course we will use C and get exposed to x86-64 assembly
  - C is (still) the industry standard
  - (You can pick up whatever other fancy systems language later once you learn one)

Yes I know there is GO, Erlang, Rust, etc.

Why C?



Software Developer

SAVE

Intel Corporation
Hudson, MA

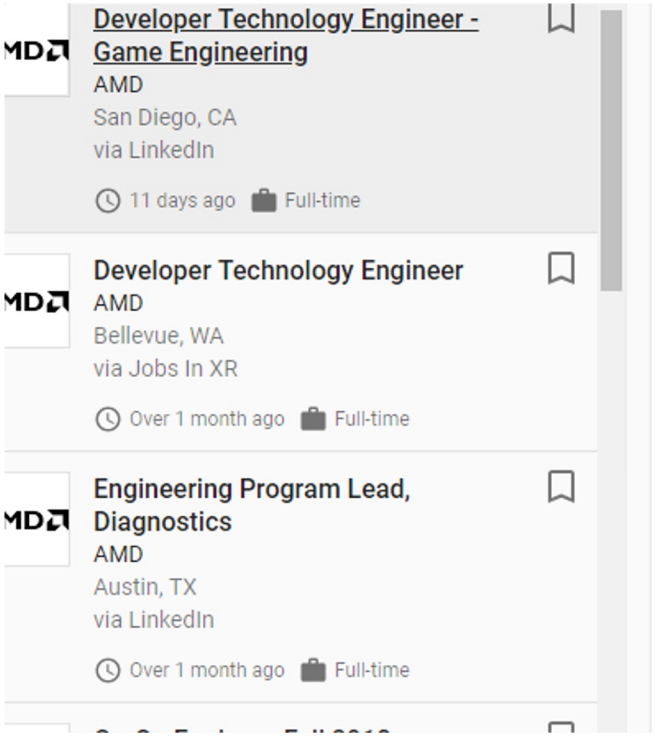
Apply on LinkedIn | Apply on Jobs Intel | Apply on Lensa.com | Apply on The Ladders

25 days ago | Full-time

Job Description

The Developer will work on design and implementation of low level software for a new architecture, developing and evaluating software technology in conjunction with work the underlying high-performance processor architecture. You will be a member of a fast-paced, multi-disciplinary software team working closely with processor core/system architects. The software team is responsible for developing the software stack - runtime support, compilers, base support for debuggers, profilers, etc. to enable applications to be built and run on the new system. The team will utilize their technology with external customer HPC workloads in the target environment through a co-design effort. This will enable the evaluation of workloads for an exascale system as design alternatives are being considered. The qualified candidate will have excellent knowledge of hardware architecture and software interaction, and parallel computing. Programming experience in C/C++ necessary. Good working knowledge of Linux. Good grasp of performance issues of large-scale HPC codes: synchronization, communication, load balance, memory access patterns. This is a hands on software engineering position requiring the ability to work as a part of a cross-functional team in a rapidly evolving technical environment.

Why C?



Developer Technology Engineer - Game Engineering
AMD
San Diego, CA
via LinkedIn
11 days ago · Full-time

Developer Technology Engineer
AMD
Bellevue, WA
via Jobs In XR
Over 1 month ago · Full-time

Engineering Program Lead, Diagnostics
AMD
Austin, TX
via LinkedIn
Over 1 month ago · Full-time

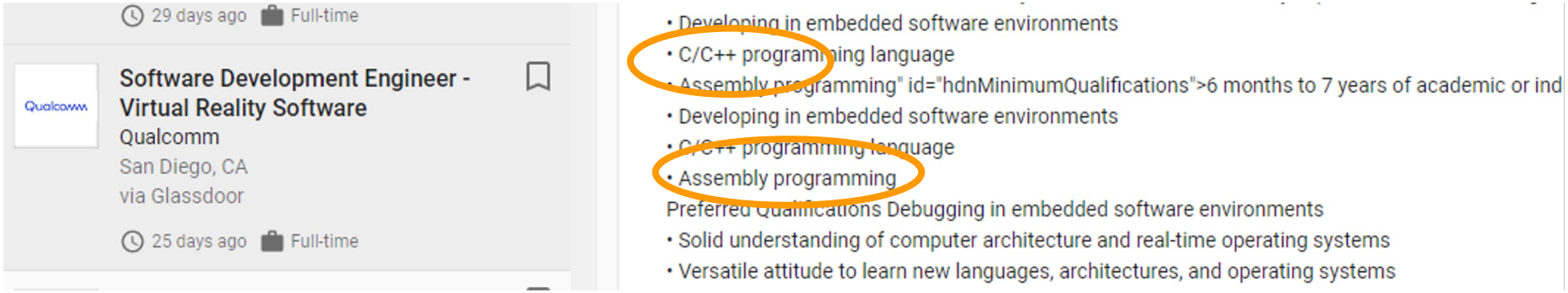
The AMD Game Engineering team works closely with external games software de
planet. We help them to fully exploit the technical capabilities of AMD's hardware
and performance and we do everything in our power to make the user understand

The Successful Candidate Will
- Work with the external game development partners of AMD to enable them to pr
- Optimize game and application performance for discrete GPUs, APUs and CPUs
- Design and implement rendering effects using established APIs
- Integrate features into game titles

The ideal candidate is a highly-skilled software designer and engineer, strong in 3
GPUs, APUs, and CPUs. You're team-driven and motivated to do things others mig

Minimum Requirements
- Has several years of experience efficiently creating C/C++ game code for Windo
language features, standard libraries and writing easy-to-understand code.
- Has practical hands-on experience with DirectX-class development tools and tec
- Has strong graphics code optimization skills, in particular shader code optimiza
- Understands that requirements are rarely perfect and is willing to extract the spi
- Has a degree in computer science or a related technical discipline, or the equiva
- Has excellent written and verbal skills.
- Is willing to travel domestically and internationally on a regular basis.

Why C? (You get the idea)



How to be successful in CS 3650

- Read the assigned reading before class
- Attend the class
  - Ask questions
  - Answer questions
- You need theoretical backgrounds from class to succeed in labs/assignments/projects

How to be successful in CS 3650

- Labs/Assignments/Projects
  - Plan ahead and start early
  - DO NOT START AT THE LAST MOMENT
  - Ask questions early
    - Setting up the environment itself could take a long time
    - Coding always takes longer than your expectation
    - Debugging could take forever

Questions?

So what exactly is C?

Here is what ‘C’ looks like

```c
#include <stdio.h>

int main(){

    puts("Hello Computer Systems!");

    return 0;
}
```

Here is what ‘C’ looks like

- compile with: clang hello.c -o hello

```c
#include <stdio.h>

int main(){

    puts("Hello Computer Systems!");

    return 0;
}
```

Here is what ‘C’ looks like

- compile with: clang hello.c -o hello

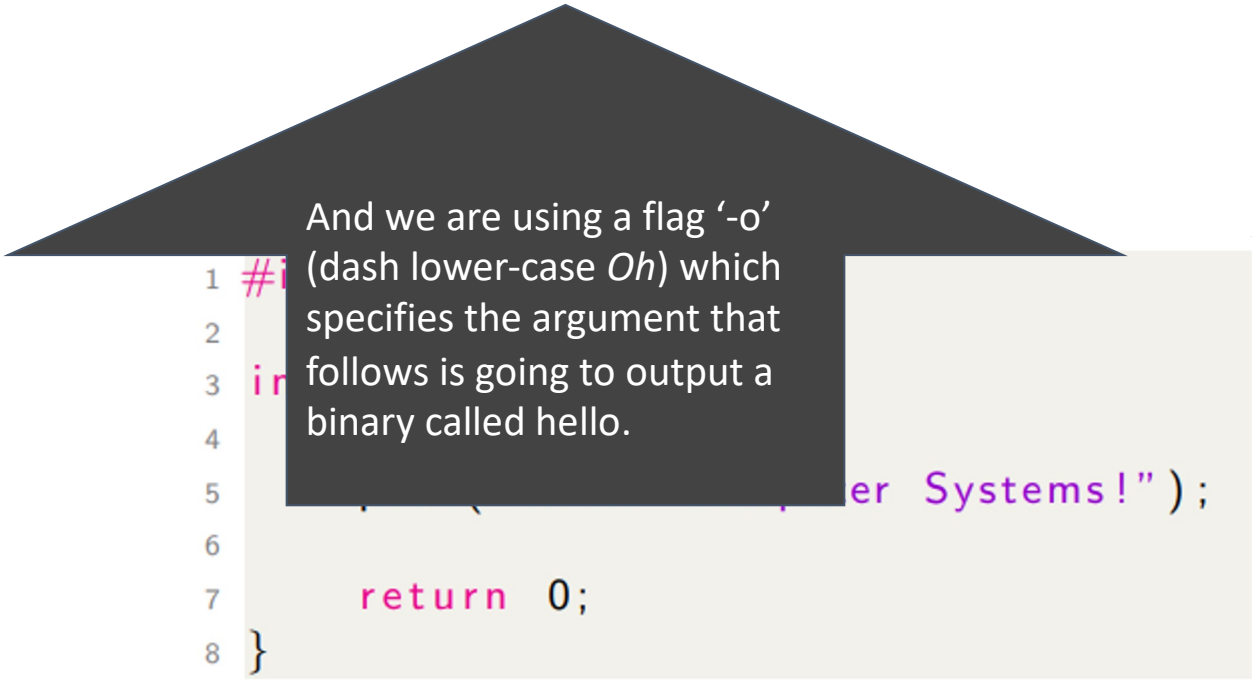
‘clang’ is the compiler

hello.c is the name of our text source code file

```
<stdio.h>

){

    puts("Hello Computer Systems!");

    return 0;
}
```

Here is what ‘C’ looks like

- compile with: clang hello.c -o hello



Here is what ‘C’ looks like

- compile with: clang hello.c -o hello

```c
#include <stdio.h>

int main(){

    puts("Hello Computer Syst   s!");

    return 0;
}
```

#include brings in a library of commands related to standard input and output (so we can print text to the screen)

Here is what ‘C’ looks like

- compile with: clang hello.c -o hello

```c
#include <stdio.h>

int main(){

    puts("Hello Compu

    return 0;
}
```

#puts prints something to the screen. *printf* will be another popular way to do this.

