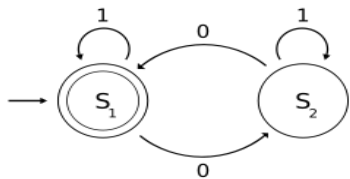
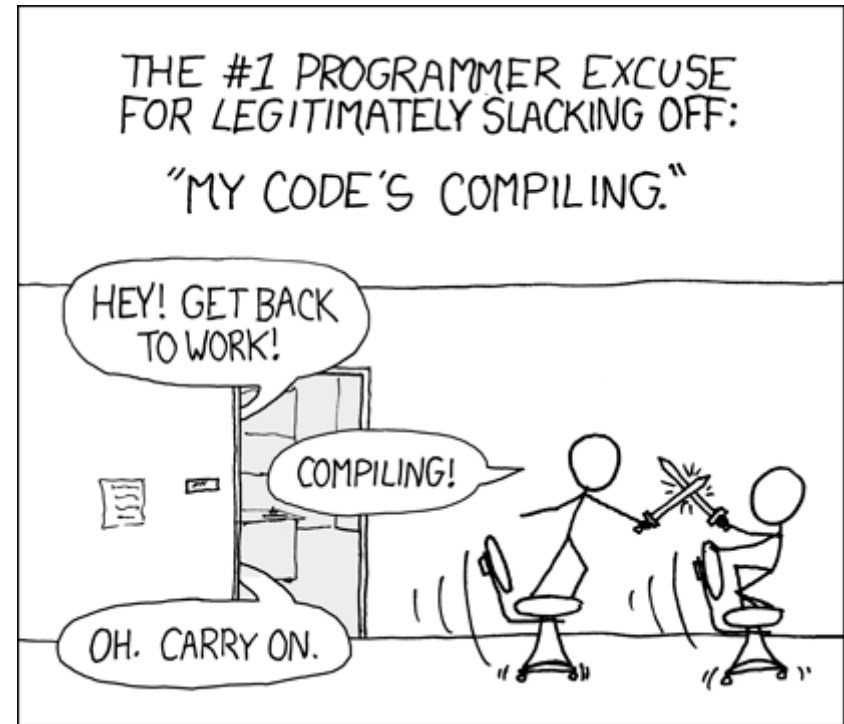


CS 432

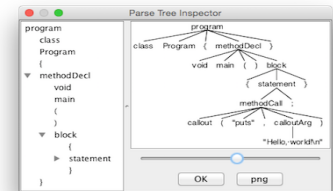
Fall 2016

Mike Lam, Professor



Compilers

Advanced Systems Elective

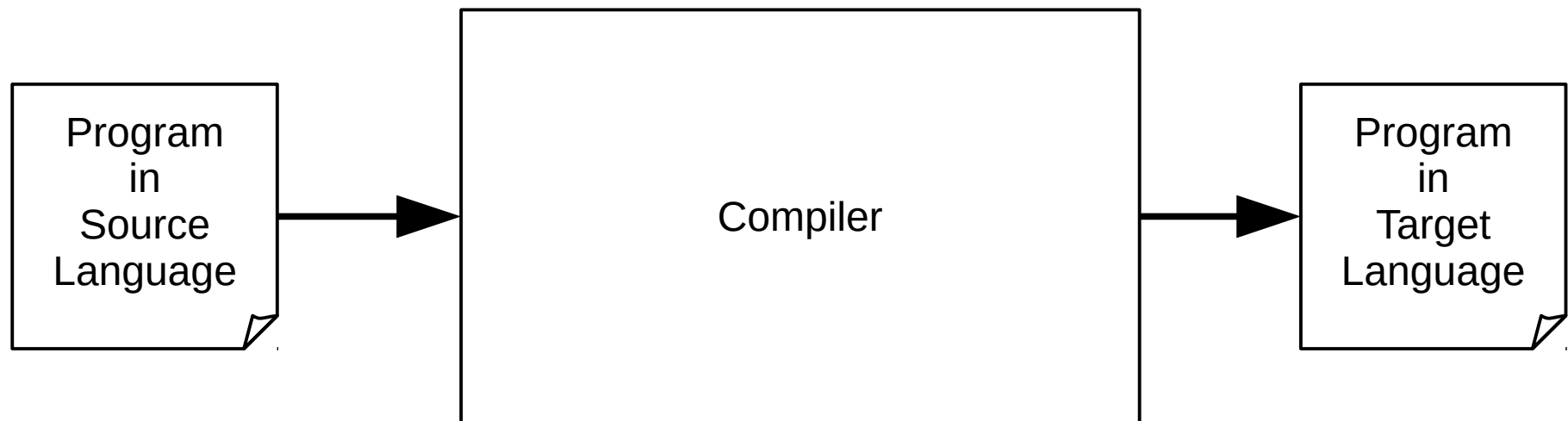


Discussion question

- What is a compiler?

