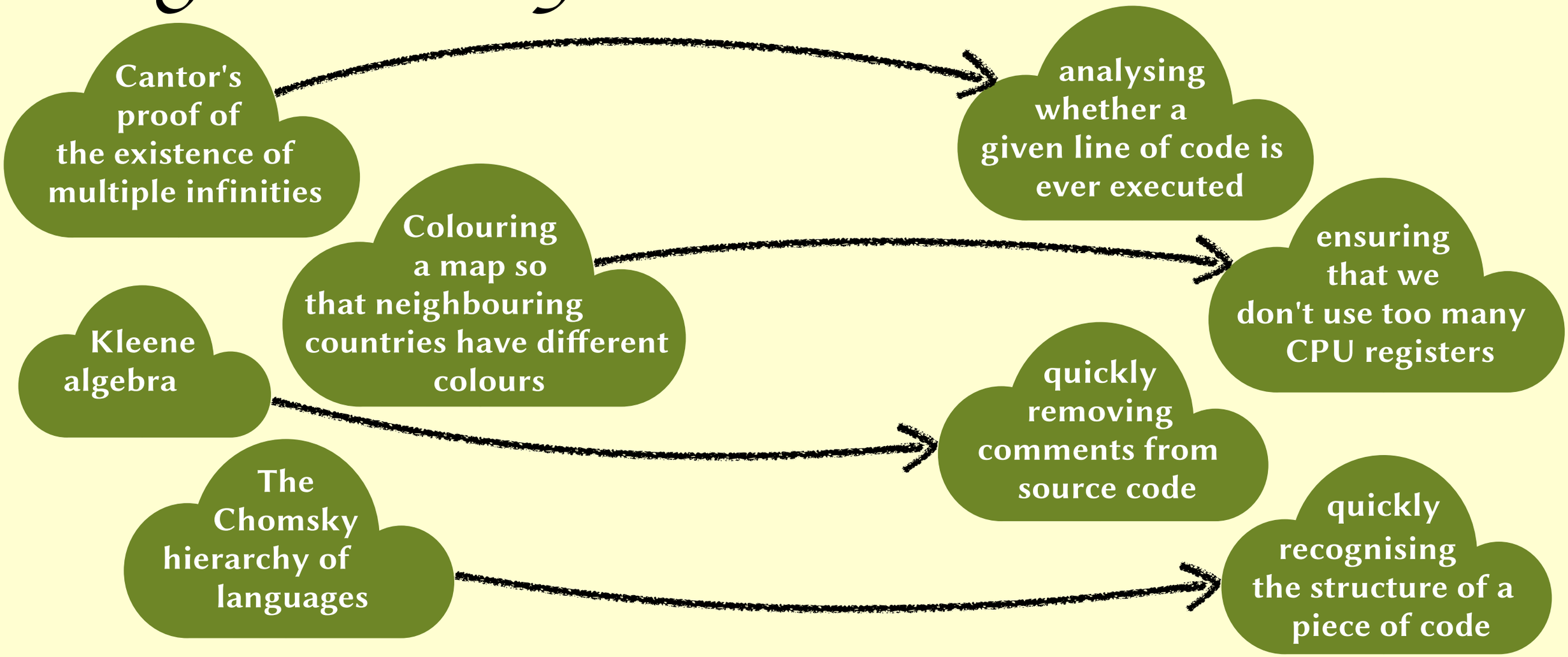
Lecture 1:
Introduction to Compilers

John Wickerson

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 **high-level synthesis**.
- I'm keen to take on **UROP** students.

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?

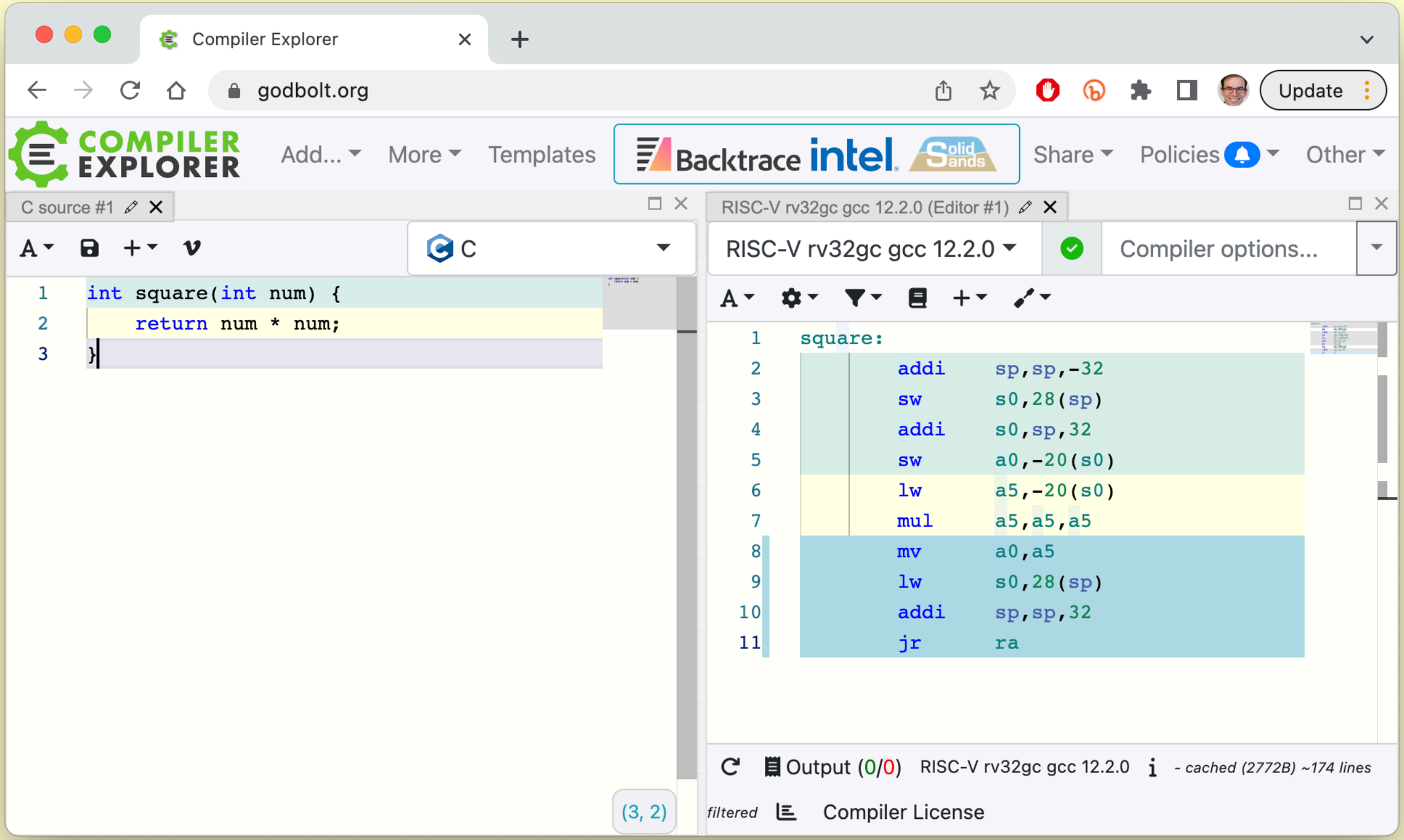
Welcome!

- **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?

