Lecture A:

Overview

CSE401/501m:

Introduction to Compiler Construction

Instructor: Gilbert Bernstein

Outline

Introductions

Administrivia

What's a compiler?

Why you might want to take this course

Outline

Introductions

Administrivia

What's a compiler?

Why you might want to take this course

Instructor

- **Gilbert Bernstein**
  - Assistant Professor (started Winter 2023)
- I research Programming Languages & Computer Graphics — especially DSLs (Domain Specific Languages)
  - I've written 5+ compilers from scratch, and worked on 10+
- I even work on compilers for knitting machines!
- This is my first time teaching an intro compilers class (please be understanding of any goofs)

Teaching Assistants

- TAs this quarter:
  - Eric Chen
  - Karen Haining
  - Sriya Bulusu
  - Bill Baxter
- Office Hours (Rooms TBA)

Mon	Tue	Wed	Thu	Fri
11:30-12:30	1:30-2:30	11:30-12:30	-	1:30-2:30
1:00-2:00	2:30-3:30	1:00-2:00	3:30-4:30	-

Welcome Back!

- I hope you had a great spring break
- We're all in this together!
  - Please talk to us! If you're having trouble don't "tough it out." Ask us for help. If things are going well, let us know.
- In this class, we want to all hold each other to a high standard ***and*** be kind to one another. Always assume the best intentions in other students, staff and in yourself
- Be realistic about your workload — make sure you have the time and energy for your commitments, academic and otherwise.
  - Do NOT "ghost" your partner! 👻

Outline

Introductions

Administrivia

What's a compiler?

Why you might want to take this course

CSE 501m

- Enhanced version of this course for 5th year BS/MS students
- M501 students will have to do a significant addition to the project or some other extra work if agreed with the instructor (papers, reports, ???)
  - ✦ more details later
- Otherwise 401 and 501m are the same (lectures, sections, assignments, infrastructure)

Prerequisites

- Required Prerequisites
  - ✦ CSE 332 (Data Abstractions)
    - and thus CSE 311 (Foundations) *i.e. language theory*
  - ✦ CSE 351 (HW/SW interface, x86_64 assembly)
- Also very useful, but not required
  - ✦ CSE 331 (Software Design & Implementation)
  - ✦ CSE 341 (Programming Languages)

Lectures & Sections

- Both are required — unique content in each
- All materials will be posted, but as aids not replacements
  - ✦ Come to class; take notes (→ do better in class!)
  - ✦ Panopto lectures are for review & unavoidable absences only — this is research backed; positive as a review, but not as a substitute
- Sections: additional examples & exercises, plus project details and tools
  - ✦ We will have sections this week (Thu) — don't miss!

Screens & Gadgets

- Gadgets reduce focus while learning
  - ✦ Bursts of Info (e.g. notifications, DMs) are addictive
  - ✦ Heavy multitaskers have trouble focusing & shutting out irrelevant information — research is clear here
- So, how should we deal with laptops and phones?
  - ✦ Just say no! (unlike Panopto, no upsides)
  - ✦ No open gadgets during class (yes, really!)
    - *(unless you are actually taking notes…)*
  - ✦ Urge to search? Ask a question! Everyone benefits!
  - ✦ Pull out a piece of paper and pencil/pen instead! 

“
STUDYING:
You are all computer scientists.
You know what FINITE AUTOMATA can do.
You know what TURING MACHINES can do.
For example, Finite Automata can add but not multiply.
Turing Machines can compute any computable function.
Turing machines are incredibly more powerful than Finite Automata.
Yet the only difference between a FA and a TM is that the TM, unlike the FA, has paper and pencil.
Think about it.
It tells you something about the power of writing.
Without writing, you are reduced to a finite automaton.
With writing you have the extraordinary power of a Turing machine. ”

—Manuel Blum

Communications

- Course website (www.cs.uw.edu/401)
- Discussion Board — Ed
  - For almost anything related to the course
  - Join in! Please help each other out. Staff will contribute.
  - Use private messages for too-specific-to-post questions, if you need to share code to ask, etc.
  - Staff will use Ed to post announcements
- Gradescope — written assignments & regrade requests
- Email to cse401-staff[at]cs for project feedback, questions, unexpected or personal situations, things that need a followup, not appropriate for ed

Requirements & Grading

- Midterm & Final Exam
  - Dates are on the course calendar
- Grading Breakdown*
  - 50% project (w/partner): ½ final state, ½ intermediate
  - 25% individual homework
  - 10% midterm
  - 15% final
- Deadlines: 11:59pm for everything

***we reserve the right to adjust as needed/appropriate**

Academic Integrity

- Let's have fun, learn, and help each other!
- BUT
  - ✦ You must not misrepresent work done by someone (or something) else as your own. Always attribute work!
  - ✦ You must not attempt to bypass learning by avoiding work, or help others do similarly
- **Read the course policy on the website carefully**
- Honest work is the foundation of your university work (and in engineering, business, life). Anything less demeans your teachers, your classmates, and ultimately yourself.
- If in doubt about whether something is ok, ask.

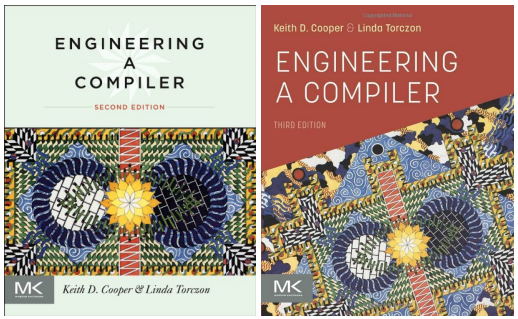
Course Project

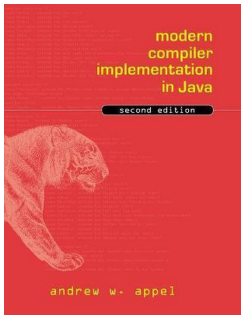
- The best way to learn about compilers is to build one!
  - ✦ (ideally several!)
- You will be writing a compiler for “MiniJava”
  - ✦ “core” parts of Java; from Appel textbook (not needed)
  - ✦ We will generate and run x86_64 code directly
    - note: this is not the same as what the JVM does
  - ✦ Completed in 4 steps throughout the quarter
    - 1. Scanner 2. Parser 3. Checking 4. Codegen
  - ✦ Additional work for CSE 501m students. Usually, add some interesting feature to MiniJava.

