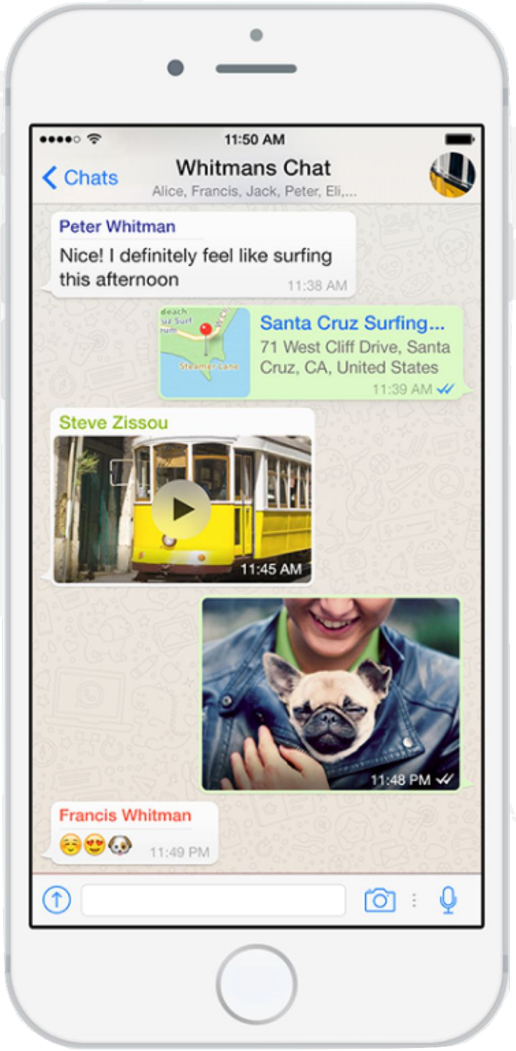
Static and dynamic code analyses for WhatsApp server

Ákos Hajdu
WhatsApp Dev Infra

2 Billion

People around the world use WhatsApp daily



100 Billion

Messages daily

Simple

So anyone can use it.

Reliable

So that messages go through no matter what.

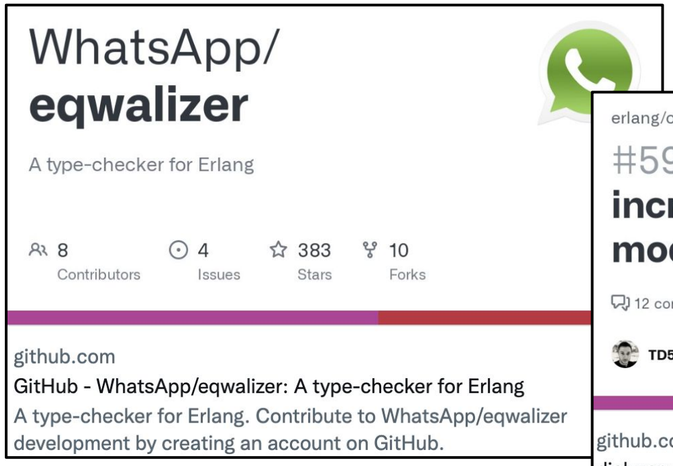
End-to-end Encrypted

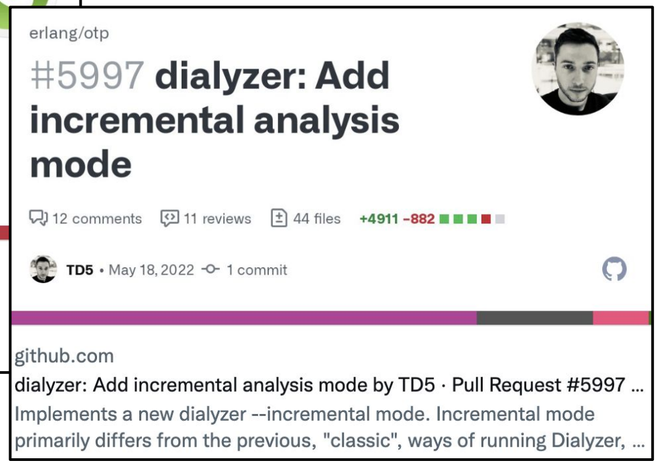
So only sender and receiver of the message can see its content.

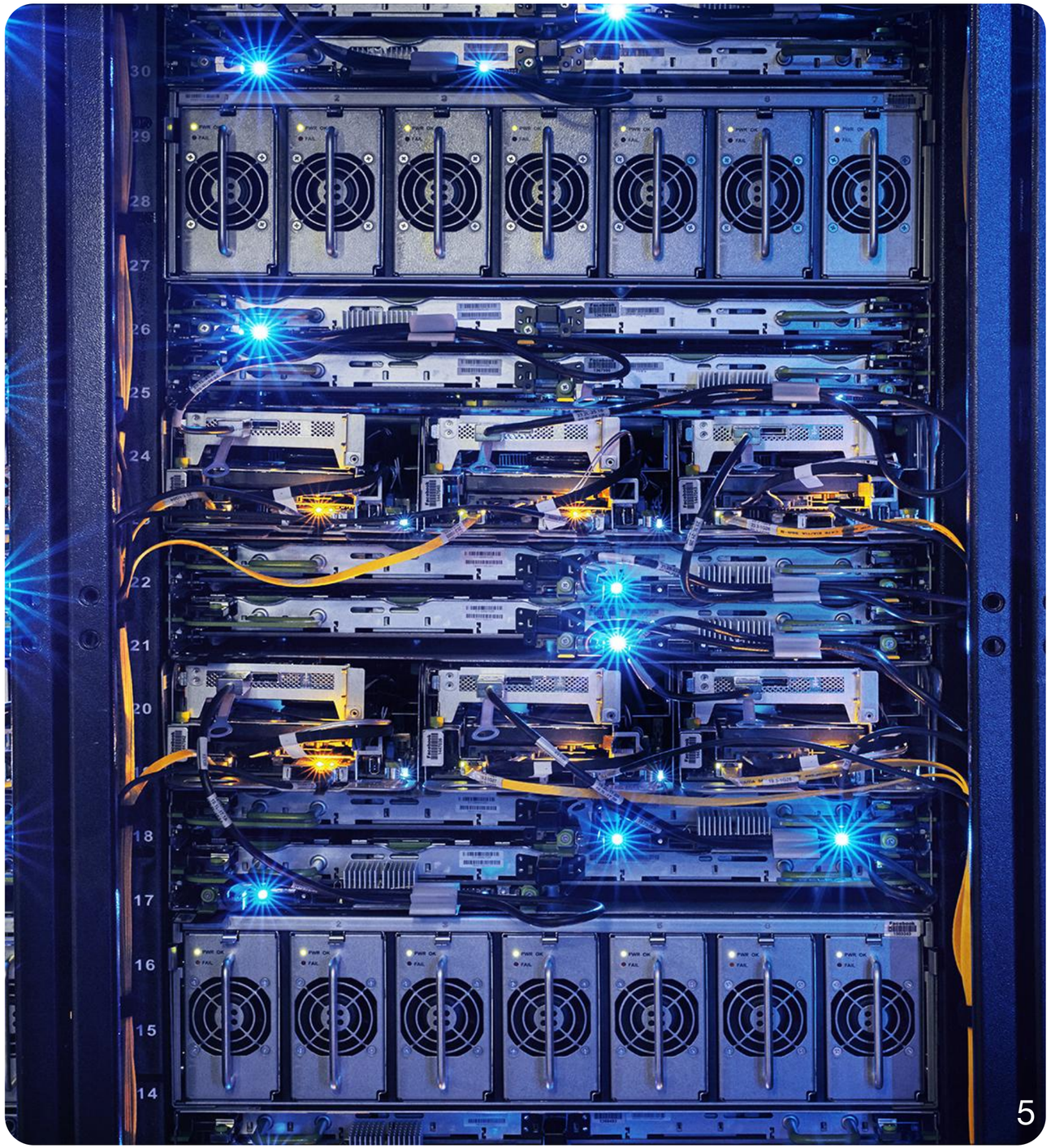
Overview

WhatsApp server

- Millions of lines of Erlang code
- Code analysis *NOT* in this talk
  - Linters
  - eqWAlizer (type checker)
    - github.com/WhatsApp/eqwalizer
  - (Incremental) Dialyzer
    - github.com/erlang/otp/pull/5997







WhatsApp server

- Millions of lines of Erlang code
- Code analysis in this talk
  - Static: InfERL
  - Dynamic: FAUSTA + Sapienz

FAUSTA: Scaling Dynamic Analysis with Traffic Generation at WhatsApp

Ke Mao
Meta
kemao@fb.com

Matteo Marescotti
Meta
mmatteo@fb.com

InfERL: Scalable and Extensible Erlang Static Analysis

Ákos Hajdu
Meta
London, UK
akoshajdu@meta.com

Matteo Marescotti
Meta
London, UK
mmatteo@meta.com

Thibault Suzanne
Meta
London, UK
tsuzanne@meta.com

Ke Mao
Meta
London, UK
kemao@meta.com

Radu Grigore
Meta
London, UK
rgrigore@meta.com

Per Gustafsson
Meta
London, UK
pergu@meta.com

Sapienz: Multi-objective Automated Testing for Android Applications

Ke Mao Mark Harman Yue Jia
CREST Centre, University College London, Malet Place, London, WC1E 6BT, UK
k.mao@cs.ucl.ac.uk, mark.harman@ucl.ac.uk, yue.jia@ucl.ac.uk

