Introduction to Compiler Construction

or

How to Construct a Self-compiling Compiler in One Semester

Christoph Kirsch

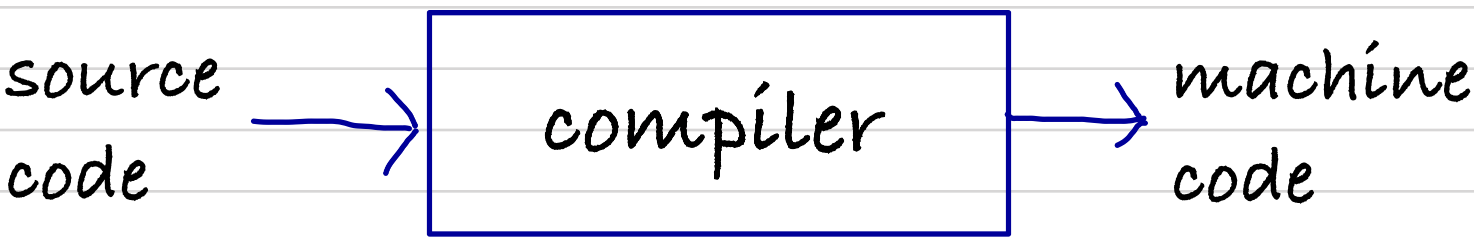
University of Salzburg

Based on Niklaus Wirth's "Compiler Construction", A-W 1996

---

what is a compiler?

a program that translates a program written in a programming language P (source code) into a program written in a machine language M (machine code)



the compiler is self-compiling if it is written in P

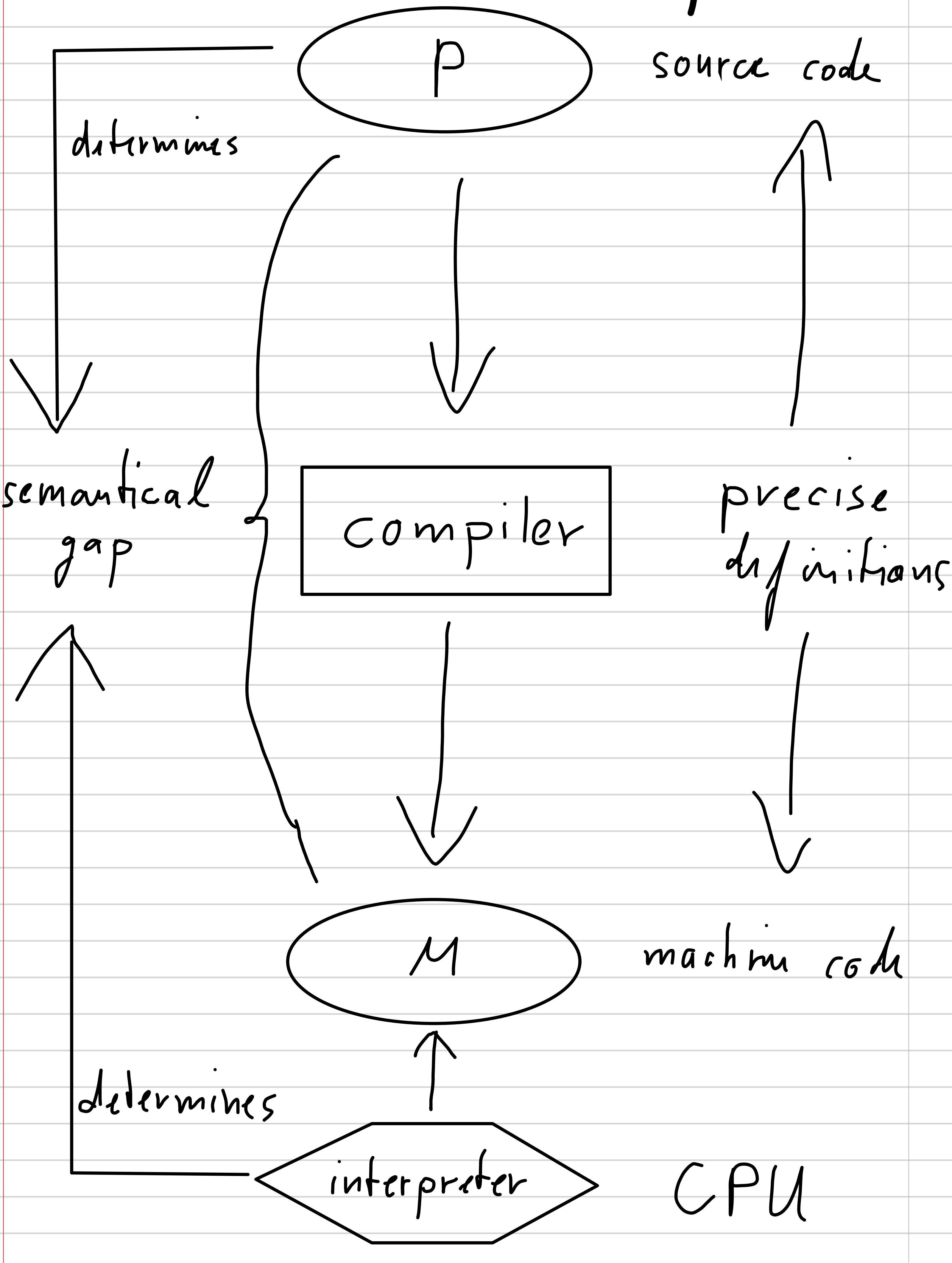
P: structured, imperative language (C, Java, Pascal...); we use C
M: unstructured machine instructions (x86, ARM...); we use (virtualized) RISC

