Introduction

Matthew Dwyer
365 Avery Hall
dwyer@cse.unl.edu
http://www.cse.unl.edu/~dwyer

Purpose

- The course will teach modern compiler techniques applied to general-purpose programming languages.

- The examples and project will also convey a detailed knowledge of state-of-the-art language-processing technologies.

Me

- Worked in the compiler industry for 6 years
  - Developed parts of three compiler products
- Have personally implemented 5 complete compilers
- Have taught compiler courses 14 times
- Lead research in areas that are closely related to compiler technology
- Have become nice in my old age
  - 5000 SLOC … 1000 SLOC

Contents

- Deterministic parsing: Parsing and the ANTLR tools.
- Semantic analysis: abstract syntax trees, symbol tables, type checking, resource allocation.
- Virtual machines and run-time environments: stacks, heaps, objects.
- Code generation: resources, templates, optimizations.
- Surveys on: garbage collection, native code generation, static analysis.

Schedule

- Lectures: 2 per week
- Office Hours: Wed. 11:00am-noon, 365 Avery
- email: Feel free to ask questions whenever you like

Grading

- Projects:
  - Equal credit spread across 5 milestone (50%)
  - Grad student milestone (rescale, 10%)
  - Extra credit for undergrads (10%)
- Quizzes:
  - 5 short quizzes during semester (25%)
- Final Exam:
  - 2 hours - cumulative (25%)

Course Materials

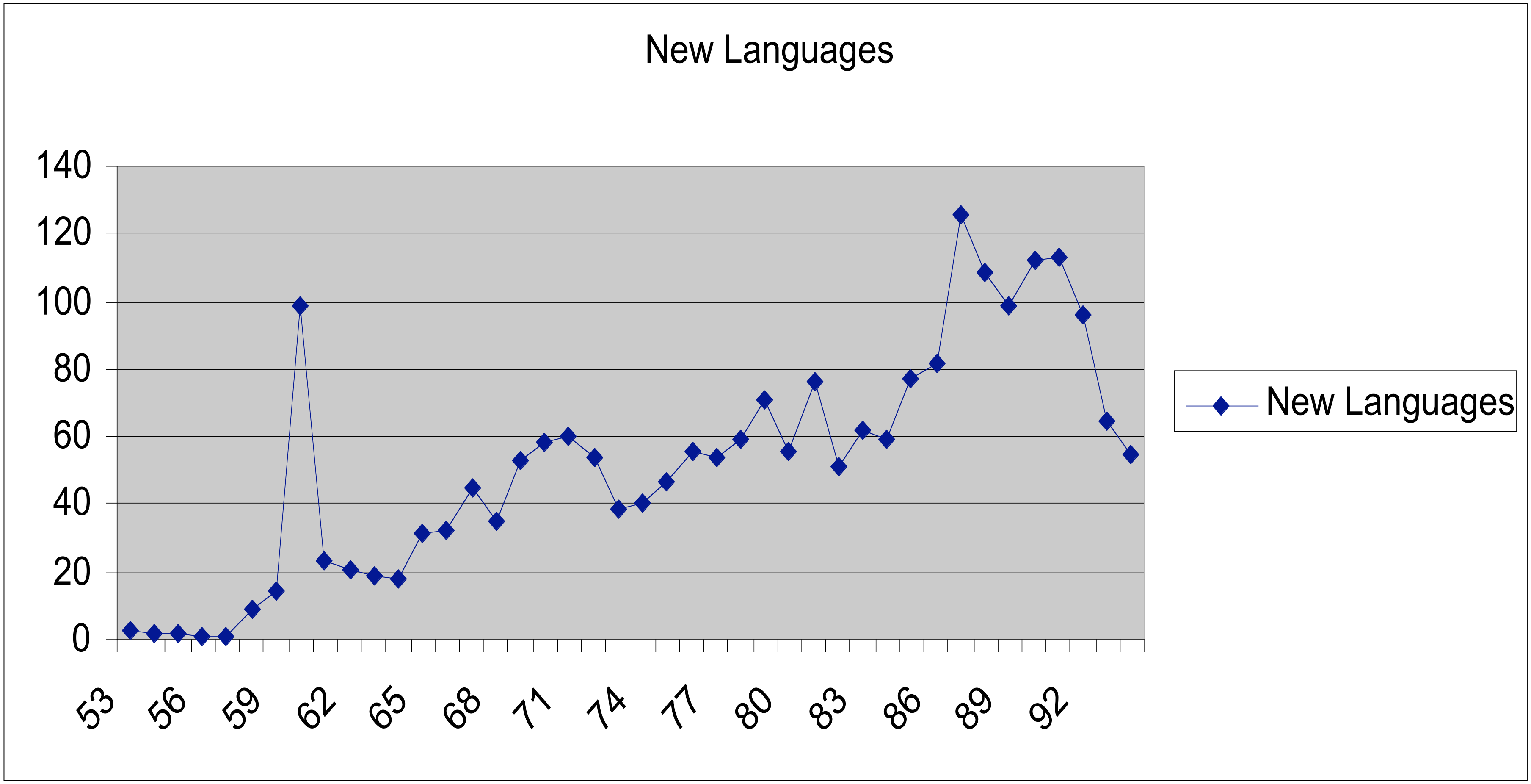
- Required Text: "Compilers : Principles, Techniques, & Tools", Aho, Lam, Sethi, Ullmann
  - This gives a comprehensive overview of course material. Will be a great reference for your career.
- Optional Text : "The Definitive ANTLR Reference", Parr
  - I have a PDF copy I can share , but I cannot post it.
- Lecture Notes: posted as PDF
- Course Web Pages: extensive tool documentation and pointers to other resources

