



Cooperative Program Analysis

Bor-Yuh Evan Chang
University of Colorado Boulder

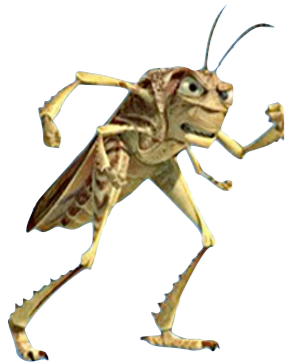


Colorado State University
September 22, 2014





A program analysis story ...







Software is everywhere and varying more and more



traditional



Software is everywhere and varying more and more



traditional



mobile



Software is everywhere and varying more and more



traditional



mobile



cloud



Software is everywhere and varying more and more



traditional



mobile



cloud



cyberphysical

Software is everywhere and varying more and more



traditional



mobile



cloud



cyberphysical

Software is everywhere and varying more and more

Software is getting more and more complex



traditional



mobile



cloud



cyberphysical

Software is everywhere and varying more and more

Software is getting more and more complex





1980s: Bug in Therac-25 kills 6

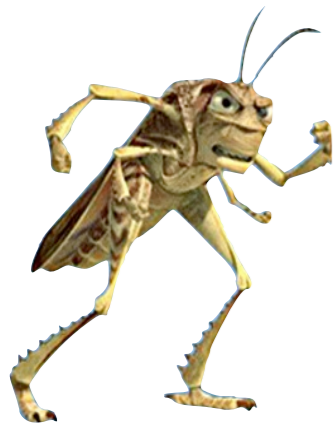




1980s: Bug in Therac-25 kills 6

2000s: Conficker worm costs \$9.1 billion in damages





1980s: Bug in Therac-25 kills 6

2000s: Conficker worm costs \$9.1 billion in damages

Today: "Don't buy this app, it crashes."

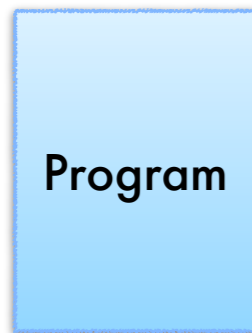




**Program
Analysis**



Program Analysis for Formal Verification



**Systematically examine the program to
“simulate” running it on “all inputs”**



Program Analysis for Formal Verification



**Systematically examine the program to
"simulate" running it on "all inputs"**



Program Analysis for Formal Verification



Systematically examine the program to
"simulate" running it on "all inputs"



Program Analysis for Formal Verification



Systematically examine the program to "simulate" running it on "all inputs"



The End?



Program Analysis for Formal Verification



Systematically examine the program to
"simulate" running it on "all inputs"



Program Analysis for Formal Verification



Systematically examine the program to
"simulate" running it on "all inputs"



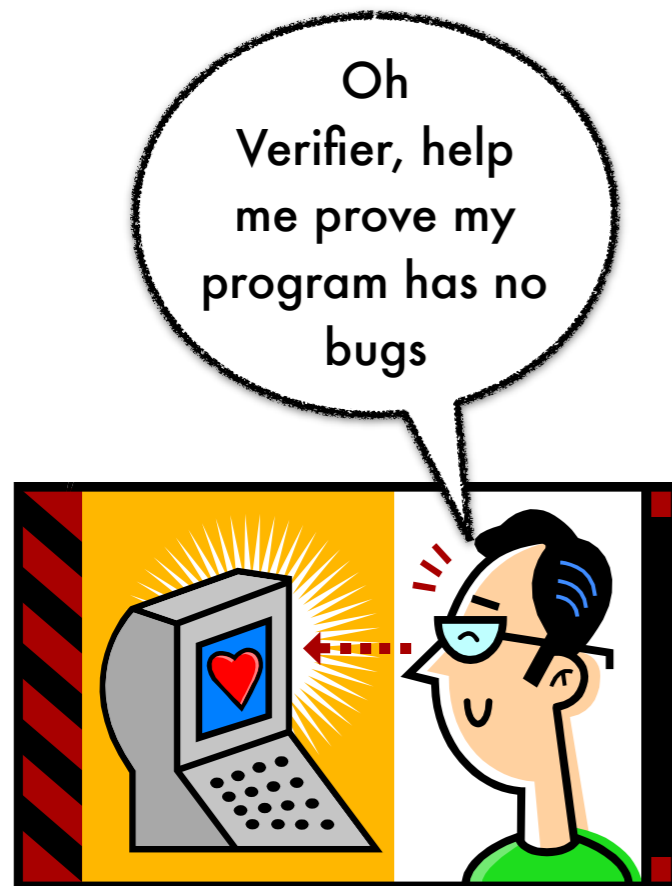
Program Analysis for Formal Verification



Undecidability necessitates the possibility of **false alarms**. We hope not too many.

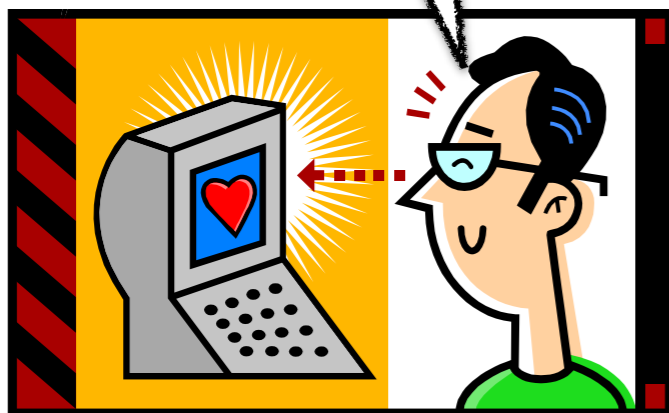
Uncooperative Program Analysis?

Uncooperative Program Analysis?



Uncooperative Program Analysis?

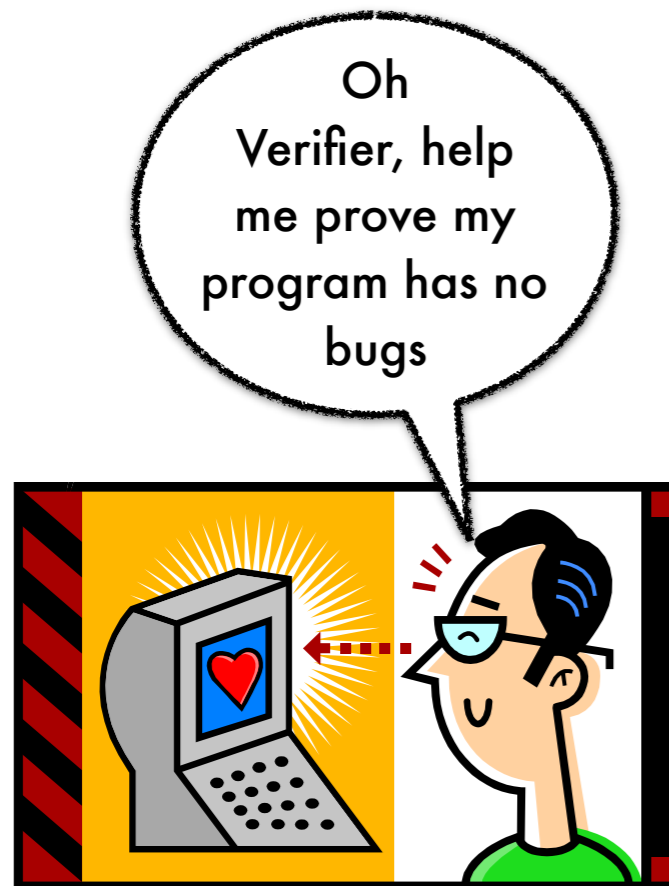
Oh
Verifier, help
me prove my
program has no
bugs



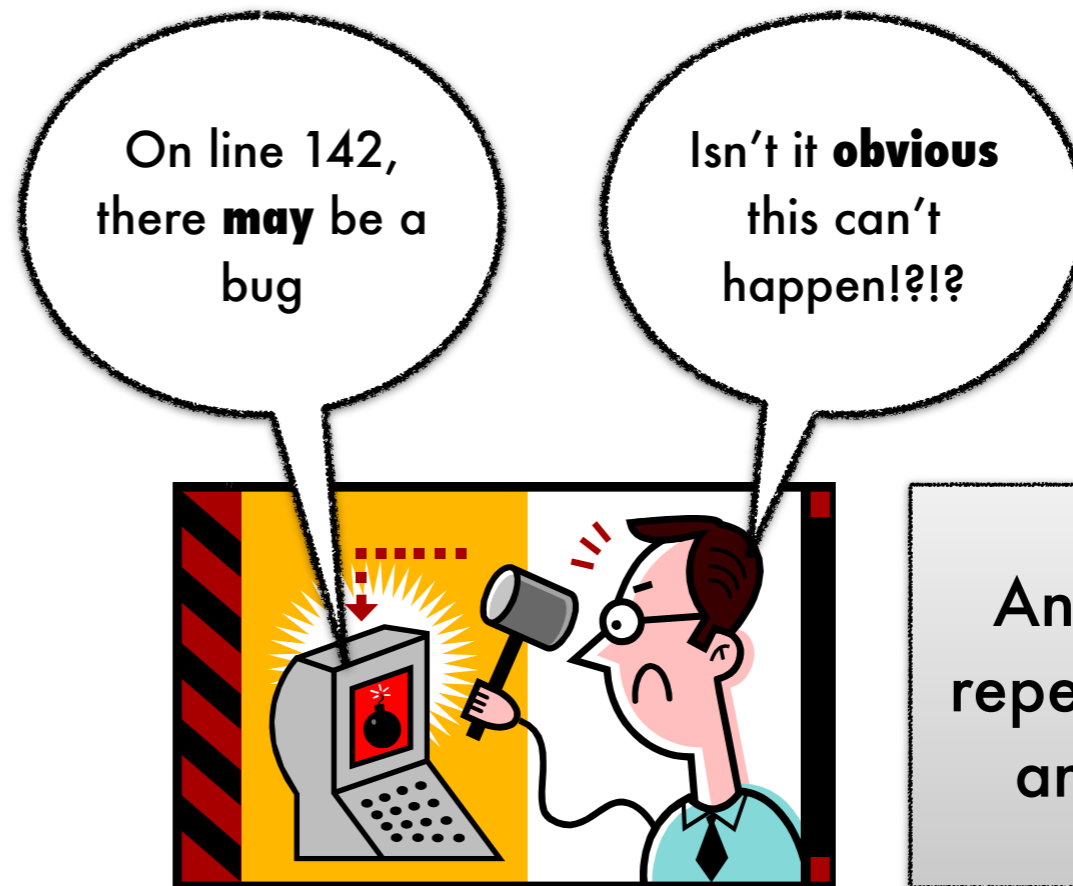
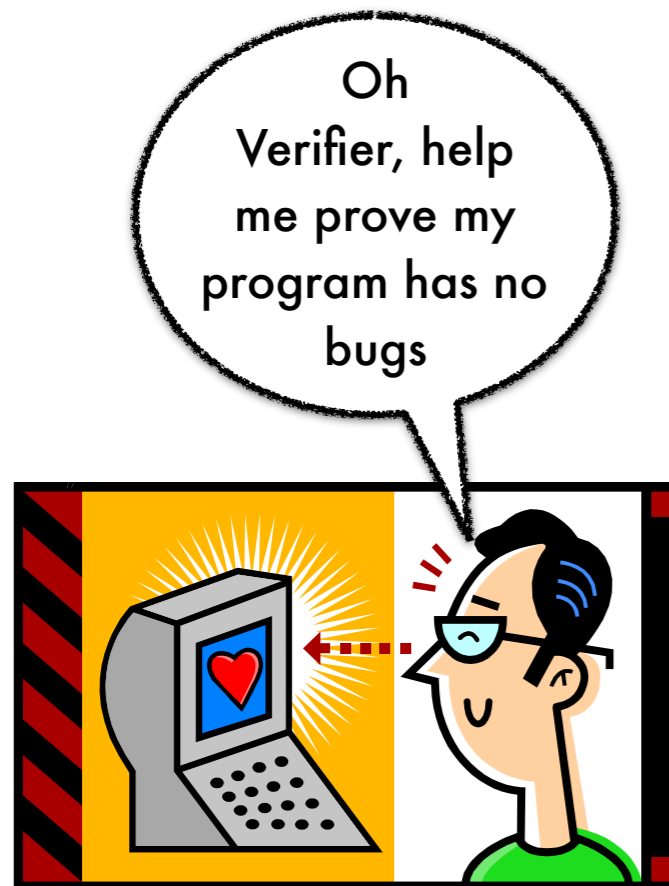
On line 142,
there **may** be a
bug



Uncooperative Program Analysis?

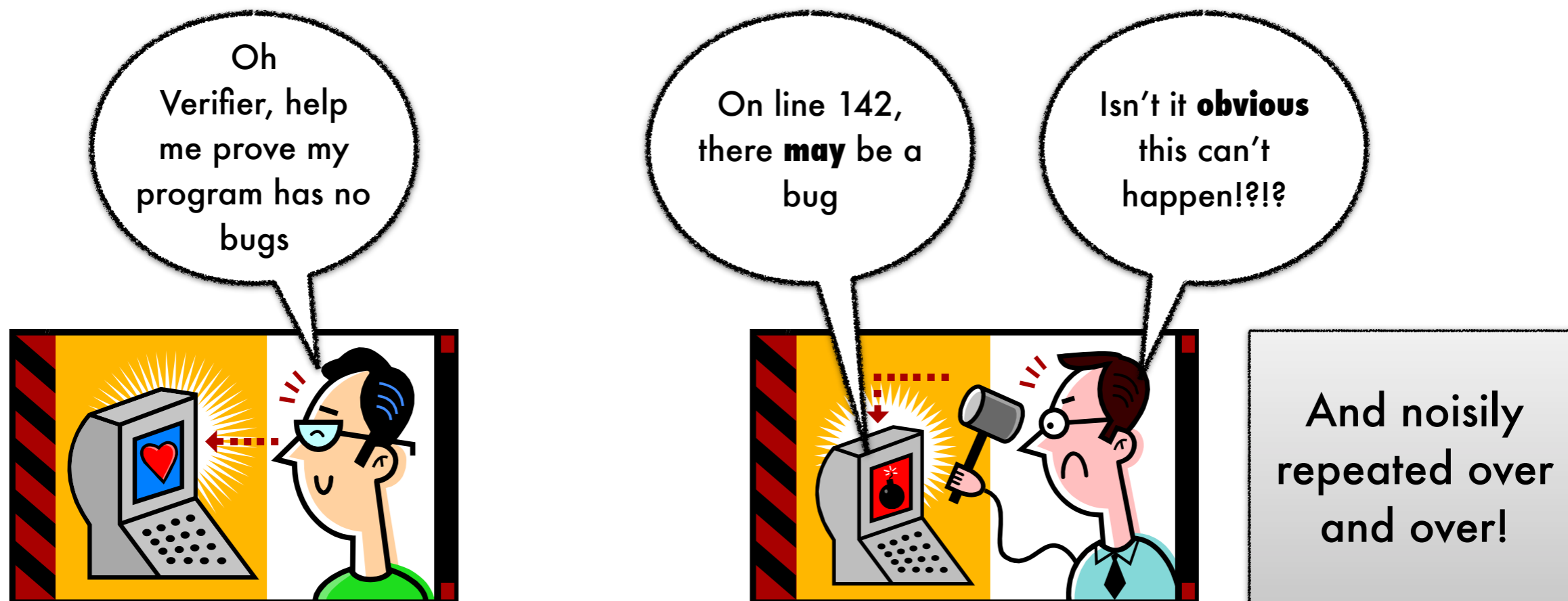


Uncooperative Program Analysis?



And noisily
repeated over
and over!

Uncooperative Program Analysis?



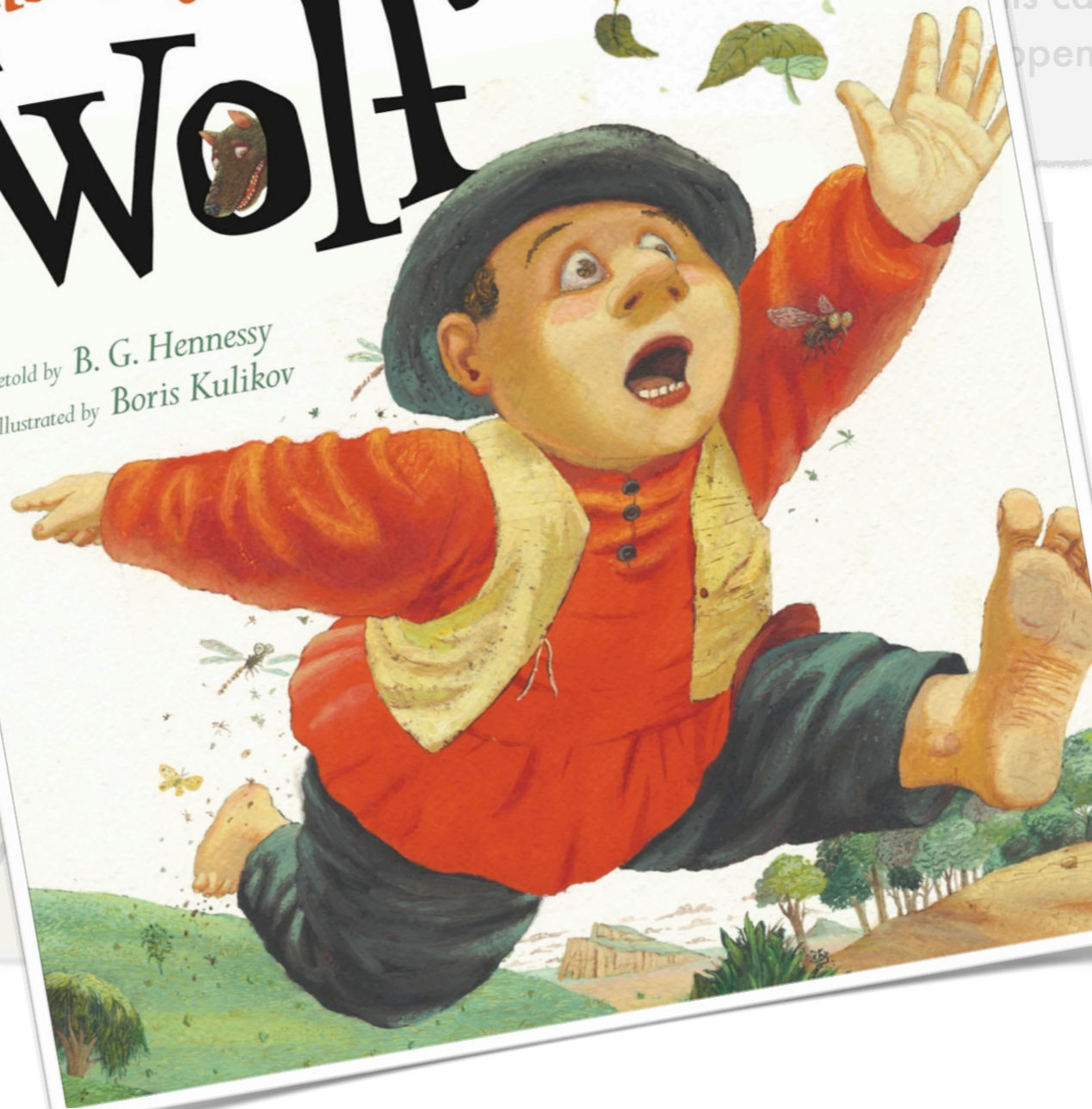
The well-known **false alarm** problem

Uncooperative Program Analysis?

Oh
Verified
me pro
program
bug

The Boy Who Cried Wolf

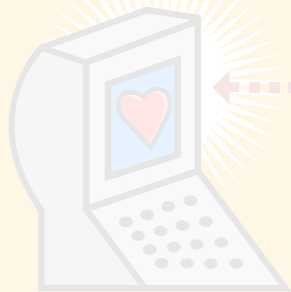
Retold by B. G. Hennessy
Illustrated by Boris Kulikov



It's obvious
this can't
open!?!?

And noisily
repeated over
and over!

The w



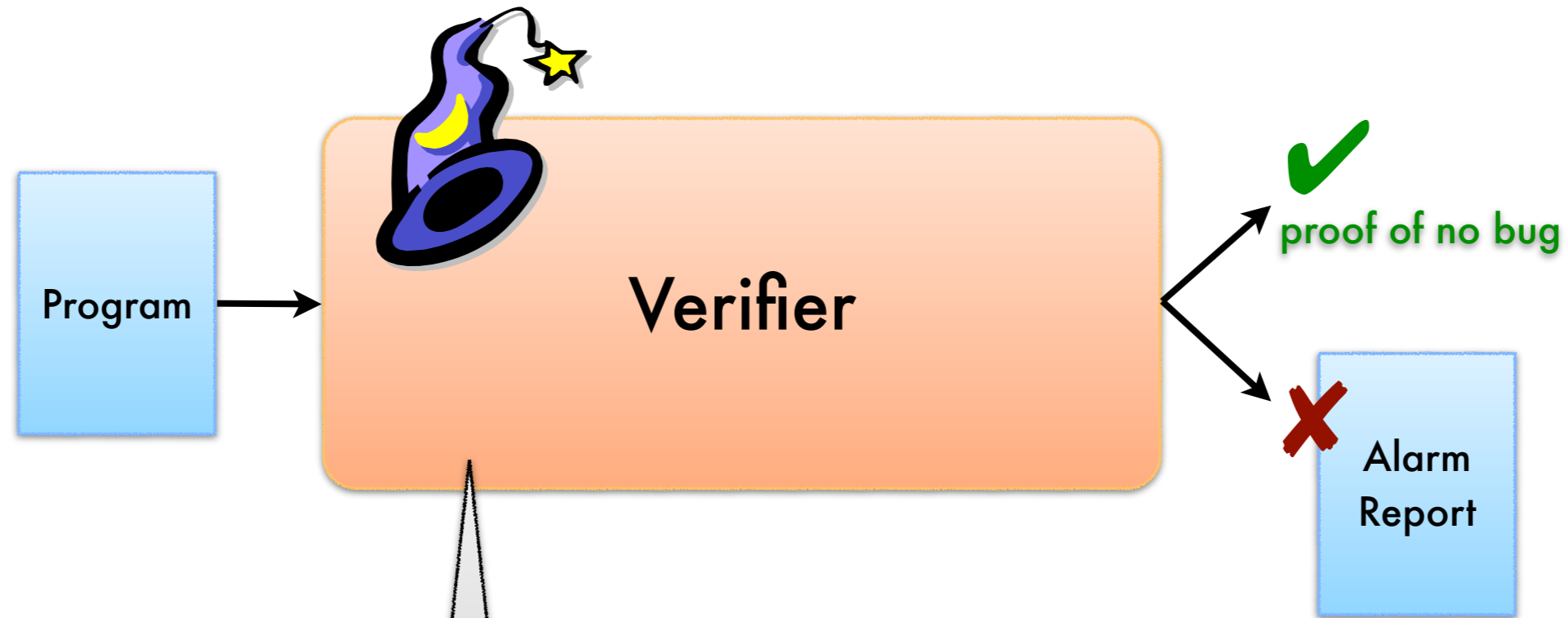
“[M]ore than a 30% [false alarm rate] easily causes problems. True bugs get lost in the false. A vicious cycle starts where low trust causes complex [true] bugs to be labeled false [alarms], leading to yet lower trust.”

“A stupid false [alarm] implies the tool is stupid.”

The traditional approach to the false alarm problem focuses on improving the verifier.

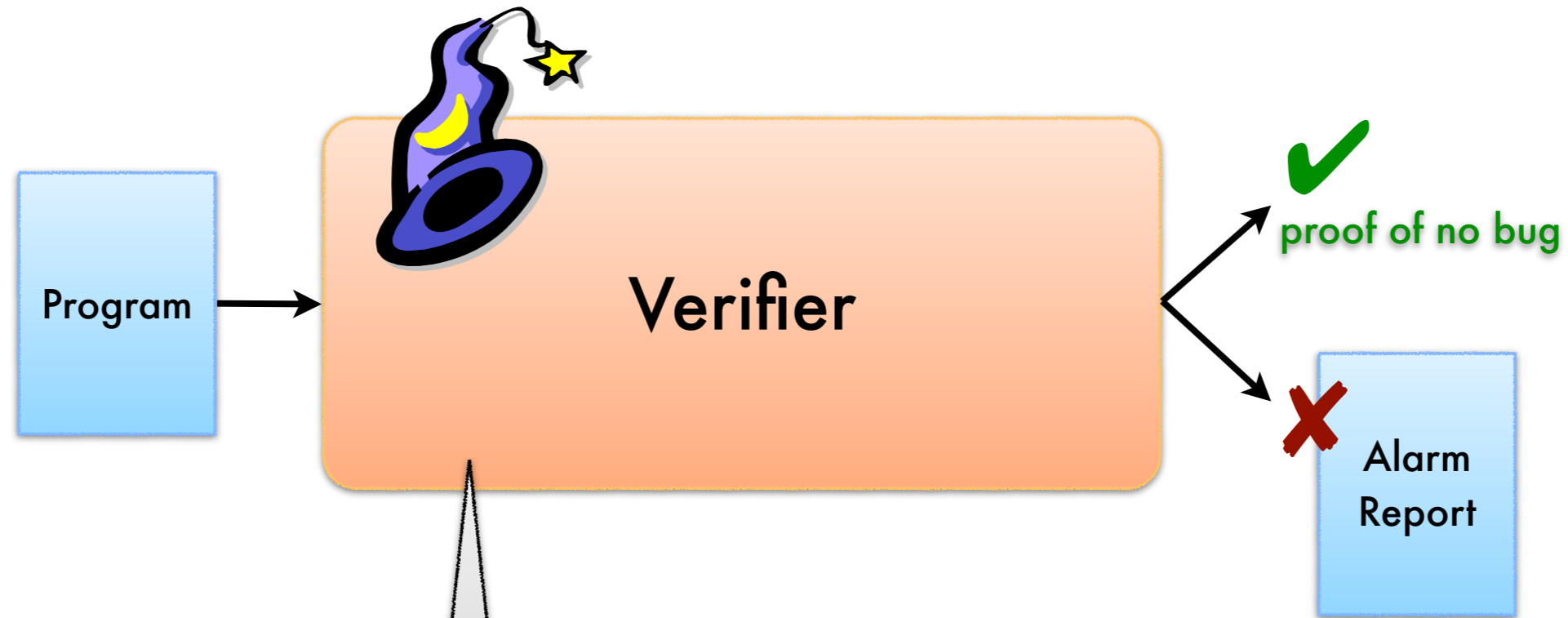


The traditional approach to the false alarm problem focuses on improving the verifier.



Redesign the verifier with more magic to hopefully reduce the number of false alarms

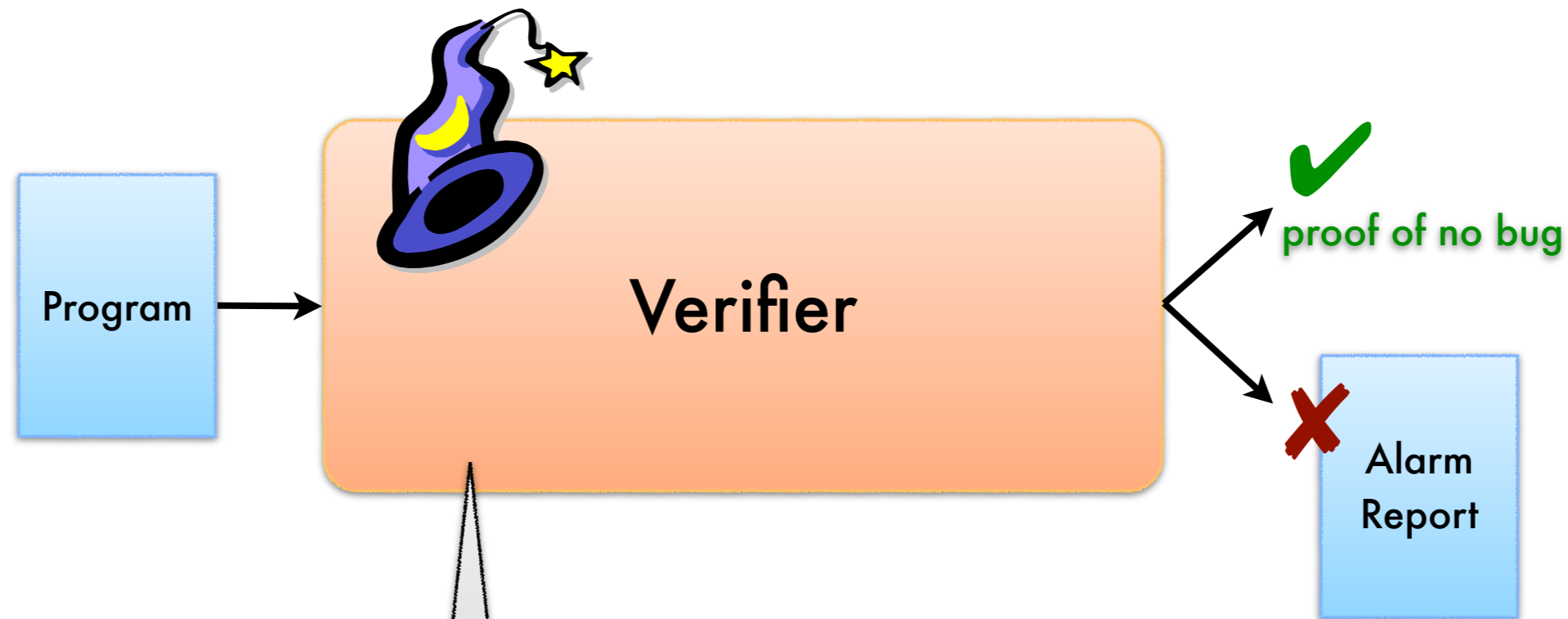
The traditional approach to the false alarm problem focuses on improving the verifier.



Redesign the verifier with more magic to hopefully reduce the number of false alarms

But it can never be perfect (undecidability)

The traditional approach to the false alarm problem focuses on improving the verifier.



Redesign the verifier with more magic to hopefully reduce the number of false alarms

But it can never be perfect (undecidability)

Also not a sufficient "excuse"

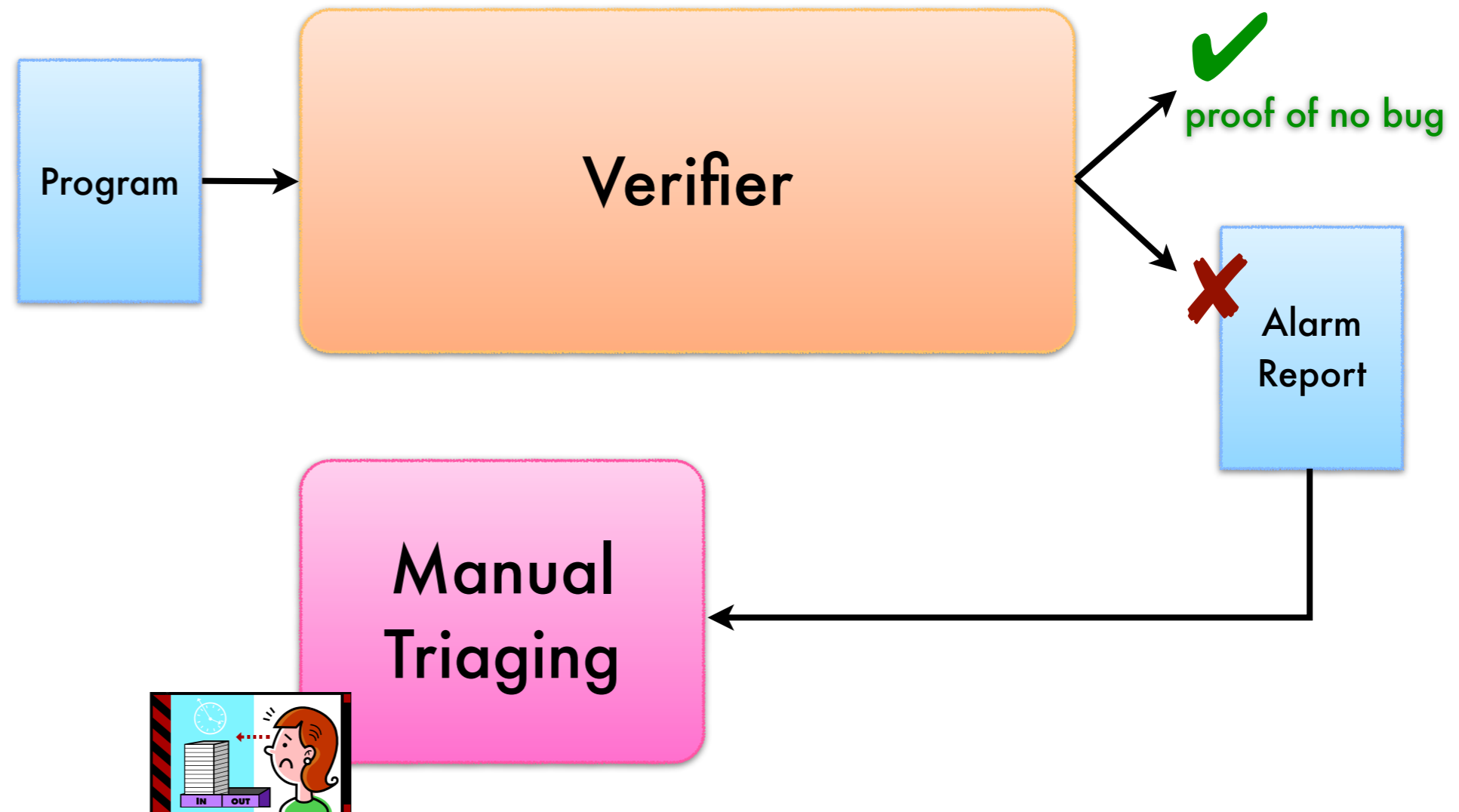
Agenda: The cooperative approach addresses the whole bug mitigation process.



