COP 3402 Systems Software

Lecture 4:
Compilers
Interpreters

Outline

1. Compiler and interpreters
2. Compilers
3. Interpreters
4. PL/0 lexical tokens

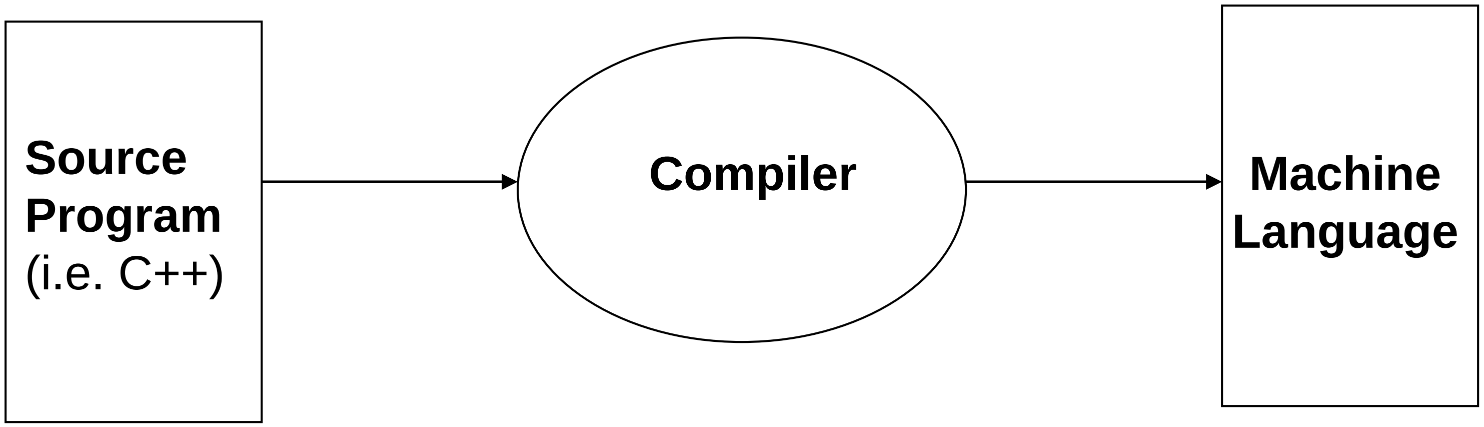
Compilers / Interpreters

- Programming languages are notations for describing computations (to programmers and computers).
- There are three general ways for performing these computations:

1. Compilation
2. Interpretation
3. Hybrid Implementation

Compilers

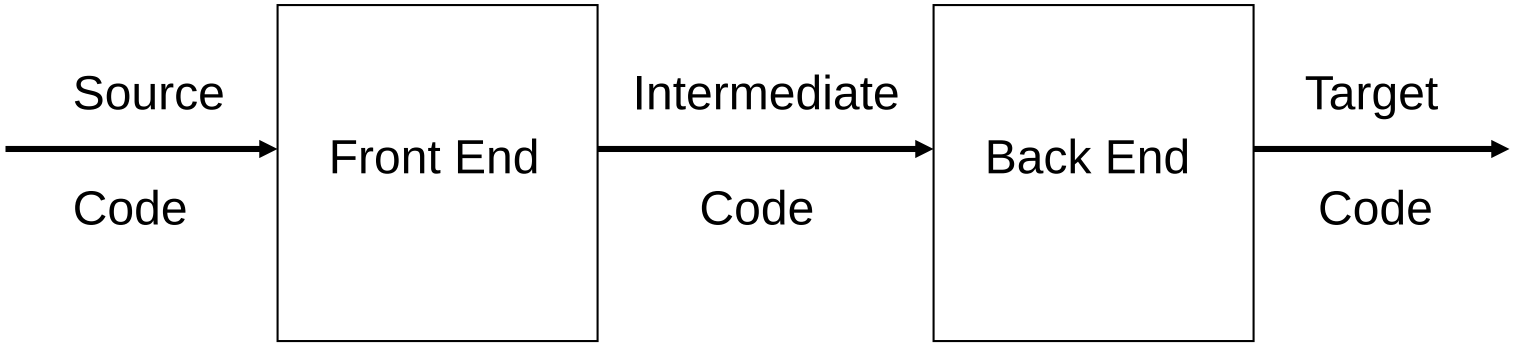
A compiler is a program that takes a high level language (such as C) as input, and translates it to a low-level representation (machine language).



