15-411/15-611 Compiler Design

Jan Hoffmann — Fall 2018

http://www.cs.cmu.edu/~janh/courses/411/18

Course Staff

- **Instructor:** Jan Hoffmann

  Office hours: Tue 10:30am-noon
  Thu 1:00pm-2:00pm at GHC 9105

- **Research**
  - Programming languages
  - Verification (quantitative properties like resource usage)

- **Teaching**
  - 15-411/611 Compiler Design
  - 15-312 Principles of Programming Languages

Nick Roberts
CS Senior

(OCaml, Haskell)

Prachi Laud
CS Senior

(OCaml)

Shalom Yiblet
CS Senior

(Haskell)

Vijay Ramamurthy
CS Senior

(OCaml)

Teaching Assistants

Communication and Resources

- **Lecture:** Tue/Thu 9:00-10:20am at GHC 4215
- **Recitation** A: Fri 02:30 - 03:20pm, BH 235A
  B: Fri 02:30 - 03:20pm, BH 235B

Do you have a conflict?

- **Website:** http://www.cs.cmu.edu/~janh/courses/411/18
- **Piazza:** Enroll from website
- **Lecture notes:** Will be available after the lecture
- **Textbook:** Andrew Appel - Modern Compiler Implementation in ML

Compilers

- **A compiler translates a programming language (source language) into executable code (target language)**

- **Quality measures for a compiler**
  - Correctness (Does the compiled code work as intended?)
  - Code quality (Does the compiled code run fast?)
  - Efficiency of compilation (Is compilation fast?)
  - Usability (Does the compiler produce useful errors and warnings?)

Compilers

- **Compiler History**
  - 1943: Plankalkül, first high-level language (Konrad Zuse)
  - 1951: Formules, first self-hosting compiler
  - 1952: A-0, term ‘compiler’ (Grace Hopper)
  - 1957: FORTRAN, first commercial compiler (John Backus; 18 PY)
  - 1962: Lisp, self-hosting compiler and GC (Tim Hart and Mike Levin)

- **Compilers today**
  - Modern compilers are complex (gcc has 7.5M LOC)
  - There is still a lot of compiler research (LLVM, verified compilation, …)
  - There is still a lot of compiler development in industry (guest lecture?)

What will you learn?

Compiler Design

- **How to structure compilers**
- **Applied algorithms and data structures**
  - Context-free grammars and parsing
  - Static single assignment form
  - Data flow analysis and type checking
  - Chordal graph coloring and register allocation
- **Focus on sequential imperative programming language**
  Not functional, parallel, distributed, object-oriented, ...
- **Focus on code generation and optimization**
  Not error messages, type inference, runtime system, ...

Focus of the Course

- **Correctness (Does the compiled code work as intended?)**
- **Code quality (Does the compiled code run fast?)**
- Efficiency of compilation (Is compilation fast?)
- Usability (Does the compiler produce useful errors and warnings?)

Software Engineering

We won't discuss this much in lecture.

- **Implementing a compiler is a substantial software project**
  - Building, organizing, testing, debugging, specifying, …
- **Understanding and implementing high-level specifications**
- **Satisfying performance constraints**
- **Make (and reevaluate) design decision**
  - Implementation language and libraries
  - Data structures and algorithms
  - Modules and interfaces
- **Revise and modify your code**

Compilers are perfect to practice software engineering.

Learning Goals I

- Distinguish the main phases of a state-of-the-art compiler
- Understand static and dynamic semantics of an imperative language
- Develop parsers and lexers using parser generators and combinators
- Perform semantic analysis

`

- Translate abstract syntax trees to intermediate representations and static single assignment form
- Analyze the dataflow in an imperative language
- Perform standard compiler optimizations

Learning Goals II

- Allocate registers using a graph-coloring algorithm
- Generate efficient assembly code for a modern architecture
- Allocate registers using a graph-coloring algorithm
- Understand opportunities and limitations of compiler optimizations

`

- Appreciate design tradeoffs how representation affects optimizations
- Automatically manage memory using garbage collection
- Develop complex software following high-level specifications

How will this work?

Your Responsibilities

- **Attend lectures**
  - Lecture notes are only supplementary material

There will be coffee.

No exams.

- **6 Labs: you will impl. compilers for subsets of C0 to x86-64 assembly**
  - Lab1-4: each worth 100 points (total 400 points)
  - Code review after Lab 3: 50 points
  - Project proposal for a Lab 6 project: 50 points
  - Lab 5-6: each 150 points (total 300 points)

With a partner or individual.

