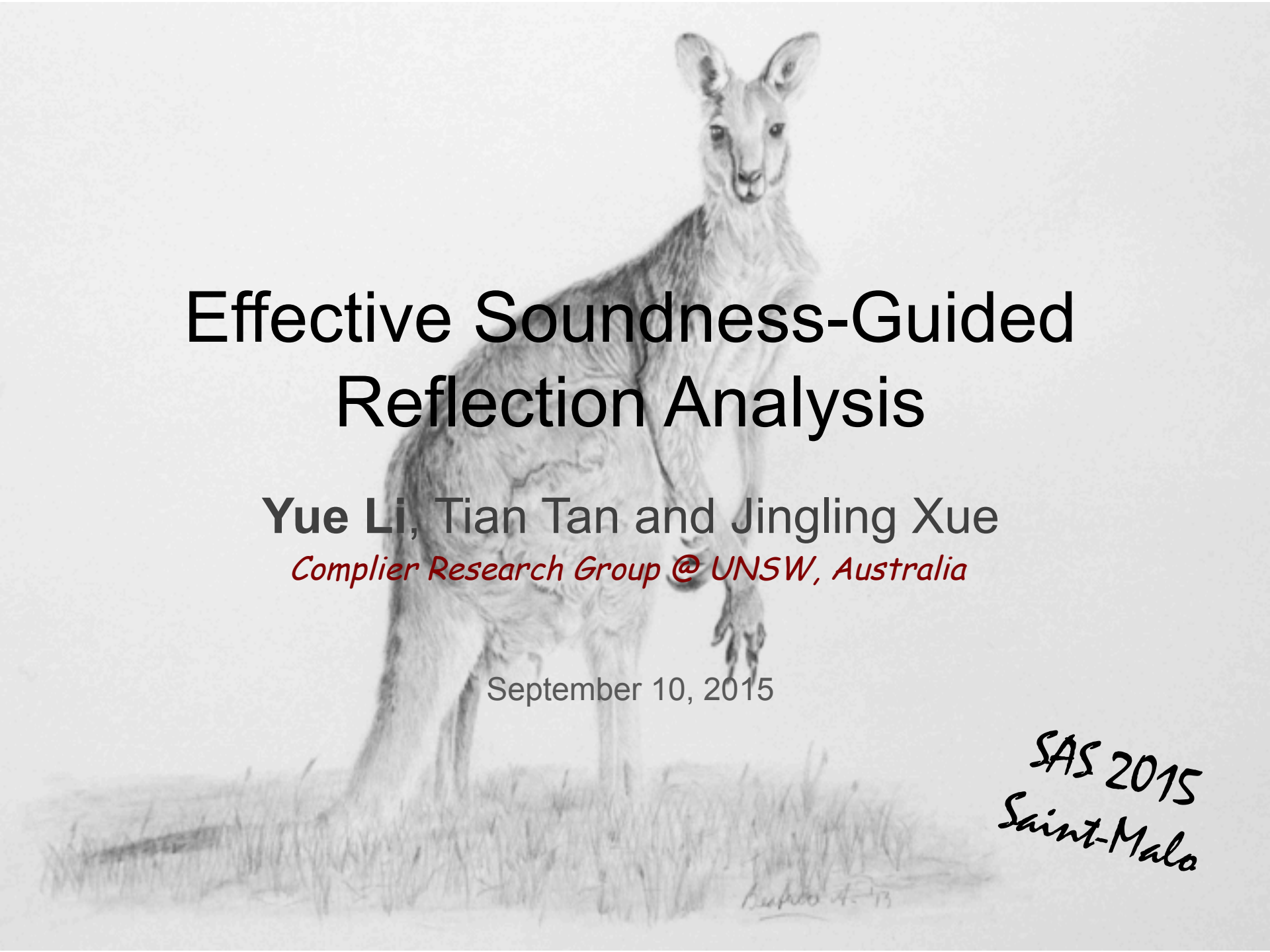


Effective Soundness-Guided Reflection Analysis

Yue Li, Tian Tan and Jingling Xue

Complier Research Group @ UNSW, Australia

September 10, 2015

SAS 2015
Saint-Malo

Static analysis for OO in practice ?

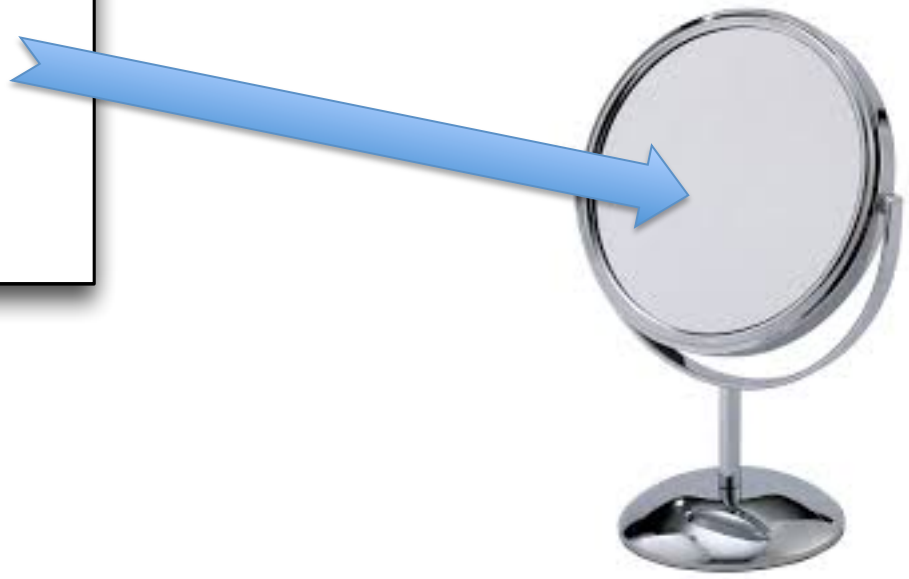
Static analysis for OO in practice ?

re re re … reflection !

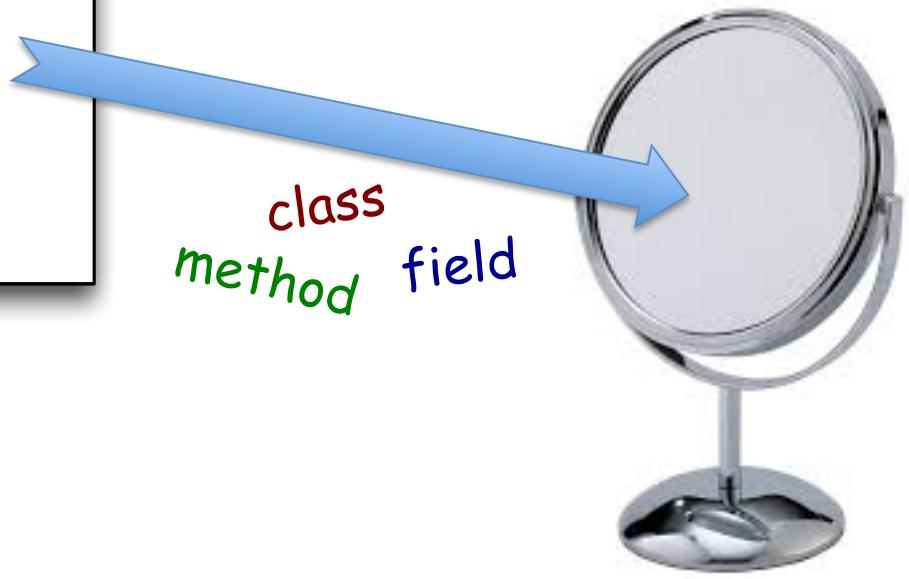
```
Class Person {
    void setName(String nm) {…};
    … …
}

Person p = new Person();
p.setName("John");
```

```
Class Person {
    void setName(String nm) {…};
    … …
}

Person p = new Person();
p.setName("John");
```

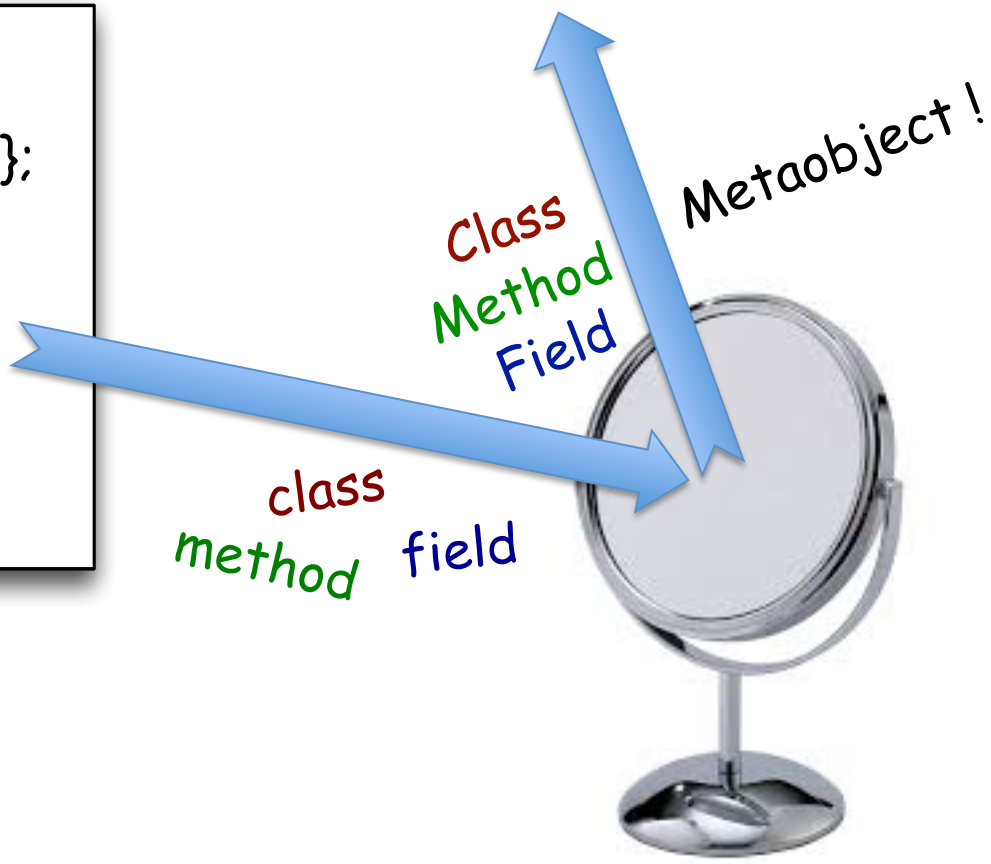
```
Class Person {
    void setName(String nm) {…};
    … …
}

Person p = new Person();
p.setName("John");
```



```
Class Person {
    void setName(String nm) {…};
    … …
}

Person p = new Person();
p.setName("John");
```

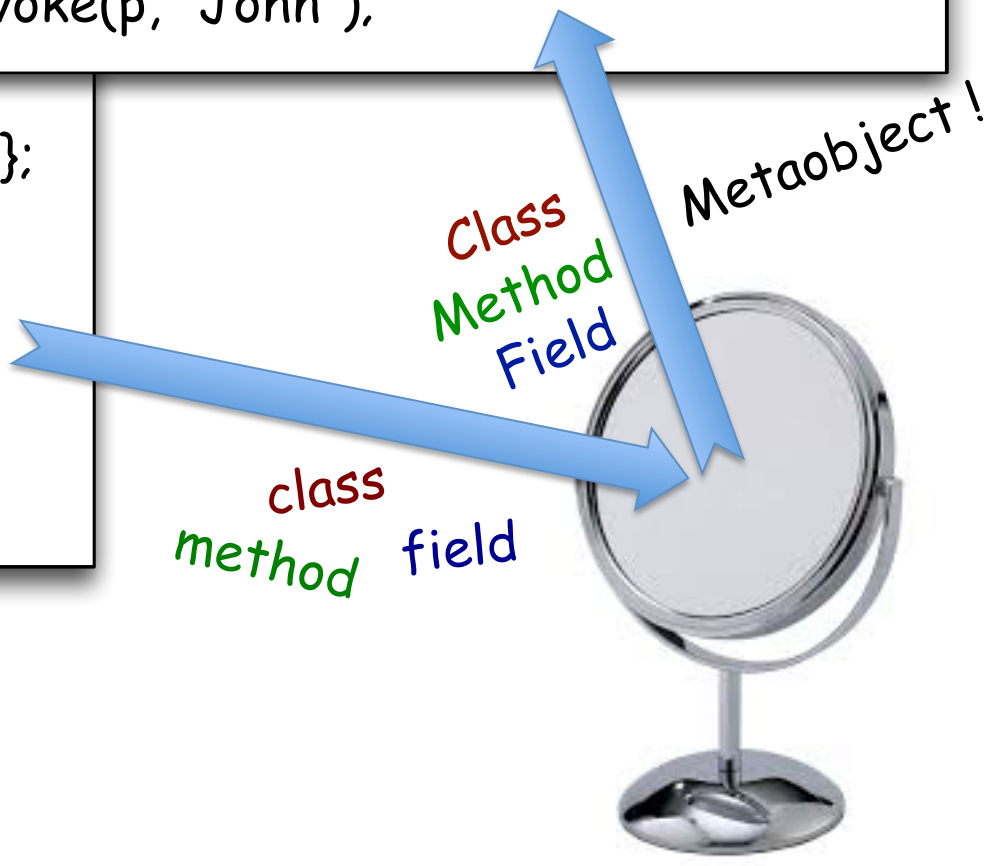


```
Class Person {
    void setName(String nm) {…};
    … …
}

Person p = new Person();
p.setName("John");
```



```
Class c = Class.forName("Person");
Method m = c.getMethod("setName", …);

Object p = c.newInstance();
m.invoke(p, "John");
```

```
Class Person {
    void setName(String nm) {…};
    … …
}

Person p = new Person();
p.setName("John");
```



```
Class c = Class.forName("Person");
Method m = c.getMethod("setName", …);

Object p = c.newInstance();
m.invoke(p, "John");
```

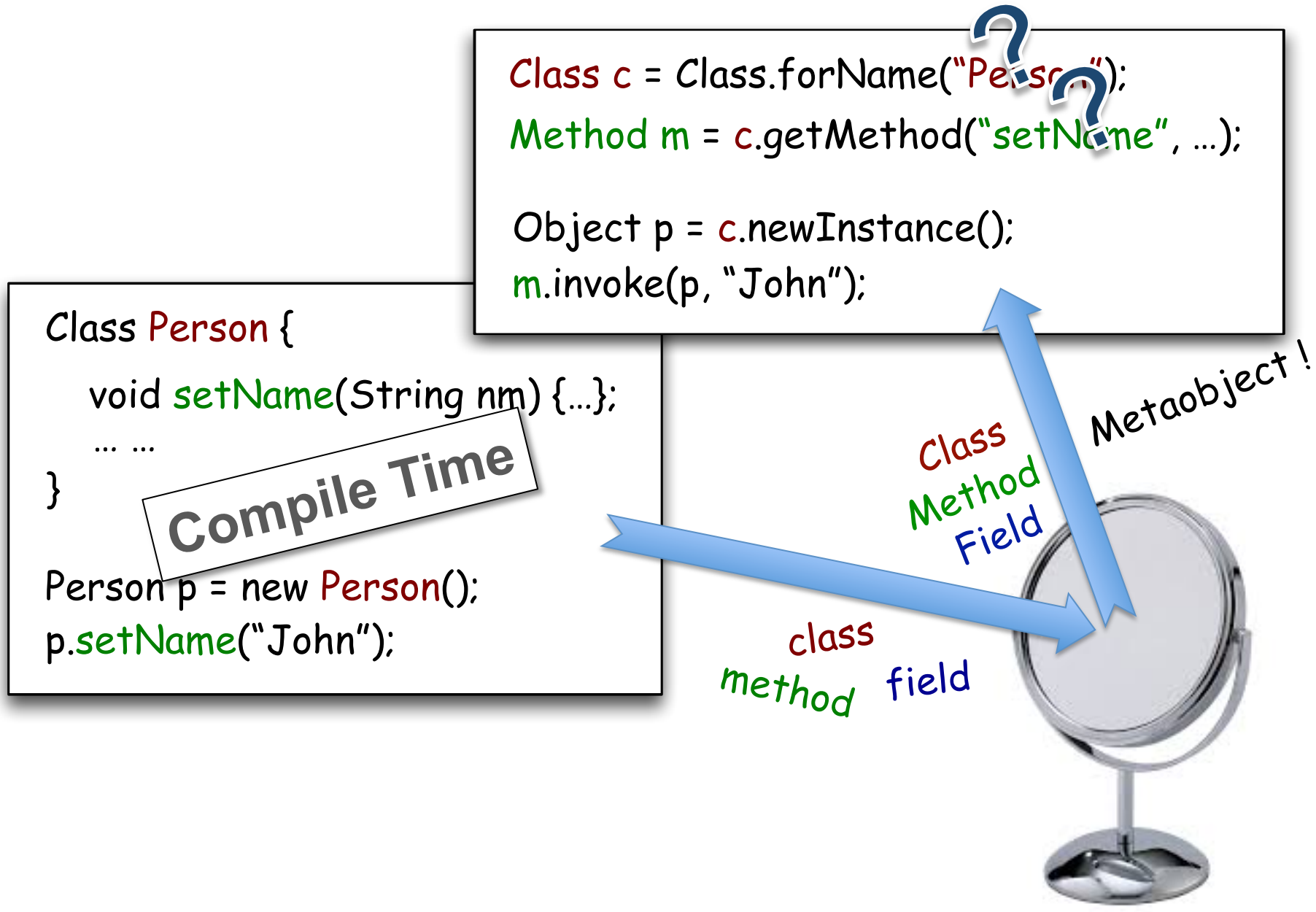
```
Class Person {
    void setName(String nm) {…};
    … …
}

Person p = new Person();
p.setName("John");
```

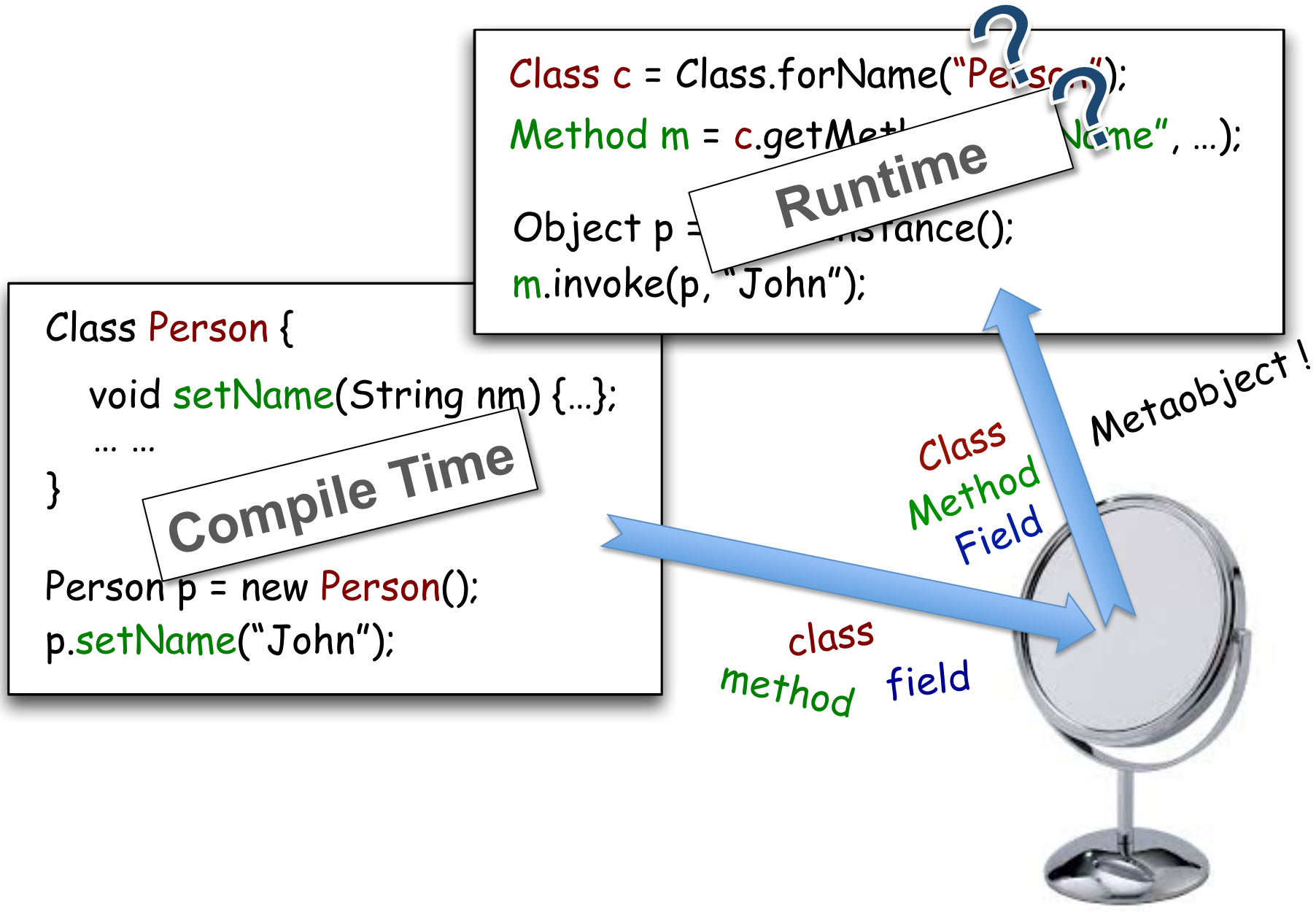
Compile Time

Class
Method
Field

Metaobject !

