Programming Paradigms

Introduction

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2022

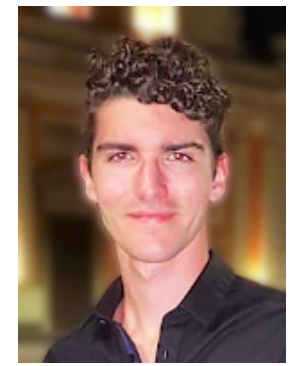
About Me: Michael Pradel

- **Since 9/2019: Full Professor at University of Stuttgart**
- **Before**
  - Studies at TU Dresden, ECP (Paris), and EPFL (Lausanne)
  - PhD at ETH Zurich, Switzerland
  - Postdoctoral researcher at UC Berkeley, USA
  - Assistant Professor at TU Darmstadt
  - Sabbatical at Facebook, Menlo Park, USA

About the Software Lab








- **My research group since 2014**
- **Focus: Tools and techniques for building reliable, efficient, and secure software**
  - Program testing and analysis
  - Machine learning, security
- **Thesis and job opportunities**

Overview

- **Motivation** ←
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization**
  - Exercises
  - Grading
- **Introduction**
  - Programming languages:
    History, paradigms, compilation, interpretation

The Role of Programming

- **Programming: Essential form of expression for a computer scientist**
  - ”The limits of my language mean the limits of my world.” (Ludwig Wittgenstein)
- **Programming languages determine what algorithms and ideas you can express**

Goal of this Course

Understand how programming languages (PLs) work

- How are languages defined?
- What language design choices exist?
- How are languages implemented?

Why Learn About PLs?

Enables you to

- choose right PL for a specific purpose
- choose among alternative ways to express things
- make best use of tools (e.g., debuggers, IDEs, analysis tools)
- understand obscure language features
- simulate useful features in languages that lack them

Concepts vs. Languages

This course is not about

- All details of a specific language
- A systematic walk through a set of languages

