Partial-Program Call Graphs

Applied Static Analysis 2016

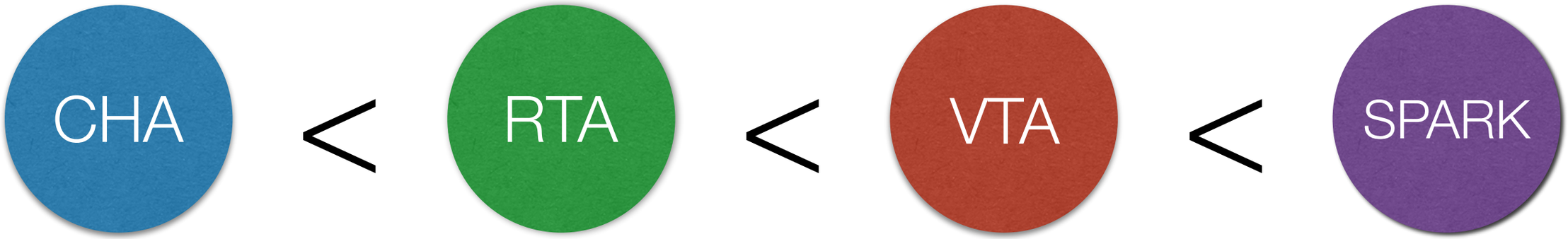
Karim Ali
@karim3ali

Michael Eichberg, Ben Hermann, Johannes Lerch, and Sebastian Proksch

Software Technology Group

TECHNISCHE UNIVERSITÄT DARMSTADT

Previously on APSA…


CHA
<
RTA
<
VTA
<
SPARK

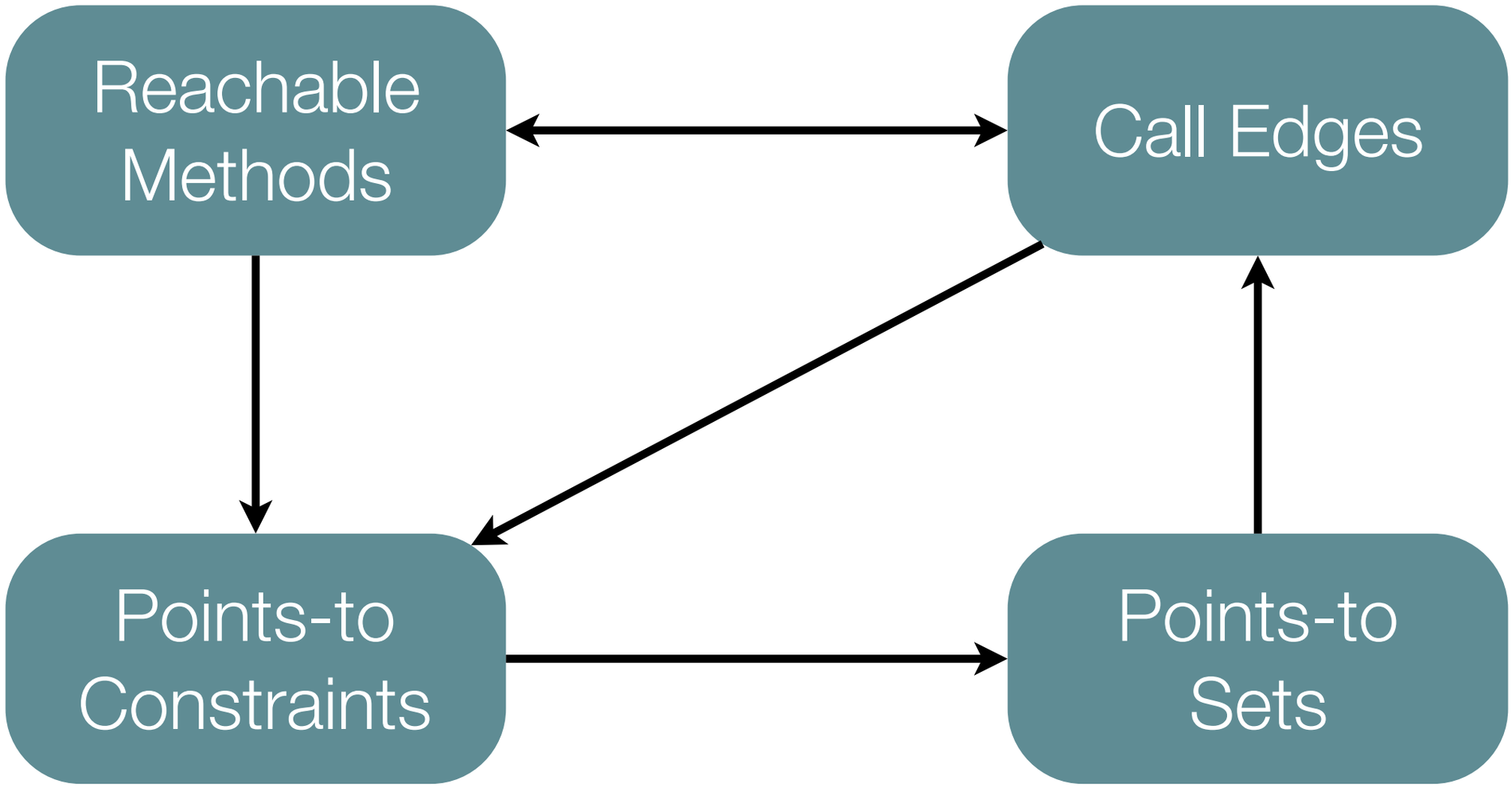
Static Call Graph Construction

Static Call Graph Construction

Determining Target Calls

Points-to Analysis

Static Call Graph Construction



Whole-Program Call Graph

Application Code

+

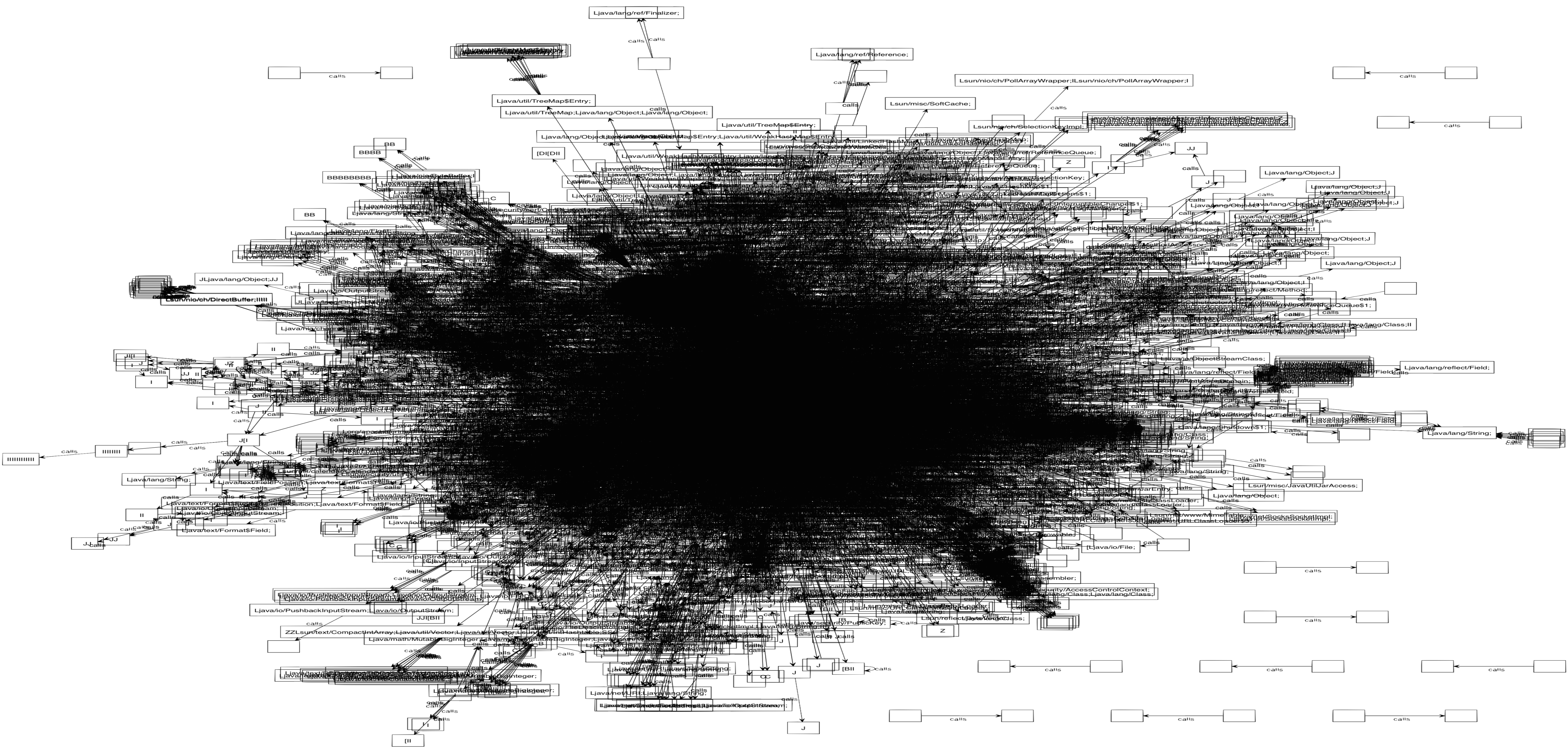
Library Code


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

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

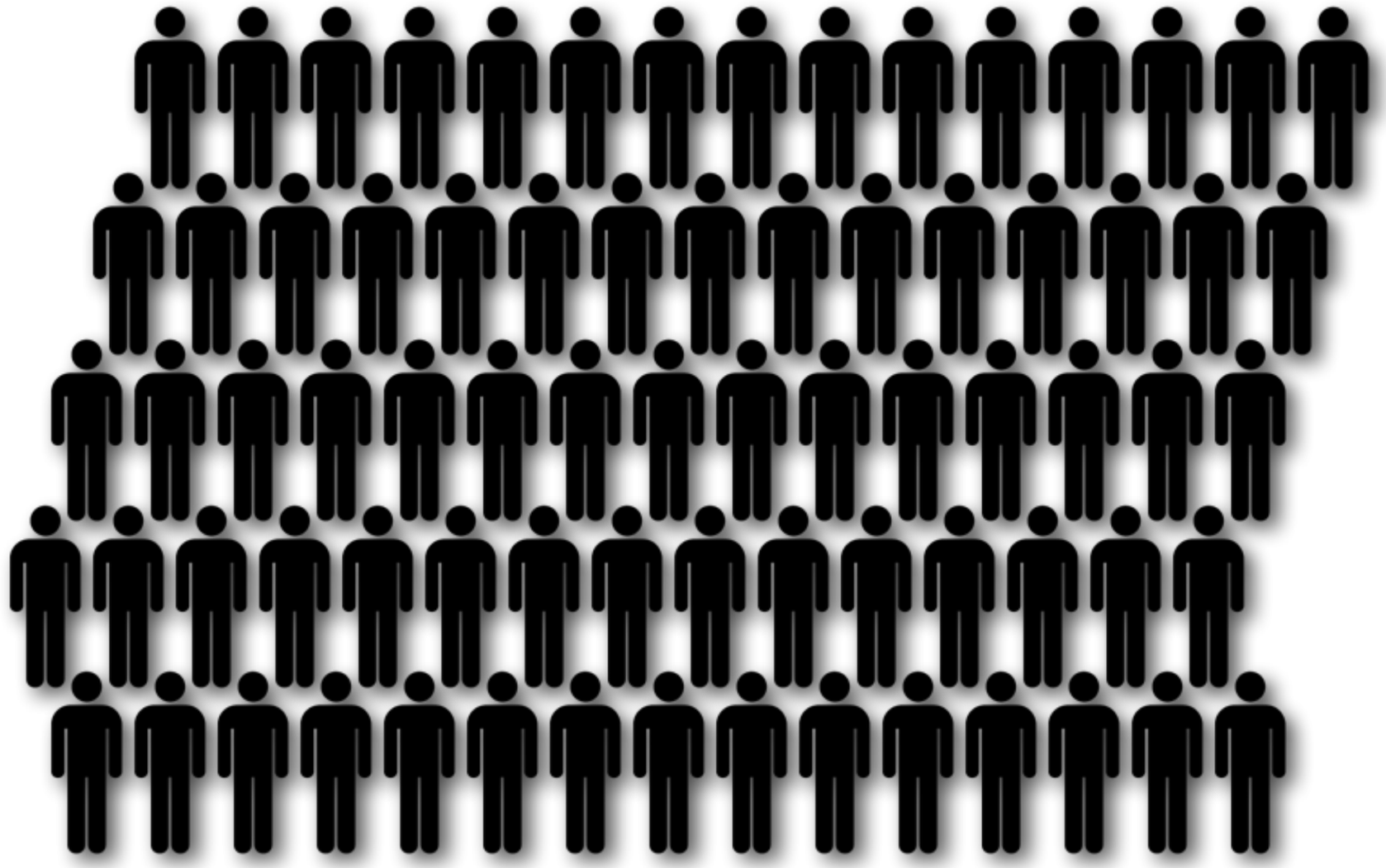
- > 20 seconds
- > 9,000 reachable methods
- > 100,000 call edges

Hello, World!



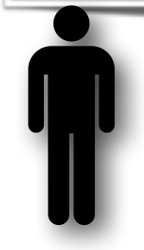
Ljava/lang/ref/Finalizer;
Ljava/lang/ref/Reference;
Lsun/nio/ch/PollArrayWrapper;ILsun/nio/ch/PollArrayWrapper;I
Lsun/misc/SoftCache;
Ljava/util/TreeMap$Entry;
Ljava/util/TreeMap;Ljava/lang/Object;Ljava/lang/Object;
JLjava/lang/Object;JJ
Ljava/lang/reflect/Field;
Ljava/lang/String;
Lsun/misc/JavaUtilJarAccess;
Ljava/text/Format$Field;
Ljava/io/PushbackInputStream;Ljava/io/OutputStream;
[Ljava/io/File;
calls

Partial-Program Call Graph



Partial-Program Call Graph

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

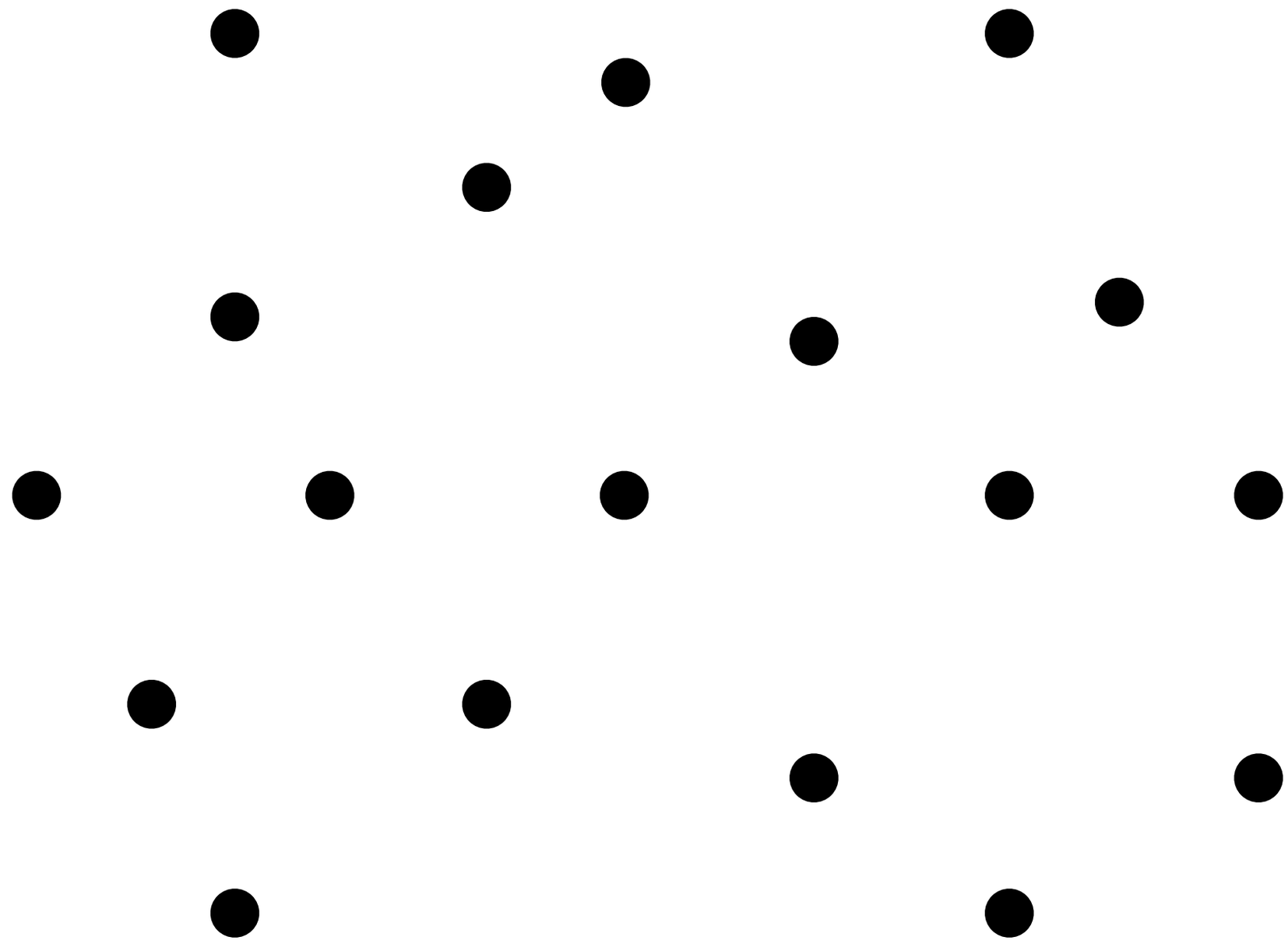
what about callbacks?

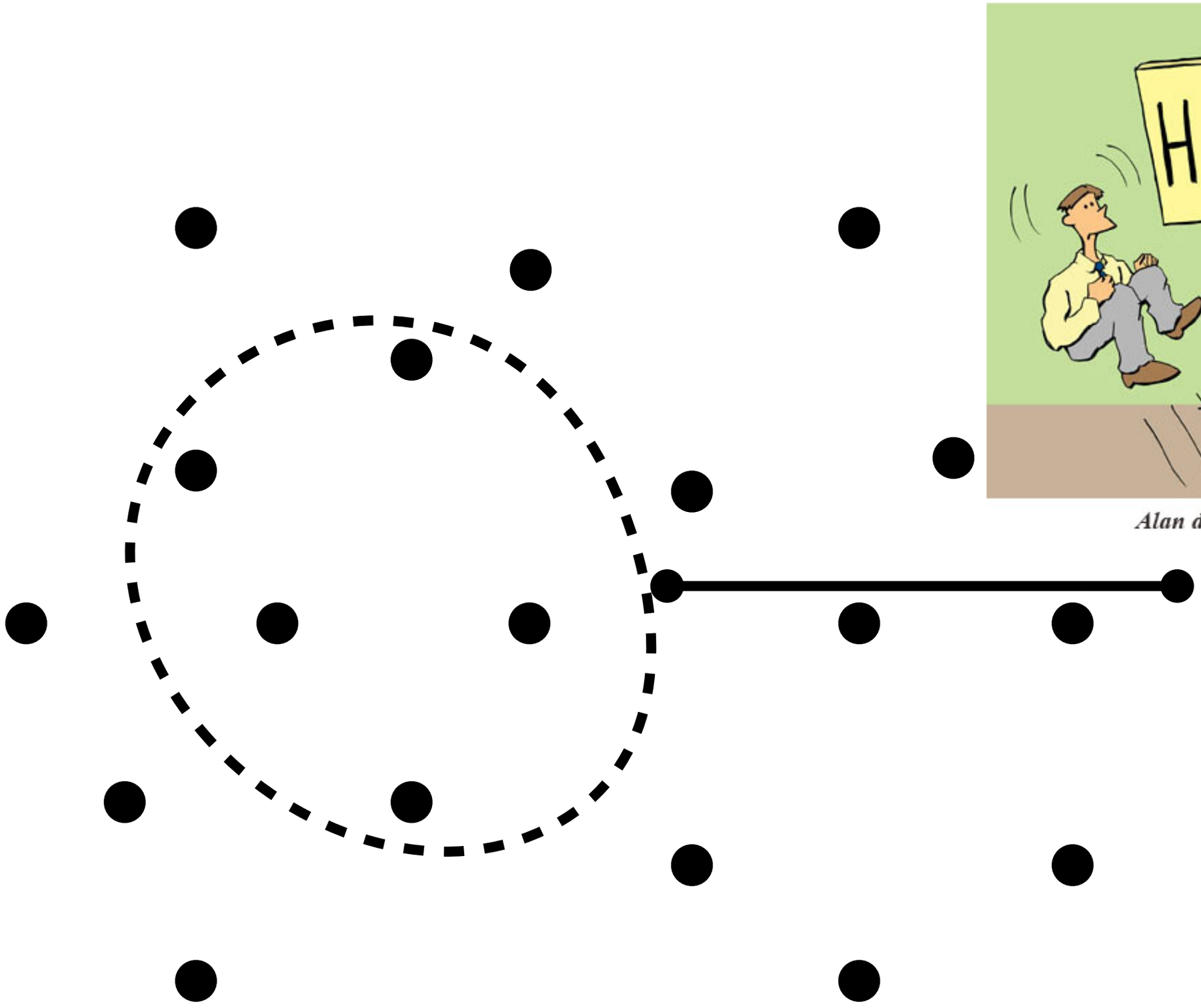
ignore non-application program elements (e.g., system libraries)?

