MUSE Site Visit: Planned Agenda

Welcome DARPA and AFRL Visitors!

MUSE Meeting in DLC 170

1:00pm-1:10pm Welcome

1:10pm-1:45pm Overview of the **Fixr** Project, Evan Chang

1:45pm-3:00pm Demo and Discussion: Analysis and Synthesis of App Commits, Shawn Meier/Vaibhav Singh

3:00pm-3:20pm Break

3:20pm-4:00pm Demo and Discussion: Harvesting and Storing App Commits, Mazin Hakeem/Sanghee Kim

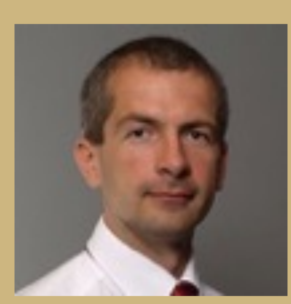
Additional Research Meetings in ECCS 1B11

4:00pm-5:15pm Graduate Students

Fixr: Mining and Understanding Bug Fixes for App-Framework Protocol Defects

Bor-Yuh Evan Chang

Ken Anderson


Pavol Cerny

Sriram Sankaranarayanan

Tom Yeh

University of Colorado Boulder

MUSE Site Visit
February 25, 2015

A bug that manifests spectacularly ...

A bug that manifests spectacularly ...

A bug that manifests spectacularly ...

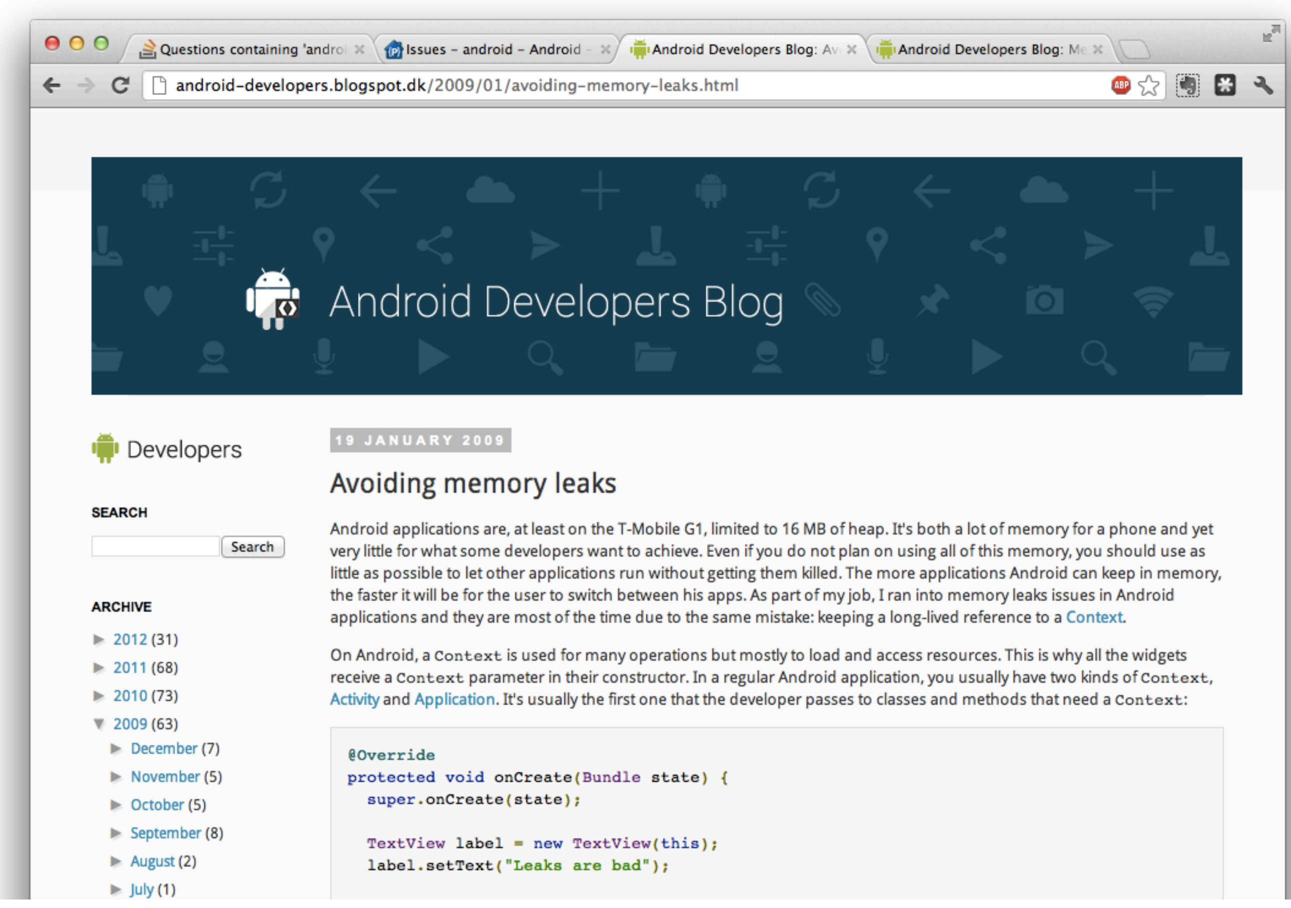
A bug that manifests spectacularly ...

A bug that manifests spectacularly ...

caused by an app-created memory leak

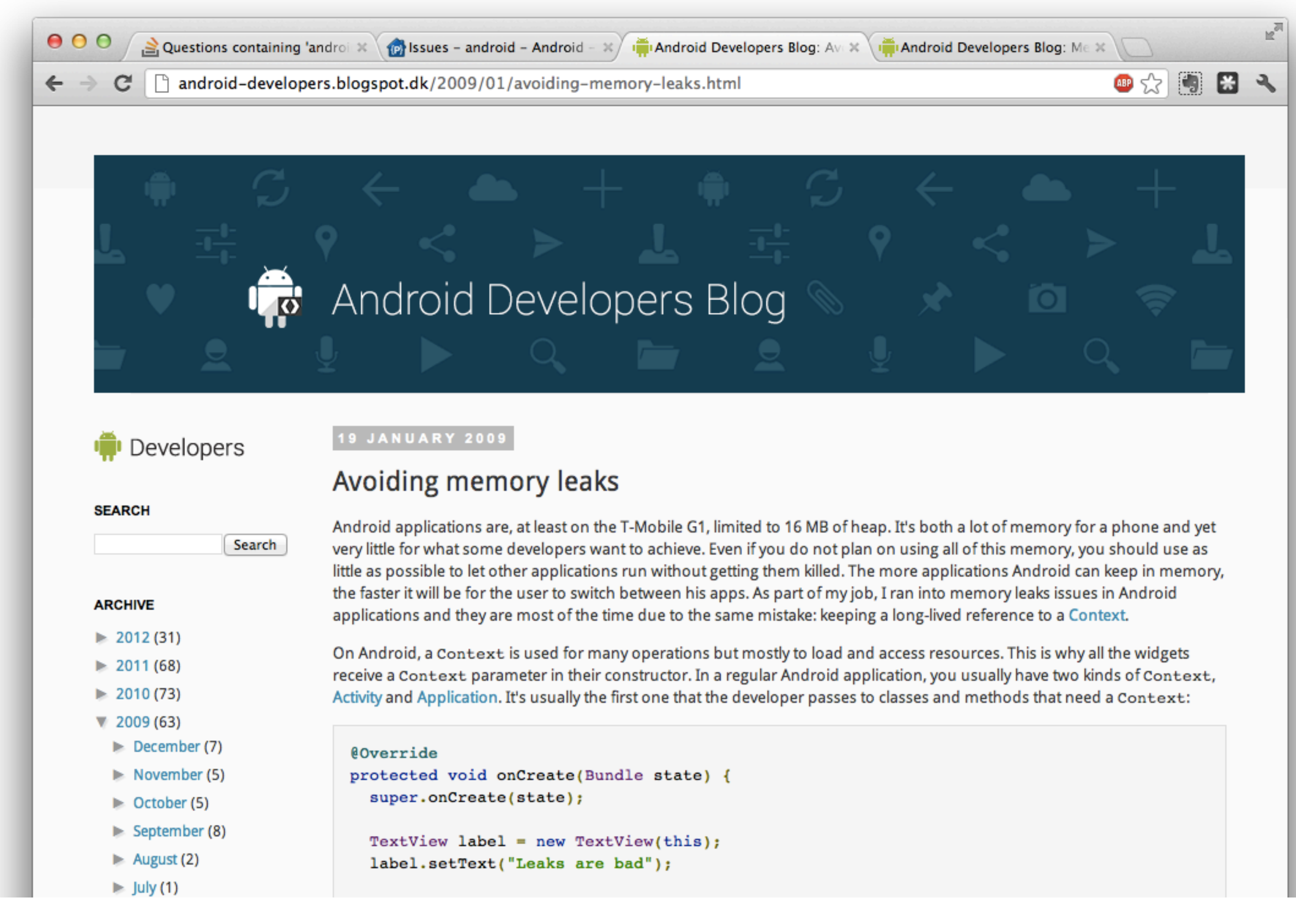
Ask framework devs ...

Ask framework devs …



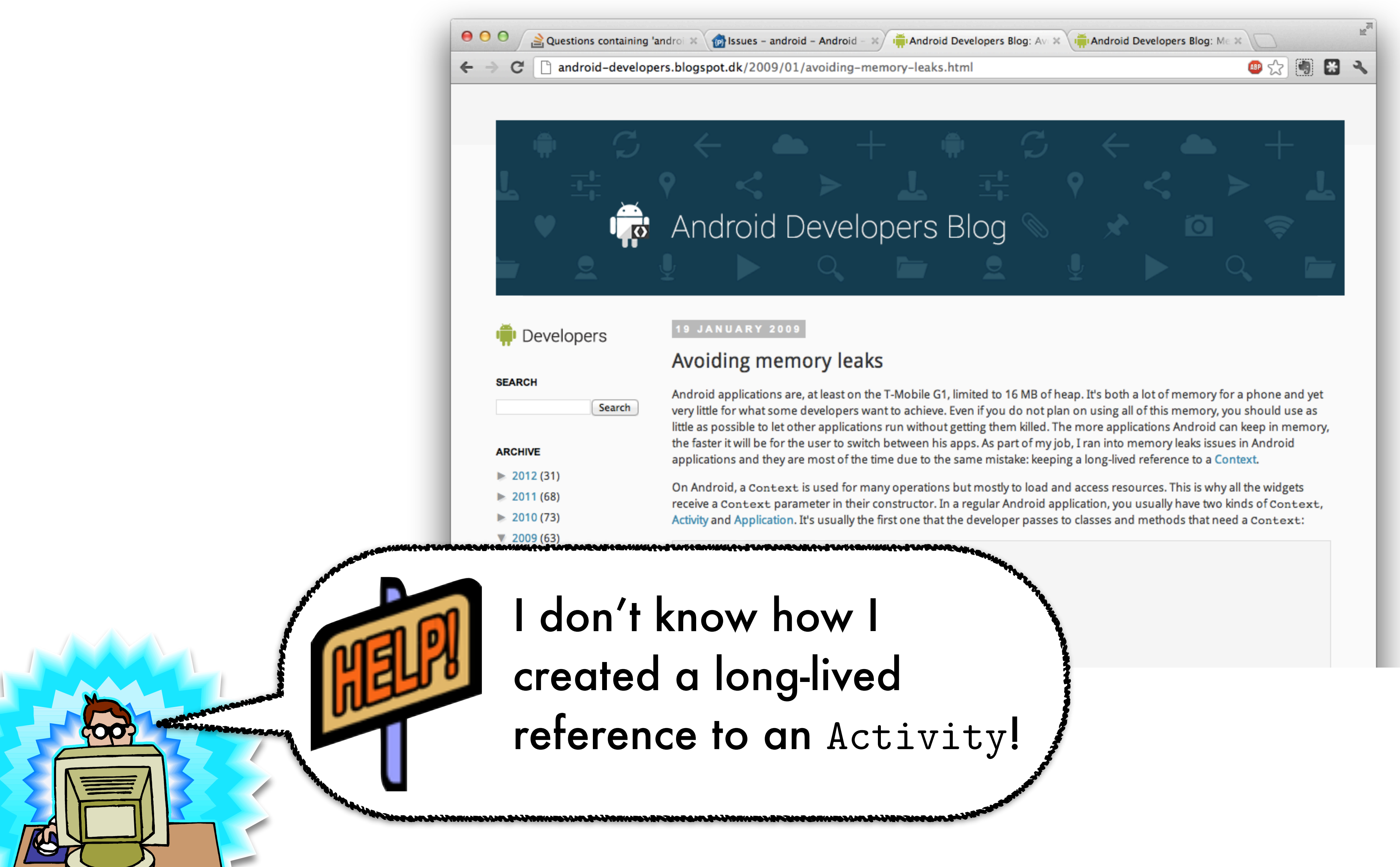
Ask framework devs …

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



Ask framework devs ...

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



Ask framework devs …

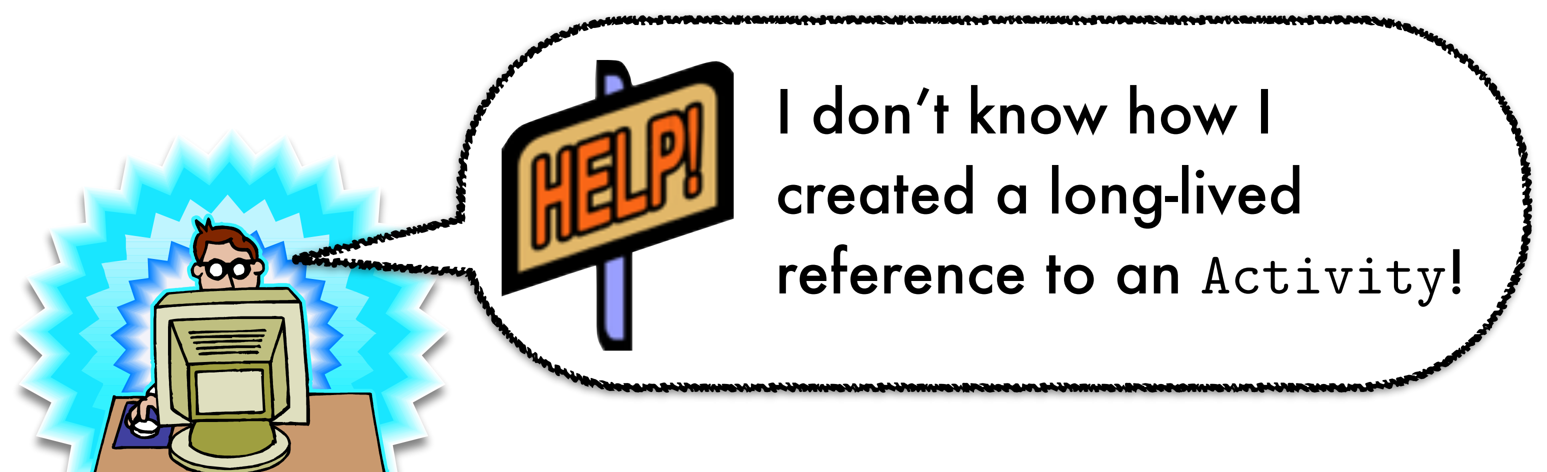
"Do not keep long-lived references to a context-activity"

Ask framework devs ...

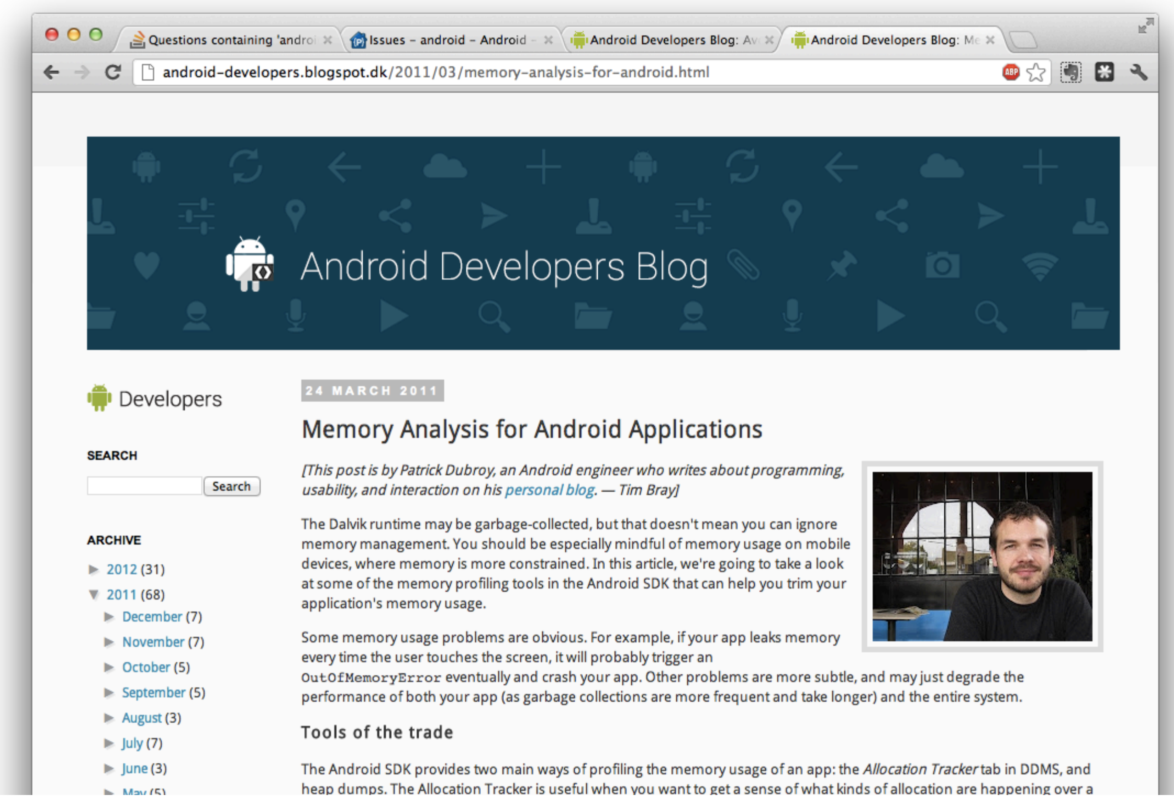
"Do not keep long-lived references to a context-activity"

Imagining a post-MUSE scenario ...

for



Elsewhere, following the state of practice for debugging leaks …



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)

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

Elsewhere, following the state of practice for debugging leaks …

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)

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

1. Run the app

Elsewhere, following the state of practice for debugging leaks ...

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)

24 MARCH 2011

Memory Analysis for Android Applications

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

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

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

Tools of the trade

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

Dalvik Debug Monitor

Info | Threads | VM Heap | Allocation Tracker | Sysinfo | Emulator Control | Event Log

Heap updates will happen after every GC for this client

ID	Heap Size	Allocated	Free	% Used	# Objects
1	8.570 MB	8.452 MB	12[illegible].320 KB	98.62%	59,281