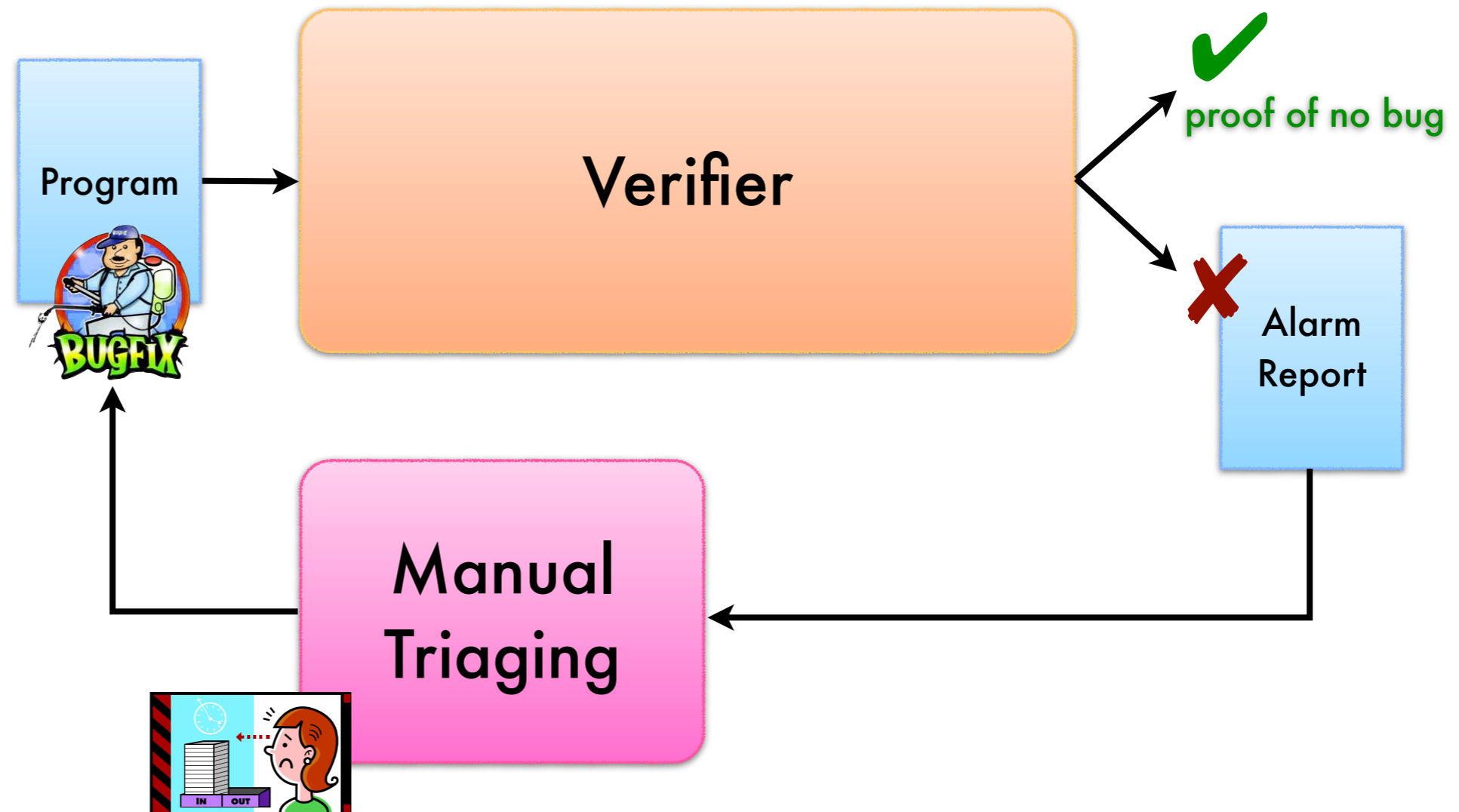
Agenda: The cooperative approach addresses the whole bug mitigation process.



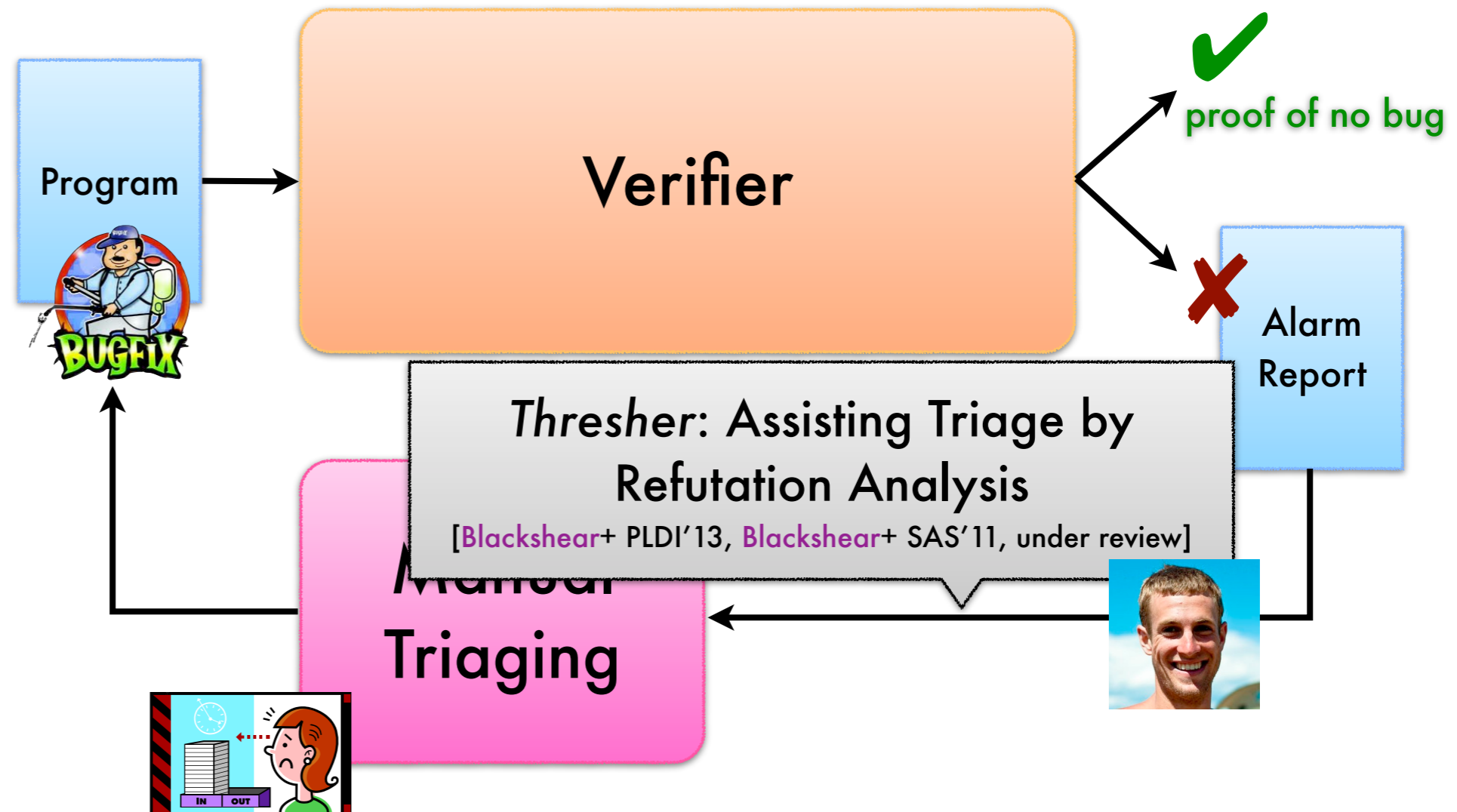
Agenda: The cooperative approach addresses the whole bug mitigation process.



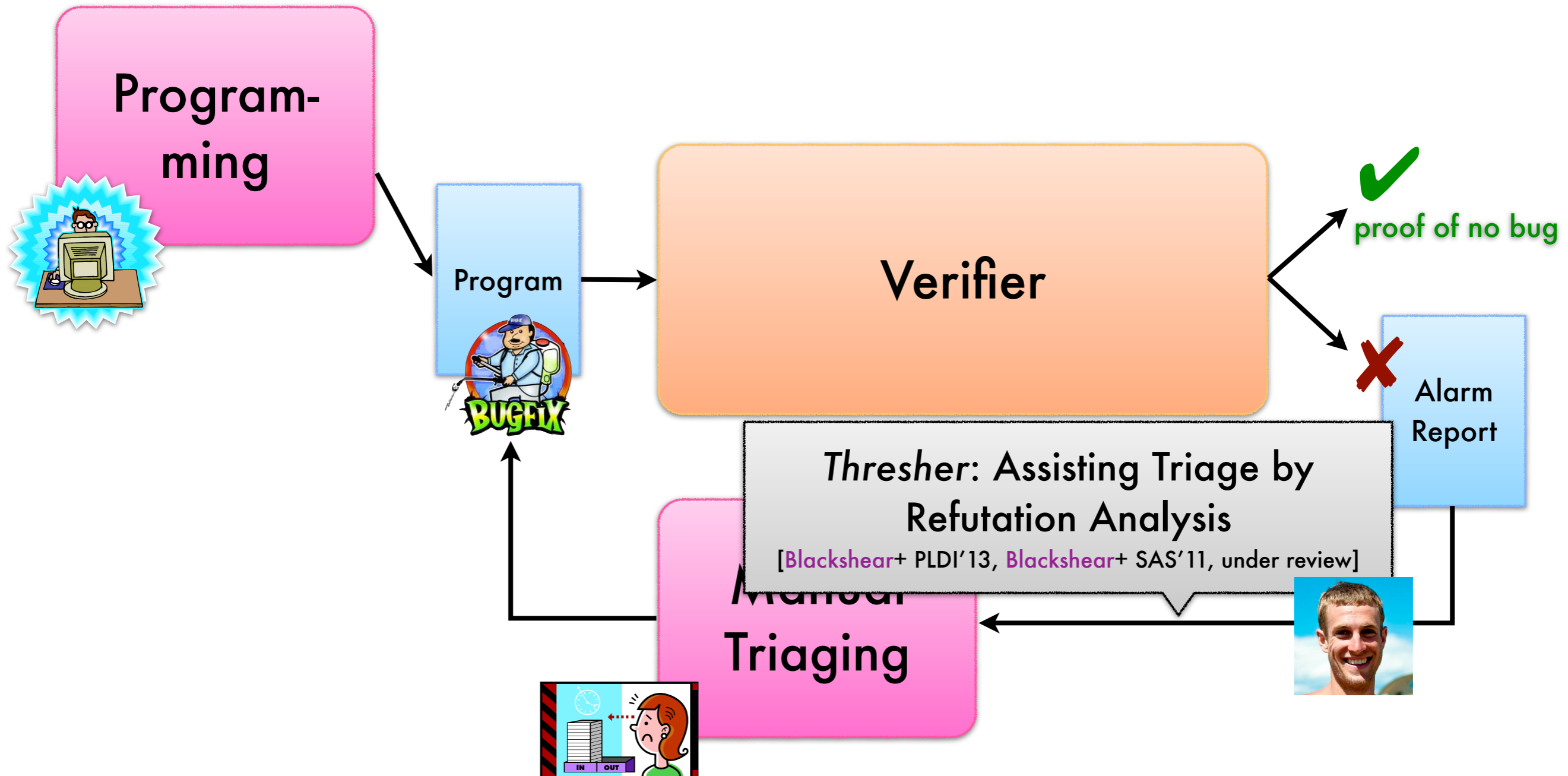
Agenda: The cooperative approach addresses the whole bug mitigation process.



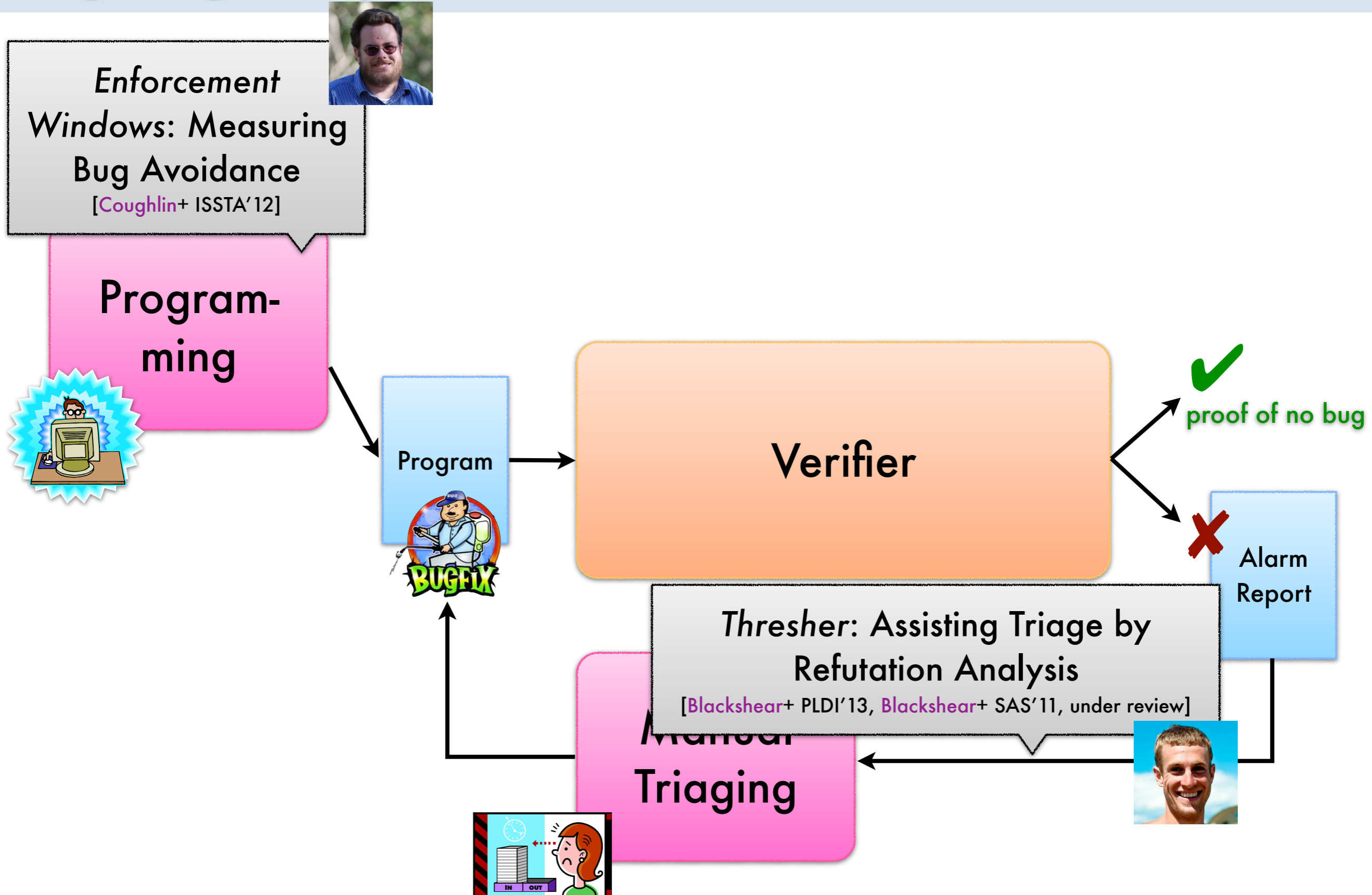
Agenda: The cooperative approach addresses the whole bug mitigation process.



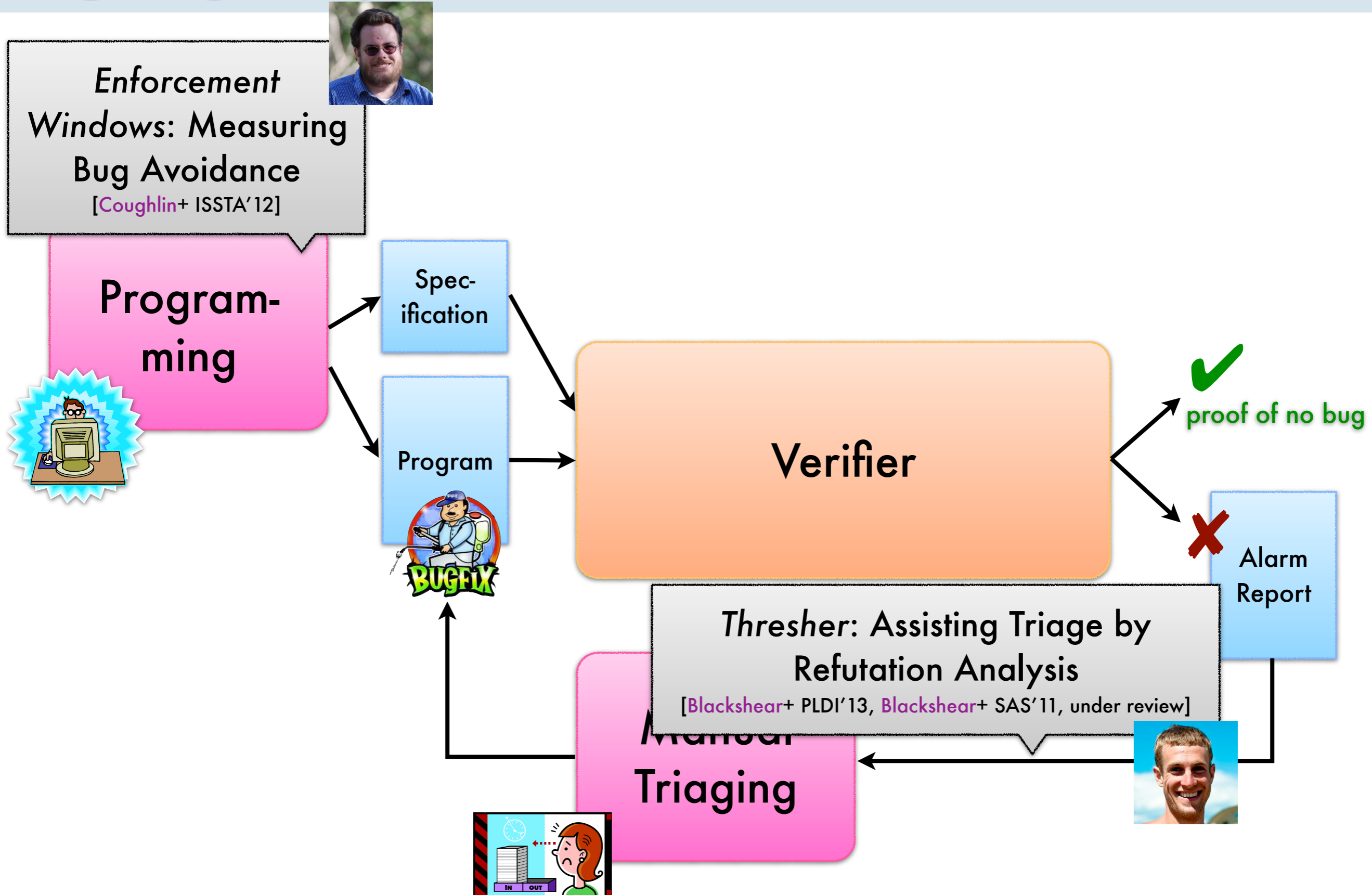
Agenda: The cooperative approach addresses the whole bug mitigation process.



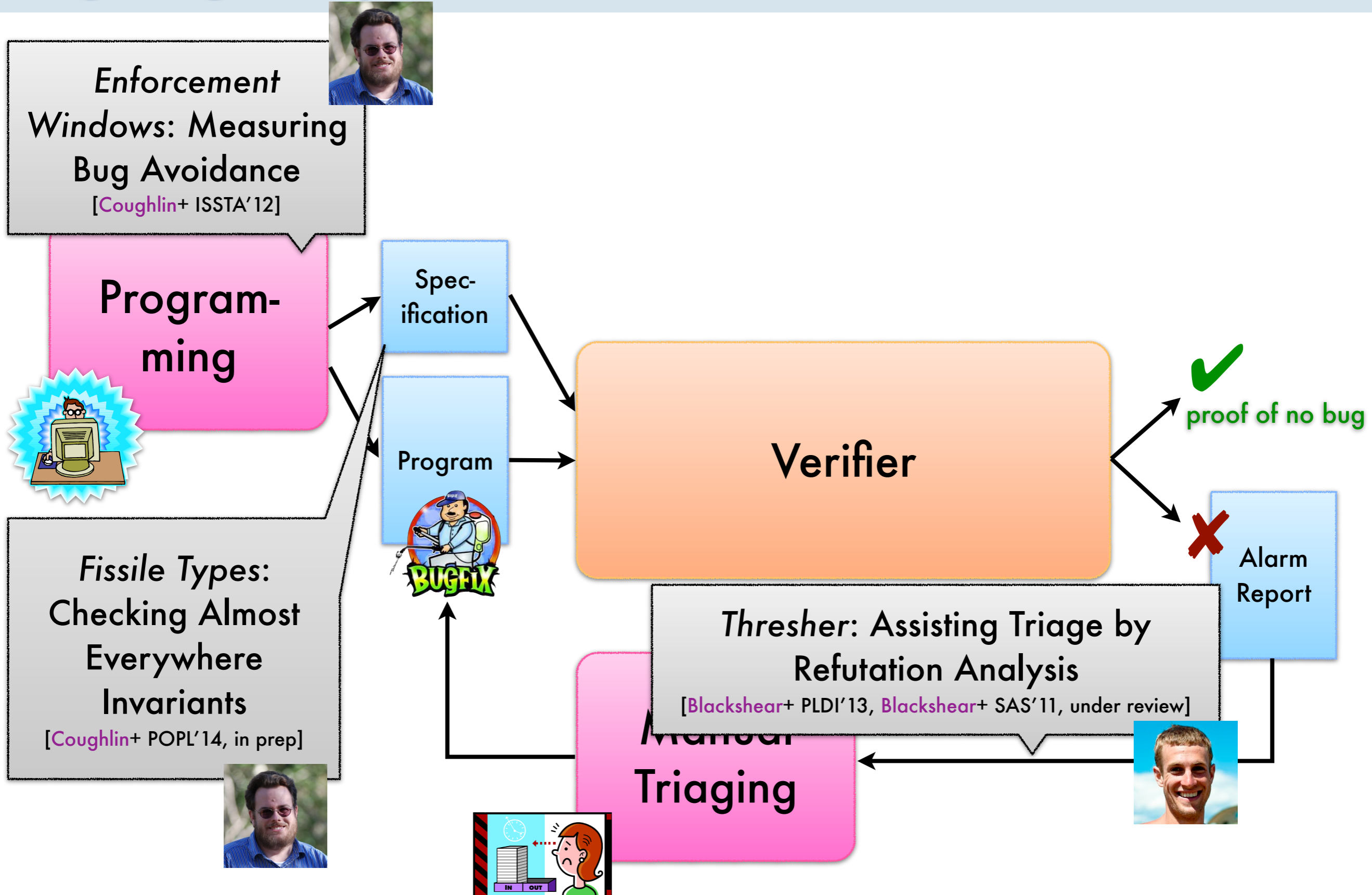
Agenda: The cooperative approach addresses the whole bug mitigation process.



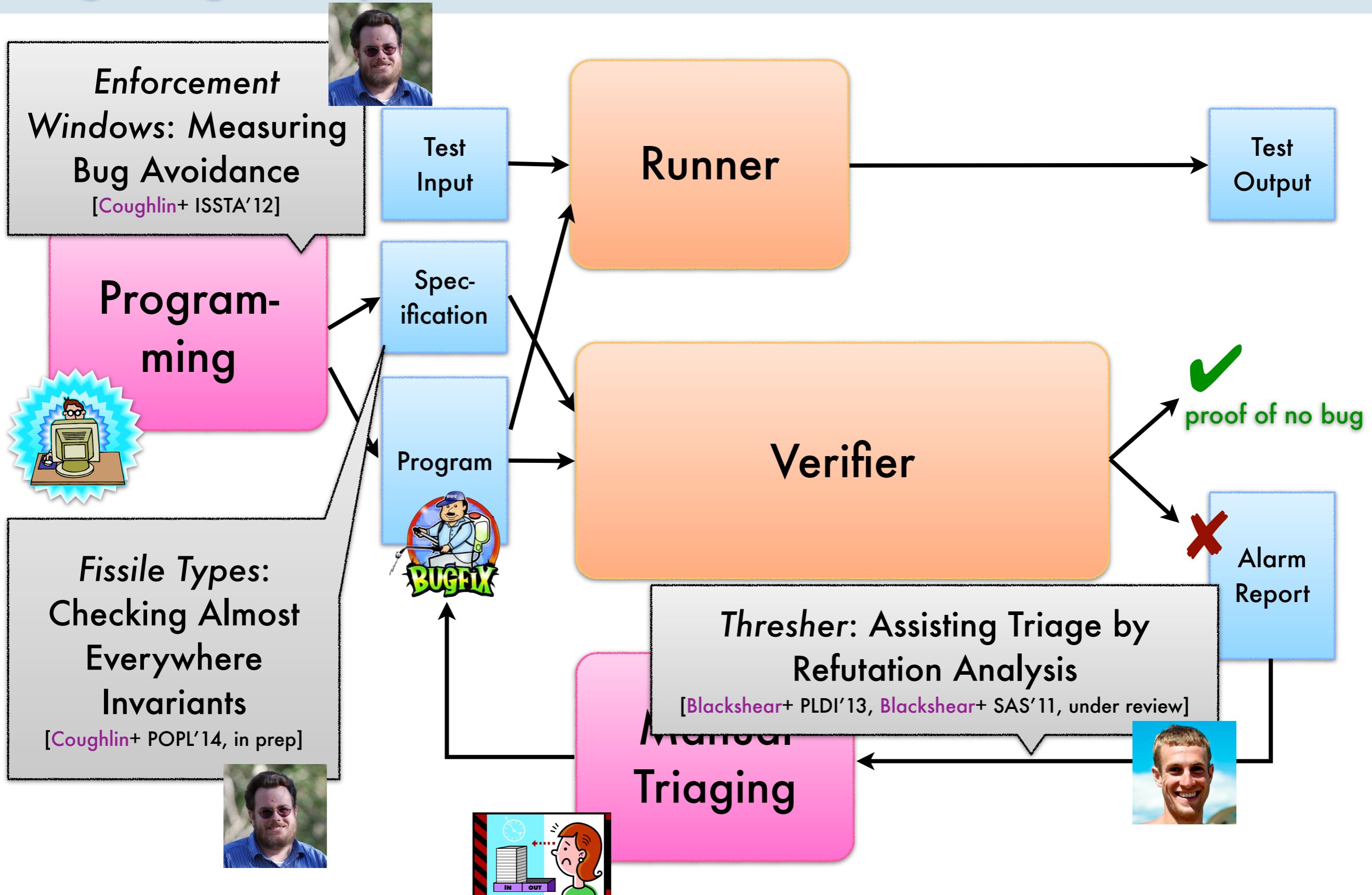
Agenda: The cooperative approach addresses the whole bug mitigation process.



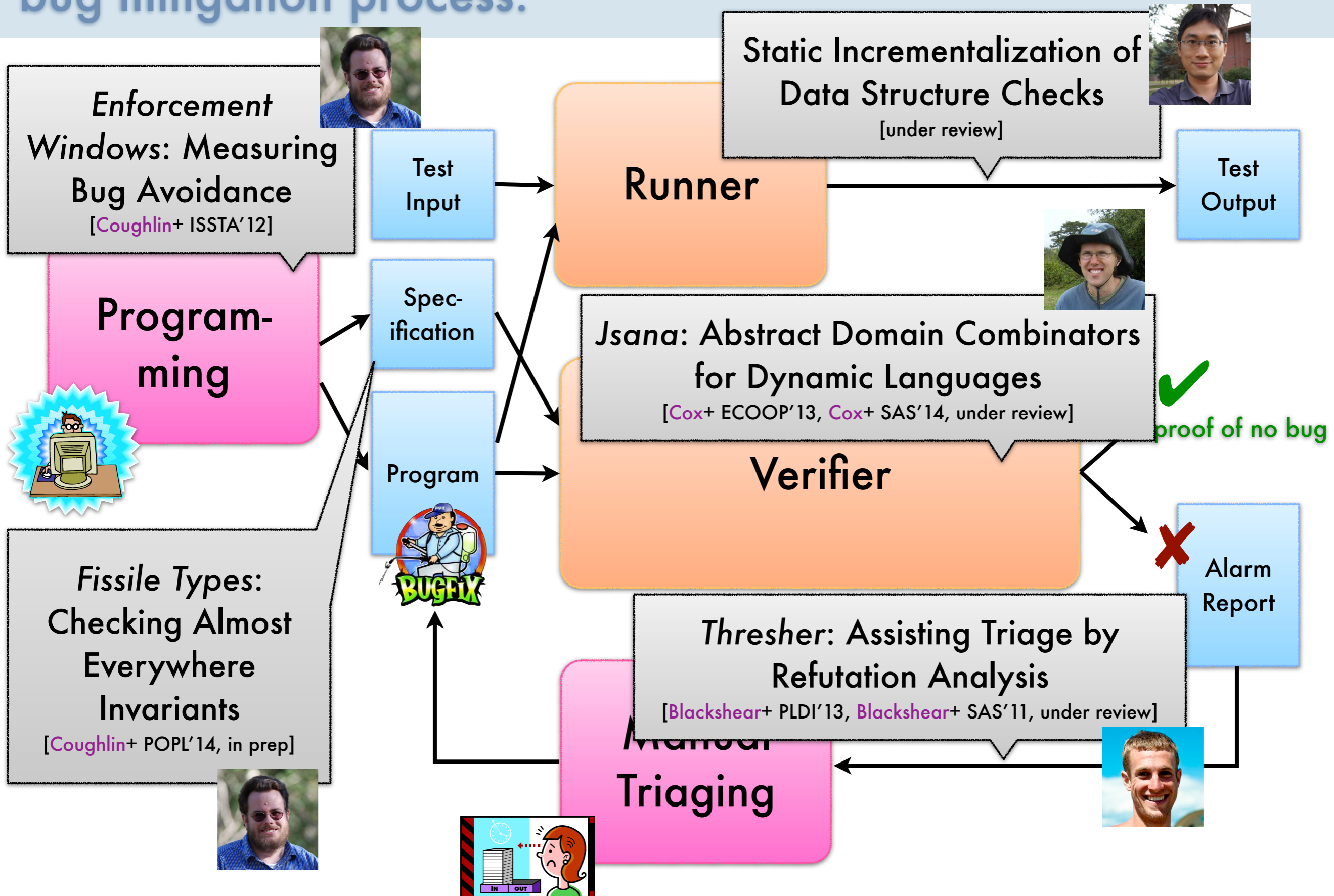
Agenda: The cooperative approach addresses the whole bug mitigation process.



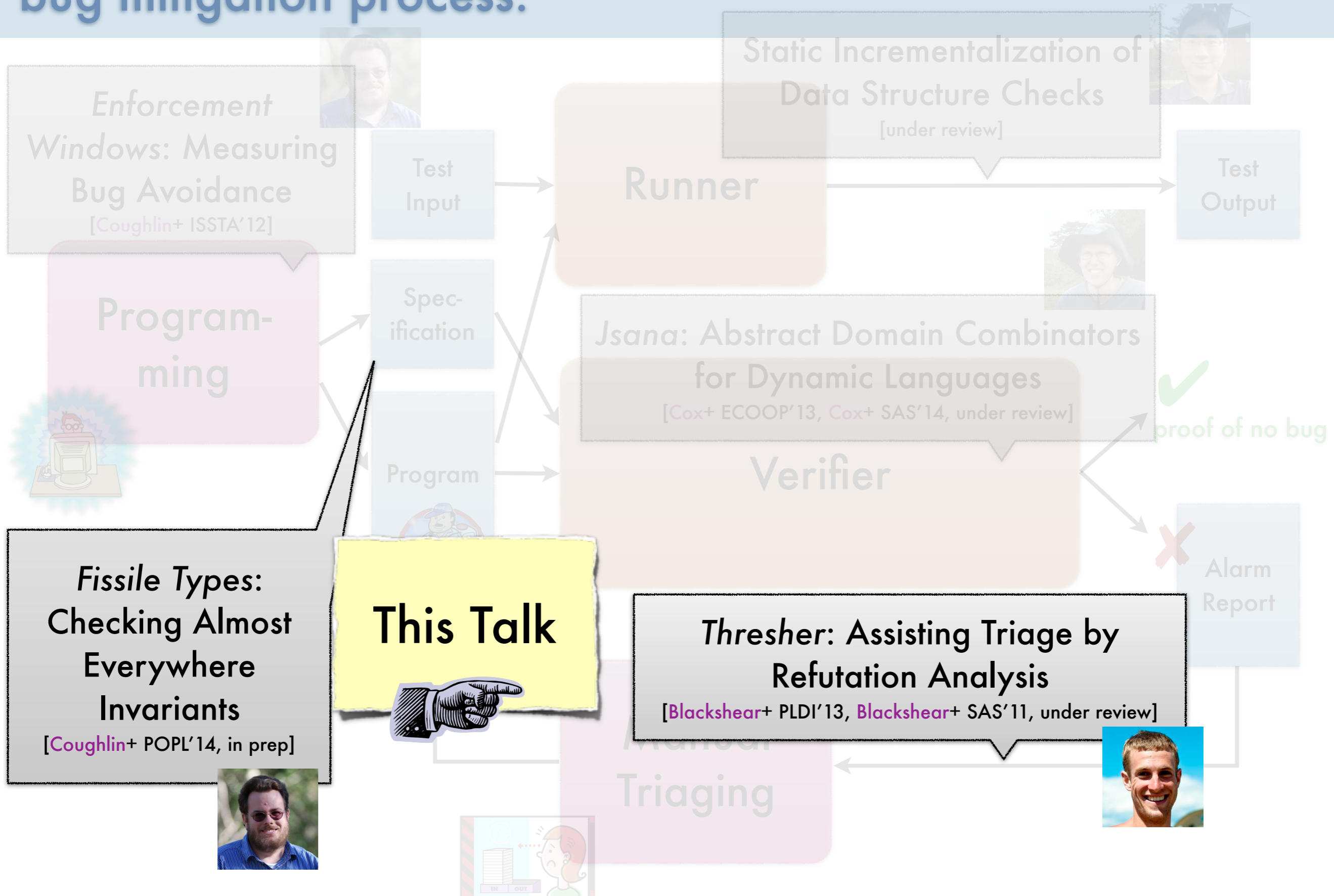
Agenda: The cooperative approach addresses the whole bug mitigation process.



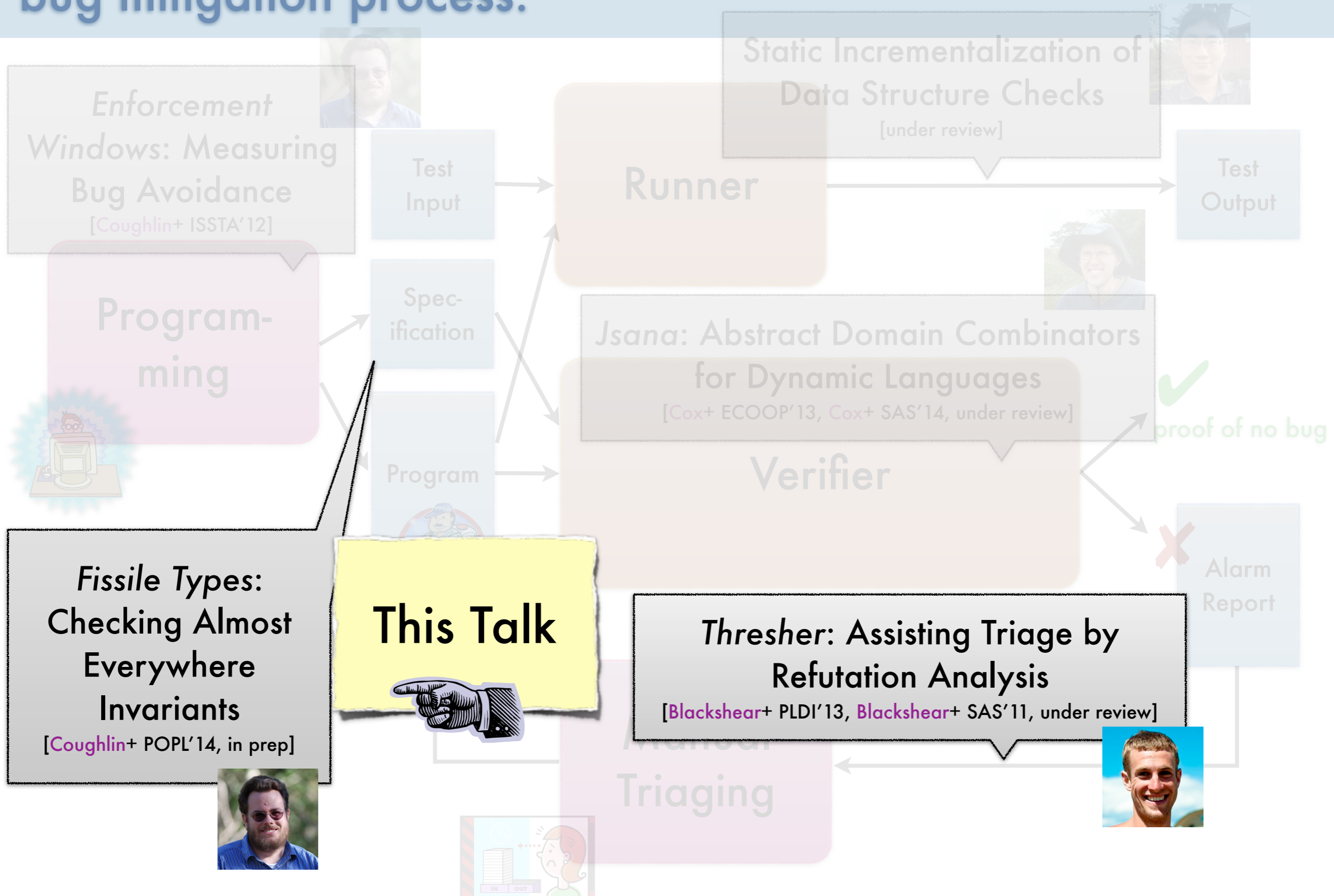
Agenda: The cooperative approach addresses the whole bug mitigation process.



