CIS 415:
Operating Systems
Introduction

Prof. Kevin Butler

Spring 2012

Welcome

- CIS 415: Operating Systems
  - Section 1: Tuesday/Thursday, 12:00-13:20, 30 Pacific
  - Section 2: Tuesday/Thursday, 14:00-15:20, 200 Deschutes
- Ensure that you're registered: this course & lab
- Instructor (me): Kevin Butler
  - 343 Deschutes Hall
  - Office Hours: Wednesday 10AM-12PM unless otherwise specified and by appointment
  - email: butler@cs.uoregon.edu

TAs and Labs

- TAs: Getty Ritter & Zhiyi Wu (Ryan Woo)

  ‣ gdritter@cs.uoregon.edu, zhiyiw@cs.uoregon.edu

  ‣ Office hours:

    - Wed 2-3 PM (Getty)

    - Mon 3-4 PM, Thurs. 3:30-4:30 PM (Ryan)

  ‣ Lab Sections: Klamath Labs

    - will cover practical skills for class

      ‣ Tues. 3-4 PM (Getty overseeing)

      ‣ Thurs. 3-4 PM (Getty overseeing)

      ‣ Fri. 4-5 PM (Ryan overseeing)

Resources

- Textbook

  ‣ Silberschatz, Galvin, and Gagne, *Operating System Concepts*, 8/e

  ‣ Recommended: UNIX/C programming books (on website)

- Web page:

  ‣ http://www.cs.uoregon.edu/classes/12S/cis415/

- Mailing list:

  ‣ cis415@cs.uoregon.edu (first mail by tommorow)

- Course Oracles: Joe Pletcher, Hannah Pruse

 ‣ pletcher@cs.uoregon.edu, hpruse@cs.uoregon.edu

Course Structure

- Lectures (what you're in now)

  ‣ focus on core OS content, also worksheets and quizzes (in-class exercises)

- Lab Sections (Run by the TAs)

  ‣ programming assignment help, tutorials, practice, material on C and UNIX, threads, signals, etc

- Grading

  ‣ 10% Assignments & Quizzes

  ‣ 20% Midterm (in-class, May 8)

  ‣ **50% Programming Projects (do not procrastinate)**

  ‣ 30% Final (Section 1: 8:00 June 13; Section 2: 13:00 June 13)

This course ...

- This course is a systems course covering fundamental and applied topics in computer operating systems, including:

  ‣ system calls and interfaces, processes, concurrent programming, virtualization, resource scheduling and management (CPU, memory, I/O), virtual memory, deadlocks, distributed synchronization, filesystems, storage, security, other topics as time permits
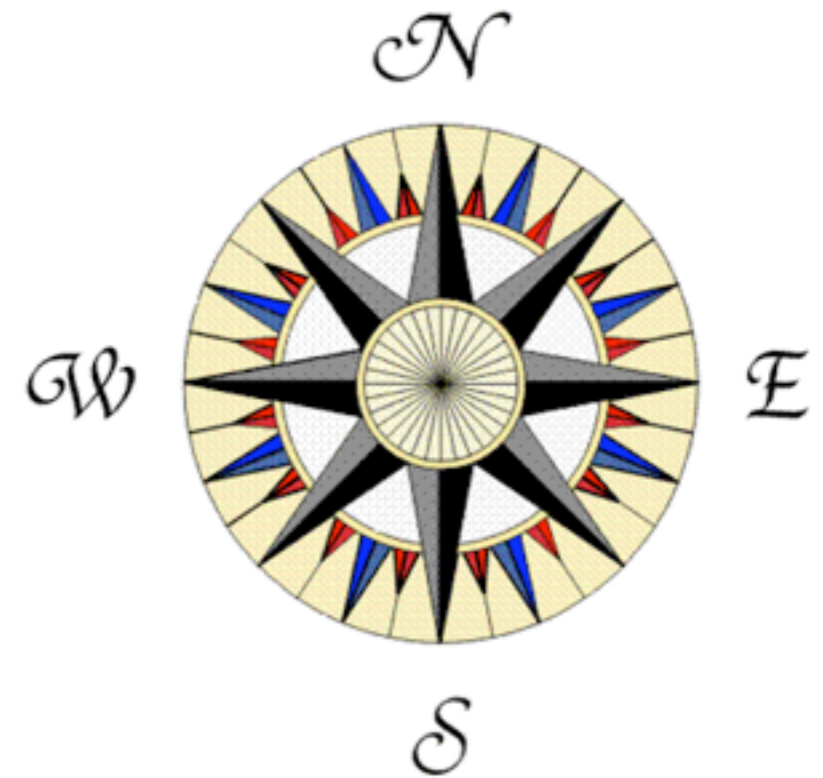
You need to understand ...

