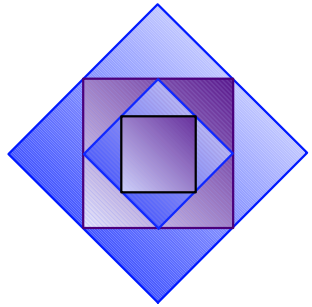
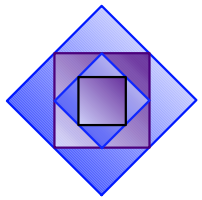
Testing Multi-threaded Java Programs with ConTest-Lite

Eitan Farchi
Shmuel Ur
Evgeny Goldin
Yarden Nir
Orit Edelstein

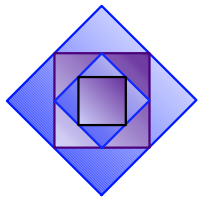
IBM Research Lab in Haifa





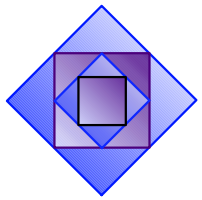
Outline

- **Why is concurrency a testing issue?**
- **Using ConTest-Lite to find concurrent bugs is simple**
- **Concurrent bugs and why they are not found**
- **Overview of ConTestLite**
  - Finding concurrent bugs
  - Debugging and coverage support
- **Methodology**
  - What are good concurrent tests?
  - Re-running tests
- **Using ConTest-Lite (continued)**



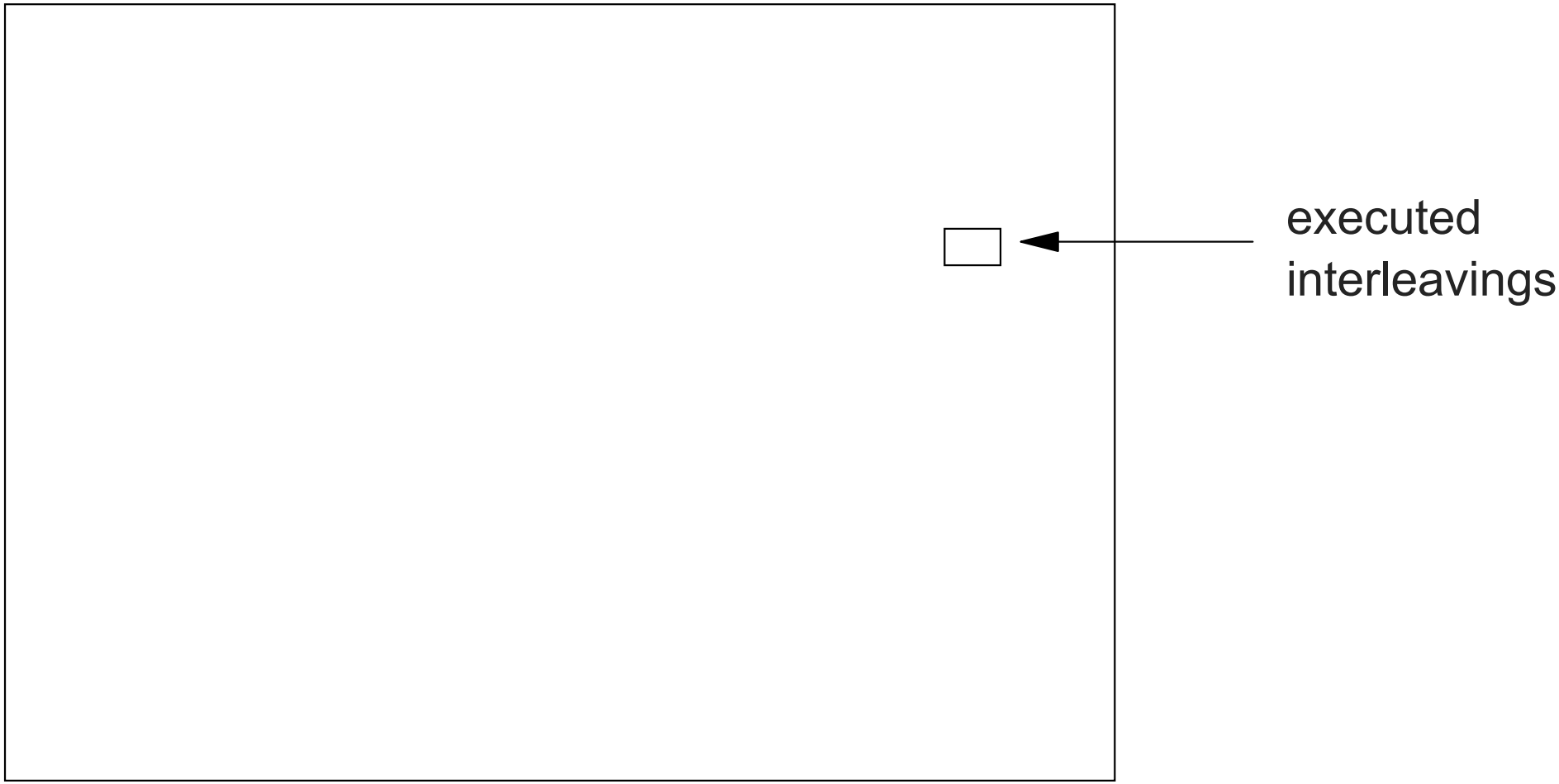
Why is Concurrent Testing Hard?

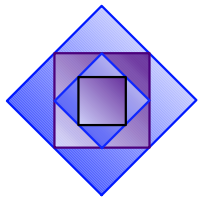
- **Concurrency introduces non-determinism**
  - Multiple executions of the same test may have different interleaving (and different results!)
    - an *interleaving* is the relative execution order of the program threads
- **Re-executing a test on a single stand-alone processor is not useful**
- **Debugging affects the timing**
- **No useful coverage measures for the interleaving space**
- **Result: Most bugs are found in system tests, stress tests, or by the customer**



Re-executing on a Stand-alone Processor Is Not Useful

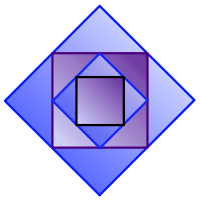
An exponential space of possible interleavings:





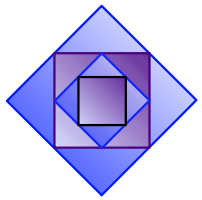
Real Life Motivation

- **Project: file system (IFS for the AS/400)**
- **Stage: component test**
- **Observed behavior: "use-count" values corrupted during stress test**
- **Time wasted:** several calendar months, more than 0.5PY
- **Main difficulty:** failure was not repeatable
- **Fault description:** incorrect use of compare-and-swap result
- **Bug should have been found during Unit Test!**

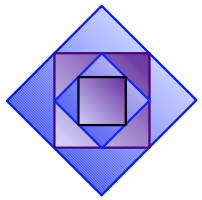


Using conTestLite to Find Concurrent Bugs is simple

- **Instrument the program under test**
- **Add conTest-Lite to the classpath used when executing the program under test**
- **Run your tests**
  - Example: Service() method is not synchronized

