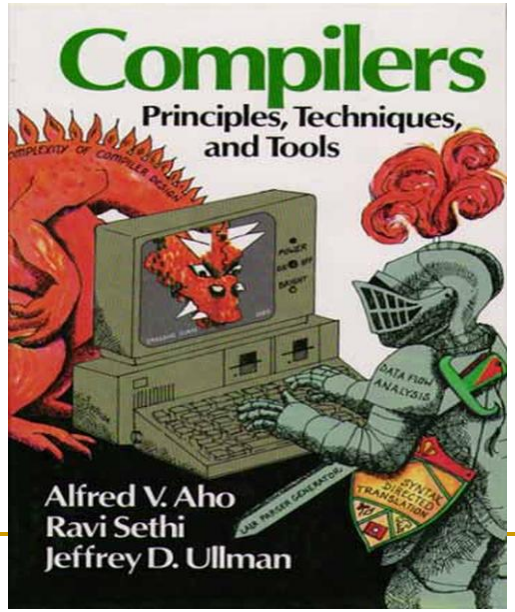
Compiler Design

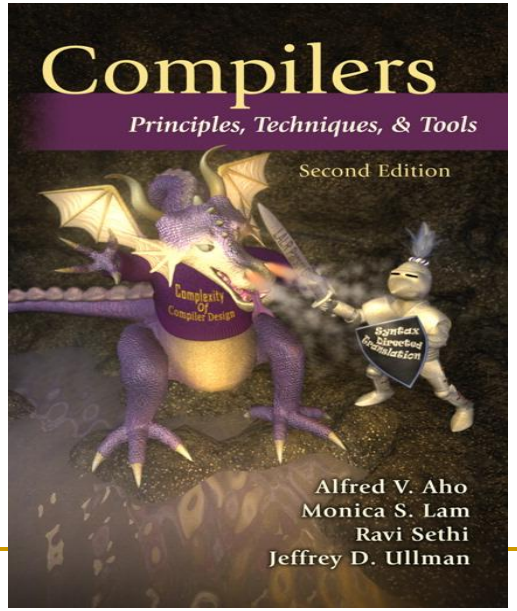
Dr. S. Suresh
Assistant Professor
Department of Computer Science
Banaras Hindu University
Varanasi – 211 005, India.

Textbook

- Compilers: Principles, Techniques, and Tools, 2/E.
  - **Alfred V. Aho**, *Columbia University*
  - **Monica S. Lam**, *Stanford University*
  - **Ravi Sethi**, *Avaya Labs*
  - **Jeffrey D. Ullman**, *Stanford University*