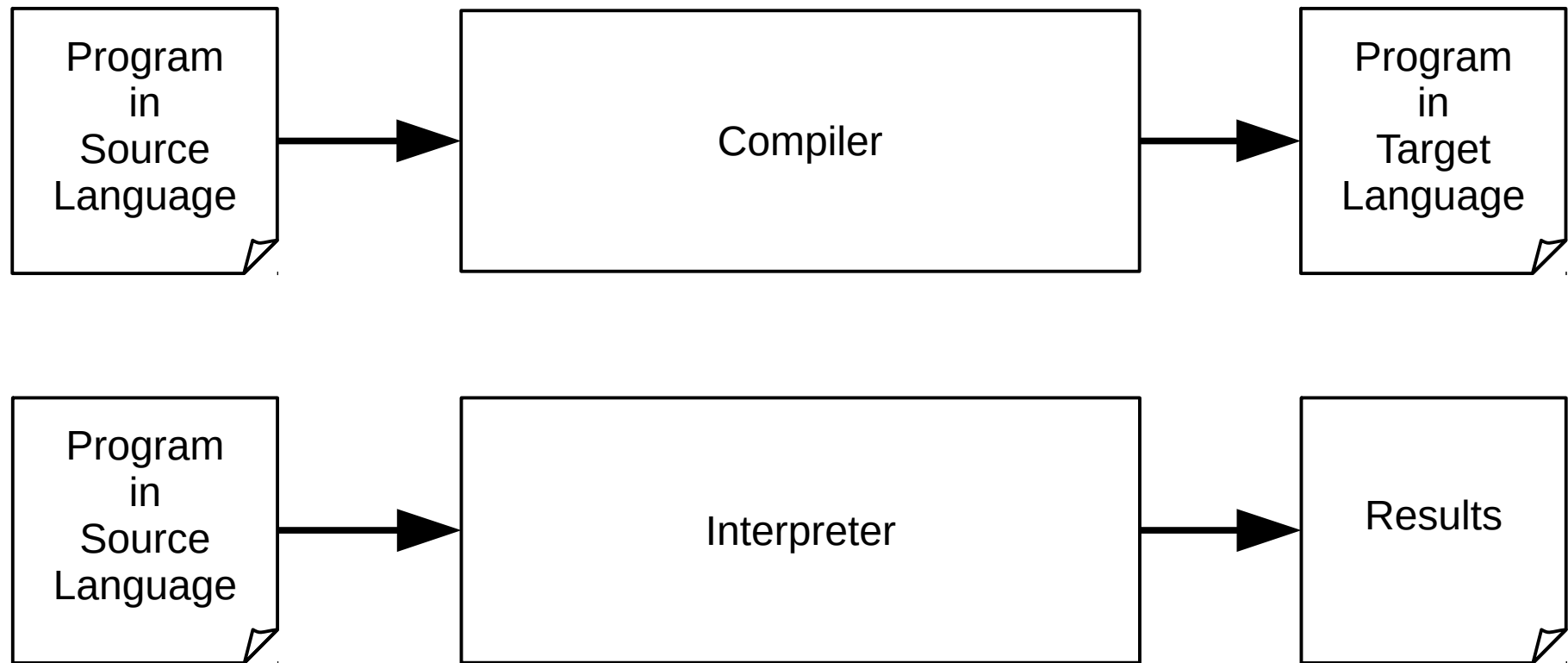
Automated translation

- A compiler is a computer program that **automatically translates** other programs from one language to another



Automated translation

- Compilation vs. interpretation:



Discussion question

- Why should we study compilers?

Compilers: a convergent topic

- Data structures
 - CS 240
- Architectures and operating systems
 - CS 261, CS 450
- Automata and language theory
 - CS 327, CS 430
- Graph algorithms
 - CS 327
- Software and systems engineering
 - CS 345, CS 361
- Greedy, heuristic, and fixed-point algorithms
 - CS 452

Reasons to study compilers

- Shows how many areas of CS can be combined to solve a truly "hard" problem (automated translation)
- Bridges theory vs. implementation gap
 - Theory informs implementation
 - Applicable in many other domains
- Practical experience with large software systems
 - My master copy is currently ~6.3K LOC
- Exposure to open problems in CS
 - Many optimization issues are subject to ongoing research

Course goal

- Fundamental question
 - "How do compilers translate from a human-readable language to a machine-executable language?"
- After this course, your response should be:
 - "It's really cool! Let me tell you..."

