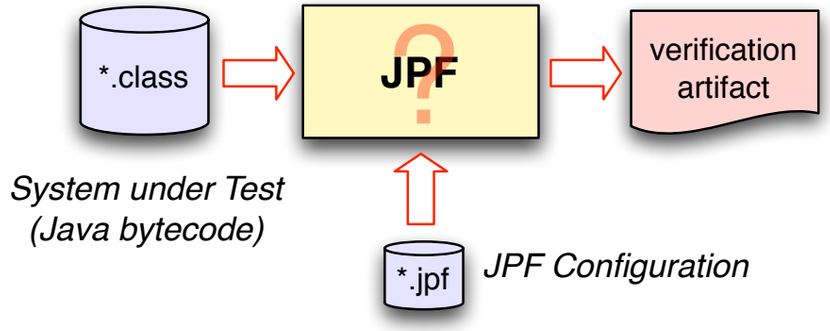
Using Java PathFinder for Program Understanding and Defect Visualization

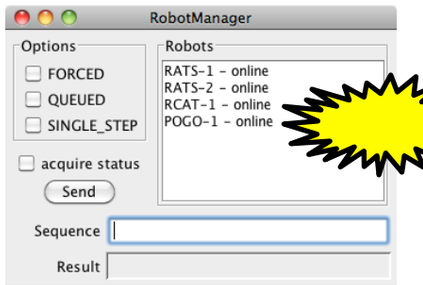
Peter C. Mehlitz
SGT / NASA Ames Research Center
<Peter.C.Mehlitz@nasa.gov>

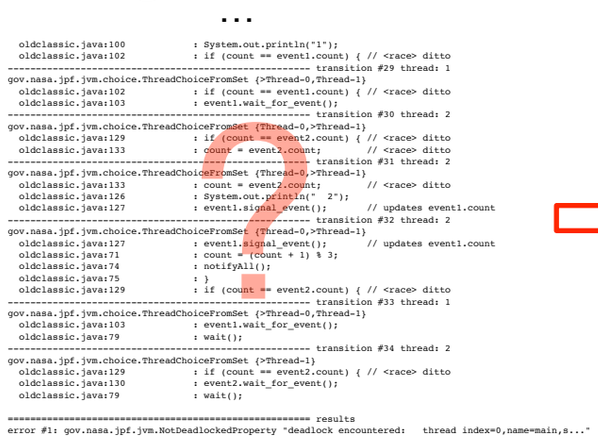
Roadmap

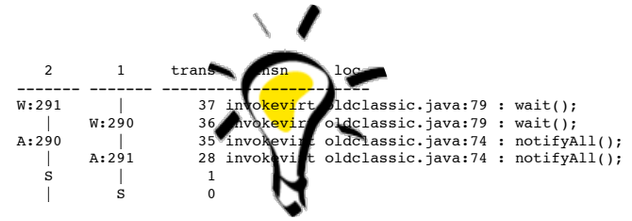
- Introduction - What is Java Pathfinder (JPF)



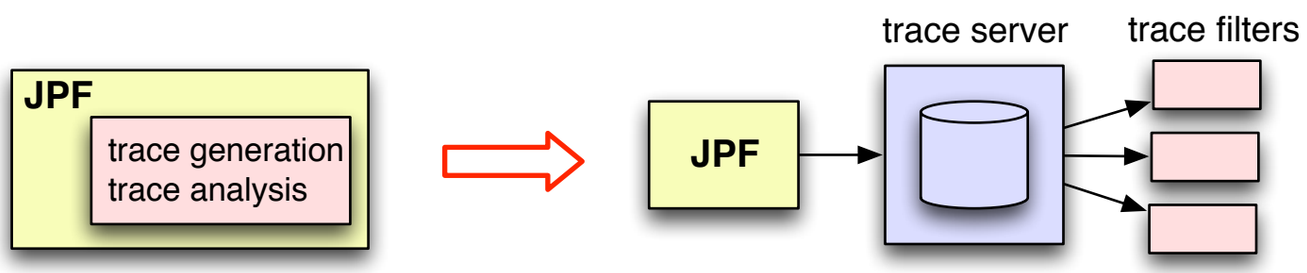
- Examples - What is in a Program Trace







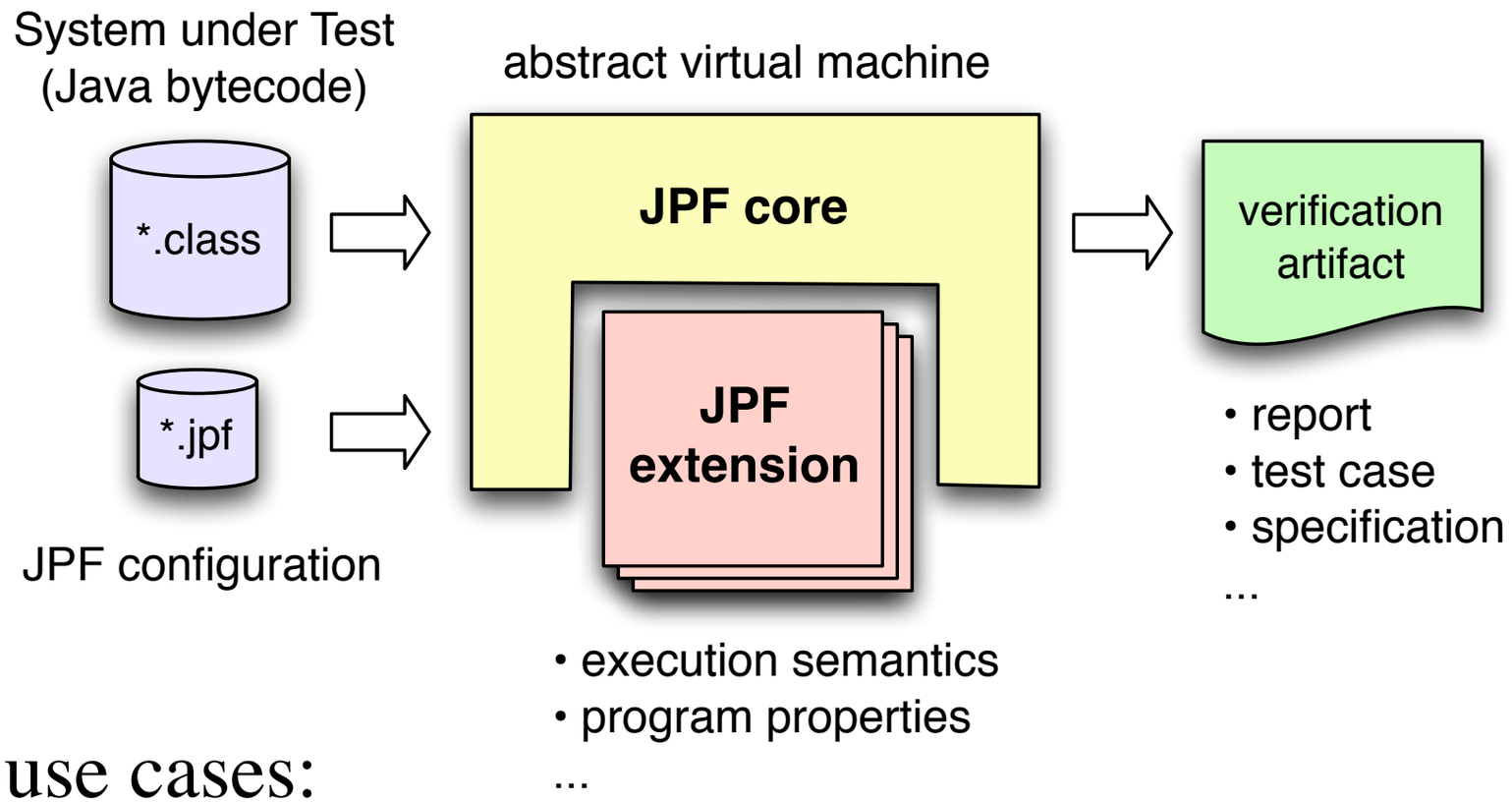
- Outlook - How *should* this be analyzed



