Introduction to Computer Systems

Today:

- Welcome to EECS 213
- Lecture topics and assignments

Next time:

- Bits & bytes
- and some Boolean algebra

Welcome to Intro. to Computer Systems

- Everything you need to know
  http://aqualab.cs.northwestern.edu/classes/EECS213/eecs-213-s10.html
- Your instructor: Fabián E. Bustamante
- Your TA: John Otto
- Communication channels:
  - Course webpage
  - News
  - eecs-213@cs

Course theme

Abstraction is good, but don’t forget reality!

- Courses to date emphasize abstraction
  - Abstract data types, asymptotic analysis, …
- Abstractions have limits
  - Especially in the presence of bugs
  - Need to understand underlying implementations
- Useful outcomes
  - Become more effective programmers
  - Prepare for later “systems” classes in CS & ECE
  - What do you need?
    - EECS 211 or equivalent & Experience with C or C++ - required
    - EECS 311 - useful

Course perspective

- Most systems courses are builder-centric
  - Operating Systems: Implement portions of an OS
  - Compilers: Write compiler for simple language
  - Networking: Implement and simulate network protocols
- This course is programmer-centric
  - To show how by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
  - Not just a course for dedicated hackers
    - We bring out the hidden hacker in everyone
  - Cover material in this course that you won’t see elsewhere

Textbooks

- Required
  - Bryant & O'Hallaron, "Computer Systems: A Programmer's Perspective", PH 2010.
- Recommended
  - Kernighan & Ritchie (K&R), "The C Programming Language, Second Edition", PH 1988
  - R. Stevens, "Advanced Programming in the Unix Environment", AW 1992; *there's a new edition by R. Stevens and S. Rago, AW 2005*

Course components

- Lectures
  - Higher level concepts
  - 10% of grade from class participation
- Labs (4)
  - The heart of the course – in-depth understanding
  - 12.5% of grade each
  - Working on teams of 2
- Homework assignments (4)
  - 10% of grade
- Exams – midterm & final
  - 20% of grade each

Policies

- Late policy
  - 10% off per day (up to 5 days late)
- 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

Facilities

- Tlab (Tech F-252, on the bridge to Ford) and Wilkinson Lab (3rd floor).
- You should all have accounts by now; problems? contact root (root@eecs.northwestern.edu)
- Need physical access to labs? Contact Carol Surma (carol@rhodes.ece.northwestern.edu)

Lab rationale

- Teach new skills and concepts
  - Data – Computer arithmetic, digital logic

    Out: 3/29 In: 4/14
  - Bomb – Assembly language, using a debugger, understanding the stack

    Out: 4/14 In: 5/3
  - Malloc – Data layout and organization, space/time tradeoffs

    Out: 5/3 In: 5/17
  - Shell – Processes, concurrency, process control, signals and signal handling

    Out: 5/17 In: 6/2

Some topics covered

- Programs and data
  - Bits arithmetic, assembly, representation of C control …
- Memory hierarch
  - Memory technology, memory hierarchy, caches, disks, locality
- Linking & exceptional control flow
  - Object files, dynamic linking, libraries, process control, …
- Virtual memory
  - Virtual mem., address translation, dynamic storage allocation
- Concurrency
  - High level & low-level I/O, threads, …
  - …
- Includes aspects of architecture and OS throughout

Hello World

- What happens and *why* when you run “hello” on your system?

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

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

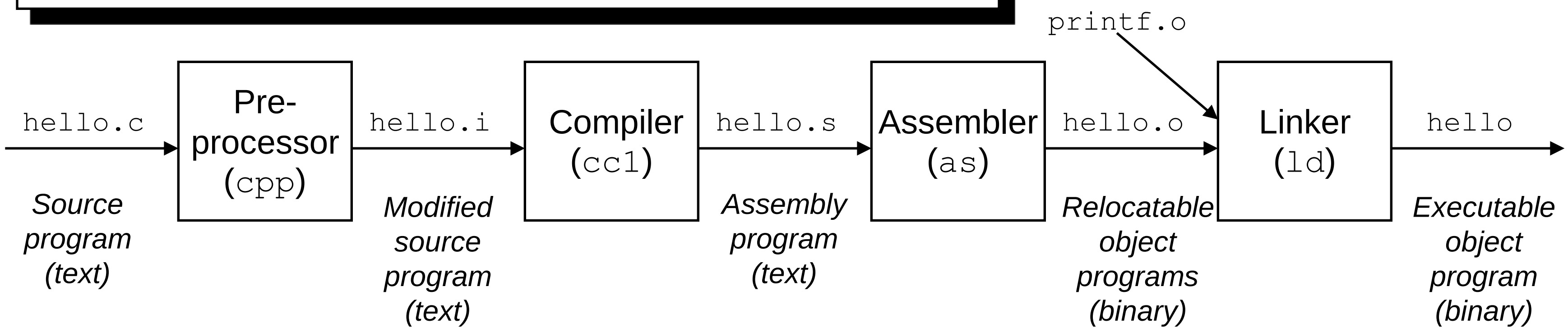
- Goal: introduce key concepts, terminology, and components

