Università della Svizzera italiana

Faculty of Informatics

DIPARTIMENTO DI INFORMATICA SISTEMISTICA E COMUNICAZIONE

Reproducing Concurrency Failures from Crash Stacks



Francesco A. Bianchi* | Mauro Pezzè*◇ | **Valerio Terragni***

* USI Università della Svizzera italiana, Switzerland

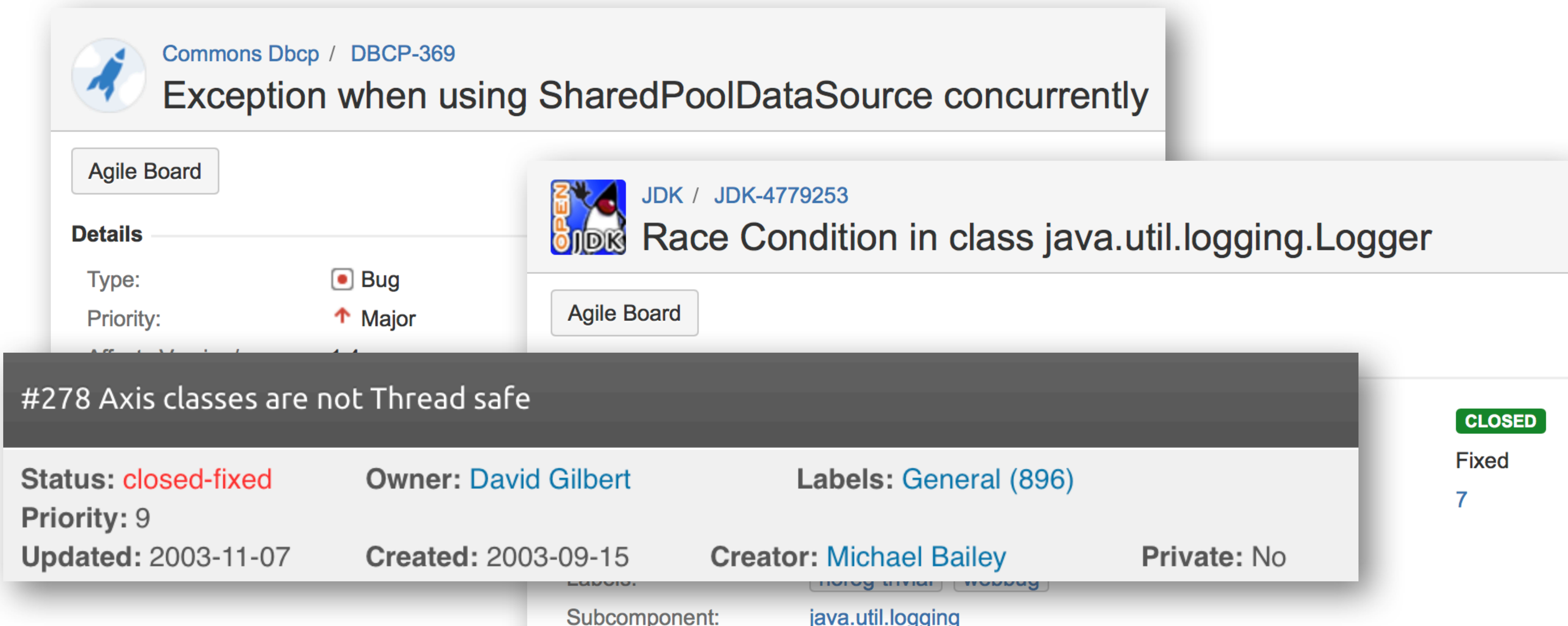
◇ Università di Milano Bicocca, Italy

ESEC/FSE 2017

Introduction

OUR GOAL

Automated reproduction of
concurrency failures manifested in the field



Reproducing Concurrency Failures

Why is it important?

Ease understanding and fixing the related concurrency fault

Difficult problem!

What is needed?

A failure-inducing

test code and **thread interleaving**

test code: runnable piece of code that exercises the program under test

thread interleaving: temporal order of shared memory accesses

State of The Art

Technique	Input	Output	
		Test code	Interleaving
ODR [Altekar SOSP '09] **LEAP** [Huang FSE '10] **CLAP** [Huang PLDI '13] **CARE** [Jiang ICSE '14] **Cortex** [Machado PPoPP '16] **STRIDE** [Zhou ICSE '12]	Execution trace	✗	✓
ESD [Zamfir EuroSys '10] Weeratunge ASPLOS '10	Memory core-dumps	✗	✓

Privacy concerns
Overhead issues
Hard to obtain in the field

State of The Art

Technique	Input	Output	
		Test code	Interleaving
ODR [Altekar SOSP '09] **LEAP** [Huang FSE '10] **CLAP** [Huang PLDI '13] **CARE** [Jiang ICSE '14] **Cortex** [Machado PPoPP '16] **STRIDE** [Zhou ICSE '12]	Execution trace	✗	✓
ESD [Zamfir EuroSys '10] Weeratunge ASPLOS '10	Memory core-dumps	✗	✓
ConCrash **(our contribution)**	**Crash stack**	✓	✓

Less privacy concerns
No overhead issues
Easily obtainable in the field

ConCrash Targets Thread-safe Classes

“A class that encapsulates synchronizations that ensure a correct behavior when the same instance of the class is accessed from multiple threads”

Crash Stack

JDK-4779253 : Race Condition in class java.util.logging.Logger

type of exception

Point Of Failure (POF)

```
java.lang.NullPointerException
  at java.util.logging.Logger.log(Logger.java:421)
  at java.util.logging.Logger.doLog(Logger.java:458)
  at java.util.Logging.Logger.log(Logger.java:482)
  at java.util.logging.Logger.info(Logger.java:996)
```

Example of Thread-safety Violation

JDK-4779253 : Race Condition in class java.util.logging.Logger

Thread 1

```
public void log(LogRecord r) {
 synchronized(this) {
  if(filter != null) {
     if(!filter.isLoggable(r)) {
      return;
     }
   }
  }
}
```

Point Of Failure (POF)

Thread 2

```
public void setFilter(Filter f) {
 this.filter = f;
}
```

```
= null
```

failure-inducing interleaving

Concurrent Test Code

JDK-4779253 : Race Condition in class java.util.logging.Logger

Sequential Prefix

```
Logger sout = Logger.getAnonymousLogger();
MyFilter myFilter0 = new MyFilter();
sout.setFilter(myFilter0);
```

Thread 1 / Thread 2

Concurrent Suffixes

```
sout.info("");                    sout.setFilter(null);
```

Set of method call sequences that exercise the public interface of a class from multiple threads.

