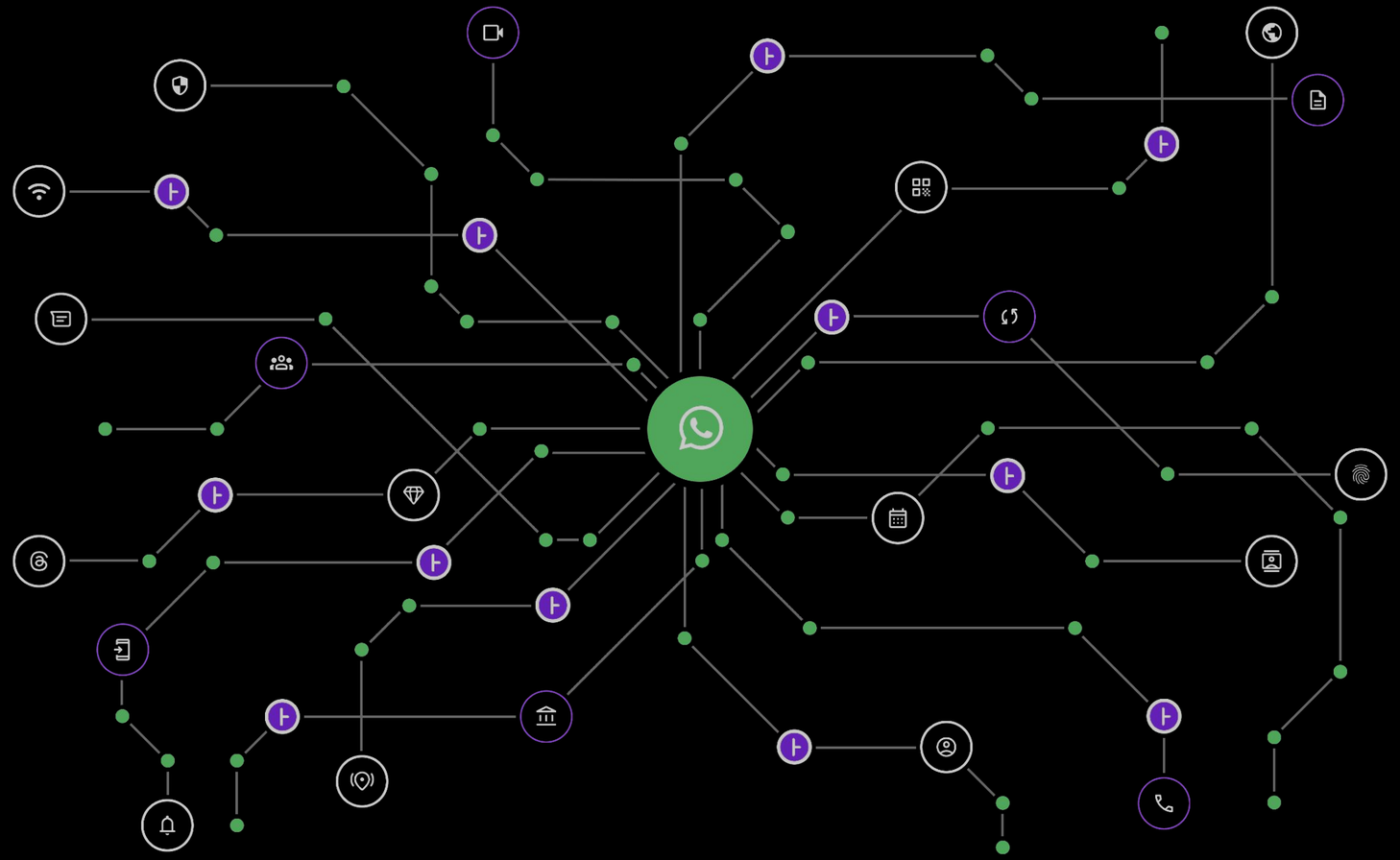
Compositional Static Callgraph Reachability Analysis for WhatsApp Android App Health



Ákos Hajdu
Meta

Motivation

Performance critical functions (transitively) calling computationally expensive ones?

```
void somethingOnTheUI() {
 checkSomething();
}

void checkSomething() {
 readFromDatabase();
}

void readFromDatabase() {
 /* Slow stuff */
}
```

Challenges

Specification

What's performance critical and what's computationally expensive?