Here is what ‘C’ looks like

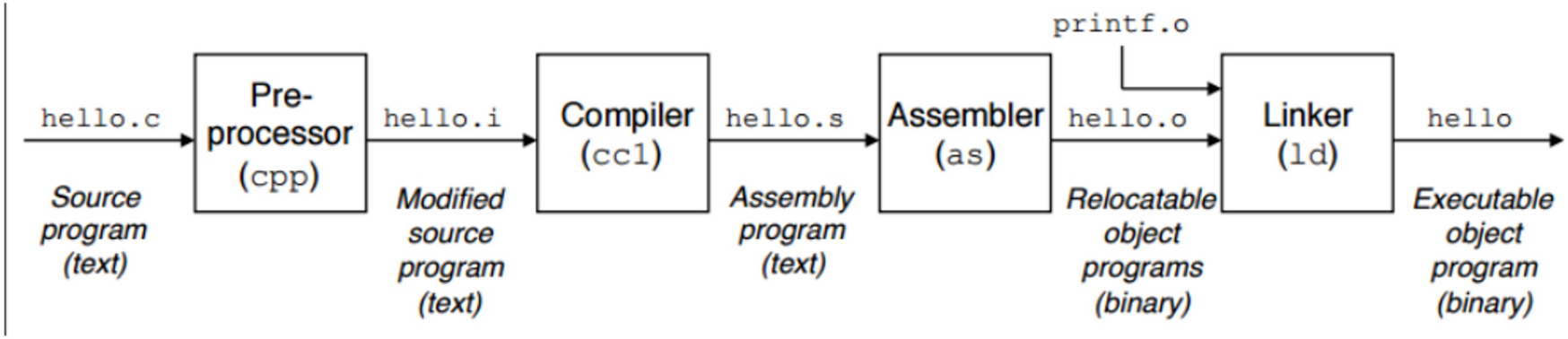
- compile with: clang hello.c -o hello

```
1 #include <stdio.h>
2
3 int main(){
4
5     puts("Hello C
6
7     return 0;
8 }
```

And finally we are done with our program and we return.

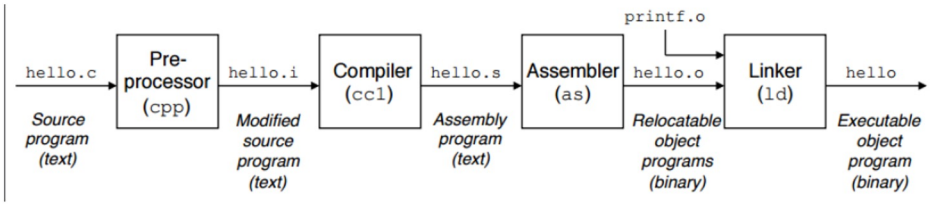
C and the compilation process

- In a picture, this is the compilation process from start to finish
- (Note in this class we'll use clang, but gcc is also fine)



Little exercise to see what compiler is doing

- Generate assembly code
  - clang -S hello.c
- Investigate assembly
- Compile assembly to executable
  - clang hello.s -o hello
- Generate Object file
  - clang -c hello.s
- View Object File
  - nl hello.o (unreadable)
- Investigate Object File
  - objdump -d hello.o
    (disassembly – shows assembly of machine instructions)
  - objdump -t hello.o (shows symbol table)



Quick view of the assembly

- How many folks have not written assembly before?

```c
#include <stdio.h>

int main(){

    puts("Hello Computer Systems!");

    return 0;
}
```

indigo.ccs.neu.edu - PuTTY

```
        .file   "hello.c"
        .text
        .globl  main
        .align  16, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# BB#0:
        pushq   %rbp
.Ltmp2:
        .cfi_def_cfa_offset 16
.Ltmp3:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
.Ltmp4:
        .cfi_def_cfa_register %rbp
        subq    $16, %rsp
        leaq    .L.str, %rdi
        movl    $0, -4(%rbp)
        callq   puts
        movl    $0, %ecx
        movl    %eax, -8(%rbp)          # 4-byte Spill
        movl    %ecx, %eax
        addq    $16, %rsp
        popq    %rbp
        ret
.Ltmp5:
        .size   main, .Ltmp5-main
        .cfi_endproc

        .type   .L.str,@object          # @.str
        .section        .rodata.str1.1,"aMS",@progbits,1
.L.str:
        .asciz  "Hello Computer Systems!"
        .size   .L.str, 24


        .ident  "clang version 3.4.2 (tags/RELEASE_34/dot2-final)"
        .section        ".note.GNU-stack","",@progbits
                                                              39,2-9        All
```

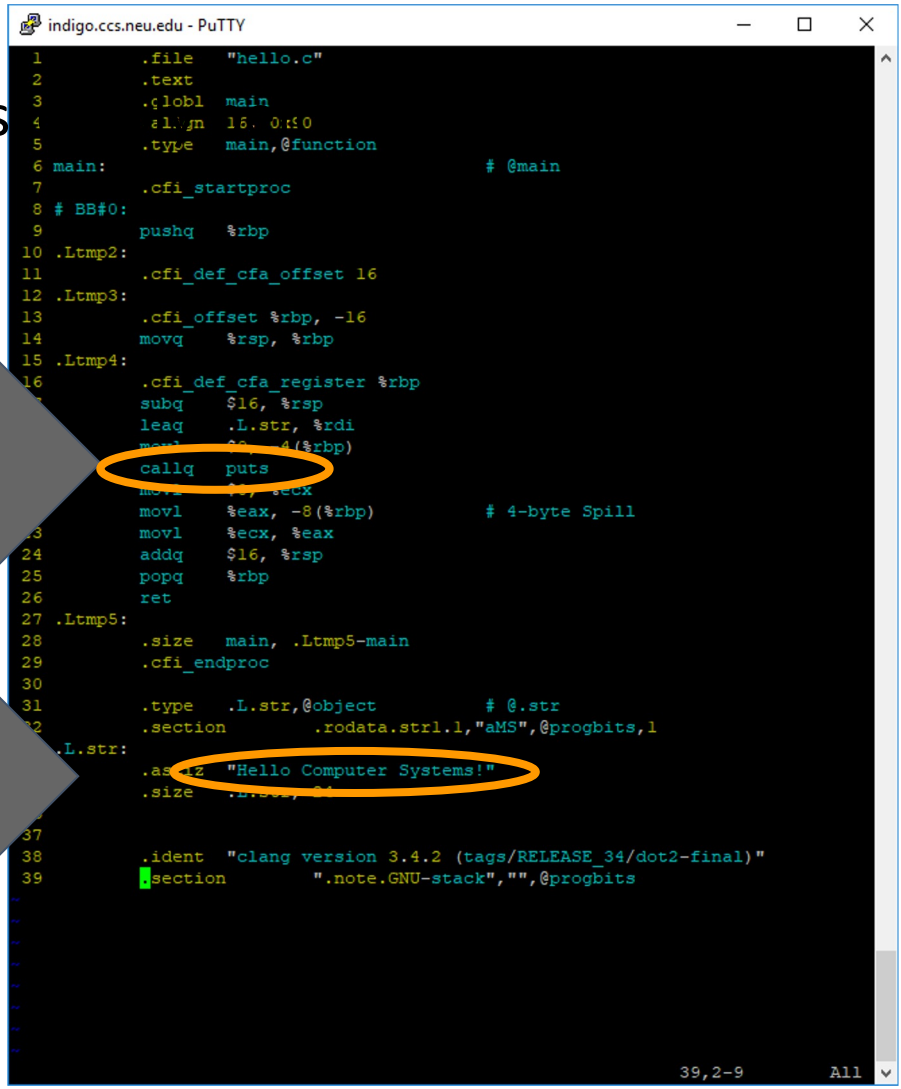
Quick view of the assembly

- How many folks have not written as

```
1 #includ
2
3 int ma
4
5     puts("Hello Computer Systems!");
6
7     return 0;
8 }
```

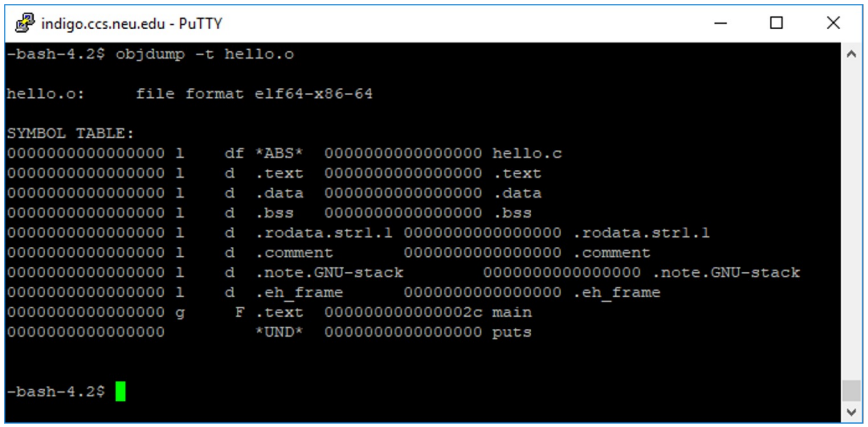
It's not too bad, you can pull out various functions to orient yourself

Our string



Quick view of objdump

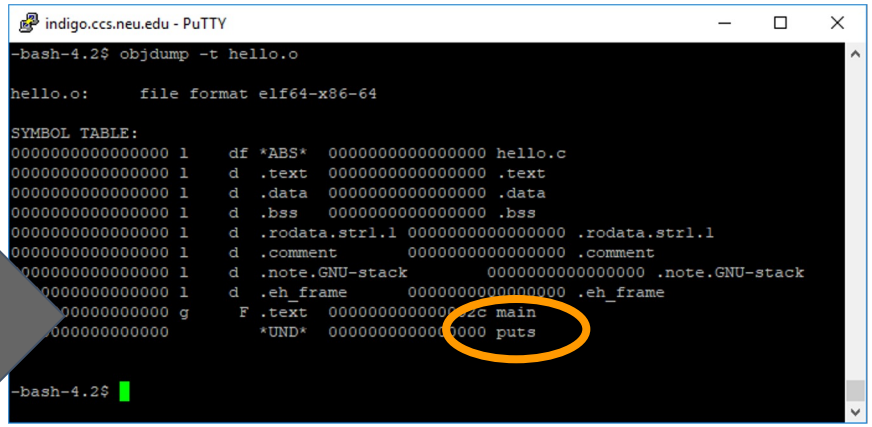
- How many folks have not used objdump before?



