Compiler Construction Lab:
Introducing Compiler Framework through LLVM

Prof. Kunal Korgaonkar, Prof. Soumyadip Bandyopadhyay
and Prof. Ramprasad Joshi

Computer Science and Information Systems
Bits Pilani Goa Campus, India

My Introduction

Academics:

- MS (By Research) – IIT Madras, India
- PhD – UC San Diego, US
- Posdoctoral – Technion, Israel

Industry Research:

- IBM Research
- Intel Labs
- AMD Research

Industry Products:

- DG2L – Pune Based Tech Startup
- Intel – Heterogenous System Design
- NSE - Core Trading Servers

About me:

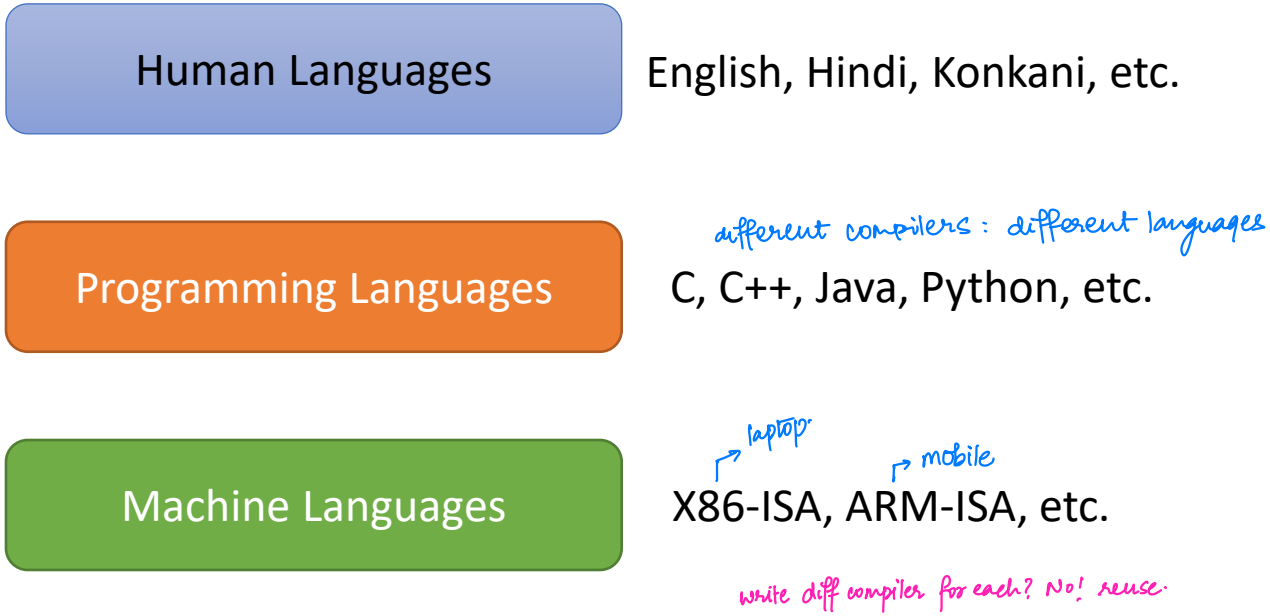
- Goan Native
- Lived at several cities in India and US
- Love travelling

Outline

- Need for Compiler Framework
- LLVM as an example
- Components of a Compiler Framework
  - Front Ends
  - Intermediate Representations
  - Back Ends
- Further Readings
- Conclusion

Why do you need a Compiler

- **Compilers help bridge the gap between programming languages and machine languages**
- For instance, they translate the lines of code written in one programming language to a set of instructions on a target machine
- Instructions are as per the instruction set architecture (ISA) language of the target machine

