Erlang @ ICFP 2022
2022.09.16.

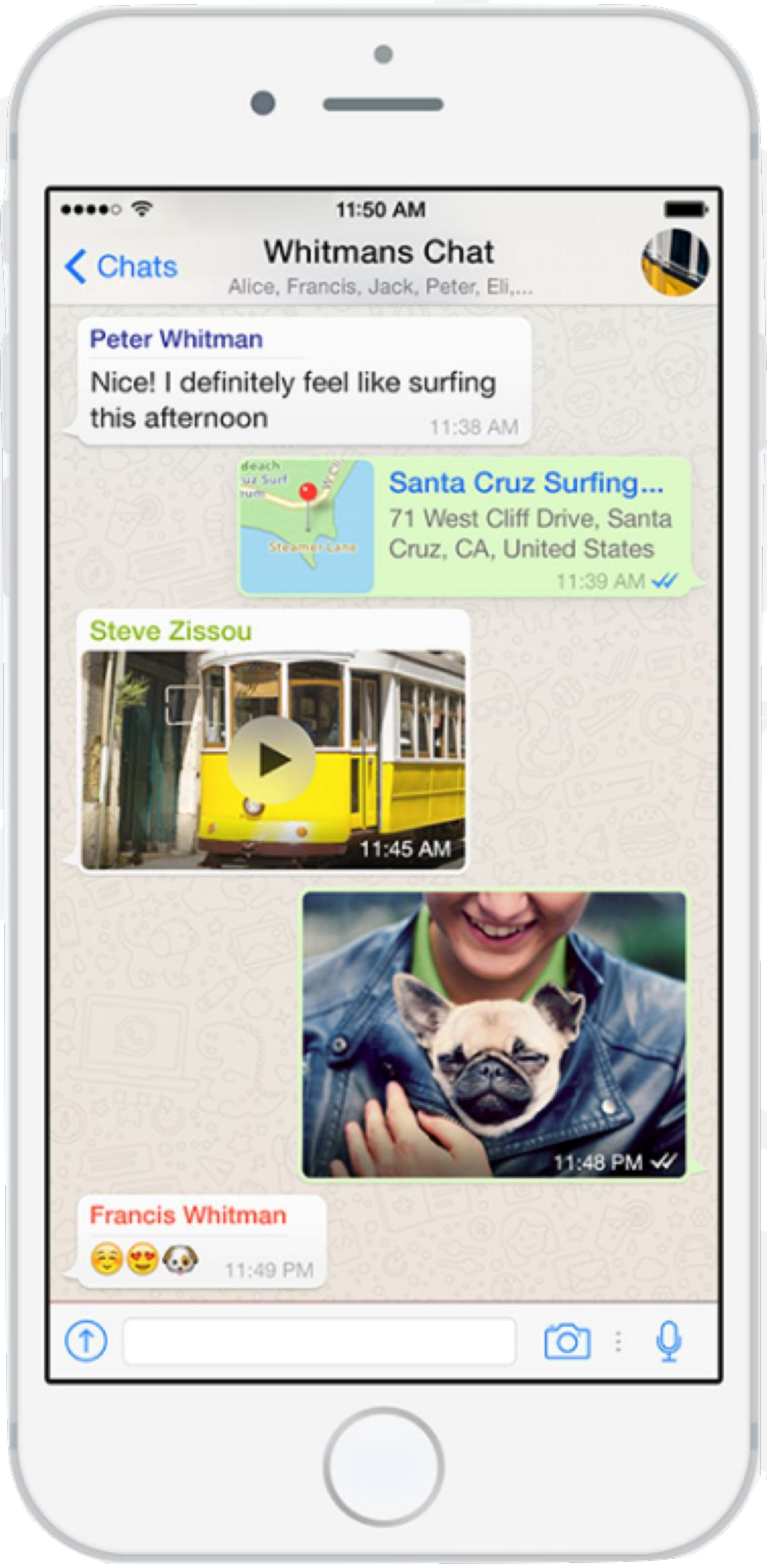
InfERL

Scalable and extensible static analysis for Erlang

Ákos Hajdu, Matteo Marescotti, Thibault Suzanne,
Ke Mao, Radu Grigore, Per Gustafsson, Dino Distefano
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.

WhatsApp server

- Millions of lines of Erlang code
- Static analysis applied so far
  - Linters
  - eqWAlizer (type checker)
    - github.com/WhatsApp/eqwalizer
  - (Incremental) Dialyzer
    - github.com/erlang/otp/pull/5997

WhatsApp/
eqwalizer

A type-checker for Erlang

8 Contributors · 4 Issues · 383 Stars · 10 Forks

github.com
GitHub - WhatsApp/eqwalizer: A type-checker for Erlang
A type-checker for Erlang. Contribute to WhatsApp/eqwalizer development by creating an account on GitHub.

erlang/otp
#5997 **dialyzer: Add incremental analysis mode**

12 comments · 11 reviews · 44 files · +4911 −882

TD5 • May 18, 2022 · 1 commit

github.com
dialyzer: Add incremental analysis mode by TD5 · Pull Request #5997 ...
Implements a new dialyzer --incremental mode. Incremental mode primarily differs from the previous, "classic", ways of running Dialyzer, ...

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

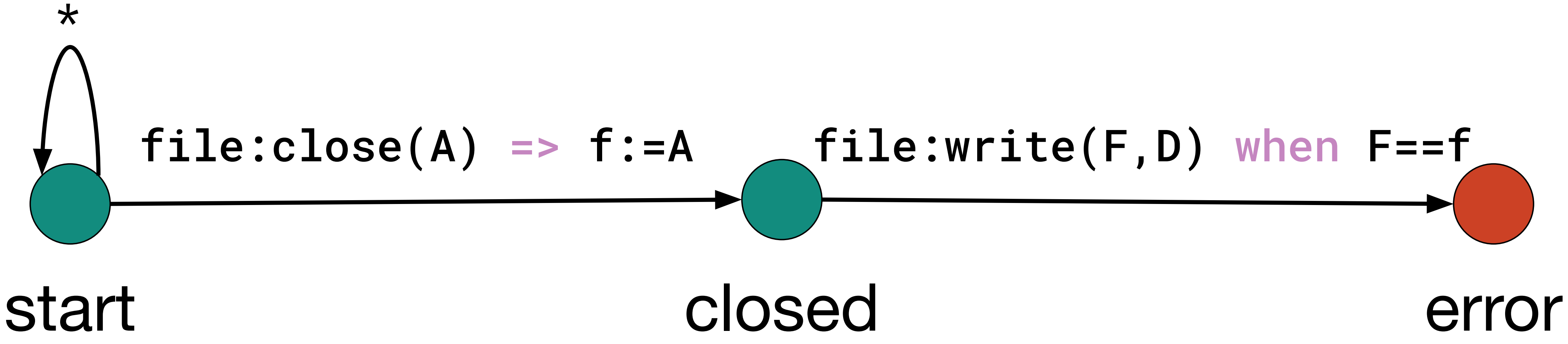
Memory safety, data races, time complexity, deadlocks, temporal properties, …