class
method
field


Class c = Class.forName("Person");
Method m = c.getMethod("setName", …);
Object p = c.newInstance();
m.invoke(p, "John");
Class Person {
void setName(String nm) {…};
… …
}
Compile Time
Person p = new Person();
p.setName("John");
Class
Method
Field
Metaobject !
class
method
field

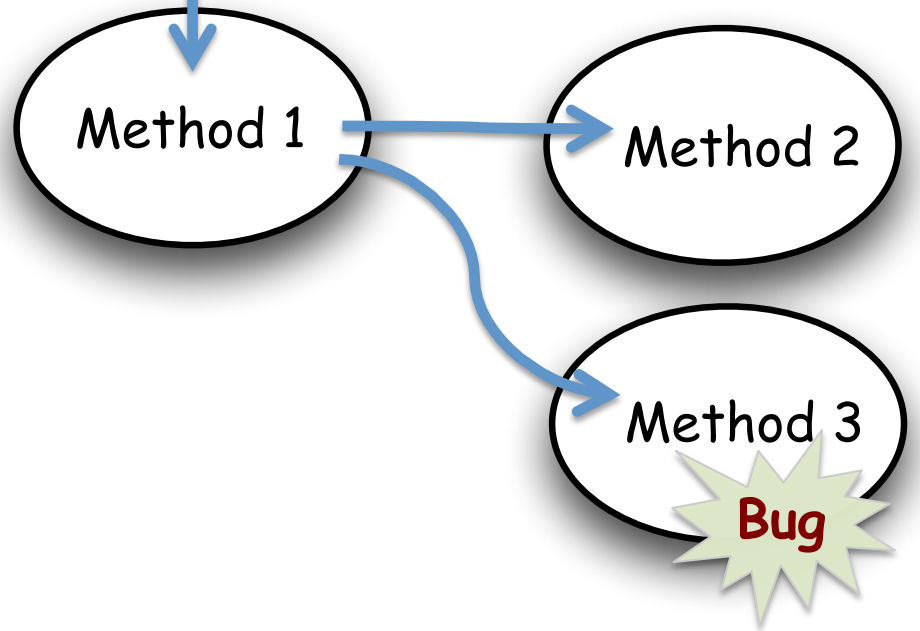

Class c = Class.forName("Person");
Method m = c.getMethod("setName", ...);
Object p = c.newInstance();
m.invoke(p, "John");
Runtime
Class Person {
void setName(String nm) {...};
... ...
}
Compile Time
Person p = new Person();
p.setName("John");
Class
Method
Field
Metaobject !
class
method
field

Bug Detection

```
Class c = Class.forName(cName);
Method m = c.getMethod(mName, …);
A a = new A();
m.invoke(a, …);
```

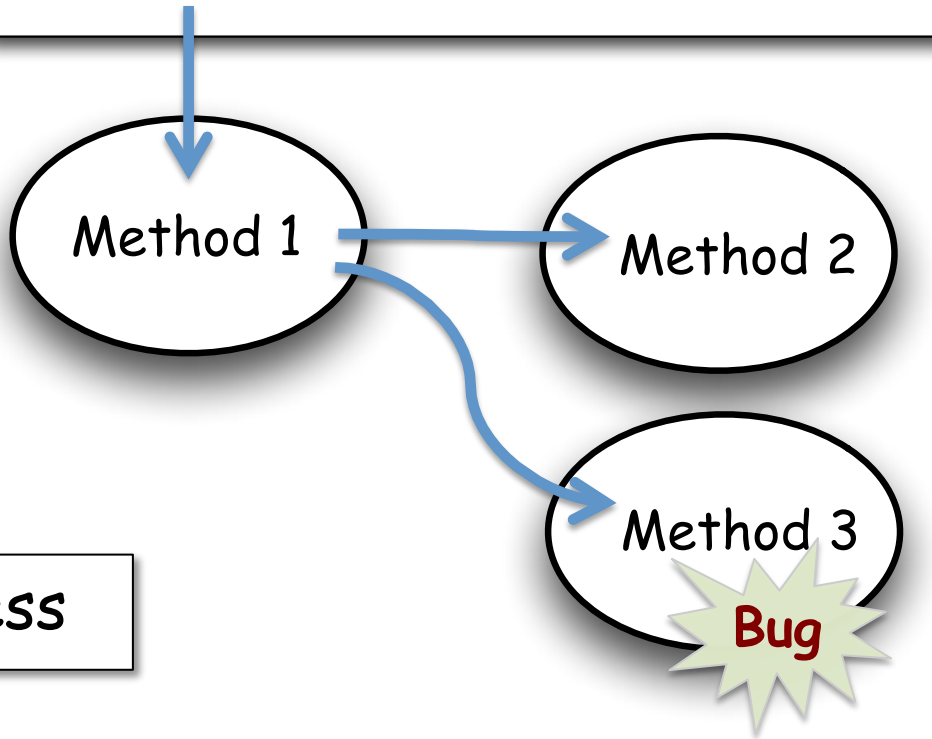
Bug Detection

```
Class c = Class.forName(cName);
Method m = c.getMethod(mName, …);
A a = new A();
m.invoke(a, …);
```



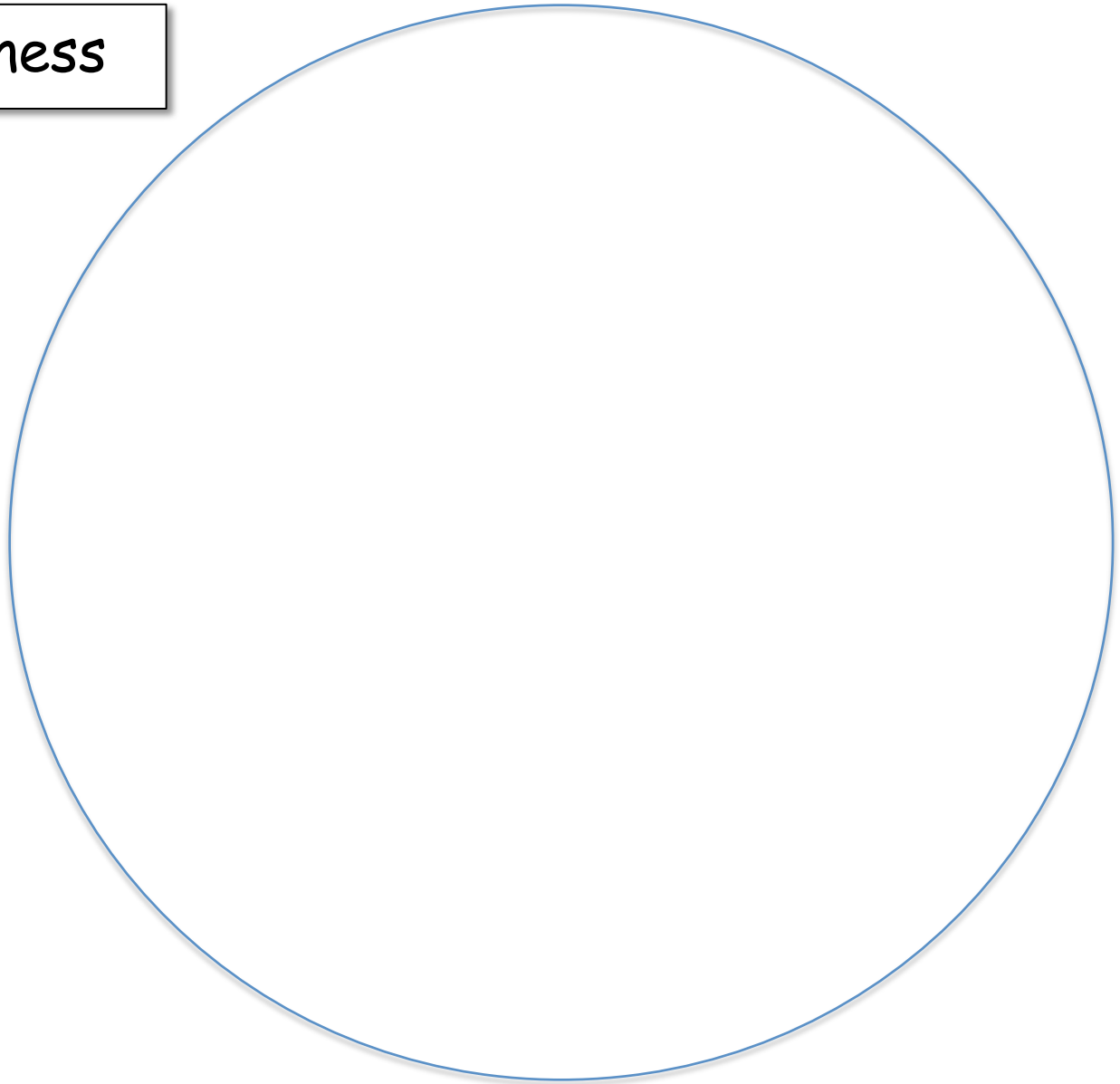
Bug Detection

```
Class c = Class.forName(cName);
Method m = c.getMethod(mName, …);
A a = new A();
m.invoke(a, …);
```

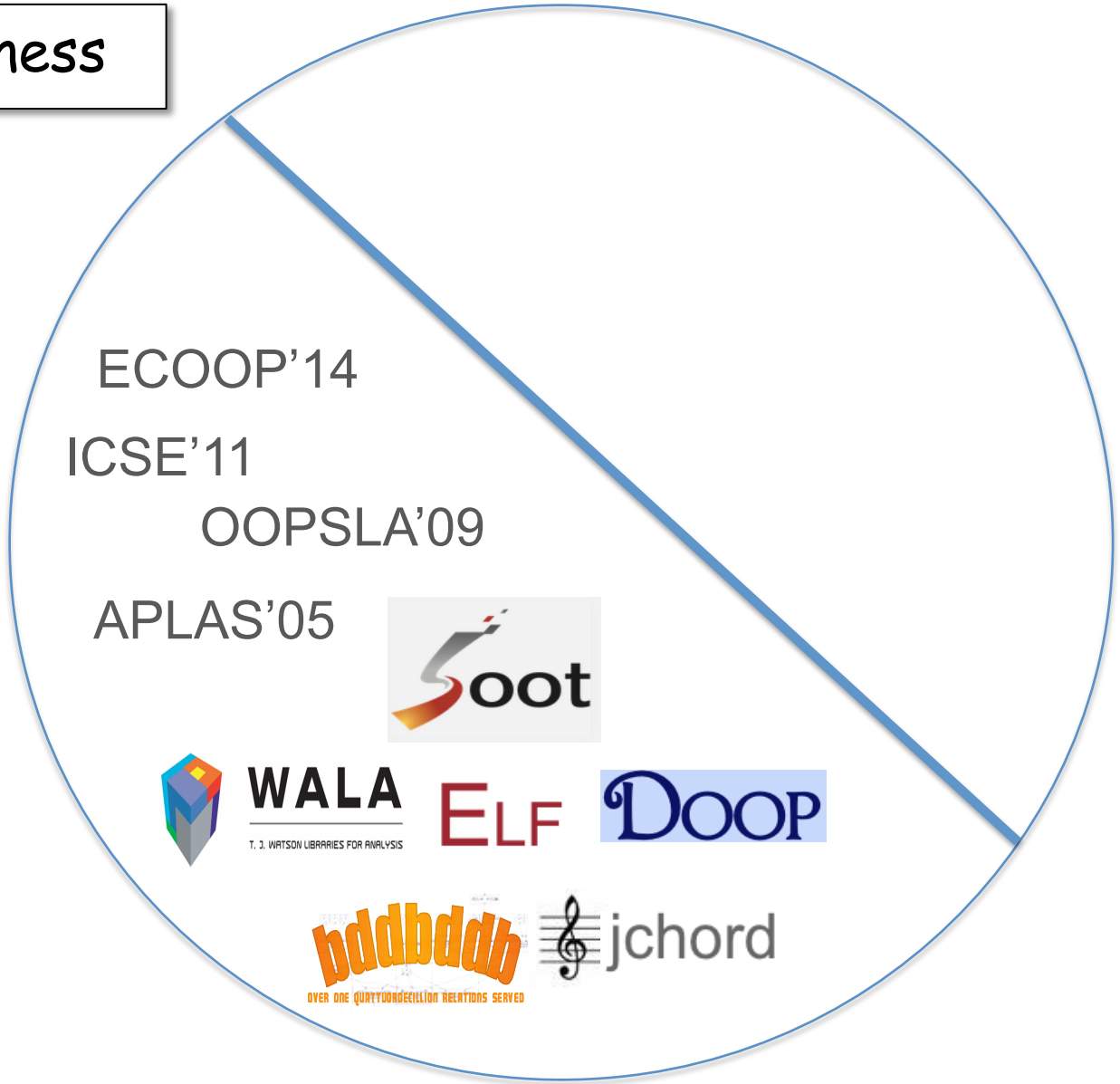


Soundness

Soundness



Soundness


ECOOP'14
ICSE'11
OOPSLA'09
APLAS'05
Soot
WALA
T. J. WATSON LIBRARIES FOR ANALYSIS
ELF
DOOP
bddbddb
OVER ONE QUATTUORDECILLION RELATIONS SERVED
jchord