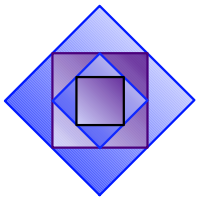


With ConTest, each test will go a long way



Outline

- **Why is concurrency a testing issue?**
- **Using conTest-Lite to find concurrent bugs is simple**
- **Concurrent bugs and why they are not found**
- **Overview of ConTest-Lite**
  - Finding concurrent bugs
  - Debugging and coverage support
- **Methodology**
  - What are good concurrent tests?
  - Re-running tests
- **Using contest-Lite (continued)**



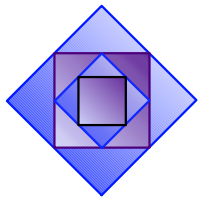
Example of an Interleaving-sensitive Test Result

- Threads A and B execute (i=0; add(&i))
- The result depends upon the interleaving
- Would not be revealed in a typical test

Program

```
add(int *x){
1     int tmp =
*x;
2     tmp++;
3    *x = tmp;
}
```

Interleaving		
	A1	A1
	A2	B1
	A3	B2
	B1	A2
	B2	B3
	B3	A3
Result:	2	1



How much is 1+10+100+1000+10000?

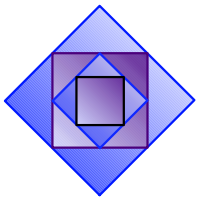
```
static final int NUM = 5;
static int Global = 0;

public static void main(String[] argv){
    Global = 0;
    threads[0] = new Adder(1);
    threads[1] = new Adder(10);
    threads[2] = new Adder(100);
    threads[3] = new Adder(1000);
    threads[4] = new Adder(10000);

    // Start Threads
    for(int i=0;i<NUM;i++){ threads[i].start(); }

    try{ Thread.sleep(1000); } catch(Exception exc){}

    // Print Results
    System.out.println(Global);
}
```



class Adder

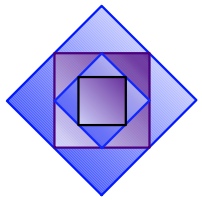
```
class Adder extends Thread

 int Local;

 Adder(int i){
        Local = i;
 }

 public void add(){
        example.Global += Local;
 }

 public void run(){
        add();
 }
```



Coverage Results Without ConTest

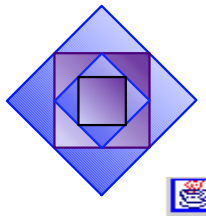
Here are the legal tasks :

Number of Tasks: 32 Covered: 1 Coverage Percentage: 3.125

Click on a column header to sort the rows by this column

ThreadA	ThreadB	ThreadC	ThreadD	ThreadE	TOTAL COVERA...	UNIQUE COVE...	OUT OF	percentage
0	1	0	0	1	0	0	1	0.0
0	1	0	0	0	0	0	1	0.0
0	0	1	0	1	0	0	1	0.0
1	0	0	1	1	0	0	1	0.0
0	0	1	0	0	0	0	1	0.0
1	0	0	1	0	0	0	1	0.0
1	1	1	1	1	1000	1	1	100.0
1	1	1	1	0	0	0	1	0.0
0	1	1	0	1	0	0	1	0.0
0	1	1	0	0	0	0	1	0.0
1	0	1	1	1	0	0	1	0.0
1	0	1	1	0	0	0	1	0.0
0	0	0	1	1	0	0	1	0.0
0	0	0	1	0	0	0	1	0.0
1	1	0	0	1	0	0	1	0.0
1	1	0	0	0	0	0	1	0.0
0	1	0	1	1	0	0	1	0.0
0	1	0	1	0	0	0	1	0.0
0	0	1	1	1	0	0	1	0.0
0	0	1	1	0	0	0	1	0.0
1	0	0	0	1	0	0	1	0.0
1	0	0	0	0	0	0	1	0.0
1	1	1	0	1	0	0	1	0.0
1	1	1	0	0	0	0	1	0.0
0	1	1	1	1	0	0	1	0.0
0	1	1	1	0	0	0	1	0.0
1	0	1	0	1	0	0	1	0.0
1	0	1	0	0	0	0	1	0.0

Print Export Close



Coverage Results with ConTest

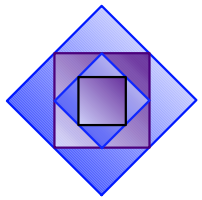
Here are the legal tasks :

Number of Tasks: 32 Covered: 32 Coverage Percentage: 100.0

Click on a column header to sort the rows by this colu...

ThreadA	ThreadB	ThreadC	ThreadD	ThreadE	TOTAL COVERAGE	UNIQUE COVERAGE	OUT OF	percentage
0	0	0	0	0	2	1	1	100.0
1	0	0	0	0	20	1	1	100.0
0	1	0	0	0	18	1	1	100.0
1	1	0	0	0	36	1	1	100.0
0	0	1	0	0	5	1	1	100.0
1	0	1	0	0	47	1	1	100.0
0	1	1	0	0	15	1	1	100.0
1	1	1	0	0	68	1	1	100.0
0	0	0	1	0	3	1	1	100.0
1	0	0	1	0	34	1	1	100.0
0	1	0	1	0	19	1	1	100.0
1	1	0	1	0	62	1	1	100.0
0	0	1	1	0	5	1	1	100.0
1	0	1	1	0	66	1	1	100.0
0	1	1	1	0	19	1	1	100.0
1	1	1	1	0	100	1	1	100.0
0	0	0	0	1	2	1	1	100.0
1	0	0	0	1	17	1	1	100.0
0	1	0	0	1	13	1	1	100.0
1	1	0	0	1	32	1	1	100.0
0	0	1	0	1	4	1	1	100.0
1	0	1	0	1	57	1	1	100.0
0	1	1	0	1	11	1	1	100.0
1	1	1	0	1	45	1	1	100.0
0	0	0	1	1	3	1	1	100.0
1	0	0	1	1	21	1	1	100.0
0	1	0	1	1	17	1	1	100.0
1	1	0	1	1	60	1	1	100.0
0	0	1	1	1	10	1	1	100.0
1	0	1	1	1	67	1	1	100.0
0	1	1	1	1	15	1	1	100.0
1	1	1	1	1	107	1	1	100.0

Print Export Close



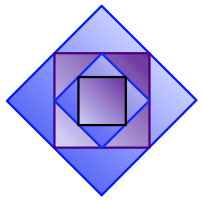
method Add

Java Source code

```
public void add(){
        example.Global += Local;
}
```

Byte Code

```
Method void add()
 0 getstatic #3 <Field int Global>
 3 aload_0
 4 getfield #2 <Field int Local>
 7 iadd
 8 putstatic #3 <Field int Global>
 11 return
```

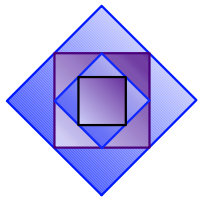


A Bug We Published...

