Introduction to Computer Systems

Chris Riesbeck, Fall 2011

Welcome to Intro. to Computer Systems

- Everything you need to know

  http://www.cs.northwestern.edu/academics/courses/213/
- Instructor: Chris Riesbeck
  - TA: Jaime Espinosa
- Communication channels:
  - Course webpage
  - News: cs.news – cs.213
- Prerequisites
  - EECS 211 or equivalent
  - Experience with C or C++
  - Useful: EECS 311

Course theme

When things break, look under the hood!

- Most CS courses emphasize abstraction
  - Abstract data types, procedural abstraction, asymptotic algorithmic analysis, OOAD, …
  - Abstraction is critical to managing complexity
- But the reality beneath is important too
  - Debugging how adding 2 positives resulted in a negative
  - Efficient implementations of new abstractions
- Useful outcomes
  - Become more effective programmers
    - Better at debugging, performance tuning, …
  - Prepare for later “systems” classes in CS & ECE
    - Compilers, Operating Systems, Networks, ...

Course perspective

- Most systems courses are application-centered
  - Operating Systems: Implement portions of an OS
  - Compilers: Write compiler for simple language
  - Networking: Implement and simulate network protocols
- This course is skills-centered
  - By knowing more about the underlying system, you will be a more effective programmer, able to
    - write programs that are more reliable and efficient
    - incorporate features that require hooks into the OS
  - We'll bring out the hidden hacker in everyone.

Topics Covered

- Programs and data
  - Bit arithmetic, assembly, program control …
  - Aspects of architecture and compilers
- Memory
  - Memory technology, memory hierarchy, caches, disks, locality
  - Aspects of architecture and OS.
- Linking & exceptional control flow
  - Object files, dynamic linking, libraries, process control, …
  - Aspects of compilers, OS, and architecture
- Virtual memory
  - Virtual memory, address translation, dynamic storage allocation
  - Aspects of architecture and OS
- I/O, networking & concurrency
  - High level & low-level I/O, net programming, threads, …
  - Aspects of networking, OS, and architecture

Course components

- Lectures
  - Higher level concepts
  - 10% of grade from class participation
- 4 Labs
  - The heart of the course – in-depth understanding
  - 2 ~ 3 weeks
  - 10% of grade each
  - Most can be done with partner
- 4 Homework assignments
  - 10% of grade
- Exams – midterm & final
  - 20% of grade each

Lab rationale

- Labs focus on new skills and concepts
  - Data Lab: computer arithmetic, digital logic.
  - Bomb Lab: assembly language, using a debugger, understanding the stack.
  - Malloc Lab: data layout, space/time tradeoffs
  - Shell Lab: processes, concurrency, process control,

Textbooks

- Required:
  - Bryant & O'Hallaron, "Computer Systems: A Programmer's Perspective", PH 2003. (csapp.cs.cmu.edu)
- Recommended:
  - Kernighan & Ritchie (K&R), "The C Programming Language, Second Edition", PH 1988
  - R. Stevens and S. Rago, "Advanced Programming in the Unix Environment", AW 2005

Policies

- Late policy
  - 10% off per day
- Cheating
  - What is cheating?
    - Sharing code: either by copying, retyping, looking at, or supplying a copy of a file.
  - What is NOT cheating?
    - Helping others use systems or tools.
    - Helping others with high-level design issues.
    - Helping others debug their code.
  - Penalty for cheating:
    - Removal from course with failing grade.

Facilities

- TLAB (Tech F-252, on the bridge to Ford)
  - A cluster of Linux machines (e.g., TLAB-11.cs.northwestern.edu)
- You should all have TLAB accounts by now; problems? contact root (root@eecs.northwestern.edu)
- Need physical access to TLAB? Contact Carol Surma (carol@rhodes.ece.northwestern.edu)

Let’s get started…

Hello World

- What happens when you run `hello.c` on your system?

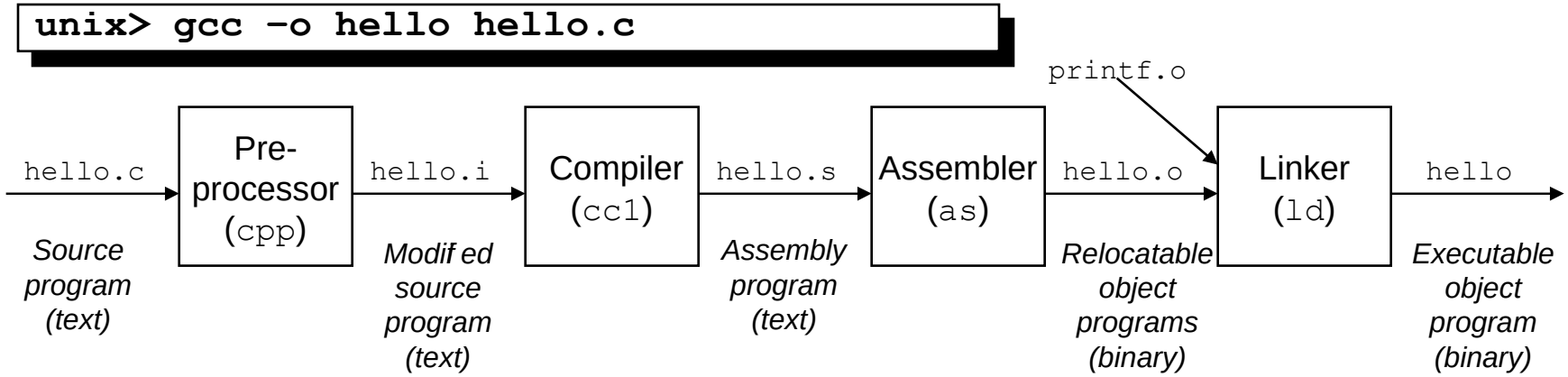
```
/* hello world */
#include <stdio.h>

int main()
{
   printf("hello, world\n");
}
```