Soundness

Best-Effort

ECOOP'14

ICSE'11

OOPSLA'09

APLAS'05

WALA

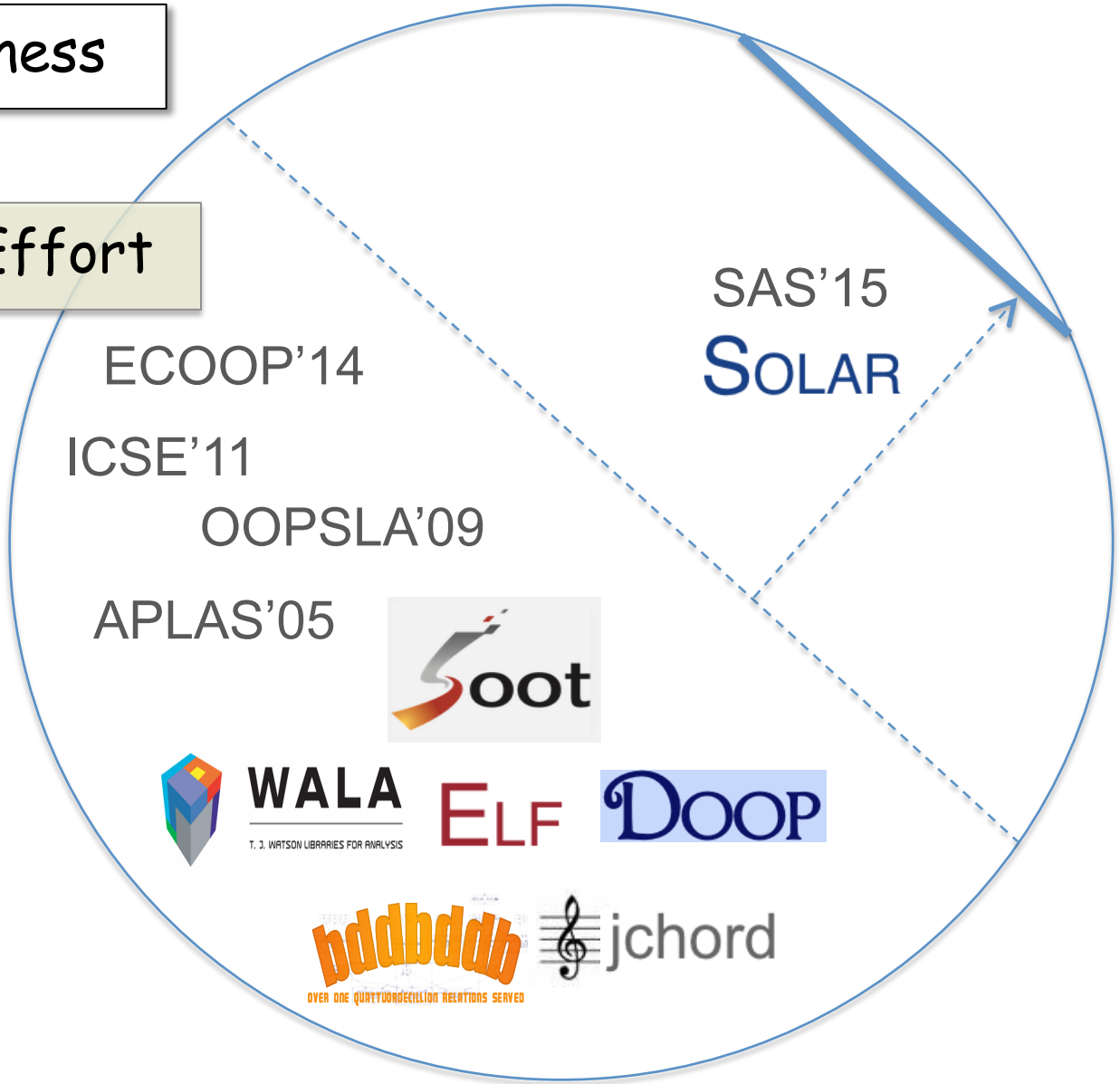
T. J. WATSON LIBRARIES FOR ANALYSIS

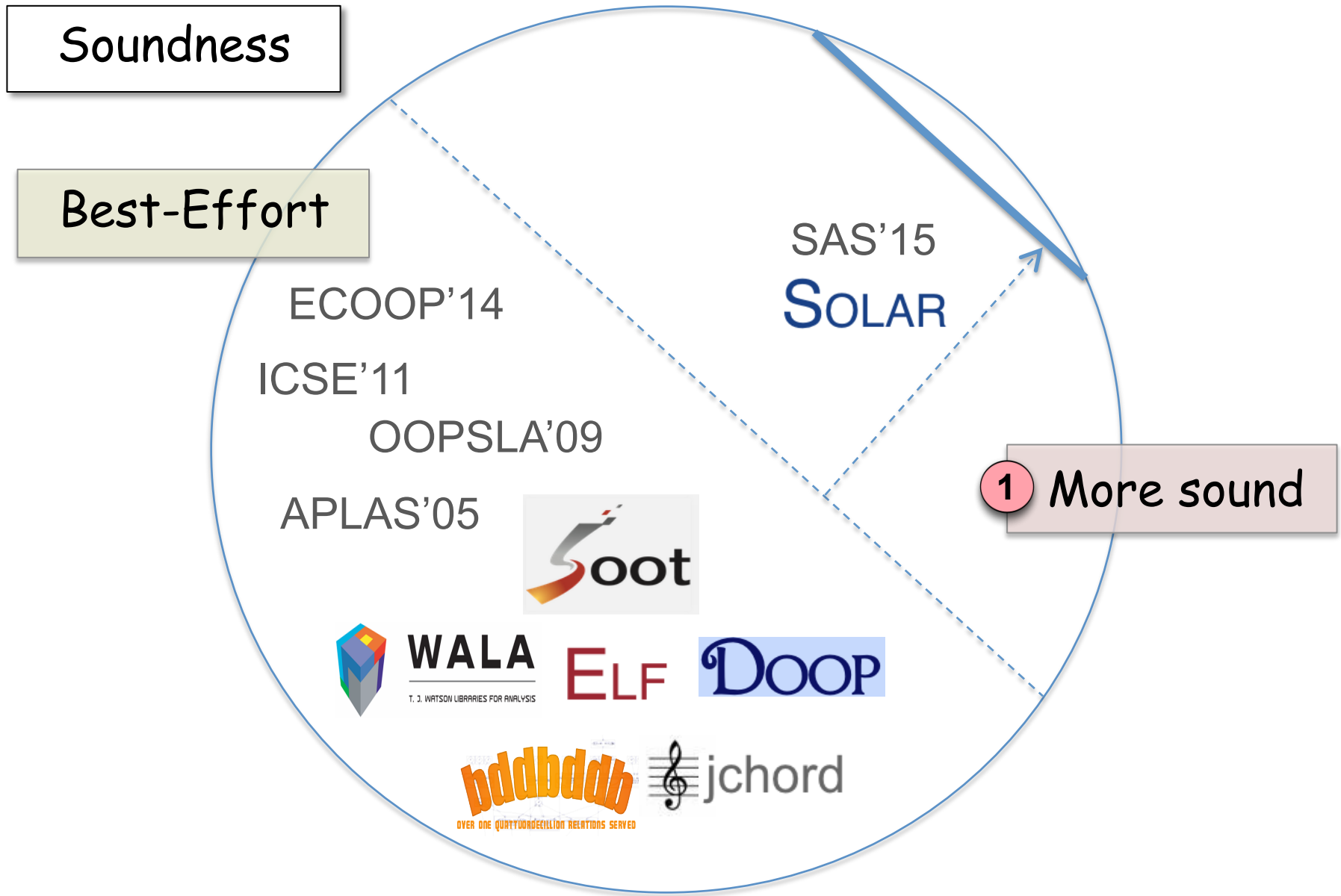
ELF

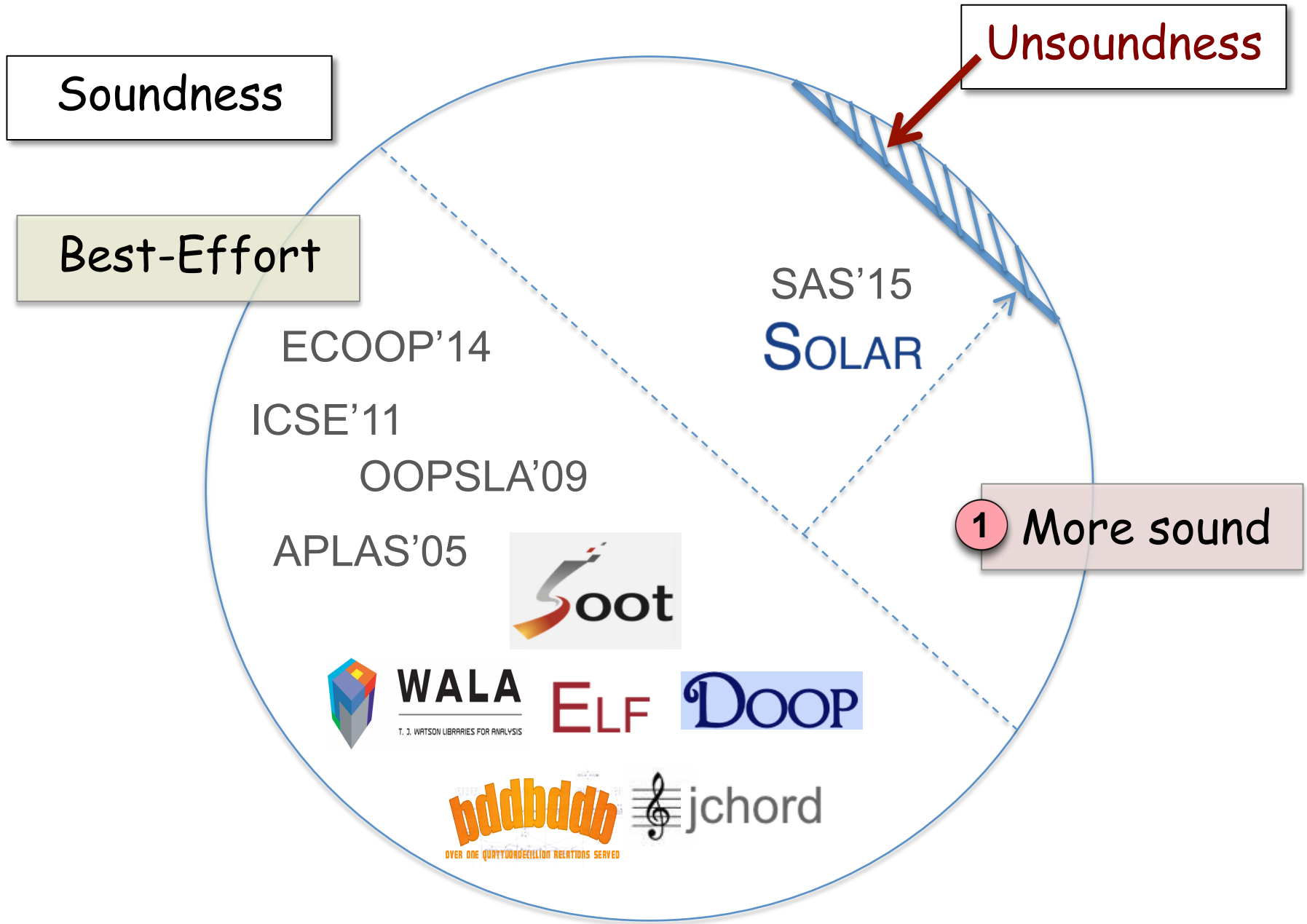
DOOP

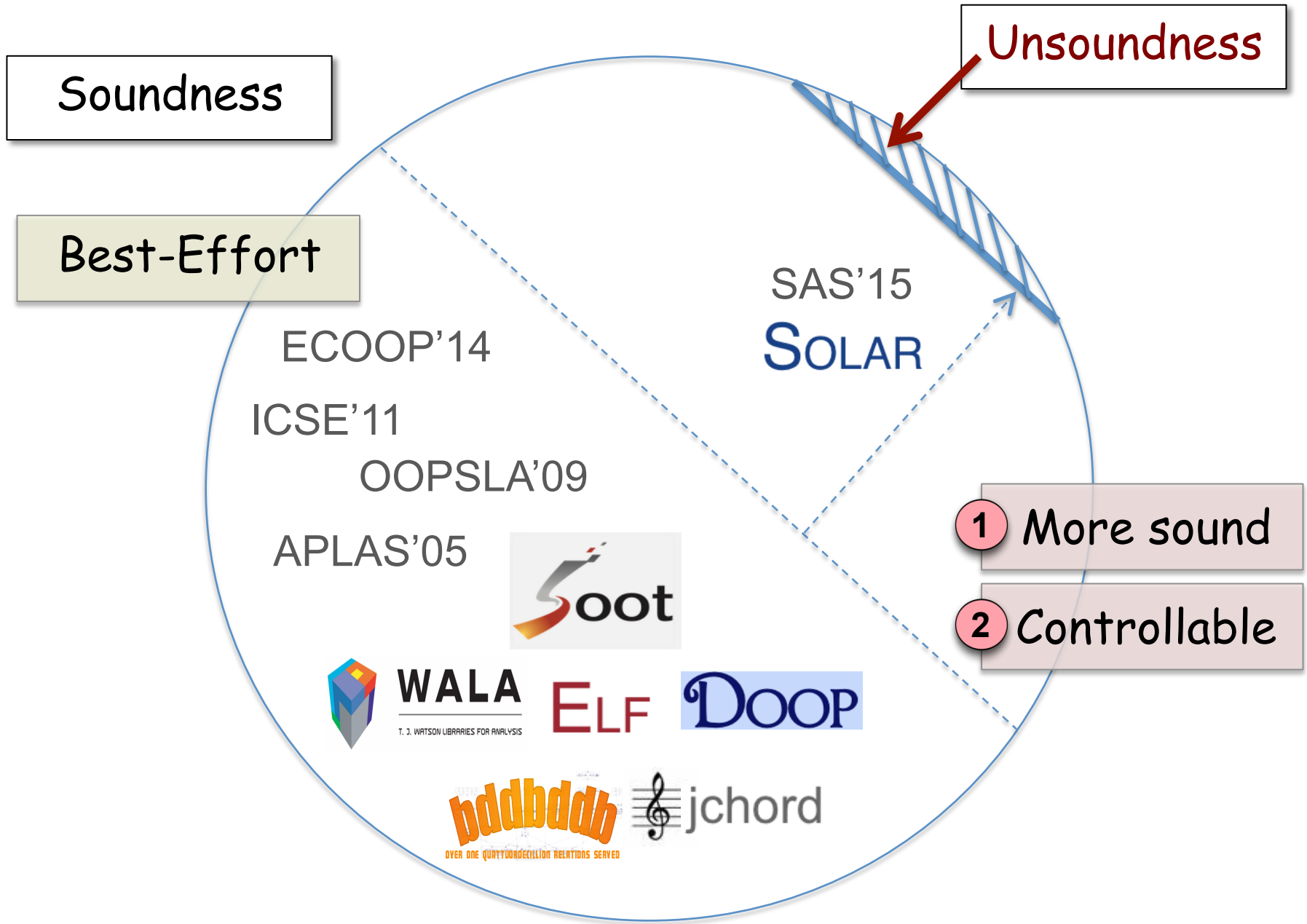
Soundness

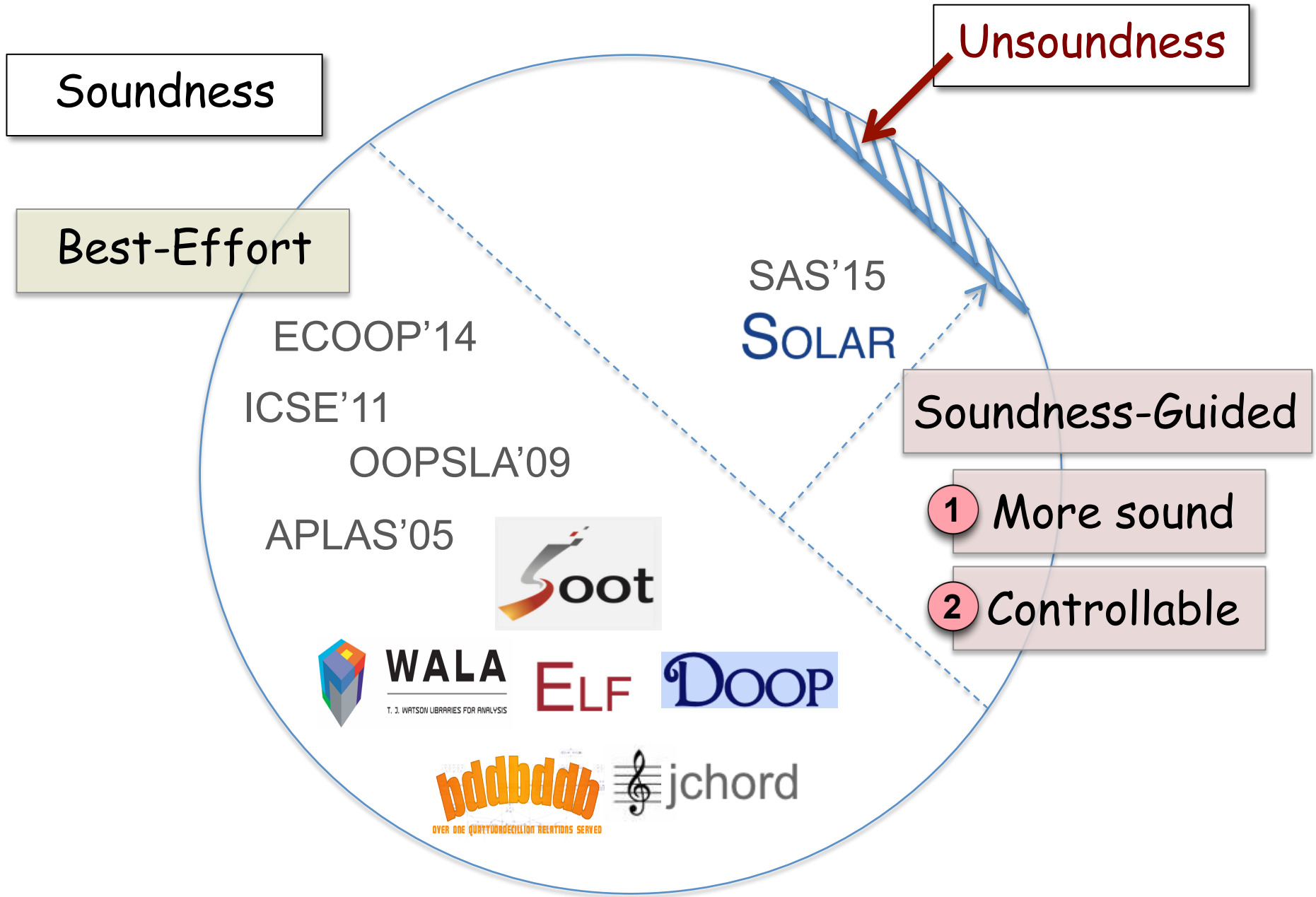
Best-Effort



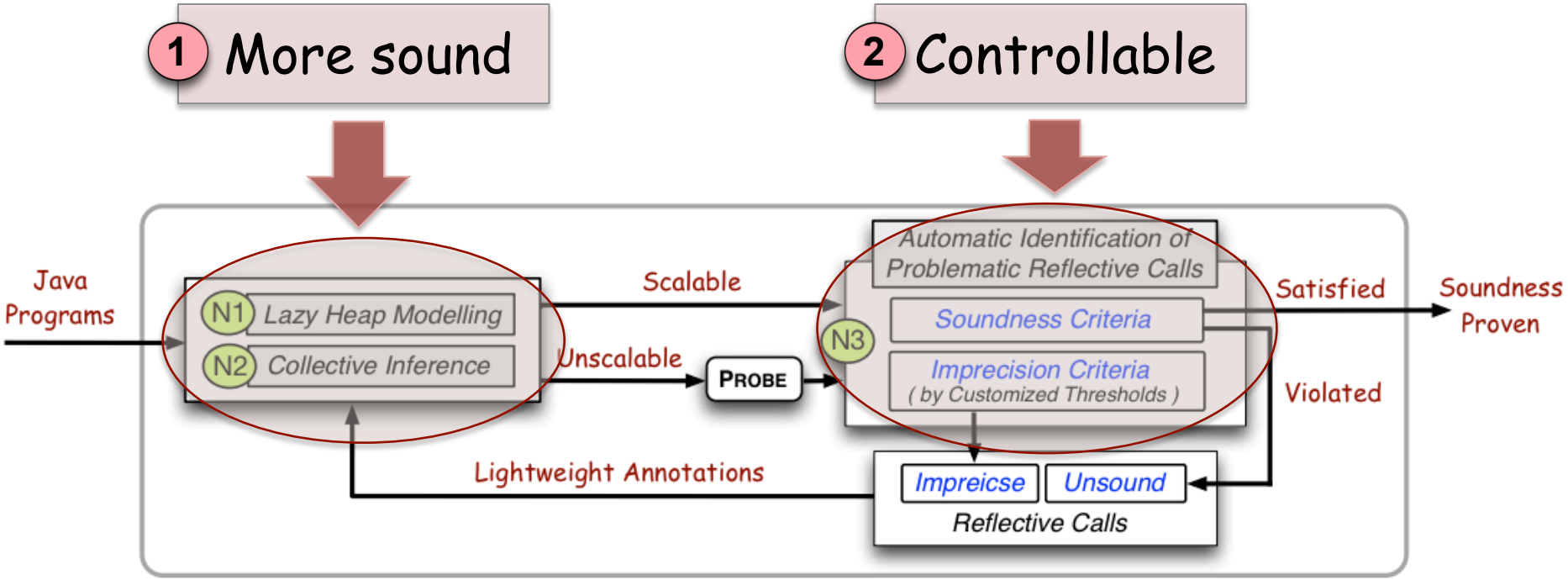

Soundness
Best-Effort
SAS'15
SOLAR
ECOOP'14
ICSE'11
OOPSLA'09
APLAS'05
Soot
WALA
T. J. WATSON LIBRARIES FOR ANALYSIS
ELF
DOOP
bddbddb
OVER ONE QUATTUORDECILLION RELATIONS SERVED
jchord
1 More sound


Soundness
Unsoundness
Best-Effort
SAS'15
SOLAR
ECOOP'14
ICSE'11
OOPSLA'09
APLAS'05
Soot
WALA
T. J. WATSON LIBRARIES FOR ANALYSIS
ELF
DOOP
bddbddb
OVER ONE QUATTUORDECILLION RELATIONS SERVED
jchord
1 More sound

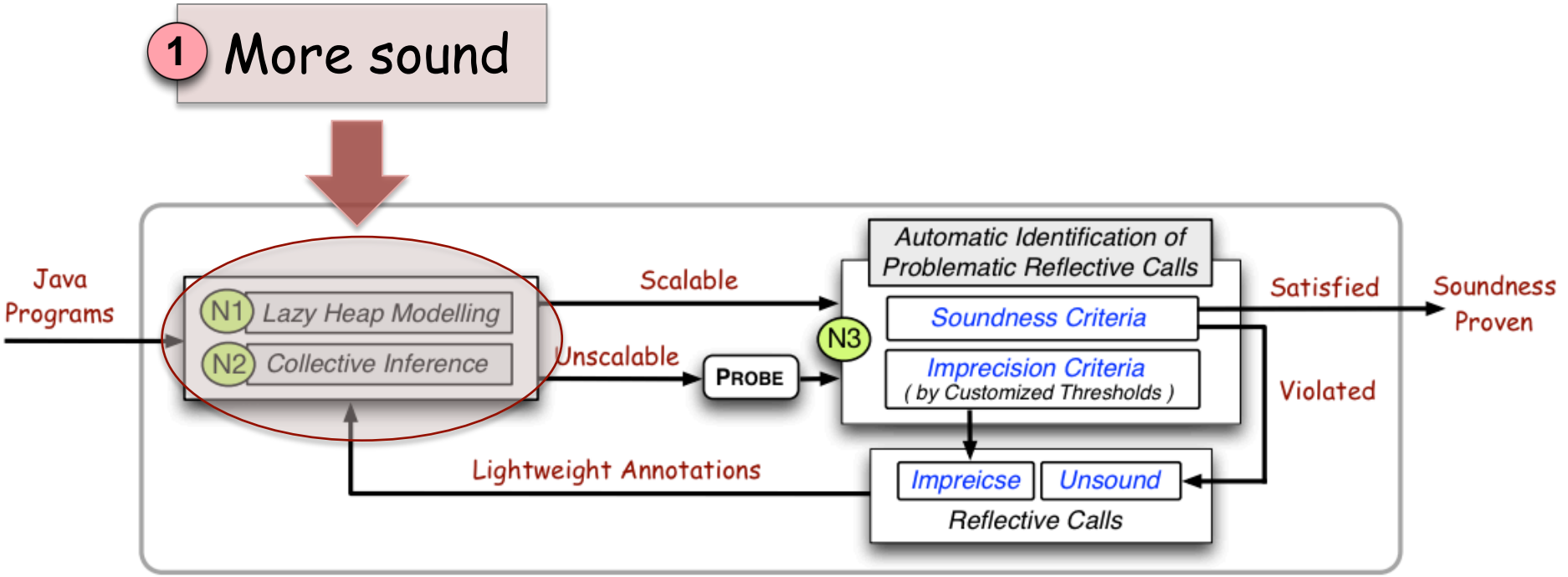

Soundness
Unsoundness
Best-Effort
SAS'15
SOLAR
ECOOP'14
ICSE'11
OOPSLA'09
APLAS'05
Soot
WALA
T. J. WATSON LIBRARIES FOR ANALYSIS
ELF
DOOP
bddbddb
OVER ONE QUATTUORDECILLION RELATIONS SERVED
jchord
1 More sound
2 Controllable


