LEVERAGING SYMBOLIC EXECUTION TO REPRODUCE FIELD FAILURES AND MIMIC USER BEHAVIOR

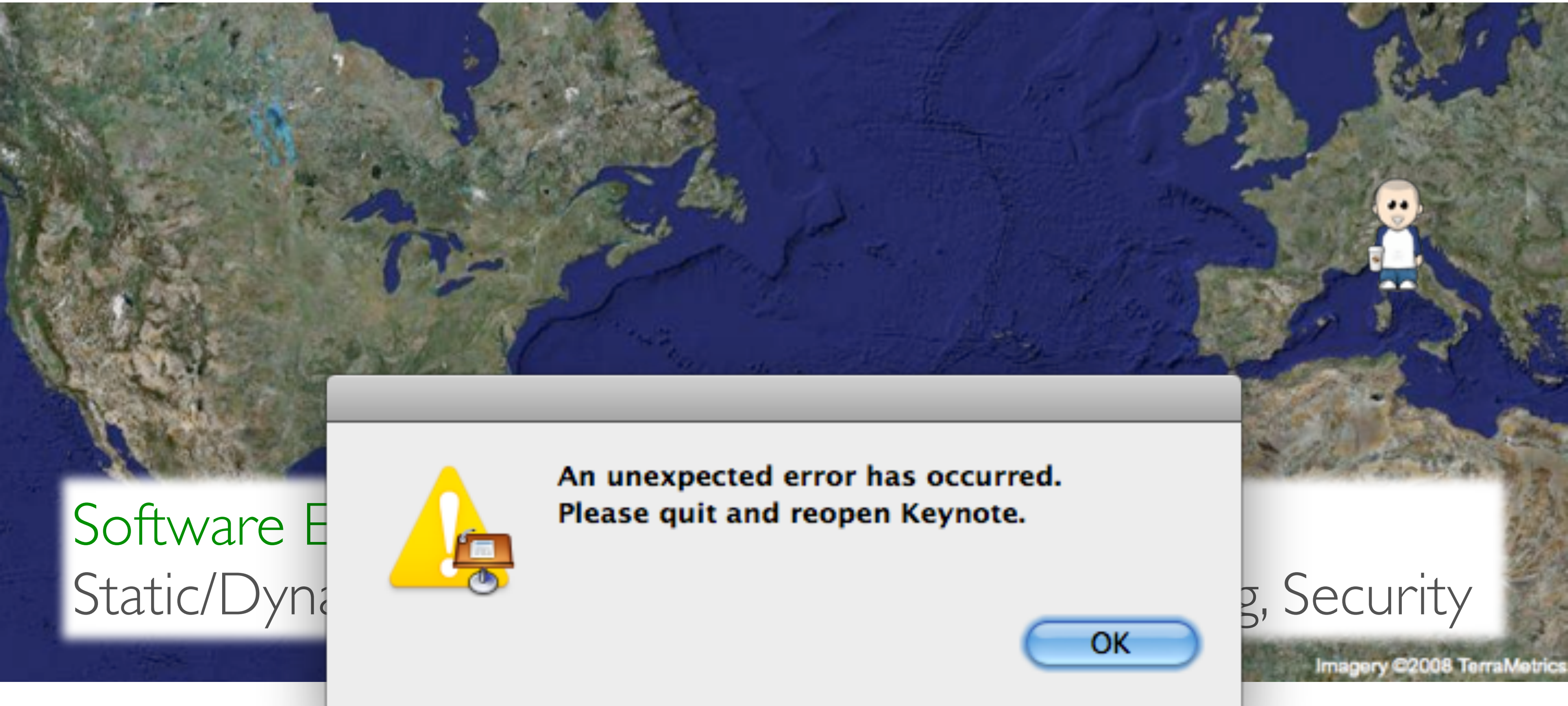
Alessandro (Alex) Orso
School of Computer Science
College of Computing
Georgia Institute of Technology

Partially supported by: NSF, IBM, MSR, and Google

Alessandro (Alex) Orso

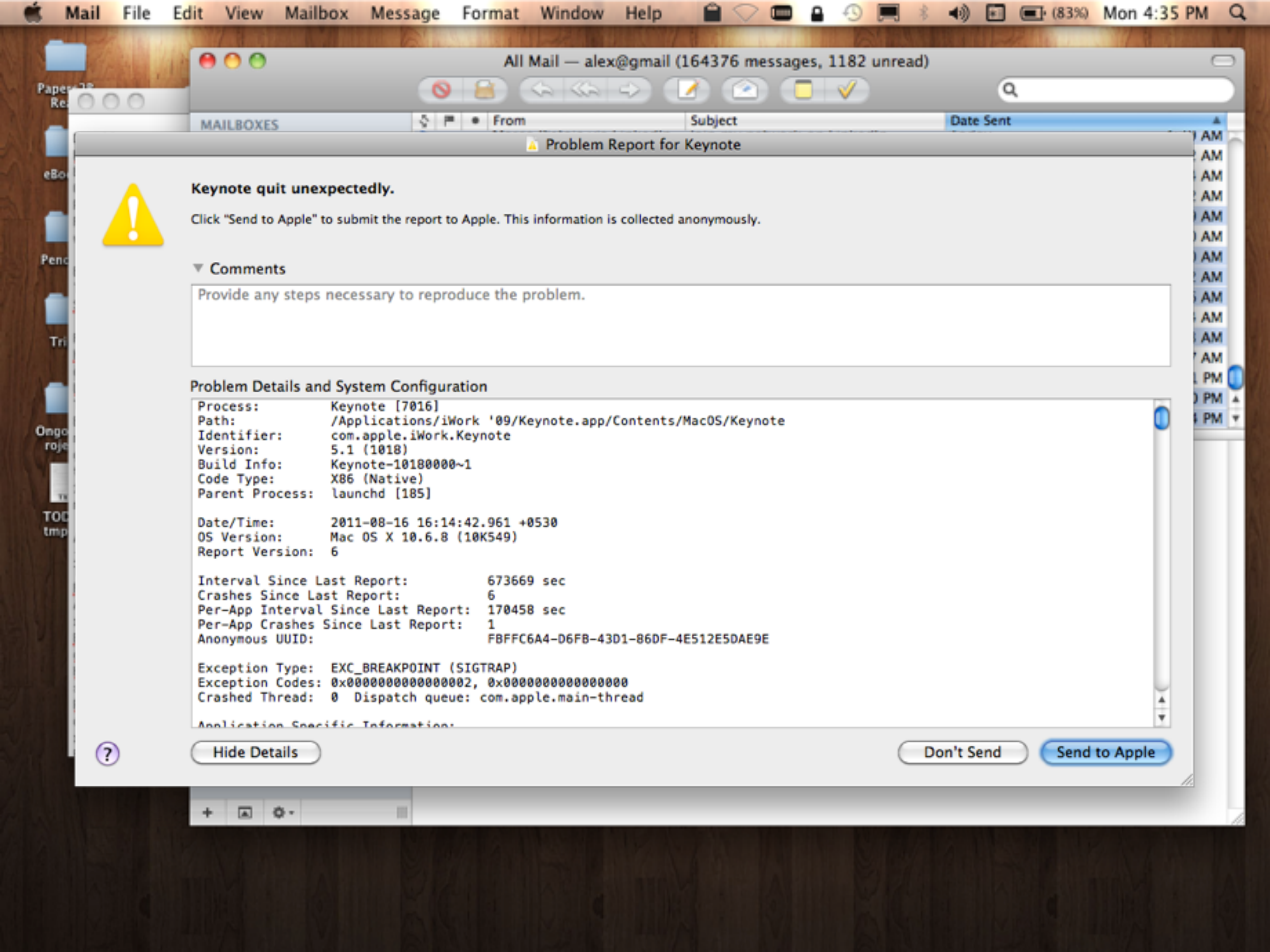
School of Computer Science
College of Computing
Georgia Institute of Technology

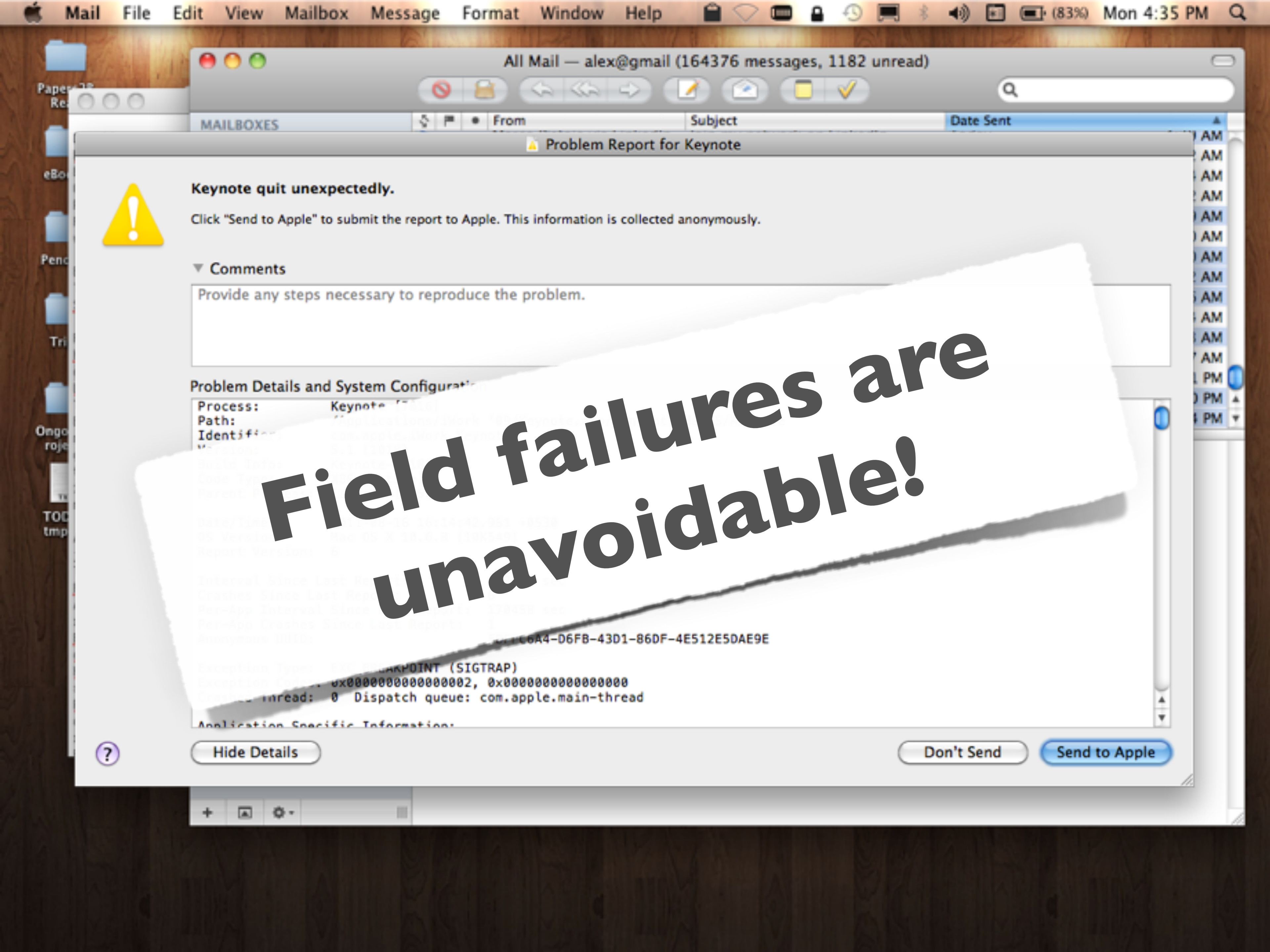
Partially supported by: NSF, IBM, MSR, and Google



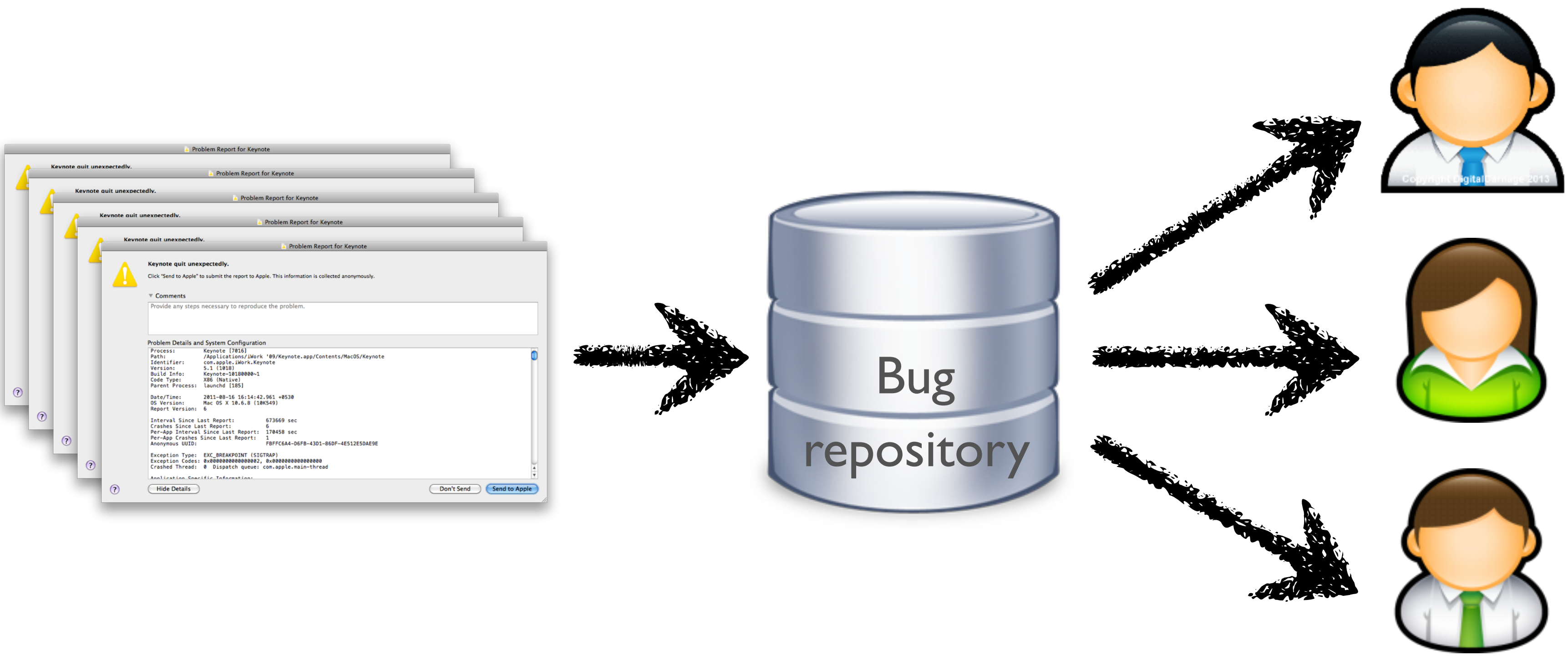
Alessandro (Alex) Orso
School of Computer Science
College of Computing
Georgia Institute of Technology