- Computer organization and architecture (CIS 314)

- Data structures and algorithms (CIS 313)

- C and UNIX programming environments (you can learn this as you go) (CIS 323, CIS 330)

- How to look things up from source material, online documentation, books, and web sites

Goals

‣ My goal: *to provide you with the tools to understand fundamentals of modern operating systems.*

  ‣ Basic technologies

  ‣ Engineering/research trade-offs

  ‣ In-depth practical OS knowledge and systems programming experience

- *This is going to be a challenging course.* The key to success is sustained effort. Failure to keep up with readings and assignments will likely result in poor grades, and ultimately little understanding of the course material.

- Pay-off: fundamental knowledge, marketable skills

Unsolicited Student Feedback

- "Thank you for the sea of knowledge you have delivered in this term. Even though I may not have been stoked about the difficulty level of the class while actually taking it, I appreciate that it was that way, and, though I doubt everyone else feels this way, I don't think you should change much."

- "Thanks for providing this project as an option. I enjoyed the larger scale project, although it caught me off-guard. It was a good opportunity to apply some of the knowledge from other classes that I hadn't really used before. It expanded my knowledge of C (and obviously UNIX). In regards to the class as a whole, I liked the volume of content and the pace of the course. Thanks for the course"

- "Thank you for a very enriching class."

- "I hope it will only take a couple of years for people to realize that those who do well in your classes also do well at getting and keeping interesting jobs." *(former student now in industry: same for grad school!)*

Course content: changes

- "I have been stunned by the misconceptions of some of our most senior engineers with respect to concurrency, but the real stunner has been the essentially complete lack of awareness of visibility issues. Visibility wasn't covered in [previous years] but it is by far our most prevalent concurrency bug (in Java). An understanding of processor-local caching, the 'volatile' keyword in Java and the "happens-before" relationship in the Java memory model is something that I will be seeking in every future employment candidate." *(Another former student now in industry)*

    ‣ Above, "visibility" refers to synchronization relating to get/set

    ‣ We're not going to be spending much time with Java, but concepts will be applicable (understand OS issues == understand Java)

- Increase focus on distributed systems & distributed concurrency (caching is already an important concept): important to ACM too

Course Projects

- Best way to understand the material is by doing

- Programming in a Linux environment and understanding systems issues

- First project will be solo, second team-based

- Building a shell in Linux, understanding how to compile a kernel, etc.

- Option: if you do very well in the first assignment and want to form a small group, you can work on an independent research project

  ‣ mobile phones, embedded systems, Windows, Linux

Course Calendar

- The course calendar as all the relevant readings, assignments and test dates

- The calendar page contains electronic links to online papers assigned for course readings.

- *Please check the website frequently for announcements and changes* to the schedule. Students are responsible for any change on the schedule.

Date	Topic	Assignments Due	Discussions (do readings before class)
04/03/12	Introduction	Assignment 0 OUT	Silberschatz, Chapter 1 Programming in C, Program Compilation, Basics, Conditionals, Looping, Arrays, Functions, Pointers, I/O, String Handling, File Access (link) C for C++ Programmers (link)
04/05/12	OS Structure and System Calls	Assignment 0 DUE	Silberschatz, Chapter 2.1-2.7 Silberschatz, Chapter 21.1-21.3 UNIX man pages: read(2), write(2), fork(2), execve(2), execl(2), wait(2), exit(2) Programming in C, Standard Libraries (link) Programming in C, Writing Larger Programs (link)
04/10/12	Processes	Project 1 OUT	Silberschatz, Chapter 3.1-3.3 Silberschatz, Chapter 21.4, 21.9 Programming in C, Process Control (link)
04/12/12	IPC and RPC		Silberschatz, Chapter 3.4-3.6 Silberschatz, Chapter 16.1, 16.2, 16.5.1-2 Programming in C, IPC (link) Programming in C, IPC:Message Queues (link) Programming in C, IPC:Shared Memory (link) Programming in C, RPC (link)
04/17/12	Threads		Silberschatz, Chapter 4.1-4.6 Programming in C, Threads (link) Programming in C, IPC: Interrupts and Signals (link)
04/19/12	Scheduling		Silberschatz, Chapter 5.1-5.8 Silberschatz, Chapter 21.5.1
04/24/12	Synchronization	Project 1 DUE	Silberschatz, Chapter 6.1-6.5 Silberschatz, Chapter 21.5.2-3 Programming in C, IPC: Semaphores (link)
04/26/12	Synch. Mechanisms	Project 2 OUT	Silberschatz, Chapter 6.6-6.8 Programming in C, Thread Programming: Synchronization (link) Programming in C, Thread Programming Examples (link) Interprocess Synchronization and Producer/Consumer (link)
05/01/12	Deadlock		Silberschatz, Chapter 7.1-7.7 Programming in C, Thread Programming Examples (look Deadlock) (link)
05/03/12	Distributed Synch.		Silberschatz, Chapter 18.1-18.7
05/08/12			Midterm Exam
05/10/12	Memory Management		Silberschatz, Chapter 8.1-8.6
05/15/12	Virtual Memory		Silberschatz, Chapter 9.1-9.4 Silberschatz, Chapter 21.6
05/17/12	VM Issues		Silberschatz, Chapter 9.5-9.9

- Consider you want to do the following:

  ‣ print "this is not a printout" on the printer

  ‣ terminate

Ceci n'est pas une pipe.