Quick view of objdump

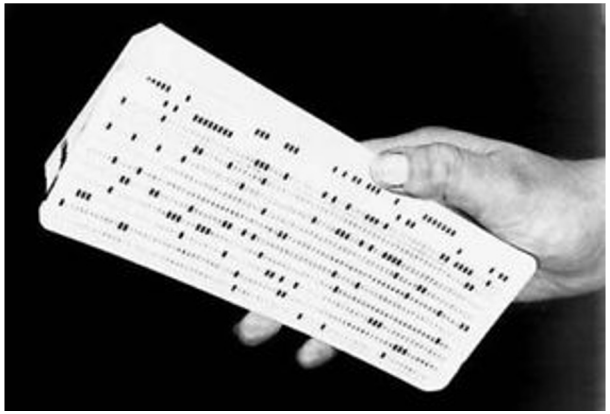
- How many folks have not used objdump before?

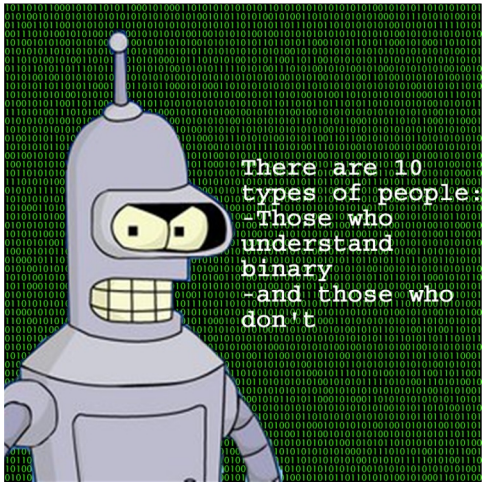
Powerful tool to pull out some information
(Can see functions/libraries used)



So compilers are pretty neat

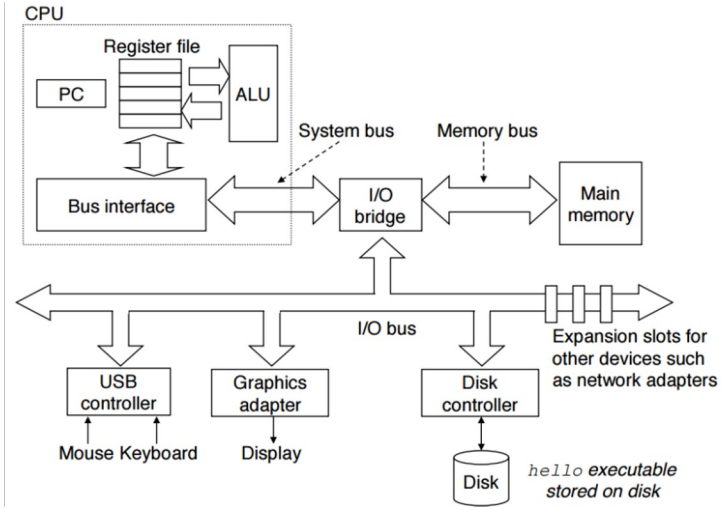
- When we start looking at some of the information taken in, we appreciate the job they do.
  - i.e. transform high level language to binary
- All of a sudden, writing some C code is not so bad!
  - (And it of course is better than pure binary!)





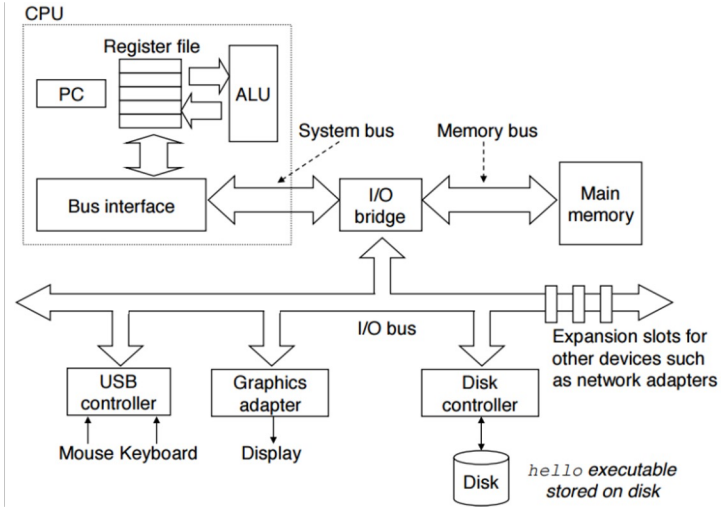
C and compilers allow us to control the system

- Core pieces of systems include hardware(left) and operating system (right)



C and compilers allow us to control the system

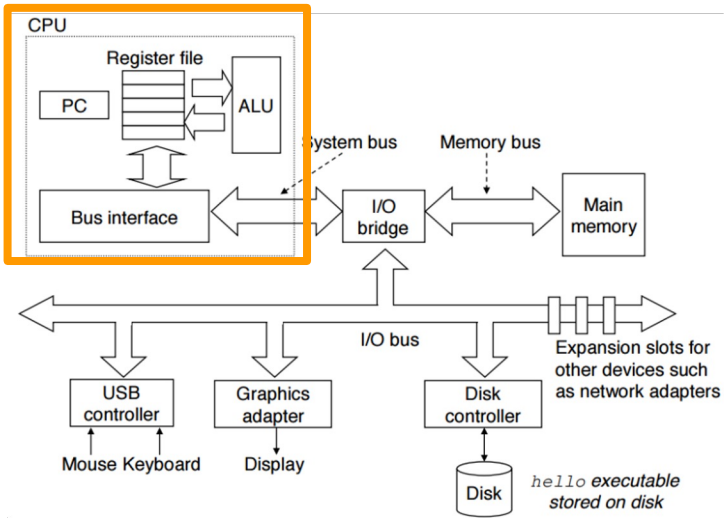
- Core pieces of systems include hardware(left) and operating system (right)



Let's take a few minutes to think about the hardware

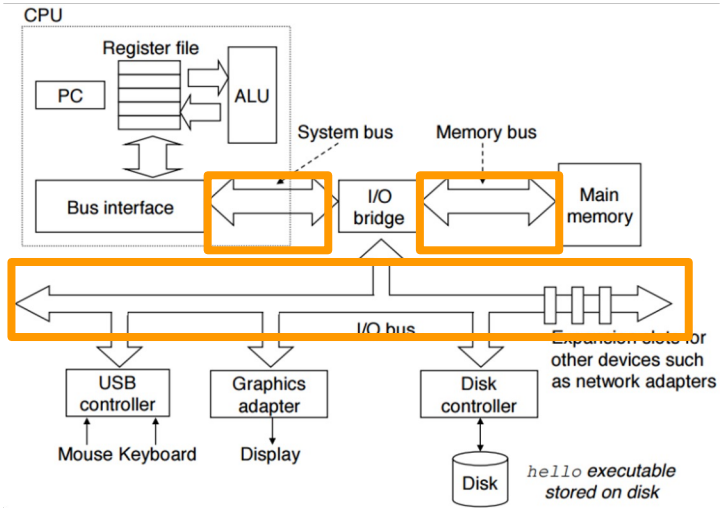
Modern Hardware Visual Abstraction

- CPU is the “brain” of modern hardware
  - That’s where 1 instruction is executed at a time
  - Only 1!
  - (Note: Modern computers have multiple cores)
- We generally measure the speed at which a CPU executes in Megahertz or Gigahertz
  - This is a metric for how ‘fast’ a CPU performs, and how complex of software can be run.