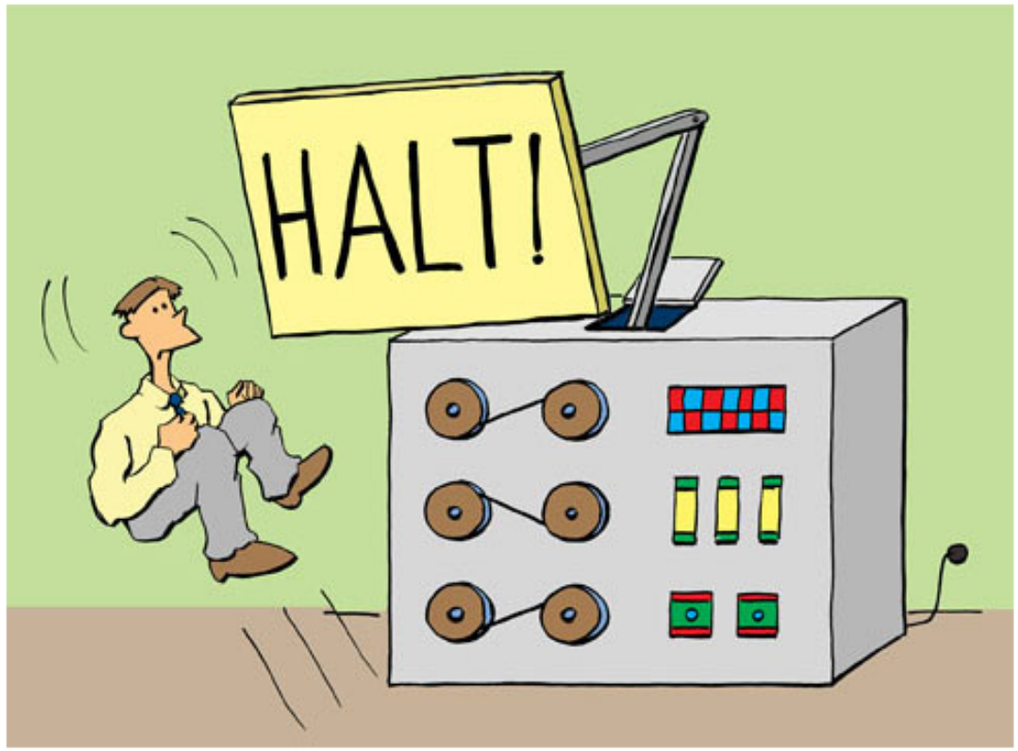
this would be unsound but better than nothing

I am NOT interested in those

Partial-Program Call Graph

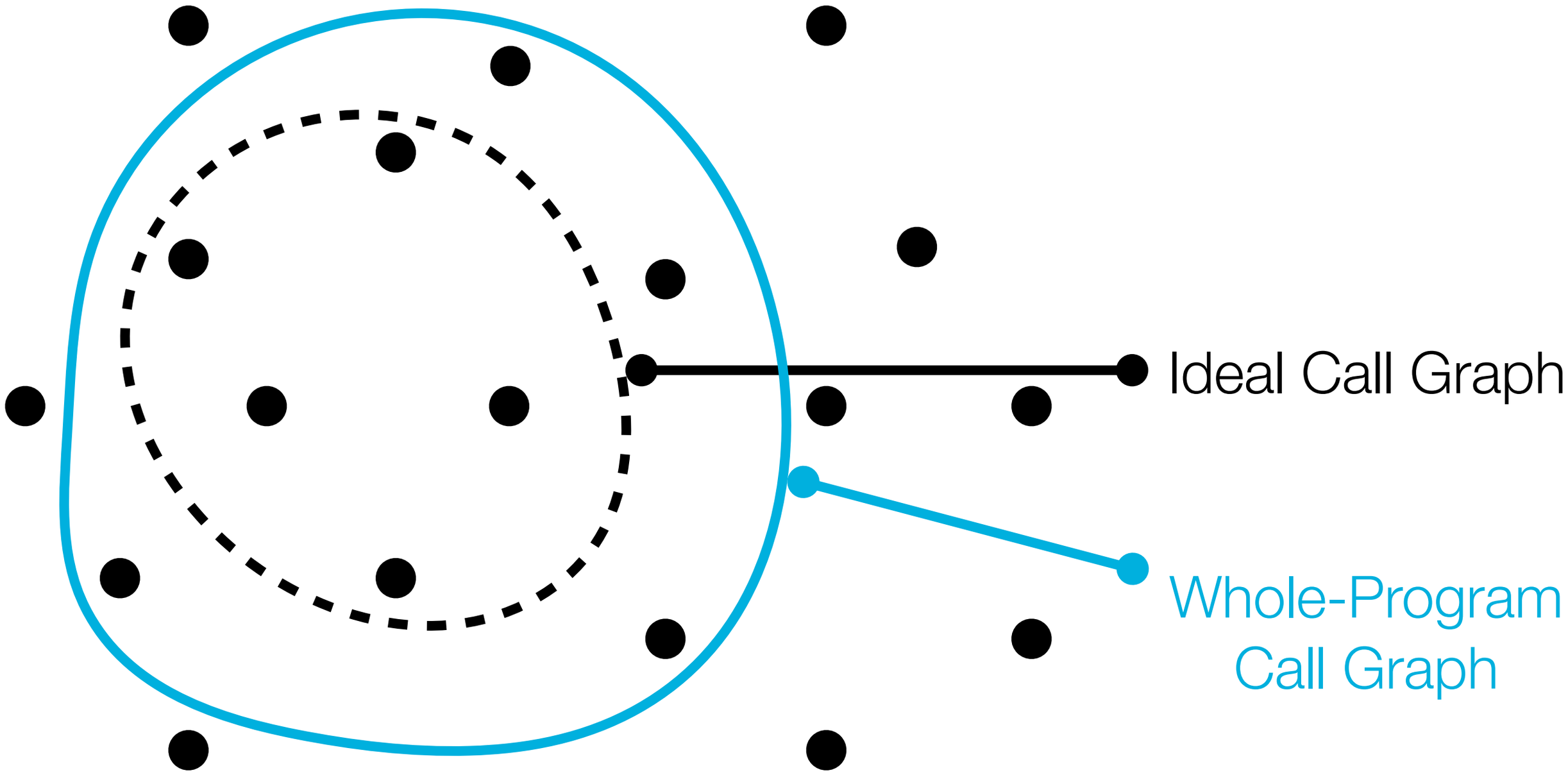


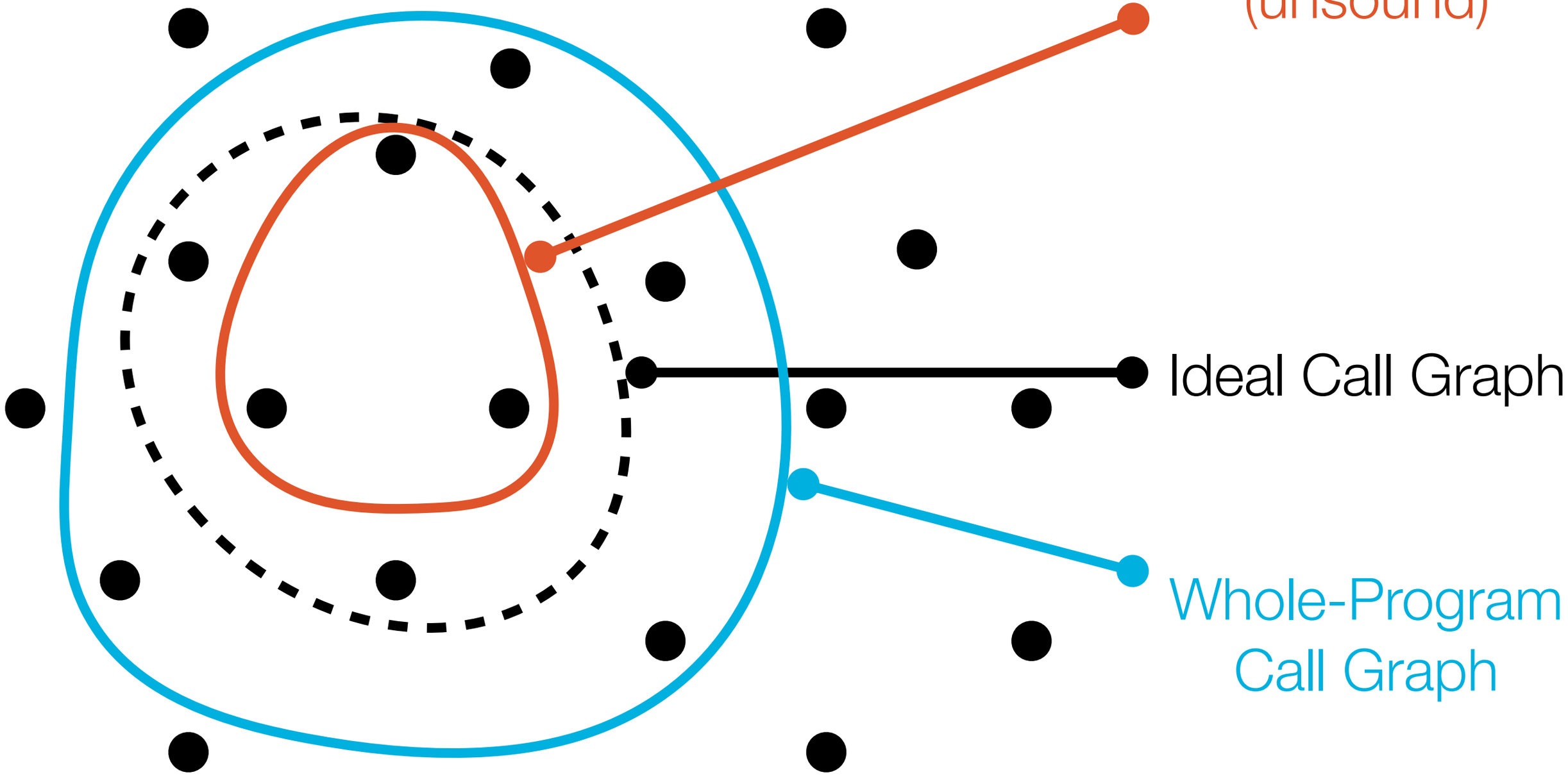


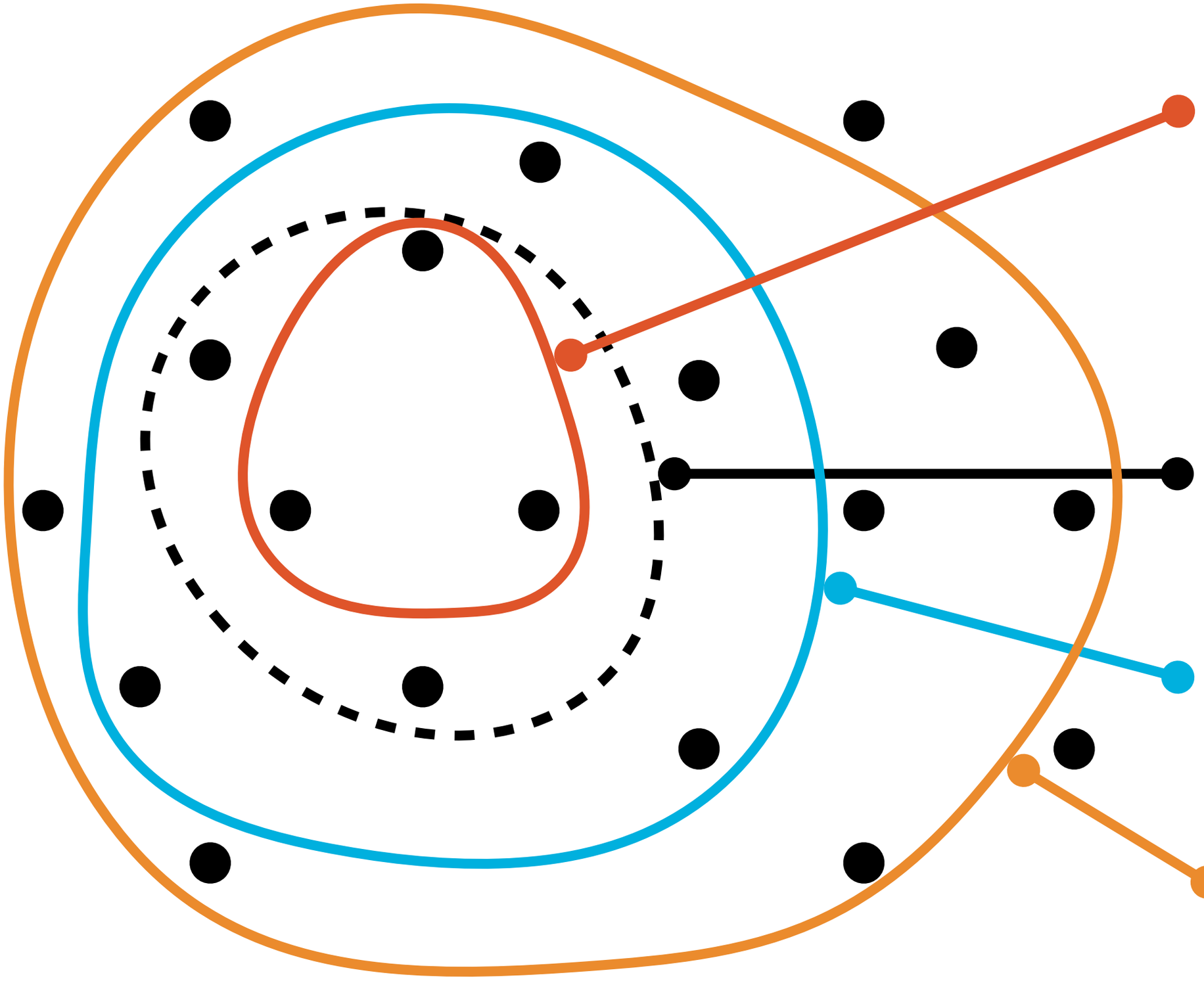


Alan designed the perfect computer

Ideal Call Graph


Ideal Call Graph
Whole-Program
Call Graph


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



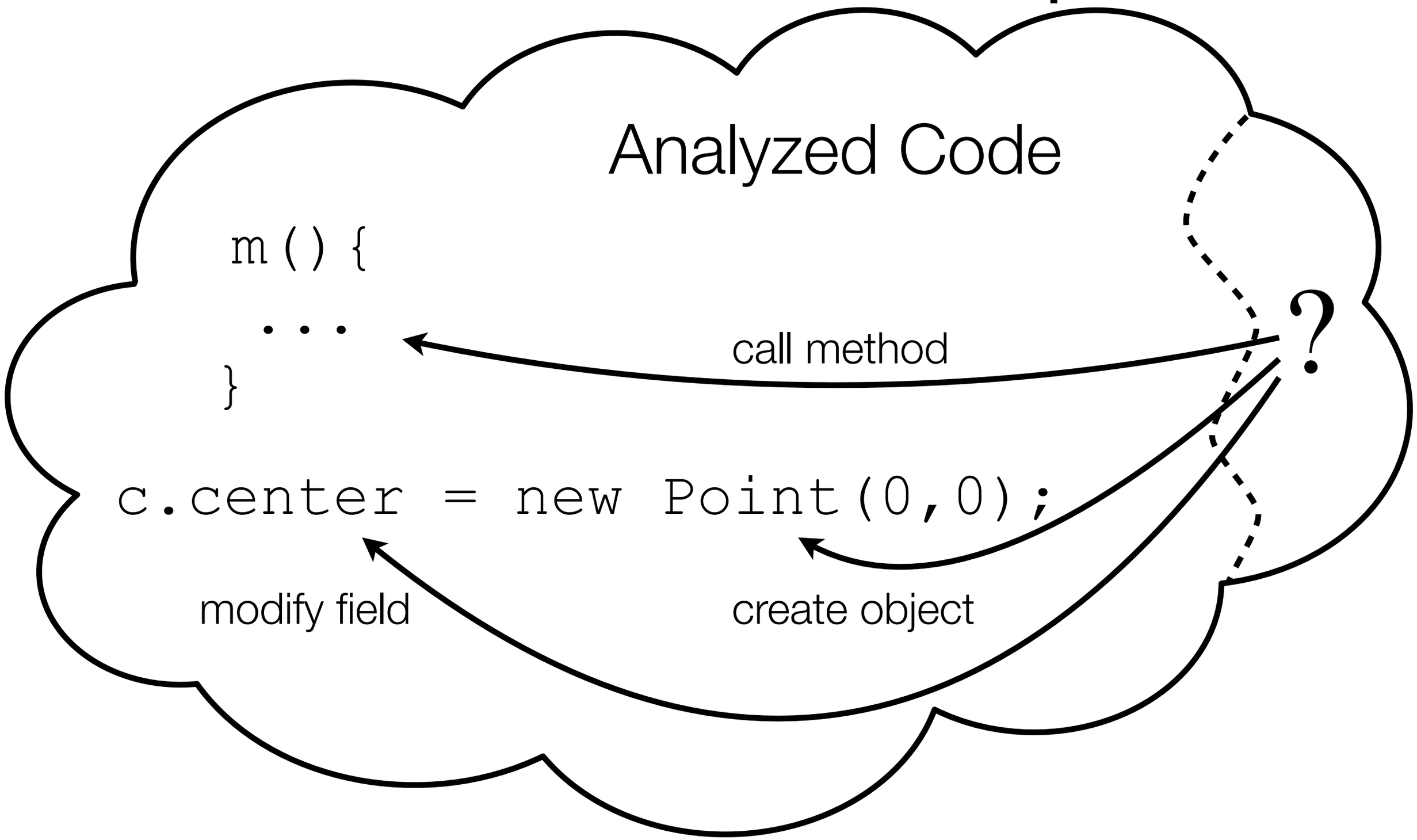
Incomplete
Call Graph
(unsound)

