CS240: Programming in C

Lecture 1: Class overview.

History

- The world in 1969
  - Fortran, PL/1 and Cobol were the primary “high-level” programming languages available.
  - Assembly language the choice for high-performance applications
  - Operating systems
    - A golden age
    - Advent of paging, segmentation, virtual memory, and access-control file systems

Multics

- An advanced operating system developed at MIT, GE and Bell Labs.
- Precursor to many of the ideas found in Unix
  - process as a locus of control
  - tree-structured file system
  - advanced access control
  - generalized device abstraction
  - command interpreter
- Written almost entirely in PL/1 and BCPL

Target hardware environment

- Port to DEC PDP-7
  - 8K 18-bit memory
  - no useful software
  - port written in assembler
- First high-level language on Unix
  - TMG: a language for writing compilers
    - primitive support for context-free syntax
- First general-purpose high-level language
  - B: C without types
  - interpreted
  - ran in 8K bytes of memory

Circa 1970 - 1971

- Unix had developed a sizable user base
- B was too slow; assembler too painful
  - string manipulation painful on byte-oriented machines
  - no floating-point
  - pointers were not easily manipulable
    - a pointer was an index into an array of words
- Extend B with (richer) types
  - Further additions of record types and typed arrays
  - Unified type syntax

By 1973

- Preprocessor
- Integration with underlying system services
  - file system
- Macros
- Unix kernel rewritten for the PDP-11 in C
  - portable I/O package
- From 1973 to 1978 ...
  - more types
  - expanded libraries and environment
  - emphasis on portability

Reasons for its success...

- Success of Unix
- It's quite simple and surprisingly uniform
  - declarations mimic expressions
  - supports low-level reasoning
    - pointers and arrays as index into regions of memory, generalized casts ....
  - supports high-level reasoning
    - types and operators well-defined
    - static compilation
  - useful set of libraries to enable portability and interaction with a real environment
- Extraordinarily stable
  - no major change since 1983

What's not to like?

- Programming model close to the machine
  - no abstraction
    - no objects, modules, datatypes, ...
  - why is abstraction important?
- No safety
  - "Well-typed programs don't go wrong" doesn't apply here.
  - Easy to generate security vulnerabilities
    - buffer overflows, unwanted overwriting of memory
- No automatic memory management
- Is programming mathematics or engineering?

Why learn C, on balance

- **Power/performance**: low-level control over the OS.
- **Understanding:** understand better the interaction between machine and software:
  - C as the expression of a low-level virtual machine
- **Betterment:** gain a better appreciation of high-level programming languages
  - C is not a panacea
  - “With power comes responsibility”

Requirements

- **Learning how to program is achieved by DOING the PROJECTS**
- **Mandatory class and lab attendence**
  - YOU HAVE TO SUBMIT ALL PROJECTS
  - You will not get a grade unless you submit all projects (with at least an attempt to solve it)

Course information

- Meetings
  - MW:3:30 - 4:20pm, SMTH 108, Jan 10 - Apr. 30
- Professor contact info:
  - Office: LWSN 3141J
  - Email: suresh@cs.purdue.edu
  - Office hours: MW 2:30 - 3:30 or by appointment
- TA:
  - TBA
- Class webpage

  http://www.cs.purdue.edu/homes/suresh/cs240-spring2011

Class attendance and taking notes

- **REQUIRED !!!**
- If you need to miss class, let me know
- Slides will be made available online before lecture but YOU ARE STRONGLY RECOMMENDED TO TAKE NOTES

Grading policy

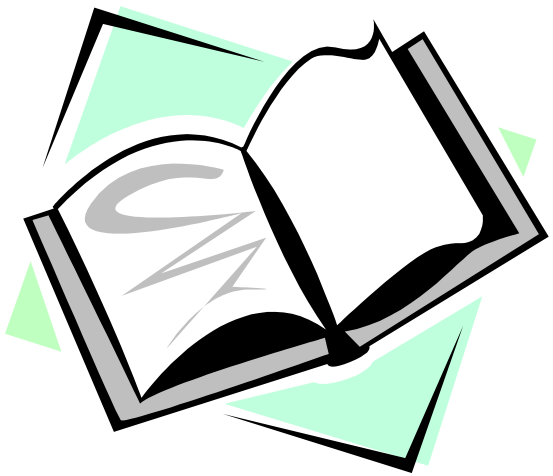
- Programming
  - Labs 15%
  - Projects 30%
- Exams
  - Midterm 1 15%
  - Midterm 2 15%
  - Final 25%

