15-411/15-611
Compiler Design

Robert Simmons, Instructor Fall 2015

https://www.cs.cmu.edu/~rjsimmon/15411-f15

Who's here?

- Me: Rob Simmons, GHC 9101
  - Office hours Tuesday 1:30 and Thursday 10:30
- Teaching Assistants (Office Hours TBA)
  - Ansul Bansal, wrote compiler in Haskell
  - Grant Della Silva, wrote compiler in OCaml
  - Matt Bryant, wrote compiler in OCaml
  - Will Crichton, wrote compiler Rust

Course Elements

- Lectures: Tues & Thurs, 9-10:30, PH 100
- Piazza (including partner search)
  - Enroll yourself from course page if not enrolled
- Lecture notes (Appel's textbook is optional)
- Project and homework-based course:
  - 30% of grade: 5 ***individual*** written homeworks
  - 40% of grade: 4 ***well-specified*** labs (w/ partner)
  - 30% of grade: 2 ***more open-ended*** labs (same partner)

  - ***Academic integrity policy applies!***
  - ***No sharing code, interfaces, ideas between groups!***

WHAT'S THIS COURSE ALL ABOUT?

This is a course about…

- …*fundamental ideas* in compilers
  - Context-free grammars and parsing
  - Single-static assignment form
  - Data flow analysis, liveness
  - Register allocation
- How do compilers impact performance of the code they produce?
- This *will* make you better at writing *compilers.*
- We hope this will make *all* your code better.

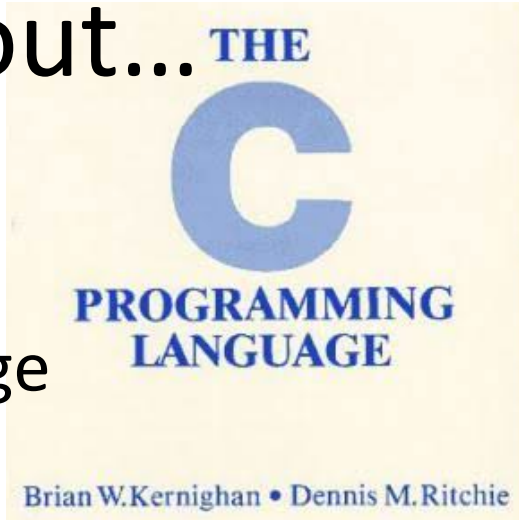
This is a course about…

- …*the design of software systems*
  - Incidentally (!), focus on the design of *compilers*
  - Real software systems are *moving targets* produced and maintained by groups of people under time pressure
- We talk about this surprisingly little, given that it's possibly the *main* point!
- We hope this will make *all* your code better.

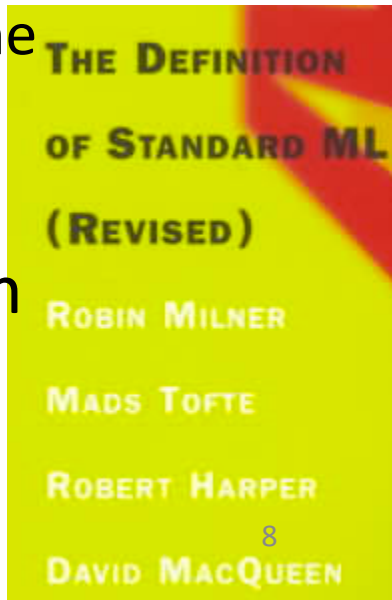
Not really a course about…

- …compilers that *are fast at compiling*
  - We'll generally prefer the simple $O(n^2)$ algorithm to the complicated O(n log n) algorithm.
  - We'll try to at least discuss the tradeoffs here
  - Many compiler projects treat this as a *really important* issue. (See: Google's Go language)
  - Test cases that time out compilers will aggressively be moved into the "optional" category, which you only have to typecheck correctly.