Dragon



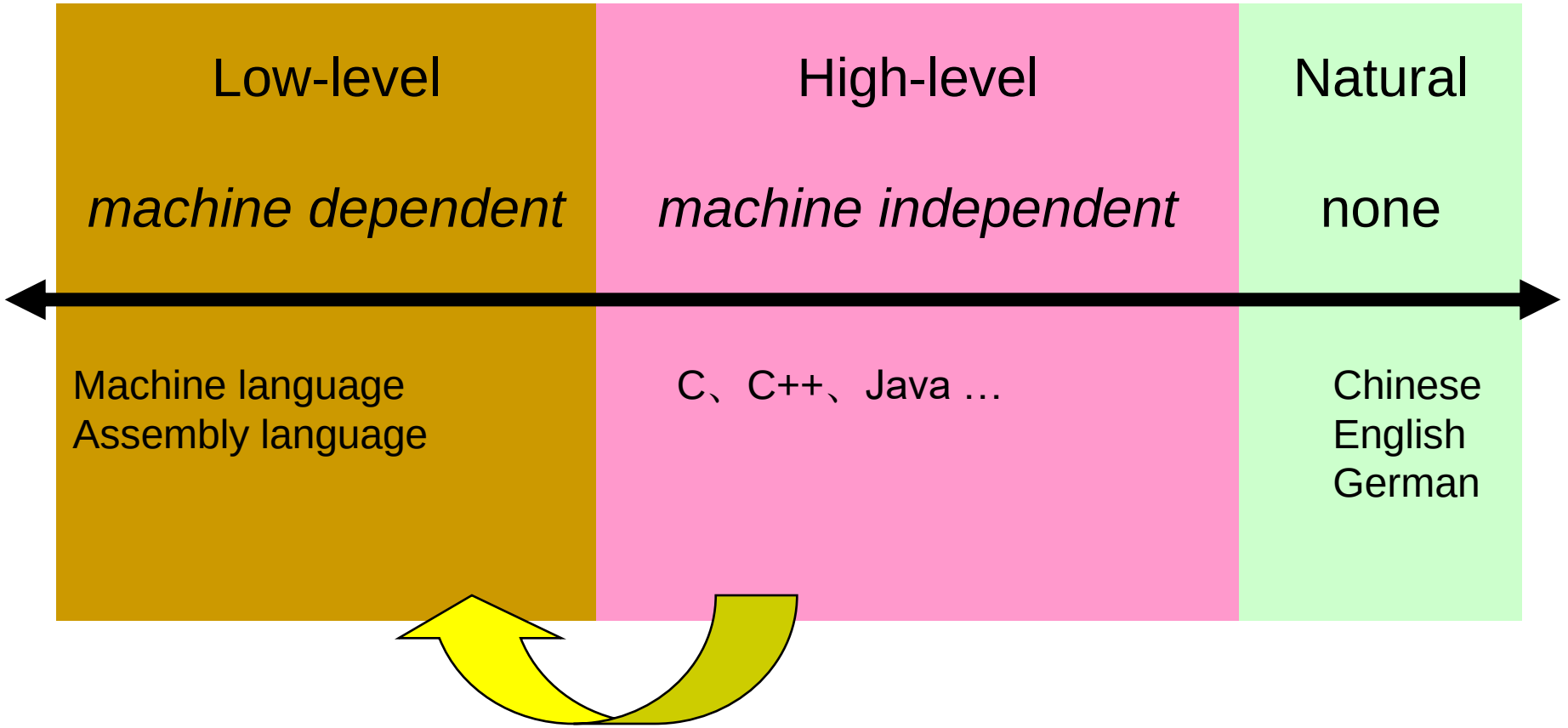
Assessment

- Total Credits: 06 (Theory 4 credits & Lab 2 Credits)
- Theory :
  - Sessional 30% (midterm 20% + attendance 10%)
  - Final exam 70%
- Laboratory :
  - Sessional 30% (regular lab performance, assignment, report, etc.)
  - Final exam 70% (viva + lab performance)

