# CSE M226: Programming Languages

## Lecture 1: Introduction to Programming Languages

Rokan Uddin Faruqui

Dept. of Computer Science and Engineering
University of Chittagong, Bangladesh
Email: rufaruqui@cu.ac.bd

# Why Study Programming Languages?

- To choose the most appropriate language for any given task
  - **Fortran** for symbolic computing or string processing
  - **C**, **C++**, or **C#** for systems programming?
  - **Fortran** or **C** for scientific computations?
  - **PHP** or **Ruby** for a web-based application?
  - **Ada** or **C** for embedded systems?
  - **Visual Basic** or **Java** for a graphical user interface?

# Why Study Programming Languages?

- Make it easier to learn new languages
  - Java and C# are easier to learn if you already know C++
  - Common Lisp if you already know Scheme
  - Haskell if you already know ML

# Why Study Programming Languages?

- **Understand obscure features**

# Why Study Programming Languages?

- **Understand obscure features**
  - variable number of arguments
  - union
  - multiple inheritance

# Why Study Programming Languages?

- **Choose among alternative ways to express things**

# Why Study Programming Languages?

- **Choose among alternative ways to express things**
  - copy constructor *instead of* temporary variables
  - executor *instead of* explicit thread creation
- **Make good use of debuggers, assemblers, linkers, and related tools**

# Why Study Programming Languages?

- **Choose among alternative ways to express things**
  - copy constructor *instead of* temporary variables
  - executor *instead of* explicit thread creation
- **Make good use of debuggers, assemblers, linkers, and related tools**

# Why Study Programming Languages?

- **Simulate useful features in languages that lack them**

# Why Study Programming Languages?

- **Simulate useful features in languages that lack them**
  - useful features missing in older languages: well-structured code, iterators
- **Make better use of language technology wherever it appears**

# Why Study Programming Languages?

- **Simulate useful features in languages that lack them**
  - useful features missing in older languages: well-structured code, iterators
- **Make better use of language technology wherever it appears**