Not really a course about…



- …compilers for *modern languages*
  - C0 is a sequential, imperative language
    - Pointers and integers are all you get!
    - Too modern: safe and well-defined, so you can't do many of the dirty tricks C compilers get to play.
    - Not modern enough: close enough to machine code *already,* optimizations for post-1985 languages (e.g. SML) aren't meaningful.
  - Compiling modern languages is covered in 15-417/617/813, HOT Compilation (Standard ML)



Not really a course about…

- …compilers for *humans*
  - We'll basically ignore error reporting on a more-than-cursory level.
  - It's *amazingly important*, and frequently it's low-hanging fruit.
  - (Warning: Opinion) These HCI issues will be the ***most glaring gap in your knowledge of compilers*** after this course!!!

Q: What do I hope you learn?

- Building, testing, debugging, ***evolving***
- Satisfying performance constraints
- Making and ***revising*** design decisions
  - Implementation language
  - Data structures and algorithms
  - Modules and interfaces
- Reading code
  - Your partner's code
  - Your own code from last month
  - Revise? Refactor? Rewrite?

A: How to learn from “failure.”

- OS, Networks projects are *too big* on purpose
  - Ensures you will make big, important mistakes
- Compilers does projects in the “wrong order”
  - Easy: Compiler Part 1, 2, 3…
  - Here: a whole compiler for growing languages…
- Difficult choices *are part of the point*
  - Always *possible* to rewrite from scratch…
  - Not *required* to update debugging/printing code…
  - Register allocation, SSA *can* be put off until later...

The Systems Requirement

- 15-411 Compiler Design
  - How are your high-level programs translated to low-level hardware instructions?
  - *How do you cope with decisions made for version 1 of the software when you're working on version 3?*
  - *Approach: many versions of the SAME kind of project.*
- 15-410 Operating Systems
  - How is the execution of your programs managed?
  - *How do you maintain abstraction and interfaces when the environment is set against you at every turn?*
  - *Approach: small number of LARGE, RELATED projects, along with in-depth code review.*
- 15-441 Computer Networks
  - How do programs communicate?
  - *How do humans cope with the bewildering number of approaches to the fundamental problem that computers aren't in the same place?*
  - *Approach: small number of UNIQUE, COMPLEX projects.*

HOW IS THIS GOING TO WORK?

Overall Expectations

- Lecture
  - You really want to attend
  - I know this is a terrible time, I will try to keep you awake if you try to get here
- 5 individual written homeworks (30% of grade)
  - Due Thursdays, 11pm.
  - ***Entirely YOUR OWN work***
  - 3 late days, any combination. After that, no credit.
- 6 partnered programming assignments (Labs)
  - ***Entirely YOUR TEAM'S work! (Acknowledge any sources in readme.txt)***

Labs 1-4

- Compiling a series of sub-languages of C0
  - Designed for 15-122
  - Small, safe, fully-specified language
  - Just big enough to be interesting to compile
  - Small enough to manage in a single semester

Labs 1-4

- Each project is a complete, end-to-end compiler
  - Lab 1: straight-line code and some arithmetic
  - Lab 2: loops and more arithmetic
  - Lab 3: functions
  - Lab 4: memory (pointers, arrays, structs)
- Compilers target x86_64 assembly
- Code must interoperate with C functions

Labs 1-4

- Test-driven development
  - Test cases first (week 1), extra credit for good tests?
  - Compiler comes next (week 2)
- Automatic assessment
  - Your compiler is graded against your test cases…
    - …and everyone else's test cases
    - …for this lab and previous labs
    - …and everyone's test cases from 2014 (states), 2013 (elements), 2012 (Lord of the Rings characters), 2011 (birds of prey), and 2010 (dinosaurs)

Labs 5-6

- Choose what to do, do it, then write a paper describing and evaluating what you did.
- Lab 5 is about producing code that runs fast
  - Discussed in lecture throughout the semester
- Lab 6 possibilities:
  - Retarget the compiler
  - Write a garbage collector
  - Implement all of the C1 language
  - Choose your own adventure

