The Design and Implementation of Programming Languages

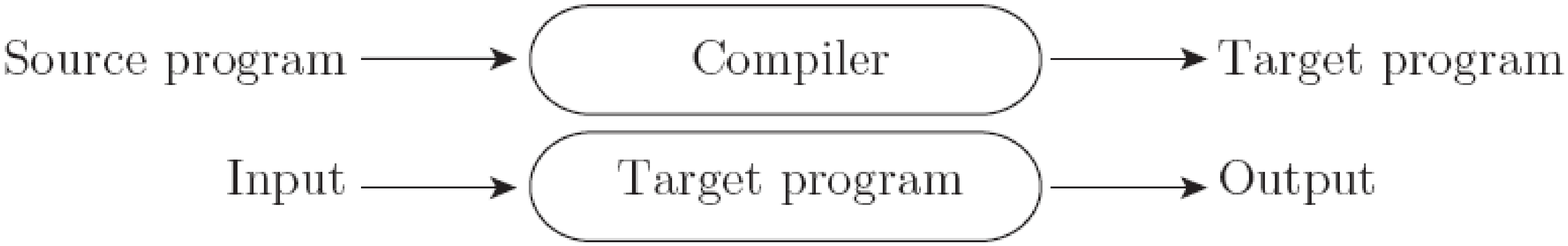
In Text: Chapter 1

Slides created by Na Meng, Faryaneh Poursardar

Language Implementation Methods

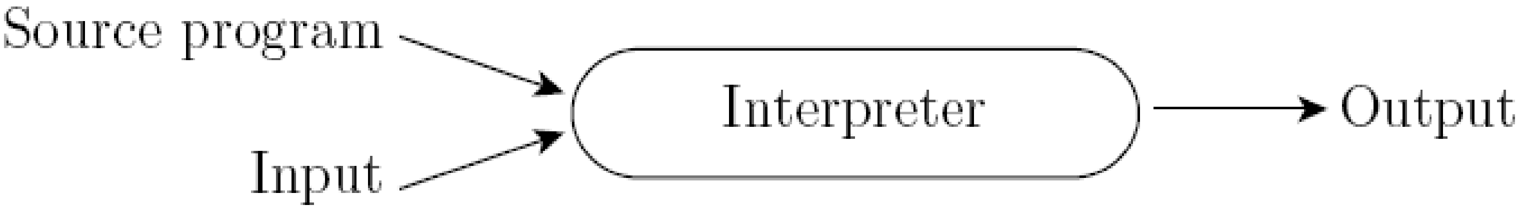
- Compilation
- Interpretation
- Hybrid

Compilation



- Translate high-level programs to machine code
- Slow translation
- Fast execution
- E.g. C, C++

