



CS202 – System Software

Dr. Manish Khare

Lecture 4



Course Content

- ***Introduction to System Software:*** Definition, System software, Machine structure, Components of a programming system, Assemblers, linker, loader, compiler, Macros, text Editor, Debugger, Program development Flow, Introduction to Operating System, Language Processor, Assembly Language, Introduction to CISC and RISC machine architecture
- **Assembler:** Basic Assembler Functions, Machine Dependent Features, Machine Independent Features, One pass and Multi pass Assembler.

Course Content

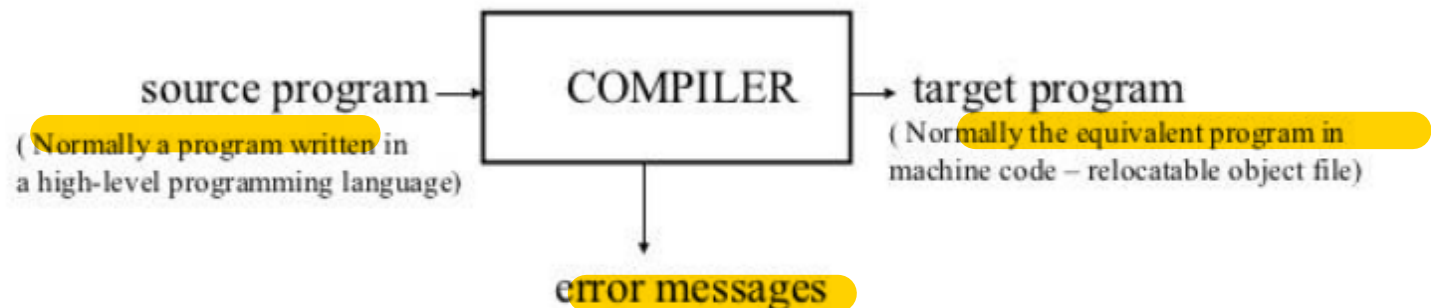
- **Linkers, Loaders, Macros and Macro Processors:** Basic Loader Function, Loader Design Options, Relocation and Linking Concepts, Design of a Linker, Case study for Linker and Loader, Macro definition, Macro expansion, Basic Macro Processor Functions and Features, Macro Processor Design Options, Implementation example for Macro Processor
- **Compilers:** Aspects of Compilation, Compiler Features, Memory Allocation, Grammar, Parsing Techniques, Compiler Design Options, Intermediate Code Generation and Optimization Techniques
- **Scanning and Parsing:** Programming language grammars, Scanning, parsing, language processor development tools
- **Interpreters:** Overview of interpretation, benefits of interpretation
- **Software Tools:** Text Editors, Debuggers, User Interfaces.