Information is bits + context

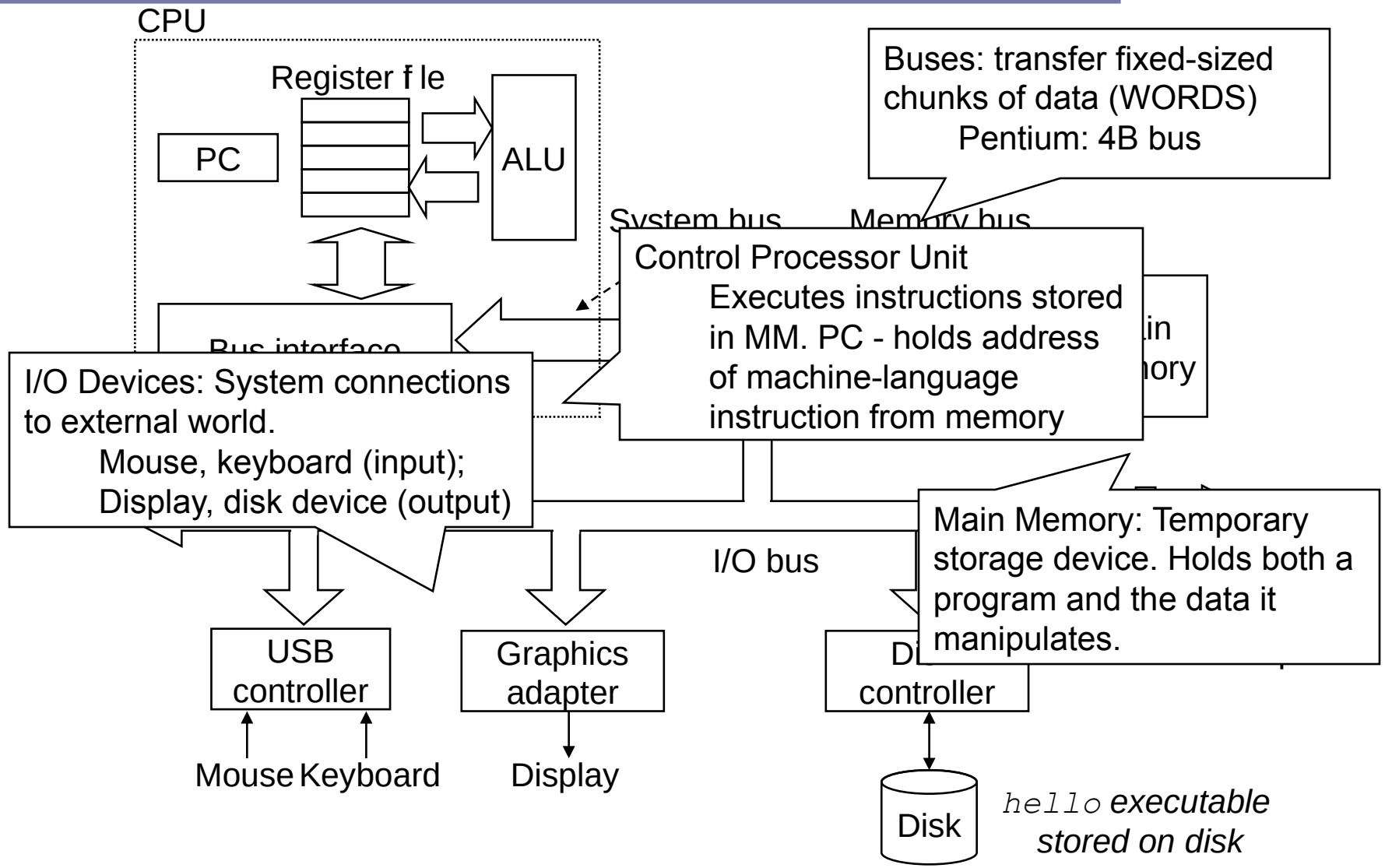
- `hello.c` is source code
  - Sequence of bits (0 or 1)
  - Manipulated in 8-bit chunks called bytes
  - Text files: bytes interpreted as ASCII characters
    - Each byte has an integer value that corresponds to some character, usually ASCII but international standards becoming more common
    - E.g., '#' -> 35
  - Binary files
    - Bytes can be data, e.g., bits in an image
    - Or parts of code, e.g., 35 as a machine instruction
- Context is important
  - The same sequence of bytes might represent a character string or machine instruction

Programs translate bytes to bytes



- Pre-processor replaces `#include <stdio.h>` with contents of `stdio.h` in `hello.i`
- Compiler translates C to assembler source (`hello.s`)
- Assembler translates assembler to machine instructions, called object code (`hello.o`)
- Linker combines object code from precompiled object libraries, e.g., `printf()`