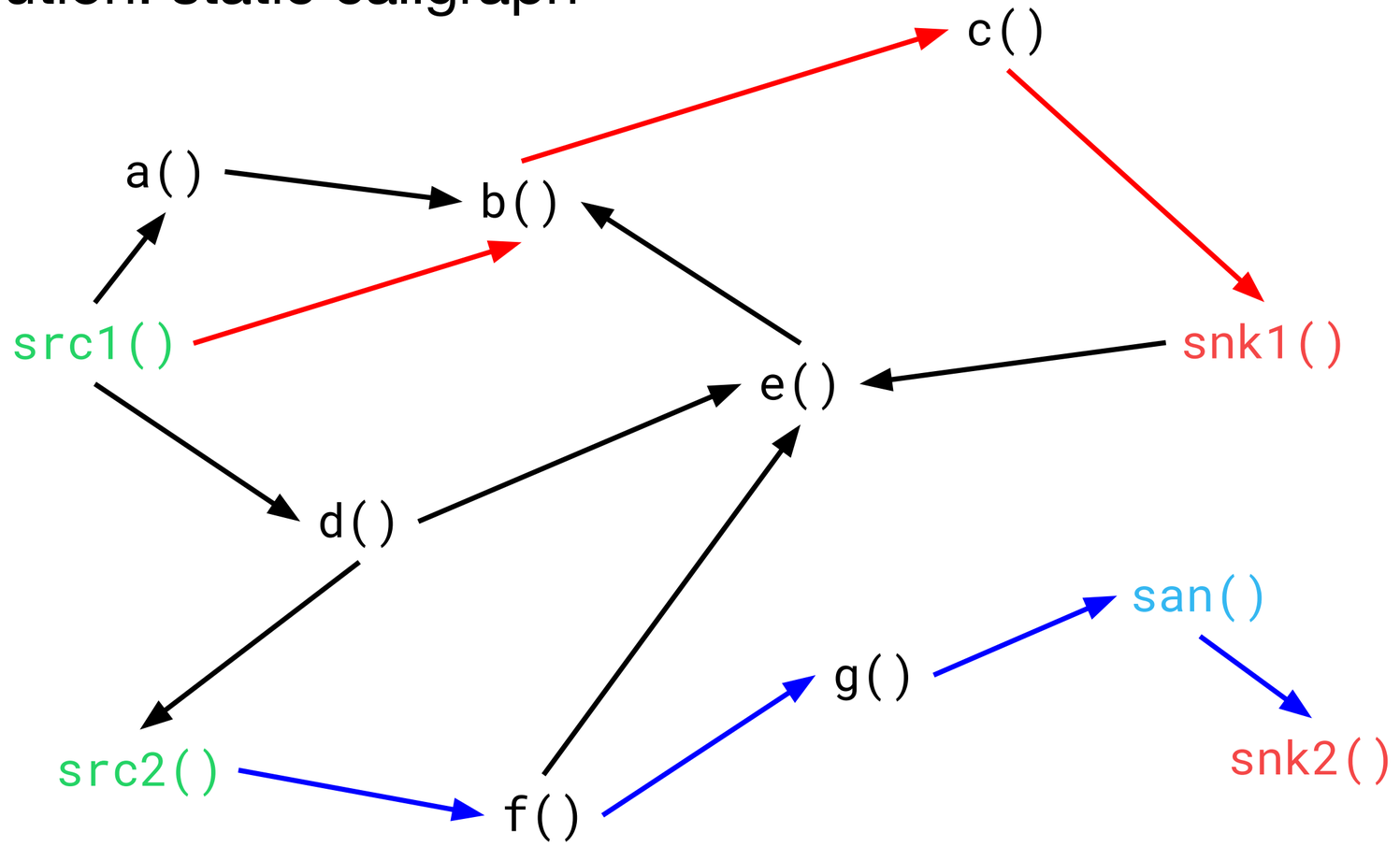
Analyzer

Automated and fast enough to give feedback on code changes

Reachability

Source to sink call chain without sanitizer

Approximate solution: static callgraph



Infer

Open-source static analysis platform

fbinfer.com
github.com/facebook/infer

Developed at Meta

Runs on tens of thousands of code changes monthly, reporting thousands of issues

Language frontends

C, C++, Objective-C, Java/Kotlin, C#, Erlang

Checkers

Memory safety, data races, deadlocks, temporal properties, annotation reachability, …