Interpretation

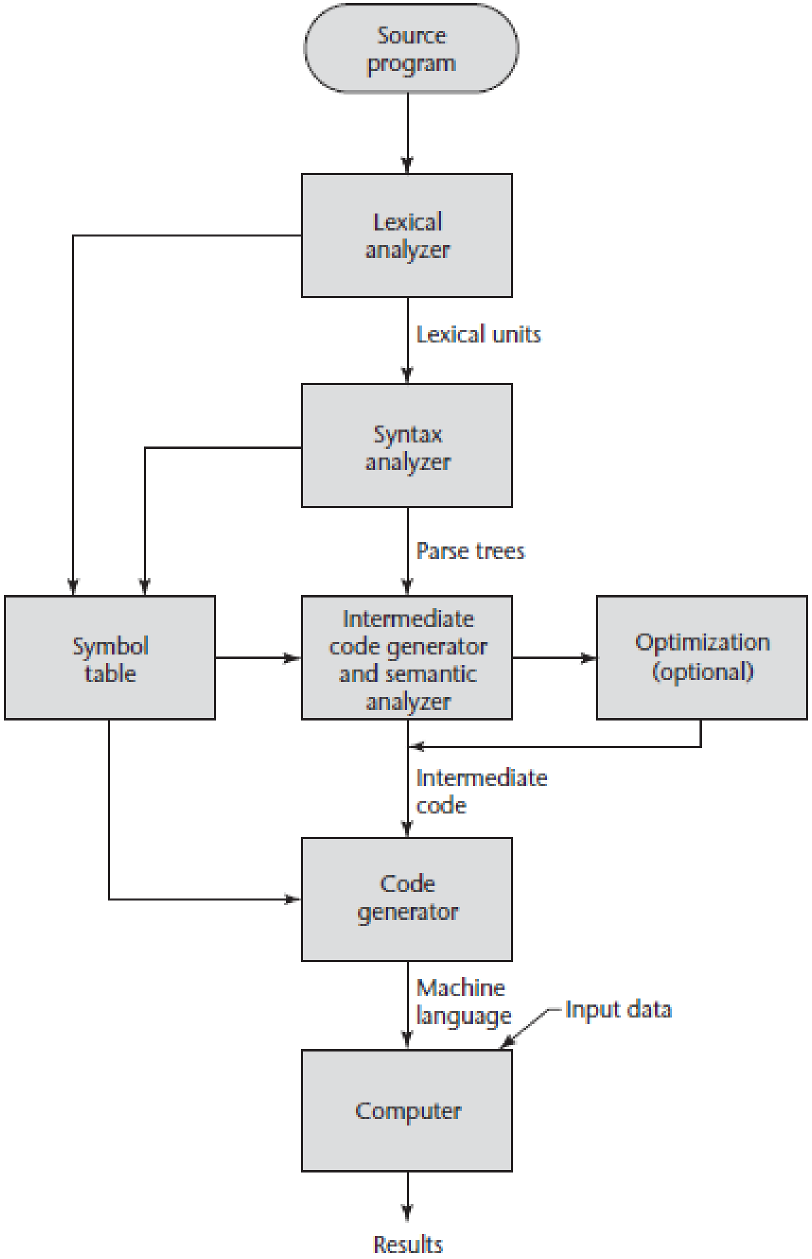


- Interpret one statement and then execute it on a virtual machine
- No translation
- Slow execution
- E.g., Basic

Compilation vs. Interpretation

- Compilation
  - Better performance
    - No runtime cost for interpretation
    - Program optimization
- Interpretation
  - Better diagnosis (with excellent source-level debugger)
  - Earlier diagnosis (execute erroneous program)

Compilation Process



Scanning (Lexical Analysis)

- Break the program into “tokens”—the smallest meaningful units
  - This can save time, since character-by-character processing is slow
- We can tune the scanner better
  - E.g., remove spaces & comments
- A scanner uses a Deterministic Finite Automaton (DFA) to recognize tokens

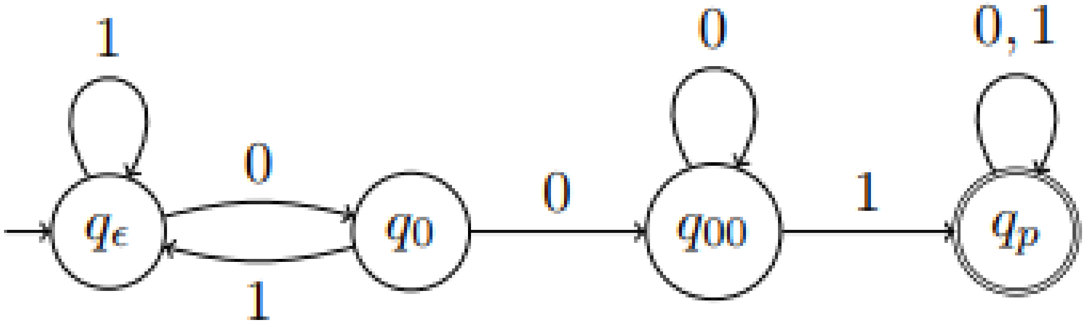
Tokens

Or lexical units are:

- Identifiers
- Special words
- Operators
- Punctuation symbols

- Scanner ignores comments

- Example of DFA
- Accepting strings having 001 substring



A running example: Greatest Common Divisor (GCD)

```
int main() {
    int i = getint(),
        j = getint();
    while (i != j) {
        if (i > j) i = i - j;
        else j = j - i;
    }
    putint(i);
}
```

Token sequence:

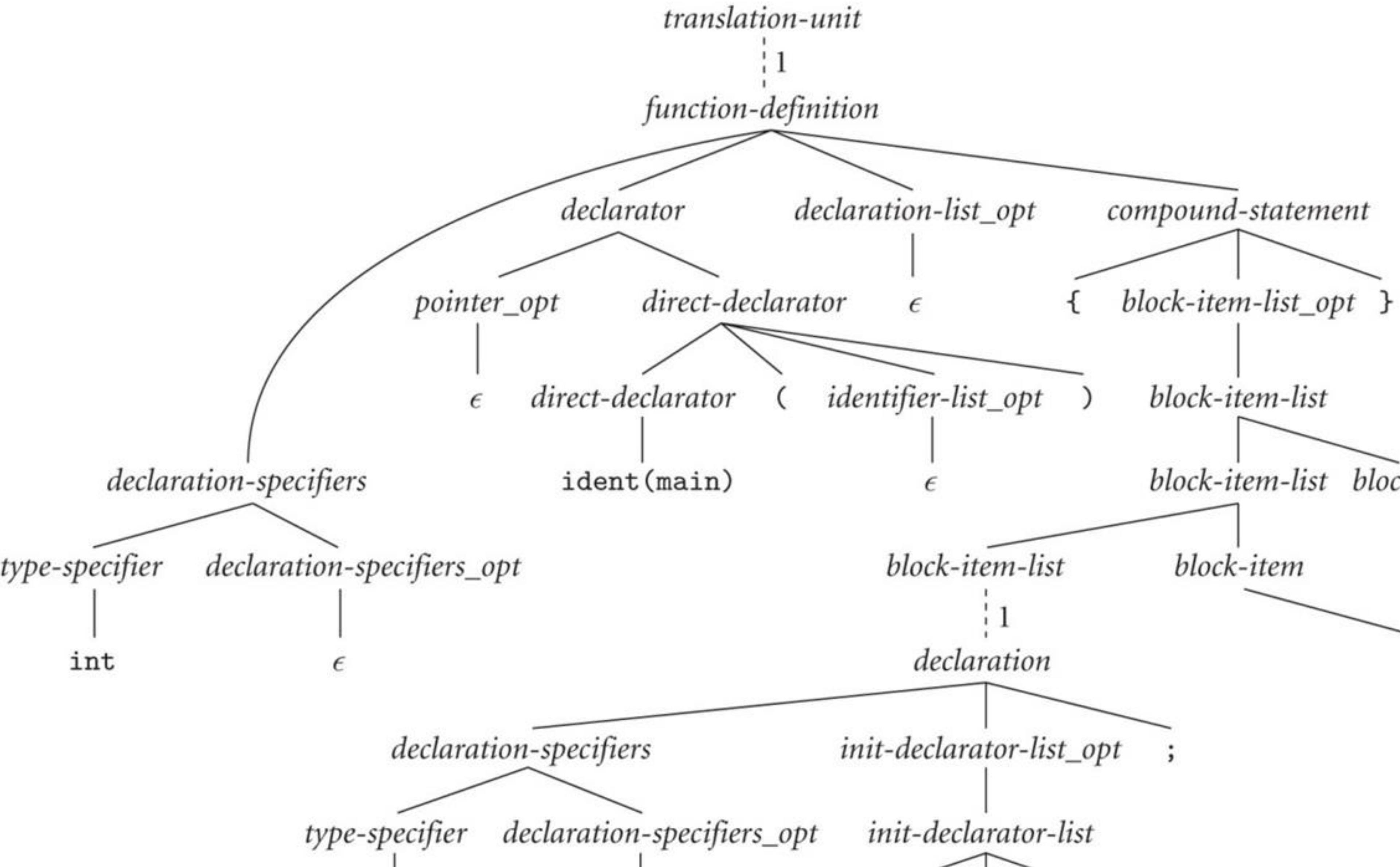
```
int     main    (     )     {
int     i       =     getint
(       )       ,     j     =
getint  (       )     ;     while
(       i       !=    j     )
{       if      (     i     >
j       )       i     =     i
-       j       ;     else j
=       j       -     i     ;
}       putint  (     i     )
;       }
```

Parsing (Syntax Analysis)

- Organize tokens into a parse tree that represents higher-level constructs (statements, expressions, subroutines)
  - Each construct is a node in the tree
  - Each construct's constituents are its children

