

15-411/15-611 Compiler Design

Jan Hoffmann — Fall 2016

<http://www.cs.cmu.edu/~janh/courses/411/16>

Teaching Staff

- **Instructor:** Jan Hoffmann

Office hours: Tue 10:30am-noon

Thu 1:00pm-2:30pm at GHC 9105

- **Teaching assistants** (office hours TBA)
 - Aaron Gutierrez (OCaml and Java)
 - Evan Bergeron (OCaml)
 - Shyam Raghavan (OCaml and Haskell)
 - Xue An Chuang (OCaml)

Communication and Resources

- **Lecture:** Tue/Thu 9:00-10:20am at BH A51
- **Recitations:** if needed
- **Website:** <http://www.cs.cmu.edu/~janh/courses/411/16>
- **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)
- 1952: A-0, first simple compiler (Grace Hopper)
- 1957: FORTRAN, first real compiler (John Backus)
- 1962: Lisp, first self-hosting compiler (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 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.

How will this work?

Your Responsibilities

No exams.

- **Attend lectures**

- Lecture notes are only supplementary material

- **6 Labs: you will implement compilers for subsets of C0 to x86-64 assembly**

- Lab 1-4: each worth 100 points (total 400 points)
- Code review after Lab 3: 50 points
- Lab 5-6: each 150 points (total 300 points)

With a partner
or individual.

- **5 Assignments: you will complete five written assignments that help you understand the material presented in the lectures**

- Assignments 1-5: each 50 points (total 250 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 (special topics)



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

Source Language

C0, a small safe 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 “taste” the hardware

Runtime system

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

Labs — Finding a Partner

Don't panic.

- **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?
- **You can find partners via *Search for Teammates* on Piazza**
- **You can commit to a team by submitting a google form (link will be posted on Piazza)**

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
- **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 Autolab**
- **Labs are submitted via GitHub**
 - Get a GitHub account and fill out a google form to register your team
 - Receive your group name (Star Wars character)
 - Receive an invitation to join your group on GitHub
 - Submit your code by pushing to your repository
- **Auto grading**
 - 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

Deadlines and Academic Integrity

- **Deadlines are 9am (before 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