Compilers

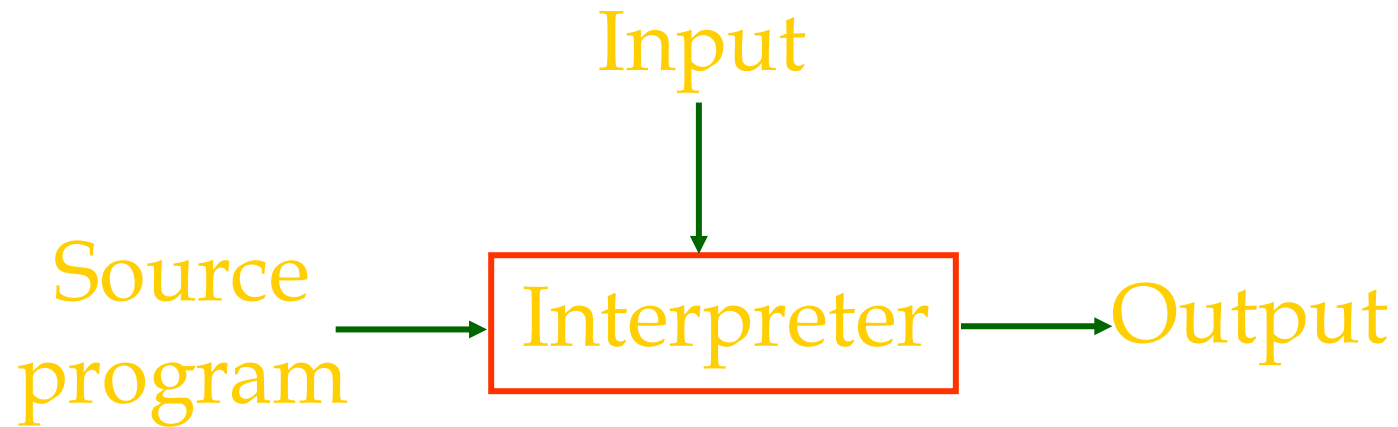


Compiler

- A compiler is a program that reads a program written in one language – the source language and translates its into an equivalent program in another language – the target language. As an important part of this translation process, the compiler reports to its user the presence of error in the source program, if any.



Interpreters



Advantage and Disadvantage of Compiler

➤ Advantage

- The translation is done one only as a separate process. The program that is run is already translated into machine code so is much faster in execution.

➤ Disadvantage

- No one can change the program without going back to the original code, editing that, and recompiling.


Difference between Compiler and Interpreter

- Compiler converts the higher level instruction into machine language, while an interpreter converts the high level instruction into an intermediate form.
- List of errors is created by the compiler after the compilation process while an interpreter stops translating after the first error.
- Compiler takes large amount of time to analyze the source code but the overall execution time is comparatively faster, whereas Interpreter takes less amount of time to analyze the source code but the overall execution time is slower.

