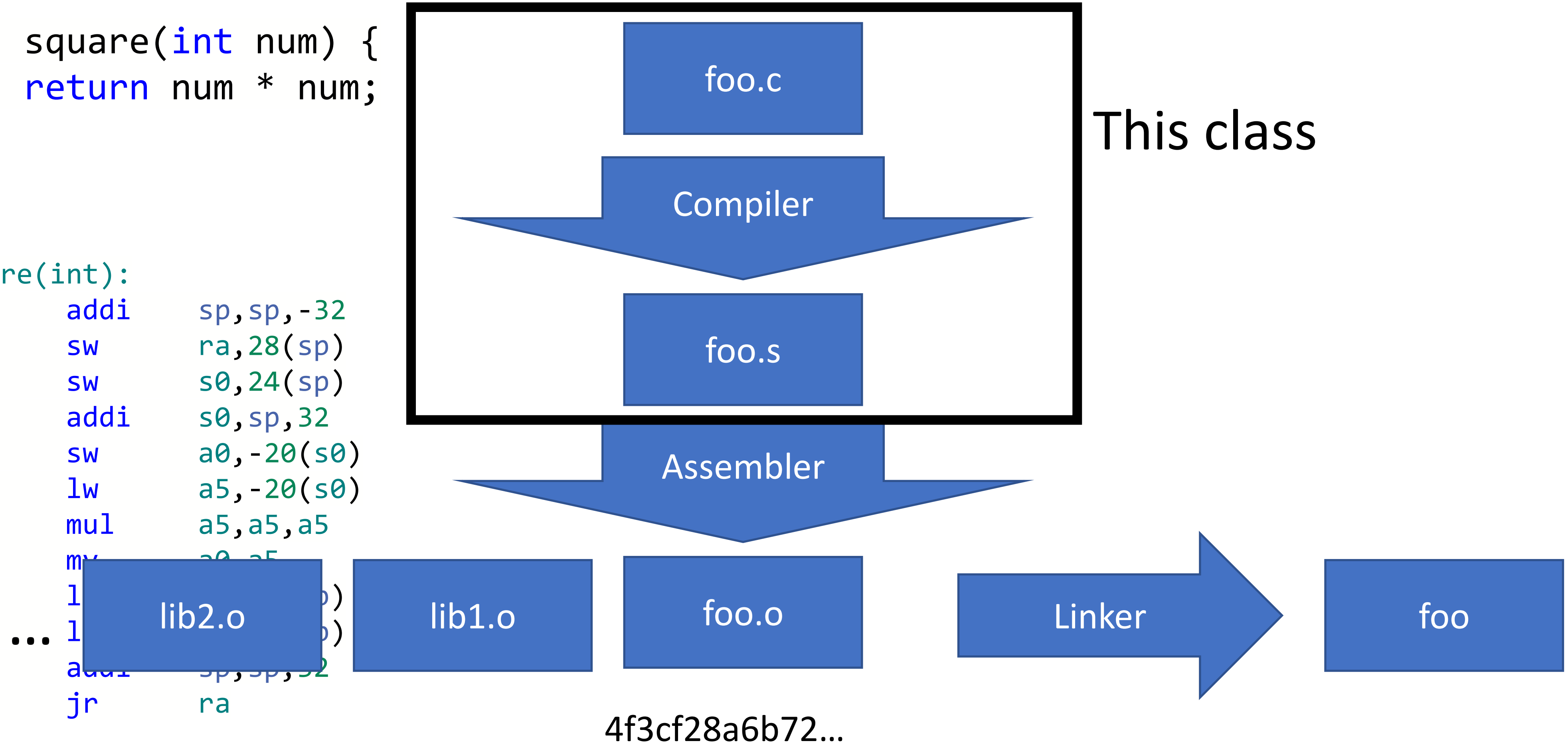
CS443: Compiler Construction

Lecture 0

What happens when you call gcc?

```c
int square(int num) {
    return num * num;
}
```

```
square(int):
        addi    sp,sp,-32
        sw      ra,28(sp)
        sw      s0,24(sp)
        addi    s0,sp,32
        sw      a0,-20(s0)
        lw      a5,-20(s0)
        mul     a5,a5,a5
        mv      a0,a5
        l               )
        l               )
        addi    sp,sp,32
        jr      ra
```



foo.c

Compiler

foo.s

This class

Assembler

...

lib2.o

lib1.o

foo.o

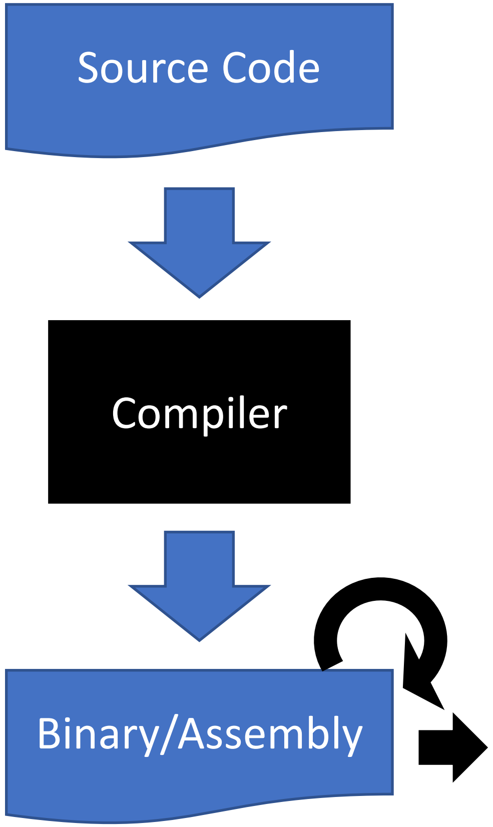
Linker

foo

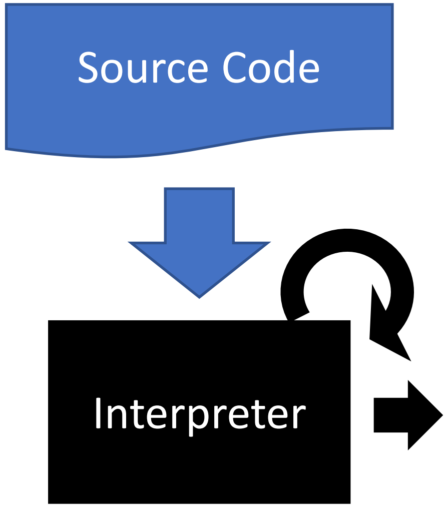
4f3cf28a6b72...

