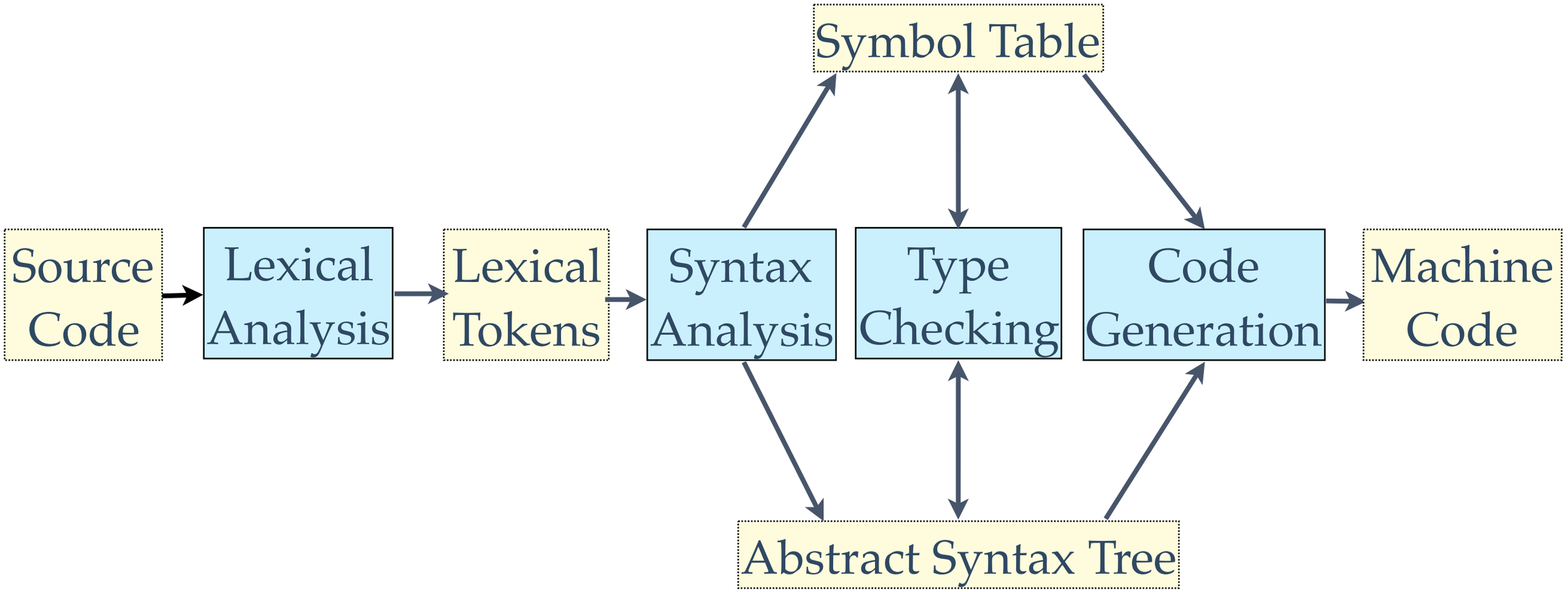
Phases of a Compiler

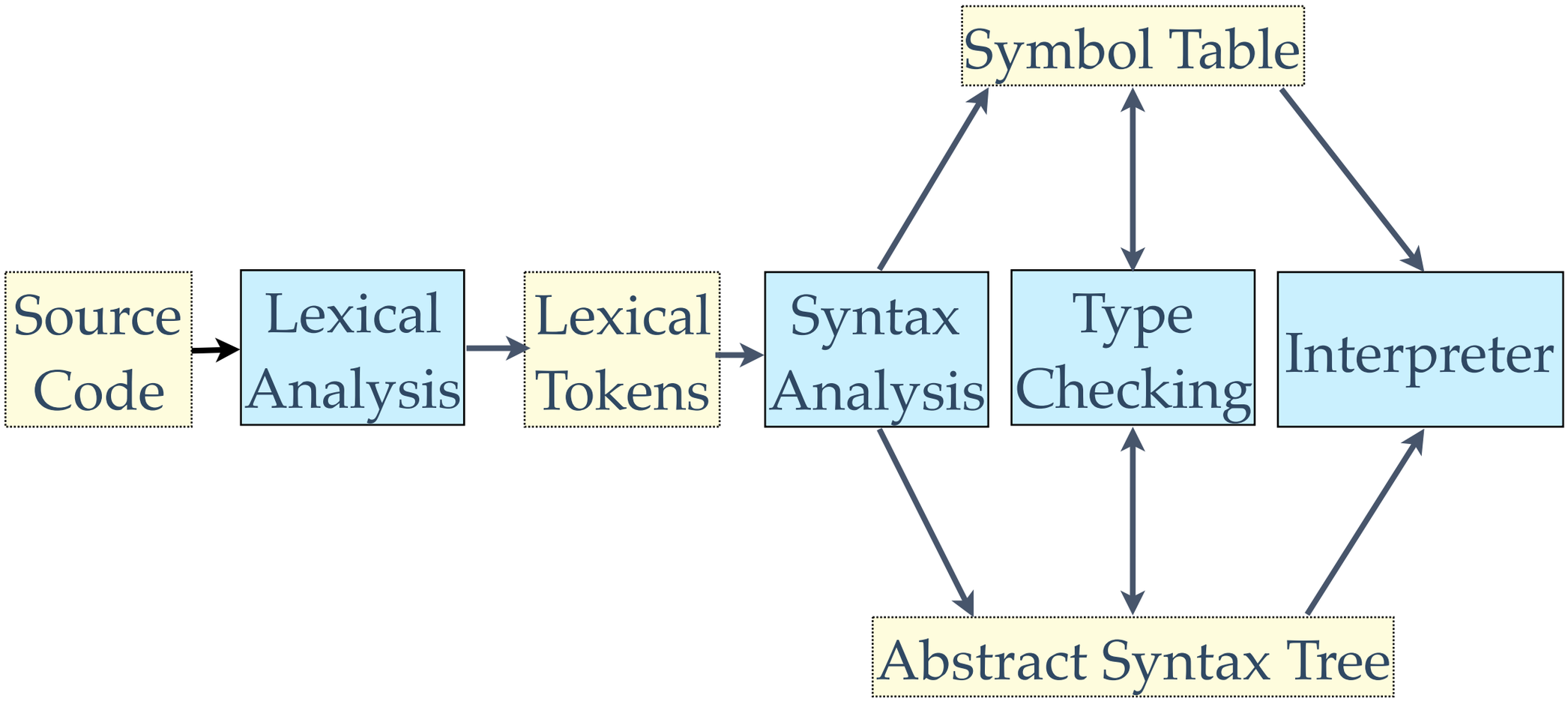
Ian Hayes

14 February 2019

Phases of a compiler



Phases of a interpreter



Lexical Analysis

- Input: a sequence of characters representing a program
- Output: a sequence of lexical tokens
- Lexical tokens: identifiers, numbers, keywords (e.g., “if”, “while”), symbols (e.g., “+”, “<=”)
- Ignores white space: blanks, tabs, newlines, carriage returns, form feeds
- Comments: treated as white space

Syntax Analysis

- Input: a sequence of lexical tokens
- Output: an abstract syntax tree and symbol table
- Symbol table
  - contains information about all identifiers that are defined within the program (plus a few predefined ones)
  - may be organised into scopes, e.g, identifiers defined within a procedure

Type Checking
a.k.a. Static Semantic Analysis

- Input: Symbol table and abstract syntax tree
- Output: Updated symbol table and abstract syntax tree
- Resolves all references to identifiers
  - updates symbol table entries with type information
  - checks abstract syntax tree for type correctness
  - updates abstract syntax tree with type coercions

Code Generation

- Input: Symbol table and (updated) abstract syntax tree
- Output: code for the target machine
- May include
  - machine-independent optimisations
  - machine-dependent optimisations
  - instruction selection
  - register allocation

Interpreter

- Input: Symbol table and (updated) abstract syntax tree
- Interprets the abstract syntax tree directly to execute the program
  - the program being interpreted may have inputs and outputs
- Less time compiling (no code generation)
- Slower to execute the program
- More commonly used for high-level dynamically typed languages