CONTINUOUS SECURITY TESTING FOR THE AUTOMOTIVE DOMAIN

*Simon Greiner, **Hans Löhr**,* Paul Duplys*

Safety, Security, Privacy
Corporate Research
Robert Bosch GmbH

*Now at SUSE

1st ITG Workshop on IT Security (ITSec) – University of Tübingen – 2020-04-02

BOSCH

Continuous Security Testing for the Automotive Domain
Agenda

- Introduction
- Requirements for Automotive Software Testing and Analysis
- The **CrATE** Framework
- Software Testing and Analysis Methods
  - Static Code Checking
  - Static Analysis Based on Semantic Code Property Graphs
  - Software Fuzzing
- Fuzzing Embedded Software: Preliminary Results
- Conclusion & Outlook

Continuous Security Testing for the Automotive Domain
Introduction and Motivation

- Current general trends
  - Increasing connectivity
  - Increasing complexity
  - More software
  - Increasing (potential) safety impact of security incidents
- Software vulnerabilities
  - Major root cause of real-world security incidents
  - Need to be avoided or detected as early as possible
- **Software security**: becoming more and more important!
  - Building secure and robust systems is essential (not only) for automated driving
  - **Software security testing and analysis** to find bugs early in the development cycle is a crucial building block

Recall of 1.4 Millions jeeps in 2014 (source: Wired)

World's largest 1 Tbps DDoS Attack launched from 152,000 hacked Smart Devices in 2016 (image source: medium.com)

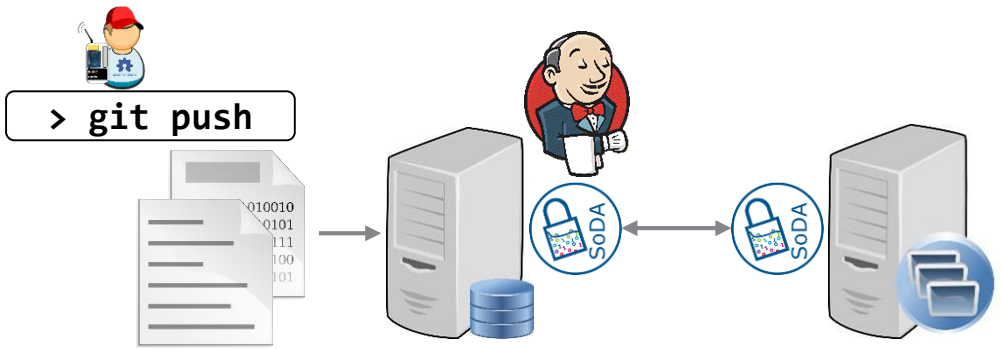
Continuous Security Testing for the Automotive Domain
Requirements