the compiler and the semantics of M determine the semantics of P

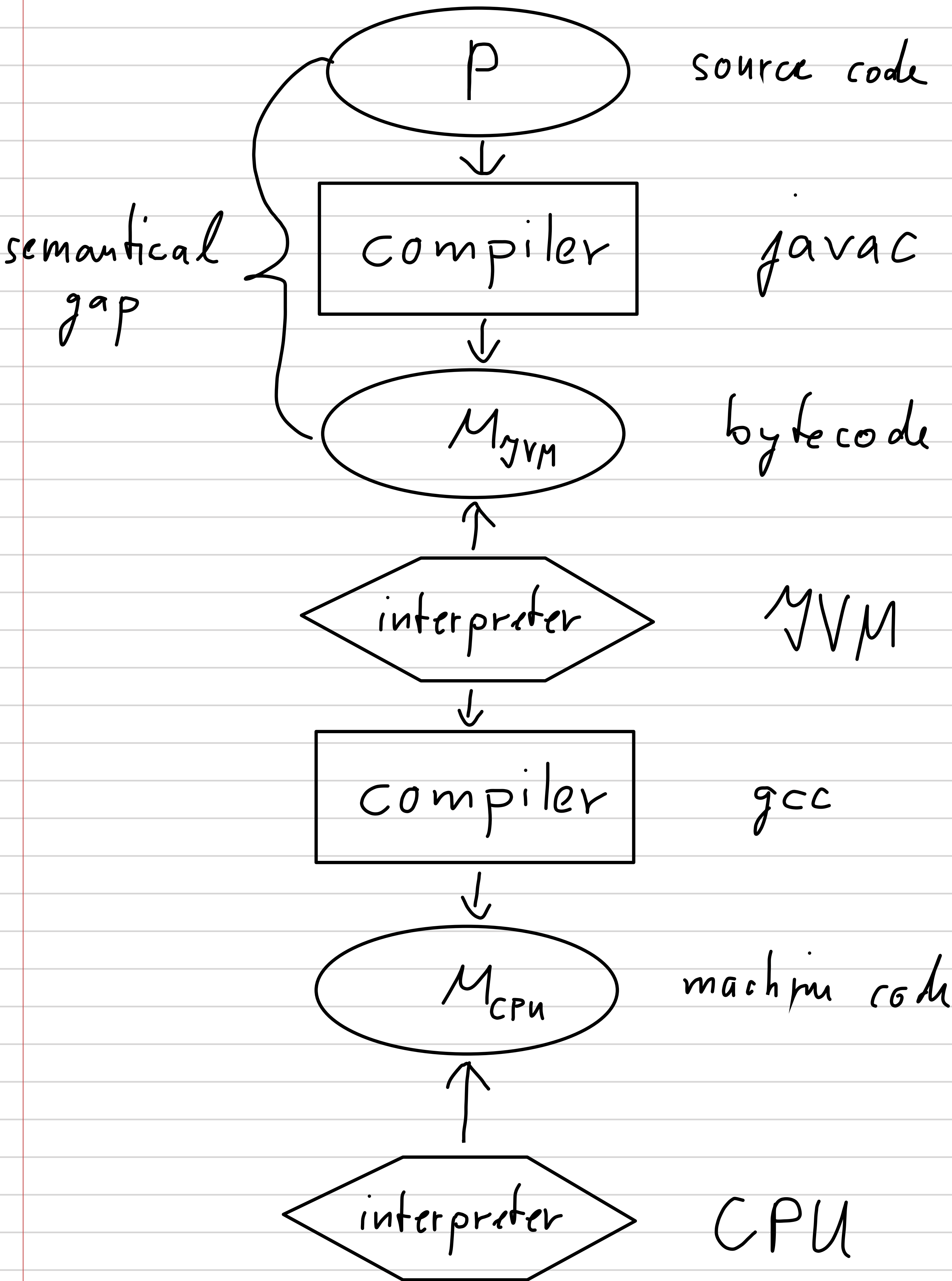
writing a compiler is difficult:
1956 Fortran compiler took 18 person-years!