Applications

Reliability

Privacy

Performance

Static analysis

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

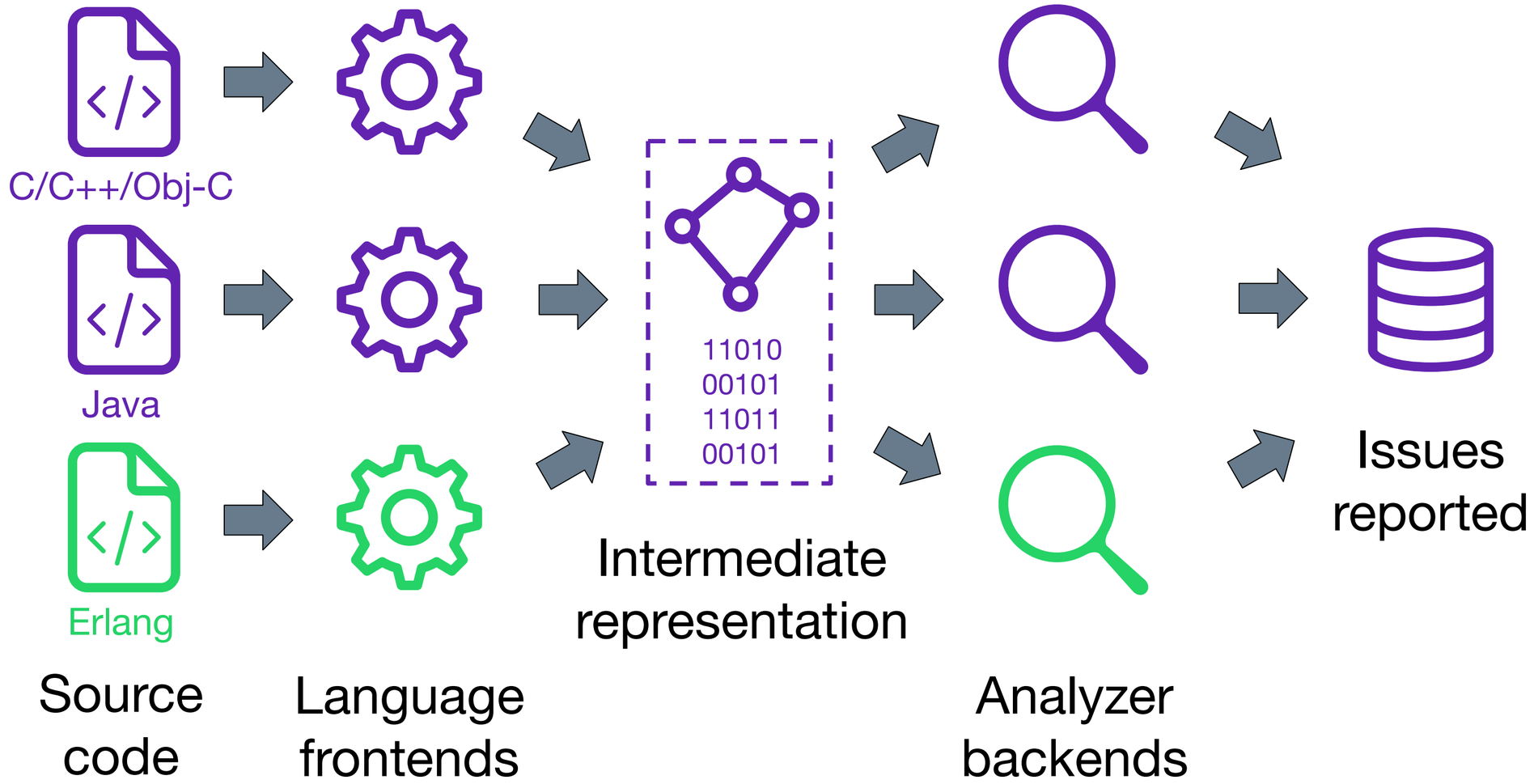
Language frontends

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

20+ analyses

Memory safety, data races, time complexity, deadlocks, temporal properties, …

Infer overview



Examples

Reliability issues

Example - Reliability

sample.erl

```erlang
-module(sample).

head([]) -> empty;
head([H|_]) -> {ok, H}.

good() ->
  {ok, 1} = head([1, 2]).

bad() ->
  empty = head([1, 2]).

bad2() ->
  {ok, 3} = head([1, 2]).

very_bad() ->
  head(123).
```

```
$ infer --pulse -- erlc sample.erl
```

```
sample.erl:3: error: No Matching Function Clause
  no matching function clause at line 3, column 1.

sample.erl:16:3: calling context starts here
15. very_bad() ->
16.   head(123).
      ^

sample.erl:16:3: in call to `head/1`
15. very_bad() ->
16.   head(123).
      ^

sample.erl:3:1: no matching function clause here
3. head([]) -> empty;
   ^

sample.erl:10: error: No Match Of Rhs
  no match of RHS at line 10, column 3.
   9. bad() ->
  10.   empty = head([1, 2]).
        ^

sample.erl:13: error: No Match Of Rhs
  no match of RHS at line 13, column 3.
  12. bad2() ->
  13.   {ok, 3} = head([1, 2]).
        ^
```

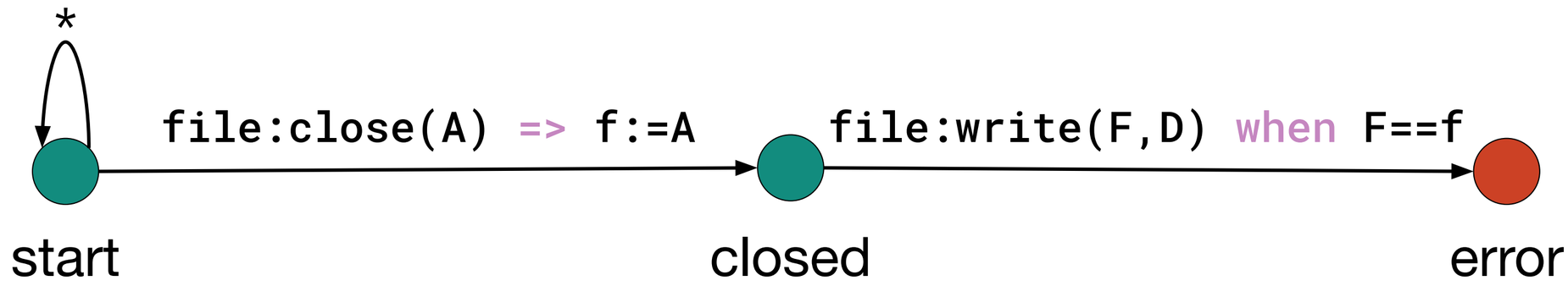
Reliability issues

```erlang
% Bad key
M = #{},
M#{2 := 3}.

% Bad map
L = [1, 2, 3],
L#{1 => 2}.

% Bad record
R = #rabbit{name="Pogi"},
R#person.name.

% No matching branch in try
tail(X) ->
  try X of
    [_ | T] -> {ok,T}
  catch
    _ -> error
  end.

% No matching case clause
tail(X) ->
  case X of
    [_ | T] -> T
  end.

% No matching function clause
tail([_ | Xs]) -> Xs.

% No match of rhs
[H | T] = [].

% No true branch in if
sign(X) ->
  if
    X > 0 -> pos;
    X < 0 -> neg
  end.
```

Examples

User-specified properties

Example: `file:write` after `file:close`



writep.topl

```
1 property WriteAfterClose
2   start -> start: *
3   start -> closed: "file:close/1"(A,Ret) => f:=A
4   closed -> error: "file:write/2"(F,D,Ret) when F==f
```

Example: `file:write` after `file:close`

Interprocedural

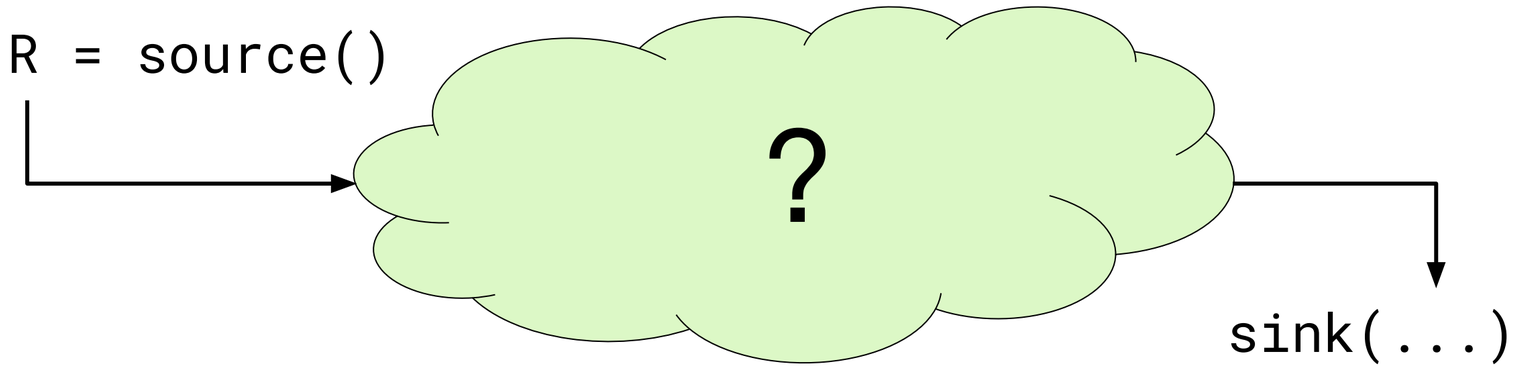
write.erl

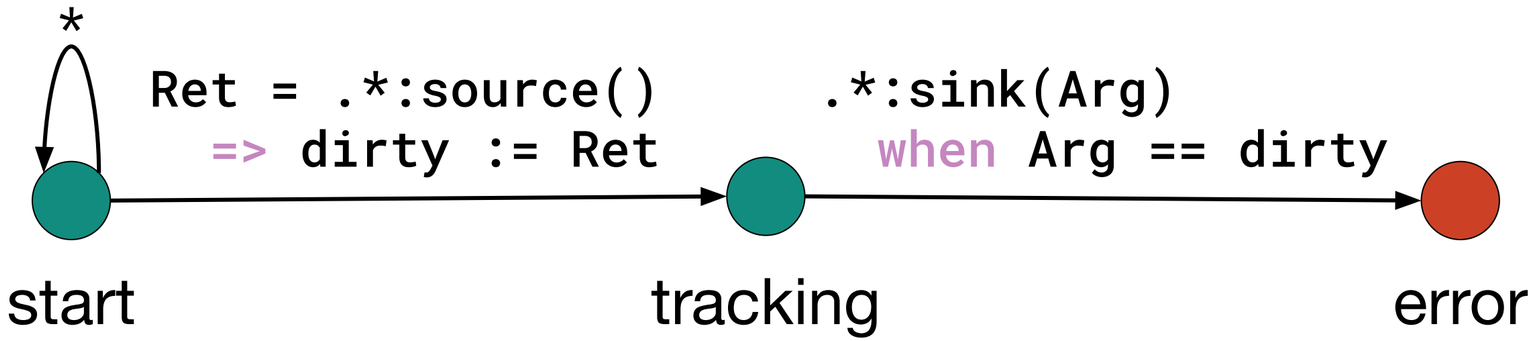
```
1 -module(write).
2
3 good(F) -> nop(F), file:write(F, "hi").
4 bad(F)  -> op(F),  file:write(F, "hi").
5 nop(_)  -> ok.
6 op(F)   -> file:close(F).
```

```
$ infer --topl-only --topl-properties writep.topl -- erlc write.erl
```

```
write.erl:4: error: Topl Error
  property WriteAfterClose reaches state error.

  3. good(F) -> nop(F), file:write(F, "hi").
  4. bad(F)  -> op(F), file:write(F, "hi").
       ^
```

Example: taint analysis

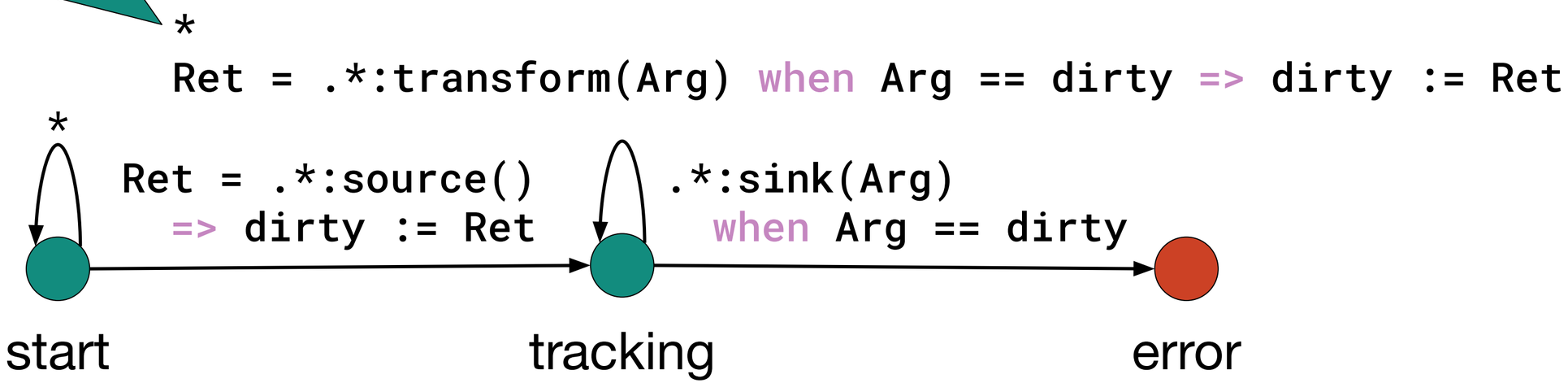




Example: taint analysis with transformations

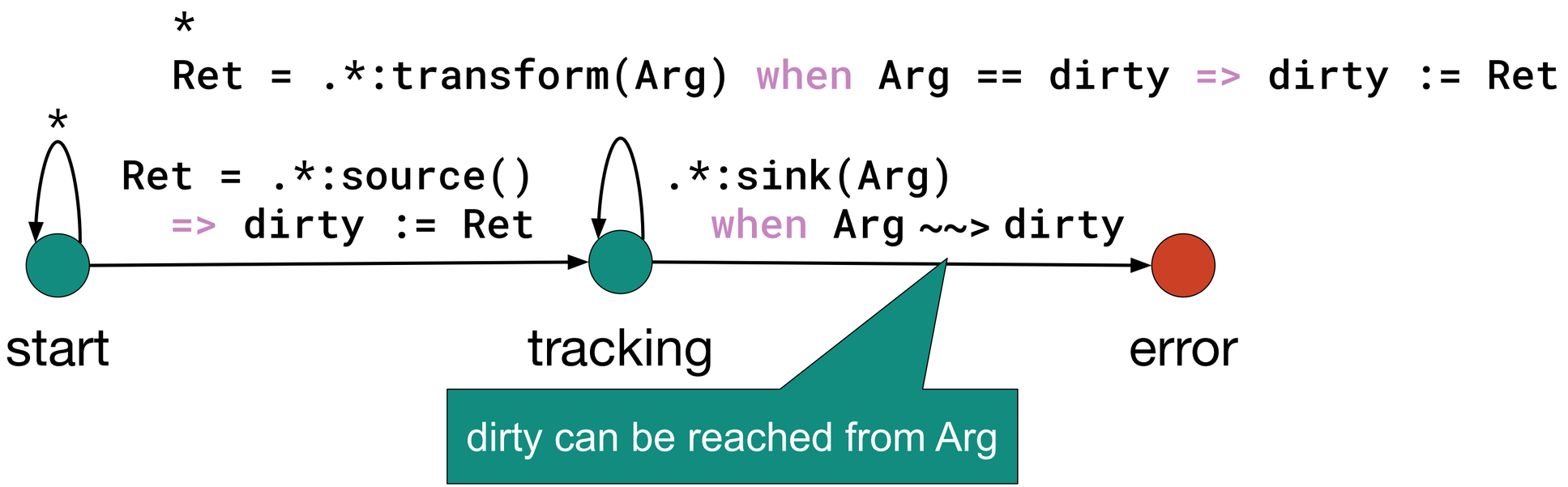
```
1 X = source(),
2 Y = transform(X),
3 sink(Y).
```

