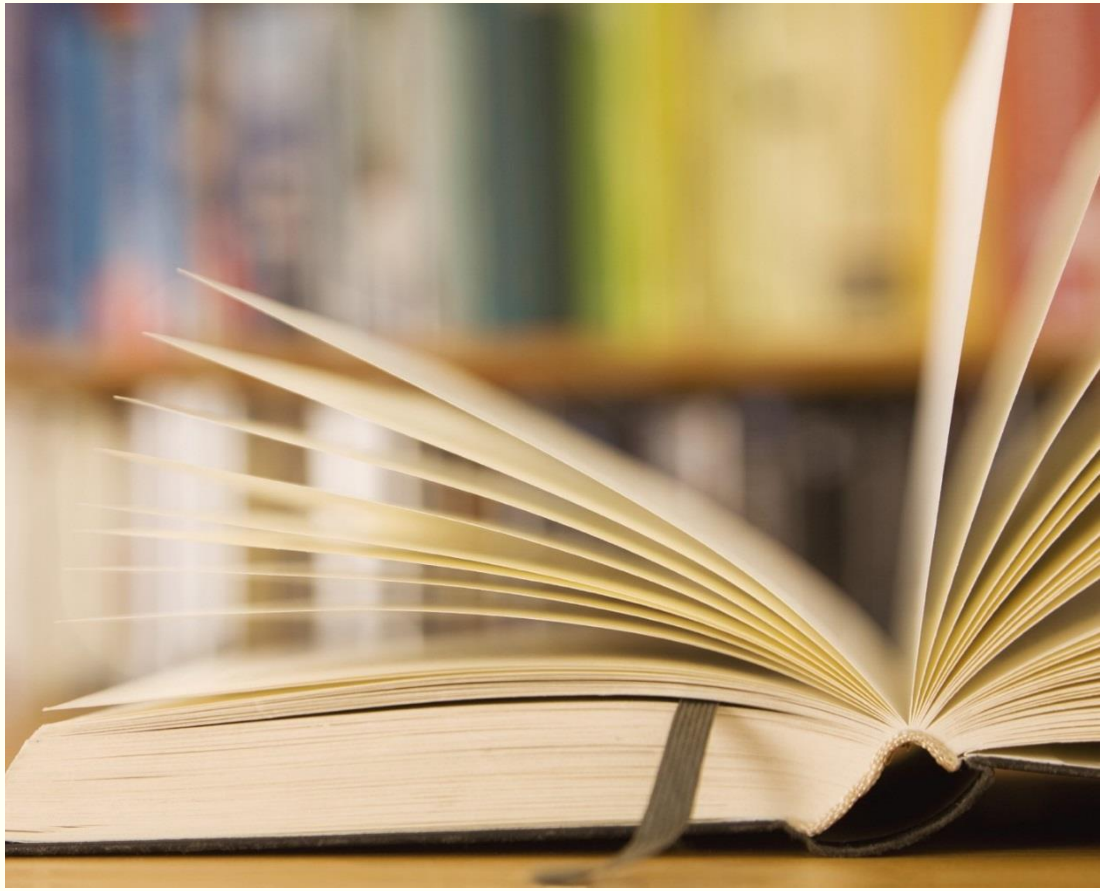
Compiler Design

Zakia Zinat Choudhury
Lecturer
Department of Computer Science & Engineering
University of Rajshahi



Introduction

Compiler

Compiler is a Computer software that translates (compiles) source code written in a high-level language (e.g., C++) into a set of machine-language instructions that can be understood by a digital computer's CPU.

