Is Program Analysis The Silver Bullet Against Software Bugs?

Papers We Love Conference — 2019

Karim Ali
University of Alberta
@karimhamdanali

Software Bugs

Invalid SSL/TLS connections earned Apple Most Epic Fail [Pwnie ’14]

Source: CVE-2014-1266

Software Bugs

Errors in ABS software led to fatal accidents and cost Toyota $3 Billion

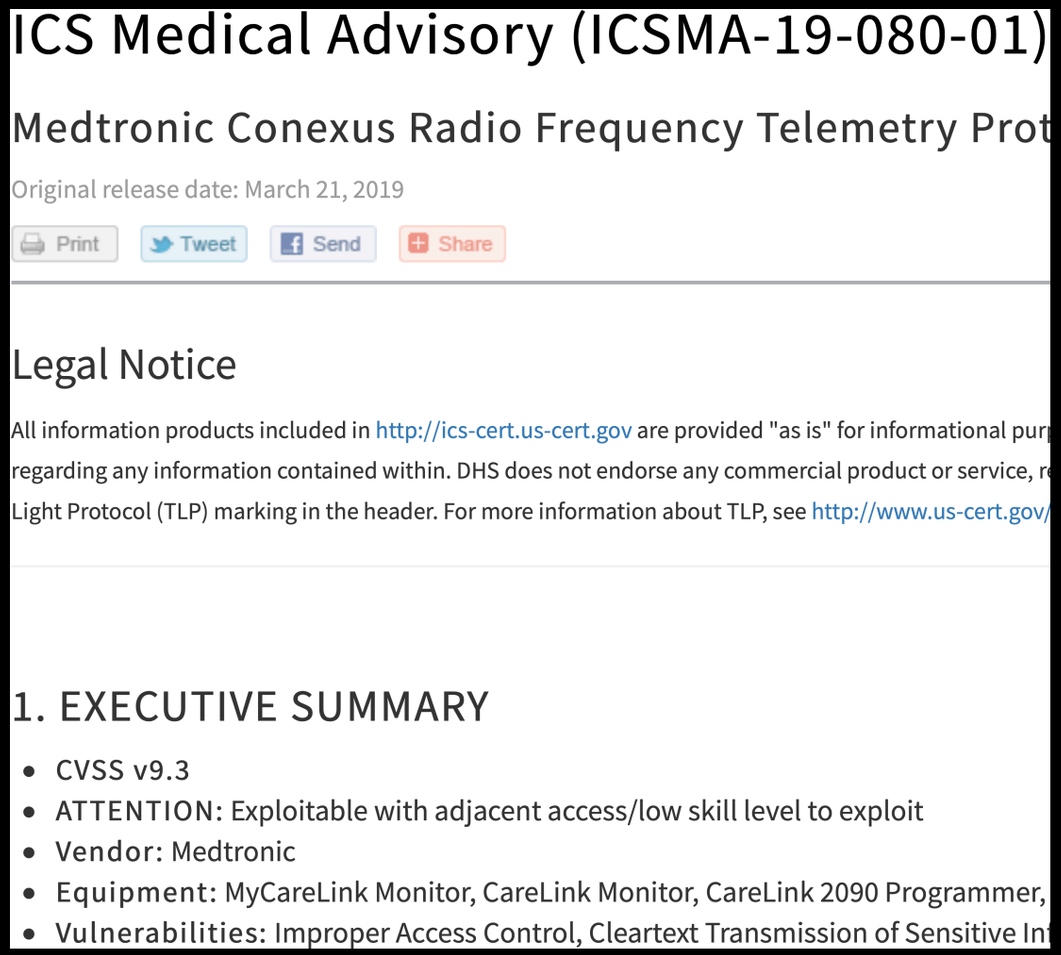
Toyota "Unintended Acceleration" Has Killed 89

A 2005 Toyota Prius, which was in an accident, is seen at a police station in Harrison, New York, Wednesday, March 10, 2010. The driver of the Toyota Prius told police that the car accelerated on its own, then lurched down a driveway, across a road and into a stone wall. (AP Photo/Seth Wenig) / **AP PHOTO/SETH WENIG**

Source: Philip Koopman, CMU

Software Bugs

Unencrypted, unauthenticated connections to some medical implants



ICS Medical Advisory (ICSMA-19-080-01)

Medtronic Conexus Radio Frequency Telemetry Prot

Original release date: March 21, 2019

Print Tweet Send Share

Legal Notice

All information products included in http://ics-cert.us-cert.gov are provided "as is" for informational pur
regarding any information contained within. DHS does not endorse any commercial product or service, r
Light Protocol (TLP) marking in the header. For more information about TLP, see http://www.us-cert.gov/

1. EXECUTIVE SUMMARY

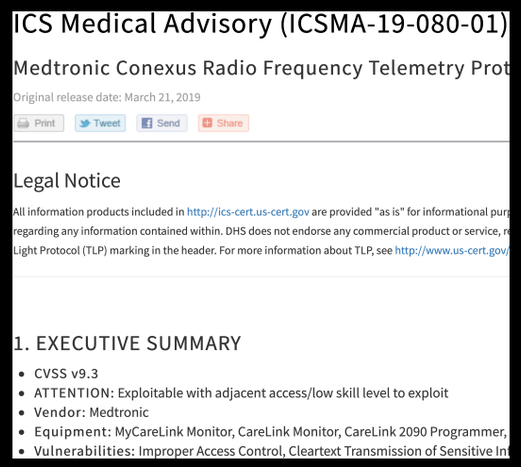
- CVSS v9.3
- ATTENTION: Exploitable with adjacent access/low skill level to exploit
- Vendor: Medtronic
- Equipment: MyCareLink Monitor, CareLink Monitor, CareLink 2090 Programmer,
- Vulnerabilities: Improper Access Control, Cleartext Transmission of Sensitive In

Source: Department of Homeland Security

Toyota "Unintended Acceleration" Has Killed 89

A 2005 Toyota Prius, which was in an accident, is seen at a police station in Harrison, New York, Wednesday, March 10, 2010. The driver of the Toyota Prius told police that the car accelerated on its own, then lurched down a driveway, across a road and into a stone wall. (AP Photo/Seth Wenig) AP PHOTO/SETH WENIG

Program Analysis



ICS Medical Advisory (ICSMA-19-080-01)

Medtronic Conexus Radio Frequency Telemetry Prot

Original release date: March 21, 2019

Print Tweet Send Share

Legal Notice

All information products included in http://ics-cert.us-cert.gov are provided "as is" for informational purp
regarding any information contained within. DHS does not endorse any commercial product or service, re
Light Protocol (TLP) marking in the header. For more information about TLP, see http://www.us-cert.gov/

1. EXECUTIVE SUMMARY

- CVSS v9.3
- ATTENTION: Exploitable with adjacent access/low skill level to exploit
- Vendor: Medtronic
- Equipment: MyCareLink Monitor, CareLink Monitor, CareLink 2090 Programmer,
- Vulnerabilities: Improper Access Control, Cleartext Transmission of Sensitive In

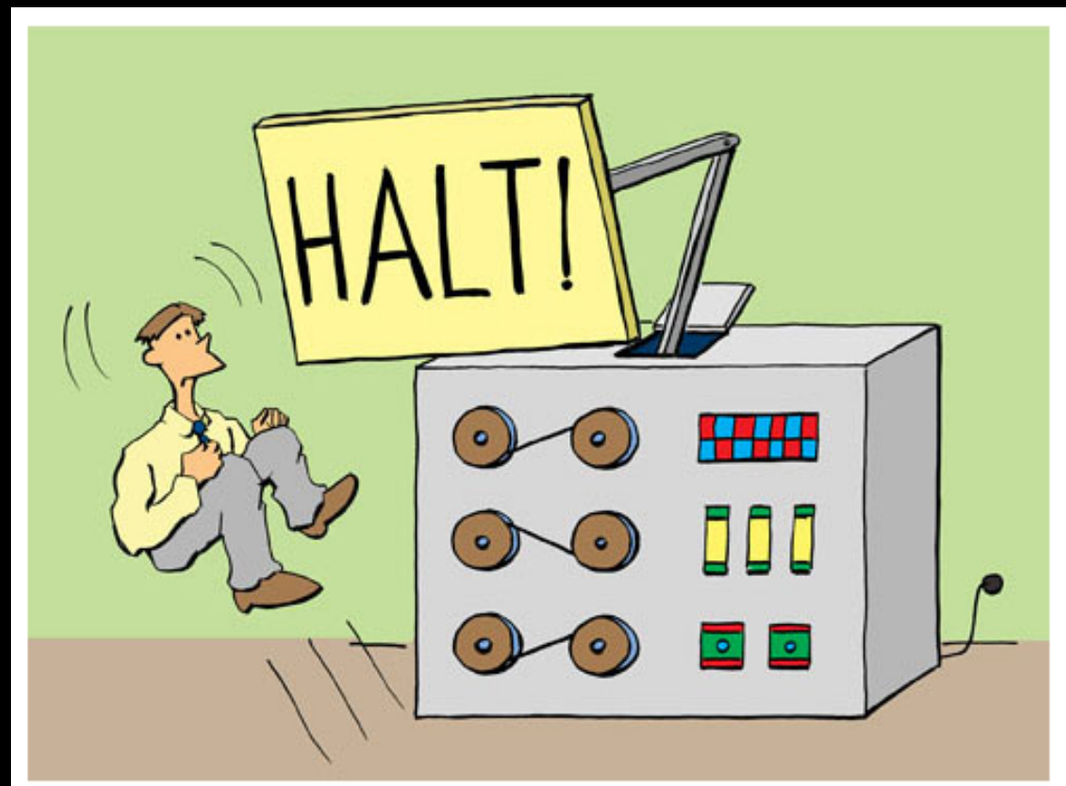
What is Program Analysis?

Program Analysis

A way of reasoning about the runtime behaviour of a program without necessarily executing it

Rice's Theorem

"For **any** interesting property *Pr* of the behaviour of a program, it is **impossible** to write an analysis that can decide for every program *p* whether *Pr* holds for *p*."



Alan designed the perfect computer

Image: CooperToons

By definition, program analysis is **undecidable**

Not quite…

Image: J. K. Simmons / Whiplash

Program Analysis

- Settle for an approximation of *Pr*
- Make it as “good” as possible

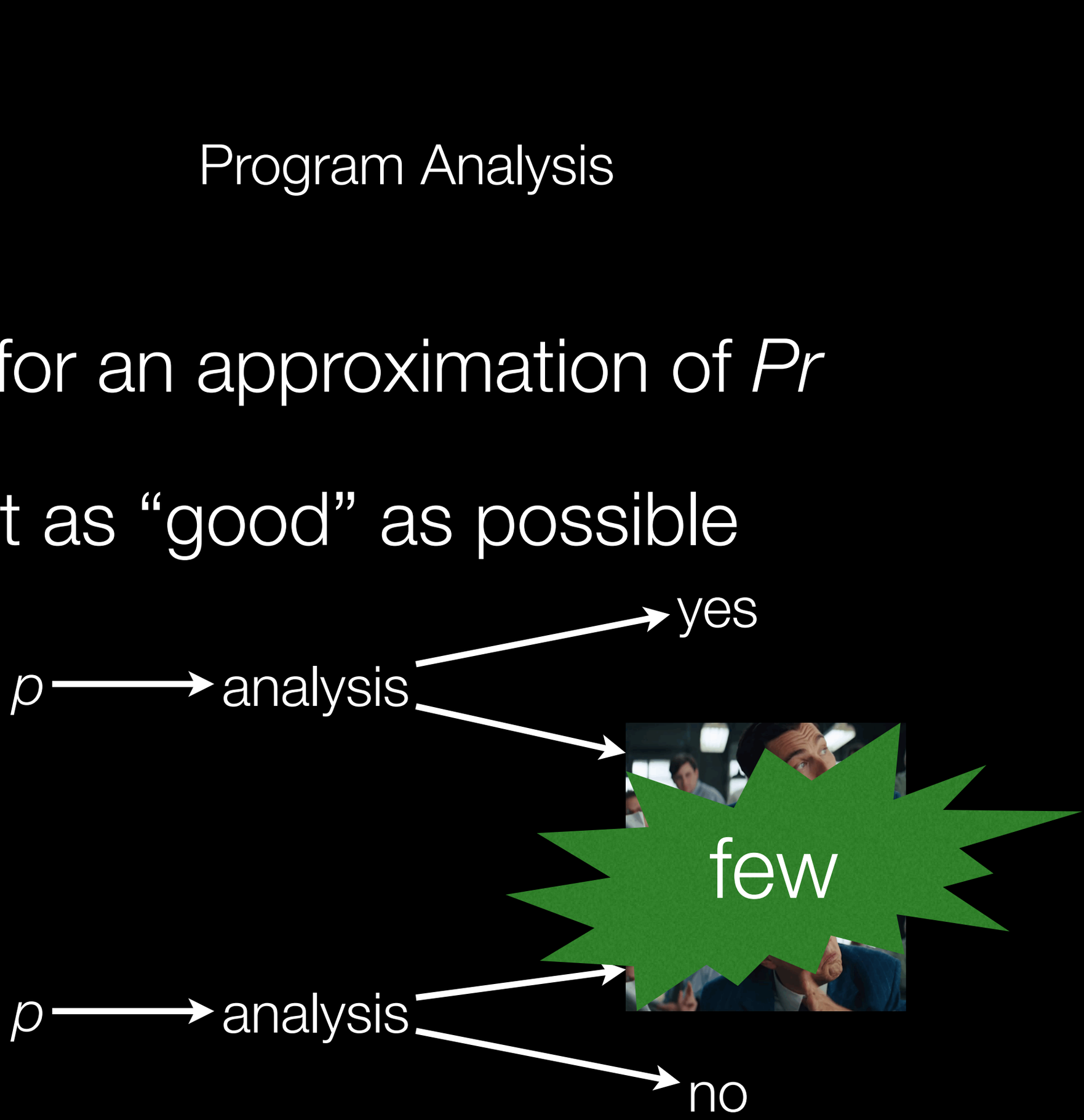
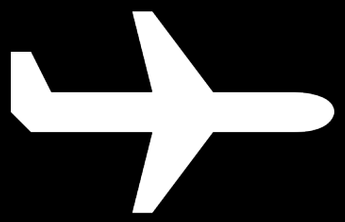
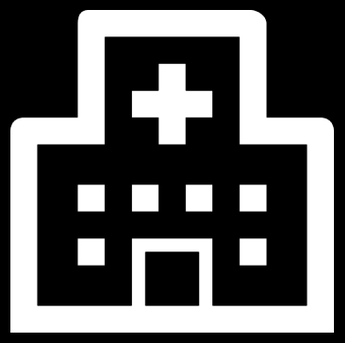


Image: Jenna Mullins / ENews





Code Navigation

Code Refactoring

Code Recommenders

Program Analysis

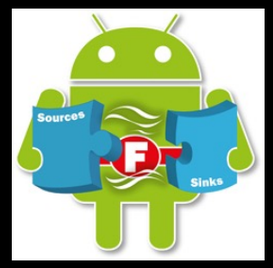
Parallelization

Constant Propagation

Static Inlining

Dead Code Elimination

CHECKMARX



UNIVERSITY OF MARYLAND
FindBugs
because it's easy

coverity
A Synopsys Company

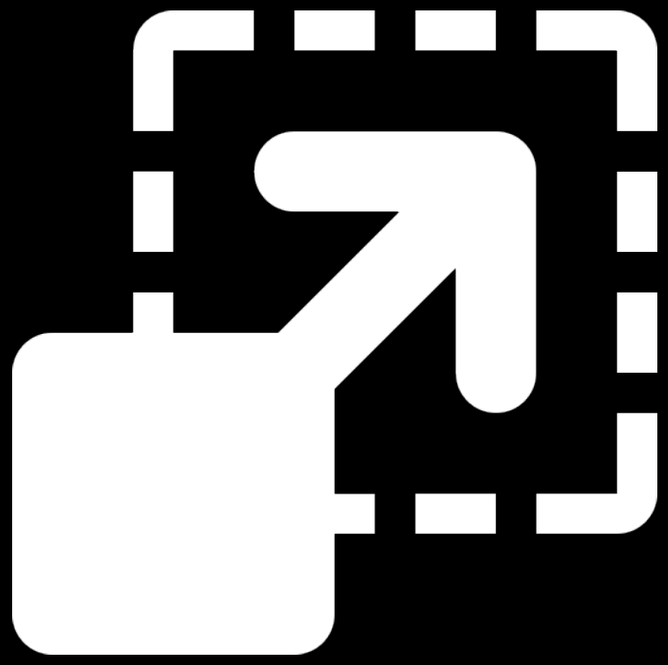
FORTIFY

AbsInt

Program Analysis in Practice

Image: Minion Special / YouTube

Program Analysis in Practice



Scalability

Precision

Usability

Collaborators

- Erick Ochoa (UAlberta)
- Spencer Killen (UAlberta)
- Kristen Newbury (UAlberta)
- Revan MacQueen (UAlberta)
- Daniil Tiganov (UAlberta)
- Jeff Cho (UAlberta)
- Johannes Späth (Paderborn)
- Lisa Nguyen (Paderborn)
- Stefan Krüger (Paderborn)
- Ondřej Lhoták (Waterloo)
- Frank Tip (Northeastern)
- Eric Bodden (Paderborn & Fraunhofer IEM)
- Mira Mezini (TU Darmstadt)
- Julian Dolby (IBM Research)
- Andrew Craik (IBM)
- Mark Stoodley (IBM)
- Vijay Sundaresan (IBM)
- Ben Livshits (Imperial College London & Brave)
- Emerson Murphy-Hill (Google)
- Justin Smith (Lafayette College)
- José Nelson Amaral (UAlberta)
- James Wright (UAlberta)
- Kirsten Thommes (Paderborn)
- René Fahr (Paderborn)

Collaborators

- Erick Ochoa (UAlberta)
- Spencer Killen (UAlberta)
- Kristen Newbury (UAlberta)
- Revan MacQueen (UAlberta)
- Daniil Tiganov (UAlberta)
- Jeff Cho (UAlberta)
- Johannes Späth (Paderborn)
- Lisa Nguyen (Paderborn)
- Stefan Krüger (Paderborn)
- **Ondřej Lhoták (Waterloo)**
- Frank Tip (Northeastern)
- Eric Bodden (Paderborn & Fraunhofer IEM)
- Mira Mezini (TU Darmstadt)
- Julian Dolby (IBM Research)
- Andrew Craik (IBM)
- Mark Stoodley (IBM)
- Vijay Sundaresan (IBM)
- Ben Livshits (Imperial College London & Brave)
- Emerson Murphy-Hill (Google)
- Justin Smith (Lafayette College)
- José Nelson Amaral (UAlberta)
- James Wright (UAlberta)
- Kirsten Thommes (Paderborn)
- René Fahr (Paderborn)

2010

2010

2010

Where do I begin?

2010

Where do I begin?

Start with this paper!

JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2007; **19**:231–252
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr.351

Research

Automatic construction of accurate application call graph with library call abstraction for Java

Weilei Zhang*, † and Barbara G. Ryder

Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ, 08854, U.S.A.

SUMMARY

Call graphs are widely used to represent calling relationships among methods. However, there is not much interest in calling relationships among library methods in many software engineering applications, such as program understanding and testing, especially when the library is very big and the calling relationships are not trivial. This paper explores approaches for generating more accurate application call graphs for Java. A new data reachability algorithm is proposed and fine tuned to resolve library callbacks accurately. Compared with an algorithm that resolves library callbacks by traversing the whole-program call graph, the fine-tuned data reachability algorithm results in fewer spurious callback edges. In empirical studies, the new algorithm shows a significant reduction in the number of spurious callback edges. On the basis of the new algorithm, a library abstraction can be calculated automatically and applied in amortized slicing and dataflow testing. Copyright © 2007 John Wiley & Sons, Ltd.

