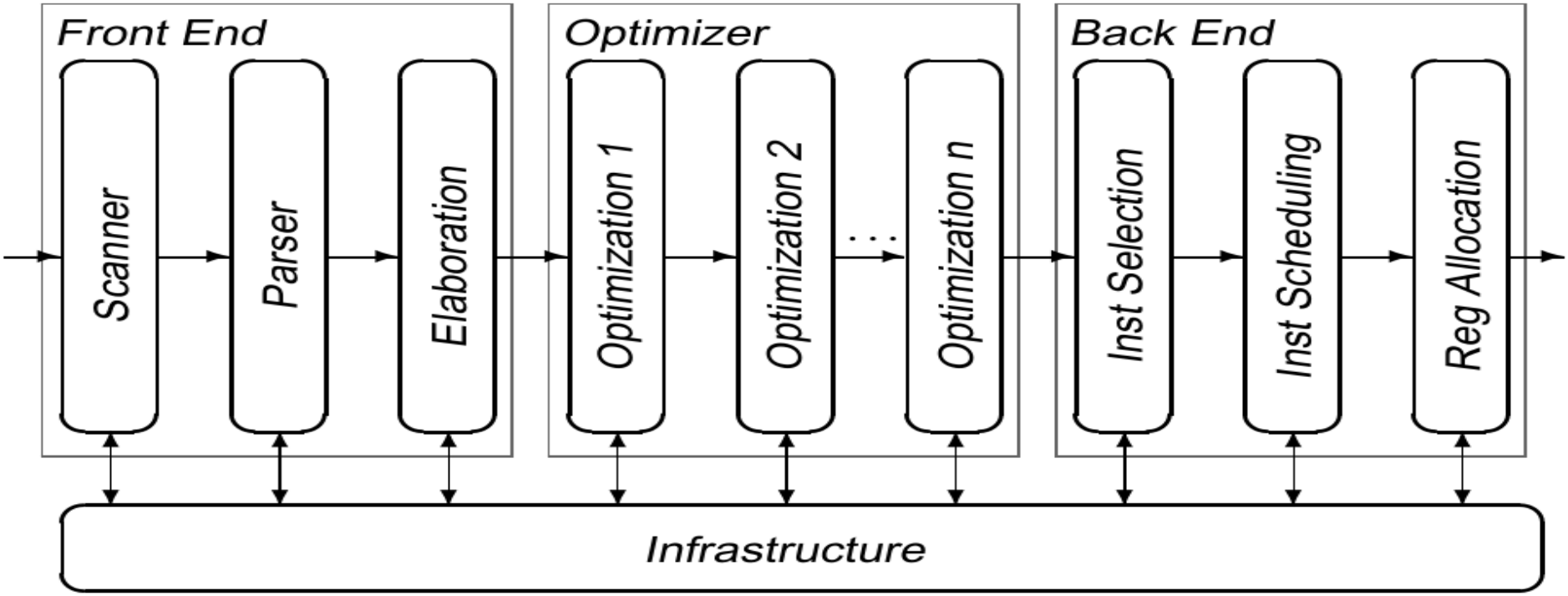
Beyond The Compiler: Advanced Tools for Better Productivity

Gábor Horváth (xazax.hun@gmail.com)