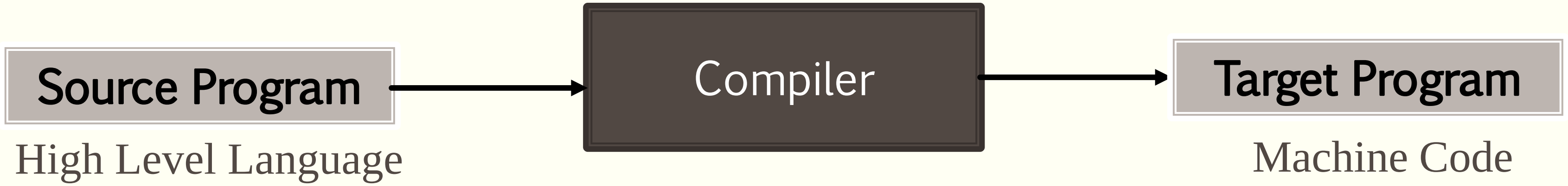


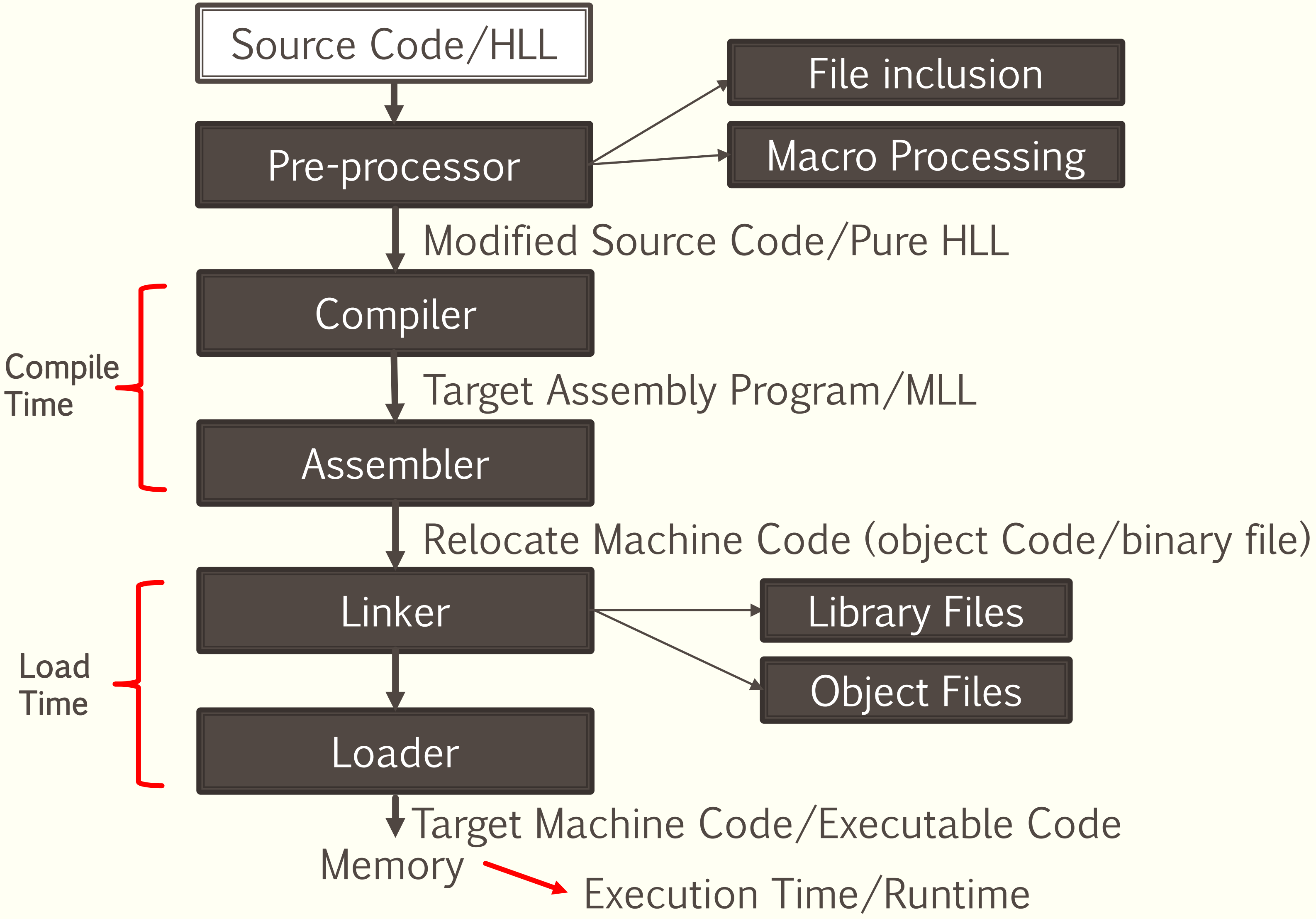
Figure: A Compiler

As an important part of this translation process, the compiler reports to its user the presence of errors in the source program.