This talk

Examples

User-specified properties

Example: file:write after file:close



writep.topl

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

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

Interprocedural

write.erl

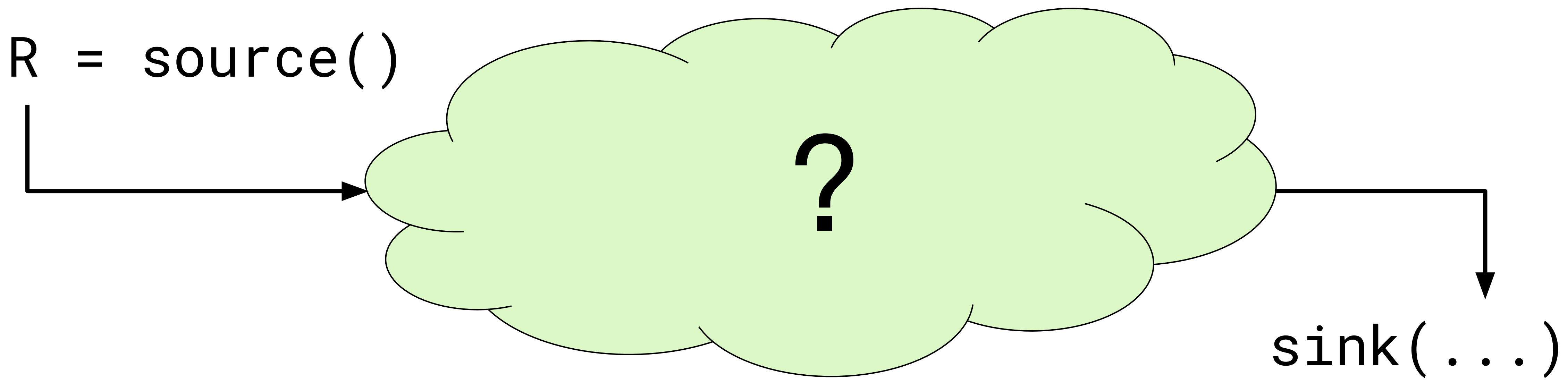
```
1 -module(write).
2
3 good(F) -> nop(F), file:write(F, "hi").
4 bad(F)  -> op(F),  file:write(F, "hi").
5 nop(_)  -> ok.
6 op(F)   -> file:close(F).
```

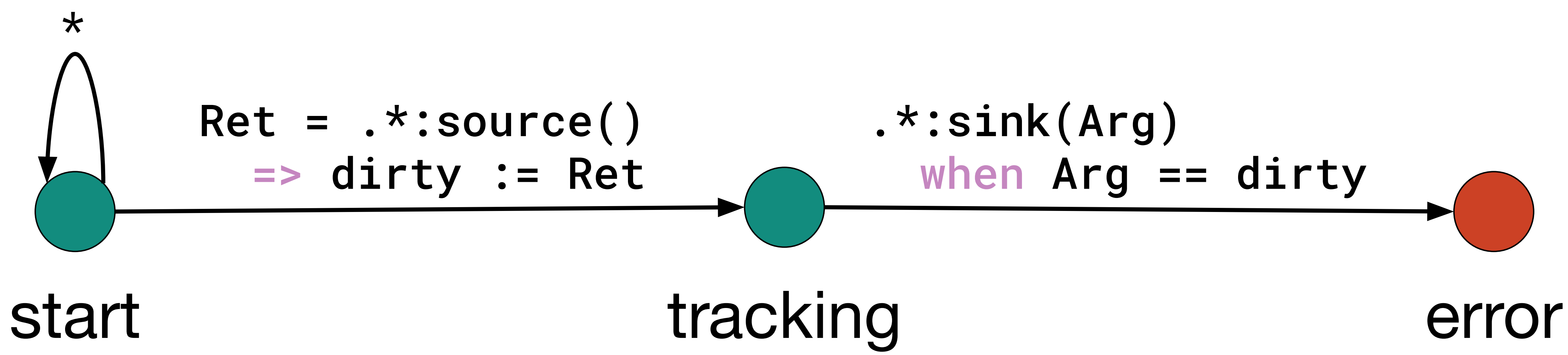
```
$ infer --topl-only --topl-properties writep.topl -- erlc write.erl
```

```
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

Built-in data structures, e.g:
`{X, Y}` is `__erlang_make_tuple(X, Y)`
`[H | T]` is `__erlang_make_cons(H, T)`

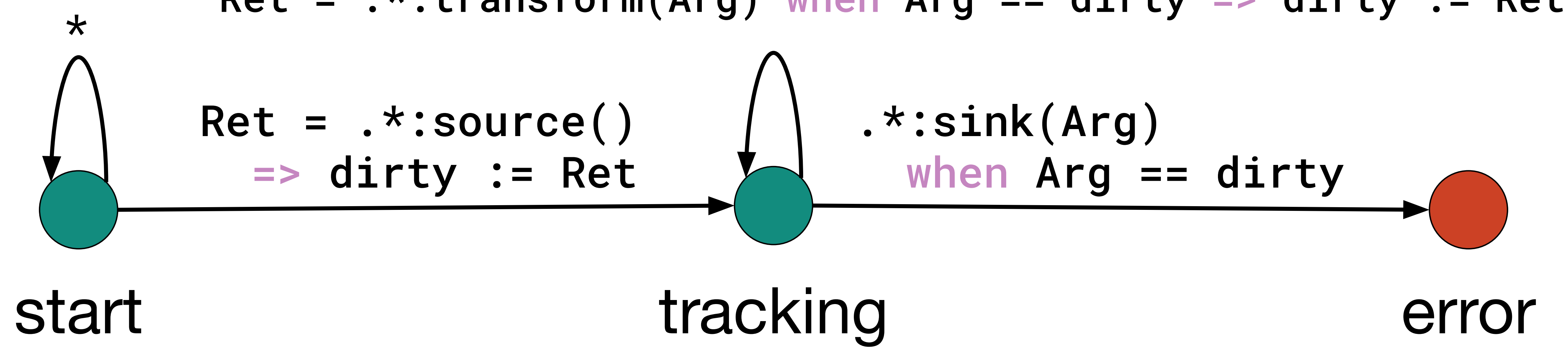
Add your functions of interest

