

Lecture 1: Introduction to Compilers

John Wickerson

*These slides are available at:
github.com/LangProc/langproc-lectures*

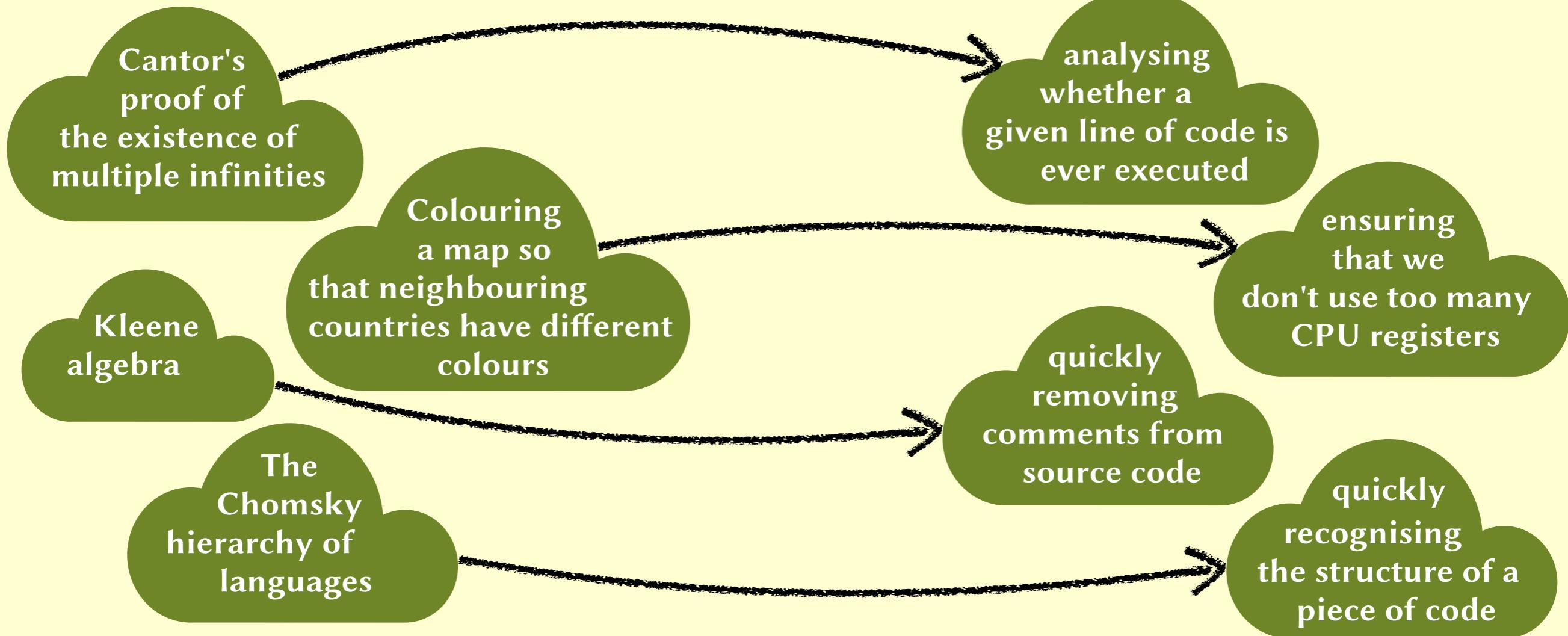
Compilers

The study of compilers is...

elegant theory

applied to

Practical Problems



About me

- I do research into **testing** and **verifying** compilers.
- I do research into how to make **EDA tools** more **reliable**.
- I'm keen to take on **UROP** students. Some recent projects:
 - 2024: finding bugs in quantum compilers
 - 2025: testing how EDA tools handle undefined behaviour

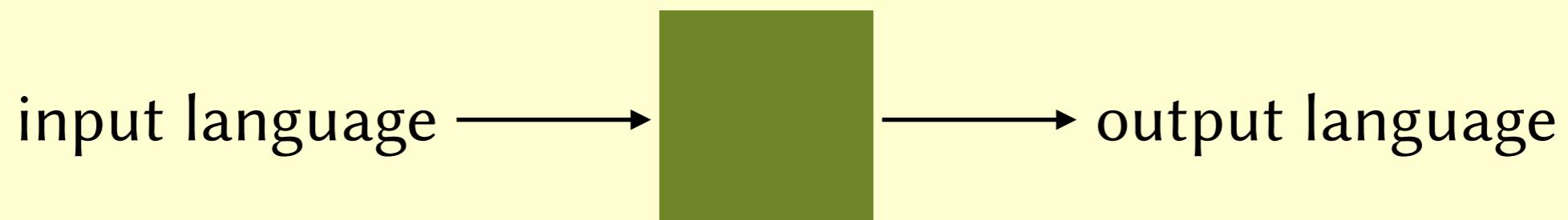
Welcome!

- What is a compiler?
- How is a compiler built?
- In which language should I implement my compiler?
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

Welcome!

- **What is a compiler?**
- How is a compiler built?
- In which language should I implement my compiler?
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

What is a compiler?



- What happens if the input is faulty?

Compilers

The screenshot shows the Compiler Explorer interface on godbolt.org. On the left, the C source code for a `square` function is displayed:

```
1 int square(int num) {
2     return num * num;
3 }
```

The code is highlighted in three colors: light green for the first two lines and light purple for the last line. On the right, the generated RISC-V assembly code is shown:

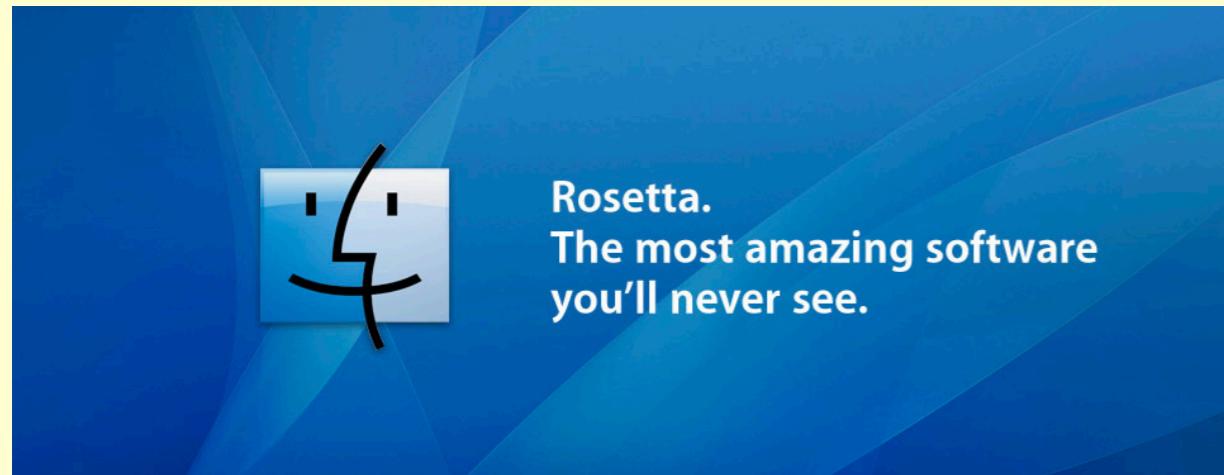
```
1 square:
2     addi    sp,sp,-32
3     sw     s0,28(sp)
4     addi    s0,sp,32
5     sw     a0,-20(s0)
6     lw     a5,-20(s0)
7     mul    a5,a5,a5
8     mv     a0,a5
9     lw     s0,28(sp)
10    addi   sp,sp,32
11    jr     ra
```

The assembly code is also color-coded by line, matching the highlighting in the C source code. The interface includes tabs for "Output (0/0)" and "Compiler License".

