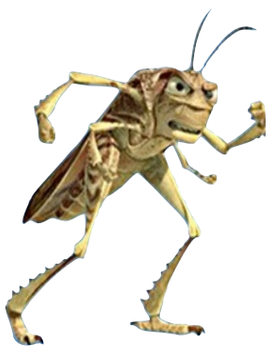



Cooperative Program Analysis for Reliable Mobile Applications

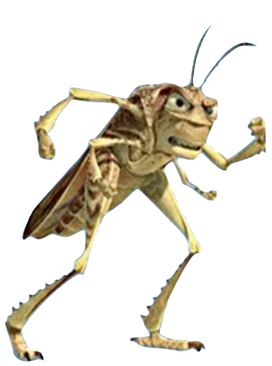
Bor-Yuh Evan Chang
University of Colorado Boulder



Huawei Vision Forum
October 21, 2015

PLV

A program analysis story ...




A380

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

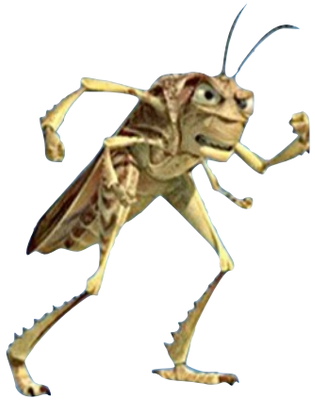
mobile

cloud

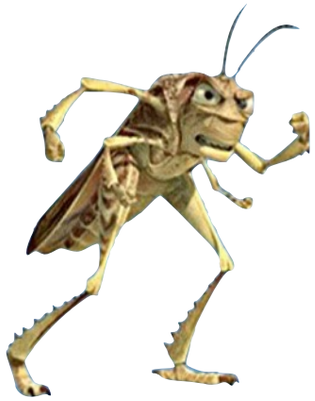
cyberphysical

Software is everywhere and varying more and more

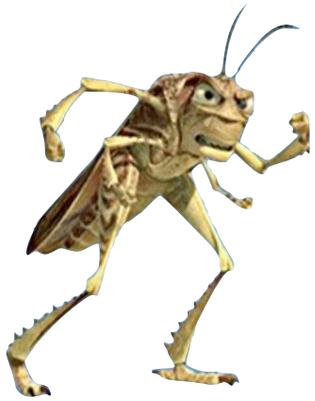
Software is getting more and more complex




HELP!

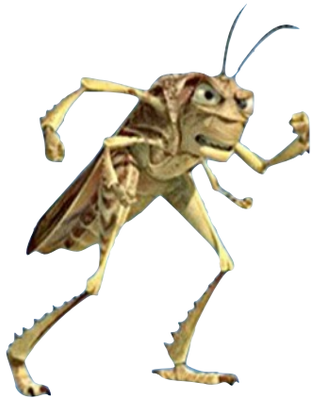


1980s Bug in Therac-25 kills 6



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

How does program analysis save the day?

Program Analysis for Formal Verification

Program

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

How does program analysis save the day?

Program Analysis for Formal Verification

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

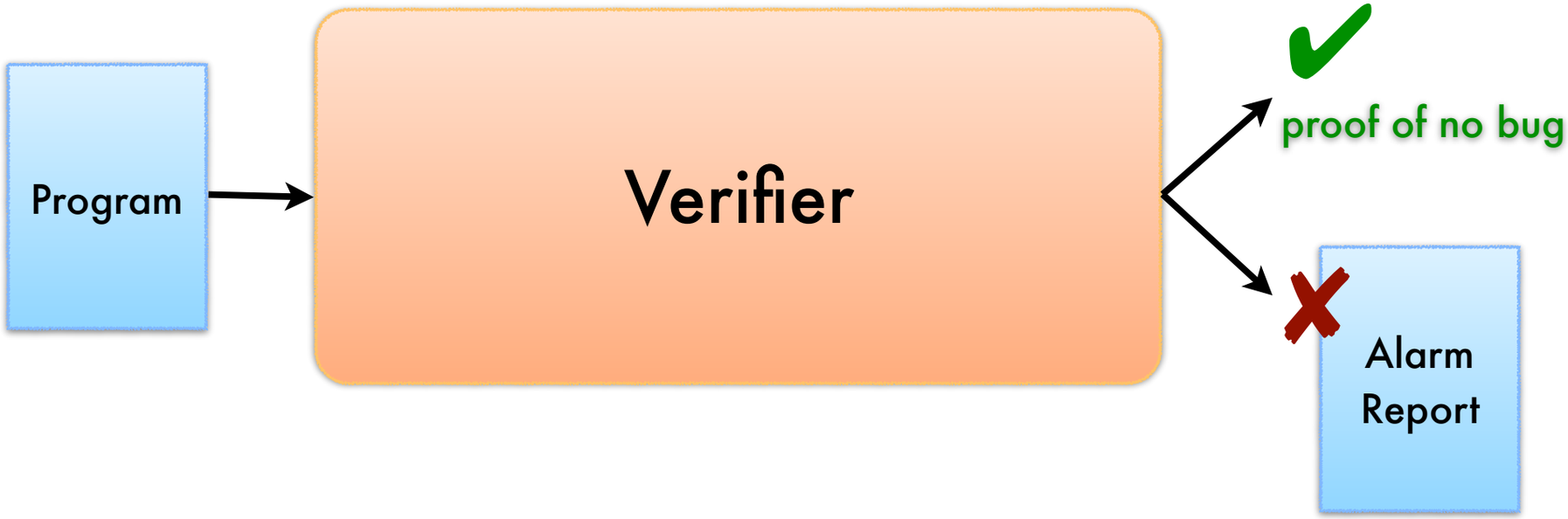
How does program analysis save the day?

Program Analysis for Formal Verification

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

How does program analysis save the day?

Program Analysis for Formal Verification

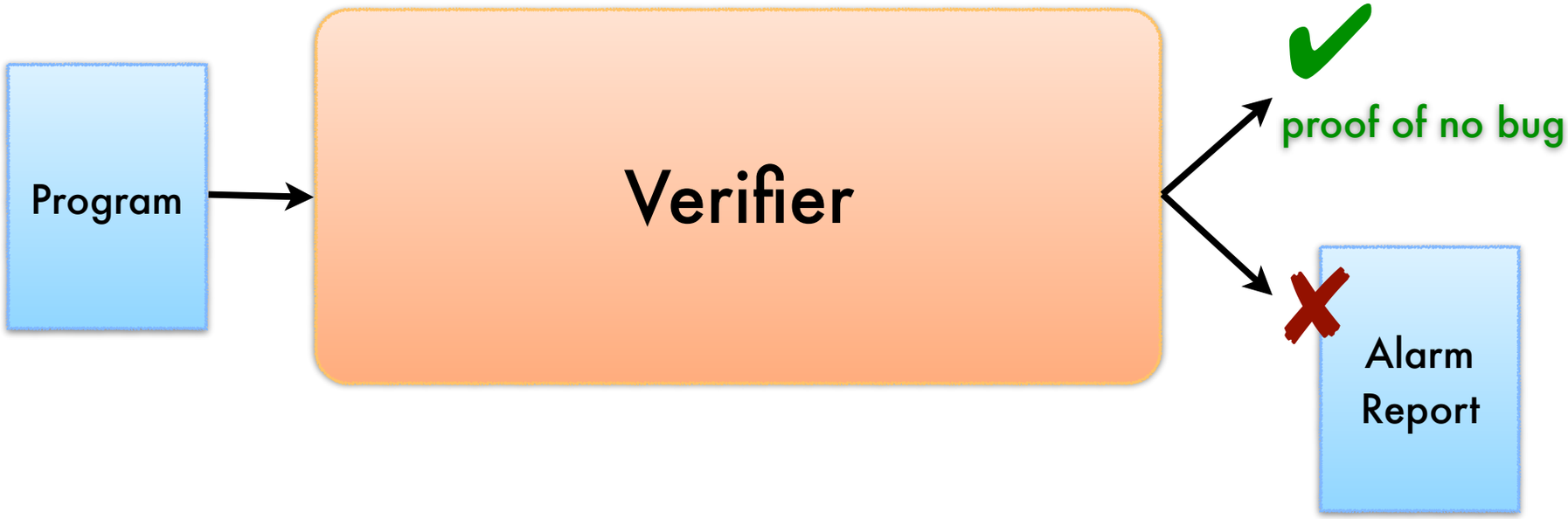


Systematically examine the program to "simulate" running it on "all inputs"

The End?

The Ugly, Hidden Truth

Program Analysis for Formal Verification



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

The Ugly, Hidden Truth

Program Analysis for Formal Verification

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

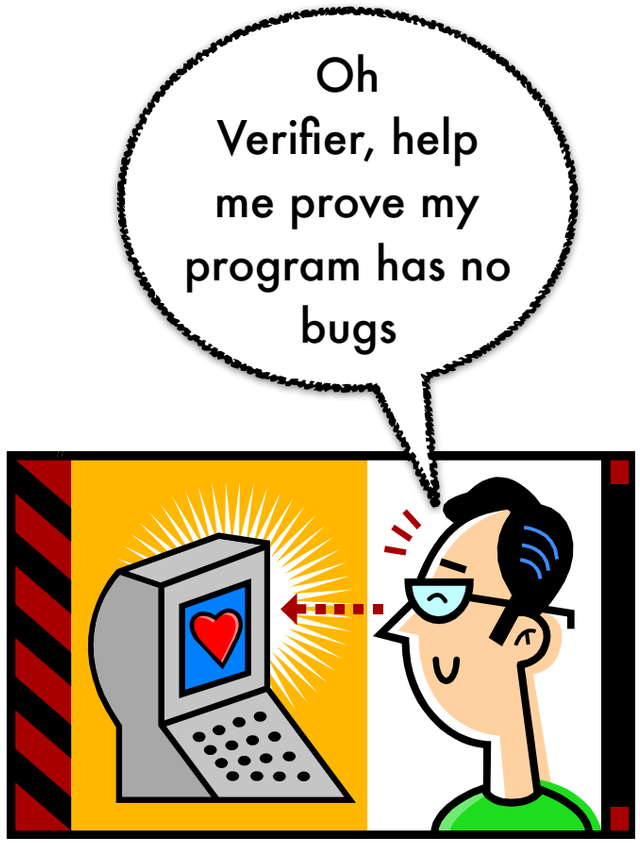
The Ugly, Hidden Truth

Program Analysis for Formal Verification

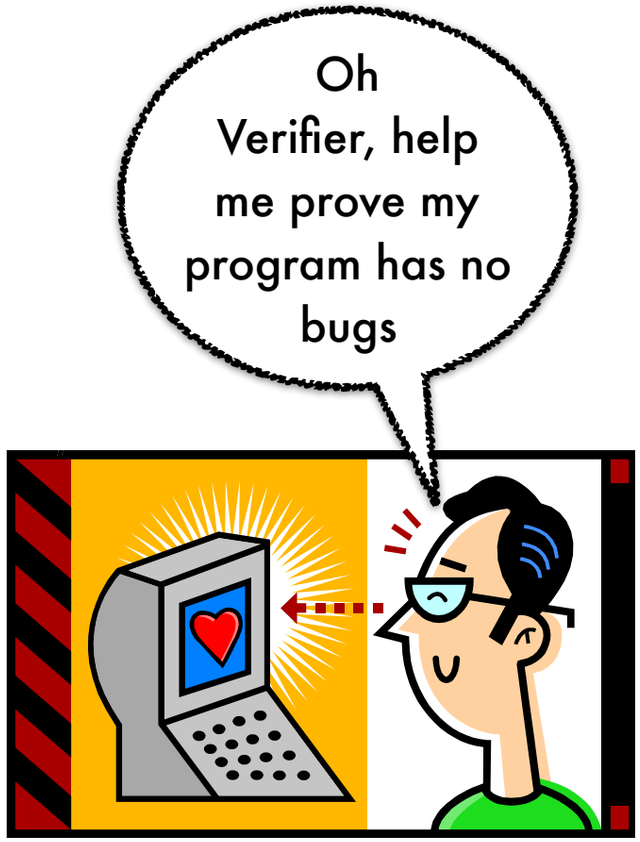
Undecidability necessitates the possibility of false alarms. We hope not too many.

Uncooperative Program Analysis?

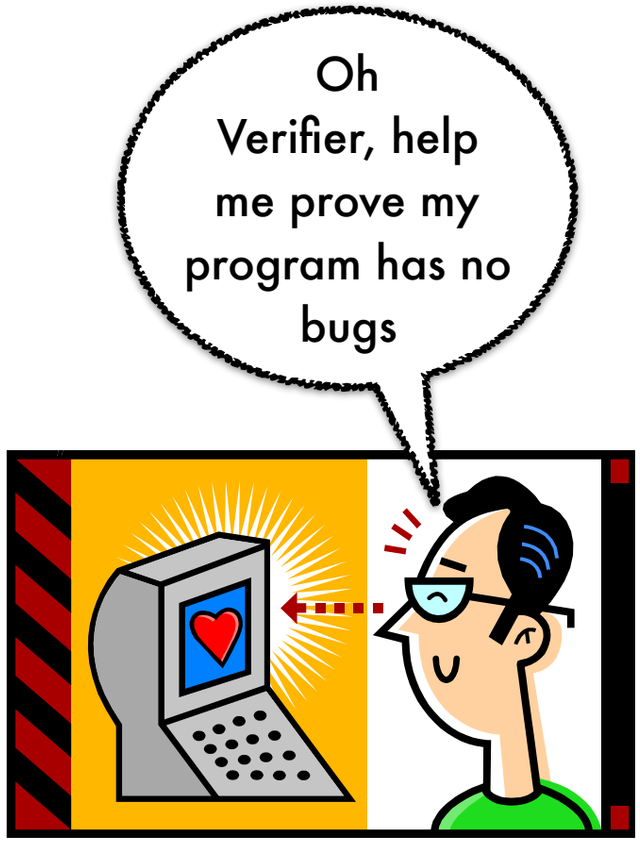
Uncooperative Program Analysis?



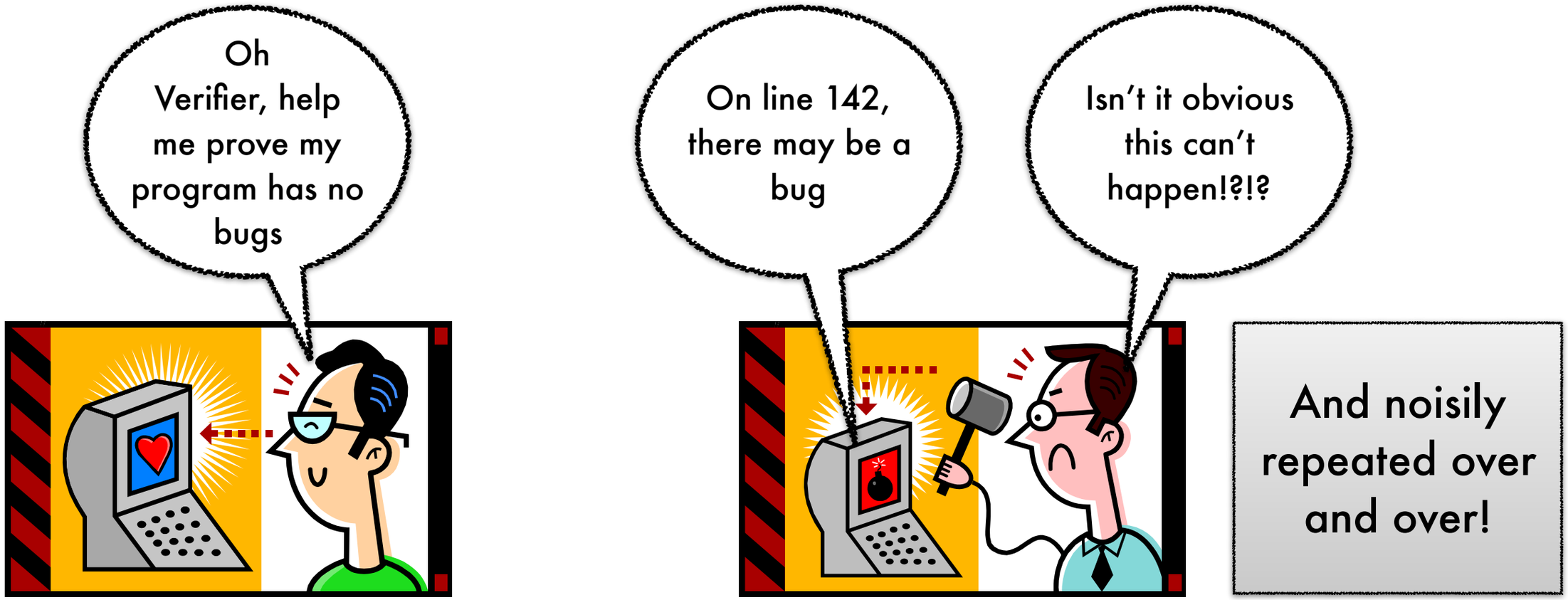
Uncooperative Program Analysis?



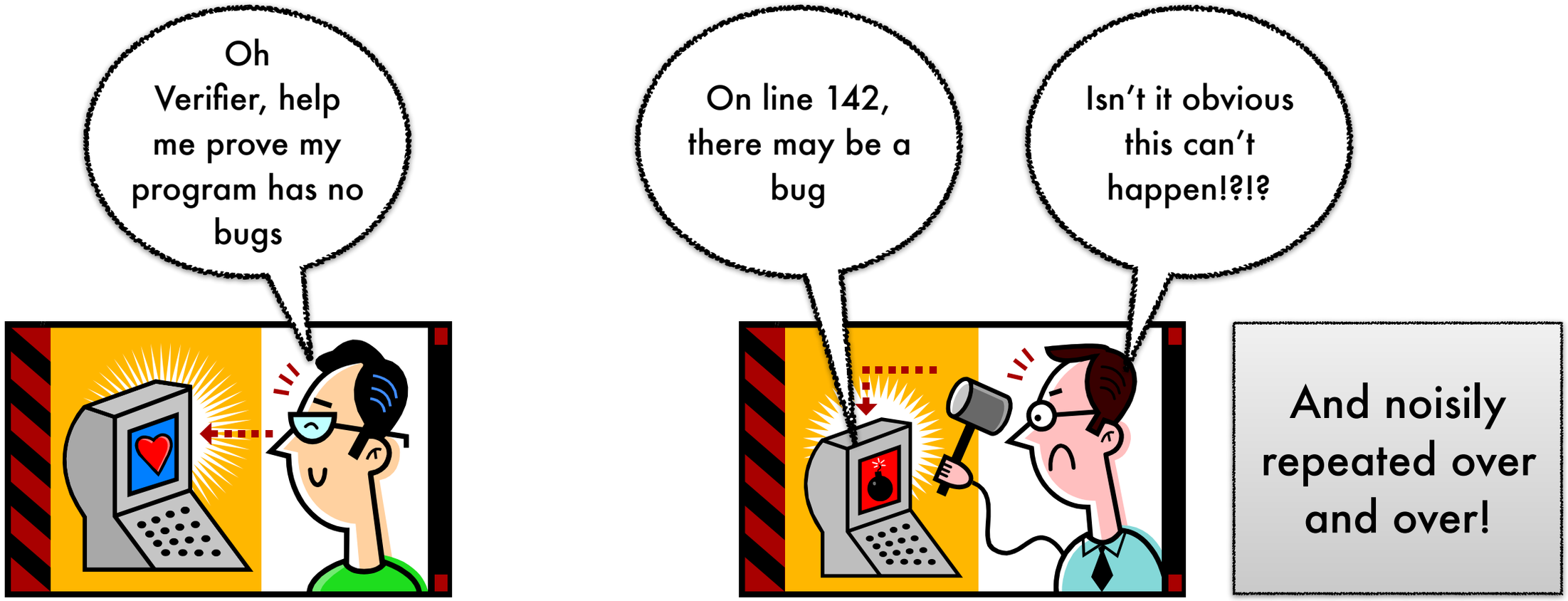
Uncooperative Program Analysis?



Uncooperative Program Analysis?

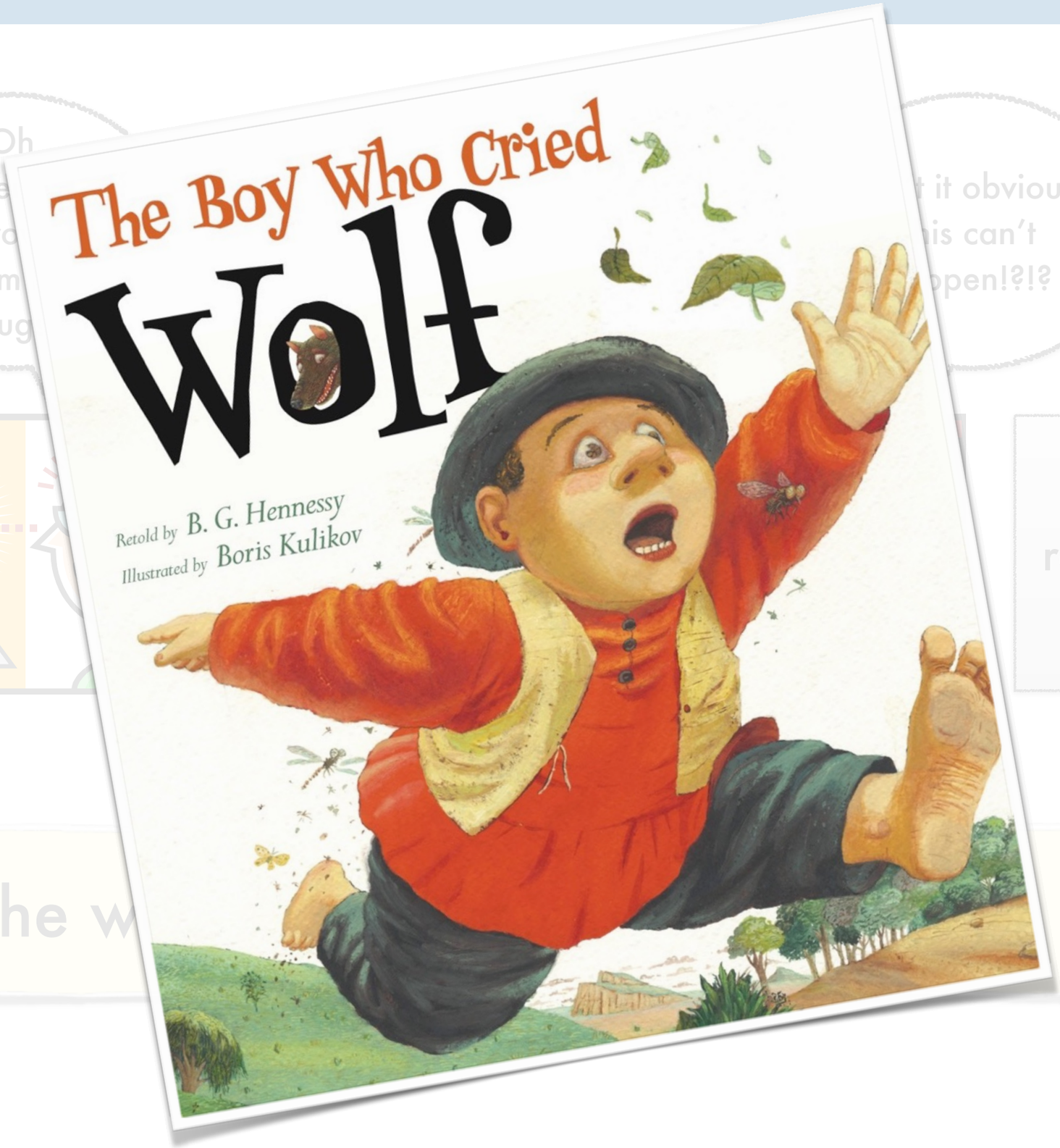


Uncooperative Program Analysis?



The well-known false alarm problem

Uncooperative Program Analysis?



Oh
Verifie
me pro
program
bug

t it obvious
is 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."

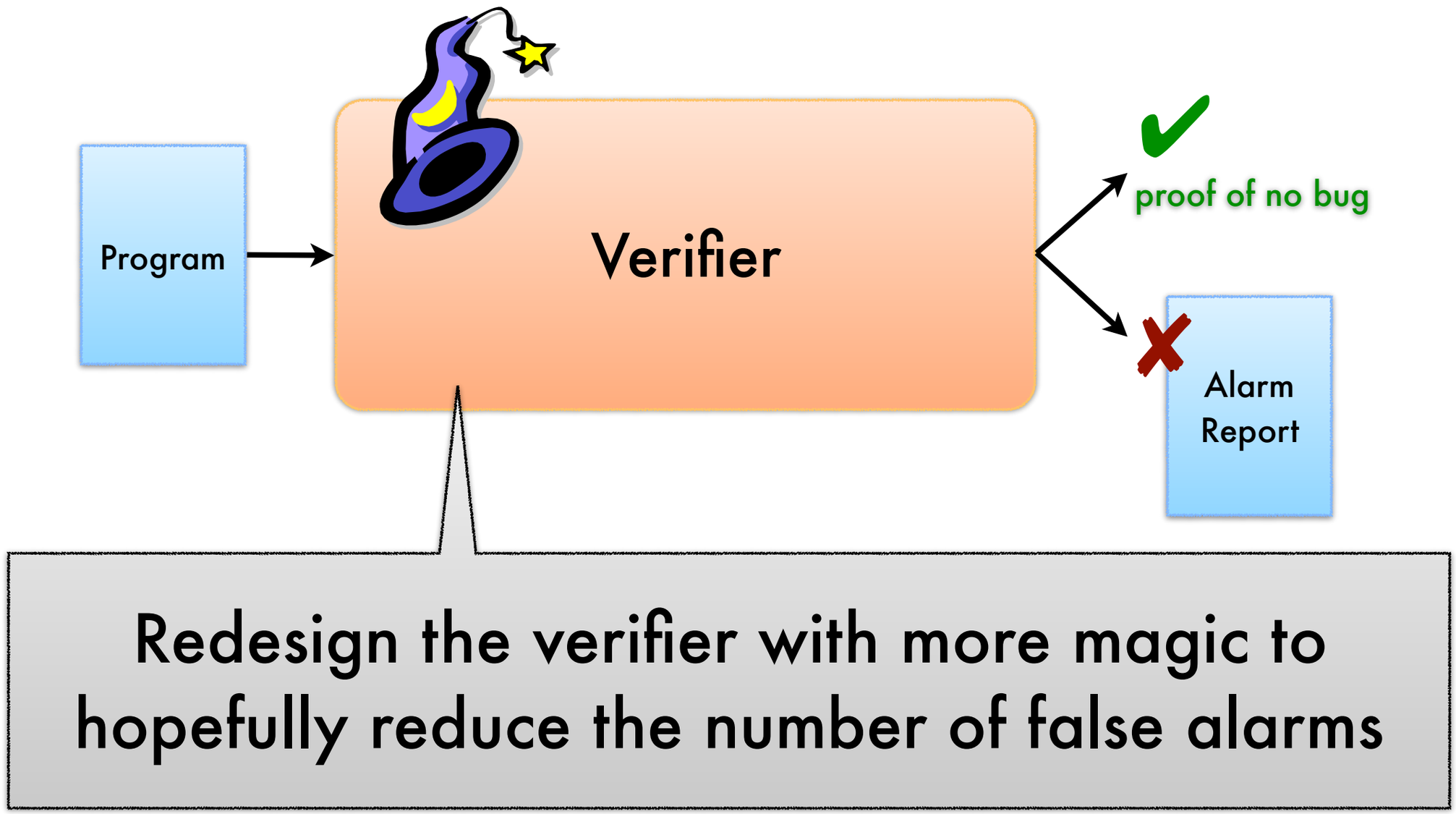
"A stupid false [alarm] implies the tool is stupid."

Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S. and Engler, D. 2010. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM*. 53, 2 (2010), 66–75.

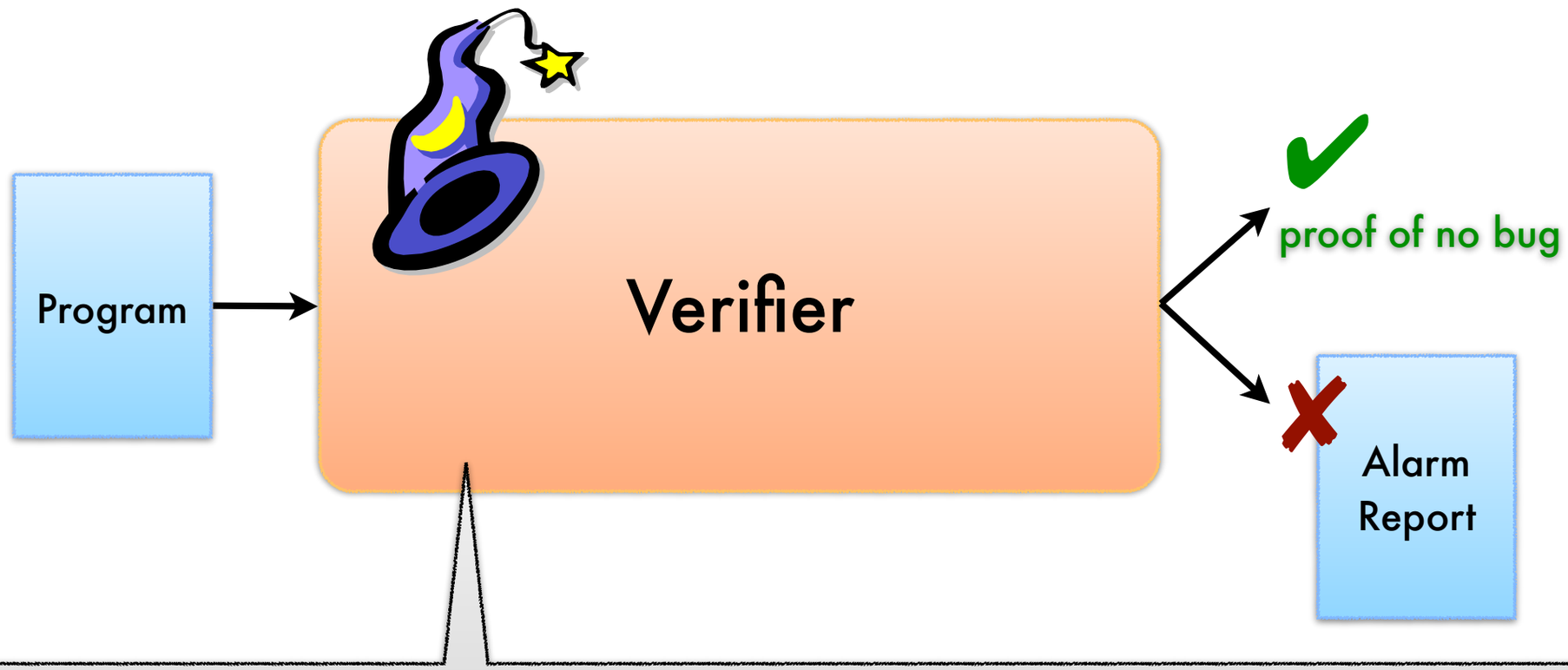
coverity®

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.



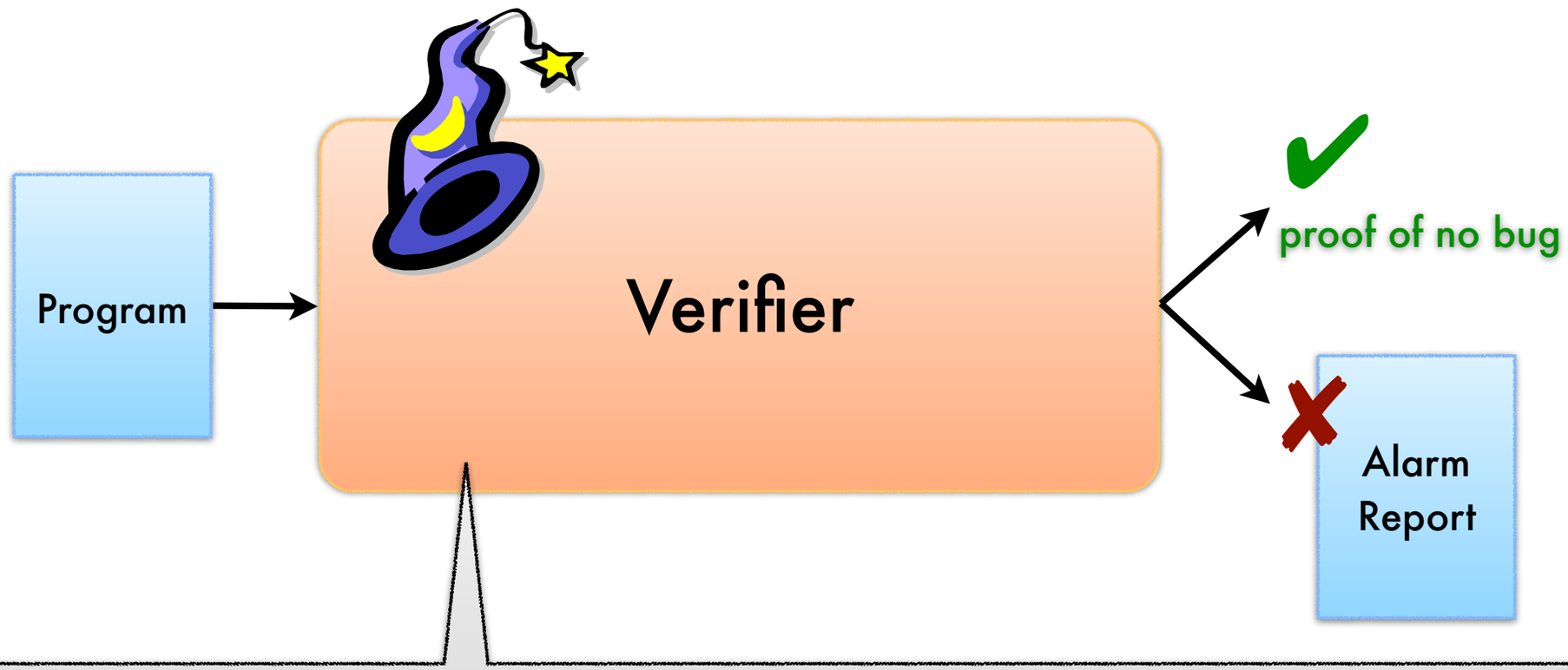
The traditional approach to the false alarm problem focuses on improving the verifier.



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

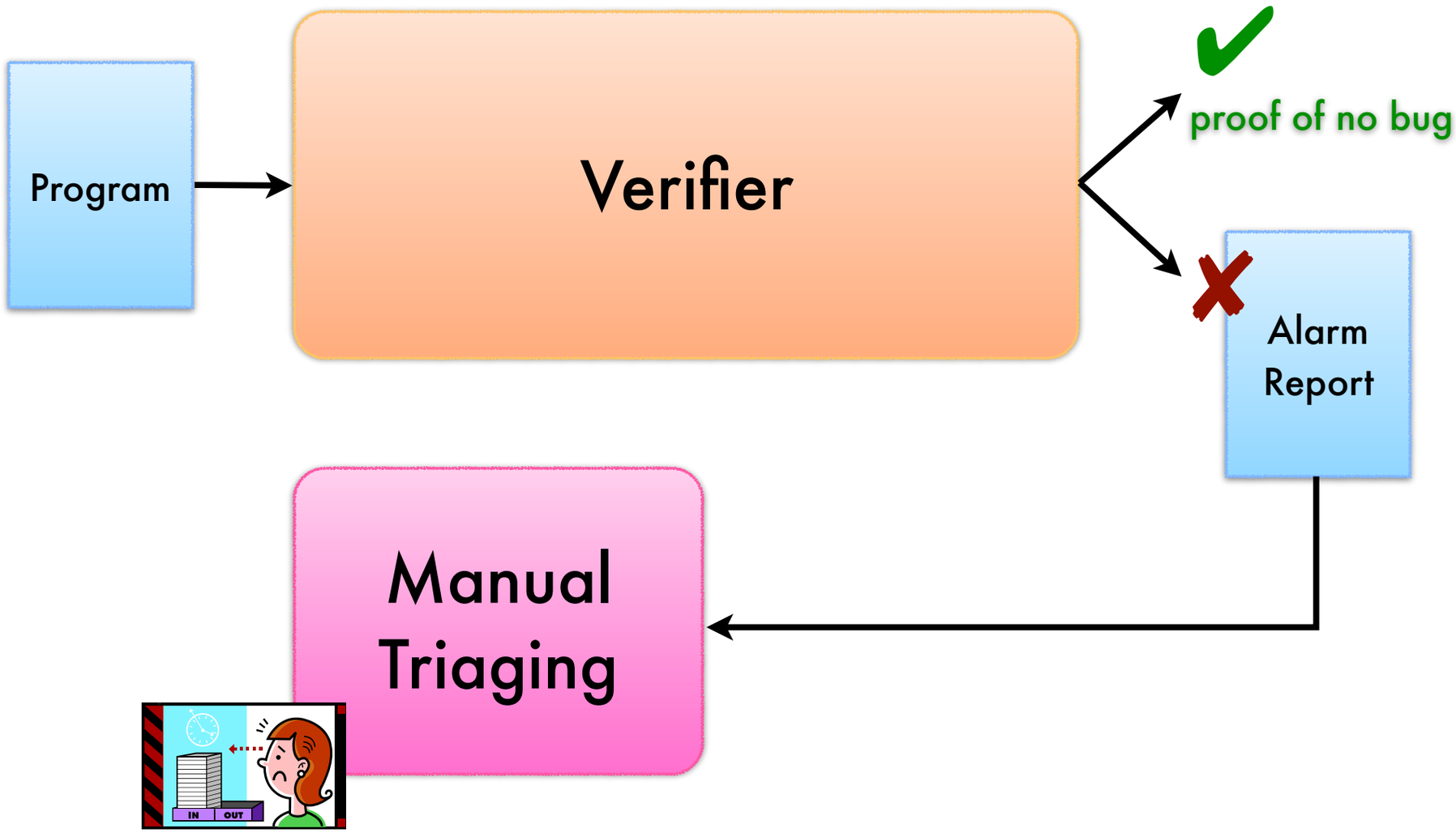
But it can never be perfect (undecidability)

Also not a sufficient "excuse"

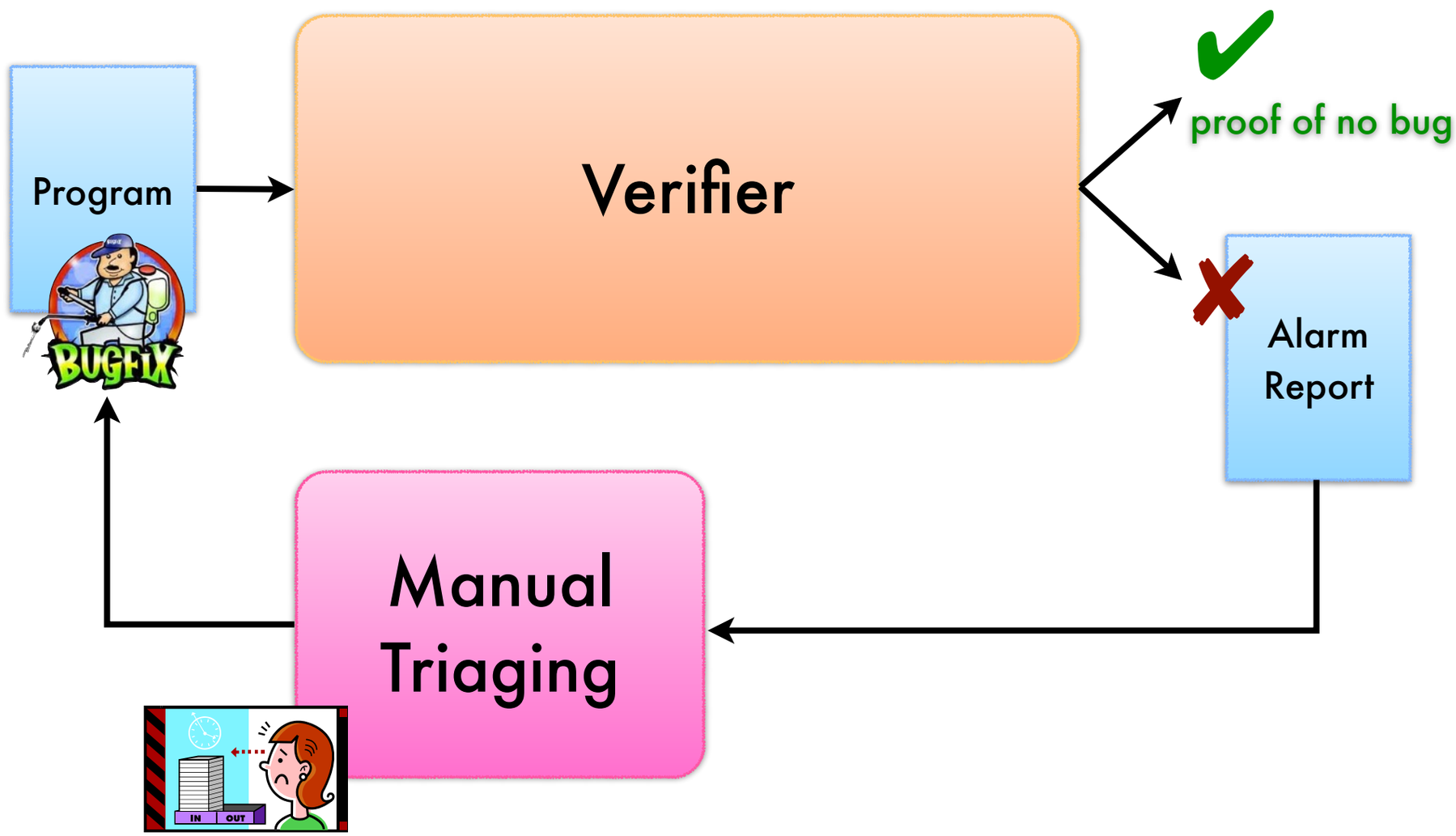
Research Agenda: The cooperative approach addresses the whole bug mitigation process.