Information is bits + context

- “Hello” is a source code
  - Sequence of bits (0 or 1)
  - 8-bit data chunks are called Bytes
  - Each byte has an integer value, corresponding to some character (ASCII, e.g. ‘#’ → 35)
  - Files made up of ASCII char. → text files
  - All other files → binary files (e.g., 35 is a part of a machine command)
- Context is important
  - Same byte sequence might represent a character string or machine instruction

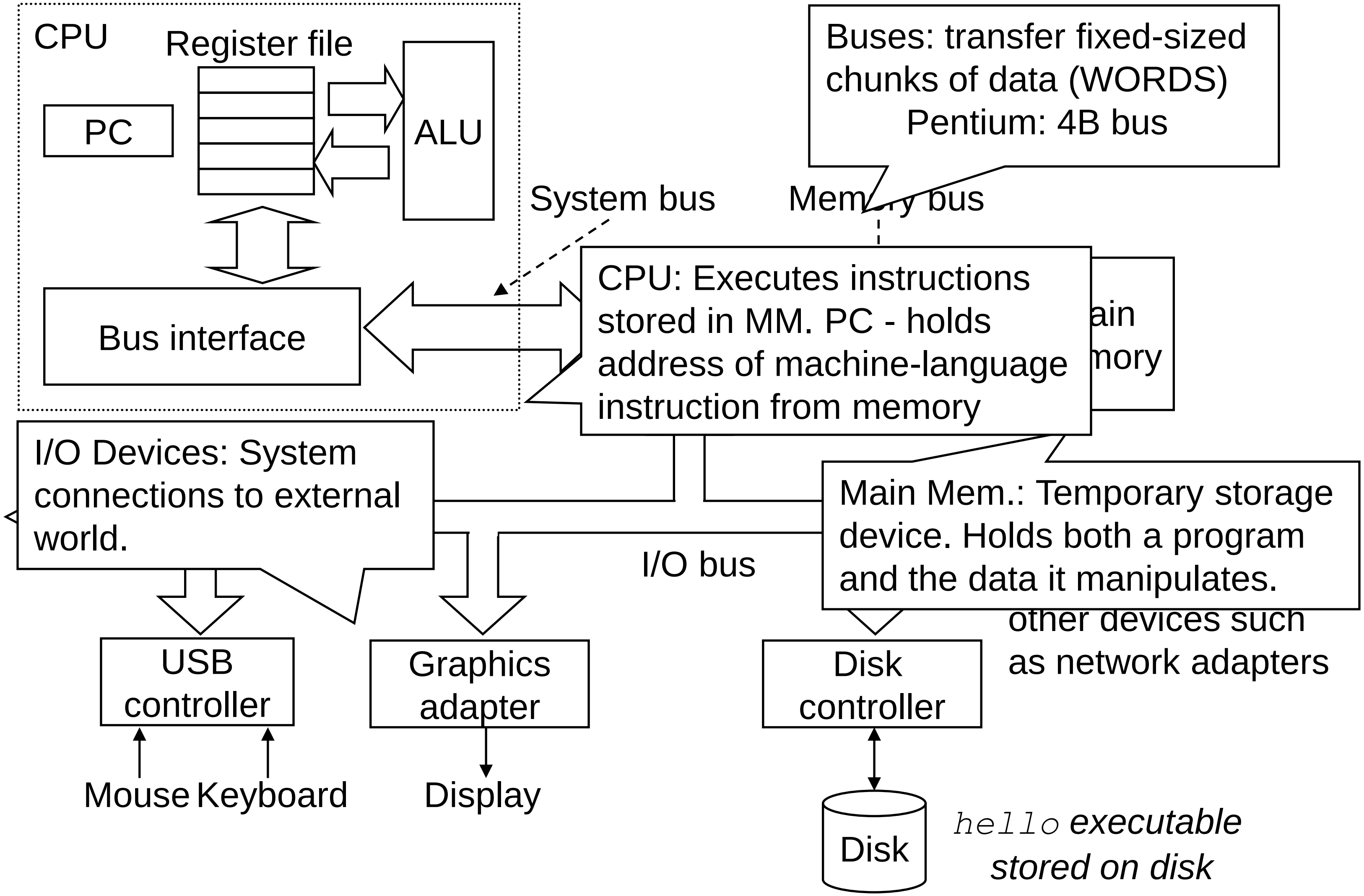
Programs translated by other programs

```
unix> gcc -o hello hello.c
```



- Pre-processing
  - E.g., #include <stdio.h> is inserted into hello.i
- Compilation (.s)
  - Each statement is an assembly language program
- Assembly (.o)
  - A binary file whose bytes encode mach. language instructions
- Linking
  - Get printf() which resides in a separate precompiled object file