Partially supported by: NSF, IBM, MSR, and Google


Mail File Edit View Mailbox Message Format Window Help
Mon 4:35 PM
All Mail — alex@gmail (164376 messages, 1182 unread)
MAILBOXES
From
Subject
Date Sent
Problem Report for Keynote
Keynote quit unexpectedly.
Click "Send to Apple" to submit the report to Apple. This information is collected anonymously.
Comments
Provide any steps necessary to reproduce the problem.
Problem Details and System Configuration
Process: Keynote [7016]
Path: /Applications/iWork '09/Keynote.app/Contents/MacOS/Keynote
Identifier: com.apple.iWork.Keynote
Version: 5.1 (1018)
Build Info: Keynote-10180000~1
Code Type: X86 (Native)
Parent Process: launchd [185]
Date/Time: 2011-08-16 16:14:42.961 +0530
OS Version: Mac OS X 10.6.8 (10K549)
Report Version: 6
Interval Since Last Report: 673669 sec
Crashes Since Last Report: 6
Per-App Interval Since Last Report: 170458 sec
Per-App Crashes Since Last Report: 1
Anonymous UUID: FBFFC6A4-D6FB-43D1-86DF-4E512E5DAE9E
Exception Type: EXC_BREAKPOINT (SIGTRAP)
Exception Codes: 0x0000000000000002, 0x0000000000000000
Crashed Thread: 0 Dispatch queue: com.apple.main-thread
Hide Details
Don't Send
Send to Apple

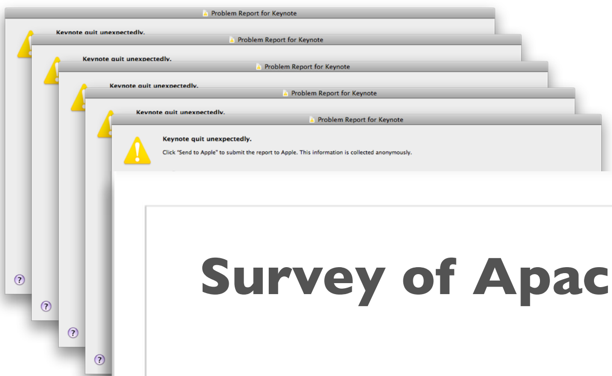

Field failures are unavoidable!
Problem Report for Keynote
Keynote quit unexpectedly.
Click "Send to Apple" to submit the report to Apple. This information is collected anonymously.
Comments
Provide any steps necessary to reproduce the problem.
Problem Details and System Configuration
Hide Details
Don't Send
Send to Apple
All Mail — alex@gmail (164376 messages, 1182 unread)
MAILBOXES
From
Subject
Date Sent
Mail File Edit View Mailbox Message Format Window Help
Mon 4:35 PM

TYPICAL DEBUGGING PROCESS



Very hard to
(1) reproduce
(2) debug

TYPICAL DEBUGGING PROCESS

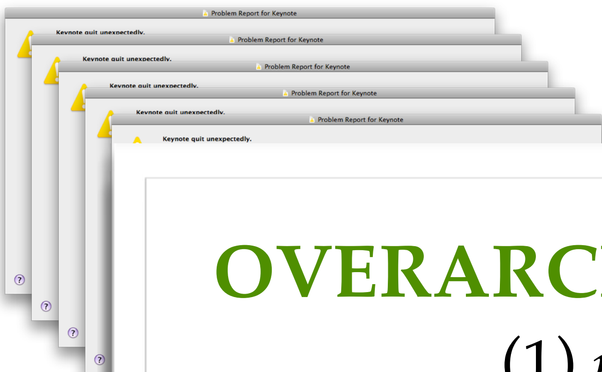


Survey of Apache, Eclipse, and Mozilla developers:

Information on *how to reproduce field failures* is the most valuable, and difficult to obtain, piece of information for investigating such failures.
[Zimmermann10]

Very hard to
(1) reproduce
(2) debug

TYPICAL DEBUGGING PROCESS

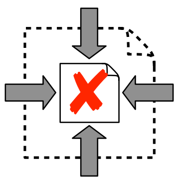


OVERARCHING GOAL: help developers
(1) *investigate* field failures,
(2) *understand* their causes,
(3) *eliminate* such causes,
(4) *prevent* future failures

to
(1) reproduce
(2) debug

OUR WORK SO FAR

Recording and replaying executions
[icsm 2007, icse 2007]



Input minimization
[woda 2006, icse 2007]

Input anonymization
[icse 2011]

Mimicking & explaining field failures
[icse '12, issta '12, issta '13, ase '13, ase '14, icst '14, hvc '16]

Mimicking user behavior
[in progress]

OUR WORK SO FAR

Recording and replaying executions
[icsm 2007, icse 2007]

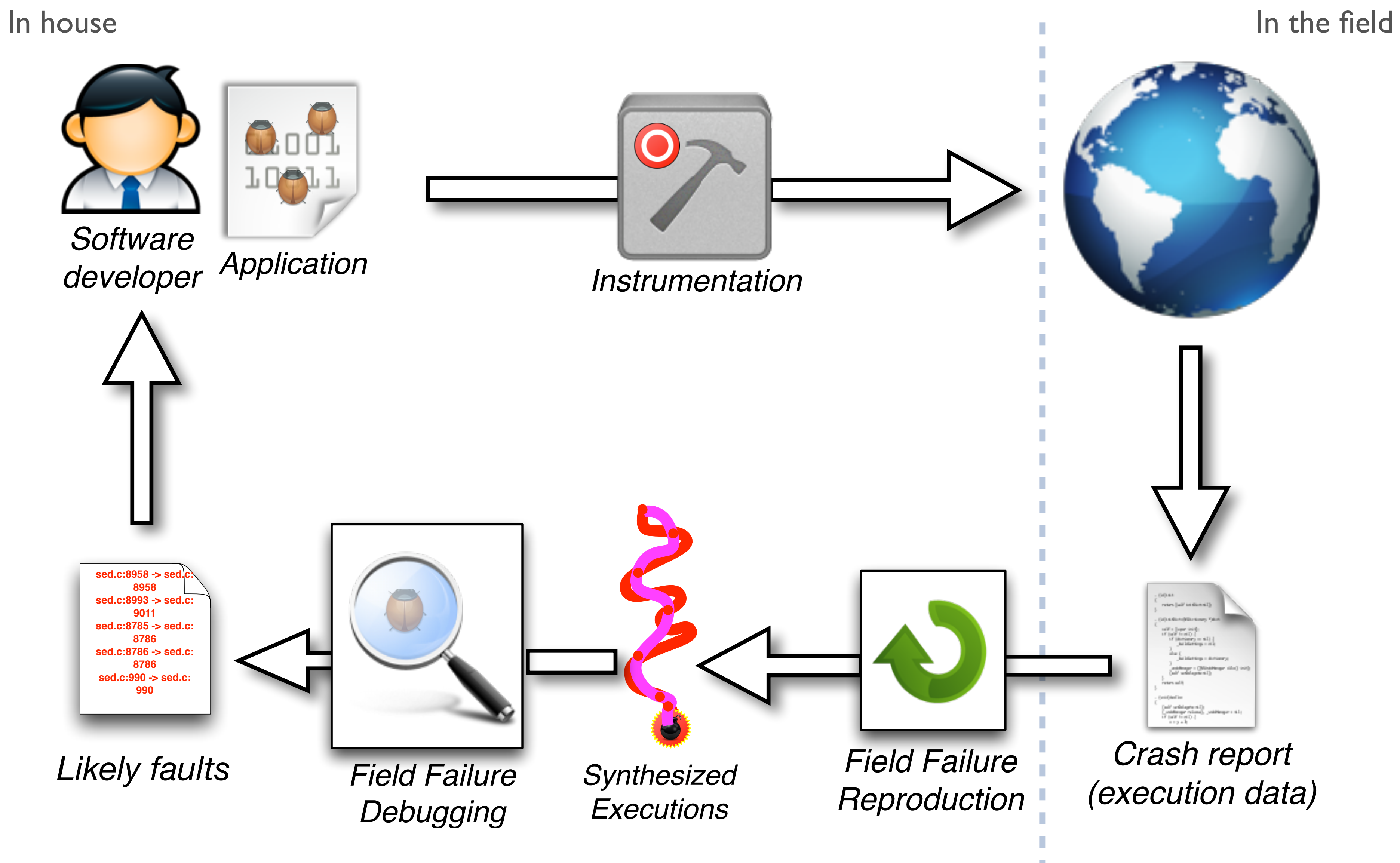
Input minimization
[woda 2006, icse 2007]

Input anonymization
[icse 2011]

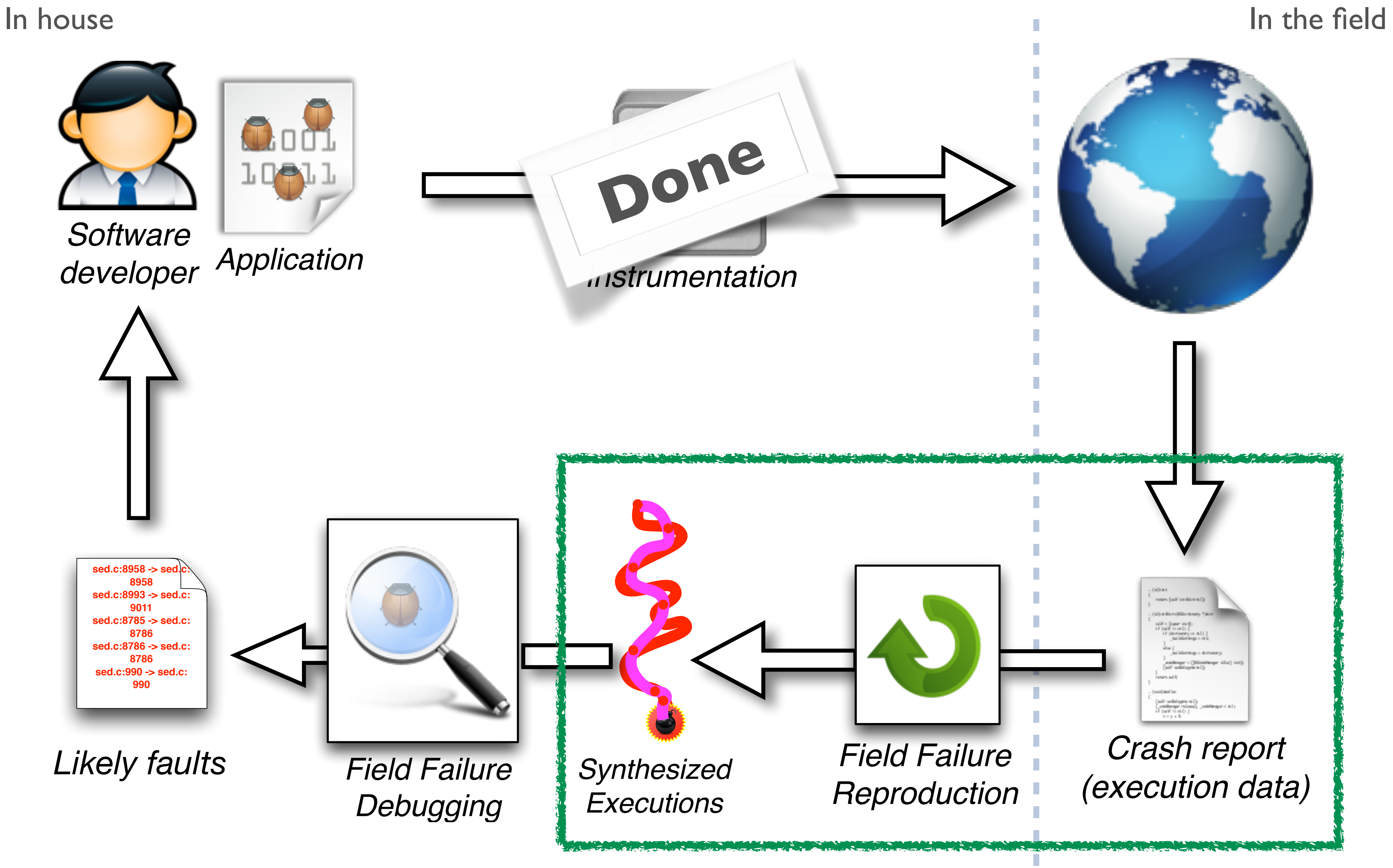
Mimicking & explaining field failures
[icse '12, issta '12, issta '13, ase '13, ase '14, icst '14, hvc '16]

Mimicking user behavior
[in progress]