Introduction - What is Java Pathfinder?

Ames Research Center

- surprisingly hard to summarize - can be used for many things
- extensible virtual machine framework for Java bytecode verification: workbench to efficiently implement all kinds of verification tools



- typical use cases:
  - software model checking (deadlock & race detection)
  - deep inspection (numeric analysis, invalid access)
  - test case generation (symbolic execution)
  - ... and many more

Introduction - What has JPF been used for?

Ames Research Center

Objective: reproduce - *and analyze* - spurious deadlock in large web application

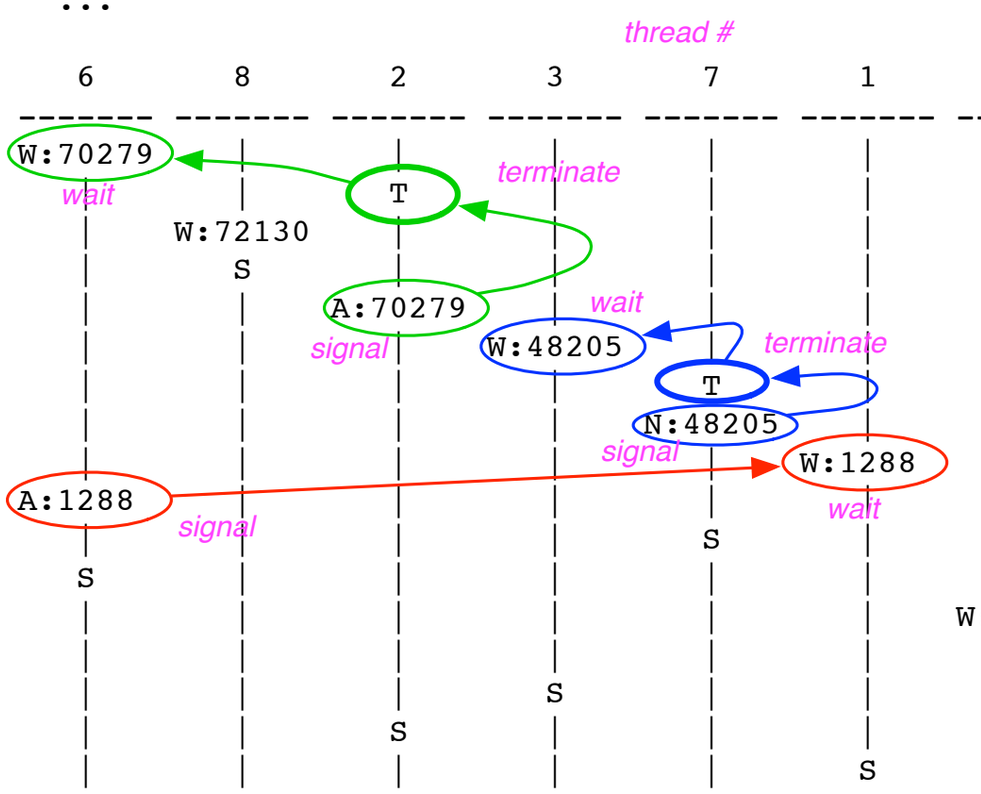
required stubbing, trace file >70MB, would have taken months to analyze manually

```
========================================== error #1
deadlock encountered:
  thread index=0,name=main,status=TERMINATED
  thread index=1,name=Thread-0,status=WAITING
  thread index=2,name=Thread-1,status=TERMINATED
  thread index=3,name=Thread-2,status=WAITING
  ...
```

3 pairs of *missed signals* in set of 18 threads

missed signal = thread A signals before thread B waits

→ B is blocked on "lost" signal




```
   trans     insn      loc
-------- ---------- ----------------------
  123009 invokevirt awtMutex.java:38 : try {wait();}
  123008     return MessageAnalyzer.java:341 : }
  122597 invokevirt MessageAnalyzer.java:96 : wai
  122593
  122572 invokevirt awtMutex.java:48 : notifyAll();
  122541 invokevirt awtQueue.java:182 : wait();
  122538     return EnforceSendMessageThread.java:39 :
  122538 invokevirt awtQueue.java:150 : notify();
      96 invokevirt Mutex.java:31 : wait();
      87 invokevirt Mutex.java:44 : notifyAll();
      33
      33
W:48252   29 invokevirt awtQueue.java:182 : wait();
   S       9
           7
           2
           0
```

```
...
========================================== statistics
elapsed time:       3:53:23
states:             new=123010, visited=235893, backtracked=357879, end=81378
search:             maxDepth=1023, constraints=0
choice generators:  thread=94664, data=0
heap:               gc=599818, new=7687542, free=12860670
instructions:       276556150
max memory:         9838MB
```

just one success story for the type of deep defects targeted by JPF

Introduction - Who is Using JPF?