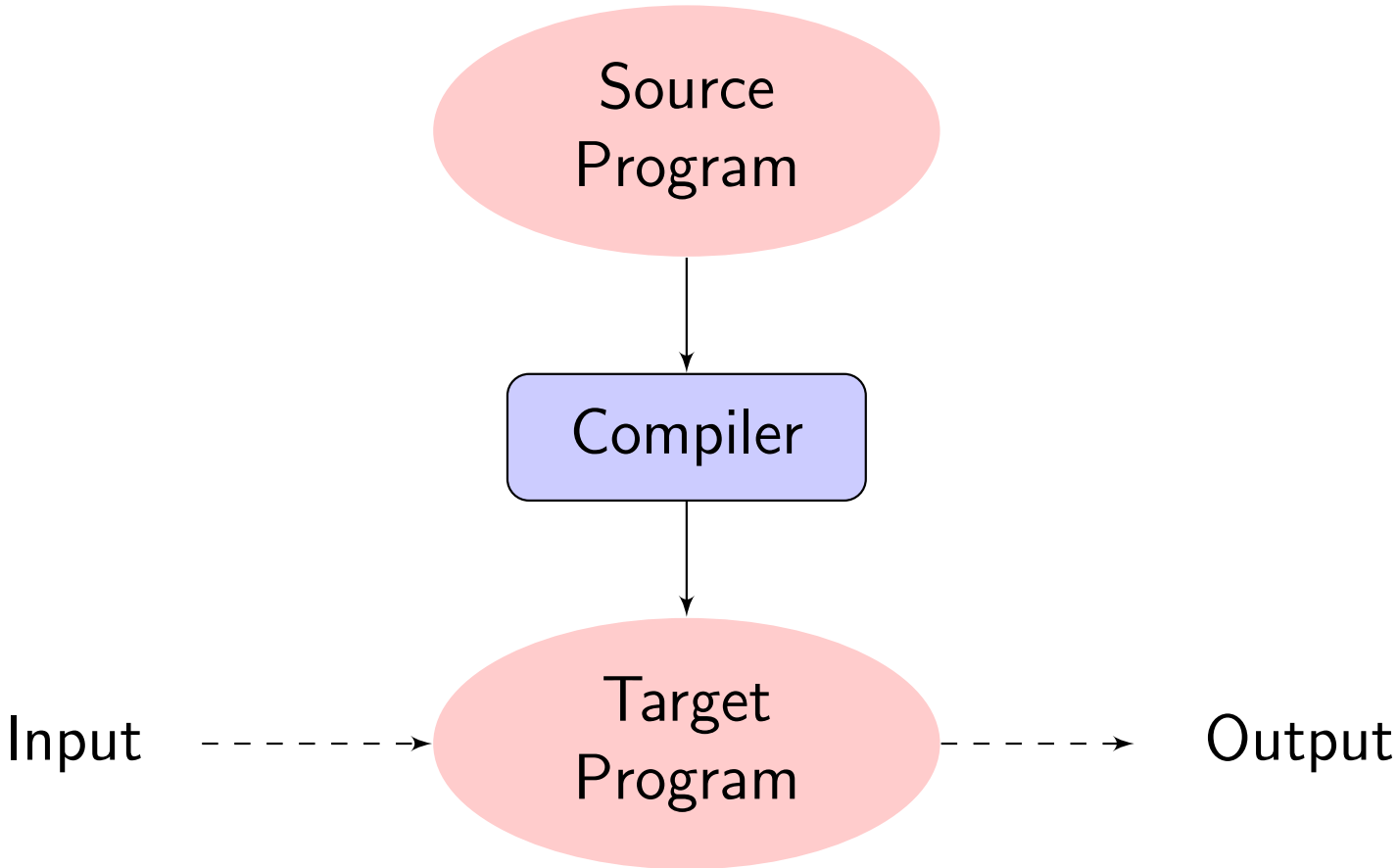
# Compilation and Interpretation

- Pure compilation
- Pure interpretation
- Mixing compilation and interpretation
- Preprocessing
- Library routines and linking
- Post-compilation assembly
- The C preprocessor
- Source-to-source translation
- Bootstrapping
- Dynamic and just-in-time compilation

# Compilation and Interpretation

## Pure Compilation

*translates* - a source program to a target program



# Compilation and Interpretation

## Pure Compilation

*translates* - a source program to a target program

- compiler is the locus of control during compilation
- target program is the locus of control during its own execution
- compiler is itself a machine language program
- machine language is commonly known as object code