Why to Learn Compiler design?

Computers are a balanced mix of software and hardware. Hardware is just a piece of mechanical device and its functions are being controlled by a compatible software. Hardware understands instructions in the form of electronic charge, which is the counterpart of binary language in software programming. Binary language has only two alphabets, 0 and 1. To instruct, the hardware codes must be written in binary format, which is simply a series of 1s and 0s. It would be a difficult and cumbersome task for computer programmers to write such codes, which is why we have compilers to write such codes.

Language Processing



Source Code/HLL

Pre-processor

File inclusion

Macro Processing

Modified Source Code/Pure HLL

Compile Time

Compiler

Target Assembly Program/MLL

Assembler

Relocate Machine Code (object Code/binary file)

Load Time

Linker

Library Files

Object Files

Loader

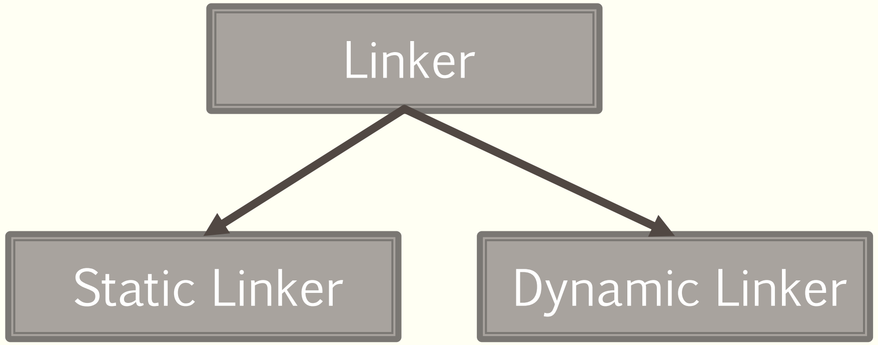
Target Machine Code/Executable Code

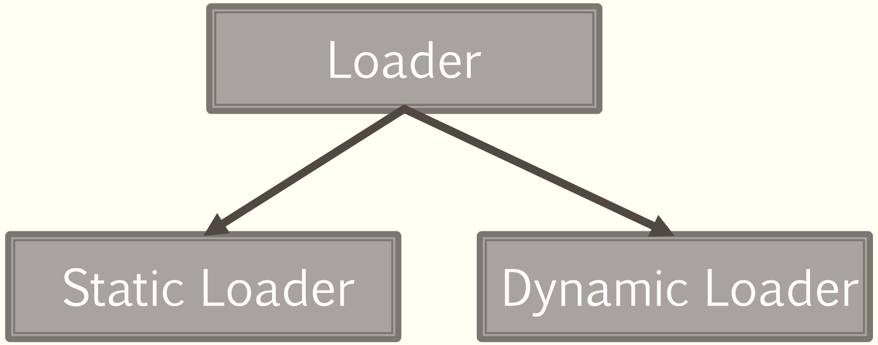
Memory

Execution Time/Runtime

```
#include<stdio.h>
#define value 50
int main(){
     int a=10, b=20, sum;
     sum=a+b;
     if(sum<value)
          printf("%d/n",sum);
   }
```

Linker & Loader





Assembler

Assembler

The Assembler is used to translate the program written in Assembly language into machine code. The source program is a input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.

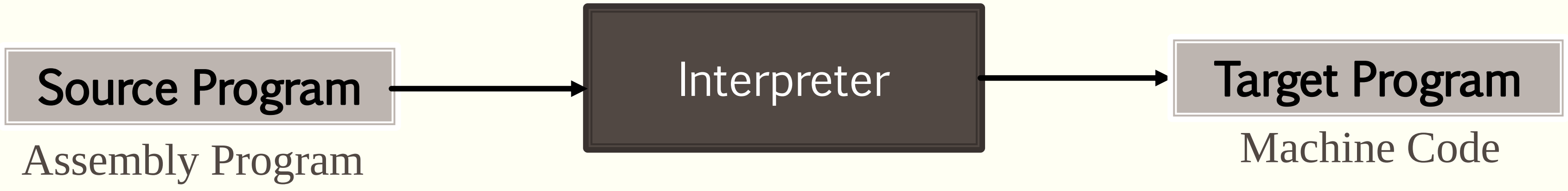


Figure: Assembler

Interpreter

Interpreter

The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter.