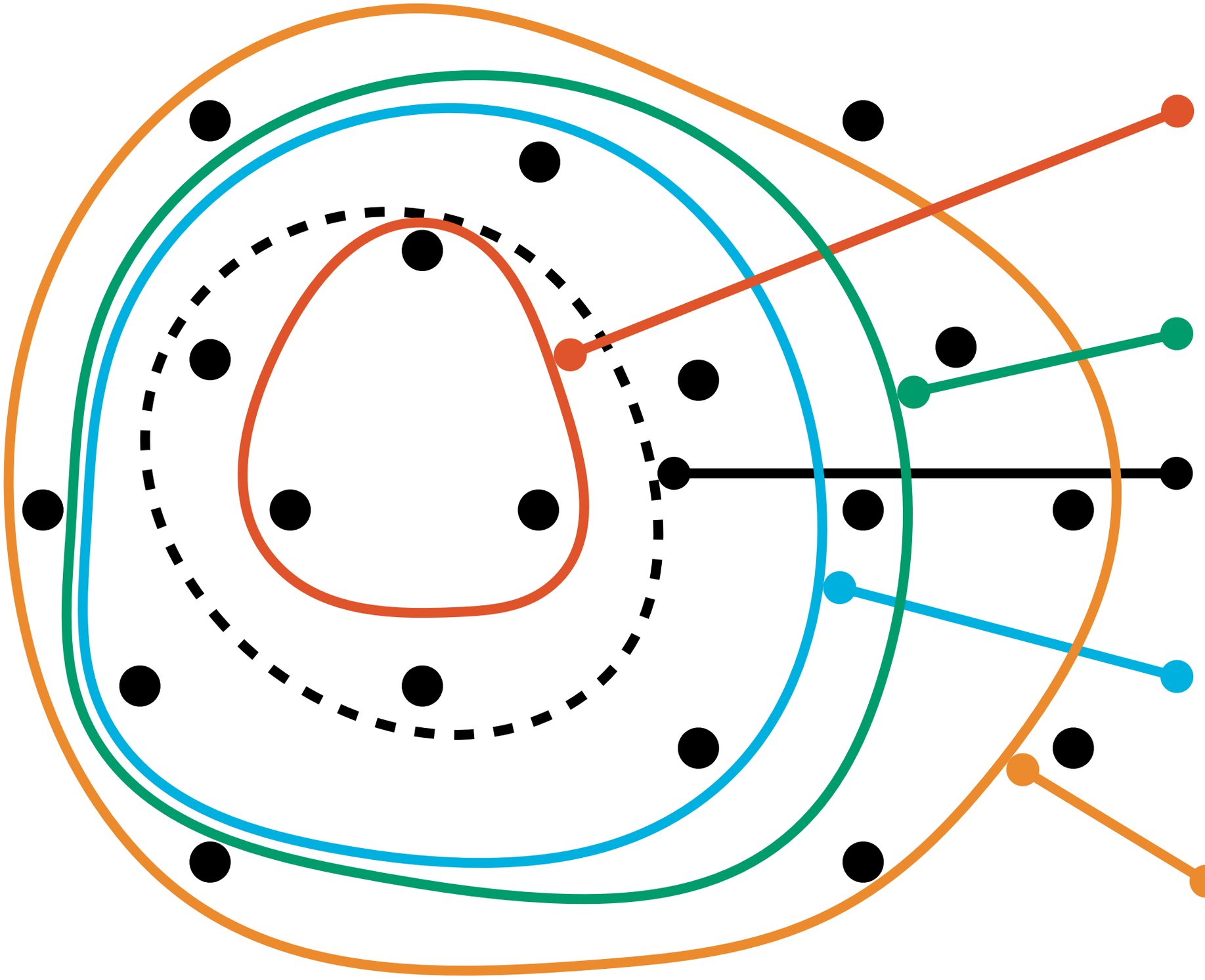
Ideal Call Graph

Whole-Program
Call Graph

Conservative
Call Graph
(highly imprecise)

Conservative Assumptions





Incomplete
Call Graph
(unsound)

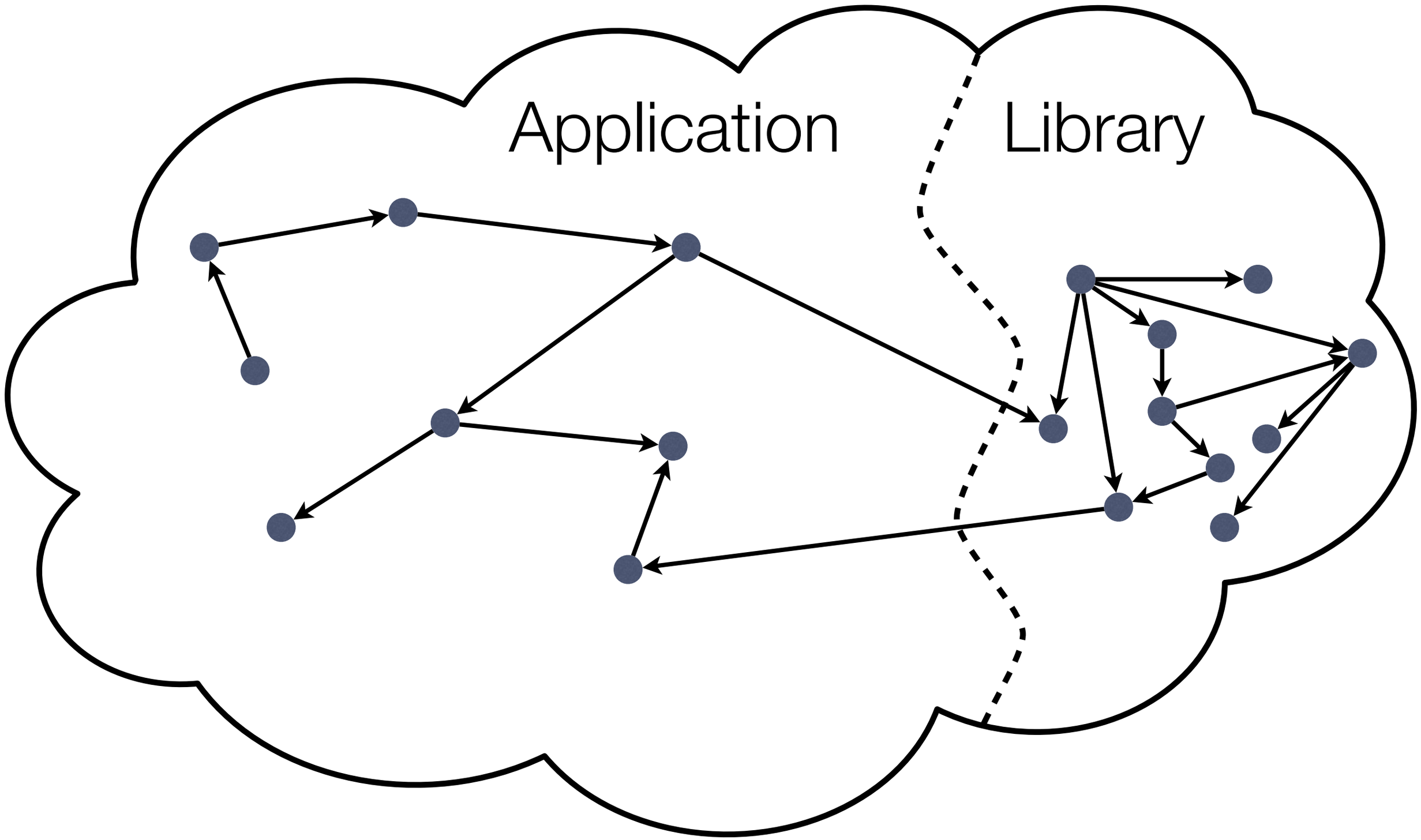
Partial-Program
Call Graph

Ideal Call Graph

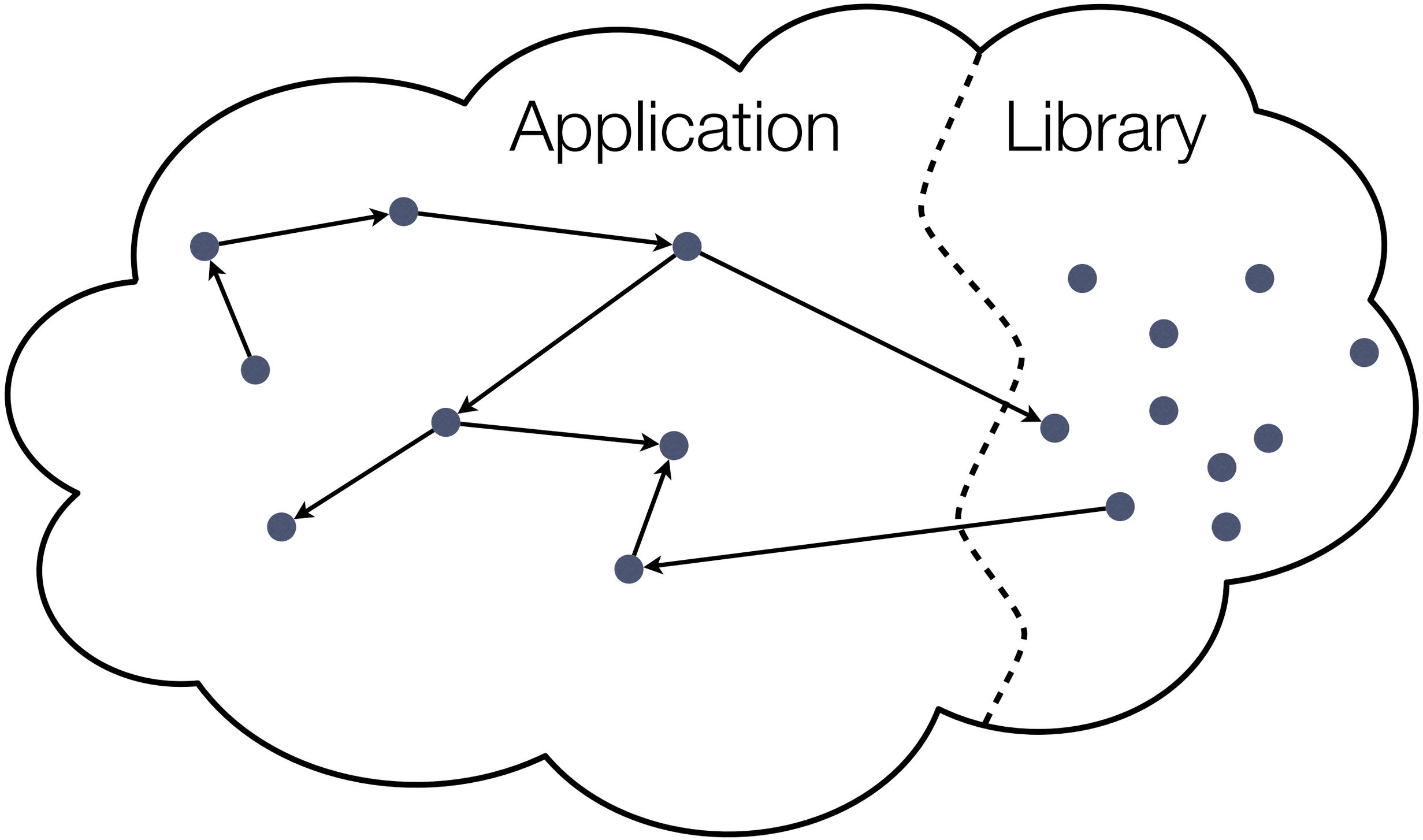
Whole-Program
Call Graph

Conservative
Call Graph
(highly imprecise)

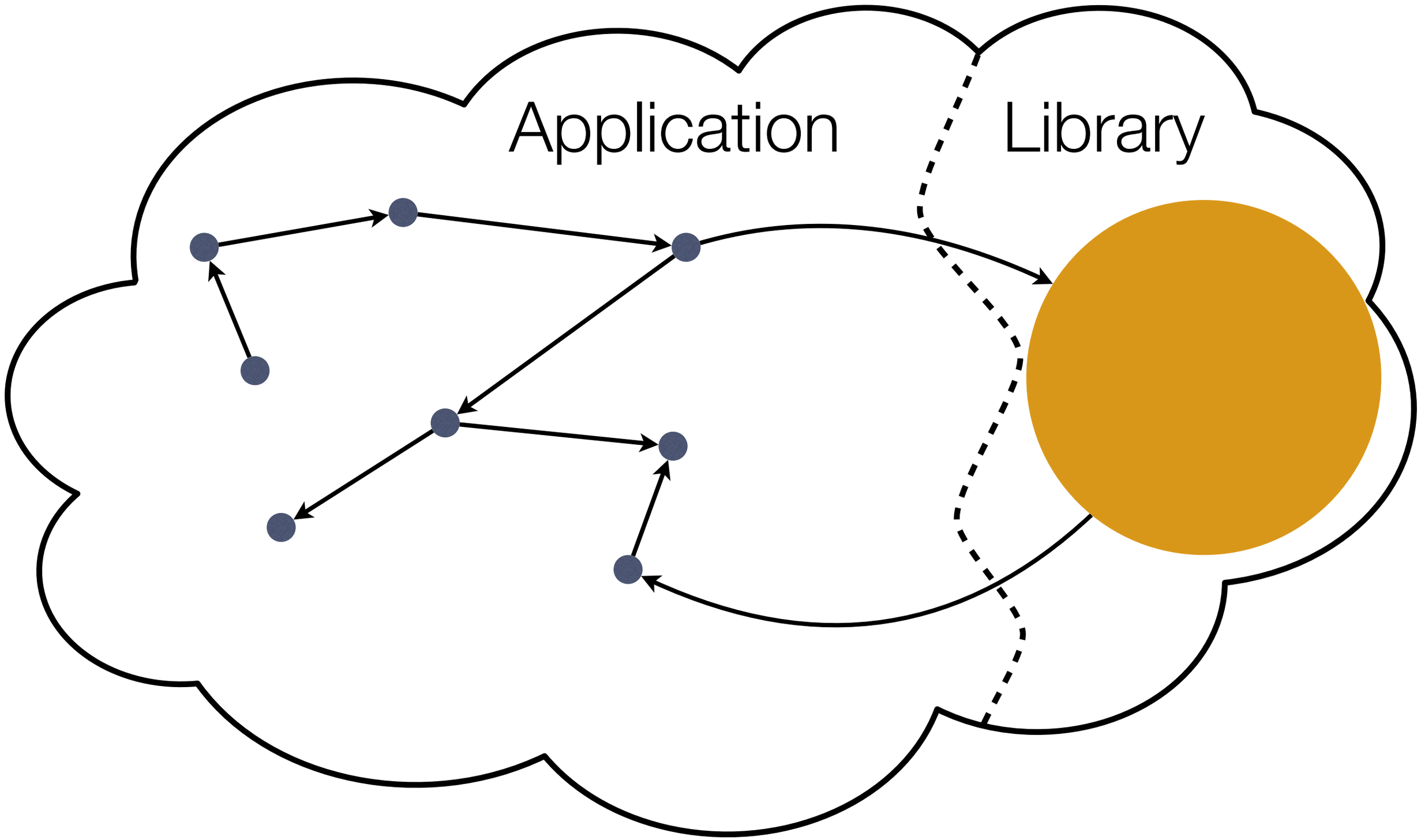
Sound Partial-Program Call Graph



Sound Partial-Program Call Graph



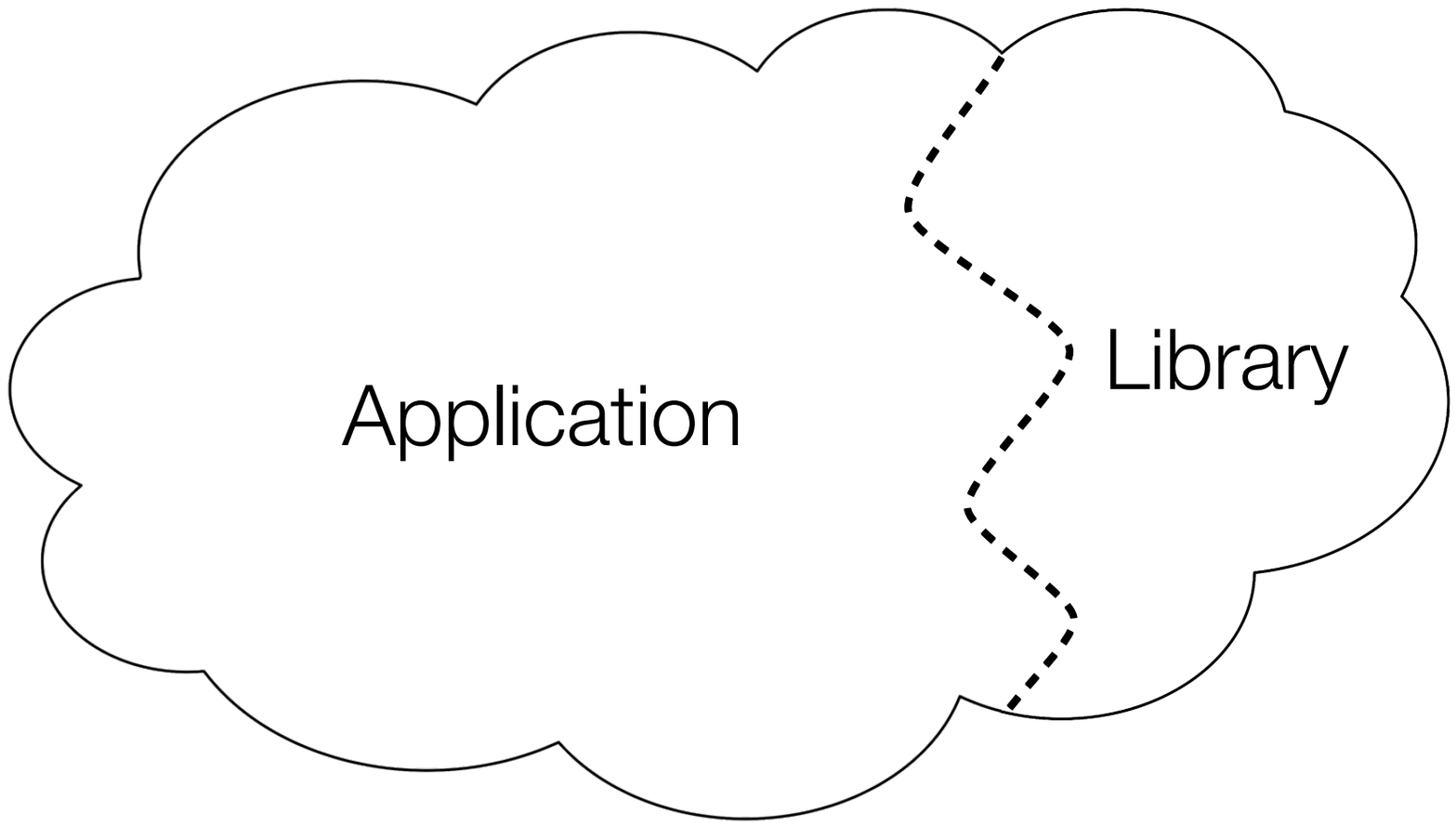
Sound Partial-Program Call Graph



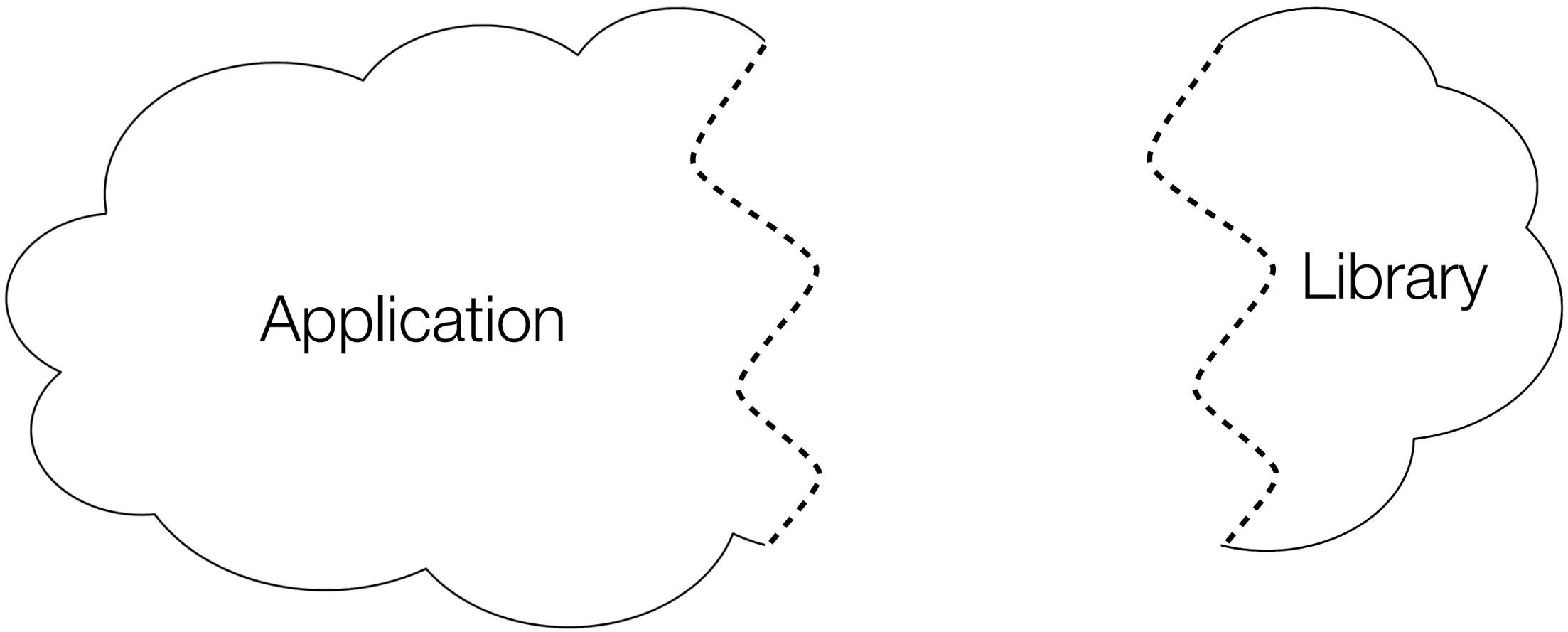
The Separate Compilation Assumption

Karim Ali and Ondřej Lhoták. 2012. Application-Only Call Graph Construction. In Proceedings of the 26th European conference on Object-Oriented Programming (ECOOP'12), 688-712.

