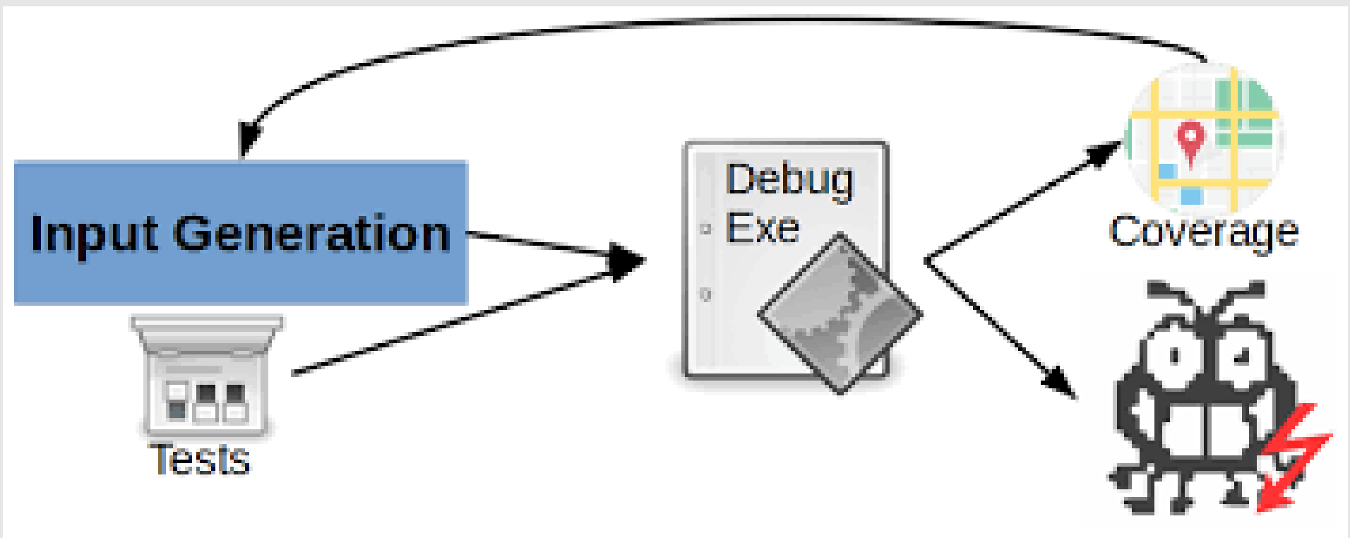
Intel Corporation Red Team Internship

Mitchell Smith
Mitchell.smith132@gmail.com
Science, Discovery, and the Universe
Computer Science - Cybersecurity

Going Remote

I joined Intel Corporation as a security research intern in November of 2020, and I am currently working with an internal red team that focuses on Field Programmable Gate Array (FPGA) security. I report directly to my team's leader, Dr. Brian Delgado, yet my internship experience has been fully remote due to the pandemic. My initial job description was to assist with fuzzing low-level firmware code, and I have since had the opportunity to branch out and work on a variety of other tasks. I have learned a great deal about research cybersecurity since joining the team.

Fuzzing from 10,000 ft



Fuzzing for Security

Fuzzing is a form of dynamic analysis which uses random input generation and mutation to test a compiled binary. Fuzzers will continue to test a given target until either the developer terminates their execution, or an input is generated which causes the source code to crash. The goal of our project is to apply LibFuzzer, an open-source, coverage-guided fuzzing utility, to a vast array of firmware code. Any crashes found can be invaluable when searching for potential vulnerabilities.