Received 29 January 2007; Revised 8 May 2007; Accepted 8 June 2007

KEY WORDS: call graph; library callback; Java analysis

1. INTRODUCTION

Call graphs are widely used as a program representation in software engineering and optimizing compilation. Construction of call graphs is usually straightforward in classical procedural languages; for example, in *C*, barring the use of function pointers, a call site has exactly one possible callee.

*Correspondence to: Weilei Zhang, Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscat away, NJ, 08854, U.S.A.
†E-mail: weileiz@cs.rutgers.edu

J. Softw. Maint. Evol.: Res. Pract. 2007; **19**:231–252
Published online in Wiley InterScience (www.interscience.wiley.com). DOI:

Research

Automatic construction of accurate application call graph with library call abstraction for Java

Weilei Zhang*, † and Barbara G. Ryder

Department of Computer Science, Rutgers University, 110 Freling

… so what is a Call Graph?

Call Graph

Scalable Propagation-Based Call Graph Construction Algorithms

Frank Tip
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
tip@watson.ibm.com

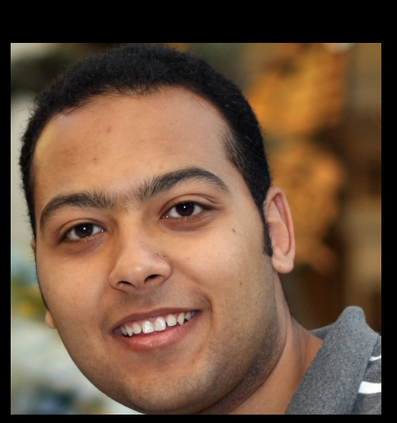
Jens Palsberg
Dept. of Computer Science
Purdue University
West Lafayette, IN 47907
palsberg@cs.purdue.edu

ABSTRACT

Propagation-based call graph construction algorithms have been studied intensively in the 1990s, and differ primarily in the number of sets that are used to approximate run-time values of expressions. In practice, algorithms such as RTA that use a single set for the whole program scale well. The scalability of algorithms such as 0-CFA that use one set per expression remains doubtful.

In this paper, we investigate the design space between RTA and 0-CFA. We have implemented various novel algorithms in the context of *Jax*, an application extractor for Java, and

Call-graph construction algorithms have been studied intensively in the 1990s. While their original formulations use a variety of formalisms, most of them can be recast as set-based analyses. The common idea is to abstract an object into the name of its class, and to abstract a set of objects into the set of their classes. For any given call site $e.m()$, the goal is then to compute a set of class names S_e that approximates the run-time values of the receiver expression e. Once the sets S_e are determined for all expressions e, the class hierarchy can be examined to identify the methods that can be invoked.



Strictly Declarative Specification of Sophisticated Points-to Analyses

Martin Bravenboer Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
martin.bravenboer@acm.org yannis@cs.umass.edu

Abstract

We present the Doop framework for points-to analysis of Java programs. Doop builds on the idea of specifying pointer analysis algorithms declaratively, using Datalog: a logic-based language for defining (recursive) relations. We carry the declarative approach further than past work by describing the full end-to-end analysis in Datalog and optimizing aggressively using a novel technique specifically targeting highly recursive Datalog programs.

As a result, Doop achieves several benefits, including full

analyses. It is, thus, not surprising that a wealth of research has been devoted to efficient and precise pointer analysis techniques. *Context-sensitive* analyses are the most common class of precise points-to analyses. Context sensitive analysis approaches qualify the analysis facts with a *context* abstraction, which captures a static notion of the dynamic context of a method. Typical contexts include abstractions of method call-sites (for a *call-site sensitive* analysis—the traditional meaning of "context-sensitive") or receiver objects (for an *object-sensitive* analysis).

Call Graph

```
Shape s;
if(*) s = new Circle();
else s = new Square();
```

```
s.draw();
```

```
class Circle extends Shape
{ void draw() { ... } }
```

```
class Square extends Shape
{ void draw() { ... } }
```

Call Graph

```
Shape s;
if(*) s = new Circle();
else s = new Square();
s.draw();
```

```
class Circle extends Shape
{ void draw() { ... } }

class Square extends Shape
{ void draw() { ... } }
```

required by every inter-procedural analysis

Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```

Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```

Main.main()

Let’s build a Call Graph

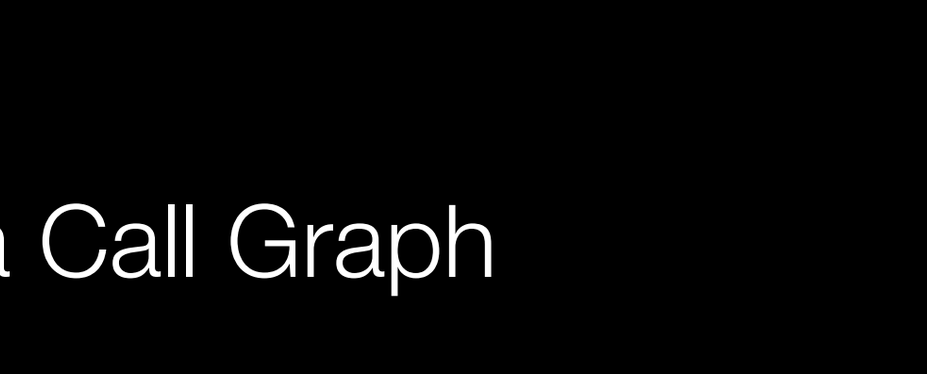
```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```



Let’s build a Call Graph

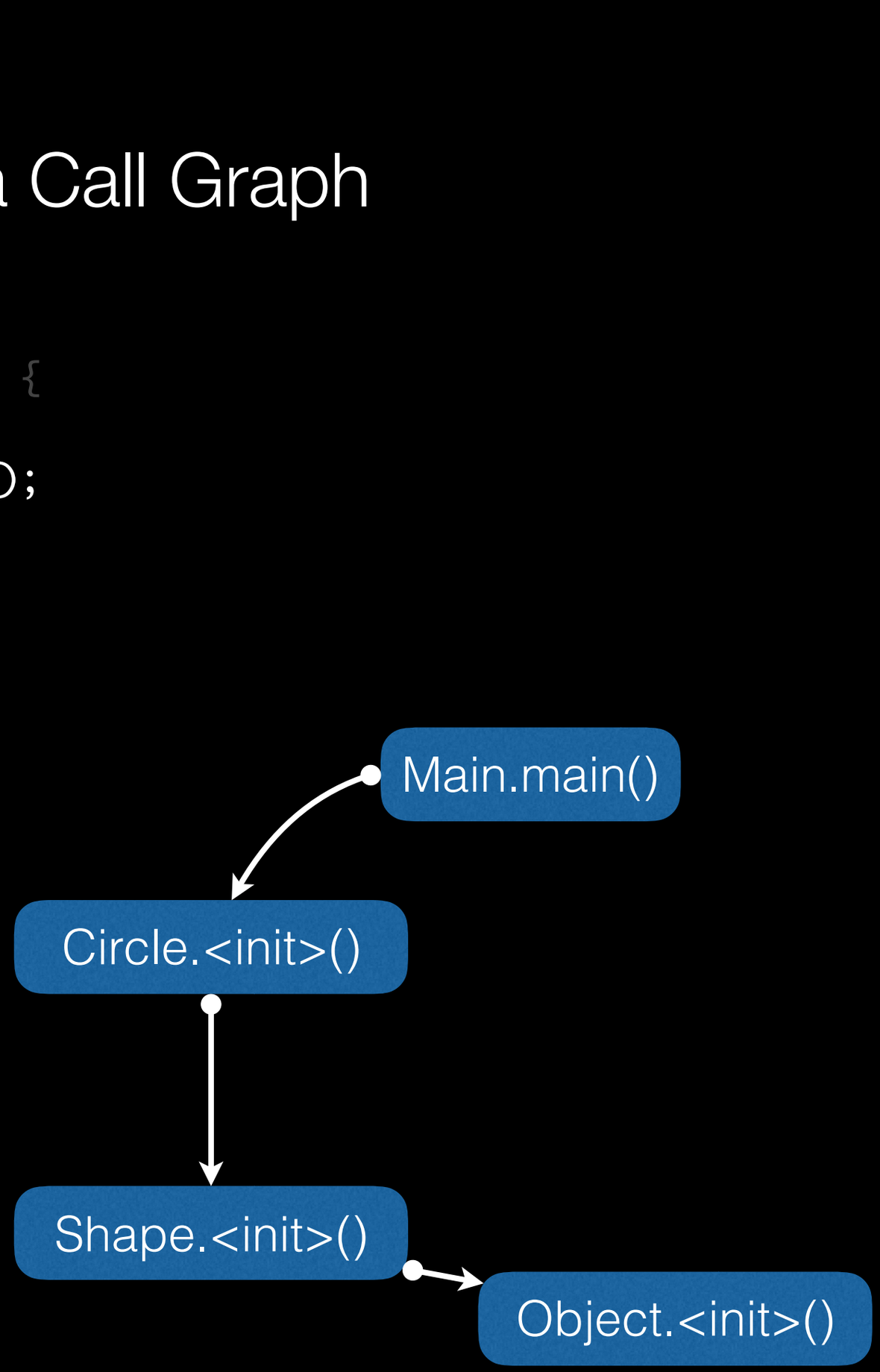
```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```



Let’s build a Call Graph

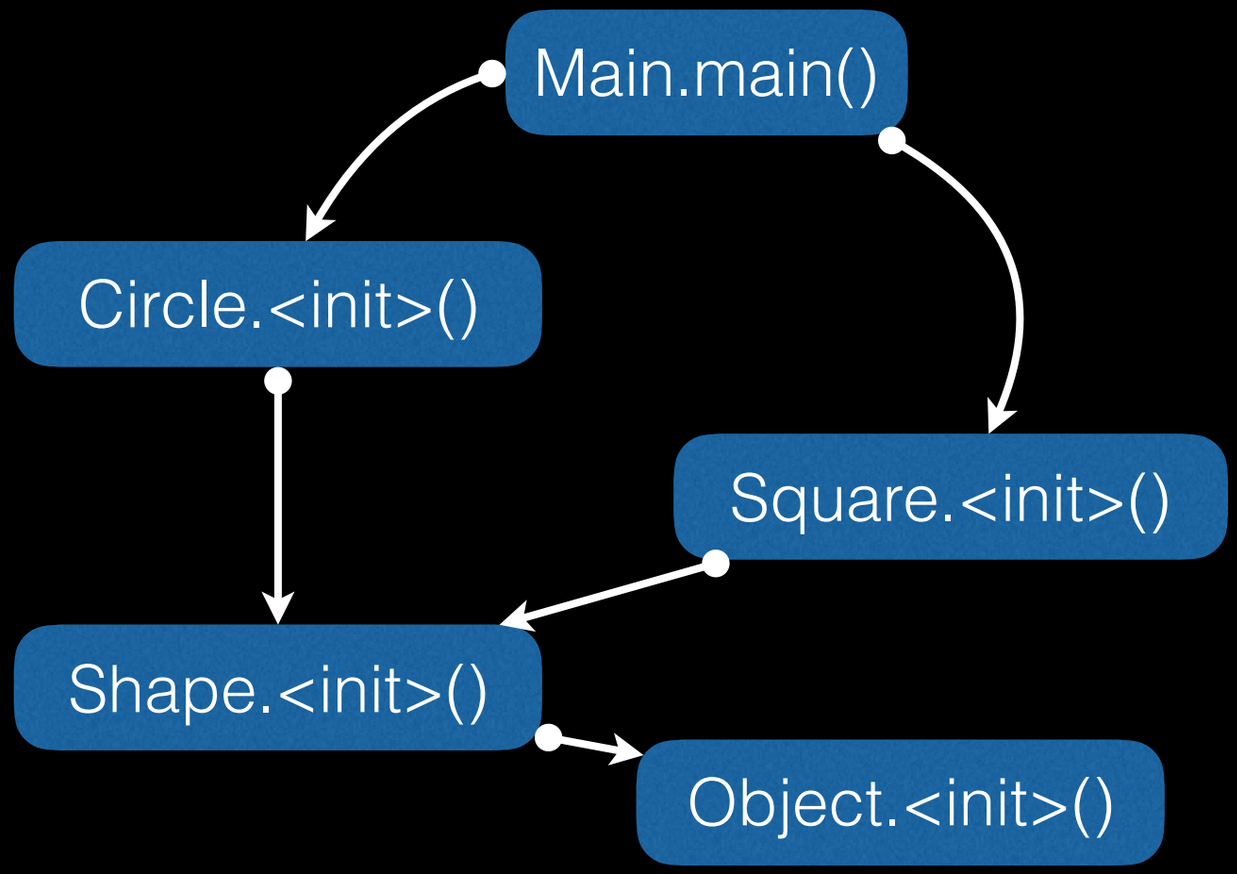
```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```



Let’s build a Call Graph

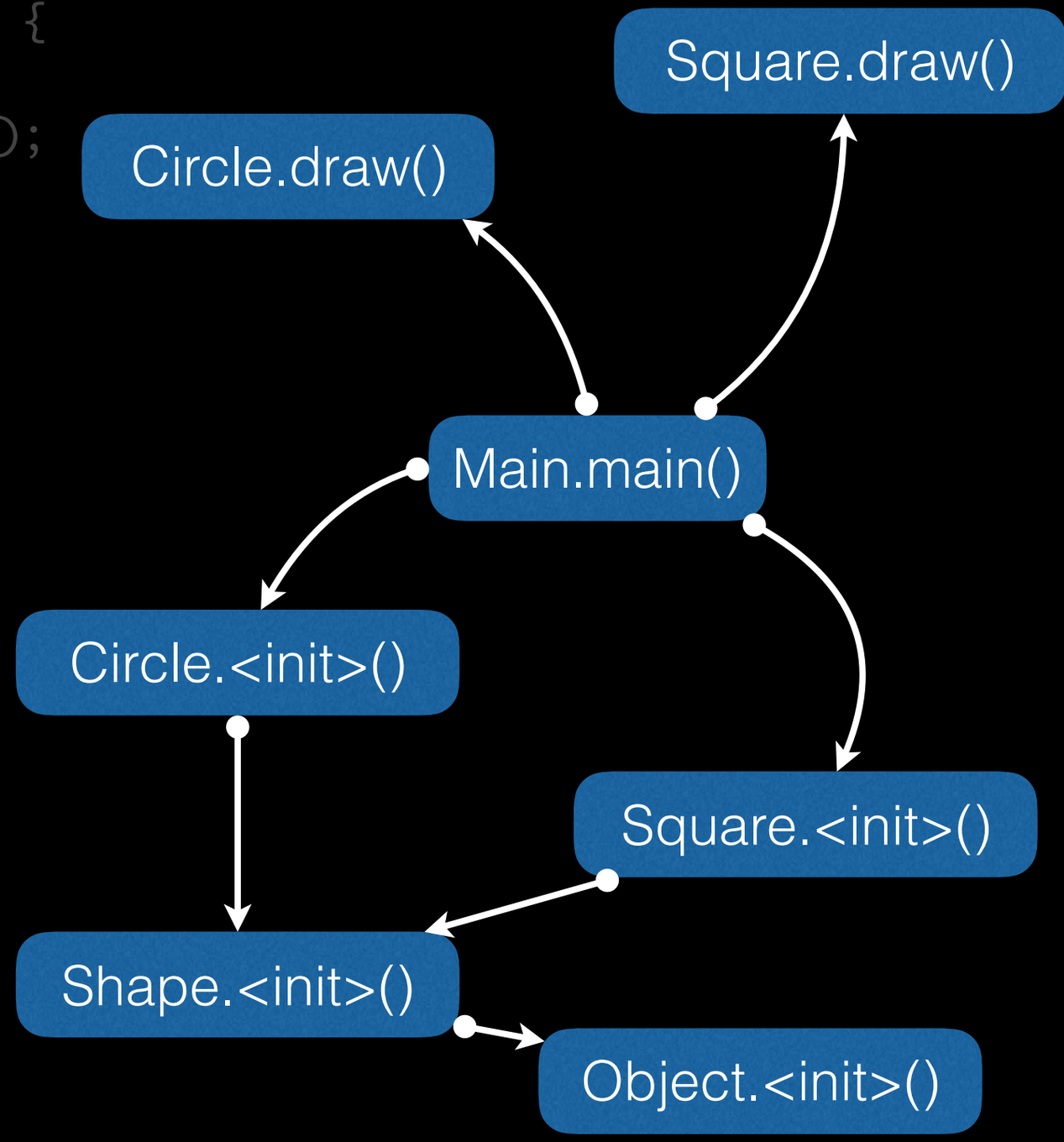
```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```



J. Softw. Maint. Evol.: Res. Pract. 2007; **19**:231–252
Published online in Wiley InterScience (www.interscience.wiley.com). DOI:

Research

Automatic construction of accurate application call graph with library call abstraction for Java

Weilei Zhang*, † and Barbara G. Ryder

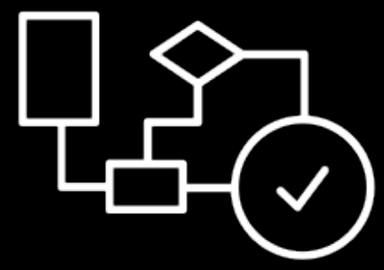
Department of Computer Science, Rutgers University, 110 Freling

Let’s build a Call Graph for `javac`

Let's build a Call Graph for `javac`

- Java 1.4
- 0.5 MB of class files
- 8 GB of RAM
- HOURS!

IRIS Reasoner



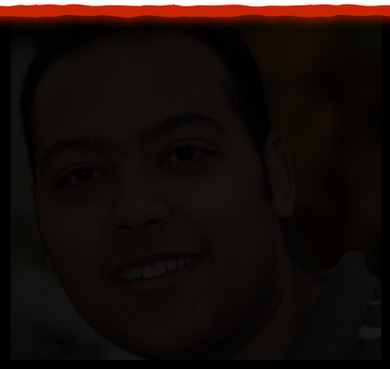
Let’s build a Call Graph for `javac`

.CLASS

IRIS Reasoner

```
Exception in thread “main"
java.lang.OutOfMemoryError: Java heap space
```

- 0.5 MB of class files
- 8 GB of RAM
- HOURS!