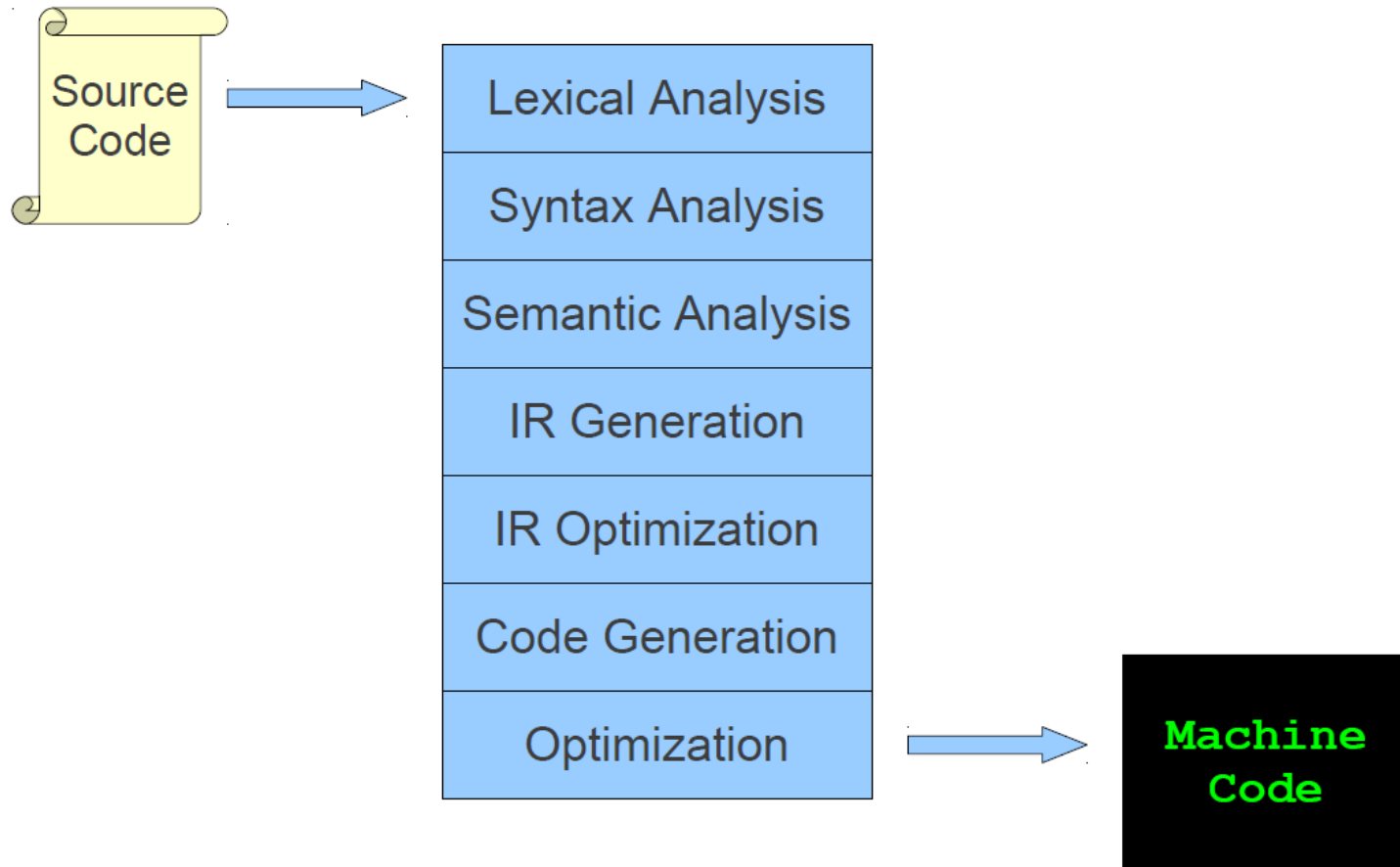
Analysis of Source Program

- In compiling any program, analysis consists of three phases
 - Linear analysis
 - Hierarchical analysis
 - Semantic analysis


- In Linear analysis, the stream of characters, making up the source program is read from left to right and grouped into tokens that are sequences of characters having a collective meaning.

- 
- In Hierarchical analysis, character or tokens, are grouped hierarchically into nested collection with collective meaning.
 - In semantic analysis, certain checks are performed to ensure that the components of a programs fit together meaningfully.