- Parse tree represents the syntactic structure of the program

GCD Parsing Tree



Semantic Analysis

- Determine the meaning of a program
- Checks for type errors
- A semantic analyzer builds and maintains a symbol table data structure that maps each identifier to the information known about it, such as the identifier's type, internal structure, and scope

Semantic Analysis

- With the symbol table, the semantic analyzer can enforce a large variety of rules to check for errors
- Sample rules:
  - Each identifier is declared before it is used
  - Any function with a non-void return type returns a value explicitly
  - Subroutine calls provide the correct number and types of arguments

Symbol Table

- The symbol table serves as a database for the compilation process.
- The primary contents of the symbol table are the *type and attribute information* of each user-defined name in the program.
- This information is placed in the symbol table by the lexical and syntax analyzers and is used by the semantic analyzer and the code generator.

Intermediate Form

- Generated after semantic analysis
- A code between source program and machine language
- In many compilers, it is in assembly language

Optimization

- Goal: perform analysis and optimization of programs
- Make code faster and smaller

- Optimizing code in machine language is hard
- Best place to perform optimization is in intermediate code

Code generator

- Goal: produce assembly/machine code from optimized low-level representation of program
- Input: optimized low-level representation of program from low-level optimizer
- Output: assembly/machine code for real or virtual machine
- Tasks:
  - Register allocation
  - Instruction selection

Discussion

- Traditionally, all phases of compilation were completed before program was executed

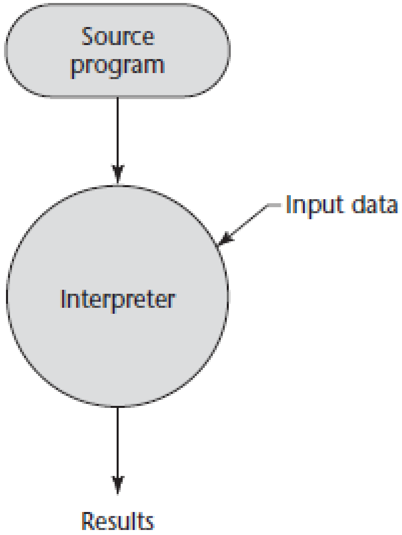
- New twist: virtual machines
  - Offline compiler:
    - Generates code for virtual machine like JVM
  - Just-in-time compiler:
    - Generates code for real machine from VM code while program is executing

- Advantages:
  - Portability
  - JIT compiler can perform optimizations for particular input

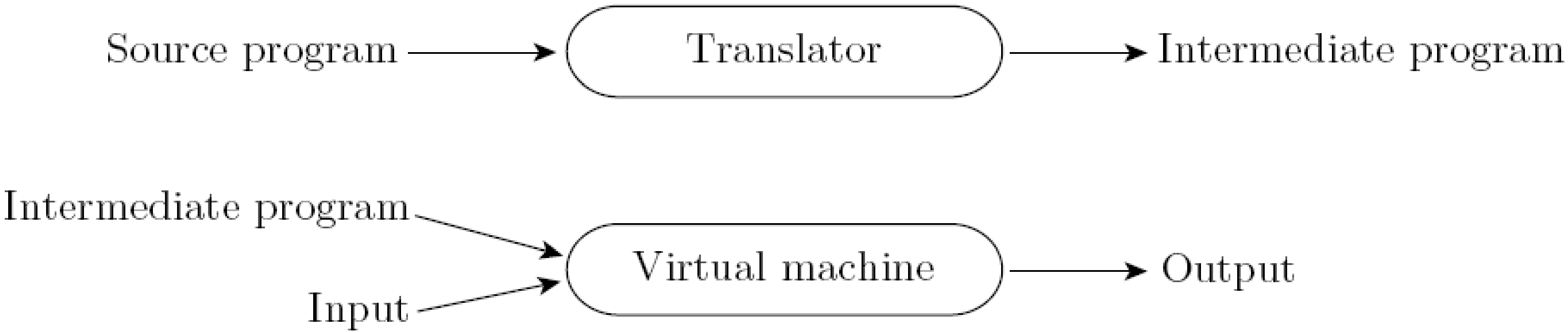
Front end & back end

- Front end
  - To analyze the source code in order to build an internal representation (IR) of the program
  - It includes: lexical analysis, syntactic analysis, and semantic analysis
- Back end
  - To gather and analyze program information from IR, to optimize the code, and to generate machine code
  - It includes: optimization and code generation