A race condition is a possible source for a defect, since the value of the variable at the time of reading depends on the scheduling. However, not all race conditions are defects. For example, the following code swaps two integers. There is a race condition, but no defect, as the swapping occurs regardless of the interleaving.

```
class Change{
   static int x = 4, y = 5;
   //Used to implement a busy wait.
   static int z1 = -1, z2 = -1;
   //Swap the value of x and y concurrently
   public static void main(String args[ ]){
     (new Thread(new ChangeA( ))).start( );
     (new Thread(new ChangeB( ))).start( );}
}
class ChangeA implements Runnable{
 public void run( ){
   Change.z1 = Change.x;
   while(Change.z2 == -1)
     System.out.println("A is waiting");
   Change.x = Change.z2;}}
class ChangeB implements Runnable{
 public void run( ){
   Change.z2 = Change.y;
   while(Change.z1 == -1)
     System.out.println("B is waiting");
   Change.y = Change.z1;}}
```

It should be noted that race conditions are execution-dependent: a program might be in a race condition in one execution and not in another. Therefore, tools that detect races at run time (or by analyzing the trace of a given run) are likely to miss some potential data races.



Creating a Critical Section is Tricky

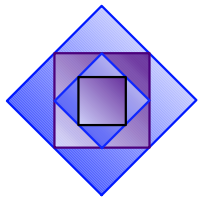
```
class ObjectOne implements Runnable {
  public void run() {
     if(RaceField.race == false)
     {
        RaceField.race = true;
        System.out.println("Mine");
     }
   }
}
class RaceField {
   public static boolean race = false;
}
```

Expected and observed Behavior:

One "Mine" is printed by the thread that won the race

Possible bug:

More than one "Mine" is printed



Atomicity is Never Assured

```
static void transfer(Transfer t) {

        balances[t.fundFrom] -= t.amount;
        balances[t.fundTo] += t.amount;
}
```

Expected Behavior:

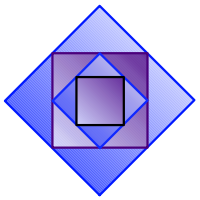
- Money should pass from one account to another

Observed Behavior:

- Sometimes the amount taken is not equal to the amount received

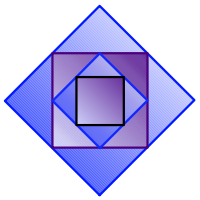
Possible bug:

- Thread switch in the middle of money transfers



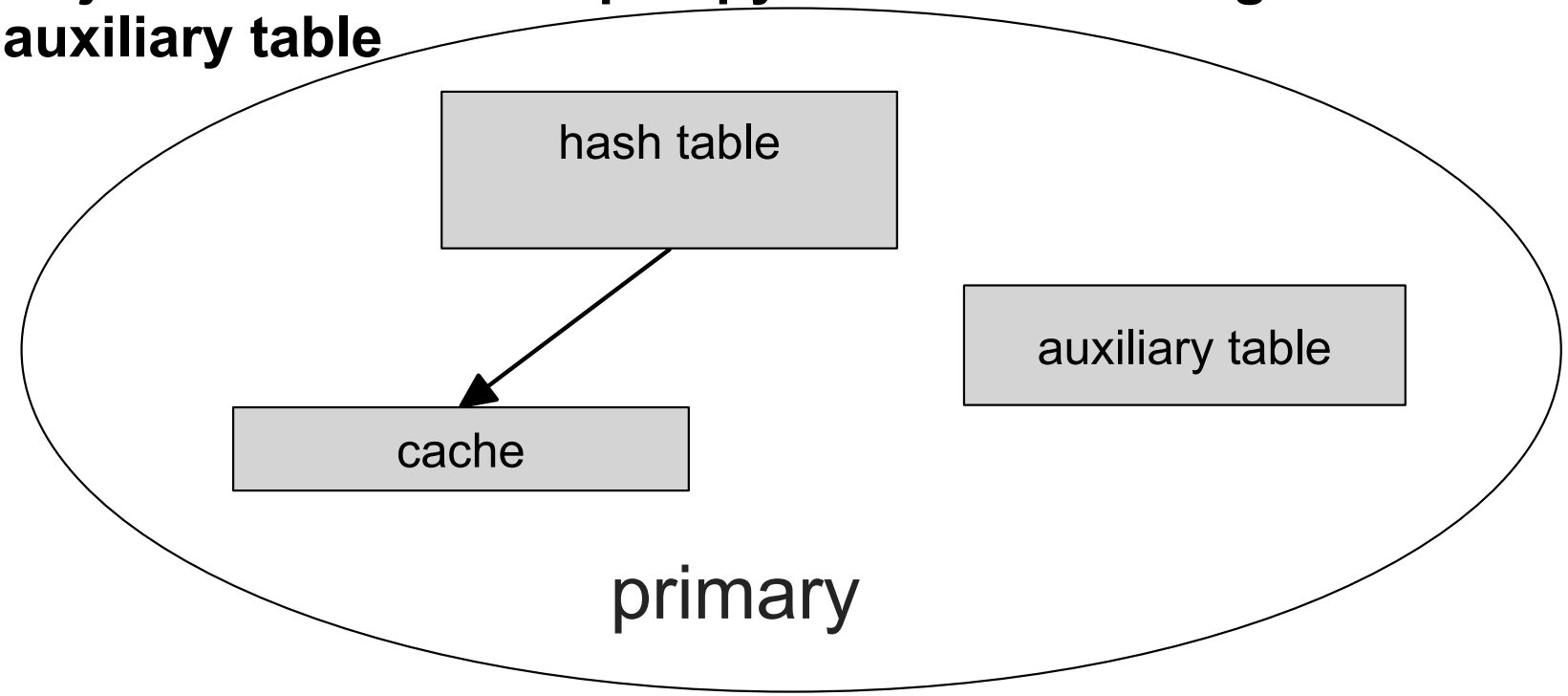
Bug found by ConTest in Websphere Site Analyzer

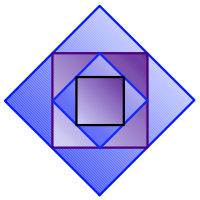
- **Crawler, a ~1000 line Java component, is part of the Websphere Site Analyzer used to perform content analysis of web sites**
- **Bug description:**
  - if (connection != null) connection.setStopFlag();
    1. connection is checked to be !null
    2. CPU is lost
    3. connection is set to null before CPU is regained
  - If this happens before connection.setStopFlag(); is executed, an exception is taken
- **This bug was found while we were still testing ConTest**
- **This bug should (also) be found in unit testing...**



A Servlet Server Bug (HA. Tomcat)

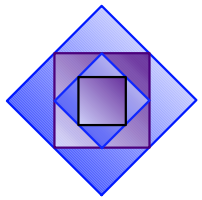
- A high availability servlet server has a hash table that is kept synchronized using a cluster
- The high availability servlet has a primary and a backup node
- The primary node caches the hash table
- Objects are trees: deep copy is obtained using an auxiliary table





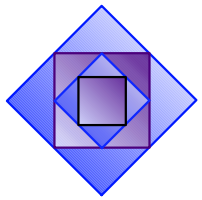
A Servlet Server Bug

- **Bug description**
  - thread1:
    - A node becomes primary
    - Start deep copying of hash table to CACHE table using the auxiliary table
  - thread2:
    - A cache miss on object A occurs
    - Start deep copying of object A from hash table to CACHE table using auxiliary table
  - thread1:
    - Complete deep copying of hash table to the CACHE table. Auxiliary table is discarded
  - thread2:
    - An attempt is made to access the auxiliary table



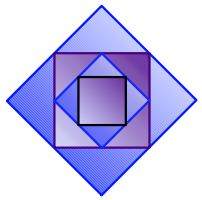
Why are these bugs not found?