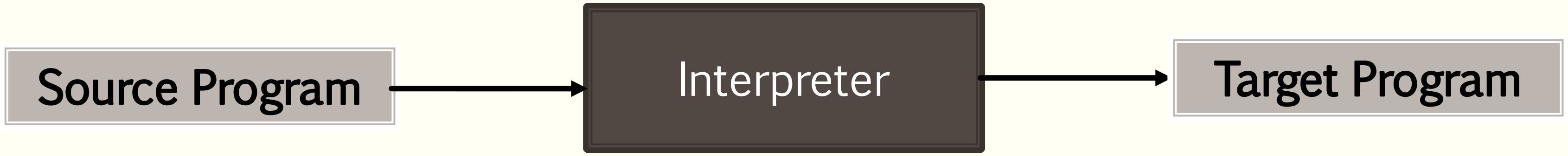
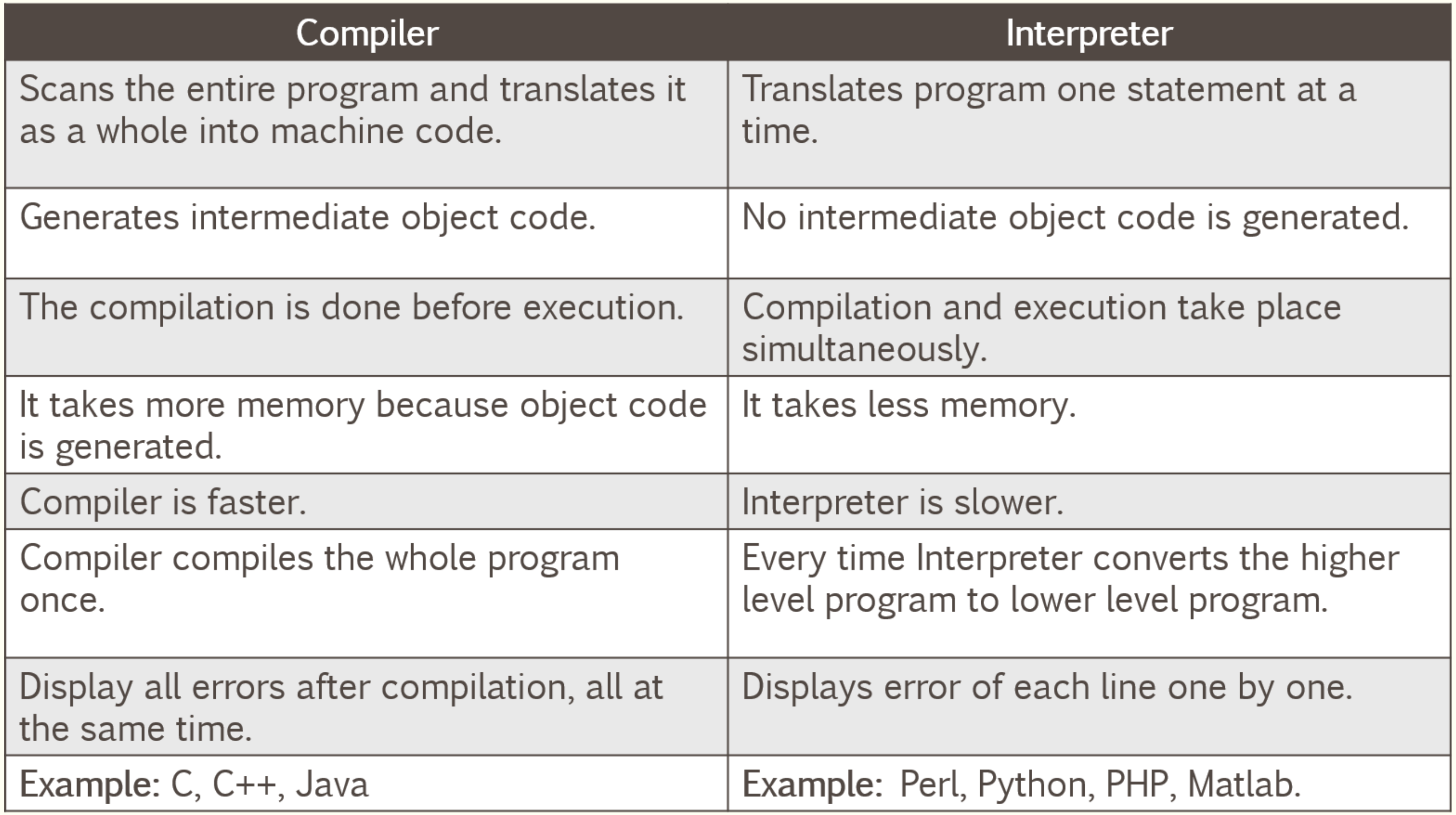