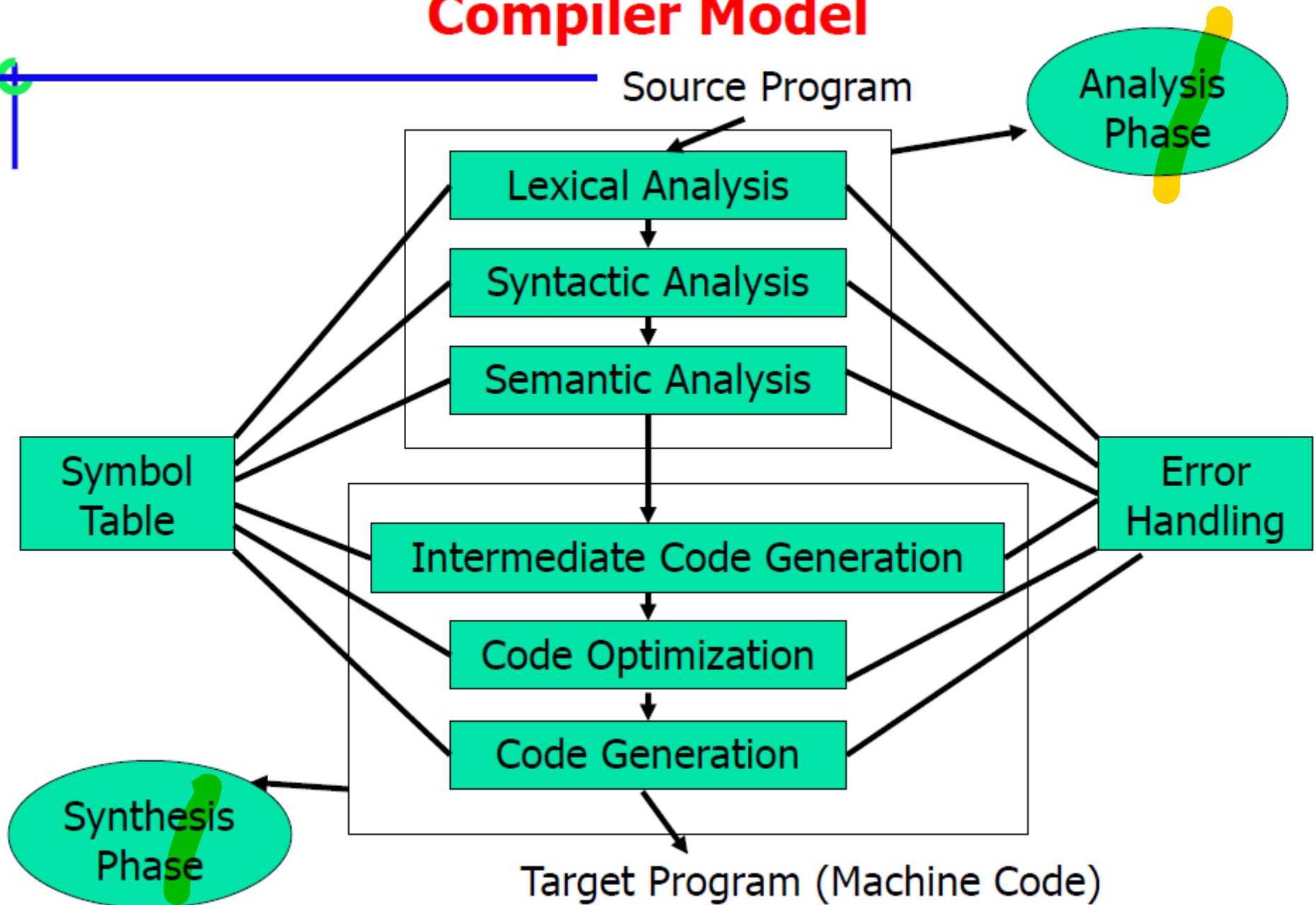
The Structure of a Modern Compiler




Compiler Model

- 
- The compiler must perform in two major parts
 - Analysis of source program
 - Synthesis of its corresponding object program.

