CS614: Advanced Compilers

Overview and Logistics

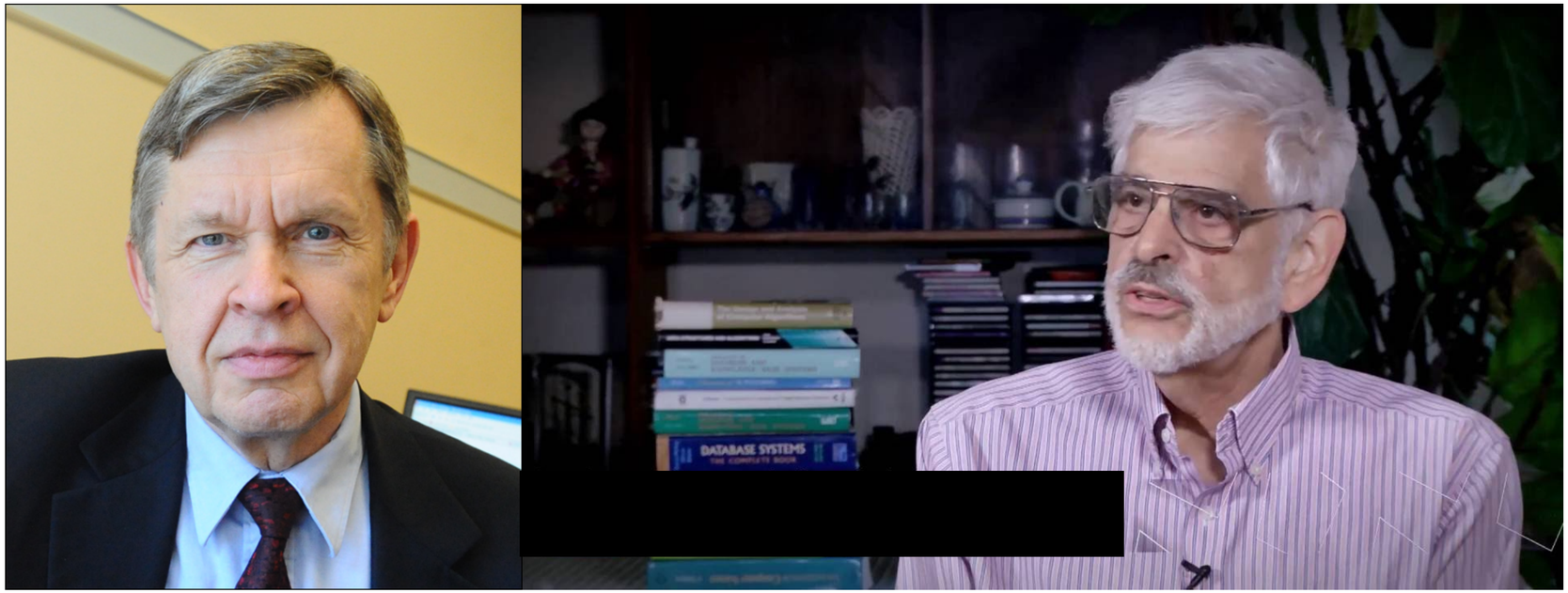
Manas Thakur
CSE, IIT Bombay

Spring 2025

Identify the legend

Grace Hopper
Credited to develop the first compiler (A), in 1952

Identify the legends



Alfred V Aho and Jeffrey D Ullman
Turing Award winners, 2020

Turing Award Winners from Compilers and PL

- **Alan Perlis** (1966, first ever Turing Award): For his influence in the area of advanced programming techniques and compiler construction.
- **Edsger Dijkstra** (1972): For fundamental contributions to programming as a high, intellectual challenge; for eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; for illuminating perception of problems at the foundations of program design.
- **Donald Knuth** (1974): For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.
- **John Backus** (1977): For profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on **FORTRAN**, and for seminal publication of formal procedures for the specification of programming languages.
- **Robert Floyd** (1978): For having a clear influence on methodologies for the creation of efficient and reliable software, and for helping to found the following important subfields of computer science: the theory of parsing, the semantics of programming languages, automatic program verification, automatic program synthesis, and analysis of algorithms.
- **Ken Iverson** (1979): For his pioneering effort in programming languages and mathematical notation resulting in what the computing field now knows as **APL**, for his contributions to the implementation of interactive systems, to educational uses of APL, and to programming language theory and practice.