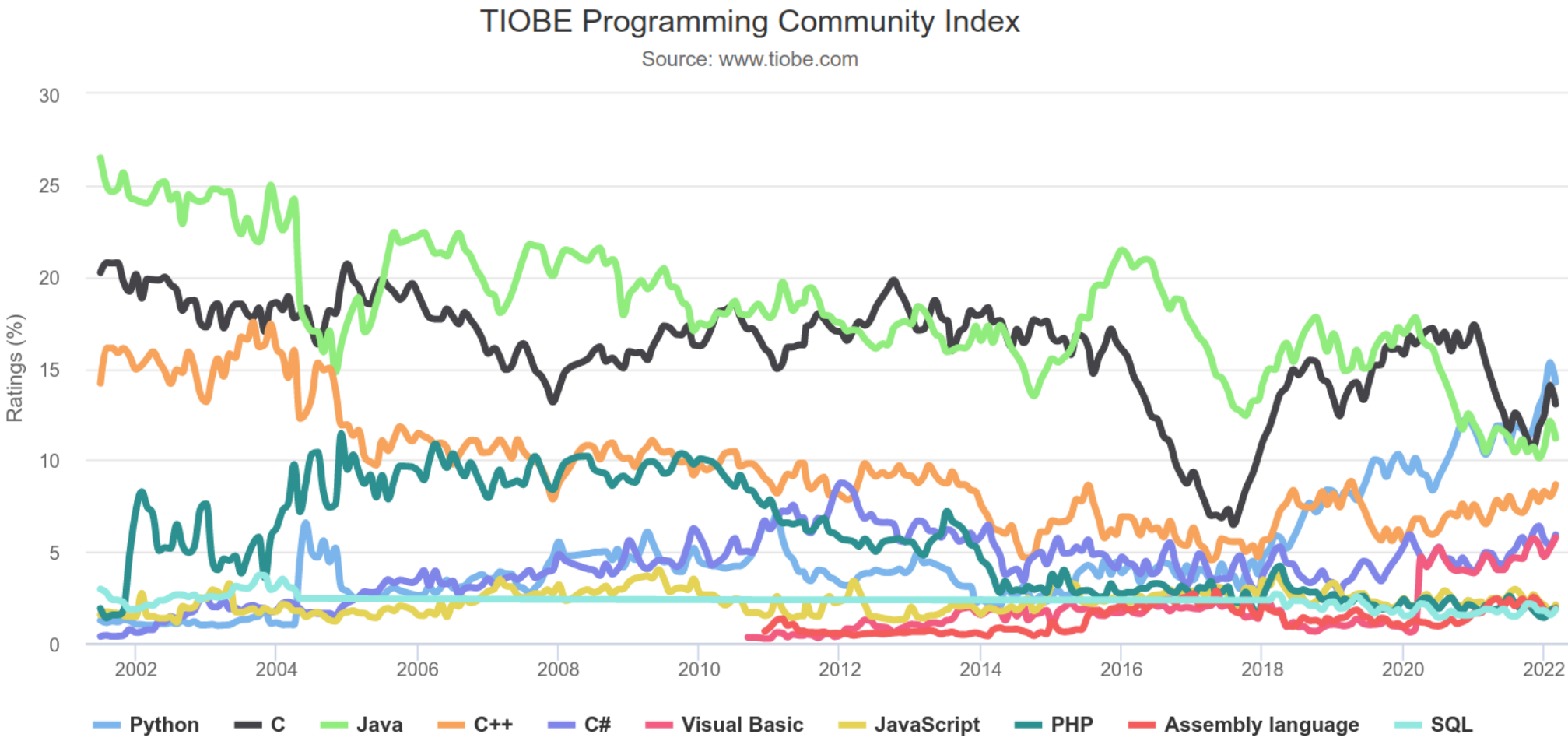
Instead, this course is about

- Concepts underlying many languages
- Various languages as examples

Isn't Knowing {Pick a PL} Enough?

- **Complex systems: Built in various languages**
  - E.g., Facebook: Wild mix of languages covering various language paradigms
- **New languages arrive regularly (and old ones fade away)**

Isn't Knowing {Pick a PL} Enough?



Plan for Today

- **Motivation**
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization** 
  - Exercises
  - Grading
- **Introduction**
  - Programming languages:
    History, paradigms, compilation, interpretation

Language

- **Written material** (slides, exercises): English
- **Lectures**: German
- **Exercise discussions**: English
- **Final exam**: Questions in English, answers in German or English

Schedule

Three weekly slots:

Mon, Wed, Fri, all 11:30am

- But: Not all slots used
- See course page for schedule:

  http://software-lab.org/teaching/summer2022/pp/

Lectures

Slides, hand-written notes, etc:

- Made available shortly after each lecture

Lecture videos:

- Old videos available on course page
- Recommendation: Use for exam preparation, not as replacement for live lectures

Exercises

- **Six graded exercises**
- **We publish on day X**
  - On the course page
- **You submit your solution by day X+7**
  - Via Ilias
- **Discussion of exercises after day X+7**
  - One discussion session for the entire course

Ilias

Platform for discussions, in-class quizzes, and sharing additional material

- Please register for the course
- Use it for all questions related to the course
- Messages sent to all students go via Ilias

Link to Ilias course on
software-lab.org/teaching/summer2022/pp/

Quizzes During the Lectures

- **A few quizzes during each lecture**
  - Check your understanding
  - Access quizzes via Ilias
- **Up to two bonus points for the final exam**
  - Partial points for answering at all
  - Full points for correct answers

Questions and Discussions

For any (non-personal) questions:

Use forum in Ilias

- Preferred language: English
- Answering each other is encouraged
- Teaching assistants and me are monitoring it

Grading

- **Exercises: Passing is prerequisite for final exam (“Schein”)**
  - Six exercises
  - Each exercise: 100 points
  - Needed to pass:
    - At least 30 points in five exercises
    - At least 360 total points
  - At least 30 points in all six exercises:
    One bonus point for the final exam
  - Your points: Published after each exercise

Final Exam

- **Final exam: Open book**
  - All printed and hand-written material allowed (incl. slides, textbooks, and a dictionary)
  - Tests your understanding, not your knowledge

Plagiarism and Cheating

- **Exercises are individual**
- **Any form of cheating and plagiarism**
  - Treated like cheating in an exam
  - I.e., failing the ”Schein”
- **Cheating includes**
  - Showing your solution to others
  - Working together in a solution
  - Using a solution from someone else

Reading Material

- **No script or book covers everything**
  - Most relevant book: *Programming Language Pragmatics* by Michael L. Scott
  - Also interesting: *Concepts of Programming Languages* by Robert W. Sebesta
- **Pointers to book chapters and web resources: Course page**

Plan for Today

- **Motivation**
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization**
  - Exercises
  - Grading
- **Introduction** 
  - Programming languages:
    History, paradigms, compilation, interpretation

History: From Bits ...

