



# Cooperative Program Analysis


Bor-Yuh Evan Chang

University of Colorado Boulder


Colorado State University
September 22, 2014

PLV



# A program analysis story ...


A380


Software is everywhere and varying more and more

traditional


Software is everywhere and varying more and more

traditional

mobile


Software is everywhere and varying more and more

traditional

mobile

cloud


Software is everywhere and varying more and more

traditional

mobile

cloud


cyberphysical

Software is everywhere and varying more and more

traditional

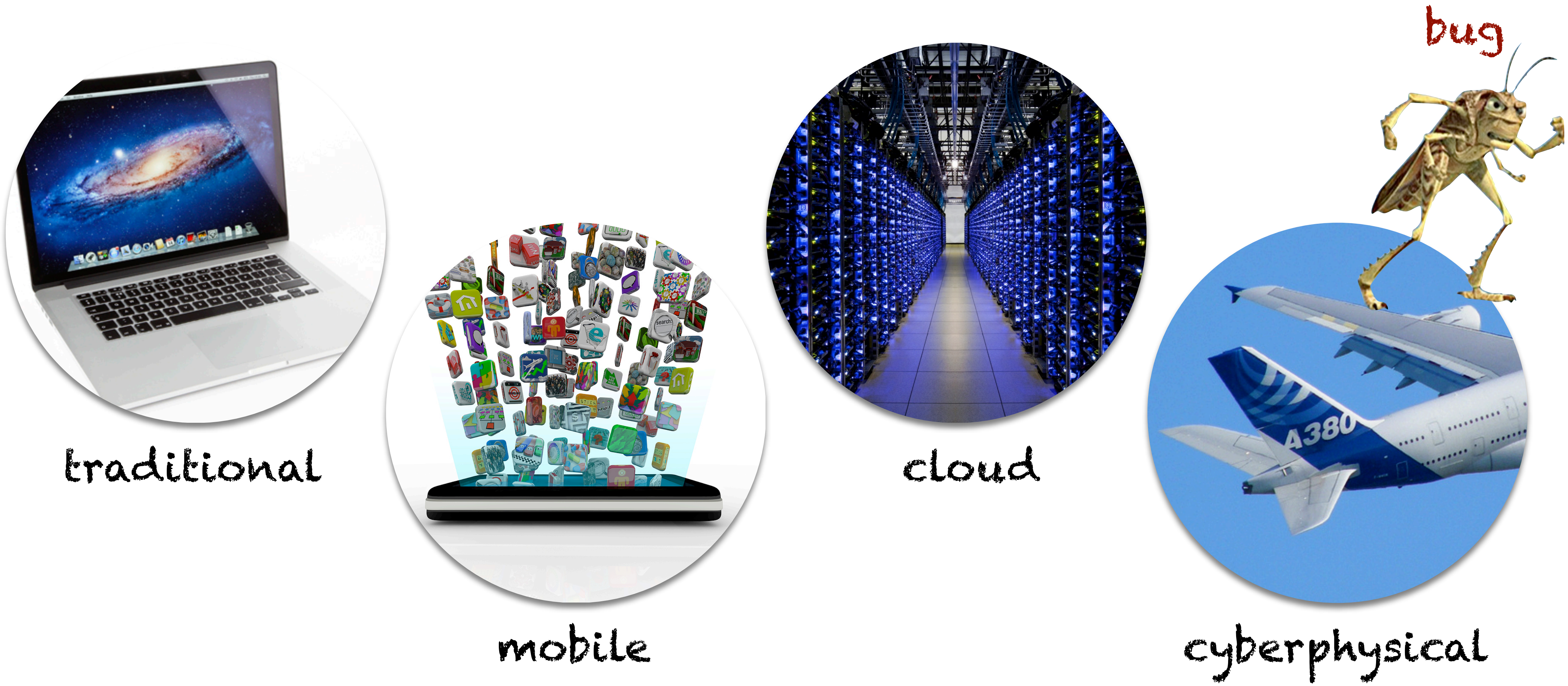
mobile

cloud


cyberphysical

Software is everywhere and varying more and more

Software is getting more and more complex




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


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"

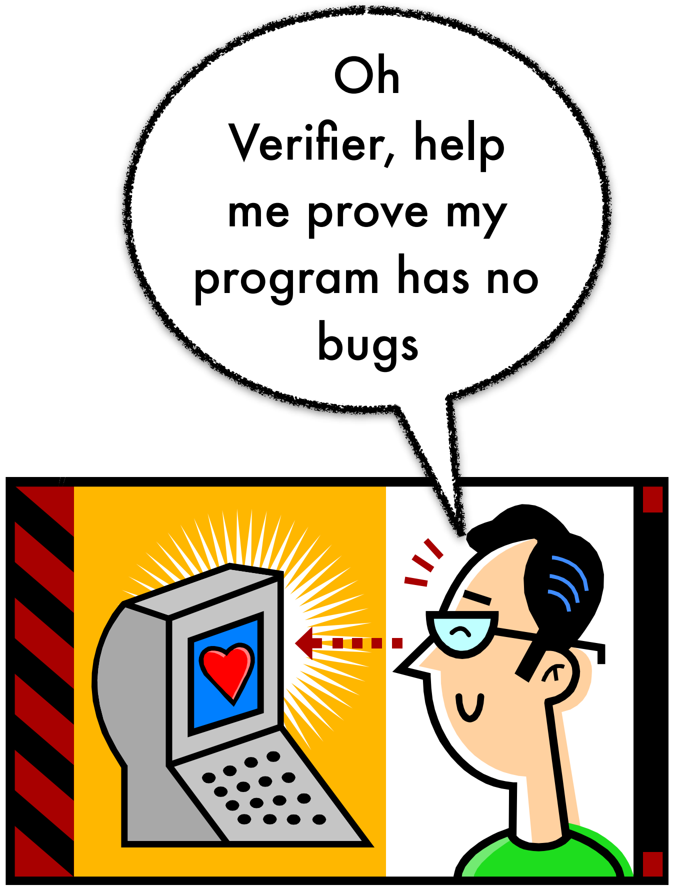
# The Ugly, Hidden Truth


## Program Analysis for Formal Verification

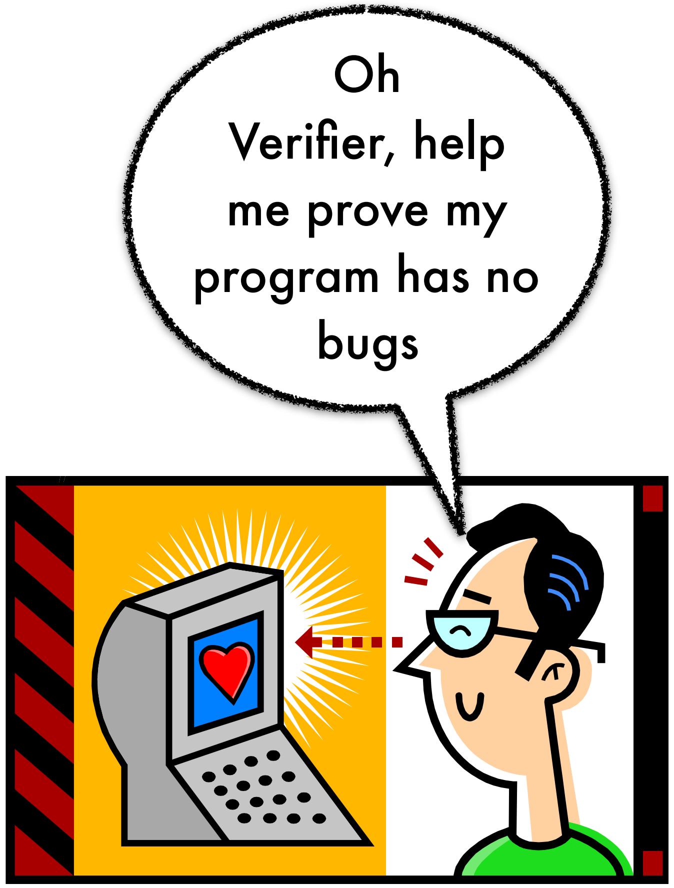

Undecidability necessitates the possibility of **false alarms**. We hope not too many.

# Uncooperative Program Analysis?

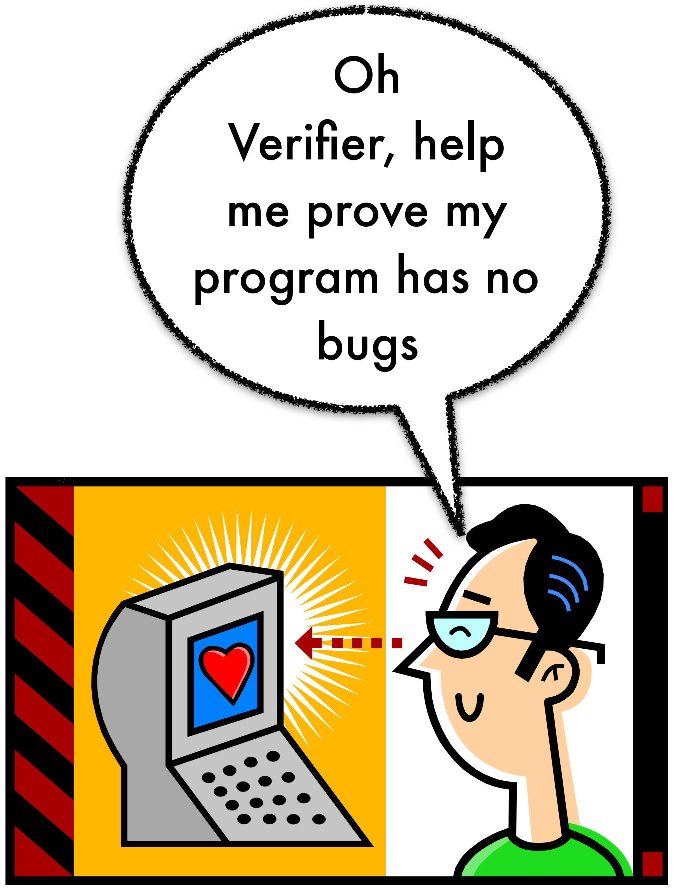
# Uncooperative Program Analysis?



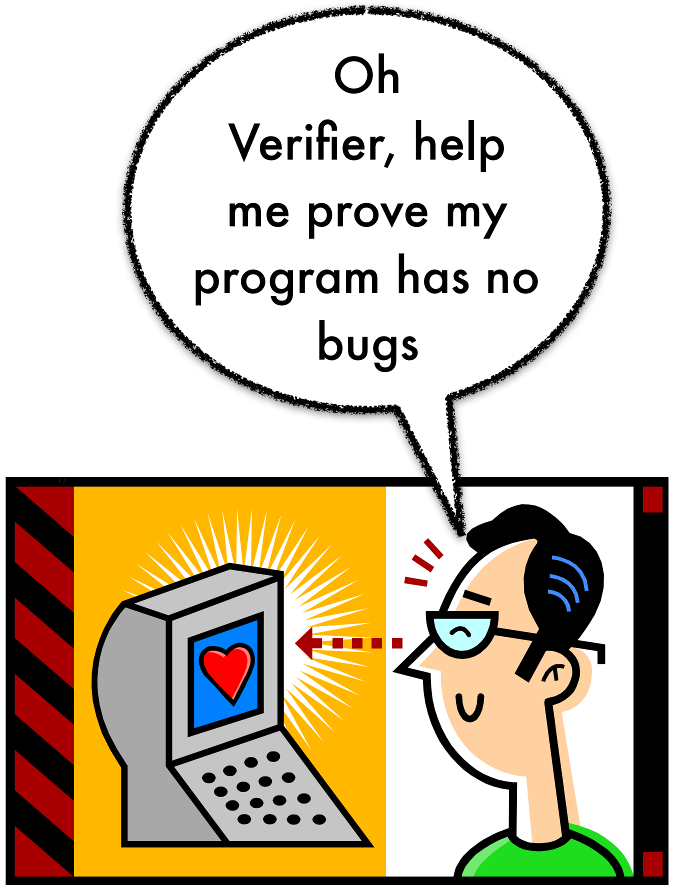
# Uncooperative Program Analysis?

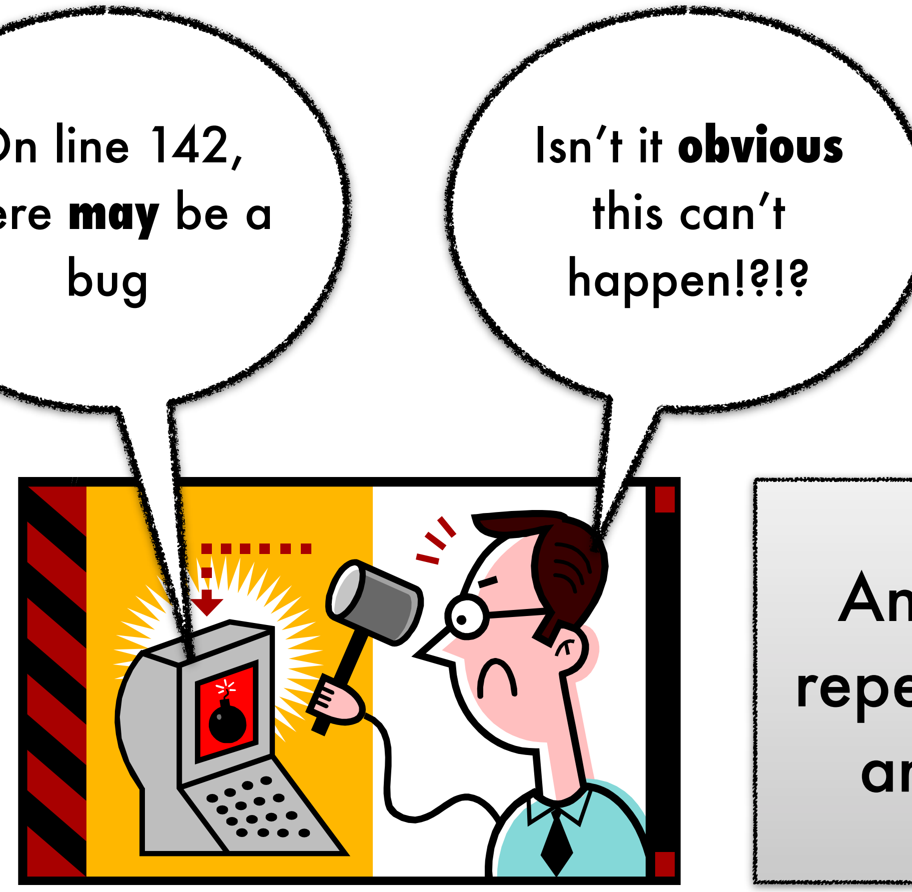



# Uncooperative Program Analysis?



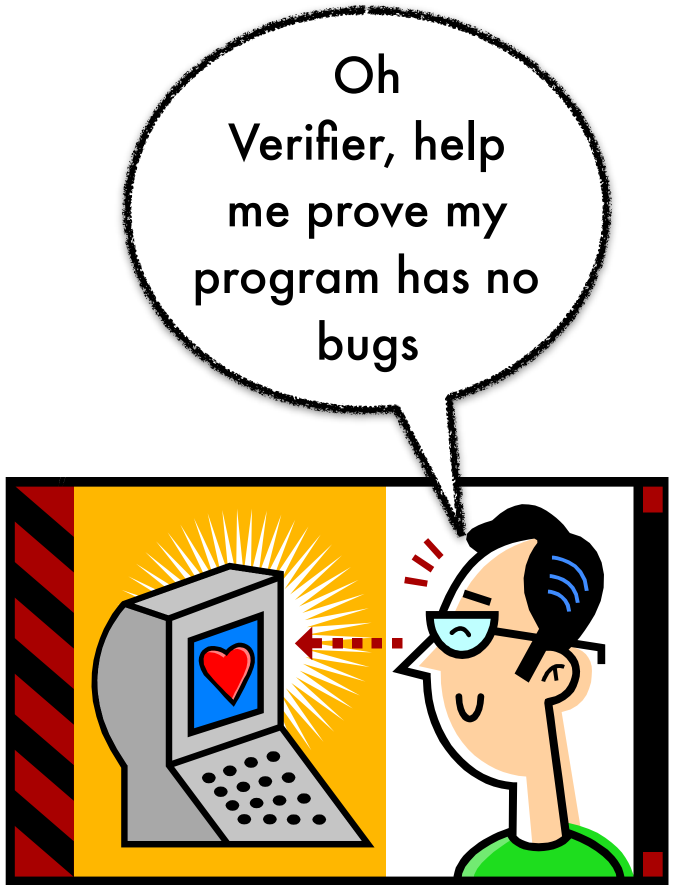

# Uncooperative Program Analysis?

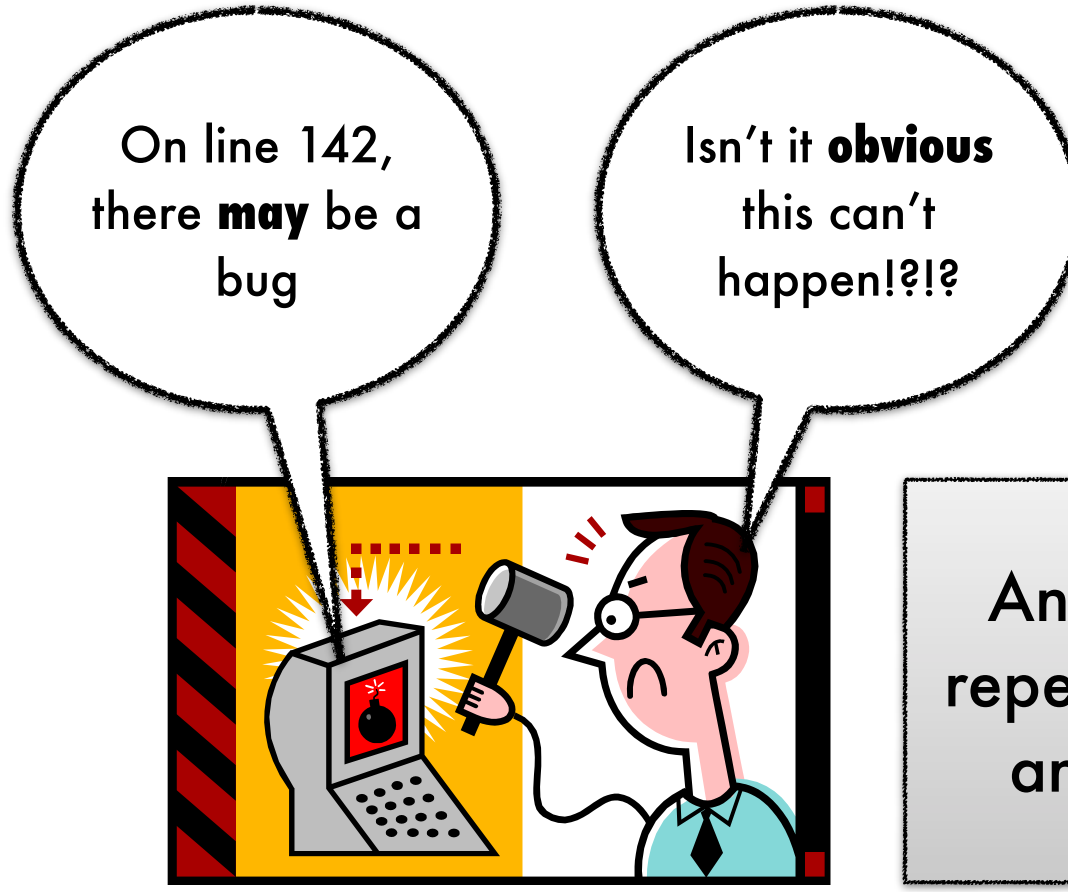



And noisily repeated over and over!

# Uncooperative Program Analysis?

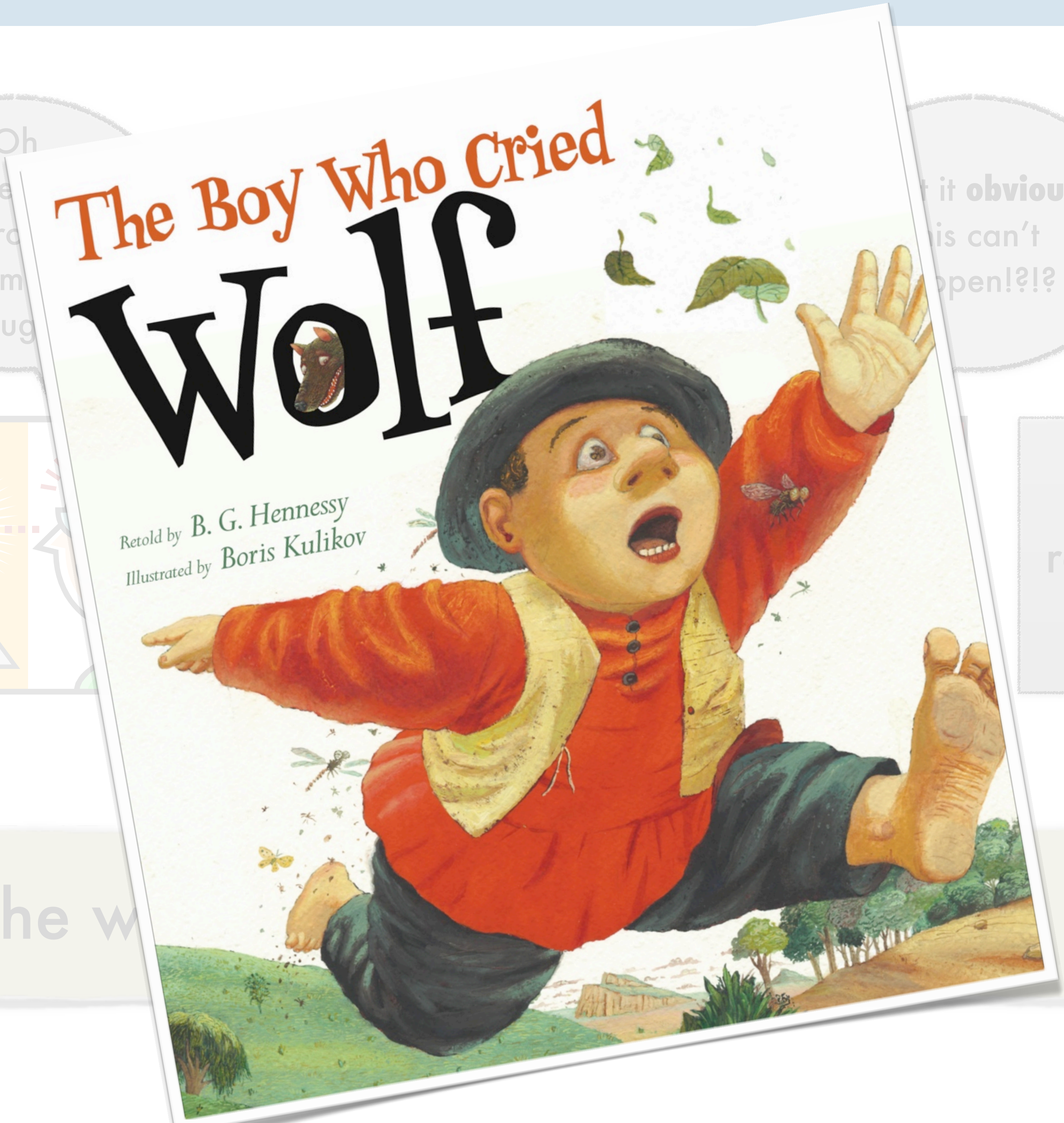



And noisily repeated over and over!

The well-known **false alarm** problem

# Uncooperative Program Analysis?




And noisily repeated over and over!

"[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."

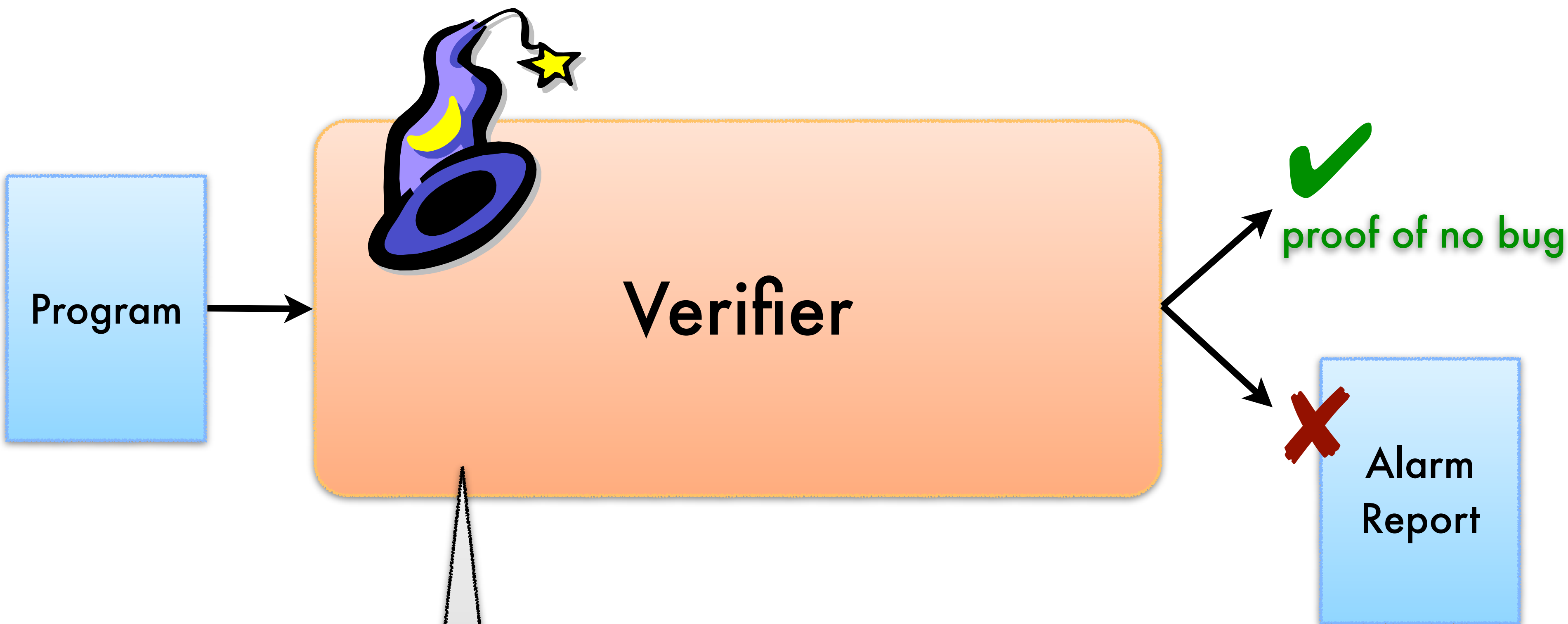
"A stupid false [alarm] implies the tool is stupid."

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®

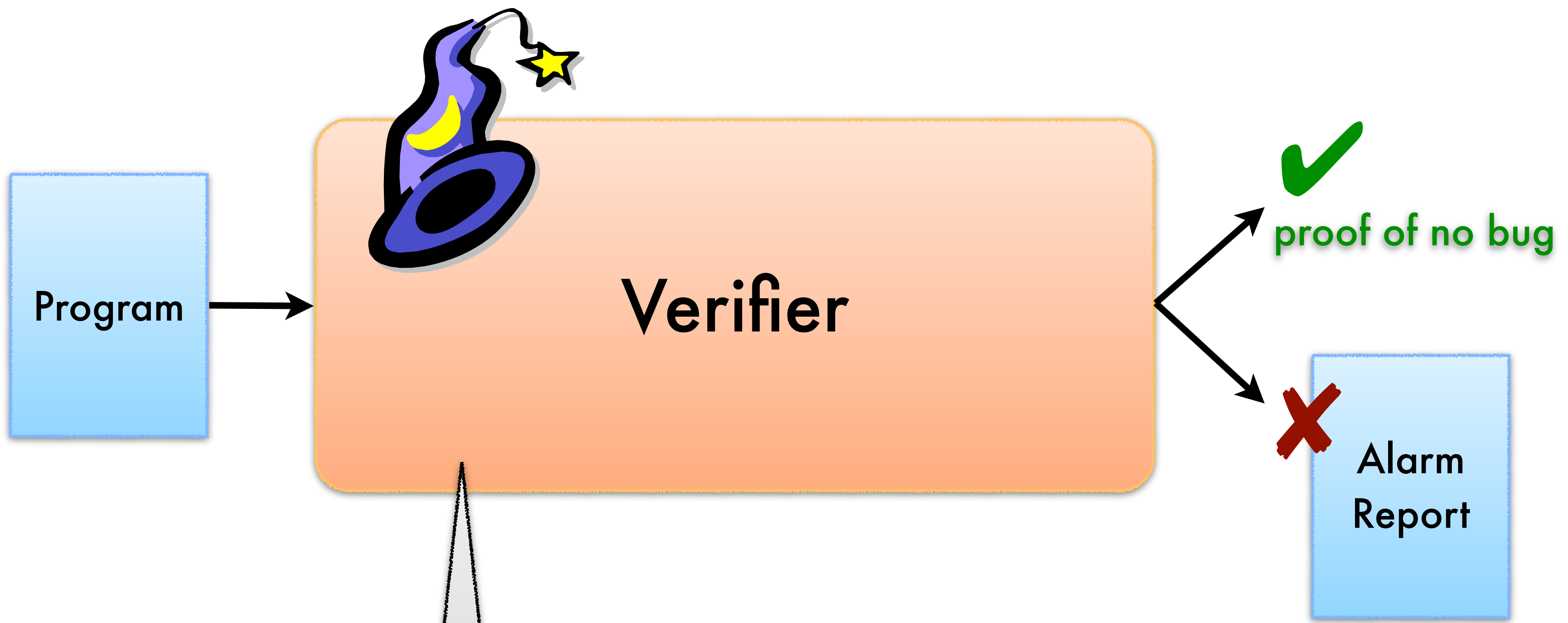
# The traditional approach to the false alarm problem focuses on improving the verifier.

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




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

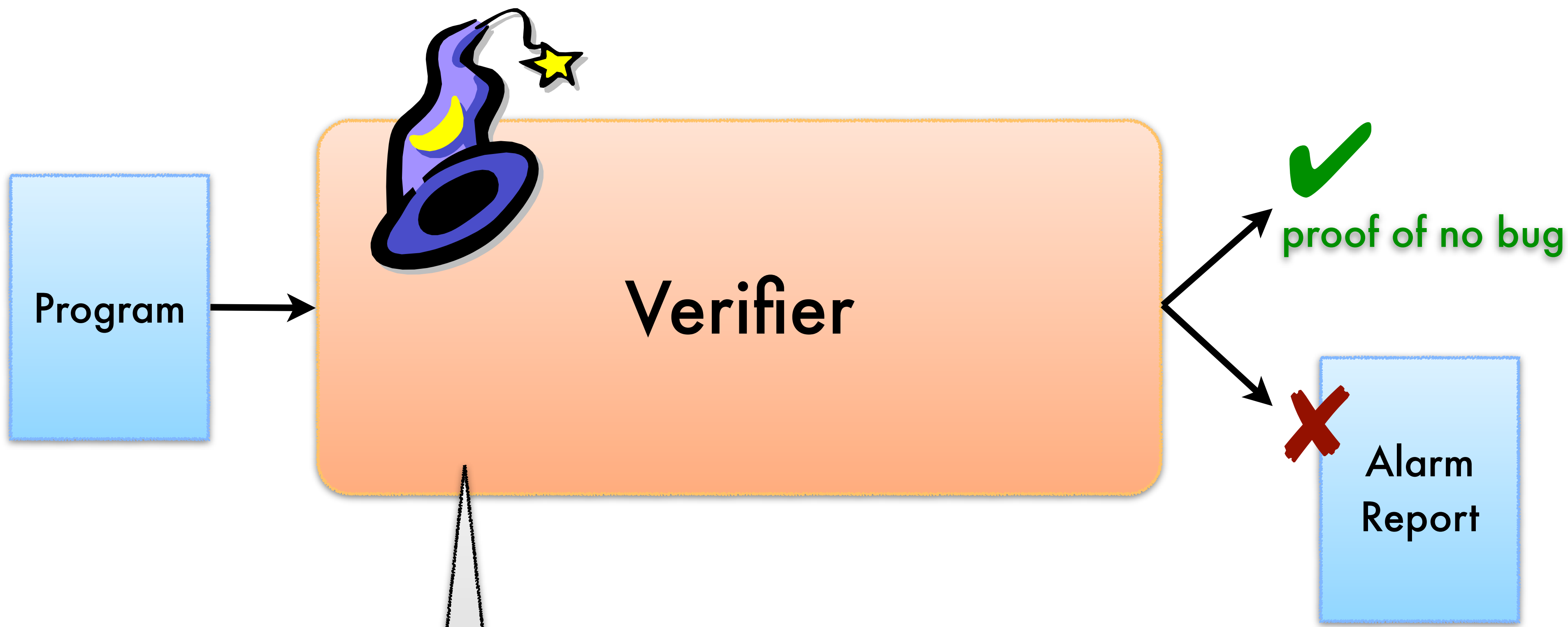
# The traditional approach to the false alarm problem focuses on improving the verifier.




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

But it can never be perfect (undecidability)

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



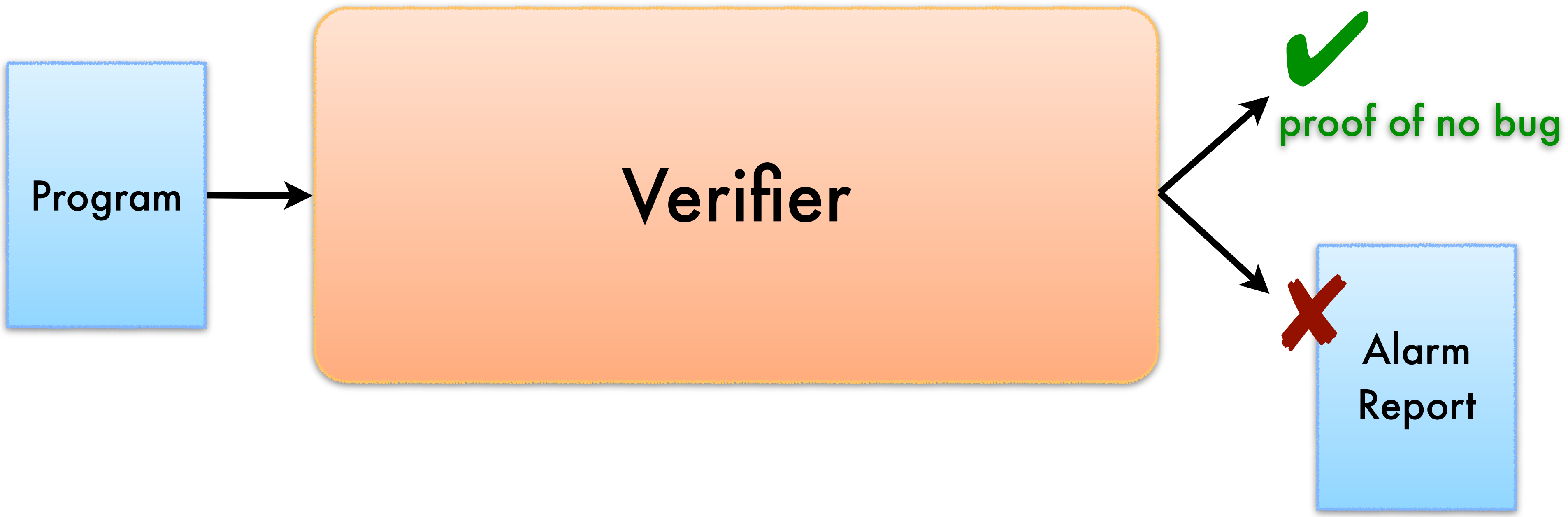

**Redesign the verifier** with more magic to hopefully reduce the number of false alarms

But it can never be perfect (undecidability)

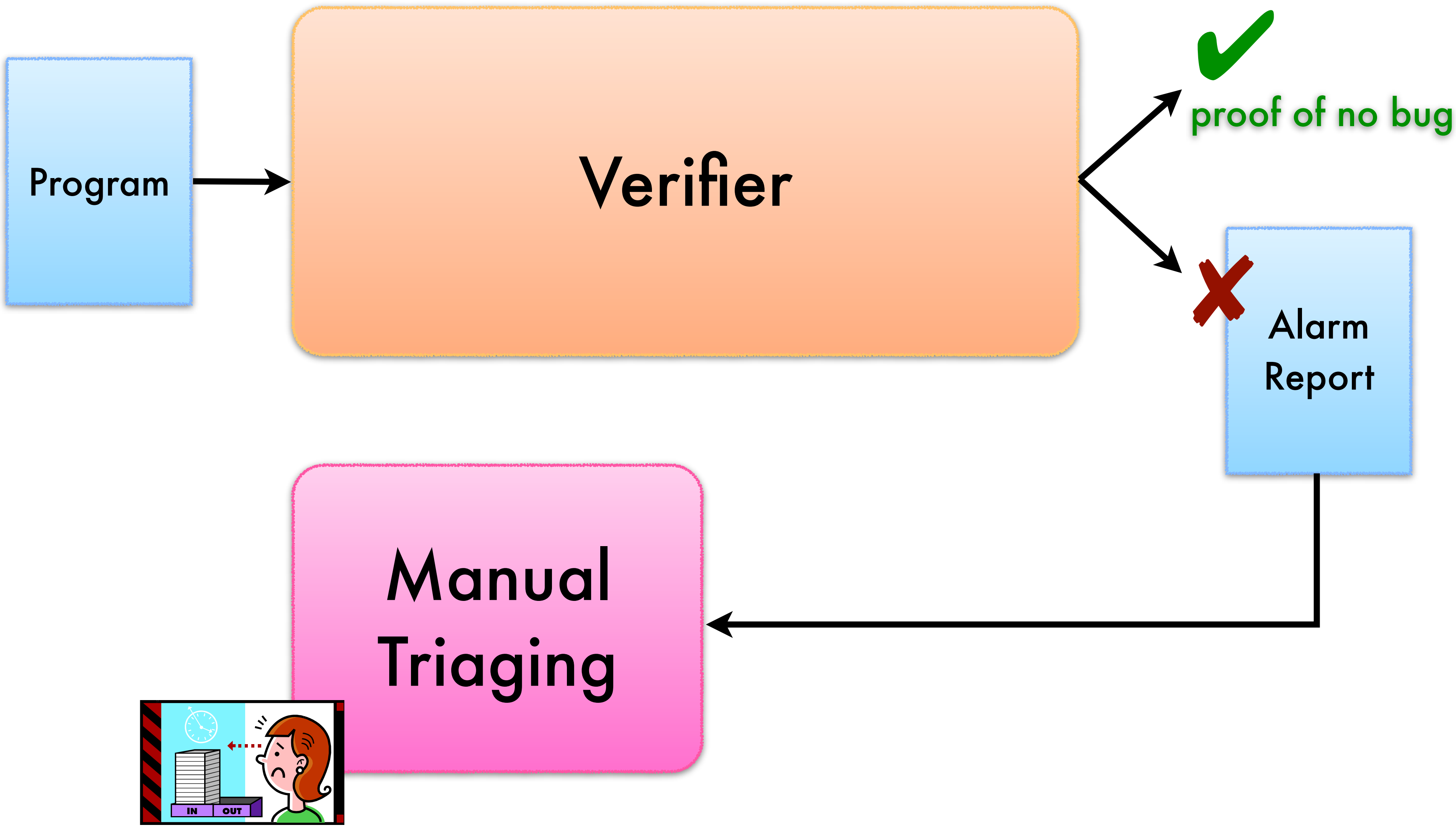
Also not a sufficient "excuse"

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

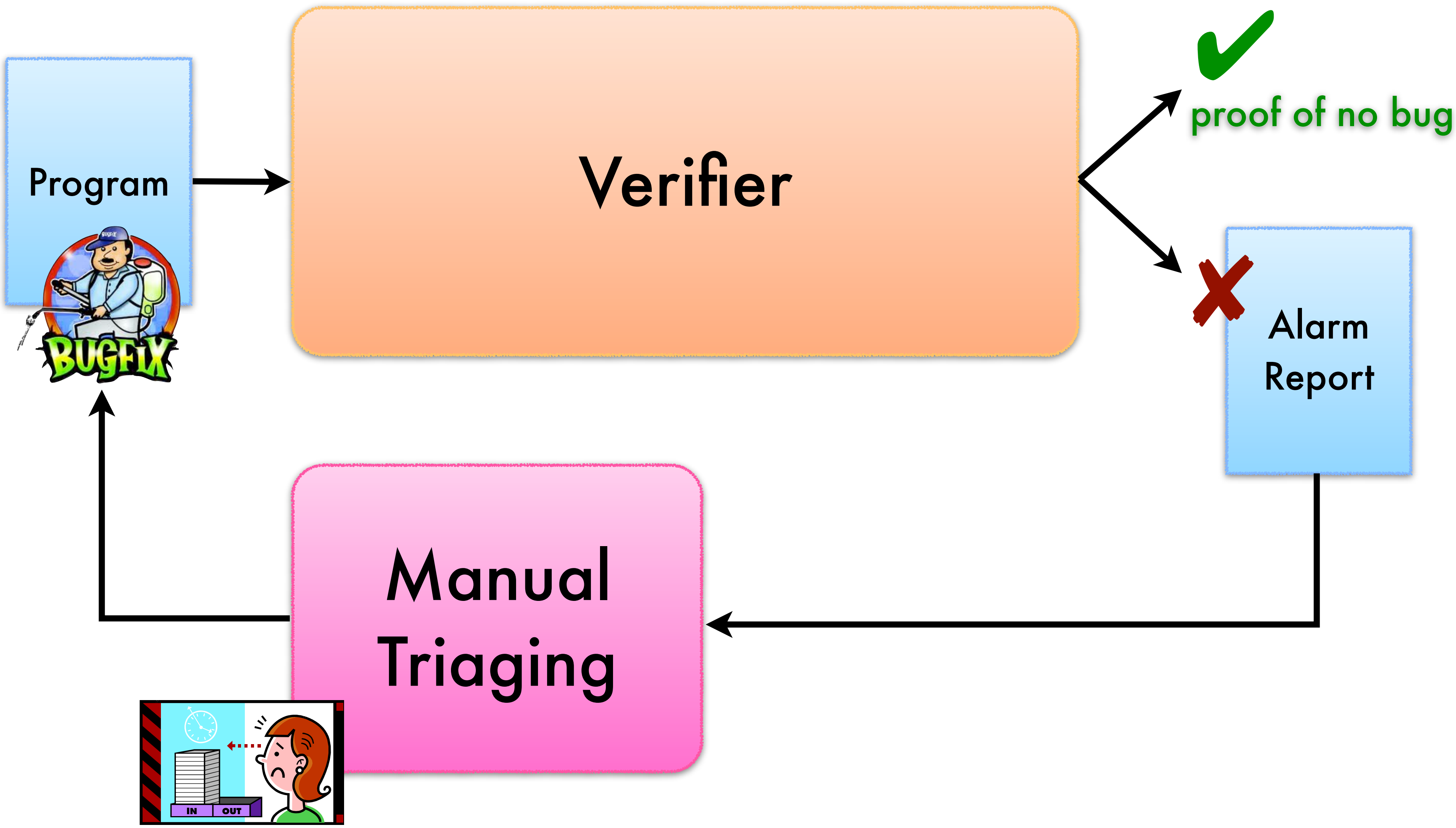
# Agenda: The cooperative approach addresses the whole bug mitigation process.



