CSE 333

Lecture 1 - Systems programming

Hal Perkins

Department of Computer Science & Engineering

University of Washington

Welcome!

Today's goals:

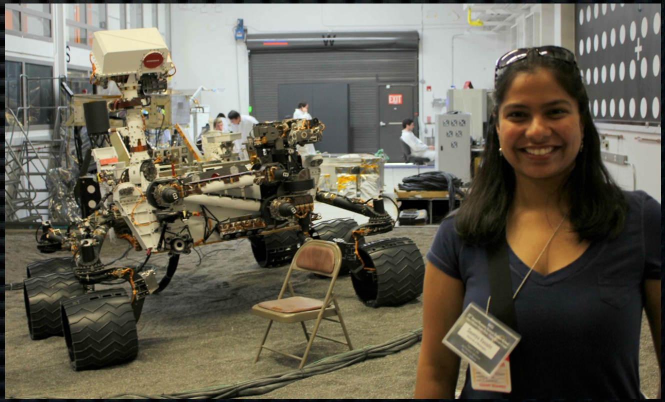
- **introductions**
- *big picture*
- *course syllabus*
- *setting some expectations*

Us

Hal Perkins

Cortney Corbin

Soumya Vasisht

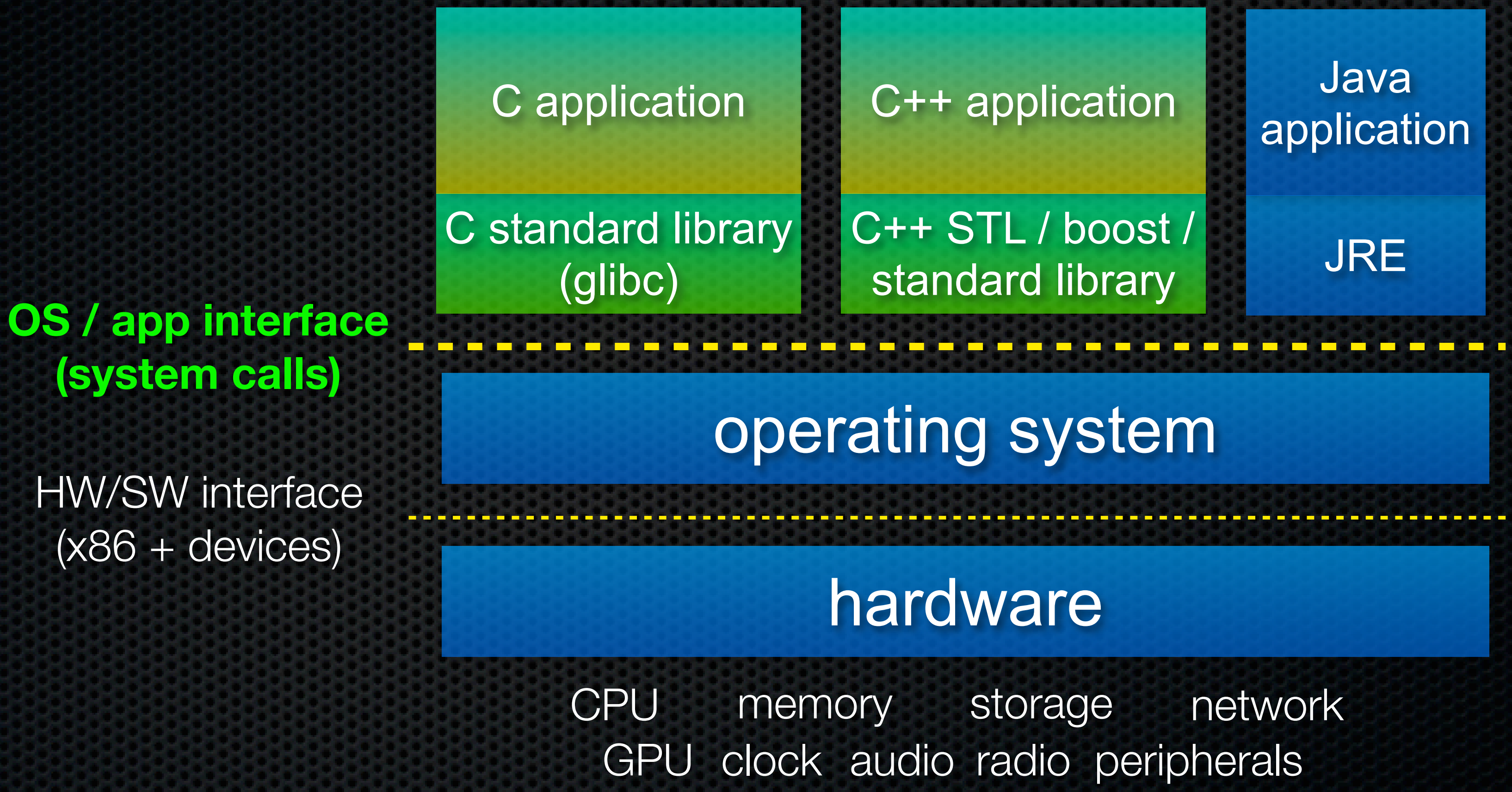


Welcome!

Today’s goals:

- *introductions*
- **big picture**
- *course syllabus*
- *setting some expectations*

Course map: 100,000 foot view



C application

C standard library (glibc)

C++ application

C++ STL / boost / standard library

Java application

JRE

OS / app interface (system calls)

operating system

HW/SW interface (x86 + devices)

