CSE431S – Translation of Computer Languages

- Course Overview
- Introduction

Doug Shook

Course Overview

- Everything regarding the course can be found on the website:

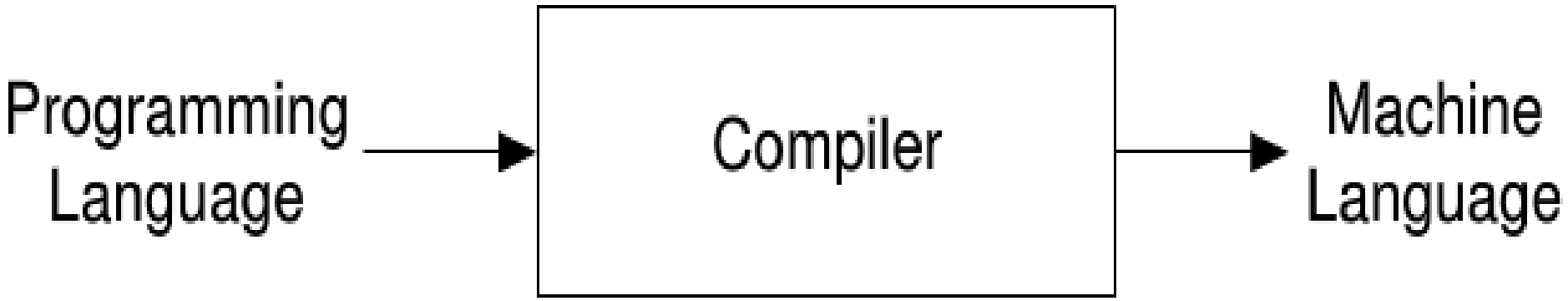
  http://cse.wustl.edu/~dshook/cse431

- If you have a question, please check the website first
  - Use Piazza!

Language Translation

- What is a language?
- Why is it necessary to translate it?
  - How does a computer see the world?
  - How does a human see the world?

What Do Compilers Do?



- Acts as a 'black box'
  - What goes on in there?

What Do Compilers Do?

```
void mm(int me_no, int noproc, int n, double a[NDIM][NDIM], double b[NDIM][NDIM], double c[NDIM][NDIM])
{
        int             i,j,k;
        double sum;
        i=me_no;
        while (i<n) {

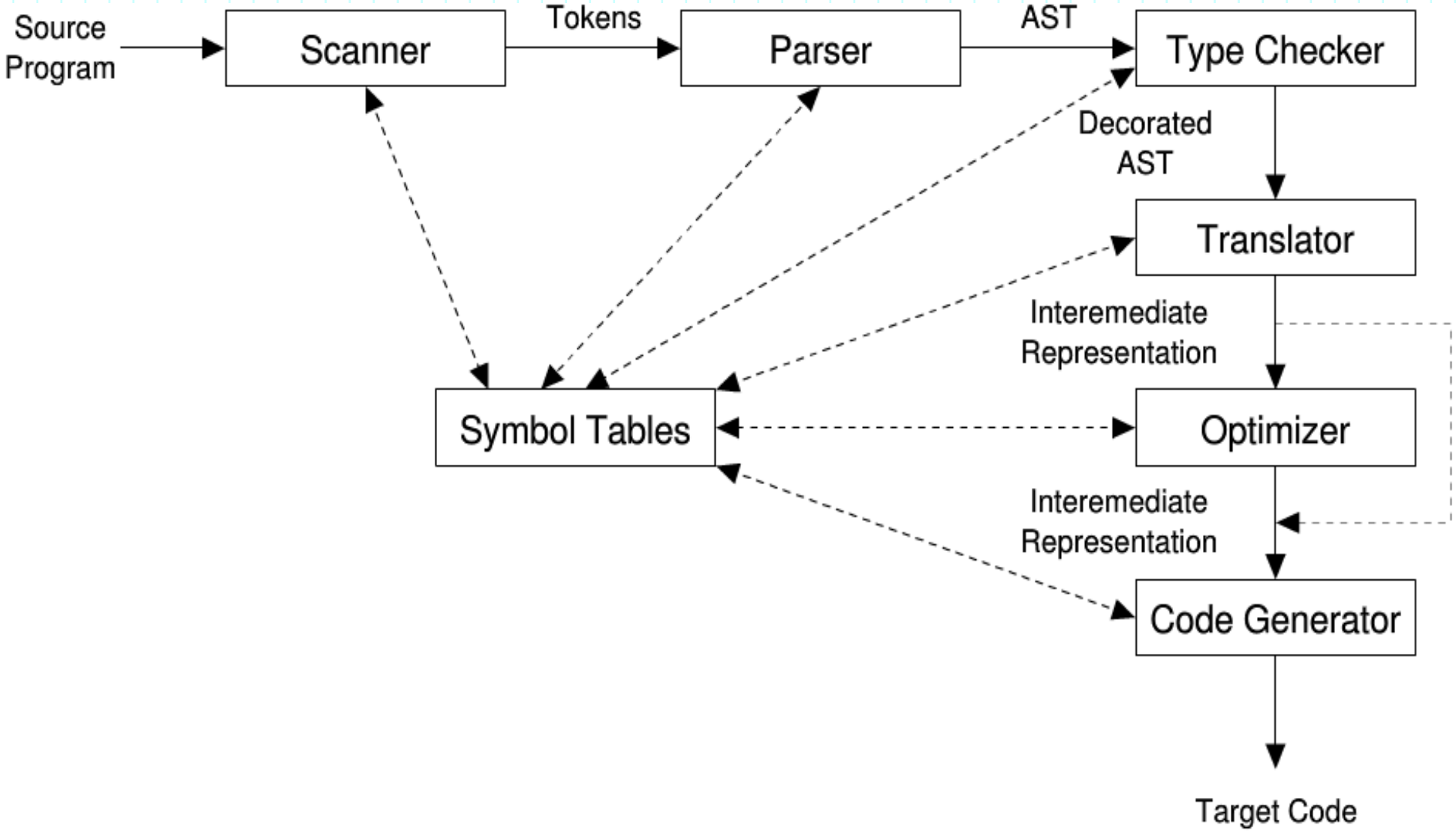
                for (j = 0; j < n; j++) {
                        sum = 0.0;
                        for (k = 0; k < n; k++) {
                                sum = sum + a[i][k] * b[k][j];
                        }
                        c[i][j] = sum;

                }
                i+=noproc;
        }
}
```