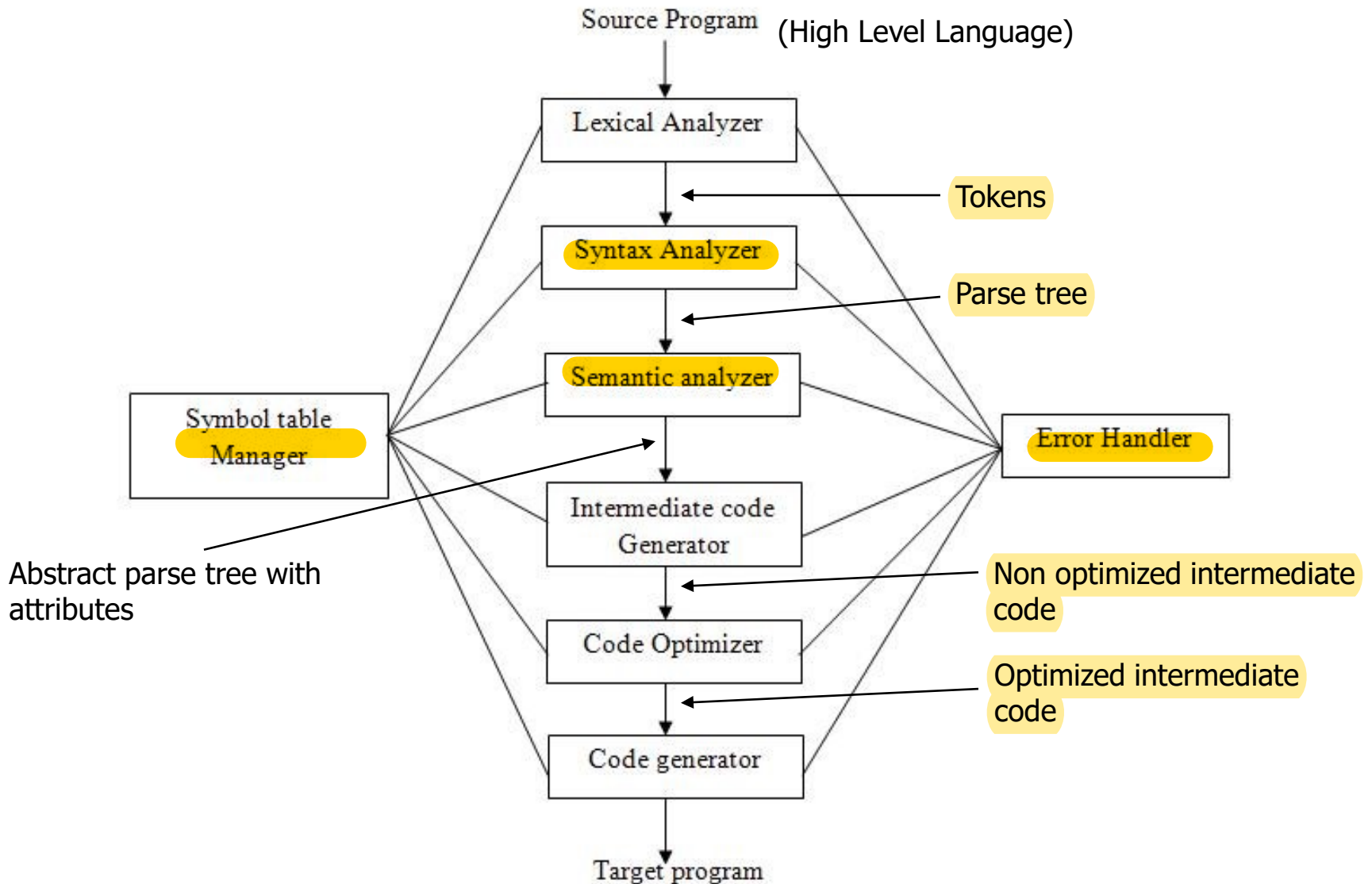
Compiler Model



The Phases of a Compiler

- 
- Conceptually, a compiler operates in phases, each of which transform the source program for one representation to another.

