COMP2010 — Compilers

Dr. Earl Barr

January 8, 2013

Who am I?

- Dr. Shin Yoo (shin.yoo@ucl.ac.uk)
  - Regression Testing, Search-based Software Engineering, Meta-heuristic Optimisation
  - Office: 66-72 Gower Street, Room G5
  - http://www.cs.ucl.ac.uk/staff/s.yoo

Shin Yoo

Research Interests

- Software Testing
  - Regression Testing, Debugging Aids, Information Theory
- Meta-heuristics, Optimisation and Machine Learning
  - Search-Based Software Engineering
- Gamefication
  - Unconventional Interface for SW Engineering

Teaching Assistant

Research

- Program Verification
- Termination Analysis
- Runtime Complexity Analysis
- Term Rewriting
- SAT Encoding
- Cryptographic Circuit Synthesis

Dr. Carsten Fuhs
c.fuhs@ucl.ac.uk

Lab: Gordon House 106, Tuesdays 10–11am, immediately after this lecture.

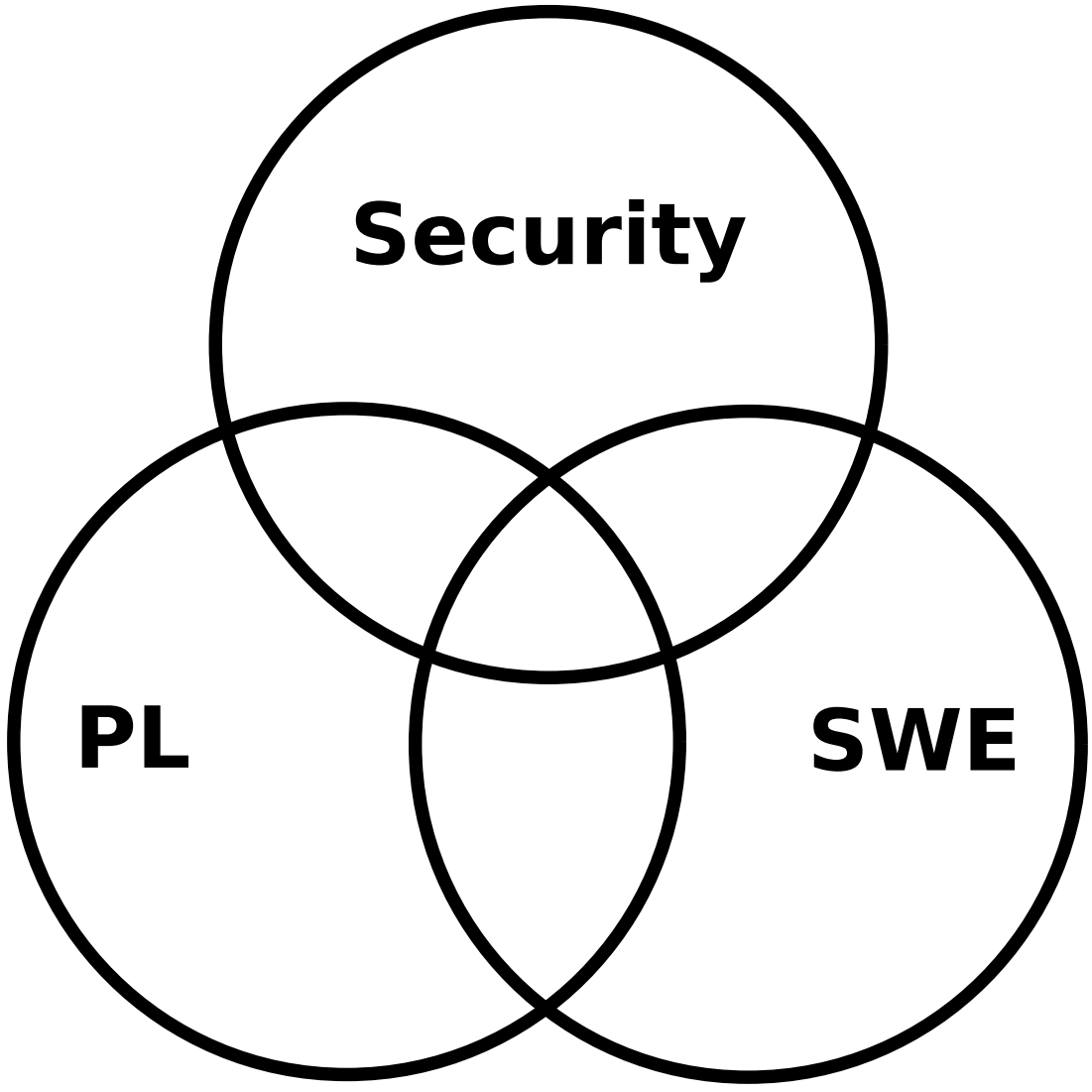
About Me

Dr. Earl Barr
7.07 Malet Place
e.barr@ucl.ac.uk
(0)20 7679 3570

www0.cs.ucl.ac.uk/people/E.Barr.html
and
earlbarr.com

Office Hours: Tuesday, 1–3PM and by appointment.