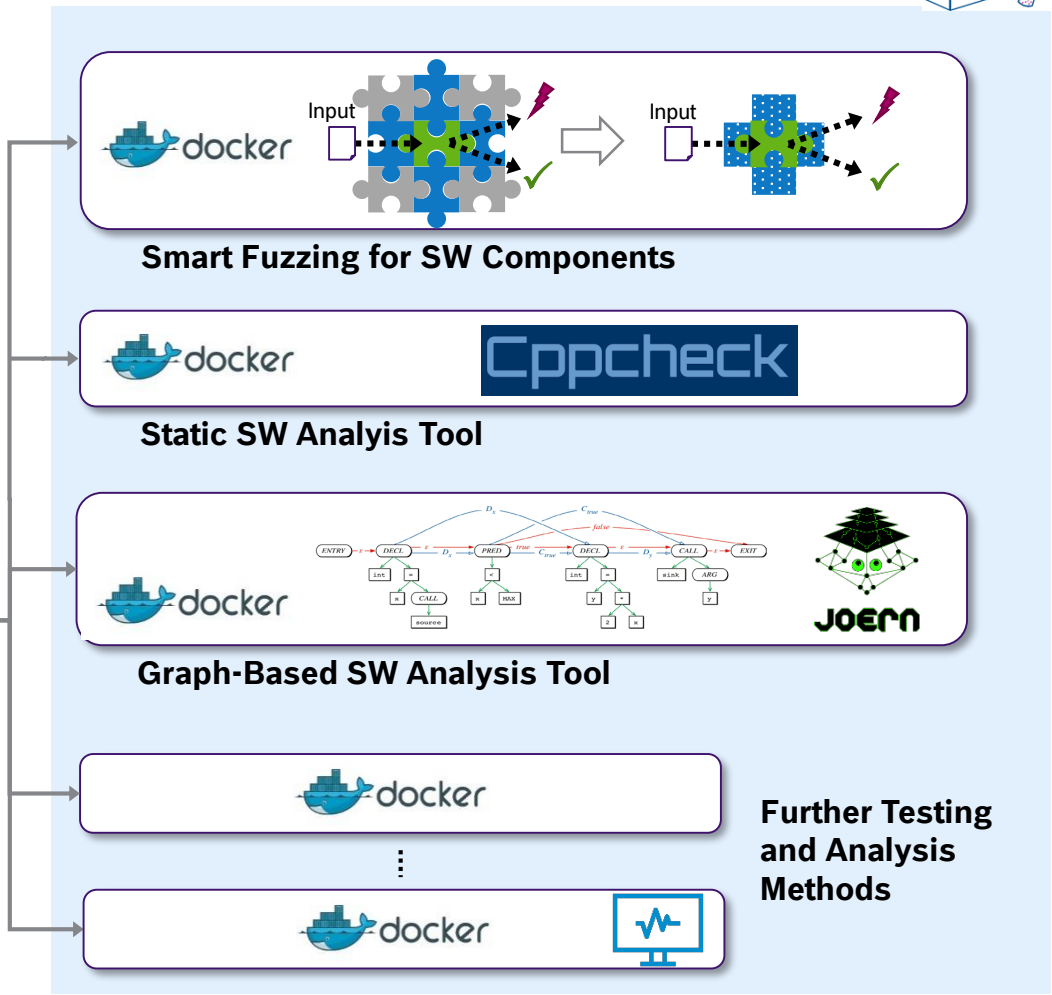
- Suitable for embedded software
  - Low-level languages (C, C++)
  - Heterogeneous build environments
  - Stateful programs
- Safety relevance
  - Cross-domain tooling, different analysis methods
- Changing tool landscape
  - Easy integration of new analysis tools / methods
- Changes and updates expensive
  - Continuous integration
  - High degree of automation
- Ease of use for developers
- Separation of duties

Continuous Security Testing for the Automotive Domain

Continuous Integration & Testing

- **CrATE:** Extensible & Scalable Framework for Automated SW Security Testing & Analysis





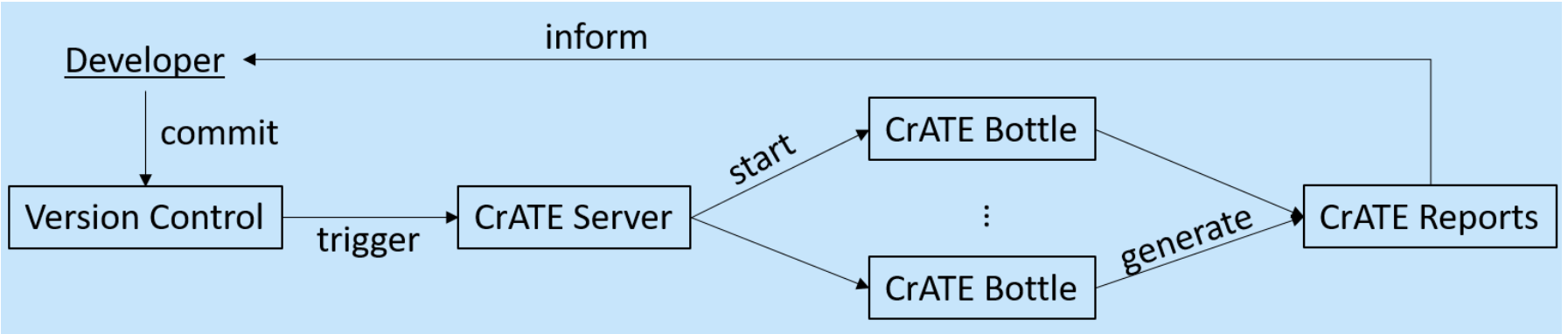
- Execute testing & analysis methods automatically
- Support different kinds of tools
- Generate report on findings and statistics

Continuous Security Testing for the Automotive Domain

The CrATE Platform

CrATE

CrATE: Continuous secu**R**ity **A**nalysis and **TE**sting



- The “CrATE Server” orchestrates test executions (starts containers, etc.)
- Analysis methods in CrATE are called “**Bottles**”
  - Several CrATE Bottles can run in parallel
- Reports on findings are generated from the results