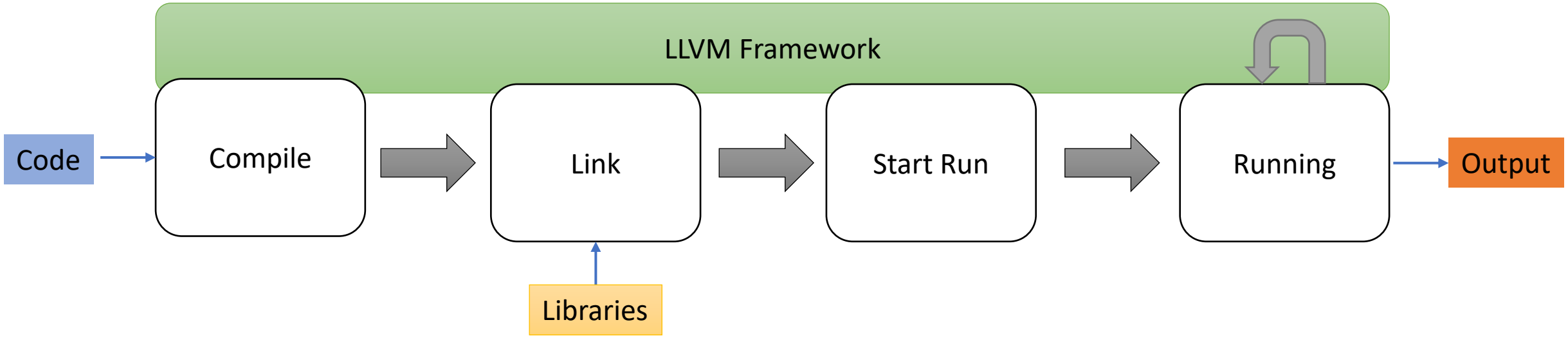


LLVM Compiler Framework

- Origins at Academia: UIUC, US
- Started by Vikram Adve (Professor) and Chris Lattner (then Student)

Features of LLVM

- **It supports multiple programming languages as input and multiple machine target executables as output**
- Very modular and component based design
- Language-agnostic design; that is, it can generate code for any target language and instruction set architecture
- Code can be optimized at any stage; That is, at compile-time, at link-time, before run-time or even during run-time



Components of LLVM - **Front Ends**

- Support for languages such as Asa, C, C++, D, Delphi, Fortran, Haskell, Julia, Objective-C, Rust and Swift
- Uses a sub-tool called **Clang** within LLVM
- Syntax processing is a large part of front end
- To the right is a simple example of syntax parsing (tree)