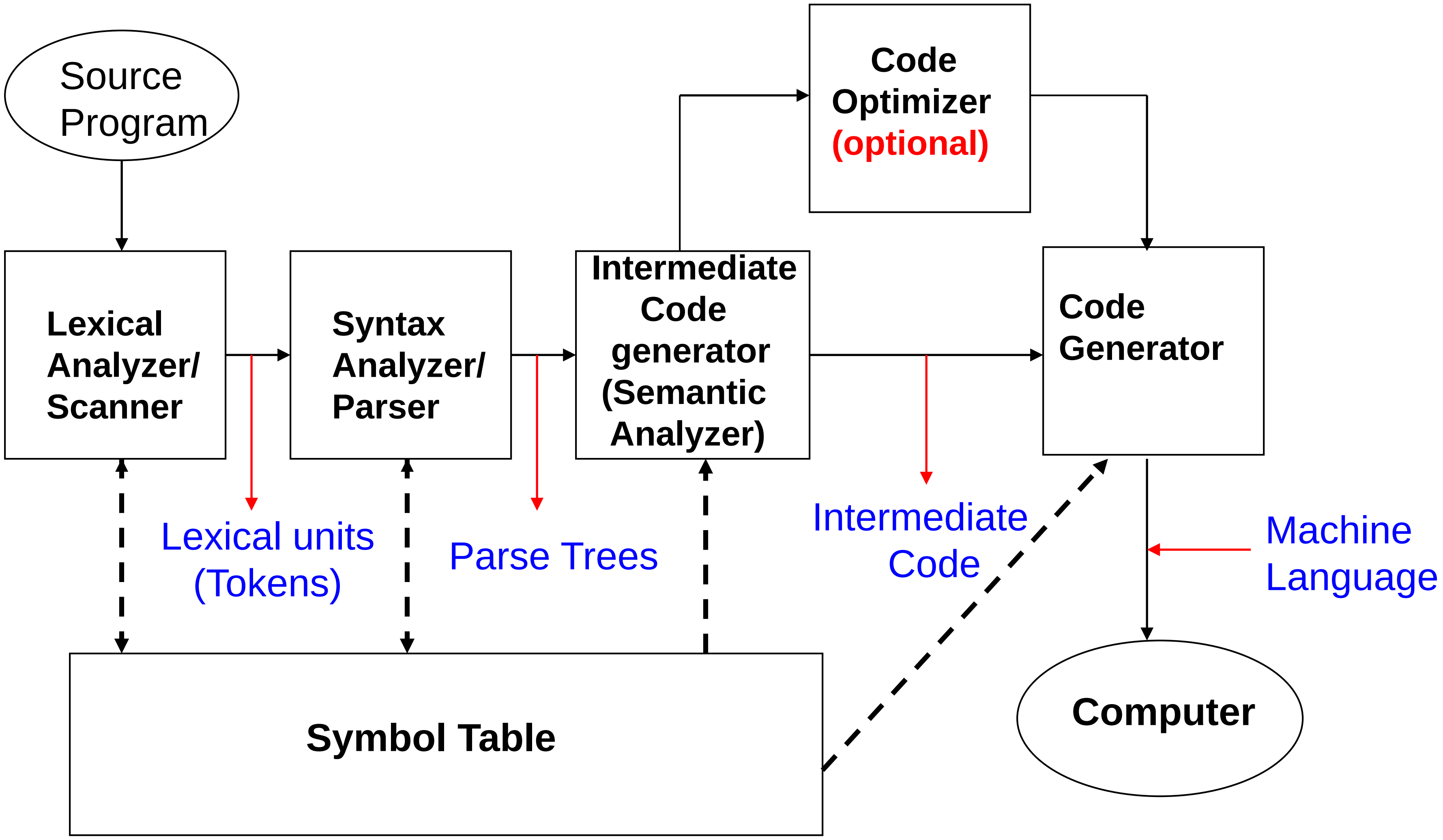
Compilers

The process of compilation takes place in several phases:

Front end	Lexical Analyzer/Scanner	→
	Syntactic Analyzer/Parser	→
	Semantic Analyzer	
Back end	Code generator	