Pure Interpretation



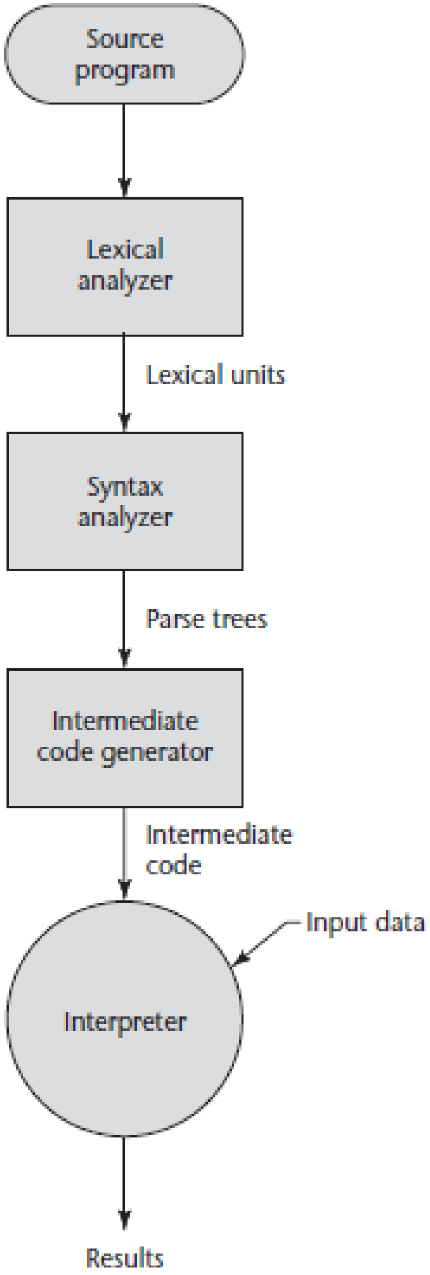
Hybrid Implementation



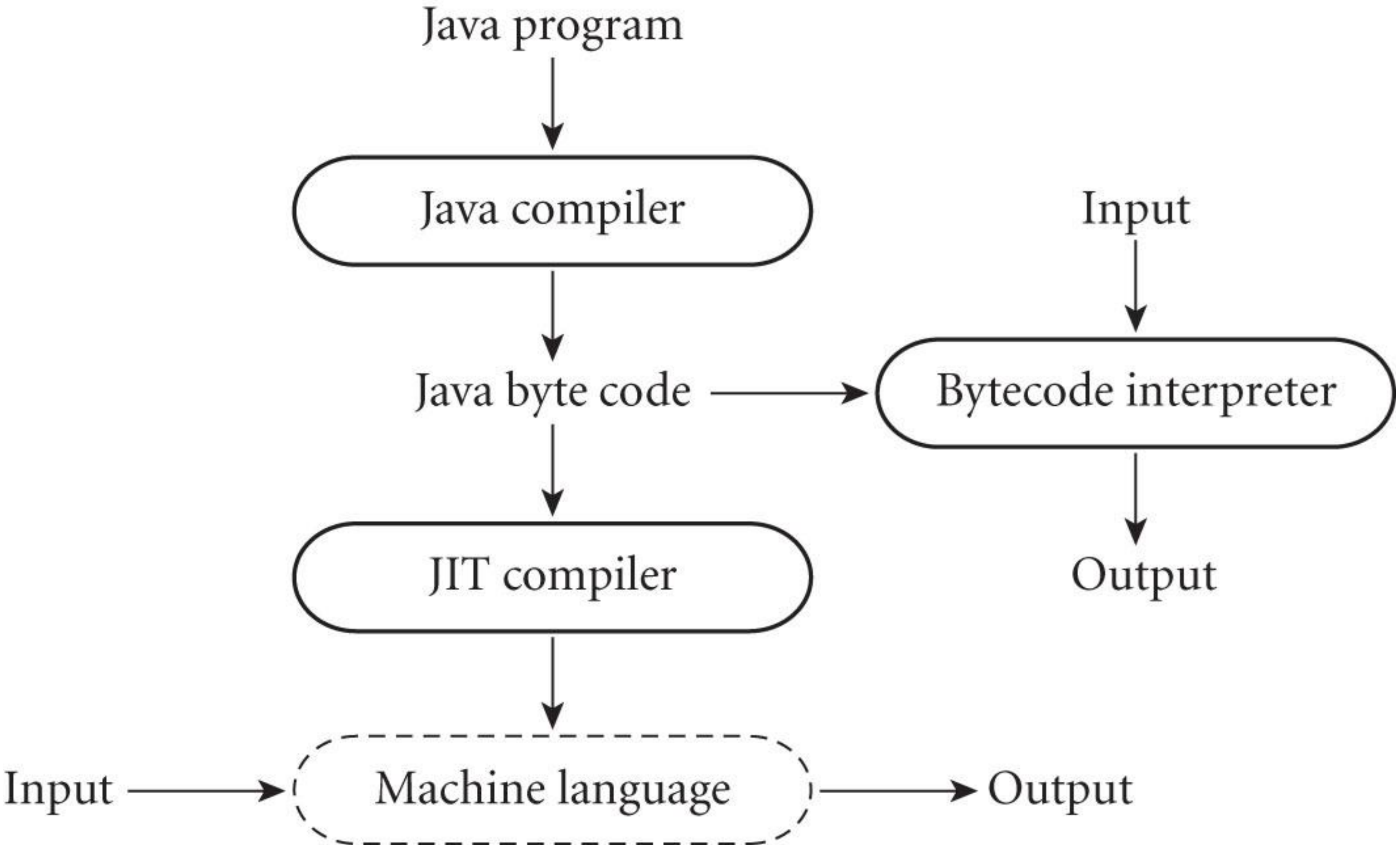
- Quick start in “Interpretation” mode
- Compile code on hot paths to speed up
  - E.g., Just-in-Time (JIT) compiler in Java Virtual Machine (JVM)

