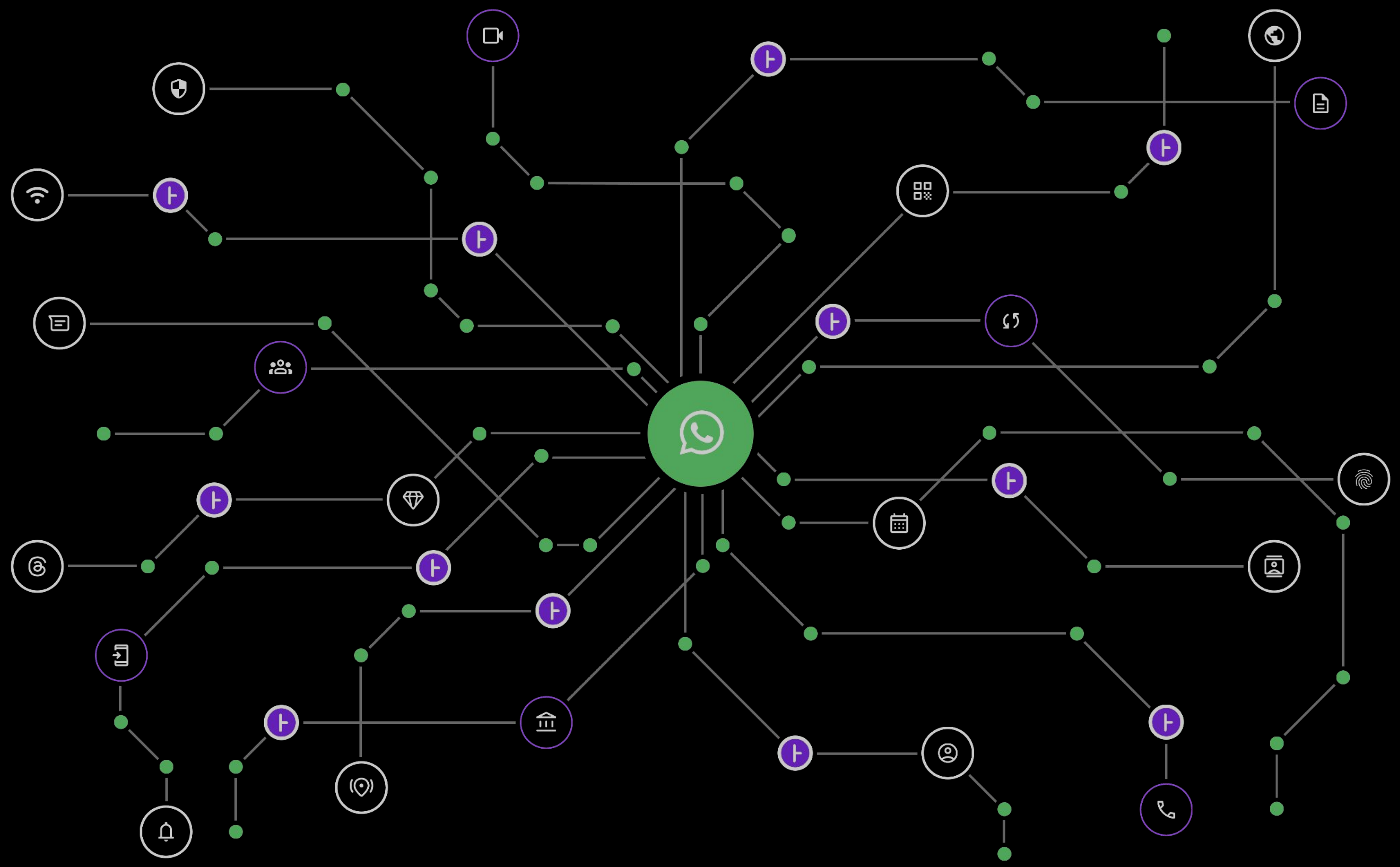
SOAP@PLDI 2025

Compositional Static Callgraph Reachability Analysis for WhatsApp Android App Health



Ákos Hajdu
Meta

Roman Lee
Meta

Gavin Weng
Meta

Nilesh Agrawal
Meta

Jérémy Dubreil

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?