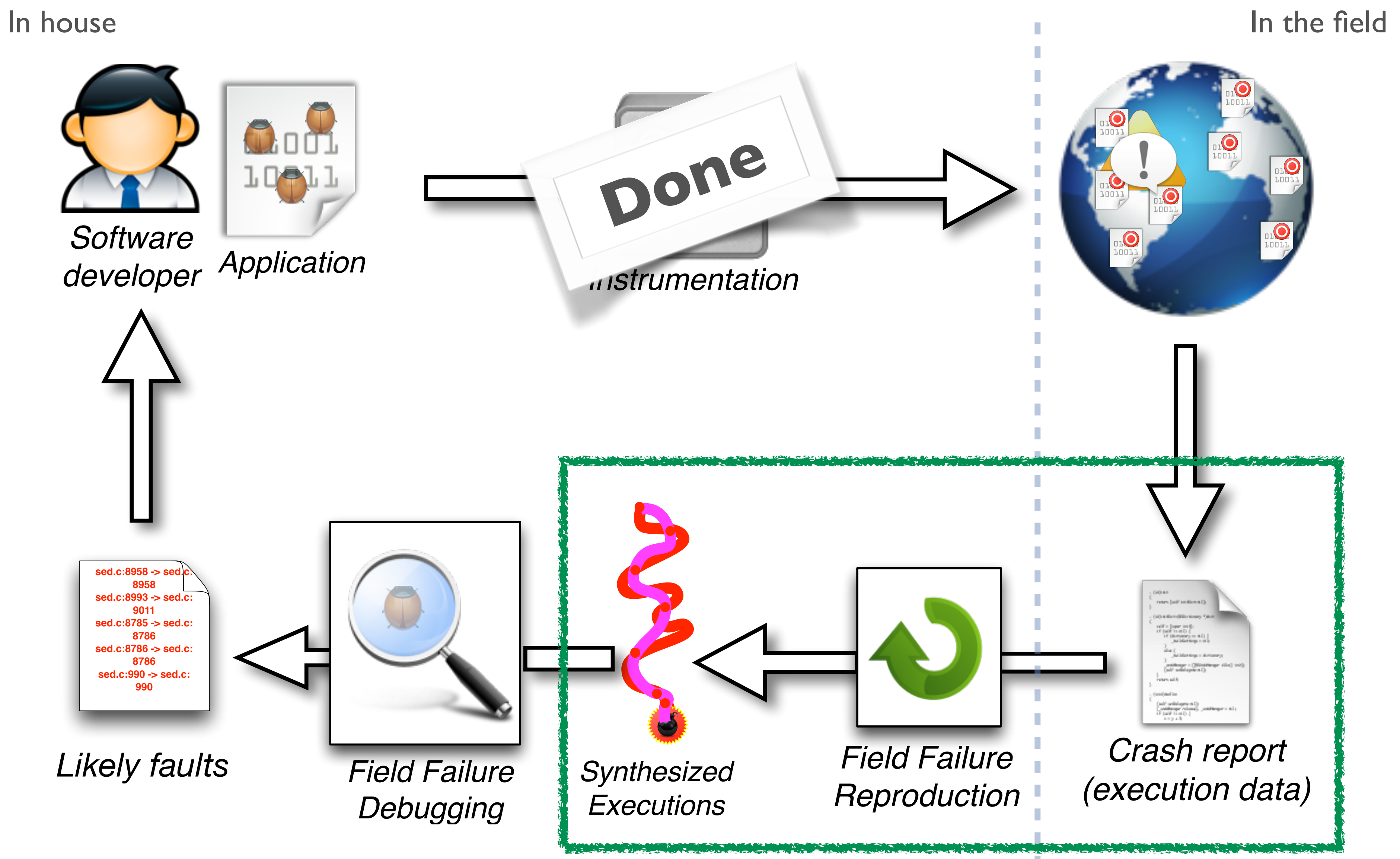
OVERALL VISION



OVERALL VISION

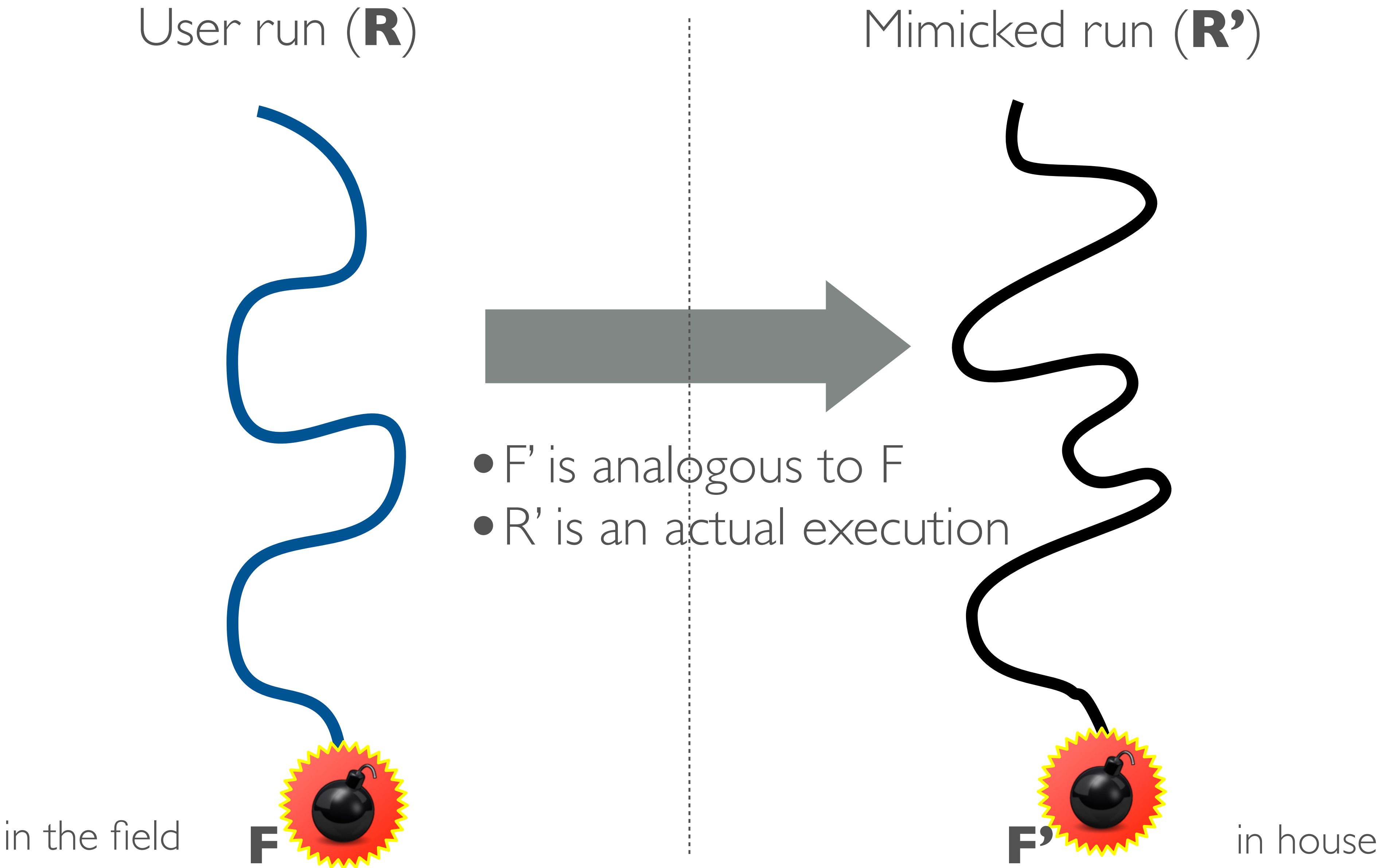


BUGREDUX

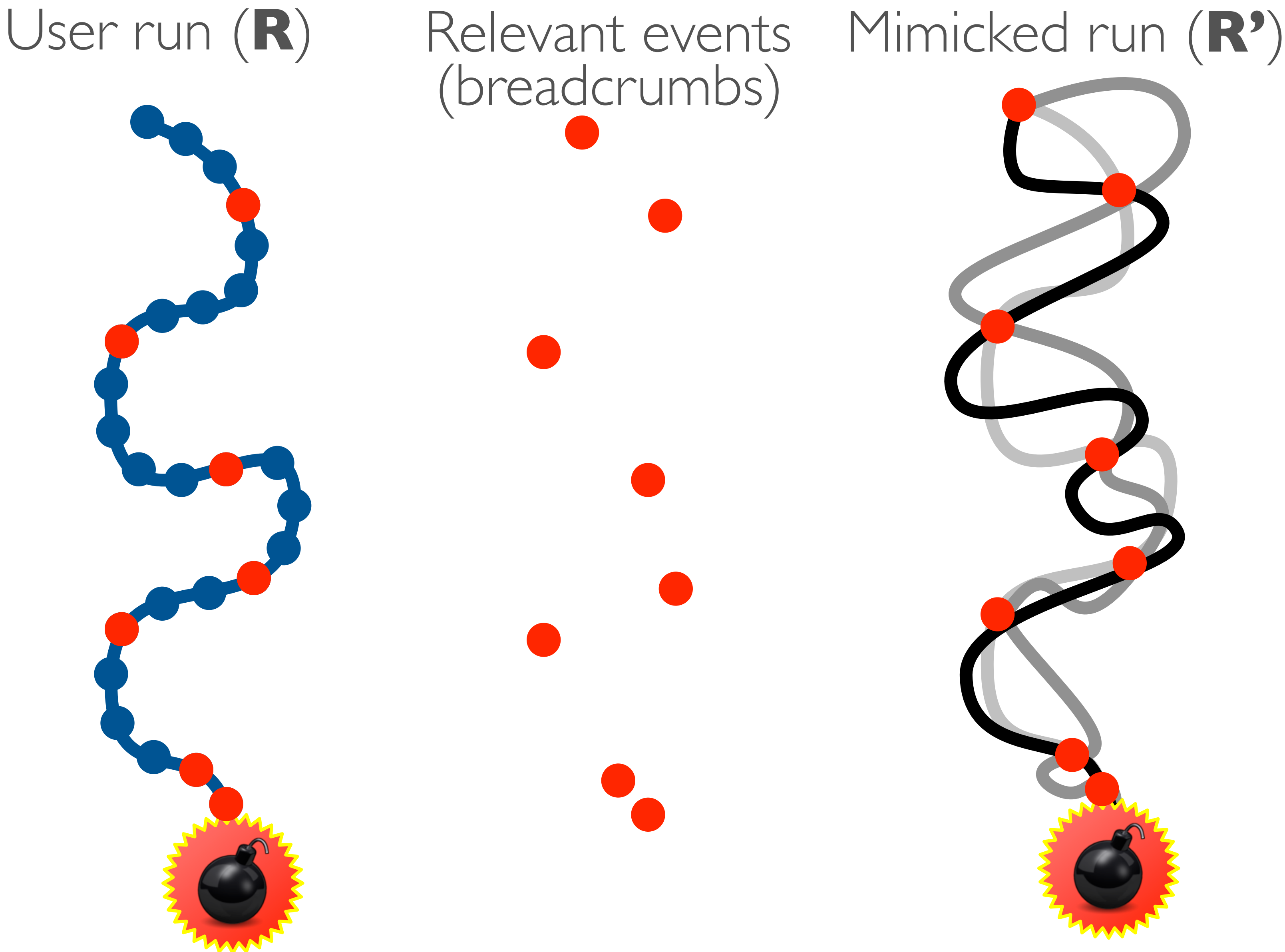


In house
In the field
Software developer
Application
Done
Instrumentation
Crash report (execution data)
Field Failure Reproduction
Synthesized Executions
Field Failure Debugging
sed.c:8958 -> sed.c: 8958
sed.c:8993 -> sed.c: 9011
sed.c:8785 -> sed.c: 8786
sed.c:8786 -> sed.c: 8786
sed.c:990 -> sed.c: 990
Likely faults

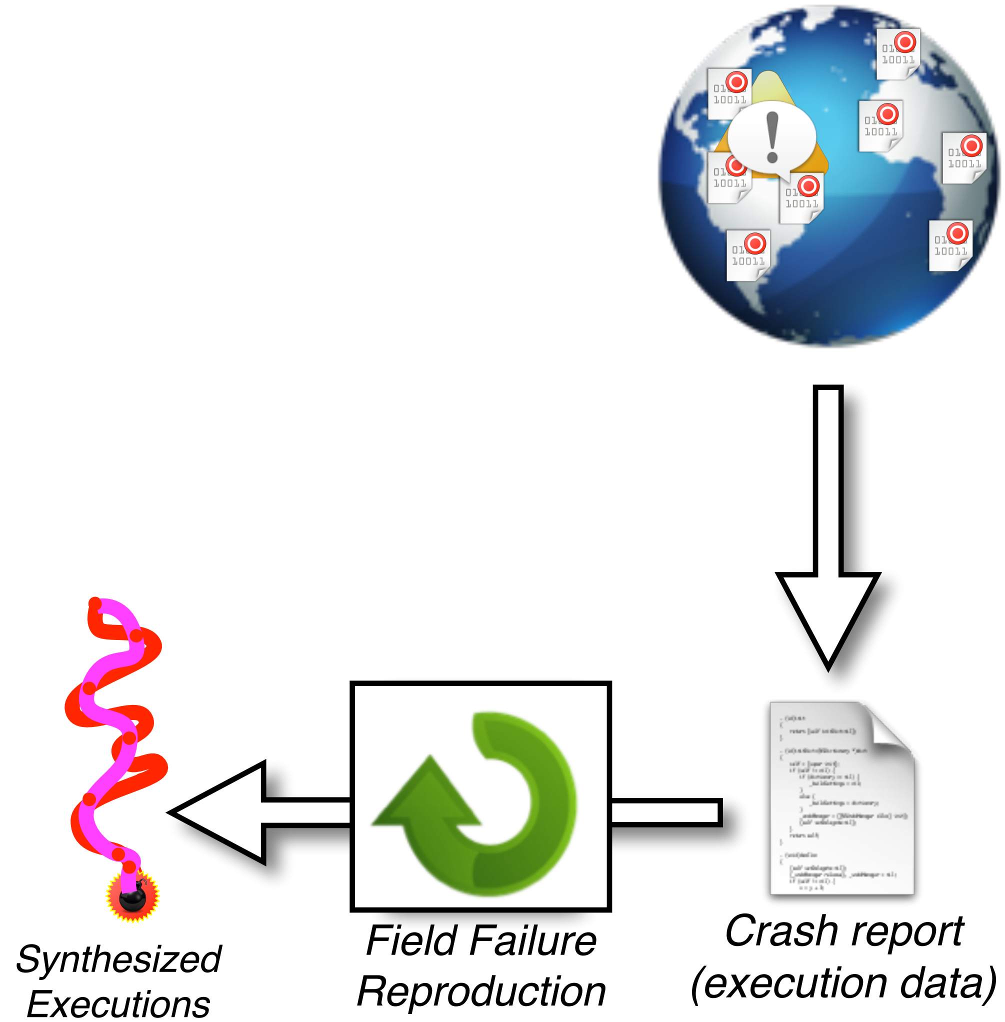
MIMICKING FIELD FAILURES



MIMICKING FIELD FAILURES



BUGREDUX



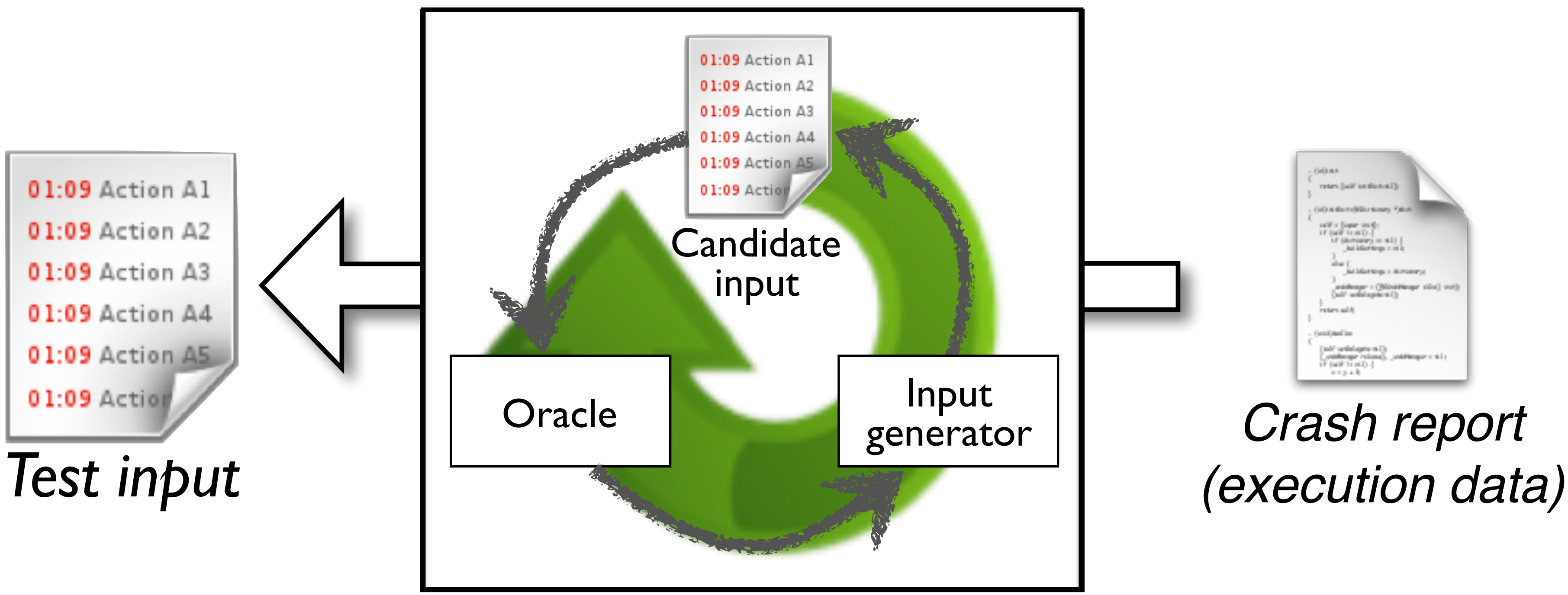
Synthesized Executions
Field Failure Reproduction
Crash report (execution data)

