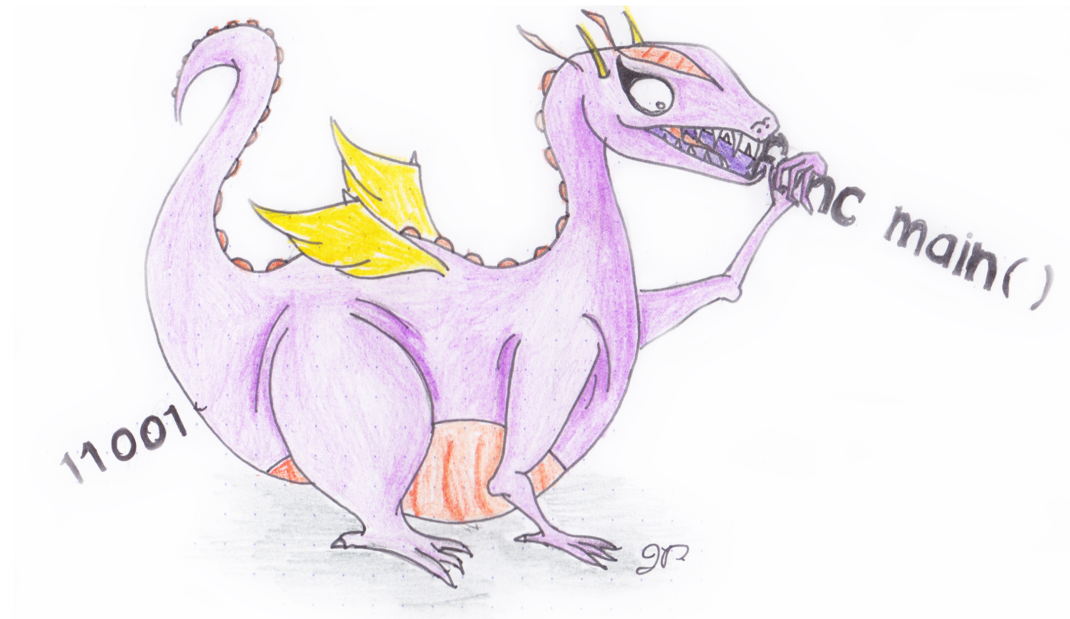
Introduction



COMP 520: Compiler Design (4 credits)

Professor Laurie Hendren

hendren@cs.mcgill.ca

TRF 10:30-11:30, MC 103

This document also serves as the course outline.

Purpose:

- This course covers modern compiler techniques and their application to both general-purpose and domain-specific languages.
- The practical aspects focus on current technologies, primarily using compiler tools effectively, for both general-purpose and domain-specific projects.

Contents:

- *Deterministic parsing:* Scanners, LR parsers, the `flex/bison` and `SableCC` tools.
- *Semantic analysis:* abstract syntax trees, symbol tables & attribute grammars, type checking, resource allocation.
- *Virtual machines and run-time environments:* stacks, heaps, objects.
- *Code generation:* resources, templates, optimizations.
- *Surveys on:* native code generation, static analysis, ...

Schedule:

- Lectures: 3 hours/week.

Prerequisites:

- COMP 273, COMP 302, (COMP 330), ability to read and write “large” programs.
- Students without COMP 330 should read the background material indicated in Week 1 of the web page ASAP.

Lecturer:

- Professor Laurie Hendren, McConnell 228, Office hours: Fridays 11:30-12:30

TAs:

- Vincent Foley (1.5 units), Go project, McConnell 226/234, Office Hours: Monday and Wednesday, 13:00-14:00. If you have classes at that time, you mail e-mail him to set up an alternate meeting time at `vfoley@gmail.com`.
- Faiz Kahn (0.5 units), OncoTime project, McConnell 226/234, Office Hours: Tuesday, 13:00-14:00, e-mail `dridon@gmail.com`

Midterm: just after the study break, date and place TBD; **Final exam:** during the final exam period

Marking Scheme:

- 10% midterm, 25% final exam, 65% assignments and project
- the 65% for assignments and projects will be divided as follows:
  - 10% for the individual assignments
  - 10% for the JOOS peephole optimizer (group)
  - 25% for content submitted at milestones (group)
  - 20% for the final compiler and report (group)
- Group members may be given different grades on the project work if the contributions are not reasonably equal.
- Assignments and project milestones are due at midnight of the due date. A penalty of 10% per day late is given. Assignments will not be accepted after solutions have been circulated.

