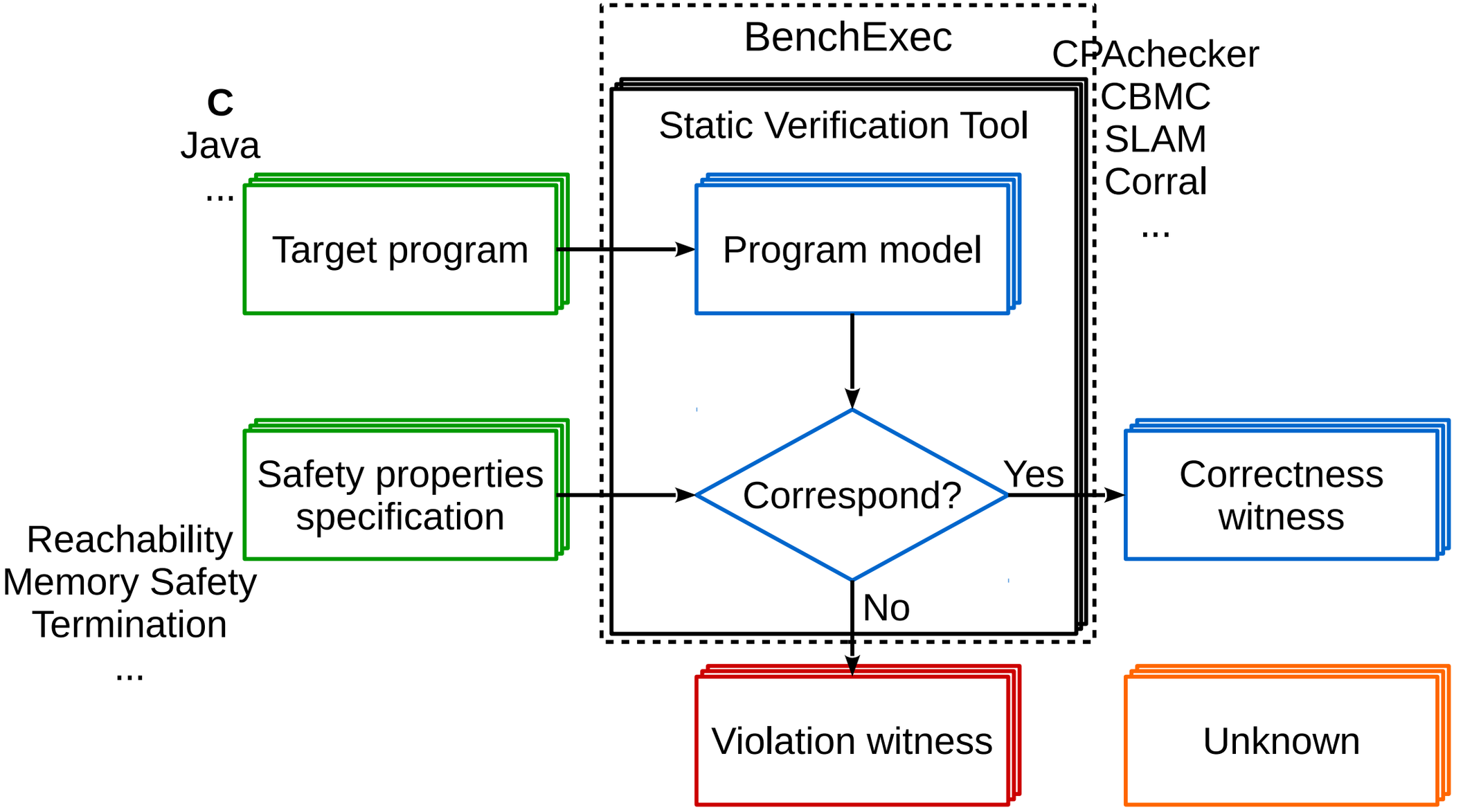
Towards Automated Static Verification of C Programs

Evgeny Novikov
Ivannikov Institute for System Programming of
the Russian Academy of Sciences,
Linux Verification Center

CPA'17, Paderborn, September 4, 2017

Static Verification Tools Workflow



Restrictions and Requirements

- Target programs
  - C source files should be preprocessed
  - all entry points should be properly accessed
  - no undefined functions
- Enormous demands of computational resources
- Safety property specifications
  - tool specific languages with limited expressive power
  - common SV-COMP language with quite low expressive power
- Witnesses are not intended for manual analysis