Challenge

Crash Stacks provides only limited information on how to generate a failure-inducing test code

Crash Stack

```
java.lang.NullPointerException
  at java.util.logging.Logger.log(Logger.java:421)
  at java.util.logging.Logger.doLog(Logger.java:458)
  at java.util.Logging.Logger.log(Logger.java:482)
  at java.util.logging.Logger.info(Logger.java:996)
```

Crashing method and Class Under Test (CUT)

Failure-inducing Test Code

```
Logger sout = Logger.getAnonymousLogger();
MyFilter myFilter0 = new MyFilter();
sout.setFilter(myFilter0);
```

CUT

Sequential Prefix

Thread 1

```
sout.info("");
```

Thread 2

```
sout.setFilter(null);
```

Crashing Method

Input Parameter

Interfering Method

Challenge

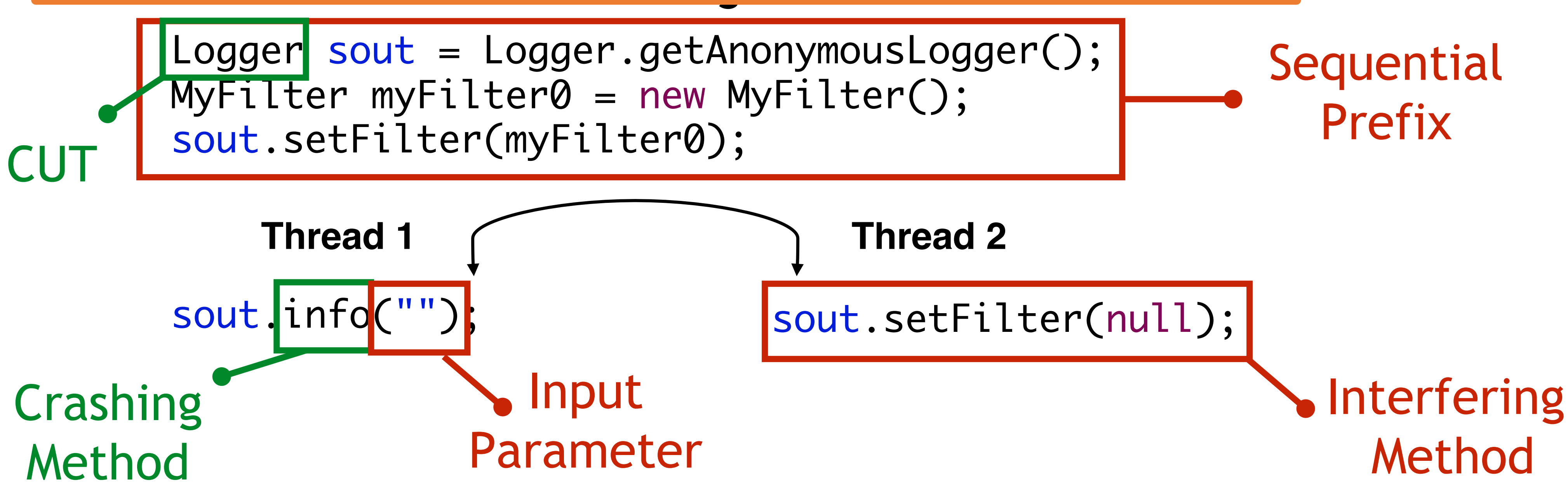
Crash Stacks provides only limited information on how to generate a failure-inducing test code

Crash Stack

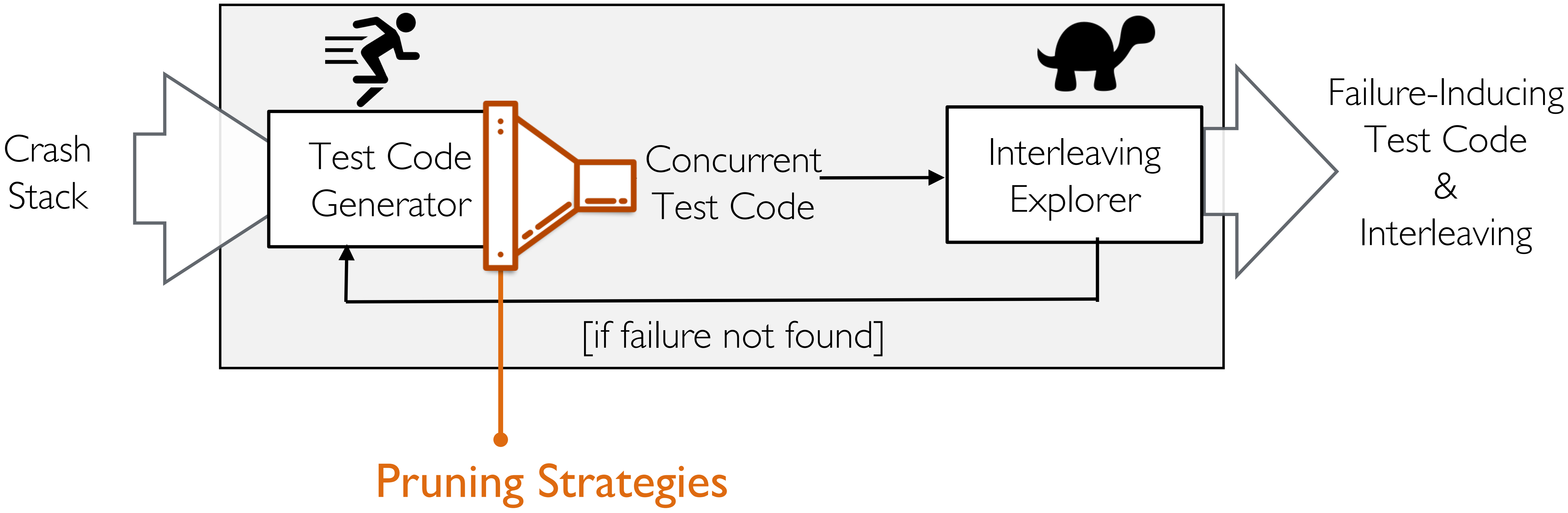
```
java.lang.NullPointerException
  at java.util.logging.Logger.log(Logger.java:421)
```