BUGREDUX

BUGREDUX


01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
Test input
Crash report
(execution data)

BUGREDUX



- **Execution data**
  - Point of failure (POF)
  - Failure call stack
  - Call sequence
  - Complete trace
- **Input generation technique**
  - Guided symbolic execution
  - Search-based input generation

SYMBOLIC EXECUTION

SS: $x=x_0$, $y=y_0$, $z=x_0+y_0$

PC: $x_0 > y_0 \wedge x_0+y_0>10$

```
foo (x, y) {
  if(x > y) {
    z = x + y;
    if(z > 10)
      assert false;
}
print("OK");
```

$x_0 = 7$

$y_0 = 4$

solver

<u>Normal execution</u>:

Input: x=4, y=3

Outcome: "OK"

<u>Symbolic execution</u>:

Input: $x=x_0$, $y=y_0$

Outcome:

failure

PC: $x_0 > y_0 \wedge$

$x_0 + y_0 > 10$

ALGORITHM (SIMPLIFIED)

Input

icfg for P

goals (list of code locations)

Output

I_f (candidate input)

Main algorithm

```
init; currGoal = first(goals)
repeat
  currState = SelNextState()
  if (!currState) backtrack or fail
  if (currState.cl == currGoal)
    if (currGoal == last(goals))
      return solve(currState.pc)
    else
      currGoal = next(goals)
      currState.goal = currGoal
  SymbolicallyExecute(currState)
```

statesSet= {<cl, pc, ss, goal>}

SelNextState

```
minDis = ∞
retState = null

foreach state in statesSet
  if (state.goal = currGoal)
    if (state.cl can reach currGoal)
      d = |shortest path state.cl, currGoal|
      if d < minDis
        minDis = d
        retState = state
return retState
```

ALGORITHM (SIMPLIFIED)

Input

icfg for P

goals (list of code locations)

Output

I_f (candidate input)

Optimizations/Heuristics

Dynamic tainting to reduce the symbolic input space

Program analysis information to prune the search space

Some randomness in the shortest path computation

solve(currState.pc)

else

currGoal = next(goals)

currState.goal = currGoal

SymbolicallyExecute(currState)

statesSet= {<cl, pc, ss, goal>}

if (state.cl can reach currGoal)

d = |shortest path state.cl, currGoal|

if d < minDis

minDis = d

retState = state

return retState

EMPIRICAL EVALUATION – RESEARCH QUESTIONS

- **RQ1**:
  Can BugRedux synthesize executions that are able to reproduce field failures?

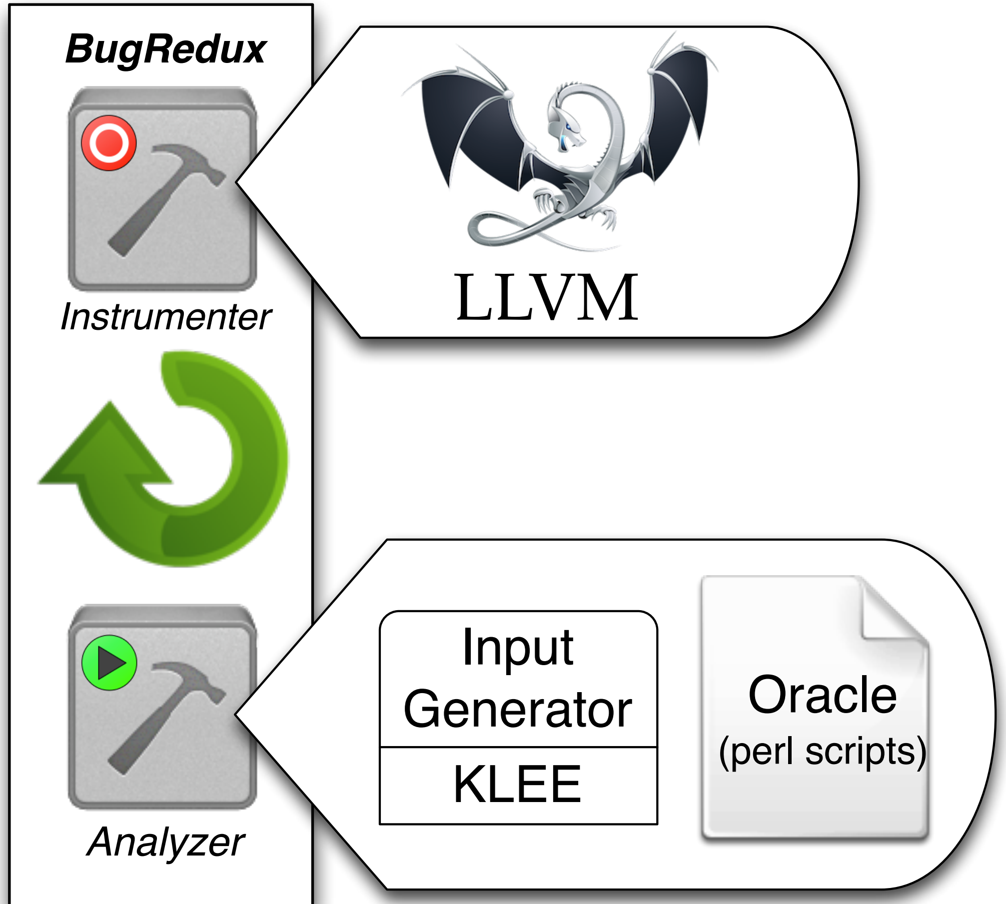
- **RQ2**:
  If so, which types of execution data provide the best cost-benefit tradeoffs?

- In addition, we gathered performance data

EMPIRICAL EVALUATION – BUGREDUX TOOL

Easily customizable!

- Tool



Field data options:

- POF
- Call Stacks
- Call Sequence
- Complete Traces

Oracle:

- inputs P, If, crash report C
- runs P(If), logs any crash C'
- returns fail if no C' or C' != C
- returns success otherwise

- Publicly available: http://www.cc.gatech.edu/~orso/software/bugredux.html

EMPIRICAL EVALUATION – FAILURES CONSIDERED

Name	Repository	Size(KLOC)	# Faults
sed	SIR	14	2
grep	SIR	10	1
gzip	SIR	5	2
ncompress	BugBench	2	1
polymorph	BugBench	1	1
aeon	exploit-db	3	1
glftpd	exploit-db	6	1
htget	exploit-db	3	1
socat	exploit-db	35	1
tipxd	exploit-db	7	1
aspell	exploit-db	0.5	1
exim	exploit-db	241	1
rsync	exploit-db	67	1
xmail	exploit-db	1	1

EMPIRICAL EVALUATION – FAILURES

Only crashing bugs

Name	Repository	Size(KLOC)	# Faults
sed	SIR	14	2
grep	SIR	10	1
gzip	SIR	5	2
ncompress	BugBench	2	1
polymorph	BugBench	1	[illegible]
aeon	exploit-db	[illegible]	[illegible]
glftpd	[illegible]	[illegible]	[illegible]
htget	[illegible]	[illegible]	1
socat	exploit-db	35	1
tipxd	exploit-db	7	1
aspell	exploit-db	0.5	1
exim	exploit-db	241	1
rsync	exploit-db	67	1
xmail	exploit-db	1	1

None of these faults can be discovered by a vanilla KLEE with a timeout of 72 hours

EMPIRICAL EVALUATION – PROTOCOL

For each program P, fault f, and test case t that reveals f

1. While recording time and size of execution data
   a. Run t against P
   b. Run t against P instrumented to collect call sequences
   c. Run t against P instrumented to collect complete traces

2. Run BugRedux with a timeout of 24 hours using POF, call stack, call sequence, and complete trace as execution data
   a. Record whether a candidate I_f is produced
   b. Record whether I_f can reproduce the failure

EMPIRICAL EVALUATION – RESULTS

Name	POF	Call Stack	Call Seq.	Compl.
sed #1				
sed #2				
grep				
gzip #1				
gzip #2				
ncompress				
polymorph				
aeon				
rsync				
glftpd				
htget				
socat				
tipxd				
aspell				
xmail				
exim				

One of three outcomes:

✘: fail

∼: synthesize

✔: (synthesize and) mimic

EMPIRICAL EVALUATION – RESULTS

Name	POF
sed #1	✗
sed #2	✗
grep	✗
gzip #1	✔
gzip #2	~
ncompress	✔
polymorph	✔
aeon	✔
rsync	✗
glftpd	✔
htget	~
socat	✗
tipxd	✔
aspell	~
xmail	✗
exim	✗

Synthesize: 9/16
Mimic: 6/16

EMPIRICAL EVALUATION – RESULTS

Name	Call Stack
sed #1	✘
sed #2	✘
grep	~
gzip #1	✔
gzip #2	~
ncompress	✔
polymorph	✔
aeon	✔
rsync	✘
glftpd	✔
htget	~
socat	✘
tipxd	✔
aspell	~
xmail	✘
exim	✘

Synthesize: 10/16
Mimic: 6/16

EMPIRICAL EVALUATION – RESULTS

Name		Call Seq.
sed #1		✔
sed #2		✔
grep		✔
gzip #1		✔
gzip #2		✔
ncompress		✔
polymorph		✔
aeon		✔
rsync		✔
glftpd		✔
htget		✔
socat		✔
tipxd		✔
aspell		✔
xmail		✔
exim		✔

Synthesize: 16/16
Mimic: 16/16

EMPIRICAL EVALUATION – RESULTS

Name	Compl.
sed #1	✗
sed #2	✗
grep	✗
gzip #1	✗
gzip #2	✗
ncompress	✗
polymorph	✗
aeon	✓
rsync	✗
glftpd	✗
htget	✗
socat	✗
tipxd	✗
aspell	✗
xmail	✗
exim	✓

Synthesize: 2/16
Mimic: 2/16

EMPIRICAL EVALUATION – RESULTS

Name	Compl.
sed #1	✗
sed #2	✗
grep	✗
gzip #1	✗
gzip #2	✗
ncompress	✗
polymorph	✗
aeon	✔
rsync	✗
glftpd	✗
htget	✗
socat	✗
tipxd	✗
aspell	✗
xmail	✗
exim	✔

Synthesize: 2/16
Mimic: 2/16

- Divergence due to lib modeling
- Limitations of constraint solver

EMPIRICAL EVALUATION – DISCUSSION

- **RQ1**
  Can BugRedux synthesize executions that are able to reproduce field failures?
  YES

- **RQ2**
  If so, which types of execution data provide the best cost-benefit tradeoffs?
  Call sequences

- **Observations**
  - [Manual examination] Faults can be distant from the failure points, so POFs and call stacks are unlikely to help
  - More information may not be always better
  - Call sequences work well, but provide a great deal of information
  - BugRedux can generate multiple mimicked executions (pass & fail)

EMPIRICAL EVALUATION – DISCUSSION

- **RQ1**
  Can BugRedux synthesize executions that are able to reproduce field failures?
  YES

- **RQ2**
  If so, which types of execution data provide the best cost-benefit tradeoffs?
  Call sequences

- **Observations**

Performance:
Average overhead for call-sequence collection: 15%
(unoptimized implementation)

  - BugRedux can generate multiple mimicked executions (pass & fail)

EMPIRICAL EVALUATION – DISCUSSION

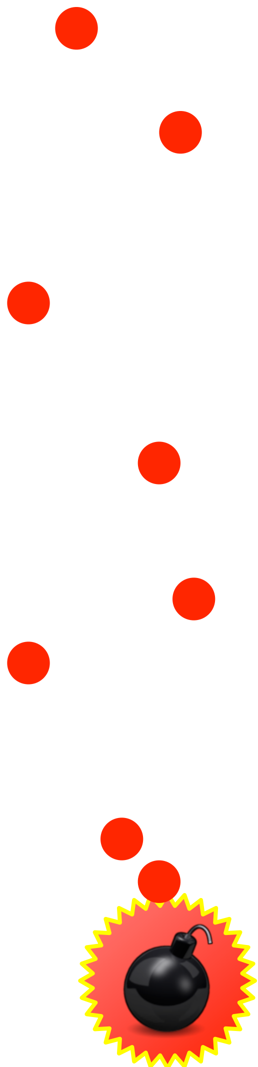
- **RQ1**
  Can BugRedux synthesize executions that are able to reproduce field failures?
  YES

- **RQ2**
  If so, which types of execution data provide the best cost-benefit tradeoffs?
  Call sequences

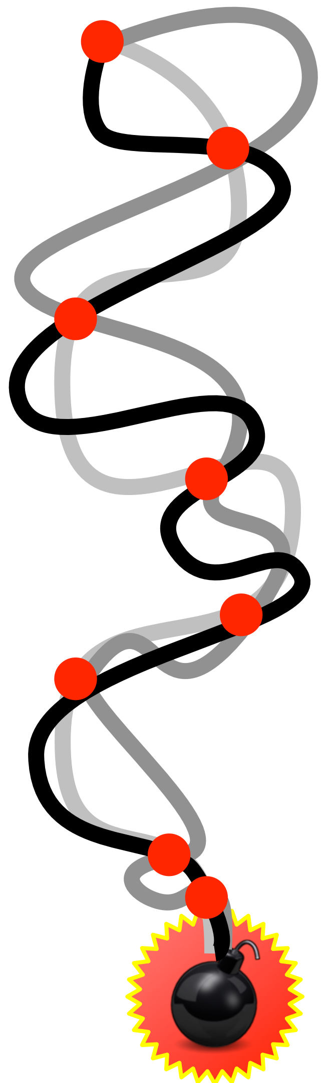
- **Observations**
  - [Manual examination] Faults can be distant from the failure points, so POFs and call stacks are unlikely to help
  - More information may not be always better
  - Call sequences work well, but provide a great deal of information
  - BugRedux can generate multiple mimicked executions (pass & fail)

MINIMIZING CALL SEQUENCES

Relevant events (breadcrumbs)