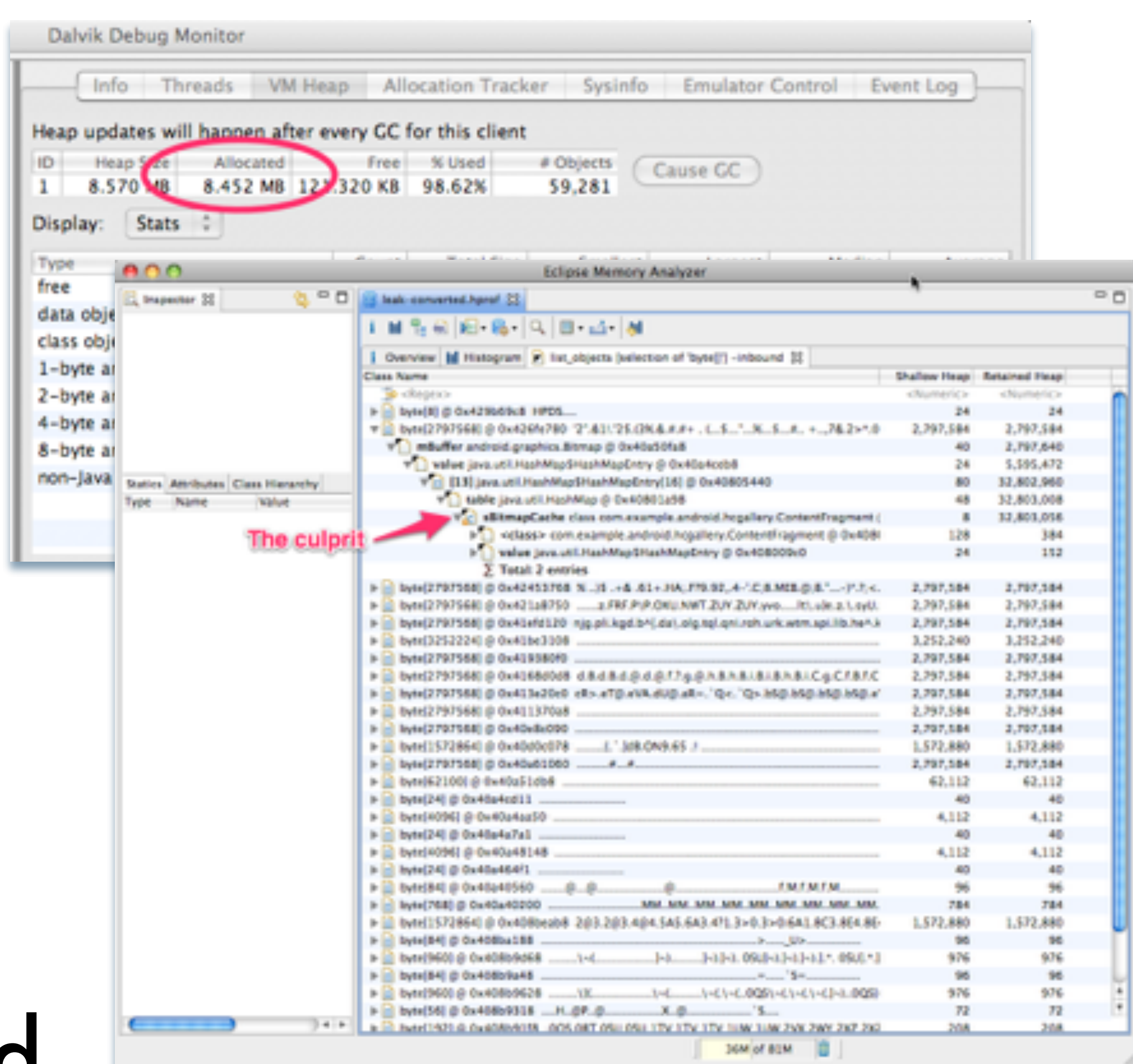
Cause GC

Display: Stats

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

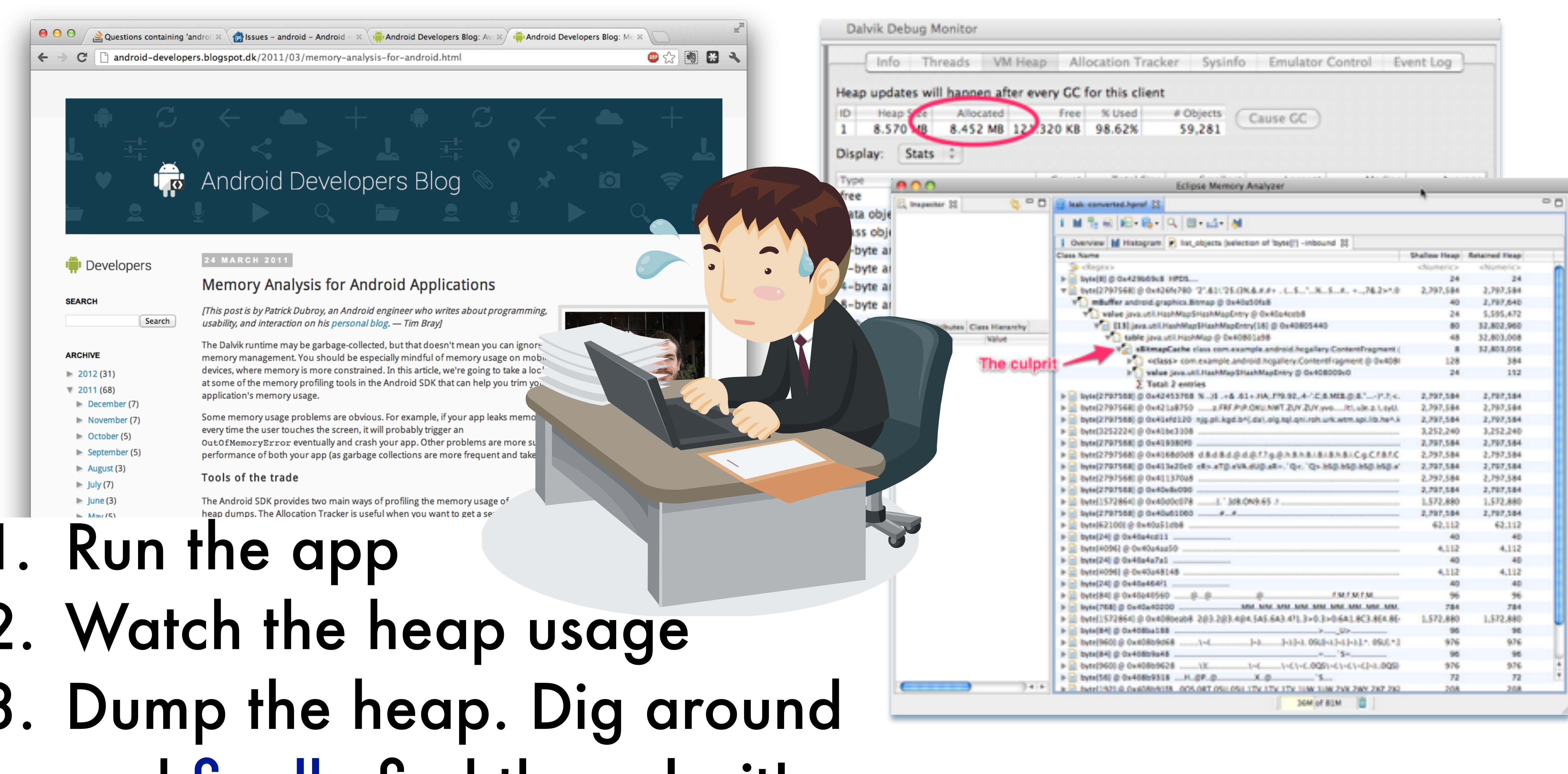
1. Run the app
2. Watch the heap usage

Elsewhere, following the state of practice for debugging leaks ...



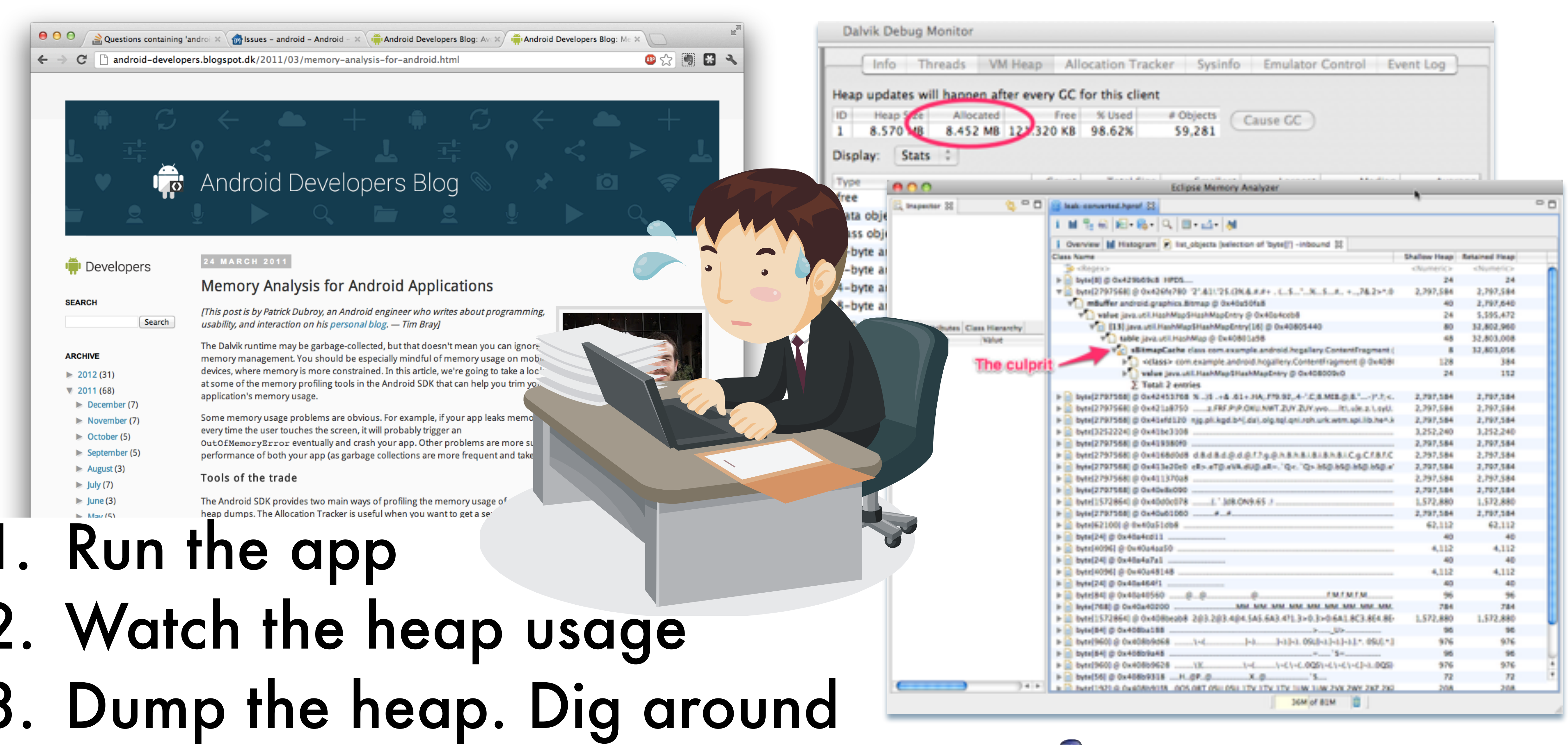
1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and *finally* find the culprit!

Elsewhere, following the state of practice for debugging leaks ...



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

Elsewhere, following the state of practice for debugging leaks ...



1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and finally find the culprit!
4. Commit a bugfix

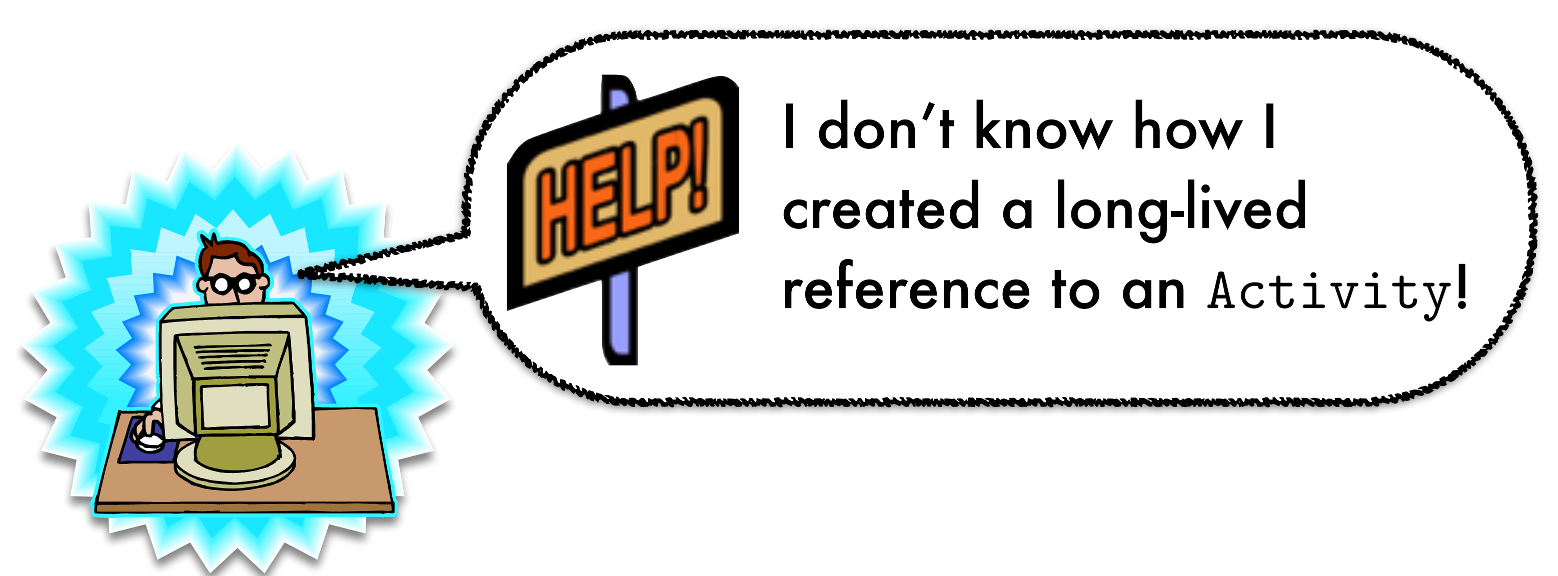
GitHub BUGFIX

Elsewhere, following the state of practice for debugging leaks ...

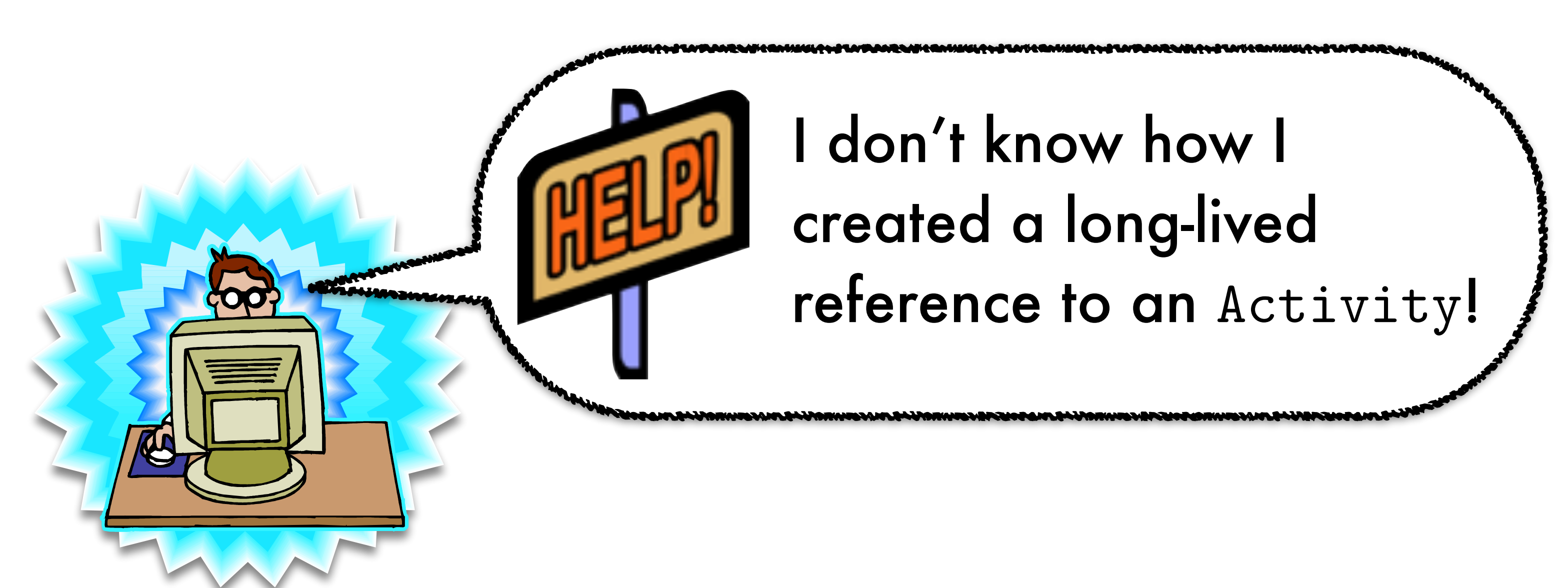
1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and finally find the culprit!
4. Commit a bugfix
5. Bugfix is picked up by **Fixr**

GitHub

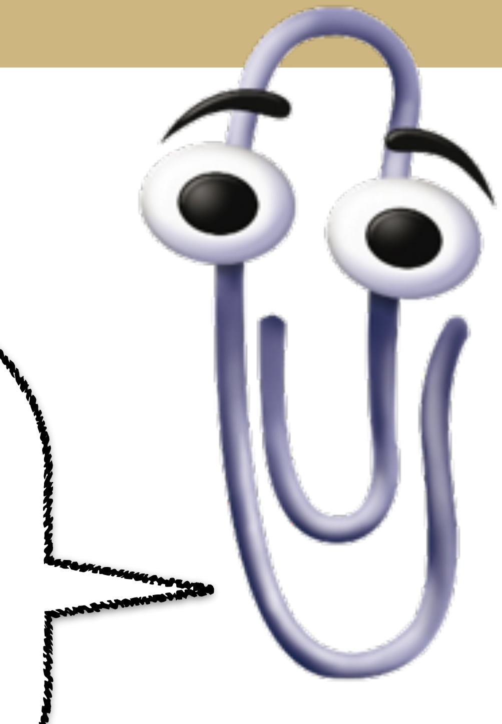
A **Fixr**-enabled IDE responds …



A **Fixr**-enabled IDE responds …

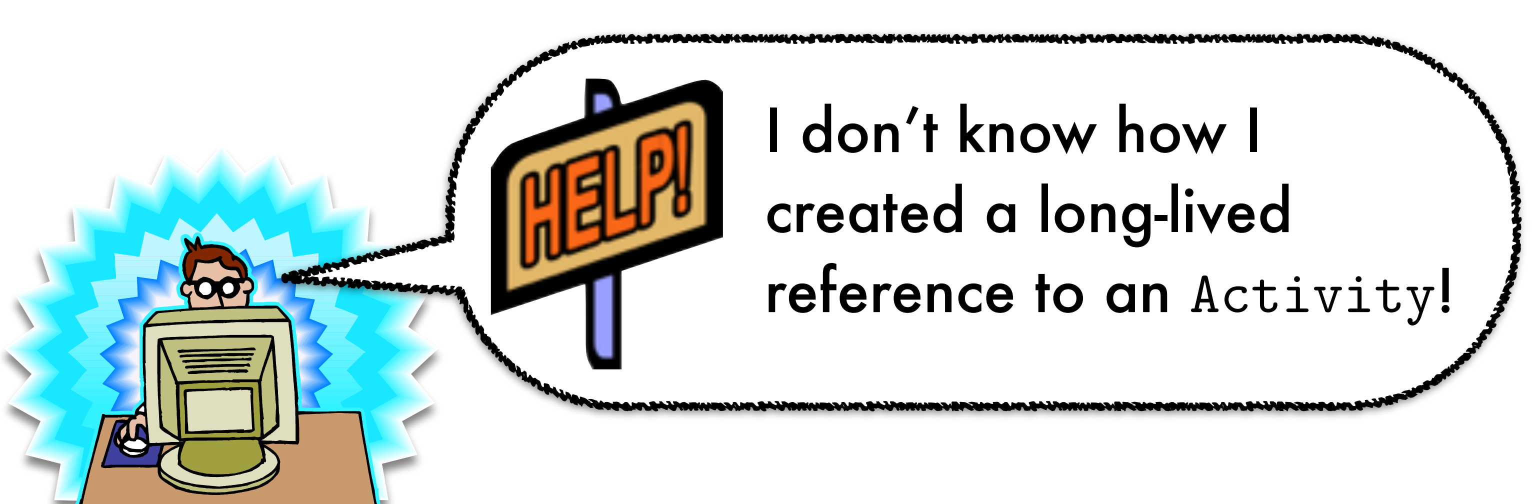


A **Fixr**-enabled IDE responds …



It looks like you've created a memory leak like and

100,000 others. Would you like to apply ?

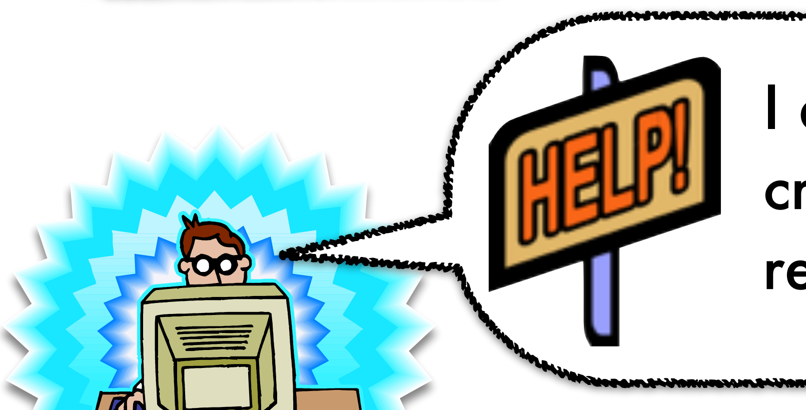


A **Fixr**-enabled IDE responds …

It looks like you've created a memory leak like and 100,000 others. Would you like to apply

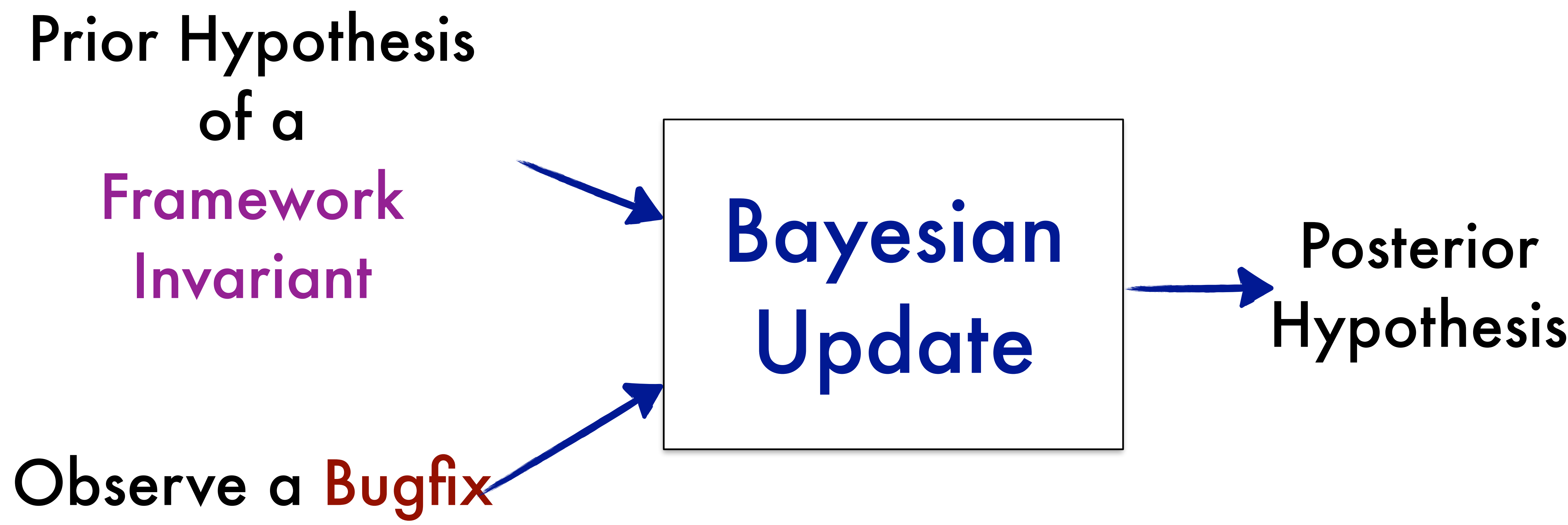
?

the bugfix is "transferred"

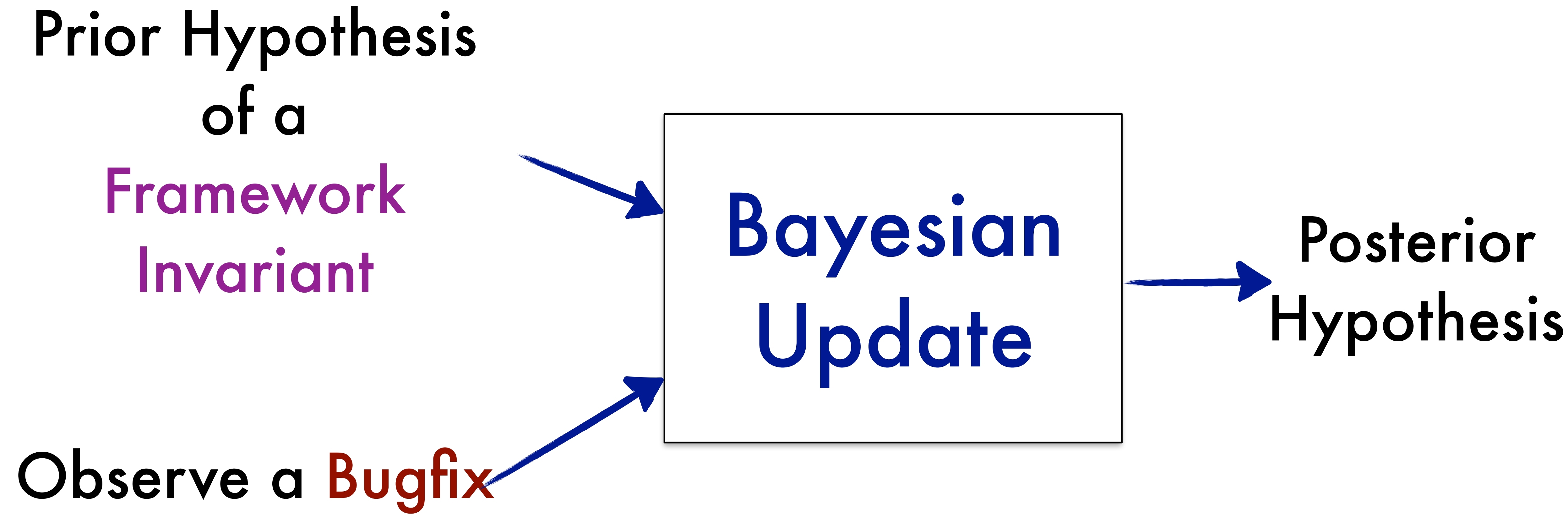


I don't know how I created a long-lived reference to an `Activity`!