(Code examples courtesy godbolt.org)

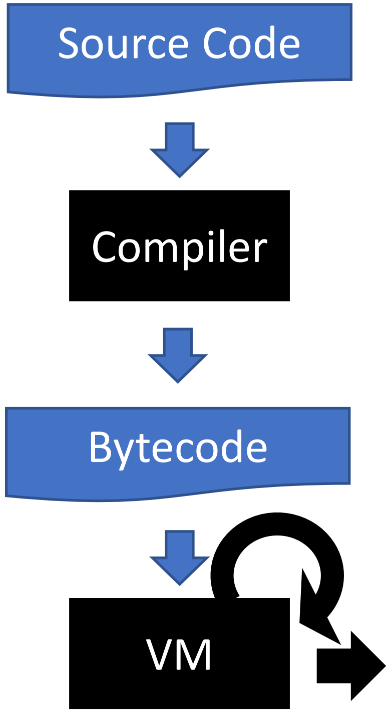
There are different ways of translating a programming language



Ex.: C, C++

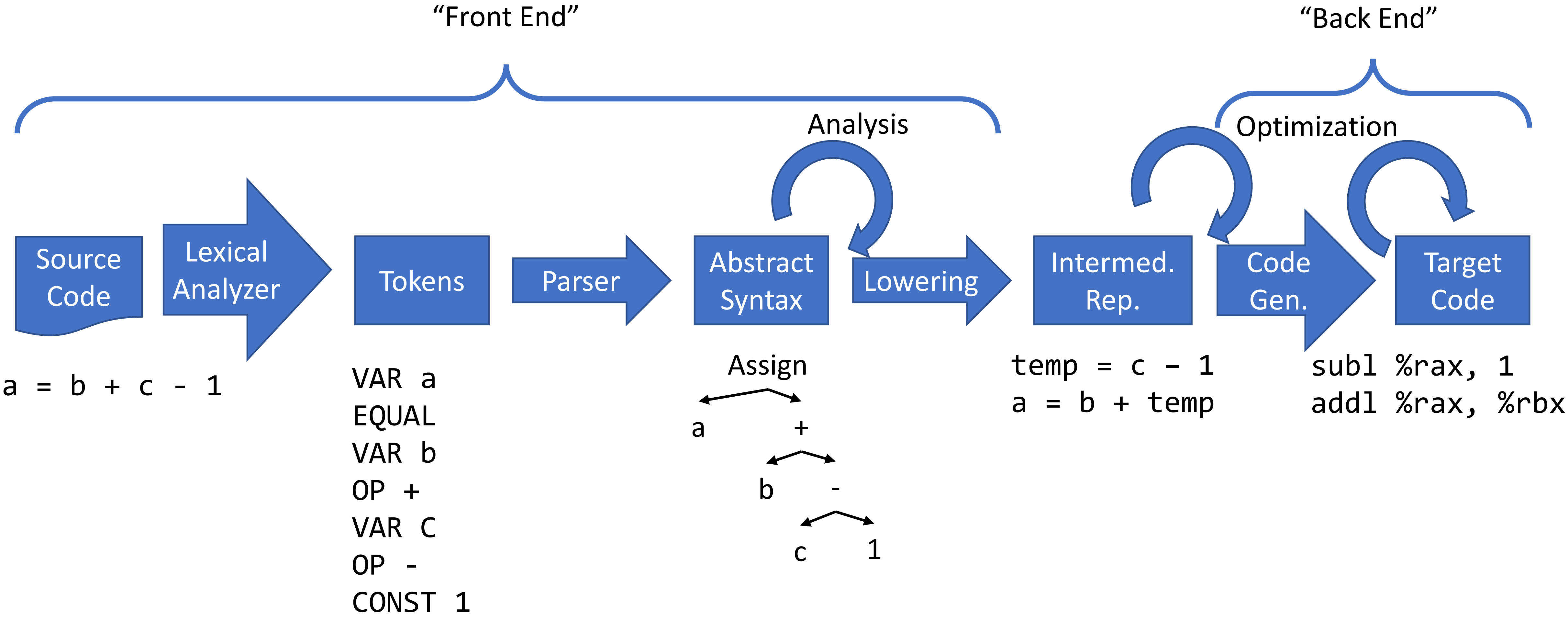


Ex.: Python



Ex.: Java

Compilers translate code in phases



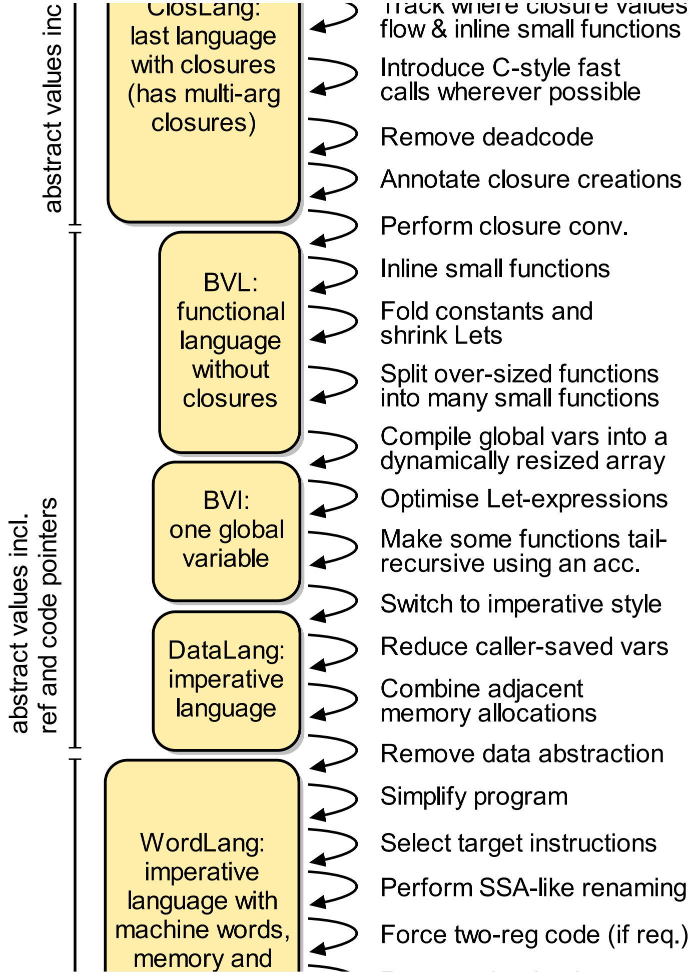
```
a = b + c - 1
```

```
VAR a
EQUAL
VAR b
OP +
VAR C
OP -
CONST 1
```

```
temp = c – 1
a = b + temp
```

```
subl %rax, 1
addl %rax, %rbx
```