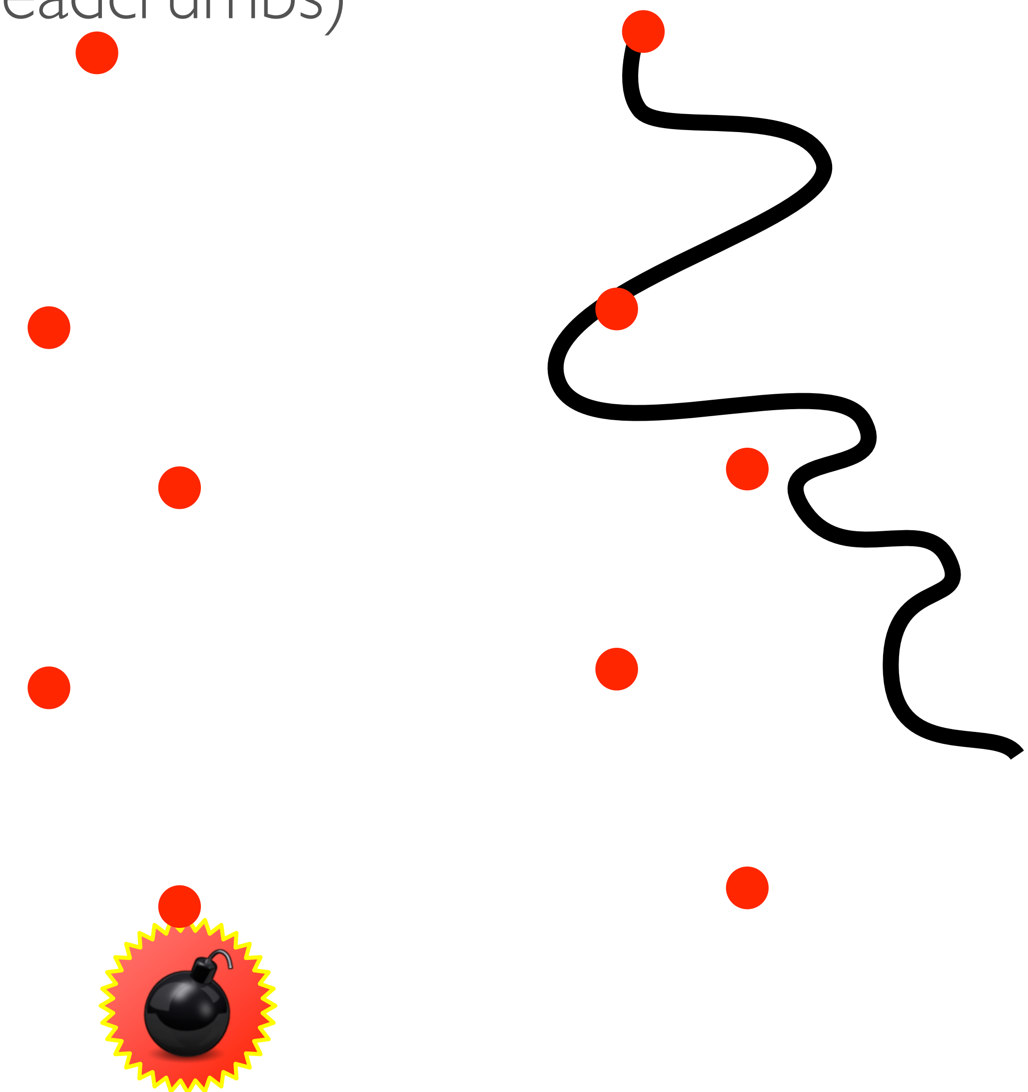
Mimicked run



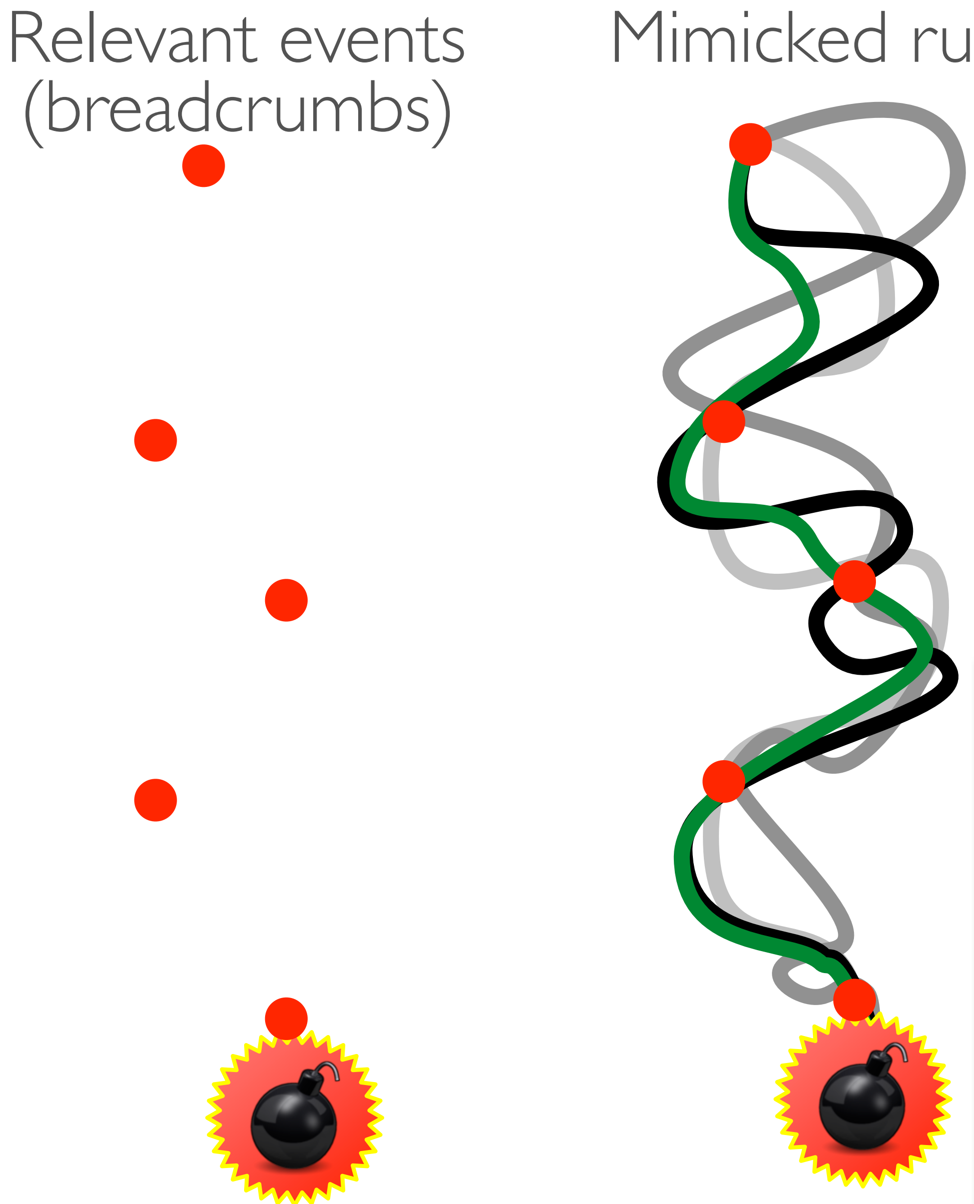
MINIMIZING CALL SEQUENCES

Relevant events (breadcrumbs)

Mimicked run



MINIMIZING CALL SEQUENCES



Mini study

- for each entry e
  - remove e from sequence
    - if BugRedux " generates a failure" ➡ continue
    - else add back e

MINIMIZING CALL SEQUENCES – RESULTS

Name	Original Length	Minimal Length
sed.fault1	73	12
sed.fault2	146	7
grep	31	2
xmail	1142	363
gzip.fault2	27	2
rysnc	23	2
aspell	516	256
socat	62	3
htget	25	2
exim	1029	326

MINIMIZING CALL SEQUENCES – RESULTS

Name	Original Length	Minimal Length
sed.fault1	72	12
htget	25	2
exim	1029	326

Summary

1. On average, only 16% of entries in the original call sequence are required to reproduce the failures–in some cases, as little as 2!

2. The number of entries needed increases with the complexity of the input that triggers the faults.

MINIMIZING CALL SEQUENCES – RESULTS

Name	Original Length	Minimal Length
sed-fault1	72	
ht…	25	2
exim	1029	326

Summary

1. On average, only … sequence are … ome cases, a… … increases with the … at triggers the faults.

Preliminary Conclusion

It seems possible to recreate observed failure with only limited (and inexpensive to collect) information

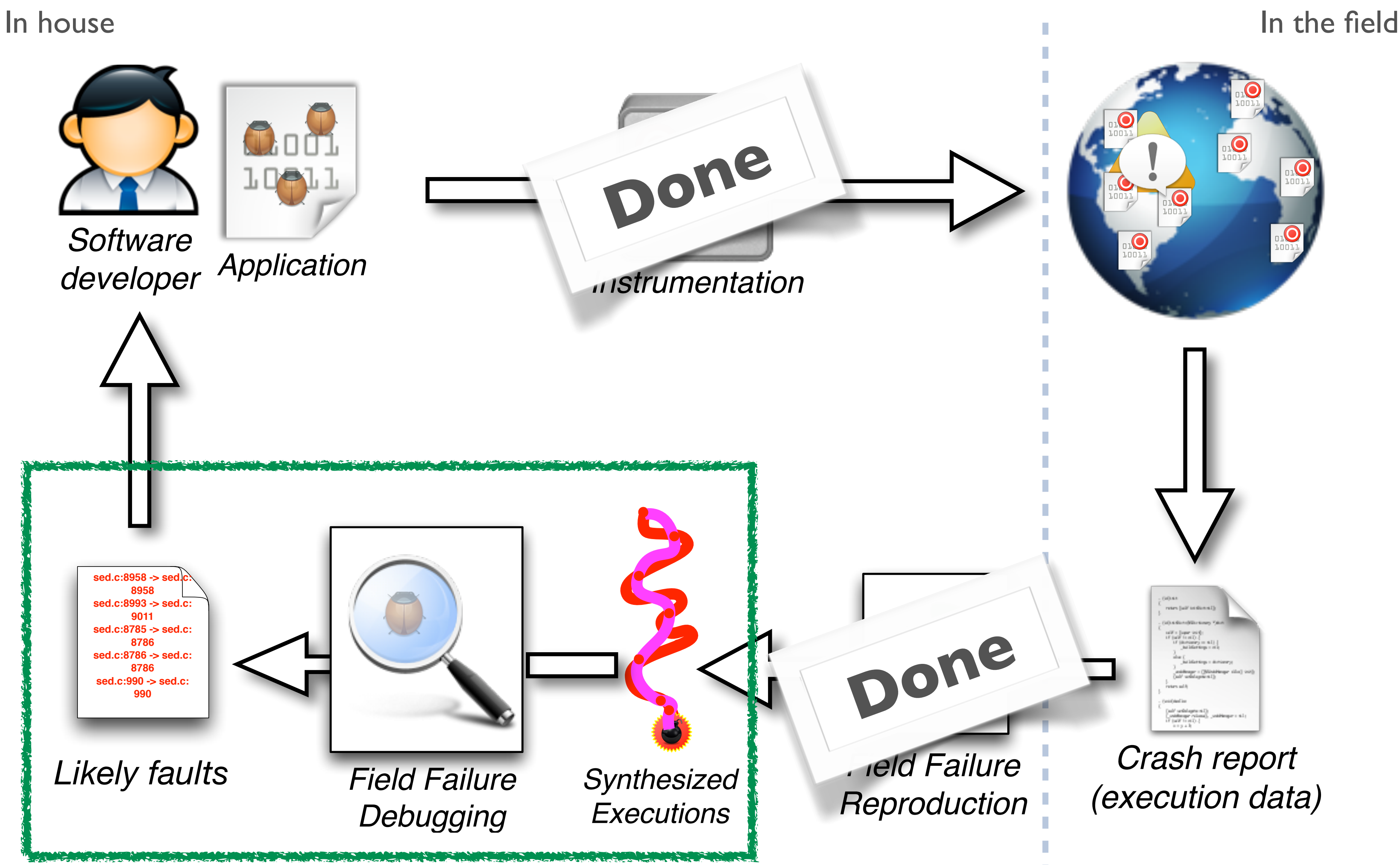
EMPIRICAL EVALUATION – DISCUSSION

- **RQ1**
  Can BugRedux synthesize executions that are able to reproduce field failures?
  YES

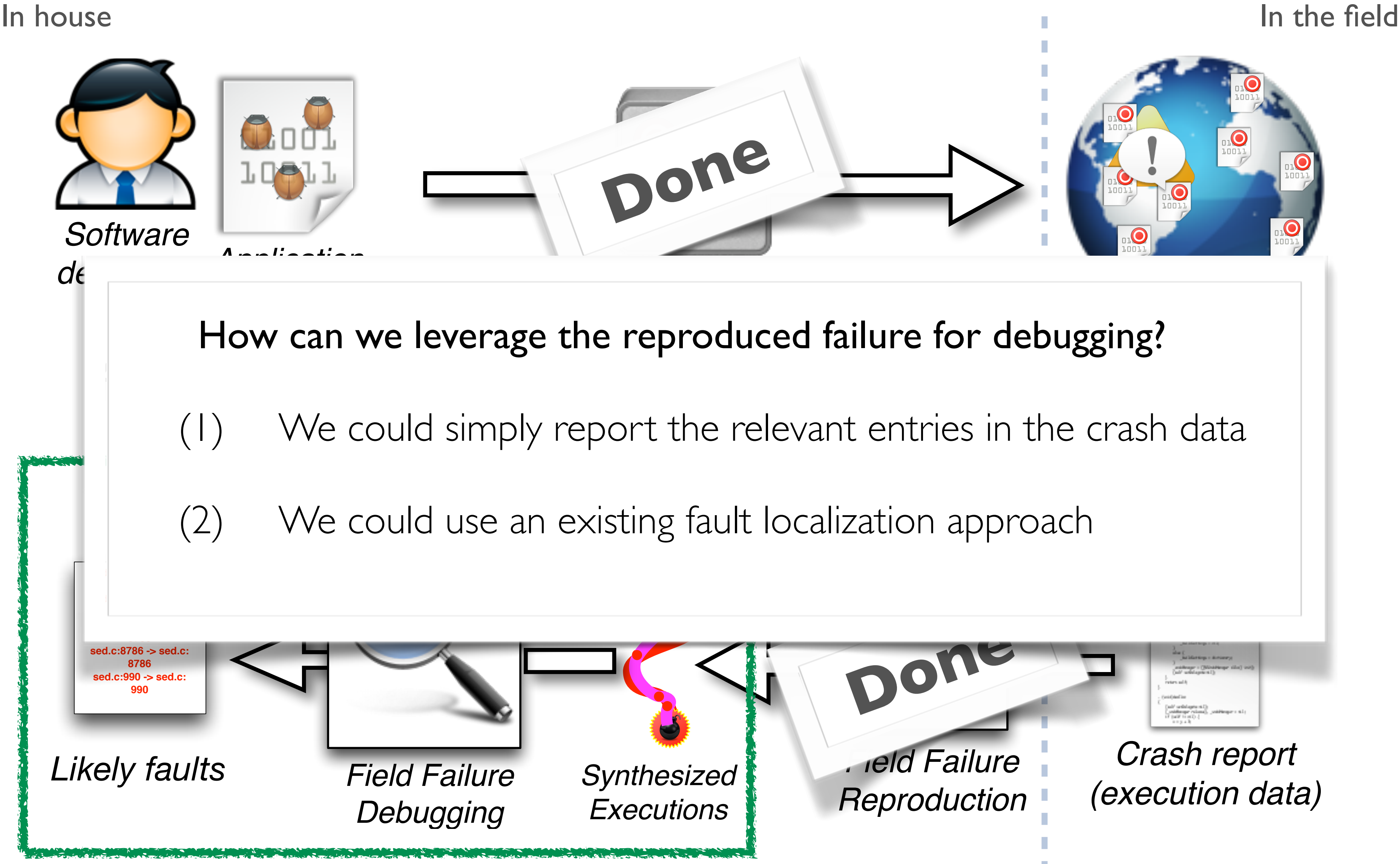
- **RQ2**
  If so, which types of execution data provide the best cost-benefit tradeoffs?
  Call sequences