Add your functions of interest



Example: taint analysis with data structures

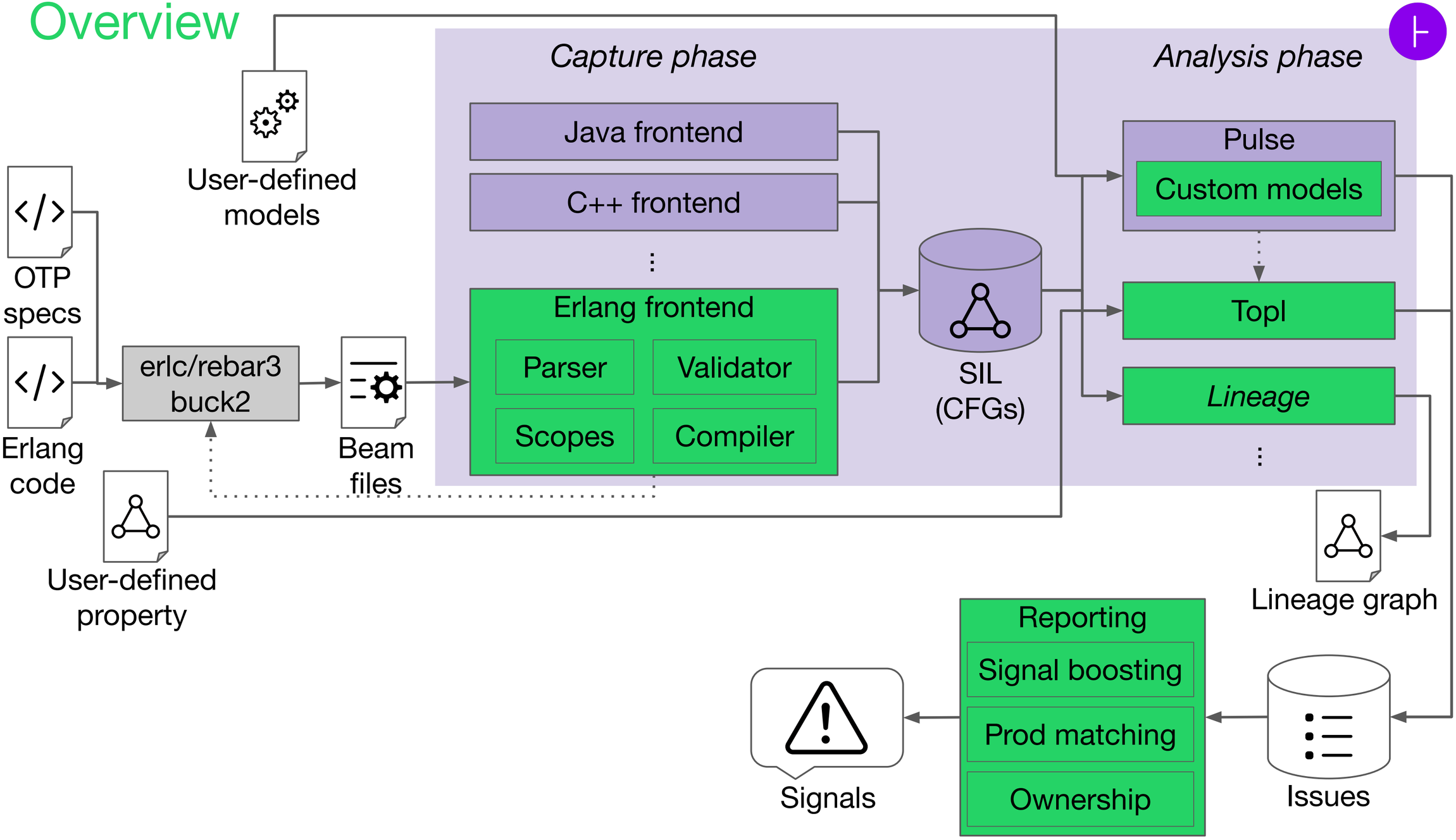
```
1 X = source()
2
3 sink(X)
4 sink([hey, 1, {"wat", X}])
```



Compilation and analysis

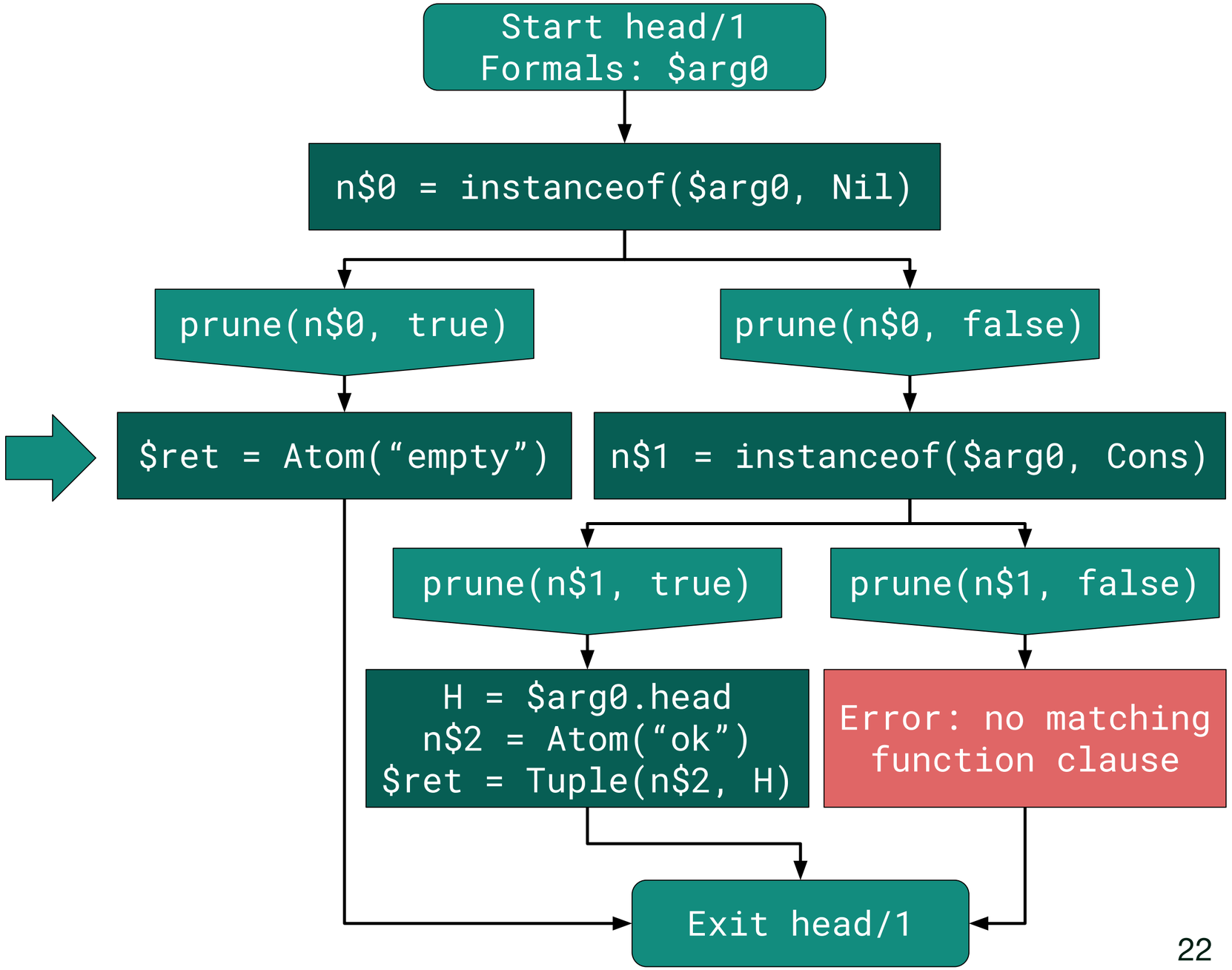
Basics

Overview


Capture phase
Analysis phase
User-defined models
OTP specs
Erlang code
erlc/rebar3 buck2
Beam files
User-defined property
Java frontend
C++ frontend
Erlang frontend
Parser
Validator
Scopes
Compiler
SIL (CFGs)
Pulse
Custom models
Topl
Lineage
Lineage graph
Reporting
Signal boosting
Prod matching
Ownership
Signals
Issues

Compilation basics

```
1 head([]) -> empty;
2 head([H|_]) -> {ok, H}.
```



Analysis basics

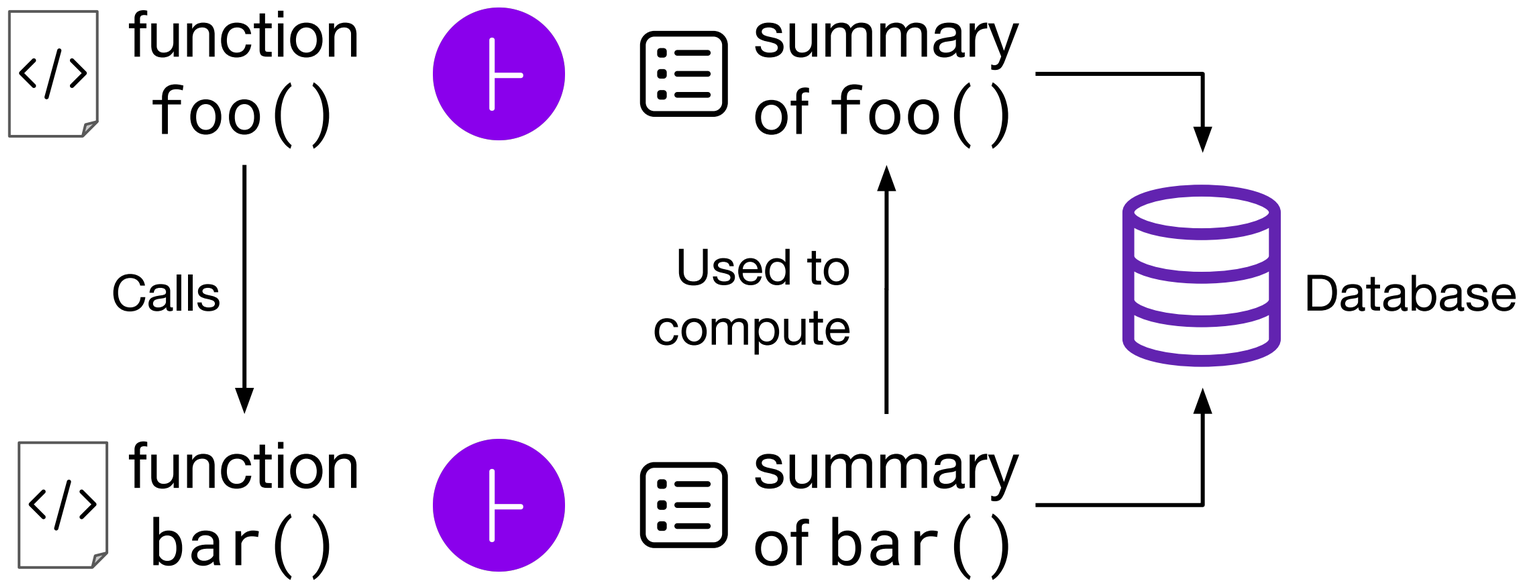
- Summaries
  - Compute once per function
  - Compact & symbolic representation of behaviors

```
1 head([]) -> empty;
2 head([H|_]) -> {ok, H}.
```

```
1. pre:  $arg0 instanceof Nil
   post: $ret = Atom(“empty”)

2. pre:  $arg0 instanceof Cons
   post: $ret = Tuple(Atom(“ok”), $arg0.head)

3. pre:  not $arg0 instanceof Nil and
         not $arg0 instanceof Cons
   post: ERROR
```

Analysis basics

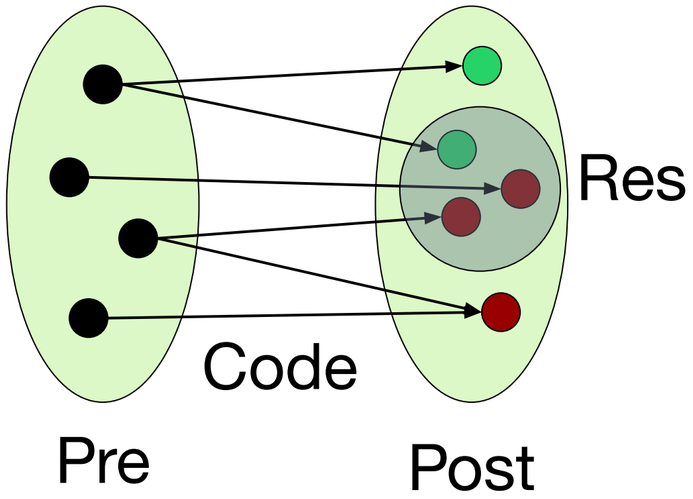
- Modular: analyze one procedure (+deps) at a time
- Compositional: summary can be used in all calling contexts



Analysis basics

- Under the hood: Pulse analyzer (+Topl)
  - Incorrectness separation logic
  - Under-approximate

[Pre] Code [ok:Res]
[Pre] Code [err:Res]



Analysis basics

Incorrectness Logic

PETER W. O'HEARN, Facebook and University College London, UK

Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about t
supported by automated testing and static analysis tools. This paper describes a s
incorrectness which is, in a sense, the other side of the coin to Hoare's logic of corre

CCS Concepts: • **Theory of computation → Programming logic**.

