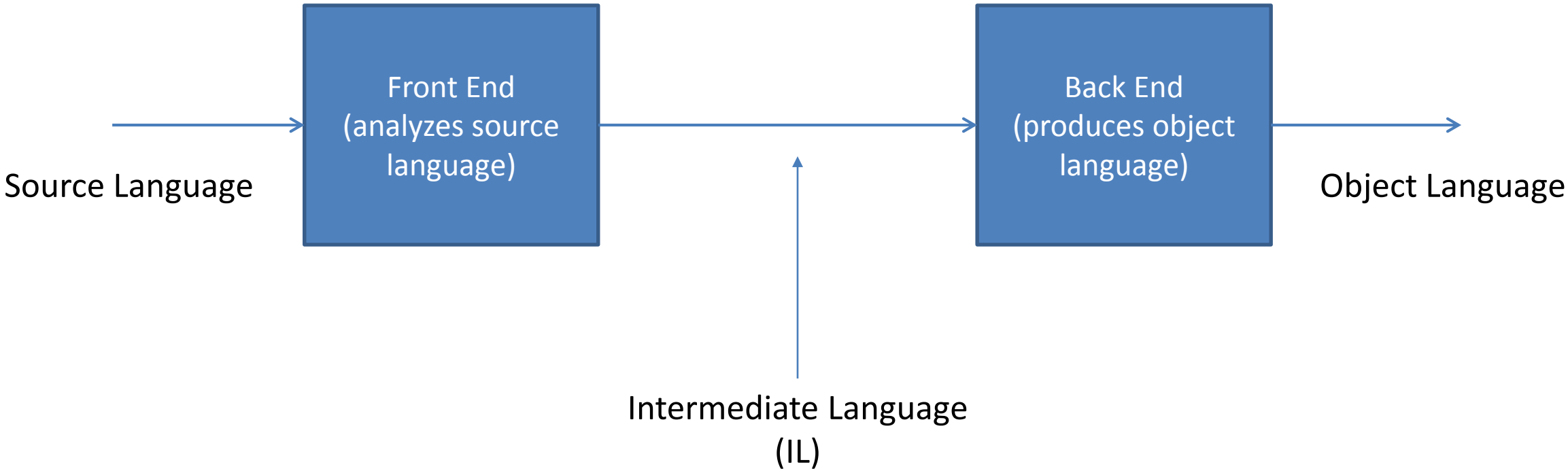
Compiler Design

Vermont Technical College
Peter C. Chapin

Why Study Compilers?

- Somebody has to write them. Why not you?
- *Know your tools!*
  - Compilers are fundamental. The more you know about them, the more effectively you can use them.
- Domain Specific Languages
  - Compiler technology useful in surprising ways.
    - Processing configuration and command languages.
    - File format transformations.