<https://godbolt.org>

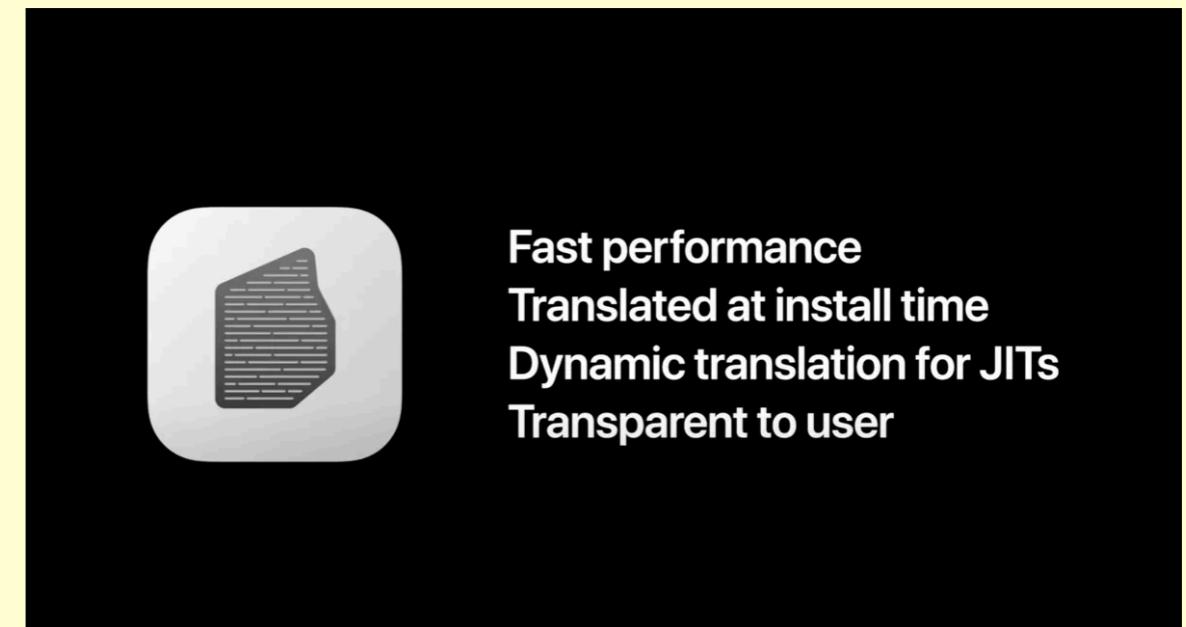
Binary translation

Rosetta v1



2006

Rosetta v2



2020

Typesetting with LaTeX

The figure shows a split-screen view of a LaTeX workflow. On the left, the Aquamacs LaTeX editor displays the source code for a paper titled "The Semantics of Transactions and Weak Memory in x86, Power, ARM, and C++". The code includes metadata like DOI, year, ISBN, conference information, and author details for Nathan Chong, Tyler Sorensen, and John Wickerson. It also defines several new commands for different types of comments (TODO, NCComment, TSComment, JWComment) with specific colors and bolding. On the right, the resulting PDF document is shown, featuring the title page with the same title and authors. The title page includes a short abstract about weak memory models and TM, followed by sections for CCS Concepts, Keywords, ACM Reference Format, and an Introduction. At the bottom of the title page, there is a note about permission to make copies and a copyright notice for PLDI'18. Below the title page, a detailed example of lock elision is presented, showing assembly-like pseudocode for two threads interacting with a shared location x .

Aquamacs File Edit Options Tools Preview LaTeX Command BCite Ref Window Help

paper.tex

```

210 \begin{document}
211
212 \title{The Semantics of Transactions and Weak Memory \\ in x86, Power,
213 ARM, and C++}
214 \renewcommand\shorttitle{The Semantics of Transactions and Weak Memory \ldots}
215
216 \setcopyright[rights retained]
217 \acmPrice{}
218 \acmDOI{10.1145/3192366.3192373}
219 \acmYear{2018}
220 \copyrightyear{2018}
221 \acmISBN{978-1-4503-5698-5/18/06}
222 \acmConference[PLDI'18]{39th ACM SIGPLAN Conference on Programming Language Design
and Implementation}{June 18--22, 2018}{Philadelphia, PA, USA}
223
224 % Reduce bullet indent
225 \setlength{\leftmargini}{\parindent}
226 \addtolength{\leftmargini}{2\labelsep}
227
228
229 \author{Nathan Chong}
230 \affiliation{ARM Ltd. \country{UK}}
231 \author{Tyler Sorensen}
232 \affiliation{Imperial College London \country{UK}}
233 \author{John Wickerson}
234 \affiliation{Imperial College London \country{UK}}
235
236 \newcommand\TODO[1]{\textcolor{red}{#1}}
237 \newcommand\NCComment[1]{\textcolor{blue!80!black}{\bf [[\ref{?}NC:]}} #1{\bf
238 ]}}
239 \newcommand\TSComment[1]{\textcolor{red!70!black}{\bf [[\ref{?}TS:]}} #1{\bf
240 ]}}
241 \newcommand\JWComment[1]{\textcolor{green!70!black}{\bf [[\ref{?}JW:]}} #1{\bf
242 ]}}
243
244 \%renewcommand\NCComment[1]{}%renewcommand\TSComment[1]{}%renewcommand\JWComment[1]{
245
246 \newcommand\totalNumberOfLitmusTests{\ref{?}}
247
248 \newcommand\isabelleqed{%

```

-:--- paper.tex 64% (228,0) Git-master (LaTeX/FPS Ref BCite Spc WordWrap)

paper.pdf (page 1 of 15)

The Semantics of Transactions and Weak Memory in x86, Power, ARM, and C++

Nathan Chong
ARM Ltd, UK

Tyler Sorensen
Imperial College London, UK

John Wickerson
Imperial College London, UK

Abstract

Weak memory models provide a complex, system-centric semantics for concurrent programs, while transactional memory (TM) provides a simpler, programmer-centric semantics. Both have been studied in detail, but their *combined* semantics is not well understood. This is problematic because such widely-used architectures and languages as x86, Power, and C++ all support TM, and all have weak memory models.

Our work aims to clarify the interplay between weak memory and TM by extending existing axiomatic weak memory models (x86, Power, ARMv8, and C++) with new rules for TM. Our formal models are backed by automated tooling that enables (1) the synthesis of tests for validating our models against existing implementations and (2) the model-checking of TM-related transformations, such as lock elision and compiling C++ transactions to hardware. A key finding is that a proposed TM extension to ARMv8 currently being considered within ARM Research is incompatible with lock elision without sacrificing portability or performance.