## Pure Interpretation

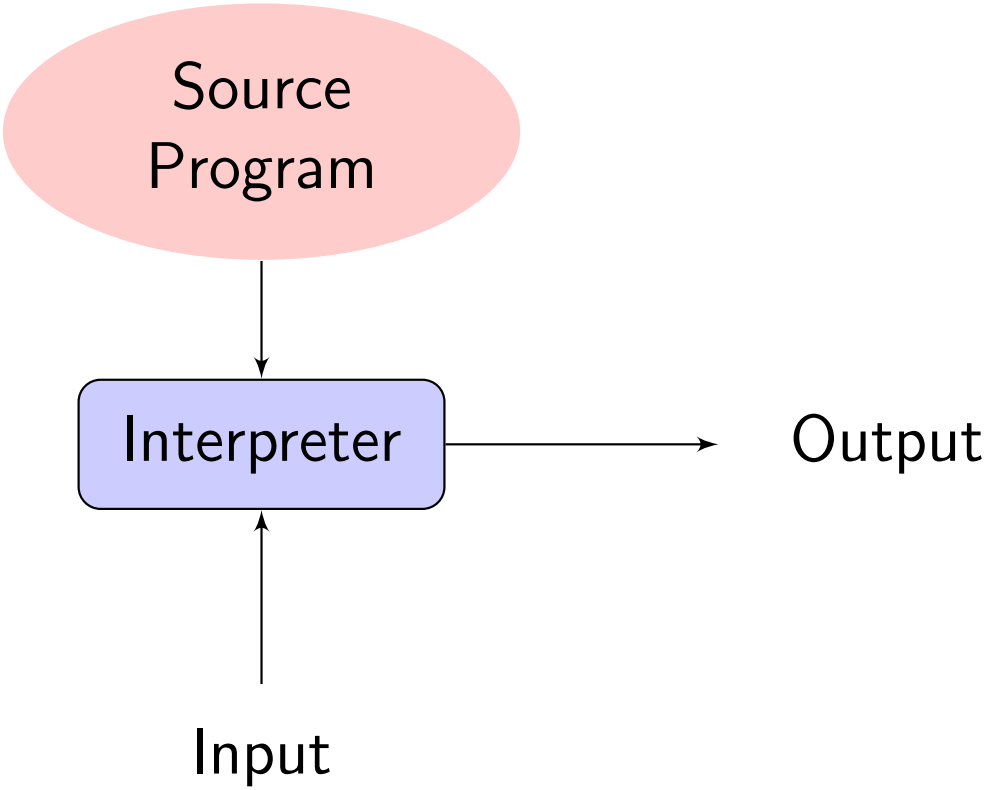



Figure: Pure Interpretation

# Compilation and Interpretation

## Pure Interpretation

- *late binding* - delaying decisions about program implementation until run time
- source-level debugger