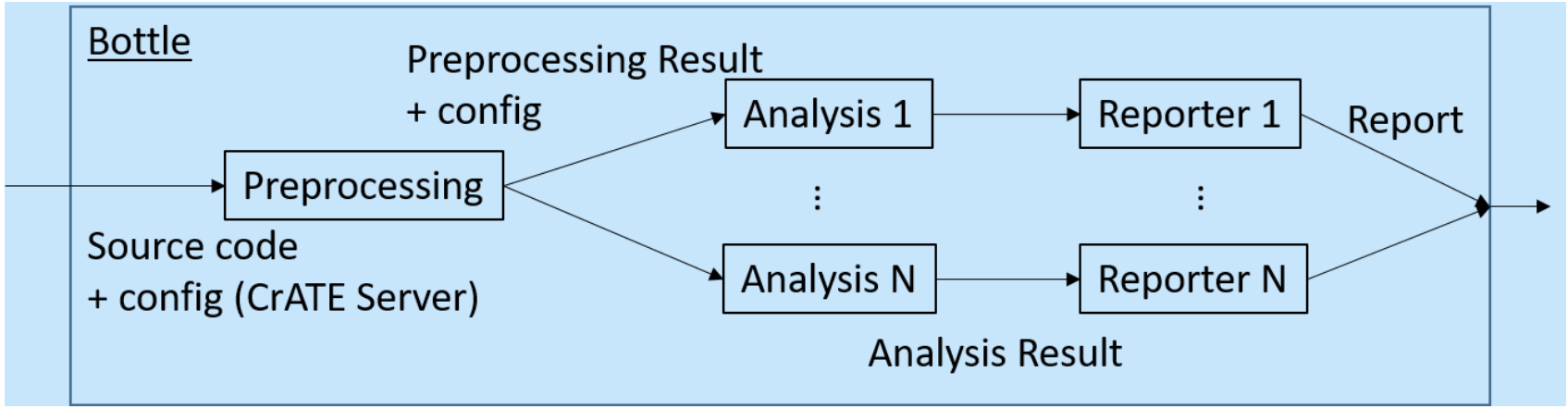
BOSCH

Continuous Security Testing for the Automotive Domain

The CrATE Bottles

CrATE

CrATE **Bottle:** Analysis method integrated into CrATE



- Preprocessing: prepares analysis, e.g., compiles & builds the software
- Several analysers can run in parallel, e.g., to fuzz different software components
- Results are integrated into a report

BOSCH

Analyis Methods for CrATE
Static Code Checking

- **"Simple" static code checking tools**
  - Work on a purely syntactic level
  - Completely automated, simple to apply
  - Useful to check adherence to coding guidelines and rules (e.g., detect forbidded functions)
  - Examples:
    - MISRA rules (subset)
    - SEI CERT Secure Coding Guidelines
  - False positives & usability vary according to specific tool and code base
- Out-of-scope: Verification tools
  - Sophisticated static software analysis (e.g., abstract interpretation, model checking)
  - Many false positives
  - Hard to automate for application to a larger code base

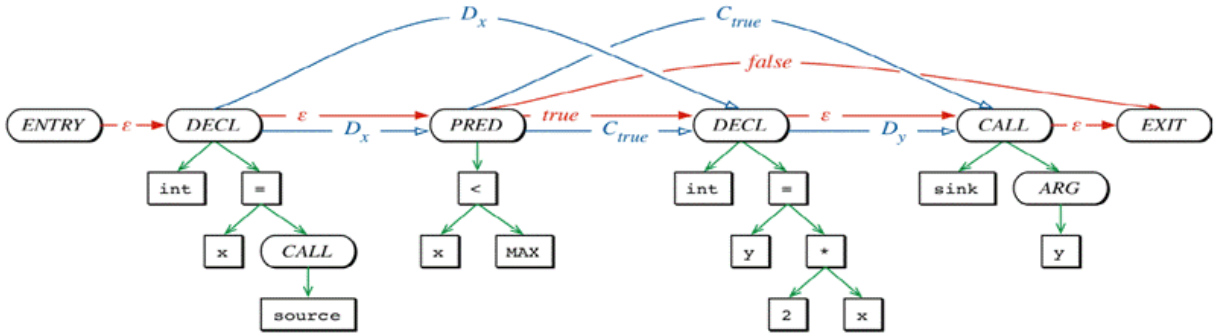
- Proof-of-Concept: **Open-source tools as CrATE Bottles**
  - cppcheck
  - flawfinder
  - CoBrA
- Integration into CrATE is straight-forward
- Commercial tools can be handled analogously