May have many more phases, several intermediate representations



abstract values inc

ClosLang: last language with closures (has multi-arg closures)

abstract values incl. ref and code pointers

BVL: functional language without closures

BVI: one global variable

DataLang: imperative language

WordLang: imperative language with machine words, memory and

Track where closure values flow & inline small functions

Introduce C-style fast calls wherever possible

Remove deadcode

Annotate closure creations

Perform closure conv.

Inline small functions

Fold constants and shrink Lets

Split over-sized functions into many small functions

Compile global vars into a dynamically resized array

Optimise Let-expressions

Make some functions tail-recursive using an acc.

Switch to imperative style

Reduce caller-saved vars

Combine adjacent memory allocations

Remove data abstraction

Simplify program

Select target instructions

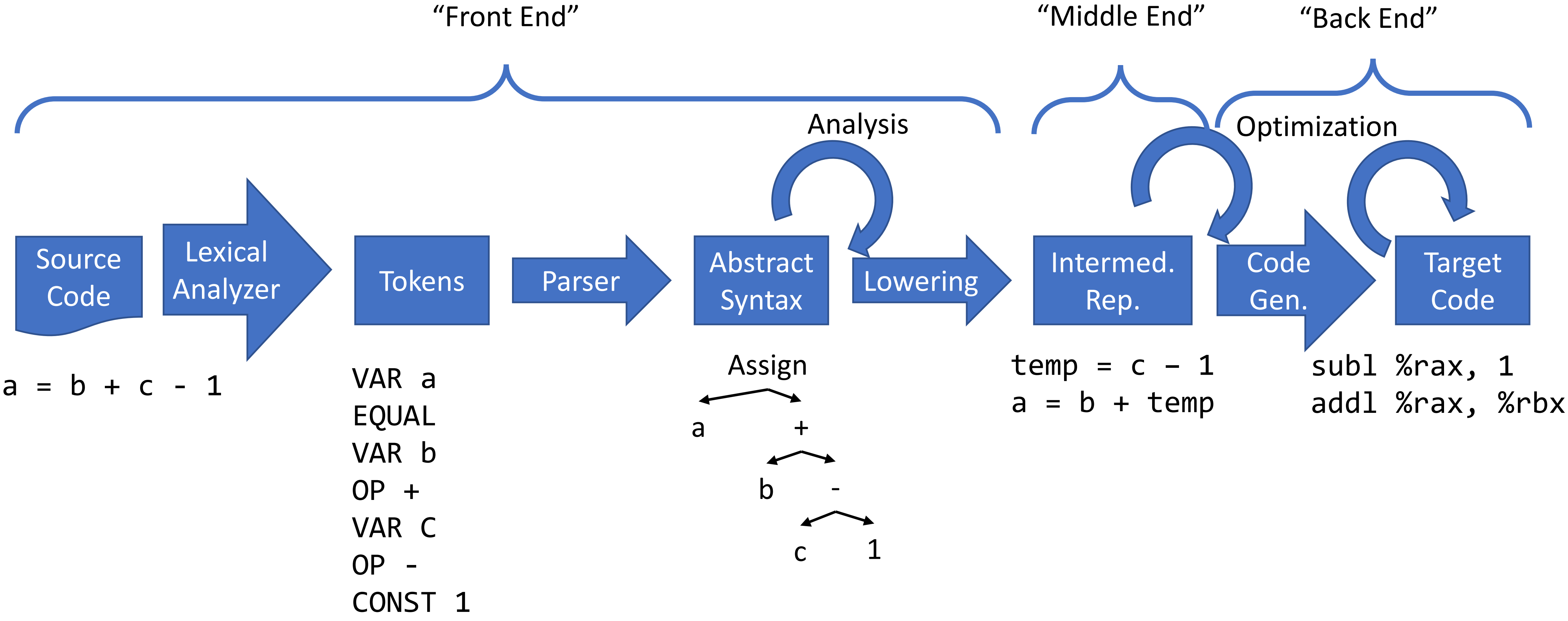
Perform SSA-like renaming

Force two-reg code (if req.)

CAKEML

A Verified Implementation of ML

Front End is language specific
Back End is machine specific



“Front End”

“Middle End”

“Back End”

Analysis

Optimization

Source Code

Lexical Analyzer

Tokens

Parser

Abstract Syntax

Lowering

Intermed. Rep.

Code Gen.

Target Code

```
a = b + c - 1
```

```
VAR a
EQUAL
VAR b
OP +
VAR C
OP -
CONST 1
```