Reference material

- The C Programming Language, Brian W. Kerninghan and Dennis M. Ritchie, 2nd Edition
- Lecture slides posted online

Readings for this lecture

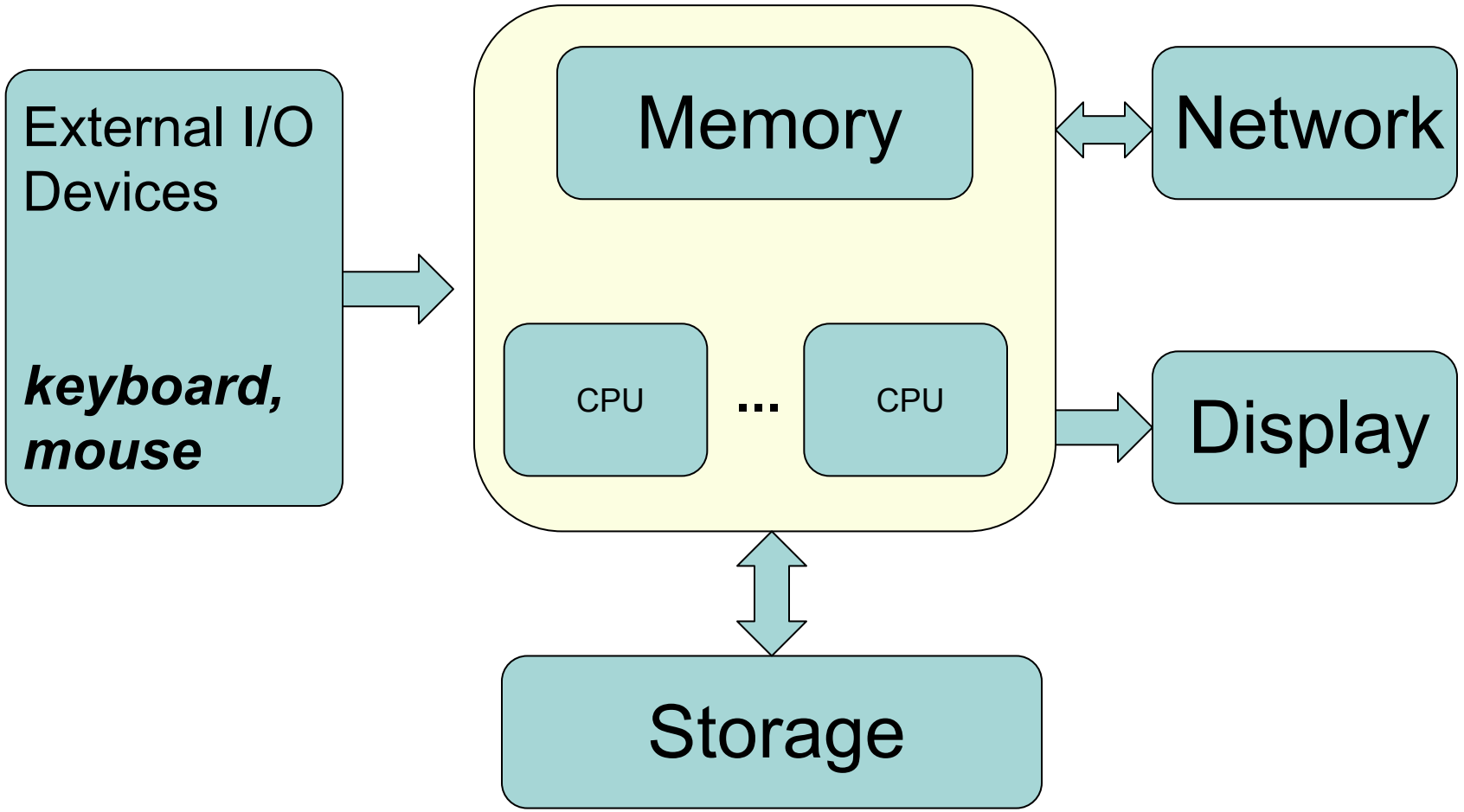
Chapter 1: Introduction



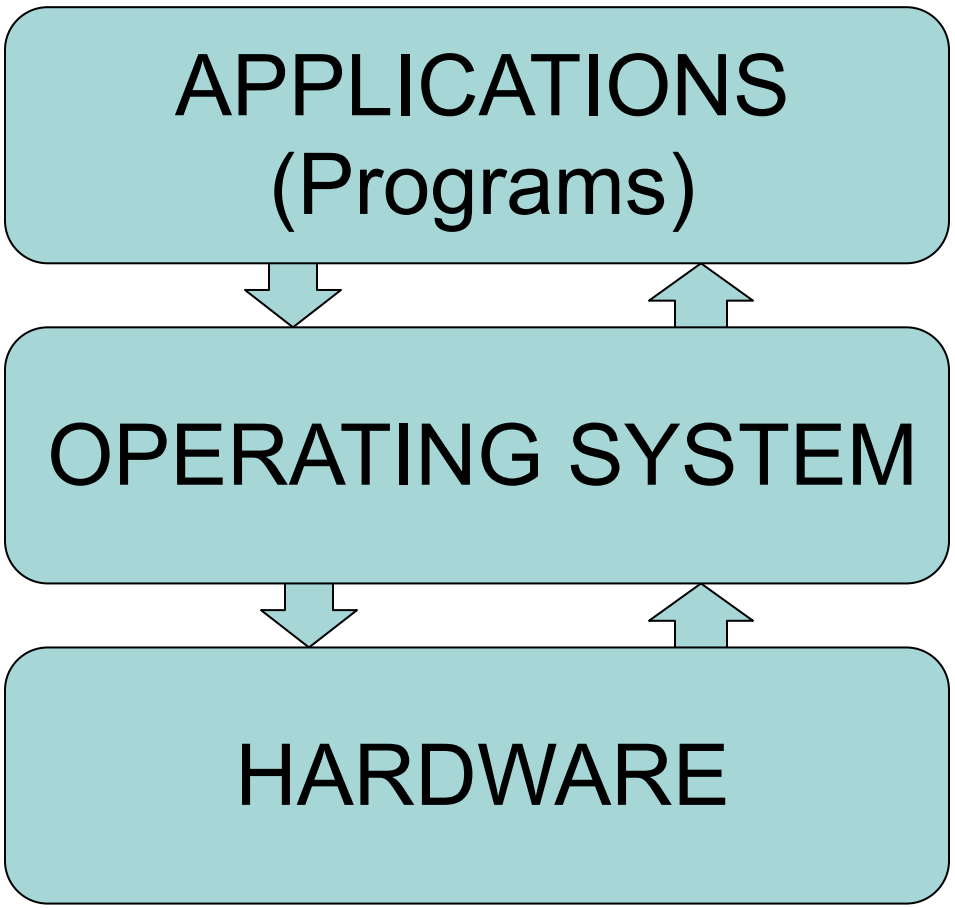
Terminology

- What's a computer?
- What is hardware/software
- What's an algorithm ?
- What's a program?
- What's a library?
- What's an operating system?
- What's a programming language ?
  - Machine language
  - Assembly language
  - High-level language

