SWEN430 - Compiler Engineering

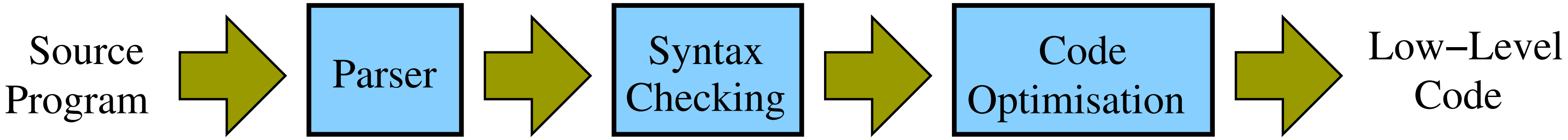
Lecture 1 - Introduction

Lindsay Groves

with thanks to David Pearce

School of Engineering and Computer Science
Victoria University of Wellington

What is a Compiler?



- Compilers translate **source programs** into **low-level code**
  - recognise program structure
  - check for certain errors (e.g. syntax errors, type errors)
  - optimise the program where possible
  - generate “low-level” code (VM code or machine code)
- Examples (all subject to active research and improvement):
  - Javac (translates Java into Java bytecode)
  - Microsoft Visual C#/C++/F#/VB (translates into .NET IL)
  - GCC (e.g. translates C/C++ into x86)
  - GHC (translates Haskell into x86)

Compiling Java

Test.java

```
class Test {
    public static void main(String[] args) {
        System.out.println("Hello␣World"); } }
```

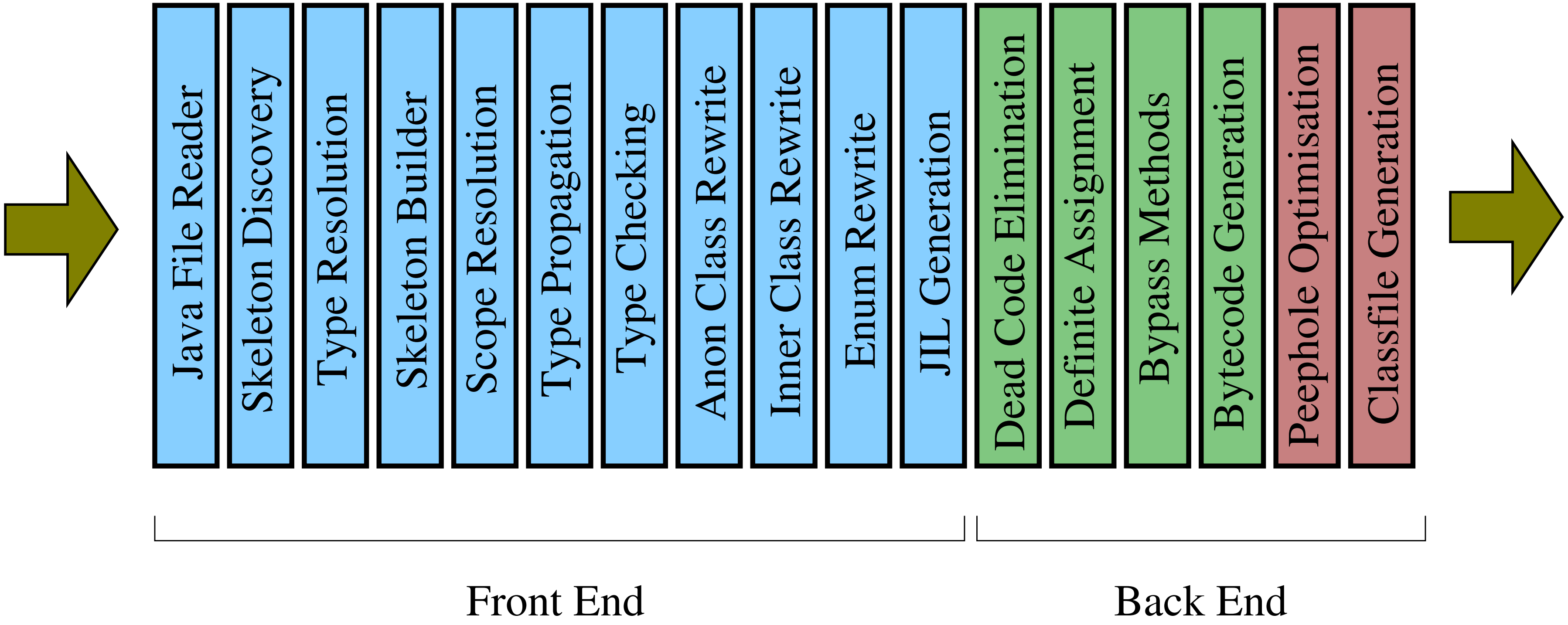
javap -verbose Test

```
Compiled from "Test.java"
class Test extends java.lang.Object
...
public static void main(java.lang.String[]);
 Code:
 Stack=2, Locals=1, Args_size=1
 0: getstatic     #2; //Field java/lang/System.out
 3: ldc           #3; //String Hello World
 5: invokevirtual #4; //Meth java/io/PrintStream.println
 8: return
}
```