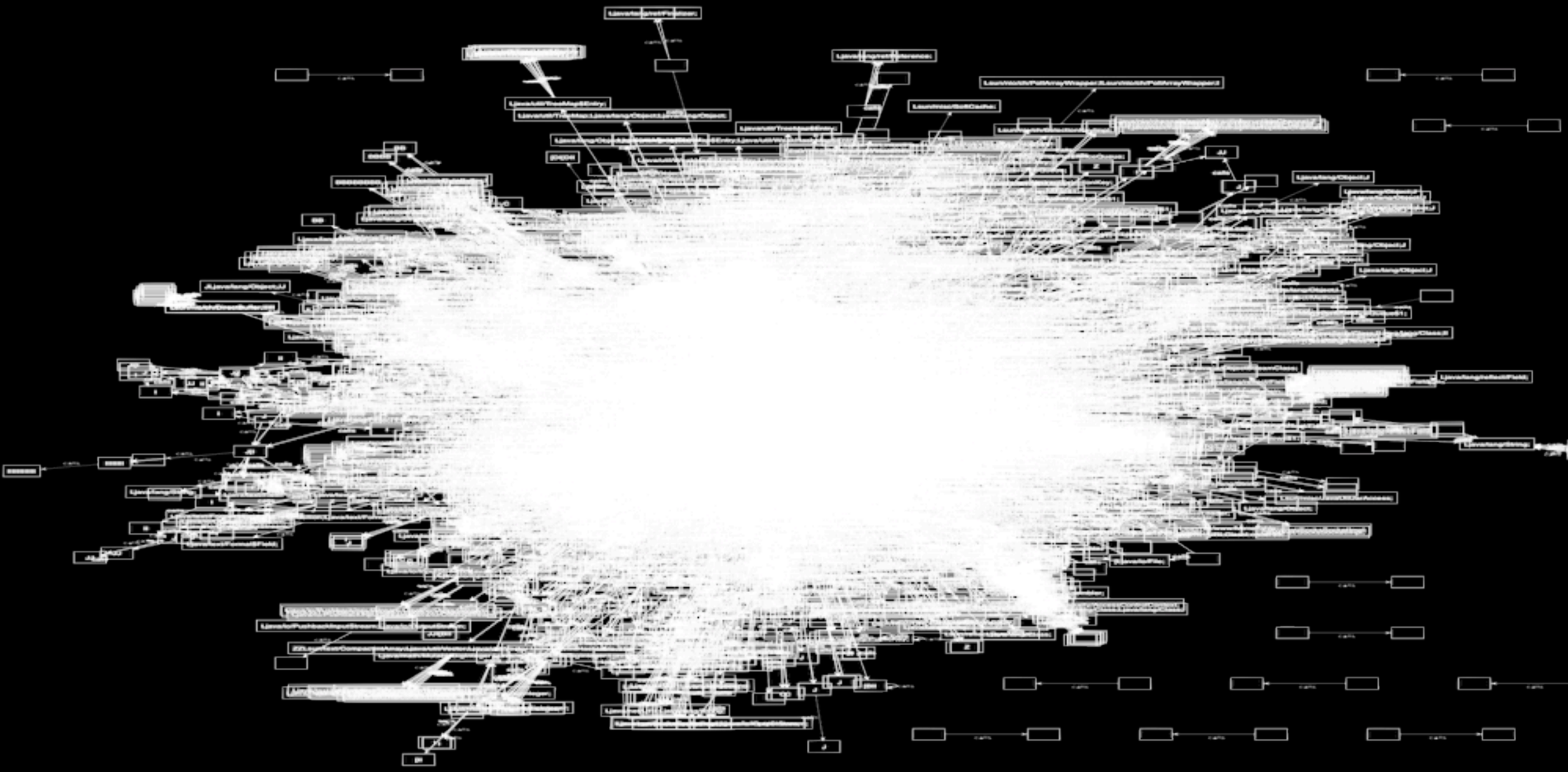
Let’s build a Call Graph for "Hello, World!"