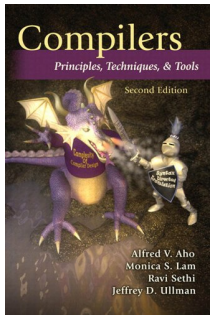
Project Groups

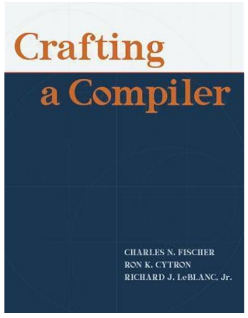
- You should work in pairs
  - ✦ Pick a partner now for the rest of the quarter — Partner choices are due next Tuesday (see calendar)
    - Make sure you agree on work strategy, deadline attitudes, etc.
  - ✦ If you are in 501m, make sure you pick a partner who is as well. (401 → 501m switches are possible if it makes sense for the specific individuals involved)
  - ✦ Partnering remotely can work very well even without hanging out in the labs (e.g. Zoom, VScode Live Share)
- We will set up repositories on the department Gitlab — let us know if you haven't used Gitlab before. Staff will retrieve your turnins from this repository

Textbooks









- Cooper & Torczon. *Engineering a Compiler. 2nd or 3rd edition.*
  - ✦ The ***official text*** for this course. 2nd edition available free online through UW Library Safari books login. 3rd edition recently released.
- Appel. *Modern Compiler Implementation in Java. 2nd edition.* (MiniJava)
- Aho, Lam, Sethi, Ullman. *"Dragon Book"*
- Fischer, Cytron, LeBlanc. *Crafting a Compiler.*

Outline

Introductions

Administrivia

What's a compiler?

Why you might want to take this course

What's a Programming Language?

- The most basic definition of a programming language
  1. Some way to **represent** a program (actions, computations) **as data**
  2. Some way to **evaluate** or **interpret** those programs (i.e. to execute the actions/computations)
- This is very abstract, so let's consider an example

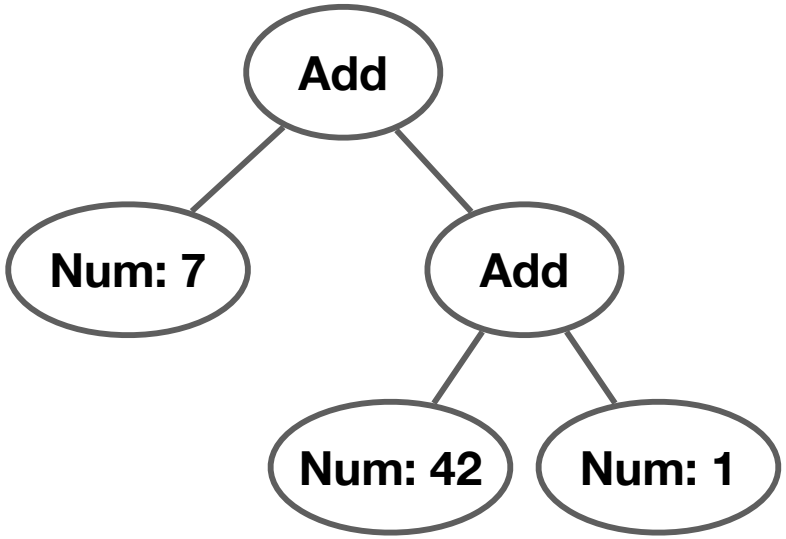
A Calculator Language

```
abstract public class Expr {}

public class Num extends Expr {
    public int i;
    public Num(int v) { i = v; }
}

public class Add extends Expr {
    public Expr e0, e1;

    public Add(Expr a0, Expr a1) {
        e0 = a0;
        e1 = a1;
    }
}
```



e.g.

7 + (42 + 1)

```
Expr e = new Add(
    new Num(7),
    new Add( new Num(42),
             new Num(1) ));
```

A Calculator Language

```
abstract public class Expr {
    public abstract int eval();
}

public class Num extends Expr { …
    public abstract int eval() {
        return i;
    }
}

public class Add extends Expr { …
    public abstract int eval() {
        return e0.eval() + e1.eval();
    }
}
```

A Calculator Language

```
abstract public class Expr {
    public abstract int eval(
}

public class Num extends Expr
    public abstract int eval(
        return i;
    }
}

public class Add extends Expr { …
    public abstract int eval() {
        return e0.eval() + e1.eval();
    }
}
```

```
Expr e = new Add(
    new Num(7),
    new Add( new Num(42),
             new Num(1) ));
```

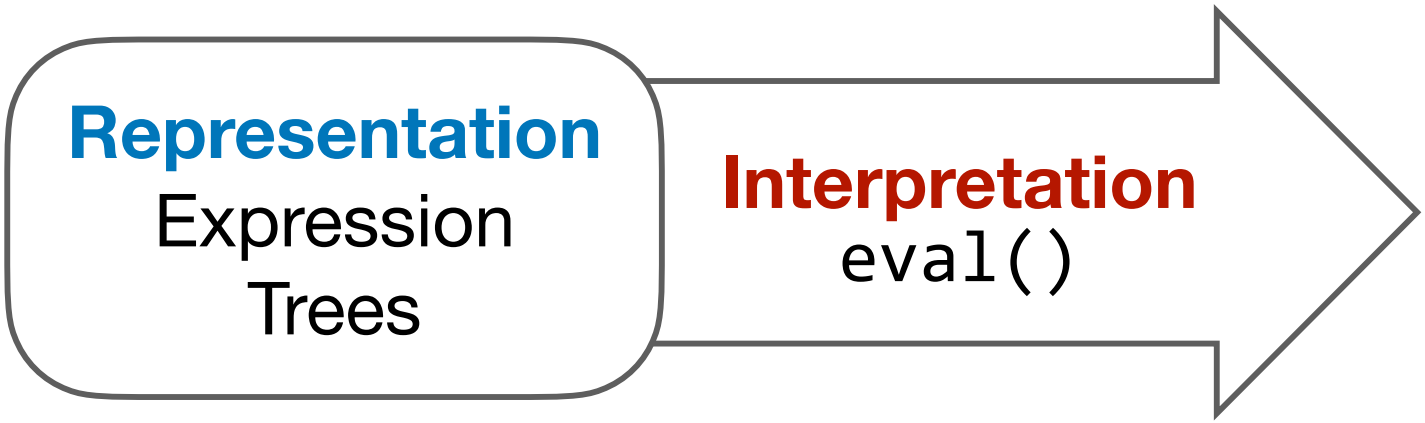
```
System.out.println(e.eval());
```

```
50
```

Nice Trick

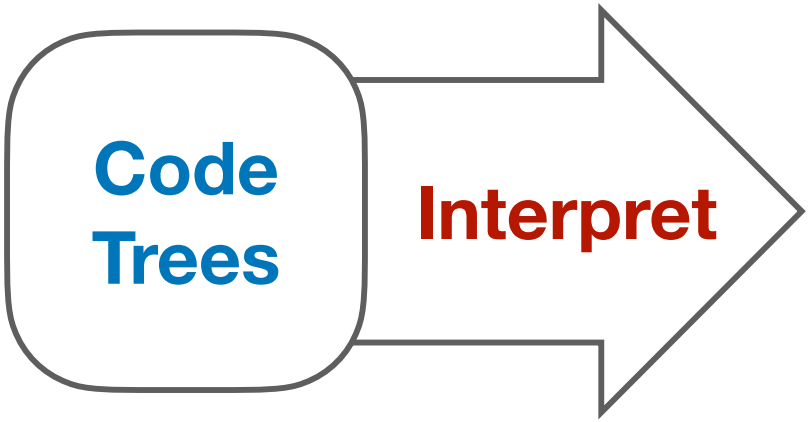
- Where's the language?
  1. Objects **represent** expressions **as trees**
  2. `eval()` method **interprets trees** by adding things up
- Objections, i.e. arguments this isn't a language
  1. The calculator language isn't useful for anything!
  2. This
```
Expr e = new Add(
     new Num(7),
     new Add( new Num(42),
              new Num(1) ));
```
     is just Java code.
  3. `eval()` just returns a number; a real language should *generate an executable program file*