# Compilation and Interpretation

## Mixing compilation and interpretation

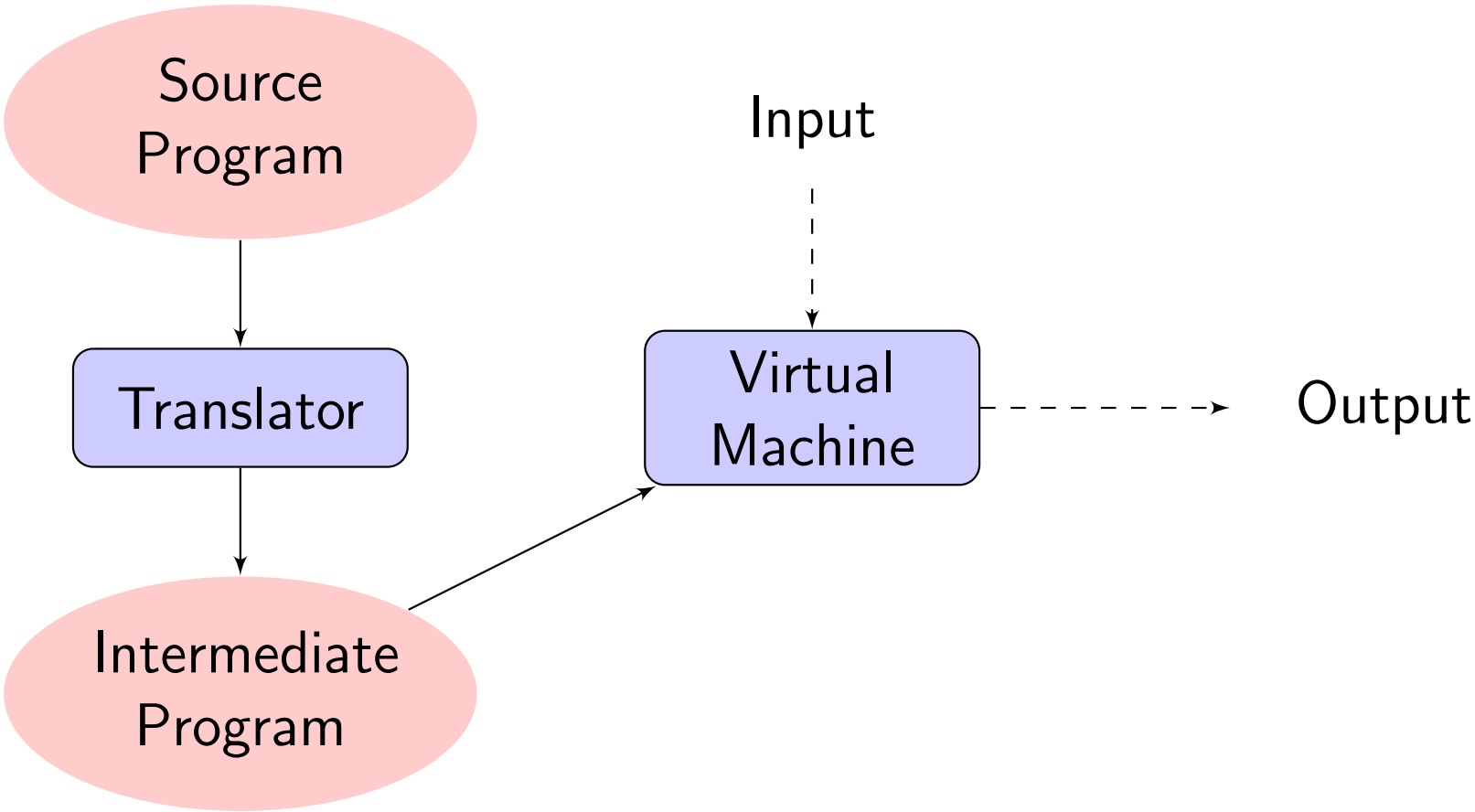



Figure: Mixing compilation and interpretation

# Compilation and Interpretation

## Preprocessing

- removes comments and white space
- groups characters together into tokens such as keywords, identifiers, numbers, and symbols
- produce an intermediate form that can be interpreted more efficiently