Academic Integrity:

- McGill University values academic integrity. Therefore all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures.

  `http://www.mcgill.ca/srr/honest/`

- In terms of this course, part of your responsibility is to ensure that you put the name of the author on all code that is submitted. By putting your name on the code you are indicating that it is completely your own work.

- If you use some third-party code you must have permission to use it and you must clearly indicate the source of the code.

Other Required McGill Statements:

- In accord with McGill Universitys Charter of Students Rights, students in this course have the right to submit in English or in French any written work that is to be graded.

Course material:

- Slides for the lectures.
- Recommended Textbook, *Crafting a Compiler* by Fischer, Cytron and LeBlanc. Available in hardcopy at the McGill Bookstore. An online version (much cheaper) is available at `http://www.coursesmart.com/0136076599`. There should also be a copy on reserve in the library, and you may find other sources for the book.
- Alternate Textbook, *Modern compiler implementation in Java (2nd edition)* by Appel and Palsberg. Free online version available via McGill Library (access via a McGill IP or VPN), `http://mcgill.worldcat.org/title/modern-compiler-implementation-in-java/oclc/56796736`.
- Online readings.
- Extensive documentation on the course web pages.

The text book and the online readings:

- are mainly background reading;
- do not discuss the Go and OncoTime projects used in this course; and
- are required for the exercises.

The slides:

- are quite detailed; and
- are available online via the web site, either just before or after the lecture.

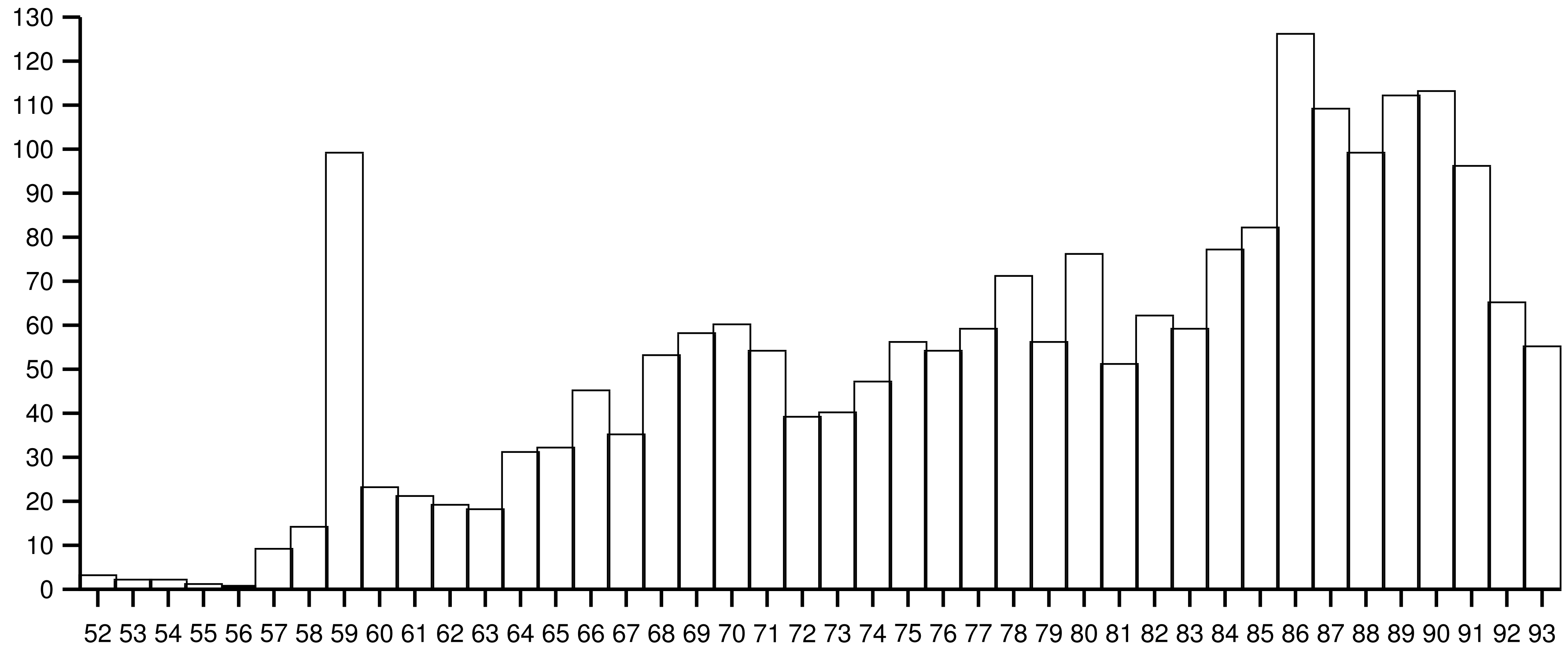
The web pages:

- aim to contain all information;
- provide on-line documentation; and
- may be updated frequently.

Github and Groups

- The main projects will be done in groups of 3 (although you may also have a group of 2 if you prefer).
- We will use the github versioning system.
- Good for maintaining source code but also for other things.
- You will use it for your submissions.
- Tasks for this week
  - Read github documentation on the website
  - Start looking for group mates, you will have to declare your group by the end of the 2nd week of lectures.

New programming languages per year:



The compiler for the FORTRAN language:

- was implemented in 1954–1957;
- was the world's first compiler;
- was motivated by the economics of programming;
- had to overcome deep skepticism;
- paid little attention to language design;
- focused on efficiency of the generated code;
- pioneered many concepts and techniques; and
- revolutionized computer programming.

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL
  LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,
     F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

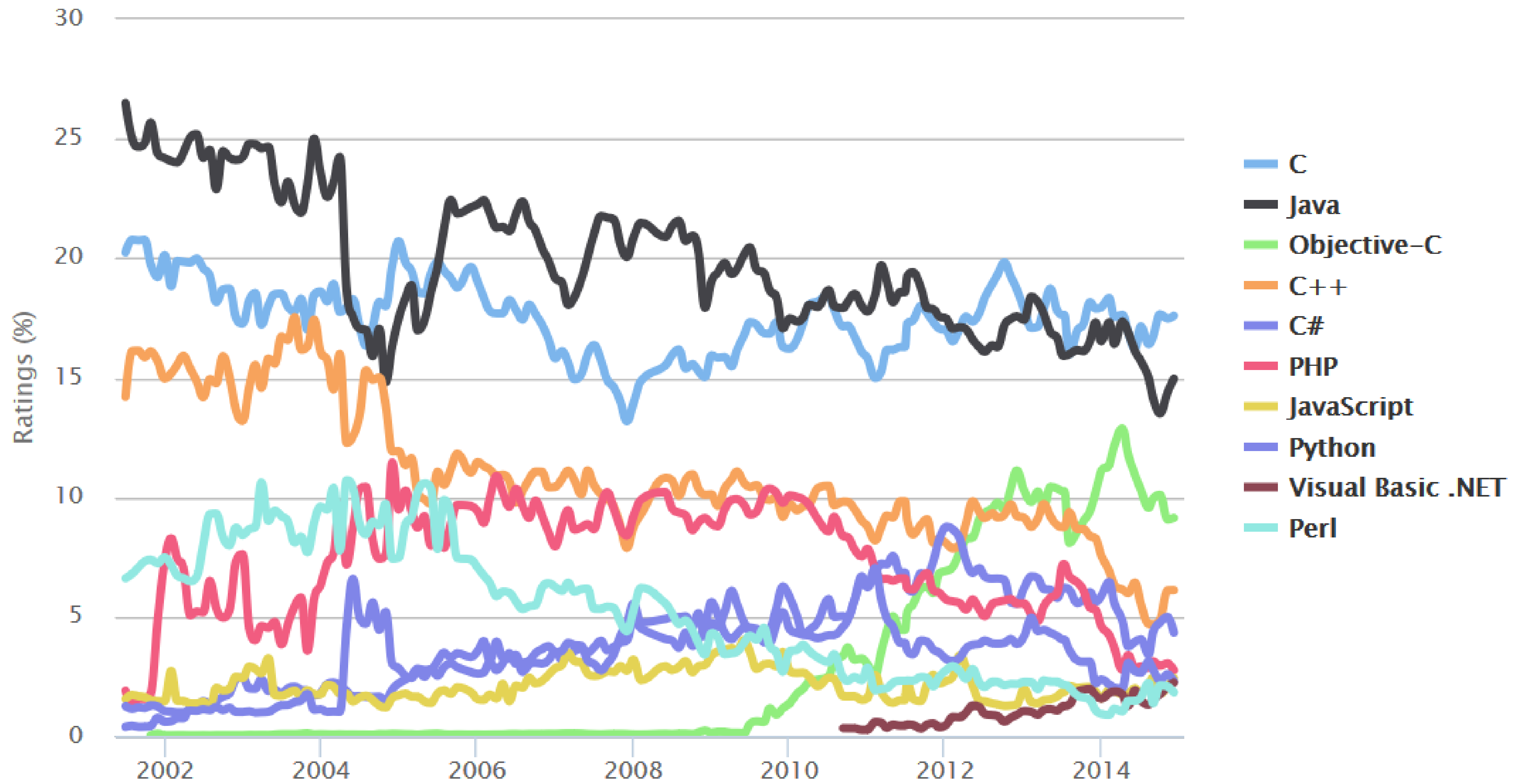
General-purpose languages:

- allow for arbitrarily useful programs to be written
- in the theoretical sense are all Turing-complete; and
- are the focus of most programming language courses.

Prominent examples are:

- C
- C++
- FORTRAN
- Java
- ...

General purpose languages fairly obviously require full-scale compiler technology to run efficiently.


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
C
Java
Objective-C
C++
C#
PHP
JavaScript
Python
Visual Basic .NET
Perl

Domain-specific languages:

- extend software design; and
- are concrete artifacts that permit representation, optimization, and analysis in ways that low-level programs and libraries do not.
- They may even be visual! (e.g. boxes & arrows)

Prominent examples are:

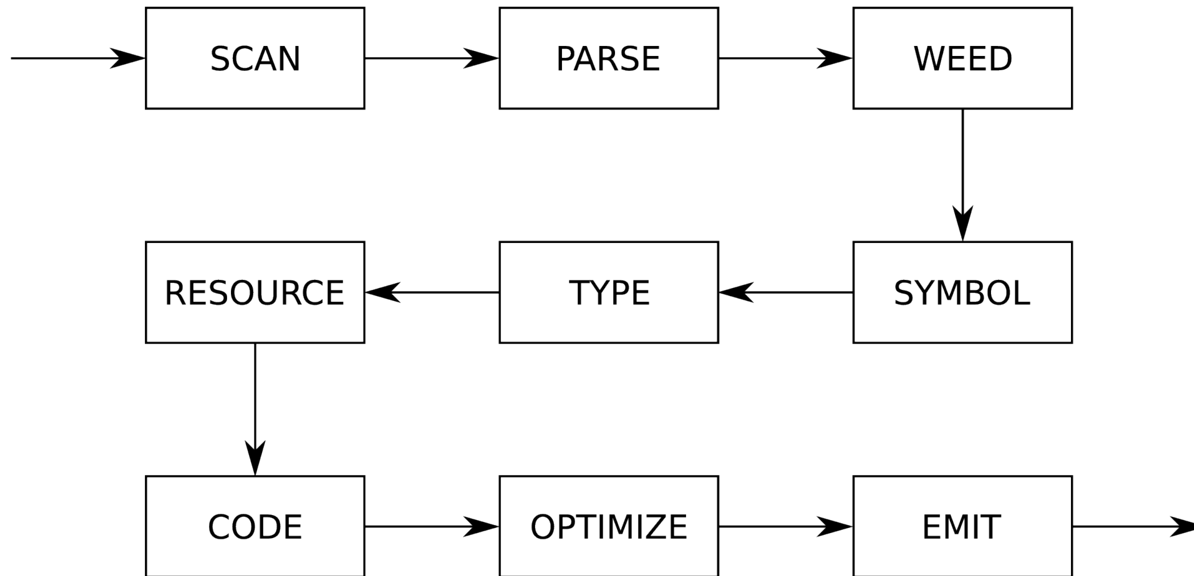
- LaTeX
- `yacc` and `lex`
- Makefiles
- HTML
- SVG
- ...

Domain-specific languages also require full-scale compiler technology.