- Simple!

```
main()
{
  printf("this is not a printout\n");
}
```

Without an OS

- Get printer manual
  - figure out how to send messages to it
- Write the program
  - put the character string "this is not a printout" in a memory buffer
  - do the stuff printer requires to send buffer to it
  - go into endless loop
    - wait for someone to turn off computer eventually

- Get hold of a computer
  - it has to be all yours
- Translate your program into machine code
- Figure out how to get program into memory
  - font panel switches
  - burn a ROM
  - punch cards
  - Somehow start program
  - Turn off computer when done

With an OS

- Put program in file

  ‣ put character string "this is not a printout" in a memory buffer

  ‣ Issue *system call* to send buffer to printer

- Compile the program and tell OS to run the program file

- That's everything!

- What's a system call?

An Operating System

- Program that acts as intermediary between a computer user and the computer hardware

  ▸ i.e., an abstract interface to programs

- Goals as a *resource manager:*

  ▸ Execute user programs and making solving user problems easier

  ▸ Make the computer system convenient and safe to user
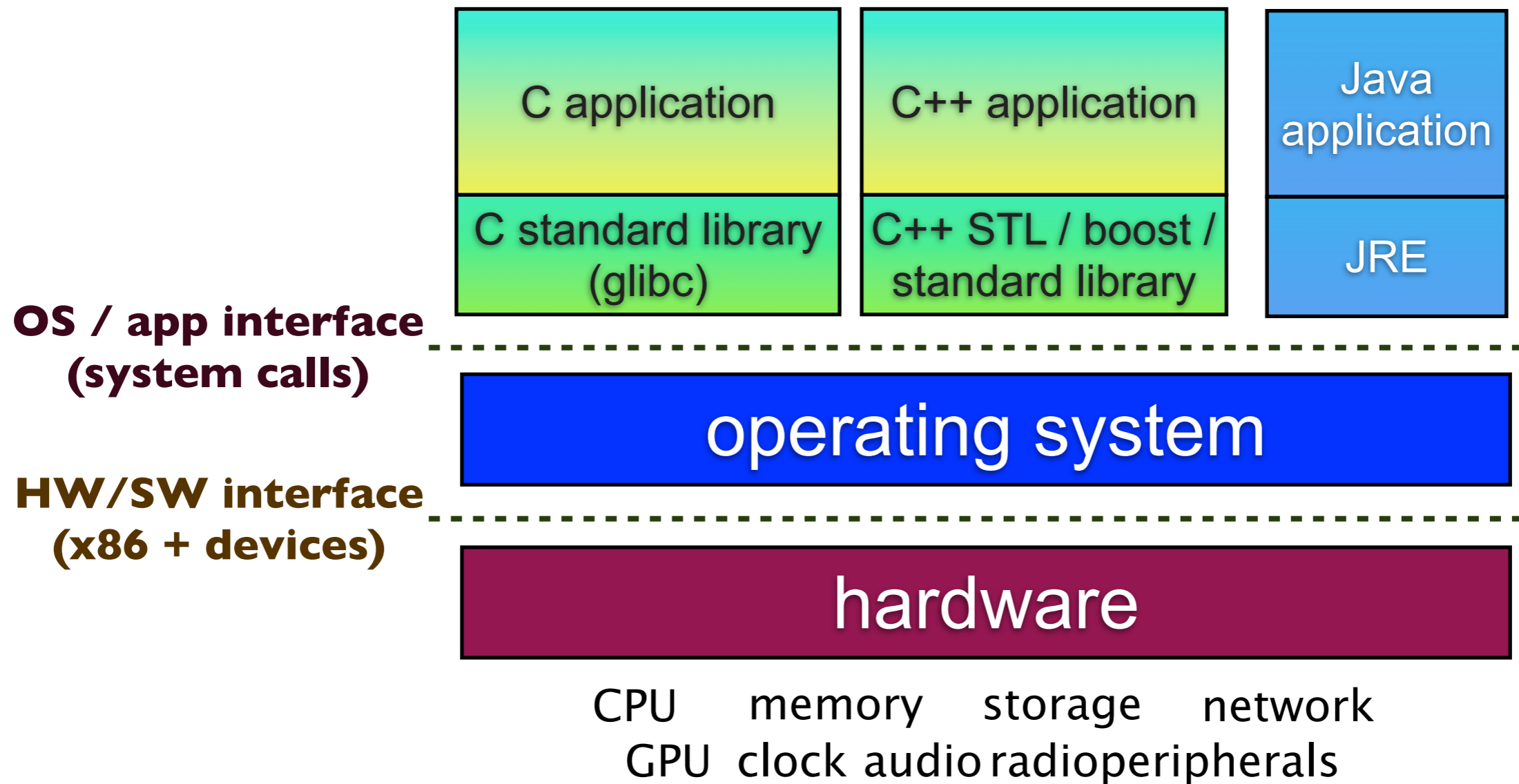
  ▸ Use the computer hardware in an efficient manner

Things your OS does

- Processes
  - ‣ hides programs from one another

- Traffic cop
  - ‣ resource management
  - ‣ who gets to run, when?