hardware

CPU memory storage network

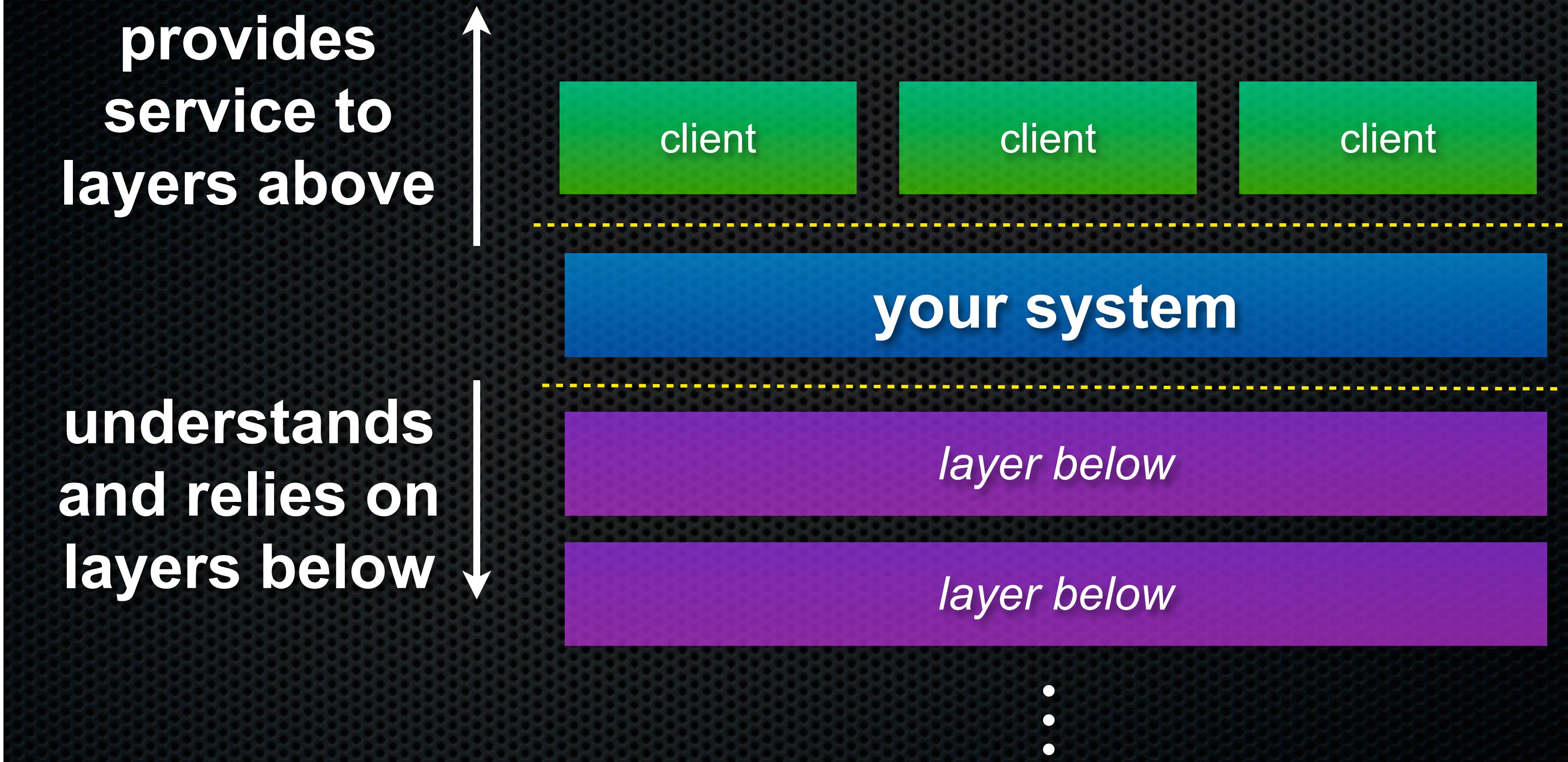
GPU clock audio radio peripherals

Software "System"

A platform, application, or other structure that:

- is composed of multiple modules
  - the system's **architecture** defines the *interfaces of* and *relationships between* the modules
- often is complex
  - in terms of its implementation, performance, management
- hopefully has requirements
  - performance, security, fault tolerance, data consistency