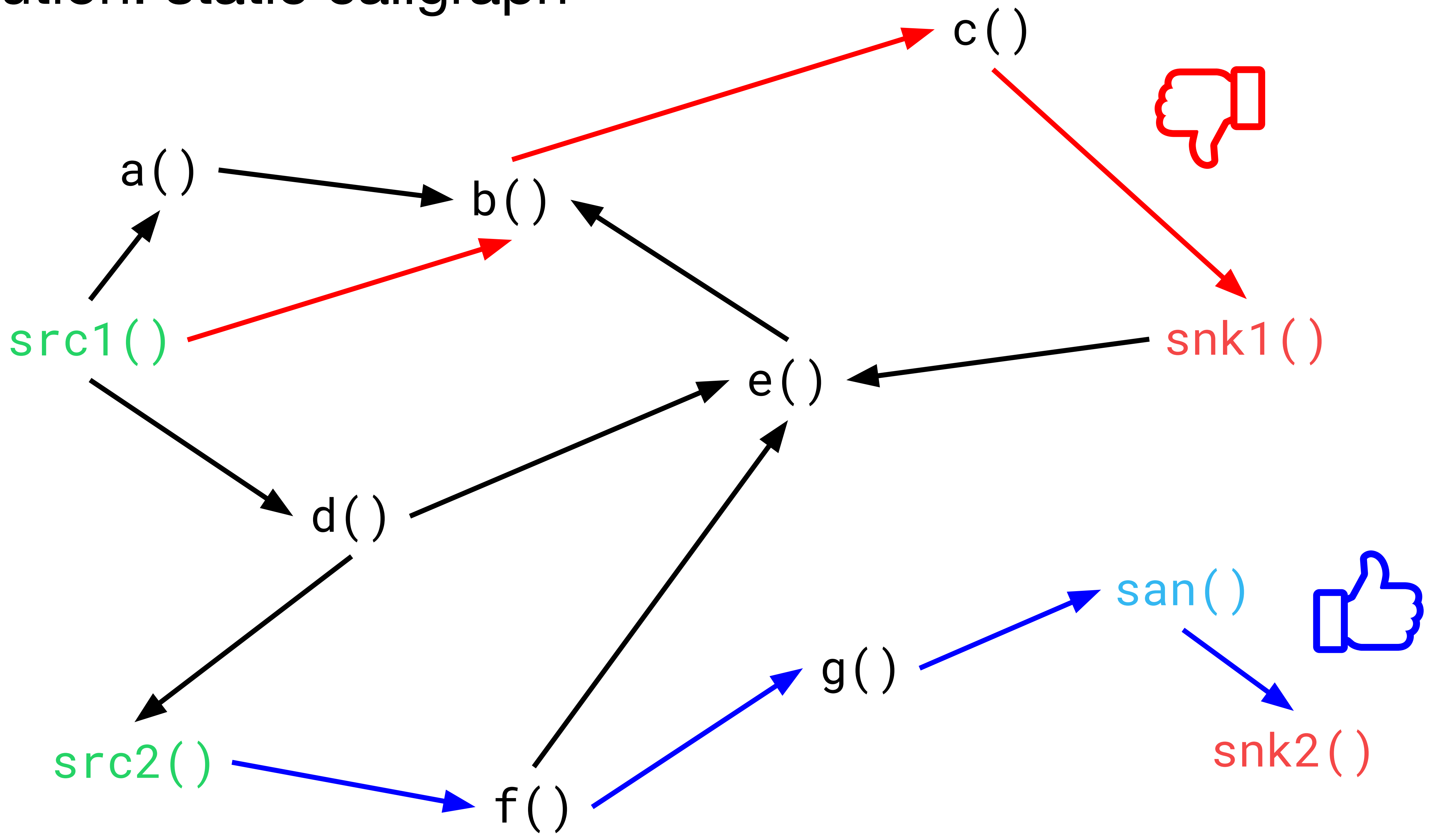
Analyzer

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

Developed at Meta - fbinfer.com

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 in code

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

void checkSomething() {
 readFromDatabase();
}

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

Infer config

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

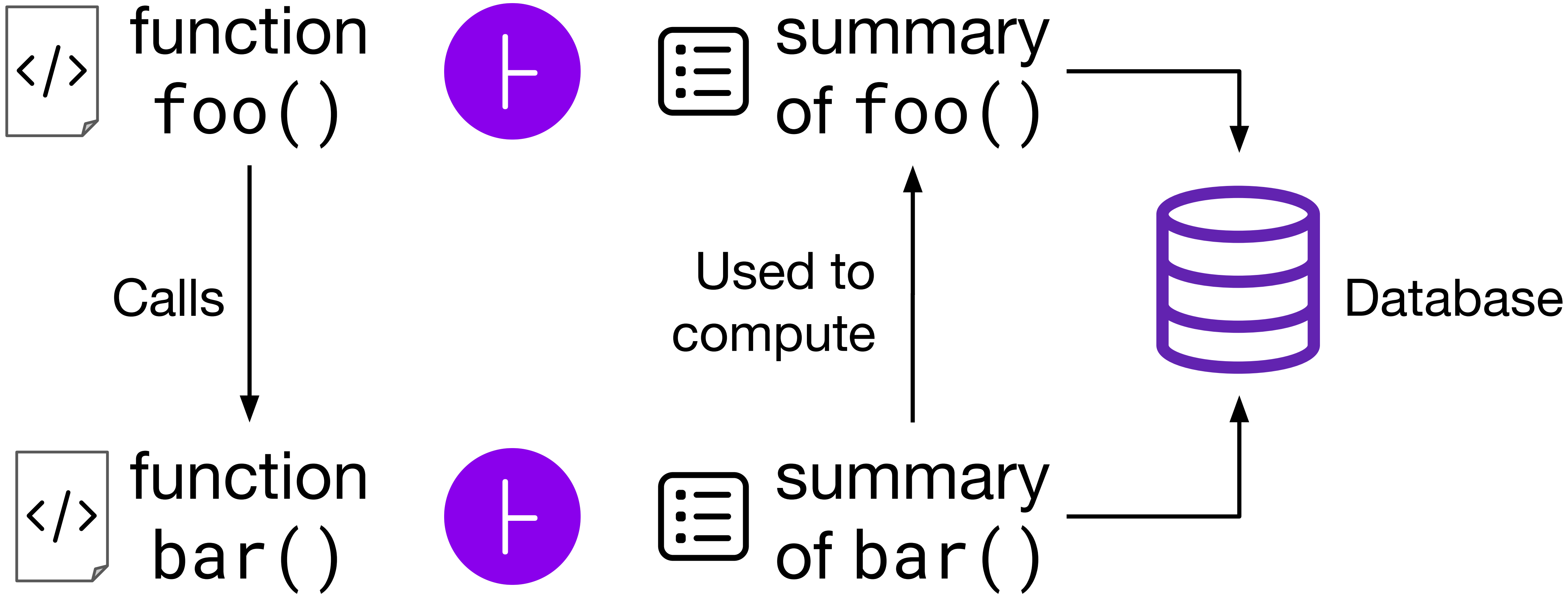
Infer checkers

Modular: analyze one procedure at a time

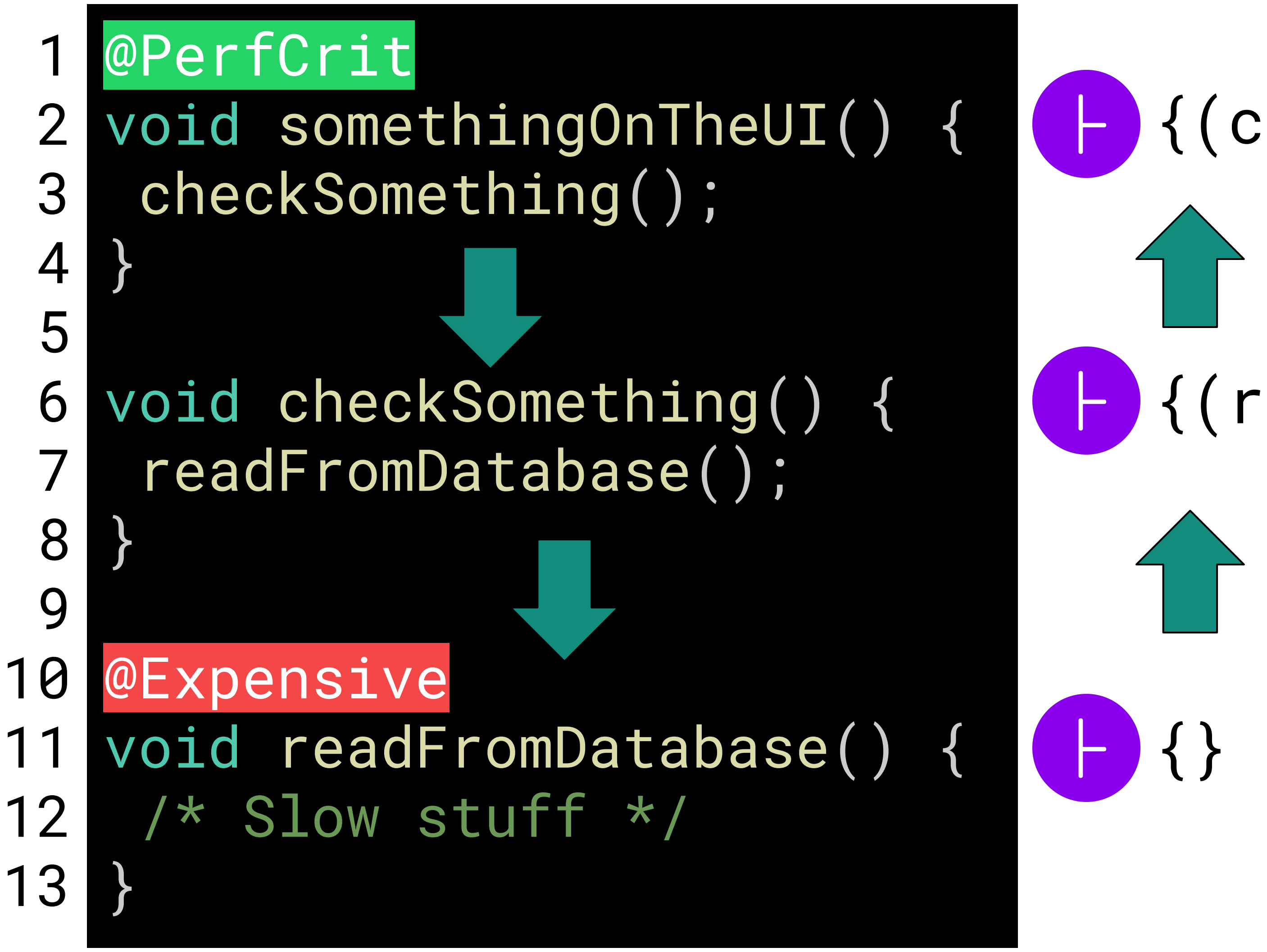
Abstract interpretation: propagate state, obtain summary

Compositional: summary can be used in all calling contexts

On-demand: analyze dependencies as needed



