Introduction

COMP 520: Compiler Design (4 credits)

Alexander Krolik

`alexander.krolik@mail.mcgill.ca`

MWF 13:30-14:30, MD 279

This document also serves as the course outline.

`http://www.cs.mcgill.ca/~cs520/2017/`.

Purpose:

- This course serves as an introduction to modern compiler techniques; and
- Explores its application to both general-purpose and domain-specific languages.

End Goal:

- Effectively use compiler tools for general-purpose and domain-specific projects; and
- Produce functional compilers for general-purpose languages.

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 available at: `http://www.cl.cam.ac.uk/teaching/2003/RLFA/notes.pdf`

Lecturer:

- Alexander Krolik, McConnell 226/234, Office Hours: Wednesdays 14:30-15:30

TAs:

- Hanfeng Chen (`hanfeng.chen@mail.mcgill.ca`), McConnell 226/234, Office Hours: Thursdays 16:00-17:00
- Prabhjot Sandhu (`prabhjot.sandhu@mail.mcgill.ca`), McConnell 226/234, Office Hours: Tuesdays 11:30-12:30

If you have class at these times, send one of us an email to arrange another meeting time.

Marking Scheme:

- **Assignments:**
  - 10% for individual assignments (2 x 5%)
  - 10% for the JOOS peephole optimizer (group)
- **GoLite Project:**
  - 35% for content submitted at milestones (group)
  - 10% for the final compiler and report (group)
  - Group members may be given different grades on the joint work if the contributions are not reasonably equal.
- **Midterm:** 10% midterm (after reading week)
- **Final exam:** 25% final exam (during exam period)
- *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 University's 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:

- Lectures & slides.
- Recommended Textbook, *Crafting a Compiler* by Fischer, Cytron and LeBlanc. Available in hardcopy at the McGill Bookstore.

  You can find a support web site with additional tutorial and reference information at `http://www.cs.wustl.edu/~cytron/cacweb/`.

  You can probably find a cheaper ebook version, for example on `www.amazon.ca` or `www.vitalsource.com`. Make sure that you get the version of the book with the three authors listed above.

  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`.
- Readings and documentation from the course website.
- Facebook group (?) or myCourses for discussions.

The textbook and online readings:

- are mainly additional background reading;
- do not discuss the Go project in this course; and
- are required for additional exercises.

The slides:

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

The course website:

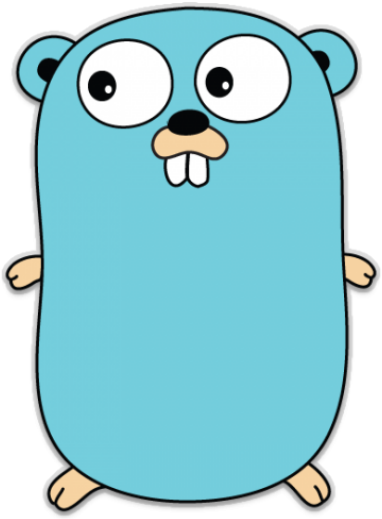
- links to all important information;
- provides online documentation for the project and tools; and
- will be updated frequently.

JOOS language and compiler:

- 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 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.