Objection #1: Utility

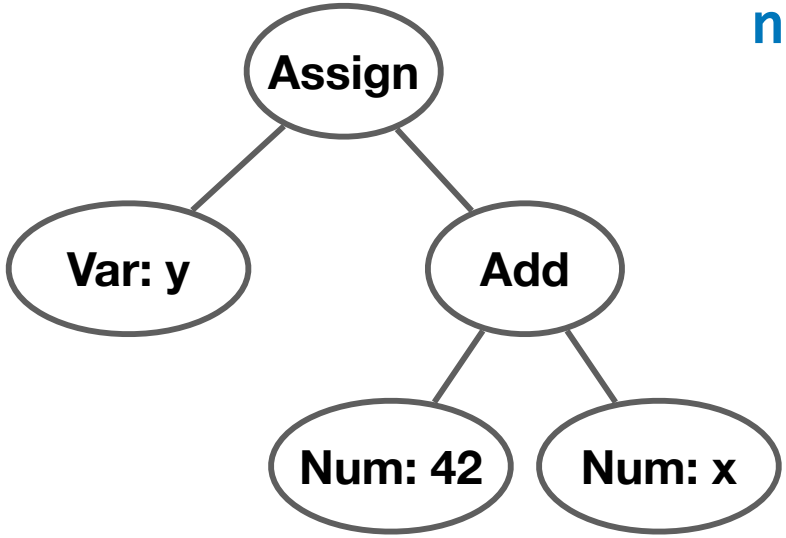


- We could make this slightly more useful by adding variables and changing the signature of eval() to specify *input values* for those variables
- What about if we added to the representation?
  - ✦ Assignments? Loops? Classes? (Each another node)
  - ✦ Couldn't we represent all of Java this way?

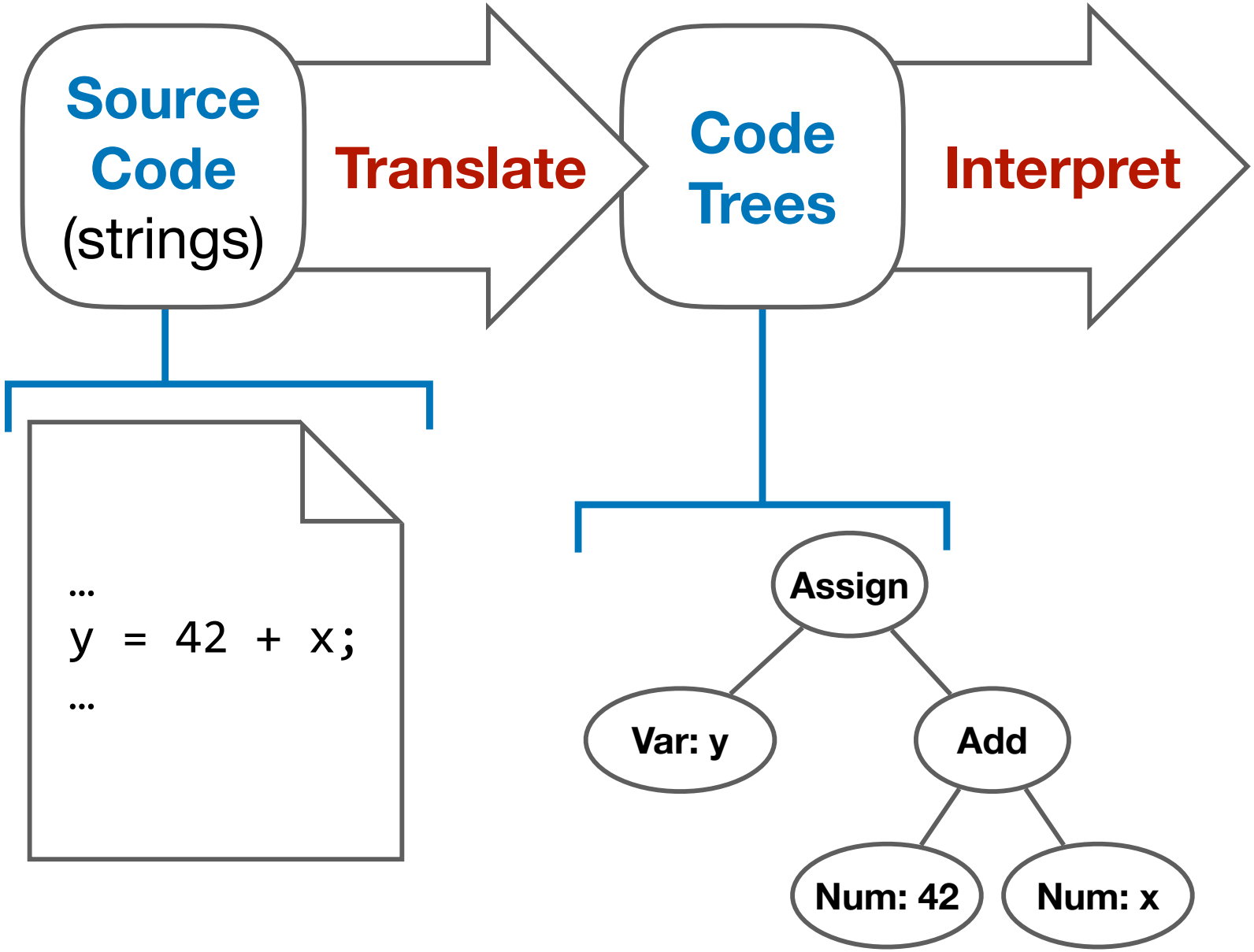
Objection #2: Input is Java!