The Separate Compilation Assumption



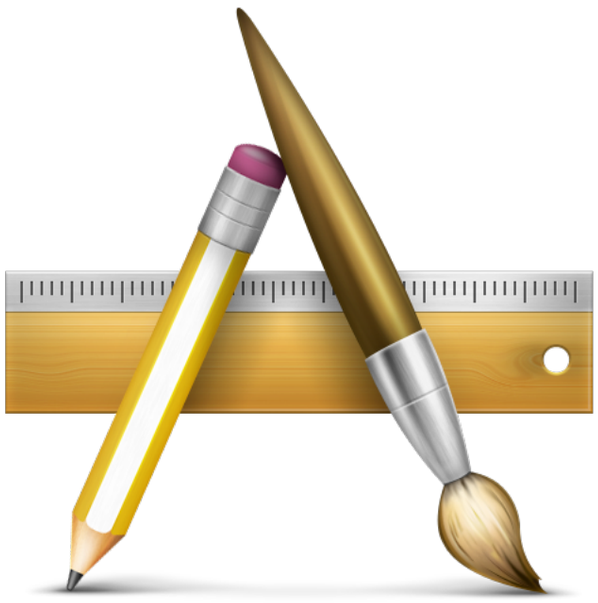
The Separate Compilation Assumption



Constraints

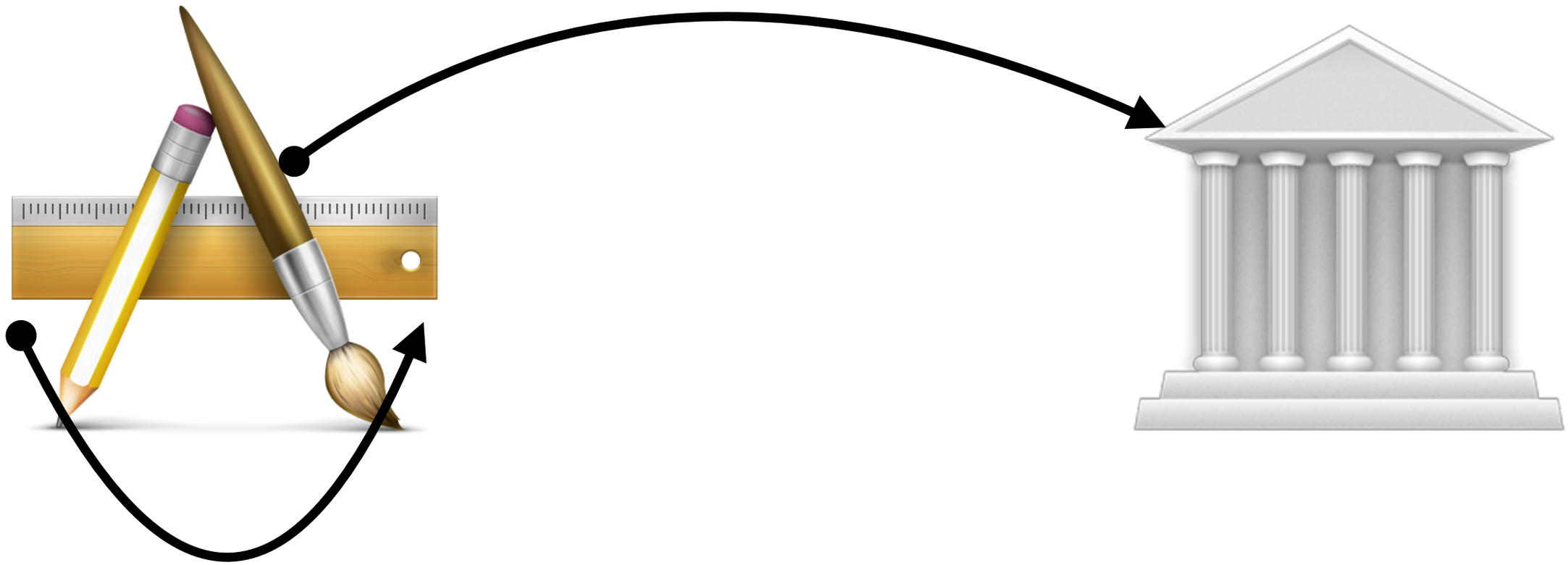
1. Class Hierarchy
2. Class Instantiation
3. Local Variables
4. Method Calls
5. Field Access
6. Array Access
7. Static Initialization
8. Exception Handling