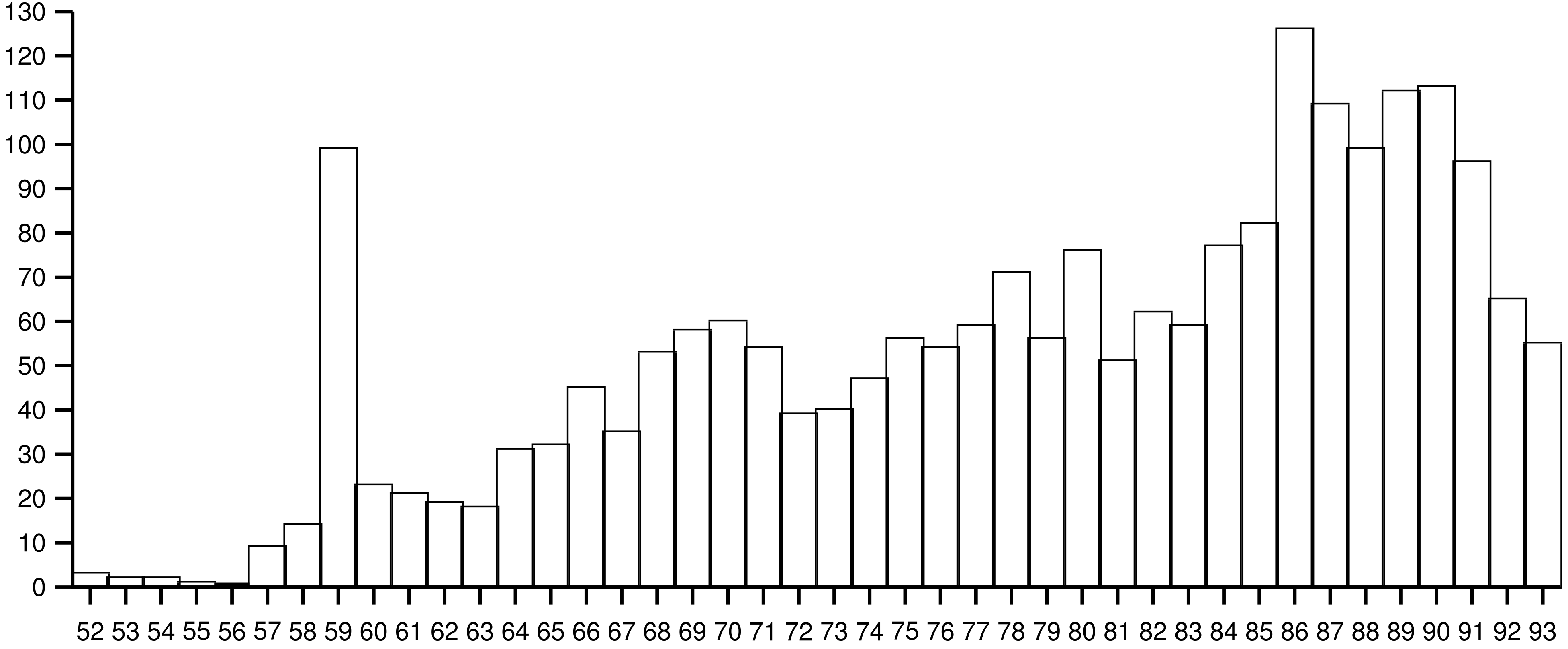
The GoLite project:

- Produce a complete compiler for a significant subset of the Go Programming Language
- 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.

GitHub and Groups:

- The GoLite project and Peephole Optimizer will be done in groups of 2 or 3.
- NOTE: Undergraduates are **strongly** recommended to work in groups of 3.
- We will use the `git` versioning system for the GoLite project.
- Get started soon:
  - Setup a GitHub account and read documentation on the website as necessary.
  - Start looking for team mates, you will have to declare your group by the add/drop deadline.

New programming languages per year:



General-purpose languages:

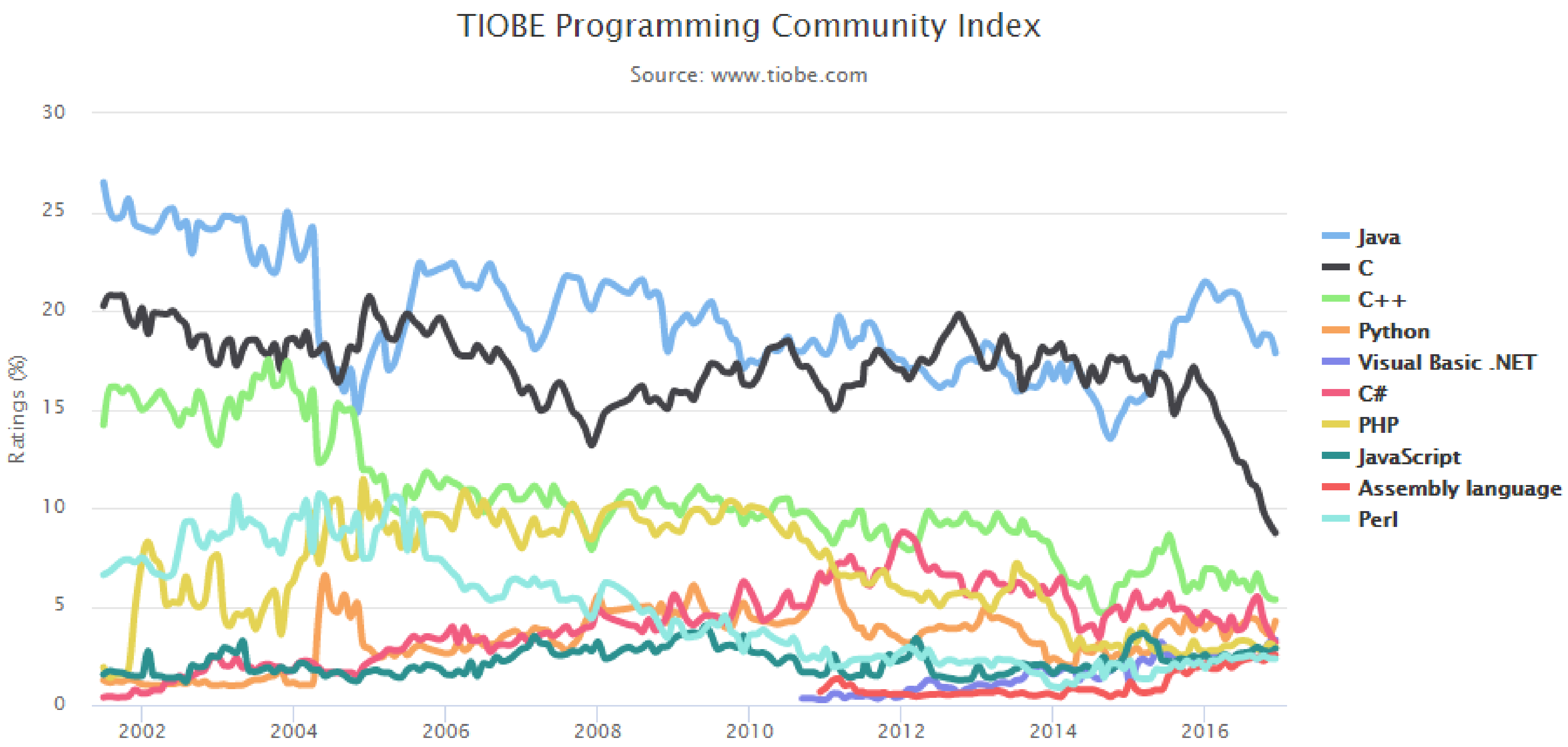
- allow for arbitrarily useful programs to be written
- in the theoretical sense are all Turing-complete; and
- are the focus of most programming language courses.

Prominent examples:

- C
- C++
- Java
- JavaScript
- Rust
- . . .

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

Popular programming languages:



http://www.tiobe.com/tiobe-index/

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:

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

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

What is a compiler?

When talking about compilers you might think of:

- GCC
- clang/LLVM
- javac
- rustc
- . . .

A modern compiler:

- takes source code written in programming language **S**;
- produces equivalent code in target language **T**; and
- may perform optimizations (performance, space, size) [COMP 621].

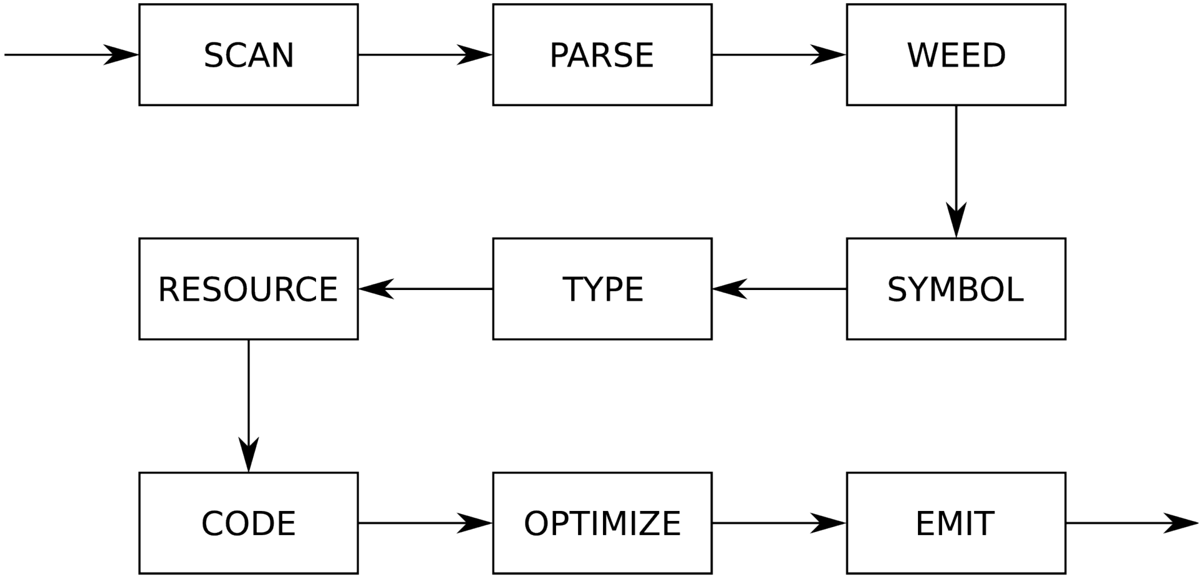
The target language **T** can be another high-level language or machine code.