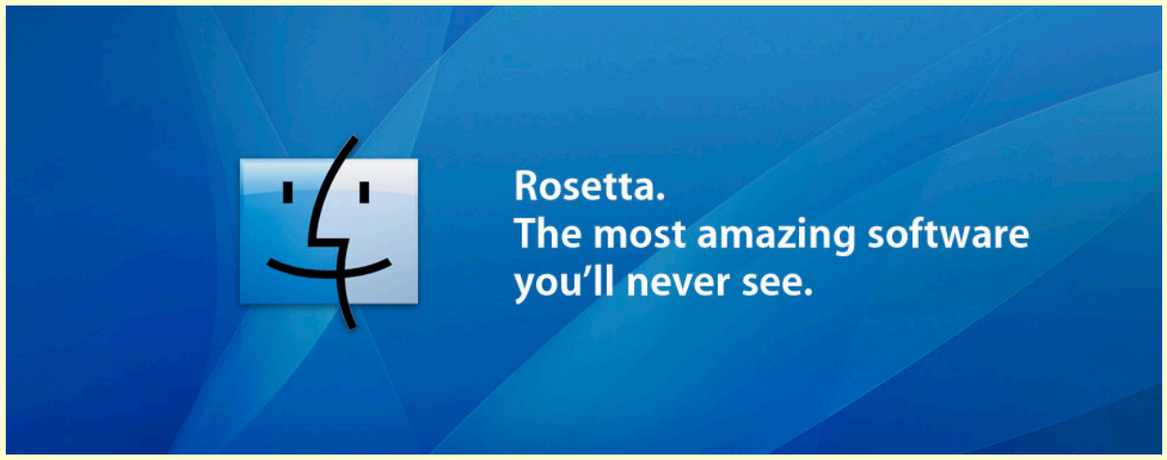
Compilers



https://godbolt.org

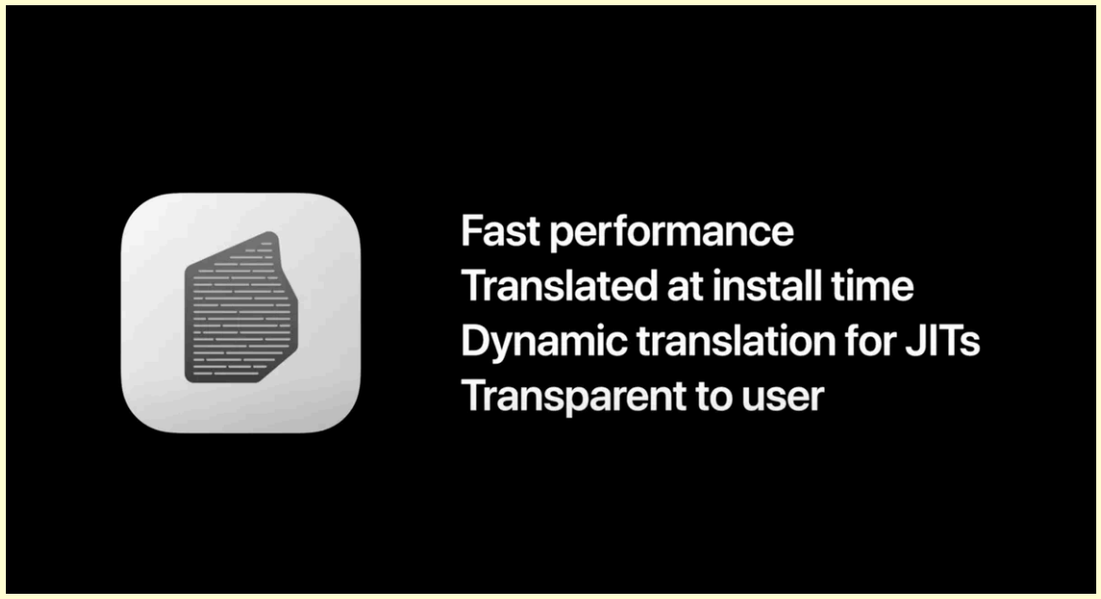
Binary translation

Rosetta v1



2006

Rosetta v2



2020

Typesetting with LaTeX

Aquamacs File Edit Options Tools Preview LaTeX Command BCite Ref Window Help — 100% — Thu 13:56