Research



Research

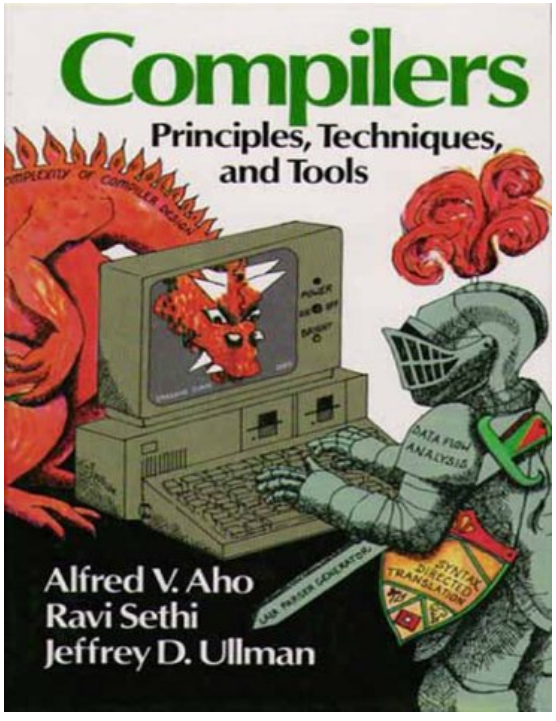
- Security
  - Online reputation, TrustDavis
  - The Great Firewall of China
  - Obfuscation
- Empirical Software Engineering
  - Measures
  - Version Control
- Program Analysis
  - Statistical (NLP)
  - Floating-point, Ariadne
  - Symbolic Execution

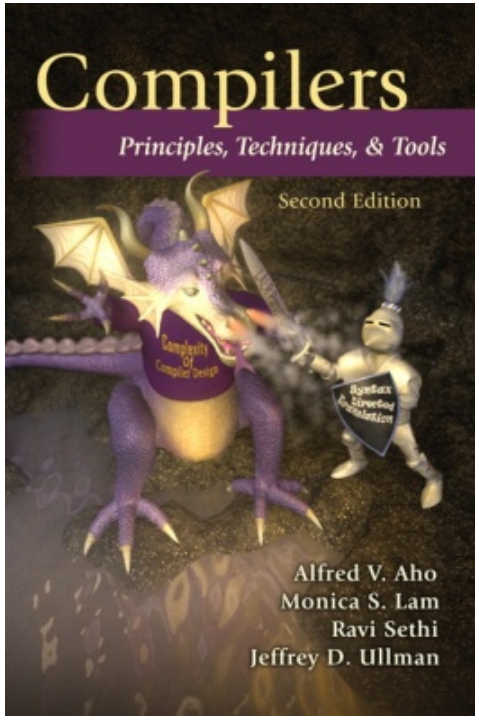
Course Material

- Lecture Slides
- Exercises/Problem Classes
- Code Examples

`https://moodle.ucl.ac.uk/course/view.php?id=10783`

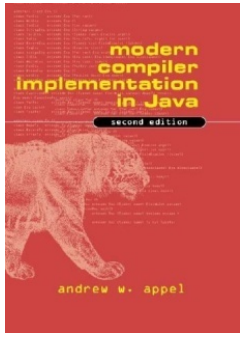
The "Dragon" Book





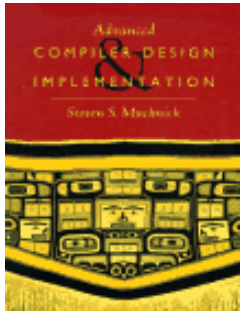
"Compilers — Principles, Techniques and Tools"
by A.V. Aho, R. Sethi, J.D. Ullman. Addison Wesley

Supplemental Books



"Modern Compiler Implementation in Java", by A.W. Appel. Cambridge University Press

"Modern Compiler Design", by D. Grune *et al.* John Wiley and Sons Ltd



"Advanced Compiler Design and Implementation", by S.S. Muchnick. Morgan Kaufmann

Selected PL Conferences

- Principles of Programming Languages (POPL)
- Programming Language Design and Implementation (PLDI)
- Compiler Construction (CC)
- OO Programming, Systems, Languages and Applications (OOPSLA/SPLASH)
- European Conference on OO Programming (ECOOP)
- Int'l Conference on Functional Programming (ICFP)
- Int'l Conference on Logic Programming (ICLP)

Tools

- JFlex — The Fast Scanner Generator for Java
  `http://www.jflex.de/`
- CUP — LALR Parser Generator for Java
  `http://www2.cs.tum.edu/projects/cup/`
- CLE — Eclipse plugin for editing CUP and JFlex files
  `http://cup-lex-eclipse.sourceforge.net/`

Carsten will discuss how to download, install and run these tools in the problem class that follows this lecture.

Assessment

- 20% Project
  - Form groups of 3 or 4 people
  - Email me the members
  - Part I: todo due date
  - Part II: todo due date
- 80% Examination (2.5 hours)

The problem classes will generally work through exercises similar to those you will see in the examination.

Questions?

Core Topics

- Compiler Phases
- Relationship between
  - syntax and semantics
  - source and machine code.
- Tools, Techniques, and Data Structures
  - Finite State Automata, aka machines, (FSA/FSM)
  - Context Free Grammar (CFG)
    - Grammar
    - Parsing
  - FSA *vs.* CFG
  - Abstract syntax tree (AST)
  - The "Visitor" design pattern

Linux Startup

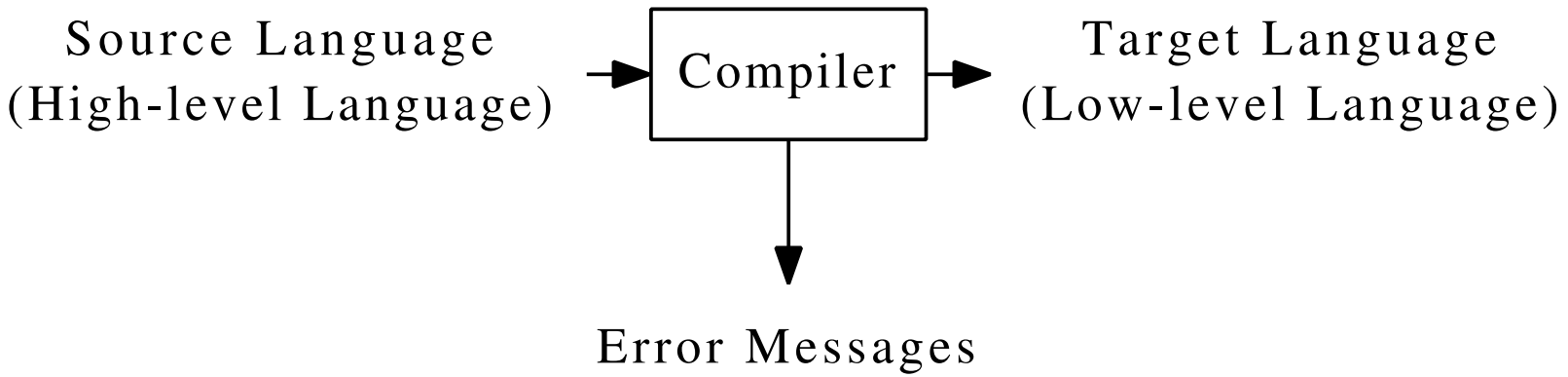
In the beginning, there were two BSD and SYSV. Their union is sysvinit: init is the first process; it checks runlevel from /etc/ inittab , then runs

```
/etc/rc.d/rc.sysinit # that runs the files in
/etc/rc.d/<runlevel> # which are symlinks to init.d
/etc/rc.d/init.d/<many scripts> # then finally runs
/etc/rc.local # the conventional place for customization.

debian/ubuntu uses /etc/rc<runlevel>.d

systemd
systemctl
/etc/systemd/system/<units> /usr/lib/systemd/system/<units>.

upstart uses sysvinit directories, with *manual* stanza.
```