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

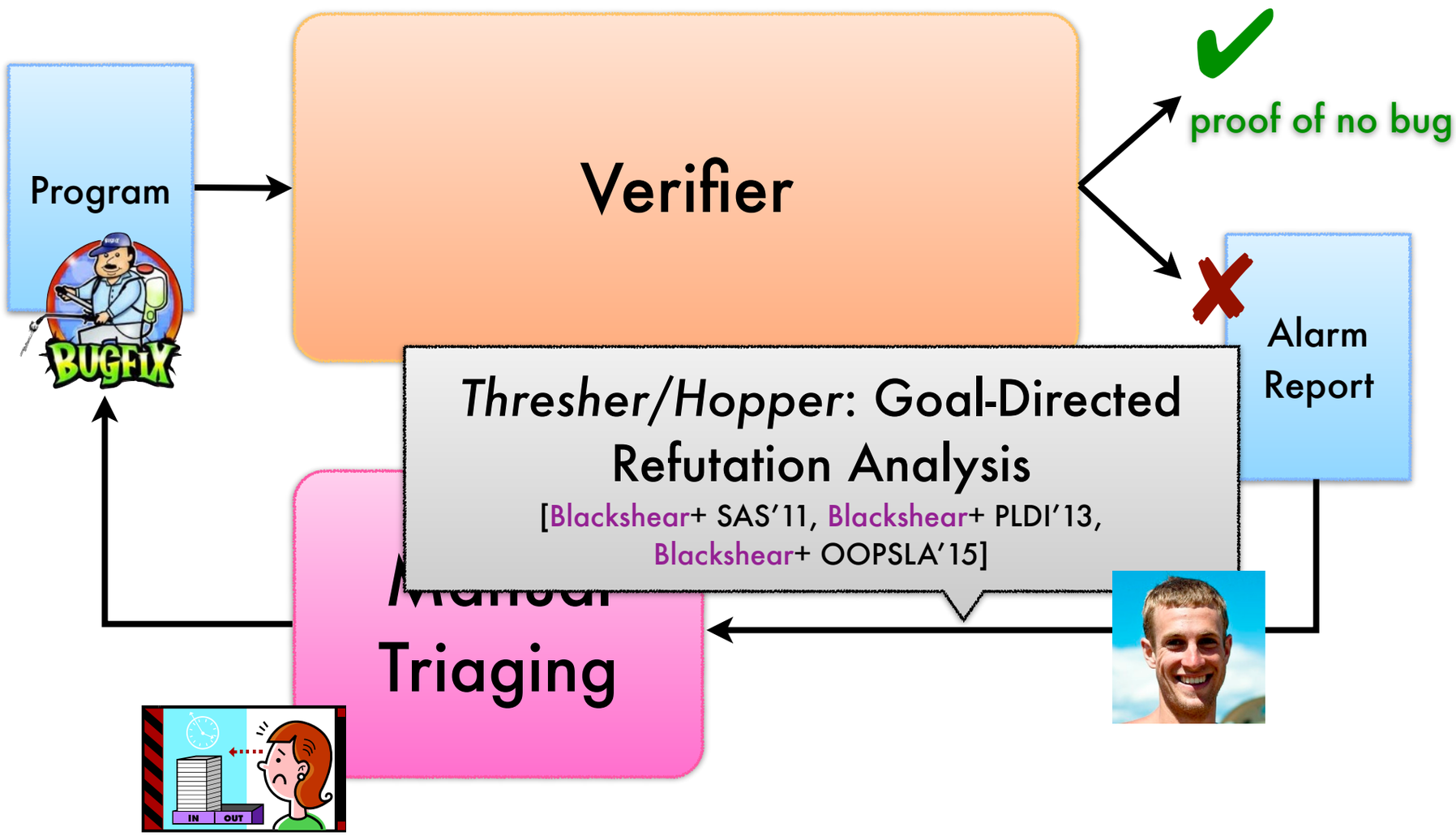
Research Agenda: The cooperative approach addresses the whole bug mitigation process.



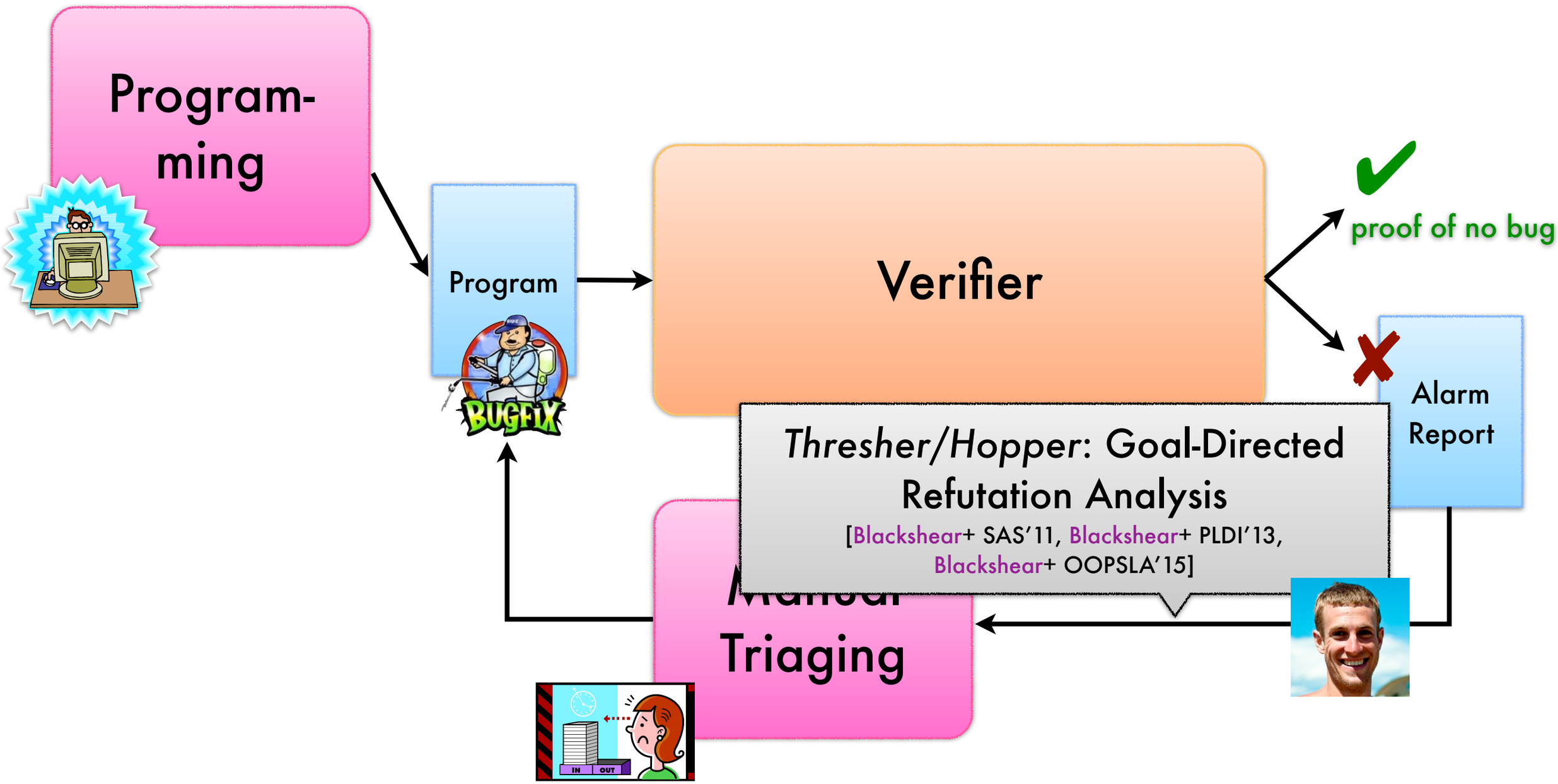
Research Agenda: The cooperative approach addresses the whole bug mitigation process.



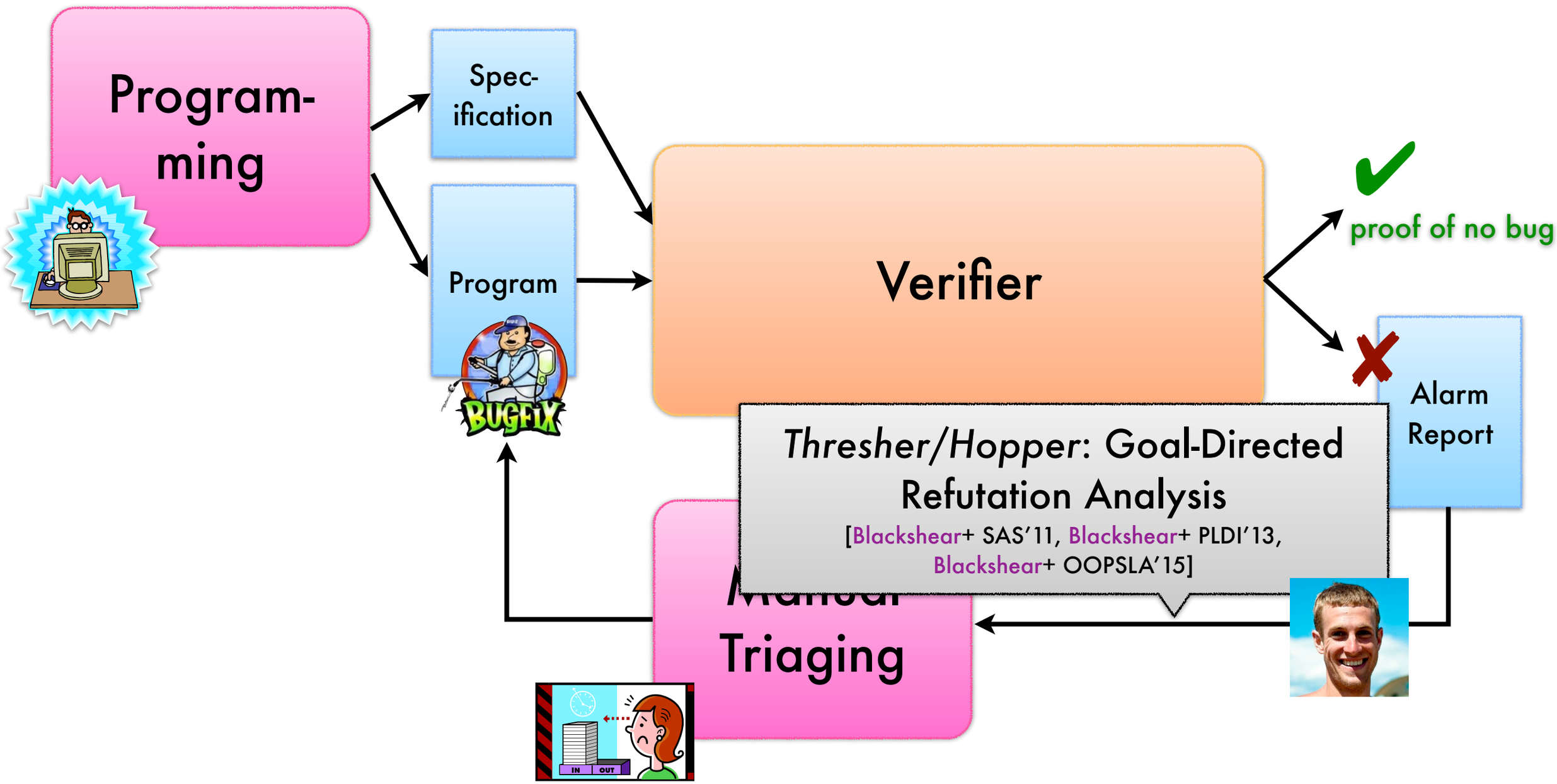
Research Agenda: The cooperative approach addresses the whole bug mitigation process.



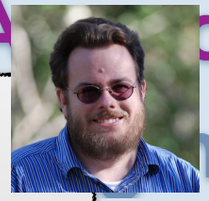
Research Agenda: The cooperative approach addresses the whole bug mitigation process.



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



Research Agenda: The cooperative approach addresses mitigation process.



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]

Program-ming

Spec-ification

Program

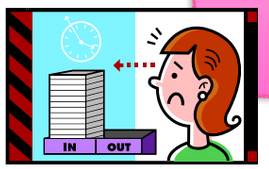
Verifier

✔ proof of no bug

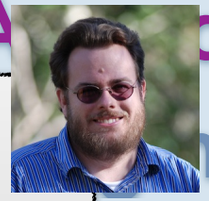
✘ Alarm Report

Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

Manual Triaging



Research Agenda: The cooperative approach addresses mitigation process.



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]

Program-ming

Test Input

Spec-ification

Program

Runner

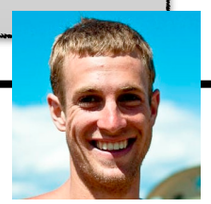
Verifier

Test Output

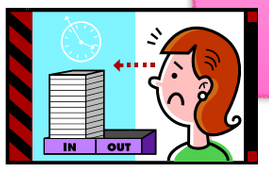
✔ proof of no bug

✘ Alarm Report

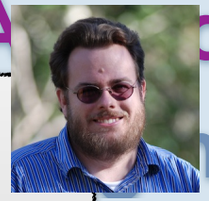
Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]



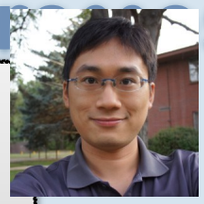
Manual Triaging



Research Agenda: The cooperative approach addresses mitigation process



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]



Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

Program-ming

Test Input

Spec-ification

Program

Runner

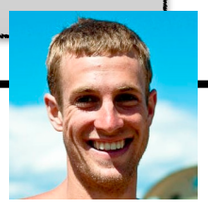
Verifier

Test Output

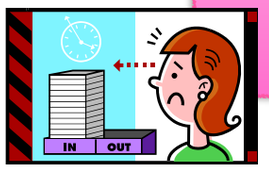
✔ proof of no bug

✘ Alarm Report

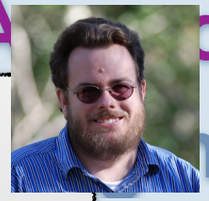
Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]



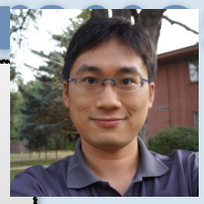
Manual Triaging



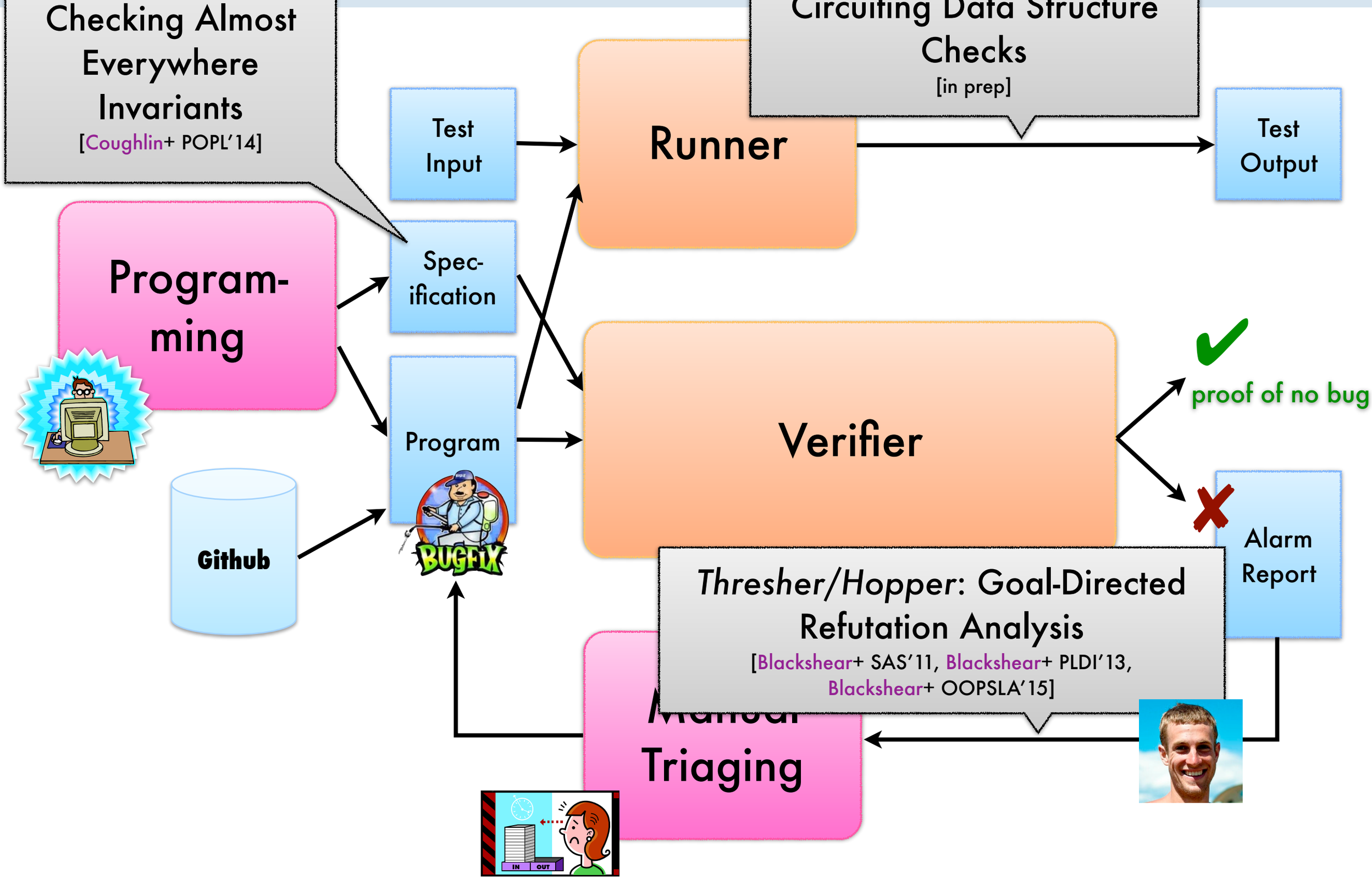
Research Agenda: The cooperative approach addresses mitigation process



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]

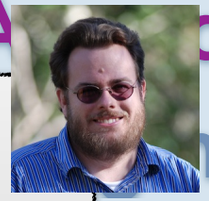


Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

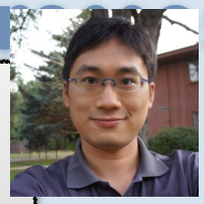


Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

Research Agenda: The cooperative approach addresses mitigation process



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]



Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

Program-ming

Test Input

Runner

Test Output

Spec-ification

Program

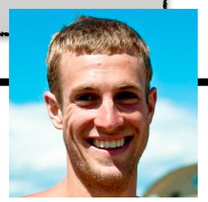
Verifier

proof of no bug

Alarm Report

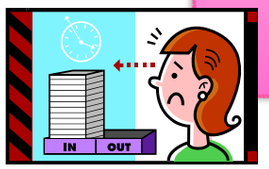
Github

Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

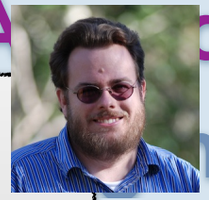


Manual Triaging

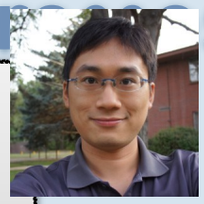
Fixr: Mining Bug Fixes from Commits
[in progress]



Research Agenda: The cooperative approach addresses mitigation process



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]



Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

Test Input

Runner

Test Output

Program-ming

Spec-ification

Jsana: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP'13, Cox+ SAS'14, Cox+ ESOP'15]

✔ proof of no bug

Program

Verifier

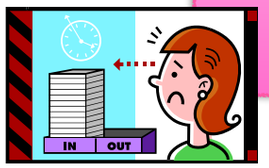
✘

Alarm Report

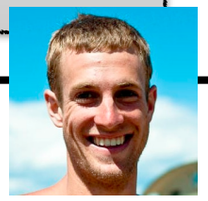
Github

Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

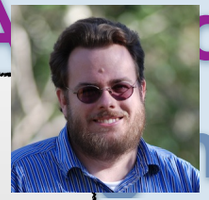
Fixr: Mining Bug Fixes from Commits
[in progress]



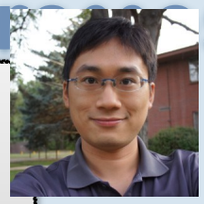
Manual Triaging



Research Agenda: The cooperative approach addresses mitigation process



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]



Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

Test Input

Runner

Test Output

Program-ming

Spec-ification

Jsana: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP'13, Cox+ SAS'14, Cox+ ESOP'15]

✔ proof of no bug

Program

Verifier

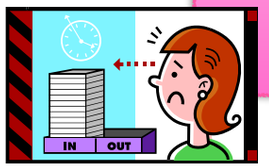
✘

Alarm Report

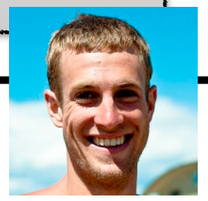
Github

Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

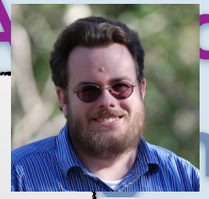
Fixr: Mining Bug Fixes from Commits
[in progress]



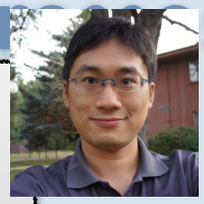
Manual Triaging



Research A da: The cooperative approach addr s mitigation proce



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]



Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

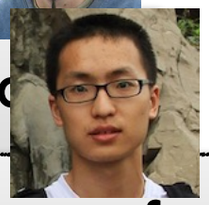
Test Input

Runner

Test Output

Program-ming

Spec-ification



Jsana: Abstract Domain Combin

Auditr: Securing Against "Inputs of Coma and Exposure"
[in progress]

of of no bug

Program

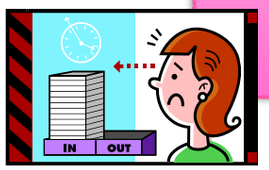
Alarm Report

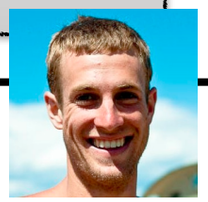
Github

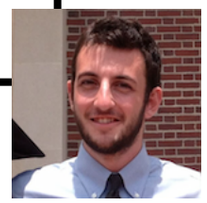
Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

Fixr: Mining Bug Fixes from Commits
[in progress]

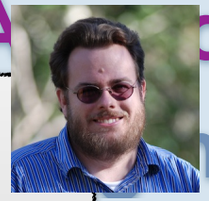
Manual Triaging







Research Agenda: The cooperative approach addresses mitigation process.

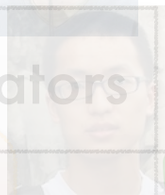


Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]

Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

Test Input → Runner → Test Output

Program-ming → Spec-ification, Program



Jsana: Abstract Domain Combinators for Dynamic Languages

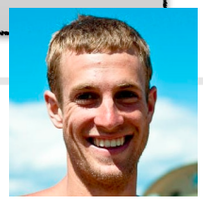
Audit: Securing Against "Inputs of Coma and Exposure"
[in progress]

Verifier → proof of no bug / Alarm Report

Github → Program

This Talk

Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]



Fixr: Mining Bug Fixes from Commits
[in progress]

Manual Triaging

This Talk: Highlights

Thresher and Hopper: Goal-Directed Refutations for Heap Reachability

Assist in triage of queries about heap relations

- Idea: Assume alarms false, prove them so automatically
- Application: Find memory leaks and eliminate crashes in Android
  - Filters out ~90% of false alarms to expose true bugs
  - Going from ~450 hours of manual work to ~30 hours
- Application: Find crashes from unexpected event orderings
  - Prove ~92% dereferences safe

This Talk: Highlights

Thresher and Hopper: Goal-Directed Refutations for Heap Reachability

Assist in triage of queries about heap relations

- Idea: Assume alarms false, prove them so automatically
- Application: Find memory leaks and eliminate crashes in Android
  - Filters out ~90% of false alarms to expose true bugs
  - Going from ~450 hours of manual work to ~30 hours
- Application: Find crashes from unexpected event orderings
  - Prove ~92% dereferences safe

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 and Hopper: Goal-Directed Refutations for Heap Reachability

Assist in triage of queries about heap relations

- Idea: Assume alarms false, prove them so automatically
- Application: Find memory leaks and eliminate crashes in Android
  - Filters out ~90% of false alarms to expose true bugs
  - Going from ~450 hours of manual work to ~30 hours
- Application: Find crashes from unexpected event orderings
  - Prove ~92% dereferences safe

Fissile Types

Strengthen type checking with symbolic analysis

- Interactive checking speeds: making
- Application: Prevent "MethodNotFound" errors in Objective-C (MacOS/iOS)

Goal-Directed Refutations for Heap Reachability

A bug that manifests spectacularly …

A bug that manifests spectacularly …

A bug that manifests spectacularly …

A bug that manifests spectacularly ...

A bug that manifests spectacularly ...

Android memory leaks underly rotation-based crashes.

Android memory leaks underly rotation-based crashes.

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 +1

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.

Android memory leaks underly rotation-based crashes.

stackoverflow Questions Tags Tour

Android: Crash on rot

phonegap ›

App crashes when ... Samsung phone

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

androidterm

Android Terminal Emulator

Project Home Downloads Wiki Issues Source

New issue Search Open issues for

Issue 20: Crashes when rotating phone horizontally

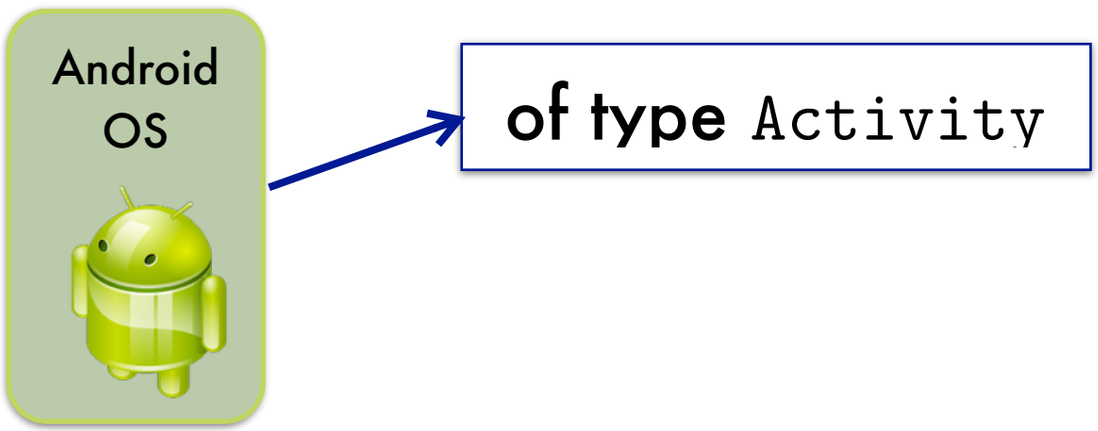
1 person starred this issue and may be notified of changes.

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

`Activity` **objects encapsulate the UI**

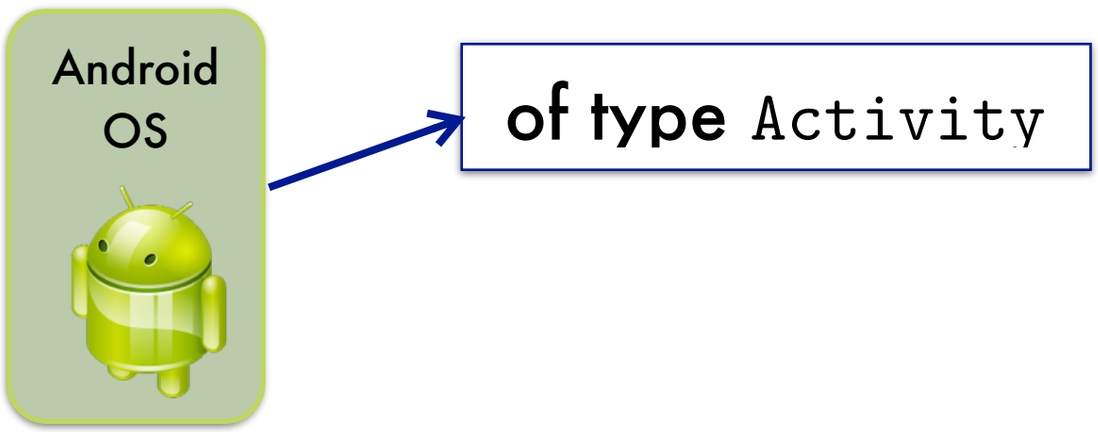
How can you have memory leaks with a garbage collected run-time?

`Activity` **objects encapsulate the UI**



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

`Activity` objects encapsulate the UI



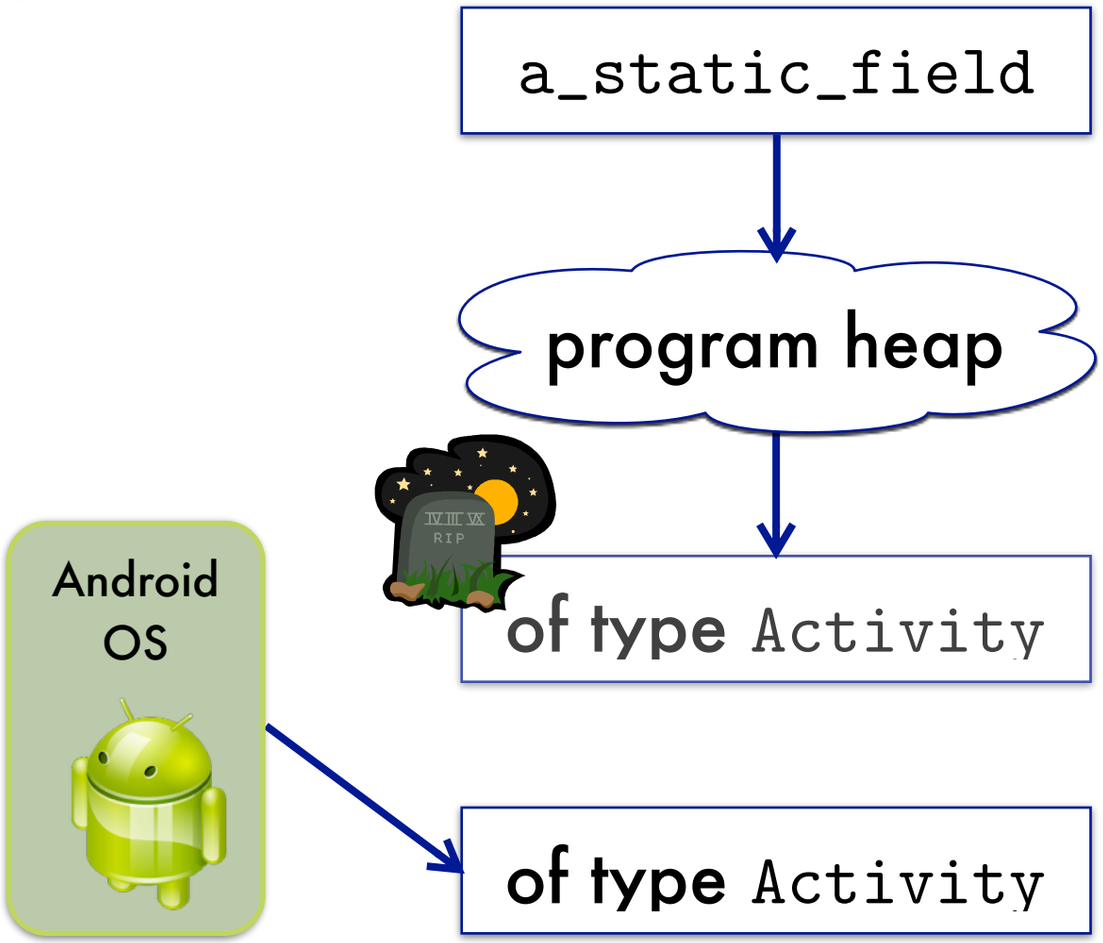
How can you have memory leaks with a garbage collected run-time?

`Activity` objects encapsulate the UI

of type `Activity`

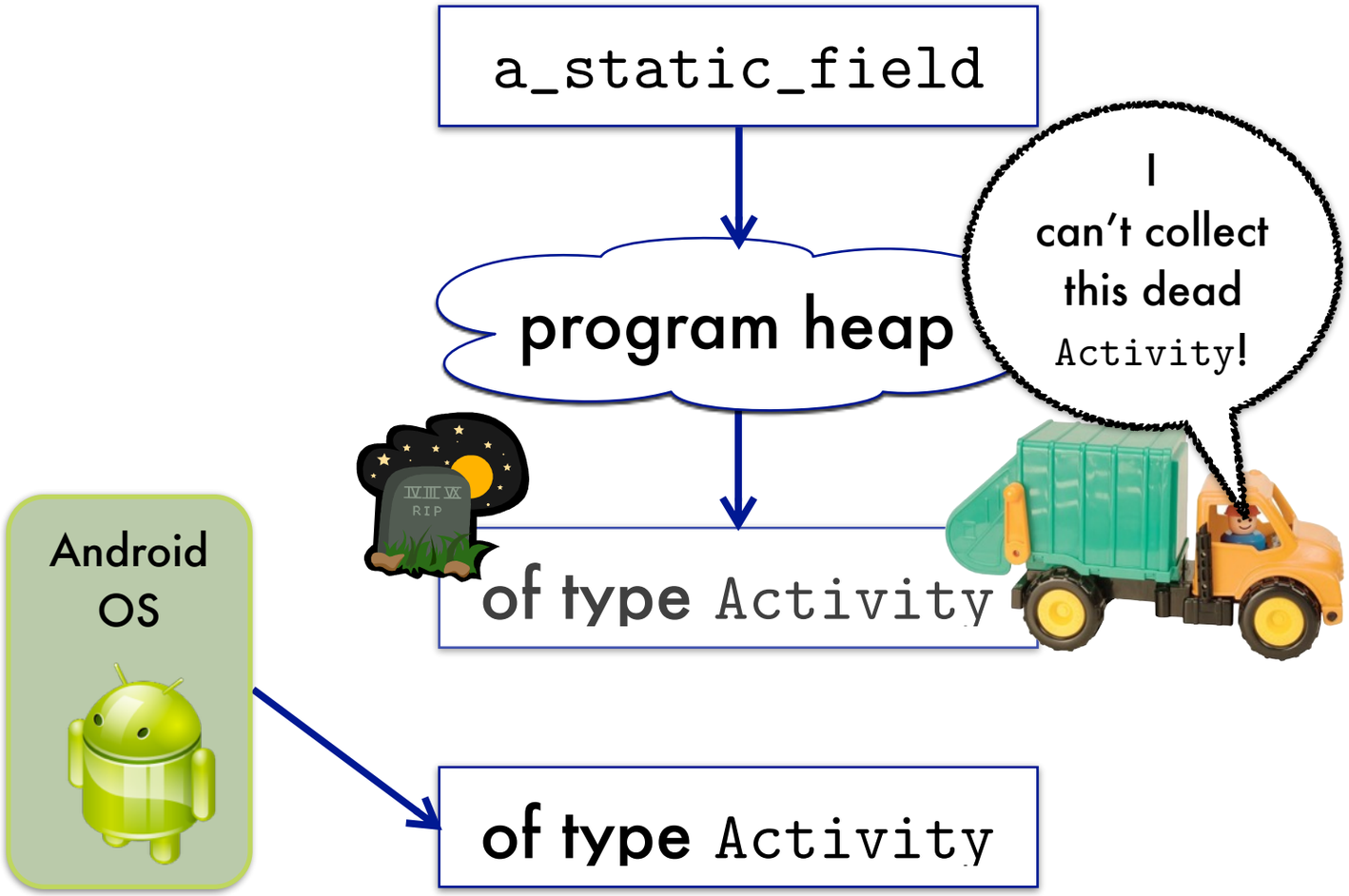
of type `Activity`

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



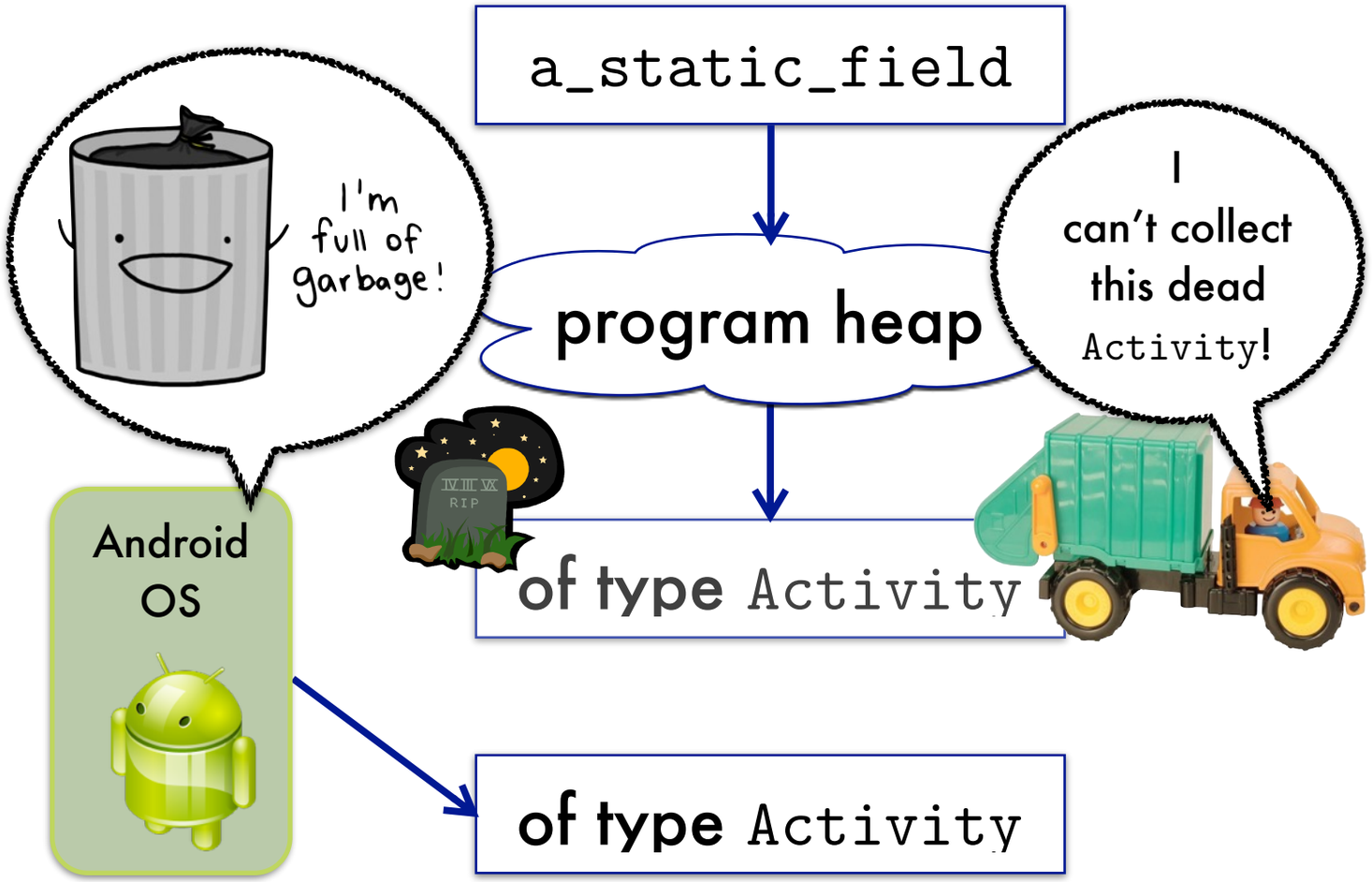
`Activity` **objects encapsulate the UI**

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



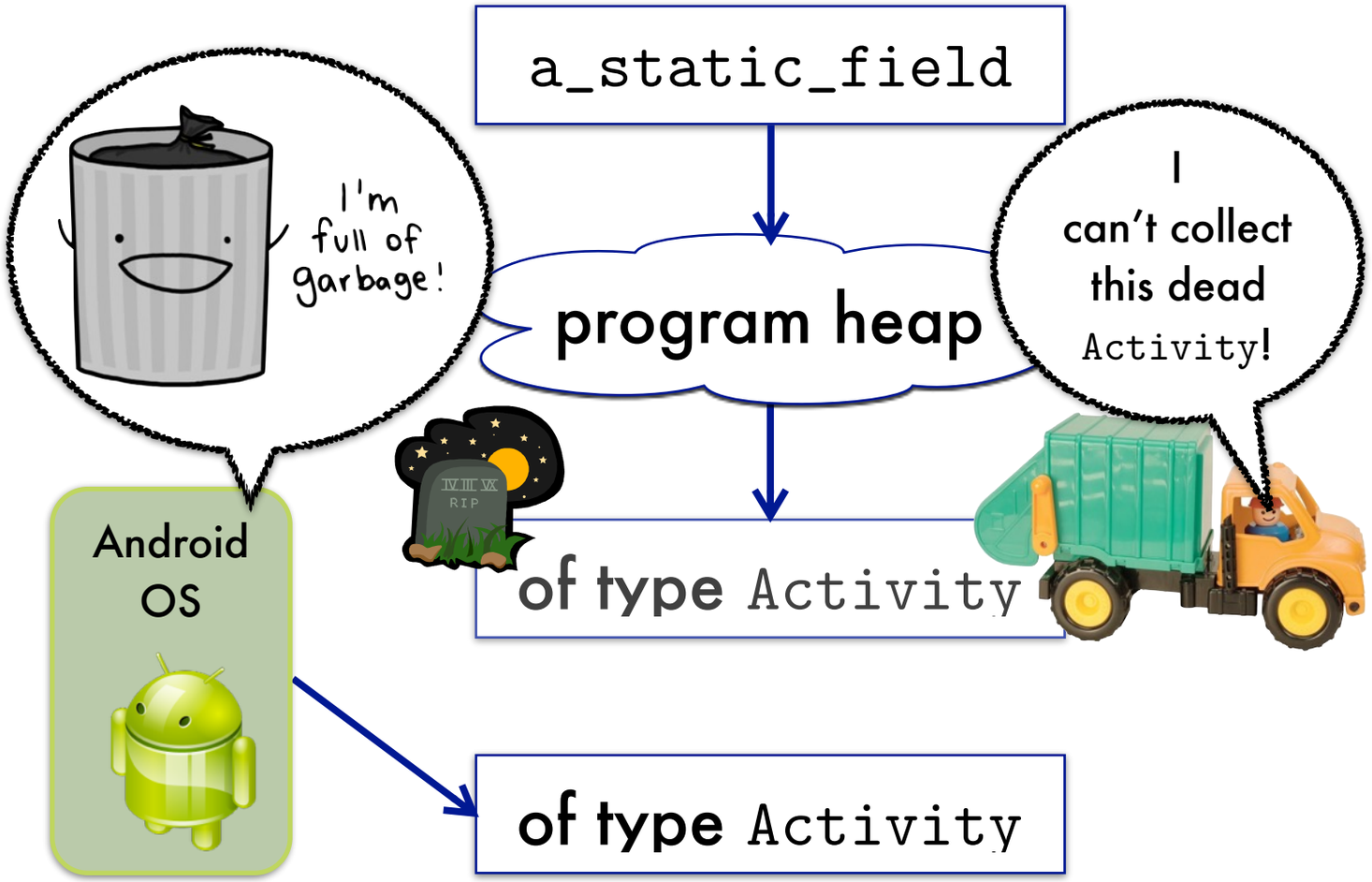
Activity objects encapsulate the UI

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



Activity objects encapsulate the UI

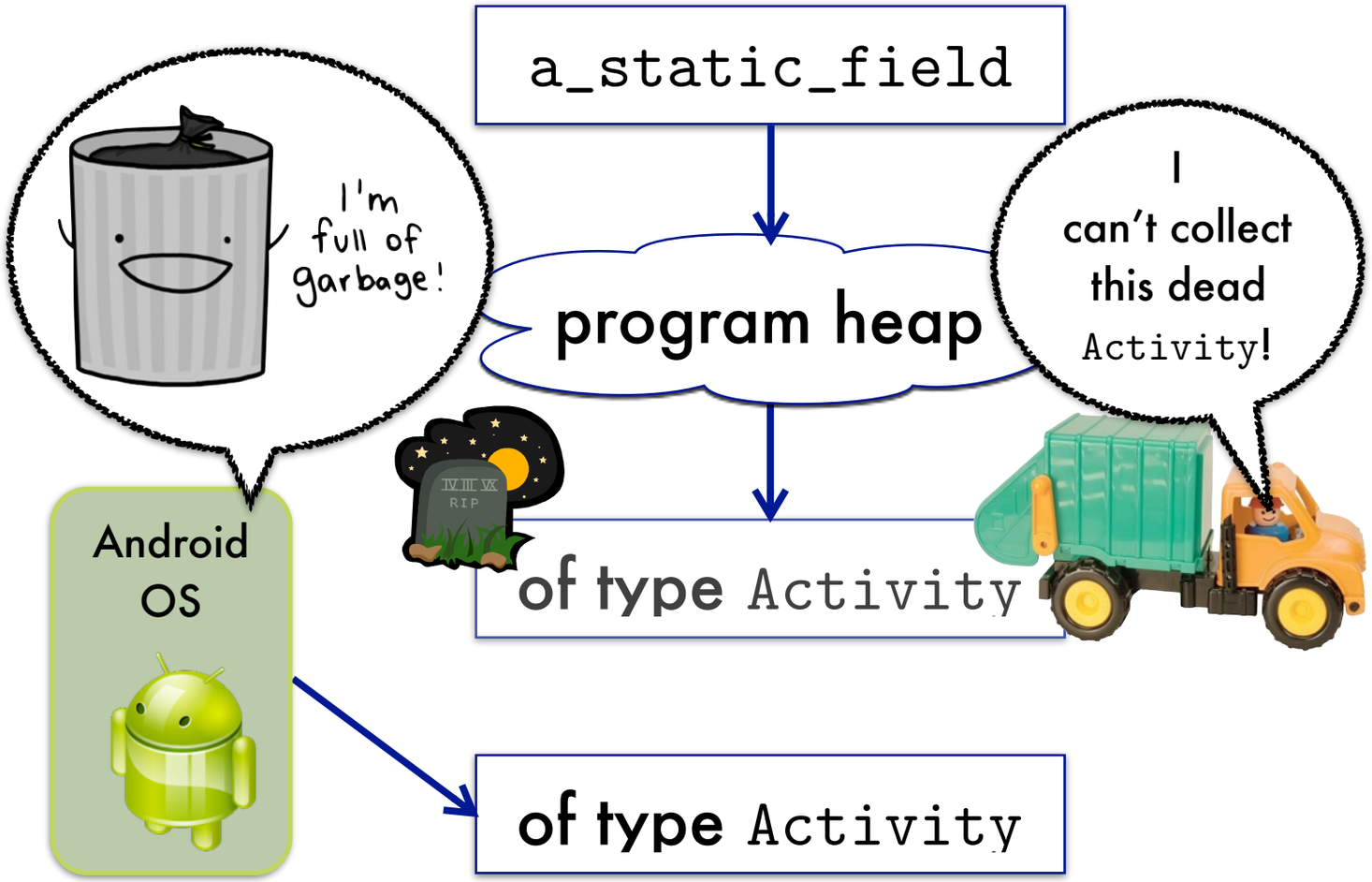
How can you have memory leaks with a garbage collected run-time?