Motivation

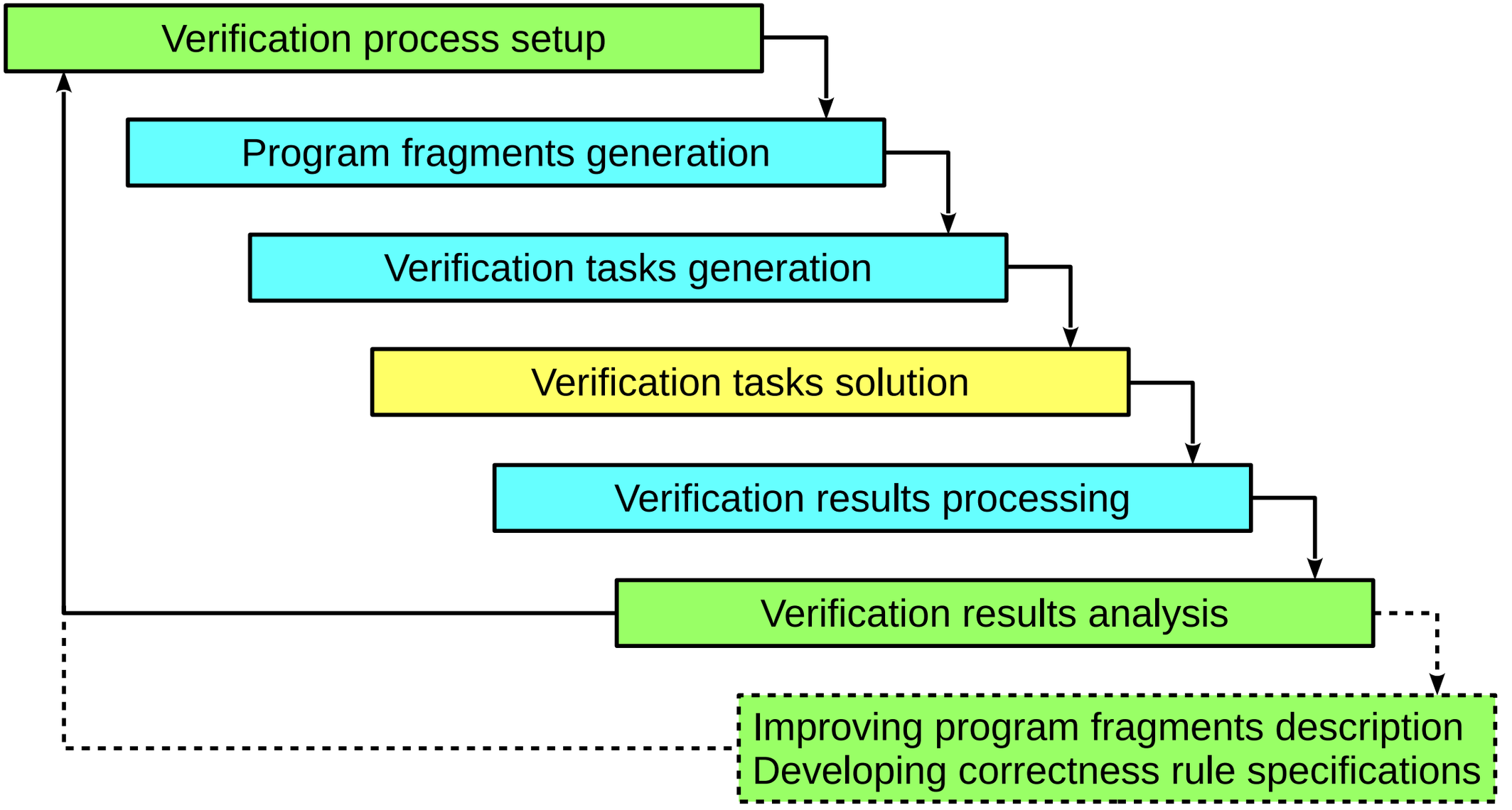
Existing Static Verification Frameworks

- SDV – Windows kernel drivers
- LDV Tools – Linux kernel drivers
- DC2, mbeddr – embedded software
- *Many case studies as proof of concept*