Objectives

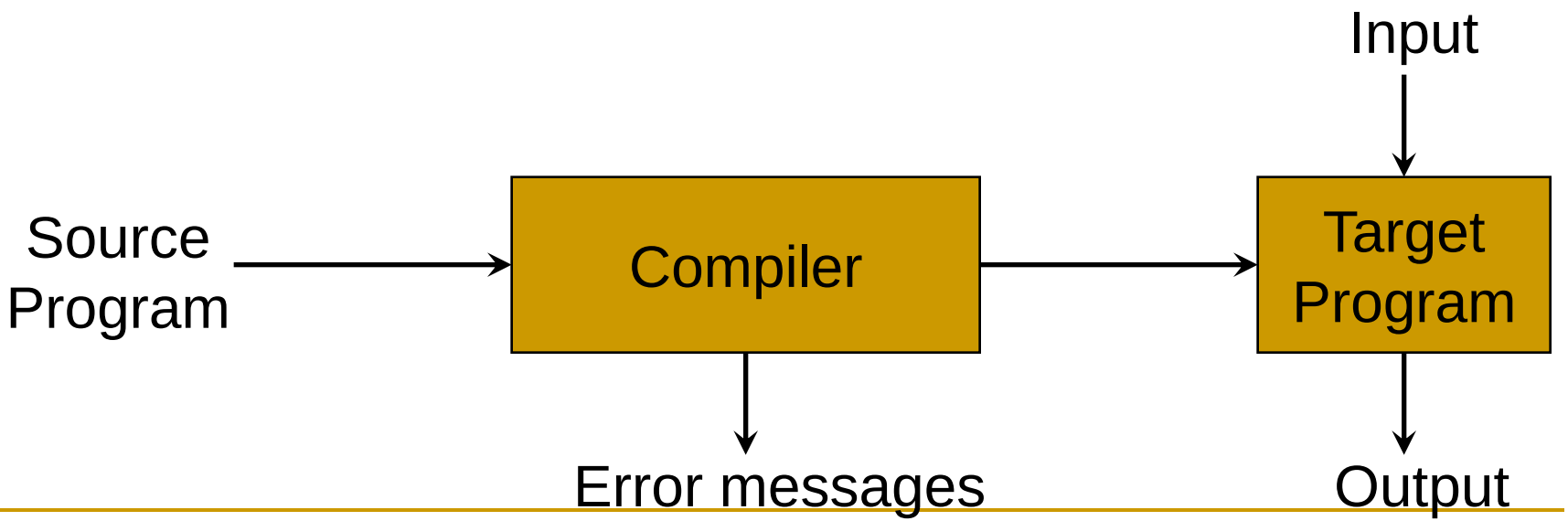
- Know the various phases of compiling process
- Know how to use compiler construction tools, such as generators of scanners and parsers
- Be able to define LL(1), LR(1), and LALR(1) grammars
- Be familiar with compiler analysis and optimization techniques
- Be able to build a compiler for a (simplified) (programming) language

Programming Languages



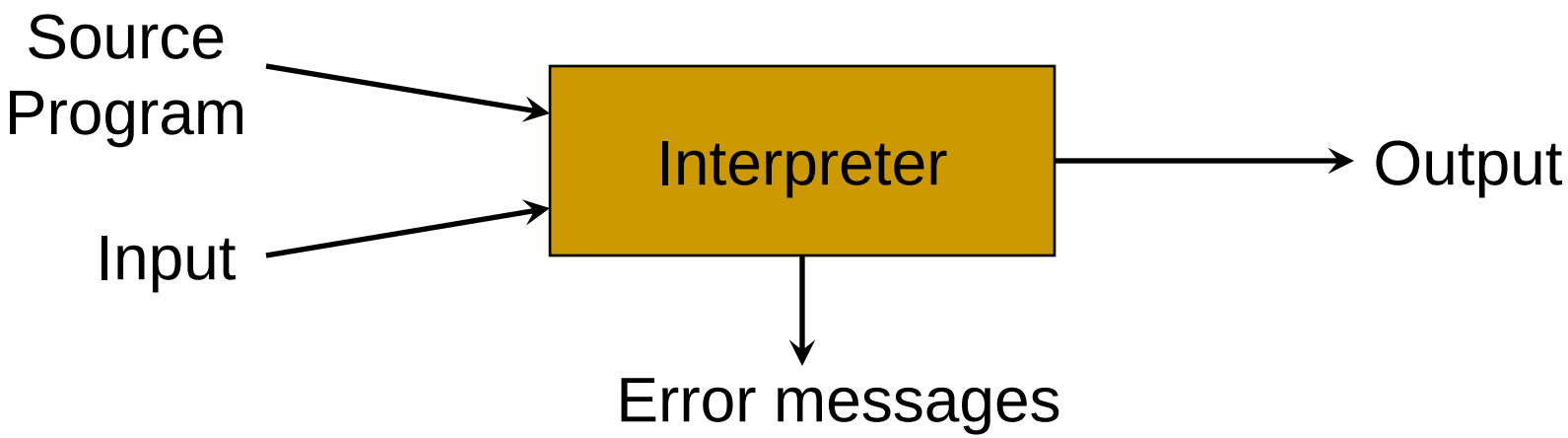
Compilers

- "*Compilation*"
  - Translation of a program written in a source language into a semantically equivalent program written in a target language



