Università della Svizzera italiana

Faculty of Informatics

DISA DIPARTIMENTO DI INFORMATICA SISTEMISTICA E COMUNICAZIONE

Effectiveness and Challenges in Generating Concurrent Tests for Thread-Safe Classes

Valerio Terragni*

* USI Università della Svizzera italiana, Switzerland

Mauro Pezzè*◇

◇ Università di Milano Bicocca, Italy

ASE 2018

5 September, Montpellier, France

Concurrent Programming is Pervasive

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"

```
public class C1 {
 private int x;
 private int y;

 public C1() { … }

 public synchronized void m1() {….}

 public void m2() {
    …
    synchronized(this){…}
    …
 }
```

Achieving Optimal Synchronization is Challenging

Thread-Safe Classes are Buggy

Commons Dbcp / DBCP-369

Exception when using SharedPoolDataSource concurrently

Agile Board

Details

Type: Bug

#278 Axis classes are not Thread

Status: closed-fixed **Owner:**

Priority: 9

Updated: 2003-11-07 **Created**

JDK / JDK-4779253

Race Condition in class java.util.logging.Logger

Agile Board

Details

Type:	Bug	Status:	CLOSED
Priority:	P4	Resolution:	Fixed
Affects Version/s:	1.4.0, 1.4.1, 7	Fix Version/s:	7
Component/s:	core-libs		
Labels:	noreg-trivial webbug		
Subcomponent:	java.util.logging		
Resolved In Build:	b16		
CPU:	generic, x86, sparc		
OS:	generic, solaris_7, windows_2000		
Verification:	Not verified		

