Architecture of Large Compilers

Tech Talk #02

Charles Averill and Jack Myrick

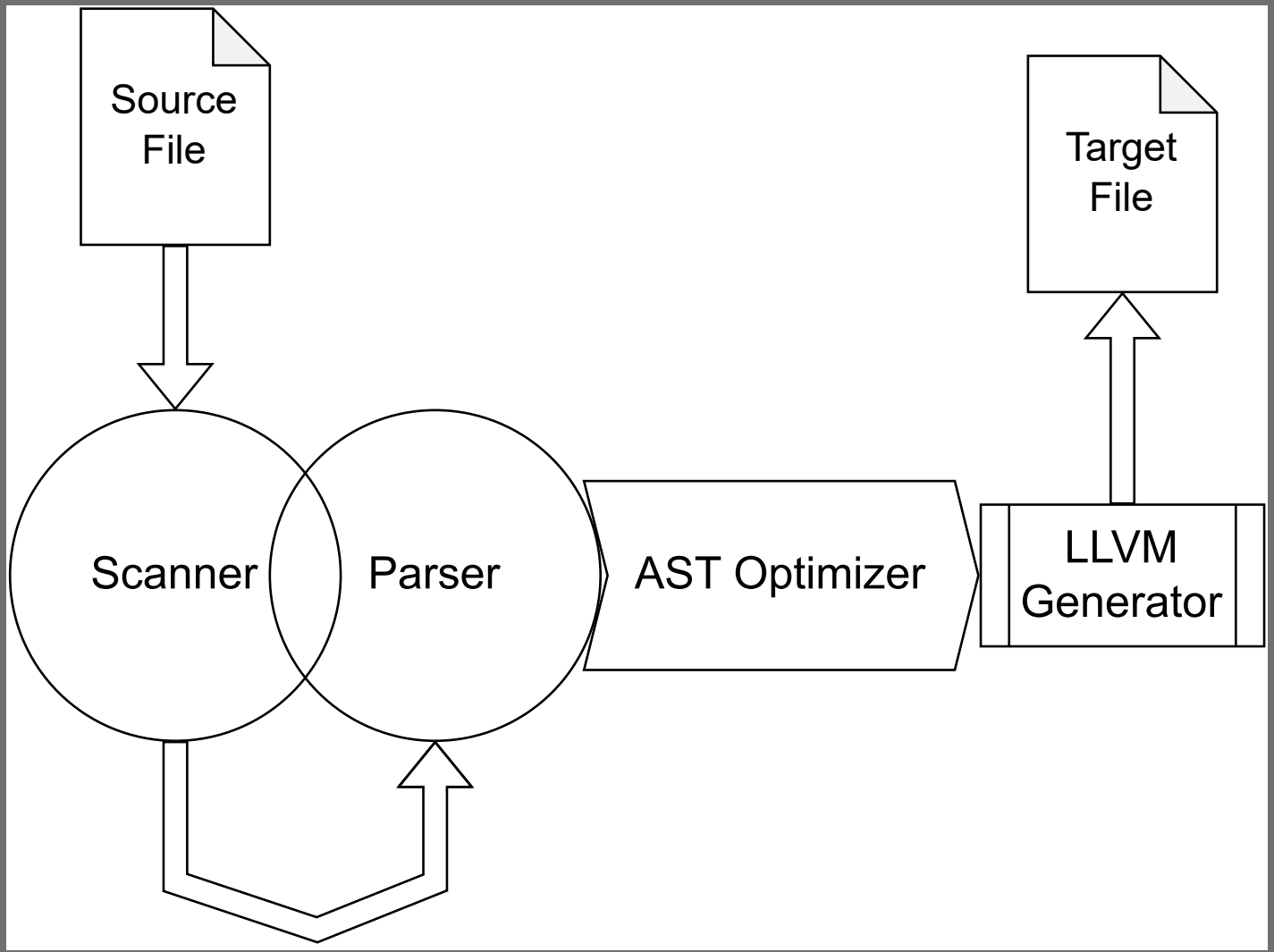
Introduction to Compiler Design
The University of Texas at Dallas

