An LLVM back end for sourir

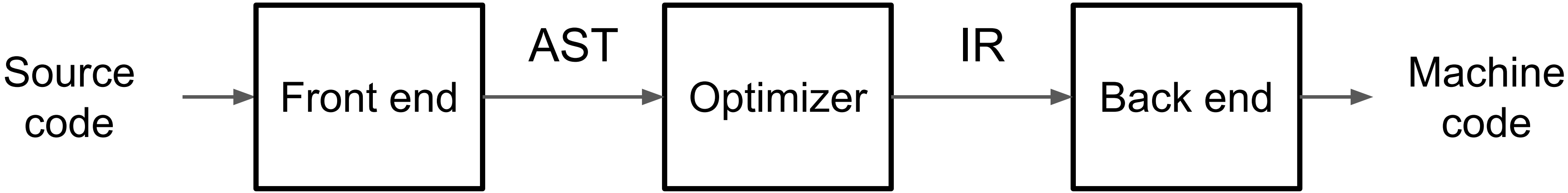
Stefan Borer
Mentor: Oliver Flückiger

Master Seminar Project @ SCG, University of Bern, FS 17

Outline

1. Compiler introduction
2. What is sourir and why?
3. Quick dive into LLVM
4. Sourir architecture
5. Demo
6. Conclusion
7. Future work

Traditional Compiler



- Modern compilers: multiple passes in optimizer

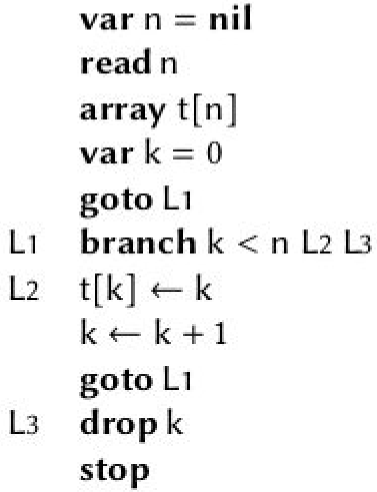
To be continued...

Sourir

Low-level programing language or
High-level intermediate representation

- Primitive datatypes, no Classes, Objects
- Program flow using labels, goto and branch
- Consists of functions

- Functions can have **version**’s
- **assume** instruction



```
     var n = nil
     read n
     array t[n]
     var k = 0
     goto L1
L1   branch k < n L2 L3
L2   t[k] ← k
     k ← k + 1
     goto L1
L3   drop k
     stop
```

Dynamic programming languages

At runtime:

- Loading of new code
- Extension of objects and definitions
- Often dynamically typed

- Use just-in-time (JIT) compiler
  → continuously iterate and dump code at latest possible time
- Optimize code based on speculations (assume types etc.)

Eg. Javascript, Smalltalk, PHP, Python

Sourir

- Designed as IR for dynamic languages
- Explicit versions of functions
- Explicit assumptions
  → Easier to reason about optimizations / deoptimizations

- out of the scope for today

LLVM

Originally: **Low Level Virtual Machine**
Today: “collection of modular and reusable compiler and toolchain technologies"

- Written for C, C++ but with language-agnostic design
- Front ends: D, Fortran, Objective-C, Python, R, Rust, …
- Back ends: x86, x86-64, PowerPC, MIPS, ARM, AMD GCN, …
- Linker, machine code translator, C++ standard library, Debugger, …
- LLVM IR
- Optimization using passes
- Offers a JIT