ORACLE Ja

Oracle Technology Network

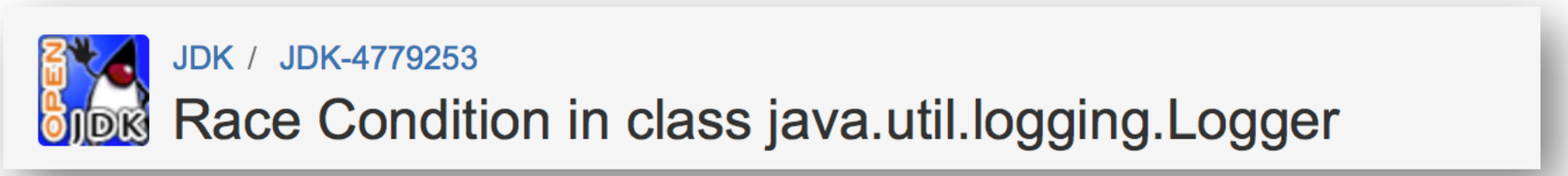
JDK-4093418 : D

CPU: generic,x86

Thread safety bug: ENV clobbered #1709

Closed **balexand** opened this issue on Nov 22, 2014 · 10 comments

Example of a Thread-Safety Violation



Thread 1

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

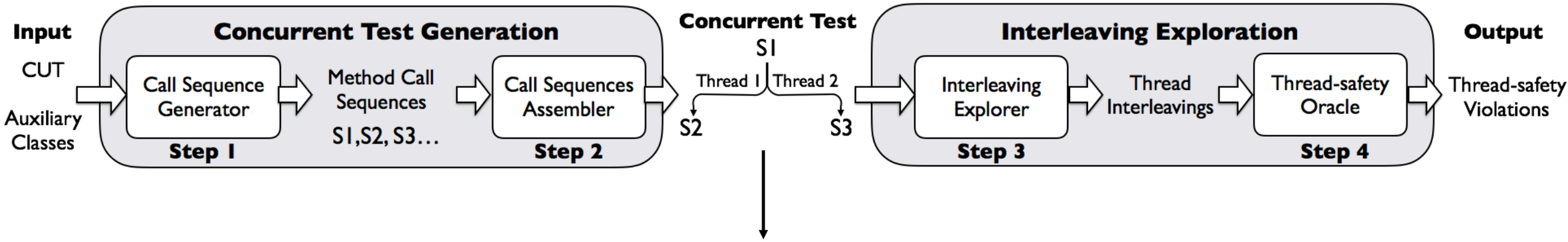
Thread 2

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

```
= null
```

NullPointer exception

Concurrent Test Generation

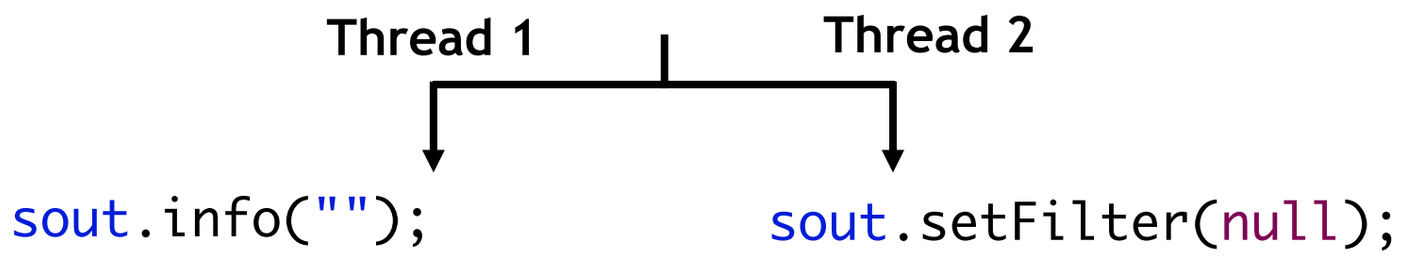


Set of method call sequences that exercise the public interface of a class from multiple threads

JDK / JDK-4779253

Race Condition in class java.util.logging.Logger

```
Logger sout = Logger.getAnonymousLogger();
Filter filter0 = new Filter();
sout.setFilter(filter0);
```



Concurrent Test Generation

State of the Art

ConSuite@ICST

Narada@PLDI
Intruder@FSE

CovCon@ICSE

Ballerina@ICSE
ConTeGe@PLDI

Omen@FSE

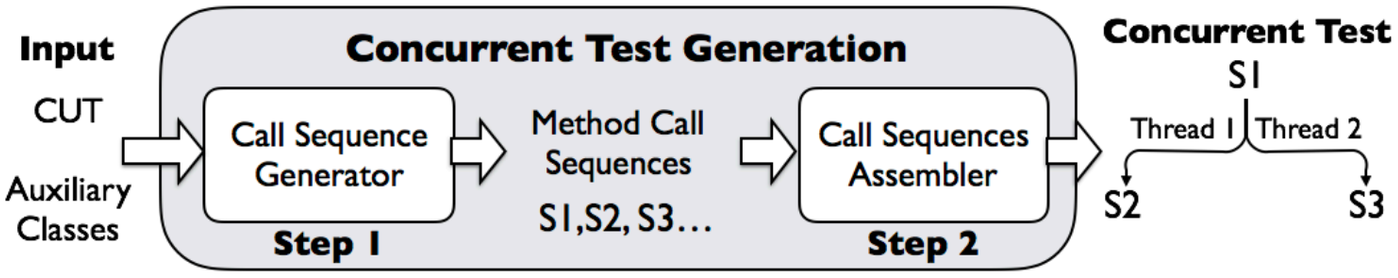
AutoConTest@ICSE
Minion@OOPSLA

Concurrent Test Generation

Contributions (Outline)

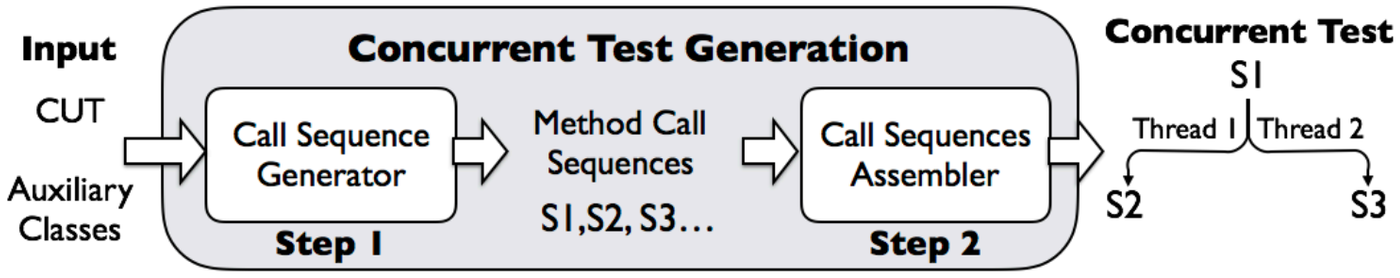
1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area