Hardware organization



Running `Hello`

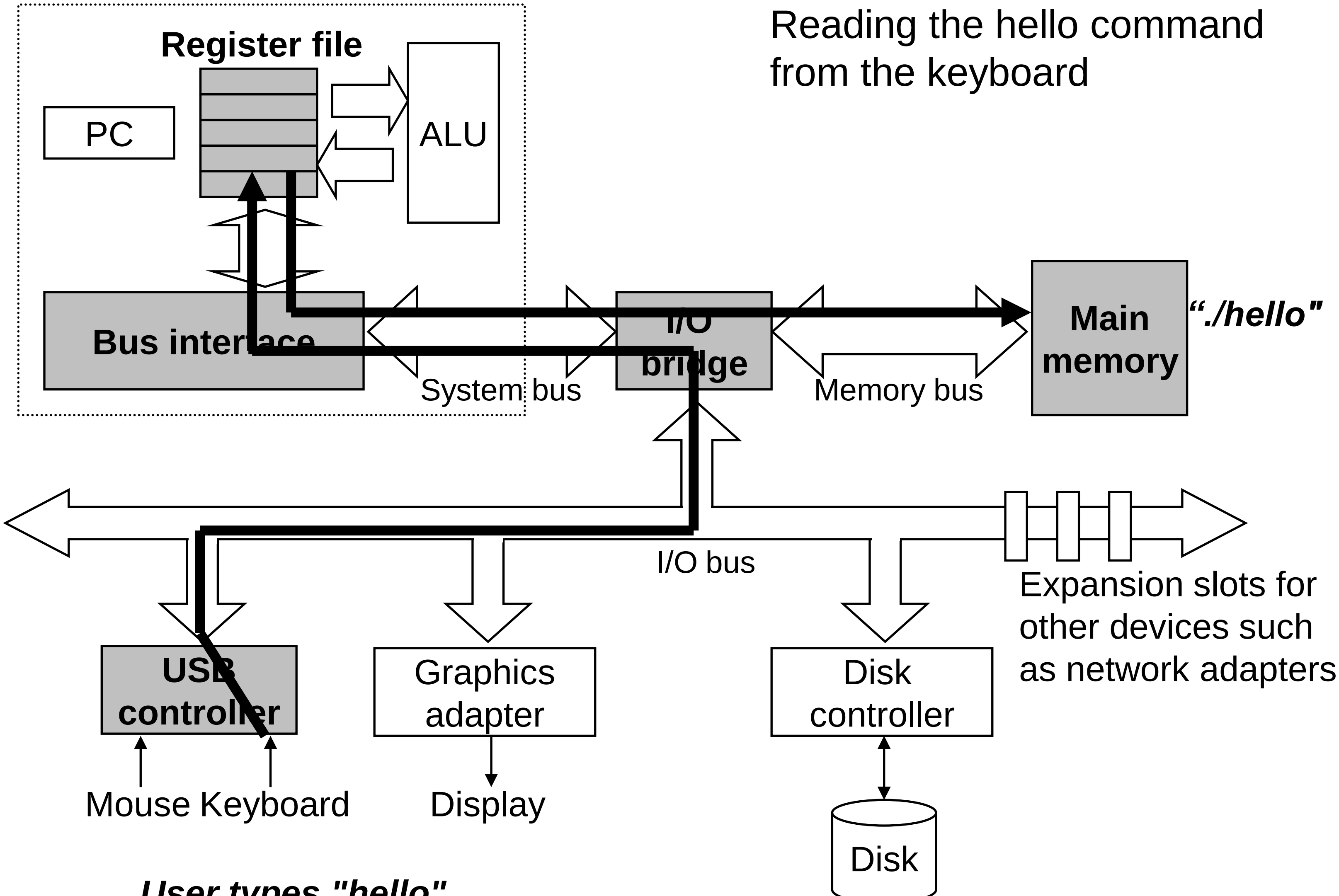
- Running `hello`

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

- What’s the shell?
- What does it do?
  - prints a prompt
  - waits for you to type command line
  - loads and runs hello program …