Activity objects encapsulate the UI

Bug: Holding reference to "old" Activity

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



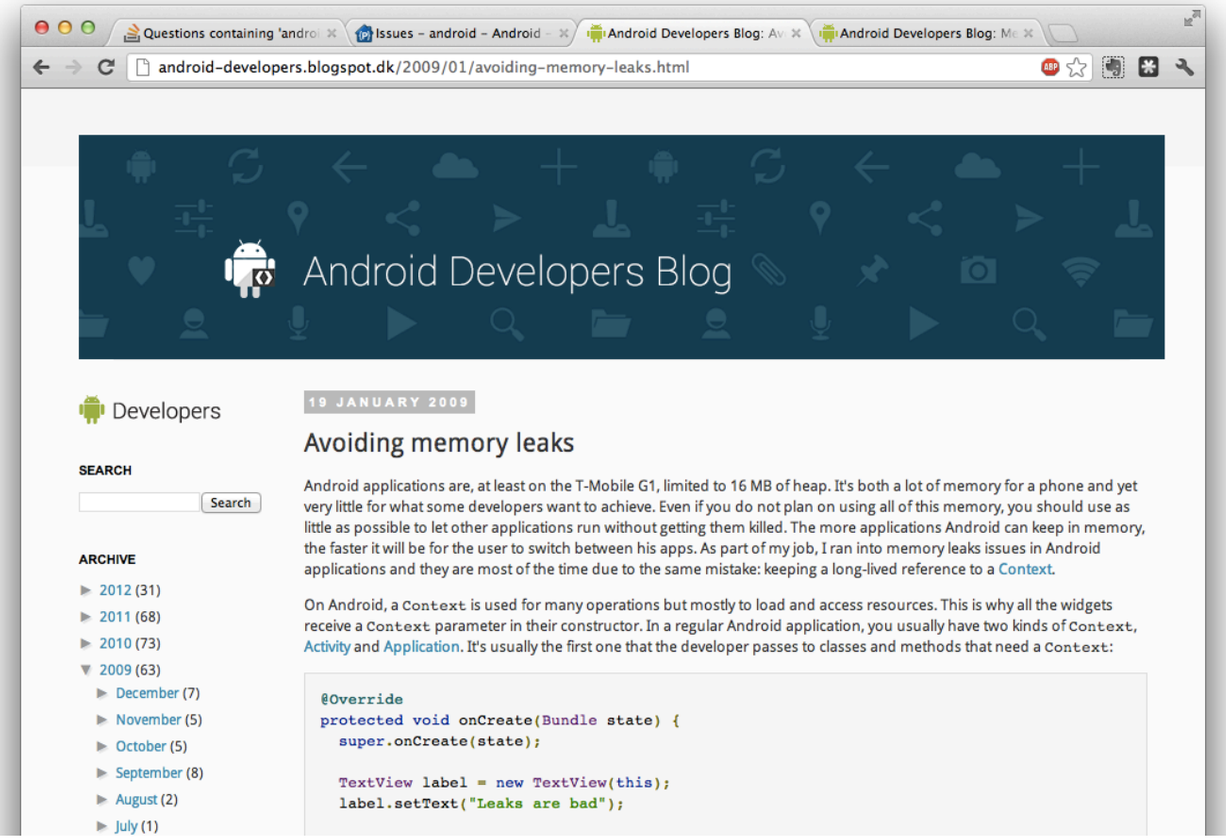
Activity objects encapsulate the UI

"an Activity leak"

Bug: Holding reference to "old" Activity

The expert recommendation ...

The expert recommendation ...



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

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

Android Developers Blog

Developers

SEARCH

Search

ARCHIVE

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

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.

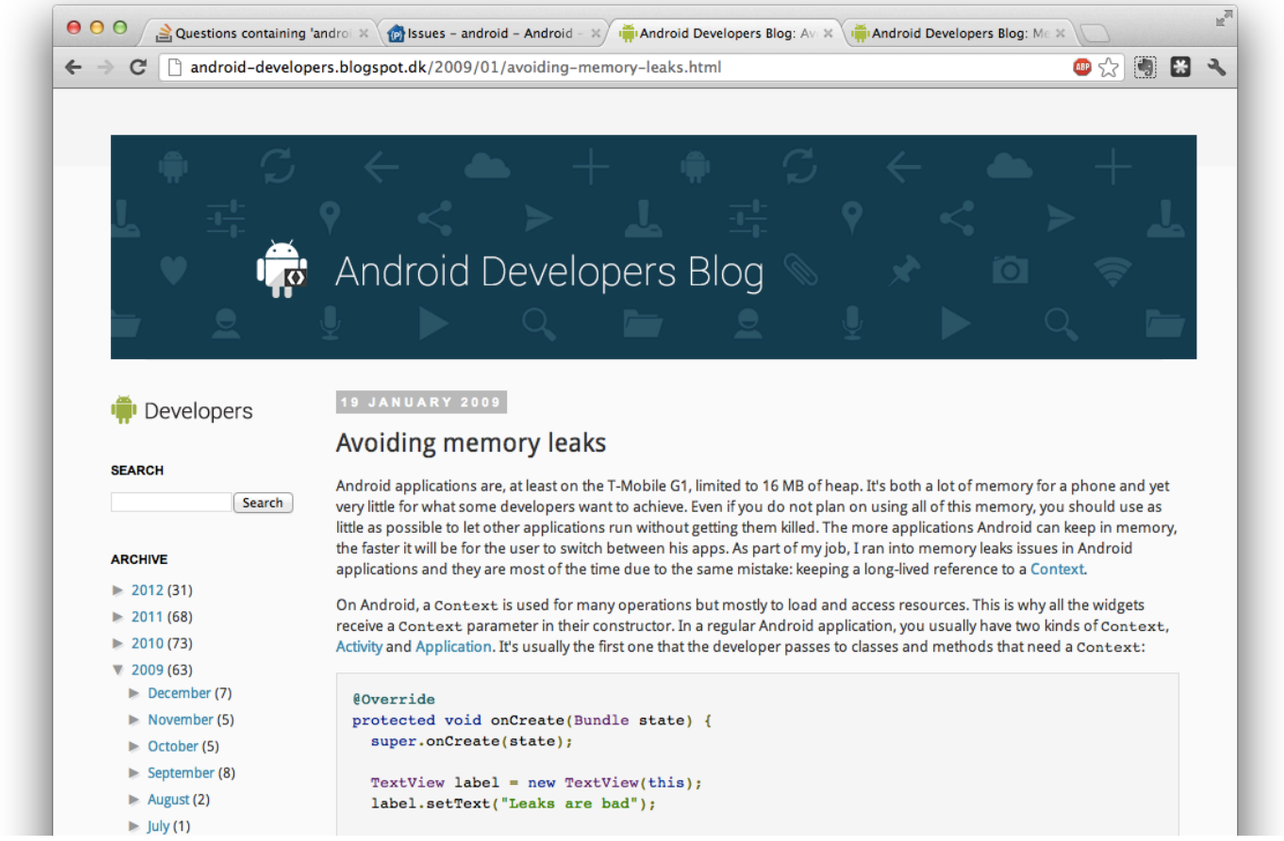
On Android, a `Context` is used for many operations but mostly to load and access resources. This is why all the widgets receive a `Context` parameter in their constructor. In a regular Android application, you usually have two kinds of `Context`, Activity and Application. It's usually the first one 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"



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

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

Android Developers Blog

Developers

SEARCH

Search

ARCHIVE

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

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.

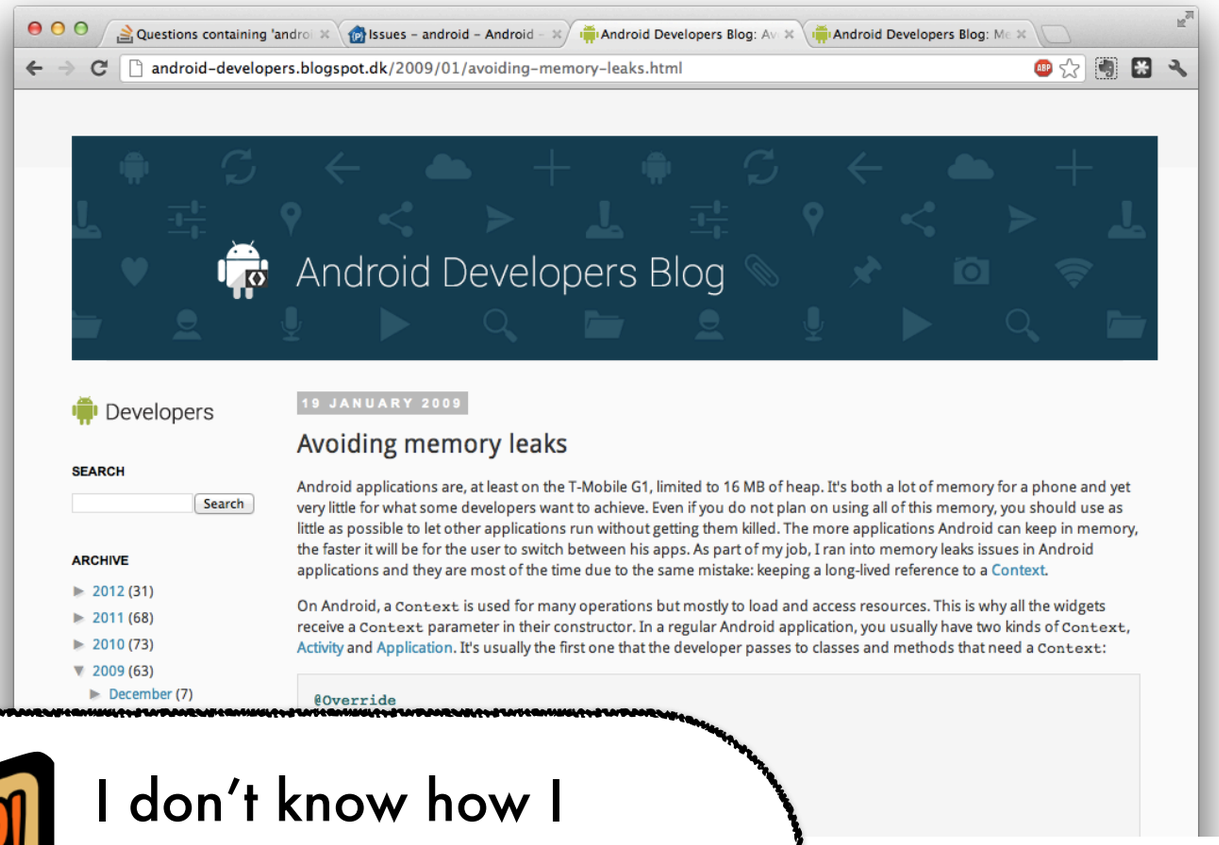
On Android, a `Context` is used for many operations but mostly to load and access resources. This is why all the widgets receive a `Context` parameter in their constructor. In a regular Android application, you usually have two kinds of `Context`, Activity and Application. It's usually the first one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