Establish entry points

- Identify unit test targets, develop corresponding harness

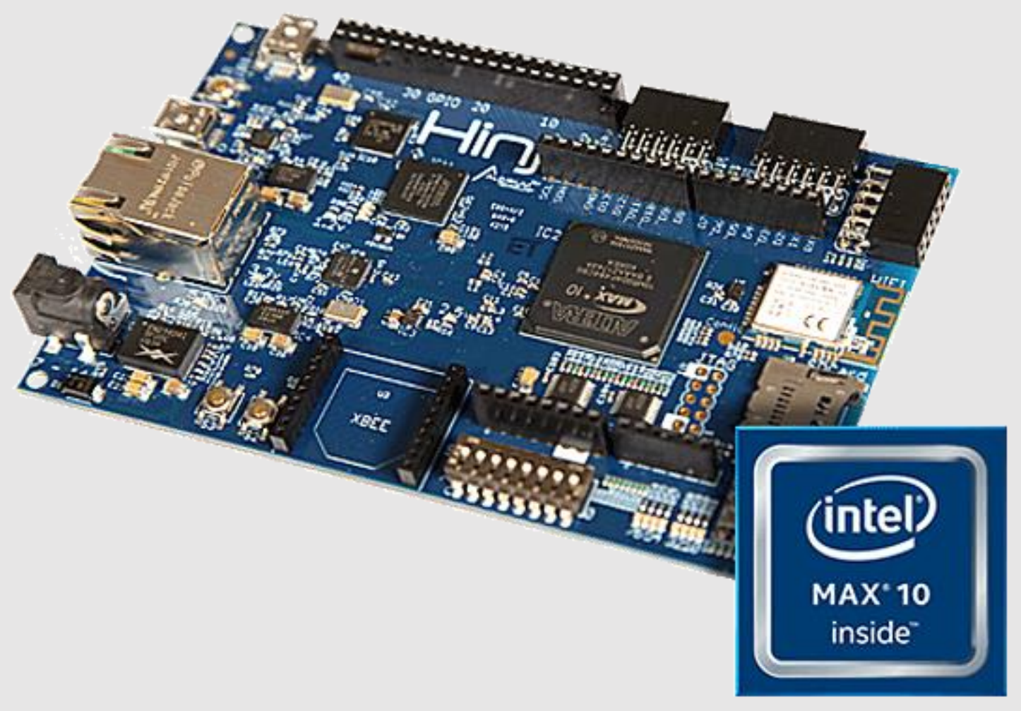
Begin fuzzing

- Generate pseudorandom input, execute program under test
  - Trace CMP instructions, apply sanitizers during runtime
- Check coverage, mutate interesting inputs

Report results

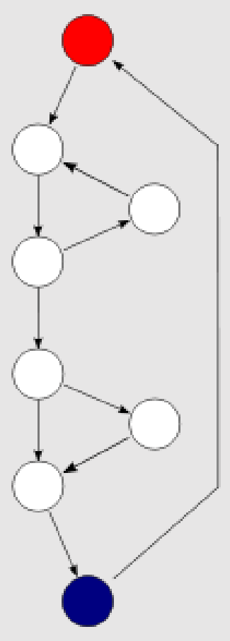
- Patch potential bugs, maintain spanning set of inputs in corpus (enable regression testing, supplement unit framework)

Why FPGAs?