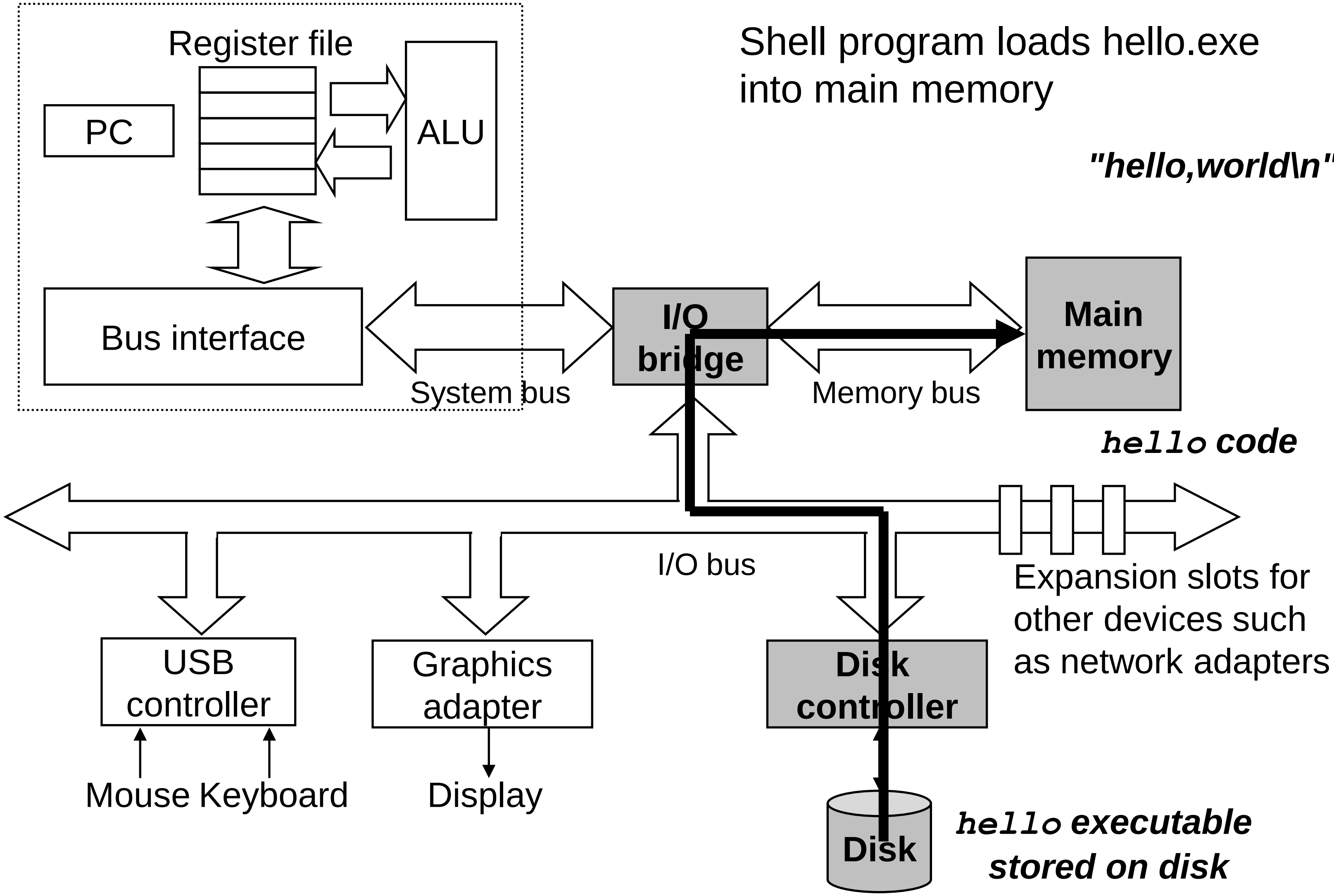
Running `Hello`

Reading the hello command from the keyboard



User types "hello"