Ames Research Center

- major user group is academic research - collaborations with >20 universities worldwide (uiuc.edu, unl.edu, byu.edu, umn.edu, Stellenbosch Za, Waterloo Ca, AIST Jp, Charles University Prague Cz, ..)
- companies not so outspoken (exception Fujitsu - see press releases, e.g. http://www.fujitsu.com/global/news/pr/archives/month/2010/20100112-02.html) , but used by several Fortune 500 companies
- lots of (mostly) anonymous and private users (~1000 hits/day on website, ~10 downloads/day, ~60 read transactions/day, initially 6000 downloads/month)
- many uses inside NASA, but mostly model verification at Ames Research Center

Introduction - History

Ames Research Center

- not a new project: around since 10 years and continuously developed:
  - 1999 - project started as front end for Spin model checker
  - 2000 - reimplementation as concrete virtual machine for software model checking (concurrency defects)
  - 2003 - introduction of extension interfaces
  - 2005 - open sourced on Sourceforge
  - 2008 - participation in Google Summer of Code
  - 2009 - new project structure, configuration and startup mechanism, moved to own server, hosting extension projects and Wiki
  - 2010 - extended summer program for students (GSoC + other funding)

- JPF now in its 5th development cycle

Introduction - Awards

- widely recognized, awards for JPF in general and for related work, team and individuals
  - 2002 - ACM Sigsoft Distinguished Paper award
  - 2003 - “Turning Goals into Reality” (TGIR) Engineering Innovation Award from the Office of AeroSpace Technology
  - 2004, 2005 - Ames Contractor Council Awards
  - 2007 - IBM's Haifa Verification Conference (HVC) award
  - 2009 - “Outstanding Technology Development” award of the Federal Laboratory Consortium for Technology Transfer (FLC)

Introduction - Where to Learn More

Ames Research Center

http://babelfish.arc.nasa.gov/trac/jpf

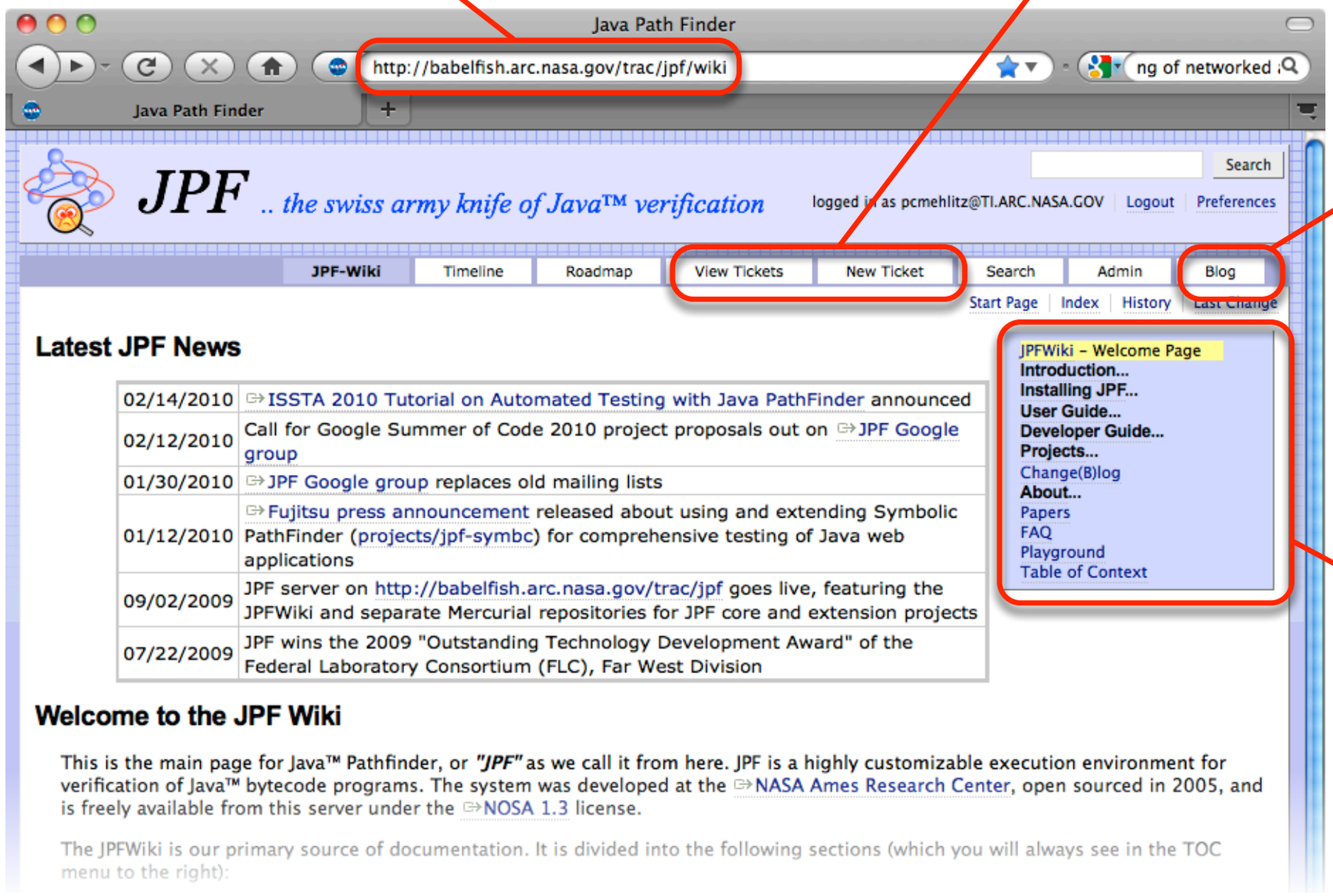
- public read access
- edit for account holders (also non-NASA)

bug tracking

- Trac ticket system

project blog

- announcements
- important changes



hierarchical navigation menu

- intro
- installation
- user docu
- developer docu
- **extension projects**

Introduction - Key Points

Ames Research Center

- JPF mostly research platform *but also* production tool (basis)
- JPF is designed for extensibility
- JPF is open source
- JPF is an ongoing collaborative development project
- JPF is moderately sized system (~200ksloc core + extensions)
- JPF represents >20 man year development effort
- JPF is pure Java application (platform independent)

- JPF cannot find all bugs
  - but as of today -
    - some of the most expensive bugs only JPF can find

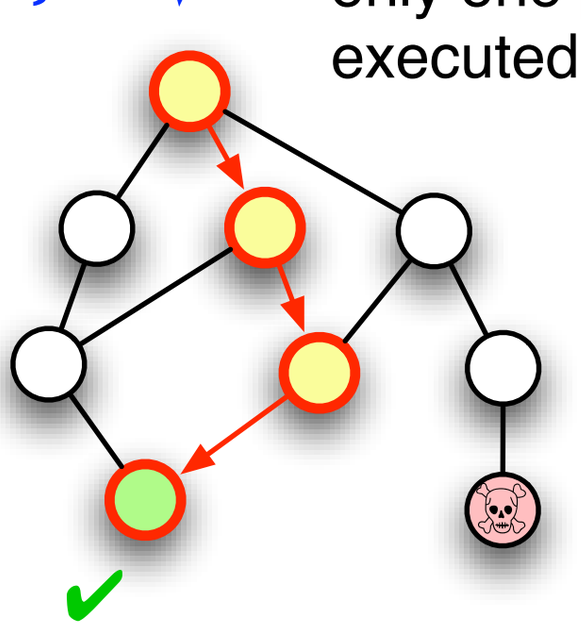
Basics - Model Checking

- Model Checking (MC) is a rigorous formal method
- MC does not suffer from false positives (like static analysis)
- MC provides traces (execution history) when it finds a defect

testing:

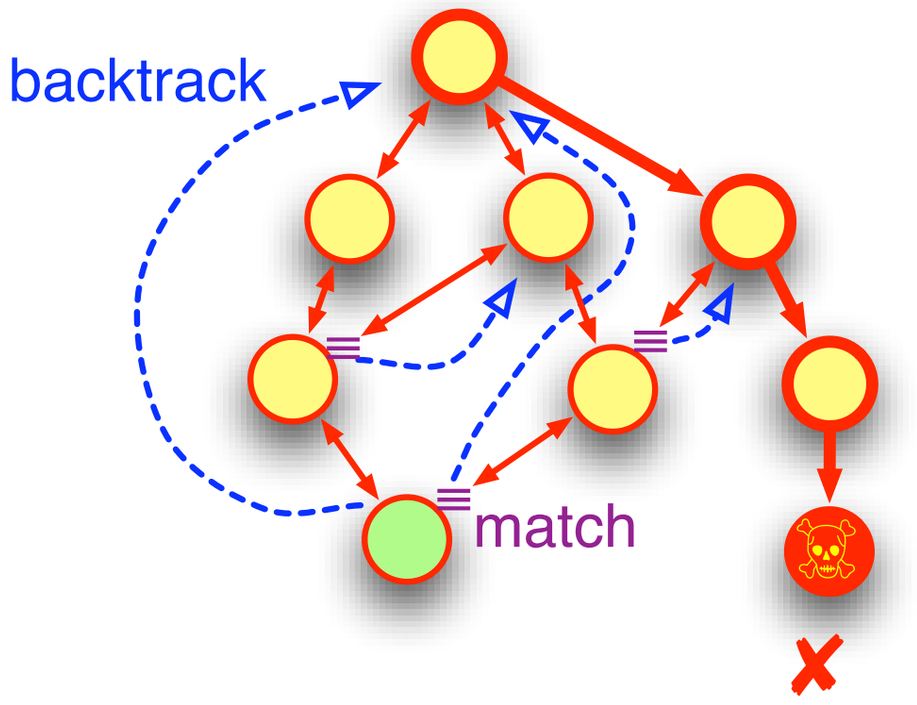
{d}

based on input set {d}
only one path
executed at a time



model checking:

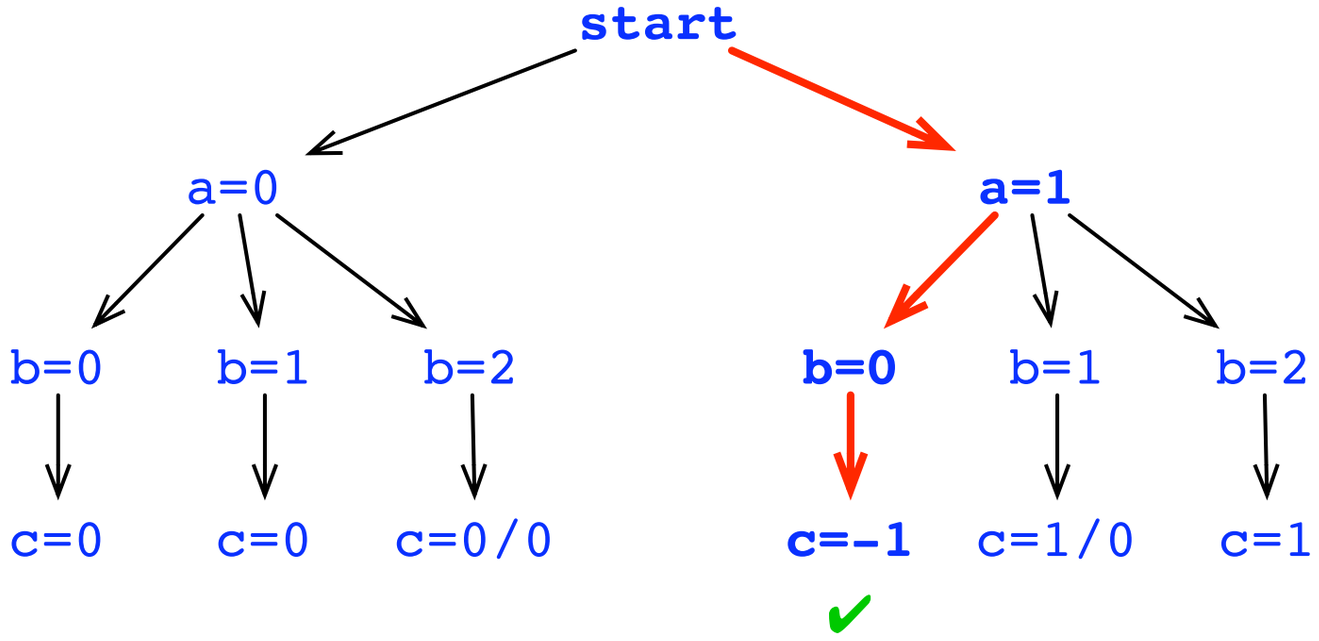
all program state are explored
until none left or defect found