Turing Award Winners from Compilers and PL

- **Tony Hoare** (1980): For his fundamental contributions to the definition and design of programming languages.
- **Niklaus Wirth** (1984): For developing a sequence of innovative computer languages, EULER, ALGOL-W, MODULA and **PASCAL**. PASCAL has become pedagogically significant and has provided a foundation for future computer languages, systems, and architectural research.
- **John Cocke** (1987): For significant contributions in the design and theory of compilers, the architecture of large systems and the development of reduced instruction set computers (RISC); for discovering and systematizing many fundamental transformations now used in optimizing compilers including reduction of operator strength, elimination of common subexpressions, register allocation, constant propagation, and dead code elimination.
- **Robin Milner** (1991): For three distinct and complete achievements: LCF, the mechanization of Scott's Logic of Computable Functions, probably the first theoretically based yet practical tool for machine assisted proof construction; **ML**, the first language to include polymorphic type inference together with a type-safe exception-handling mechanism; CCS, a general theory of concurrency. In addition, he formulated and strongly advanced full abstraction, the study of the relationship between operational and denotational semantics.
- **Ole-Johan Dahl and Kristen Nygaard** (2001): For ideas fundamental to the emergence of object-oriented programming, through their design of the programming languages Simula I and **Simula 67**.
- **Alan Key** (2003): For pioneering many of the ideas at the root of contemporary object-oriented programming languages, leading the team that developed **Smalltalk**, and for fundamental contributions to personal computing.

Turing Award Winners from Compilers and PL

- **Peter Naur** (2005): For fundamental contributions to programming language design and the definition of Algol 60, to compiler design, and to the art and practice of computer programming.
- **Fran Allen** (2006): For pioneering contributions to the theory and practice of optimizing compiler techniques that laid the foundation for modern optimizing compilers and automatic parallel execution.
- **Barbara Liskov** (2008): For contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.
- **Alfred Aho and Jeffey Ullman** (2020): For fundamental algorithms and theory underlying programming language implementation and for synthesizing these results and those of others in their highly influential books, which educated generations of computer scientists.
- **Next?**

- **16 out of 58 Turing Awards from Compilers and PL!! Even more to researchers* originally from CPL.**

*e.g. John McCarthy (**LISP**).

Also thanks to

B. Deshpande and V. Krishna Nandivada
For teaching me compilers

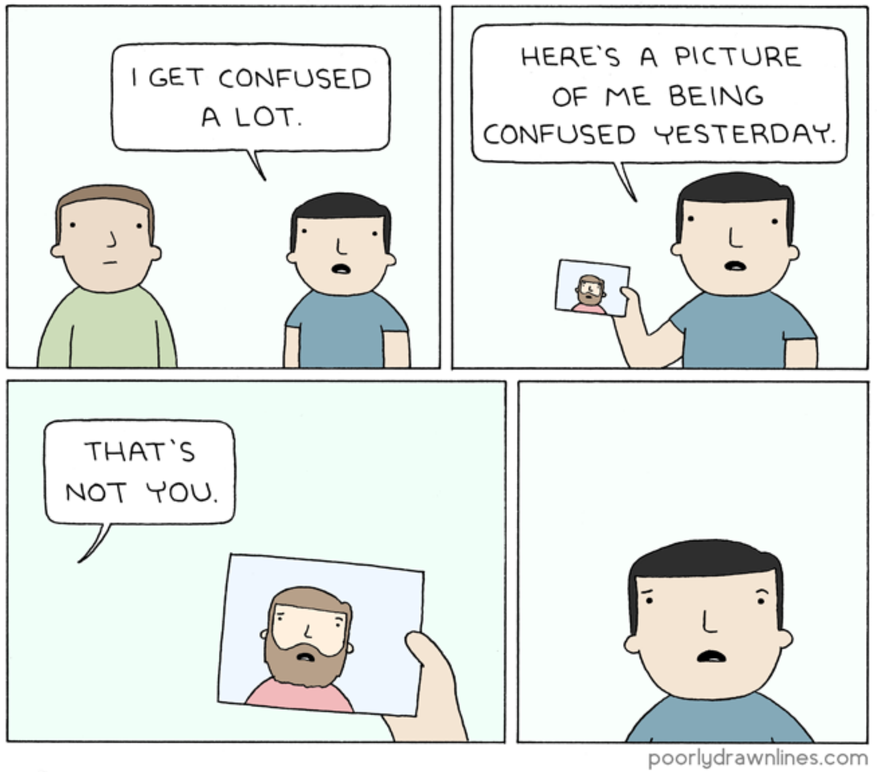
Rupesh Nasre and Yannis Smaragdakis
For allowing liberal use of material

Jan Vitek and Vijay Sundaresan
For collaborating on real compilers

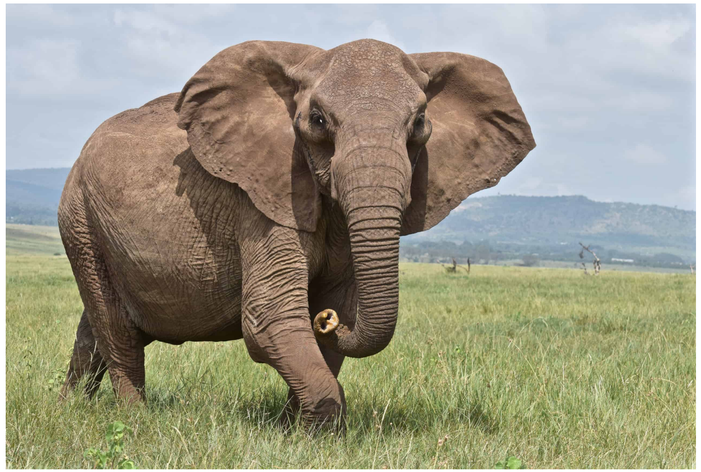
What is “Programming”?