Summary: Mine framework specifications with bugfixes



Summary: Mine framework specifications with bugfixes



The **Fixr** Loop:
Create as many observations as possible

Simple motivating example: A well-understood Android bug

Simple motivating example: A well-understood Android bug

a common misuse of the framework

Bug
(on Android <4)

aView.setTag(..., anObject)

Bug
(on Android <4)

aView.setTag(..., anObject)

if anObject can reach aView

Bug
(on Android <4)

aView.setTag(..., anObject)

if anObject can reach aView

Framework Invariant

```
class View {
  static WeakHashMap<View, SparseArray<Object>> sTags;
  Object mTag;
}
```

aView.setTag(..., anObject)

if anObject can reach aView

Framework Invariant

```
class View {
  static WeakHashMap<View, SparseArray<Object>> sTags;
  Object mTag;
}
```

because of an unspecified class invariant: sTags' values (:Object) must not reach their keys (:View)

Bug
(on Android <4)

aView.setTag(..., anObject)

if anObject can reach aView

Framework Invariant

```
class View {
  static WeakHashMap<View, SparseArray<Object>> sTags;
  Object mTag;
}
```

because of an unspecified class invariant: sTags' values (:Object) must not reach their keys (:View)

A Fix

Bug
(on Android <4)

aView.setTag(..., anObject)

if anObject can reach aView

Framework Invariant

```
class View {
  static WeakHashMap<View, SparseArray<Object>> sTags;
  Object mTag;
}
```

because of an unspecified class invariant: sTags' values (:Object) must not reach their keys (:View)

A Fix

~~aView.setTag(..., anObject)~~
aView.setTag(...anObject...)

uses mTag instead

Bug
(on Android <4)

aView.setTag(..., anObject)

if anObject can reach aView

Framework Invariant

```
class View {
  static WeakHashMap<View, SparseArray<Object>> sTags;
  Object mTag;
}
```

invariant

because of an unspecified class invariant: sTags' values (:Object) must not reach their keys (:View)

Goal: Produce this repair specification: *bug pre*, *framework invariant*, *fix suggestion*

Challenges

Given a bugfix commit, how do we summarize and generalize the fix (to be able “transfer”)?

How do we find bugfix commits?

Challenges

Given a bugfix commit, how do we summarize and generalize the fix (to be able "transfer")?

a specification of the View.setTag repair

How do we find bugfix commits?

Challenges

Given a bugfix commit, how do we summarize and generalize the fix (to be able "transfer")?

a specification of the View.setTag repair

How do we find bugfix commits?

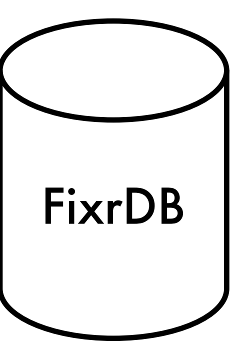
an instance of a View.setTag fix

Fixr

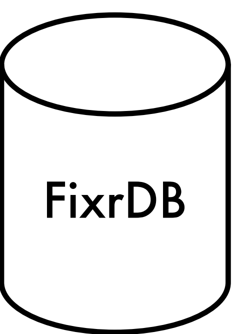
Components and Workflows

Workflow 1a: Harvesting bugfix commits

Workflow 1a: Harvesting bugfix commits

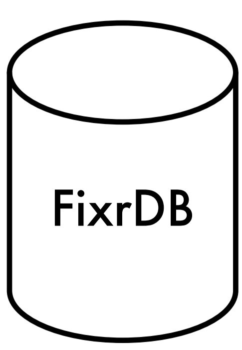


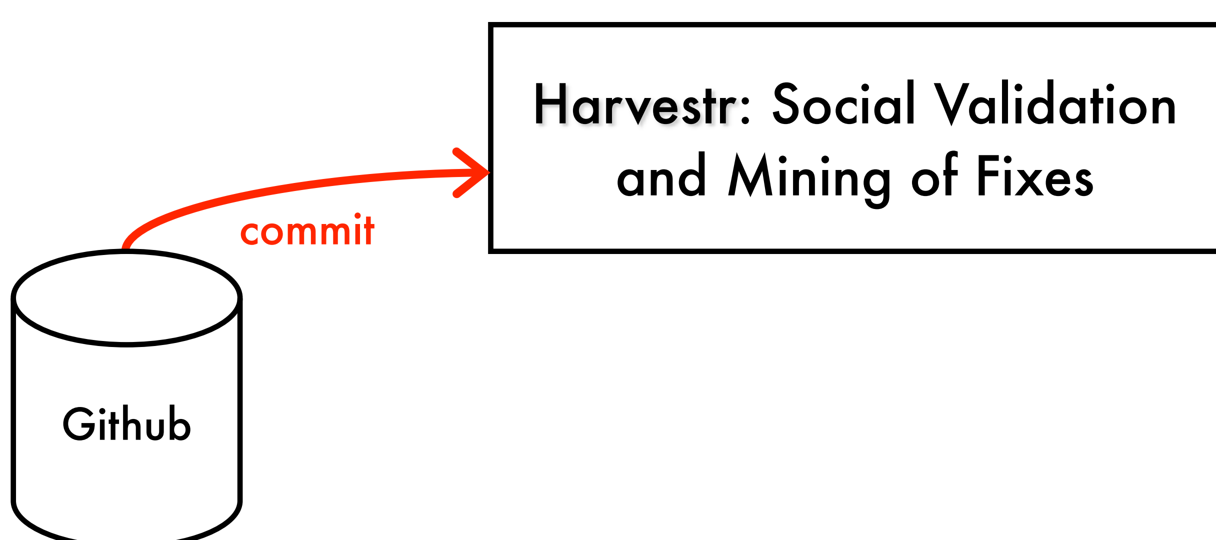
Workflow 1a: Harvesting bugfix commits



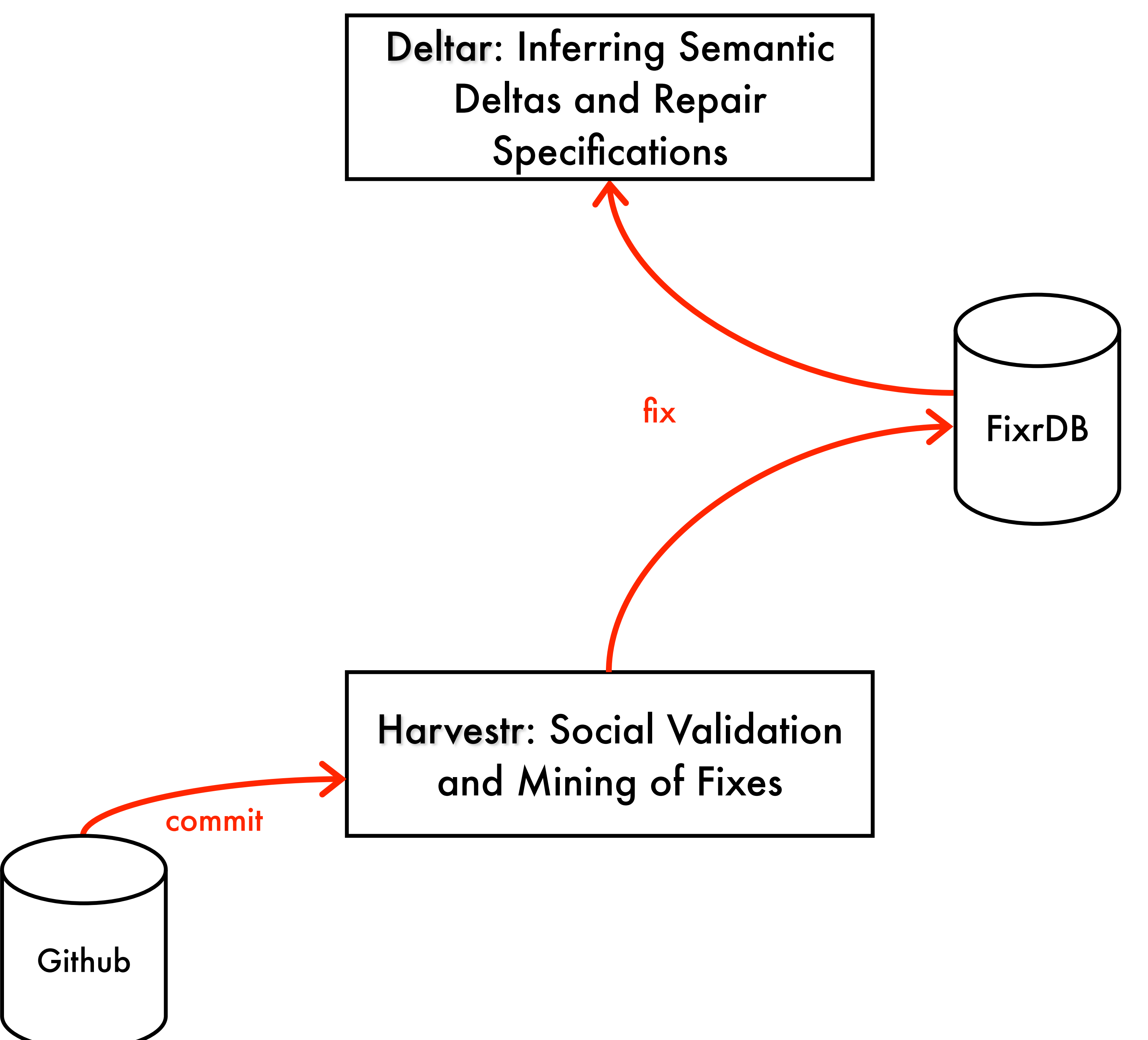
Harvestr: Social Validation and Mining of Fixes