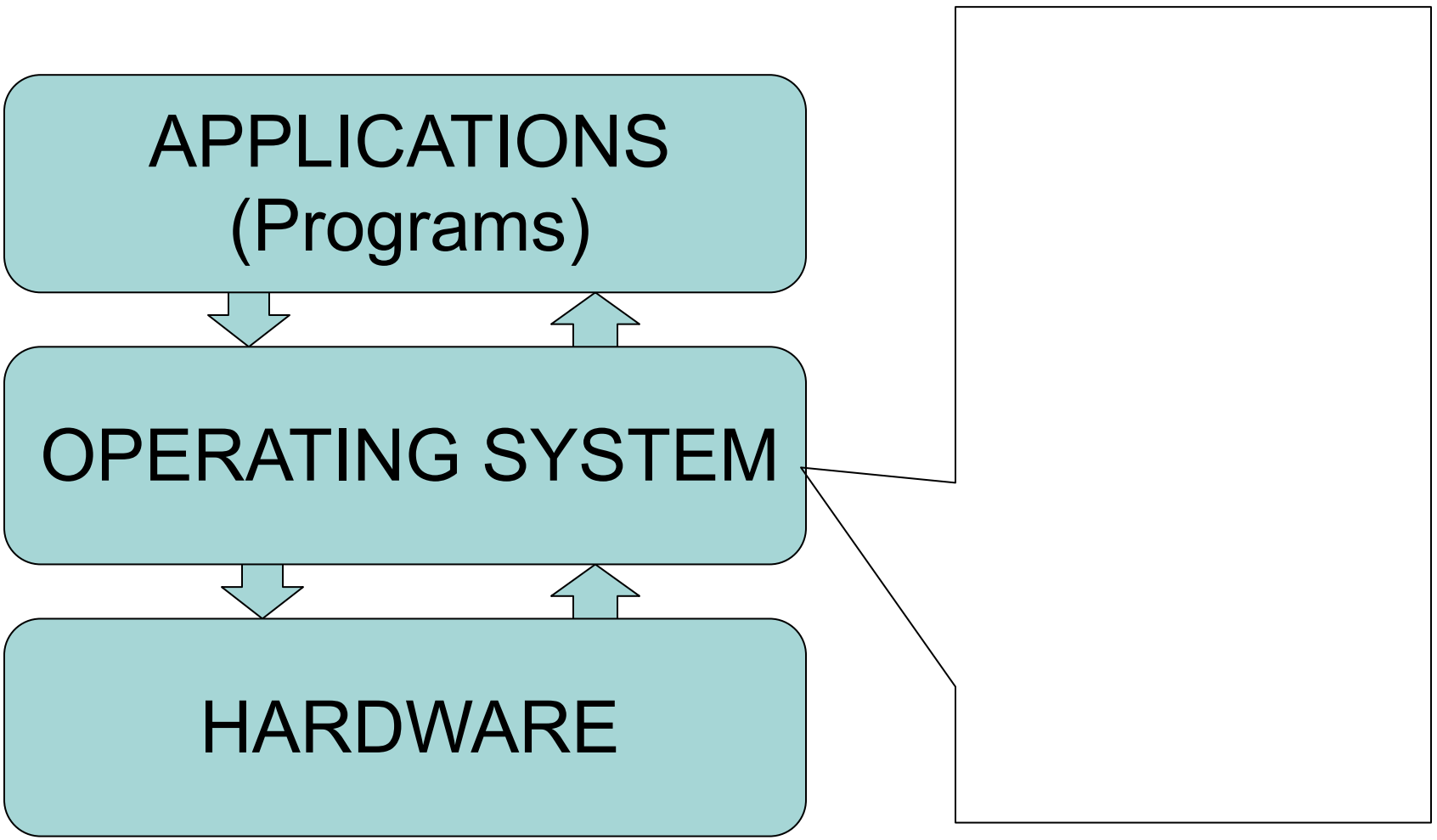
Computer architecture



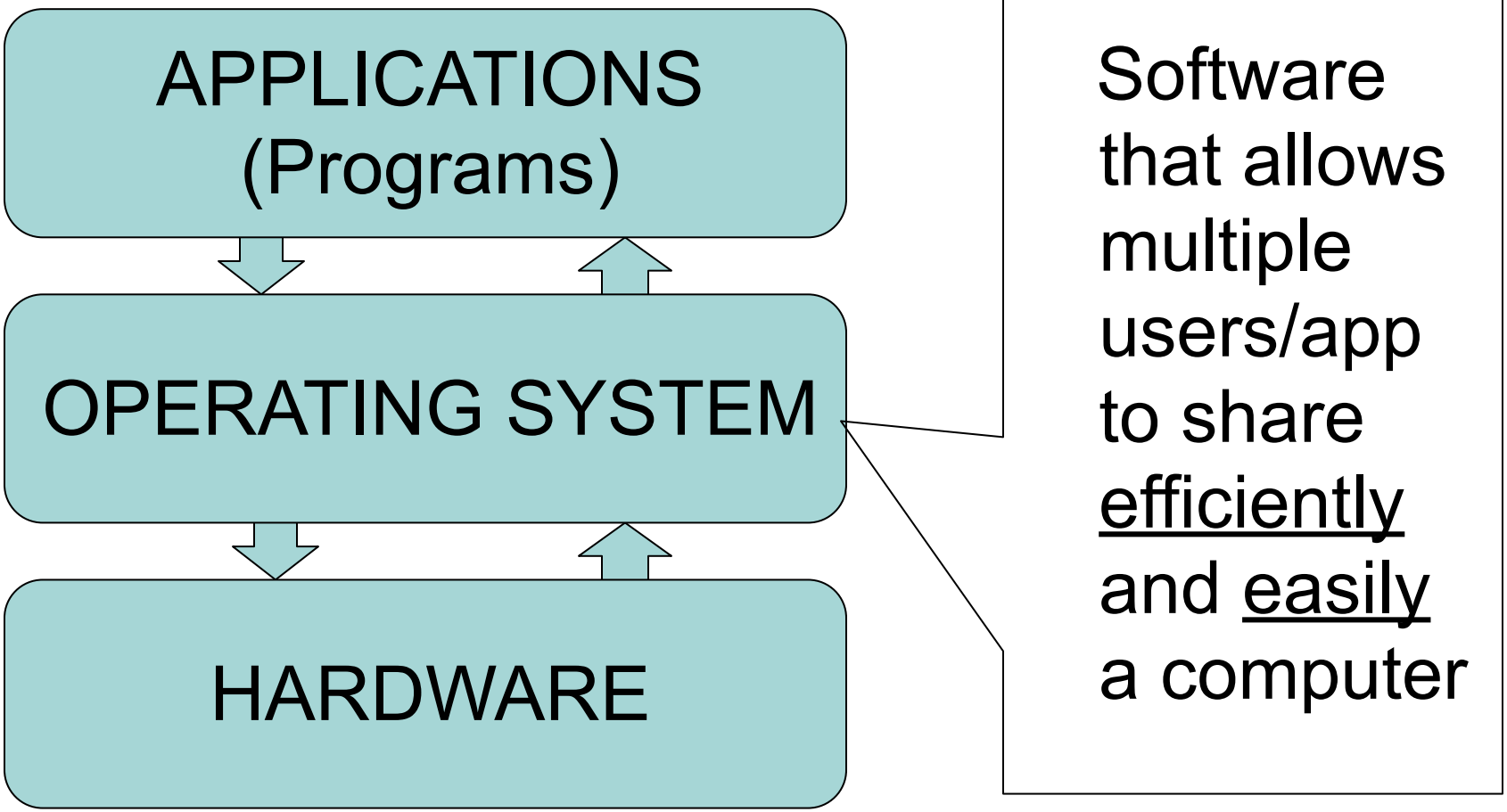
System Layers



System Layers



System Layers



Operating System

- Management of the processes and their access to resources
  - Memory
  - CPU access
  - I/O
  - Network
  - Other devices
- Interaction with the user
  - Graphic interface
  - Other devices

Algorithm/Program

- **Algorithm**: procedure for solving a problem in finite steps
- **Program**: set of instructions to the CPU, stored in memory, read and executed by the CPU
  - Differentiate between the abstract (*algorithm*) and concrete (*program source*) description of a computation
  - Differentiate between the concrete description of a computation and its execution (*translated machine code*) on hardware

Machine and assembly language

- **Machine language** : binary information, specific to a CPU
  - How a CPU interprets data: e.g. how are memory addresses represented, how is an instruction coded, etc
  - This is the **binary or executable code**
- **Assembly language**: easier to write for people, using symbols, requires an assembler
  - Still need to think in terms of low level CPU steps
  - Still hardware-specific

High-level language

- Closer to human language
  - *How much closer?*
- Needs a compiler to convert it to machine language
  - Demands precision in its definition
- One can write programs in many high-level languages for the same CPU
  - What does this imply about the kinds of computation a HLL can (or cannot) perform?
- More portable (more details later)