Compiling Java

- Java Language Specification:
  - Details what is **syntactically correct** Java code
  - Details how Java code **should execute**
  - `http://docs.oracle.com/javase/specs/jls/se7/html/index.html`
- Java Virtual Machine Specification:
  - Details what is **syntactically correct** Java Classfile
  - Details how Java bytecodes **should be executed**
  - `http://docs.oracle.com/javase/specs/jvms/se7/html/index.html`

JKit Java Compiler



Java File Reader

Skeleton Discovery

Type Resolution

Skeleton Builder

Scope Resolution

Type Propagation

Type Checking

Anon Class Rewrite

Inner Class Rewrite

Enum Rewrite

JIL Generation

Dead Code Elimination

Definite Assignment

Bypass Methods

Bytecode Generation

Peephole Optimisation

Classfile Generation

Front End

Back End

- Previously developed at VUW by David J. Pearce
- Used for research, teaching and fun!
- Currently has **90 classes** ($\sim$79 KLOC) and **287 JUnit tests**

Compiling Whiley

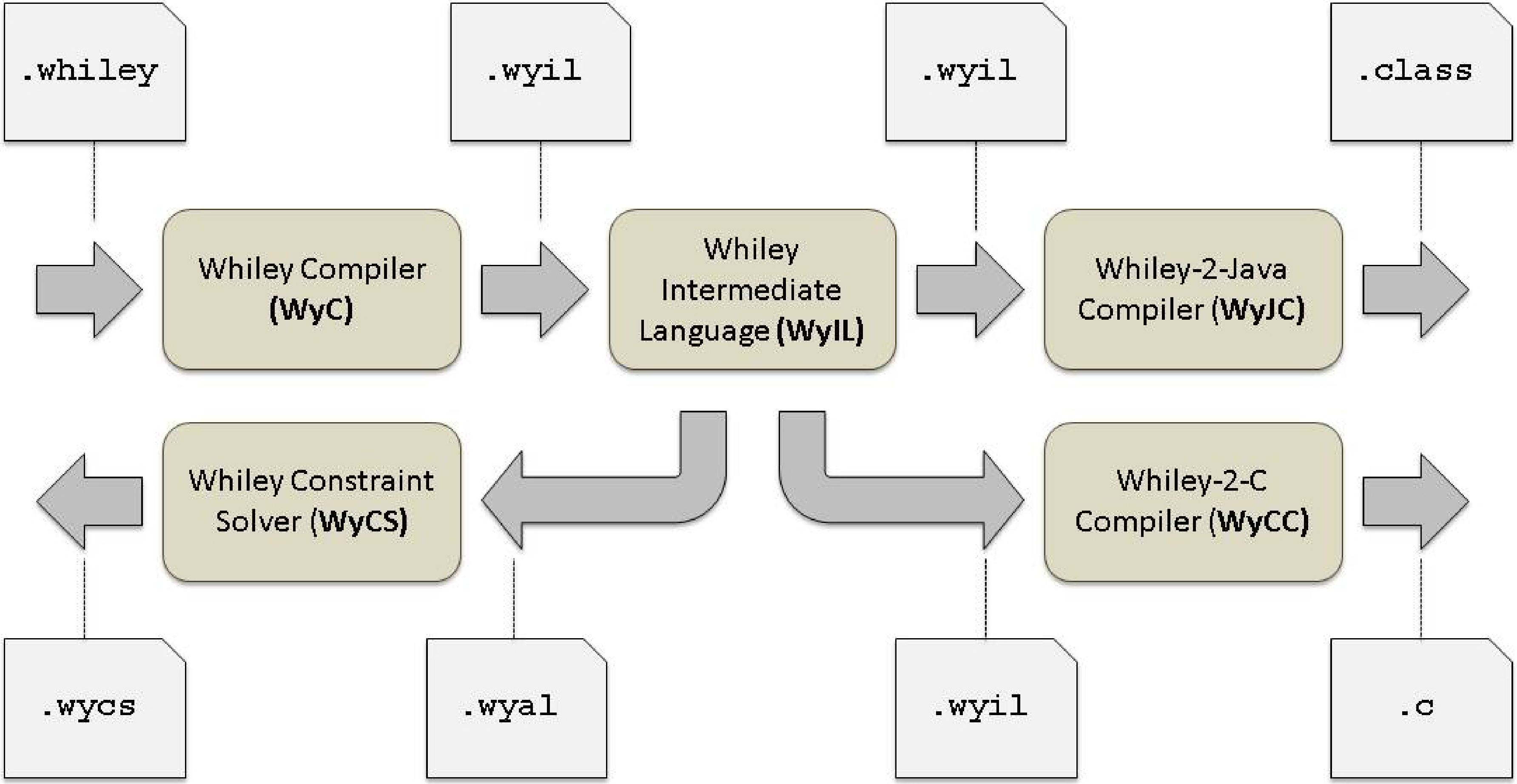
test.whiley

```
type nat is (int n) where n >= 0

function sum(nat[] xs) → nat:
    int r = 0
    int i = 0
    while i < |xs| where i >= 0 && r >= 0:
        r = r + xs[i]
    return r
```