Workflow 1a: Harvesting bugfix commits

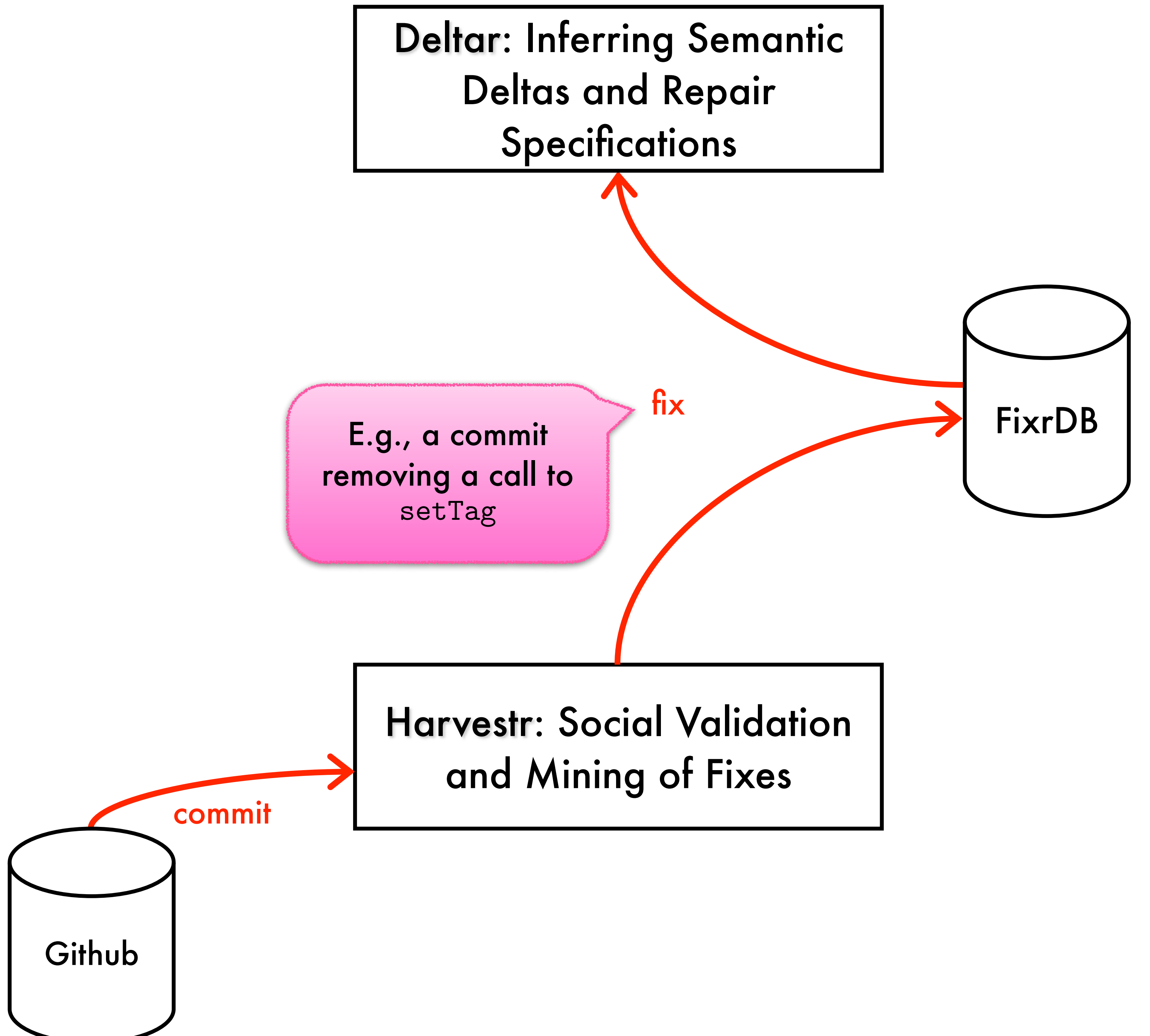




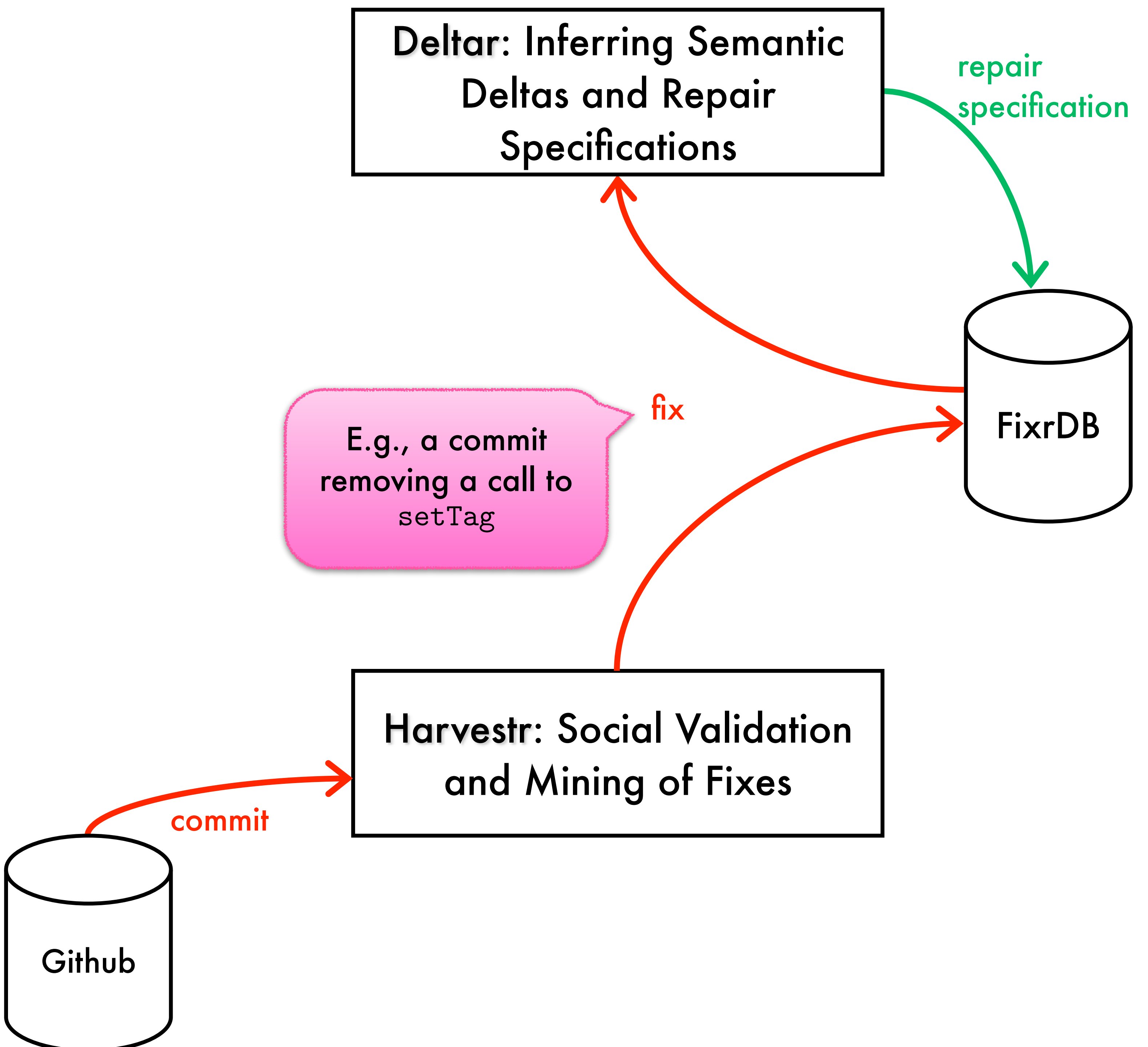
Workflow 1a: Harvesting bugfix commits



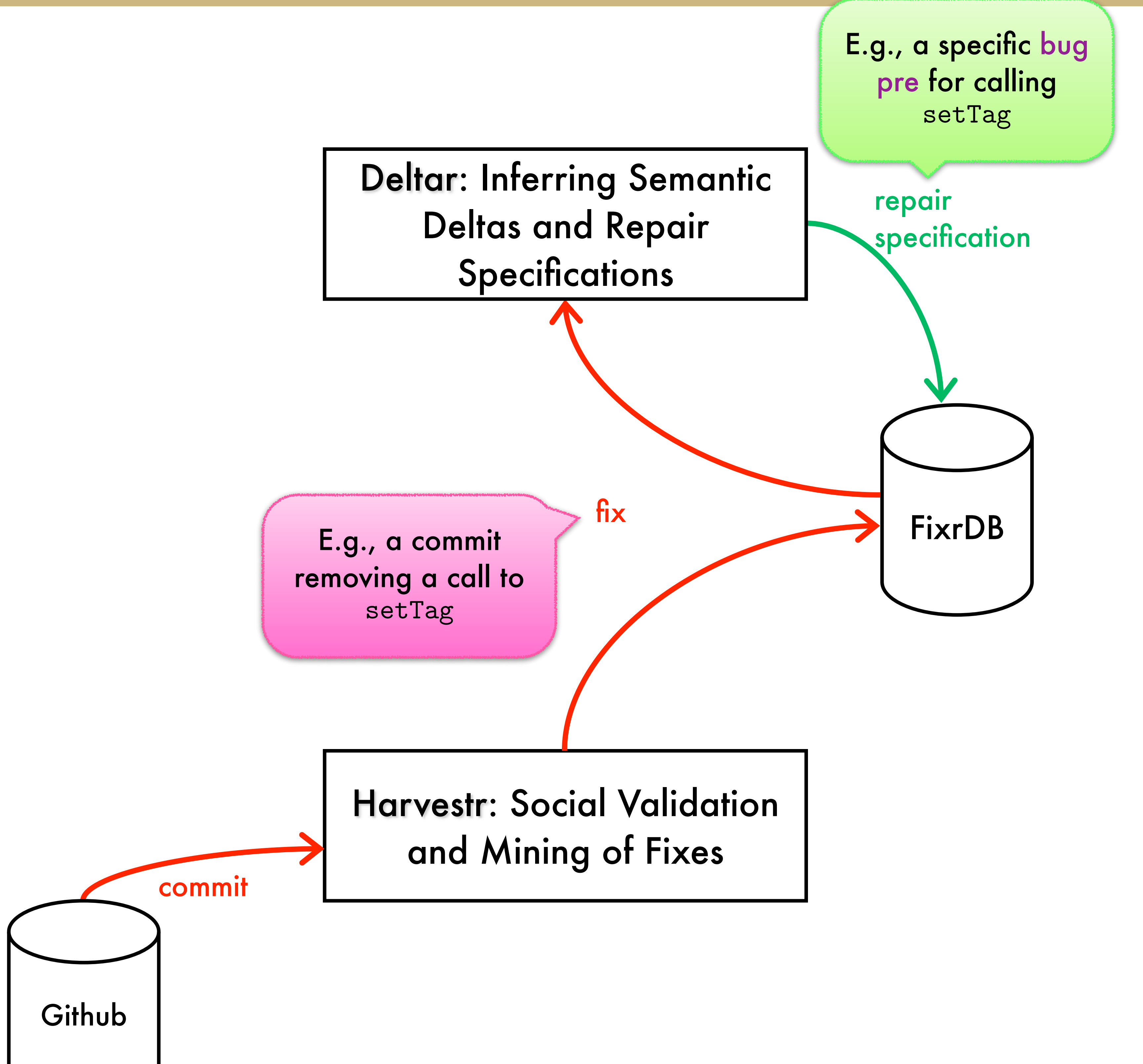
Workflow 1a: Harvesting bugfix commits



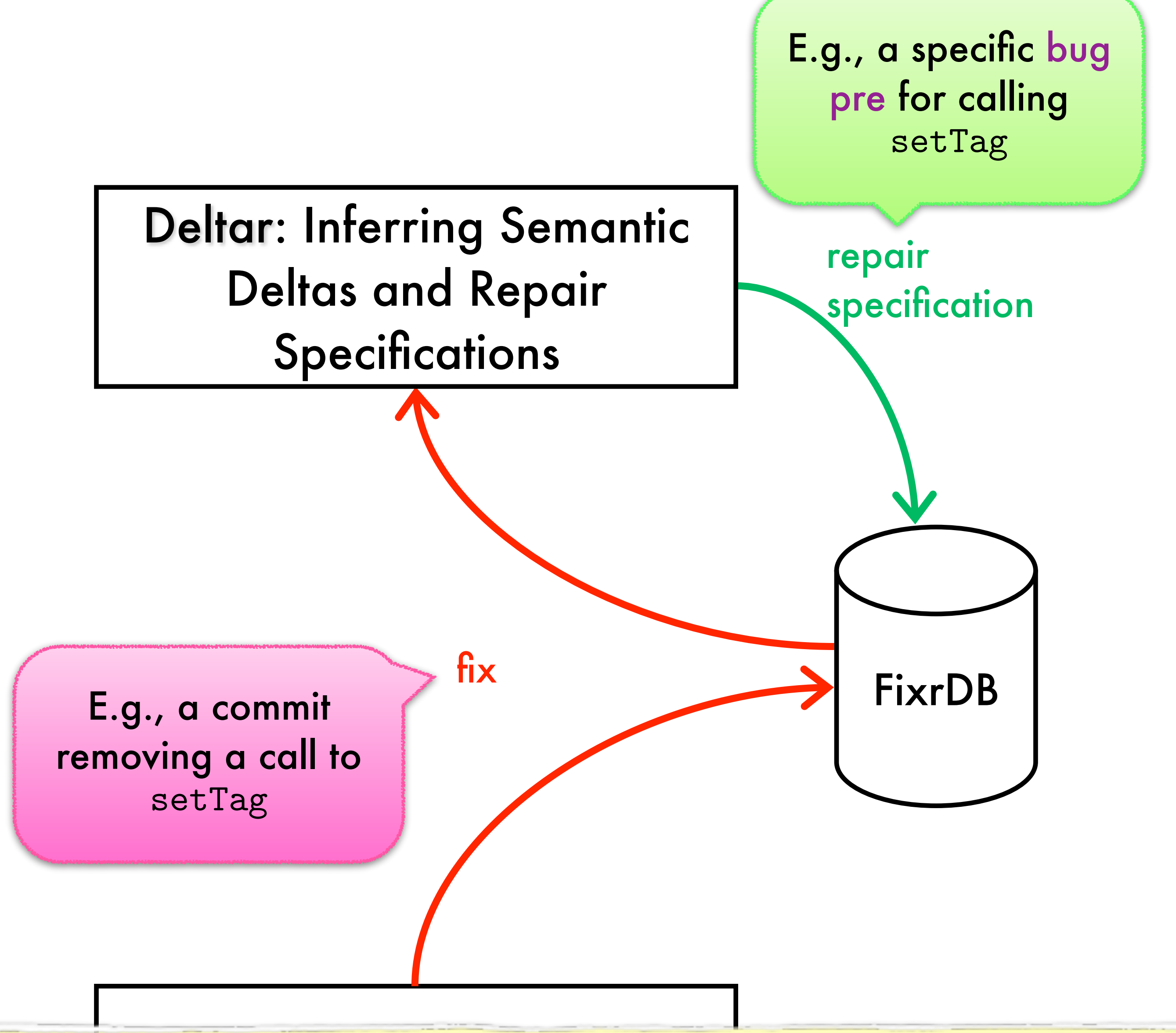
Workflow 1a: Harvesting bugfix commits



Workflow 1a: Harvesting bugfix commits

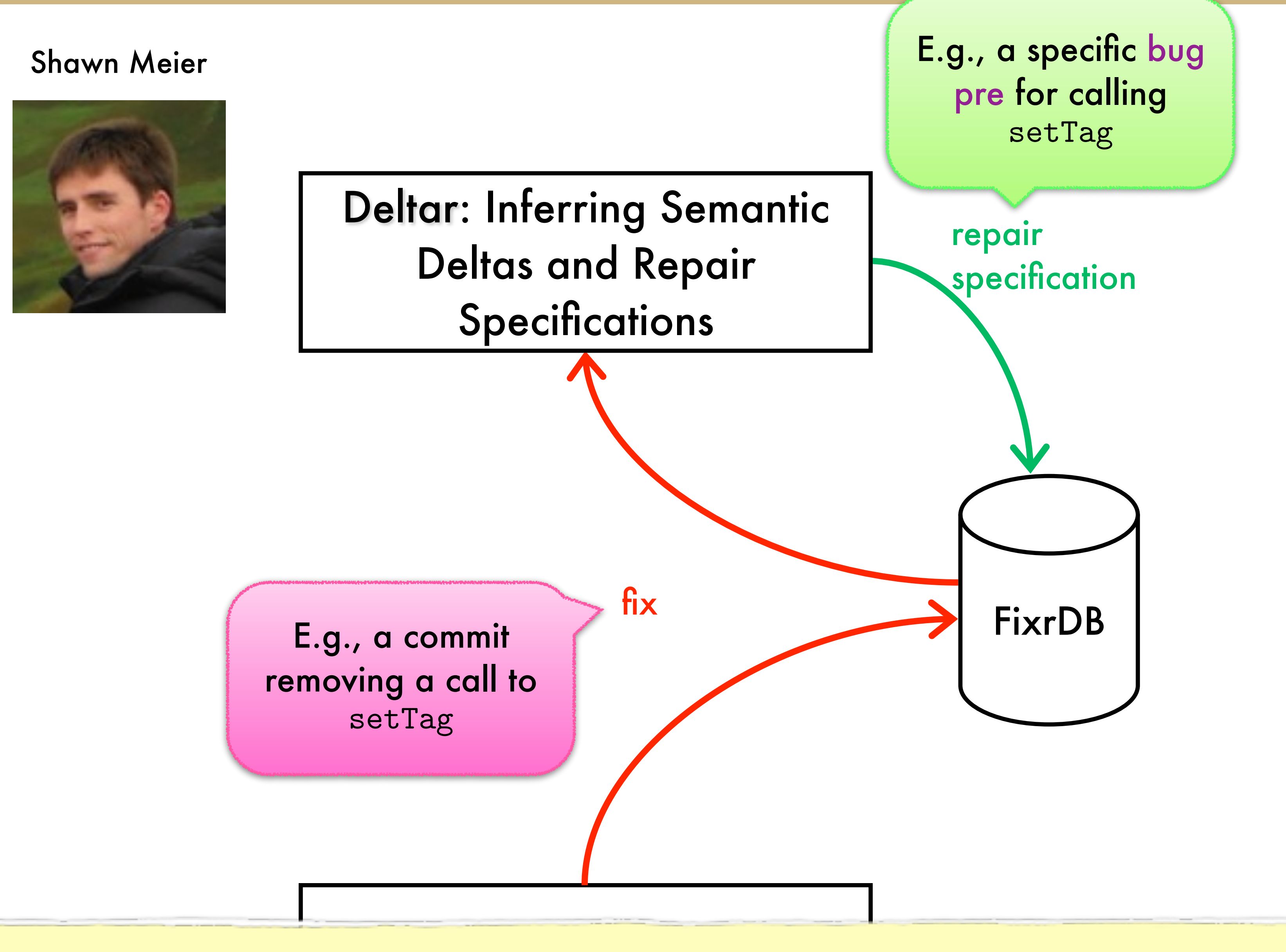


Workflow 1a: Harvesting bugfix commits



Component: Deltar *maps* bugfixes to candidate repair specifications (including bug pre)

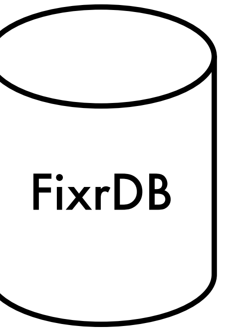
Workflow 1a: Harvesting bugfix commits



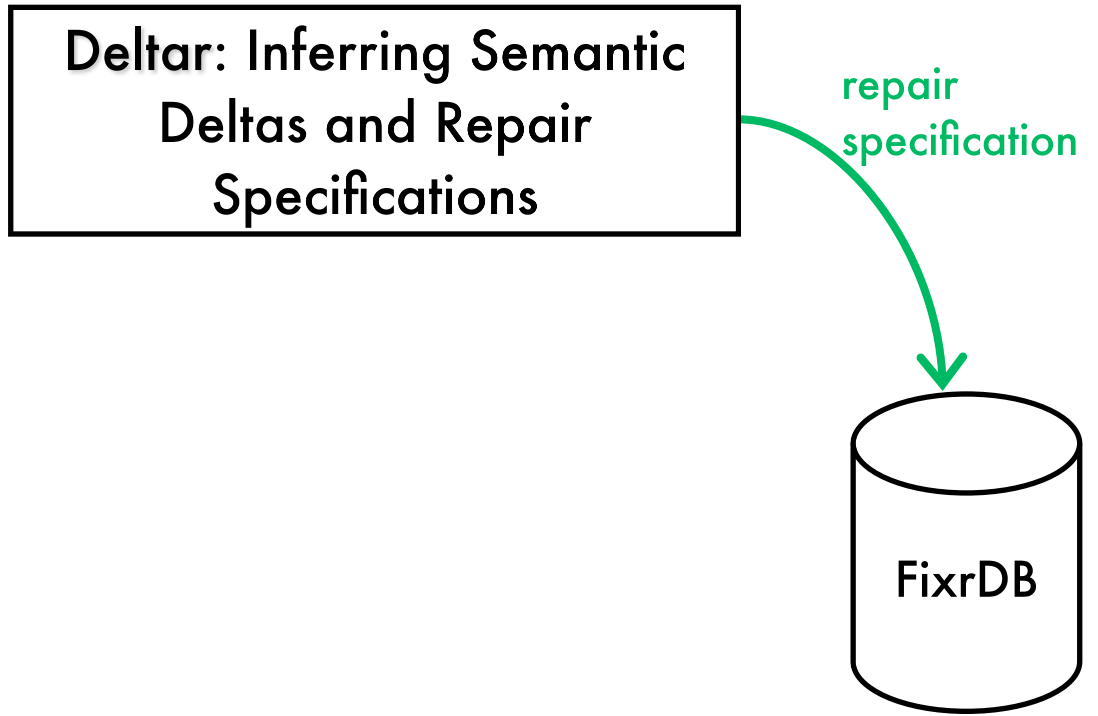
Component: Deltar *maps* bugfixes to candidate repair specifications (including bug pre)

Workflow 1b: Aggregating repairs

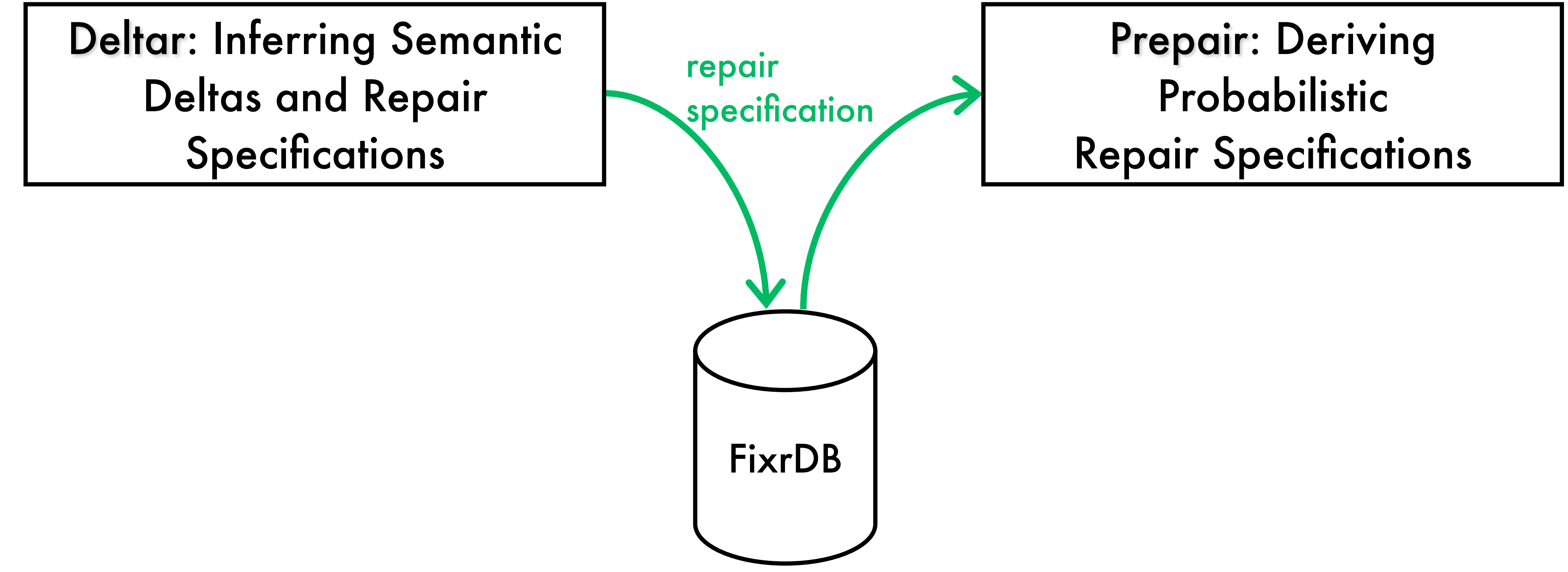
Deltar: Inferring Semantic Deltas and Repair Specifications