Additional Key Words and Phrases: Proofs, Bugs, Static Analysis

ACM Reference Format:
Peter W. O'Hearn. 2020. Incorrectness Logic. *Proc. ACM Program. Lang.* 4, POPL, A
32 pages. https://doi.org/10.114

1 INTRODUCTION

When reasoning informally
go wrong, as well as about w
if we give it a large string?"
hypothesis that reasoning a
in a principled logical syste
demonstrated to have sound
the relationship of the princ

We explore our hypothesis
logic of program correctness
than correctness. Hoare's th

{*pre-condition*}*code*{*po*

Local Reasoning about the Presence of Bugs: Incorrectness Separation Logic

Azalea Raad[1], Josh Berdine[2], Hoang-Hai Dang[1],
Derek Dreyer[1], Peter O'Hearn[2,3], and Jules Villard[2]

e Systems (MPI-SWS),
pus, Germany
College London, UK

of work on local reasoning for
r proving their *presence*. We
easoning about the presence of
lations: 1) separation logic and
eory of this new *incorrectness*
ive a begin-anywhere, intra-
hat has no false positives *by*
owards transferring modular,
am verification to bug catching.

Logic · Bug Catching

Finding Real Bugs in Big Programs with Incorrectness Logic

QUANG LOC LE, University College London and Meta, UK
AZALEA RAAD, Imperial College London and Meta, UK
JULES VILLARD, Meta, UK
JOSH BERDINE, Meta, UK
DEREK DREYER, MPI-SWS, Germany
PETER W. O'HEARN, Meta and University College London, UK

Incorrectness Logic (IL) has recently been advanced as a logical theory for compositionally proving the *presence* of bugs—dual to Hoare Logic, which is used to compositionally prove their *absence*. Though IL was motivated in large part by the aim of providing a logical foundation for bug-catching program analyses, it has remained an open question: is IL useful only retrospectively (to explain *existing* analyses), or can it actually be useful in developing *new* analyses which can catch real bugs in big programs?

In this work, we develop Pulse-X, a new, automatic program analysis for catching memory errors, based

Analysis basics

Empty under-approximates

$$[p]C[\epsilon\colon \mathit{false}]$$

Unit

$$[p]\texttt{skip}[\mathit{ok}\colon p][\mathit{er}\colon \mathit{false}]$$

Iterate zero

$$[p]C^\star[\mathit{ok}\colon p]$$

Choice (where $i = 1$ or 2)

$$\frac{[p]C_i[\epsilon\colon q]}{[p]C_1 + C_2[\epsilon\colon q]}$$

Consequence

$$\frac{p' \Leftarrow p \quad [p]C[\epsilon\colon q] \quad q \Leftarrow q'}{[p']C[\epsilon\colon q']}$$

Sequencing (short-circuit)

$$\frac{[p]C_1[\mathit{er}\colon r]}{[p]C_1; C_2[\mathit{er}\colon r]}$$

Iterate non-zero

$$\frac{[p]C^\star; C[\epsilon\colon q]}{[p]C^\star[\epsilon\colon q]}$$

Error

$$[p]\texttt{error}()[\mathit{ok}\colon \mathit{false}][\mathit{er}\colon p]$$

Disjunc

$$\frac{[p_1]C[\epsilon}{[p_1 \vee$$

Sequen

$$\frac{[p]C_1[\epsilon}{[p]C}$$

Backward

$$\frac{[p(n) \wedge n}{[p(0)}$$

Assume

$$[p]\texttt{assume}$$

$$\texttt{while}\, B\, \texttt{do}\, C \;=_{def}\; (\texttt{assume}(B);\, C)^\star; \texttt{assume}(\neg$$

$$\texttt{if}\, B\, \texttt{then}\, C\, \texttt{else}\, C' \;=_{def}\; (\texttt{assume}(B);\, C) + (\texttt{assume}$$

$$\texttt{assert}(B) \;=_{def}\; \texttt{assume}(B) + (\texttt{assume}(\neg B)$$

Assignment

$$[p]x = e[\mathit{ok}\colon \exists x'.p[x'/x] \wedge x = e[x'/x]][\mathit{er}\colon \mathit{false}]$$

Nondet Assignment

$$[p]x = \texttt{nondet}()[\mathit{ok}\colon \exists x' p][\mathit{er}\colon \mathit{false}]$$

Constancy

$$\frac{[p]C[\epsilon\colon q]}{[p \wedge f]C[\epsilon\colon q \wedge f]} \quad \mathit{Mod}(C) \cap \mathit{Free}(f) = \emptyset$$

Local Variable

$$\frac{[p]C(y/x)[\epsilon\colon q]}{[p]\texttt{local}\, x.C[\epsilon\colon \exists y.q]} \quad y \notin \mathit{Free}(p, C)$$

Substitution I

$$\frac{[p]C[\epsilon\colon q]}{([p]C[\epsilon\colon q])(e/x)} \quad \Big(\mathit{Free}(e) \cup \{x\}\Big) \cap \mathit{Free}(C) = \emptyset$$

Substitution II

$$\frac{[p]C[\epsilon\colon q]}{([p]C[\epsilon\colon q])(y/x)} \quad y \notin \mathit{Free}(p, C, q)$$

Generic Semantics for arbitrary state sets Σ

$$[\![C]\!]\epsilon \subseteq \Sigma \times \Sigma$$
$$[\![B]\!] : \Sigma \to \mathit{Bool}$$

$$\begin{aligned}
[\![\texttt{skip}]\!]ok &= \{(\sigma,\sigma) \mid \sigma \in \Sigma\} & [\![\texttt{skip}]\!]er &= \emptyset \\
[\![\texttt{error}()]\!]ok &= \emptyset & [\![\texttt{error}()]\!]er &= \{(\sigma,\sigma) \mid \sigma \in \Sigma\} \\
[\![\texttt{assume}\ B]\!]ok &= \{(\sigma,\sigma) \mid [\![B]\!]\sigma = \texttt{true}\} & [\![\texttt{assume}\ B]\!]er &= \emptyset \\
[\![C^\star]\!]\epsilon &= \textstyle\bigcup_{i \in Nat} [\![C^{\,i}]\!]\epsilon & [\![C_1 + C_2]\!]\epsilon &= [\![C_1]\!]\epsilon \cup [\![C_2]\!]\epsilon \\
[\![C_1; C_2]\!]\epsilon &= \{(\sigma_1,\sigma_3) \mid \exists \sigma_2.(\sigma_1,\sigma_2) \in [\![C_1]\!]ok \text{ and } (\sigma_2,\sigma_3) \in [\![C_2]\!]\epsilon\} \\
&\quad \cup \Big(\mathit{if}\ (\epsilon = ok)\ \mathit{then}\ \emptyset\ \mathit{else}\ \{(\sigma_1,\sigma_2) \mid (\sigma_1,\sigma_2) \in [\![C_1]\!]er\}\Big)
\end{aligned}$$

Semantics of mutation and local variables

$$\Sigma = \mathit{Variables} \to \mathit{Values}$$
$$[\![e]\!] : \Sigma \to \mathit{Values}$$

$$\begin{aligned}
[\![x = e]\!]ok &= \{(\sigma, (\sigma \mid x \mapsto [\![e]\!]\sigma)) \mid \sigma \in \Sigma\} & [\![x = e]\!]er &= \emptyset \\
[\![x = \texttt{nondet}()]\!]ok &= \{(\sigma, (\sigma \mid x \mapsto v)) \mid \sigma \in \Sigma, v \in \mathit{Values}\} & [\![x = \texttt{nondet}()]\!]er &= \emptyset \\
[\![\texttt{local}\, x.C]\!]\epsilon &= \{((\sigma \mid x \mapsto v), (\sigma' \mid x \mapsto v)) \mid (\sigma,\sigma') \in [\![C]\!]\epsilon, v \in \mathit{Values}\}
\end{aligned}$$

Compilation and analysis

Challenges

Dynamic types & pattern matching

- Avoid false positives due to potential non-exhaustive pattern matching

```
1 -spec len(list()) -> integer().
2 len([]) -> 0;
3 len([_ | T]) -> 1 + len(T).
```

- Return value of unknown functions (mostly from OTP)

```
-spec sort(list()) -> list().
```

```
1 min(Xs) ->
2     case lists:sort(Xs) of
3       [] -> error;
4       [X|_] -> {ok, X}
5     end.
```

https://www.erlang.org/doc/reference_manual/typespec.html

Unbounded data structures

- *(Bounded full support: tuples, records)*
- Lists: approximate, loop unrolling bound
- Maps: recency abstraction

```erlang
-spec ok(map()) -> any().
ok(M) ->
   case maps:is_key(key, M) of
       true -> maps:get(key, M);
       _ -> nope
   end.
```

```erlang
-spec bad(map()) -> any().
bad(M) ->
   maps:get(key, M).
```

Closures, higher-order functions, recursion

- Currently
  - Limited support
  - Found compiler bug in scoping
    - github.com/erlang/otp/issues/5379

```
1 (X=1)
2  +
3 (begin F=(fun () -> X=2 end), F() end)
```

- Plans
  - Unknown closures
    - Specialization*
    - Defunctionalization**
  - Recursion
    - Fixed-point computation
    - Custom models (for standard library)

```
1 g(F) -> F().
2
3 f() -> 1.
4
5 main() -> g(fun f/0).
```

```
1 g_f() -> f().
2
3
4
5 main_f() -> g_f().
```

*Corentin De Souza: Higher-order function specialization in Infer
Infer @ PLDI 2022.
**John C Reynolds. 1972. Definitional interpreters for higher-order programming languages. Proc. of the ACM annual conf. Vol. 2. 717–740.