paper.tex

```
\begin{document}

\title{The Semantics of Transactions and Weak Memory \\ in x86, Power,
ARM, and C++}
\renewcommand\shorttitle{The Semantics of Transactions and Weak Memory \ldots}

\setcopyright{rightsretained}
\acmPrice{}
\acmDOI{10.1145/3192366.3192373}
\acmYear{2018}
\copyrightyear{2018}
\acmISBN{978-1-4503-5698-5/18/06}
\acmConference[PLDI'18]{39th ACM SIGPLAN Conference on Programming Language Design
and Implementation}{June 18--22, 2018}{Philadelphia, PA, USA}

% Reduce bullet indent
\setlength\leftmargini{\parindent}
\addtolength\leftmargini{2\labelsep}


\author{Nathan Chong}
\affiliation{ARM Ltd. \country{UK}}
\author{Tyler Sorensen}
\affiliation{Imperial College London \country{UK}}
\author{John Wickerson}
\affiliation{Imperial College London \country{UK}}

\newcommand\TODO[1]{\textcolor{red}{#1}}
\newcommand\NCComment[1]{\textcolor{blue!80!black}{{\bf [[\ref{?}NC:} #1{\bf
]]}}}
\newcommand\TSComment[1]{\textcolor{red!70!black}{{\bf [[\ref{?}TS:} #1{\bf
]]}}}
\newcommand\JWComment[1]{\textcolor{green!70!black}{{\bf [[\ref{?}JW:} #1{\bf
]]}}}

%\renewcommand\NCComment[1]{}\renewcommand\TSComment[1]{}\renewcommand\JWComment[1]{
}

\newcommand\totalNumberOfLitmusTests{\ref{?}}