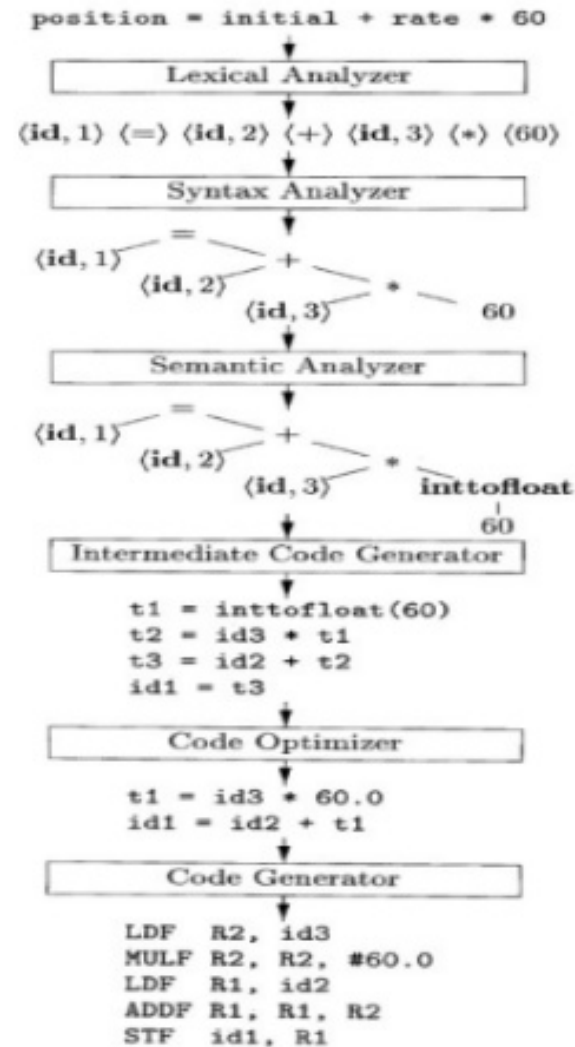
The Phases of a compiler



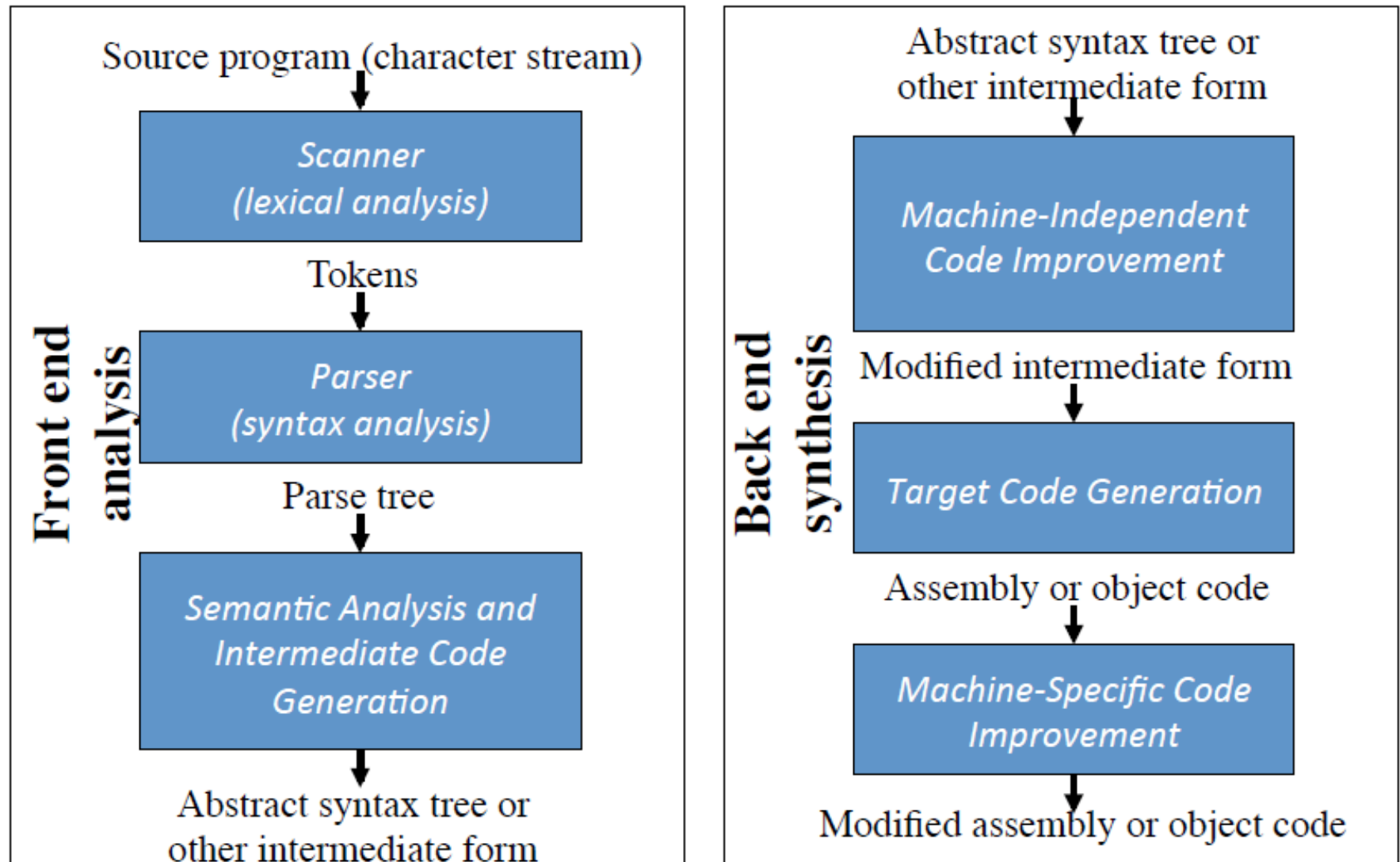
The Phases of a compiler

1	position	...
2	initial	...
3	rate	...

SYMBOL TABLE



Front end and Back end in Compiler



Front end and Back end in Compiler

➤ Front end

- The front end consists of those phases or parts of phases, that depend primarily on the source language and largely independent of the target language.
- These normally include lexical and syntactic analysis, the creation of the symbol table, semantic analysis and the generation of the intermediate code.
- The front end also includes the error handling that goes along with each of these phases.

Front end and Back end in Compiler

➤ Back end

- The back end includes those portion of the compiler that depend on the target machine and generally these portion do not depend on the source language, just the intermediate language.
- In the back end, we find aspects of the code optimization phase, and we find code generation, along with the necessary error handling, and symbol table operation.