Assign

a

+

b

-

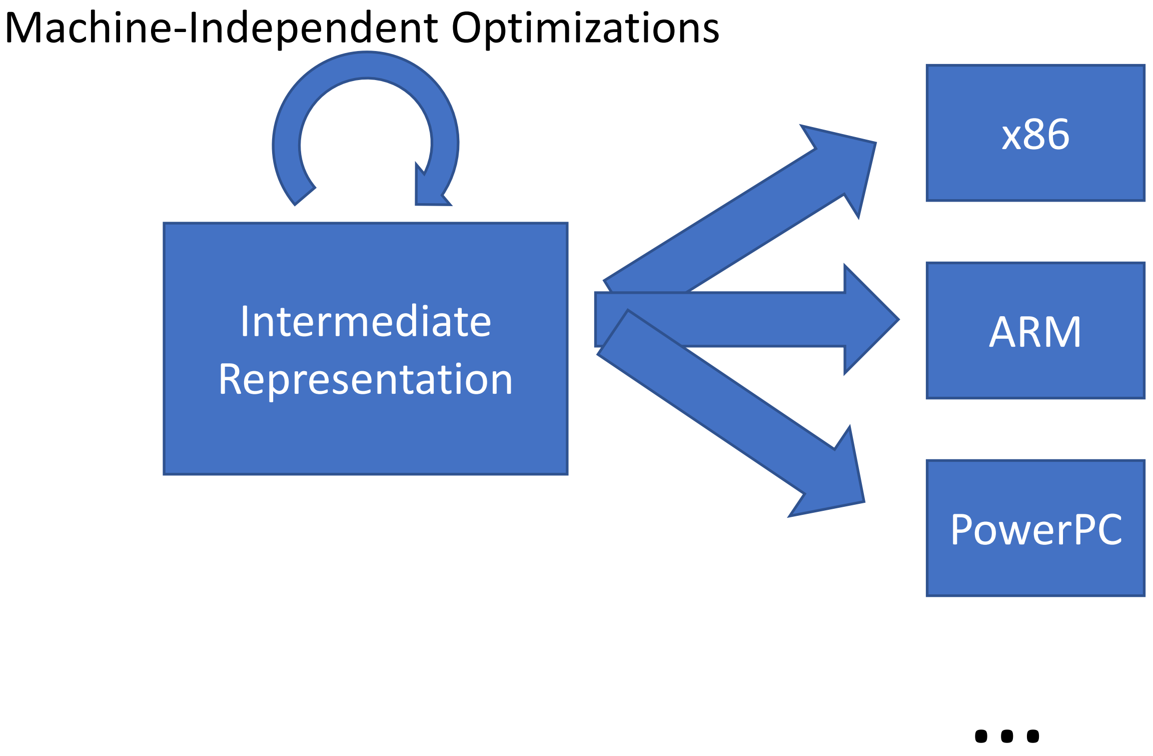
c

1

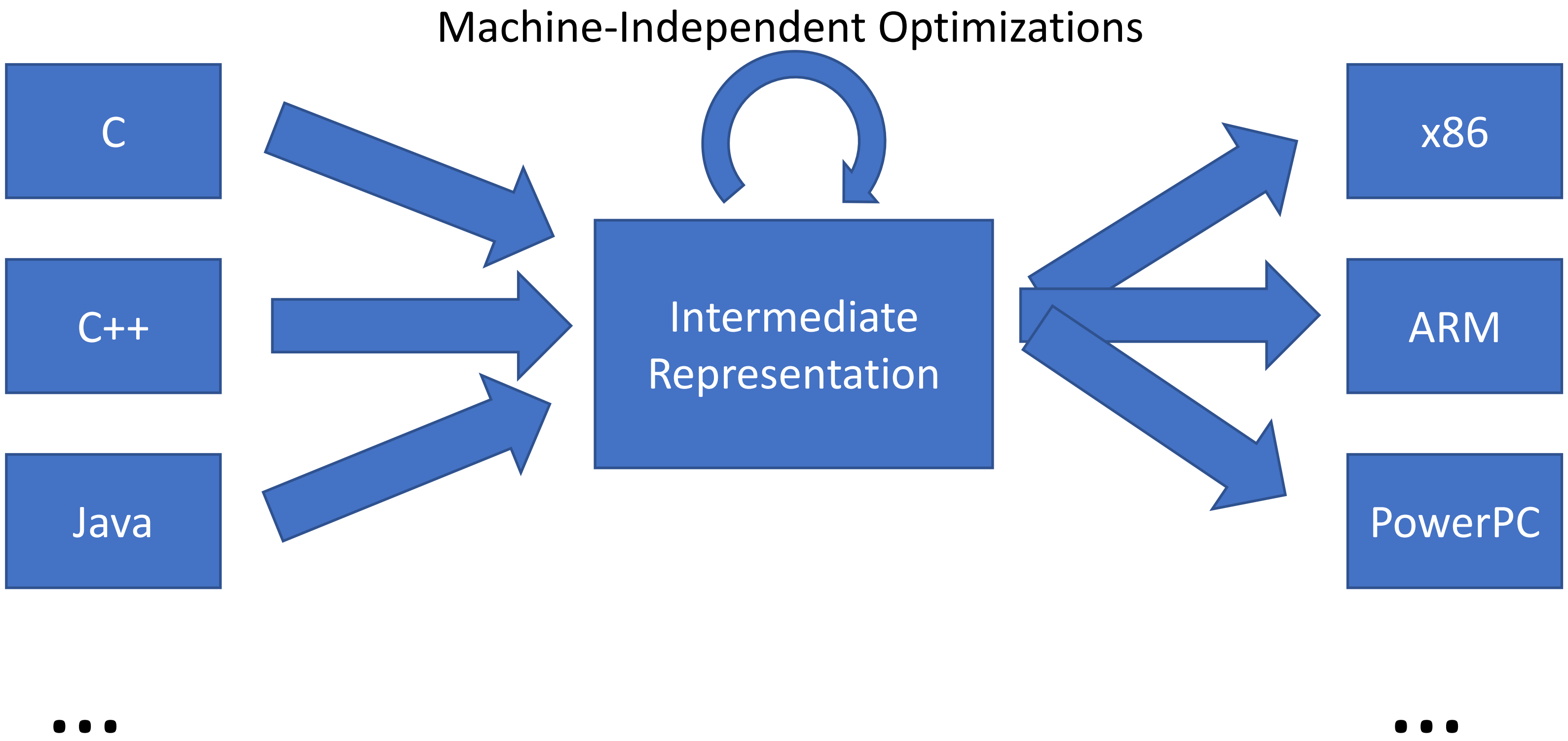
```
temp = c – 1
a = b + temp
```

```
subl %rax, 1
addl %rax, %rbx
```