```
Ret = __erlang_make_.*(A)       when A == dirty => dirty := Ret

Ret = __erlang_make_.*(A, B)    when A == dirty => dirty := Ret
Ret = __erlang_make_.*(A, B)    when B == dirty => dirty := Ret

Ret = __erlang_make_.*(A, B, C) when A == dirty => dirty := Ret
Ret = __erlang_make_.*(A, B, C) when B == dirty => dirty := Ret
Ret = __erlang_make_.*(A, B, C) when C == dirty => dirty := Ret

*
Ret = .*:transform(Arg) when Arg == dirty => dirty := Ret
```



Examples

Reliability issues

Example - Reliability

sample.erl

```erlang
1  -module(sample).
2
3  head([]) -> empty;
4  head([H|_]) -> {ok, H}.
5
6  good() ->
7    {ok, 1} = head([1, 2]).
8
9  bad() ->
10   empty = head([1, 2]).
11
12 bad2() ->
13   {ok, 3} = head([1, 2]).
14
15 very_bad() ->
16   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

```
% 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.
```

Challenges

Context

Support for Java, C/C++/Obj-C, C#, …

Built-in analyzers: Pulse, Topl, …

Compositional, interprocedural

Functional

Higher-order functions, closures

Recursion

Captured variables, scopes

```
map(_, []) -> [];
map(F, [H | T]) -> [F(H) | map(F, T)].

main() ->
    X = 1,
    map(fun(Y) -> Y + X end, [1, 2, 3]).
```

Let it crash

User vs runtime exceptions

Fault tolerance with supervisors

```
head([H | _]) -> H.
```

Dynamic typing

Dynamic types used pervasively

Pattern matching seldom complete

```
len([]) -> 0;
len([_ | T]) -> 1 + len(T).
```

Concurrency

Designed with concurrency in mind

Primitives for send/receive

High-level abstractions

```
receive
  {From, X, Y} -> From ! X + Y
  {From, X} -> From ! X + 1
end
```

Scalability

WhatsApp server is huge

Early signals for developers

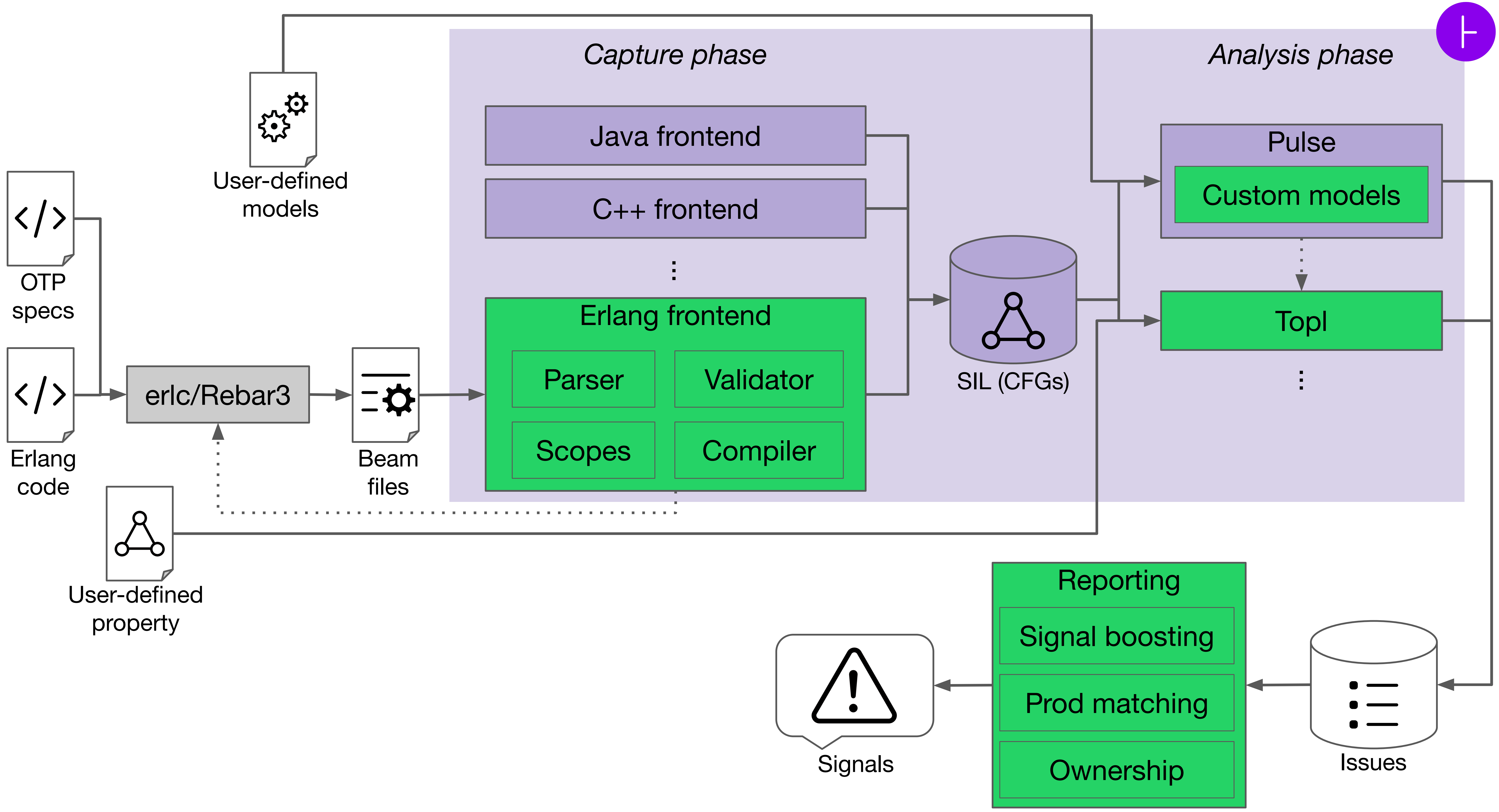
Erlang-specific abstractions

```
[X * Y || X <- [1, 2, 3],
          Y <- [1, 2, 3]]
```