Course design theory

- Big ideas
 - E.g., "A compiler is a large software system consisting of a sequence of phases"
- "Enduring understandings" (stuff you should remember in five years)
 - E.g., "Large problems (such as automated translation) can sometimes be solved by composing existing solutions to smaller problems."
- Learning objectives
 - E.g., "Identify and discuss the technical and social challenges of building a large software system such as a compiler."
- Activities and assignments flow from learning objectives
 - E.g., "Draw a diagram illustrating the major phases of a modern compiler."
- Exams reflect activities and assignments

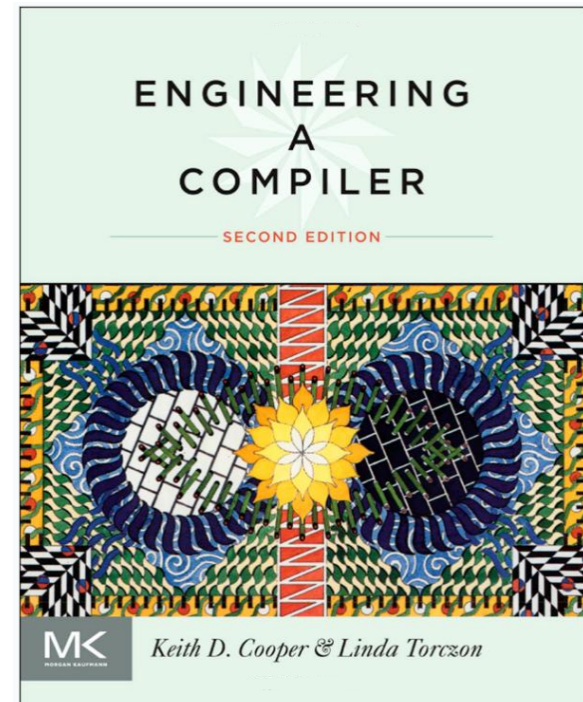
Course format

- Website: <https://w3.cs.jmu.edu/lam2mo/cs432/>
- Weekly schedule
 - Monday: intro lecture
 - Tuesday: initial reading & quiz
 - Wednesday: mini-lecture and discussion
 - Thursday: detailed reading
 - Friday: application activity
- Formative vs. summative assessment
- Goal: engaged learning

Course textbook

- Engineering a Compiler, 2nd Edition
 - Keith Cooper and Linda Torczon

- Decaf/ILOC references
 - PDFs on website



Semester-long project

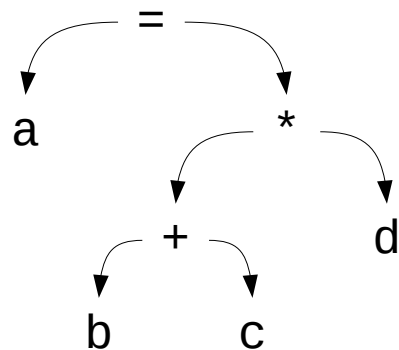
- Compiler for "Decaf" language
 - Implementation in Java
 - Maven build system w/ fully-integrated test suite
 - Six projects comprised of "pieces" of the full system
- Submission: code + reflection + review
 - Code can be written in teams of two
 - Benefits vs. costs of working in a team
 - Reflection paper must be individual
 - Graded code reviews after project submission

Compiler rule #1

- "The compiler must preserve the *meaning* of the program being compiled."
 - What is a program's *meaning*?

Intermediate representation

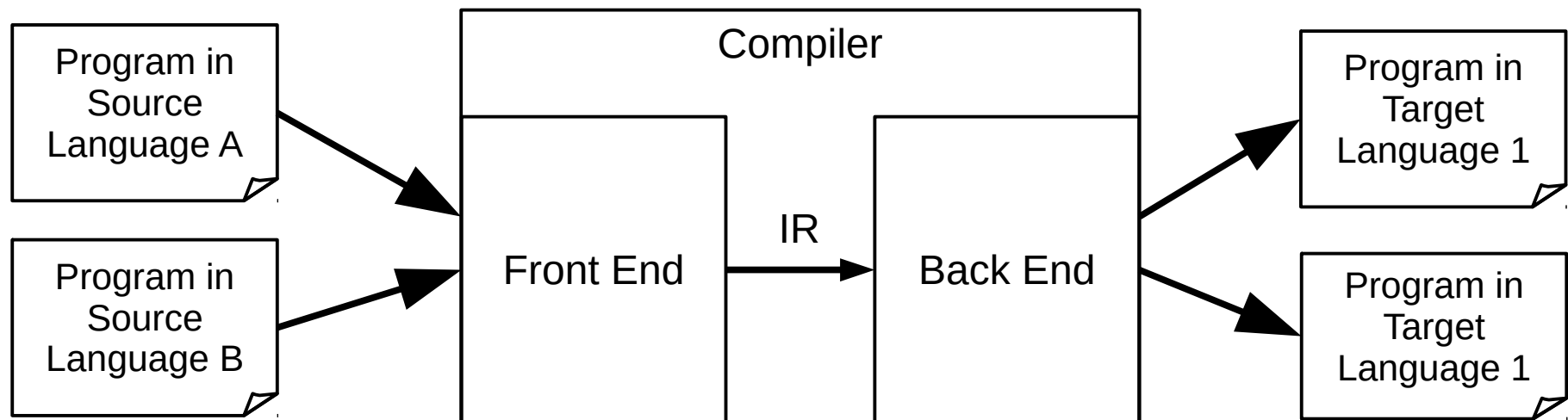
- Compilers encode a program's meaning using an intermediate representation (IR)
 - Tree- or graph-based: abstract syntax tree (AST), control flow graph (CFG)
 - Linear: register transfer language (RTL), Java bytecode, intermediate language for an optimizing compiler (ILOP)



```
load b → r1
load c → r2
add r1, r2 → r3
load d → r4
mult r3, r4 → r5
store r5 → a
```