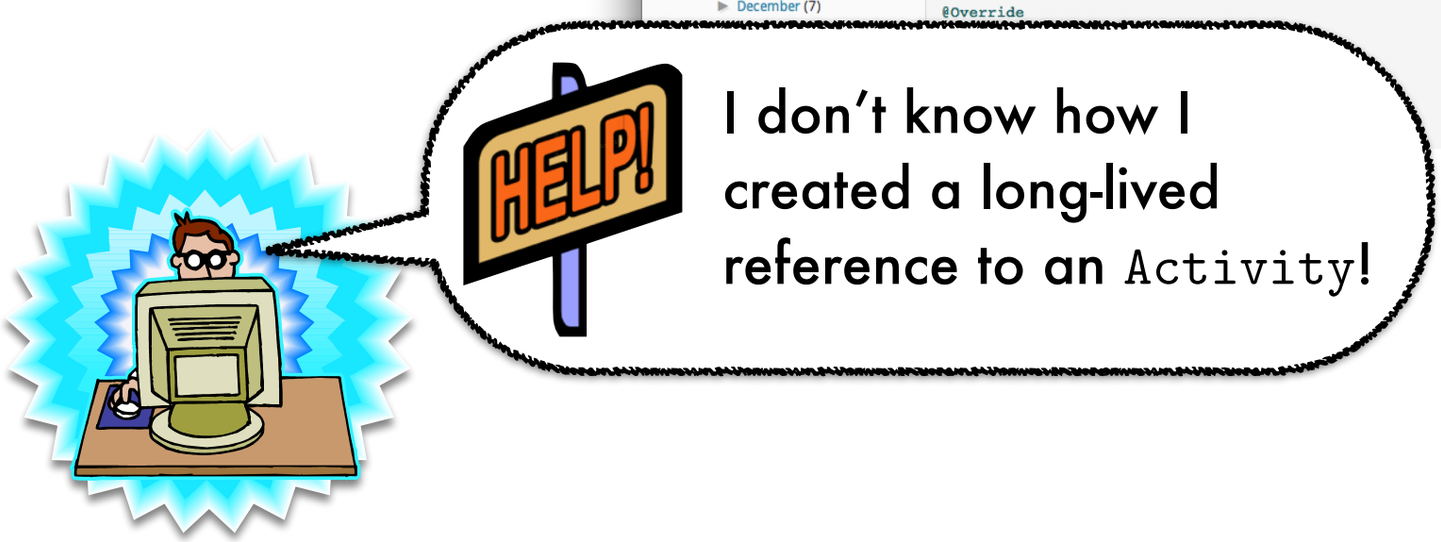
```
@Override
protected void onCreate(Bundle state) {
  super.onCreate(state);

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

The expert recommendation ...

"Do not keep long-lived references to a context-activity"





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 ...

Questions containing 'androi | Issues - android - Android | Android Developers Blog: Av | Android Developers Blog: Me

android-developers.blogspot.dk/2011/03/memory-analysis-for-android.html

Android Developers Blog

Developers

SEARCH

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.

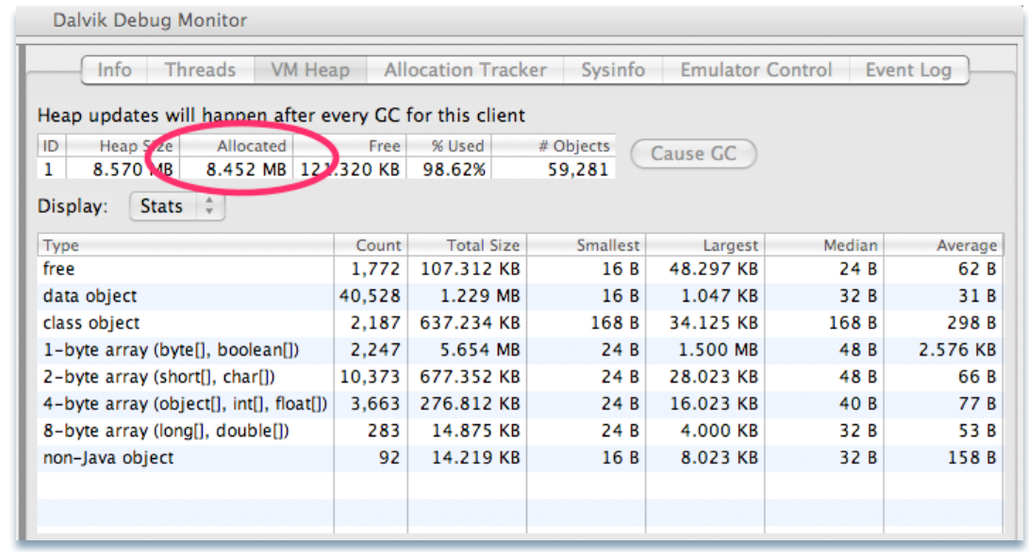
Tools of the trade

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 ...

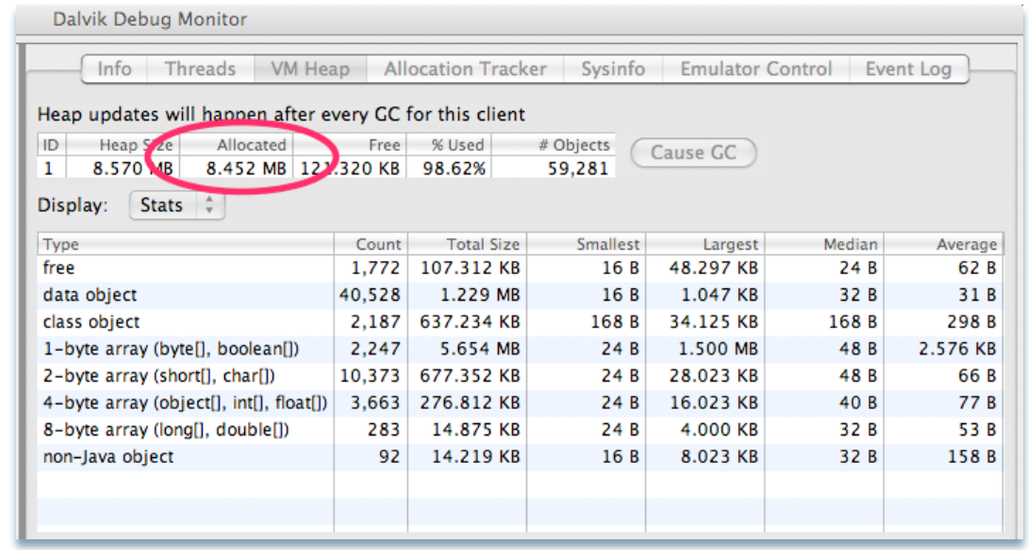
1. Run the app

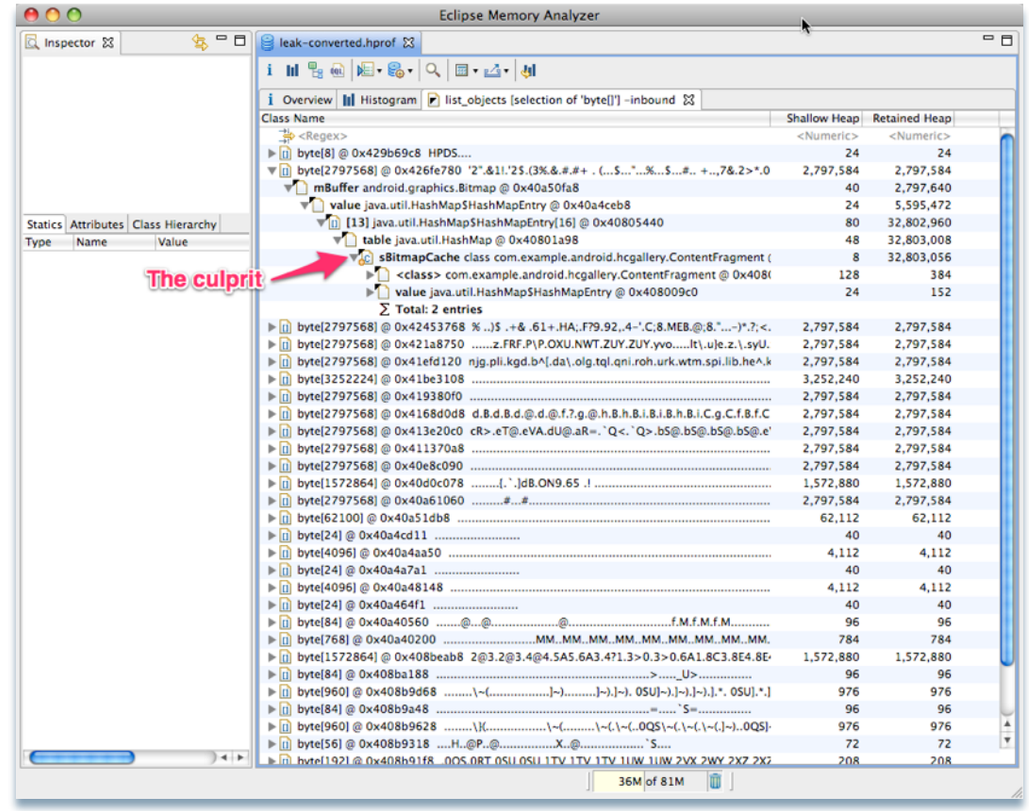
The state of practice in debugging `Activity` leaks ...



1. Run the app
2. Watch the heap usage

The state of practice in debugging `Activity` leaks ...





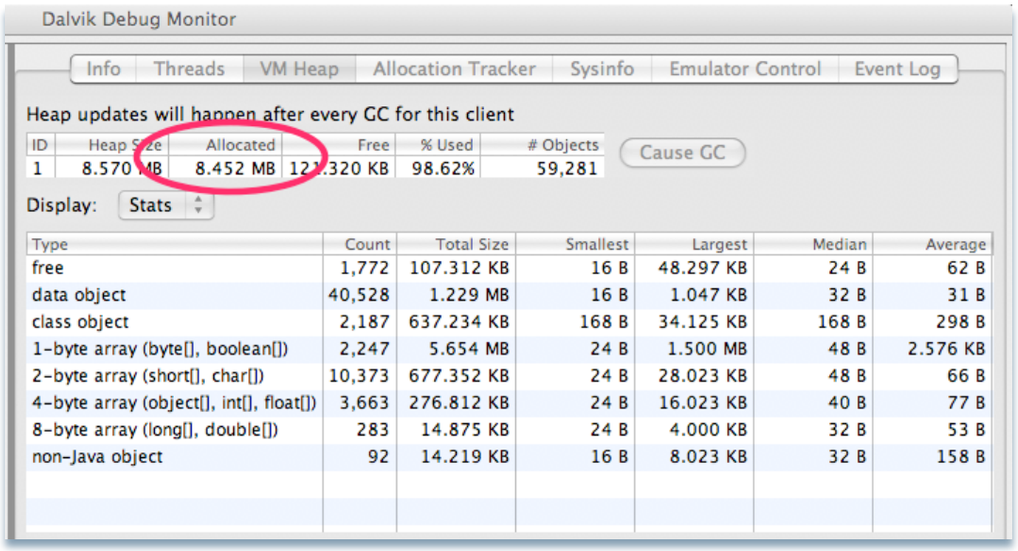
1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and hope to find the culprit

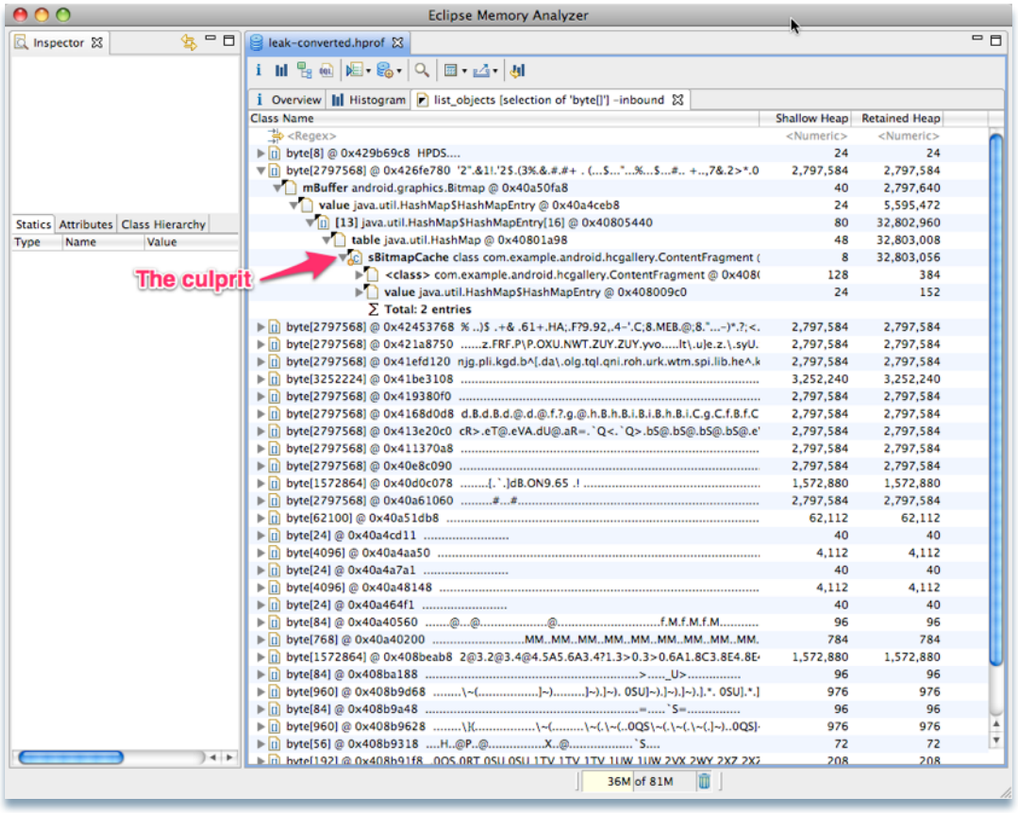
The state of practice in debugging `Activity` leaks ...

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

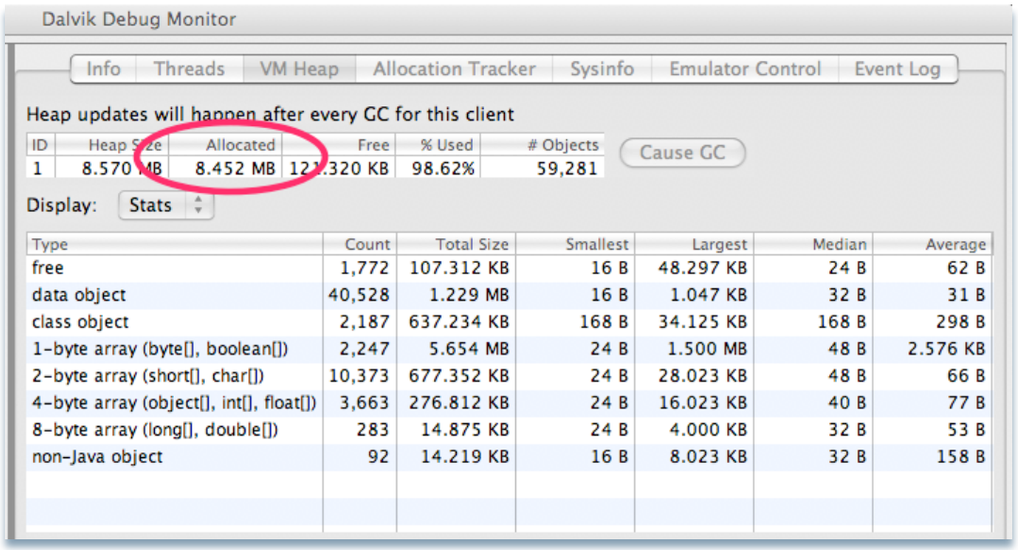


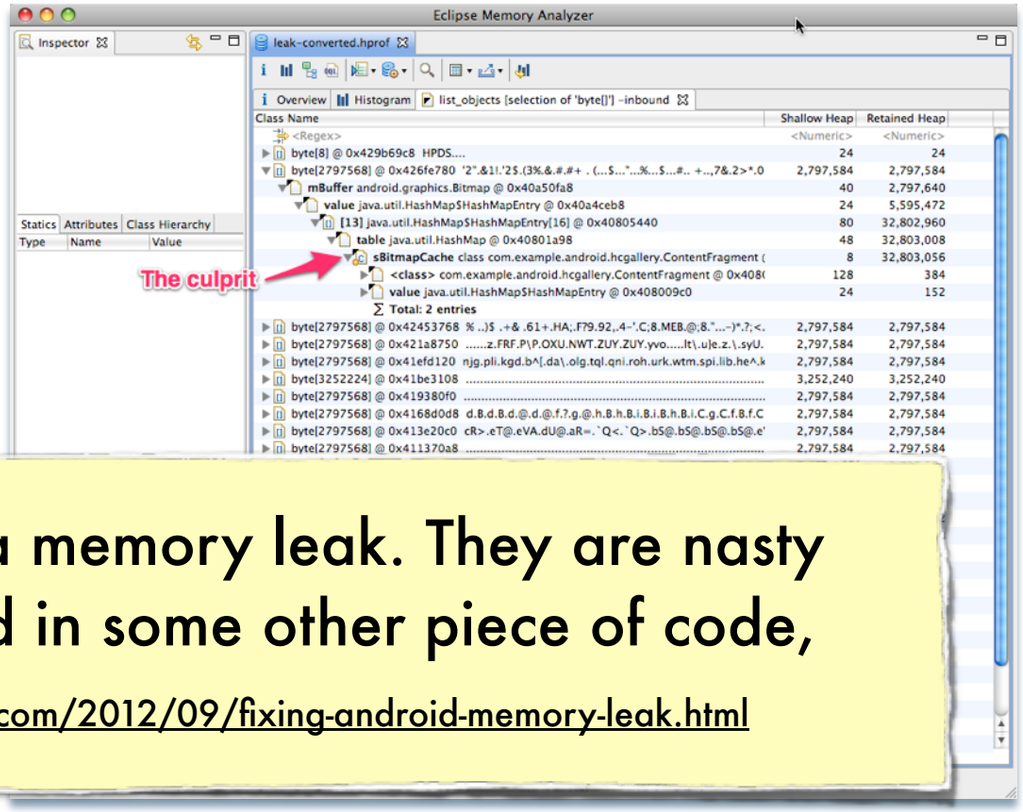


The state of practice in debugging `Activity` leaks ...

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?





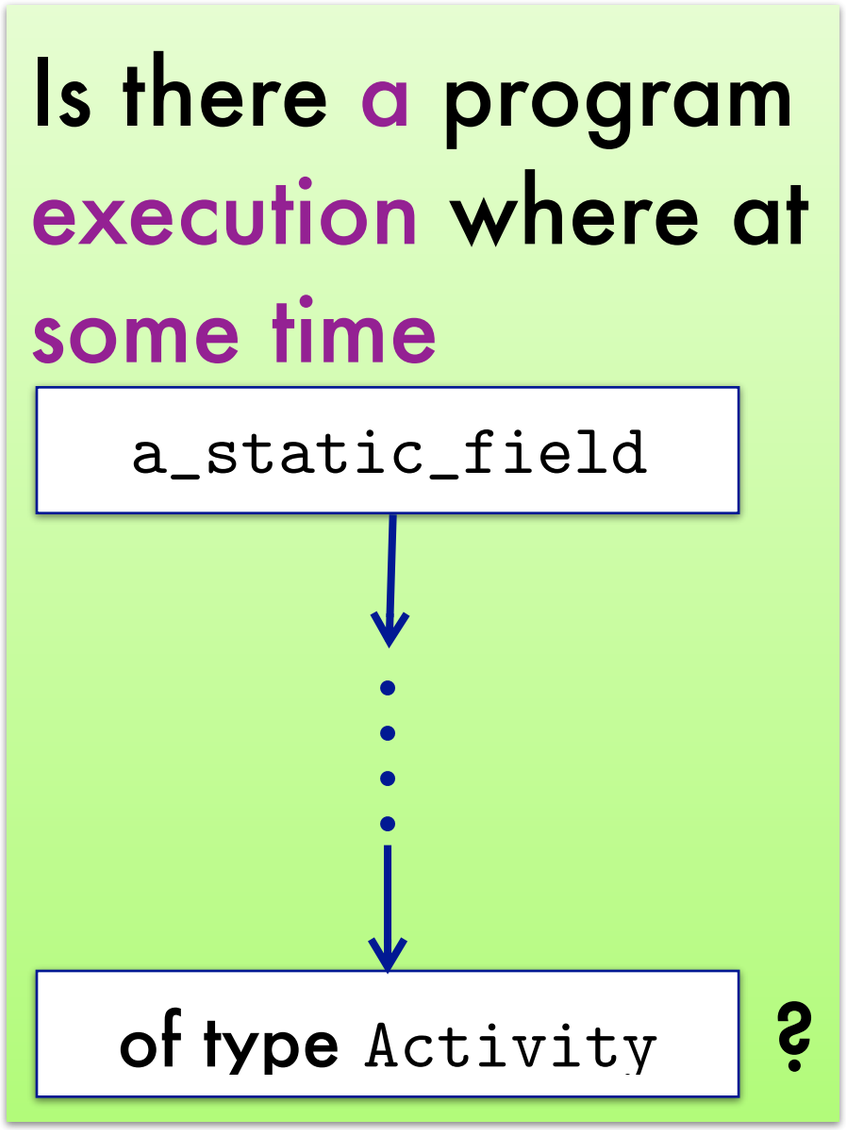
3. Dump the heap. Dig

"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?

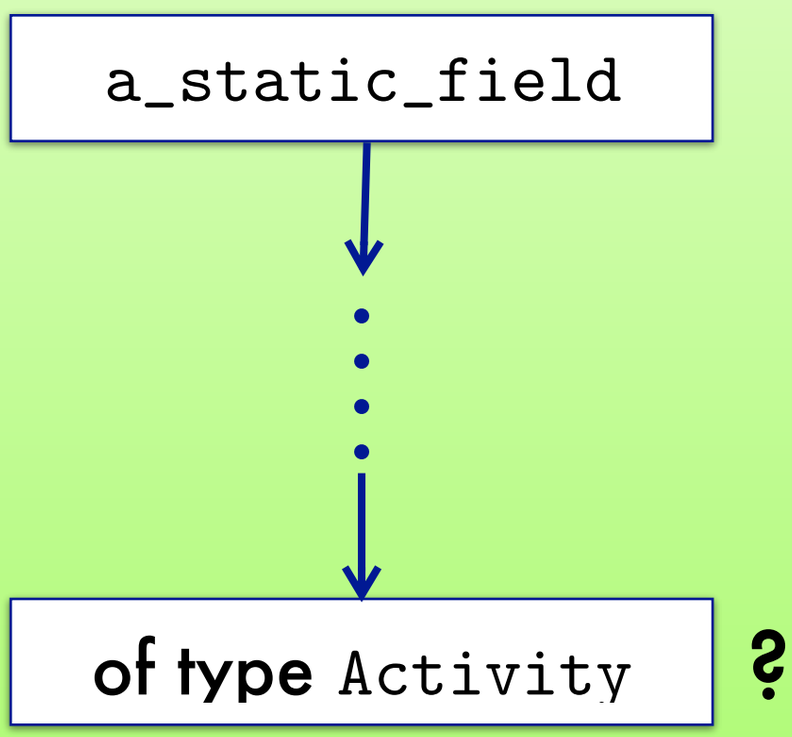


Example

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



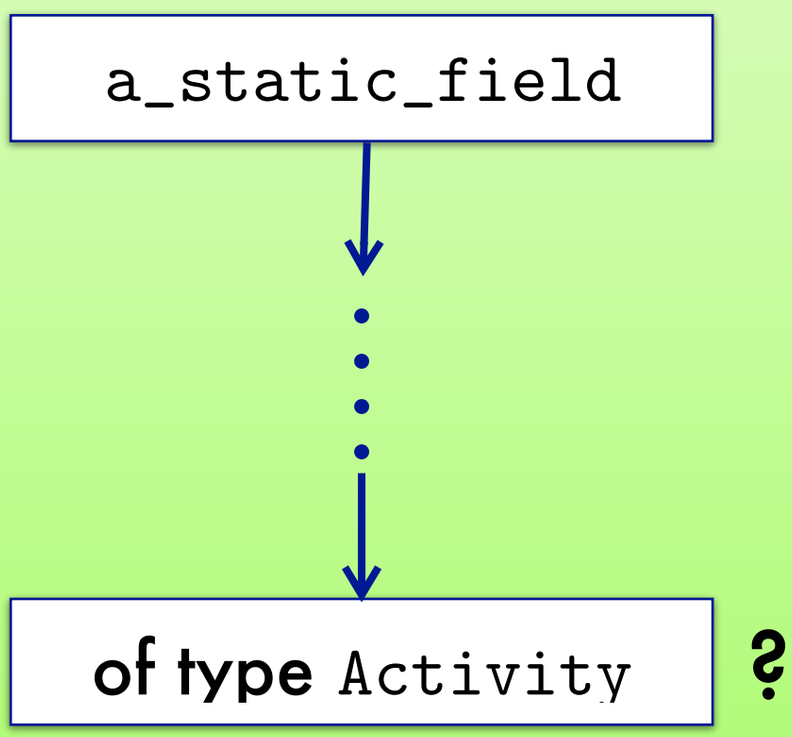
Example

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



Example

Can be answered with a points-to analysis

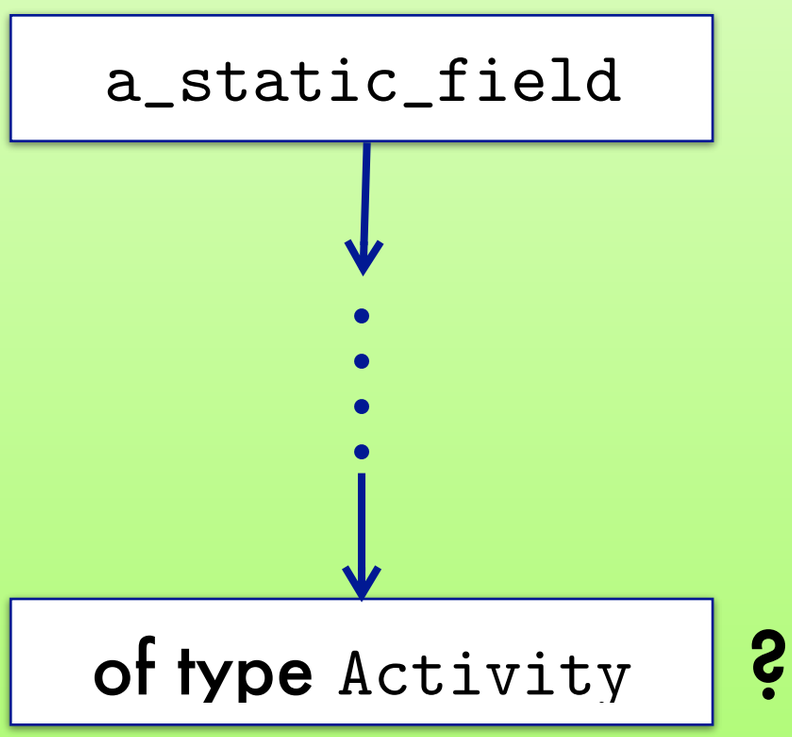
with approximation

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



Example

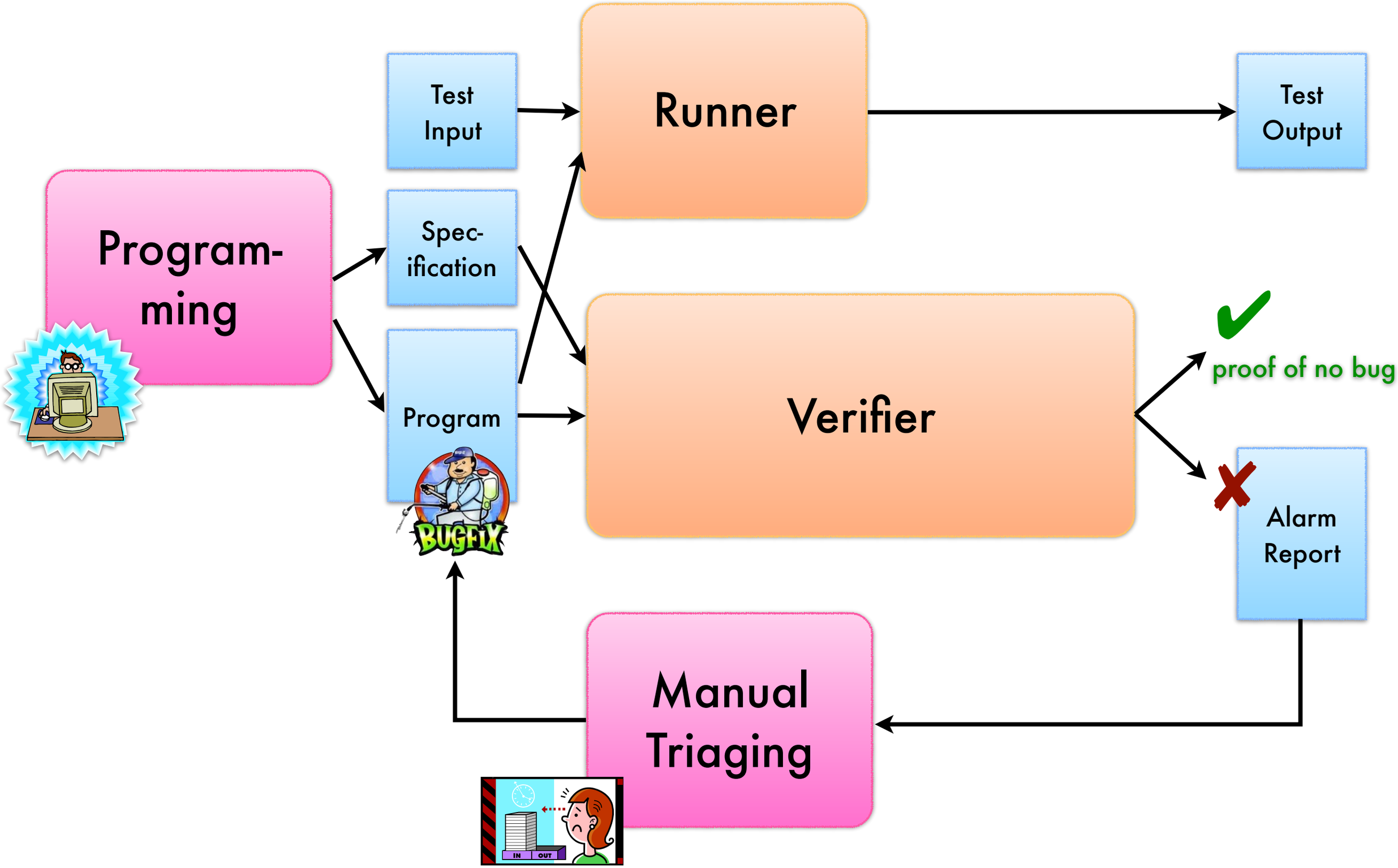
Can be answered with a points-to analysis

with approximation

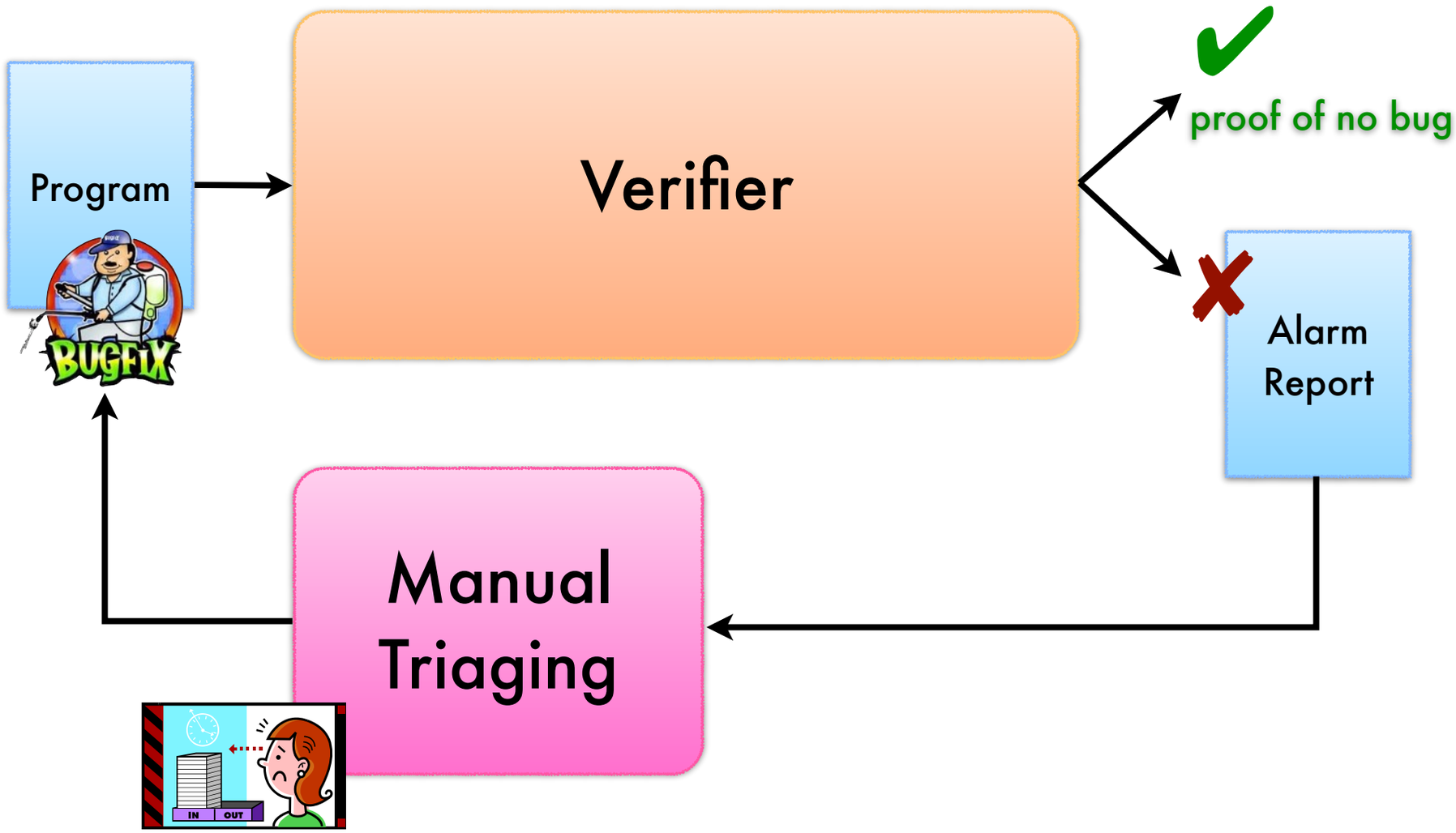
Hidden Truth

Some pointer relations may be false

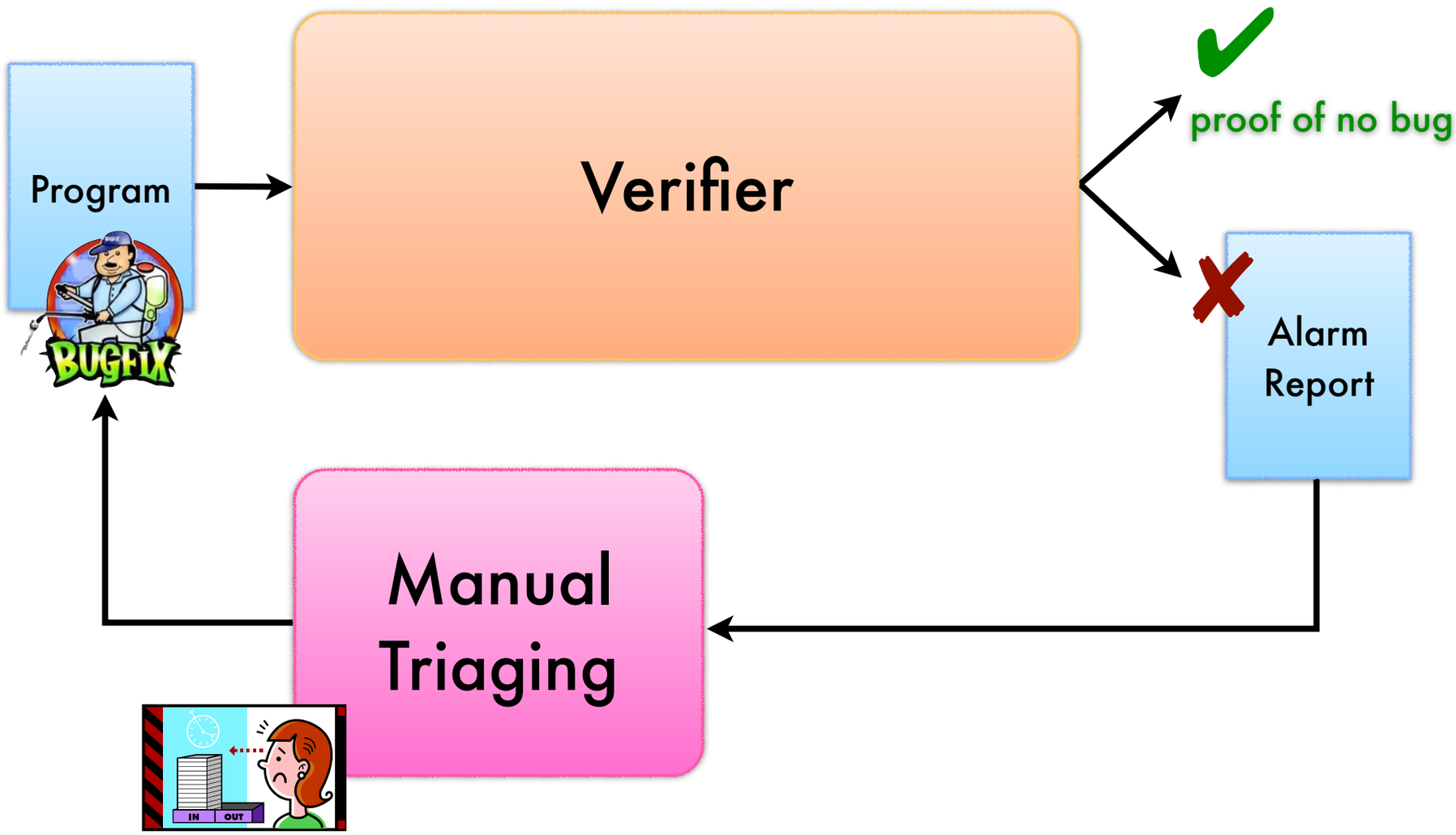
But with the cooperative approach ...



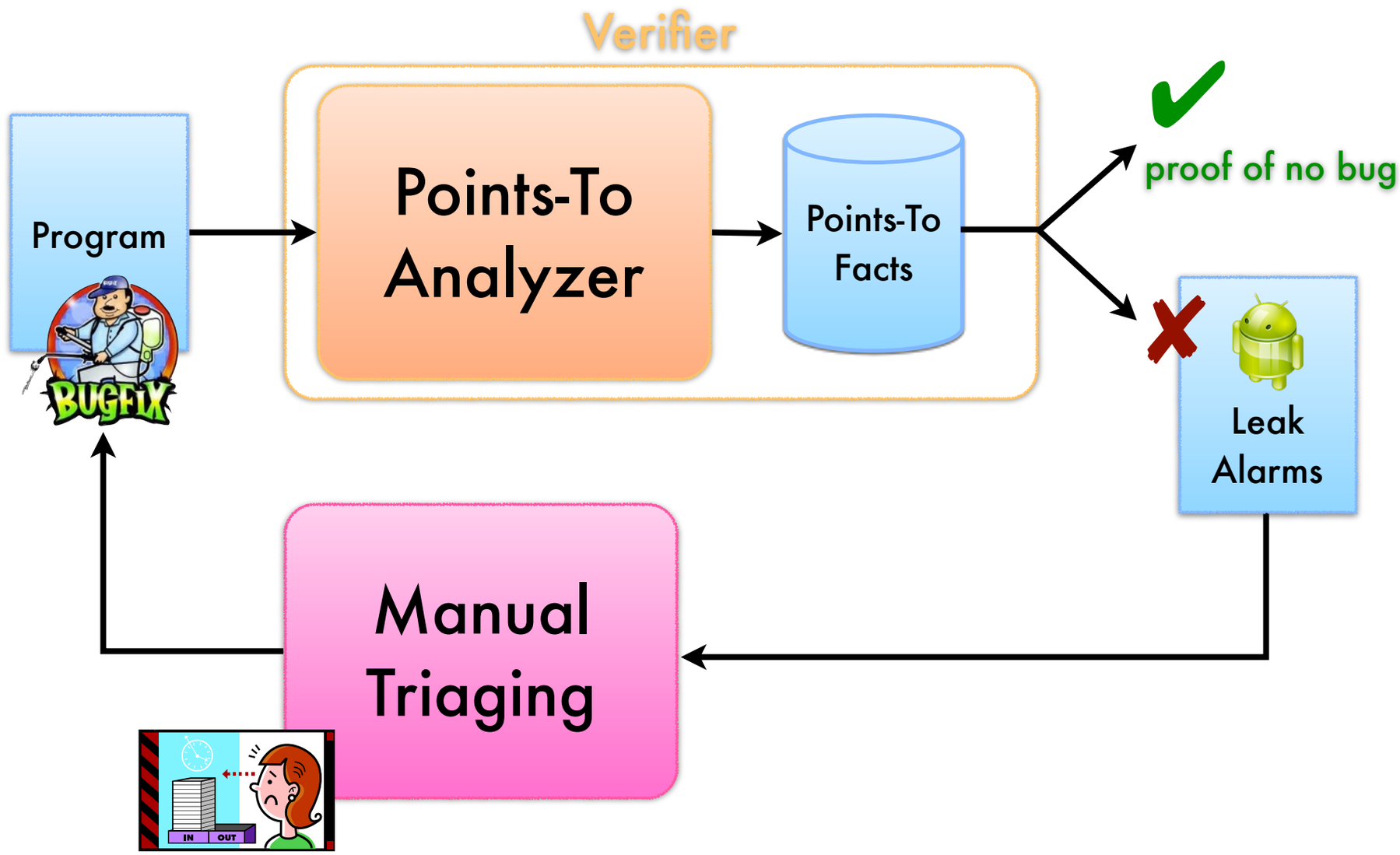
But with the cooperative approach ...



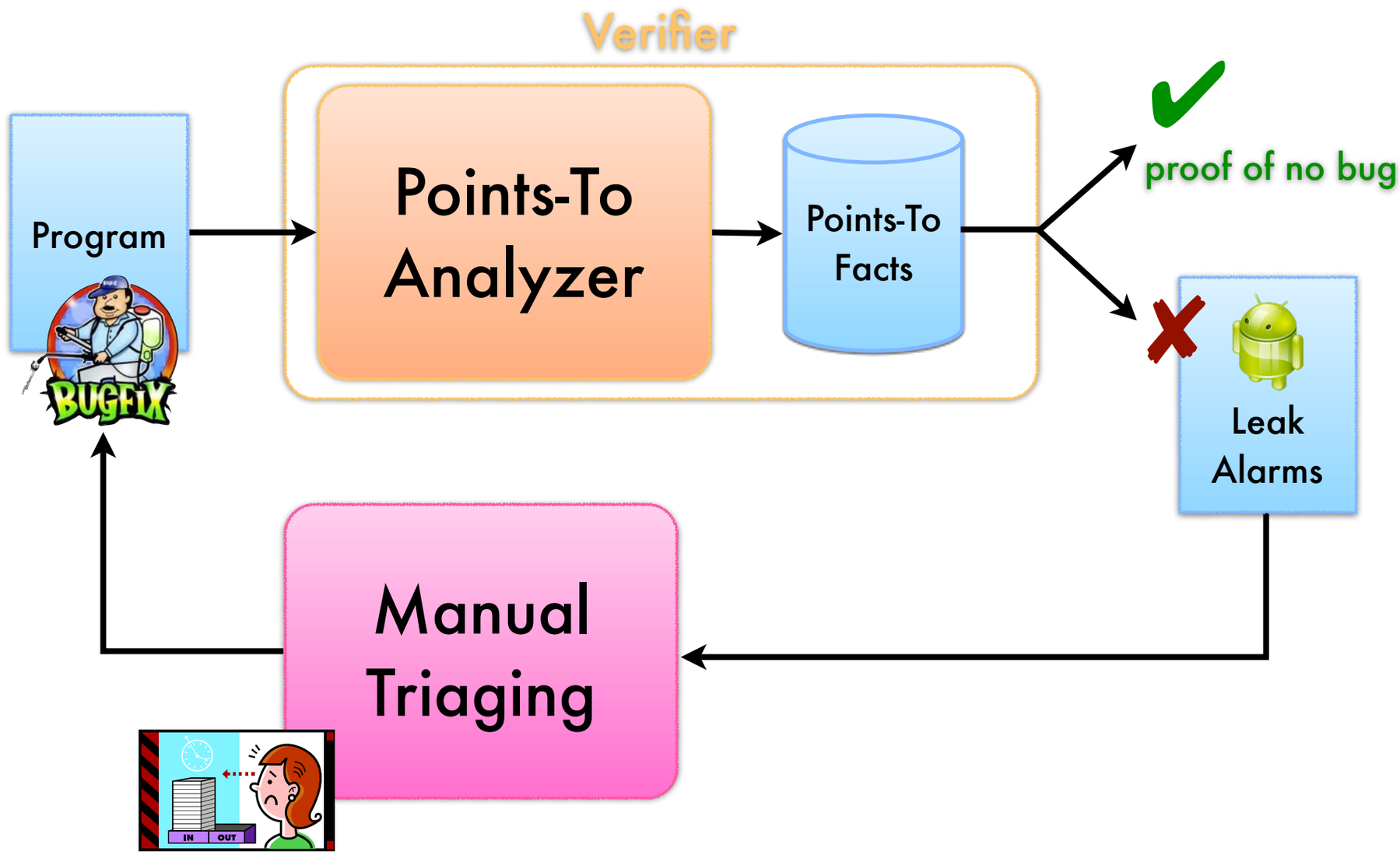
Thresher addresses alarm **triage** in a particularly challenging domain.



Thresher addresses alarm **triage** in a particularly challenging domain.

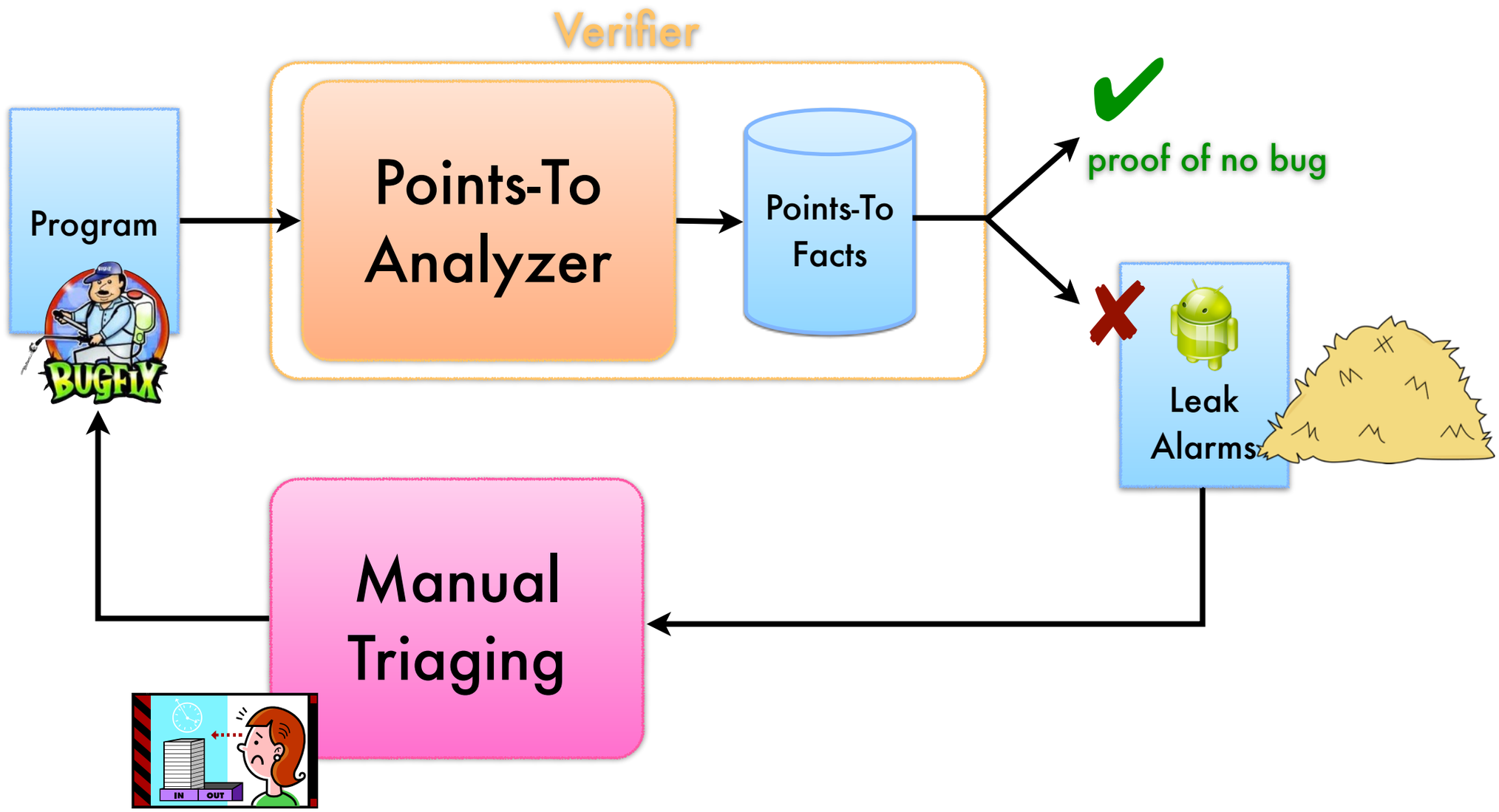


Thresher addresses alarm **triage** in a particularly challenging domain.



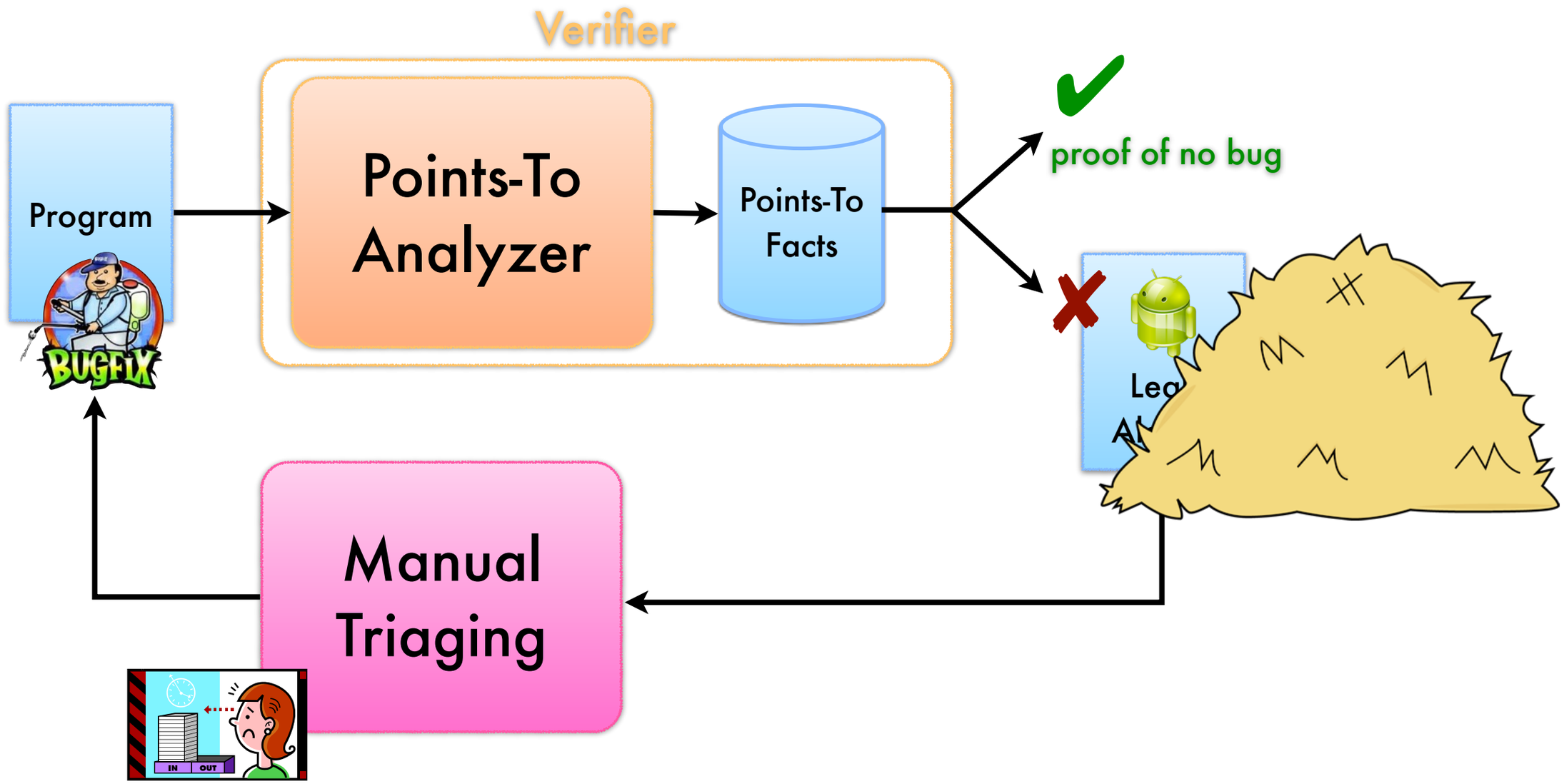
Known: Precise points-to analysis challenging

Thresher addresses alarm **triage** in a particularly challenging domain.



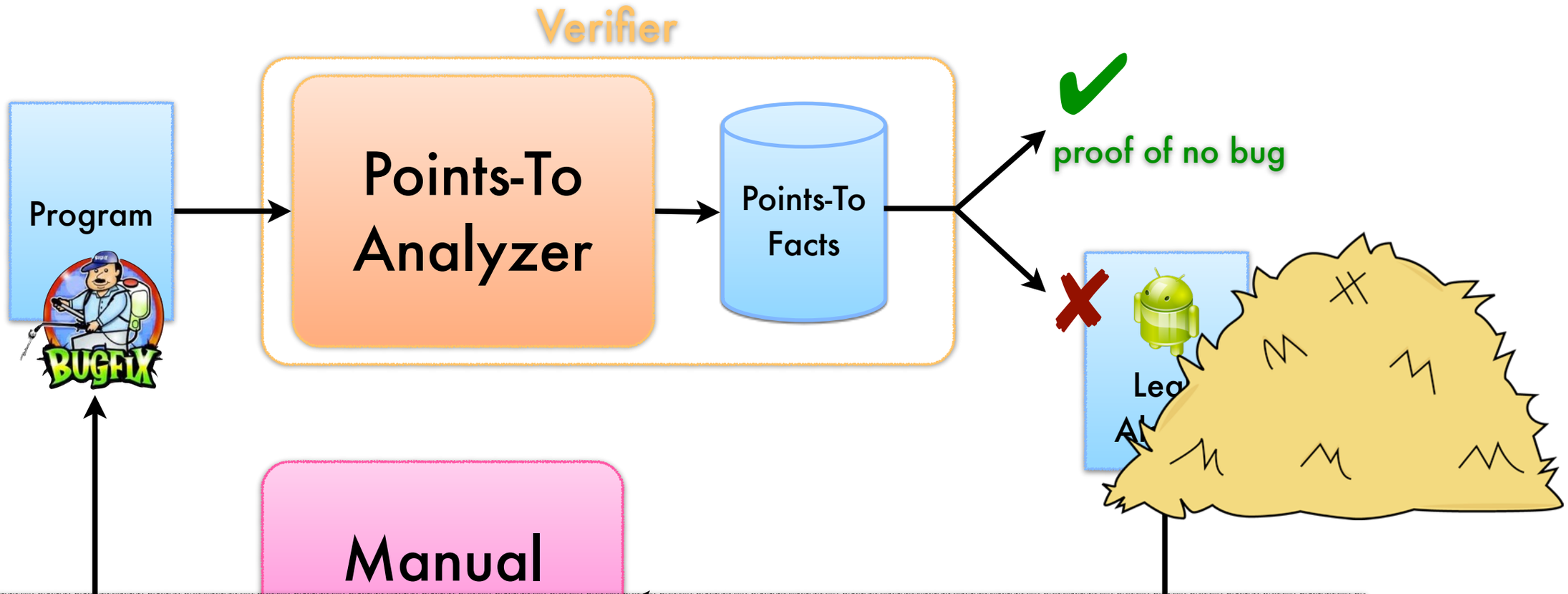
Known: Precise points-to analysis challenging

Thresher addresses alarm **triage** in a particularly challenging domain.



Known: Precise points-to analysis challenging

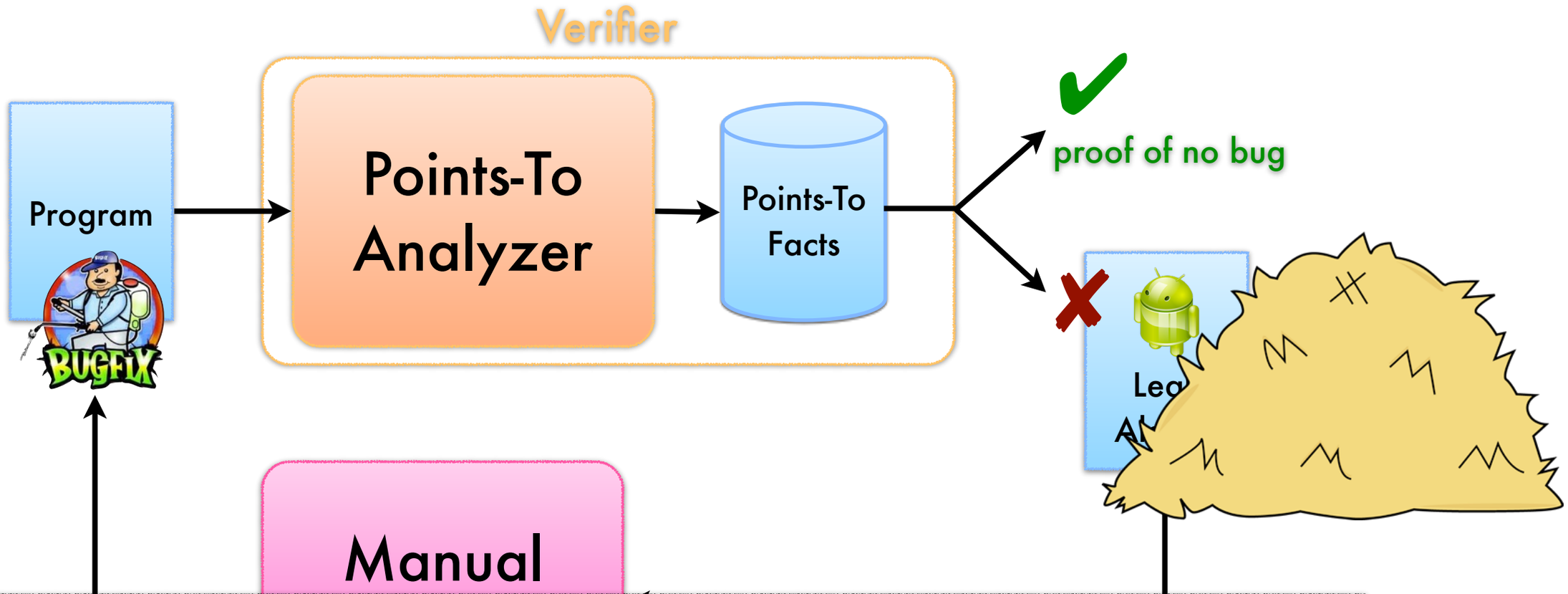
Thresher addresses alarm **triage** in a particularly challenging domain.



Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?"

Known: Precise points-to analysis challenging

Thresher addresses alarm **triage** in a particularly challenging domain.

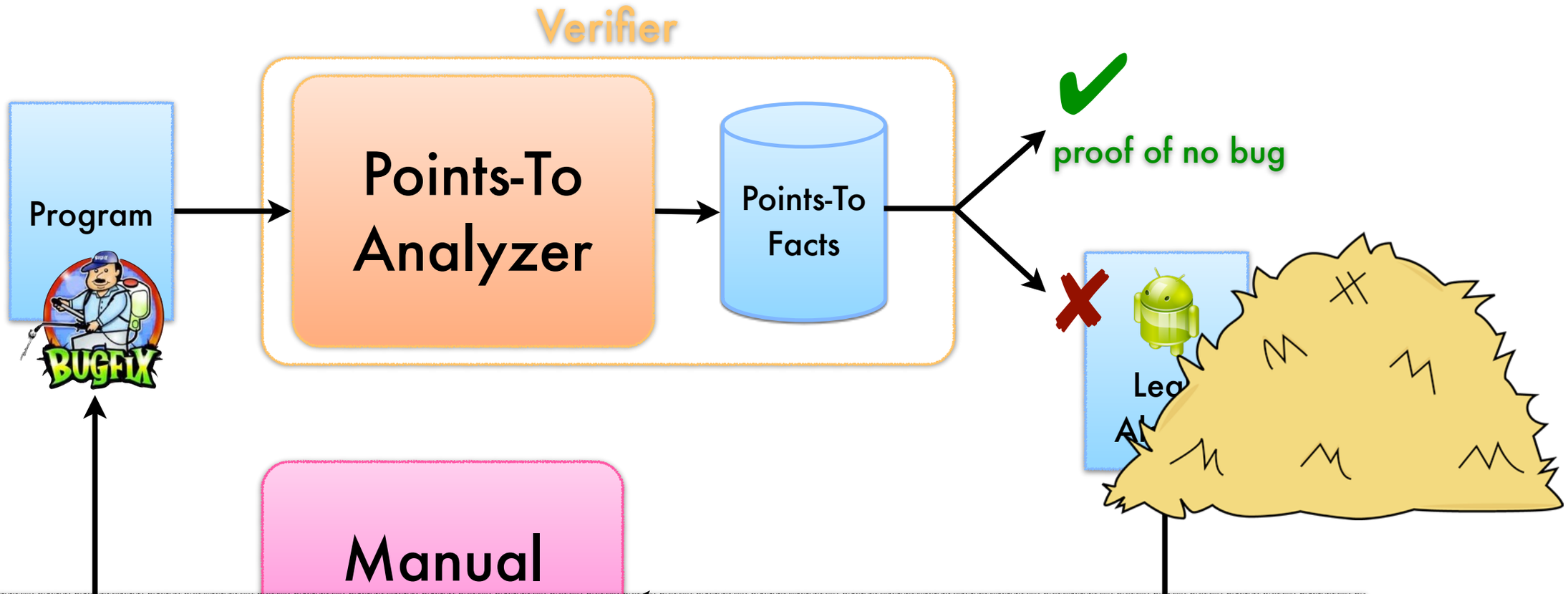


Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?"

- 75 papers, 9 PhD theses

Known: Precise points-to analysis challenging

Thresher addresses alarm **triage** in a particularly challenging domain.

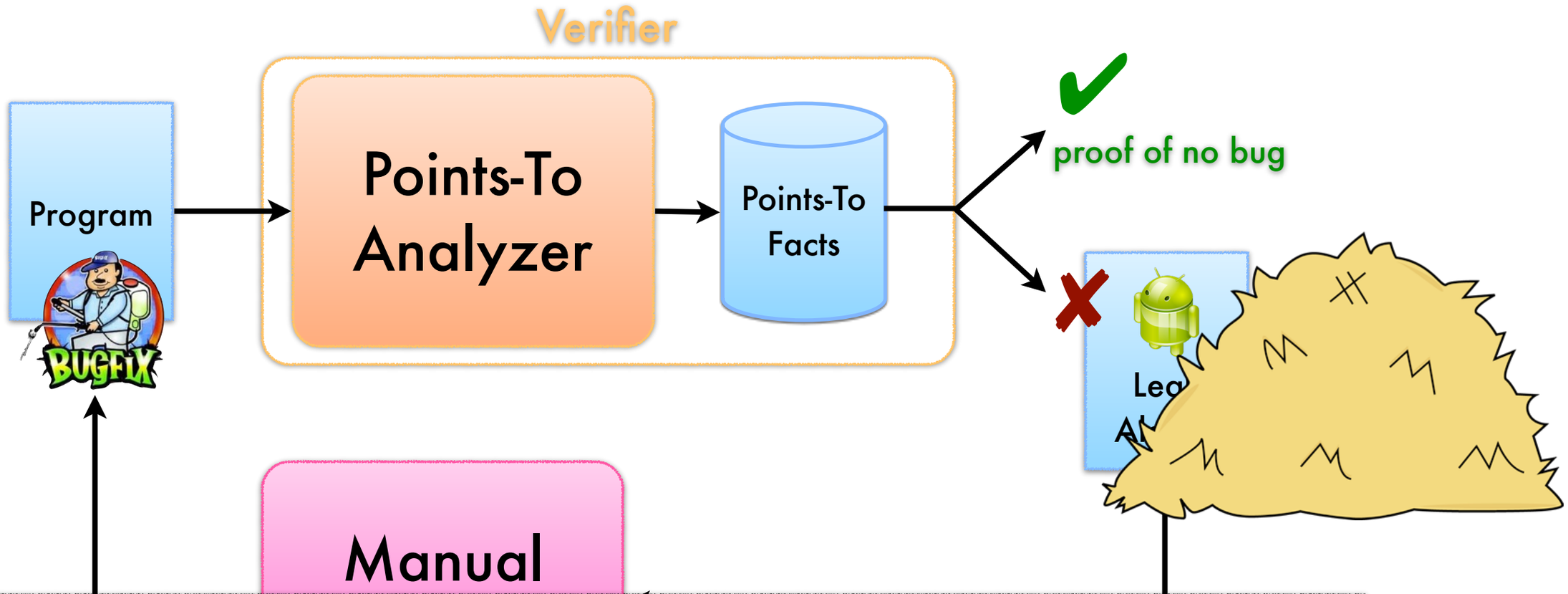


Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?" (2001)

- 75 papers, 9 PhD theses

Known: Precise points-to analysis challenging

Thresher addresses alarm **triage** in a particularly challenging domain.

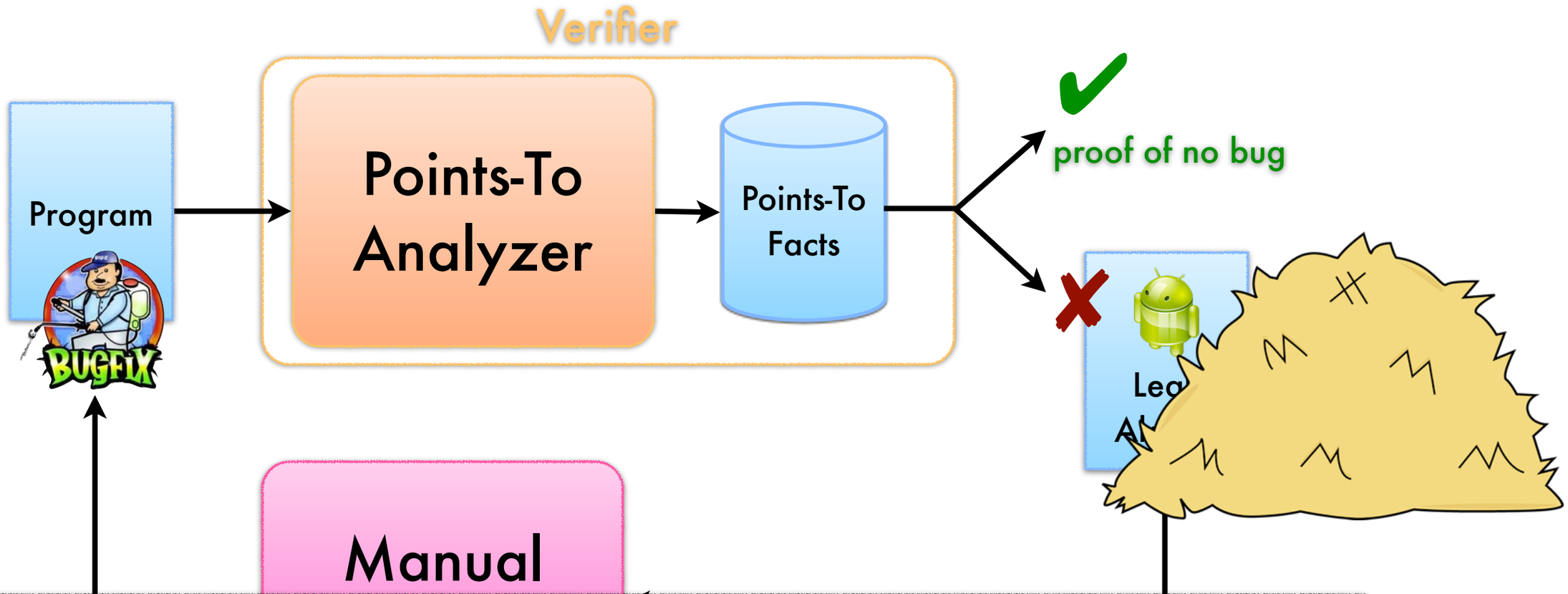


Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?" (2001)

- 75 papers, 9 PhD theses

Known: Precise points-to analysis challenging

Thresher addresses alarm **triage** in a particularly challenging domain.



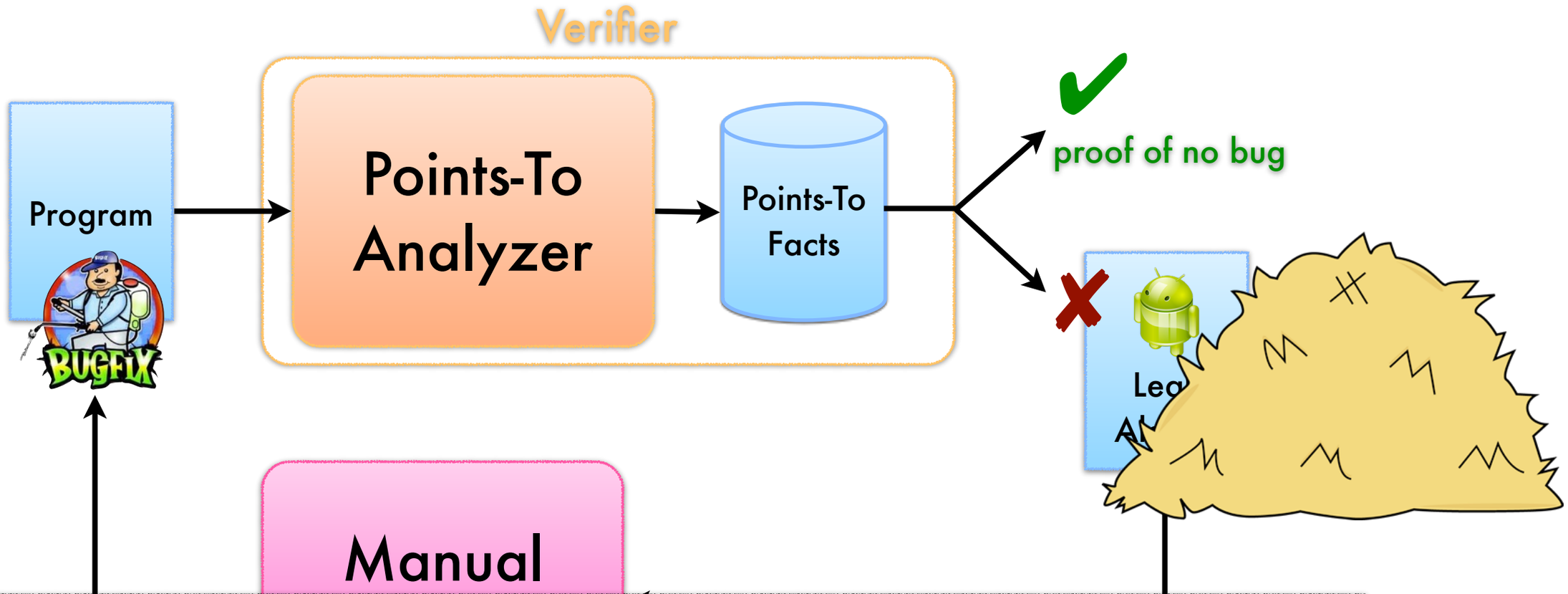
Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?" (2001)

- 75 papers, 9 PhD theses

Dagstuhl 13162: Pointer Analysis (2013)

Known: Precise points-to analysis challenging

Thresher addresses alarm **triage** in a particularly challenging domain.



Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?" (2001)

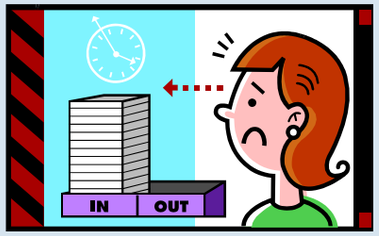
- 75 papers, 9 PhD theses

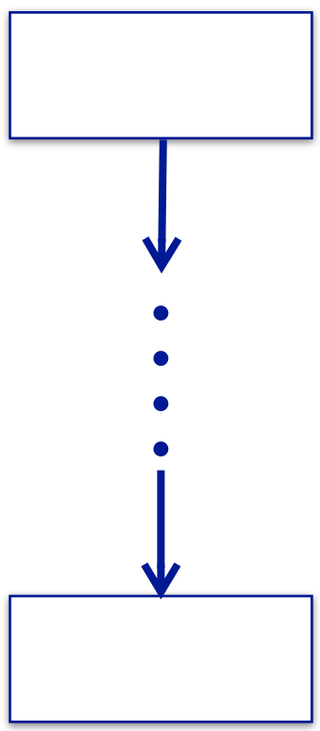
Dagstuhl 13162: Pointer Analysis (2013)