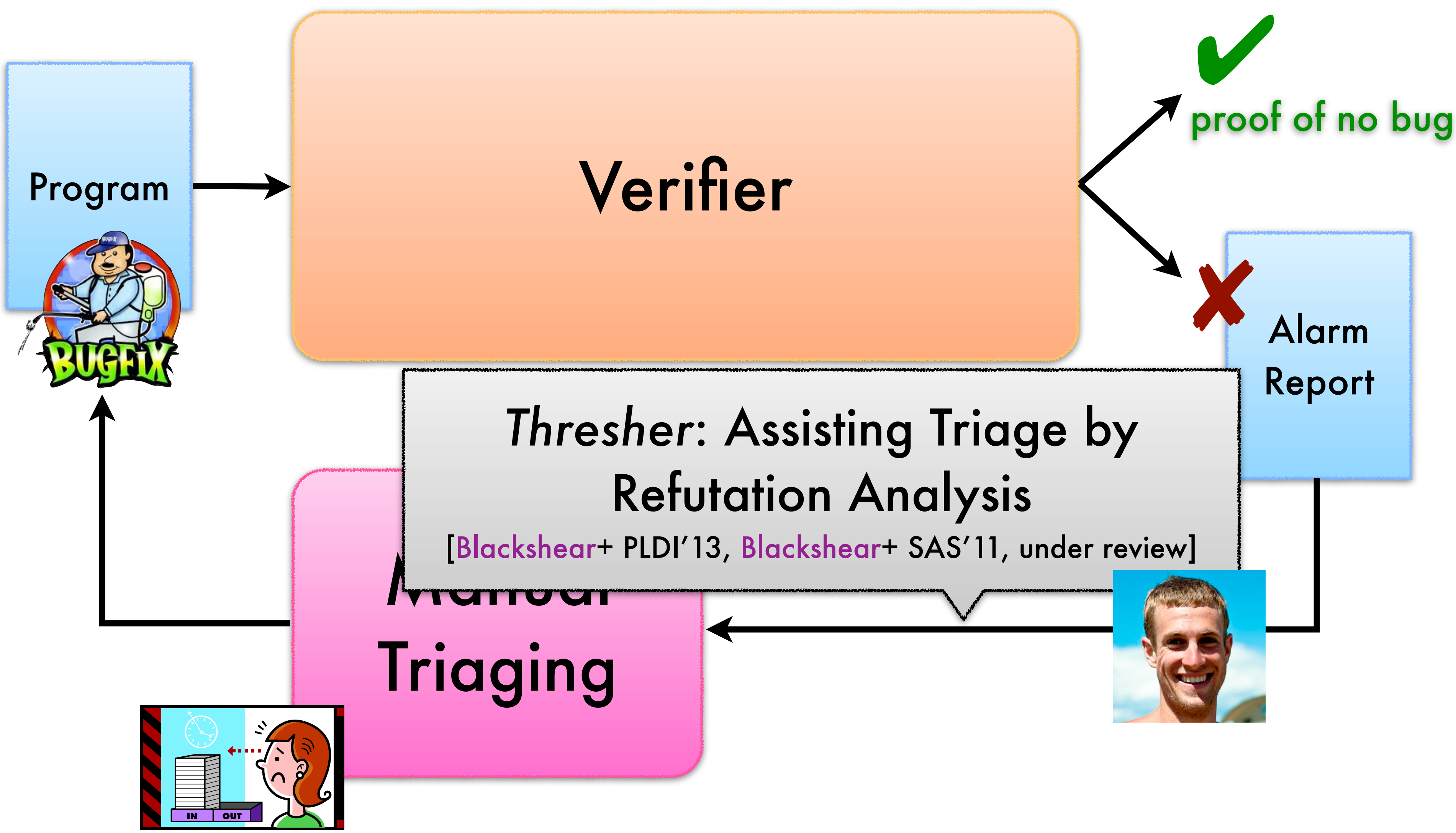
# Agenda: The cooperative approach addresses the whole bug mitigation process.



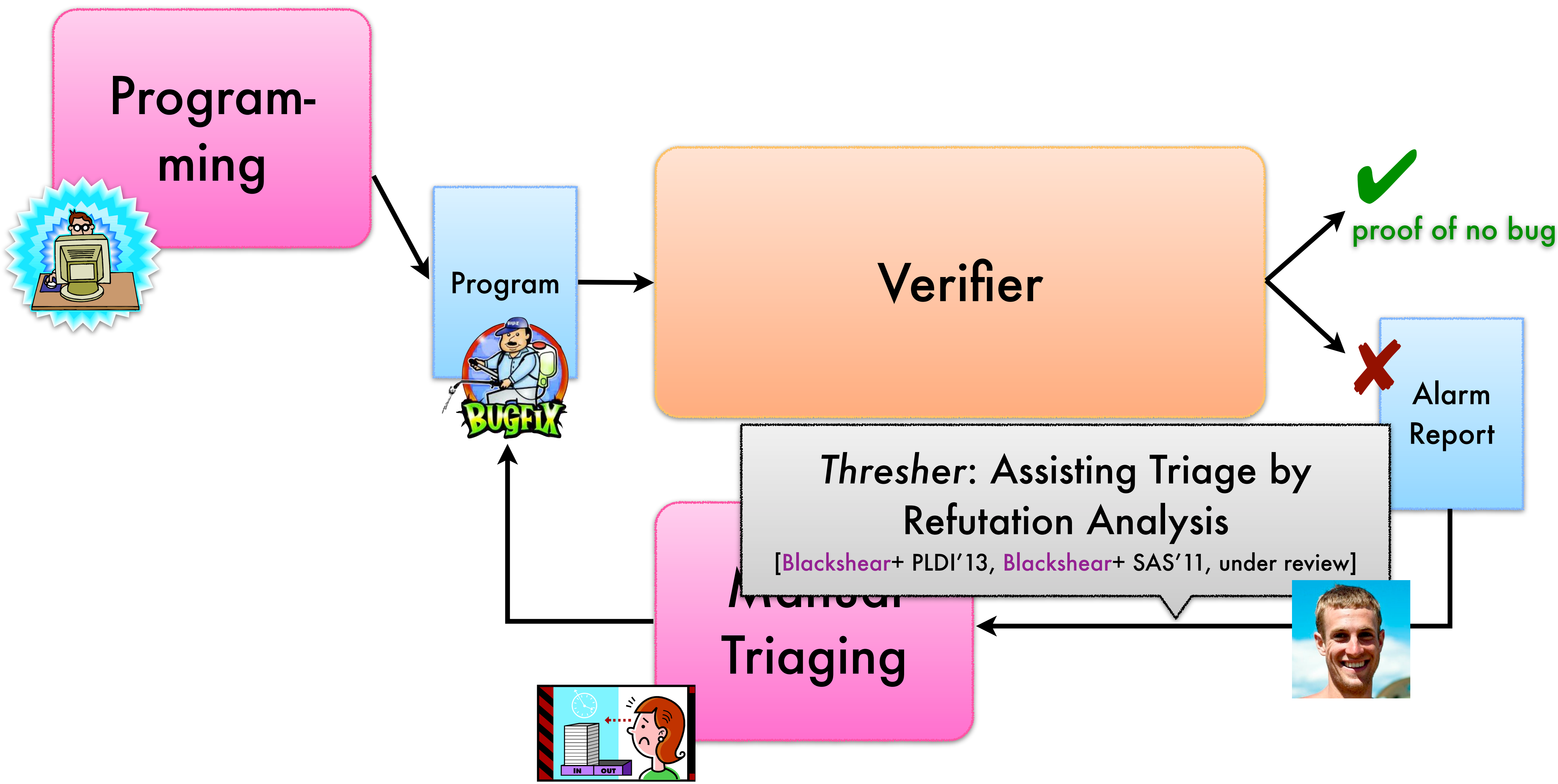
# Agenda: The cooperative approach addresses the whole bug mitigation process.



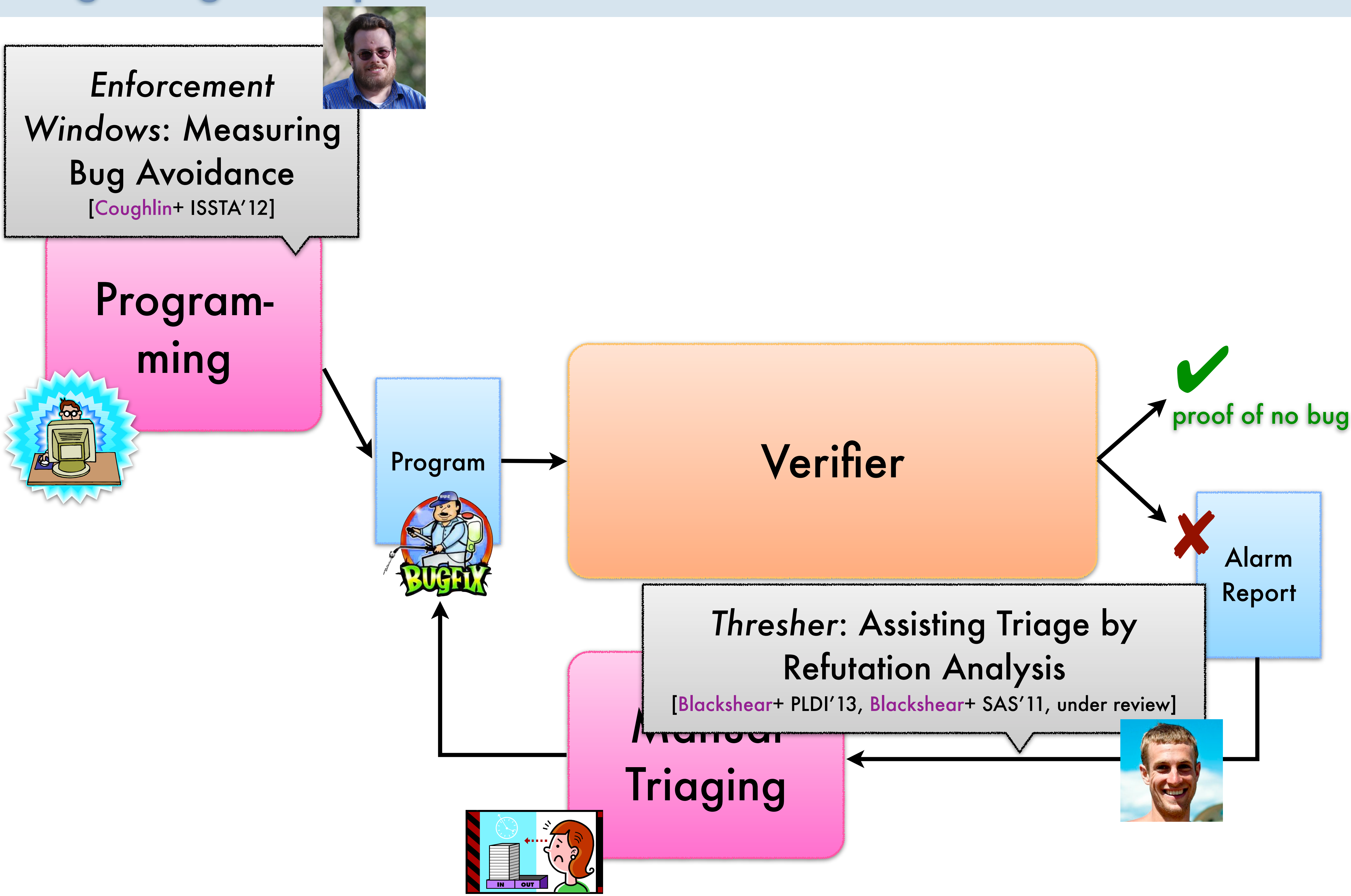
# Agenda: The cooperative approach addresses the whole bug mitigation process.



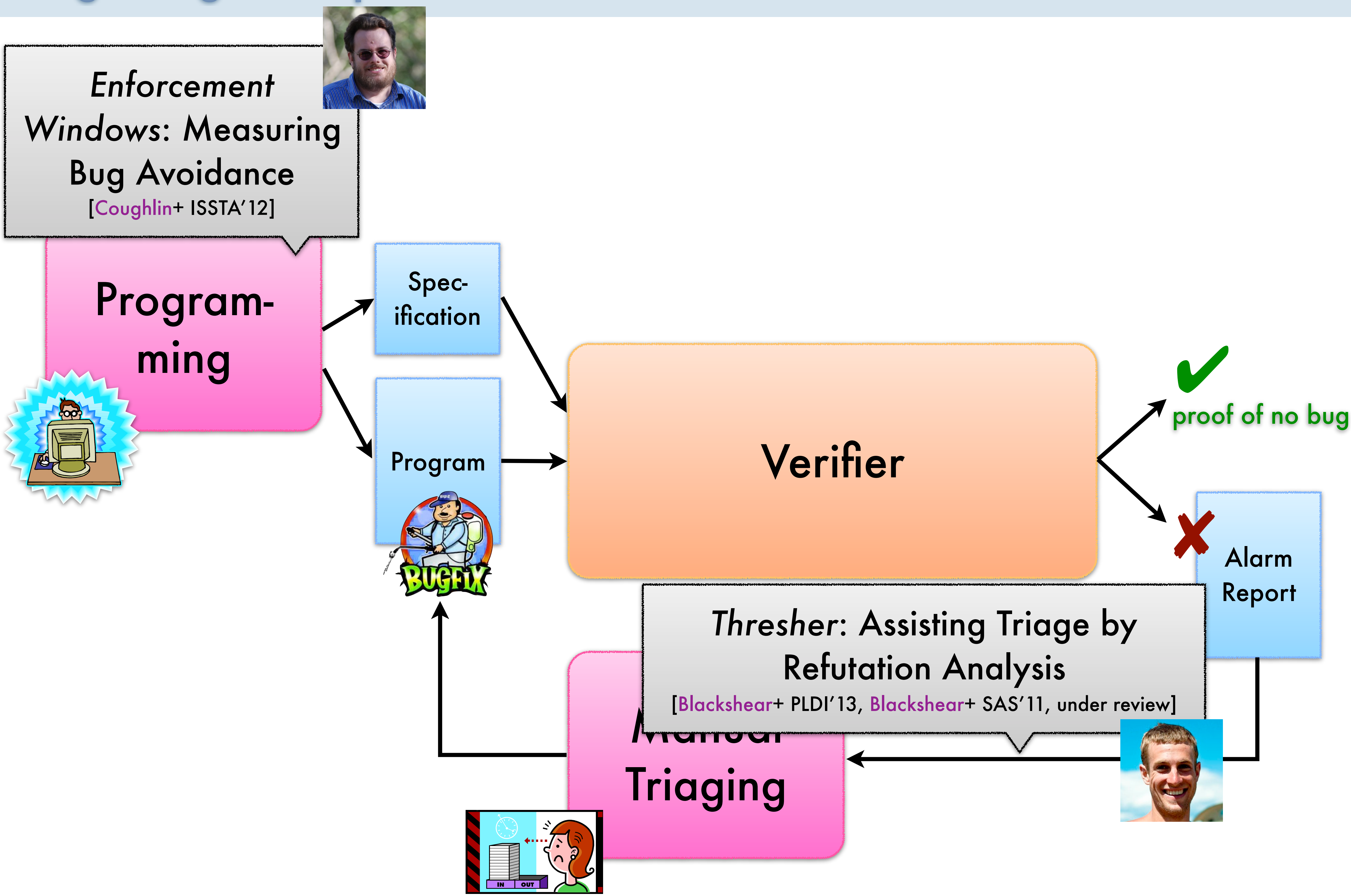
# Agenda: The cooperative approach addresses the whole bug mitigation process.



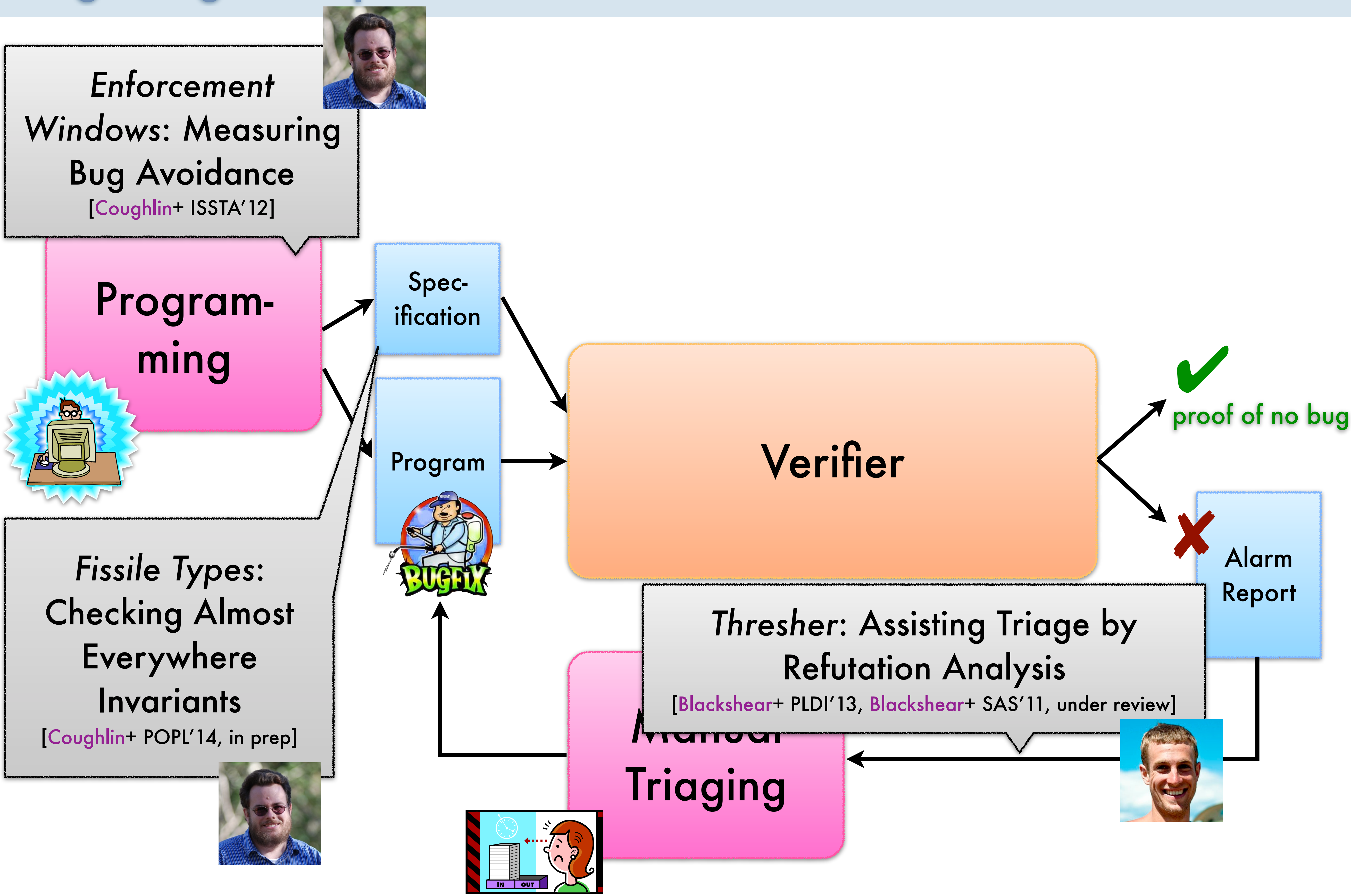
# Agenda: The cooperative approach addresses the whole bug mitigation process.



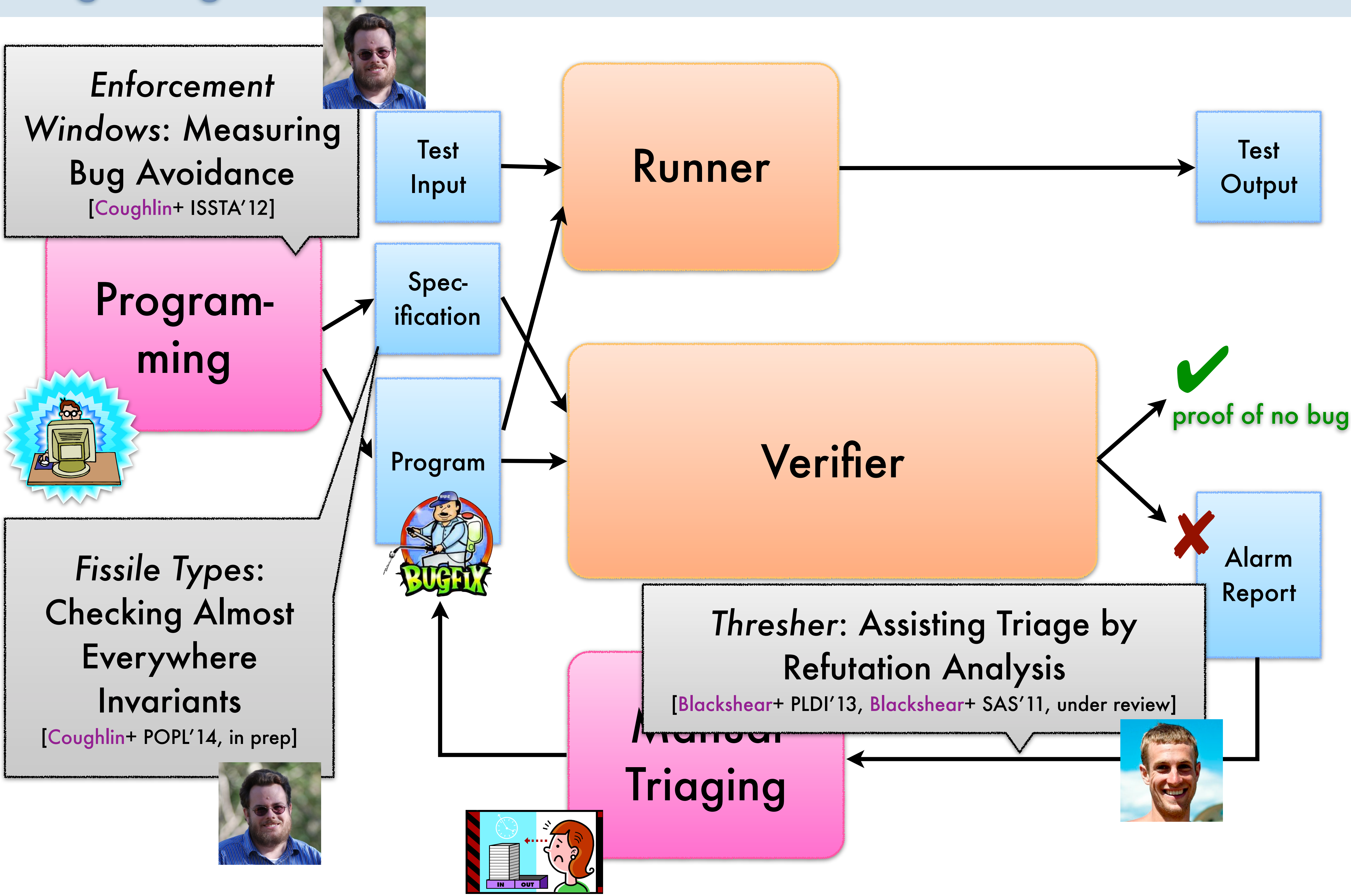
# Agenda: The cooperative approach addresses the whole bug mitigation process.



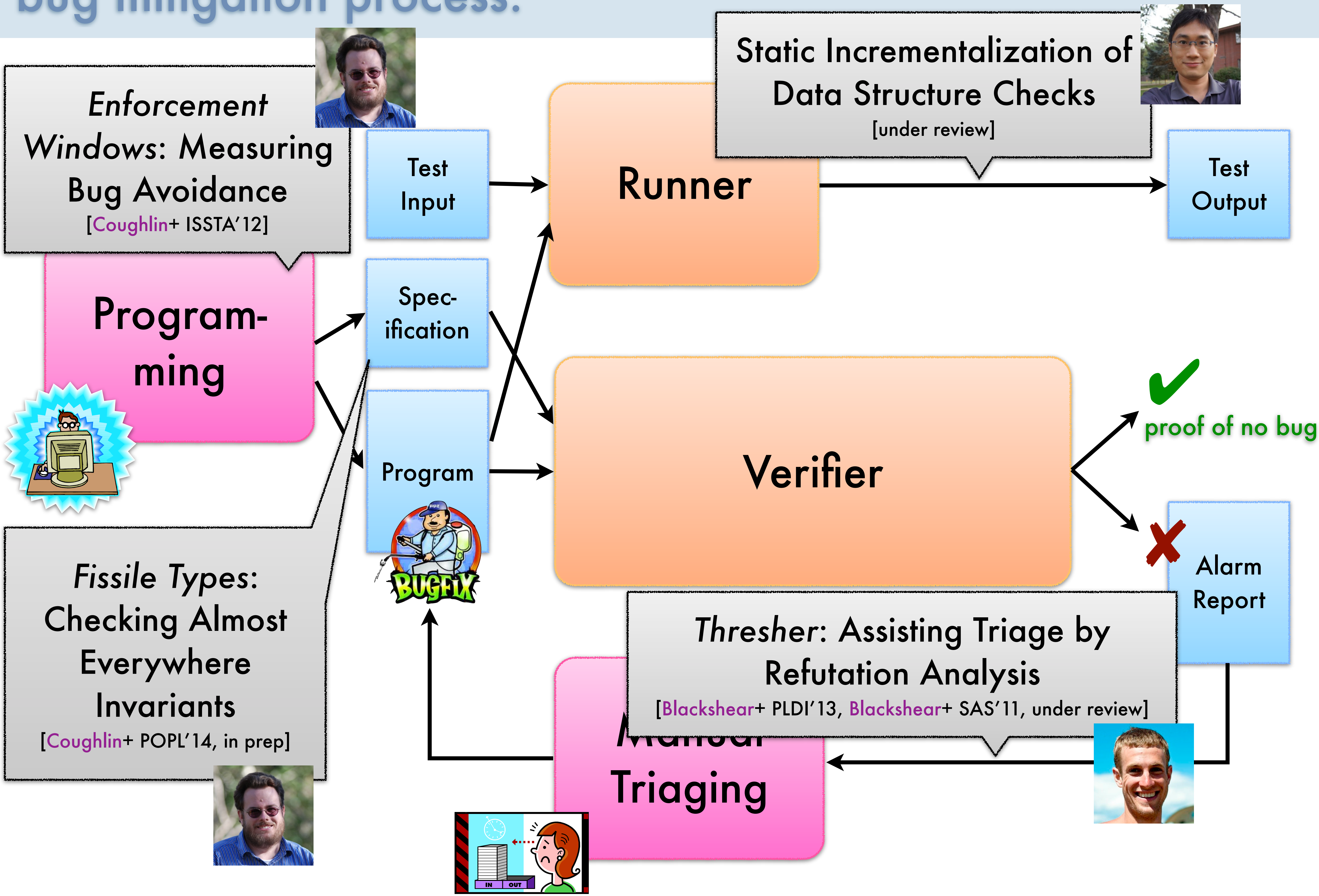
# Agenda: The cooperative approach addresses the whole bug mitigation process.



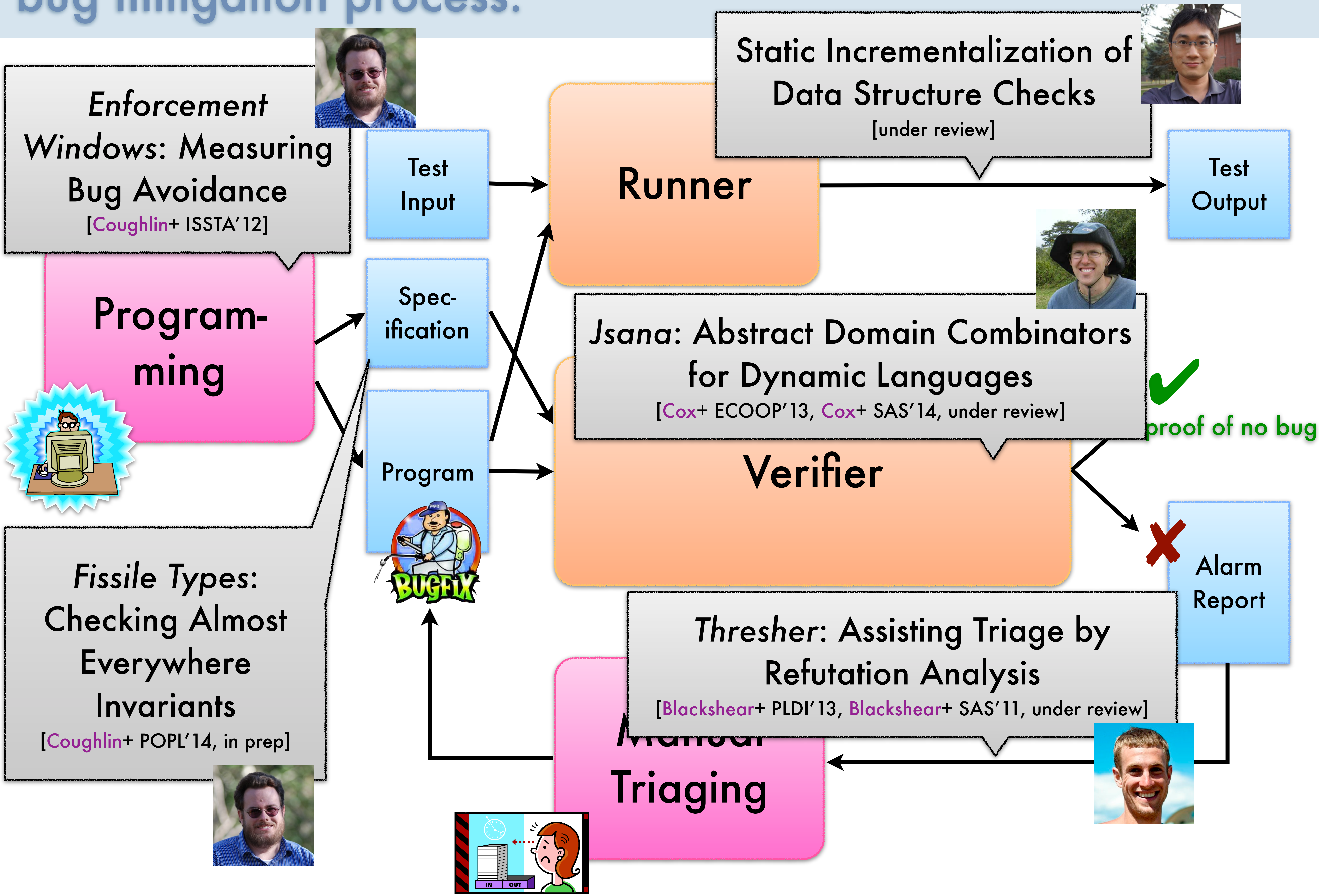
# Agenda: The cooperative approach addresses the whole bug mitigation process.



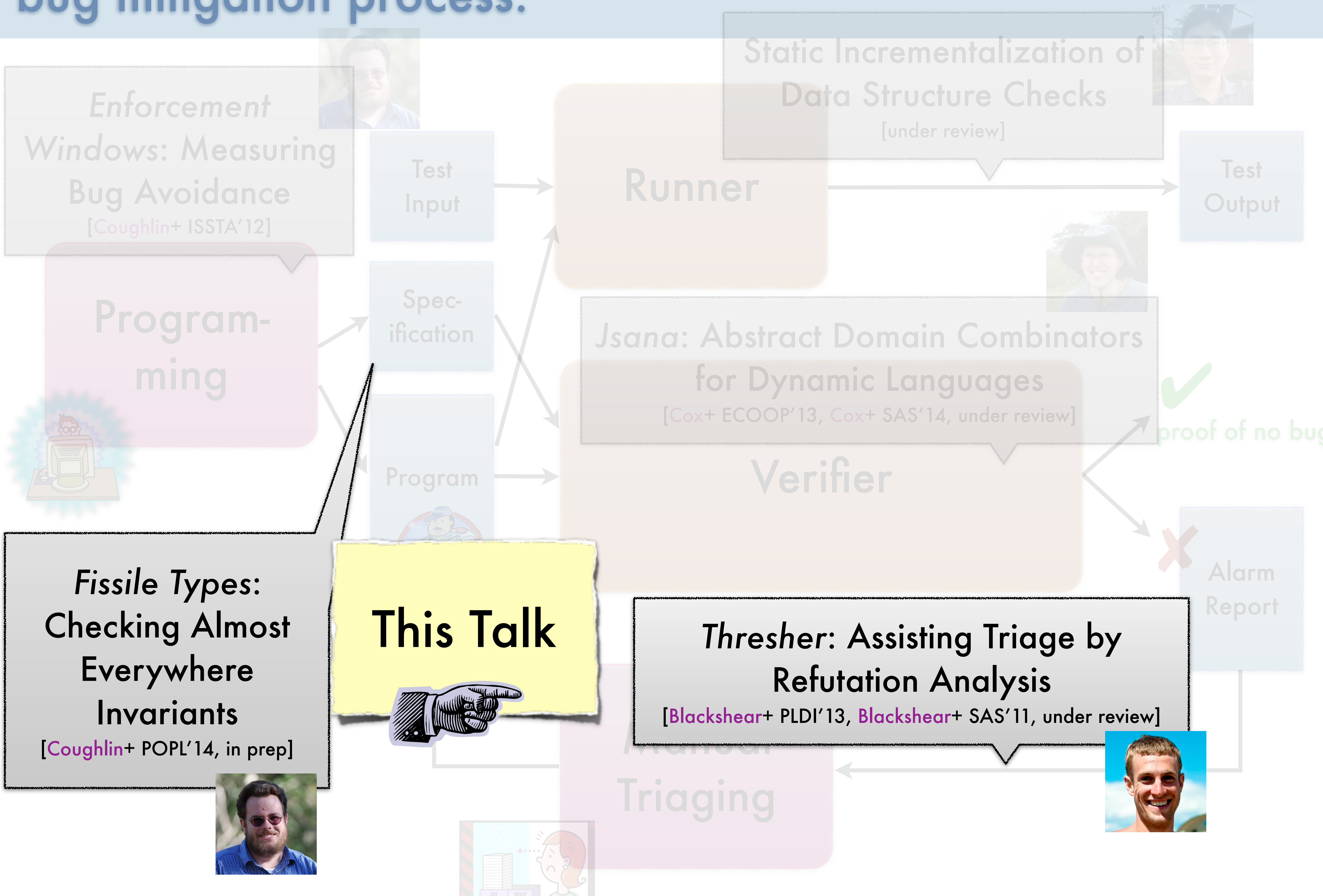
# Agenda: The cooperative approach addresses the whole bug mitigation process.



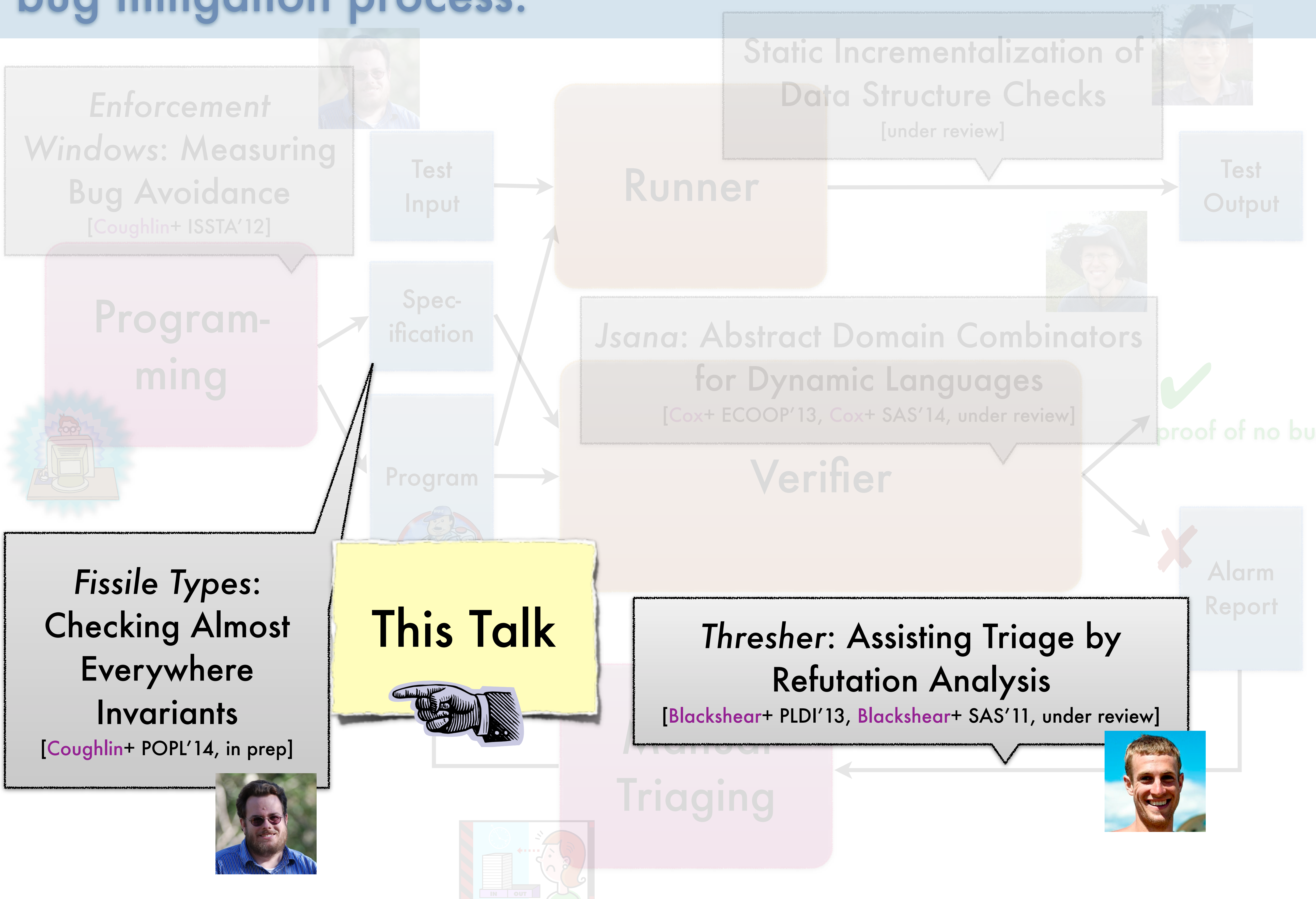
# Agenda: The cooperative approach addresses the whole bug mitigation process.



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



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



# This Talk: Highlights

Thresher: Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations

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

# This Talk: Highlights

Thresher: Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations

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

Fissile Types: Checking Reflection with Almost Everywhere Invariants

Strengthen type checking with symbolic analysis

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

# This Talk: Highlights

Thresher: Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations

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

Fissile Types: Checking Reflection with Almost Everywhere Invariants

Strengthen type checking with symbolic analysis

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

# Thresher: Precise Refutations for Heap Reachability

# What are heap reachability queries?

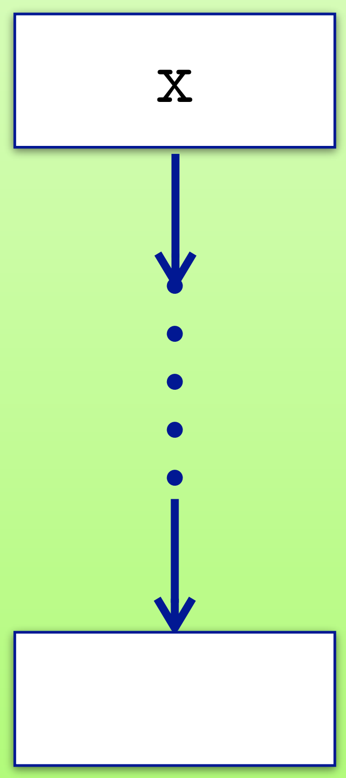
# What are heap reachability queries?

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

# What are heap reachability queries?

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

Is there a program execution where at some time variable

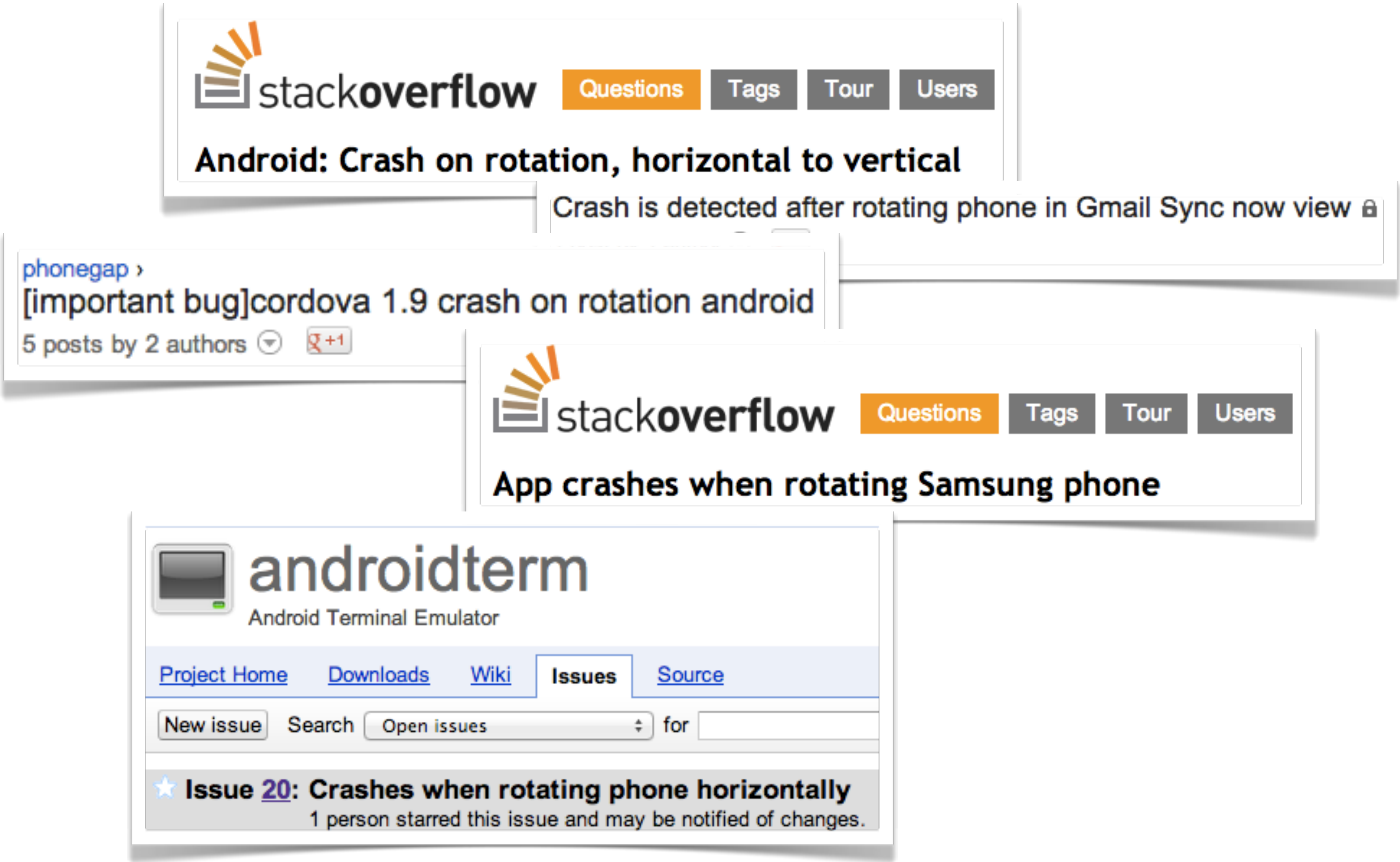



of type T ?

Example

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

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



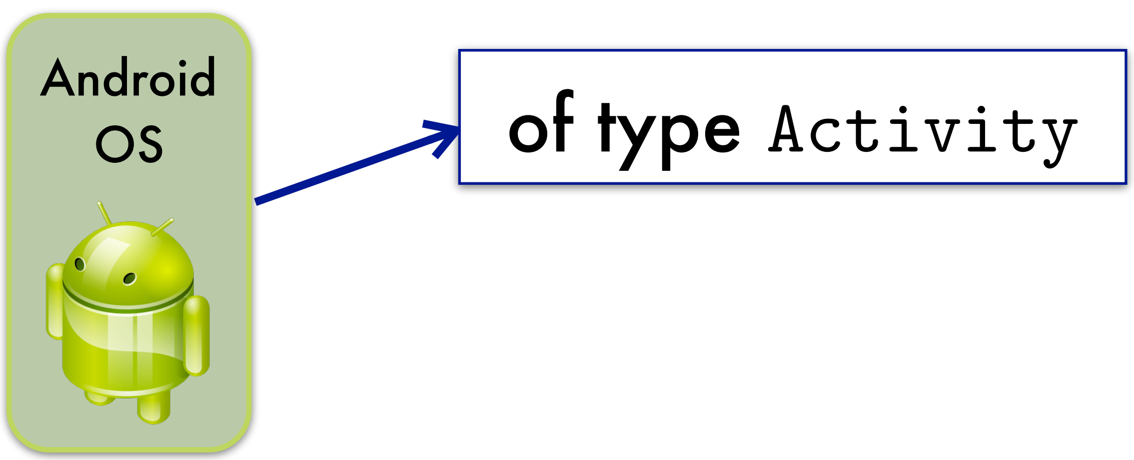
# How is this useful? We identify memory leaks that cause your app to crash!