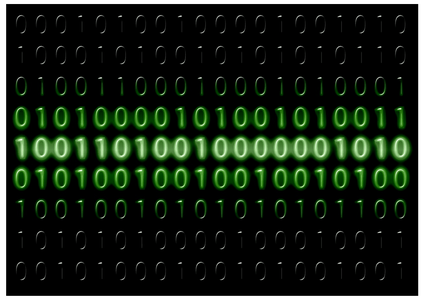
- A way to tell the computer what to do.
- But what does a computer understand?
  - Only binary.
  - With this definition, a program is only what one writes in binary.
- What have you been doing all throughout last K years in C, C++, Java, Python, Shellscript, etc then? :D

Confused?



A compiler “generates” programs





Let’s handle the elephant in the room.

High-level specification

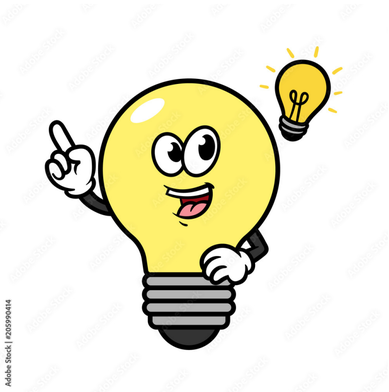
Compiler

Binary programs

LLMs vs Compilers

Why the name “Compiler”?

- People used to write libraries of subroutines (e.g., floating-point operations).
- If a programmer wanted to use two libraries, they would copy the subroutines for both by hand, change addresses manually, and use them in their applications.
- Grace Hopper added a “call” operation whereby
  - the machine would copy the code (later called a *loader*);
  - and update the addresses (later called a *linker*)



- The name “Compiler” was used because it put together (i.e., “compiled”) a set of subroutines!

In retrospect

- “I had a running compiler and nobody would touch it. They told me computers could only do arithmetic; they could not do programs.”
- “To programmers, it was obviously a foolish and arrogant dream to imagine that any mechanical process could possibly perform the mysterious feats of invention required to write an **efficient** program.”
- “We simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs.”

- See where we are now!

Compiler advancements in *recent* times

- New languages being designed
- New architectures being fabricated
- New compilers being developed
- New challenges being faced
- New solutions being researched

TNW

Could new programming language Mojo spark your career in AI and ML?

10 Sept • By Kirstie Mcdermott

InfoQ.com

Kotlin Reaches 2.1.0, Bringing New Language Features, Compiler Updates, and More

9 Dec • By Sergio De Simc

THENEWSTACK

Meta Releases Open Source React Compiler

Phoronix

17 May 2024 • By Loraine Lawson

Clang 20 Compiler Adds Support For Xtensa CPU Target

3 days ago • By Michael Larabel

InfoQ.com

Oracle Ships GraalVM Java JIT Compiler - but Only in Its Own JDK

4 Dec • By Karsten Silz

In this course

- **Overview of compilation.** Front-end recap. AST traversal using JavaCC and JTB. Object structures. Field and method resolution. IR generation for OO languages.
- **Intro to program optimization.** Control-flow graphs. Iterative dataflow analysis. Constant propagation. SSA form. SSA-enabled optimizations: conditional constant propagation, value numbering, partial redundancy elimination.
- **Interprocedural optimizations.** Call-graph construction. Class hierarchy analysis. Rapid type analysis. Pointer analysis. Devirtualization and method inlining.
- **Memory management.** Runtime layouts. Register allocation: Liveness and graph coloring, bitwidth-awareness.
- **Code generation.** Instruction selection. Pipelining and instruction scheduling.
- **Cache optimizations.** Data prefetching. Layout ordering. Loop transformations.
- **Parallelization.** Memory models. Loop-dependence analysis. MHP analysis.
- **Managed runtimes.** Dynamic compilation. Speculation and deoptimization.

Logistics & Evaluation

- Two lectures per week
  - Slot 9 (Mon and Thu 3:30-4:55 PM)
  - CC 103
- Office hours: Fri 4-5 PM (CC 308, 402)

Plagiarism:
No means No.

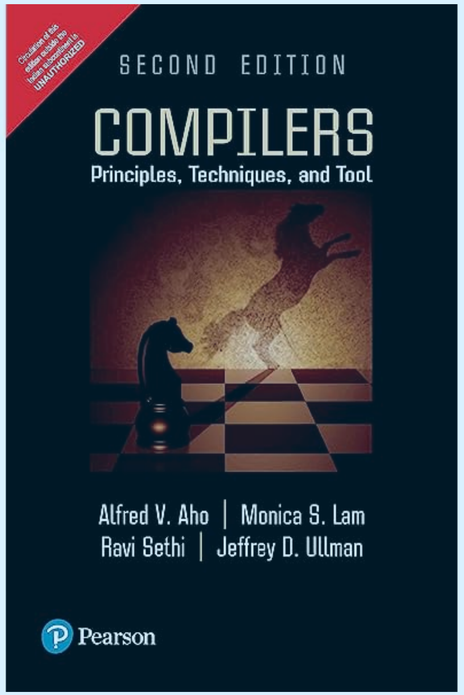
- First instance: Zero
- Second: One grade less
- Third: Fail and report