State of the Art



Tool name	Venue	Year	Category
Ballerina	ICSE	2012	Random-based
ConTeGe	PLDI	2012	
ConSuite	ICST	2013	Coverage-based
AutoConTest	ICSE	2016	
CovCon	ICSE	2017	
Omen	OOPSLA	2014	Sequential-test-based
Narada	PLDI	2015	
Intruder	FSE	2015	
Minion	OOPSLA	2016	

Random-Based

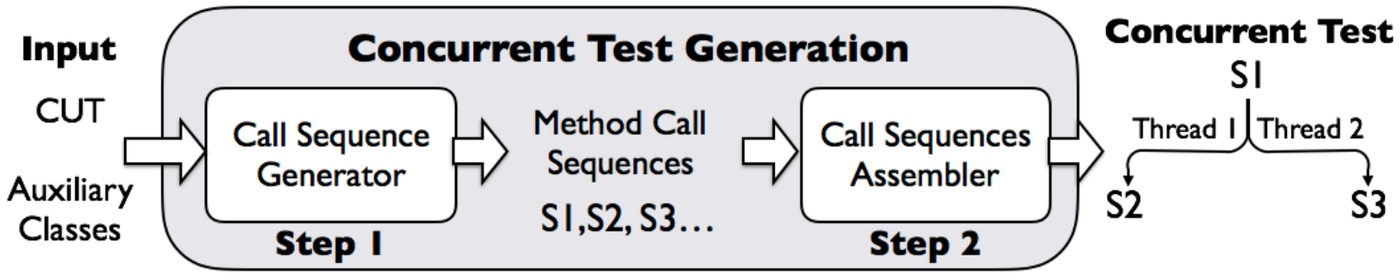


Tool name	Venue	Year	Category
Ballerina	ICSE	2012	**Random-based**
ConTeGe	PLDI	2012	

Low computational analysis

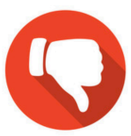
Many randomly generated tests are needed
Many redundant tests are generated

Coverage-Based



Tool name	Venue	Year	Category
ConSuite	ICST	2013	
AutoConTest	ICSE	2016	**Coverage-based**
CovCon	ICSE	2017	

Limited number of redundant tests

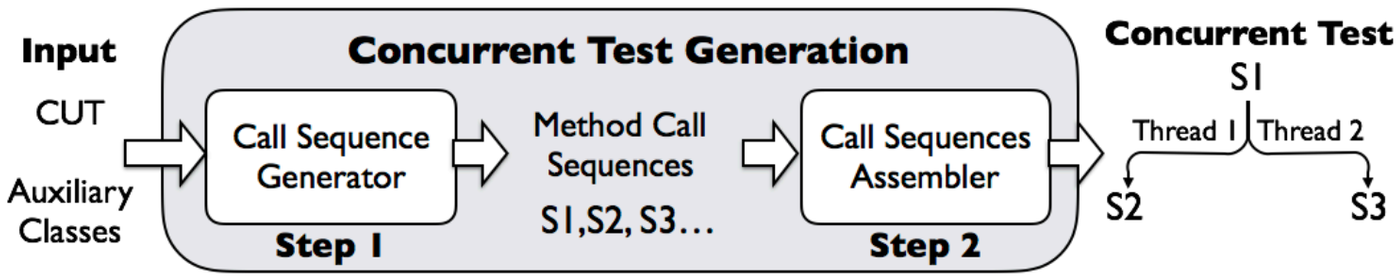


Difficult trade-off between:

A precise computation of coverage targets

Low analysis computational cost

Sequential-Test Based

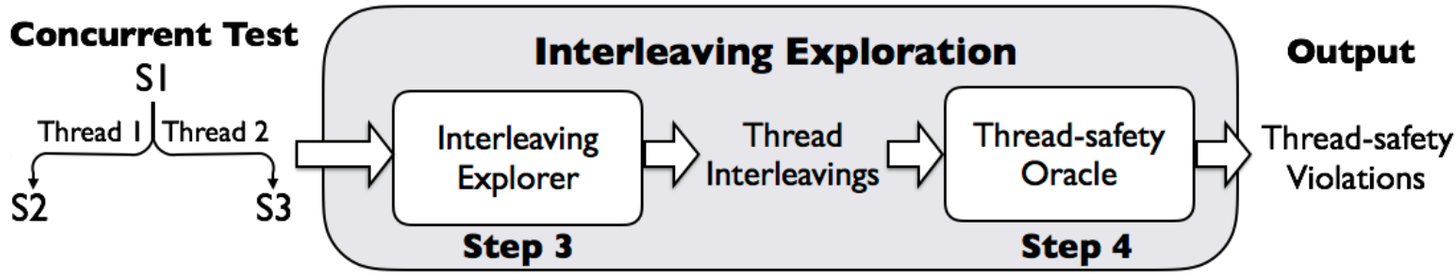


Tool name	Venue	Year	Category
Omen	OOPSLA	2014	**Sequential-test-based**
Narada	PLDI	2015	
Intruder	FSE	2015	
Minion	OOPSLA	2016	