CCS Concepts • Theory of computation → Parallel computing models; Program semantics;

Keywords Shared Memory Concurrency, Weak Memory, Transactional Memory, Program Synthesis

ACM Reference Format:

Nathan Chong, Tyler Sorensen, and John Wickerson. 2018. The Semantics of Transactions and Weak Memory in x86, Power, ARM, and C++. In *Proceedings of 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'18)*. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3192366.3192373>

1 Introduction

Transactional memory [28] (TM) is a concurrent programming abstraction that promises scalable performance without programmer pain. The programmer gathers instructions into *transactions*, and the system guarantees that each appears to be performed entirely and instantaneously, or not

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PLDI'18, June 18–22, 2018, Philadelphia, PA, USA
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5698-5/18/06.
<https://doi.org/10.1145/3192366.3192373>

Initially: $[X0] = x = 0$

lock()	$x \leftarrow x + 2$	lock()
LDR W5,[X0]		MOV W7,#1
ADD W5,W5,#2	$x \leftarrow x + 2$	STR W7,[X0]
STR W5,[X0]		unlock()
unlock()		

Test: $x = 2$

Web browsers

John Wickerson

Projects People Publications Service Teaching Fun Contact

 John Wickerson

I'm a Lecturer in the Circuits and Systems group, which is part of the Department of Electrical and Electronic Engineering at Imperial College London.

My research aims to improve the reliability of high-performance computing with the help of formal methods.

Projects

Here is a selection of research projects I am or have been involved in.

1 a

1 a

Elements

```
<!DOCTYPE html>
<html lang="en" xml:lang="en" class="nyofzrym idc0_335">
  <head>...</head>
  ...<body> == $0
    <><nav class="navbar navbar-expand-sm navbar-dark bg-dark" ark">...</nav> flex
    <><div class="jumbotron jumbotron-fluid" id="jumbotron">...</div>
      <!-- jumbotron -->
    <><div class="container">...</div>
    <><div class="container">...</div>
      <!-- container -->
    <><div id="contactbox" class="container-fluid">...</div>
      <!-- contactbox -->
    <><nav class="navbar navbar-dark bg-dark">...</nav>
      flex
      <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965Dz0rT7abK41JSQtQIAqVgRVzpbzo5smXKp4YfRvH+8abTE1Pi6jizo" crossorigin="anonymous"></script>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-U02eT0CpHqdSJQ6hJty5KVphtPhzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
      <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
    <><script>...</script>
```

html.nyofzrym.idc0_335 body

Styles Computed Layout Event Listeners

Filter :hov .cls + □

element.style {

}

body {

background-color: #FFFFCC;

font-family: 'Linux Libertine', Times, serif;

Google Translate

A screenshot of the Google Translate website. The title bar shows "Google Translate". The URL in the address bar is "translate.google.co.uk/?sl=en&tl=fr&text=I%20love%20co...". The main content area displays the text "I love compilers!" in English on the left and its French translation "J'adore les compilateurs !" on the right. The interface includes language detection, document mode, and various sharing and feedback options.

Google Translate

Text Documents

DETECT LANGUAGE FRENCH ENGLISH GREEK ENGLISH FRENCH

I love compilers! J'adore les compilateurs !

Send feedback

Who made the first compiler?



Corrado Böhm
(1923–2017)

- Described a compiler in his 1951 PhD thesis, but didn't implement it.

Who made the first compiler?



Corrado Böhm
(1923–2017)



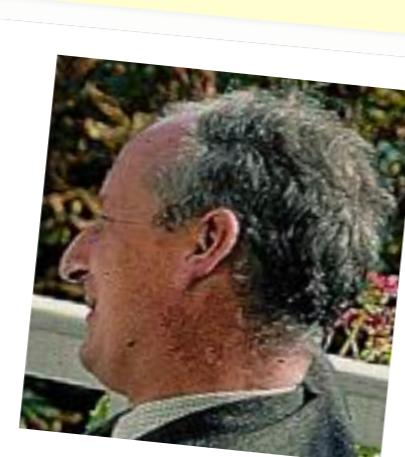
Alick Glennie
(1925–2003)

- Designed AUTOCODE in 1951, which translated human-readable instructions into machine code for the Manchester Mark I.
- Nowadays we'd probably call that an "assembler".

Who made the first compiler?



Corrado Böhm
(1923–2017)



Alick Glennie
(1925–2003)



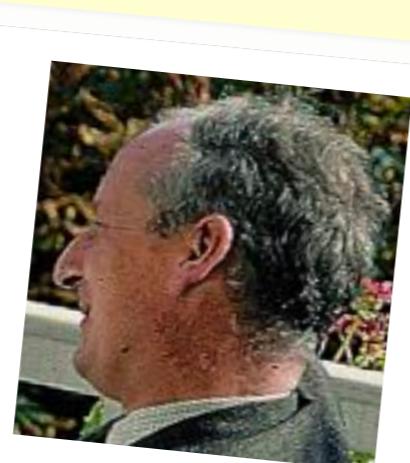
Grace Hopper
(1906–1992)

- Designed the "A-0" compiler in 1952.
- Replaced a series of function calls (identified numerically) with their corresponding definitions.
- First use of the term "compiler", but nowadays we'd call it a "linker".
- Many were skeptical about computers writing their own programs!

Who made the first compiler?



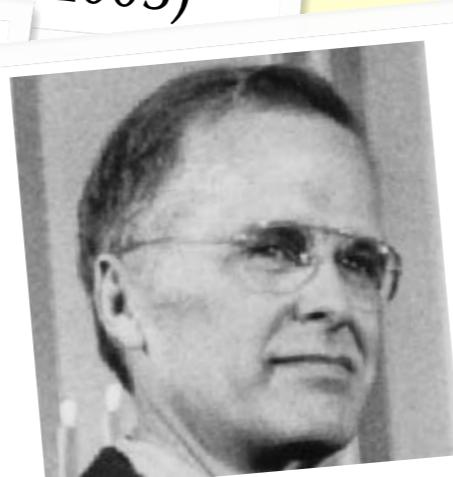
Corrado Böhm
(1923–2017)



Alick Glennie
(1925–2003)



Grace Hopper
(1906–1992)



John Backus
(1924–2007)

- Led the development of the first high-level language, FORTRAN, and its compiler. About 20000 lines of code, 3 years, 6 people.
- "Produced code of such efficiency that its output would startle the programmers who studied it."
- Introduced "basic blocks" and "symbolic registers".
- Advancing compiler technology allows programmers to work at ever higher levels of abstraction (e.g. **register** keyword in C)

Welcome!



What is a compiler?