Known: Precise points-to analysis ~~challenging~~

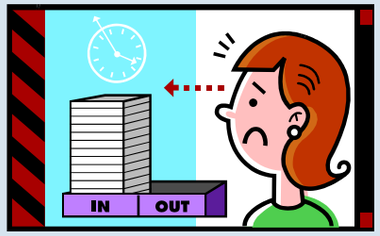
enough impossible?

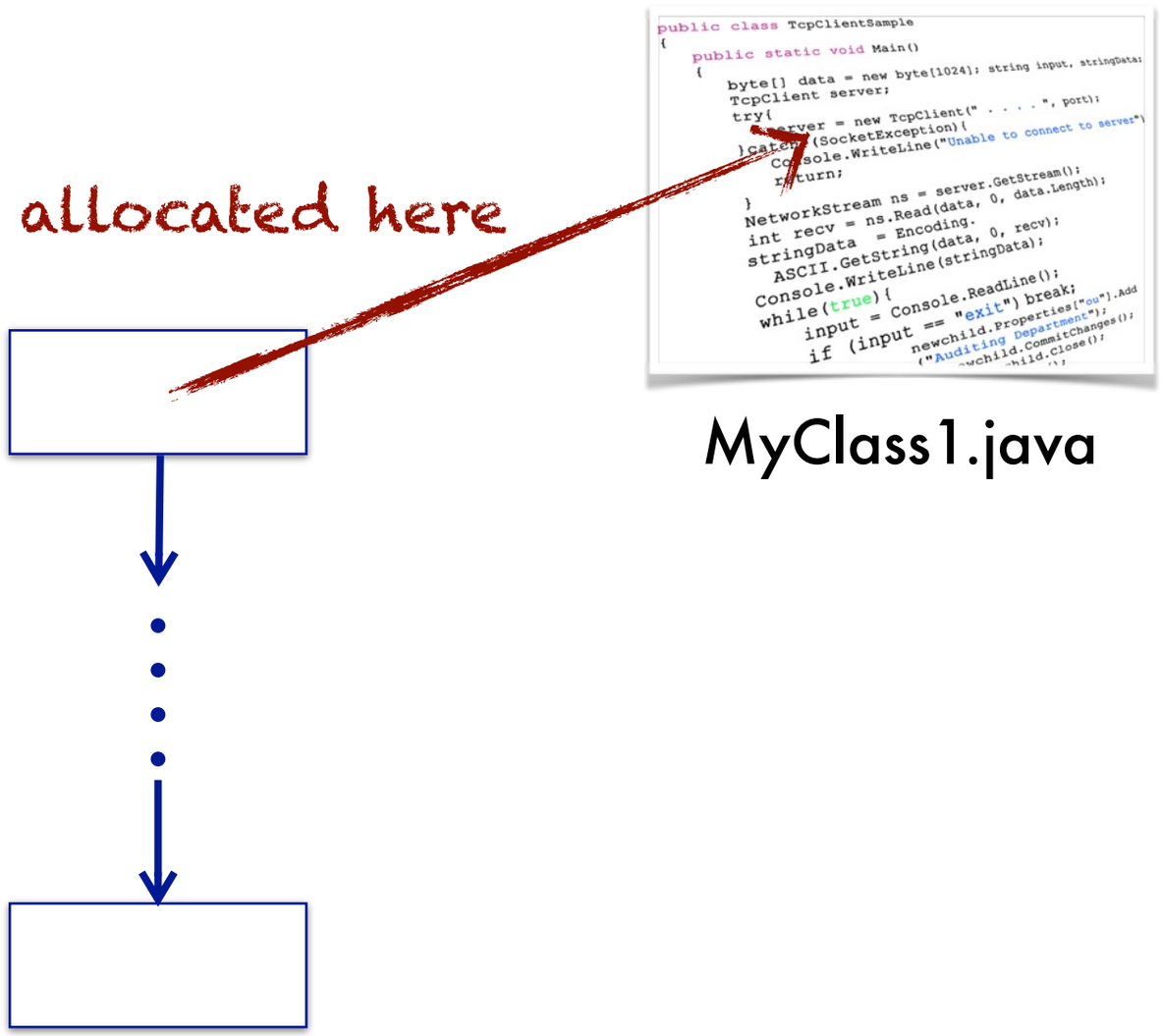
Manual triage is particularly hard for heap reachability reports.



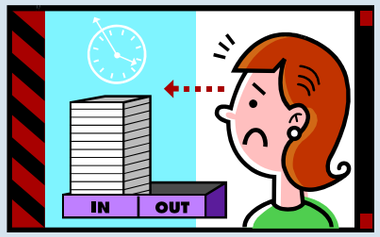


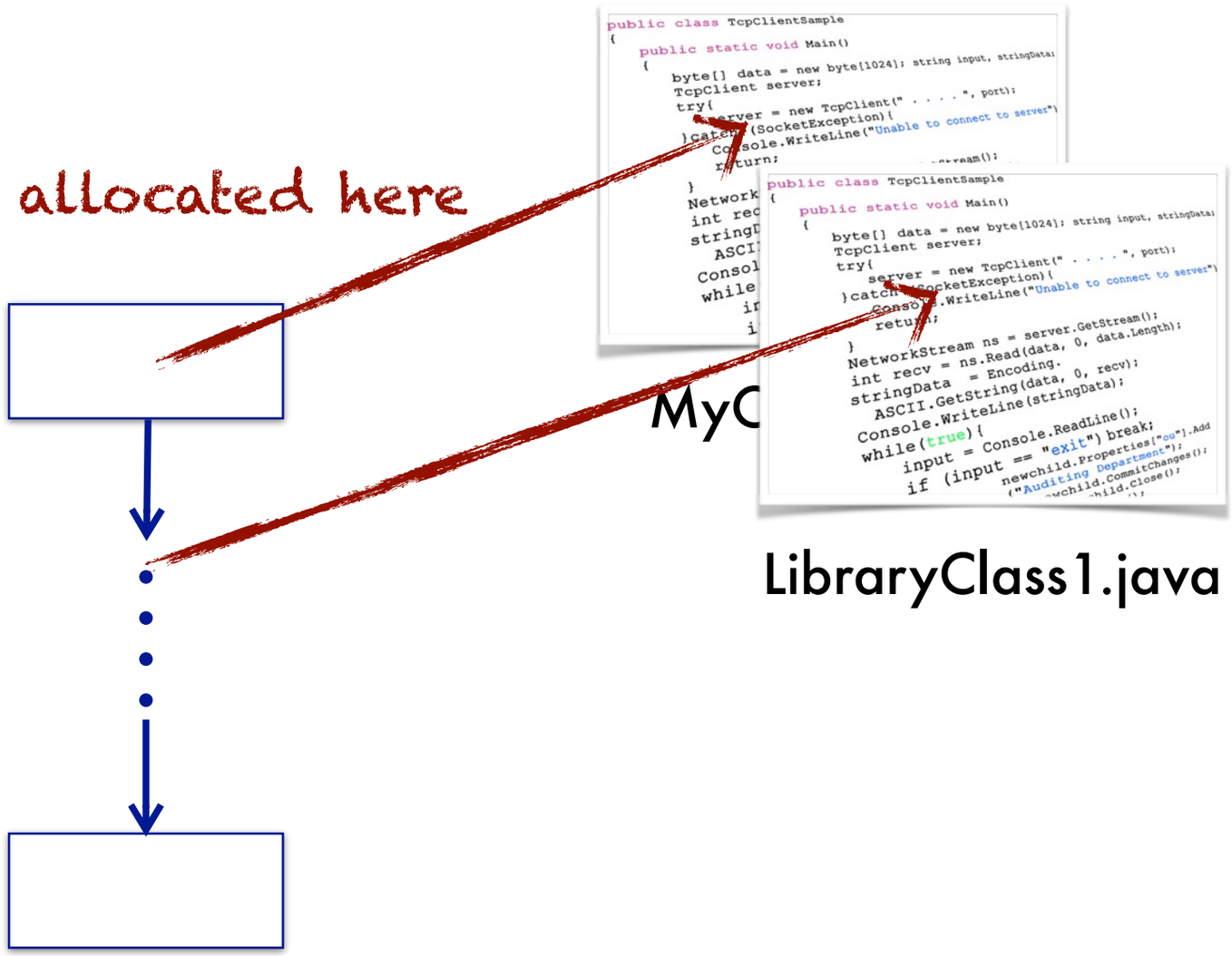
Manual triage is particularly hard for heap reachability reports.



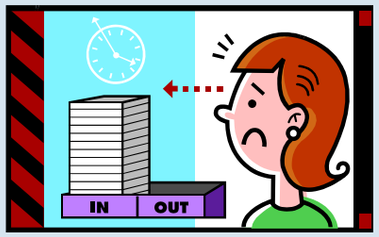


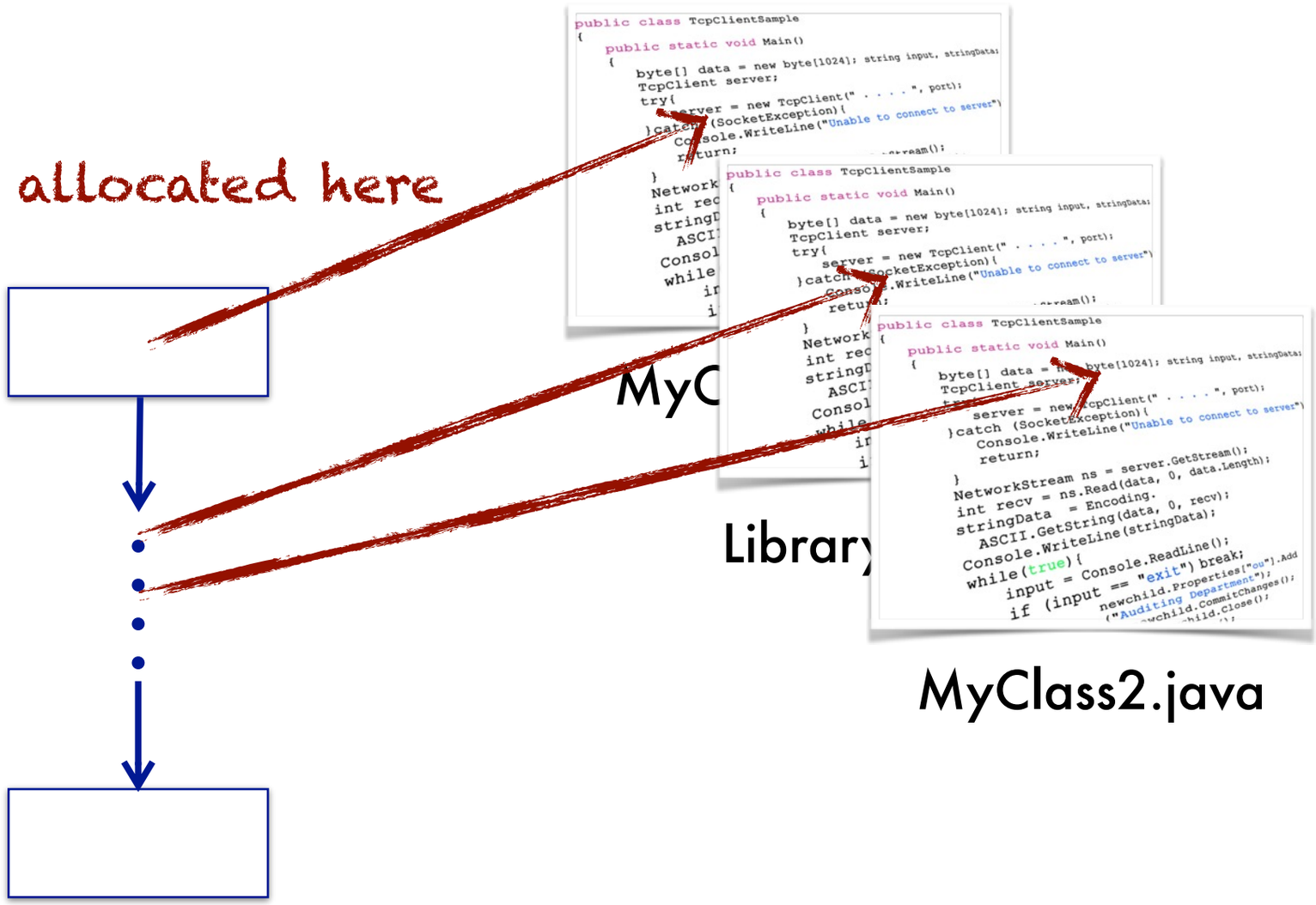
Manual triage is particularly hard for heap reachability reports.



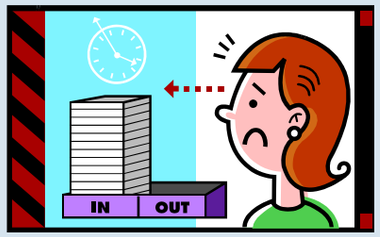


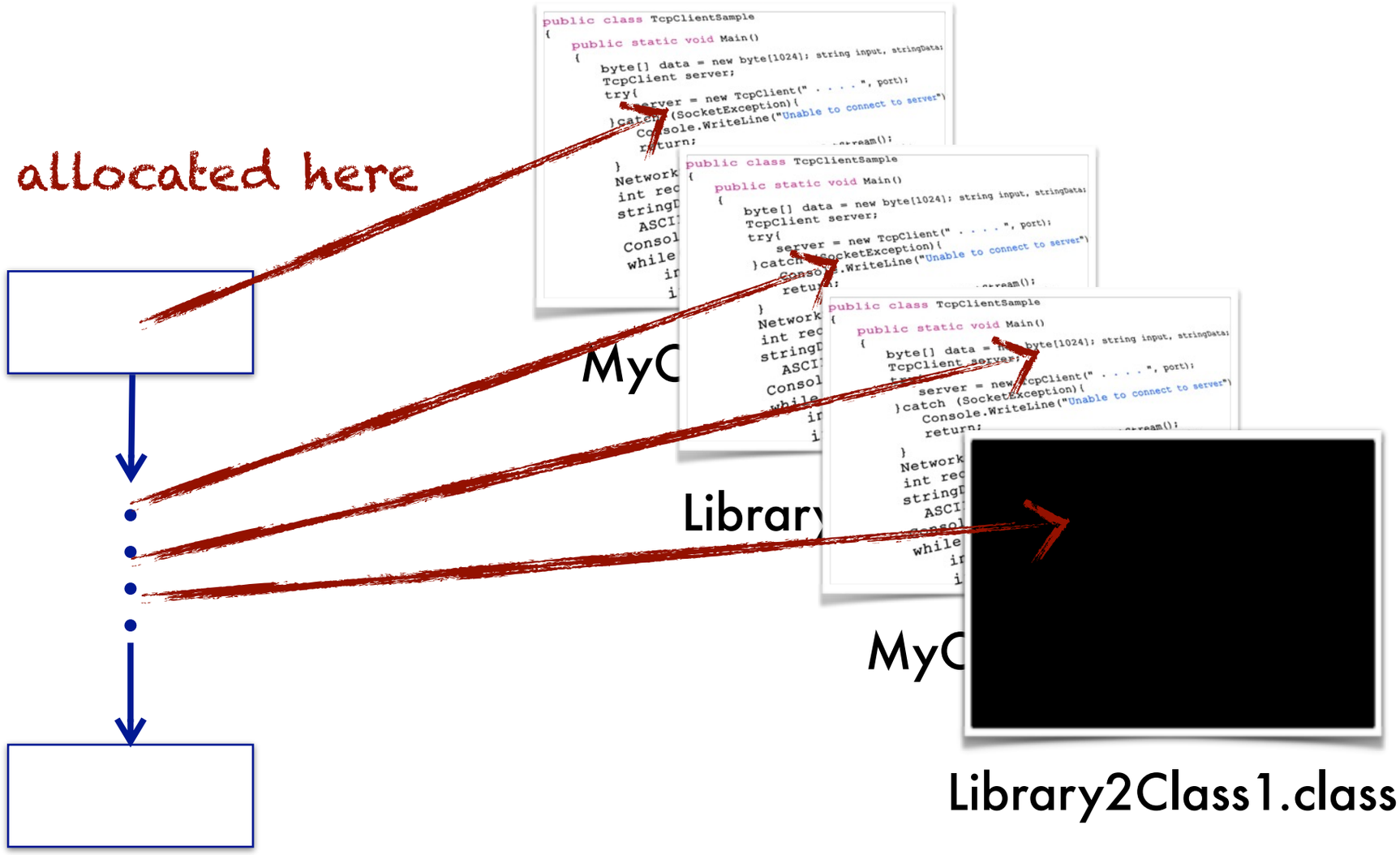
Manual triage is particularly hard for heap reachability reports.



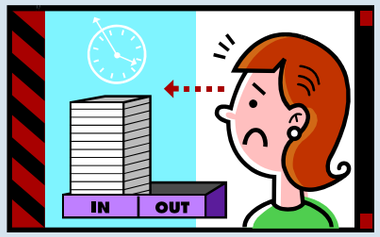


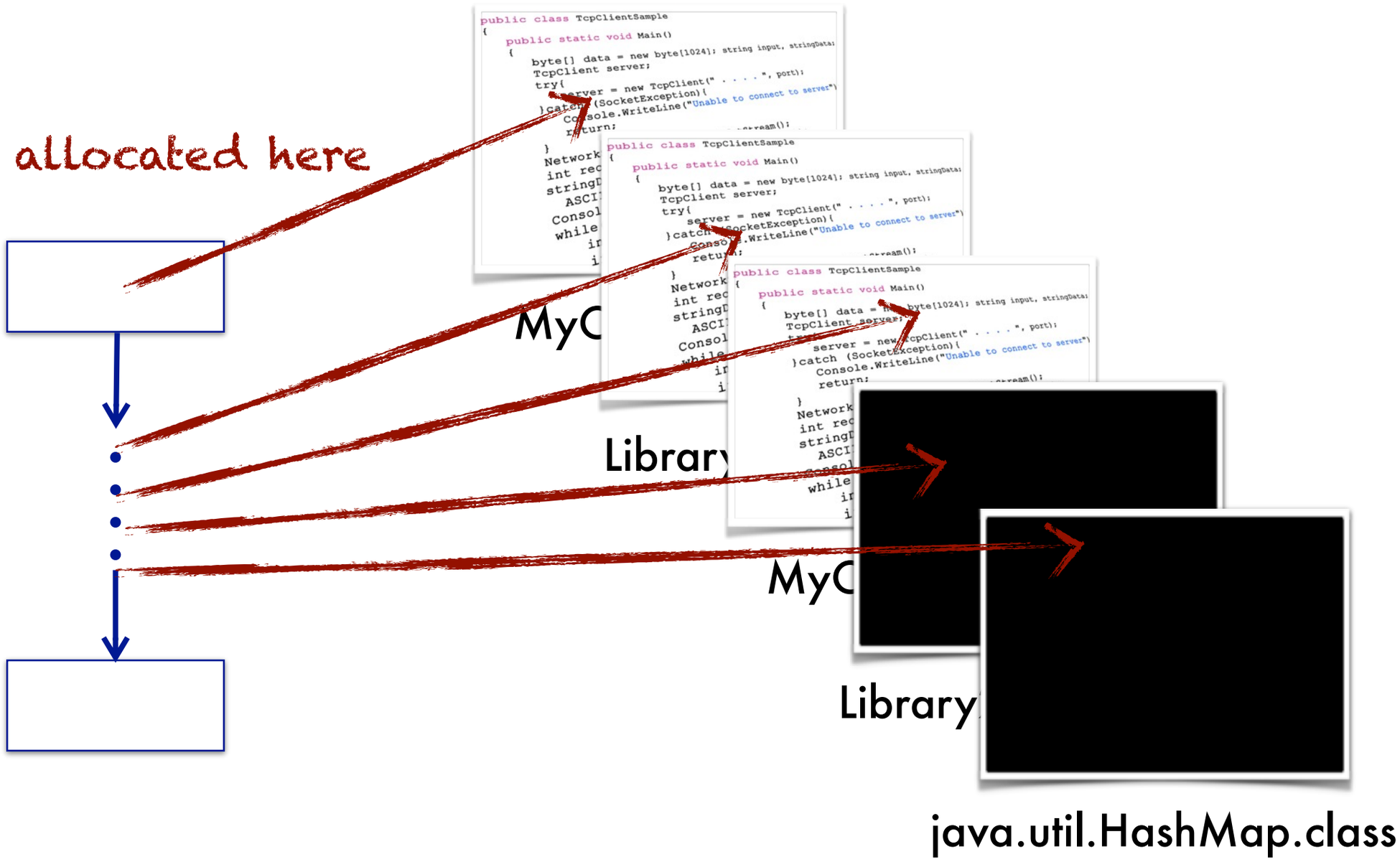
Manual triage is particularly hard for heap reachability reports.



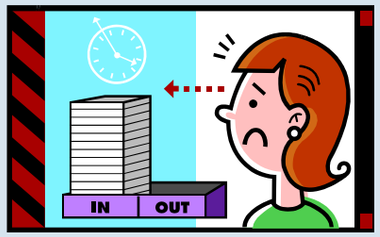


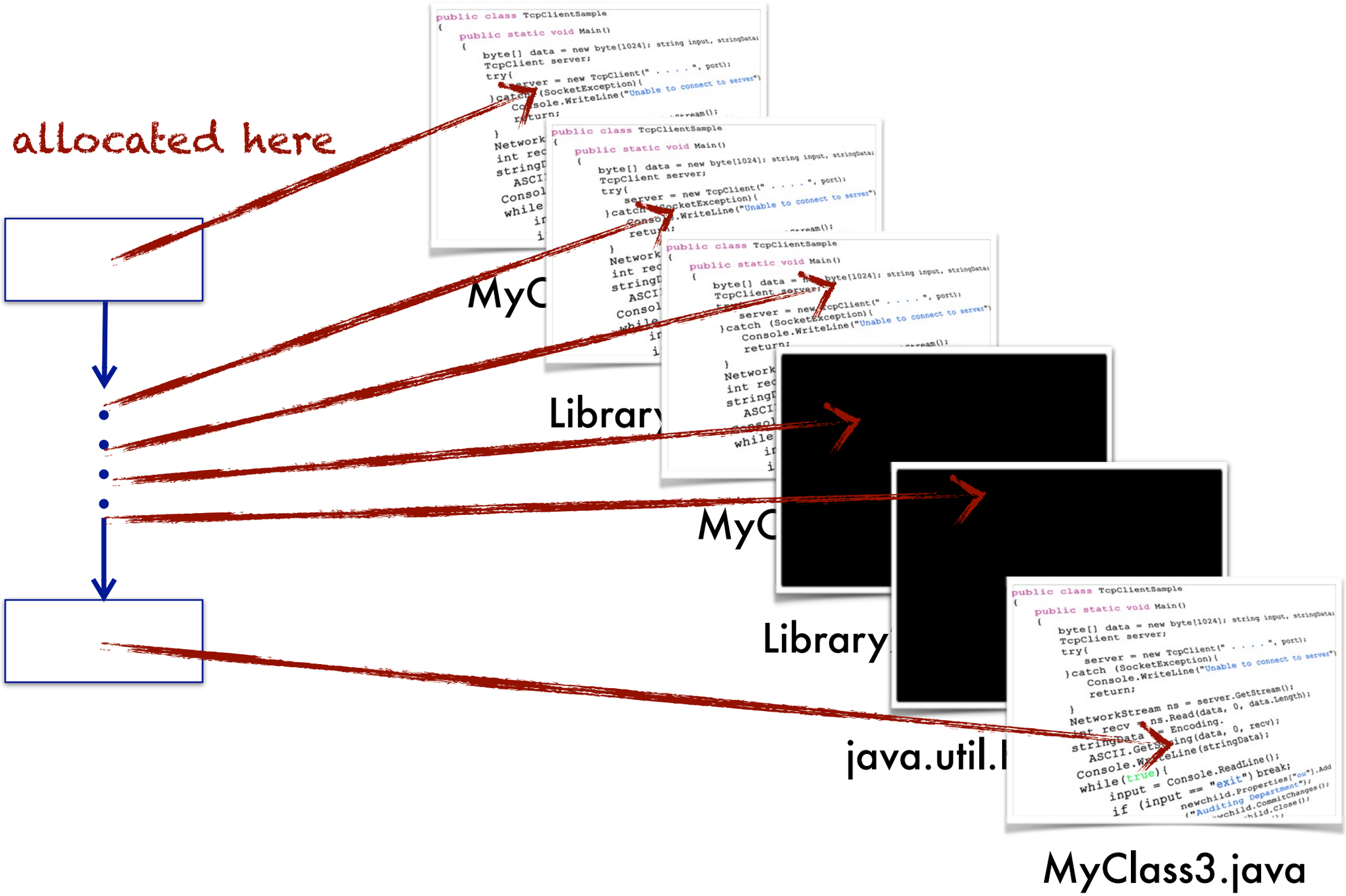
Manual triage is particularly hard for heap reachability reports.



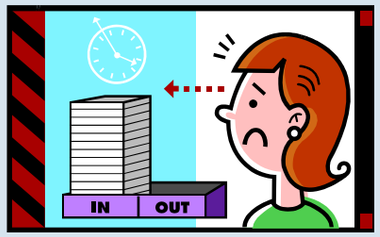


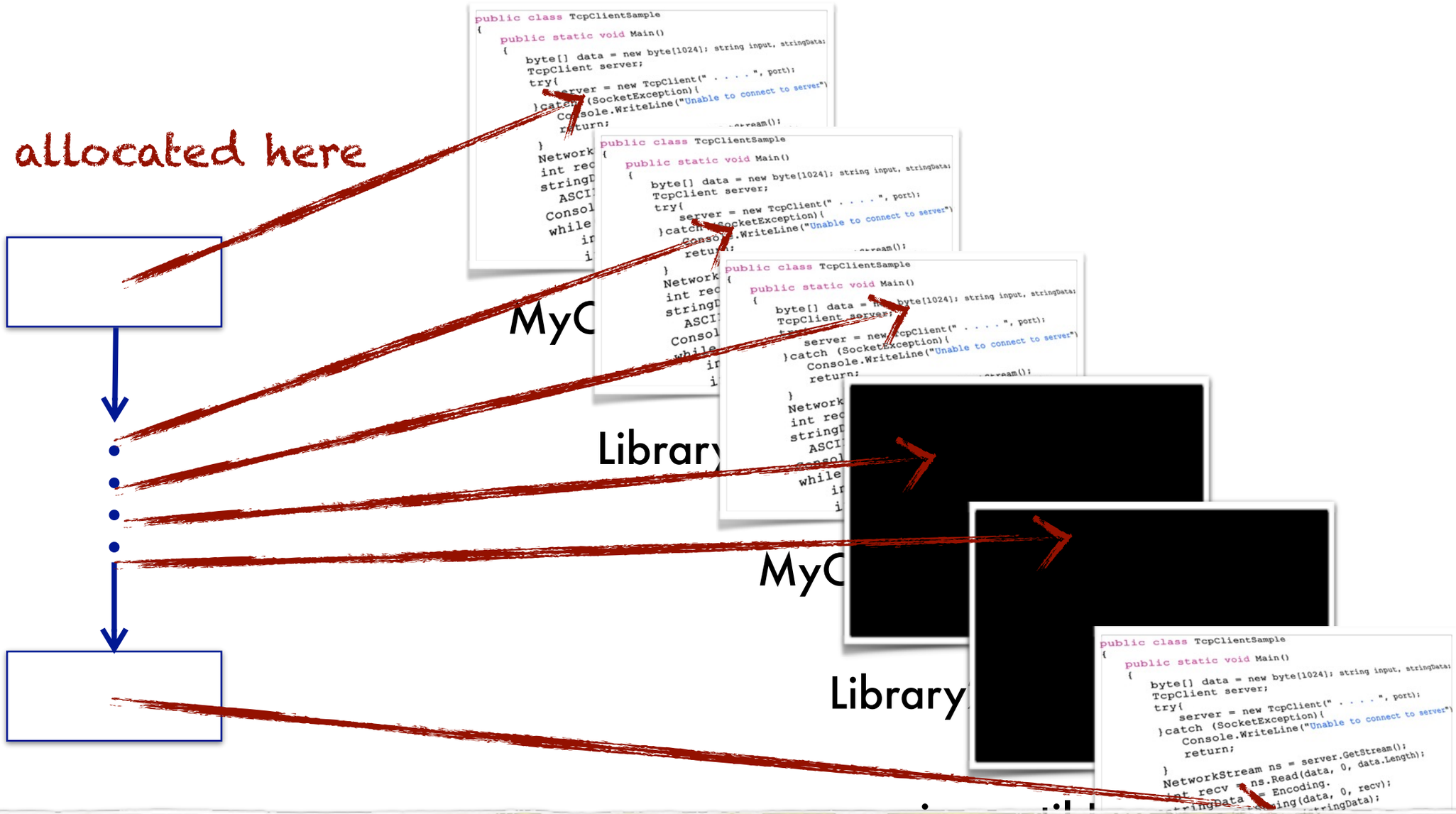
Manual triage is particularly hard for heap reachability reports.





Manual triage is particularly hard for heap reachability reports.

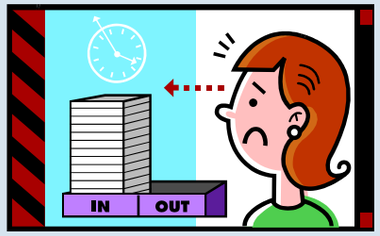




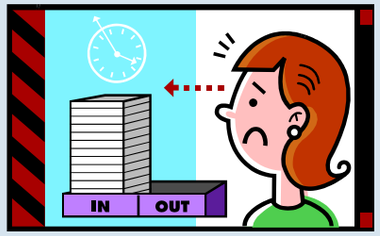
Get abstract heap path + maybe allocation sites

Guesstimate: >1 to 2 hours per alarm to triage "well"

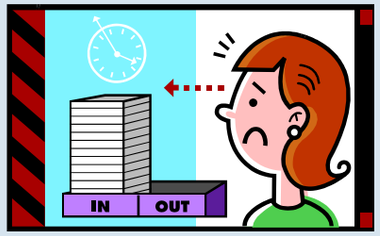
Examining manual triage ...



Examining manual triage ...

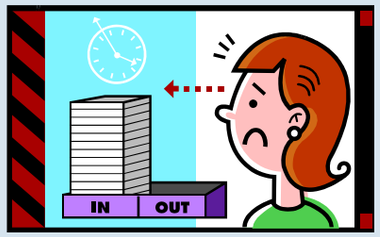


Examining manual triage …



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.

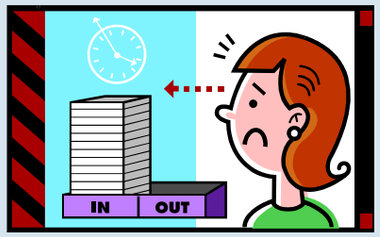
Examining manual triage ...



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.

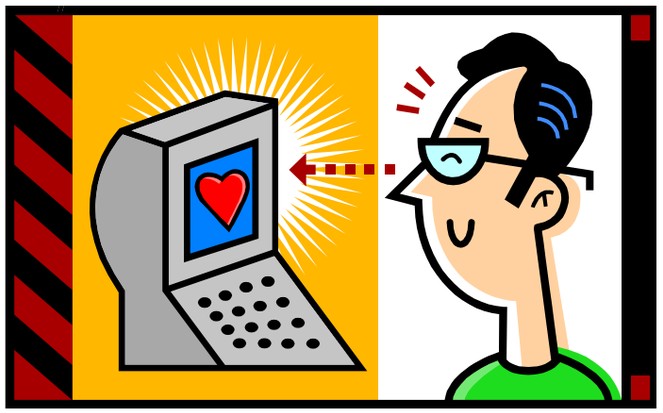
We can do this with analysis!

Examining manual triage ...

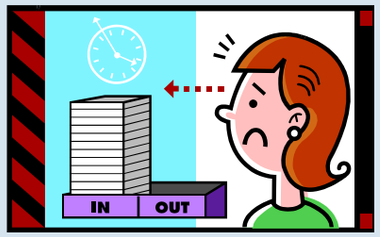


What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what's happening.

We can do this with analysis!

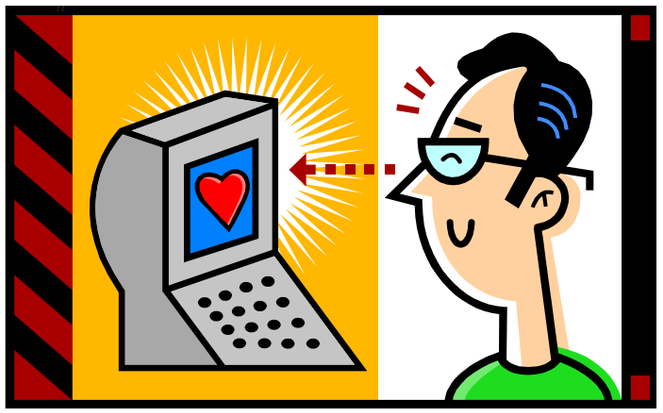


Examining manual triage …



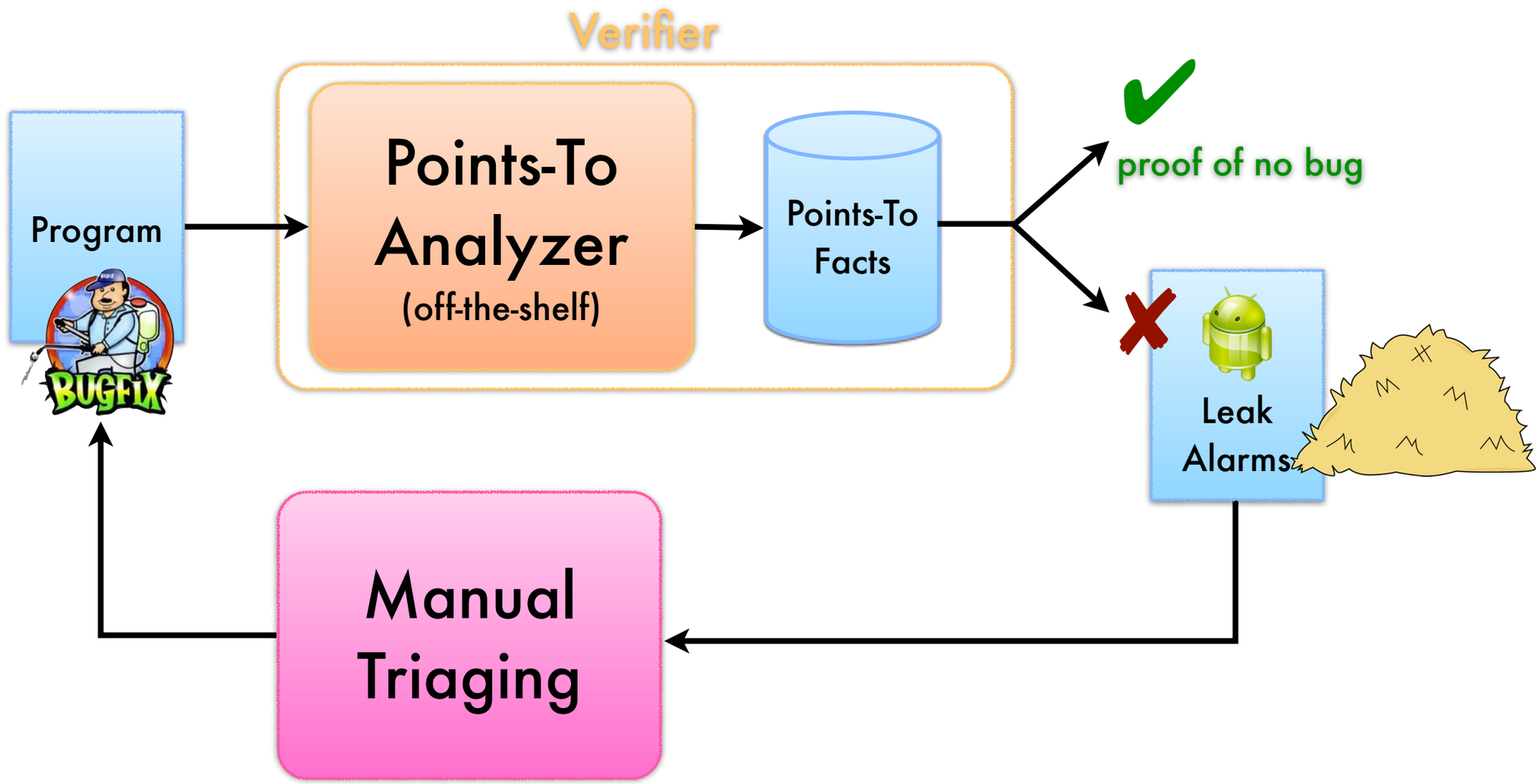
What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what's happening.

We can do this with analysis!

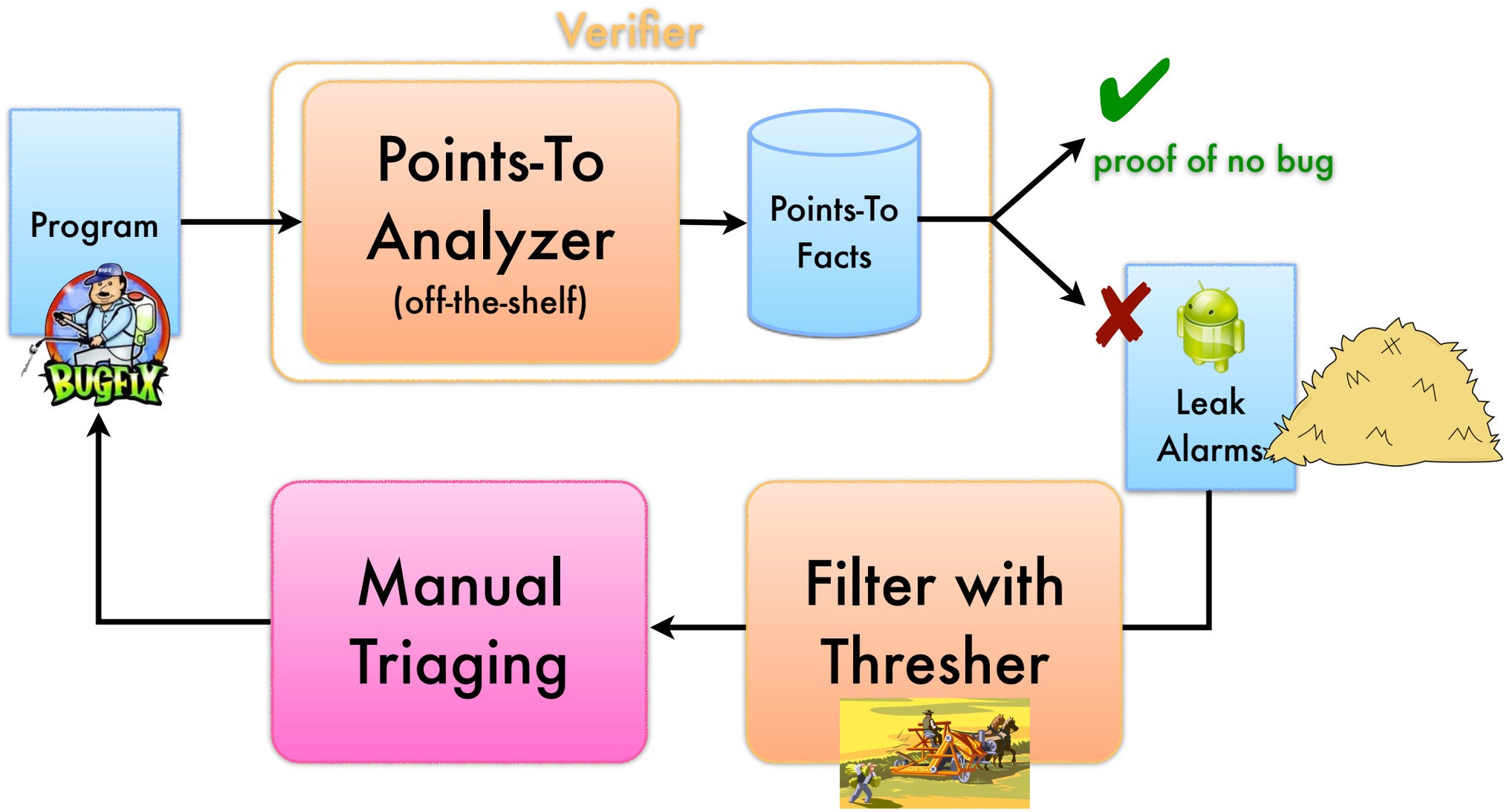


If we filter most false alarms, the user can triage more quickly and get to true bugs earlier (without frustration).

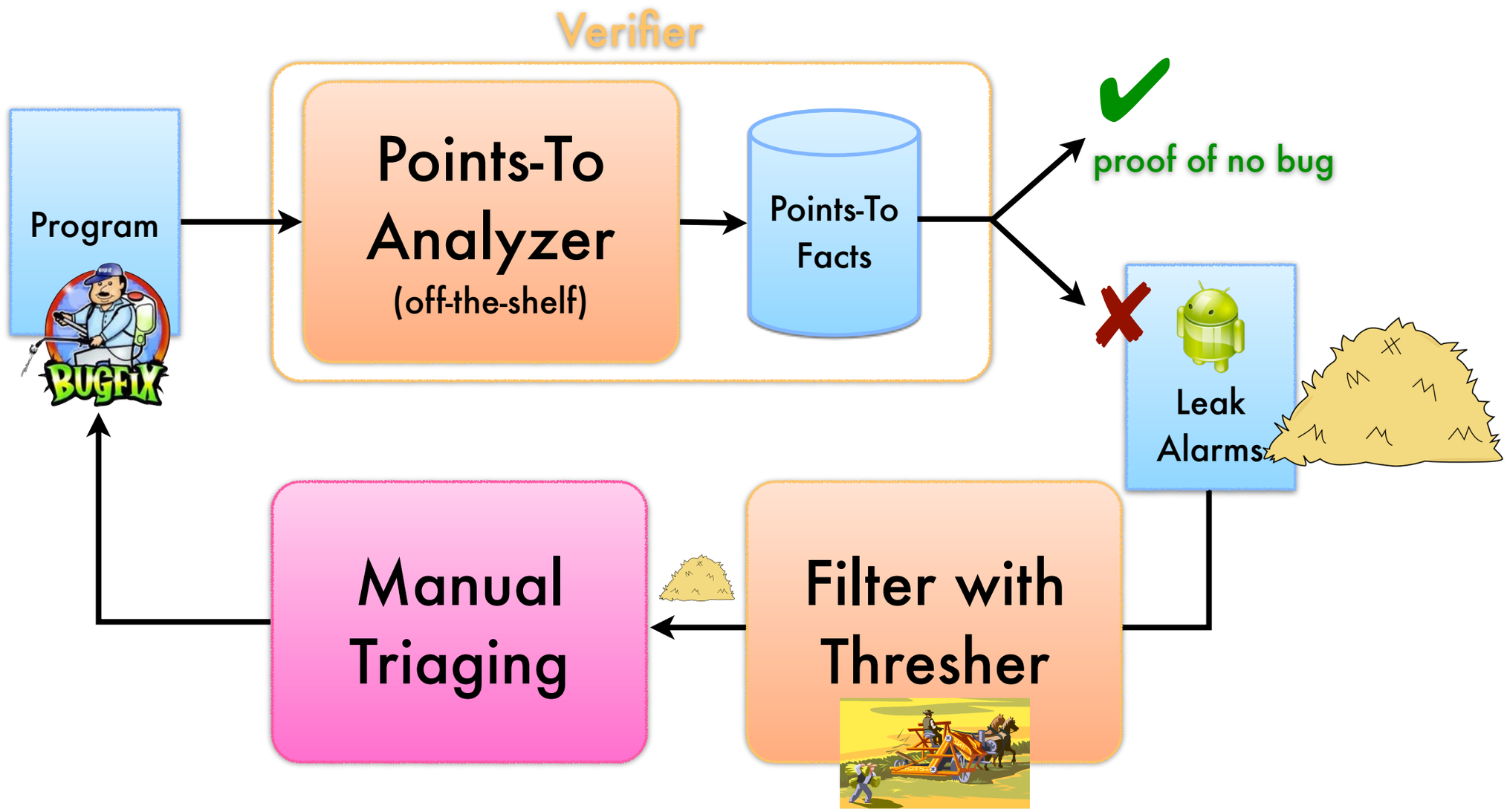
Thresher filters out false alarms by refuting them one-by-one.



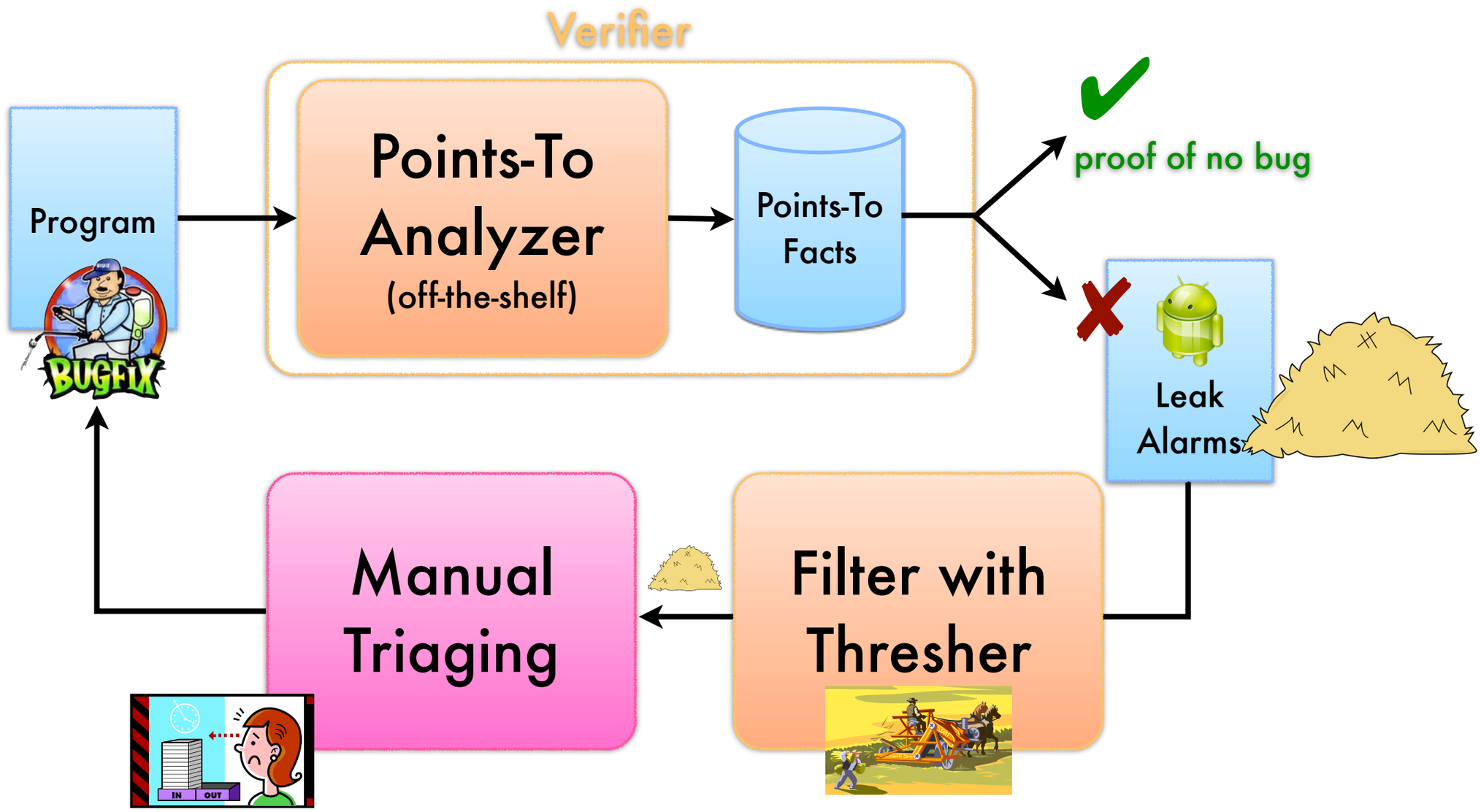
Thresher filters out false alarms by refuting them one-by-one.



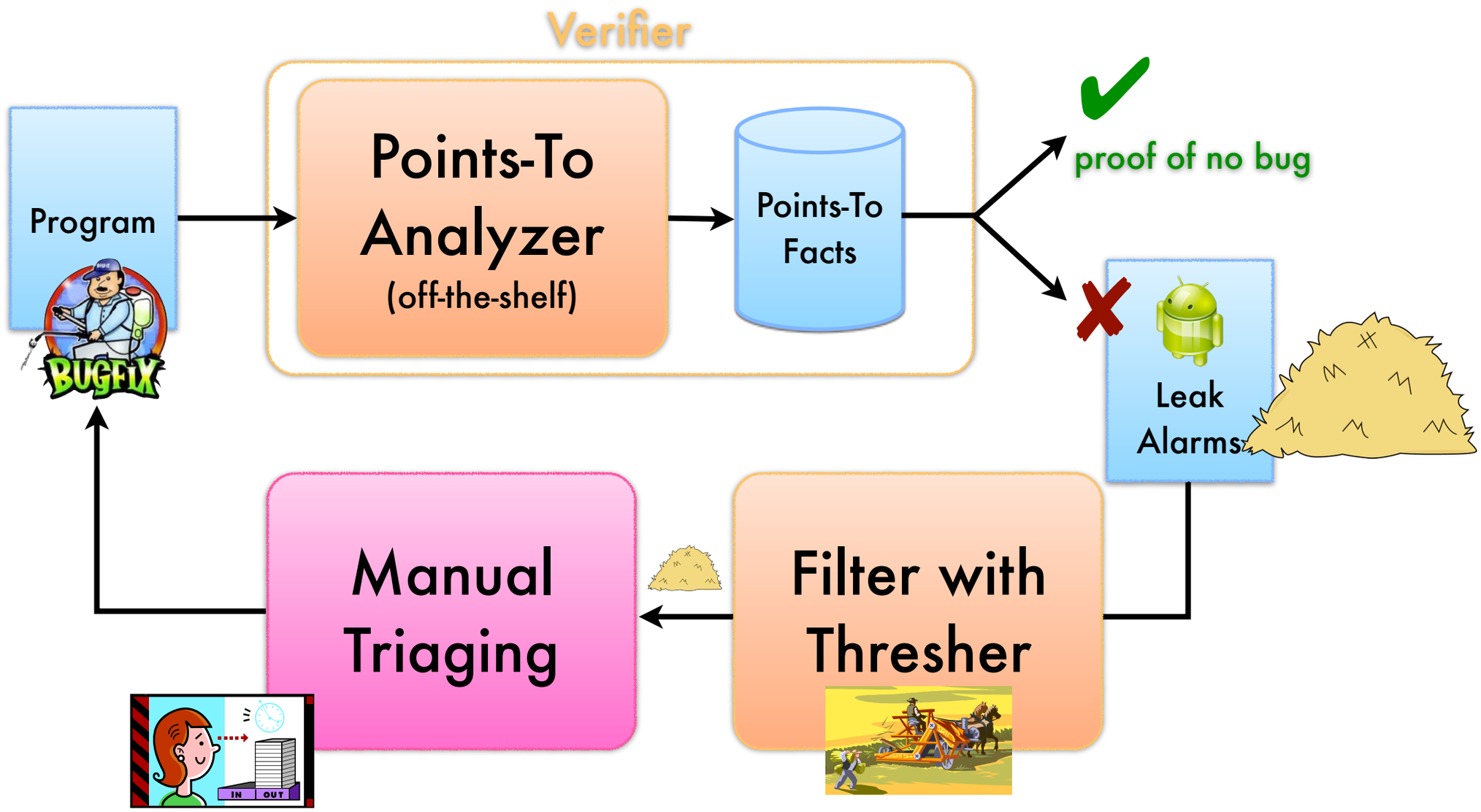
Thresher filters out false alarms by refuting them one-by-one.



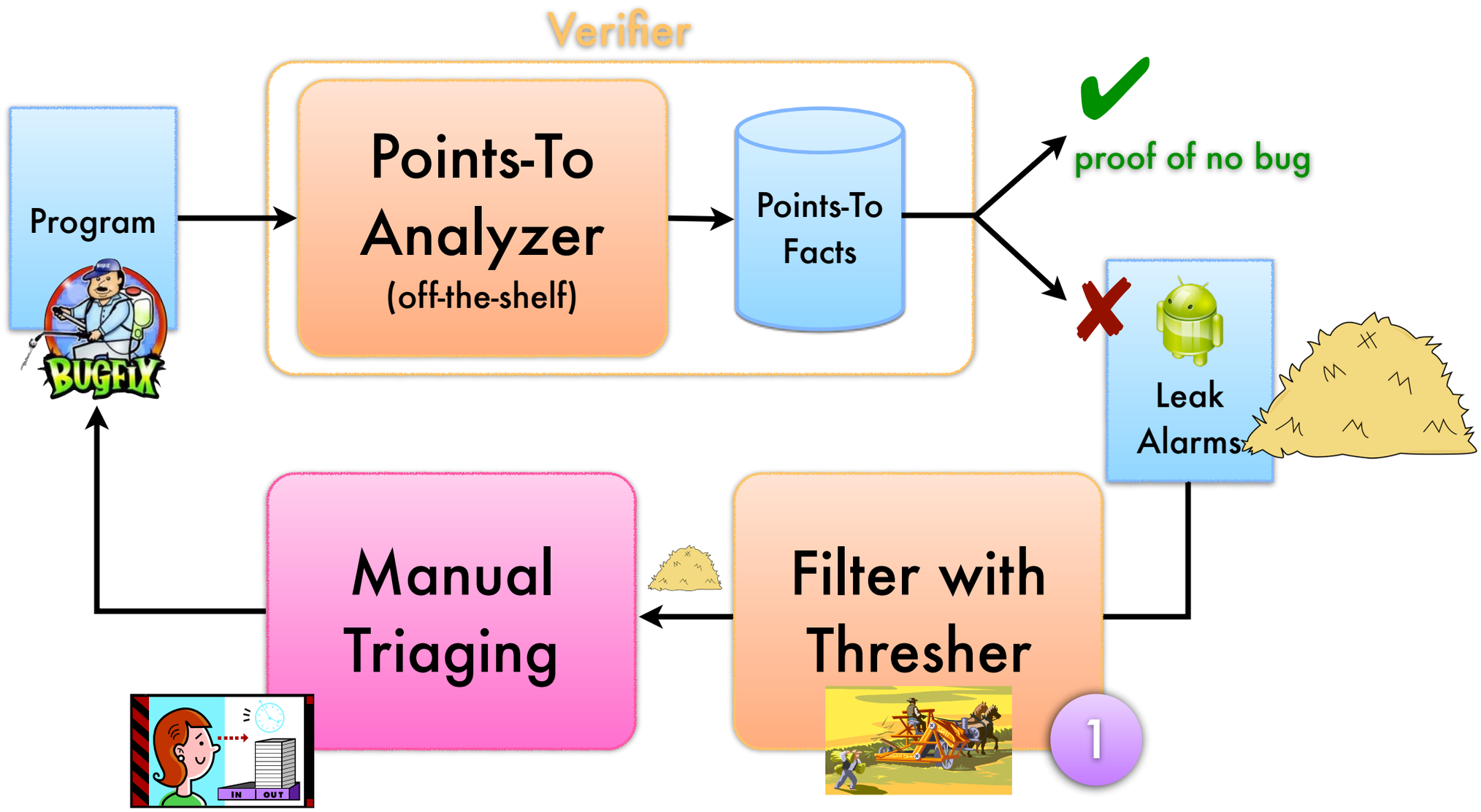
Thresher filters out false alarms by refuting them one-by-one.



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

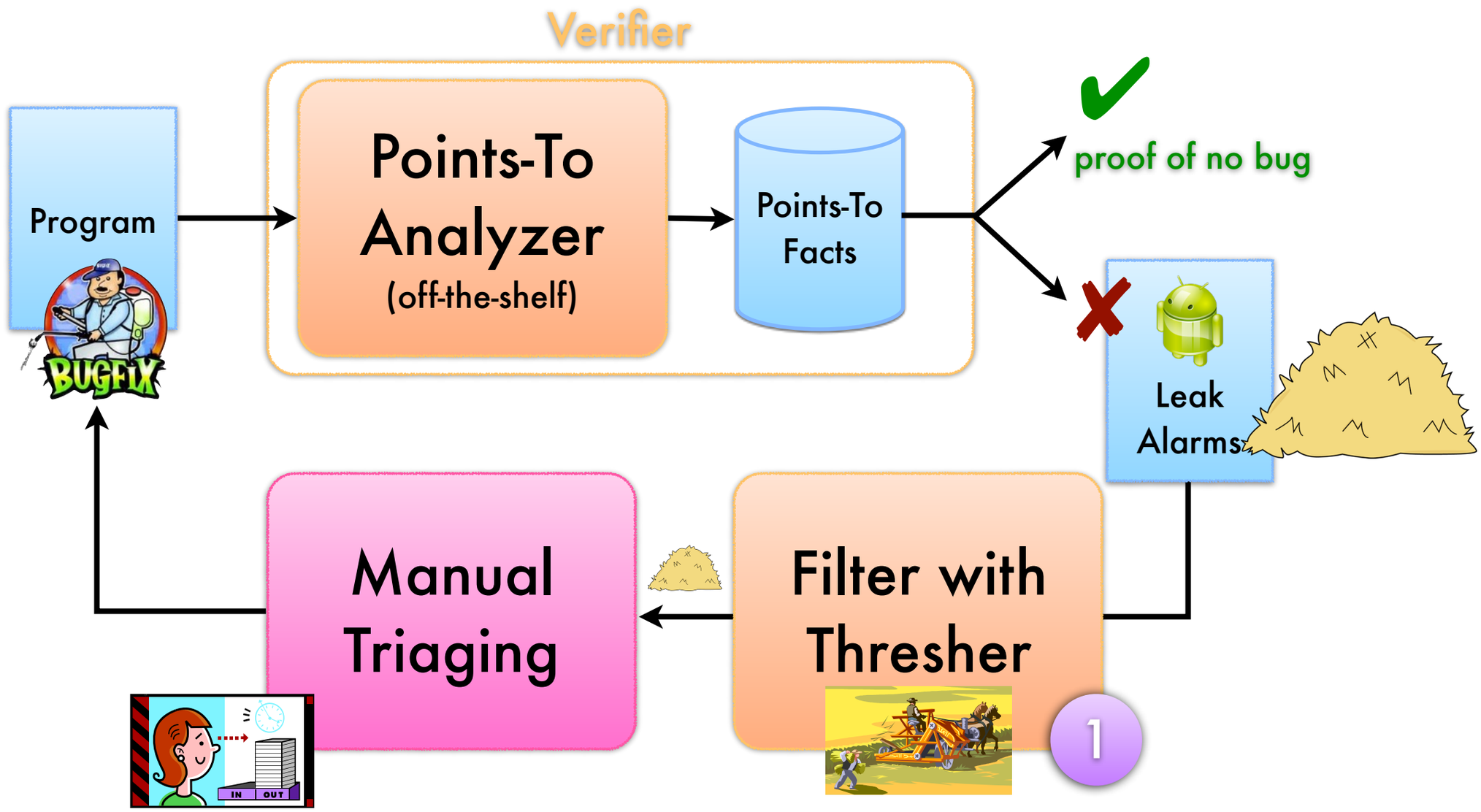


Thresher filters out false alarms by refuting them one-by-one.



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

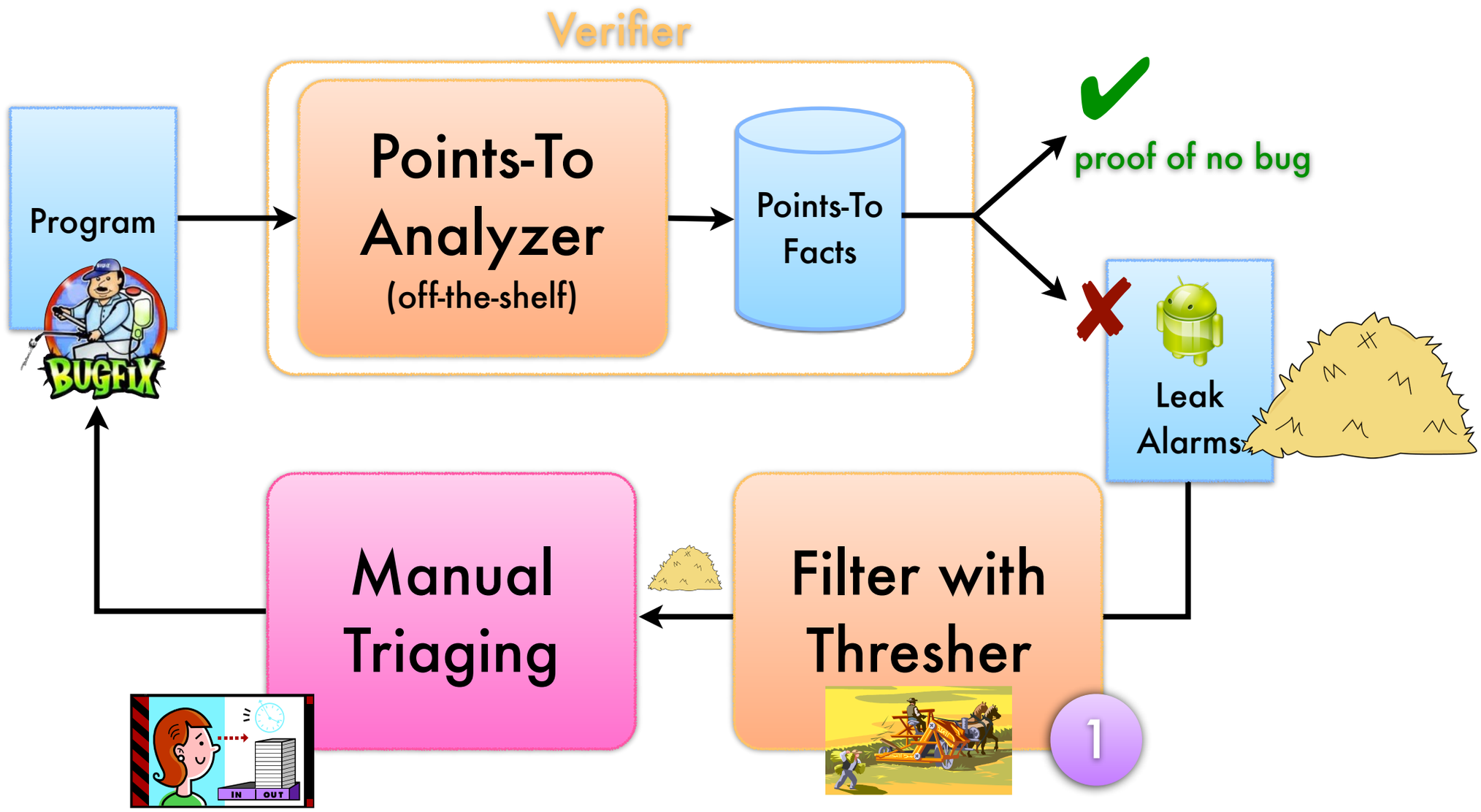
Thresher filters out false alarms by refuting them one-by-one.



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

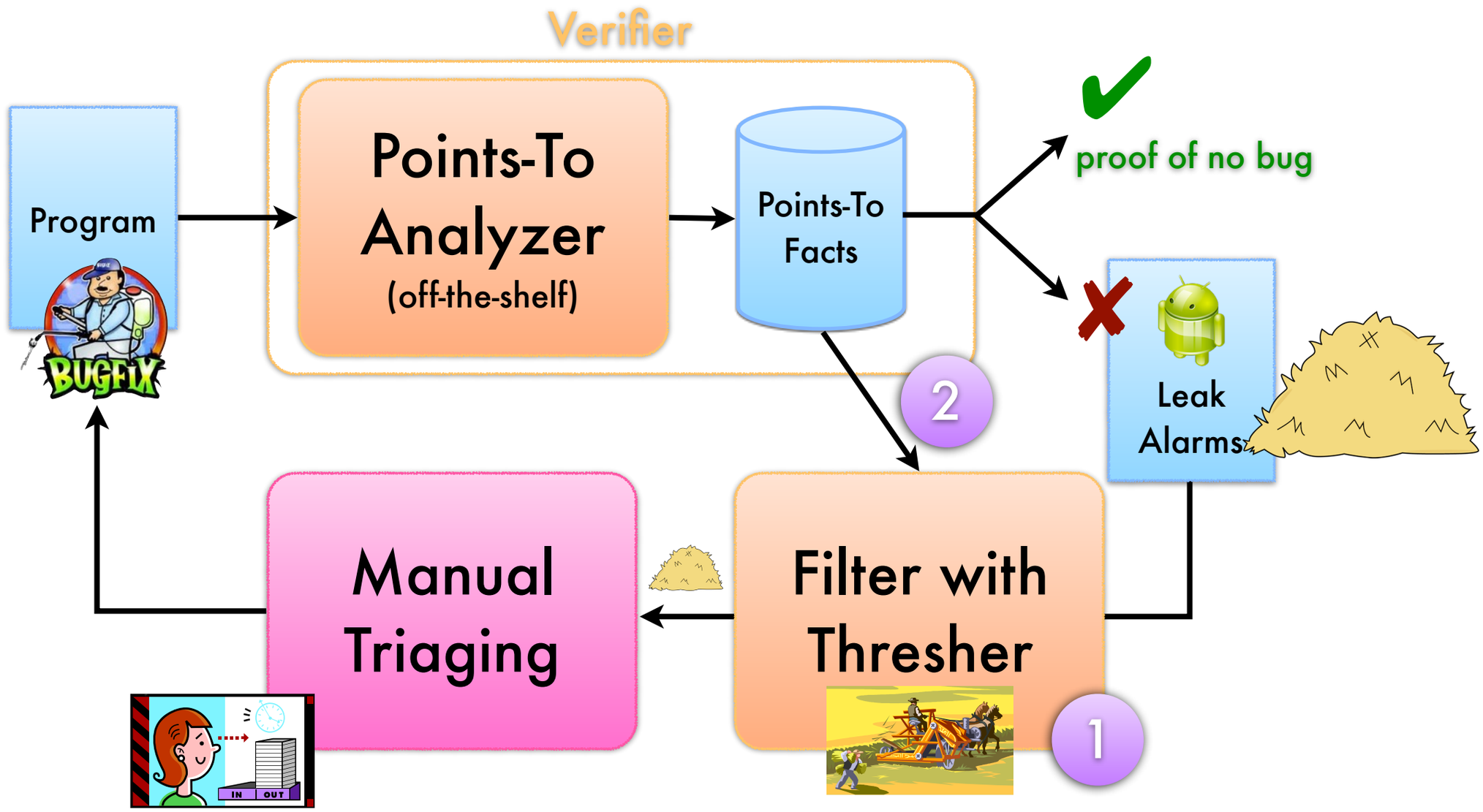
*-sensitive

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

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
PulsePoint	unknown	16
StandupTimer	2K	25
DroidLife	3K	3
SMSPopUp	7K	5
aMetro	20K	54
K9Mail	40K	208
Total	**72K**	**311**

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms
PulsePoint	unknown	16
StandupTimer	2K	25
DroidLife	3K	3
SMSPopUp	7K	5
aMetro	20K	54
K9Mail	40K	208
Total	**72K**	**311**

`staticfield-Activity` **pairs**

Is Thresher effective at filtering?

Program	LOC	Points-To Alarms
PulsePoint	unknown	16
StandupTimer	2K	25
DroidLife	3K	3
SMSPopUp	7K	5
aMetro		54
K9Mail		208
Total		**311**

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

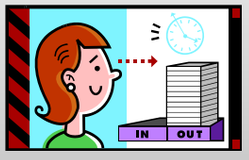
Is Thresher effective at filtering?

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?

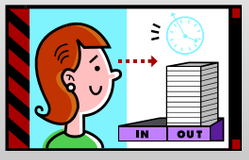
Program	LOC	Points-To Alarms	Thresher Refuted	True Bugs
PulsePoint	unknown	16	8	8
StandupTimer	2K	25	15	0
DroidLife	3K	3	0	3
SMSPopUp	7K	5	1	4
aMetro	20K			36
K9Mail	40K			64
Total	**72K**			**115**

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

		Alarms	Reported	Bugs	Time (s)	Alarm %	red %
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**

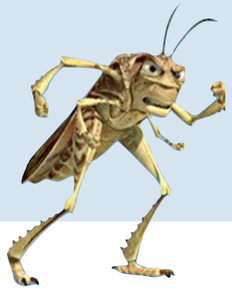
False alarms down to 17% from 63% (points-to analysis only)

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 ...

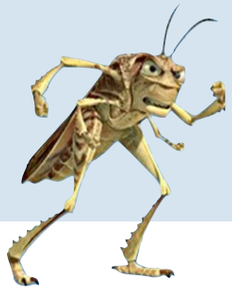