Interpreters

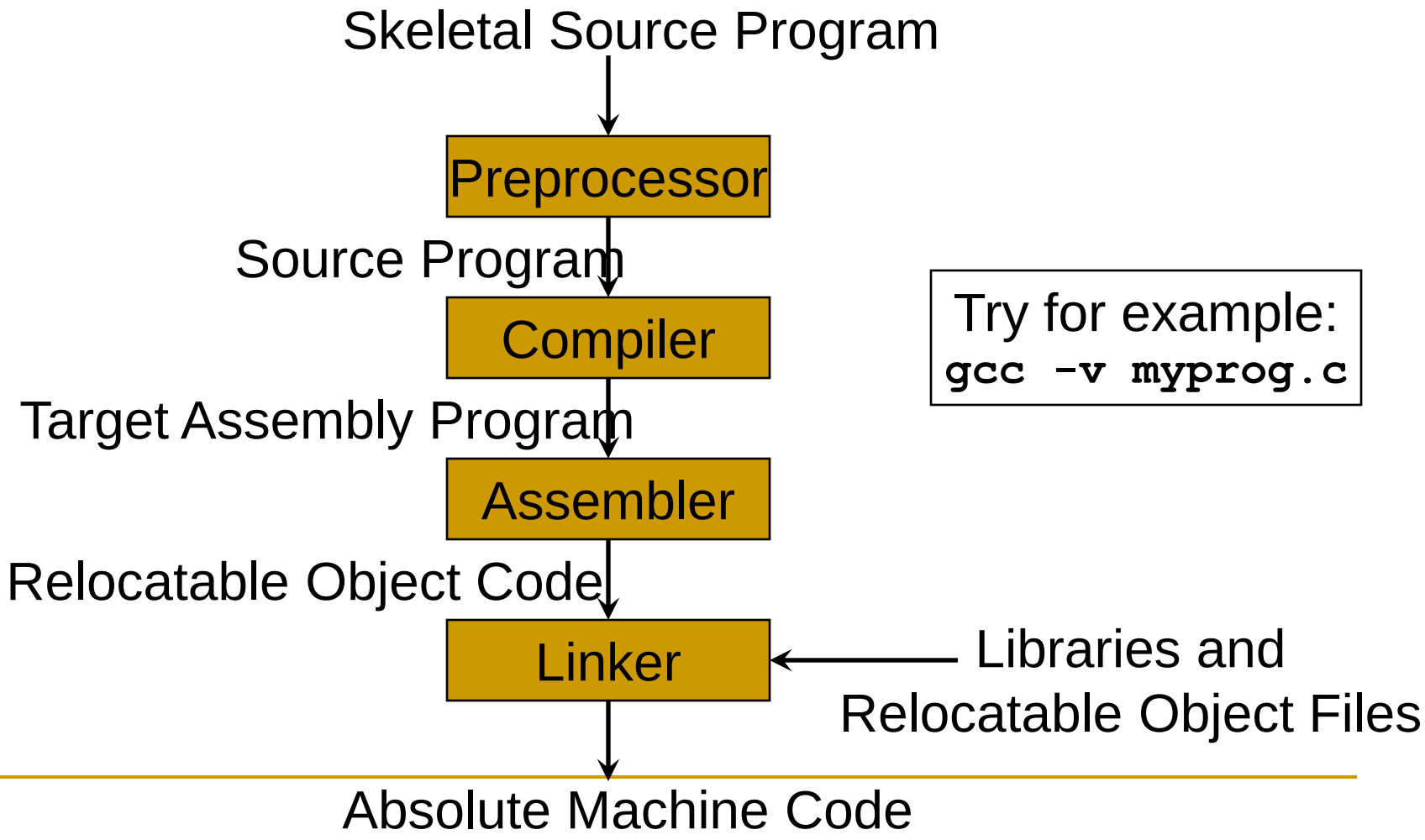
- "*Interpretation*"
  - Performing the operations implied by the source program