- Two exams (mid-sem, end-sem) — 25+25
- Four programming assignments — 10+15+10+10
  - Option to read papers in the second half — 20
- Ungraded* tutorials to help you in both theory and practice
- Discussion on Slack (cs614-spring25.slack.com)

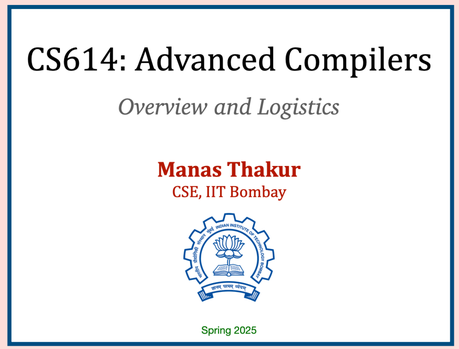
CS614 Slack

*Wait for 3 days.

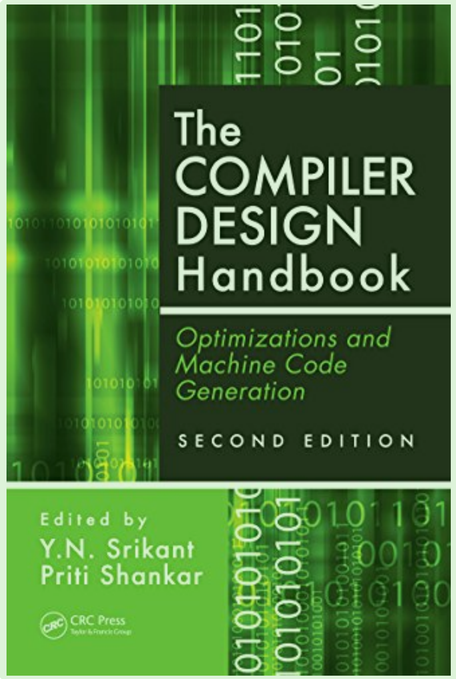
Learning Resources

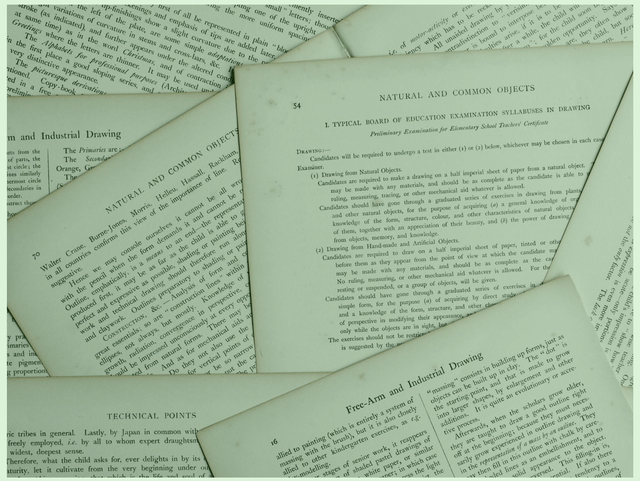


Recaps

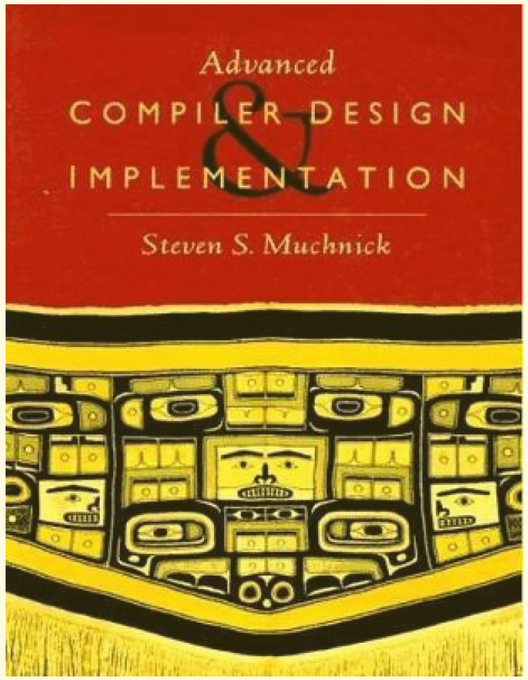


Slides: Necessary but not sufficient.