# Android memory leaks underly rotation-based crashes.

`Activity` objects encapsulate the UI

# Android memory leaks underly rotation-based crashes.

`Activity` objects encapsulate the UI




# Android memory leaks underly rotation-based crashes.

`Activity` objects encapsulate the UI

Android OS

of type `Activity`

# Android memory leaks underly rotation-based crashes.

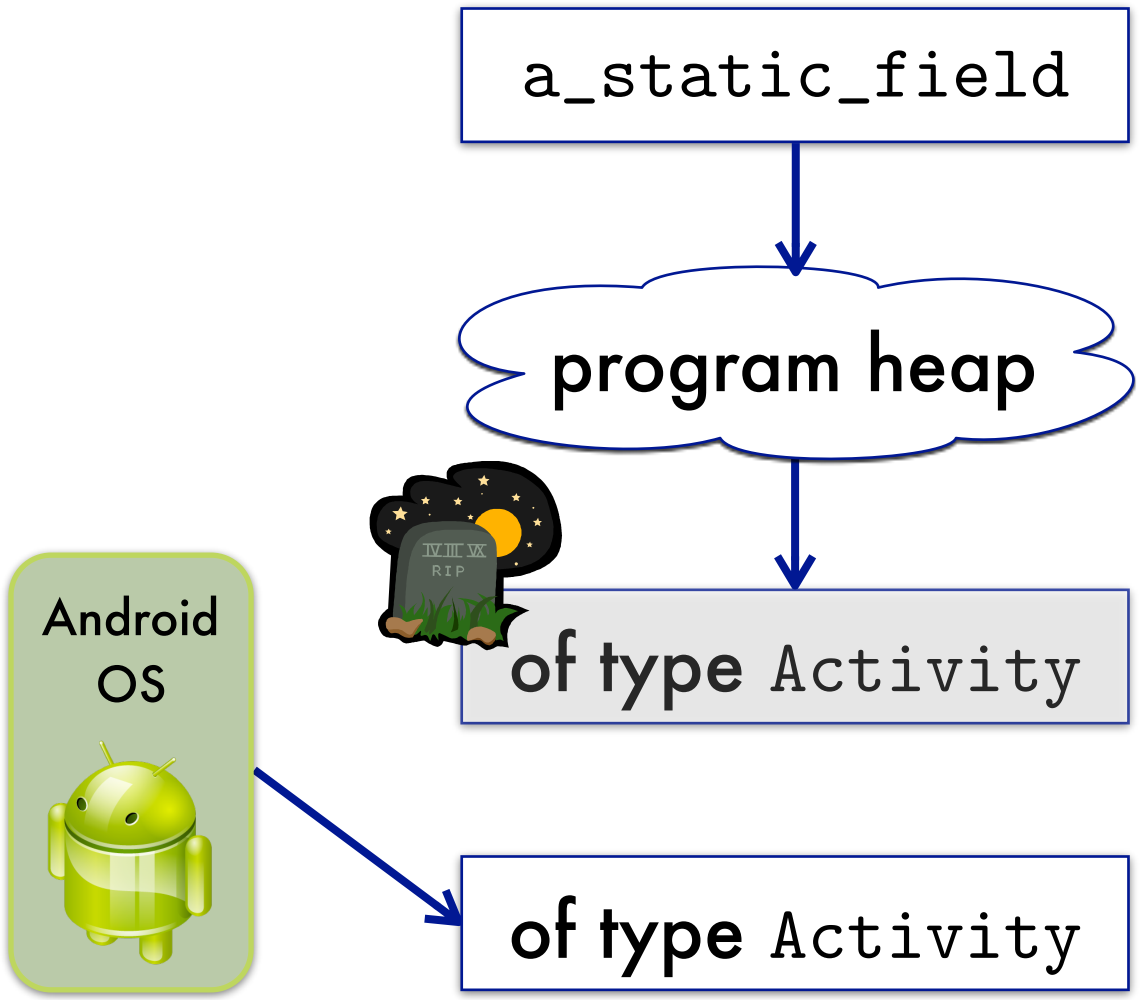
`Activity` objects encapsulate the UI


of type `Activity`

Android OS

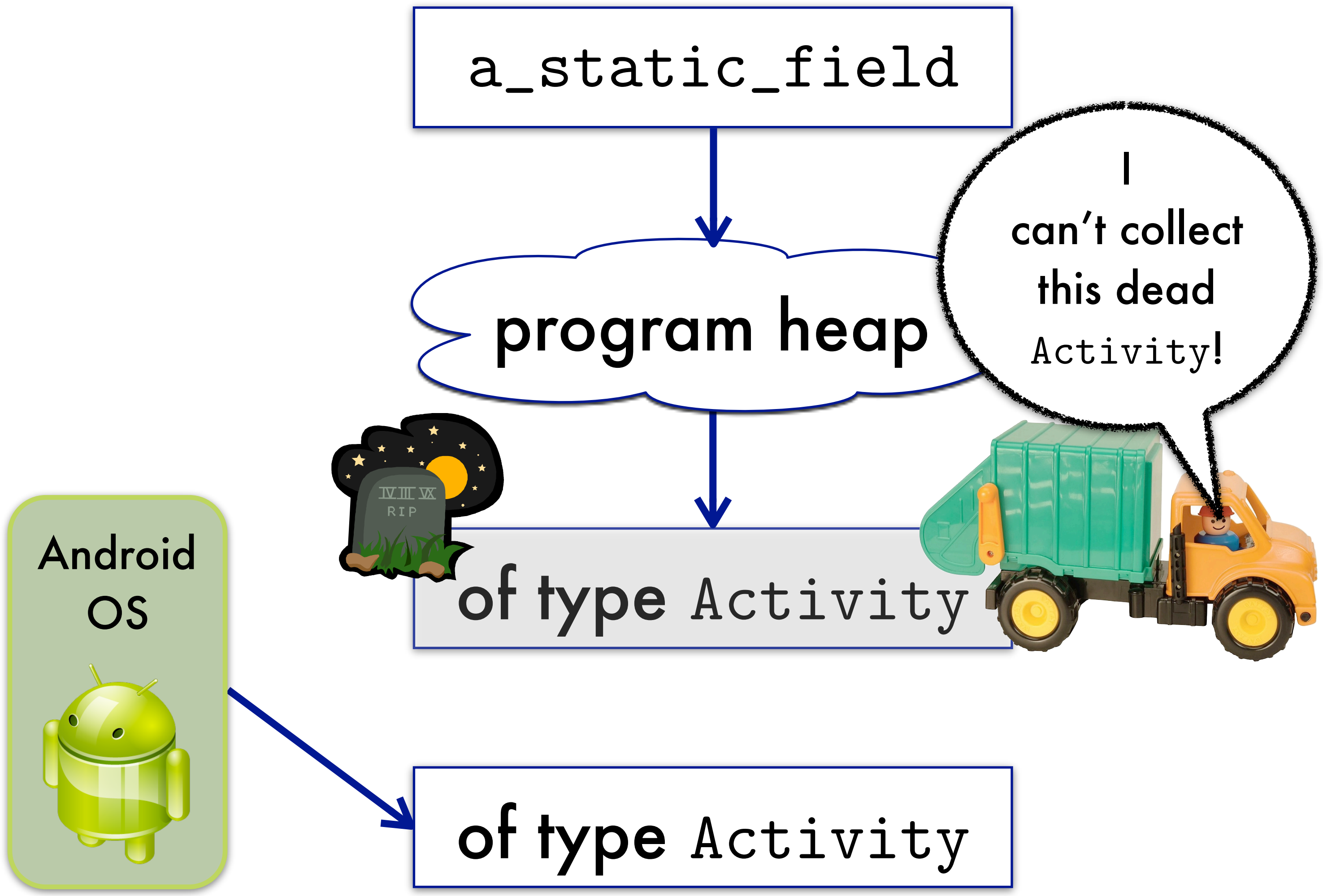
of type `Activity`

# Android memory leaks underly rotation-based crashes.



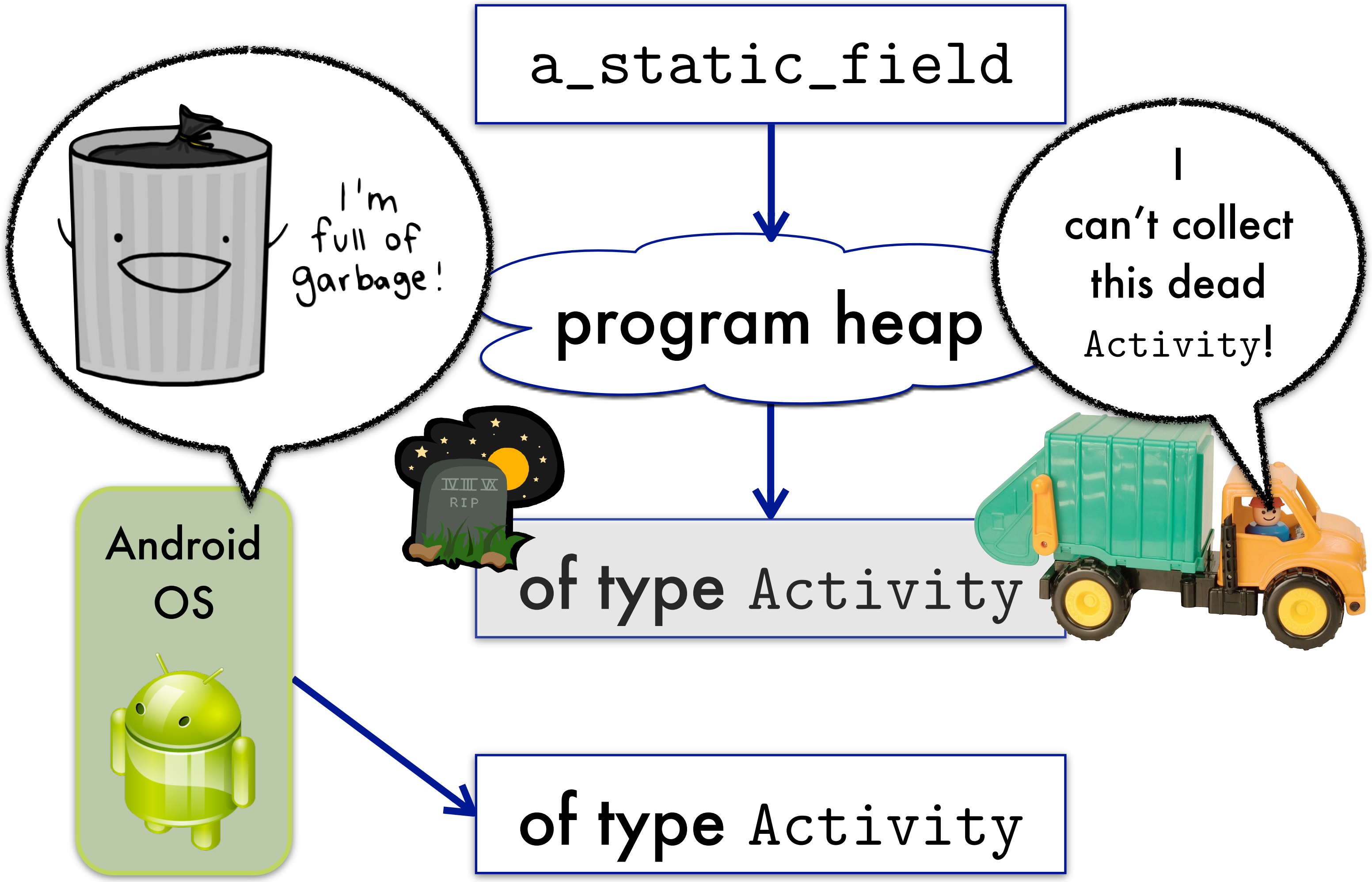

`Activity` objects encapsulate the UI

# Android memory leaks underly rotation-based crashes.



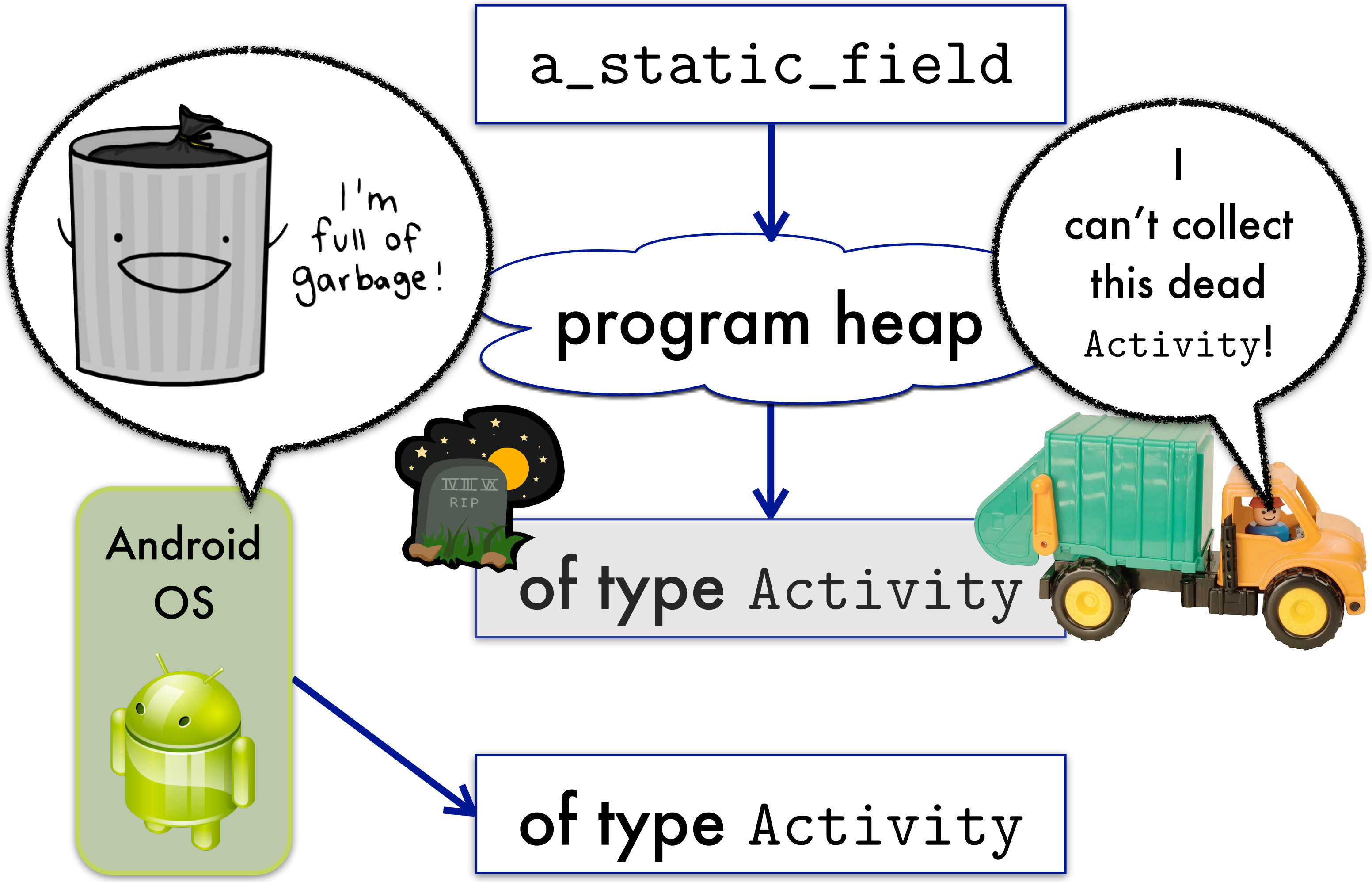

Activity objects encapsulate the UI

# Android memory leaks underly rotation-based crashes.

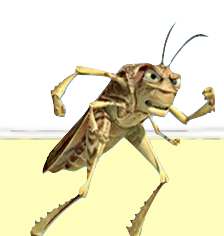



**Activity objects encapsulate the UI**

# Android memory leaks underly rotation-based crashes.

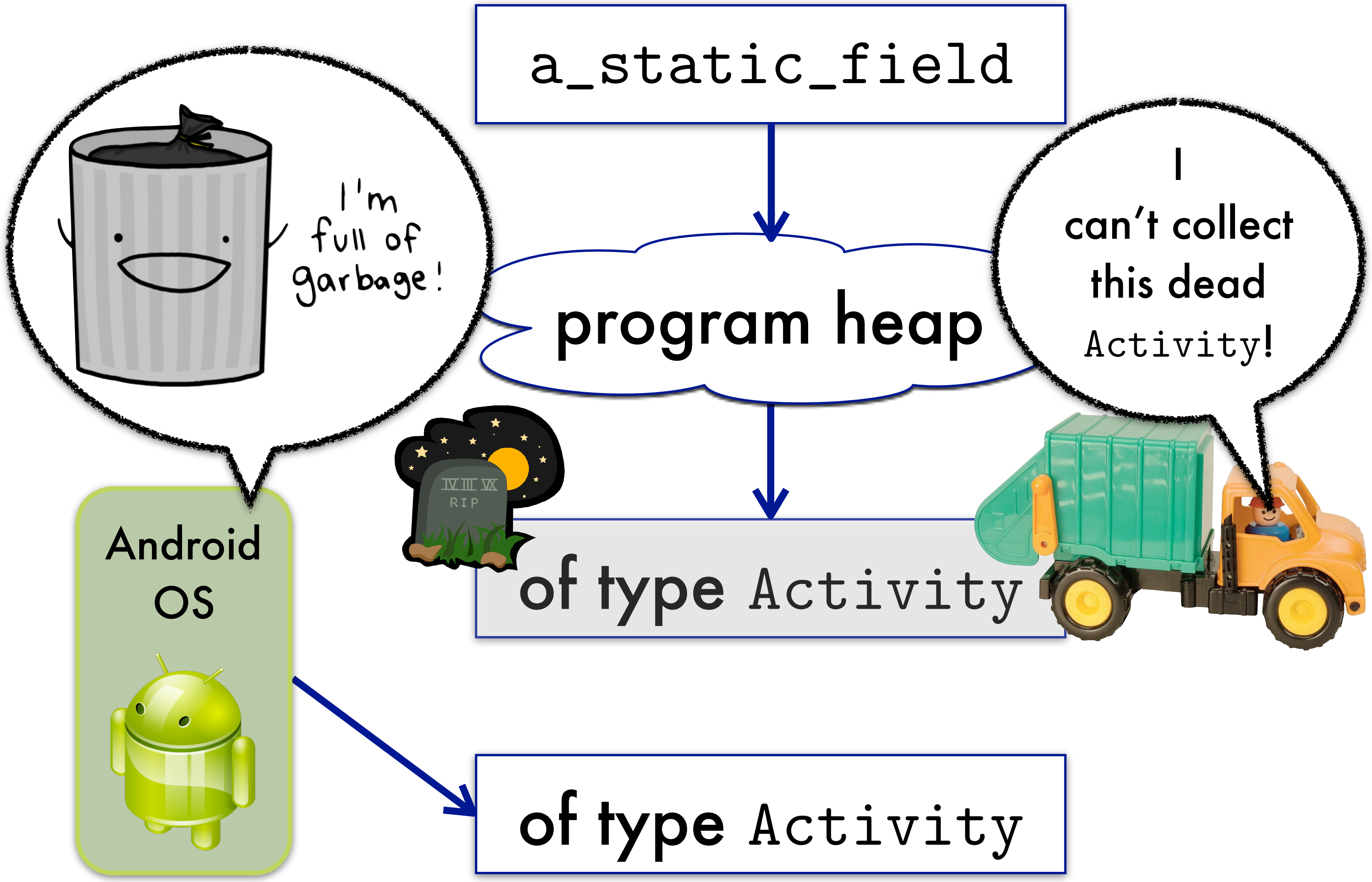



Activity objects encapsulate the UI



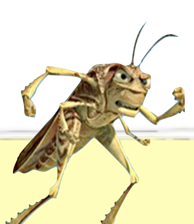
Bug: Holding reference to “old” Activity

# Android memory leaks underly rotation-based crashes.




Activity objects encapsulate the UI

"an Activity leak"



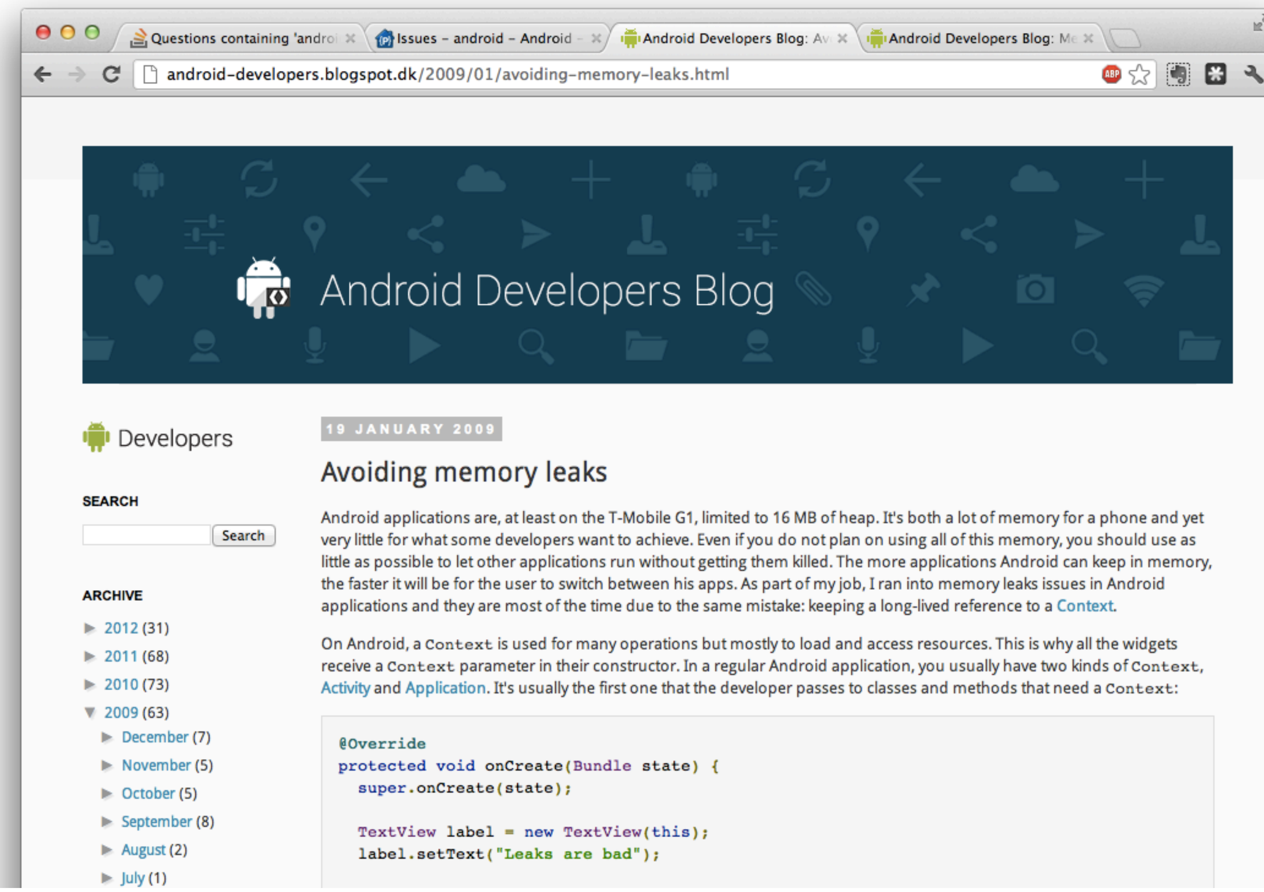
Bug: Holding reference to "old" Activity

# The expert recommendation ...

# The expert recommendation …

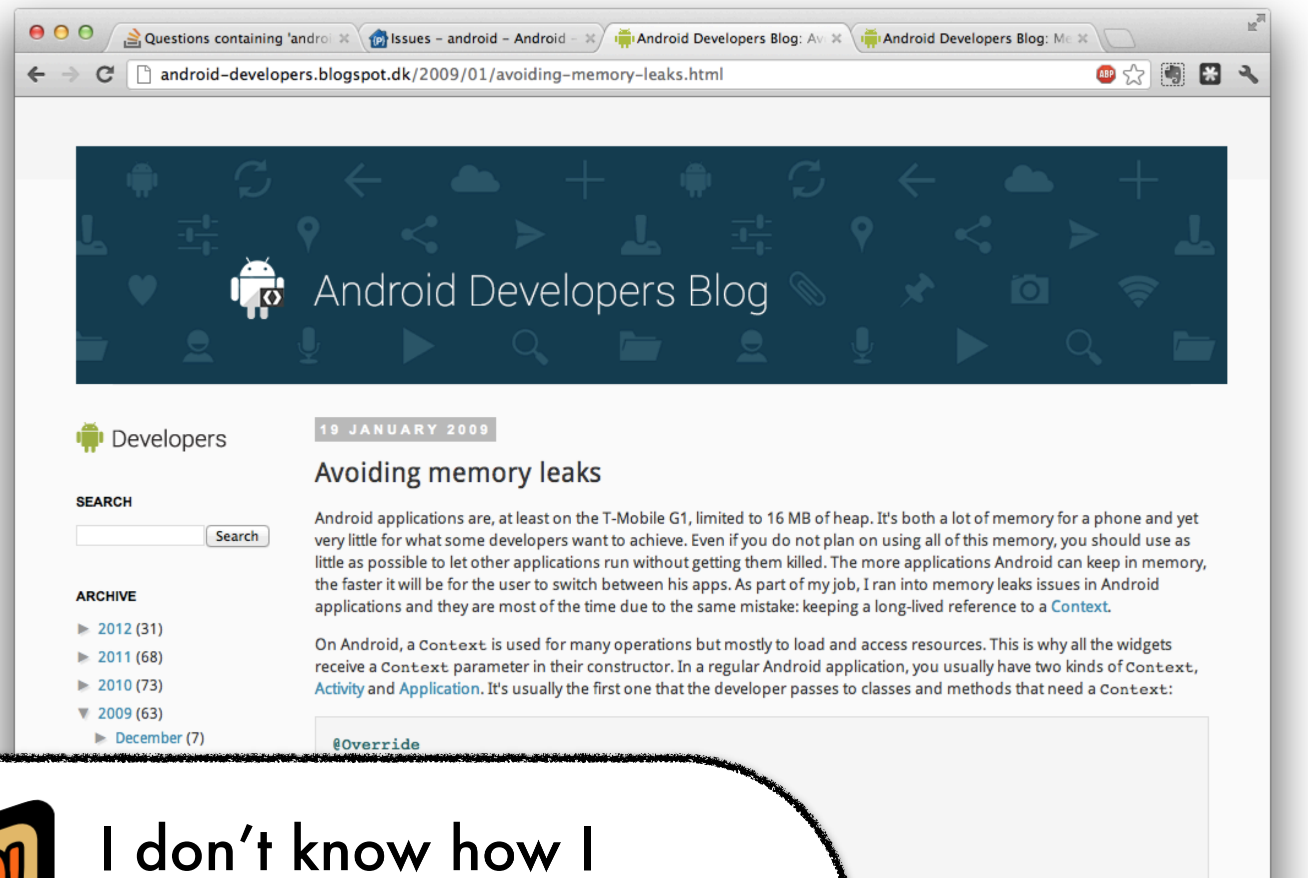
# The expert recommendation ...

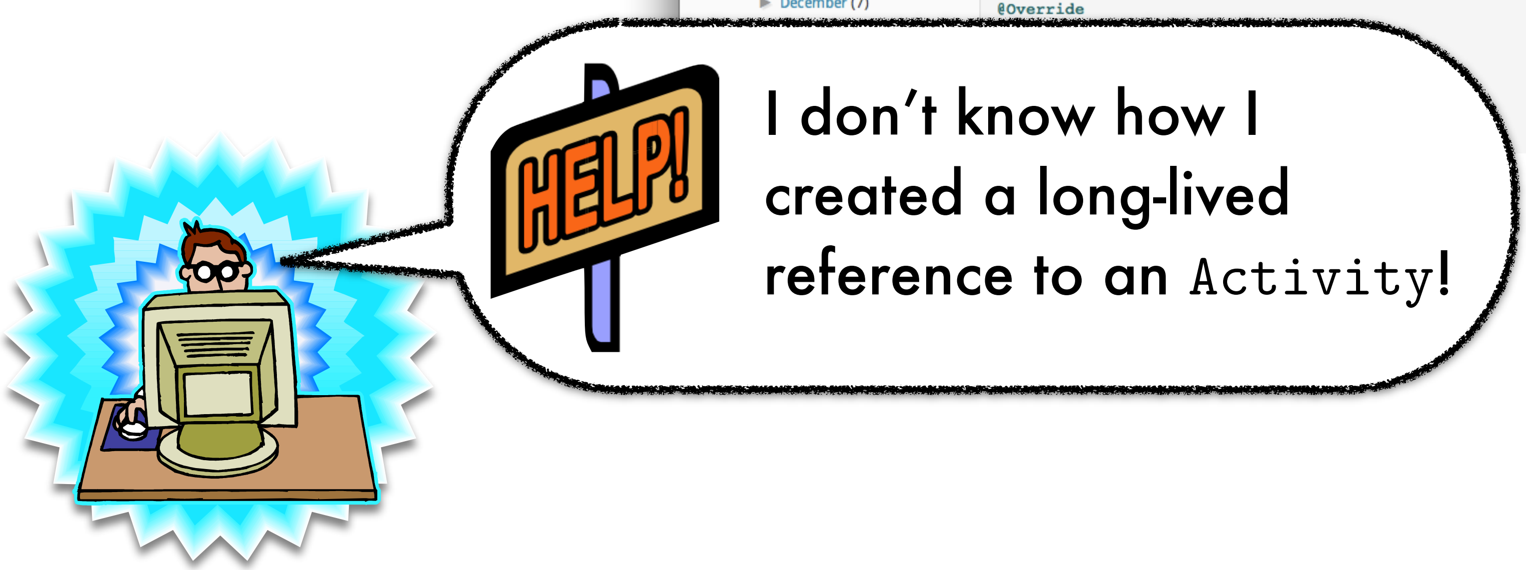
"Do not keep long-lived references to a context-activity"



# The expert recommendation …

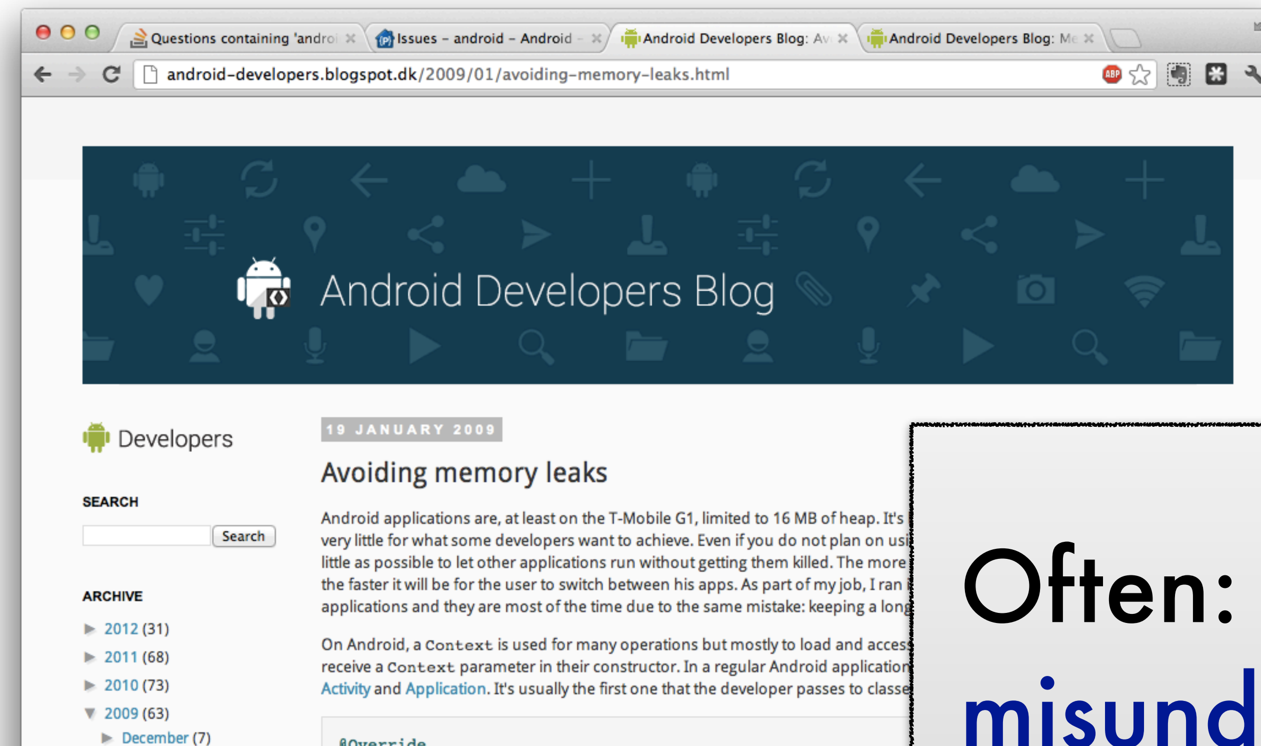
"Do not keep long-lived references to a context-activity"



# The expert recommendation ...

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

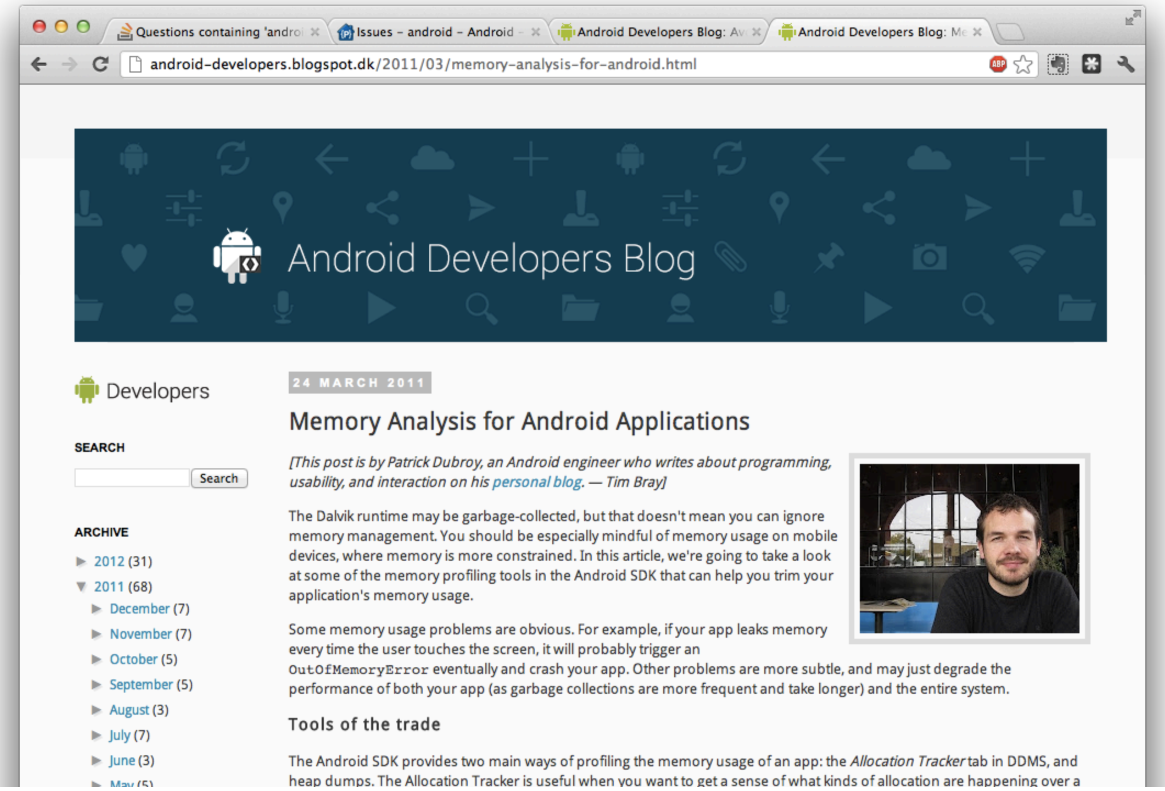




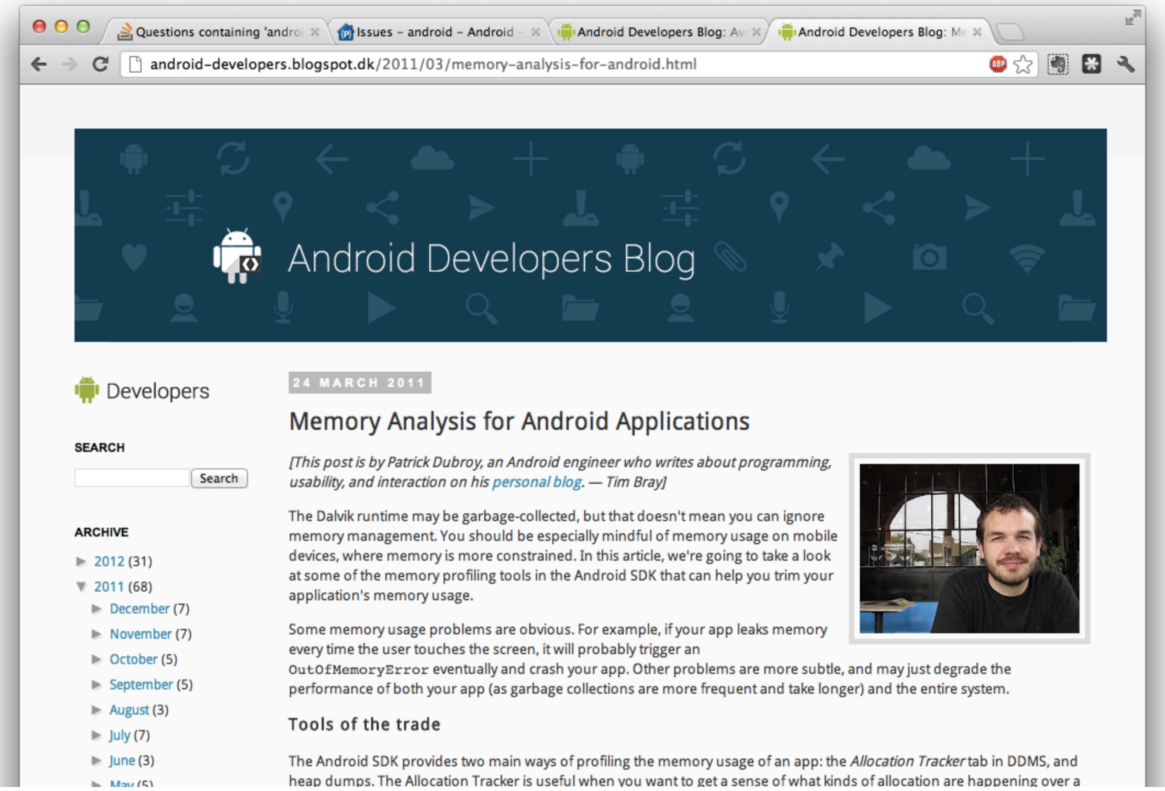
I don't know how I created a long-lived reference to an `Activity`!