Generate only those tests that reveal the considered type of bug

Require a seeded sequential test suites in input

Interleaving Exploration & Thread-Safety Oracle



			Interleaving Explorer			Thread-Safety Oracle	
Tool name	**Venue**	**Year**	**Random**	**Selective**	**Exhaustive**	**Implicit**	**Internal**
Ballerina	ICSE	2012			✓	✓	
ConTeGe	PLDI	2012	✓			✓	
ConSuite	ICST	2013		✓			✓
AutoConTest	ICSE	2016		✓			✓
CovCon	ICSE	2017	✓			✓	
Omen	OOPSLA	2014		✓		✓	
Narada	PLDI	2015	✓				✓
Intruder	FSE	2015		✓			✓
Minion	OOPSLA	2016		✓		✓	

Contributions (Outline)

1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area

Subjects

JaConTeBe: A Benchmark Suite of Real-World Java Concurrency Bugs

ASE 2015

Ziyi Lin*, Darko Marinov†, Hao Zhong‡, Yuting Chen‡, and Jianjun Zhao‡

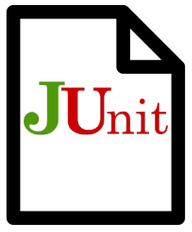
Code base (label)	# of subjects (bugs)	Description
Apache DBCP (dbcp)	4	Database connection pool
Apache Derby (derby)	5	Relational database
Apache Groovy (groovy)	6	Dynamic language for JVM
OpenJDK (jdk)	20	Java Development Kit
Apache Log4J (log4j)	5	Logging library
Apache Lucene (lucene)	2	Search library
Apache Pool (pool)	5	Object-pooling API
Total	**47**	

Evaluation Setup

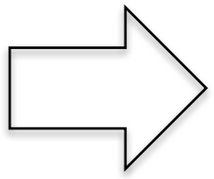
JaConTeBe 47 subjects

Buggy code base

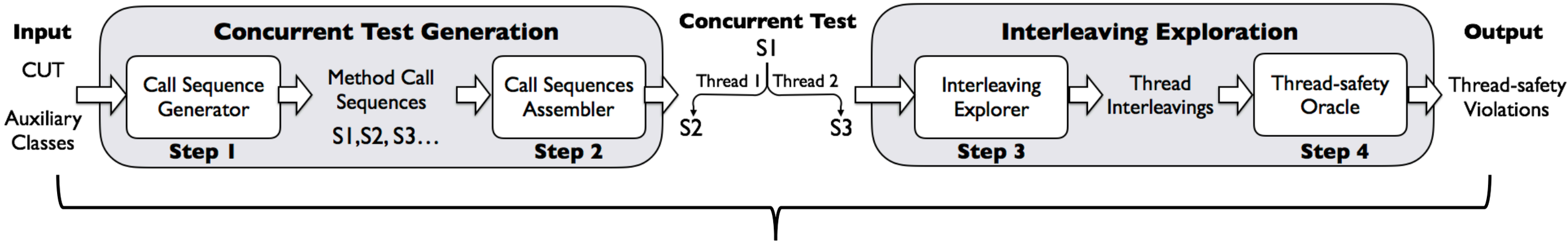
Bug report



Manually-written test



Class Under Test (CUT)
Auxiliary classes



Time budget **one hour** per subject

10 runs per subject

ConTeGe	ConTeGeJPF	AutoConTest	CovCon	CovConJPF	Omen	Narada	Intruder

Fault Type	Failure Type	ConTeGe	ConTeGeJPF	AutoConTest	CovCon	CovConJPF	Omen	Narada	Intruder
inconsistent synchronization	endless loop								
	logic								
race atomicity violations	endless loop								
	exception								
	logic								
resource deadlock	endless hang								
wait-notify deadlock	endless hang								