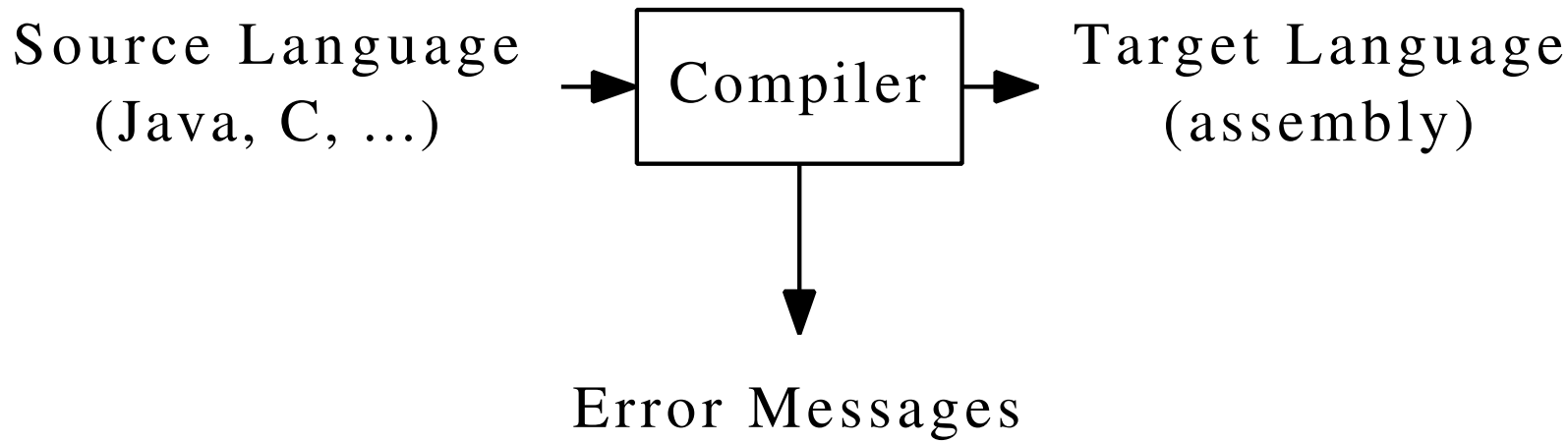
On example: on this system, which normally runs ubuntu, /etc/ init .d/x11−common starts X11.

Shell startup: /etc/ profile

Compilers translate from a high-level to a low-level programming language.



Compilers *typically* translate from a high-level programming language to assembly.



Most compilers target assembly because it's easier to produce and debug the compiler. Exceptions include just in time (JIT) compilers, first generation C++ compilers, and Microsoft's Visual Studio.

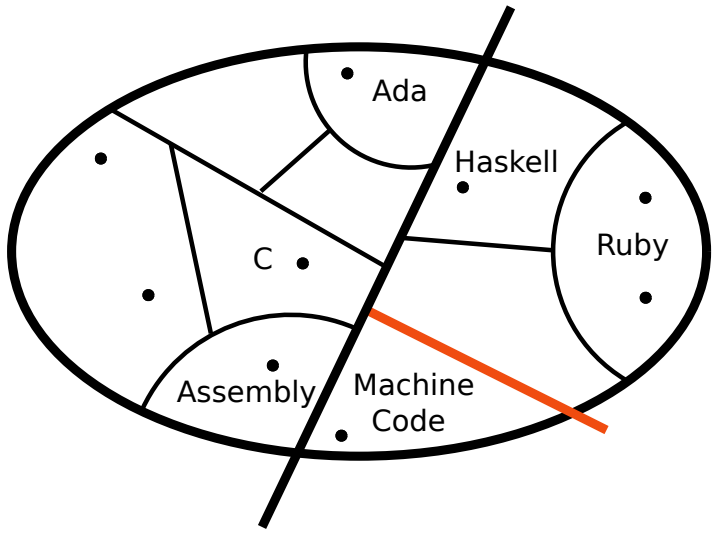
Example Assembly Listing

Label	Directive / Instruction (GAS)	Object Code (Not Actual)
.data		
hw:	.string "hello world\n"	00000010 10000000 01100110 00000110
		01100010 10000000 00000000 00000000
.text		
.globl _start		
_start:		
	movl $SYS_write, %eax	10000010 10000000 01111111 11111100
	movl $1, %ebx	10001000 10000000 01000000 00000010
	movl $hw, %ecx	11001010 00000001 00000000 00000000
	movl $12, %edx	00010000 10111111 11111111 11111011
	int $0x80	10000110 10000000 11000000 00000101
	movl $SYS_exit, %eax	10000001 11000011 11100000 00000100
	xorl %ebx, %ebx	00000000 00000000 00000000 00010100
	int $0x80	00000000 00000000 00001011 10111000

Questions

- What languages do you know?

