Combining Type-Analysis with Points-To Analysis for Analyzing Java Library Source-Code

Nicholas Allen
Padmanabhan Krishnan
Bernhard Scholz

Oracle Labs,
Brisbane, Australia

ORACLE

Disclaimer

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.

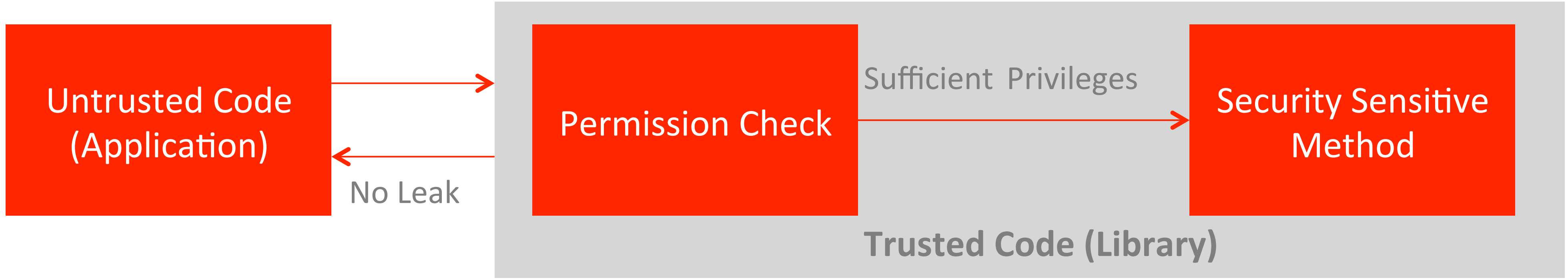
Agenda

- Motivation: Security analysis of libraries
- Background: Points-to, Static analysis
- Types and Most General Application
- Experiments
- Future Work & Conclusion

ORACLE

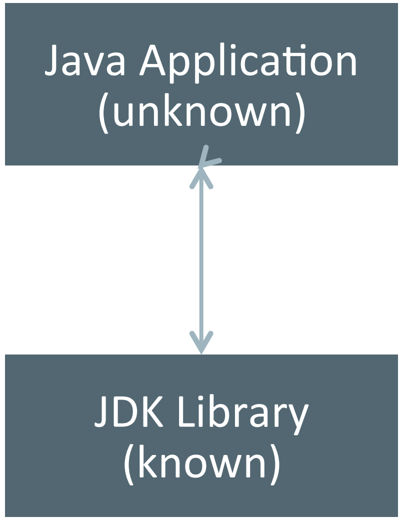
Motivation

- Context: Security vulnerabilities in the JDK library
  - Reason about JDK library without an application
- Security enforced by library
  - Enforcement transparent to application
  - Library code is known vs. application code is unknown



Static Program Analysis Challenge for OO-Programs

- Major building block for security analysis
  - Points-to analysis reasoning about heap and program variables
- Open/closed world problem
  - Application code is ***unknown***
  - JDK Library code is ***known***
- How to reason about unknown applications?
  - Abstractions for interactions between application & library
  - Heap Abstractions for the library for all applications
  - Points-to relationship between variables and heap objects