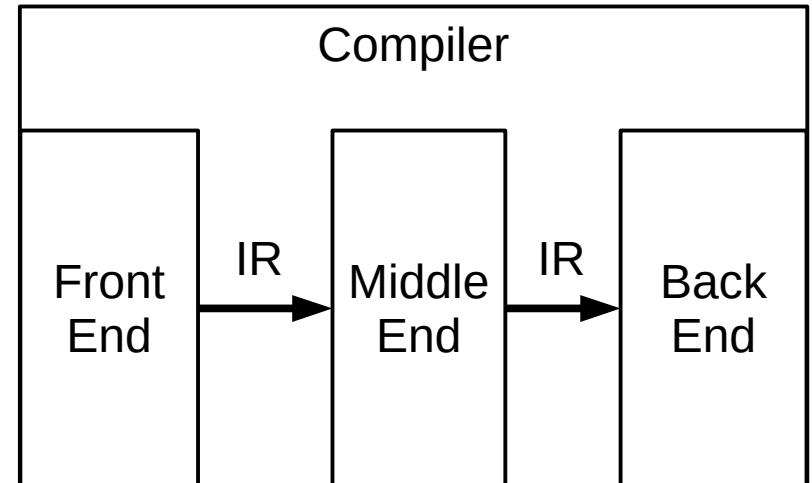
Standard compiler framework

- Front end: understand the program (src → IR)
- Back end: encode in target language (IR → targ)
- Primary benefit: easier *retargetting* to different languages or architectures



Modern compiler framework

- Multiple front-end passes
 - Lexing/scanning and parsing
 - Tree analysis processing
- Multiple middle-end passes
 - Local optimizations
 - Global optimizations
- Multiple back-end passes
 - Instruction selection and scheduling
 - Register allocation
 - Linking



Compiler rule #2

- The compiler should *help* the programmer in some way
 - What does *help* mean?

Discussion question

- As a programmer, what would be the most important design goals for a compiler?

Compiler design goals

- Translate to target language/architecture
- Optimize for fast execution
- Minimize memory use
- Catch software defects early
- Provide helpful error messages
- Run quickly
- Be easily extendable

Differing design goals

- What differences might you expect in compilers designed for the following applications?
 - A just-in-time compiler for running server-side user scripts
 - A compiler used in an introductory programming course
 - A compiler used to build scientific computing codes to run on a massively-parallel supercomputer
 - A compiler that targets a number of diverse systems
 - A compiler that targets an embedded sensor network platform

Decaf language

- Simple imperative language similar to C or Java
- Example:

```
// add.decaf - simple addition example
```

```
def int add(int x, int y)
{
    return x + y;
}
```

```
def int main()
{
    int a;
    a = 3;
    return add(a, 2);
}
```

```
$ java -jar decaf-1.0.jar -i add.decaf
5
```

Before Wednesday

- Readings
 - "Engineering a Compiler" (EAC) Ch. 1 (23 pages)
 - Decaf reference ("Resources" page on website)
- Tasks
 - Complete first reading quiz on Canvas
 - Complete source intro survey on Canvas
 - Install Java and Maven on your system
 - Download the reference compiler from Canvas ("Files")

Upcoming events

- CS senior night
 - **Wednesday, Sept. 14, 5:00-6:30pm**
 - Graduation info, job fairs, photos, etc.
 - Senior students only
- CS career fair
 - **Monday, Sept. 22, 10:00am-3:00pm**
 - 8-10 companies looking for CS majors
 - Internships and full-time positions

See you Wednesday

- Have a great semester!