Spring 2023

Overview

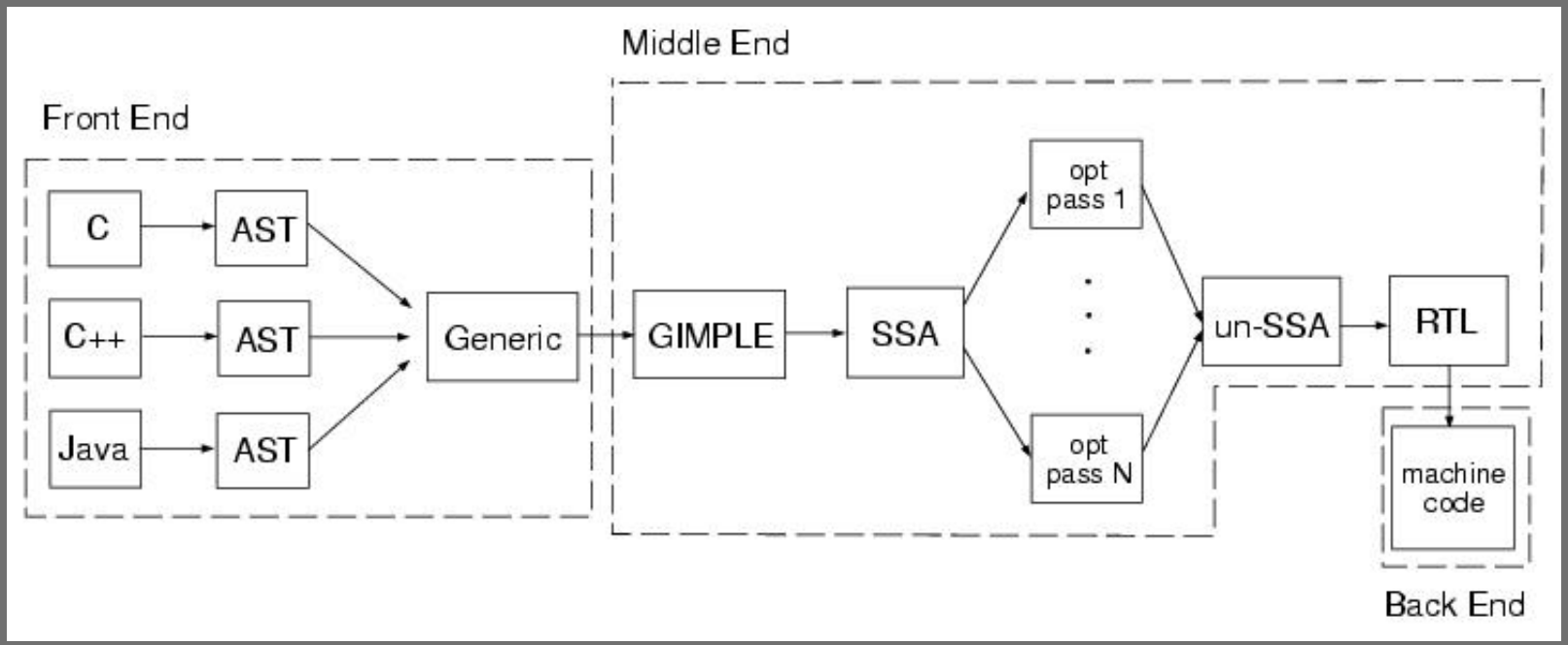
- Our compiler, ECCO, is a fairly simple compiler. An accurate description would be that it is a "single-pass optimizing C to LLVM compiler"
- ECCO works - it translates its source language to its target language and (hopefully) preserves semantic equivalence
- ECCO-generated code is not fast by default, it's essentially a 1:1 mapping of high-level code to low-level code
- Production compilers generate very fast code. What about them is different?