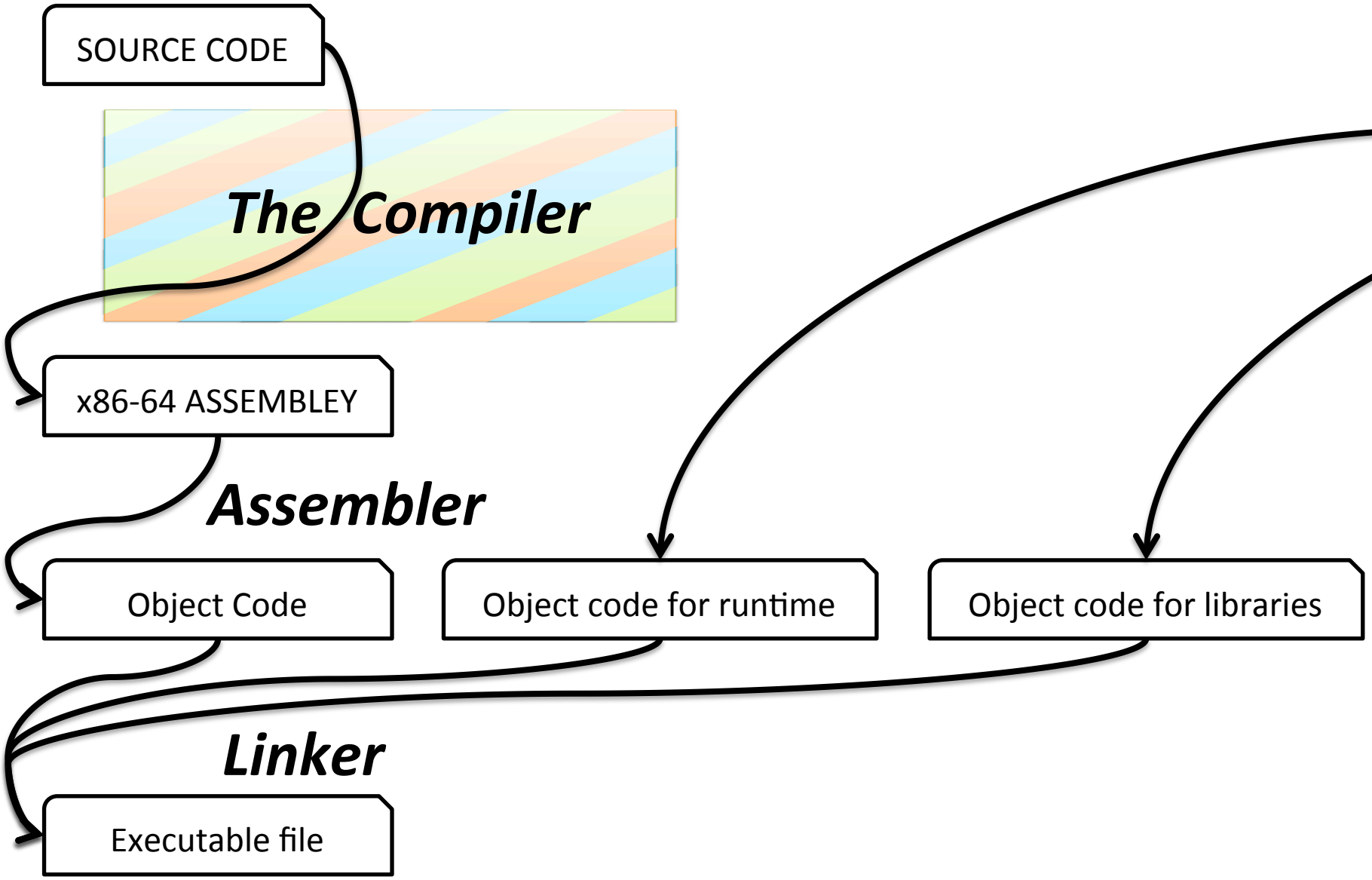
Labs: Code

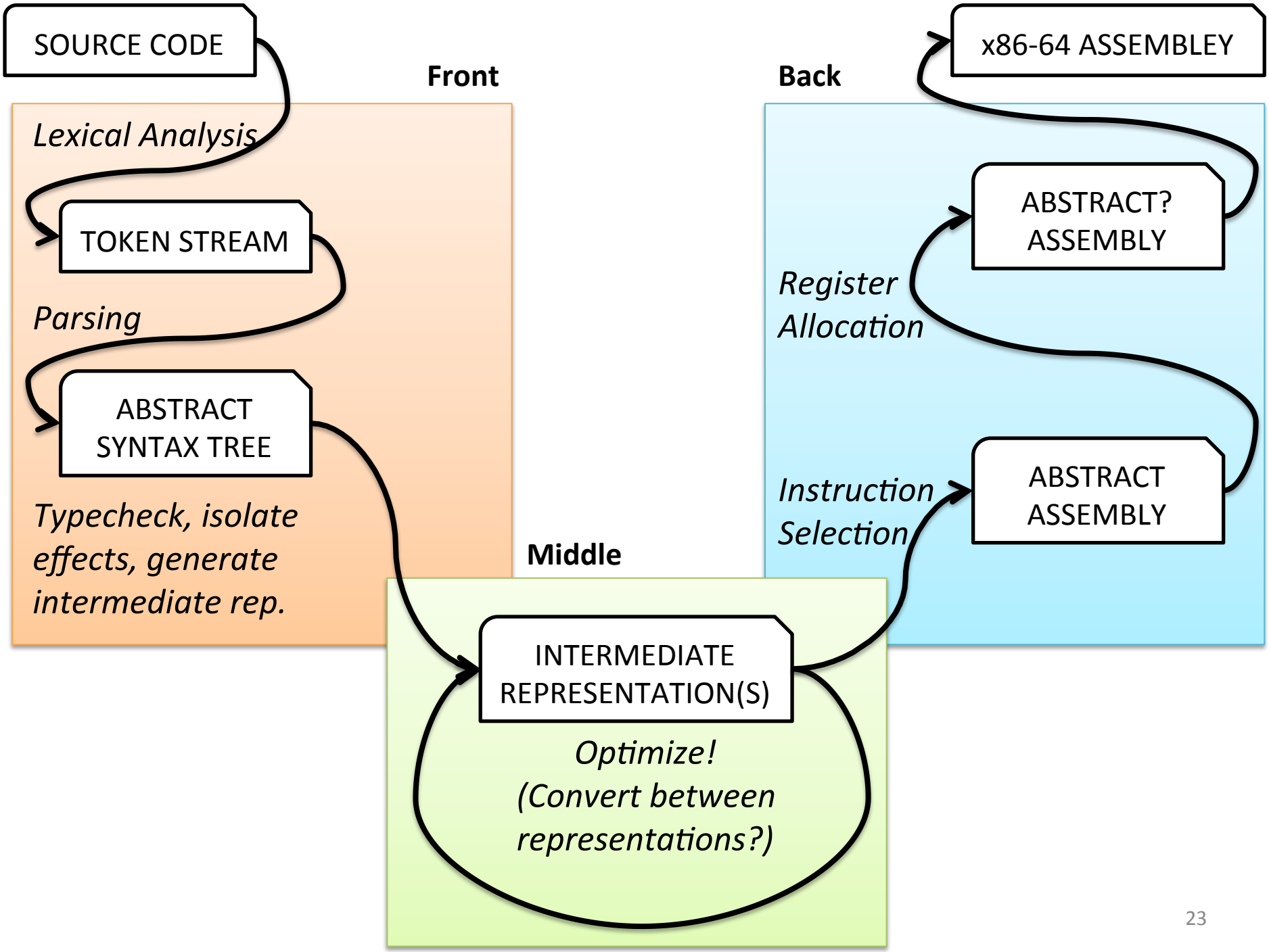
- You get to choose your own implementation language
  - Standard ML, Haskell, OCaml are supported
  - Starter code exists for Rust, Java, and Scala
  - Any other language is permitted