```
class HashMap {
  static Object[] EMPTY = new Object[2]; ...
  HashMap() { this.tbl = EMPTY; capacity initially empty }

  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 ...

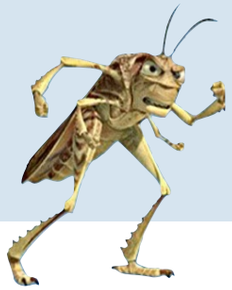


```
class HashMap {
  static Object[] EMPTY = new Object[2]; ...
  HashMap() { this.tbl = EMPTY; capacity initially empty }

  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 ...



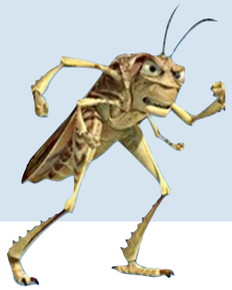
null object pattern: should not be written to

```
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 `HashMap`s 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;
  }

  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

return 0

return "evil" content

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

Find the Android's HashMap bug ...

What if you store **passwords** in a `HashMap`?

```
cla
  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 0

return "evil" content

An "evil" implementation of the `Map` interface can corrupt `EMPTY`. Then, all `HashMap`s 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/

```
cla
  s
  H

  v

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

return 0

return "evil" content

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

Contribution: Addressed the false alarm problem with

a "smart and precise filter"

a goal-directed refutation analysis

Another more challenging reason for ...

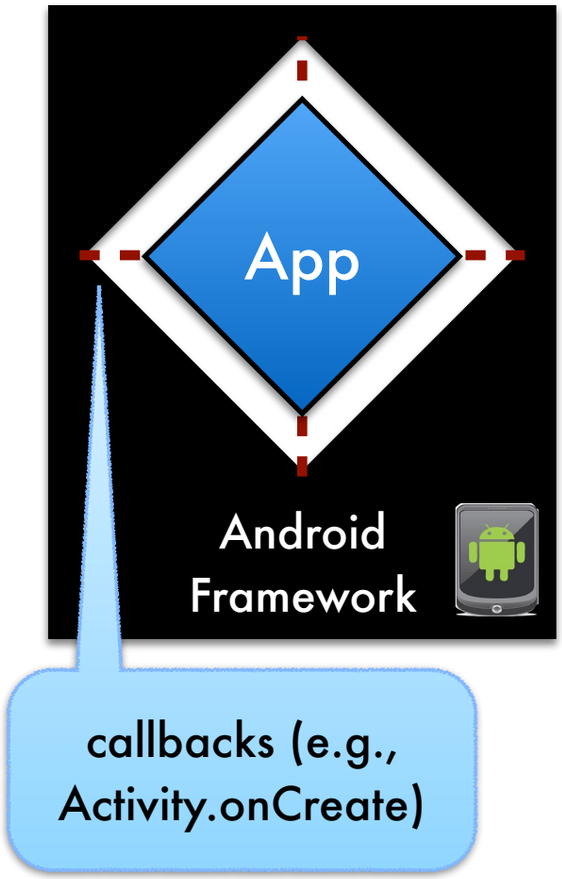
Another more challenging reason for ...

Roughly, 3% of all commits fix NullPointerExceptions

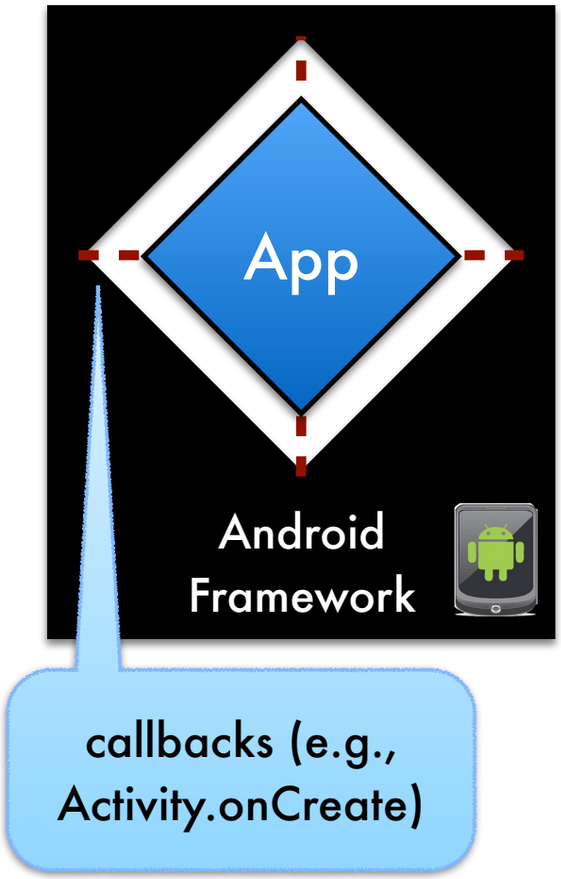
Callback-oriented programming ...

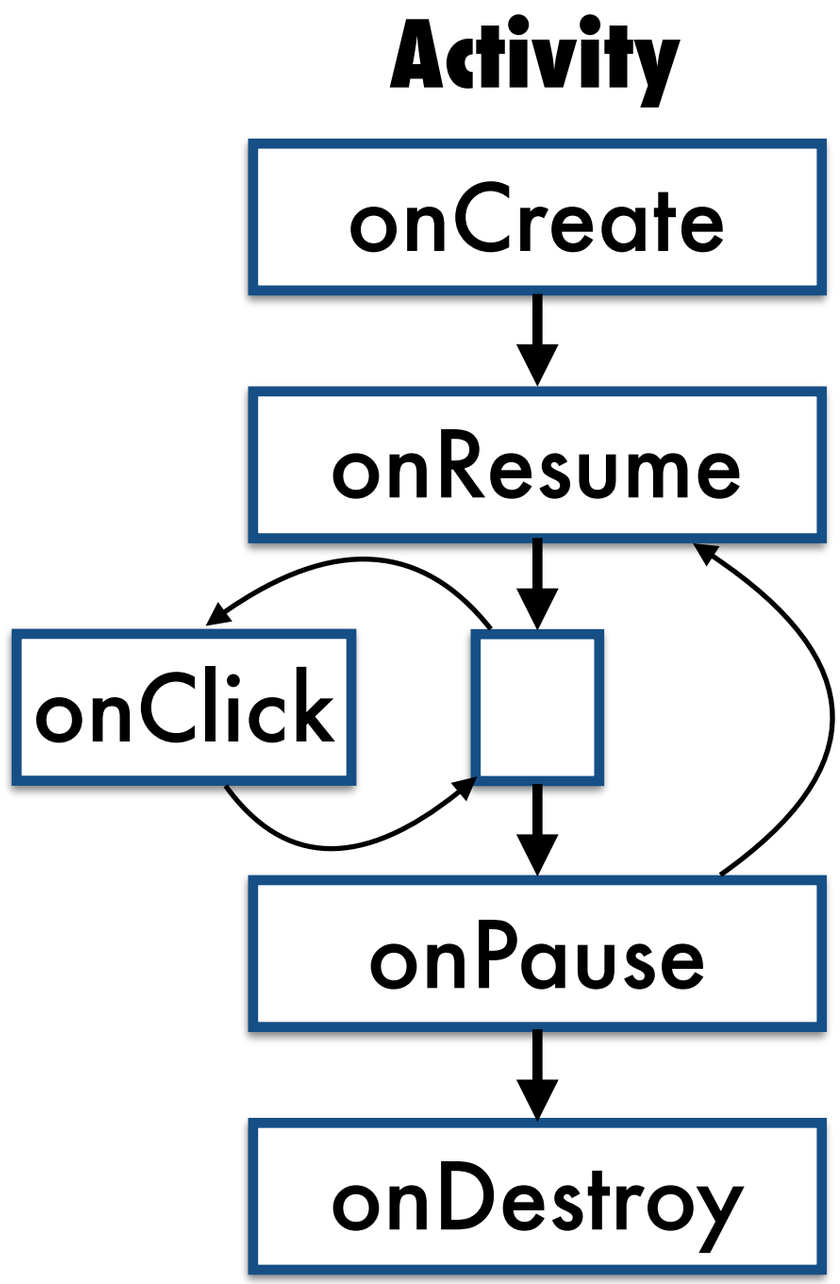
Callback-oriented programming ...

Callback-oriented programming …

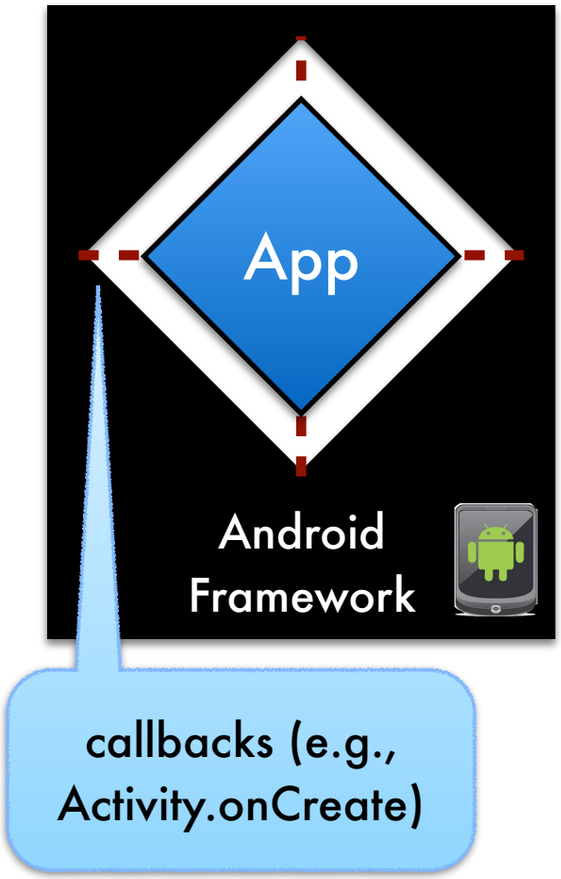


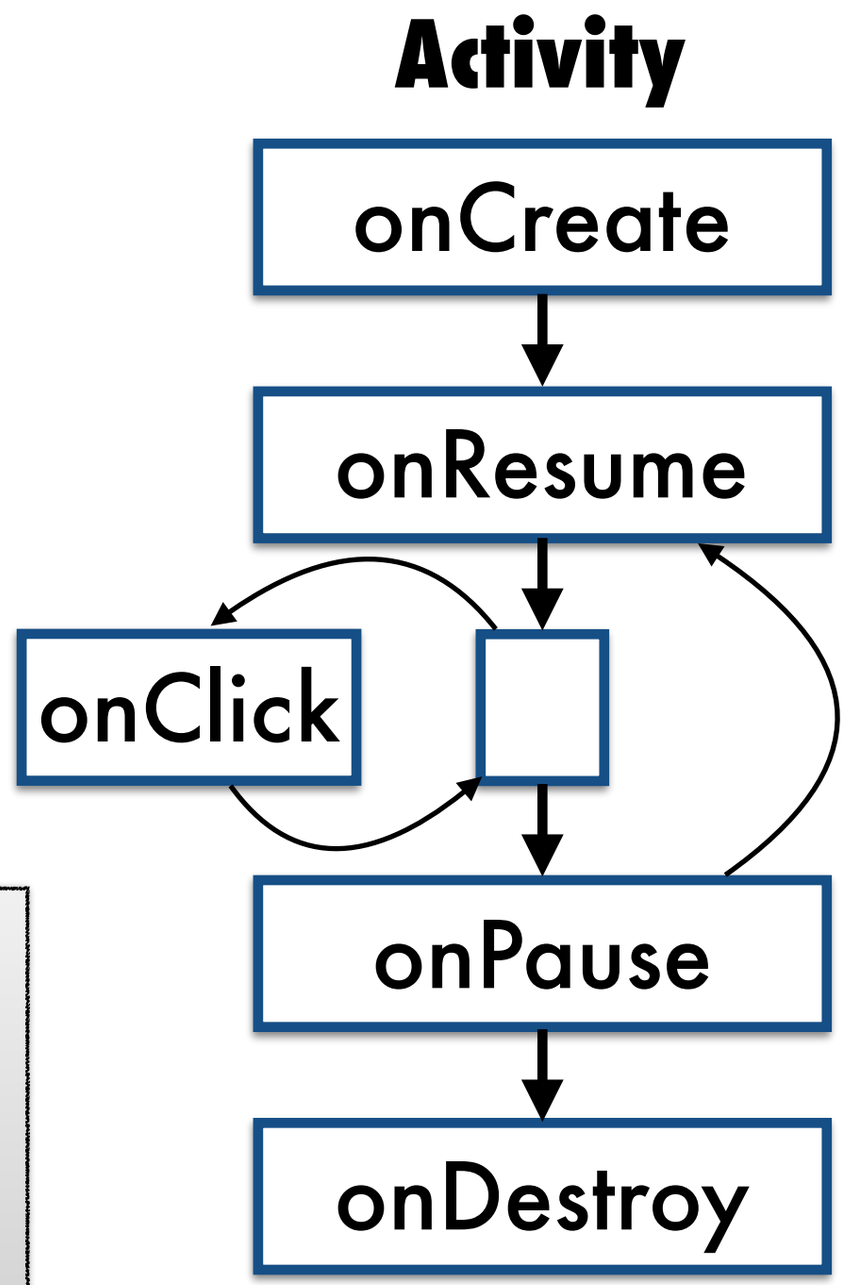
Callback-oriented programming …





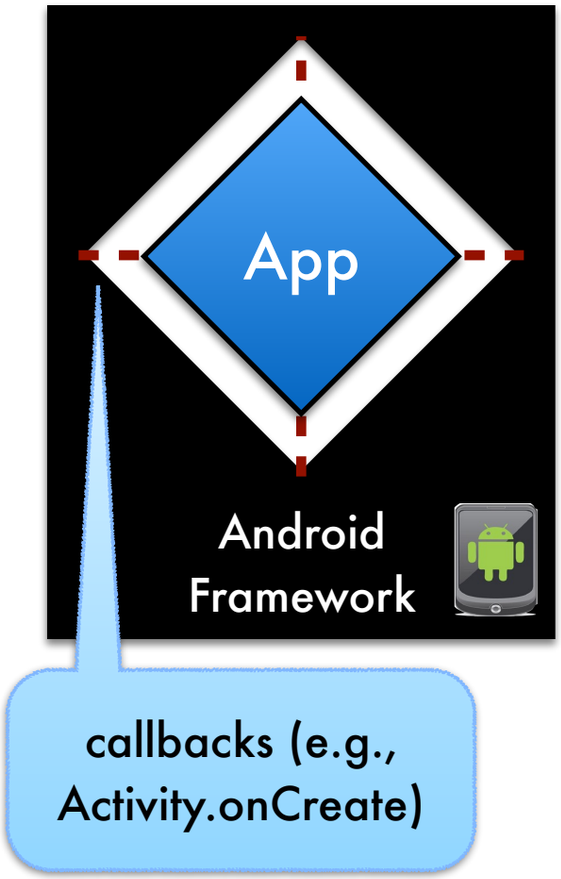
Callback-oriented programming …

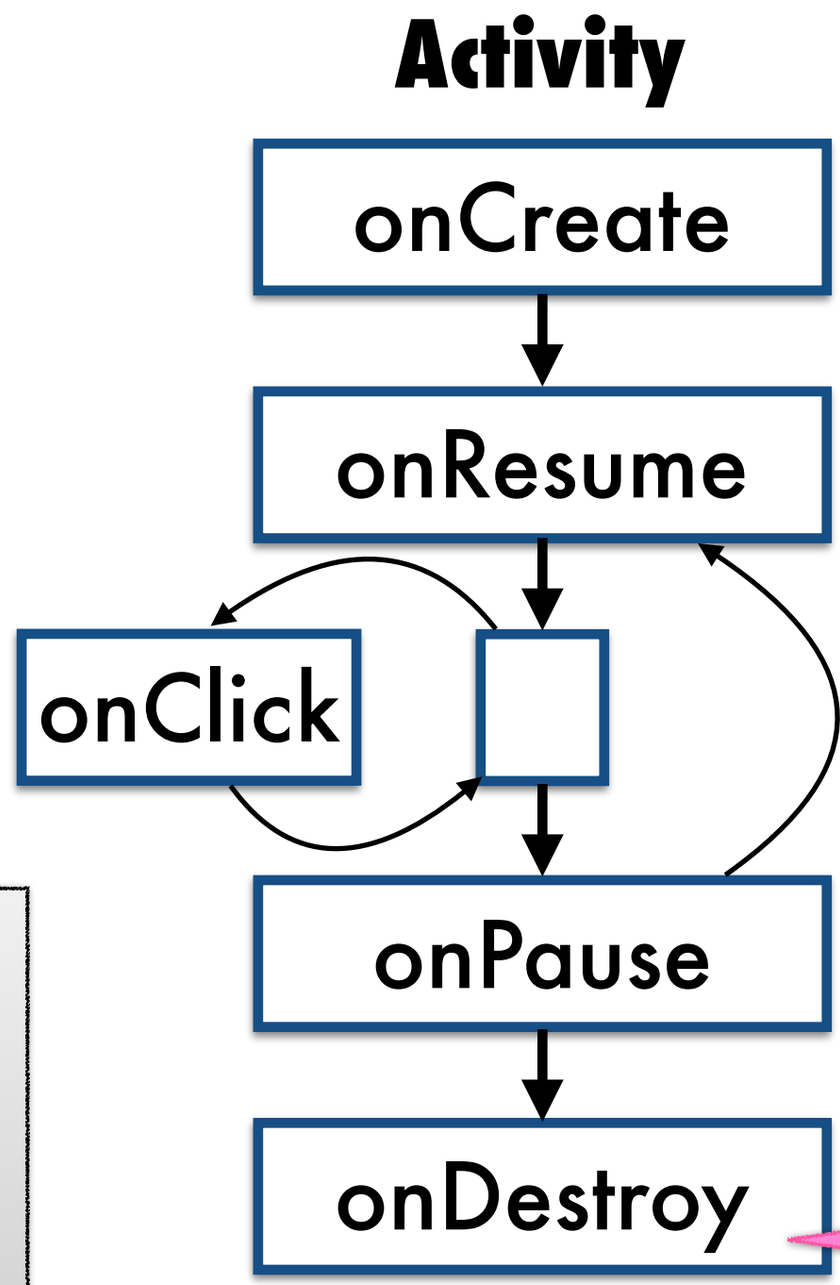




Android components have an ordered, event-driven lifecycle

Callback-oriented programming …



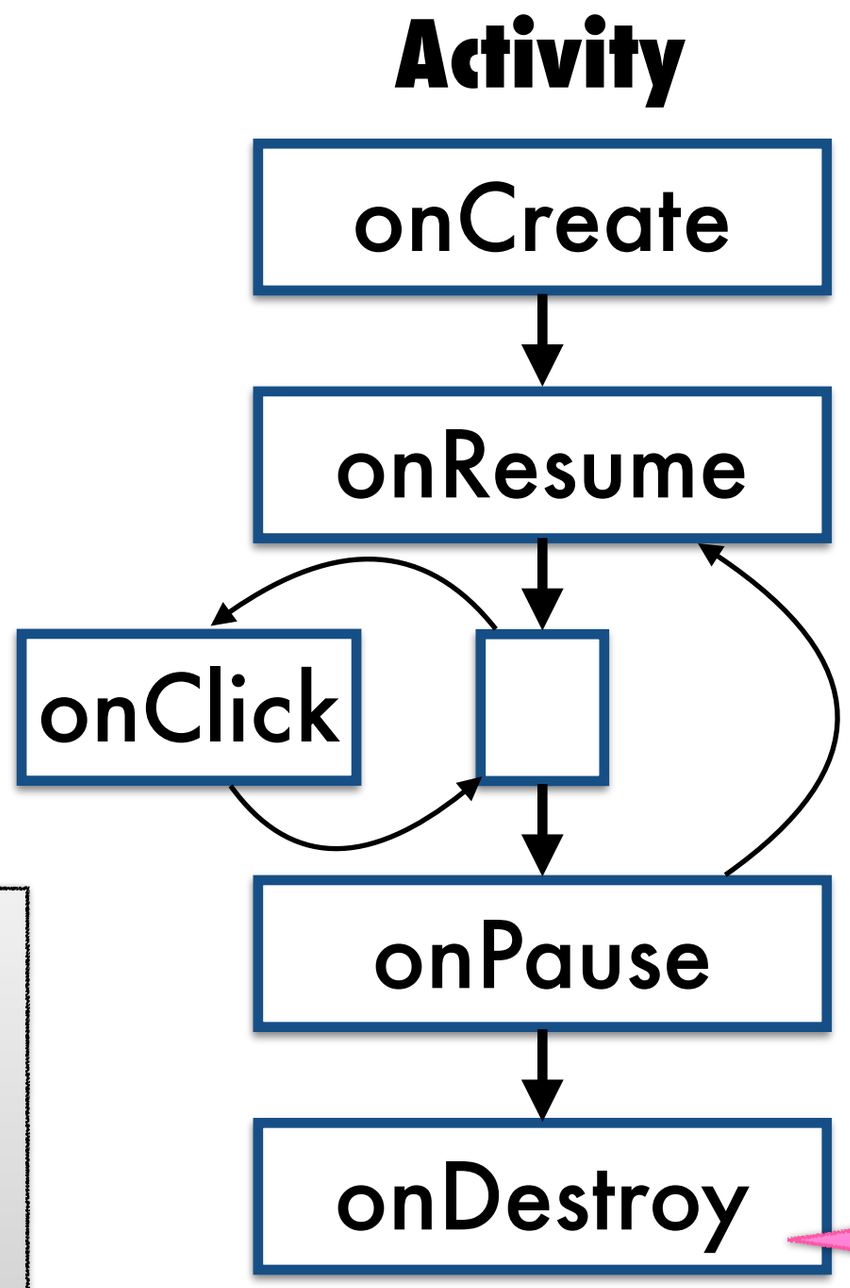



```
mHostDb = null;
mService = null;
```

