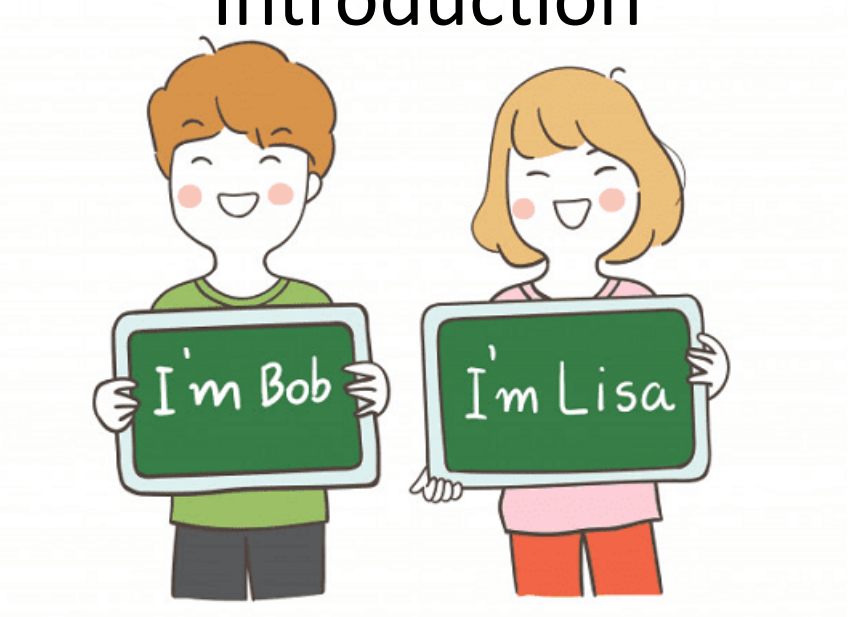
Compilation

0368-3133

Lecture 1a:

Introduction



Slides credit: Shachar Itzhaky, Eran Yahav, Mooly Sagiv

Administrata

Staff

- Lecturer: Noam Rinetzky
  - maon@cs.tau.ac.il
  - Check Point 342
  - Office hours: set by mail
- T.A.: Noa Schiller

Lectures & Recitations

- Lecture (1): Tuesday 9-12 ➔ “Theory”
- Recitations (3): Thursday 16-19 ➔ “Practice”

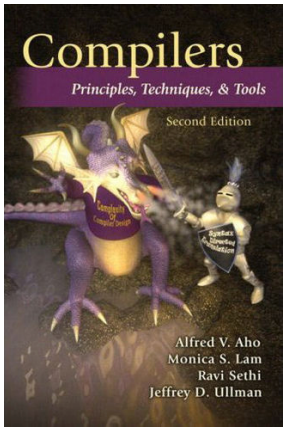
- Physical participation is highly recommended!
  - Limited online participation (no Q/A)*
  - Videos will be available

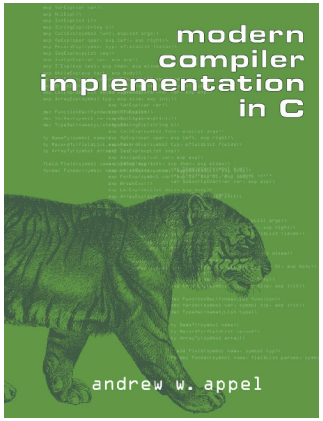
<<

*Indiercly via chats ➔ friends usually works

Textbooks

- Modern Compiler Design
  - Grune et al.
- Compilers: Principles, Techniques and Tools
  - Aho et al.
- Modern Compiler Implementation in Java/C/ML
  - Andrew W. Appel





Practical Compiler Project

- Build a compiler ***L*** *(an OOP language)*⇒ ***MIPS***
  - In Java
  - Groups of 3
  - Managed by the T.A.

Grades

- Compiler project 40%
  - Ex 1: 3% (lexical analysis)
  - Ex 2: 4% (syntax analysis)
  - Ex 3: 8% (semantic analysis)
  - Ex 4: 10% (intermediate representation + data-flow analysis)
  - Ex 5: 15% (code generation)
- Final exam 60% - must pass!

Workload—A Heads Up

- Compiler pr
  - Ex 1: 3%
  - Ex 2: 4%
  - Ex 3: 8% (semantic analysis)
  - Ex 4: 10%
  - Ex 5: 15% (code generation)
- Final exam 6

The compiler project may be the most significant programming task in your studies

Grade ~ log(|Work|)

You will eat your own dog food!

Students in Reserve Duty (Group B/C)

- Can submit only some of the exercises
  - Exercise percentage moves to exam

- Can join 3-person groups
  - i.e., groups of 4

Students in Reserve Duty (Group C)

- Two options for alternative evaluation:
  - Home exam* (עובר בינארי)
  - Home exam* + do Ex 5 alone (ציון רגיל)

*Home exam = Class exam in 7 days

Course Overview

Modern Compiler Design. Chapter 1

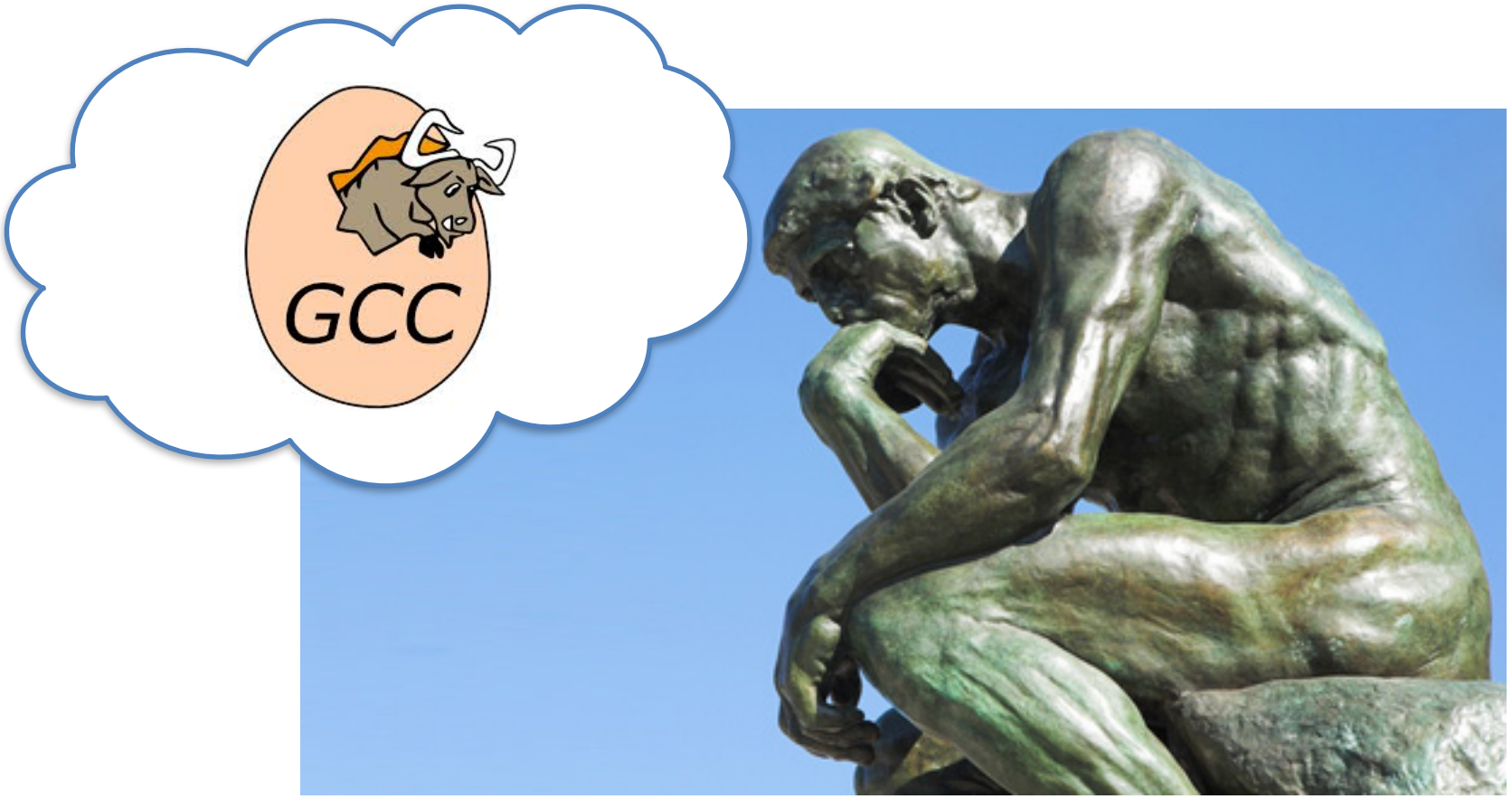
Course Goals

- What is a compiler
- How does it work
- (Reusable) techniques & tools

- Programming languages implementation
  - Procedure calls, polymorphism, runtime systems (e.g., memory management), ...
- Execution environments
  - Assemblers, linkers, loaders, OS

To compilers, and beyond

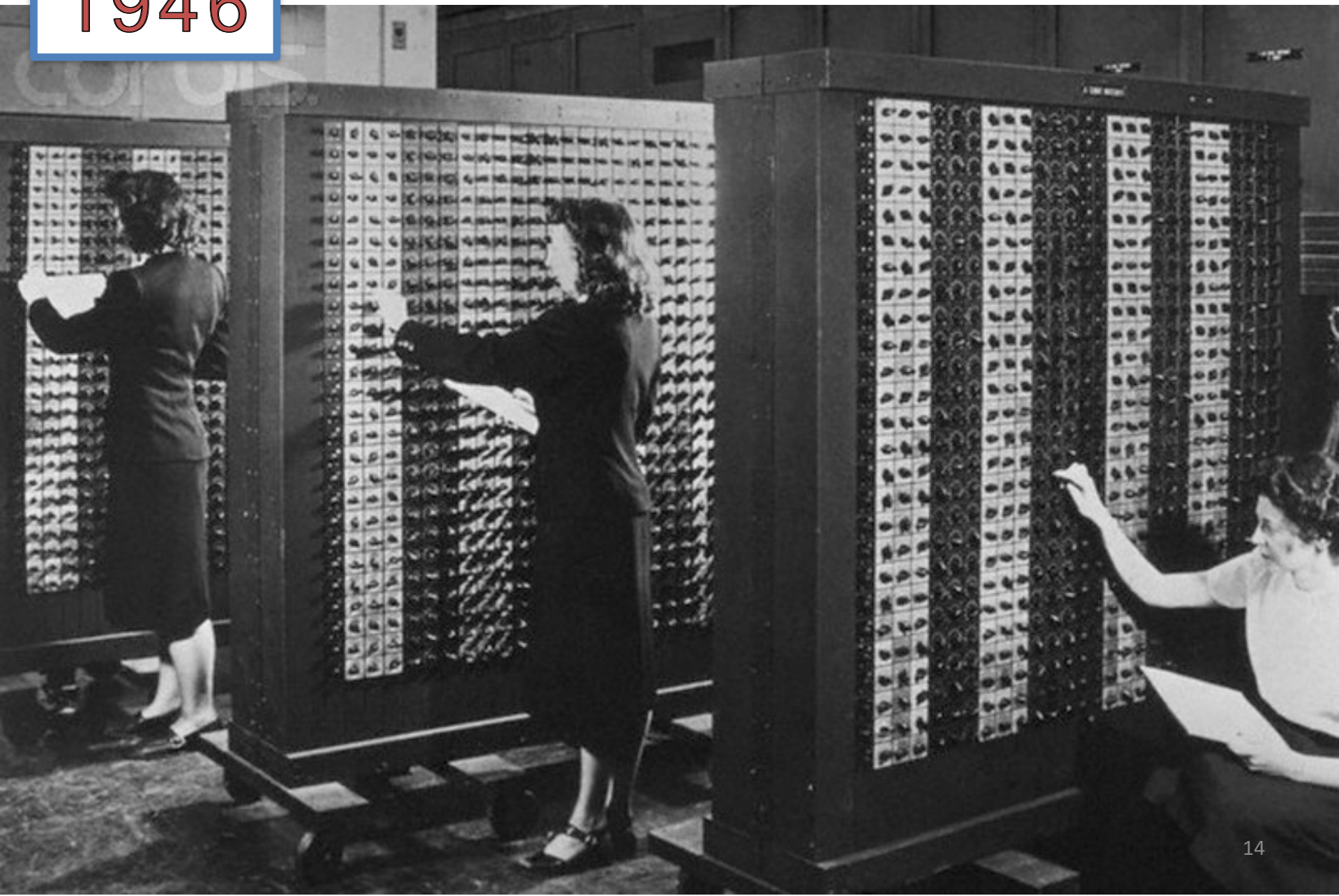
What is a Compiler?

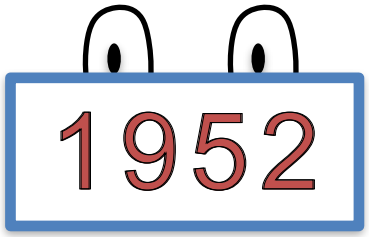


1946

ENIAC

Electronic **N**umerical **I**ntegrator **a**nd **C**omputer





“The Education of a Computer” (Grace Hopper)

UNIVAC **U**niversal **A**utomatic **C**omputer

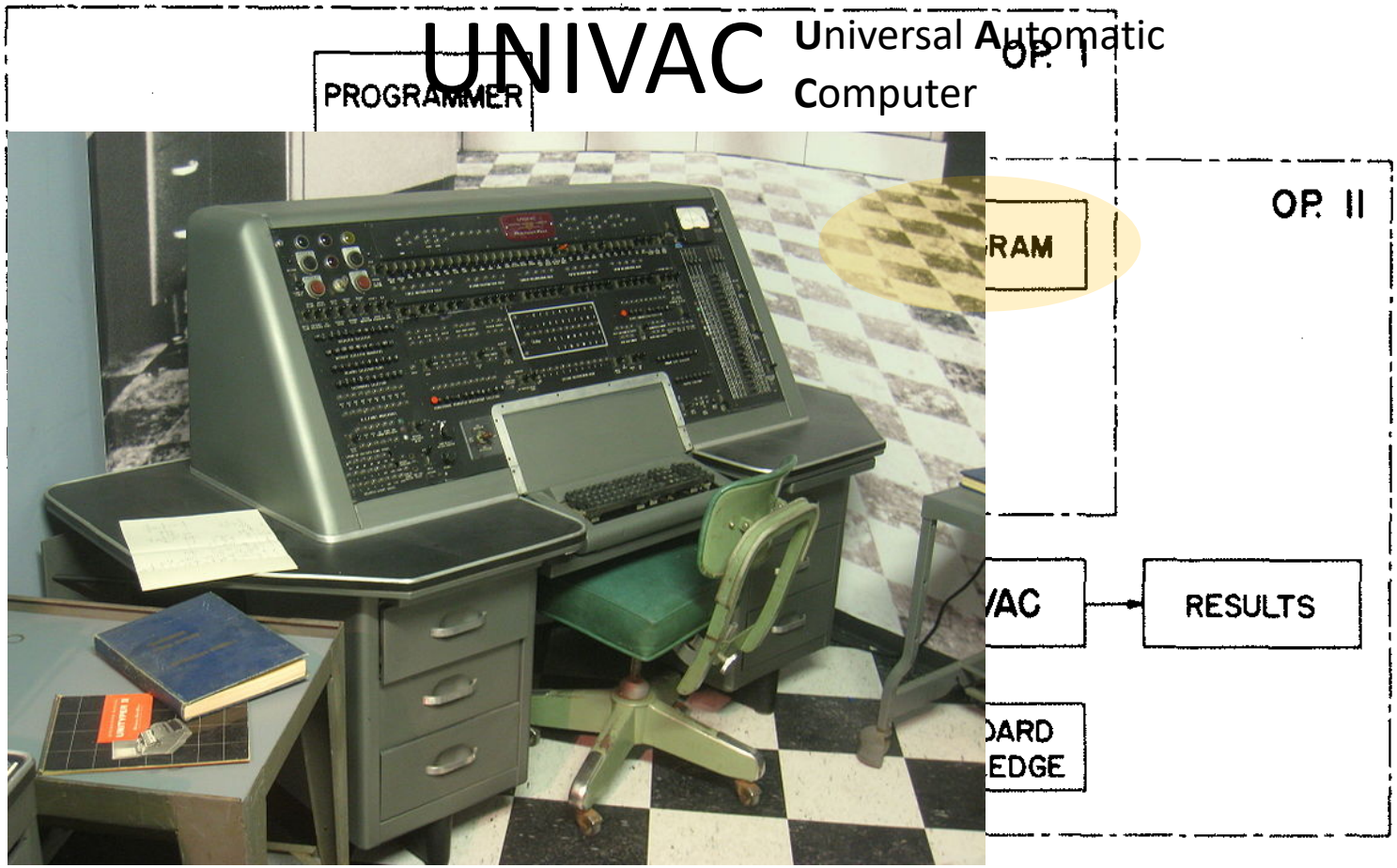
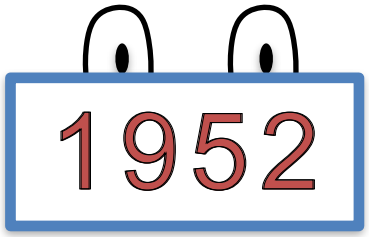


Fig. 4 - SOLUTION OF A PROBLEM



"The Education of a Computer" (Grace Hopper)