Cppcheck

Flawfinder

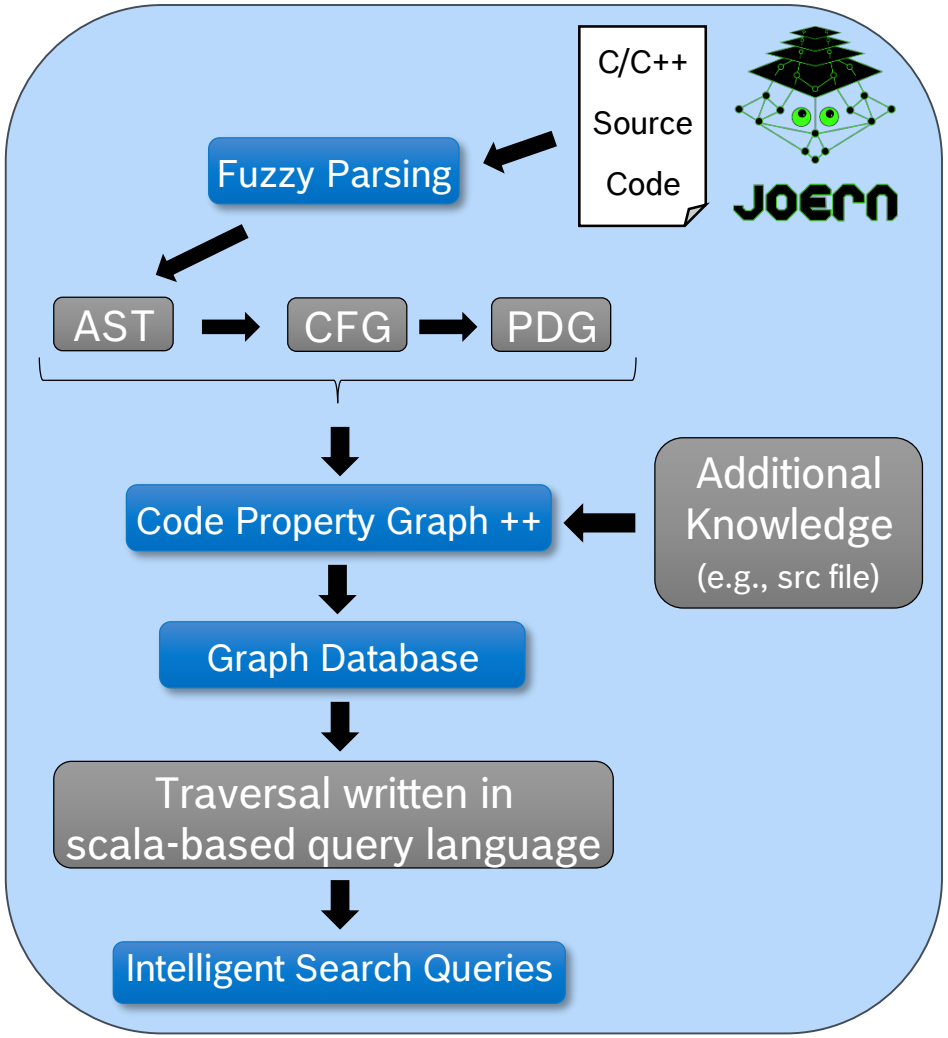
Code Browser and Analyzer

Analyis Methods for CrATE
Code Property Graphs (CPGs)