Fault Type	Failure Type	ConTeGe	ConTeGeJPF	AutoConTest	CovCon	CovConJPF	Omen	Narada	Intruder
inconsistent synchronization	endless loop								
	logic								
race atomicity violations	endless loop								
	exception								
	logic								
resource deadlock	endless hang								
wait-notify deadlock	endless hang								

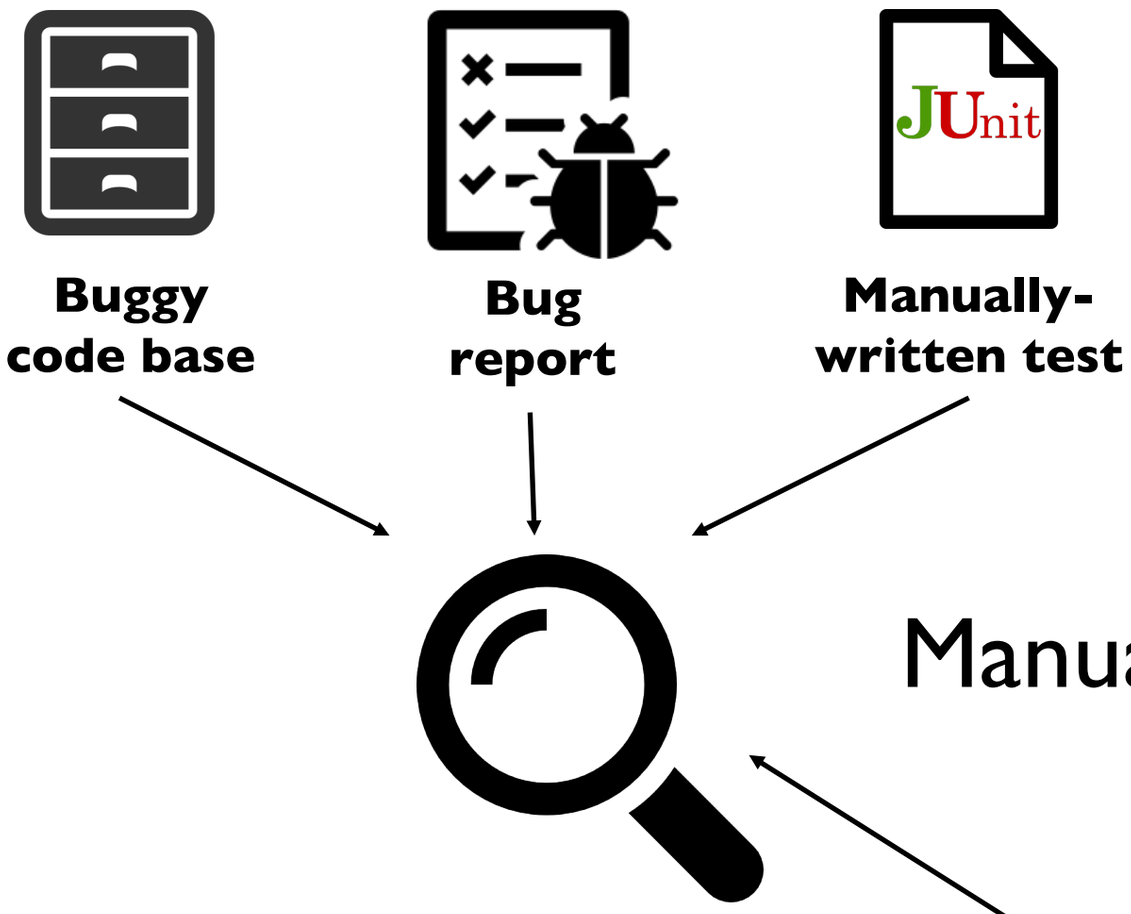
- Automated concurrent test generators find (8 out of 47) 17% of the faults
- None of them alone finds more than (6 out of 47) 13% of the faults

Contributions (Outline)

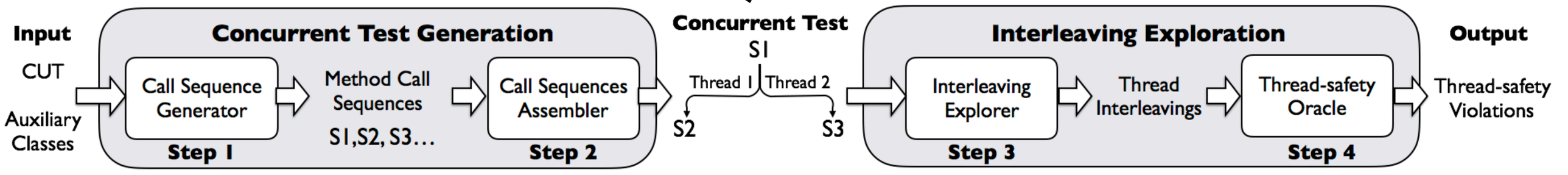
1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area