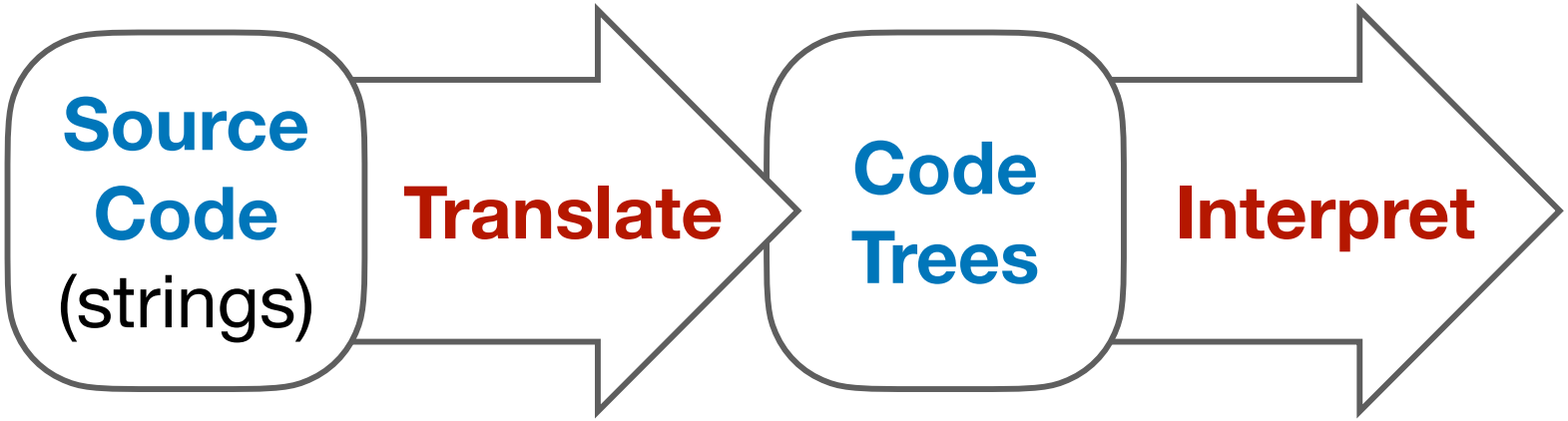

```
Stmt s = new Assign(
    new Var("y"),
    new Add( new Num(42),
             new Var("x")));
```



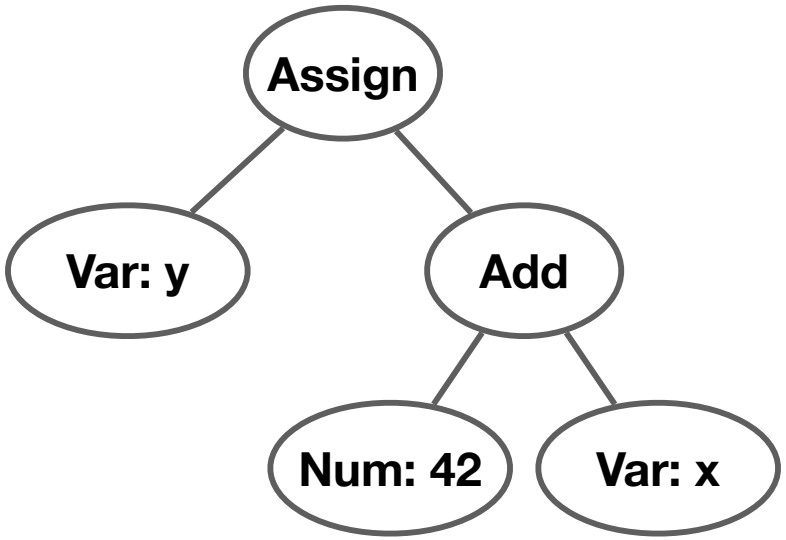
Objection #2: Input is Java!