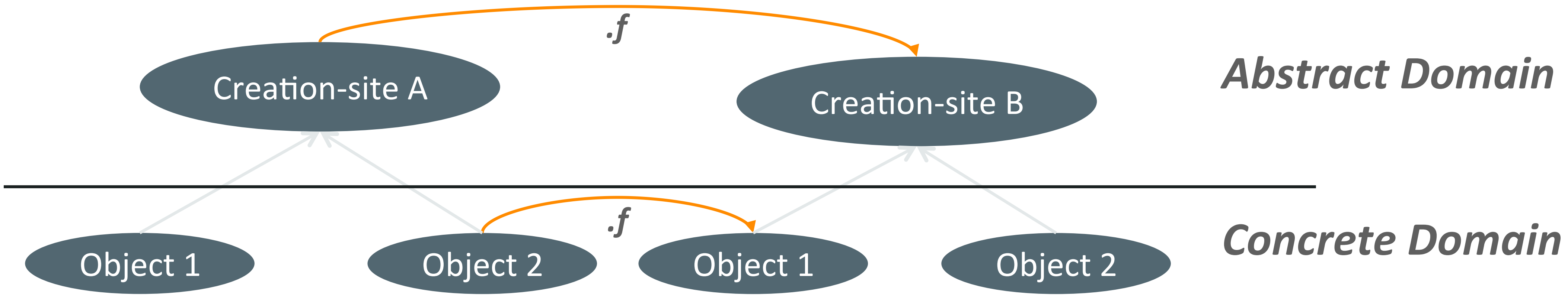


Background

Context-insensitive, flow-insensitive Anderson's style points-to for Java

Points-To Analysis

- Flow-insensitive, inclusion-based, context-insensitive points-to
- Abstract domain
  - Program variables
    - Local, actual/formal parameters, return-values, bases, this-variables
  - Heap-allocated objects
    - Creation-site as an abstraction for dynamically created objects with fields



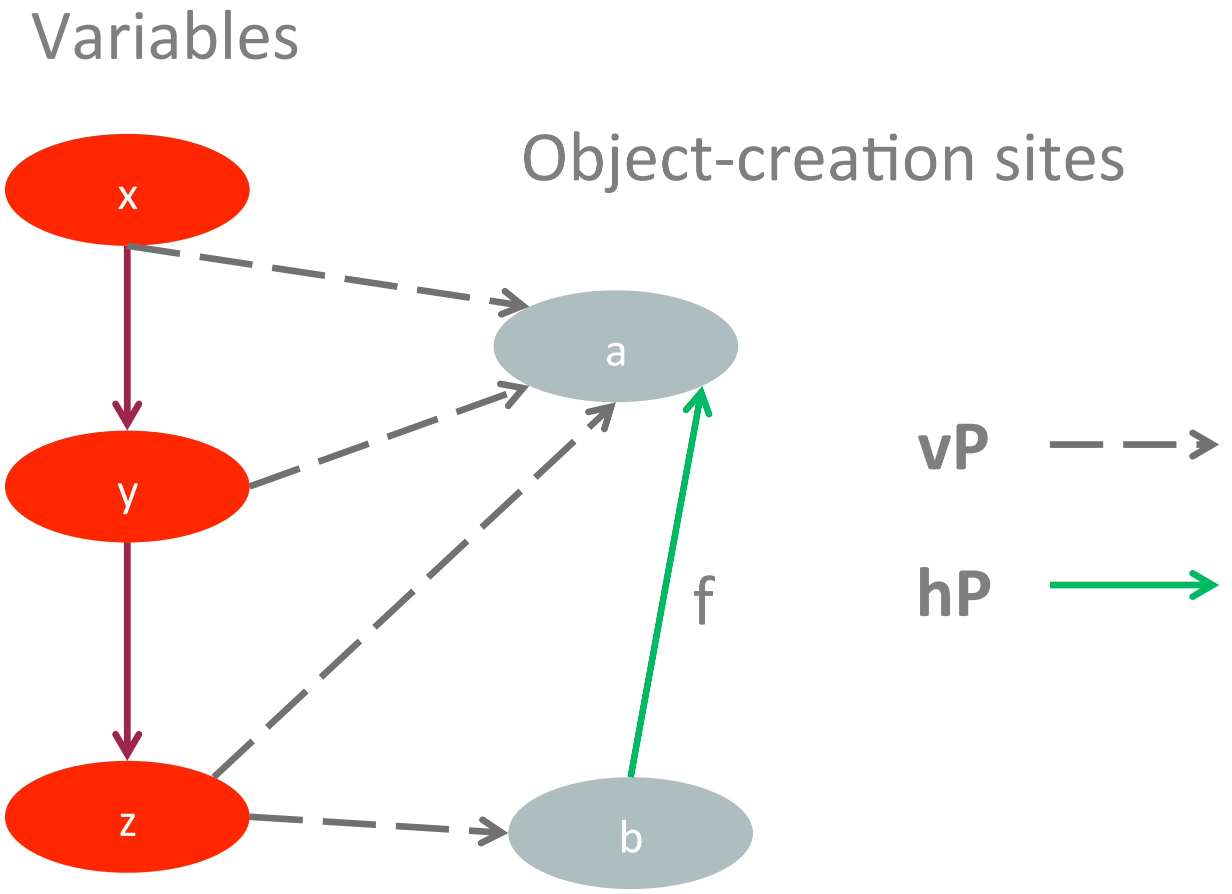
Points-To Analysis in Datalog

	Java Code	Datalog Encoding
Allocations	h: v = new C() ;	vP(v,h) :- “h: v = new C()”.
Store	v_1.f = v_2;	hP(h_1,f,h_2) :- “v_1.f = v_2”, vP(v_1,h_1), vP(v_2,h_2).
Load	v_2 = v_1.f;	vP(v_2, h_2) :- “v_2 = v_1.f”, hP(h_1,f,h_2), vP(v_1,h_1).
Moves, Arguments	v_2 = v_1;	vP(v_2, h) :- “v_2 = v_1”, vP(v_1,h).