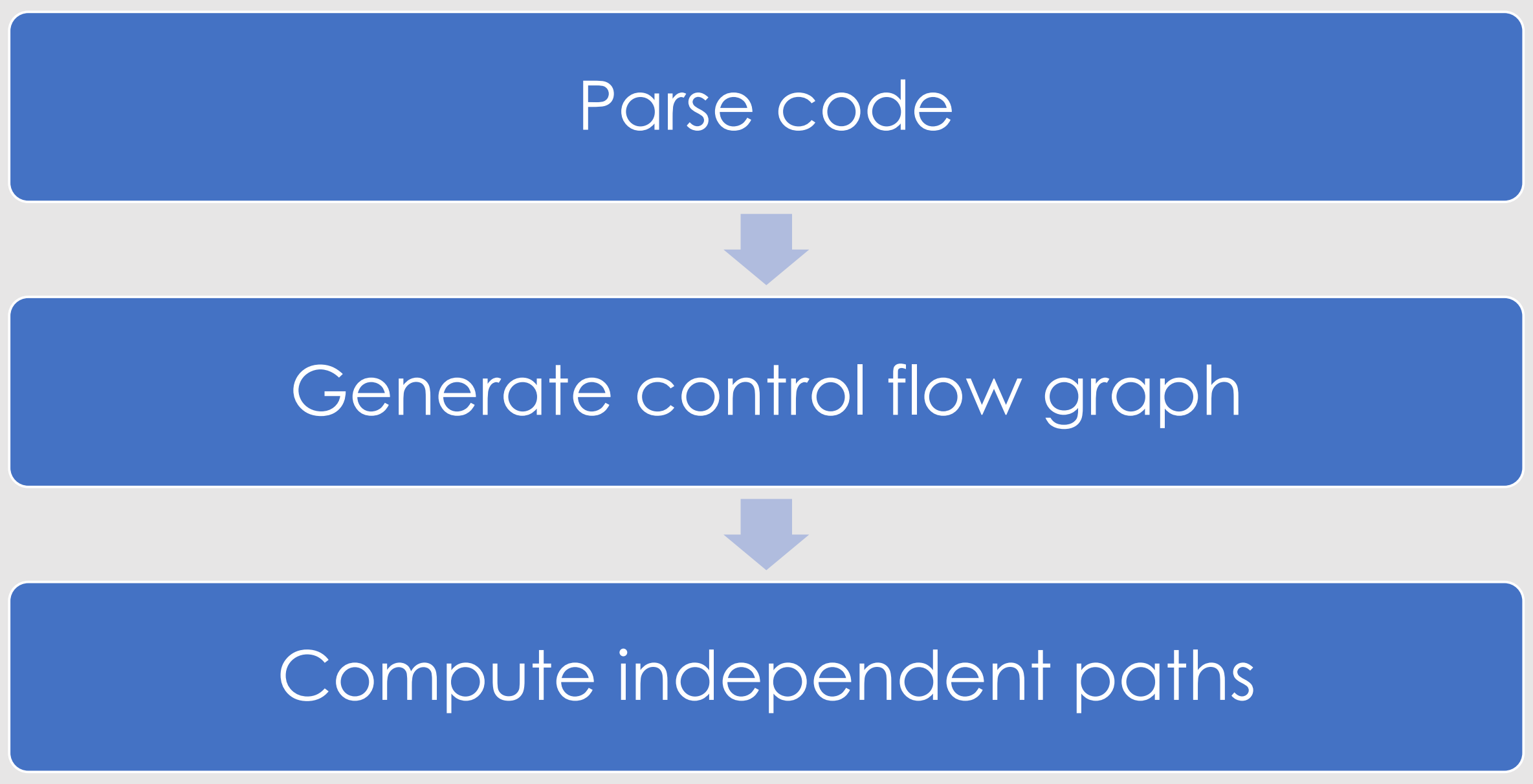


FPGAS can be thought of as generalized computer processors. They possess a vast array of logic blocks which allow for circuits to be reconfigured after production.

Cyclomatic Complexity



Some common metrics used to judge source code are unit test coverage, line count, and comment to code ratio. A similar approach is to compute the number of independent paths through a program (cyclomatic complexity) and use this value to gauge its intricacy. The complexity of any given program can thus be measured as a function of its length, conditionals, nesting, and so on. I developed a script to compute this statistic across any given C codebase. Tools used: pmccabe, GNU complexity