Soundness
Unsoundness
Best-Effort
SAS'15
SOLAR
ECOOP'14
ICSE'11
OOPSLA'09
APLAS'05
Soot
Soundness-Guided
1 More sound
2 Controllable
WALA
T. J. WATSON LIBRARIES FOR ANALYSIS
ELF
DOOP
bddbddb
OVER ONE QUATTUORDECILLION RELATIONS SERVED
jchord

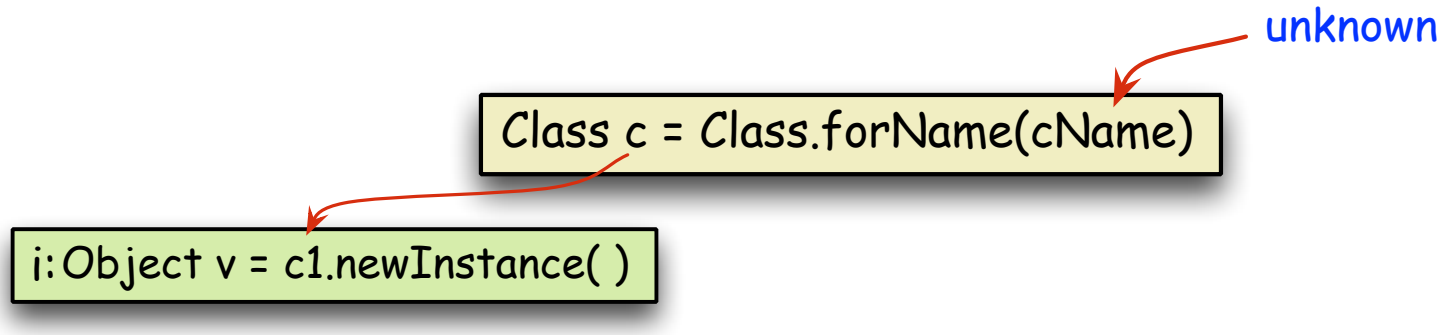
SOLAR


1 More sound
2 Controllable
Java Programs
N1 Lazy Heap Modelling
N2 Collective Inference
Scalable
Unscalable
PROBE
Automatic Identification of Problematic Reflective Calls
N3
Soundness Criteria
Imprecision Criteria
(by Customized Thresholds)
Satisfied
Violated
Soundness Proven
Lightweight Annotations
Impreicse
Unsound
Reflective Calls

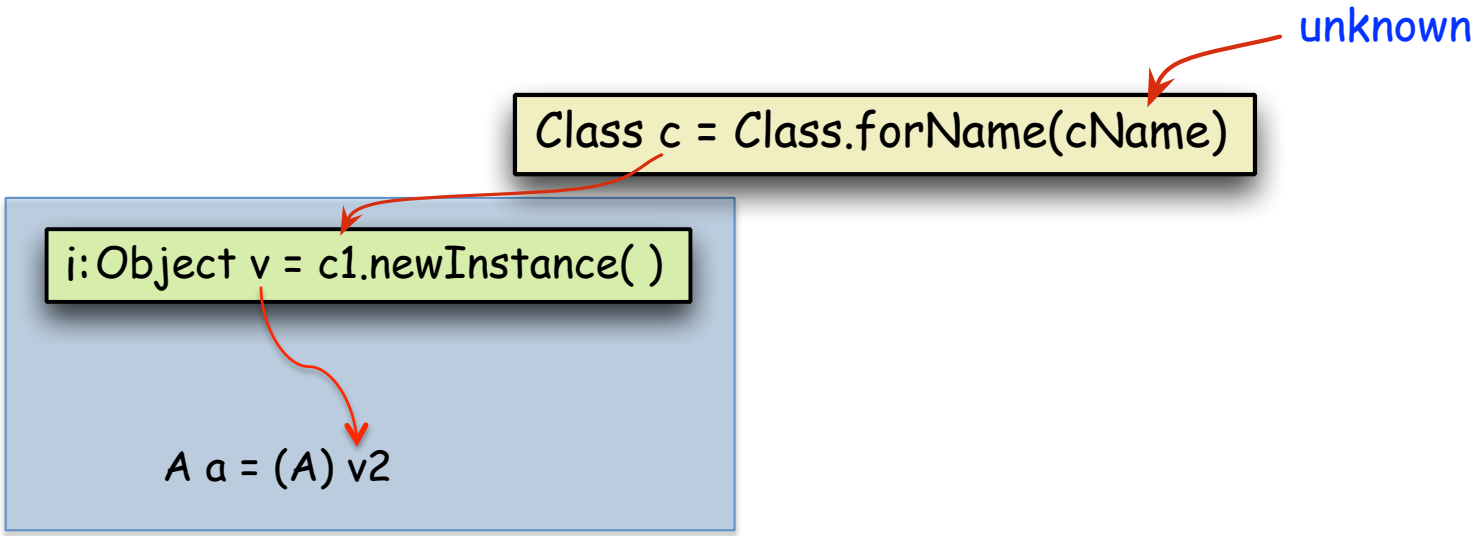
SOLAR


1 More sound
Java Programs
N1 Lazy Heap Modelling
N2 Collective Inference
Scalable
Unscalable
PROBE
Automatic Identification of Problematic Reflective Calls
N3
Soundness Criteria
Imprecision Criteria
(by Customized Thresholds)
Satisfied
Soundness Proven
Violated
Lightweight Annotations
Impreicse
Unsound
Reflective Calls

The Challenging Problem

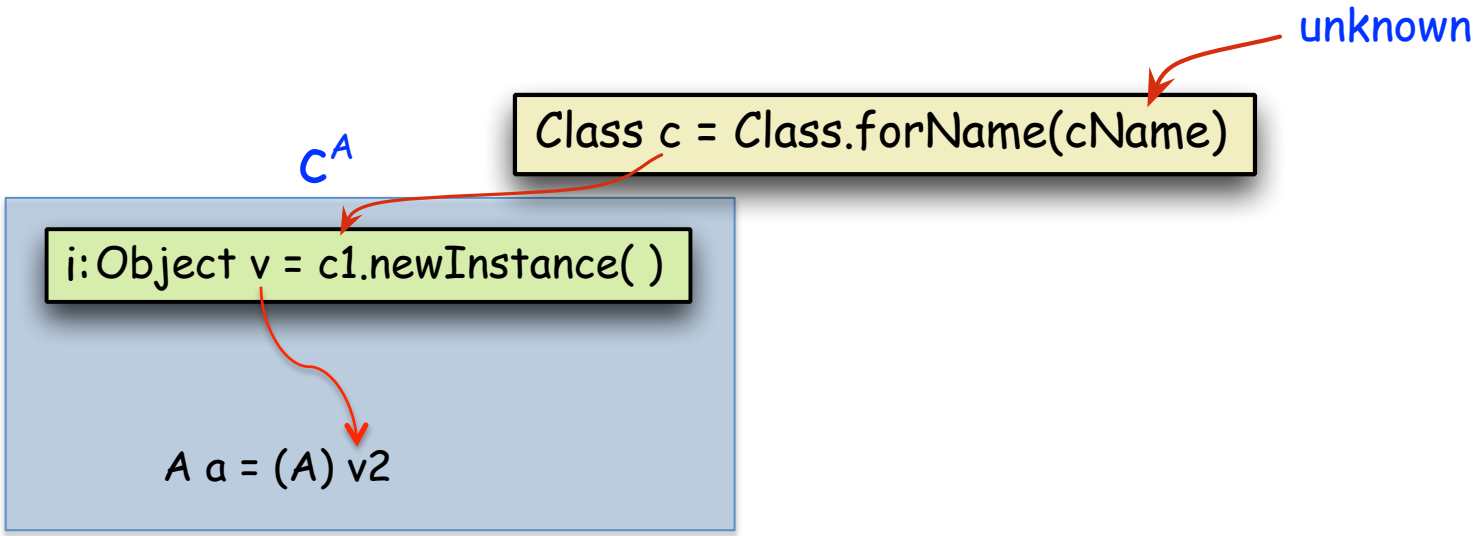


Existing Approach



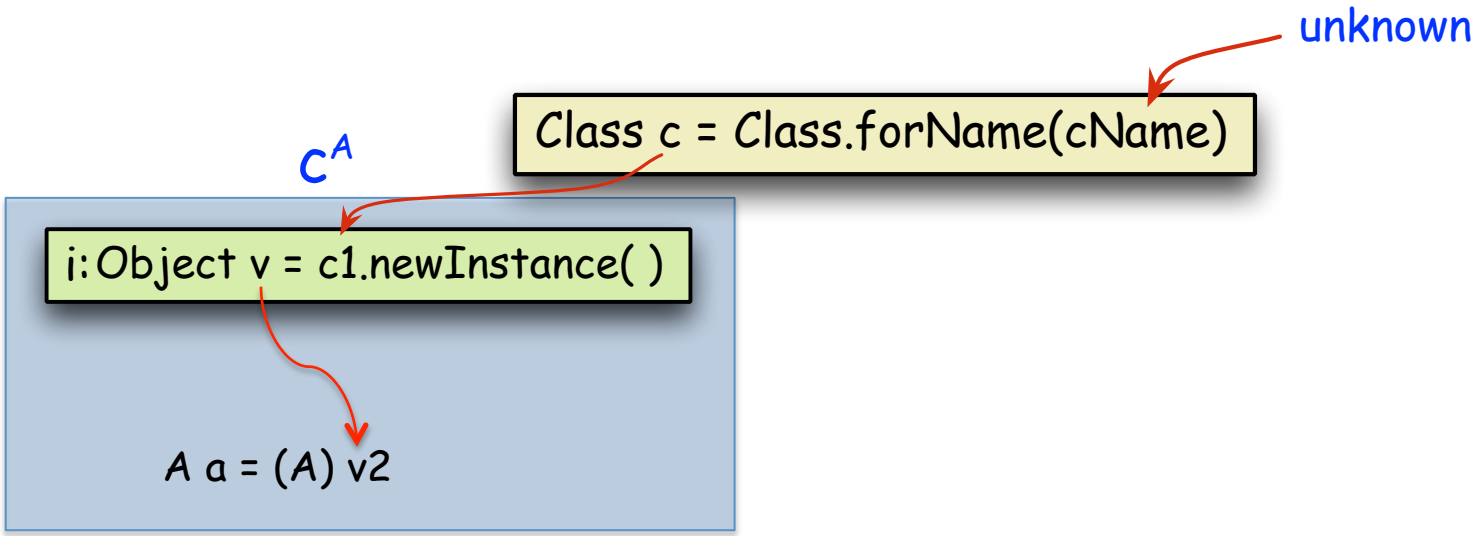
Intra-procedural post-dominant cast operations

Existing Approach



Intra-procedural post-dominant cast operations

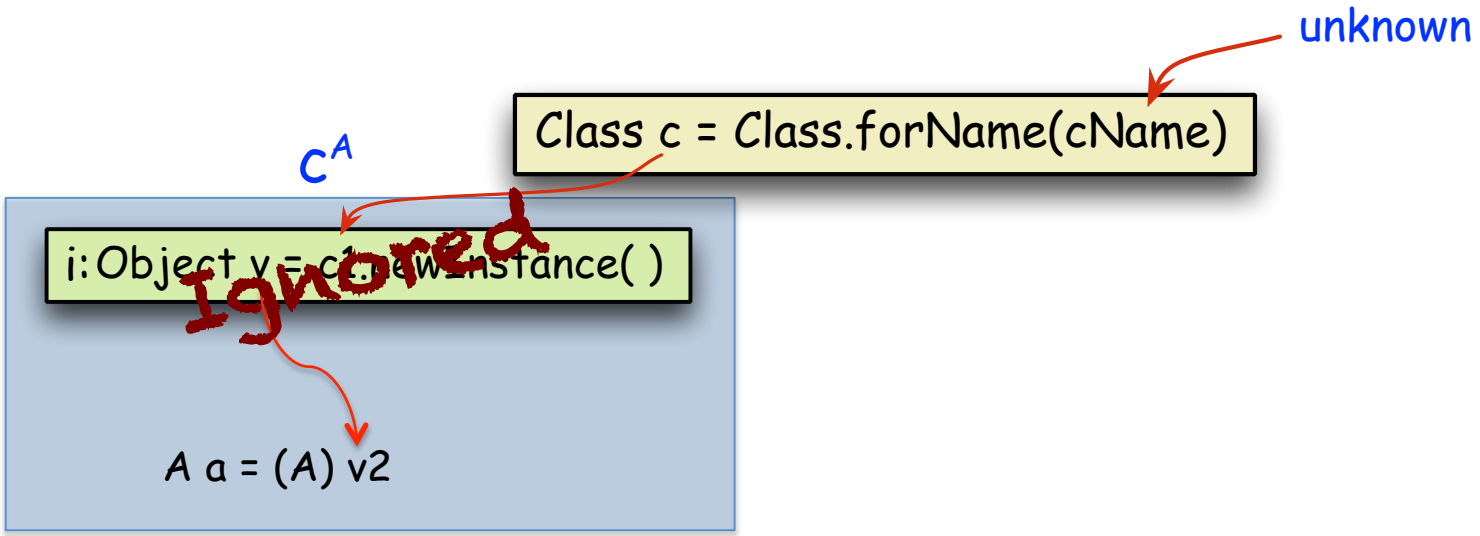
Existing Approach



Intra-procedural post-dominant cast operations

only works for this intra-post-dominance pattern

Existing Approach



Intra-procedural post-dominant cast operations

only works for this intra-post-dominance pattern

Lazy Heap Modeling (LHM)

Lazy Heap Modeling (LHM)

Observation

A reflectively created object
(returned by newInstance())
is usually used in two cases in practice

Lazy Heap Modeling (LHM)

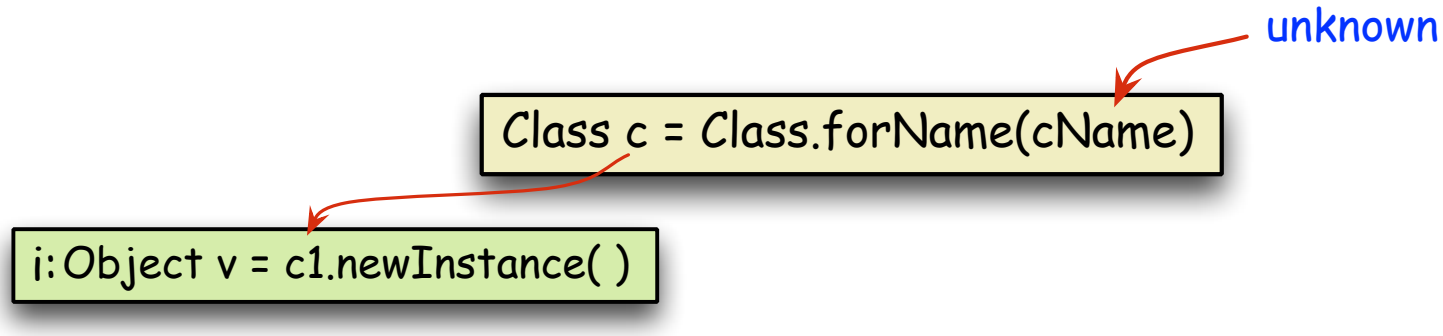
Observation

A reflectively created object
(returned by newInstance())
is usually used in two cases in practice

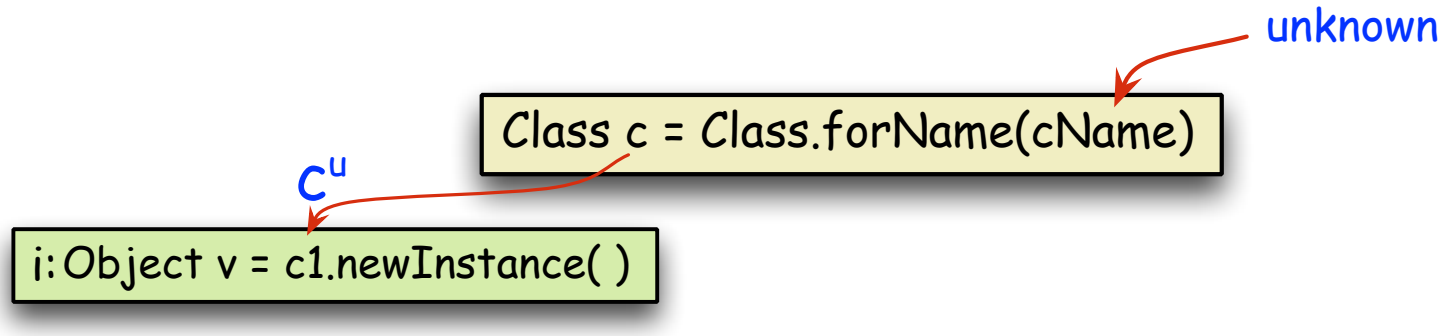
Intuition

The side effect of a newInstance() call
can be modeled
lazily at these usage points

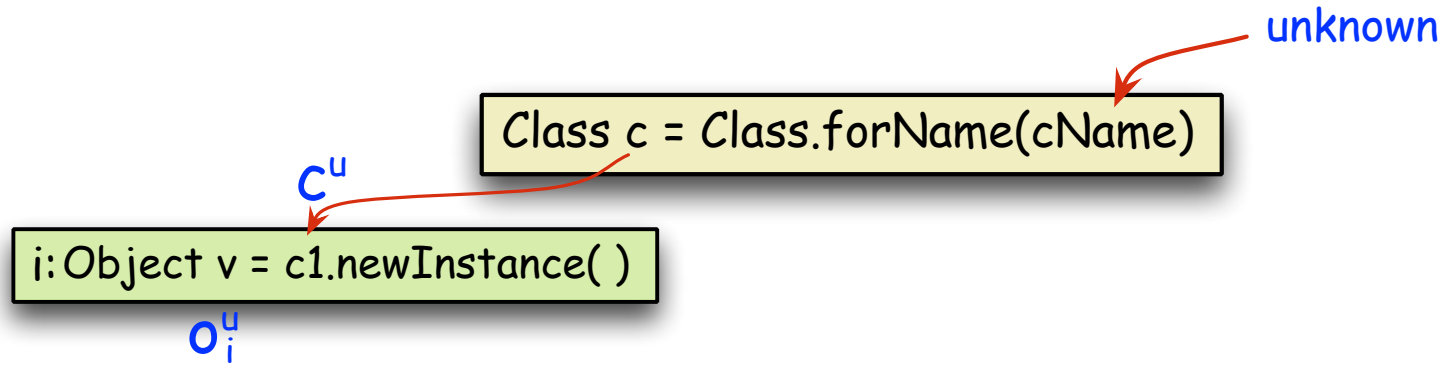
Lazy Heap Modeling (LHM)