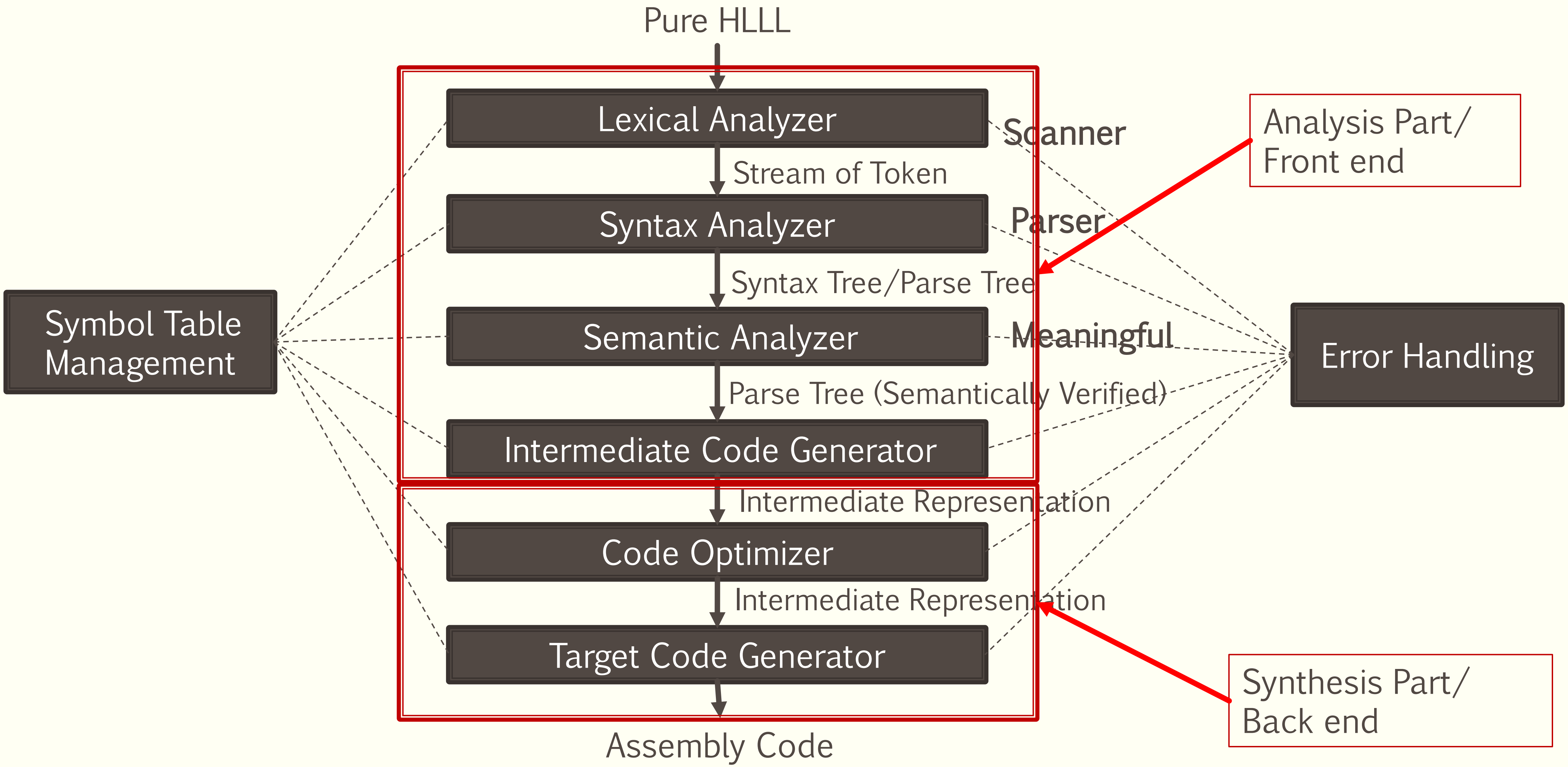
Figure: Interpreter

Differences Between Compiler & Interpreter



Compiler	Interpreter
Scans the entire program and translates it as a whole into machine code.	Translates program one statement at a time.
Generates intermediate object code.	No intermediate object code is generated.
The compilation is done before execution.	Compilation and execution take place simultaneously.
It takes more memory because object code is generated.	It takes less memory.
Compiler is faster.	Interpreter is slower.
Compiler compiles the whole program once.	Every time Interpreter converts the higher level program to lower level program.
Display all errors after compilation, all at the same time.	Displays error of each line one by one.
Example: C, C++, Java	**Example:** Perl, Python, PHP, Matlab.

The Structure of a Compiler