Compilation and Analysis

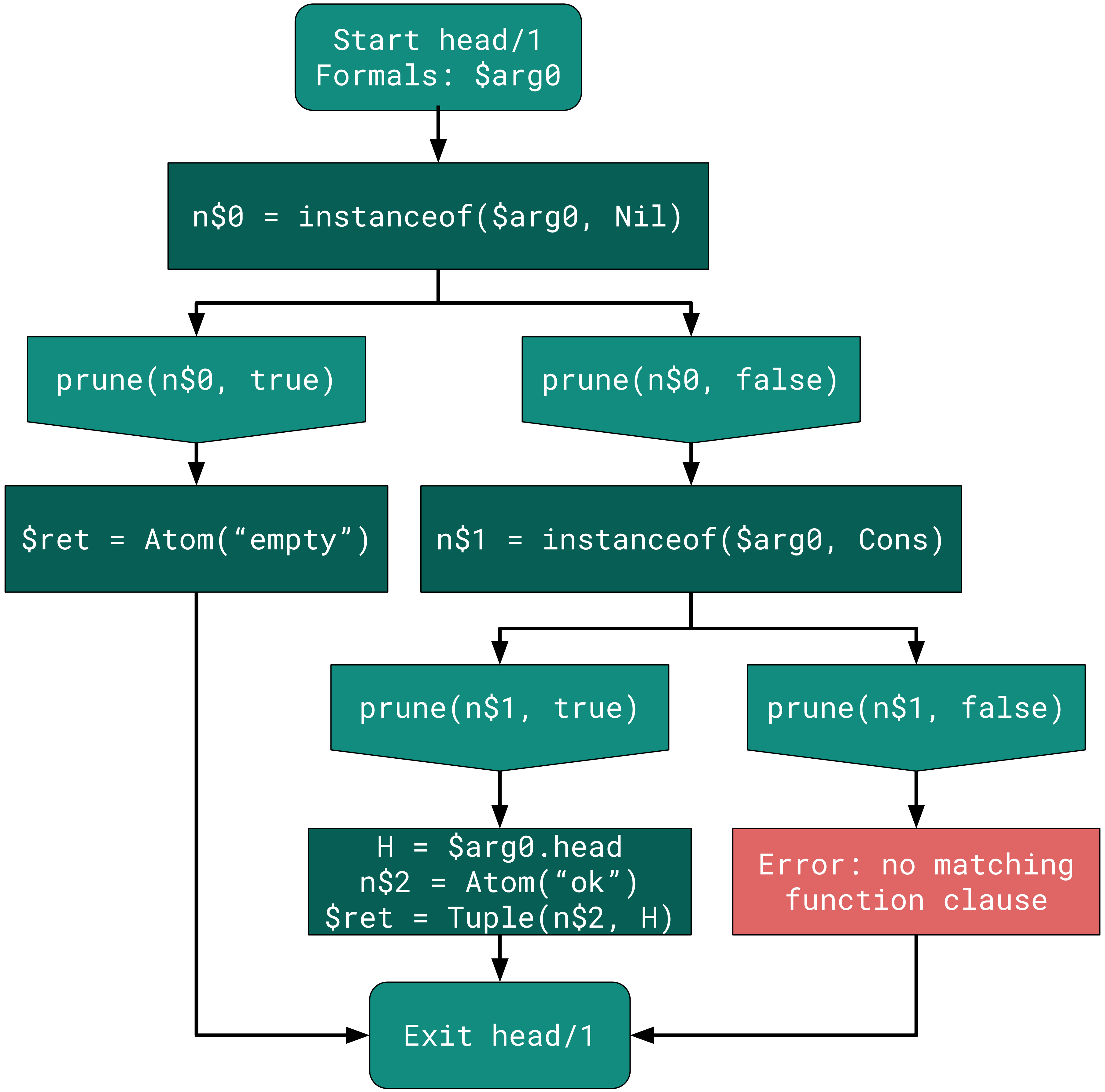
Basics

Overview


Capture phase
Analysis phase
Java frontend
C++ frontend
Erlang frontend
Parser
Validator
Scopes
Compiler
SIL (CFGs)
Pulse
Custom models
Topl
User-defined
models
OTP
specs
Erlang
code
erlc/Rebar3
Beam
files
User-defined
property
Reporting
Signal boosting
Prod matching
Ownership
Signals
Issues

Compilation basics

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



Analysis basics

- Summaries
  - Compute once per function
  - Compact representation of interesting behaviors

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

$\vdash$

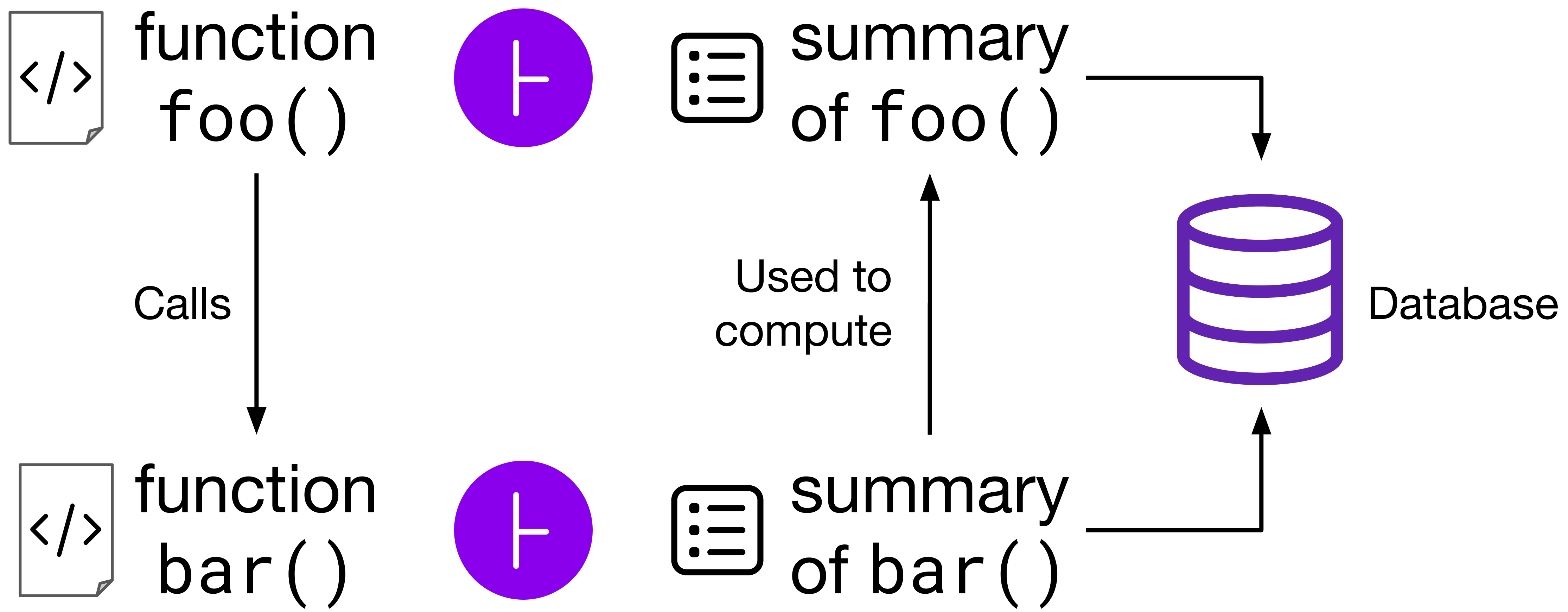
```
1. pre:  $arg0 instanceof Nil
   post: $ret = Atom("empty")

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]

$$[[Res]] \subseteq [[Code]]_{ok/err}([[Pre]])$$

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 program
like to have correctness, you might find yourself spending most of your time reason
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 sen

CCS Concepts: • **Theory of co**

Additional Key Words and Phra

ACM Reference Format:
Peter W. O'Hearn. 2020. Incorr
32 pages. https://doi.org/10.114

1 INTRODUCTION

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
or proving their *presence*. We
easoning about the presence of
lations: 1) separation logic and
eory of this new *incorrectness*

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

Compilation and Analysis

Details

Type specs

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

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

- Return value of unknown functions (from OTP)

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

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

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

Data structures

- Tuples, records: full support
- 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
  - Scope analysis: captured vs local
    - github.com/erlang/otp/issues/5379

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

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