Concurrency - send/receive

- Currently
  - send: no-op
  - receive: nondet.

```
1 -module(concurrent).
2 -export([f/0]).
3
4 f() ->
5   receive
6     {From, X} when is_integer(X) -> From ! X + 1;
7     {From, _} -> From ! oops
8   end,
9   f().
```

send

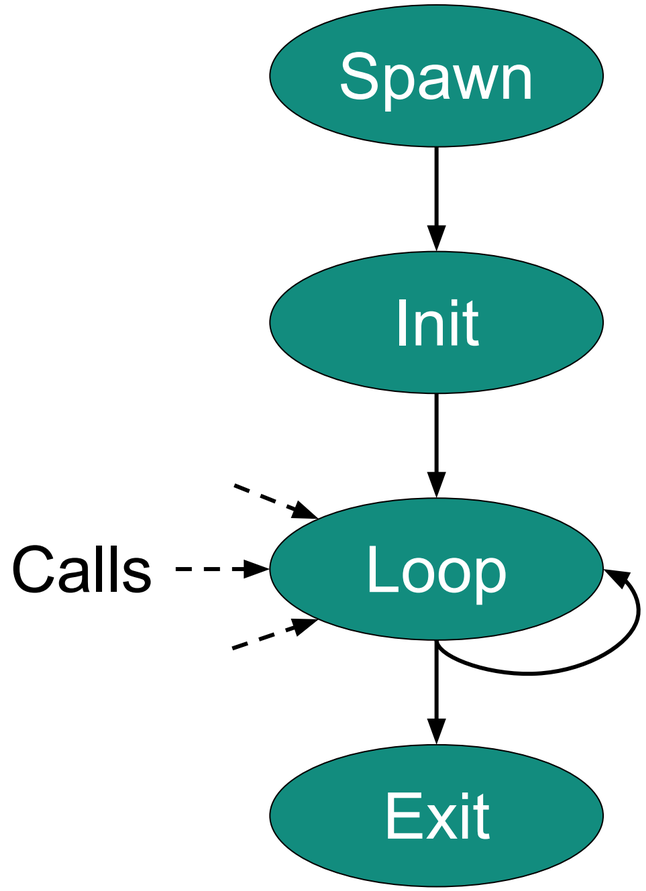
- Plans
  - Generalization of function call
  - Connect send/receive
  - Create summaries
  - Fixed-point computation

```
$ erl
1> c(concurrent).
{ok,concurrent}
2> C = spawn(concurrent, f, []).
<0.91.0>
3> C ! {self(), 1}.
{<0.84.0>,1}
4> flush().
Shell got 2
ok
```

Concurrency - high-level abstractions

- Some supported

- Example: gen_server
  - Distributed objects/actors
  - Rely on dynamic dispatch and specialization



https://learnyousomeerlang.com/what-is-otp

Library functions

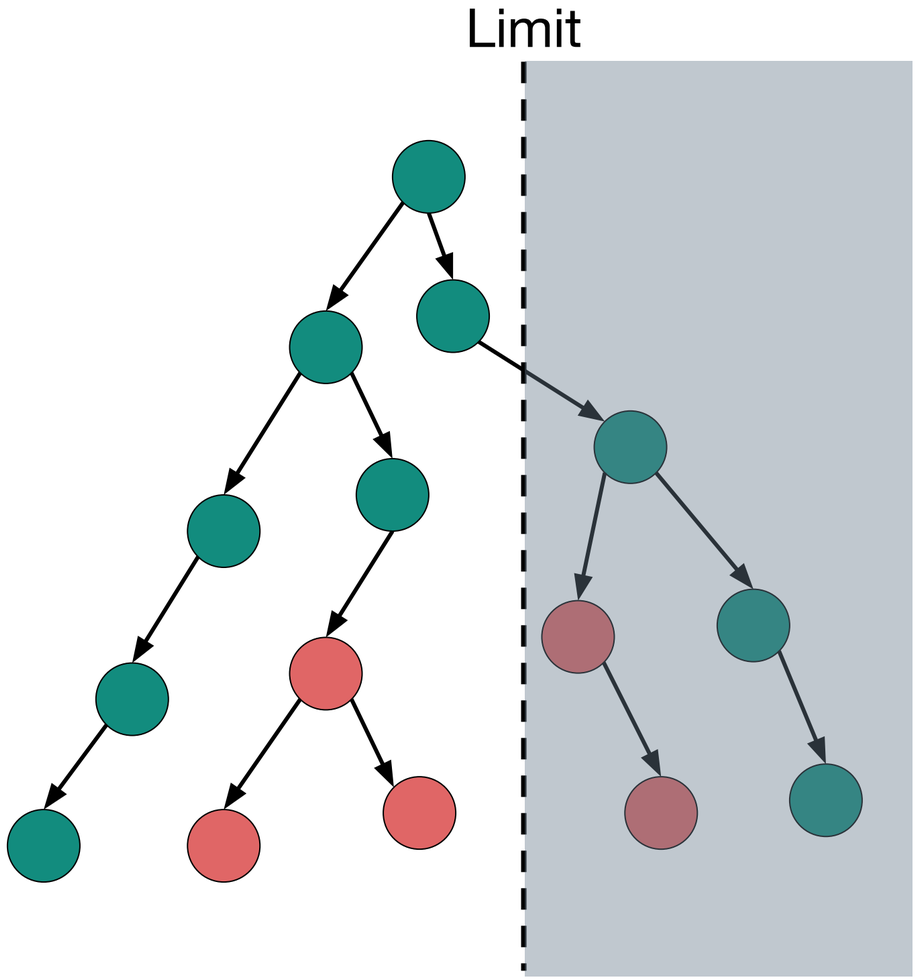
- OTP: use type specs
- User-defined models

```
"pulse-models-for-erlang": [
  {
    "selector": [
      "MFA",
      {
        "module": "some_module",
        "function": "complicated_function",
        "arity": 0
      }
    ],
    "behavior": [
      "ReturnValue",
      [
        "Tuple",
        [
          [ "Atom", "true" ],
          null
        ]
      ]
    ]
  }
]
```

```
-module(some_module).

complicated_function() ->
   {true, nondet()}.
```

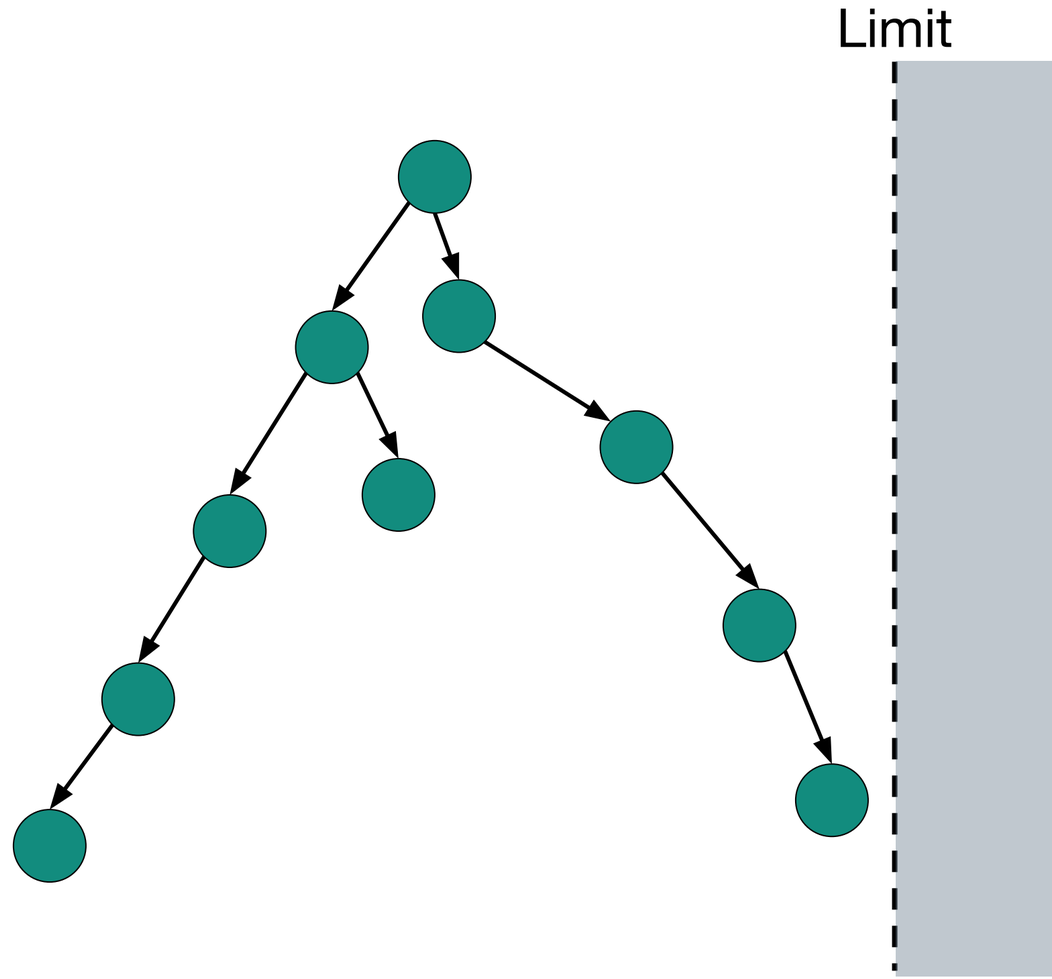
Topl

- Analyzer for user-defined (temporal) properties
  - Extends Pulse
- Scalability issues
  - Hit internal limit during exploration
  - False negatives
- Improvements
  - More expensive normalization
    - Triggered selectively
  - Dynamic types in solver
  - Garbage collector



Topl

- Analyzer for user-defined (temporal) properties
  - Extends Pulse
- Scalability issues
  - Hit internal limit during exploration
  - False negatives
- Improvements
  - More expensive normalization
    - Triggered selectively
  - Dynamic types in solver
  - Garbage collector



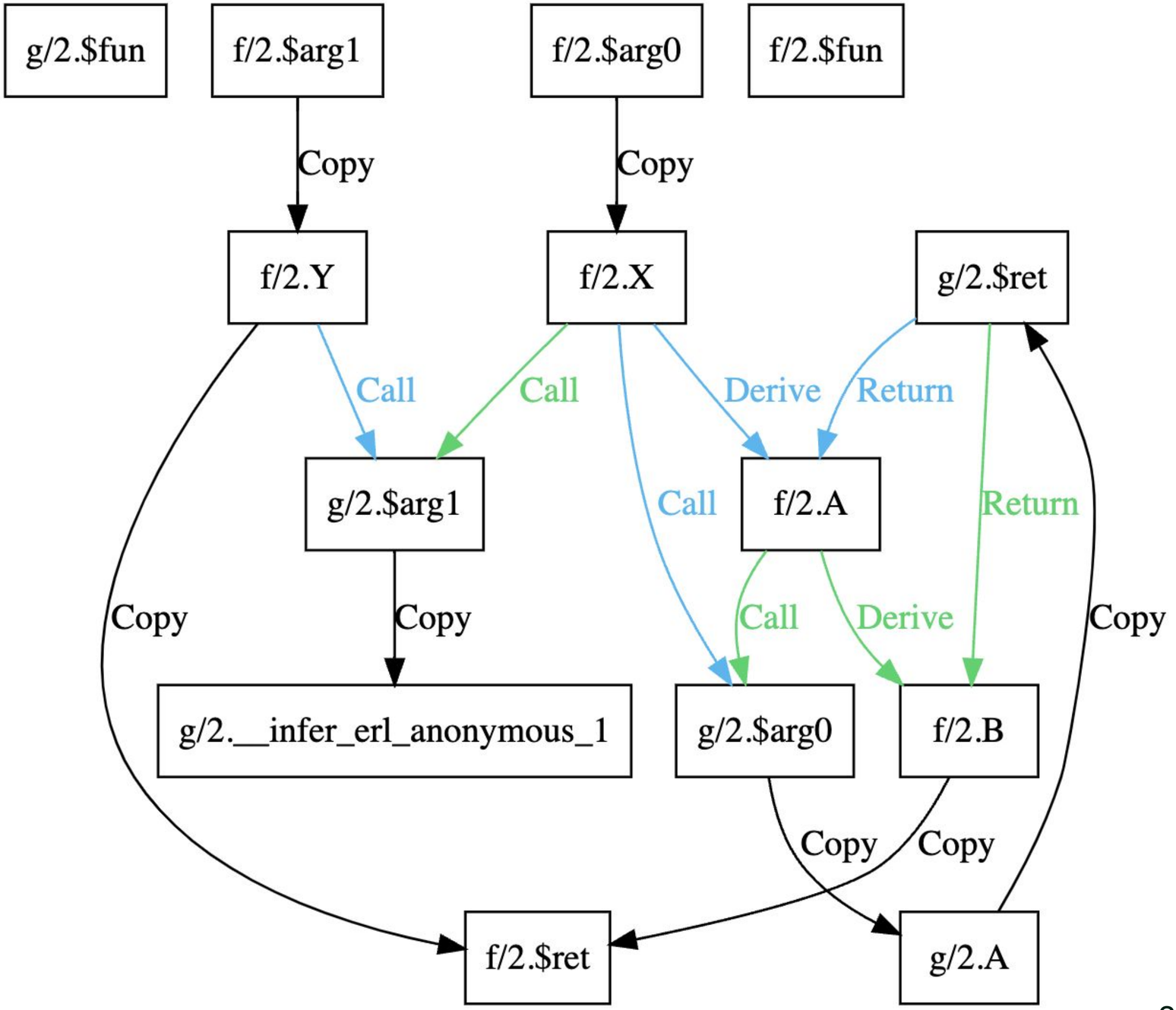
Compilation and analysis

Lineage

Lineage - Example

```
1 -module(a).
2 -export([f/2]).
3
4 f(X, Y) ->
5   A = g(X, Y),
6   B = g(A, X),
7   B + Y.
8
9 g(A, _) -> A.
```

```
$ infer --lineage
  --lineage-json-report
  -- erlc a.erl
```



Lineage - Main Characteristics

- Produces a (data-flow) graph
- Simple
  - not path/value sensitive
  - partially context sensitive
- … but efficient
  - at least 2x faster than Pulse
  - memory use is negligible in practice
- … and getting smarter
  - has recently became field-sensitive
  - will soon handle some polymorphism

Lineage - Analysis basics

AbstractState = Variable → 2^{Location}

- Remember locations where variables were last written
- Alternative view: set of variable → location bindings

L: Y := … X …

- For each X → L’ in the abstract state: generate an edge (X, L’) → (Y, L)
- Replace all bindings for Y with the one binding Y → L

Join

- Union of all bindings

Lineage - Analysis basics

Summary = Graph x Tito

Vertex = (Variable x Location) $\uplus$ (FunctionName x {$ret, $arg0, $arg1, …})

Tito = $2^{\text{ArgumentIndex}}$

- Analyze function, remember whole graph
- Plus set of arguments flowing into return

L: Y := call f(X_0,X_1)

- For k in {0,1}
  - For each $X_k \rightarrow$ L' in abstract state
    - Generate edge (X_k, L') $\rightarrow$ (f, $\$arg_k$)
    - If k in Tito of f: generate edge (X_k, L') $\rightsquigarrow$ (Y, L)
- Generate edge (g, $ret) $\rightarrow$ (Y, L)
- Replace all bindings for Y with Y $\rightarrow$ L

Lineage - Analysis basics

After abstract interpretation obtain summary

- Collect all generated edges into graph
- Compute Tito: reachability *while ignoring vertices (f, arg_k$)*
  - Gives context-sensitivity

Recursive calls?