ECCO Overview



gcc

- Fun fact: what do you think GCC stands for?
- We use it as a C compiler



GIMPLE

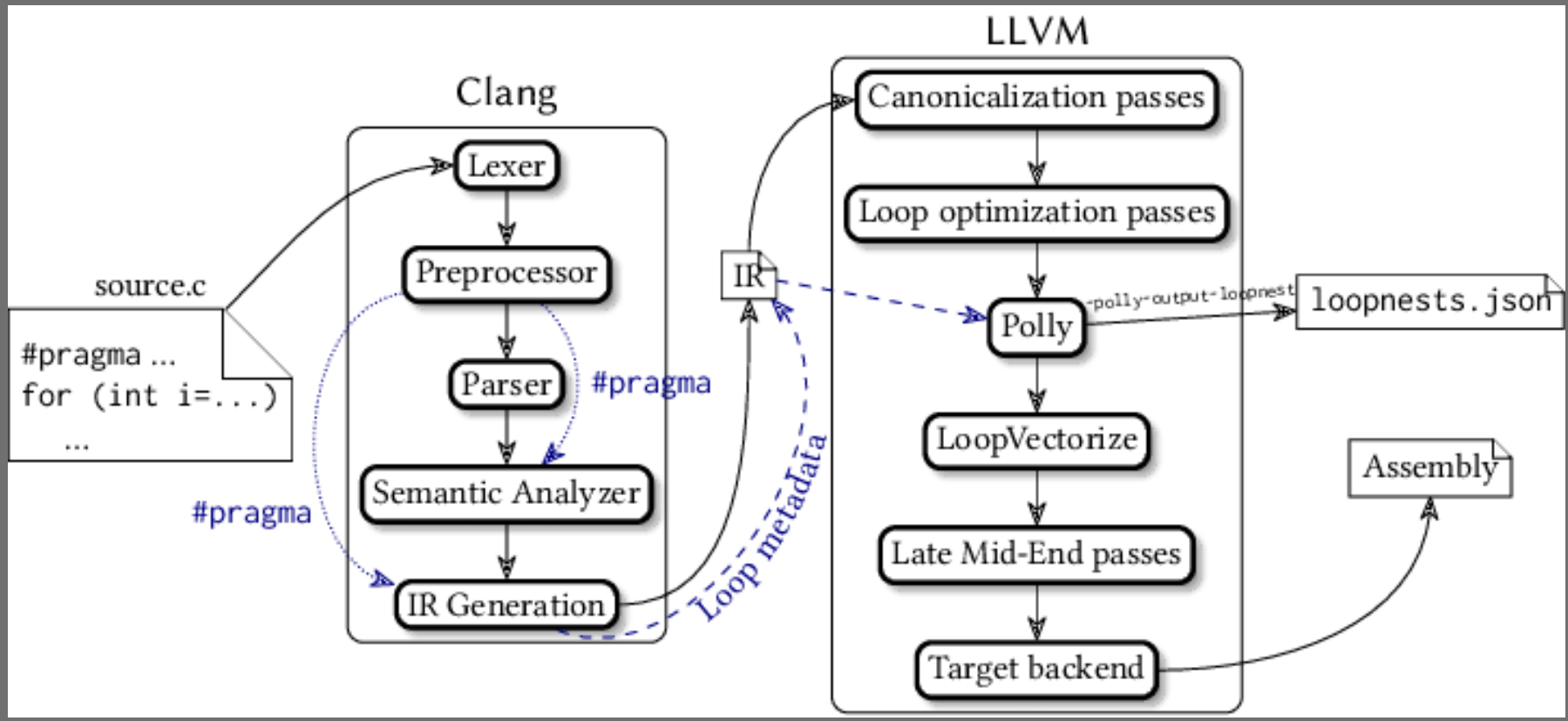
- GIMPLE is one of GCC's intermediate representations
- Expressions have at most 3 operands
- All control flow is turned into GOTO and conditional statements
- Gets converted to SSA (same idea as LLVM-IR), is optimized, back to GIMPLE, then into RTL
  - RTL is similar to LLVM-IR - also an abstraction of CPU hardware
  - Type system is much smaller - just integers and floats
  - Less debuggable than LLVM-IR

Gimple Example

RTL Example

GCC Internals Wiki Page

clang



We're implementing a simple clang!

