

ORACLE®

# Efficient Analysis using Soufflé

**An Experience Report.**

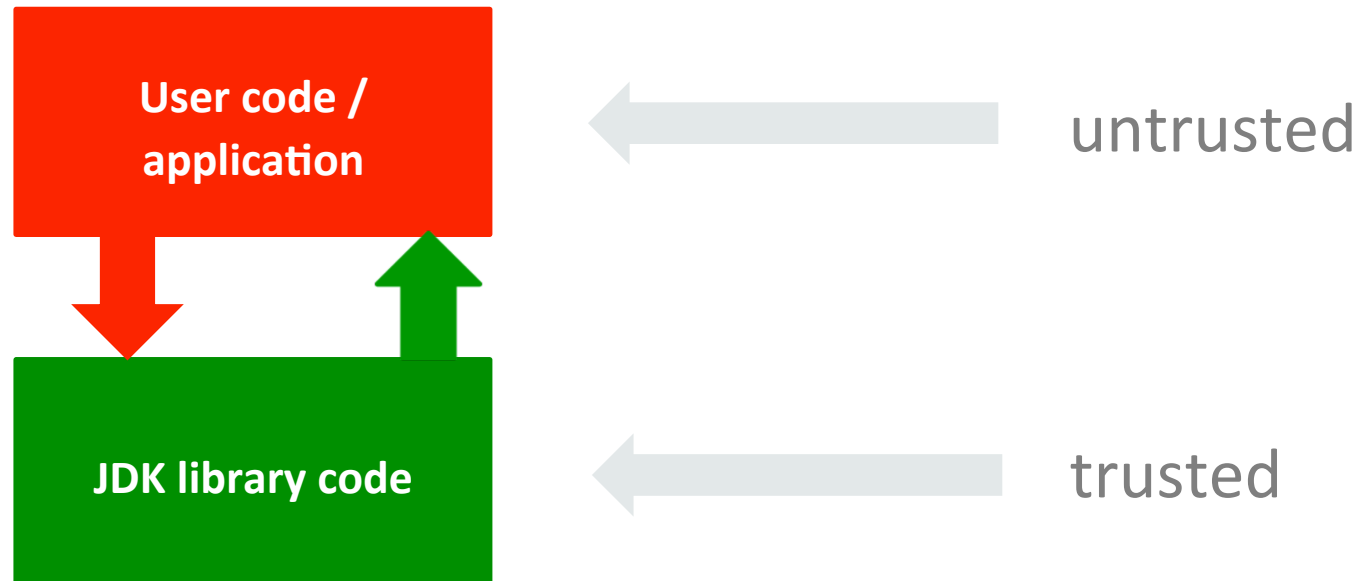
Bernhard Scholz, Pavle Subotic, Herbert Jordan, Paddy Krishnan, Ragavendra Kagalavadi  
Ramesh, Cristina Cifuentes  
Oracle Labs Australia  
July 2016

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Aim: To analyse the JDK for vulnerabilities  
using analyses encoded in Datalog

# Open World Analysis

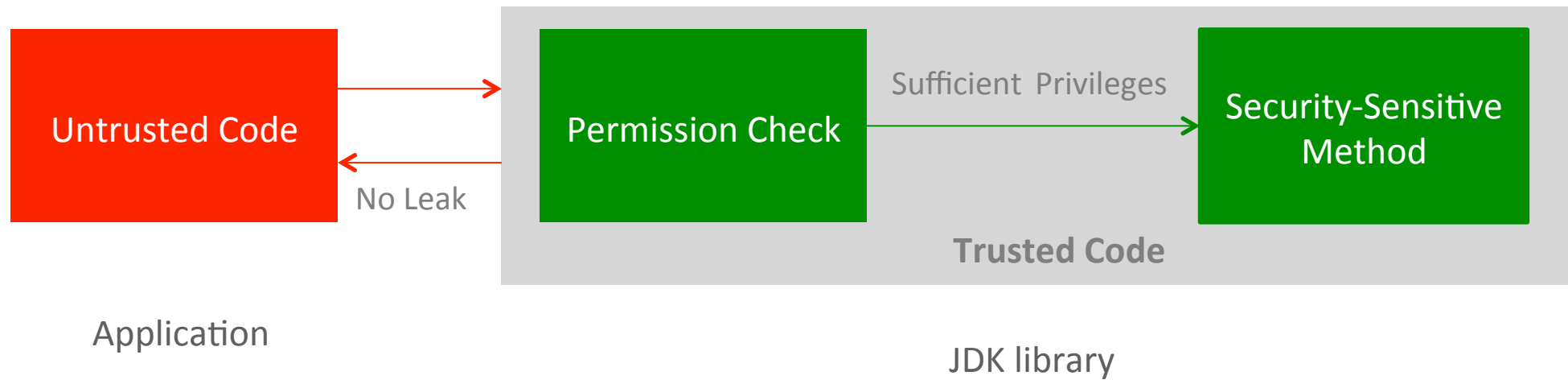
[SOAP'15 – Combining Type-Analysis with Points-To Analysis for Analyzing Java Library Source-Code]