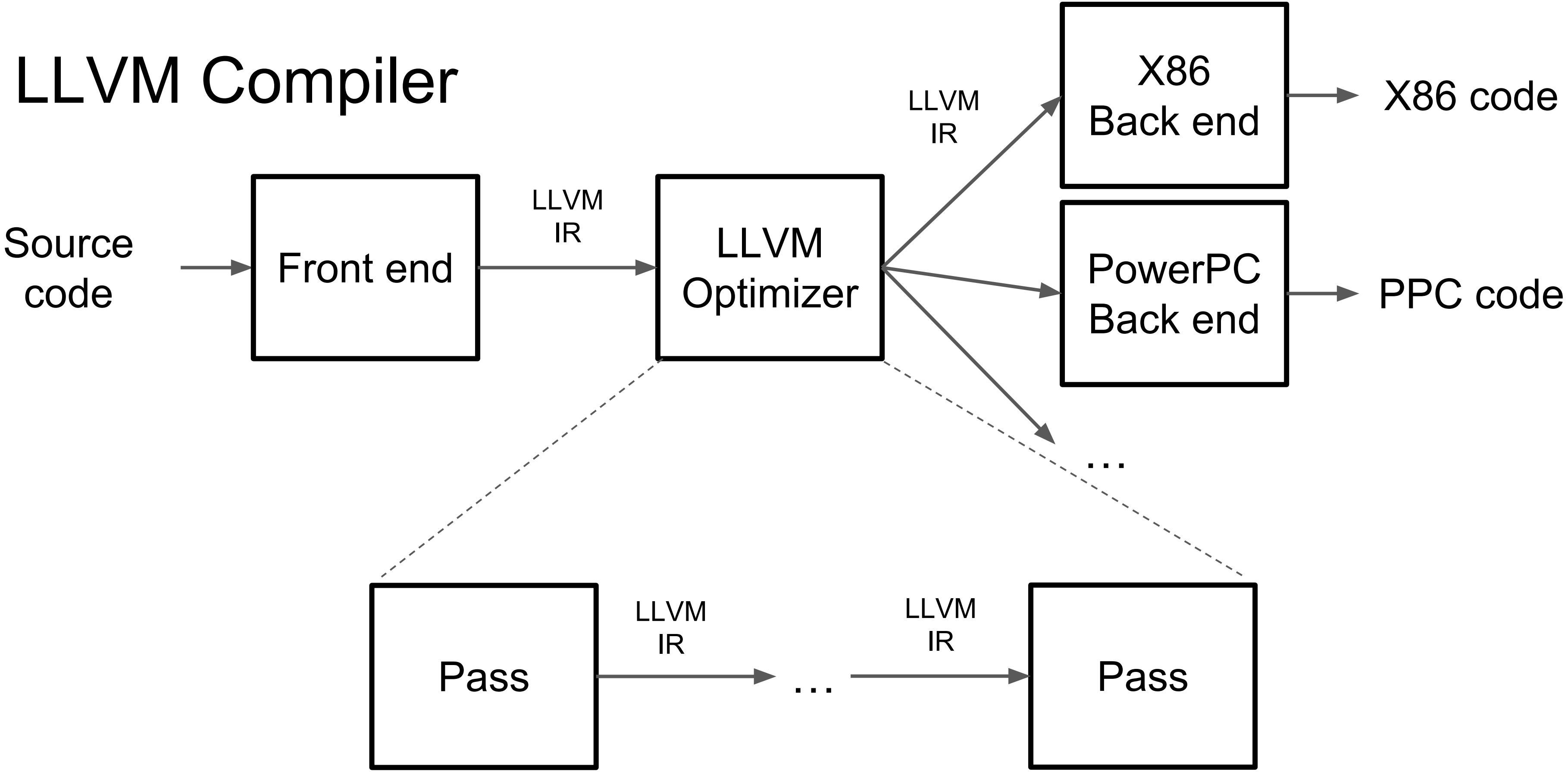
LLVM IR

- Heart of LLVM
- Strongly typed RISC instruction set
- Infinite set of registers
- Static single assignment (SSA) form

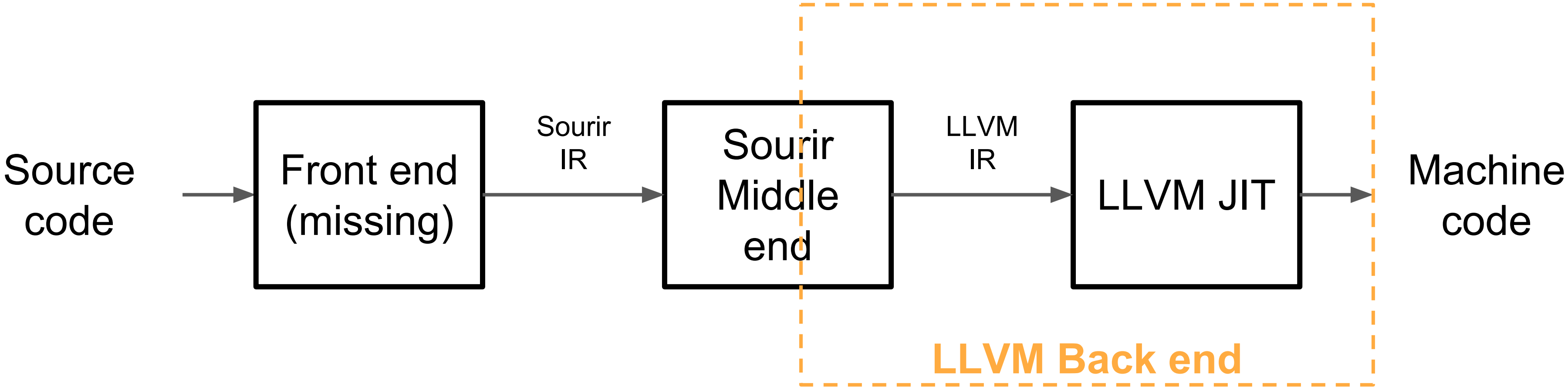
Three equivalent forms:

- C++ object format
- Plain text (assembly)
- bitcode

LLVM Compiler



Sourir JIT


Source code
Front end (missing)
Sourir IR
Sourir Middle end
LLVM IR
LLVM JIT
Machine code
LLVM Back end

Demo

Conclusion

LLVM

- handy and fast
- But: “official” support doesn’t mean good documentation

Ocaml

- Function programming is fun
- But: irritating syntax
- Inconvenient setup

Future work

- Basic features left: arrays, print/read, drop, booleans
- Advanced: version, assume
- Optimization as LLVM passes
- Front end for high level language