Phases of a modern compiler:

Compilation is divided into distinct phases. 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, including multiple levels of optimization.



Phases as illustrated in the textbook, ”Crafting a Compiler":

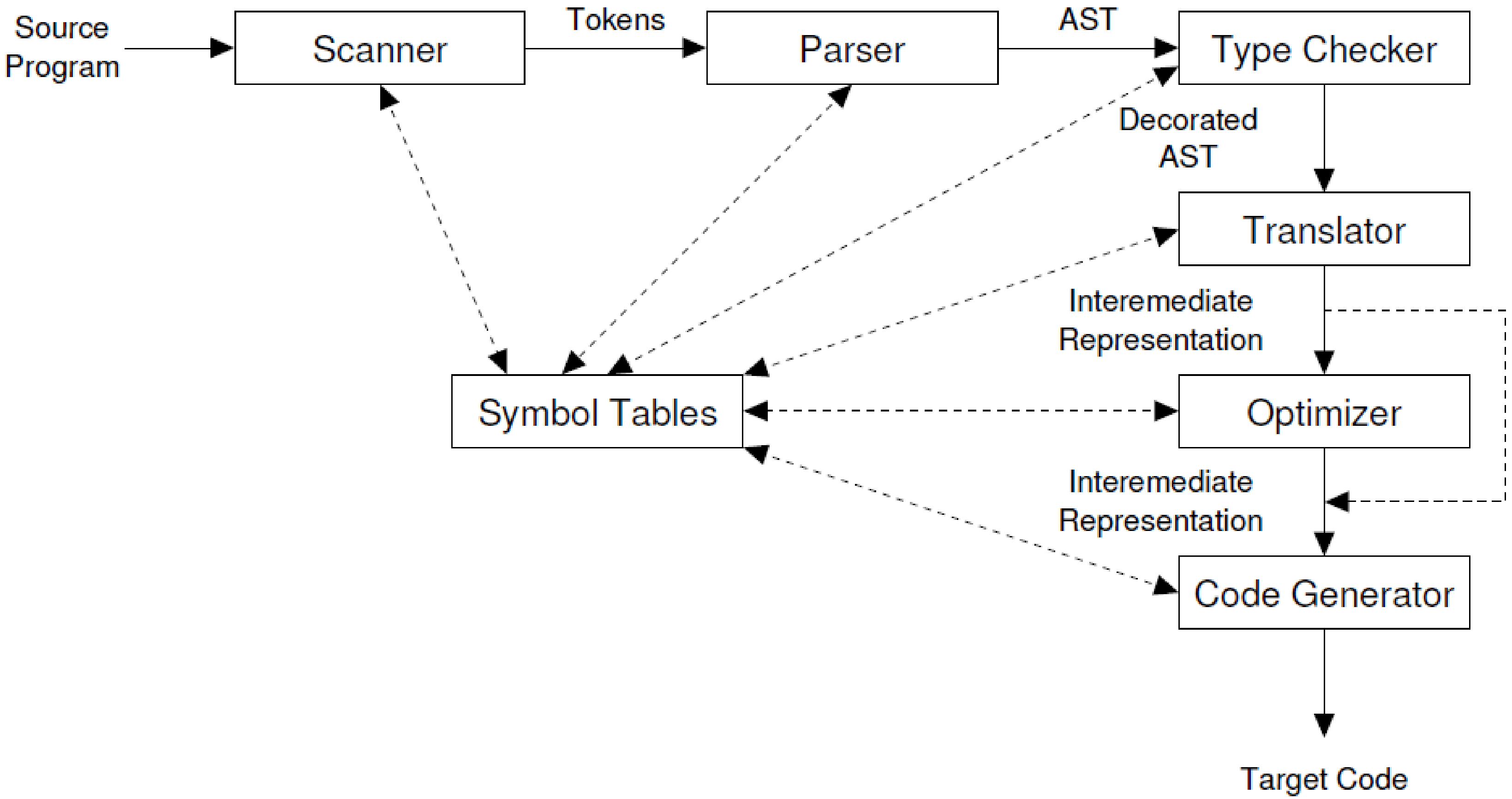


Figure 1.4: A syntax-directed compiler. AST denotes the Abstract Syntax Tree.