We will do it in one semester!

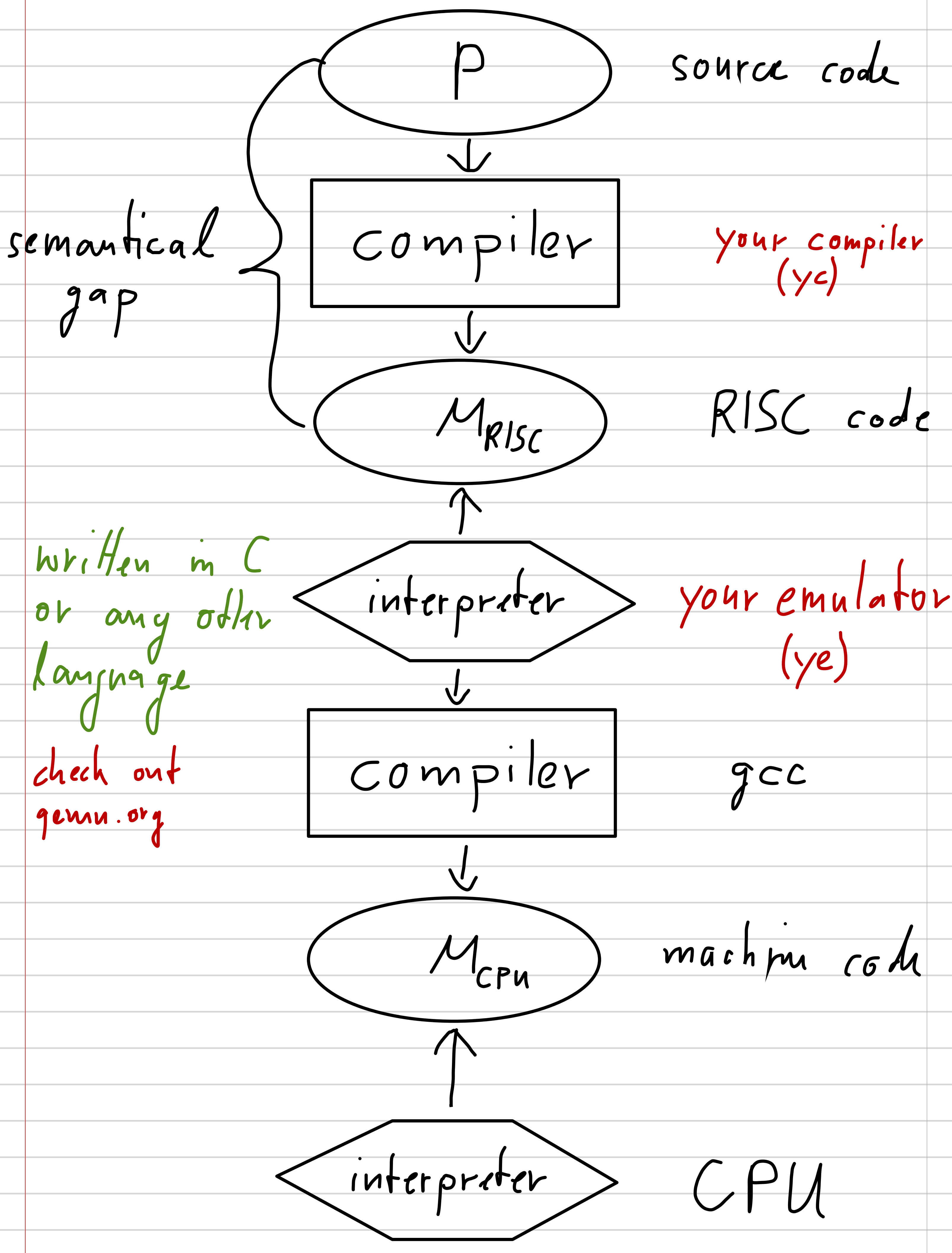
Mind the Gap!