Implication:

The search space of candidate failure-inducing test codes is very huge

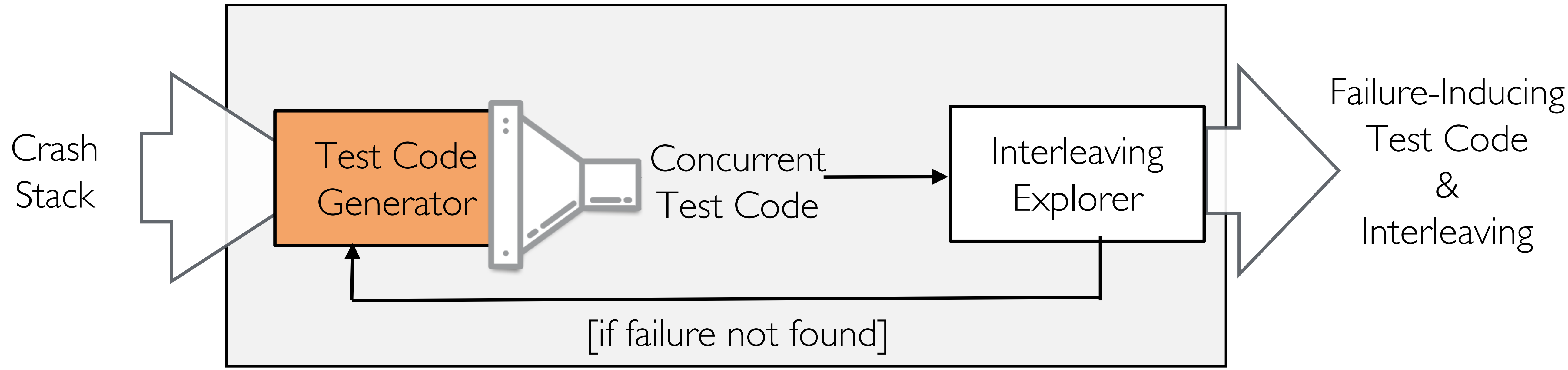


ConCrash



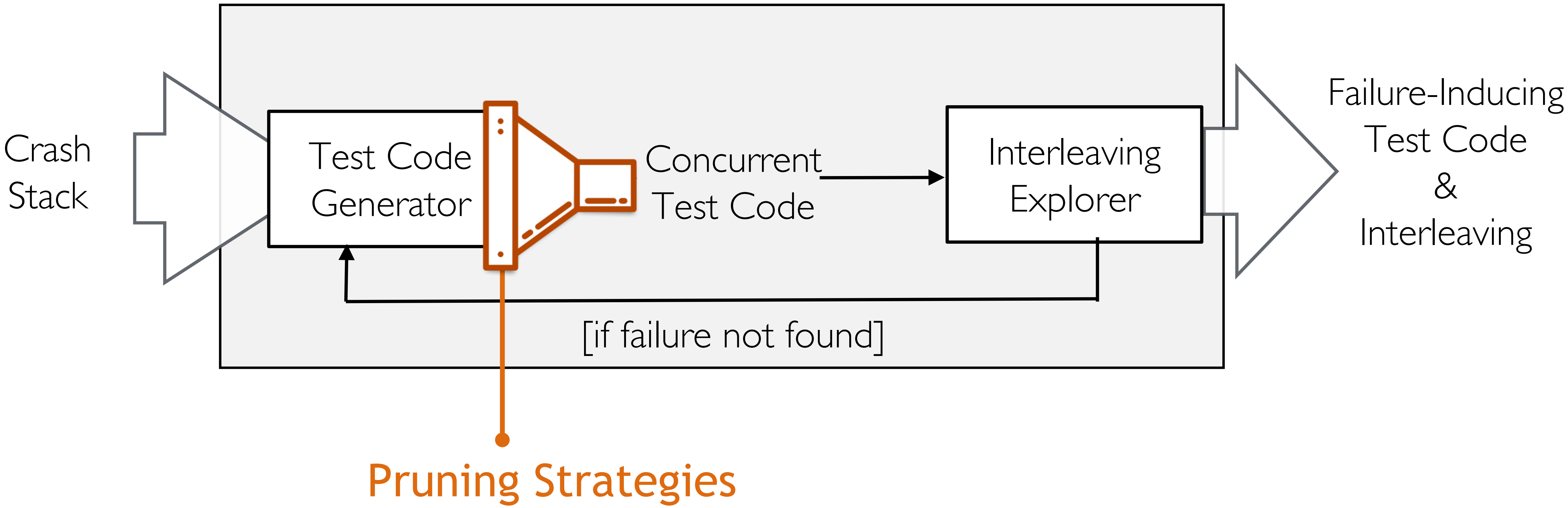
Avoid exploring the interleaving space of **redundant** and **irrelevant** test codes

Test Code Generator



- Build on top of AutoConTest [Terragni and Cheung ICSE '16]
- Systematically explores test codes with fixed pool of input parameters
- It performs state matching to prune redundant test codes.