The Geometry of Programs



All Programs

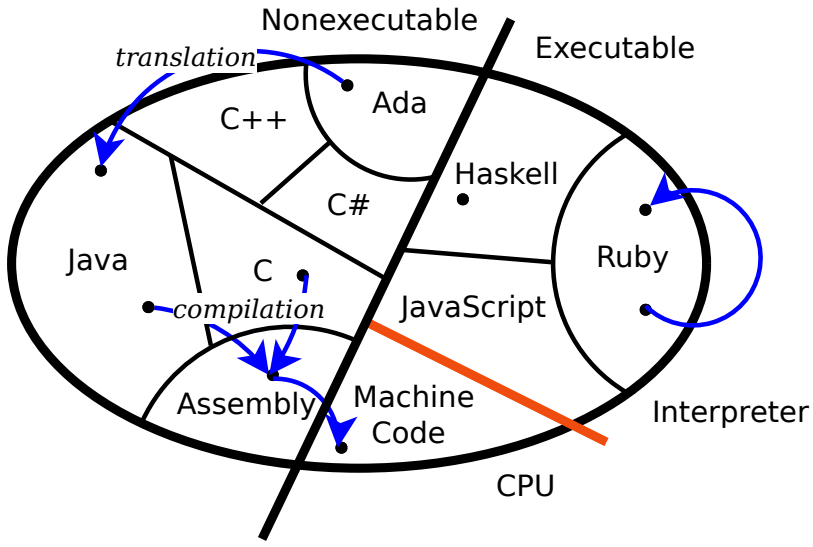
Questions

- What languages do you know?
- Are the languages disjoint?

Questions

- What languages do you know?
- Are the languages disjoint?
- What does the heavy black line mean?

The Geometry of Programs

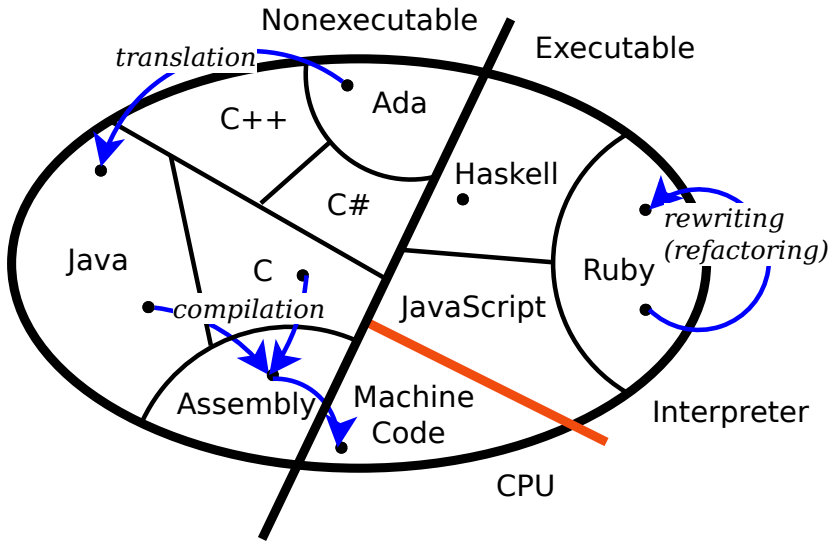


All Programs

Questions

- What languages do you know?
- Are the languages disjoint?
- What does the heavy black line mean?
- What does the self-loop mean?

The Geometry of Programs



All Programs

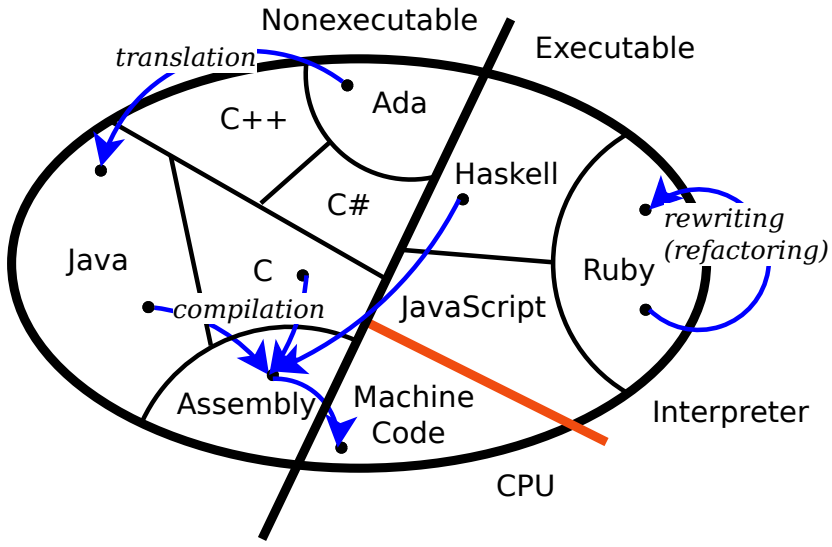
Questions

- What languages do you know?
- Are the languages disjoint?
- What does the heavy black line mean?
- What does the self-loop mean?
- Is the distinction between executable and nonexecutable inherent to a language?

Questions

- What languages do you know?
- Are the languages disjoint?
- What does the heavy black line mean?
- What does the self-loop mean?
- Is the distinction between executable and nonexecutable inherent to a language?
- Are executable and nonexecutable mutually exclusive?

The Geometry of Programs



All Programs

Why Do We Need Compilers?

- To raise the abstraction level.
- To find a middle ground between human minds and machine execution.
- To avoid the difficulty of writing and maintaining programs written in assembly.

In short, to bridge the semantic gap.

Bridging the Semantic Gap

Why Are Compilers Special?