- **Frame of mind when the program is written**
- **Requires thread switching at precise locations**
- **Typical testing environment**
  - Thread switch occurs at repeating locations
  - Execution is almost deterministic
  - No load/stress
- **Not enough of the right kind of tests**
- **Not enough tests**



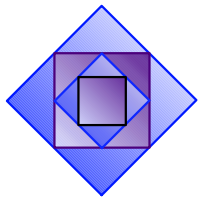
Outline

- **Why is concurrency a testing issue?**
- **Using conTestLite to find concurrent bugs is simple**
- **Concurrent bugs and why they are not found**
- **Overview of ConTest-Lite**
  - Finding concurrent bugs
  - Debugging and coverage support
- **Methodology**
  - What are good concurrent tests?
  - Re-running tests
- **Using contestLite (continued)**



How Does ConTest-Lite Find Bugs?

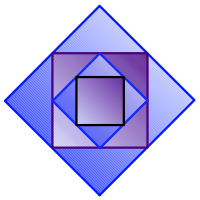
- **We instrument every concurrent event**
  - Concurrent events are the events whose order determines the result of the program
- **At every concurrent event, a random based decision is made whether to cause a context switch**
  - For example, using a sleep statement
- **Philosophy:**
  - Modify the program in such a way that it will be more likely to **exhibit** bugs (without introducing new bugs)
  - Minimize impact on the testing process
  - Re-use existing tests
  - Utilize the time computers are not being used (nights, weekends, etc.)



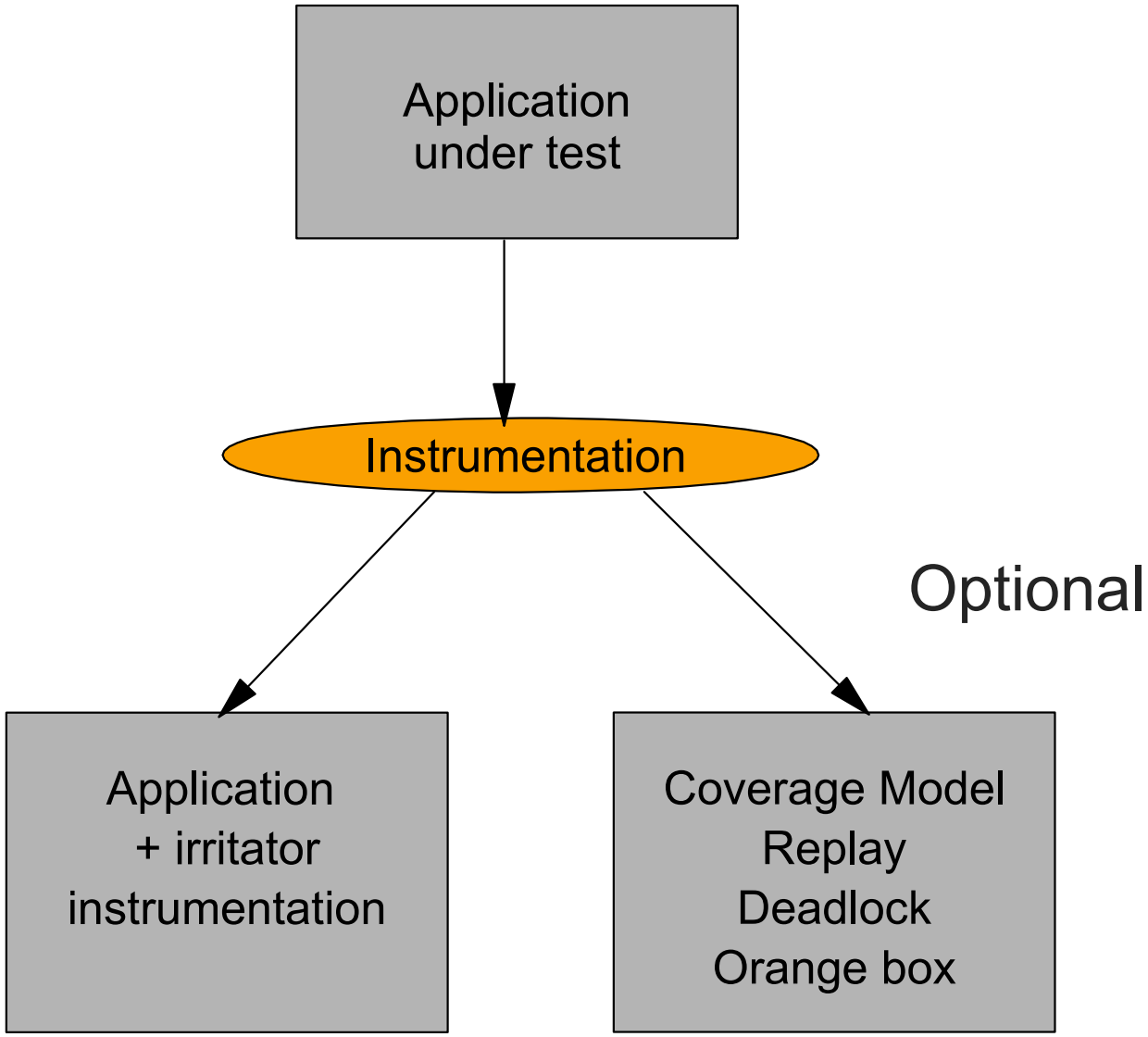
ConTest-Lite Overview

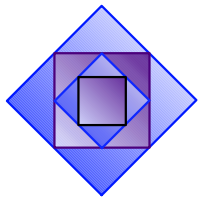
- **ConTest is composed of the following components**
  - An instrumentation engine that
    - Creates hooks for the irritator and for coverage printing
    - Generates coverage models

  The instrumentation is done at the bytecode level
  - An irritator that randomly, or using heuristics, generates new interleaving on-the-fly
  - Seed replay component
  - Coverage component
  - Deadlock component
  - Orange box component