The Pipeline



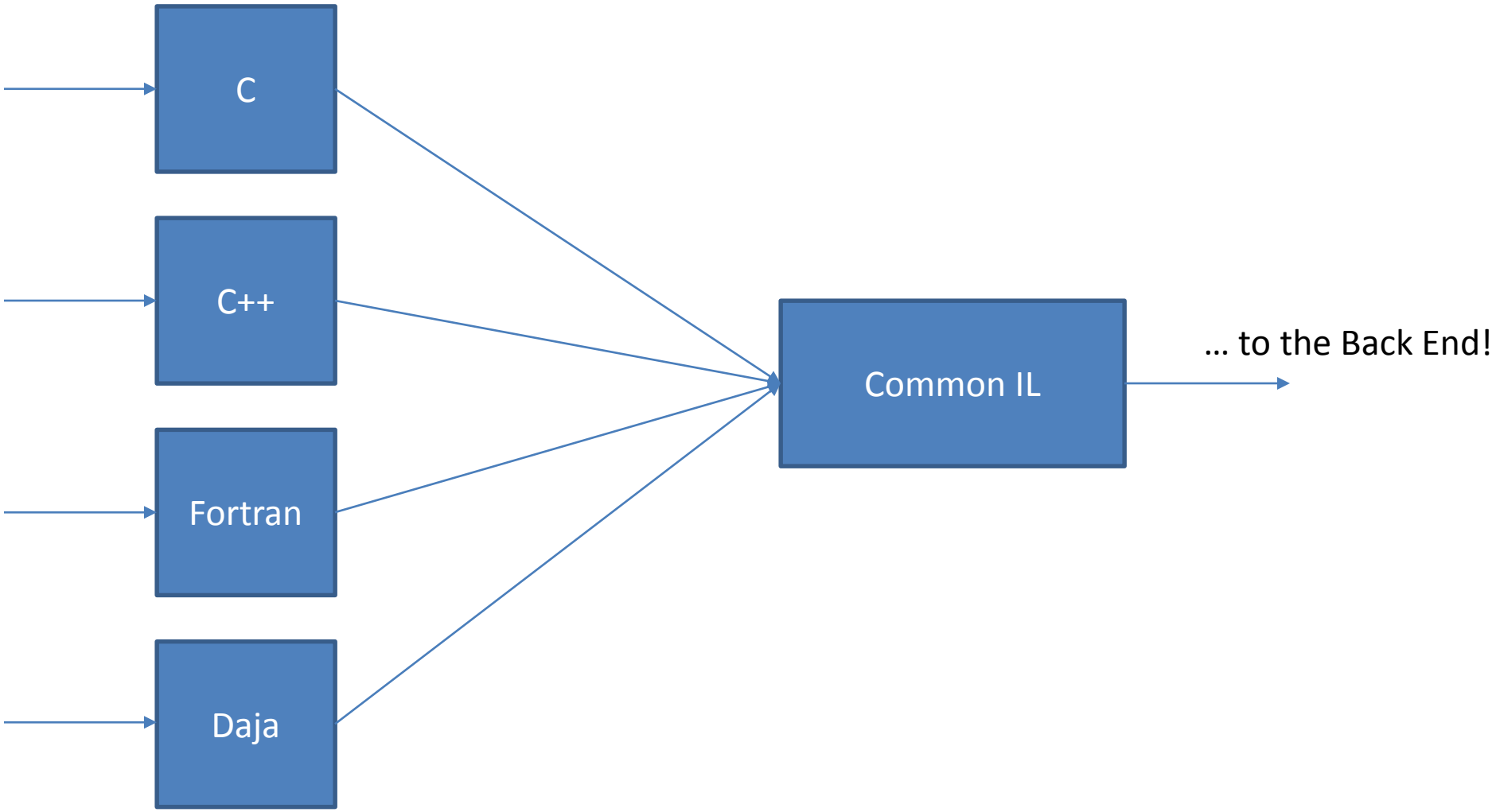
Language Levels

- Source Language
  - Usually high level, abstract.
  - e. g., C, Java, etc
- Object Language
  - Usually low level, concrete.
  - e. g., Machine language, assembly language, C?
- Intermediate Language
  - Easy for the front end to produce, easy for the back end to consume