- Memory management
  - ‣ protection from other programs' mistakes

- Security
  - ‣ protection from other programs' malice

- System call interface
  - ‣ abstract, simplified interface to services
  - ‣ like a function library but communicates with the OS

- Portability
  - ‣ programs don't have to take into account details of their environment

- Device management

- Communication
  - ‣ between processes
  - ‣ to devices & networks

- GUIs and user interfaces

- Applications (e.g., web browser)

- Compilers

- Libraries

- These are implemented as user-space programs but not really the core of the OS

  ‣ Linux

  ‣ GNU/Linux

 ‣ Ubuntu vs Debian vs Fedora vs...

Where does an OS fit?

C application	C++ application	Java application
C standard library (glibc)	C++ STL / boost / standard library	JRE

OS / app interface (system calls)

operating system

HW/SW interface (x86 + devices)

hardware

CPU   memory   storage   network
GPU clock audio radioperipherals

Operating System History

- 1950s: Simplify operators' job

- 1960s: Structure, concepts, everything

- 1970s: Small and flexible (UNIX)

- 1980s: Individual user systems (PCs)

- 1990s: Internet, Windows

- 2000s: Security

- 2010s: Embedded, Highly distributed

Operating Systems (1950s)

- Primitive systems

  ▸ Little memory, programs stored on tape

- Single user

  ▸ Batch processing

  ▸ Computer executes one function at a time

- No overlap of I/O and computation
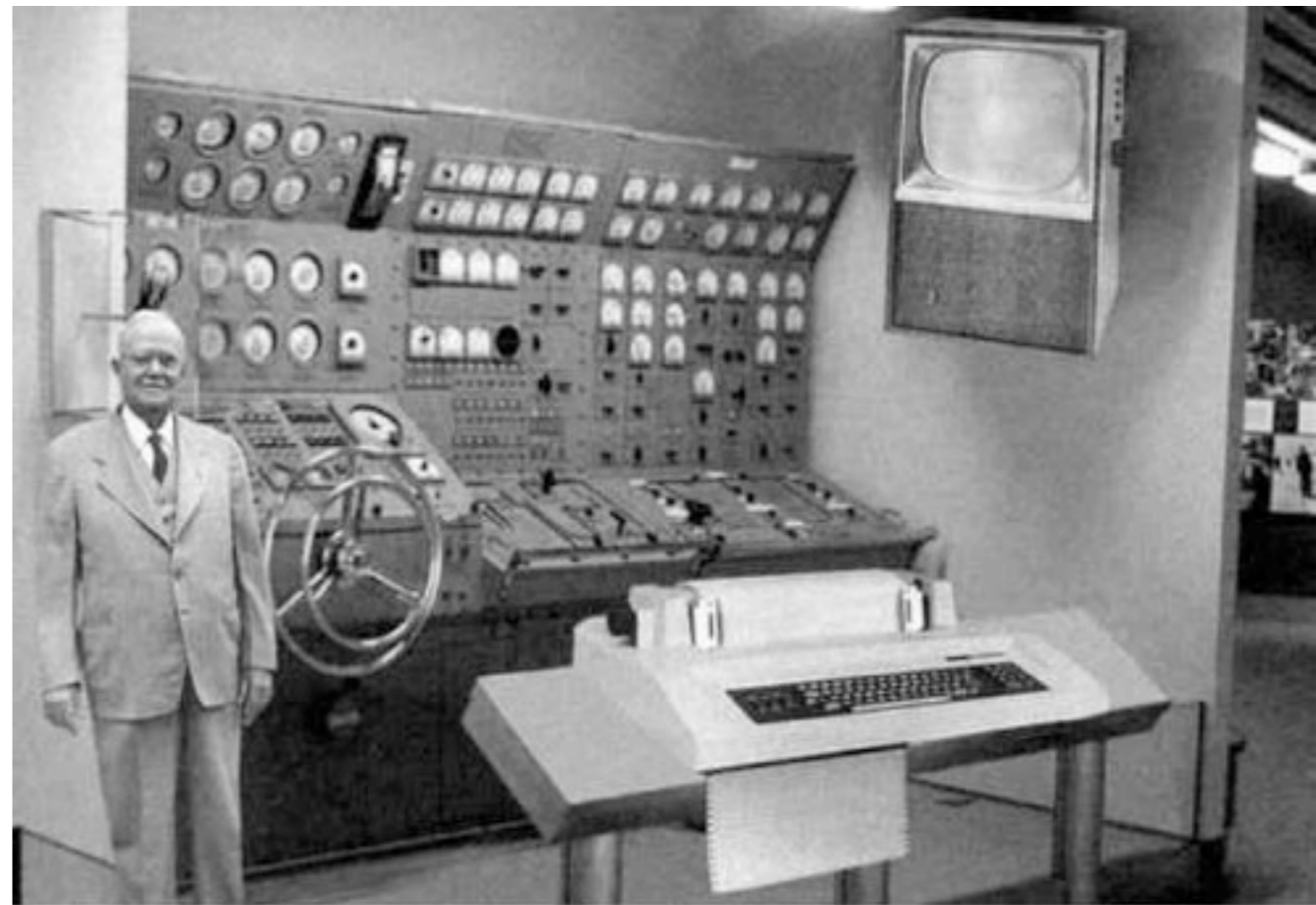
Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