First electronic computers: Programmed in machine language

- Sequence of bits
- Example: Calculate greatest common divisor

```
55 89 e5 53  83 ec 04 83  e4 f0 e8 31  00 00 00 89  c3 e8 2a 00
00 00 39 c3  74 10 8d b6  00 00 00 00  39 c3 7e 13  29 c3 39 c3
75 f6 89 1c  24 e8 6e 00  00 00 8b 5d  fc c9 c3 29  d8 eb eb 90
```

History: From Bits ...

First electronic computers: Programmed in machine language

- Sequence of bits
- Example: Calculate greatest common divisor

```
55 89 e5 53  83 ec 04 83  e4 f0 e8 31  00 00 00 89  c3 e8 2a 00
00 00 39 c3  74 10 8d b6  00 00 00 00  39 c3 7e 13  29 c3 39 c3
75 f6 89 1c  24 e8 6e 00  00 00 8b 5d  fc c9 c3 29  d8 eb eb 90
```

Machine time more valuable than developer time

... over Assembly ...

Human-readable abbreviations for
machine language instructions

- Less error-prone, but still very machine-centered
- Each new machine: Different assembly language
- Developer thinks in terms of low-level operations

... over Assembly ...

Greatest common divisor in x86:

```
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        andl    $-16, %esp
        call    getint
        movl    %eax, %ebx
        call    getint
        cmpl    %eax, %ebx
        je      C
A:      cmpl    %eax, %ebx
        jle     D
        subl    %eax, %ebx
B:      cmpl    %eax, %ebx
        jne     A
C:      movl    %ebx, (%esp)
        call    putint
        movl    -4(%ebp), %ebx
        leave
        ret
D:      subl    %ebx, %eax
        jmp     B
```

... to High-level Languages

- **1950s: First high-level languages**
  - Fortran, Lisp, Algol
- **Developer thinks in mathematical and logical abstractions**

... to High-level Languages

Greatest common divisor in Fortran:

```
subroutine gcd_iter(value, u, v)
  integer, intent(out) :: value
  integer, intent(inout) :: u, v
  integer :: t

  do while( v /= 0 )
     t = u
     u = v
     v = mod(t, v)
  enddo
  value = abs(u)
end subroutine gcd_iter
```

Today: 1000s of Languages

- **New languages gain traction regularly**
- **Some long-term survivors**
  - Fortran, Cobol, C

Today: 1000s of Languages

- **New languages gain traction regularly**
- **Some long-term survivors**
  - Fortran, Cobol, C

Poll:

Your favorite programming language?

See LiveVoting in Ilias

What Makes a PL Successful?

- **Expressive power**
  - But: All PLs are Turing-complete
- **Ease of learning (e.g., Basic, Python)**
- **Open source**
- **Standardization: Ensure portability across platforms**
- **Excellent compilers**
- **Economics**
  - E.g., C# by Microsoft, Objective-C by Apple

PL Spectrum

- **Broad classification**
  - Declarative ("what to compute"):

    E.g., Haskell, SQL, spreadsheets
  - Imperative ("how to compute it"):

    E.g., C, Java, Perl
- **Various PL paradigms:**

Sequential
Statically typed
Functional
Distributed-memory parallel
Dynamically typed
Shared-memory parallel
Logic
Dataflow

- **Most languages combine multiple paradigms**

Example: Imperative PL

C implementation for GCD:

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}
```

Example: Imperative PL

C implementation for GCD:

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}
```

Example: Imperative PL

C implementation for GCD:

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}
```

Statements that influence subsequent statements

Assignments with side effect of changing memory

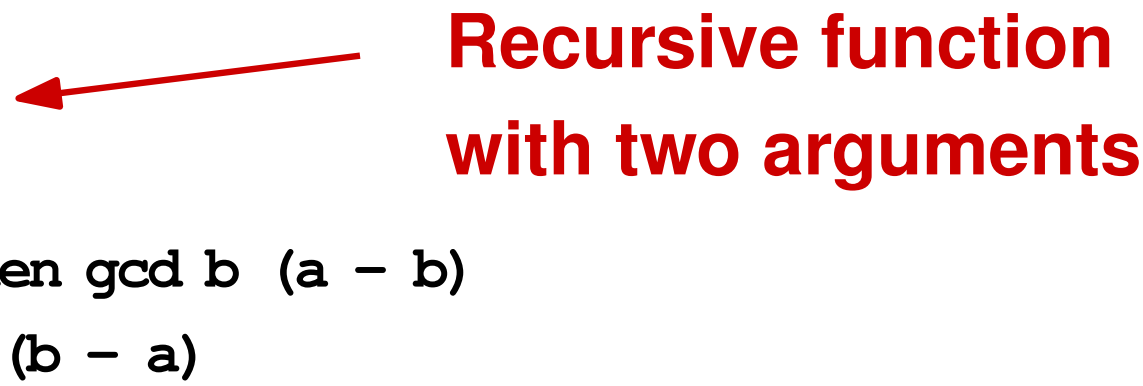
Example: Functional PL

OCaml implementation of GCD

```
let rec gcd a b =
  if a = b then a
  else if a > b then gcd b (a - b)
       else gcd a (b - a)
```

Example: Functional PL

OCaml implementation of GCD




```
let rec gcd a b =
  if a = b then a
  else if a > b then gcd b (a - b)
       else gcd a (b - a)
```

Example: Functional PL

OCaml implementation of GCD

```
let rec gcd a b =
  if a = b then a
  else if a > b then gcd b (a - b)
       else gcd a (b - a)
```

Recursive function with two arguments

Focus on mathematical relationship between inputs and outputs

Example: Logic PL

Prolog implementation of GCD

```
gcd(A,B,G) :- A = B, G = A.
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G).
```

Example: Logic PL

Prolog implementation of GCD

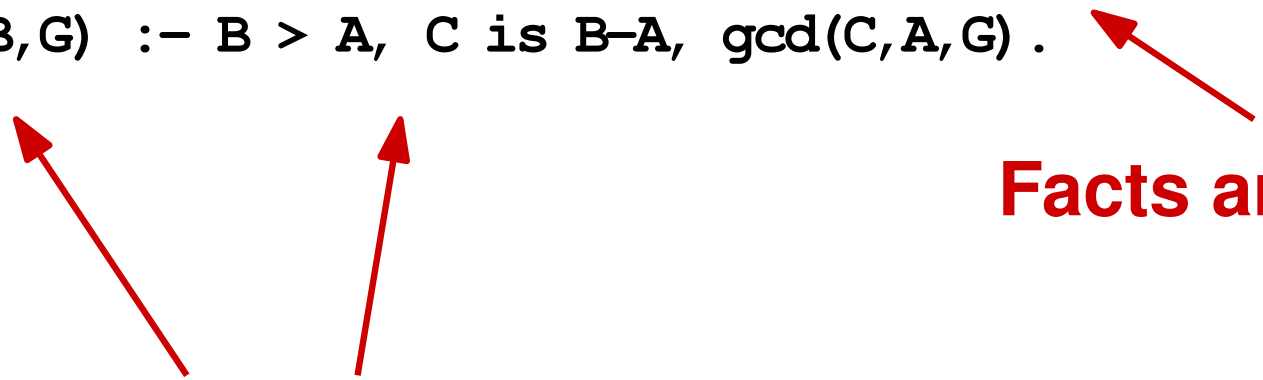
```
gcd(A,B,G) :- A = B, G = A.
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G).
```

Facts and rules

Example: Logic PL

Prolog implementation of GCD