P
source code
determines
semantical gap
compiler
precise definitions
M
machine code
determines
interpreter
CPU

Java!


P
source code
compiler
javac
semantical gap
M_{JVM}
bytecode
interpreter
JVM
compiler
gcc
M_{CPU}
machine code
interpreter
CPU

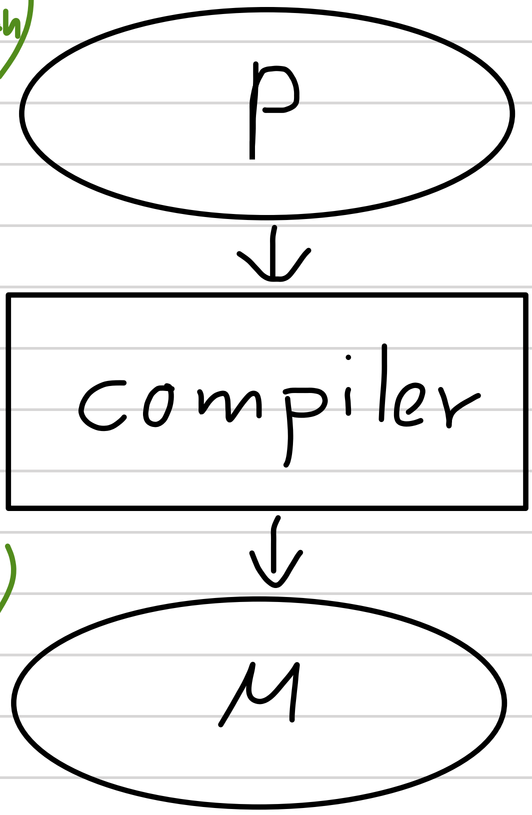
What Do We Do?


P
source code
compiler
your compiler (yc)
semantical gap
M_{RISC}
RISC code
written in C or any other language
interpreter
your emulator (ye)
check out qemu.org
compiler
gcc
M_{CPU}
machine code
interpreter
CPU

Separate Compilation

(as opposed to independent compilation)

partial description of a program, refers to items (variables, code, etc.) defined elsewhere



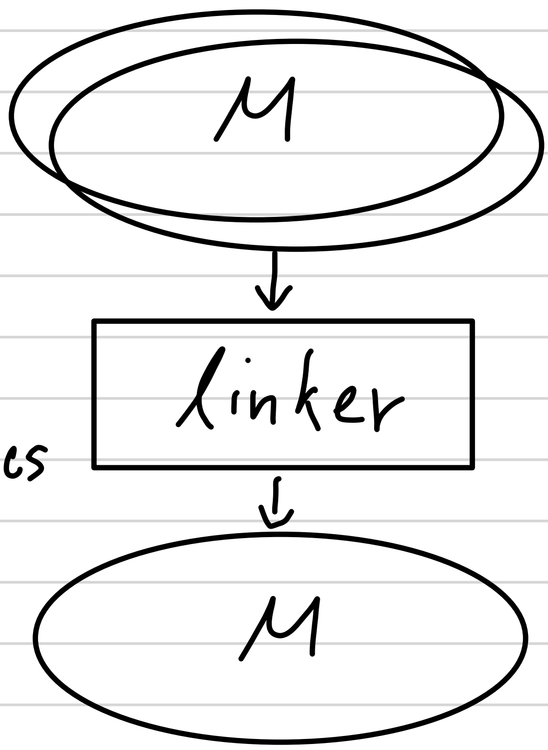
source code (.c)

gcc

object code (.o)

object code:
machine code + symbolic references (named)