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

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

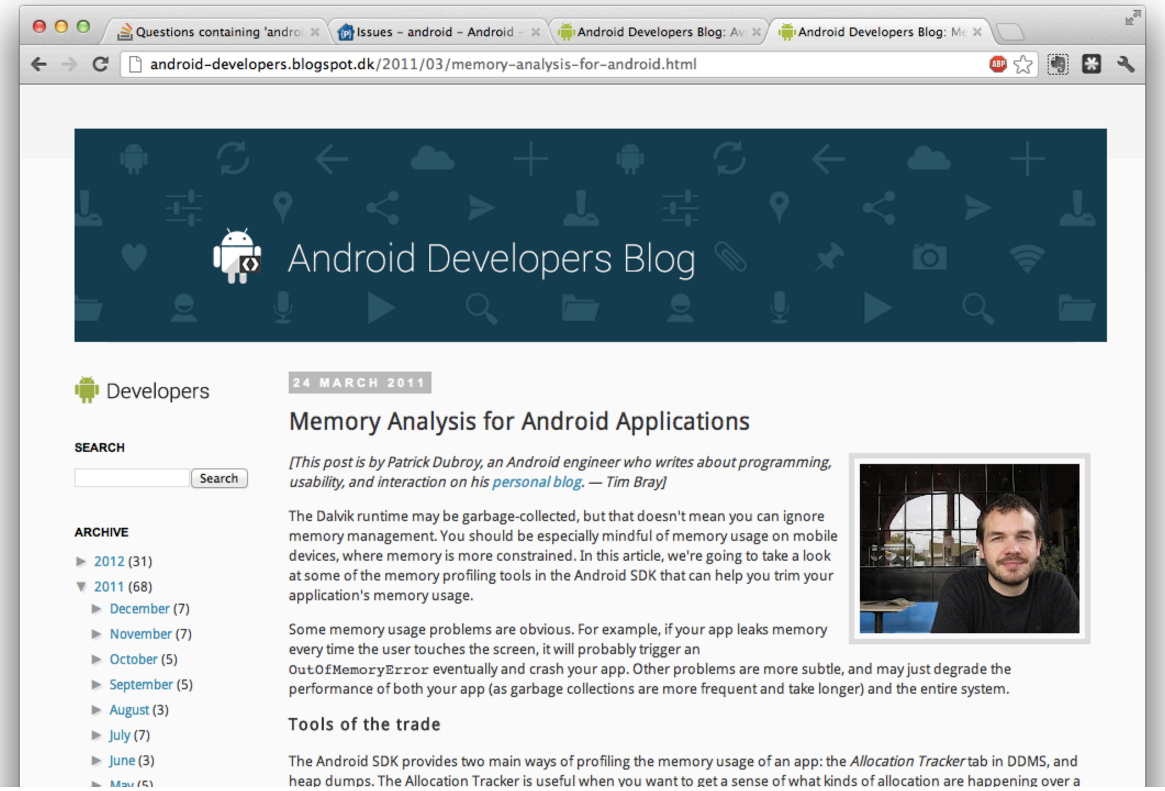


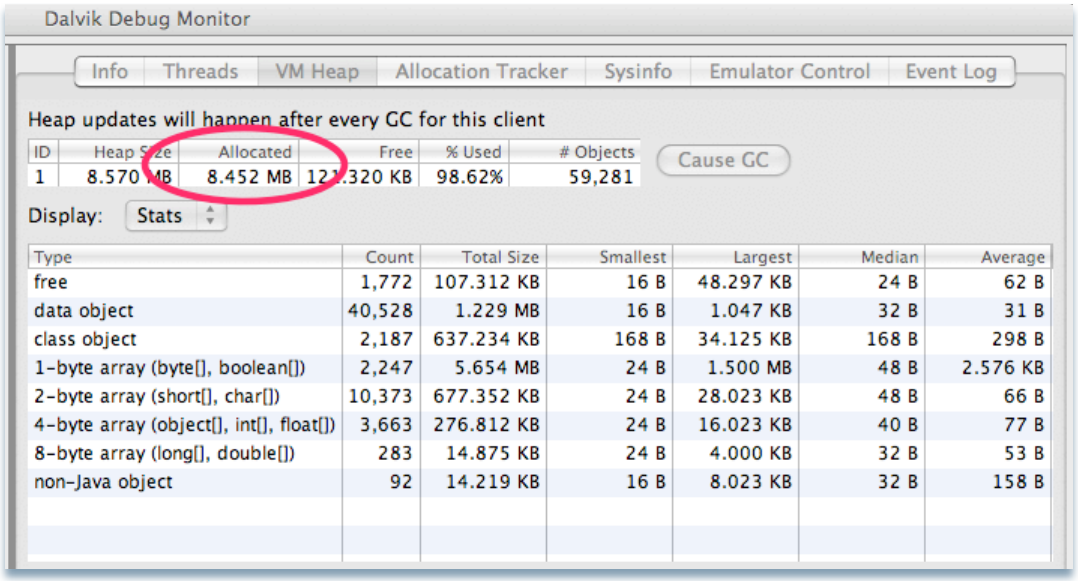
# The state of practice in debugging `Activity` leaks …




1. Run the app

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



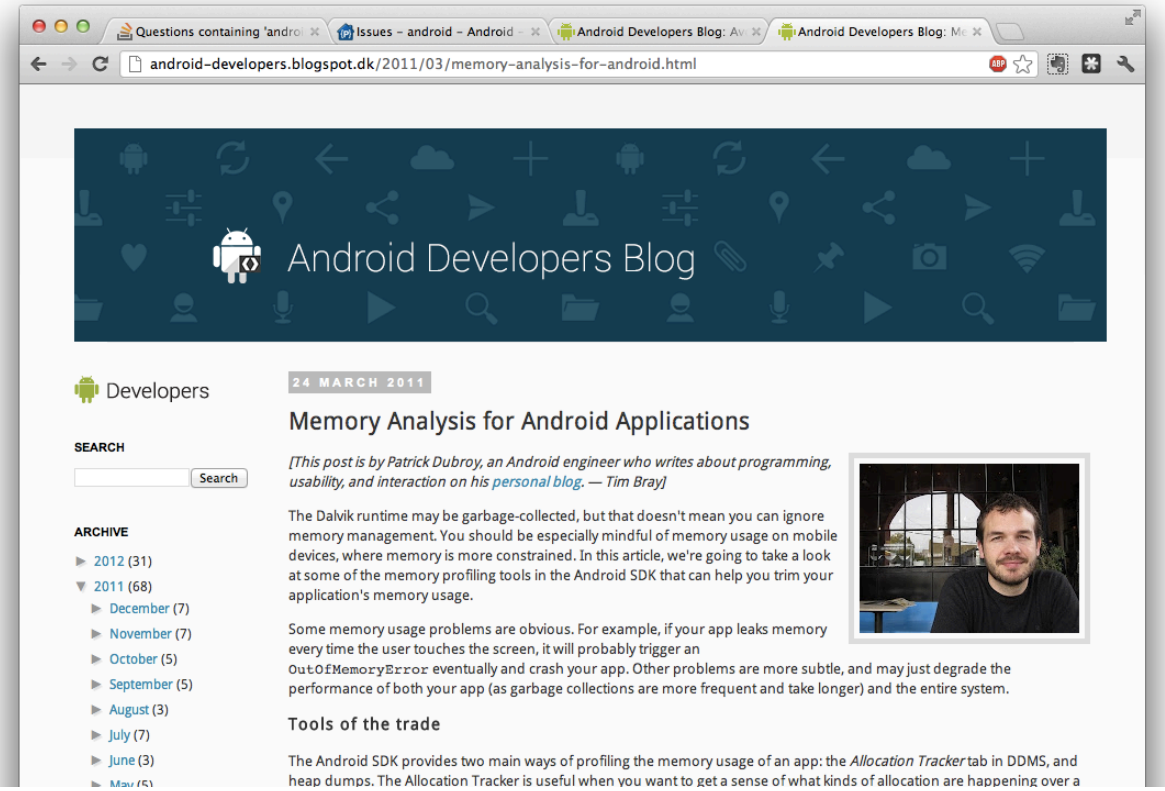



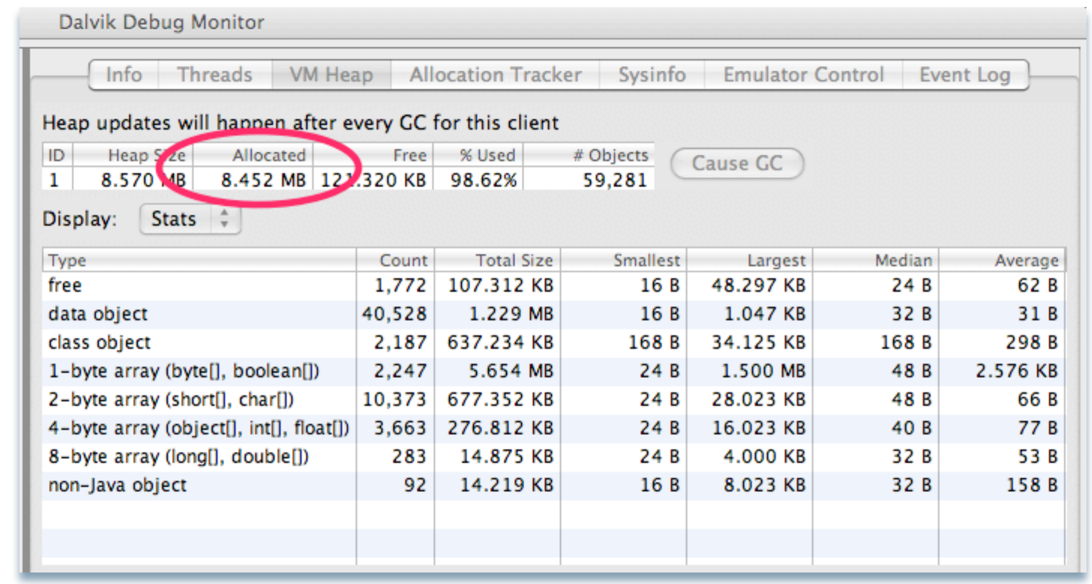

| ID | Heap Size | Allocated | Free | % Used | # Objects |
|---|---|---|---|---|---|
| 1 | 8.570 MB | 8.452 MB | 121.320 KB | 98.62% | 59,281 |

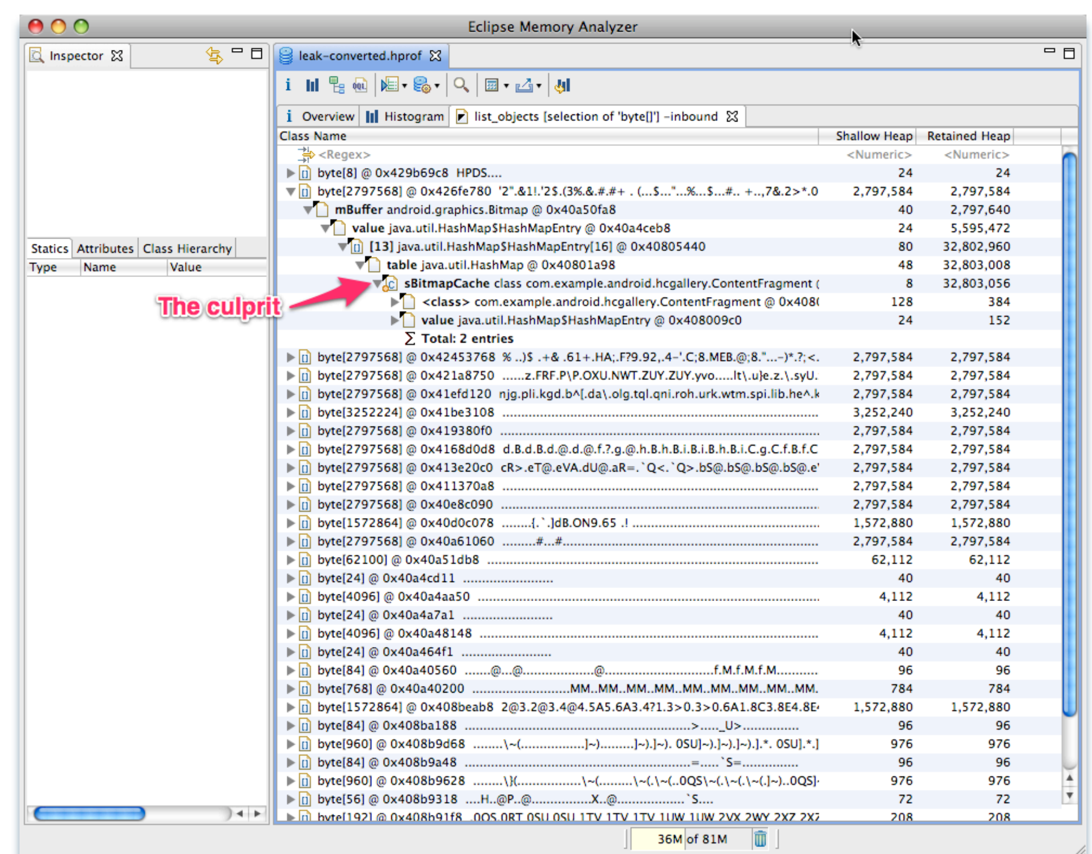
| Type | Count | Total Size | Smallest | Largest | Median | Average |
|---|---|---|---|---|---|---|
| free | 1,772 | 107.312 KB | 16 B | 48.297 KB | 24 B | 62 B |
| data object | 40,528 | 1.229 MB | 16 B | 1.047 KB | 32 B | 31 B |
| class object | 2,187 | 637.234 KB | 168 B | 34.125 KB | 168 B | 298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

1. Run the app
2. Watch the heap usage

# The state of practice in debugging `Activity` leaks …



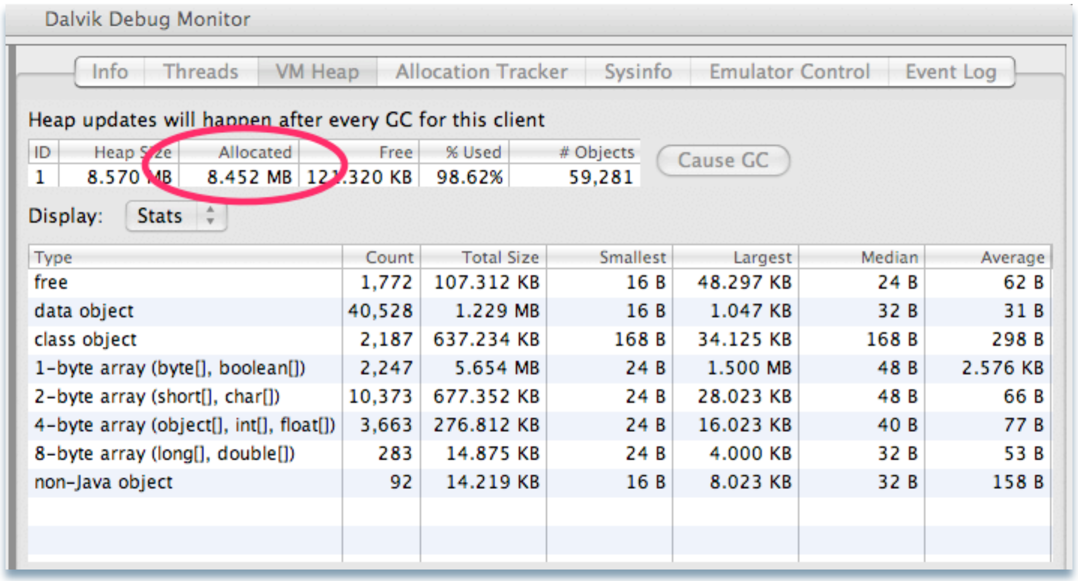

1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and **hope** to find the culprit

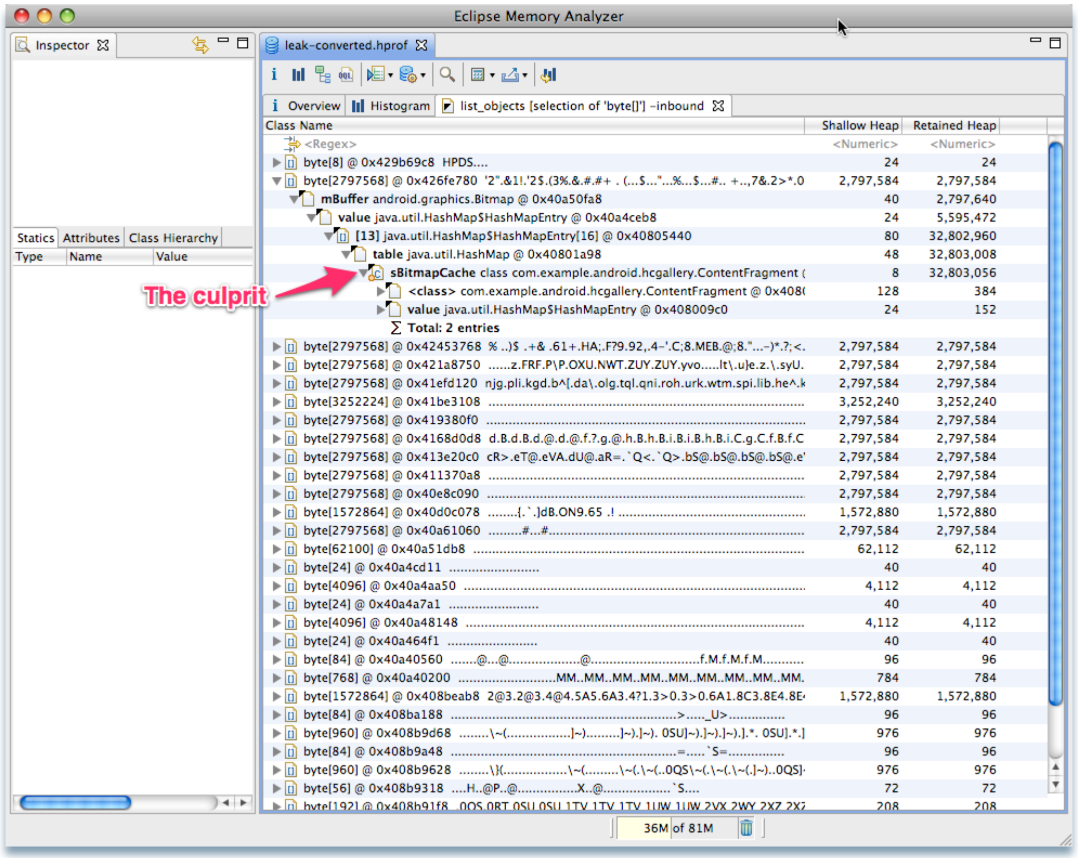
# The state of practice in debugging `Activity` leaks ...

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

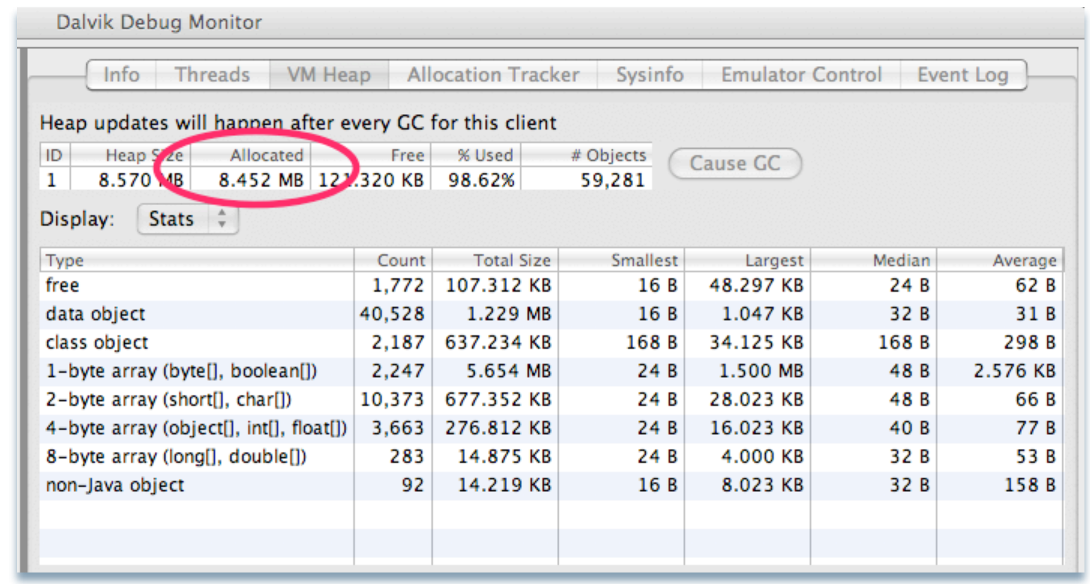


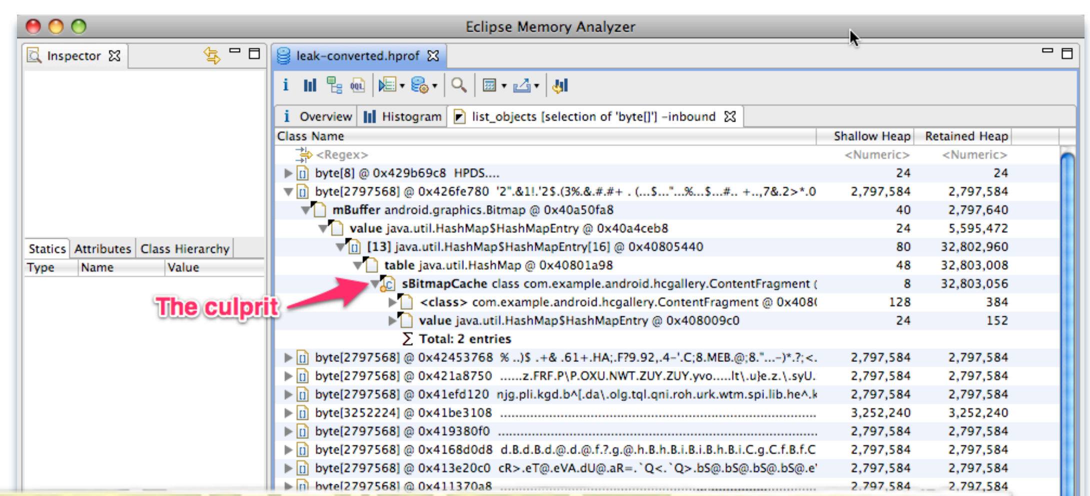



# The state of practice in debugging `Activity` leaks ...

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

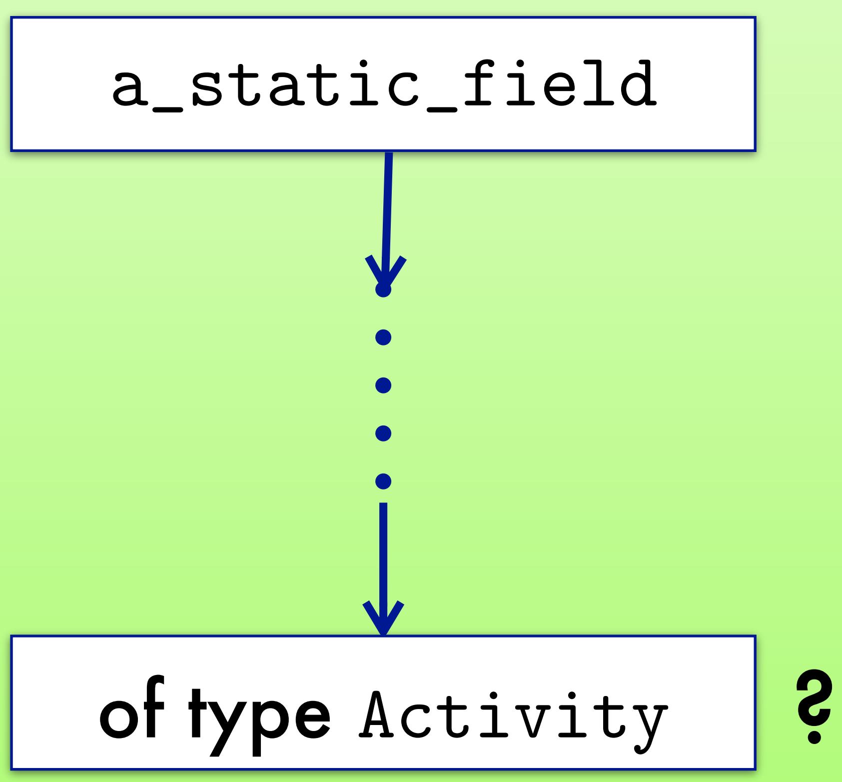
"One of the most dreaded bugs in Android is a memory leak. They are nasty because one piece of code causes an issue and in some other piece of code, your application crashes." – http://therockncoder.blogspot.com/2012/09/fixing-android-memory-leak.html


Program
Analysis

# Answering “Is there an `Activity` leak?” with program analysis …

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

Is there a program execution where at some time

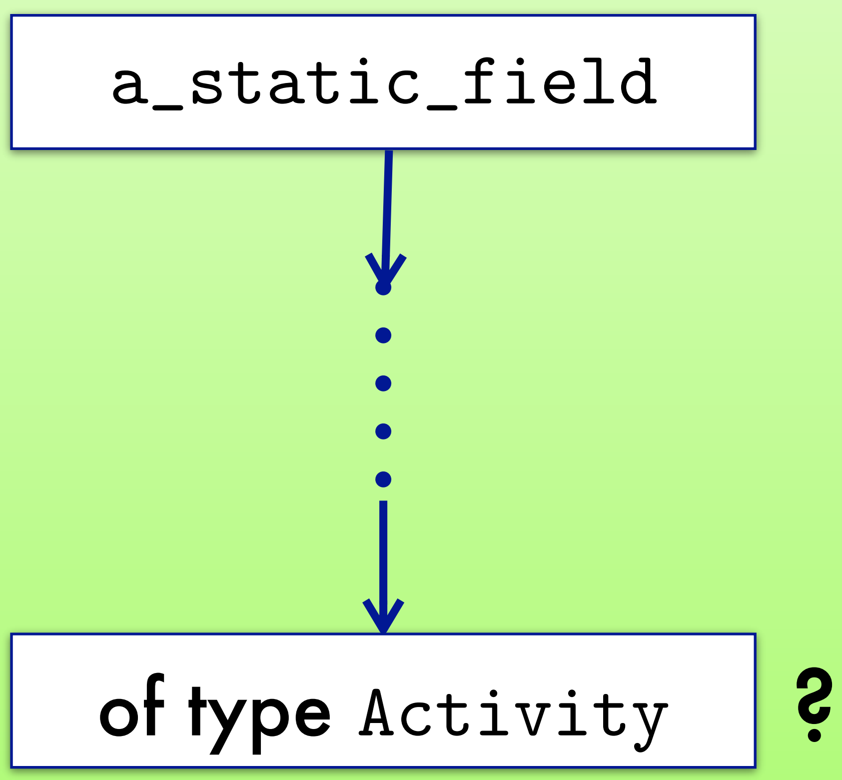



Example

# 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



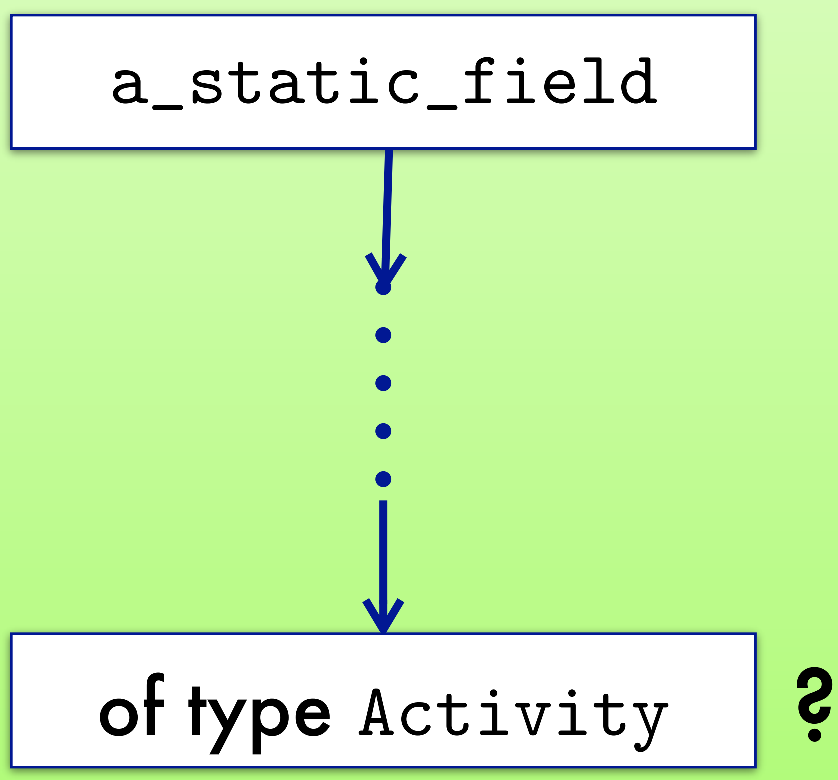

Example

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

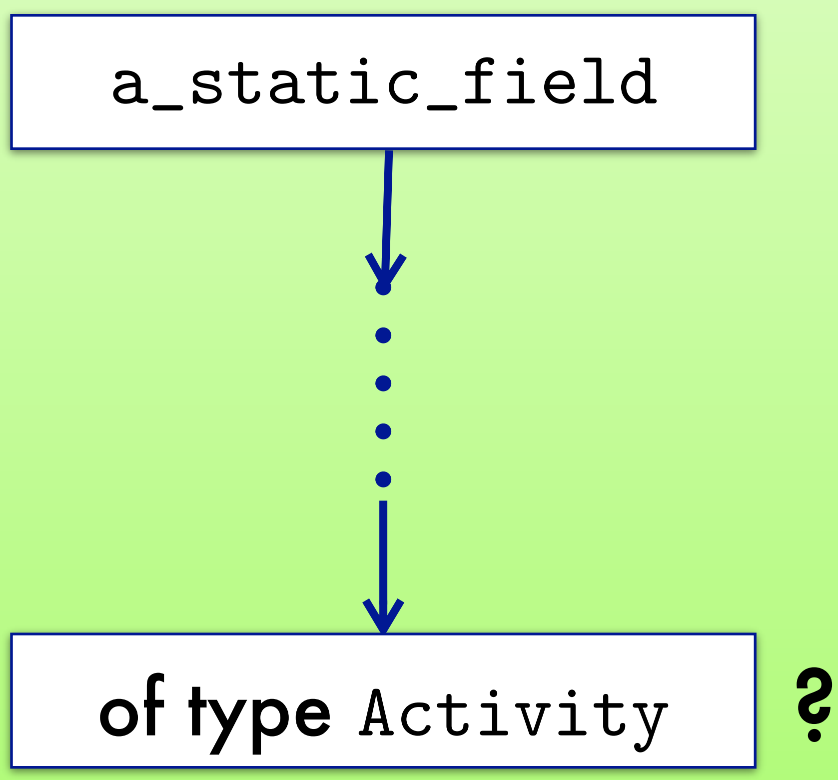
Hidden Truth

with **approximation**

# 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

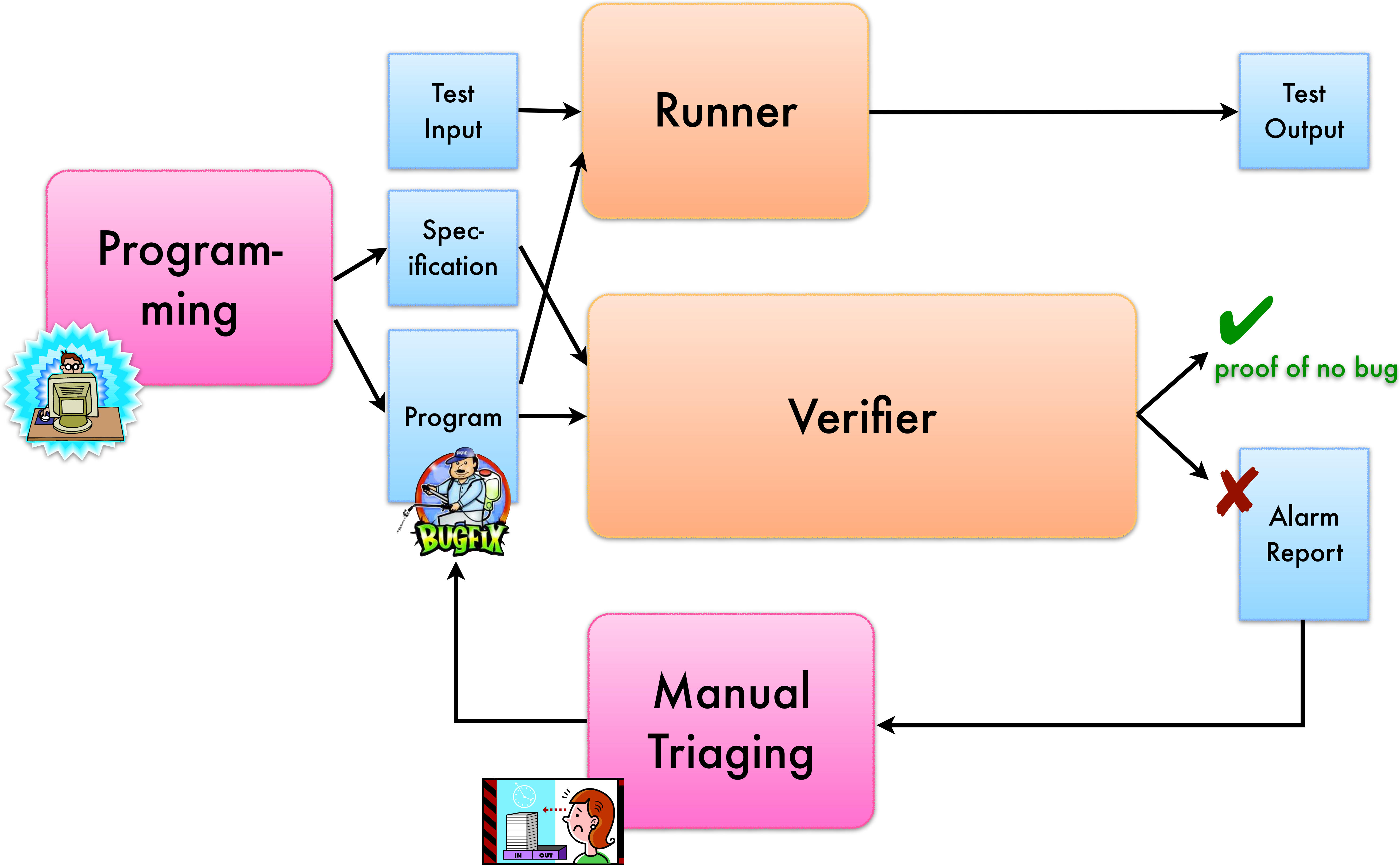
Can be answered with a points-to analysis

Hidden Truth

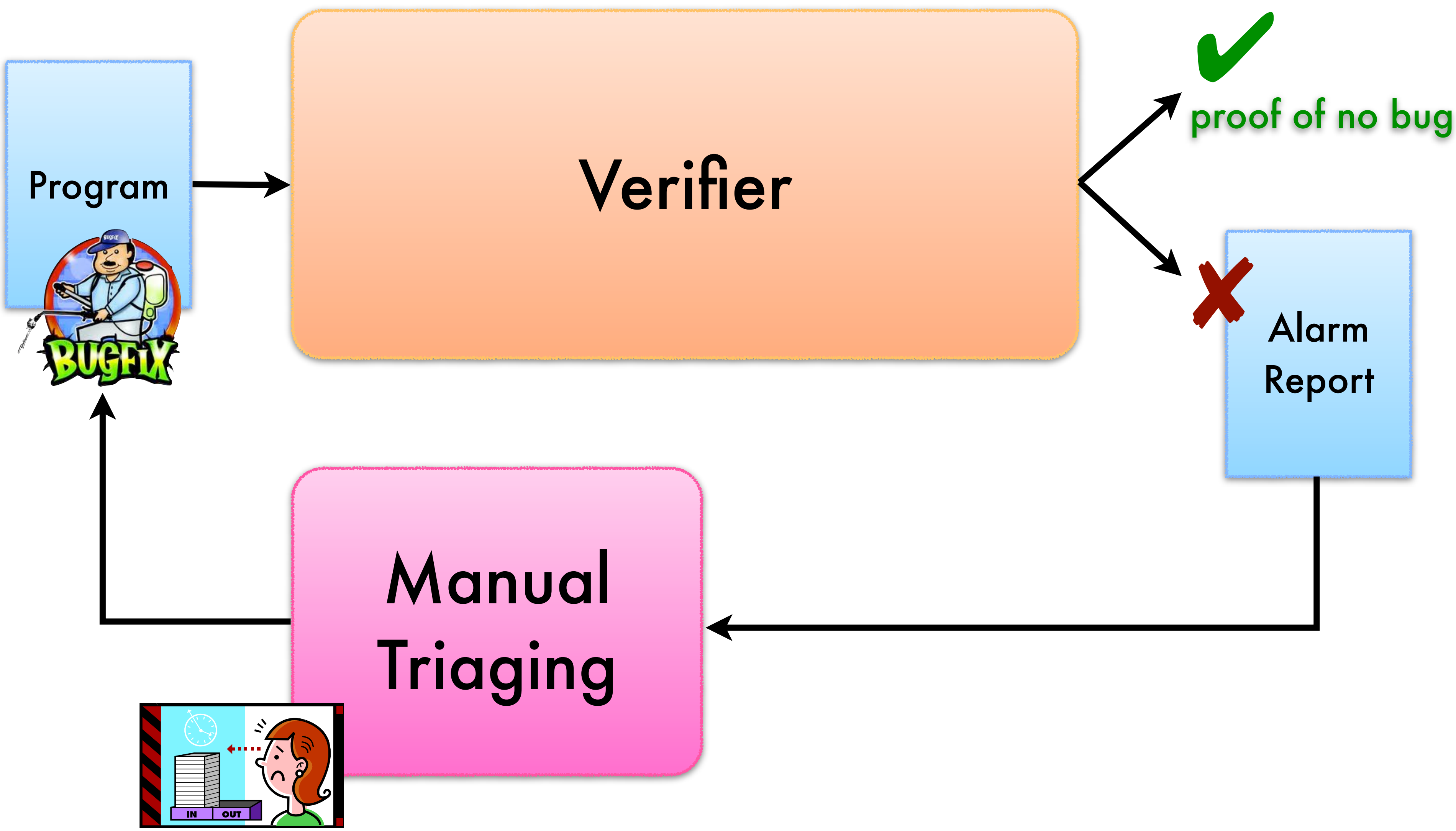
with **approximation**

Some pointer relations **may** be false

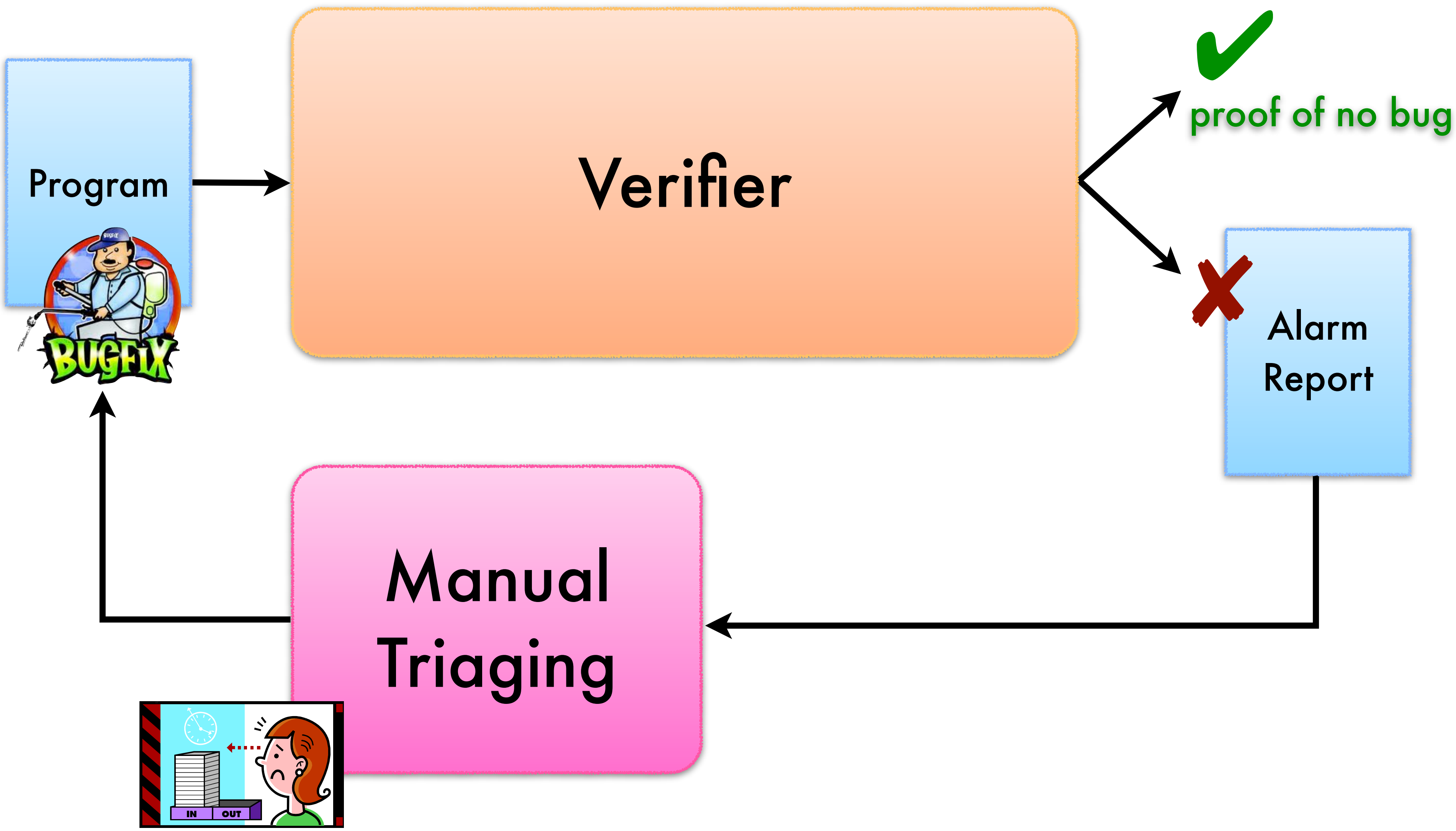
# But with the cooperative approach ...



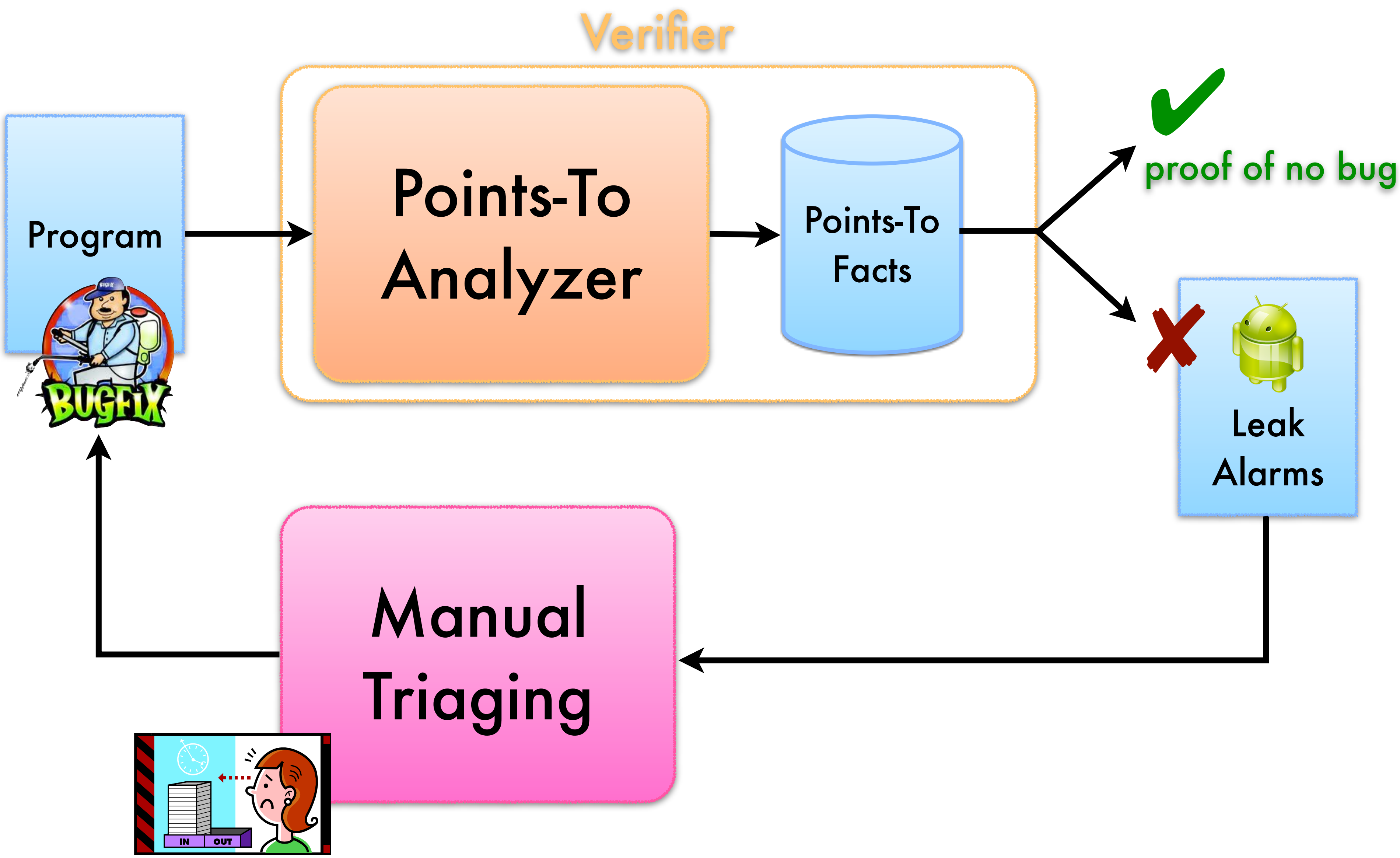
# But with the cooperative approach ...



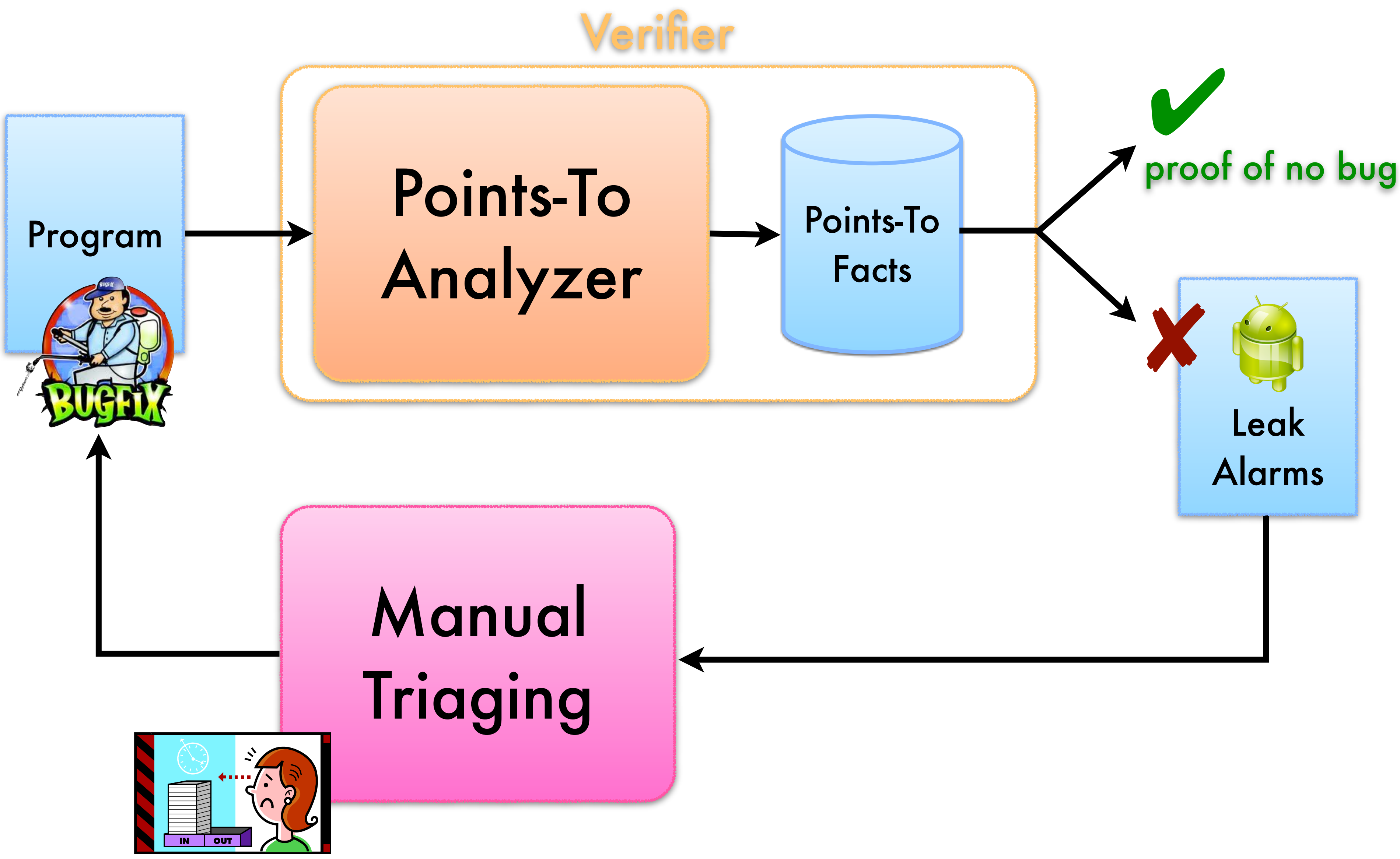
# Thresher addresses alarm **triage** in a particularly challenging domain.