\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 at all. To achieve this, a typical TM system tracks each transaction's memory accesses, and if it detects a conflict (i.e., another thread concurrently accessing the same location, at least one access being a write), resolves it by aborting the transaction and rolling back its changes.

1.1 Motivating Example: Lock Elision in ARMv8

One important use-case of TM is *lock elision* [22, 46], in which the lock/unlock methods of a mutex are skipped and the critical region (CR) is instead executed speculatively inside a transaction. If two CRs do not conflict, this method allows them to be executed simultaneously, rather than serially. If a conflict is detected, the transaction is rolled back and the system resorts to acquiring the mutex as usual.

Lock elision may not apply to all CRs, so an implementation must ensure mutual exclusion between transactional and non-transactional CRs. This is typically done by starting each transactional CR with a read of the lock variable (and self-aborting if it is taken) [31, §16.2.1]. If the mutex is subsequently acquired by a non-transactional CR then the TM system will detect a conflict on the lock variable and abort the transactional CR.

Thus, reasoning about lock elision requires a concurrency model that accounts for both modes, transactional and non-transactional. In particular, systems with memory models weaker than *sequential consistency* (SC) [39] must ensure that the non-transactional lock/unlock methods synchronise sufficiently with transactions to provide mutual exclusion.

In their seminal paper introducing lock elision, Rajwar and Goodman argued that "correctness is guaranteed without any dependence on memory ordering" [46, §9]. In fact, by drawing on a decade of weak memory formalisations [5, 24, 45] and by extending state-of-the-art tools [4, 40, 55], we show it is straightforward to contradict this claim *automatically*.

Example 1.1 (Lock elision is unsound under ARMv8). Consider the program below, in which two threads use CRs to update a shared location x.

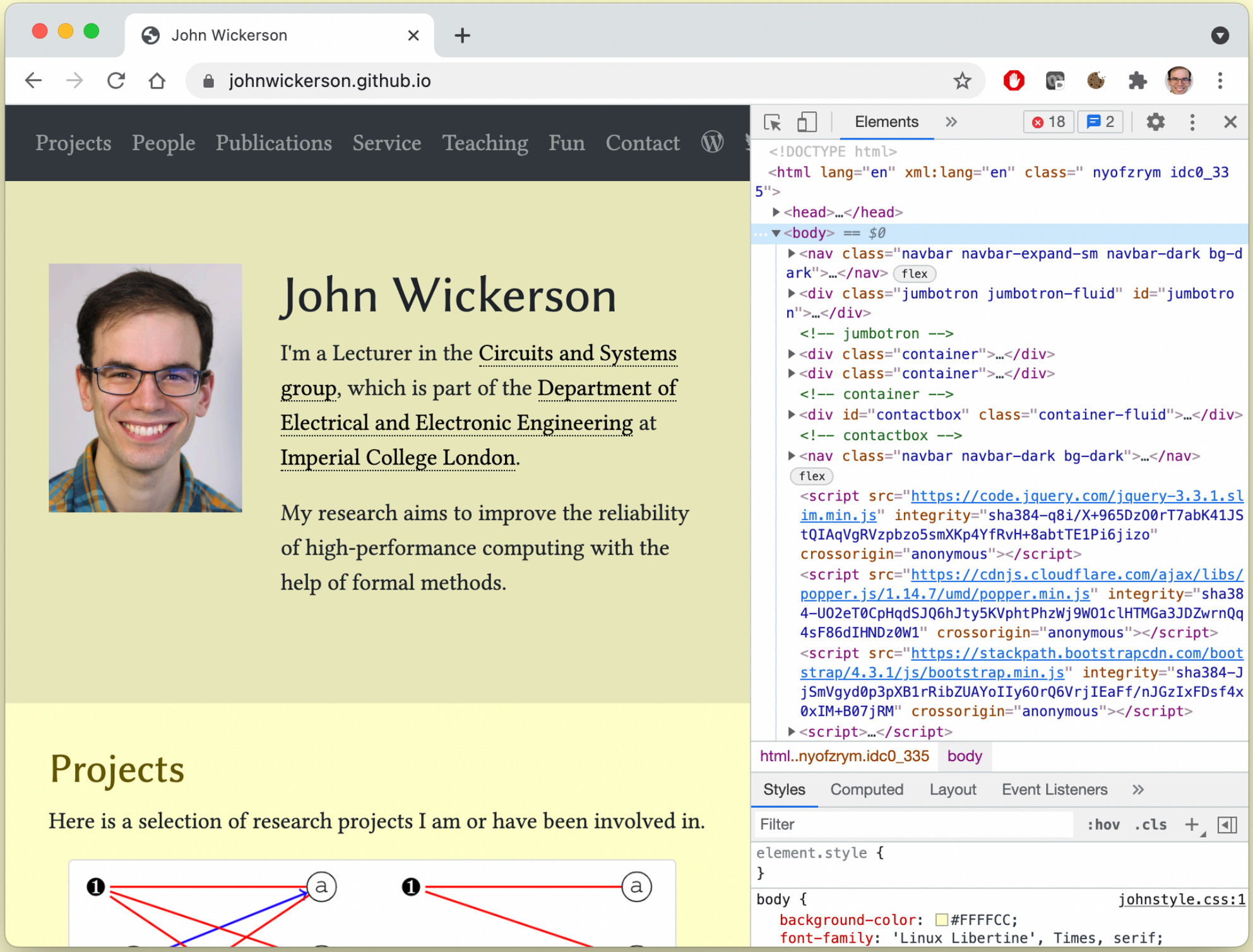
Initially: [X0] = x = 0	
lock()	lock()
LDR W5,[X0] } $x \leftarrow x+2$	MOV W7,#1 } $x \leftarrow 1$
ADD W5,W5,#2	STR W7,[X0]
STR W5,[X0]	unlock()
unlock()	
Test: x = 2	

PLDI'18, June 18–22, 2018, Philadelphia, PA, USA

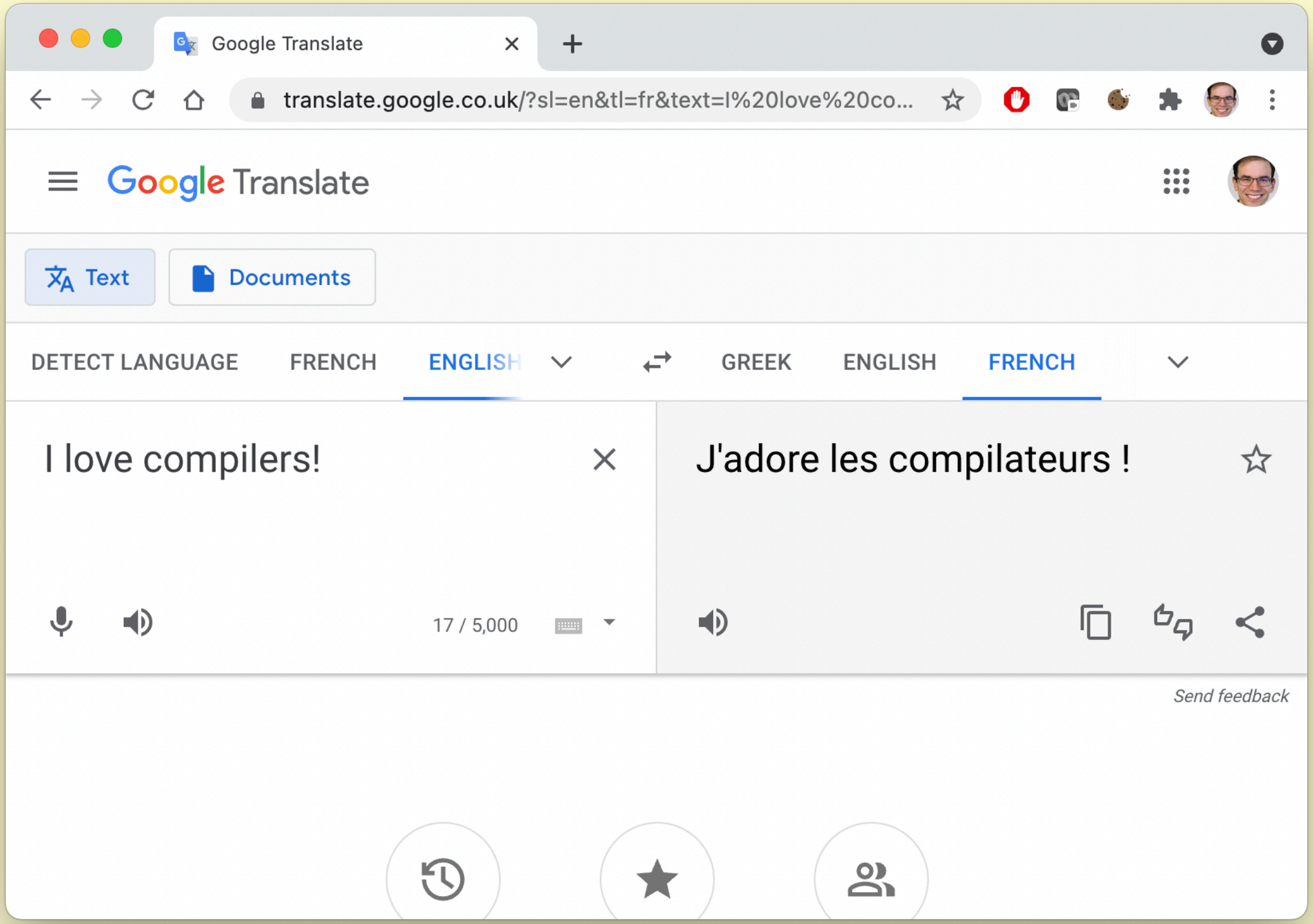
ACM ISBN 978-1-4503-5698-5/18/06.
https://doi.org/10.1145/3192366.3192373

Page 1 of 15

Web browsers



Google Translate

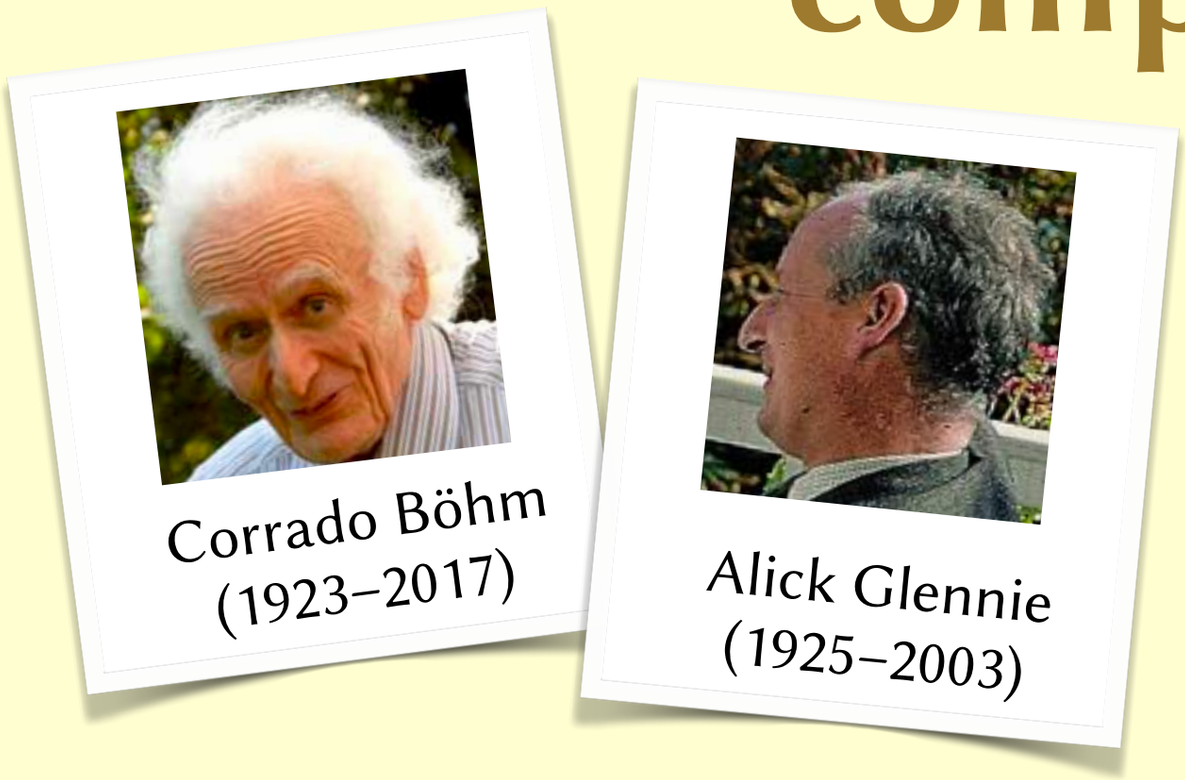


Who made the first compiler?

Corrado Böhm
(1923–2017)

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