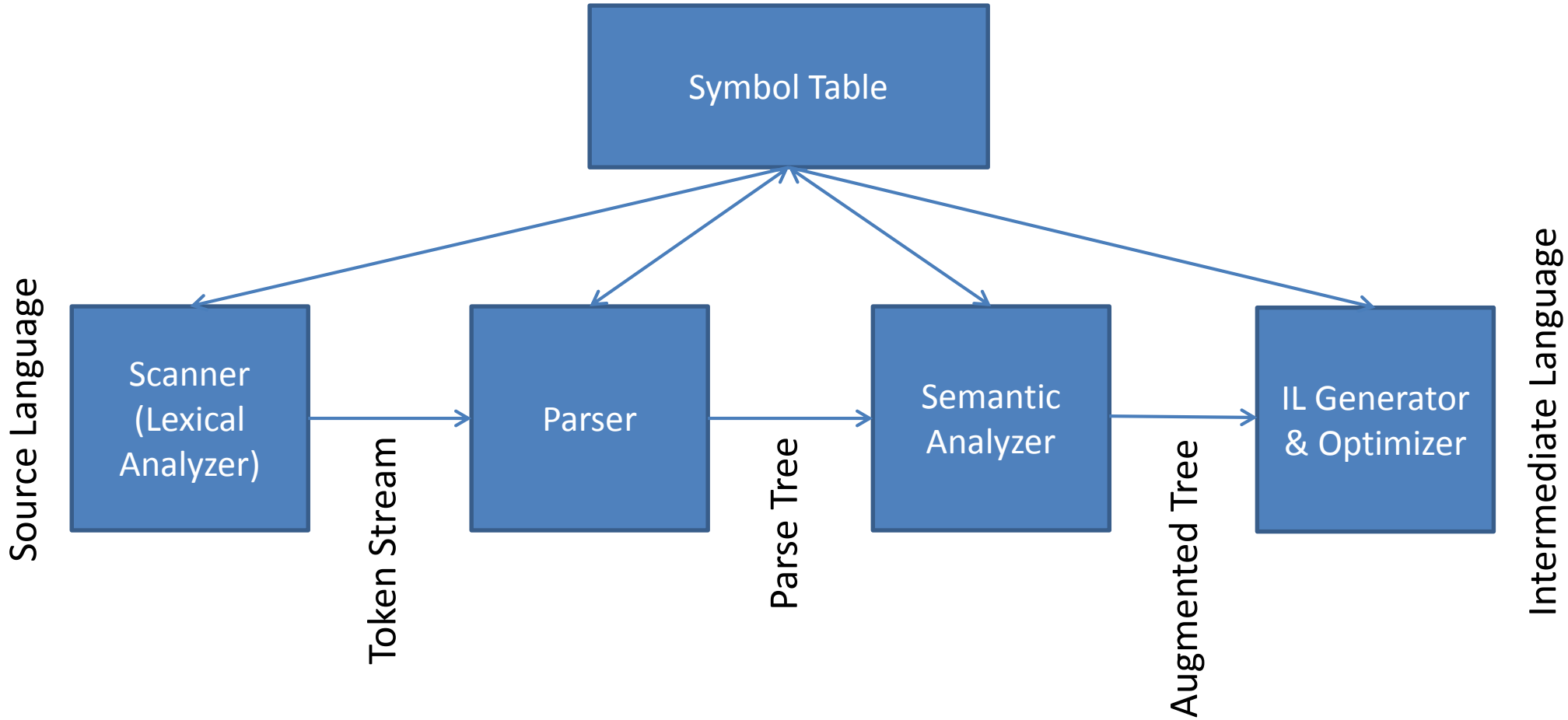
Front End

- The front end “knows” the source language
  - To compile different source languages, change the front end. As long as they all generate the same IL they can use a common back end.
  - gcc. The “GNU Compiler Collection”
    - C, C++, Objective-C, Fortran, Java, Ada, Go
    - All use the same back end (in theory).
  - Open Watcom
    - C, C++, FORTRAN
    - All use the same back end.