Annotation reachability - Analysis



```
 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 }
```

⊢ {(checkSomething, 3, readFromDatabase, @Expensive)}

⊢ {(readFromDatabase, 7, readFromDatabase, @Expensive)}

⊢ {}

Report an issue: source function has sink in summary

`(somethingOnTheUI, @PerfCrit, readFromDatabase, @Expensive)`

Path: reconstruct from summary recursively

`somethingOnTheUI() → checkSomething() → readFromDatabase()`

Annotation reachability - Analysis

Abstract state in F: set of (G, N, H, A) tuples

- Just one step towards sink

Initial state: empty

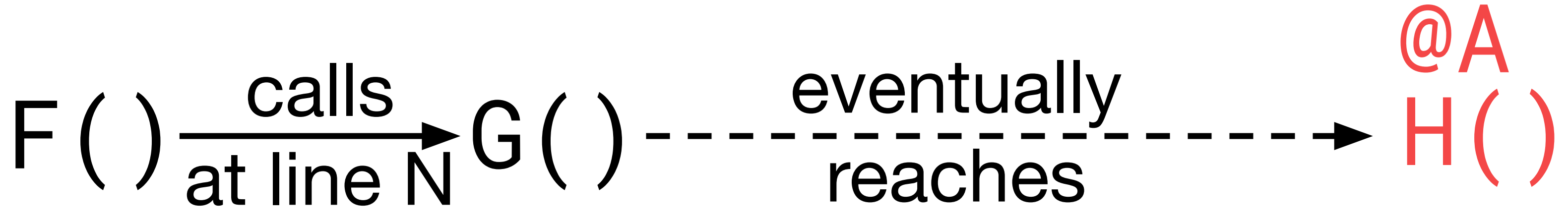
Traverse instructions in-order

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



F() calls at line N G() eventually reaches @A H()