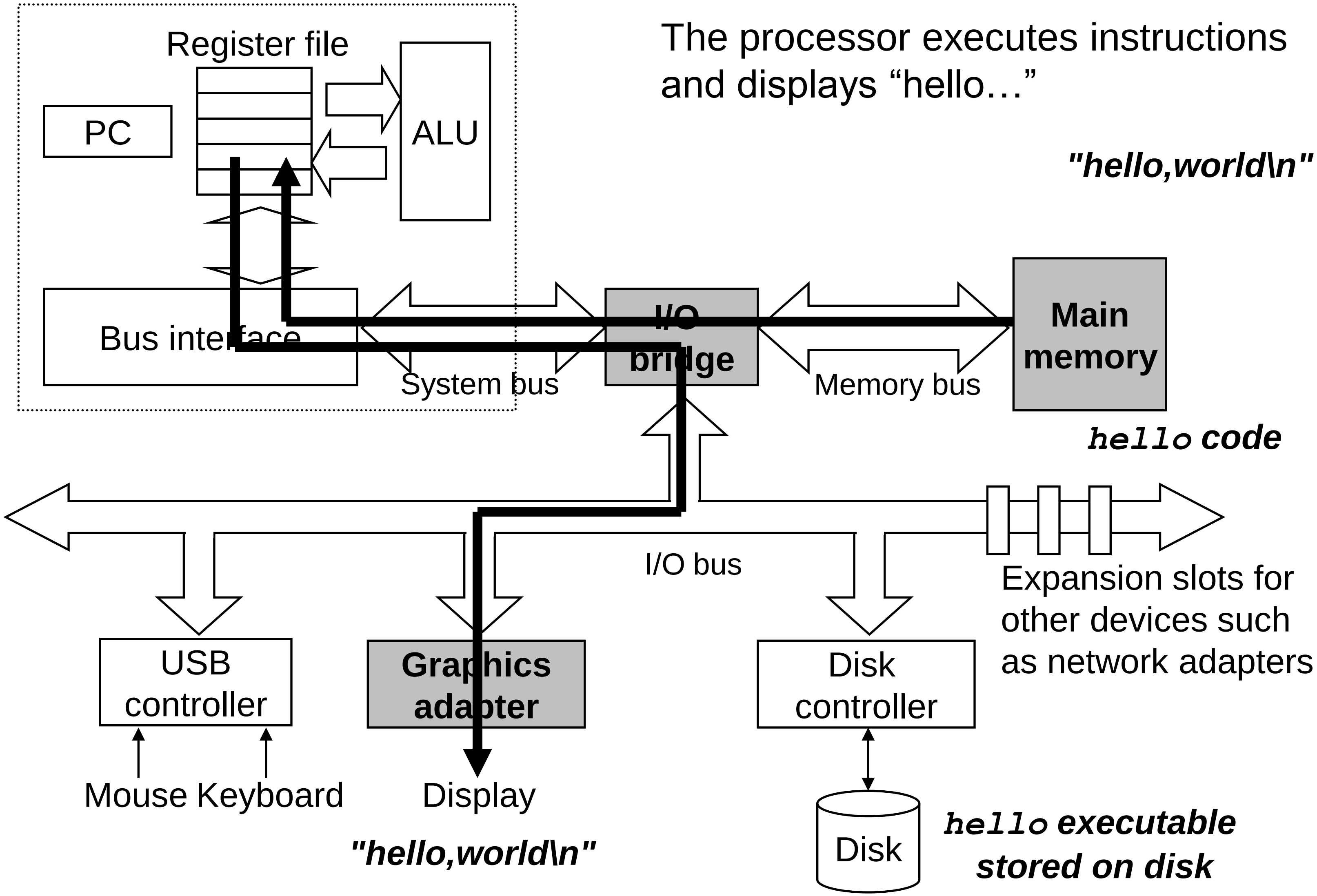
Running `Hello`



Shell program loads hello.exe into main memory

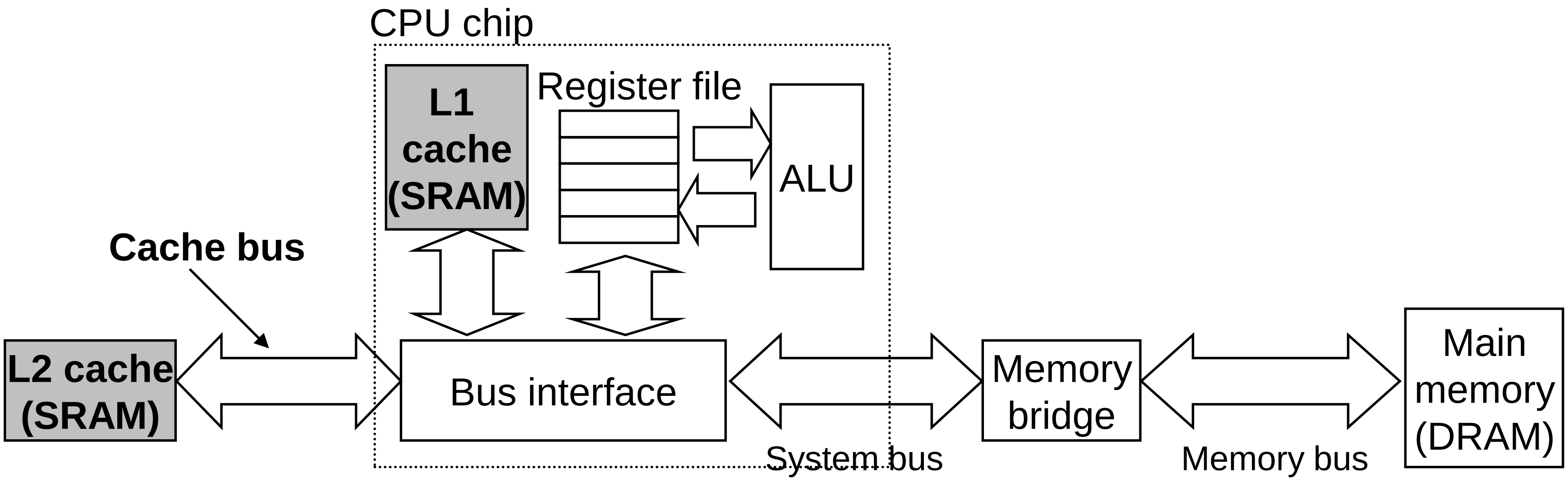
Running `Hello`

The processor executes instructions and displays "hello…"