Annotation reachability - Specification

Java annotations

```
@PerfCrit
void somethingOnTheUI() {
 checkSomething();
}

void checkSomething() {
 readFromDatabase();
}

@Expensive
void readFromDatabase() {
 /* Slow stuff */
}
```

```
"annotation-reachability-custom-pairs":
[{
 "sources": ["PerfCrit"],
 "sinks": ["Expensive"]
}]
```

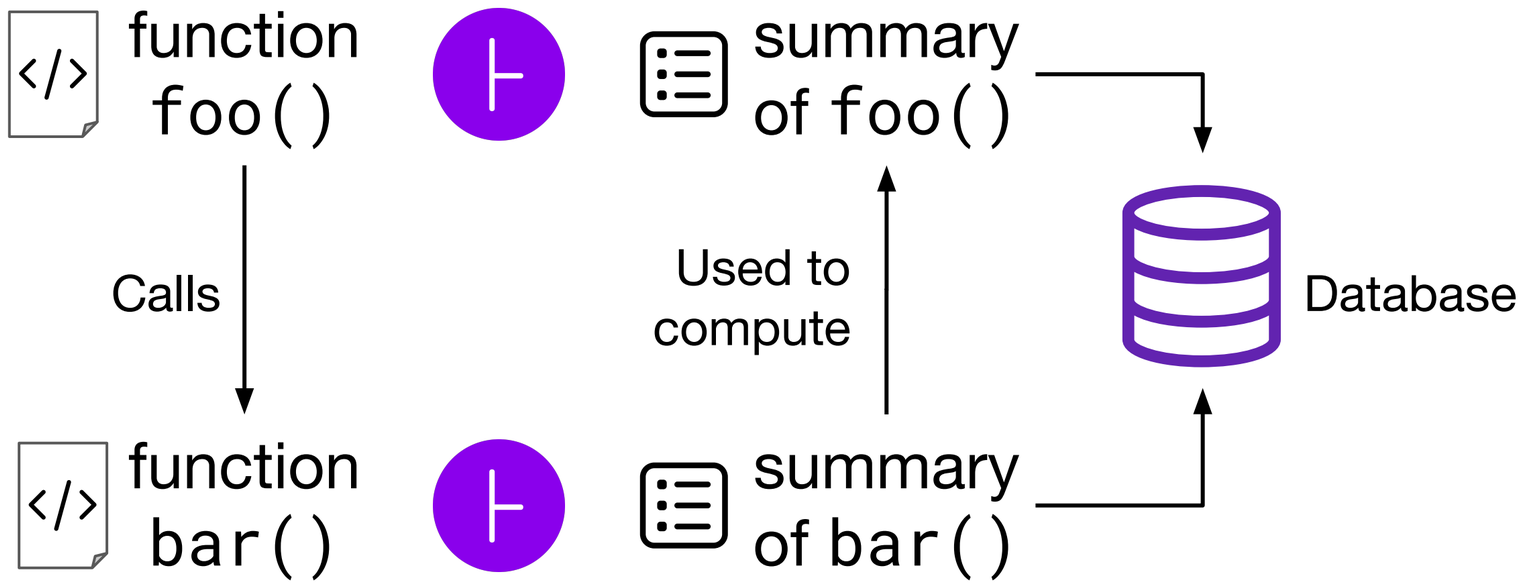
Infer checkers

Modular: analyze one procedure at a time

Abstract interpretation: propagate state, obtain summary

Compositional: summary can be used in all calling contexts

On-demand: analyze dependencies as needed



Annotation reachability - Analysis

Abstract state at F: set of (G, N, H, A) tuples
Just one step towards sink

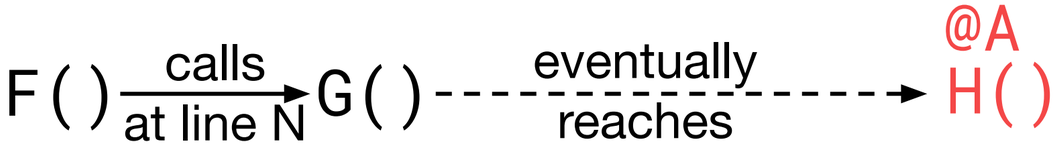
Initial state: empty

Traverse instructions in-order

Transfer function: call to some G
If G is sink → add entry for G (via G)
If G can reach H → add entry for H (via G)
Check for sanitizer