Java

- Java is a high-level OOP language designed to write once, run anywhere
- To accomplish this, Java bytecode runs in the Java Virtual Machine (JVM) to abstract computer architecture details away
- The JVM has four registers:
  1. PC: program pointer, points to a position in the program store
  2. VARS: all local variables are addressed relative to this register
  3. OPTOP: points to the topmost cell of the operand stack
  4. FRAME: points to the first cell of the execution environment

JVM

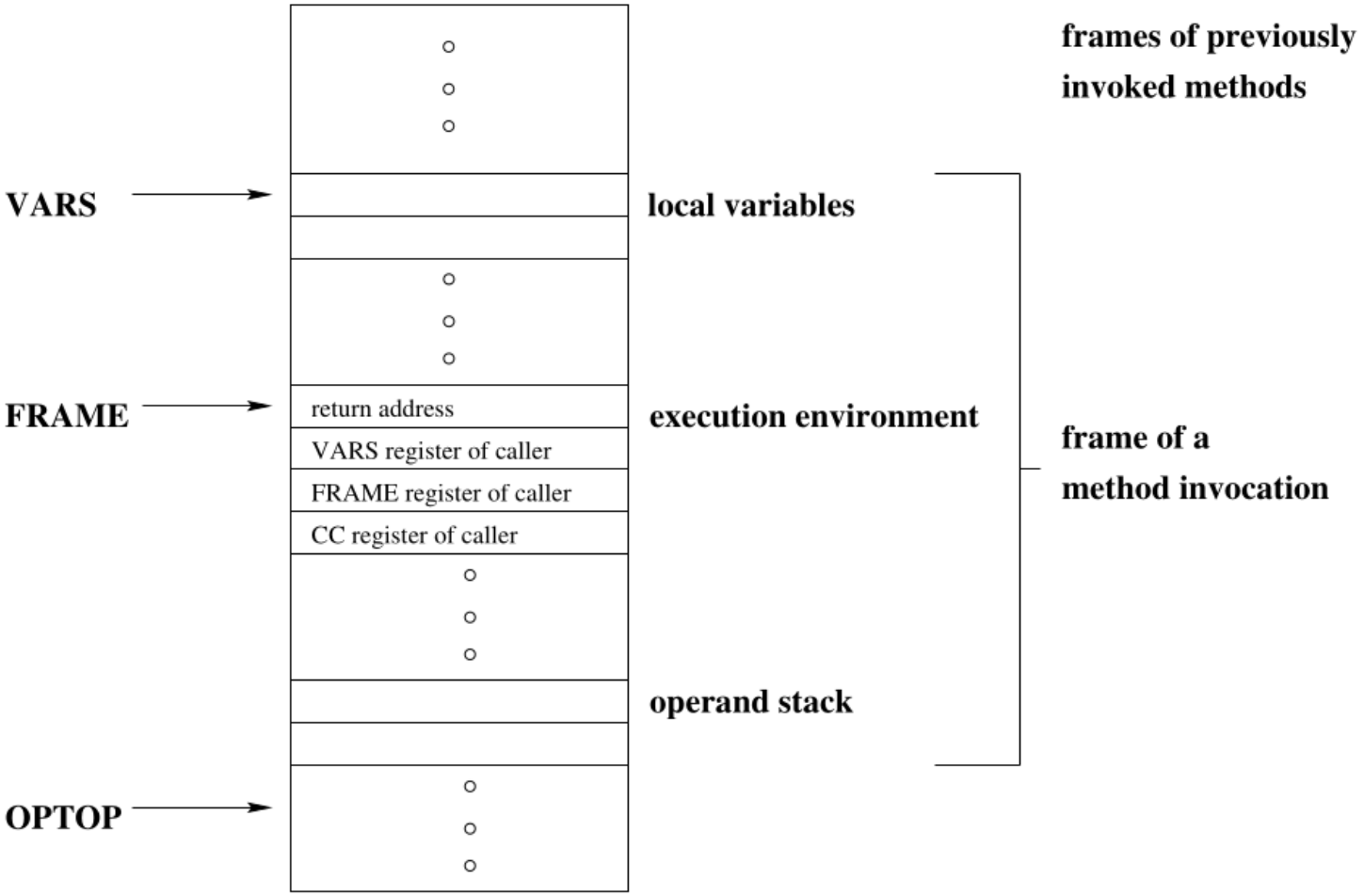


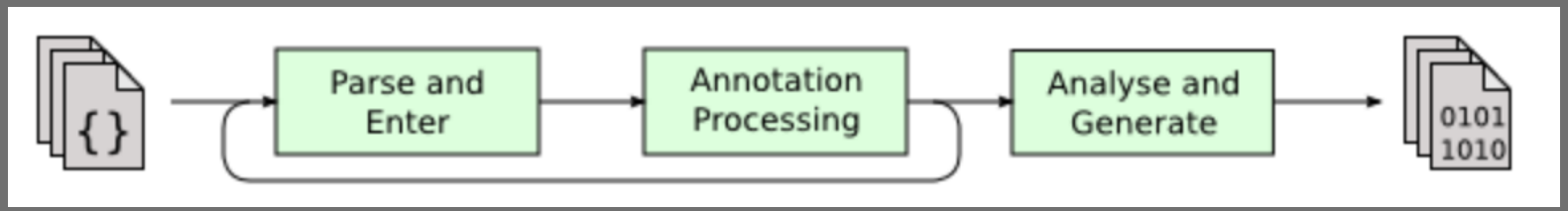
Figure 1. Registers and stack

javac

- Java's primary compiler is javac, and is included in the Java Development Kit (JDK)
- javac compiles Java source code into Java bytecode, which the JVM understands when running
- javac is itself written in Java and its source is available under the GNU General Public License as part of OpenJDK

Compilation Overview

- javac takes in .java files and outputs bytecode as .class files
- Divided into three stages:
  1. Parse and Enter
  2. Annotation Processing
  3. Analyse and Generate



javac Architecture

Parse and Enter

- "All the source files specified on the command line are read, parsed into syntax trees, and then all externally visible definitions are entered into the compiler's symbol tables." (OpenJDK Compilation Overview)
- Source files converted into a stream of tokens by the Scanner
- The Parser reads this stream to create syntax trees using a TreeMaker
- Each tree is passed to Enter, which enters class and member declarations into the symbol table

Annotation Processing

- "All appropriate annotation processors are called. If any annotation processors generate any new source or class files, the compilation is restarted, until no new files are created." (OpenJDK Compilation Overview)
- "Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate." (Java Annotations Tutorial)
- Uses for annotations:
  1. Information for the compiler
  2. Compile-time and deployment-time processing
  3. Runtime processing
- Examples: @Override, @Deprecated, @SuppressWarnings

Analyse and Generate

- "Finally, the syntax trees created by the parser are analyzed and translated into class files. During the course of the analysis, references to additional classes may be found. The compiler will check the source and class path for these classes; if they are found on the source path, those files will be compiled as well, although they will not be subject to annotation processing." (OpenJDK Compilation Overview)
- The work is done by a series of visitors
  - Attr - top level classes are "attributed" (names, expressions, etc. resolved and associated with types or symbols)
  - Flow - flow analysis for unreachable statements and assignment
  - TransTypes - translates generic types to standard types
  - Lower - processing of syntactic sugar
  - Gen - generation of code for methods using bytecode