- Small translation cost
- Medium execution speed

Hybrid Implementation System



Hybrid Implementation (Java)



Implementation Strategies in Practice

- Preprocessing
- Library routines and linking
- Post-compilation assembly
- Source-to-source translation
- Bootstrapping

Preprocessing (Basic)

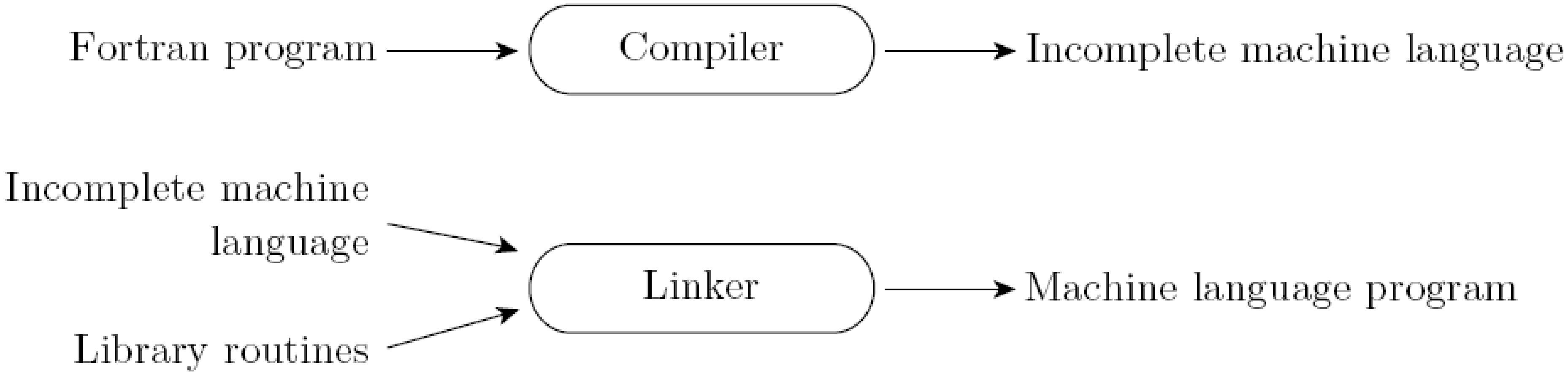
- An initial translator
  - to remove comments and white spaces,
  - to group characters together into tokens such as keywords, identifiers, numbers, and symbols,
  - to expand abbreviations in the style of a macro assembler, and
  - to identify higher-level syntactic structures, such as loops and subroutines
- Goal
  - To provide an intermediate form that mirrors the structure of the source, but can be interpreted more efficiently