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

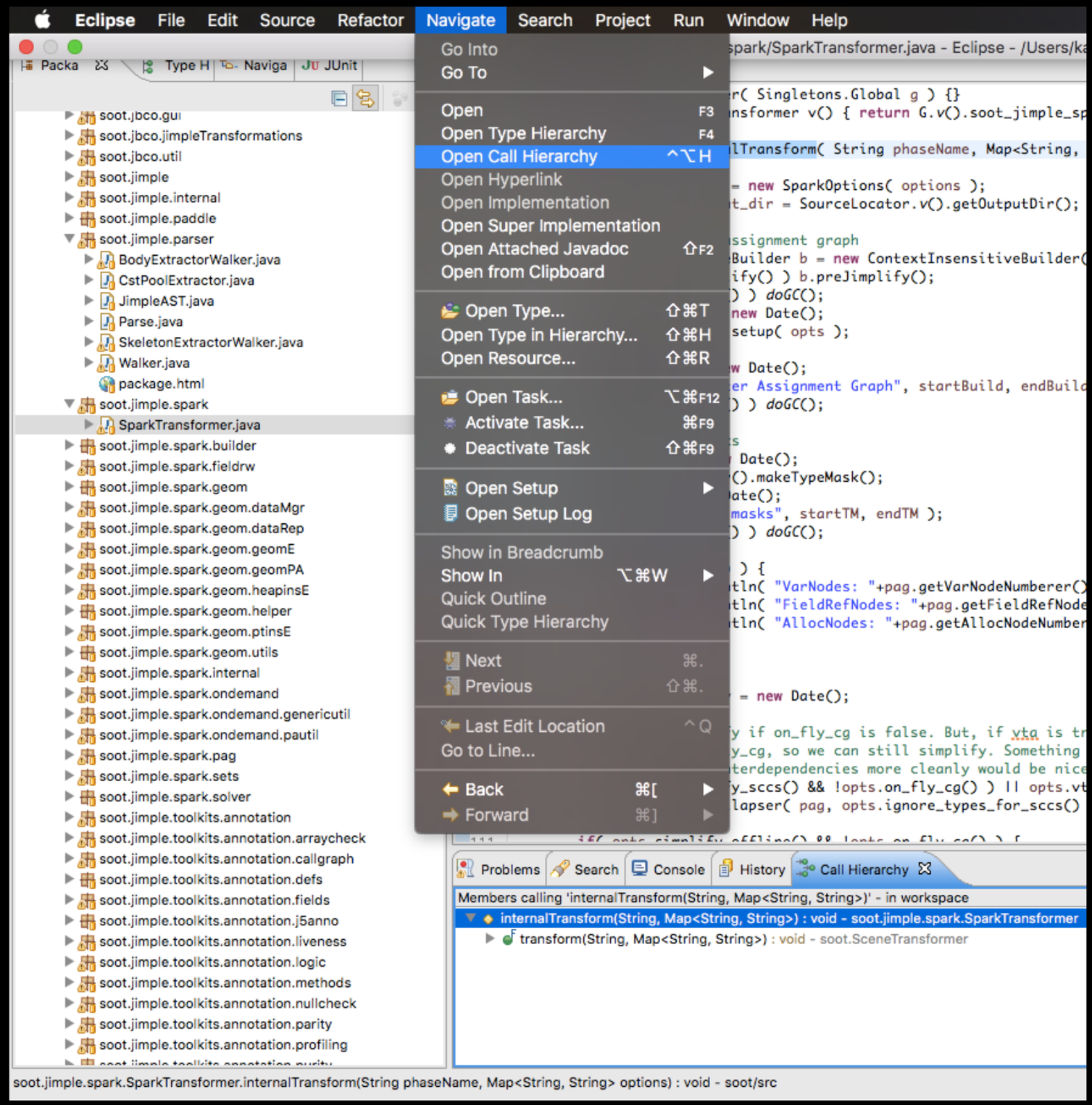
```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

- > 30 seconds
- > 5,000 reachable methods
- > 23,000 call edges

Hello, World!




Java


Eclipse File Edit Source Refactor Navigate Search Project Run Window Help
Go Into
Go To
Open F3
Open Type Hierarchy F4
Open Call Hierarchy ^⌥H
Open Hyperlink
Open Implementation
Open Super Implementation
Open Attached Javadoc ⇧F2
Open from Clipboard
Open Type... ⇧⌘T
Open Type in Hierarchy... ⇧⌘H
Open Resource... ⇧⌘R
Open Task... ⌥⌘F12
Activate Task... ⌘F9
Deactivate Task ⇧⌘F9
Open Setup
Open Setup Log
Show in Breadcrumb
Show In ⌥⌘W
Quick Outline
Quick Type Hierarchy
Next ⌘.
Previous ⇧⌘.
Last Edit Location ^Q
Go to Line...
Back ⌘[
Forward ⌘]
soot.jimple.parser
BodyExtractorWalker.java
CstPoolExtractor.java
JimpleAST.java
Parse.java
SkeletonExtractorWalker.java
Walker.java
package.html
soot.jimple.spark
SparkTransformer.java
Problems Search Console History Call Hierarchy
Members calling 'internalTransform(String, Map<String, String>)' - in workspace
internalTransform(String, Map<String, String>) : void - soot.jimple.spark.SparkTransformer
transform(String, Map<String, String>) : void - soot.SceneTransformer
soot.jimple.spark.SparkTransformer.internalTransform(String phaseName, Map<String, String> options) : void - soot/src

J. Softw. Maint. Evol.: Res. Pract. 2007; **19**:231–252
Published online in Wiley InterScience (www.interscience.wiley.com). DOI:

Research

Automatic construction of accurate application call graph with library call abstraction for Java

Weilei Zhang*, † and Barbara G. Ryder

Department of Computer Science, Rutgers University, 110 Freling

Alone?

Not Alone!

I'd like to ignore library code

whole-program analysis always pulls in the world for completeness. The problem is that the world is fairly large

what about callbacks?

ignore non-application program elements (e.g., system libraries)?

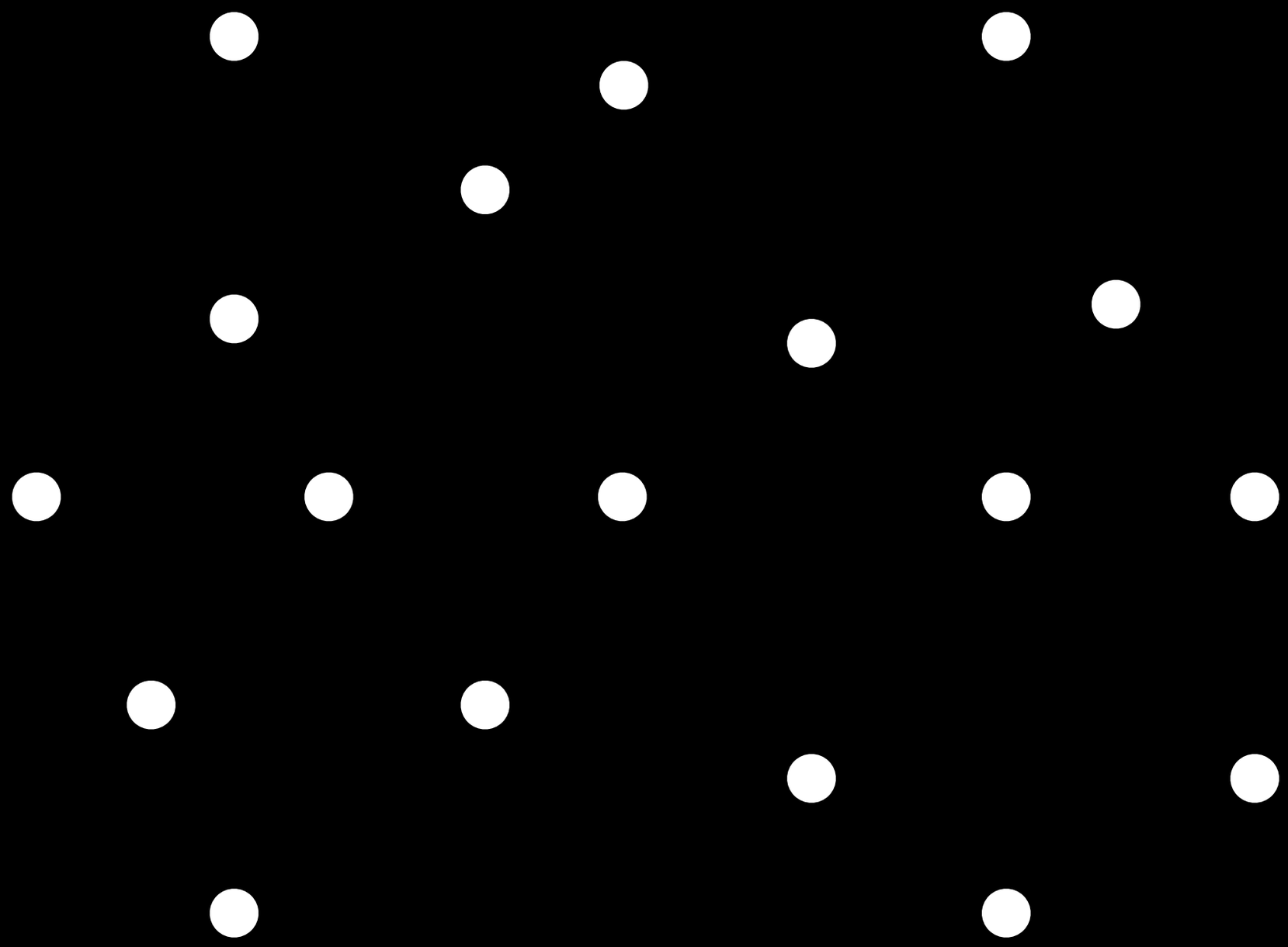
this would be unsound but better than nothing

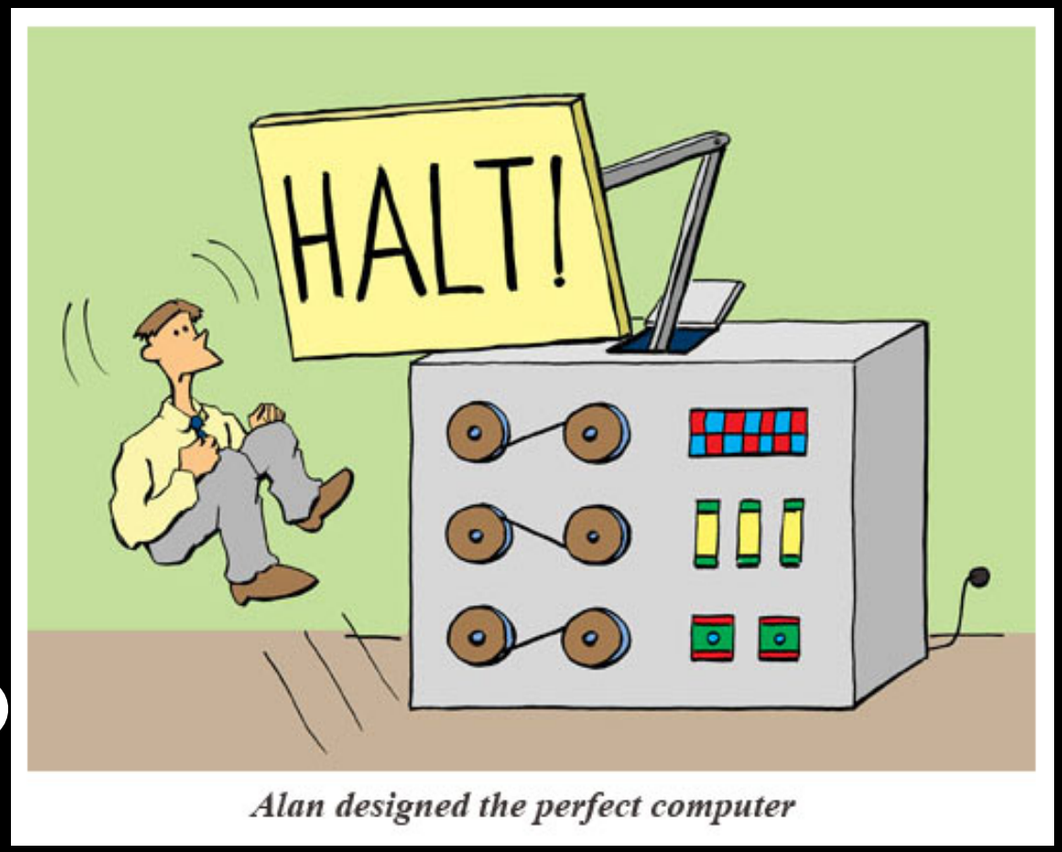
I am NOT interested in those

Partial-Program Analysis

Sound and Precise

Partial-Program Analysis

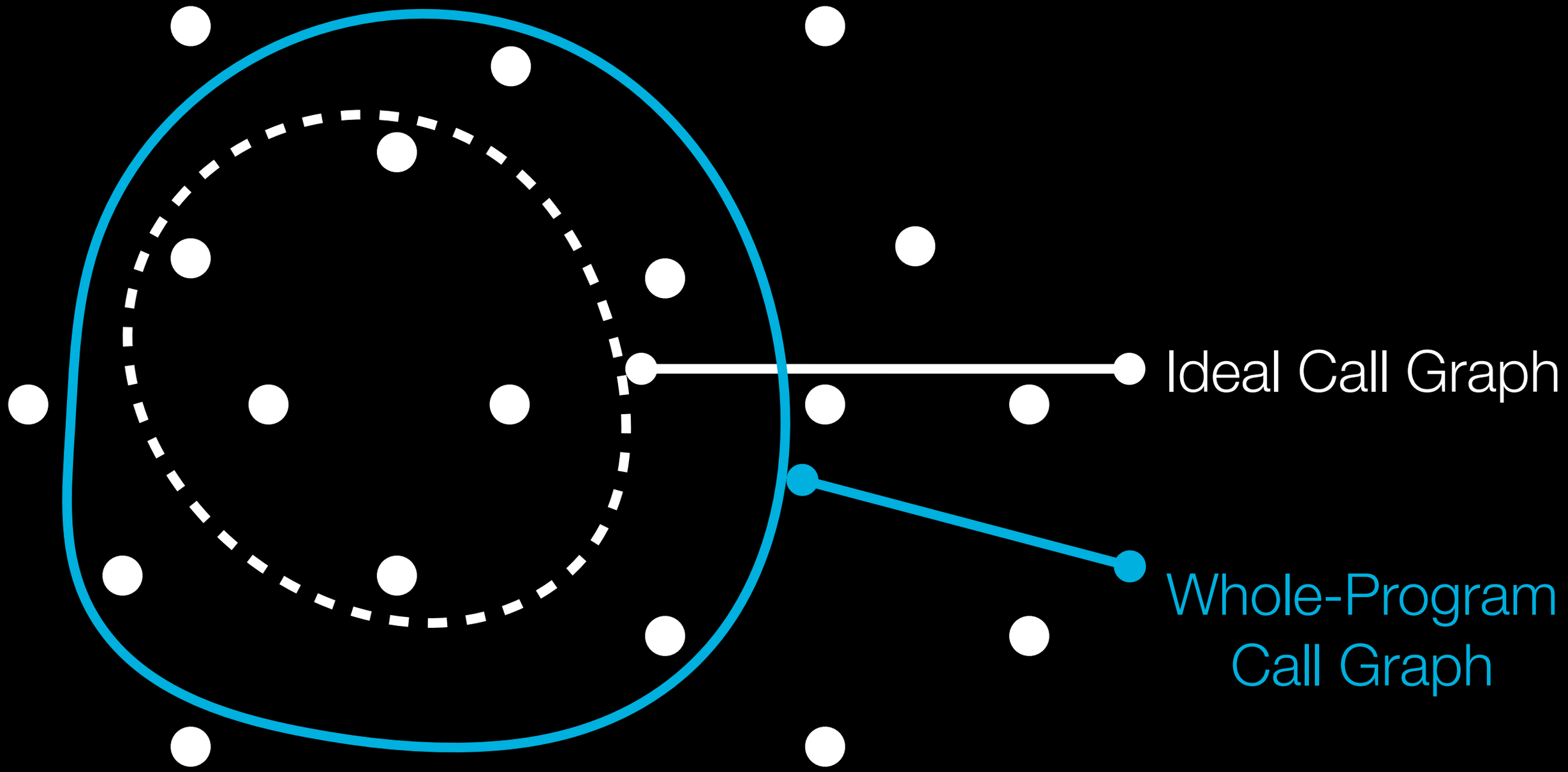


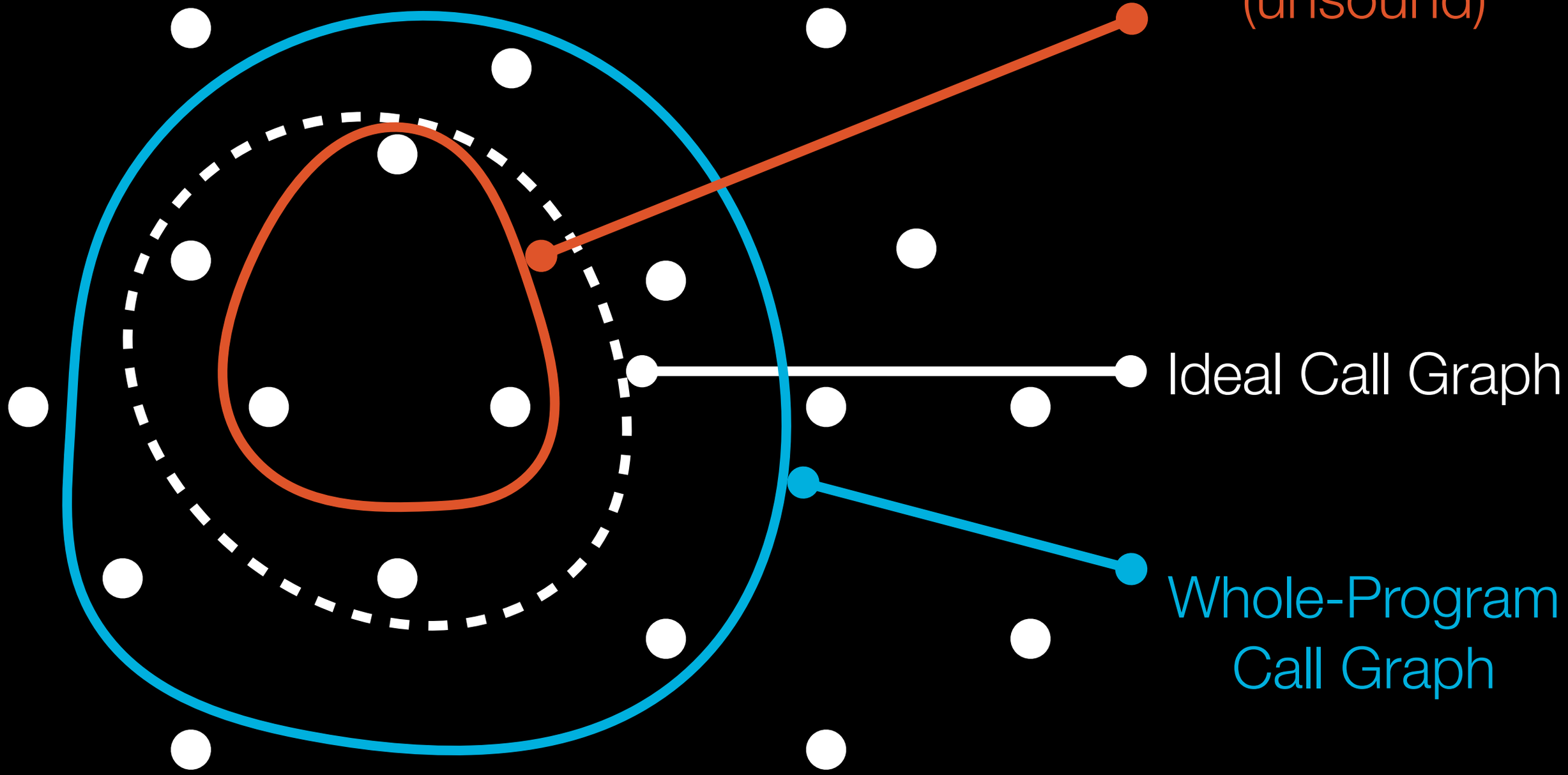


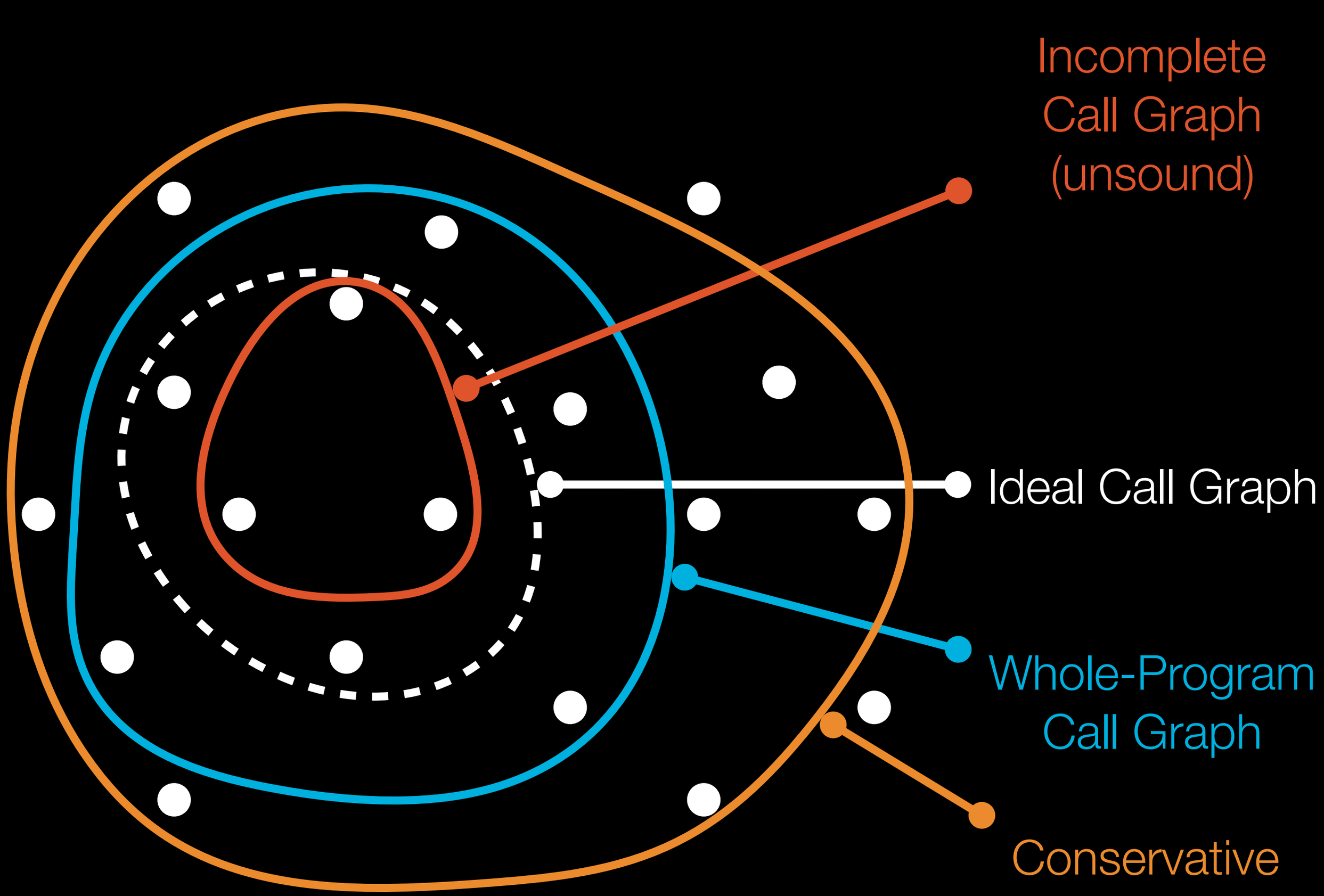
Alan designed the perfect computer

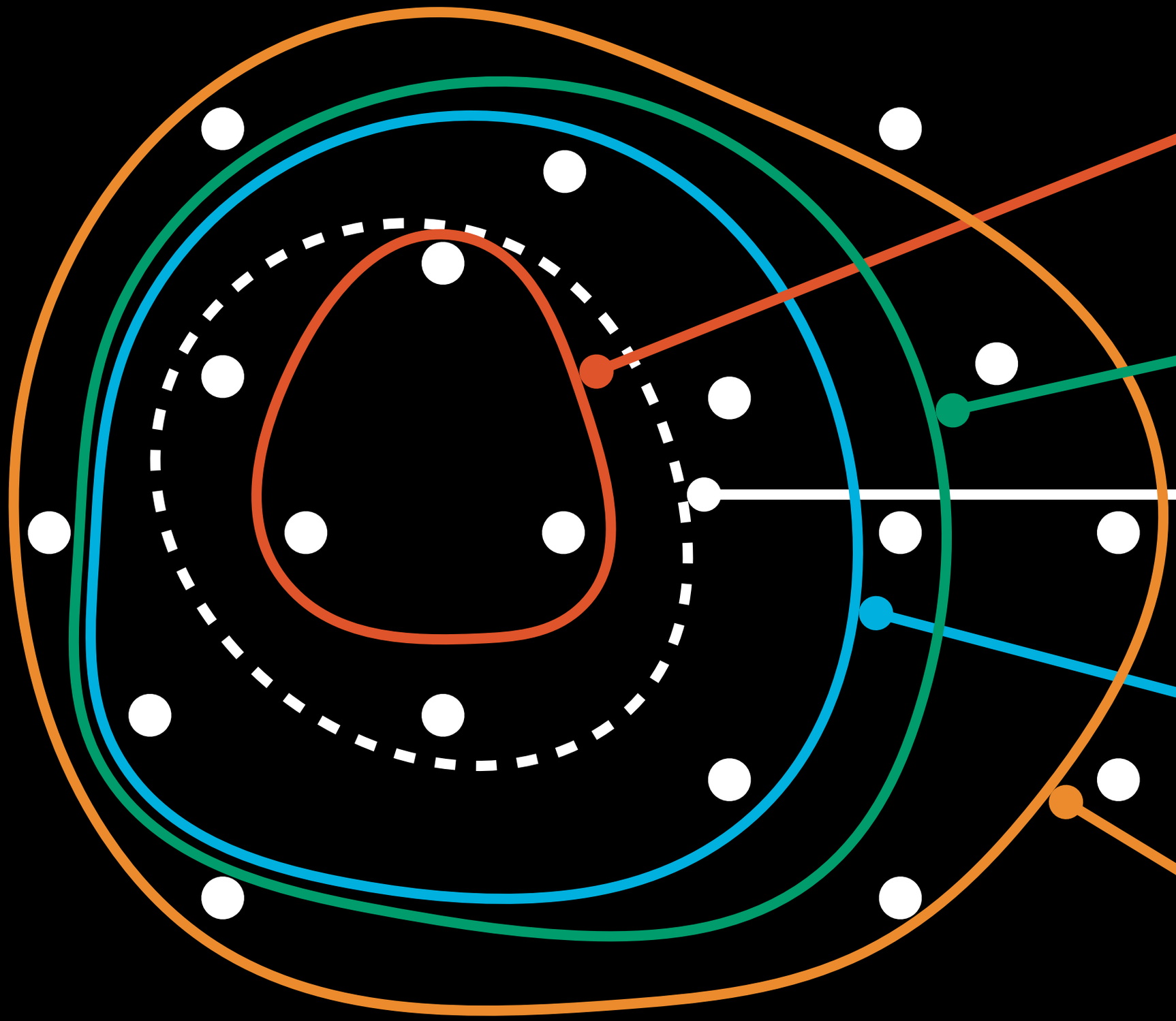
Ideal Call Graph

Image: CooperToons


Ideal Call Graph
Whole-Program
Call Graph


Incomplete
Call Graph
(unsound)
Ideal Call Graph
Whole-Program
Call Graph


Incomplete
Call Graph
(unsound)
Ideal Call Graph
Whole-Program
Call Graph
Conservative
Call Graph
(highly imprecise)



Incomplete
Call Graph
(unsound)

Partial-Program
Call Graph

Ideal Call Graph

Whole-Program
Call Graph

Conservative
Call Graph
(highly imprecise)

The Separate Compilation Assumption

Source: Ali and Lhoták. Application-Only Call Graph Construction. [ECOOP '12]

The Separate Compilation Assumption

All of the library classes can be compiled in the absence of the application classes.

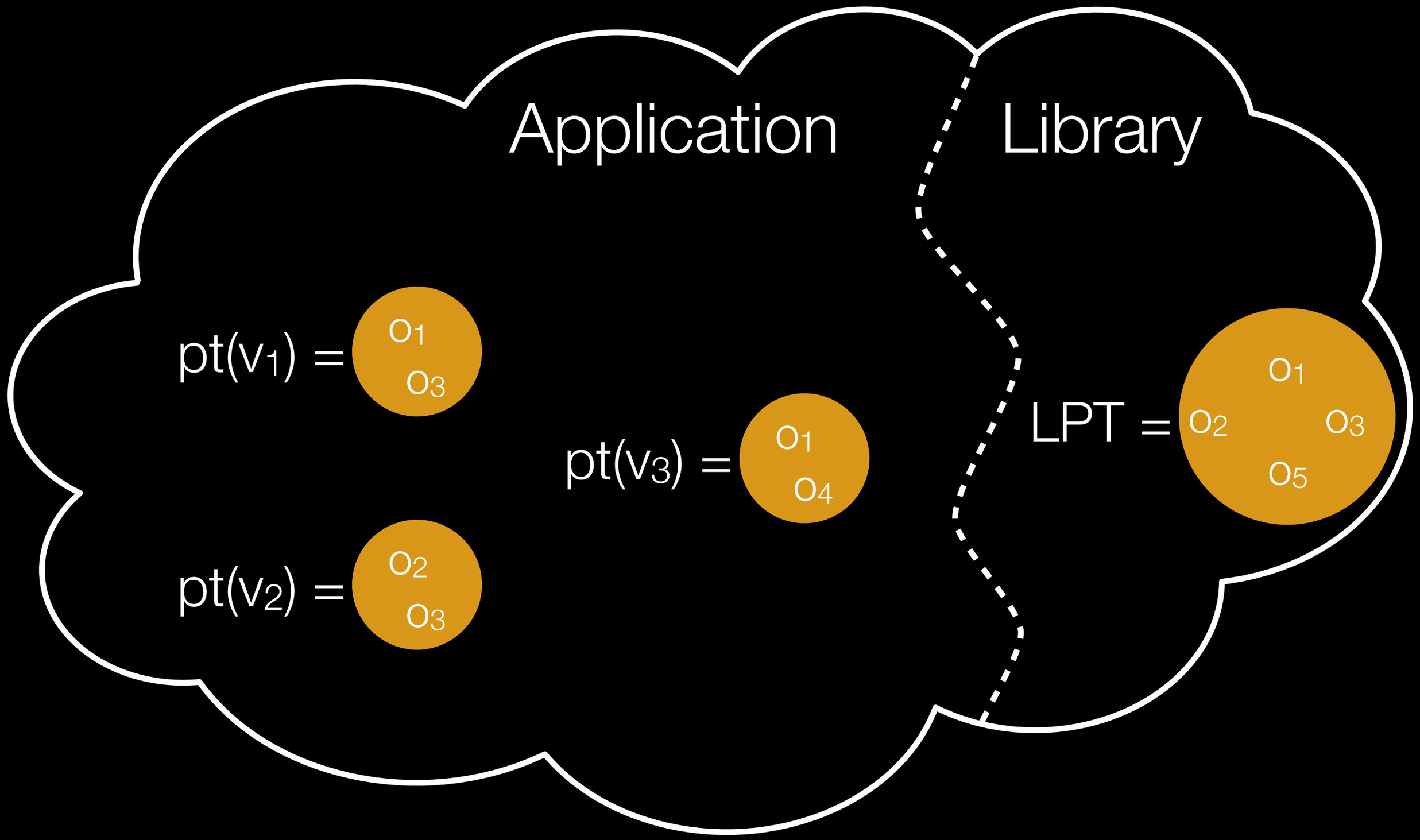
Constraints

1. Class Hierarchy
2. Class Instantiation
3. Local Variables
4. Method Calls
5. Field Access
6. Array Access
7. Static Initialization
8. Exception Handling

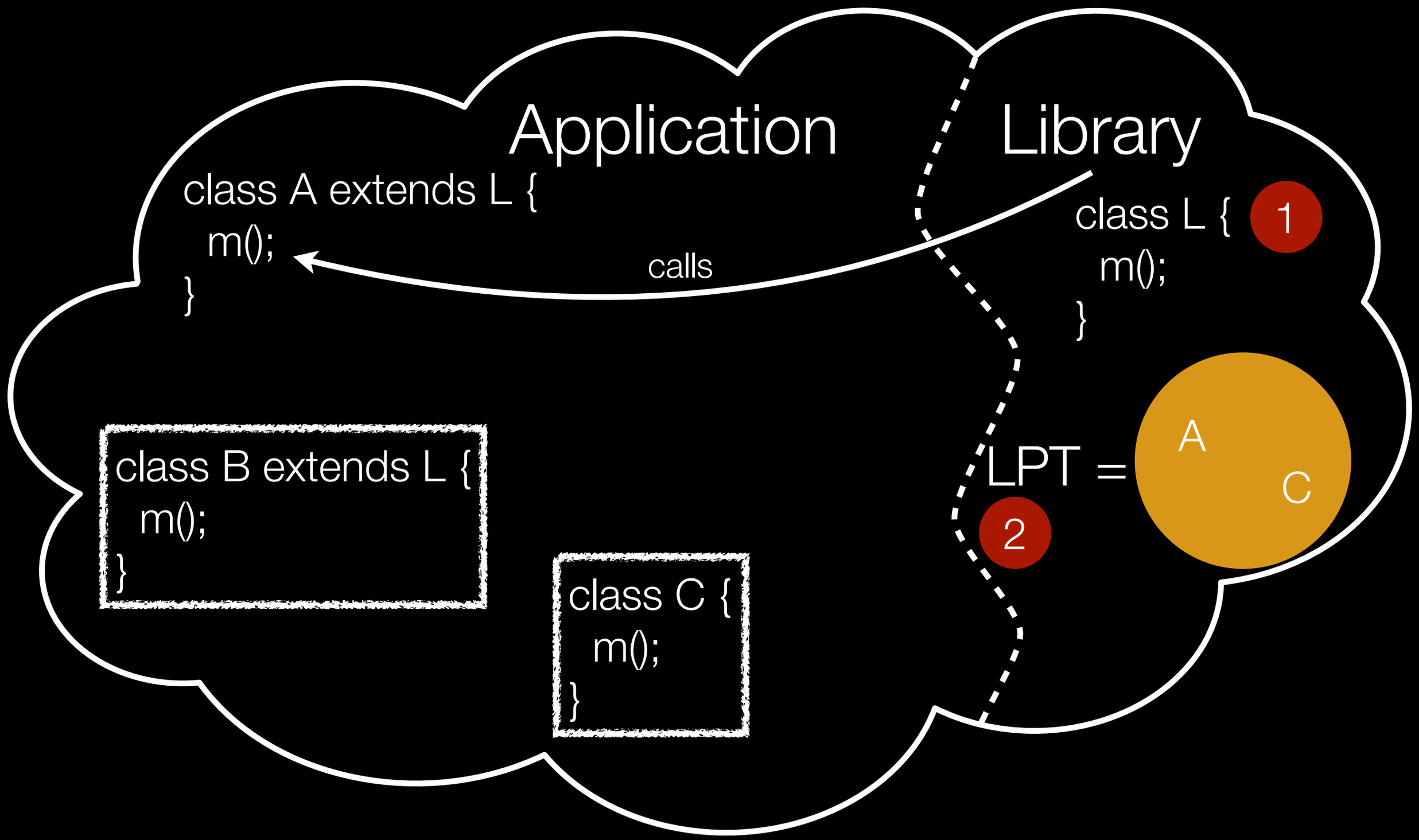
Constraints

1. Class Hierarchy
2. Class Instantiation
3. Local Variables
4. Method Calls
5. Field Access
6. Array Access
7. Static Initialization
8. Exception Handling

Library Points-to Set (LPT)



Library Callbacks



Source: Ali and Lhoták. Averroes: Whole-Program Analysis Without The Whole Program. [ECOOP '13]


JAR
JAR
TamiFlex
AVERROES
SCA
JAR
Placeholder
Library

Evaluation

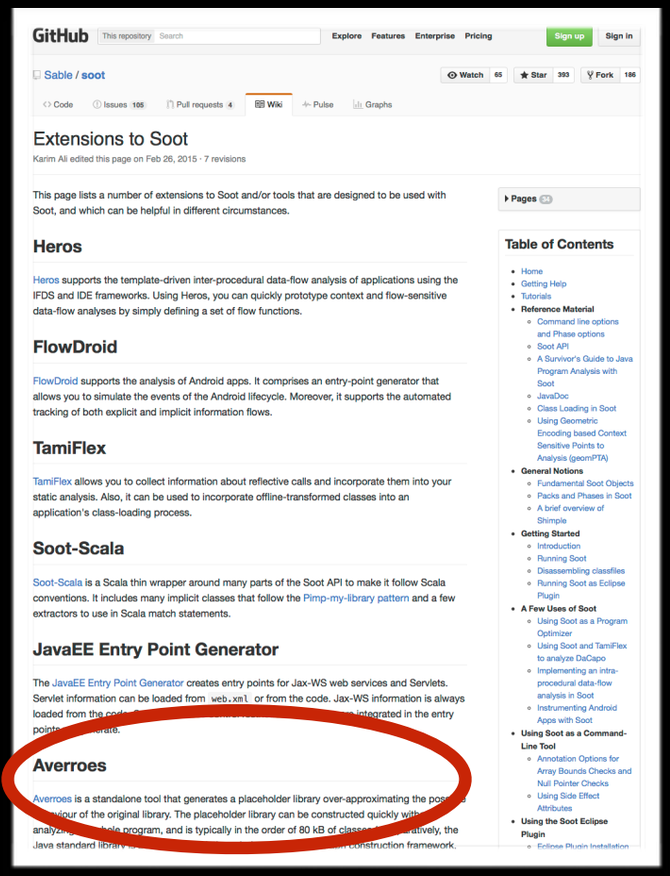
600× smaller library

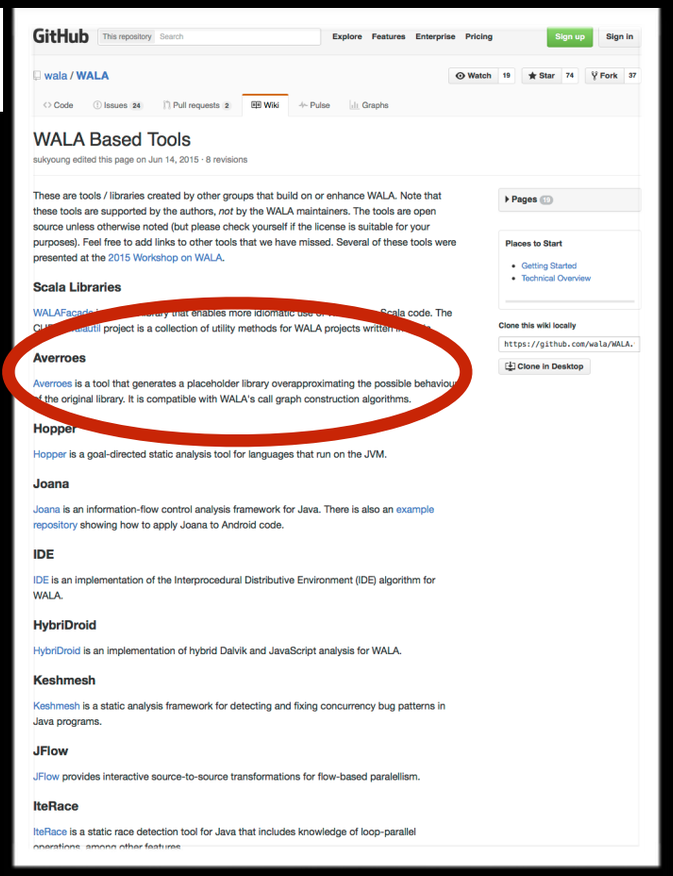
7× faster analysis

AVERROES

6× less memory

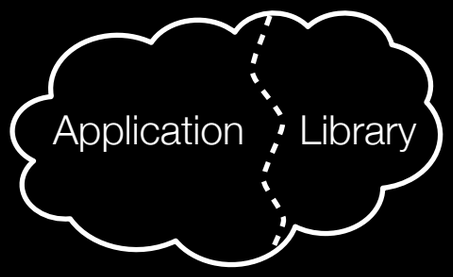
Precise & Sound



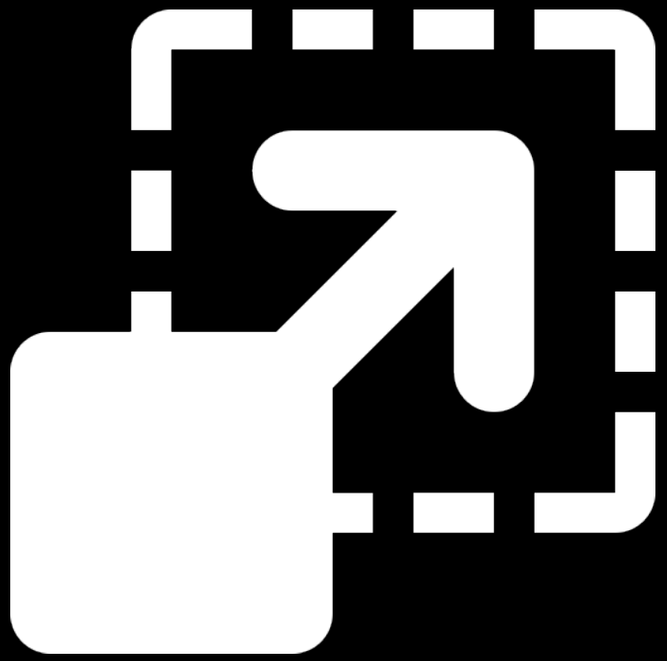


AVERROES