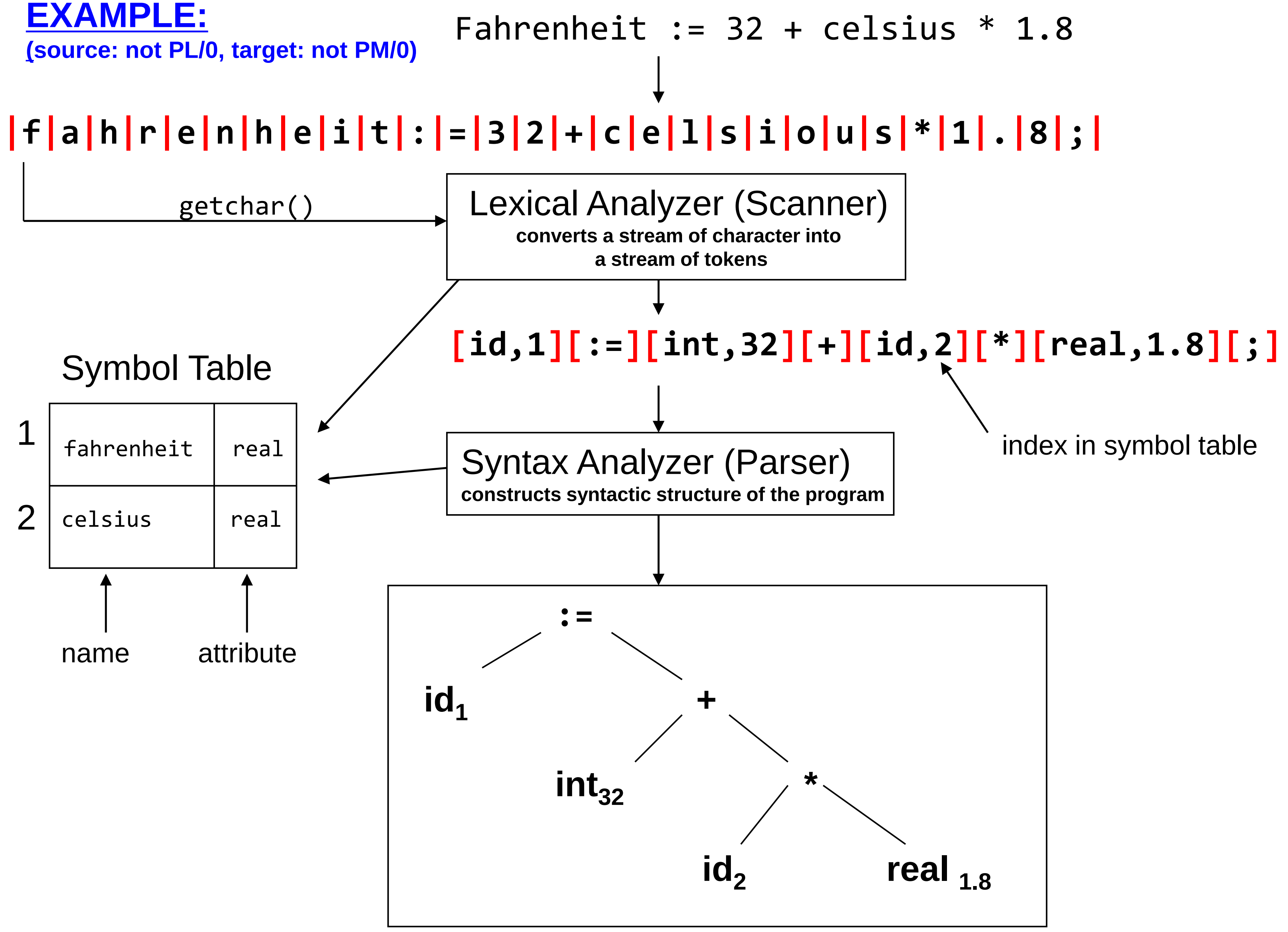


Compilers


Source Program
Code Optimizer (optional)
Lexical Analyzer/ Scanner
Syntax Analyzer/ Parser
Intermediate Code generator (Semantic Analyzer)
Code Generator
Lexical units (Tokens)
Parse Trees
Intermediate Code
Machine Language
Symbol Table
Computer

EXAMPLE:

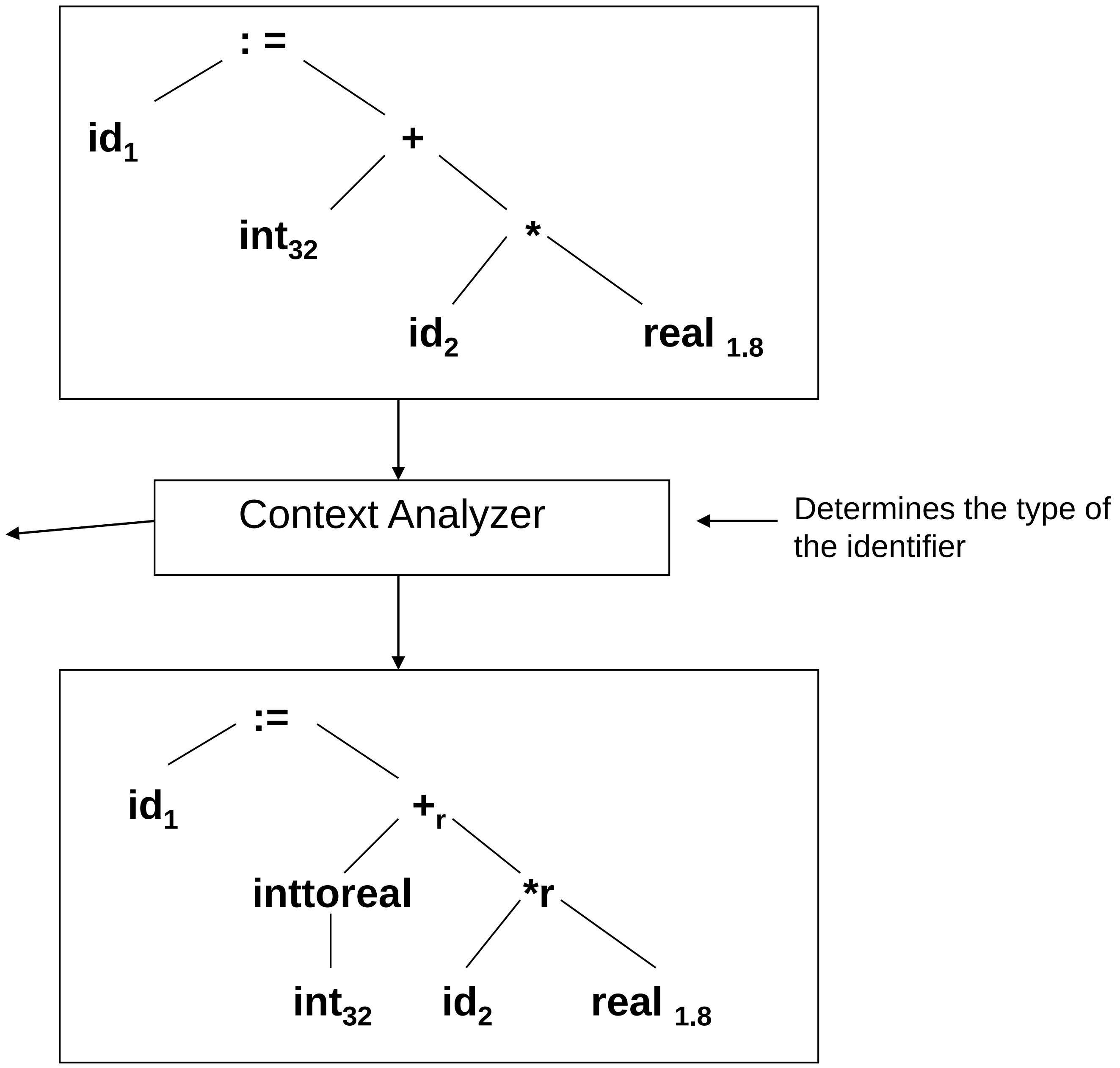
(source: not PL/0, target: not PM/0)