Pruning Strategies



Pruning Strategies

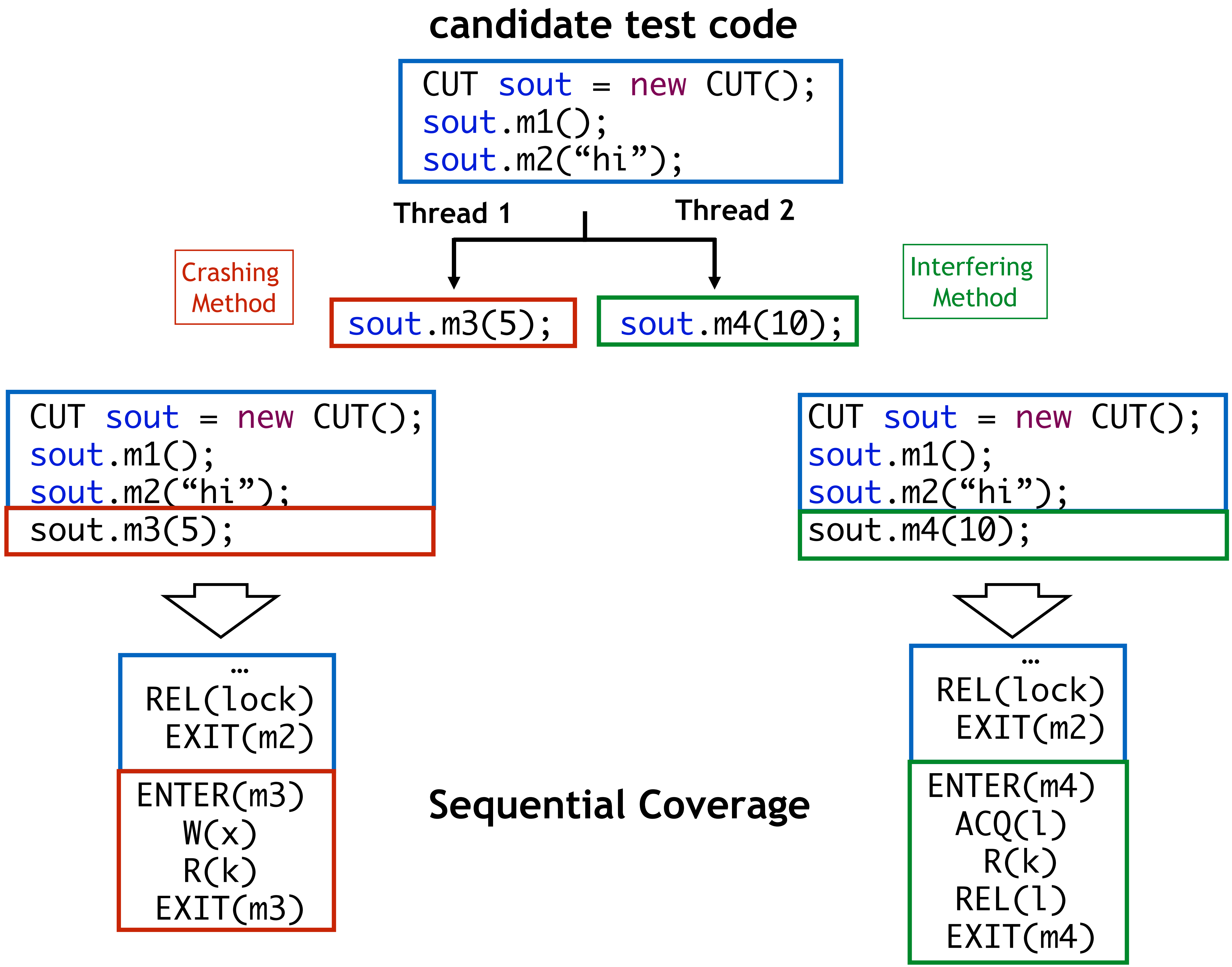
Rely on information obtained by executing the call sequences of a test code **sequentially**

Low computational cost
Good proxy

Sequential Coverage (Terragni and Cheung ICSE '16)

- write W(x) and read R(x) of shared memory x
- lock acquire ACQ(l) and lock release REL(l)
- method enter ENTER(m) and exit EXIT(m)

Pruning Strategies (cont.)



Pruning Strategy : PS-Exception

Prunes a candidate test code if one of its method call sequences throws an exception sequentially

```
CUT sout = new CUT();
sout.m1();
sout.m2("hi");
sout.m9(null);
```

Crashing Method

```
…
REL(lock)
EXIT(m2)
ENTER(m9)
R(x)
```

```
java.lang.NullPointerException
```