Agenda: The cooperative approach addresses the whole bug mitigation process.



Agenda: The cooperative approach addresses the whole bug mitigation process.



This Talk: Highlights

Thresher: Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations

- ▶ Idea: Assume alarms false, prove them so automatically
- ▶ Filters out ~90% of false alarms to **expose true bugs**
- ▶ Going from ~450 hours of manual work to ~30 hours
- ▶ Application: Find memory leaks and **eliminate crashes in Android**

This Talk: Highlights

Thresher: Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations

- ▶ Idea: Assume alarms false, prove them so automatically
- ▶ Filters out ~90% of false alarms to **expose true bugs**
- ▶ Going from ~450 hours of manual work to ~30 hours
- ▶ Application: Find memory leaks and **eliminate crashes in Android**

Fissile Types: Checking Reflection with Almost Everywhere Invariants

Strengthen type checking with symbolic analysis

- ▶ Interactive checking speeds: making **IDE integration possible**
- ▶ Application: Prevent “MethodNotFound” errors in Objective-C (MacOS/iOS)

This Talk: Highlights

Thresher: Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations

- ▶ Idea: Assume alarms false, prove them so automatically
- ▶ Filters out ~90% of false alarms to **expose true bugs**
- ▶ Going from ~450 hours of manual work to ~30 hours
- ▶ Application: Find memory leaks and **eliminate crashes in Android**

Fissile Types: Checking Reflection with Almost Everywhere Invariants

Strengthen type checking with symbolic analysis

- ▶ Interactive checking speeds: making **IDE integration possible**
- ▶ Application: Prevent “MethodNotFound” errors in Objective-C (MacOS/iOS)

Thresher: Precise Refutations for Heap Reachability

What are heap reachability queries?

What are heap reachability queries?

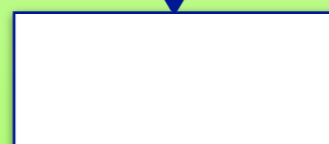
Can an object ever be reached from another object via pointer dereferences?

What are heap reachability queries?

Can an object ever be reached from another object via pointer dereferences?

Is there a program execution where at some time variable

x



of type T ?

Example

How is this useful? We identify memory leaks that cause your app to crash!



How is this useful? We identify memory leaks that cause your app to crash!



stackoverflow

Questions

Tags

Tour


Users

Android: Crash on rotation, horizontal to vertical

Crash is detected after rotating phone in Gmail Sync now view 

[phonegap](#) >

[important bug]cordova 1.9 crash on rotation android

5 posts by 2 authors  



stackoverflow

Questions

Tags

Tour

Users

App crashes when rotating Samsung phone



androidterm

Android Terminal Emulator

[Project Home](#)

[Downloads](#)

[Wiki](#)

Issues

[Source](#)

[New issue](#)

Search

Open issues



for

★ **Issue 20: Crashes when rotating phone horizontally**

1 person starred this issue and may be notified of changes.

How is this useful? We identify memory leaks that cause your app to crash!



stackoverflow

Questions

Tags

Tour

Users

Android: Crash on rotation, horizontal to

How can you have memory leaks with a garbage collected run-time?

phonegap >

AndroidTerm

Android Terminal Emulator

[Project Home](#)

[Downloads](#)

[Wiki](#)

[Issues](#)

[Source](#)

[New issue](#)

Search

Open issues



for

★ **Issue 20: Crashes when rotating phone horizontally**

1 person starred this issue and may be notified of changes.

Android memory leaks underly rotation-based crashes.

Activity objects
encapsulate the UI



Android memory leaks underly rotation-based crashes.

Activity objects
encapsulate the UI



of type Activity



Android memory leaks underly rotation-based crashes.

Activity objects
encapsulate the UI

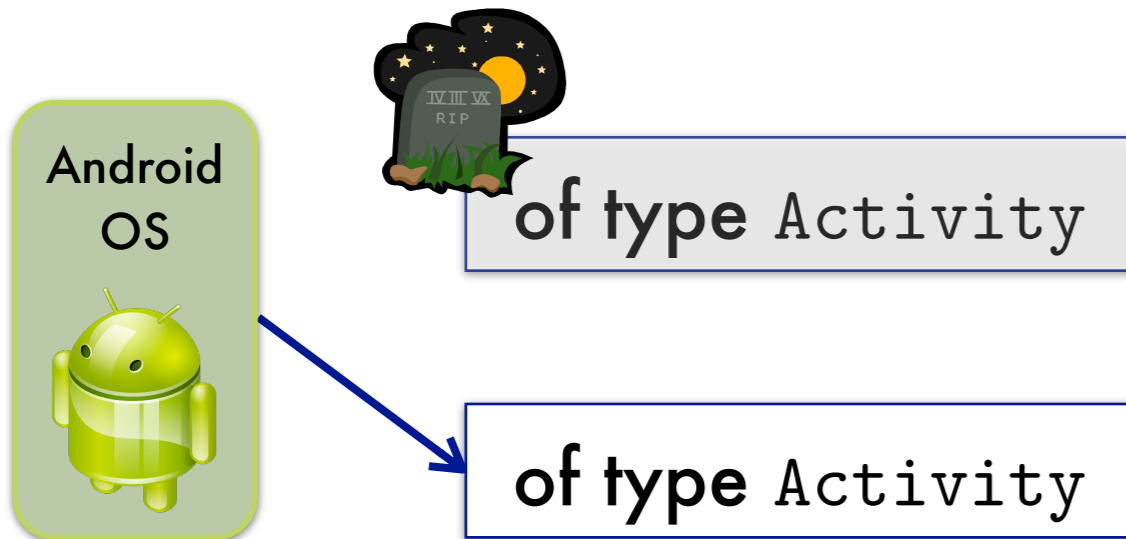


of type Activity

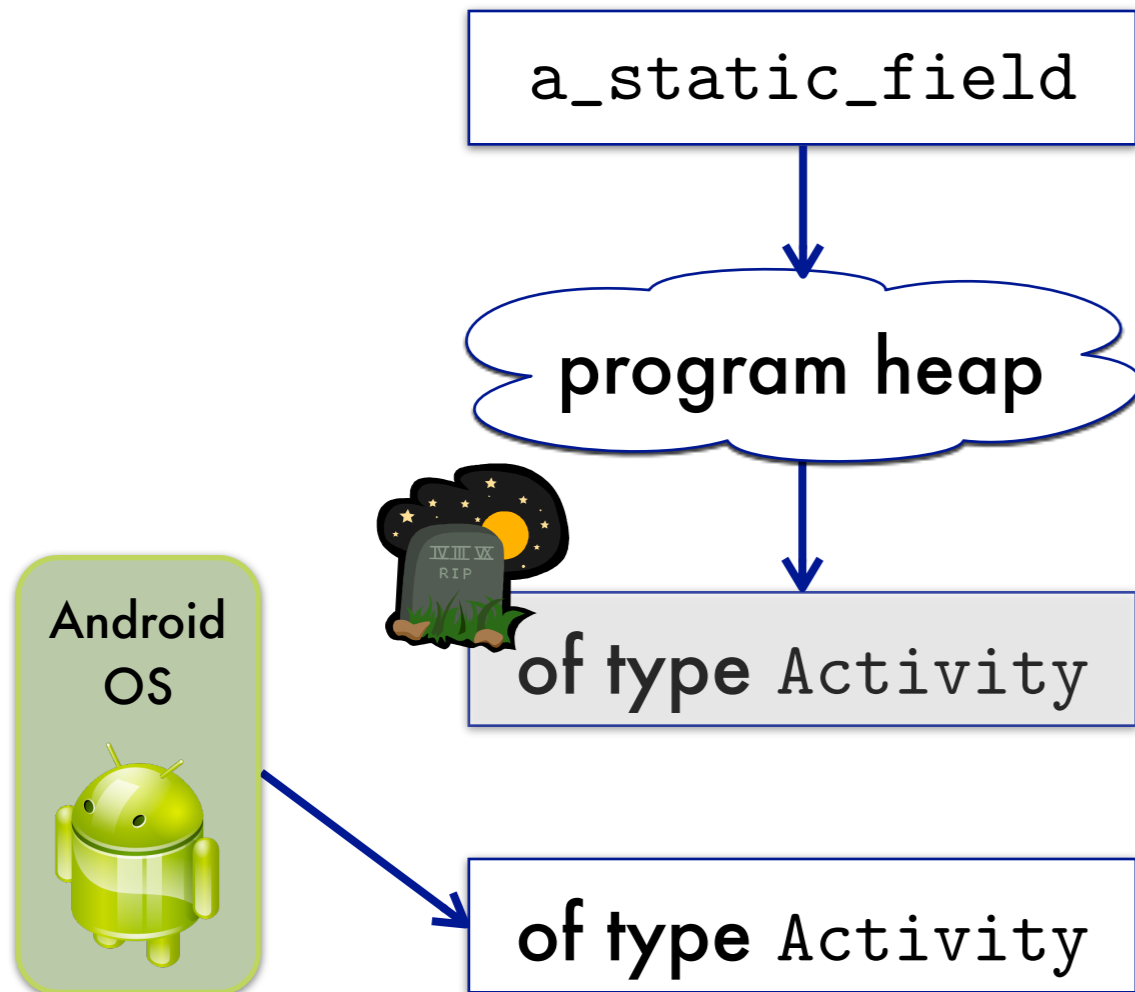


Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI



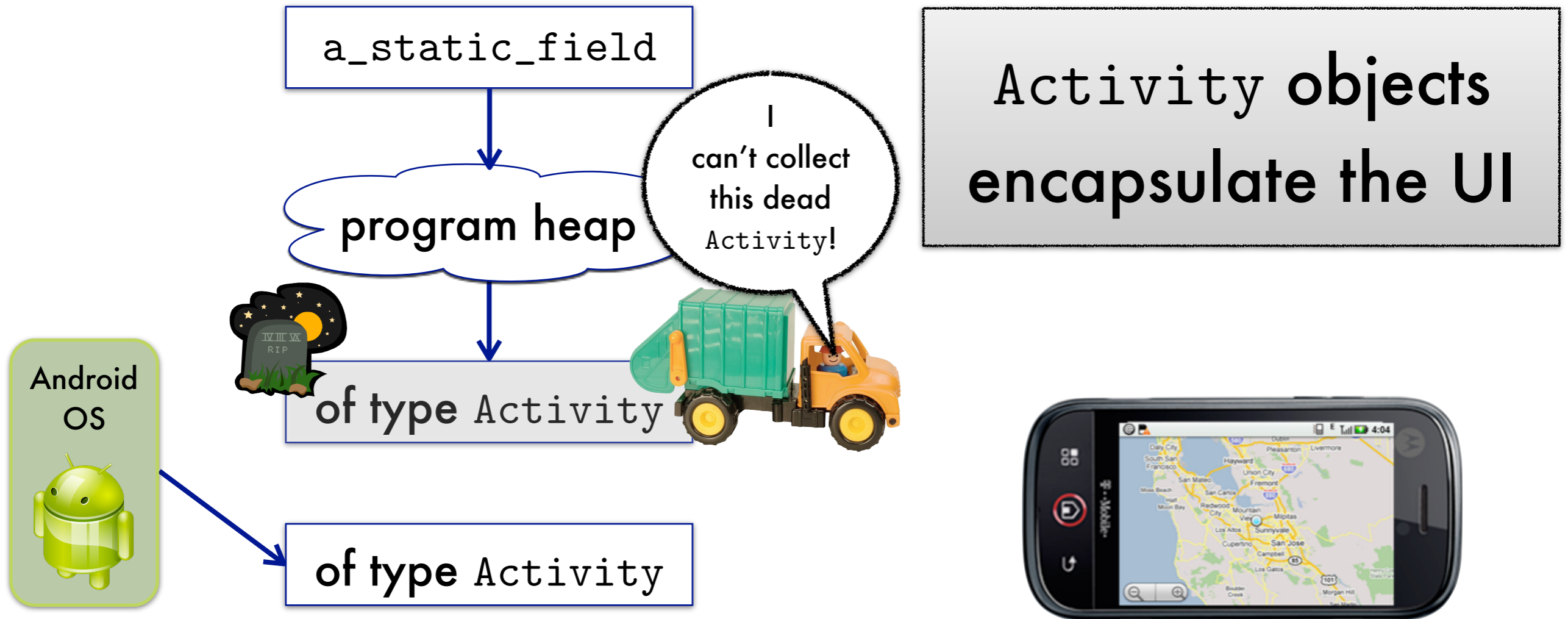
Android memory leaks underly rotation-based crashes.



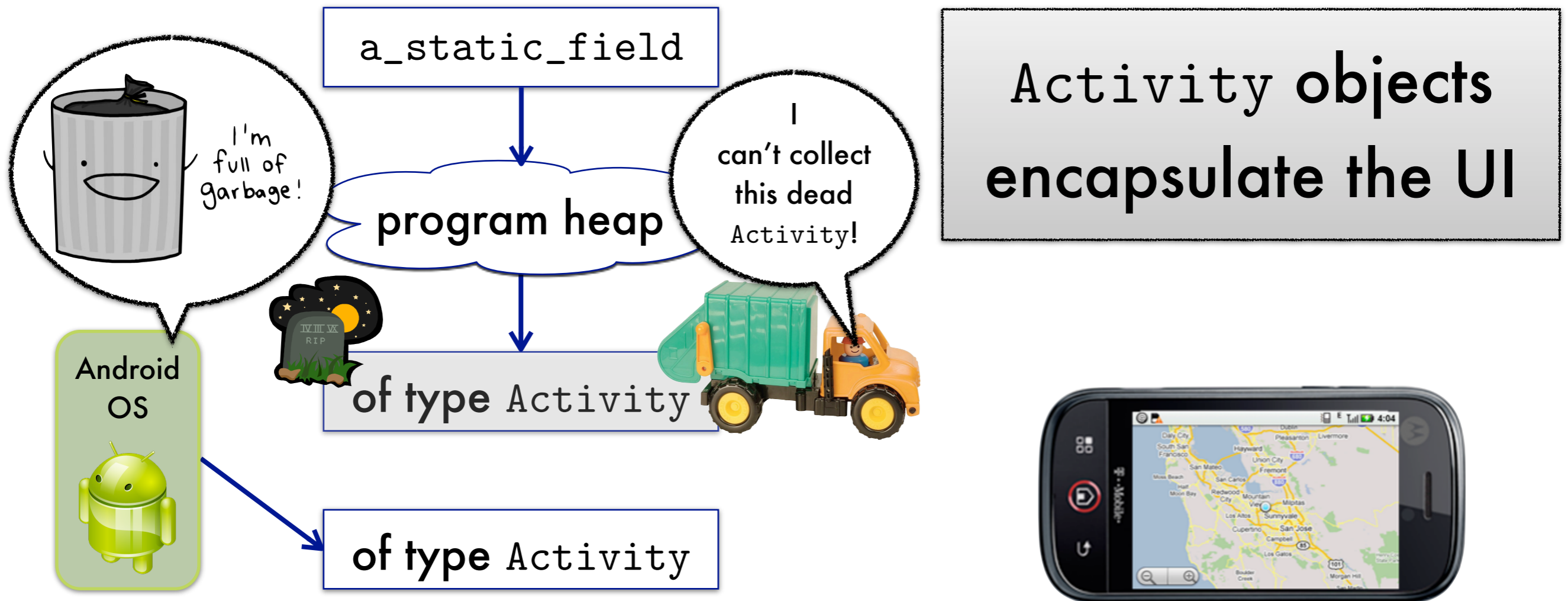
Activity objects encapsulate the UI



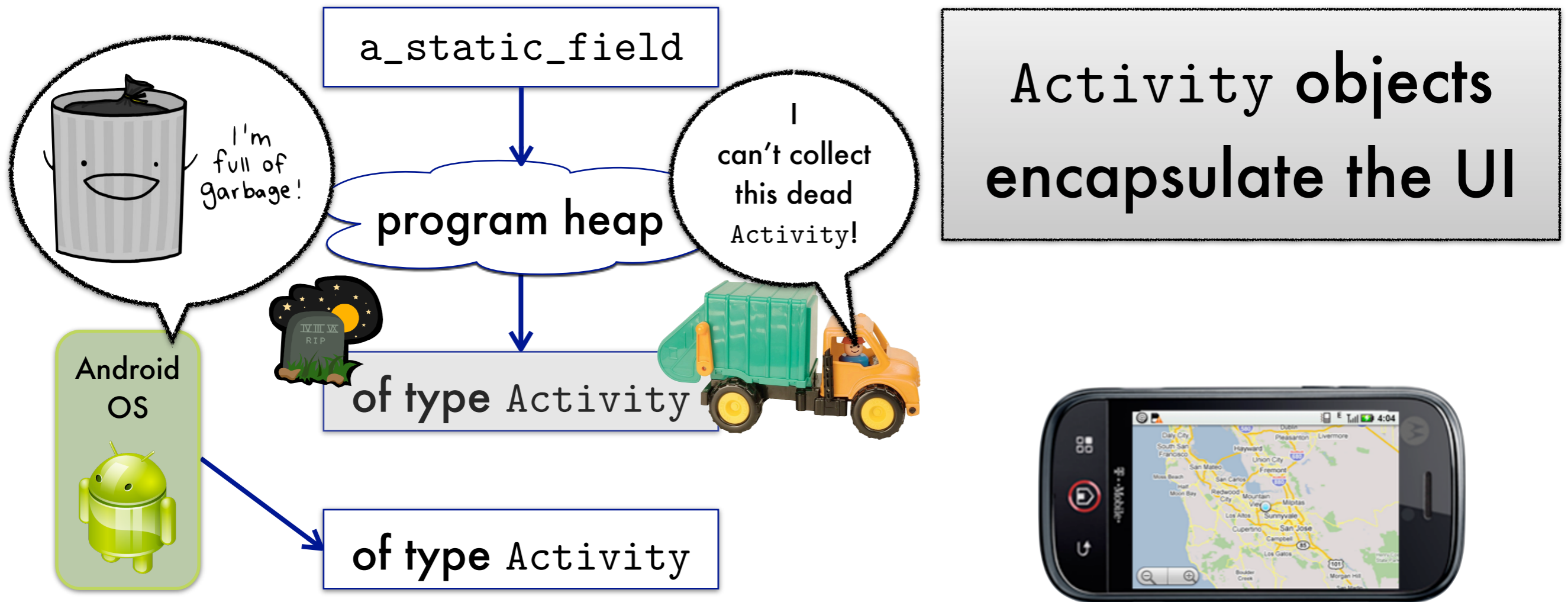
Android memory leaks underly rotation-based crashes.



Android memory leaks underly rotation-based crashes.

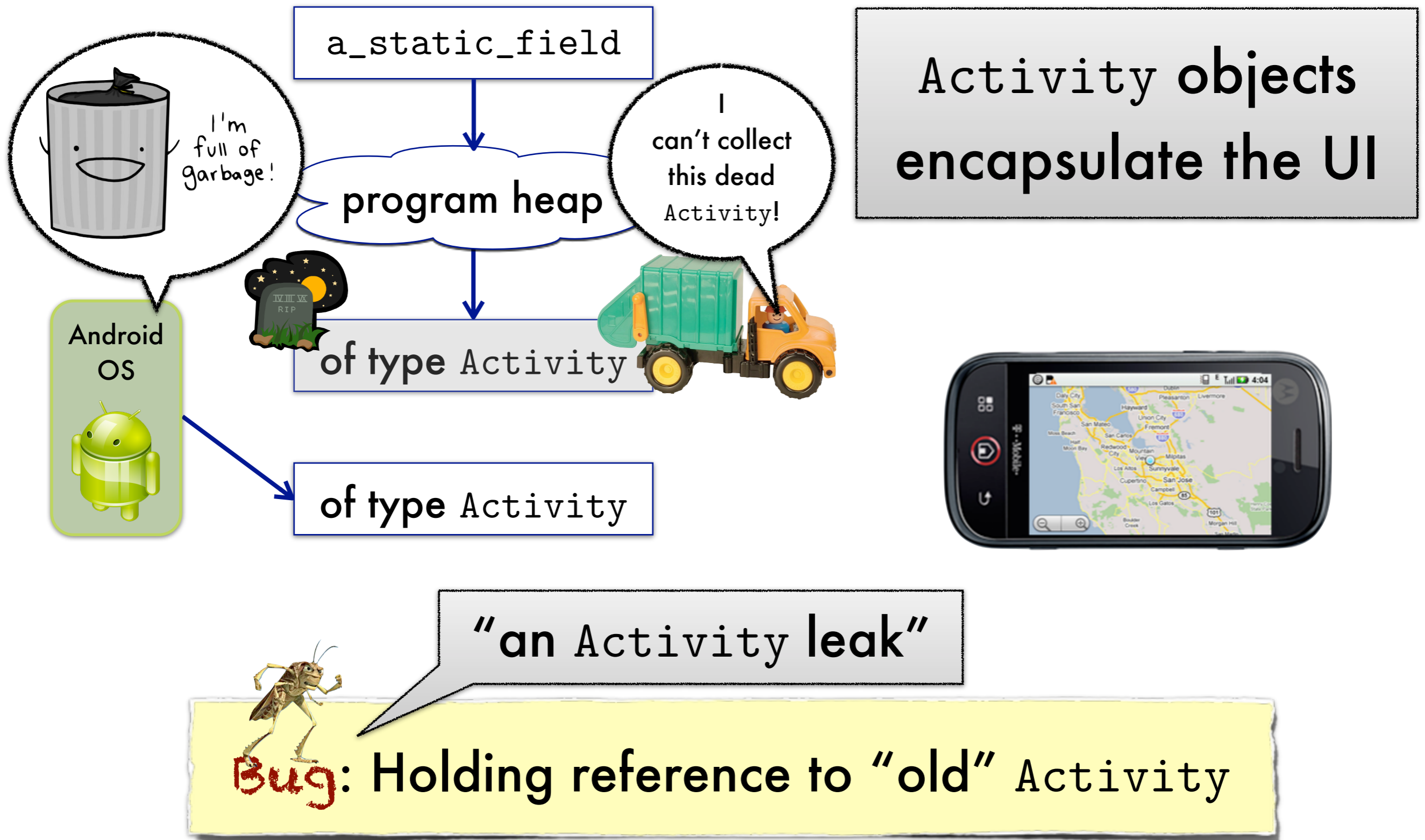


Android memory leaks underly rotation-based crashes.



Bug: Holding reference to "old" Activity

Android memory leaks underly rotation-based crashes.



The expert recommendation ...



The expert recommendation ...

A screenshot of a web browser displaying an article from the Android Developers Blog. The browser's address bar shows the URL: android-developers.blogspot.dk/2009/01/avoiding-memory-leaks.html. The page features a dark blue header with the Android Developers Blog logo and a grid of icons. Below the header, the article is dated 19 JANUARY 2009 and titled "Avoiding memory leaks". The text explains that Android applications are limited to 16 MB of heap memory and discusses the importance of avoiding memory leaks by properly managing Context objects. A code snippet is provided, showing an @Override method onCreate that initializes a TextView and sets its text to "Leaks are bad". The left sidebar includes a search bar, an archive list for years 2012 through 2009, and a monthly breakdown for 2009.

Questions containing 'andro' x Issues - android - Android - x Android Developers Blog: Av x Android Developers Blog: Me x

android-developers.blogspot.dk/2009/01/avoiding-memory-leaks.html

Android Developers Blog

Developers

19 JANUARY 2009

Avoiding memory leaks

Android applications are, at least on the T-Mobile G1, limited to 16 MB of heap. It's both a lot of memory for a phone and yet very little for what some developers want to achieve. Even if you do not plan on using all of this memory, you should use as little as possible to let other applications run without getting them killed. The more applications Android can keep in memory, the faster it will be for the user to switch between his apps. As part of my job, I ran into memory leaks issues in Android applications and they are most of the time due to the same mistake: keeping a long-lived reference to a [Context](#).

On Android, a [Context](#) is used for many operations but mostly to load and access resources. This is why all the widgets receive a [Context](#) parameter in their constructor. In a regular Android application, you usually have two kinds of [Context](#), [Activity](#) and [Application](#). It's usually the first one that the developer passes to classes and methods that need a [Context](#):

```
@Override
protected void onCreate(Bundle state) {
    super.onCreate(state);

    TextView label = new TextView(this);
    label.setText("Leaks are bad");
}
```

SEARCH

ARCHIVE

- ▶ 2012 (31)
- ▶ 2011 (68)
- ▶ 2010 (73)
- ▼ 2009 (63)
 - ▶ December (7)
 - ▶ November (5)
 - ▶ October (5)
 - ▶ September (8)
 - ▶ August (2)
 - ▶ July (1)

The expert recommendation ...



“Do not keep long-lived references to a context-activity”

A screenshot of a web browser displaying an article from the Android Developers Blog. The browser's address bar shows the URL: android-developers.blogspot.dk/2009/01/avoiding-memory-leaks.html. The page features a dark blue header with the Android Developers Blog logo and a grid of icons. Below the header, the article is dated 19 JANUARY 2009 and titled "Avoiding memory leaks". The text explains that Android applications are limited to 16 MB of heap memory and that memory leaks can occur due to long-lived references to Context. It also mentions that Context is used for many operations, such as loading and accessing resources. A code snippet is provided, showing an override of the onCreate method in an Android class, which creates a TextView and sets its text to "Leaks are bad".

Developers

19 JANUARY 2009

Avoiding memory leaks

Android applications are, at least on the T-Mobile G1, limited to 16 MB of heap. It's both a lot of memory for a phone and yet very little for what some developers want to achieve. Even if you do not plan on using all of this memory, you should use as little as possible to let other applications run without getting them killed. The more applications Android can keep in memory, the faster it will be for the user to switch between his apps. As part of my job, I ran into memory leaks issues in Android applications and they are most of the time due to the same mistake: keeping a long-lived reference to a [Context](#).

On Android, a [Context](#) is used for many operations but mostly to load and access resources. This is why all the widgets receive a [Context](#) parameter in their constructor. In a regular Android application, you usually have two kinds of [Context](#), [Activity](#) and [Application](#). It's usually the first one that the developer passes to classes and methods that need a [Context](#):

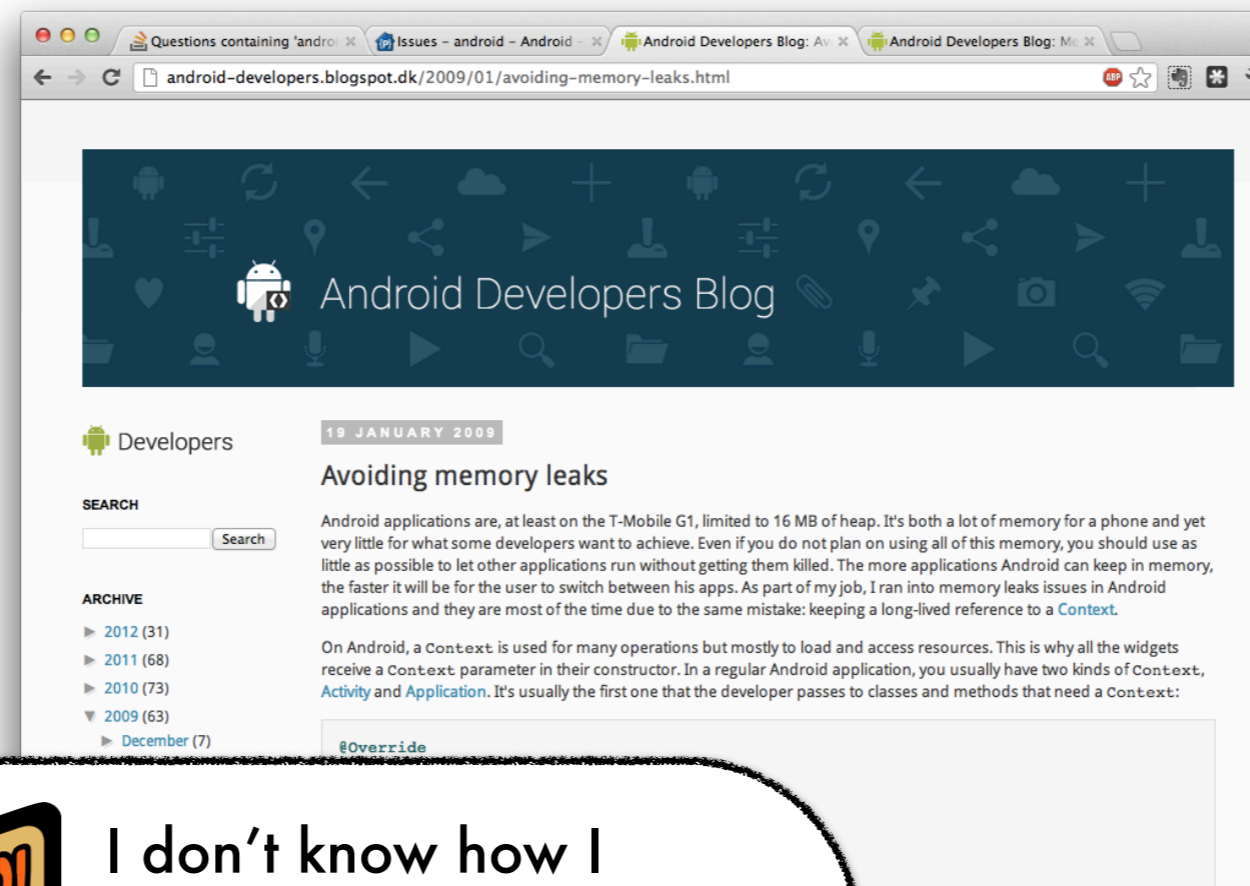
```
@Override
protected void onCreate(Bundle state) {
    super.onCreate(state);

    TextView label = new TextView(this);
    label.setText("Leaks are bad");
```

The expert recommendation ...



“Do not keep long-lived references to a context-activity”



I don't know how I created a long-lived reference to an Activity!

The expert recommendation ...



“Do not keep long-lived references to a context-activity”



I don't know how I created a long-lived reference to an Activity!

Often: A misunderstanding of a library causes the **library** to keep the Activity reference.

The state of practice in debugging Activity leaks ...



The screenshot shows a web browser window displaying a blog post. The browser's address bar shows the URL: `android-developers.blogspot.dk/2011/03/memory-analysis-for-android.html`. The page header features the Android Developers Blog logo and a navigation bar with various icons. The main content area is titled "Memory Analysis for Android Applications" and is dated "24 MARCH 2011". The author is identified as Patrick Dubroy. The article text discusses memory management in the Dalvik runtime and mentions tools like the Allocation Tracker and heap dumps. A small portrait of the author is visible on the right side of the article.

Developers

SEARCH

ARCHIVE

- ▶ 2012 (31)
- ▼ 2011 (68)
 - ▶ December (7)
 - ▶ November (7)
 - ▶ October (5)
 - ▶ September (5)
 - ▶ August (3)
 - ▶ July (7)
 - ▶ June (3)
 - ▶ May (5)

24 MARCH 2011

Memory Analysis for Android Applications

[This post is by Patrick Dubroy, an Android engineer who writes about programming, usability, and interaction on his [personal blog](#). — Tim Bray]

The Dalvik runtime may be garbage-collected, but that doesn't mean you can ignore memory management. You should be especially mindful of memory usage on mobile devices, where memory is more constrained. In this article, we're going to take a look at some of the memory profiling tools in the Android SDK that can help you trim your application's memory usage.

Some memory usage problems are obvious. For example, if your app leaks memory every time the user touches the screen, it will probably trigger an `OutOfMemoryError` eventually and crash your app. Other problems are more subtle, and may just degrade the performance of both your app (as garbage collections are more frequent and take longer) and the entire system.

Tools of the trade

The Android SDK provides two main ways of profiling the memory usage of an app: the *Allocation Tracker* tab in DDMS, and heap dumps. The Allocation Tracker is useful when you want to get a sense of what kinds of allocation are happening over a

The state of practice in debugging Activity leaks ...



1. Run the app

The state of practice in debugging Activity leaks ...



24 MARCH 2011

Memory Analysis for Android Applications

[This post is by Patrick Dubroy, an Android engineer who writes about programming, usability, and interaction on his *personal blog*. — Tim Bray]

The Dalvik runtime may be garbage-collected, but that doesn't mean you can ignore memory management. You should be especially mindful of memory usage on mobile devices, where memory is more constrained. In this article, we're going to take a look at some of the memory profiling tools in the Android SDK that can help you trim your application's memory usage.

Some memory usage problems are obvious. For example, if your app leaks memory every time the user touches the screen, it will probably trigger an `OutOfMemoryError` eventually and crash your app. Other problems are more subtle, and may just degrade the performance of both your app (as garbage collections are more frequent and take longer) and the entire system.

Tools of the trade

The Android SDK provides two main ways of profiling the memory usage of an app: the *Allocation Tracker* tab in DDMS, and heap dumps. The Allocation Tracker is useful when you want to get a sense of what kinds of allocation are happening over a

Dalvik Debug Monitor

Info Threads VM Heap Allocation Tracker Sysinfo Emulator Control Event Log

Heap updates will happen after every GC for this client

ID	Heap Size	Allocated	Free	% Used	# Objects	Cause GC
1	8.570 MB	8.452 MB	121.320 KB	98.62%	59,281	

Display: Stats

Type	Count	Total Size	Smallest	Largest	Median	Average
free	1,772	107.312 KB	16 B	48.297 KB	24 B	62 B
data object	40,528	1.229 MB	16 B	1.047 KB	32 B	31 B
class object	2,187	637.234 KB	168 B	34.125 KB	168 B	298 B
1-byte array (byte[], boolean[])	2,247	5.654 MB	24 B	1.500 MB	48 B	2.576 KB
2-byte array (short[], char[])	10,373	677.352 KB	24 B	28.023 KB	48 B	66 B
4-byte array (object[], int[], float[])	3,663	276.812 KB	24 B	16.023 KB	40 B	77 B
8-byte array (long[], double[])	283	14.875 KB	24 B	4.000 KB	32 B	53 B
non-Java object	92	14.219 KB	16 B	8.023 KB	32 B	158 B

1. Run the app
2. Watch the heap usage

The state of practice in debugging Activity leaks ...



Suppose we're lucky and find a possible culprit. Now what?

- ▶ Where in the **code** is this object allocated?
- ▶ What about the object that references it?
- ▶ Where is the reference created?
- ▶ Is this reference needed?
- ▶ For what periods?