Operating Systems (1960s)

- Multiprogramming
  - ‣ Timesharing
  - ‣ Multiple programs run concurrently

- Many operating systems concepts invented
  - ‣ Virtual memory, Hierarchical File Systems, Synchronization, Security and many more

- End up with slow, complex systems on limited hardware (Multics)

Operating Systems (1970s)

- Becoming more available

  ‣ UNIX

    - First OS written in a high-level language

- Becoming more flexible

  ‣ Extensible system

  ‣ Community forms beyond developers

- Performance focus

  ‣ Optimization of algorithms from 1960s

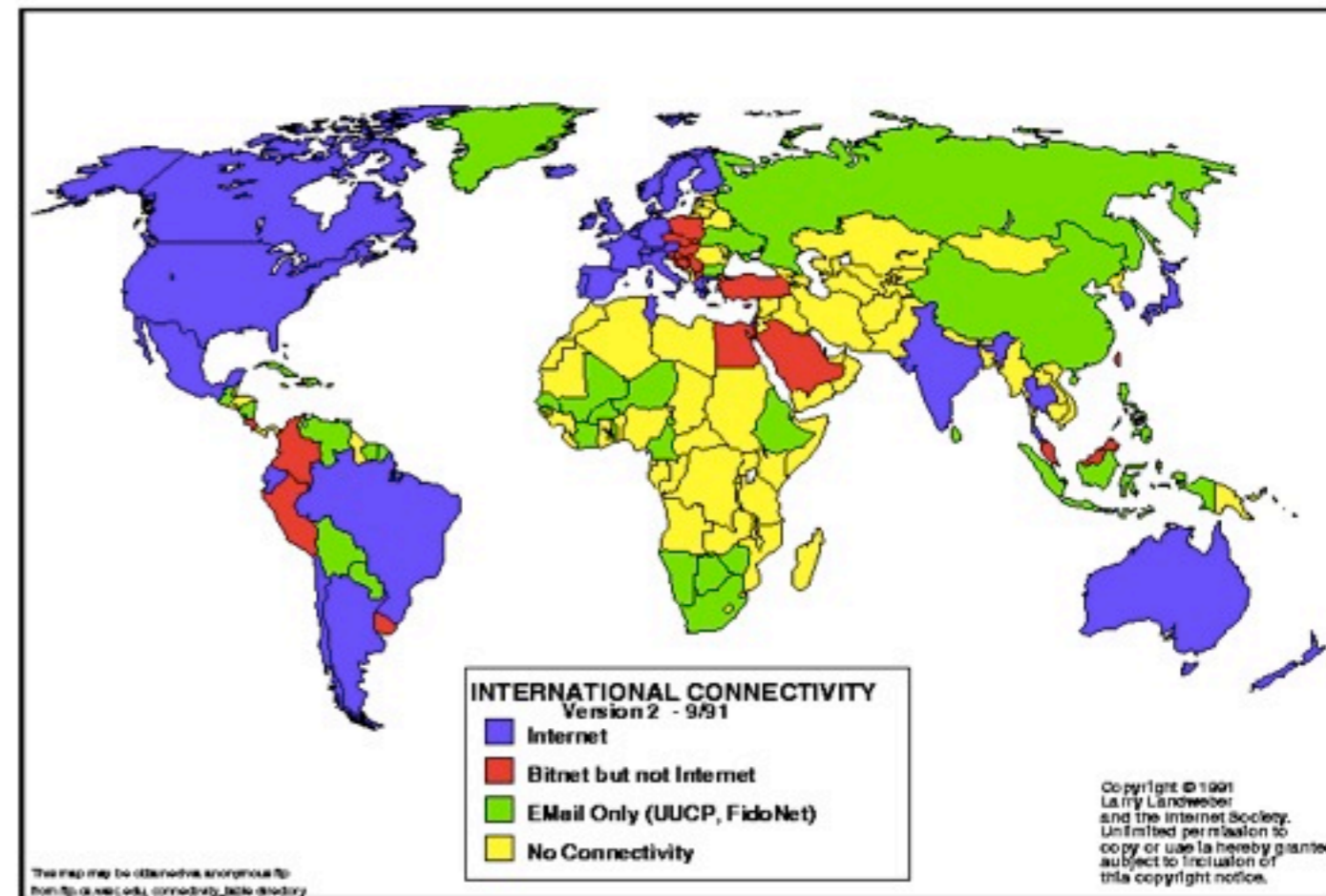- **Critical Mass Reached**

    ‣ A variety of well-known systems, concepts

    ‣ UNIX fragments

- **PC Emerges**

    ‣ Simple, single user, no network

    ‣ Simple OSes: DOS

- **Graphical User Interfaces**

    ‣ X Windows and Apple Macintosh

Operating Systems (1990s)

- Connect to Internet

  ‣ "Real OSes" for PCs

    - NT/2000+, Linux, eventually Mac OS X

- Server Systems Galore

  ‣ Mainframes even re-emerge

- Complex Systems and Requirements

  ‣ Parallel, Real-time, Distributed, etc.