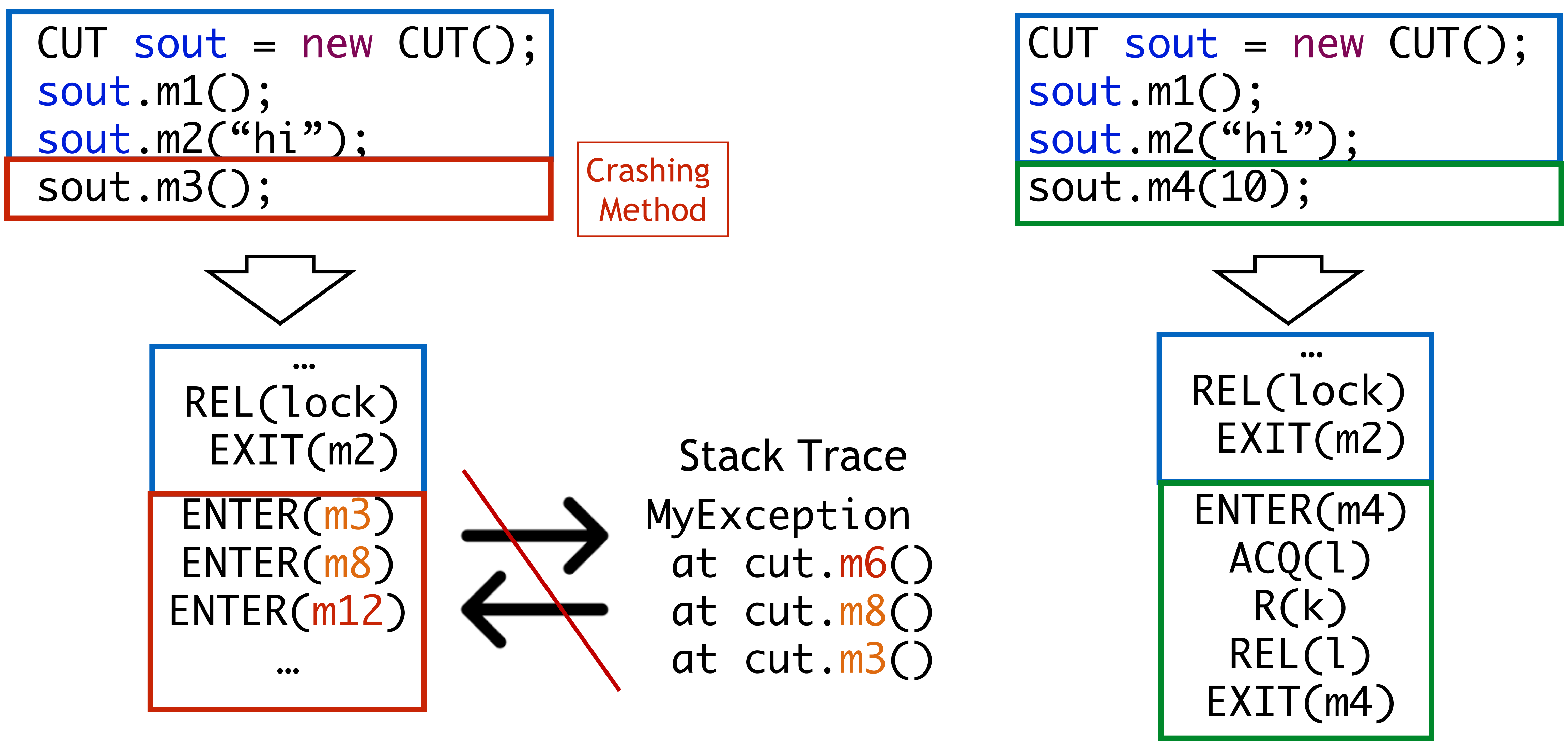
```
CUT sout = new CUT();
sout.m1();
sout.m2("hi");
sout.m4(10);
```

```
…
REL(lock)
EXIT(m2)
ENTER(m4)
ACQ(l)
R(k)
REL(l)
EXIT(m4)
```

Our focus are concurrent (not sequential) failures!