Front end and Back end in Compiler

	Front end	Back end
Tasks	<ul style="list-style-type: none">• Find validity of a source statement.• Analyze the source statement lexically, syntactically, and semantically.• Construct suitable representation of source statement	<ul style="list-style-type: none">• Memory allocation.• Code generation
Phases of compiler	<ul style="list-style-type: none">• Lexical analysis• Syntax analysis• Semantic analysis• Generation of intermediate code	<ul style="list-style-type: none">• Code generation• Code optimization
	<ul style="list-style-type: none">• Error handling• Symbol table management	
Output	The intermediate representation produced by front end has two components (i). Table of information (ii). Intermediate Code	The output comprises of relocatable object code ready for the loader, to load and run the program

Phases and Passes of Compiler

- Each individual unique step in compilation process is called a phase such as lexical analysis, syntax analysis and so on.
- Different phases may be combined into one or more than one group, this group of phases makes the concept of passes.
- If all the phases are combined into one group then this is called as one pass compiler otherwise more than one pass constitute the multipass compiler

Difference between one-pass and multipass compiler

Single-pass (One pass)

In a single-pass compiler all the phases of compiler design are grouped in one pass.

A one-pass compiler is a compiler that passes through the source code of each compilation unit only once.

A one pass compiler does not “look back” at code previously processed.

Multi-pass

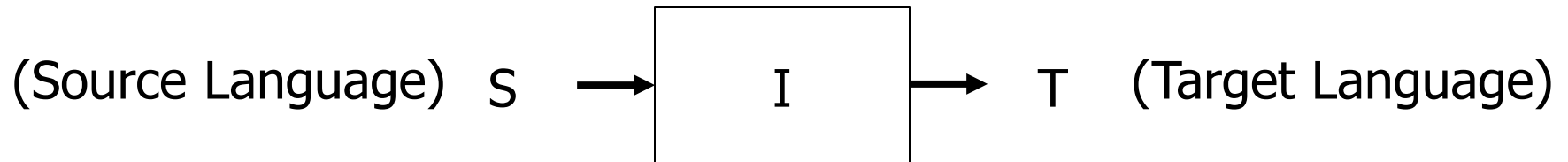
In a multi-pass compiler the different phases of a compiler design are grouped into multiple phases.

A multi-pass compiler is a type of compiler that processes the source code of a program several times.

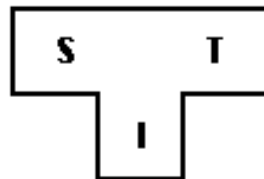
Each pass takes the result of the previous pass as the input, and creates an intermediate output

Bootstrapping

- Bootstrapping is an important concept in building new compiler.
- For constructing any new compiler, we require three language.
 - Source Language (S) – which actually compiles
 - Target Language (T) – which generate code for target language
 - Implementation Language (I) – the language in which it is written



- This can be shown in the form of T-diagram



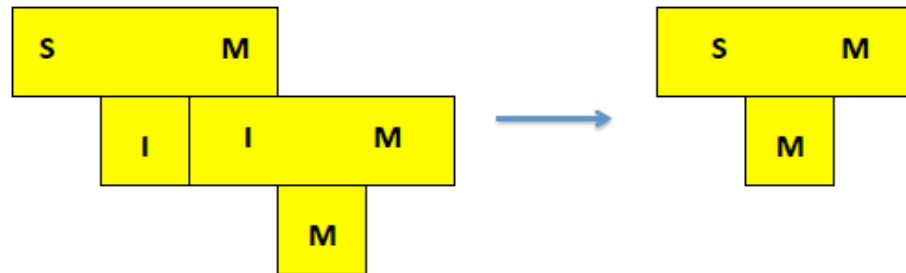
- represents a compiler for Source S , Target T , implemented in I

Cross Compiler

- A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.
- For example, a compiler that runs on a Windows 7 PC but generates code that runs on Android smartphone is a cross compiler.

Composing Compilers

- Compiling a compiler we get a new one: the result is described by composing T-diagrams.
- A shape of the basic transformation, in the most general case, is the following



- Note: by writing this transformation, we implicitly assume that we can execute programs written in M.
- A compiler of S to M can be written in any language having a host compiler for M.

Compilers: Portability Criteria

➤ Portability

- Retargetability
- Rehostability

- A retargetable compiler is one that can be modified easily to generate code for a new target language.
- A rehostable compiler is one that can be moved easily to run on a new machine.
- A portable compiler may not be as efficient as a compiler designed for a specific machine, because we cannot make any specific assumption about the target machine.

Using Bootstrapping to port

- We have a **host compiler/interpreter** of L for M.
- Write a compiler of L to N in language L itself.

Example:
L Pascal
M P-code