- **4 Assignments: you will complete five written assignments that help you understand the material presented in the lectures**
  - Assignments 1-4: each 50 points (total 200 points)

Individual.

Labs — Overview

- **Labs (700 points)**
  - Lab 1: tests and compiler for L1 (straight-line code)
  - Lab 2: tests and compiler for L2 (conditionals and loops)
  - Lab 3: tests and compiler for L3 (functions)
  - Lab 4: tests and compiler for L4 (memory)
  - Lab 5: compiler and paper (optimizations)
  - Lab 6: code and paper (you choose)

Auto graded.

TA graded.

- **Code review (50 points)**
  - You show your code for Lab 3 and get feedback
  - We expect that every team member is familiar with all components
  - We expect that every team member wrote about half of the code

Support for 411/611 Comes From …

Jane Street

Helps to

- Improve the grading infrastructure
- Pay for AWS cost
- Coffee at lectures

Source Language: C0

Subset of C

- Small
- Safe
- Fully specified
- Rich enough to be representative and interesting
- Small enough to manage in a semester

Target Language

x86-64 architecture

- Widely used
- Quirky, but you can choose the instructions you use
- Low level enough you can get a taste of the hardware

Runtime system

- C0 uses the ABI (Application Binary Interface) for C
- Strict adherence (internally, and for library functions)

Finding a partner for the labs

I strongly suggest to work in teams of two.

Labs — Finding a Partner

Don’t panic.

There are two options

1. **You fill out a questionnaire and we *suggest* a partner (staff selection)**
   - Suggestion is not binding but it’s expected that you team up

2. **You team up with somebody yourself (self selection)**
   - Like in previous iterations of the course

Register your team on of before
Tuesday 9/4.

Option 1: Staff Selection

- **You fill out a questionnaire about**
  - Your plans and goals for the class
  - Your strengths and work style
  - And your time constraints
- **We suggest a partner with complem. strength and similar plans/goals**
- **You meet with your partner and (hopefully) decide to team up**
- **Advantages:**
  - You will get a partner who is a good match
  - You will likely meet somebody new
  - Prepares you for working in a software company

Until Friday 8/31

On Saturday 9/1

Until Tuesday 9/4

Option 1: Example Questions we Ask

- What programming language would you prefer to use?
- Are you more interested in theory or in building systems?
- Are you familiar with x86 assembly?
- How much time would be so much that you would rather drop?
- How much effort do you plan to invest in Compilers, on average?
- What grade are you aiming for in Compilers?
- Do you prefer to collaborate when writing code?

Option 2: Self Selection

- **Pick your partner carefully!**
- **Have an honest discussion about your goals and expectations**
  - What grades you are willing to accept?
  - How much time will you spend?
  - What times of day you work best?
- **Find somebody who's a good match**

That's not necessarily your best friend.

- **Go through the questionnaire and compare your answers**

Consider switching to Option 1 if there are mismatches.

Labs — Picking a Programming Language

- You can freely choose a programming language to use
- I strongly suggest to use a typed functional language
  - Writing a compiler is a killer app for functional programming
  - Almost every team used Haskell or OCaml last year

Extra recitation on Friday 5 pm!

- We provide starter code for the following languages
  - SML, OCaml, Haskell, and Java
- When picking a language also consider the availability of parser generators and libraries