- (**v**,**h**) ∈ **vP** if variable **v** may point to an object of creation-site **h**
- ($\mathbf{h_1}$,**f**,$\mathbf{h_2}$) ∈ **hP** if object of $\mathbf{h_1}$ may point to an object of $\mathbf{h_2}$ via field **f**

Points-To Example

```
a:x=new Foo()
y=x;
if (cond) {
   z = y;
} else {
   b:z=new G();
   z.f = y;
}
 …
```



Types and Most General Application

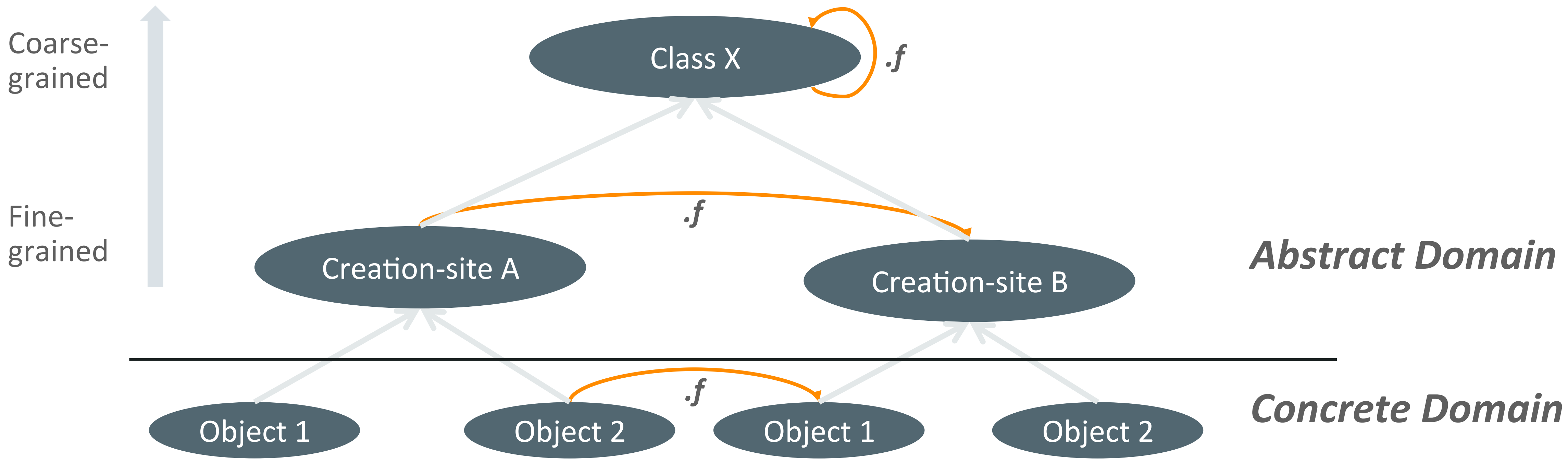
Context-insensitive Anderson's style points-to for Java

Extending Points-To for Unknown Applications

- Applications have a contract how they interact with the library
  - ***Via Types and via public interfaces***
  - Enforced by the programming language
- Extend the points-to analysis with types
  - Abstract domain is extended
  - Semantic equations are altered
- New Abstract Domain
  - Object-creation sites can be summarized by their classes/types
  - Reason about objects from unknown application code

Amalgamate Points-To with Type Analysis

- Assume creation-site A and B create instances of class X



Classes: Universal Quantifiers for Object-Creation sites

- Interpretation of a class ***X*** in abstract domain:
  - Subsumes all object-creation sites of class ***X*** and its sub-classes
  - May subsume unknown and known object-creation sites
  - Sub-classes may be known or unknown
- Heap-Abstraction
  - Classes (vs. object-creation sites) produce higher abstraction level
  - If there exists an edge between two classes ***X*** and ***Y*** via field ***f***
    - there exist two object creation sites of type ***X*** and ***Y*** connected by field ***f***
- Lattice permits co-existence of type- and points-to analysis