Analysis of the Tools Limitations

JaConTeBe 47 subjects



Manual Investigation



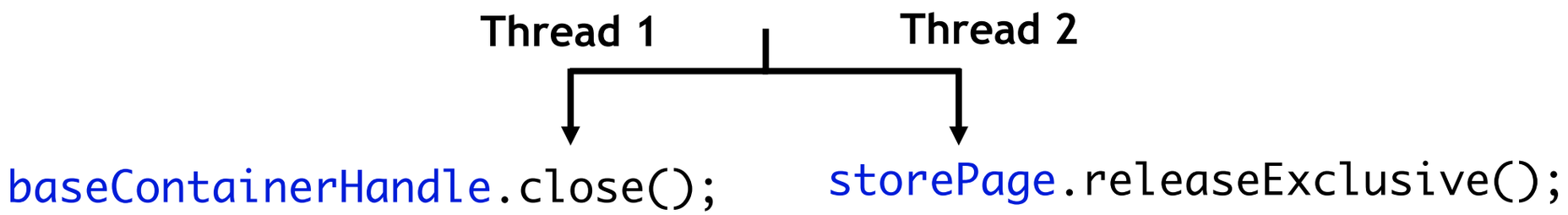
Common Issue 1: Invalid Assumptions

40% faults violate at least one of the following assumptions

- Two threads only
- One shared object under test
- No static invocations

manually-written test : derby5

```
…
storePage.setExclusive(baseContainerHandle);
baseContainerHandle.addObserver(storePage);
```



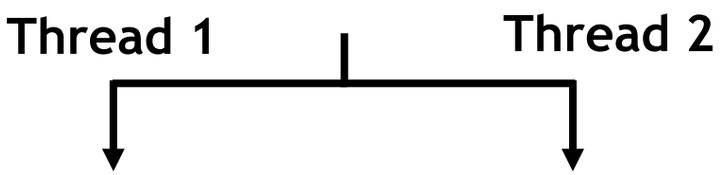
Assumption violated : 2 shared objects under test

Common Issue 2: Environmental Dependencies

25% of the faults require environmental dependencies (DB, files …)

manually-written test : jdk6_2

```
String dirA = projectBase  + “/base/a”;
String dirB = projectBase  + “/base/b”;
```

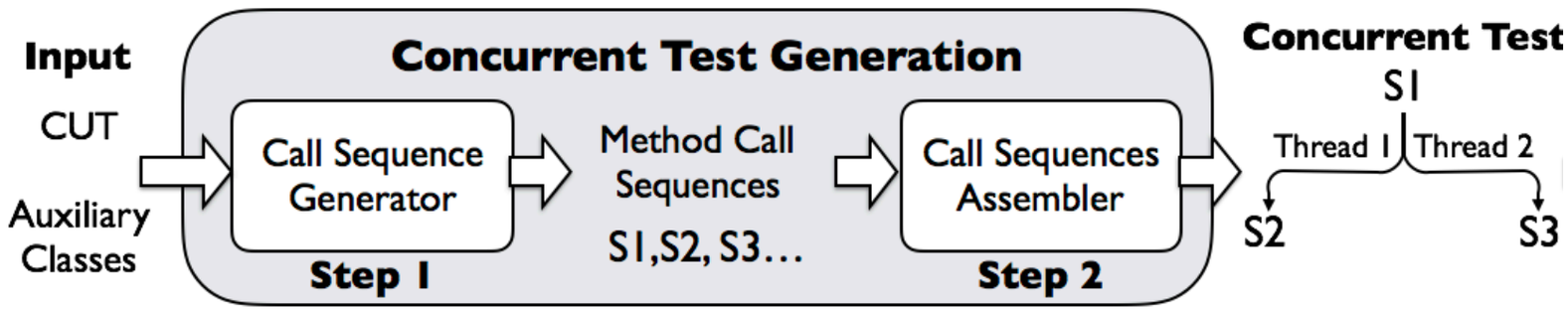



```
File fileA = new File(dirA);
fileA.mkdirs();
```

```
File fileB = new File(dirB);
fileB.mkdirs();
```