Logistics

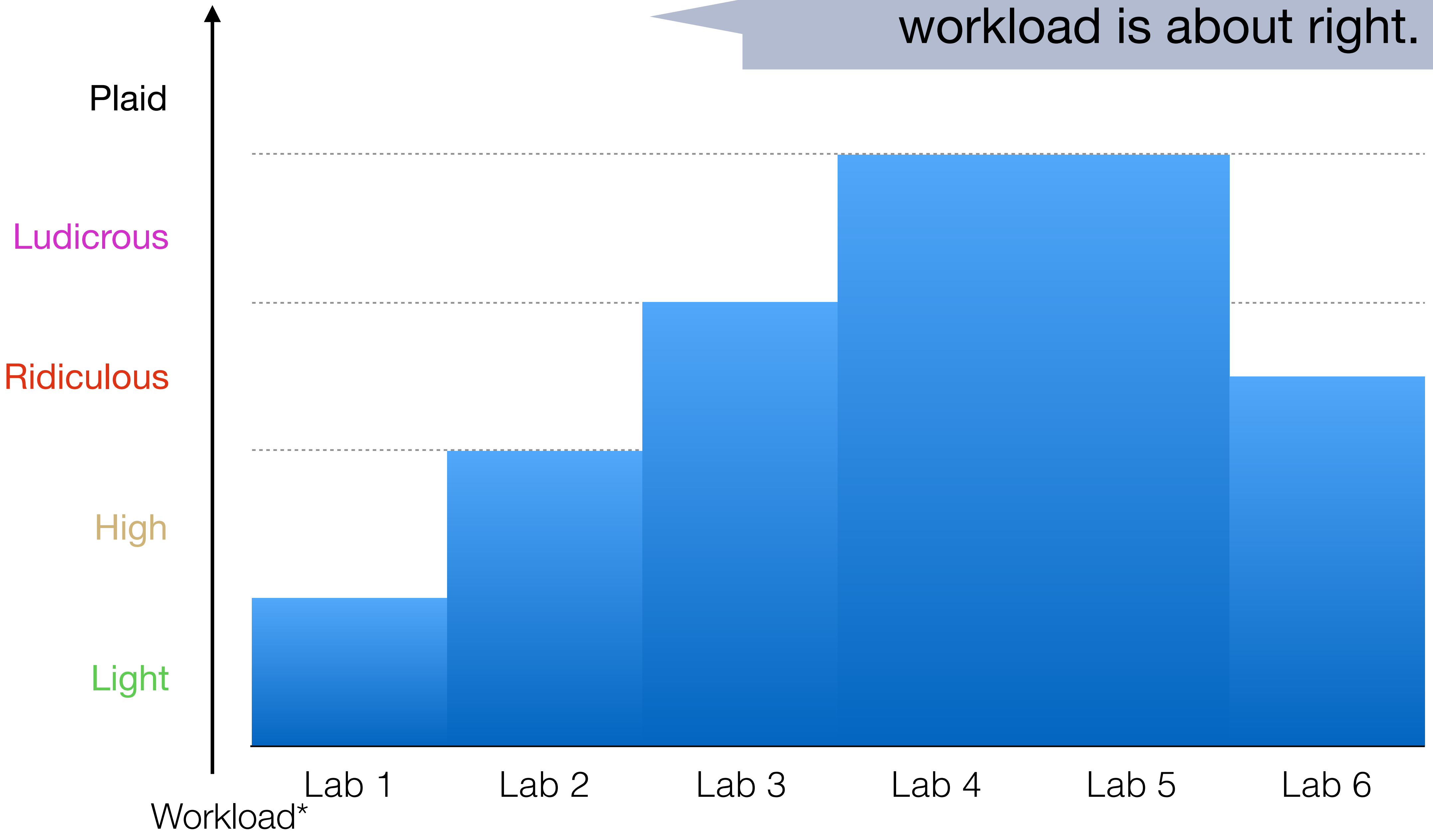
- **Assignments are submitted via Gradescope**
- **Labs are submitted via GitHub**
  - Get a GitHub account and fill out a google form to register your team
  - Receive your group name
  - Receive an invitation to join your group on GitHub
  - Submit your code by pushing to your repository
- **Auto grading with Notolab (new this year: Based on Autolab)**
  - Your compiler is tested against the test cases of other groups
  - And test cases from previous years
  - You can submit as often as you like
  - Best submission before the deadline counts

Advice

- **Labs are difficult and take time**
  - Plan ahead!
  - Set up meetings with lab partners
  - Talk to us and others about design decisions
- **Don’t start the compiler after the tests**
- **Errors carry over to the next lab**
- **Submit early and often**
- **Compilers are complex**
  - That’s part of the fun

Workload Over the Semester

The scale is a joke but the relative workload is about right.



* scale from the movie Spaceballs.

This Year’s Theme

Harry Potter

Deadlines and Academic Integrity

- **Deadlines are midnight (after class); being late results in a late day**
  - You have six (6) late days for the labs (see details online)
  - You have three (3) late days for the assignments (details online)
- **Talk to me or your undergrad advisor if you cannot make a deadline for personal reasons (religious holidays, illness, …)**
- **Don’t cheat! (details online)**
  - Use code only from the standard library, add to Readme
  - Don’t use code from other teams, earlier years, etc.
  - If in doubt talk to the instructor
  - The written assignments need to be completed individually (1 person)

Things you Should Use

- Debugger
- Profiler
- Test programs
- Standard library
- Lecture notes

Well-Being

- **This is only a course!**
  - Take care of yourself
  - Watch out for others
- **Get help if you struggle or feel stressed**
  - If you or anyone you know experiences any academic stress, difficult life events, or feelings like anxiety or depression seek support
  - Counseling and Psychological Services (CaPS) is here to help:
    Phone: 412-268-2922
    Web: http://www.cmu.edu/counseling/

Who should take this course?