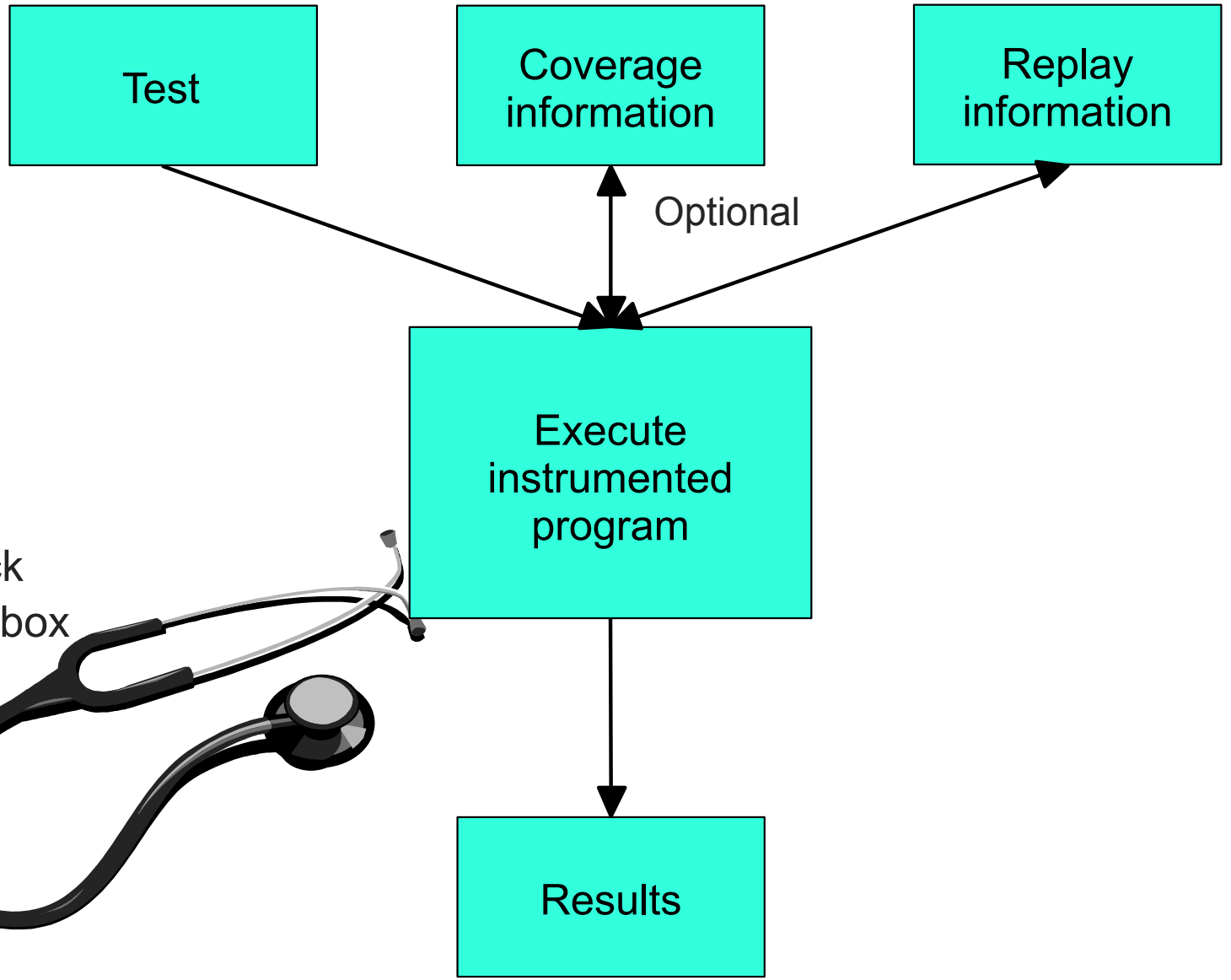


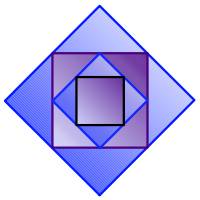
ConTest-Lite Architecture - Static View





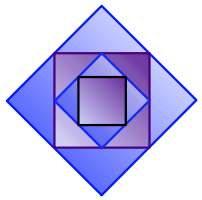
ConTest Architecture - Dynamic View





Why Bugs are Found with ConTest

- Thread switches in many places
- Random interleaving; the interleaving changes between runs of the same test
- Result: executing tests many times is more likely to find bugs



Coverage with ConTest-Lite

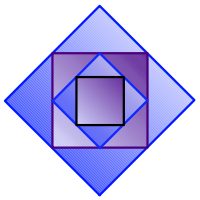
Here are the legal tasks :

Number of Tasks: 350 Covered: 151 Coverage Percentage: 43.142857

Click on a column header to sort the rows by this column

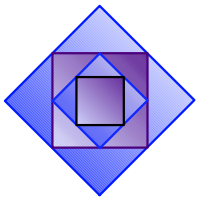
classesName	TOTAL COVER...	UNIQUE COV...	OUT OF	percentage
org.apache.tomcat.core.ResponseImpl	2657	35	58	60.344826
org.apache.jasper.compiler.DumbParseEventListener	0	0	17	0.0
org.apache.tomcat.logging.LogDaemon	153	4	5	80.0
ReaperLongServlet	0	0	2	0.0
org.apache.jasper.compiler.ForwardGenerator	0	0	2	0.0
org.apache.tomcat.util.SessionUtil$PriviledgedIdGenerator	0	0	2	0.0
org.apache.tomcat.adapter.AdapterHandler	0	0	7	0.0
org.apache.jasper.compiler.Parser$SetProperty	0	0	3	0.0
org.apache.tomcat.session.StandardManagerHA	2629	38	72	52.77778
SimpleSessionServlet	0	0	2	0.0
org.apache.tomcat.request.JspInfo	0	0	9	0.0
org.apache.tomcat.facade.ServletInputStreamFacade	0	0	7	0.0
org.apache.tomcat.service.connector.Ajpv12InputStream	0	0	4	0.0
org.apache.jasper.compiler.Mark$IncludeState	0	0	1	0.0
org.apache.tomcat.util.BuffTool	0	0	10	0.0
org.apache.jasper.servlet.JasperLoader	44	2	14	14.285714
org.apache.jasper.compiler.Parser$Tag	0	0	2	0.0
org.apache.tomcat.service.PoolTcpEndpoint	1081	11	27	40.74074
org.apache.tomcat.context.SCAction	65	4	4	100.0
org.apache.tomcat.util.CookieTools	42	6	7	85.71429
com.ibm.hatomcat.EasyVector	19	2	3	66.666664
com.ibm.hatomcat.RootObjectEntry	451	7	9	77.77778
org.apache.jasper.compiler.JspParseEventListener$BufferHandler	0	0	2	0.0
org.apache.jasper.compiler.ServletWriter	0	0	12	0.0
org.apache.jasper.compiler.ConstantPool	0	0	2	0.0
org.apache.tomcat.util.RecycleBufferedInputStream	214	3	3	100.0
org.apache.tomcat.util.xml.XmlAction	57	4	4	100.0
org.apache.tomcat.facade.HttpServletRequestFacade	320	11	51	21.568628
org.apache.jasper.servlet.ServletEngine	0	0	3	0.0
org.apache.jasper.CommandLineContext	0	0	31	0.0

Print Export Close



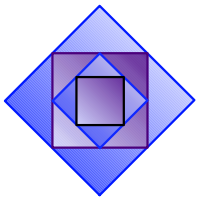
Replay of Tests with ConTest-Lite

- **Replay is very important to enable debugging**
- **Replay in ConTest-Lite is done by collecting the seed of the random function and using it again**
  - Very efficient; replay is as fast as the original run
- **Replay is not guaranteed**
  - Different interleaving in parts of the application that are not instrumented
  - Context switch due to different workloads
  - In ConTest, replay is guaranteed but the **entire** program needs to be instrumented