Can (and usually do) swap out back ends to target different machines


Machine-Independent Optimizations
Intermediate Representation
x86
ARM
PowerPC
…

Compiler collections also swap out front ends for different languages


Machine-Independent Optimizations

C

C++

Java

…

Intermediate Representation

x86

ARM

PowerPC

…

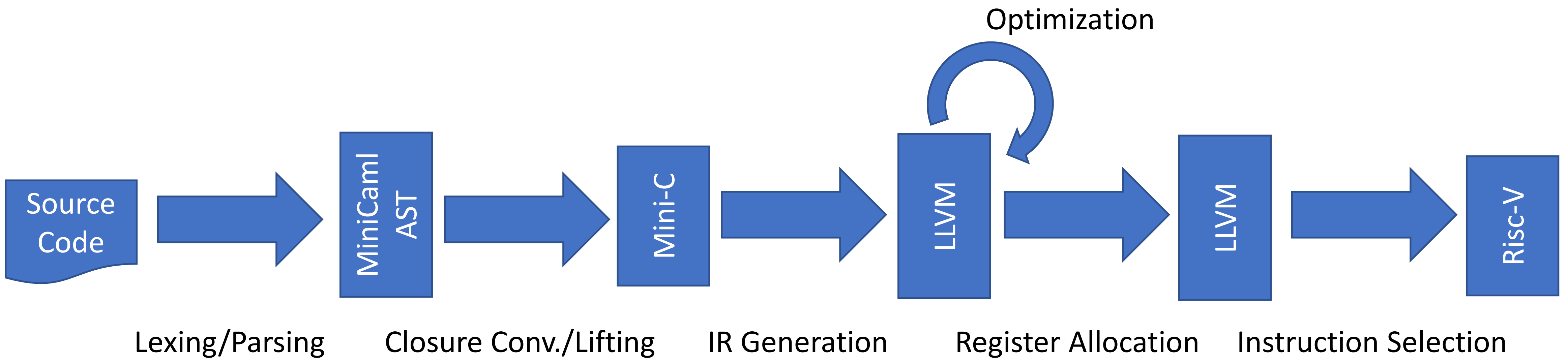
A Small ML Compiler

Higher-order
Typed
Structured Data
Nested Expressions
Unlimited Variables

First-order
Typed
Structured Data
Nested Expressions
Unlimited Variables

First-order
Typed
Structured Data
Flat Expressions
Unlimited Variables

First-order
Untyped
No Structured Data
Flat Expressions
32 Hardware Registers



Optimization

Source Code

MiniCaml AST

Mini-C

LLVM

LLVM

Risc-V

Lexing/Parsing

Closure Conv./Lifting

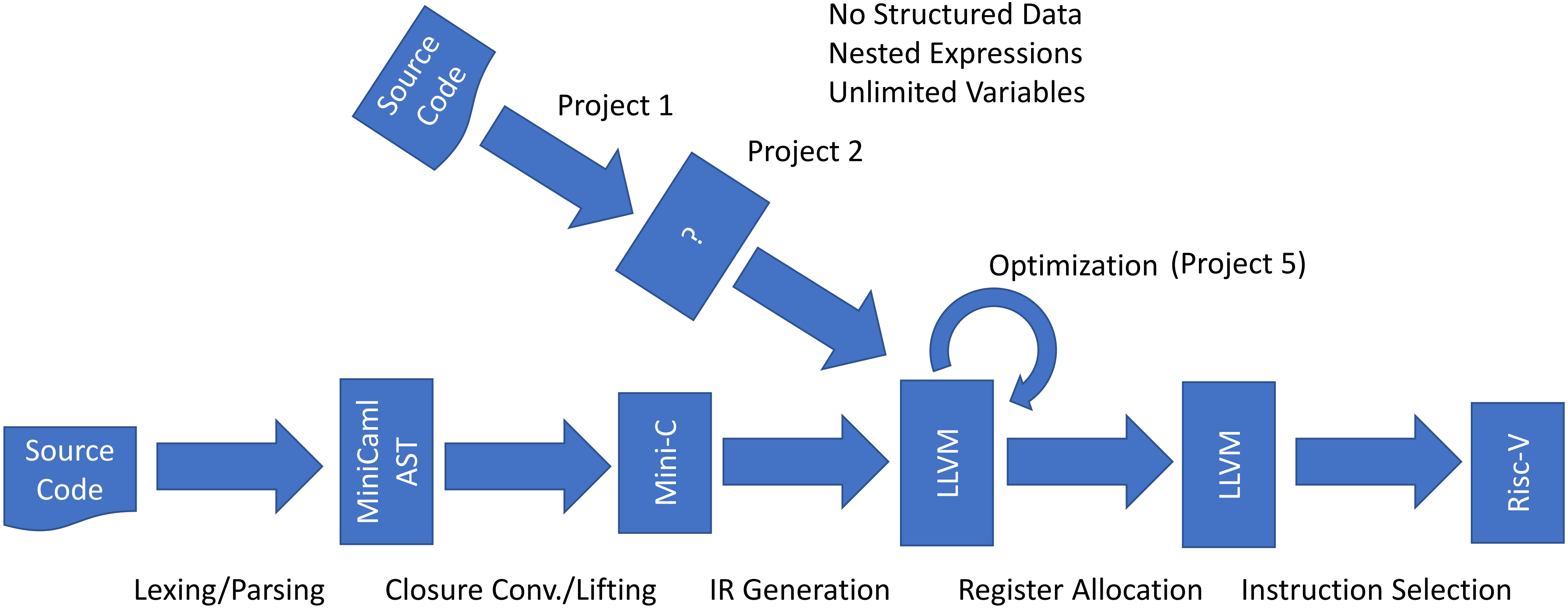
IR Generation

Register Allocation

Instruction Selection

Course Projects

First-order
Typed
No Structured Data
Nested Expressions
Unlimited Variables



(Project 4)

(Project 3)

(Project 6)

Projects

- ~7 projects, 2-3 weeks each (Except Project 0, Due 8/29)
- Mostly (entirely?) programming – graded with automated tests
- Work individually or in pairs
- Handed out + submitted via GitHub

Late Days:

- 6 per student, extend deadline 24 hours
- No more than 2 per assignment
- If a pair, must both use a late day*

Fair warning: lots of programming!

More bad news (for most of you)

- Projects will be in **OCaml**
    - Good news: If you know Haskell or Racket, can learn it quickly.
    - Haskell w/o monads
    - Racket w/ **types** and way fewer parens
    - Tutorial on Thursday
        - Try to set it up on your machine by then if you want to follow along

Background

- Prerequisite: CS440 (Programming Languages and Translators)
    - Abstract syntax, working with ASTs (will review very briefly today)
    - Building an interpreter (will review on Project 0)
    - Functional programming
    - If you’re not familiar with the above, I suggest brushing up in the next couple weeks.