Hardware organization



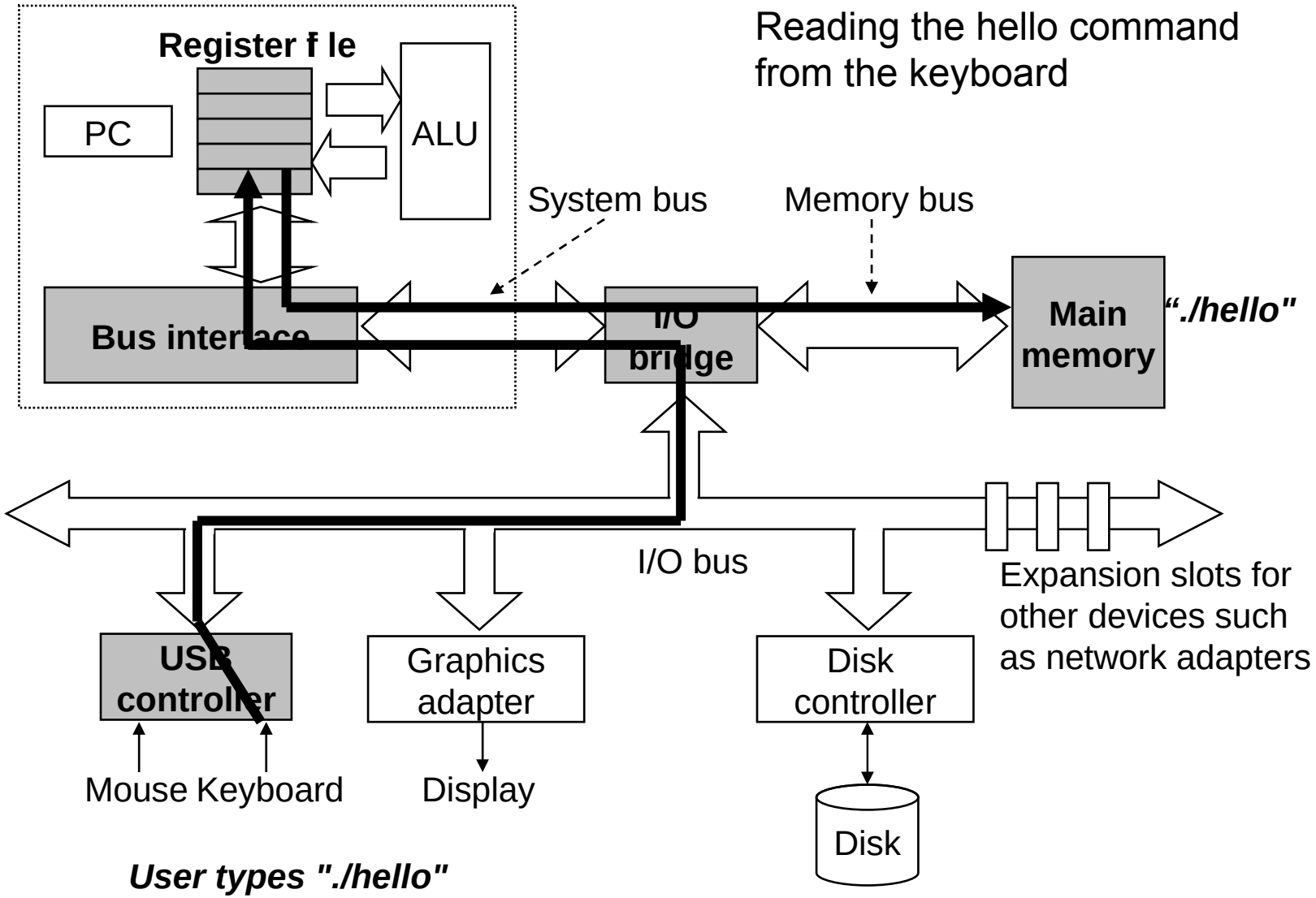
Running `Hello`

- Running `hello`

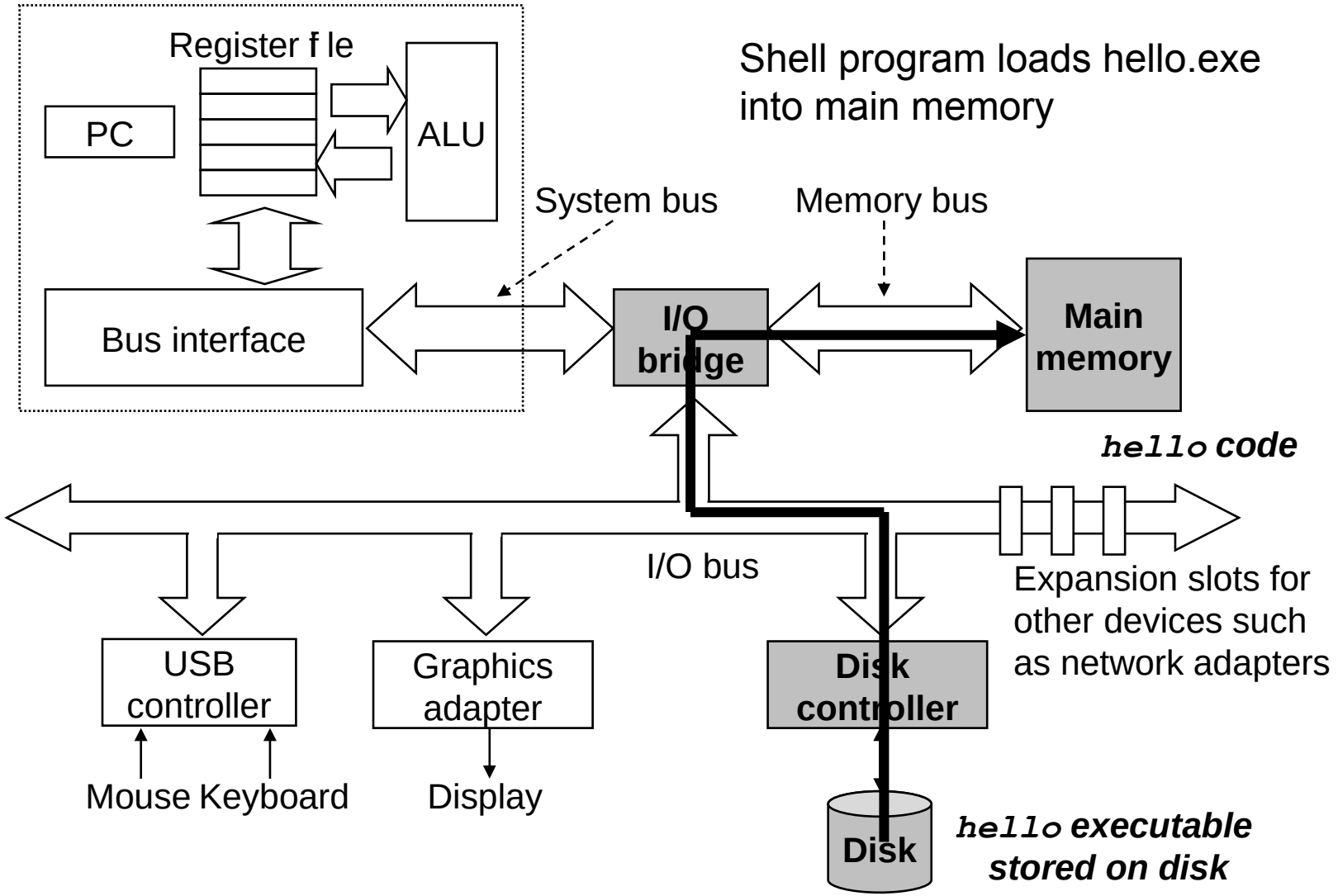
```
unix> ./hello
hello, world
unix>
```

- What's the shell?
  - Another program (many available in Unix)
- What does it do?
  - prints a prompt
  - waits for you to type command line
  - loads and runs hello program …