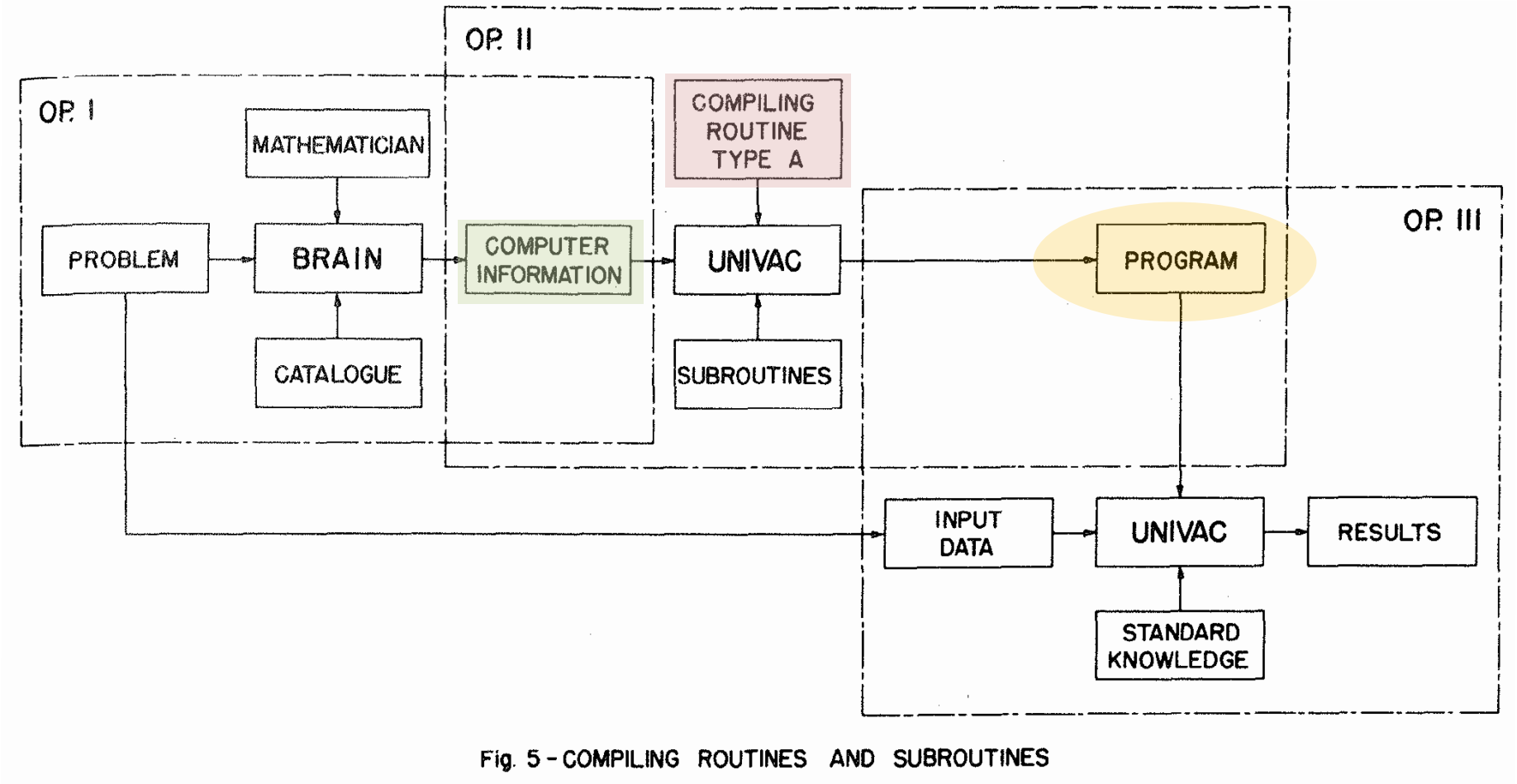
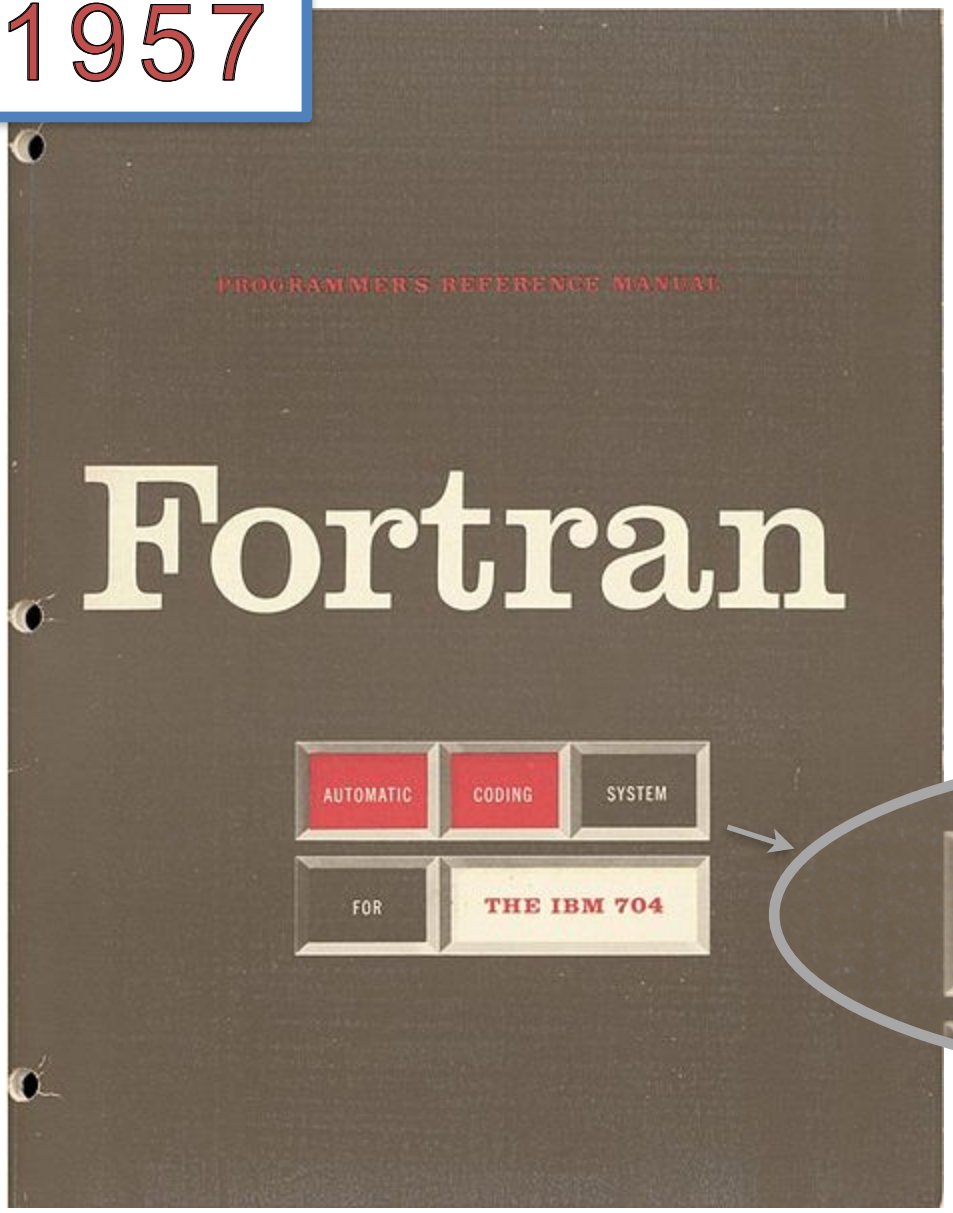


Fig. 5 - COMPILING ROUTINES AND SUBROUTINES

1957

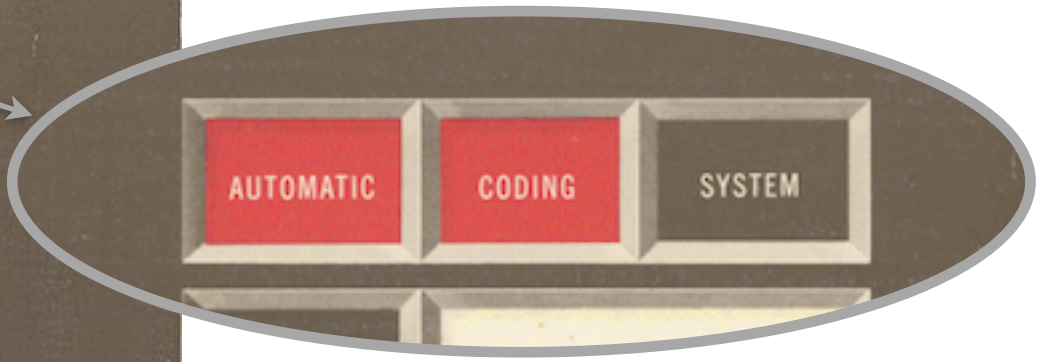


C ◄ FOR COMMENT	STATEMENT NUMBER 1–5	CONTINUATION 6	FORTRAN STATEMENT 7–72	IDENTIFICATION 73–80
C			PROGRAM FOR FINDING THE LARGEST VALUE	
C		X	ATTAINED BY A SET OF NUMBERS	
			DIMENSION A(999)	
			FREQUENCY 30(2,1,10), 5(100)	
			READ 1, N, (A(I), I = 1,N)	
	1		FORMAT (I3/(12F6.2))	
			BIGA = A(1)	
	5		DO 20 I = 2,N	
	30		IF (BIGA-A(I)) 10,20,20	
	10		BIGA = A(I)	
	20		CONTINUE	
			PRINT 2, N, BIGA	
	2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
			STOP 77777	

John Backus and team at IBM

(Fortran = Formula Translator)

The first complete compiler



A

ABAP4
ABC
ACPI Source Language
ACS
Action!
Actionscript (Flash 5)
ActionScript (Flash 8)
Actionscript (Flash MX)
ActionScript 3.0
Ada
ADVPL
Agda
aheui
Algol-60
Algol-68
Alpha-Five-Xbasic
Amazon States Language
amharic
Amiga-E
AMOS
Andl
AngelScript
Anguish
Ansible
Ante
APC
APL
AppleScript
Arduino
Arena
Arendelle
Argh!
ArnoldC
AsciiDots
ASP (C#)
ASP (JavaScript)
ASP (VBE)
ASP (VBS)
ASP.NET
Assembler (6502, Apple II)
Assembler (6502, C64)
Assembler (68000, Amiga)
Assembler (68000, Atari ST)
Assembler (68008)
Assembler (8051)
Assembler (ARM, Android)
Assembler (ARM, RISC OS)
Assembler (DG-Nova)
Assembler (HLA)
Assembler (HP-85)
Assembler (IBM 370)
Assembler (Intel)
Assembler (Itanium)
Assembler (MIPS)
Assembler (MMIX)
Assembler (PA-RISC)
Assembler (PDP-11)
Assembler (PDP-8)
Assembler (PPC, Darwin)
Assembler (SPARC)
Assembler (TAS)
Assembler (VP)
Assembler (Win32)
Assembler (X1)
Assembler (X8)
Assembler (z390)
Assembler (Z80 Console)
Assembler (Z80, CP/M)
Assembler (ZX81)
Asterisk
ATS
AutoHotkey
AutoIT3
Automator
AviSynth
AWK

B

B
BAL
BASIC-PICAXE
BASIC
Bato
Batsh
bc
BCPL
BeanShell
Beef
Befunge
Beta
BibTex
Binary Lambda Calculus
BIT
BitZ
Blender
BlitzMax
BlitzPlus
BMC Remedy
Boo
Brainfuck
BrightScript
BS2000
BSP
BuddyScript
Buzz
Byte Syze

C

C*
C/AL
C#
C (Amiga Anywhere)
C (ANSI)
C (Curses)
C (GEM)
C (Intuition)
C (K&R)
C (OpenGL)
C (PresentationManager)
C (Windows)
C (X11 Athena)
C++
C++ (.NET CLI)
C++ (Epoc)
C++ (FLTK)
C++ (Gtk++)
C++ (ISO)
C++ (MFC)
C++ (Qt)
CA-Easytrieve Plus
Caché Object Script
CAML-Light
Carp
Casio BASIC
CDuce
Centura
Ceylon
ChaiScript
Chef
Chicken
CICS-COBOL
Clarion
Clean
Clipper
Clojure
CLP
CMake
COBOL
Cobra
Cocoa
Coconut
CoDScript
CoffeeScript
ColdFusion
CommandScript
Common Lisp
Console Postscript
ConTeXt
Cool
CoolBasic
cpl
Crystal
CSS
CUDA
CWEB
CYBOL

D

D
D4
Dafny
Darkbasic
Dart
Databasic
Dataflex
dBase
dc
DCL
Delphi
Di
DI
DI
Dr
DV
DV
DV

E

E
eC
E
E
E
E
ELENA 3.0
ELENA 4.0
Elixir
Elliott
Elm
Emojicode
Emoticon
EOS 2
Erlang
ERRE
Eta
Euphoria
Eve
Ezhil

F

F#
F
Falcon
FALSE
Felix
Fennel
Ferite
Fetlang
Filemaker
Fjölnir
Flaming Thunder
Focal
FOCUS
Folders
Forth
Fortran
Fortran77
Fortran90
FortranIV
Fortress
FreeBASIC
Frink
Full Metal Jacket

G

G-code
Gambas
GameMonkey Script
Genero BDL
Genie
Gentee-simple
Gentee
GLBasic
GML
Go
Gofer
GoogleGadgets

ophone
V

HTML
Hubot
Human
HyperTalk

I

IBM-Exec
IBM-Exec2
ici
ICL SCL
Icon
IDC
IDL
Idris
Imba
Inform
Informix 4GL
Ingres ABF
InstallScript
Intercal
Io
Iptscrae

J

J
Jade
Jako
Janet
Jason
Java
Java (Mobile)
Java (Servlet)
Java (Swing)
Java Server Pages
JavaScript
JCL
Jess
Jorf
JSFuck
JudoScript
Julia

K

K
K3
K4
K5
Kitten
Kix
Kotlin
Kumir
Kylix

L

LÖVE
L

C

M

m4
MACRO-10
MACRO-11
Macromedia-Flex
MAD
make
Malbolge
MAMASH
Maple
Marmelade
Mathematica
MATLAB
Matrix
MAX-MSP
Maxima
MAXScript
MCSBL
MDM Zinc
MEL
MetaPost
MEX
Microtik
MiniScript
mIRC-Alias
mIRC-Commandline
mIRC-Script
MivaScript
MML-AXE10
Modula-2
Modula-3
MoHAA-Script
MOO
Mouse
MPD
MPLAB IDE
MS Small Basic
MSDOS
MSIL
MuLisp
Mumps
MySQL FUNCTION
Mythryl

N

Natural
Nemerle
newLISP
NewtonScript
Nice
Nim
Nit
Node.js
Noor
NSIS
NXC

O

Oberon.oberon
Oberon.std
Objective-C
OCaml
Occam
Octave
Omnimark
Ook
OpenVMS
OPL.dialog
OPL.simple
Oz

P

ParaSail
Parser
Pascal (Windows)
Pascal
Pawn
PBASIC
PDF
PEARL
PeopleCode
Perl
Perl 6
Pharo
PHP+GD
PHP
Picat
Piet
Pike
PILOT
PL-SQL
PL1
Plankalkül
Pocket Calculator
POP-11
Portugol
PostgreSQL
Postscript
POV-Ray
Powerbasic
Powerflex
PowerScript
Powershell
PPL
PQN-PROC
PRAAT
ProC
Processing
Profan
Prograph
Progress
Prolog
Punyforth
PureBasic (Console)
PureBasic (Messagebox)
PureBasic (Window)
Python 2
Python 3

Q

Q
Qalb
QCL
qore
QuakeC
Quartz Composer
QuickBASIC
Quorum

R

R
Racket
Rapira
ratfor
Rational Rose
React-VR
REALbasic
RealText
Reason
Rebol-view
REBOL
Redcode
REFAL-2
Refal
Regular Expression
Rexx (simple)
Rexx (window)
Rey
rhine
RPG IV v3-4
RPG IV v5
RPG IV v7.1
RPL
RSL
Ruby
Rust

S

S-Plus
SAKO
SAL
SApp
SAS
Sather
Sawzall
Scala
Scheme
Scilab
Scratch
Seed7
Self
SenseTalk
Setl2
Shakespeare
Shen
SilverBasic
SIMPLE
Simula
SinclairBasic
Skip
Smalltalk (simple)
Smalltalk (window)
Smalltalk MT
SMIL
SML
Snobol
Spin
Spiral
SPL4
Splunk SPL
Spry
SPSS
SQL (Advantage)
SQL (DB2)
SQL (Oracle)
SQL
sqlplus
SQR
Squeak
Squirrel
SSI
ST-Guide
Stata
Sucro
SuperCollider
SVG
Swift
SyMAL
Symsyn
szl

T

T-SQL
T
T9
TACL
TAL
Tcl
TECO
TeX
Texinfo
Thue
TI BASIC
TI Extended BASIC
TI-59
TI-8x
TinyFugue
Tk
Toy
Trans
troff
TSO CLIST
Turing Machine
Turing
TypeScript

U

Ubercode
UniComal
Unix Shell
unlambda
UnrealScript
Ursala

V

Vala
var'aq
Vatical
VAX Macro
VAX-11 Macro
VBA (Excel)
VBA (Word)
VBScript
Velato
Velocity
Verilog
Vexi
VHDL
Vim script
Visual Basic
Visual Basic .NET
Visual Basic 6
Visual FoxPro
Visual Prolog
VisualWorks Smalltalk
VMS-DCL
VRML
VSL
VVVV

W

wenyan
Whenever
Whirl
Whitespace
wml
Wolfram
WSH

X

X++
Xbase++
xblite
XHTML
XLogo
XPL0
XQuery
XSL-FO
XSLT

Y

Yacas

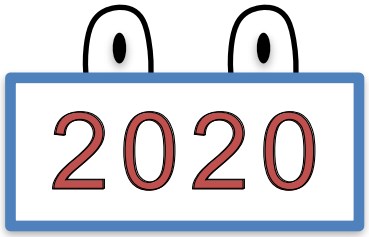
Z

Zig
ZIM

source: http://helloworldcollection.de

High Level Programming Languages

- Imperative Algol, PL1, Fortran, Pascal, Ada, Modula, C
  - Closely related to “von Neumann” Computers
- Object-oriented Simula, Smalltalk, Modula3, C++, Java, C#, Python
  - Data abstraction and ‘evolutionary’ form of program development
    - Class as implementation of an abstract data type (data+code)
    - Objects instances of a class
    - Inheritance + generics
- Functional Lisp, Scheme, ML, Miranda, Hope, Haskel, OCaml, F#
  - First class functions
- Logic Programming Prolog



What is a Compiler?

"A compiler is **computer software** that **translates** computer code written in one programming language (the **source language**) into another language (the **target language**). …primarily to a lower level language (e.g. assembly language, object code, or machine code) to create an **executable program**."