Who made the first compiler?



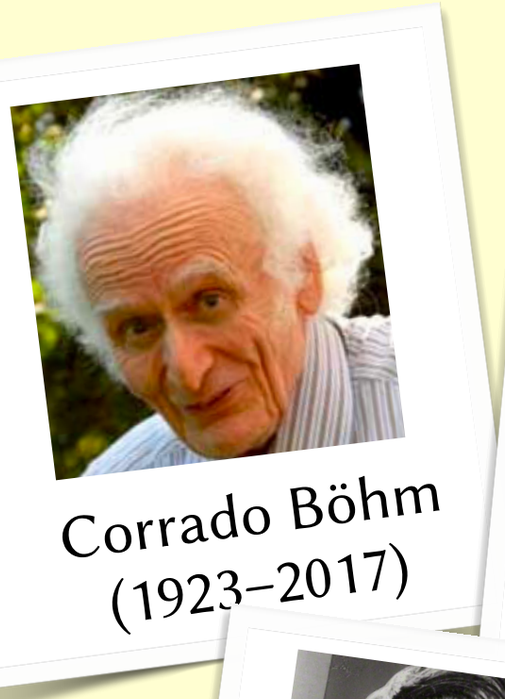
- Designed AUTOCODE in 1951, which translated human-readable instructions into machine code for the Manchester Mark I.

- Nowadays we'd probably call that an "assembler".

Who made the first compiler?

- 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?

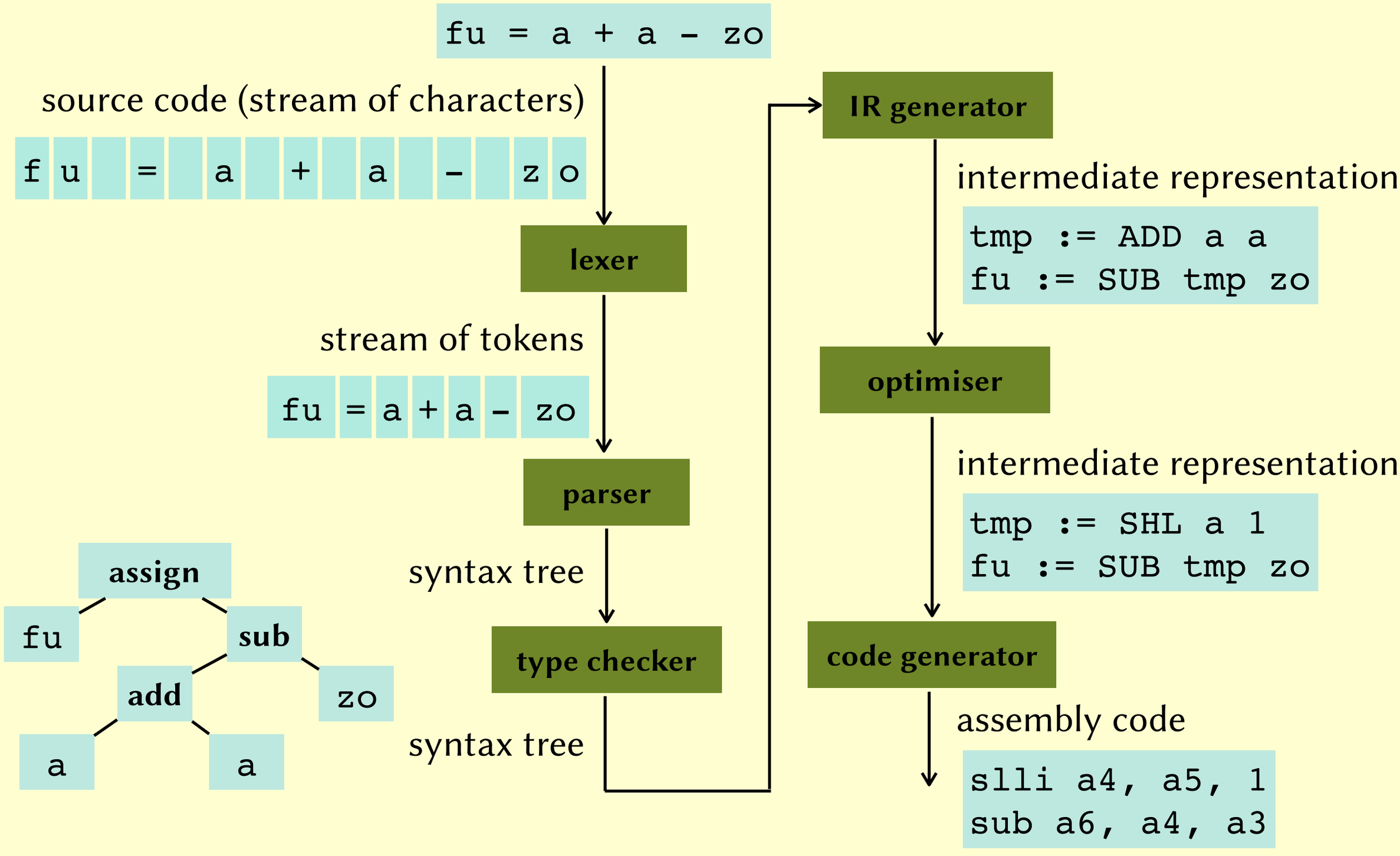
- How is a compiler built?
- In which language should I implement my compiler?
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

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?

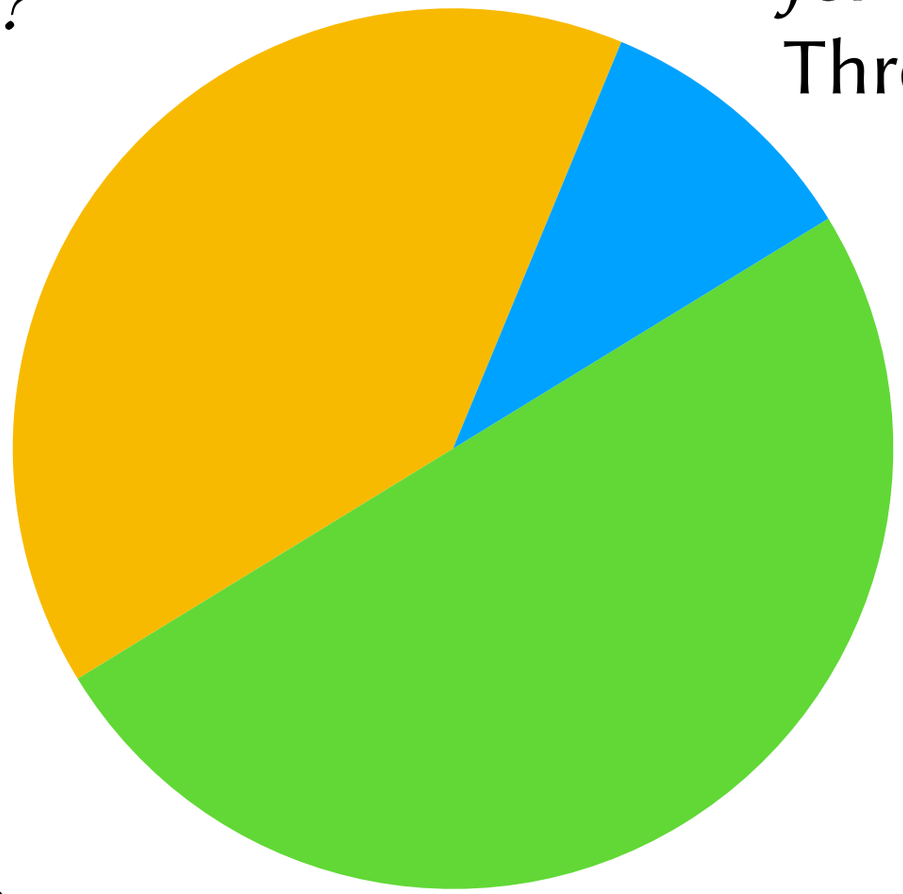
Anatomy of a compiler



Aims and assessment

Knowledge
Do you understand the principles of compiler construction?
Exam, Summer term.

Skills
Can you use standard tools for compiler construction?
Three labs this term.



(This is just the Compilers half of the IAC module)

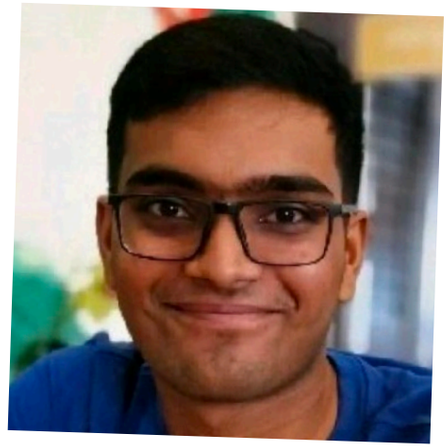
Application
Can you work in pairs to build a working C compiler?
Coursework, due end of term.

(Tentative) Schedule

Week		Mon 1100-1300 (403)	Thu 1100-1300 (usually 508)	Fri 1300-1400 (usually 408)
1	13 Jan	*(No lab)*	Introduction	Lexing
2	20 Jan	Lexing exercise due: Fri 31 Jan 1500	More lexing	Parsing
3	27 Jan		More parsing	Type checking