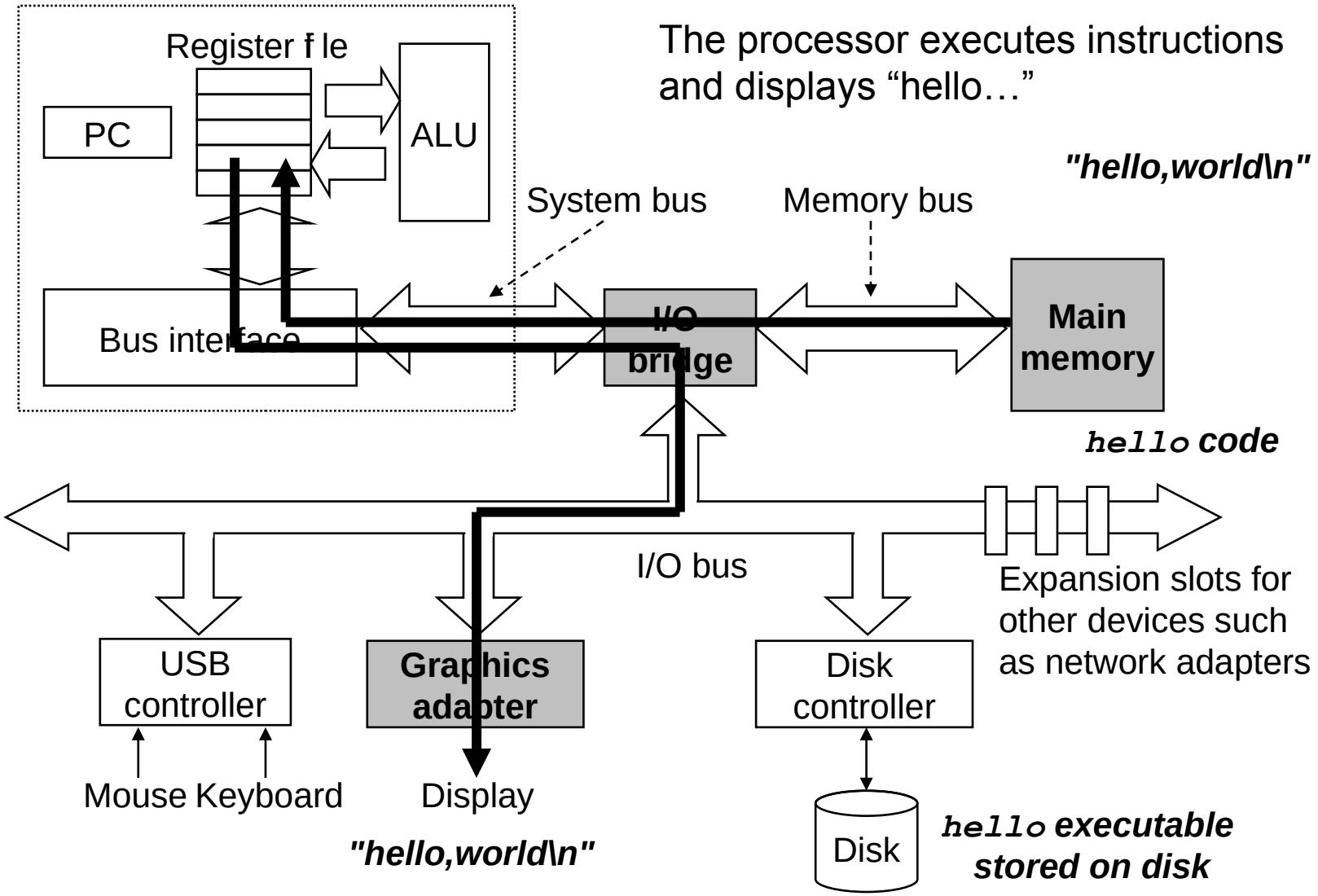
Running `Hello`



Running `Hello`

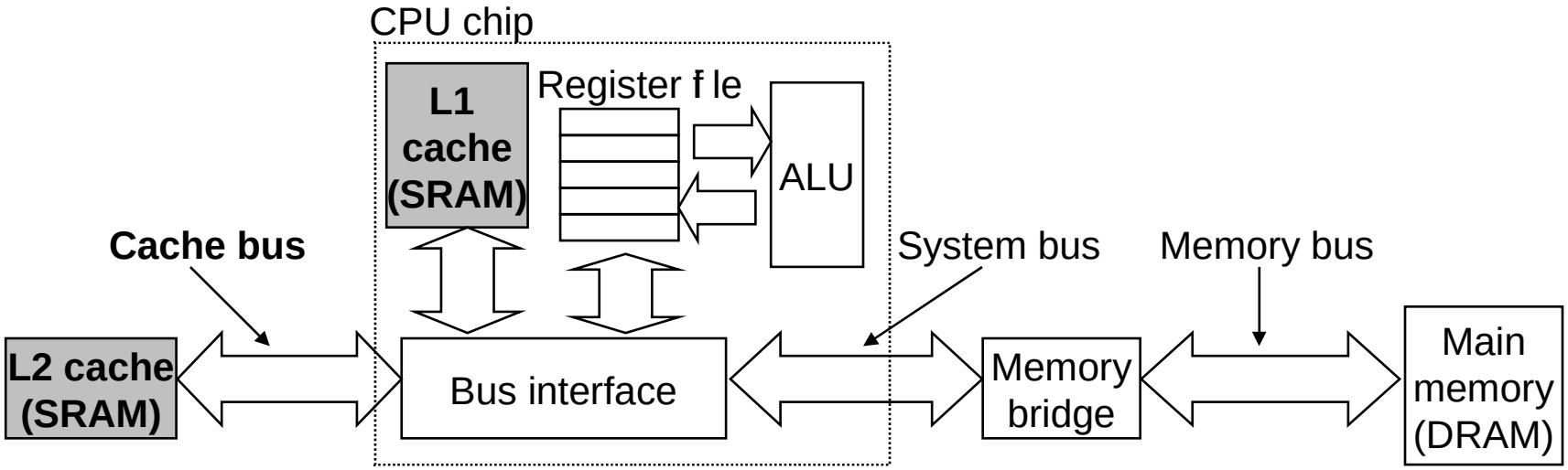


Running `Hello`



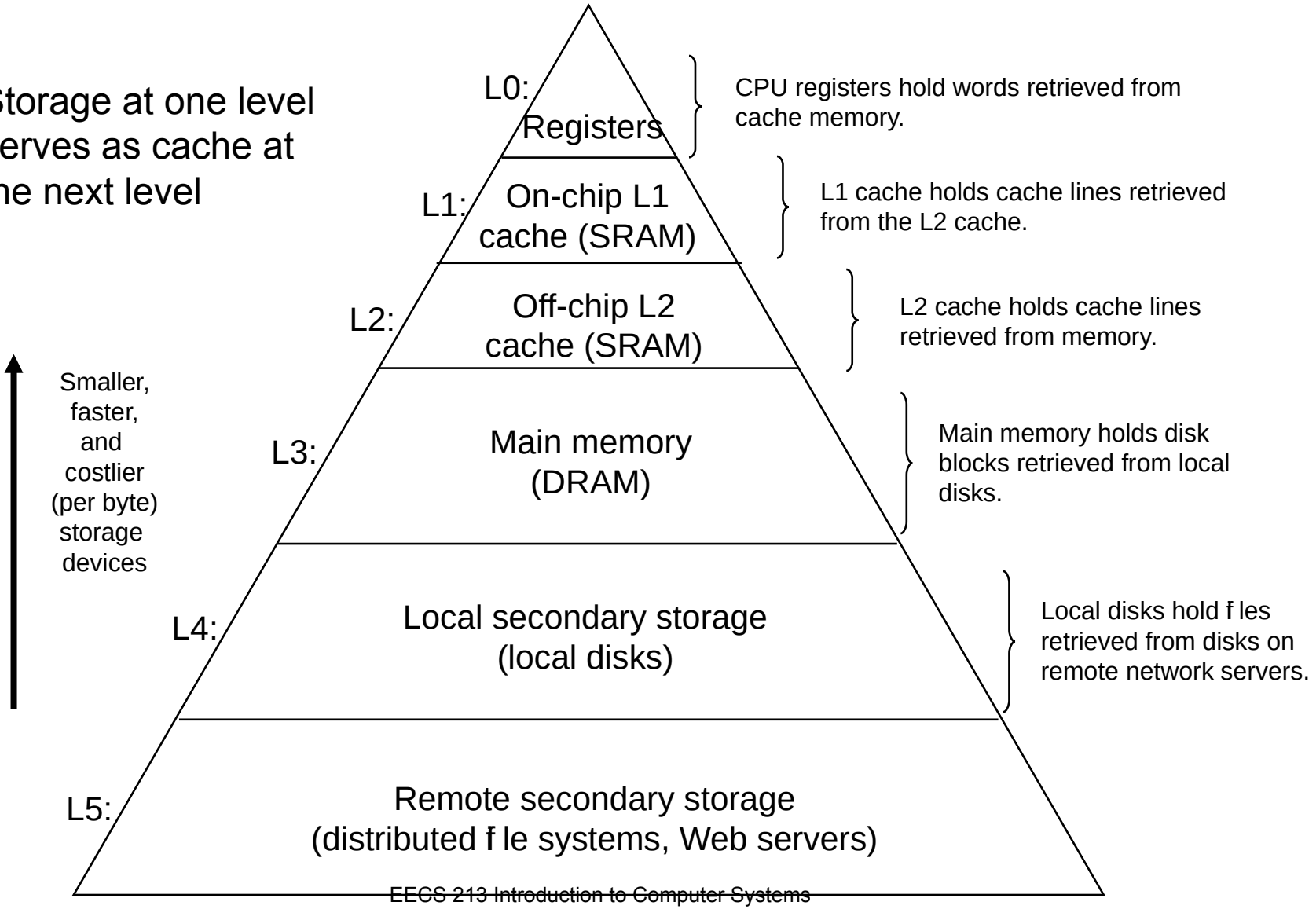
Caches matter

- System spends a lot of time moving info. around