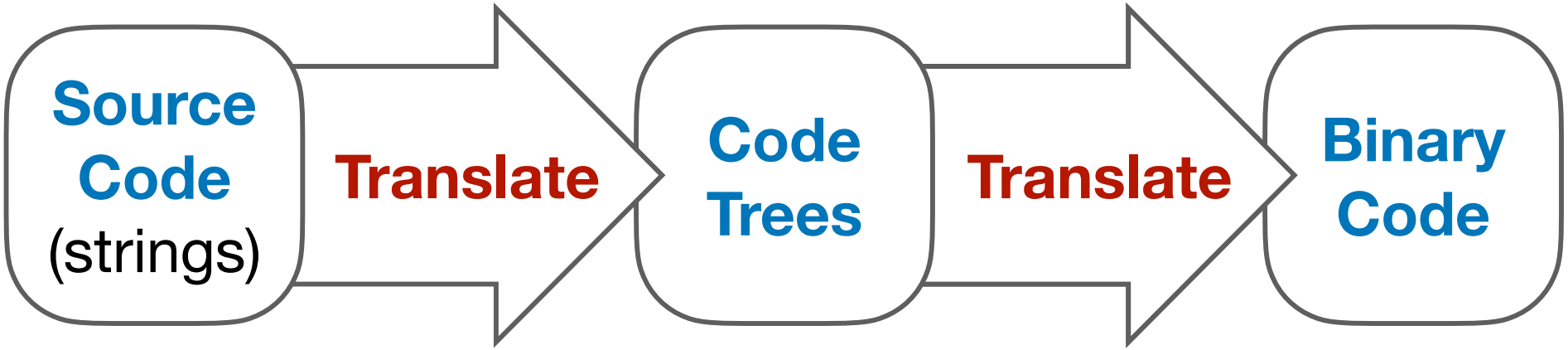
Objection #3: No Exec. Output



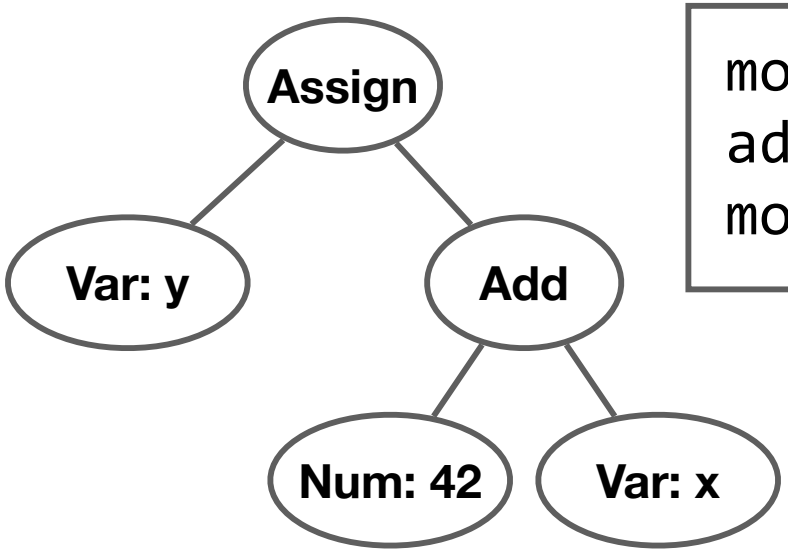

```
…
y = 42 + x;
…
```



Objection #3: No Exec. Output

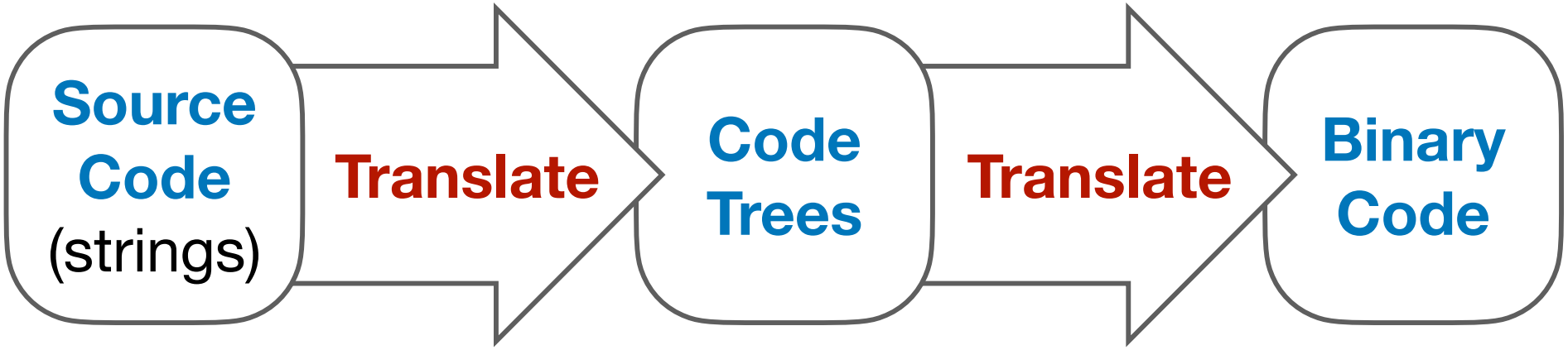



```
…
y = 42 + x;
…
```



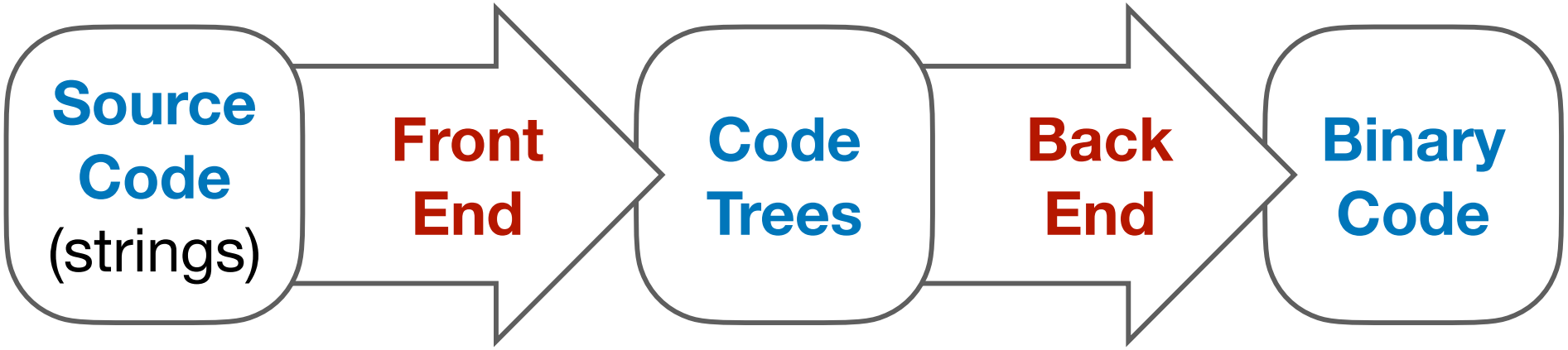

```
movq -8(%rbp),%rax
addq $42,%rax
movq %rax,-16(%rbp)
```

This is a Compiler



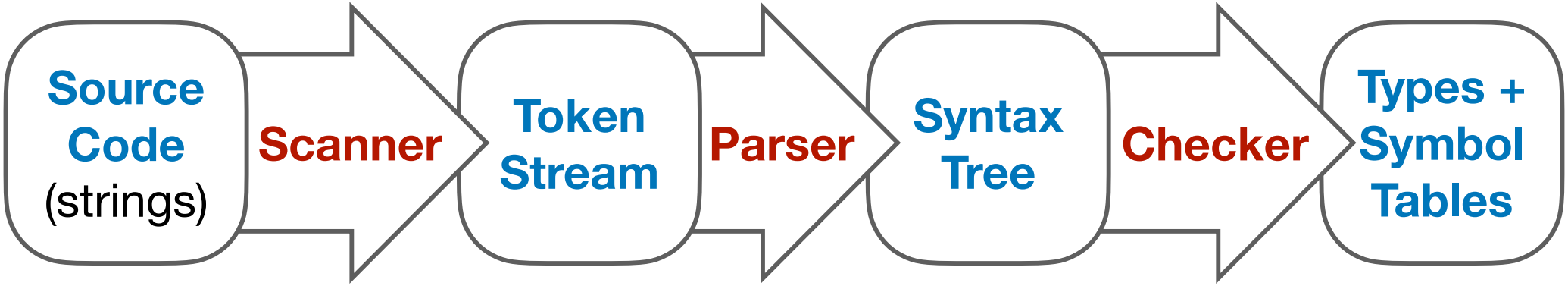
- A **Compiler** is a program that translates code from one representation into code in a different representation
- Every compiler is actually made up of a bunch of smaller compilers, each of which we call a **compiler pass**
- In between the passes are **intermediate representations** (aka. IRs)

Front End vs. Back End

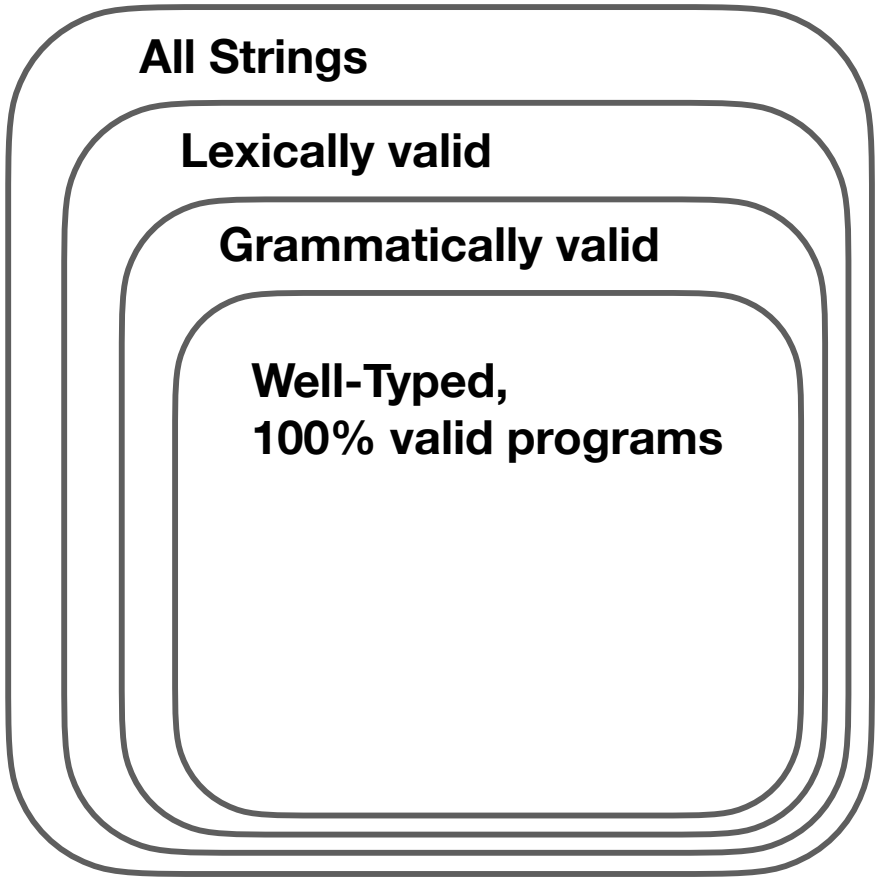


- Traditionally Compilers are divided into a front end and a back end, each made up of more **passes** & **IRs**
- The front end is responsible for ingesting programs and deciding whether or not they are allowed/safe
- The back end is responsible for optimizing programs, managing/planning resource use, and generating code