C1. Class Hierarchy



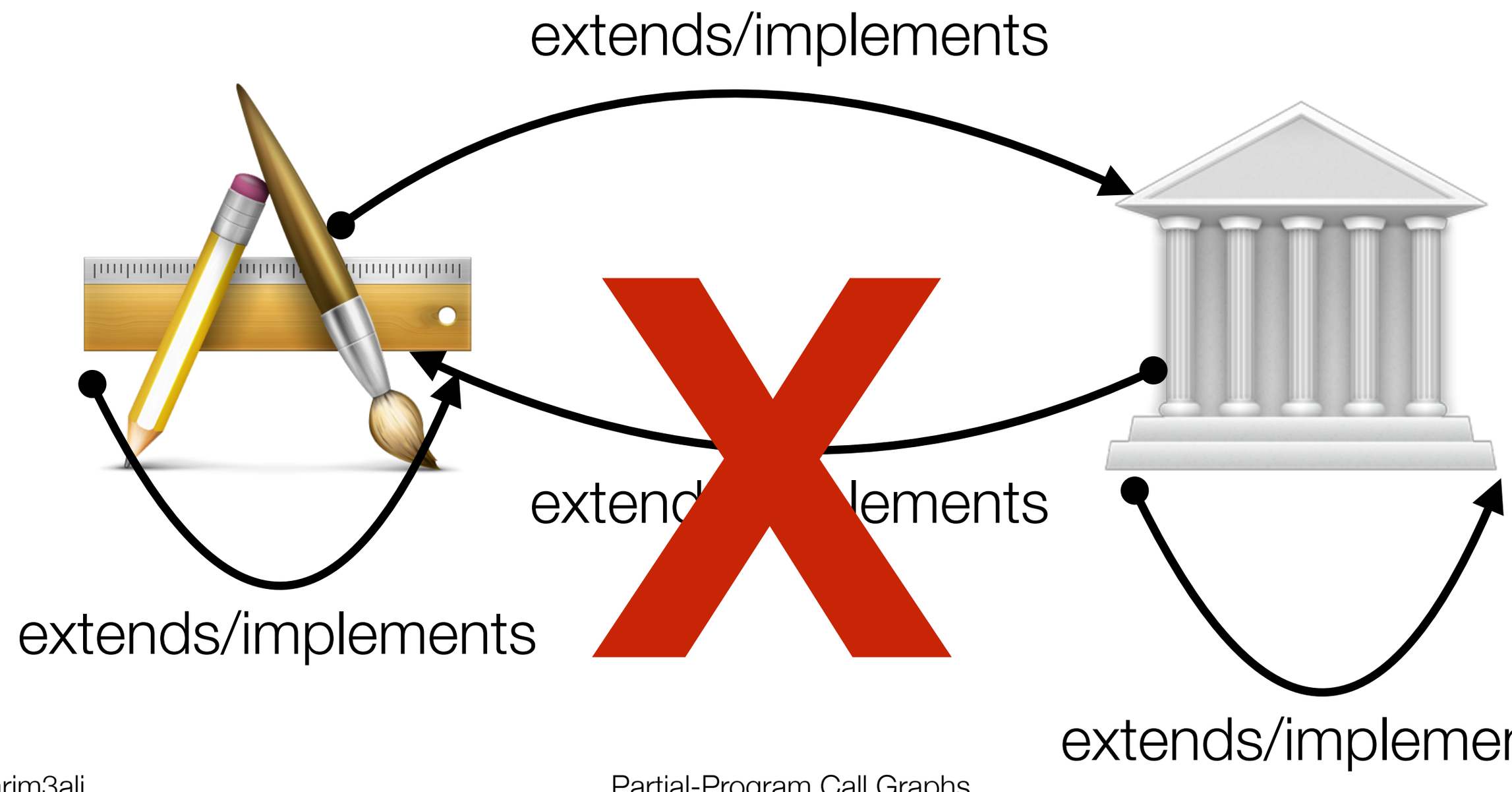
C1. Class Hierarchy

extends/implements

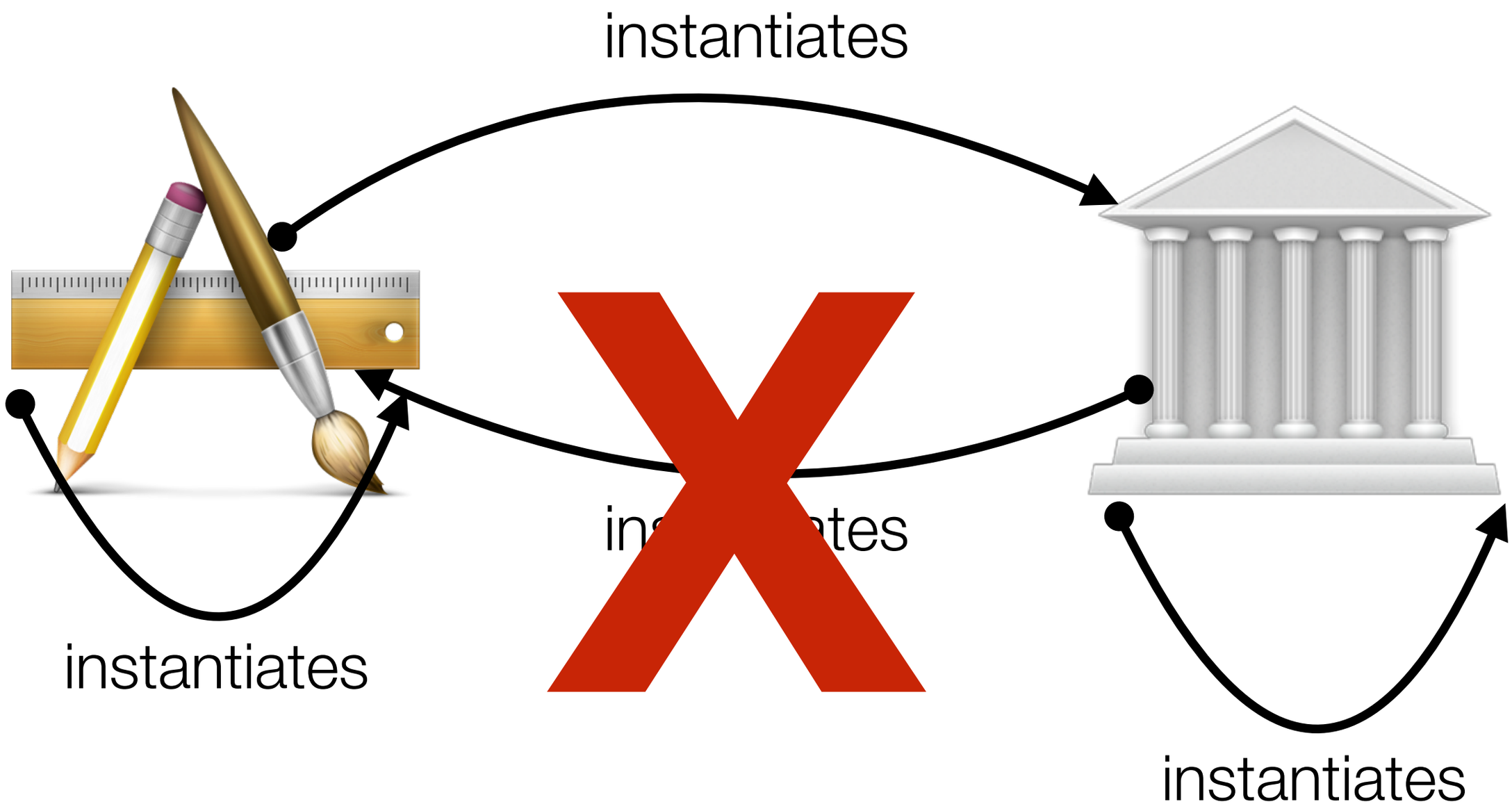


extends/implements

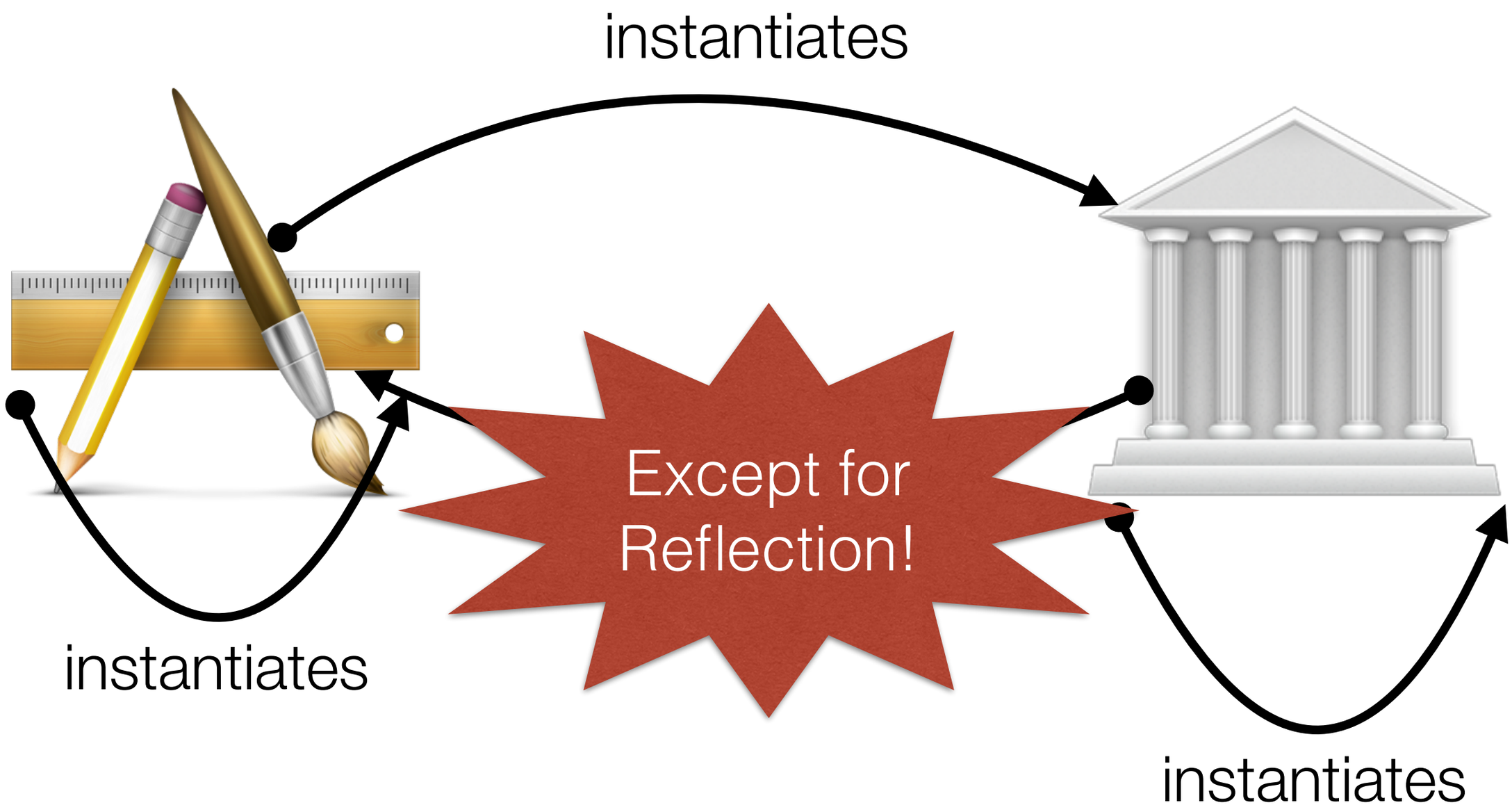
C1. Class Hierarchy



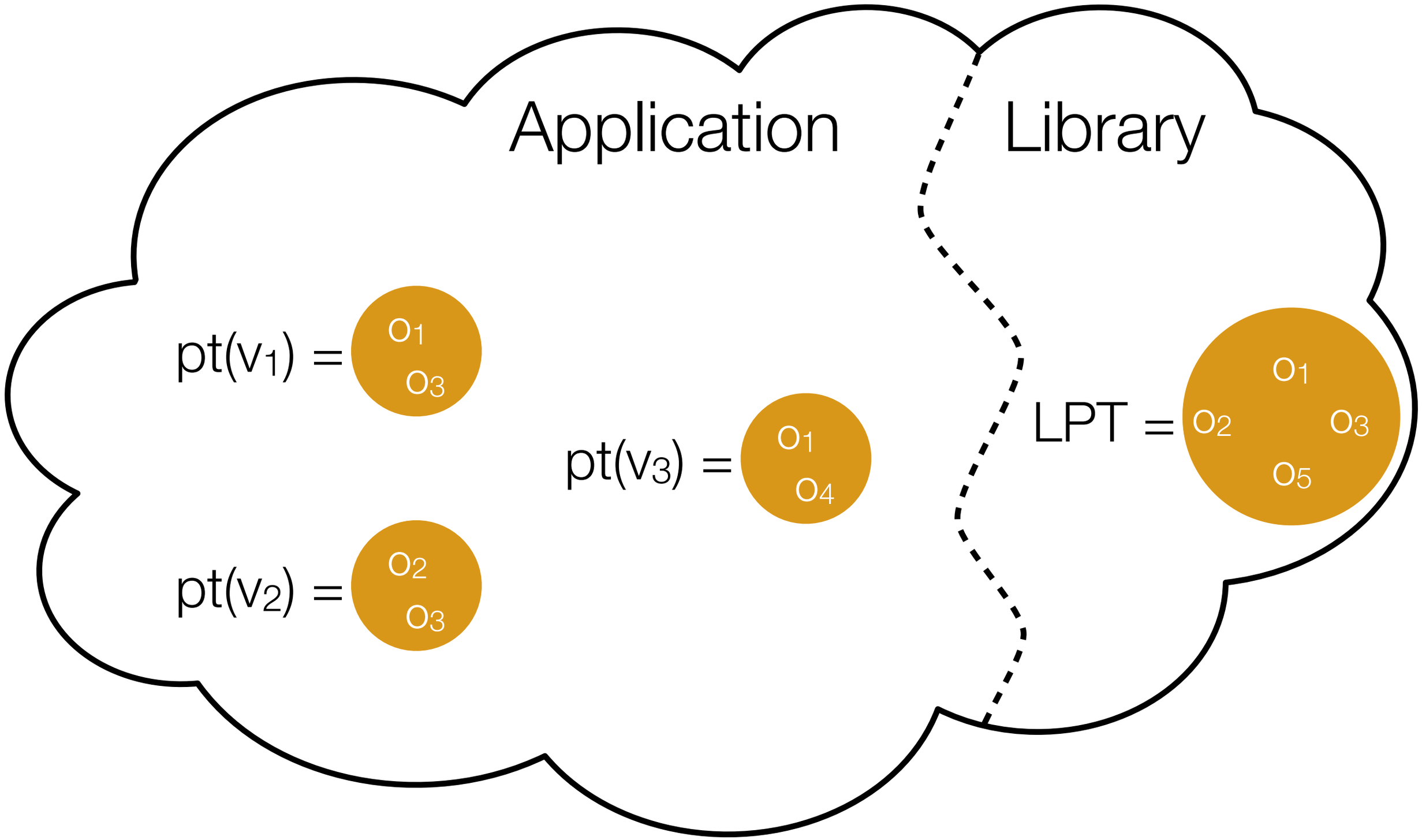
C2. Class Instantiation



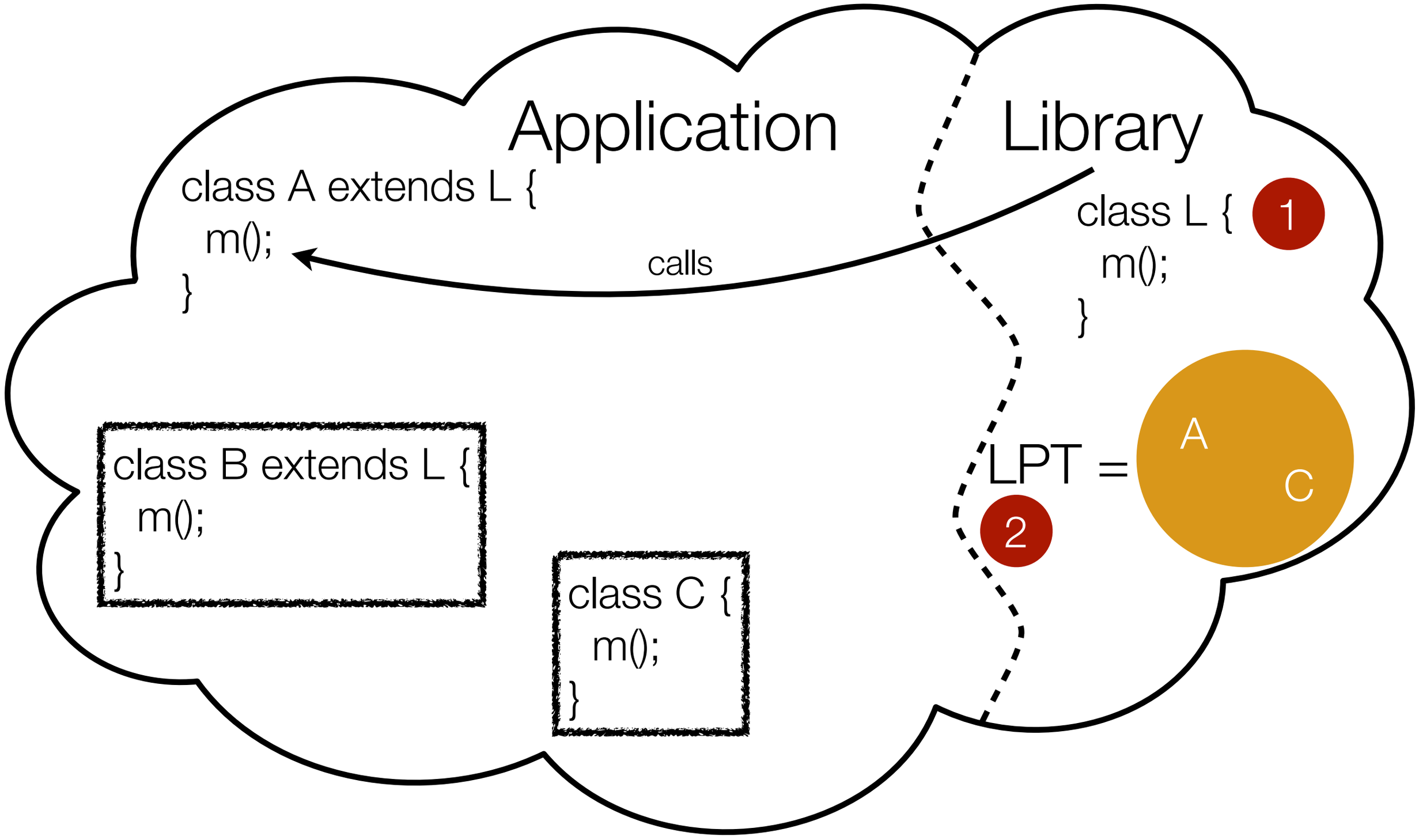
C2. Class Instantiation



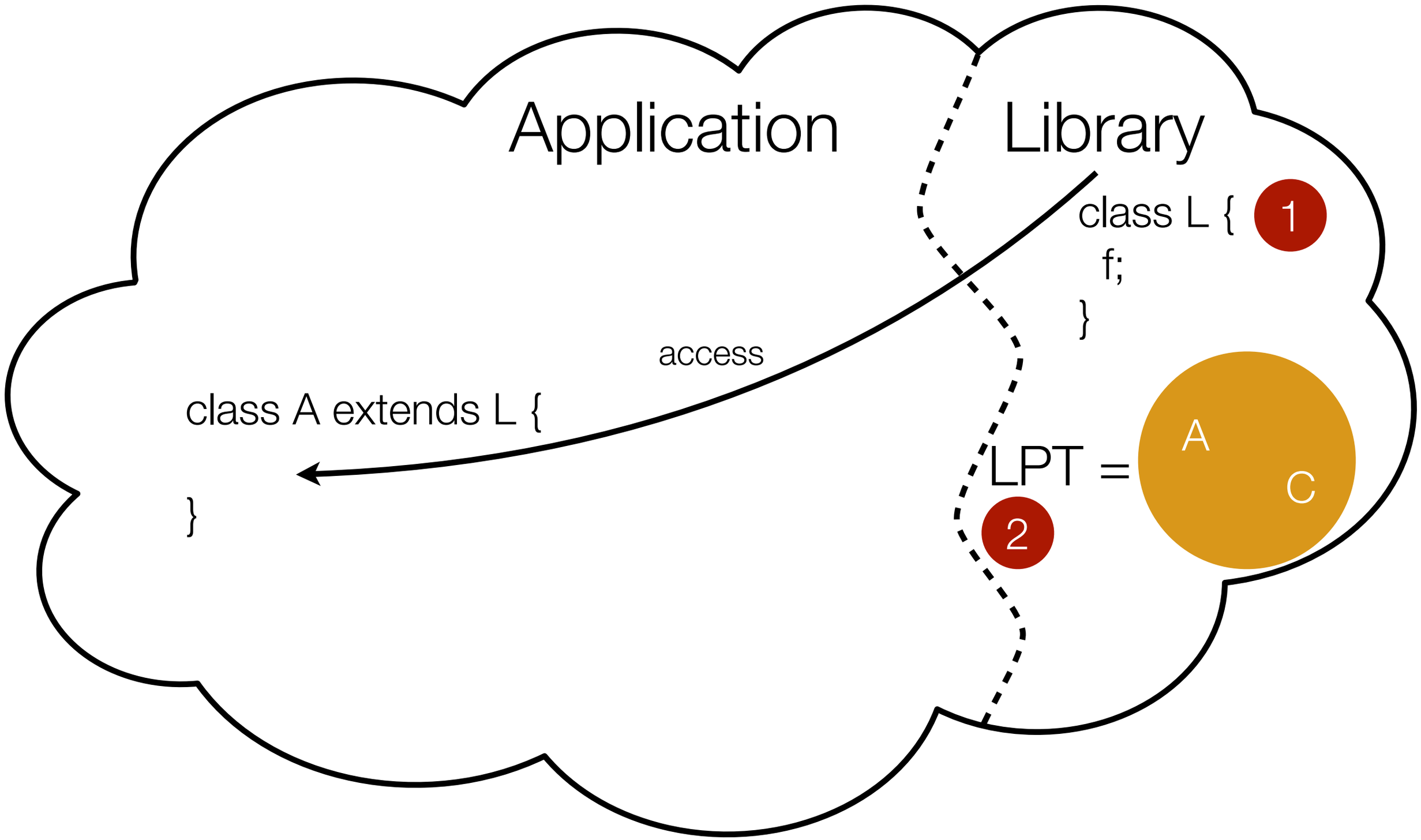
C3. Local Variables



C4. Method Calls



C5. Field Access



C6. Array Access

LPT = 

C7. Static Initialization

The library causes the loading and static initialization of classes that it instantiates.

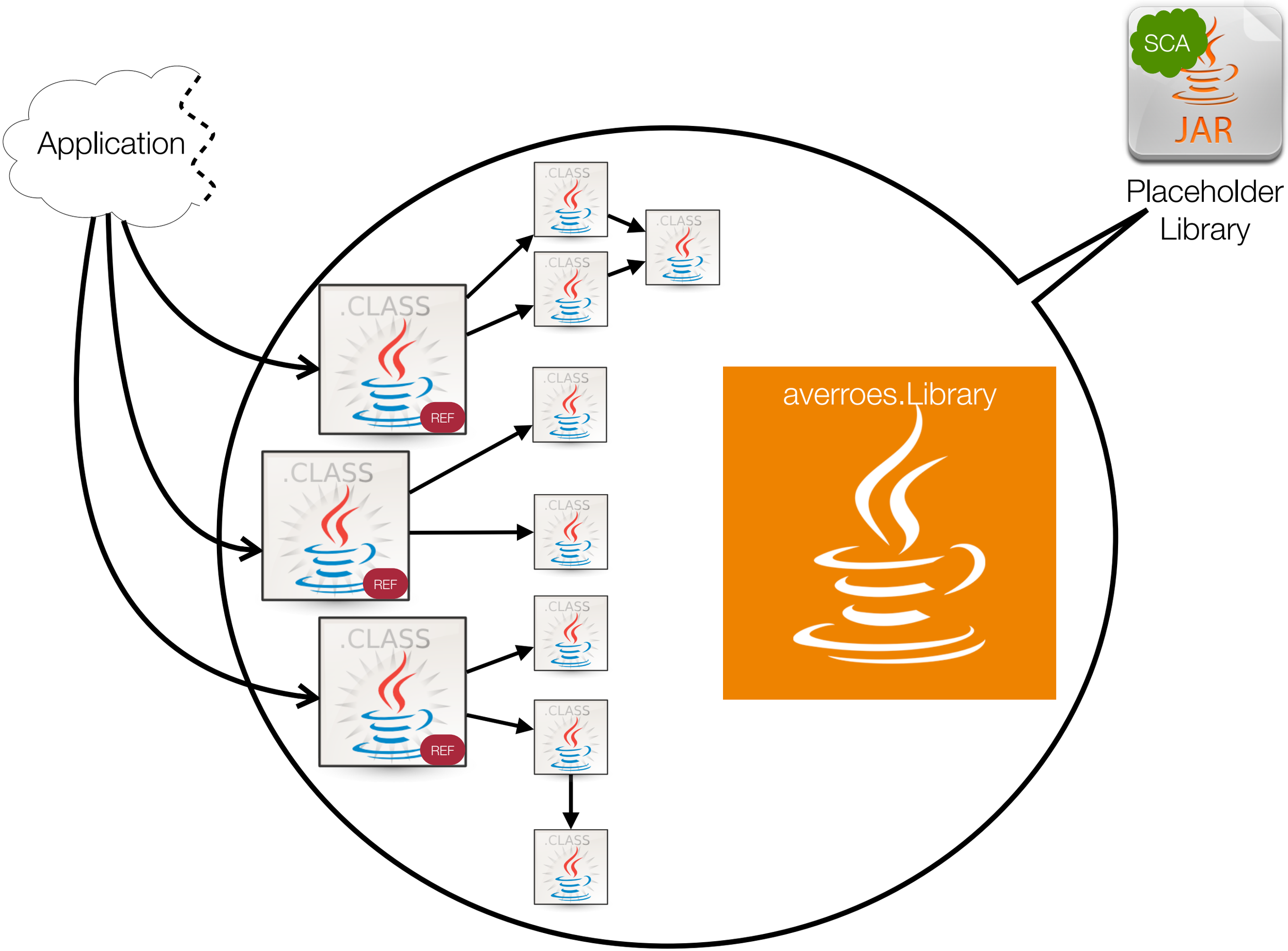
C8. Exception Handling

LPT = 

Karim Ali and Ondřej Lhoták. 2013. Averroes: Whole-Program Analysis Without the Whole Program. In Proceedings of the 27th European conference on Object-Oriented Programming (ECOOP'13), 378-400.

AVERROES

Placeholder Library


Application
SCA
JAR
Placeholder
Library
.CLASS
REF
averroes.Library


averroes.Library

```
public class AverroesLibraryClass {

  public static Object libraryPointsTo;
  public static void doItAll() {
    // Class instantiation

    // Library callbacks

    // Field writes

    // Array element writes

    // Exception Handling
  }
}
```

```
<modifiers> T method(T1, ..., Tn) {
    T1 r1 := @parameter1: T1;
    ...
    Tn rn := @parametern: Tn;
    C r0 = @this: C;

    Averroes.libraryPointsTo = r0;
    Averroes.libraryPointsTo = r1;
    ...
    Averroes.libraryPointsTo = rn;

    Averroes.doItAll();
    return (T) Averroes.libraryPointsTo;
}
```

Identity Statements

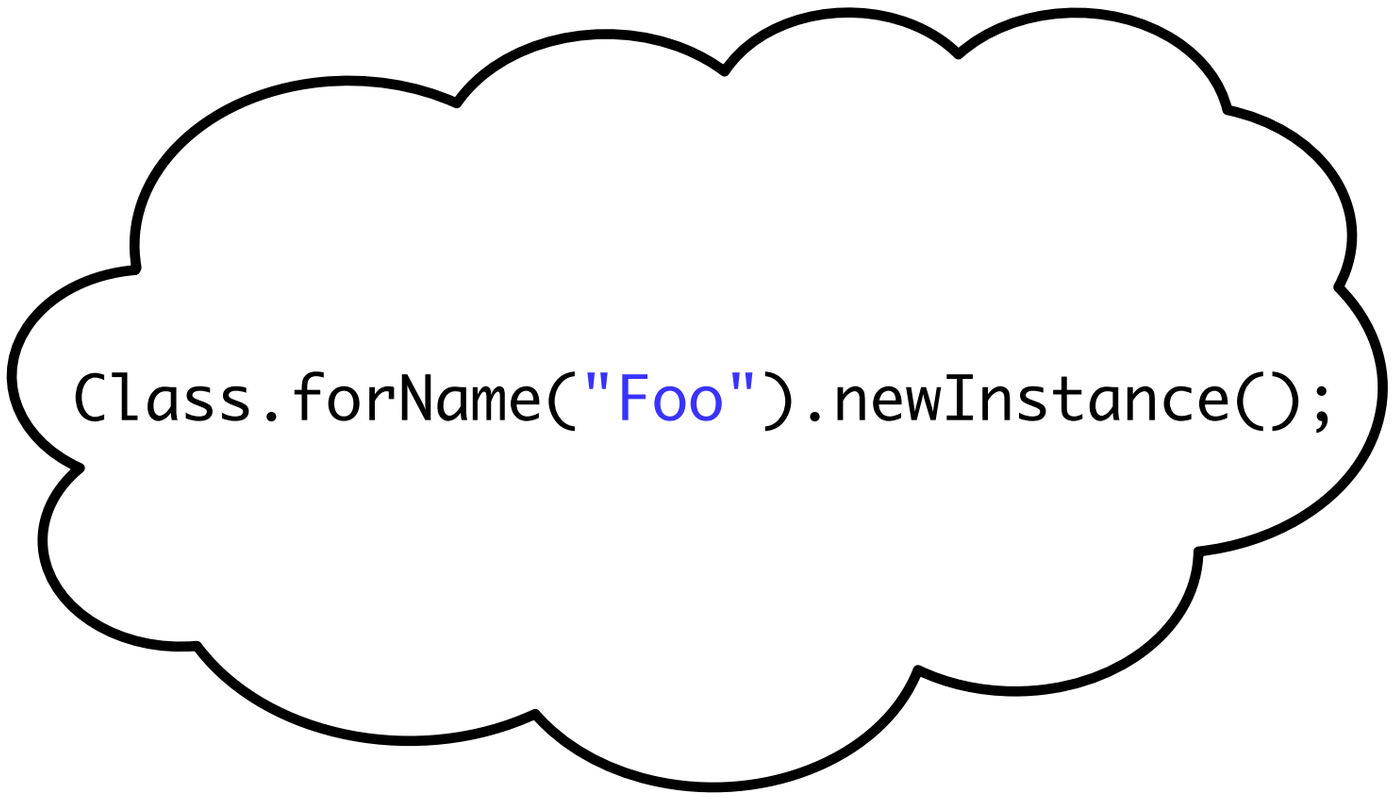
Parameter Assignments

Method Footer

Only for non-static methods

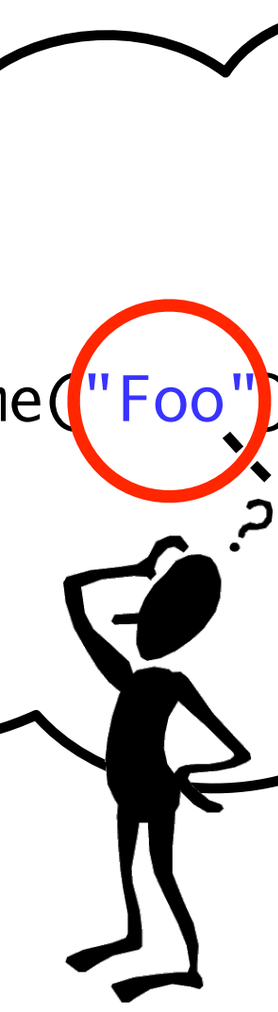
Modelling Reflection

Application



Application

```
Class.forName("Foo").newInstance();
```



Placeholder Library

```
new Foo();
```


JAR
Constant Pool
JAR
AVERROES
SCA
JAR
Placeholder Library
TamiFlex
Uses of Reflection

Does this actually work in practice?