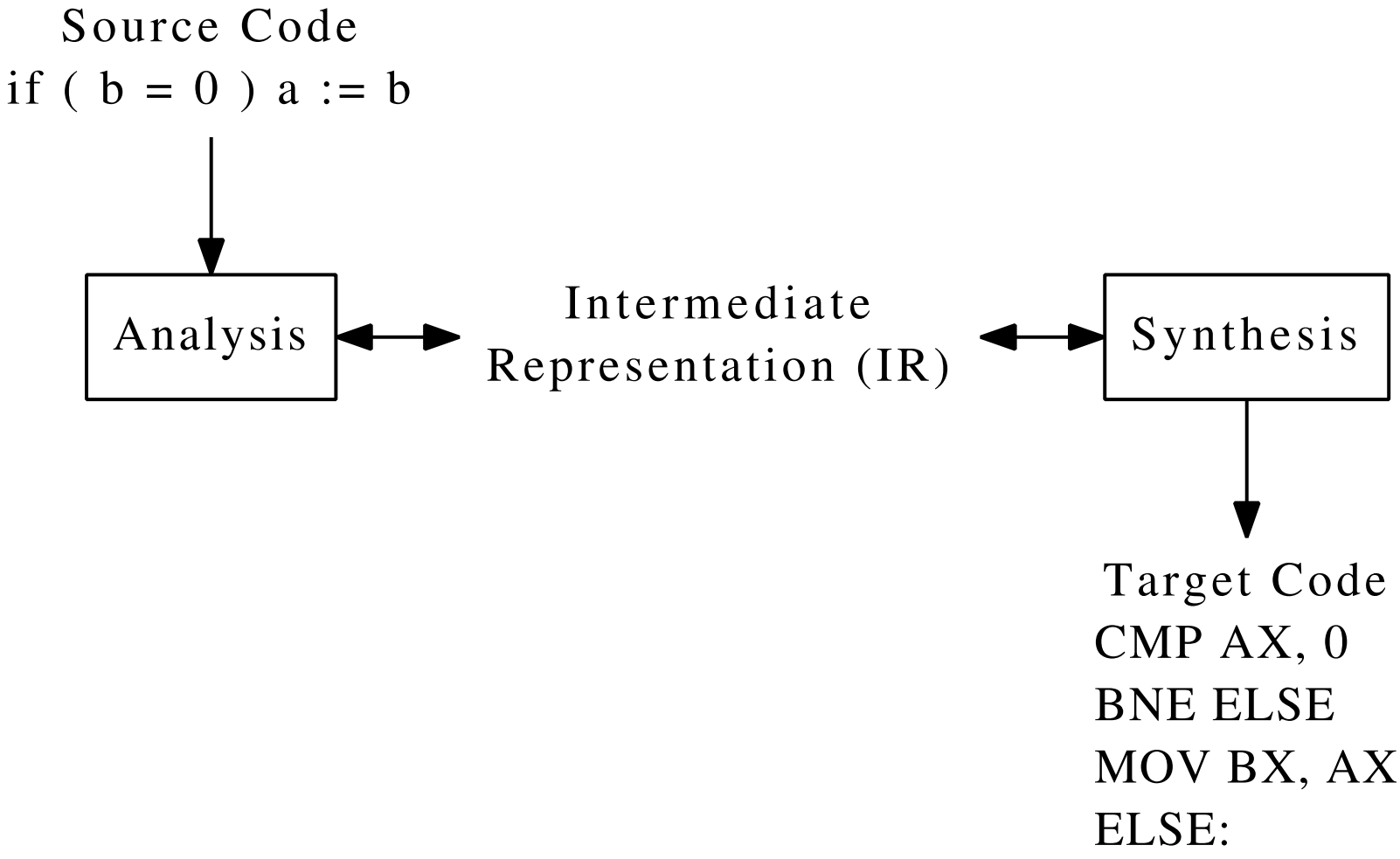
Correctness

Compilers must translate a source code into *equivalent* machine code.

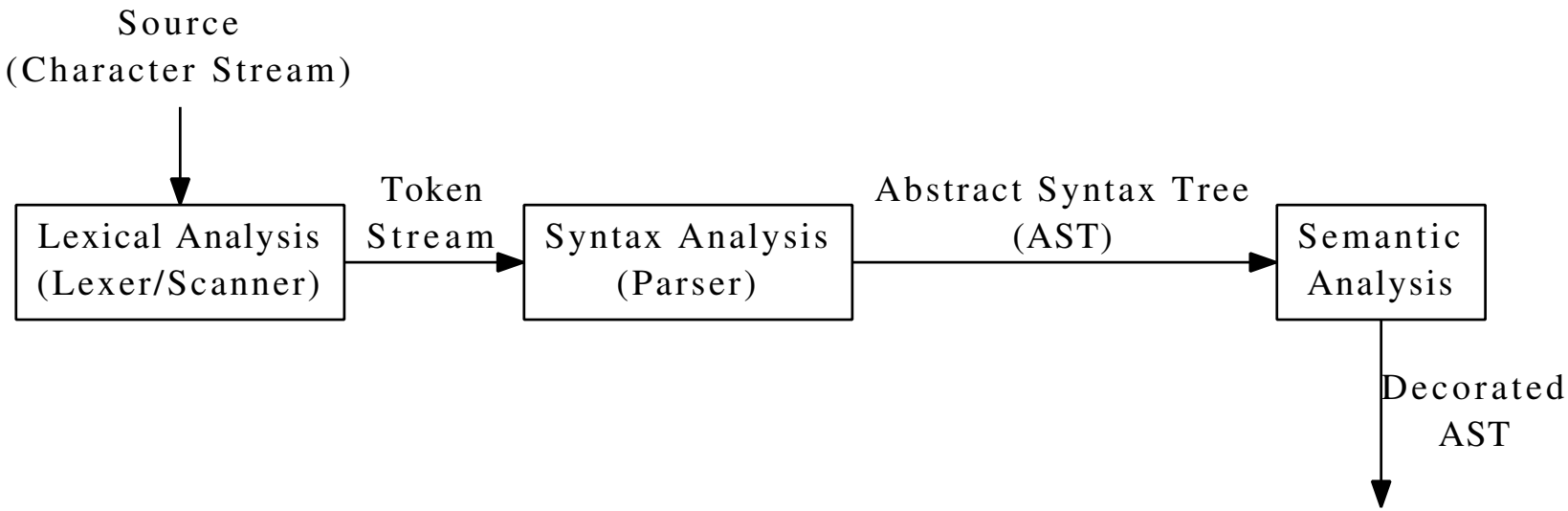
Historically, translation "efficiency" was a goal. I believe that progression of Moore's law has replaced that goal with