# Compilation and Interpretation

## Library routines and linking

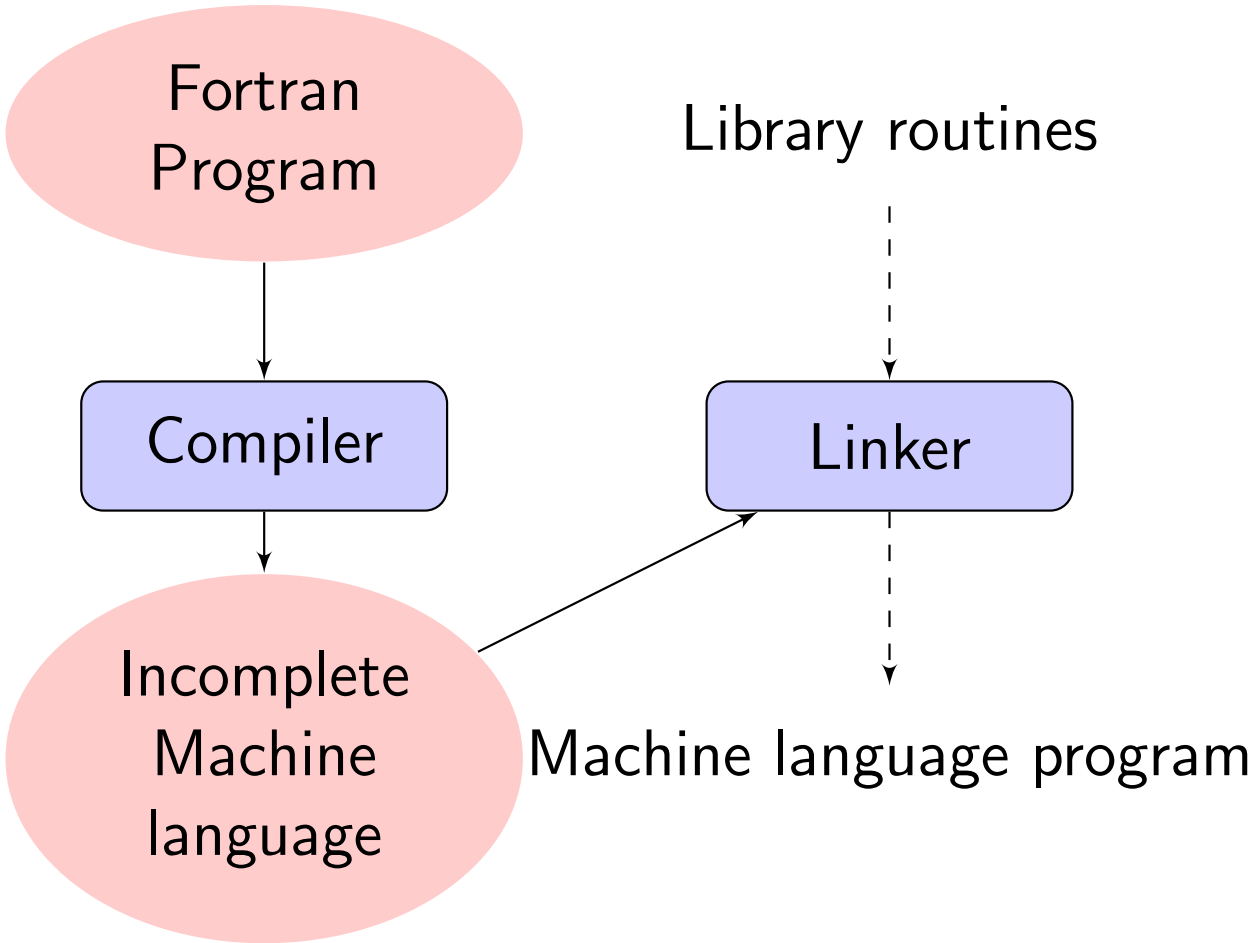



Figure: Fortran program Compilation