Multiple Front Ends



Front End Pipeline



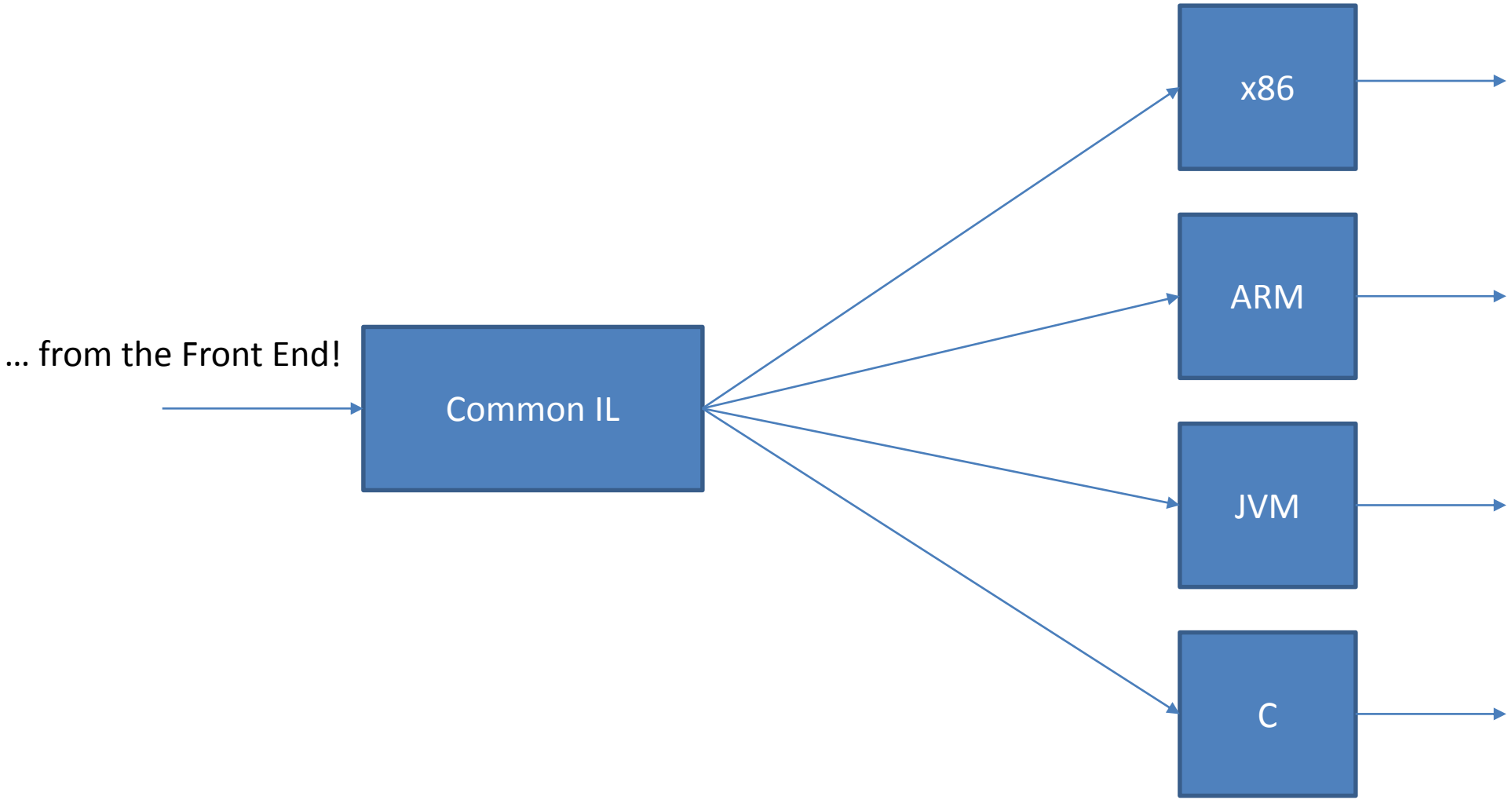
Lexical Analysis

- Consider the declaration "`int x = 2*a + 1;`"
  - Tokens:
    - **`int`**, `IDENTIFIER`, `=`, `2`, `*`, `IDENTIFIER`, `+`, `NUMERIC_LITERAL`, `;`
    - The IDENTIFIER tokens have *attributes* of "x" and "a"
    - The NUMERIC_LITERAL token has an attribute of "1"
  - All tokens have attributes specifying position in source file
    - … so that good error messages can be produced later
  - Comments and whitespace not in the token stream (typically)
  - No attempt to verify the structure of the program (yet!)