3. Dump the heap. Dig around and **hope** to find the culprit

Dalvik Debug Monitor

Heap updates will happen after every GC for this client

ID	Heap Size	Allocated	Free	% Used	# Objects	Cause GC
1	8.570 MB	8.452 MB	121,320 KB	98.62%	59,281	

Display: Stats

Type	Count	Total Size	Smallest	Largest	Median	Average
free	1,772	107.312 KB	16 B	48.297 KB	24 B	62 B
data object	40,528	1.229 MB	16 B	1.047 KB	32 B	31 B
class object	2,187	637.234 KB	168 B	34.125 KB	168 B	298 B
1-byte array (byte[], boolean[])	2,247	5.654 MB	24 B	1.500 MB	48 B	2.576 KB
2-byte array (short[], char[])	10,373	677.352 KB	24 B	28.023 KB	48 B	66 B
4-byte array (object[], int[], float[])	3,663	276.812 KB	24 B	16.023 KB	40 B	77 B
8-byte array (long[], double[])	283	14.875 KB	24 B	4.000 KB	32 B	53 B
non-Java object	92	14.219 KB	16 B	8.023 KB	32 B	158 B

Eclipse Memory Analyzer

leak-converted.hprof

list_objects (selection of 'byte[]' -inbound)

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
byte[8] @ 0x429b69c8 HPDS...	24	24
byte[2797568] @ 0x426fe780 '2' & !'2\$ (3%&##+ (.S...%...\$...#...+...7&2>*0	2,797,584	2,797,584
mBuffer android.graphics.Bitmap @ 0x40a50fa8	40	2,797,640
value java.util.HashMap\$HashMapEntry @ 0x40a4ceb8	24	5,595,472
[13] java.util.HashMap\$HashMapEntry[16] @ 0x40805440	80	32,802,960
table java.util.HashMap @ 0x40801a98	48	32,803,008
sBitmapCache class com.example.android.hcgallery.ContentFragment (8	32,803,056
<class> com.example.android.hcgallery.ContentFragment @ 0x4080	128	384
value java.util.HashMap\$HashMapEntry @ 0x408009c0	24	152
Σ Total: 2 entries		
byte[2797568] @ 0x42453768 %..jS .+& .61+.HA.;F79.92.4-'.C.8.MEB.@.8.'...-?;<	2,797,584	2,797,584
byte[2797568] @ 0x421a8750z.FRF.Pi.OXU.NWT.ZUY.ZUY.yvo.....lt\,u.e.z.\,syU.	2,797,584	2,797,584
byte[2797568] @ 0x41efd120 njg.pli.kgd.b^[\,da\,olg.tql.qni.roh.urk.wtm.spl.lib.he^k	2,797,584	2,797,584
byte[3252224] @ 0x41be3108#.....	3,252,240	3,252,240
byte[2797568] @ 0x419380f0#.....	2,797,584	2,797,584
byte[2797568] @ 0x4168d0d8 d.B.d.B.d.@.d.@.f.7.g.@.h.B.h.B.i.B.h.B.i.C.g.C.f.B.f.C	2,797,584	2,797,584
byte[2797568] @ 0x413e20c0 cR>.eT@.eVA.dU@.aR=. ' Q<.' Q>.@.bS@.bS@.bS@.e'	2,797,584	2,797,584
byte[2797568] @ 0x411370a8#.....	2,797,584	2,797,584
byte[2797568] @ 0x40e8c090#.....	2,797,584	2,797,584
byte[1572864] @ 0x40d0c078#.....	1,572,880	1,572,880
byte[2797568] @ 0x40a61060#.....	2,797,584	2,797,584
byte[62100] @ 0x40a51db8#.....	62,112	62,112
byte[24] @ 0x40a4cd11#.....	40	40
byte[4096] @ 0x40a4aa50#.....	4,112	4,112
byte[24] @ 0x40a4a7a1#.....	40	40
byte[4096] @ 0x40a48148#.....	4,112	4,112
byte[24] @ 0x40a464f1#.....	40	40
byte[84] @ 0x40a40560#.....	96	96
byte[768] @ 0x40a40200#.....	784	784
byte[1572864] @ 0x408beab8 2@3.2@3.4@4.SA5.6A3.471.3>0.6A1.8C3.8E4.8E'	1,572,880	1,572,880
byte[84] @ 0x408ba188#.....	96	96
byte[960] @ 0x408b9d68#.....	976	976
byte[84] @ 0x408b9a48#.....	96	96
byte[960] @ 0x408b9628#.....	976	976
byte[56] @ 0x408b9318#.....	72	72
byte[192] @ 0x408b91f8#.....	208	208

36M of 81M

The state of practice in debugging Activity leaks ...



Suppose we're lucky and find a possible culprit. Now what?

- ▶ Where in the **code** is this object allocated?
- ▶ What about the object that references it?
- ▶ Where is the reference created?
- ▶ Is this reference needed?
- ▶ For what periods?

Dalvik Debug Monitor

Info Threads VM Heap Allocation Tracker Sysinfo Emulator Control Event Log

Heap updates will happen after every GC for this client

ID	Heap Size	Allocated	Free	% Used	# Objects	Cause GC
1	8.570 MB	8.452 MB	121.320 KB	98.62%	59,281	

Display: Stats

Type	Count	Total Size	Smallest	Largest	Median	Average
free	1,772	107.312 KB	16 B	48.297 KB	24 B	62 B
data object	40,528	1.229 MB	16 B	1.047 KB	32 B	31 B
class object	2,187	637.234 KB	168 B	34.125 KB	168 B	298 B
1-byte array (byte[], boolean[])	2,247	5.654 MB	24 B	1.500 MB	48 B	2.576 KB
2-byte array (short[], char[])	10,373	677.352 KB	24 B	28.023 KB	48 B	66 B
4-byte array (object[], int[], float[])	3,663	276.812 KB	24 B	16.023 KB	40 B	77 B
8-byte array (long[], double[])	283	14.875 KB	24 B	4.000 KB	32 B	53 B
non-Java object	92	14.219 KB	16 B	8.023 KB	32 B	158 B

Eclipse Memory Analyzer

leak-converted.hprof

Inspector

list_objects [selection of 'byte[]' -inbound]

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
byte[8] @ 0x429b69c8 HPDS...	24	24
byte[2797568] @ 0x426fe780 '2''2\$(3%&##+ (.!\$...%...\$...#...+...7&2>*0	2,797,584	2,797,584
mBuffer android.graphics.Bitmap @ 0x40a50fa8	40	2,797,640
value java.util.HashMap\$HashMapEntry @ 0x40a4ceb8	24	5,595,472
[13] java.util.HashMap\$HashMapEntry[16] @ 0x40805440	80	32,802,960
table java.util.HashMap @ 0x40801a98	48	32,803,008
sBitmapCache class com.example.android.hcgallery.ContentFragment (8	32,803,056
<class> com.example.android.hcgallery.ContentFragment @ 0x4080...	128	384
value java.util.HashMap\$HashMapEntry @ 0x408009c0	24	152
Σ Total: 2 entries		
byte[2797568] @ 0x42453768 %..j5 .+& .61+.HA.;F79.92.4-.C.8.MEB.@.8.'...-)?;<	2,797,584	2,797,584
byte[2797568] @ 0x421a8750z.FRF.P!P.OXU.NWT.ZUY.ZUY.yvo.....lt .u.e.z .syU.	2,797,584	2,797,584
byte[2797568] @ 0x41efd120 njg.pli.kgd.b^[da .olg.tql.qni.roh.urk.wtm.spi.lib.he^k	2,797,584	2,797,584
byte[3252224] @ 0x41be3108	3,252,240	3,252,240
byte[2797568] @ 0x419380f0	2,797,584	2,797,584
byte[2797568] @ 0x4168d0d8 d.B.d.B.d.@.d.@.f.7.g.@.h.B.h.B.i.B.h.B.i.C.g.C.f.B.f.C	2,797,584	2,797,584
byte[2797568] @ 0x413e20c0 <r>.eT@.eVA.dU@.aR=. 'Q<.'Q>.bs@.bs@.bs@.e'	2,797,584	2,797,584
byte[2797568] @ 0x411370a8	2,797,584	2,797,584

The culprit

3. Dump the heap. Dig

“One of the most dreaded bugs in Android is a memory leak. They are nasty because one piece of code causes an issue and in some other piece of code, your application crashes.” – <http://therockncoder.blogspot.com/2012/09/fixing-android-memory-leak.html>



**Program
Analysis**

Answering “Is there an Activity leak?” with program analysis ...

Can an object ever be reached from another object via pointer dereferences?

Is there a program execution where at some time

`a_static_field`



`of type Activity` ?

Example

Answering “Is there an Activity leak?” with program analysis ...

Can an object ever be reached from another object via pointer dereferences?

Is there a program execution where at some time

`a_static_field`



`of type Activity` ?

Example

Can be answered with a points-to analysis

Answering “Is there an Activity leak?” with program analysis ...

Can an object ever be reached from another object via pointer dereferences?

Is there a program execution where at some time

a_static_field



of type Activity ?

Example

Can be answered with a points-to analysis

with **approximation**

Hidden Truth

Answering “Is there an Activity leak?” with program analysis ...

Can an object ever be reached from another object via pointer dereferences?

Is there a program execution where at some time

a_static_field



of type Activity ?

Example

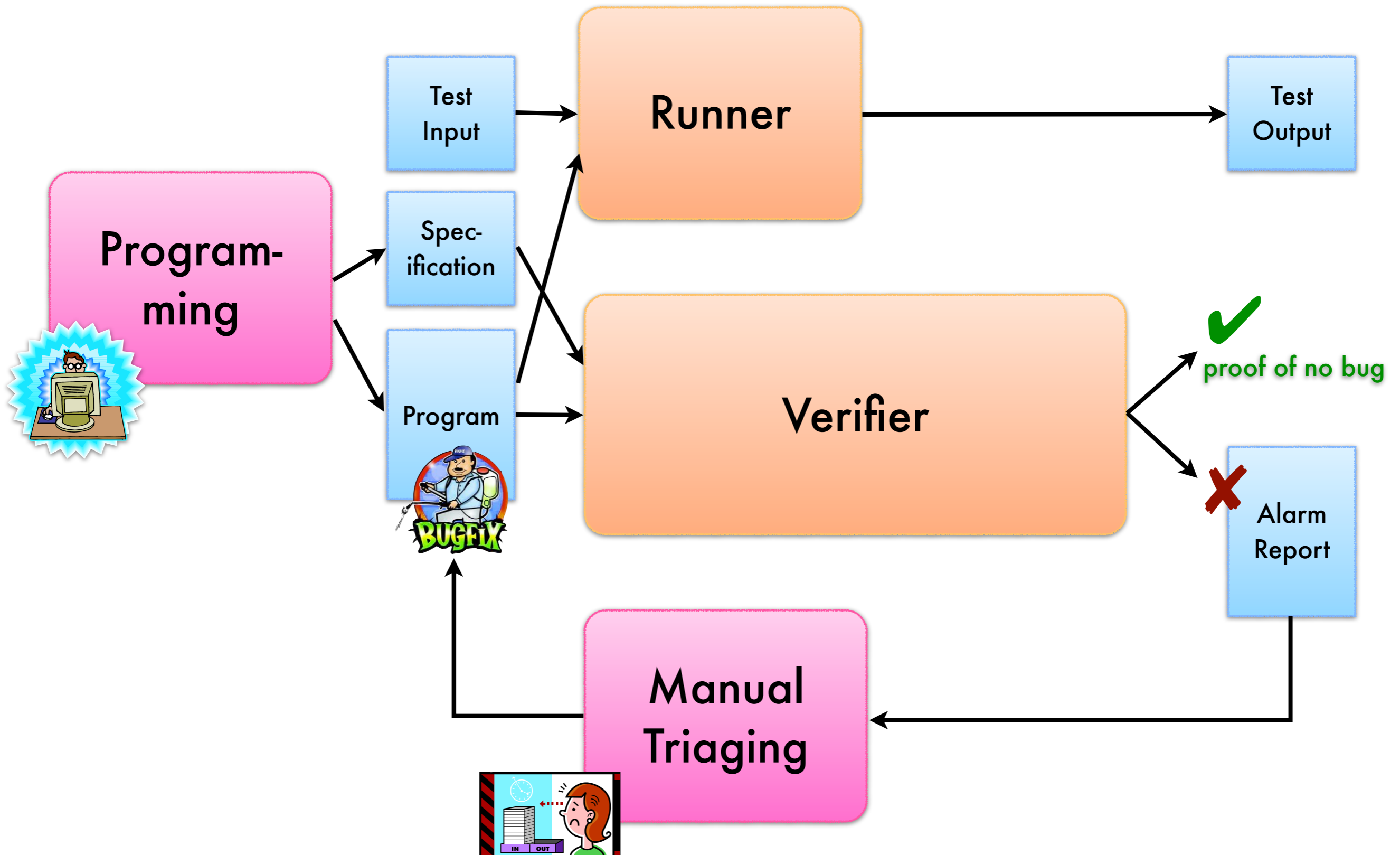
Can be answered with a points-to analysis

with **approximation**

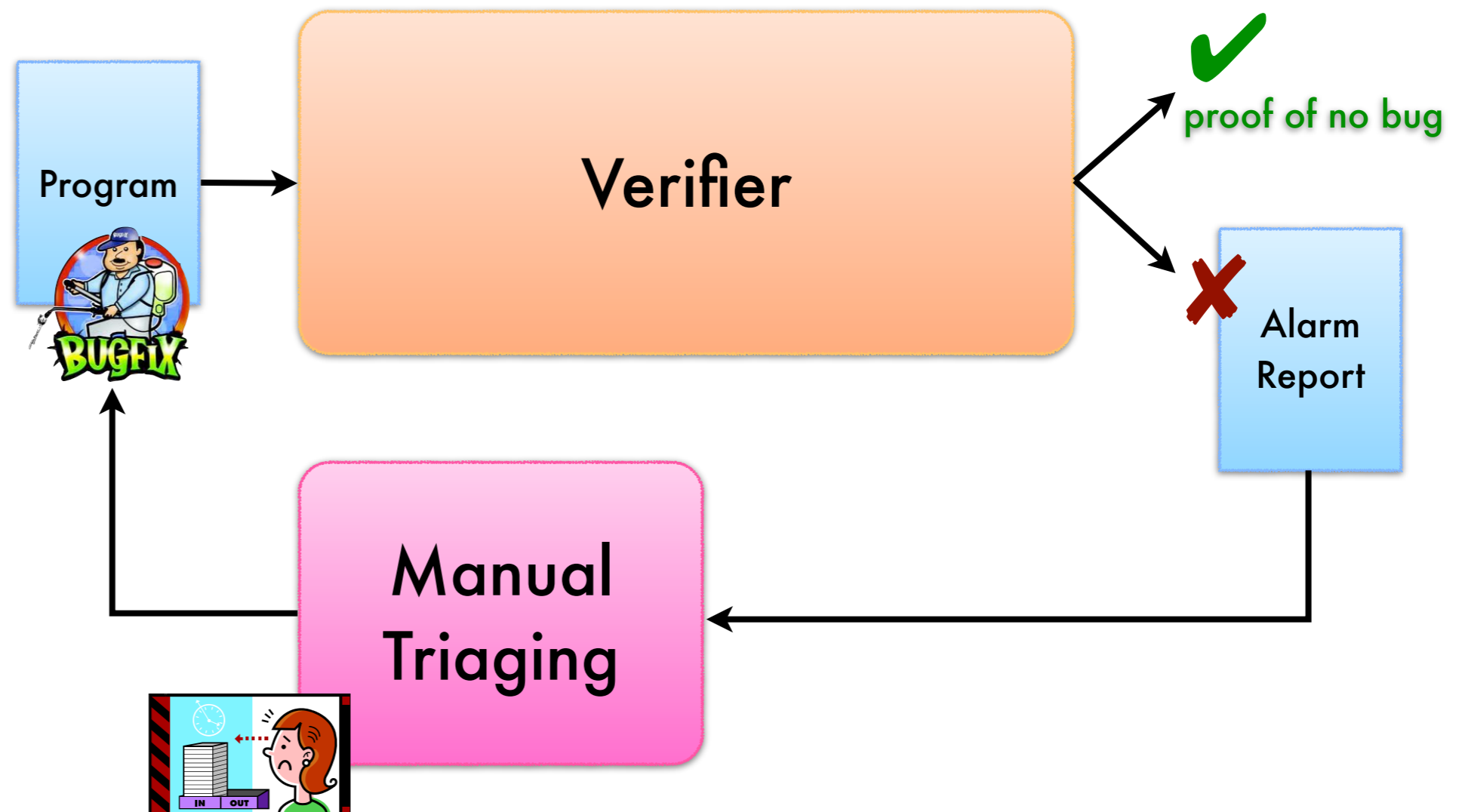
Hidden Truth

Some pointer relations **may** be false

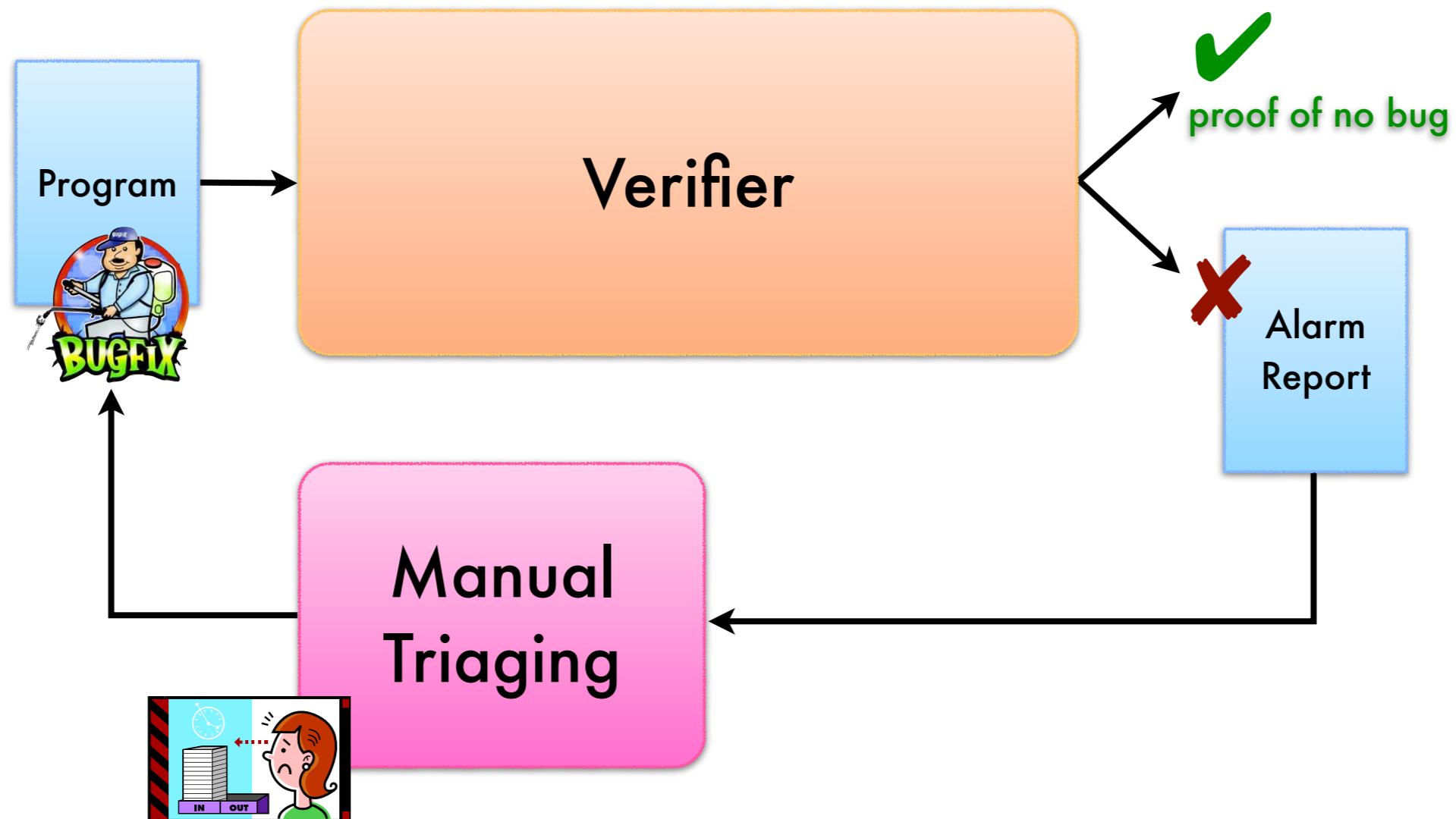
But with the cooperative approach ...



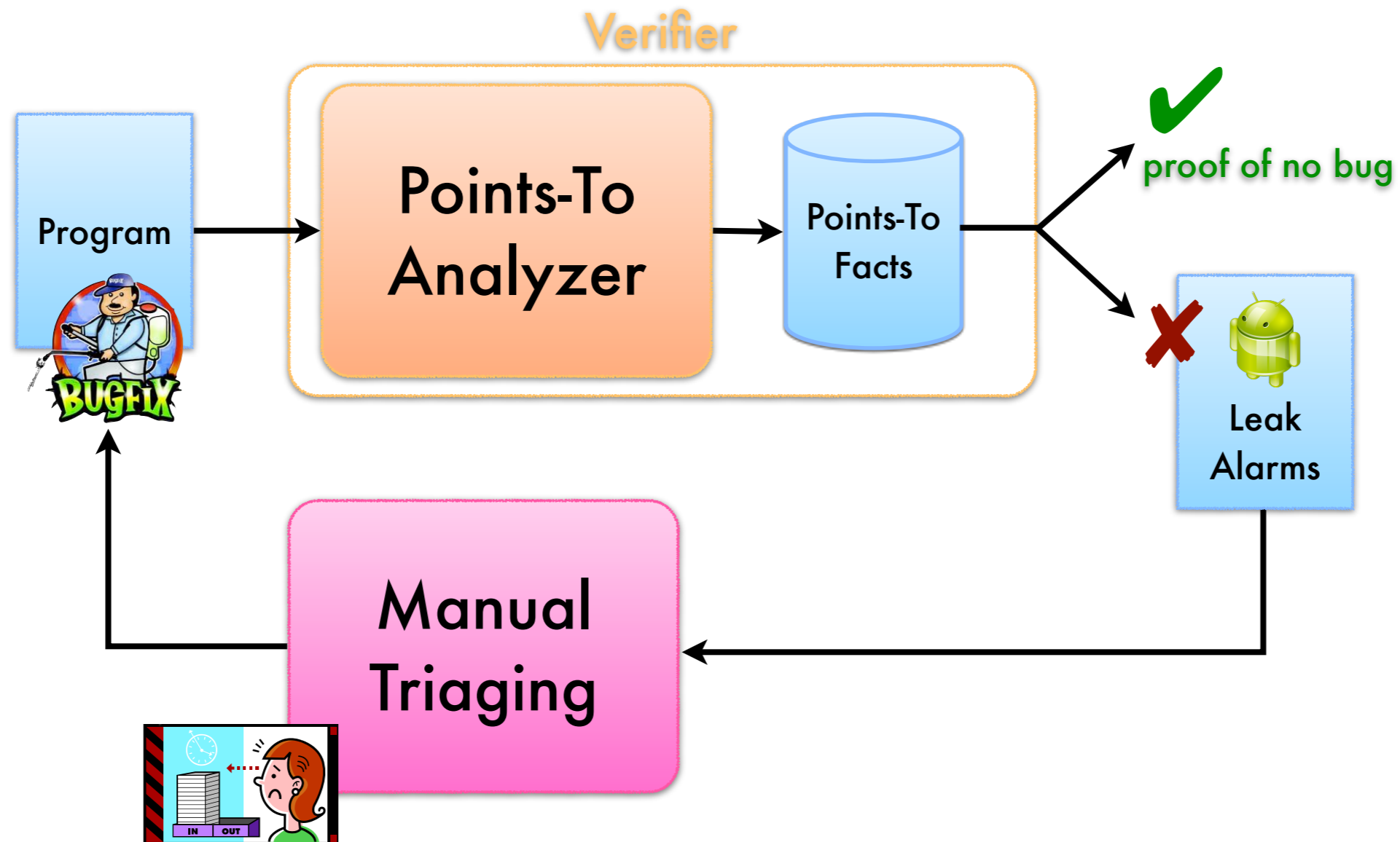
But with the cooperative approach ...



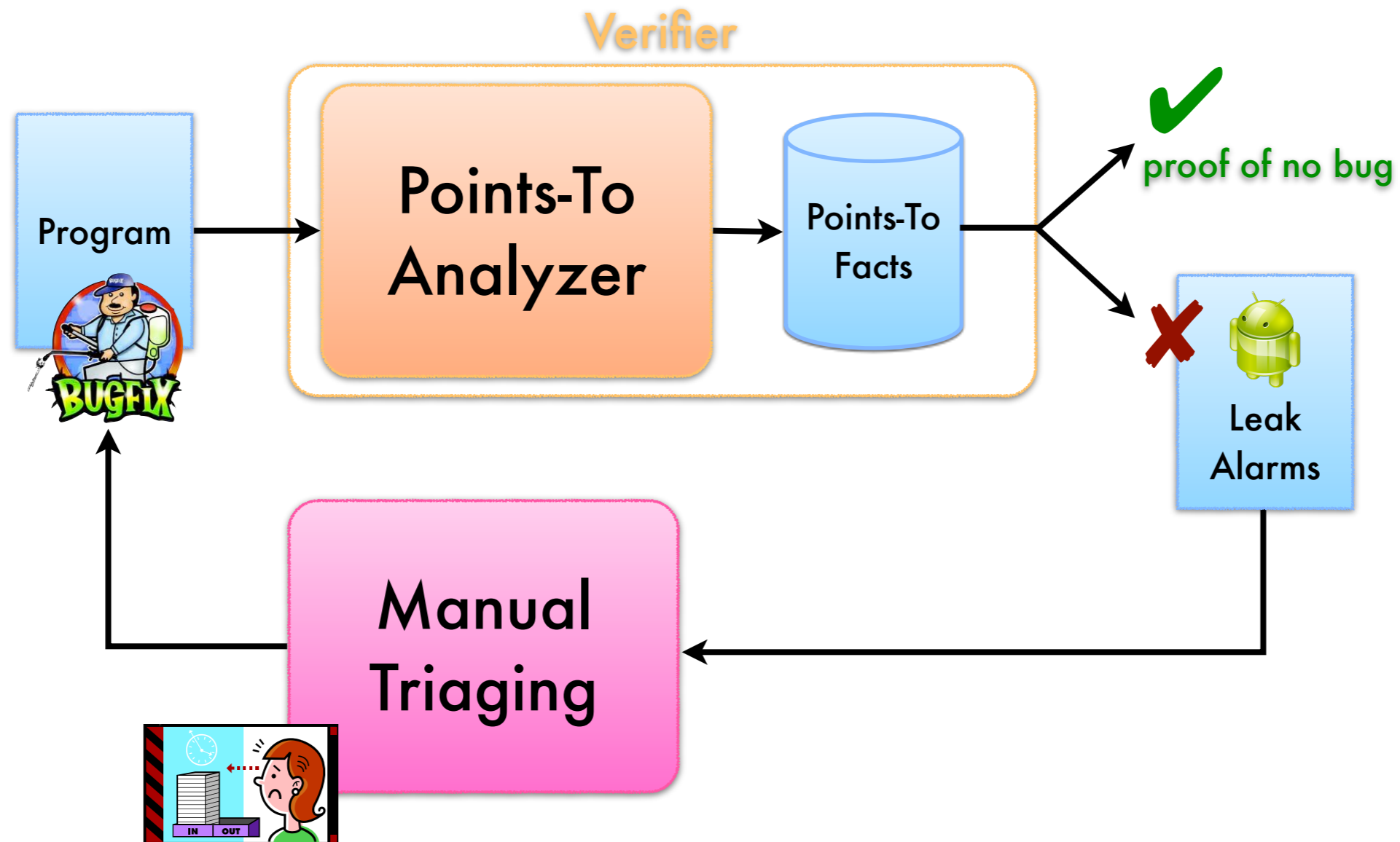
Thresher addresses alarm triage in a particularly challenging domain.



Thresher addresses alarm triage in a particularly challenging domain.

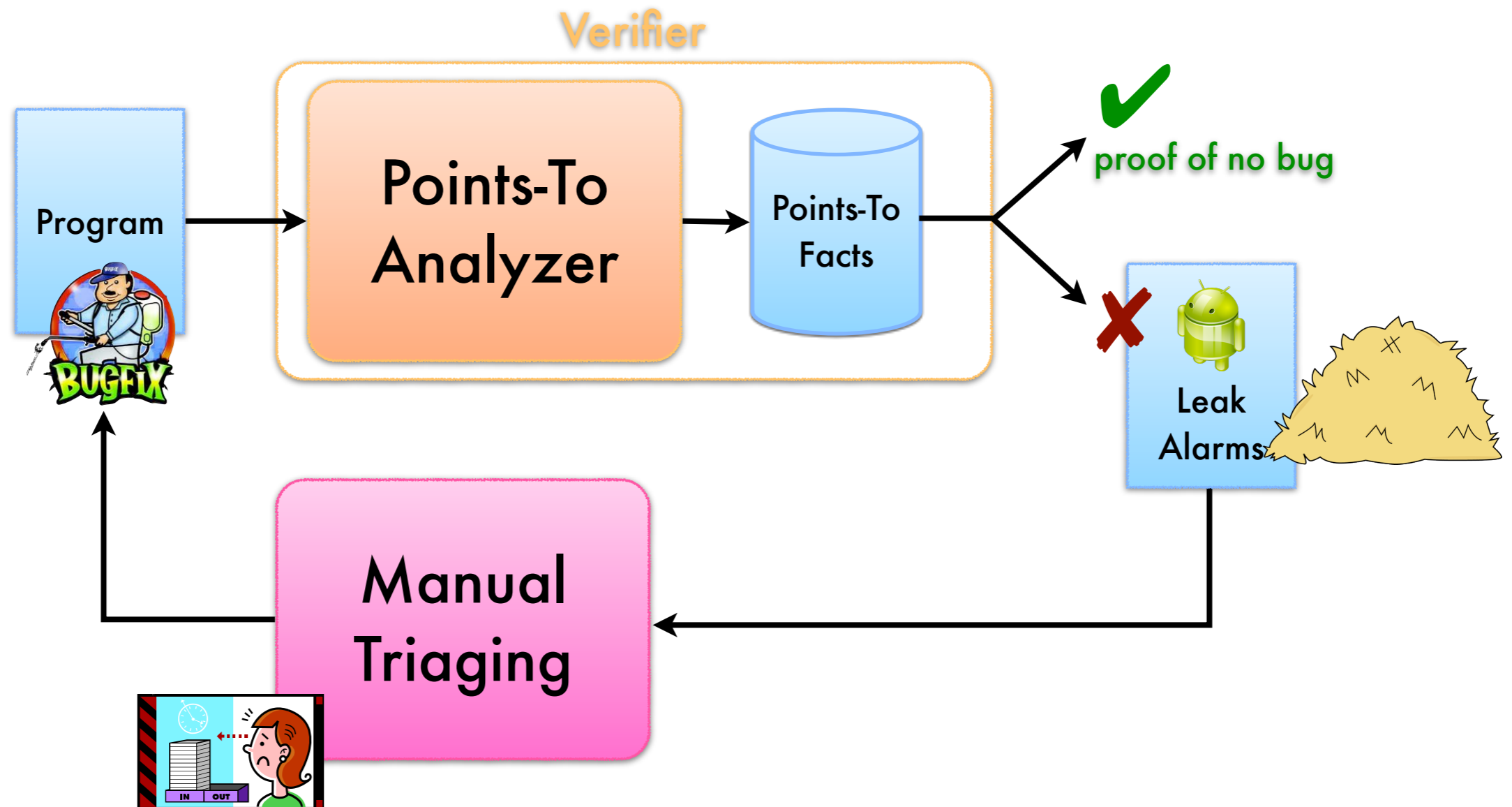


Thresher addresses alarm triage in a particularly challenging domain.



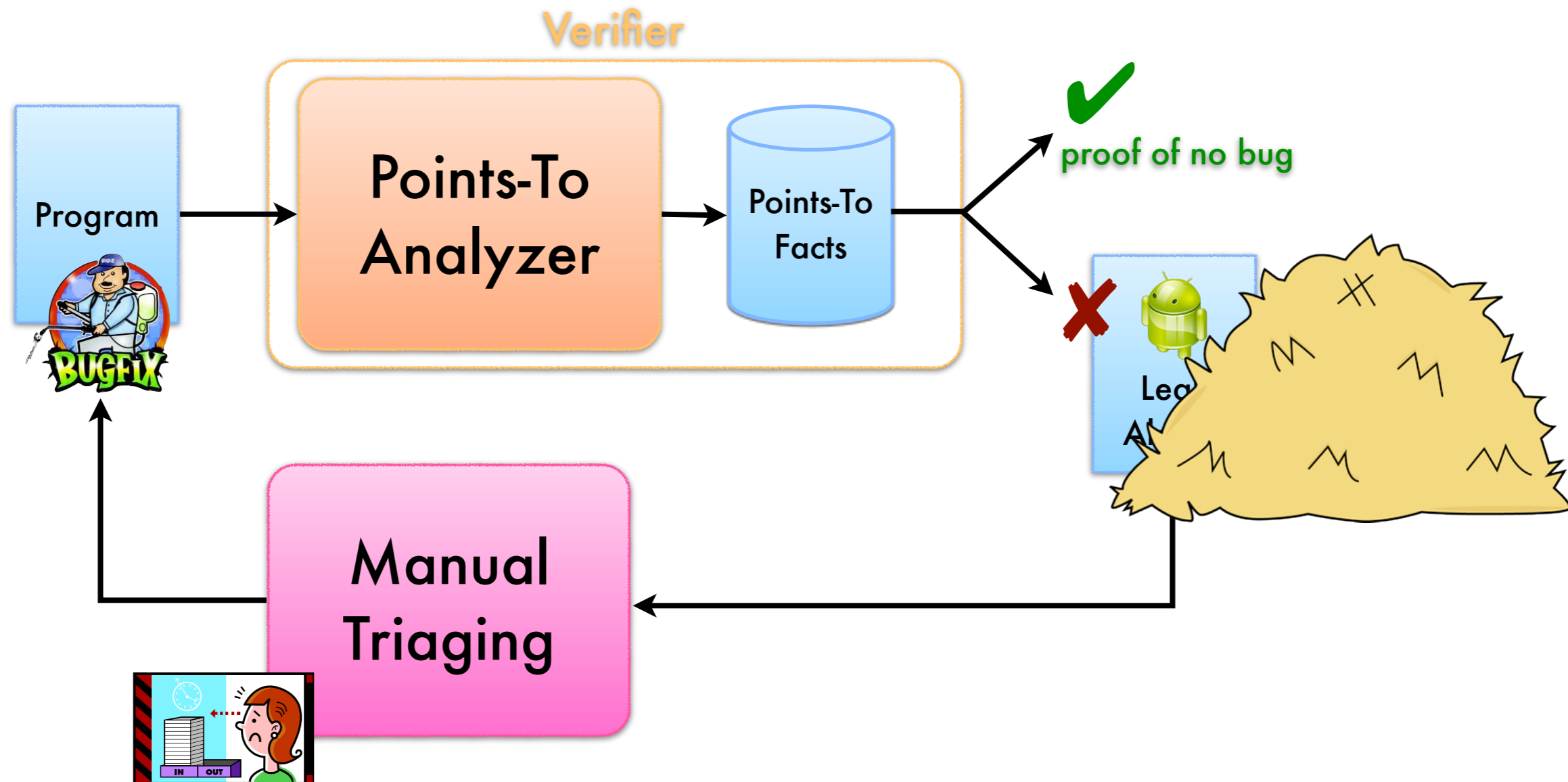
Known: Precise points-to analysis challenging

Thresher addresses alarm triage in a particularly challenging domain.



