Beyond The Compiler: Advanced Tools for Better Productivity

Gábor Horváth
xazax.hun@gmail.com

Agenda

- **What are compilers? Where to find them?**
- **The high level picture of compiler development**
- **Tools, you should use**
- **Demo, Demo, Demo, Demo, Demo, …**
  - Things will break, be gentle

Compilers

- **Compile from Source Language to Target Language**
  - C++ → X86, Haskell → C
- **The further the two languages the more complex the compiler**
- **Complex software utilizing almost every area of Computer Science**

Where can we find compilers?

- **Native languages, interpreted languages (bytecode, JIT), preprocessing**
- **Machine Learning**
  - TensorFlow AXL
- **Big Data**
  - Apache Flink: Serialization
  - Apache Spark
- **Browsers, configurations**
- **GPU Drivers**

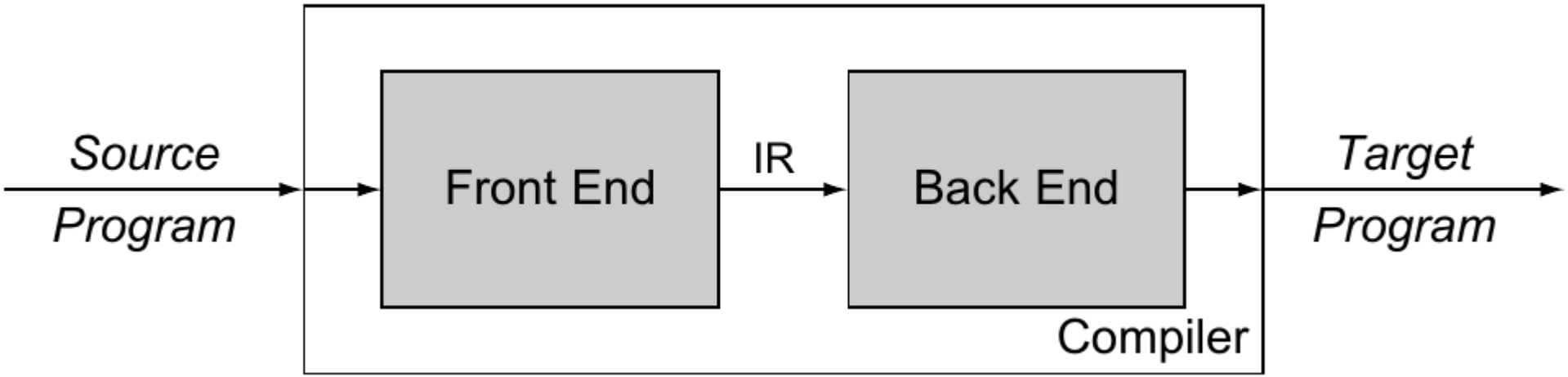
Complexity of a Compiler

- **Most of CS theory in practice**
  - Greedy algorithms (register allocation)
  - Heuristic search (instruction scheduling)
  - Graph algorithms (dead code elimination)
  - Deterministic automaton (lexing)
  - Fixed point algorithms (dataflow analysis)
  - Math: lattice, number theory, graph reduction, …

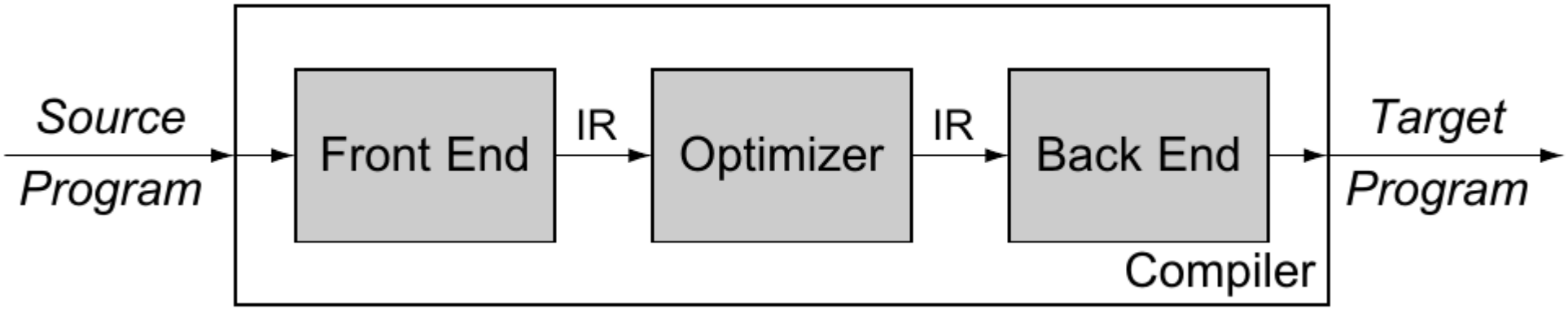
Trade-off

- Register usage vs run time
- Code size vs run time
- Time spent optimizing vs time spent running
- Security vs performance