-- *Wikipedia*

What is a Compiler?

source language

C
C++
Pascal
Java

txt

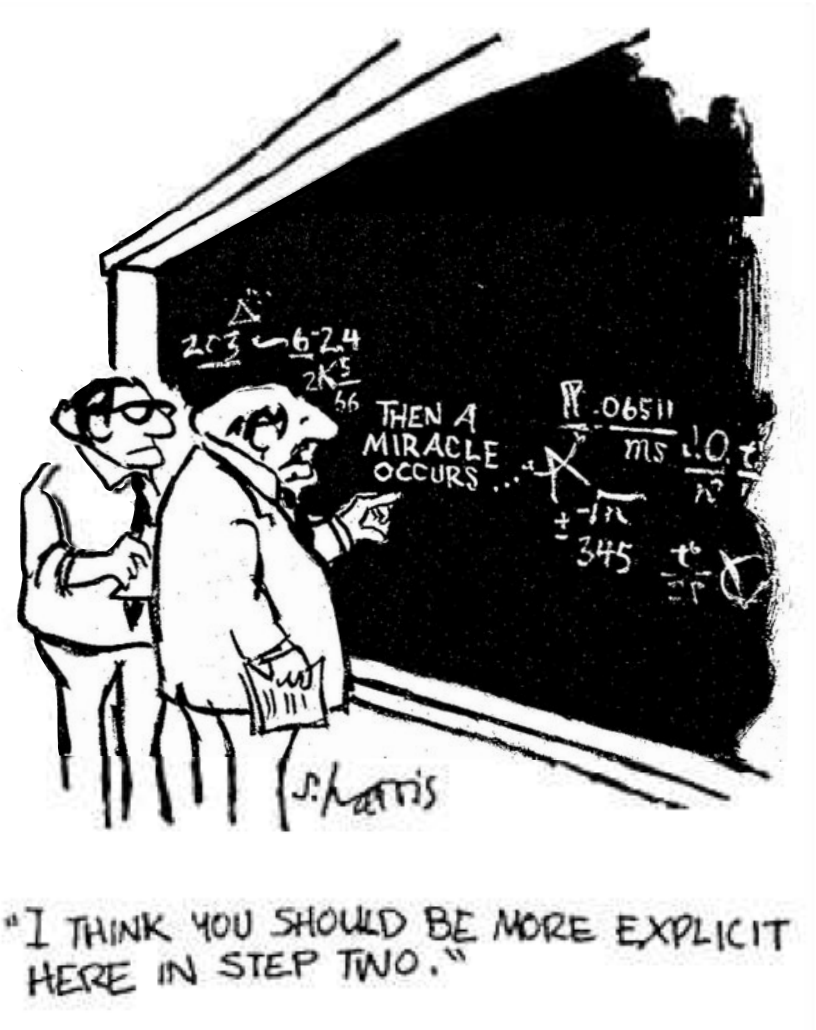
Perl
JavaScript
Python
Ruby

Source text

Lisp
Scheme
ML
OCaml

Prolog

Postscript
TeX



target language

IA32
IA64
ARM
SPARC

exe

Java Bytecode

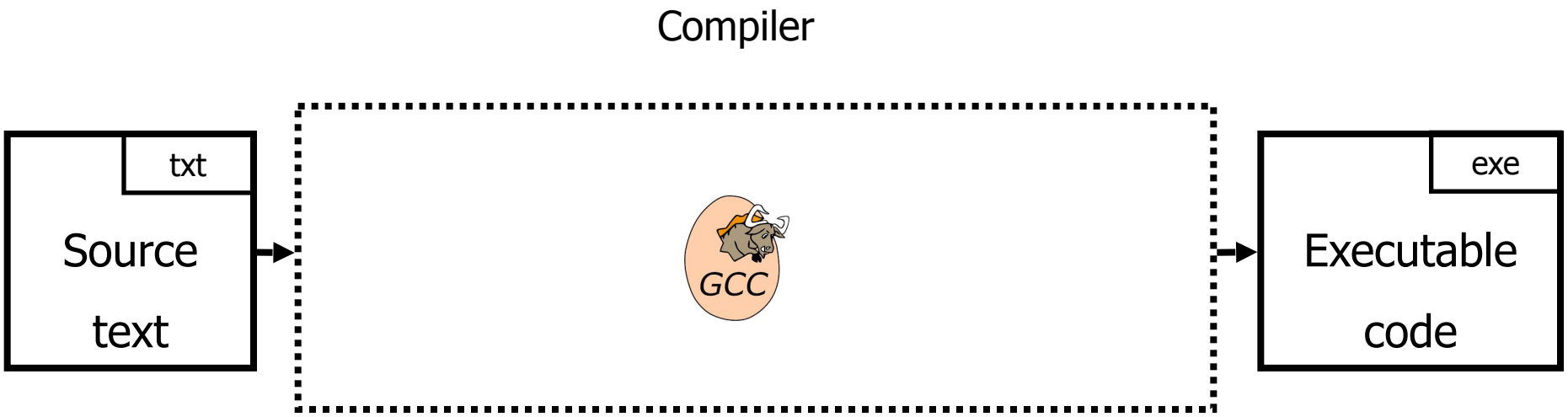
Executable code

C
C++
Pascal
Java

JavaScript

PDF
Bit Map
…

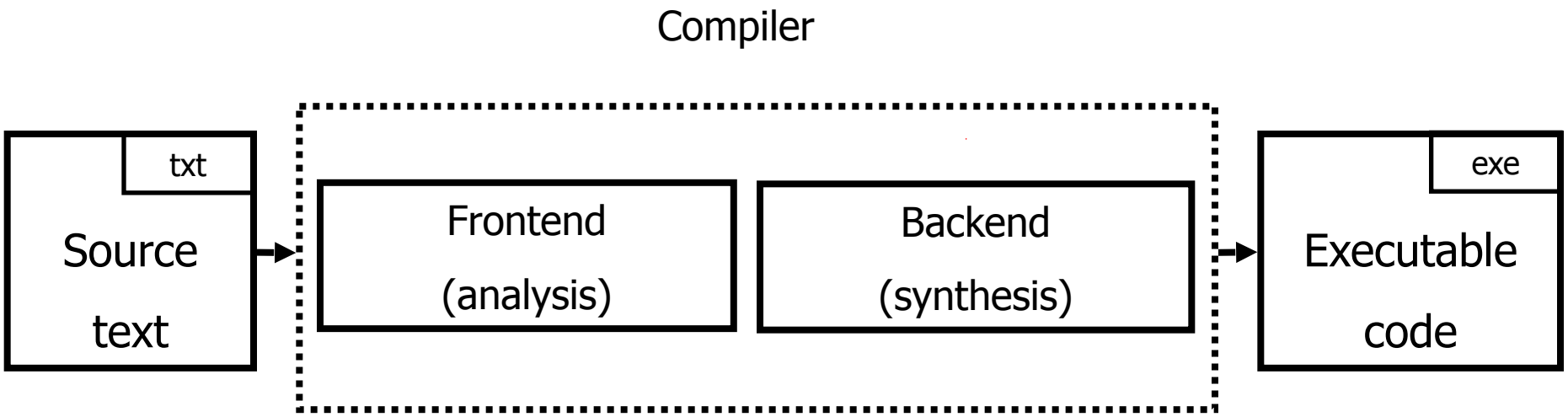
What is a Compiler?




```
int a, b;
a = 2;
b = a*2 + 1;
```

```
MOV R1,2
SAL R1
INC R1
MOV R2,R1
```

Anatomy of a (2-Stage) Compiler




```
int a, b;
a = 2;
b = a*2 + 1;
```

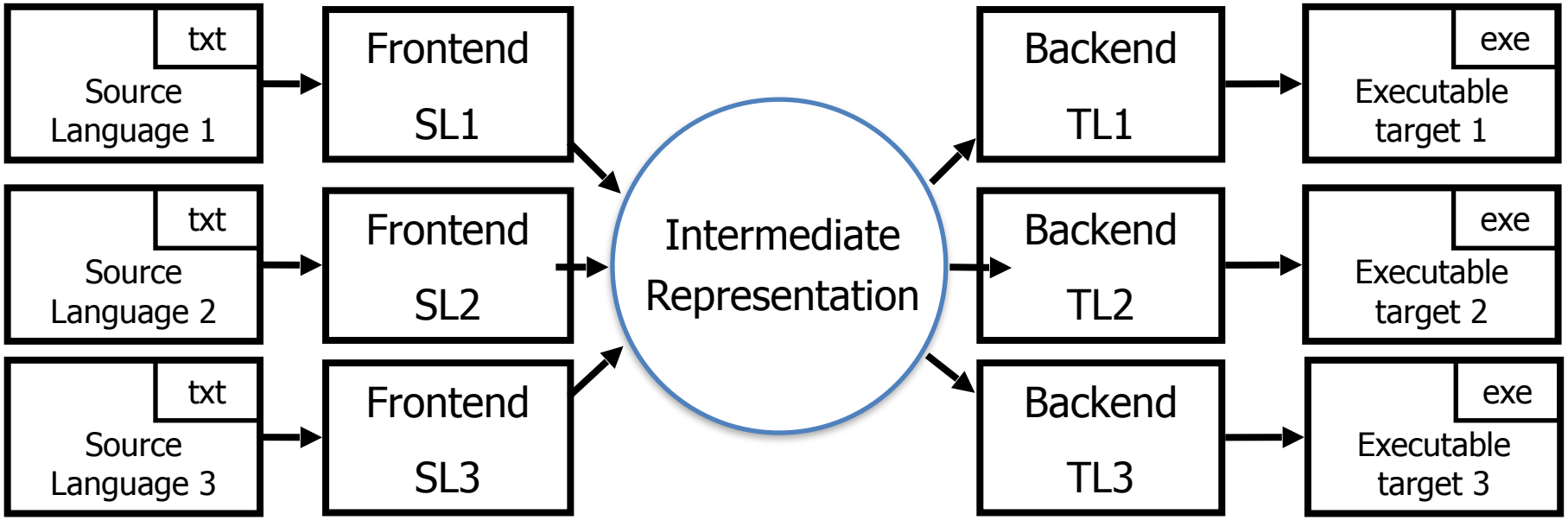
```
MOV R1,2
SAL R1
INC R1
MOV R2,R1
```

Why?

Modularity

```
var
  x, y : integer;
begin
  x := 2;
  y := x*2 + 1;
end.
```

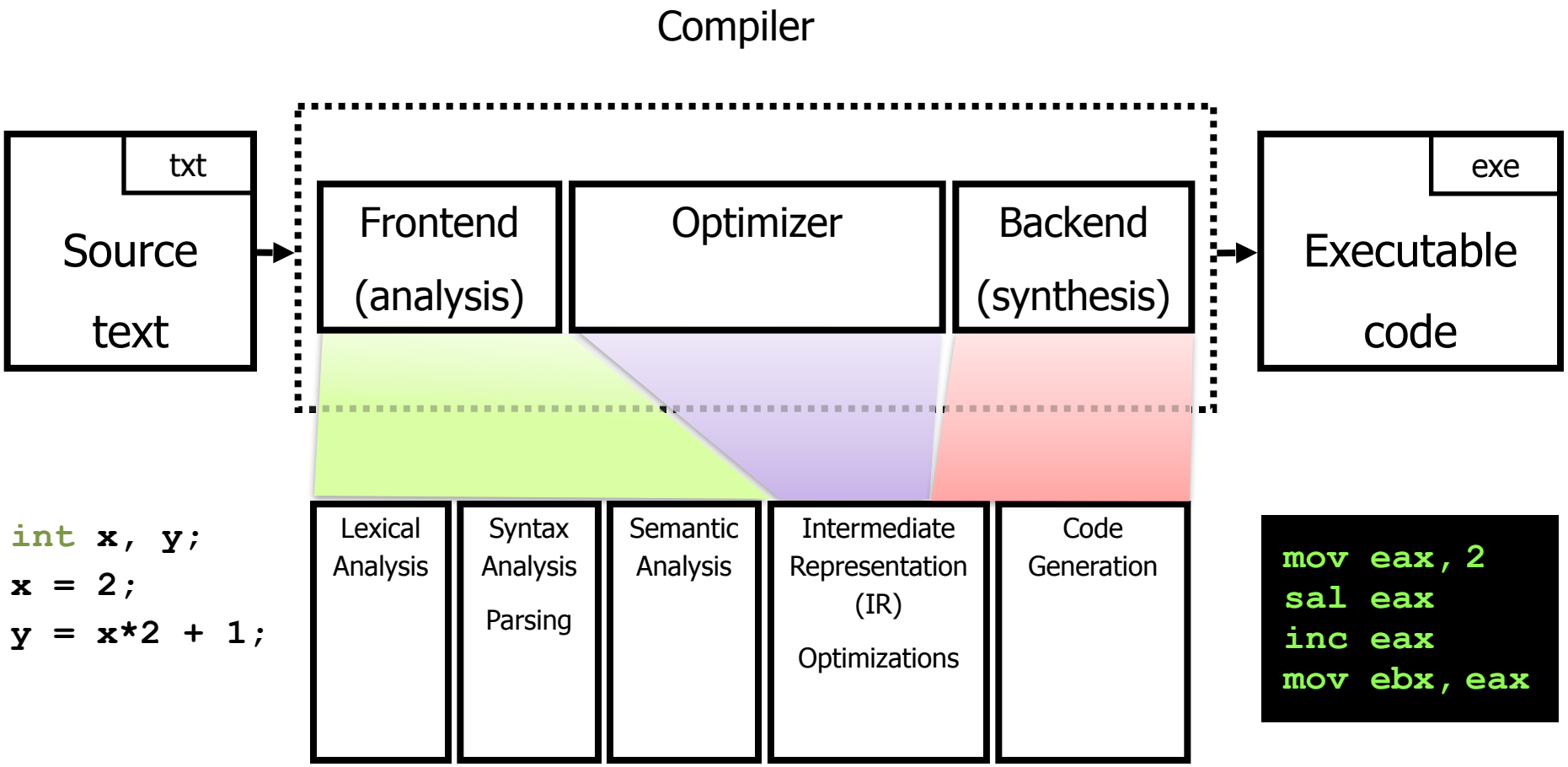
```
iconst 2
iconst 1
ishl
istore_2
iinc 2, 1
```




```
int x, y;
x = 2;
y = x*2 + 1;
```

```
mov eax, 2
sal eax
inc eax
mov ebx, eax
```

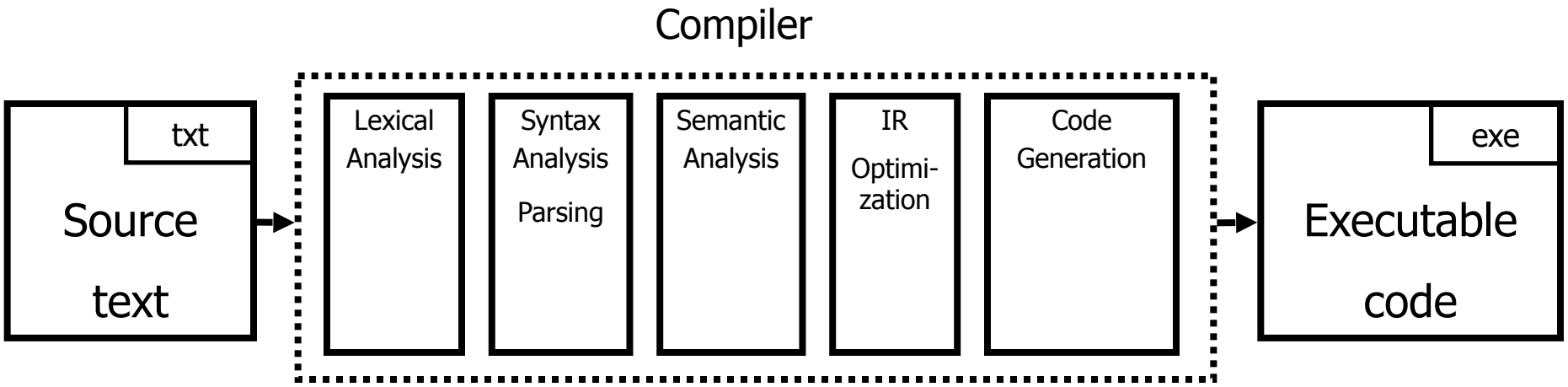
Anatomy of a (3-Stage) Compiler



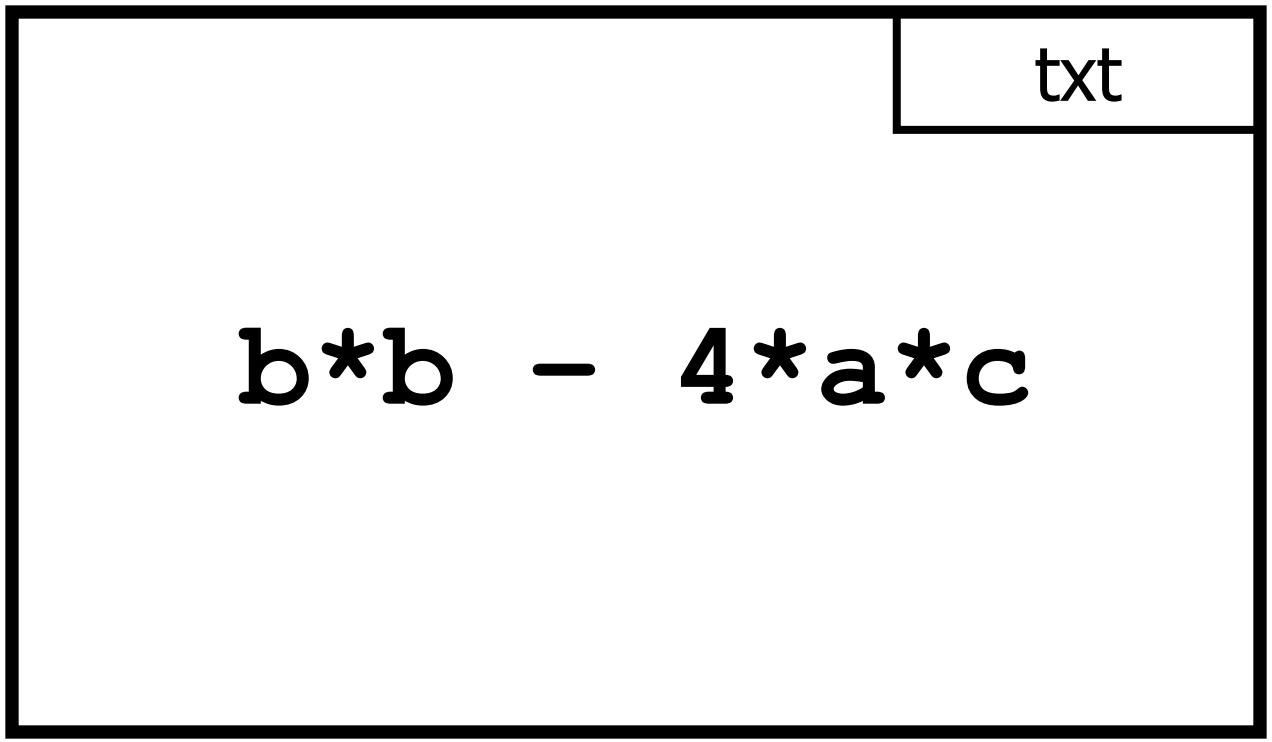

```
int x, y;
x = 2;
y = x*2 + 1;
```

```
mov eax, 2
sal eax
inc eax
mov ebx, eax
```