Analysis Part & Synthesis Part

- The analysis part breaks up the source program into constituent pieces and imposes a grammatical structure on them.

  The analysis part is often called the front end of the compiler.

- The synthesis part constructs the desired target program from the intermediate representation and the information in the symbol table.

  The synthesis part is the back end of the compiler.

Symbol Table

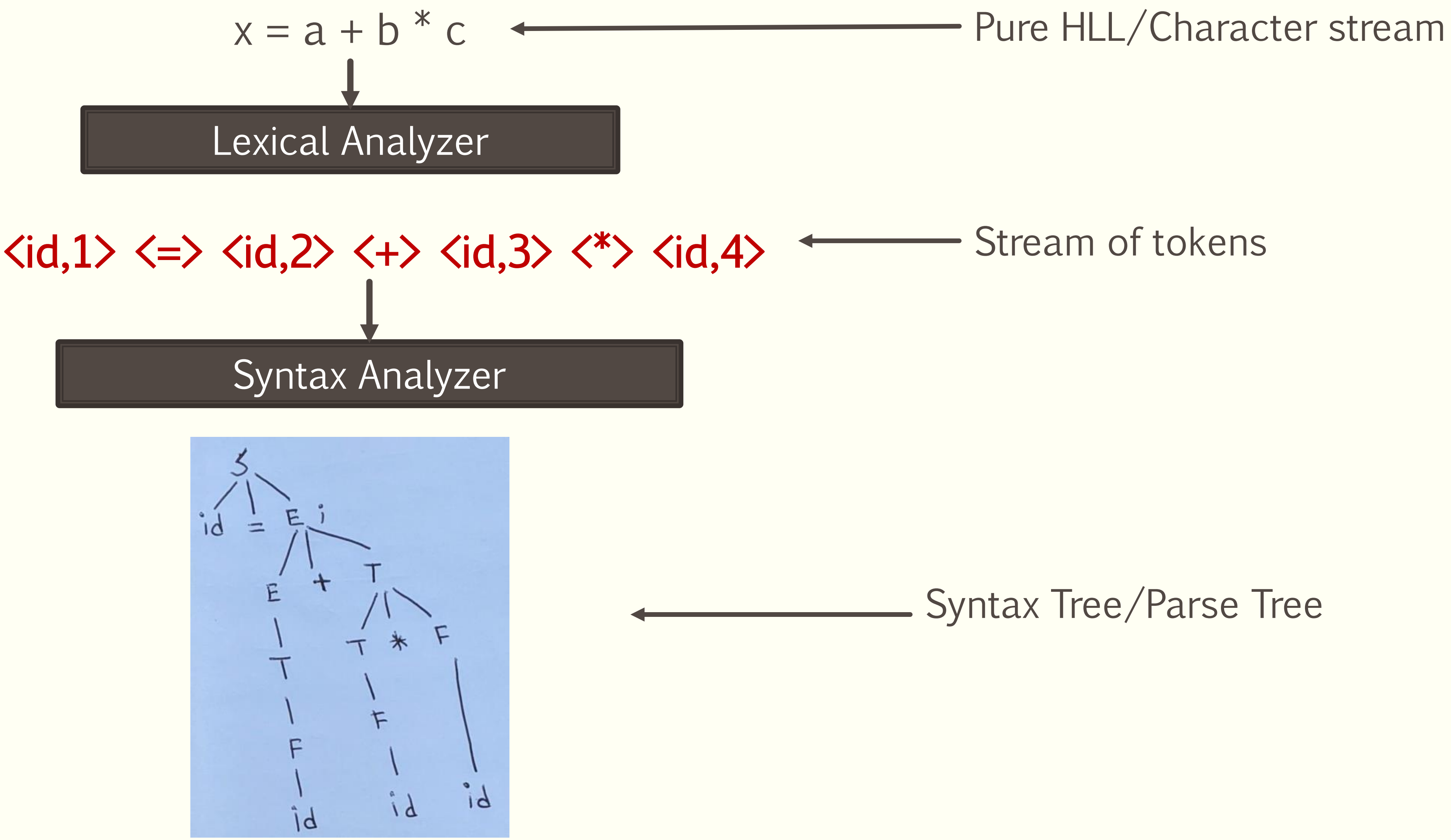
- The analysis part also collects information about the source program and stores it in a data structure called a **symbol table**, which is passed along with the intermediate representation to the synthesis part.
- The symbol table is a data structure containing a record for each variable name, with fields for the attributes of the name.
- The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.