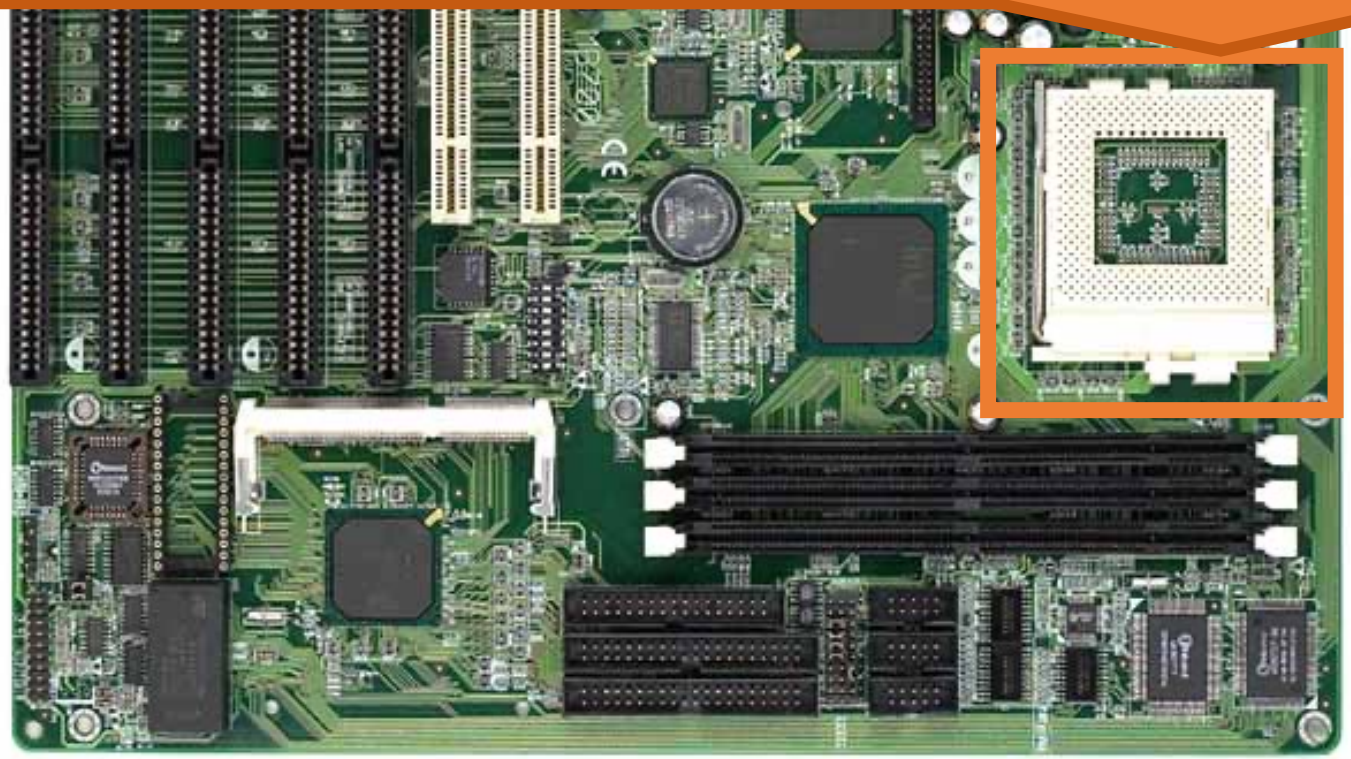


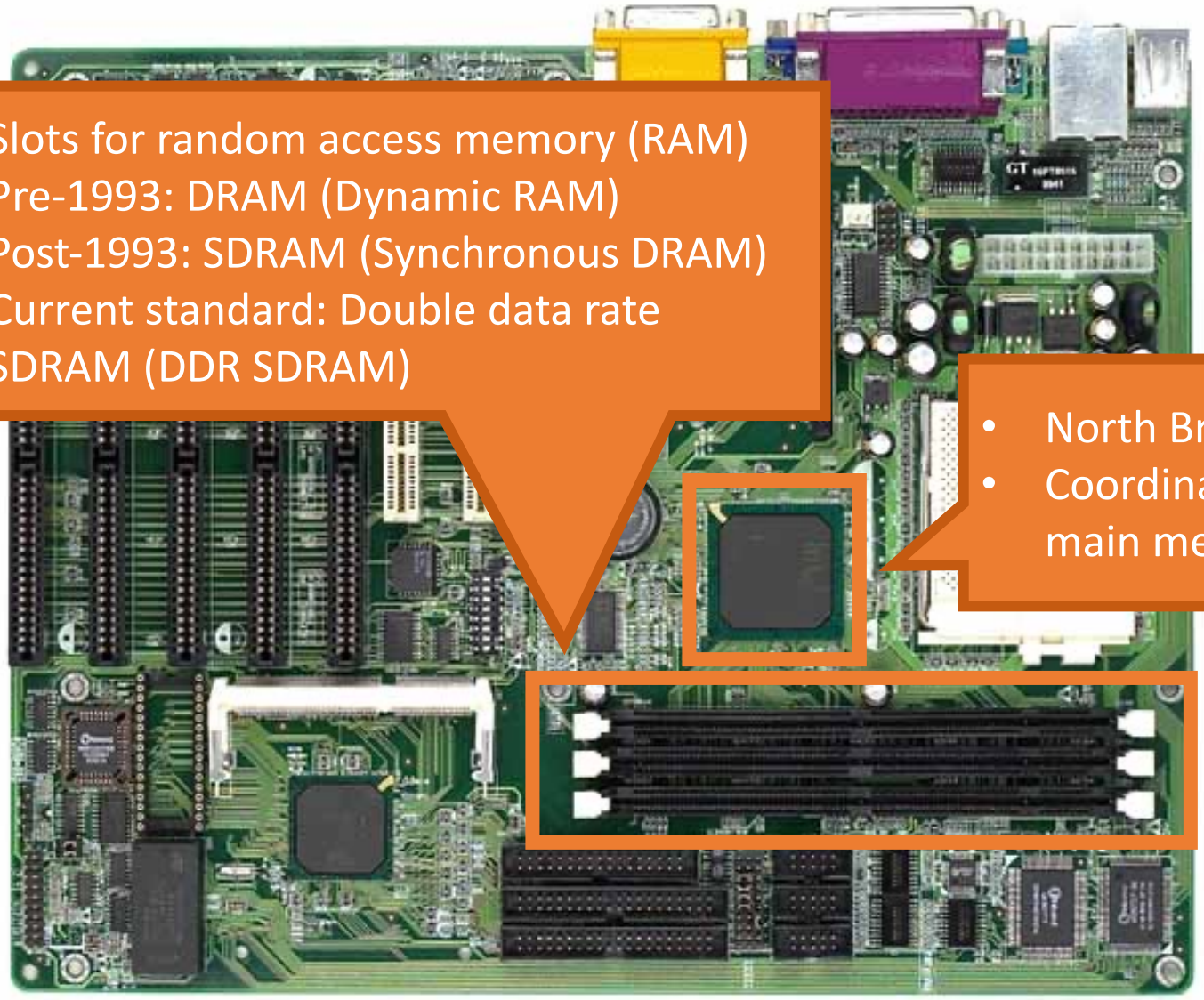
Modern Hardware Visual Abstraction

- Beyond the CPU, a number of devices may also be connected.
- Buses transfer information from devices and memory into the CPU.
- There is a lot going on, and this needs to be managed
- Note: Busses can be thought of as simple networks, with many things hardcoded



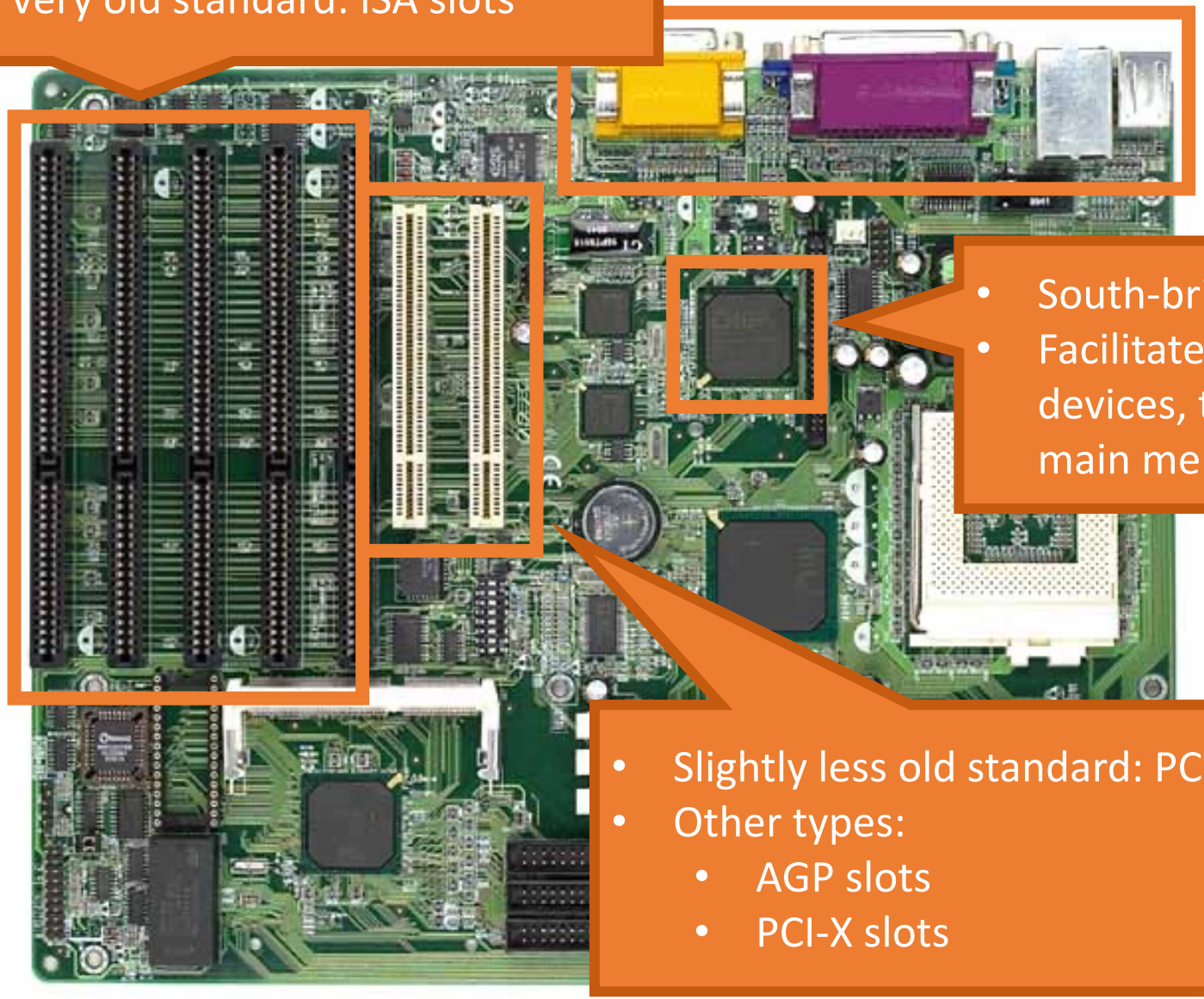
- CPU Socket
- Many different physical socket standards
  - This a Pentium 1 socket
- Physical standard is less important than Instruction Set Architecture (ISA)
  - IBM PCs are Intel 80386 compatible
  - Original x86 design
  - Intel, AMD, VIA
- Today's dominant ISA: x86-64, developed by AMD




• Slots for random access memory (RAM)
• Pre-1993: DRAM (Dynamic RAM)
• Post-1993: SDRAM (Synchronous DRAM)
• Current standard: Double data rate SDRAM (DDR SDRAM)
• North Bridge
• Coordinates access to main memory

