Compiler Construction

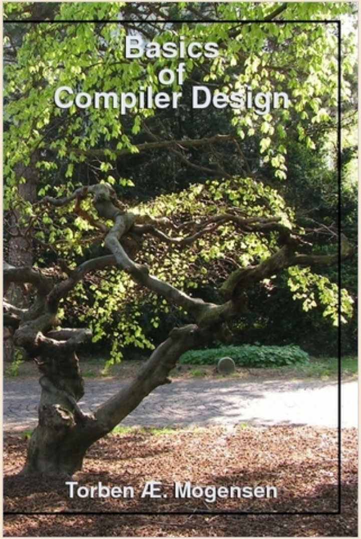
Lecture 16

Bootstrapping

Jeremy Yallop, Lent 2023

jeremy.yallop@cl.cam.ac.uk

Recommended book



Chapter 13 of

Basics of Compiler Design

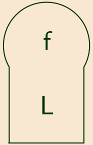
Torben Ægidius Mogensen

`http://hjemmesider.diku.dk/~torbenm/Basics/`

Notation

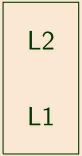
Notation: programs, interpreters, machines

A program



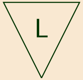
Computes function f
written in language L

An interpreter



Interprets language L2
written in language L1

A machine



Executes code
in language L

Notation

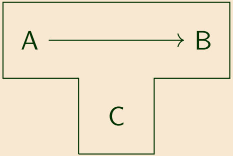
Examples

Compiling compilers

Full bootstrap

Notation: compilers

A compiler

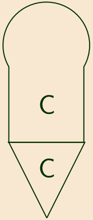


Translates language A into language B
Written in language C

Examples

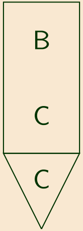
Executing programs

To execute a program



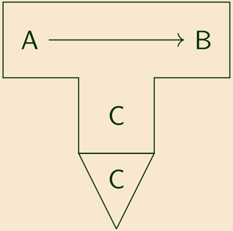
we run it on a machine

To execute an interpreter



we run it on a machine

To execute a compiler



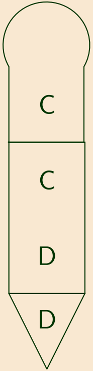
we run it on a machine

Interpreting a program

Run a program written in language C

on an interpreter for C written in language D

on a D machine



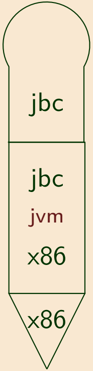
(Note: the languages must match)