ORACLE

Points-To Analysis with Types

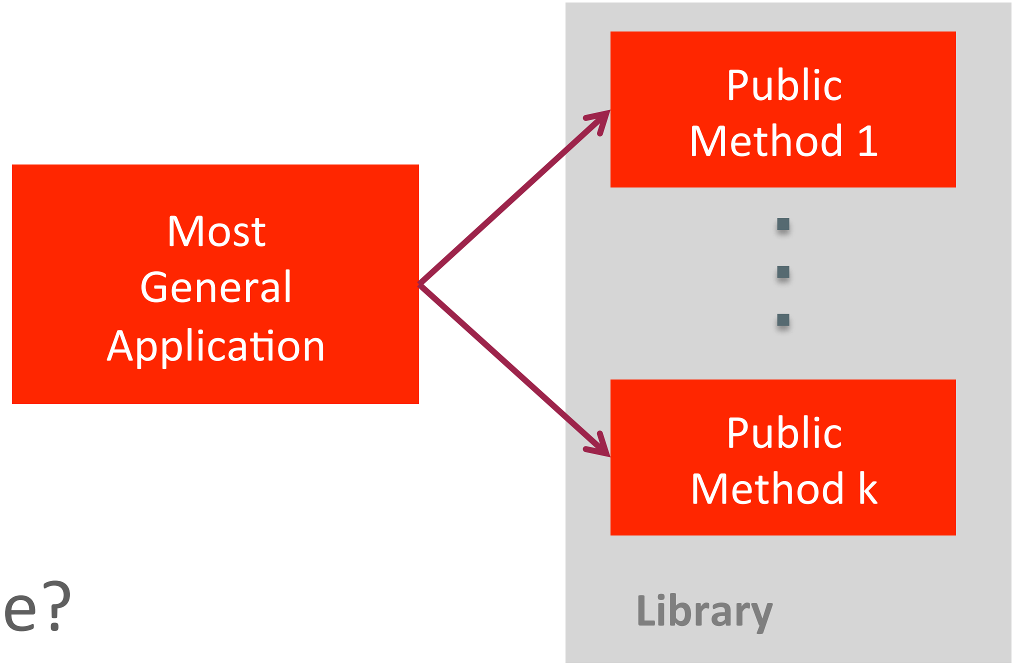
	Java Code	Datalog Encoding
Allocations	h: v = new C() ;	vP(v,h) :- "h: v = new C()".
Store	v_1.f = v_2;	hP(o,f,h_2) :- "v_1.f = v_2", vP(v_1,h_1), **isOf(o,h_1)**, vP(v_2,h_2).
Load	v_2 = v_1.f;	vP(v_2, h_2) :- "v_2 = v_1.f", hP(h_1,f,h_2), vP(v_1,h_1).
Moves, Arguments	v_2 = v_1;	vP(v_2, h) :- "v_2 = v_1", vP(v_1,h).

- Relations **vP** and **hP** extended for types,
- Adapt store semantics: if type t ➔ update all creation-sites of type t

ORACLE

Interactions with Unknown Applications

- Construct Most General Application
  - Mimicking the behaviour of all applications
  - Over-approximation
- Worst-case assumptions
  - All public interfaces are called by MGA
  - Parameters of invocations ***are types***
  - However, no program variables in MGA
- Which heap-abstraction as an initial state?