Two Roads

- Simple – users need just to set up verification process properly
- Hard – users need to develop and to implement specific algorithms

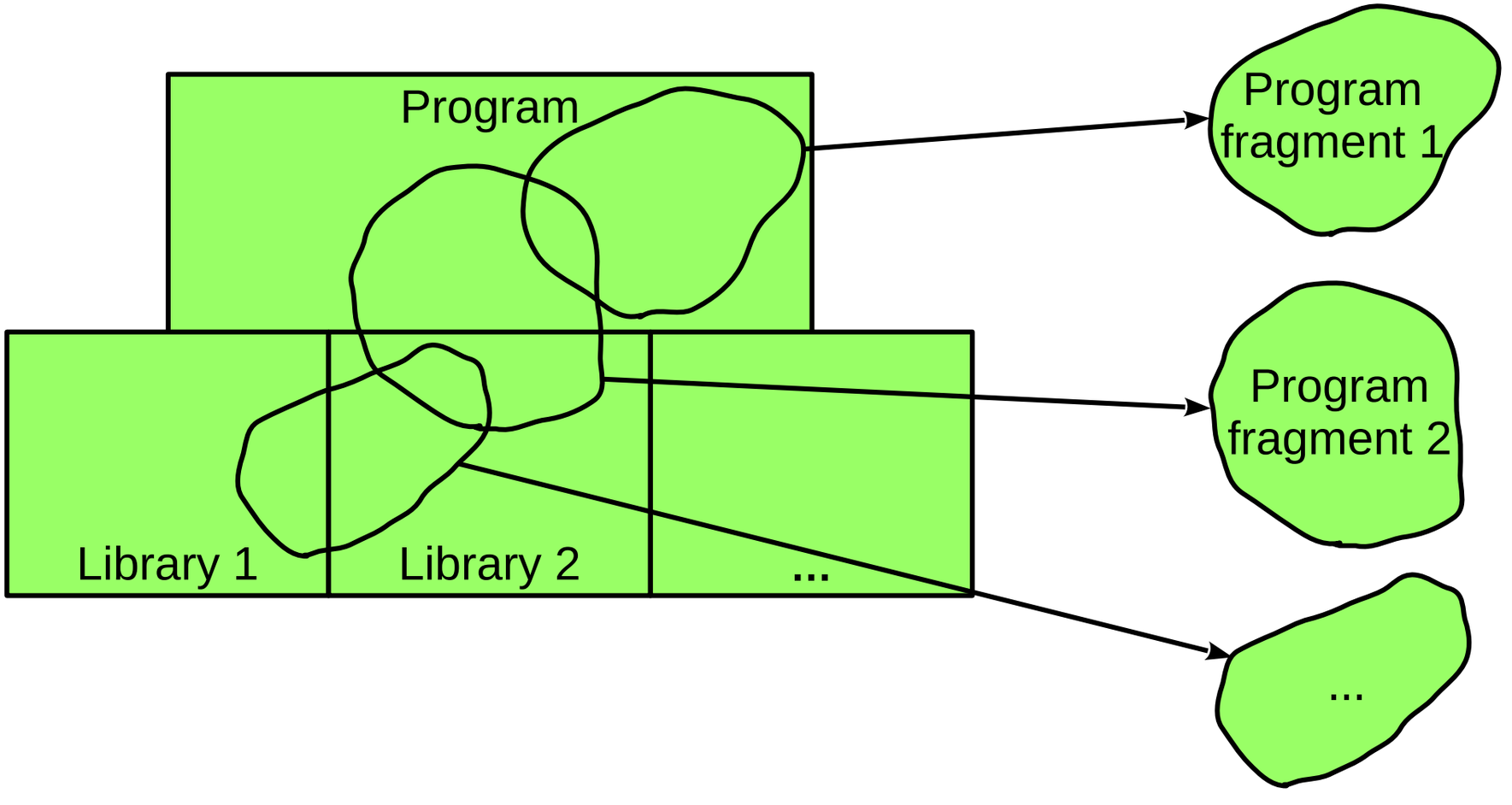
C Programs Static Verification Workflow



Verification Process Setup

- Provide target programs
- Choose correctness rules to be checked
- Specify appropriate configuration options