WALA
T. J. WATSON LIBRARIES FOR ANALYSIS


Soot


DOOP

Library Size

Original Library
~ 50 MB

Library Size

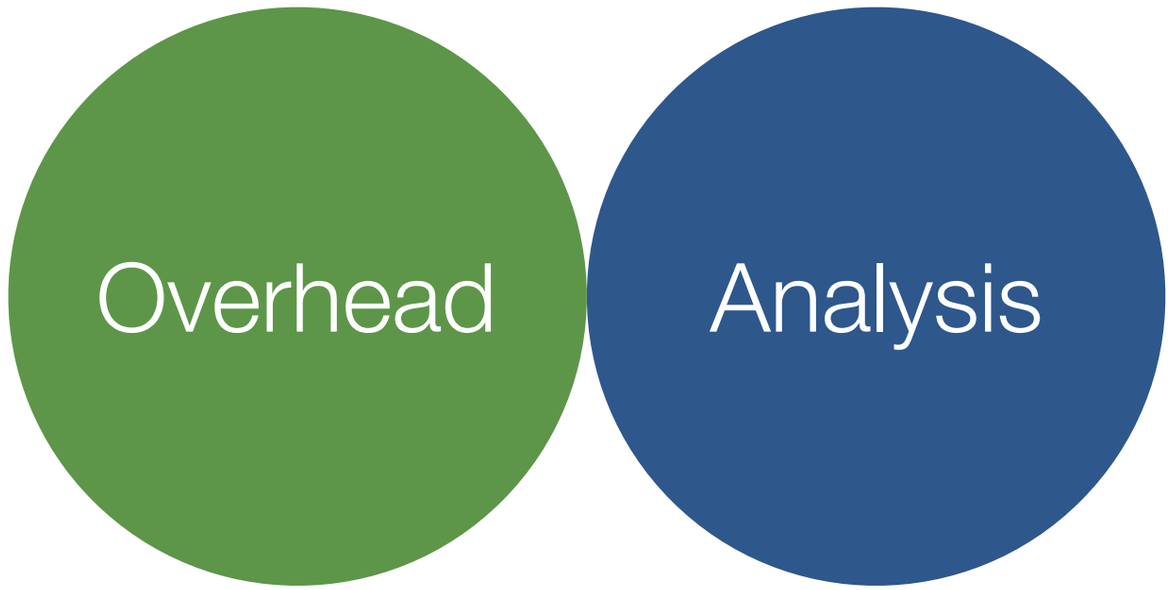
Original Library
~ 50 MB

Placeholder Library
~ 80 KB

Execution Time

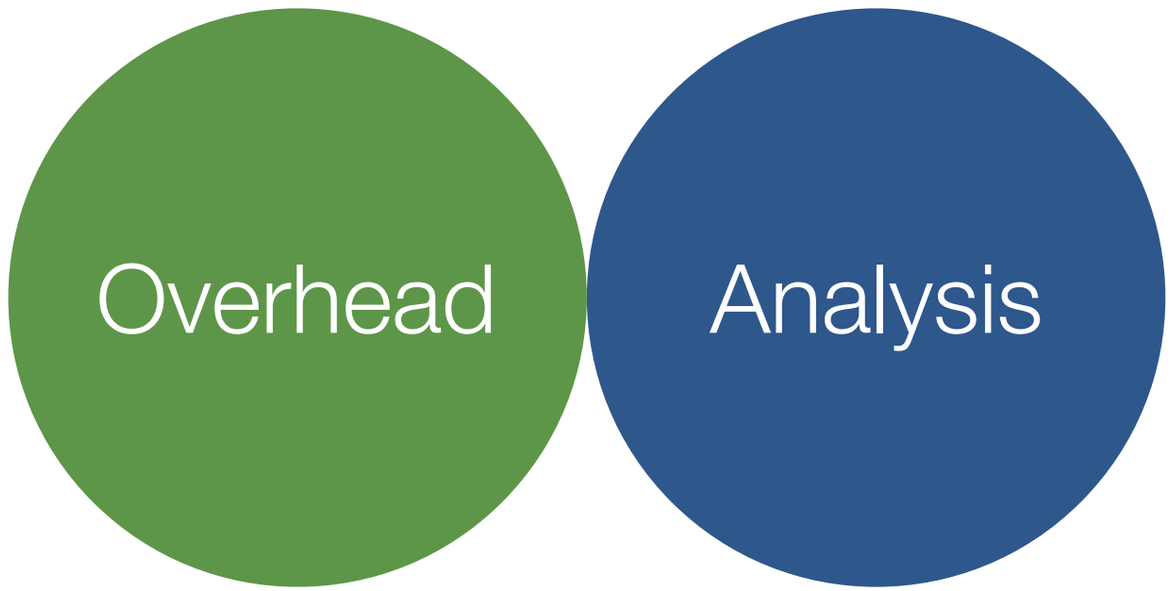
Execution Time

Whole-Program



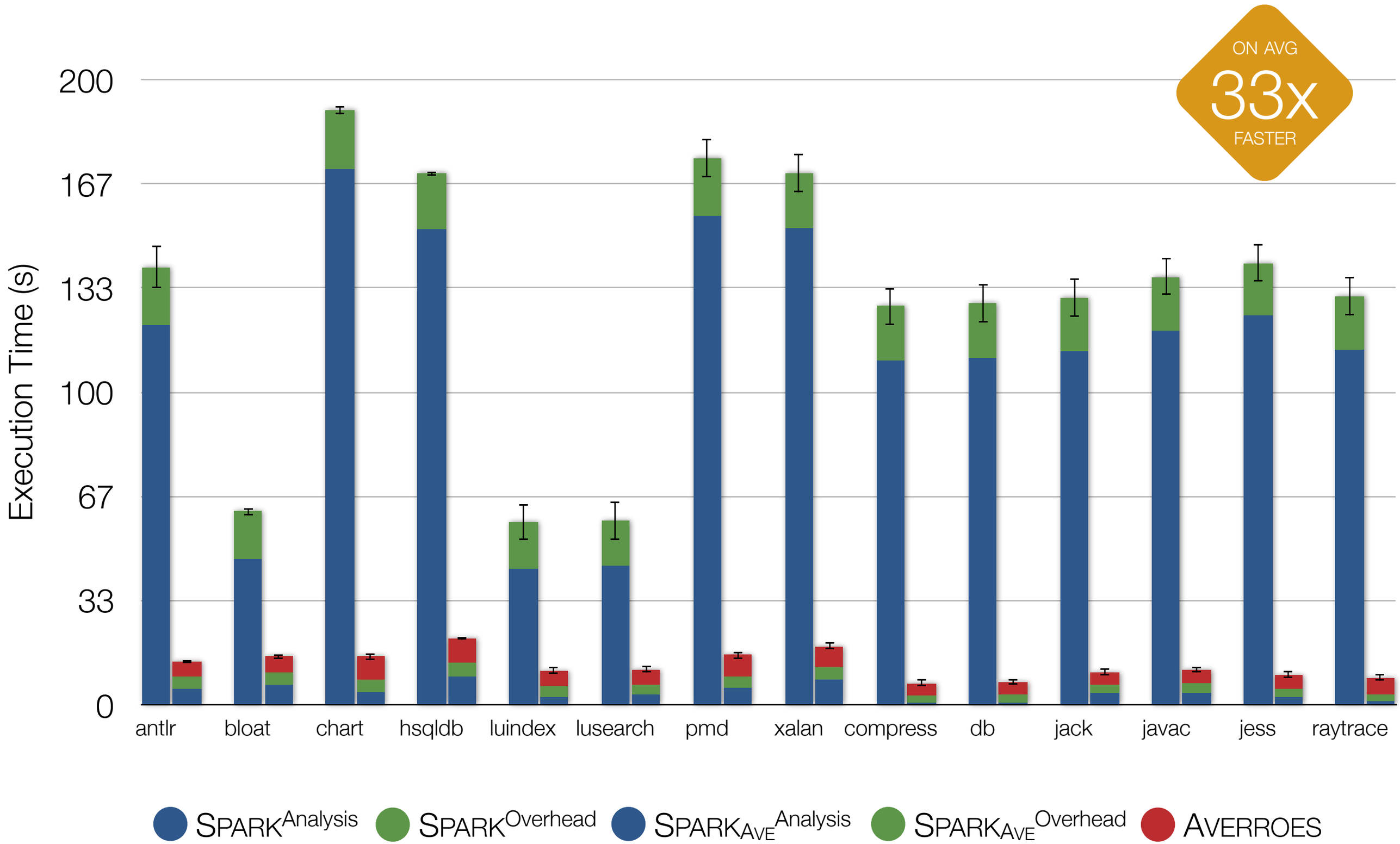
Execution Time

Whole-Program

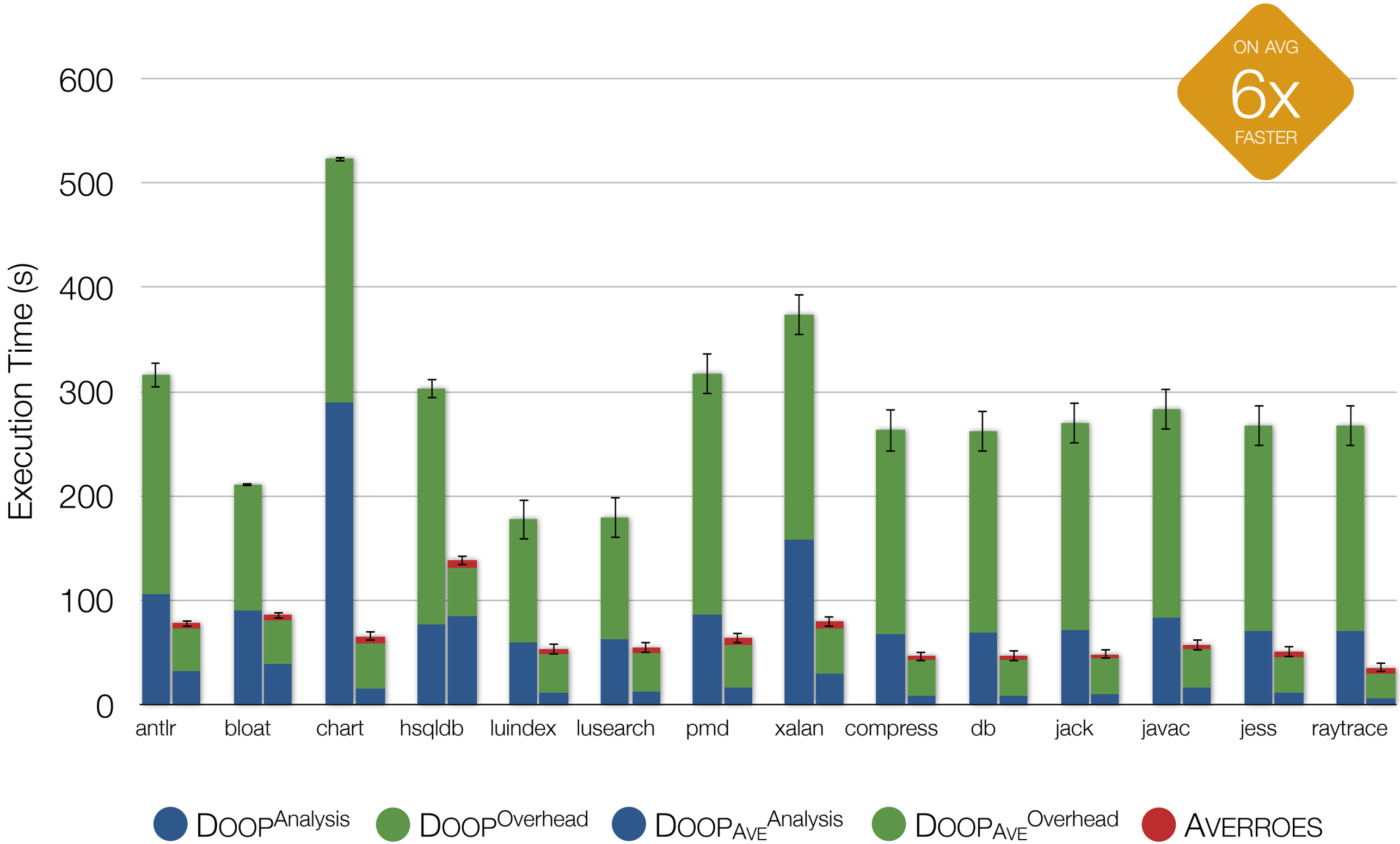


Averroes

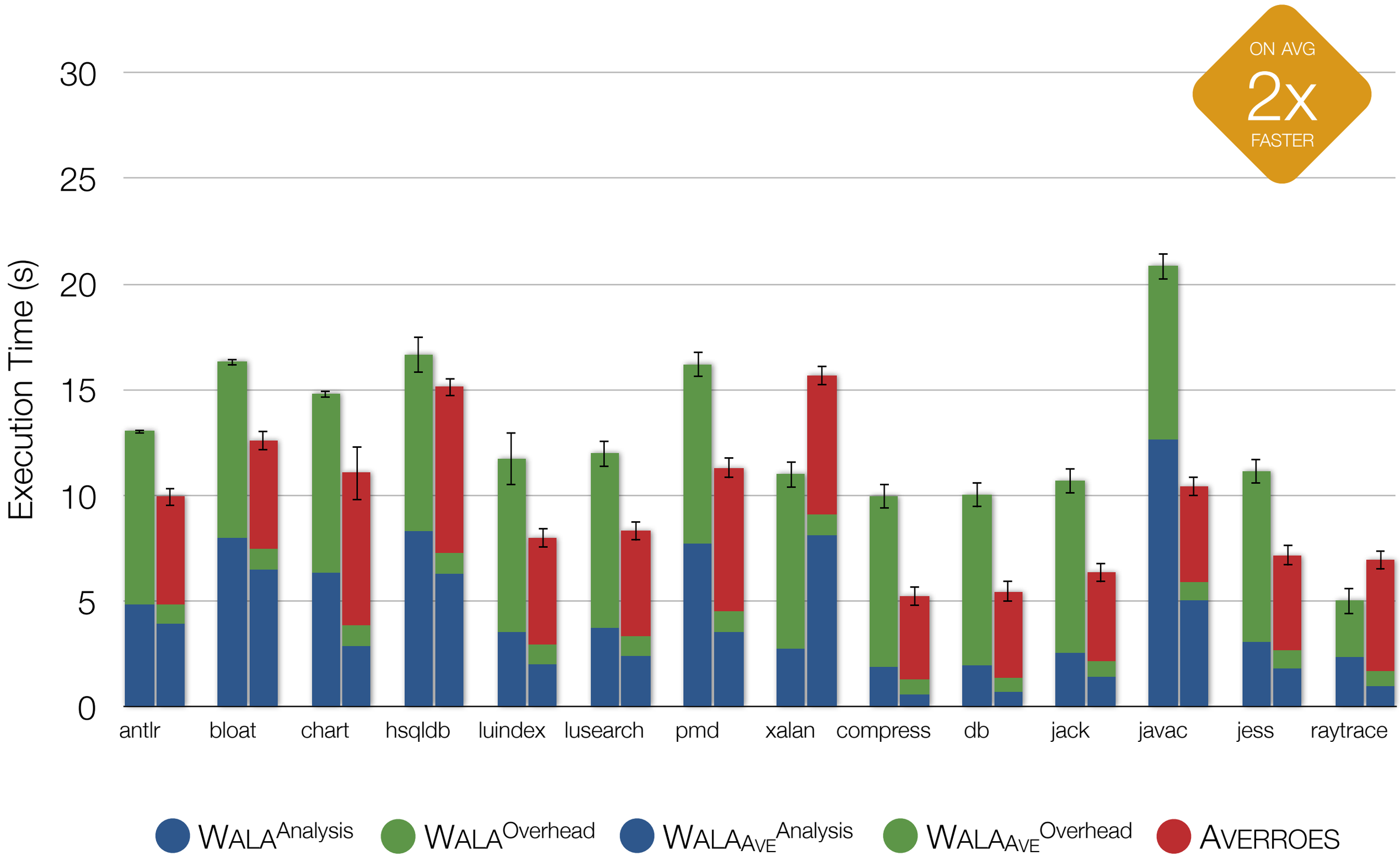
Execution Time - SPARK



Execution Time - DOOP

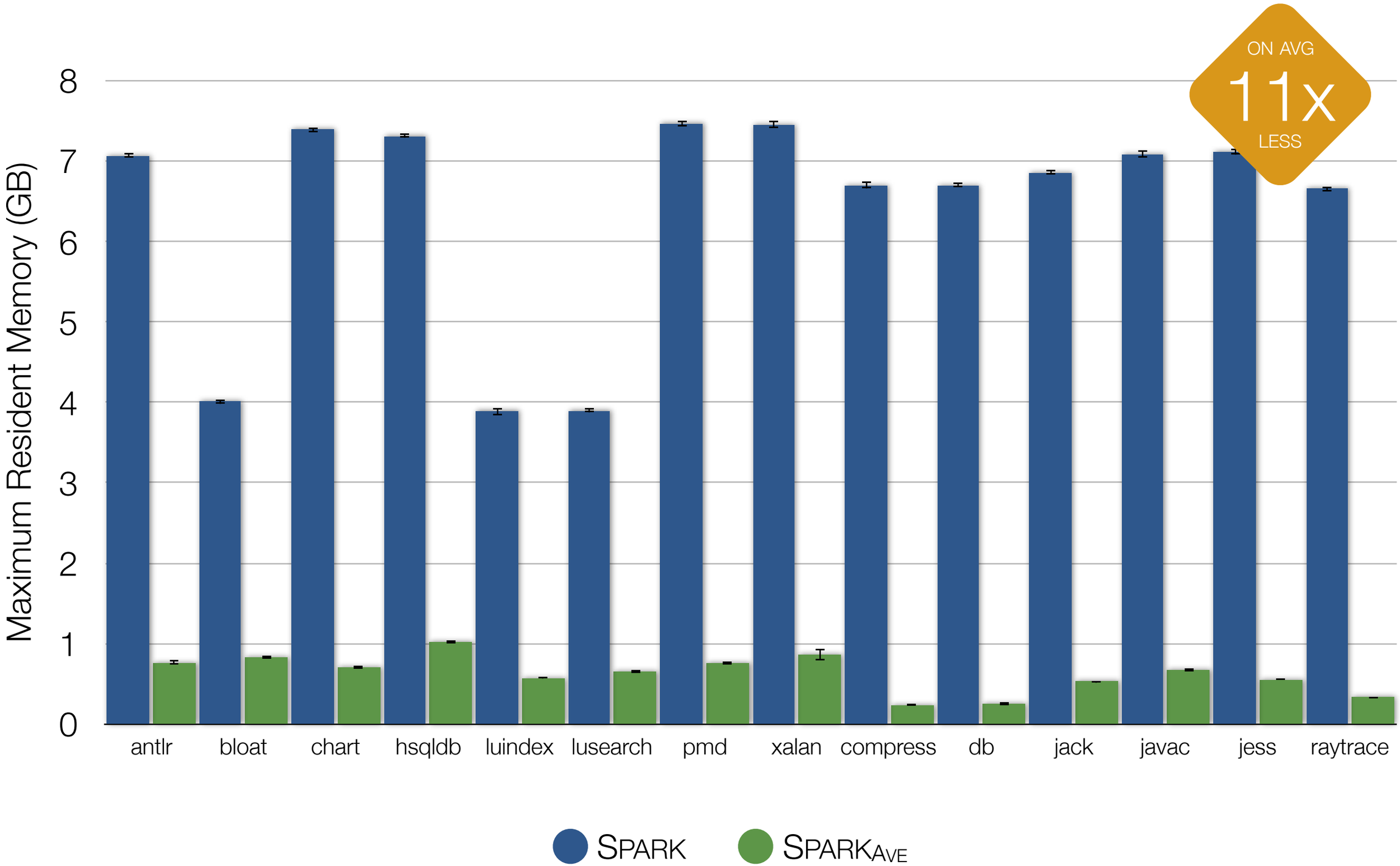


Execution Time - WALA

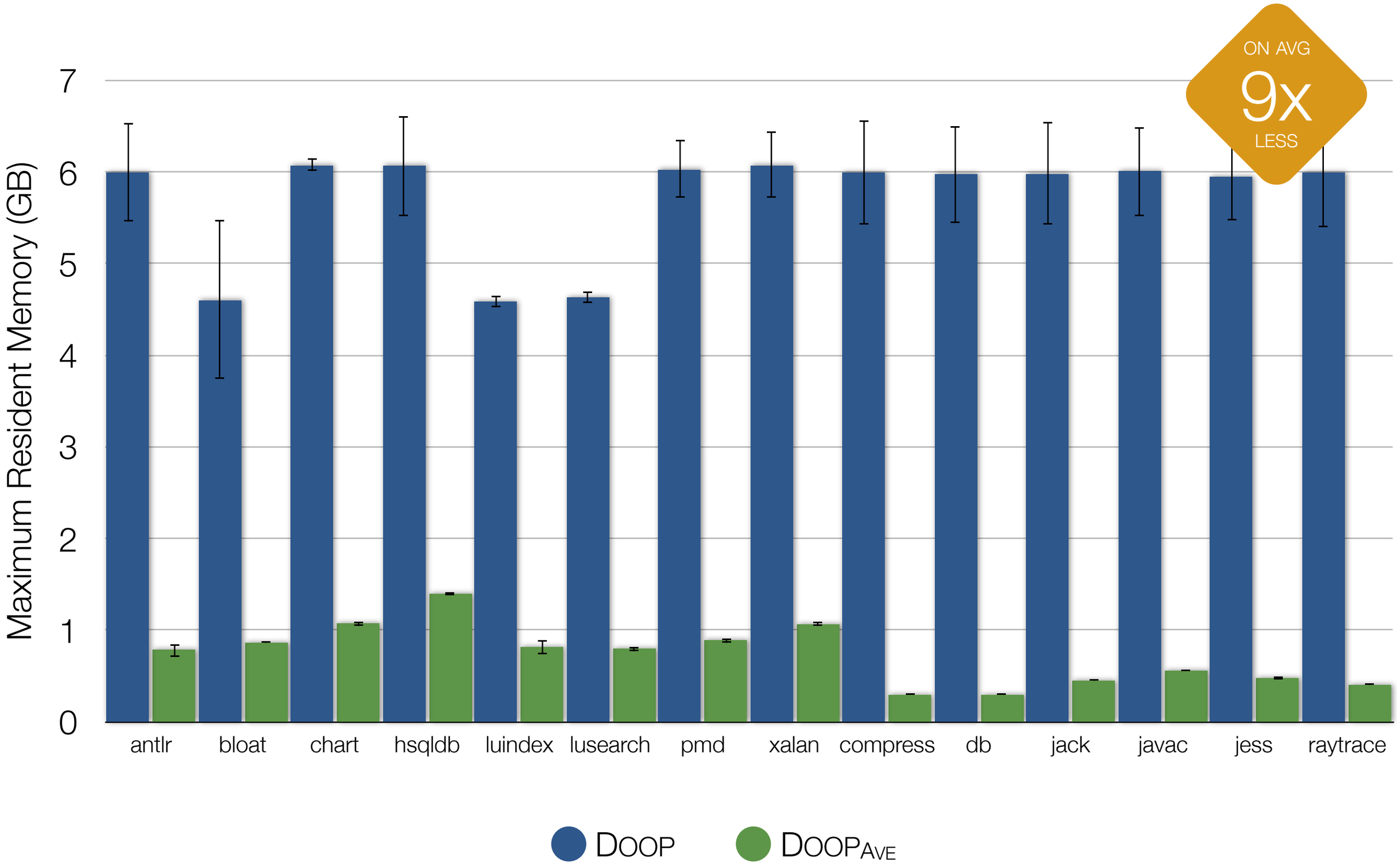


Memory Usage

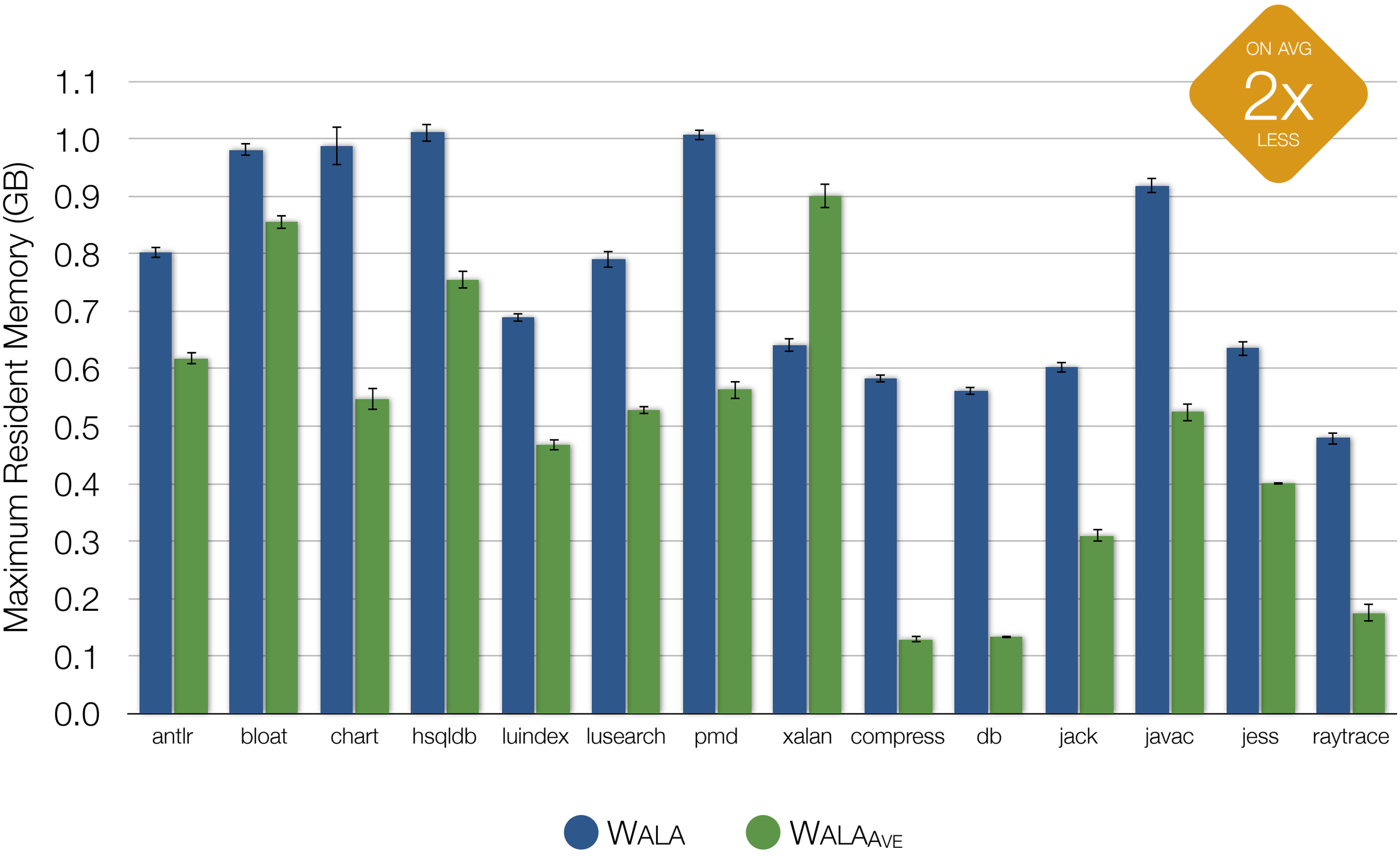
Memory Usage - SPARK



Memory Usage - DOOP



Memory Usage - WALA



Sound & Precise

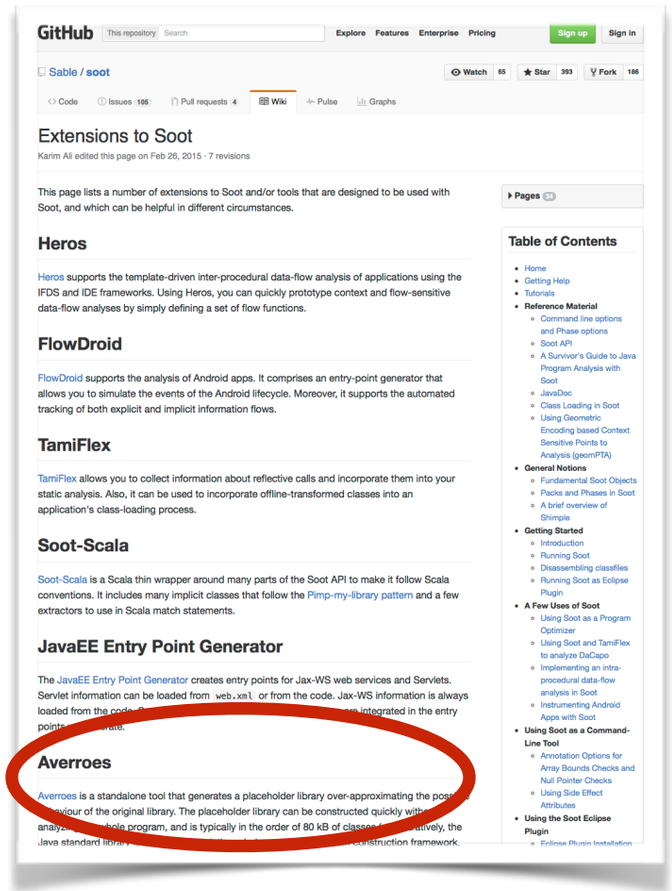
Karim Ali. The Separate Compilation Assumption. PhD Thesis, University of Waterloo. 2014.
http://hdl.handle.net/10012/8835

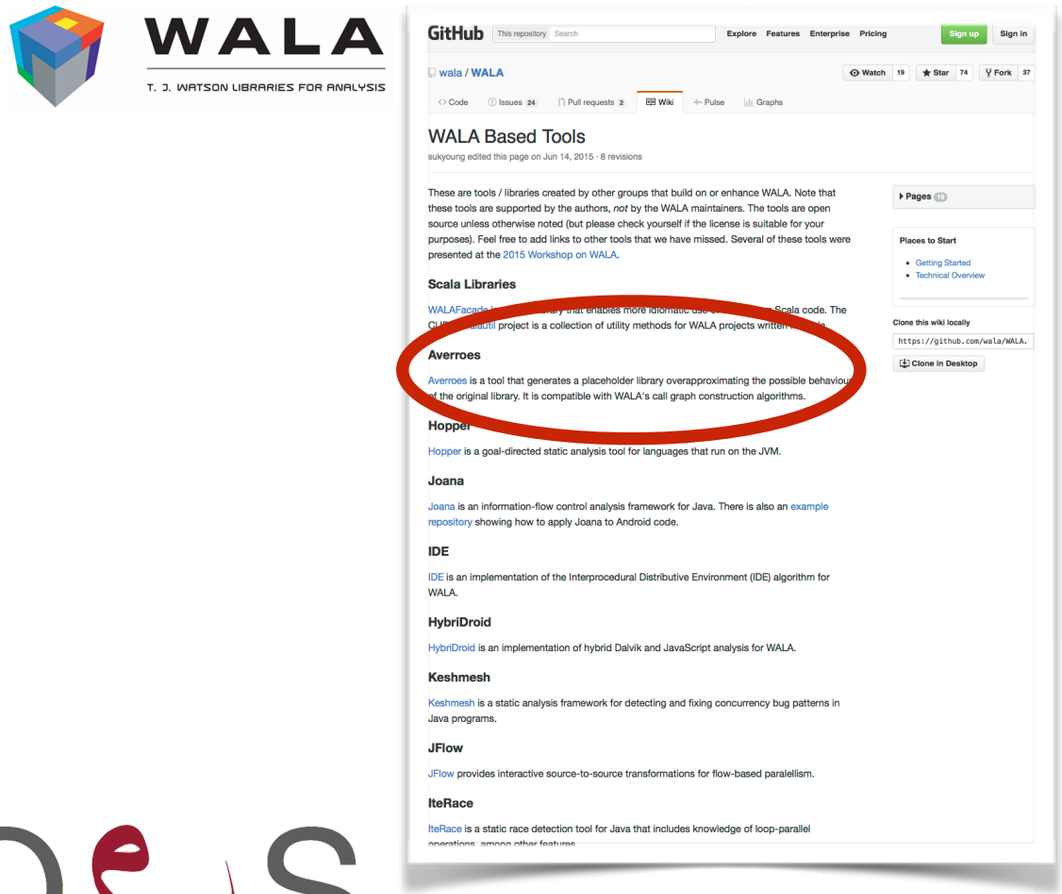
600x smaller library

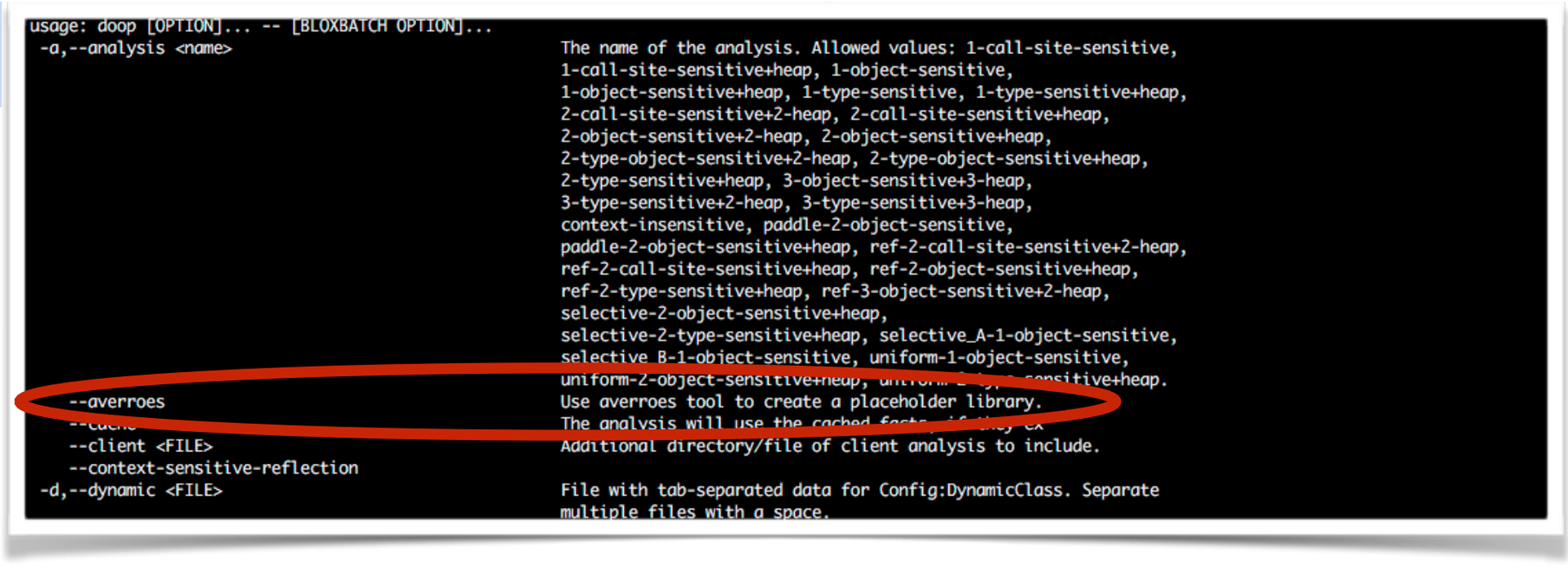
Up to 33x faster analysis

Up to 11x less memory

Sound & Precise


GitHub
Sable / soot
Extensions to Soot
Karim Ali edited this page on Feb 26, 2015 · 7 revisions
This page lists a number of extensions to Soot and/or tools that are designed to be used with Soot, and which can be helpful in different circumstances.
Table of Contents
Heros
Heros supports the template-driven inter-procedural data-flow analysis of applications using the IFDS and IDE frameworks. Using Heros, you can quickly prototype context and flow-sensitive data-flow analyses by simply defining a set of flow functions.
FlowDroid
FlowDroid supports the analysis of Android apps. It comprises an entry-point generator that allows you to simulate the events of the Android lifecycle. Moreover, it supports the automated tracking of both explicit and implicit information flows.
TamiFlex
TamiFlex allows you to collect information about reflective calls and incorporate them into your static analysis. Also, it can be used to incorporate offline-transformed classes into an application's class-loading process.
Soot-Scala
Soot-Scala is a Scala thin wrapper around many parts of the Soot API to make it follow Scala conventions. It includes many implicit classes that follow the Pimp-my-library pattern and a few extractors to use in Scala match statements.
JavaEE Entry Point Generator