Reasons to learn compiler technology:

- understand existing languages;
- appreciate current limitations;
- talk intelligently about language design;
- implement your very own general purpose language; and
- implement lots of useful domain-specific languages
  - encoding everything in XML is not always the best way to go (see Makefiles vs. Ant's build.xml)

The phases of a modern compiler:



The individual phases:

- are modular software components;
- have their own standard technology; and
- are increasingly being supported by automatic tools.

Advanced backends may contain an additional 5–10 phases.

Phases as illustrated in text, ”Crafting a Compiler”

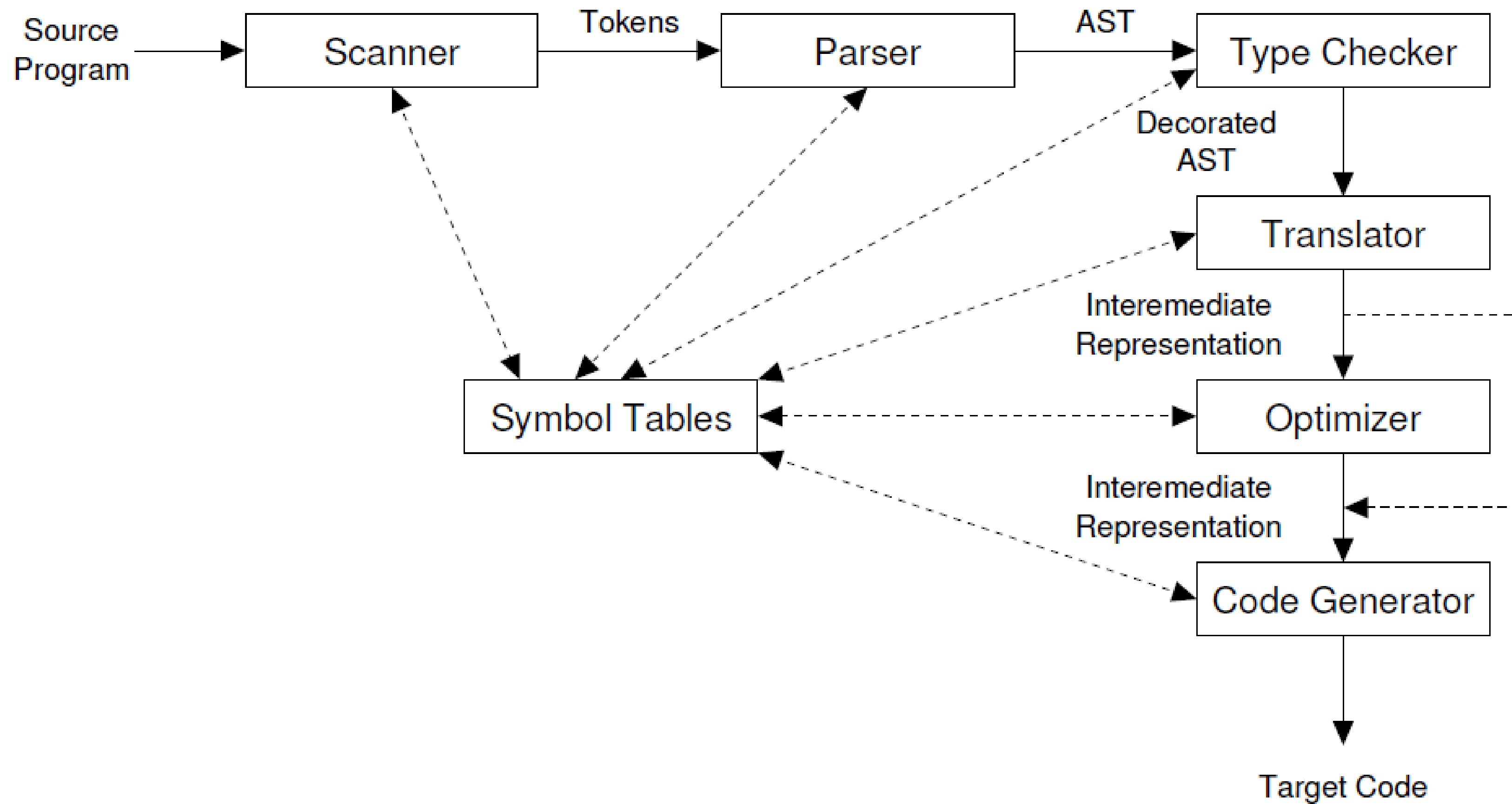
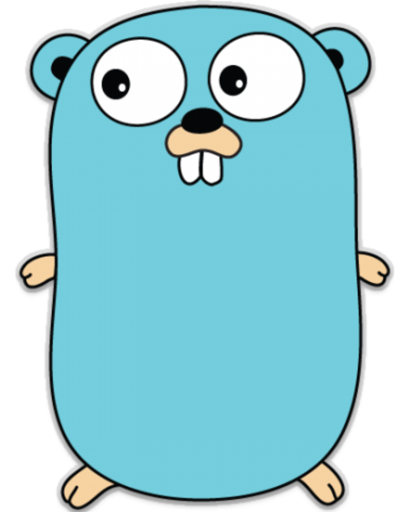