Journey inside a compiler

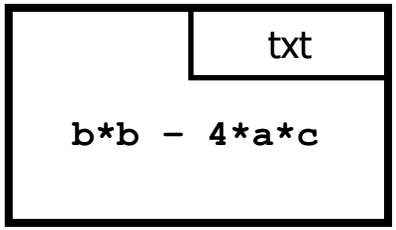


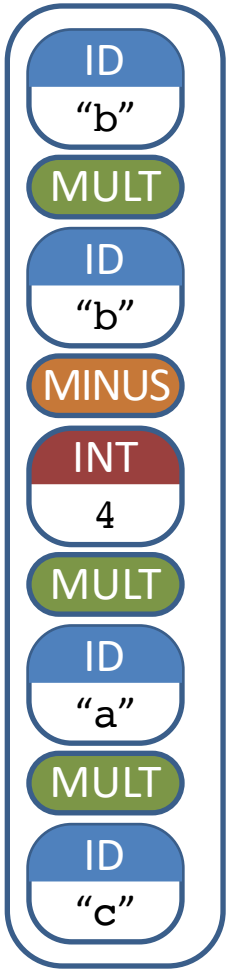
Journey inside a compiler



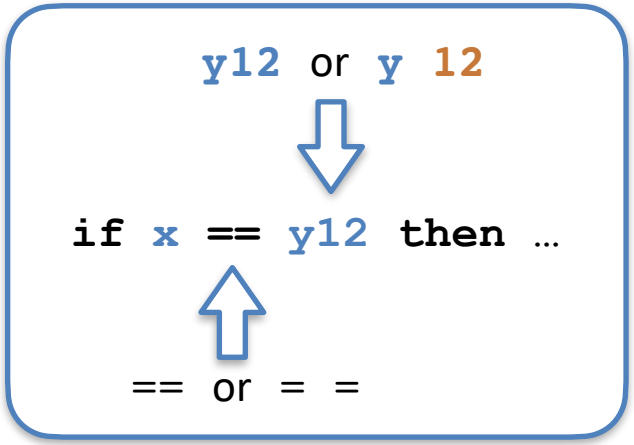
Journey inside a compiler

Recognizing words



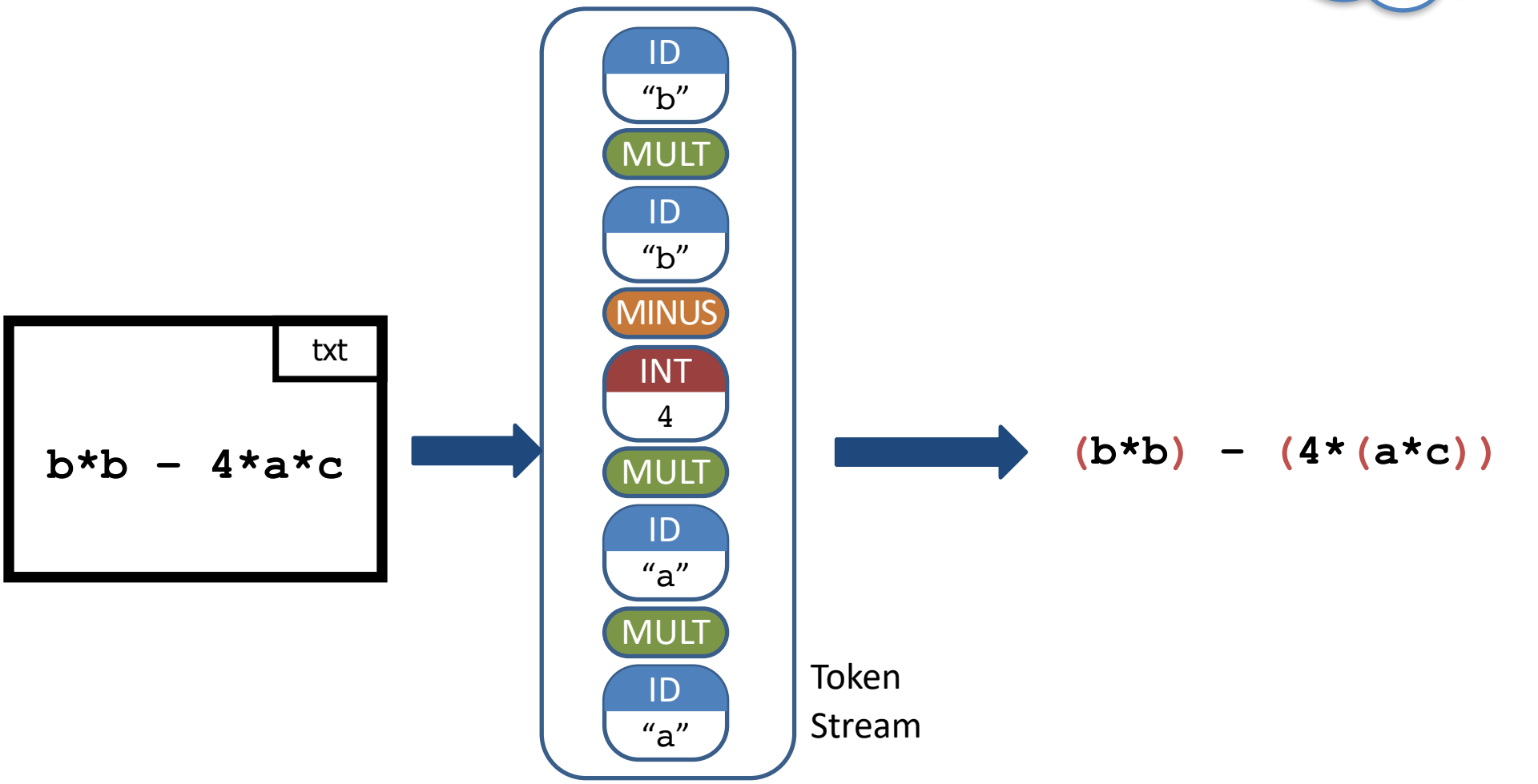


Token Stream



Journey inside a compiler

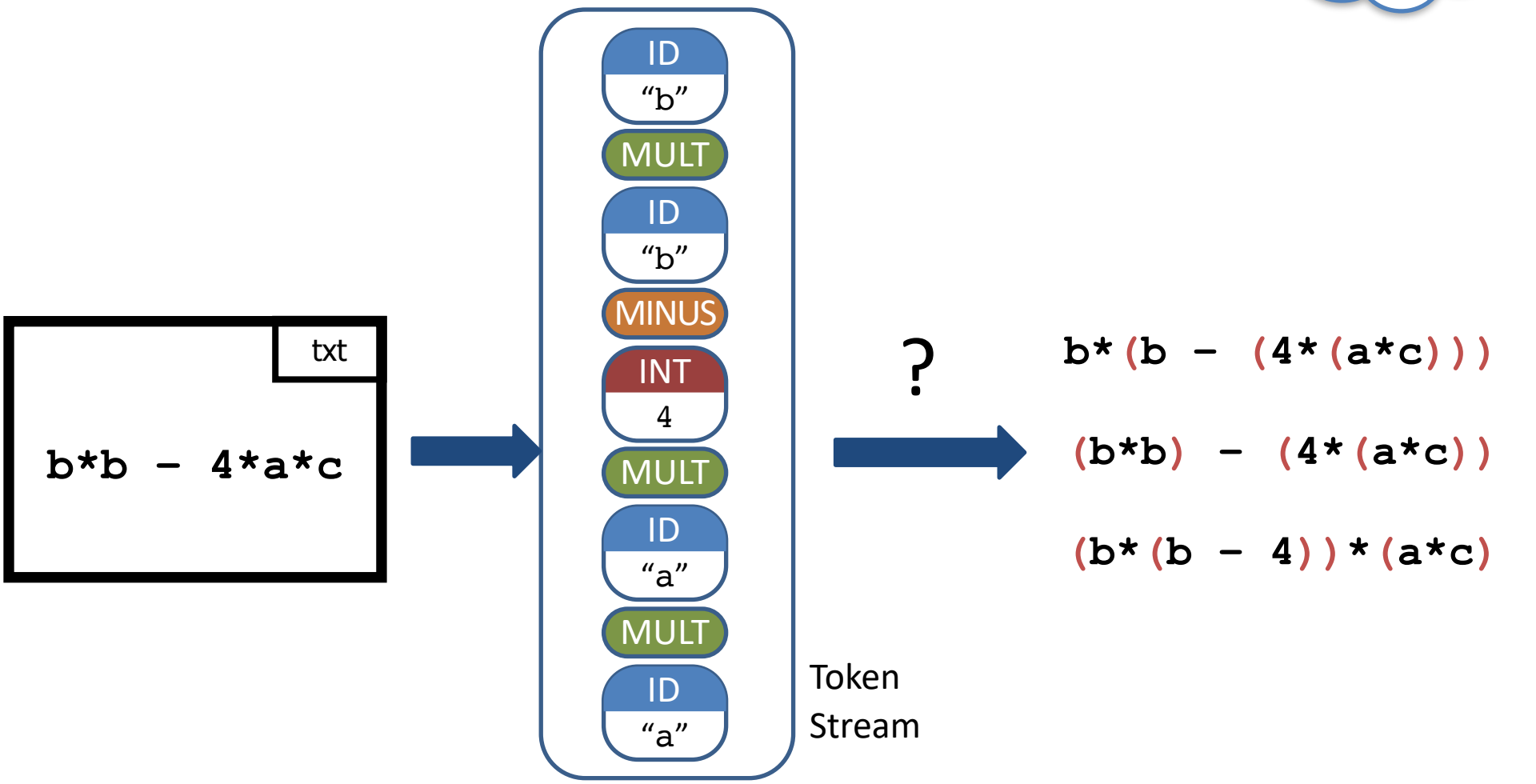
Understanding Sentence Structure



Lexical Analysis | Syntax Analysis | Semantic Analysis | IR Opt. | Code Gen.

Journey inside a compiler

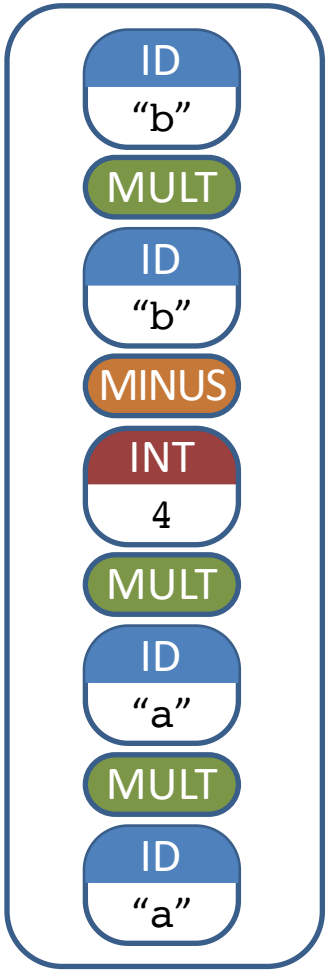
Understanding Sentence Structure



Lexical Analysis | Syntax Analysis | Semantic Analysis | IR Opt. | Code Gen.

Journey inside a compiler

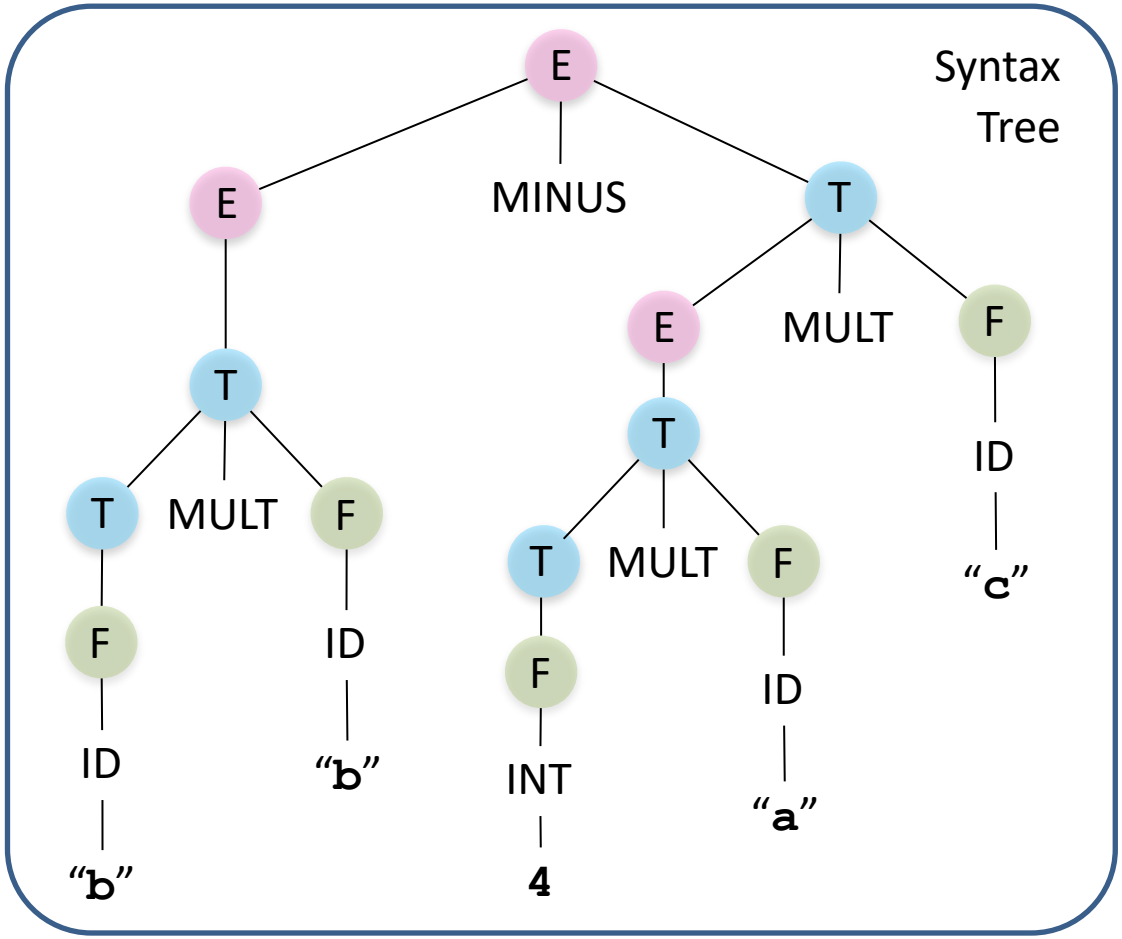
Understanding Sentence Structure



Token Stream

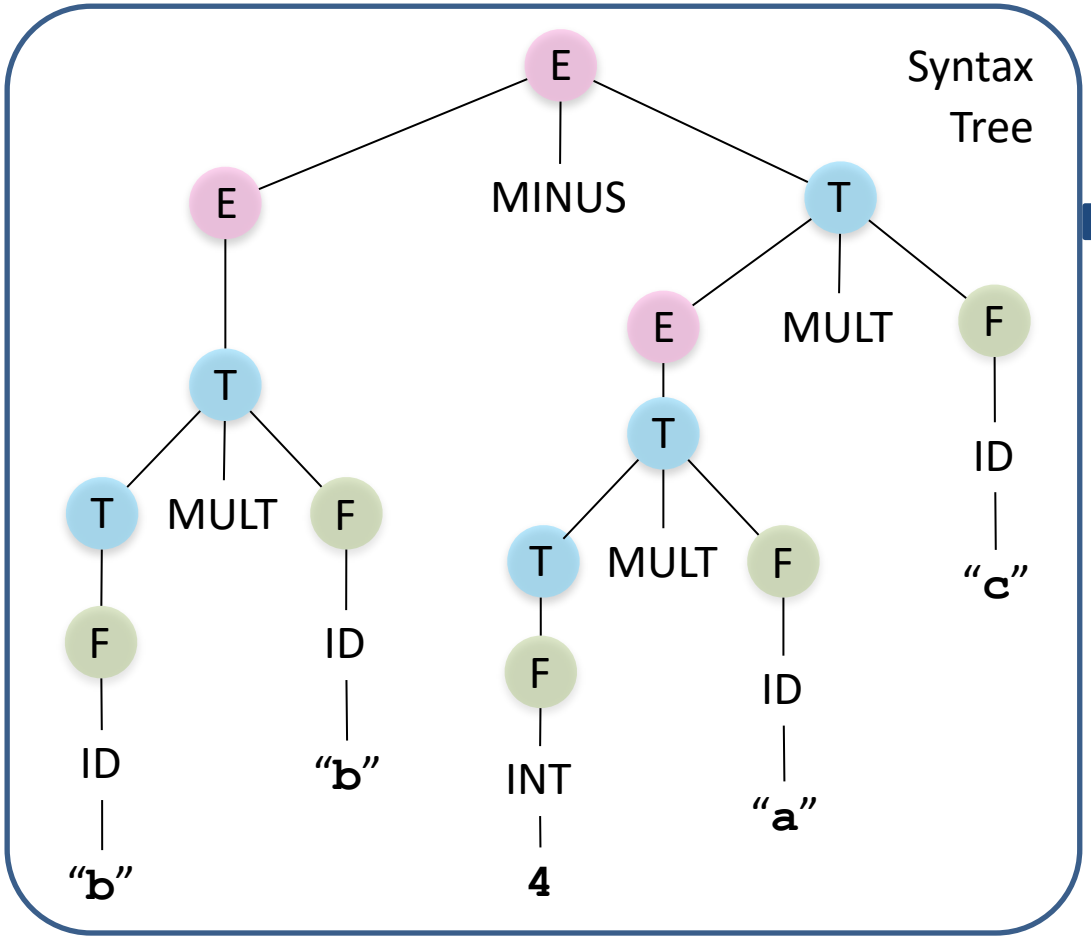
Grammar

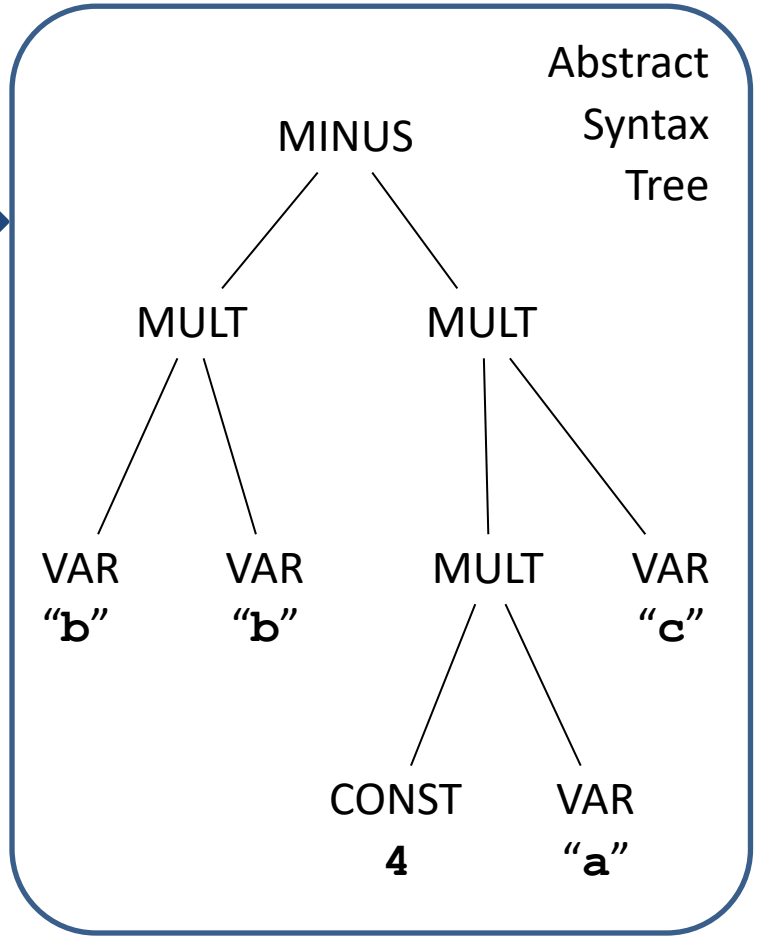
E → E PLUS T
E → E MINUS T
T → T MULT F
T → T DIV F
F → ID
F → INT