4	03 Feb	Parsing exercise due: Fri 07 Feb 1500	*(No lecture)*	*(No lecture)*
5	10 Feb	Code gen exercise due: Fri 14 Feb 1500	Interm. representation	Code generation
6	17 Feb	Compiler project due: Fri 21 Mar 1500	C-to-RISCV compilation	Data flow analysis
7	24 Feb		Static analysis	Memory management
8	03 Mar		Compiler correctness	Class quiz 1 (lexing?)
9	10 Mar		Compiling concurrency	Class quiz 2 (parsing?)
10	17 Mar		Industrial guest?	Class quiz 3 (code gen?)

Teaching assistants

Quentin Corradi



Johan Jino

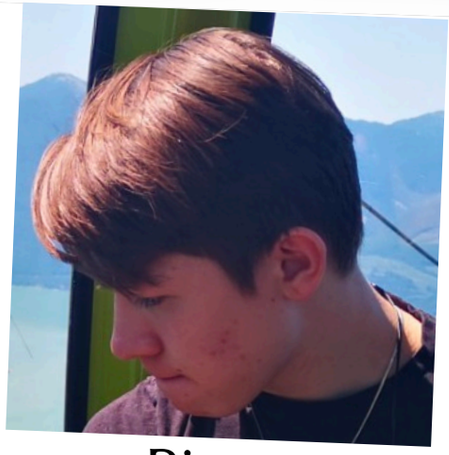
William Huynh

Samuel Wang

Richard Lu

Ryan Voecks

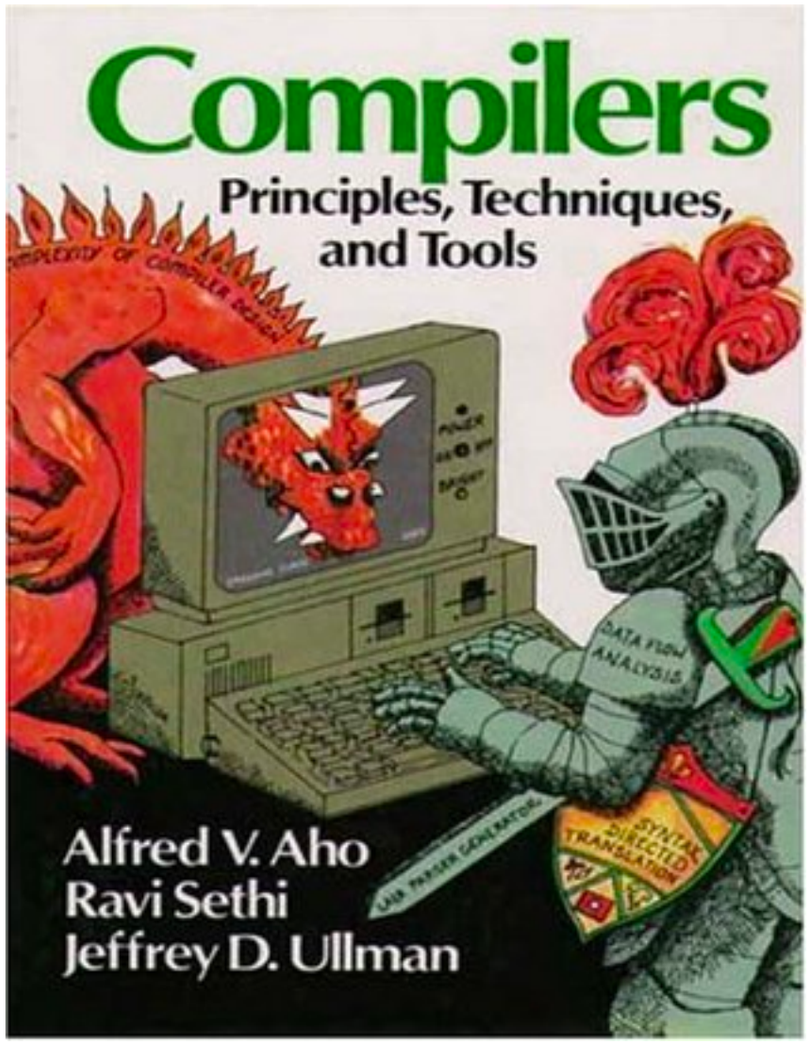
Michalis Pardalos



Pierce Wiegerling

Filip Wojcicki

Recommended Reading



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

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?

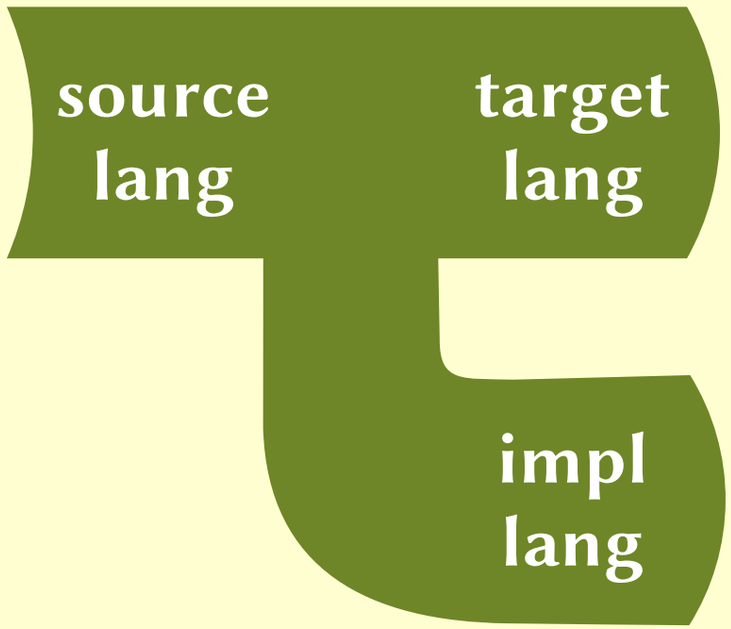
Welcome!

✓ What is a compiler?

✓ How is a compiler built?

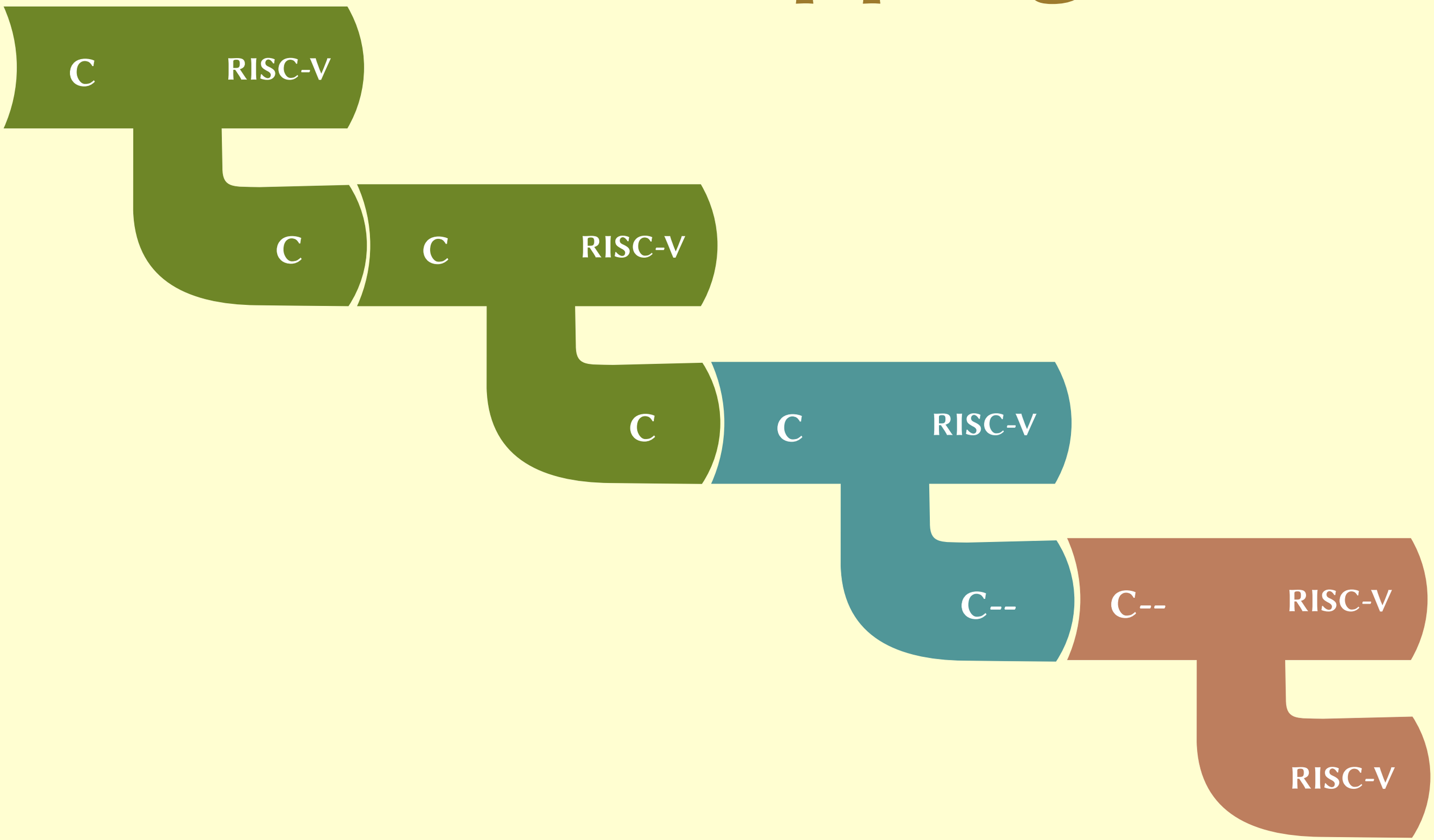
- **In which language should I implement my compiler?**
- To compile or to interpret?
- What makes a good compiler?
- What is the programmer/compiler contract?

Which language?

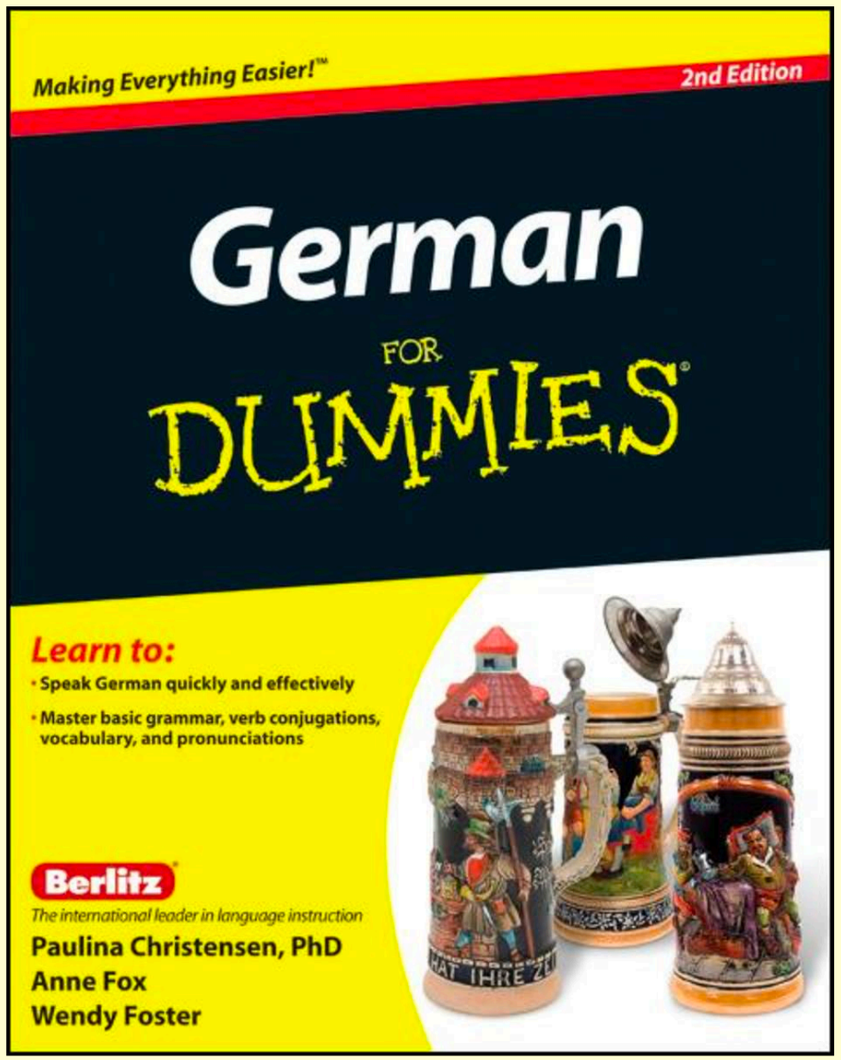


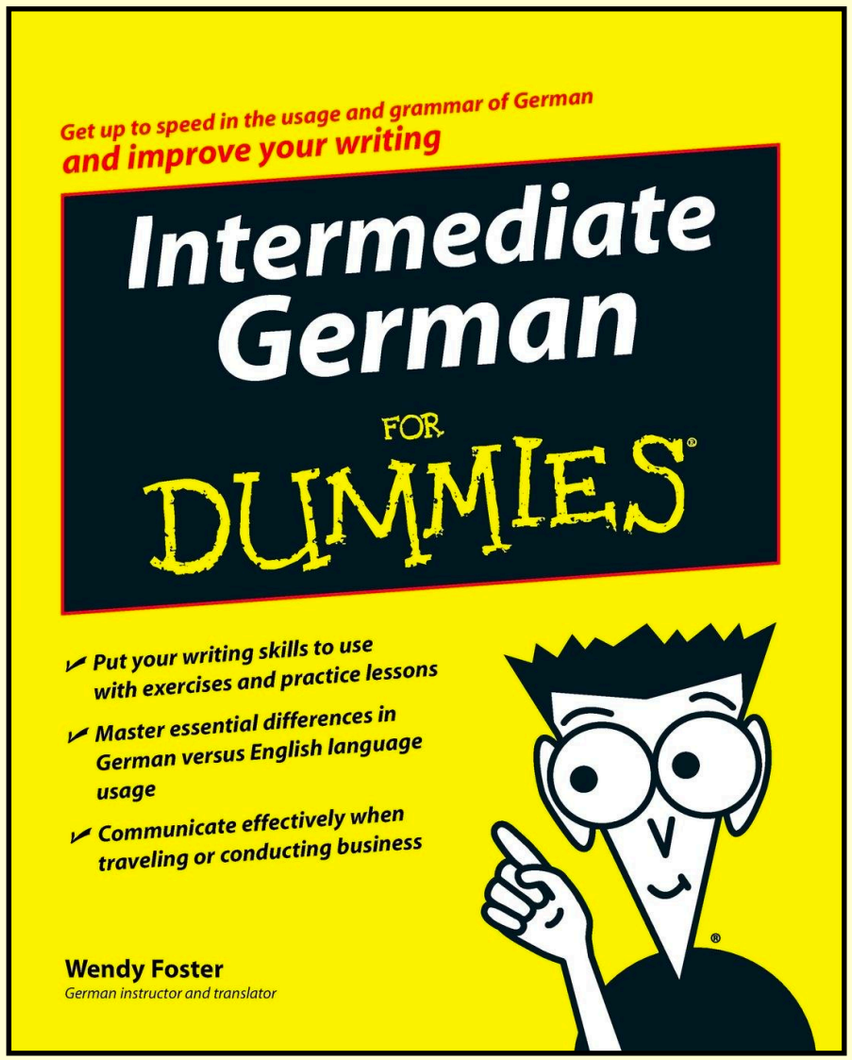
- If source=impl then compiler is called *self-hosting*.
