



# COMP 150

## Implementation of Modern Programming Languages


---

Lecture 1  
Introduction


January 18, 2007

## The Perfect Language




- What features would the perfect programming language have?
  - Good for everything, from device drivers to web-based apps
  - Easy to write complex programs
  - Safe, automatically verified
  - Performs well on a variety of hardware




Tufts University Computer Science

2

## Challenges




- Is it possible to have one perfect language for all programming tasks?
- Is “*easy to use*” compatible with “*automatically verified*” and “*high performance*”?
- Does good performance require the programmer to know something about the hardware?




Tufts University Computer Science

3

## Programming trends




- Software complexity
  - Many layers of abstraction
  - Modularity, reuse, maintainability
- Software reliability
  - Errors and security holes
  - Too difficult to check manually
- Performance
  - Underlying hardware complex, diverse
  - Encapsulation inhibits many optimizations




Tufts University Computer Science

4

## Hardware trends




- Complex ISAs
- Multi-core processors
- Deep memory hierarchies
- Non-uniform cost execution
- Other concerns (e.g., power)




Tufts University Computer Science

5

## Big picture



- Languages
  - Becoming more abstract, moving away from the hardware
- Hardware
  - Becoming more complex to program, exposing more low-level details
- Result
  - Bigger gap between languages and hardware
  - Significant burden on **compilers**
  - Good news: more computing power



Tufts University Computer Science

6

## What is compilation?

- Last semester:
  - Write a program in some language
  - Translate into machine code
  - Link and execute
- How is “modern” compilation different?



## Languages

- Diverse languages
  - Fortran
  - C, C++
  - Prolog
  - ML
  - Java, C#
  - Ruby, Python, Perl
- What are the differences between these?



## Expressiveness

- Abstractions
  - What are the first-class entities?
- Data structures
  - Generics, reusables, libraries
- Ease of use
  - Quick prototyping vs long-term maintainability (both?)



## Correctness

- Typing
  - Strongly typed or weakly typed?
- Automatic checking
  - Compile-time checking?
  - Run-time checking?
- Language support for safety
- Dealing with legacy code



## Correctness

- Concurrency
  - First-class entity in some languages
  - What are the primitives?
    - Fork/join
    - Locking
    - Transactions
    - Others?
  - Problems:
    - Race conditions, atomicity constraints



## Execution environment

- Compiled vs interpreted
  - And things in between
- Virtualized vs directly on hardware
  - What is a “binary”?
- Are programs static entities?
  - Dynamic libraries, dynamic class loading
- Rich runtime vs all under programmer control
  - Costs explicit vs hidden



## Trends in system design

- Multiple levels of translation
  - Programming language
  - Virtual machine
  - Virtual OS
  - ABI/ISA
  - Micro-ops
- Rich run-time support
  - Automatic memory management
  - Run-time safety checks
  - Dynamic system configuration (e.g., class loading)



## Course topics

- Current, new programming languages
- Virtual machines
- Just-in-time compilation
- Dynamic optimization
- Binary translation, binary instrumentation
- Compiling for...
  - ...security and correctness
  - ...memory performance
  - ...scientific computing



## Class

- Seminar class
  - Read and discuss papers
  - Programming projects – mini research projects
- Discussion leader
  - 20 minute presentation of paper
  - Focus on issues and discussion questions
- Paper reviews
  - When you're not the leader
  - Write a short critique



## Goals

- Learn about latest topics in compilers and programming language implementation
  - Algorithms and system designs
  - Tinker with real systems
- Learn about how PL research is done
  - Kinds of research approaches
  - Evaluation and methodology
  - Paper writing process
  - Conference submissions and reviews



## Semester projects

- Kinds of projects
  - Experiment with a cutting-edge PL tool or VM
  - Add to an existing system
    - A new compiler algorithm
    - A new run-time component
  - Perform a study or survey
- Teams are encouraged



## Projects

- Stages:
  - Project proposal
  - Mid-semester status report
  - Presentation of results
  - Final report
- Systems
  - Java VMs: JikesRVM or HotSpot
  - Instrumentation: PIN, valgrind
  - Dynamic optimization: Dynamo



## Grading

- Class participation – 30%
- Paper critiques – 30%
- Projects – 40%
  - Proposal – 10%
  - Mid-term status – 5%
  - Presentation – 10%
  - Report – 15%



## Next time...

- Set up class mailing list  
(Possibly a class wiki as well)
- Assign discussion leaders for upcoming weeks
- Talk about project ideas