Exams will be based on material covered in lecture notes and course projects

Why Study Compilers?

- understand existing languages;
- appreciate current limitations;
- talk intelligently about language design;
- see a great example of theory in practice;
- implement your very own general-purpose language; and
- implement lots of useful domain-specific languages.

Language Birth Rates



Domain Specific languages

- extend software design; and
- are concrete artifacts that permit representation, optimization, and analysis in ways that low-level programs and libraries do not.
- Prominent examples are:
  - LaTeX
  - Lex/Yacc
  - HTML
- Domain-specific languages require full-scale compiler technology.

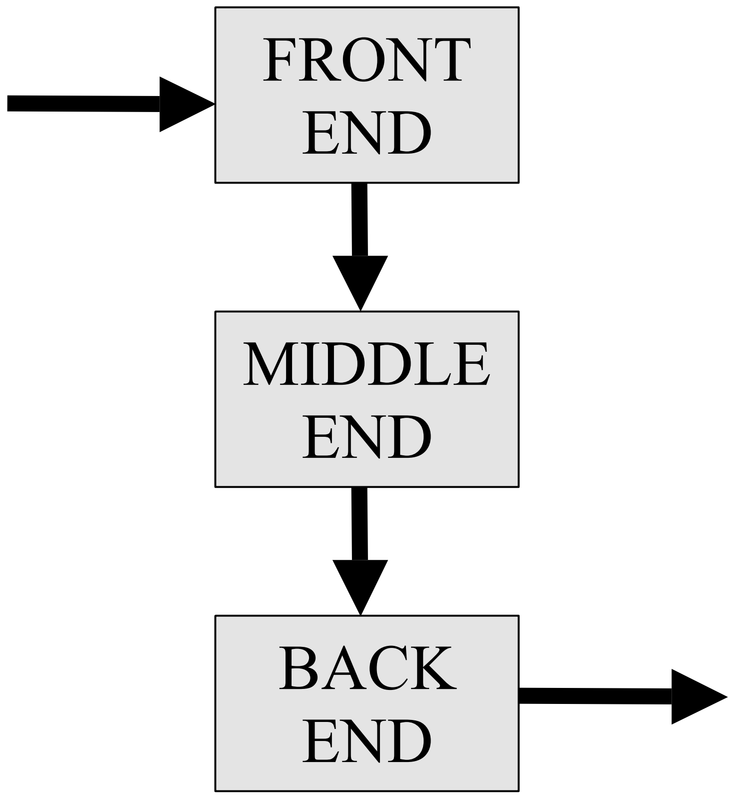
FORmula TRANslation Compiler

- implemented in 1954--1957;
- the world's first compiler;
- for the domain of scientific/mathematical programming;
- was motivated be the economics of programming (*speedcoding*);
- had to overcome deep skepticism;
- paid little attention to language design;
- focused on efficiency of the generated code;
- pioneered many concepts and techniques; and
- revolutionized computer programming.

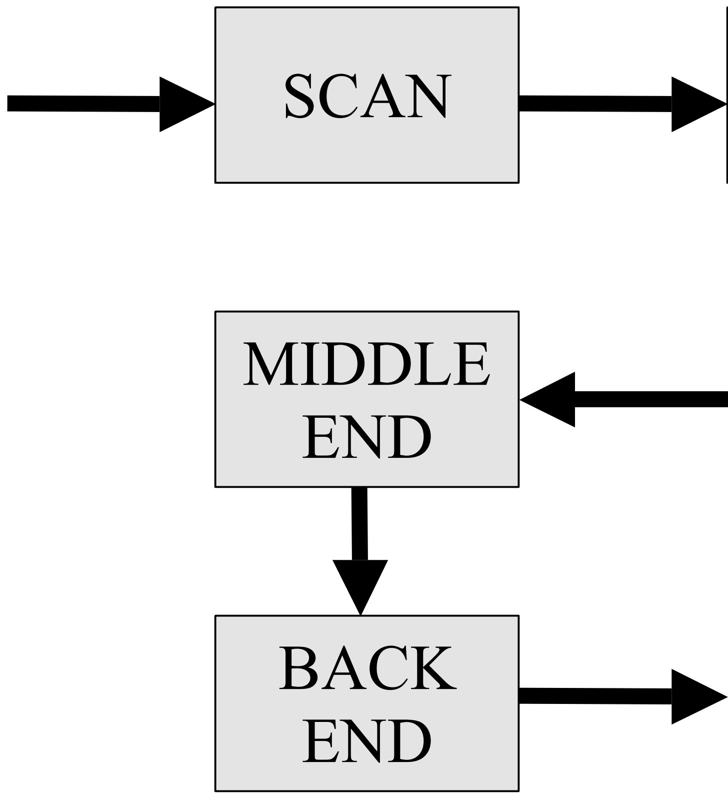
Theory in Practice

- in the 60s *compilation* was art;
- in the 70s *compilation* was studied by theoreticians;
- in the 80s *compilation* was studied as a software product line;
- and it is probably the most mature *software domains* you will ever work in.