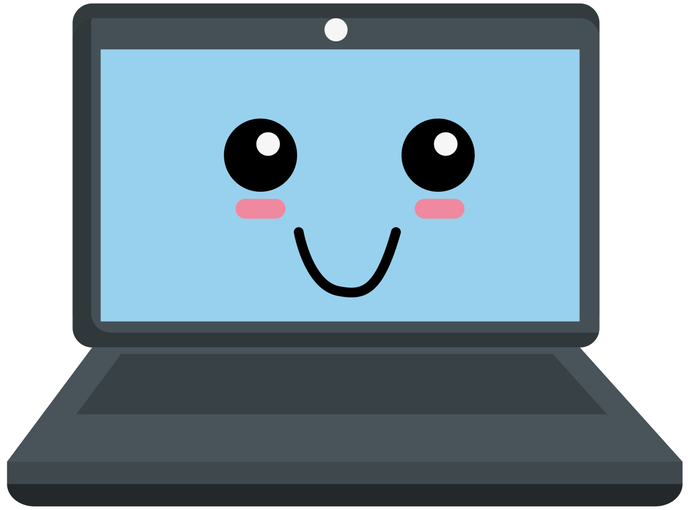
Advanced Topics



Representations and Optimizations

Attendance and Participation Credits

- You are expected to attend all the classes (heck, we just have about 27)
- No marks for attending classes, but there are **PCs** for participating actively
- Earning PCs:
  - Answer interesting questions, ask interesting questions
  - Make K-liners and memes, write blogs, share on Slack
- 1 PC *ki keemat*:
  - 12 hours assignment extension
  - 1 post-assignment mistake correction
  - 1 *cutting-chai-pe-charcha*
- Finally, **5 marks for course participation**

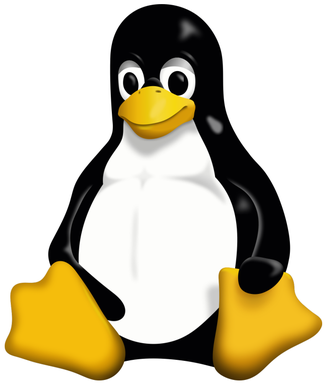


Your friends for programming assignments

- **Java**
  - If you know any OO, you'll be good in max a couple of weeks.

- An IDE: Will help you reduce mistakes
  - Eclipse/IntelliJ/VSCode (we encourage **Eclipse**).

- (Very) little bit of shell scripting.



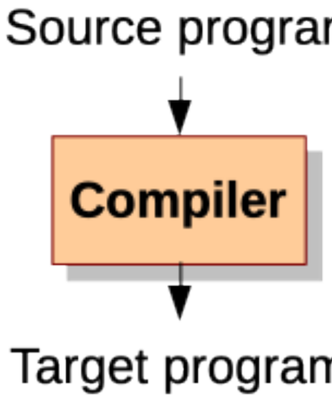
- OS: Though anyone should work, we would be able to support only Linux queries.

Activity Break (5 mins)

- Form groups of three and do the following:
  - Recall what does a compiler do at the lower level, along with its phases.
  - Find out one recent development about compilers and discuss:
    - its implications
    - association with its phases
    - future work

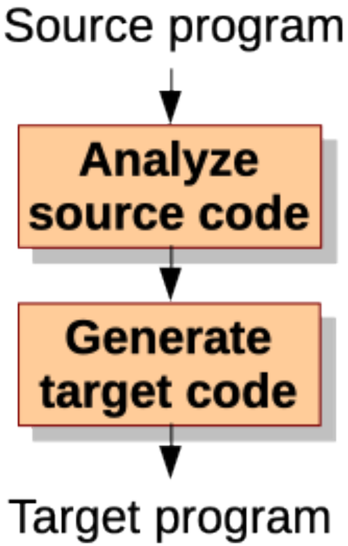
Overview of Compilation

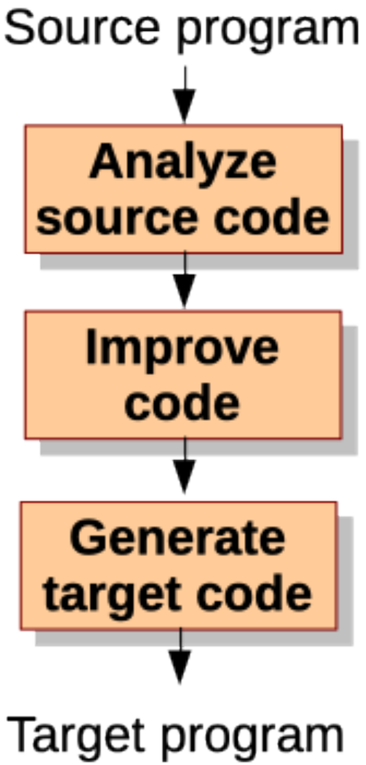
How many phases should a compiler have?