Preprocessing (C)

- Conditional compilation
  - Delete portions of code to allow several versions of a program to be built from the same source
  - Copy the extra content(library/header) into the program

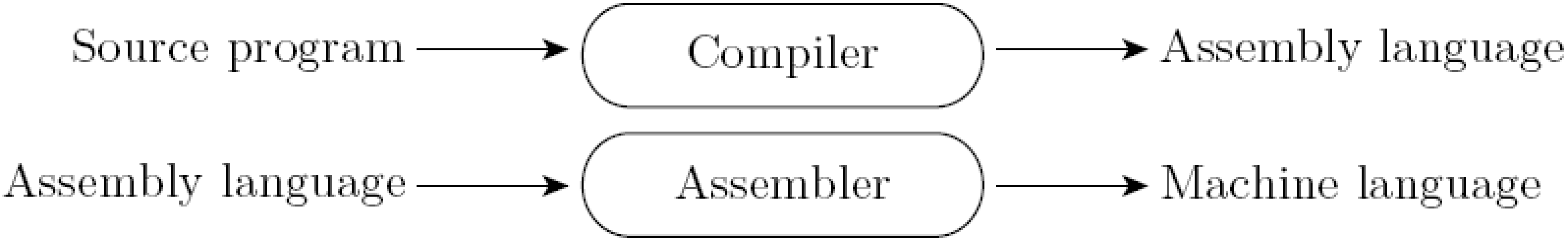
Library routines and linking (Fortran)

- The compilation of source code counts on the existence of a library of subroutines invoked by the program



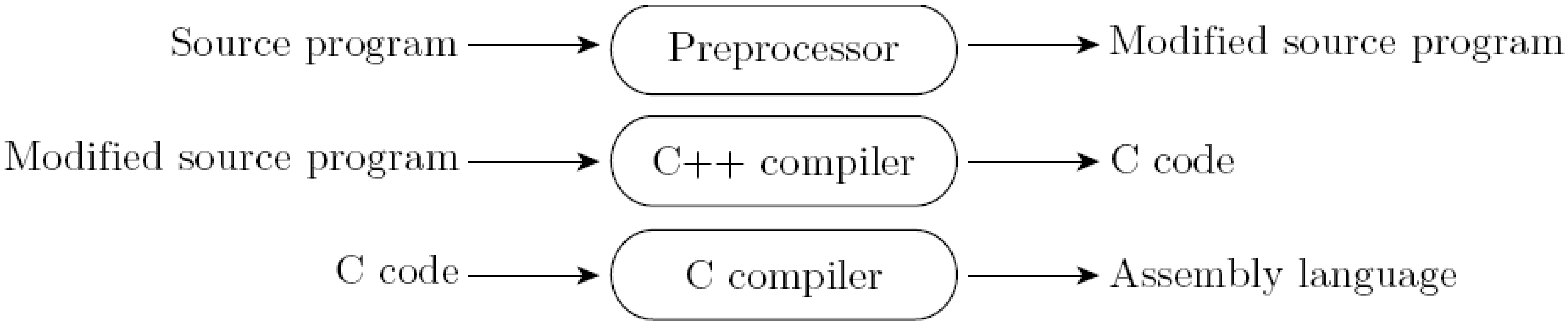
Post-compilation assembly

- Source code is first compiled to assembly code, and then the assembler translates it to machine code
  - To facilitate debugging (assembly code is easier to read)
  - To isolate the compiler from changes in the format of machine language files (only the commonly shared assembler must be changed)



Source-to-Source Translation

- AT&T C++ compiler
  - To translate C++ programs to C programs
  - To facilitate reuse of compilers or language support



Bootstrapping

- Many compilers are self-hosting:
  - They are written in the language they compile
  - Bootstrapping is used to compile the compiler in the first place