The compiler for the FORTRAN language:

- was implemented in 1954–1957;
- was the world's first complete 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
```

Reasons to learn compiler technology:

- understand existing languages and their implementations;
- appreciate current limitations;
- talk intelligently about language design;
- implement your very own general purpose language; and
- implement lots of useful domain-specific languages

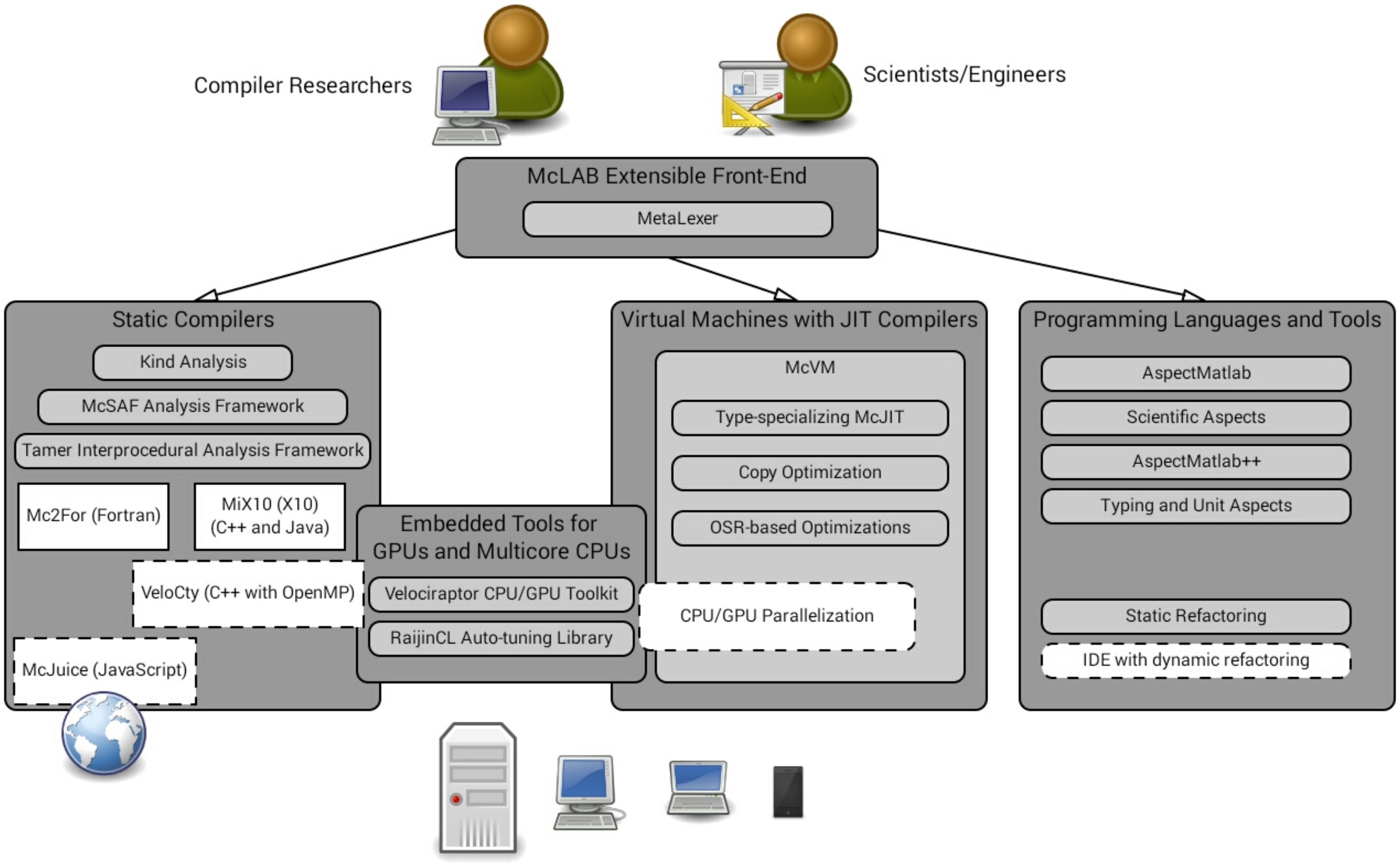
The top 10 list of reasons why we use C for compilers:

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 (almost always) 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.

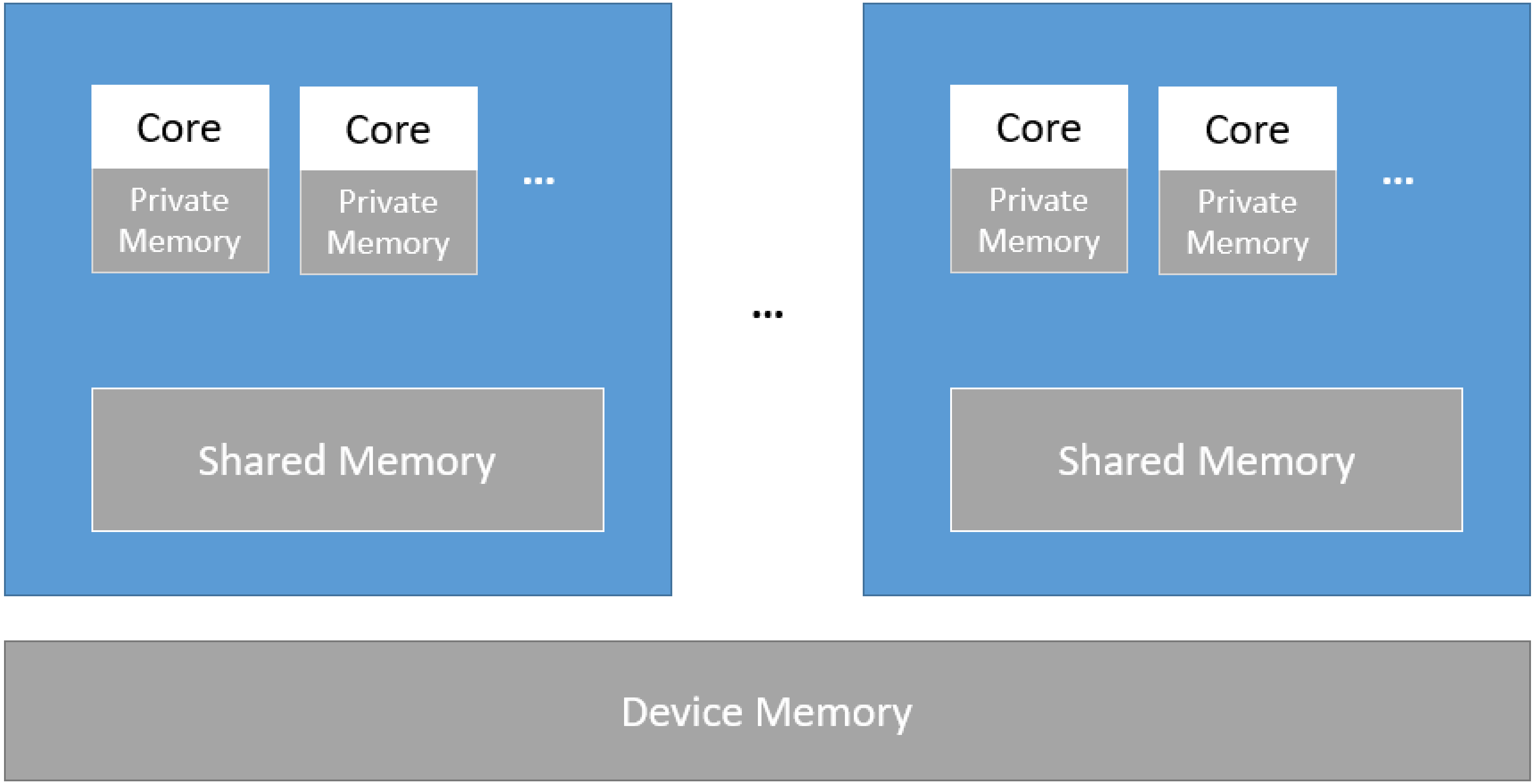
The top 10 list of reasons why we use Java for compilers:

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.

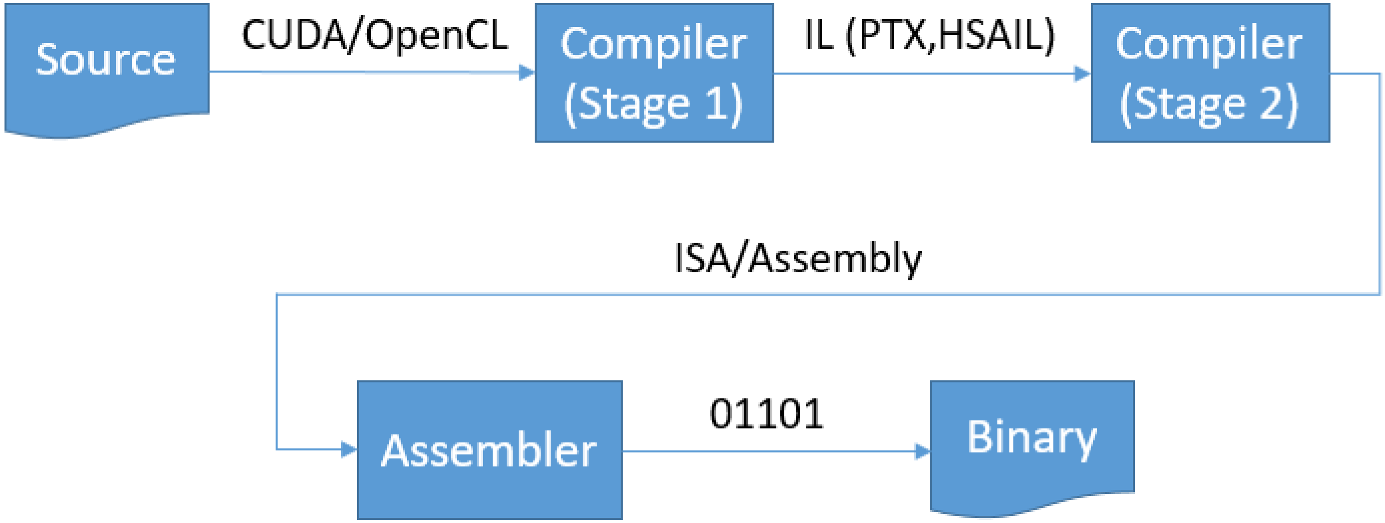
SOCS Compiler Research Group (McLAB)



My Research in Compilers:



My Research in Compilers:



Bootstrapping as illustrated in the textbook, ”Crafting a Compiler":

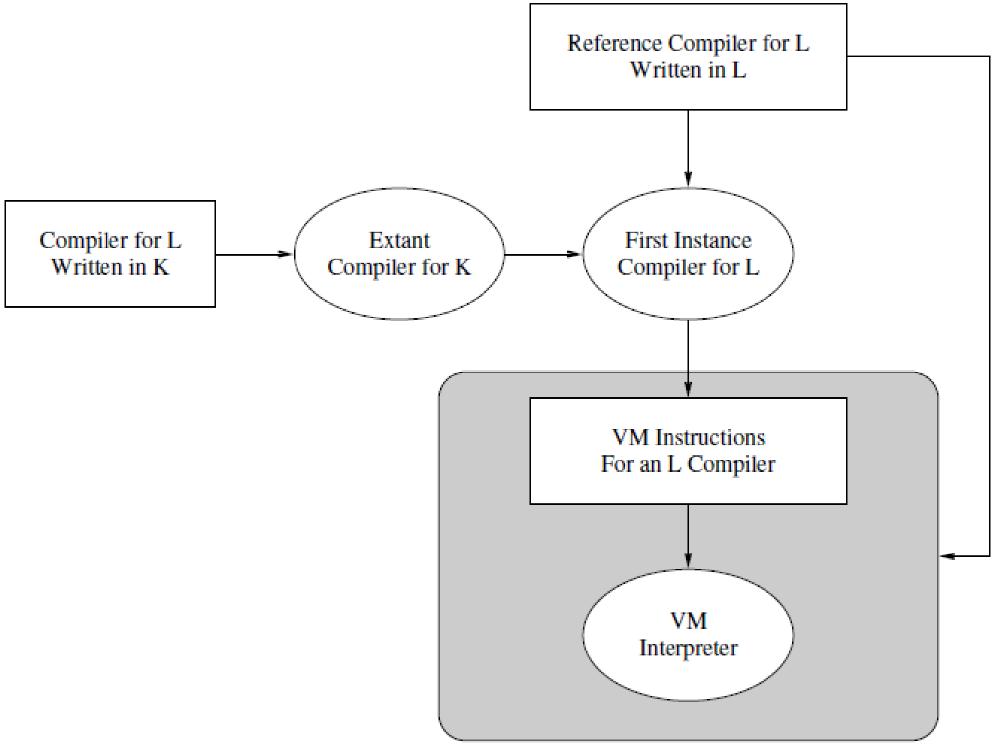
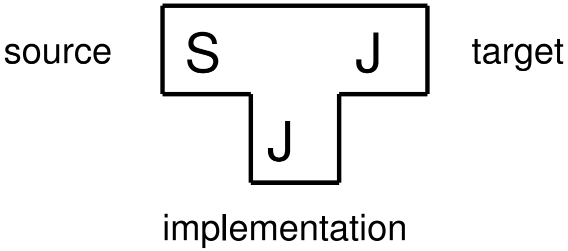