Flow- and path-insensitive

Overapproximate

- E.g. calls guarded by infeasible conditions

Underapproximate

- E.g. dynamic dispatch, lambdas

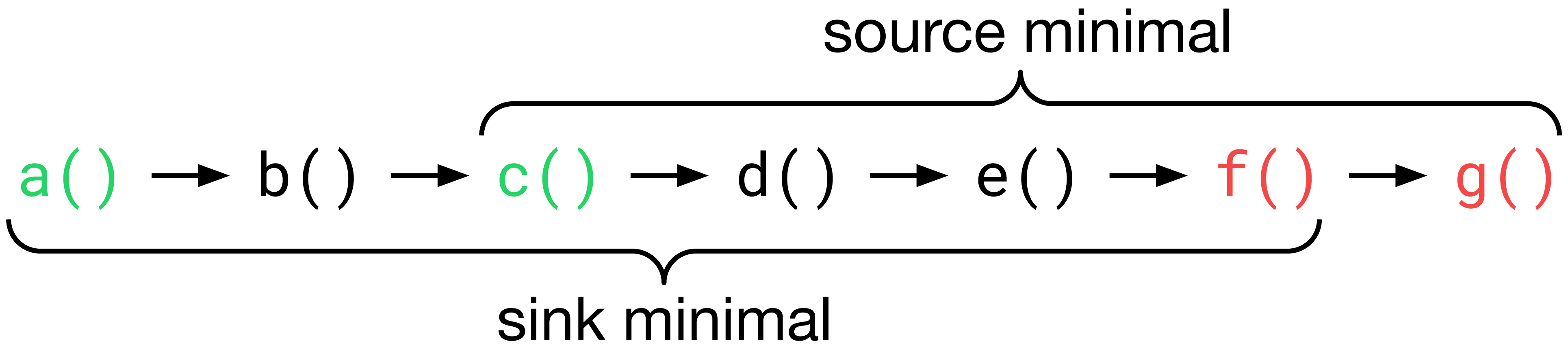
Annotation reachability - extensions

Regular expressions

- External code
- Compactness
- Other languages

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

Path minimization



Reachability analysis for WhatsApp Android

Specification

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

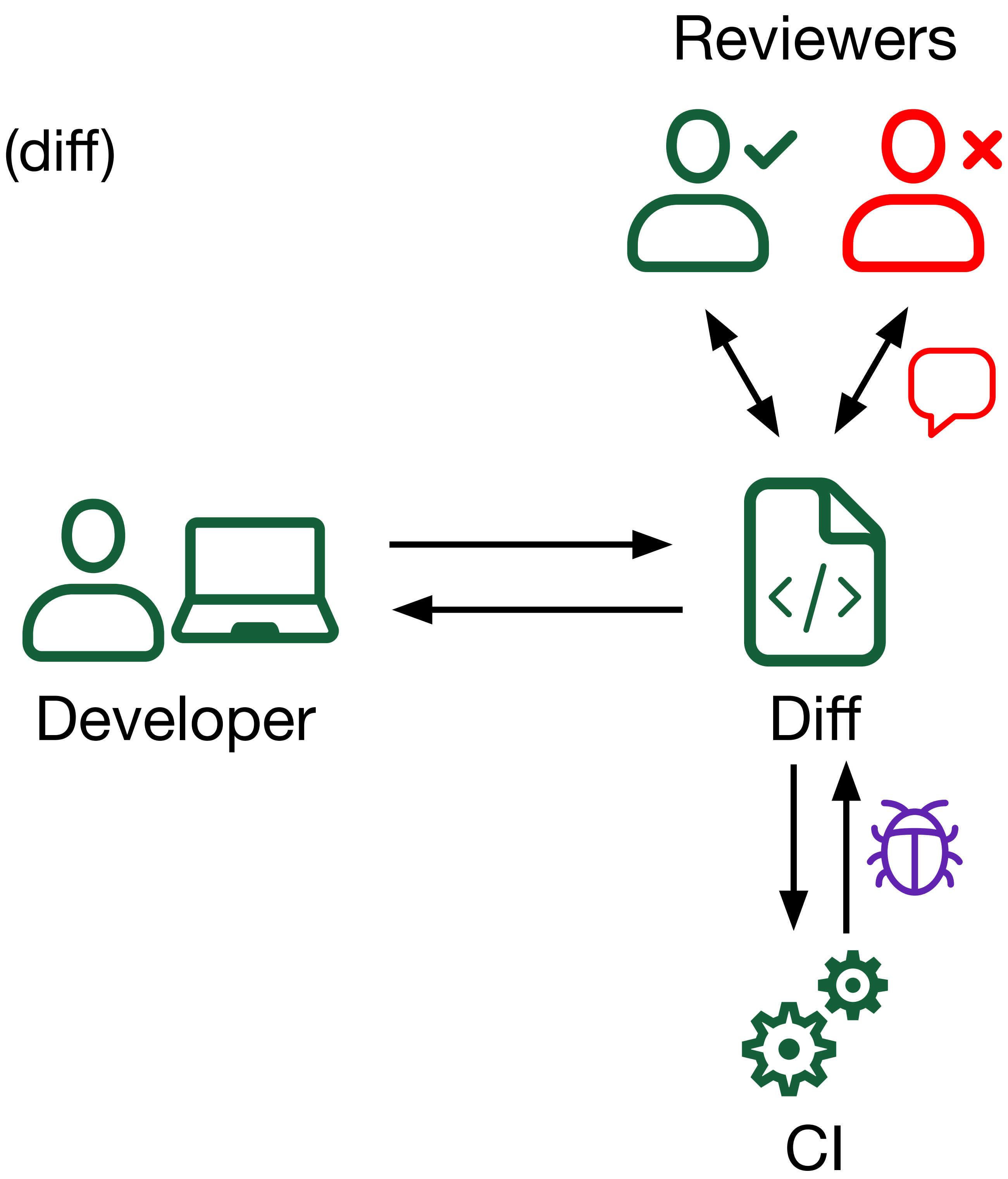
#1 Sources: performance critical (e.g. UI event handlers)
Sinks: computationally expensive (e.g. worker thread, file IO)
Sanitizers: not shipped in production (tests, debug utils)