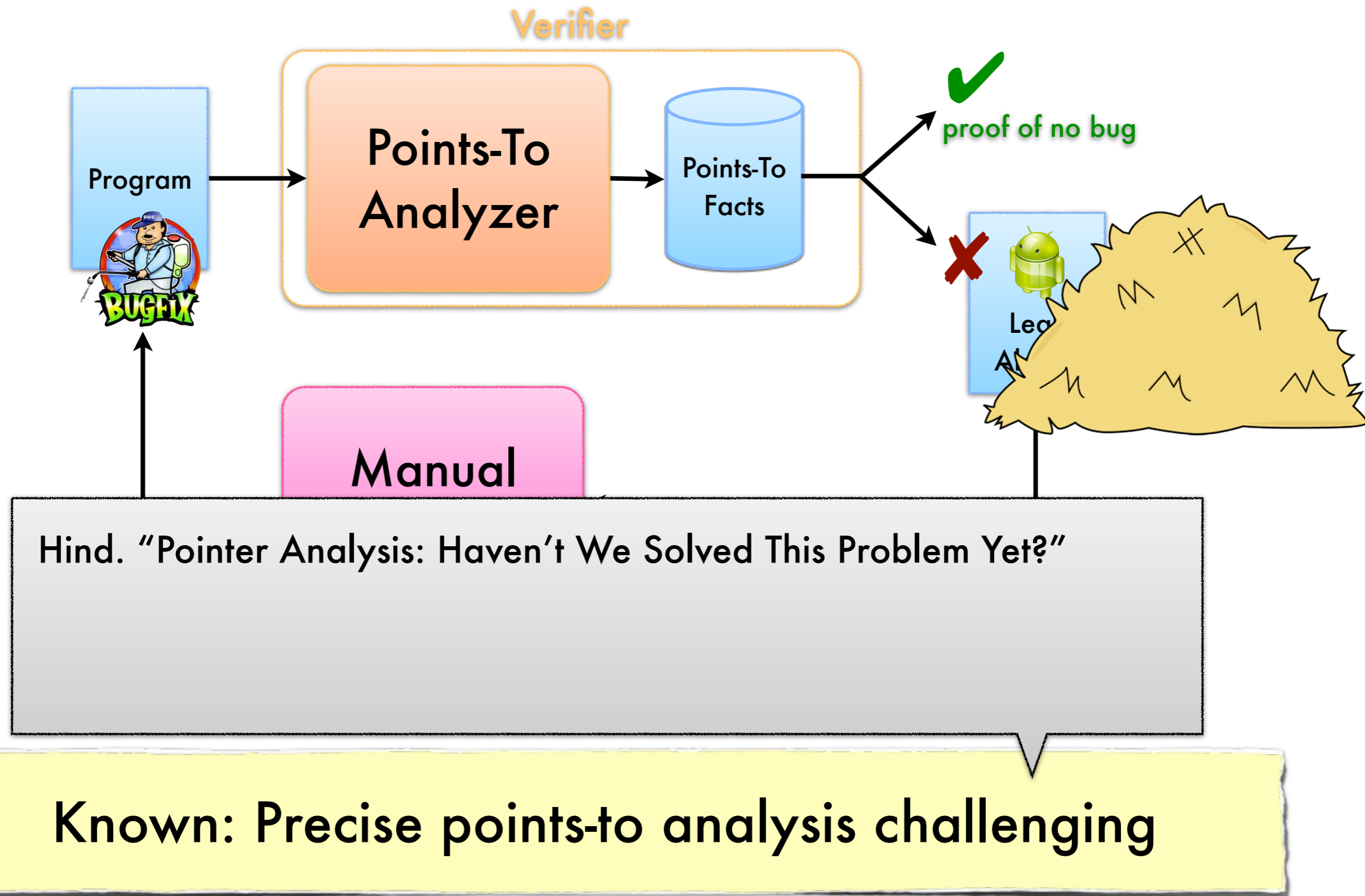
Known: Precise points-to analysis challenging

Thresher addresses alarm triage in a particularly challenging domain.

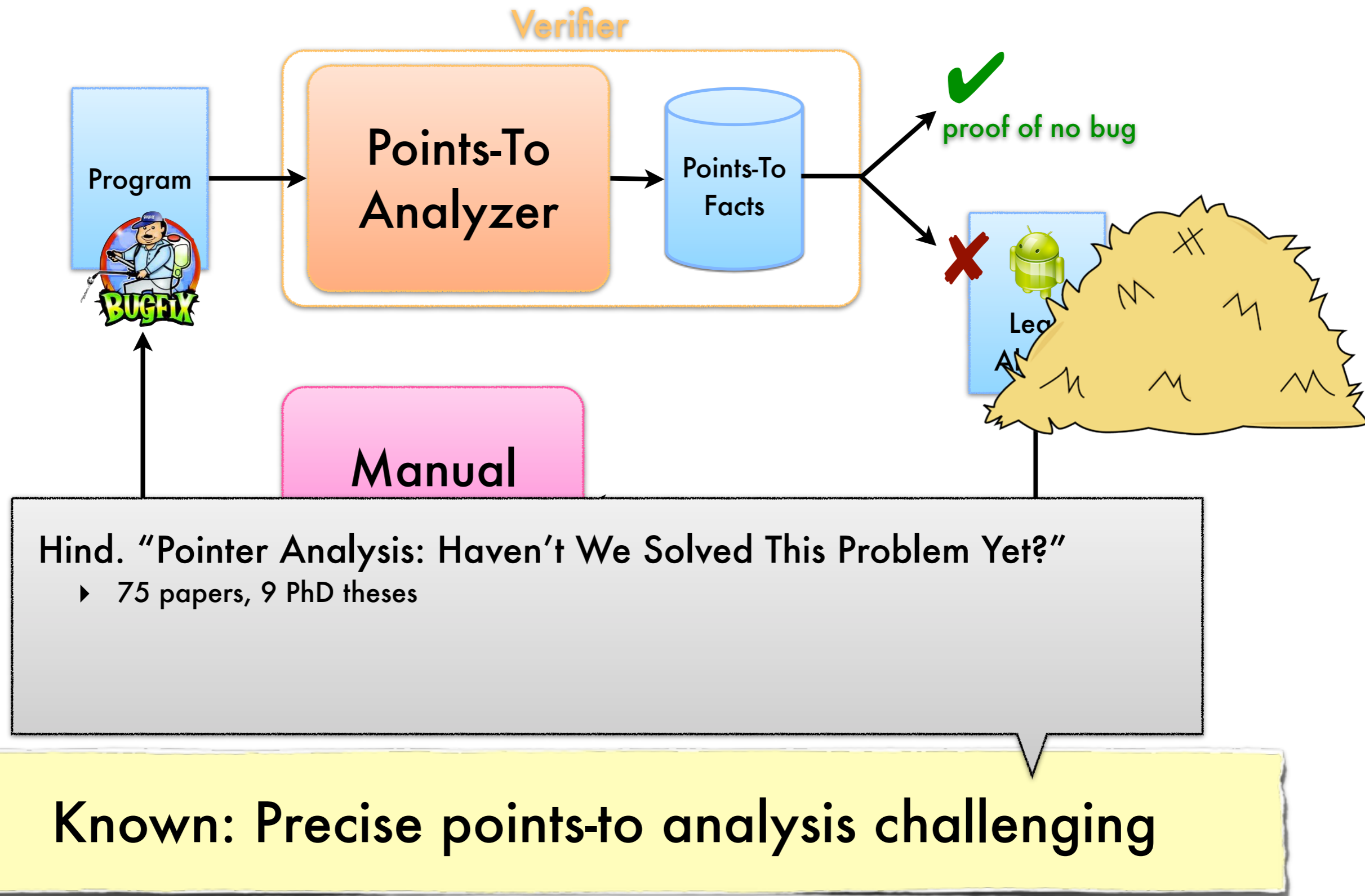


Known: Precise points-to analysis challenging

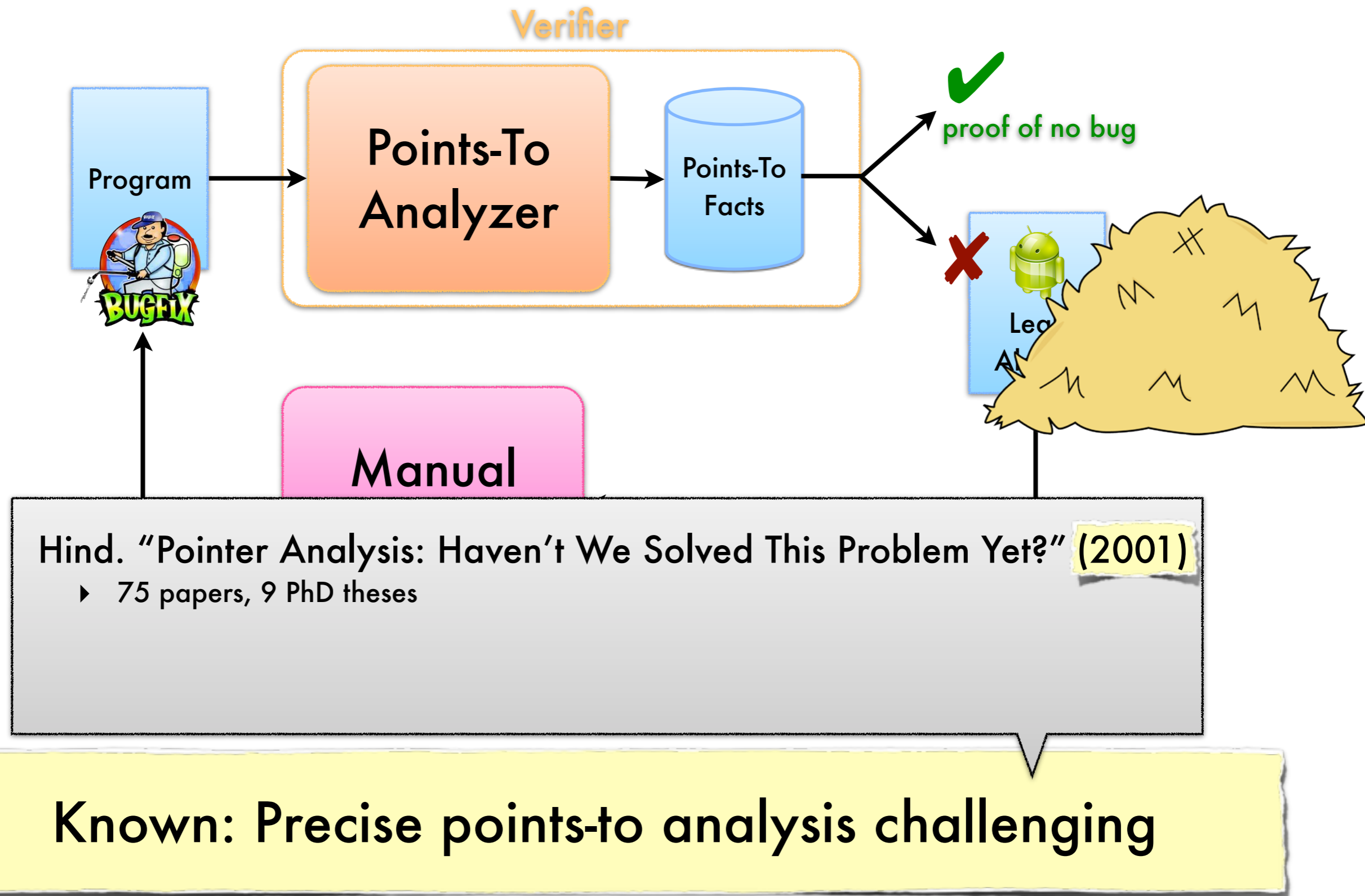
Thresher addresses alarm triage in a particularly challenging domain.



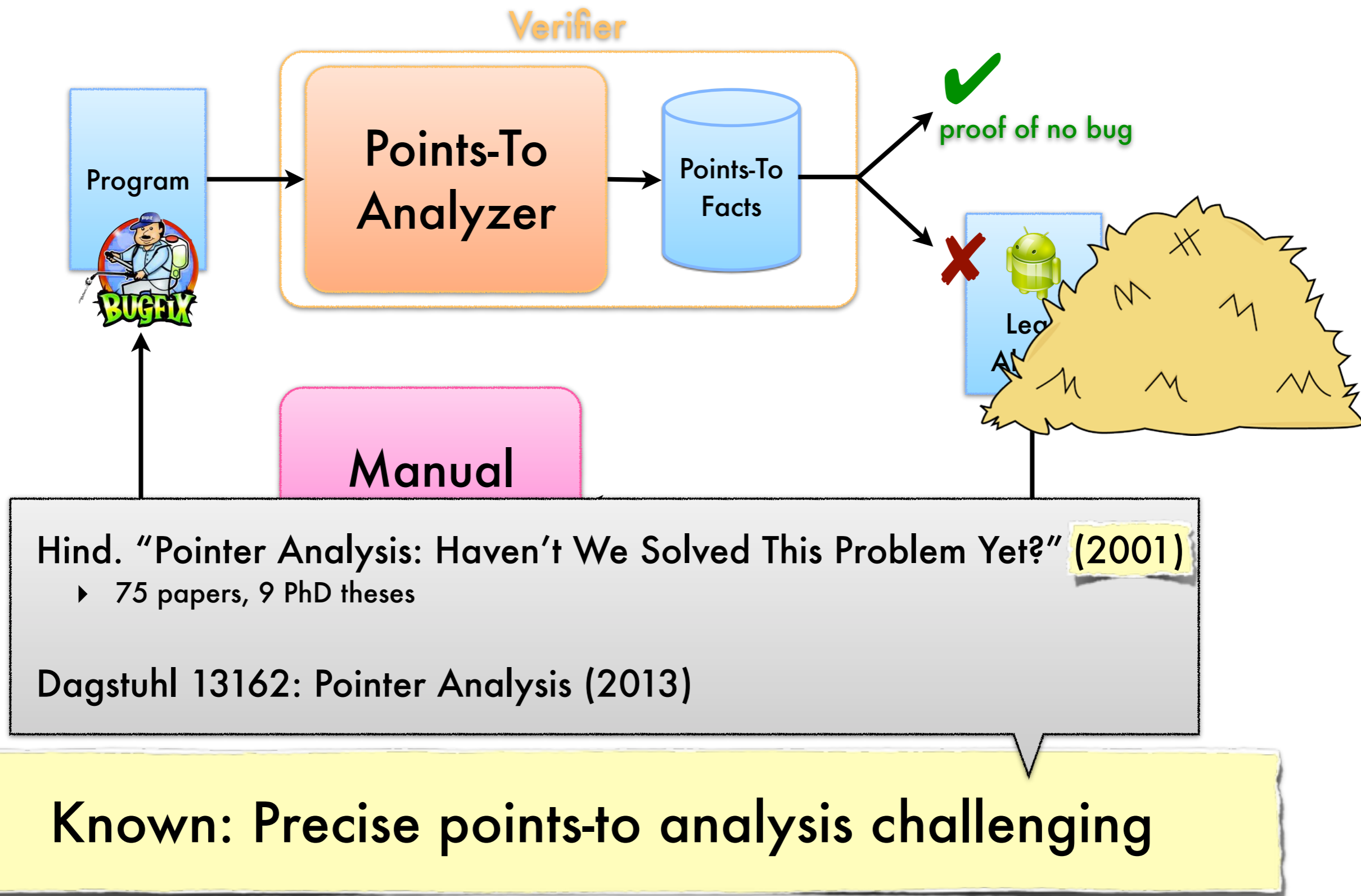
Thresher addresses alarm triage in a particularly challenging domain.



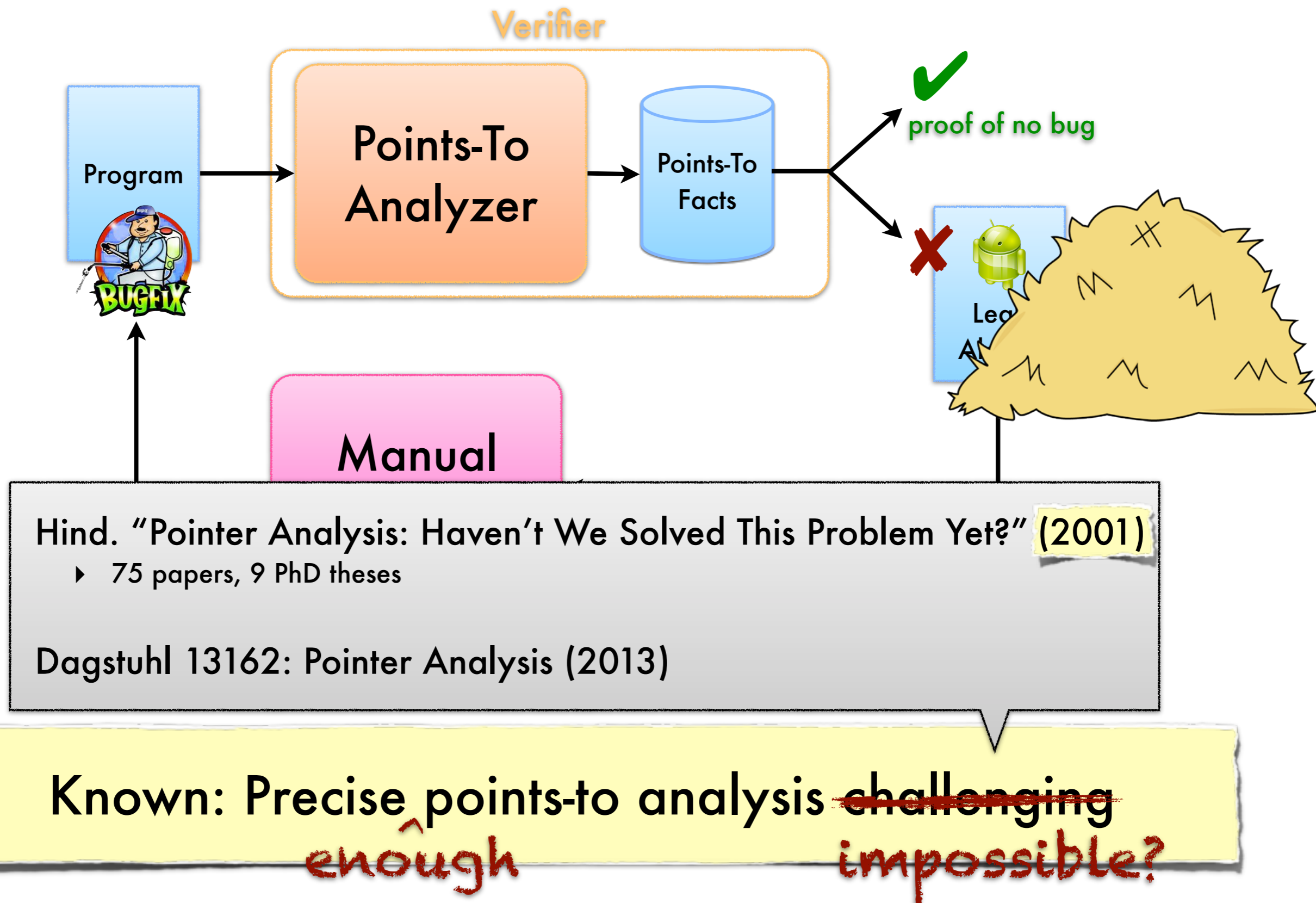
Thresher addresses alarm triage in a particularly challenging domain.



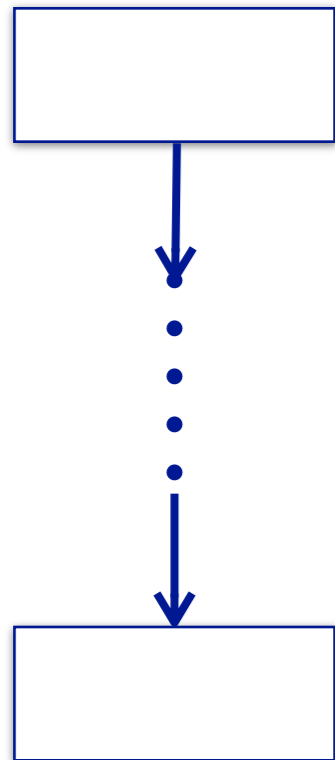
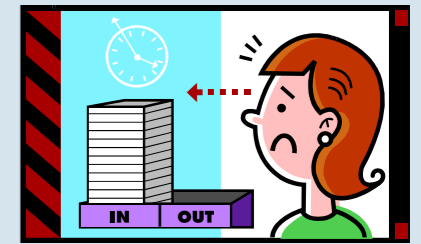
Thresher addresses alarm triage in a particularly challenging domain.



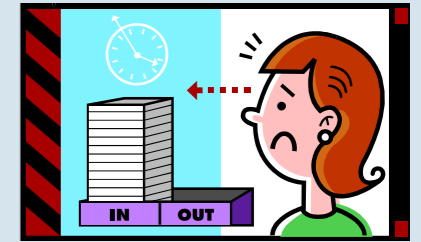
Thresher addresses alarm triage in a particularly challenging domain.



Manual triage is particularly hard for heap reachability reports.



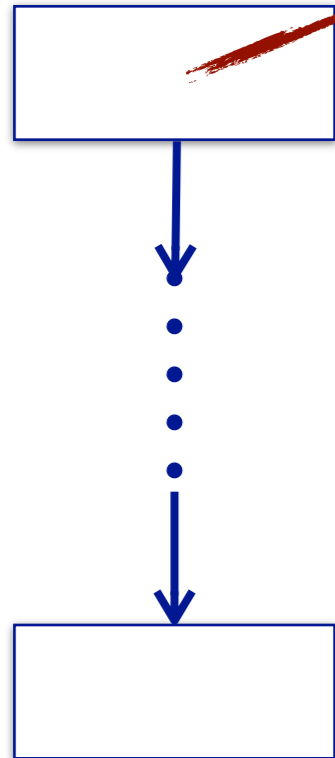
Manual triage is particularly hard for heap reachability reports.



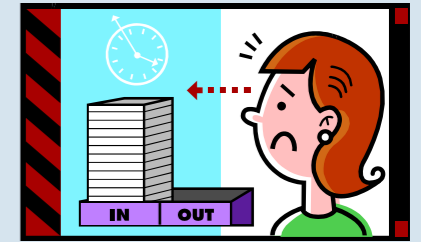
allocated here

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
            return;
        }
        NetworkStream ns = server.GetStream();
        int recv = ns.Read(data, 0, data.Length);
        stringData = Encoding.
        ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        while (true){
            input = Console.ReadLine();
            if (input == "exit") break;
            newchild.Properties["ou"].Add
            ("Auditing Department");
            if (input == "exit") break;
            newchild.CommitChanges();
            newchild.Close();
            ~child();
        }
    }
}
```

MyClass1.java



Manual triage is particularly hard for heap reachability reports.



allocated here



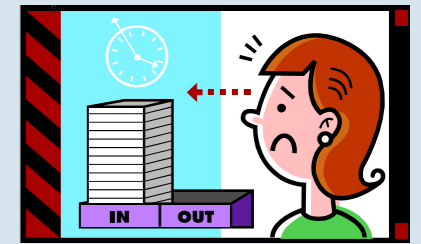
```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
```

MyC

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
NetworkStream ns = server.GetStream();
int recv = ns.Read(data, 0, data.Length);
stringData = Encoding.ASCII.GetString(data, 0, recv);
Console.WriteLine(stringData);
while (true){
    input = Console.ReadLine();
    if (input == "exit") break;
    newchild.Properties["ou"].Add("Auditing Department");
    if (input == "Auditing Department")
        newchild.CommitChanges();
    newchild.Close();
}
```

LibraryClass1.java

Manual triage is particularly hard for heap reachability reports.



allocated here



```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

MyC

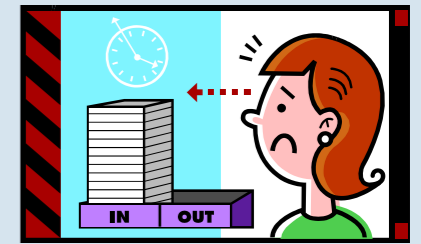
```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

Library

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
NetworkStream ns = server.GetStream();
int recv = ns.Read(data, 0, data.Length);
stringData = Encoding.
ASCII.GetString(data, 0, recv);
Console.WriteLine(stringData);
while(true){
    input = Console.ReadLine();
    if (input == "exit") break;
    newchild.Properties["ou"].Add
    ("Auditing Department");
    newchild.CommitChanges();
    newchild.Close();
}
```

MyClass2.java

Manual triage is particularly hard for heap reachability reports.



allocated here



```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

MyC

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

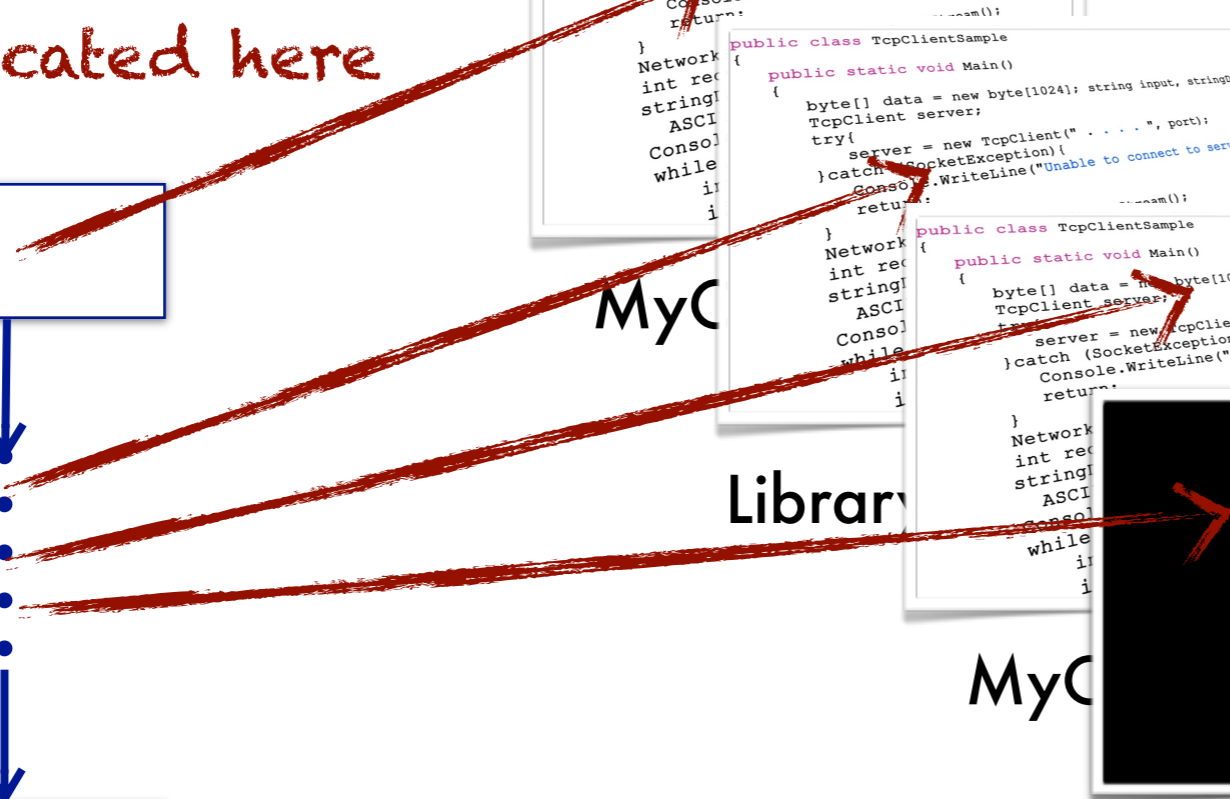
Library

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

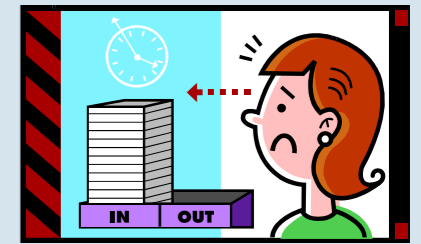
MyC



Library2Class1.class



Manual triage is particularly hard for heap reachability reports.



allocated here



```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCI
Conso
while
i;
i;
```

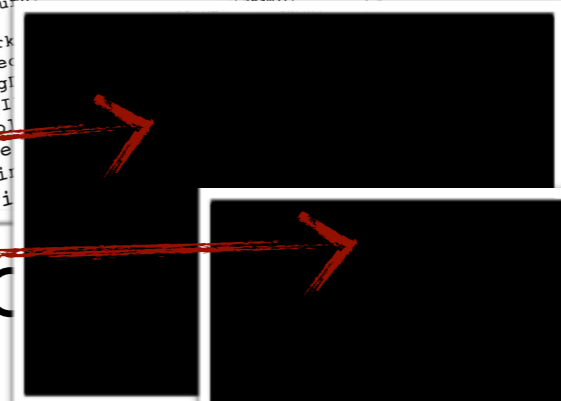
MyC

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCI
Conso
while
i;
i;
```

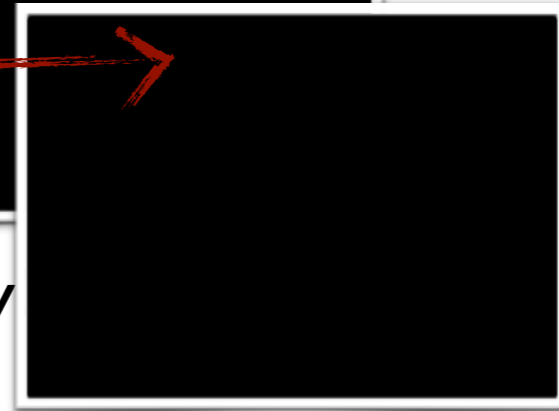
Library

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCI
Conso
while
i;
i;
```

MyC

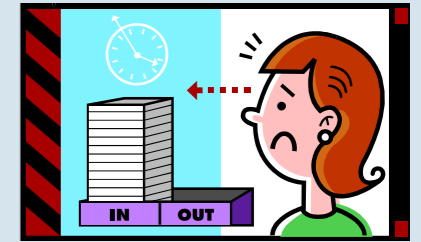


Library



java.util.HashMap.class

Manual triage is particularly hard for heap reachability reports.



allocated here



```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

MyC

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

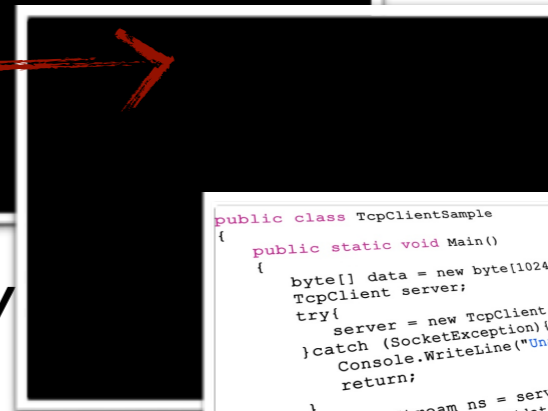
Library

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
Network
int rec
string
ASCII
Console
while
i;
i;
```

MyC



Library

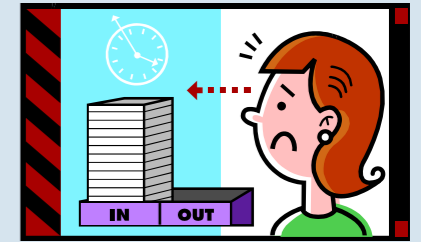


java.util.

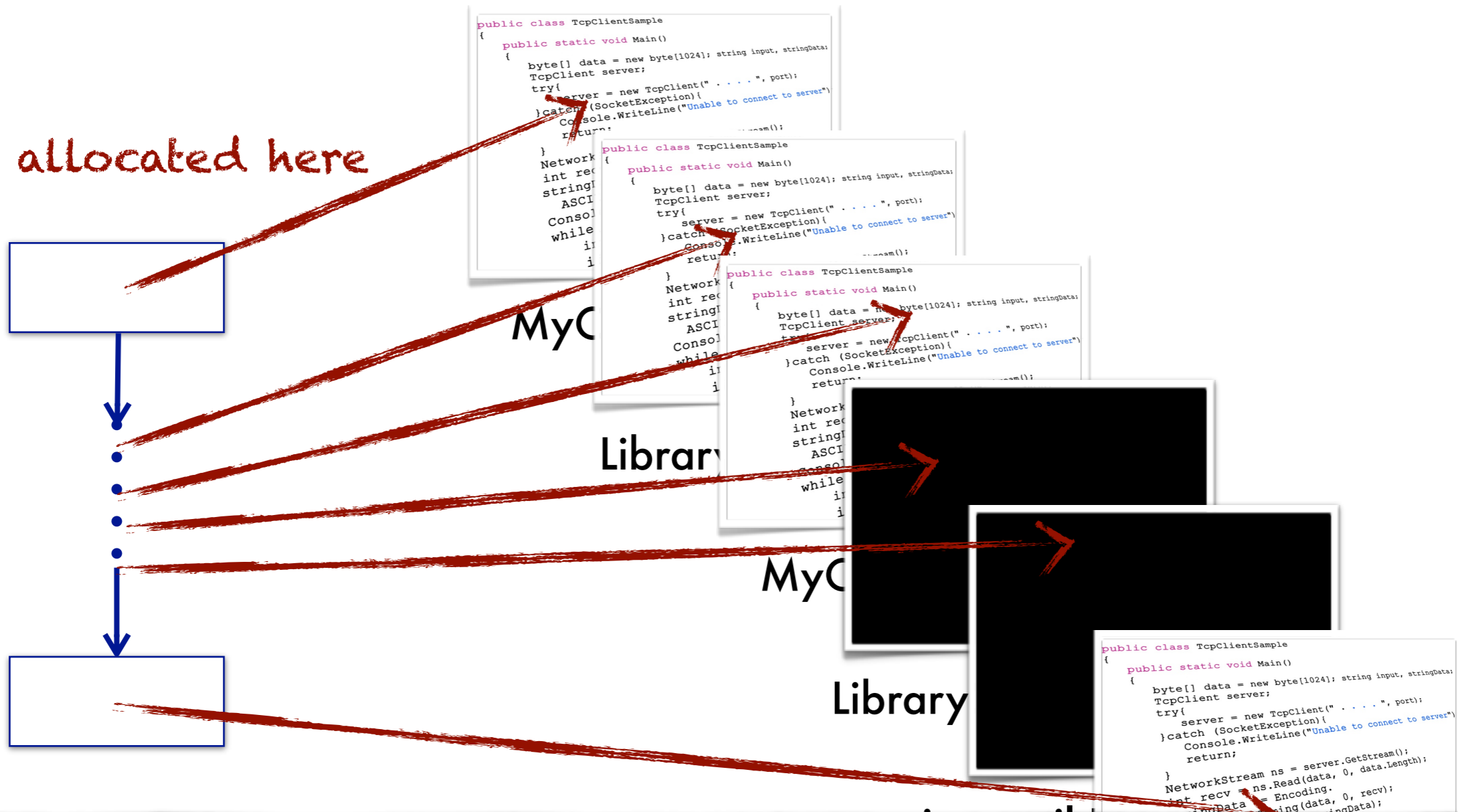
```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
        }
        return;
    }
}
NetworkStream ns = server.GetStream();
int recv = ns.Read(data, 0, data.Length);
stringData = Encoding.ASCII.GetString(data, 0, recv);
Console.WriteLine(stringData);
while (true) {
    input = Console.ReadLine();
    if (input == "exit") break;
    newchild.Properties["ou"].Add("Auditing Department");
    newchild.CommitChanges();
    newchild.Close();
}
```

MyClass3.java

Manual triage is particularly hard for heap reachability reports.