- Larger storage devices are slower than smaller ones
  - Register file ~ 100 Bytes & Main memory ~ millions of Bytes
- Easier and cheaper to make processors run faster than to make main memory run faster
  - Standard answer – cache



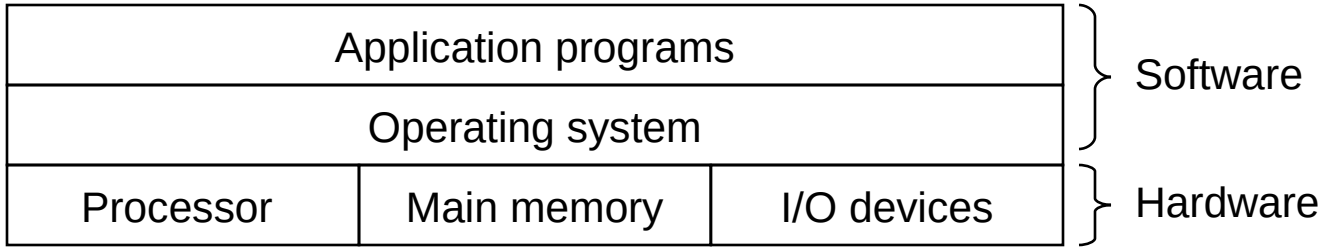