Journey inside a compiler

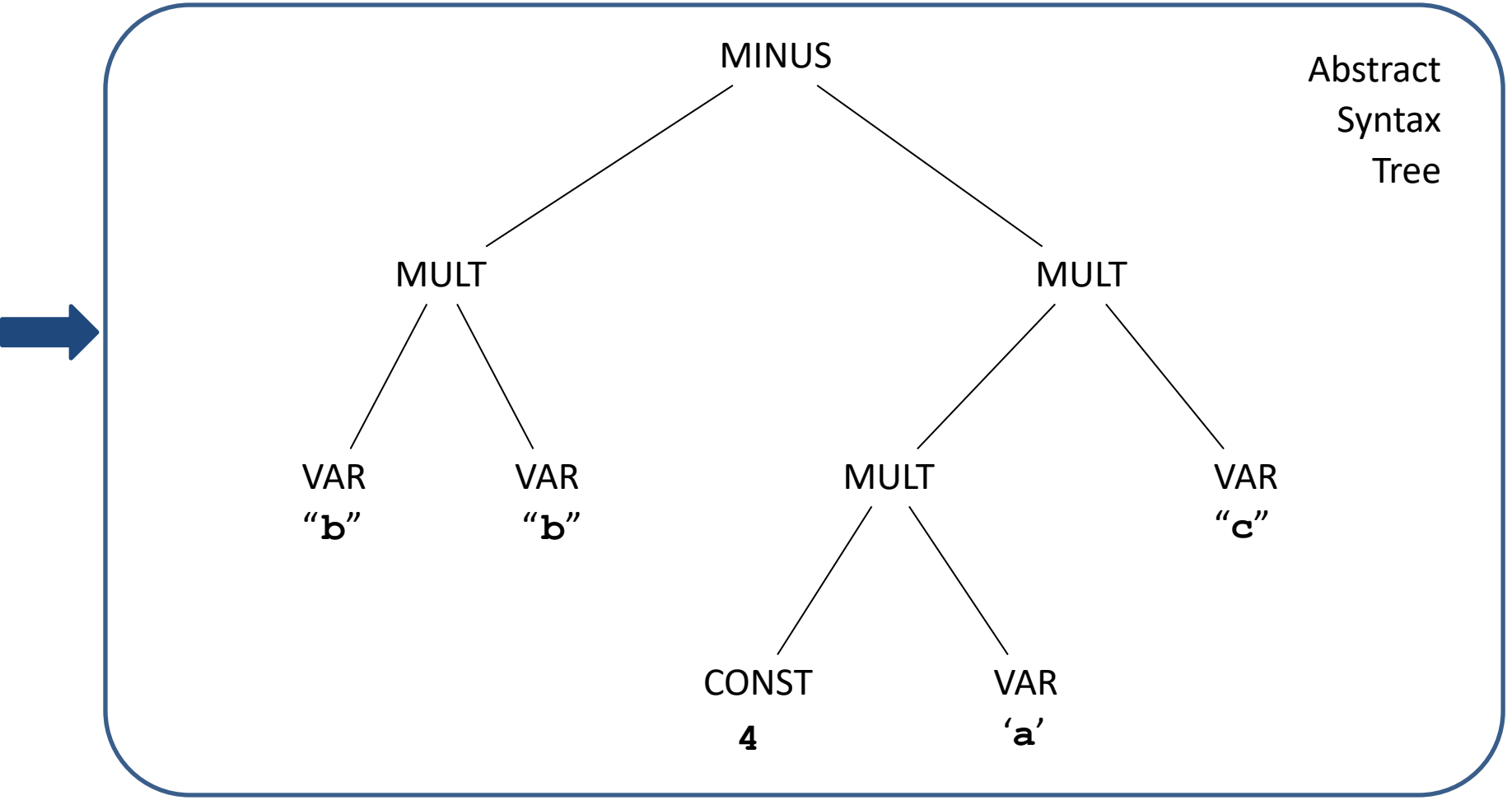
Extracting essence





Journey inside a compiler

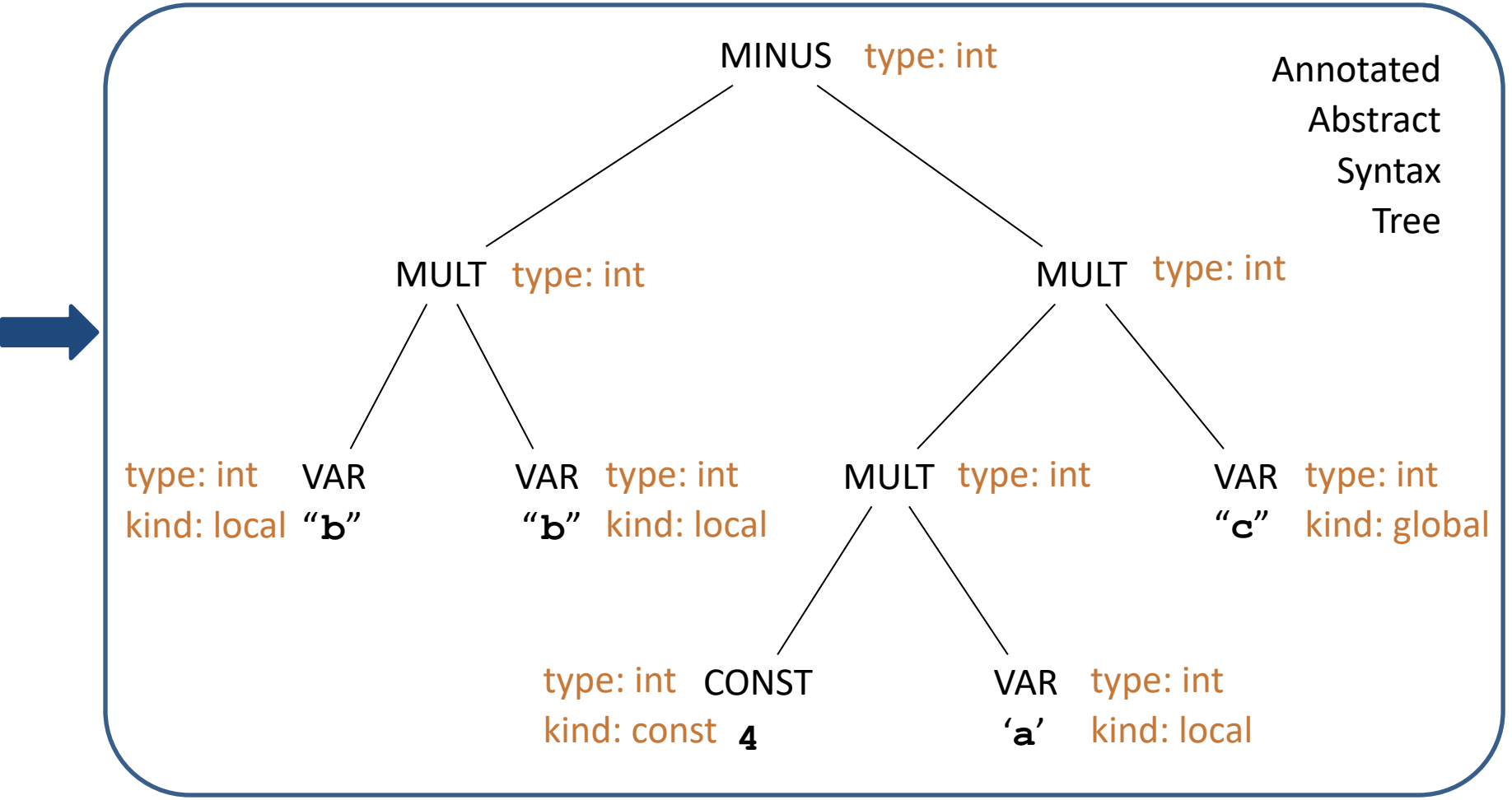
Understanding "meaning"



Lexical Analysis | Syntax Analysis | Semantic Analysis | IR Opt. | Code Gen.

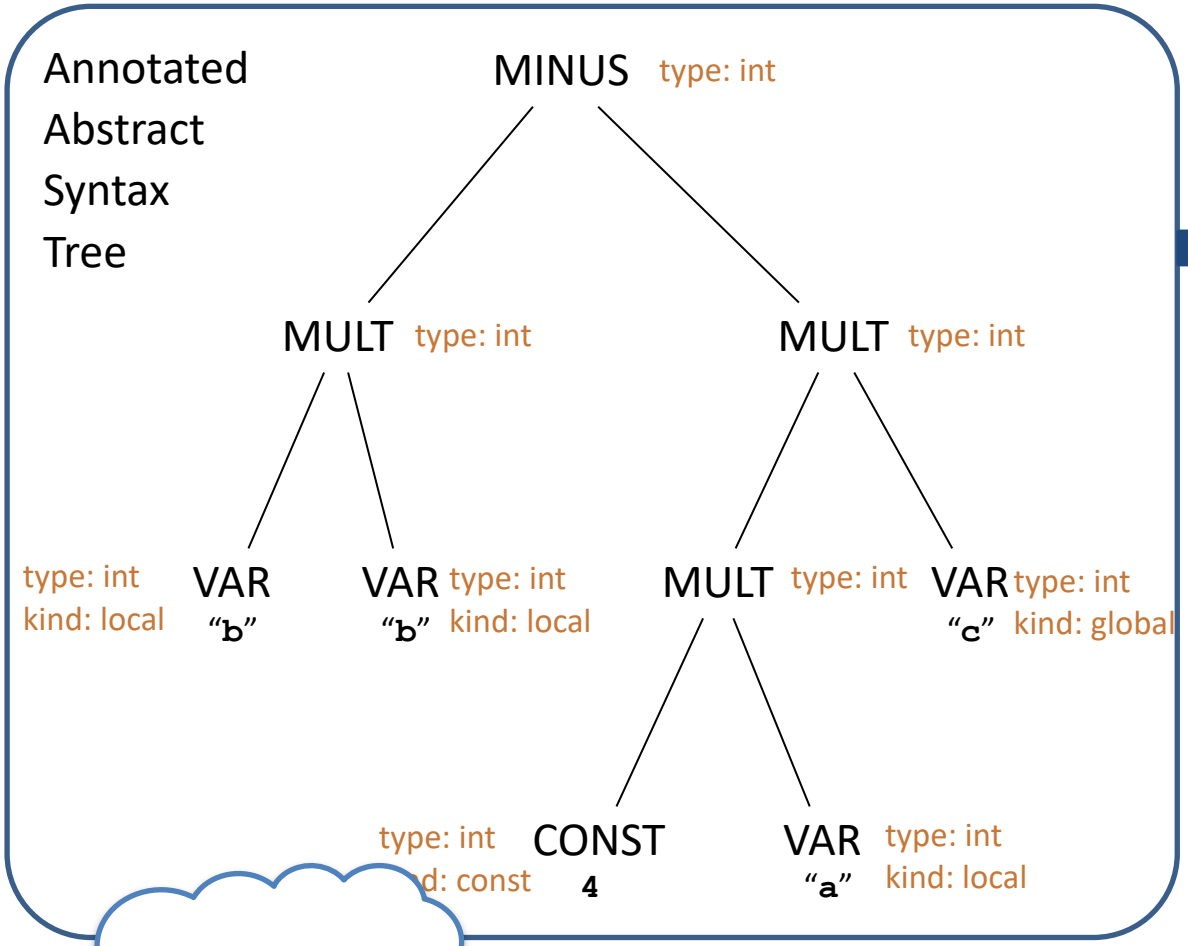
Journey inside a compiler

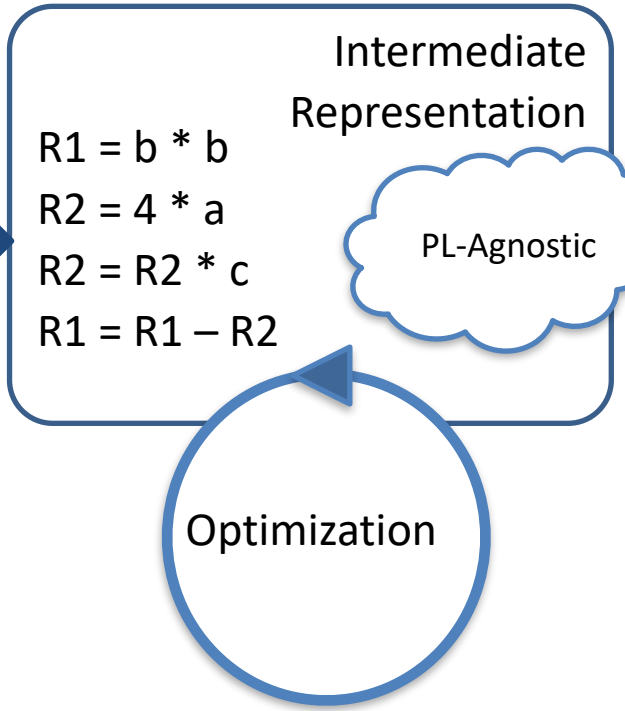
Understanding "meaning"



Lexical Analysis | Syntax Analysis | Semantic Analysis | IR Opt. | Code Gen.

Journey inside a compiler





Can be thought as a PL-specific IR

IR Optimizations

- Many optimization heuristics
  - Constant propagation
    - Leverage compile-time information to save work at runtime (pre-computation)

```
width = 10;
length = 20;
z = x * width * length;
```

```
width = 10;
length = 20;
z = x * 200;
```

IR Optimizations

- Many optimization heuristics
  - Constant propagation
    - Leverage compile-time information to save work at runtime (pre-computation)
  - Dead code elimination

```
x = y;
z = t;
x = y + z;
```


```
z = t;
x = y + z;
```

IR Optimizations

- Many optimization heuristics
  - Constant propagation
    - Leverage compile-time information to save work at runtime (pre-computation)
  - Dead code elimination
  - Loop optimizations: **hoisting**, unrolling, …

```
for (int i = 0; i < 100; ++i) {
      array[i] = x + y;
}
```


```
int t = x + y;
for (int i = 0; i < 100; ++i) {
      array[i] = t;
}
```

IR Optimizations

- Many optimization heuristics
  - Constant propagation
    - Leverage compile-time information to save work at runtime (pre-computation)
  - Dead code elimination
  - Loop optimizations: hoisting, **unrolling**, …