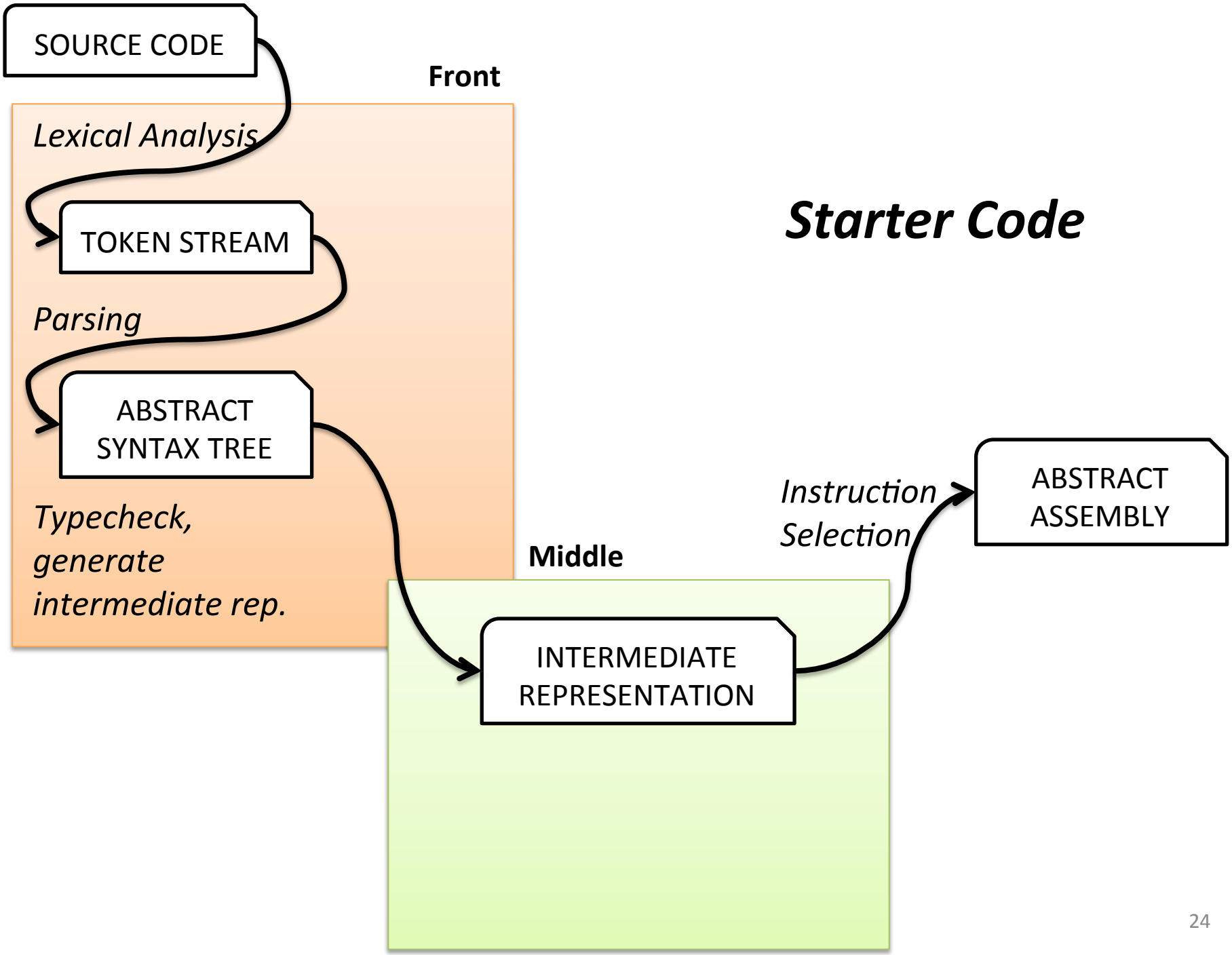
Labs: Partners

- You can find partners after class, on piazza
- Each one is responsible for all the code
  - Read all the code!
  - Strong suggestion: swap roles between labs
  - Everyone has to pull their weight
- Commit by Thursday of next week
- Contact me if you're having partner issues