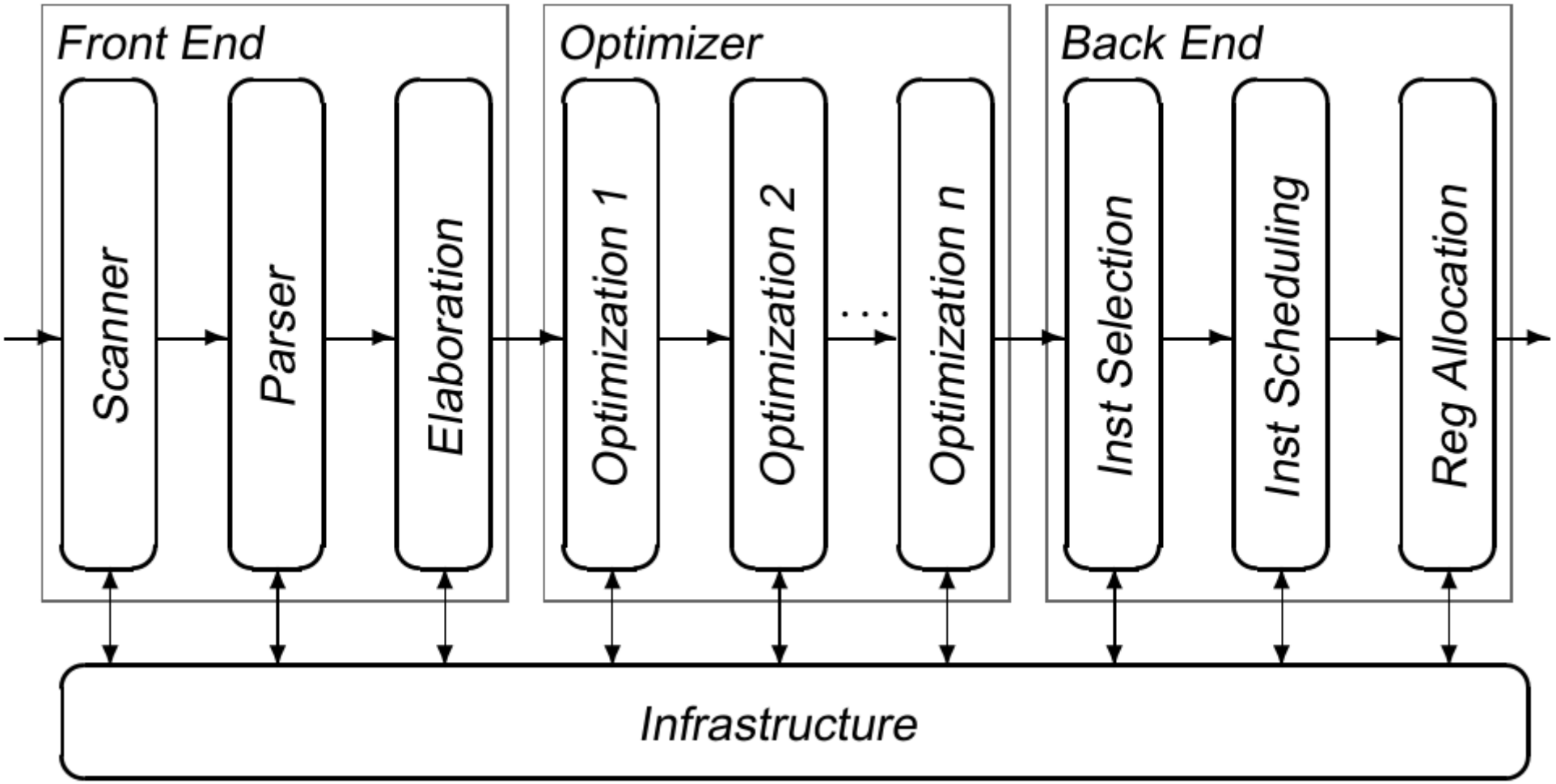
Architecture



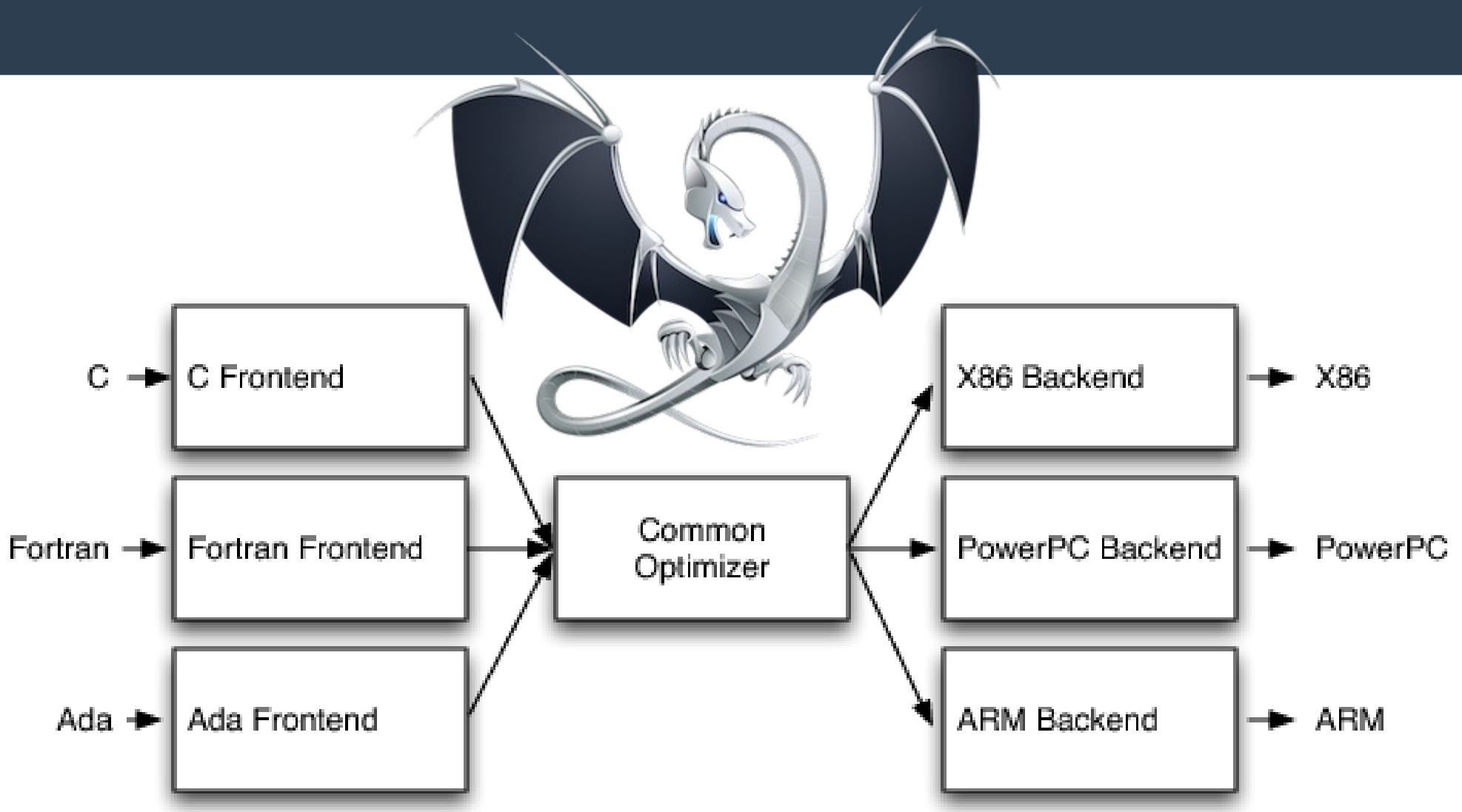
Architecture



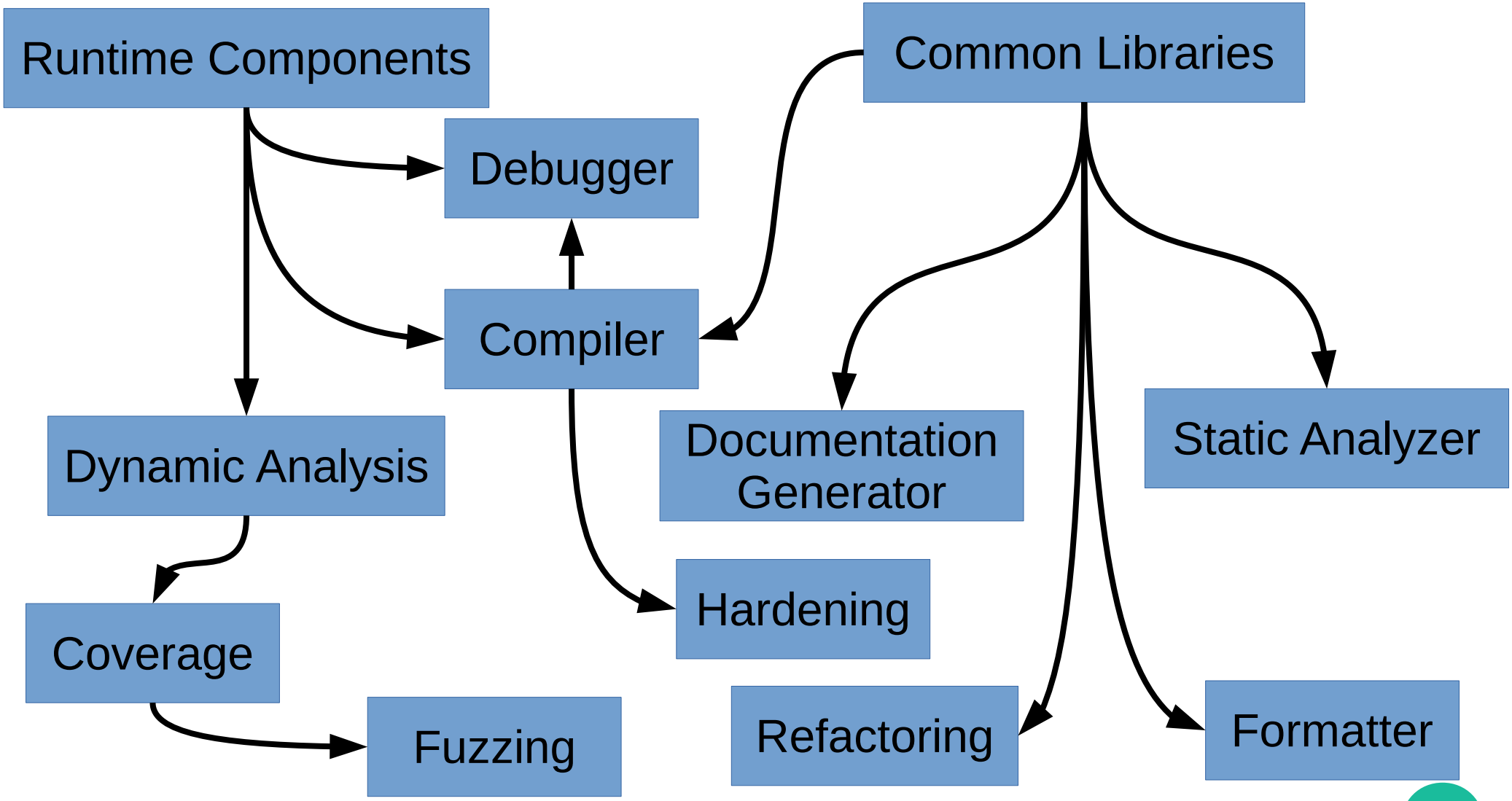
Architecture



Architecture – LLVM



Architecture – For Real



What are the Expectations?

- **Demo (inc.cpp, fact.cpp, loop.cpp)**

- **To generate correct code**
- **To generate fast code**
- **Help to find bugs**
  - Advice: -Wall -Wextra

Undefined behavior

- Demo (undefined.cpp)

- How scary it is?
- How to detect it?
- The dark middle ages are (almost) over

Static Analysis

- **Analyze the code without running it**
- **Formatting**
- **Optimization**