INTERNATIONAL CONNECTIVITY
Version 2 - 9/91

- Internet
- Bitnet but not Internet
- EMail Only (UUCP, FidoNet)
- No Connectivity

Copyright ©1991
Larry Landweber
and the Internet Society.
Unlimited permission to
copy or use is hereby granted
subject to inclusion of
this copyright notice.

The map may be otherwise anonymous ftp
from ftp.cs.wisc.edu, connectivity_table directory

- Challenges facing us now include
  - ‣ Security
  - ‣ Multicore
  - ‣ Ubiquitous
  - ‣ Virtual Machines
  - ‣ Embedded
  - ‣ Mobile

The Holy Trinity
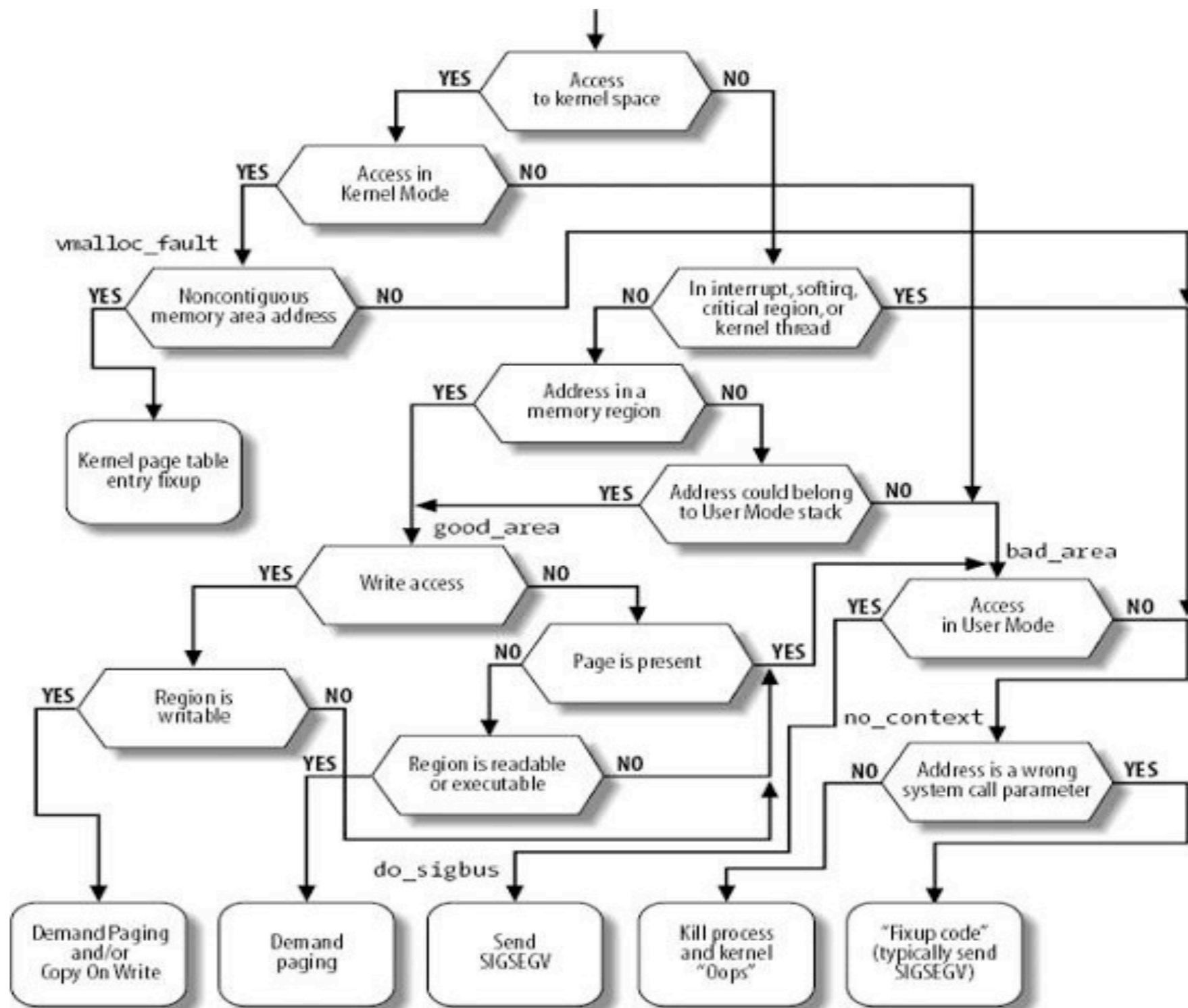
- Where are things heading?