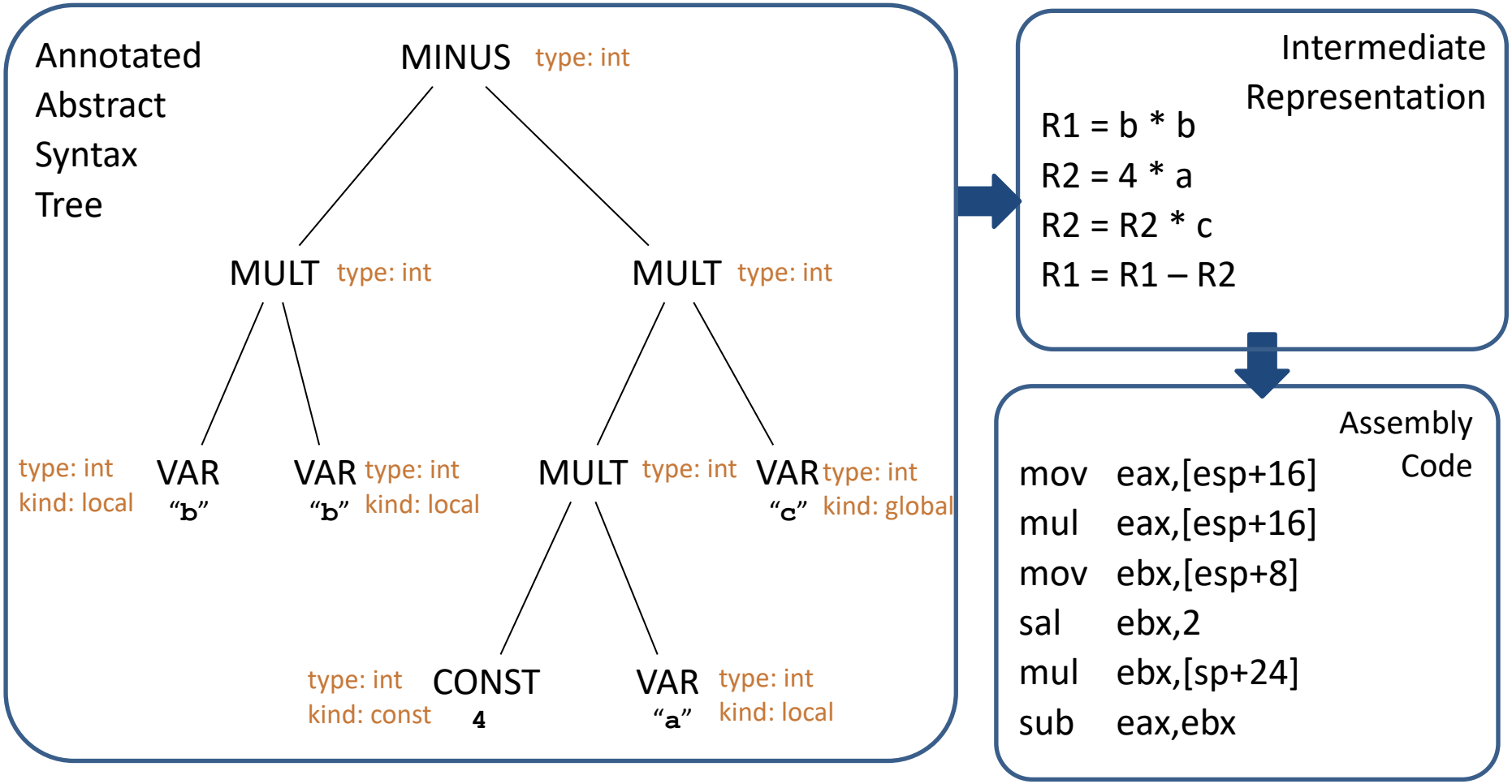
```
for (int i = 0; i < 80; ++i) {
      delete array[i];
}
```


```
for (int i = 0; i < 80; i += 4) {
      delete array[i];
      delete array[i+1];
      delete array[i+2];
      delete array[i+3];
}
```

IR Optimizations

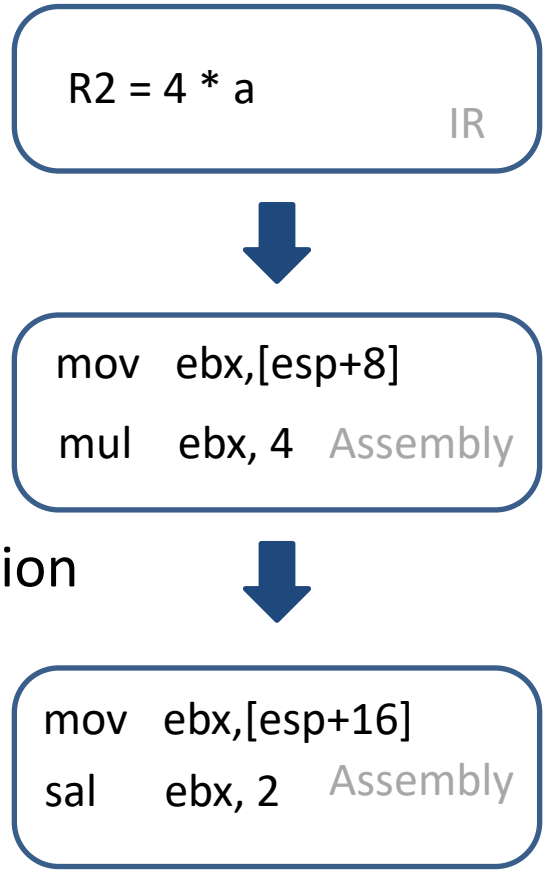
- Many optimization heuristics
  - Constant propagation
    - Leverage compile-time information to save work at runtime (pre-computation)
  - Dead code elimination
  - Loop optimizations: hoisting, unrolling, …
- "Optimal code" is out of reach
  - Undecidable/NP-complete problems
  - Use approximation and/or heuristics
  - Must preserve correctness, should (mainly) improve code
- The majority of compilation time is spent in optimization

Journey inside a compiler



Machine code generation

- Assigning variables to memory locations
- Register allocation
  - Optimal register assignment is NP-Complete
  - In practice, known heuristics perform well
- Instruction selection
  - Convert IR to actual machine instructions
  - Machine code optimizations: peephole optimization



- Modern architecture challenges
  - Multicores
  - Memory hierarchies
  - SIMD instructions

(not covered)

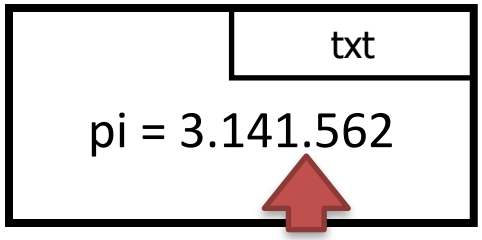
Error Checking

Check on every stage.
Report as soon as possible.

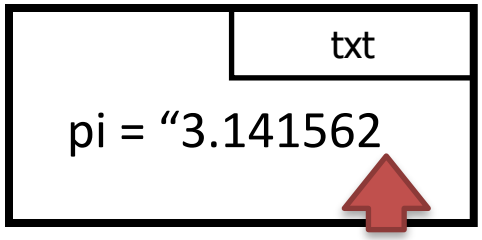
- Lexical analysis: illegal tokens
- Syntax analysis: illegal syntax
- Semantic analysis: incompatible types, undefined variables, …
- Even at runtime: division by zero, array bounds,…
  - (Compiler generates the error checking vode)

- Every phase tries to recover and proceed with compilation

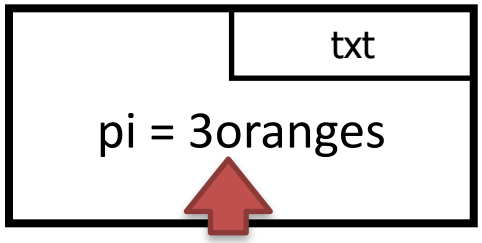
Lexical Errors



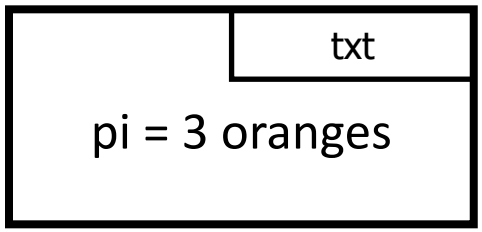
Illegal token



Illegal token (*unterminated string*)

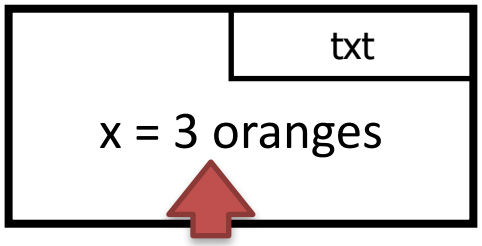


Illegal token (*malformed identifier*)

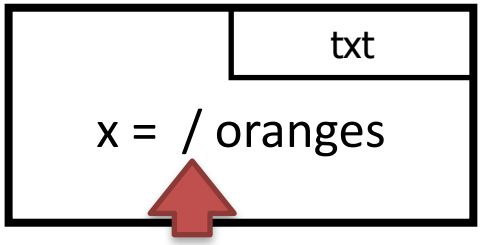


⟨ID,“`pi`”⟩, ⟨EQ⟩, ⟨INT, 3⟩, ⟨ID,“`oranges`”⟩

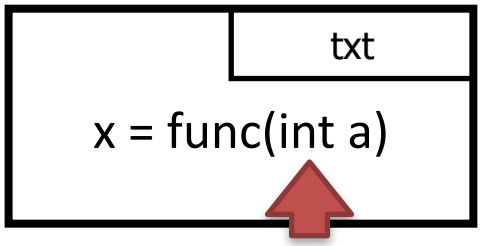
Syntax Errors



Missing operator

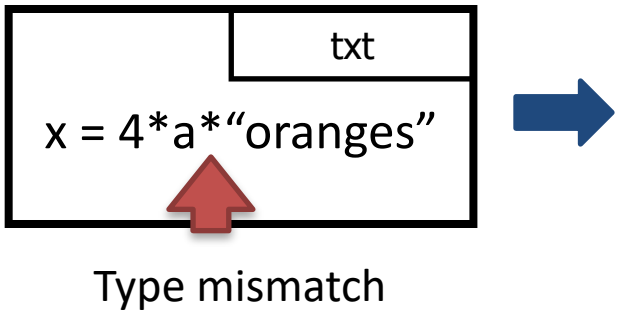


Wrong number of arguments to operator “/”