Analysis of libraries, rather than complete applications

# Unguarded Caller-Sensitive Methods

[SOAP'15 – Understanding Caller-Sensitive Method Vulnerabilities]



# Existing Datalog Engines

DOOP context-**insensitive** points-to analysis over OpenJDK 7  
w/o open world w/call graph construction on the fly

| Tool      | Time       | Memory   |
|-----------|------------|----------|
| BDDDBDDDB | 30 minutes | 5.7GB    |
| mZ (Z3)   | <b>x</b>   | <b>x</b> |
| LogicBlox | 20 minutes | 100 GB   |

On Intel Xeon E5-2660 (2.2 GHz) machine with 256 GB RAM  
Time out set to 24 hours

# Problem Size – OpenJDK 7 b147

---

| <b>Program Element</b>          | <b>Size</b>        |
|---------------------------------|--------------------|
| Variables                       | 1.5 Million        |
| Allocation Sites (heap objects) | 361 Thousand       |
| Methods                         | 160 Thousand       |
| Invocation Sites                | 590 Thousand       |
| Types                           | 17 Thousand        |
| <b>Size of CI Points-To Set</b> | <b>866 Million</b> |

---



## With Soufflé

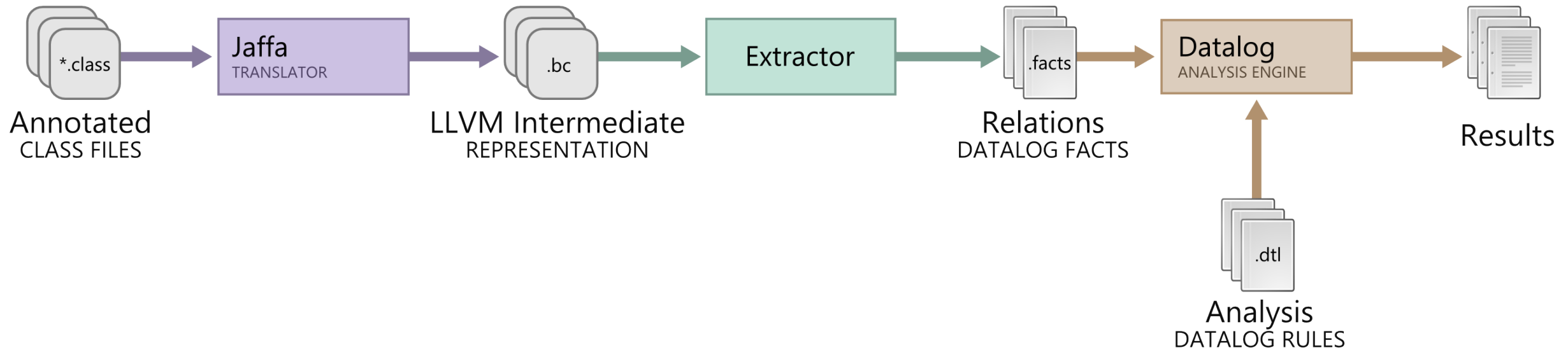
DOOP context-**insensitive** points-to analysis over OpenJDK 7  
w/o open world w/call graph construction on the fly

| Tool           | Time       | Memory   |
|----------------|------------|----------|
| BDDDBDDDB      | 30 minutes | 5.7GB    |
| mZ (Z3)        | <b>x</b>   | <b>x</b> |
| LogicBlox      | 20 minutes | 100 GB   |
| <b>Soufflé</b> | 40 seconds | 7.5 GB   |

On Intel Xeon E5-2660 (2.2 GHz) machine with 256 GB RAM  
Time out set to 24 hours