Storage devices form a hierarchy

Storage at one level serves as cache at the next level



Operating System

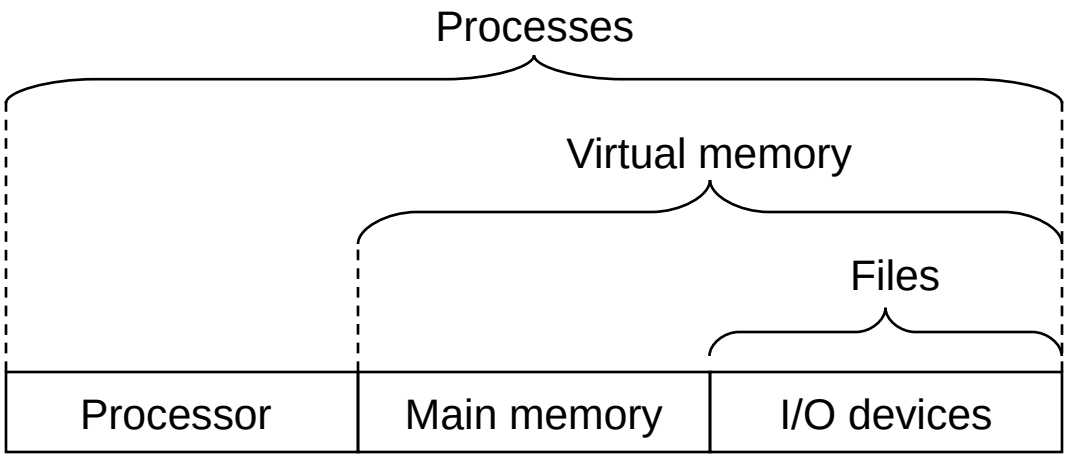
- OS – a layer of software interposed between the application program and the hardware