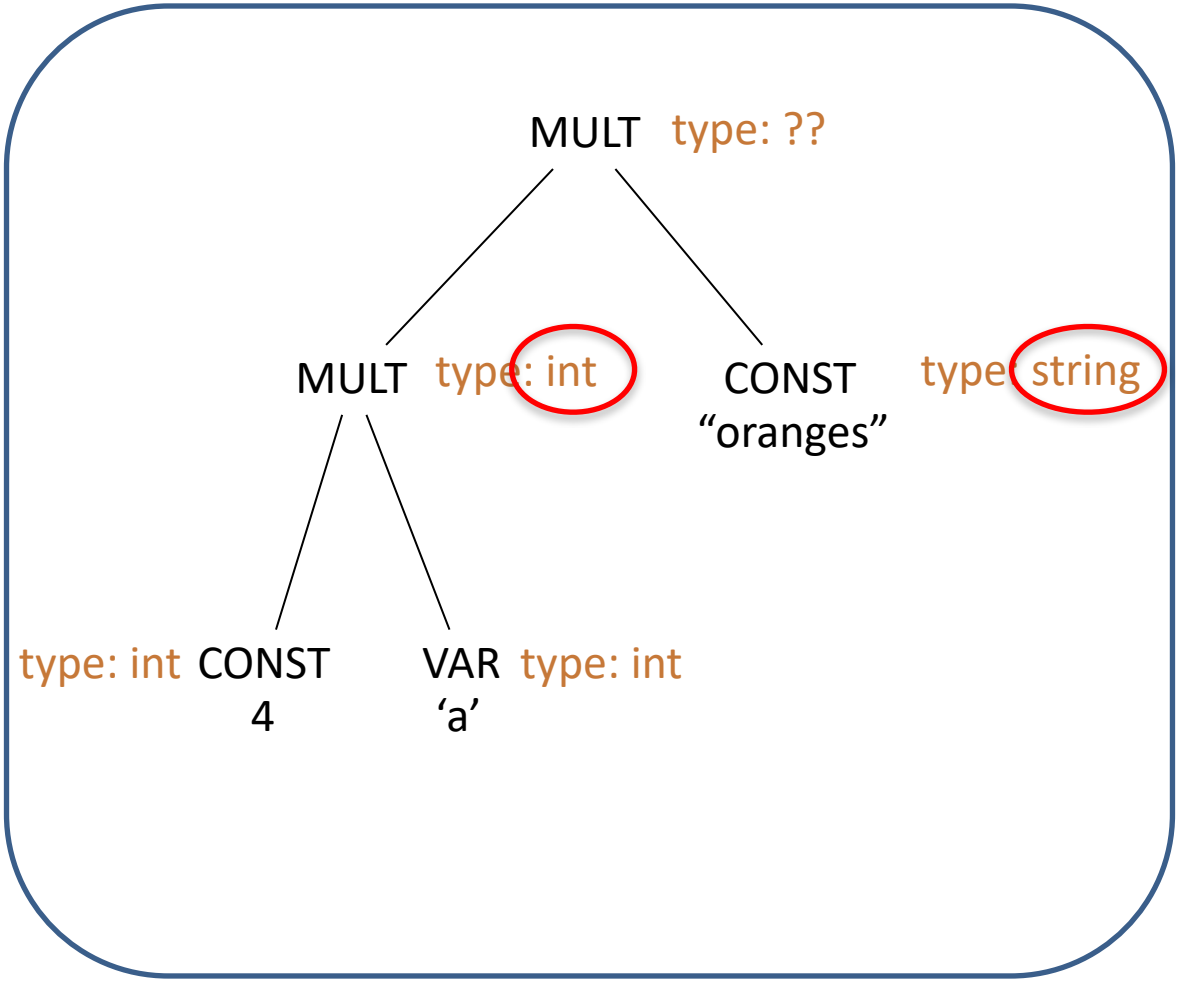


A declaration is not expected inside a call

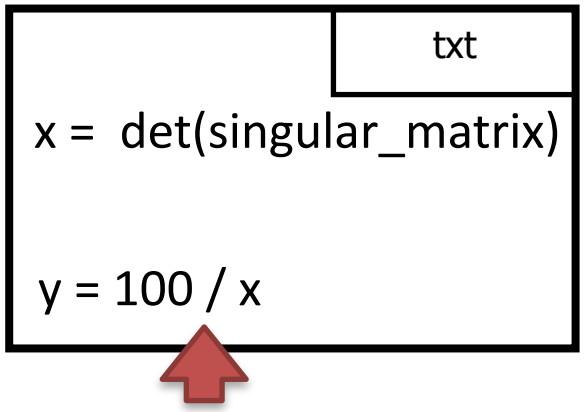
Semantic Errors: Type Checking



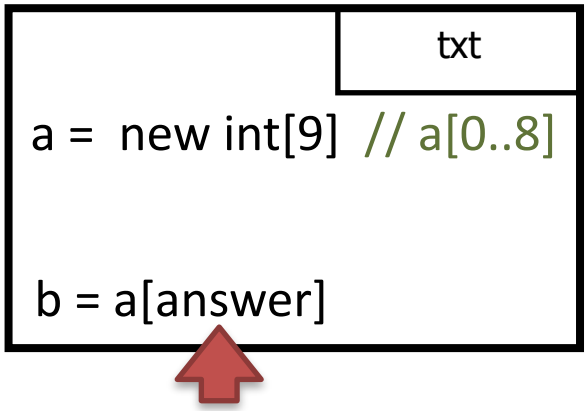
Type mismatch



Runtime Errors

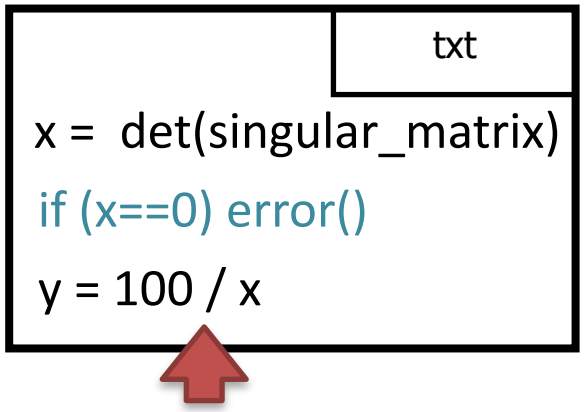


Division by zero

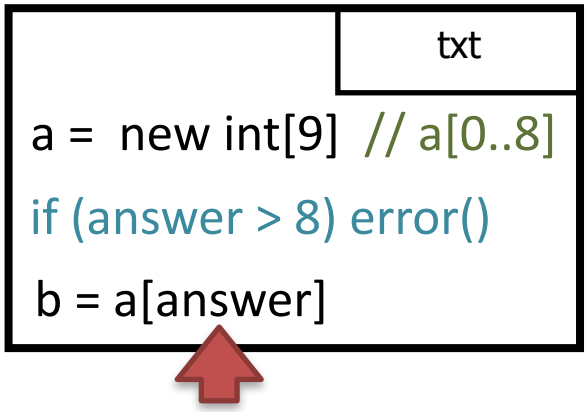


Array index out of bounds: 42 > 8

Code Generated to Capture Runtime Errors



Division by zero



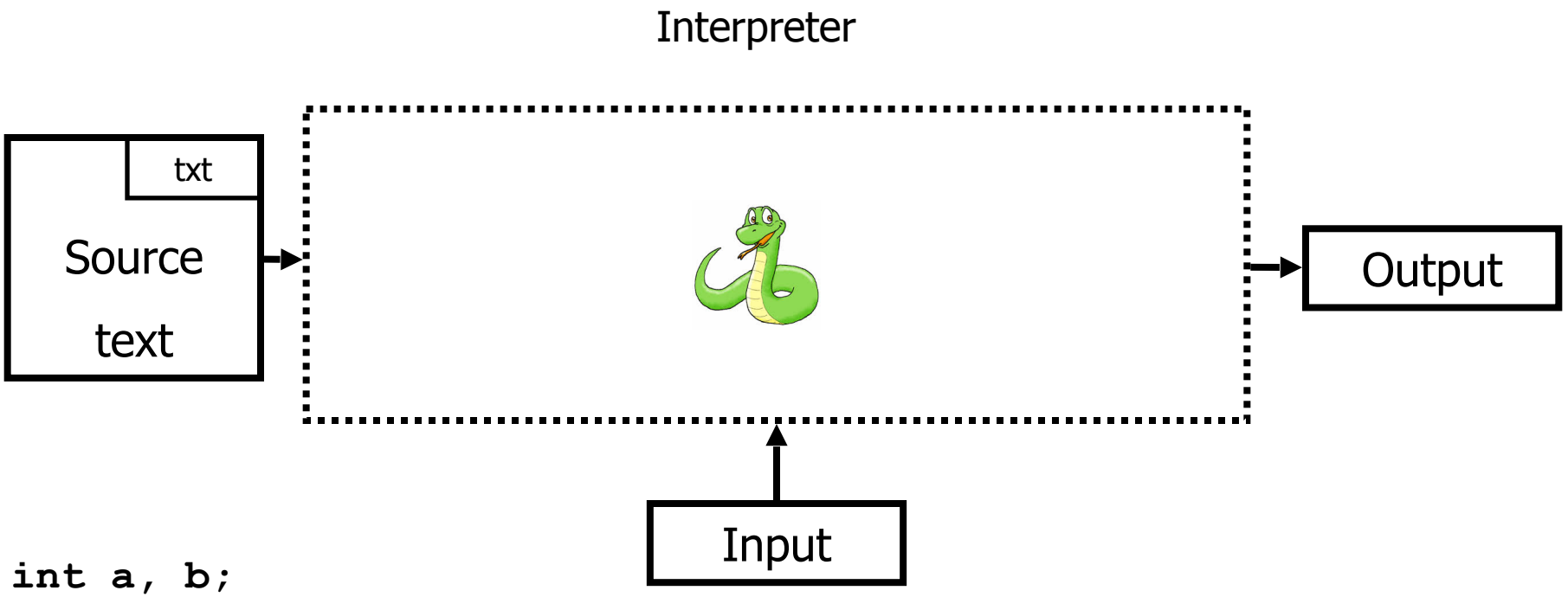
Array index out of bounds: 42 > 8

Shortcuts

- Avoid generating machine code
  - Use assembler
- Generate C code

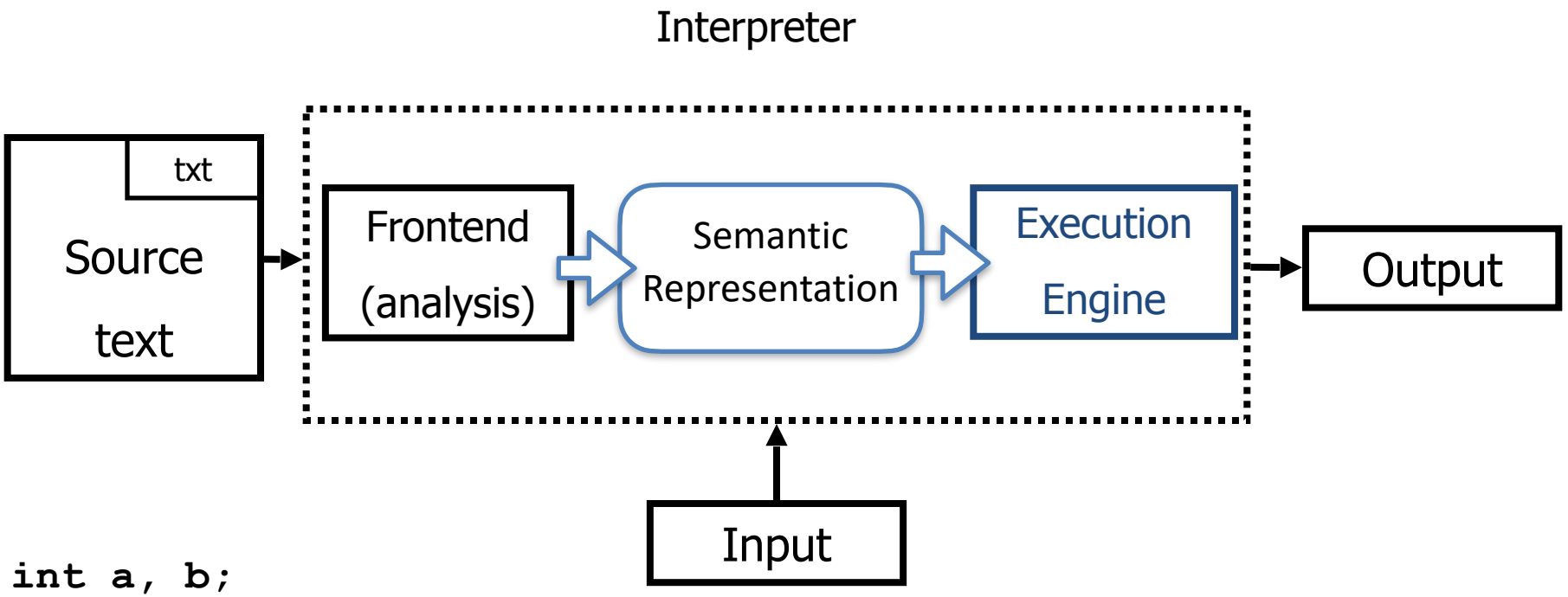
Compilers and Interpreters

What is a Interpreter?



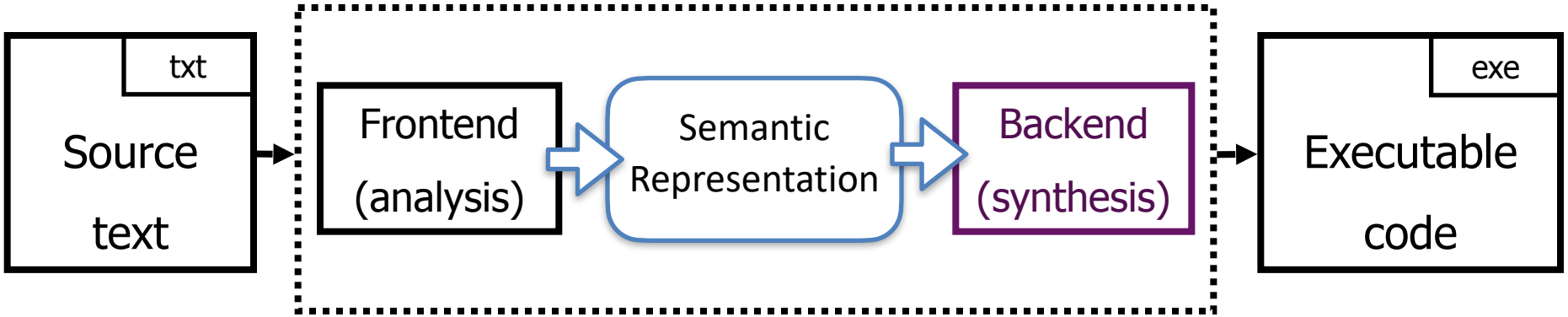

```
int a, b;
a = 2;
b = a*2 + 1;
```

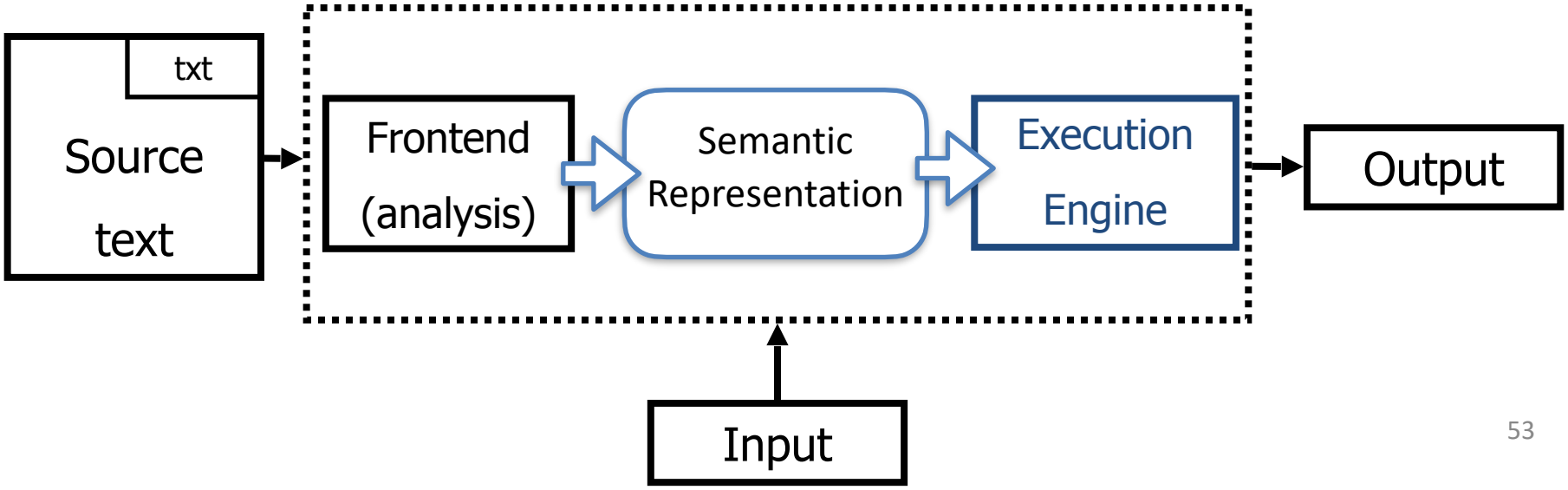
Anatomy of an Interpreter



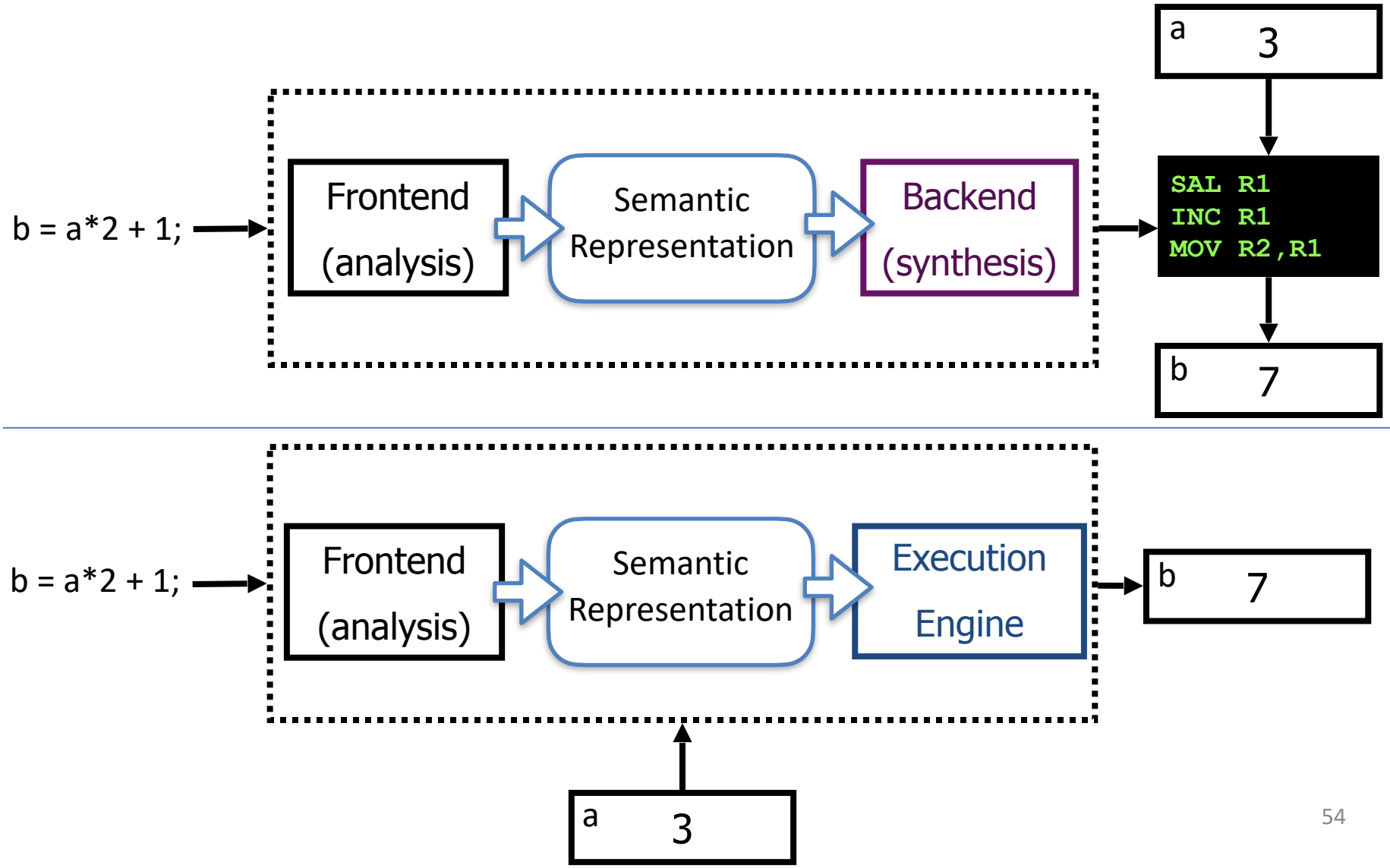

```
int a, b;
a = 2;
b = a*2 + 1;
```

Compiler vs. Interpreter

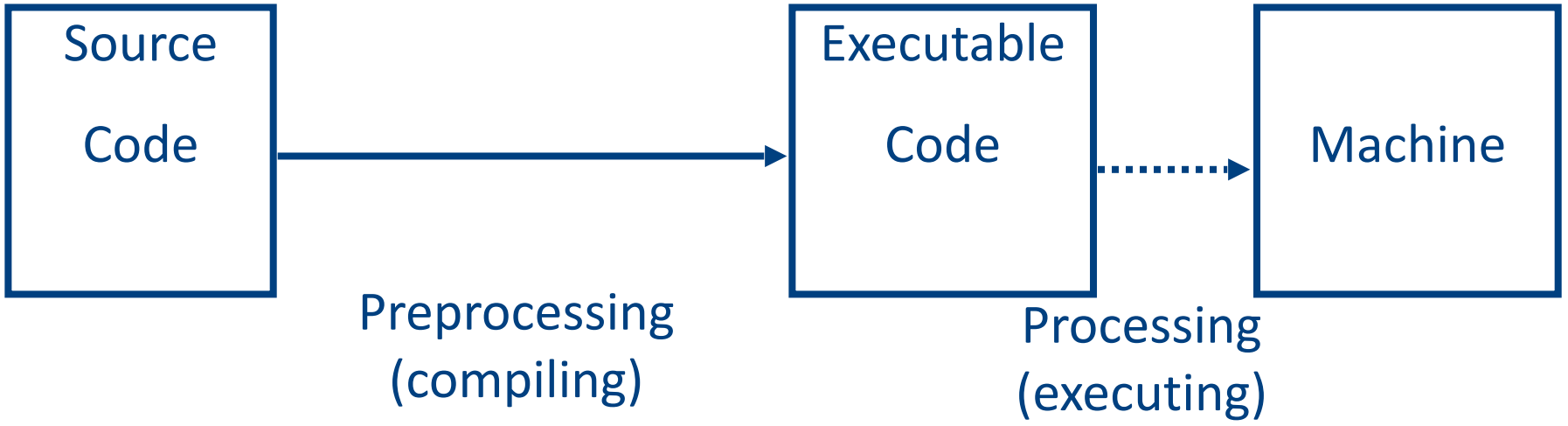


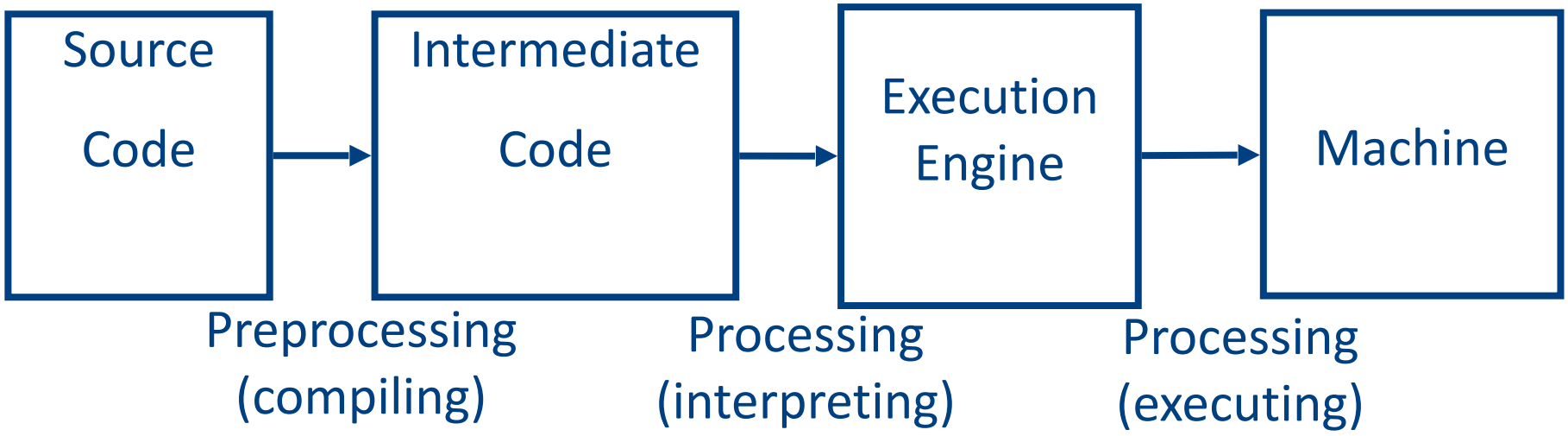


Compiler vs. Interpreter



Compiler vs. Interpreter





Compiled programs are usually more efficient than interpreted

```
scanf(“%d”,&x);
y = 5 ;
z = 7 ;
x = x + y * z;
printf(“%d”,x);
```

Compiler


```
add   %fp,-8, %l1
mov   %l1, %o1
call  scanf
mov   5, %l0
st    %l0,[%fp-12]
mov   7,%l0
st    %l0,[%fp-16]
ld    [%fp-8],%l0
add   %l0, 35 ,%l0
st    %l0,[%fp-8]
ld    [%fp-8], %l1
mov   %l1, %o1
call  printf
```

Compilers report input-independent possible errors

- Input-program

```
scanf(“%d”,  &y);
if (y < 0)
      x = 5;
...
if (y <= 0)
      z = 1 / (x + 5 * y);
```

- Compiler-Output
  - “line 88: x may be used before set”

Interpreters report input-specific definite errors

- Input-program

```
scanf(“%d”,  &y);
if (y < 0)
      x = 5;
...
if (y <= 0)
      z = 1 / (x + 5 * y);
```

- Input data
  - y = -1 : division by zero
  - y = 0 : variable x is not defined
  - y = 1 : ✔

Compiler Interpreter

- Programs written in high-level languages
- Process programs (written in high-level languages)
- Share functionality

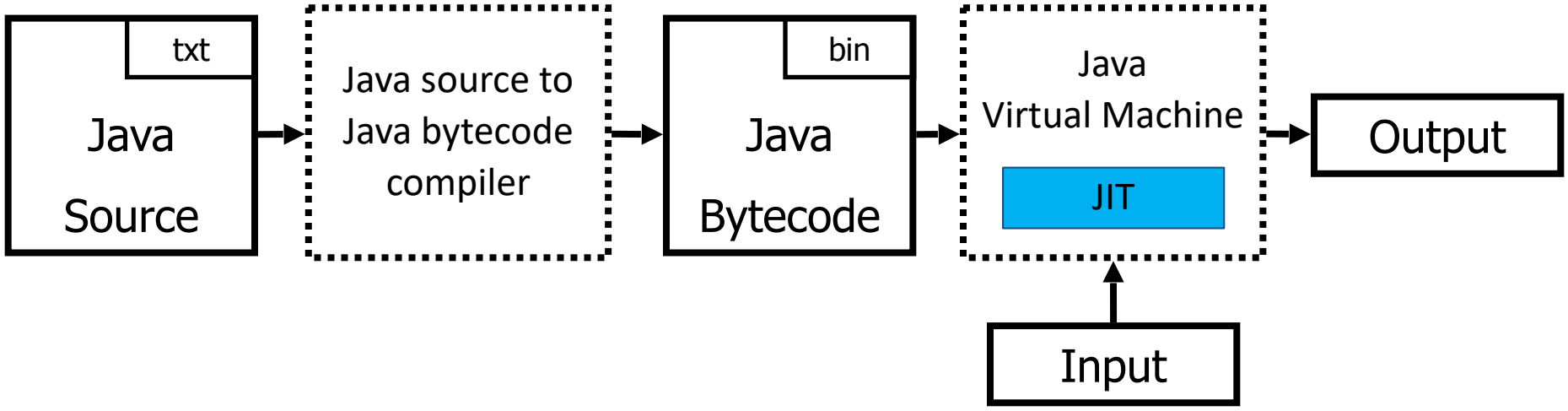
- Transform code
- Report potential errors
- More efficient code

- Run code
- Report input-specific (definite) errors
- More reponsive
- Conceptually simpler
  - “Define the language”
- Easier to port

“Compiled Lang.” vs. “Interpreted Lang.”