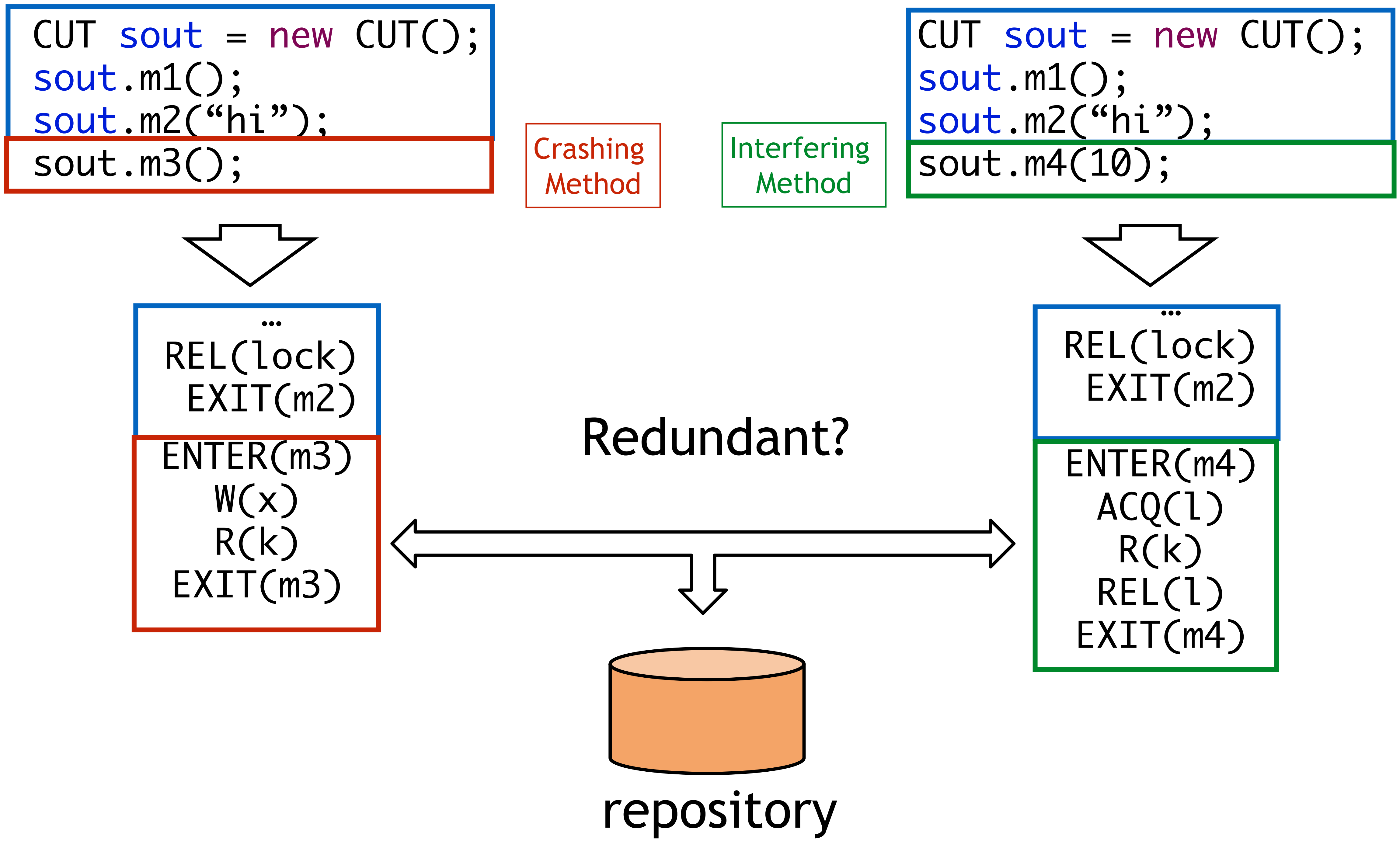
Pruning Strategy : PS-Stack

Prunes a candidate test code if the sequential coverage of the crashing method does not match the crash stack



Pruning Strategy : PS-Redundant

Prunes a candidate test code if the sequential coverages of the concurrent suffixes are redundant



Pruning Strategy : PS-Interfere

Prunes a candidate test code if the concurrent suffixes do not access (at least one write) the same shared memory location

```
CUT sout = new CUT();
sout.m1();
sout.m2(“hi”);
sout.m3();
```

Crashing Method

Interfering Method

```
CUT sout = new CUT();
sout.m1();
sout.m2(“hi”);
sout.m4(10);
```

```
…
REL(lock)
EXIT(m2)
ENTER(m3)
W(x)
EXIT(m3)
```

Shared memory accessed

x y

```
…
REL(lock)
EXIT(m2)
ENTER(m4)
ACQ(l)
R(y)
REL(l)
EXIT(m4)
```

Pruning Strategy : PS-Interleave

Prunes a candidate test code if the concurrent suffixes are mutually exclusive

```
CUT sout = new CUT();
sout.m1();
sout.m2(“hi”);
sout.m1();
```

Crashing Method

```
…
REL(lock)
EXIT(m2)

ENTER(m1)
ACQ(l)
W(x)
REL(l)
EXIT(m1)
```

Interfering Method

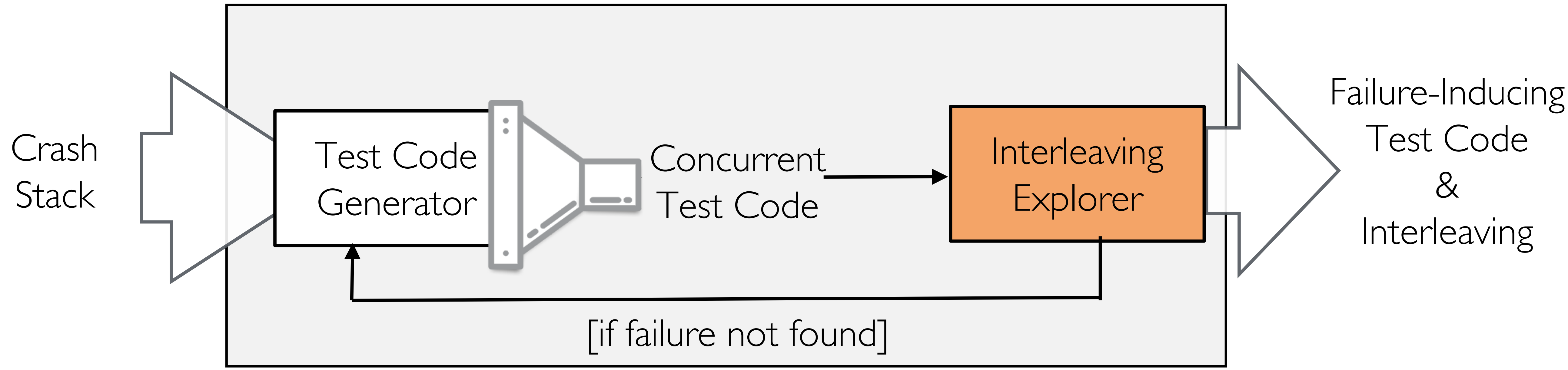
```
CUT sout = new CUT();
sout.m1();
sout.m2(“hi”);
sout.m4(10);
```

```
…
REL(lock)
EXIT(m2)

ENTER(m4)
ACQ(l)
R(x)
REL(l)
EXIT(m4)
```

Cannot interleave!

Interleaving Explorer



- Relies on Cortex [Machado et al. PPoPP'16]
- Uses symbolic execution and constraint solving to identify failure inducing interleavings

Evaluation

RQ1: ConCrash effectiveness

RQ2: Contribution of each Pruning Strategy

RQ3: Comparison with Testing Approaches

Subjects

10 real, known and fixed concurrency faults of thread-safe classes in 5 popular codebases

Class Under Test	Code Base	SLOC	# Methods	Type of Except.	Crash Stack Depth
PerUserPoolDataSource	Commons DBCP	719	68	ConcurrentModif.	4
SharedPoolDataSource		546	44	ConcurrentModif.	4
IntRange	Commons Math	278	44	AssertionError	1
BufferedInputStream	Java JDK	304	12	NullPointerExc.	2
Logger		528	45	NullPointerExc.	4
PushbackReader		143	13	NullPointerExc.	1
NumberAxis	JFreeChart	1,662	119	IllegalArgumentExc.	2
XYSeries		200	28	ConcurrentModif.	4
Category	Log4j	387	43	NullPointerExc.	1
FileAppender		185	13	NullPointerExc.	2