Basics - Model Checking Example

Ames Research Center

- Testing only covers one execution



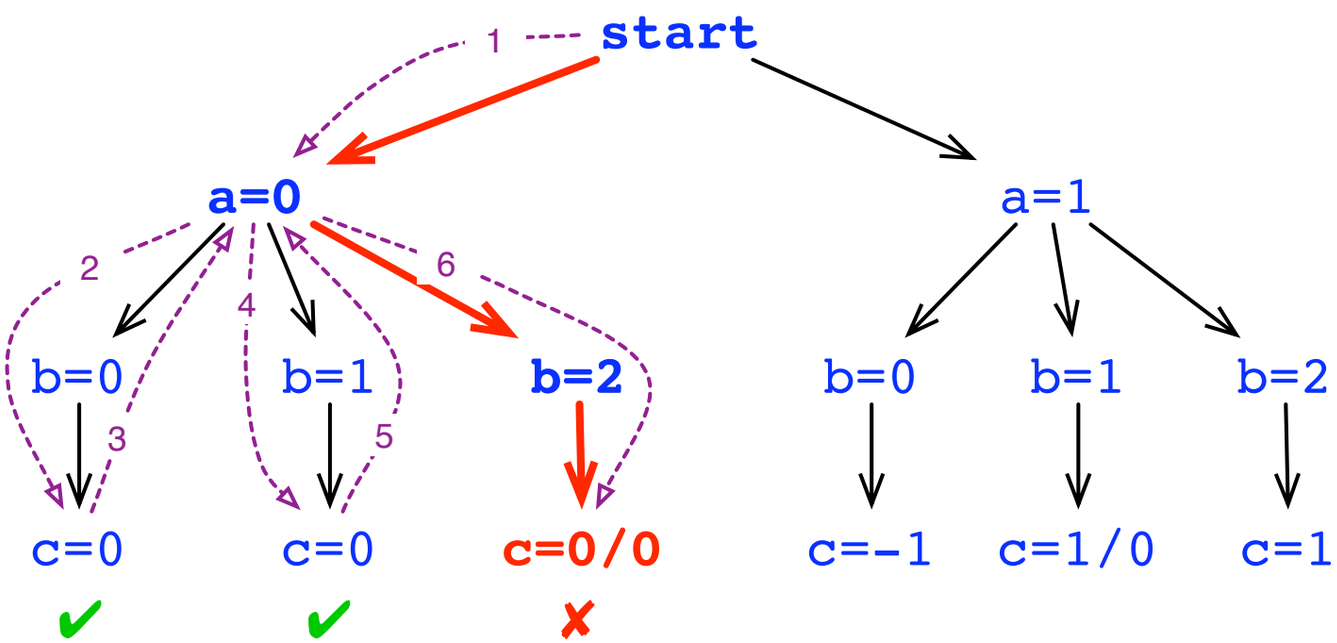
① `Random random = new Random()`

② `int a = random.nextInt(2)`

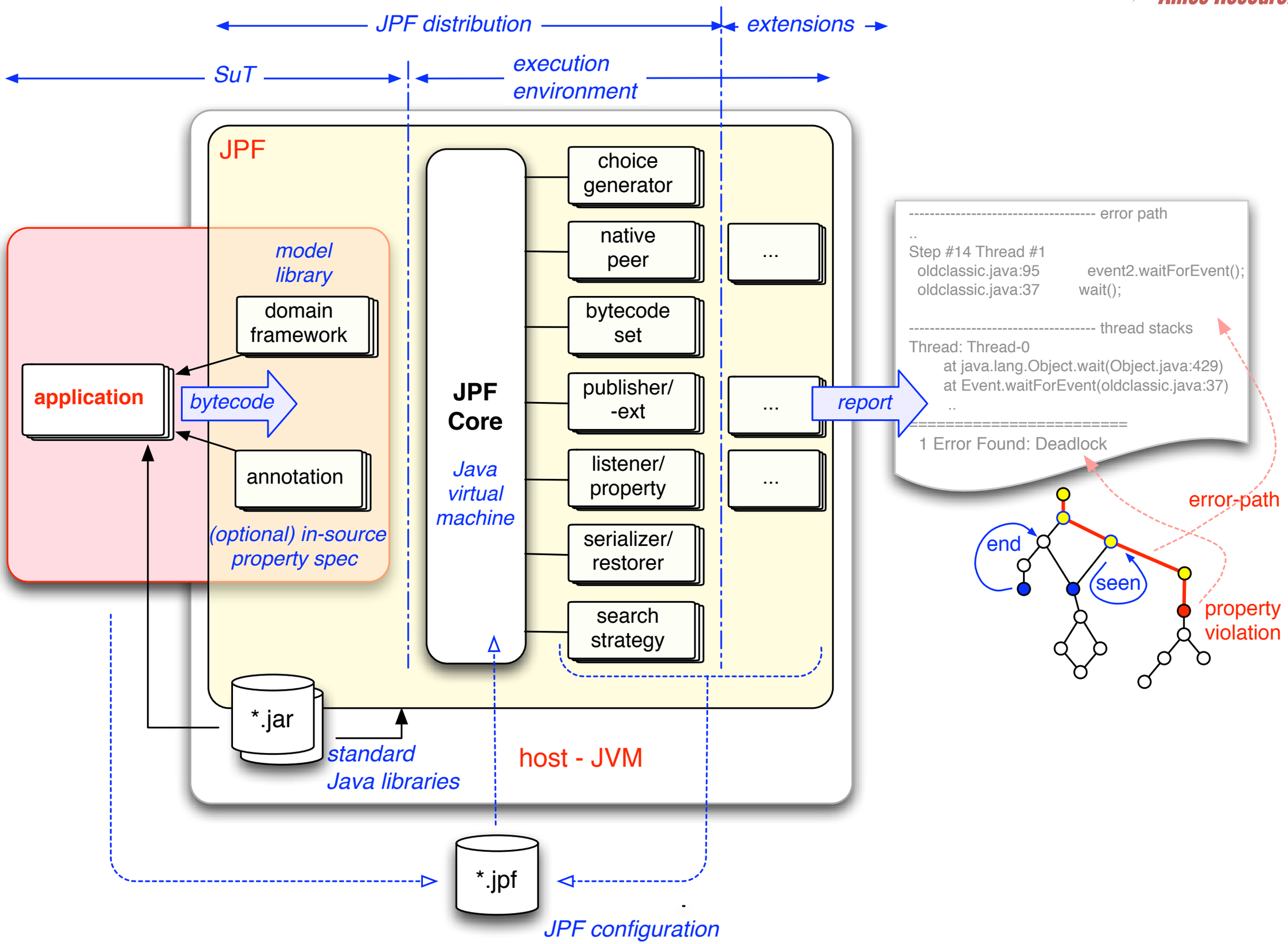
③ `int b = random.nextInt(3)`

④ `int c = a/(b+a -2)`

- Model checking (theoretically) covers all executions

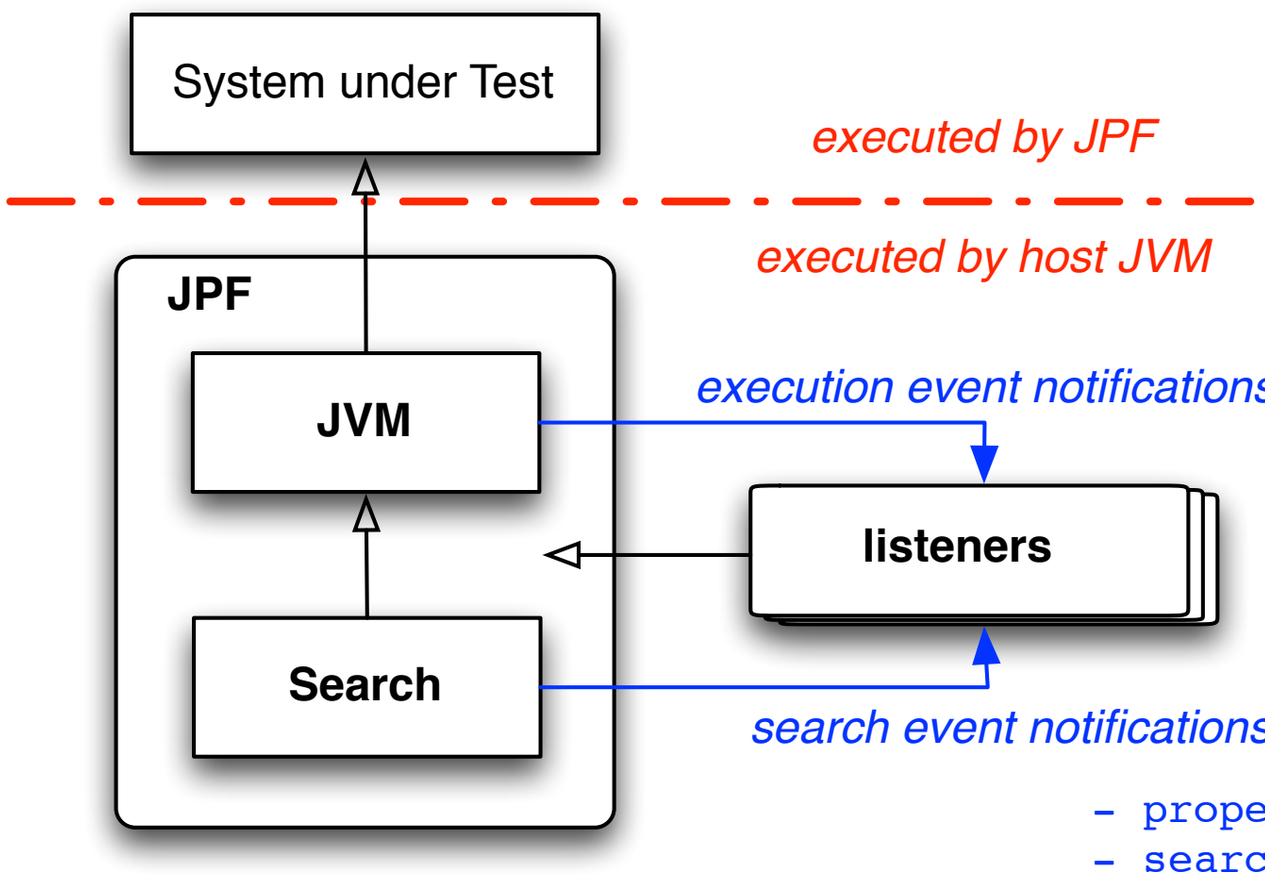


Basics - JPF Under the Hood



Basics - JPF Listeners

- runtime configured “JPF plugins”
- can observe state space search and bytecode execution
- suitable extension mechanism for trace generation & analysis



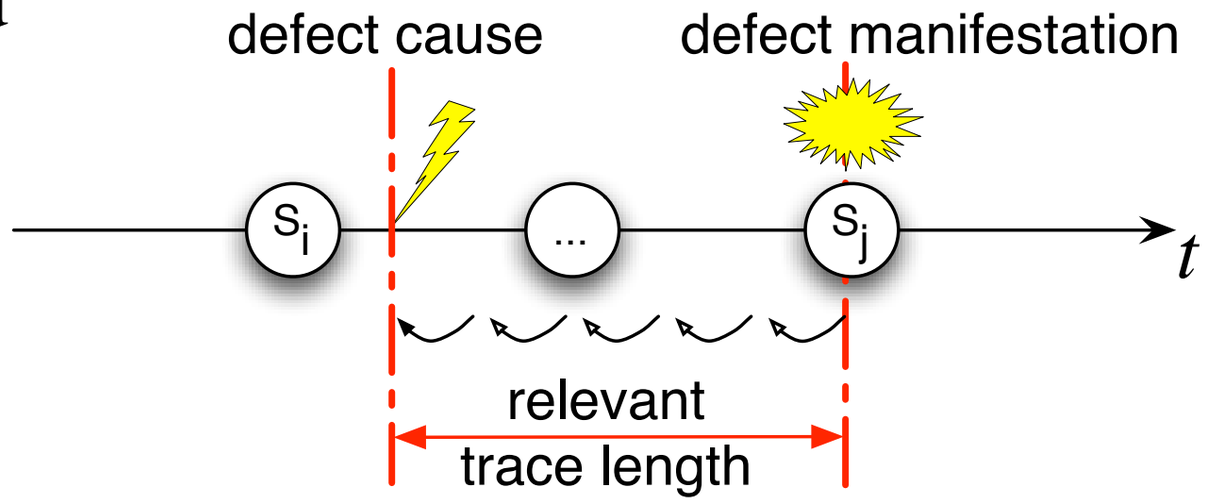

```
- classLoaded()
- threadScheduled()
- threadNotified()
  ...
- executeInstruction()
- instructionExecuted()
- objectCreated()
  ...
- exceptionThrown()
  ...
- choiceGeneratorAdvanced()
  ...
```

```
- +listener=<listener-class>
- @JPFConfig(..)
- listener.autoload=<annotations>
- jpf.addListener(..)
  ...
```

```
- propertyViolated()
- searchContraintHit
- searchFinished()
  ...
```

Examples - What is in a Trace?

- finding a defect is only the beginning - we have to understand what caused it
- JPF can remember the whole execution history that lead to the defect manifestation
- BUT: manifestation can be a long way downstream from where the bug was caused



- we need to find the shortest, most appropriate representation of the relevant trace that explains the defect