- quality error messages
- automatic parallelization
- verification
- space and time-to-completion efficiency

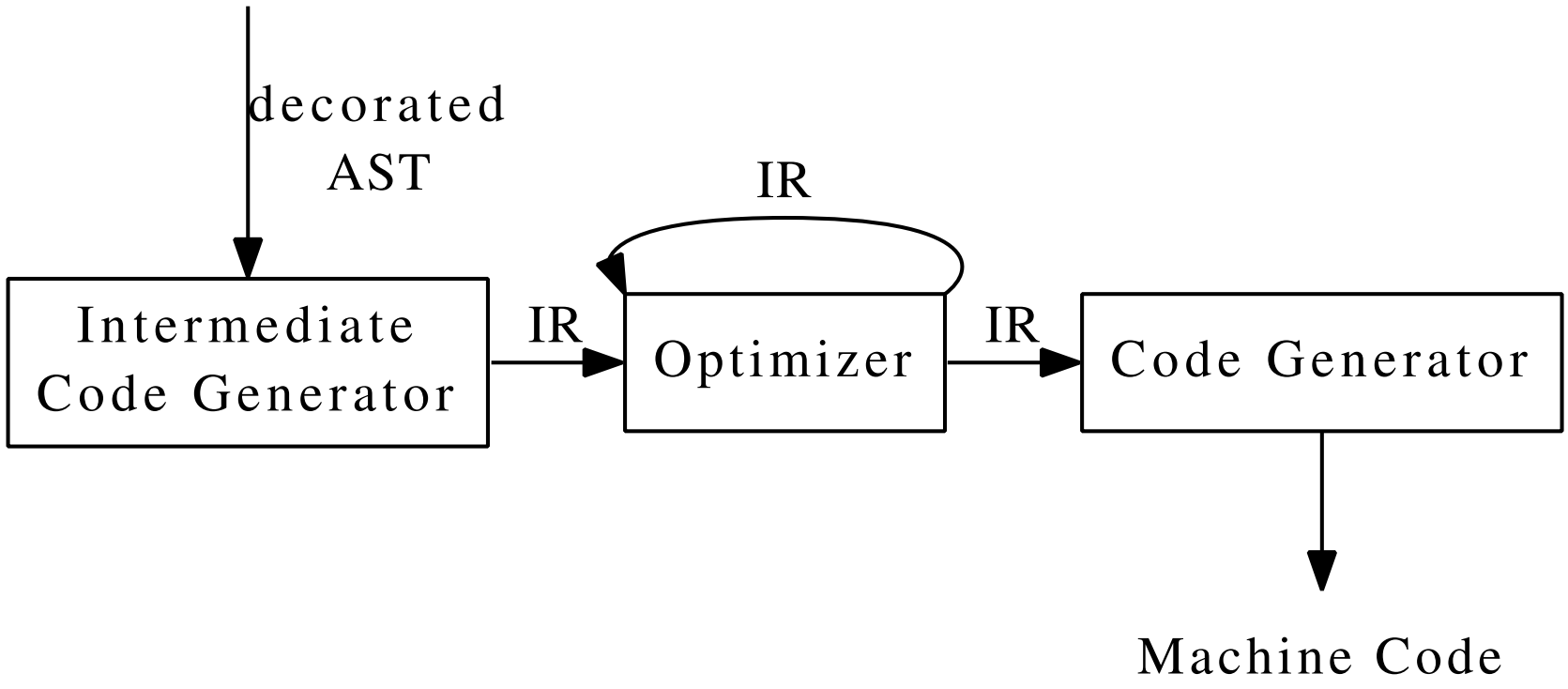
High-level Compiler Structure



Front-End


Source
(Character Stream)
Lexical Analysis
(Lexer/Scanner)
Token
Stream
Syntax Analysis
(Parser)
Abstract Syntax Tree
(AST)
Semantic
Analysis
Decorated
AST

Back-End


decorated
AST
IR
Intermediate
Code Generator
IR
Optimizer
IR
Code Generator
Machine Code

Lexical Analysis

Goal

Extract lexemes from source code.

- "I like science fiction"
  $\Rightarrow$ "I" "like" "science" "fiction"
- "if (a = 0) a := b"
  $\Rightarrow$ "if" "(" "a" "=" "0" ")" "a" ":=" "b"