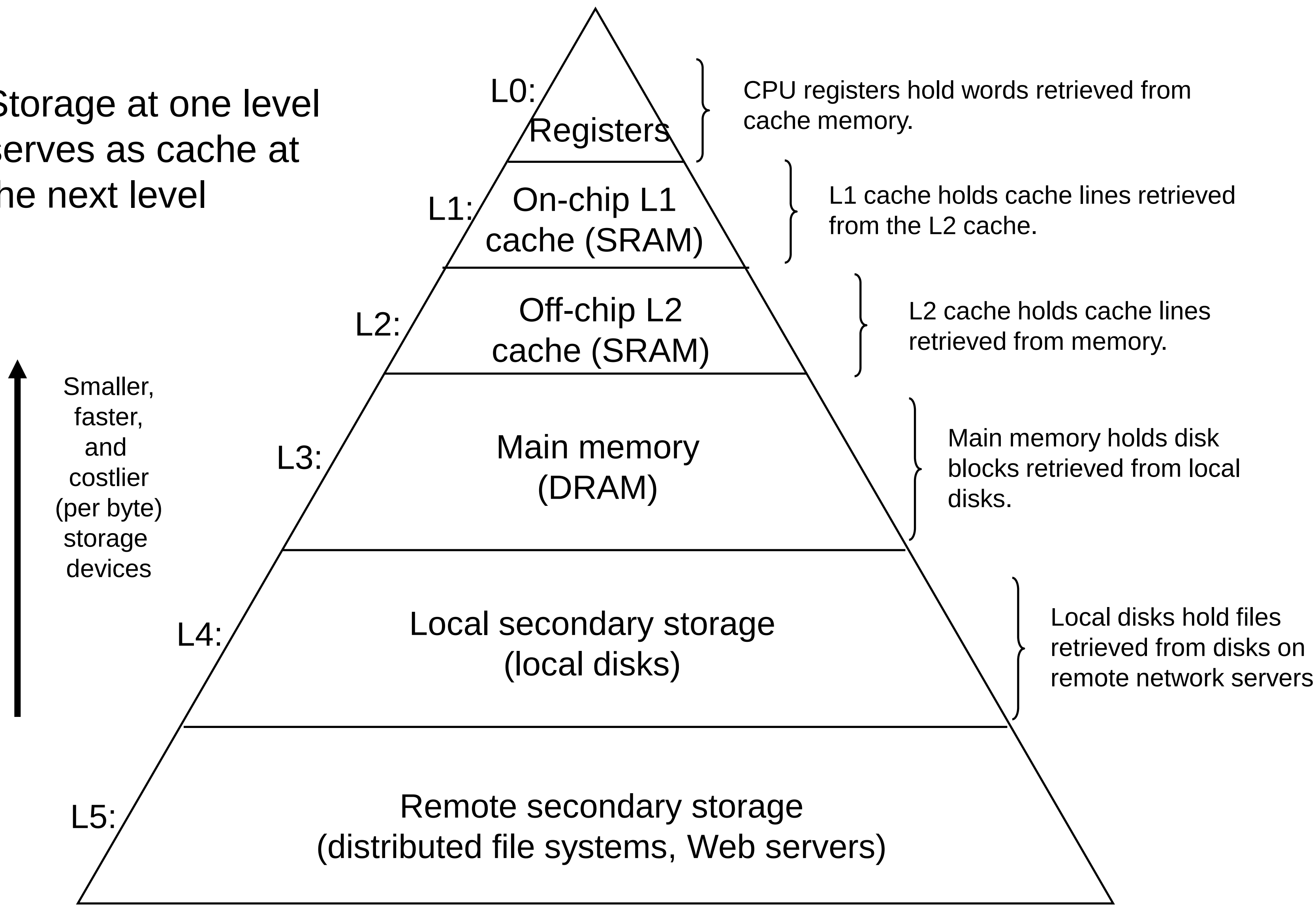
Caches matter

- 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



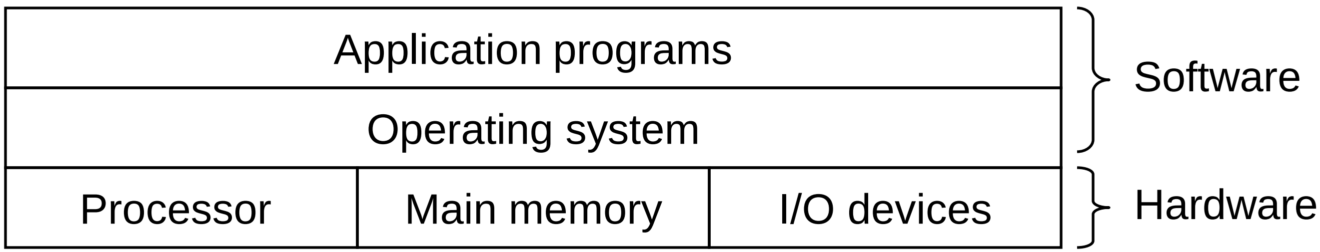
Storage devices form a hierarchy

Storage at one level serves as cache at the next level



Operating system

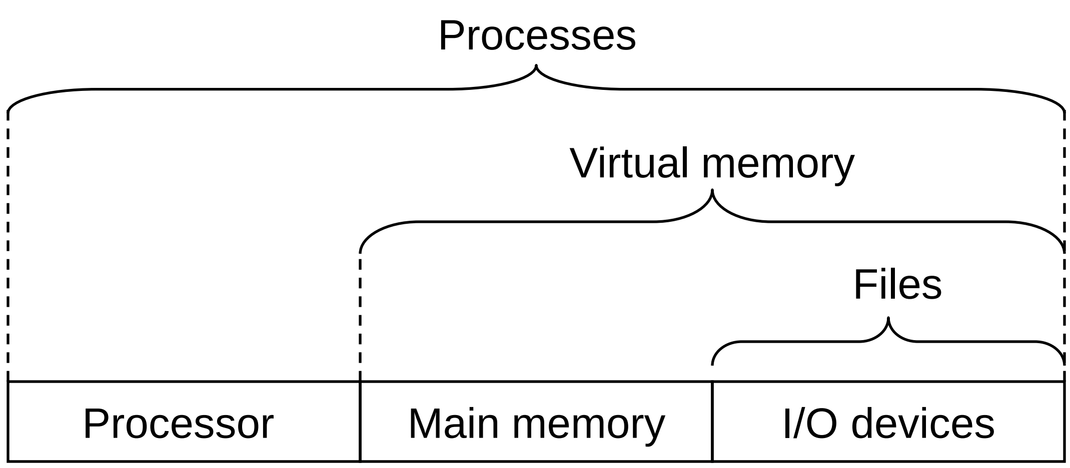
- OS – a layer of software interposed between the application program and the hardware