Example 1 - Deadlock (1)

```
class FirstTask extends Thread {
  Event event1, event2;
  int   count = 0;
  ...
  public void run () {
    count = event1.count;

    while (true) {
      if (count == event1.count) {
        event1.wait_for_event();
      }

      count = event1.count;
      event2.signal_event();
    }
  }
}
```

```
class SecondTask extends Thread {
  Event event1, event2;
  int   count = 0;
  ...
  public void run () {
    count = event2.count;

    while (true) {
      event1.signal_event();

      if (count == event2.count) {
        event2.wait_for_event();
      }

      count = event2.count;
    }
  }
}
```

```
class Event {
  int count = 0;

  public synchronized void signal_event () {
    count++;  notifyAll();
  }
  public synchronized void wait_for_event () {
    .. wait(); ..
  }
```

Example 1 - Deadlock (2)

- Report - shows *what* happened, but not *why*

```
JavaPathfinder v5.x - (C) RIACS/NASA Ames Research Center

========================== system under test
application: oldclassic.java

========================== search started: 2/11/10 11:10 AM
1
  2
1
  2
1
...

========================== error #1
gov.nasa.jpf.jvm.NotDeadlockedProperty
deadlock encountered:
  thread index=0,name=main,status=TERMINATED,this=null,target=null,...
  thread index=1,name=Thread-0,status=WAITING,this=FirstTask@295,lockCount=1,...
  thread index=2,name=Thread-1,status=WAITING,this=SecondTask@322,lockCount=1,...
...
```