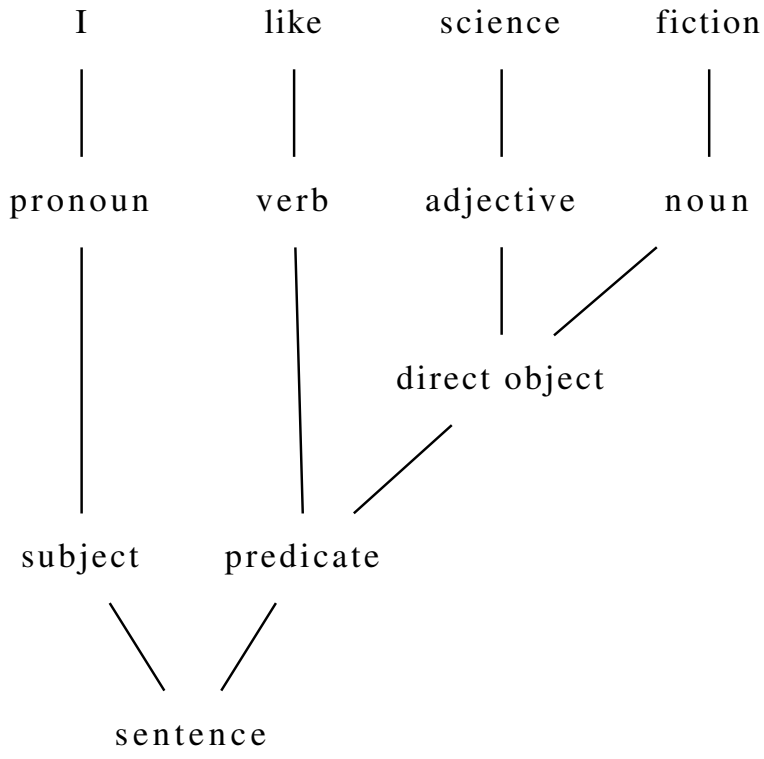
Note on terminology

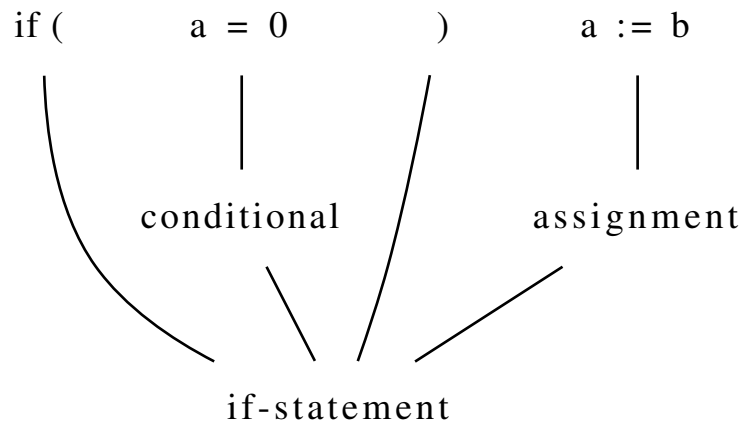
Intuitively, a lexeme is a word or symbol in the source. In linguistics, token is a synonym for lexeme, but, traditionally in computer science, token denotes a partition over lexemes, which linguists call a "token type".

Syntax Analysis

Goal

Recognize phrase structure.





Semantic Analysis

Goal

Semantically validate the input source code.

Science	fiction	likes	me.
adjective	noun	verb	pronoun/direct object

Semantic Analysis

Goal

Semantically validate the input source code.

Science	fiction	likes		me.
adjective	noun	verb	pronoun/direct object	

The syntax is correct, but the semantics is wrong.

Semantic Analysis

Goal

Semantically validate the input source code.

Science	fiction	likes	me.
adjective	noun	verb	pronoun/direct object

The syntax is correct, but the semantics is wrong.

if (a = 0)	a := b
conditional	assignment

Semantic Analysis

Goal

Semantically validate the input source code.

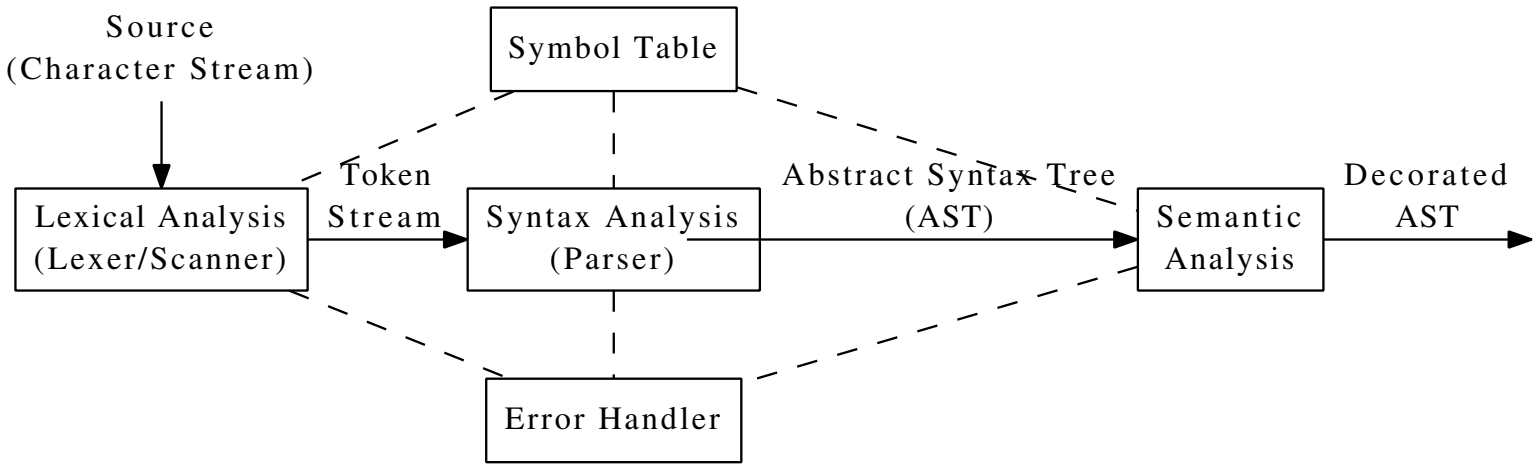
Science	fiction	likes	me.
adjective	noun	verb	pronoun/direct object

The syntax is correct, but the semantics is wrong.

if (a = 0)	a := b
conditional	assignment

What if "int a;" precedes the if? What if "String a;" does?

Error Handling and Symbol Table



Intermediate Code Generator

Goal

Produce intermediate code.

- Portable: machine- and language-independent
- Cleaner separation of front- and back-ends
- Ease of cross-compilation and optimization

Intermediate Representation

Three levels: High IR (HIR), Mid IR (MIR) and Low IR (LIR)

3-address code, aka TAC or 3AC:

```
if (a < b + c)                    _t1 = b + c;
    a = a - c;                    _t2 = a < _t1;
c = b * c;                        IfZ _t2 Goto _L0;
                                  _t3 = a - c;
                                    a = _t3;
                              _L0:  _t4 = b * c;
                                    c = _t4;
```

Other Examples: LLVM IR, GCC's Gimple.

Optimizer

Goal

Rewrite intermediate code to run faster or use less space.