# The Soufflé Datalog Compiler

# The Soufflé Framework



Extensional DB: input facts  
Intensional DB: Datalog rules

# Problem Size – OpenJDK 7 b147

| <b>Program Element</b>          | <b>Size</b>         |
|---------------------------------|---------------------|
| Variables                       | 1.5 Million         |
| Allocation Sites                | 361 Thousand        |
| Methods                         | 160 Thousand        |
| Invocation Sites                | 590 Thousand        |
| Types                           | 17 Thousand         |
| <b>Size of CI Points-To Set</b> | <b>866 Million</b>  |
| <b>EDB (text, in-memory)</b>    | <b>3.6GB, 872MB</b> |
| <b>EDB # facts</b>              | <b>16,810,032</b>   |

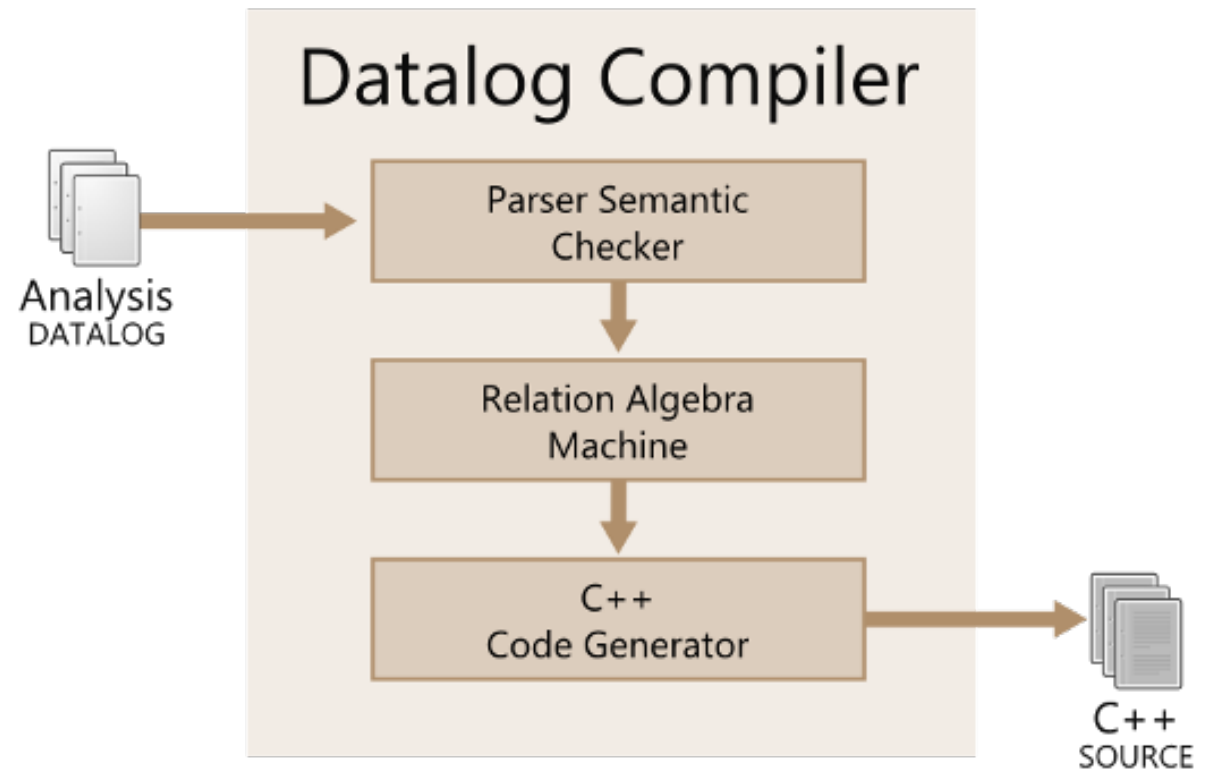
# Soufflé Key Contributions

1. Staged compilation
2. Auto-index selection
3. Brie data structure

# 1- Staged Compilation

[CAV'16 – Soufflé: On Synthesis of Datalog for Program Analyzers]

- Efficient semi-naïve evaluation
  - Elimination of copying overheads: single merge operation
- Relational Algebra Machine
  - Performs relational algebra operations & fixed-point calculations; fixed tables
- Translate abstract machine program to C++ code
  - Generate specialised OpenMP/C++
  - Execution in memory