Example 1 - Deadlock (3)

- traces shows why, but not very efficiently

```
================== trace #1
------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
      [2843 insn w/o sources]
  oldclassic.java:47                : Event       new_event1 = new Event();
…
------------------------------------ transition #29 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  oldclassic.java:102               : if (count == event1.count) {
  oldclassic.java:103               : event1.wait_for_event();
------------------------------------ transition #30 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {Thread-0,>Thread-1}
  oldclassic.java:129               : if (count == event2.count) {
  oldclassic.java:133               : count = event2.count;
...
  oldclassic.java:126               : System.out.println("  2");
  oldclassic.java:127               : event1.signal_event();
...
  oldclassic.java:71                : count = (count + 1) % 3;
  oldclassic.java:74                : notifyAll();
  oldclassic.java:75                : }
  oldclassic.java:129               : if (count == event2.count) {
------------------------------------ transition #33 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  oldclassic.java:103               : event1.wait_for_event();
  oldclassic.java:79                : wait();
------------------------------------ transition #34 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-1}
  oldclassic.java:129               : if (count == event2.count) {
  oldclassic.java:130               : event2.wait_for_event();
  oldclassic.java:79                : wait();
```

Example 1 - Deadlock (3)

- traces shows why, but not very efficiently

```
================== trace #1
------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
      [2843 insn w/o sources]
  oldclassic.java:47                : Event       new_event1 = new Event();
…
------------------------------------ transition #29 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  oldclassic.java:102               : if (count == event1.count) {
  oldclassic.java:103               : event1.wait_for_event();
------------------------------------ transition #30 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {Thread-0,>Thread-1}
  oldclassic.java:129               : if (count == event2.count) {
  oldclassic.java:133               : count = event2.count;
...
  oldclassic.java:126               : System.out.println("  2");
  oldclassic.java:127               : event1.signal_event();
...
  oldclassic.java:71                : count = (count + 1) % 3;
  oldclassic.java:74                : notifyAll();
  oldclassic.java:75                : }
  oldclassic.java:129               : if (count == event2.count) {
------------------------------------ transition #33 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  oldclassic.java:103               : event1.wait_for_event();
  oldclassic.java:79                : wait();
------------------------------------ transition #34 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-1}
  oldclassic.java:129               : if (count == event2.count) {
  oldclassic.java:130               : event2.wait_for_event();
  oldclassic.java:79                : wait();
```

missed signal

Example 1 - Deadlock (4)

Ames Research Center

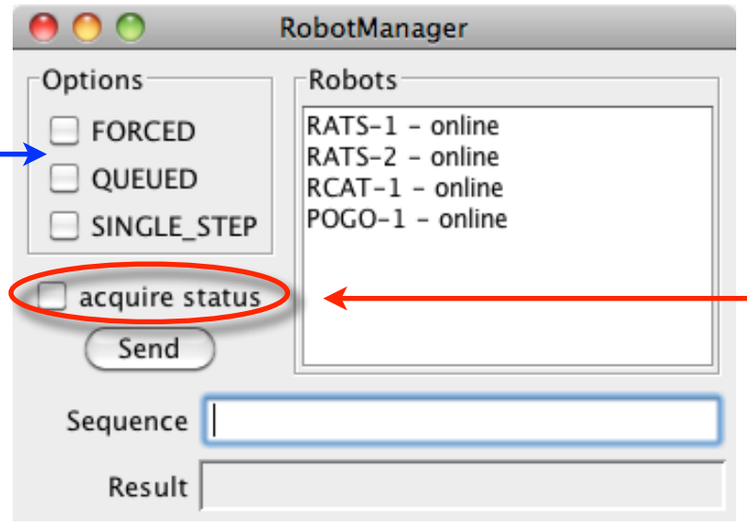
- creating trace of threading relevant operations explains deadlock in 4 lines: *missed signal*

```
report.console.property_violation = error,snapshot
listener = .listener.DeadlockAnalyzer

====================== error #1
gov.nasa.jpf.jvm.NotDeadlockedProperty
deadlock encountered:
...
====================== snapshot #1
thread index=1,name=Thread-0,status=WAITING ...
  waiting on: Event@290
  ...

thread index=2,name=Thread-1,status=WAITING ...
  waiting on: Event@291
  ...

======================== thread ops #1
   2        1     trans    insn     loc
------- ------- ----------------------
W:291       |        37 invokevirt oldclassic.java:79 : wait();
   |     W:290       36 invokevirt oldclassic.java:79 : wait();
A:290      |         35 invokevirt oldclassic.java:74 : notifyAll();
   |    A:291       28 invokevirt oldclassic.java:74 : notifyAll();
   S       |         1
   |       S         0
```

Example 2 - Multithreaded GUI (1)

- extended GUI application that uses concurrent data acquisition
- thread structure not obvious because of large portions of framework / library code (swing)
- application logic mostly implemented as callback actions
- two overlaid non-determinisms: user input and scheduling sequence
- “impossible” to test

Thread 1:
user input



Thread 2:
data acquisition

Example 2 - Multithreaded GUI (2)

- JPF finds defect that is untestable, but..

general, low level defect

UIShell

Properties | Report | Test Output | Verify Output | Components | Trace | Script

```
JavaPathfinder v5.x - (C) RIACS/NASA Ames Research Center

========================================================== system under test
application: RobotManager.java

========================================================== search started: 2/17/10 10:03 AM

========================================================== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.NullPointerException: calling 'processSequence(Ljava/lang/String;)Ljava/lang/String;' o
        at RobotManager.sendSequence(RobotManager.java:266)
        at RobotManagerView.sendSequence(RobotManager.java:538)
        at RobotManagerView$3.actionPerformed(RobotManager.java:339)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:113)
        at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:41)
        at javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:387)
        at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:242)
        at javax.swing.AbstractButton.doClick(AbstractButton.java:187)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:65)

========================================================== snapshot #1
thread index=1,name=AWT-EventQueue-0,status=RUNNING,this=java.awt.EventDispatchThread@2595,priori
  call stack:
        at RobotManager.sendSequence(RobotManager.java:266)
        at RobotManagerView.sendSequence(RobotManager.java:538)
        at RobotManagerView$3.actionPerformed(RobotManager.java:339)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:113)
        at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:41)
        at javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:387)
        at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:242)
        at javax.swing.AbstractButton.doClick(AbstractButton.java:187)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:65)

thread index=2,name=Thread-1,status=SLEEPING,this=RobotStatusAcquisitionThread@2635,priority=5,lo
  call stack:
```