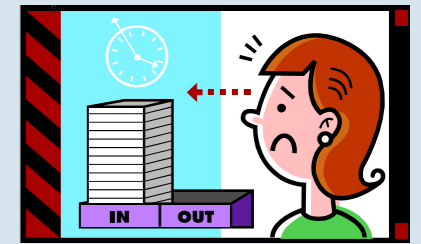


allocated here

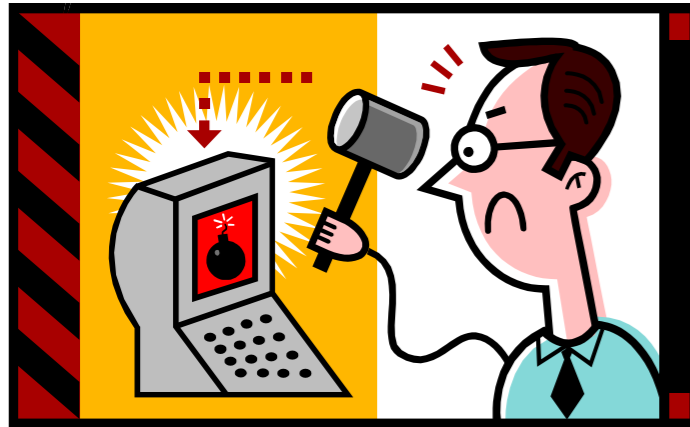
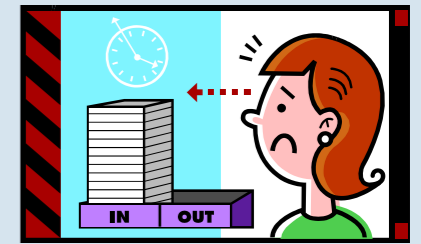


Get abstract heap path + maybe allocation sites
Guesstimate: >1 to 2 hours per alarm to triage "well"

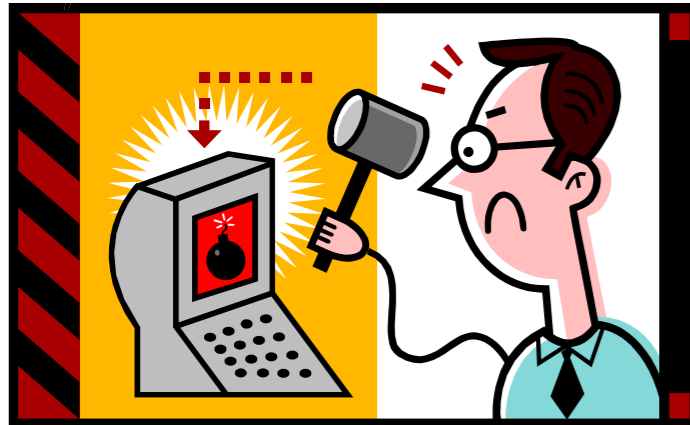
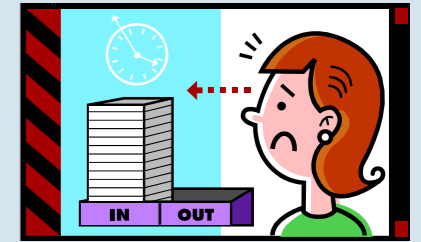
Examining manual triage ...



Examining manual triage ...

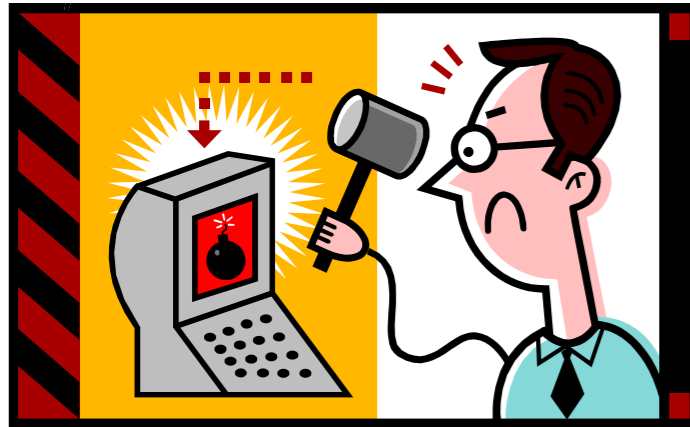
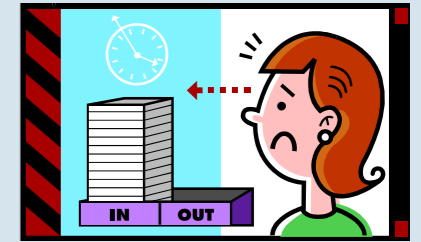


Examining manual triage ...



What does the user need to do with an alarm?
He starts at, say, line 142 and **traces back** to see **if a bug is possible** given what's happening.

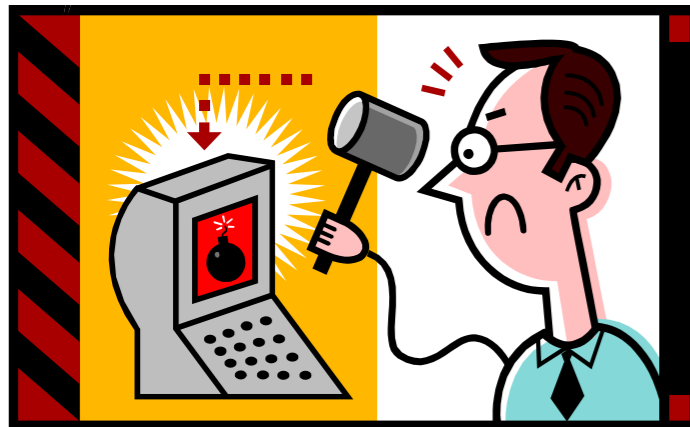
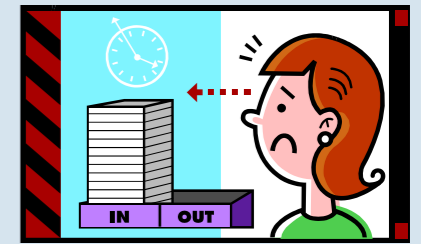
Examining manual triage ...



What does the user need to do with an alarm?
He starts at, say, line 142 and **traces back** to see **if a bug is possible** given what's happening.

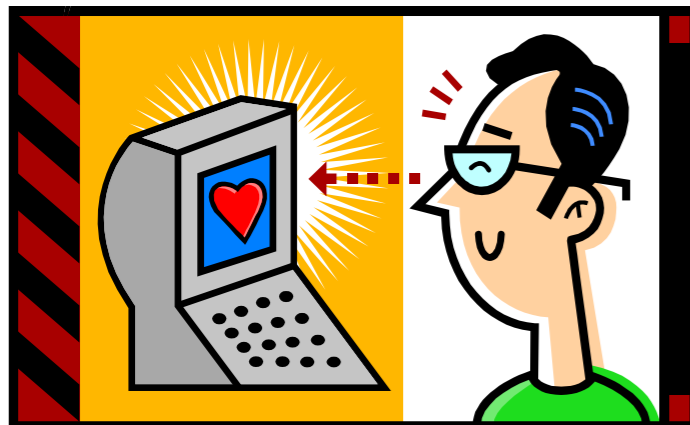
We can do this with analysis!

Examining manual triage ...

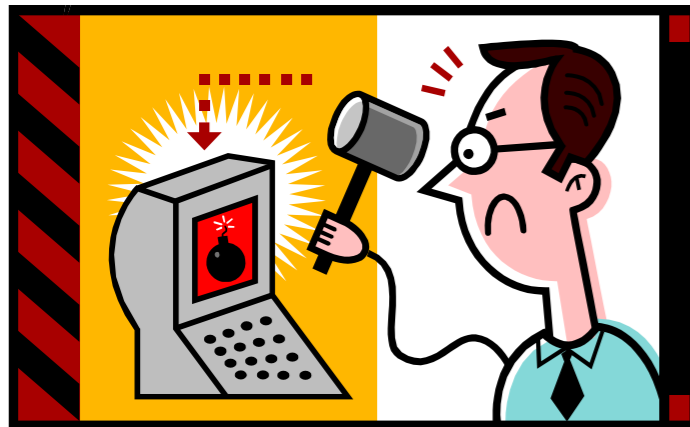
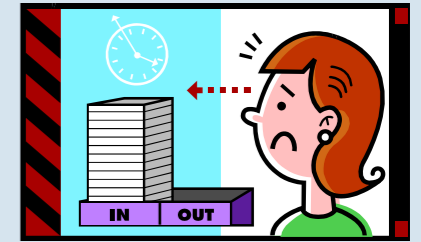


What does the user need to do with an alarm?
He starts at, say, line 142 and **traces back** to see **if a bug is possible** given what's happening.

We can do this with analysis!

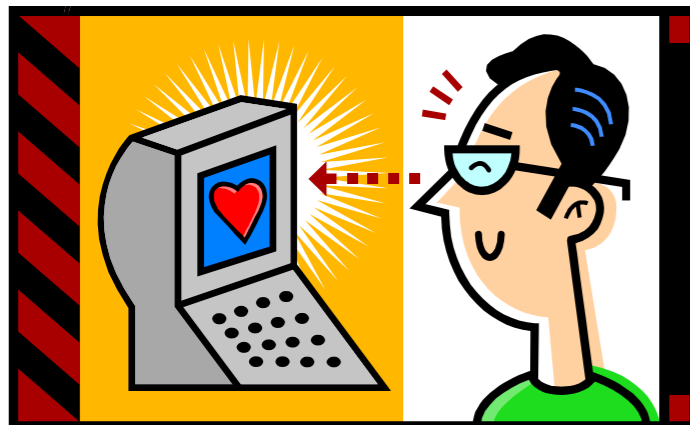


Examining manual triage ...



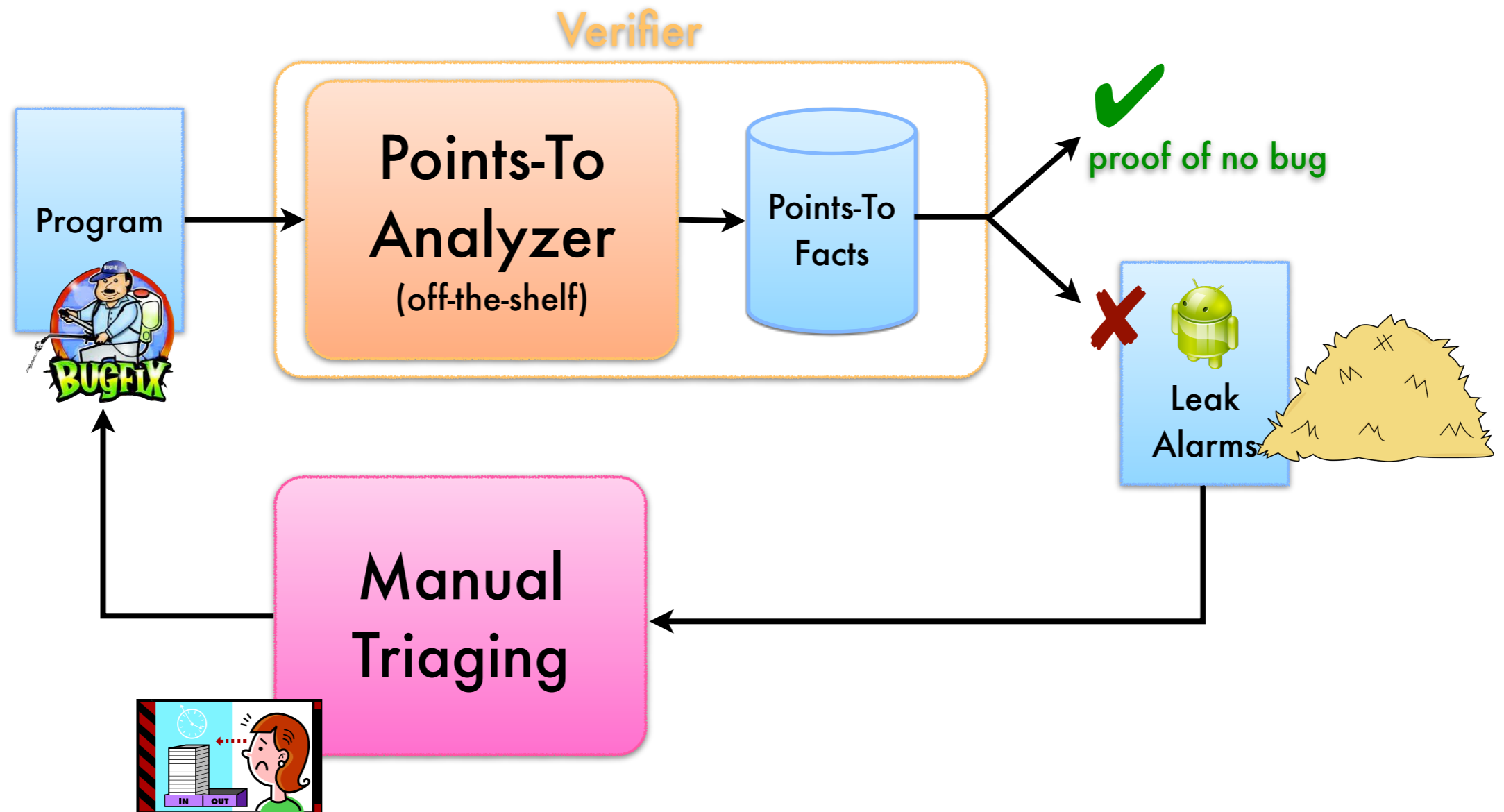
What does the user need to do with an alarm?
He starts at, say, line 142 and **traces back** to see **if a bug is possible** given what's happening.

We can do this with analysis!

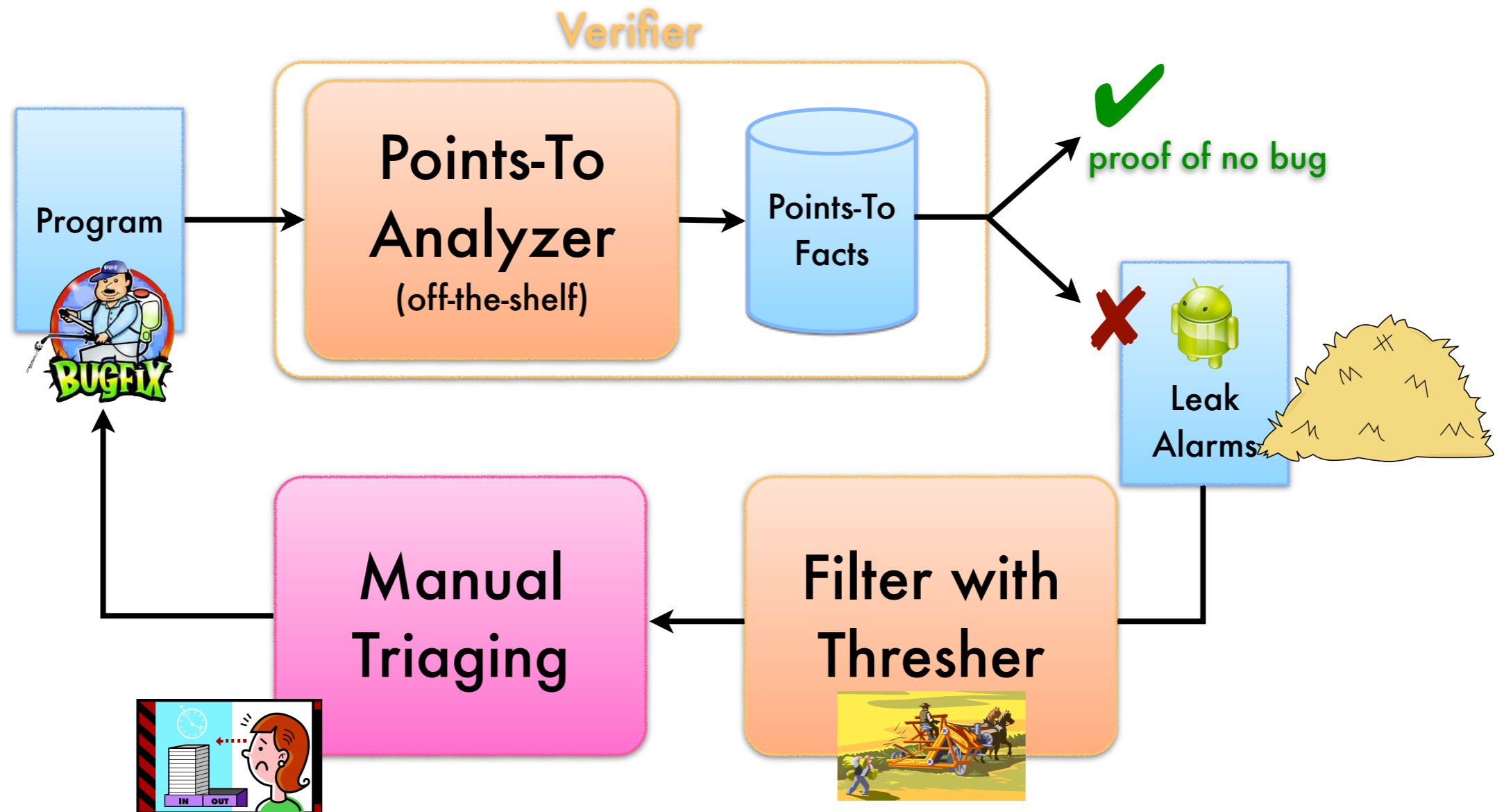


If we filter most false alarms, the user can triage more quickly and get to true bugs earlier (without frustration).

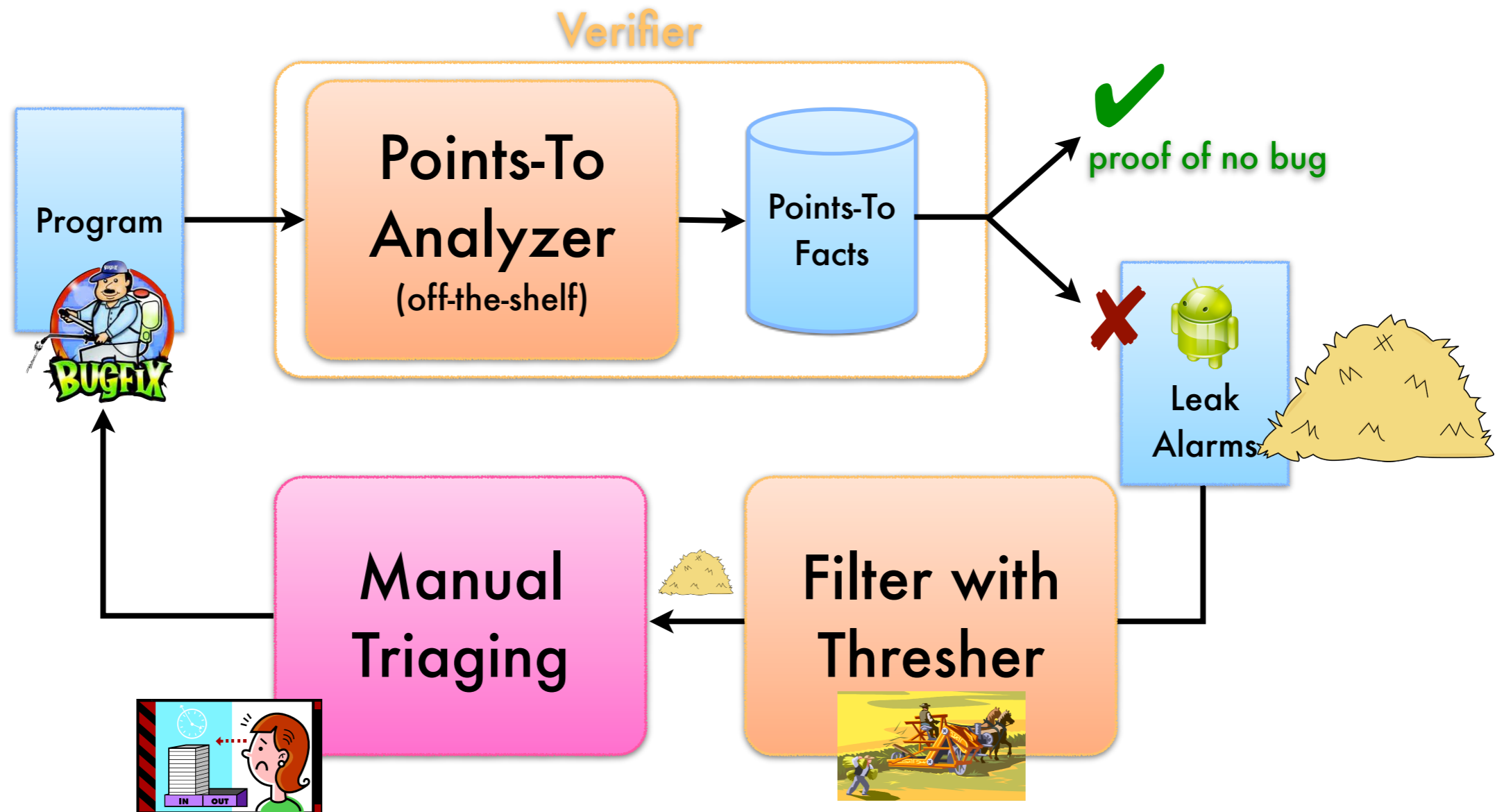
Thresher filters out false alarms by refuting them one-by-one.



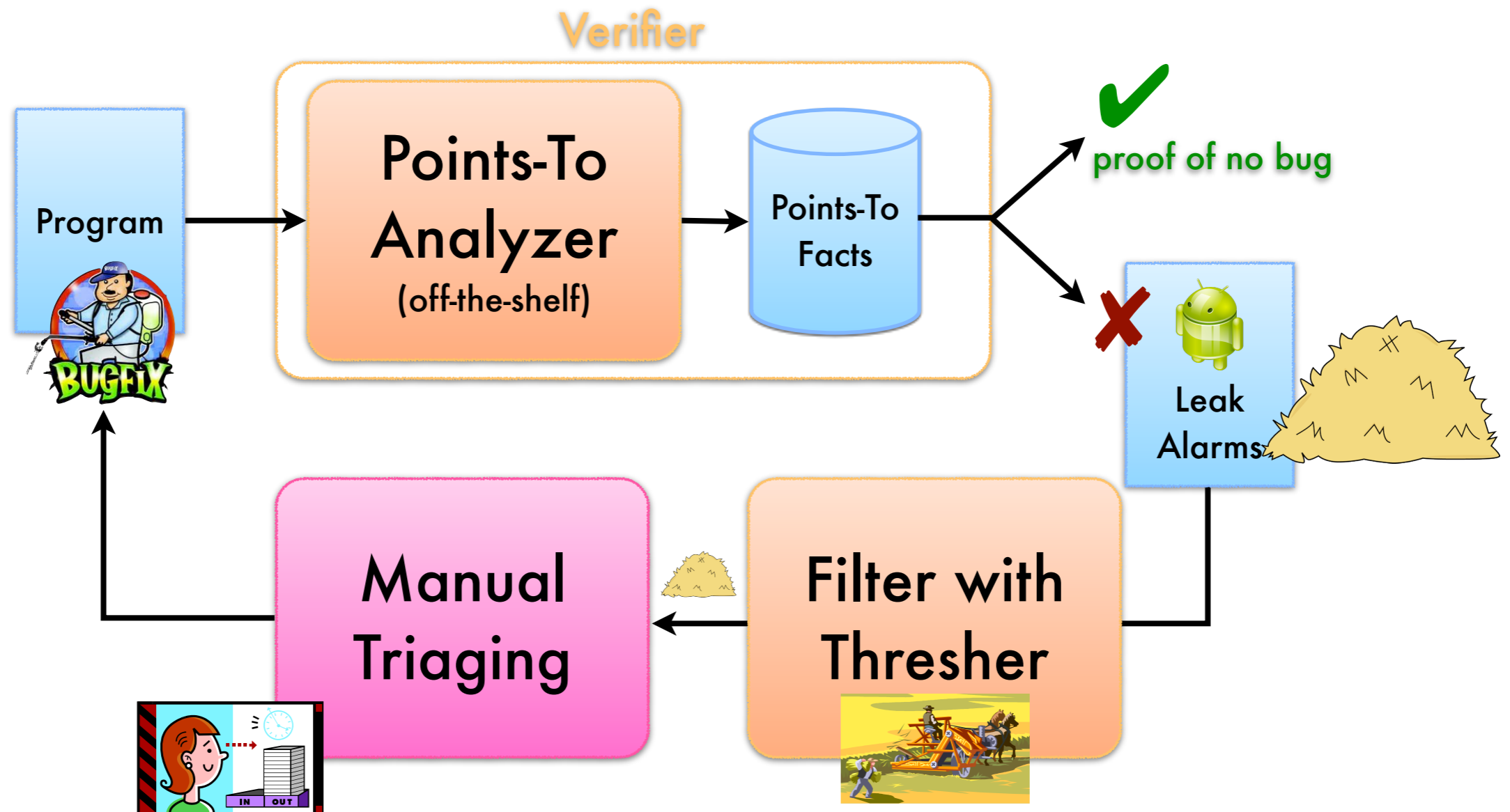
Thresher filters out false alarms by refuting them one-by-one.



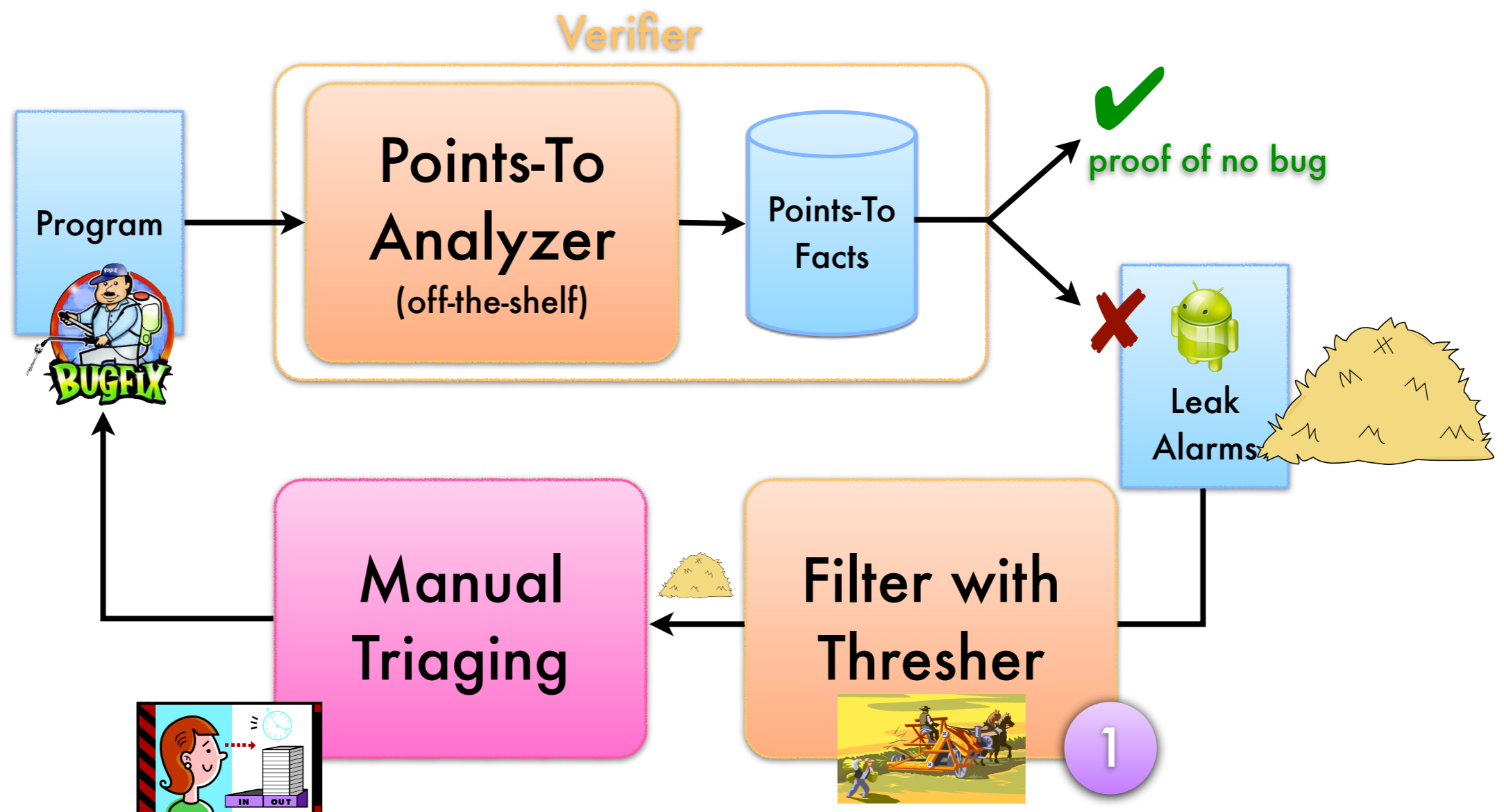
Thresher filters out false alarms by refuting them one-by-one.



Thresher filters out false alarms by refuting them one-by-one.

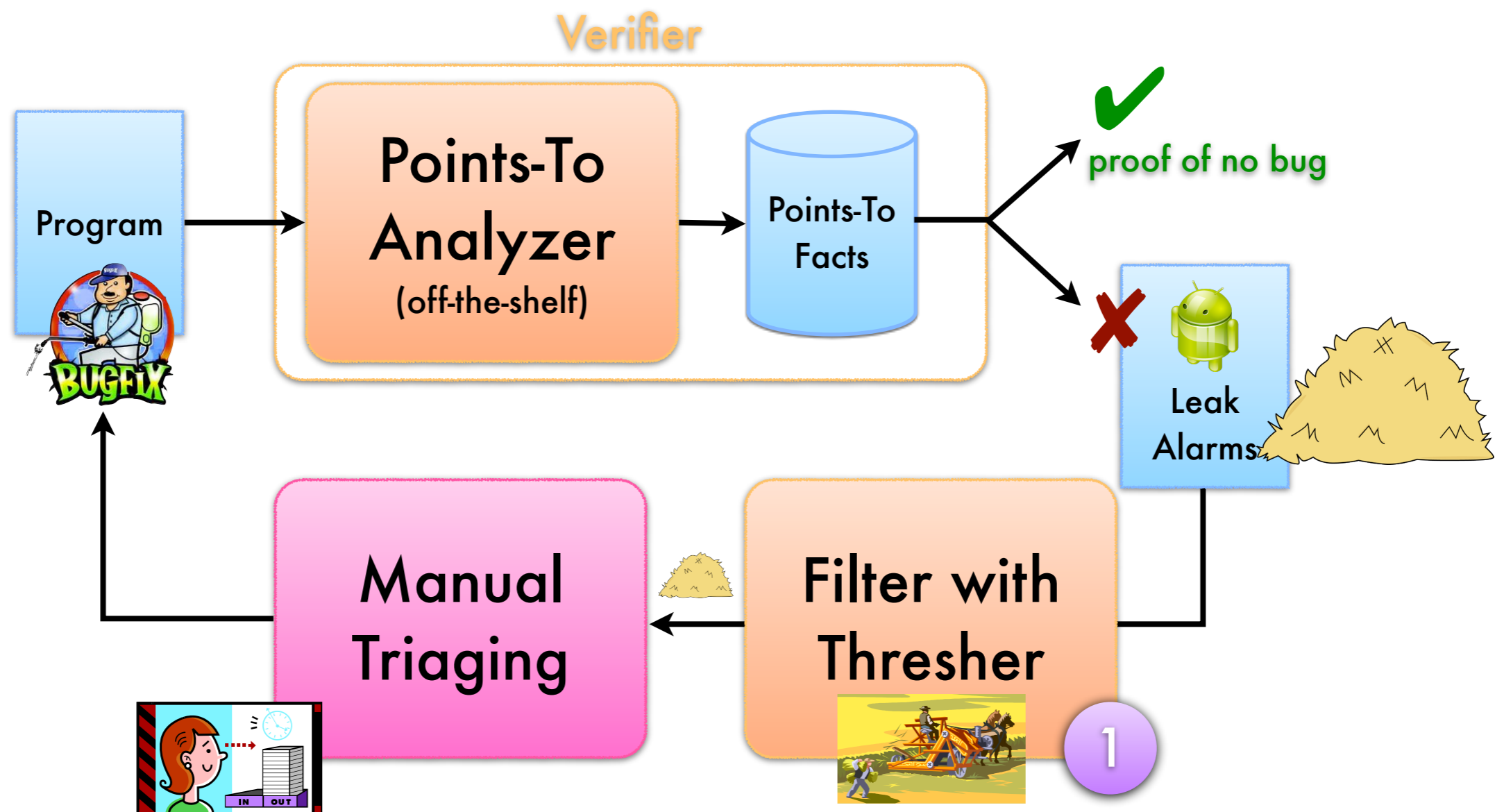


Thresher filters out false alarms by refuting them one-by-one.



Idea 1: Refute points-to on-demand with **second** “uber-precise” filter analysis

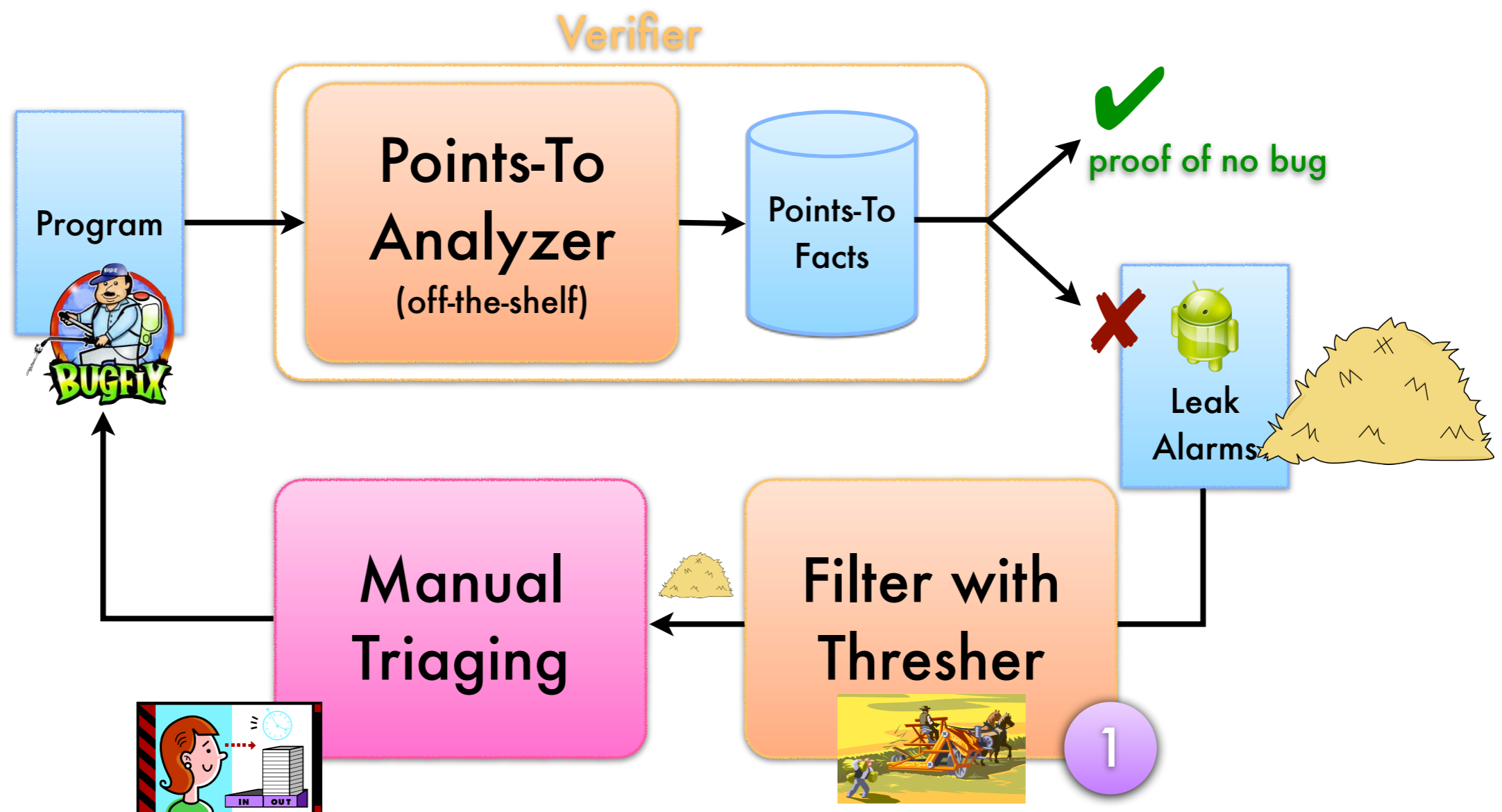
Thresher filters out false alarms by refuting them one-by-one.