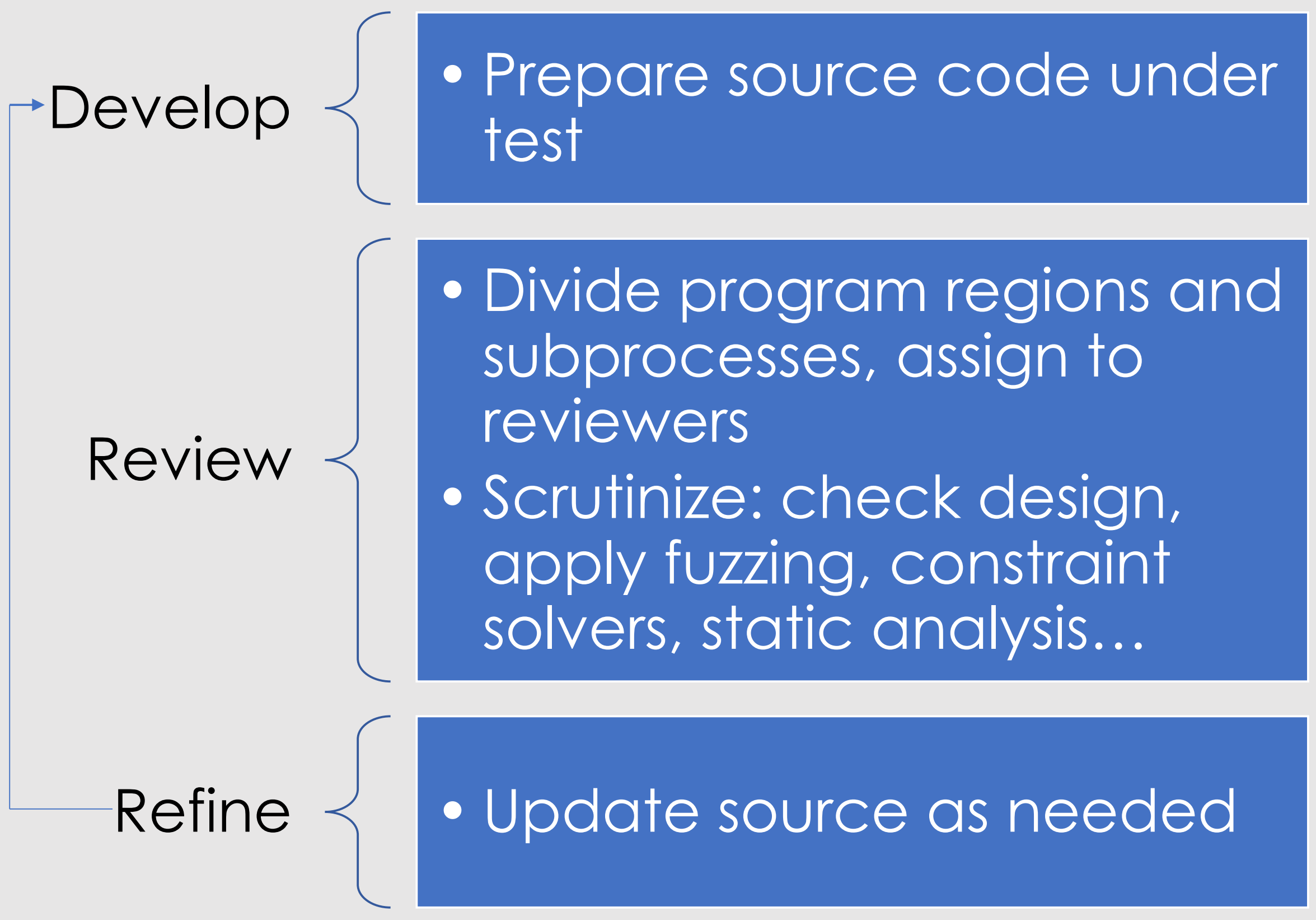
Parse code

Generate control flow graph

Compute independent paths

Code Review

Code review is an iterative refinement process which requires several developers, especially ones who did not work on the code, to scrutinize the program of interest.