- How is a compiler built?
- In which language should I implement my compiler?
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

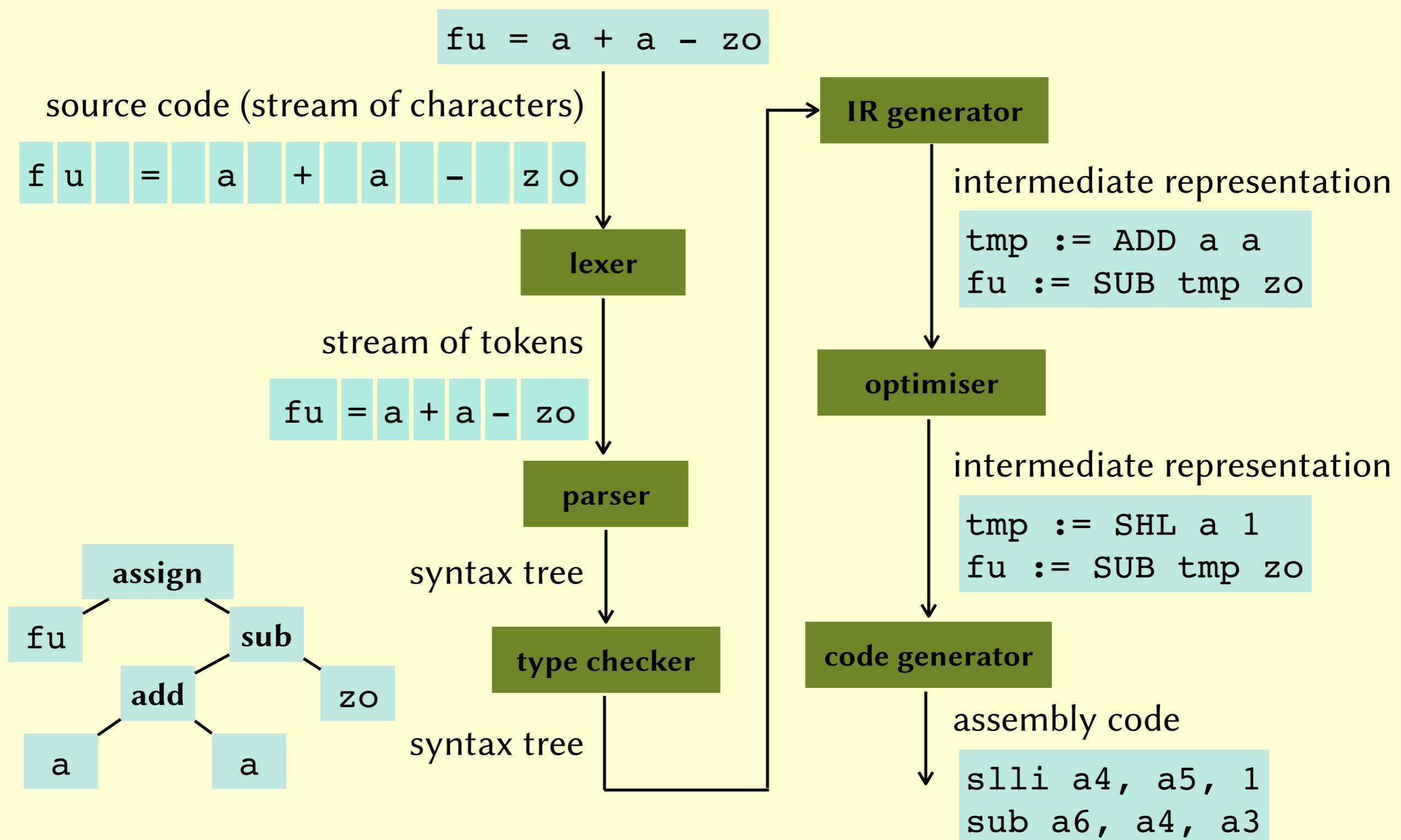
Welcome!



What is a compiler?

- **How is a compiler built?**
- In which language should I implement my compiler?
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

Anatomy of a compiler

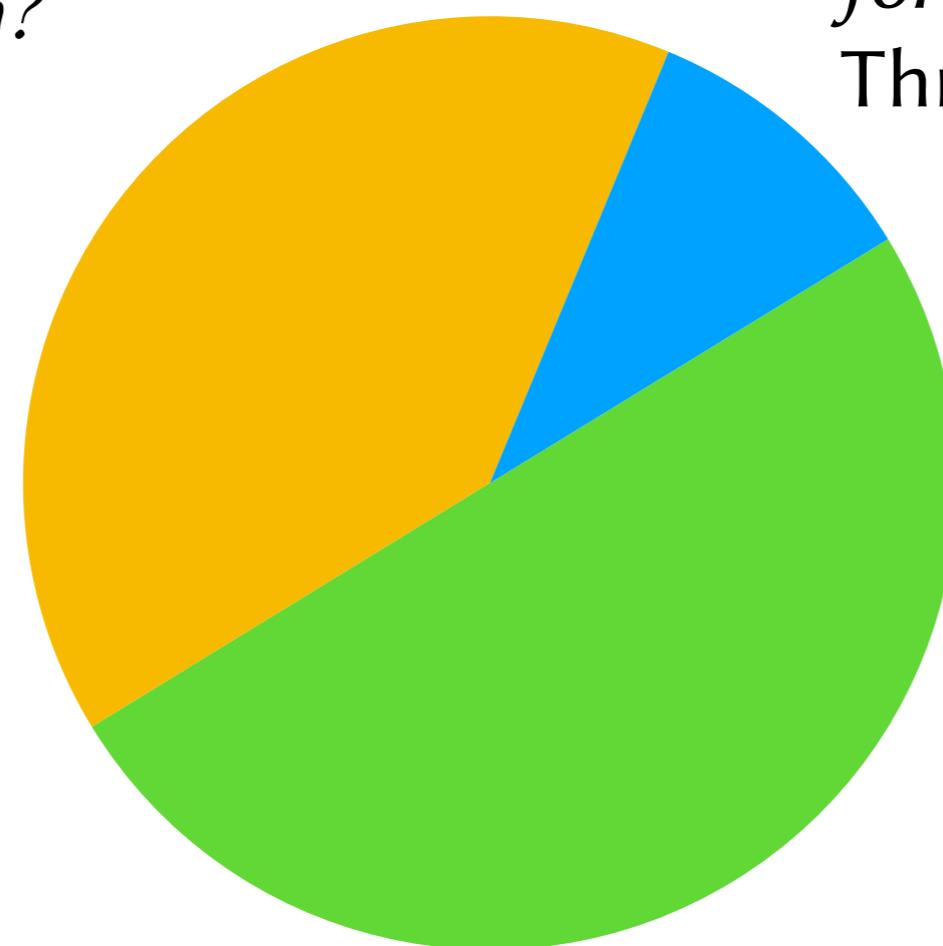


Aims and assessment

Knowledge

*Do you understand the principles
of compiler construction?*

Exam, Summer term.



Application

Can you work in pairs to build a working C compiler?
Coursework, due end of term.

Skills

*Can you use standard tools
for compiler construction?*

Three labs this term.