Websites to know

- Course website: http://cs.iit.edu/~smuller/cs443-f24/
  - Full syllabus/policies/schedule/lecture notes. Go there.
- Canvas
- Github Classroom (links will be handed out with projects)
- Discord

Exams

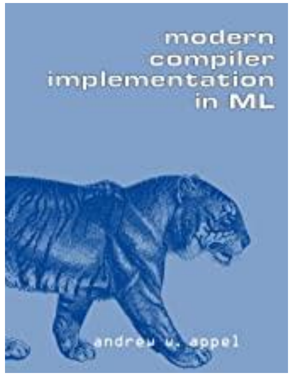
- Midterm (Oct. 15)
- Final Exam (during finals week, schedule posted by Registrar)

- Open book, open notes

Grading

- 50% Projects
- 20% Midterm
- 30% Final

Textbook



- **Appel. *Modern Compiler Implementation in ML***
  (Highly recommended)
  (Also have C, Java versions)

- **OCaml Programming: Correct + Efficient + Beautiful**
  (Free online, link on course website)

Academic Honesty

- Submitted solutions must be your own work (and your partner)
- Can discuss course concepts with other students, but don’t share/look at code.
- If using online resources/code (incl. generative AI):
    - Don’t search for code that substantially solves the assigned problem. Be reasonable.
    - If using small snippets of code, *cite them* (e.g., URL in a comment)

Office Hours

- Wednesday, 10:30-11:30am (Zoom)
- Thursday, 2-3pm (SB 218E)

OK, back to programming languages

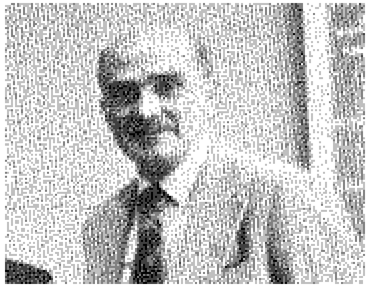
First-order
Typed
No Structured Data
Nested Expressions
Unlimited Variables
Simple
Easy to compile