Join: set join

Summary: state at exit



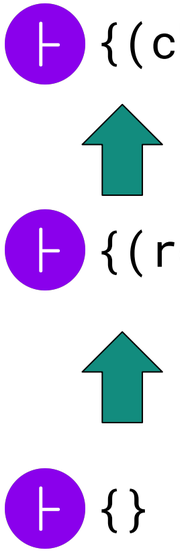
Flow- and path-insensitive

Overapproximate
E.g. calls guarded by infeasible conditions

Underapproximate
E.g. dynamic dispatch, lambdas

Annotation reachability - Analysis

```
1  @PerfCrit
2  void somethingOnTheUI() {
3   checkSomething();
4  }
5
6  void checkSomething() {
7   readFromDatabase();
8  }
9
10 @Expensive
11 void readFromDatabase() {
12  /* Slow stuff */
13 }
```



Report an issue: source function has sink in summary

(somethingOnTheUI, @PerfCrit, readFromDatabase, @Expensive)

Path: reconstruct from summary recursively

somethingOnTheUI() → checkSomething() → readFromDatabase()

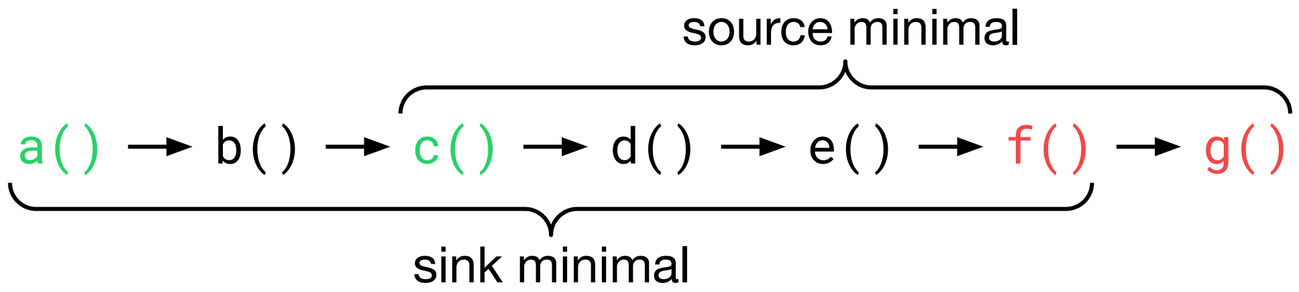
Annotation reachability - extensions

Regular expressions

External code, compactness, other languages

```
"annotation-reachability-custom-models": {
  "Expensive": ["com\.library\.SomeClass\..*"]
}
```

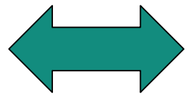
Path minimization



Loop highlighting

Rely on SCC computation

```
void source() {
 sink();
}
```



```
void source() {
 for (...) sink();
}
```

Reachability analysis for WhatsApp Android

Specification

Collaborate with WhatsApp app health team: 2 properties, 8 annotations, 11 regexps

#1
Sources: performance critical (e.g. UI event handlers)
Sinks: computationally expensive (e.g. worker thread, file IO)
Sanitizers: not shipped in production (tests, debug utils)