(Tentative) Schedule

Week	Mon 1100-1300 (403)	Thu 1100-1300 (usually 508)	Fri 1300-1400 (usually 408)
1	12 Jan <i>(No lab)</i>	Introduction	Lexing
2	19 Jan Lexing exercise due: Fri 30 Jan 1500	More lexing	Parsing
3	26 Jan	More parsing	Type checking
4	02 Feb Parsing exercise due: Fri 06 Feb 1500	Interm. representation	Code generation
5	09 Feb Code gen exercise due: Fri 13 Feb 1500	C-to-RISCV compilation	Data flow analysis
6	16 Feb	Industrial guest? <i>(no lecture)</i>	<i>(no lecture)</i>
7	23 Feb	<i>(no lecture)</i>	Memory management
8	02 Mar Compiler project due: Fri 20 Mar 1500	Static analysis	Class quiz 1 (lexing?)
9	09 Mar	Compiler correctness	Class quiz 2 (parsing?)
10	16 Mar	Compiling concurrency	Class quiz 3 (code gen?)

Teaching assistants

litem.org/us/courses/88178



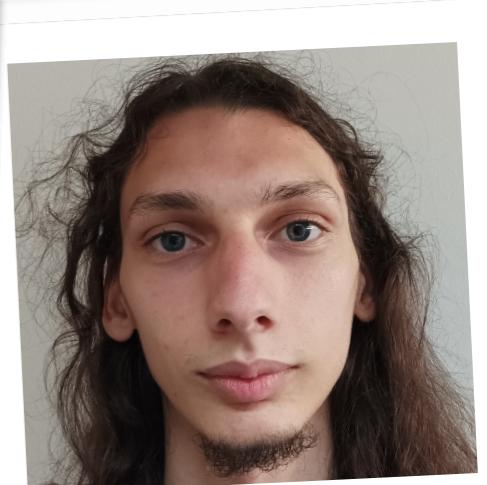
Michalis Pardalos



Ryan Voecks



Dimitris
Alexopoulos



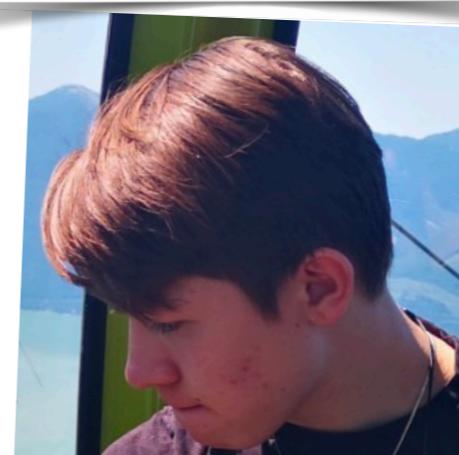
Quentin Corradi



Ilan Iwumbwe



Benny Liu



Pierce
Wiegerling

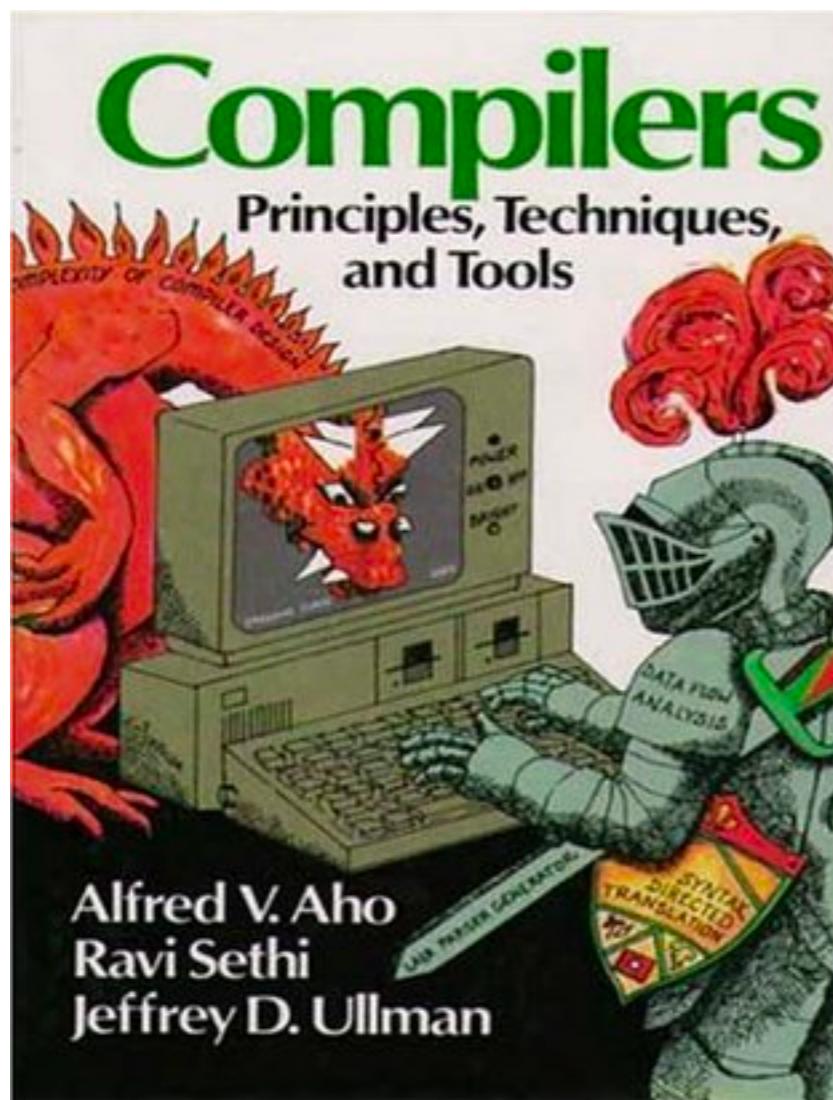


Filip Wojcicki

Other support

- Ed discussion forum: <https://edstem.org/us/courses/88178>
- Github issues

Recommended Reading



- Aho, Sethi, and Ullman: *Compilers: Principles, Techniques, and Tools* (a.k.a. *The Dragon Book*).

Mainly Chapter 2.

Welcome!



What is a compiler?



How is a compiler built?

- In which language should I implement my compiler?
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

Welcome!



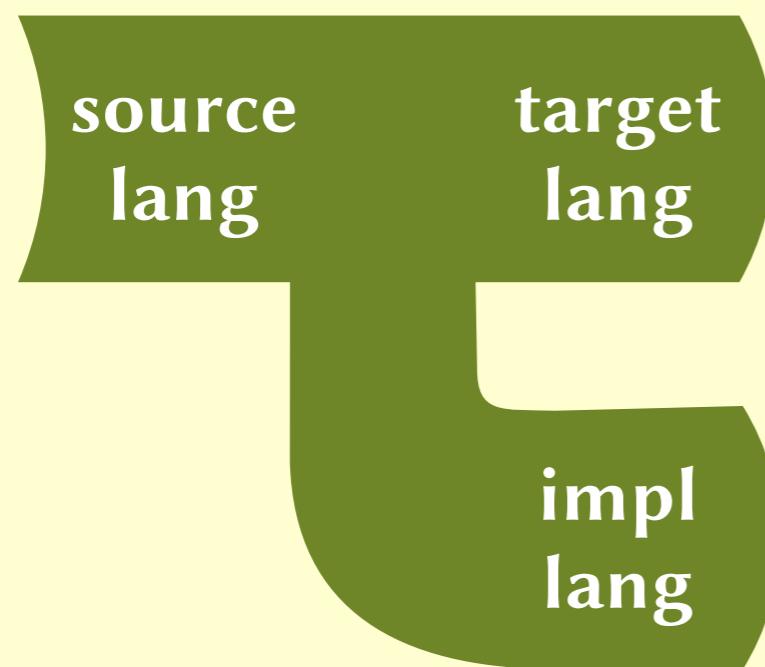
What is a compiler?