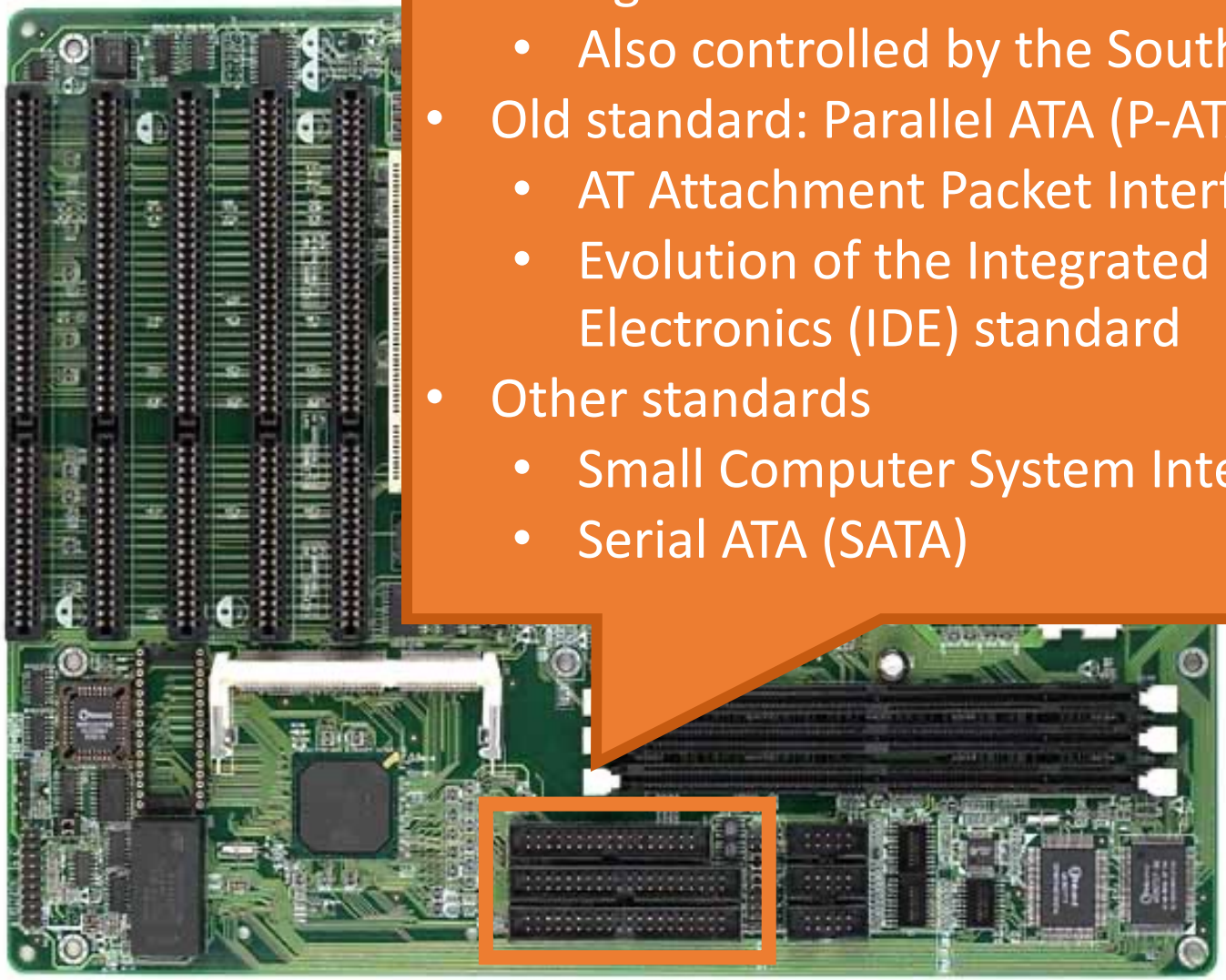
- I/O device slots
- Attached to the south-bridge bus
- Very old standard: ISA slots

- Built in I/O also on the PCI/ISA bus

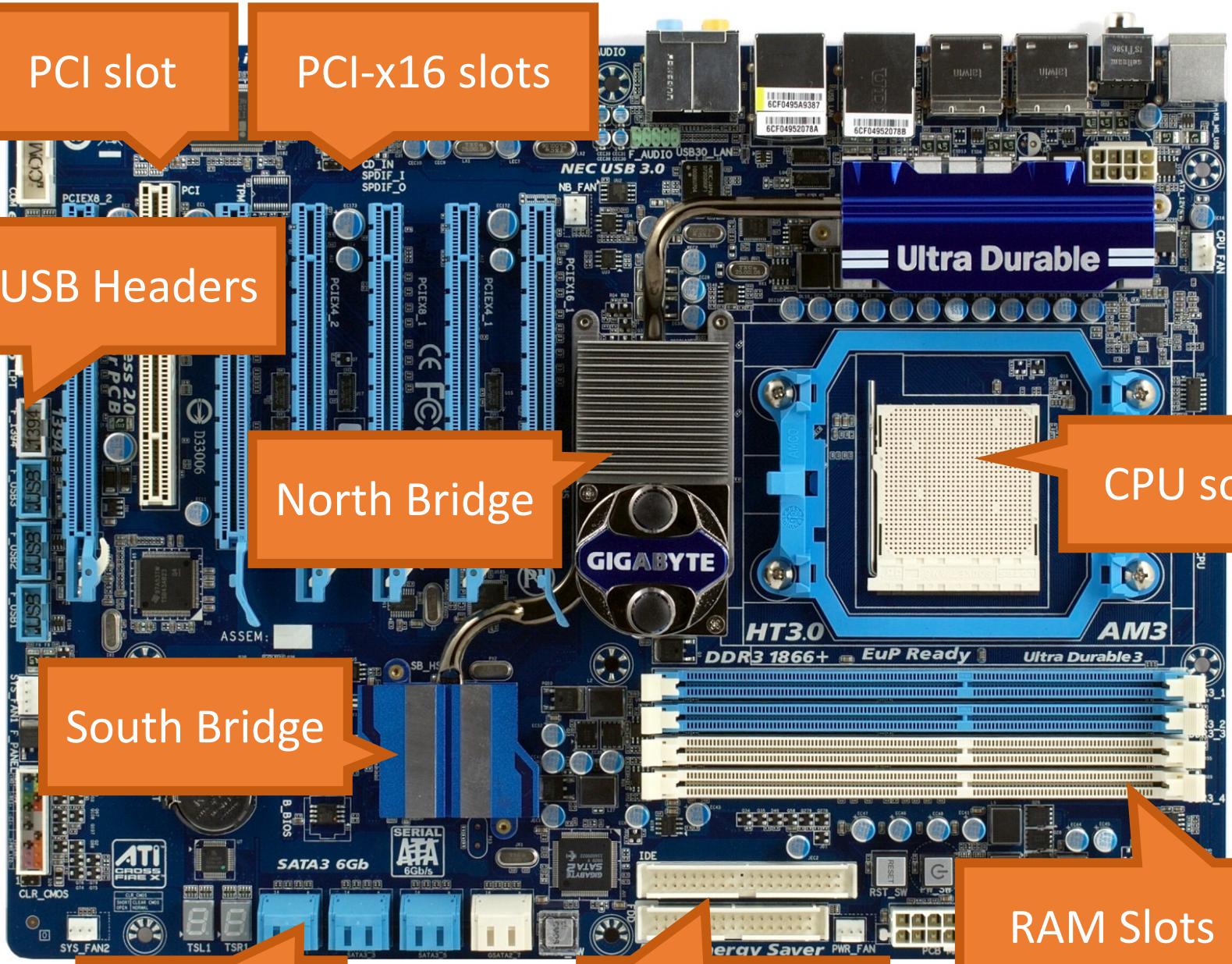


- South-bridge
- Facilitates I/O between devices, the CPU, and main memory

- Slightly less old standard: PCI slots
- Other types:
  - AGP slots
  - PCI-X slots



- Storage connectors
  - Also controlled by the South Bridge
- Old standard: Parallel ATA (P-ATA)
  - AT Attachment Packet Interface (ATAPI)
  - Evolution of the Integrated Drive Electronics (IDE) standard
- Other standards
  - Small Computer System Interface (SCSI)
  - Serial ATA (SATA)


PCI slot
PCI-x16 slots
USB Headers
North Bridge
CPU socket
South Bridge
RAM Slots
SATA Plugs
PATA Connectors
Ultra Durable
GIGABYTE
HT3.0
AM3
DDR3 1866+
EuP Ready
Ultra Durable 3
NEC USB 3.0
SATA3 6Gb

C and compilers allow us to control the system

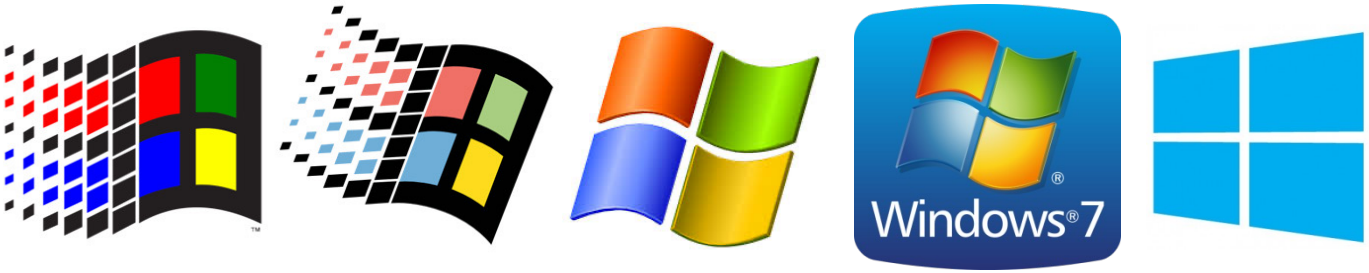
- Core pieces of systems include hardware(left) and operating system (right)

Let's take a moment to think about operating systems

What is an Operating System?

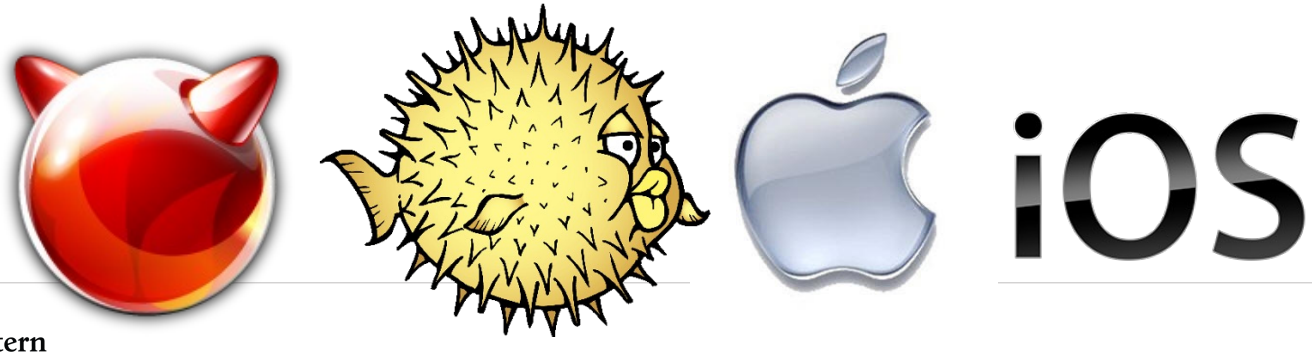
Many Different OSes

Windows



Linux

BSD



Many Different OSes

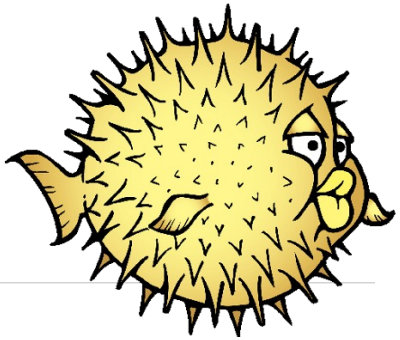
Windows

Linux

Operating Systems are actively developed!
(read: co-ops/jobs)

You can actively contribute to the open source ones now!