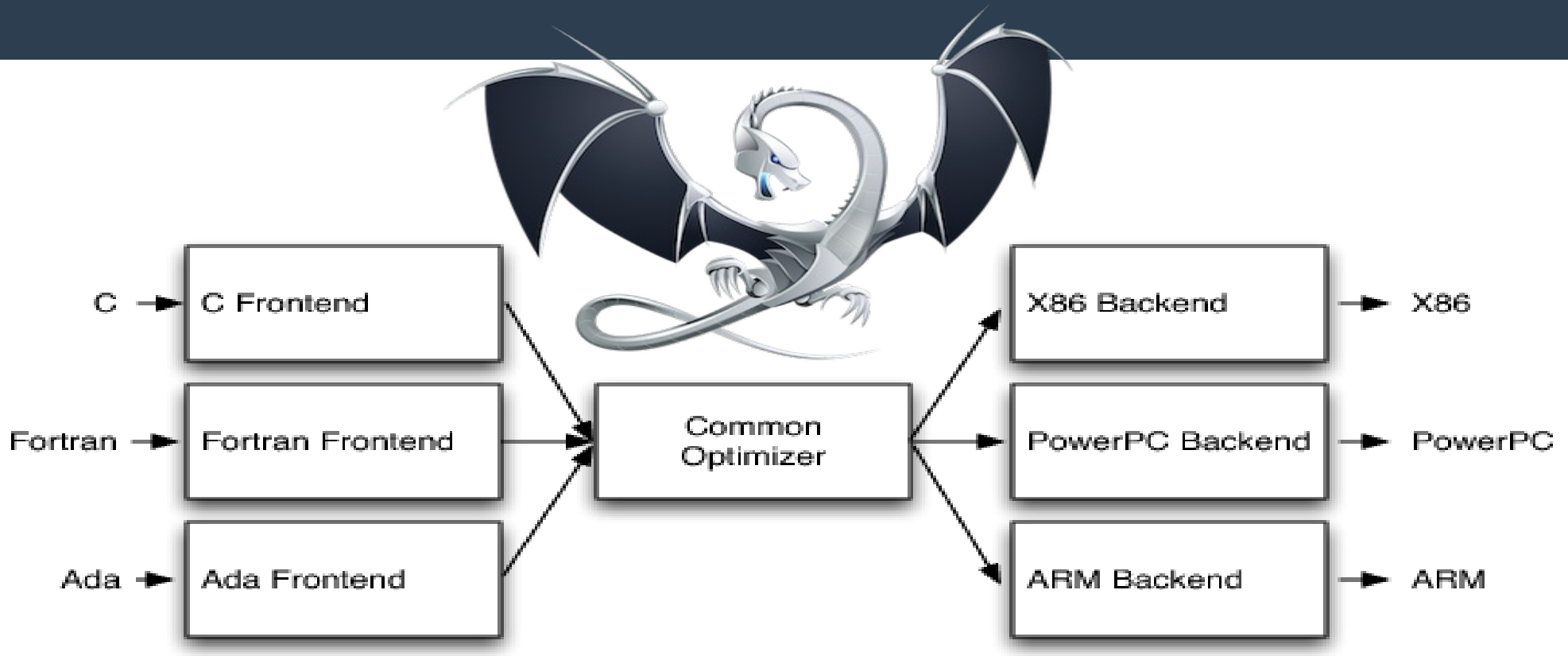
Where can we find compilers?