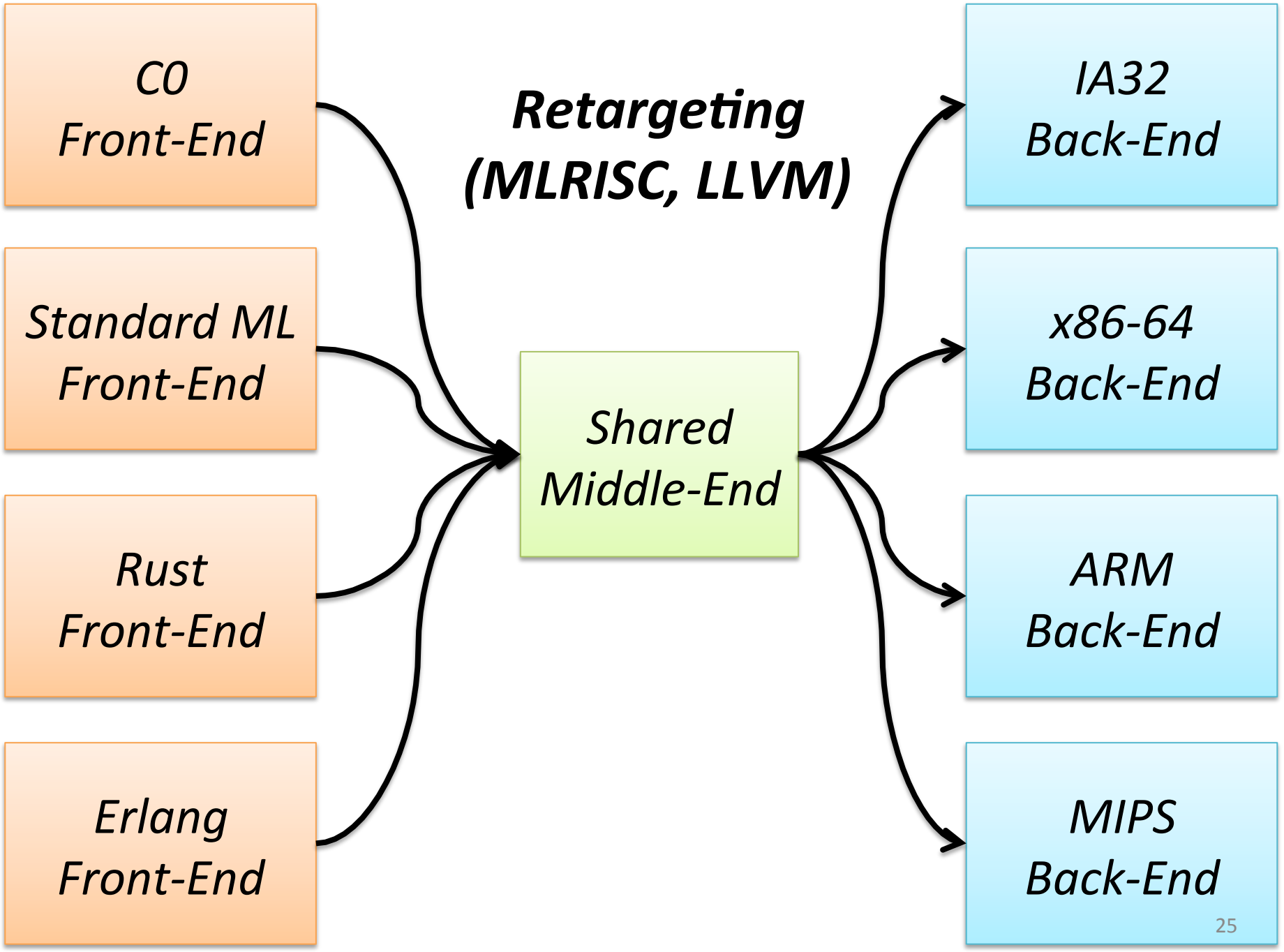
WHAT IS A COMPILER, EVEN?


SOURCE CODE
The Compiler
x86-64 ASSEMBLEY
Assembler
Object Code
Object code for runtime
Object code for libraries
Linker
Executable file


SOURCE CODE
Front
Back
x86-64 ASSEMBLEY
Lexical Analysis
TOKEN STREAM
ABSTRACT?
ASSEMBLY
Parsing
Register
Allocation
ABSTRACT
SYNTAX TREE
Instruction
Selection
ABSTRACT
ASSEMBLY
Typecheck, isolate
effects, generate
intermediate rep.
Middle
INTERMEDIATE
REPRESENTATION(S)
Optimize!
(Convert between
representations?)



Starter Code


C0
Front-End
Standard ML
Front-End
Rust
Front-End
Erlang
Front-End
Retargeting
(MLRISC, LLVM)
Shared
Middle-End
IA32
Back-End
x86-64
Back-End
ARM
Back-End
MIPS
Back-End