- What if you've written the first compiler for a new language?

Bootstrapping



Analogy


Making Everything Easier!™
2nd Edition
German
FOR
DUMMIES
Learn to:
• Speak German quickly and effectively
• Master basic grammar, verb conjugations, vocabulary, and pronunciations
Berlitz
The international leader in language instruction
Paulina Christensen, PhD
Anne Fox
Wendy Foster


Get up to speed in the usage and grammar of German
and improve your writing
Intermediate
German
FOR
DUMMIES
Put your writing skills to use with exercises and practice lessons
Master essential differences in German versus English language usage
Communicate effectively when traveling or conducting business
Wendy Foster
German instructor and translator


B2
Deutsch üben
Lesen & Schreiben B2
Hueber

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!

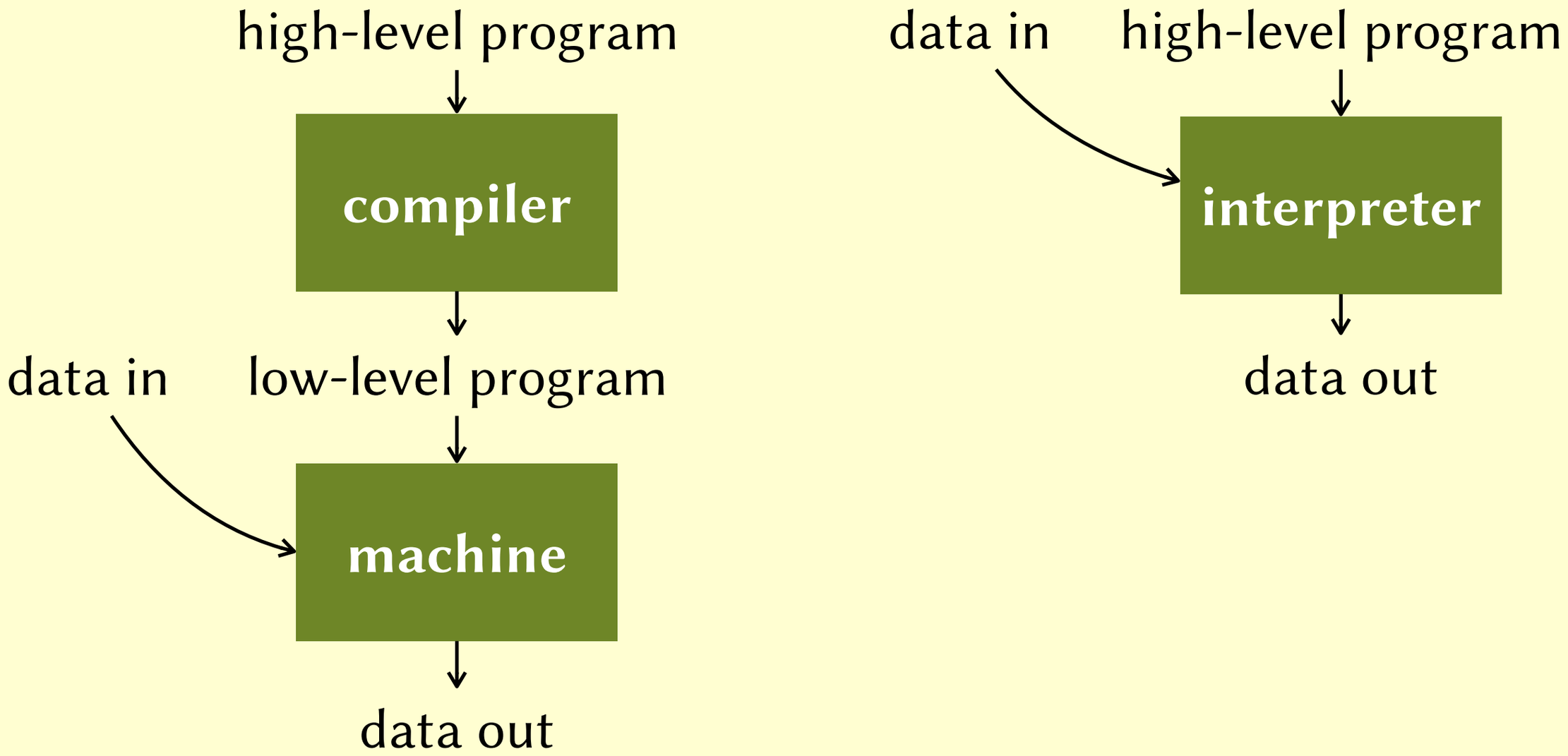
✓ What is a compiler?

✓ How is a compiler built?

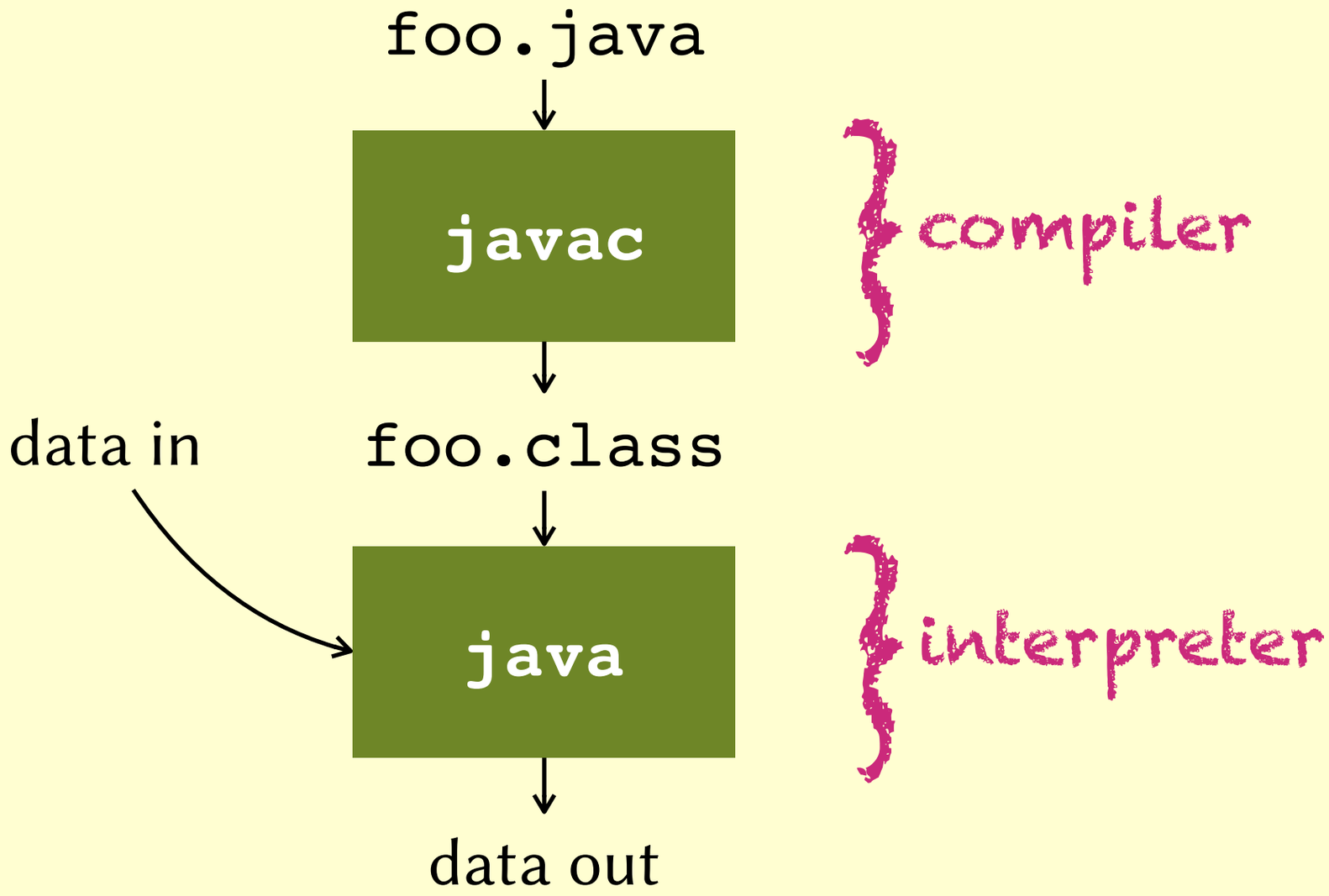
✓ In which language should I implement my compiler?

- **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]*

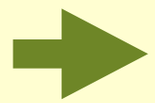
or, more simply:

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!

Acknowledgement: https://blog.regehr.org/archives/213

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?