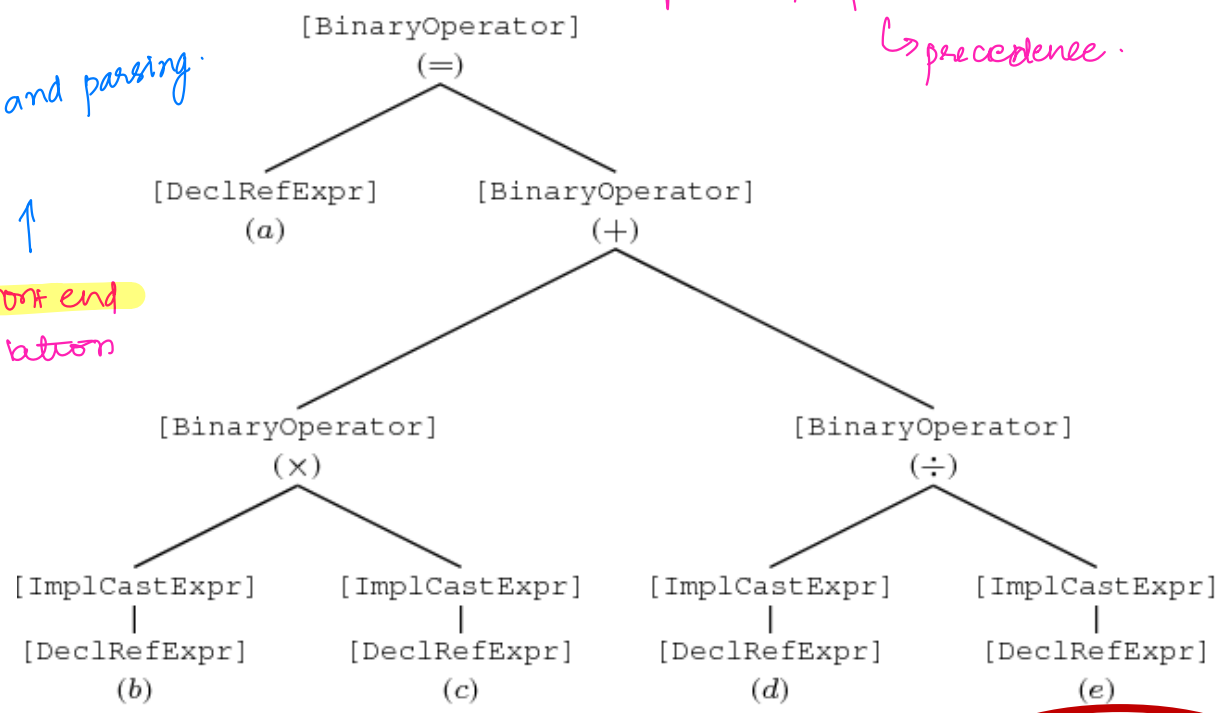
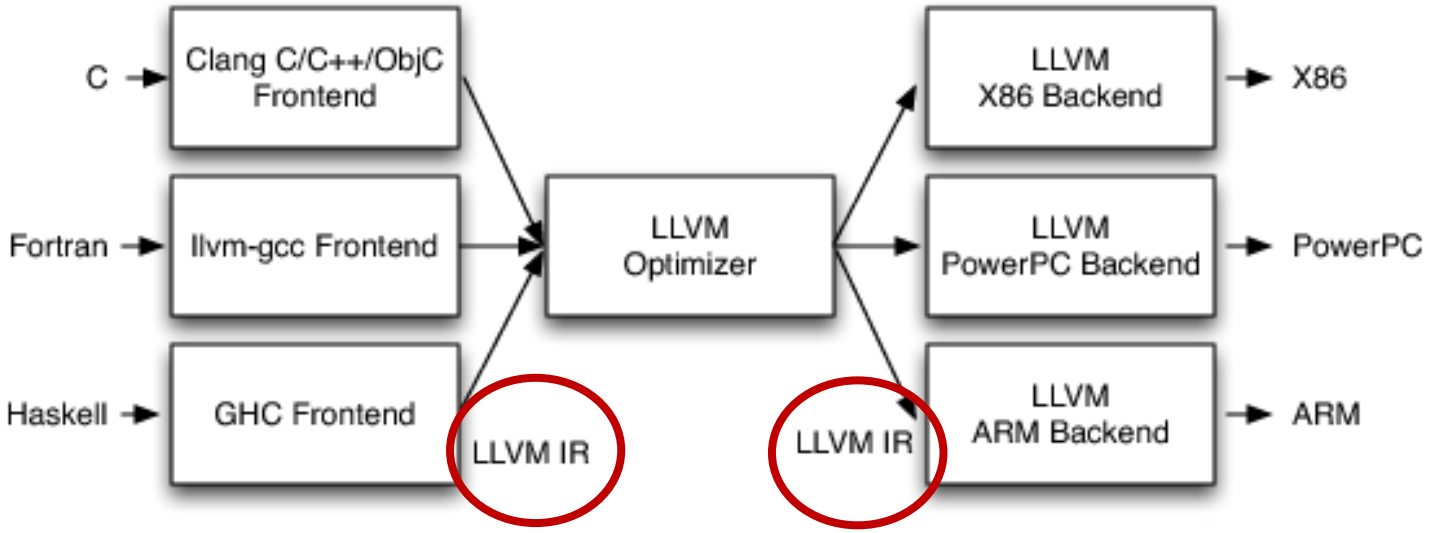


Fig. 1. Example of abbreviated Clang AST for the expression $a = b \times c + d \div e$.