The JavaEE Entry Point Generator creates entry points for Jax-WS web services and Servlets.
Averroes
Averroes is a standalone tool that generates a placeholder library over-approximating the possible behaviour of the original library.


GitHub
wala / WALA
WALA Based Tools
sukyoung edited this page on Jun 14, 2015 · 8 revisions
These are tools / libraries created by other groups that build on or enhance WALA. Note that these tools are supported by the authors, not by the WALA maintainers. The tools are open source unless otherwise noted (but please check yourself if the license is suitable for your purposes). Feel free to add links to other tools that we have missed. Several of these tools were presented at the 2015 Workshop on WALA.
Scala Libraries
Averroes
Averroes is a tool that generates a placeholder library overapproximating the possible behaviour of the original library.
Hopper
Hopper is a goal-directed static analysis tool for languages that run on the JVM.
Joana
Joana is an information-flow control analysis framework for Java. There is also an example repository showing how to apply Joana to Android code.
IDE
IDE is an implementation of the Interprocedural Distributive Environment (IDE) algorithm for WALA.
HybriDroid
HybriDroid is an implementation of hybrid Dalvik and JavaScript analysis for WALA.
Keshmesh
Keshmesh is a static analysis framework for detecting and fixing concurrency bug patterns in Java programs.
JFlow
JFlow provides interactive source-to-source transformations for flow-based paralellism.
IteRace
IteRace is a static race detection tool for Java that includes knowledge of loop-parallel
Places to Start
Getting Started
Technical Overview
Clone this wiki locally
https://github.com/wala/WALA.
Clone in Desktop

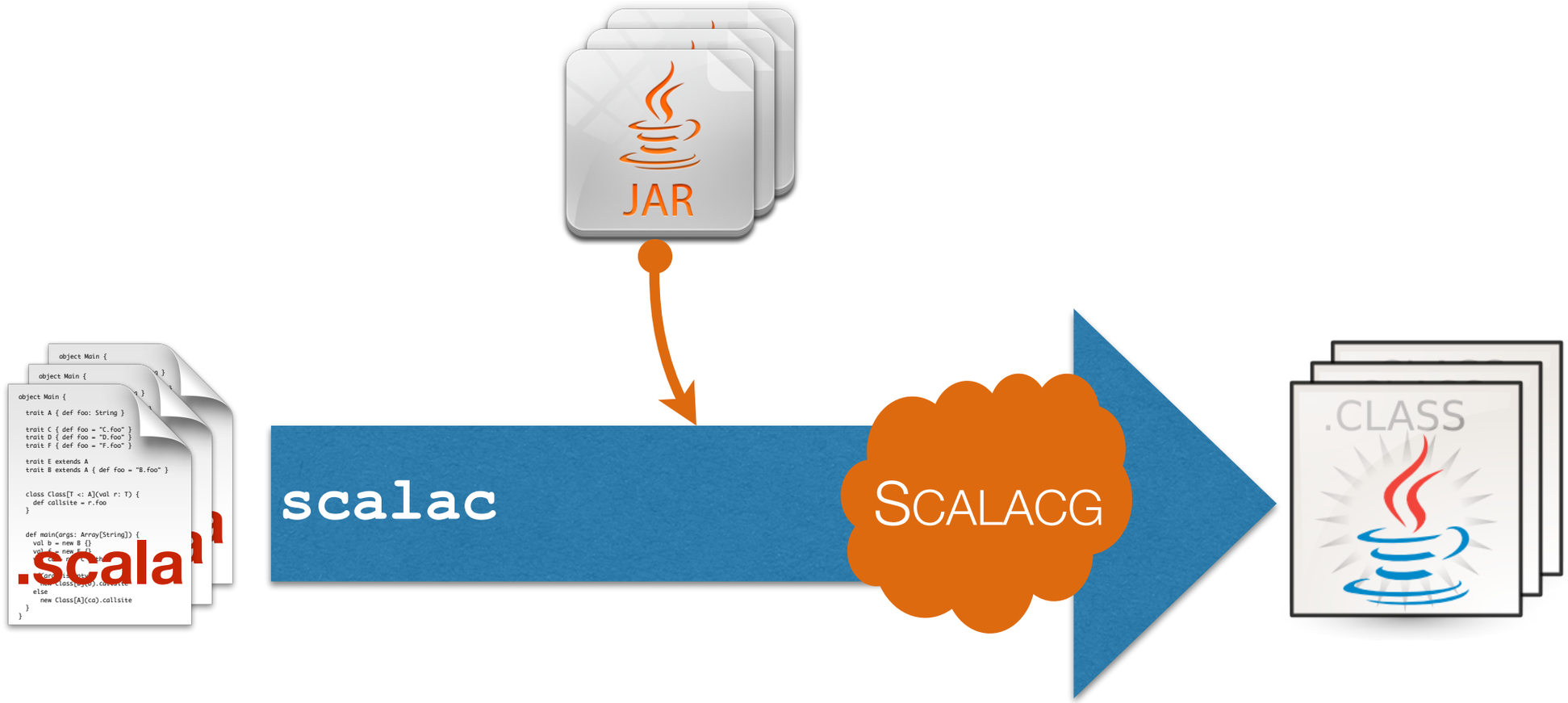

usage: doop [OPTION]... -- [BLOXBATCH OPTION]...
-a,--analysis <name>
The name of the analysis. Allowed values: 1-call-site-sensitive,
1-call-site-sensitive+heap, 1-object-sensitive,
1-object-sensitive+heap, 1-type-sensitive, 1-type-sensitive+heap,
2-call-site-sensitive+2-heap, 2-call-site-sensitive+heap,
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,
--averroes
Use averroes tool to create a placeholder library.
--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.

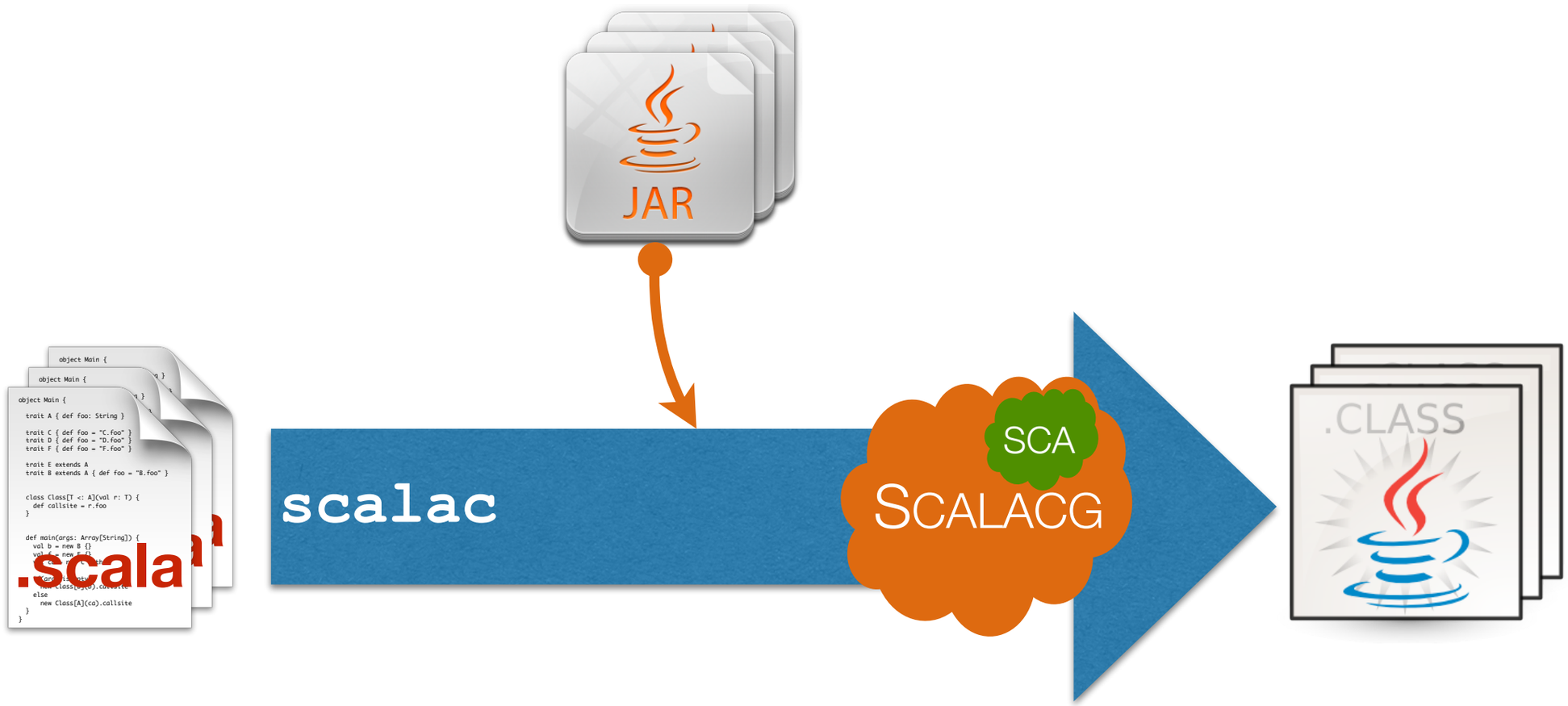
… but do we need such tools in practice?