Figure 1.4: A syntax-directed compiler. AST denotes the Abstract Syntax Tree.

The JOOS example:

- Java's Object-Oriented Subset
- is compiled to Java bytecode;
- illustrates a general purpose language;
- allows client-side programming on the web;
- is used to teach by example;
- has source code available;

The Go project:



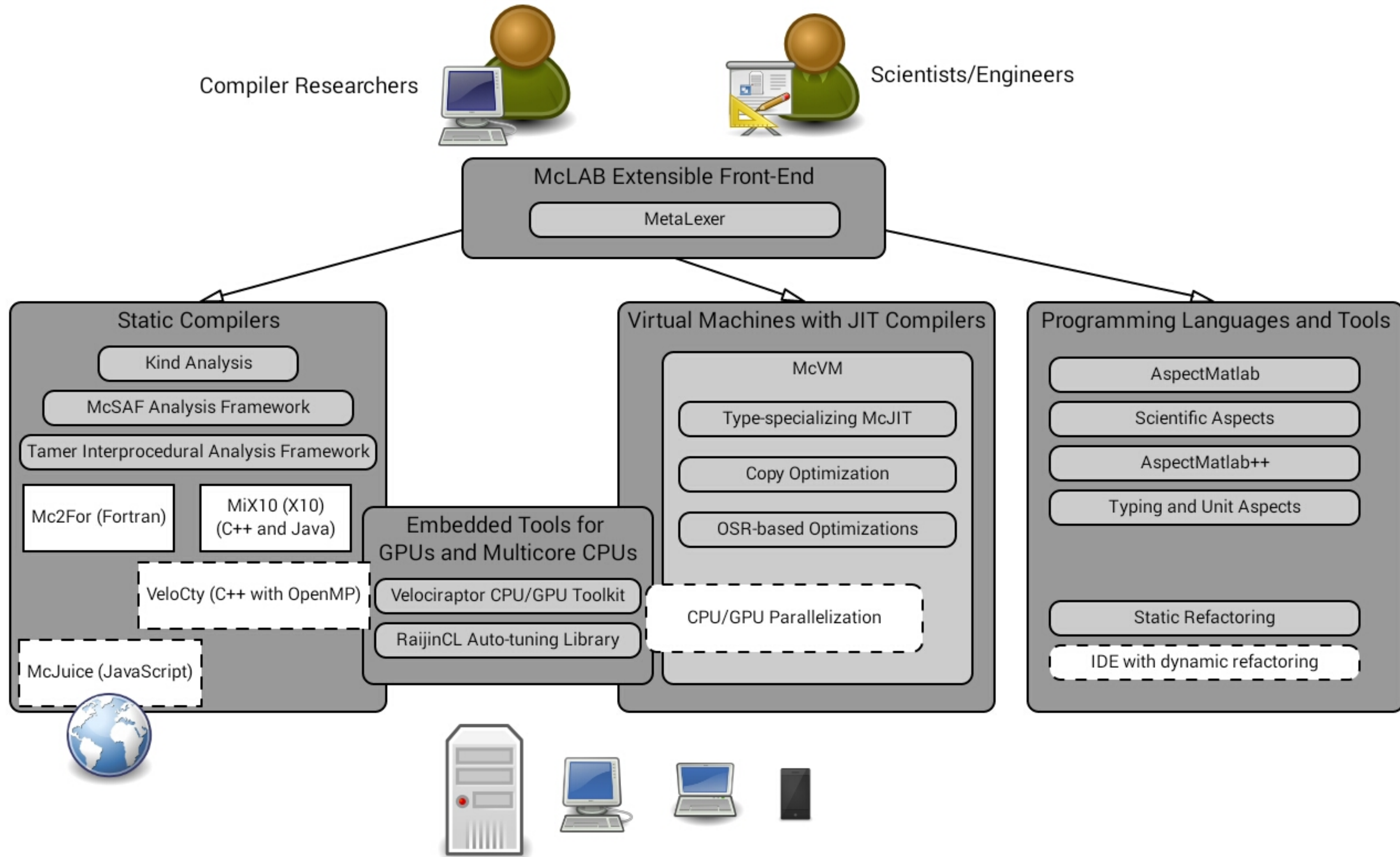
- A complete compiler for a significant subset of the Go Programming Language (`https://golang.org`)
- Example of a modern language, "Go is an open source programming language that makes it easy to build simple, reliable and efficent software."
- Invented by Robert Griesemer, Rob Pike and Ken Thompson at Google.
- Can use any compiler toolkit in any language (will learn C and Java tools in class).
- Can produce any kind of output code, high-level code [C, JavaScript, Python, ...], or low-level code [assembly, LLVM, Java bytecode, Android code, ...]. We will learn Java bytecode in class.