  ▶ cloud computing

  ▶ ubiquitous mobile devices

    • ARM Cortex A8 = 2000 MIPS @ 1 GHz

    • Intel Penium 3 = 2,054 MIPS @ 600 MHz

  ▶ what operating systems are running in cloud and mobile systems?

- **What does it do?**

  ‣ Mostly behind the scenes…

- **Example**

  ‣ Page Fault Handling

Page Fault Handling

- Cause: Access a virtual memory location not backed by a physical page

- Trap generated by hardware

- Handler in OS determines how to obtain memory

- If page is still on disk, then handler
  - allocates physical page
  - makes I/O request to disk via file system and driver

- Driver copies page from disk into new physical page

- OS restarts the process at the trapped instruction

Page Fault Handling

- There are multiple processes, so the OS has to make trade-offs

    ‣ What is there are no physical pages available?

    ‣ The disk is slower than memory access, so how to process?

    ‣ There may be multiple outstanding disk requests, so what order should they be processed?

    ‣ How does the OS interact with hardware effectively?

    ‣ Many others…

Learning About OSes

- OS has many protocols like page fault handling

  ‣ You will need to know them

- OS designers add layers of indirection concepts to simplify programming (e.g., virtual memory)

  ‣ You will need to understand these concepts

- The design of protocols using these concepts involves trade-offs (e.g., optimize disk read performance)

  ‣ You will need to understand why OS protocols are written the way that they are
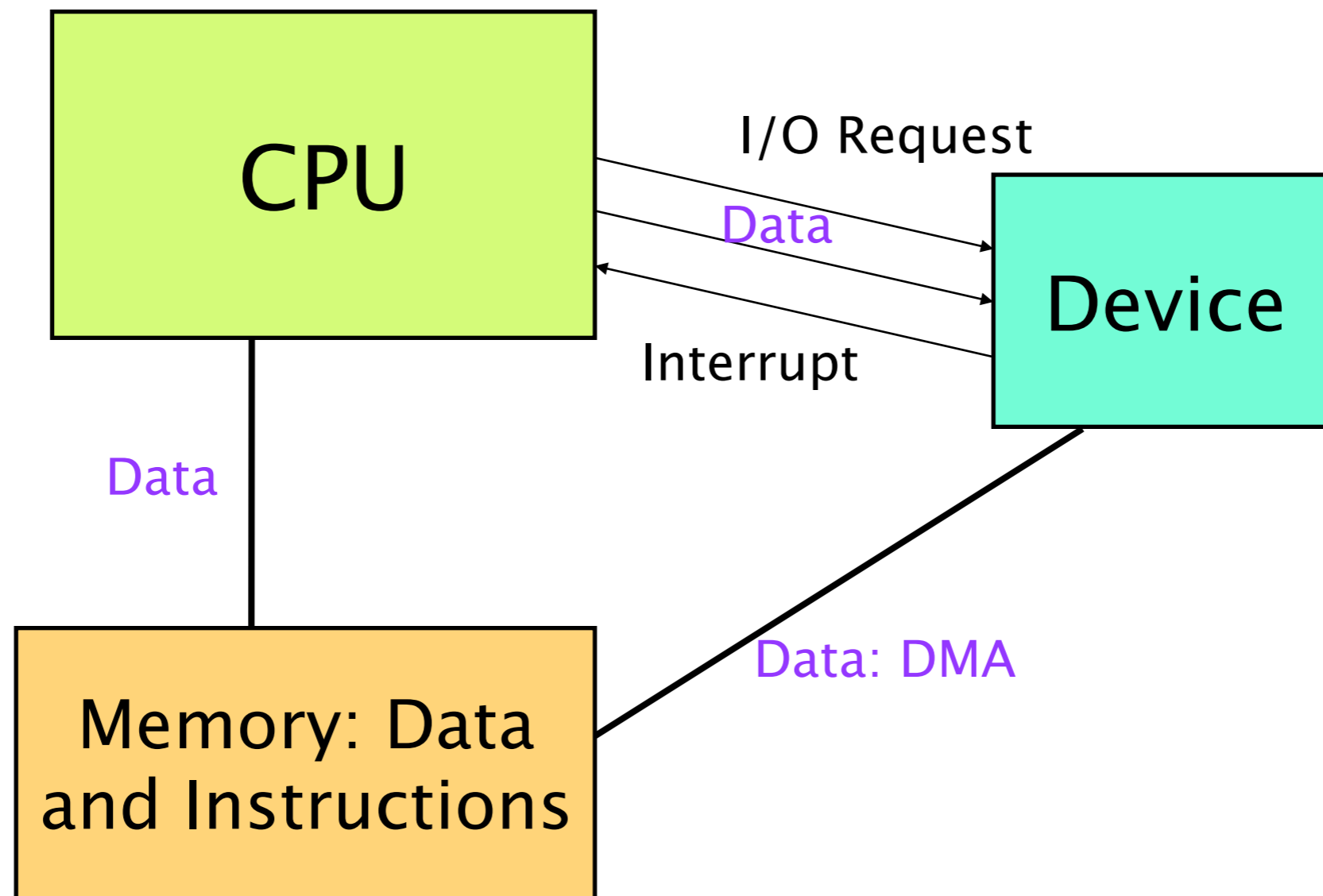
Some Basics