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

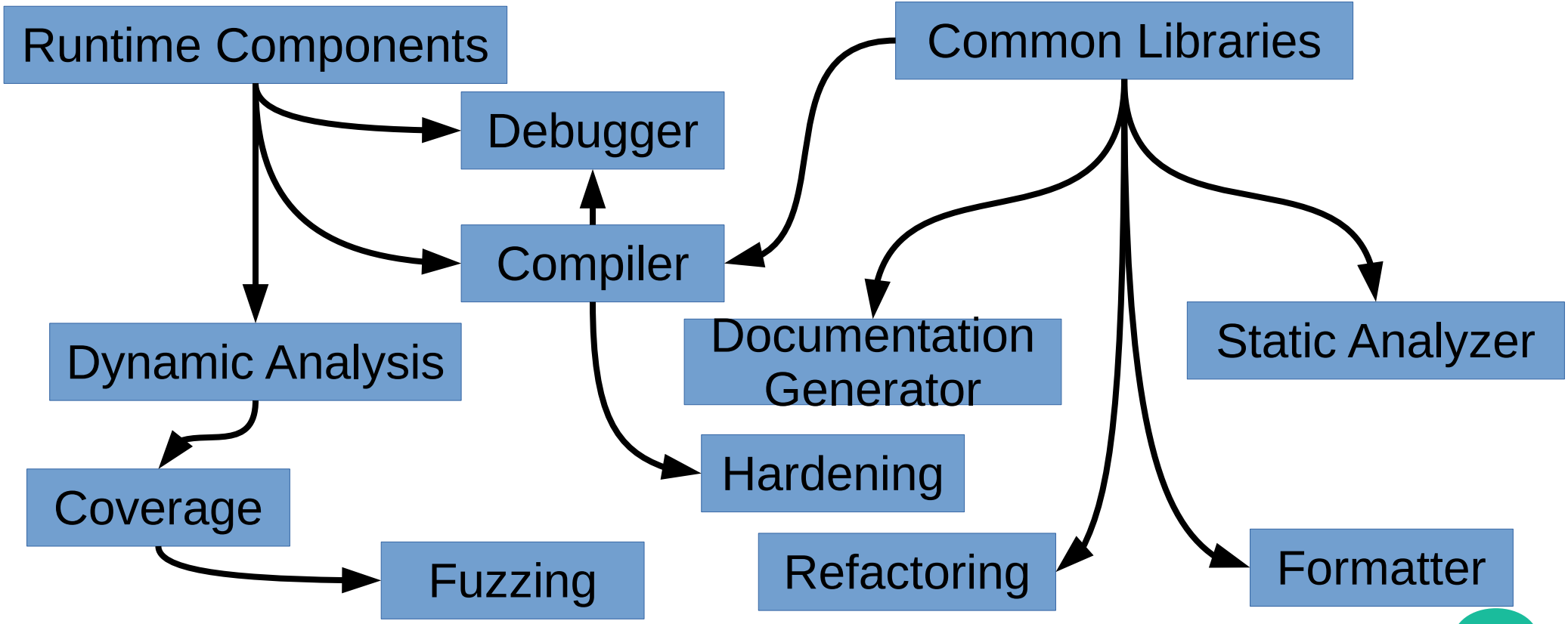
Architecture



Architecture – LLVM



Architecture – For Real



Why do we need other tools?

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

- Deal with complexity
- Undefined behavior is not the problem
- No UB → fewer optimizations

Undefined behavior

- Demo (undefined.cpp)

- How scary it is?
- How to detect it?
- https://blog.regehr.org/archives/1520

Static Analysis

- Analyze the code without running it
- Formatting
- Optimization
- Warnings
- Clang-tidy
- Clang Static Analyzer
- CppCheck

Clang Format Demo

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

Warnings

- Demo (annotation.cpp)

- Warnings are helpful
- Keep your build clean!
- Utilize annotations

Clang Tidy Demo

- Demo (tidy.cpp)

- Code smells
- Performance problems
- Refactoring
- Modernizing

Clang Static Analyzer Demo

- **Demo (memory.cpp)**

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

CppCheck

- Demo (cppcheck.cpp)

- Easy to use
- Less precise
- Ability to describe APIs

Coverity Scan

- **Free for open source projects**
- **https://scan.coverity.com**

Dynamic Analysis

- **Sanitizers**
  - Memory Sanitizer
  - Address Sanitizer
  - Thread Sanitizer
  - Undefined Behaviour Sanitizer
  - Bonus, if we have time
- **https://www.youtube.com/watch?v=V2_80g0eOMc**

Sanitizer Demo

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

- **Run tests with sanitizers**
- **Have good coverage**
- **Not all platforms are supported**
- **Not all combinations are supported**
- **Optimizations can backfire!**

How good is your coverage?

- **Sanitizers need reasonable coverage**
- **Measure!**
- **Multiple coverage measures exists**
  - Line, statement, branch, condition, path, ...
- **SanitizerCoverage for tools**
- **GCC compatible GCOV coverage**
- **Source based coverage**

Source Based Coverage

- **No demo this time, we save time!**
- **Usage example**
  - 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=a.profdata

Security

- **Optimization + Security ?!**
- **Demo (memset.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?**
- **Generate tests to increase the coverage**
  - Coverage-guided, genetic algorithm
- **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**

PGO and LTO

- **PGO use run time statistics for cost models instead of static estimates**
  - https://clang.llvm.org/docs/UsersManual.html#profile-guided-optimization
  - Sampling needs debug symbols (at least line tables)
  - Instrumentation more precise, better results, more overhead
- **LTO can optimize across translation units**
  - https://clang.llvm.org/docs/ThinLTO.html
  - https://llvm.org/docs/LinkTimeOptimization.html
  - ThinLTO can scale

Perf

- Useful command line utility
- Find hotspots
- Find the reason of hot spots

- perf stat -B -e cache-references,cache-misses,cycles,instructions,branches,faults,migrations ./a.out
- perf record ./a.out && perf report

Compiler explorer

- **https://godbolt.org/**

Code Comprehension

- **Understanding code is different activity than writing**
- **https://github.com/Ericsson/CodeCompass**
- **http://modelserver.inf.elte.hu:34540/#wsid=llvm**

Big Picture

- **Warnings on and keep the build clean**
- **Automate the formatting**
- **Use sanitizers, measure coverage**
- **Use static analysis tools, fix warnings**
- **Use refactoring tools to modernize the code**
- **Develop fuzzing for mission critical parts**
- **Run LTO and PGO on release builds only**
  - Might give you the performance budget for hardening!
- **Track false positives, differential view**
  - https://github.com/Ericsson/codechecker

And this is just scratching the surface! :)

Thanks for the attention!
Questions?