How is a compiler built?

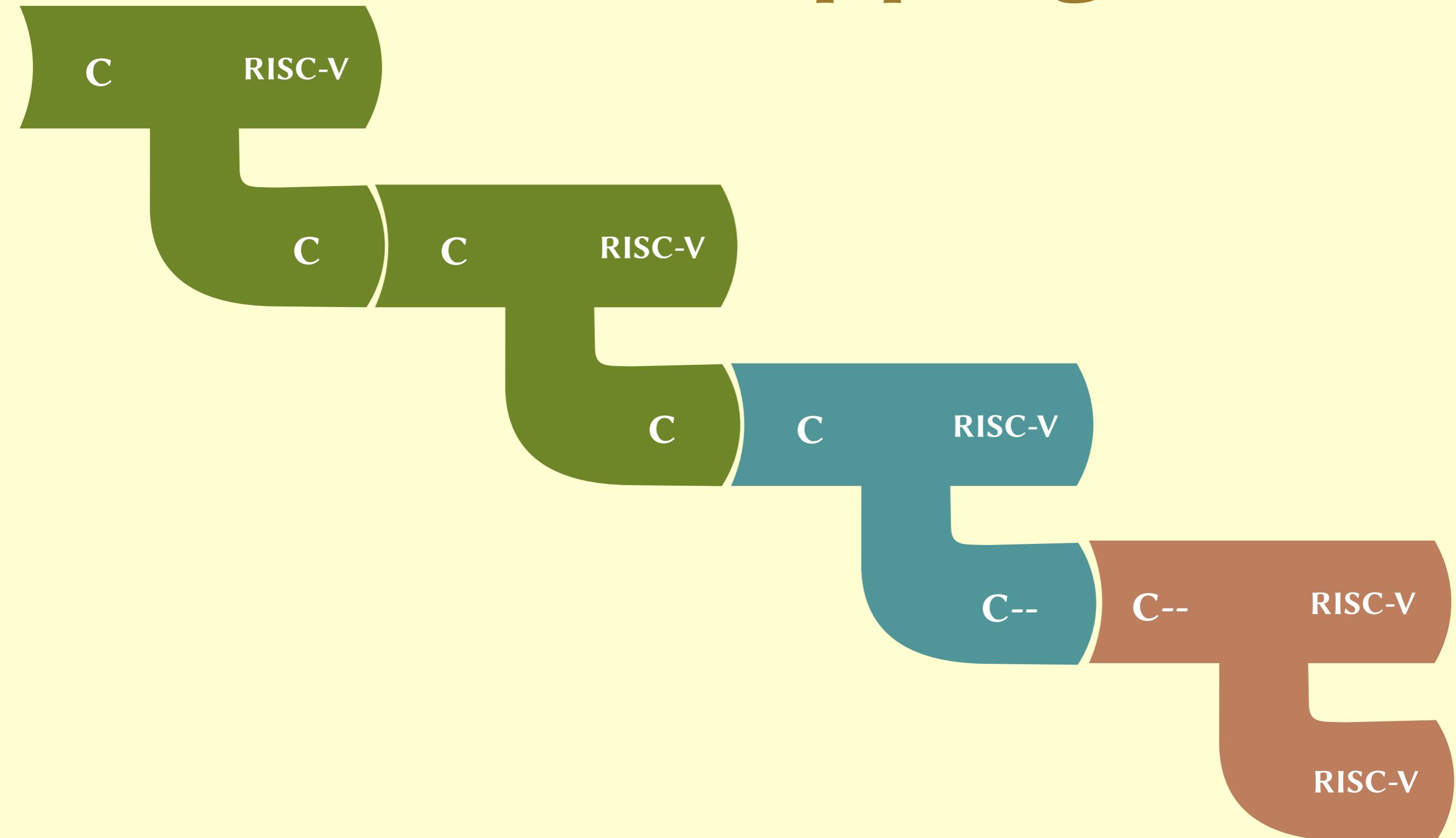
- **In which language should I implement my compiler?**
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

Which language?