(static) Linker:
resolves symbolic references into direct references



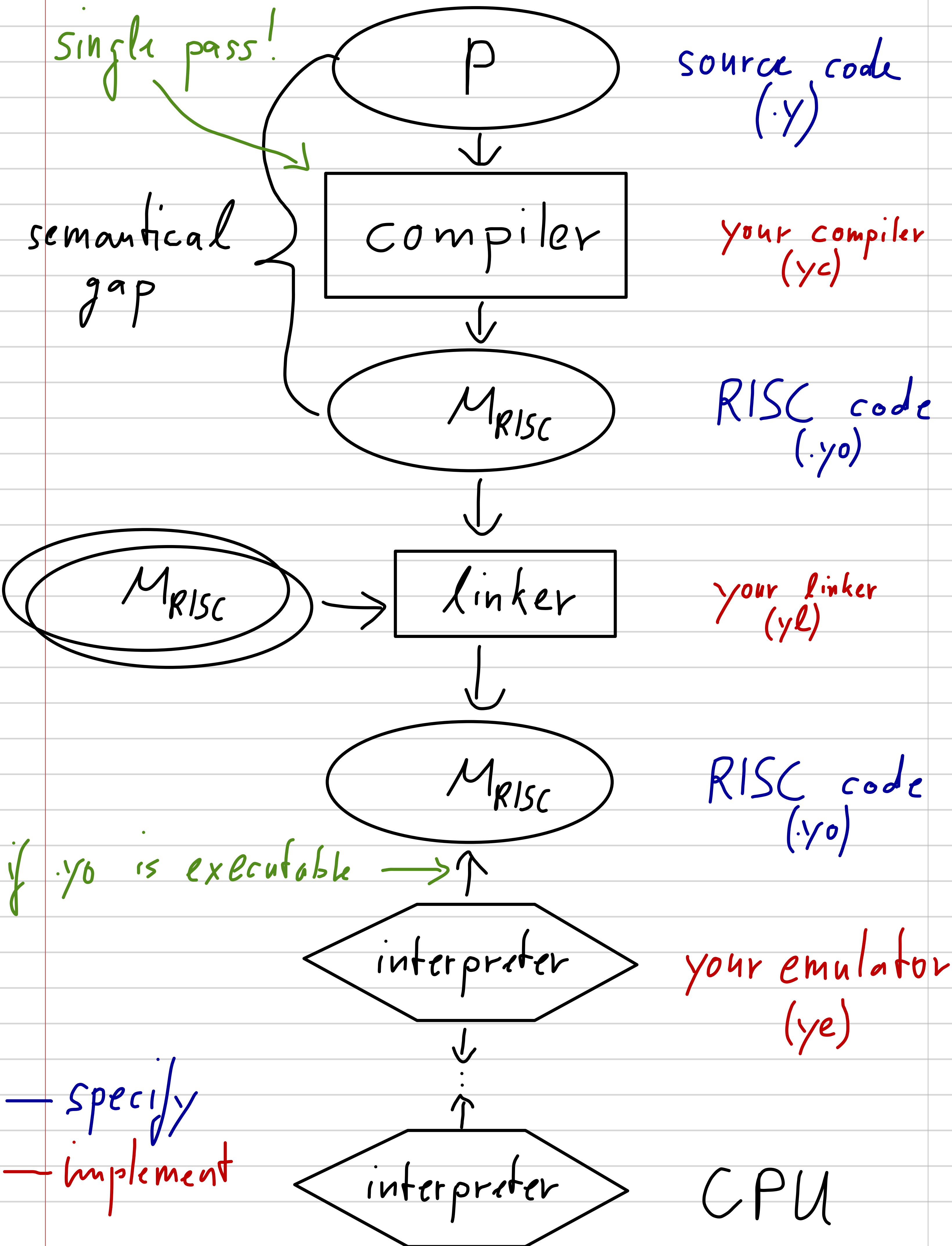
.o, ..., .o

gcc

.o not necessarily executable

executable:
machine code without symbolic references (only direct references, i.e., addresses)

What Do We Really Do?