- Assume all arguments are in tito
- No iteration to find a lfp

Dynamic analysis

FAUSTA: Scaling Dynamic Analysis with Traffic Generation at WhatsApp
Ke Mao, Timotej Kapus, Lambros Petrou, Ákos Hajdu, Matteo Marescotti,
Andreas Löscher, Mark Harman, Dino Distefano
ICST 2022

FAUSTA

Fully **AU**tomated **S**erver **T**esting and **A**nalysis

Client-server traffic specifications

Extensible Messaging and Presence Protocol (XMPP / RFC 6120)

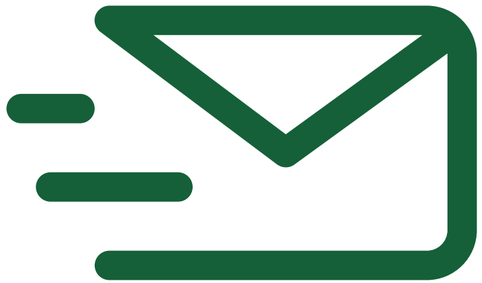
Specification

```
<relayelection spec:call-id="id">
  <te spec:latency="int(x, y)">
     <spec:ip_with_port>
  </te>
</relayelection>
```

Stanza

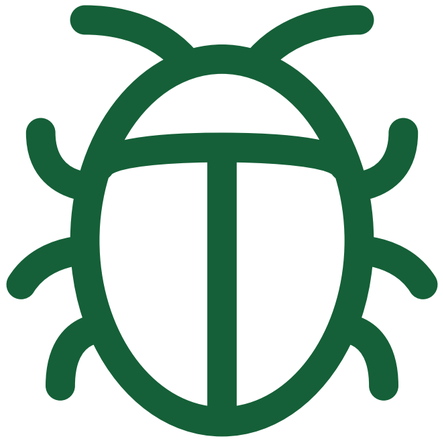
```
<relayelection call-id="123-4">
  <te latency="42">
     127.0.0.1:5678
  </te>
</relayelection>
```

Traffic generation



Realistic traffic

for validating critical workflows



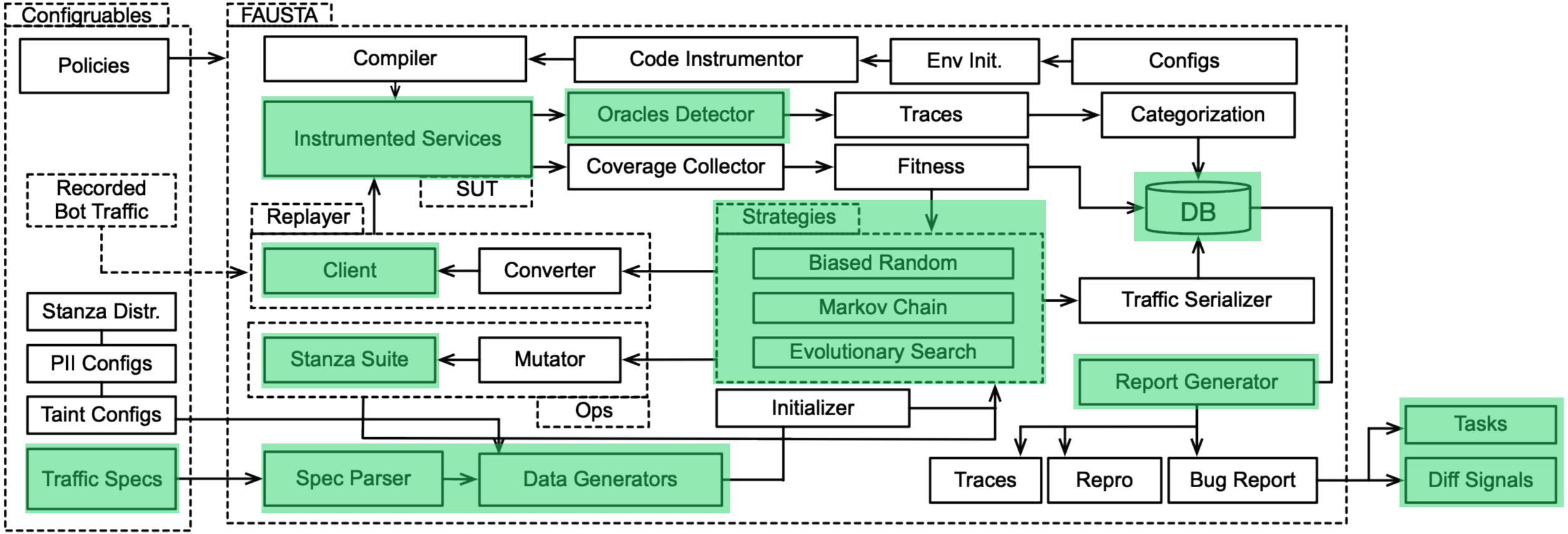
Unrealistic/invalid traffic

to exercise corner-case behaviors

Synthesized traffic w/ artificial PII

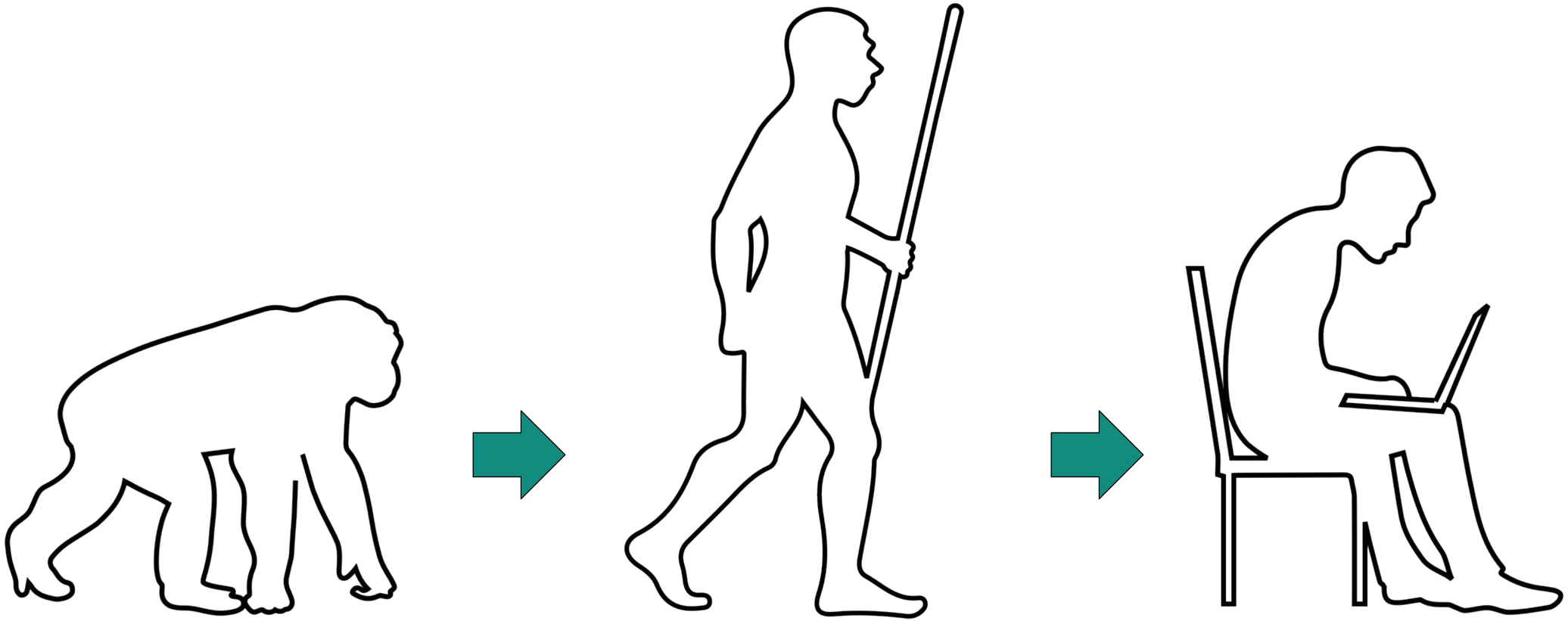
to prevent privacy regressions

FAUSTA architecture



Sapienz

Sapienz

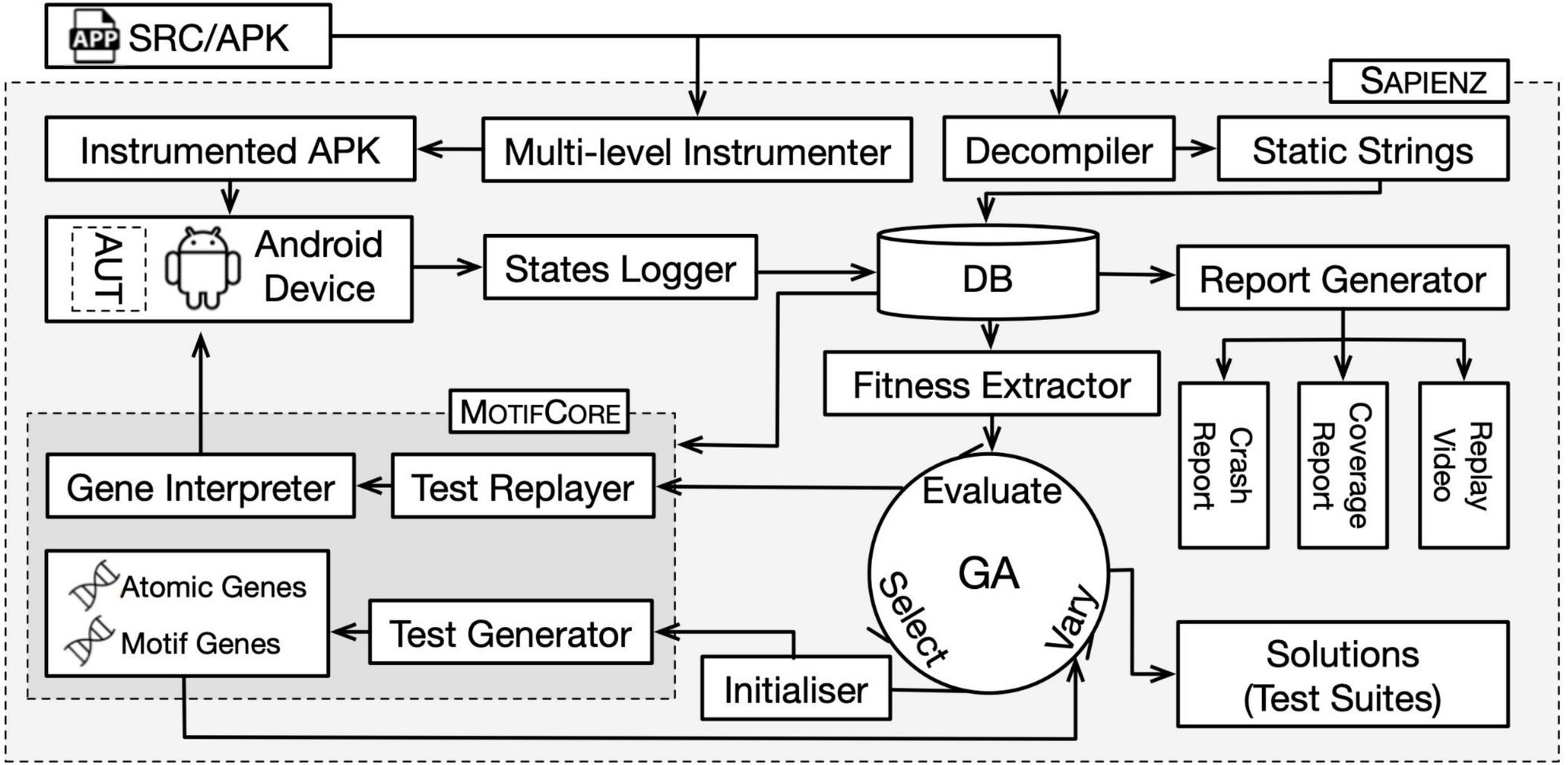


Random fuzzer

sapienz

Human tester

Sapienz



Mao, Ke, Mark Harman, and Yue Jia. "Sapienz: Multi-objective automated testing for Android applications." Proceedings of the 25th international symposium on software testing and analysis. 2016.