The OncoTime project (?):

- Example of a domain-specfic language;
- Designed to read data from mySQL database;
- Underlying representation is a sequence of patient events;
- High-level operators to find sub-sequences;
- Built-in analysis functions;
- Generates high-level code with visualizations of data (Python?).

`http://acfro.com/wp-content/uploads/2012/07/image003.jpg`

My Compiler Research Group (McLAB)



My Radiation Oncology Research Group (HIG)

- Composed of Computer Scientists, Medical Physicists, and Radiation Oncologists;
- Studying ways of using data to improve the patients' experience and improve safety.
- Several COMP/MDPH 396 projects available, check `http://www.mcgill.ca/science/category/tags/available-396-projects/`.
- talk to me if you are interesting in undergraduate or graduate research in either the McLab or HIG groups.

The top 10 list of reasons why we use C for compilers:

10) it's tradition;
9) it's (truly) portable;
8) it's efficient;
7) it has many different uses;
6) ANSI C will never change;
5) you must learn C at some point;
4) it teaches discipline (the hard way);
3) methodology is language independent;
2) we have `flex` and `bison`; and
1) you can say that you have implemented a large project in C.

The top 10 list of reasons why we use Java for compilers:

10) you already know Java from previous courses;
9) run-time errors like null-pointer exceptions are easy to locate;
8) it is relatively strongly typed, so many errors are caught at compile time;
7) you can use the large Java library (hash maps, sets, lists, . . .);
6) Java bytecode is portable and can be executed without recompilation;
5) you don't mind slow compilers;
4) it allows you to use object-orientation;
3) methodology is language independent;
2) we have `sablecc`, developed at McGill; and
1) you can say that you have implemented a large project in Java.

Bootstrapping as illustrated in text, ”Crafting a Compiler”