Intermediate Summary:

Specify:

source code: .y
object code: .yo

Implement:

your compiler: yc (in .y for self compilation)
your linker: yl (for separate compilation)
your emulator: ye (for controlling the gap!)

Motivation:

self compilation:

your compiler needs to be able to compile everything you use in the implementation of the compiler

this will teach you the semantics of programming languages in full detail and thus make you a better programmer!

separate compilation:

modularity and thus scalability

semantical gap:

understanding the trade-off between compilation and interpretation

Covered Topics

constants:
- natural numbers
- strings (arrays of characters)

no pointers! just references
→ no dot, just ->, [ ]

data:
- integer
- character
- boolean

} basic

- record
- array

} composite

- type-safe assignment (imperative language!)
- type-safe arithmetic and boolean expressions

control:
- if then else
- while loop

possibly nested!

structure:
- procedure with arguments and return value
- call by value (for basic types)
- call by reference (for arrays, records)
- local variables
- module that may import procedures, variables, and constants from other modules

no tools other than a compiler!

Systems Engineering

- bootstrapping:
  - use language for which there is a compiler
  - identify a subset that covers all topics
  - make subset "easily" parseable through grammar engineering and preprocessing (as last resort)
  - implement compiler in that subset

	compiler	output
time	compilation performance	execution performance
space	code size data size	code size data size

- optimizations:
  - only basics are covered, e.g. constant folding!
  - advanced topics if time permits

- error handling:
  - report as many actual errors as possible
  - always terminate, never crash

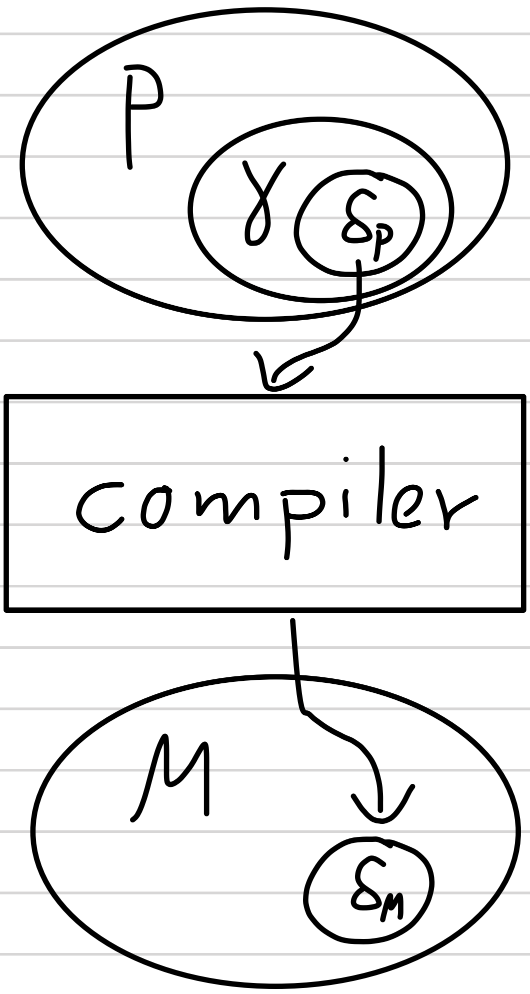
Advanced Topics

we construct a
a single-pass compiler

not covered:

- multi-pass compilation:
  ⇝ creates internal representation (IR)
- cross compilation:
  ⇝ generated output targets other machine than the one on which the compiler executes
- incremental compilation:

⇝ only compile local change δ_P, only considering local depending context γ, only generating δ_M