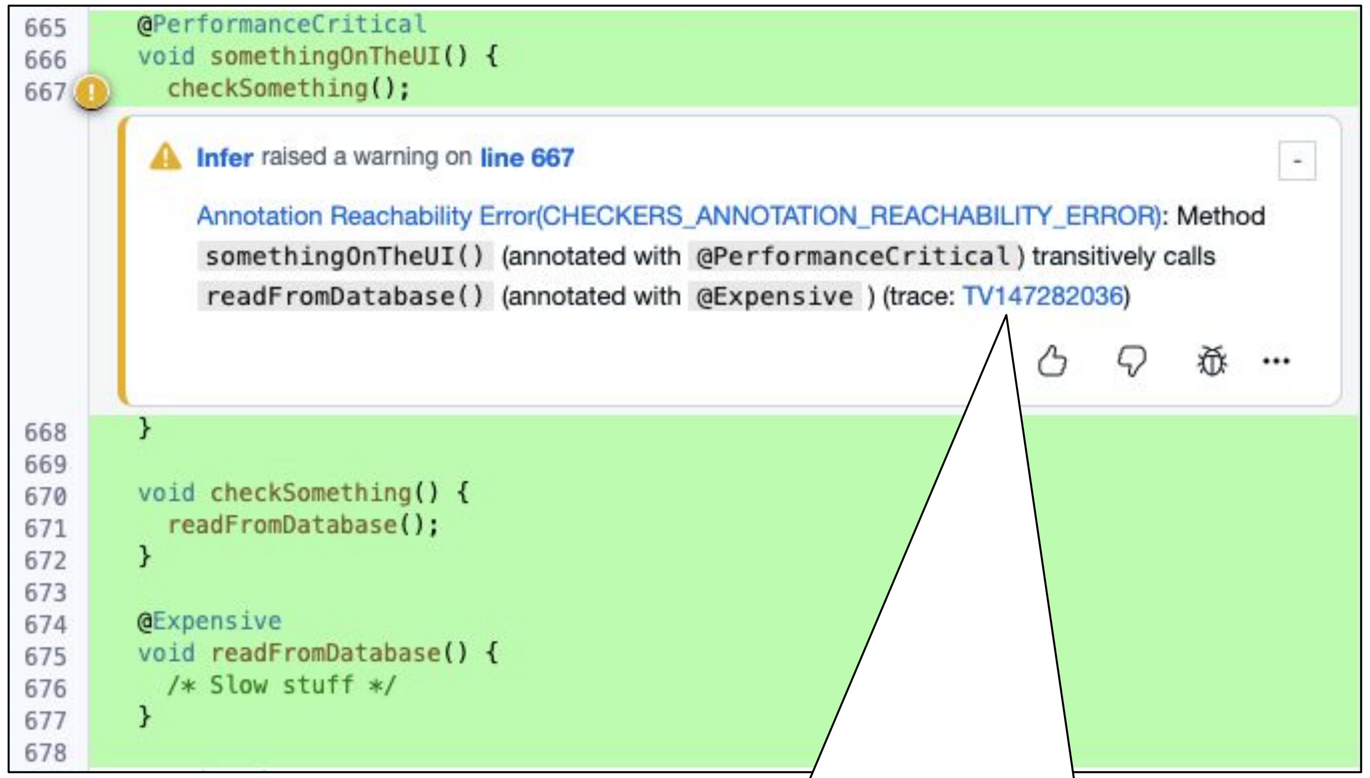
#2 Incompatible threading annotations (worker thread calling main)

	Annotations			Regexps			Coverage		
Prop.	src	snk	san	src	snk	san	src	snk	san
#1	1	2	2	2	3	3	0.356%	2.689%	0.198%
#2	2	1	0	0	3	0	16.267%	0.002%	0.000%

Deployment

Code changes (diff)