15-411 in the Curriculum

- **15-213 Introduction to Computer Systems**

Prerequisite

- **15-411 Compiler Design**
  - How are high-level programs translated to machine code?
- **15-410 Operating System Design and Implementation**
  - How is the execution of programs managed?
- **15-441 Computer Networks**
  - How do programs communicate?

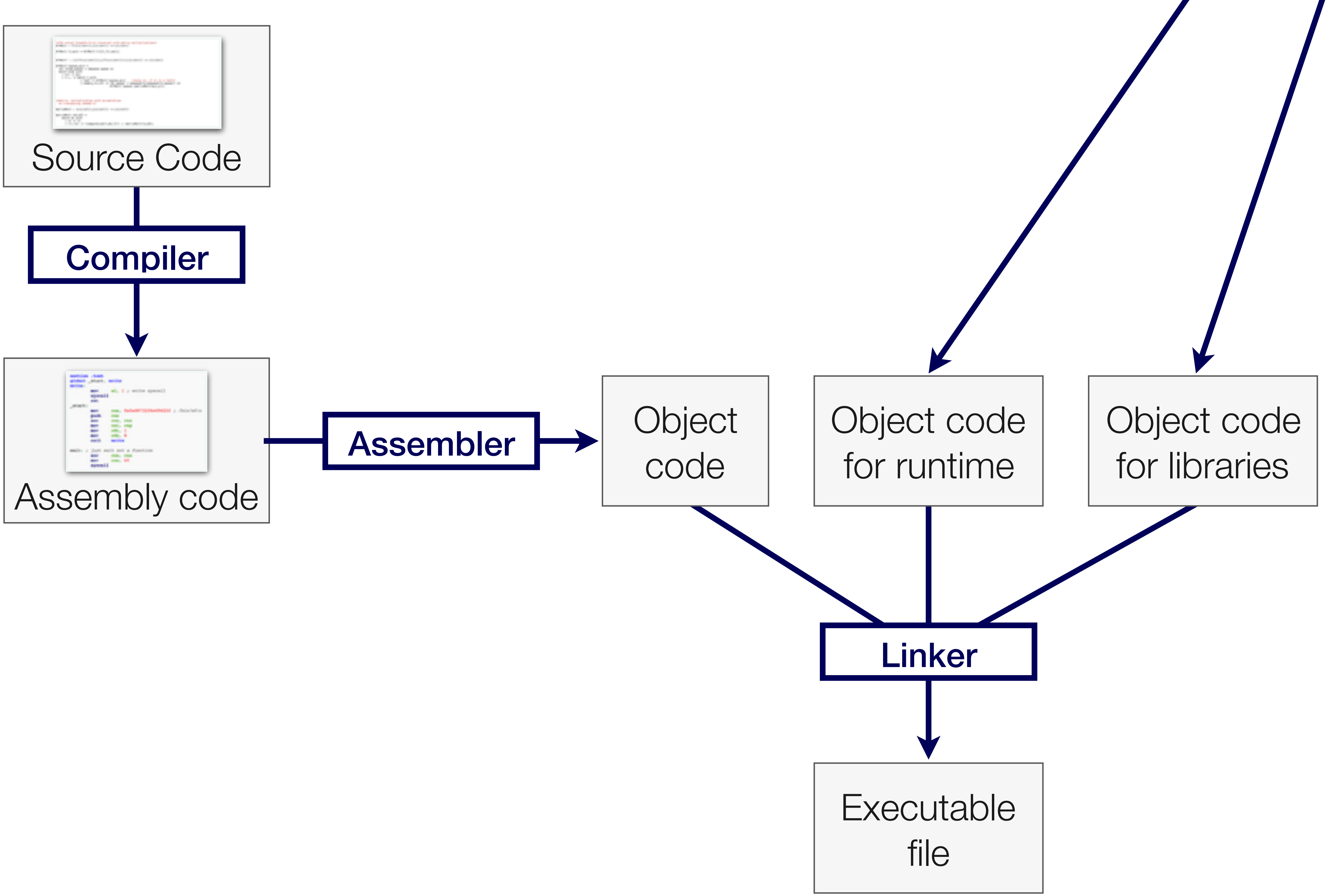
System requirement

- **15-417 HOT Compilation**
  - How to compile higher-order typed languages?