```
g(F) -> F().

f() -> 1.

main() -> g(fun f/0).
```

```
g_f() -> f().



main_f() -> g_f().
```

*Corentin De Souza: Higher-order function specialization in Infer, talk at Infer workshop 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: nondeterministic

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

send

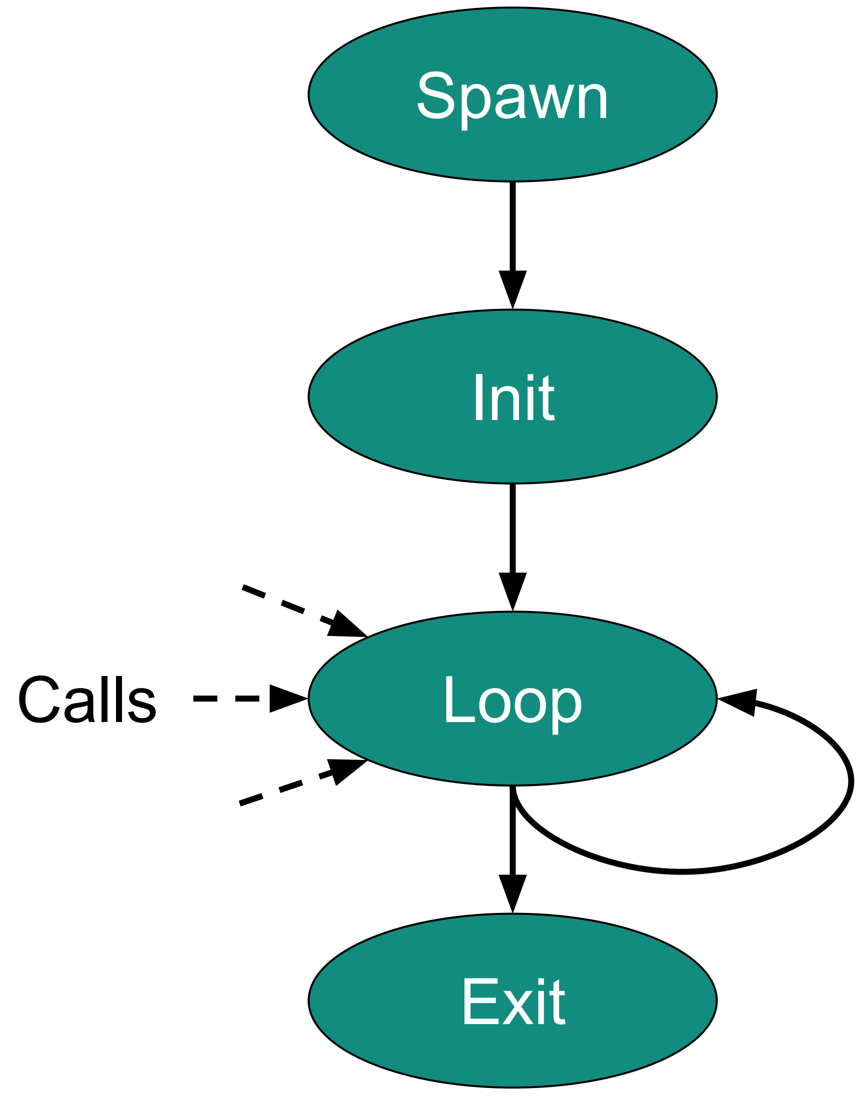
```
-module(concurrent).
-export([f/0]).