- Two primary goal
  - Protect resources from misuse by applications
  - Provide simple and uniform mechanisms for manipulating low-level hardware devices

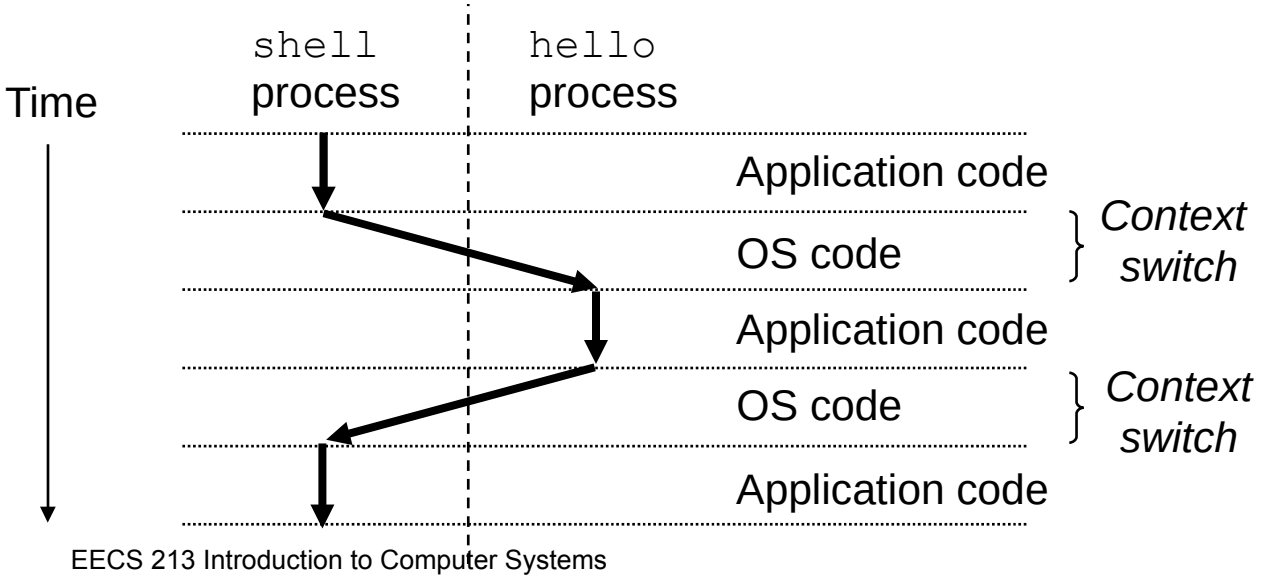
OS Abstractions

- Files – abstractions of I/O devices
- Virtual Memory – abstraction for main memory and I/O devices
- Processes – abstractions for processor, main memory, and I/O devices