Storage Hierarchy

Scheduling
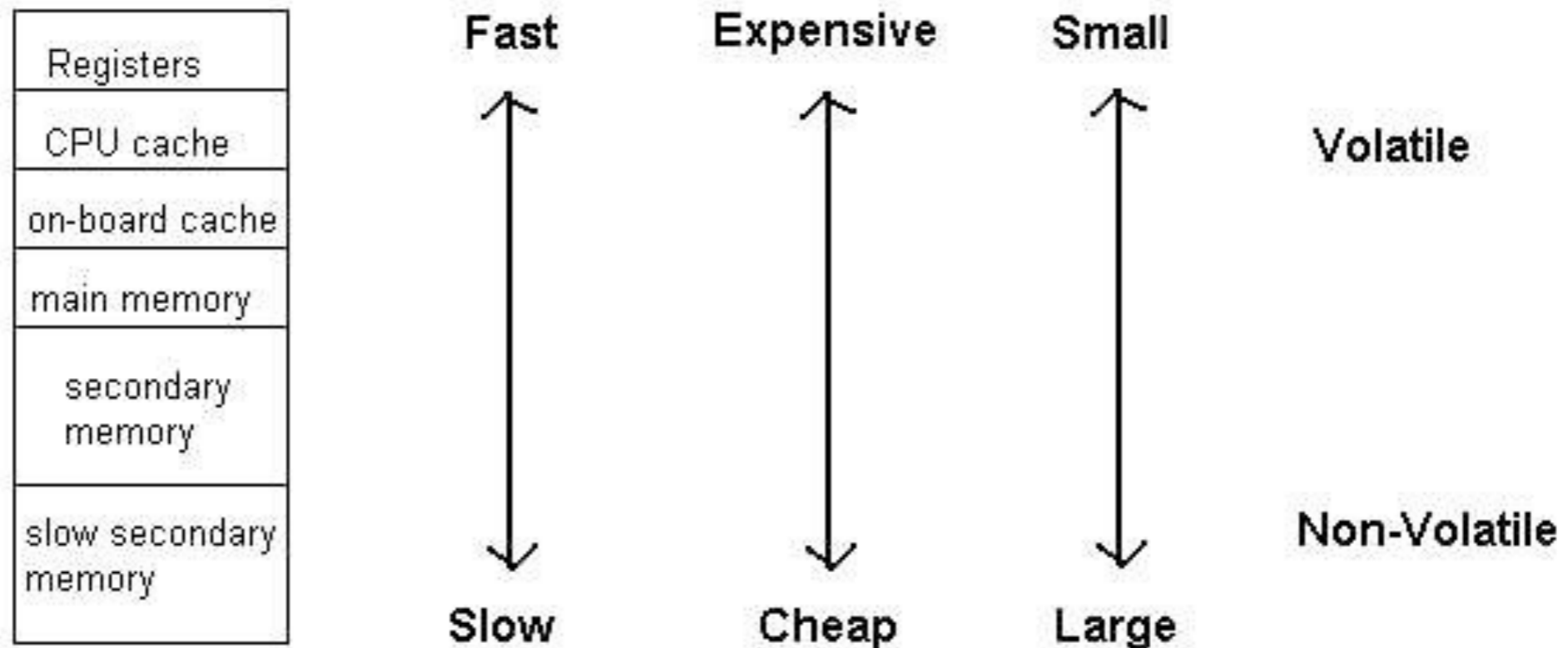
- Determine which task to perform given that there are:

  ‣ Multiple user processes

  ‣ Multiple hardware components

- Provide effective performance

  ‣ Responsive to users, CPU utilization

- Provide fairness

  ‣ Do not starve low priority processes

- Control access to shared resources

  ‣ E.g., Files

- Ensure that only authorized processes can access a file

  ‣ User's process can access user's files

  ‣ Most file systems enable sharing among users

  ‣ Some operating systems represent devices as files

Assignment 0 (due Thurs)

- Complete the following survey:

  ‣ http://www.surveymonkey.com/s/F5S8VF8

- Download the following virtual machine image:

  ‣ http://www.cs.uoregon.edu/Classes/12S/cis415/Ubuntu10-cis415.vmdk

  ‣ Only download on campus (it's about 3 GB)

  ‣ Preferred: keep the image on a portable hard drive

  ‣ username: cis415 password: 415s12

- Set up a subversion or git repository in home dir.

Next Time

- Next class
  - ▸ Background on Computer Systems