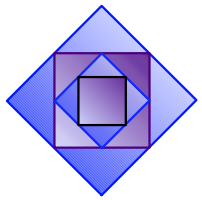
Orange Box

- `if(connection!=null)`
  `connection.setStopFlag();`
- **We know that connection==null but we don't know why**
- **Orange box to the rescue**
  - The last *N* program locations and values of variables are printed



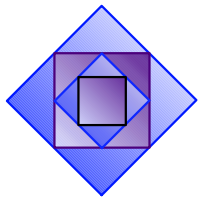
Deadlock

- **The program, or part of it, is in deadlock**
- **Deadlock support provides information on:**
  - Locks and the threads that wait on them
  - Threads and the locks they are waiting on
  - Lock owners



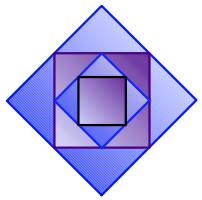
Partial Instrumentation (Pragma)

- **Sometimes it is desirable that only part of a file will be instrumented. The reasons could be:**
  - Performance
  - Debugging, adding print statements without changing the timing

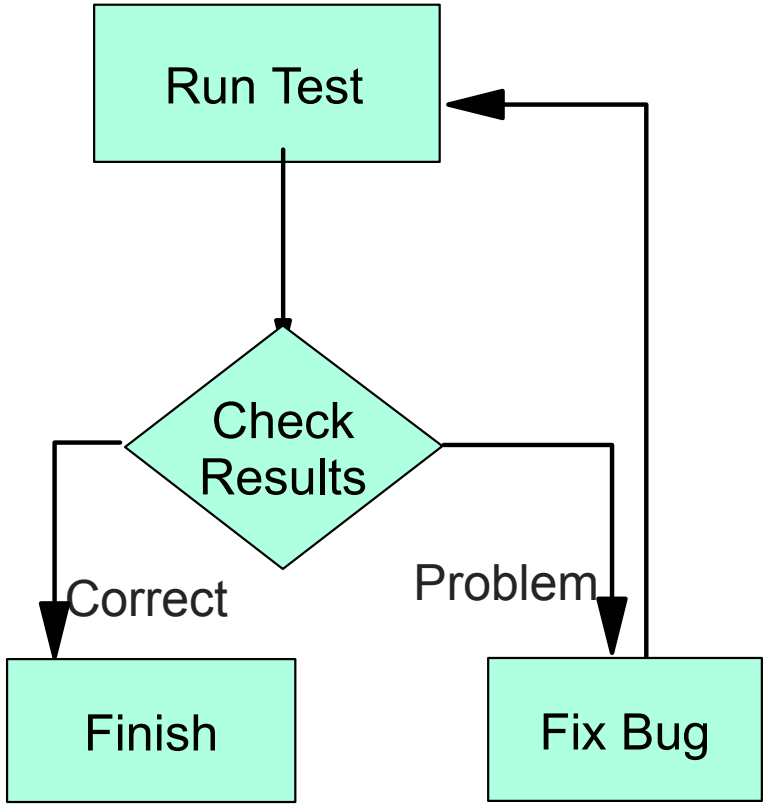


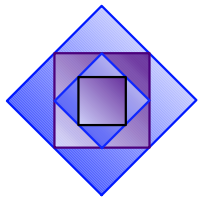
Outline

- **Why is concurrency a testing issue?**
- **Using conTestLite to find concurrent bugs is simple**
- **Concurrent bugs and why they are not found**
- **Overview of ConTest-Lite**
  - Finding concurrent bugs
  - Debugging and coverage support
- **Methodology**
  - What are good concurrent tests?
  - Re-running tests
- **Using contestLite (continued)**