Virtual machines
JIT compilers

Just-in-time Compiler (Java HotSpot VM)



Just-in-time (JIT) compilation: bytecode interpreter (in the JVM) compiles hot program fragments to native machine code during interpretation to avoid expensive re-interpretation

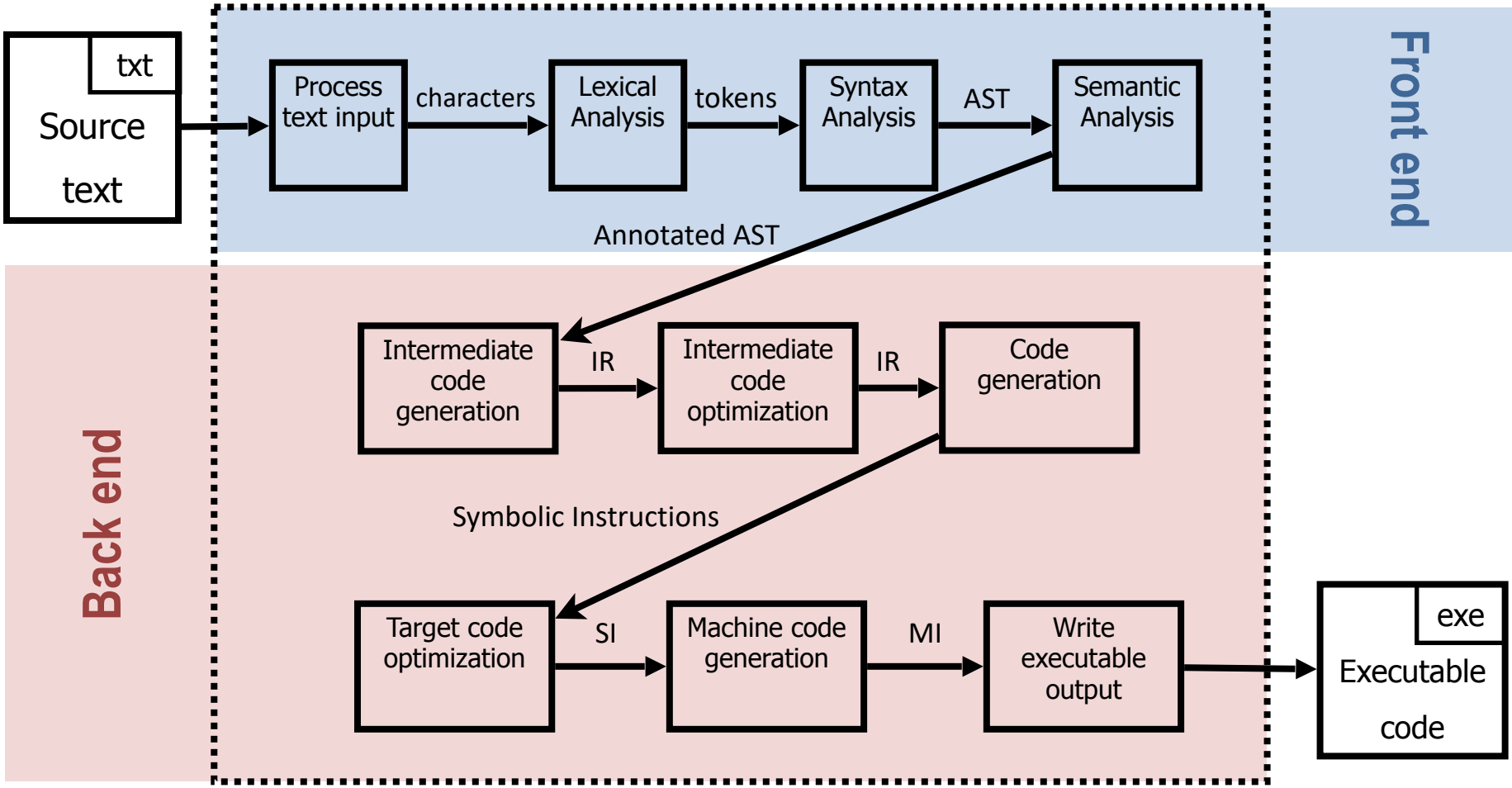
- C1 compiler - a quick compiler (simple optimizations)
- C2 compiler - a slow compiler (aggressive advance optimization for"very hot" code fragments)


conclusion

Summary

- ✓ Compiler = a program that translates code from source language (high level) to target language (low level)
- ✓ Compiler constructed from modular phases
  - Reusable components
  - Different front/back end combinations
- ✓ Compilers play a critical role
  - Bridge programming languages to the machine
  - Many useful techniques and algorithms
  - Many useful tools (e.g., lexer/parser generators)

The Real Anatomy of a Compiler



Compiler: Past vs Present

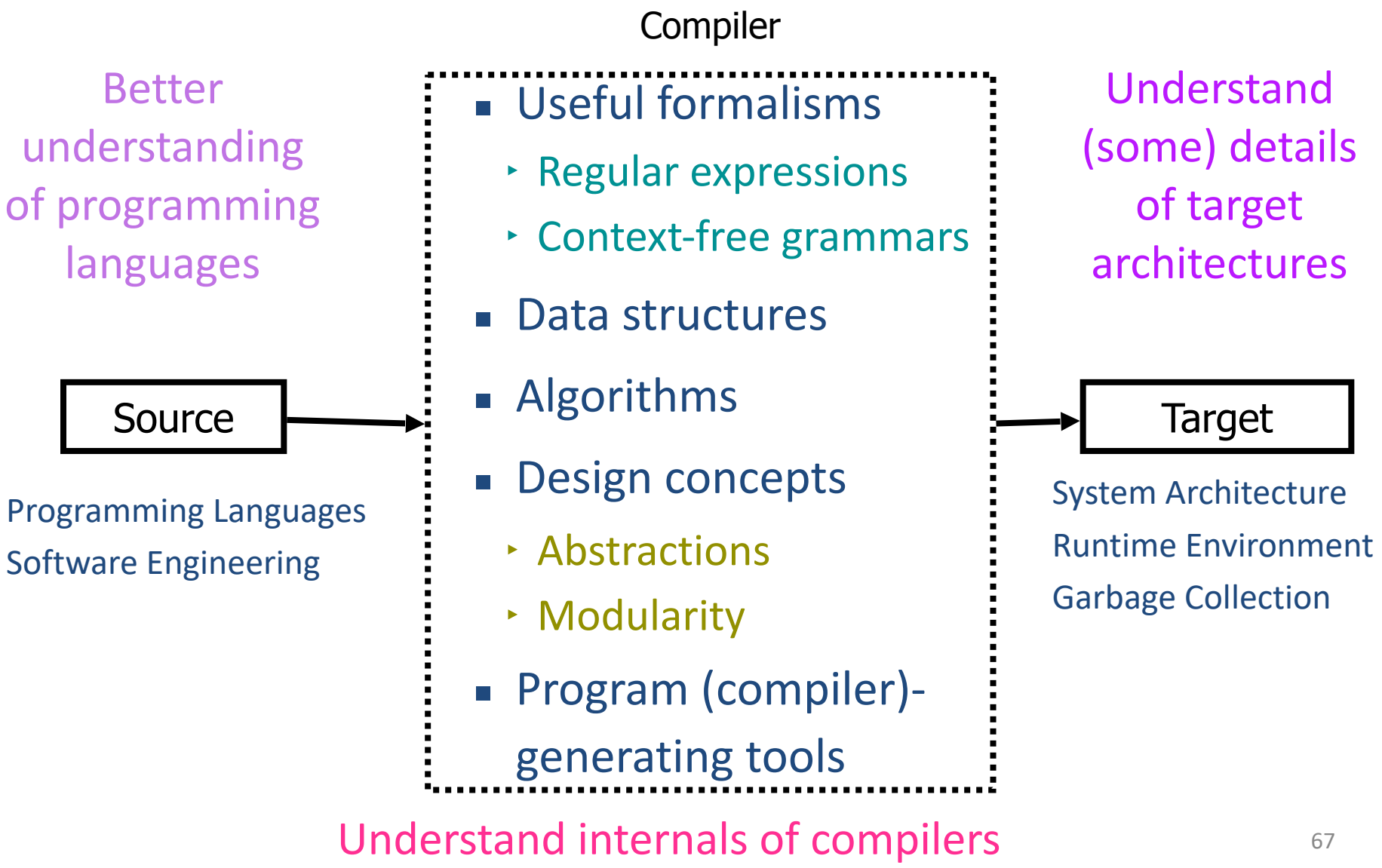
- Past: Lexical and syntactic analysis considered difficult

- Today: Lexical and syntactic analysis “solved.” Bulk effort in analysis and optimizations

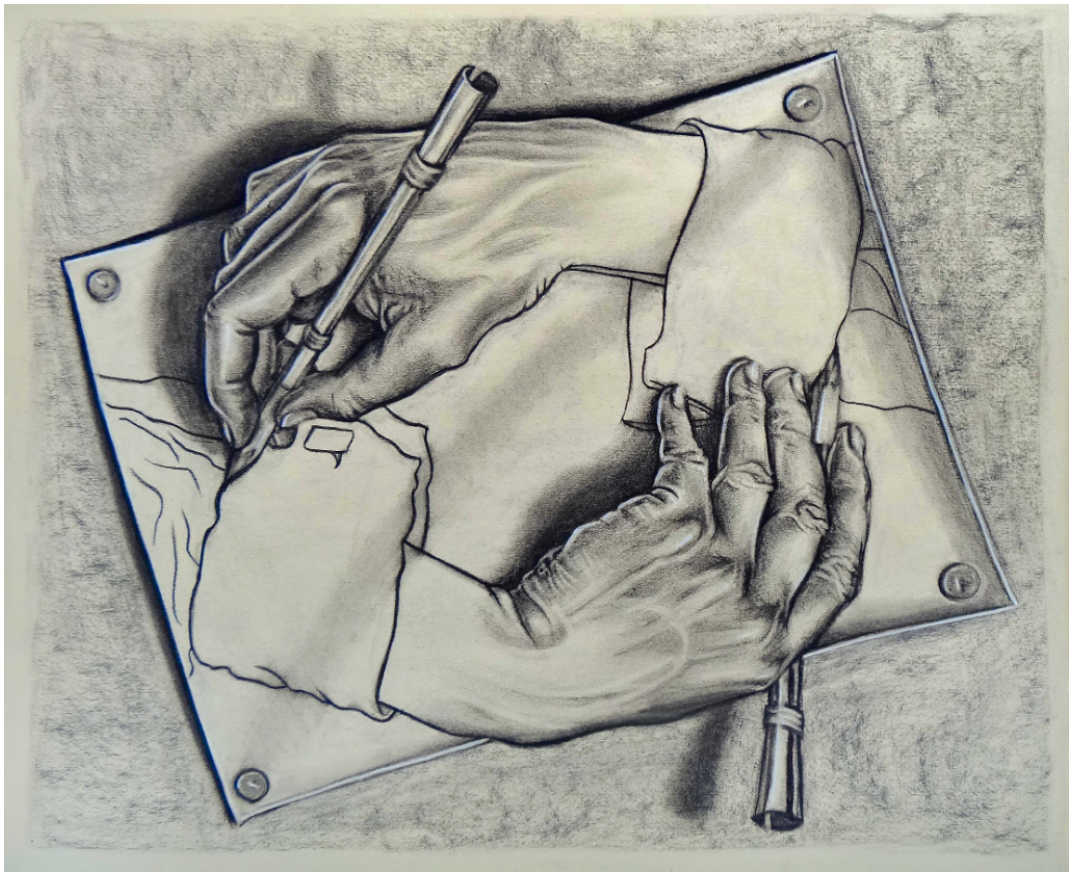
Why should you care?

- Every person in this class will build a “compiler” some day
  - Or wish they knew how to build one…
- But that is not all!

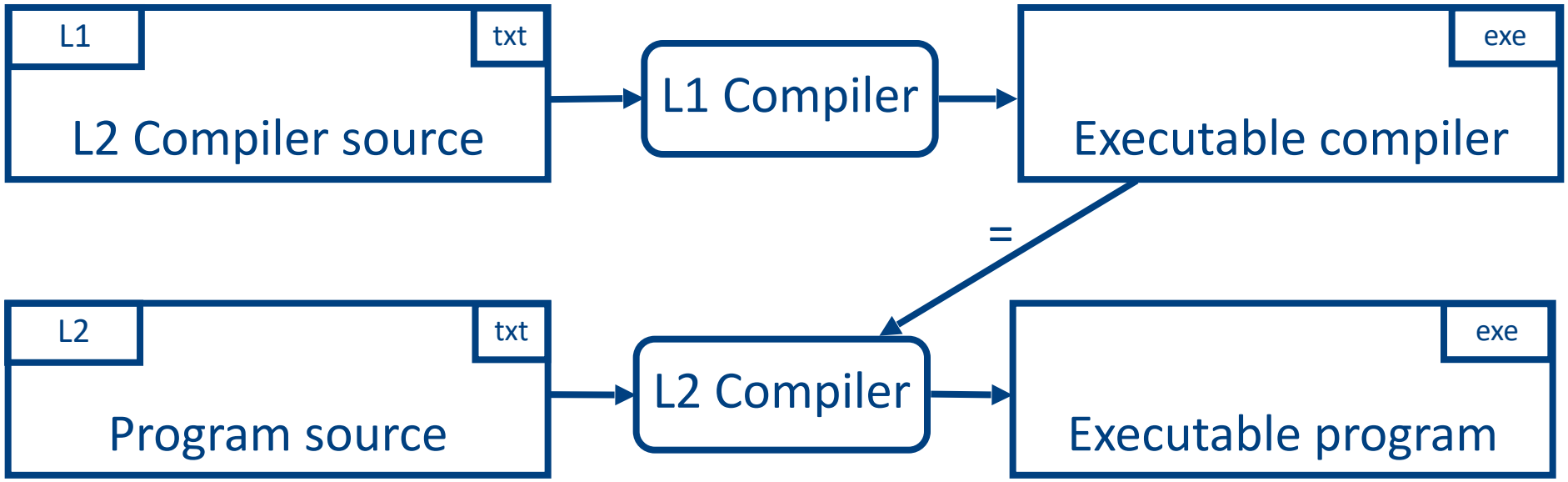
Why should you care?



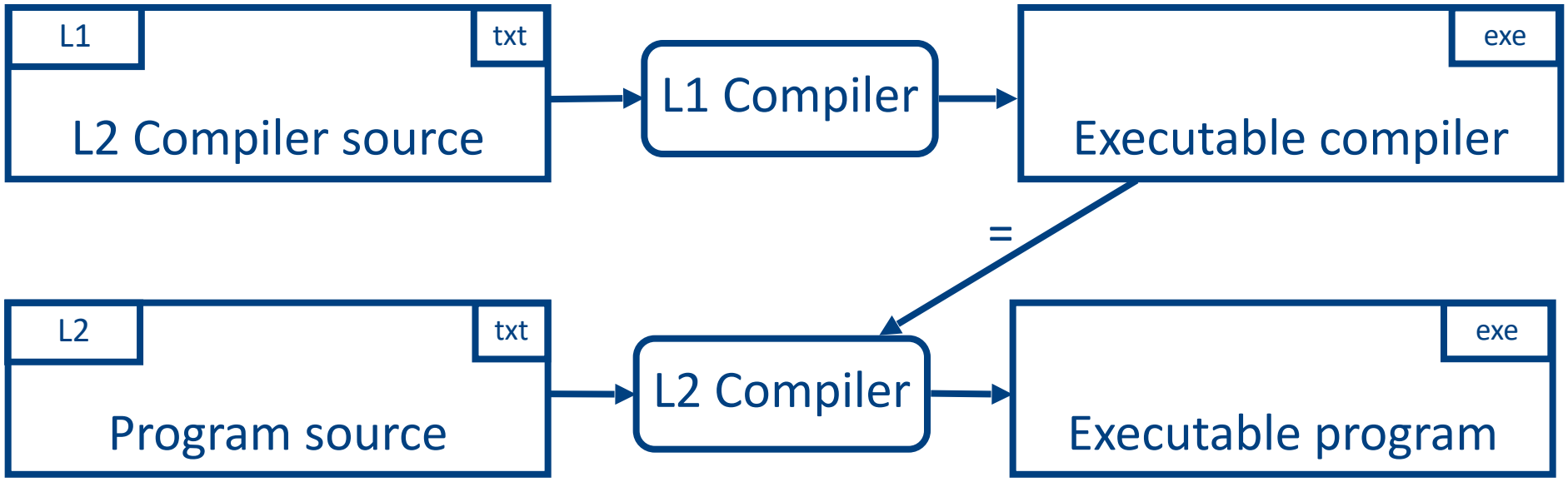
One More Thing:
How to compiler a compiler?



How to compile a compiler?



How to compile a compiler?



Can a compiler for L2 be written in L2?

Can a compiler compile itself?

Bootstrapping a compiler

L1 Simple txt

L2^S compiler source

L1 Compiler

simple exe

L2^s executable compiler

=

L2 advanced txt

L2 compiler source

L2^s Compiler

Inefficient adv. exe

L2^I executable compiler

=

L2 advanced txt

L2 compiler source

L2^I Compiler

Efficient adv. exe

L2 executable compiler

The End