Components of LLVM - **Intermediate Representation**

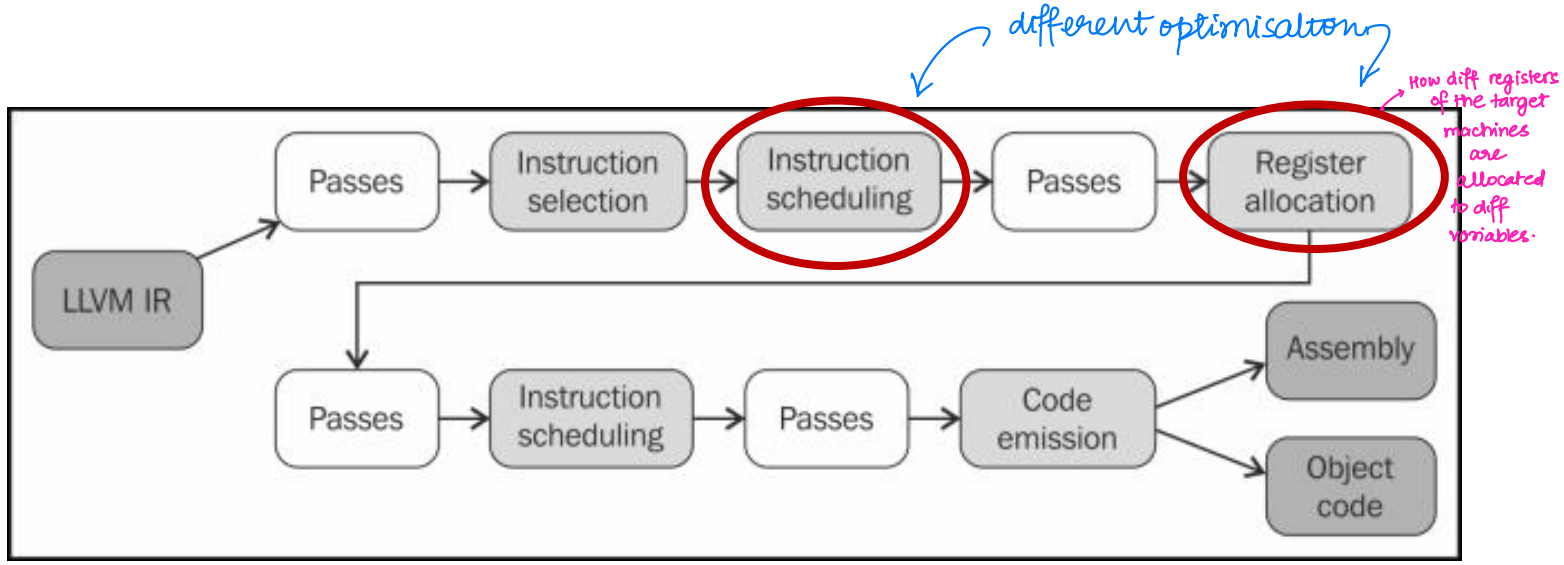
↳ goes to optimization.

- LLVM produces **LLVM IR (Intermediate Representation)**
- Common Representation closer to assembly and a reduced instruction set (RISC style)
- IR used at many levels; other representations such as Data Flow Graph are also present in the framework



Components of LLVM - **Back Ends**

- Targets can be
  - **CPUs** (x86, ARM, etc.)
  - **GPUs** (NVIDIA PTX – Parallel Thread Execution) and
  - **Web Browsers** (webassembly)
- Optimization passes and finally code emission
- Examples of passes for register allocation or for instruction scheduling



Some References – **Must Read!**

- The Architecture of Open Source Applications, Chris Lattner https://www.aosabook.org/en/llvm.html
- LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation, Chris Lattner and Vikram Adve https://llvm.org/pubs/2003-09-30-LifelongOptimizationTR.pdf

Conclusion

- Compilers are one of the most important parts of the computing industry
- Compilers convert human labor and creativity into something that can run and be alive for you and work for you – the various programs
- Apart from an understanding of programming, it is important to have an understanding on what goes on inside the compiler too
- Tools like LLVM can help you achieve this goal
- LLVM is also a great example of what can be done achieved inside a university setting
- Welcome to the “The Compiler Construction Lab”!