```
gcd(A,B,G) :- A = B, G = A.
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G).
```



Facts and rules

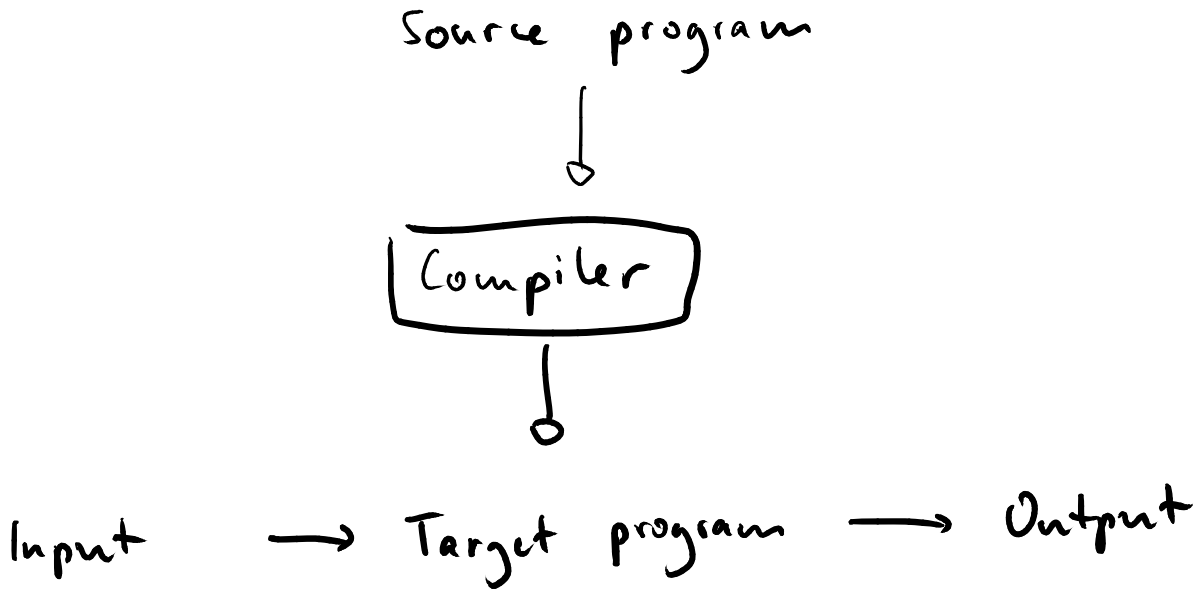
Focus on logical relationships between variables

Compilation and Interpretation

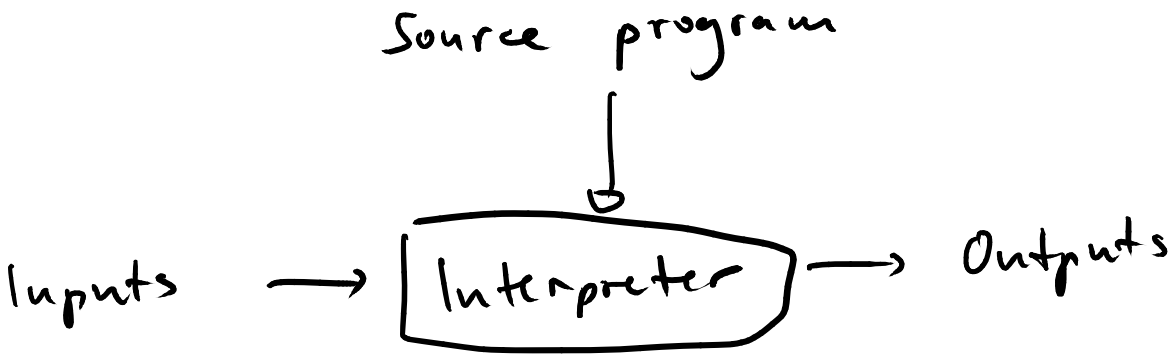
Different ways of executing a program

- Pure compilation