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

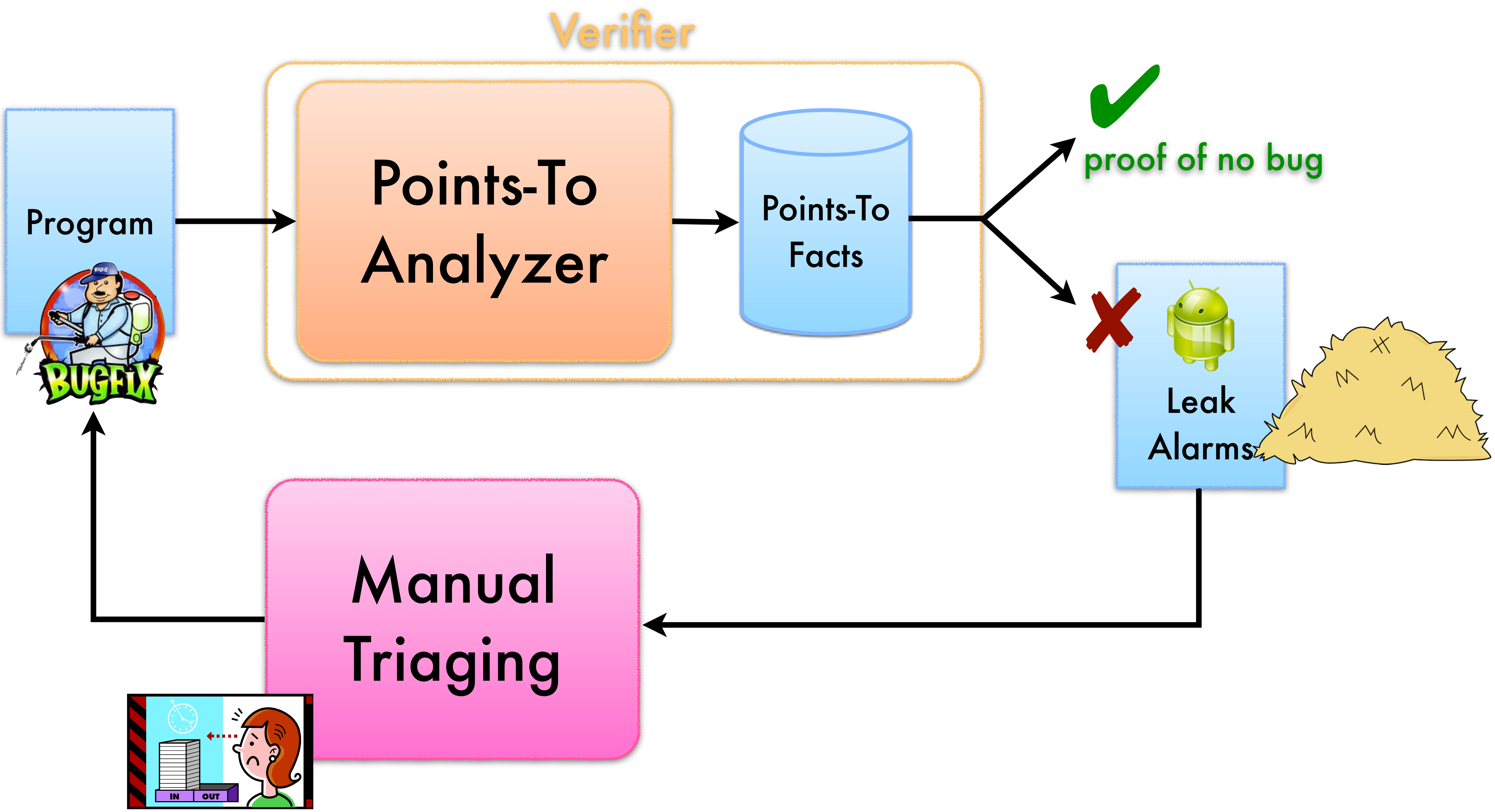


# Thresher addresses alarm **triage** in a particularly challenging domain.



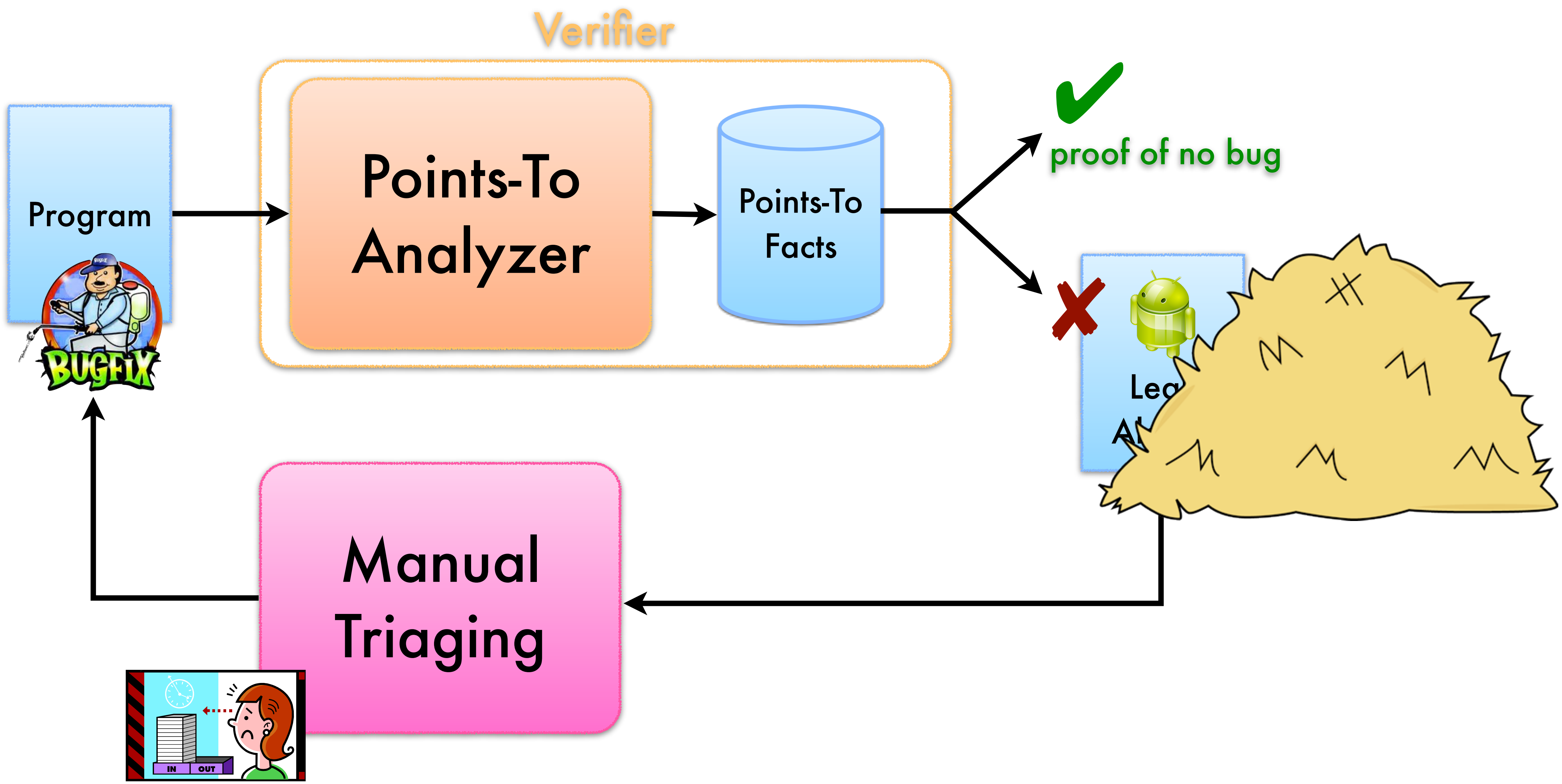

Known: Precise points-to analysis challenging

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



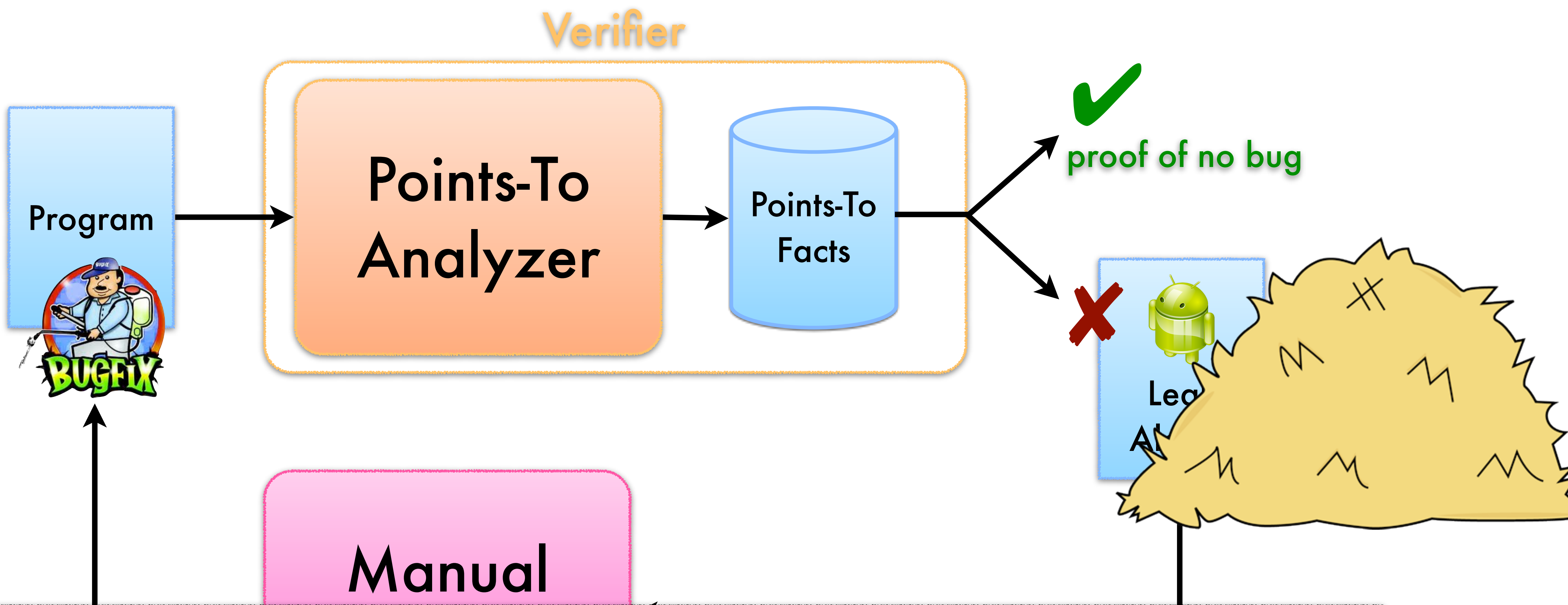

Known: Precise points-to analysis challenging

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




Known: Precise points-to analysis challenging

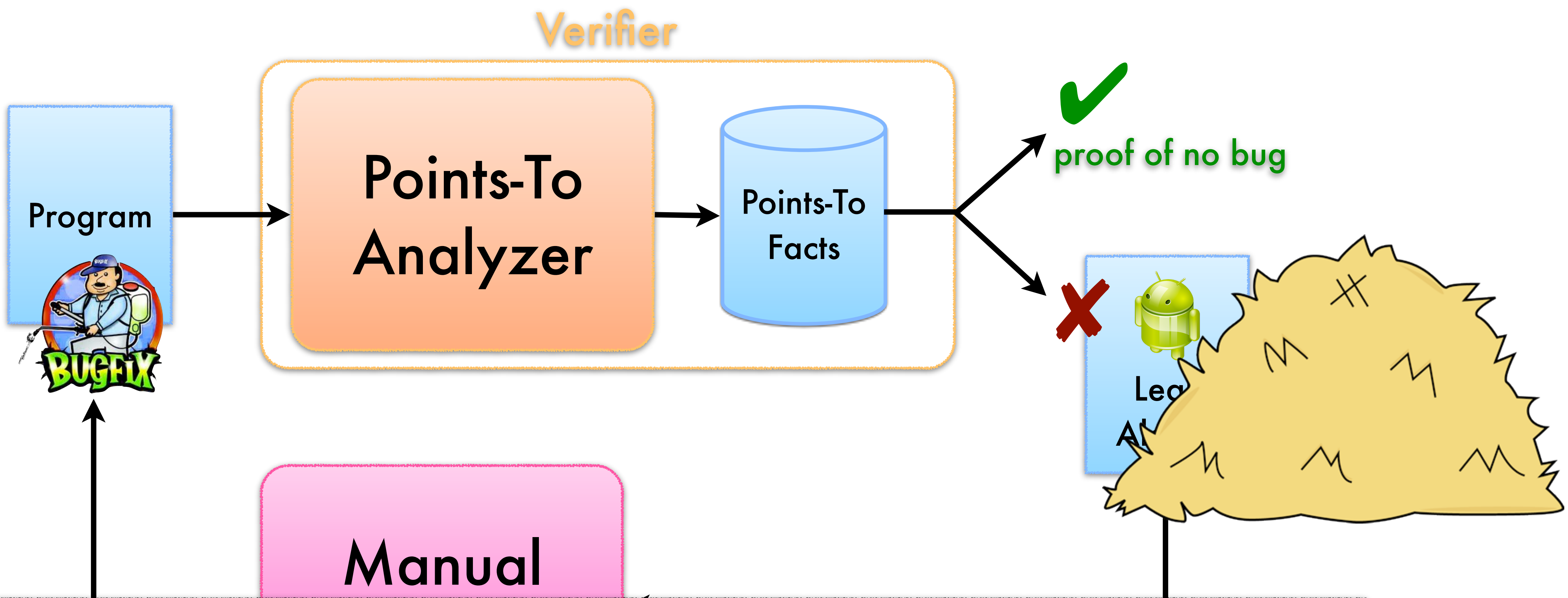
# Thresher addresses alarm **triage** in a particularly challenging domain.




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.

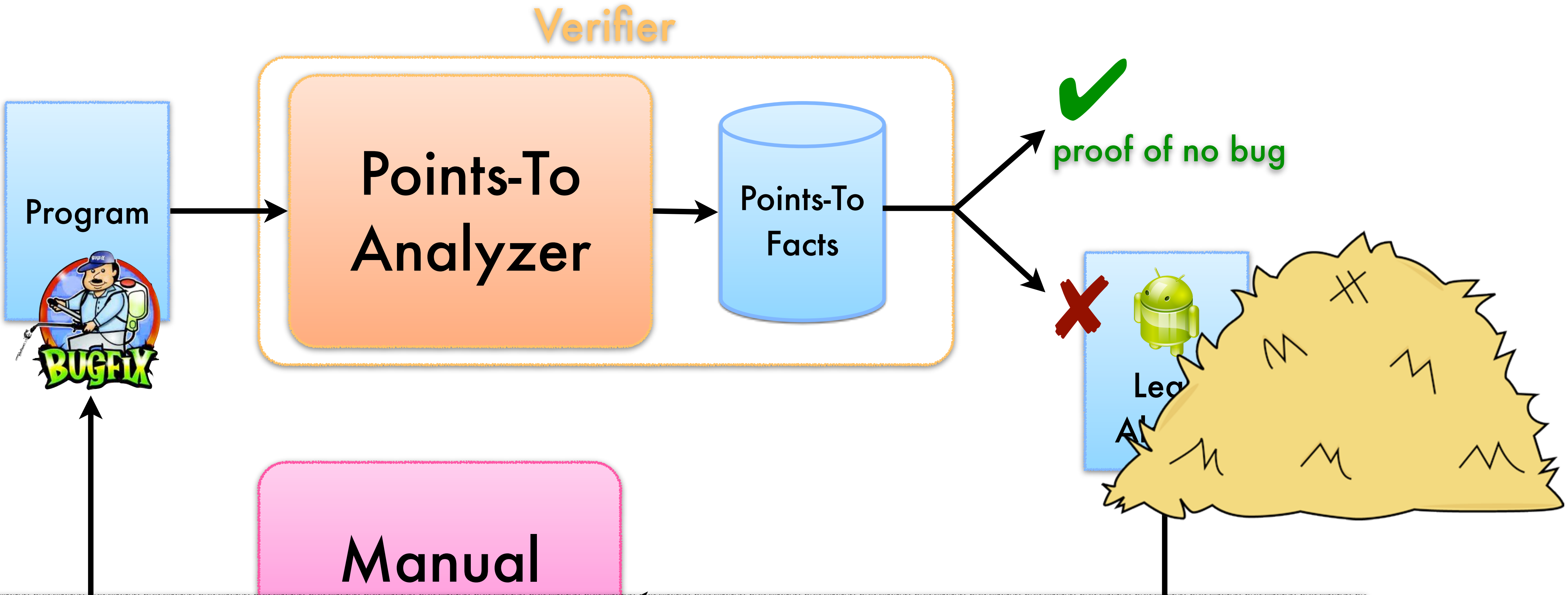



Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?"

- 75 papers, 9 PhD theses

Known: Precise points-to analysis challenging

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

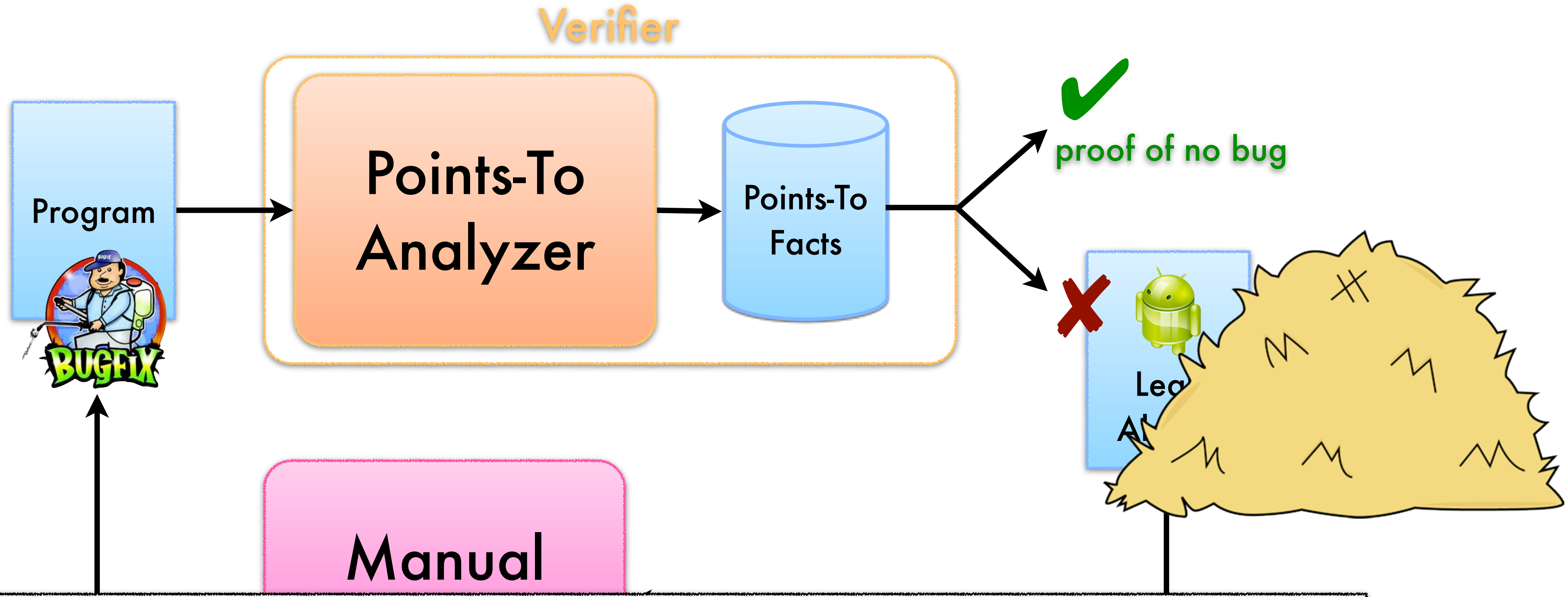



Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?" (2001)

- 75 papers, 9 PhD theses

Known: Precise points-to analysis challenging

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



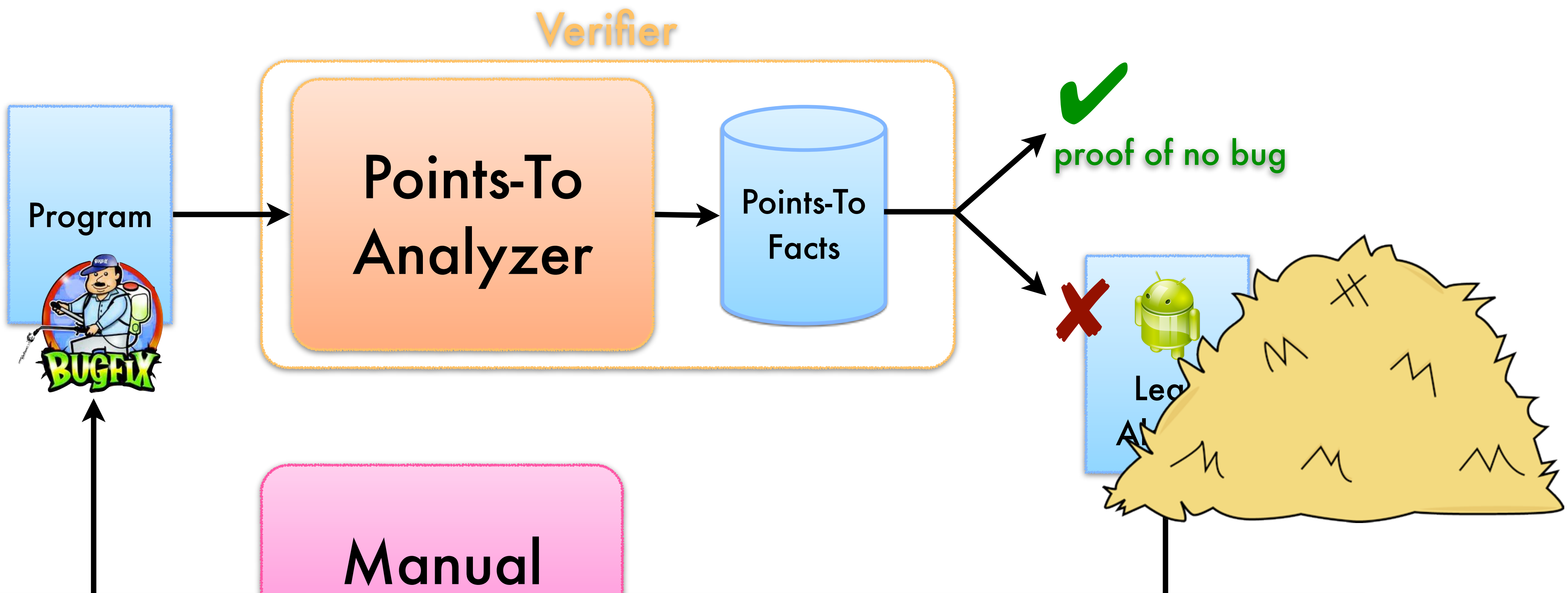

Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?" (2001)

- 75 papers, 9 PhD theses

Dagstuhl 13162: Pointer Analysis (2013)

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

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



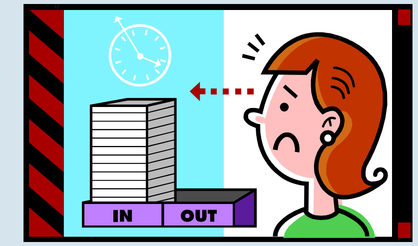

Hind. "Pointer Analysis: Haven't We Solved This Problem Yet?" (2001)

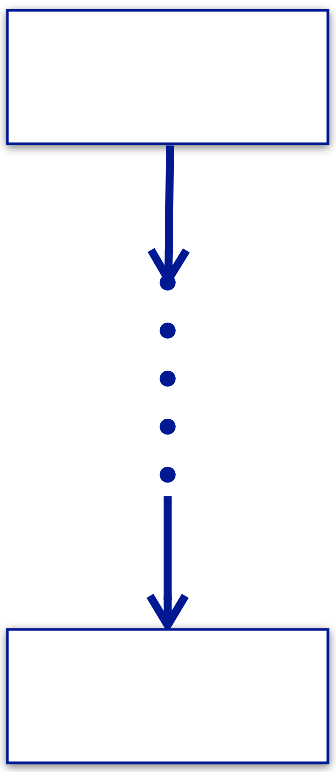
- 75 papers, 9 PhD theses

Dagstuhl 13162: Pointer Analysis (2013)

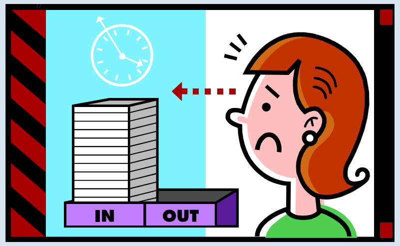
Known: Precise ^enough^ points-to analysis ~~challenging~~ impossible?

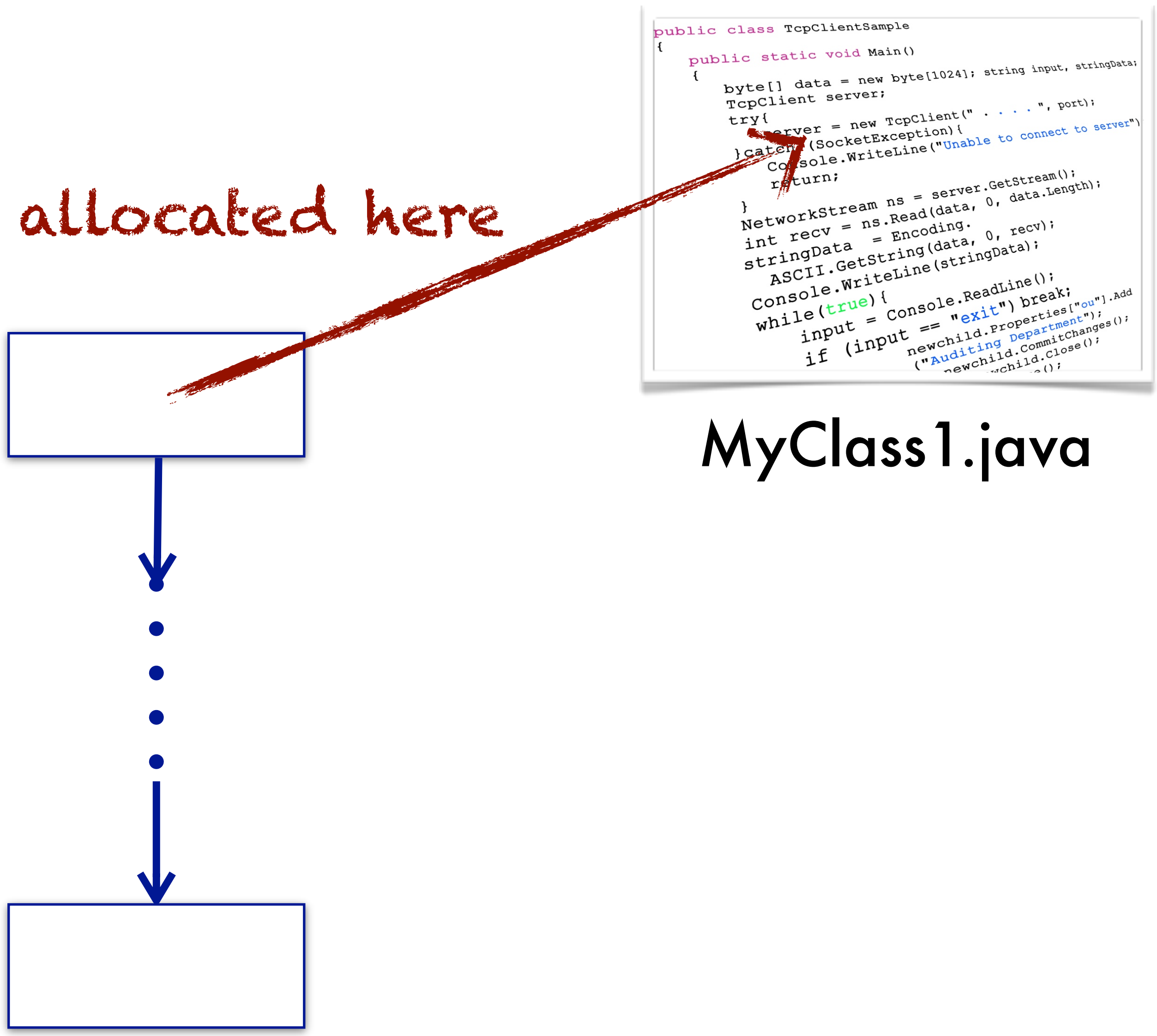
# Manual triage is particularly hard for heap reachability reports.



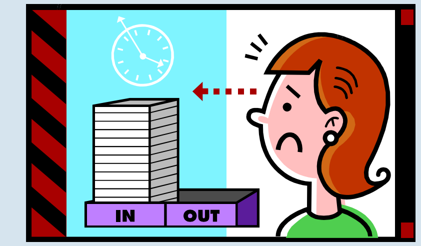



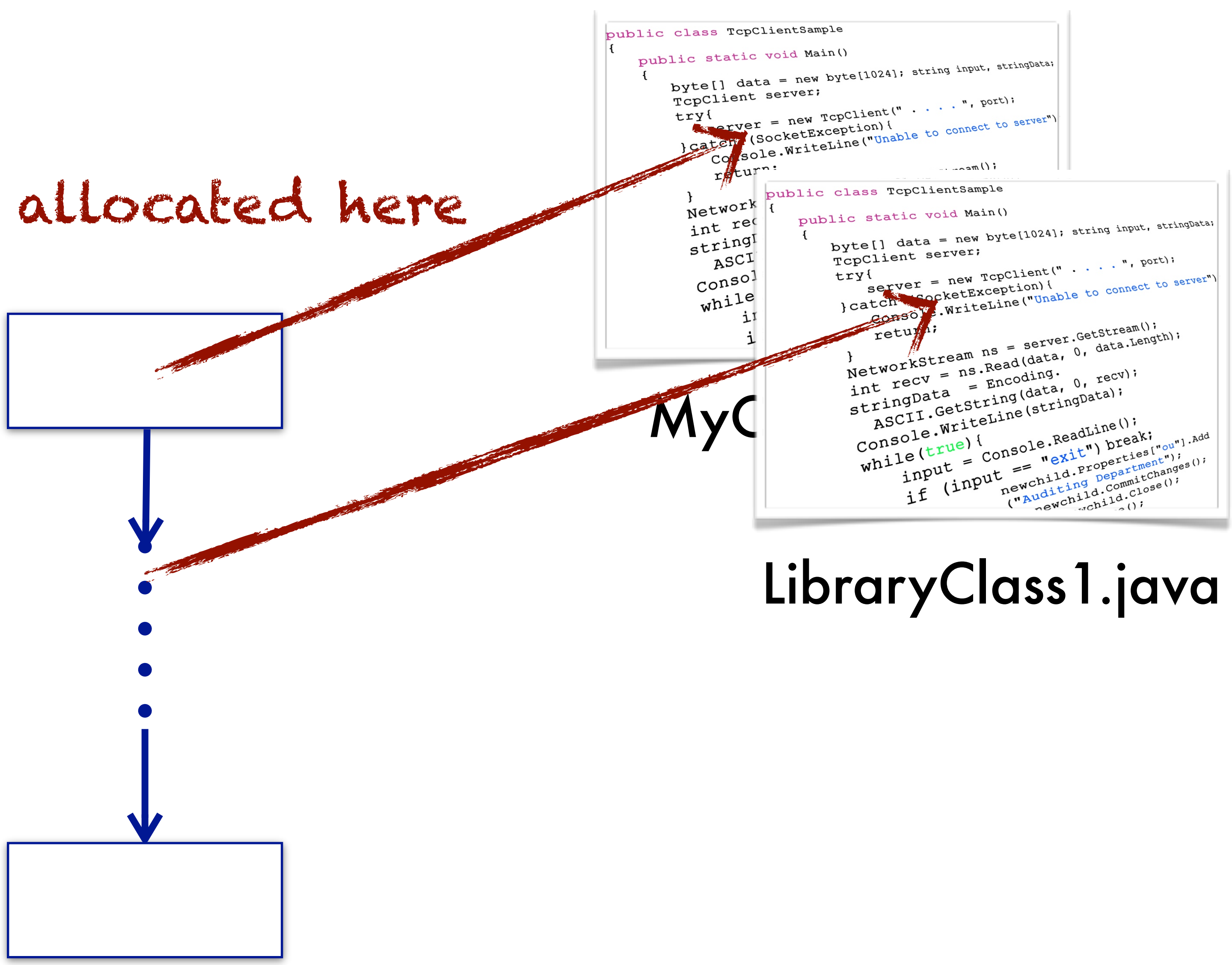
# Manual triage is particularly hard for heap reachability reports.



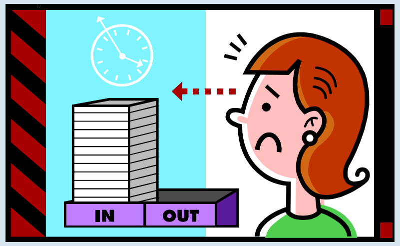



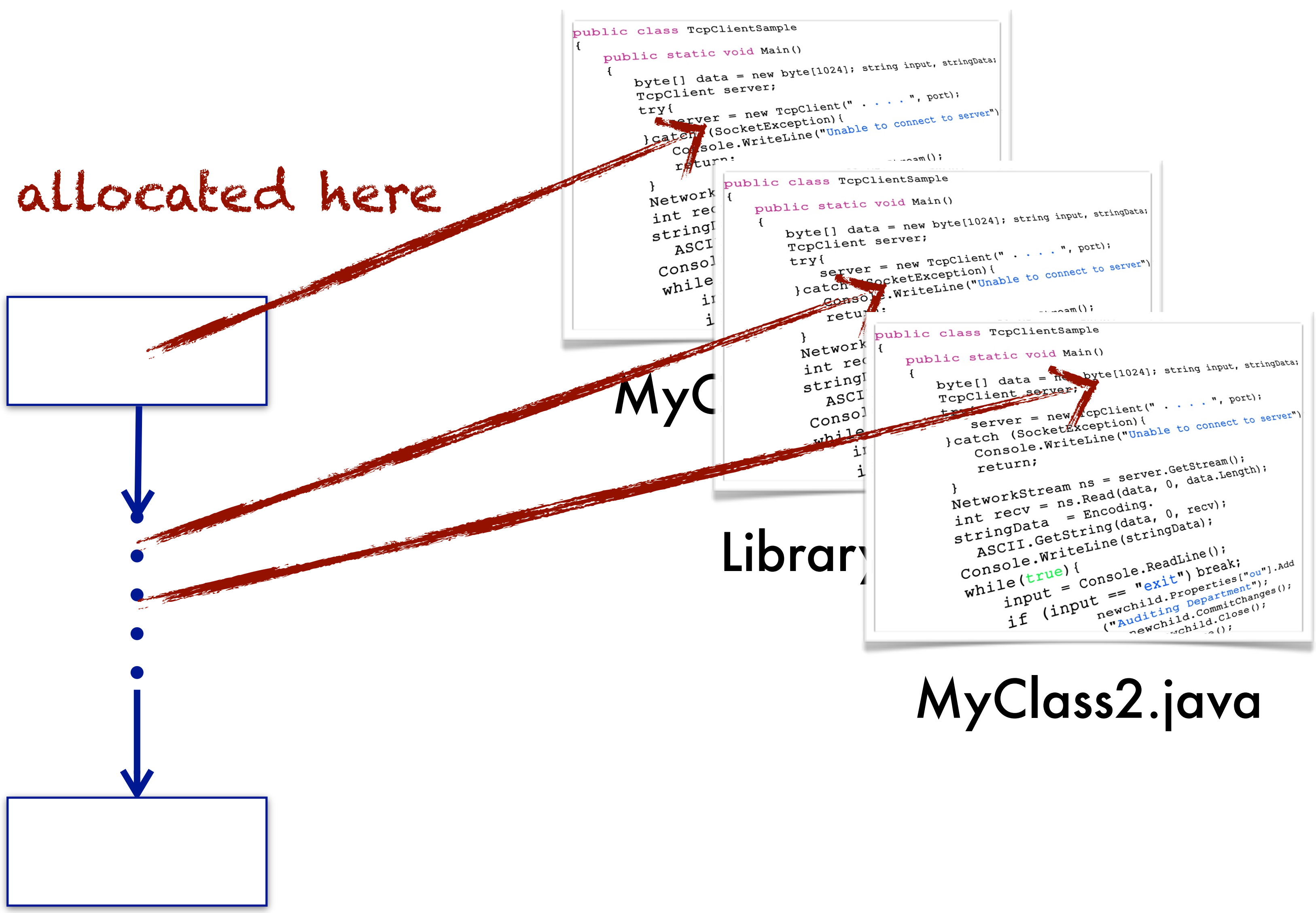
# Manual triage is particularly hard for heap reachability reports.



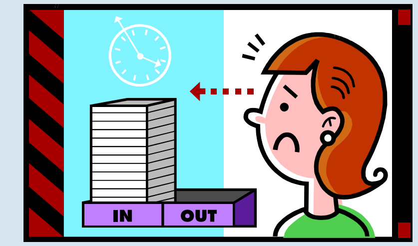


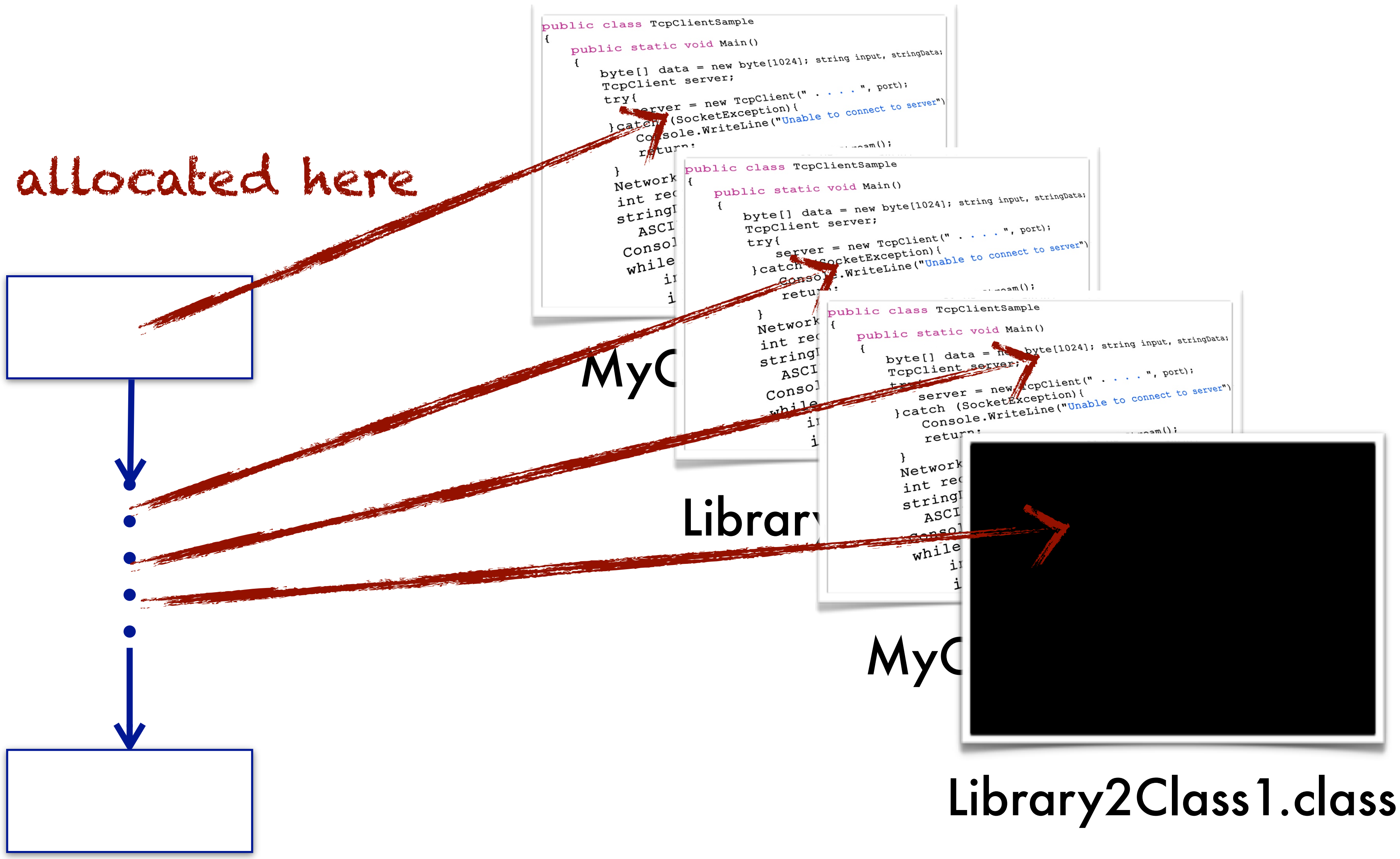
# Manual triage is particularly hard for heap reachability reports.



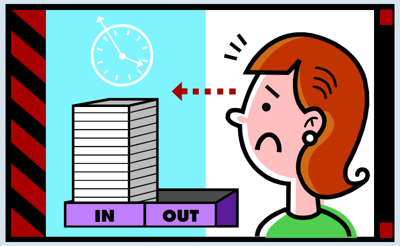


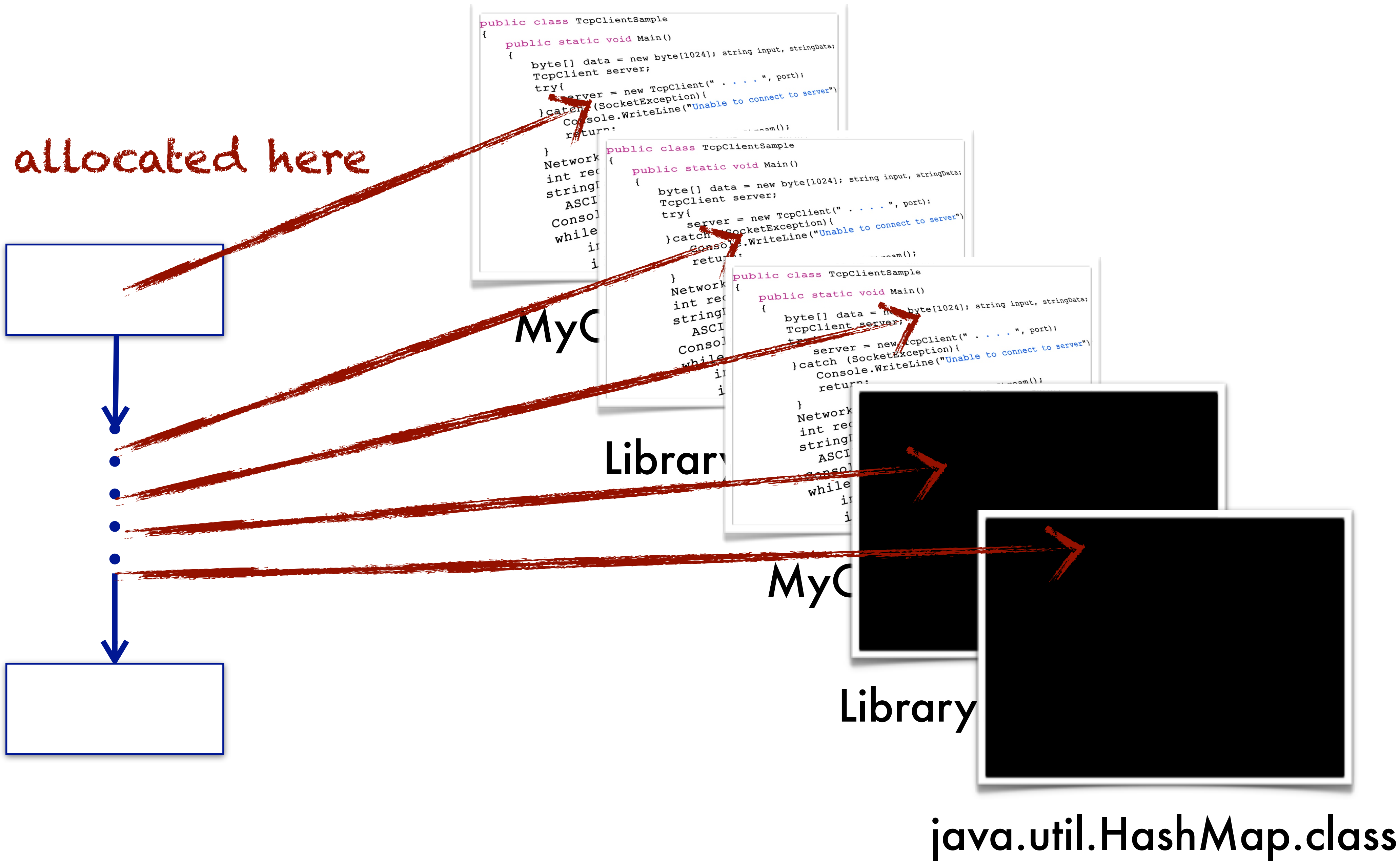
# Manual triage is particularly hard for heap reachability reports.