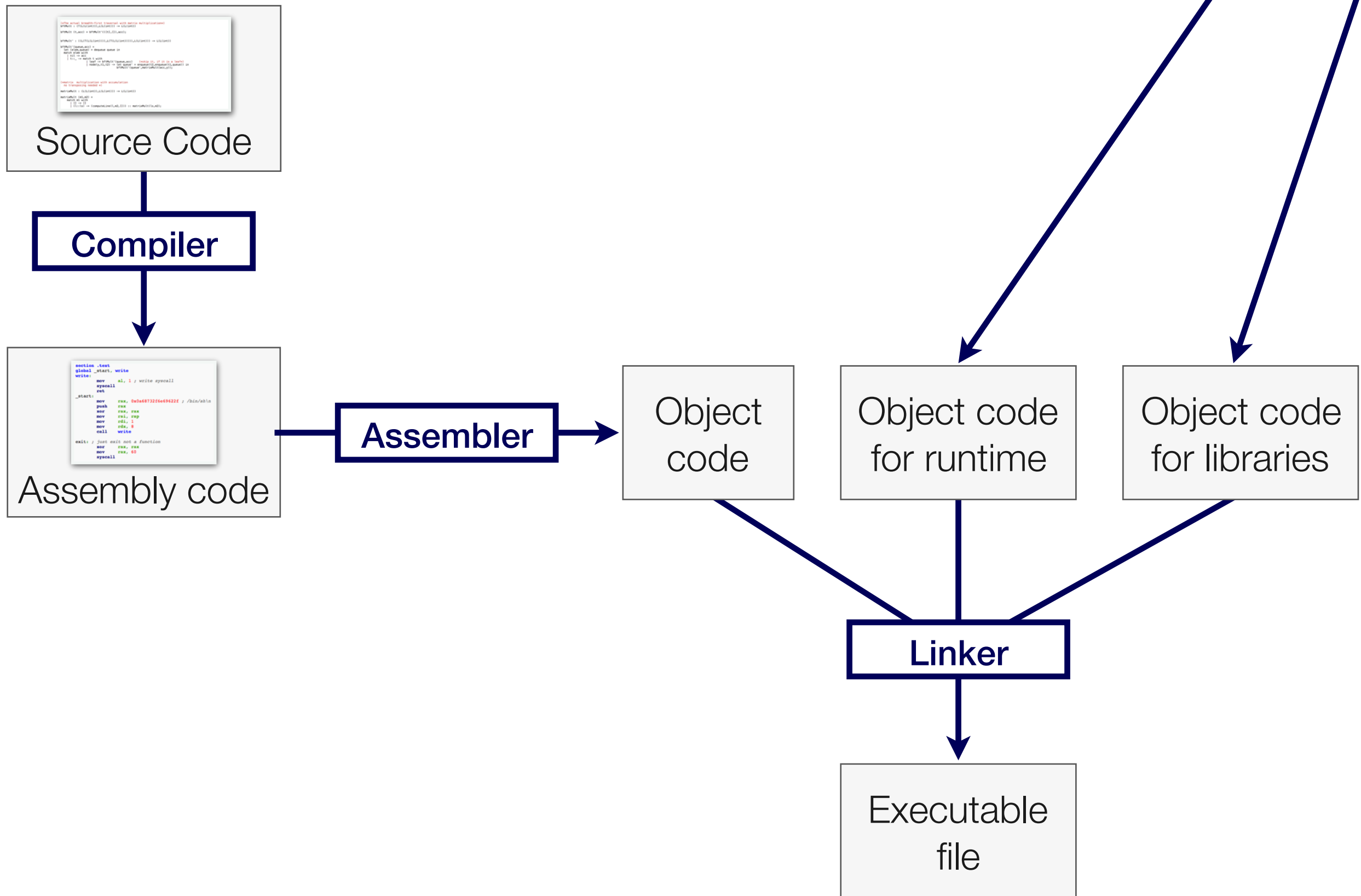
- **15-417 HOT Compilation**

- 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

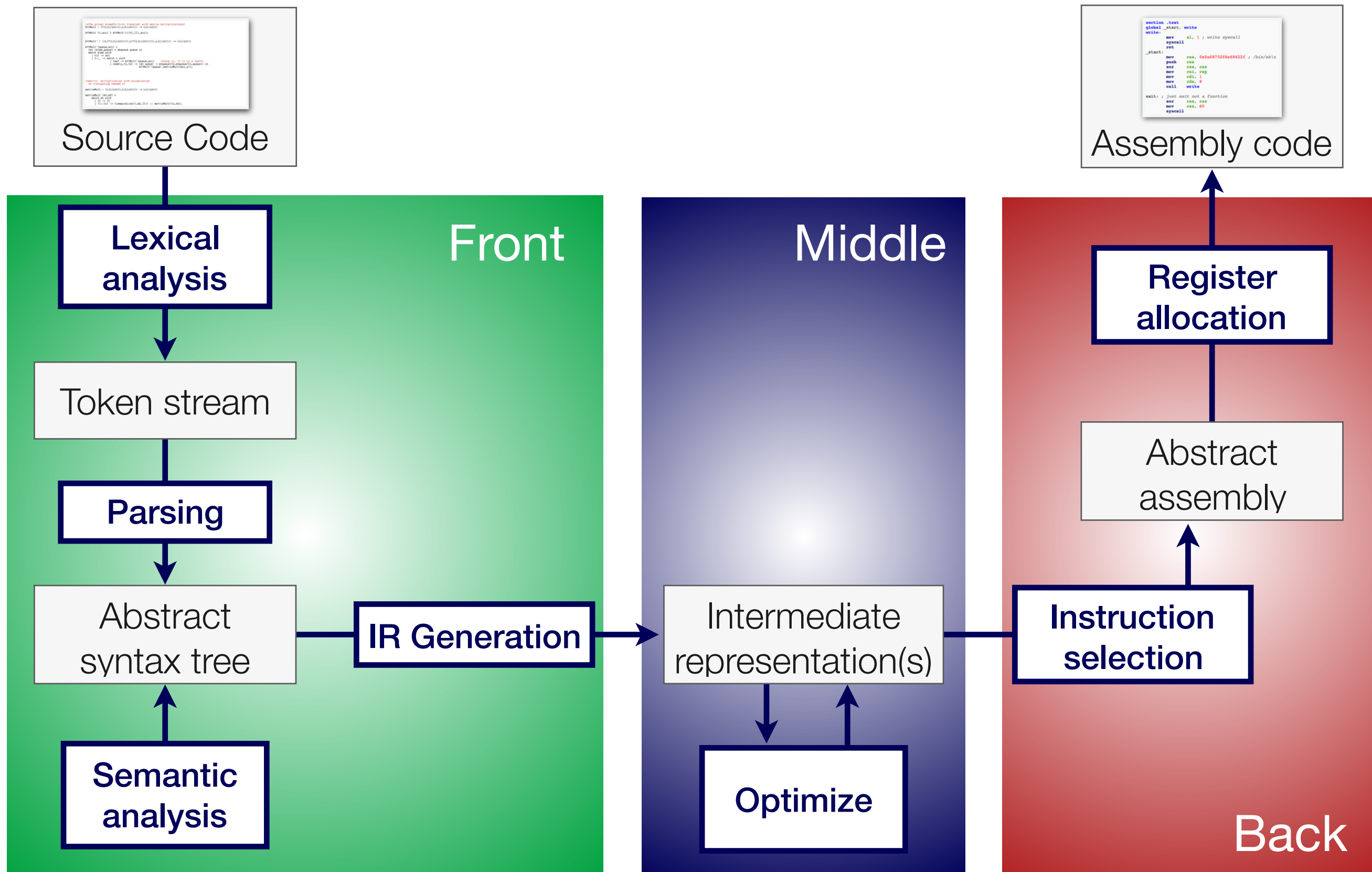
A closer look at a compiler



Compiler in Context