- **Observations**
  - [Manual examination] Faults can be distant from the failure points, so POFs and call stacks are unlikely to help
  - More information may not be always better
  - Call sequences work well, but provide a great deal of information
  - BugRedux can generate multiple mimicked executions (pass & fail)

OVERALL VISION



OVERALL VISION

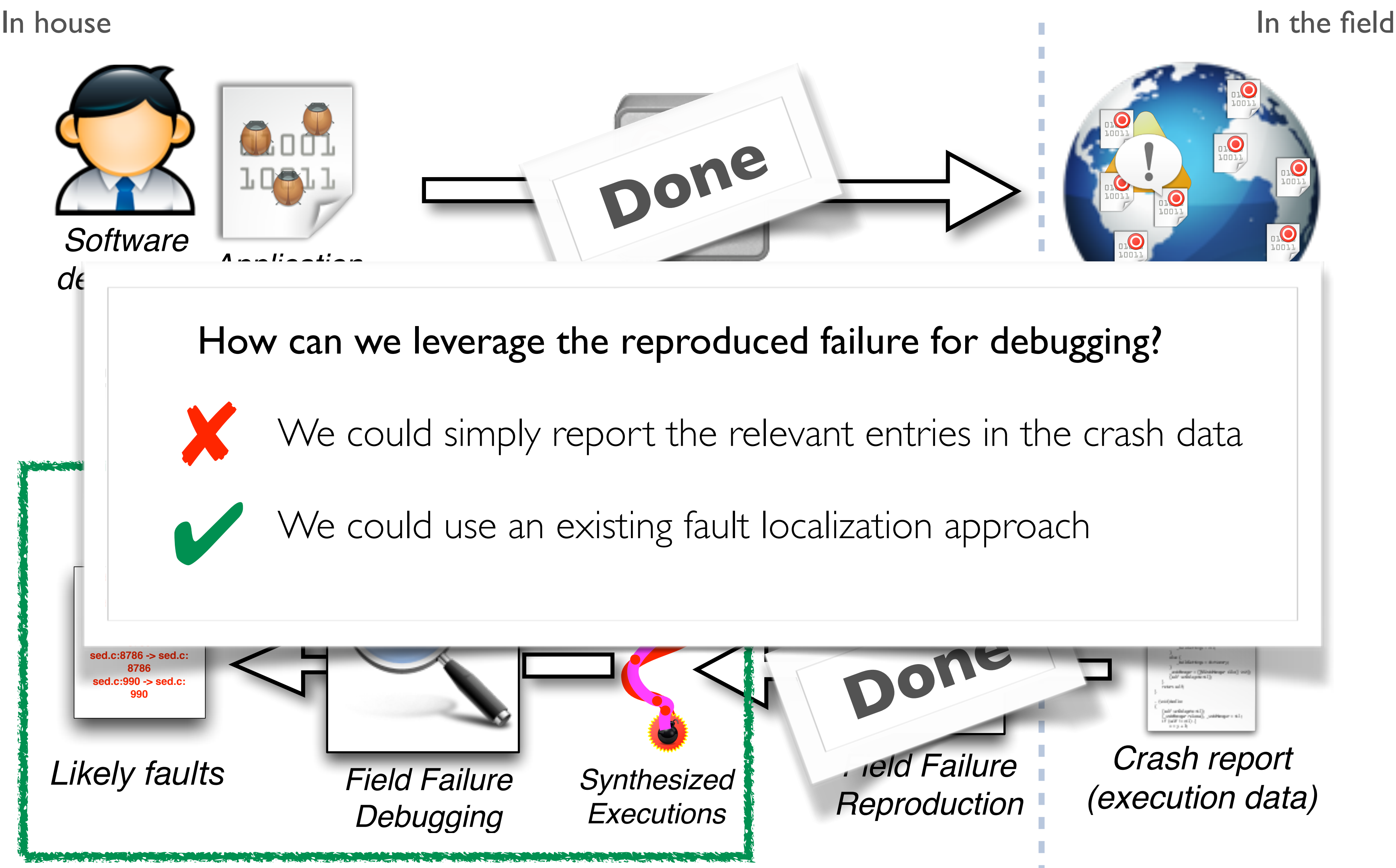


How can we leverage the reproduced failure for debugging?

(1) We could simply report the relevant entries in the crash data

(2) We could use an existing fault localization approach

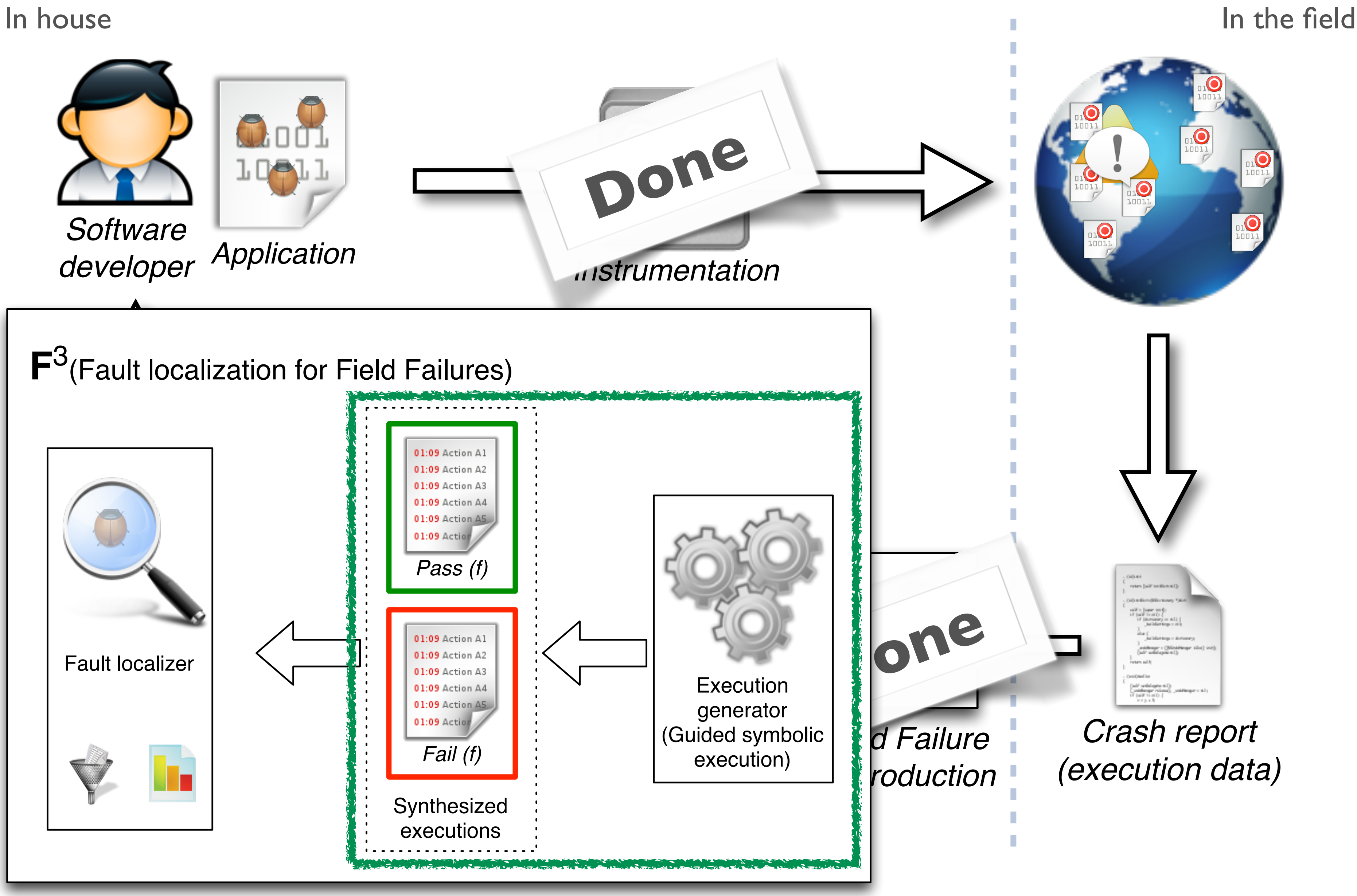
OVERALL VISION



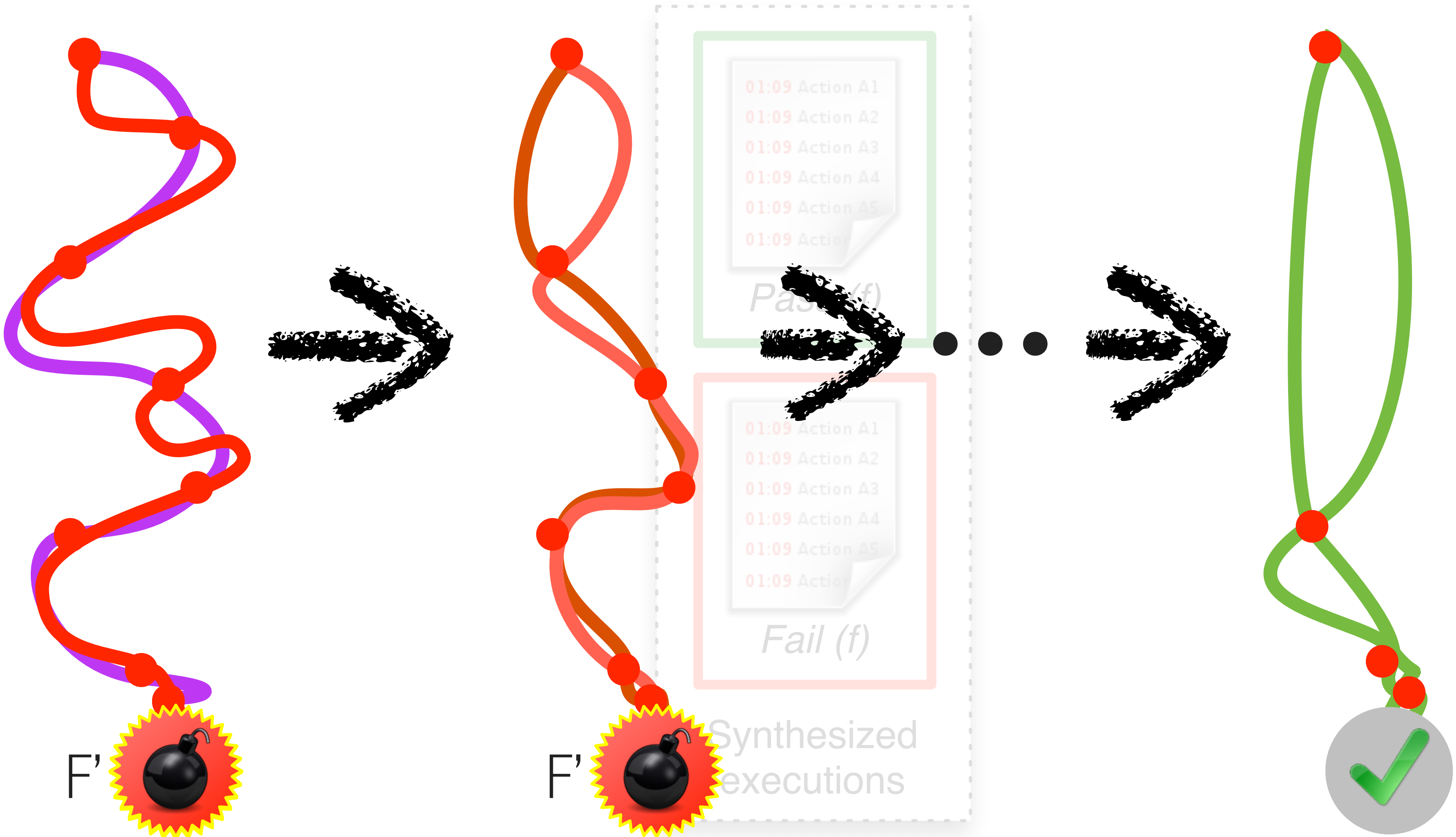
How can we leverage the reproduced failure for debugging?