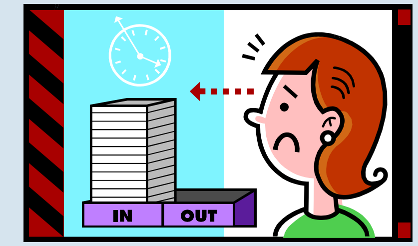


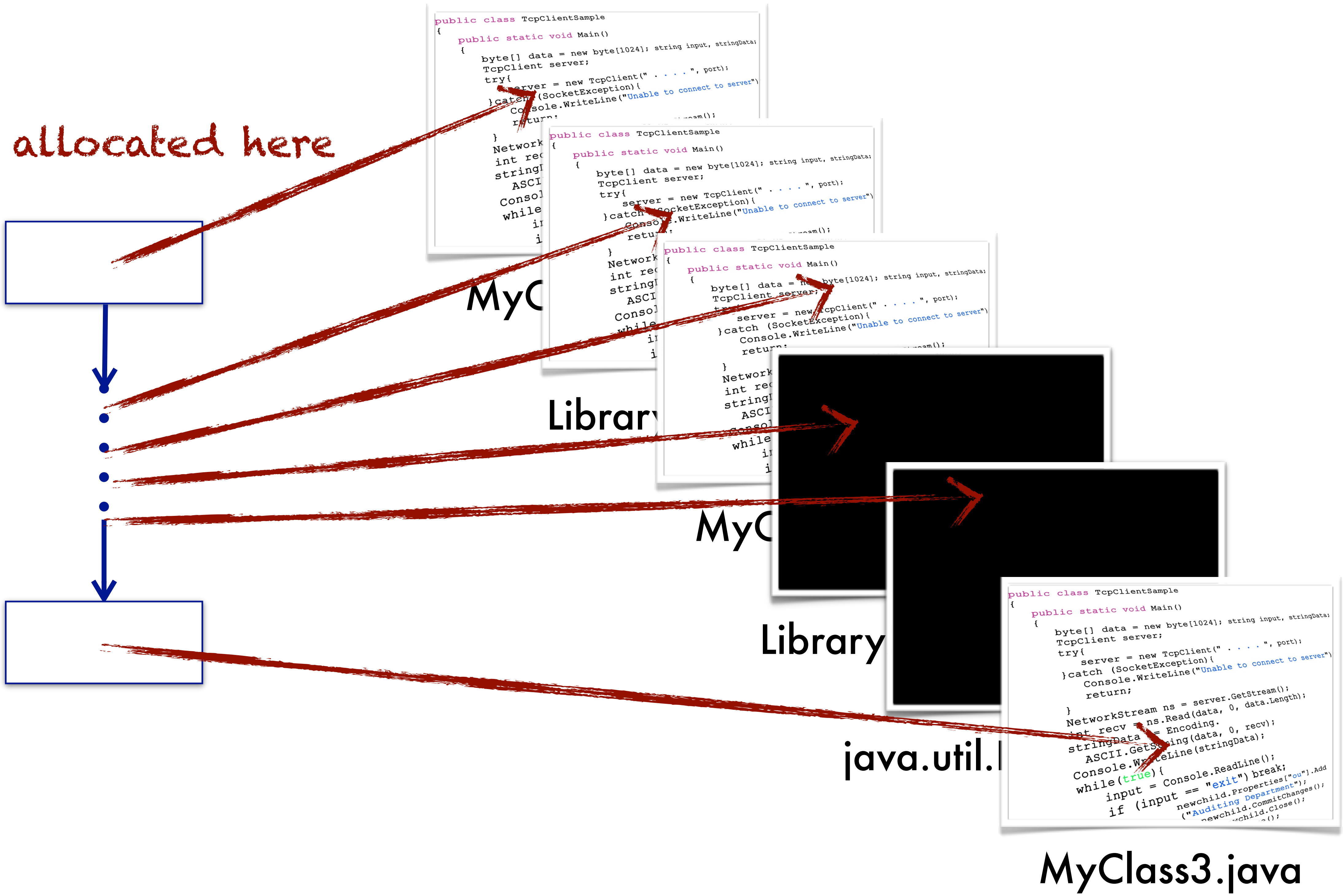
# Manual triage is particularly hard for heap reachability reports.



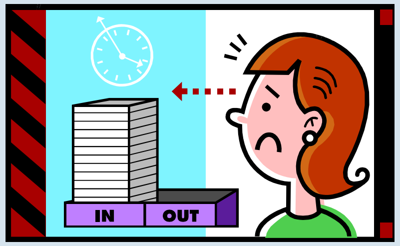



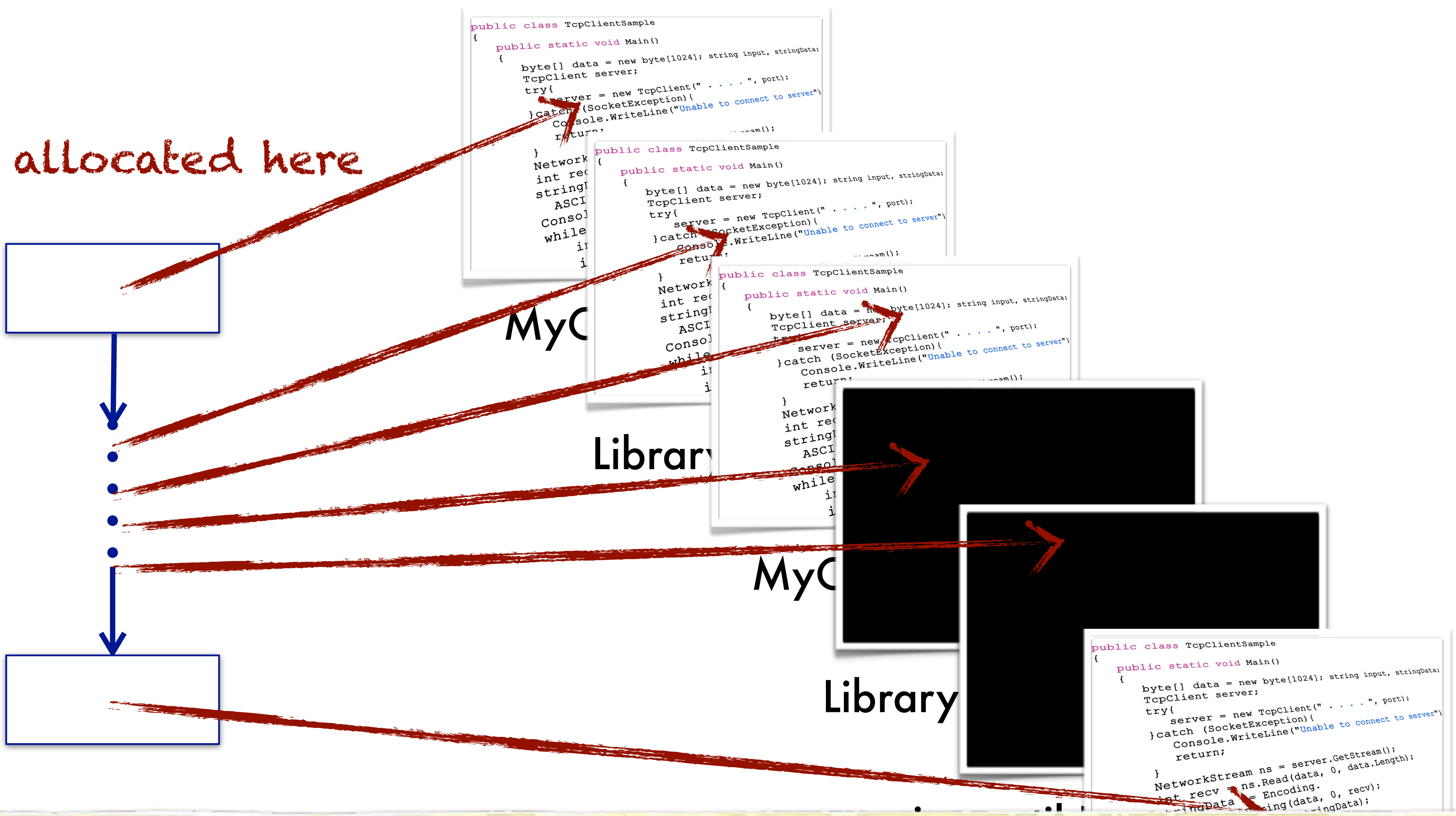
# Manual triage is particularly hard for heap reachability reports.





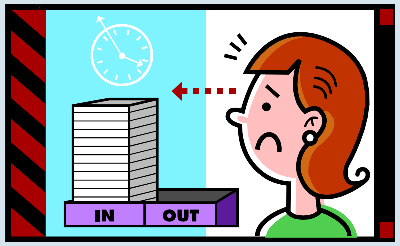
# Manual triage is particularly hard for heap reachability reports.



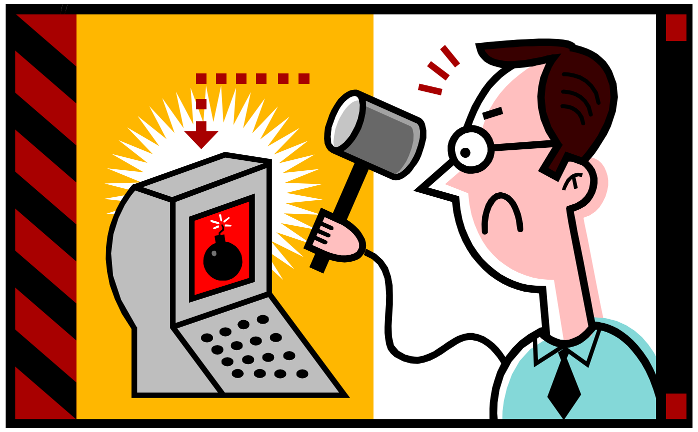



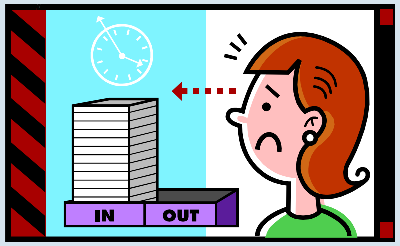

Get abstract heap path + maybe allocation sites
Guesstimate: >1 to 2 hours per alarm to triage “well”

# Examining manual triage ...

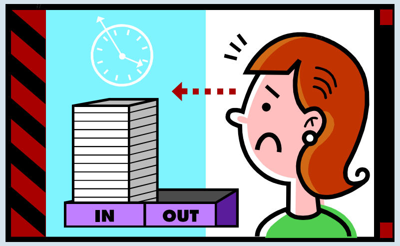


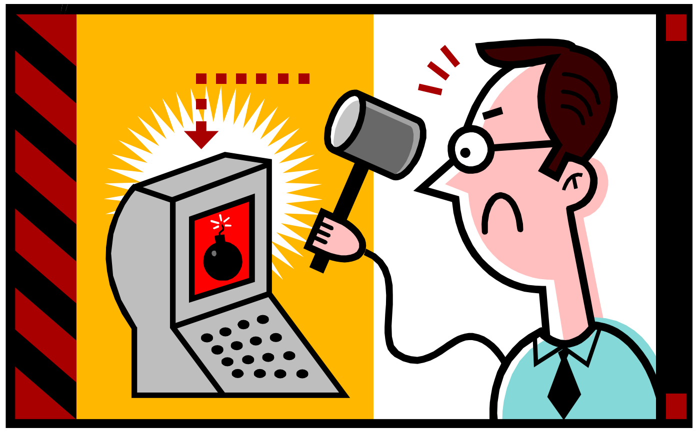
# Examining manual triage ...





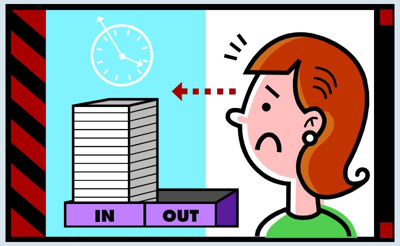
# Examining manual triage …

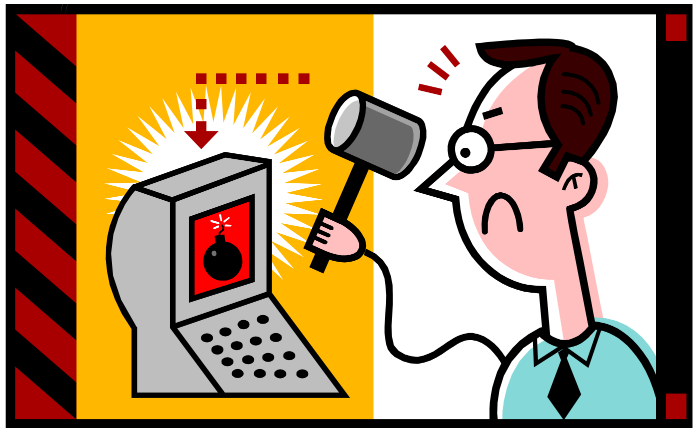


What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what's happening.

# Examining manual triage ...

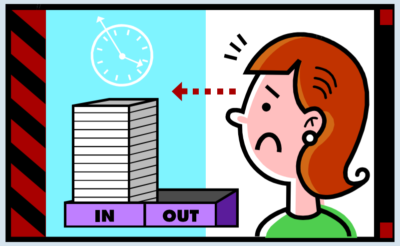


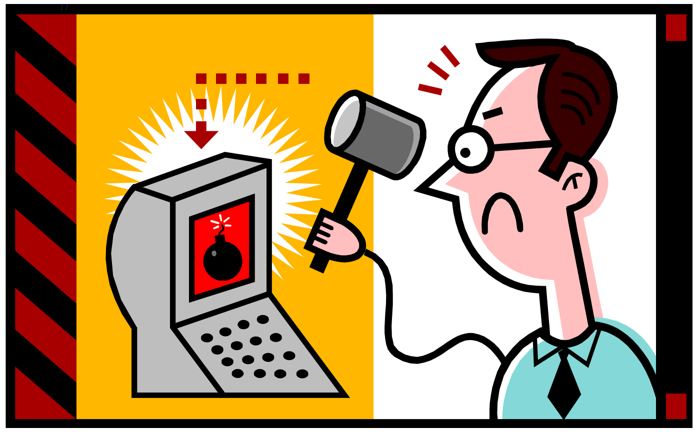



What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what's happening.

We can do this with analysis!

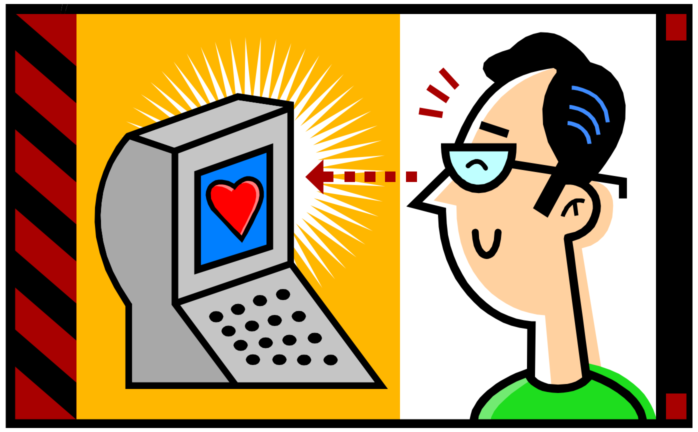
# Examining manual triage ...



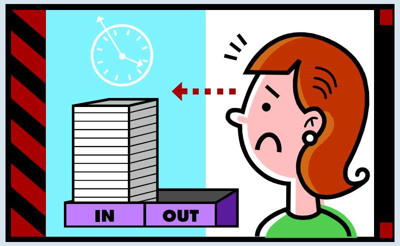



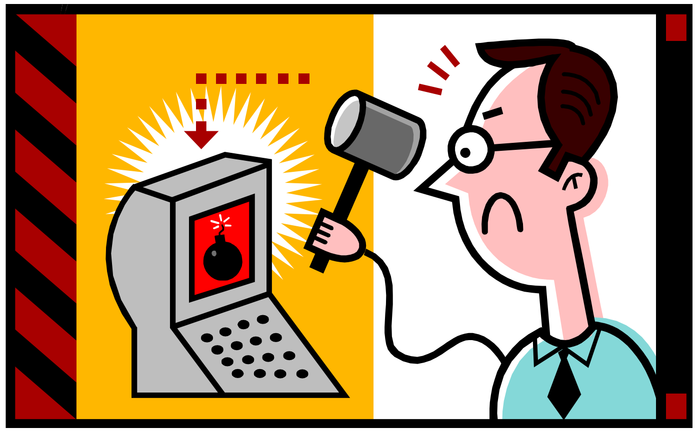
What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what's happening.

We can do this with analysis!



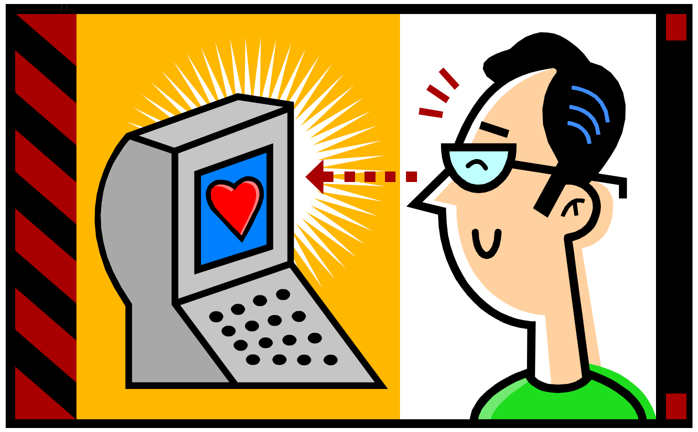
# Examining manual triage …



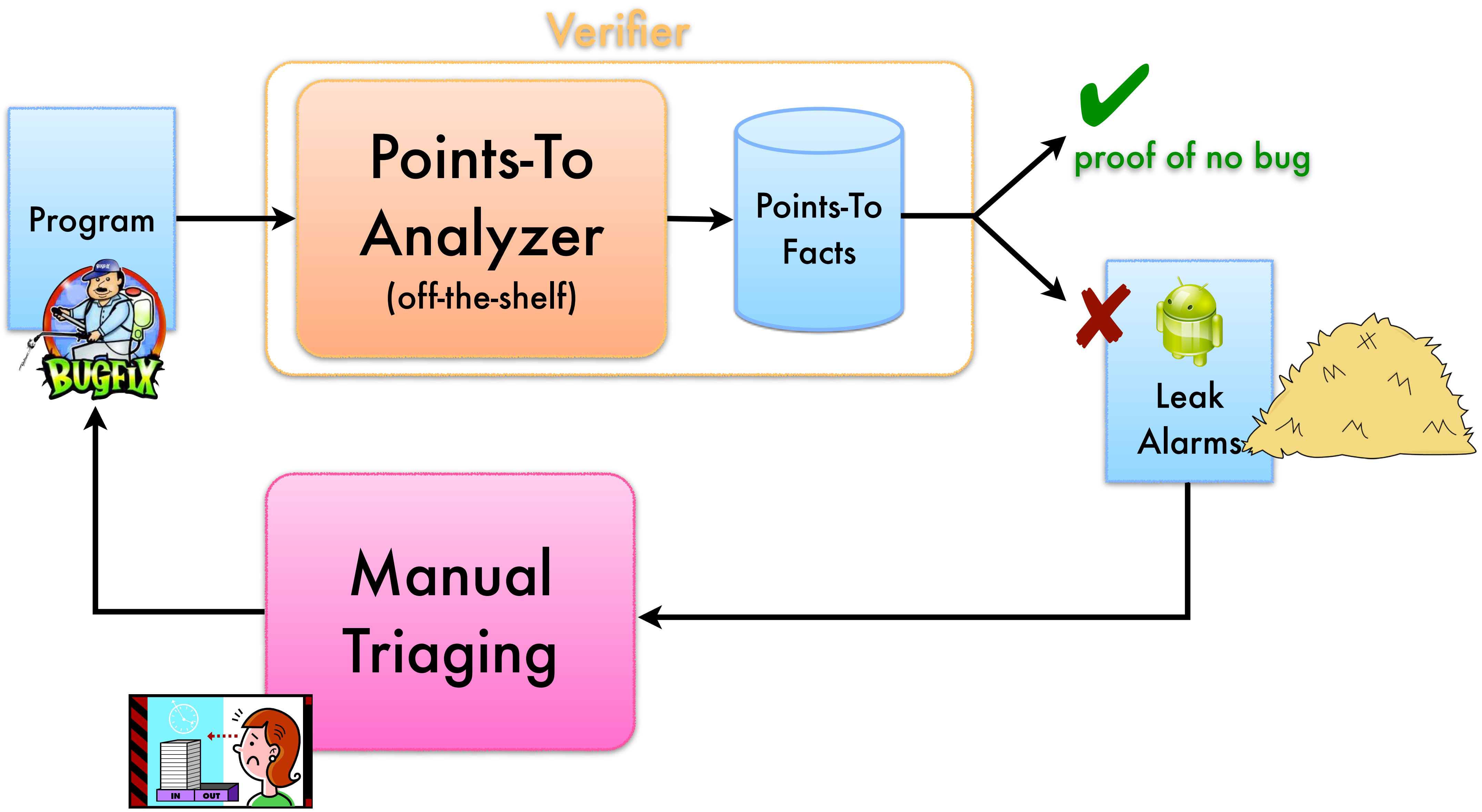
What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what's happening.

**We can do this with analysis!**

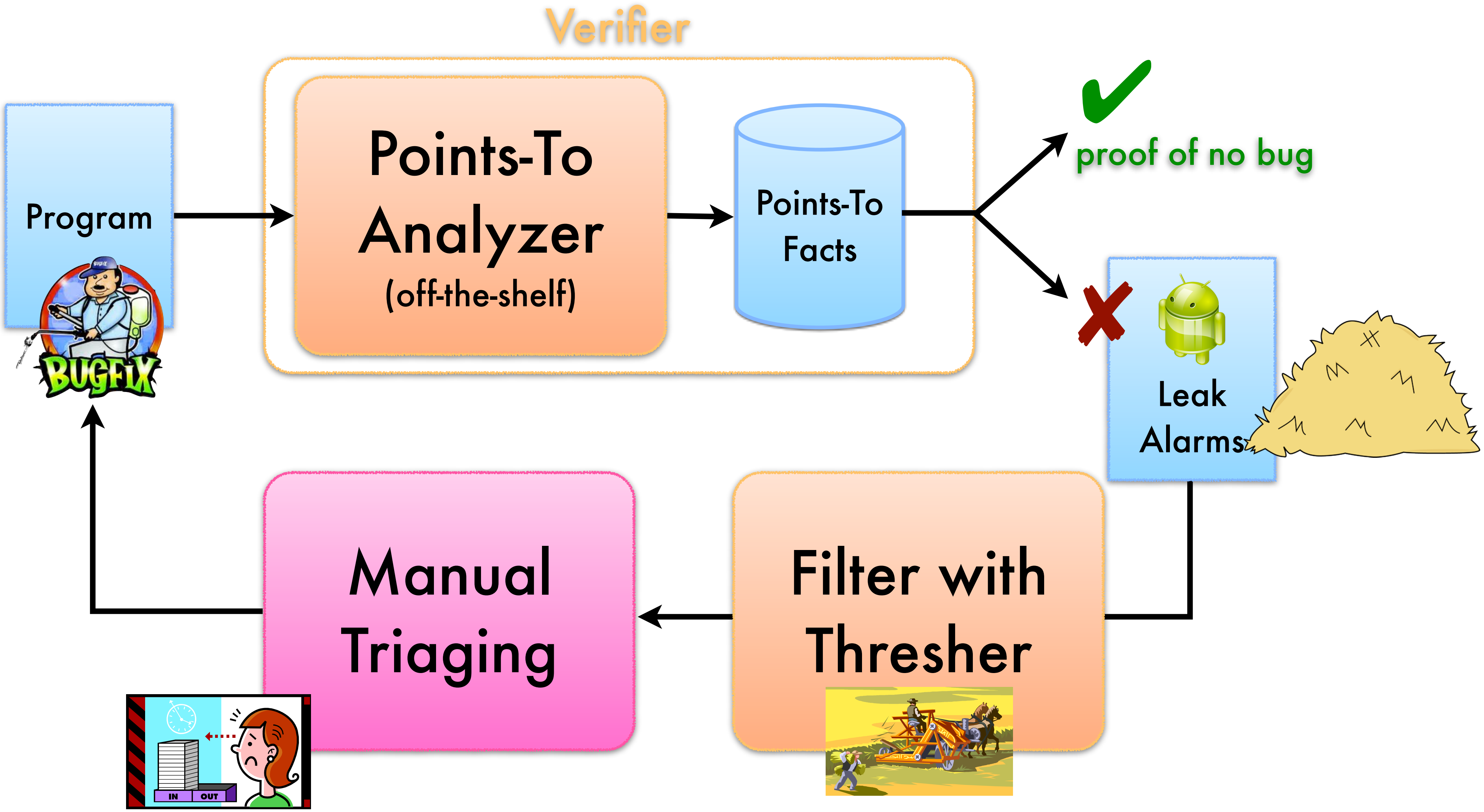


If we filter most false alarms, the user can triage more quickly and get to true bugs earlier (without frustration).

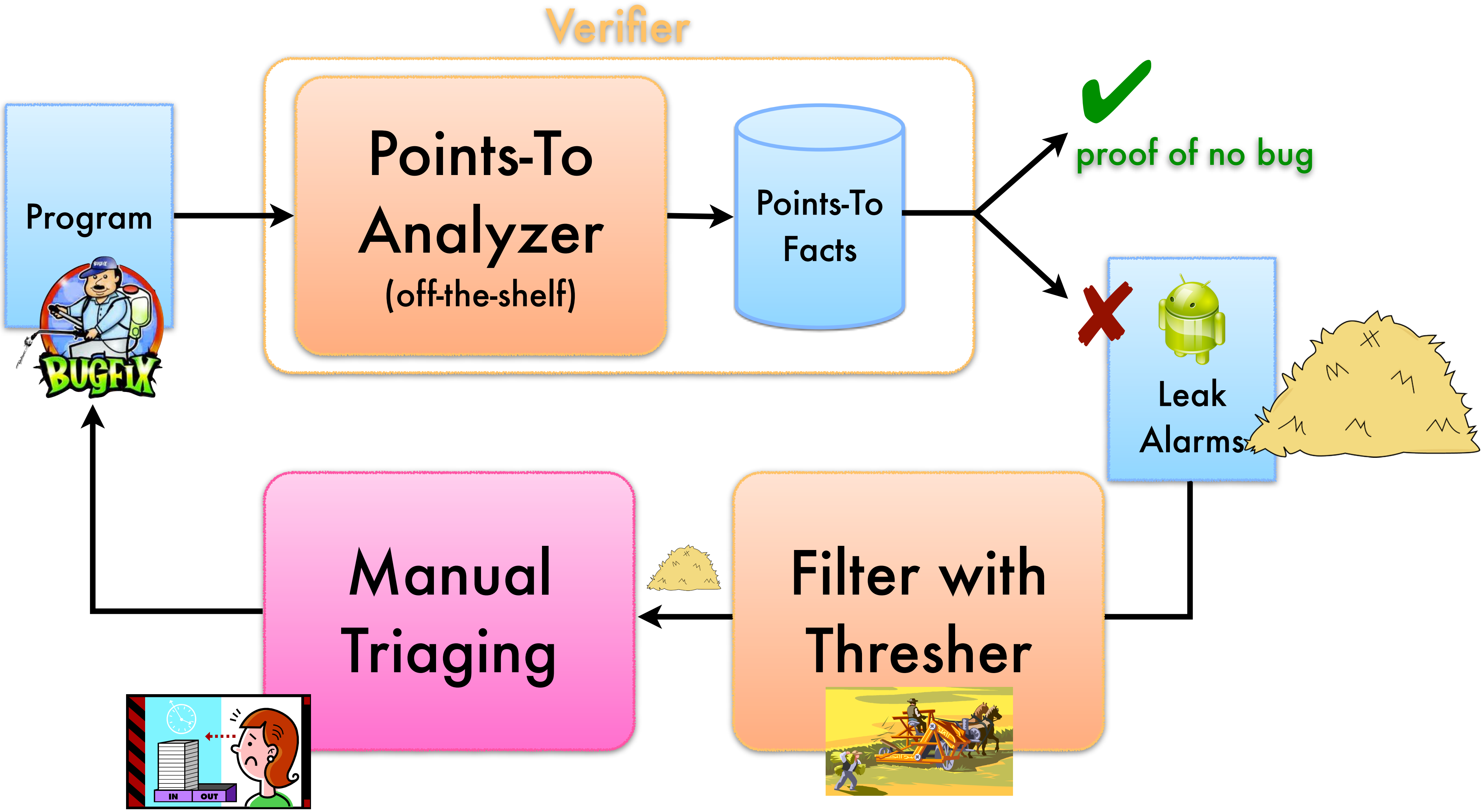
# Thresher **filters** out false alarms by refuting them one-by-one.



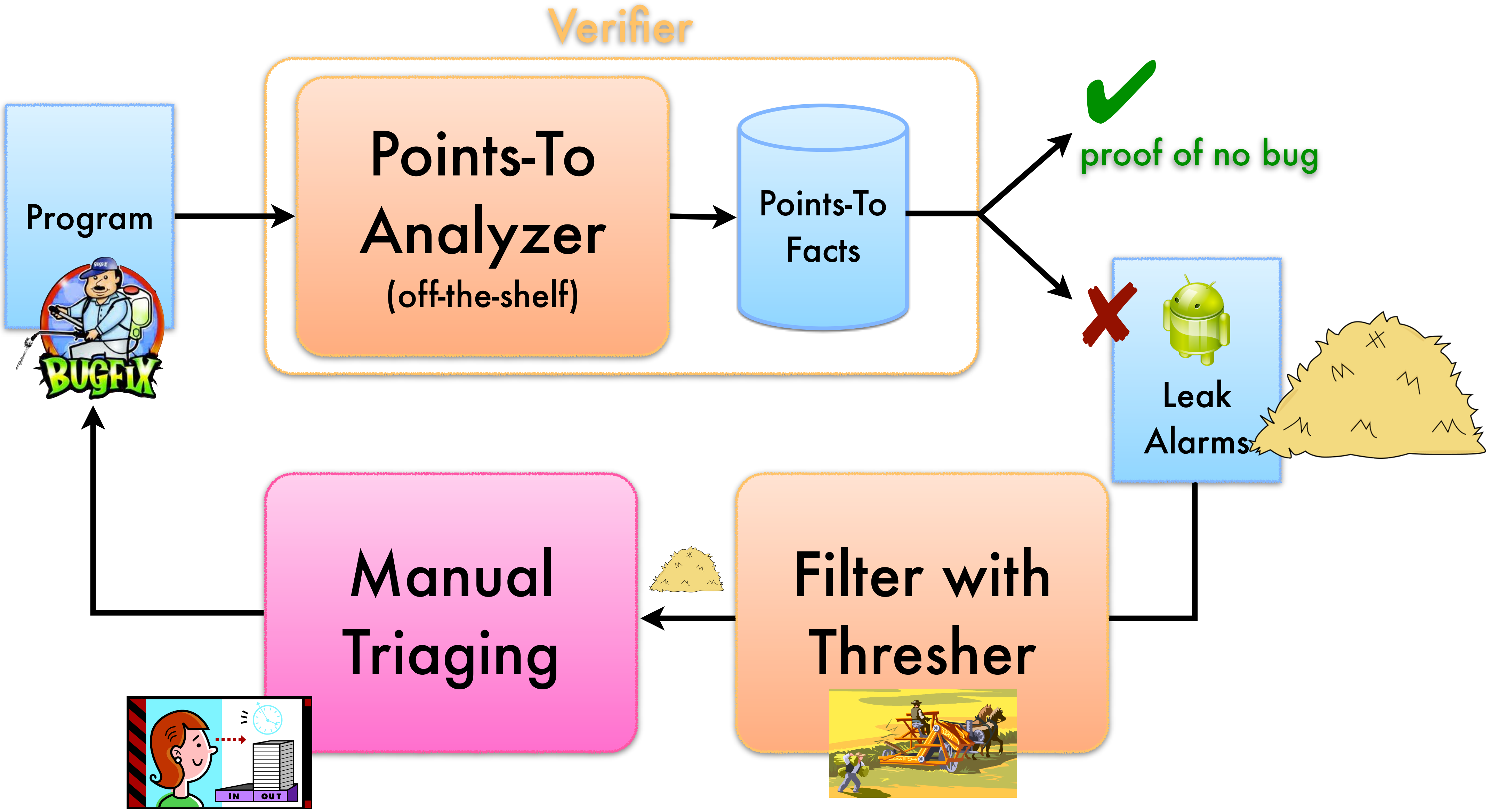
# Thresher **filters** out false alarms by refuting them one-by-one.



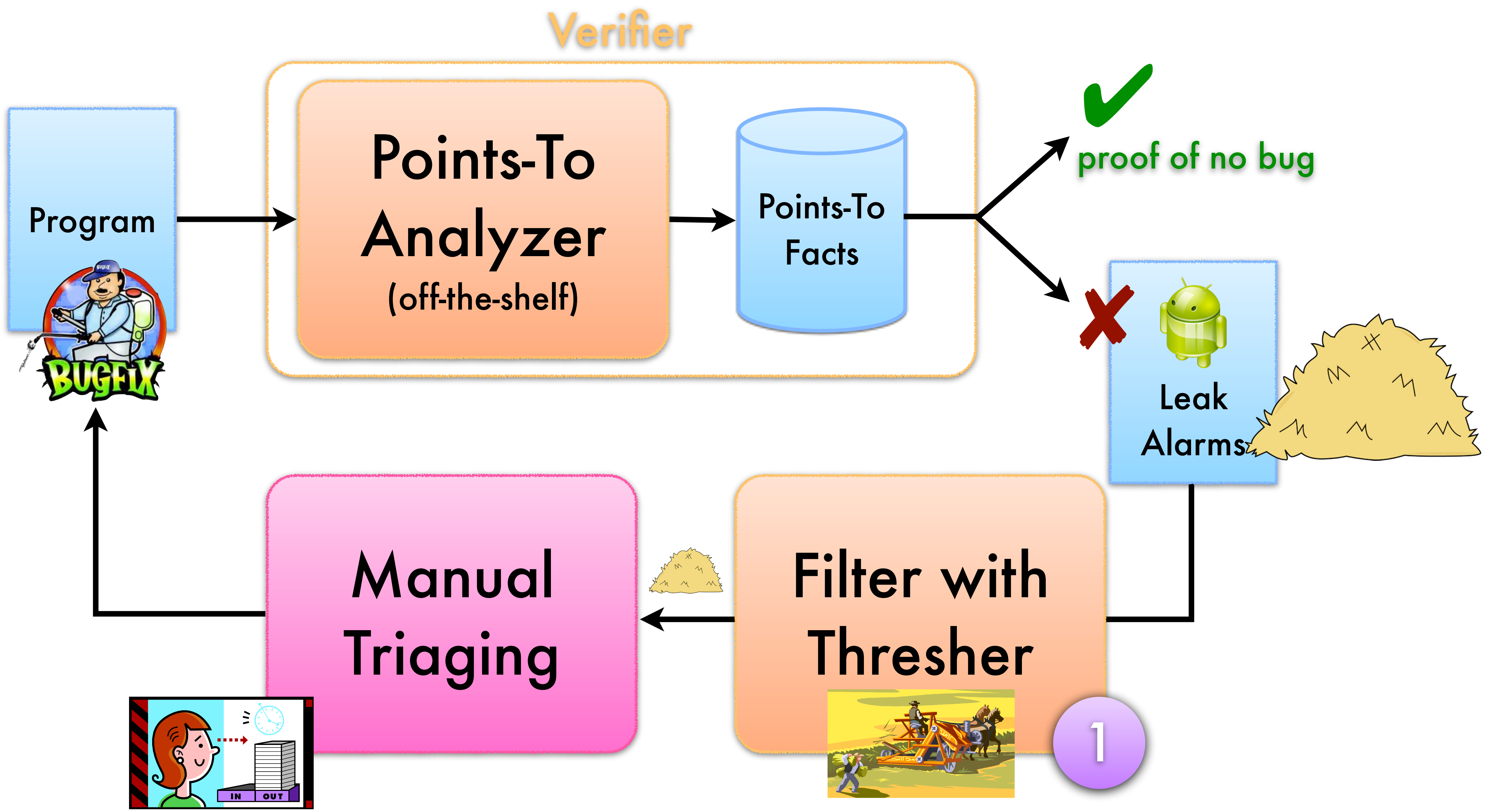
# Thresher **filters** out false alarms by refuting them one-by-one.



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

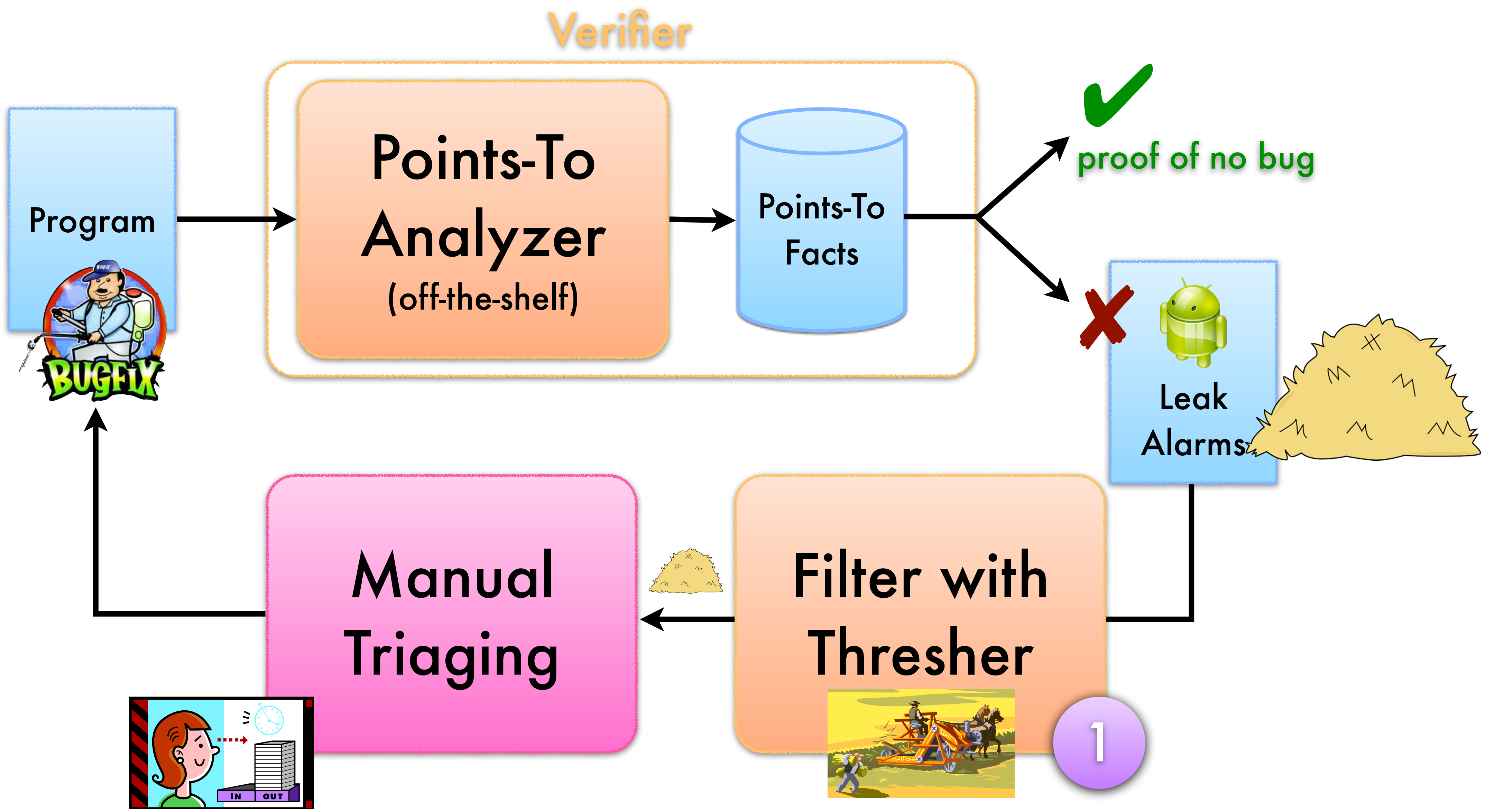


# Thresher filters out false alarms by refuting them one-by-one.




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

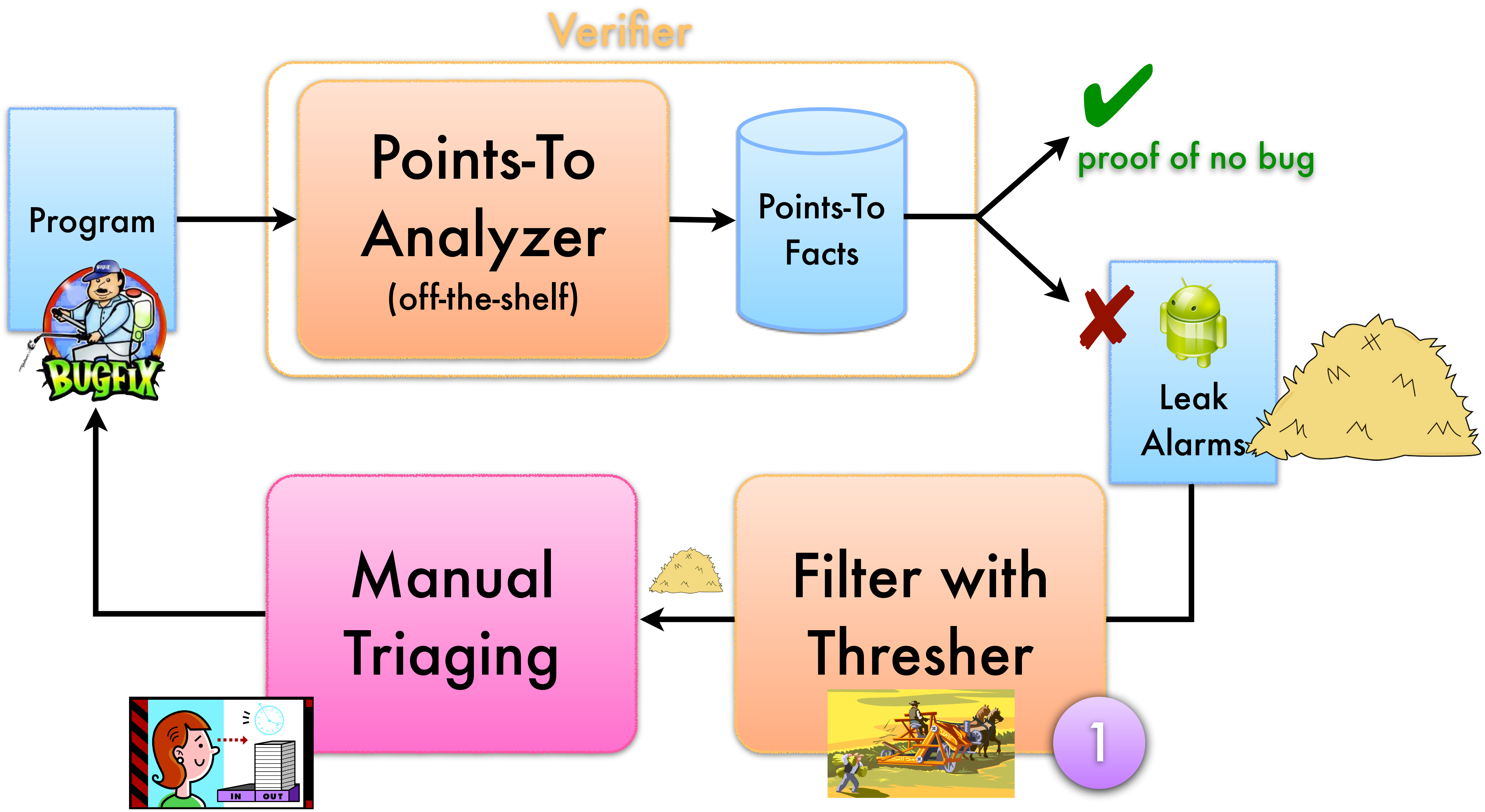
# Thresher **filters** out false alarms by refuting them one-by-one.