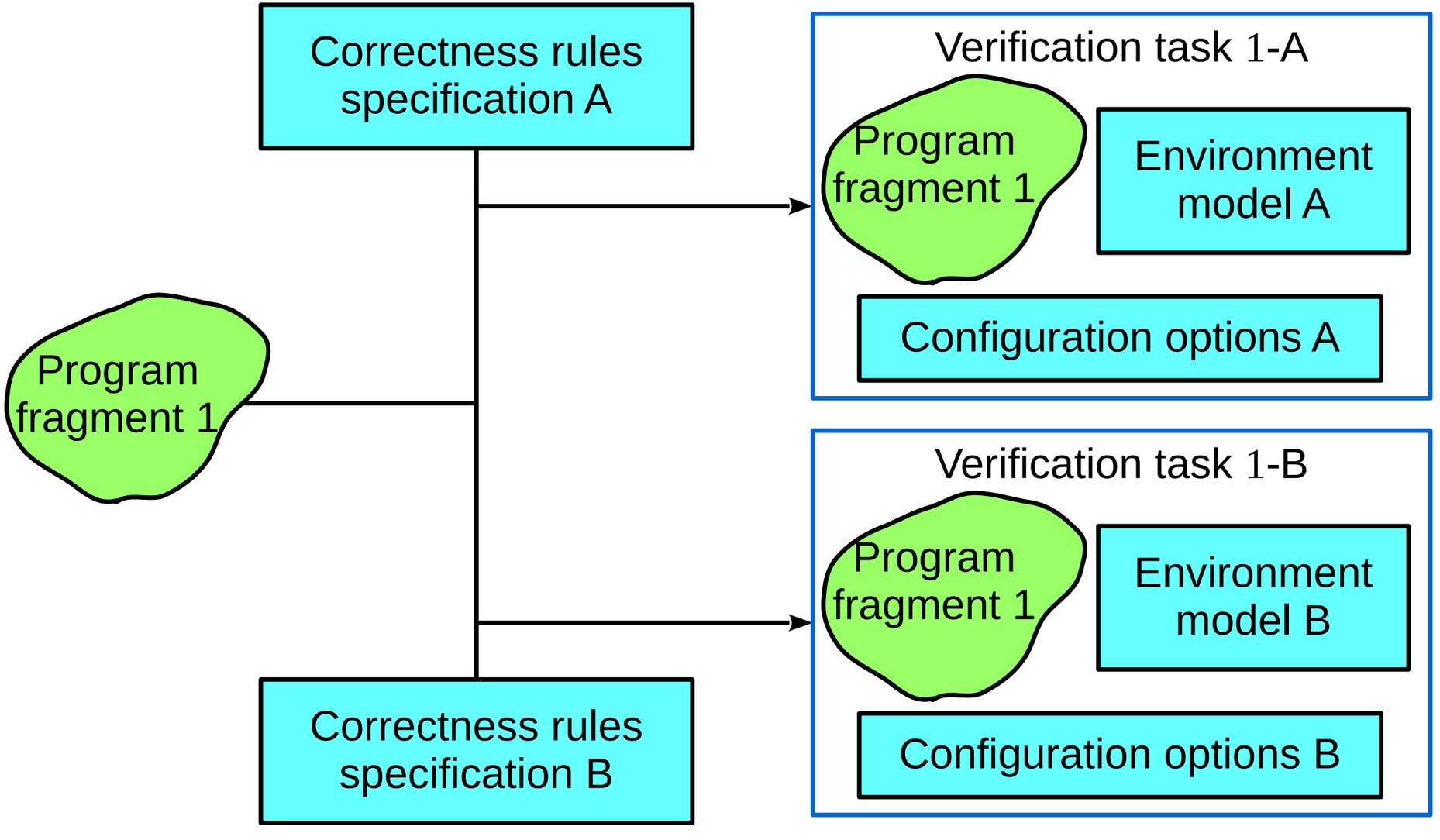
Program Fragments Generation



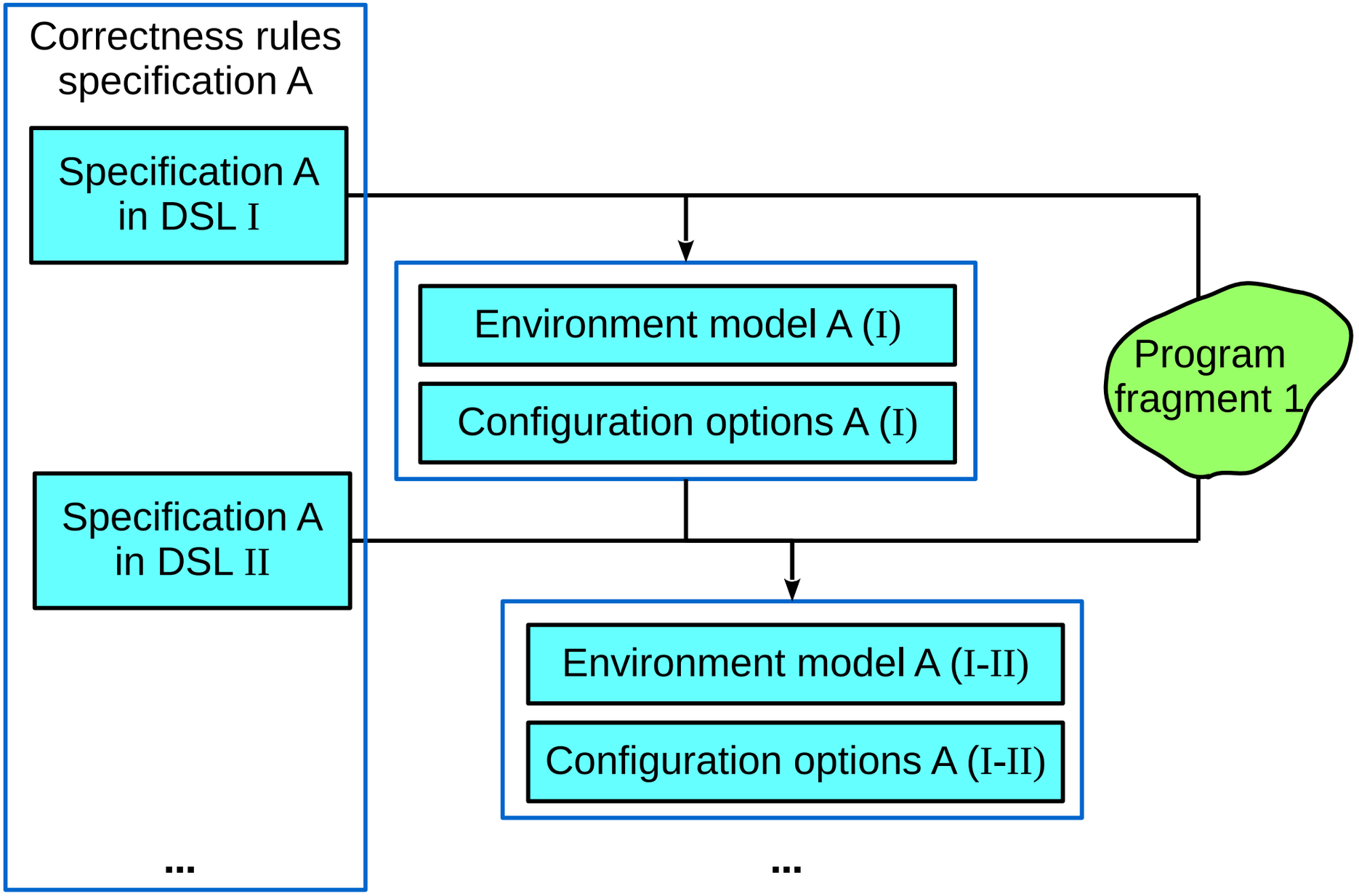
Using C Program Builds to Generate Program Fragments

- Build target C programs and intercept build commands
- Parse build commands to get
  - input and output file paths for all commands
  - preprocessor options and all referred header files for compilation commands
- Generate program fragments on the base of their high-level description and obtained information on build commands