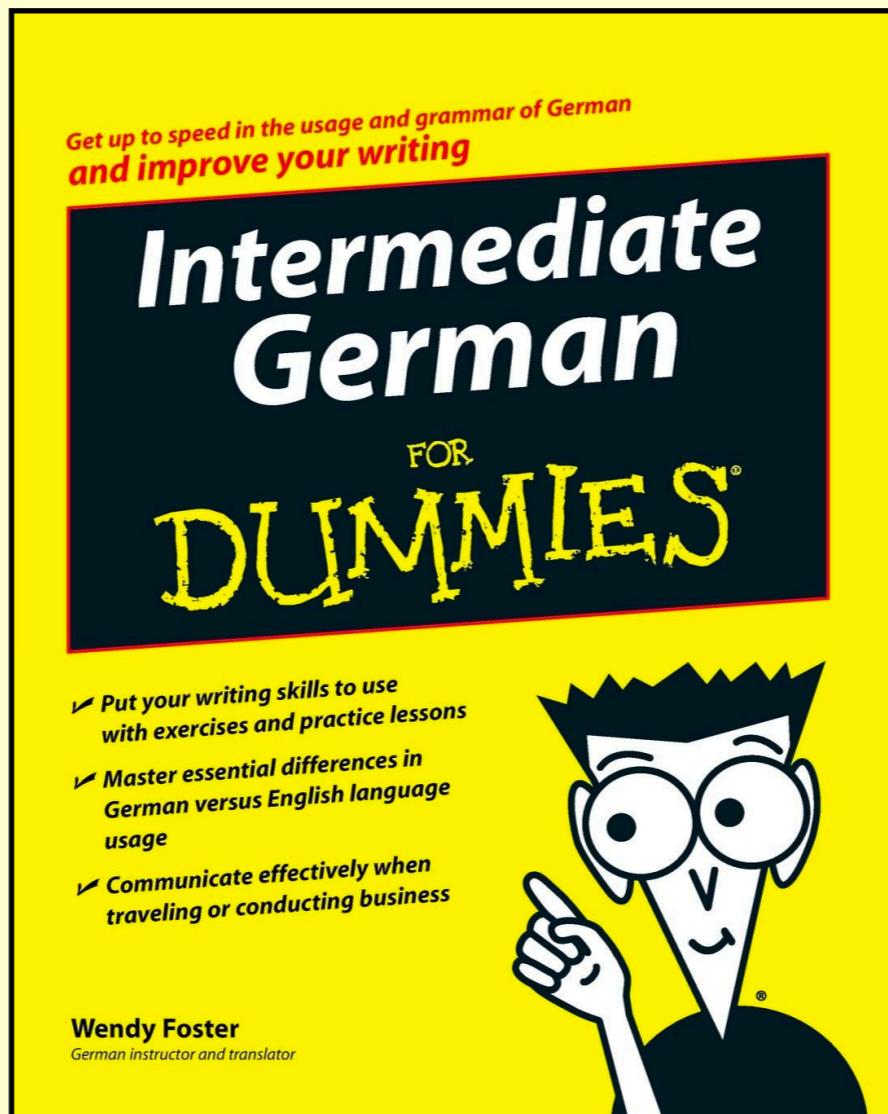
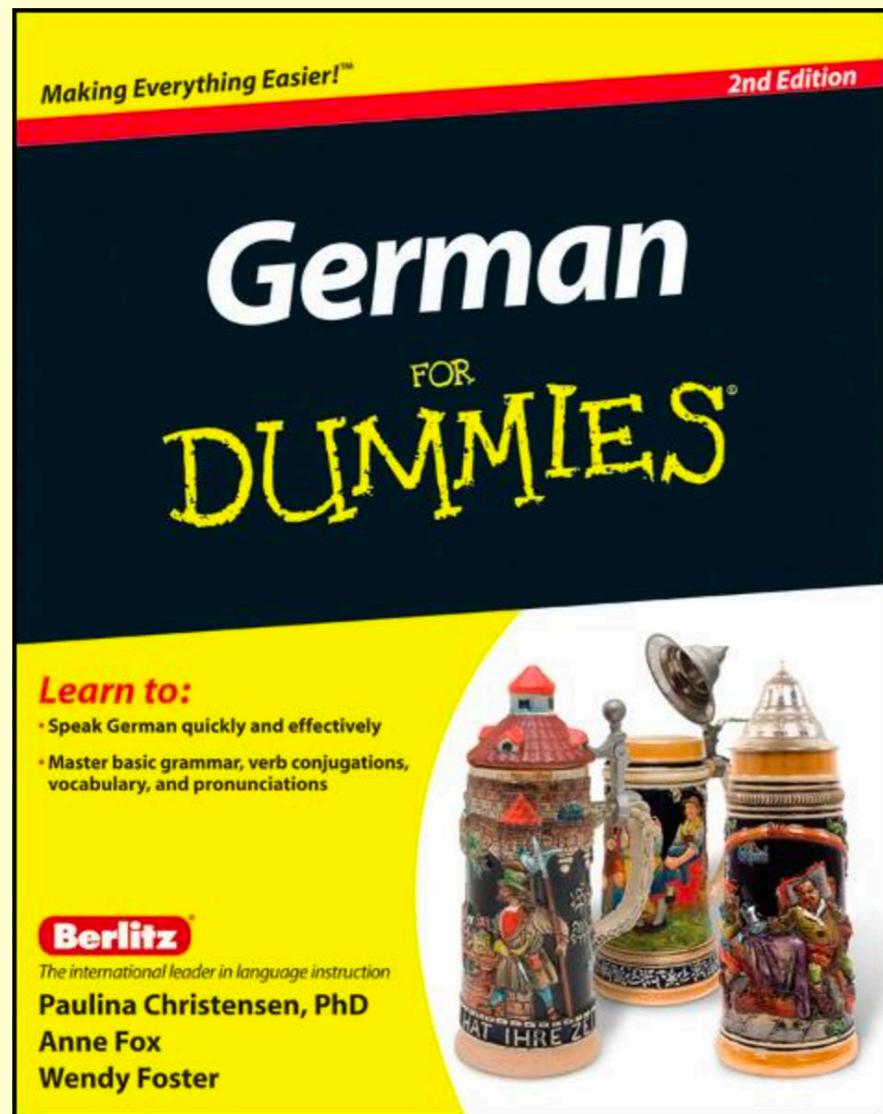


- If $\text{source}=\text{impl}$ then compiler is called *self-hosting*.
- What if you've written the first compiler for a new language?

Bootstrapping



Analogy



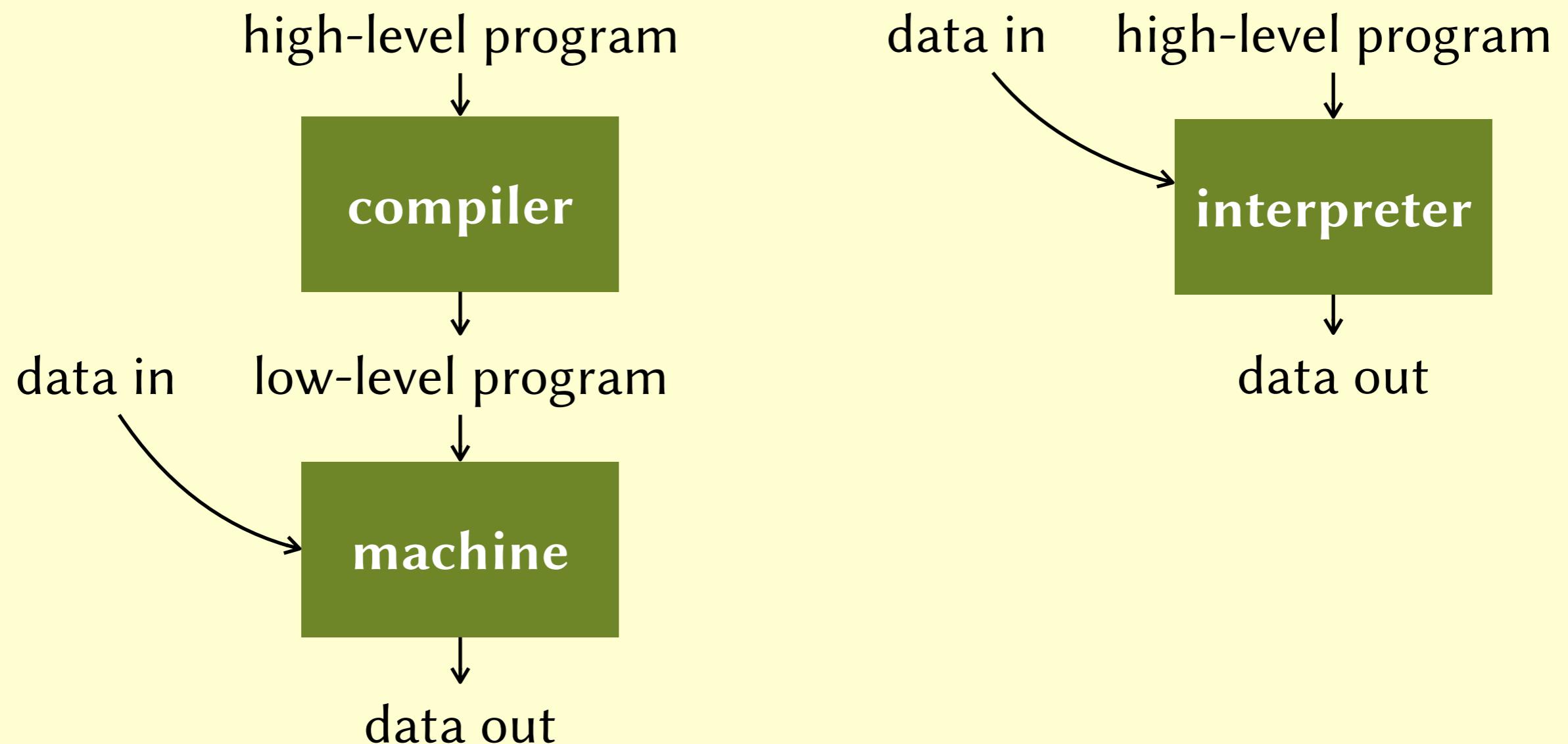
Welcome!