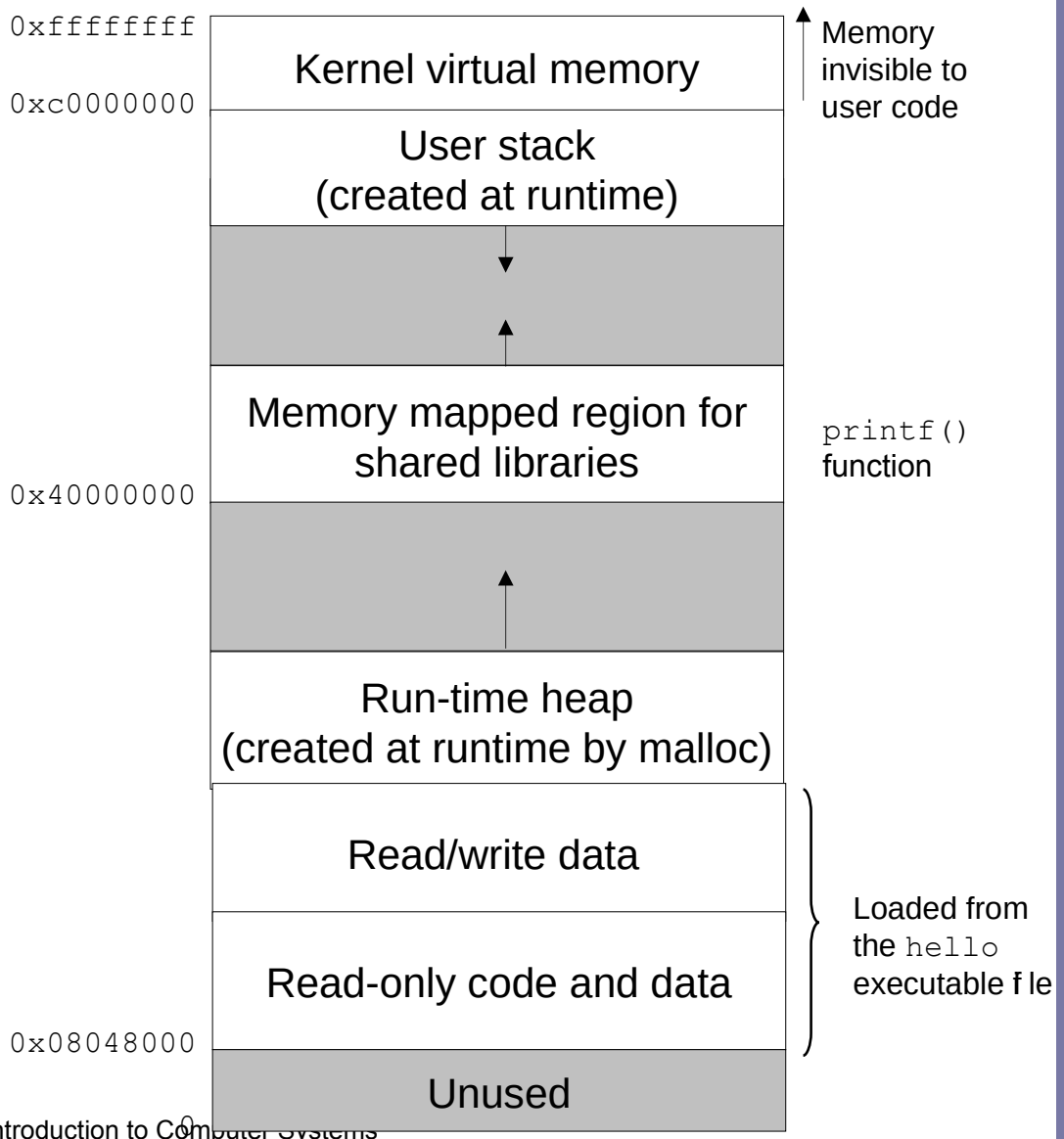
Processes

- Process
  - OS's abstraction of a running program
  - OS provides the illusion that a process is the only one there
- Context switch
  - Saving context of one process, restoring that of another one
  - Distorted notion of time



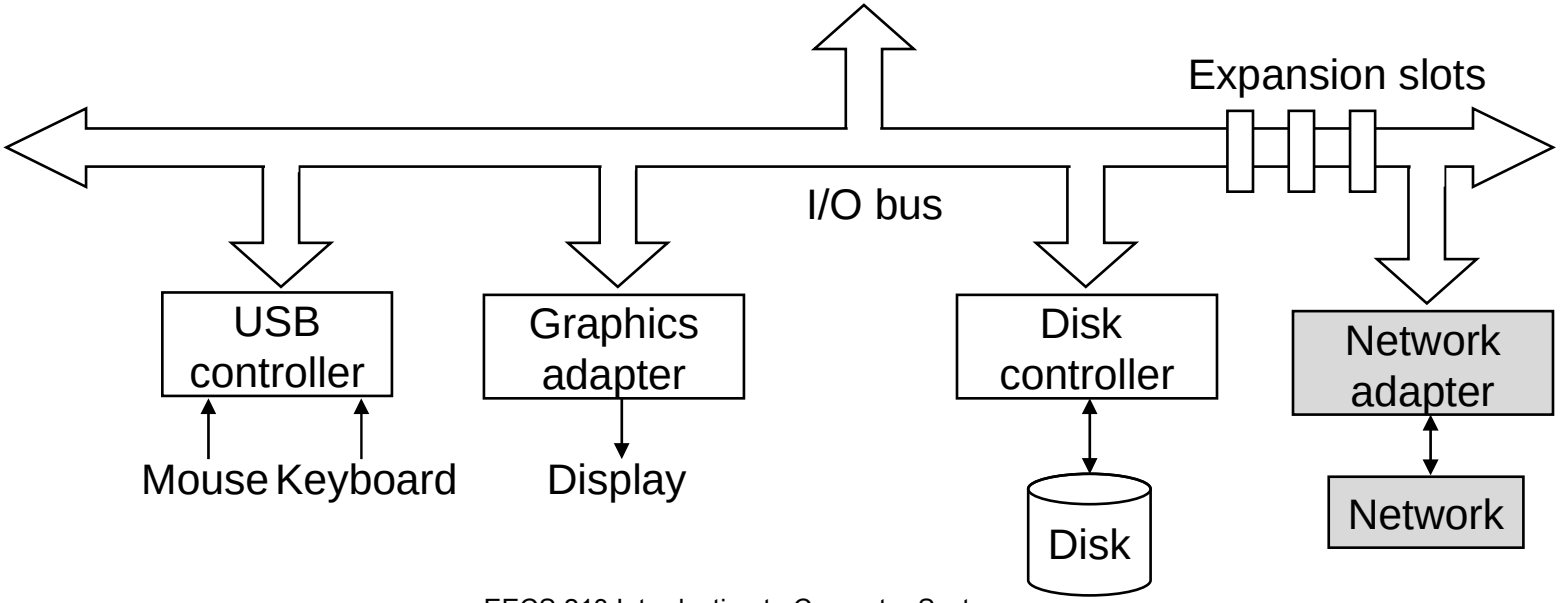
Virtual Memory

- Illusion that each process has exclusive use of a large main memory
- Example
  - Virtual address space for Linux



Networking

- Talking to other systems
- Network is treated as another I/O device
- Many system-level issues arise in presence of network
  - Coping with unreliable media
  - Cross platform compatibility
  - Complex performance issues



Conclusions

- A computer system is more than just hardware
  - A collection of intertwined hardware and software that must cooperate to achieve the end goal – running applications
- The rest of our course will expand on this