```
usage: doop [OPTION]... -- [BLOXBATCH OPTION]...
 -a,--analysis <name>                  The name of the analysis. Allowed values: 1-call-site-sensitive,
                                       1-call-site-sensitive+heap, 1-object-sensitive,
                                       1-object-sensitive+heap, 1-type-sensitive, 1-type-sensitive+heap,
                                       2-call-site-sensitive+2-heap, 2-call-site-sensitive+heap,
                                       2-object-sensitive+2-heap, 2-object-sensitive+heap,
                                       2-type-object-sensitive+2-heap, 2-type-object-sensitive+heap,
                                       2-type-sensitive+heap, 3-object-sensitive+3-heap,
                                       3-type-sensitive+2-heap, 3-type-sensitive+3-heap,
                                       context-insensitive, paddle-2-object-sensitive,
                                       paddle-2-object-sensitive+heap, ref-2-call-site-sensitive+2-heap,
                                       ref-2-call-site-sensitive+heap, ref-2-object-sensitive+heap,
                                       ref-2-type-sensitive+heap, ref-3-object-sensitive+2-heap,
                                       selective-2-object-sensitive+heap,
                                       selective-2-type-sensitive+heap, selective_A-1-object-sensitive,
                                       selective_B-1-object-sensitive, uniform-1-object-sensitive,
                                       uniform-2-object-sensitive+heap, uniform-2-type-sensitive+heap.
    --averroes                         Use averroes tool to create a placeholder library.
    --cache                            The analysis will use the cached facts, if they ex
    --client <FILE>                    Additional directory/file of client analysis to include.
    --context-sensitive-reflection
 -d,--dynamic <FILE>                   File with tab-separated data for Config:DynamicClass. Separate
                                       multiple files with a space.
```



AVERROES

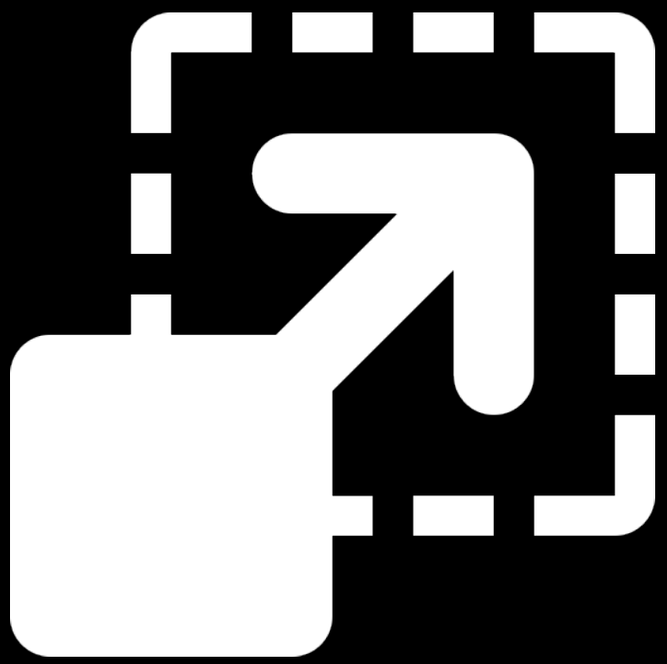


Scalability

Program Analysis in Practice

Precision

Program Analysis in Practice



Scalability

Precision

Security-Related Static Analyses

Security-Related Static Analyses

```
public void main(String[] args) {
   Object x = null;
   Object y = x;
   y.toString();
}
```

Null-Pointer Analysis

Security-Related Static Analyses

```
public void main(String[] args) {
   String x = args[0];
   String y = x;
   SQL.execute(''SELECT * FROM
User where userId='' + y );
}
```

Taint Analysis

Security-Related Static Analyses

```
public void main(String[] args) {
   File x = new File();
   File y = x;
   y.close();
}
```

Typestate Analysis

Static Data-Flow Analysis

Precise

Static Data-Flow Analysis

Precise Static Data-Flow Analysis

```
public void main(String[] args) {
  File x = new File();
  this.z = x;
  foo(x);
  x.close();
  foo(x);
}
```

```
public void foo(File y){
  y.write(...);
}

public void foo(){
  this.a.write(...);
}
```

Precise Static Data-Flow Analysis

```
public void main(String[] args) {
  File x = new File();
  this.z = x;
  foo(x);
  x.close();
  foo(x);
}
```

```
public void foo(File y){
  y.write(...);
}

public void foo(){
  this.a.write(...);
}
```

Context-Sensitive

Precise Static Data-Flow Analysis

```
public void main(String[] args) {
  File x = new File();
  this.z = x;
  foo(x);
  x.close();
  foo(x);
}
```

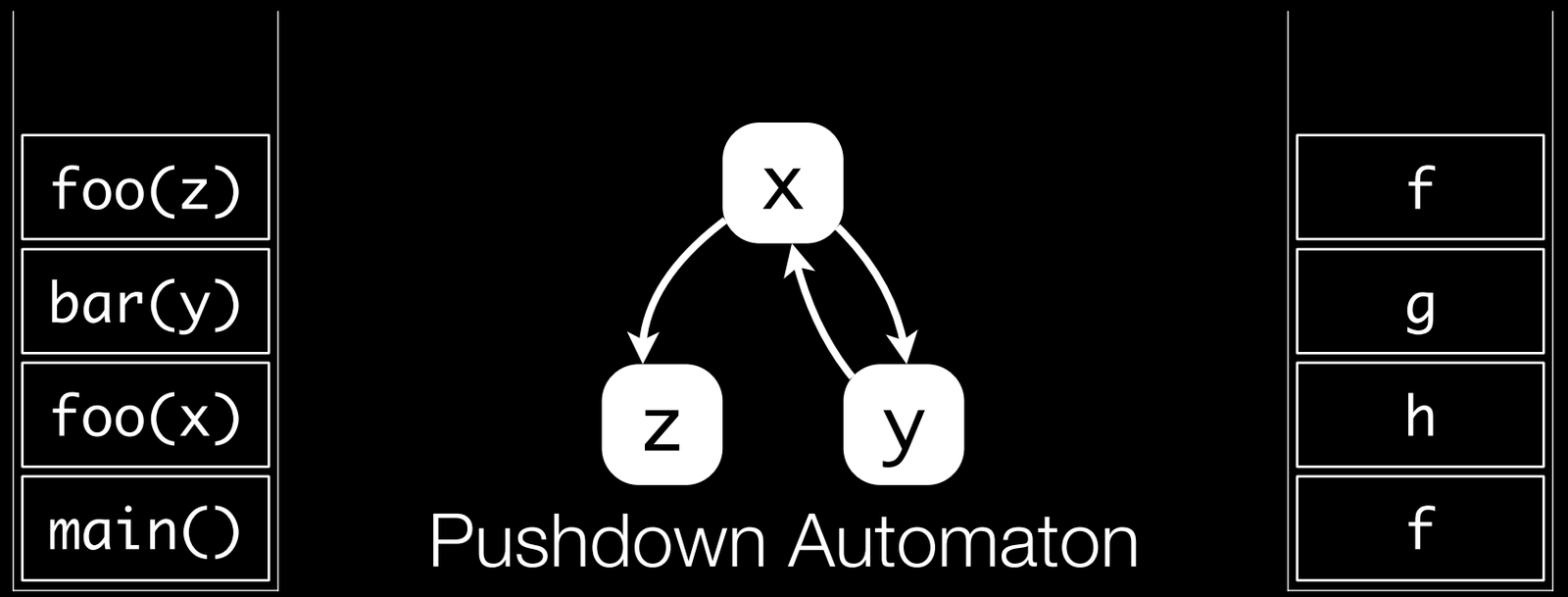
```
public void foo(File y){
  y.write(...);
}