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

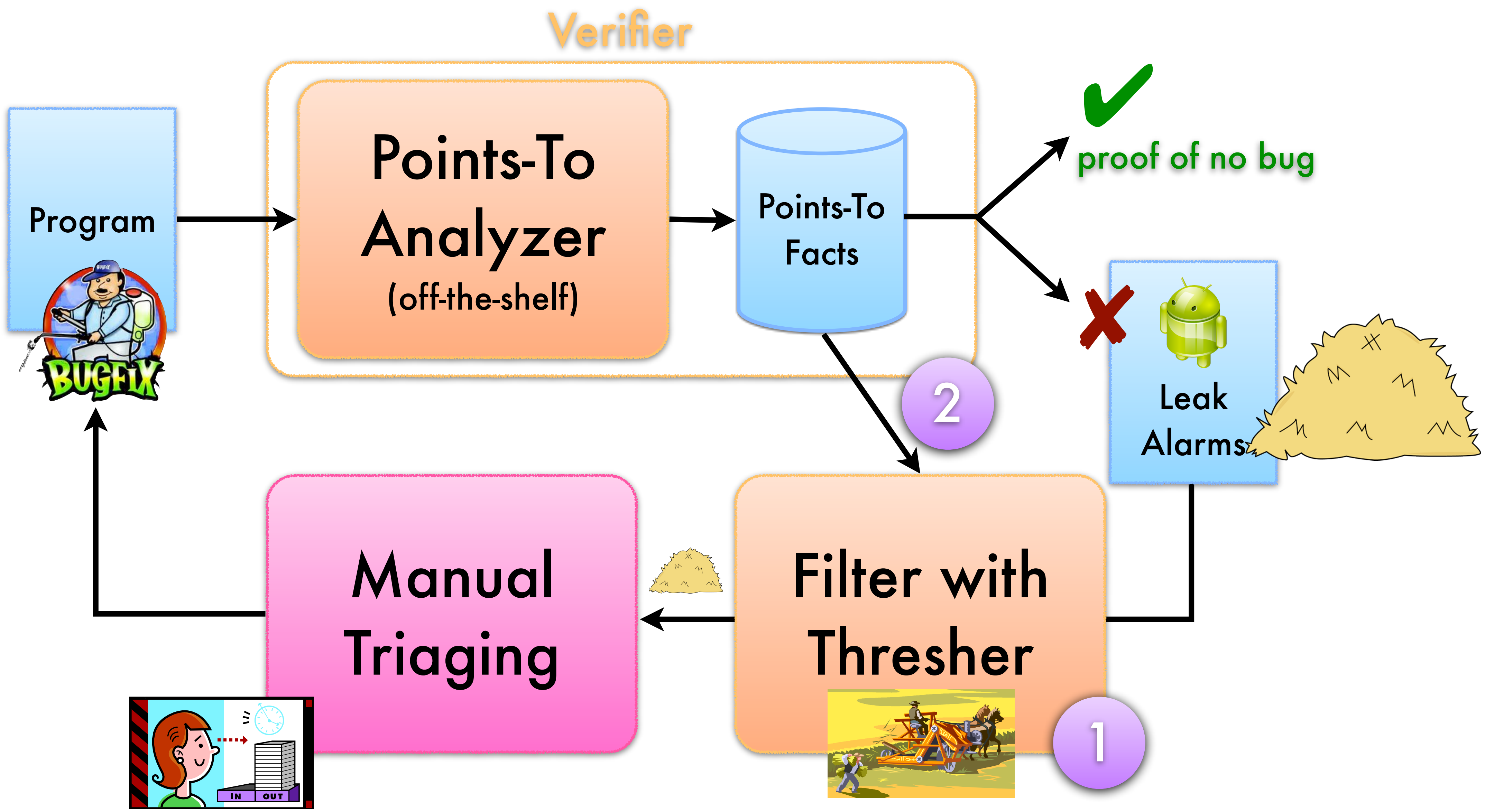
*-sensitive

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




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

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




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

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

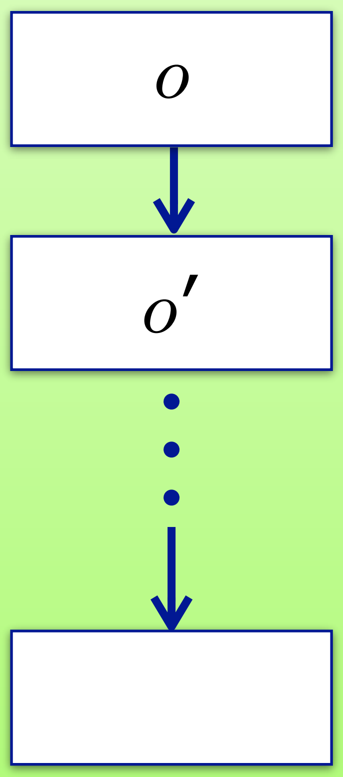
# Refutation analysis is "Proof by Contradiction" with the "But Why?" game

1

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

1

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

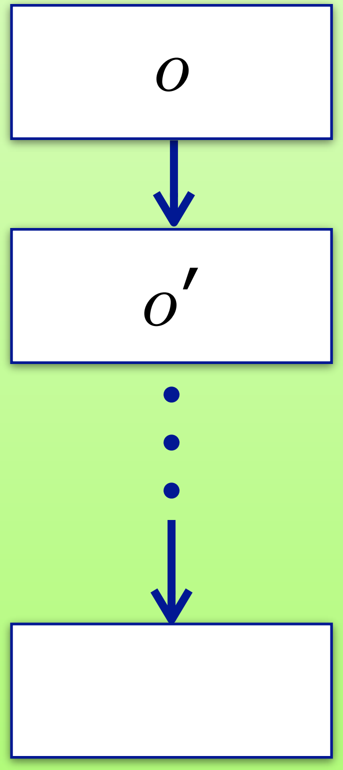



of type T .

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



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



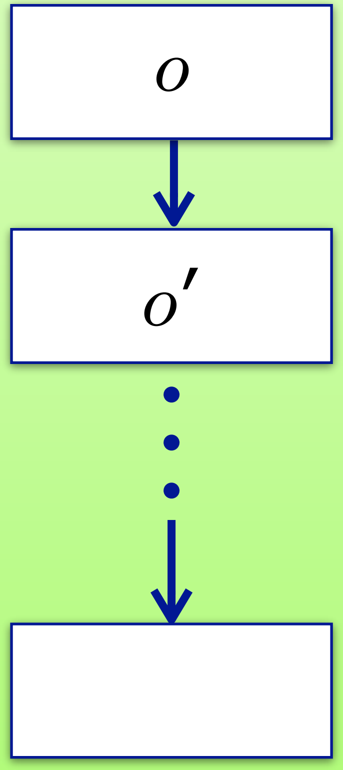

of type T .

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

# Refutation analysis is "Proof by Contradiction" with the "But Why?" game

1

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




of type T .

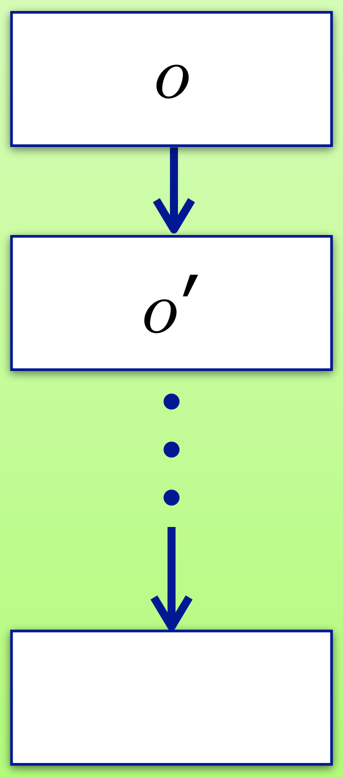
A. Why does object $o$ possibly point to $o'$ ?

B. Because statement $s$ may execute to make $o$ point to $o'$

# Refutation analysis is "Proof by Contradiction" with the "But Why?" game

1

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




of type T .

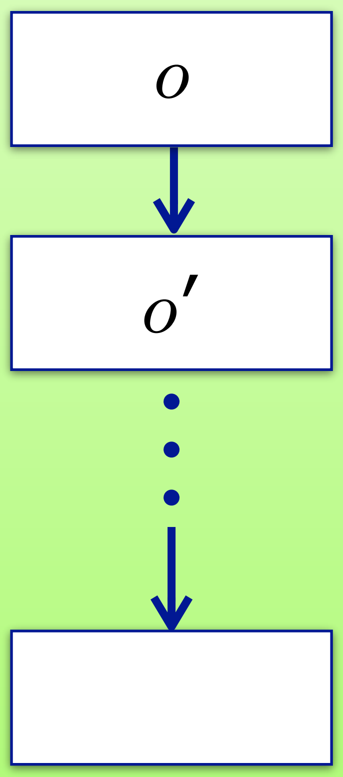
A. Why does object $o$ possibly point to $o'$ ?

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$ ?

# Refutation analysis is "Proof by Contradiction" with the "But Why?" game

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




of type T .

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

B. Because statement $s$ may execute to make $o$ point to $o'$

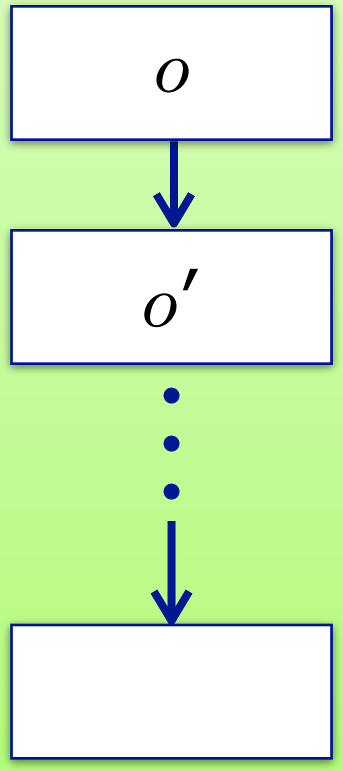
A. Why does statement $s$ cause $o$ to point to $o'$ ?

B. Because before statement $s$, the program state could satisfy formula $\varphi$

# Refutation analysis is "Proof by Contradiction" with the "But Why?" game

1

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




of type T .

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

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$ ?

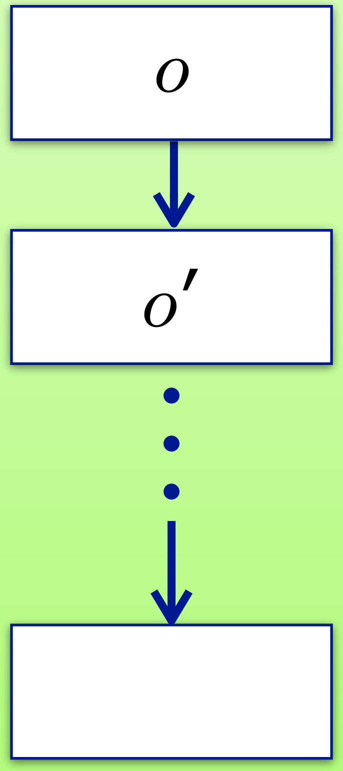
B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

# Refutation analysis is "Proof by Contradiction" with the "But Why?" game



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




of type T .

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

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$ ?

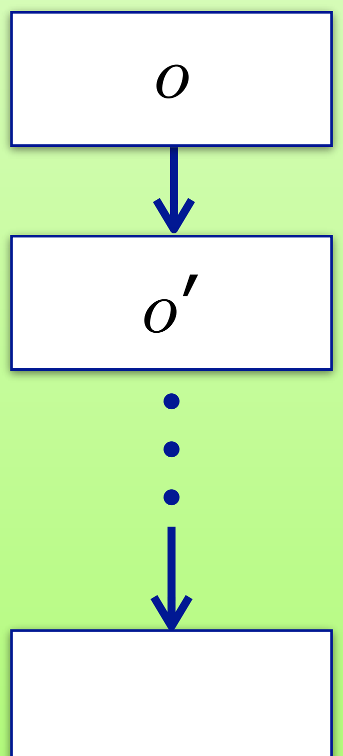
B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

# Refutation analysis is "Proof by Contradiction" with the "But Why?" game

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




of type T .

A just asks "but why?"

B reasons about program semantics

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

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$ ?

B. Because before statement $s$, the program state could satisfy formula $\varphi$

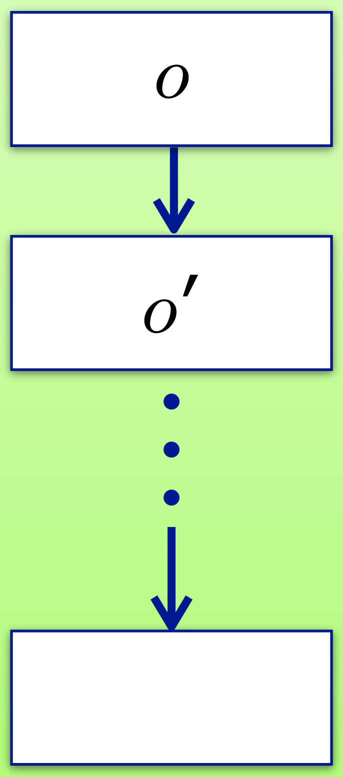
A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

# Refutation analysis is "Proof by Contradiction" with the "But Why?" game

1

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




of type T .

A just asks "but why?"

B reasons about program semantics

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

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$ ?

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

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

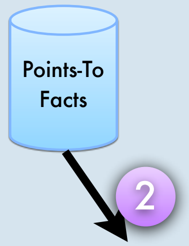
# Leverage first analysis by designing specialized constraint forms

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

# Leverage first analysis by designing **specialized** constraint forms

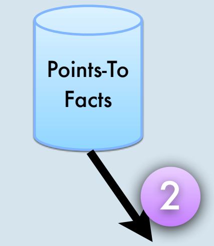



B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

# Leverage first analysis by designing **specialized** constraint forms

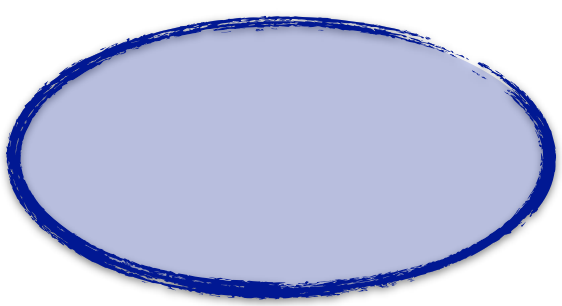



B. Because before statement $s$, the program state could satisfy formula $\varphi$

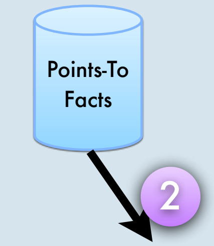
A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

$\varphi$



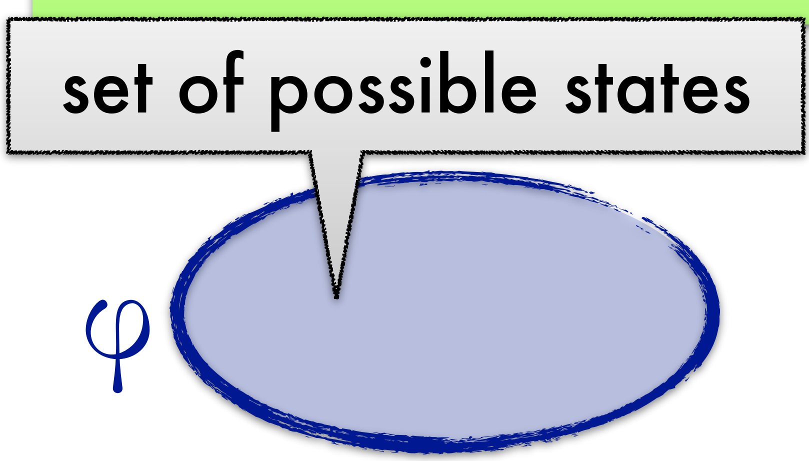
# Leverage first analysis by designing **specialized** constraint forms



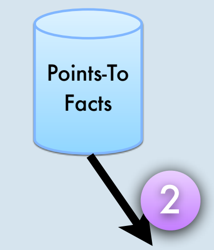

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$



# Leverage first analysis by designing specialized constraint forms



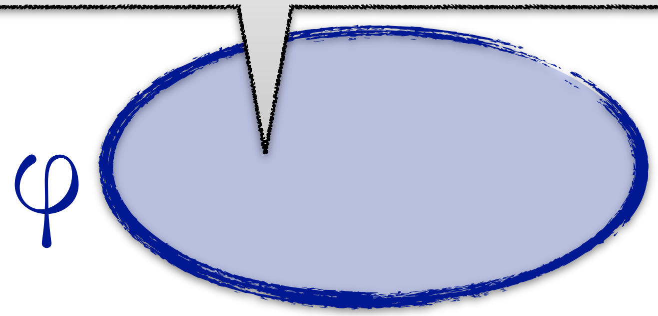

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

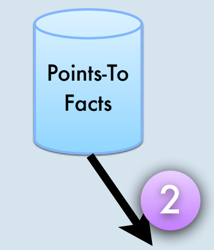
… statement $s'$, … a $\varphi'$

if empty, then refuted (A wins)

set of possible states



# Leverage first analysis by designing **specialized** constraint forms




B. Because before statement $s$, the program state could satisfy formula $\varphi$

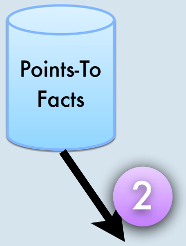
A. Why can the state before statement $s$ satisfy $\varphi$?

B. … statement $s'$, … a $\varphi'$

if empty, then refuted (A wins)

set of possible states

# Leverage first analysis by designing specialized constraint forms

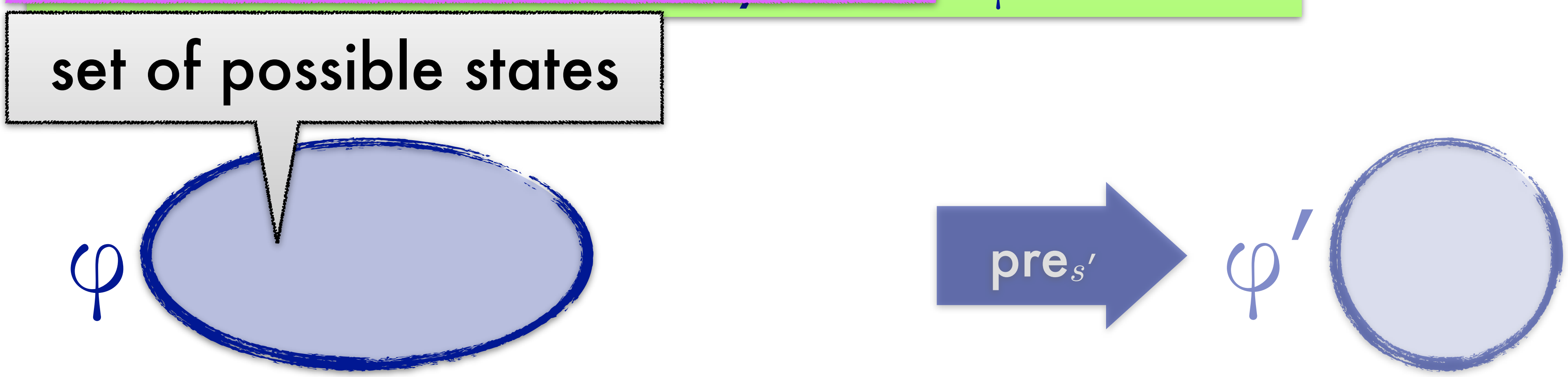



B. Because before statement $s$, the program state could satisfy formula $\varphi$

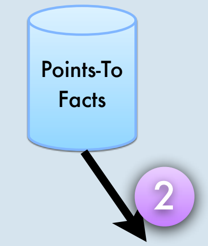
A. Why can the state before statement $s$ satisfy $\varphi$?

statement $s'$,
a $\varphi'$

if empty, then refuted (A wins)



# Leverage first analysis by designing specialized constraint forms



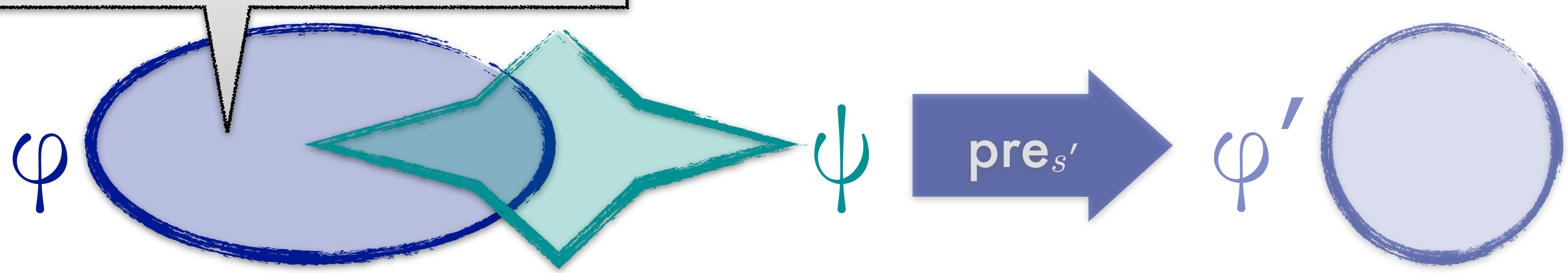

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

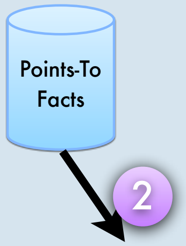
statement $s'$,
a $\varphi'$

if empty, then refuted (A wins)

set of possible states



# Leverage first analysis by designing **specialized** constraint forms



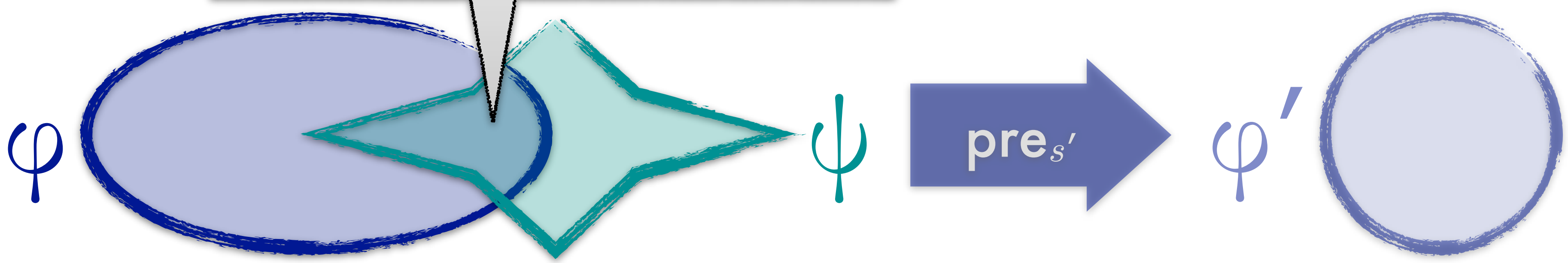

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

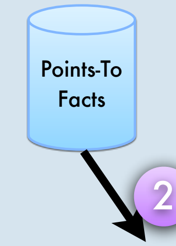
… statement $s'$,
… a $\varphi'$

if empty, then refuted (A wins)

set of possible states



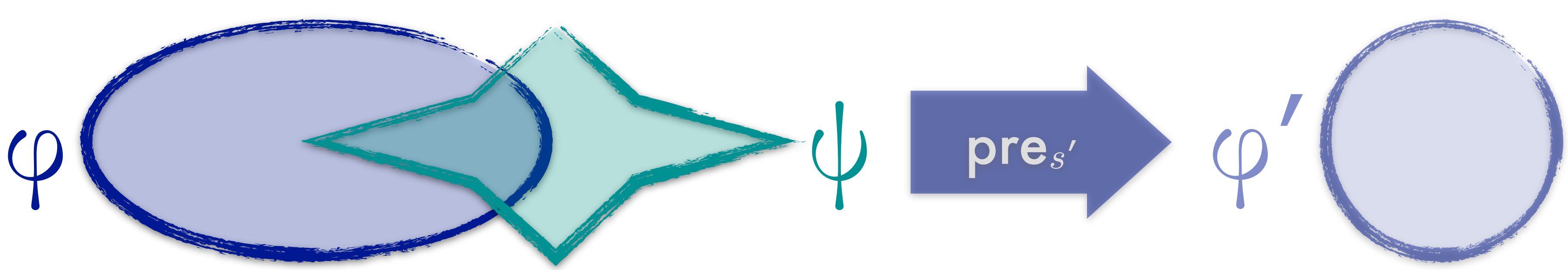
# Leverage first analysis by designing **specialized** constraint forms



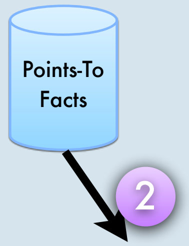

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$



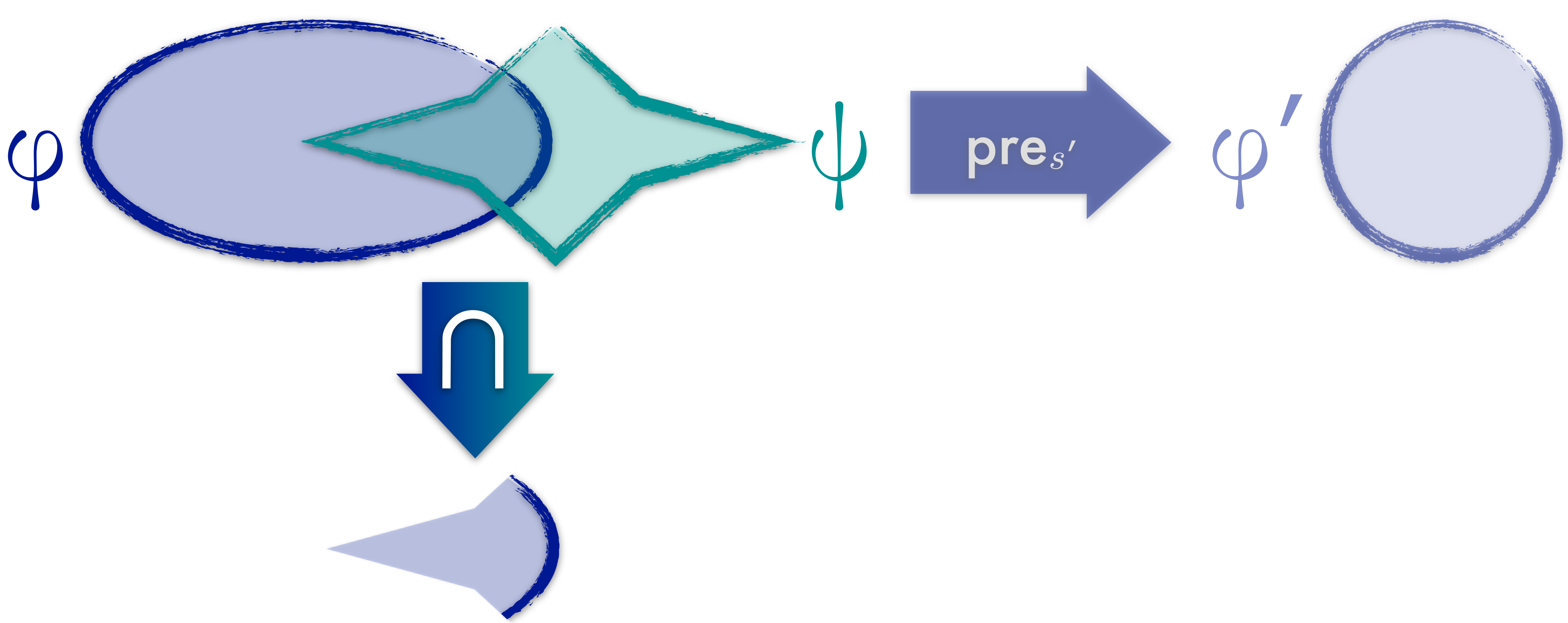
# Leverage first analysis by designing specialized constraint forms



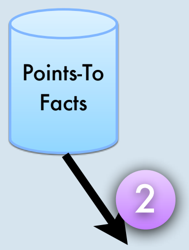

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$



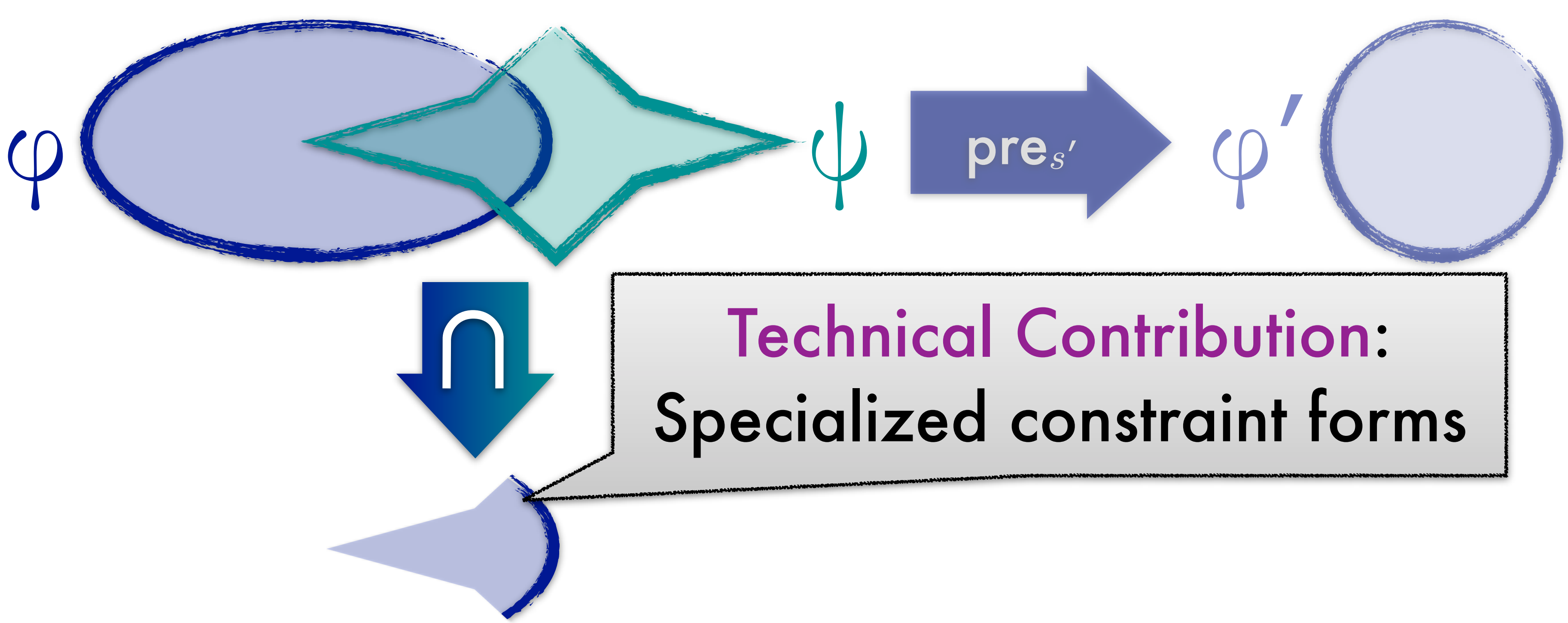
# Leverage first analysis by designing **specialized** constraint forms



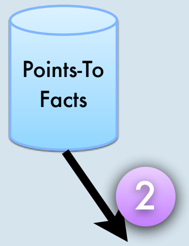

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$



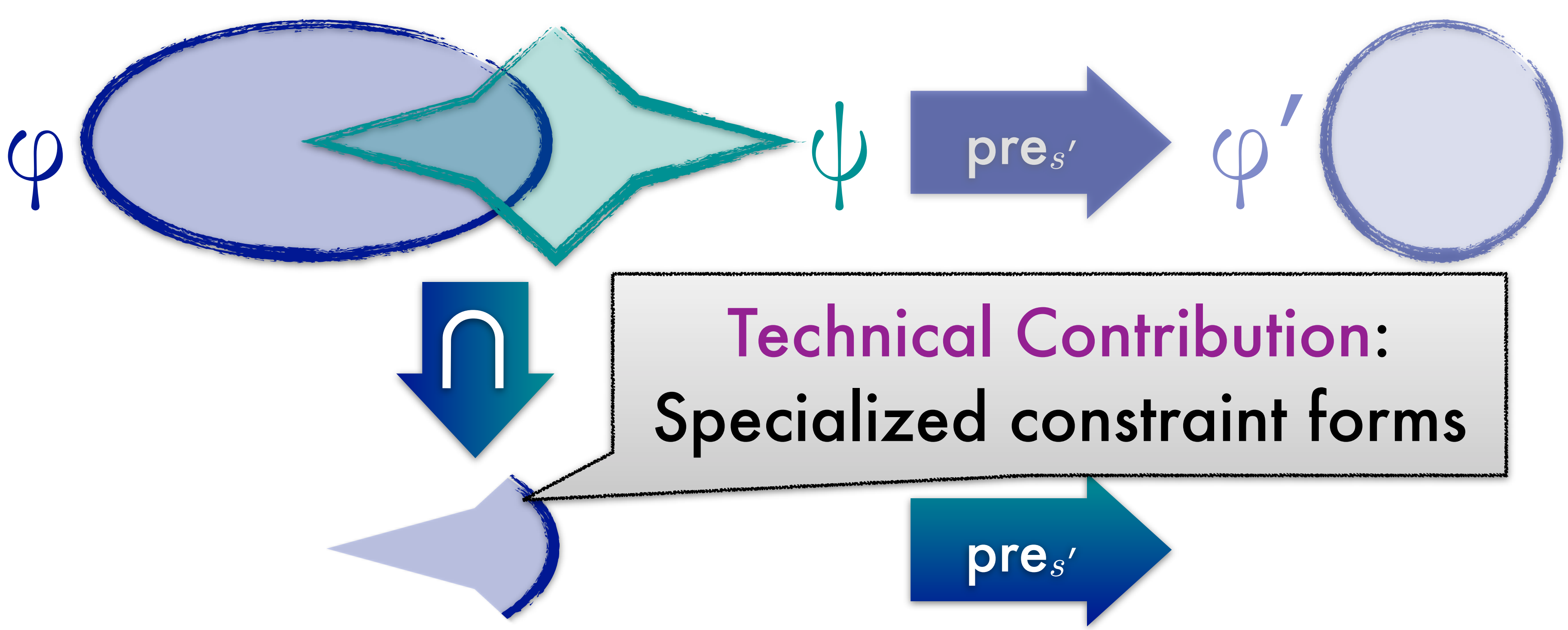
# Leverage first analysis by designing specialized constraint forms



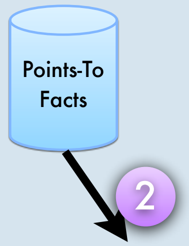

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$



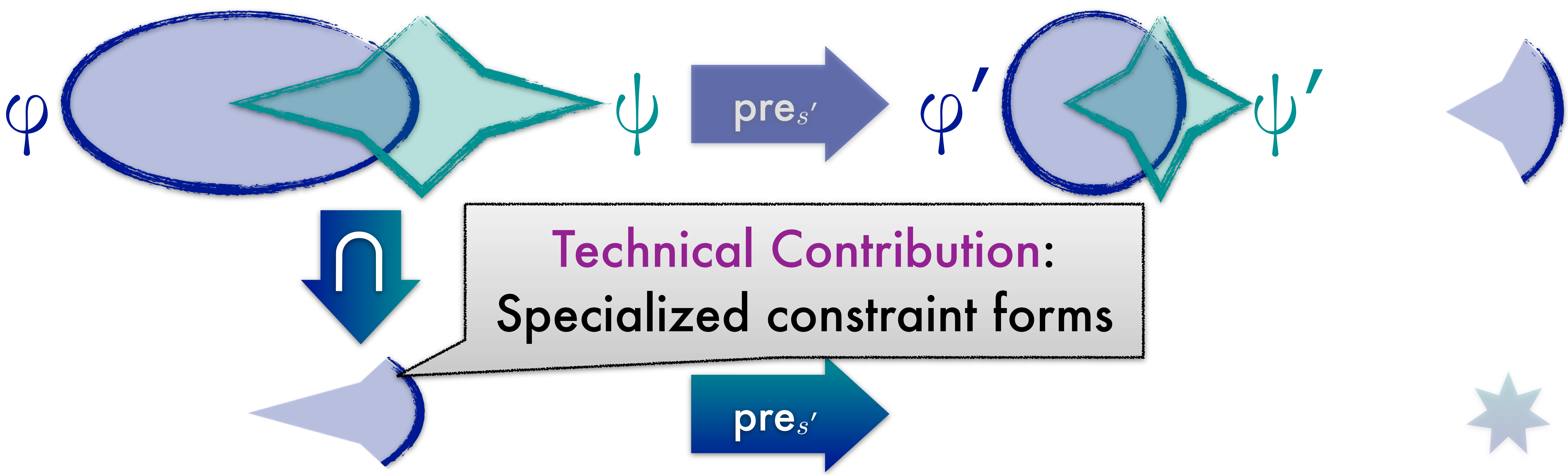
# Leverage first analysis by designing specialized constraint forms



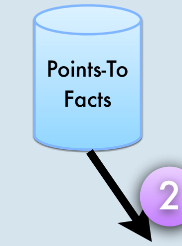

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$



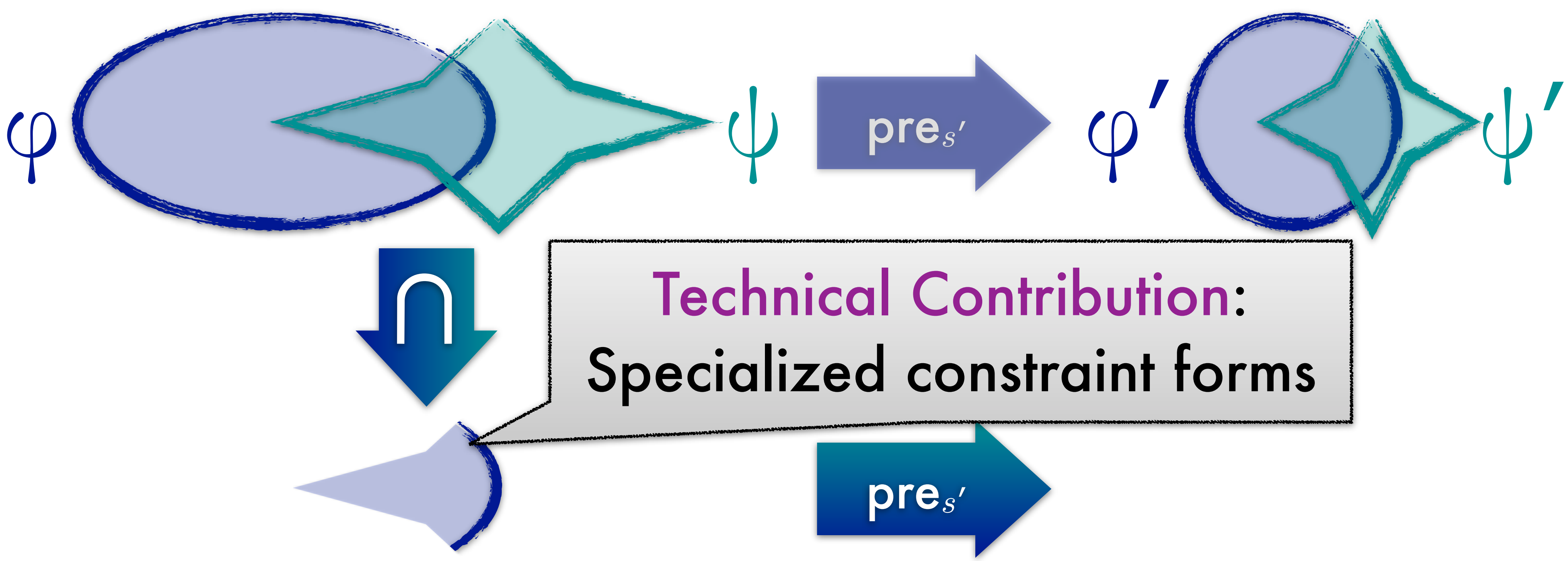
# Leverage first analysis by designing specialized constraint forms

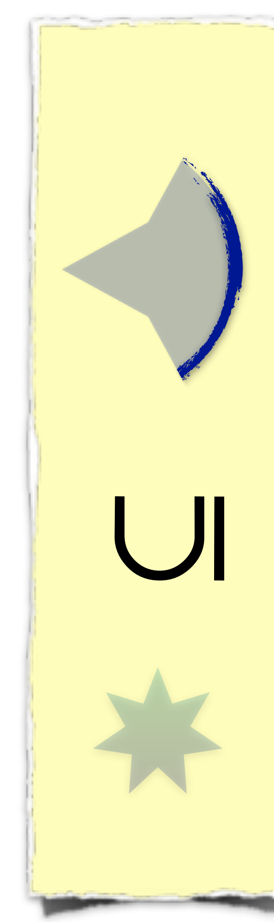



B. Because before statement $s$, the program state could satisfy formula $\varphi$

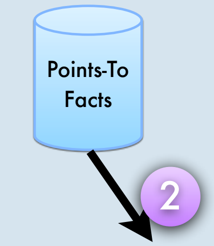
A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$



# Leverage first analysis by designing specialized constraint forms

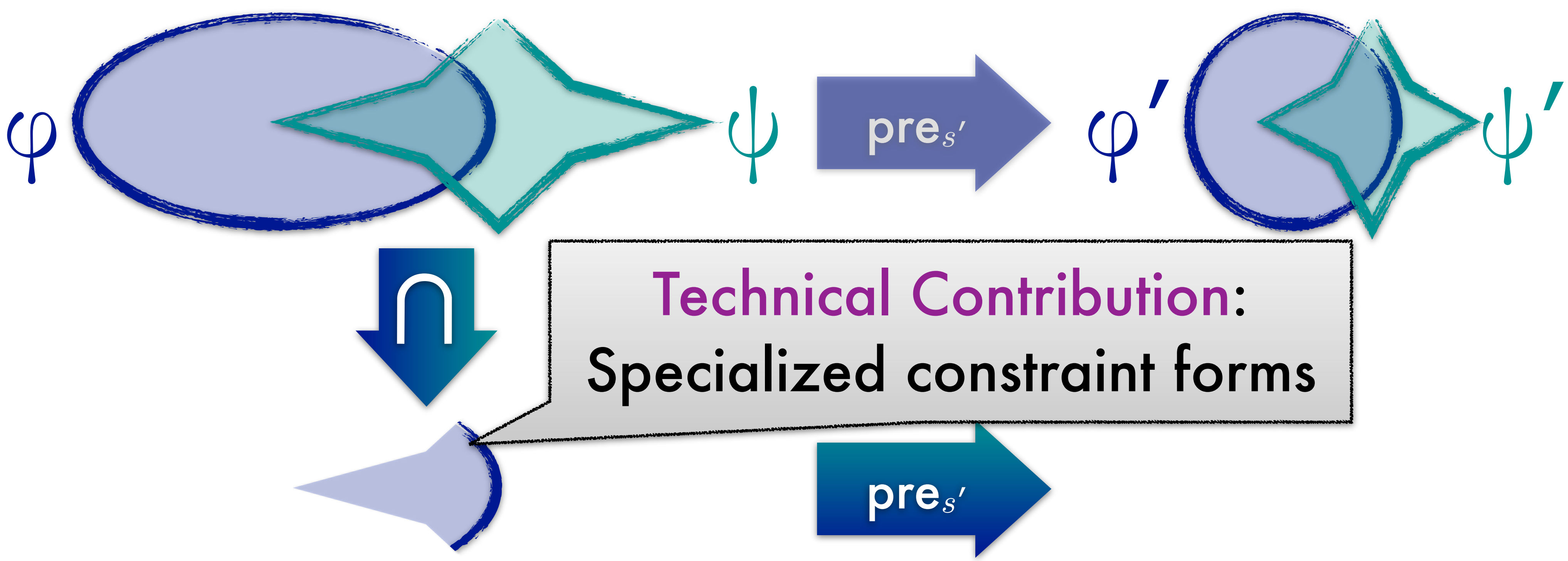


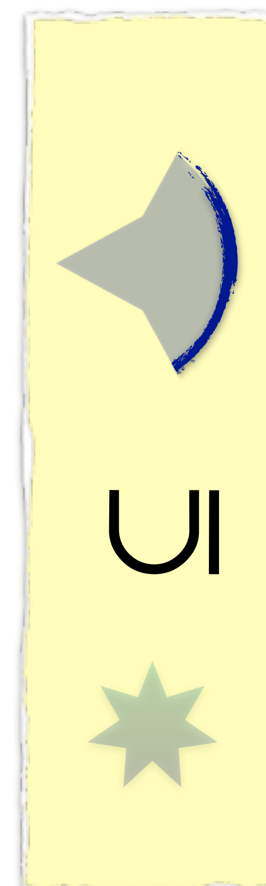

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

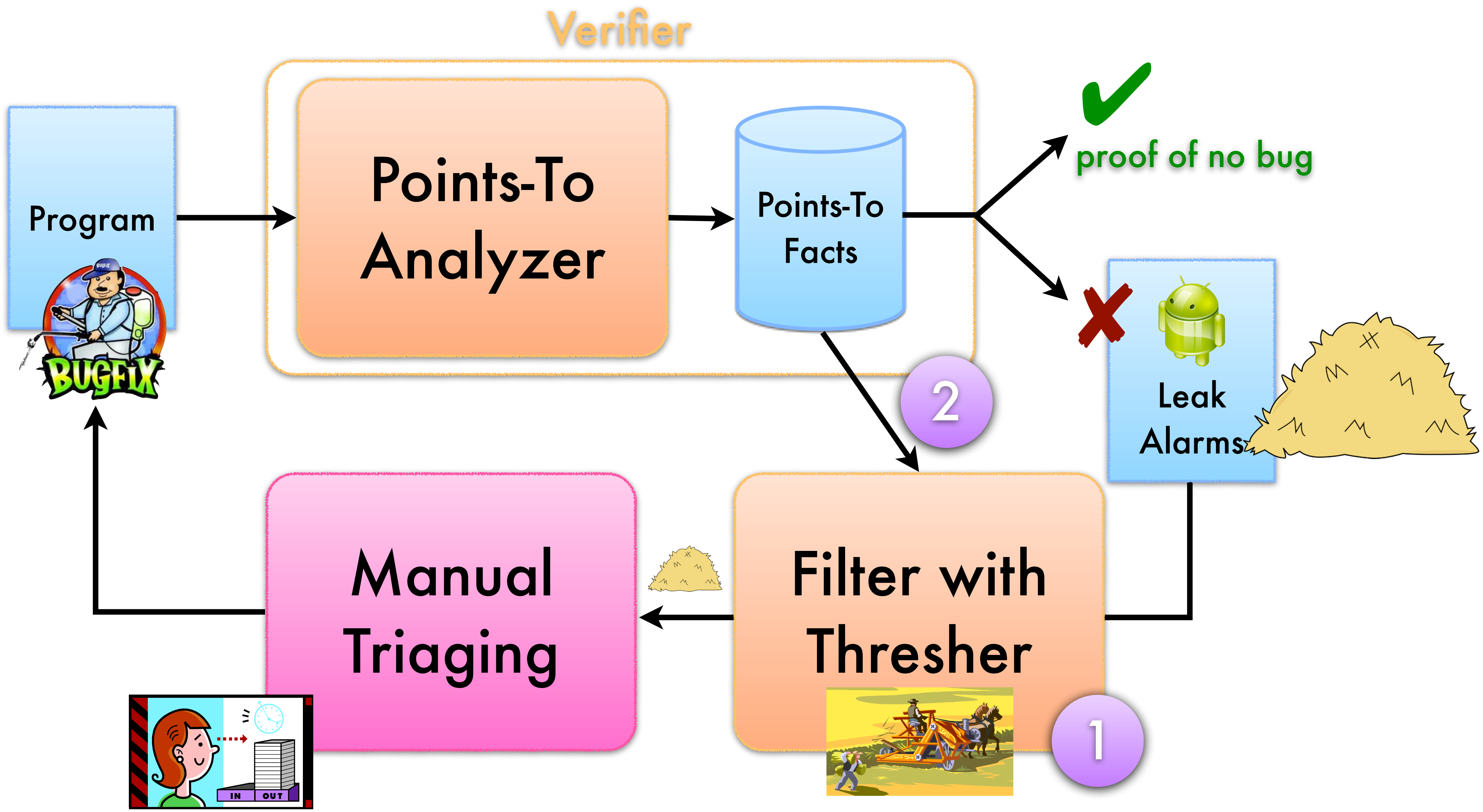
B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

**Specialized constraint forms makes finding refutations feasible**



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




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

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

# Is Thresher effective at filtering?


Thresher analyzes Java VM bytecode

7 Android app benchmarks

- 2,000 to 40,000 source lines of code
- + 880,000 sources lines of Android framework code

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

# Is Thresher effective at filtering?

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

# Is Thresher effective at filtering?

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

`staticfield-Activity` pairs

# Is Thresher effective at filtering?

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

triage "well" at ~1–2 hours per alarm

staticfield-Activity pairs

# Is Thresher effective at filtering?

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

staticfield-Activity pairs

Filtered

# Is Thresher effective at filtering?

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

`staticfield-Activity` pairs

Filtered

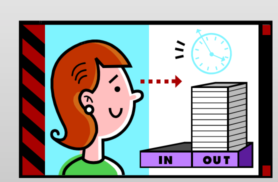
# Is Thresher effective at filtering?