- Pure interpretation
- Mixing compilation and interpretation
  - Virtual machines
  - Just-in-time compilation

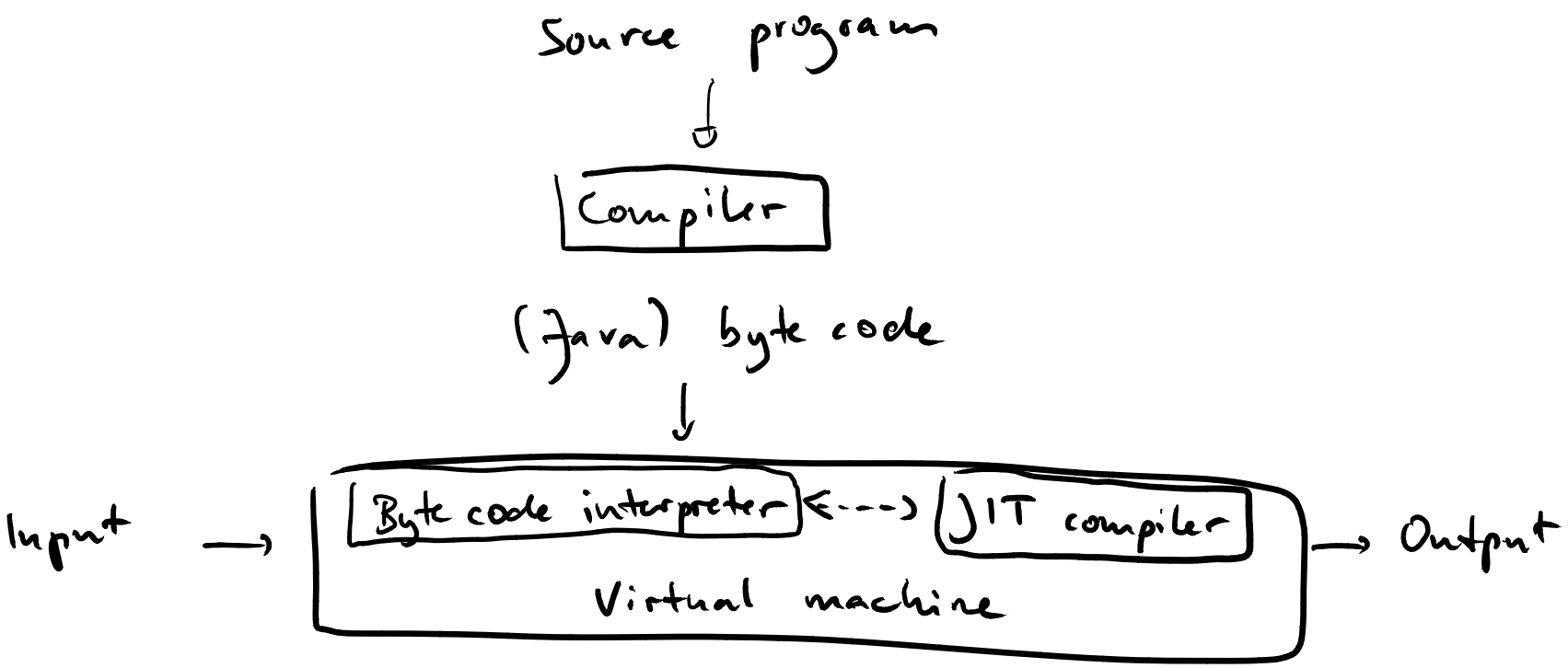
Compilation



Interpreter



Virtual machine + Just-in-time Compilation



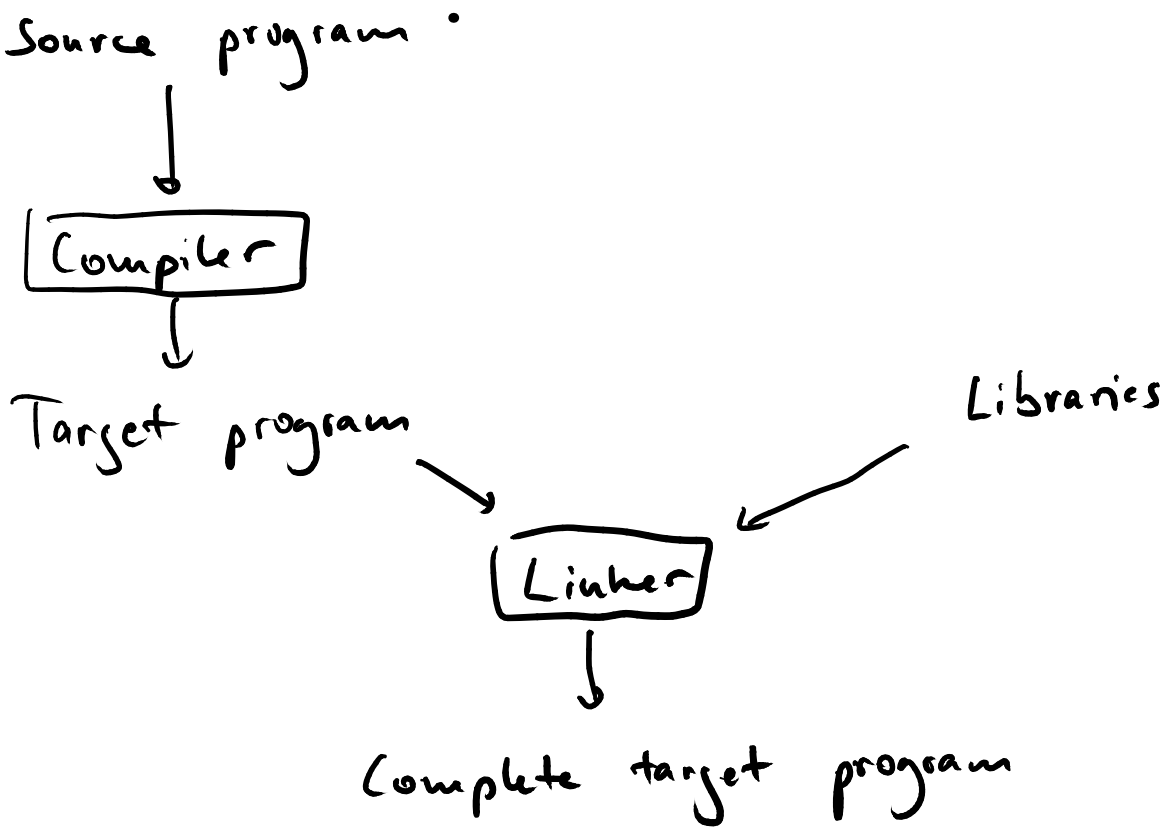
PL Design vs. Implementation