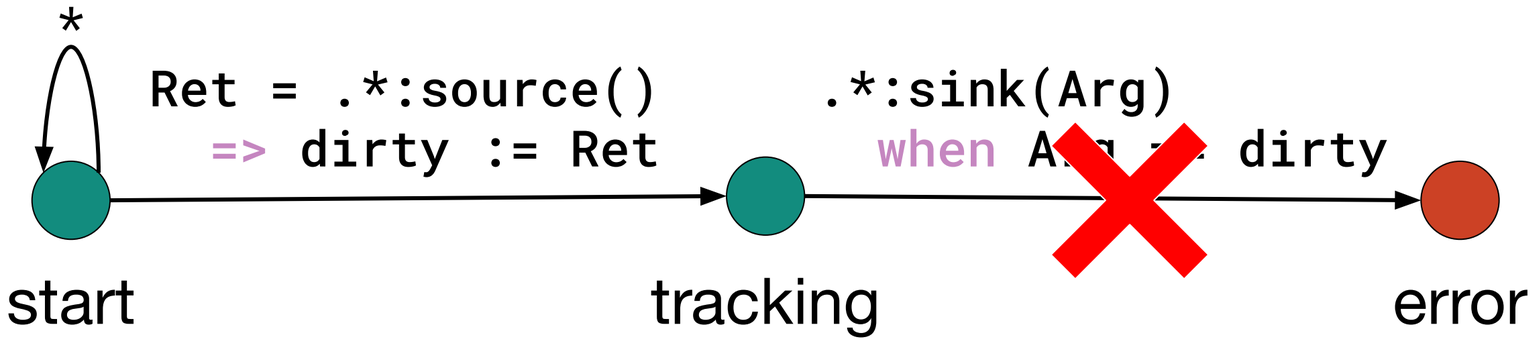
Alshahwan, Nadia, et al. "Deploying search based software engineering with Sapienz at Facebook." Search-Based Software Engineering: 10th International Symposium, SSBSE 2018, Montpellier, France, September 8-9, 2018, Proceedings 10. Springer International Publishing

Dynamic + Static

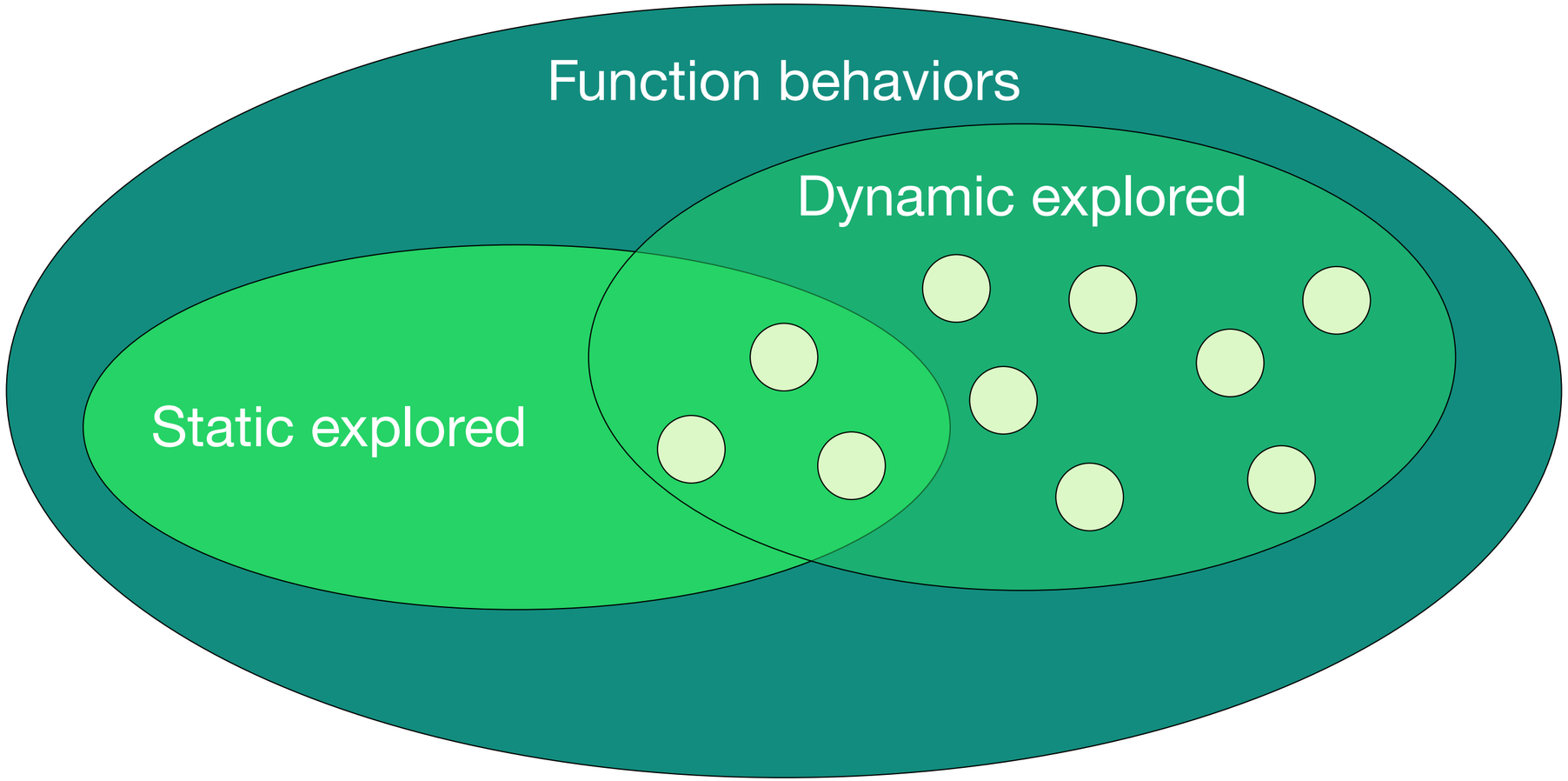
Static analysis limitations

- Complex functions cannot be analyzed
  - Under-approximation stops
- Dynamic to the rescue!
  - Track input/output pairs using dynamic analysis
  - Use them as summaries

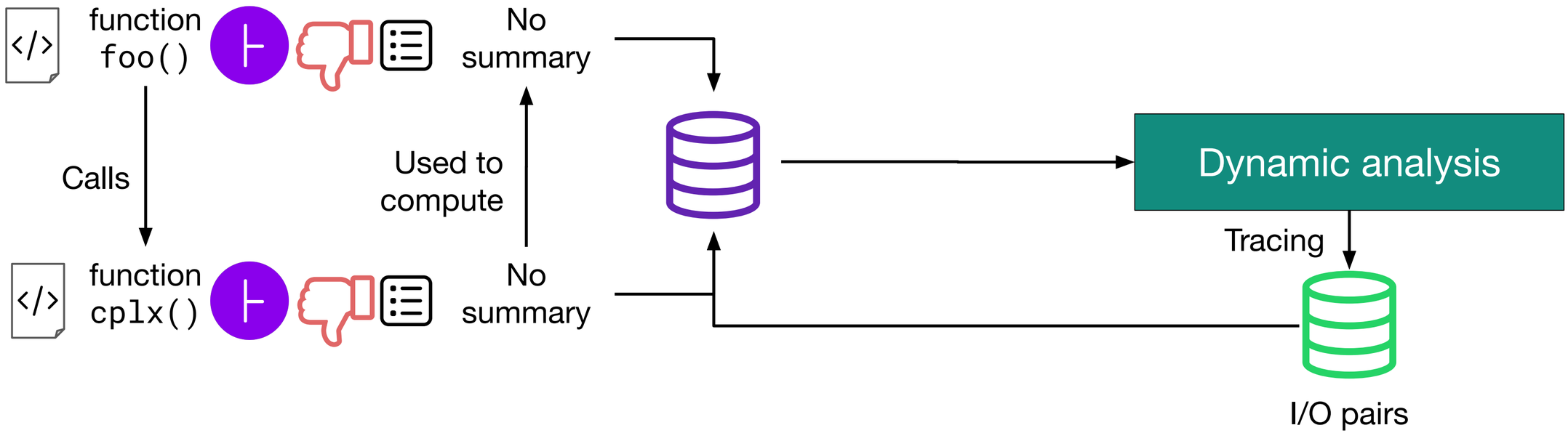
```
complex(Arg) ->
  ...

foo() ->
  X = source(),
  Y = complex(X),
  sink(X),
  Y.
```