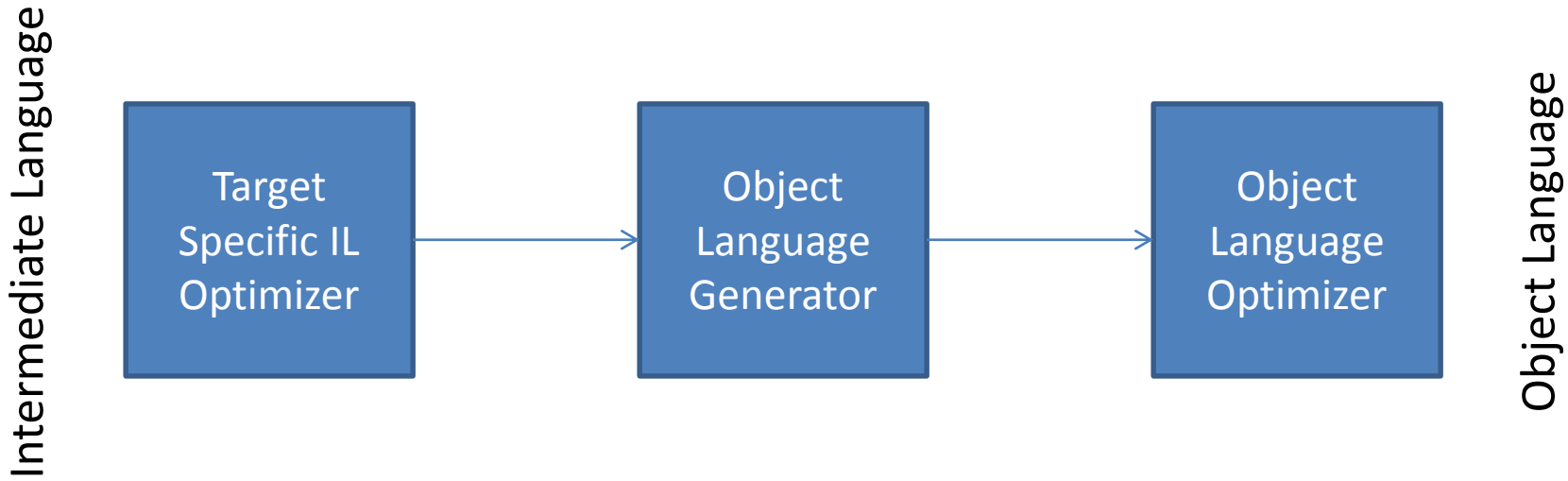
Back End

- The back end “knows” the object language
  - To target different systems, change the back end. As long as they all accept the same IL, they can use a common front end.
  - gcc
    - Supports multiple targets by providing a separate back end for each. All source languages work on all targets (in theory).
  - Open Watcom
    - Targets 16 and 32 bit x86 processors, Alpha, MIPS, x64 under development… using multiple back ends.

Multiple Back Ends



Back End Pipeline



Implementation Language

- Common to write compilers in the language they compile
  - Compiler writers are experts in “their” language… want to use it too!
  - Easier to get help from the community
  - Compiler can compile itself; no dependency on another product
- … but it is not necessary!
  - First compiler for a language obviously can’t be in that language!
  - Some languages are more suitable for compilers than others
  - Community and cultural reasons may dictate the choice of implementation language

Course Organization

- The classic compiler pipeline suggests…
  - Starting with scanners and working our way from left to right in the previous diagrams!
  - The book does this also
    - I think all compiler texts do.
- *I'm going to do something slightly different*…
  - Build a simplistic system from start to finish
  - Go back and enhance it to add features exploring other aspects of compiler design.

Write a Compiler?

- Yes!
  - It is traditional for students to write a small compiler for a simple language in a course like this.
  - The difference is that we'll explore the whole pipeline early and study parts in more depth as we need them (and as time allows).
  - We'll use Java as our implementation language
  - To start we'll use either the JVM or LLVM as a back end.
    - … depending on our whim at the time!
    - Both are pre-built for us and very advanced and flexible.

Daja

- *Daja* is a simplified subset of the D programming language
  - Suitable for education: difficult features of D are deleted.
- D?
  - D is a language created by Walter Bright intended to compete with C++ but with nicer/easier-to-use features.
  - See the D language web page for more information

Let’s get started!