Android components have an ordered, event-driven lifecycle

Callback-oriented programming …

callbacks (e.g., Activity.onCreate)



But, lifecycles of different components and other callbacks can interleave …

```
mHostDb = null;
mService = null;
```

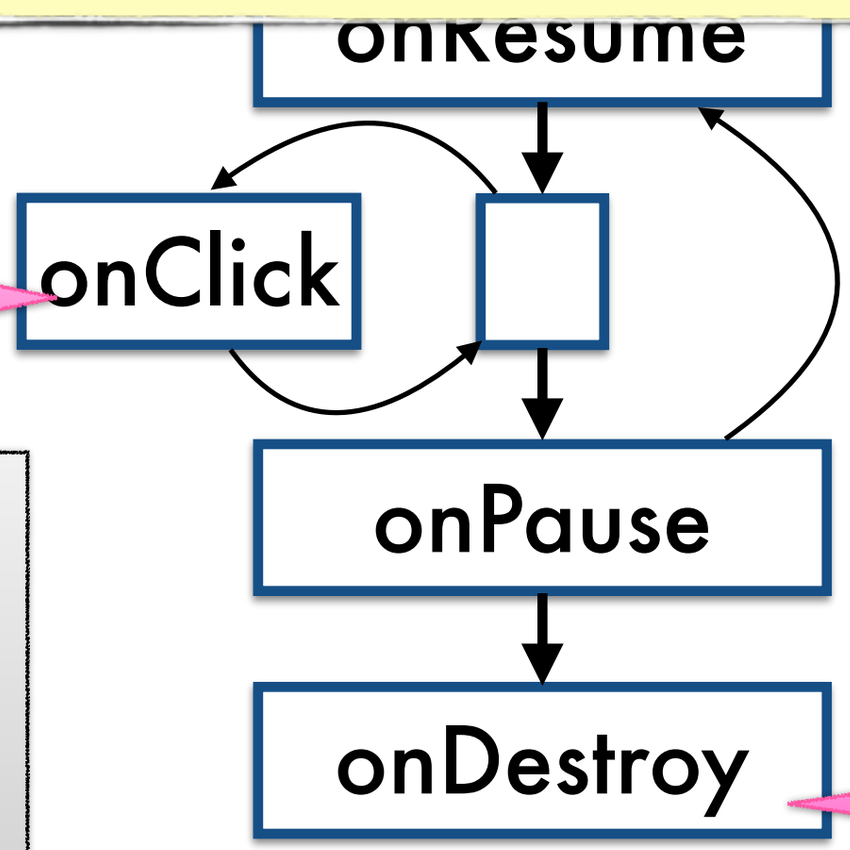
Android components have an ordered, event-driven lifecycle

Callback-oriented programming …

Challenge: Verifying safety (of dereferences) depends on callback interleaving

safe?

```
mHostDb.s();
```

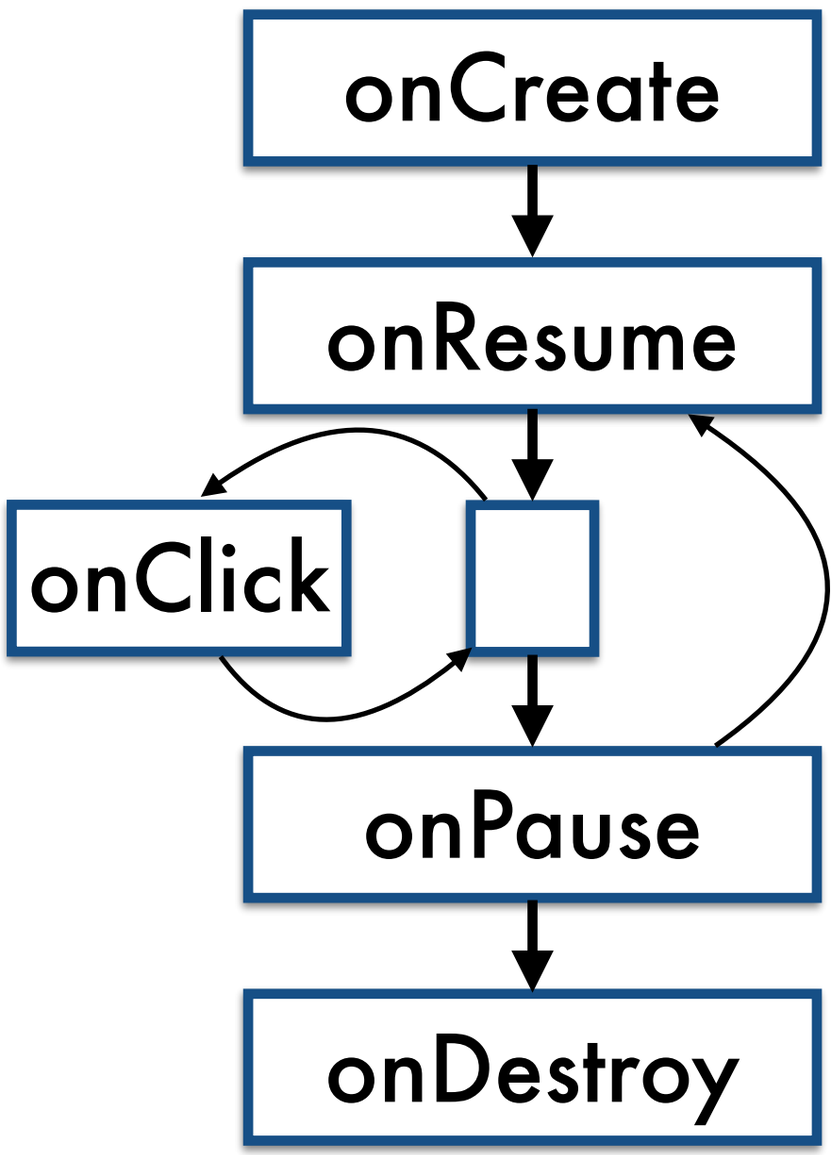



```
mHostDb = null;
mService = null;
```

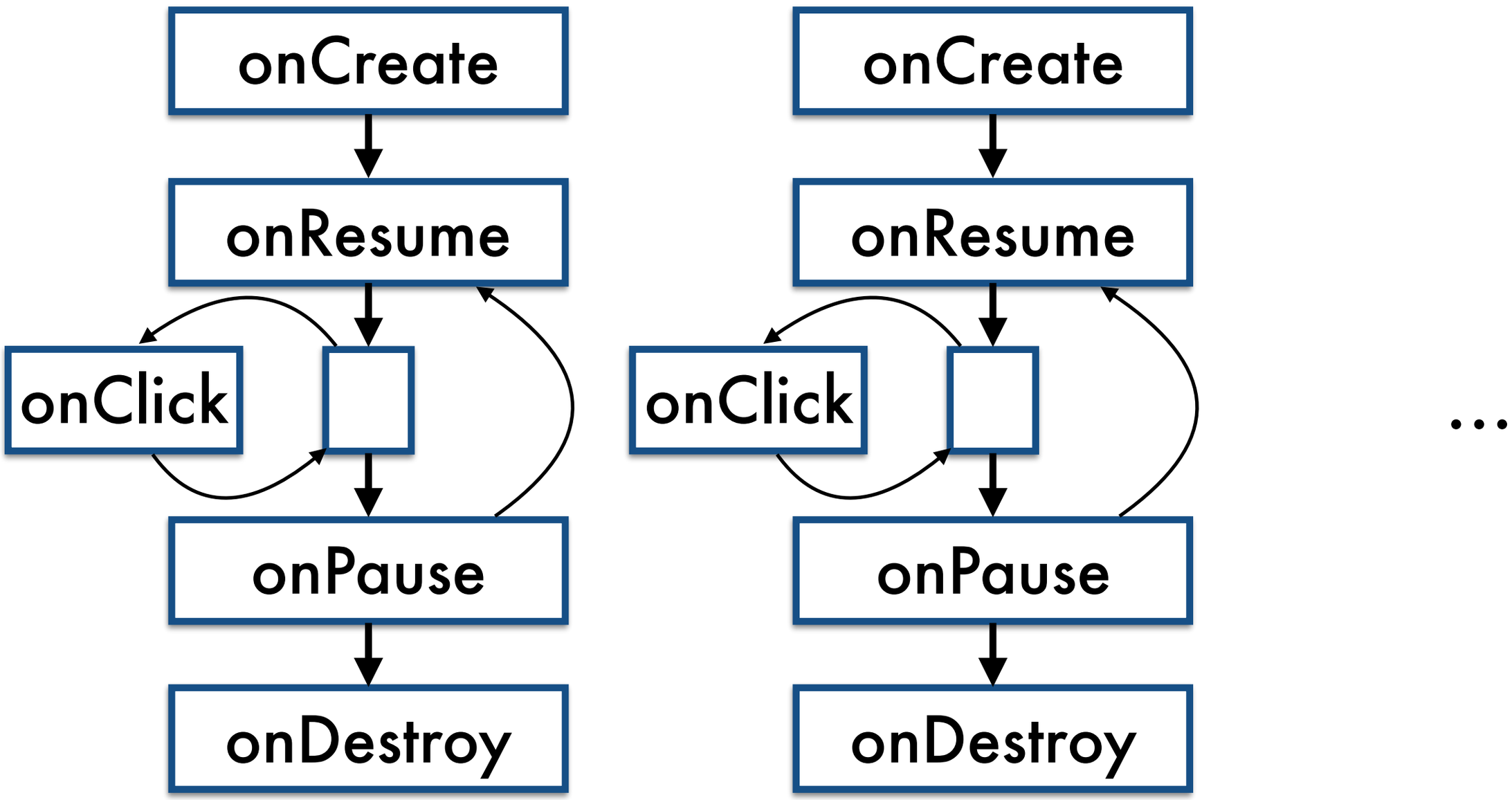
But, lifecycles of different components and other callbacks can interleave …

Android components have an ordered, event-driven lifecycle

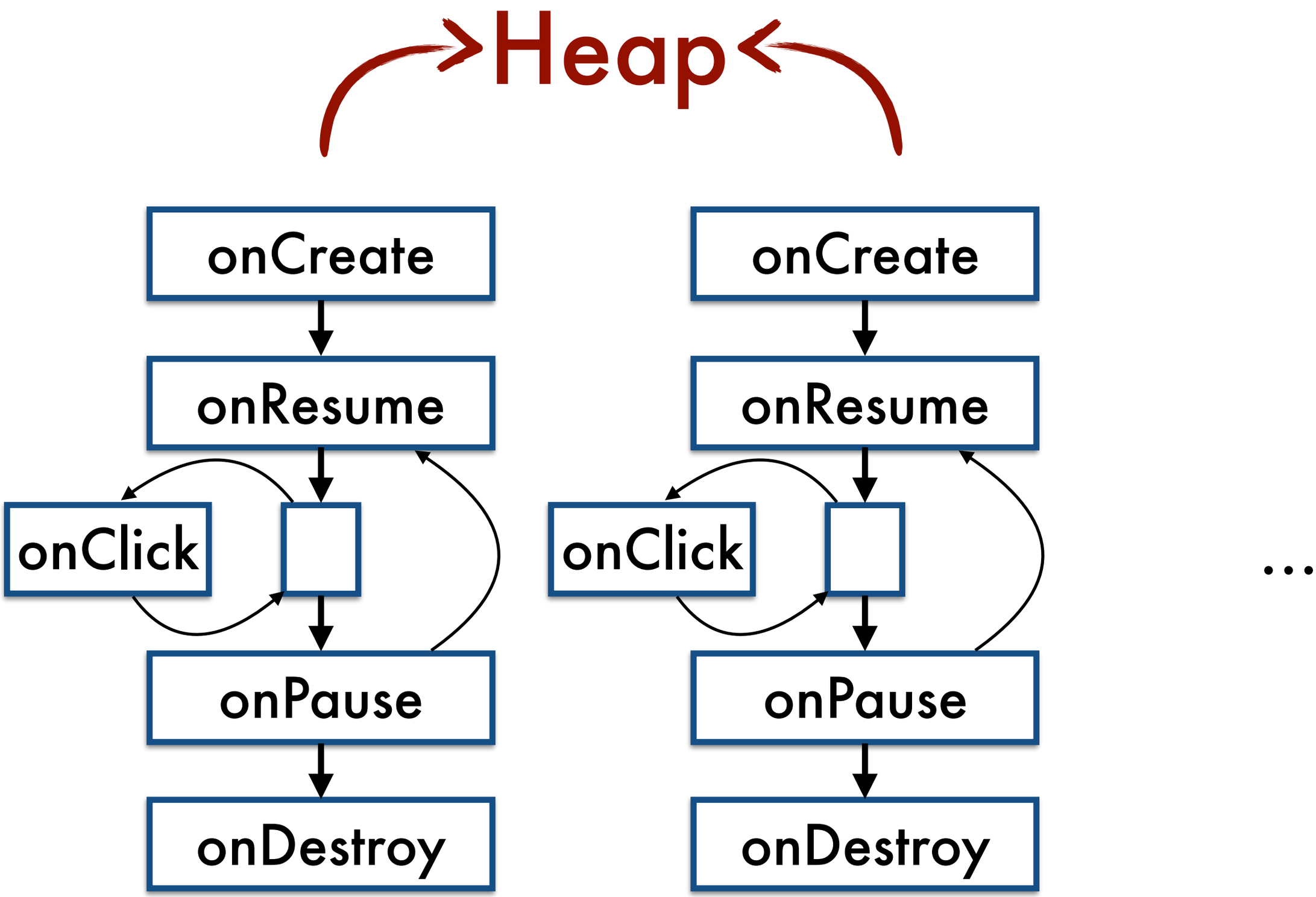
… over a shared, global heap



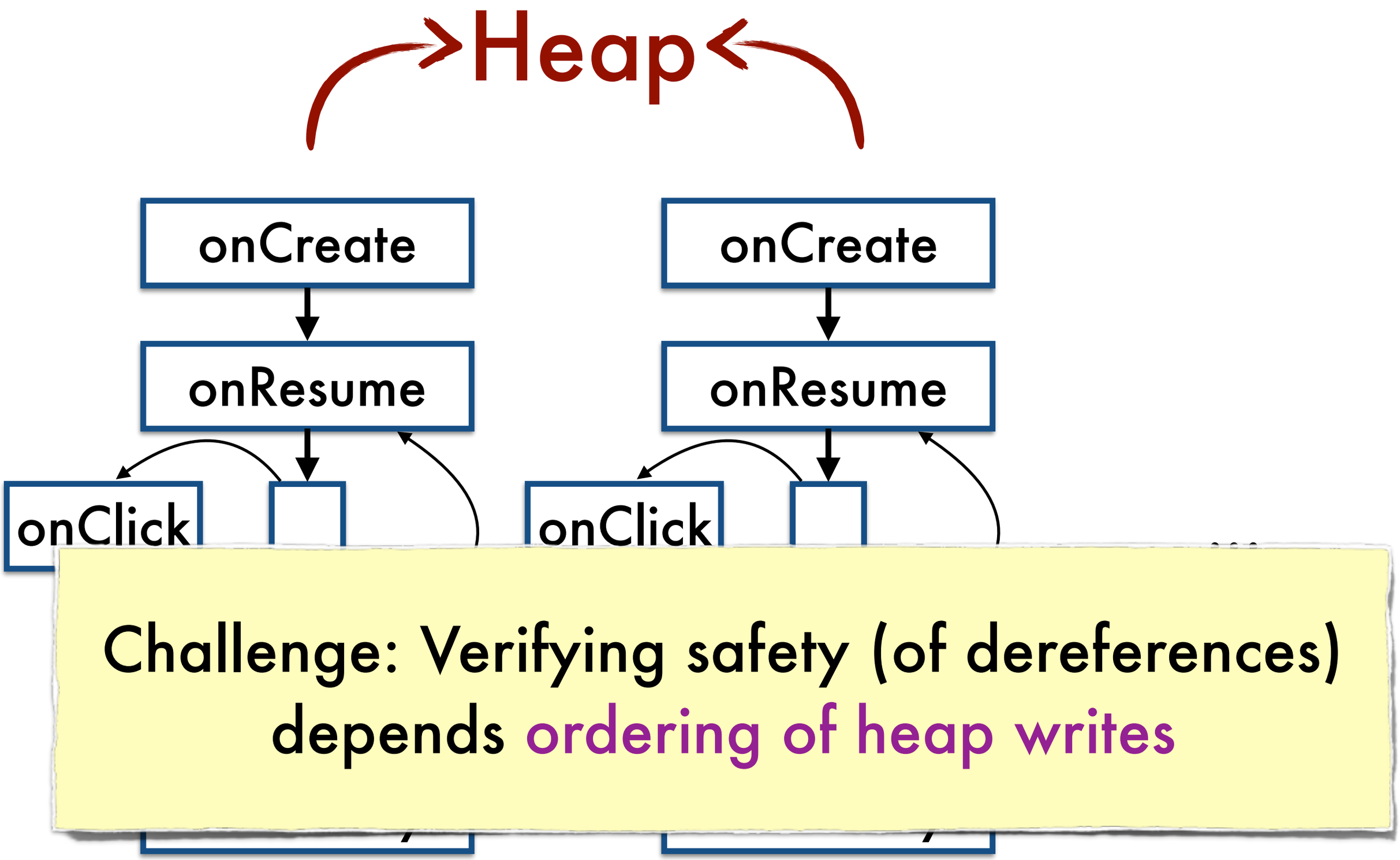
… over a shared, global heap



… over a shared, global heap

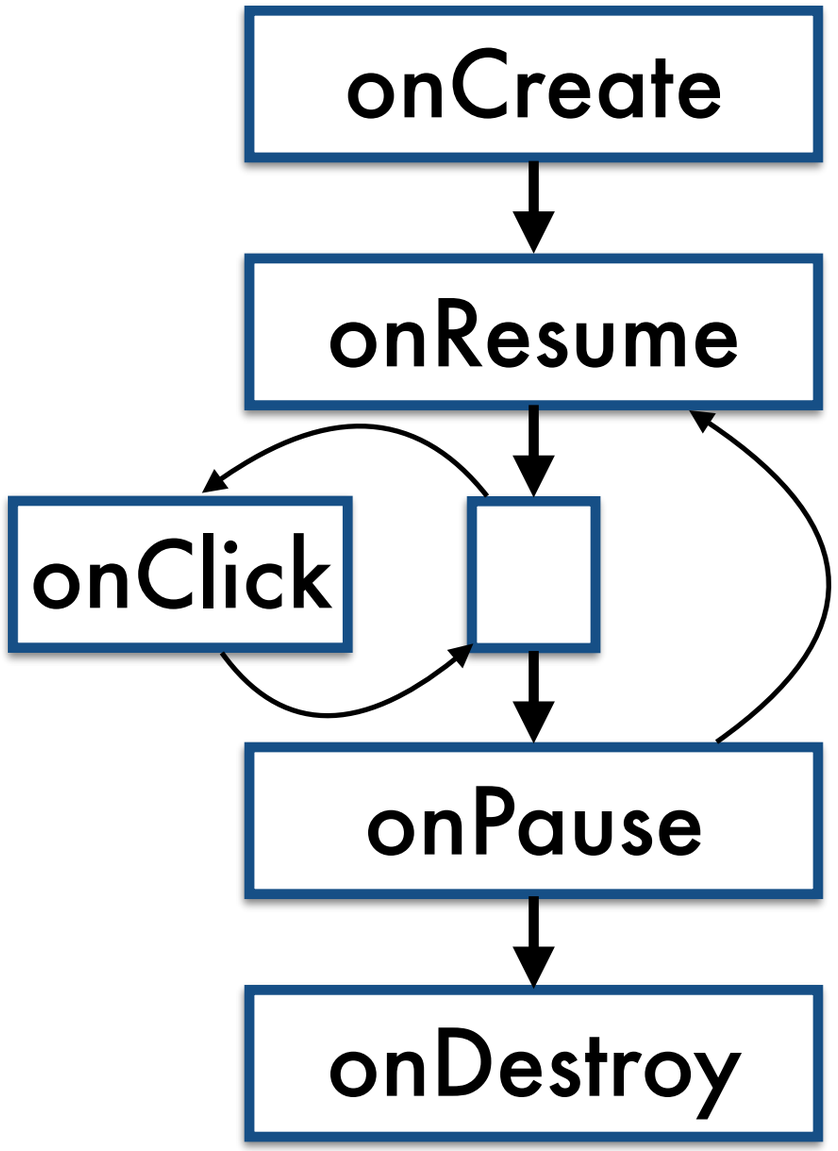


… over a shared, global heap

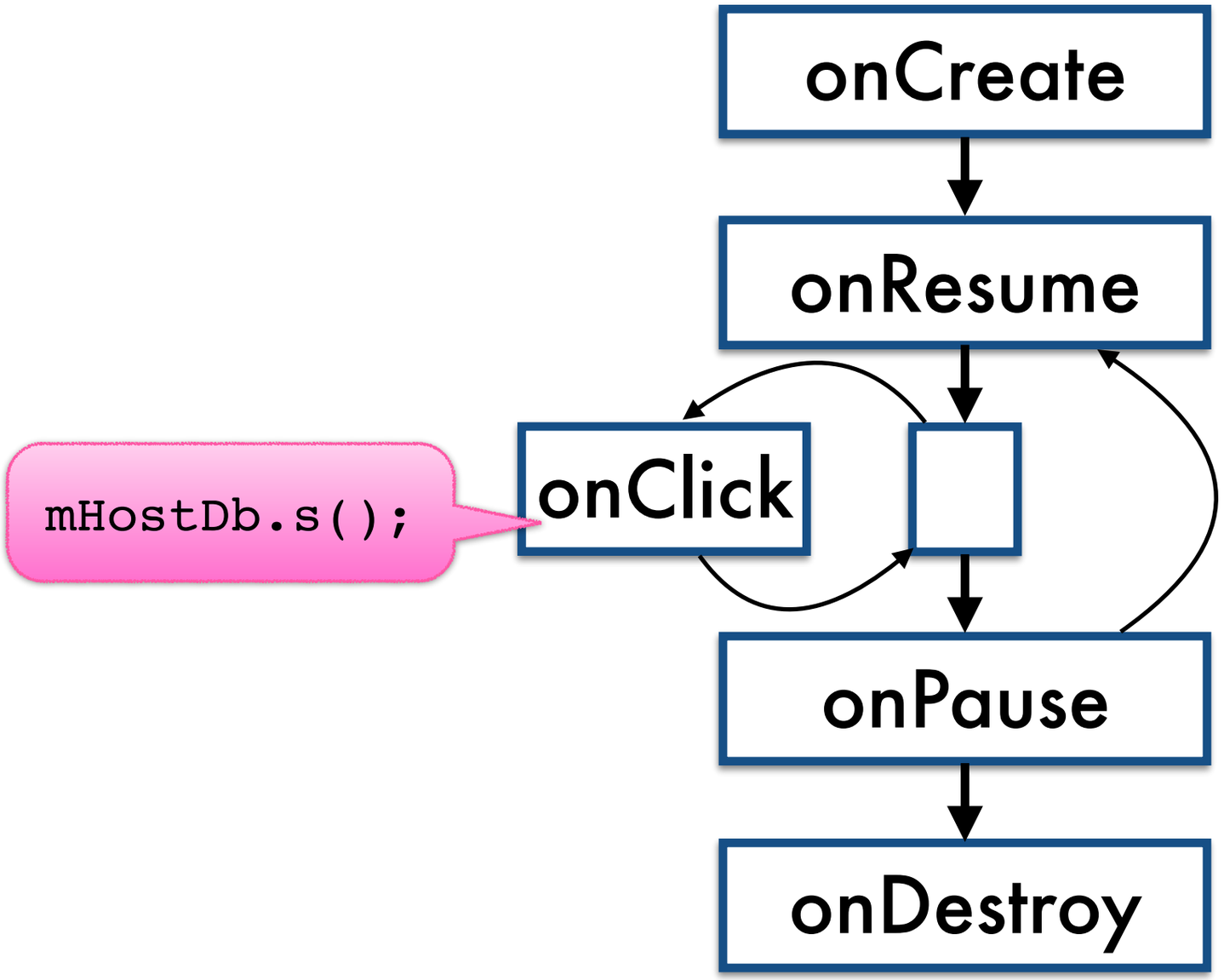


Challenge: Verifying safety (of dereferences) depends ordering of heap writes

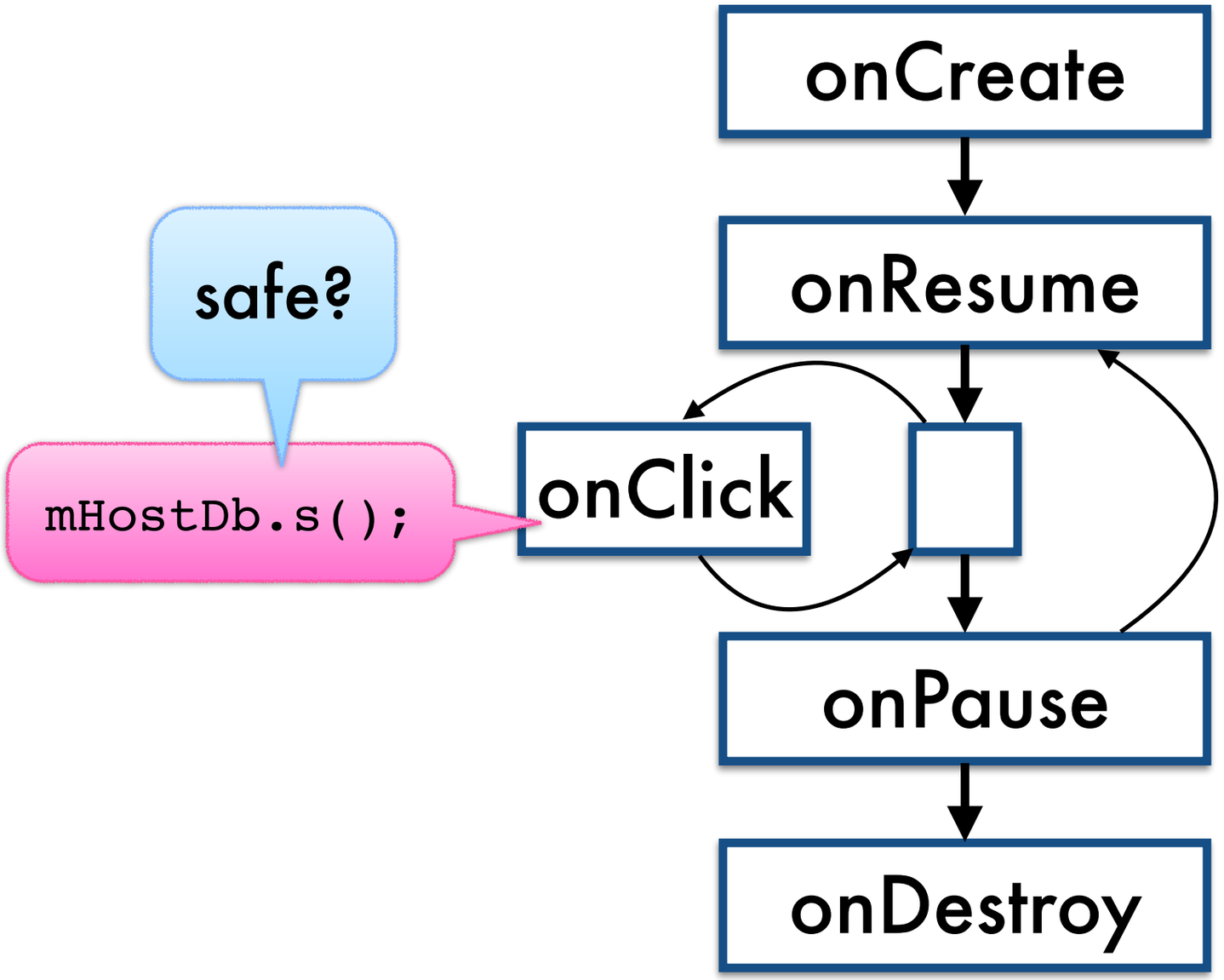
But verifying safety shouldn't be so hard ...



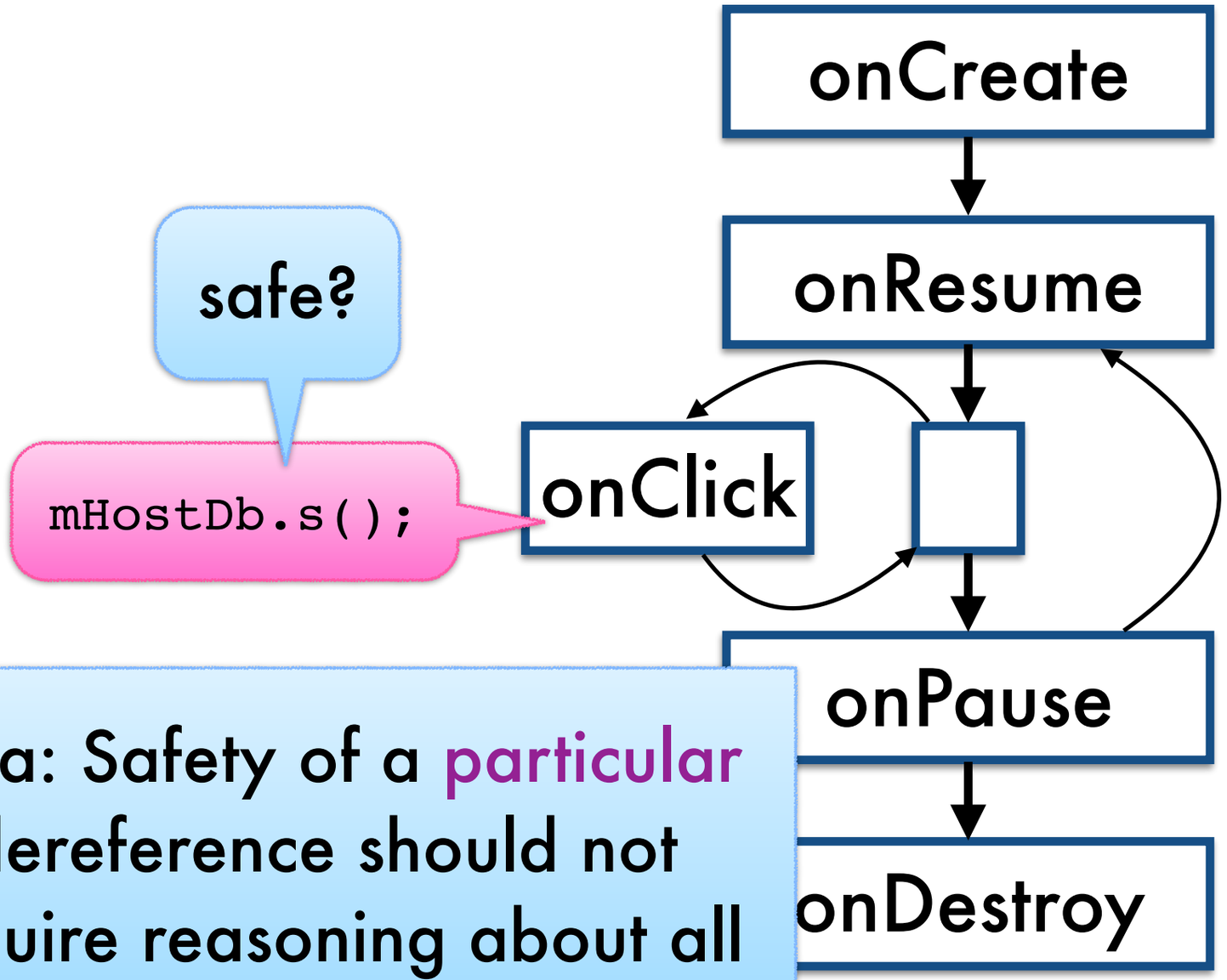
But verifying safety shouldn't be so hard ...



But verifying safety shouldn't be so hard ...

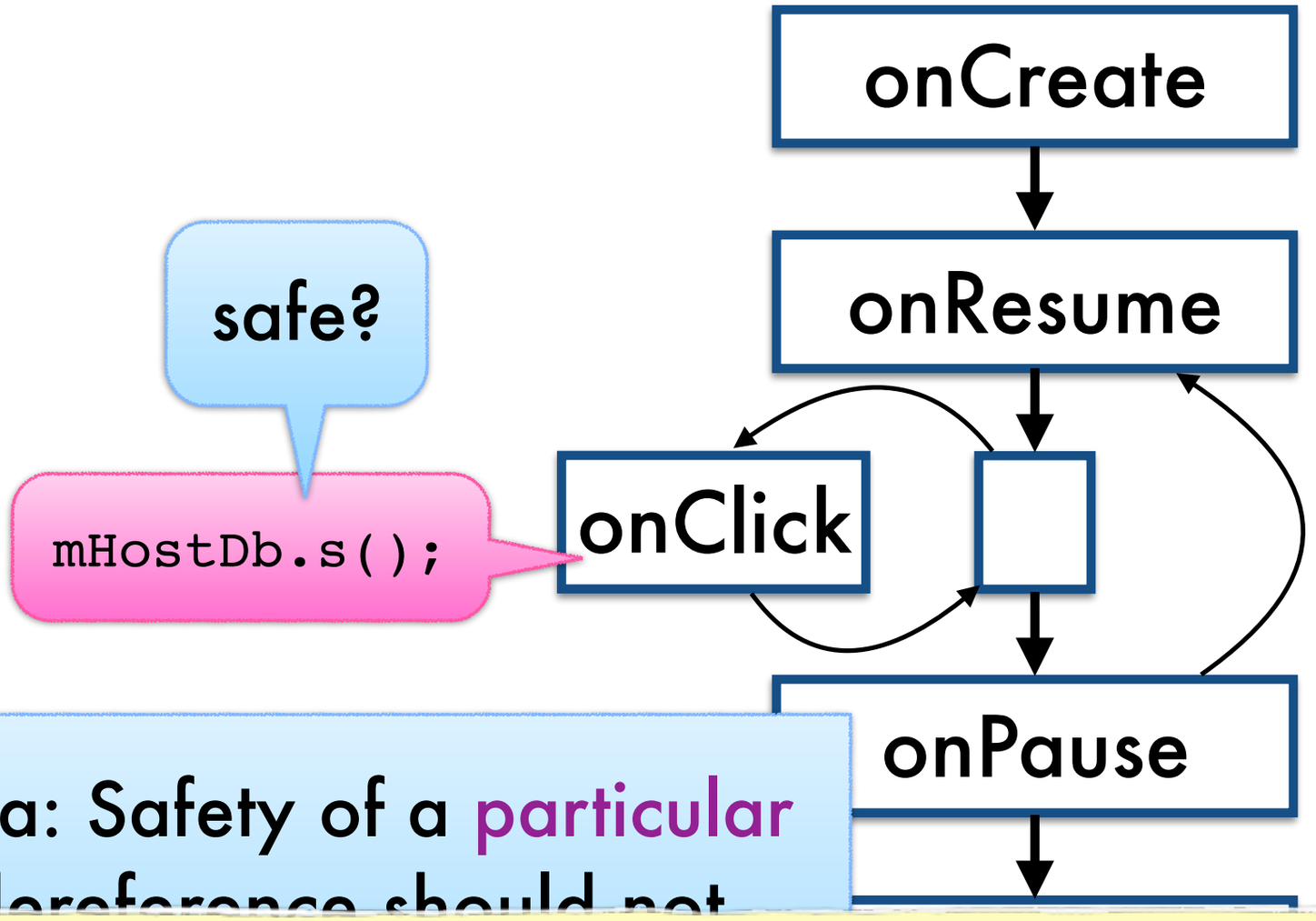


But verifying safety shouldn't be so hard ...



Idea: Safety of a particular dereference should not require reasoning about all callback interleavings

But verifying safety shouldn't be so hard ...



Idea: Safety of a particular dereference should not

A "smart" goal-directed analysis could consider relevant callback orderings without considering all of them—jumping analysis

Evaluation: Proving dereferences safe

Evaluation: Proving dereferences safe

10 open source Android apps

3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Evaluation: Proving dereferences safe

10 open source Android apps

3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Event product graph would have 10^{10} to 10^{111} nodes (with one instance per class)

Evaluation: Proving dereferences safe

10 open source Android apps

3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Evaluation: Proving dereferences safe

10 open source Android apps

3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Compared 3 analyses

Nit: type inference
Thresher: no jumping
Hopper: jumping

Is Hopper effective at proving safety?

	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is Hopper effective at proving safety?

	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
[illegible]	[illegible]	5921	3643	698	330 (53)
[illegible]	[illegible]	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Many derefs

Is Hopper effective at proving safety?

		unproven derefs			
	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is Hopper effective at proving safety?

Type inference | No jumping | Jumping

unproven derefs

	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is Hopper effective at proving safety?

		unproven derefs			
	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is Hopper effective at proving safety?

		unproven derefs			
	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Thr and Hop: 10 second budget per deref

Is Hopper effective at proving safety?

		unproven derefs			
	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is Hopper effective at proving safety?

		unproven derefs			
	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Thresher has 74% fewer than Nit

Is Hopper effective at proving safety?

		unproven derefs			
	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Hopper has 54% fewer than Thresher

Is Hopper effective at proving safety?

		unproven derefs				
	KLOC	Deref	Nit	Thr	Hop (Impr %)	% Proven
drupaleditor	3	928	679	179	72 (60)	92
npr	5	829	617	181	51 (72)	94
duckduckgo	11	1969	1341	518	143 (72)	93
lastfm	13	4840	3528	954	477 (50)	90
github	19	3603	2520	601	290 (52)	92
seriesguide	32	8184	5438	986	625 (37)	92
connectbot	33	2190	1562	316	74 (77)	97
textsecure	38	5921	3643	698	330 (53)	94
k-9	55	19032	11968	3104	1988 (36)	90
wordpress	57	15066	9775	2431	1362 (44)	91
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**	**92**

Is Hopper effective at proving safety?

		unproven derefs				
	KLOC	Deref	Nit	Thr	Hop (Impr %)	% Proven
drupaleditor	3	928	679	179	72 (60)	92
npr	5	829	617	181	51 (72)	94
duckduckgo	11	1969	1341	518	143 (72)	93
lastfm	13	4840	3528	954	477 (50)	90
github	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	92
seriesguide	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	92
connectbot	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	97
textsecure	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	94
k-9	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	90
wordpress	57	15066	9775	2431	1362 (44)	91
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**	**92**

Compare with state-of-the art NPE checking work that reports 84-91% proven on normal Java programs!

Remaining manual triage ...

Remaining manual triage ...

Triaged 200 alarms (from Hopper), 189 false

Remaining manual triage …

Triaged 200 alarms (from Hopper), 189 false

- Reasons: insufficient Android modeling, imprecise container and string domains

Remaining manual triage …

Triaged 200 alarms (from Hopper), 189 false

- Reasons: insufficient Android modeling, imprecise container and string domains
- Only 17 false alarms due to timeouts

Remaining manual triage ...

Triaged 200 alarms (from Hopper), 189 false

- Reasons: insufficient Android modeling, imprecise container and string domains
- Only 17 false alarms due to timeouts

Found 11 bugs in 4 apps
(lastfm, seriesguide, connectbot, wordpress)

5 bugs due to bad ordering assumptions

10/11 patches accepted

Remaining manual triage ...