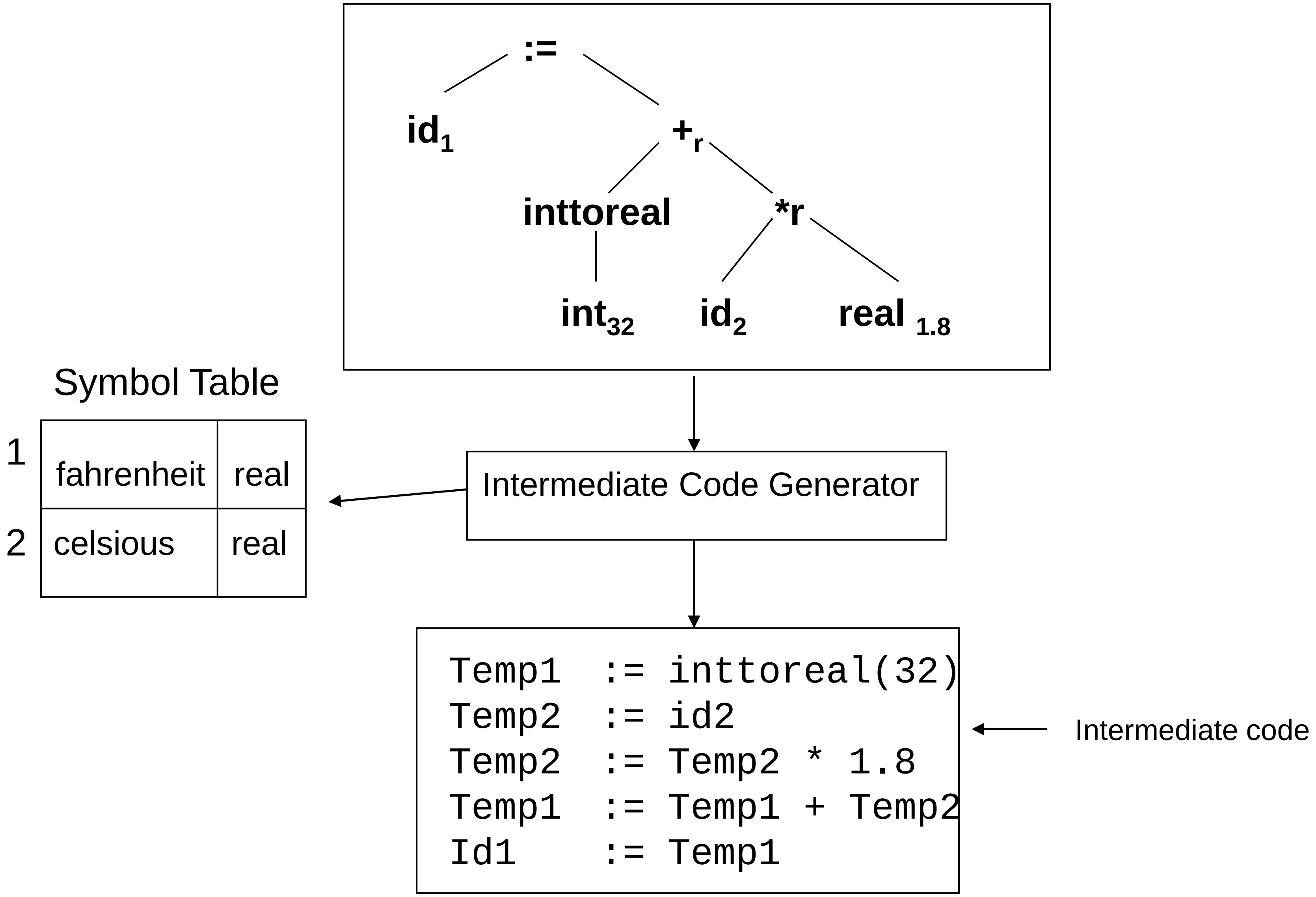


Fahrenheit := 32 + celsius * 1.8
|f|a|h|r|e|n|h|e|i|t|:|=|3|2|+|c|e|l|s|i|o|u|s|*|1|.|8|;|
getchar()
Lexical Analyzer (Scanner)
converts a stream of character into a stream of tokens
[id,1][:=][int,32][+][id,2][*][real,1.8][;]
index in symbol table
Symbol Table
1
fahrenheit
real
2
celsius
real
name
attribute
Syntax Analyzer (Parser)
constructs syntactic structure of the program
:=
id1
+
int32
*
id2
real 1.8