Compiler vs. Interpreter

	Compiler	Interpreter
Translation method	Translates program as a whole	One statement at a time
Debugging	Harder	Easier
Intermediate code generation	Yes	No

Preprocessors, Compilers, Assemblers, and Linkers



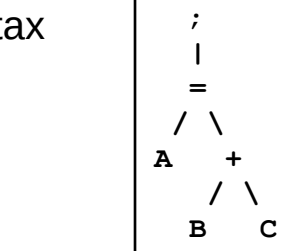
The Analysis-Synthesis Model of Compilation

- There are two parts to compilation:
  - *Analysis* determines the operations implied by the source program which are recorded in a tree structure
  - *Synthesis* takes the tree structure and translates the operations therein into the target program

Tools that Use the Analysis-Synthesis Model

- *Programming Languages (C、C++…)*
- *Scripts (Javascript、bash)*
- *Editors* (syntax highlighting)
- *Pretty printers* (e.g. Doxygen)
- *Static checkers* (e.g. Lint and Splint)
- *Interpreters*
- *Text formatters* (e.g. TeX and LaTeX)
- *Silicon compilers* (e.g. VHDL)
- *Query interpreters/compilers* (Databases)

The Phases of a Compiler

Phase	Output	Sample
Programmer (source code producer)	Source string	`A=B+C;`
Scanner (performs *lexical analysis*)	Token string	`'A', '=', 'B', '+', 'C', ';'` And *symbol table* with names
Parser (performs *syntax analysis* based on the grammar of the programming language)	Parse tree or abstract syntax tree	
Semantic analyzer (type checking, etc)	Annotated parse tree or abstract syntax tree	

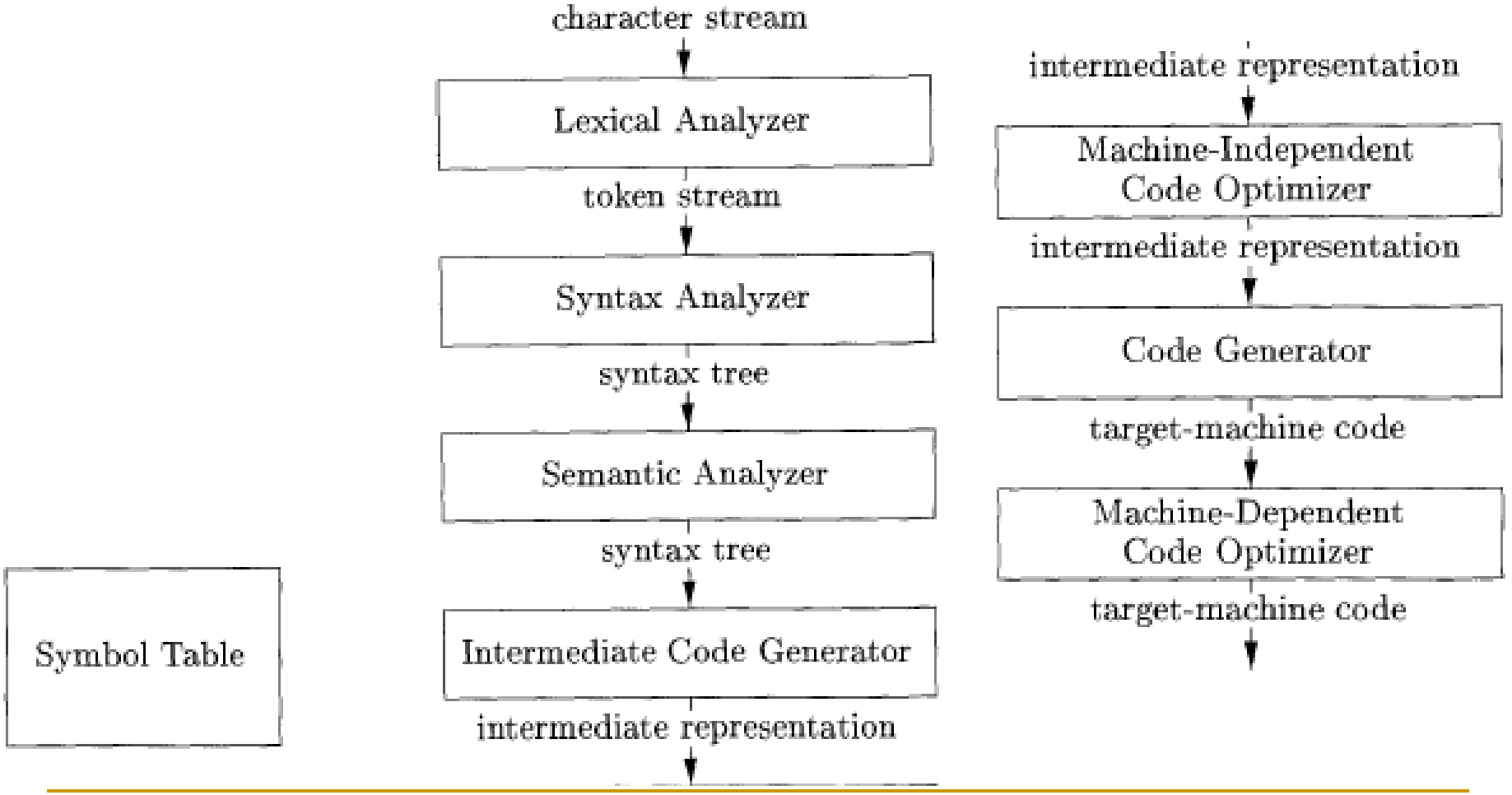
Intermediate code generator	Three-address code, quads, or RTL	`int2fp B t1`
		`+ t1 C t2`
		`:= t2 A`