f() ->
  receive
    {From, X} when is_integer(X) -> From ! X + 1;
    {From, _} -> From ! oops
  end,
  f().
```

Concurrency - high-level abstractions

- Not yet supported

- Example plan: gen_server
    - Distributed objects/actors
    - Treat as objects (Infer has support)



Library functions

- OTP: use type specs
- User-defined models
  - Plan: more behaviors
  - Plan: more user-friendly

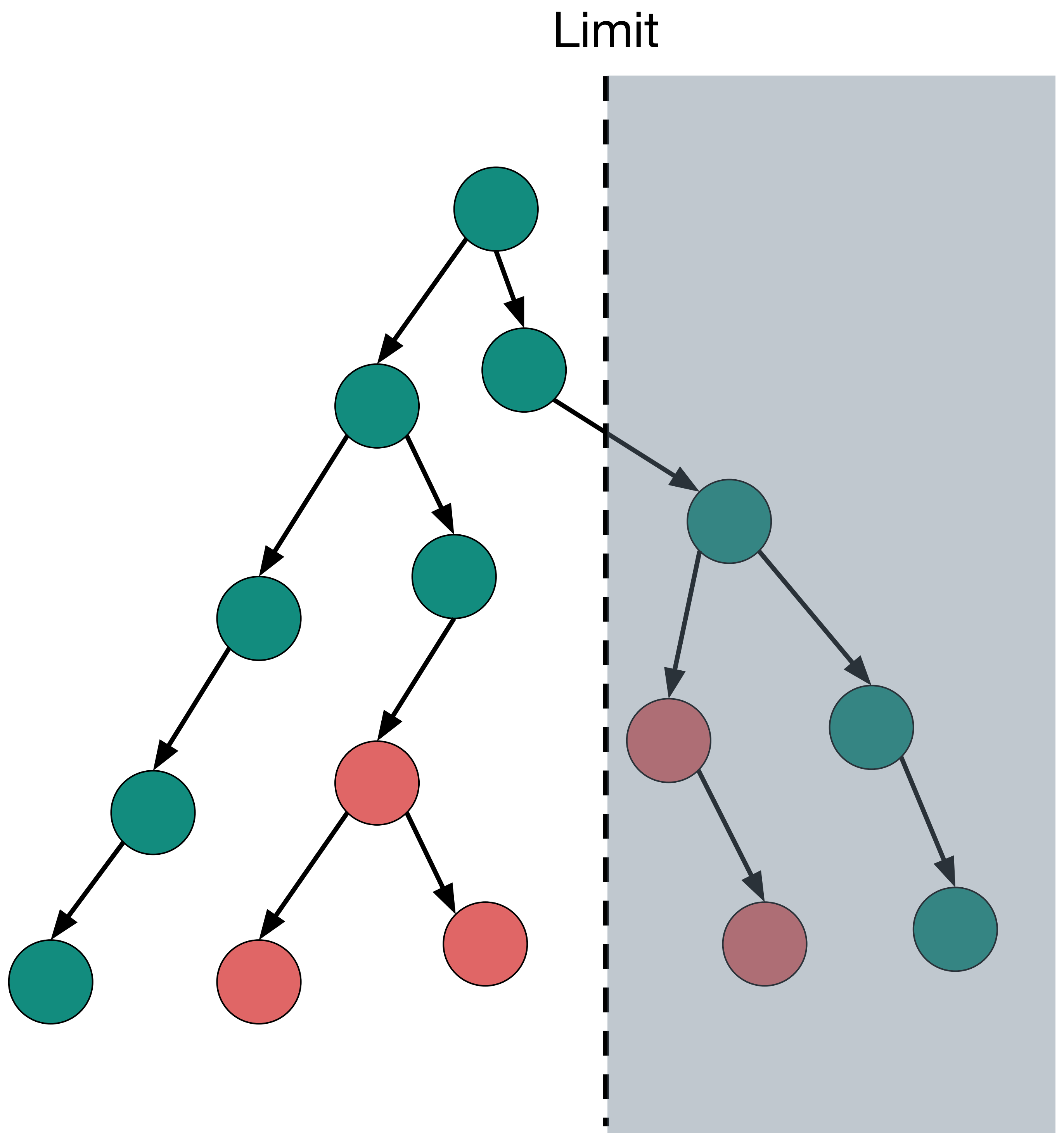
```
{
 "pulse-models-for-erlang": [
   {
     "selector": [
       "MFA",
       {
         "module": "some_module",
         "function": "complicated_function",
         "arity": 0
       }
     ],
     "behavior": [
       "ReturnValue",
       [
         "Tuple",
         [
           [ "Atom", "true"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

```
-module(some_module).

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