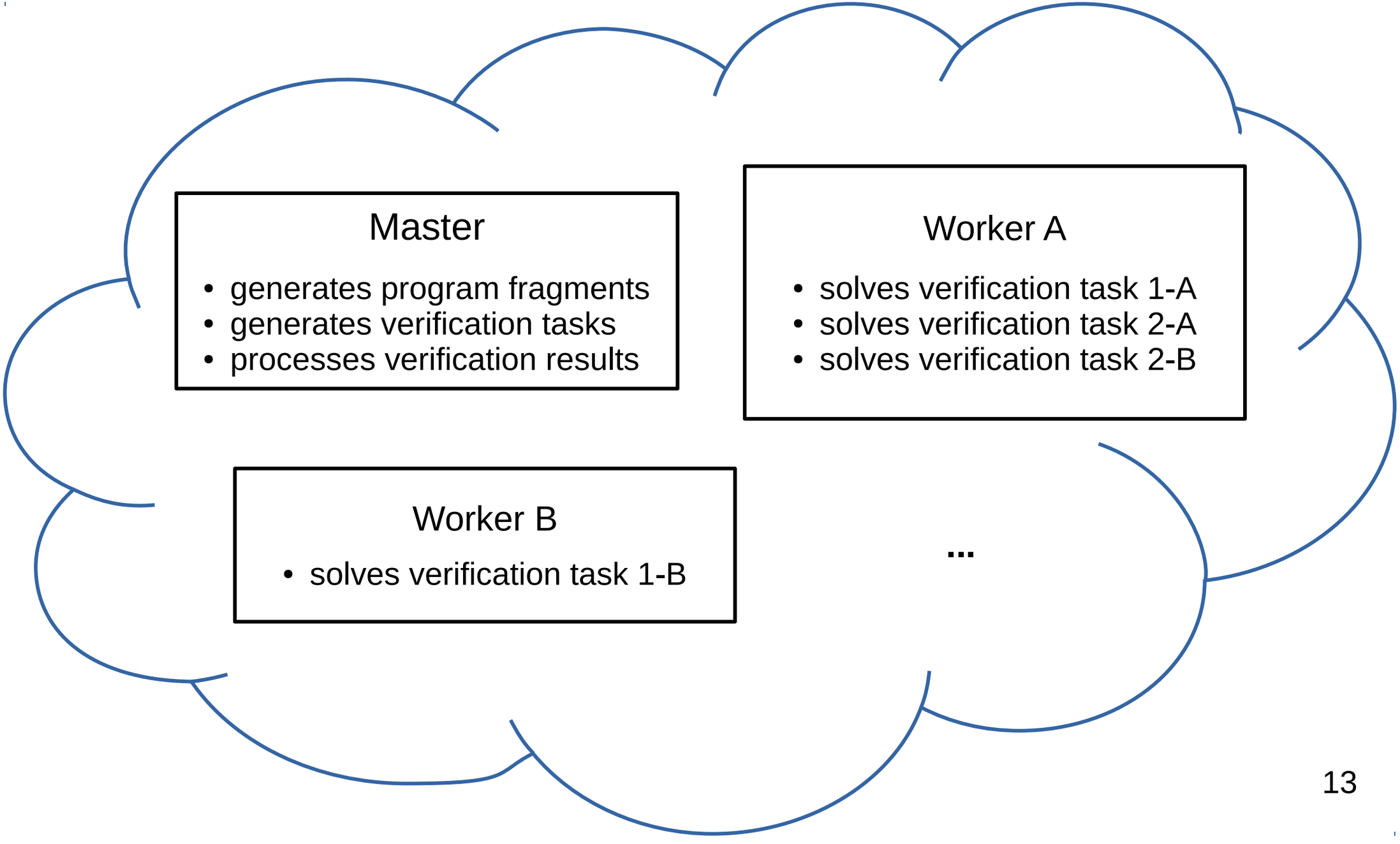
Verification Tasks Generation for All Correctness Rule Specifications