- Currently developed at VUW by David J. Pearce
- Currently, **106 KLOC**, spread over **270 classes**
- See: `http://whiley.org`, `http://www.ohloh.net/p/whiley`

Compiling Whiley



.whiley
.wyil
.wyil
.class
Whiley Compiler (WyC)
Whiley Intermediate Language (WyIL)
Whiley-2-Java Compiler (WyJC)
Whiley Constraint Solver (WyCS)
Whiley-2-C Compiler (WyCC)
.wycs
.wyal
.wyil
.c

Other PL/compiler design projects at VUW

- Wyvern
  - Collaboration between CMU (Jonathan Aldrich) and VUW (Alex Potanin)
  - `http://www.cs.cmu.edu/~aldrich/securemobileweb/spec-rationale.html`
  - `https://github.com/wyvernlang`
- Grace
  - Collaboration between VUW (James Noble, Michael Homer), Portland State (Andrew Black) and Pomona College (Kim Bruce)
  - `http://gracelang.org/`

The While Language — simplified version of Whiley

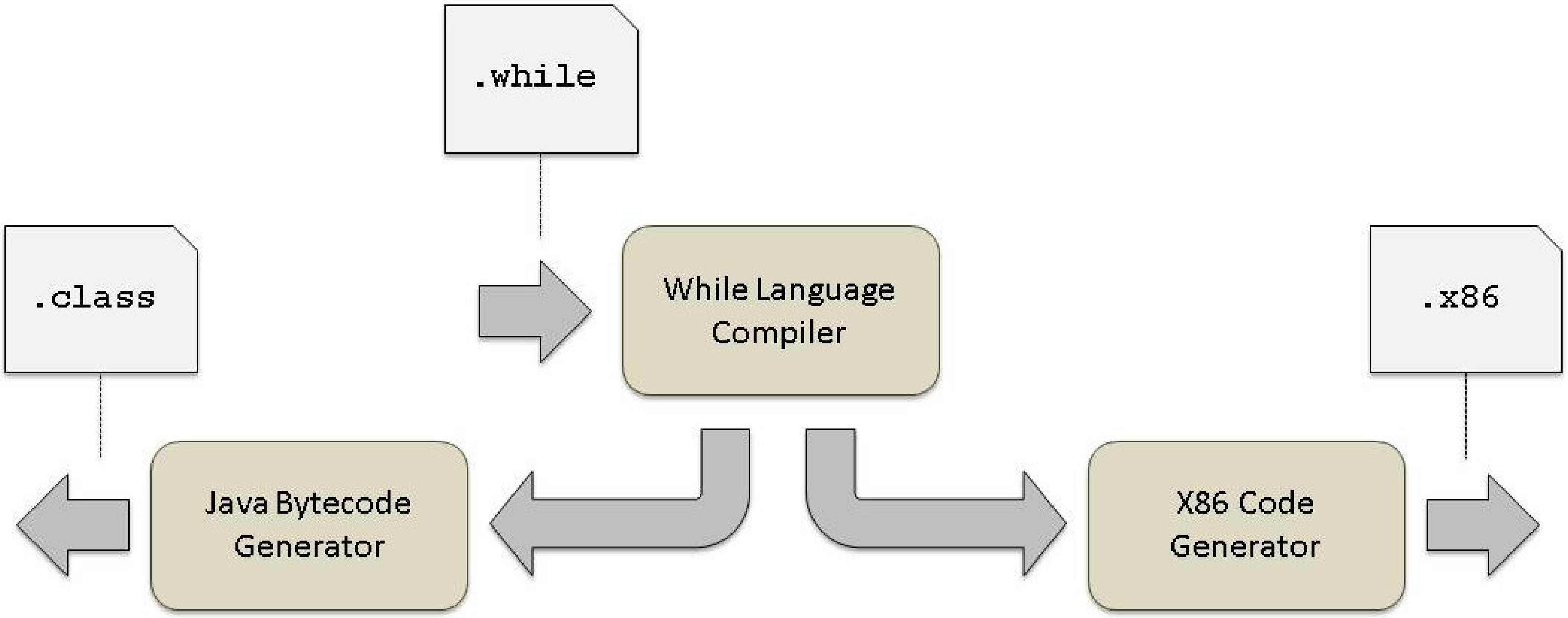
test.while

```
type Point is {int x, int y}

Point move(Point p, int dx, int dy) {
  return {x: p.x + dx, y: p.y + dy};
}
```