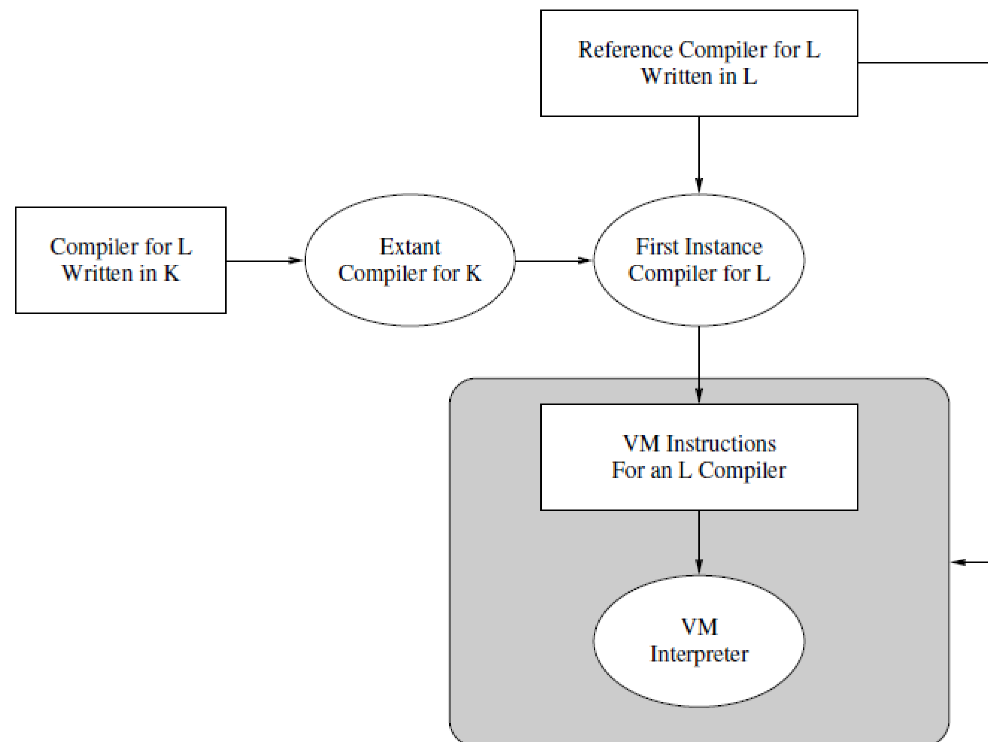
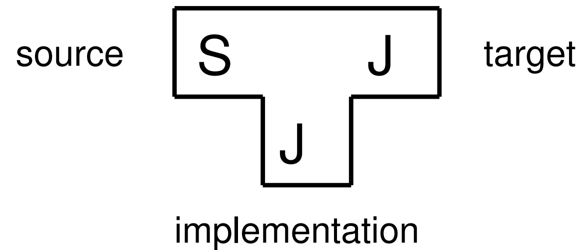


Figure 1.2: Bootstrapping a compiler that generates VM instructions. The shaded portion is a portable compiler for L that can run on any architecture supporting the VM.

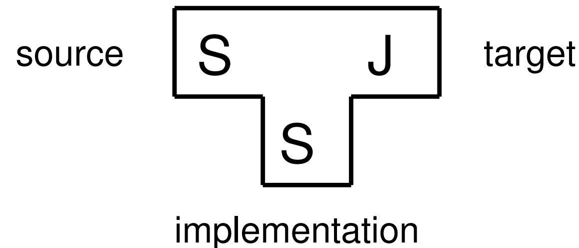
How to bootstrap a compiler (SCALA example):

- we are given a source language (L in the reading), say SCALA; and
- a target language (M in the reading), say Java.

We need the following:



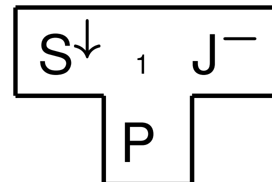
Of course, actually we like SCALA much better than Java and would therefore rather implement SCALA in itself:



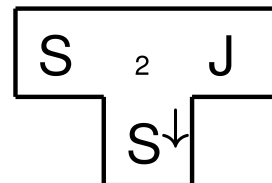
Define the following:

- $S^{\downarrow}$ is a simple subset of SCALA;
- J^{-} is inefficient Java code, and
- P is our favourite programming language, here “Pizza”.

We can easily implement:

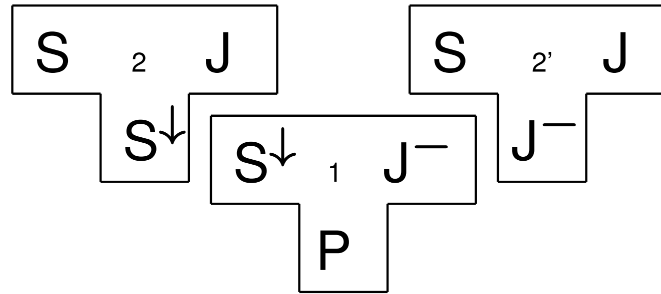


and in parallel, using $S^{\downarrow}$, we can implement:



using basically our favourite language.

Combining the two compilers, we get:



which is an inefficient SCALA compiler (based on generated Java code) generating efficient Java code.

A final combination gives us what we want, an efficient SCALA compiler, written in SCALA, running on the Java platform.