# Compilation and Interpretation

## Post-compilation assembly

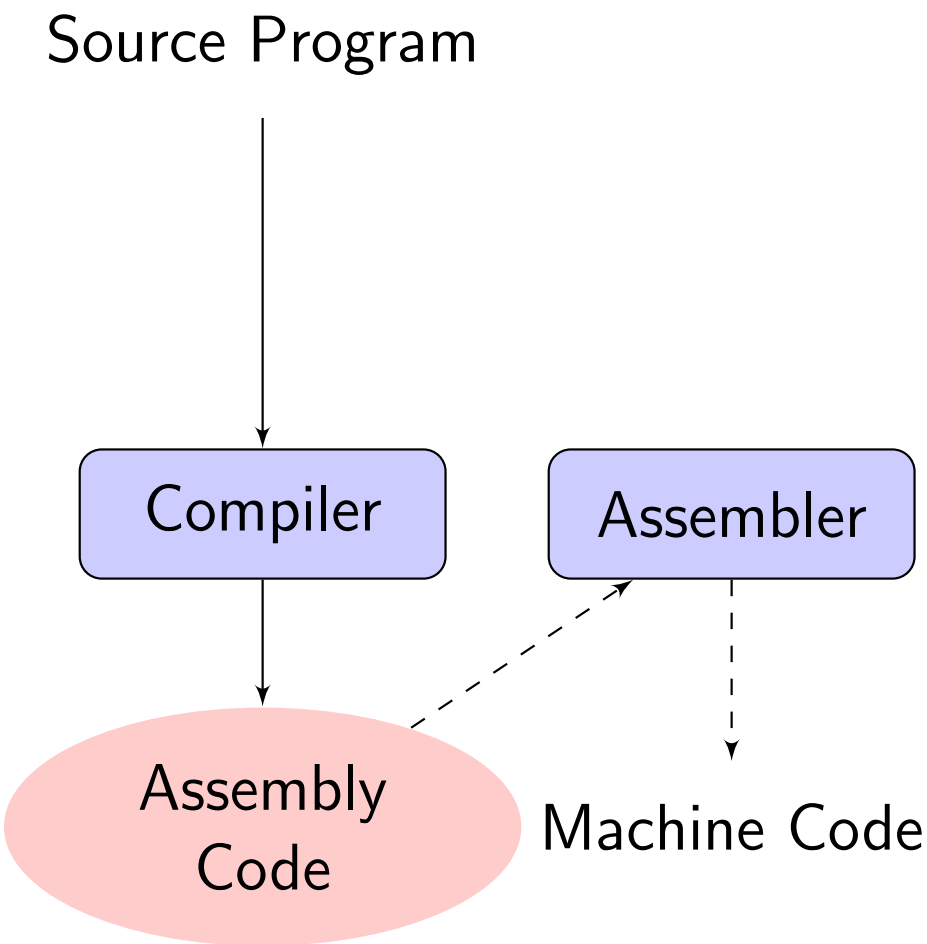
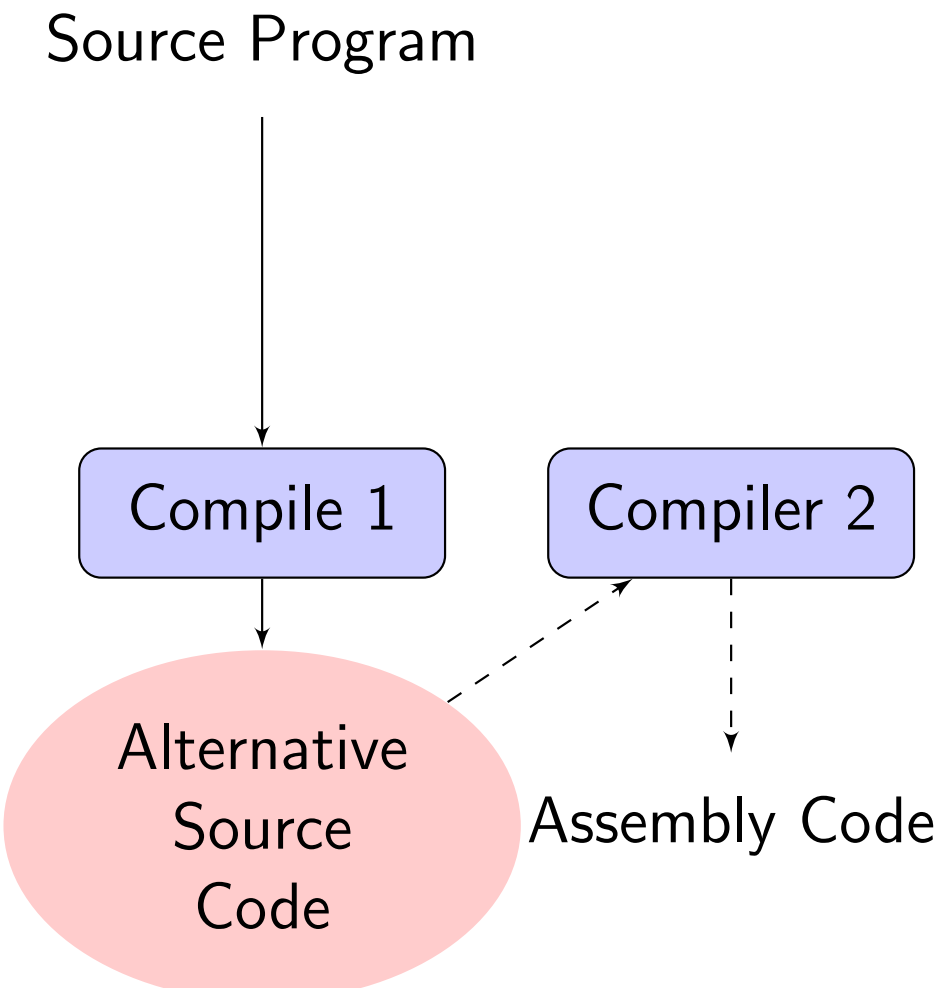