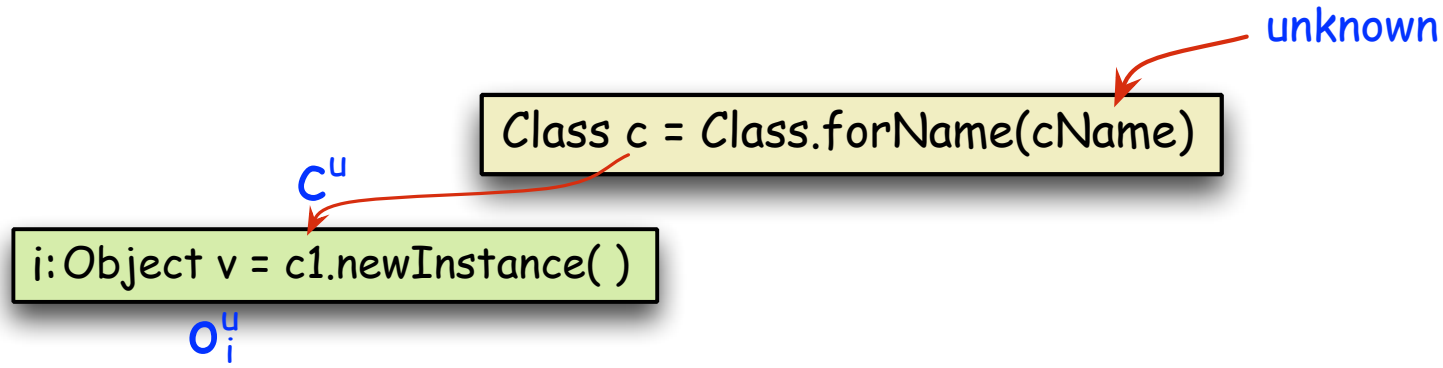
Lazy Heap Modeling (LHM)



Lazy Heap Modeling (LHM)

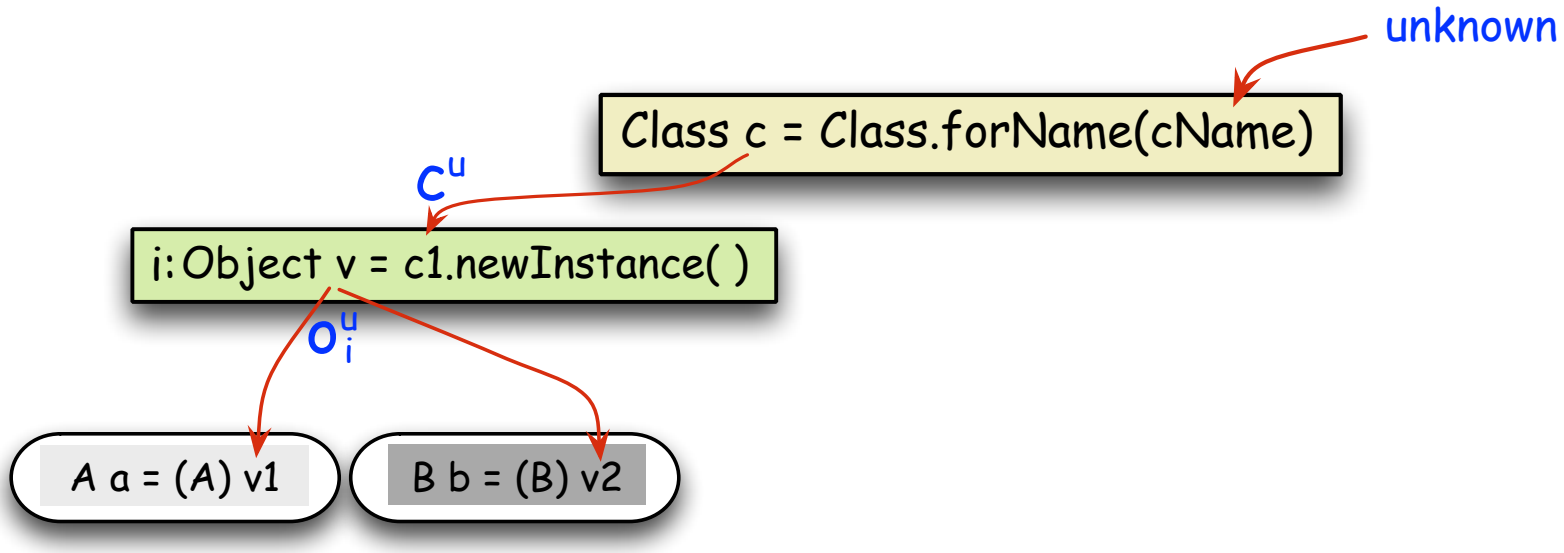


Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

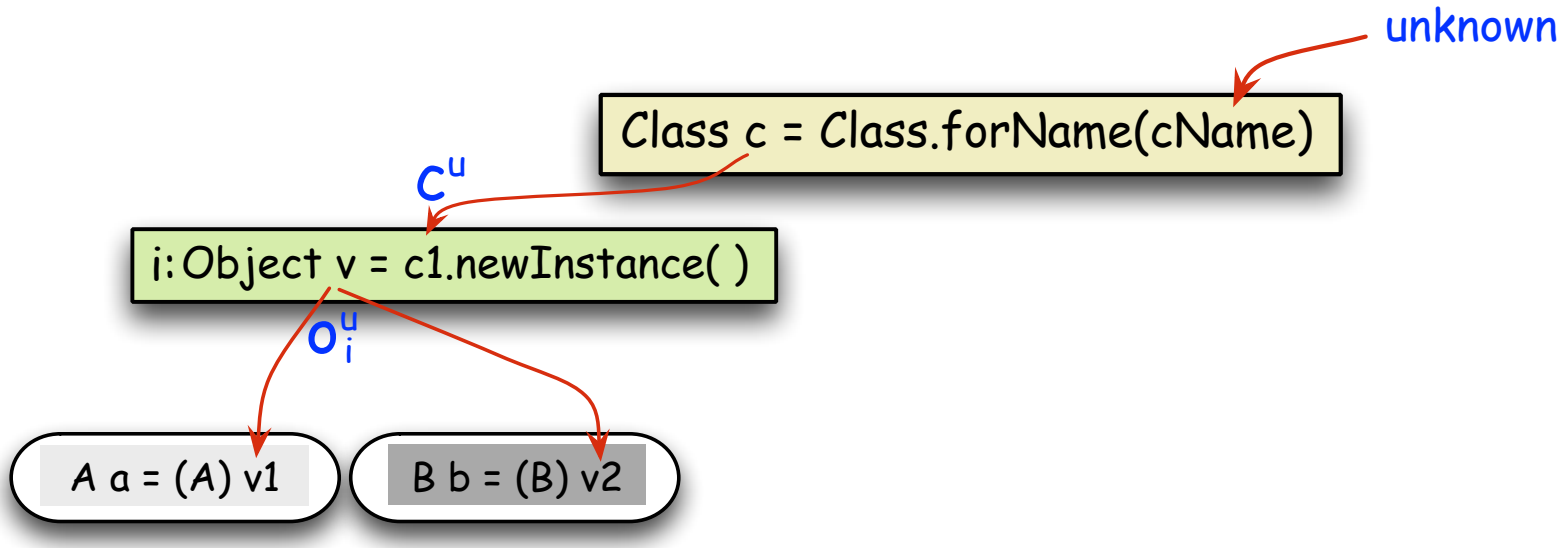
Lazy Heap Modeling (LHM)



Case (I)

Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Lazy Heap Modeling (LHM)

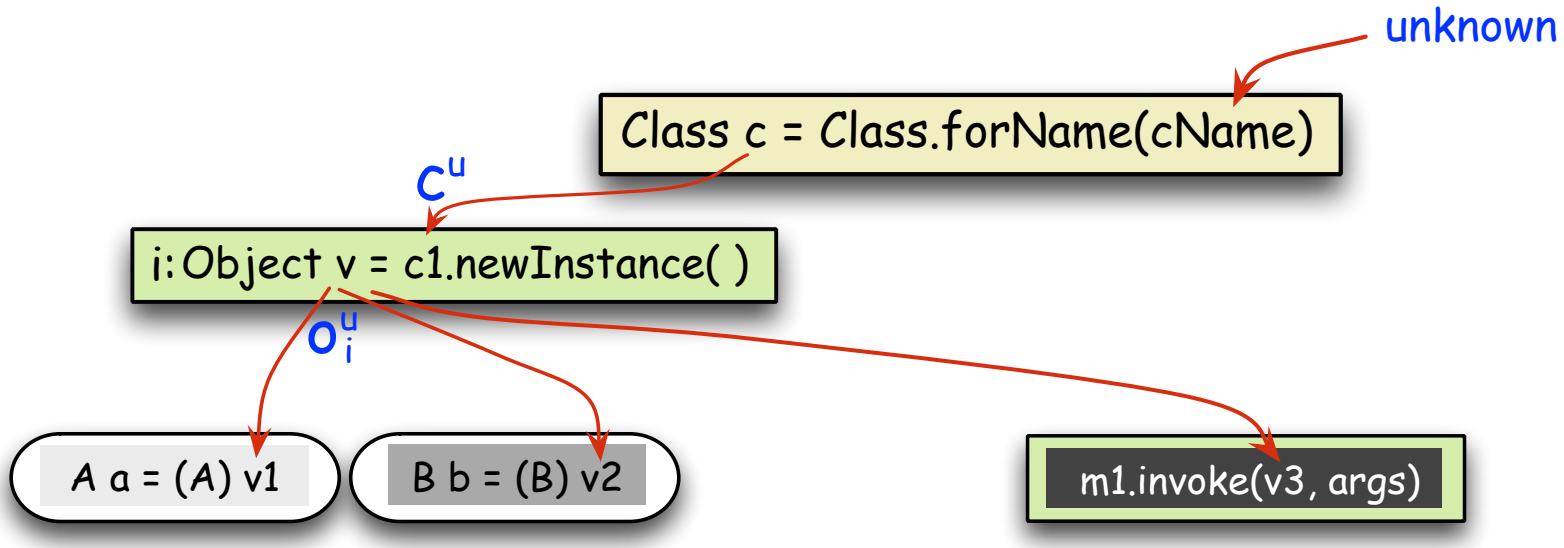


Case (I)

Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b

Lazy Heap Modeling (LHM)



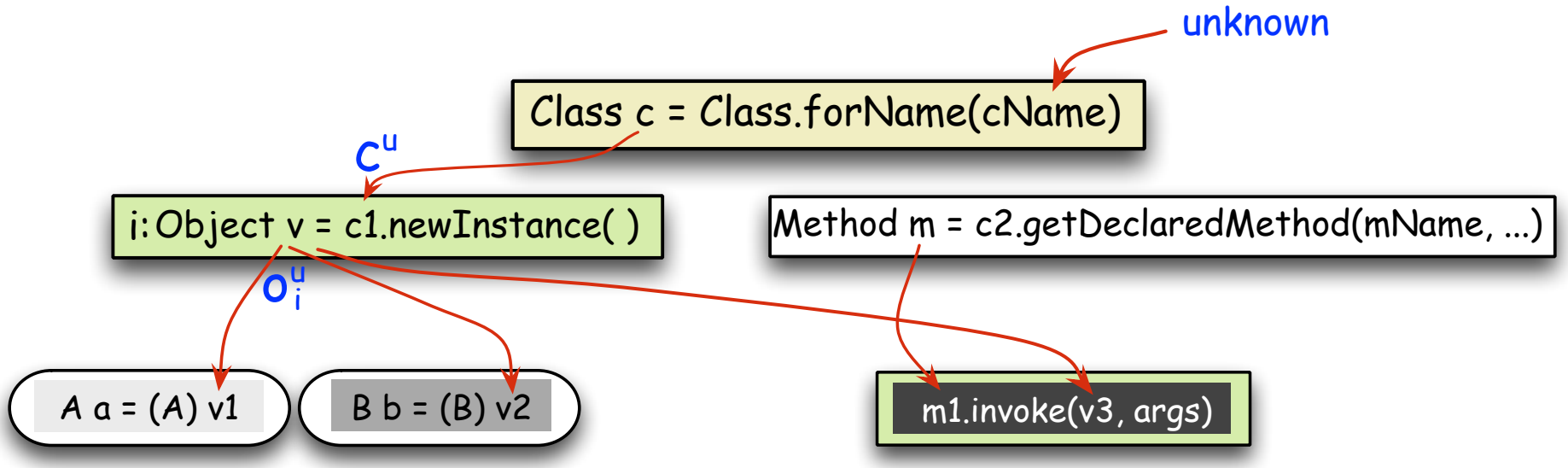
Case (I)

Case (II)

Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b

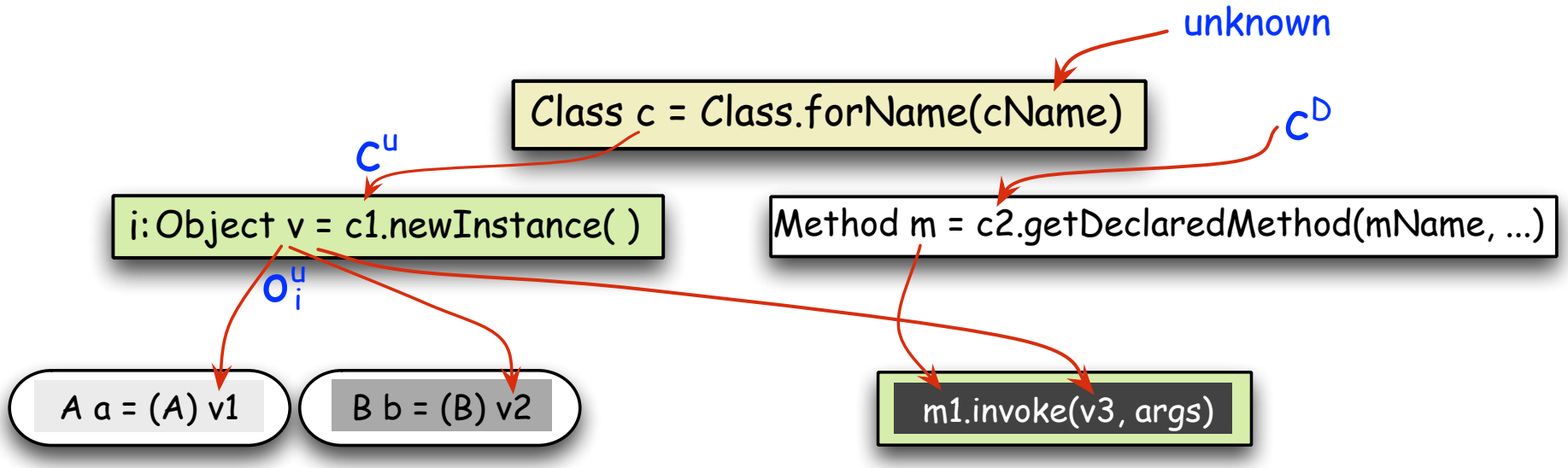
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b

Lazy Heap Modeling (LHM)



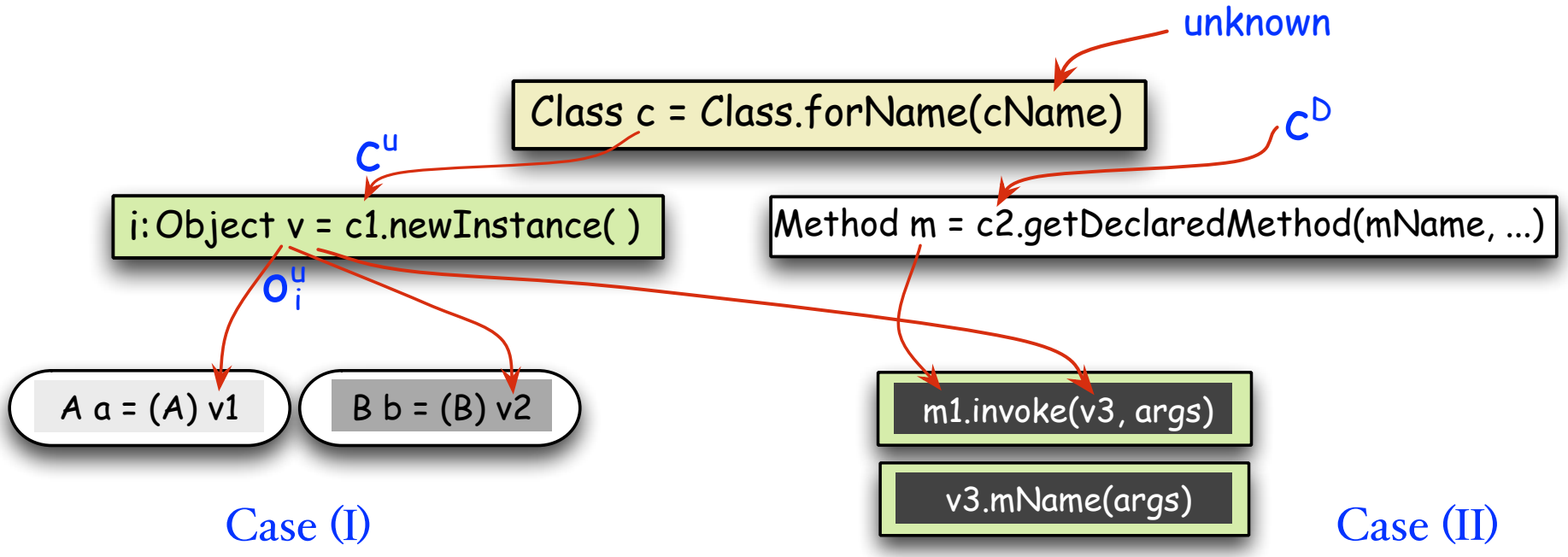
Case (I)

Case (II)

Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b

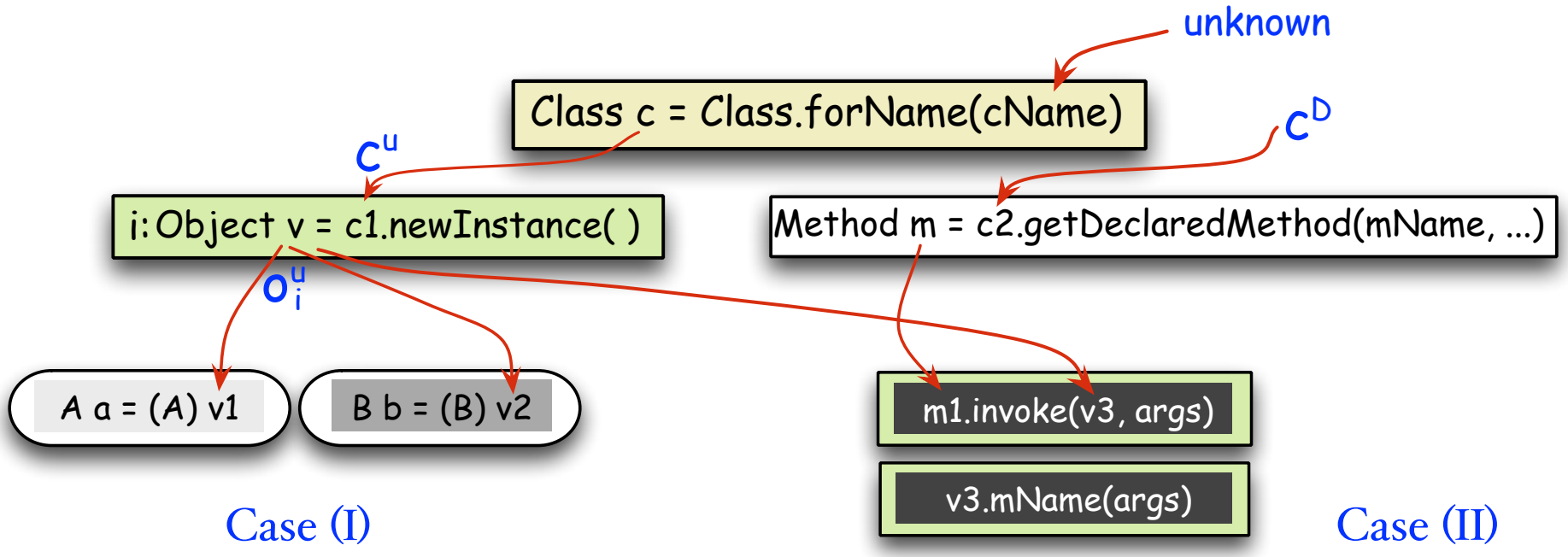
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b