Things you Should Know (Learn)

- **C0 programming language**
  - The source language
- **x86-64 assembly**
  - The target language
- **Functional programming**
  - Highly recommend
- **Git version control**
  - For submitting labs

A closer look at a compiler



Source Code

Compiler

Assembly code

Assembler

Object code

Object code for runtime

Object code for libraries

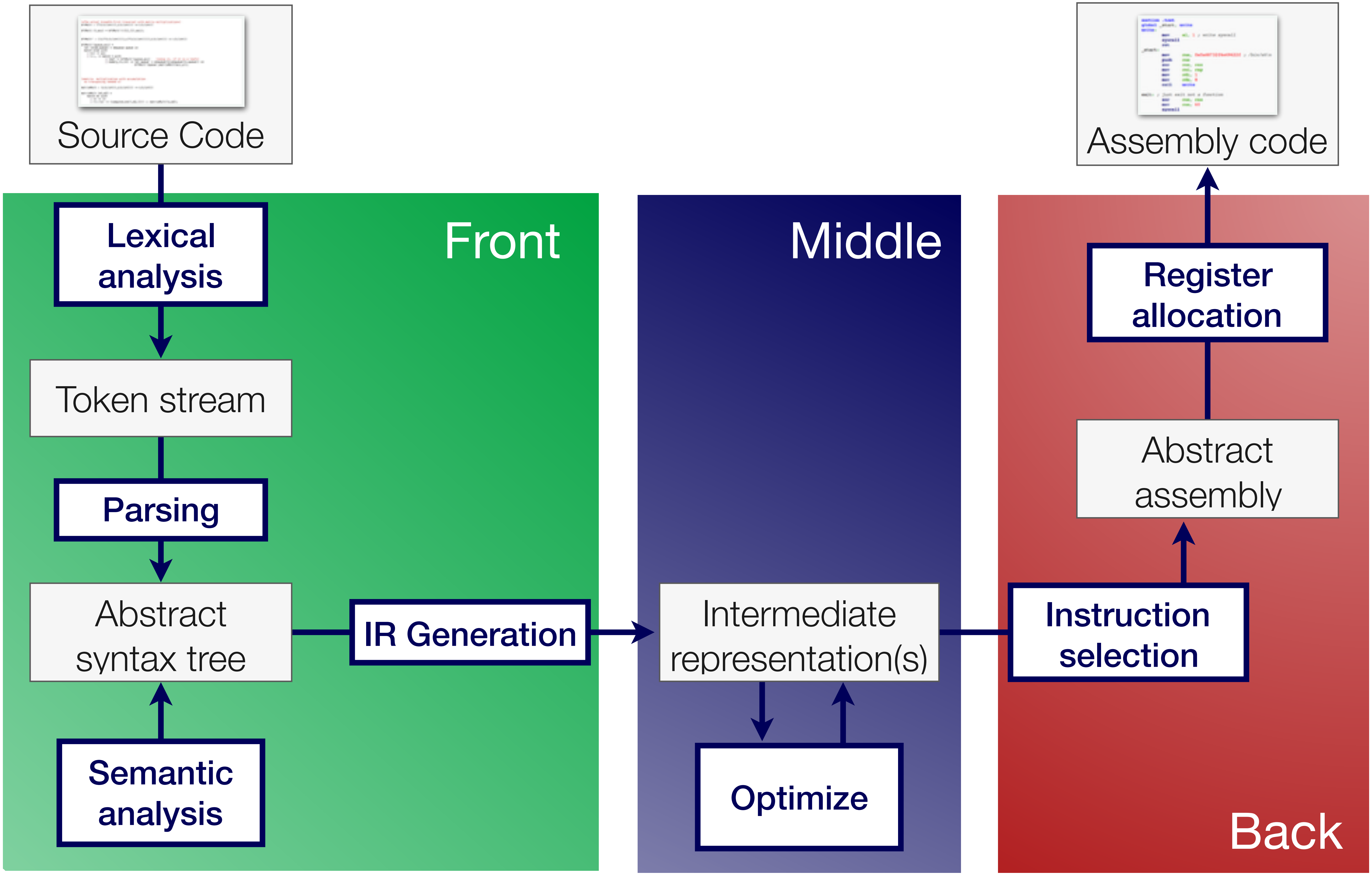
Linker

Executable file

Compiler in Context

Organizing a Compiler

- Split work into different compiler phases **!!**
- Phases transform one program representation into another
- Every phase is as simple as possible
- Phases can be between different types of program representations
- Phases can be on the same program representation



Source Code

Front

Lexical analysis

Token stream

Parsing

Abstract syntax tree

Semantic analysis

IR Generation

Middle

Intermediate representation(s)

Optimize

Back

Instruction selection

Abstract assembly

Register allocation

Assembly code

Compiler Phases

Topics of this week's recitation (You can skip if you took 312)

Reminder: inductive definitions

See: Bob Harper's "Practical Foundations for Programming Languages"