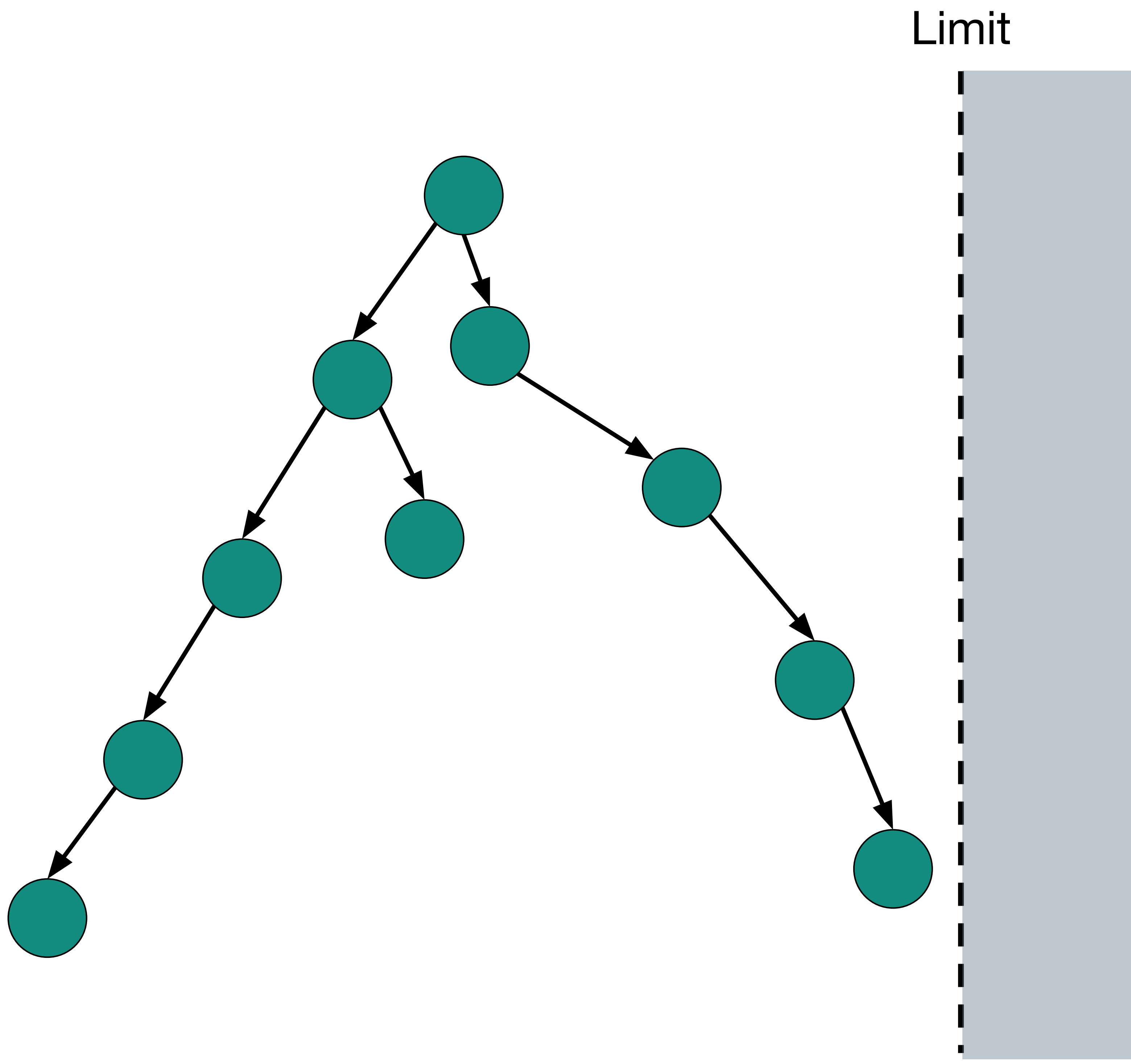
Topl

- 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



Results

Test functions

- ~750 small functions
  - github.com/facebook/infer/tree/main/infer/tests/codetoanalyze/erlang

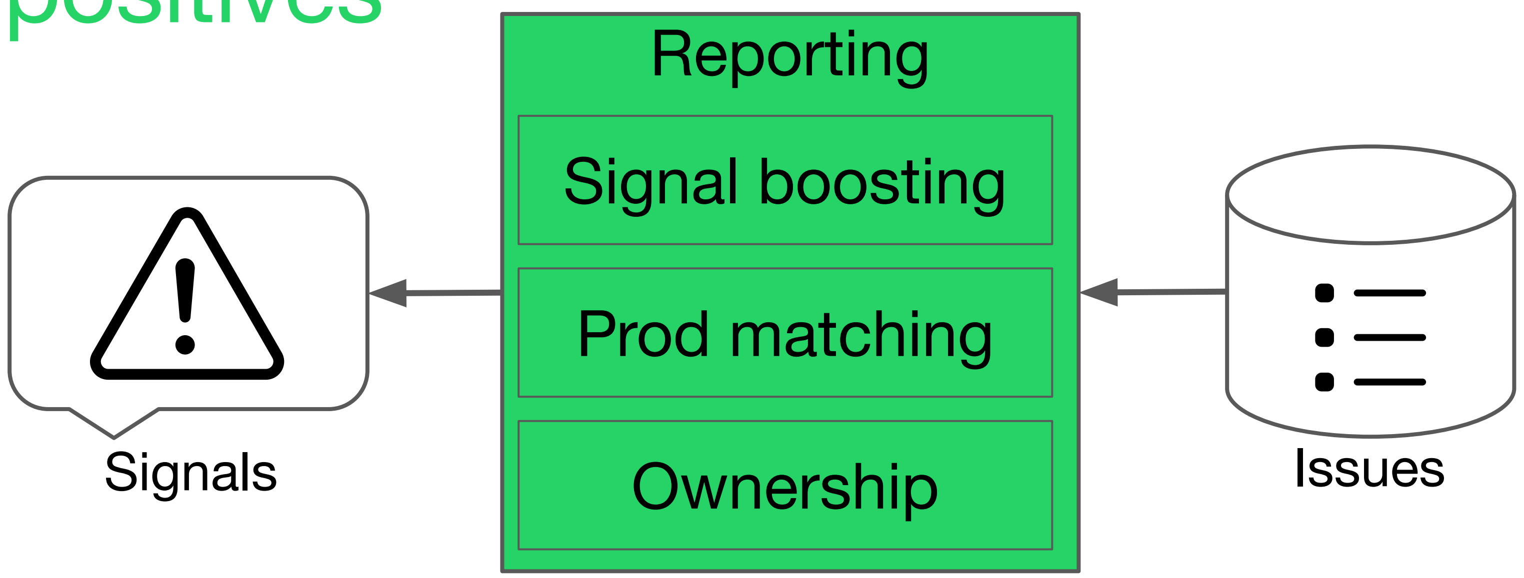
```
test_add_Ok() ->
  X = 2,
  Y = 3,
  case X + Y of
    5 -> ok;
    _ -> ?CRASH
  end.
```

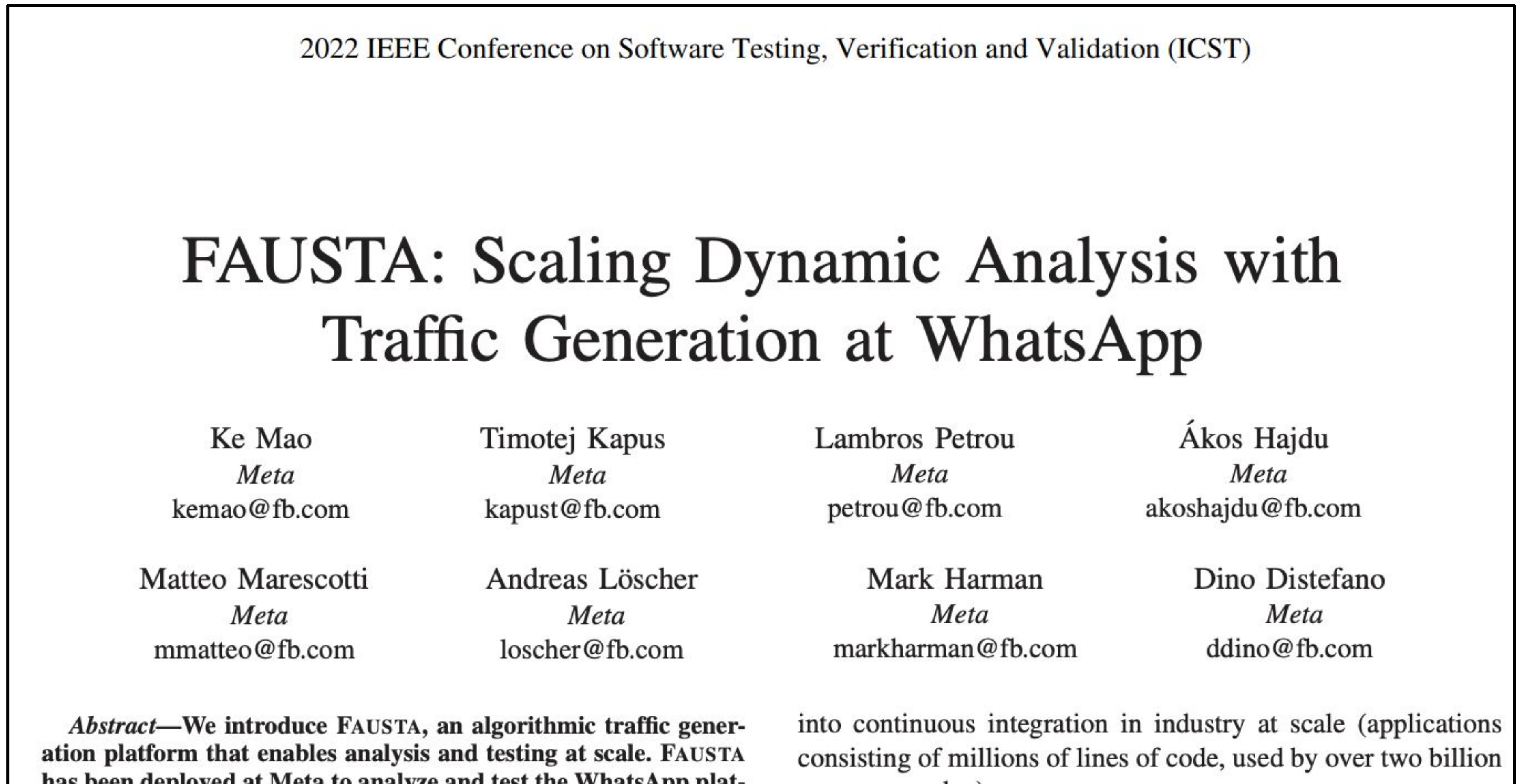
```
test_add_Bad() ->
  X = 2,
  Y = 3,
  case X + Y of
    5 -> ?CRASH;
    _ -> ok
  end.
```

Compile & run	**Infer**	
Ok	Ok	True negative
Ok	Issue	False positive
Crash	Ok	False negative
Crash	Issue	True positive

WhatsApp server

- 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: FAUSTA





2022 IEEE Conference on Software Testing, Verification and Validation (ICST)

FAUSTA: Scaling Dynamic Analysis with Traffic Generation at WhatsApp

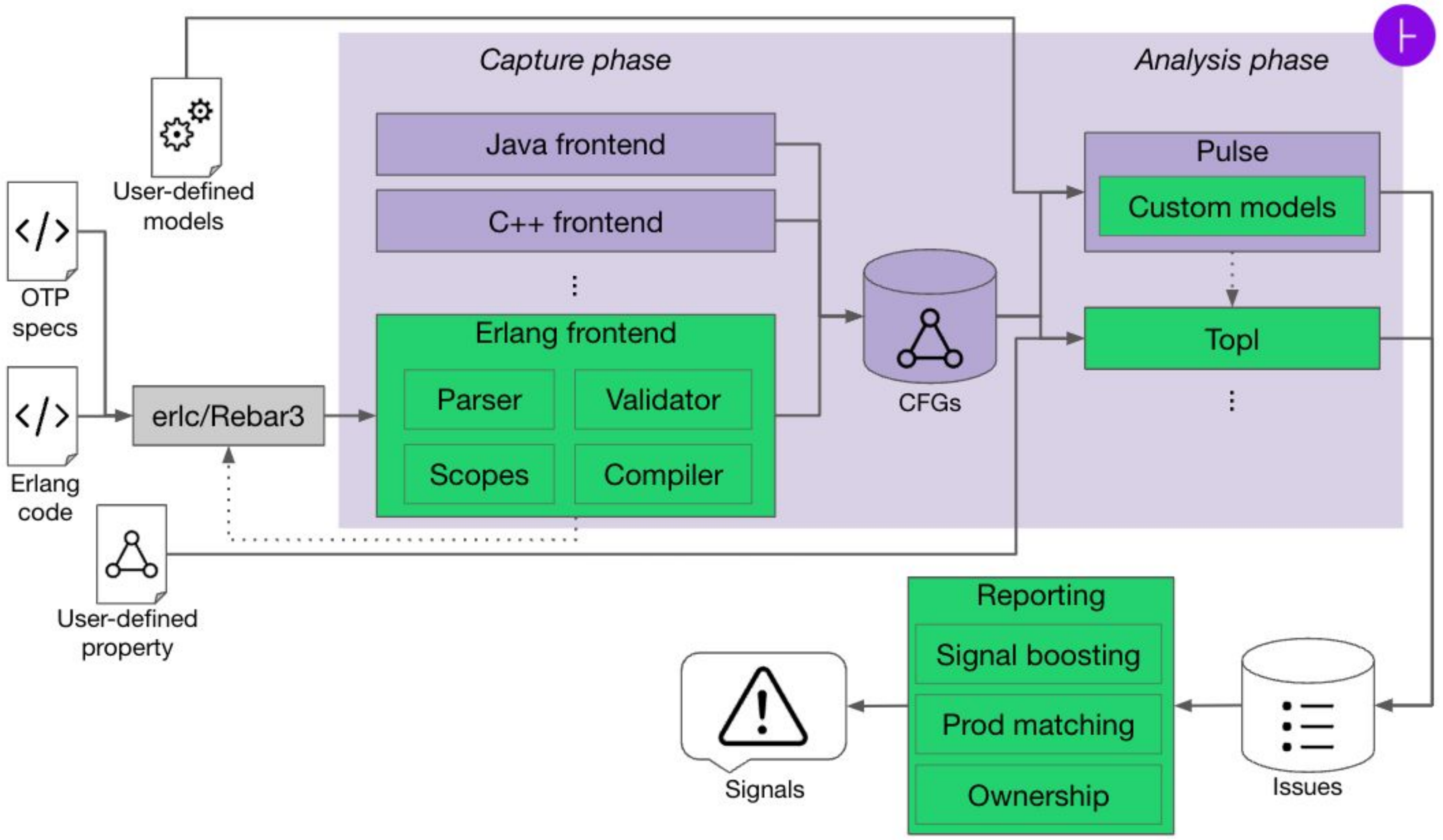
Ke Mao, Meta, kemao@fb.com
Timotej Kapus, Meta, kapust@fb.com
Lambros Petrou, Meta, petrou@fb.com
Ákos Hajdu, Meta, akoshajdu@fb.com
Matteo Marescotti, Meta, mmatteo@fb.com
Andreas Löscher, Meta, loscher@fb.com
Mark Harman, Meta, markharman@fb.com
Dino Distefano, Meta, ddino@fb.com

Abstract—We introduce FAUSTA, an algorithmic traffic generation platform that enables analysis and testing at scale. FAUSTA

into continuous integration in industry at scale (applications consisting of millions of lines of code, used by over two billion

Summary

- InfERL: scalable and extensible static analysis for Erlang
- Reliability issues + user-defined properties
- Challenges: functional, let it crash, dynamic typing, concurrent, scalability
- Erlang frontend
- Extensions to Pulse and Topl
- Promising results on WhatsApp server
- Available open-source



github.com/facebook/infer/tree/main/infer/src/erlang