- A **simple** imperative language
- **Statements**: for, while, if, switch
- **Expressions**: binary, unary, invocation
- **Types**: bool, int, strings, arrays, records

While Language Compiler



- Lack of modules / imports will **simplify internals**
- **No intermediate language**: code generation directly off AST
- Targets: **Java Bytecode** and **x86 Assembly Language**

Compilers and Interpreters

- A compiler translates programs into machine code that can be executed directly on hardware

- An interpreter executes programs directly without translating into matching code

- Actually, an interpreter usually translates the program into some form of intermediate language, which is then interpreted

- And a compiler may translate programs into a form of virtual machine code, which may then be translated into machine code, executed by an emulator (i.e. interpreted), or executed directly by hardware/firmware.

Compilers and Interpreters

- Both need to parse the program and do some analysis on identifiers, types, etc.
- Both translate to some lower level form — what that is may differ considerably
- Often differ in when/how errors are detected

Course Organisation

- **Lectures**
  - Tuesday and Thursday 1:10pm to 2:00pm in OK524
- **People**
  - A/Prof Lindsay Groves (course coordinator and lecturer)
    Co257, 463 5656, `lindsay@ecs.vuw.ac.nz`
  - Dr David J. Pearce (guest lecturer/adviser)
    Co231, 463 5833, `djp@ecs.vuw.ac.nz`

Lecture Topics (tentative)

- Course introduction (1)
- Compiler structure (1)
- Parsing (2)
- Type checking (3)
- Static analysis (3)
- Code generation: Java bytecode (3)
- Code generation: x86 machine code (4)
- Register allocation and code optimisation (2)
- Miscellaneous topics: readings (3)

Assessment (tentative!)

- **Assignment 1 (10%)** — Parsing and Interpretation
- **Assignment 2 (10%)** — Type Checking
- **Assignment 3 (10%)** — Java Bytecode
- **Assignment 4 (10%)** — x86 Machine Code
- **Exam (60%)** — 2 hours
- Mandatory Requirements
  - *At least 40% average across four assignments*
  - *At least 40% on exam*
- Late Penalties:
  - Late work will be penalised 10% per weekday after the deadline
  - Each student has three "late days"

Recommended Books

- There is **no set text**, but the following are recommended:
  - *Modern Compiler Implementation in Java*, Andrew Appel. (closed reserve, and on-line)
  - *Engineering a Compiler*, Keith D. Cooper and Linda Toczon. See Chapter 8. [1 copy in library]
  - *Compilers: Principles, Techniques and Tools*, Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman. See Chapter 10. [1 copy in library]
  - *Advanced Compiler Design and Implementation*, Steve S. Muchnick. See Chapter 9.
  - *Optimizing Compilers for Modern Architectures*, Randy Allen and Ken Kennedy. See Chapter 4.4 and 11.
- Other good books available on-line (see course web site)
- Many lecture notes, tutorials etc on-line

Why SWEN430?

Why should I take SWEN430?

- Learn **how compilers work**
- Create **a working compiler for a realistic imperative language**
- Learn **Java Bytecode** and **x86 Assembly**
- Improve your **programming skills**
  - Working with a large complex code base
  - Learn to use programming languages more intelligently
  - Learn techniques you can use, e.g. to implement **DSLs**
- Understand aspects of **programming language design**

Class Representative Election!!

What to do now ...

- Read the Wikipedia article on compilers
- Download Appel's book (or any other compiler book) and read chapter 1.
- Download the While compiler code and compile it
- Review the COMP261 notes on parsing