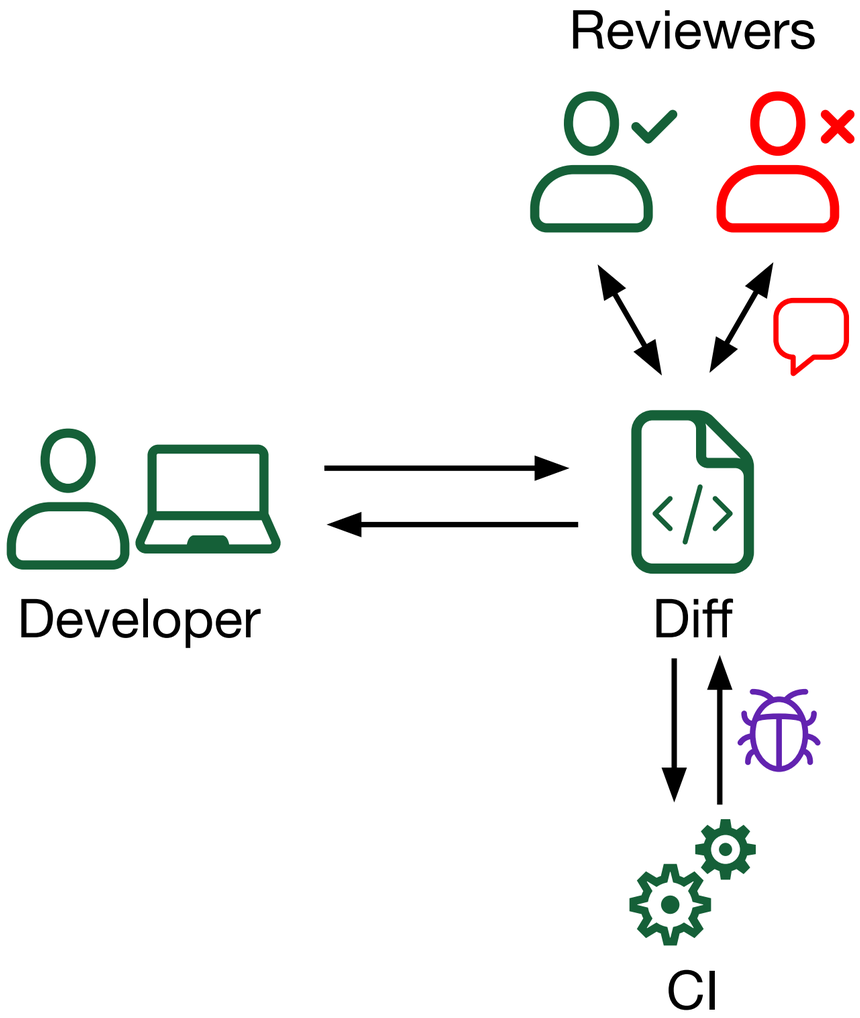
#2 Incompatible threading annotations (worker thread calling main)