## 2- Auto Index Selection

- Index to speed up searches in tables for equi-joins
  - Example:  $\{(x,y,z) \text{ in } A \mid A.x = \text{“p”} \wedge A.y = \text{“q”}\}$
  - Search denoted by columns  $\{x,y\}$
- Index: total order via lexicographical order on column
  - Example:  $x < y < z$
- An index may cover more than one search
- Prefix sets of lexicographical order constitute valid searches
  - Example  $x < y < z \Rightarrow \{x\}, \{x,y\}, \{x,y,z\}$
- Maximum Matching solves optimal index selection problem in P
  - Application of Dilworth’s theorem

## 2- Auto Index Selection – Experiment, OpenJDK 7

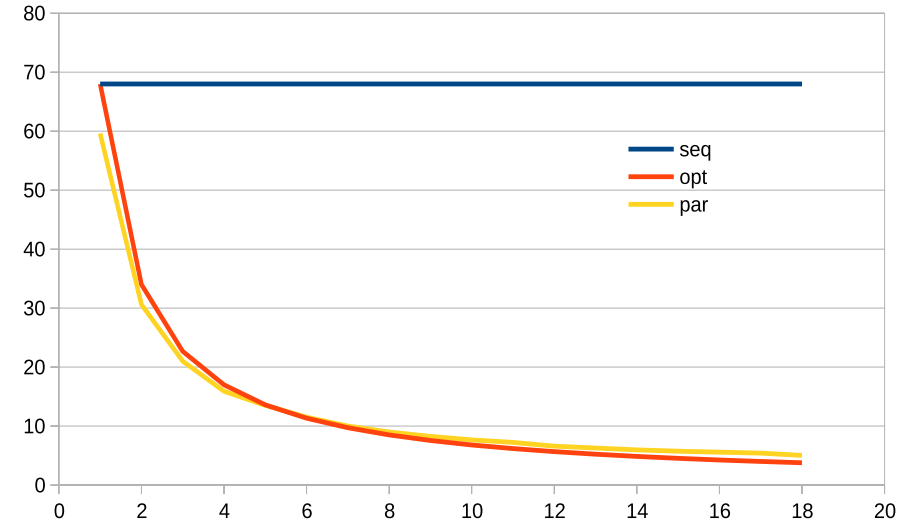
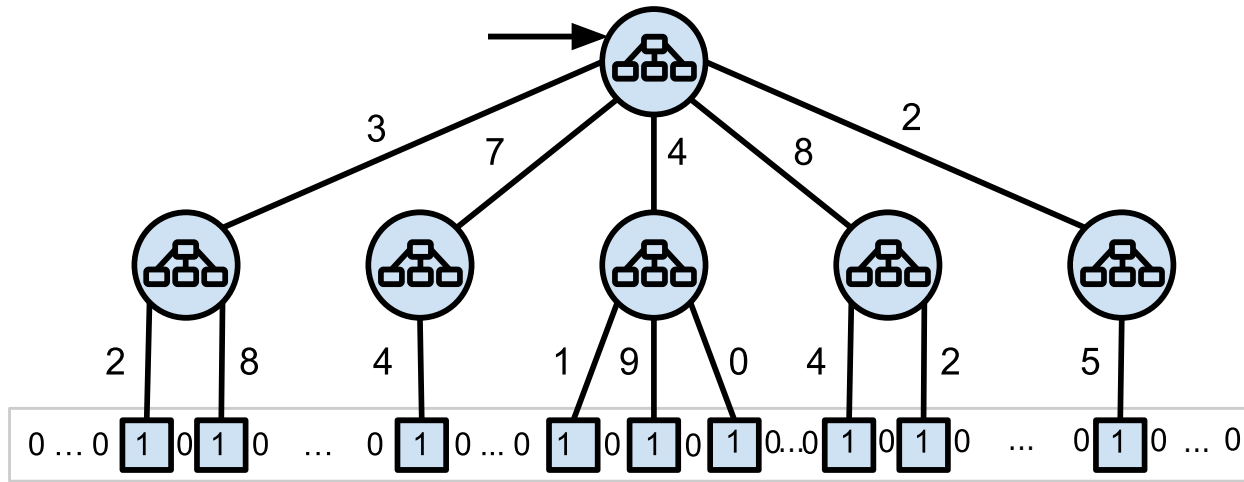
Exhaustive index selection vs. optimal auto index selection with reuse

|                   | <b>Runtime (min)</b> | <b>Memory (GB)</b> |
|-------------------|----------------------|--------------------|
| No auto-index, CI | 16:30                | 81.8               |
| Auto-index, CI    | 0:42                 | 10.3               |
| No auto-index, CS | *                    | *                  |
| Auto-index, CS    | 5h, 30 min           | 18.9               |