public void foo(){
  this.a.write(...);
}
```

Field-Sensitive

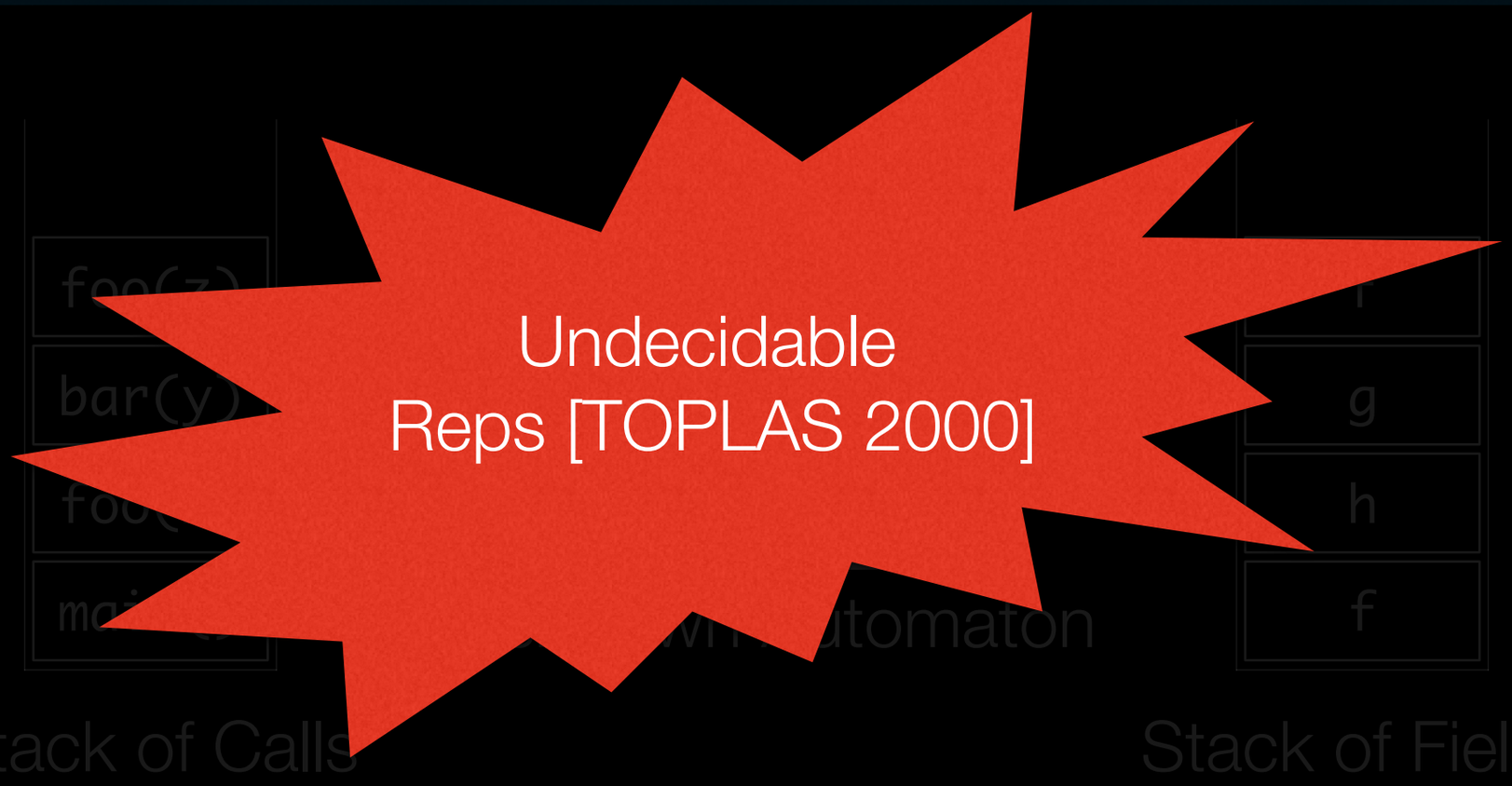
Precise Static Data-Flow Analysis

Context-Sensitive ∧ Field-Sensitive



Precise Static Data-Flow Analysis

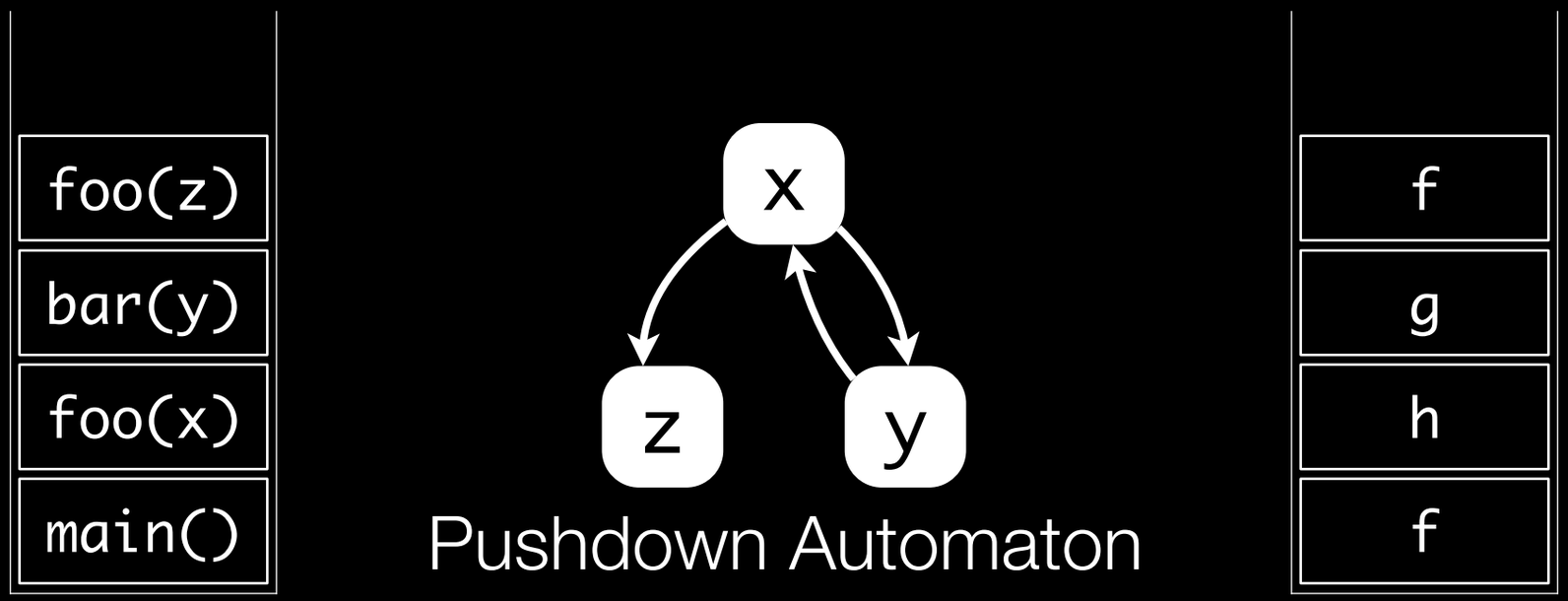
Context-Sensitive ∧ Field-Sensitive



Source: Thomas W. Reps. Undecidability of Context-Sensitive Data-Dependence Analysis. [TOPLAS '00]

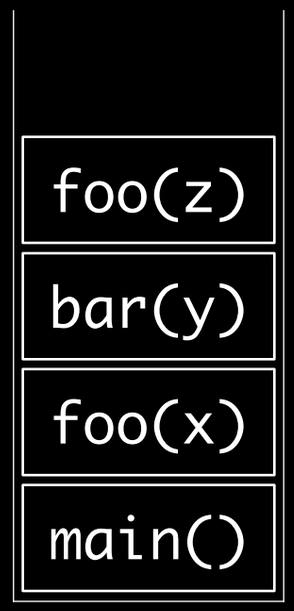
Precise Static Data-Flow Analysis

Context-Sensitive ∧ Field-Sensitive

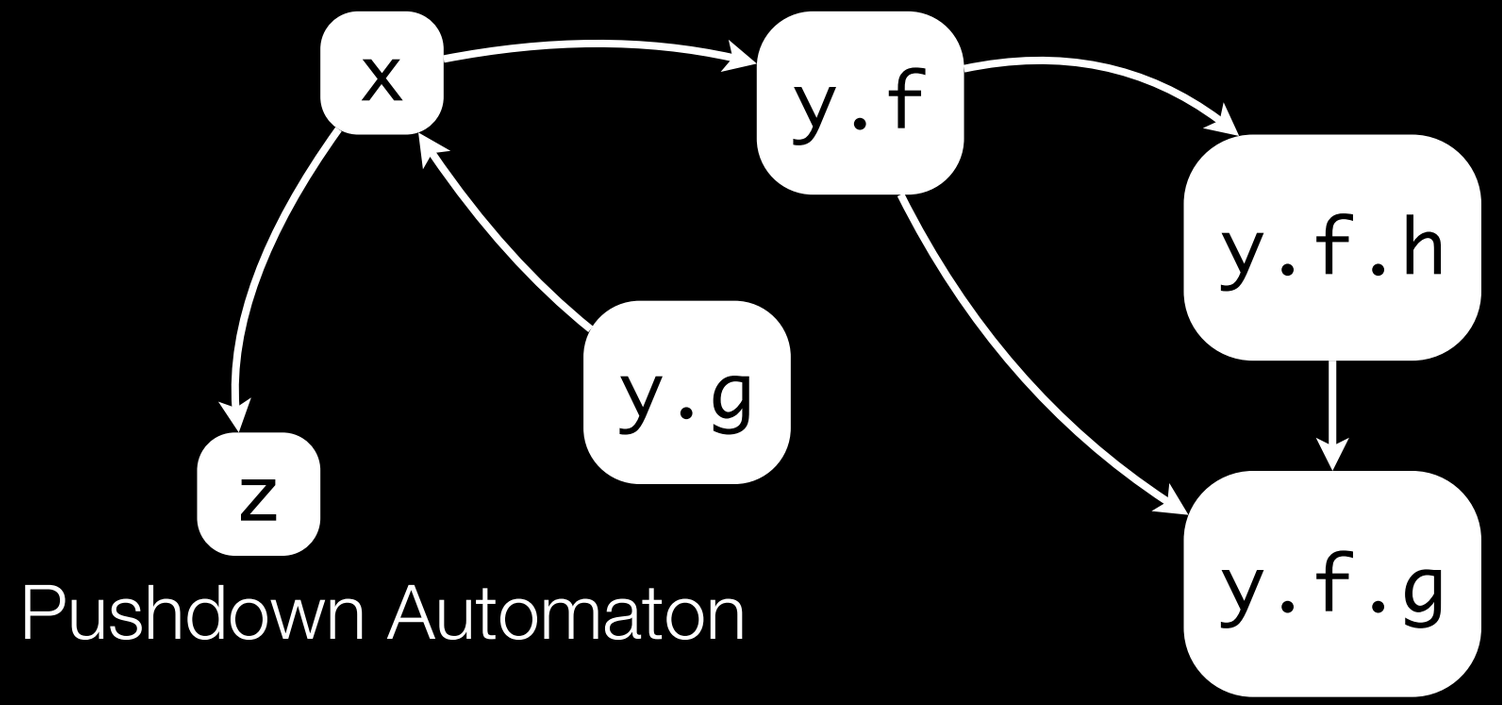


Precise Static Data-Flow Analysis

Context-Sensitive ∧ Field-Sensitive



Stack of Calls



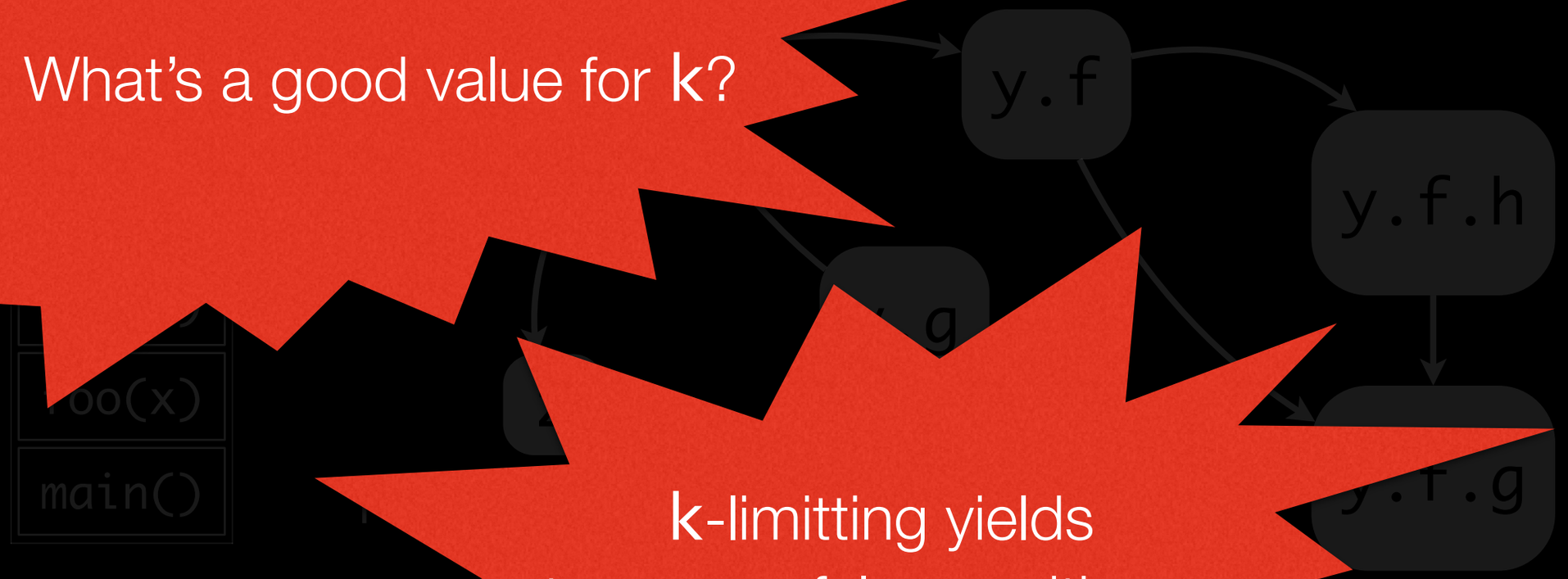
Pushdown Automaton

k-limitting
Access Paths/Graphs

Precise Static Data-Flow Analysis

Context-Sensitive ∧ Field-Sensitive

What's a good value for **k**?



k-limitting yields
too many false positives

Synchronized Pushdown Systems (SPDS)

Source: Späeth et al. Context-, Flow-, and Field-Sensitive Data-Flow Analysis using Synchronized Pushdown Systems. [POPL '19]

Synchronized Pushdown Systems

Context-Sensitive ∧ Field-Sensitive

Synchronized Pushdown Systems

Context-Sensitive ∧ Field-Sensitive

over-approximation

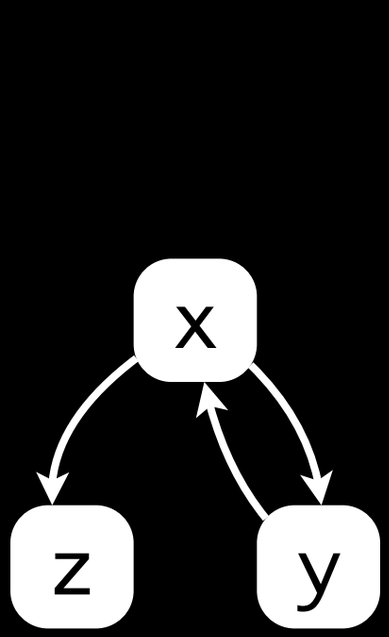
Never encountered
in practice

Context-Sensitive ∧ Field-Sensitive

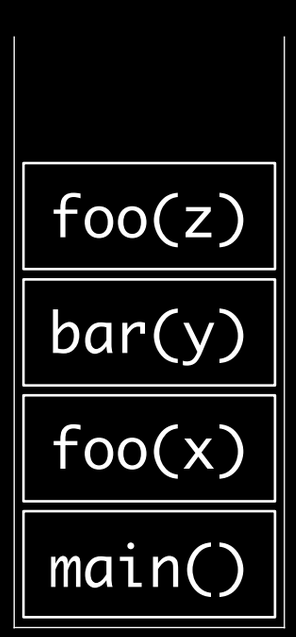
Synchronized Pushdown Systems

Context-Sensitive ∧ Field-Sensitive

Pushdown System of Calls

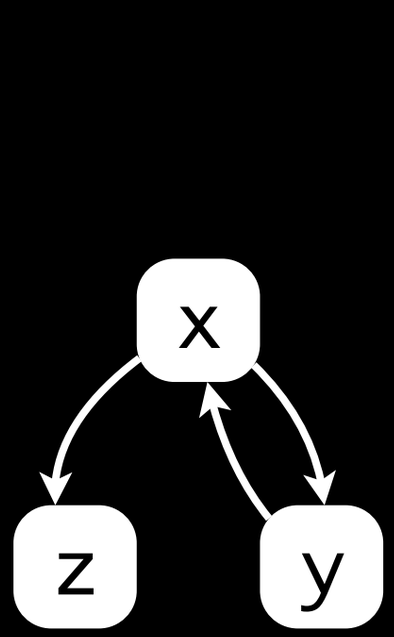


Variables

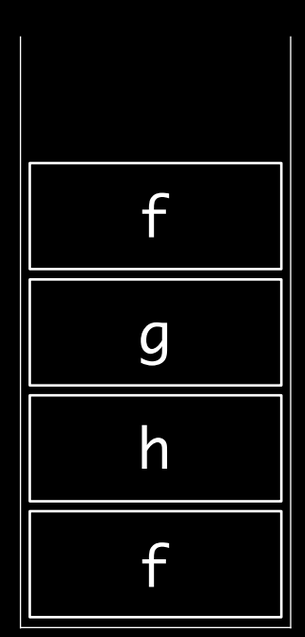


Stack of Calls

Pushdown System of Fields



Variables



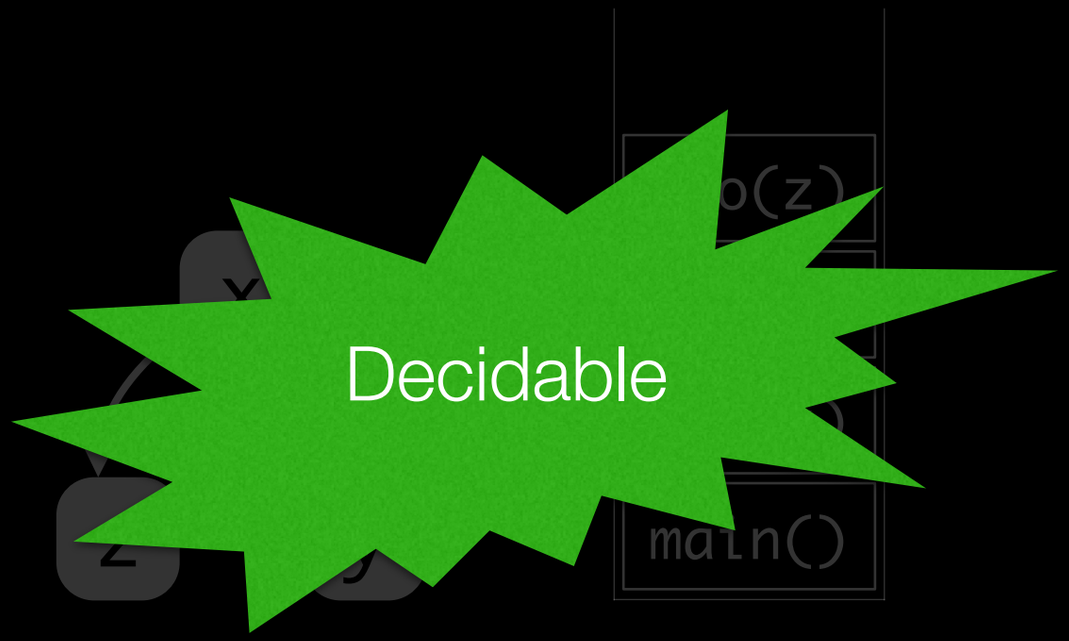
Stack of Fields

Synchronized Pushdown Systems

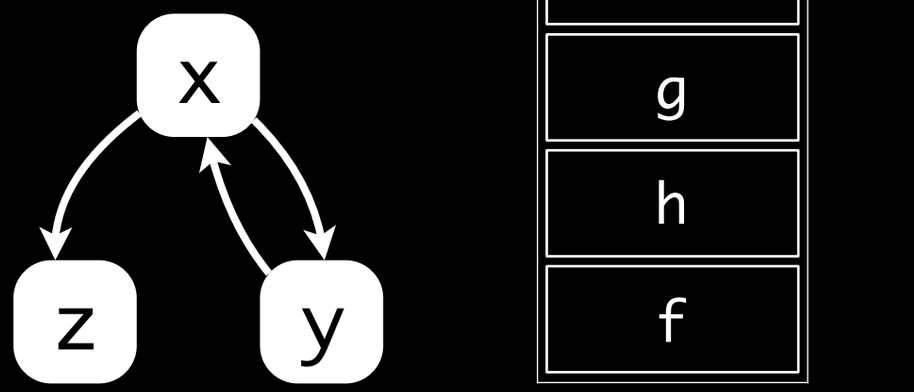
Context-Sensitive ∧ Field-Sensitive

Pushdown System of Calls

Pushdown System of Fields



Variables Stack of Calls



Variables Stack of Fields

Synchronized Pushdown Systems

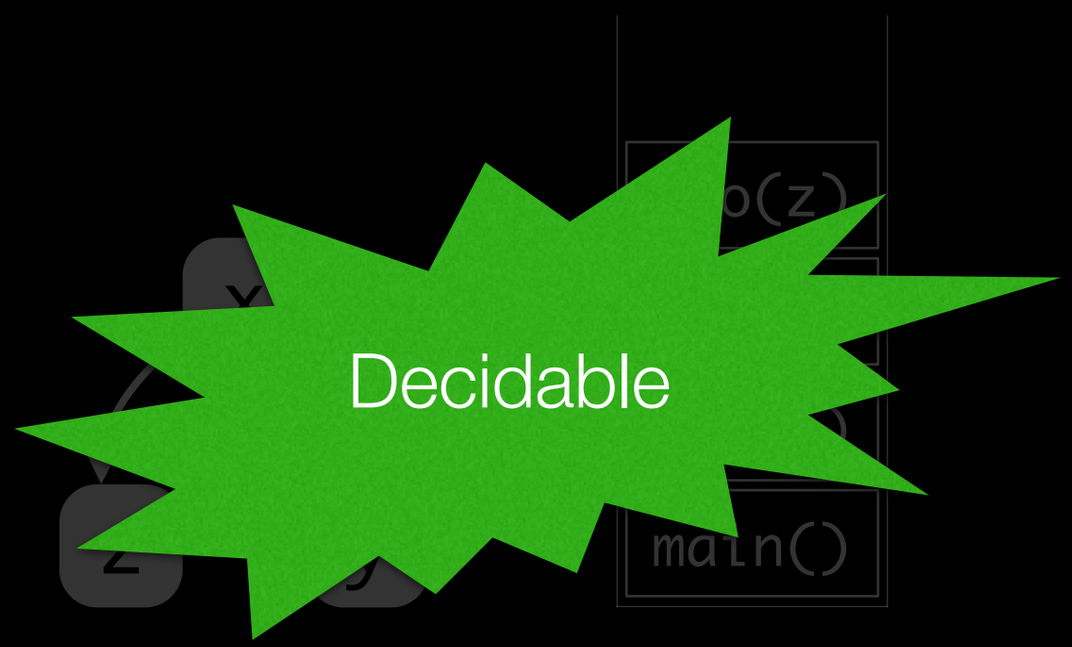
Context-Sensitive

Pushdown System of Calls

∧

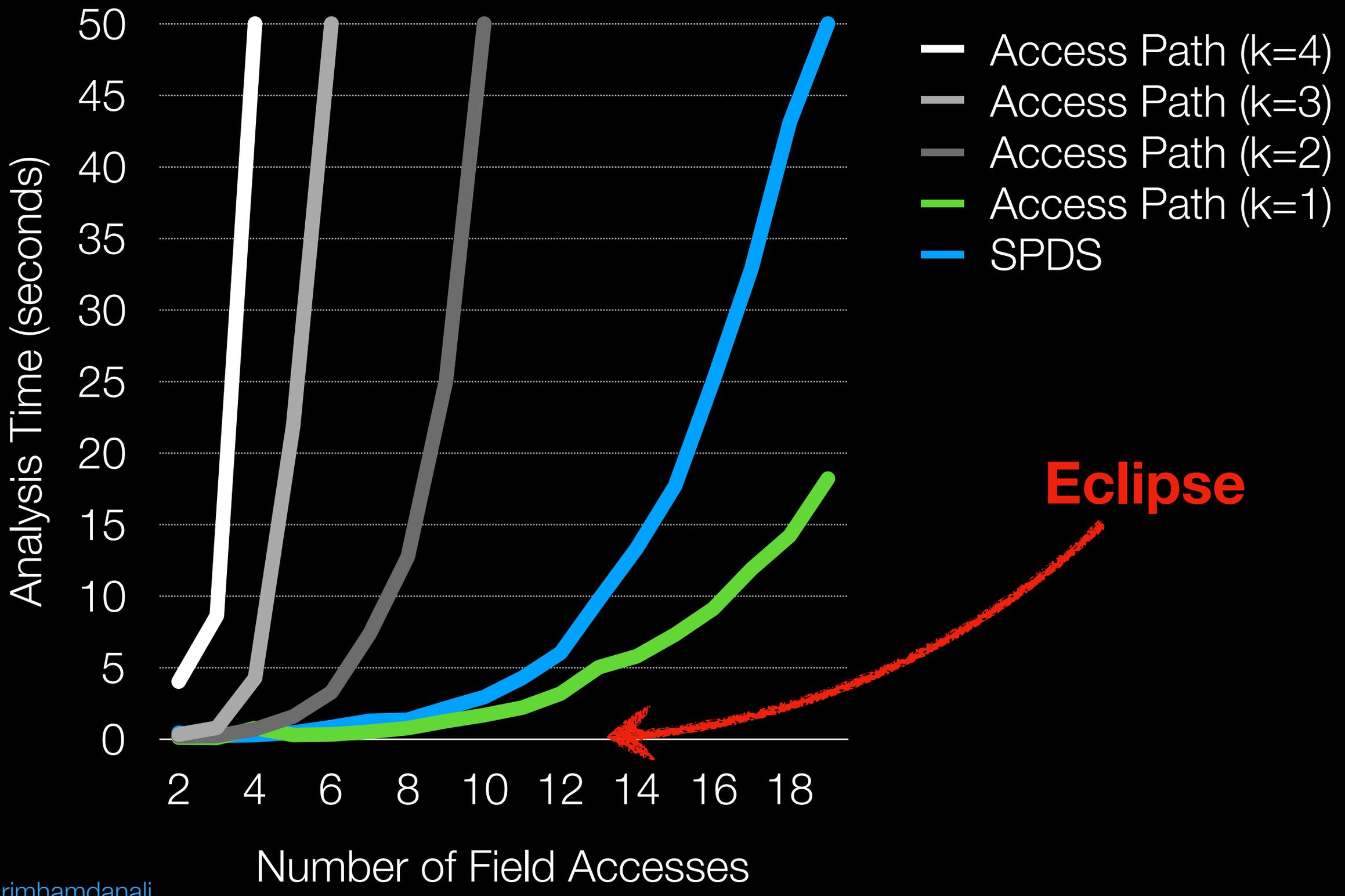
Field-Sensitive

Pushdown System of Fields



SPDS Evaluation

SPDS Evaluation



… but is it useful in practice?