Black (Orange) box view

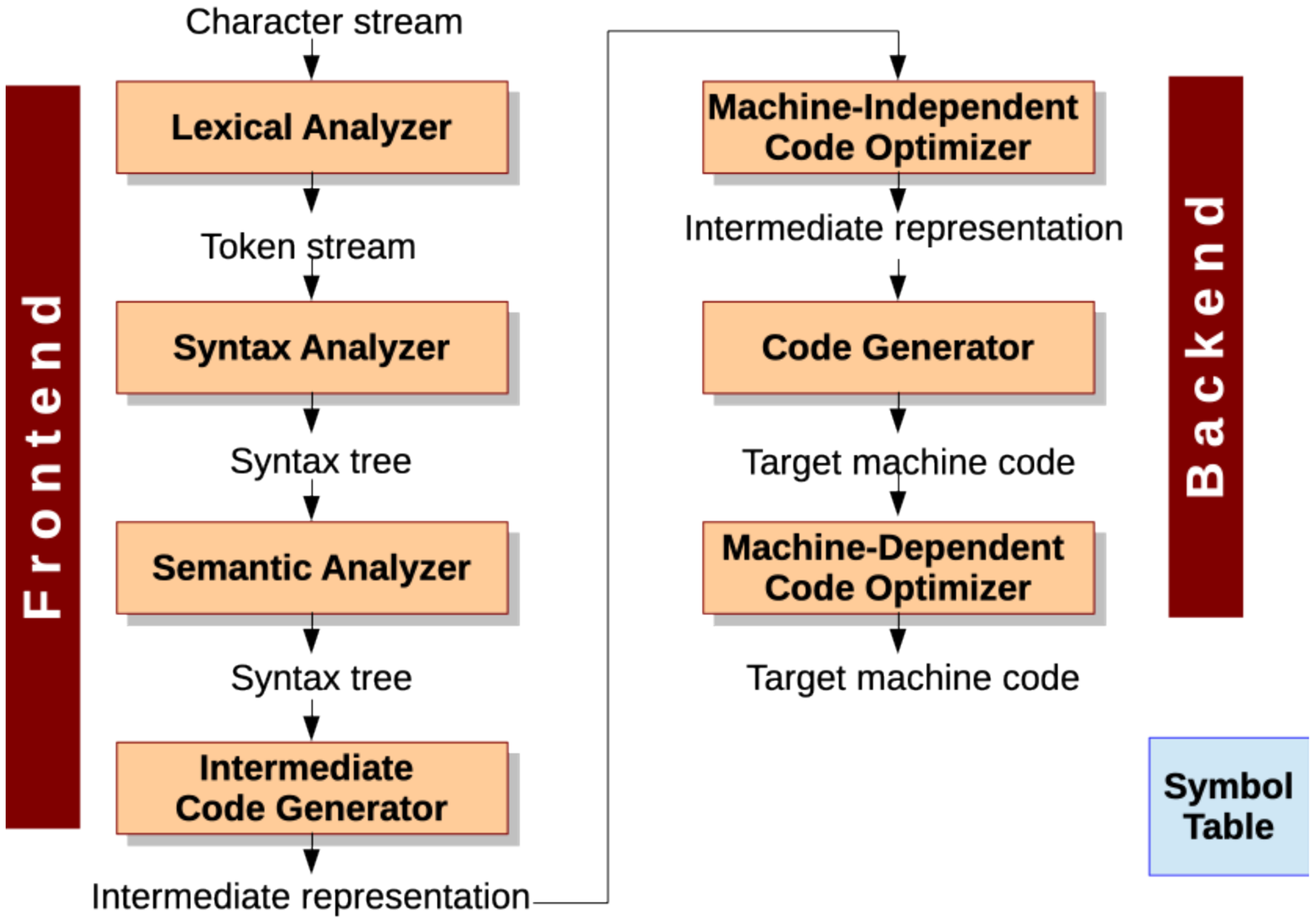
What we must have



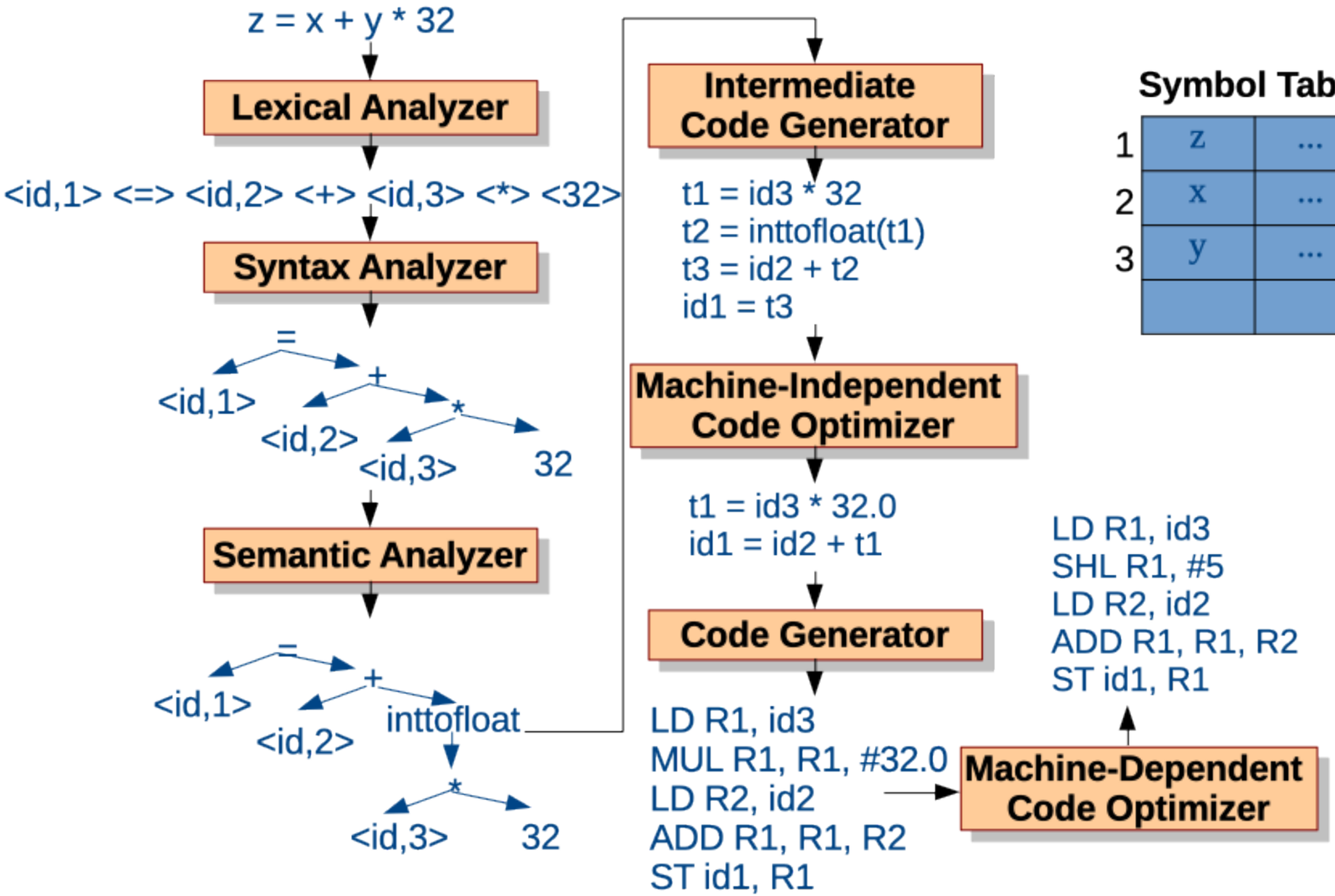


What we additionally have

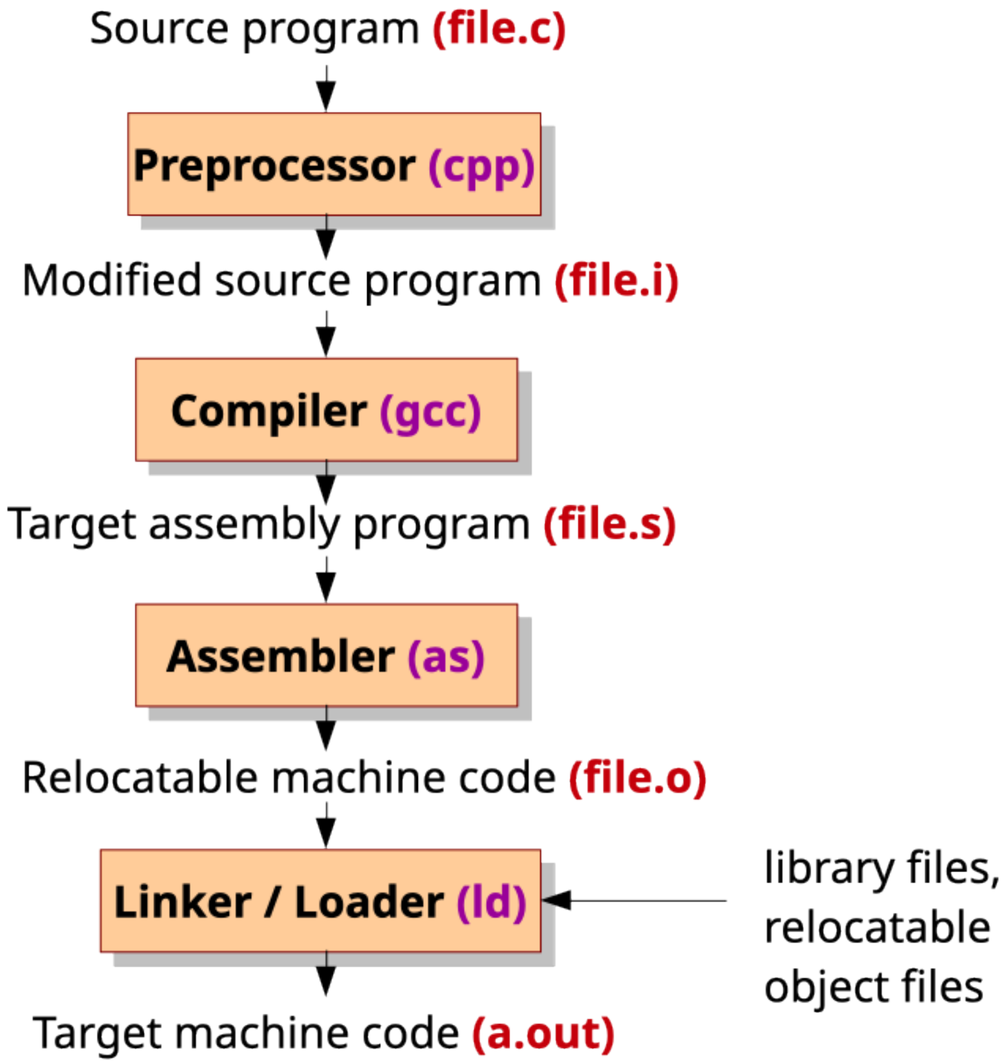
Typical phases in a compiler



Compilation phases: Example

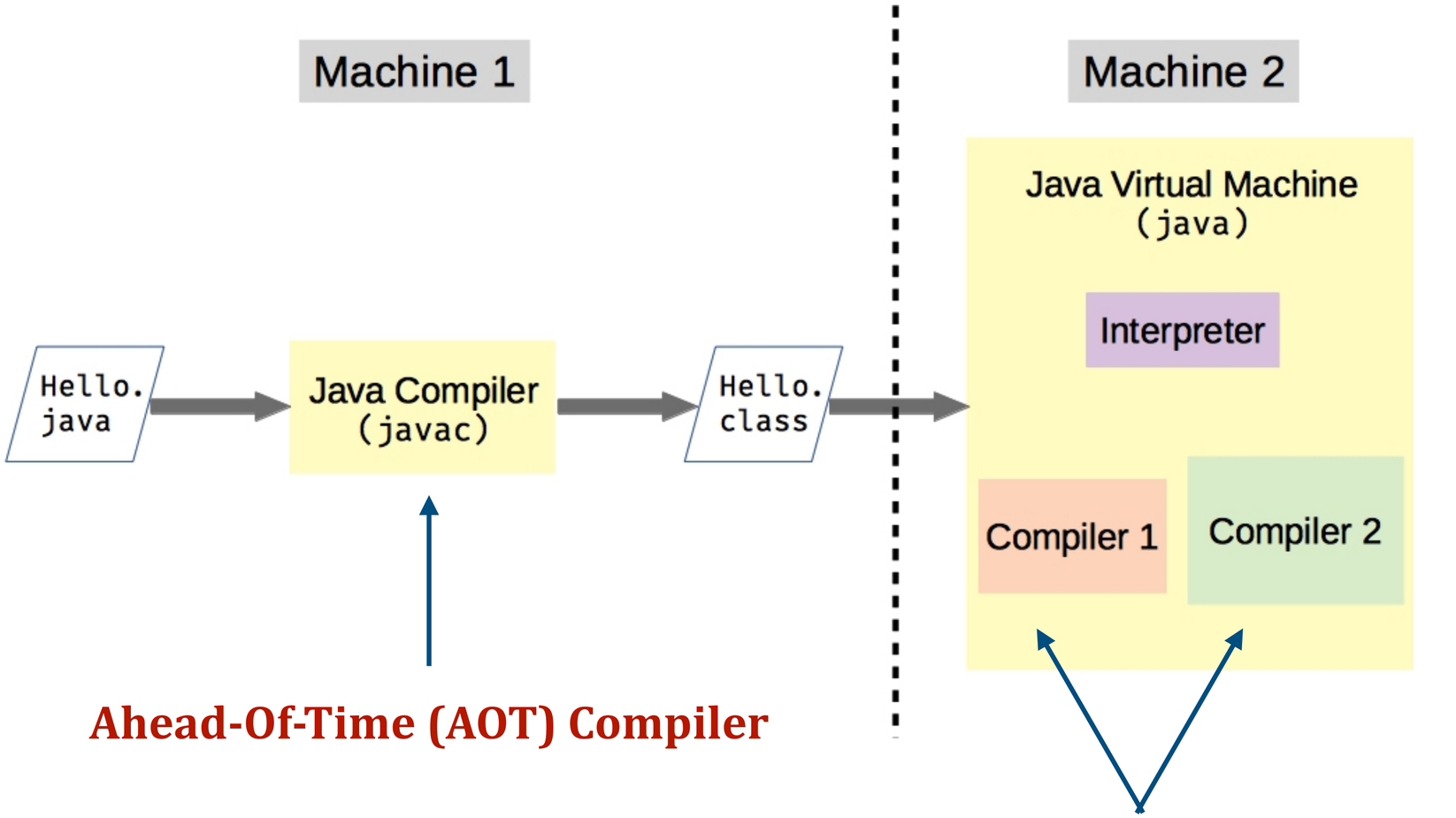


The Complete Picture for C



- **Discussion:**
  - Where does the OS come in?
  - Which parts can be done independent of the architecture?
  - Which parts can be done offline?
  - What happens for interpreted languages?
  - What happens for dynamic languages?
  - What are the tradeoffs?

The Partial Picture for Java



Demo

- Compilation
- Compilation vs interpretation
- Assembly generation
- Bytecode generation
- Just-in-time compilation

Day 1 Sign Off

- As compiler designers, we can play with (i.e., develop and improve) the techniques that play with code that is written to play with machines!

- Know how the food you eat is cooked, and learn more about compilers!

Next Class

Thursday, January 9th, 3:30 PM

CC 103

p.s. No, I cannot cook food.