Source: Fabian Yamaguchi, ShiftLeft

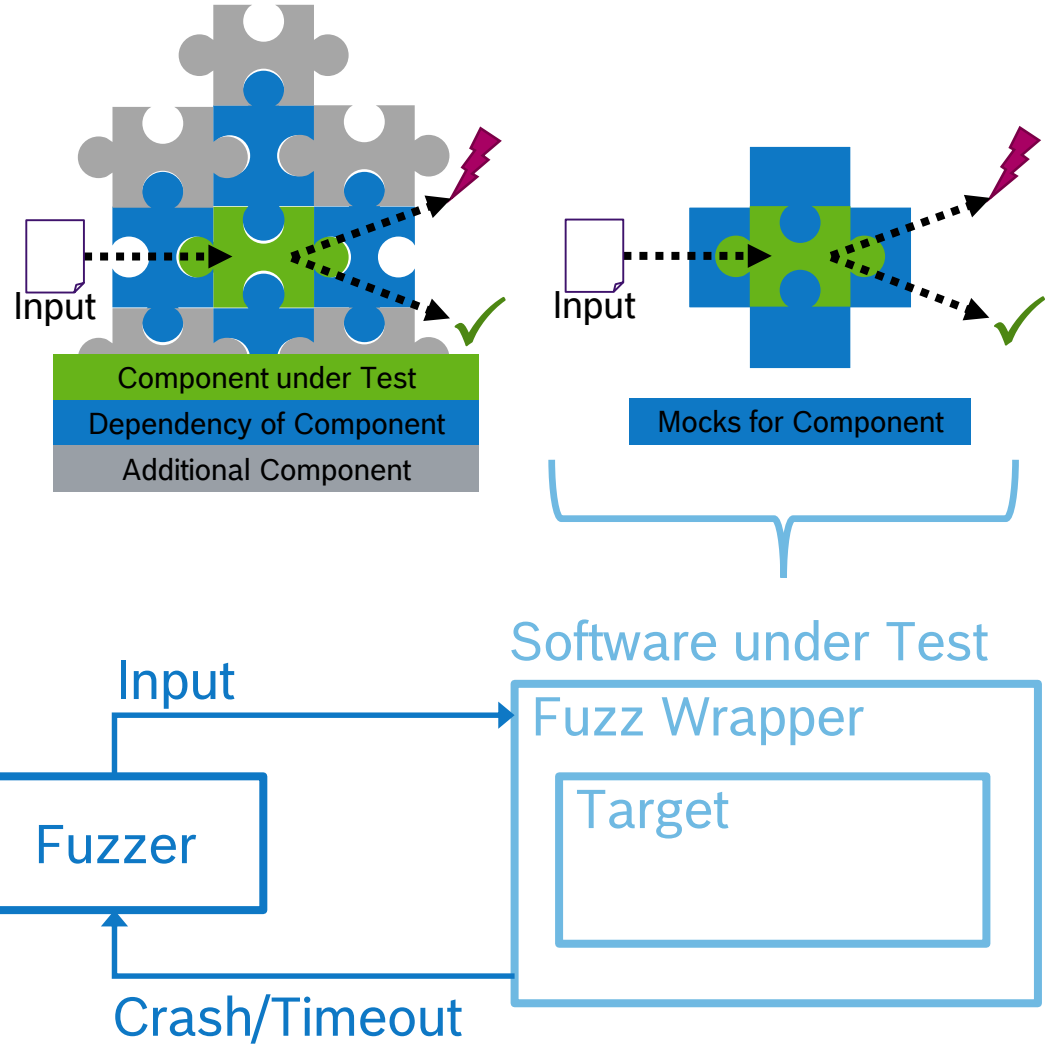
- Generate CPGs as a combination of different graphs
- Formulate search queries in the graph
  - Goes beyond simple syntactic checking
  - Can take control flow and data flow into account
  - Queries can be used interactively (see scientific literature) => Not well suited for CrATE
  - Queries can be run as scripts => Usable in CrATE