RQ1 : Effectiveness

Average results of 5 runs with a time budget of 5 hours

Class Under Test	Success Rate
PerUserPoolDataSource	100%
SharedPoolDataSource	100%
IntRange	100%
BufferedInputStream	100%
Logger	100%
PushbackReader	100%
NumberAxis	100%
XYSeries	100%
Category	100%
FileAppender	100%
AVG	**100%**

Failure is reproduced in all runs

RQ1 : Effectiveness

Average results of 5 runs with a time budget of 5 hours

Class Under Test	Success Rate	Failure Reprod. Time (sec)
PerUserPoolDataSource	100%	63
SharedPoolDataSource	100%	42
IntRange	100%	13
BufferedInputStream	100%	15
Logger	100%	70
PushbackReader	100%	7
NumberAxis	100%	30
XYSeries	100%	107
Category	100%	25
FileAppender	100%	92
AVG	100%	**46**

Average failure reproduction time is less than 1 minute

RQ1 : Effectiveness

Average results of 5 runs with a time budget of 5 hours

Class Under Test	Success Rate	Failure Reprod. Time (sec)	# Tests Retained after Pruning
PerUserPoolDataSource	100%	63	2
SharedPoolDataSource	100%	42	2
IntRange	100%	13	1
BufferedInputStream	100%	15	2
Logger	100%	70	3
PushbackReader	100%	7	1
NumberAxis	100%	30	1
XYSeries	100%	107	8
Category	100%	25	1
FileAppender	100%	92	5
AVG	100%	46	**3**

Effective test code generation

RQ1 : Effectiveness

Average results of 5 runs with a time budget of 5 hours

Class Under Test	Success Rate	Failure Reprod. Time (sec)	# Tests Retained after Pruning	Test Size (# method calls)
PerUserPoolDataSource	100%	63	2	4
SharedPoolDataSource	100%	42	2	4
IntRange	100%	13	1	4
BufferedInputStream	100%	15	2	5
Logger	100%	70	3	5
PushbackReader	100%	7	1	4
NumberAxis	100%	30	1	3
XYSeries	100%	107	8	6
Category	100%	25	1	5
FileAppender	100%	92	5	10
AVG	**100%**	**46**	**3**	**5**

Small test codes

RQ2 : Pruning Strategies

Failure Reproduction Time (sec)

Class Under Test	NO-Pruning (seconds)
PerUserPoolDataSource	15,456
SharedPoolDataSource	9,240
IntRange	204
BufferedInputStream	77
Logger	6,520
PushbackReader	33
NumberAxis	508
XYSeries	2,758
Category	348
FileAppender	540
AVG	**3,569**

RQ2 : Pruning Strategies

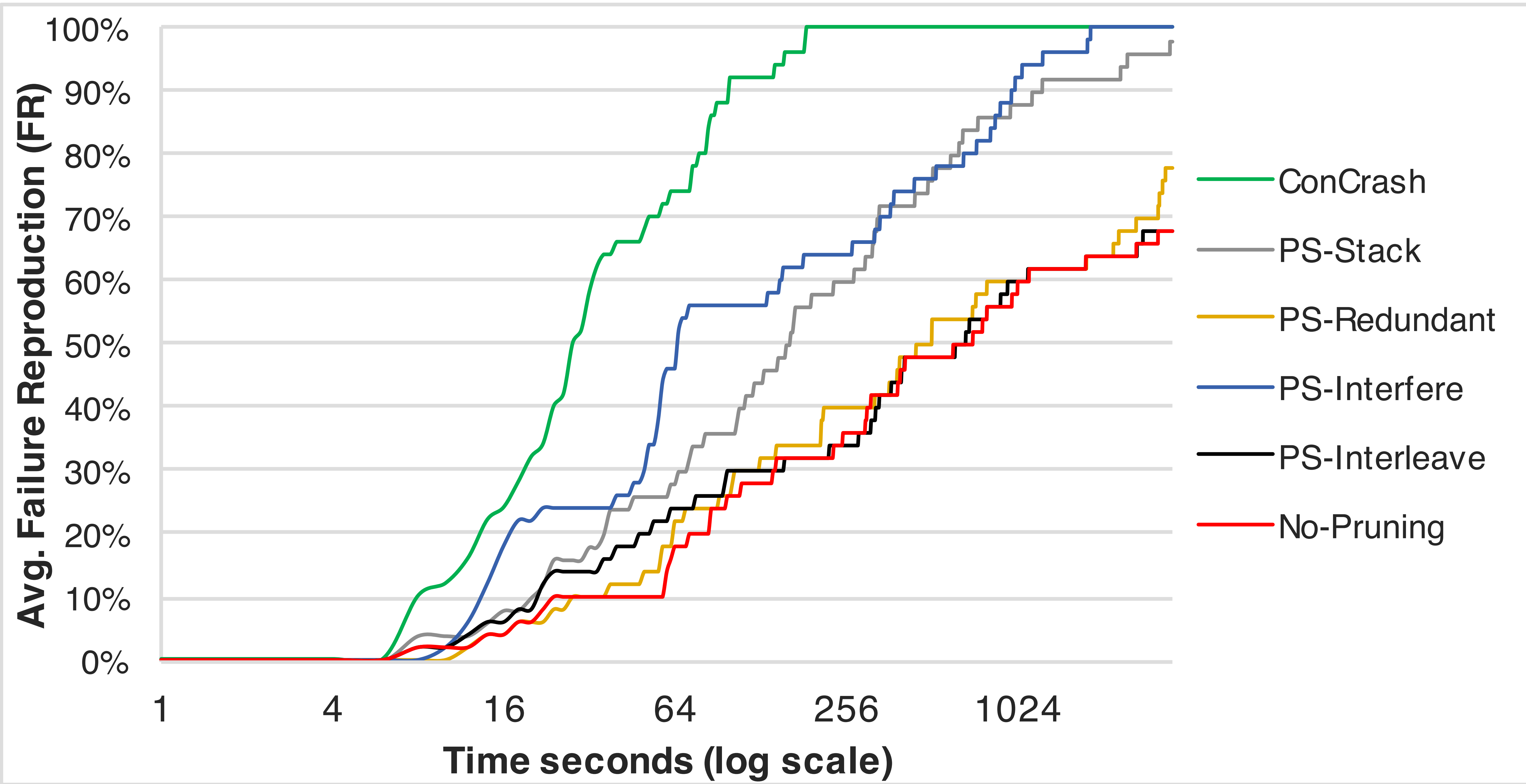
Failure Reproduction Time (sec)