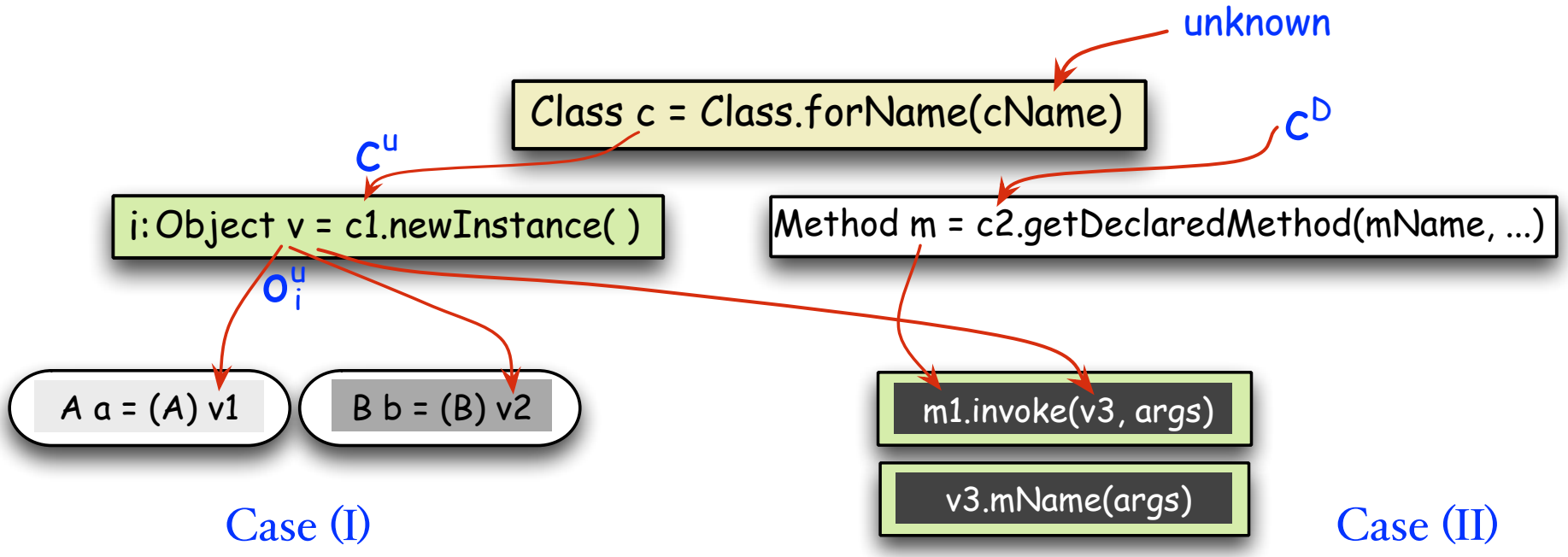
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b
D	o_i^D	i	v3

Lazy Heap Modeling (LHM)

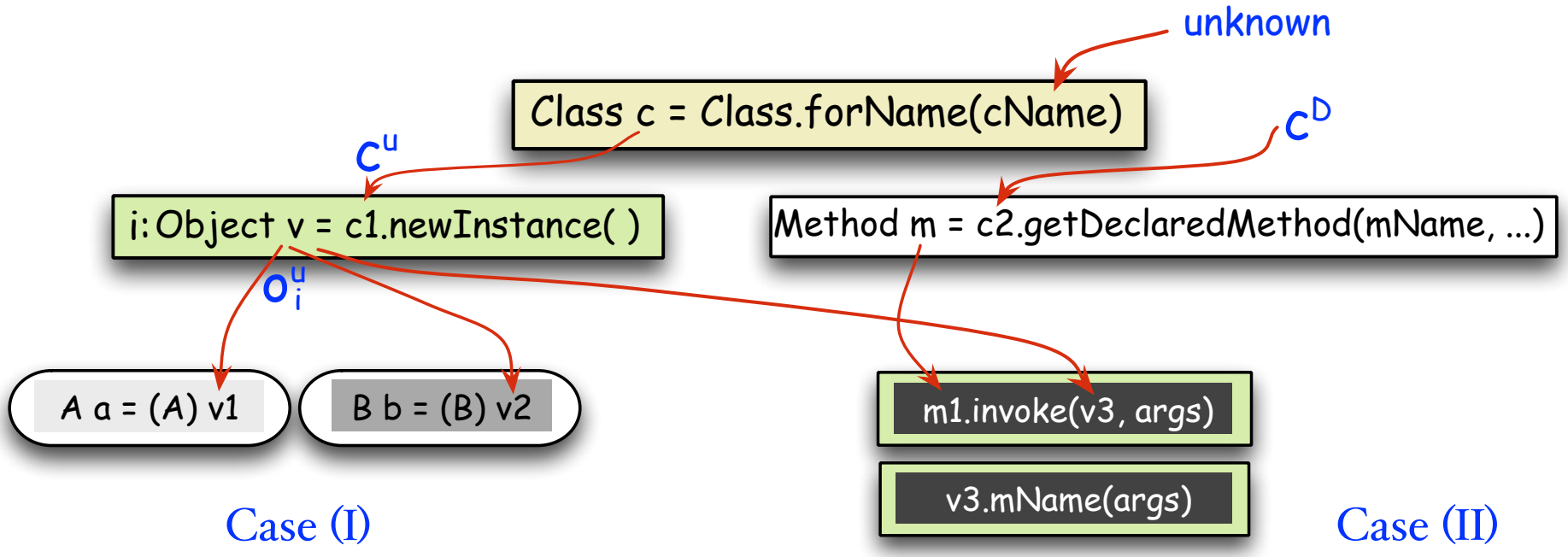


Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b
D	o_i^D	i	v3
B	o_i^B	i	v3

?

Lazy Heap Modeling (LHM)



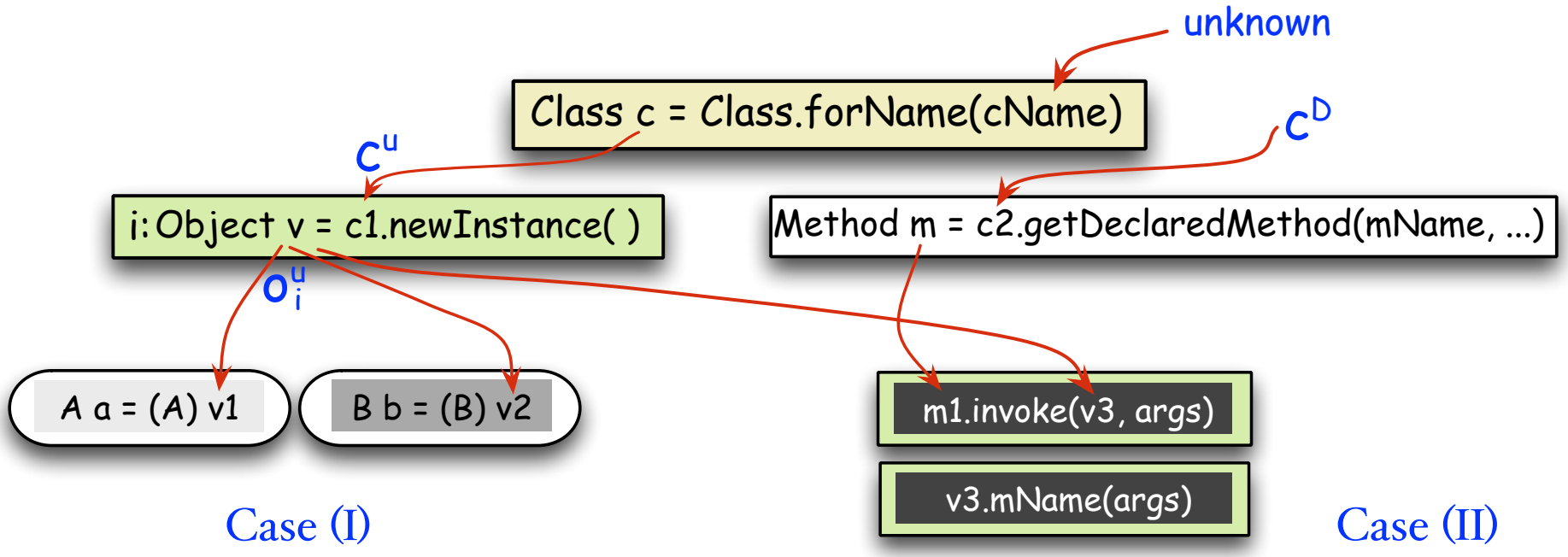
Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b
D	o_i^D	i	v3
B	o_i^B	i	v3

X

?

Lazy Heap Modeling (LHM)

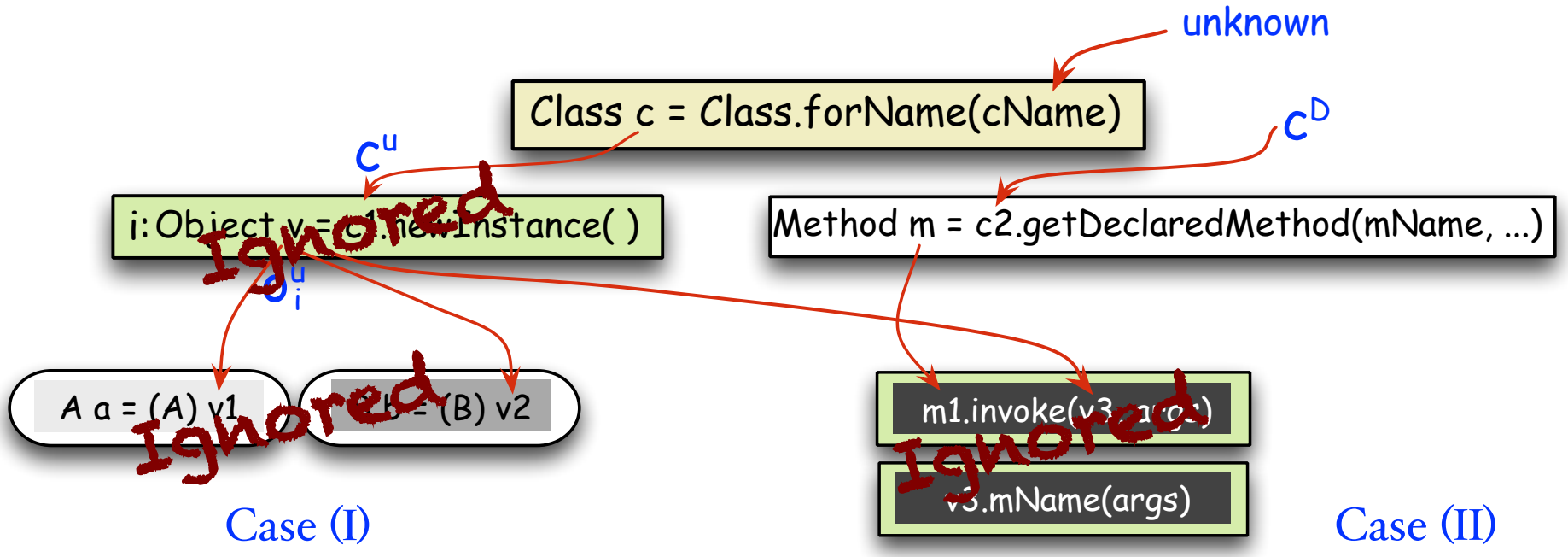


Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b
D	o_i^D	i	v3
B	o_i^B	i	v3
B	o_i^B	i	b, v3

?

Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b
D	o_i^D	i	v3
B	o_i^B	i	v3
B	o_i^B	i	b, v3

X

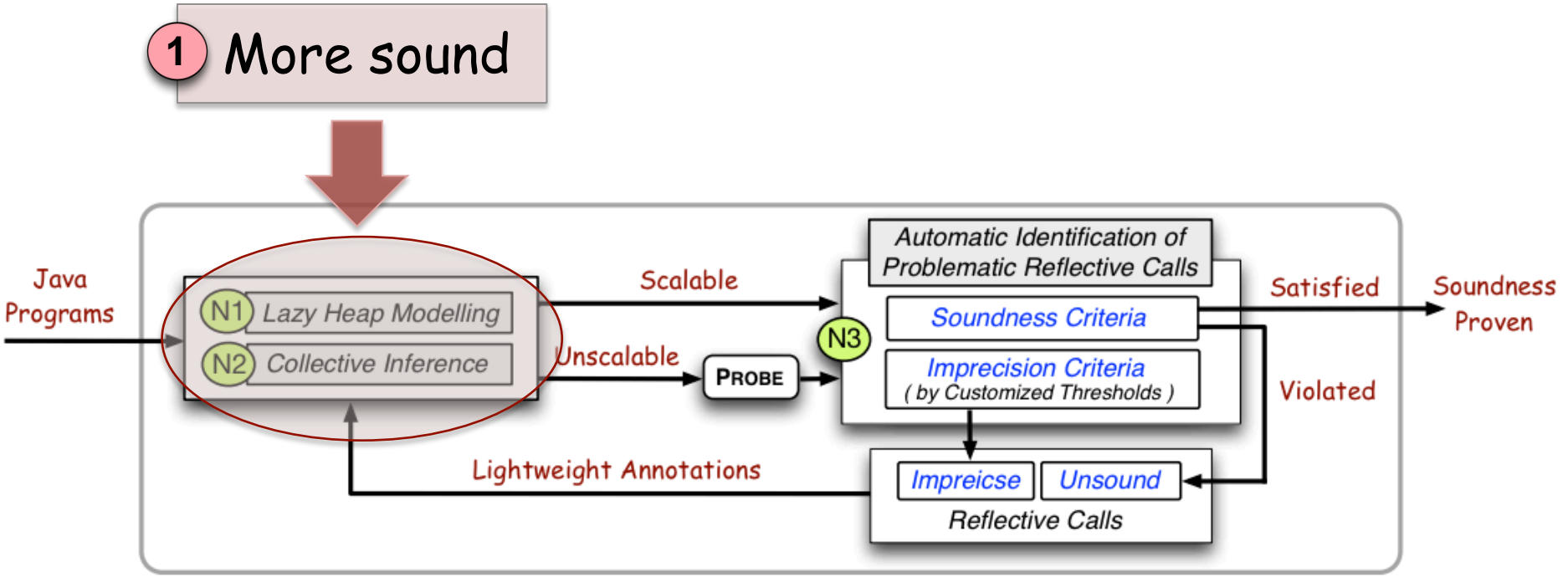
?

Inference System of Solar

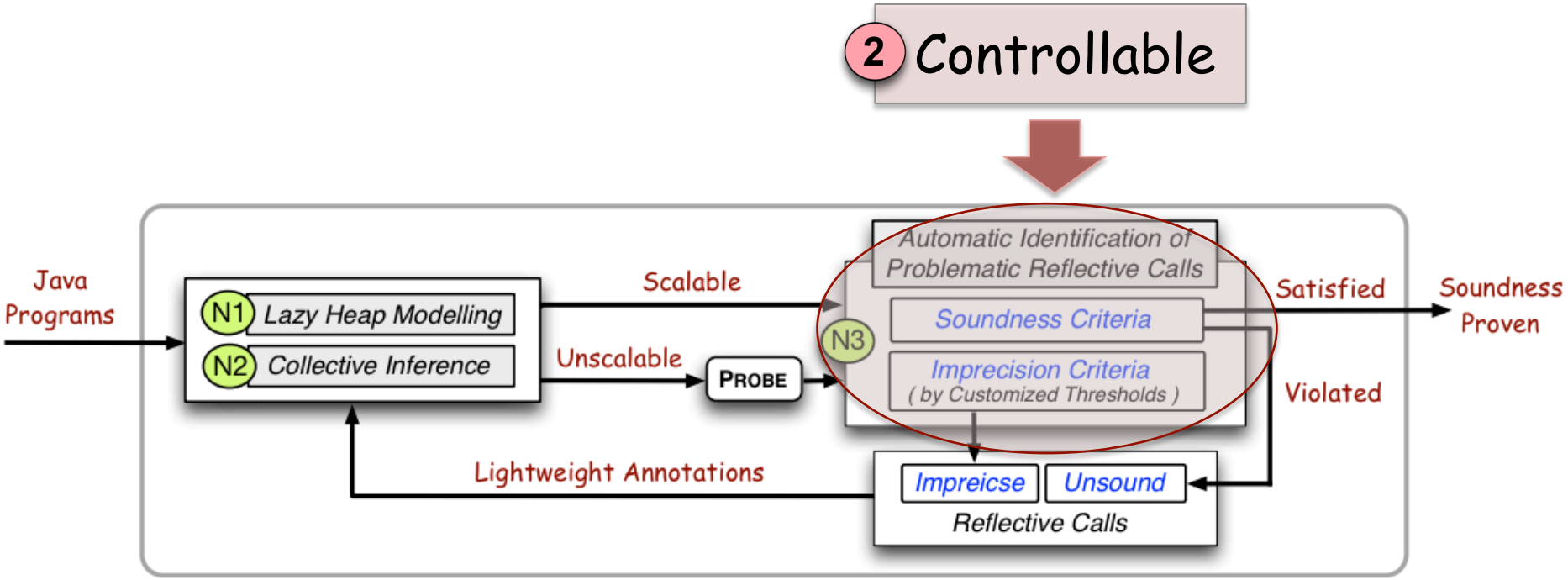
Collective Inference

o = newInstance() ⬅➡ *m.invoke(o,…), f.get(o), f.set(o, …)*

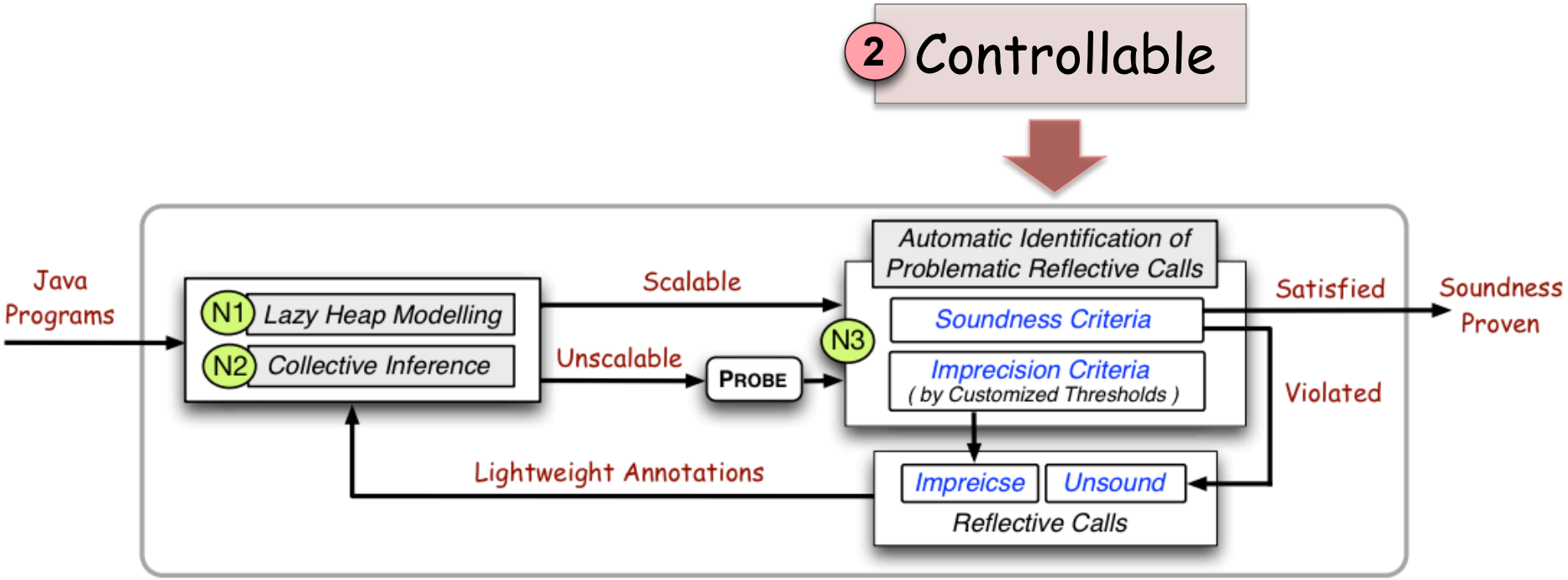
SOLAR


1 More sound
Java Programs
N1 Lazy Heap Modelling
N2 Collective Inference
Scalable
Unscalable
PROBE
Automatic Identification of Problematic Reflective Calls
N3
Soundness Criteria
Imprecision Criteria
(by Customized Thresholds)
Satisfied
Soundness Proven
Violated
Lightweight Annotations
Impreicse
Unsound
Reflective Calls