Tria se

Fixing possible NPE in OverviewActivity.launchSearch #449

Merged Uw

Fixing possible NPEs when CheckInDialogFragment.newInstance returns null #450

Merged

Fix possible NPE in HostListActivity.onCreateContextMenu #60

Merged kruton merged 1 commit into connectbot:master from sblackshear:service_npe_fix on Mar 28

Re e

container

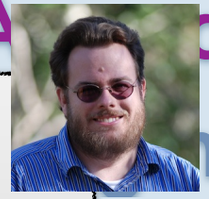
Only 17 false

Found 11 bugs in 4 apps
(lastfm, seriesguide, connectbot, wordpress)

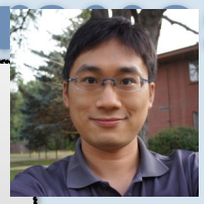
5 bugs due to bad ordering assumptions

10/11 patches accepted

Research A da: The cooperative approach addr s mitigation proce



Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]



Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

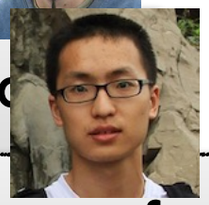
Test Input

Runner

Test Output

Program-ming

Spec-ification



Jsana: Abstract Domain Combin

Auditr: Securing Against "Inputs of Coma and Exposure"
[in progress]

of of no bug

Program

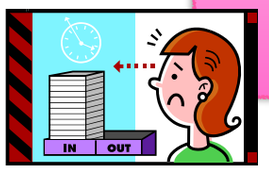
Alarm Report

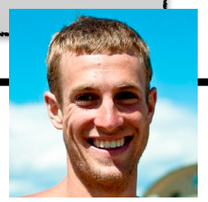
Github

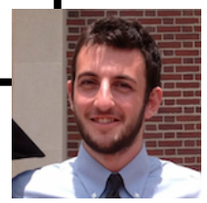
Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

Fixr: Mining Bug Fixes from Commits
[in progress]

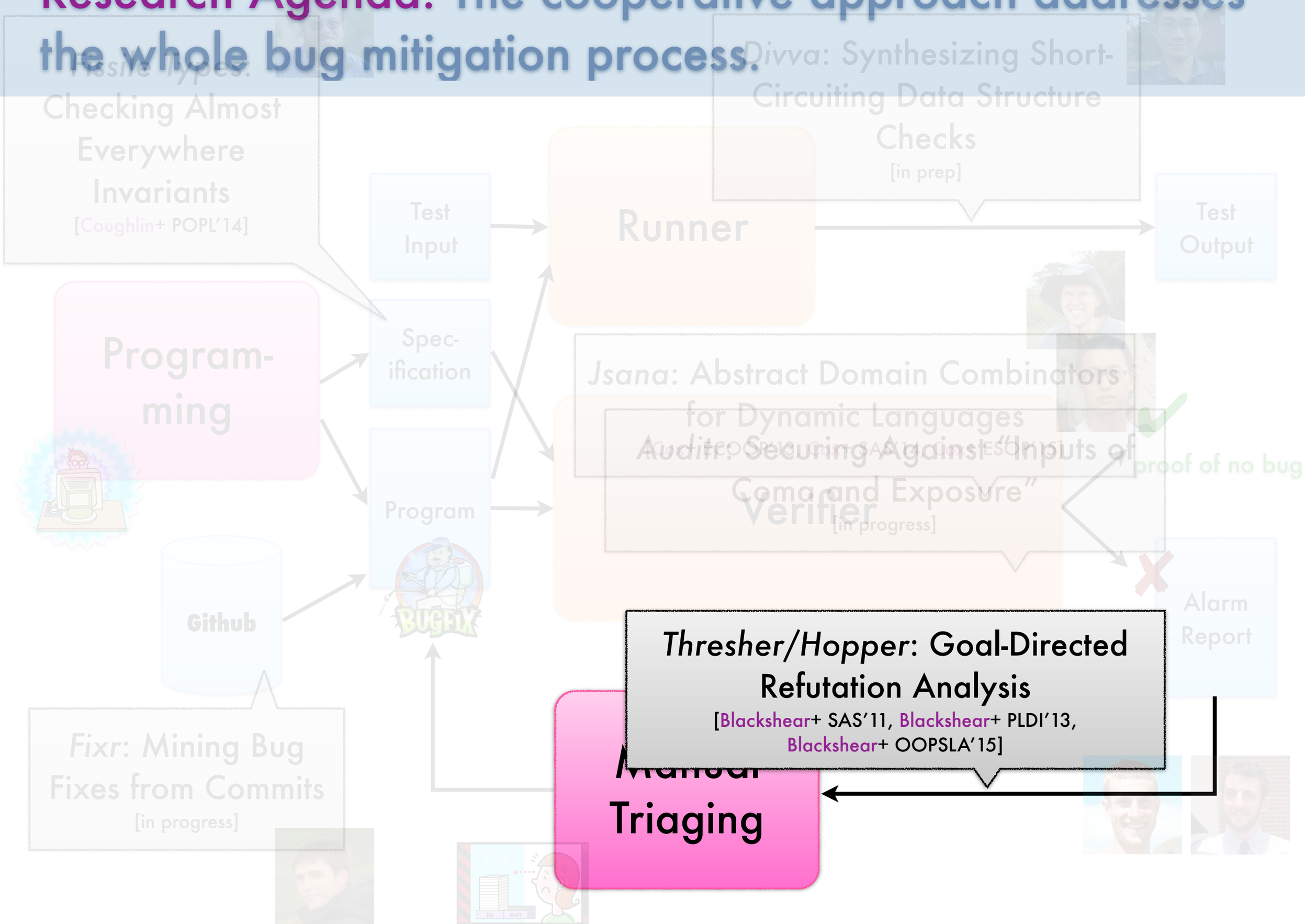
Manual Triaging



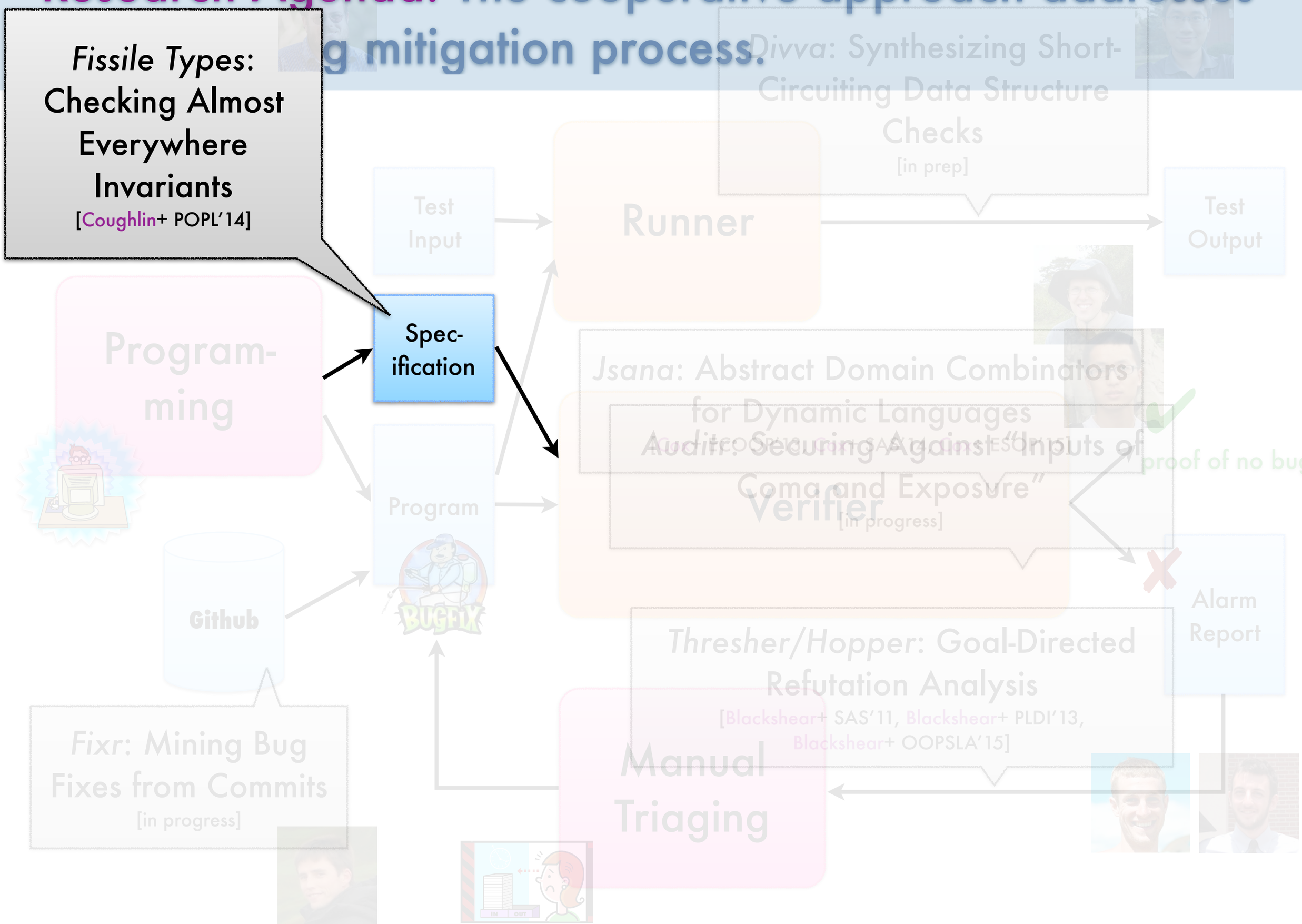




Research Agenda: The cooperative approach addresses the whole bug mitigation process.



Research Agenda: The cooperative approach addresses g mitigation process.



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.

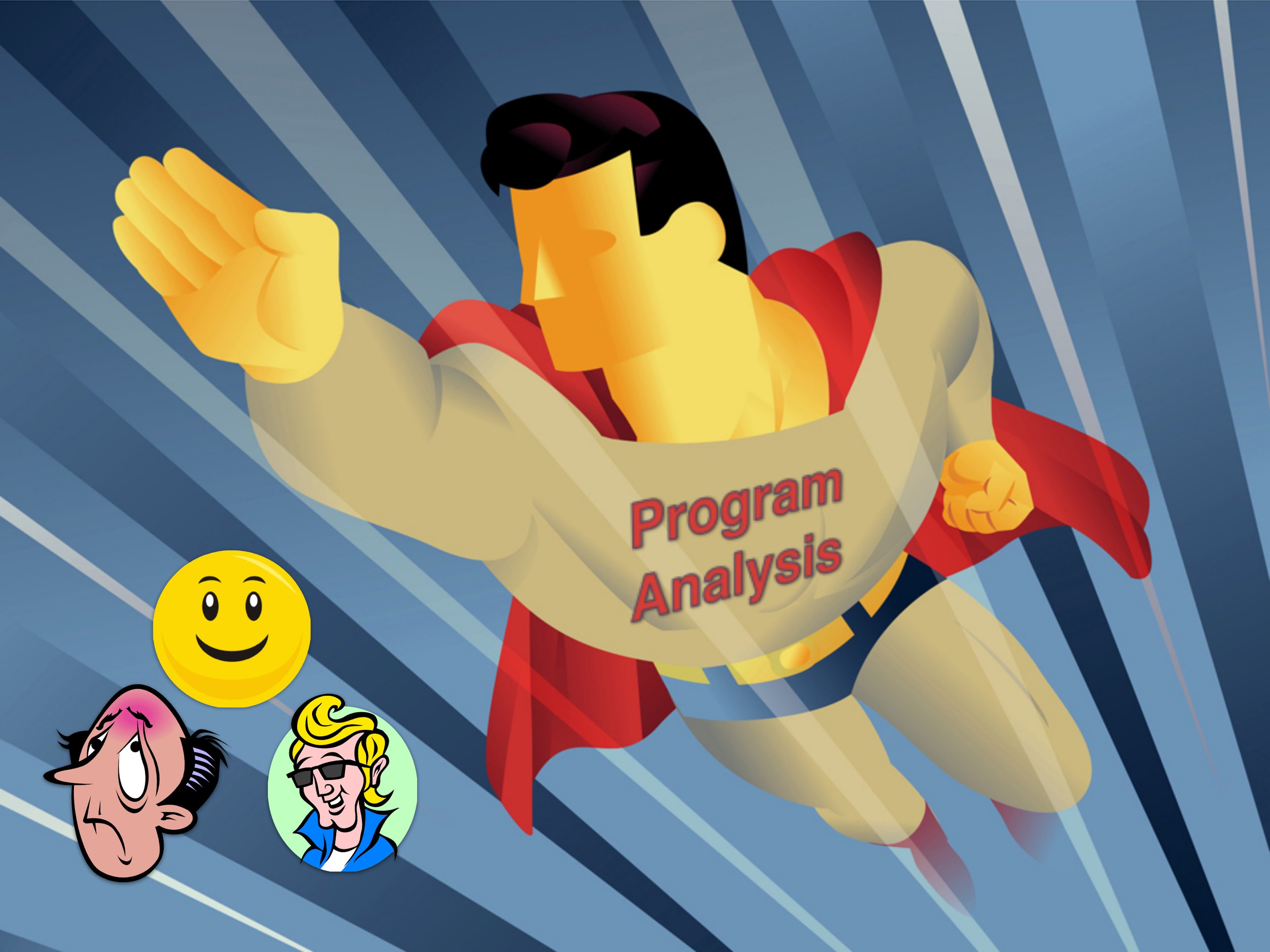
Wh
obj
by

"Web 2.0" developers think it's cool.

static safety.

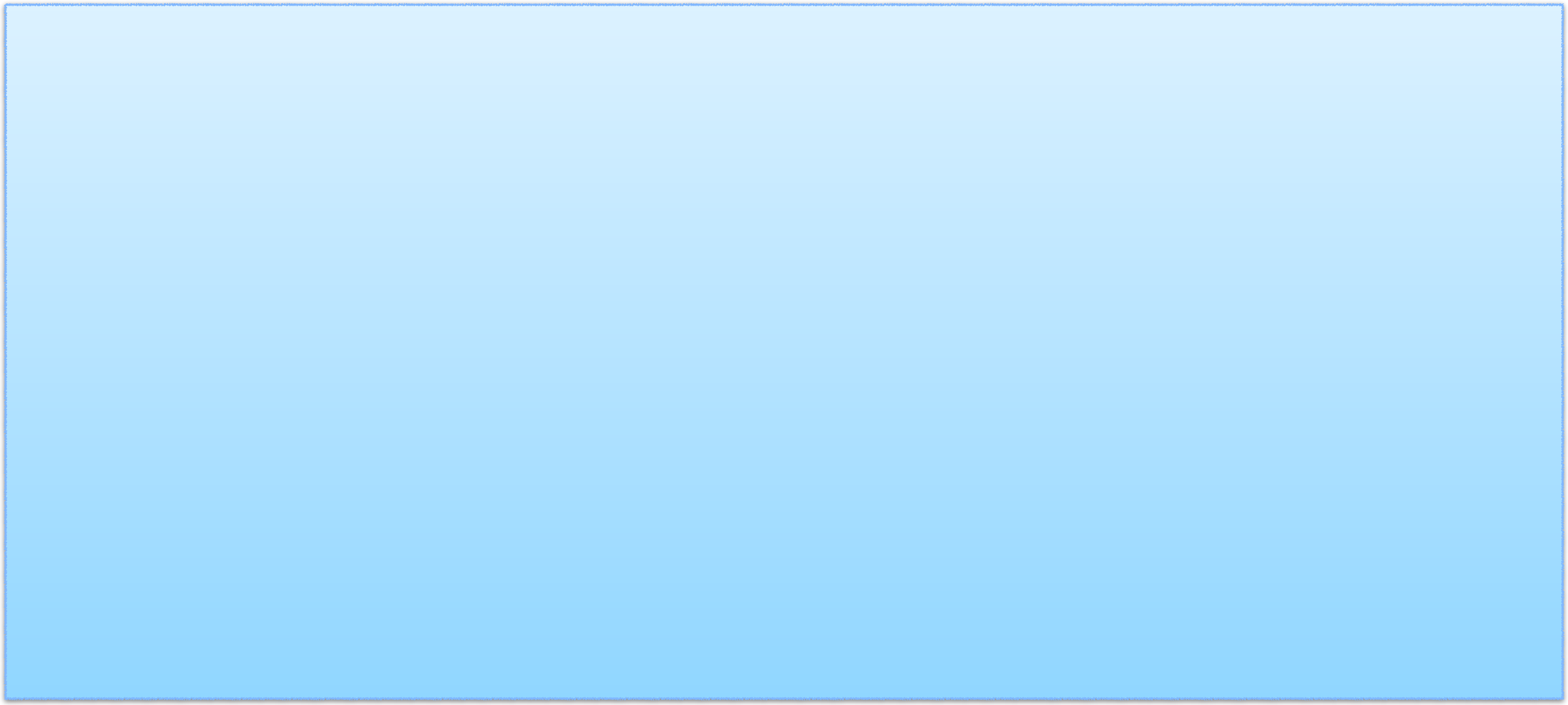
"MethodNotFound" checked at run time


Program
Analysis


Program
Analysis

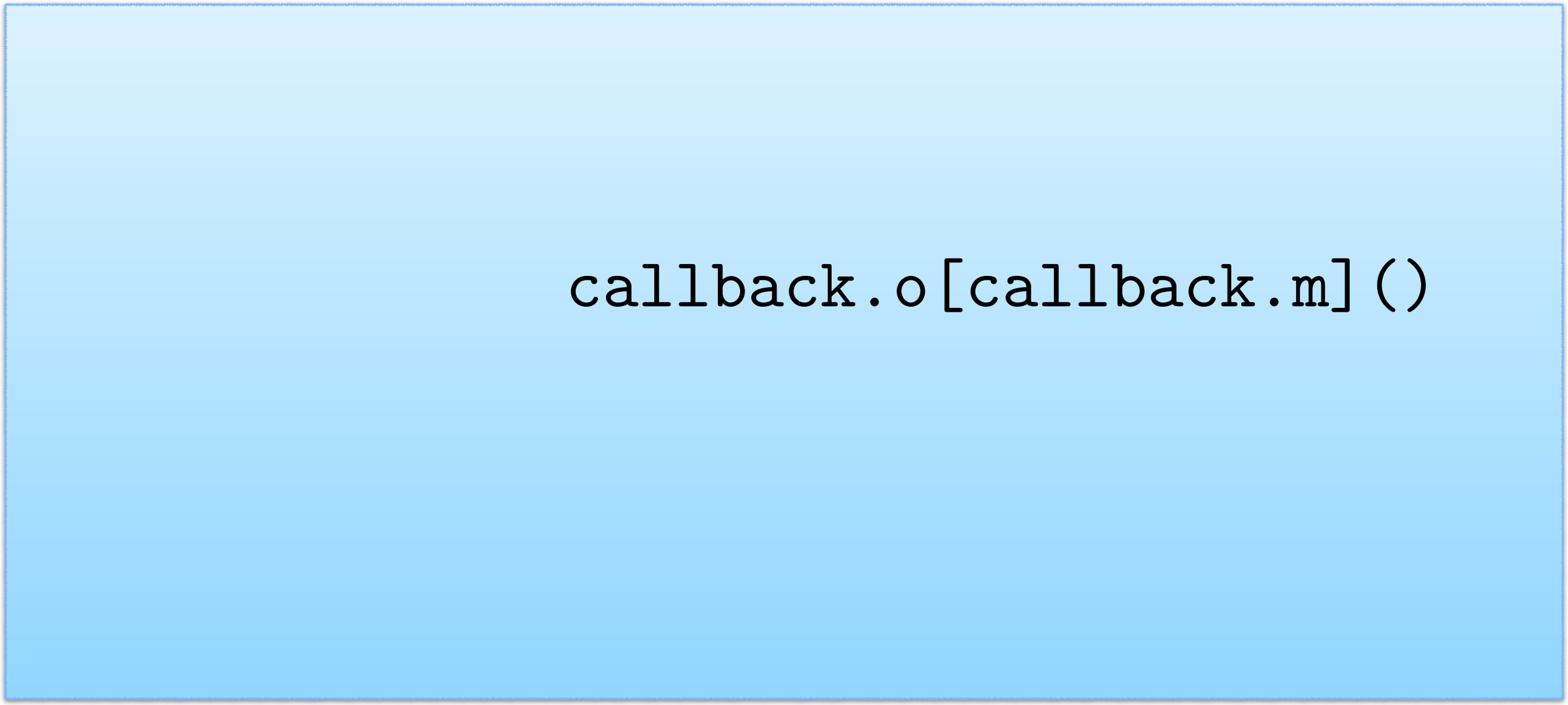
Programs are often
(1) safe, (2) not type safe, (3) but almost so

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



Program

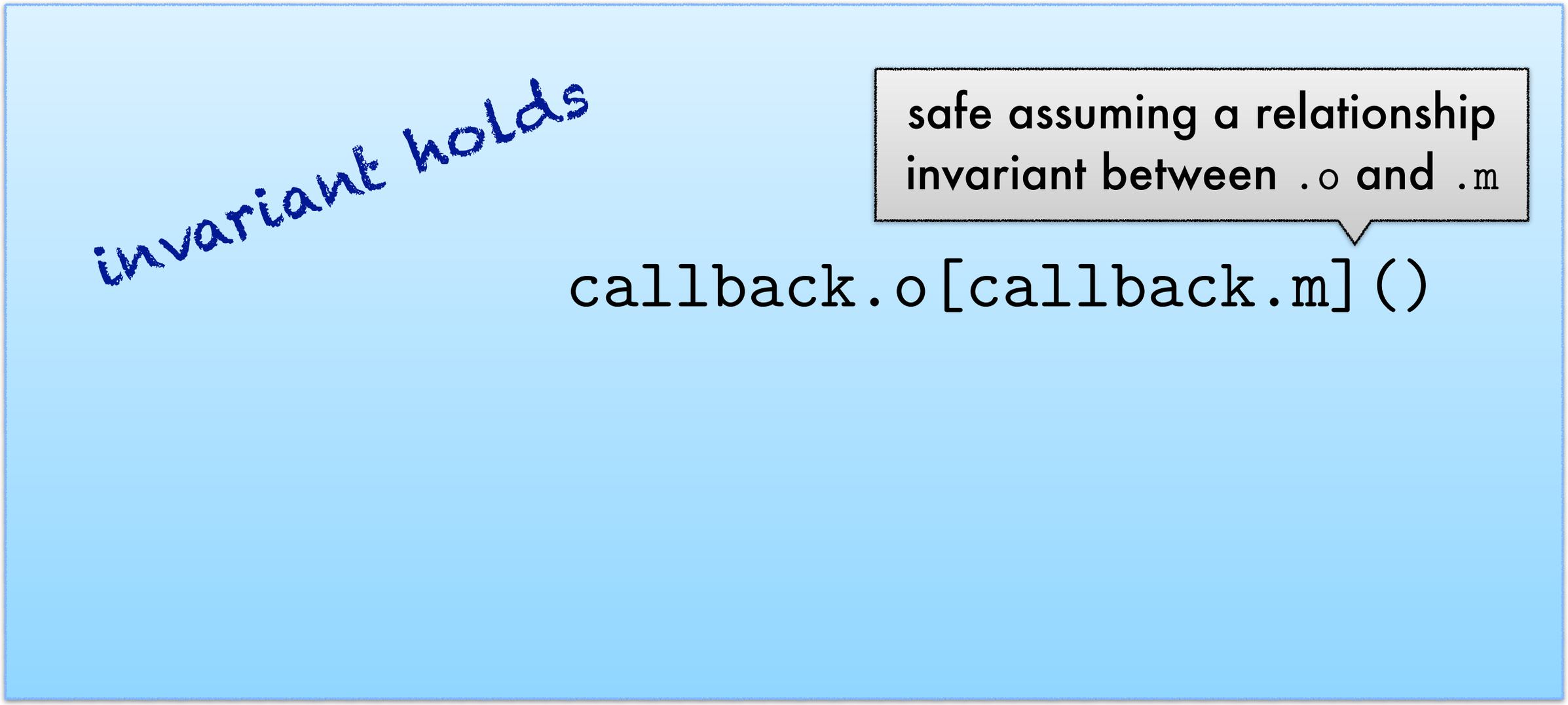
Programs are often
(1) safe, (2) not type safe, (3) but almost so

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

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

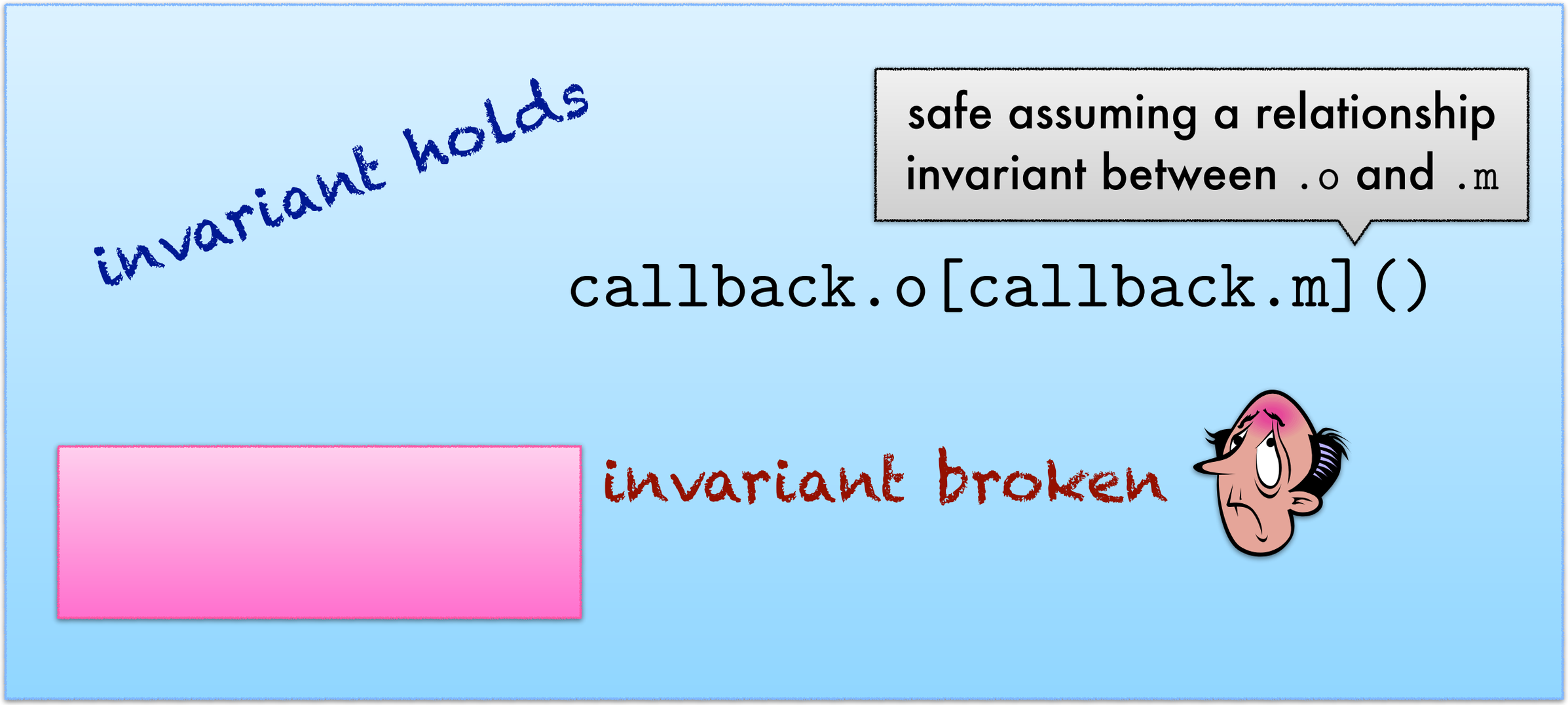
Program

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



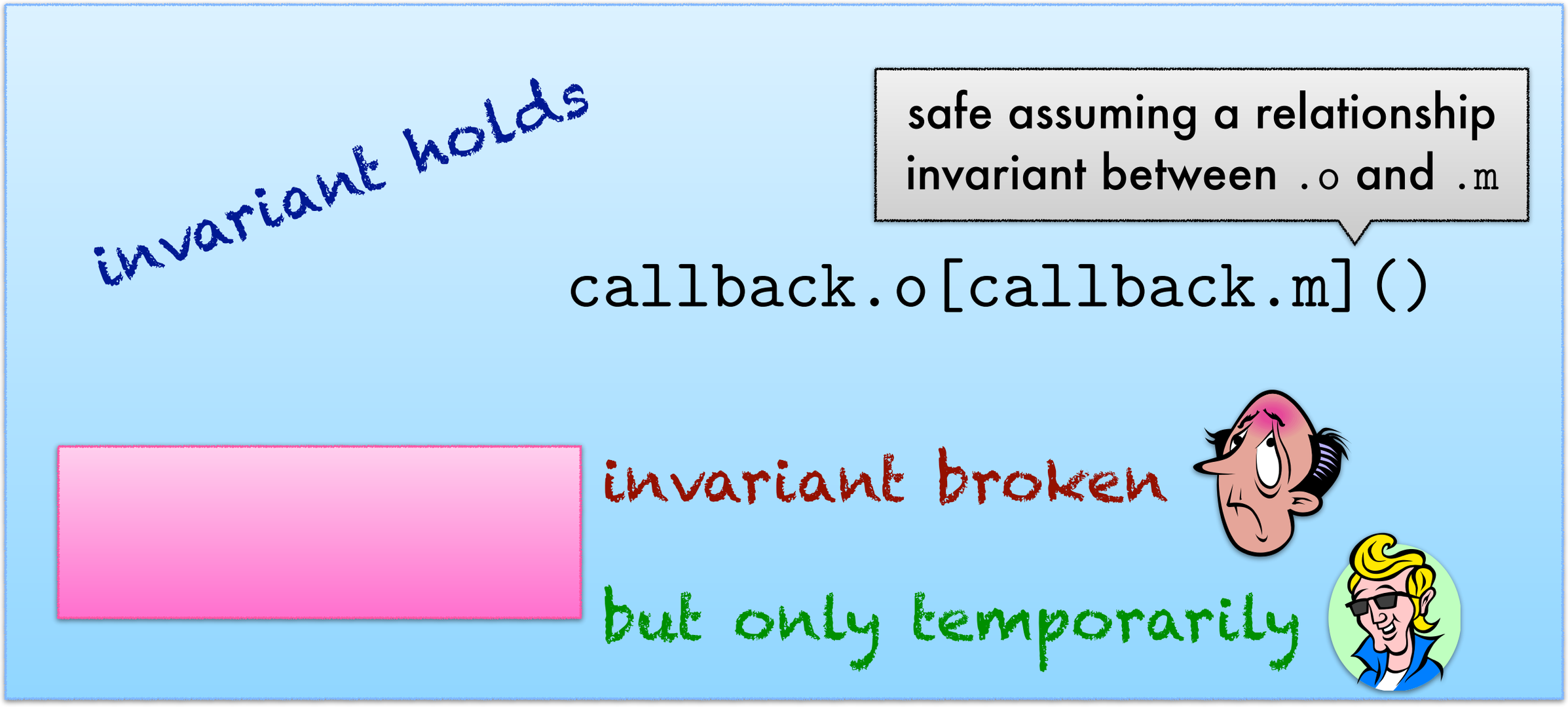
Program

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



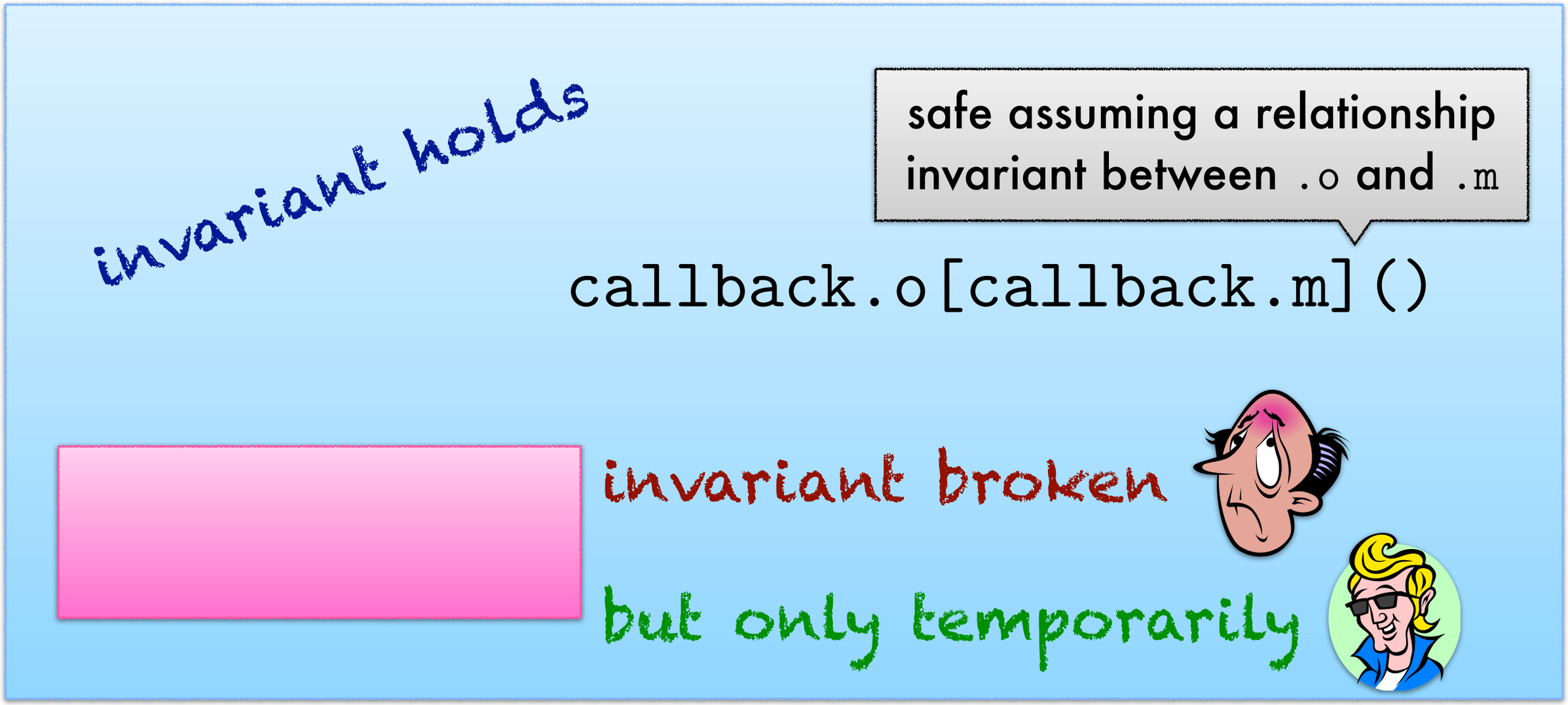
Program

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



Program

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



Program

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 reflection checks in system libraries
- needed only 136 annotations in benchmarks (total)

Is Fissile effective at proving reflective call safety?

Including Skim, Adium, and OmniGraffle

Fissile analyzes Objective-C

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

Type annotations

seeded with 76 reflection checks in system libraries

needed only 136 annotations in benchmarks (total)

Is Fissile effective at proving reflective call safety?

Including Skim, Adium, and OmniGraffle

Fissile analyzes Objective-C

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?

Including Skim, Adium, and OmniGraffle

Fissile analyzes Objective-C

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?

Including Skim, Adium, and OmniGraffle

Fissile analyzes Objective-C

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

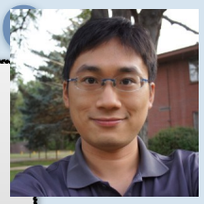
places requiring a check of the invariant

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 gation process.

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14]



Divva: Synthesizing Short-Circuiting Data Structure Checks
[in prep]

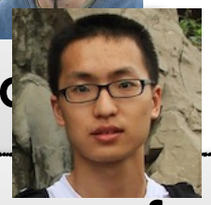
Test Input

Runner

Test Output

Program-ming

Spec-ification



Jsana: Abstract Domain Combin

Program

Auditr: Securing Against "Inputs of Coma and Exposure"
[in progress]

of of no bug

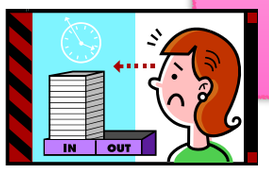
Alarm Report

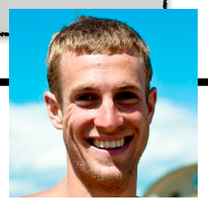
Github

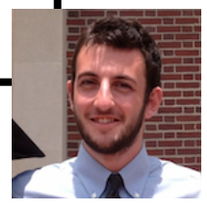
Thresher/Hopper: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, Blackshear+ OOPSLA'15]

Fixr: Mining Bug Fixes from Commits
[in progress]

Manual Triaging







Future Directions

Future Directions

The Cooperative Approach

- Unit Test Synthesis: test input + test code to unequivocally show bug
- Evidence for Alarms: alarm explanations, "probability" of bug
- Hardening: synthesize efficient dynamic checks
- Patch Synthesis: synthesize bug fixes
- Input Debugging: bugs in input
- Performance/Scalability/Security Bugs: beyond correctness bugs
- Analysis Engines: in new software domains

Future Directions

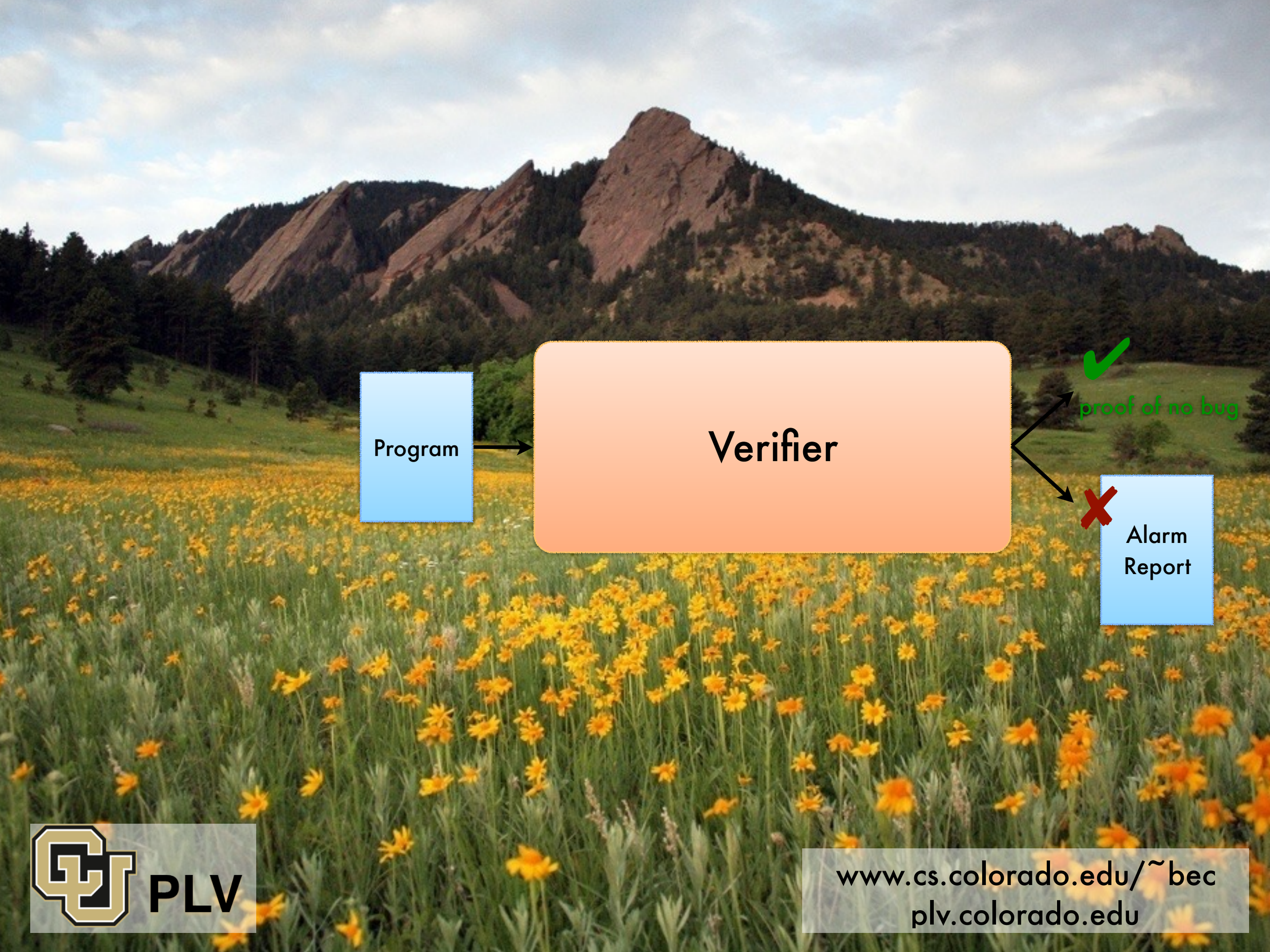
The Cooperative Approach

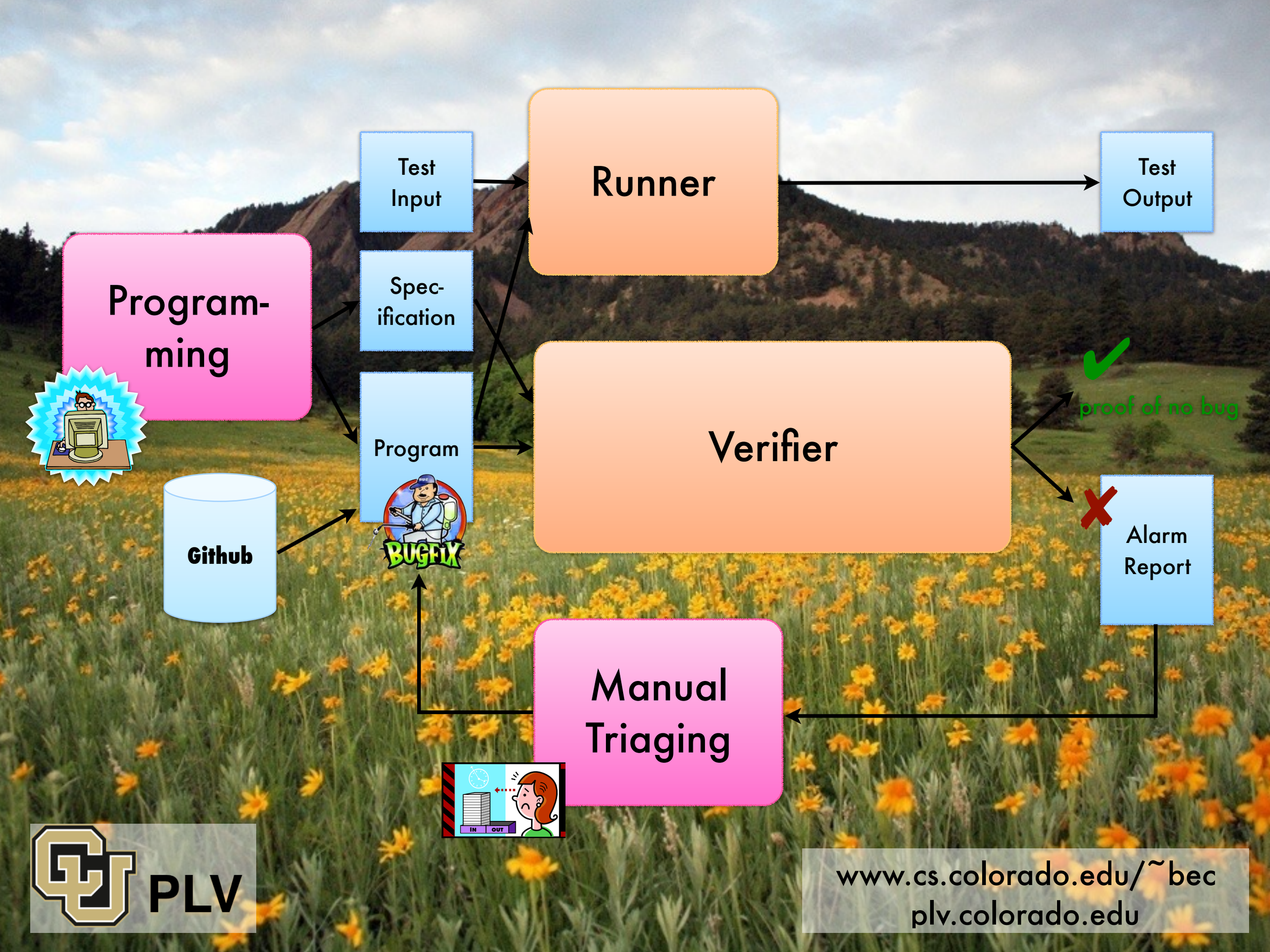
- Unit Test Synthesis: test input + test code to unequivocally show bug
- Evidence for Alarms: alarm explanations, "probability" of bug
- Hardening: synthesize efficient dynamic checks
- Patch Synthesis: synthesize bug fixes
- Input Debugging: bugs in input
- Performance/Scalability/Security Bugs: beyond correctness bugs
- Analysis Engines: in new software domains

Software Domains

- Dynamically-Typed: web, science+engineering
- Concurrent: event-driven systems, user-interactive systems, servers
- Distributed: "big data" software


PLV
www.cs.colorado.edu/~bec
plv.colorado.edu


Program
Verifier
proof of no bug
Alarm Report
PLV
www.cs.colorado.edu/~bec
plv.colorado.edu


Test Input
Runner
Test Output
Program-ming
Spec-ification
Program
Verifier
proof of no bug
Alarm Report
Github
BUGFIX
Manual Triaging
IN
OUT
PLV
www.cs.colorado.edu/~bec
plv.colorado.edu