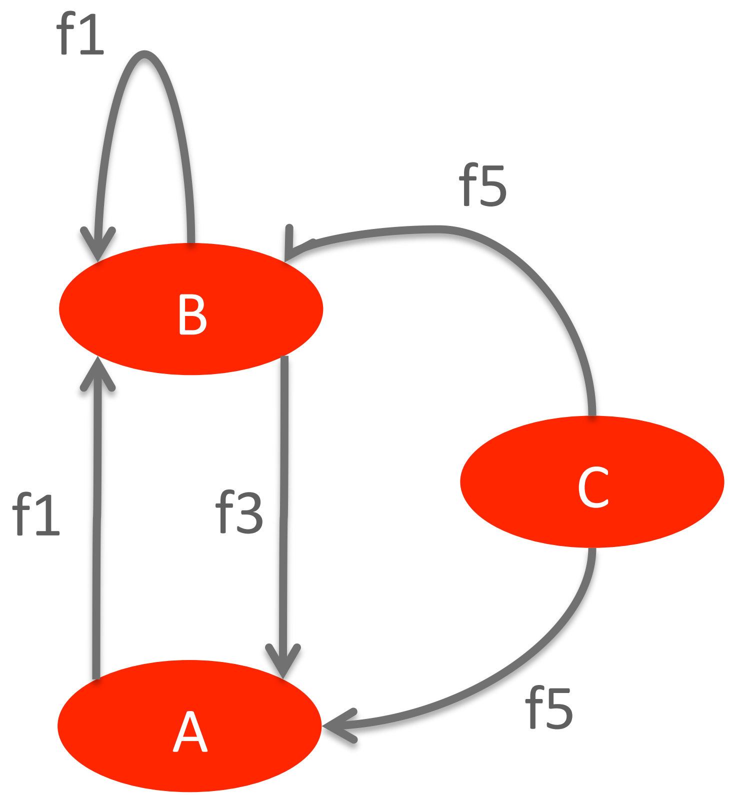


Initial State for Most General Application (MGA)

- Construction
  - Nodes of the initial heap abstractions are public classes
  - Connect class with public accessible fields in heap abstraction with their type
  - Public sub-classes inherit connection
- Private fields are excluded
  - Only library can change field contents
- Less connection in the initial state produce makes points-to more precise
- Assumption
  - Object/Root class is owned by library

Example: Initial State

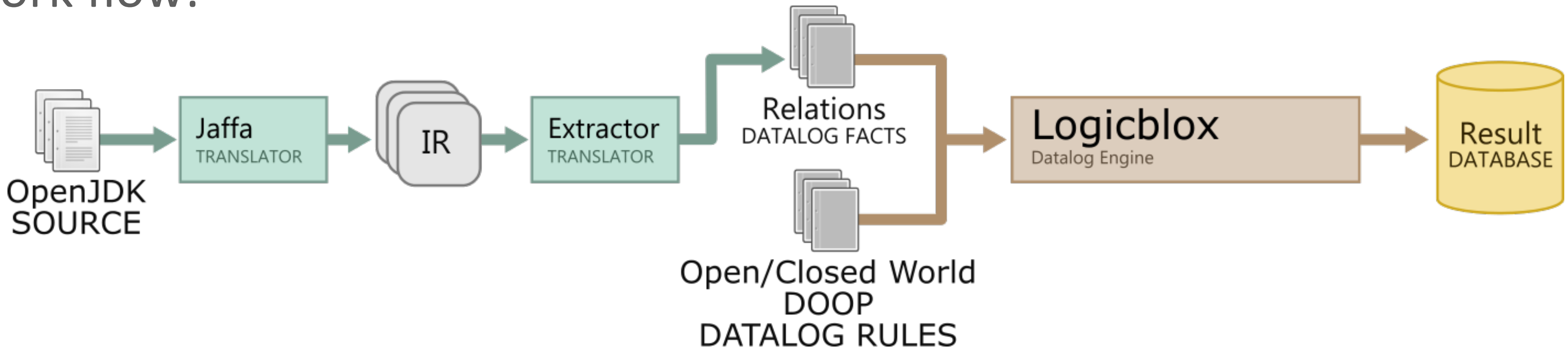
```
class A {
 public B f1;
 private C f2; }

class B extends A {
 public A f3;
 private A f4 ; }

class C {
 public A f5; }
```



Experiments

Experimental Setup

- Extended DOOP framework for open/closed world assumption
- DOOP runs on Logicblox
- Analysis of OpenJDK library: version 7, build 147
- Machine: Intel Xeon E5-2660 (2.2GHz), 256GB Ram
- Work flow:



Experiment with OpenJDK 7 / build 147

	CHA	without MGA	with MGA
Call-Graph Edges (#)	3,030,157	378,495	851,127
Points-To (#)	n/a	384,207,724	661,970,750
Runtime (seconds)	30	1719	3662

- Class Hierarchy Analysis (CHA)
  - Type analysis of objects
- Points to analysis with Most General Application (MGA)
  - 124% more call graph edges in call graph
  - 73% more variable/object relations in points-to set

ORACLE

Future Work

- How to deal with reflection?
- Can the notion of MGA be extended to security properties?
- How to prove the semantic correctness?
- How to improve precision and runtime of the specifications?

Conclusion

- Amalgamate type-analysis with points-to
- Types used to summarize unknown objects in application
- Analysis handles types and creation-sites uniformly
- Over-approximating applications with Most General Application
- Overhead for large code is manageable

Hardware and Software

Engineered to Work Together