Idea 1: Refute points-to on-demand with **second** "uber-precise" filter analysis

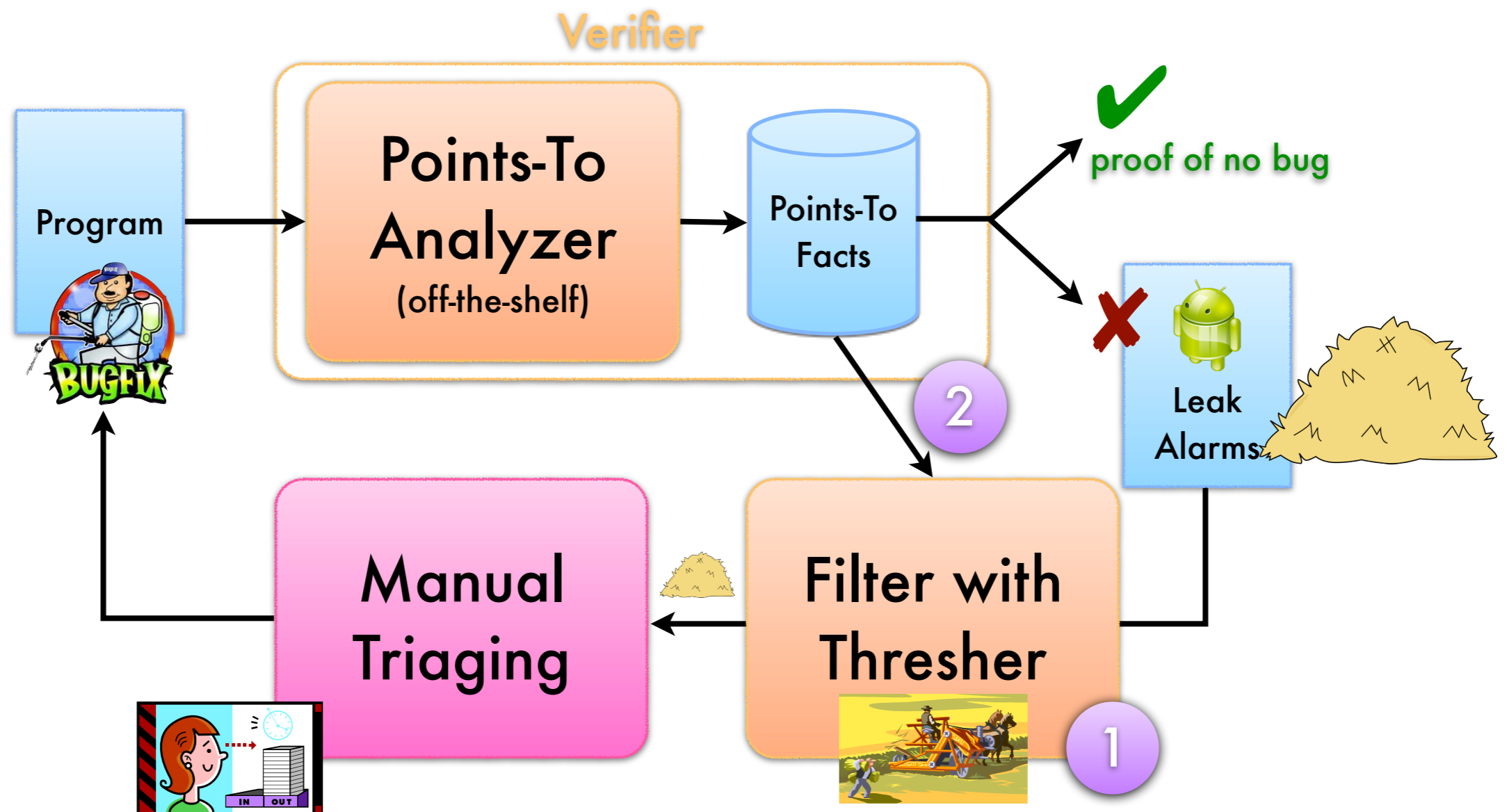
*-sensitive

Thresher filters out false alarms by refuting them one-by-one.



Idea 1: Refute points-to on-demand with **second** “uber-precise” filter analysis

Thresher filters out false alarms by refuting them one-by-one.



Idea 1: Refute points-to on-demand with **second** “uber-precise” filter analysis

Idea 2: Leverage the **facts** from the first analysis in the filter analysis to scale

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an
execution where at
some time

o



o'



of type T .

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an execution where at some time

o



o'



of type T .

A. Why does object o possibly point to o' ?

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an execution where at some time

o



o'



of type T .

- A. Why does object o possibly point to o' ?
- B. Because statement s may execute to make o point to o'

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an execution where at some time

o



o'



of type T .

- A. Why does object o possibly point to o' ?
- B. Because statement s may execute to make o point to o'
- A. Why does statement s cause o to point to o' ?

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an execution where at some time

o



o'



of type T .

- A. Why does object o possibly point to o' ?
- B. Because statement s may execute to make o point to o'
- A. Why does statement s cause o to point to o' ?
- B. Because before statement s , the program state could satisfy formula φ

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an
execution where at
some time

o



o'



of type T .

- A. Why does object o possibly point to o' ?
- B. Because statement s may execute to make o point to o'
- A. Why does statement s cause o to point to o' ?
- B. Because before statement s , the program state could satisfy formula φ
- A. Why can the state before statement s satisfy φ ?

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an execution where at some time

o



o'



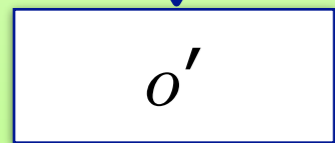
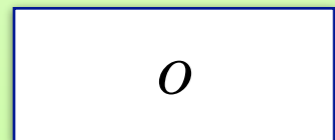
of type T .

- A. Why does object o possibly point to o' ?
 - B. Because statement s may execute to make o point to o'
- A. Why does statement s cause o to point to o' ?
 - B. Because before statement s , the program state could satisfy formula φ
- A. Why can the state before statement s satisfy φ ?
 - B. Because before the previous statement s' , the state could satisfy formula φ'

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an execution where at some time



A just asks “but why?”

B reasons about program semantics

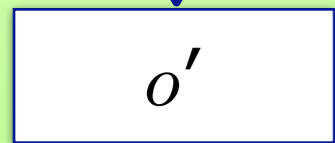
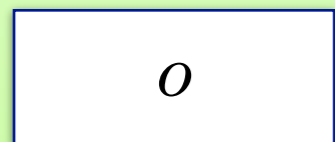
of type T .

- A. Why does object o possibly point to o' ?
- B. Because statement s may execute to make o point to o'
- A. Why does statement s cause o to point to o' ?
- B. Because before statement s , the program state could satisfy formula φ
- A. Why can the state before statement s satisfy φ ?
- B. Because before the previous statement s' , the state could satisfy formula φ'

Refutation analysis is “Proof by Contradiction” with the “But Why?” game



There **may** be an execution where at some time



A just asks “but why?”
B reasons about program semantics

of type T .

- A. Why does object o possibly point to o' ?
- B. Because statement s may execute to make o point to o'
- A. Why does statement s cause o to point to o' ?
- B. Because before statement s , the program state could satisfy formula φ
- A. Why can the state before statement s satisfy φ ?
- B. Because before the previous statement s' , the state could satisfy formula φ'

Theorem: If **B** can't give an answer, contradiction.
The alarm is false. It's been **refuted**. (**A** wins)

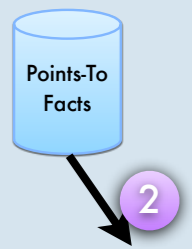
Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

Leverage first analysis by designing specialized constraint forms

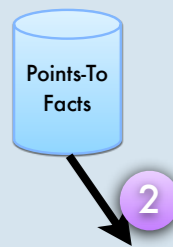


B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

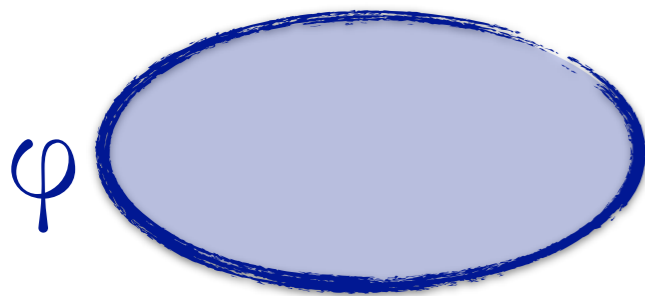
Leverage first analysis by designing specialized constraint forms



B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'



Leverage first analysis by designing specialized constraint forms

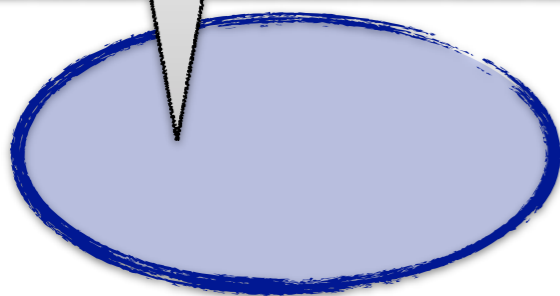
B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

set of possible states

φ



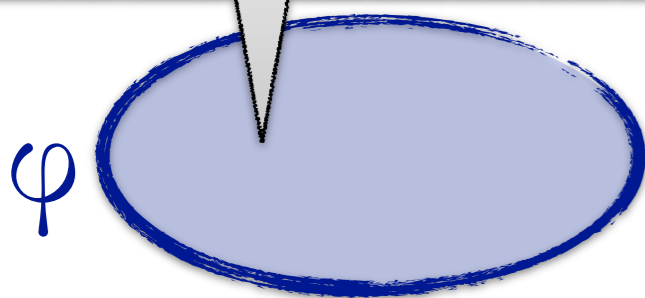
Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

if empty, then refuted (A wins) statement s' , a φ'

set of possible states



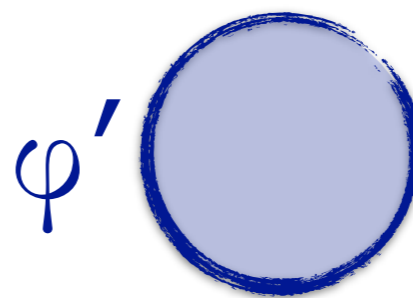
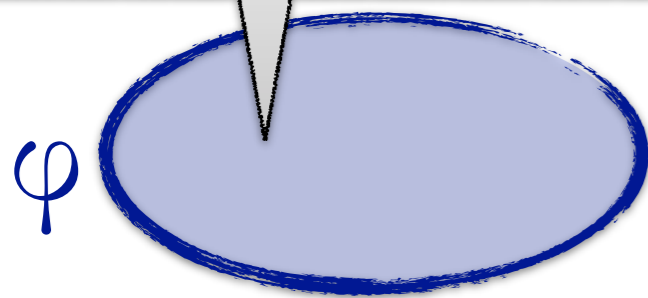
Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

if empty, then refuted (A wins) statement s' , a φ'

set of possible states



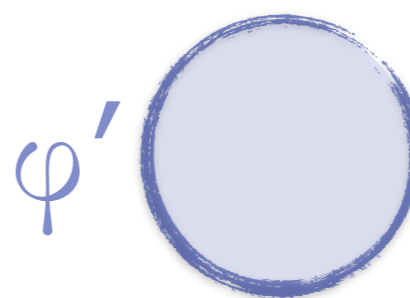
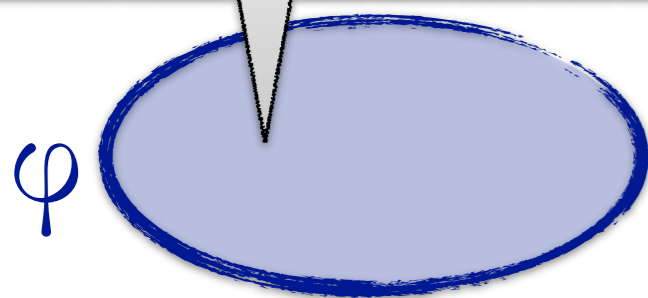
Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

if empty, then refuted (A wins) statement s' , a φ'

set of possible states



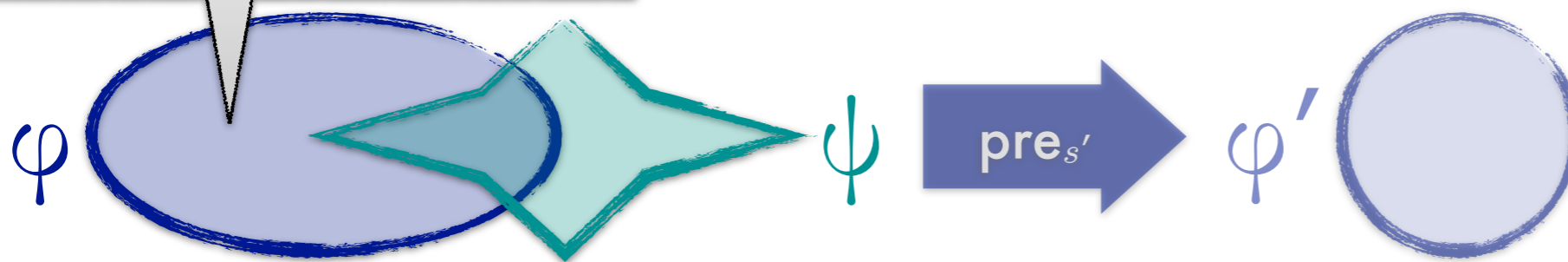
Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

if empty, then refuted (A wins) statement s' , a φ'

set of possible states



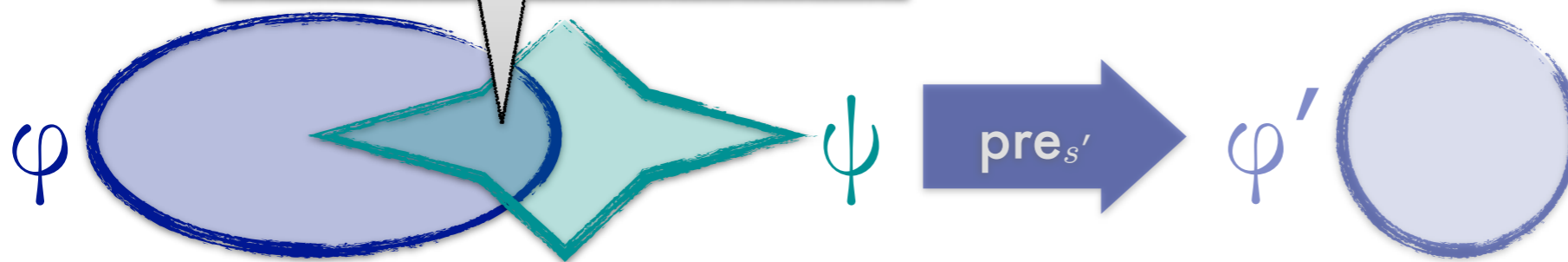
Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

if empty, then refuted (A wins) statement s' , a φ'

set of possible states

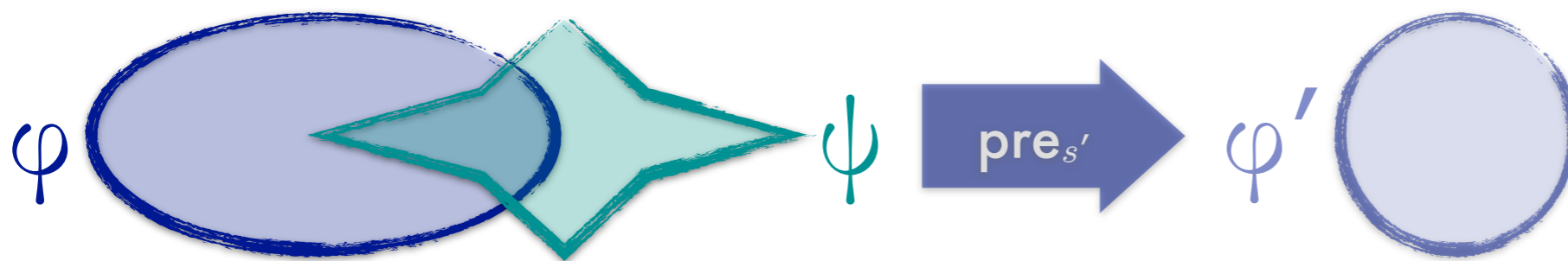


Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

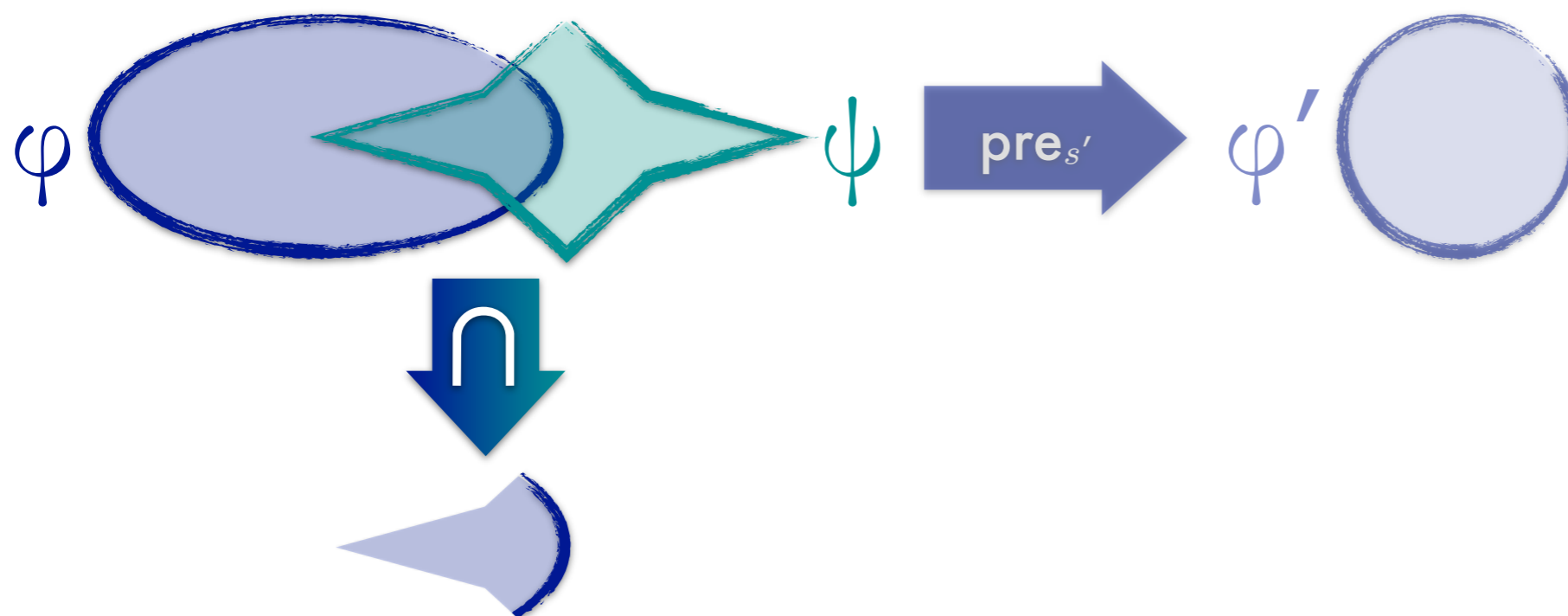


Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

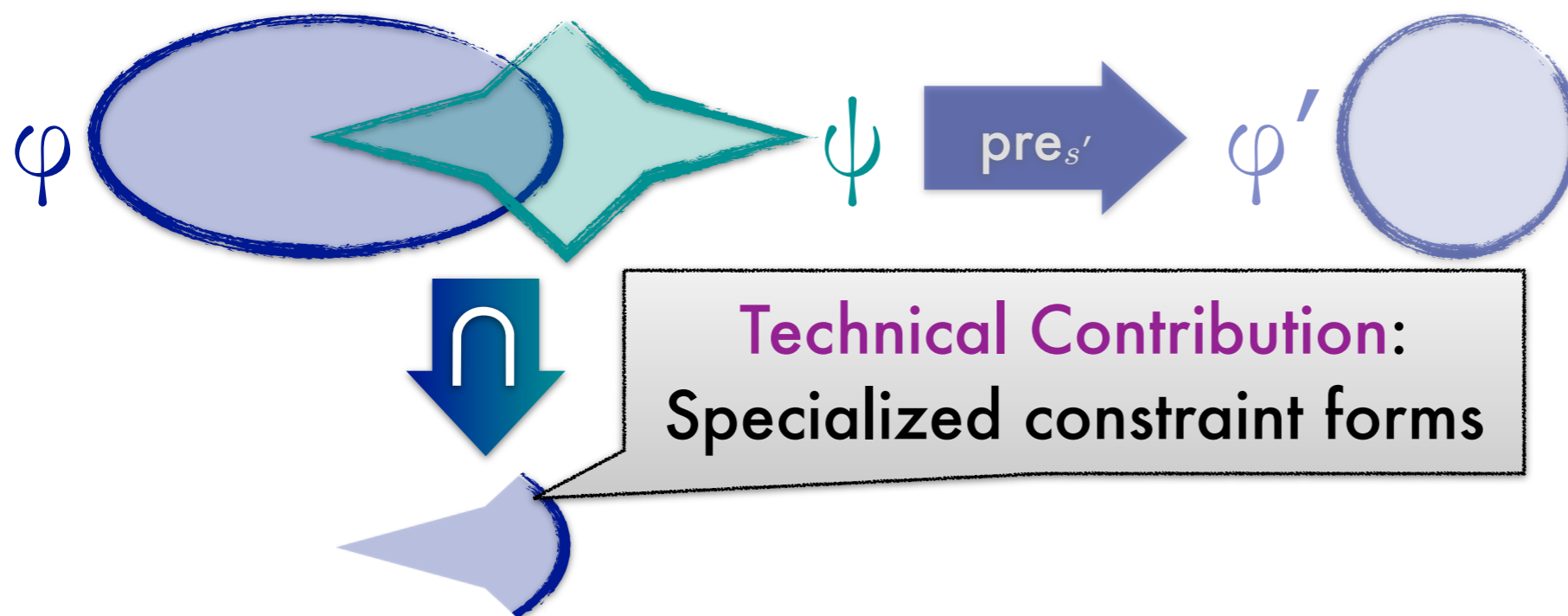


Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

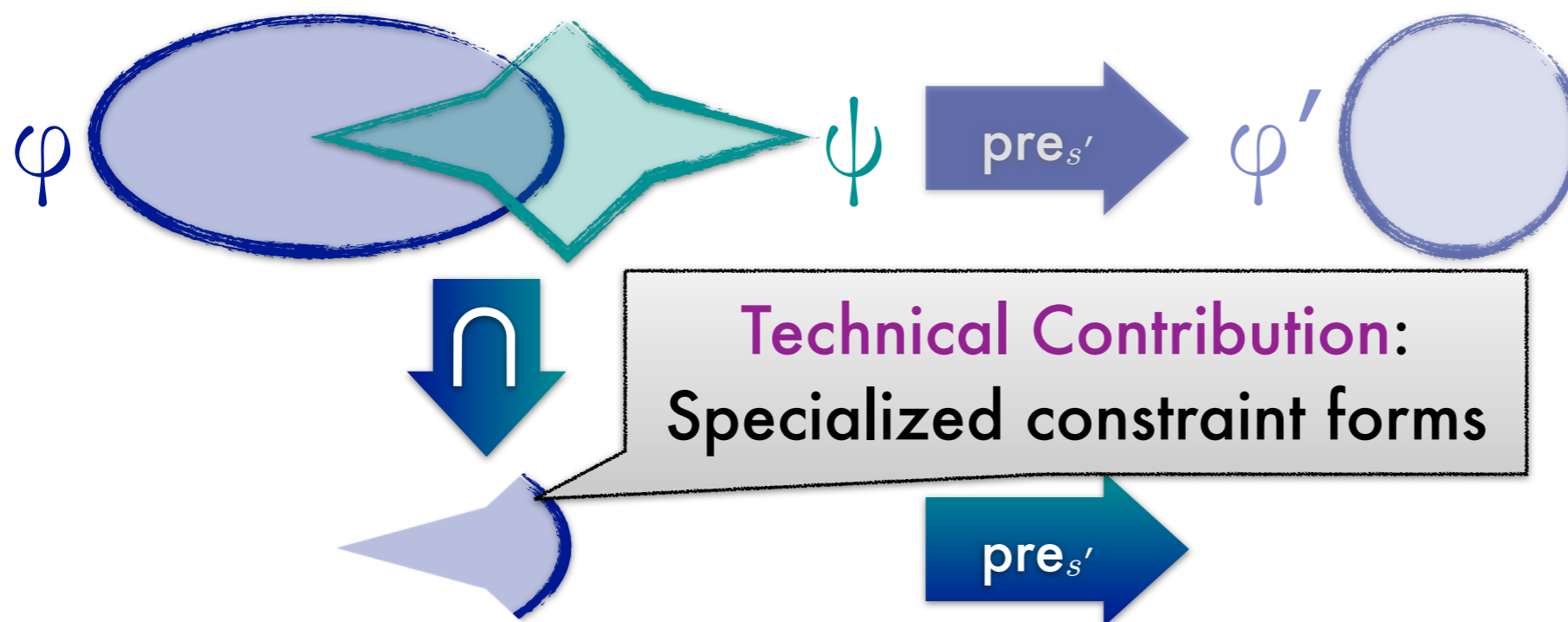


Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

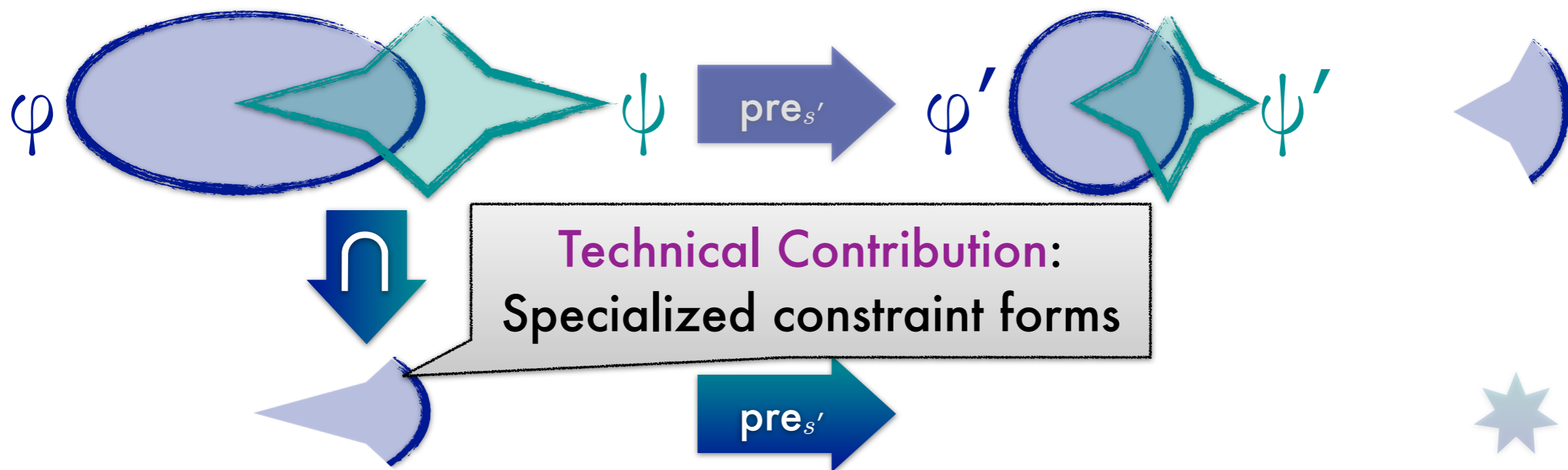


Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

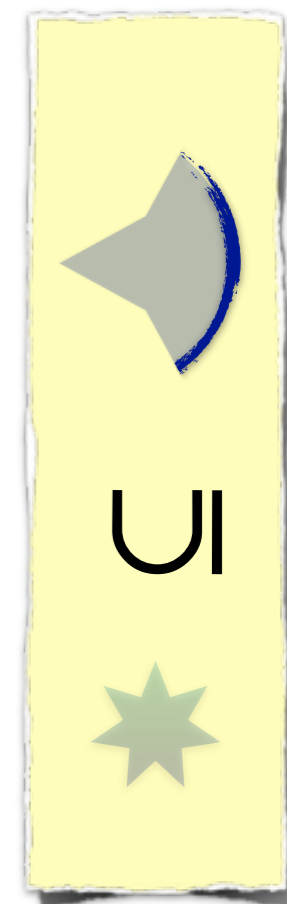
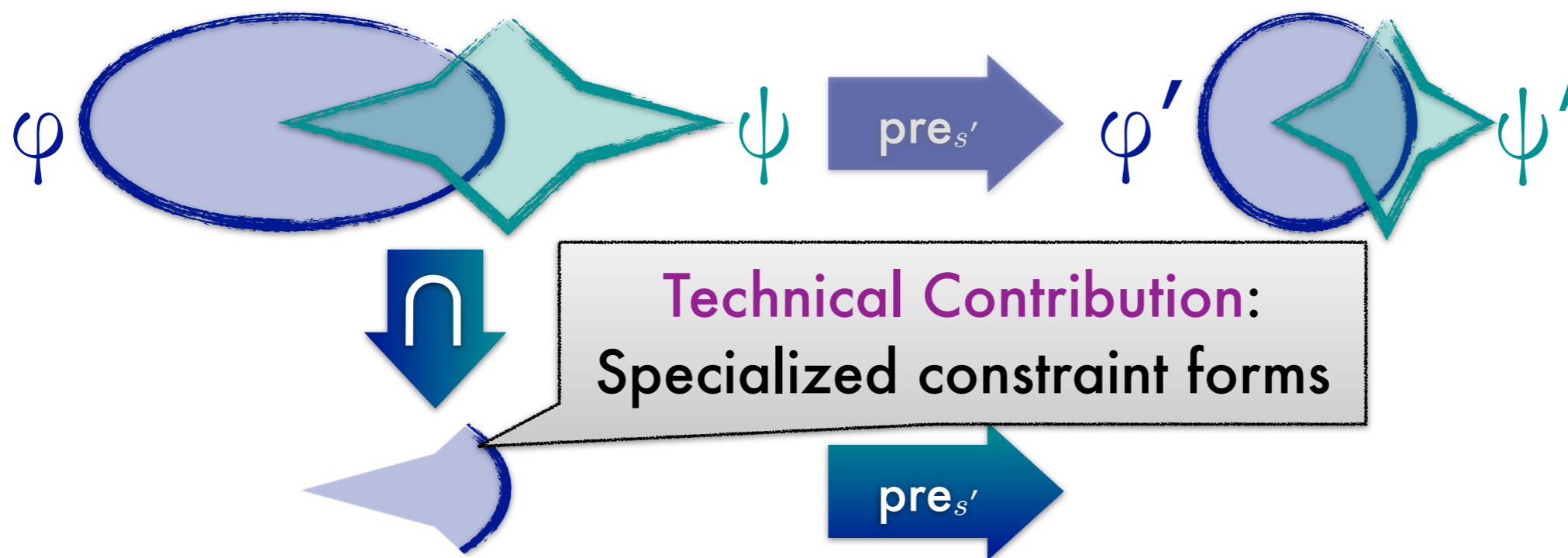


Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'



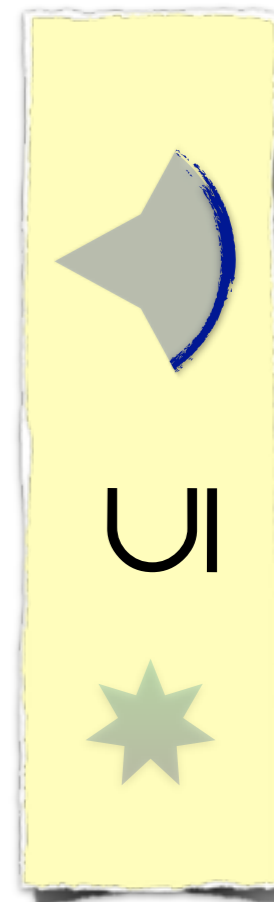
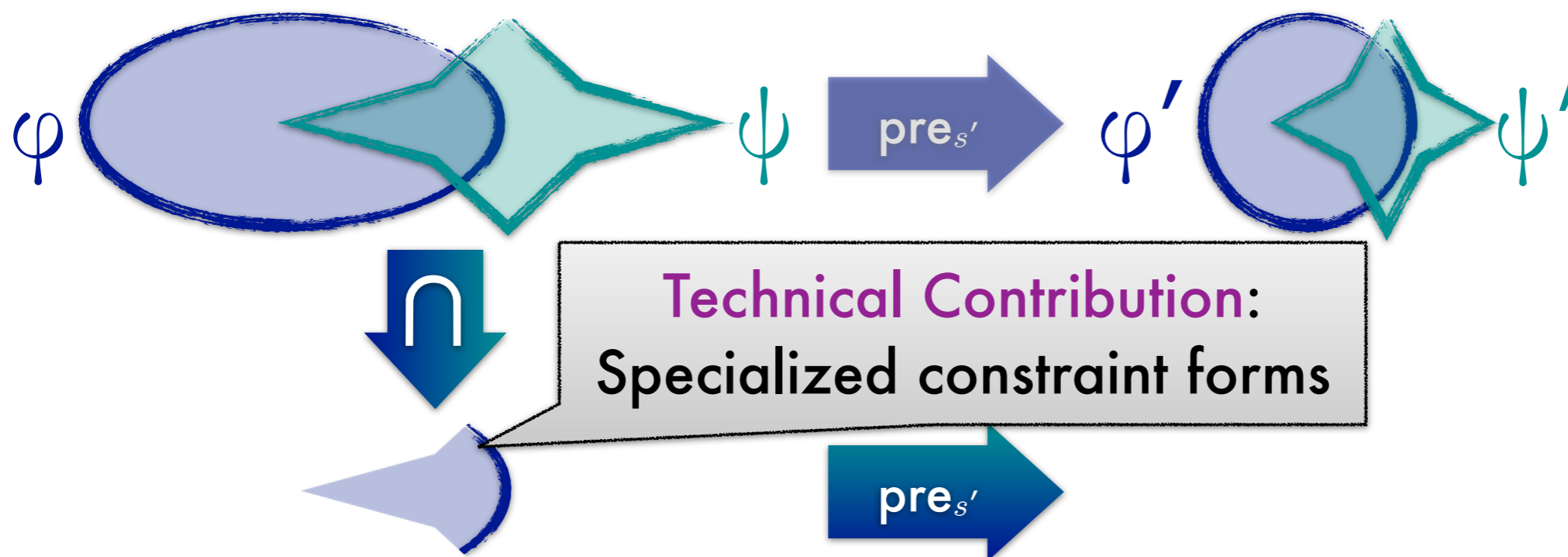
Leverage first analysis by designing specialized constraint forms

B. Because before statement s , the program state could satisfy formula φ

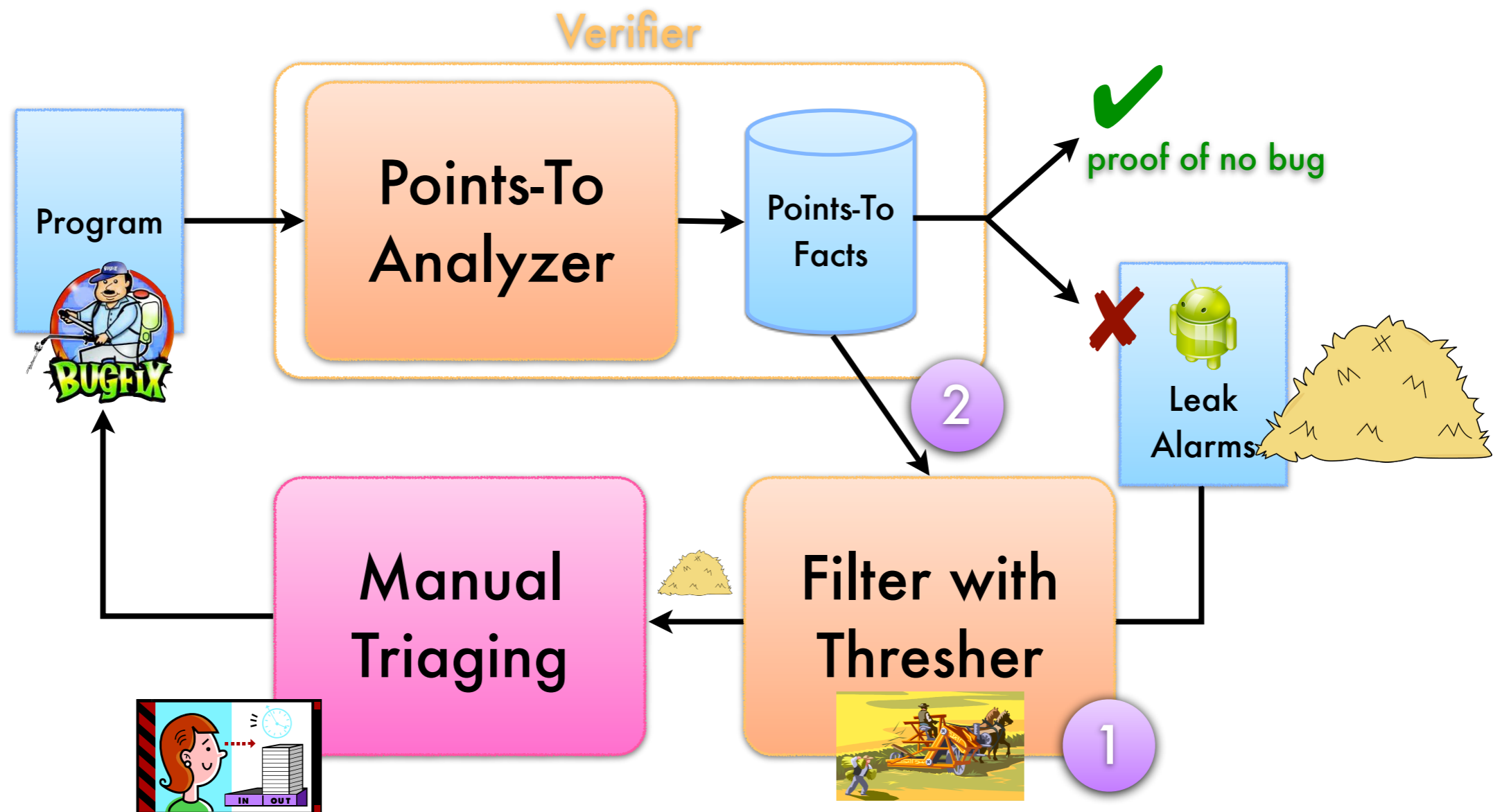
A. Why can the state before statement s satisfy φ ?