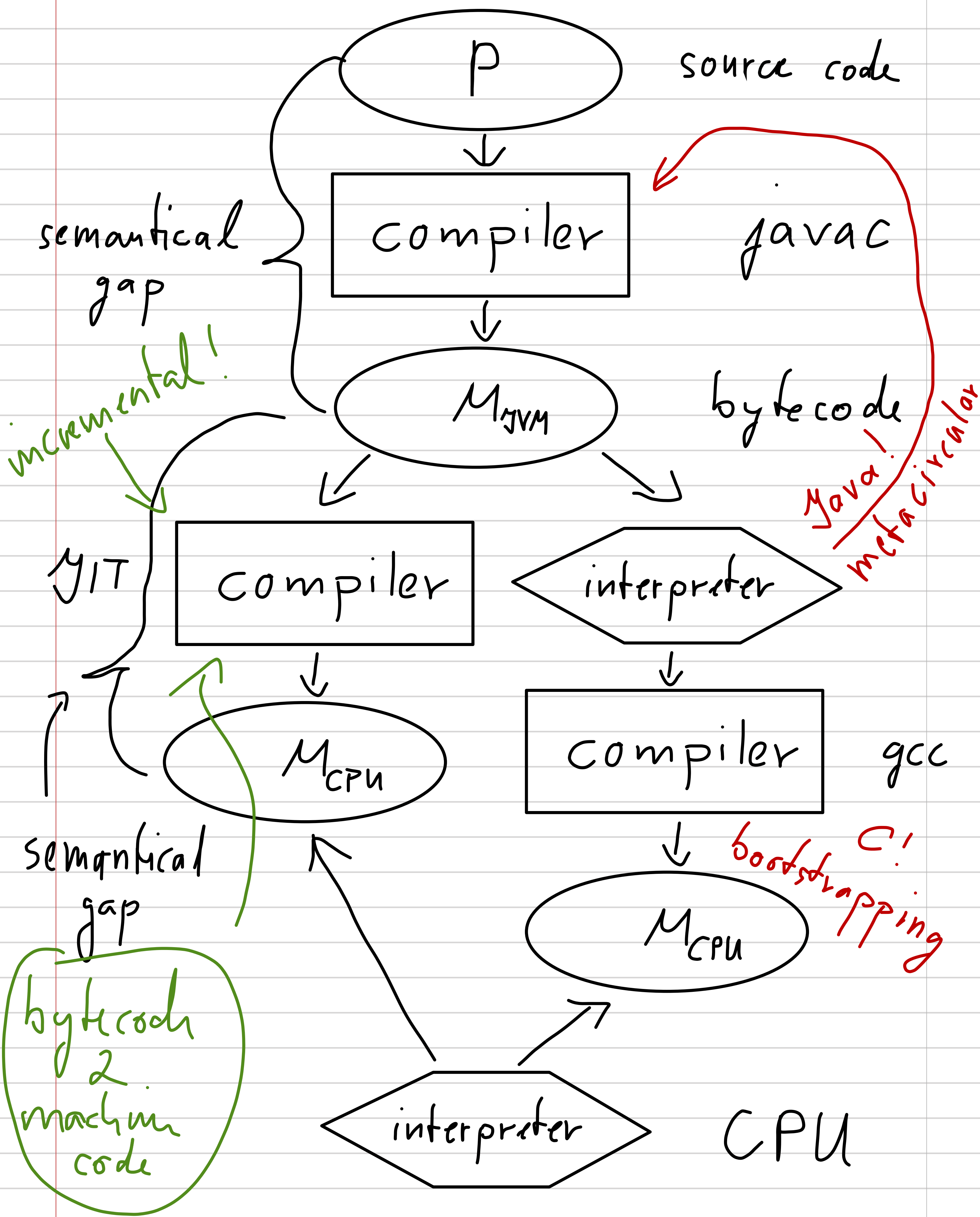


⇝ compilation complexity independent of program size!

- just-in-time (JIT) compilation

we only do ahead-of-time (AOT) compilation

Just-in-Time (JIT) Compilation



Summary:

- we construct:
  - a self-compiling, single-pass compiler
  - a static linker
  - a RISC emulator
- understand covered topics
- systems vs. software engineering

  systems ⇕ performance, software ⇕ reuse
- more things to do now:
  - play with existing systems (gcc, javac, qemu, ...)
  - set up your development environment
  - read books, papers, articles on compilers!