IITRAN

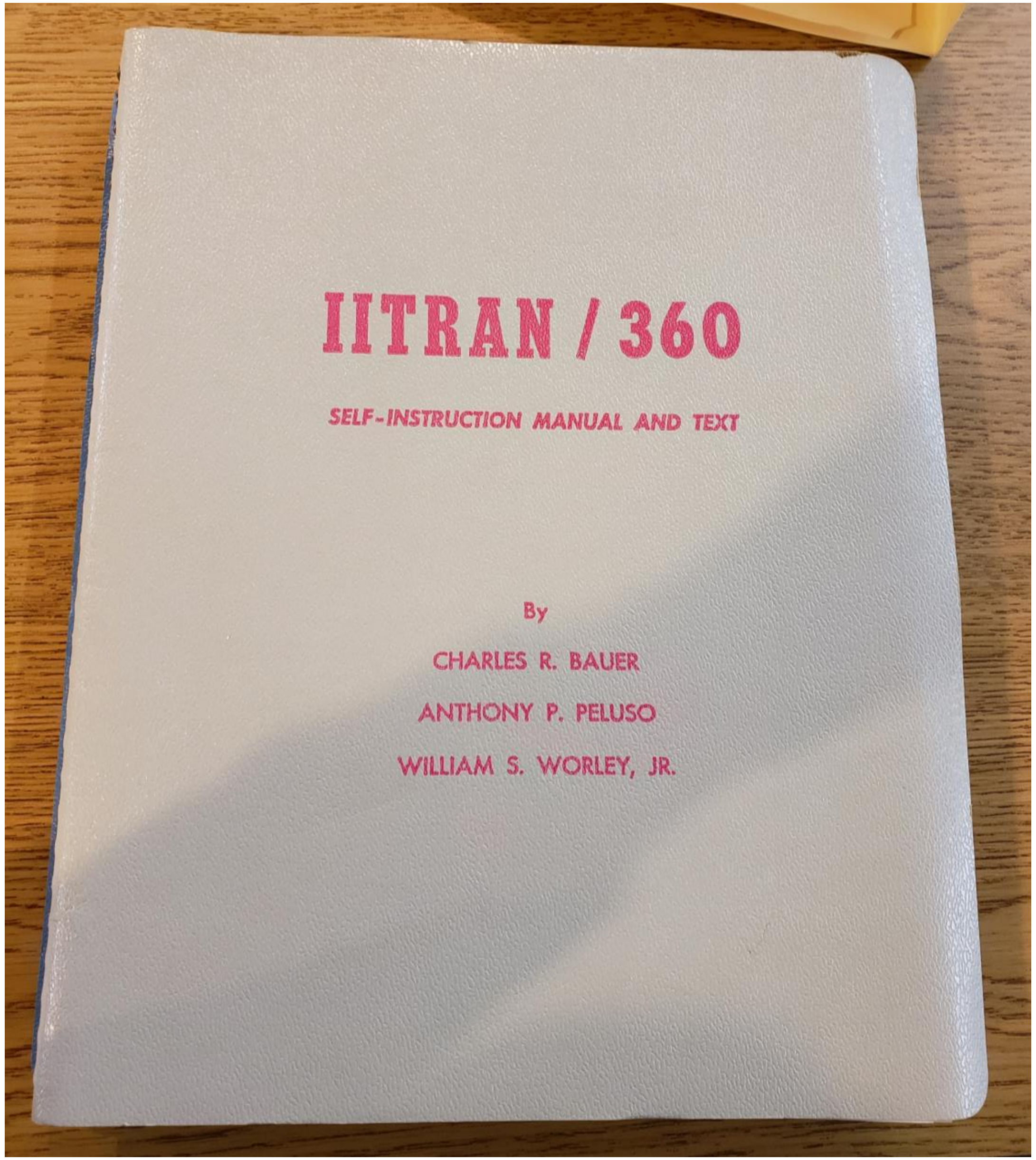
IITRAN/7040 – 1964
IITRAN/360 - 1966

IITRAN

Robert Dewar



Charles Bauer



IITRAN / 360

SELF-INSTRUCTION MANUAL AND TEXT

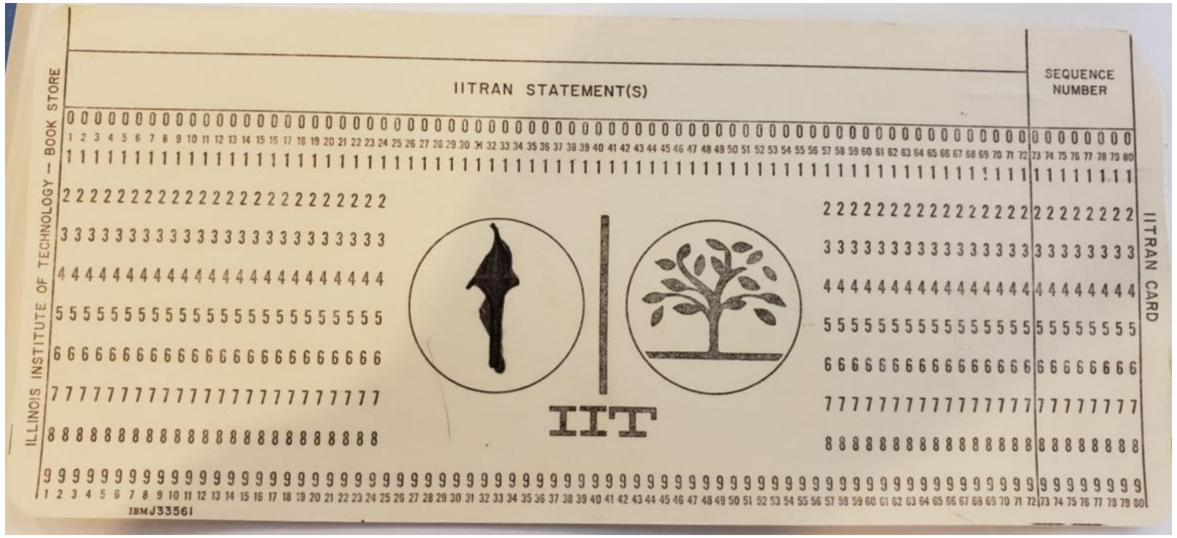
By

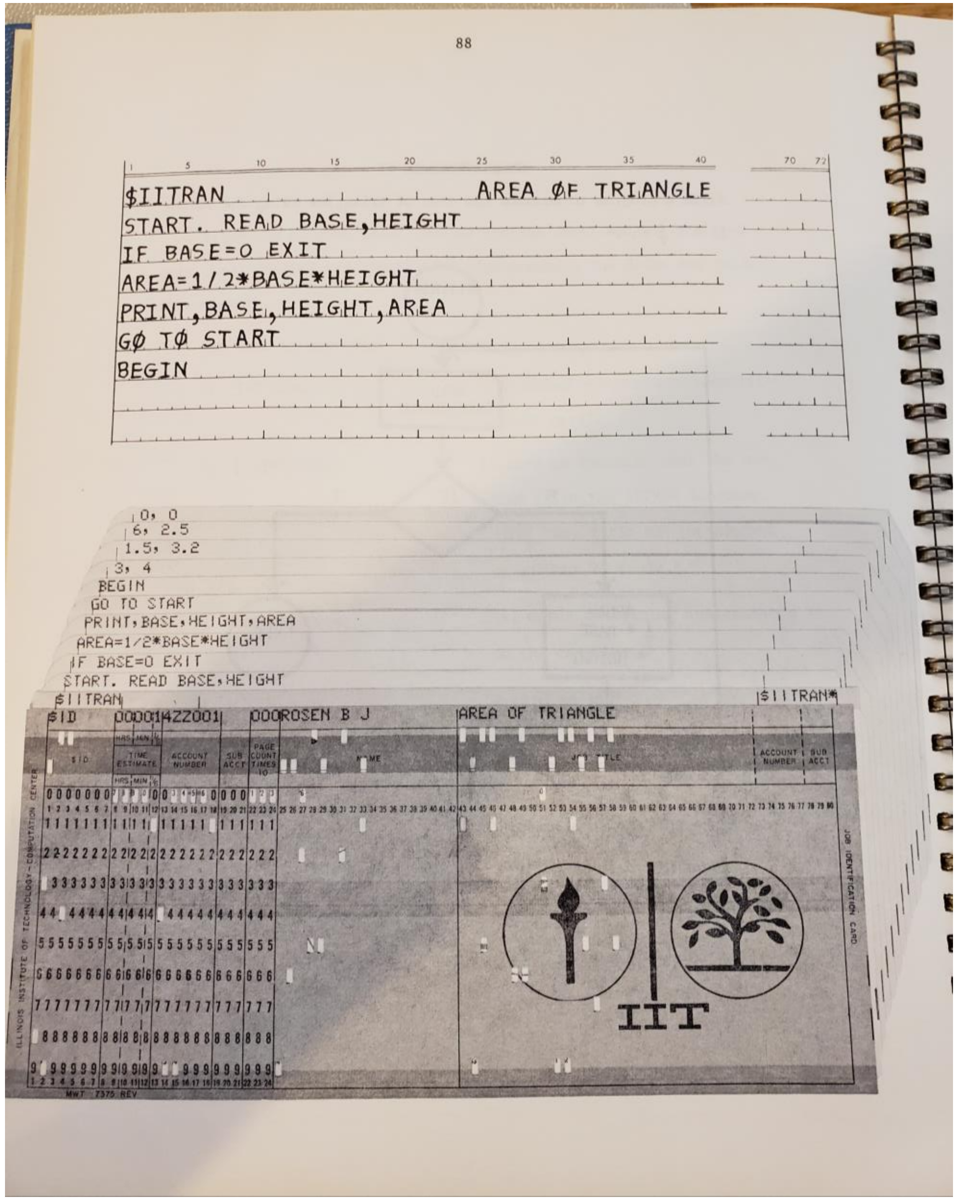
CHARLES R. BAUER

ANTHONY P. PELUSO

WILLIAM S. WORLEY, JR.