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

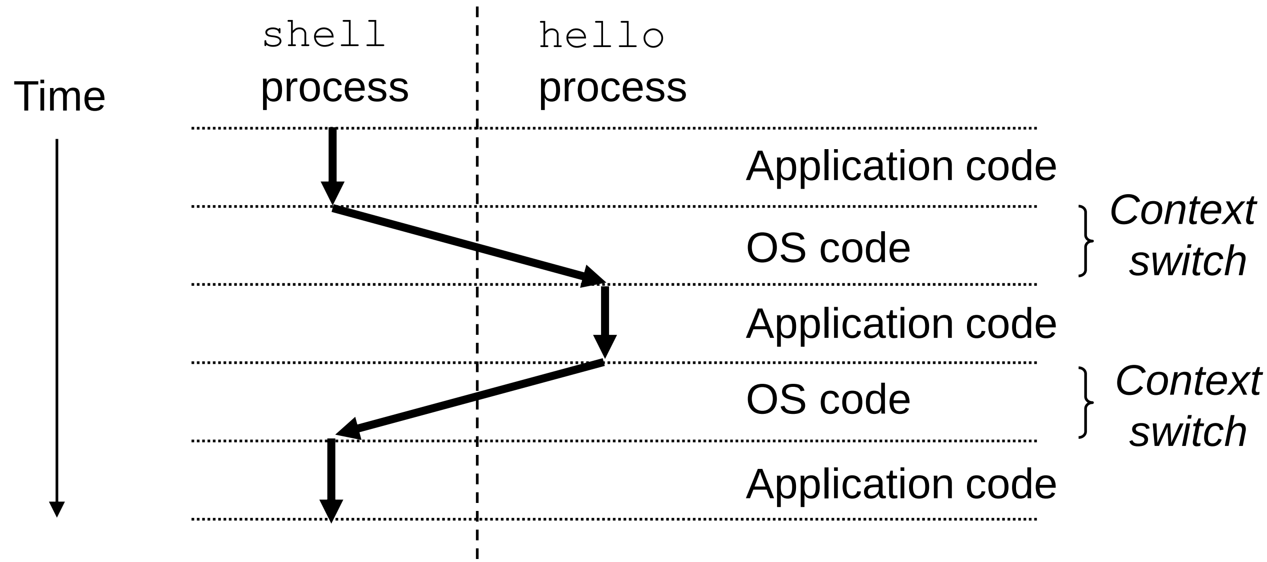
OS Abstractions

- 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



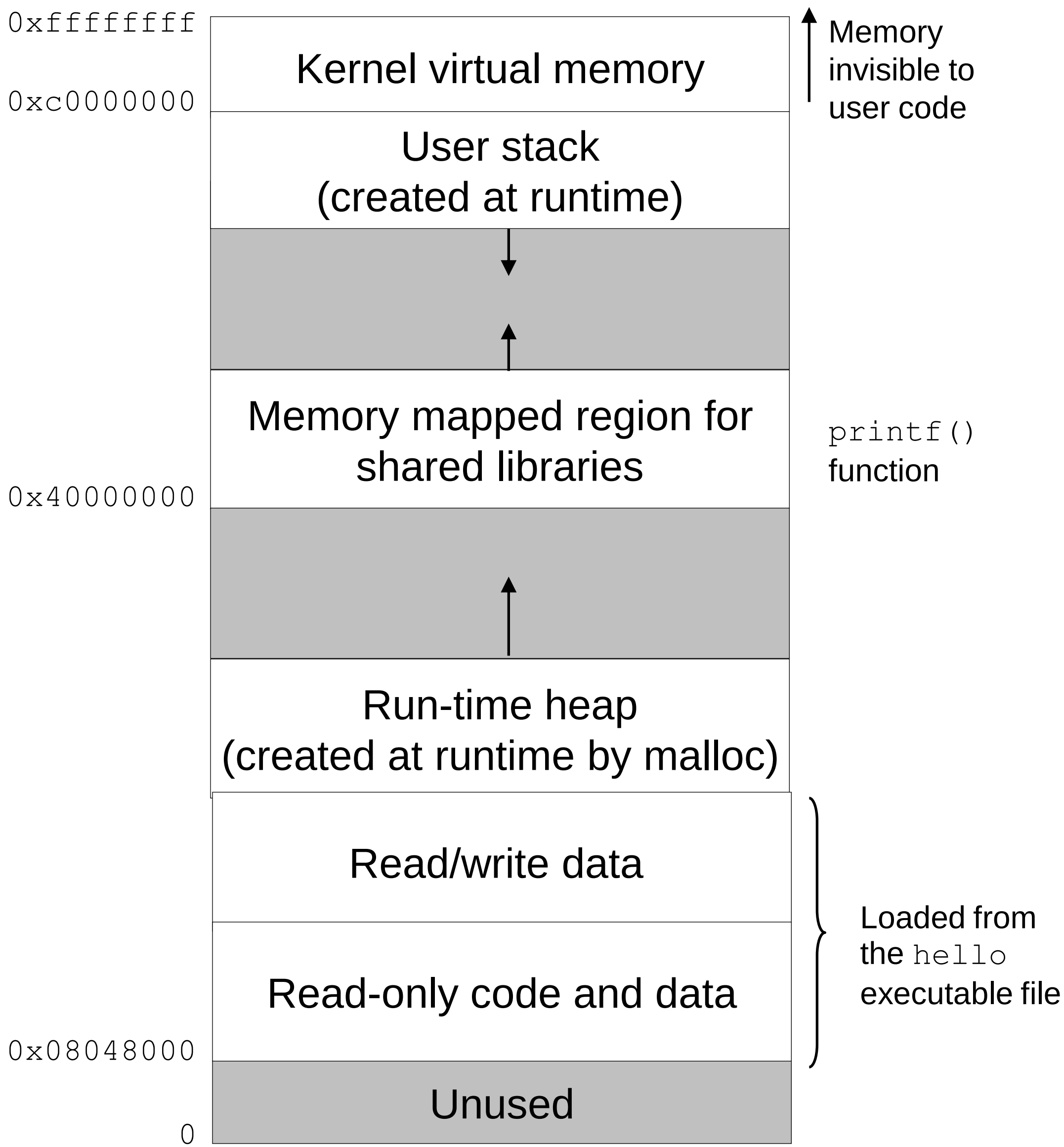
Processes

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



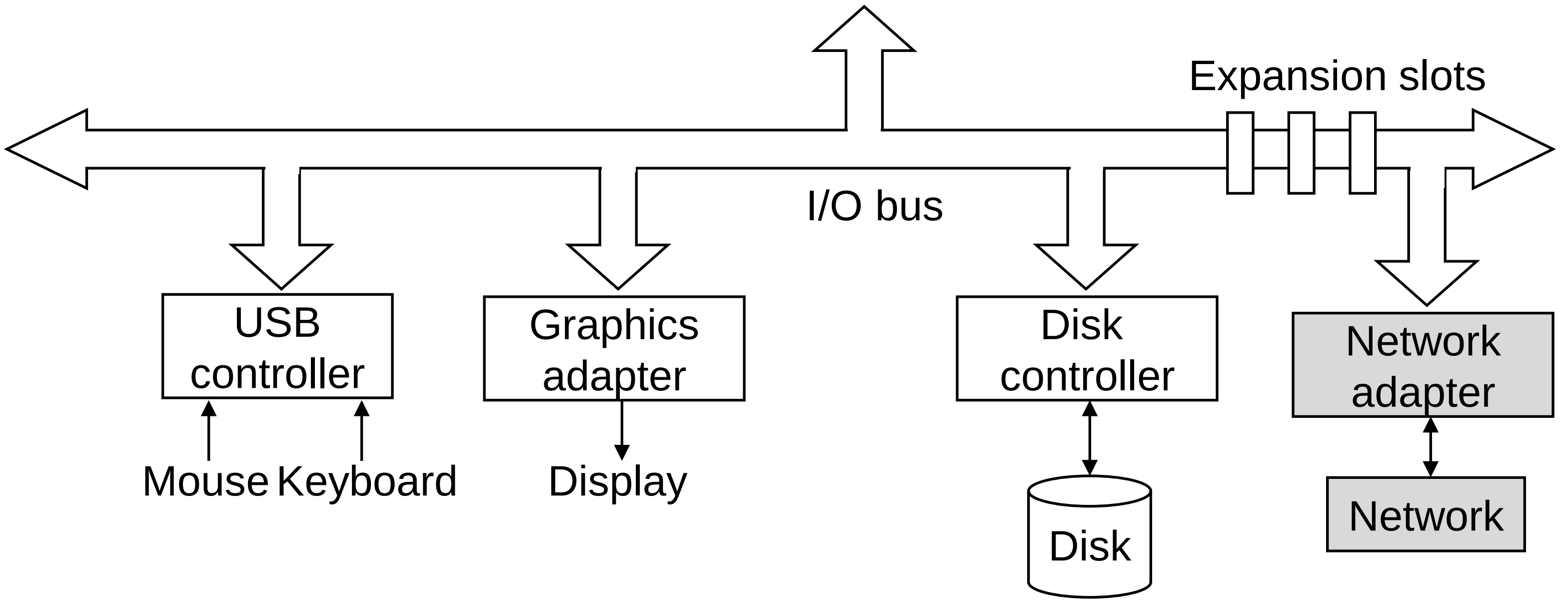
Virtual memory

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



Networking

- Talking to other systems
- Network – seen 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 HW & SF that must cooperate to achieve the end goal – running applications
- The rest of our course will expand on this