- ✗ We could simply report the relevant entries in the crash data
- ✓ We could use an existing fault localization approach

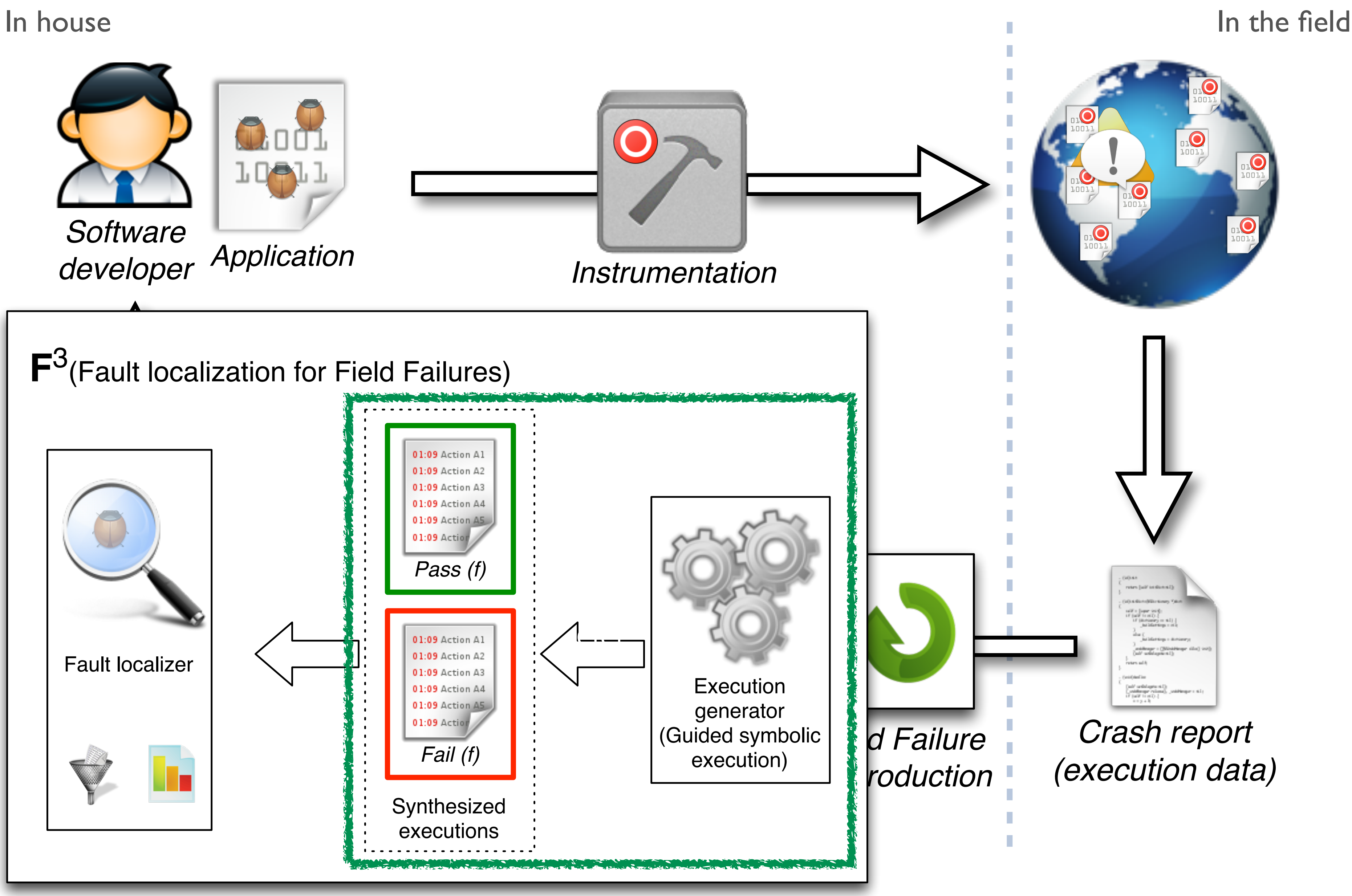
F3


In house
In the field
Software developer
Application
Done
nstrumentation
F^3(Fault localization for Field Failures)
Fault localizer
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
Pass (f)
Fail (f)
Synthesized executions
Execution generator (Guided symbolic execution)
one
d Failure
roduction
Crash report (execution data)

GENERATING MULTIPLE EXECUTIONS



F3


In house
In the field
Software developer
Application
Instrumentation
F3(Fault localization for Field Failures)
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
Pass (f)
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
Fail (f)
Synthesized executions
Execution generator (Guided symbolic execution)
Fault localizer
d Failure
roduction
Crash report (execution data)

EMPIRICAL STUDY

- **RQ1**: Can F^3 **synthesize multiple** passing and failing executions for a given set of crash data?
- **RQ2**: Can F^3 use these synthesized executions to **perform fault localization** effectively?
- **RQ3**: Do our **optimizations** actually improve the effectiveness of fault localization and, if so, to what extent?

GENERATED EXECUTIONS (RQ1)

Faults	# Failing	# Passing
exim	598	4
xmail	303	1001
sed.fault2	54	30
sed.fault1	1017	296
grep	567	137
aspell	134	10
htget	44	210
gzip.fault2	5	27
socat	46	5
rsync	156	2576

executions generated using original crash data

executions generated using reduced crash data

executions generated using an empty list

GENERATED EXECUTIONS (RQ1)

Faults	# Failing	# Passing
exim	598	4
xmail	303	1001
sed.fault2	54	30
gzip.fault2	5	27
socat	46	5
rsync	156	2576

RQ1: Can F^3 synthesize multiple passing and failing executions for a given set of crash data?
Yes

executions generated using original crash data
executions generated using reduced crash data
executions generated using an empty list

FAULT LOCALIZATION RESULT (RQ2)

Faults	Ochiai+	
	# Suspicious Entities	Rank of Real Fault
exim	3	1
xmail	3	1
sed.fault2	11	1
sed.fault1	19	13
grep	72	12
aspell	0 / 45	NA / 1
htget	0 / 93	NA / 1
gzip.fault2	80	3
socat	14	11
rsync	28	6

FAULT LOCALIZATION RESULT (RQ2)

Faults	Ochiai+	
	# Suspicious Entities	Rank of Real Fault
exim	3	1
xmail	3	1
sed.fault2	11	1
sed.fault1	19	13
grep	72	12
aspell	0 / 45	NA / 1
htget	0 / 93	NA / 1
gzip.fault2	80	3
socat	14	11
rsync	28	6

Worst-case scenario

FAULT LOCALIZATION RESULT (RQ2)

Faults	Ochiai+		Best Case
	# Suspicious Entities	Rank of Real Fault	
exim	3	1	1
xmail	3	1	1
sed.fault2	11	1	1
sed.fault1	19	13	3
grep	72	12	12
aspell	0 / 45	NA / 1	1
htget	0 / 93	NA / 1	1
gzip.fault2	80	3	3
socat	14	11	6
rsync	28	6	1

Worst-case scenario

FAULT LOCALIZATION RESULT (RQ2)

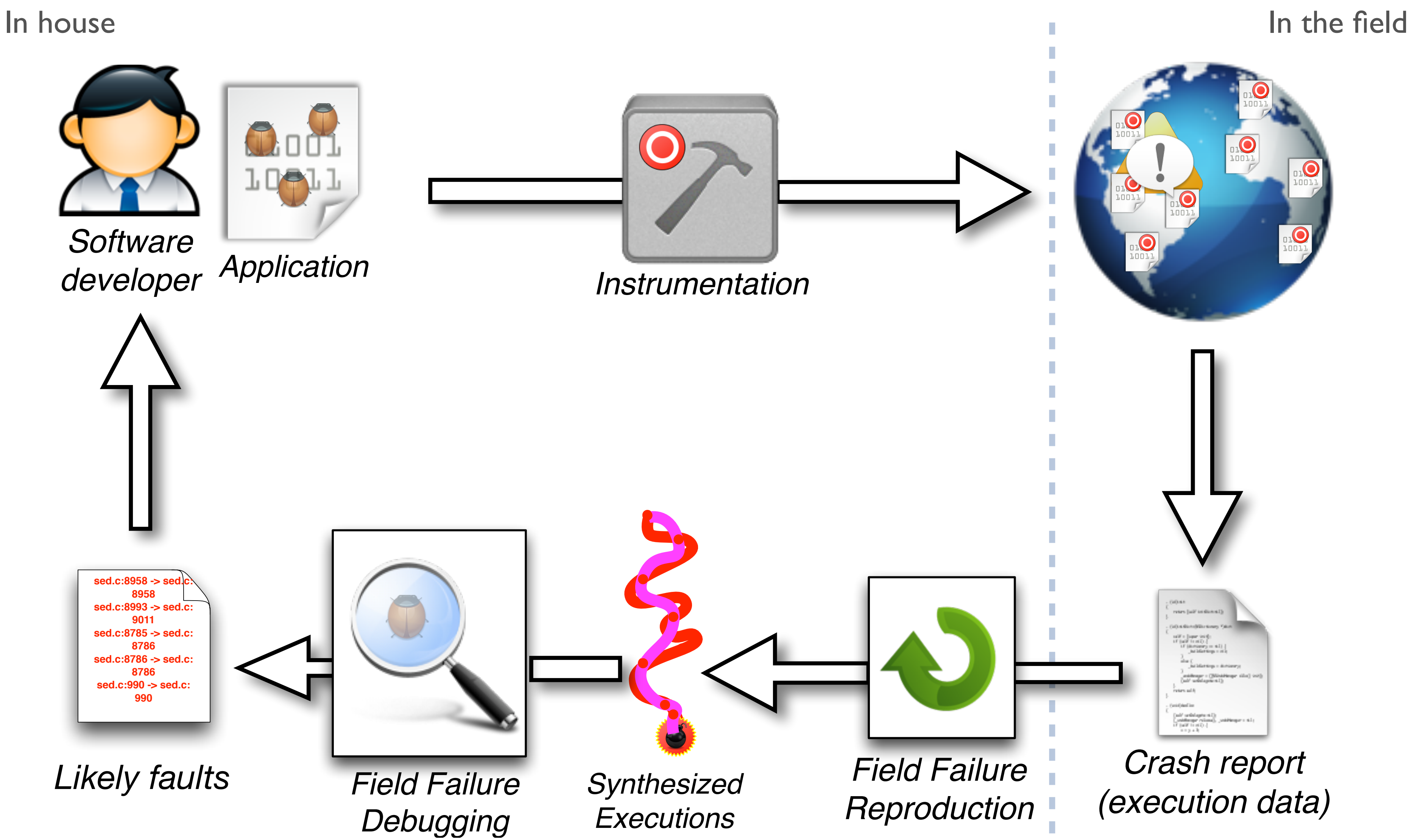
Faults	Ochiai+		Best Case
	# Suspicious Entities	Rank of Real Fault	
exim	3	1	1
xmail	3	1	1
socat	14	11	6
rsync	28	6	1

Worst-case scenario

RQ2: Can F^3 use these synthesized executions to perform fault localization effectively?