Symbol Table

1	fahrenheit	real
2	celsious	real





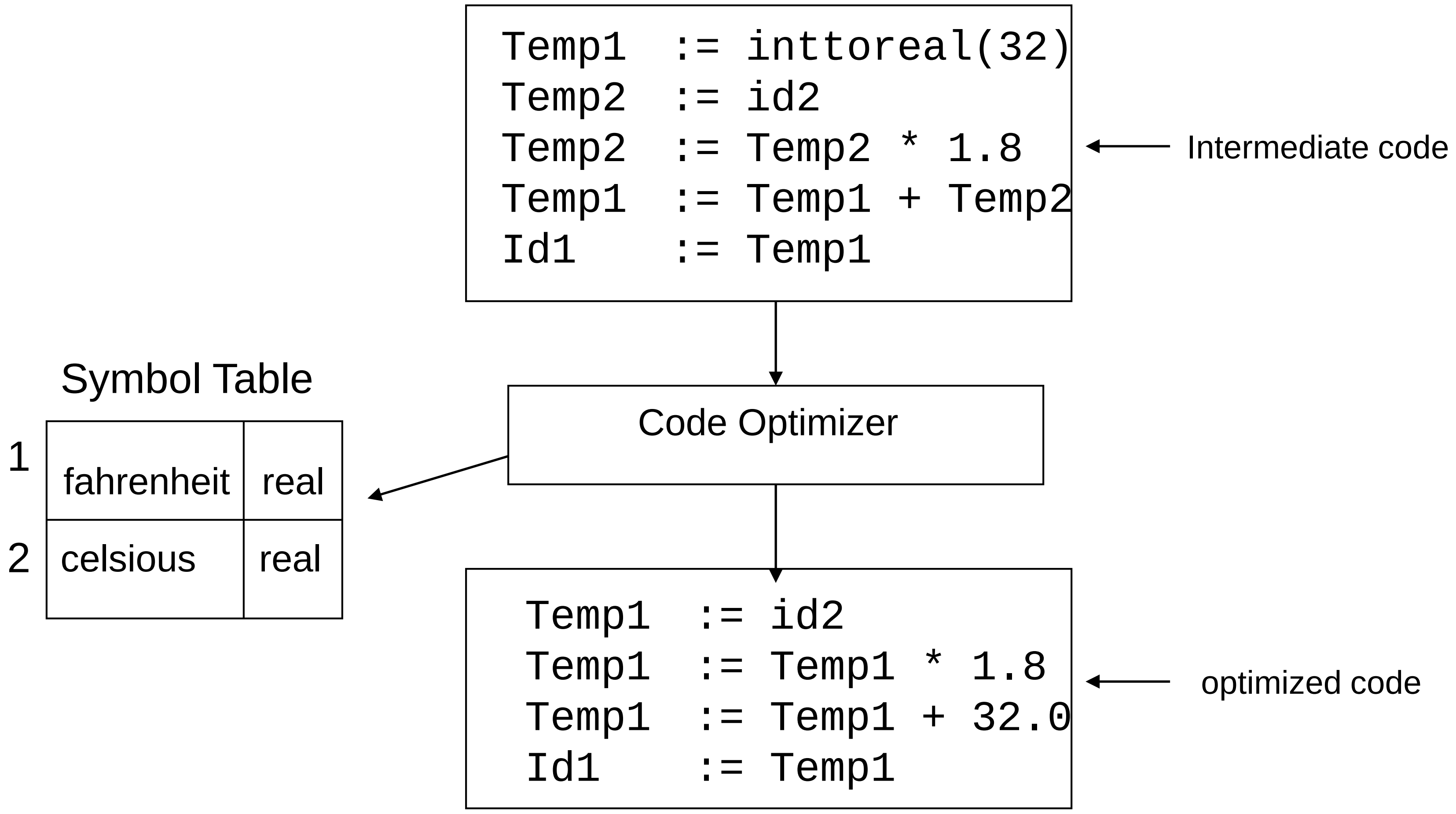
Symbol Table

1	fahrenheit	real
2	celsious	real

Intermediate Code Generator

```
Temp1  := inttoreal(32)
Temp2  := id2
Temp2  := Temp2 * 1.8
Temp1  := Temp1 + Temp2
Id1    := Temp1
```