What Do Compilers Do?

```
0000000000400934 <mm>:
        push   %rbp
        mov    %rsp,%rbp
        push   %r14
        push   %r13
        push   %r12
        push   %rbx
        sub    $0x8,%rsp
      → callq  mcount@plt
        mov    %r9,%r13
        mov    %edi,%r14d
        lea    -0x1(%rdx),%eax
        lea    0x8(,%rax,8),%r11
        mov    $0x0,%r12d
      ↓ jmp    76
2d:     movsd  (%rdi,%rax,1),%xmm1
        mulsd  (%r10),%xmm1
        addsd  %xmm1,%xmm0
        add    $0x8,%rax
        add    $0x1f40,%r10
        cmp    %r11,%rax
      ↑ jne    2d
        movsd  %xmm0,(%rbx)
        add    $0x1,%r9d
        add    $0x8,%rbx
        cmp    %edx,%r9d
      ↓ je     73
5c:     movslq %r9d,%rax
        lea    (%r8,%rax,8),%r10
        mov    $0x0,%eax
        mov    %r12,-0x28(%rbp)
        movsd  -0x28(%rbp),%xmm0
      ↑ jmp    2d
73:     add    %esi,%r14d
76:     cmp    %edx,%r14d
      ↓ jge    9a
        test   %edx,%edx
      ↑ jle    73
        movslq %r14d,%rbx
        imul   $0x1f40,%rbx,%rbx
        lea    (%rcx,%rbx,1),%rdi
```

What Do Compilers Do?



Two Phases of a Compiler

- Analysis
  - Goal of the front end (scanner, parser)
  - Use *syntax* to break program into base components
    - What do we mean by syntax?
- Synthesis
  - Goal of the back end (Type Checker, Translator, Optimizer, Generator)
  - Generate code that will run on the desired platform
    - Must retain original *semantics*

Scanner

- Segments the input into *tokens*
  - For example:
    - 'i' 'n' 't' becomes int
    - Comments are stripped away
- Input: Source Code (C++, Java, etc.)
- Output: Tokens

Parser

- Use output from scanner to build an intermediate representation (IR)
  - Often times, an Abstract Syntax Tree (AST)
  - Generic form of the program
  - Driven by *grammars*
    - Define language syntax
- Input: Tokens
- Output: AST

Semantic Analysis

- Traverses the AST and verifies its meaning
  - For example: int + int = string?
  - Others?
- Decorates the AST with type information at each node
- Input: AST
- Output: Decorated AST

Translation

- The decorated AST only captures *static* semantics
  - Valid statements
- Translation captures *runtime* semantics by translating to an Intermediate Representation
  - This while loop statement is valid but...
- Input: Decorated AST
- Output: Intermediate Representation

Intermediate Representation

- Could be as simple as an AST
  - As complex as an entire language
- Important that it is as generic as possible
  - Allows for multiple front ends / back ends
    - Front end -> ?
    - Back end -> ?

Optimization

- Creates a more efficient form of the IR
  - Must retain original semantics!

- Examples:
  - Loop unrolling
  - Function Inlining
  - Machine specific optimizations

- Input: IR
- Output: IR

Code Generation

- Turns the IR into machine code that can be executed on the target device
- IRs are usually designed with this process in mind
- Input: IR
- Output: Machine Code

Why study compilers?

- Compilers have a HUGE impact on performance
  - Code never gets through "untouched"
- In some ways represent the line between software and hardware
- Pulls together many different fields
  - Computation
  - Machine Architectures
  - Data Structures
  - Many more

A Puzzle







- What is the solution?
- How can we represent the solution?
  - Keep it simple!
  - Hint: Try to encode your moves somehow