SOLAR


2 Controllable
Java Programs
N1 Lazy Heap Modelling
N2 Collective Inference
Scalable
Unscalable
PROBE
Automatic Identification of Problematic Reflective Calls
N3
Soundness Criteria
Imprecision Criteria
(by Customized Thresholds)
Satisfied
Violated
Soundness Proven
Lightweight Annotations
Impreicse
Unsound
Reflective Calls

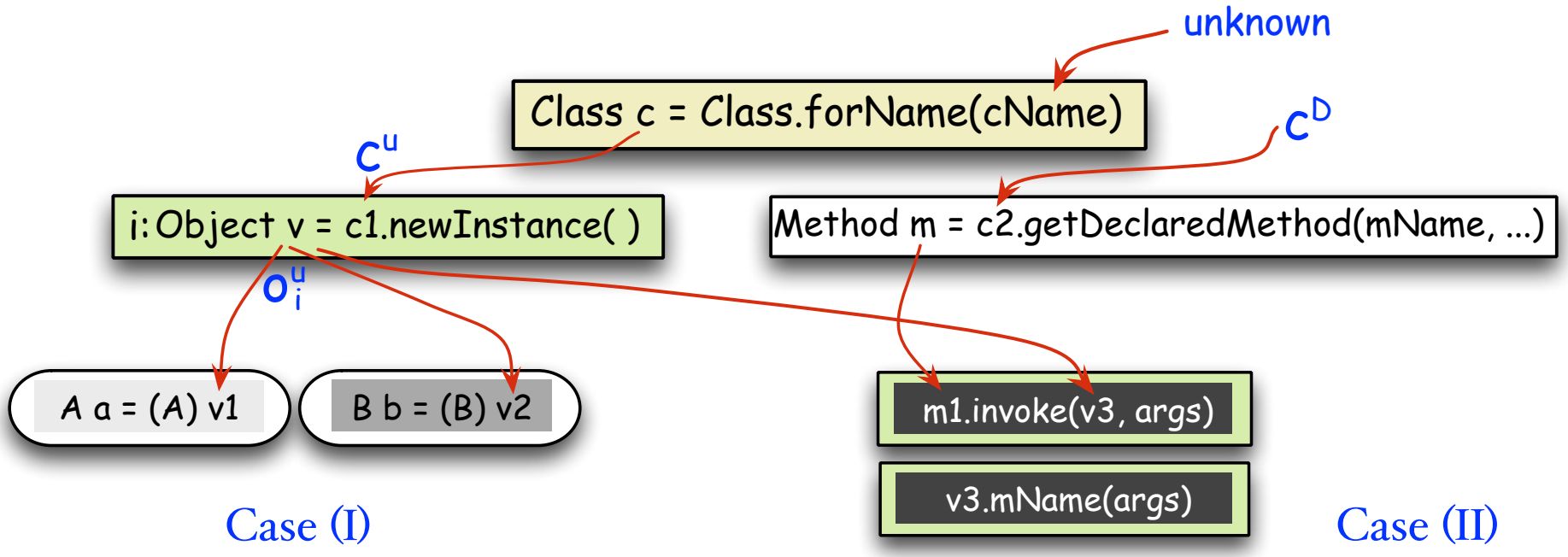
SOLAR



Soundness Criteria

If the information in a program is not enough to help infer the reflective targets, the soundness criteria is not satisfied

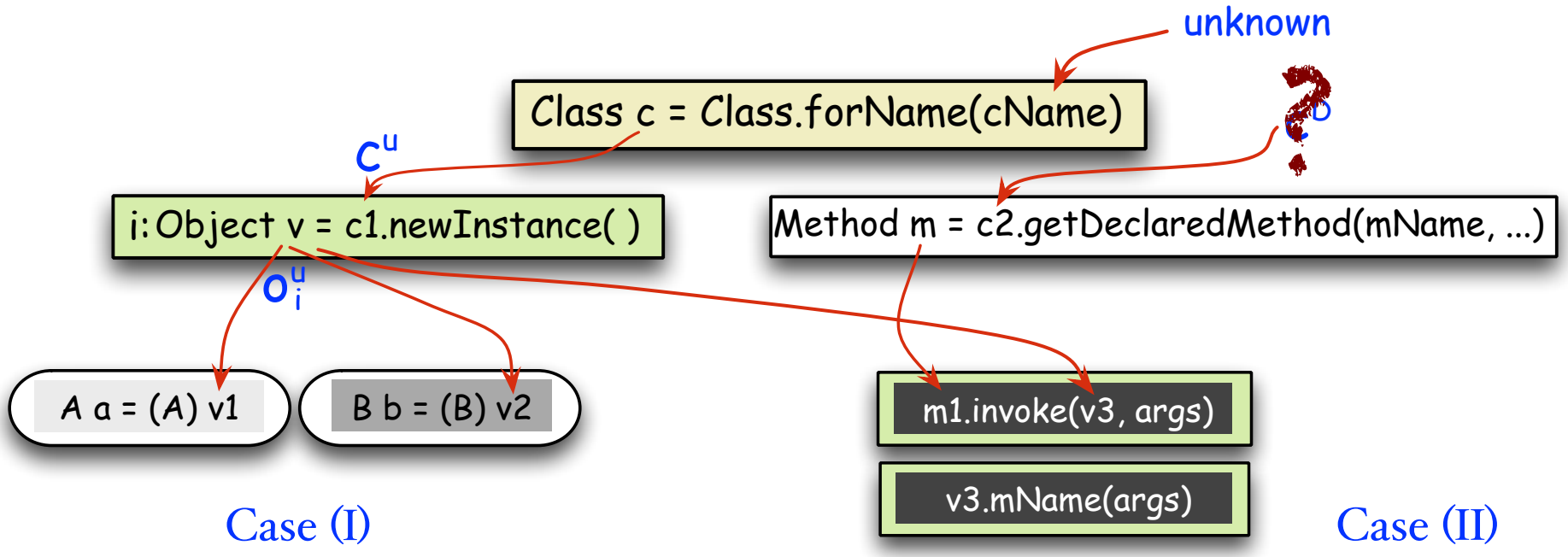
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b, v3
D	o_i^D	i	v3

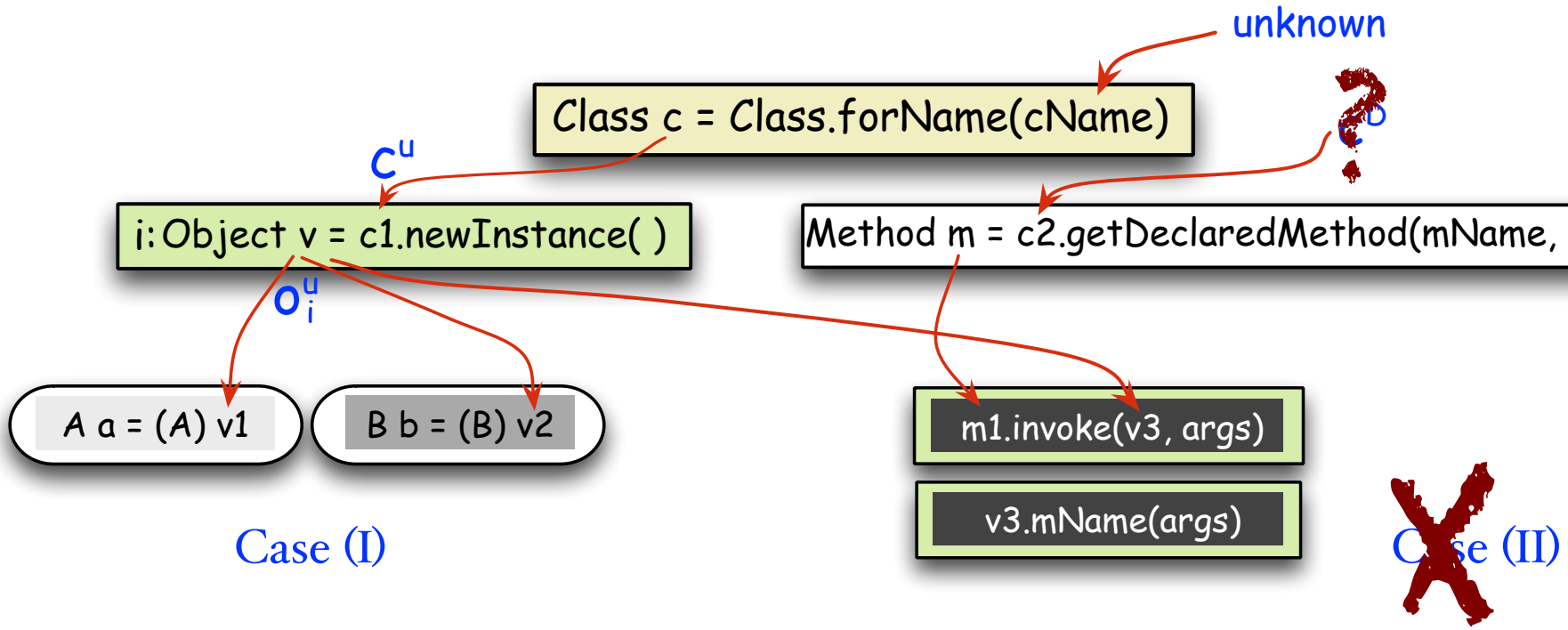
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b, v3
D	o_i^D	i	v3

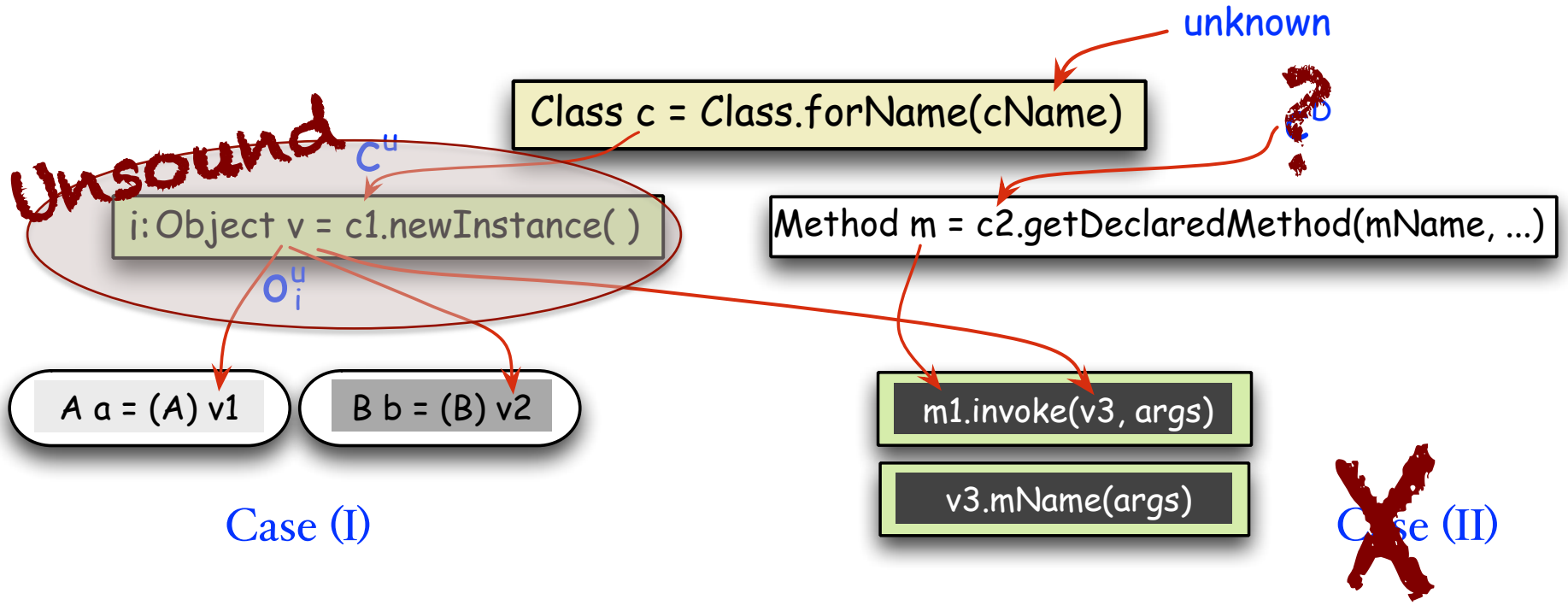
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b, v3
D	o_i^D	i	v3

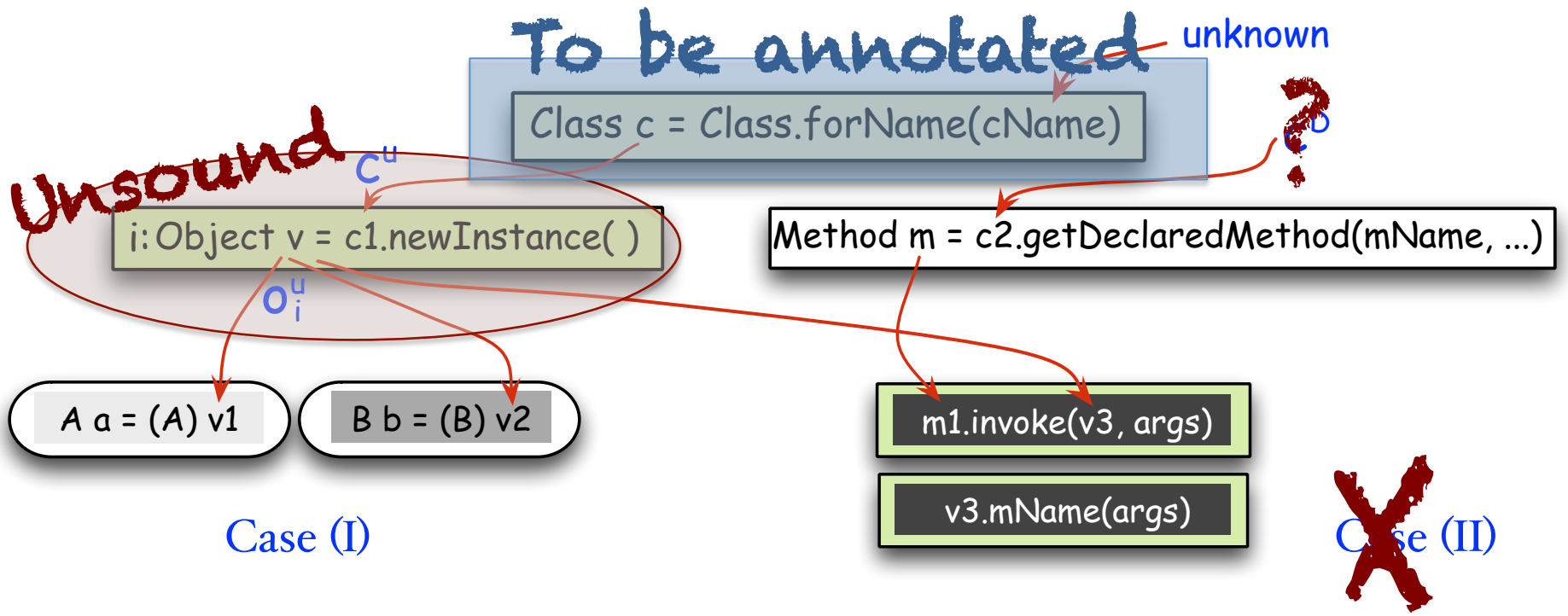
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b, v3
D	o_i^D	i	v3

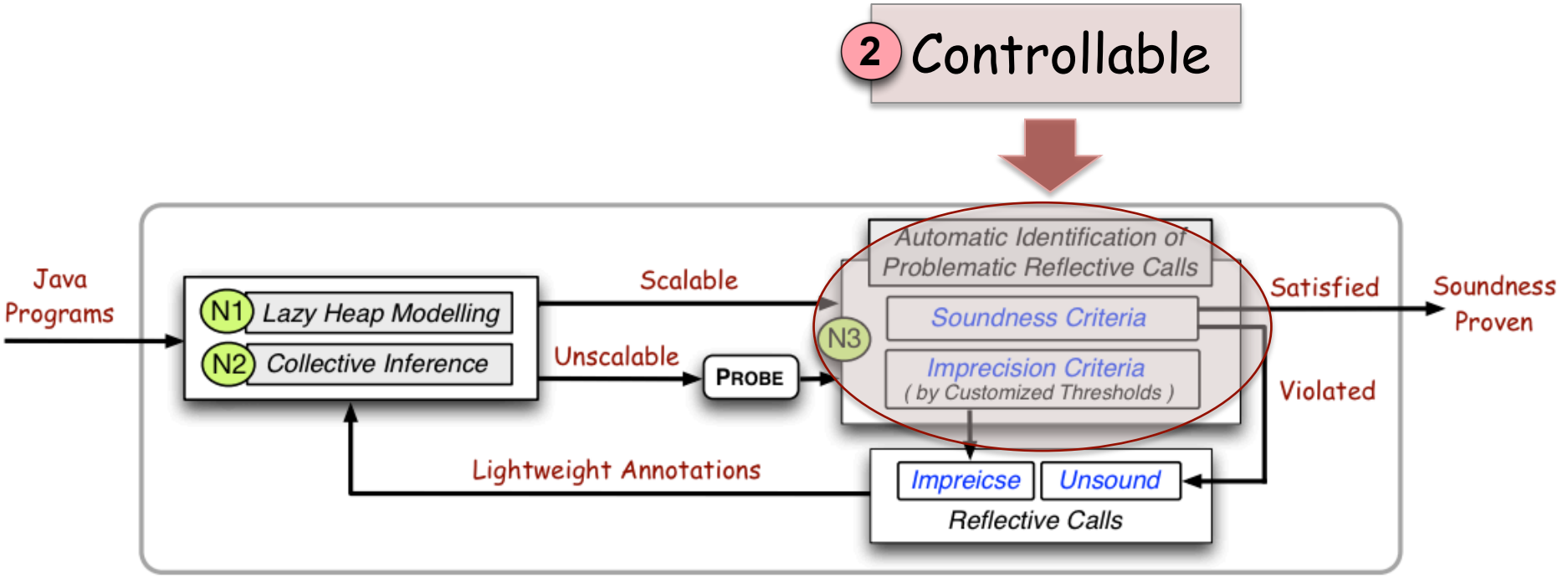
Lazy Heap Modeling (LHM)



Abstract Heap Objects
of newInstance()
are created lazily
(at LHM points)