Figure 1.2: Bootstrapping a compiler that generates VM instructions. The shaded portion is a portable compiler for L that can run on any architecture supporting the VM.

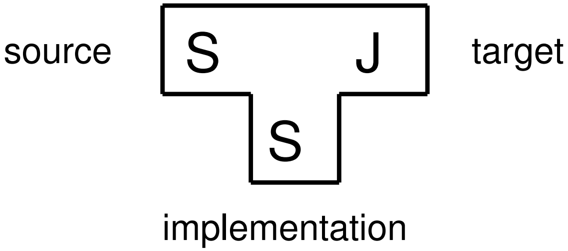
How to bootstrap a compiler (SCALA example):

- 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:



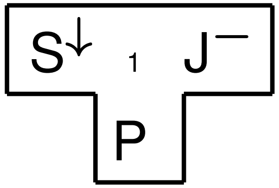
Of course, actually we like SCALA much better than Java and would therefore rather implement SCALA in itself:



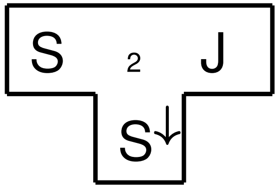
Define the following:

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

We can easily implement:

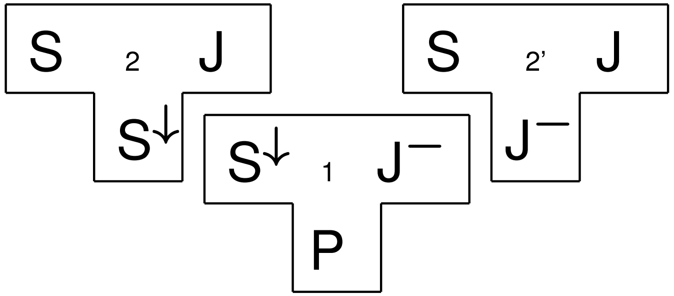


and in parallel, using $S^{\downarrow}$, we can implement:



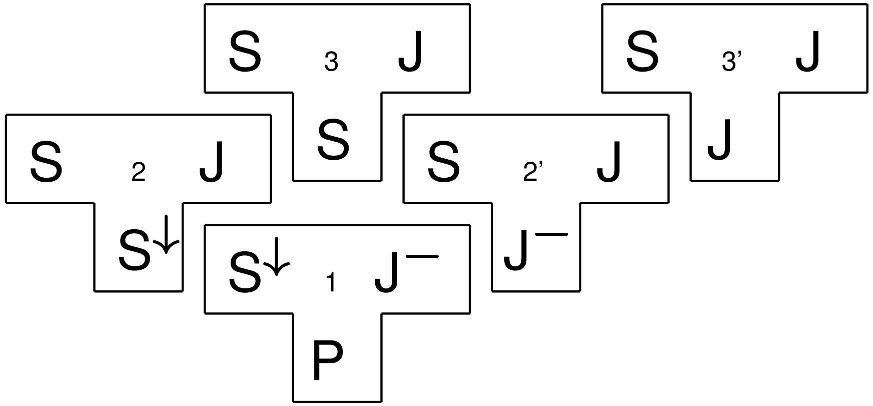
using basically our favourite language.

Combining the two compilers (compile ② with ①), we get ②':



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

A final combination (compiling ③ with ②') gives us what we want, an efficient SCALA compiler, written in SCALA, running on the Java platform ③'.


S
3
J
S
S
2
J
S↓
S↓
1
J—
P
S
2'
J
J—
S
3'
J
J