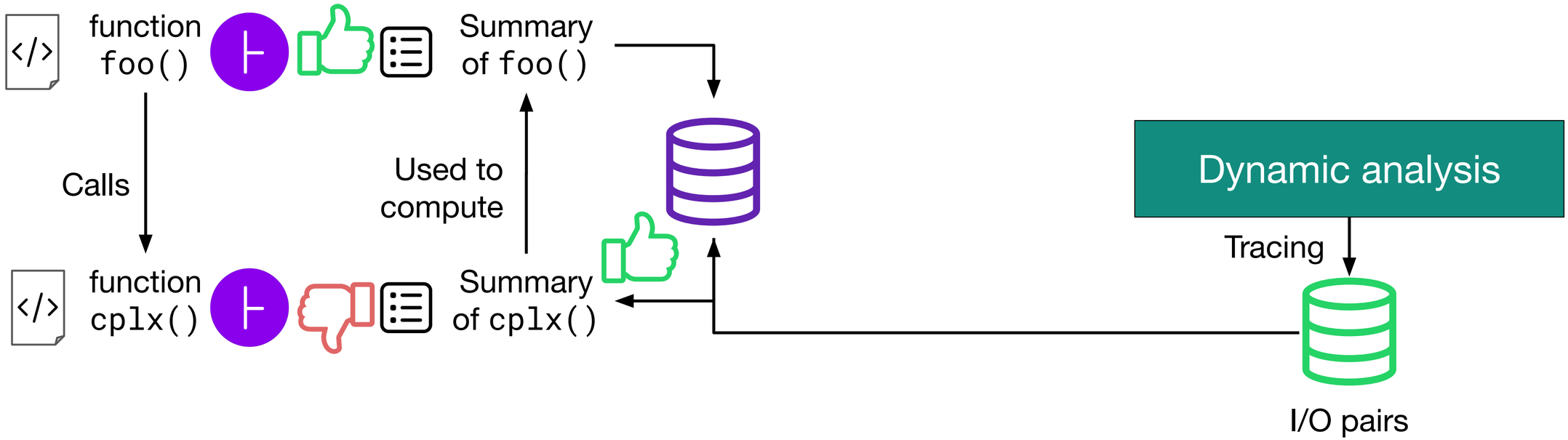
Under-approximation revisited



Dynamic + static

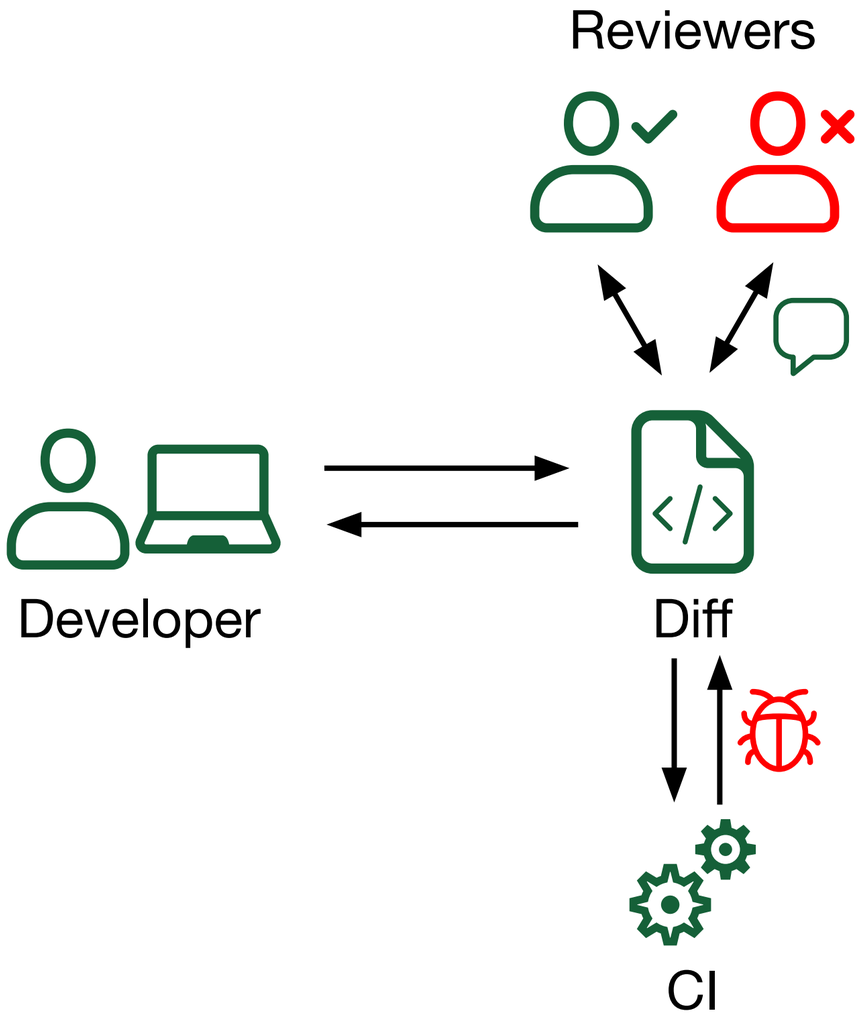


Dynamic + static



Deployment

Deployment



infer_report_example/CodeSample.java View Options

This file was **added**.

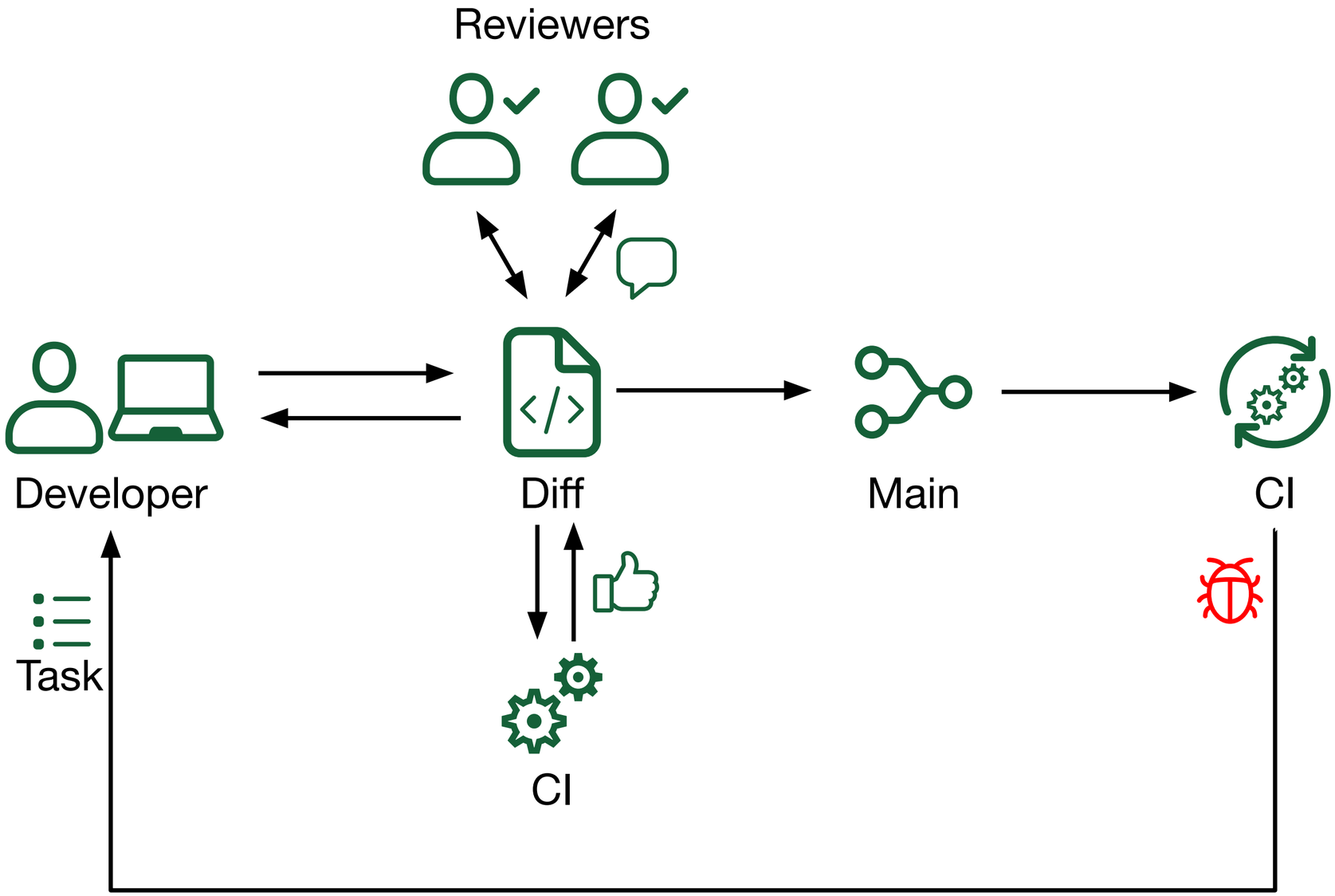
```
1  public class CodeSample {
2    public String computeSomething(boolean flag) {
3      if (flag) {
4        return null;
5      }
6      else {
7        return "something";
8      }
9    }
10
11   public int doStuff() {
12     String s = computeSomething(true);
13     return s.length();
```

Line 13 Previous · Next · Reply

There may be a Null Dereference: object s last assigned on line 12 could be null and is dereferenced at line 13

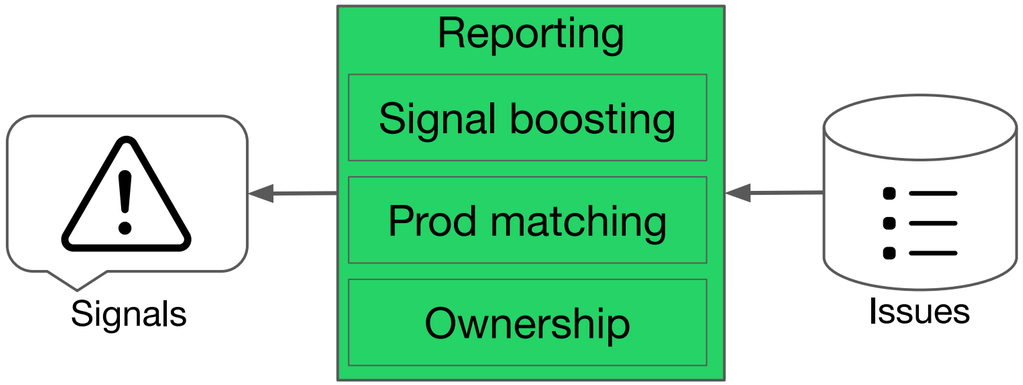
```
14   }
15 }
```

Deployment



Static analysis

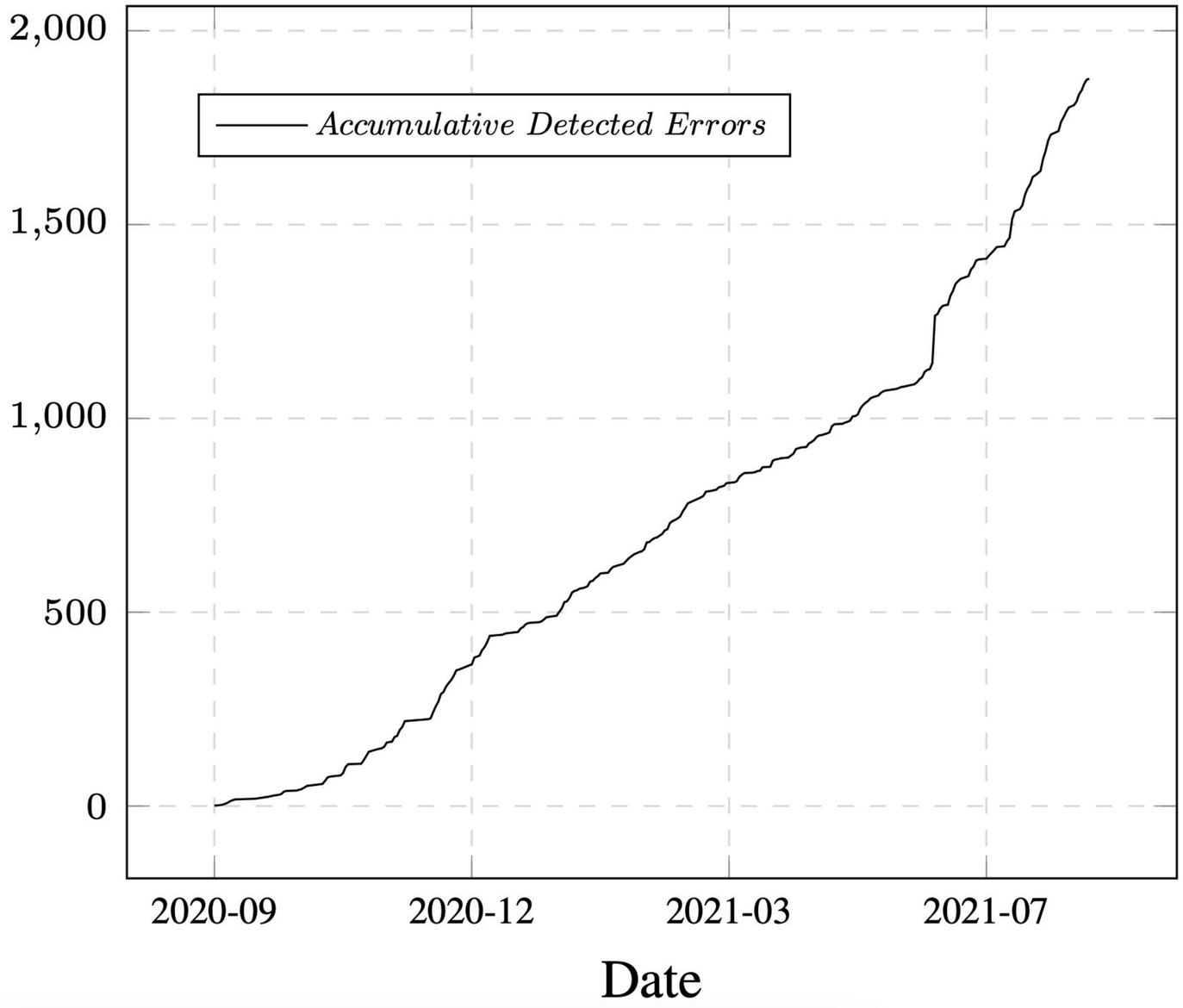
- Initial deployment with reliability: many false positives
- Taint properties + prod matching
  - 200 found
  - 21 surfaced
  - Fixed: 2 high-pri, 1 very high-pri
- Signal boosting
  - Dynamic analysis



Dynamic analysis (FAUSTA) - WhatsApp server

- Sep 2020 Start time of reliability error dataset
- Mar 2021 Start time of coverage collection
- Jun 2021 Comparison of two FAUSTA versions
- Aug 2021 End time of the dataset under analysis

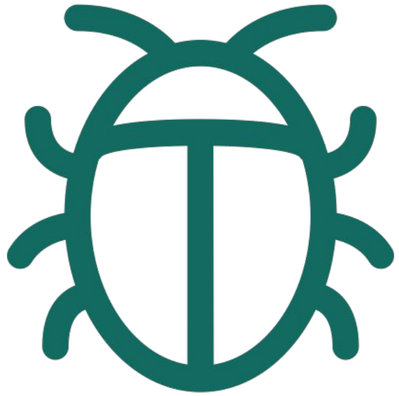
Dynamic analysis (FAUSTA) - WhatsApp server



Dynamic analysis (FAUSTA) - WhatsApp server

25m

Time-to-signal



1 876

Unique errors

74%

Fix rate

hajduakos.github.io