Interpreting a Java program

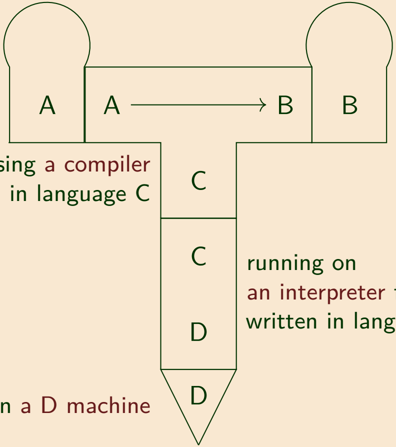
Run a program written in Java byte code

on an interpreter for Java byte code written in x86 code

on a x86 machine



Running a compiler on an interpreter



compile a program
in language A

into a program
in language B

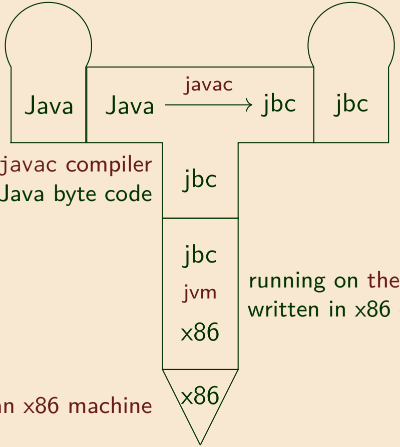
using a compiler
written in language C

running on
an interpreter for C
written in language D

running on a D machine

Running javac on the JVM

compile a program in Java



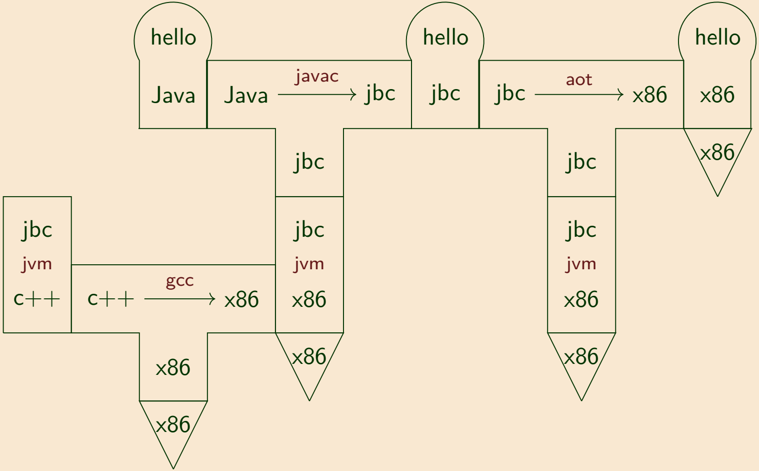
into a program in Java byte code

using the javac compiler written in Java byte code

running on the JVM written in x86 code

running on an x86 machine

Ahead-of-time compilation for Java

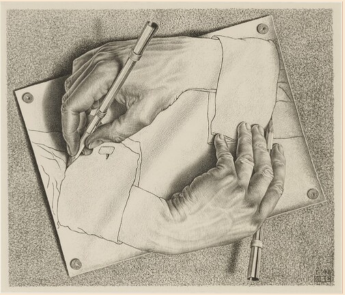


Thanks to David Greaves for the example

Compiling compilers

A puzzle

The OCaml compiler
is written in OCaml



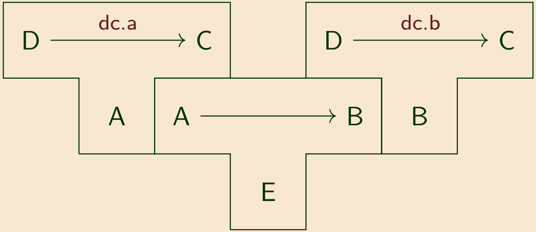
Puzzle: how was the compiler compiled?

Translating translators

Compilers can be translated, just like any other program:

a compiler from D to C
in language A

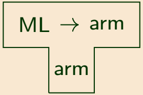
a compiler from D to C
in language B



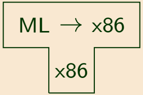
compile programs from A to B

Porting a compiler to a new platform

We have:
a compiler from ML to arm
that runs on arm

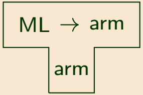


We want:
a compiler from ML to x86
that runs on x86

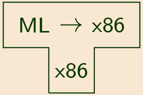


Porting a compiler to a new platform

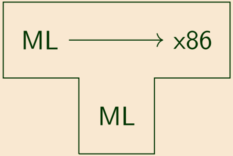
We have:
a compiler from ML to arm
that runs on arm



We want:
a compiler from ML to x86
that runs on x86

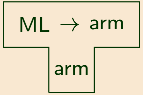


1. write an ML-to-x86 compiler in ML

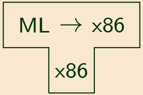


Porting a compiler to a new platform

We have:
a compiler from ML to arm
that runs on arm

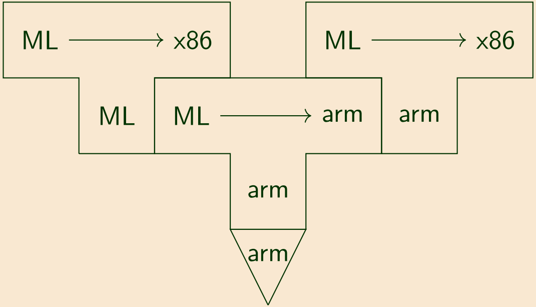


We want:
a compiler from ML to x86
that runs on x86



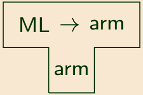
1. write an ML-to-x86 compiler in ML

2. compile the compiler for arm

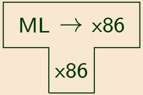


Porting a compiler to a new platform

We have:
a compiler from ML to arm
that runs on arm



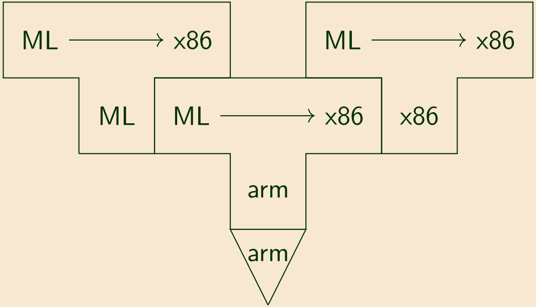
We want:
a compiler from ML to x86
that runs on x86



1. write an ML-to-x86 compiler in ML

2. compile the compiler for arm

3. run the compiler on arm to compile itself



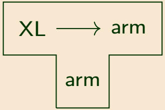
Full bootstrap

Half and full bootstraps

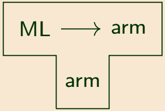
Previous example: *half bootstrap* (needs existing compiler for the language).

New example: *full bootstrap* (no existing ML compiler for the language)

We want:
a compiler from **XL** to arm
that runs on arm

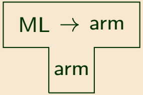


We have:
a compiler from ML to arm
that runs on arm

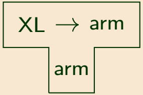


Full bootstrap

We have:
a compiler from ML to arm
that runs on arm

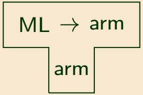


We want:
a compiler from **XL** to arm
that runs on arm

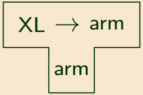


Full bootstrap

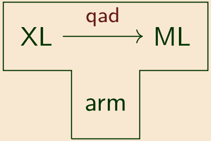
We have: a compiler from ML to arm that runs on arm