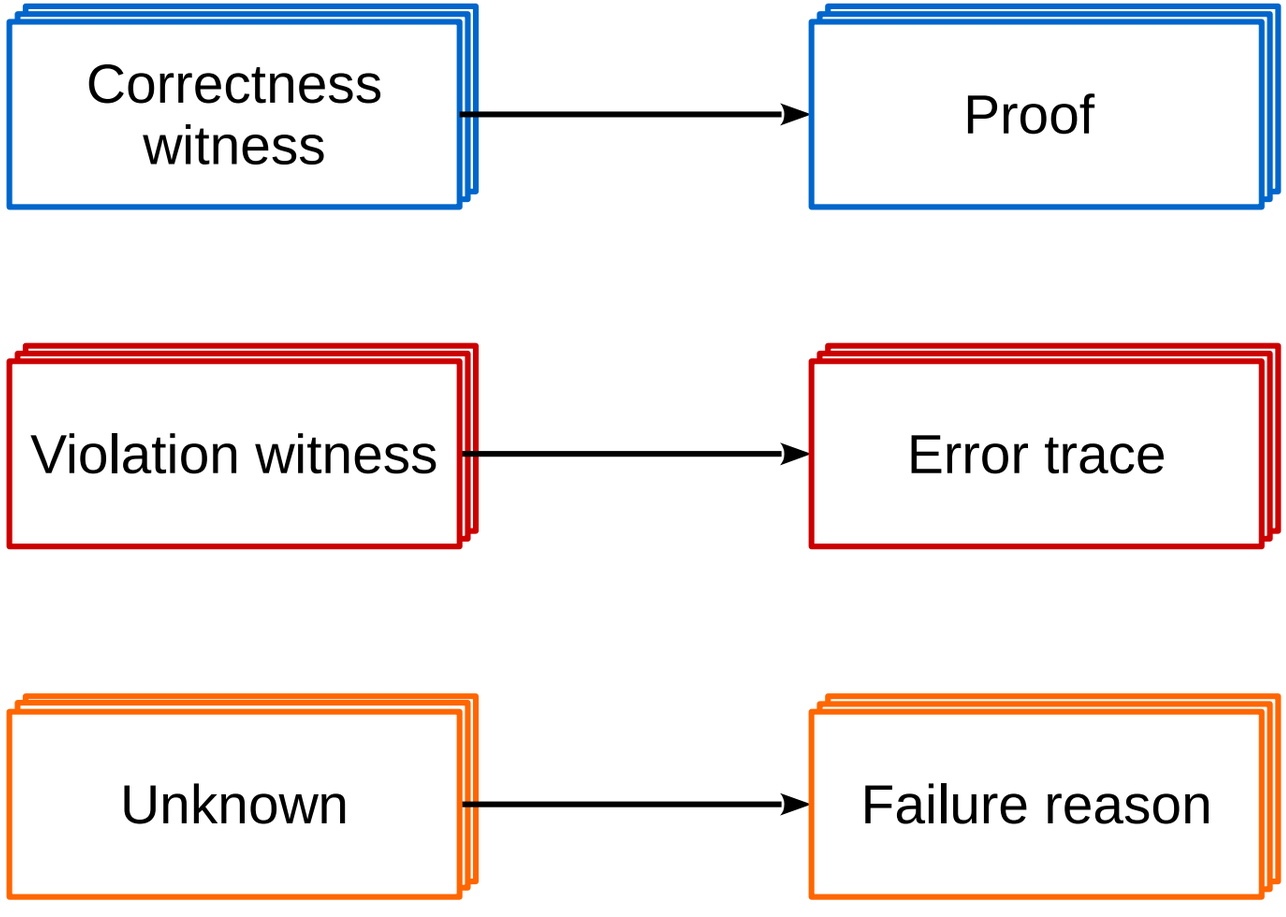
Verification Tasks Generation for One Correctness Rules Specification