IITRAN/360




88

```
$IITRAN                          AREA ØF TRIANGLE
START. READ BASE,HEIGHT
IF BASE=O EXIT
AREA=1/2*BASE*HEIGHT
PRINT,BASE,HEIGHT,AREA
GØ TØ START
BEGIN
```

```
0, 0
6, 2.5
1.5, 3.2
3, 4
BEGIN
GO TO START
PRINT,BASE,HEIGHT,AREA
AREA=1/2*BASE*HEIGHT
IF BASE=0 EXIT
START. READ BASE,HEIGHT
$IITRAN                                                   $IITRAN*
$ID   00001 4ZZ001   000ROSEN B J        AREA OF TRIANGLE
```

$ID
TIME ESTIMATE
ACCOUNT NUMBER
SUB ACCT
PAGE COUNT TIMES 10
NAME
JOB TITLE
ILLINOIS INSTITUTE OF TECHNOLOGY—COMPUTATION CENTER
JOB IDENTIFICATION CARD
IIT
MWT 7375 REV

Abstract Syntax

- BNF (Backus-Naur Form)

type ::= INTEGER | CHARACTER | LOGICAL

Type casts

bop ::= + | - | * | / | <-   *uop* ::= ~ | NOT | INT | CH | LG

exp ::= x | *num* | *char* | *exp bop exp* | *uop exp*

stmt ::= STOP | IF *exp* THEN *stmt* (ELSE *stmt*) | WHILE *exp stmt*
| DO *stmtlist* | *type varlist*

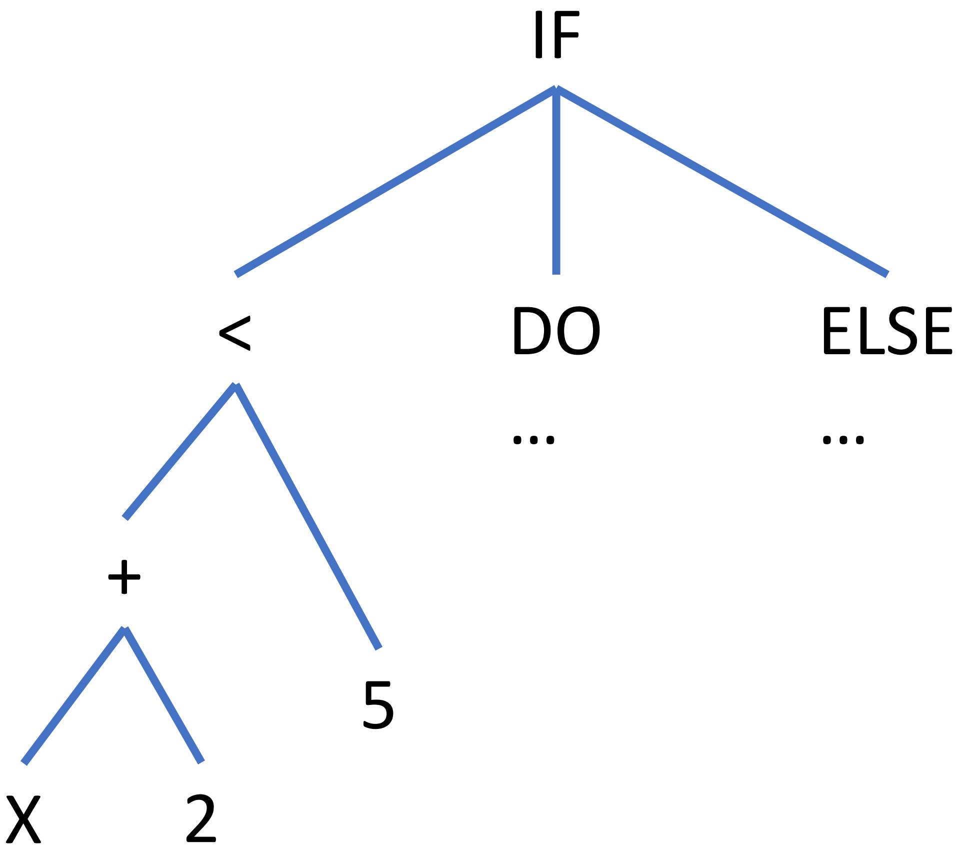
Not actually BNF, but you know what we mean

varlist ::= x | x *varlist*

stmtlist ::= *stmt* | *stmt stmtlist*

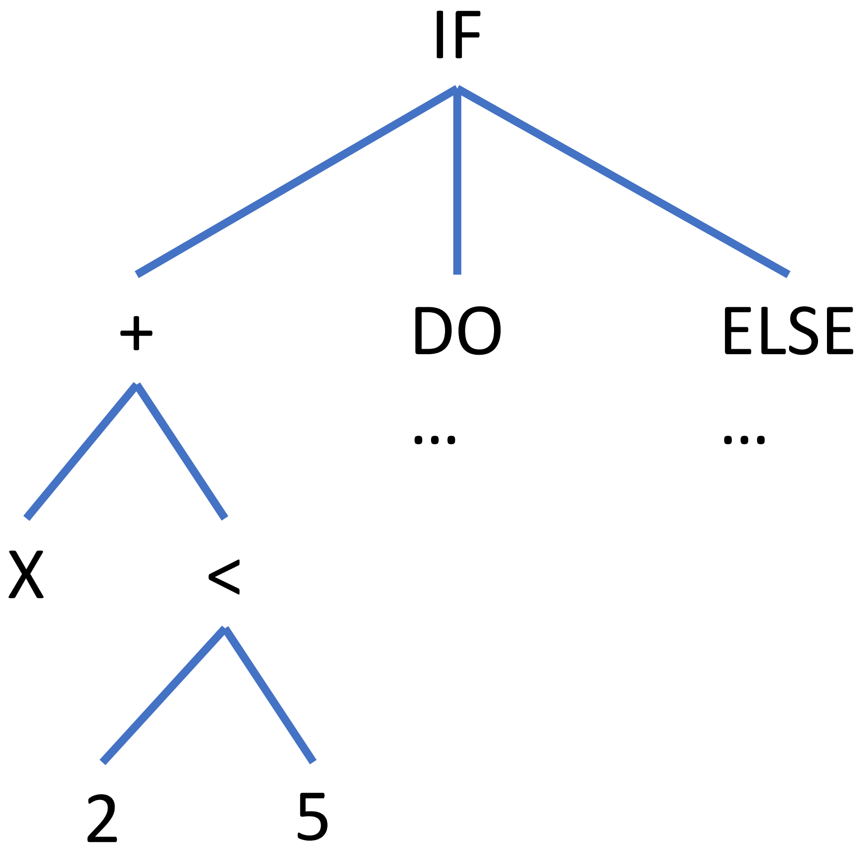
Abstract Syntax Trees (ASTs)

IF X + 2 < 5 DO … ELSE …



Abstract Syntax is not Concrete Syntax

IF X + 2 < 5 DO … ELSE …


IF
+
DO
ELSE
…
…
X
<
2
5