Optimizer	Three-address code, quads, or RTL	`int2fp B t1` `+ t1 #2.3 A`

Code generator	Assembly code	`MOVF #2.3,r1`
		`ADDF2 r1,r2`
		`MOVF r2,A`

Peephole optimizer	Assembly code	`ADDF2 #2.3,r2` `MOVF r2,A`

The Phases of a Compiler



The Grouping of Phases

- Compiler *front* and *back ends*:
  - Front end: *analysis* (*machine independent*)
  - Back end: *synthesis* (*machine dependent*)
- Compiler *passes:*
  - A collection of phases is done only once (*single pass*) or multiple times (*multi pass*)
    - Single pass: usually requires everything to be defined before being used in source program
    - Multi pass: compiler may have to keep entire program representation in memory

Compiler-Construction Tools

- Software development tools are available to implement one or more compiler phases
  - *Scanner generators*
  - *Parser generators*
  - *Syntax-directed translation engines*
  - *Automatic code generators*
  - *Data-flow engines*

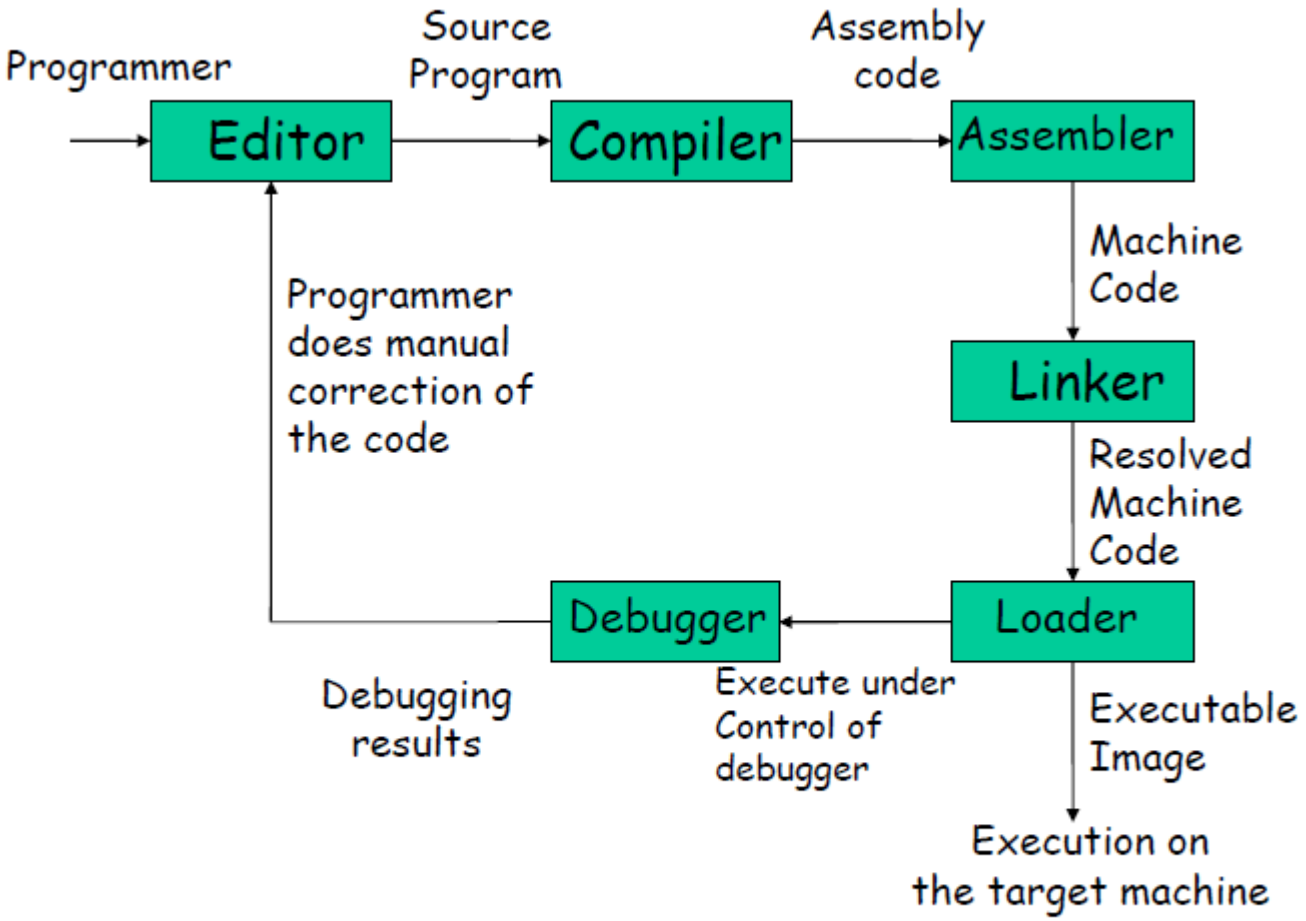
Some early machines and implementations

- IBM developed 704 in 1954. All programming was done in assembly language. Cost of software development far exceeded cost of hardware. Low productivity.
- Speedcoding interpreter (1953): programs ran about 10 times slower than hand written assembly code.
- John Backus (in 1954): Proposed a program that translated high level expressions into native machine code. Skeptism all around. Most people thought it was impossible.
- Fortran I project (1954-1957): The first compiler was released.

Fortran I

- The first compiler had a huge impact on the programming languages and computer science. The whole new field of compiler design was started.
- More than half the programmers were using Fortran by 1958.
- The development time was cut down to half.
- Led to enormous amount of theoretical work (lexical analysis, parsing, optimization, structured programming, code generation, error recovery etc.).
- Modern compilers preserve the basic structure of the Fortran I compiler !!!

Program development and processing



Syllabus Outline

- Introduction
- Lexical Analysis and Lex/Flex
- Syntax Analysis and Yacc/Bison
- Syntax-Directed Translation
- Type Checking
- Run-Time Environments
- Intermediate Code Generation
- Code Generation and Optimization