A Basic Front End



- Scan: Break the input file into a sequence of tokens
- Parse that sequence into a tree
- Check the code using additional rules/analyses (types and semantics)



Back Ends

- A lot more stuff going on
- Major parts (first-to-last & most-to-least complicated)
  - ✦ Target-independent optimization
  - ✦ Target-specific optimization
  - ✦ Code generation

LLVM -O2 optimization passes

```
targetlibinfo
tti
no-aa
tbaa
scoped-noalias
assumption-
  cache-tracker
basicaa
ipsccp
globalopt
deadargelim
domtree
instcombine
simplifycfg
basiccg
prune-eh
inline-cost
inline
functionattrs
domtree
sroa
early-cse
lazy-value-info
jump-threading
correlated-
  propagation
simplifycfg
domtree
instcombine
```

```
tailcallelim
simplifycfg
reassociate
domtree
loops
loop-simplify
lcssa
loop-rotate
licm
loop-unswitch
instcombine
scalar-evolution
loop-simplify
lcssa
indvars
loop-idiom
loop-deletion
loop-unroll
mldst-motion
domtree
memdep
gvn
memdep
memcpyopt
sccp
domtree
bdce
instcombine
lazy-value-info
```

```
jump-threading
correlated-
propagation
domtree
memdep
dse
loops
loop-simplify
lcssa
licm
adce
simplifycfg
domtree
instcombine
barrier
float2int
domtree
loops
loop-simplify
lcssa
loop-rotate
branch-prob
block-freq
scalar-evolution
loop-accesses
loop-vectorize
instcombine
scalar-evolution
slp-vectorizer
```

```
simplifycfg
domtree
instcombine
loops
loop-simplify
lcssa
scalar-evolution
loop-unroll
instcombine
loop-simplify
lcssa
licm
scalar-evolution
alignment-from-
  assumptions
strip-dead-
  prototypes
elim-avail-
  extern
globaldce
constmerge
verify
```

This is all target-independent optimization only

104 Passes

* The exact list of passes used depends on the version and flags supplied

Back Ends

- A lot more stuff going on
- Major parts (first-to-last & most-to-least complicated)
  - ✦ Target-independent optimization
  - ✦ Target-specific optimization
  - ✦ Code generation

Covered in Lecture. May be on Final.

Part of Project

Outline

Introductions

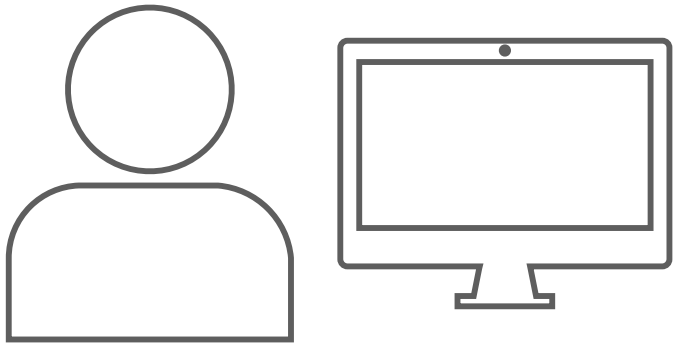
Administrivia

What's a compiler?

Why you might want to take this course

Demystify the Computing Stack

- Before you learned CS, your computer was run by the army of gremlins inside it.



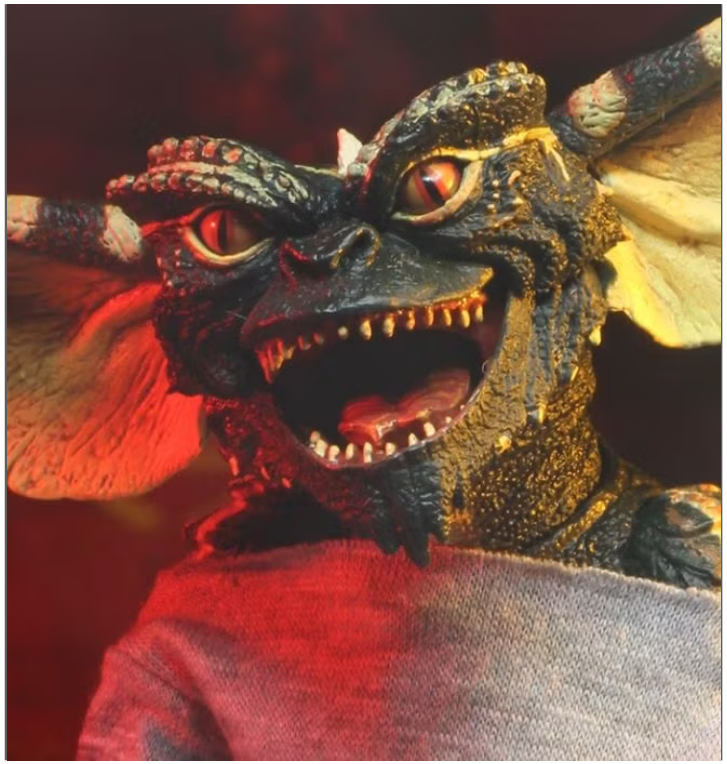
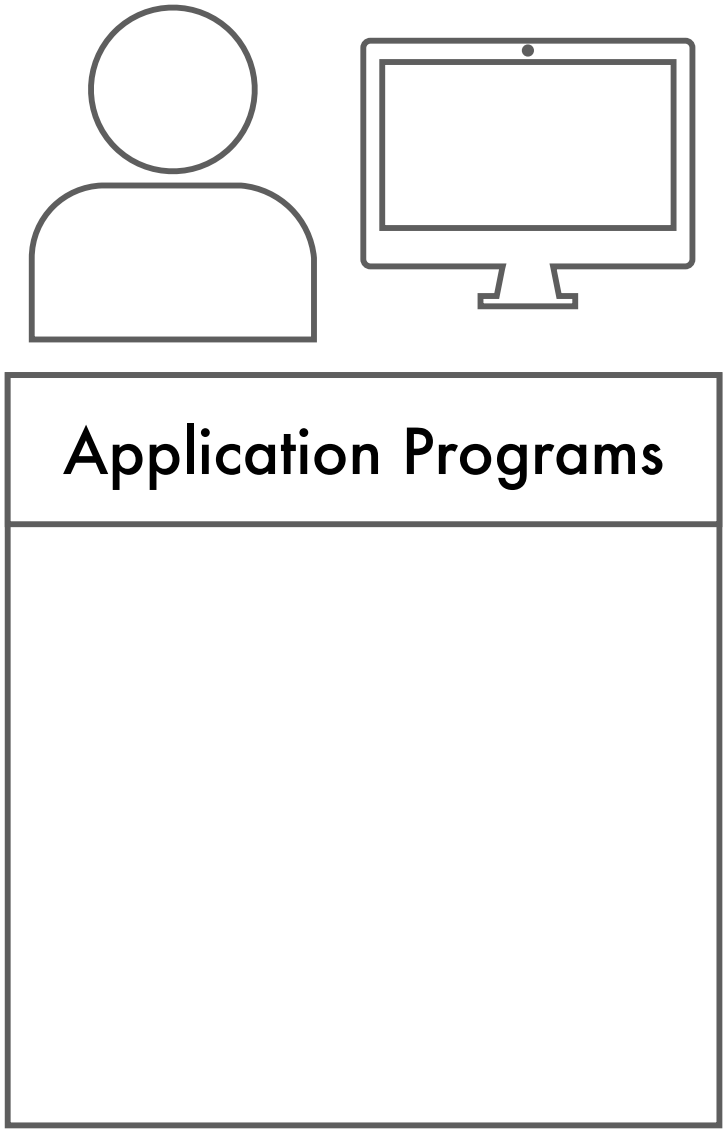