times of improvement with respect to No-Pruning

Class Under Test	NO-Pruning (seconds)	PS-Stack	PS-Redundant	PS-Interfere	PS-Interleave
PerUserPoolDataSource	15,456	29.4x	1.0x	21.2x	1.0x
SharedPoolDataSource	9,240	25.5x	1.3x	23.7x	1.0x
IntRange	204	1.3x	1.5x	12.1x	1.0x
BufferedInputStream	77	1.2x	1.2x	1.8x	3.0x
Logger	6,520	2.5x	2.0x	12.0x	1.9x
PushbackReader	33	1.7x	1.0x	2.9x	1.1x
NumberAxis	508	1.7x	1.1x	9.8x	1.0x
XYSeries	2,758	16.7x	1.0x	2.1x	1.0x
Category	348	1.3x	1.0x	5.8x	1.0x
FileAppender	540	1.1x	1.6x	4.4x	1.0x
AVG	**3,569**	**7.3x**	**1.2x**	**11.0x**	**1.1x**

low (>1.0x and <2.0x). medium (≥ 2.0 and < 10.0) high (≥ 10.0)

RQ2 : Pruning Strategies



RQ3: Comparison with Testing Approaches

ConTeGe [Pradel and Gross PLDI '12] (random-based)
AutoConTest [Terragni and Cheung ICSE '16] (coverage-based)

Class Under Test	ConTeGe Success Rate	ConTeGe Failure Reprod. Time (sec)	AutoConTest Success Rate	AutoConTest Failure Reprod. Time (sec)
PerUserPoolDataSource	0%	>18,000	0%	>18,000
SharedPoolDataSource	0%	>18,000	0%	>18,000
IntRange	0%	>18,000	100%	23
BufferedInputStream	80%	4,487	0%	>18,000
Logger	0%	>18,000	0%	>18,000
PushbackReader	20%	5,796	-	-
NumberAxis	0%	>18,000	100%	93
XYSeries	40%	12,387	0%	>18,000
Category	100%	14,410	-	-
FileAppender	0%	>18,000	-	-

Conclusion

Reproducing Concurrency Failures

Why is it important?
Ease understanding and fixing the related concurrency fault

Difficult problem!

What is needed?
A failure-inducing
test code and **thread interleaving**

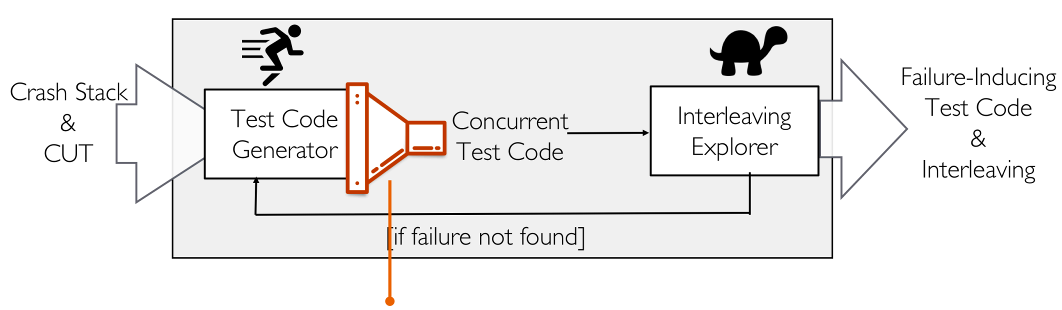
- test code: runnable piece of code that exercises the program under test
- thread interleaving: temporal order of shared memory accesses

State of The Art

Technique	Input	Output: Test code	Output: Interleaving
ODR [Altekar SOSP '09] **LEAP** [Huang FSE '10] **CLAP** [Huang PLDI '13] **CARE** [Jiang ICSE '14] **Cortex** [Machado PPoPP '16] **STRIDE** [Zhuo ICSE '12]	Execution trace	✗	✓
ESD [Zamfir EuroSys '10] Weeratunge ASPLOS '10	Memory core-dumps	✗	✓
ConCrash (our contribution)	**Crash stack**	✓	✓

Less privacy concerns
No overhead issues
Easily obtainable in the field

ConCrash



Pruning Strategies

Avoid exploring the interleaving space of redundant and irrelevant test code

RQ1 : Effectiveness

Average results of 5 runs with a time budget of 5 hours

Class Under Test	Success Rate	Failure Reprod. Time (sec)	# Tests Retained after Pruning	Test Size (# method calls)
PerUserPoolDataSource	100%	63	2	4
SharedPoolDataSource	100%	42	2	4
IntRange	100%	13	1	4
BufferedInputStream	100%	15	2	5
Logger	100%	70	3	5
PushbackReader	100%	7	1	4
NumberAxis	100%	30	1	3
XYSeries	100%	107	8	6
Category	100%	25	1	5
FileAppender	100%	92	5	10
AVG	**100%**	**46**	**3**	**5**

Artifact is available!

ConCrash

http://star.inf.usi.ch/star/software/concrash/

- Tool
- Subjects
- Experimental data