Compiler Pass

- A Compiler pass refers to the traversal of a compiler through the entire program.
- Compiler pass are two types: Single Pass Compiler, and Two Pass Compiler *or* Multi Pass Compiler. These are explained as following below.

- A combined or group of all the phases in a single module of compiler design is known as a single-pass compiler.
- A two-pass or multi-pass compiler is a type of compiler that processes the source code or abstract syntax tree of a program multiple times. In multi-pass compiler the phases are divided into two parts.

Example of the Structure of a Compiler



x = a + b * c

Pure HLL/Character stream

Lexical Analyzer

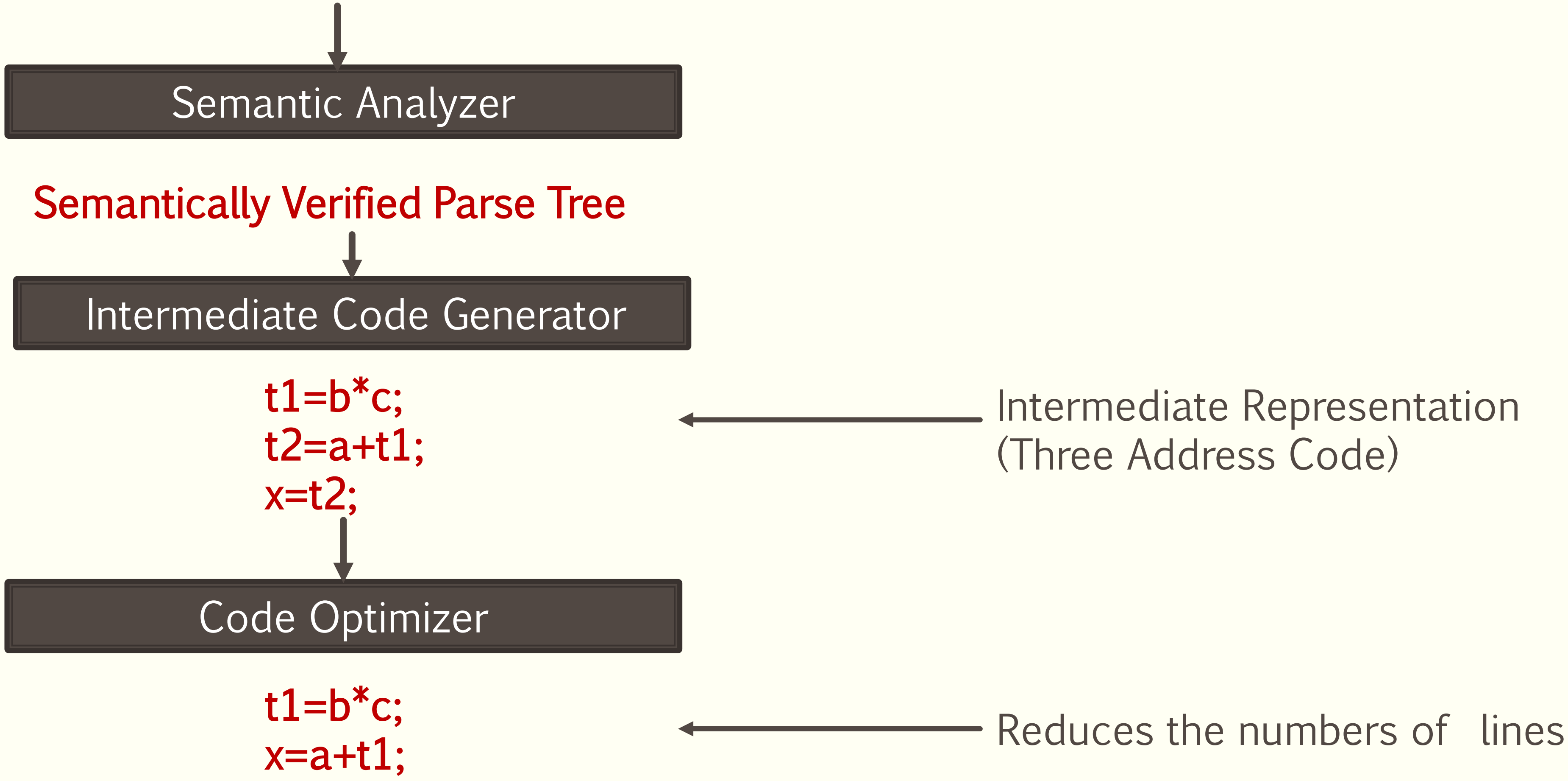
<id,1> <=> <id,2> <+> <id,3> <*> <id,4>

Stream of tokens

Syntax Analyzer

Syntax Tree/Parse Tree

Example of the Structure of a Compiler



Semantic Analyzer

Semantically Verified Parse Tree

Intermediate Code Generator

```
t1=b*c;
t2=a+t1;
x=t2;
```

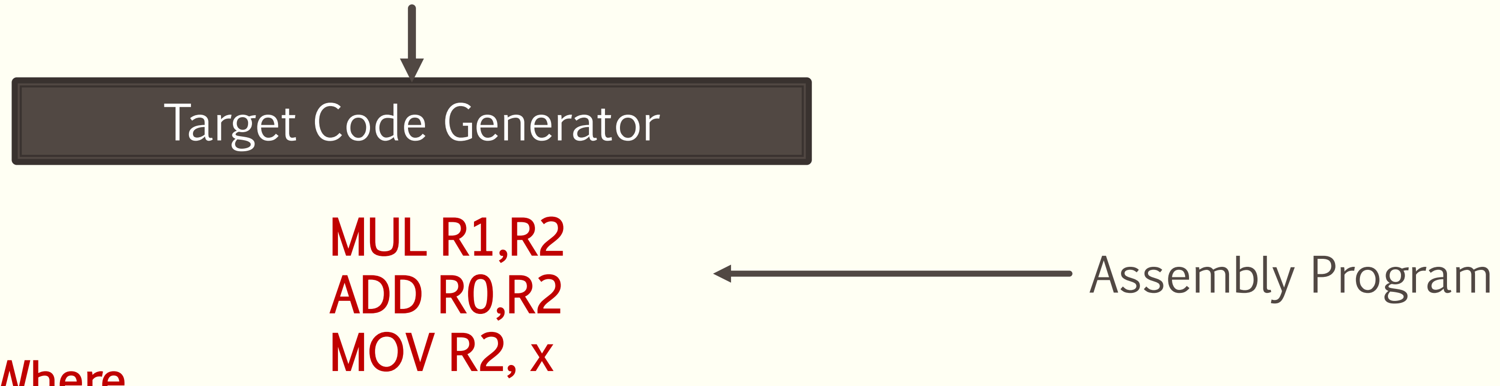
Intermediate Representation
(Three Address Code)

Code Optimizer

```
t1=b*c;
x=a+t1;
```

Reduces the numbers of lines

Example of the Structure of a Compiler



Target Code Generator

MUL R1,R2
ADD R0,R2
MOV R2, x

Assembly Program

Where,
R0=a
R1=b
R2=c

Thank You