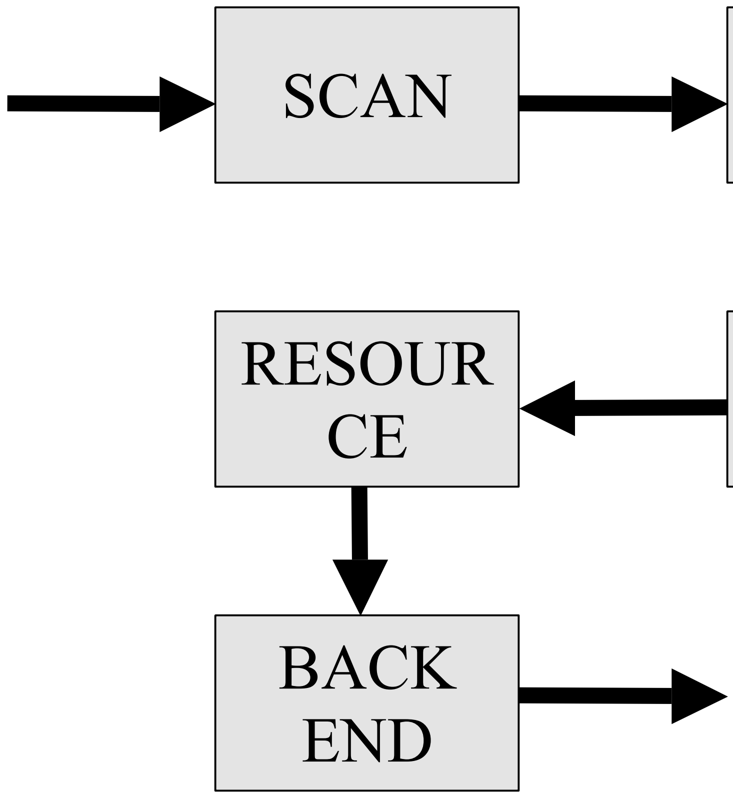
Compiler Architecture



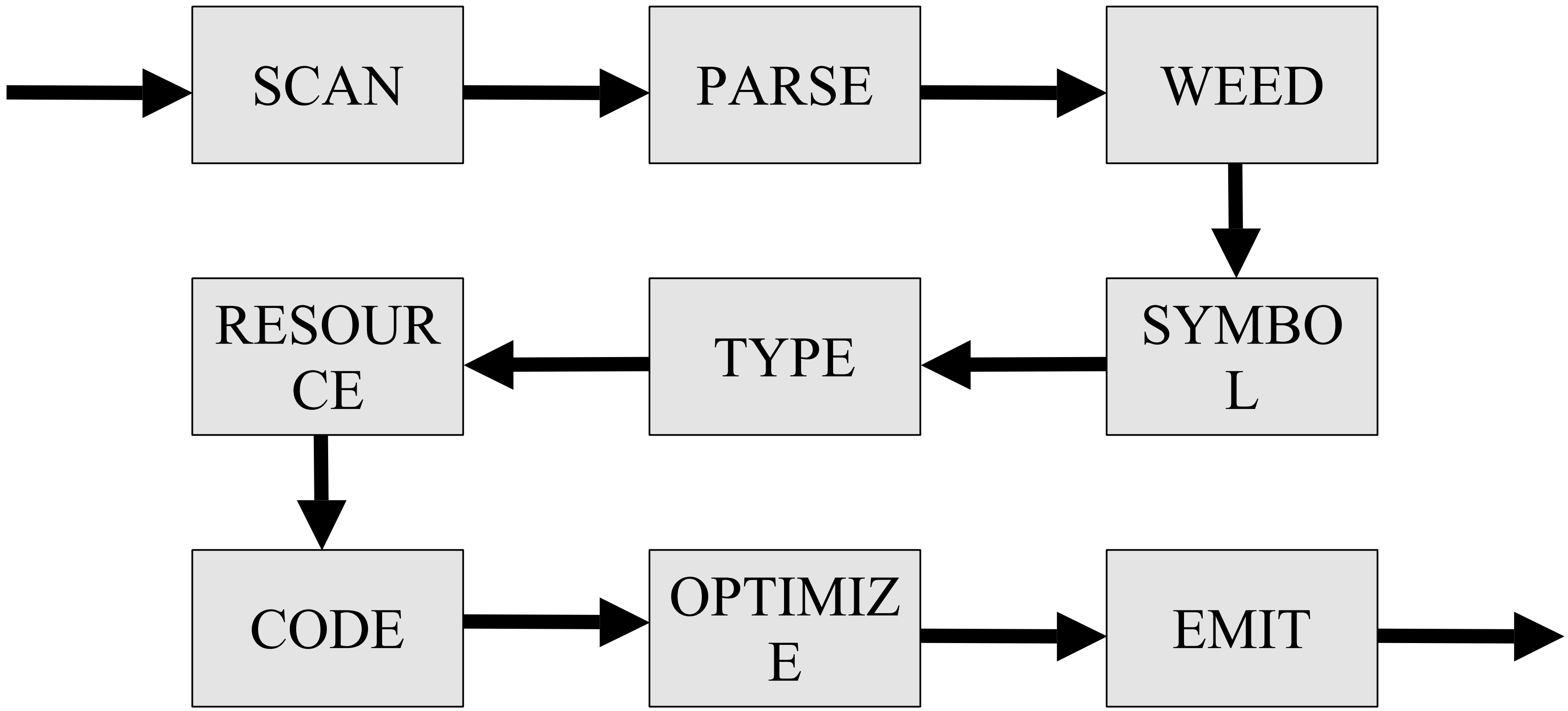
Compiler Architecture



Compiler Architecture



Compiler Architecture



Compilers as Software Domain

- Many thousands of compilers have been built;
- A standard architecture with standard components and interfaces has evolved;
- DSLs and DSL compilers for *compiler construction* are in common use
  - Front ends: antlr, flex, bison, sablecc, llgen, …
  - Middle ends: memphis, pag, …
  - Back ends: beg, twig, …

Primary Project: Static Java

- A subset of Java;
- compiled into Java Virtual Machine code;
- illustrates a general-purpose language;
- illustrates details of VM-based run-time;
- used to teach by example;
- the source code for *sjc* will be studied;
- and you will upgrade *sjc* to add several new features.
- *sjc* source code is available in Java.

Example : Static Java

```
public class Factorial
{
    public static void main(String[] args)
    {
        StaticJavaLib.println(factorial(StaticJavaLib.getIntArgument(args, 0)));
    }

    static int factorial(int n)
    {
        int result;
        int i;

        StaticJavaLib.assertTrue(n >= 1);
        result = 1;
        i = 2;
        while (i <= n)
        {
            result = result * i;
            i = i + 1;
        }

        return result;
    }
}
```

Course Projects

- You will work alone;
- You will develop an extension of the SJC compiler in Java;
- You will become expert in using complex Java APIs and design patterns;
- Final exam serves as a check that everyone did the work they were supposed to do;
- Grades are based on how well your compiler does on a group of test cases.

Grad Project: Software Tools