\* Timed out after running for 4 days using 55 GB



# 3- Brie Data Structure for Binary Relations in Soufflé



- Hybrid data structure as a fixed-depth Trie with sparse bitmaps
  - Nearly lock-free
  - Use of geometric encoding
  - Becomes scalable for large amount of data (but only binary relations)

# Exploratory Work before Soufflé

[ASWEC'15 – A Datalog Source-to-Source Translator for Static Program Analysis: An Experience Report]

- Datalog to SQL translator
  - using a relational database as back-end for evaluating Datalog rules
- Various SQL optimisations
- Insufficient for large data
  - Insensitive points-to for OpenJDK analysis too slow (~8h)

# Soufflé – Lessons Learned

- Compilation vs synthesis
  - ✓ Use of program synthesis
  - ✗ Use of relational database to evaluate Datalog rules
- Insights into semi-naïve algorithm
  - ✓ Distinct read and write phases
  - ✗ Copying tables to keep track of **previous**, **current**, **delta** and **new** relations
- Use of cache-sensitive, in-memory data structures is vital
  - ✓ B-Trees, geometric encoding of sets and binary relations
  - ✗ Bloom filters, hash-tables: demands sparseness which increases memory usage
- Parallelisation
  - ✓ Optimistic locking of B-trees during insert
  - ✗ Locking potentially maximal subtree

# Performance Improvements Over Time

DOOP context-insensitive points-to: OpenJDK 7 w/o call-graph construction

| <b>Versions</b>                          | <b>Runtime</b> |
|--|----------------|
| SQL Source-to-source translation         | ~8h            |
| RAM / Hashtables                         | ~8h            |
| RAM / Google B-Trees                     | ~3h            |
| Compiler / C++ Templates for Index Order | 15m            |
| + Auto Index                             | 10m            |
| + Fast B-Tree & Indexed Tables           | 2m 40s         |
| + Tries                                  | <1m            |

# Context-Sensitive 201H Points-To Analysis Over OpenJDK 7

[CC'16 – Staged Points-to Analysis for Large Code Bases]

| Tool           | Time      | Memory |
|----------------|-----------|--------|
| BDDDBDDDB      | x         | x      |
| mZ (Z3)        | x         | x      |
| LogicBlox      | x         | x      |
| <b>Soufflé</b> | 4.5 hours | 186 GB |

On Intel Xeon E5-2660 (2.2 GHz) machine with 256 GB RAM  
Time out set to 24 hours

# Soufflé is Open Source

<https://github.com/oracle/souffle/>

<https://github.com/souffle-lang/>

- Listen to a preview of the CAV'16 presentation:
  - <https://youtu.be/8WM0im4RV7M>, “An Experience Report: Efficient Analysis using Souffle”, Bernhard Scholz, University of Sydney, 8<sup>th</sup> July 2016

The Soufflé Datalog compiler provides high performance in program analysis and scales well to the JDK codebase

bernhard.scholz@sydney.edu.au  
cristina.cifuentes@oracle.com



# Integrated Cloud

## Applications & Platform Services

ORACLE®

# 0-Day Exploit Example: CVE-2012-4681

## Public method in sun.awt.Toolkit

```
public static Field getField(final Class klass,  
                             final String fieldName) {  
    return AccessController.doPrivileged(  
        new PrivilegedAction<Field>() {  
            public Field run() {  
                try {  
                    Field field = klass.getDeclaredField(fieldName) ;  
                    field.setAccessible(true) ;  
                    return field ;  
                } ... }  
            } ... }  
        }  
    )  
}
```

# Points-to Analysis: 2-Type-Sensitive + Heap

## Closed vs open world over rt.jar

| Assumption   | Runtime   | # VarPointsTo Facts |
|--------------|-----------|---------------------|
| Closed world | 1:10 hour | ~200 million        |
| Open world   | 14 hours  | ~1 billion          |



Missing out on many inputs and outputs from/to untrusted code