BSD

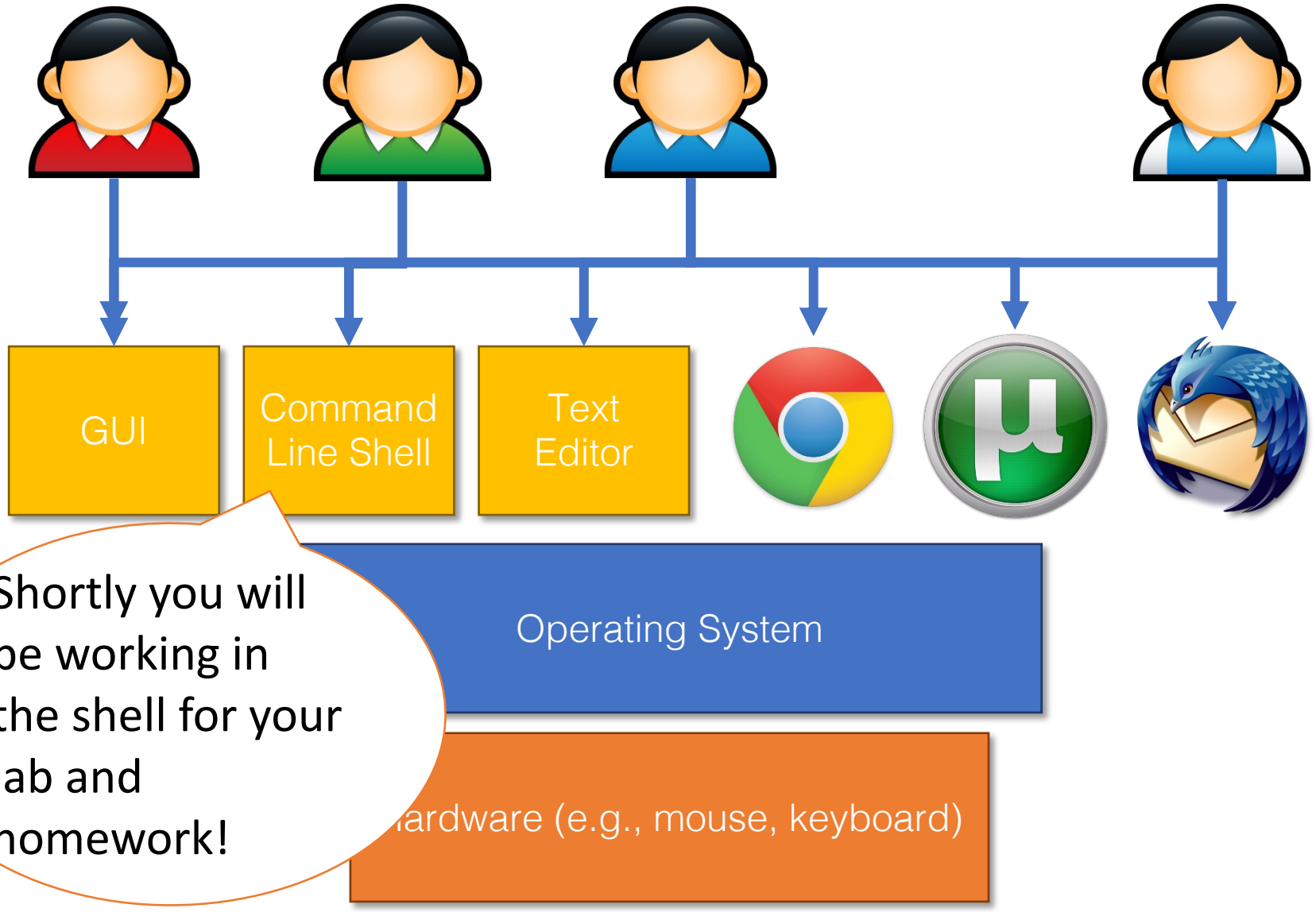


iOS

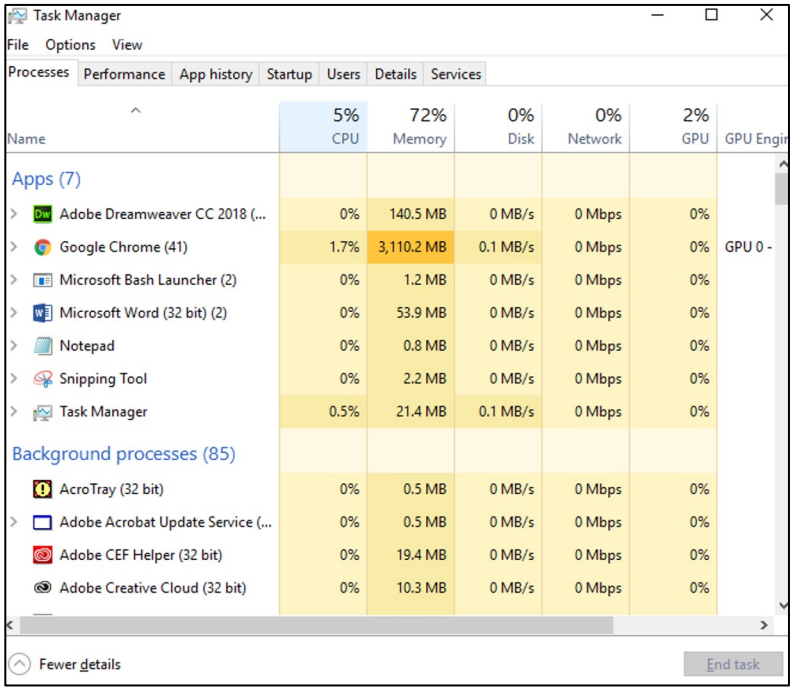
What is an Operating System?

- OS is software that sits between user programs and hardware

- OS provides interfaces to computer hardware
  - User programs do not have to worry about details



Shortly you will be working in the shell for your lab and homework!



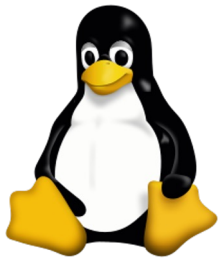
- OS is a resource manager and control program
  - Controls execution of user programs
  - Decides between conflicting requests for hardware access
  - Attempts to be efficient and fair
  - Prevents errors and improper use

Two Common OS Families

- POSIX
  - Anything Unix-ish
  - e.g. Linux, BSDs, Mac, Android, iOS, QNX
- Windows
  - Stuff shipped by Microsoft
- Many other operating systems may exist specific to a domain (e.g. an operating system for a car, handheld gaming device, or smart refrigerator)

In this course, we will work in a POSIX Environment. Our Khoury machines are Unix based.

Who, what, why, ... Linux? https://www.linuxfoundation.org/



- Linux is a family of free open source operating systems
  - That means the code is freely available, and you can contribute to the project!
- It was created by Linus Torvalds
  - Variants of Linux are: Ubuntu, Debian, Fedora, Gentoo Linux, Arch Linux, CentOS, etc.
  - They all operate under roughly the same core code, which is called the kernel.
  - Often they differ by the software, user interface, and configuration settings.
  - So very often linux software for one flavor of linux will run on the other with few or no changes.
- Generally we (as systems programmers) like Linux, because it is a clean and hackable operating system.
- When many folks think of Unix-like operating systems, they may think of a hacker using a 'command-line interface' to program.

Over 30 years ago...

On Monday, August 26, 1991 at 2:12:08 AM UTC-4, Linus Benedict Torvalds wrote:
> Hello everybody out there using minix -
>
> I'm doing a (free) operating system (**just a hobby, won't be big and**
> **professional like gnu**) for 386(486) AT clones. This has been brewing
> since april, and is starting to get ready. I'd like any feedback on
> things people like/dislike in minix, as my OS resembles it somewhat
> (same physical layout of the file-system (due to practical reasons)
> among other things).
>
> I've currently ported bash(1.08) and gcc(1.40), and **things seem to work**.
> This implies that I'll get something practical within a few months, and
> I'd like to know what features most people would want. **Any suggestions**
> **are welcome, but I won't promise I'll implement them :-)**
>
> Linus (torv...@kruuna.helsinki.fi)
>
> PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
> **It is NOT protable (uses 386 task switching etc)**, and **it probably never**
> **will support anything other than AT-harddisks, as that's all I have :-(**.

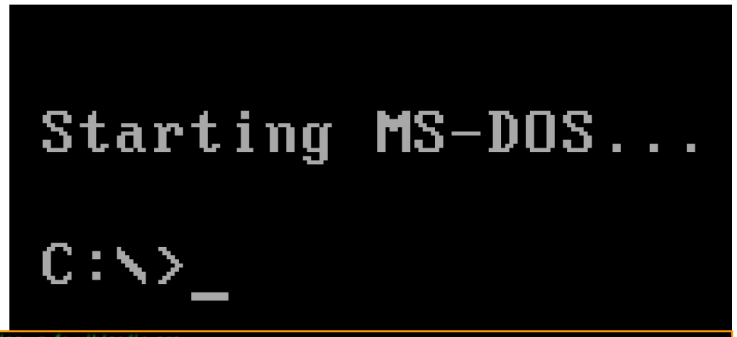
Over 30 years ago...

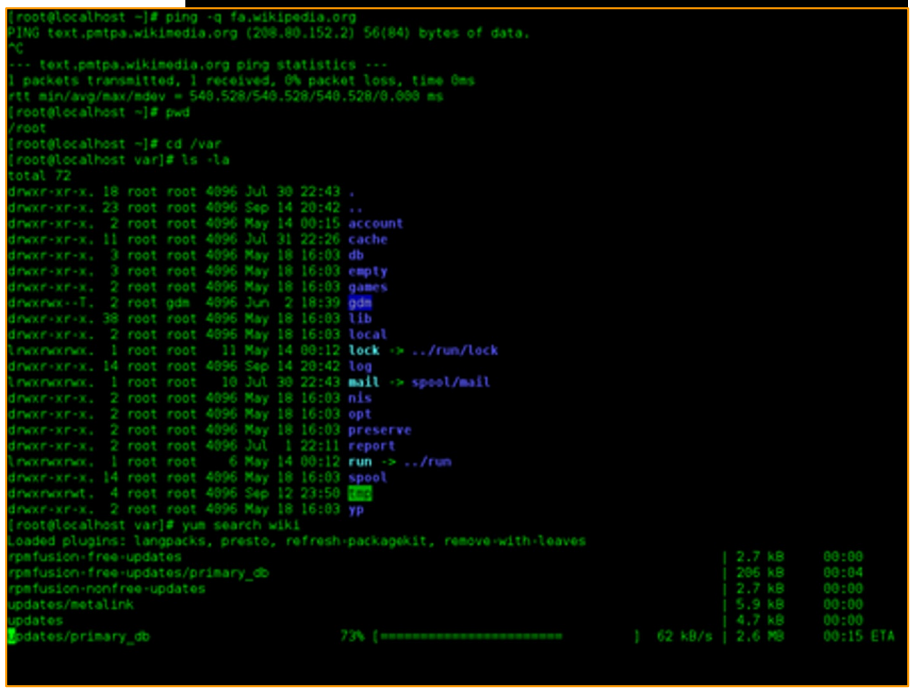
On Monday, August 26, 1991 at 2:12:08 AM UTC-4, Linus Benedict Torvalds wrote:
> Hello everybody out there using minix -
>
> I'm doing a (free) op
> **professional like gnu**
> since april, and is s
> things people like/di
> (same physical layout
> among other things).
>
> I've currently ported
> This implies that I'll get s actical within a few months, and
> I'd like to know what feat people would want. **Any suggestions**
> **are welcome, but I won't e I'll implement them :-)**
>
> Linus rv...@kruuna.helsinki.fi)
>
> PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
> **It is NOT protable (uses 386 task switching etc)**, and **it probably never**
> **will support anything other than AT-harddisks, as that's all I have :-(**.