- Extend the code-generation strategy to support test coverage adequacy measures;
- Record branch coverage for each program execution;
- Optimize branch coverage to minimize run-time cost by exploiting dominance information;
- Evaluate the benefits of your optimization;
- You will extend compiler you build in the course project.

Project Tools

- Compiler as built makes significant use of Eclipse plugins;
- Compile test suite runs in Eclipse;
- Grades will be calculated based on how well your compiler does against the test suite;
- Check that you can access a machine with Eclipse 3.1;
- You should be able to install the rest of the tools yourself.

Myth?

People are better at optimizing their programs than compilers

```
for (i = 0; i < N; i++) {
  a[i] = a[i] * 2000;
  a[i] = a[i] / 10000;
}
```

```
b = a;
for (i = 0; i < N; i++) {
  *b = *b * 2000;
  *b = *b / 10000;
  b++;
}
```

Which loop runs faster?

The Answer is …

LOOP	Optimization	SPARC	MIPS	Alpha
array	none	20.5	21.6	7.85
array	opt	8.8	12.3	3.26
array	super	7.9	11.2	2.96
pointer	none	19.5	17.6	7.55
pointer	opt	12.4	15.4	4.09
pointer	super	10.7	12.9	3.94

Why?

- Pointers confuse most compilers
  - difficult to tell what they refer to
  - ese arrays whenever possible
- Compilers use sophisticated register allocation algorithms
  - languages don't give enough control
  - it is too expensive to compute a good assignment by hand
- High-level languages are for people
  - write clear code and let the compiler optimize it

For Next Time

- Browse web site
  - Send bug reports
- Read chapter 1 of Dragon (start working through chapter 2)
- Assess your computing platform and start installing tools
  - Let me know of problems and I’ll try to help