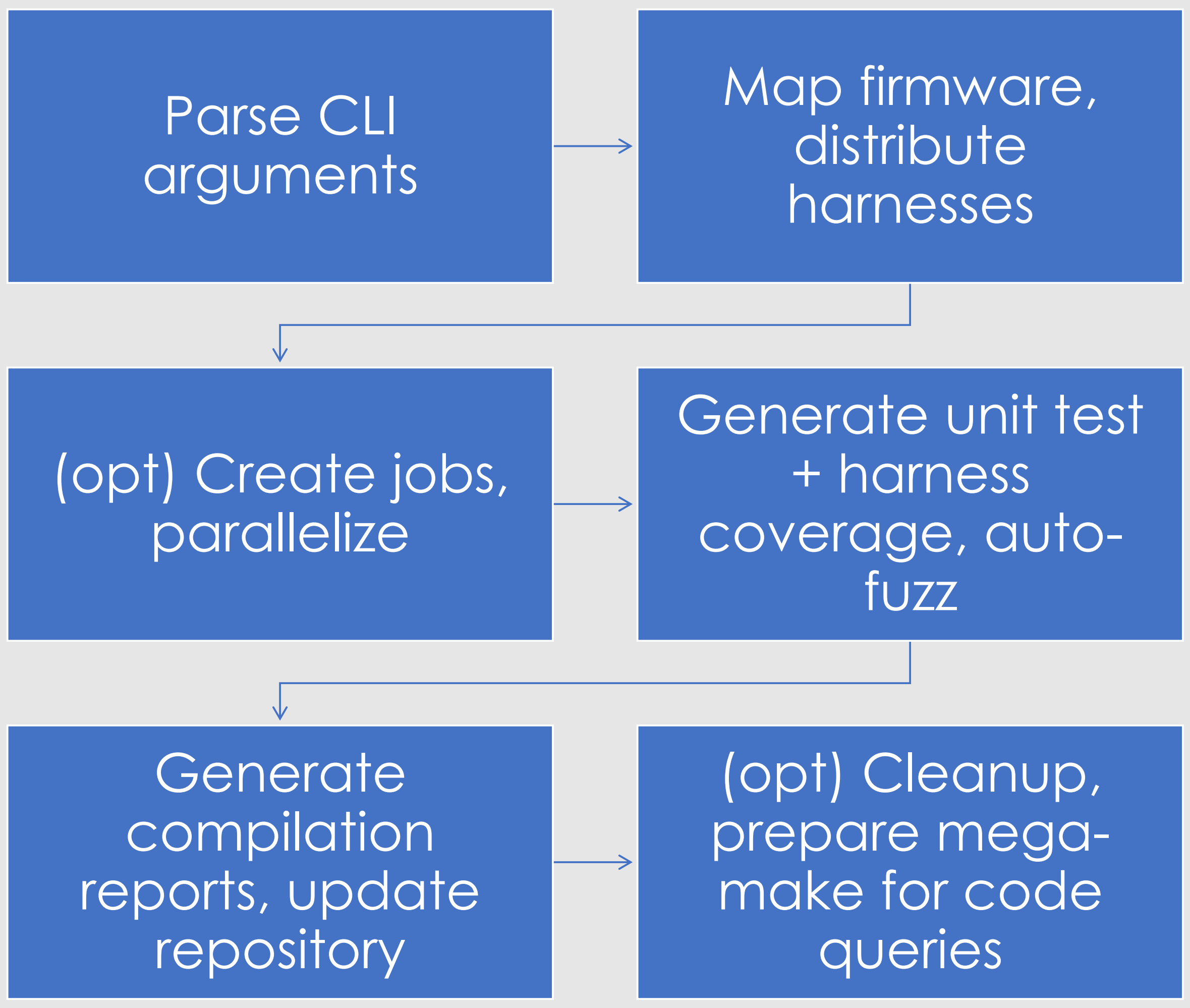
Develop
- Prepare source code under test

Review
- Divide program regions and subprocesses, assign to reviewers
- Scrutinize: check design, apply fuzzing, constraint solvers, static analysis…

Refine
- Update source as needed

Impact

I identified three unique bugs in the firmware codebase while fuzzing. I also worked on preparing our harnesses for docker their first internal rollout. To support this task, I helped establish internal style guidelines, documented related tools (e.g. queries for global variable filtering), and developed several automation scripts which I hope will outlive my time in this role (example below).



Parse CLI arguments

Map firmware, distribute harnesses

(opt) Create jobs, parallelize

Generate unit test + harness coverage, auto-fuzz

Generate compilation reports, update repository

(opt) Cleanup, prepare mega-make for code queries

Future Work

Fuzzing
- Prepare variable tables for harness generation
- Add fuzzing automation to internal CI system
  - Smart target recognition

Symbolic execution
- Test programs with abstract symbols in place of concrete parameters
- Validate logic, identify errors in complex code

Expand code query tools
- Automate style assertions
- Add regression testing
- Introduce taint tracking

Reach-ability analysis
- Define network of entities and connections between them
- Examine possible states

Acknowledgements

I would like to thank my supervisor Dr. Brian Delgado, my manager Geoffrey Strongin, and the rest of my colleagues at Intel Corporation for the wonderful internship experience.