B. Because before the previous statement s' , the state could satisfy formula φ'

Specialized constraint forms makes finding refutations **feasible**



Summary: Thresher assists the user with alarm triaging by effectively filtering out many false alarms.



Idea 1: Refute points-to on-demand with **second** “uber-precise” filter analysis

Idea 2: Leverage the **facts** from the first analysis in the filter analysis to scale

Is Thresher effective at filtering?

Thresher analyzes **Java VM** bytecode

7 Android app benchmarks

2,000 to 40,000 source lines of code

+ 880,000 sources lines of Android framework code



Off-the-shelf, state-of-the-art points-to analysis from WALA

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)	False Alarm %	Filtered %
PulsePoint	unknown	16	8	8	95	0	100
StandupTimer	2K	25	15	0	1068	100	60
DroidLife	3K	3	0	3	1	0	-
SMSPopUp	7K	5	1	4	46	0	100
aMetro	20K	54	18	36	18	0	100
K9Mail	40K	208	130	64	374	18	90
Total	72K	311	172	115	1602	17	88

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)	False Alarm %	Filtered %
PulsePoint	unknown	16	8	8	95	0	100
StandupTimer	2K	25	15	0	1068	100	60
DroidLife	3K	3	0	3	1	0	-
SMSPopUp	7K	5	1	4	46	0	100
aMetro	20K	54	18	36	18	0	100
K9Mail	40K	208	130	64	374	18	90
Total	72K	311	172	115	1602	17	88

staticfield-
Activity pairs

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)	False Alarm %	Filtered %
PulsePoint	unknown	16	8	8	95	0	100
StandupTimer	2K	25	15	0	1068	100	60
DroidLife	3K	3	0	3	1	0	-
SMSPopUp	7K	5	1	4	46	0	100
aMetro		54	18	36	18	0	100
K9Mail		208	130	64	374	18	90
Total		311	172	115	1602	17	88

triage "well"
at ~1-2 hours
per alarm

staticfield-
Activity pairs

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted
PulsePoint	unknown	16	8
StandupTimer	2K	25	15
DroidLife	3K	3	0
SMSPopUp	7K	5	1
aMetro	20K	54	18
K9Mail	40K	208	130
Total	72K	311	172

staticfield-
Activity pairs

Filtered



Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs
PulsePoint	unknown	16	8	8
StandupTimer	2K	25	15	0
DroidLife	3K	3	0	3
SMSPopUp	7K	5	1	4
aMetro	20K	54	18	36
K9Mail	40K	208	130	64
Total	72K	311	172	115

staticfield-
Activity pairs

Filtered



Is Thresher effective at filtering?

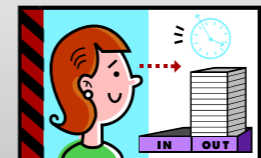
Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs
PulsePoint	unknown	16	8	8
StandupTimer	2K	25	15	0
DroidLife	3K	3	0	3
SMSPopUp	7K	5	1	4
aMetro	20K	54	18	36
K9Mail	40K	208	130	64
Total	72K	311	172	115

staticfield-
Activity pairs

Filtered



Manual



Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs
PulsePoint	unknown	16	8	8
StandupTimer	2K	25	15	0
DroidLife	3K	3	0	3
SMSPopUp	7K	5	1	4
aMetro	20K			36
K9Mail	40K			64
Total	72K			115

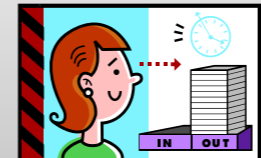
triage "well"
at 10-15
minutes per

staticfield-
Activity pairs

Filtered



Manual



Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs
PulsePoint	unknown	16	8	8
StandupTimer	2K	25	15	0
DroidLife	3K	3	0	3
SMSPopUp	7K	5	1	4
aMetro	20K	54	18	36
K9Mail	40K	208	130	64
Total	72K	311	172	115

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)
PulsePoint	unknown	16	8	8	95
StandupTimer	2K	25	15	0	1068
DroidLife	3K	3	0	3	1
SMSPopUp	7K	5	1	4	46
aMetro	20K	54	18	36	18
K9Mail	40K	208	130	64	374
Total	72K	311	172	115	1602

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)
PulsePoint	unknown	16	8	8	95
StandupTimer	2K	25	15	0	1068
DroidLife	3K	3	0	3	1
SMSPopUp	7K	5	1	4	46
aMetro	20K	54	18	36	18
K9Mail	40K	208	130	64	374
Total	72K	311	172	115	1602

< ~coffee to lunch break

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)
PulsePoint	unknown	16	8	8	95
StandupTimer	2K	25	15	0	1068
DroidLife	3K	3	0	3	1
SMSPopUp	7K	5	1	4	46
aMetro	20K	54	18	36	18
K9Mail	40K	208	130	64	374
Total	72K	311	172	115	1602

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)	False Alarm %
PulsePoint	unknown	16	8	8	95	0
StandupTimer	2K	25	15	0	1068	100
DroidLife	3K	3	0	3	1	0
SMSPopUp	7K	5	1	4	46	0
aMetro	20K	54	18	36	18	0
K9Mail	40K	208	130	64	374	18
Total	72K	311	172	115	1602	17

% after filtering

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)	False Alarm %	Filtered %
PulsePoint	unknown	16	8	8	95	0	100
StandupTimer	2K	25	15	0	1068	100	60
DroidLife	3K	3	0	3	1	0	-
SMSPopUp	7K	5	1	4	46	0	100
aMetro	20K	54	18	36	18	0	100
K9Mail	40K	208	130	64	374	18	90
Total	72K	311	172	115	1602	17	88

% after filtering

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs	Thresher Time (s)	False Alarm %	Filtered %
PulsePoint	unknown	16	8	8	95	0	100
StandupTimer	2K	25	15	0	1068	100	60
DroidLife	3K	3	0	3	1	0	-
SMSPopUp	7K	5	1	4	46	0	100
aMetro	20K	54	18	36	18	0	100
K9Mail	40K	208	130	64	374	18	90
Total	72K	311	172	115	1602	17	88

False alarms down to **17%** from **63%** (points-to analysis only)

Thresher filters **88%** of false alarms from points-to analysis

Guesstimate

Triage "well" without versus with: ~450 hours versus ~30 hours

Triage "ok" without: ~30 hours

		Alerts	Rejected	Bugs	Time (s)	Alert %	ed %
PulsePoint	unknown	16	8	8	95	0	100
StandupTimer	2K	25	15	0	1068	100	60
DroidLife	3K	3	0	3	1	0	-
SMSPopUp	7K	5	1	4	46	0	100
aMetro	20K	54	18	36	18	0	100
K9Mail	40K	208	130	64	374	18	90
Total	72K	311	172	115	1602	17	88

False alarms down to 17% from 63% (points-to analysis only)

Thresher filters 88% of false alarms from points-to analysis

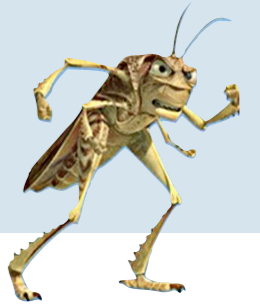
Android
OS





... in the process of finding leaks in apps

Find the Android's HashMap bug ...

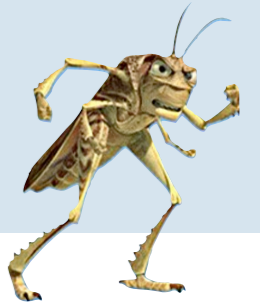


```
class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}
```

Find the Android's HashMap bug ...

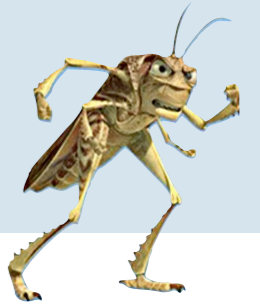


```
class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}
```

Find the Android's HashMap bug ...



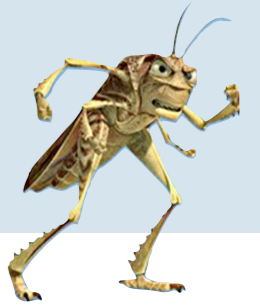
null object pattern: should not be written to

```
class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}
```

Find the Android's HashMap bug ...



null object pattern: should not be written to

```
class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}
```

allocate new
backing array
on first write

Find the Android's HashMap bug ...

null object pattern: should not be written to

```
class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}
```

allocate new
backing array
on first write



Find the Android's HashMap bug ...

null object pattern: should not be written to

```
class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}
```

allocate new
backing array
on first write



An "evil" implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.

Find the Android's HashMap bug ...

null object pattern: should not be written to

```
class HashMap {  
    static Object[] EMPTY = new Object[2]; ...  
    HashMap() { this.tbl = EMPTY; capacity initially empty }  
  
    void put(Object key, Object val) {  
        if (need capacity) {  
            this.tbl = new Object[more capacity];  
            copy from old table  
        }  
        this.tbl[bucket using hash of key] = val;  
    }  
}
```

allocate new
backing array
on first write



```
HashMap(Map m) {  
    if (m.size() < 1) { this.tbl = EMPTY; }  
    else { this.tbl = new Object[at least m.size()]; }  
    copy from m  
}  
return "evil" content
```

return 0

An "evil" implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.

Find the Android's HashMap bug ...

What if you store
passwords in a HashMap?

```
class  
s  
H  
v  
}  
HashMap(Map m) {  
    if (m.size() < 1) { this.tbl = EMPTY; }  
    else { this.tbl = new Object[at least m.size()]; }  
    copy from m  
}  
return "evil" content  
}
```

return 0

this.tbl = EMPTY;

this.tbl = new Object[at least m.size()];

copy from m

return "evil" content

An "evil" implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.

Find the Android's HashMap bug ...

What if you store passwords in a HashMap?

We reported this, Google fixed it

<https://android-review.googlesource.com/#/c/52183/>



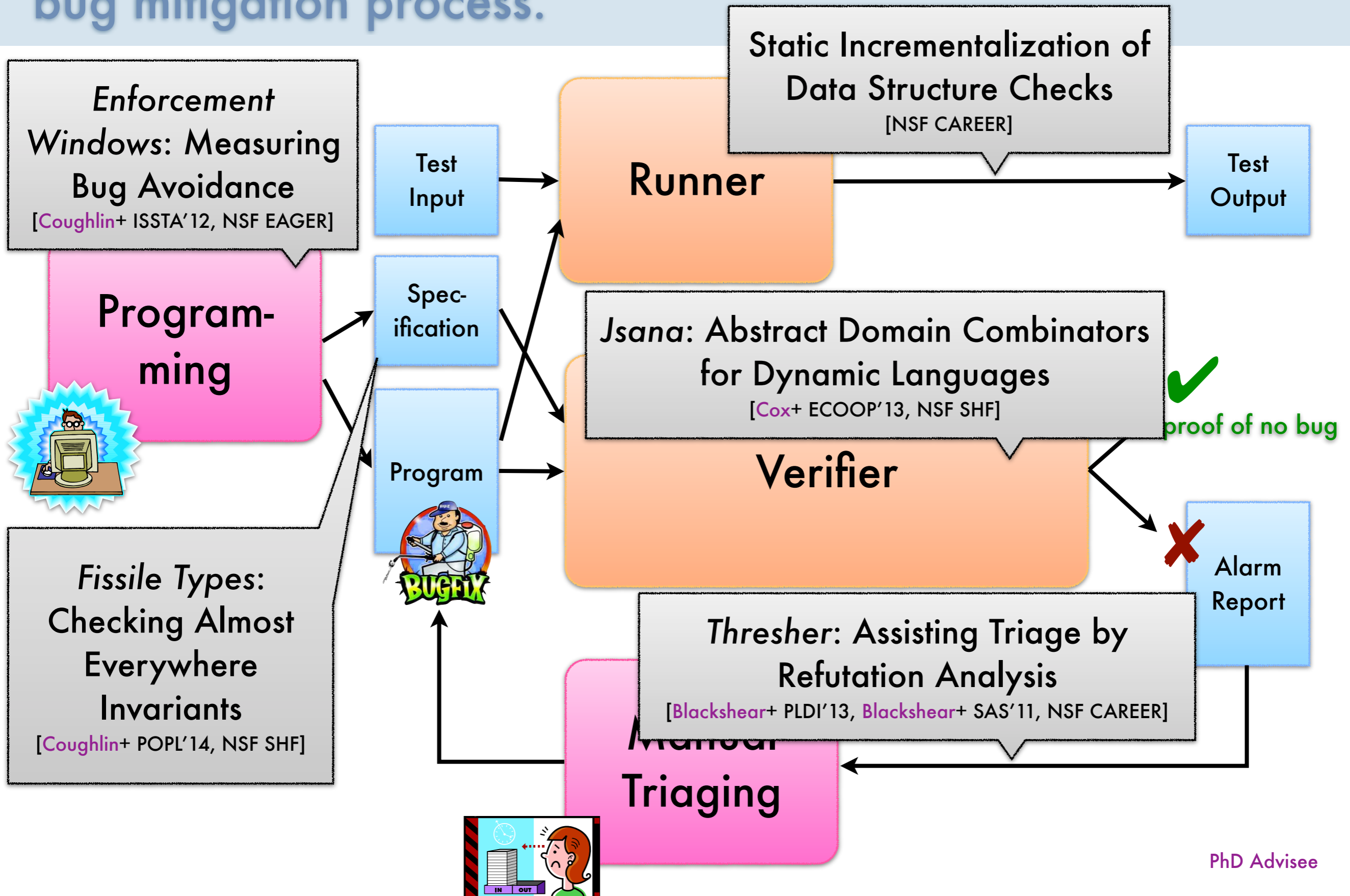
```
HashMap(Map m) {  
    if (m.size() < 1) { this.tbl = EMPTY; }  
    else { this.tbl = new Object[at least m.size()]; }  
    copy from m  
}
```

return "evil" content

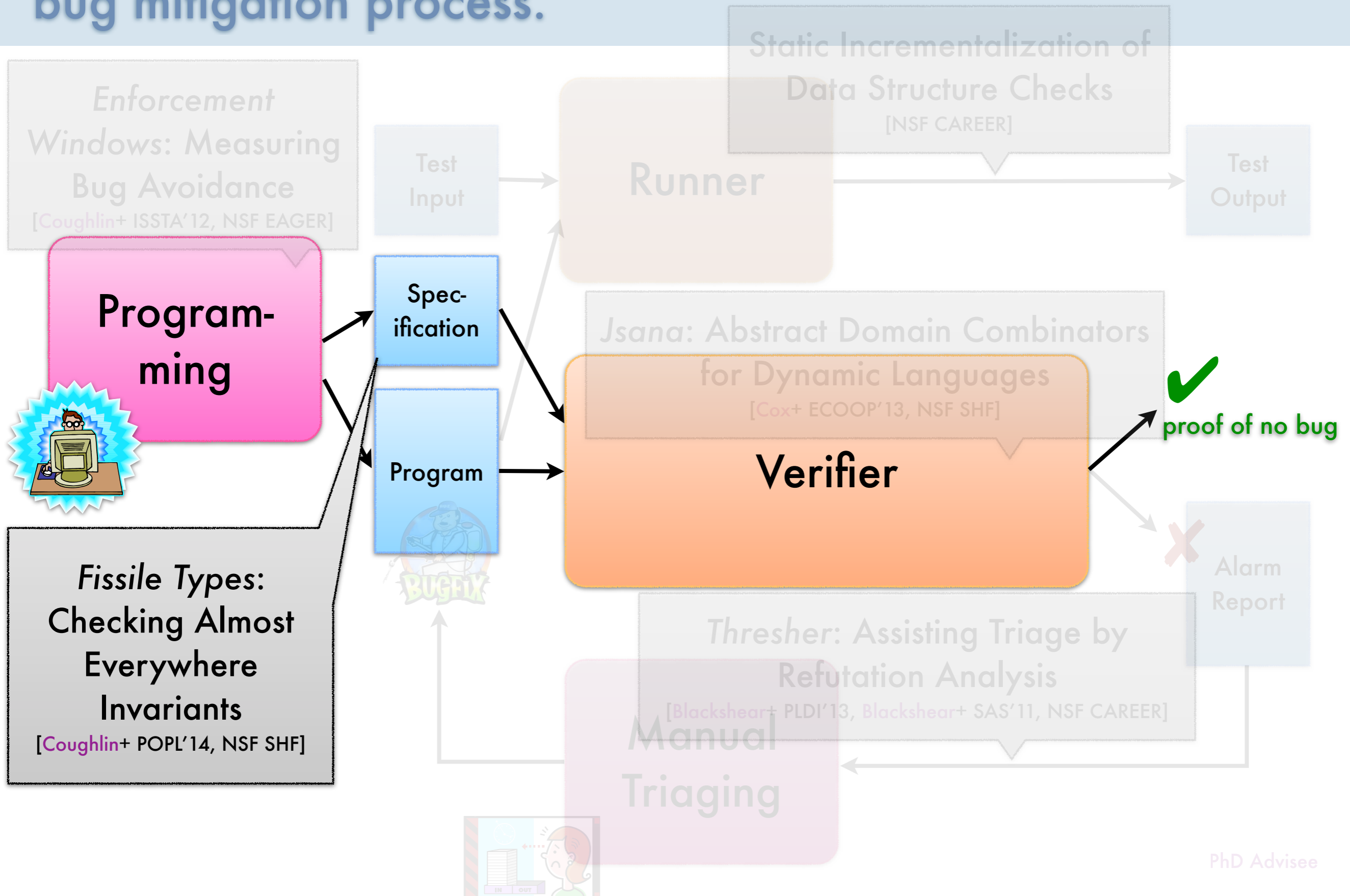
An "evil" implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.

Contribution: Addressed the
false alarm problem with
a "smart and precise filter"
a refutation analysis

Agenda: The cooperative approach addresses the whole bug mitigation process.



Agenda: The cooperative approach addresses the whole bug mitigation process.



**Fissile Types:
Checking Reflection
with Almost
Everywhere
Invariants**

Method Reflection and the Great Divide

Method Reflection and the Great Divide

```
object [string] ()
```

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

`object [string] ()`

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

`object [string] ()`

type system designers



"web 2.0" developers



Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

`object [string] ()`

type system designers



"web 2.0" developers



Type system designers **worry**.

What gets called? What if
object has **no method** named
by string?

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

`object [string] ()`

type system designers



Type system designers **worry**.

What gets called? What if object has **no method** named by string?

"web 2.0" developers



"Web 2.0" developers think it's **cool**.

I can flexible and compact code, so I will take it **over static safety**.

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

`object [string] ()`

type system designers



"web 2.0" developers



Type system designers **worry**.

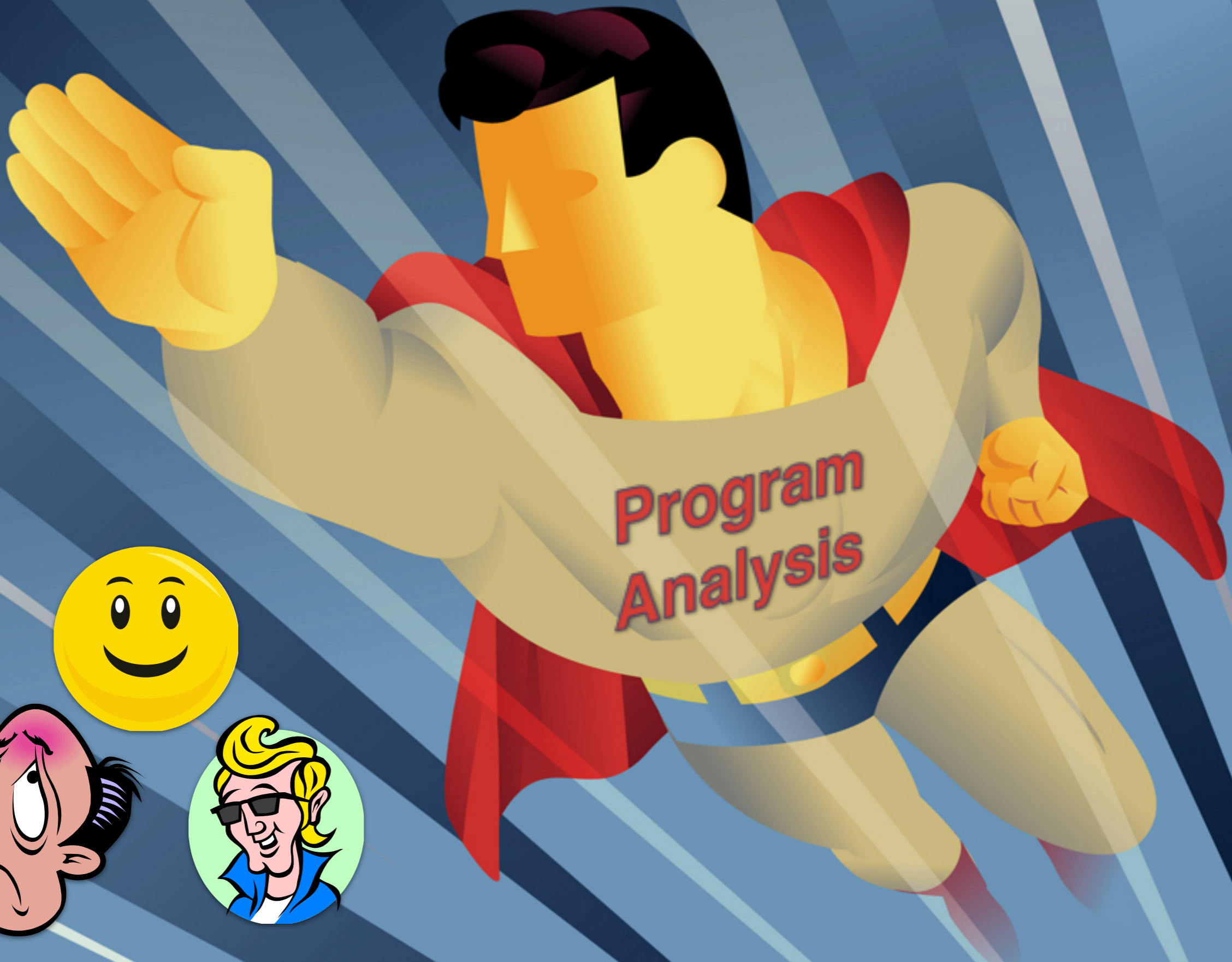
"Web 2.0" developers think it's **cool**.

When
obj
by

"MethodNotFound" checked at run time

static safety.





**Program
Analysis**



Programs are often

(1) safe, (2) not type safe, (3) but almost so

Programs are often

(1) safe, (2) not type safe, (3) but almost so



Program

Programs are often

(1) safe, (2) not type safe, (3) but almost so

```
callback.o [callback.m] ()
```

Program

Programs are often

(1) safe, (2) not type safe, (3) but almost so

safe assuming a relationship
invariant between `.o` and `.m`

```
callback.o[callback.m]()
```

Program

Programs are often

(1) safe, (2) not type safe, (3) but almost so

invariant holds

safe assuming a relationship
invariant between `.o` and `.m`

```
callback.o[callback.m]()
```

Program

Programs are often

(1) safe, (2) not type safe, (3) but almost so

invariant holds

safe assuming a relationship
invariant between `.o` and `.m`

```
callback.o[callback.m]()
```

invariant broken



Program

Programs are often

(1) safe, (2) not type safe, (3) but almost so

invariant holds

safe assuming a relationship
invariant between .o and .m

```
callback.o[callback.m]()
```

invariant broken



but only temporarily



Program

Programs are often

(1) safe, (2) not type safe, (3) but almost so

invariant holds

safe assuming a relationship
invariant between .o and .m

callback.o [callback.m] ()

invariant broken



but only temporarily



Program

Tolerate "temporary" violation with



Safe but not type safe ...

Safe but not type safe ...

```
class Callback:
    var sel: Str
    var obj: Obj

    def call():
        this.obj[this.sel]()

    def update(s: Str, o: Obj | respondsTo s):
        this.sel = s
        this.obj = o
```

Safe but not type safe ...

```
class Callback:
    var sel: Str
    var obj: Obj | respondsTo sel

    def call():
        this.obj[this.sel]()

    def update(s: Str, o: Obj | respondsTo s):
        this.sel = s
        this.obj = o
```

Safe but not type safe ...

```
class Callback:
```

```
  var sel: Str
```

```
  var obj: Obj | respondsTo sel
```

Type specifies a global
relationship invariant

```
  def call():
```

```
    this.obj[this.sel]()
```

```
  def update(s: Str, o: Obj | respondsTo s):
```

```
    this.sel = s
```

```
    this.obj = o
```

Safe but not type safe ...

```
class Callback:
```

```
  var sel: Str
```

```
  var obj: Obj | respondsTo sel
```

Type specifies a global
relationship invariant

```
  def call():
```

```
    this.obj[this.sel]()
```

Call is safe because
of the invariant

```
  def update(s: Str, o: Obj | respondsTo s):
```

```
    this.sel = s
```

```
    this.obj = o
```

Safe but not type safe ...

```
class Callback:
```

```
  var sel: Str
```

```
  var obj: Obj | respondsTo sel
```

Type specifies a global
relationship invariant

```
  def call():
```

```
    this.obj[this.sel]()
```

Call is safe because
of the invariant

```
  def update(s: Str, o: Obj | respondsTo s):
```

```
    this.sel = s
```

```
    this.obj = o
```


Safe but not type safe ...

```
class Callback:
```

```
  var sel: Str
```

```
  var obj: Obj | respondsTo sel
```

Type specifies a global
relationship invariant

```
  def call():
```

```
    this.obj[this.sel]()
```

Call is safe because
of the invariant

```
  def update(s: Str, o: Obj | respondsTo s):
```

```
    this.sel = s
```

```
    this.obj = o
```

relationship invariant violated

Safe but not type safe ...

```
class Callback:
```

```
  var sel: Str
```

```
  var obj: Obj | respondsTo sel
```

Type specifies a global
relationship invariant

```
  def call():
```

```
    this.obj[this.sel]()
```

Call is safe because
of the invariant

```
  def update(s: Str, o: Obj | respondsTo s):
```

```
    this.sel = s
```

```
    this.obj = o
```

relationship invariant violated

relationship invariant restored

Safe but not type safe ...

```
class Callback:
```

```
  var sel: Str
```

```
  var obj: Obj | respondsTo sel
```

Type specifies a global
relationship invariant

```
  def call():
```

```
    this.obj[this.sel]()
```

Call is safe because
of the invariant

```
  def update(s: Str, o: Obj | respondsTo s):
```

```
    this.sel = s
```

```
    this.obj = o
```

relationship invariant violated

relationship invariant restored

Tolerate “temporary” violation with



Is Fissile effective at proving reflective call safety?

Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

Type annotations

seeded with 76 `respondsTo` in system libraries

needed only 136 annotations in benchmarks (total)



Is Fissile effective at proving reflective call safety?

Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

Type annotations

Proved **86%** of check sites (up from 76%) at
interactive speeds (~4 to 90 kloc/s)

benchmarks (total)



Is Fissile effective at proving reflective call safety?

Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

places requiring a check of the invariant

Proved **86%** of check sites (up from 76%) at
interactive speeds (~4 to 90 kloc/s)

benchmarks (total)



Is Fissile effective at proving reflective call safety?

Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

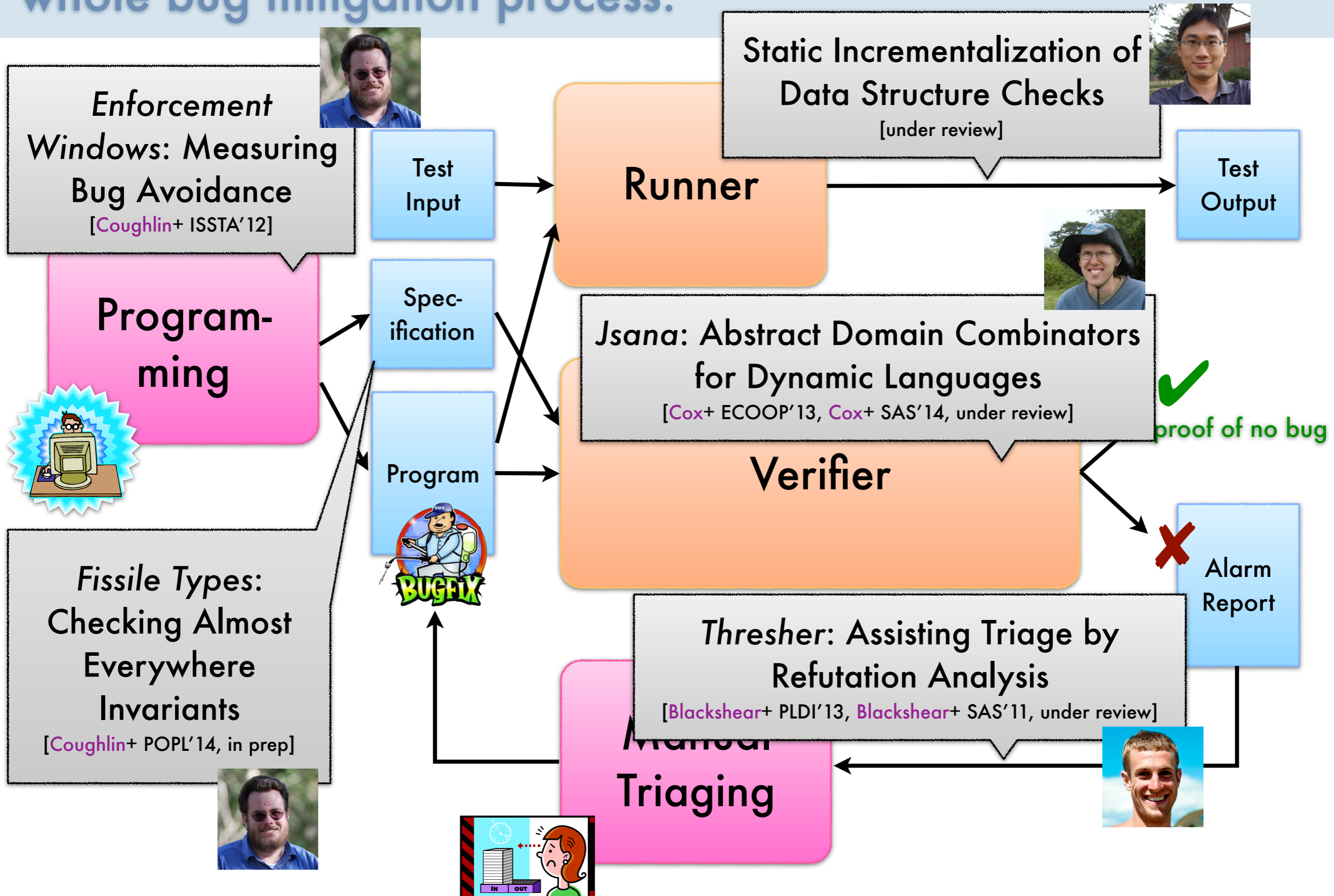
places requiring a check of the invariant

Proved **86%** of check sites (up from 76%) at
interactive speeds (~4 to 90 kloc/s)

Big Deal: makes IDE integration possible



Summary: The cooperative approach addresses the whole bug mitigation process.





www.cs.colorado.edu/~bec
pl.cs.colorado.edu



www.cs.colorado.edu/~bec
pl.cs.colorado.edu