- **Warnings**
  - First line of defense
  - Enable them and keep the build clean!
- **Clang-tidy**
  - Errors, Performance Smells, Refactoring
- **Clang Static Analyzer**

Clang Format Demo

- **Demo (format.cpp)**
- **Reflow comments**
- **Reflow string literals**
- **Respect macros**
- **Respect column limits**
  - Tex for code
- **Productivity boost**

Clang Static Analyzer Demo

- **Demo (memory.cpp)**

- **Deep analysis**
- **Takes more time than compilation**
- **Sources of unknown**
  - Conservative approach

Dynamic Analysis

- **Sanitizers**
  - Memory Sanitizer
  - Address Sanitizer
  - Thread Sanitizer
  - Undefined Behaviour Sanitizer
  - Bonus!
- **https://www.youtube.com/watch?v=V2_80g0eOMc**

Sanitizer Demo

- **Demo**
  - memory.cpp, thread.cpp, ubsan.cpp, dead.cpp

- **Occasionally run tests with sanitizers**
- **Need good coverage**
- **Not all platforms are supported**
- **Not all combinations are supported**
- **Optimizations can backfire!**

Sanitizer Idea

- **If the compiler can optimize upon, the sanitizer find the issue**
  - Not quite there yet
- **If the compiler does surprising optimization and the sanitizers do not find it**
  - Report a bug: it is either a sanitizer or a compiler bug

How good is your coverage?