Type	Object	Location	Pointed by
A	o_i^A	i	a
B	o_i^B	i	b, v3
D	o_i^D	i	v3

SOLAR



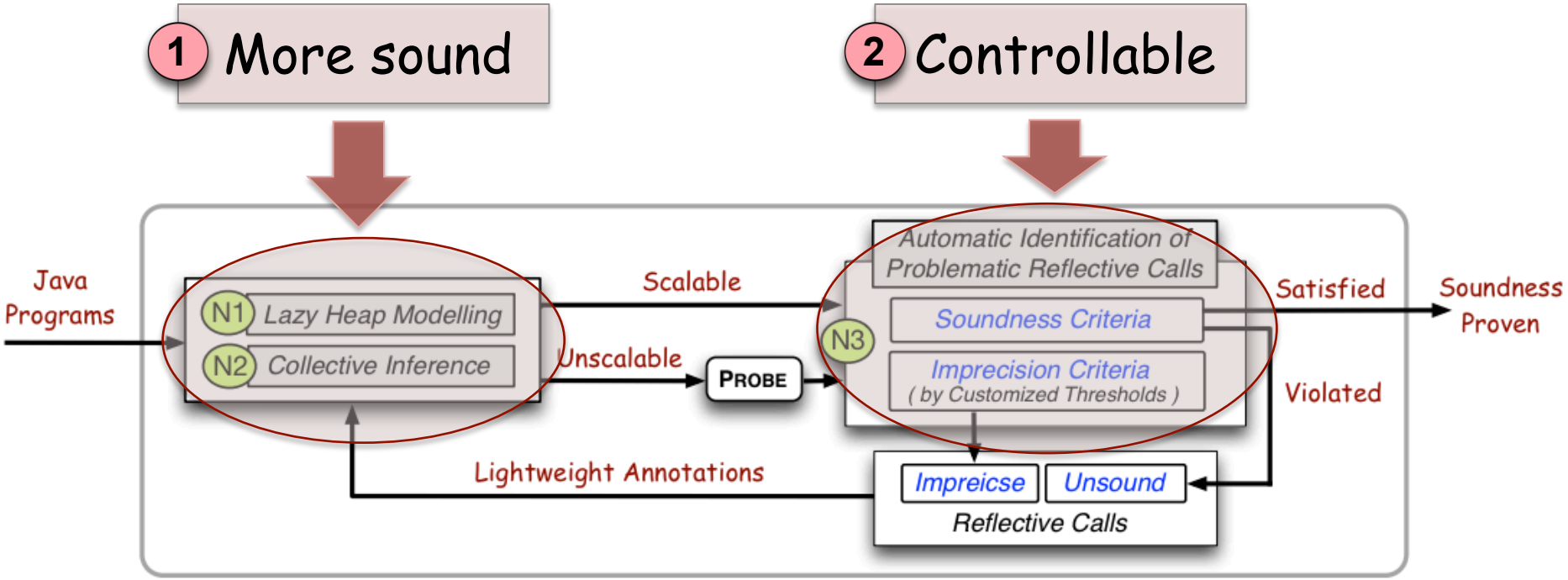
Soundness Criteria

If the information in a program is not enough to help infer the reflective targets, the soundness criteria is not satisfied

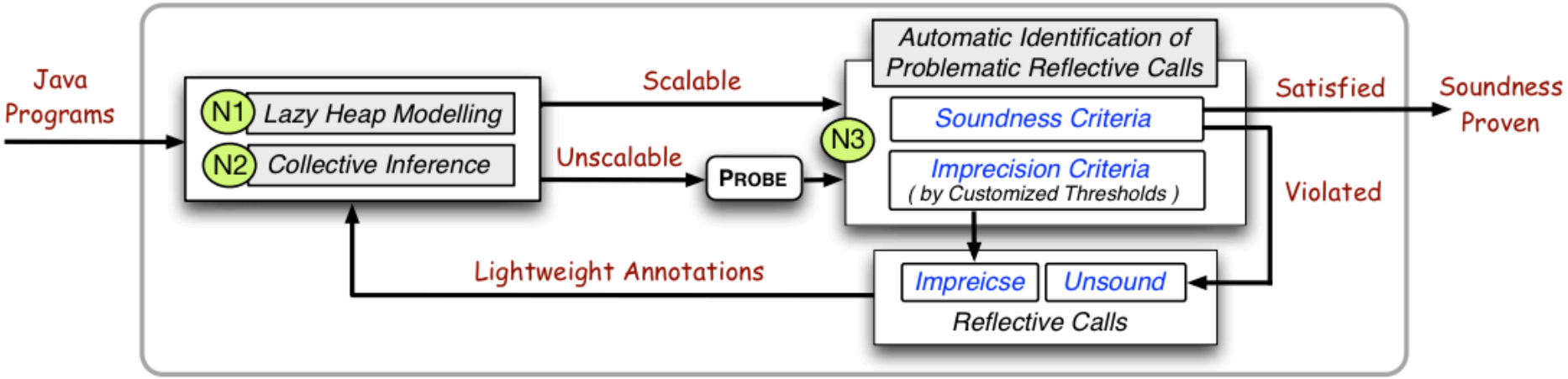
Imprecision Criteria
(by Customized Thresholds)

The number of the reflective targets resolved or inferred

SOLAR


1 More sound
2 Controllable
Java Programs
N1 Lazy Heap Modelling
N2 Collective Inference
Scalable
Unscalable
PROBE
Automatic Identification of Problematic Reflective Calls
N3
Soundness Criteria
Imprecision Criteria
(by Customized Thresholds)
Satisfied
Violated
Soundness Proven
Lightweight Annotations
Impreicse
Unsound
Reflective Calls

SOLAR


Java Programs
N1 Lazy Heap Modelling
N2 Collective Inference
Scalable
Unscalable
PROBE
Automatic Identification of Problematic Reflective Calls
N3
Soundness Criteria
Imprecision Criteria
(by Customized Thresholds)
Satisfied
Soundness Proven
Violated
Lightweight Annotations
Impreicse
Unsound
Reflective Calls

Evaluation

SOLAR vs DOOP ELF

Evaluation

SOLAR VS DOOP ELF

Large real-world Java benchmarks and applications

Large and reflection-rich Java library: JDK 1.6

1 More sound

Recall

Recall: measured by the number of **true** reflective targets discovered at reflective call sites that are **dynamically executed** under **certain inputs**

1 More sound

Recall

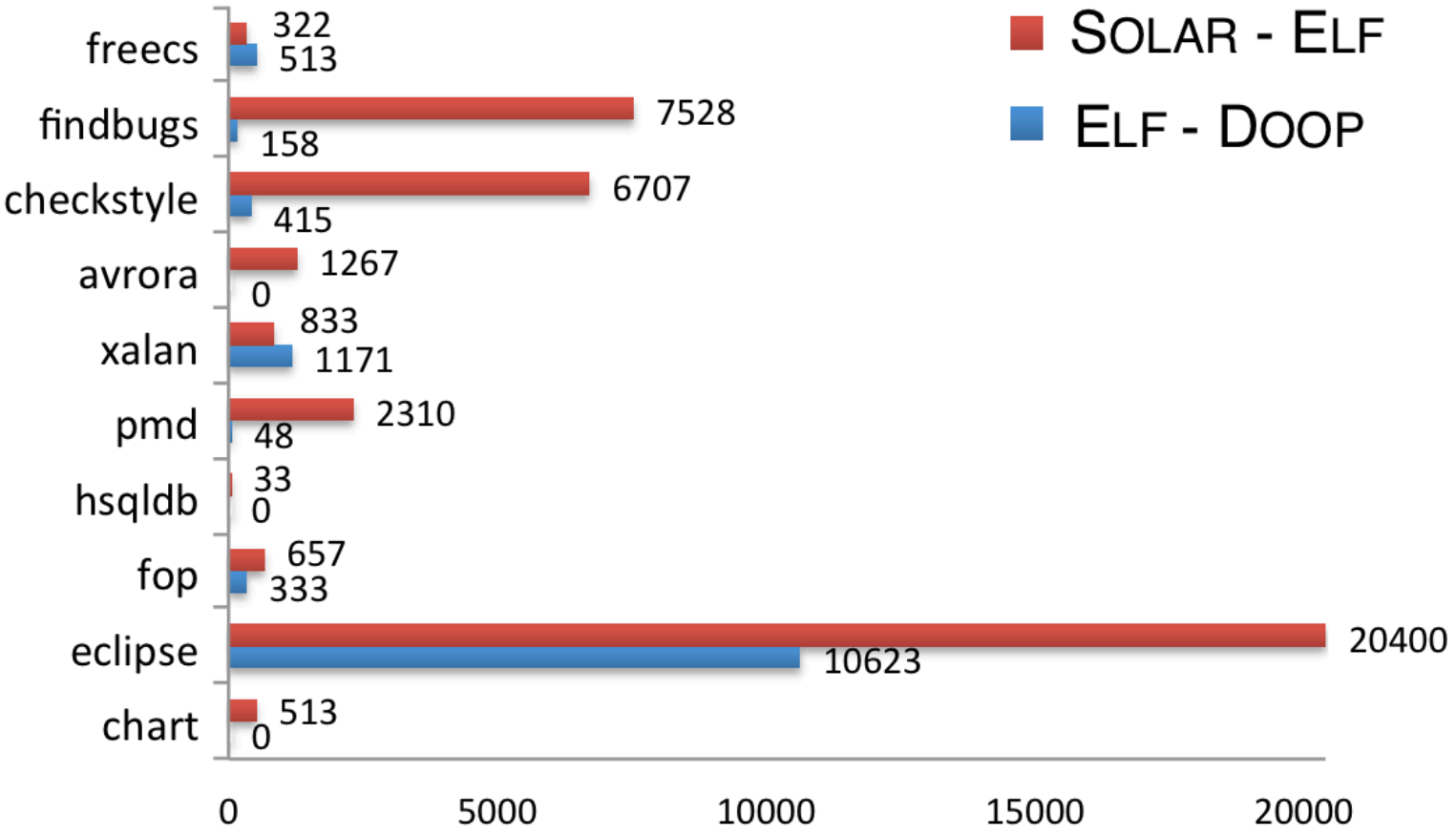
Recall: measured by the number of **true** reflective targets discovered at reflective call sites that are **dynamically executed** under **certain inputs**

Only Solar achieves total recall :
all the true reflective targets found in recall are resolved by Solar.

The benefit of achieving higher recall:
more true call graph edges discovered

The benefit of achieving higher recall: more true call graph edges discovered

The true call graph edges are computed by instrumentation at runtime



*The figure shows the more true call graph edges found in recall by Solar than Elf **(Solar - Elf)** and by Elf than Doop **(Elf - Doop)***

1 More sound

Precision

Insight: Soundness ←→ precision.
Solar achieves higher recall (more sound), indicates worse precision ?

1 More sound

Precision

Insight: Soundness ←→ precision.
Solar achieves higher recall (more sound), indicates worse precision ?

No!
Solar maintains nearly the same precision as Doop and Elf (2 popular clients).

Precision

		chart	eclipse	fop	hsqldb	pmd	xalan	avrora	checkstyle	findbugs	freecs	**Average**
Devir	DOOP	–	94.94	93.04	–	92.65	93.49	94.79	93.16	92.32	95.46	93.72
Call	ELF	93.53	88.07	92.34	94.80	92.87	92.70	94.50	93.19	92.53	94.94	92.93
(%)	SOLAR	93.51	87.69	92.26	94.51	92.39	92.65	92.43	93.39	92.37	95.26	92.63
Safe	DOOP	–	59.34	53.68	–	45.40	57.97	56.12	50.19	45.78	59.71	53.24
Cast	ELF	49.80	40.71	55.40	53.65	48.24	59.24	57.27	51.79	48.54	59.14	52.07
(%)	SOLAR	49.53	38.04	54.21	53.11	44.53	59.11	52.56	49.40	43.60	57.96	49.79

Devir Call: the percentage of the virtual calls whose targets can be disambiguated
Safe Case: the percentage of the casts that can be statically shown to be safe

2 Controllable

2 Controllable

In 10 evaluated programs, 7 can be analyzed scalably and soundly by Solar with full automation.

2 Controllable

In 10 evaluated programs, 7 can be analyzed scalably and soundly by Solar with full automation.

For the remaining 3 programs, Probe is scalable and reports 13 unsound calls and 1 imprecise call.

2 Controllable

In 10 evaluated programs, 7 can be analyzed scalably and soundly by Solar with full automation.

For the remaining 3 programs, Probe is scalable and reports 13 unsound calls and 1 imprecise call.

After manual check, all the identified 14 unsound/imprecise calls are the true ones.

2 Controllable

In 10 evaluated programs, 7 can be analyzed scalably and soundly by Solar with full automation.

For the remaining 3 programs, Probe is scalable and reports 13 unsound calls and 1 imprecise call.

After manual check, all the identified 14 unsound/imprecise calls are the true ones.

Probe also reports 7 corresponding annotation points for these 14 unsound/imprecise calls

2 Controllable

In 10 evaluated programs, 7 can be analyzed scalably and soundly by Solar with full automation.

For the remaining 3 programs, Probe is scalable and reports 13 unsound calls and 1 imprecise call.

After manual check, all the identified 14 unsound/imprecise calls are the true ones.

Probe also reports 7 corresponding annotation points for these 14 unsound/imprecise calls

After the 7 light-weight annotations, Solar can analyze these 3 programs scalably and soundly

Performance

	chart	eclipse	fop	hsqldb	pmd	xalan	avrora	checkstyle	findbugs	freecs	**Average**
DOOP	–	321	779	–	226	254	188	256	718	422	–
ELF	3434	5496	2821	1765	1363	1432	932	1463	2281	1259	1930
SOLAR	4543	10743	4303	2695	2156	1701	3551	2256	8489	2880	3638

1. More sound
2. Controllable

Potential Impact

1 More sound

Reflection Analysis

Potential Impact

1 More sound

SOLAR

Potential Impact

1 More sound

SOLAR

Hundreds of Papers & Tools

Nothing needs to change

Potential Impact

1 More sound

SOLAR

Hundreds of Papers & Tools

Nothing needs to change

More bugs, etc.

Potential Impact

1 More sound

SOLAR → *Hundreds of Papers & Tools*

Nothing needs to change

More bugs, etc.

2 Controllable

- A static bug verification tool reports no bugs.

  4 reflective calls are identified unsoundly analyzed by Solar

Potential Impact

1 More sound

SOLAR → *Hundreds of Papers & Tools*

Nothing needs to change

More bugs, etc.

2 Controllable

- A static bug verification tool reports no bugs.

  4 reflective calls are identified unsoundly analyzed by Solar

- A static bug detection tool reports 10 bugs.

  All reflective calls are reported soundly analyzed by Solar

http://www.cse.unsw.edu.au/~corg/solar

SOLAR - Effective Soundness-Guided Reflection Analysis

Authors

Yue Li Tian Tan Jingling Xue

CORG
COmpiler Research Group@UNSW

Description

SOLAR is a static analysis framework that strives to automate sound reflection analysis for Java programs (under some assumptions) introduced in our paper titled "Effective Soundness-Guided Reflection Analysis", SAS'2015. SOLAR can identify the places in a program where reflection is resolved unsoundly or imprecisely, enabling lightweight annotations to improve the quality of analysis and make the analysis controllable.

SOLAR is implemented on top of ELF. To ease the understanding of the Datalog rules used in our implementation, we have rewritten the rules inherited from ELF and added new ones in a uniform manner. Users are expected to understand how different parts of the Java reflection API are handled easily and precisely when applying SOLAR to analyse their applications.

SOLAR can output its reflection analysis results with the format that is supported by Soot. You can let Soot receive the results of SOLAR easily by following the instructions in the tutorial.

Downloads

The tar.gz file includes four important files:

- tutorial: a step-by-step guide to installing SOLAR on top of DOOP (version r160113) by using the two patch files (available below). The main differences between SOLAR and DOOP are also summarised.
- solar.patch: a patch file for updating the Datalog rules and some auxiliary scripts used in DOOP.
- gen.patch: a patch file for updating the fact generator (version r958) provided by DOOP.
- _Unknown_.class: an auxiliary class which is used to introduce the Unknown type into the existing type system. In addition, this class will help SOLAR soundly model the pointer-affecting methods, such as getClass() and toString(), which are directly invoked on the unknown objects. See the comments in the _Unknown_.java file for details.

- SOLAR-0.1.tar.gz

Acknowledgements

The authors wish to thank the DOOP team for making DOOP available, and LogicBlox Inc. for providing us its Datalog engine.

Static analysis for OO in practice ?

Static analysis for OO in practice ?

Reflection

Thank You

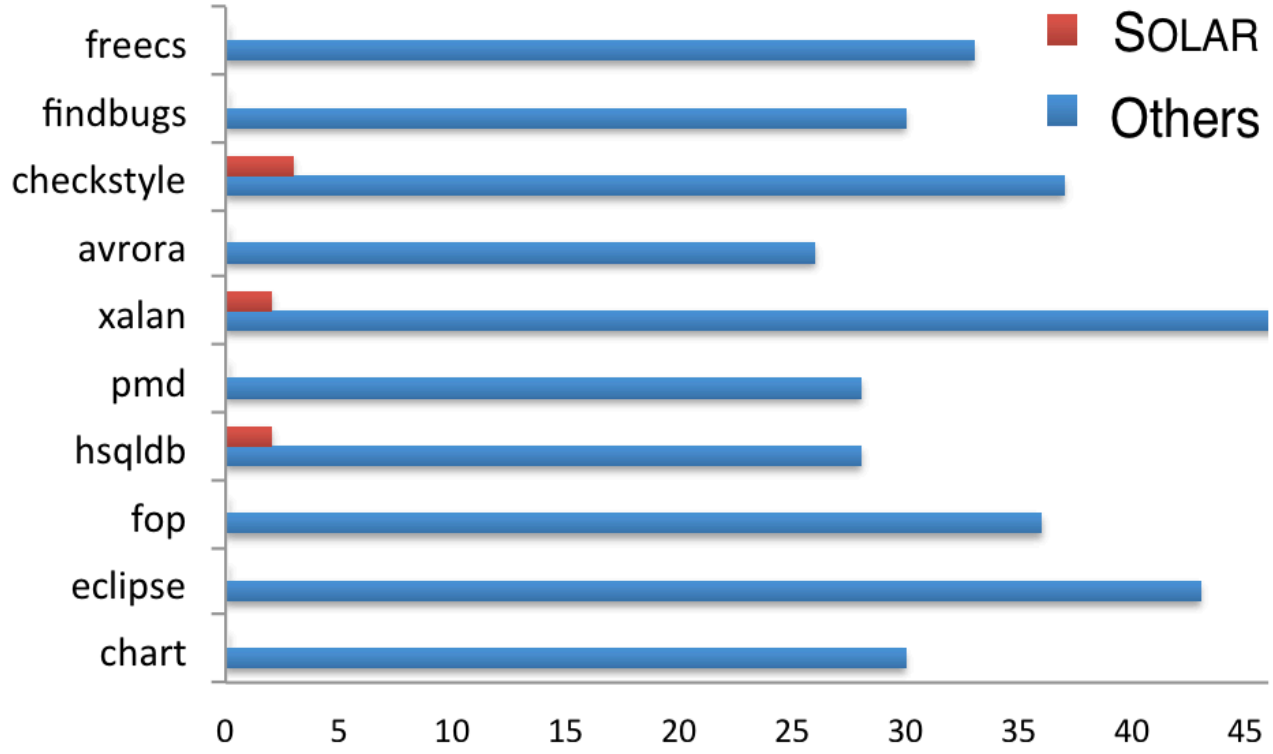
Yue Li

CORG @ UNSW, Australia

September 10, 2015

1. More sound
2. Controllable

→ Light-weight Annotations



The number of annotations required for improving the soundness of unsoundly resolved reflective calls. Others: 338 vs Solar: 7