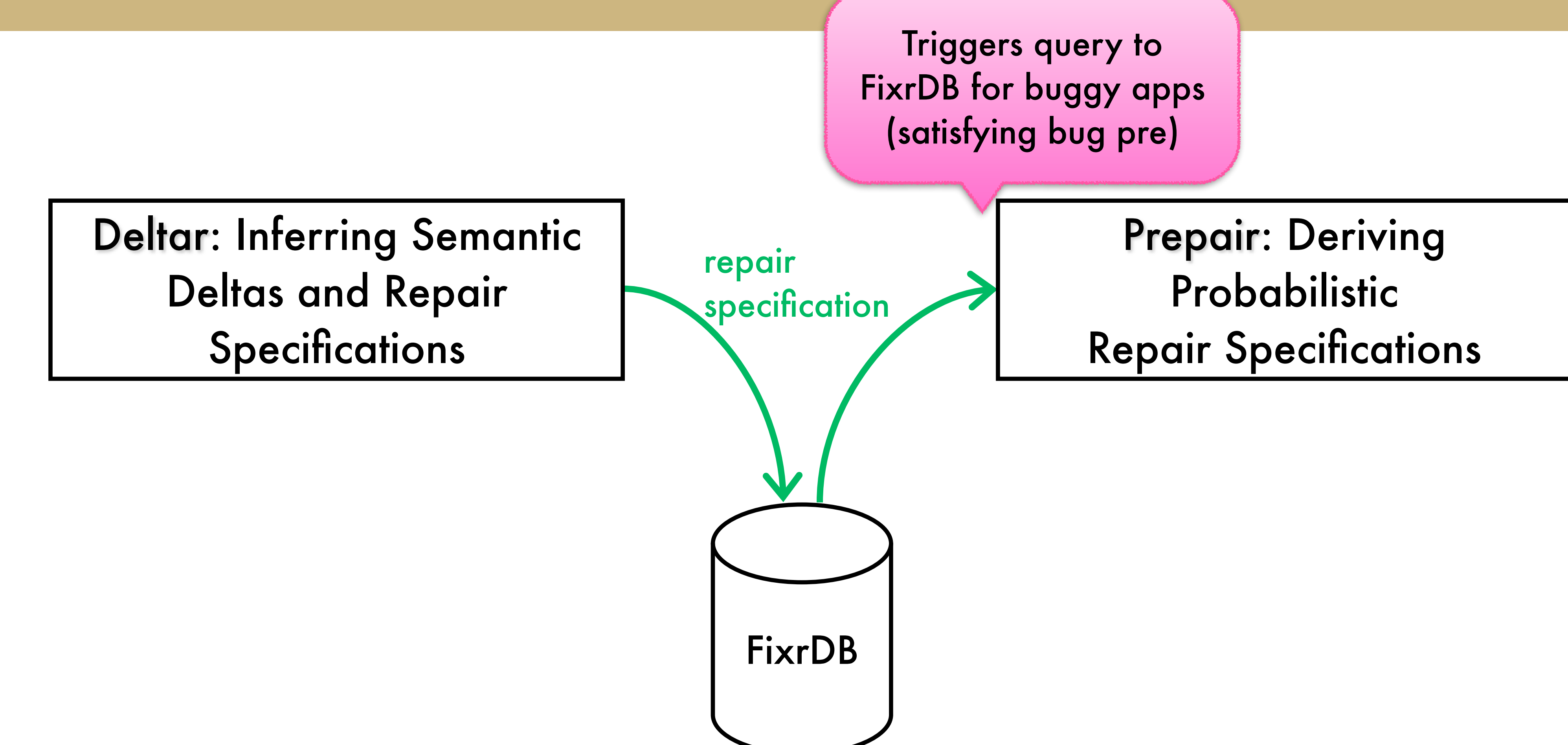
Workflow 1b: Aggregating repairs



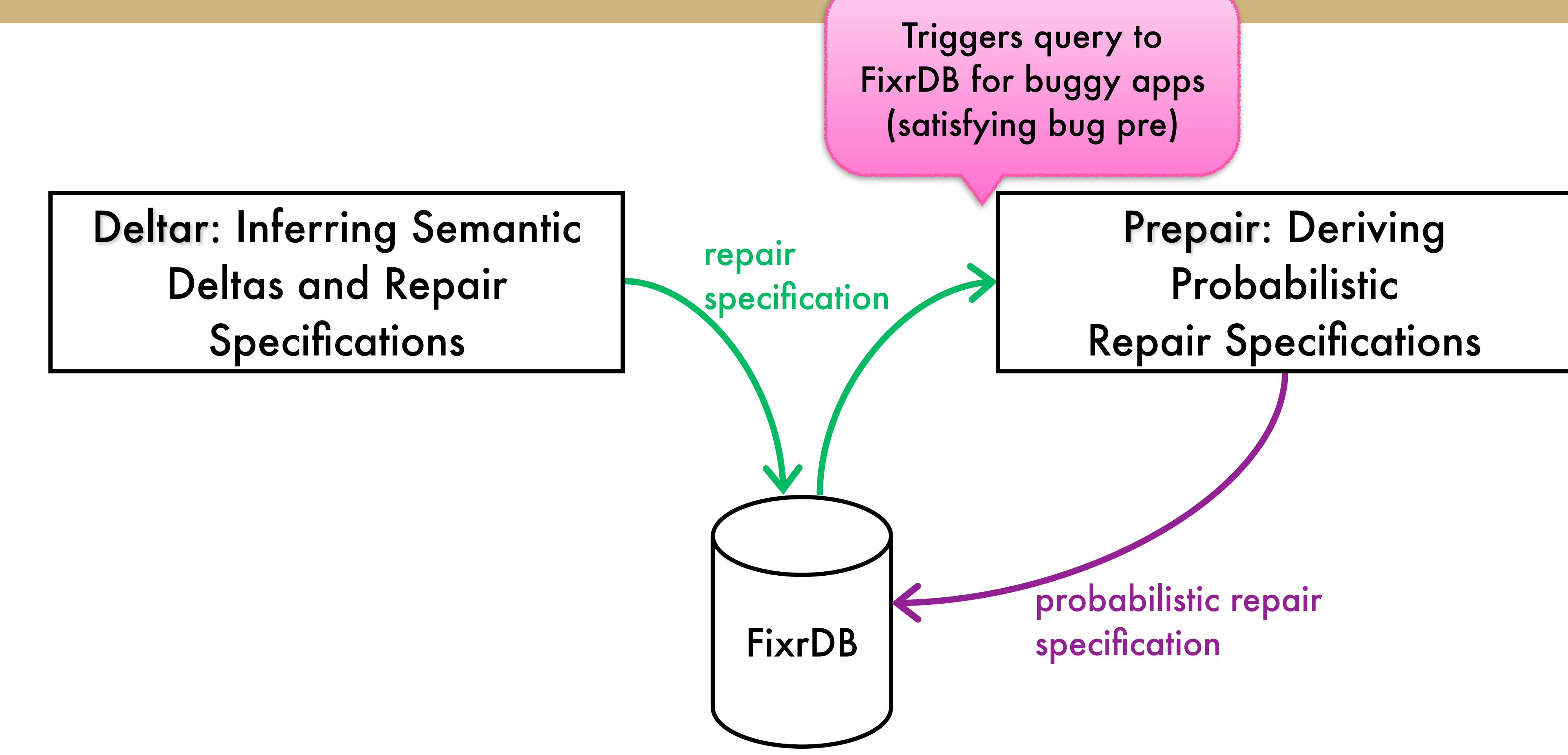
Workflow 1b: Aggregating repairs



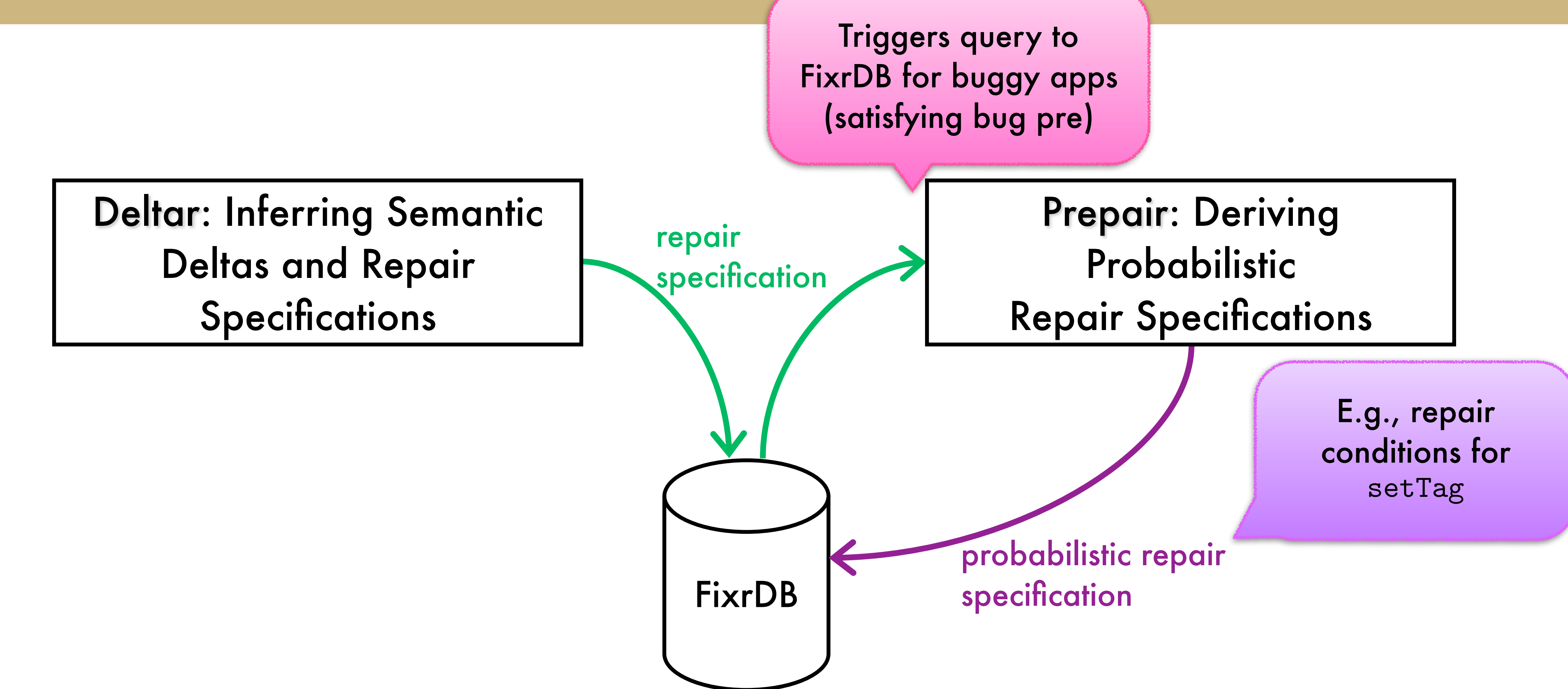
Workflow 1b: Aggregating repairs



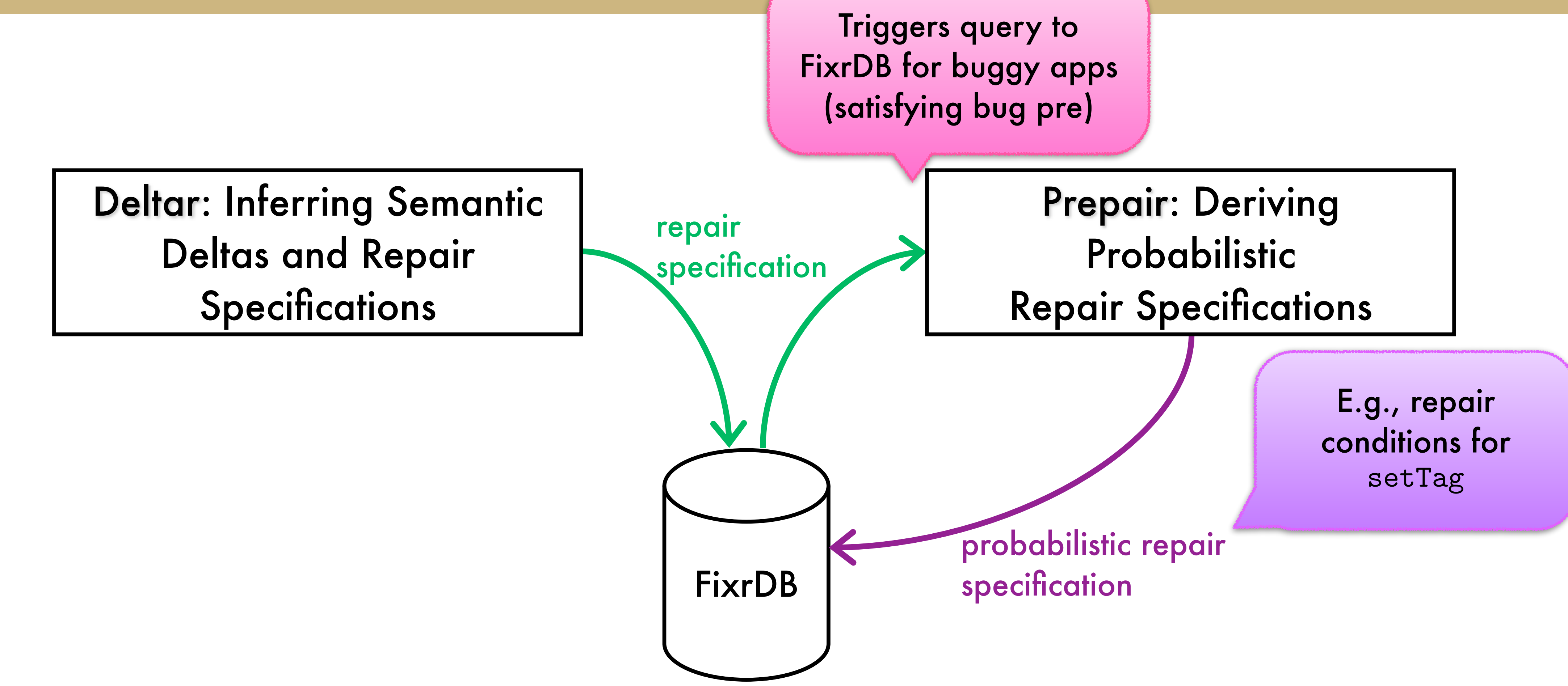
Workflow 1b: Aggregating repairs



Workflow 1b: Aggregating repairs



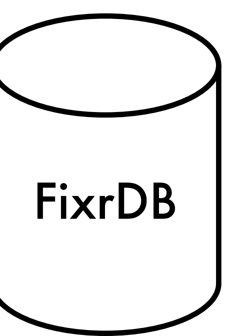
Workflow 1b: Aggregating repairs



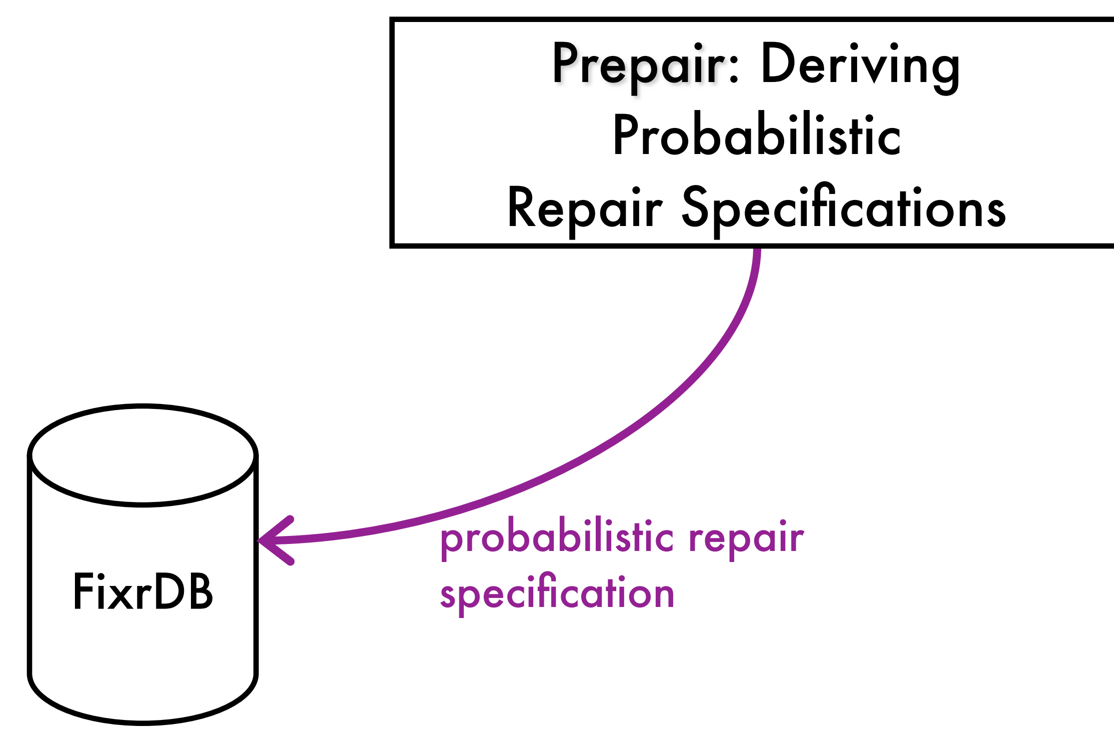
Component: Prepair *reduces* candidate repair specifications to generalized probabilistic repair specifications

Workflow 1c: Synthesizing patches

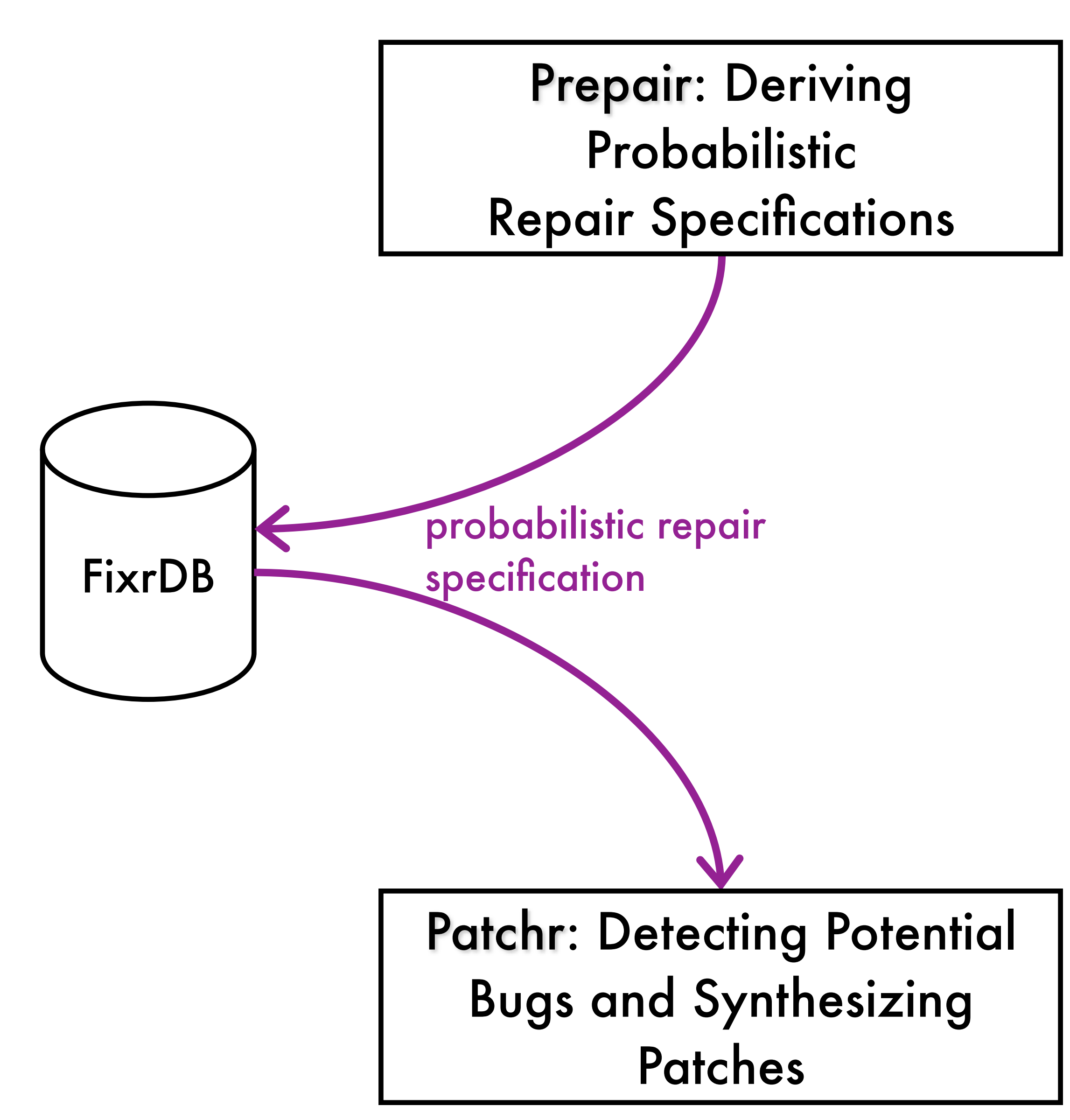
Prepair: Deriving Probabilistic Repair Specifications