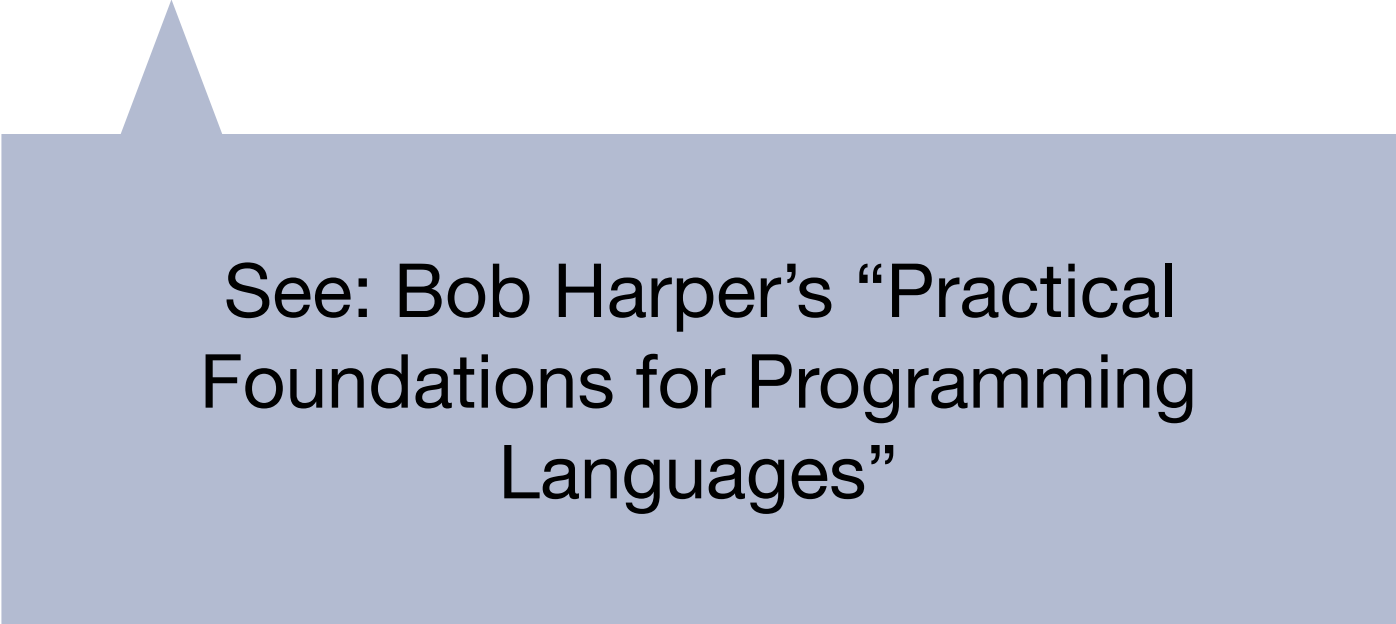
Organizing a Compiler

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



Compiler Phases

Reminder: inductive definitions



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