- **Some PLs are easier to compile than others**
- **E.g., runtime code generation**
  - Code to execute: Unknown at compile time
  - Hard to compile
  - Easy to interpret

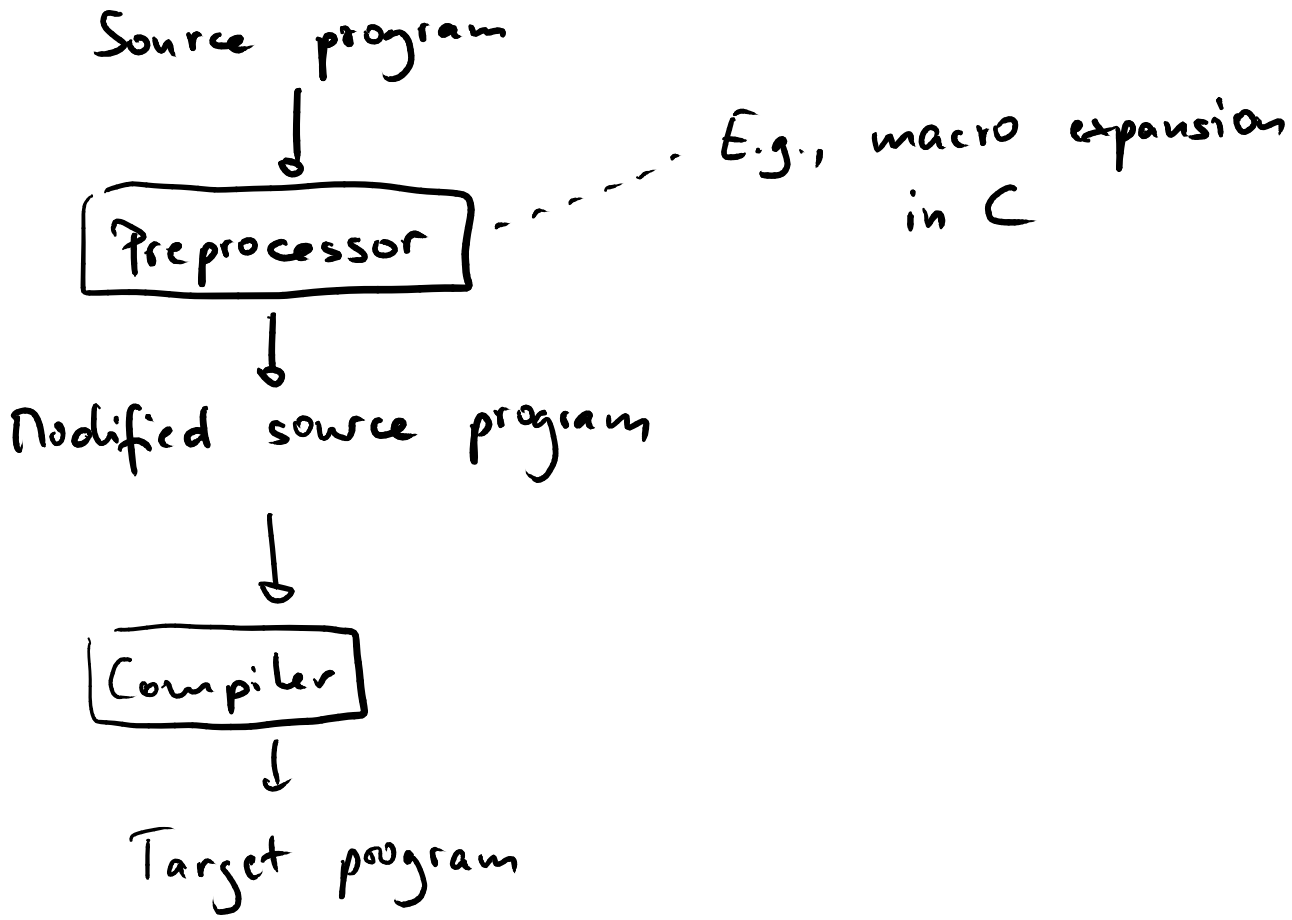
Other Tools

- Linkers
- Preprocessors
- Source-to-source compilers

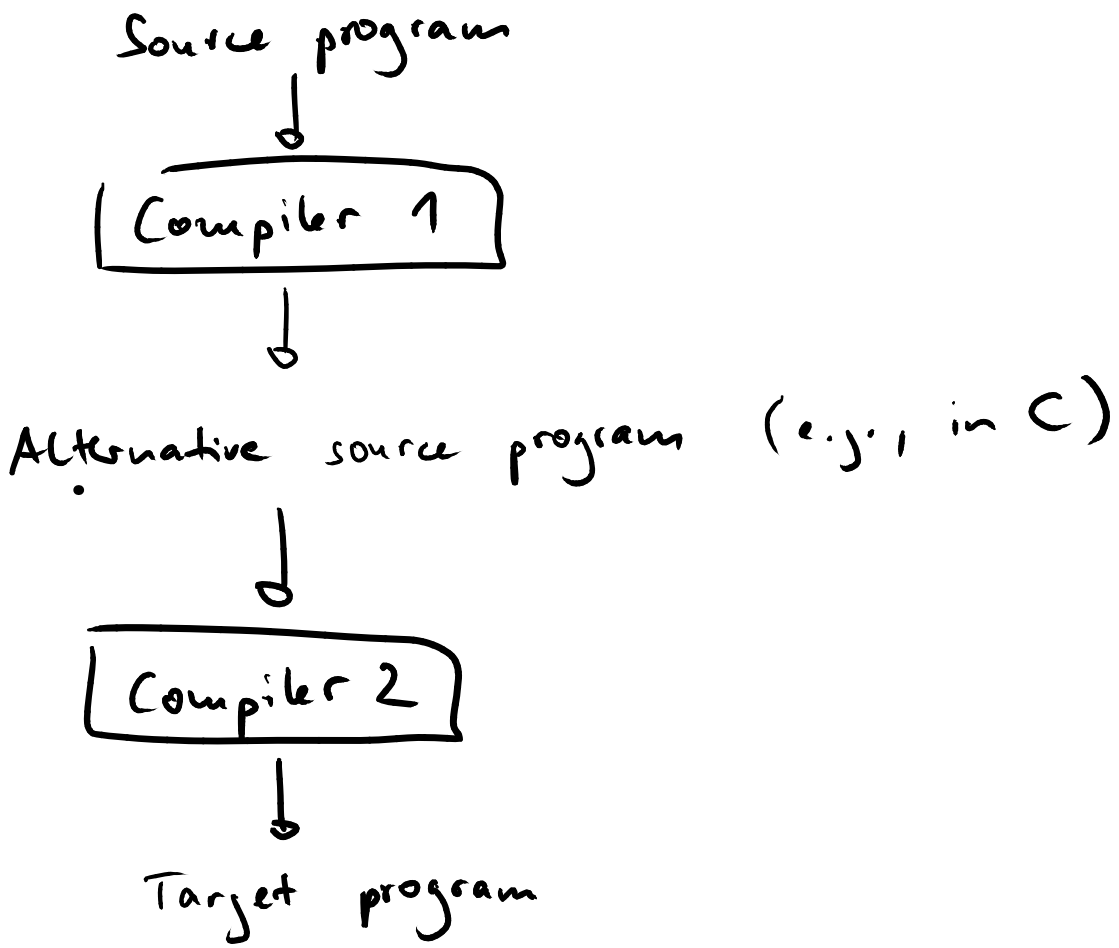
Linking


Source program
Compiler
Target program
Libraries
Linker
Complete target program

Preprocessors



Source-to-source compiler



Plan for Today

- **Motivation**
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization**
  - Exercises
  - Grading
- **Introduction**
  - Programming languages:
    History, paradigms, compilation, interpretation