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

`staticfield-Activity` pairs

Filtered

Manual



# Is Thresher effective at filtering?

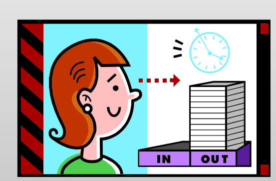
| Program | LOC | Points-To Alarms | Thresher Refuted | True Bugs |
|---|---|---|---|---|
| PulsePoint | unknown | 16 | 8 | 8 |
| StandupTimer | 2K | 25 | 15 | 0 |
| DroidLife | 3K | 3 | 0 | 3 |
| SMSPopUp | 7K | 5 | 1 | 4 |
| aMetro | 20K | | | 36 |
| K9Mail | 40K | | | 64 |
| **Total** | **72K** |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

triage "well" at 10–15 minutes per

`staticfield`-`Activity` pairs

Filtered

Manual



# Is Thresher effective at filtering?

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

# Is Thresher effective at filtering?

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

# Is Thresher effective at filtering?

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

< ~coffee to lunch break

# Is Thresher effective at filtering?

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

# Is Thresher effective at filtering?

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

% after filtering

# Is Thresher effective at filtering?

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

% after filtering

# Is Thresher effective at filtering?

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

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

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

Guesstimate

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

Triage "ok" without: ~30 hours

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

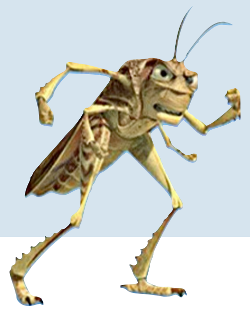
False alarms down to 17% from 63% (points-to analysis only)

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

Android
OS


... in the process of finding leaks in apps

# Find the Android's HashMap bug ...



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

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

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

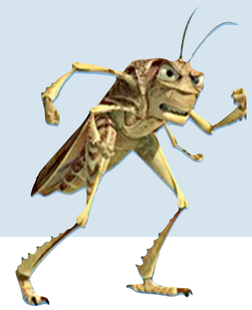
# Find the Android's HashMap bug …

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

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

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

# Find the Android's HashMap bug …



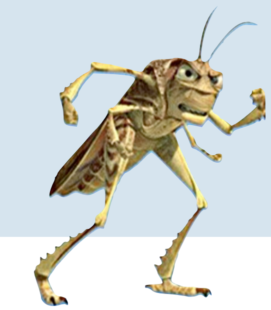
null object pattern: should not be written to

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

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

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

# Find the Android's HashMap bug ...



null object pattern: should not be written to

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

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

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

allocate new backing array on first write

# Find the Android's HashMap bug ...

null object pattern: should not be written to

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

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

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

allocate new backing array on first write

# Find the Android's HashMap bug ...

null object pattern: should not be written to

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

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

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

allocate new backing array on first write

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

# Find the Android's HashMap bug ...

null object pattern: should not be written to

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

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

  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 `HashMaps` created in the future will be corrupted.

# Find the Android's HashMap bug ...

```
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
  }
}
```

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

return 0

return "evil" content

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

# Find the Android's HashMap bug ...

```
cla
  s
  H
  v
  }
```

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

**We reported this, Google fixed it**

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

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

return 0

return "evil" content

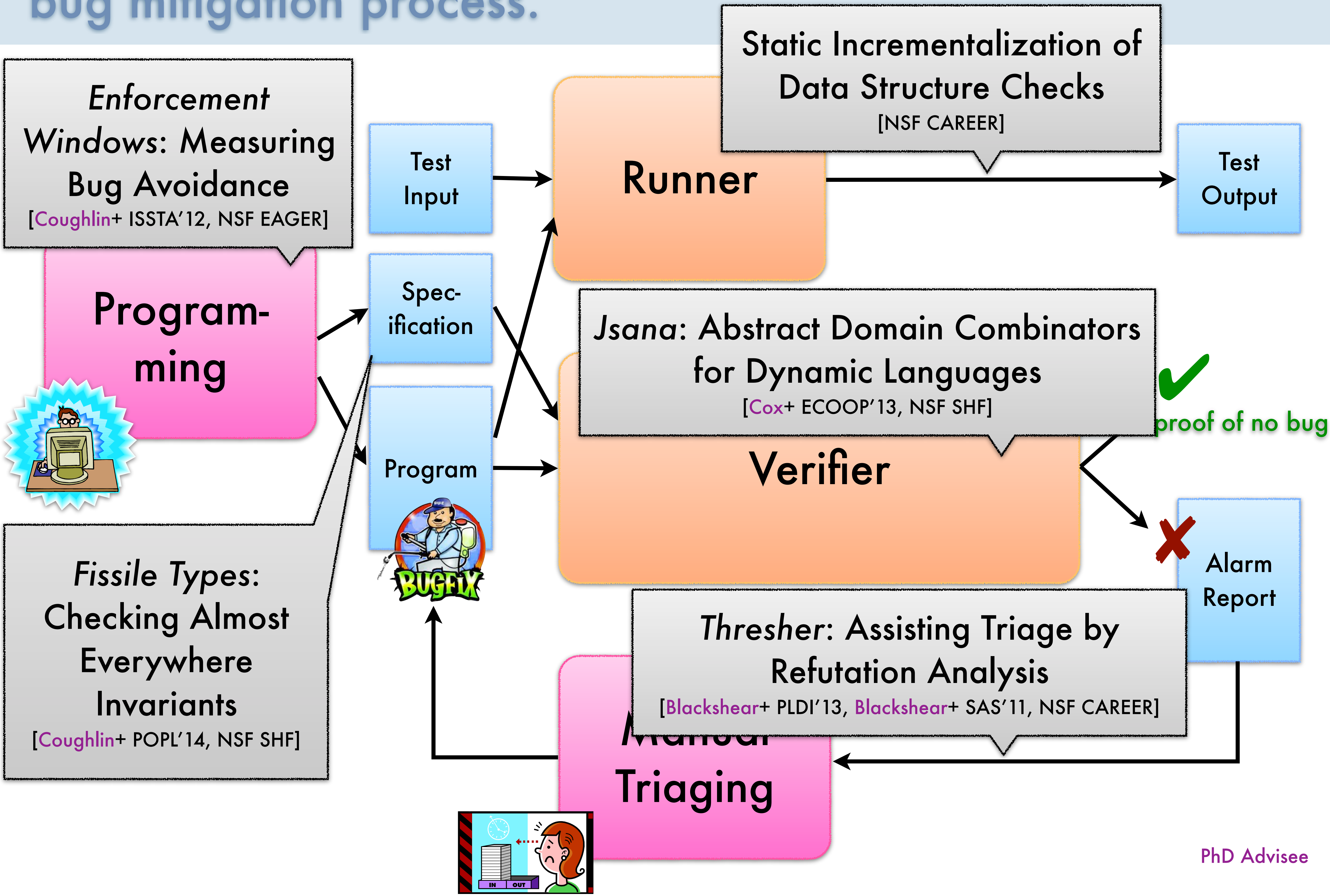
An "evil" implementation of the `Map` interface can corrupt `EMPTY`. Then, all `HashMap`s created in the future will be corrupted.

Contribution: Addressed the false alarm problem with

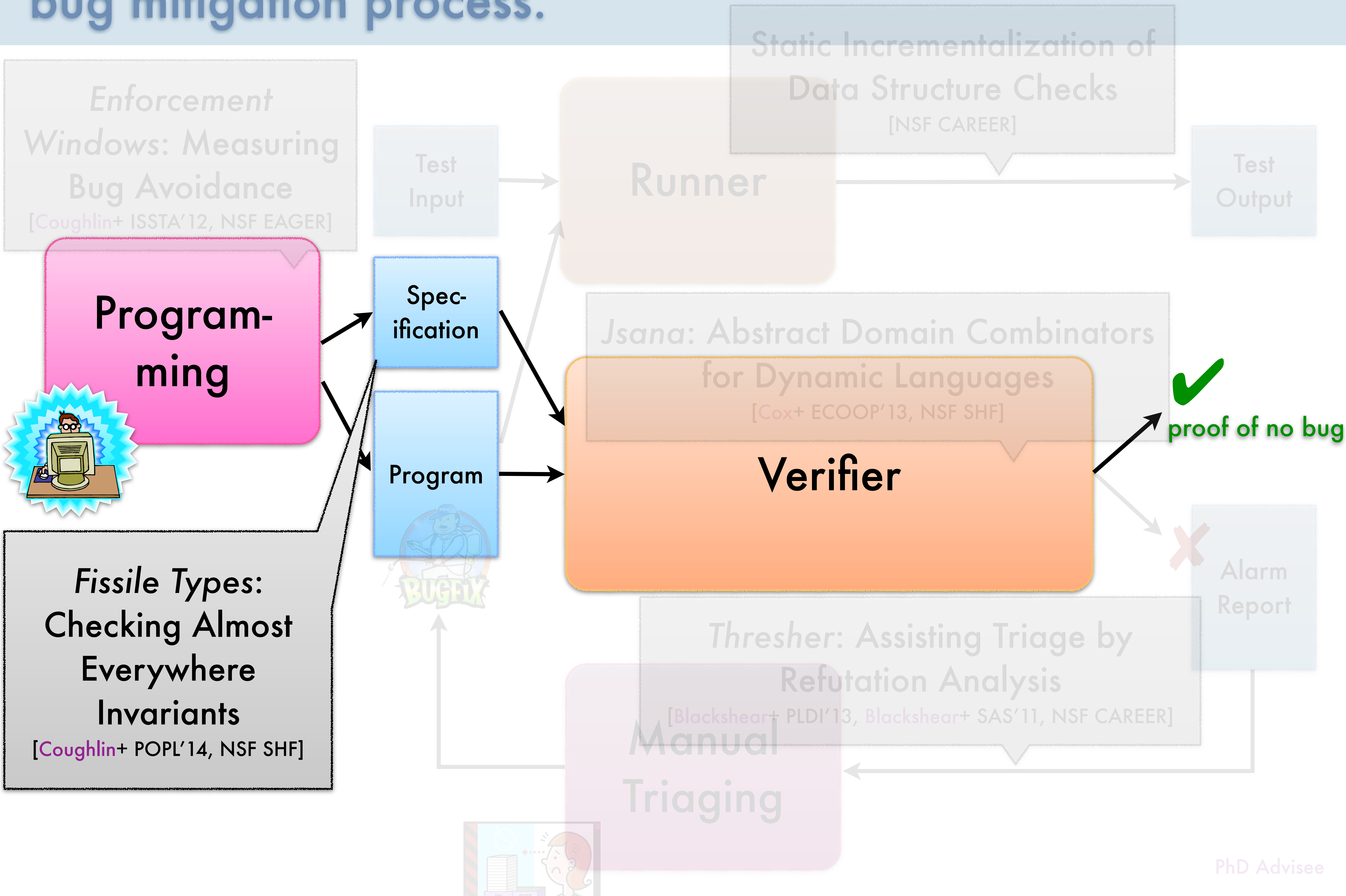
a “smart and precise filter”

a refutation analysis

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



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



# Fissile Types: Checking Reflection with Almost Everywhere Invariants

# Method Reflection and the Great Divide

# Method Reflection and the Great Divide

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

# Method Reflection and the Great Divide

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

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

# Method Reflection and the Great Divide

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

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

type system designers

"web 2.0" developers

# Method Reflection and the Great Divide

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

`object[string]()`

type system designers

"web 2.0" developers

Type system designers **worry**.

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

# Method Reflection and the Great Divide

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

`object[string]()`

**type system designers**

**"web 2.0" developers**

Type system designers **worry**.

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

"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

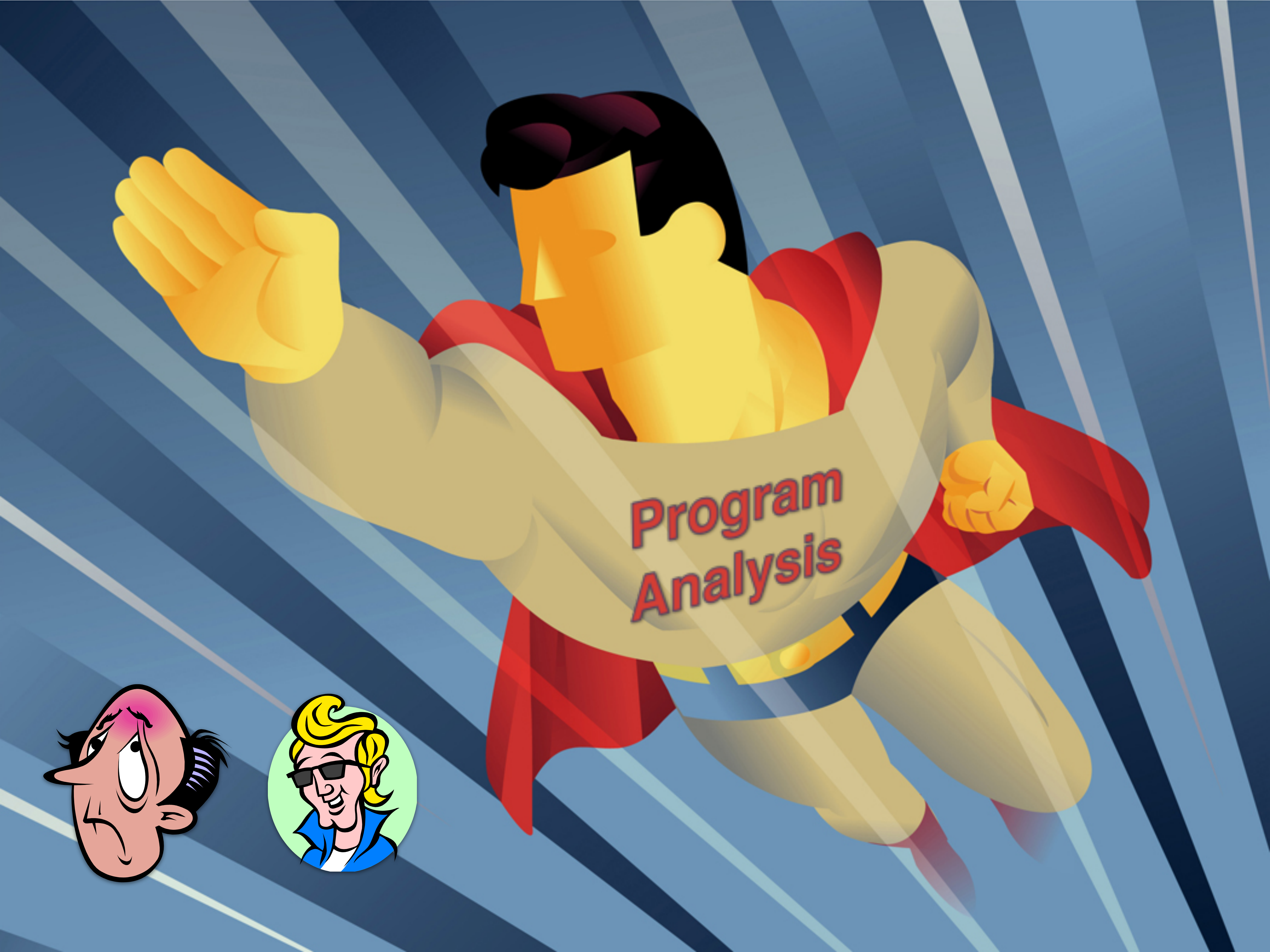
obj

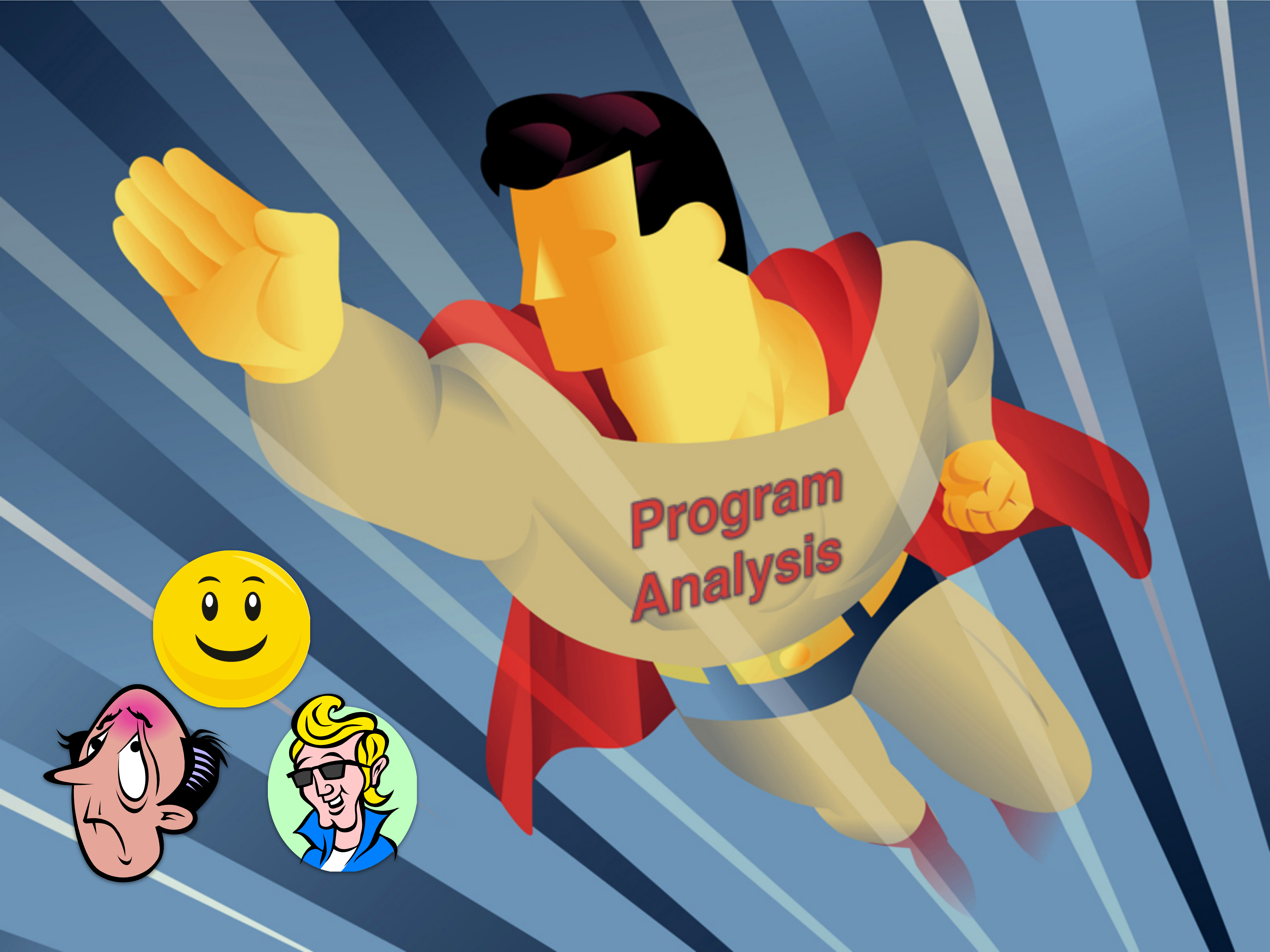
by

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

static safety.

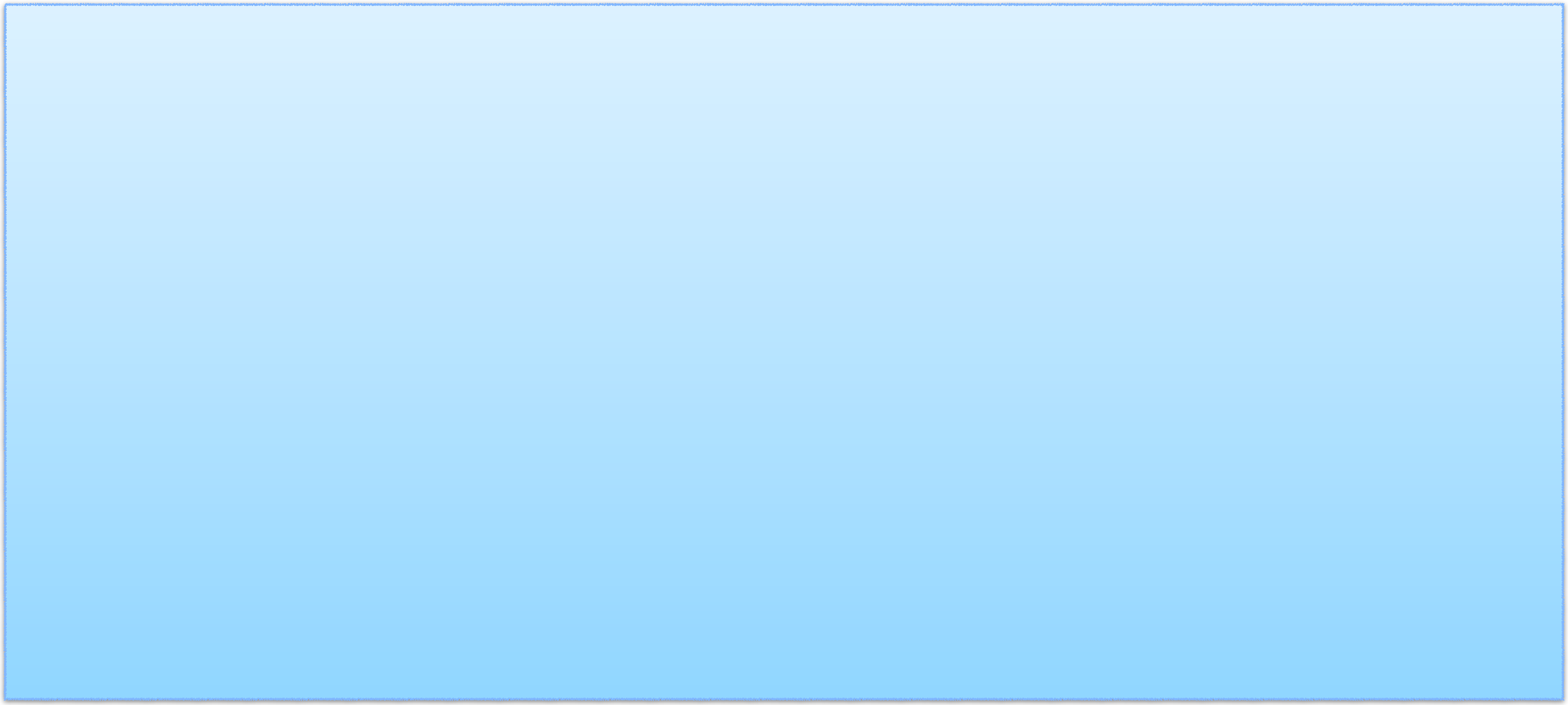
"MethodNotFound" checked at run time


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

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

Program

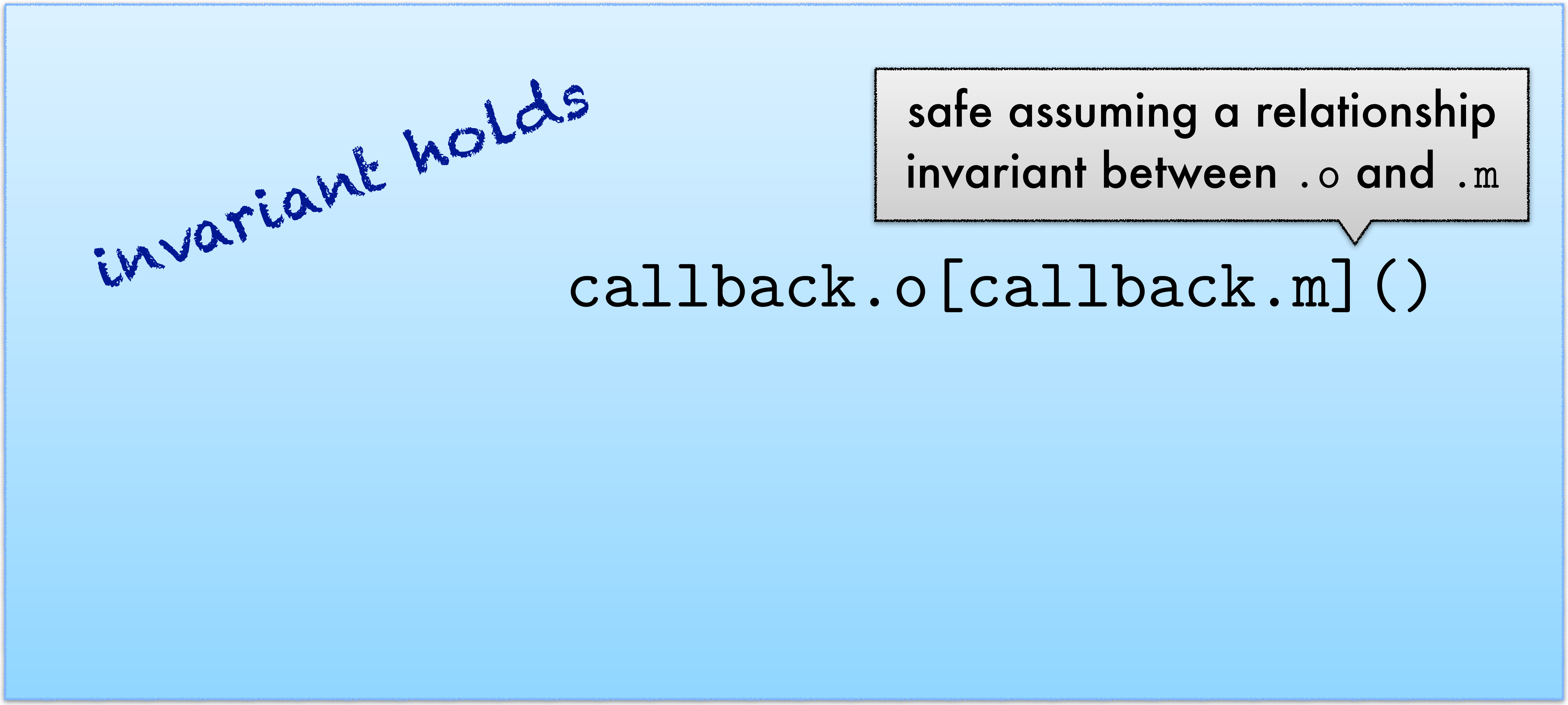
# Programs are often (1) safe, (2) not type safe, (3) but almost so

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

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

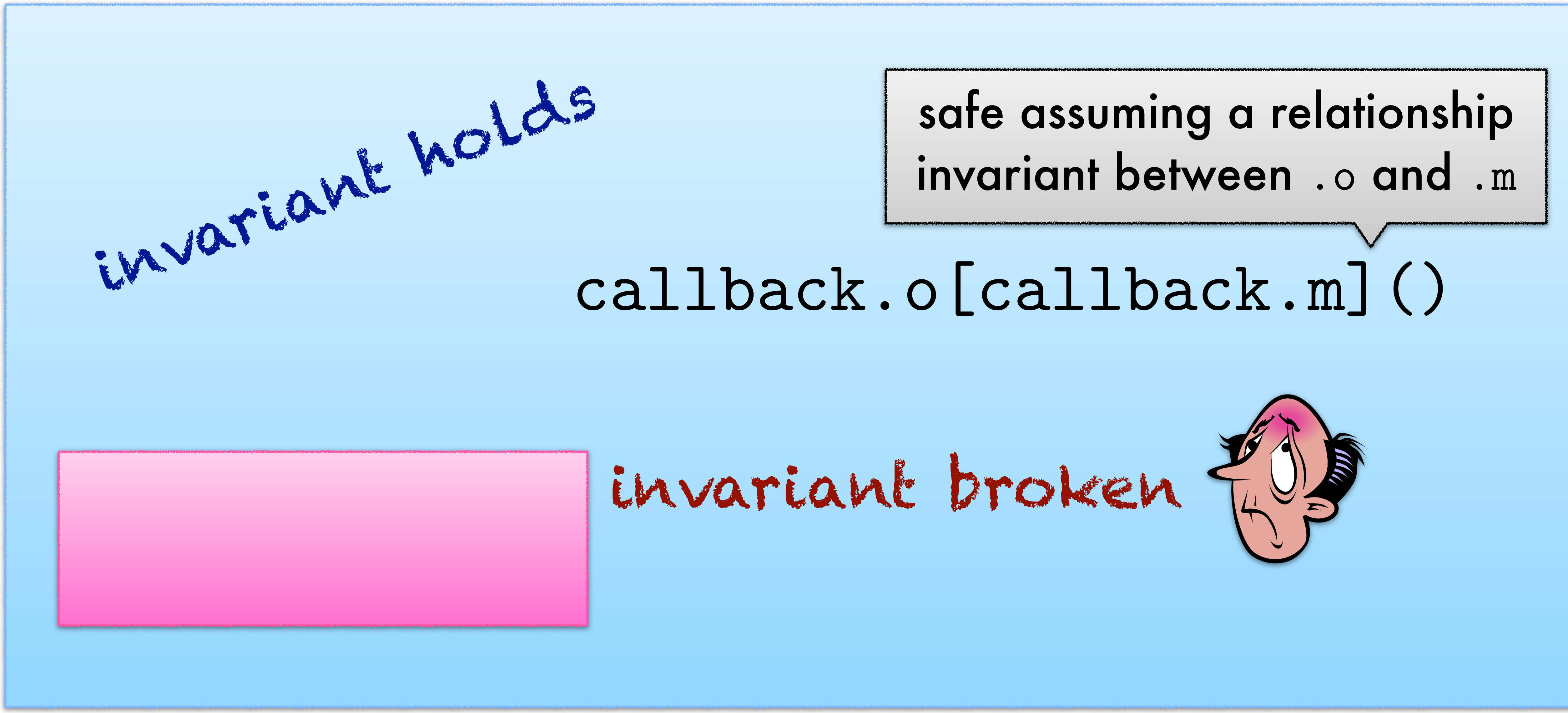
Program

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



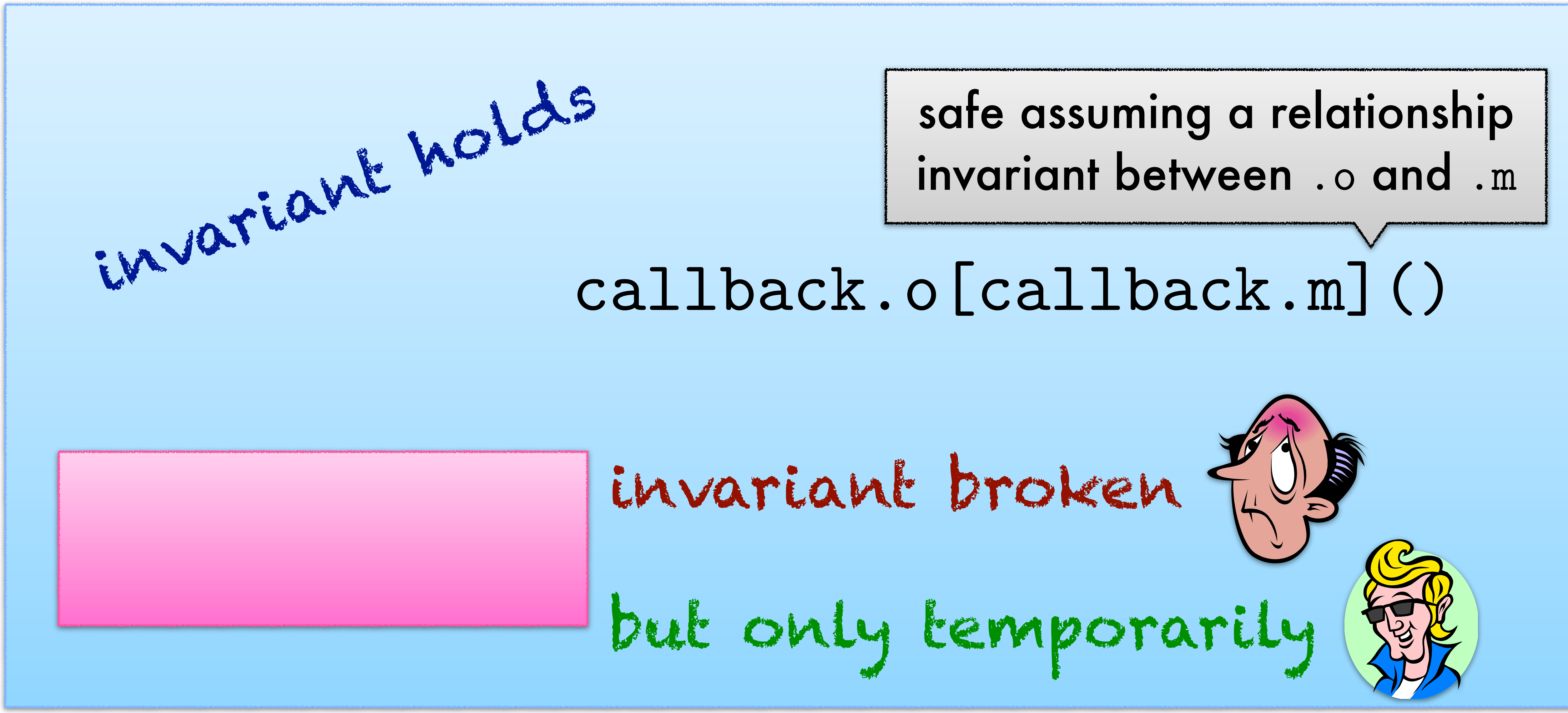

Program

# 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

invariant holds

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

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

invariant broken 

but only temporarily 

Program

**Tolerate “temporary” violation with** 

# Safe but not type safe ...

# Safe but not type safe ...

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

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

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

# Safe but not type safe ...

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

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

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

# Safe but not type safe ...

Type specifies a global relationship invariant

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

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

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

# Safe but not type safe ...

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

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

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

Type specifies a global relationship invariant

Call is safe because of the invariant

# Safe but not type safe ...

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

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

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

Type specifies a global relationship invariant

Call is safe because of the invariant

# Safe but not type safe ...

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

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

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

Type specifies a global relationship invariant

Call is safe because of the invariant

relationship invariant violated

# Safe but not type safe ...

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

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

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

Type specifies a global relationship invariant

Call is safe because of the invariant

relationship invariant violated

relationship invariant restored

# Safe but not type safe ...

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

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

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

Type specifies a global relationship invariant

Call is safe because of the invariant

relationship invariant violated

relationship invariant restored

**Tolerate "temporary" violation with** 

# Is Fissile effective at proving reflective call safety?

- Fissile analyzes Objective-C source
- 9 benchmarks (6 libraries + 3 apps)
  - 1,000 to 176,000 lines of code
  - 461,000 lines in total
- Type annotations
  - seeded with 76 `respondsTo` in system libraries
  - needed only 136 annotations in benchmarks (total)

# Is Fissile effective at proving reflective call safety?

Fissile analyzes Objective-C source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

Type annotations

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

benchmarks (total)

# Is Fissile effective at proving reflective call safety?

Fissile analyzes Objective-C source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

places requiring a check of the invariant

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

benchmarks (total)

# Is Fissile effective at proving reflective call safety?

Fissile analyzes Objective-C source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

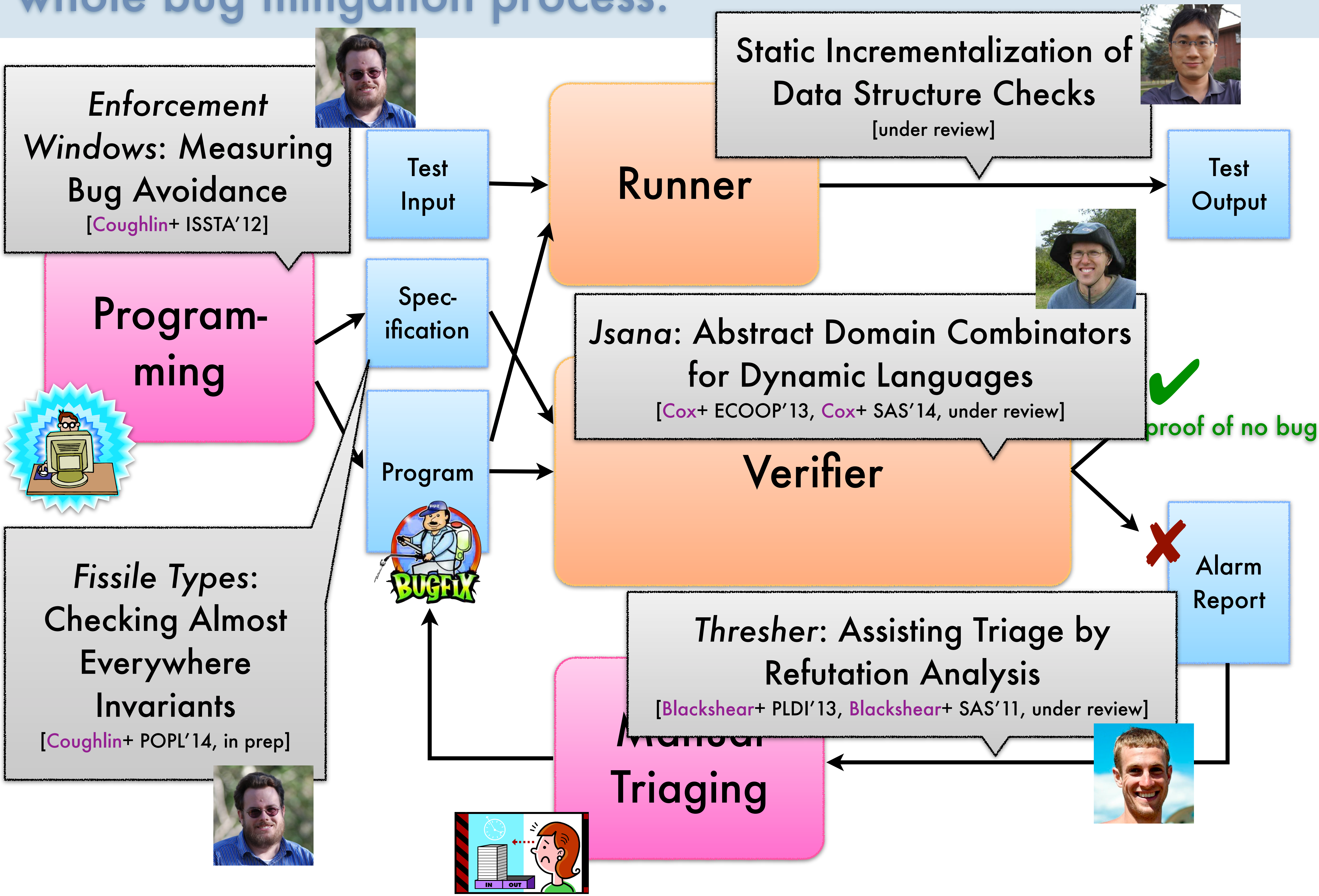
461,000 lines in total

places requiring a check of the invariant

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

Big Deal: makes IDE integration possible

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



PLV

www.cs.colorado.edu/~bec

pl.cs.colorado.edu


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