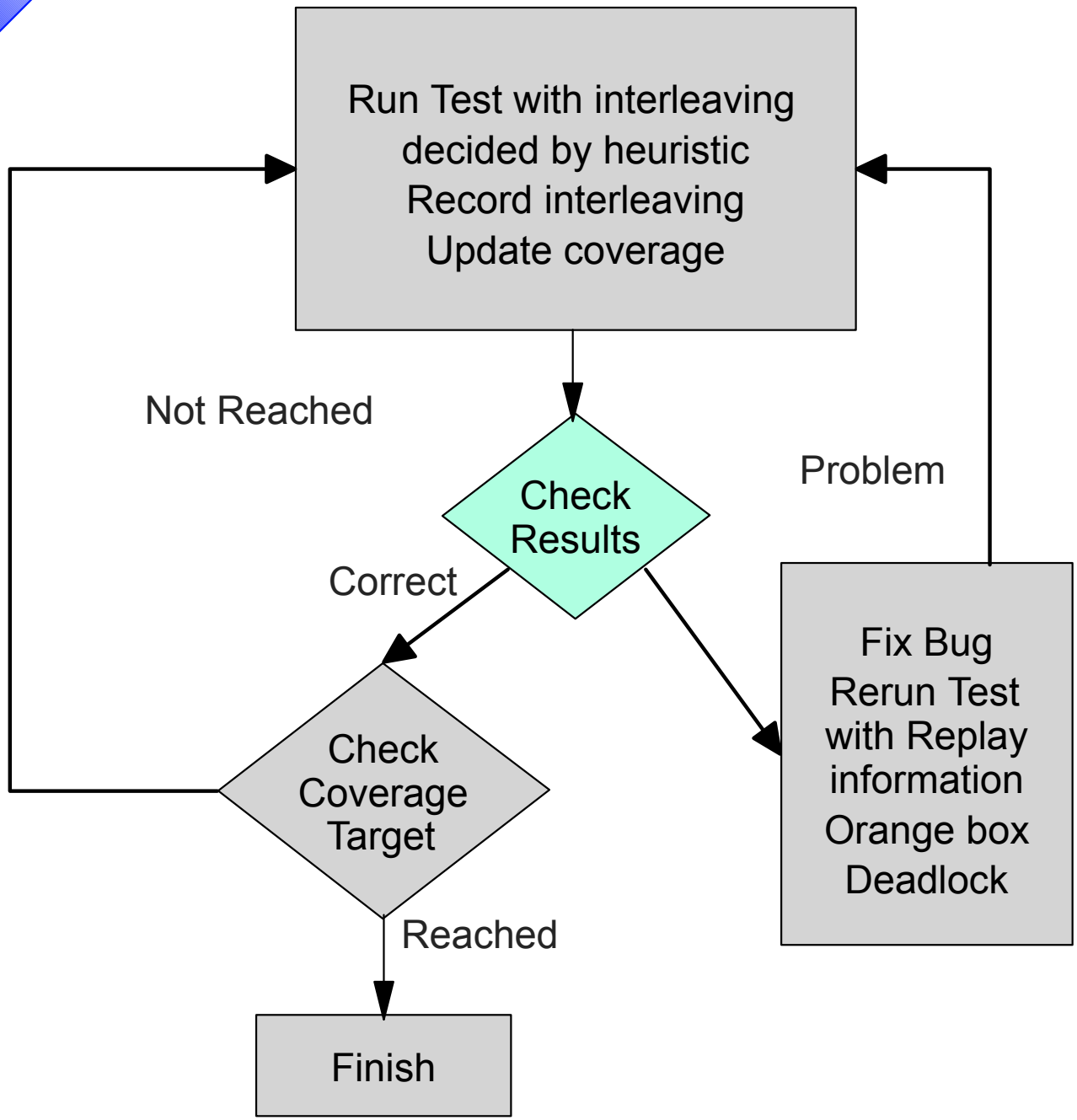


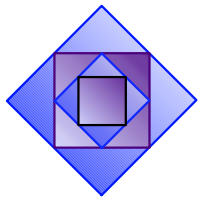
User Scenario Before ConTest





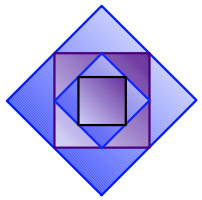
User Scenario Before ConTest





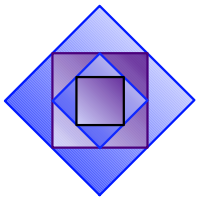
Good Tests - Multi-threading

- **Multi-threading is a MUST**
  - Simple test, multi-threaded application
  - Multi-threaded test, simple application
  - Multi-threaded test, multi-threaded application
- **Do you introduce contentions?**
- **Is the test risk driven?**
- **Running the tests in different environments**
  - What is an environment?
  - What is the importance of the environment?



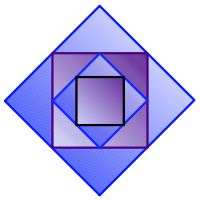
Good Tests - Automation

- **A test should be automated if it will be executed more than 10 times by hand**
- **Automation requires programming**
  - Cost is high per test
- **Automation can sometimes be done with dedicated tools (capture replay)**
- **Very useful for regression**



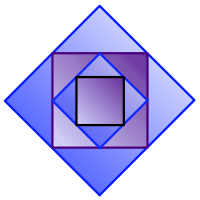
Good Tests - Comprehensive Suite

- **Every test is good -- what about the test suite?**
- **Measurement of coverage**
  - Regular coverage measurements (method, block)
  - Multi-threaded coverage measurements (interference)
- **Working with test generators**
  - Expected results
  - Automatic selection



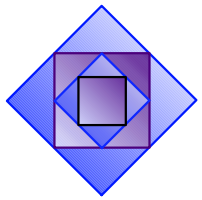
Re-running Tests

- **Without ConTest you run each test a few times in every environment**
- **With ConTest you run each test many times in one or more environments**
- **Benefits**
  - Possible to use fewer tests
  - Possible to test in fewer environments



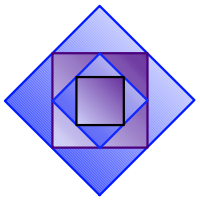
Re-running Tests - Problem and Opportunities

- **What about expected results?**
  - Getting expected results
  - Working without expected results
- **Finding a bug; how do you find it again?**
  - Running with seed
  - Maintaining additional information
- **How many times should a test be executed?**



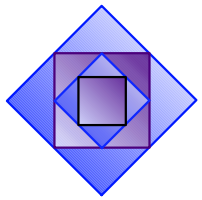
Checklist to Remember

- Many threads exist in test, application, or both
- The threads should have contention between them
- A test should contain automatic result verification
- Each test should be executed many times; seed should be saved after each run that failed
- Testing should be risk driven



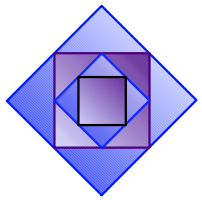
Lessons from Hursley Deployment

- **Fully automate the following:**
  - Implement while(currentCoverage < requiredCoverage);
  - On failure associate replay information with
    - log files
    - error result
    - jvm that failed
    - Also if several JVMs are invoked by the test
  - Support efficient instrumentation of a partial set of a jar file chosen using an exclude list or using a regular expression
  - Provide a specific exclude list for the rt.jar



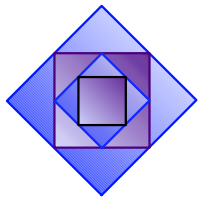
Outline

- **Why is concurrency a testing issue?**
- **Using conTestLite to find concurrent bugs is simple**
- **Concurrent bugs and why they are not found**
- **Overview of ConTest-Lite**
  - Finding concurrent bugs
  - Debugging and coverage support
- **Methodology**
  - What are good concurrent tests?
  - Re-running tests
- **Using contestLite (continued)**



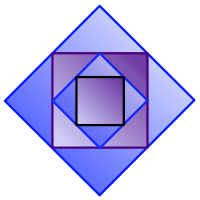
Partial Replay

- A debugging aid
- Increases the likelihood that the same synchronization bug will reoccur
- Seed of the pseudo random sequence is save and then used to reproduce the same sequence of delays that occurred in the original run



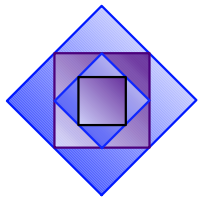
Partial Replay Limitations

- **Partial replay increases the chance that the bug will reoccur but does not guarantee it**
- **Program timing is determined by many factors:**
  - The scheduler algorithm
  - The computer architecture
  - The network delays
  - The system load
- **To increase the chance of replay neutralize factors that impact timing**
  - For example, run on a dedicated machine to neutralize system load



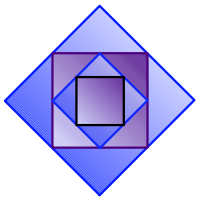
Use of Partial Replay

- Specify the seed property seed = true
- At runtime a seed value will be written to the playBackSeed{timeStamp}.txt file
- To replay this run, reset the seed property to seed = 987654321 (the number taken from the playBackSeed file)



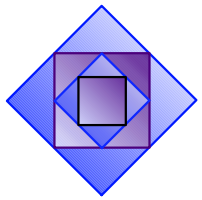
Using ConTest Heuristics

- **A heuristic is the algorithm used to**
  - Create "noise"
  - Increase the chance of revealing a synchronization bug
- **The amount of noise can be controled by a strength property**
- **Play with the heuristics to determine the maximum noise you may introduce and still get a reasonable performance**



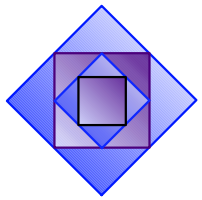
Using ConTest Heuristics

- **To use the sleeps heuristics with strength 50 specify the following in the property file:**
  - mode = sleeps
  - sleepsStrength = 50
- **The current available heuristics are**
  - yields
  - sleeps
  - synchYields



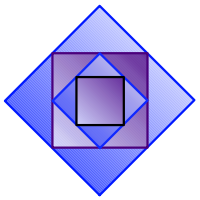
Concurrent Coverage

- **Concurrent coverage is used to check that every instrumented location has been reached**
- **Instrumenting: a list of instrumented locations is created**
- **An instrumented location is identified by**
  - file name, method name, class name, line number
  - instrumentation type
- **For every executed test a coverage trace is created**
- **Coverage is analyzed using FoCuS (Please contact us to get FoCuS)**