- ✓ What is a compiler?
- ✓ How is a compiler built?
- ✓ In which language should I implement my compiler?
 - To compile or to interpret?
 - What makes a good compiler?
 - What is the programmer/compiler contract?

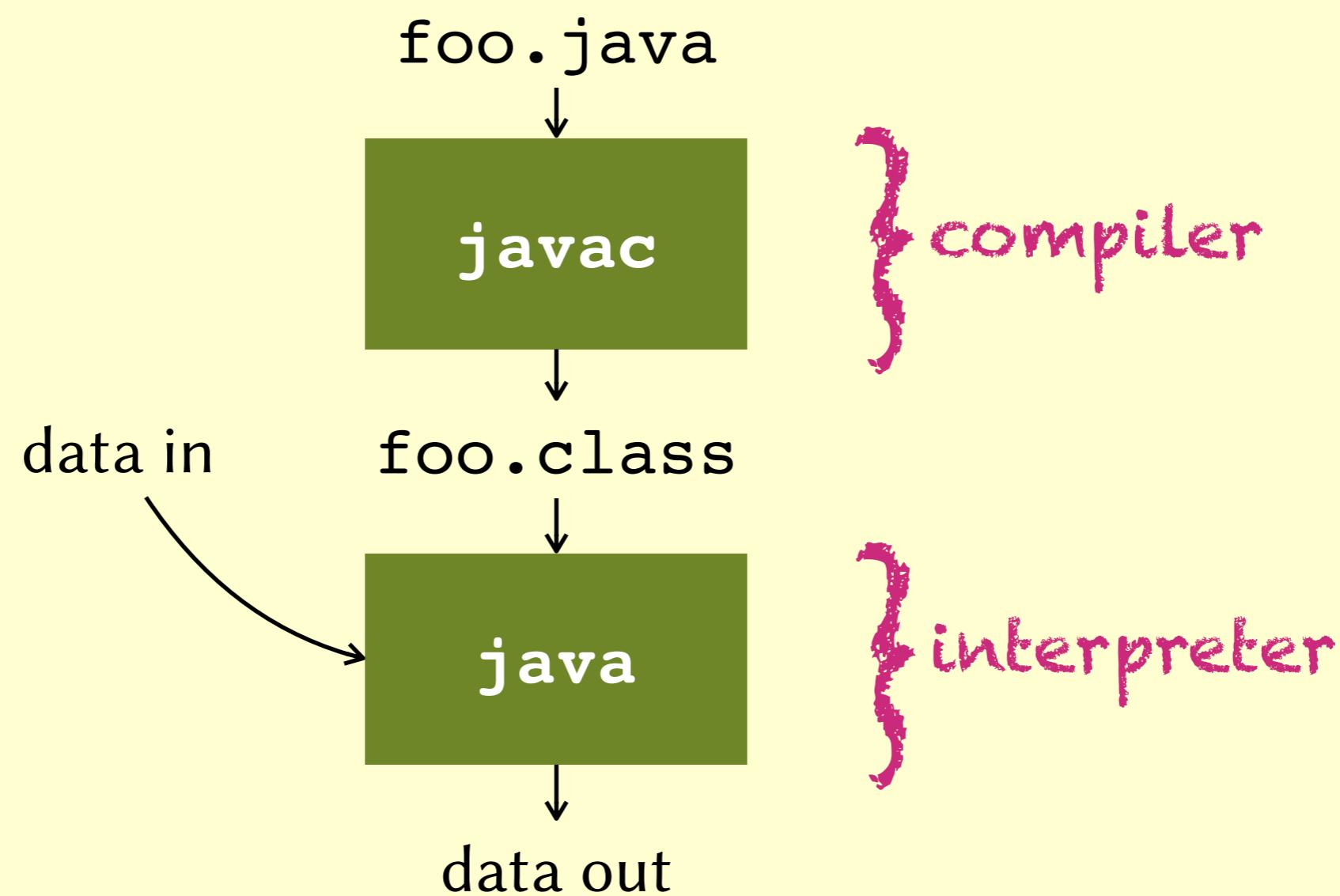
Welcome!

- ✓ What is a compiler?
- ✓ How is a compiler built?
- ✓ In which language should I implement my compiler?
- **To compile or to interpret?**
- What makes a good compiler?
- What is the programmer/compiler contract?

Compilers vs. Interpreters



Java (an unusual case)



Welcome!

- ✓ What is a compiler?
- ✓ How is a compiler built?
- ✓ In which language should I implement my compiler?
- ✓ To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

Welcome!

- ✓ What is a compiler?
 - ✓ How is a compiler built?
 - ✓ In which language should I implement my compiler?
 - ✓ To compile or to interpret?
-
- **What makes a good compiler?**
 - What is the programmer/compiler contract?

Assessing compilers

- Given **valid input**, does it always produce an output program?
- Is the output program always **correct**?
- How **large** is the output program?
- Given **invalid input**, does it always give feedback?
- Is this feedback **useful** to the user of the compiler?
- Does the compiler finish **quickly**?
- Is the compiler **deterministic**?

Welcome!

- ✓ What is a compiler?
 - ✓ How is a compiler built?
 - ✓ In which language should I implement my compiler?
 - ✓ To compile or to interpret?
 - ✓ What makes a good compiler?
-
- What is the programmer/compiler contract?

Welcome!

- ✓ What is a compiler?
- ✓ How is a compiler built?
- ✓ In which language should I implement my compiler?
- ✓ To compile or to interpret?
- ✓ What makes a good compiler?
- **What is the programmer/compiler contract?**

What should this do?

If a side effect on a scalar object is unsequenced relative to either another side effect on the same scalar object or a value computation using the value of the same scalar object, the behavior is undefined. *[C++ language standard, 2011]*

or, more simply:

Whenever there are two different accesses to the same variable, at least one of which is a write, then they must have a sequence-point between them.

```
int main() {
    int x = 2;
    printf ("%d\n", x++ + ++x);
}
```

What should this do?

```
int foo (int x) {  
    return (x+1 > x);  
}
```

```
int foo (int x) {  
    return 1;  
}
```

- `int` is short for `signed int`. Signed variables are *not allowed* to overflow.
- Case (`x == INT_MAX`): Addition will overflow. Do anything!
- Case (`x != INT_MAX`): Addition won't overflow. Must return 1.
- So, just return 1 in both cases!

Welcome!

- ✓ What is a compiler?
- ✓ How is a compiler built?
- ✓ In which language should I implement my compiler?
- ✓ To compile or to interpret?
- ✓ What makes a good compiler?
- ✓ What is the programmer/compiler contract?