Further Reading

The Hitchhiker's Guide to javac

The Hacker's Guide to Javac

A Formal Introduction to the Compilation of Java

The Java Language Specification

The Java Virtual Machine Specification

TypeScript

- TypeScript is a superset of JavaScript that adds optional static typing
- TypeScript transpiles to JavaScript

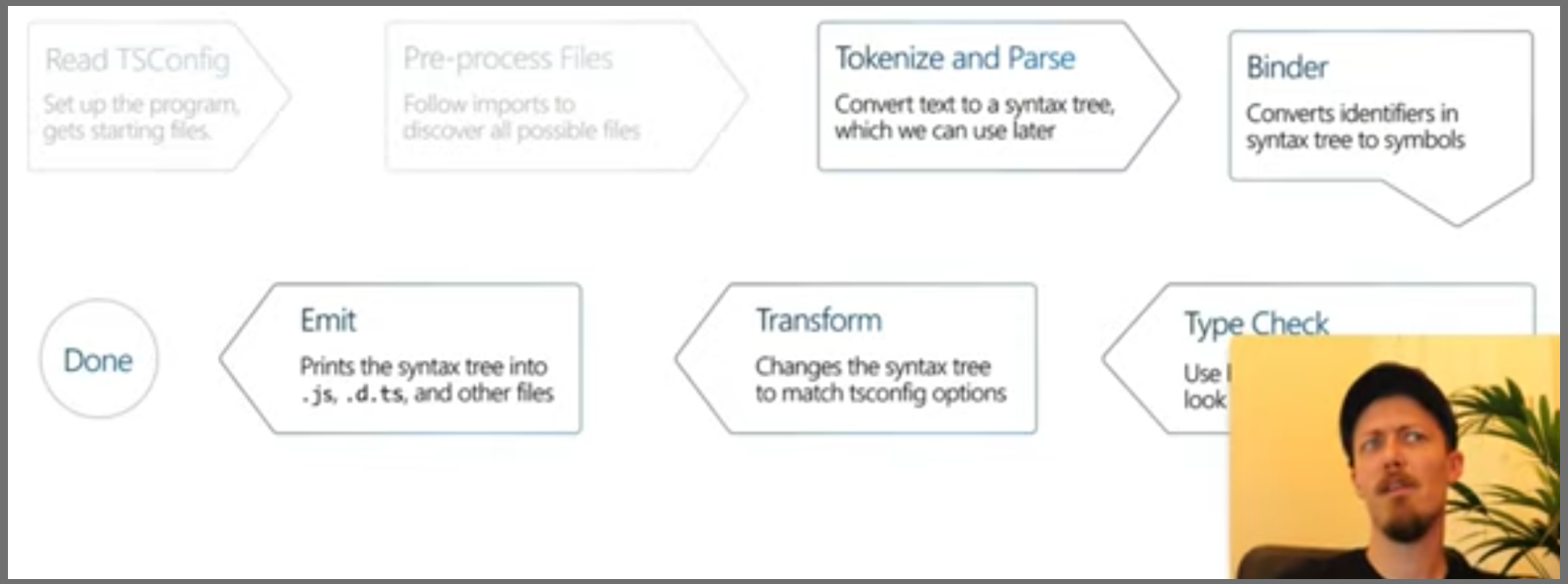
tsc

- TypeScript's primary compiler is tsc
- tsc compiles TypeScript source code into JavaScript source code, which can be run in the browser by a JavaScript engine like V8 or in a JavaScript runtime like Node, Deno, or Bun
- tsc is itself written in TypeScript and its source is published on GitHub

Compilation Overview

- I found my information through this repo. The video linked there is a good watch.
  1. Preprocessing for files to include through imports or requires
  2. AST Node Generation
  3. Binder binds Symbols (one for each named entity) with scopes
  4. Generating a SourceFile along with its Symbols
  5. Build a global view of all files in the compilation as a Program
  6. TypeChecker assigns Types to Symbols along with generating semantic Diagnostics
  7. Emitter outputs as .js

Compilation Architecture Summary



Summary of Compilation Process