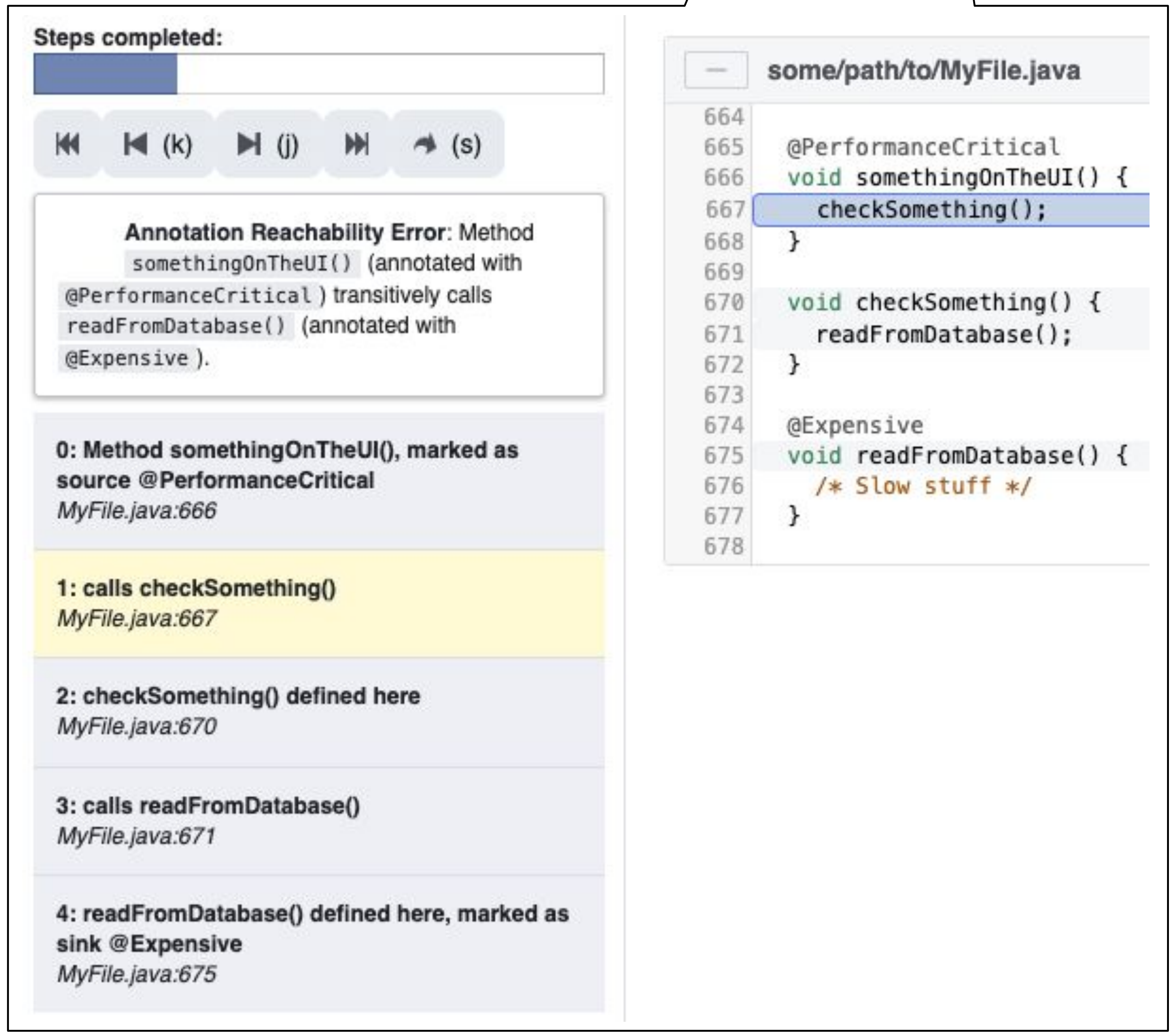


```
665   @PerformanceCritical
666   void somethingOnTheUI() {
667     checkSomething();
```

⚠ **Infer** raised a warning on **line 667**

Annotation Reachability Error(CHECKERS_ANNOTATION_REACHABILITY_ERROR): Method `somethingOnTheUI()` (annotated with `@PerformanceCritical`) transitively calls `readFromDatabase()` (annotated with `@Expensive`) (trace: TV147282036)

```
668   }
669
670   void checkSomething() {
671     readFromDatabase();
672   }
673
674   @Expensive
675   void readFromDatabase() {
676     /* Slow stuff */
677   }
678
```



Steps completed:

Annotation Reachability Error: Method `somethingOnTheUI()` (annotated with `@PerformanceCritical`) transitively calls `readFromDatabase()` (annotated with `@Expensive`).

0: Method somethingOnTheUI(), marked as source @PerformanceCritical
MyFile.java:666

1: calls checkSomething()
MyFile.java:667

2: checkSomething() defined here
MyFile.java:670

3: calls readFromDatabase()
MyFile.java:671

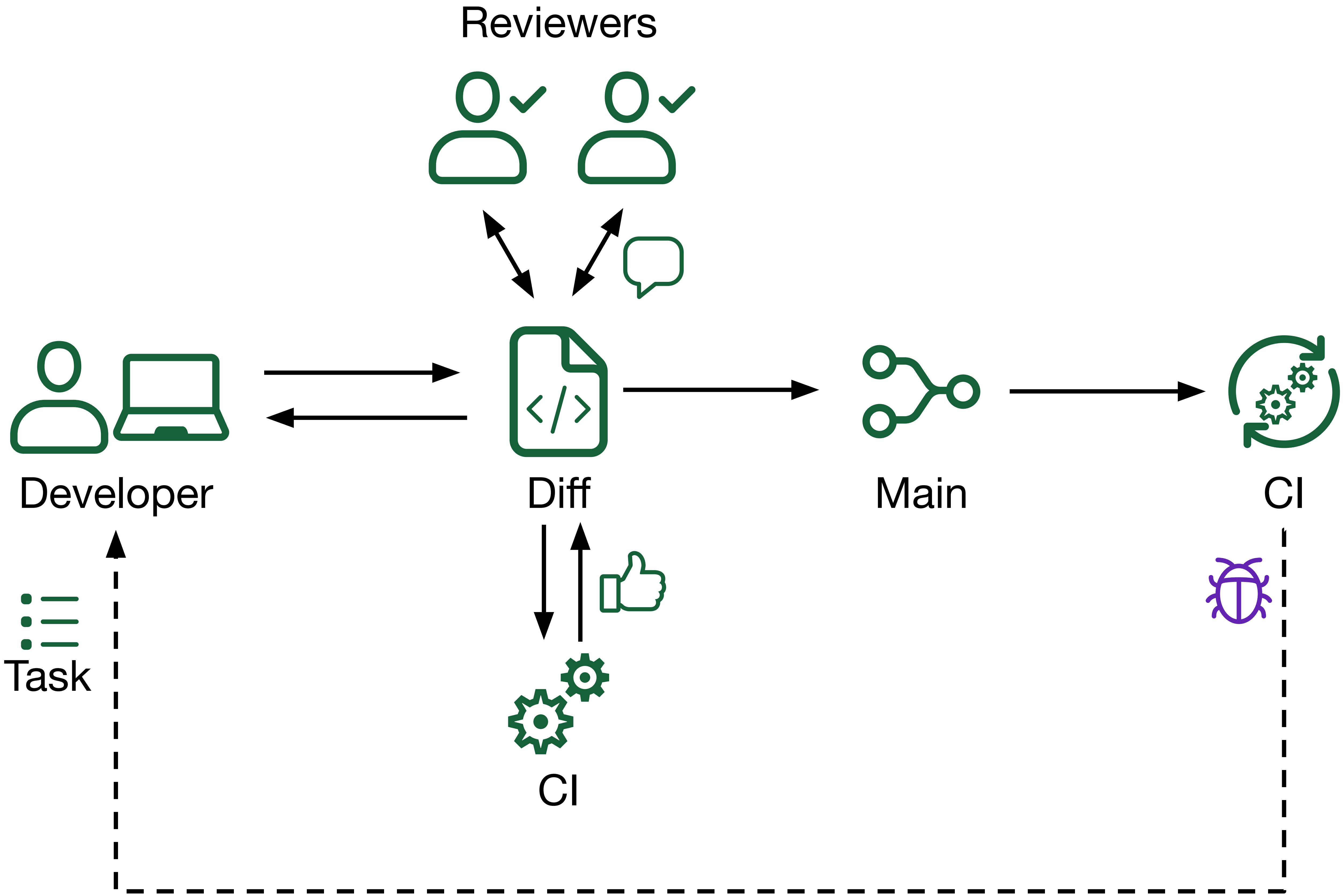
4: readFromDatabase() defined here, marked as sink @Expensive
MyFile.java:675