Linux platforms: Alpha, ARC, ARM, ARM64, Apple M1 C6x, H8/300, Hexagon, Itanium, m68k, Microblaze, MIPS, NDS32, Nios II, OpenRISC, PA-RISC, PowerPC, RISC-V, s390, SuperH, SPARC, Unicore32, x86, x86-64, XBurst, Xtensa

The command line interface

- The command line interface is at the highest level just another program.
- Linux and Mac have terminals built-in, and Windows as well (cmd and powershell).
- From it, we can type in the names of programs to perform work for us





Shell demo

- ls
- cd (cd ~, /, ..)
- pwd
- tree
- tab
- up/down arrow
- history

Why the command line?

- You might argue “I love GUI interfaces, so simple and sleek looking”
- The command line is a lot faster than moving your mouse
- It is also very convenient for ‘scripting’ behavior that you could not so easily do in a GUI environment.
  - Executing a few commands in a row in a script is a piece of cake!
- And if you are working remotely, you often will not have any GUI environment at all!
  - (Often machines you need to access do not have a monitor attached)

Example shell script

mikeshah@DESKTOP-DDNGQVA: /mnt/c/Windows/System32

```
 1 # Lines that start with a 'hashmark' or 'pound sign'
 2 # are comments that are ignored.
 3 # You should use them liberally!
 4
 5 # This line is special and tells us we have an executable script.
 6 #!/bin/bash
 7
 8 # Output hello and two items read in as command-line arguments
 9 echo "Hello $1 $2"
10 echo "What is your age?"
11 # Read in a value
12 read myAge
13 echo "That is great you are $myAge years old!"
```

Example shell script

- I wrote this script in a text editor called ‘vim’
- You will have to learn VIM (or emacs) in this course.
  - It’s a great skill to have.

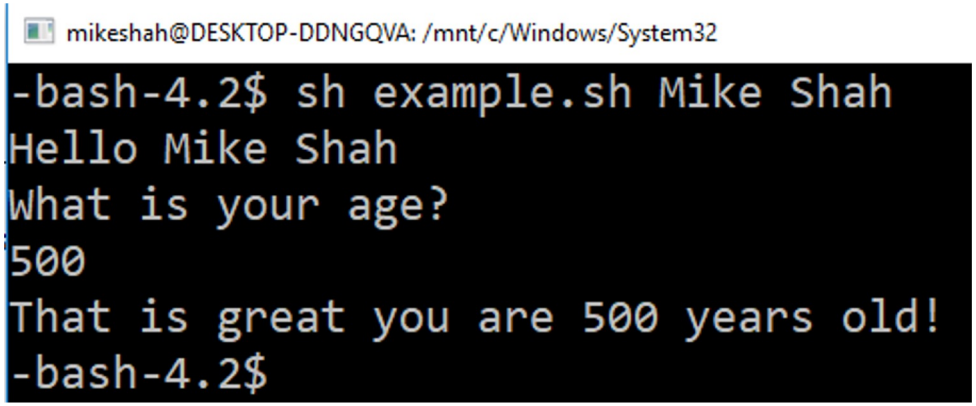
mikeshah@DESKTOP-DDNGQVA: /mnt/c/Windows/System32

```
 1 # Lines that start with a 'hashmark' or 'pound sign'
 2 # are comments that are ignored.
 3 # You should use them liberally!
 4
 5 # This line is special and tells us we have an executable script.
 6 #!/bin/bash
 7
 8 # Output hello and two items read in as command-line arguments
 9 echo "Hello $1 $2"
10 echo "What is your age?"
11 # Read in a value
12 read myAge
13 echo "That is great you are $myAge years old!"
```

Example shell script Executing

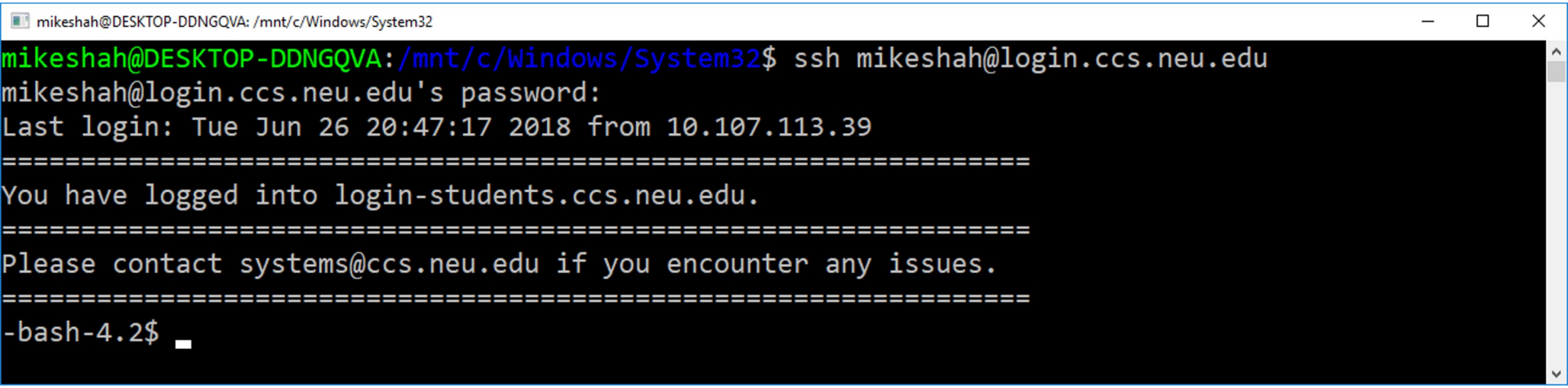
- (Am I really 500 years old? Time flies when you are having fun!)

  - Note “Mike Shah” are the first and second arguments passed into this program



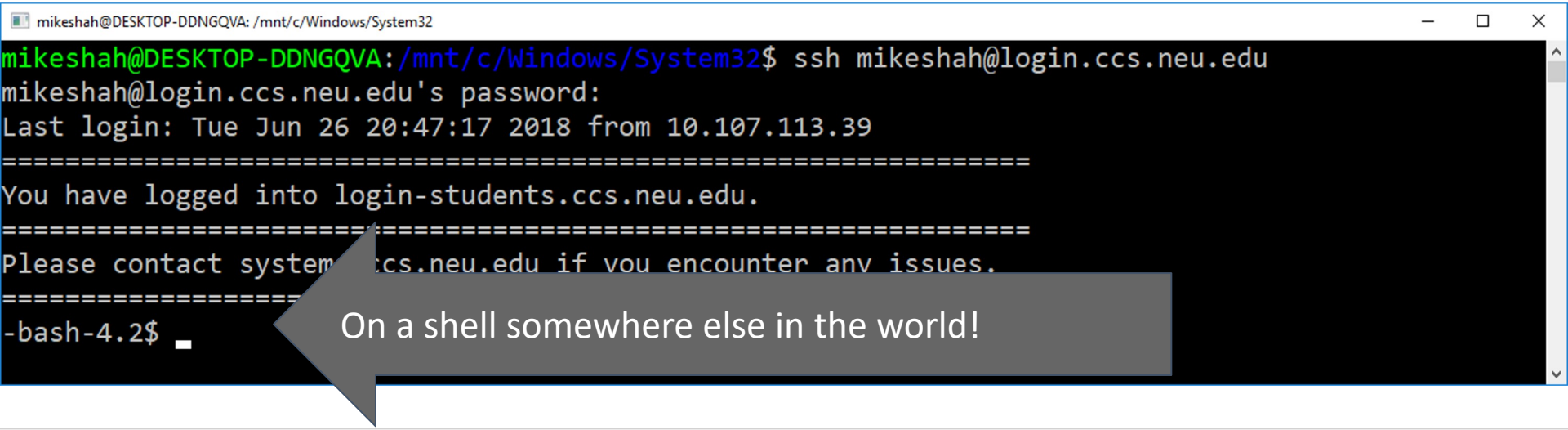
ssh - secure shell

- Our tool for remote access--which we will do for all of our work!
- ssh some_user_name@login.ccs.neu.edu
- After typing in my password successfully, I am now executing commands on a machine somewhere on Northeastern's campus



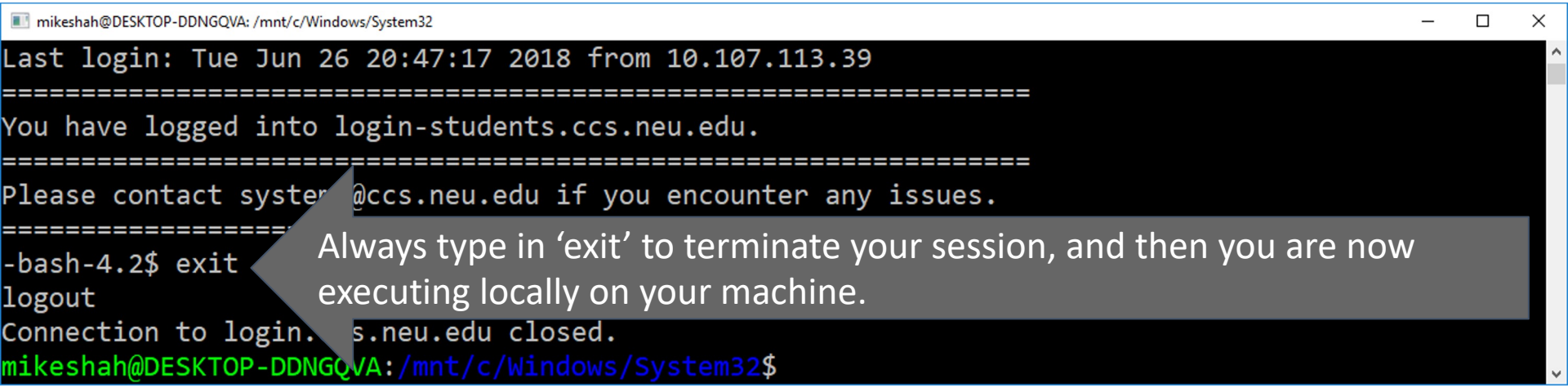
ssh - secure shell

- Our tool for remote access--which we will do for all of our work!
- ssh some_user_name@login.ccs.neu.edu
- After typing in my password successfully, I am now executing commands on a machine somewhere on Northeastern's campus