image: Gremlins movie

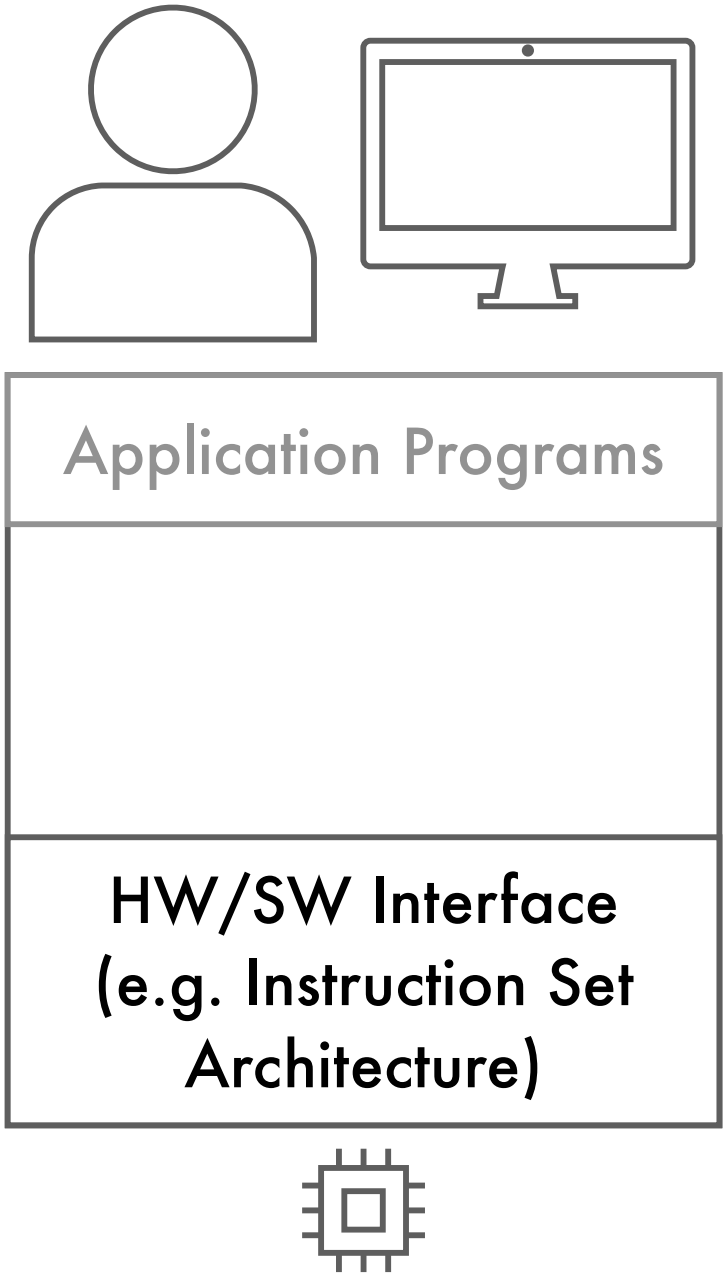
Demystify the Computing Stack

- Before you learned CS, your computer was run by the army of gremlins inside it.
- Intro courses demystified programming