	Unoptimized	Optimized
	`_t1 = b * 0;`	
	`a = 1;`	`a = 1;`
	`_t2 = a < _t1;`	
	`IfZ _t2 Goto _L0;`	
	`_t3 = a - c;`	
	`a = _t3;`	
`_L0:`	`_t4 = b * c;`	`c = b * c;`
	`c = _t4;`	

In this snippet of code from a program, can the optimizer drop "a = 1;"?

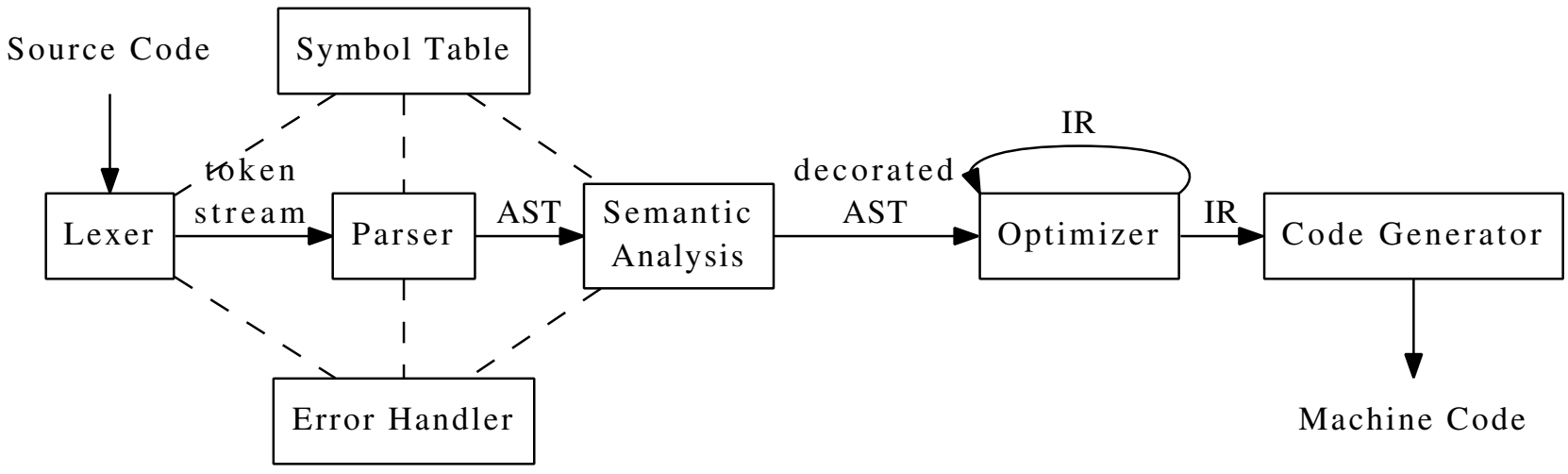
Code Generator

Goal

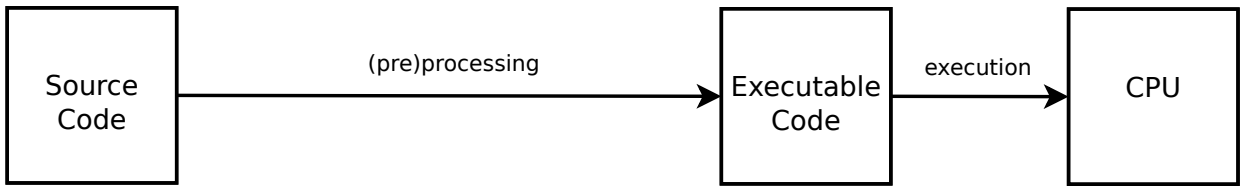
Output target program (usually assembly code).

Label	Directive / Instruction (GAS)	Object Code (Not Actual)
.data		
hw:	.string "hello world\n"	00000010 10000000 01100110 00000110
		01100010 10000000 00000000 00000000
.text		
.globl _start		
_start:		
	movl $SYS_write, %eax	10000010 10000000 01111111 11111100
	movl $1, %ebx	10001000 10000000 01000000 00000010
	movl $hw, %ecx	11001010 00000001 00000000 00000000
	movl $12, %edx	00010000 10111111 11111111 11111011
	int $0x80	10000110 10000000 11000000 00000101
	movl $SYS_exit, %eax	10000001 11000011 11100000 00000100
	xorl %ebx, %ebx	00000000 00000000 00000000 00010100
	int $0x80	00000000 00000000 00001011 10111000

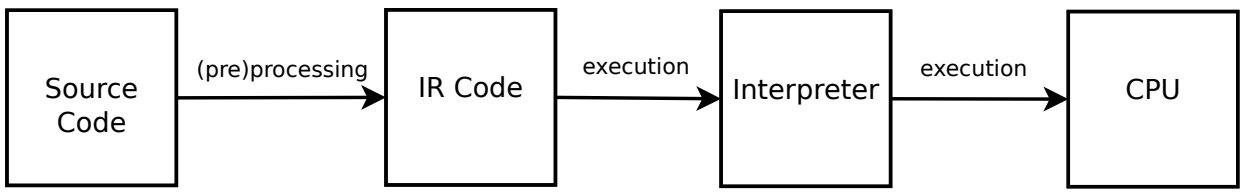
All Phases


Source Code
Symbol Table
token
stream
Lexer
Parser
AST
Semantic
Analysis
decorated
AST
IR
Optimizer
IR
Code Generator
Machine Code
Error Handler

To Compile or to Interpret



Compilation



Interpretation

Questions?

Questions for you

- What's the read evaluate print loop, or REPL?
- Can you think of other opportunities for hybridizing compilers and interpreters?
- Why not always interpret? Put differently, why compile?
- I said that compilation "naturally" decomposes into the phases we've discussed today. "Naturally" is a loaded term; after this lecture and you've thought about it, can you tell me if you agree at the start of the next lecture?

This class is about compilers, *not* interpreters.
Questions concerning interpreters will not appear again.