Common Issue 3: Inadequacy for Wait-Notify

19 % of the faults require the execution of wait()-notify()



(Step 1) Feedback-directed approach

Sequential (single-thread) execution of call sequences

Discards sequences that throw exceptions or **never terminates (time-out)**

```
ClassA sout = new ClassA();
sout.m1();
sout.m2();
sout.m3();
```

```
public void m3() {
…
lock.wait();
…
}
```

Contributions (Outline)

1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area

Adaptive Configuration

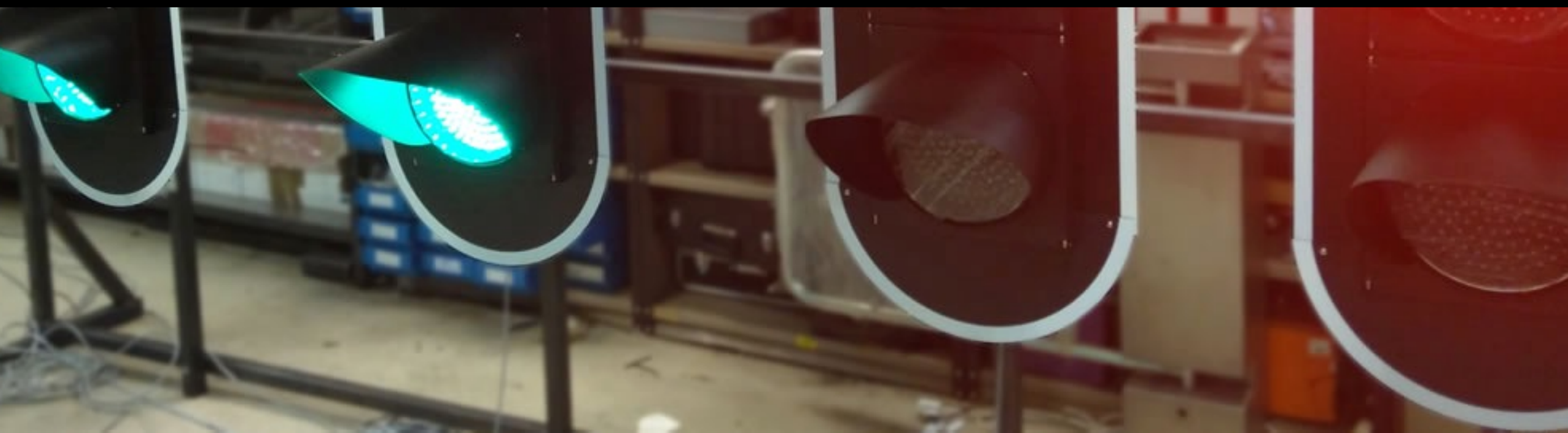
Automatically identify a proper configuration
for a given class under test

Search Space Reduction

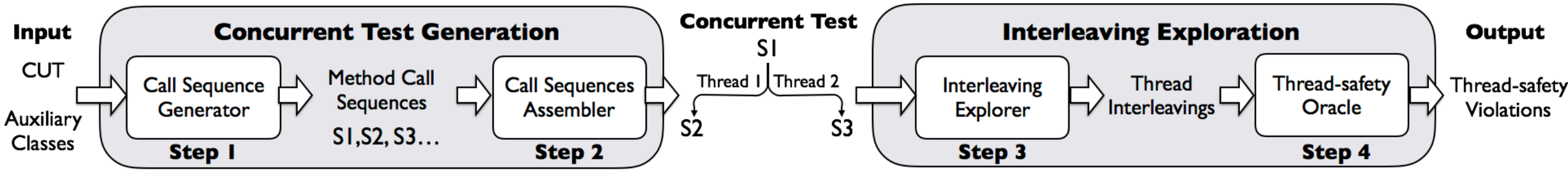
Reduce the search space by identifying methods whose concurrent interactions cannot lead to concurrency failures