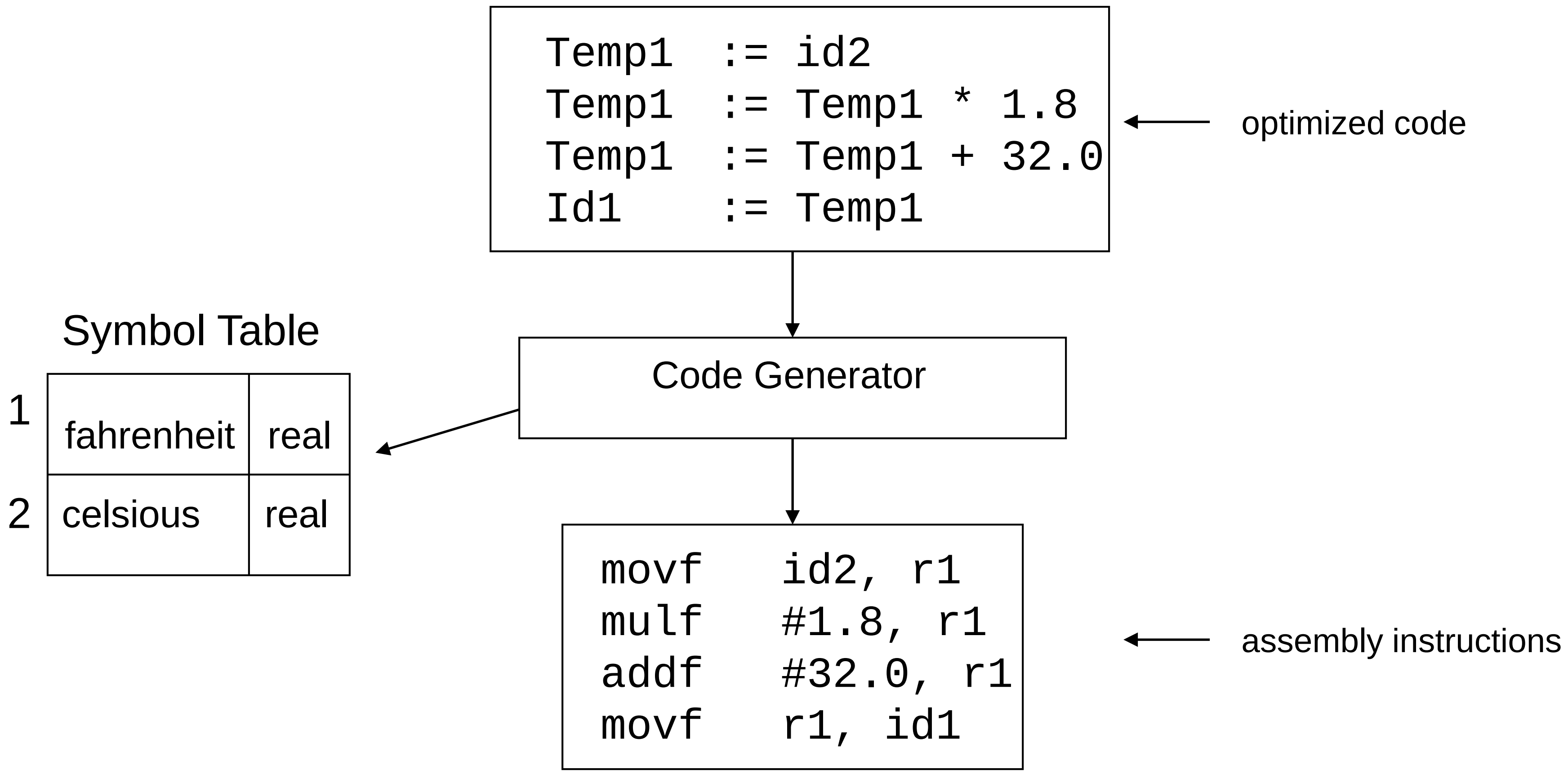
Intermediate code



```
Temp1  := inttoreal(32)
Temp2  := id2
Temp2  := Temp2 * 1.8
Temp1  := Temp1 + Temp2
Id1    := Temp1
```

1	fahrenheit	real
2	celsious	real

```
Temp1  := id2
Temp1  := Temp1 * 1.8
Temp1  := Temp1 + 32.0
Id1    := Temp1
```



```
Temp1  := id2
Temp1  := Temp1 * 1.8
Temp1  := Temp1 + 32.0
Id1    := Temp1
```

Symbol Table

1	fahrenheit	real
2	celsious	real

```
movf   id2, r1
mulf   #1.8, r1
addf   #32.0, r1
movf   r1, id1
```

Compilers

<u>Lexical Analyzer:</u>
transforms a stream of characters of the source program and produces **lexical tokens**; it discards white space and comments between the tokens

Lexical tokens of a program are:

- Identifiers
- Numbers
- Reserved words
- Arithmetic and logical operators
- …

<u>Syntax Analyzer:</u>
gets tokens from the lexical analyzer and uses them to construct a hierarchical structure called **parse tree**.

Parse trees represent the syntactic structure of the program.

Compilers

Intermediate Code Generator:

produces a program in a different language representation:

- Assembly language
- Language similar to assembly language
- Language higher than assembly language

Note: Semantic Analysis is an integral part of the intermediate code generator

Optimization:

makes programs smaller or faster or both.

most optimization is done at the level of intermediate code.
(for example, tree reduction, vectorization).

See The **LLVM** Compiler Infrastructure http://llvm.org/

Compilers

Code Generator:
translates the optimized intermediate code into machine language.

Symbol Table:
serves as a database for the compilation process.

contains type and attribute information of each user-defined name in the source program.