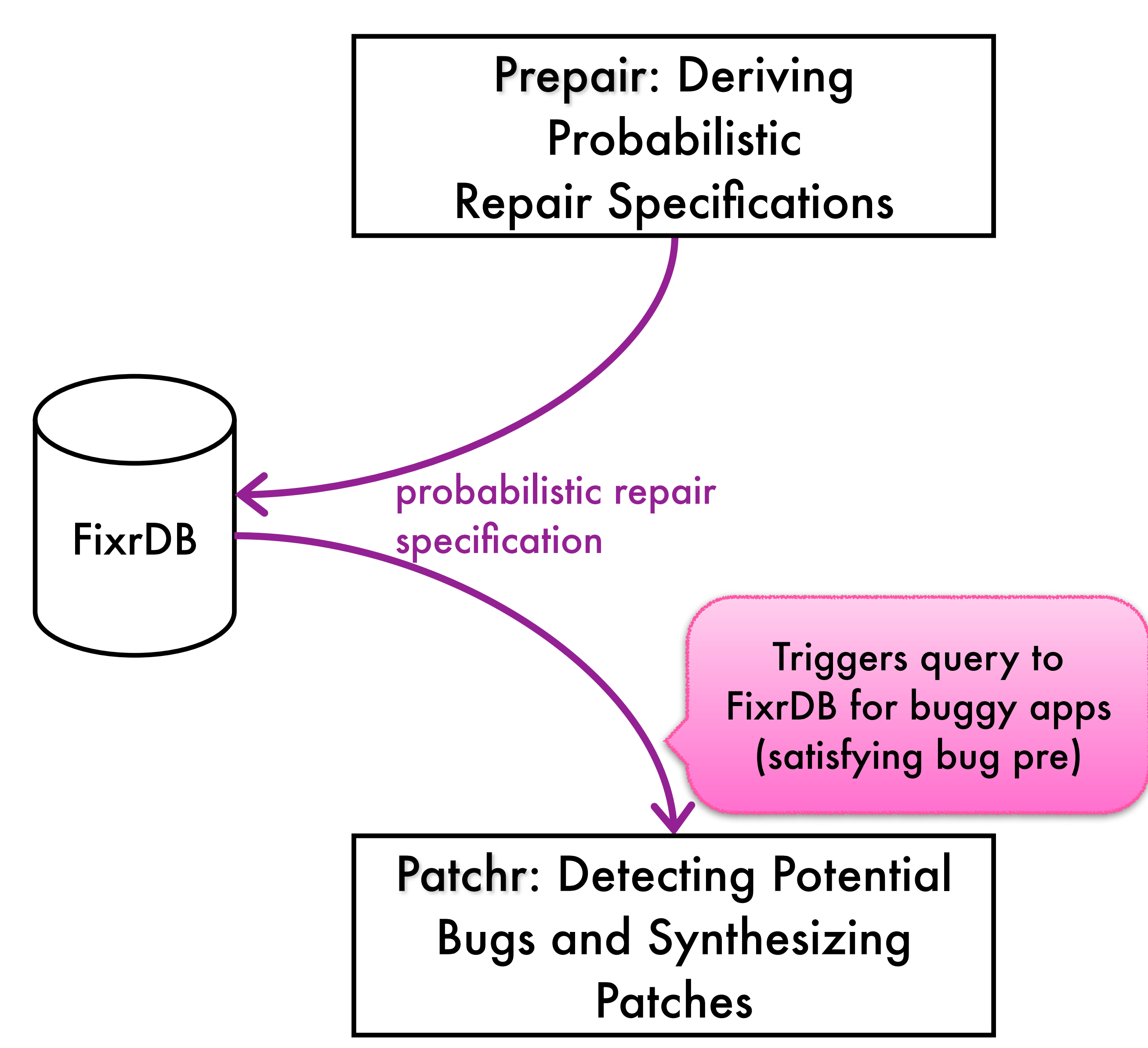
Workflow 1c: Synthesizing patches



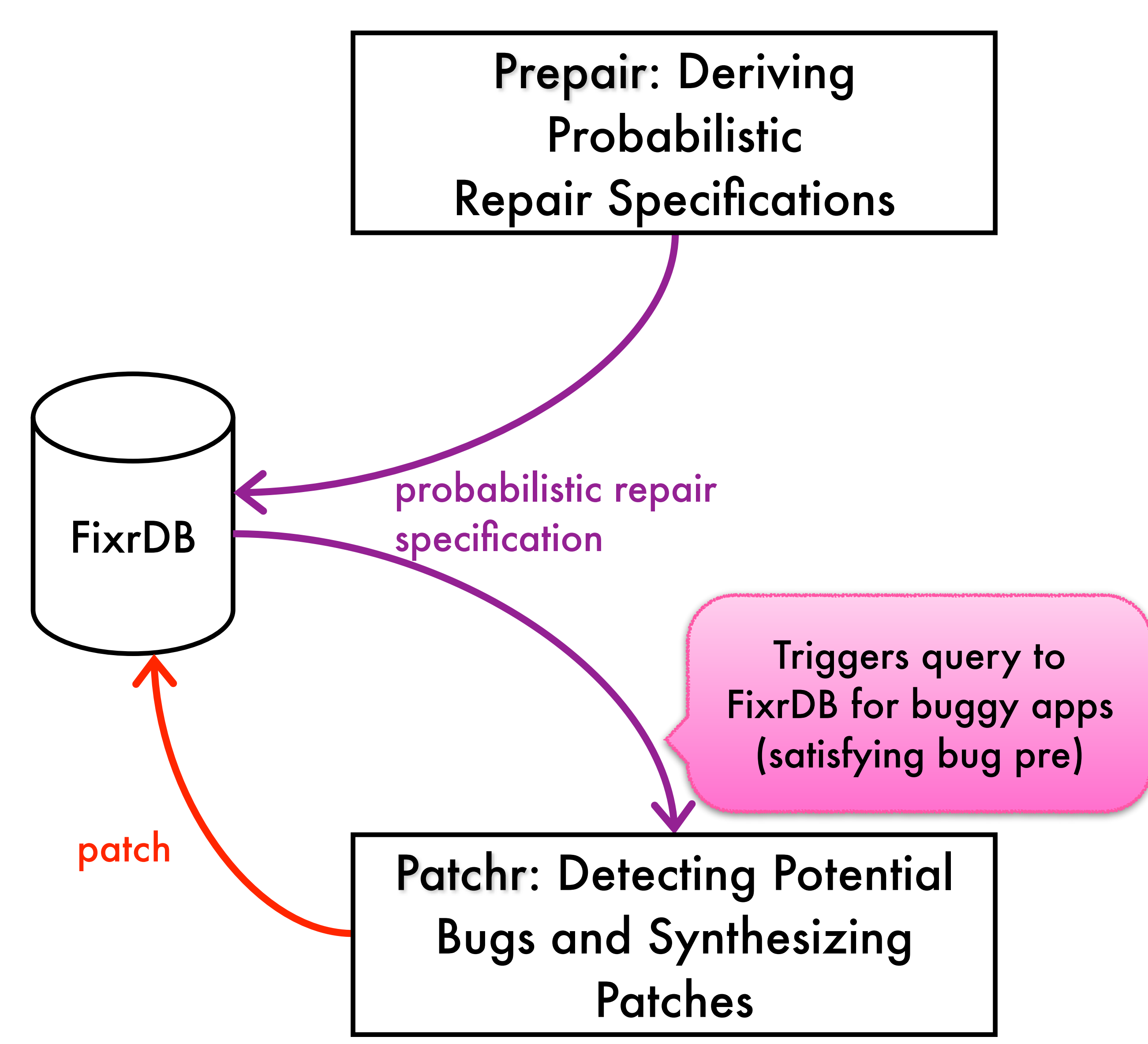
Workflow 1c: Synthesizing patches



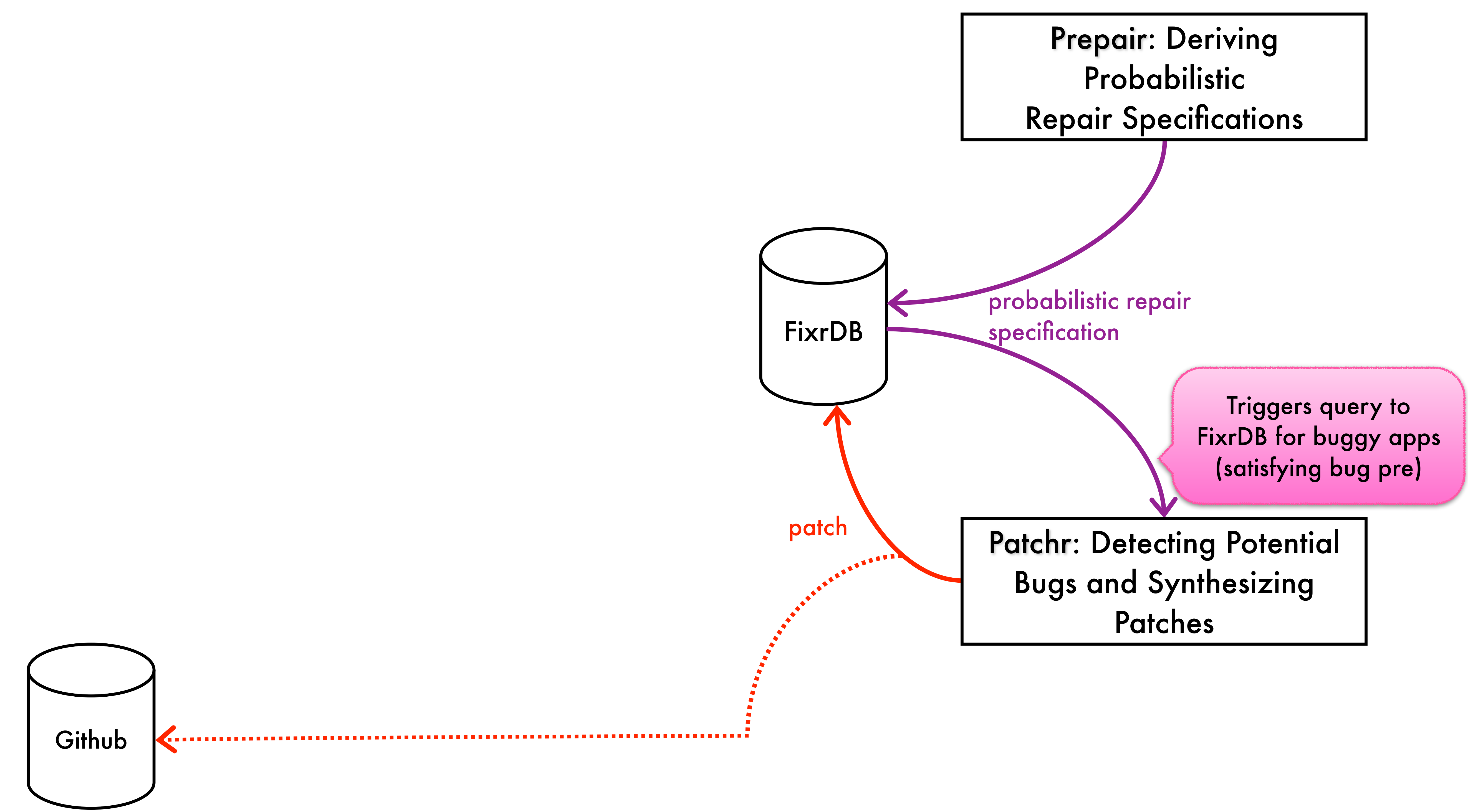
Workflow 1c: Synthesizing patches



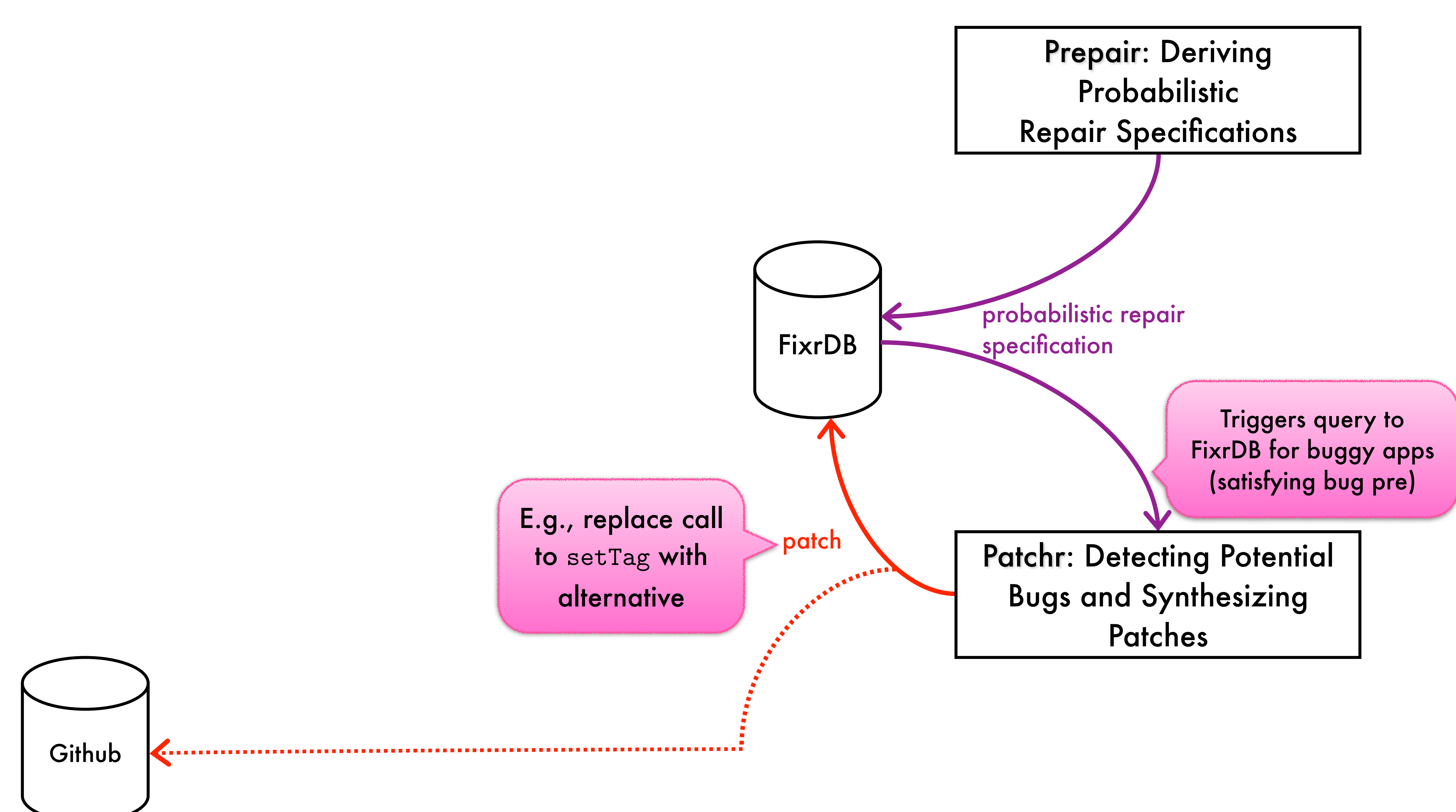
Workflow 1c: Synthesizing patches



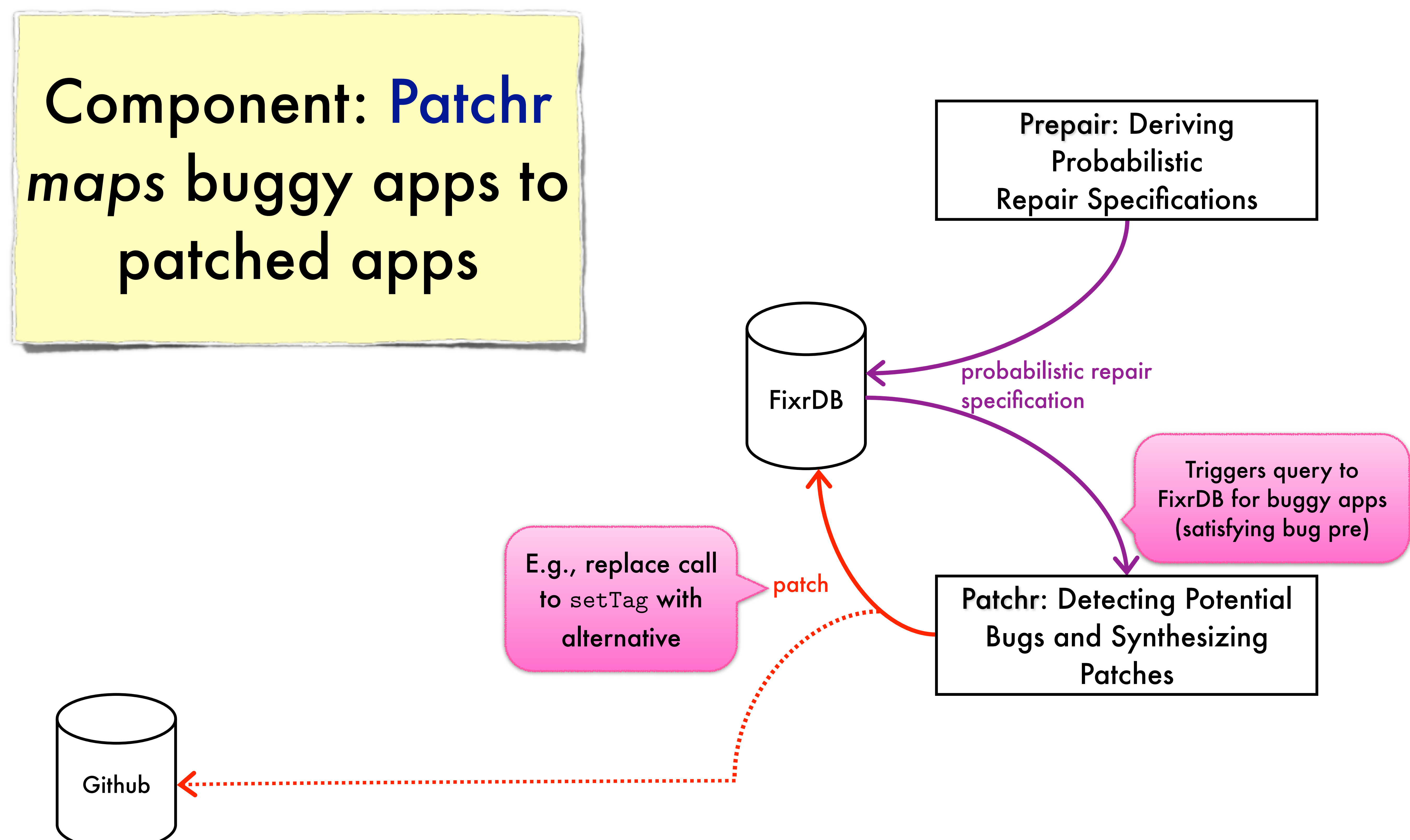
Workflow 1c: Synthesizing patches



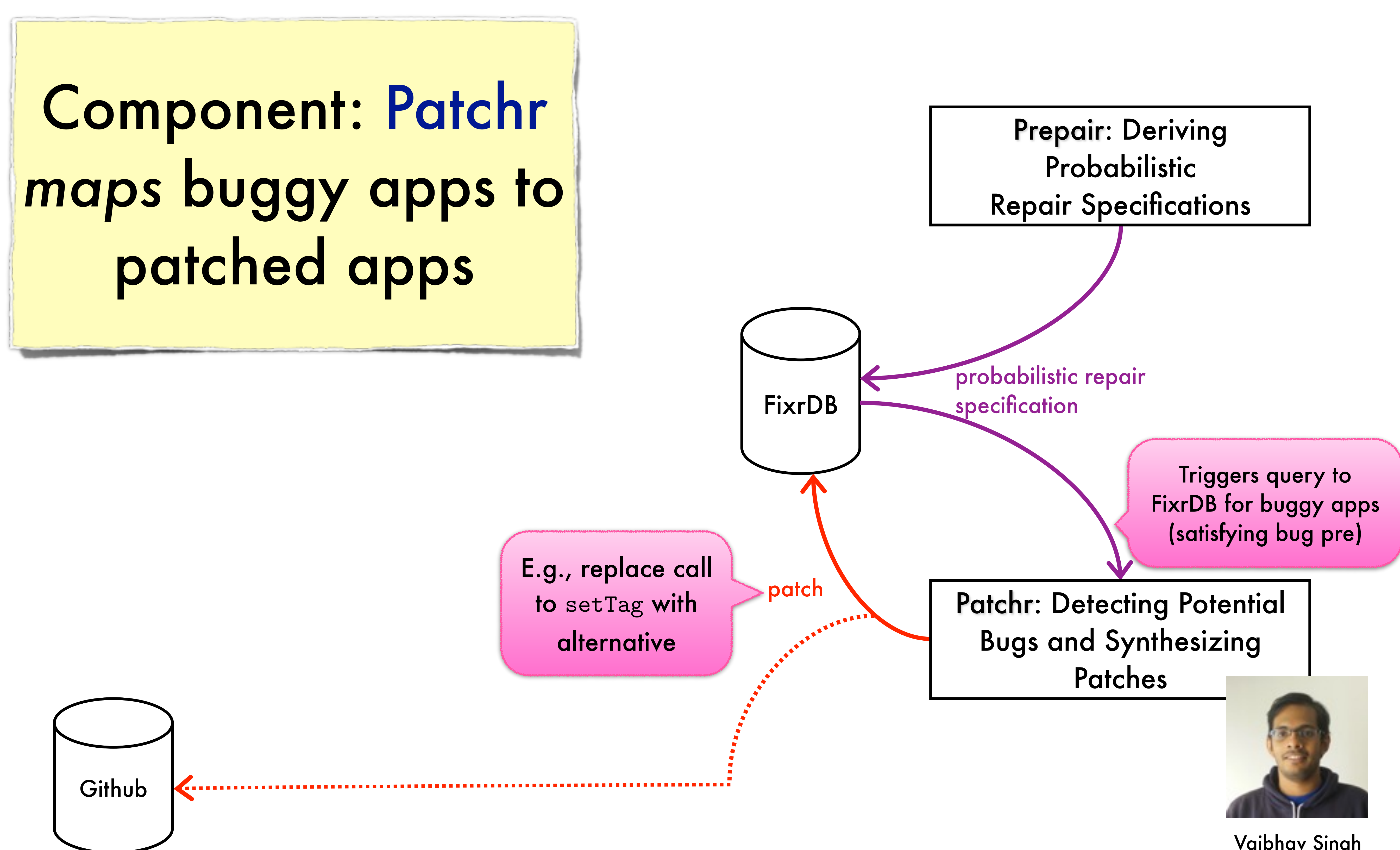
Workflow 1c: Synthesizing patches



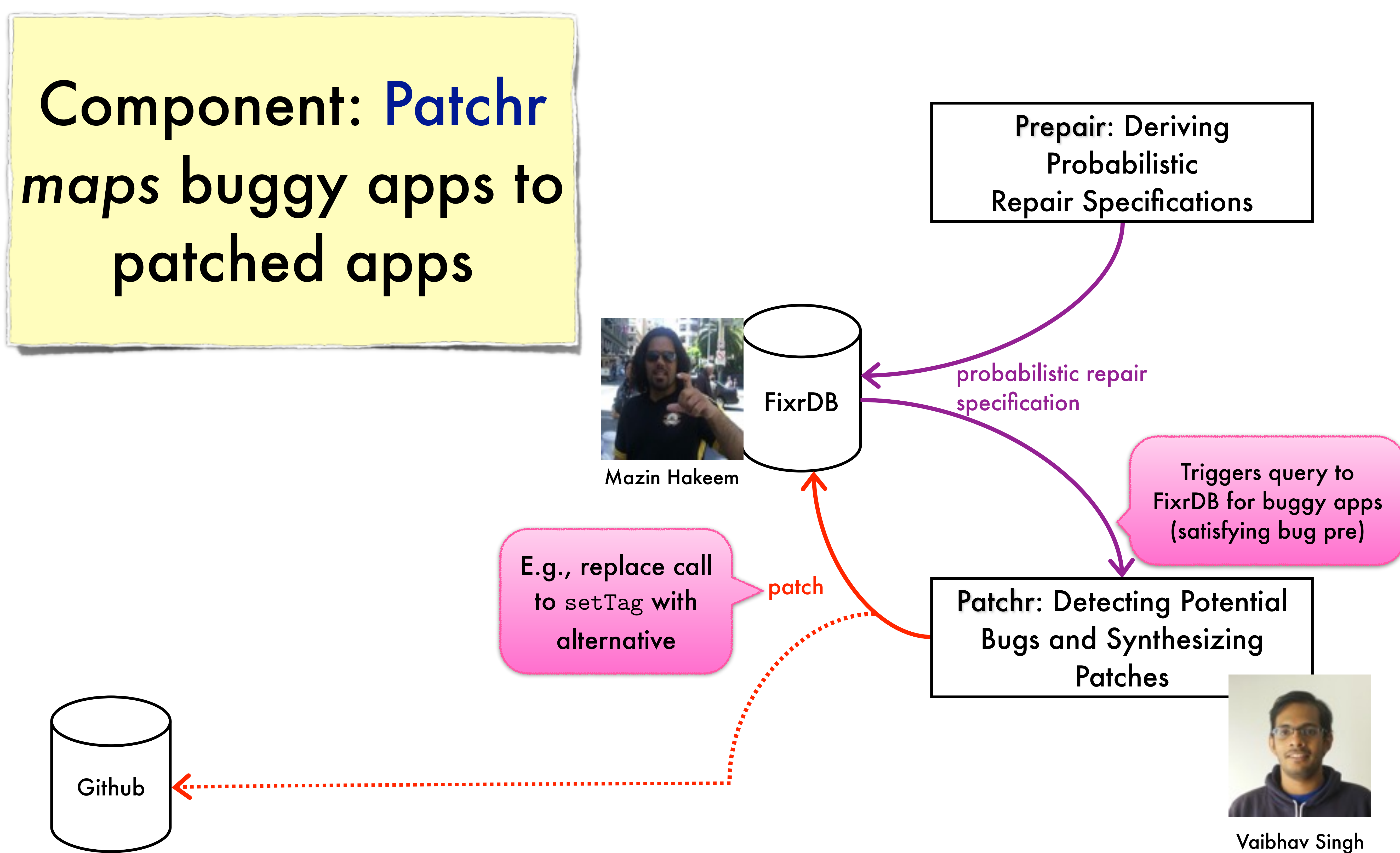
Workflow 1c: Synthesizing patches



Workflow 1c: Synthesizing patches



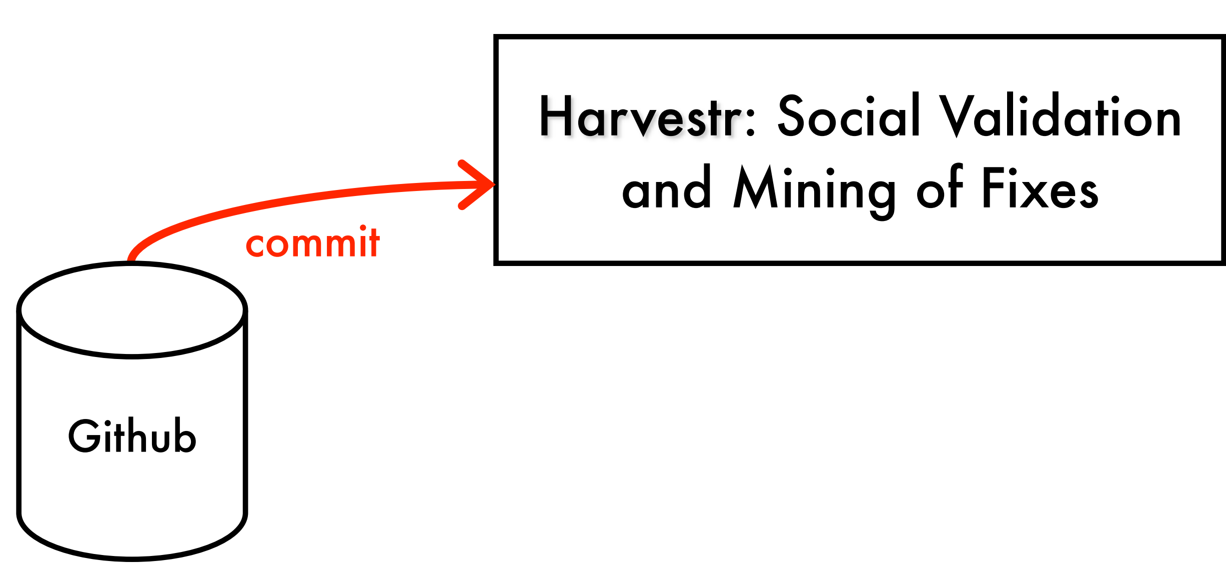
Workflow 1c: Synthesizing patches



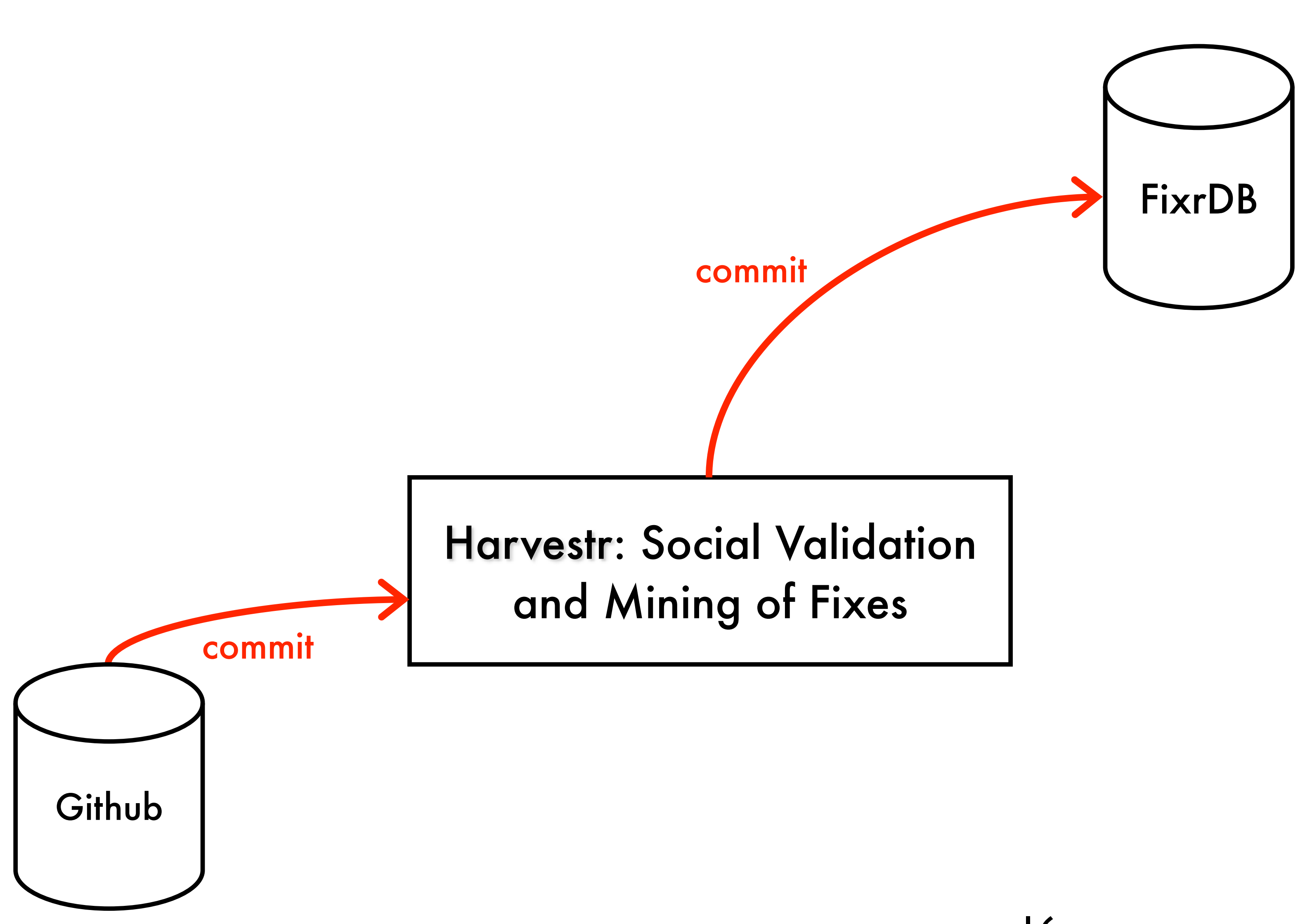
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation

Harvestr: Social Validation and Mining of Fixes

Workflow 0: Continuous commit harvesting, buggy app patching, and social validation

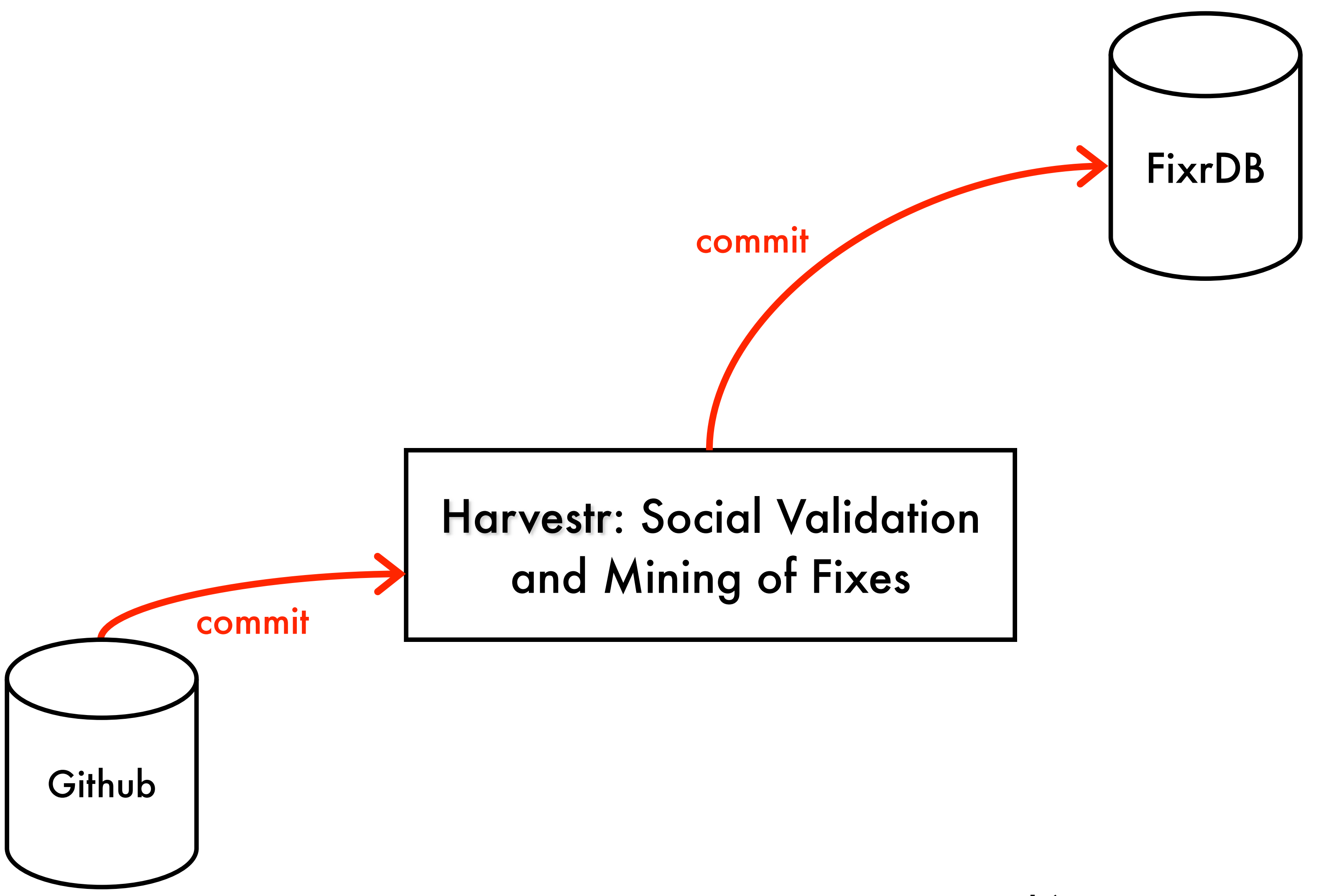


Workflow 0: Continuous commit harvesting, buggy app patching, and social validation

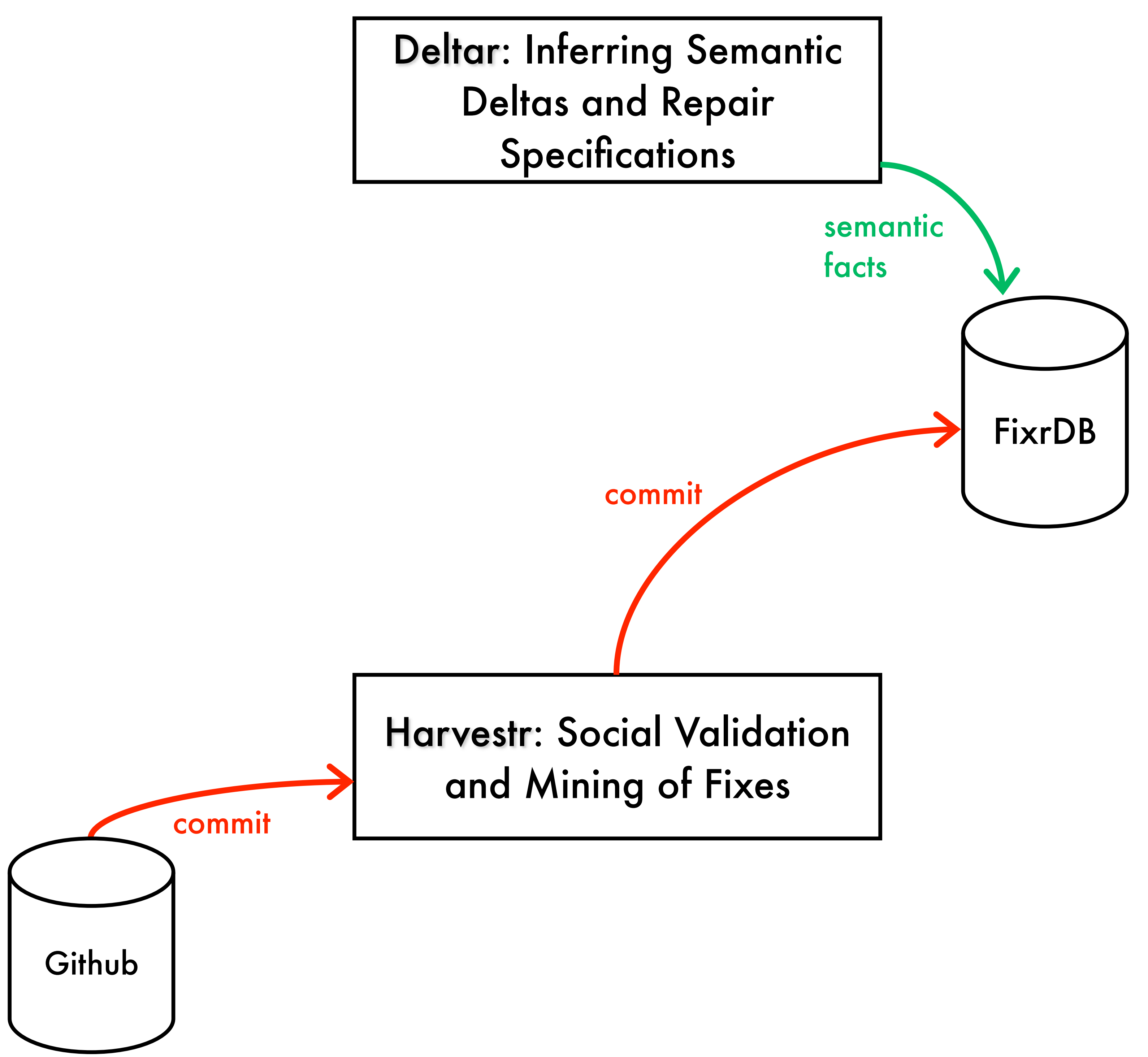


Workflow 0: Continuous commit harvesting, buggy app patching, and social validation

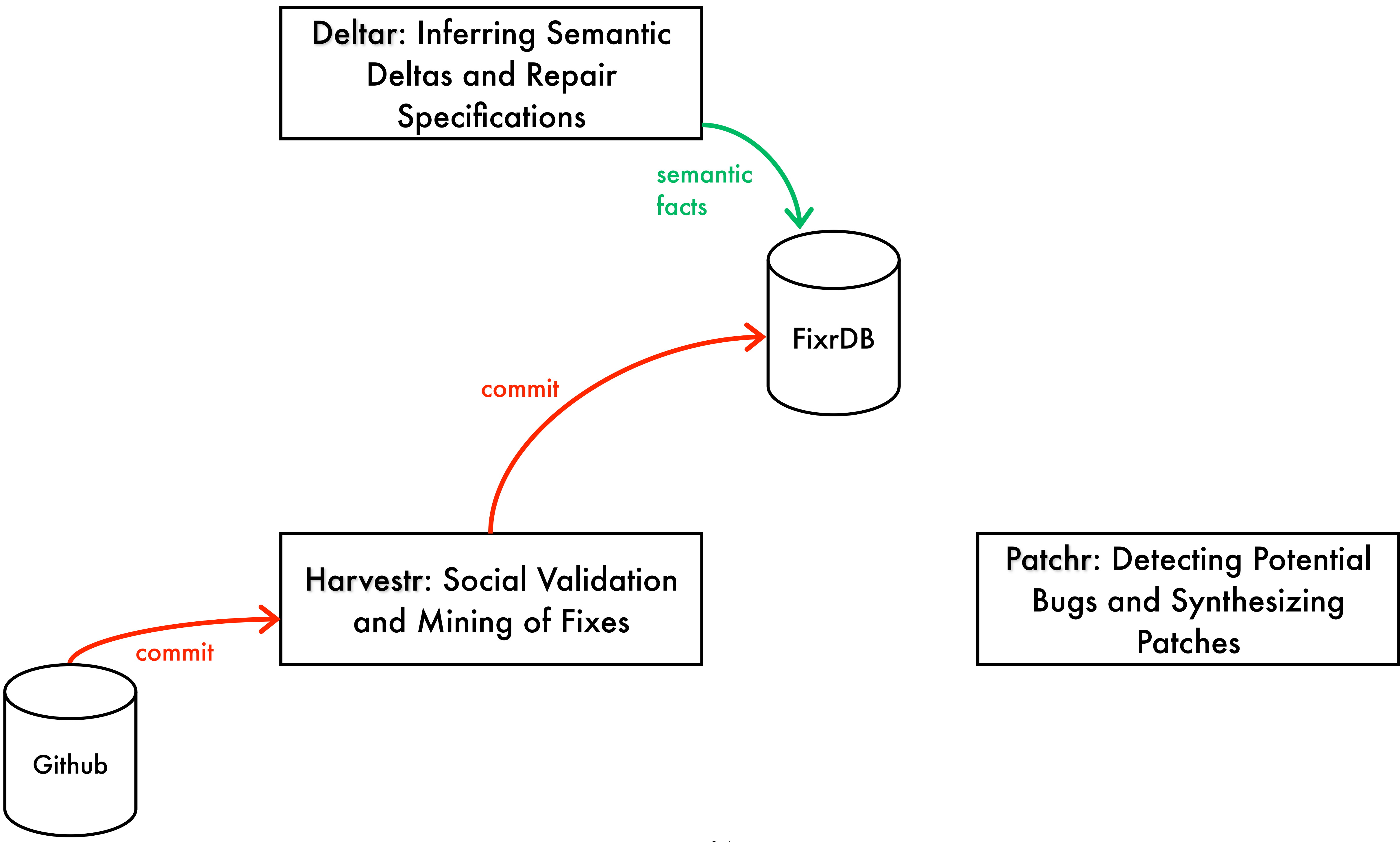
Deltar: Inferring Semantic Deltas and Repair Specifications



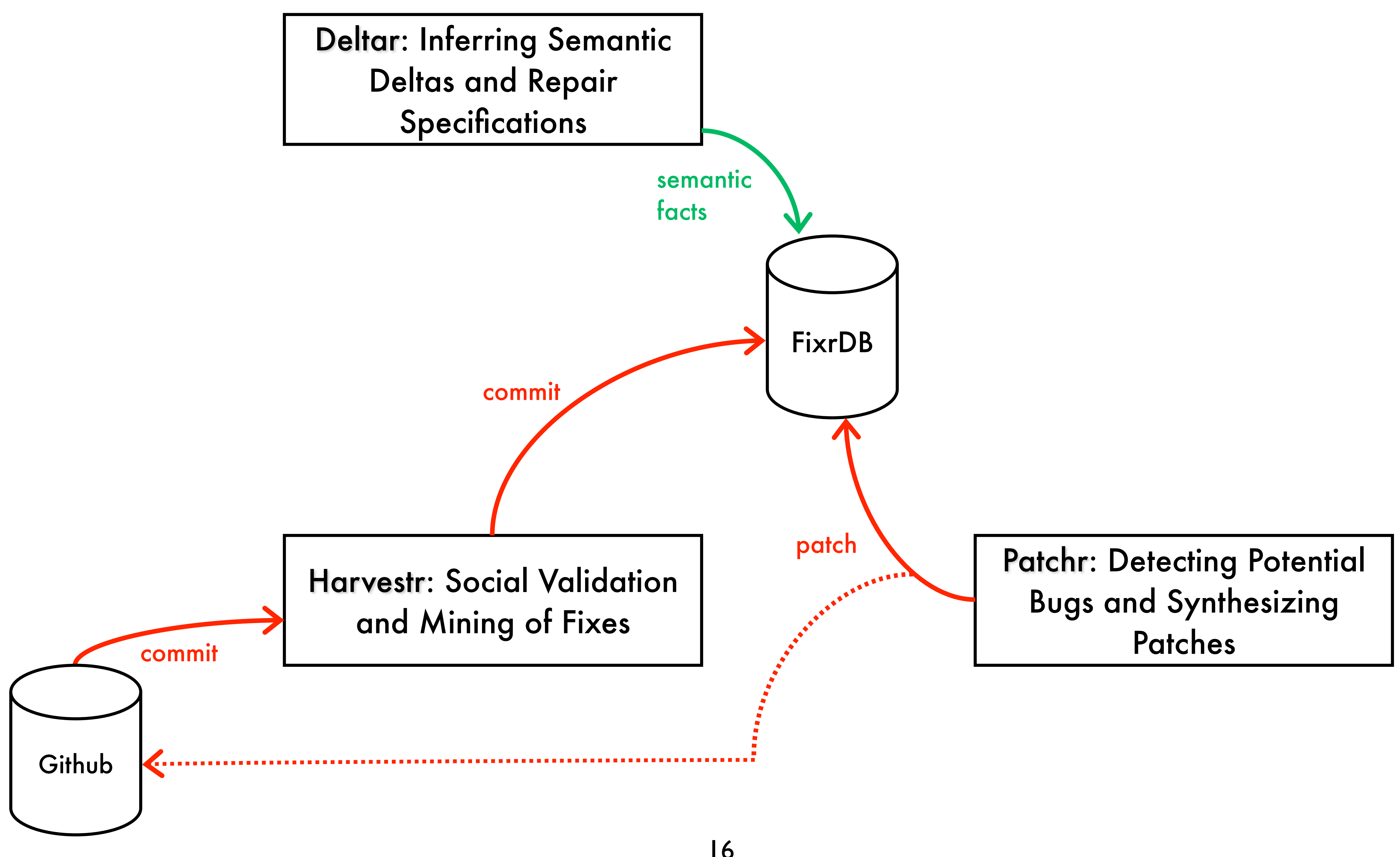
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation



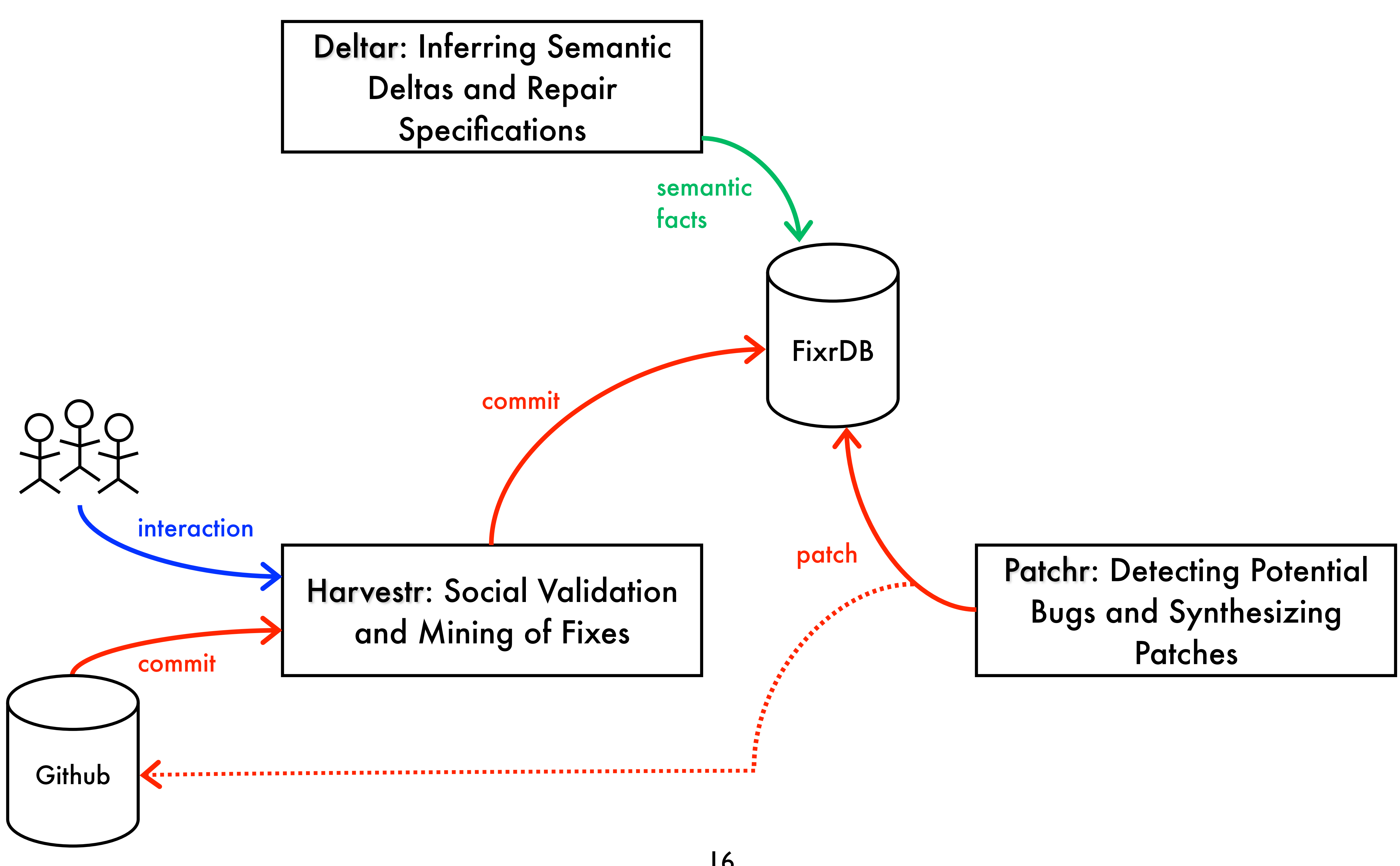
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation



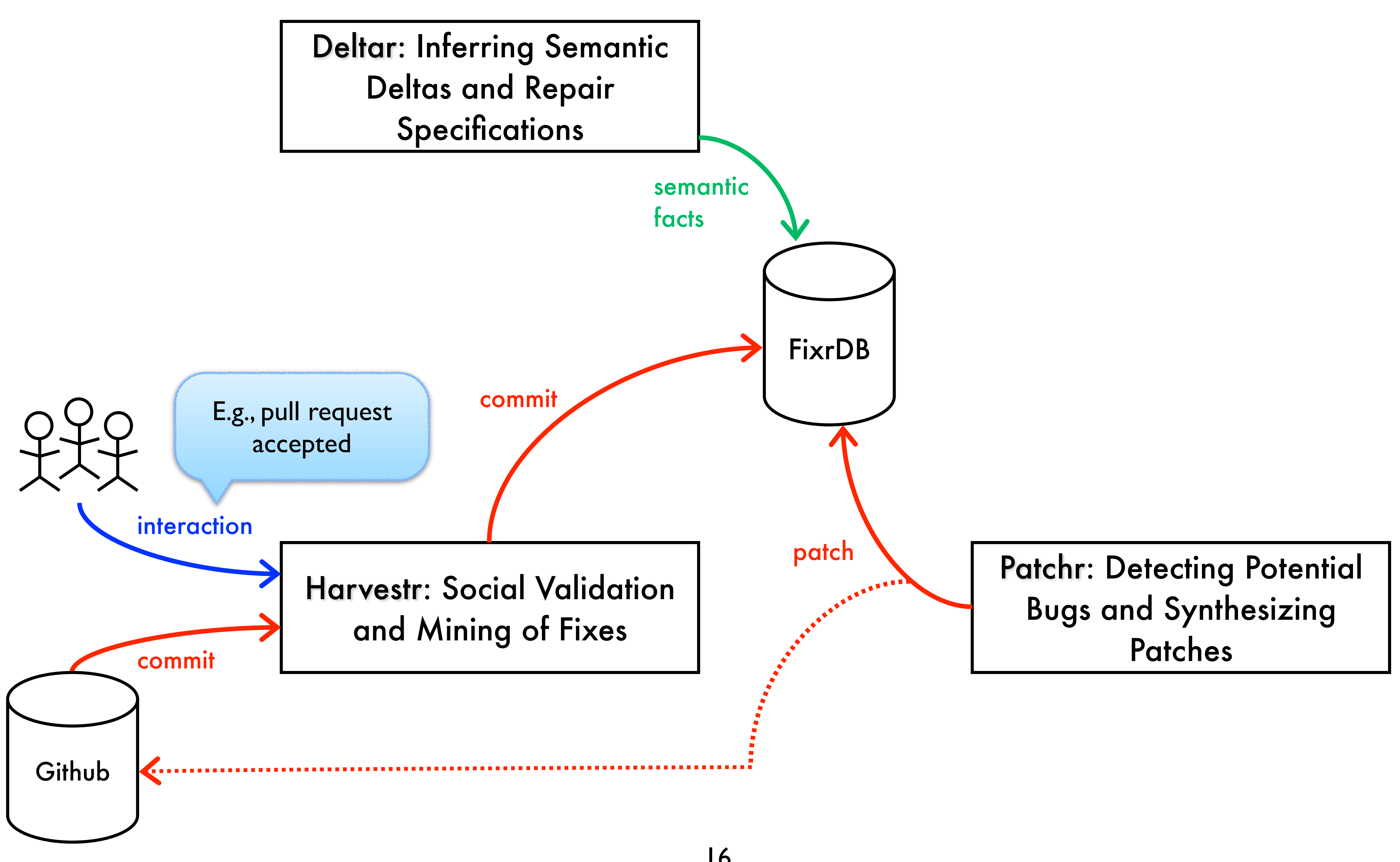
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation



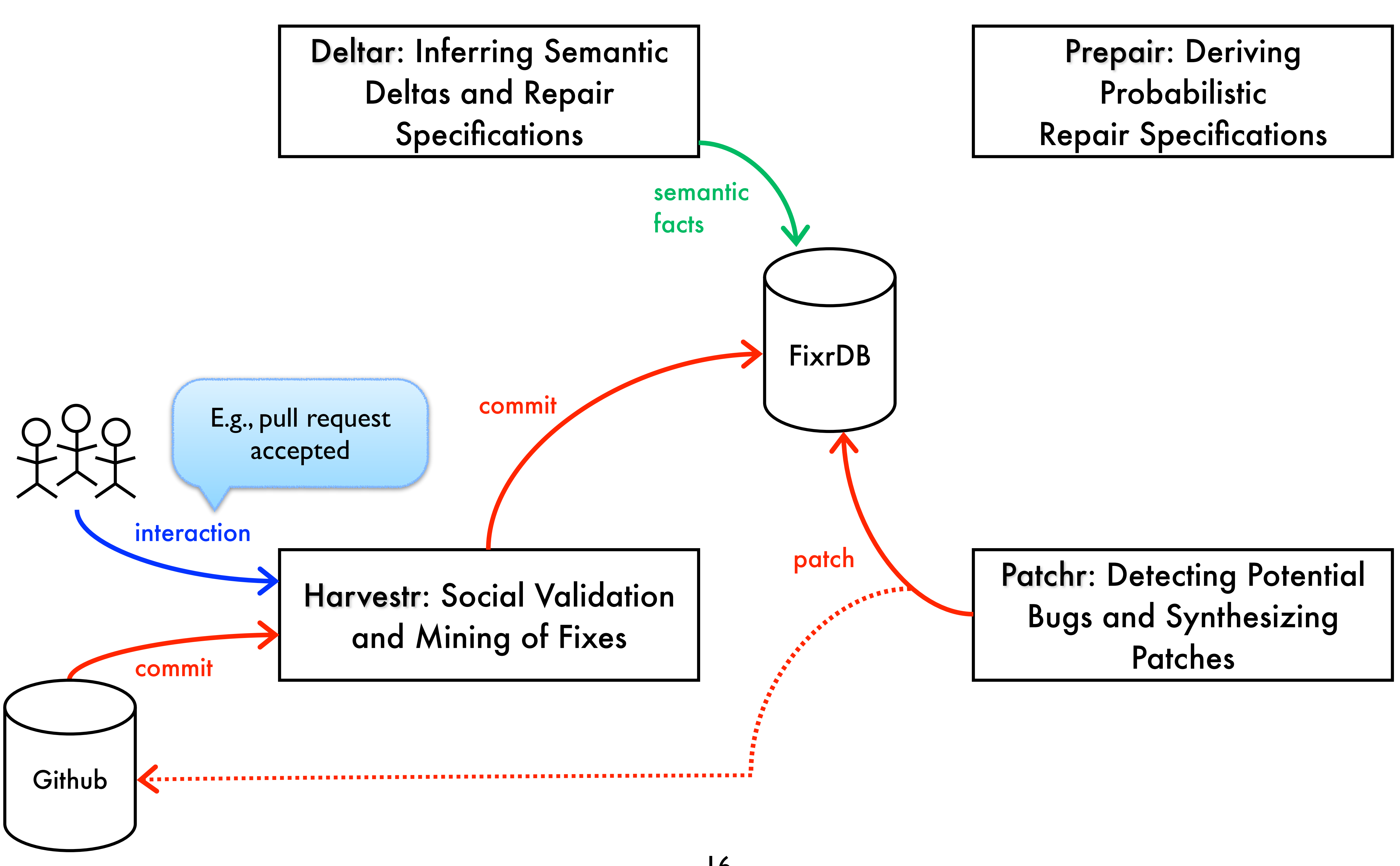
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation



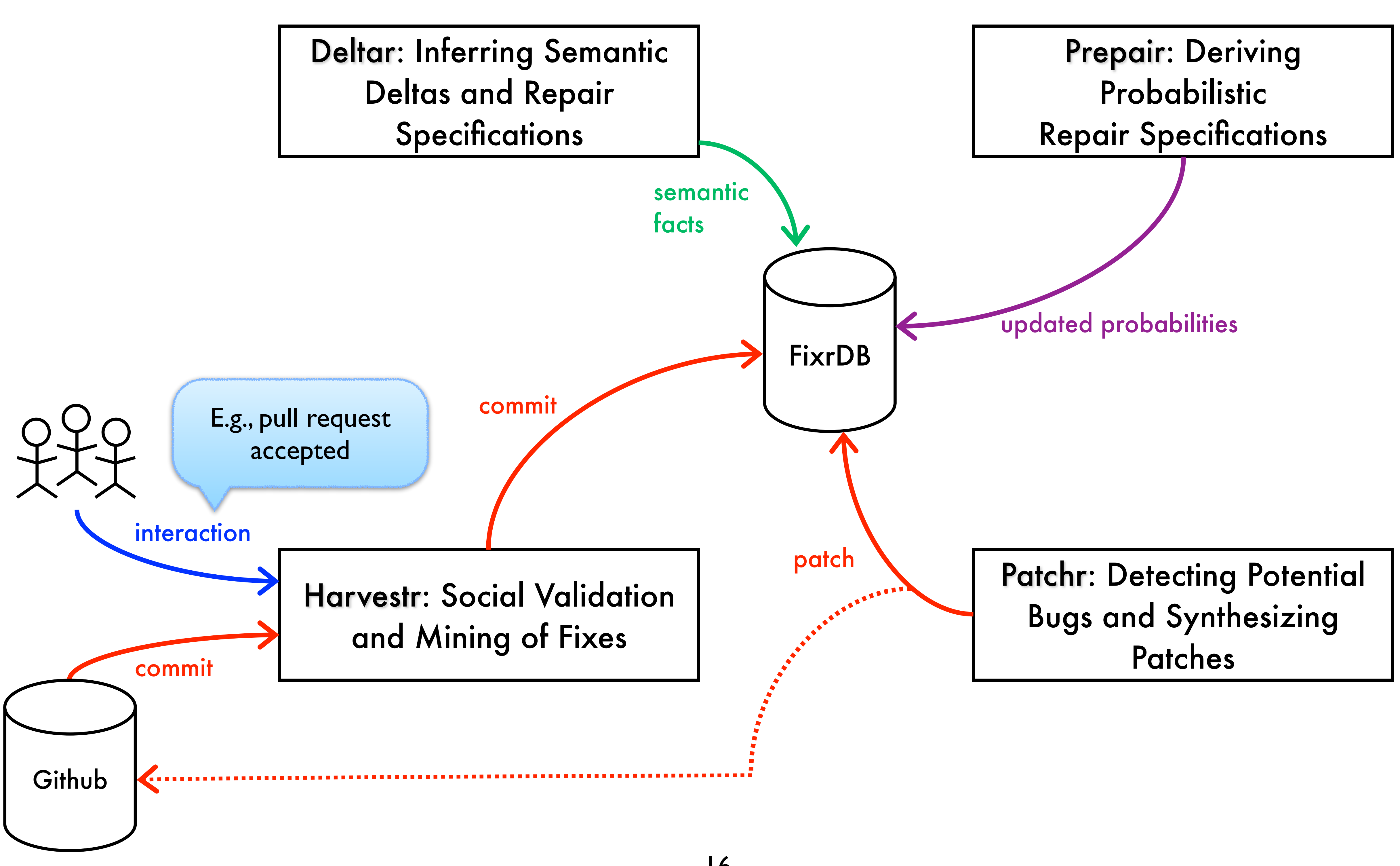
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation



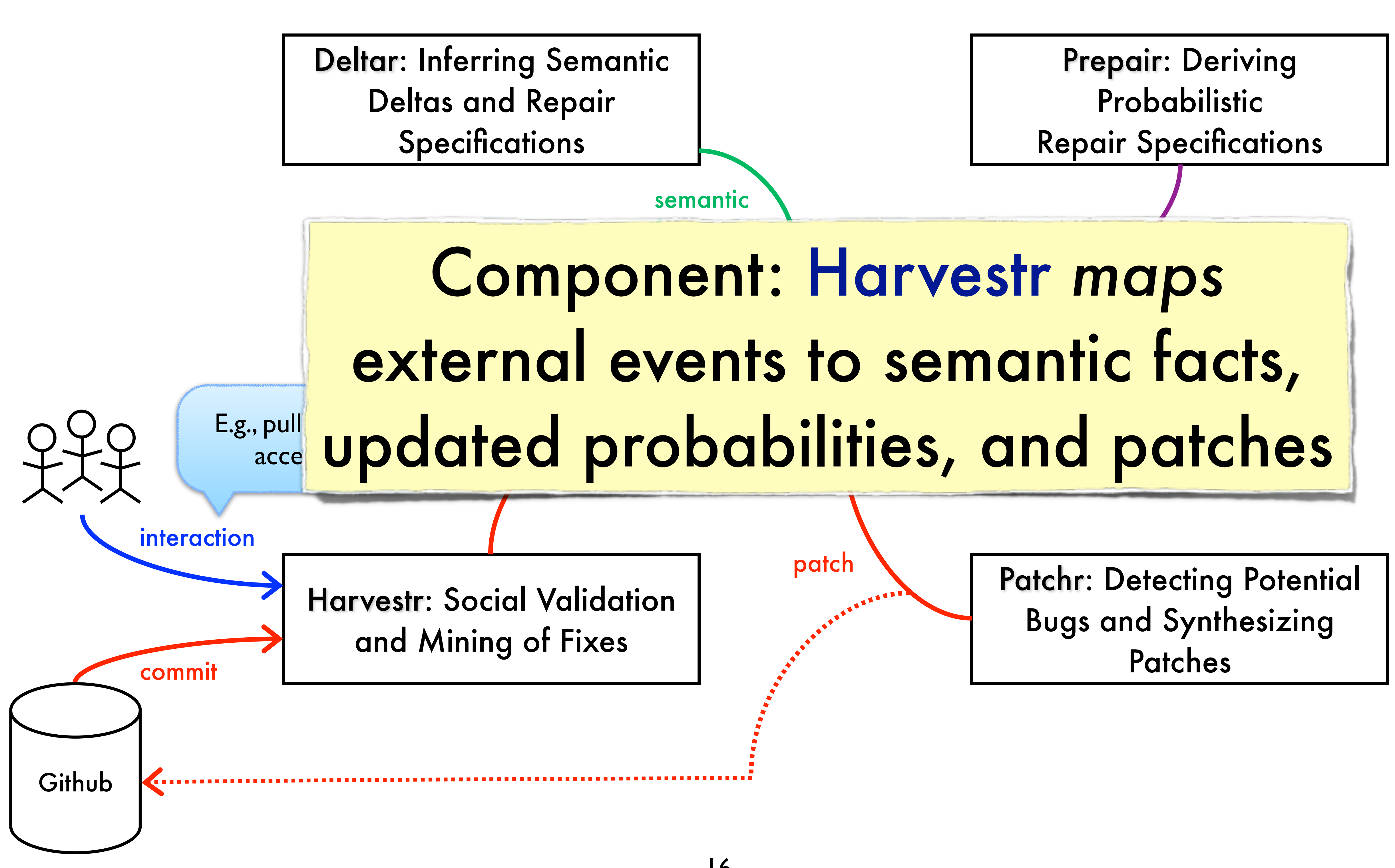
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation



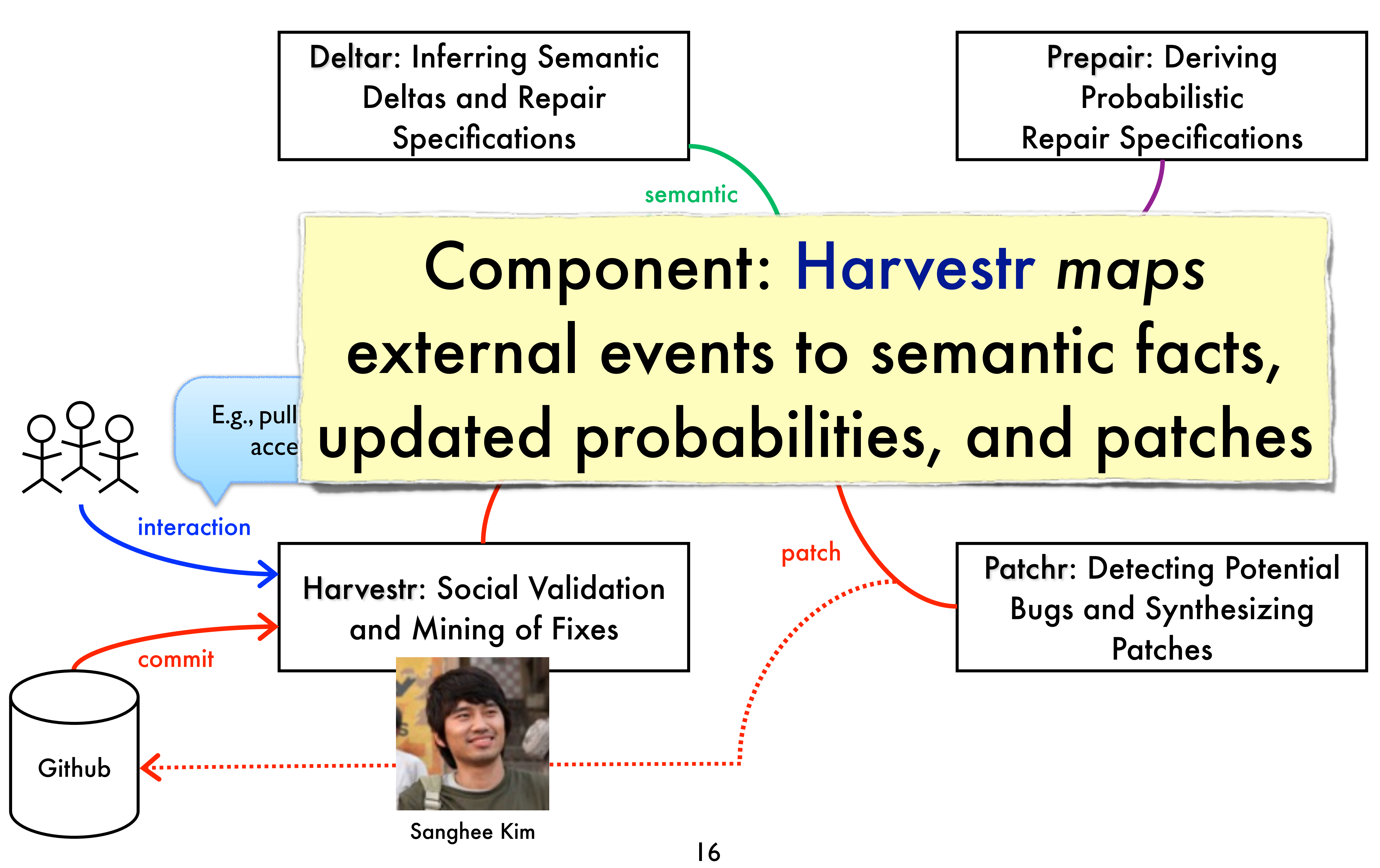
Workflow 0: Continuous commit harvesting, buggy app patching, and social validation

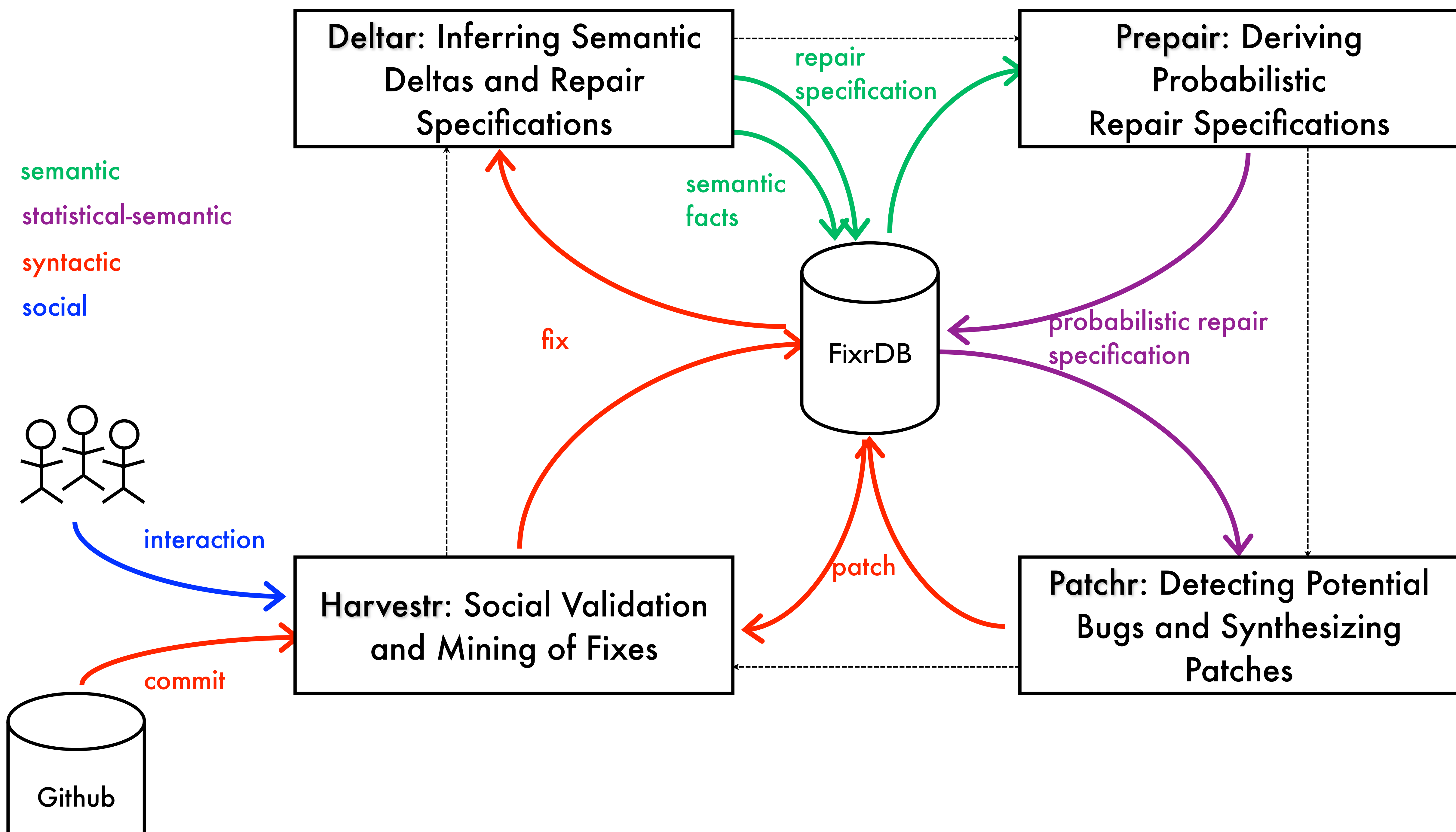


Workflow 0: Continuous commit harvesting, buggy app patching, and social validation



Workflow 0: Continuous commit harvesting, buggy app patching, and social validation




Deltar: Inferring Semantic Deltas and Repair Specifications
Prepair: Deriving Probabilistic Repair Specifications
repair specification
semantic facts
semantic
statistical-semantic
syntactic
social
fix
FixrDB
probabilistic repair specification
interaction
Harvestr: Social Validation and Mining of Fixes
patch
Patchr: Detecting Potential Bugs and Synthesizing Patches
commit
Github

Bor-Yuh Evan Chang

symbolic
program analysis

Sriram Sankaranarayanan

numerical-probabilistic
program analysis

Deltar: Inferring Semantic Deltas and Repair Specifications

repair
specification

Prepair: Deriving Probabilistic Repair Specifications

semantic
statistical-semantic
syntactic
social

semantic
facts

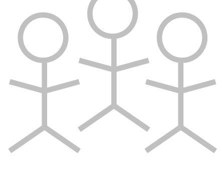
software engineering
for big data

fix

Ken Anderson

FixrDB

probabilistic repair
specification



interaction

Harvestr: Social Validation and Mining of Fixes

patch

Patchr: Detecting Potential Bugs and Synthesizing Patches

commit

user-centered
big data analytics

program synthesis

Tom Yeh

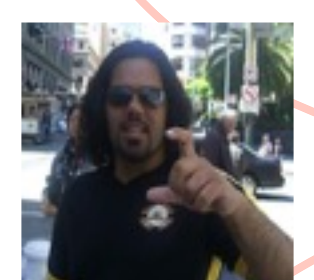
Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

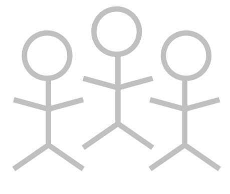
repair specification

semantic facts

semantic
statistical-semantic
syntactic
social



probabilistic repair specification



interaction

patch

Harvestr: Social Validation and Mining of Fixes

Patchr: Detecting Potential Bugs and Synthesizing Patches

commit