We want: a compiler from **XL** to arm that runs on arm

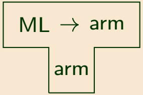


1. write a quick-and-dirty (QAD) **XL-to-ML** compiler in ML

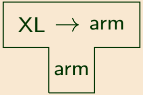


Full bootstrap

We have:
a compiler from ML to arm
that runs on arm

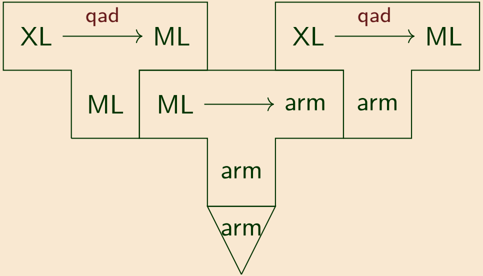


We want:
a compiler from **XL** to arm
that runs on arm



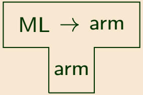
1. write a quick-and-dirty (QAD) **XL-to-ML** compiler in ML

2. compile the QAD compiler for arm



Full bootstrap

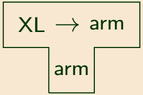
We have:
a compiler from ML to arm
that runs on arm



ML → arm

arm

We want:
a compiler from **XL** to arm
that runs on arm



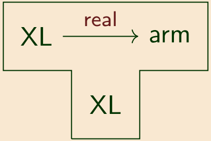
XL → arm

arm

1. write a quick-and-dirty (QAD) **XL-to-ML** compiler in ML

2. compile the QAD compiler for arm

3. Write a real **XL-to-arm** compiler in **XL**

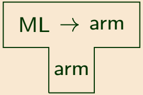


$XL \xrightarrow{real} arm$

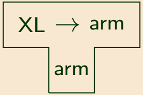
XL

Full bootstrap

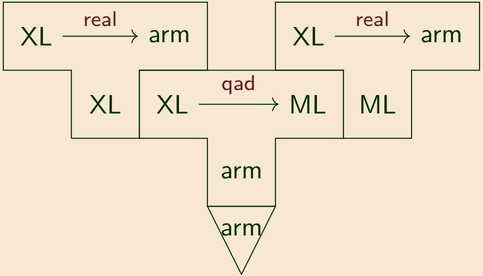
We have:
a compiler from ML to arm
that runs on arm



We want:
a compiler from **XL** to arm
that runs on arm

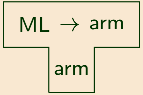


1. write a quick-and-dirty (QAD) **XL-to-ML** compiler in ML
2. compile the QAD compiler for arm
3. Write a real **XL-to-arm** compiler in **XL**
4. Use the QAD compiler to compile the real compiler to ML

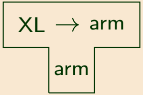


Full bootstrap

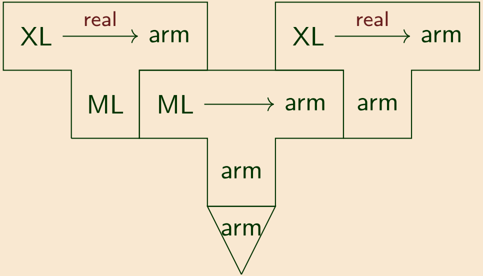
We have:
a compiler from ML to arm
that runs on arm



We want:
a compiler from **XL** to arm
that runs on arm

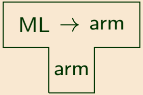


1. write a quick-and-dirty (QAD)
   XL-to-ML compiler in ML
2. compile the QAD compiler for arm
3. Write a real **XL-to-arm** compiler in **XL**
4. Use the QAD compiler to compile
   the real compiler to ML
5. Compile the resulting ML program to arm



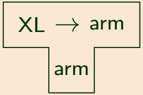
Full bootstrap

We have:
a compiler from ML to arm
that runs on arm



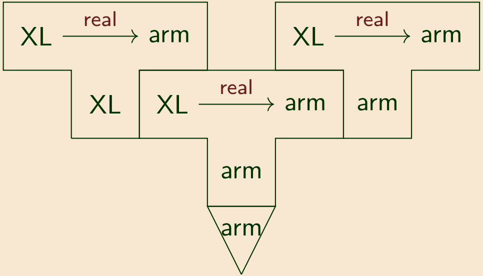
ML → arm
arm

We want:
a compiler from **XL** to arm
that runs on arm



XL → arm
arm

1. write a quick-and-dirty (QAD) **XL-to-ML** compiler in ML
2. compile the QAD compiler for arm
3. Write a real **XL-to-arm** compiler in **XL**
4. Use the QAD compiler to compile the real compiler to ML
5. Compile the resulting ML program to arm
6. Use the generated compiler to compile itself



XL $\xrightarrow{\text{real}}$ arm
XL $\xrightarrow{\text{real}}$ arm
XL
XL $\xrightarrow{\text{real}}$ arm
arm
arm
arm

Observations

The *speed* of the quick-and-dirty compiler does not matter much
(We could even use a quick-and-dirty interpreter instead)

‘We don’t need to give the quick-and-dirty compiler to users

Once the real compiler works, we can discard the quick-and-dirty compiler altogether