Top ranked for 5 faults, within 15 for all others

CURRENT AND FUTURE WORK



Testing is rarely representative

Erroneous assumptions

Limited resources

Well-known but not well-studied problem

Debugging

Executions

Testing is rarely representative

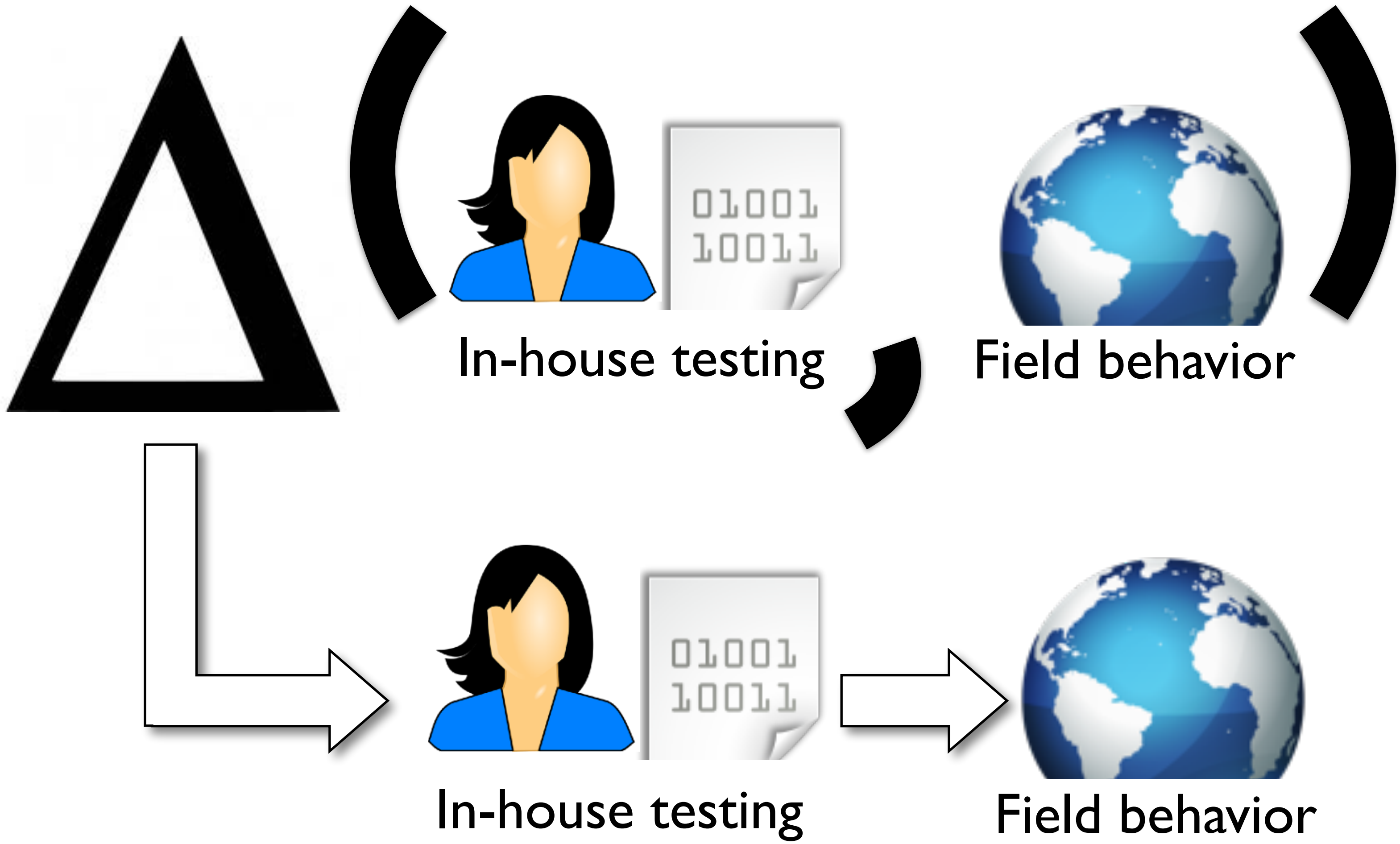
Erroneous assumptions

Limited resources

Need to bridge the gap between field executions and in-house tests

Debugging *Executions* *Reproduction* *(execution data)*

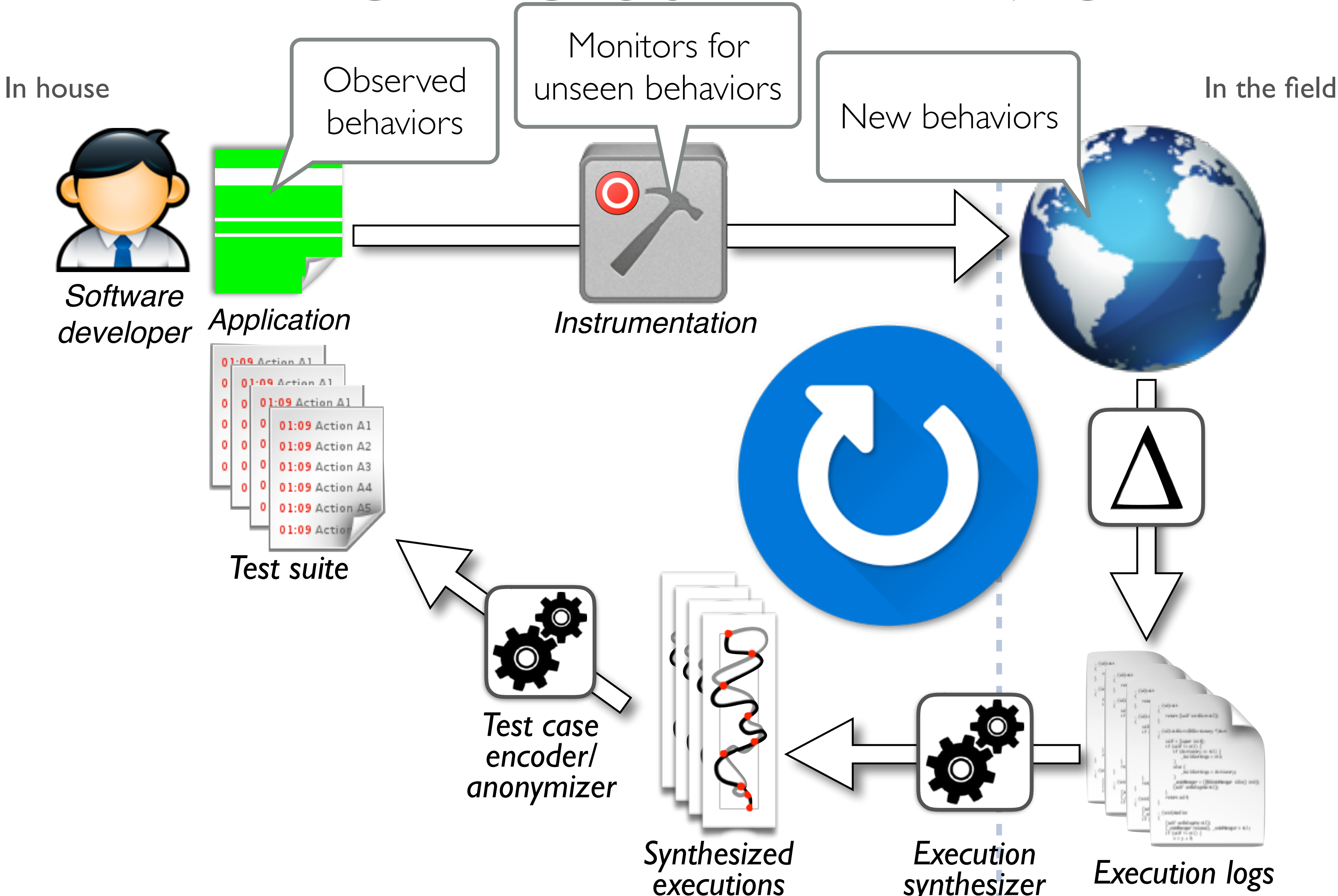
Two main steps



Debugging

Executions

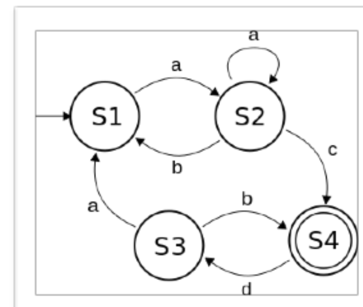
MIMICKING USER BEHAVIOR



MIMICKING USER BEHAVIOR

PROJECT STATUS

- **Behavioral difference detection**



- **Execution synthesis**

Synthesized executions

Execution synthesizer

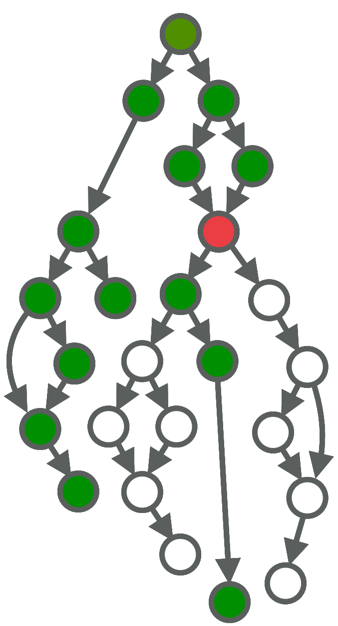
Execution logs

TWO KLEE-RELATED BYPRODUCTS

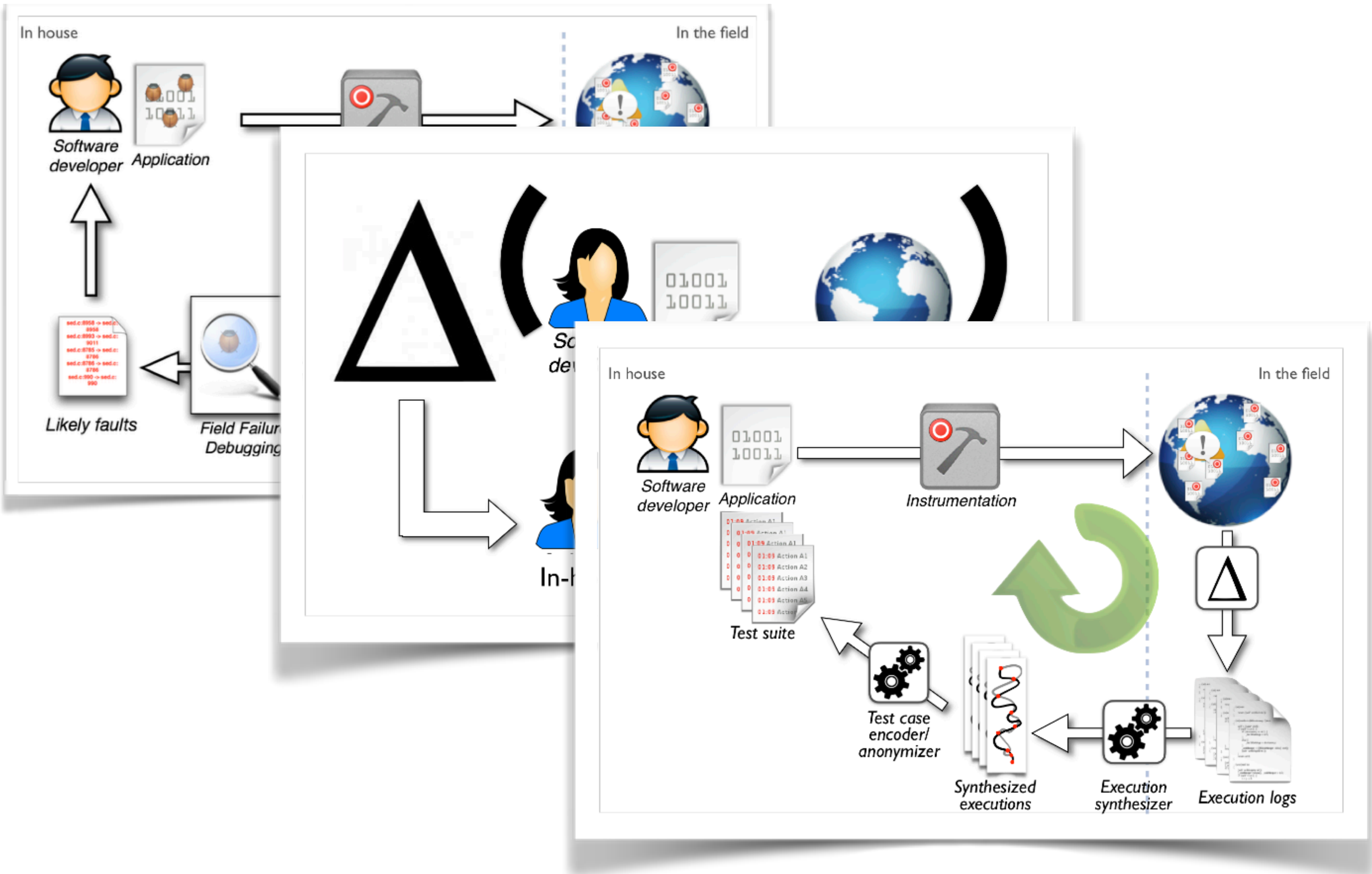
- **String-enabled Klee**

KLEE + Z3str3

- **Local Symbolic Execution**



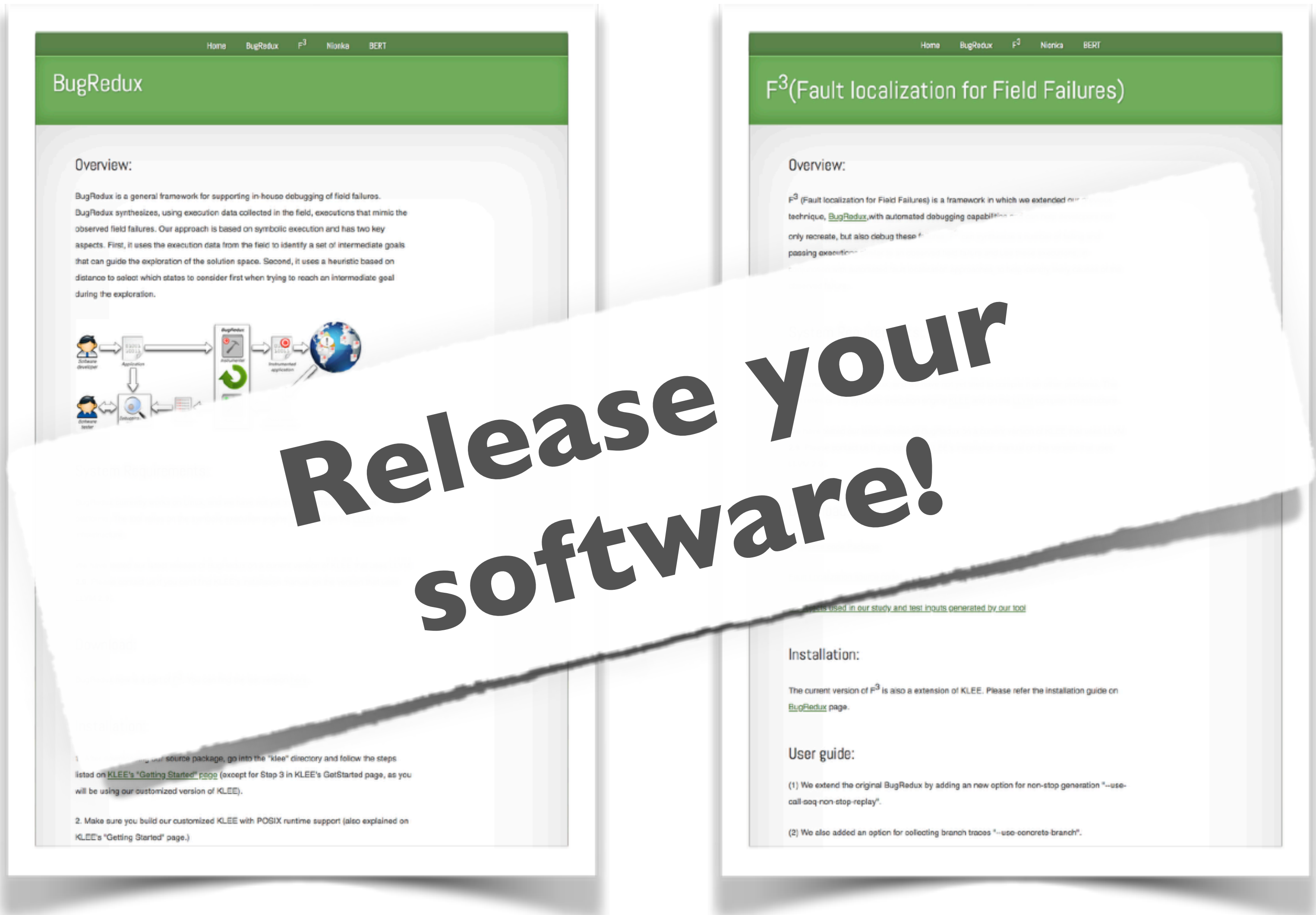
CONCLUSION

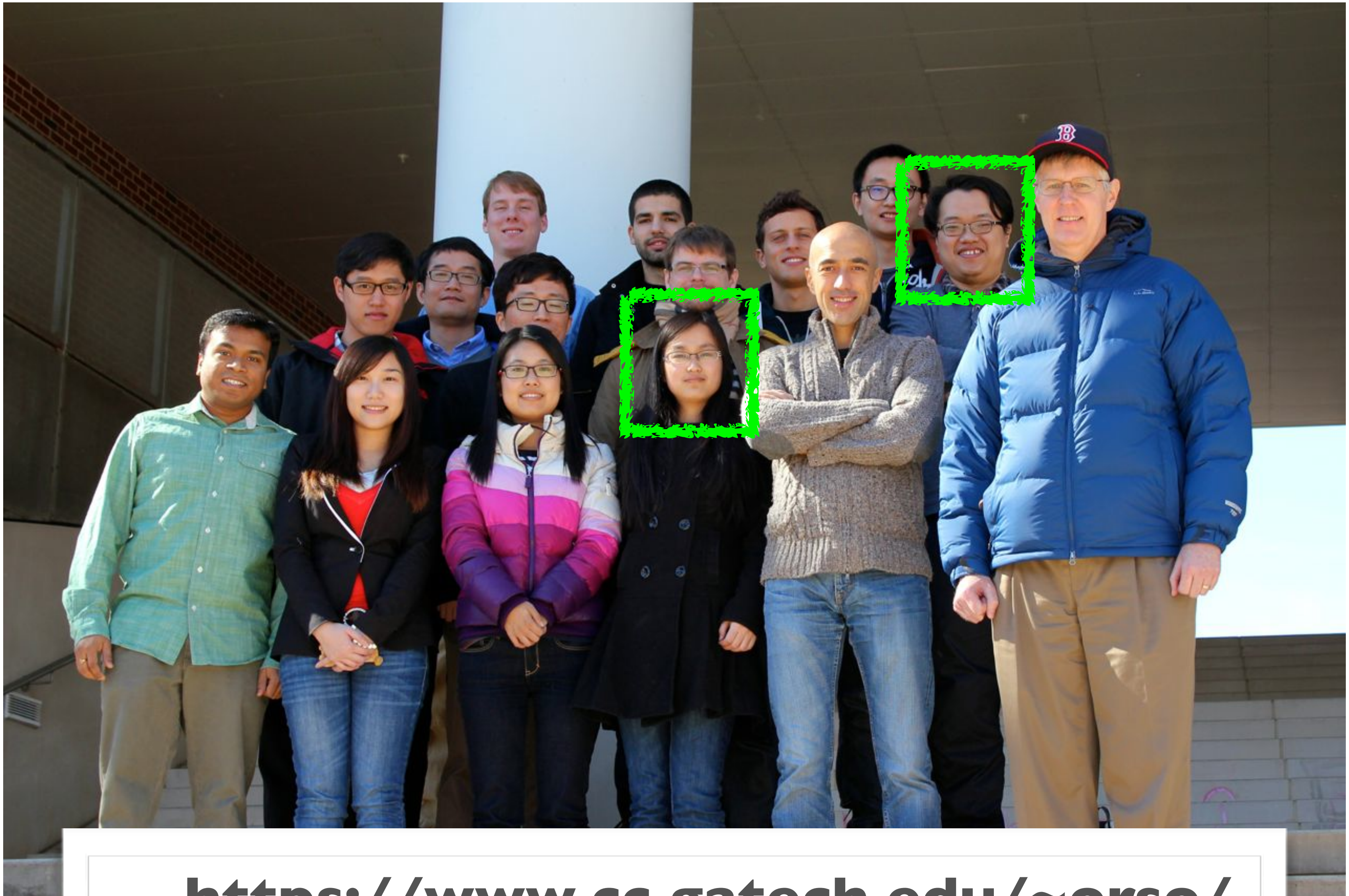

In house
In the field
Software developer
Application
Likely faults
Instrumentation
Test suite
Test case encoder/ anonymizer
Synthesized executions
Execution synthesizer
Execution logs
01001
10011

CONCLUSION


In house
In the field
Software developer
Application
Likely faults
In the field
encoder/ anonymizer
Synthesized executions
Execution synthesizer
Execution logs

META CONCLUSION





https://www.cc.gatech.edu/~orso/