```java
public class MessageEncryptor {
    public String ALGORITHM = "AES";

    public void encrypt(String data, Key key) throws GeneralSecurityException {

    }
}
```

CogniCrypt.org

Eclipse Foundation

68% are insecure

(Maven has > 2.7 million artifacts)

95% are insecure

(10,000 most recent Android apps on AndroZoo)

Security Advisory ID SYMSA1460

Initial Publication Date: 29 Aug 2018
Advisory Status: Closed
Advisory Severity: Medium
CVSS Base Score: 5.6

Summary

Symantec has released an update to address an issue that was discovered in the Norton Identity Safe for Android product.

Affected Products

Norton Identity Safe for Android		
CVE	**Affected Version(s)**	**Remediation**
CVE-2018-12240	Prior to 5.3.0.976	Upgrade to 5.3.0.976

Issues

CVE-2018-12240	
Severity/CVSSv3:	Medium / 5.6 AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L
References:	Security Focus: BID 105146 / NVD: CVE-2018-12240
Impact:	Privilege escalation
Description:	The Norton Identity Safe product may be susceptible to a privilege escalation issue via a hard coded IV, which is a type of vulnerability that can potentially increase the likelihood of encrypted data being recovered without adequate credentials.

Symantec CVE-2018-12240

SPDS

CogniCrypt

Precision

Program Analysis in Practice

Usability

SCA Analysis Results – With TeamMentor

Filter Set: Security Auditor View My Issues

4286 | 2335 | 8 | 1334 | 7963

Critical (4286)

Group By: Category

- Command Injection – [0 / 3]
  - Exec.java:107 (Command Injection)
  - Exec.java:289 (Command Injection)
  - WSDLScanning.java:148 (Command Injection)
- Cross-Site Scripting: DOM – [0 / 2]
- Cross-Site Scripting: Persistent – [0 / 3227]
- Cross-Site Scripting: Reflected – [0 / 822]
- Password Management: Hardcoded Password – [0 / 19
- Path Manipulation – [0 / 6]
- Privacy Violation – [0 / 148]
- Race Condition: Singleton Member Field – [0 / 8]
- SQL Injection – [0 / 49]
- XPath Injection – [0 / 2]

Advanced...

Analysis Evidence

- ParameterParser.java:615 – getParameterValues(return)
- ParameterParser.java:615 – Assignment to values
- ParameterParser.java:623 – Return values[0]
- ParameterParser.java:597 – getRawParameter(return)
- ParameterParser.java:597 – Return
- Challenge2Screen.java:641 – getRawParameter(return)
- Challenge2Screen.java:641 – Assignment to protocol

Project Summary | Exec.java

```
        ByteArrayOutputStream output = new ByteArrayOutputStream();
        ByteArrayOutputStream errors = new ByteArrayOutputStream();
        ExecResults results = new ExecResults(Arrays.asList(command).toStr
        BitSet interrupted = new BitSet(1);
        boolean lazyQuit = false;
        ThreadWatcher watcher;

        try
        {
            // start the command
            child = Runtime.getRuntime().exec(command);

            // get the streams in and out of the command
            InputStream processIn = child.getInputStream();
            InputStream processError = child.getErrorStream();
            OutputStream processOut = child.getOutputStream();

            // start the clock running
            if (timeout > 0)
            {
                watcher = new ThreadWatcher(child, interrupted, timeout);
                new Thread(watcher).start();
            }

            // Write to the child process' input stream
            if ((input != null) && !input.equals(""))
            {
                try
                {
                    processOut.write(input.getBytes());
                    processOut.flush();
```

Issue | Issue | Recom | History | Diagra | Correla | Screen

Issue: Exec.java:107 (Command Injection)

User:

Analysis:

Command Injection (Input Validation and

The method execOptions() in Exec.java calls exec() with a command built from untrusted data. This call can cause the

SCA Analysis Results – With TeamMentor

Filter Set: Security Auditor View

My Issues

4286 | 2335 | 8 | 1334 | 7963

Critical (4286)

Group By: Category

- Command Injection – [0 / 3]
  - Exec.java:107 (Command Injection)
  - Exec.java:289 (Command Injection)
  - WSDLScanning.java:148 (Command Injection)
- Cross-Site Scripting: DOM – [0 / 2]
- Cross-Site Scripting: Persistent – [0 / 3227]
- Cross-Site Scripting: Reflected – [0 / 822]
- Password Management: Hardcoded Password – [0 / 19
- Path Manipulation – [0 / 6]
- Privacy Violation – [0 / 148]
- Race Condition: Singleton Member Field – [0 / 8]
- SQL Injection – [0 / 49]
- XPath Injection – [0 / 2]

Project Summary

```
            ByteArrayOu
            ByteArrayOu
            ExecResults
            BitSet inte
            boolean lazy
            ThreadWatche