- **Sanitizers only useful with reasonable coverage**
- **Should have a way to measure it**
- **Multiple coverage measures can be defined**
  - Line, statement, branch, condition, path, ...
- **SanitizerCoverage**
  - Used with tools
- **GCOV like coverage, compatible with gcc**
- **Source based coverage**

Source Based Coverage Demo

- **Demo (memory.cpp)**

- **Usage**
  - clang++ -fprofile-instr-generate -fcoverage-mapping a.cpp
  - LLVM_PROFILE_FILE="a.profraw" ./a.out
  - llvm-profdata merge -sparse a.profraw -o a.profdata
  - llvm-cov show ./a.out -instr-profile=foo.profdata

Coverage Sanitizer Demo

- **Demo (memory.cpp)**

- **Edge, BB, Func**
  - clang++ -fsanitize-coverage=func a.cpp
  - UBSAN_OPTIONS=coverage=1 ./a.out
  - sancov a.sancov -not-covered-functions a.out

Security

- **Optimization + Security ?!**
- **Demo (memset.cpp)**

- **What is the solution?**
  - memset_solution.cpp

Security!

- **http://blog.quarkslab.com/clang-hardening-cheat-sheet.html**
- **Checked Memory functions**
  - -D_FORTIFY_SOURCE=2
- **Address Space Randomization**
  - -fpie -pie
- **Additional Protection**
  - -fstack-protector, -fsanitize=safe-stack, -fsanitize=cfi
  - -Wformat -Wformat-security -Werror=format-security

Fuzz Testing!

- **Did we cover the critical cases?**
- **Measure the coverage, and create input to increase it!**
  - Coverage-guided, evolutionary
- **http://llvm.org/docs/LibFuzzer.html**
- **Sanitizers + Fuzzing = <3**
- **Heartbleed within minutes**
  - https://www.youtube.com/watch?v=qTkYDA0En6U

Lambda Calculus

```
expr =  var
        | "λ", var, ".", exp
        | expr, " ", expr
        | "(", expr, ")";
```

- **Haskell like, dynamicly typed language**
  - Church, 1930
- **λx.λy.x - True**
- **λp.λa.λb.p a b – If p Then a Else b**

Fuzz Testing Demo

- **Demo**
  - Lambda

Fuzz Testing

- **Harder for multi layered application**
  - E.g., testing sema
- **Harder to test correctness**
  - Semantics preserving transformation on inputs
  - Behavioral equivalence
- **Can use initial corpus**
- **The fuzzed application should be deterministic and preferably fast**
  - To be coverage guided

Link Time Optimization

- Usually the larger context available the more optimization possibilities can be found by the compiler
- Translation units are a natural barrier
- Needs more memory, less scalable
- Can work across language boundaries
- Needs linker plugin support, key flag: -flt
- http://hubicka.blogspot.hu/2014/04/linktime-optimization-in-gcc-2-firefox.html

Profile Guided Optimization

- **Instead of static estimates use run time statistics for cost models (still estimation)**
- **Sampling vs Instrumentation**
  - Completely separate methods
  - Sampling needs debug symbols (at least line tables)
  - Some formats need conversion
- **Instrumentation**
  - clang++ -O2 -fprofile-instr-generate a.cc -o a
  - LLVM_PROFILE_FILE="a-%p.profraw" ./a
  - llvm-profdata merge -output=a.profdata a-*.profraw
  - clang++ -O2 -fprofile-instr-use=a.profdata a.cc -o a

Big Picture

- **Turn compiler warnings on and keep the build clean**
- **Run clang format in the editor**
- **Occasional test runs with sanitizers**
  - Record coverage precisely
- **Occasional runs of Static Analyzer**
- **Occasional runs of Clang Tidy**
- **Develop fuzzing for security/mission critical parts**
- **Run LTO and PGO on release builds only**
- **Track false positives, differential view**
  - https://github.com/Ericsson/codechecker

There are more tools beyond the compiler! :)

Thanks for the attention!
Questions?

Bonus!

- **Write standard conforming code!**
- **http://blog.regehr.org/archives/1307**
- **Upcoming Type Based Alias Analysis sanitizer**