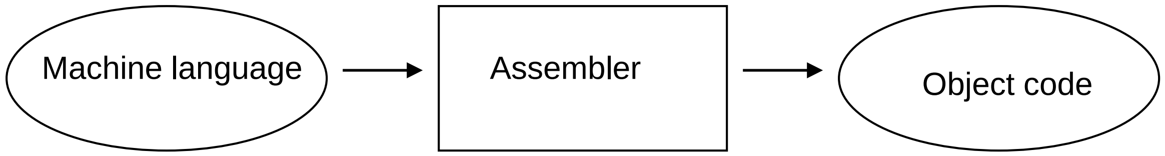
Symbol Table

Index	name	type	attributes
1	fahrenheit	real	
2	celsious	real	

Compilers

<u>Machine Language</u>
A program in machine language (assembly language) needs, in general, to be translated to object code for execution

Assembler is a program that translates machine language into object code



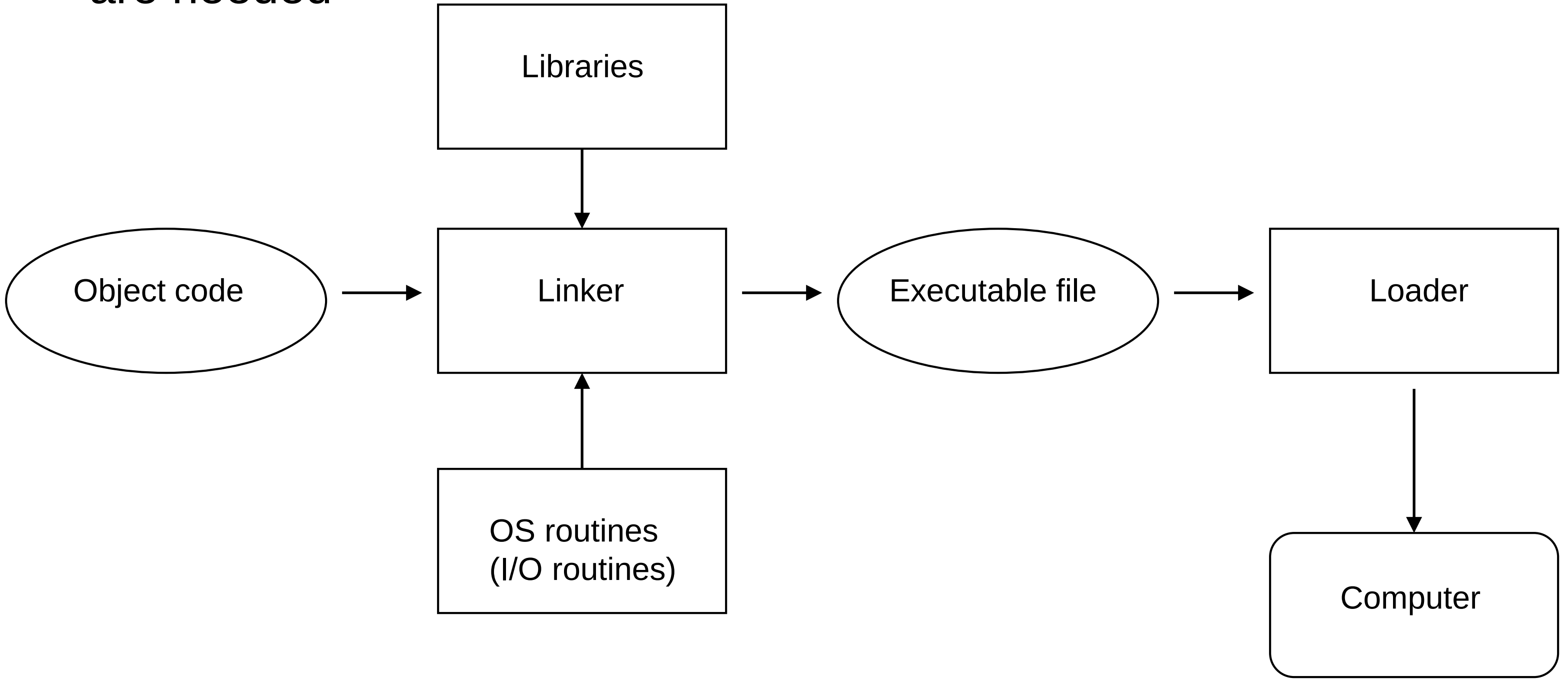
Compilers

Machine Language

To run a program in object code, in general,

- some other code (libraries) and
- some routines from the operating system (i.e. input/output routines)