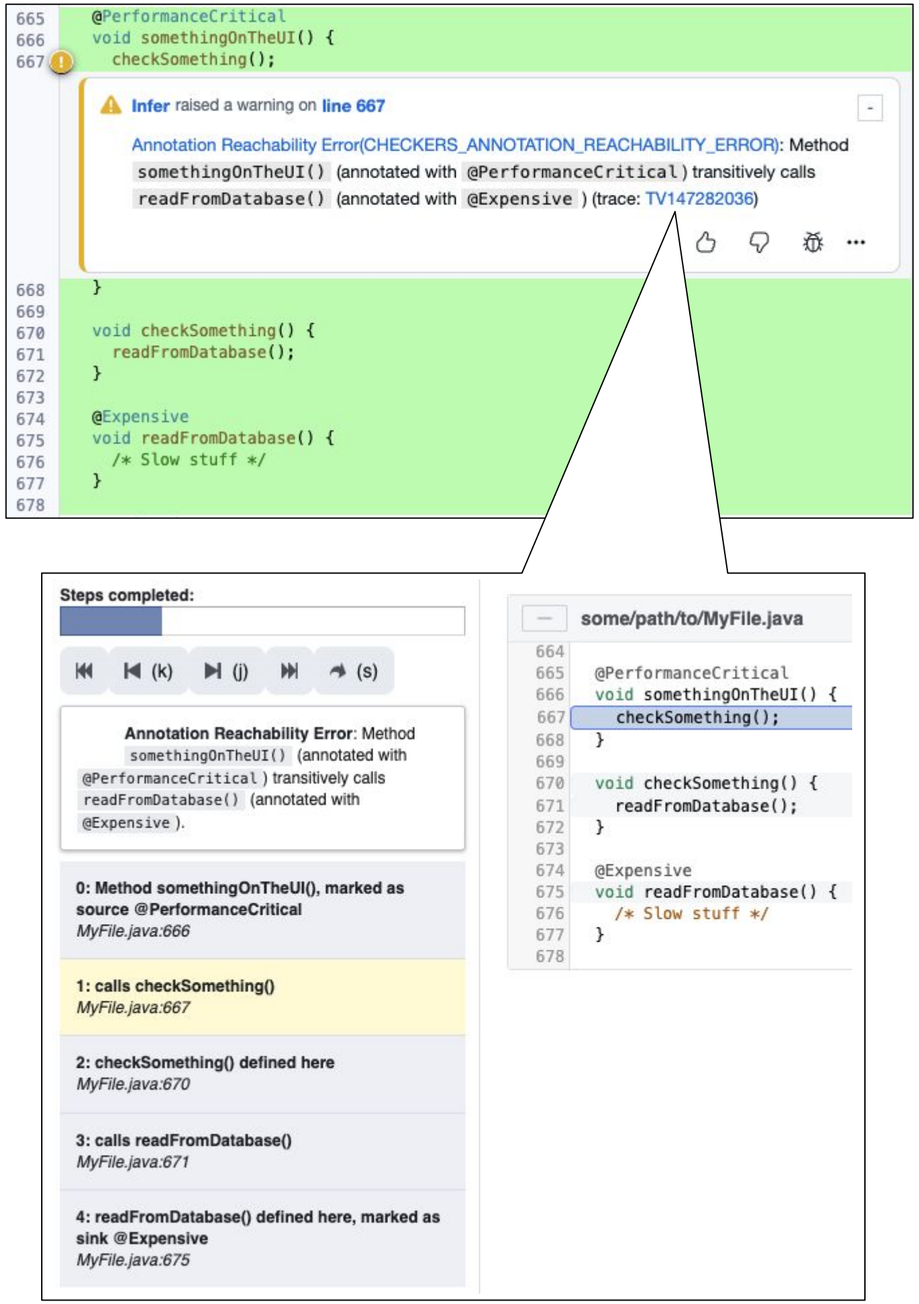
WA Android function coverage

Prop.	Sources	Sinks	Sanitizers
#1	0.356%	2.689%	0.198%
#2	16.267%	0.002%	0.000%

Deployment

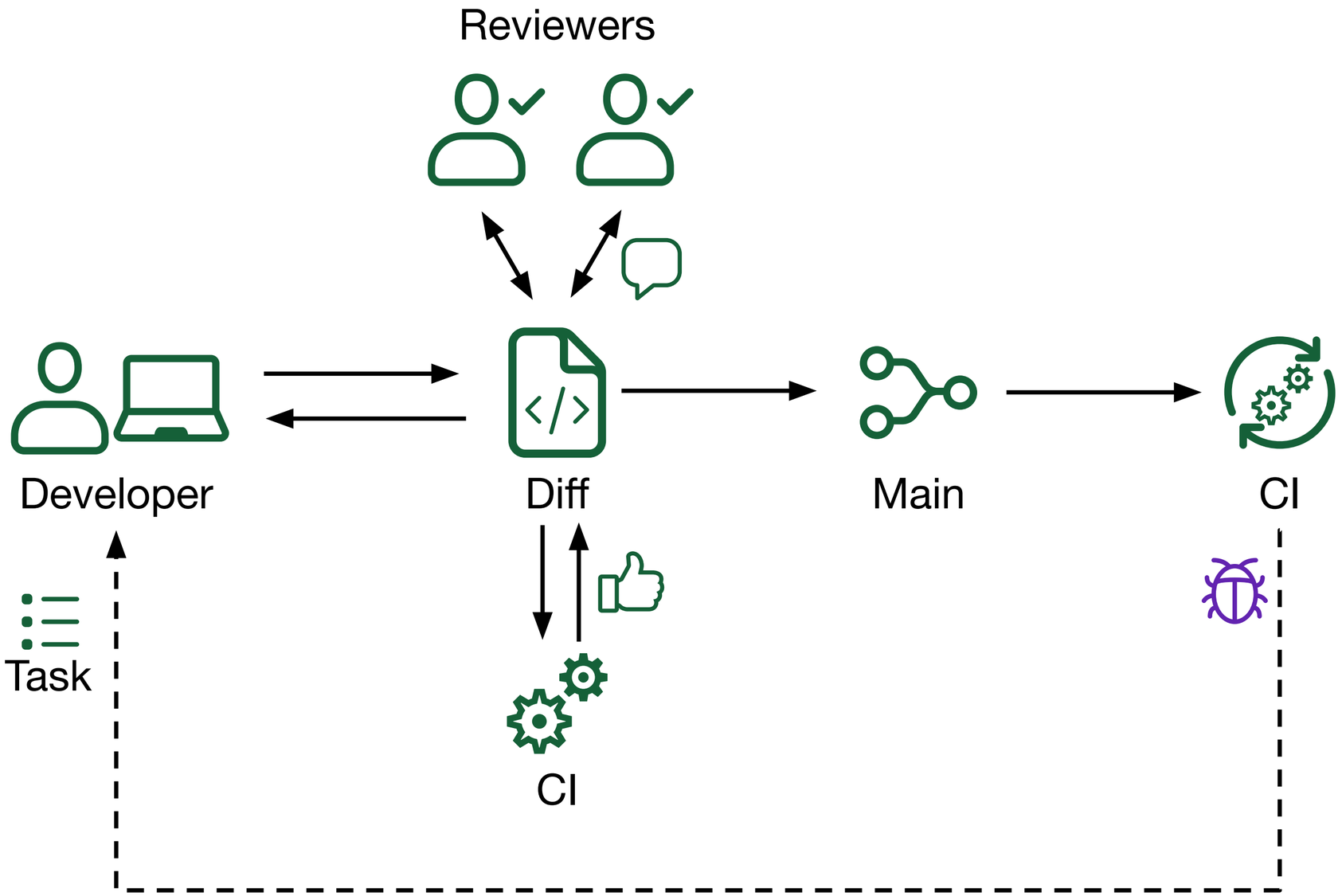
Diff-time





Deployment

Continuous



Deployment

Pre-existing issues

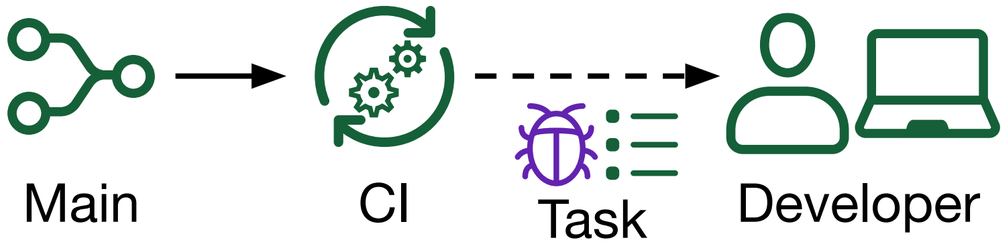
- Quality
- Volume

	No sink min.	Sink min.
No source min.	12 500	1 600
Source min.	10 600	1 100

Diffs: shadow mode → early adopters → full rollout

Results

Pre-existing



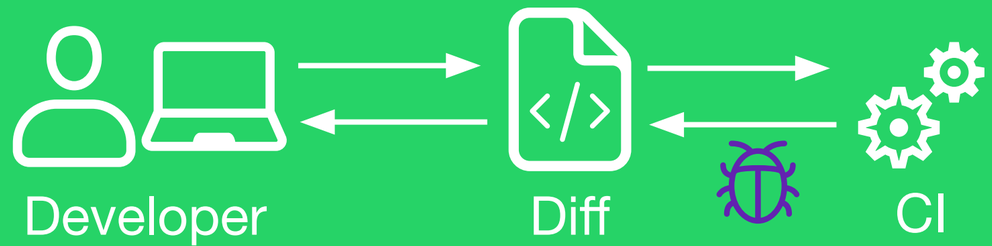
59 tasks filed, 7 fixed

Example:

reduced ANRs by 0.56%

1.25% chat loading speedup globally

Diffs



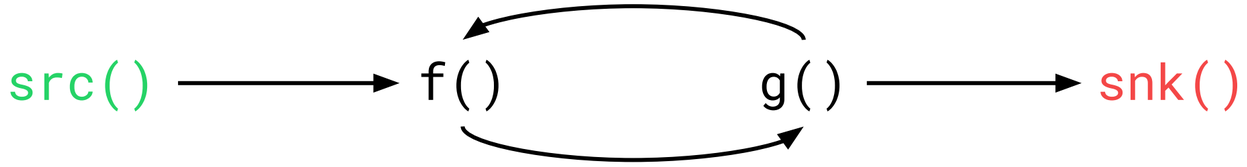
3 months

174 reported, 92 fixed

53% fixrate

Unfixed

Adding new sources and sinks



Kotlin conversion

Mutual recursion & scheduling

```
void source() { // Needs path-sensitivity
 if (is_debug()) sink();
}
```

Flow- and path-insensitivity

```
void source() { // Needs flow-sensitivity
 beginSanitizing();
 sink();
 endSanitizing();
}
```

Performance

Hard to measure: caching, multiple checkers, parallelism

Continuous (p90 execution time)

33 mins
Compilation

53 mins
Analysis (all checkers)

15 mins
Reachability only

Diffs (p90 execution time)

32 mins
Total time: parent+current, all checkers, changed files (and deps) only

Takeaway

Fast enough to run multiple times a day in continuous mode
Provide timely feedback on diffs
Reachability is not the bottleneck

Summary

Callgraph reachability for WhatsApp Android app health via Infer

3 months, prevented 92 regressions + 7 pre-existing fixes, end-user measurable impact