AST: Abstract Syntax Tree
CFG: Control Flow Graph
PDG: Program Dependency Graph

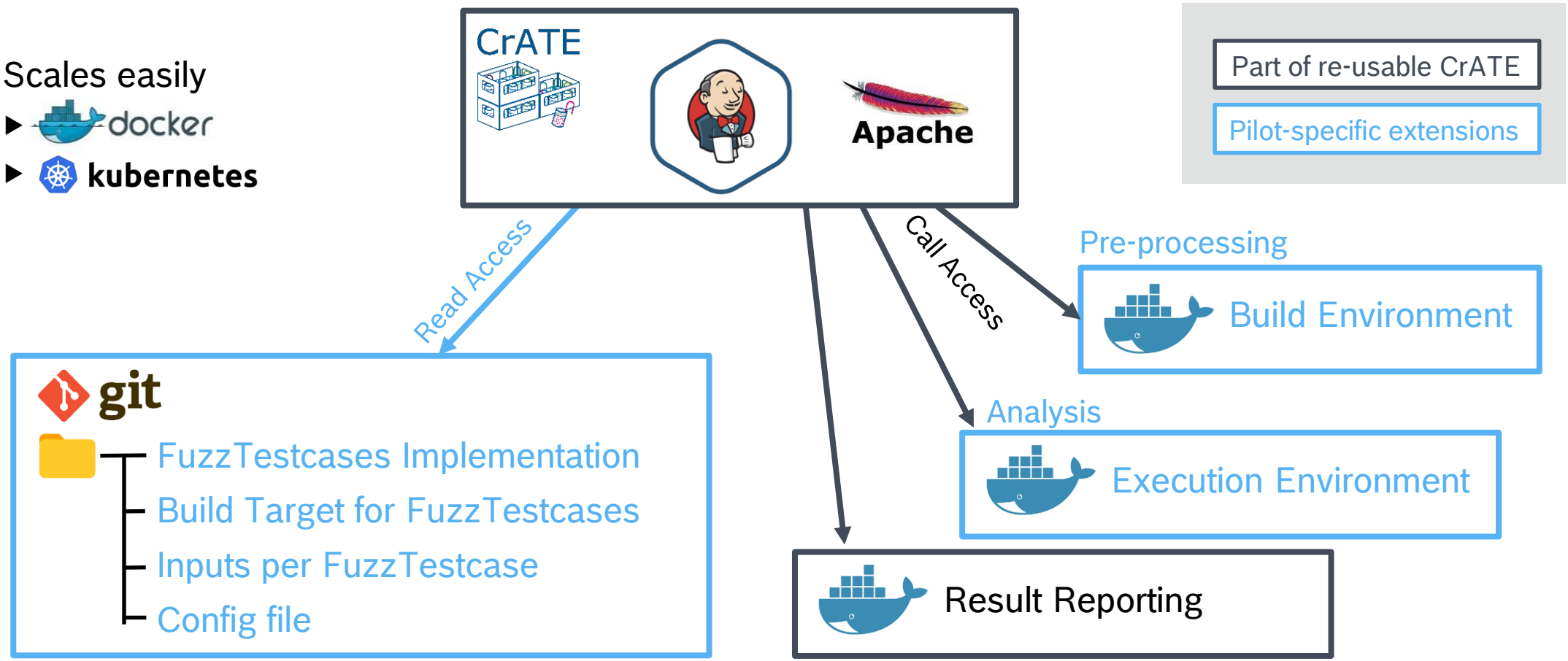
Analyis Methods for CrATE
Fuzzing of Software Components

- Fuzzing: Testing a program with a large number of different inputs (deviating from spec, “random”)
- Fuzzer: Generates inputs, executes tests
- Goal: Find crashes, timeouts, or other observable incorrect behavior
- Fuzz Wrapper:
  - Test harness to fuzz a software component
  - Needs to be implemented for each component (derived from an abstract wrapper class)
- Fuzzers in CrATE: **afl**, **LibFuzzer**
  - Coverage-guided open-source fuzzers
  - Well-known, state-of-the-art tools
- Out of scope:
  - Protocol fuzzing, black-box fuzzing of network interfaces



CrATE Integration Process
Project Integration: Fuzzing as an Example

- Scales easily
  - docker
  - kubernetes



Fuzzing Embedded Software with CrATE
Some Preliminary Results

- Inter-process communication middleware for ECUs
  - Found bugs in early development version
  - Communicated to project team, fixed in later versions
- Open-source logging library
  - Used in automotive projects at Bosch
  - Found bugs, communicated to open-source project
- Open-source parser (json format)
  - Used in automotive projects at Bosch
  - Found bugs, communicated to open-source project
- Network communication library for ECUs
  - Project integrated into CrATE, Fuzzing is just starting
  - Flexibility of build architecture and Fuzz Wrapper proved essential for successful integration

- Preliminary Conclusions
  - Fuzzing can reveal bugs that are hard to find with other methods
  - Scalable platform makes continuous fuzzing easy & painless
  - Integration effort for a new project: ~ 1-2 days (roughly)

Continuous Security Testing for the Automotive Domain

Conclusion & Outlook

CrATE

- Security and robustness testing for automotive software
  - Early in the development cycle
  - In a continuous integration pipeline
- CrATE: platform for continuous security testing
  - Flexible, scalable
  - Supports various analysis methods
  - SW vulnerabilities can be detected early, before code is released
- Acceptance in development teams requires
  - High degree of automation
  - Low entry barrier for project teams

- **Outlook**
  - More pilot projects to get more (representative) code
  - More analysis methods
  - Evaluation and benchmarking
  - Investigation of useful combinations of analysis methods
    - E.g., use results from code property graph-based queries to guide a fuzzer?

BOSCH

THANK YOU!

… QUESTIONS?