Figure: Pure compilation

# Compilation and Interpretation

## Source-to-source translation

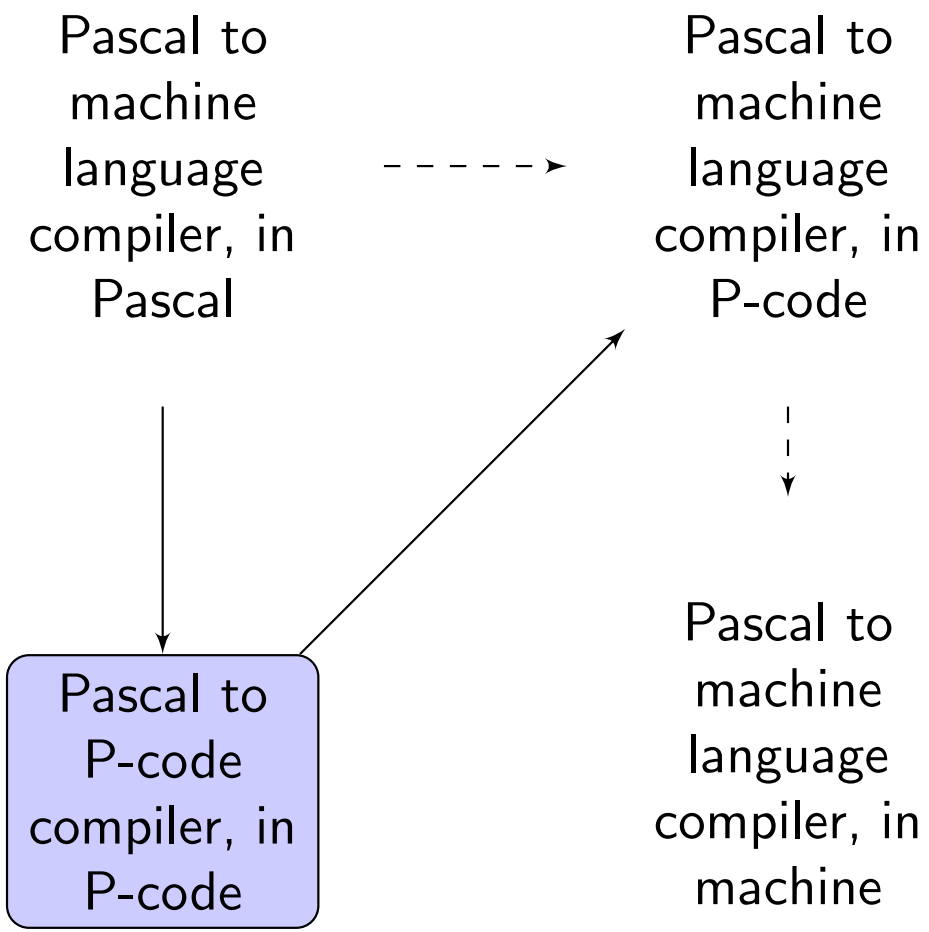
Example - AT&T's original compiler for **C++**



# Compilation and Interpretation

## Bootstrapping

*Self-hosting* - **Ada** compilers in **Ada**, **C** compilers in **C**.