            try
            {
                // star
                child =

                // get
                InputSt
                InputSt
                OutputS

                // star
                if (time
                {
                    wate
                    new
                }
```

precise

responsive

seamless

tailored

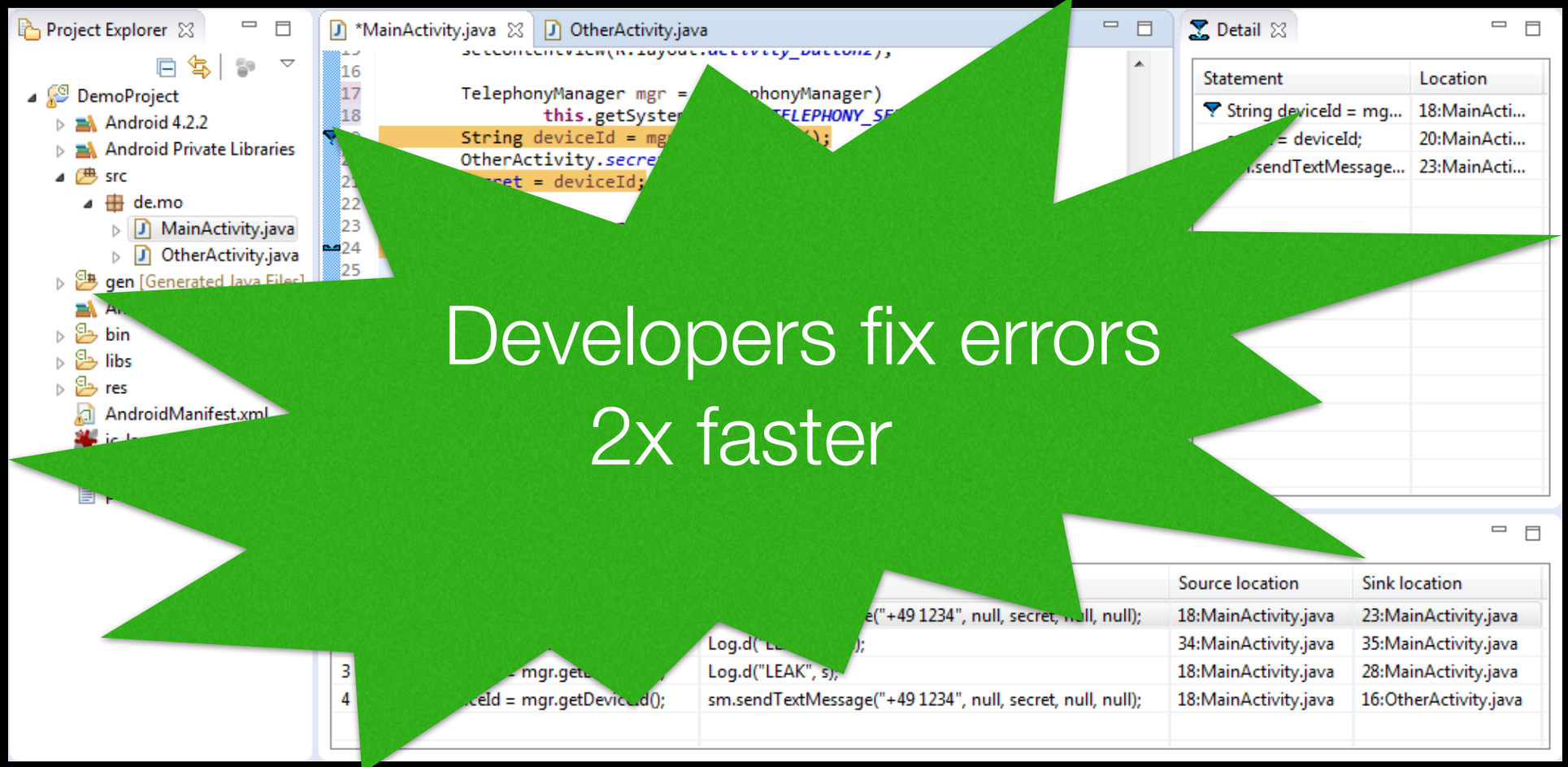
Sources: Smith et al. Questions Developers Ask While Diagnosing Potential Security Vulnerabilities with Static Analysis. [FSE '15]

Xiao et al. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. [CSCW '14]

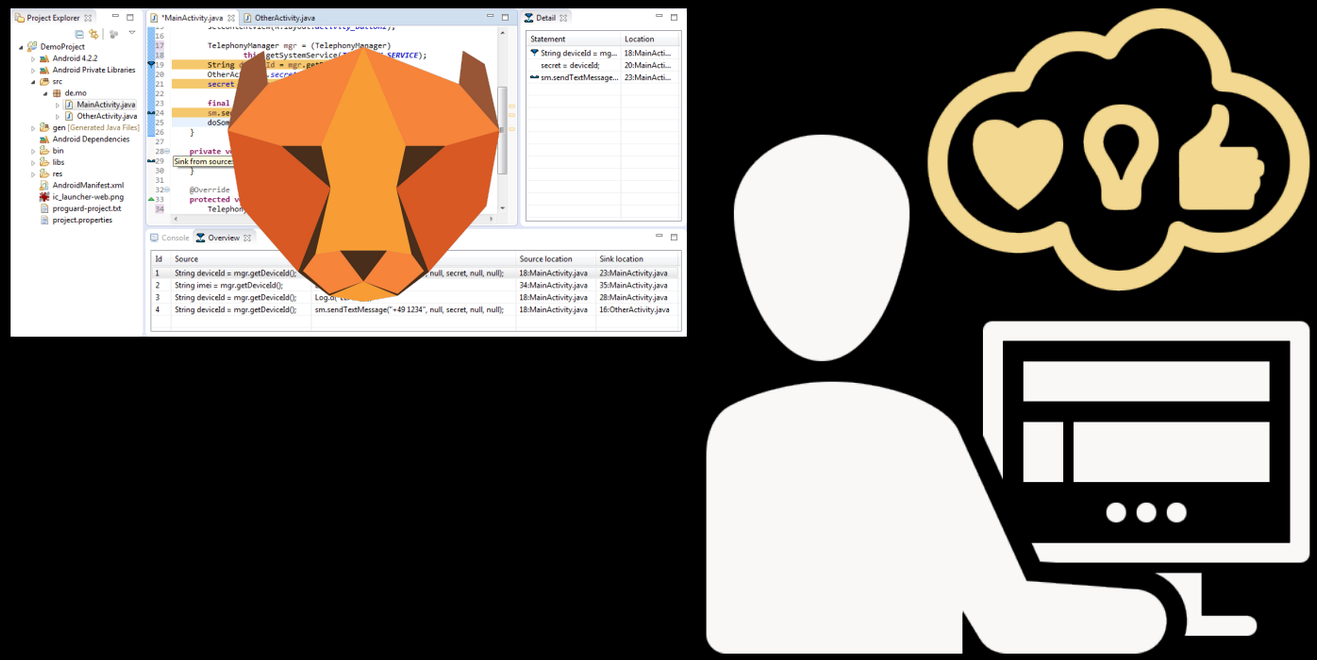
Johnson et al. Why Don’t Software Developers Use Static Analysis Tools to Find Bugs? [ICSE '13]

Just-In-Time Static Analysis

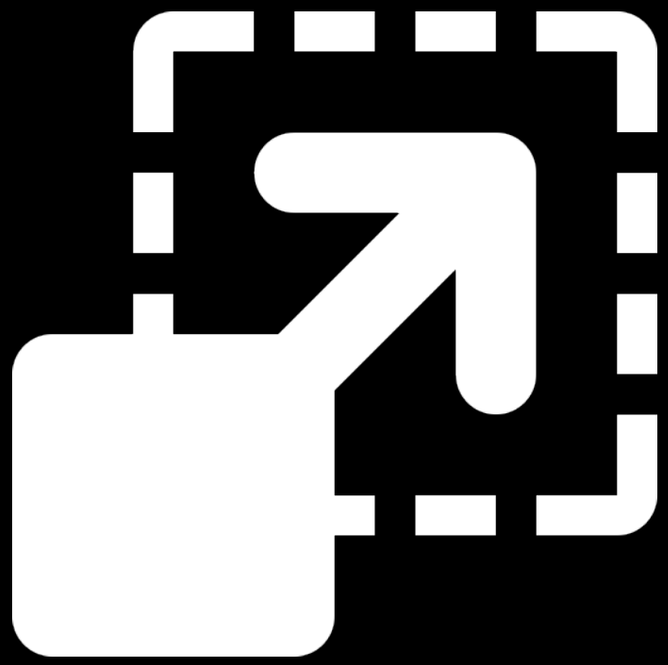
Just-In-Time Static Analysis (Cheetah)



https://github.com/secure-software-engineering/cheetah


Project Explorer
DemoProject
Android 4.2.2
Android Private Libraries
src
de.mo
MainActivity.java
OtherActivity.java
gen [Generated Java Files]
Android Dependencies
bin
libs
res
AndroidManifest.xml
ic_launcher-web.png
proguard-project.txt
project.properties
*MainActivity.java
OtherActivity.java
Sink from source
Detail
Statement
Location
Console
Overview
Id
Source
Source location
Sink location
1 String deviceId = mgr.getDeviceId(); 18:MainActivity.java 23:MainActivity.java
2 String imei = mgr.getDeviceId(); 34:MainActivity.java 35:MainActivity.java
3 String deviceId = mgr.getDeviceId(); 18:MainActivity.java 28:MainActivity.java
4 String deviceId = mgr.getDeviceId(); sm.sendTextMessage("+49 1234", null, secret, null, null); 18:MainActivity.java 16:OtherActivity.java

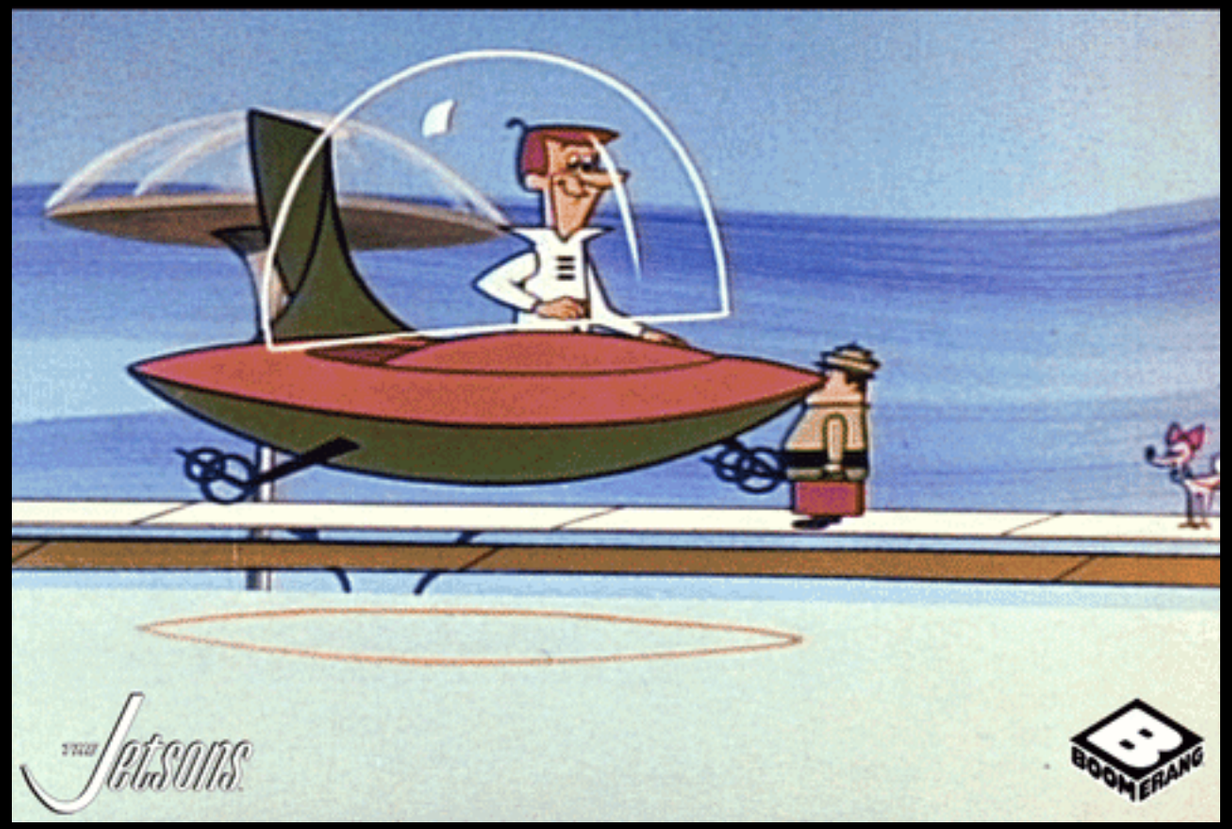
Usability



Scalability

Precision

Usability



Where do we go from here?

Image: Boomerang Toons / GIPHY

S W A N

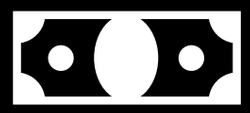
Swift Analysis Framework

themaplelab/swan

Discriminants

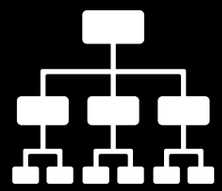
Call Frequency

Method Size



Budget

Method Size



Algorithm

Nested Knapsack

Search Space

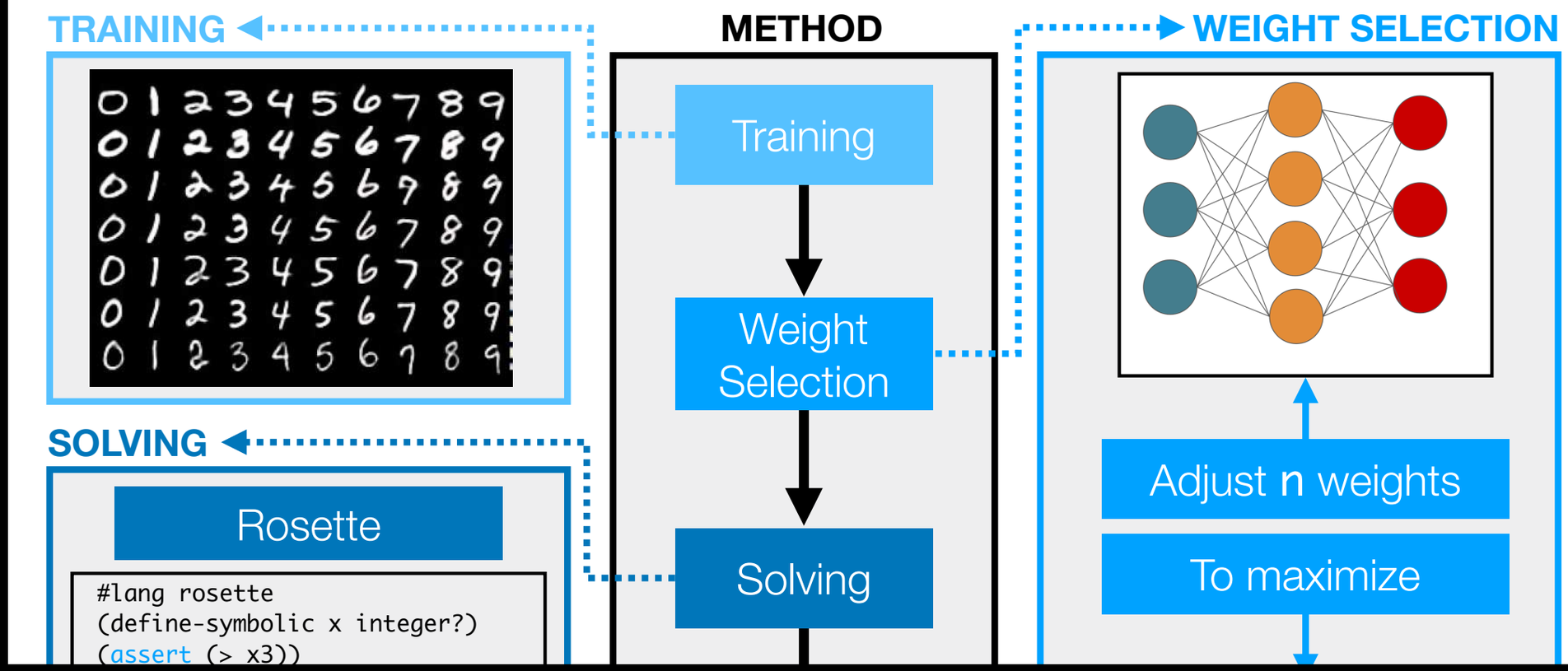
All IDT Methods

Post-Inlining Transformations

Estimate Post-Inlining Transformations

themaplelab/openj9

- Understanding the internals of neural networks is limited due to their complexity
- Fixing errors in neural networks without retraining is hard and currently not supported
- We use Rosette to solve for changes in weights to a neural network
- Rosette is able to represent neural networks and their results as symbolic values, which can then be solved for, under the assertion that a given data point is correct



Fixing Neural Networks using Solver-Aided Languages

coming soon...

Android Security Improvement update: Helping developers harden their apps, one thwarted vulnerability at a time

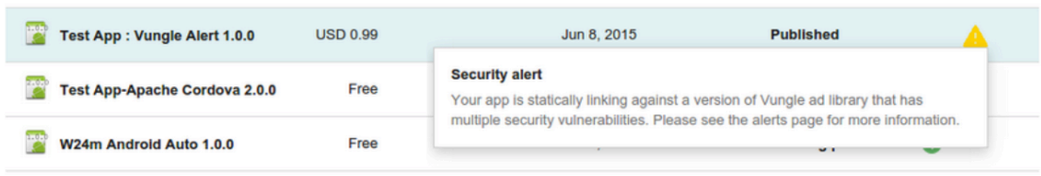
28 February 2019

Posted by Patrick Mutchler and Meghan Kelly, Android Security & Privacy Team

Helping Android app developers build secure apps, free of known vulnerabilities, means helping the overall ecosystem thrive. This is why we launched the Application Security Improvement Program five years ago, and why we're still so invested in its success today.

What the Android Security Improvement Program does

When an app is submitted to the Google Play store, we scan it to determine if a variety of vulnerabilities are present. If we find something concerning, we flag it to the developer and then help them to remedy the situation.



Think of it like a routine physical. If there are no problems, the app runs through our normal tests and continues on the process to being published in the Play Store. If there is a problem, however, we provide a diagnosis and next steps to get back to healthy form.

Over its lifetime, the program has helped more than 300,000 developers to fix more than 1,000,000 apps on Google Play. In 2018 alone, the program helped over 30,000 developers fix over 75,000 apps. The downstream effect means that those 75,000 vulnerable apps are not distributed to users with the same security issues present, which we consider a win.

Google

Android Mobile Security

Image: Android Developers Blog

DOI:10.1145/3338112

Key lessons for designing static analyses tools deployed to find bugs in hundreds of millions of lines of code.

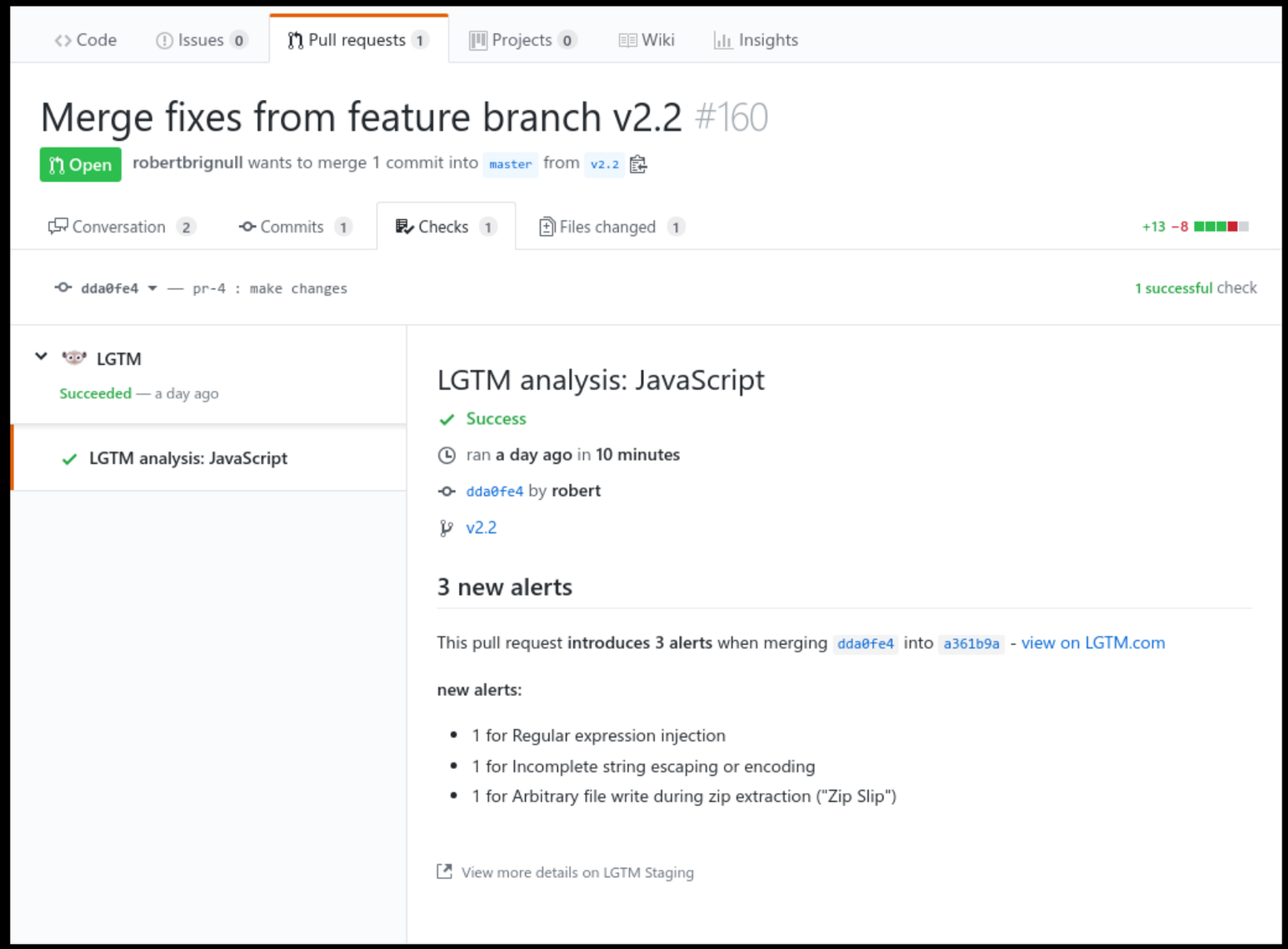
BY DINO DISTEFANO, MANUEL FÄHNDRICH, FRANCESCO LOGOZZO, AND PETER W. O'HEARN

Scaling Static Analyses at Facebook

Facebook

Infer, Zoncolan, SapFix

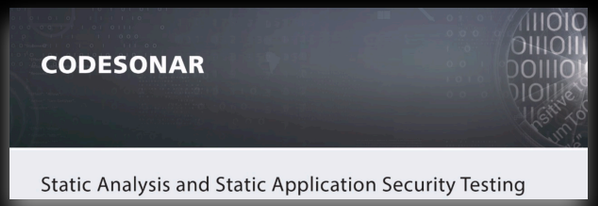
Source: Distefano et al. Scaling Satic Analsyes at Facebook. [CACM '19]

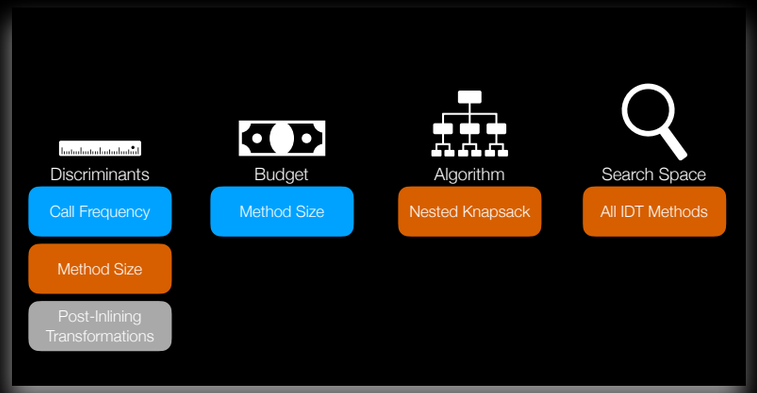


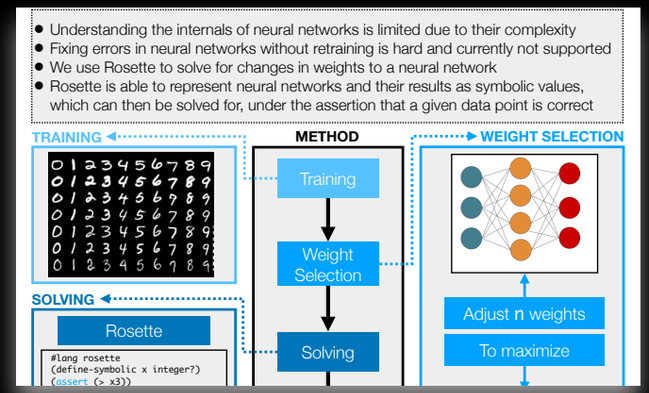
Semmle

Continuous Security Analysis

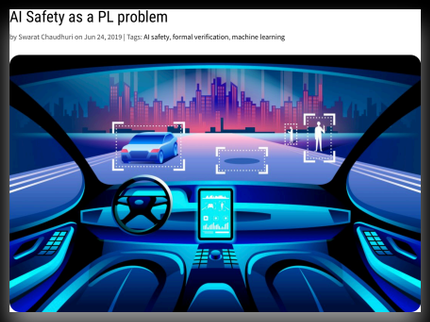
Image: LGTM.com

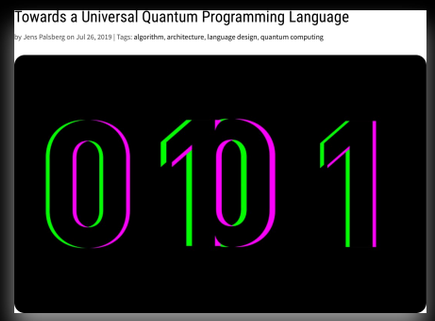


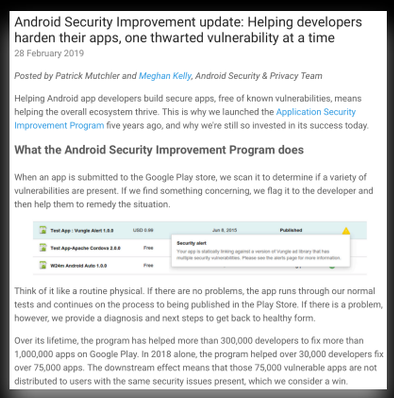


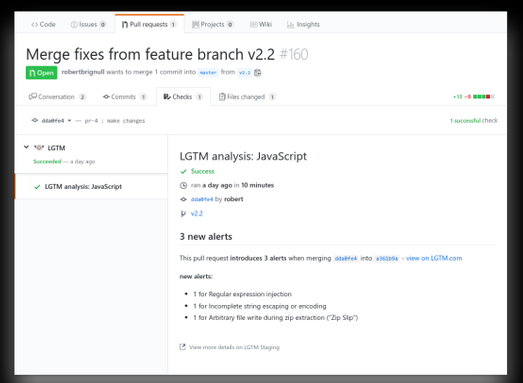


Future of Program Analysis



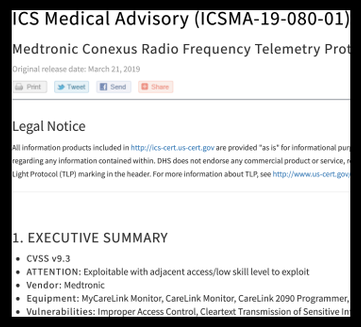






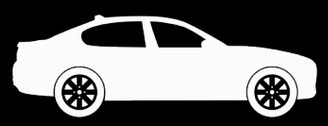
Extra Images: SIGPLAN Blog

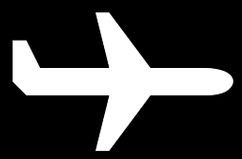
Program Analysis

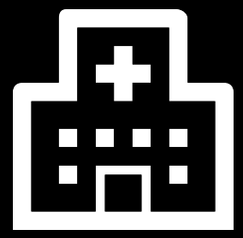


Program Analysis

@karimhamdanali







Code Navigation

Code Refactoring

Code Recommenders

Program Analysis

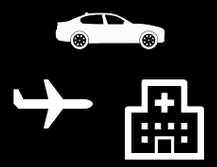
Parallelization

Constant Propagation

Static Inlining

Dead Code Elimination

Program Analysis



Code Navigation

Code Refactoring

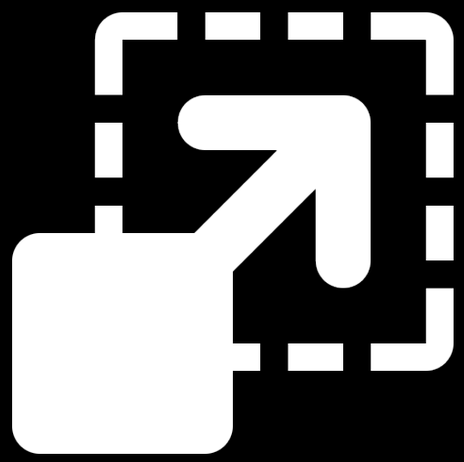
Code Recommenders

Program Analysis

coverity
A Synopsys Company

IFY

AbsInt

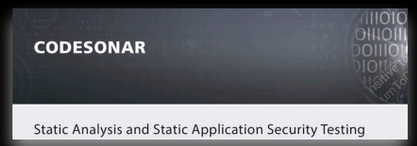


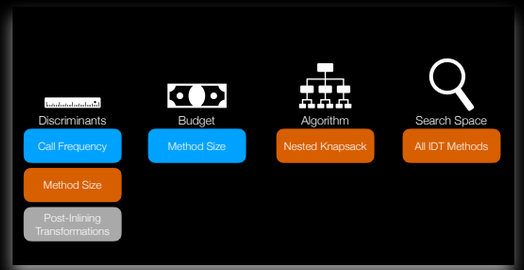
Scalability

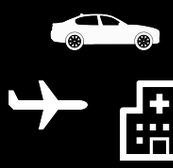
Precision

Usability

Program Analysis





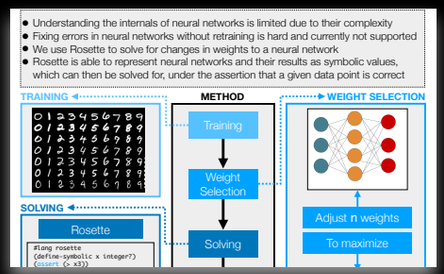


Code Navigation

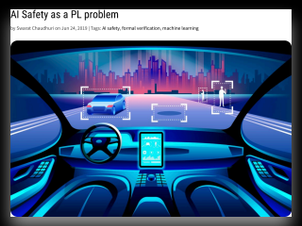
Code Refactoring

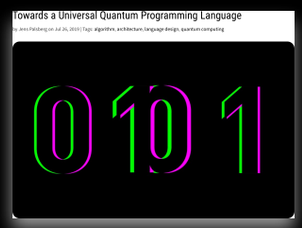
Code Recommenders

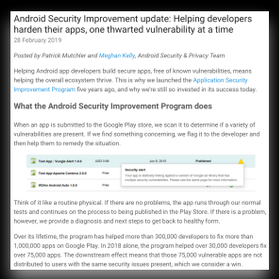
Program Analysis

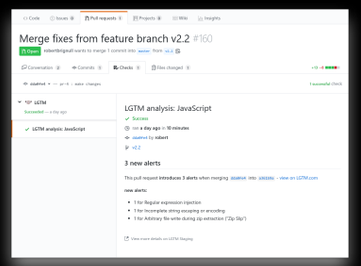


Future of Program Analysis



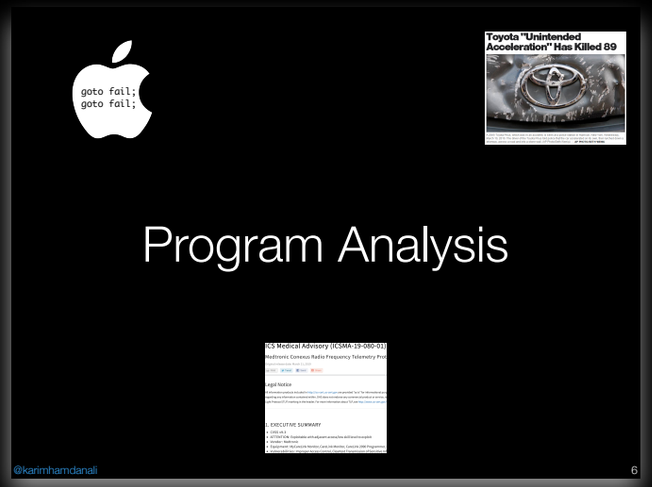


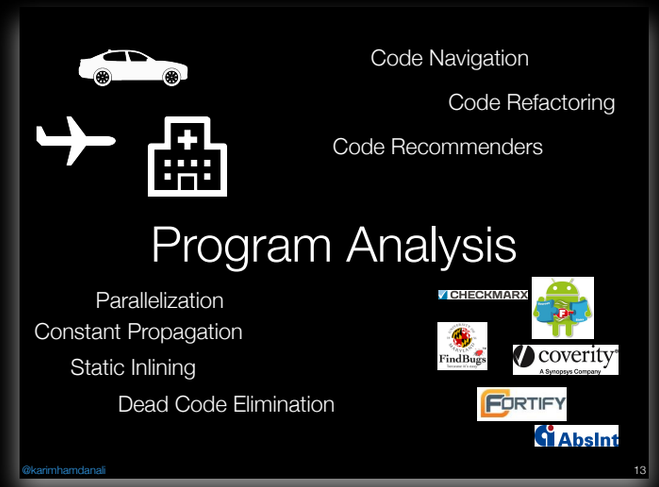




Extra Images: SIGPLAN Blog

@karimhamdanali





Is Program Analysis The Silver Bullet Against Software Bugs?

Karim Ali
University of Alberta
@karimhamdanali