are needed



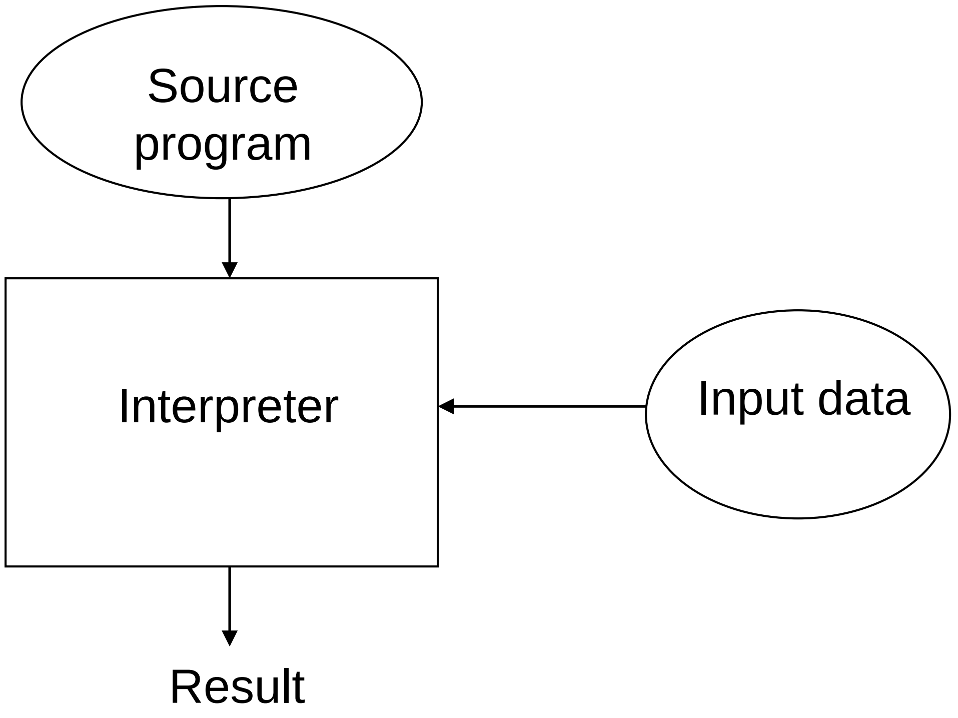
Interpreters

Programs are interpreted (executed) by another program called the interpreter.

> **Advantages: Easy implementation of many source-level debugging operations because all run-time errors refer to source-level units.**
>
> **Disadvantages: 10 to 100 times slower because statements are interpreted each time the statement is executed.**

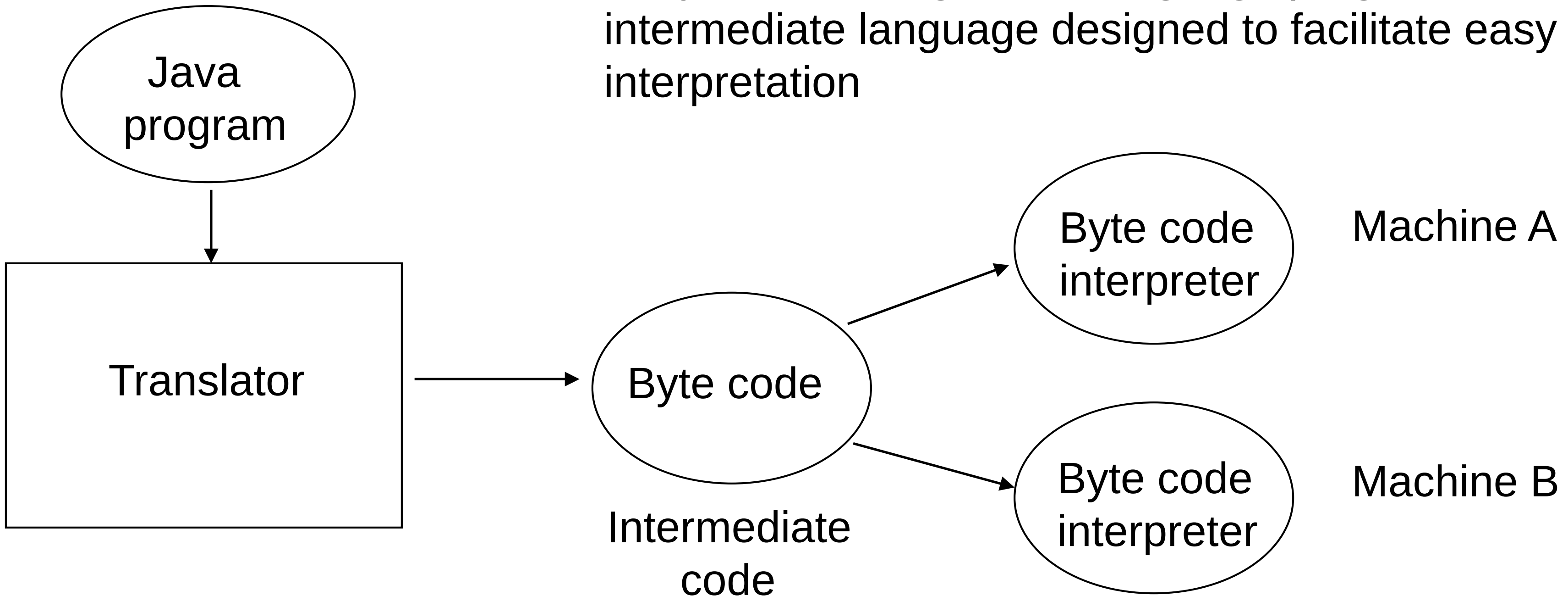
Background:
Early sixties → APL, SNOBOL, Lisp.
By the 80s → rarely used.
Recent years → Significant comeback (some Web scripting languages such as JavaScript and PHP)

Interpreters



Hybrid Implementation Systems

They translate high-level language programs to an intermediate language designed to facilitate easy interpretation



Interpreters

Just-In-Time (JIT) implementation

Programs are translated to an intermediate language.

During execution, the intermediate language methods
are compiled into machine code when they are called.

The machine code version is kept for subsequent calls.

.NET and Java programs are implemented as JIT systems.