Handling Wait and Notify

Revise the 4 steps framework to handle
wait/notify synchronization primitives



Handling Wait and Notify

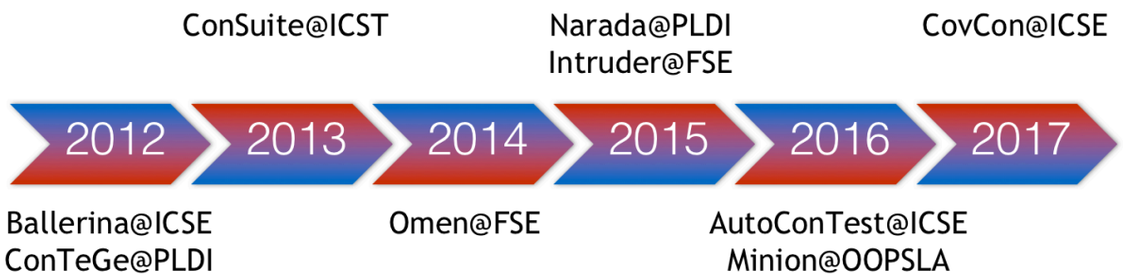


Revise the 4 steps framework to handle wait/notify synchronization primitives

Conclusion

Concurrent Test Generation

State of the Art



Fault Type	Failure Type	ConTeGe	ConTeGeJPF	AutoConTest	CovCon	CovConJPF	Omen	Narada	Intruder
inconsistent synchronization	endless loop								
	logic								
race atomicity violations	endless loop								
	exception								
	logic								
resource deadlock	endless hang								
wait-notify deadlock	endless hang								

- Automated concurrent test generators find (8 out of 47) 17% of the faults
- None of them alone finds more than (6 out of 47) 13% of the faults

Subjects

JaConTeBe: A Benchmark Suite of Real-World Java Concurrency Bugs

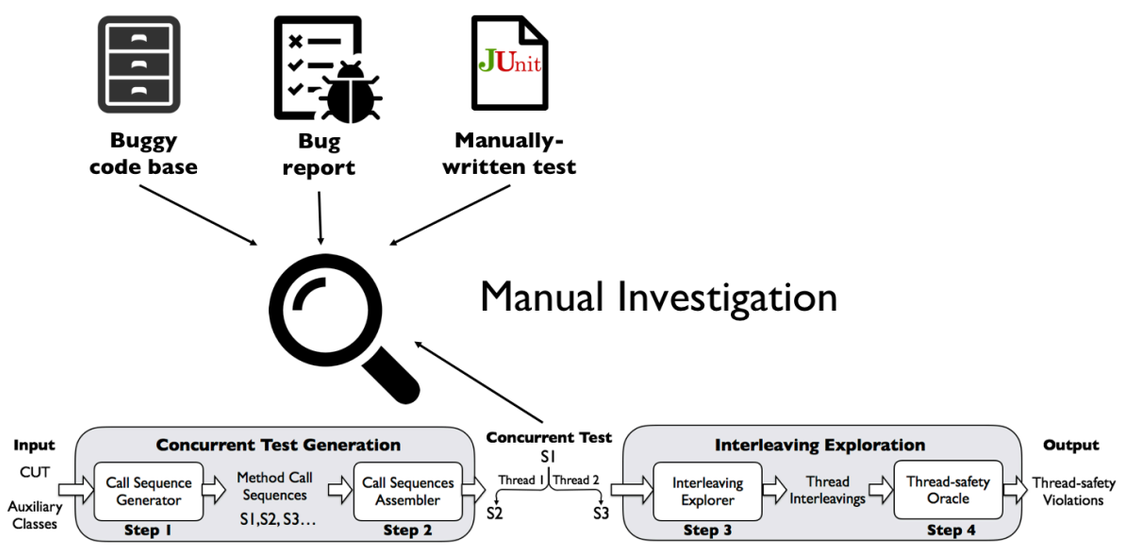
ASE 2015

Ziyi Lin*, Darko Marinov†, Hao Zhong‡, Yuting Chen‡, and Jianjun Zhao‡

Code base (label)	# of subjects (bugs)	Description
Apache DBCP (dbcp)	4	Database connection pool
Apache Derby (derby)	5	Relational database
Apache Groovy (groovy)	6	Dynamic language for JVM
OpenJDK (jdk)	20	Java Development Kit
Apache Log4J (log4j)	5	Logging library
Apache Lucene (lucene)	2	Search library
Apache Pool (pool)	5	Object-pooling API
Total	**47**	

Analysis of the Tools Limitations

JaConTeBe 47 subjects



Artifact is available!

http://star.inf.usi.ch/star/software/contest2018

- Runnable scripts
- Experimental data

Contributions (Outline)

1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area