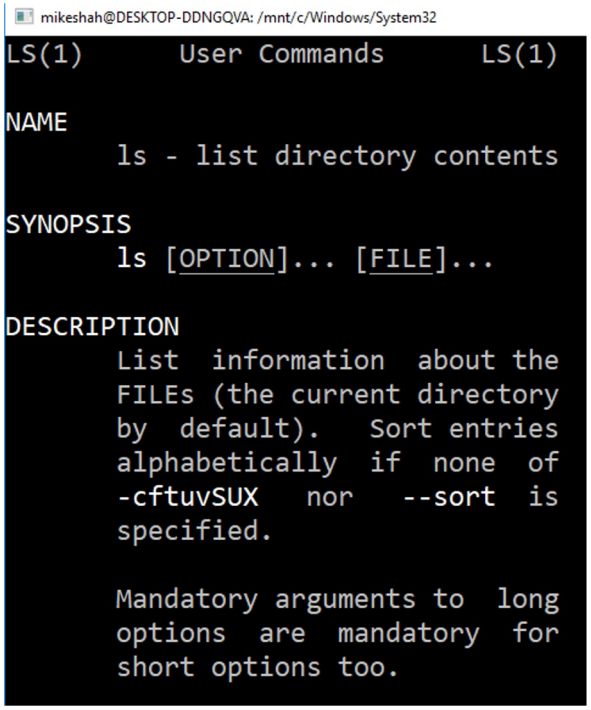
ssh - secure shell

- Our tool for remote access--which we will do for all of our work!
- ssh some_user_name@login.ccs.neu.edu
- After typing in my password successfully, I am now executing commands on a machine somewhere on Northeastern's campus



Feeling overwhelmed or forgetting a command?

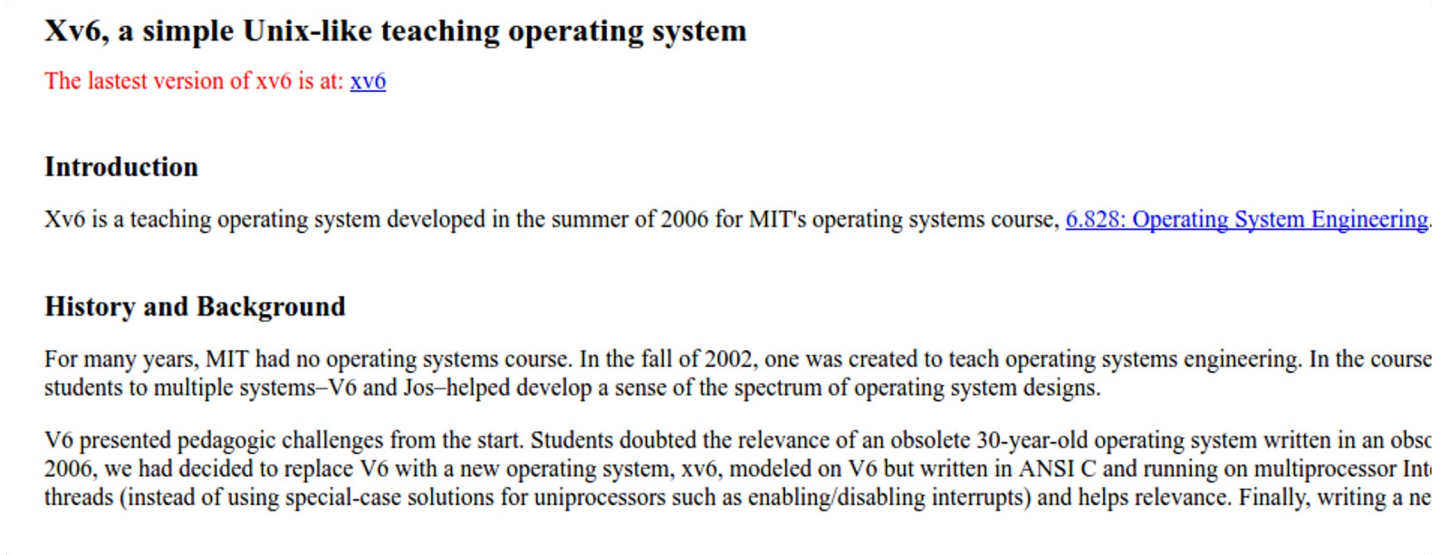
- Luckily there are built-in ‘manual pages’
- Called the ‘man pages’ for short.
- Simply type ‘man command_name’ for help
  - (Hit ‘q’ to quit the page when you are done)



What is xv6?

A teaching operating system! (i.e. small version of Unix)

- https://pdos.csail.mit.edu/6.828/2012/xv6.html



Xv6, a simple Unix-like teaching operating system

The lastest version of xv6 is at: xv6

Introduction

Xv6 is a teaching operating system developed in the summer of 2006 for MIT's operating systems course, 6.828: Operating System Engineering.

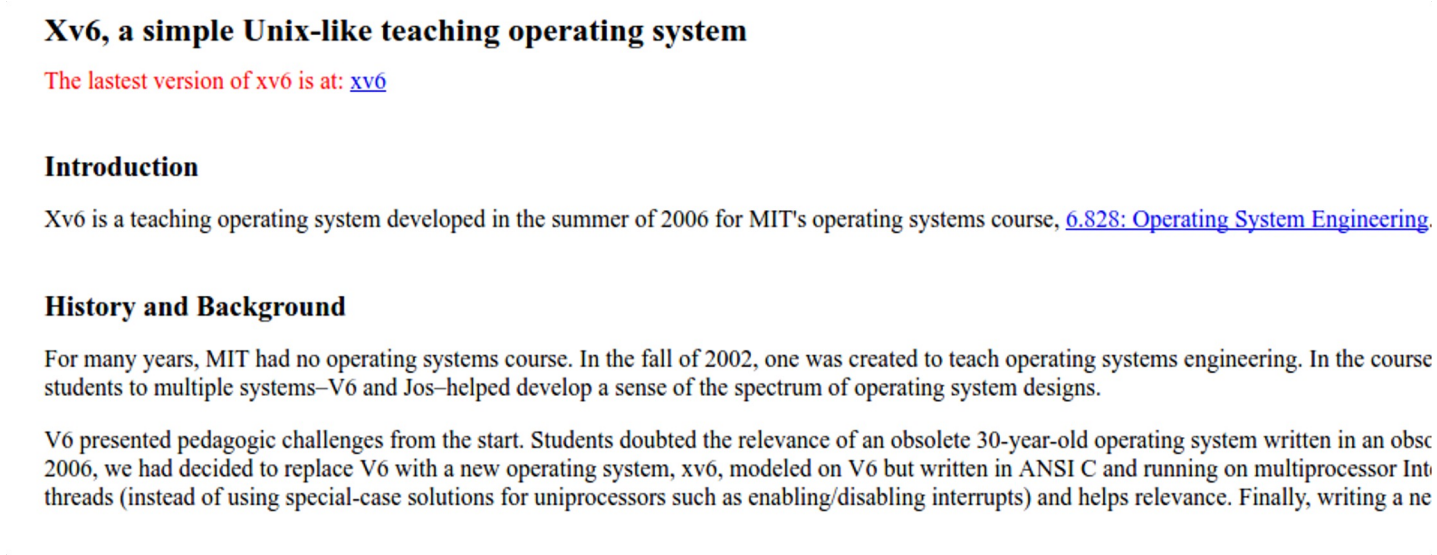
History and Background

For many years, MIT had no operating systems course. In the fall of 2002, one was created to teach operating systems engineering. In the course students to multiple systems–V6 and Jos–helped develop a sense of the spectrum of operating system designs.

V6 presented pedagogic challenges from the start. Students doubted the relevance of an obsolete 30-year-old operating system written in an obsc 2006, we had decided to replace V6 with a new operating system, xv6, modeled on V6 but written in ANSI C and running on multiprocessor Int threads (instead of using special-case solutions for uniprocessors such as enabling/disabling interrupts) and helps relevance. Finally, writing a ne

A ~~teaching~~ small & manageable operating system!

- https://pdos.csail.mit.edu/6.828/2012/xv6.html



Xv6, a simple Unix-like teaching operating system

The lastest version of xv6 is at: xv6

Introduction

Xv6 is a teaching operating system developed in the summer of 2006 for MIT's operating systems course, 6.828: Operating System Engineering.

History and Background

For many years, MIT had no operating systems course. In the fall of 2002, one was created to teach operating systems engineering. In the course students to multiple systems–V6 and Jos–helped develop a sense of the spectrum of operating system designs.

V6 presented pedagogic challenges from the start. Students doubted the relevance of an obsolete 30-year-old operating system written in an obsc 2006, we had decided to replace V6 with a new operating system, xv6, modeled on V6 but written in ANSI C and running on multiprocessor Int threads (instead of using special-case solutions for uniprocessors such as enabling/disabling interrupts) and helps relevance. Finally, writing a ne

xv6

- We will be using xv6 to build and implement some Operating Systems features
- This will give you experience adding features to a large piece of software.

Summary

- We are going to learn about computer systems
  - This includes software (e.g. compilers), hardware, and some operating system concepts.

- We are going to work in a Unix environment
  - This work will be performed on a command-line
  - In this course we can access a command-line either:
    - Through SSH or a Virtual Machine

- One final thing
  - Even with the best planning…
  - Some things may change this semester that are beyond our control
  - Everyone (including us) needs to be flexible
  - If you have an issue, it is better to tell us early than at the last minute

- I'm looking forward to being your guide to Computer Systems