Karim Ali, Marianna Rapoport, Ondřej Lhoták, Julian Dolby, and Frank Tip. 2015. Type-Based Call Graph Construction Algorithms for Scala. ACM Transactions in Software Engineering and Methodologies 25, 1, Article 9 (December 2015), 43 pages.

Karim Ali, Marianna Rapoport, Ondřej Lhoták, Julian Dolby, and Frank Tip. 2014. Constructing Call Graphs of Scala Programs. In Proceedings of the 28th European Conference on Object-Oriented Programming (ECOOP'14), 54-79.

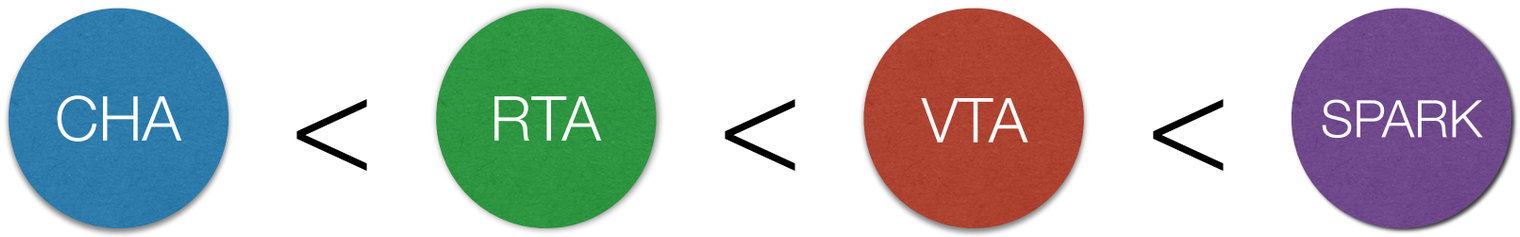

.scala
scalac
SCALACG
.CLASS


JAR
.scala
scalac
SCALACG
.CLASS


JAR
.scala
scalac
SCA
SCALACG
.CLASS

AVERROES

https://github.com/karimhamdanali/averroes


CHA
<
RTA
<
VTA
<
SPARK

@karim3ali

Whole-Program Call Graph

Application Code

+

Library Code

@karim3ali

Partial-Program Call Graphs

CHA < RTA < VTA < SPARK

@karim3ali

Whole-Program Call Graph

+

Partial-Program Call Graph

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

CHA < RTA < VTA < SPARK

@karim3ali

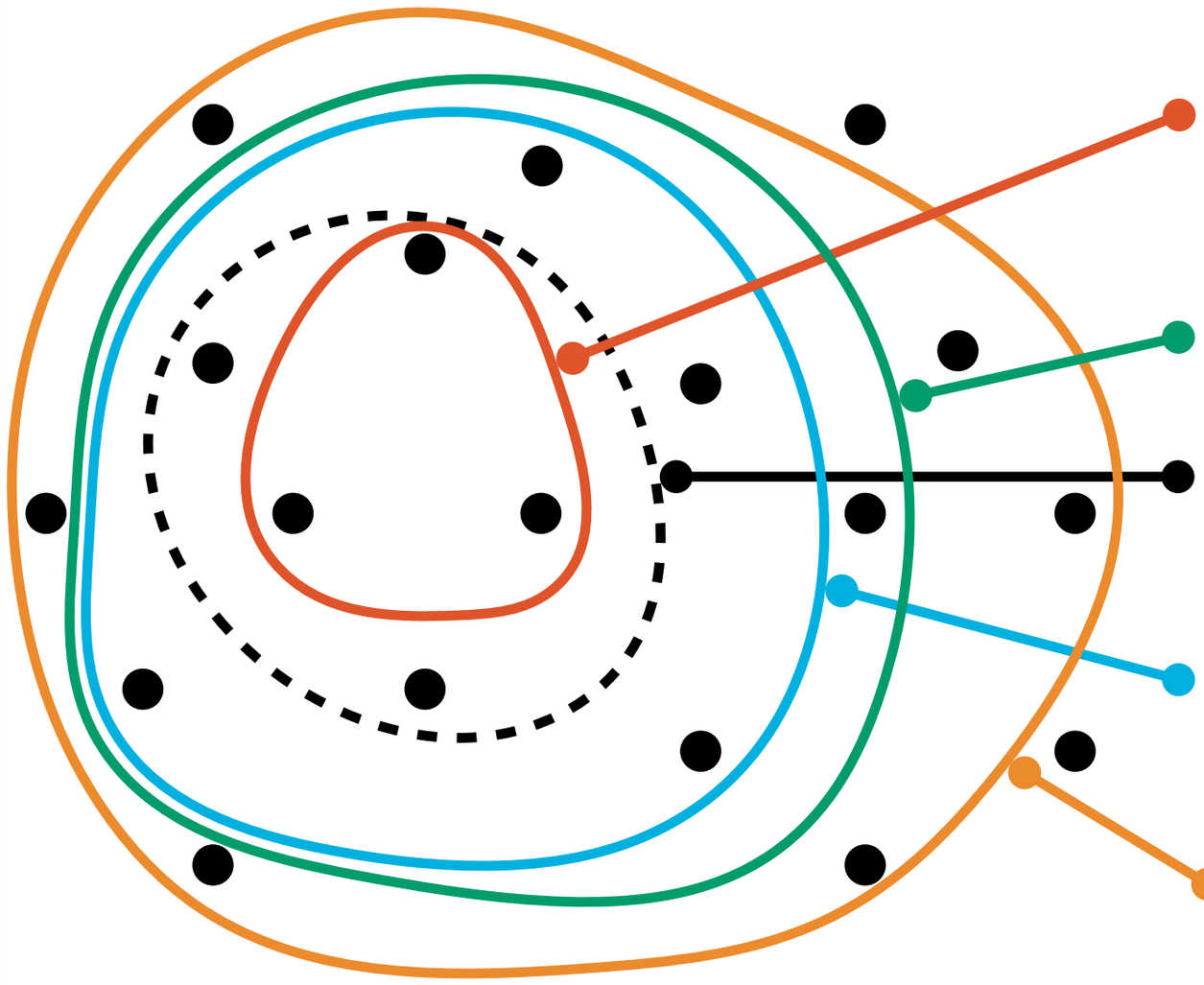
Whole-Program Call Graph

Partial-Program Call Graph

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

on program
m libraries)?



Incomplete Call Graph (unsound)

Partial-Program Call Graph

Ideal Call Graph

Whole-Program Call Graph

Conservative Call Graph (highly imprecise)

CHA < RTA < VTA < SPARK

@karim3ali

Whole-Program Call Graph

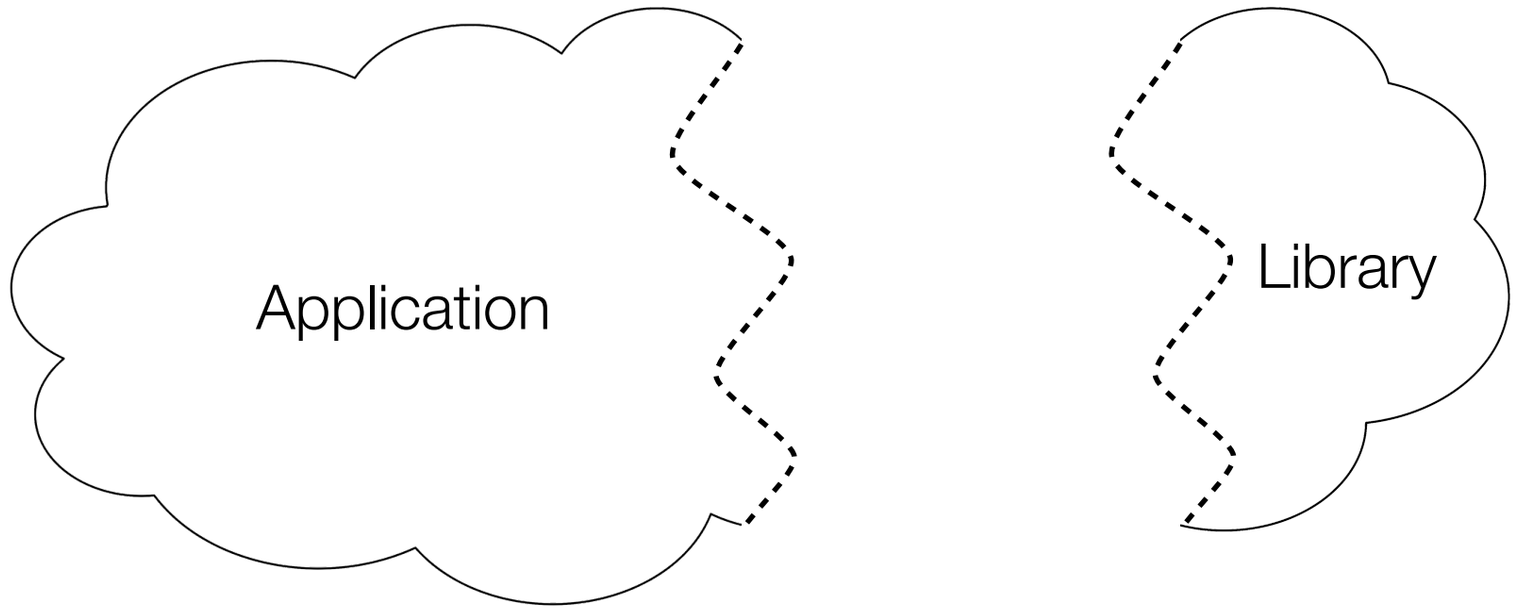
Partial-Program Call Graph

I'd like to ignore library code

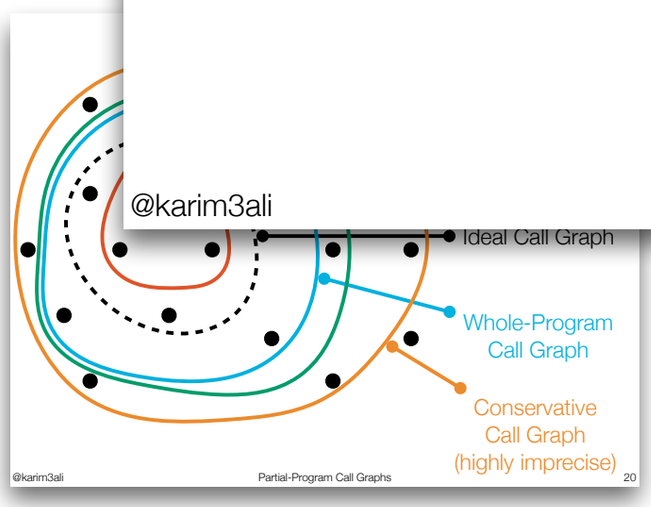
whole-program analysis always pulls in the world for completeness. The problem is that the world is fairly large

on program
m libraries)?

The Separate Compilation Assumption



@karim3ali Partial-Program Call Graphs



CHA < RTA < VTA < SPARK

@karim3ali

Whole-Program Call Graph

Partial-Program Call Graph

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

on program
m libraries)?

600x smaller library

Up to 33x faster analysis

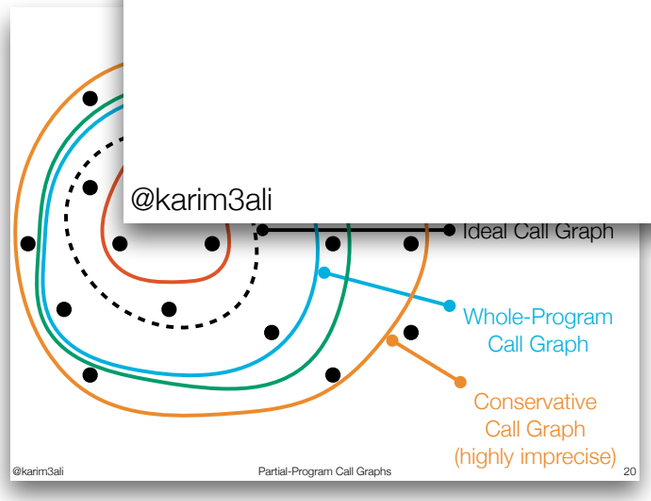
AVERROES

Up to 11x less memory

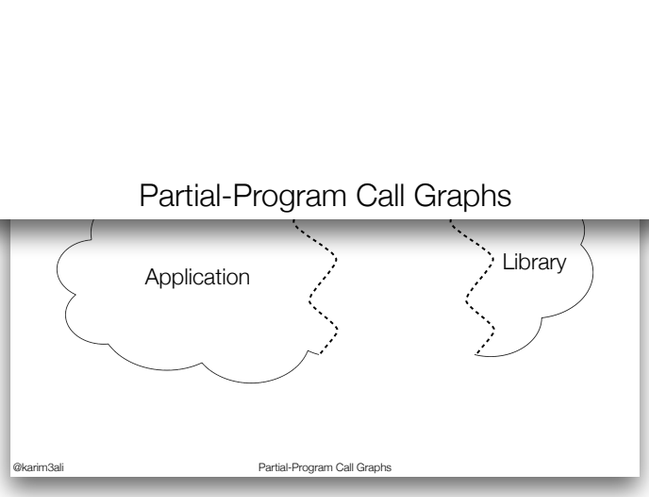
Sound & Precise

@karim3ali

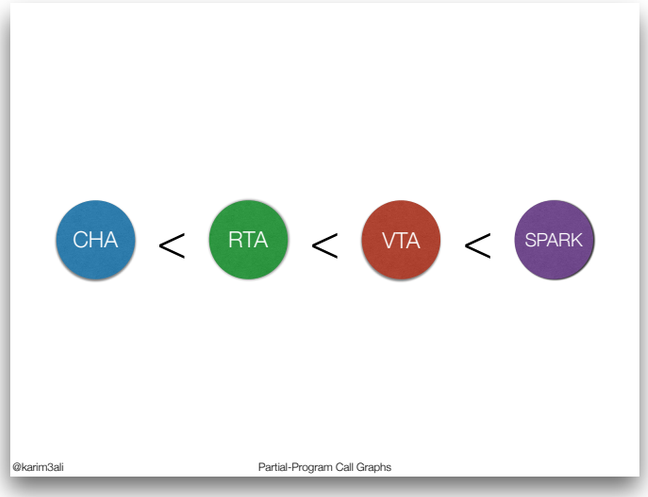
Partial-Program Call Graphs

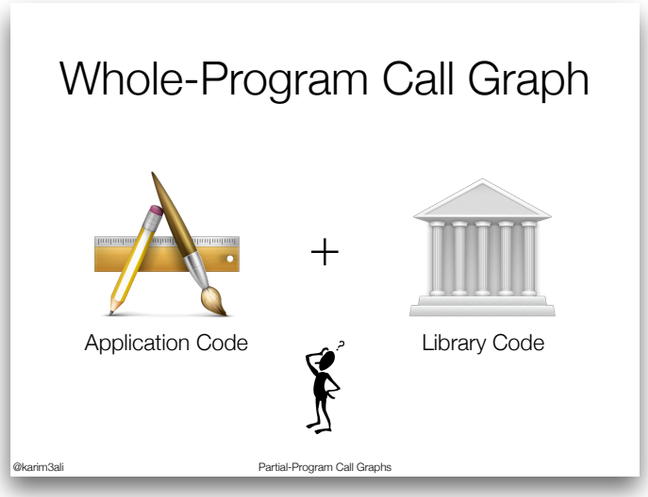


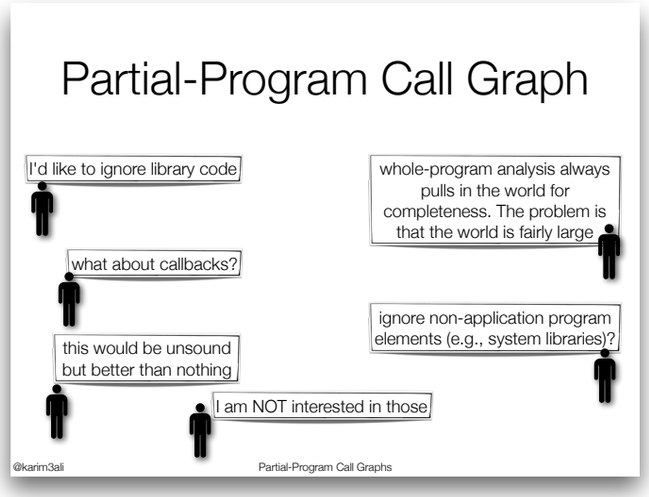
@karim3ali Partial-Program Call Graphs 20



@karim3ali Partial-Program Call Graphs



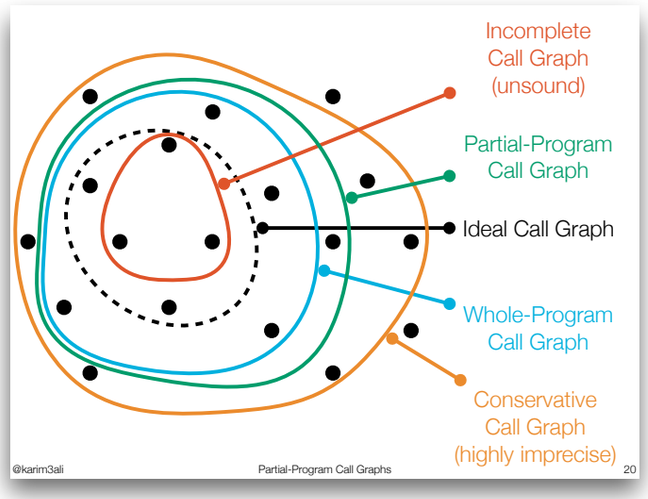


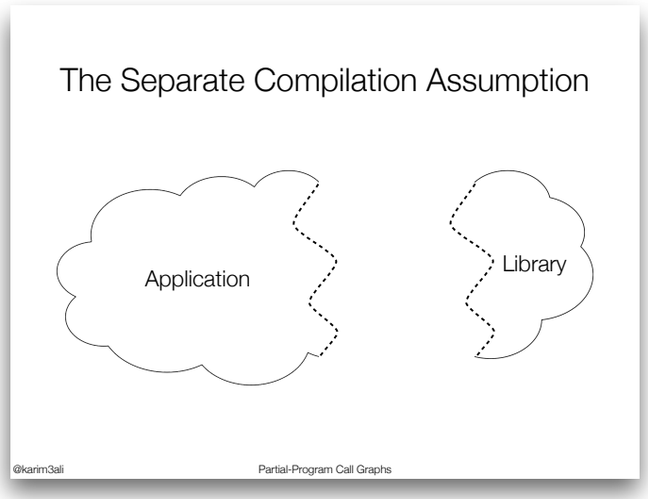


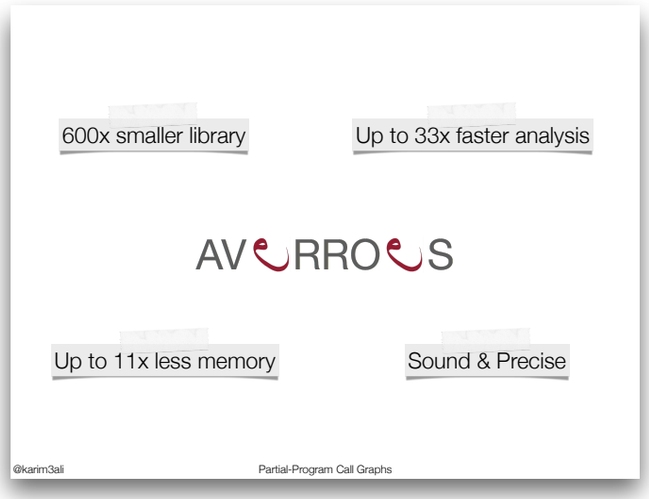
Partial-Program Call Graphs

Applied Static Analysis 2016

Karim Ali
@karim3ali







Practice Time!

Requirements

- Clone the following repo into your Eclipse workspace

  - `git clone` https://github.com/karimhamdanali/averroes

- Do a `git pull` https://github.com/stg-tud/apsa and add the Eclipse project “callgraphs” to your Eclipse setup