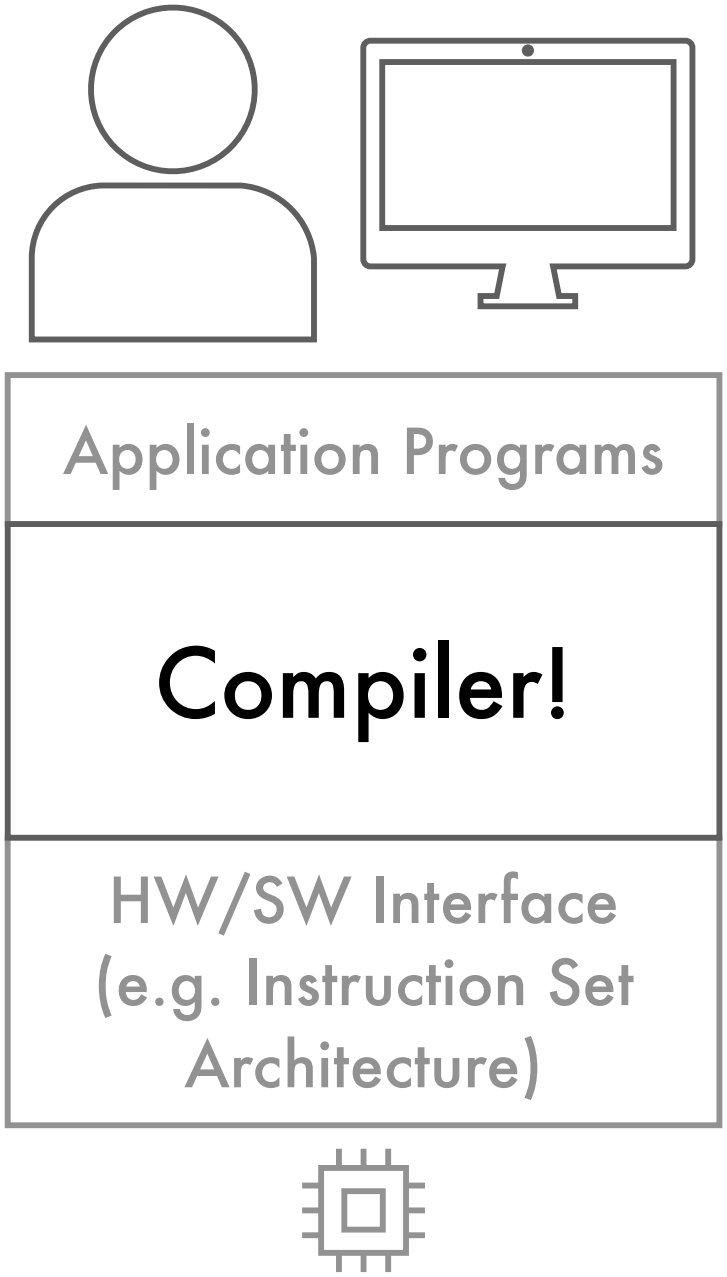
Demystify the Computing Stack

- Before you learned CS, your computer was run by the army of gremlins inside it.
- Intro courses demystified programming
- CSE 351 demystified the HW/SW interface



Demystify the Computing Stack

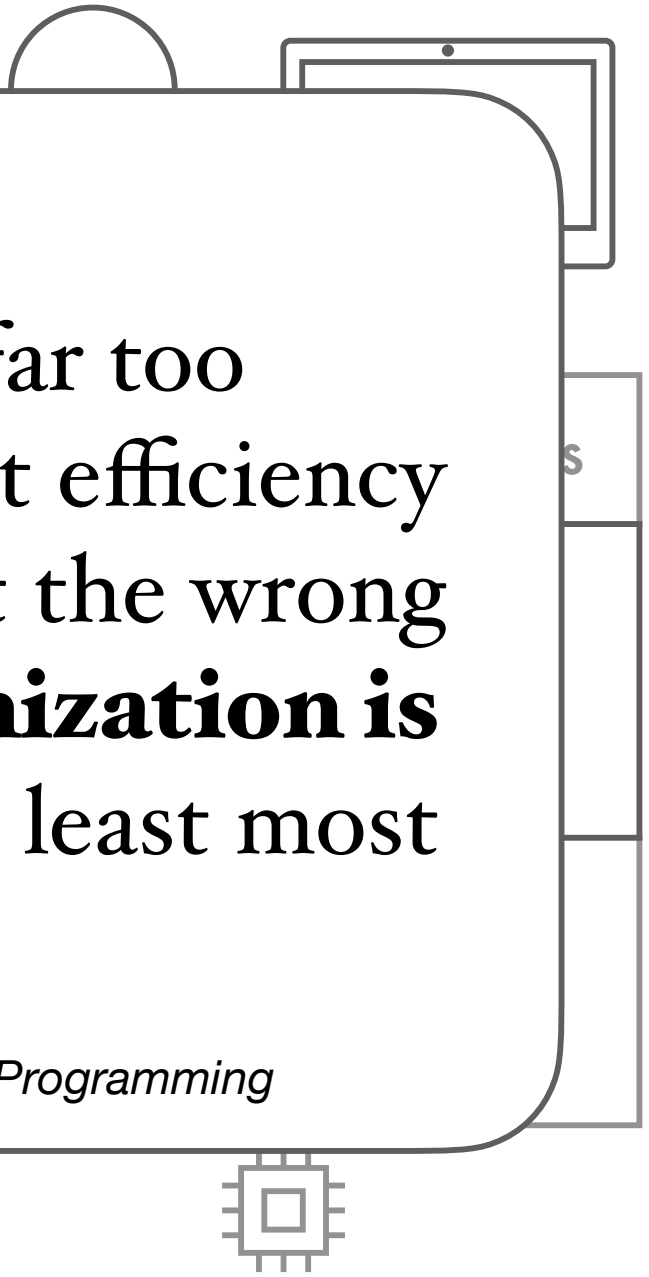
- Before you learned CS, your computer was run by the army of gremlins inside it.
- Intro courses demystified programming
- CSE 351 demystified the HW/SW interface
- BUT, those classes didn't demystify how high-level code turns into low-level code



Demystify the Computing Stack

- Before you learned CS, your
- In
- C
- B

> "The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; **premature optimization is the root of all evil** (or at least most of it) in programming."
>
> — *Donald Knuth, in the Art of Computer Programming*



Compilers as Prototype System

- A lot of other systems are (sometimes secretly) compilers
  - HCI - GUI Toolkits, Document Languages (pdf, TeX, etc)
  - Graphics - Shader languages, CUDA
  - Networking - Software-defined networks
  - Web - frameworks, HTML templates, the browser
  - ML/AI & Datascience - Tensorflow, PyTorch, Matlab
  - Databases - SQL, data analytics
  - Hardware - VHDL/Verilog
  - Computer Algebra Systems - Mathematica, SAGE

YOU will eventually write a Compiler!

- You ***might*** work on a big, serious compiler project one day.
  Many graduates from this class have!
- But, you ***will*** write simple parsers and interpreters for "little languages"
  - ✦ command languages, config files, XML, JSON, network protocols, semi-structured data, web templating, …
- If you ***enjoy*** working with compilers, there are many jobs available
  - ✦ ML/AI systems, data science/analytics, GPU programming, …

The Grand Tour!

- **Theory** — grammars, DFAs, PDAs, pattern matching, fixed-points
- **Algorithms** — graphs, dynamic programming, approximation, graph coloring
- **Systems** — allocation, scheduling, naming, synchronization, locality
- **Architecture** — instruction sets, pipelining, memory hierarchy
- **Engineering** — tradeoffs, large complex code bases, advanced testing

ok, a grand tour of computer science, not of the solar system

image: NASA

Questions?

Week 1

- **Monday** — This Lecture video
- **Wednesday** — Hal Perkins guest lectures
  - on Regular Expressions
  - ***start reading ch. 1, 2.1-2.4 (entire book available through Safari Online!)***
- **Thursday** — Sections as normal
  - review of Regular Expressions and HW1 released
- **Friday** — Gilbert back; lecture in person as normal
  - second half of regular-expressions/lexing

Before next class…

- Familiarize yourself with the course website
- Read the syllabus and academic integrity policy
- Find a partner!
  - ✦ (and you get to meet other people 😀)
- Show up to lecture in person on Wednesday, Friday; Lectures on Thursday

Course Credits

- An incomplete list of sources and ancestors of this course
  - UW CSE 401 (Chambers, Snyder, Notkin, Perkins, Ringenburg, Henry, …)
  - UW CSE PMP 582/501 (Perkins & others)
  - Rice CS 412 (Cooper, Kennedy, Torczon)
  - Cornell CS 412-3 (Teitelbaum, Perkins)
  - Many books (Appel; Cooper/Torczon; Aho, [[Lam,] Sethi,] Ullman [Dragon Book]; Fischer, [Cytron,] LeBlanc; Muchnick, …)
- Attributions will be on a best effort basis…