# Compilation and Interpretation

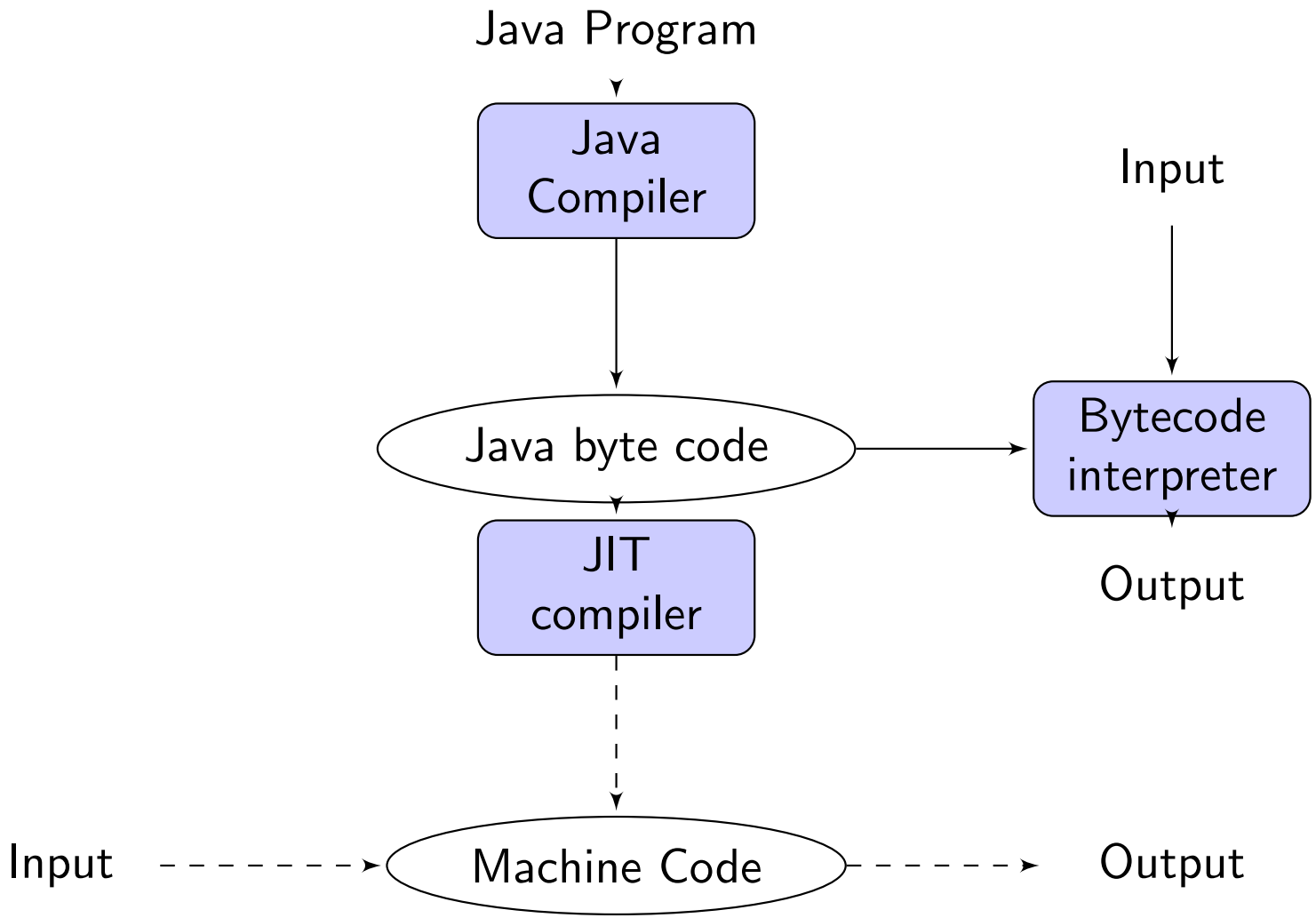



Figure: Just-In Time compilation

# An overview of Compilation

Scanner (lexical analysis)

Parser (syntax analysis

Semantic analysis and intermediate code generation

Machine-independent code improvement (optional)

Target code generation

Machine-specific code improvement (optional)

Figure: Phases of compilation

# An overview of Compilation

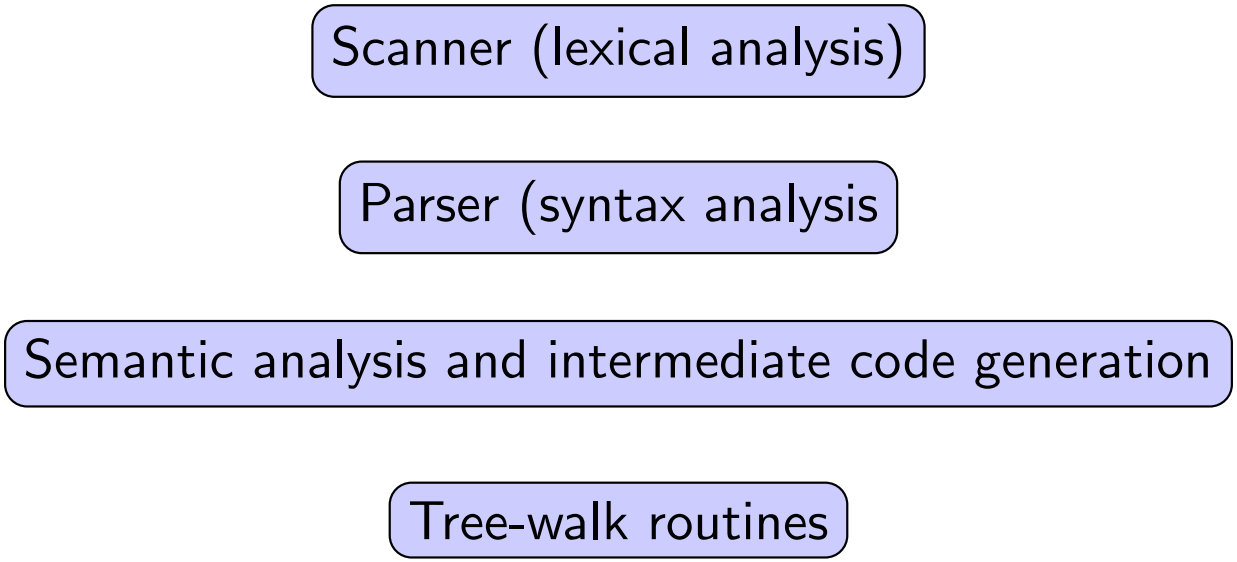



Figure: Phases of interpretation