A layered view



provides service to layers above

client

client

client

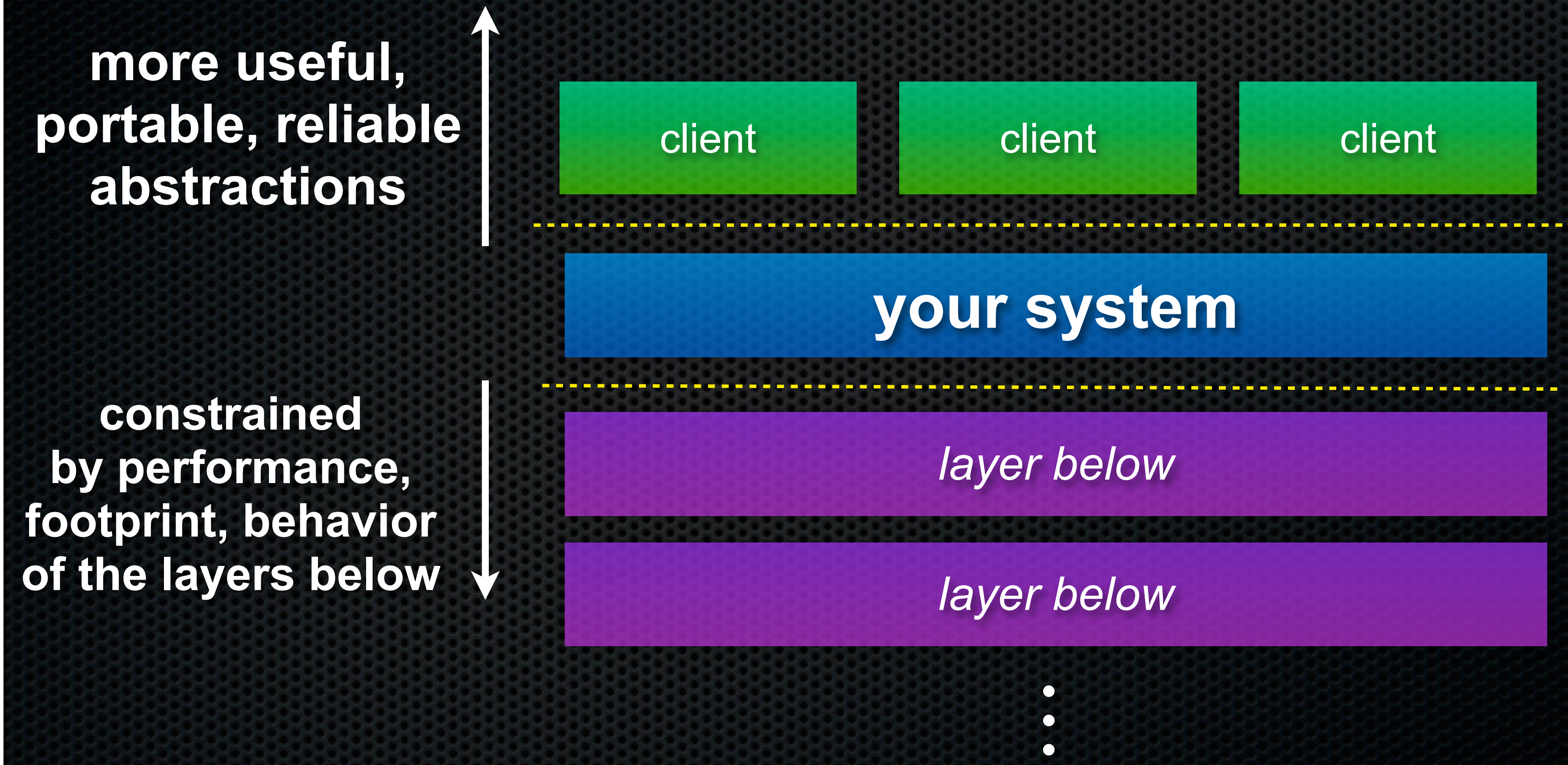
your system

understands and relies on layers below

layer below

layer below

A layered view



more useful, portable, reliable abstractions

client

client

client

your system

layer below

layer below

constrained by performance, footprint, behavior of the layers below

Example system

Operating system

- a software layer that abstracts away the messy details of hardware into a useful, portable, powerful interface
- modules:
  - file system, virtual memory system, network stack, protection system, scheduling subsystem, ...
  - each of these is a major system of its own!
- design and implementation has tons of engineering tradeoffs
  - e.g., speed vs. (portability, maintainability, simplicity)

Another example system

Web server framework

- a software layer that abstracts away the messy details of OSs, HTTP protocols, and storage systems to simplify building powerful, scalable Web services
- modules:
    - HTTP server, HTML template system, database storage, user authentication system, ...
- also has many, many tradeoffs
    - programmer convenience vs. performance
    - simplicity vs. extensibility

Systems programming

The programming skills, engineering discipline, and knowledge you need to build a system

- ***programming***: C / C++
- ***discipline***: testing, debugging, performance analysis
- ***knowledge***: long list of interesting topics
  - concurrency, OS interfaces and semantics, techniques for consistent data management, algorithms, distributed systems, ...
  - most important: deep understanding of the "layer below"
    - *quiz: how many copies of your data are made when you use the read( ) system call to read from a file?*

Programming languages

Assembly language / machine code

- *(approximately)* directly executed by hardware
- tied to a specific machine architecture, not portable
- no notion of structure, few programmer conveniences
- possible to write really, really fast code
- necessary for a few critical parts of the operating system
- extraordinarily painful and fragile

Programming languages

Structured but low-level languages (C, C++)

- hides some architectural details, is mostly portable, has a few useful abstractions like types, arrays, procedures, objects
- permits (forces?) programmer to handle low-level details like memory management, locks, threads
- low-level enough to be **fast** and to give the programmer **control** over resources
    - double-edged sword: low-level enough to be complex, error-prone
    - a useful shield: engineering discipline

Programming languages

High-level languages (Python, Ruby, JavaScript, ...)