some/path/to/MyFile.java

```
664
665   @PerformanceCritical
666   void somethingOnTheUI() {
667     checkSomething();
668   }
669
670   void checkSomething() {
671     readFromDatabase();
672   }
673
674   @Expensive
675   void readFromDatabase() {
676     /* Slow stuff */
677   }
678
```

Deployment

Continuous



Deployment

Pre-existing issues

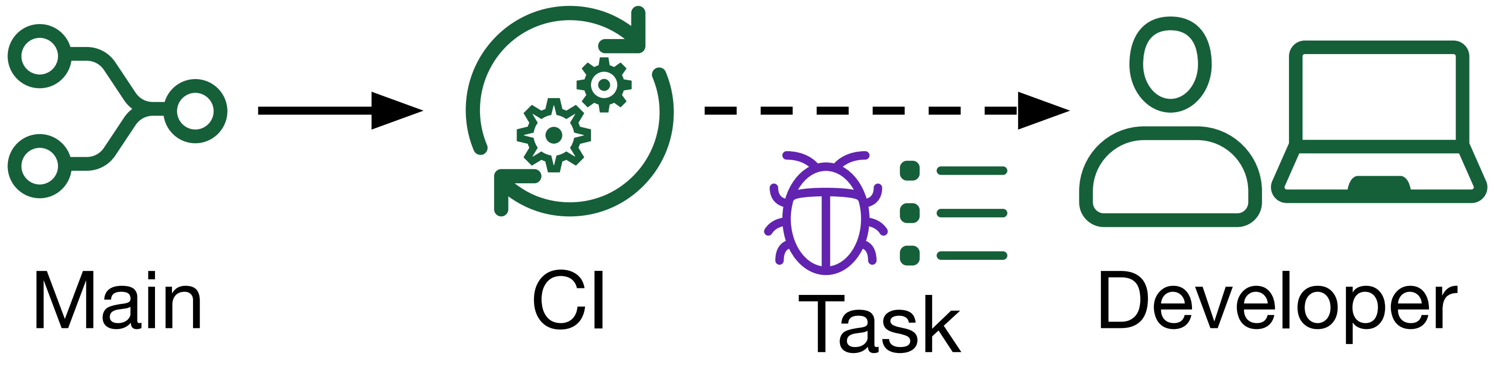
- Quality
- Volume

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

Code changes: shadow mode → early adopters → full rollout

Results

Pre-existing



Main

CI

Task

Developer

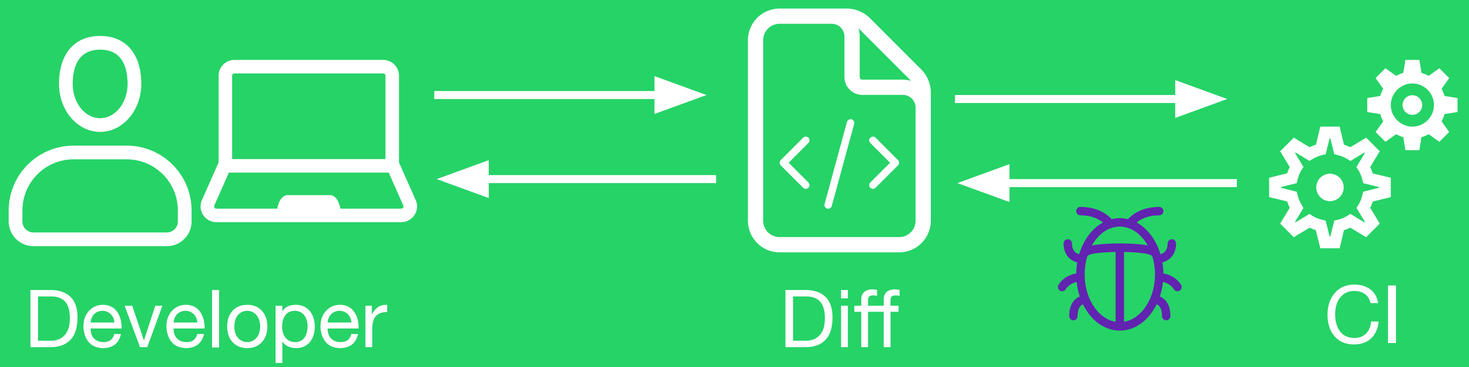
59 tasks filed, 7 fixed

Example:

reduced ANRs by 0.56%

1.25% chat loading speedup globally

Code changes



Developer

Diff

CI

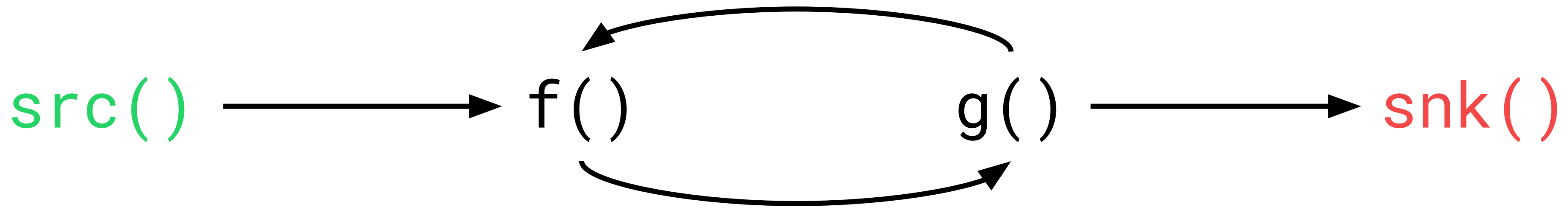
3 months

174 reported, 92 fixed

53% fixrate

Unfixed

Kotlin conversion



src() f() g() snk()

Mutual recursion & scheduling

Flow- and path-insensitivity

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

```
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

Code changes (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

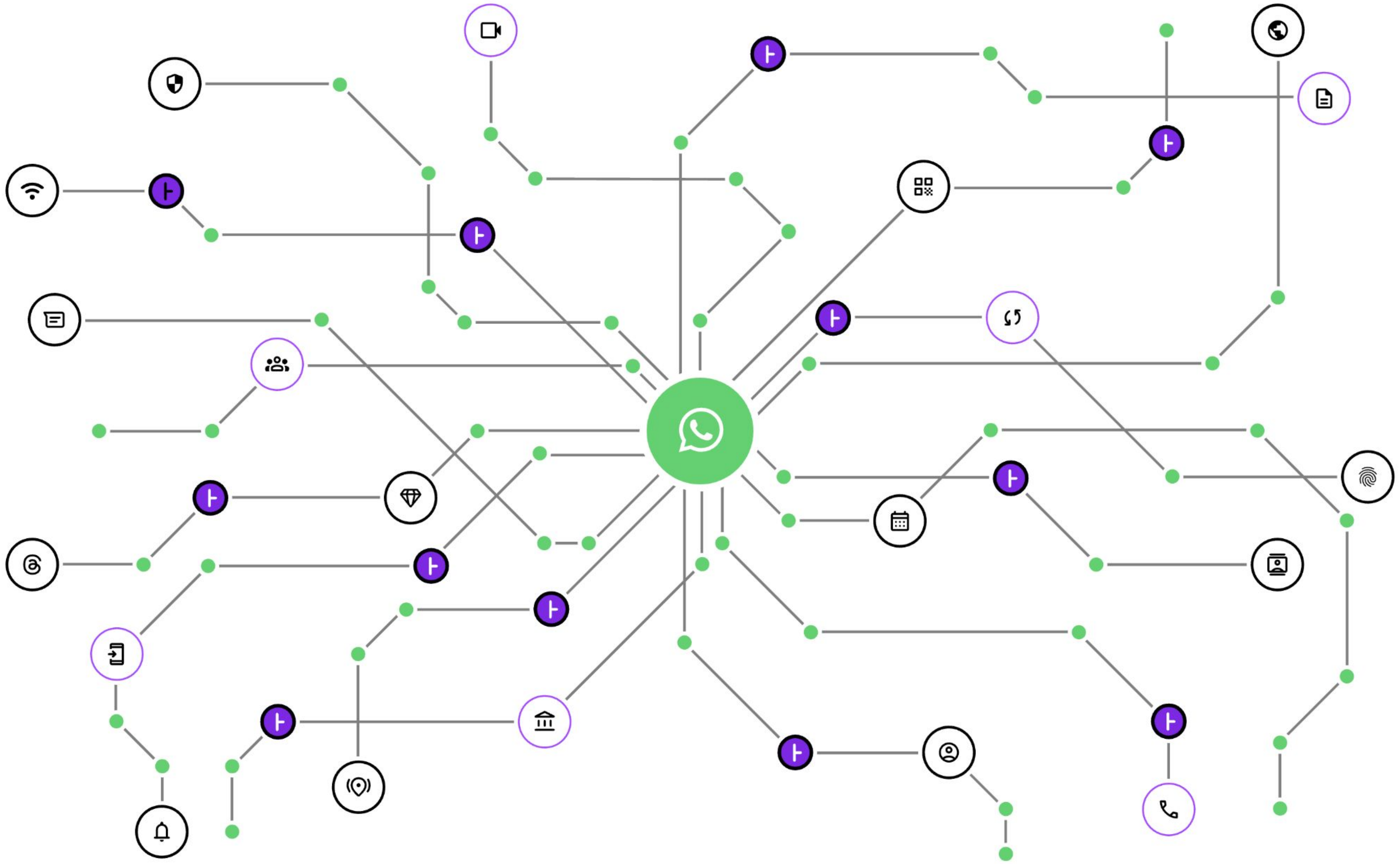
Provide timely feedback on code changes

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



hajduakos.github.io