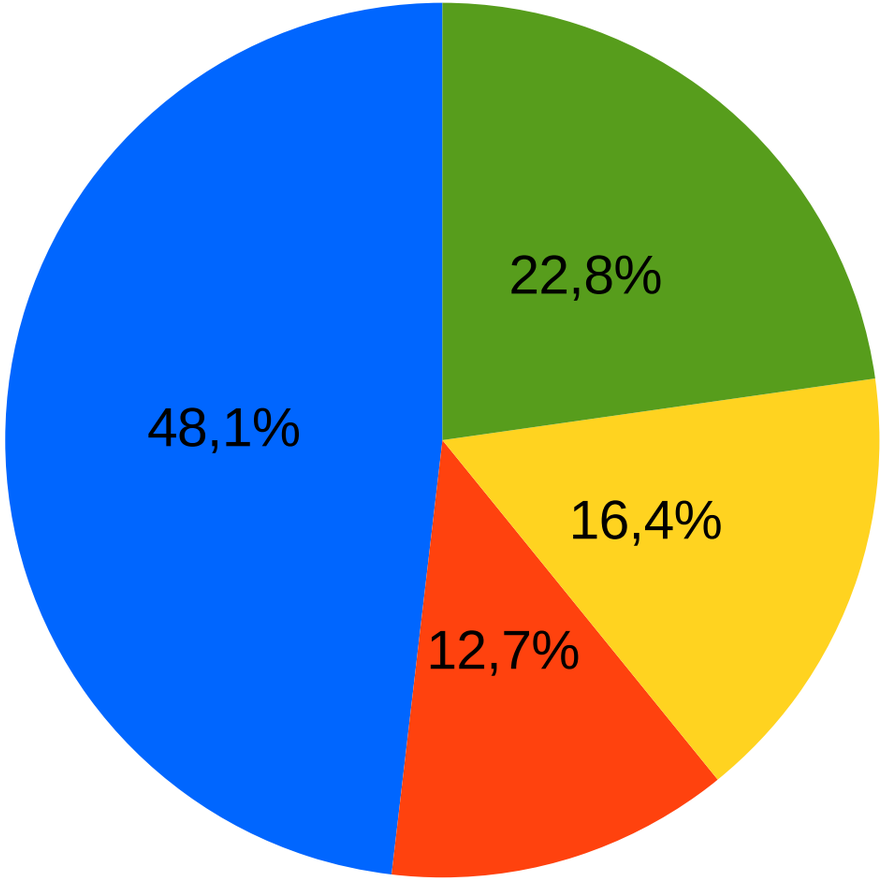
Verification Tasks Solution

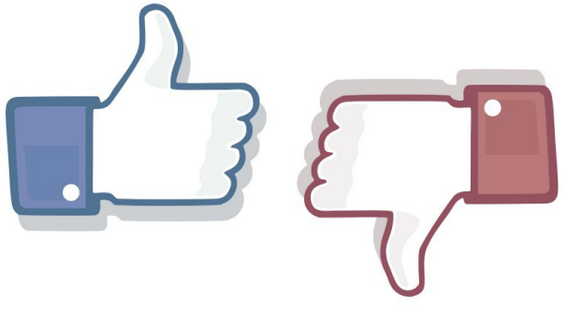


Verification Results Processing



Verification Results Analysis





Improving Program Fragments Description & Developing Correctness Rule Specifications

- Incrementally improve program fragments description
- Incrementally improve existing correctness rule specifications
  - improve specifications in DSLs
  - try different static verification tools or/and their configuration options
- Develop new correctness rule specifications

Klever: Static Verification Framework for GNU C Programs

Static verification workflow step	Current implementation status
Verification process setup	The Linux kernel in form of archives, source trees or Git repositories. Correctness rule specifications base
Program fragments generation	Each fragment is either individual Linux kernel module or group of interconnected modules
Verification tasks generation	Specifications for checking correctness of Linux kernel API usage, memory safety and absence of data races
Verification tasks solution	Running CPAchecker and Ultimate Automizer on standalone servers and in VerifierCloud
Verification results processing	Processing of violation witnesses
Verification results analysis	Convenient means for comprehensive analysis by several experts
Development of correctness rule specifications	Aspect-oriented GNU C, templates, Linux specific DSLs

Static Verification of All Loadable Modules of the Linux Kernel

Type of correctness rules	Warnings	Mean CPU time for verification of one module, min	Total CPU time for verification of all modules, days
Correct usage of the Linux kernel API	**463**	0,3	68,7
Memory safety	**1267**	4,9	14,4
Absence of data races	**89**	1,0	3,2

Bugs and False Alarms

Type of correctness rules	Bugs	False alarms
Correct usage of the Linux kernel API	**110 (26%)**	**309 (74%)**
Memory safety	**28 (8%)**	**337 (92%)**
Absence of data races	**53 (60%)**	**36 (40%)**

Reasons of False Alarms

Type of correctness rules	Inaccurate rule specifications	Inaccurate static verification tool
Correct usage of the Linux kernel API	69%	31%
Memory safety	85%	15%
Absence of data races	58%	42%

Reasons of Missed Bugs

Found bugs	Inaccurate rule specifications	Timeouts
61%	**31%**	**8%**

Future Work

- Support automated static verification for the Linux kernel and for several critical user space applications and libraries
- Improve existing and develop new correctness rule specifications
- Improve verification results representation
- Extend GUI to get more statistics and to simplify routine tasks
- Support static verification within clusters
- Integrate more static verification tools