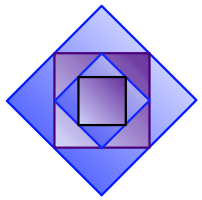
Download ConTest-Lite at:

http://w3.haifa.ibm.com/softwaretesting/ConTest



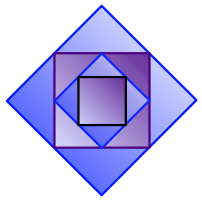
"Appendix" outline

- **A deadlock bug found by conTest in WEBSM (CBJ)**
- **Using contest advance features:**
  - Partial instrumentation of code segments
  - Deadlock support
  - Orange box support



WEBSM (CBJ) Deadlock

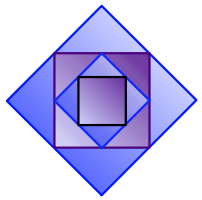
- **WEBSM is a web system management tool provided by Tivoli on AIX.**
- **WEBSM uses CBJ for its remote method invocation**
- **Bug scenario**
  - A return value of a remote invocation is written to a cyclic buffer and read by the application thread that invoked the remote call.
  - If a parameter is a reference to a remote class instance, CBJ loads the class remotely.



WEBSM (CBJ) Deadlock

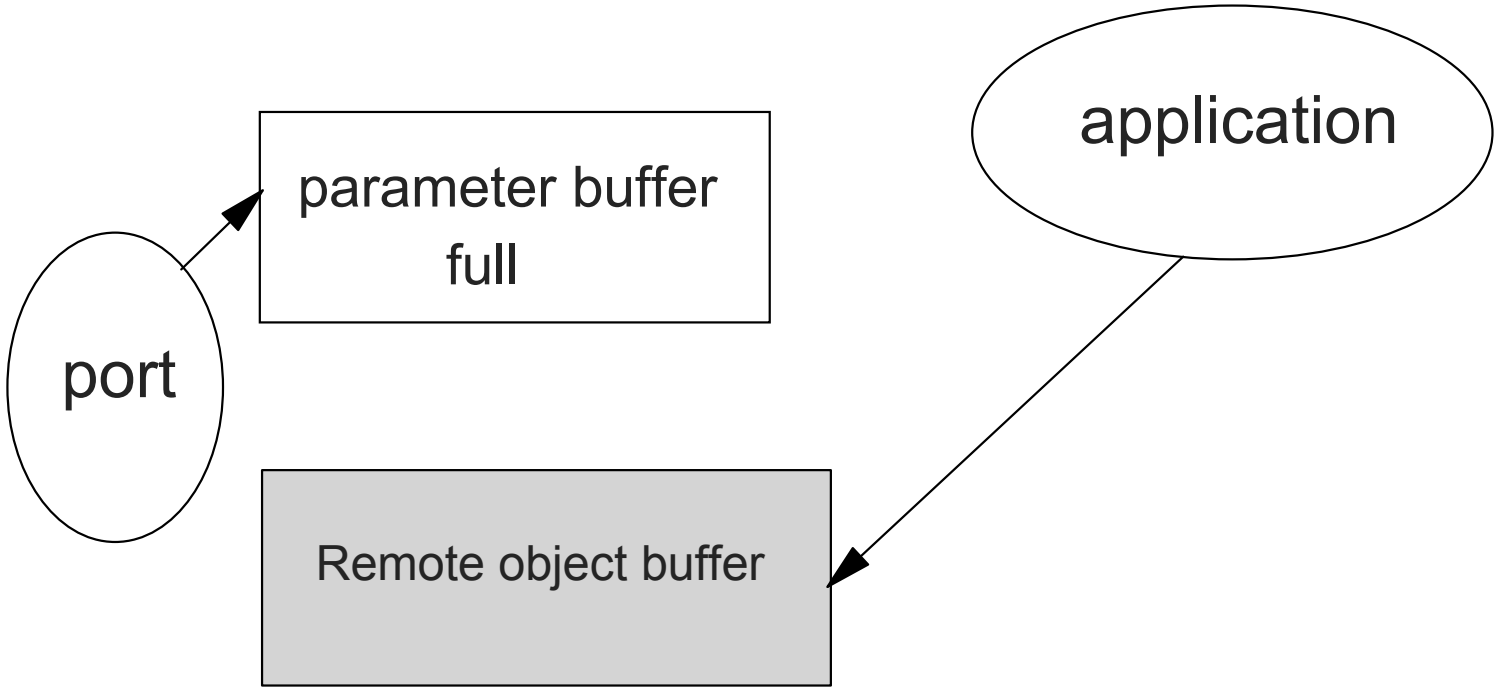
- **It may be that the application thread waits for the remote class to load while a parameter is returned**
- **If**
  - the cyclic buffer where parameters are written is full and
  - the next parameter wins the race

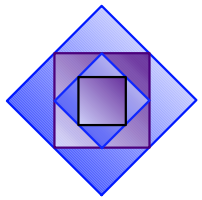
  a deadlock is created



WEBSM (CBJ) Deadlock

- **The thread listening on the port attempts to write to a full parameter buffer**
  - This buffer is never emptied since the application thread is waiting for the remote class instance (that should be written to a second buffer by the thread listening on the port. However, it is never written as this thread is waiting for the parameter buffer to be emptied.)

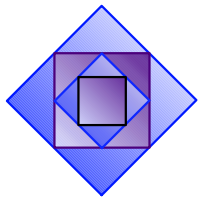




The test case that found the CBJ Deadlock

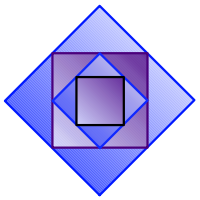
```
public Hashtable getHashApp() throws Exception {
/* App.class exists on server's classpath not on the
client !*/

      App app = new App();
      int b1[] = new int[10000];
      for (int i=0; i < b1.length; i++)
          b1[i] = i;
      int b2[] = new int[10000];
      for (int i=0; i < b2.length; i++)
          b2[i] = i;
      Hashtable h = new Hashtable();
      h.put("aaaa", b1);
      h.put("ccc", app);
      h.put("eee", b2);
      return h;
  }
```



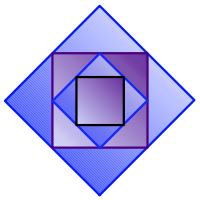
Partial Instrumentation

- **Partial instrumentation is important as**
  - Coverage is collected only on the instrumented parts
    - Used, for example, when coverage needs to be collected only on new functionality
  - Uninstrumented print statements can be added without affecting replay
  - Performance is enhanced
- **Partial instrumentation can be done**
  - by choosing a subset of the application files
  - by instrumenting part of a specific file



Instrumenting Part of a File

- **In the java file**
  - Instrumentation.pragma ("off"); will turn the instrumentation off
  - Instrumentation.pragma ("on"); will turn it on
- **conTest's jar file should be available to the application when compiling**



Deadlock and Orange Box Support

- **Agent1 and Agent2 try to capture two different locks, but in reverse order**
- **Thus, the program has a deadlock**
- **An auxiliary thread**
  - activates the deadlock support mechanism
  - calls the required APIs to print deadlock information
- **Orange box is also used by introducing an auxiliary thread**