- focus on productivity and usability over performance
- powerful abstractions shield you from low-level gritty details (bounded arrays, garbage collection, rich libraries, ...)
- usually interpreted, translated, or compiled via an intermediate representation
- slower (by 1.2x-10x), less control

Discipline

Cultivate good habits, encourage clean code

- coding style conventions
- unit testing, code coverage testing, regression testing
- documentation (code comments, design docs)
- code reviews

Will take you a lifetime to learn

- but oh-so-important, especially for systems code
    - avoid write-once, read-never code

Knowledge

Tools

- gcc, gdb, g++, objdump, nm, gcov/lcov, valgrind, IDEs, race detectors, model checkers, ...

Lower-level systems

- UNIX system call API, relational databases, map/reduce, Django, jQuery, ...

Systems foundations

- transactions, two-phase commit, consensus, RPC, virtualization, cache coherence, applied crypto, ...

Welcome!

Today's goals:

- *introductions*
- *big picture*
- **course syllabus**
- *setting some expectations*

C / C++ programming

Major focus of this course

- ~2 weeks of diving deeper into C
  - review some material from 351 and go deeper
- ~4 weeks of a (sane subset) of C++
- exposure to programming tools
  - unit testing frameworks, performance profiling and analysis, revision control systems

Interacting with UNIX and standard libraries

The “layers below” we will be relying on

- learn C’s standard library and some of C++’s STL
    - including memory management (malloc/new, free/delete)
    - we’ll look at some of C++11 and boost
- learn aspects of the UNIX system call API
    - I/O: storage, networking
    - process management, signals

Potential additional topics

Concurrency

- threads
- perhaps asynchronous I/O and event-driven programming

Security

- will be mindful of security topics as they come up
- e.g., how to avoid buffer overflow issues in C/C++

Welcome!

Today's goals:

- *introductions*
- *big picture*
- *course syllabus*
- **setting some expectations**

What you will be doing

- Attending lectures and sections
  - lecture: ~25 of them, MWF here
  - sections: ~9 of then, Thu 9:40, also here
  - Take notes!!! Don’t expect everything to be on the web
- Doing programming projects
  - ~4 of them, successively building on each other
  - includes C, C++; files, networking
- Doing programming exercises
  - one per lecture, due before the next lecture begins
  - coarse-grained grading (0, 1, 2 or 3)

Project Deadlines

Goal: keep everyone on schedule, consistent at 11 pm, allow a little slack - but how?

Option 1:

- 10% deduction each day late, max 2 days per assignment
- 2 free “late days” during the quarter

Option 2:

- No late penalties, assignments due at deadline
- 4 free “late days” for the quarter, no more than 2 per assignment

Which should we use? Alternatives?

No late days for exercises in any case - due 9 am before lecture

Requirements

CSE351 is a prerequisite

- I assume you have just a little exposure to C
- I assume you know what a linked list, tree, hash table is

You need access to a CSE linux environment

- undergraduate labs, ssh into `attu.cs`, use CSE home VMs

Textbooks

Recommended (strongly):

- Computer Systems, A Programmer’s Perspective (“**CSAAP**”)
    - [2nd Ed]. CSE351 textbook; do you already have it?
- C: A Reference Manual (“**CARM**”) [5th Ed]
- C++ Primer (“**C++P**”) [5th Ed]

Optional (but cool):

- Effective C++ [3rd Ed]

We’ll also take advantage of online resources...

Communications

Discussion board (gopost)

- Best for most things - everyone benefits from exchanges
- Hint: post something, then unread postings will be tagged

cse333-staff@cs

- Use for communications that don’t belong on the board

Perspective: Real software, real APIs, real bugs

- We’ll try to help as best we can, but mostly expect ideas for what to try, where to look. Debugging is your job.

Collaboration

Some of the projects will be individual, some in teams

- assume individual unless explicitly stated otherwise

Cross-team collaboration is useful and expected

- help other teams with programming fundamentals, concepts

Plagiarism and cheating is verboten

- helping other teams with assignments, debugging their code
- relying on help without attributing in your writeups

For Wednesday

Homework #0 is due (a short survey):

- https://catalyst.uw.edu/webq/survey/perkins/171716

Exercise 0 is due

- http://www.cs.washington.edu/education/courses/cse333/12su/exercises/ex0.html

See you on Wednesday!