Example 2 - Multithreaded GUI (3)

- trace too long to find out what causes it

too deep
to understand

UIShell

Properties | Report | Test Output | Verify Output | Components | Trace | Script

```
  RobotManager.java:257          : return onlineRobots.get(robotName);
  RobotManager.java:537          : Robot robot = model.getOnlineRobot(selectedRobotName);
  RobotManager.java:538          : String result = model.sendSequence(robot, sequence);
------------------------------------------------------ transition #169 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-1,AWT-EventQueue-0}
  RobotManager.java:245          : listModel.changed(robot);
  RobotManager.java:184          : int idx = robots.indexOf(robot);
------------------------------------------------------ transition #170 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-1,AWT-EventQueue-0}
  RobotManager.java:184          : int idx = robots.indexOf(robot);
      [23 insn w/o sources]
  RobotManager.java:184          : int idx = robots.indexOf(robot);
  RobotManager.java:185          : if (idx >= 0) {
  RobotManager.java:186          : fireContentsChanged(this, idx, idx);
      [17 insn w/o sources]
  RobotManager.java:188          : }
  RobotManager.java:246          : }
  RobotManager.java:151          : Thread.sleep(3000);
      [3 insn w/o sources]
------------------------------------------------------ transition #171 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>AWT-EventQueue-0,Thread-1}
  RobotManager.java:538          : String result = model.sendSequence(robot, sequence);
  RobotManager.java:266          : return robot.processSequence(sequence);

====================================================== results
error #1: gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty "java.lang.NullPointerException: calling

====================================================== statistics
elapsed time:       0:00:17
states:             new=8812, visited=8917, backtracked=17557, end=0
search:             maxDepth=836, constraints=0
choice generators:  thread=8220, data=592
heap:               gc=18126, new=15942, free=11676
instructions:       870251
max memory:         84MB
loaded code:        classes=350, methods=4109

====================================================== search finished: 2/17/10 10:03 AM
```

Example 2 - Multithreaded GUI (4)

- generic high level trace analysis can help (e.g. hierarchical method execute/return log with MethodAnalyzer listener)
- still too time consuming in many cases

```
listener=.listener.MethodAnalyzer
method.include=*Robot*
...
```

```
========================================================= method ops #1
1: >-  ......... RobotManagerView$2@642.itemStateChanged(Ljava/awt/event/ItemEvent;)V
1: >-  .......... RobotManagerView@413.acquireRobotStatus(Z)V
1: >-  ........... RobotManager@287.startAcquisitionThread()V
1: >-  ............ RobotStatusAcquisitionThread@2635.<init>(LRobotManager;)V
1:   < ............ RobotStatusAcquisitionThread@2635.<init>(LRobotManager;)V
1:   < ........... RobotManager@287.startAcquisitionThread()V
1:   < .......... RobotManagerView@413.acquireRobotStatus(Z)V
                          ...
1: >-  ......... RobotManagerView$5@606.itemStateChanged(Ljava/awt/event/ItemEvent;)V
1:   < ......... RobotManagerView$5@606.itemStateChanged(Ljava/awt/event/ItemEvent;)V
1: >-  .... RobotManager$ListModel@323.getSize()I
1:   < .... RobotManager$ListModel@323.getSize()I
1: >-  ............ RobotManagerView$6@848.valueChanged(Ljavax/swing/event/ListSelectionEvent;)V
1: >-  ............... RobotManager$ListModel@323.getElementAt(I)Ljava/lang/Object;
1:   < ............... RobotManager$ListModel@323.getElementAt(I)Ljava/lang/Object;
1:   < ............ RobotManagerView$6@848.valueChanged(Ljavax/swing/event/ListSelectionEvent;)V
1: >-  ........ RobotManagerView$3@885.actionPerformed(Ljava/awt/event/ActionEvent;)V
1: >-  ......... RobotManagerView@413.sendSequence()V
1: >-  .......... RobotManager@287.isRobotOnline(Ljava/lang/String;)Z
1:   < .......... RobotManager@287.isRobotOnline(Ljava/lang/String;)Z
1: >-  .......... RobotManager@287.getOnlineRobot(Ljava/lang/String;)LRobot;
----------------------------------------
2: >-  .. RobotManager@287.getListModel()LRobotManager$ListModel;
2:   < .. RobotManager@287.getListModel()LRobotManager$ListModel;
2: >-  .. RobotManager$ListModel@323.getSize()I
2:   < .. RobotManager$ListModel@323.getSize()I
2: >-  .. RobotManager$ListModel@323.getElementAt(I)Ljava/lang/Object;
2:   < .. RobotManager$ListModel@323.getElementAt(I)Ljava/lang/Object;
2: >-  .. RobotManager@287.isRobotOnline(Ljava/lang/String;)Z
2:   < .. RobotManager@287.isRobotOnline(Ljava/lang/String;)Z
2: >-  .. RobotManager@287.setRobotOnline(LRobot;Z)V
----------------------------------------
1:   < .......... RobotManager@287.getOnlineRobot(Ljava/lang/String;)LRobot;
1: >-  .......... RobotManager@287.sendSequence(LRobot;Ljava/lang/String;)Ljava/lang/String;
```

thread 1

thread 2

Example 2 - Multithreaded GUI (5)

Ames Research Center

- long trace shows scheduling sequence is involved
- problem specific refined trace analysis:
  "show only overlapping method invocations on same objects"
- reduced trace easily shows missing sync between RobotManagerView.sendSequence() and data acquisition thread

```
listener=.listener.OverlappingMethodAnalyzer
method.include=*Robot*
...
```

```
0:   < . RobotManager.main([Ljava/lang/String;)V                    thread 1
-----------------------------------------
1: >-  ........ RobotManagerView$3@885.actionPerformed(Ljava/awt/event/ActionEvent;)V
1: >-  ......... RobotManagerView@413.sendSequence()V
1: >-  .......... RobotManager@287.getOnlineRobot(Ljava/lang/String;)LRobot;
-----------------------------------------
2: >-< .. RobotManager@287.getListModel()LRobotManager$ListModel;     thread 2
2: >-< .. RobotManager@287.isRobotOnline(Ljava/lang/String;)Z
2: >-  .. RobotManager@287.setRobotOnline(LRobot;Z)V
-----------------------------------------
1:   < .......... RobotManager@287.getOnlineRobot(Ljava/lang/String;)LRobot;
1: >-  .......... RobotManager@287.sendSequence(LRobot;Ljava/lang/String;)Ljava/lang/String;
```

Example 2 - Multithreaded GUI (6)

- how expensive was this refined analyzer?

MethodAnalyzer

functional

```
public class MethodAnalyzer extends ListenerAdapter {

  enum OpType { CALL (">  "),                  // invokeX breaks transition (e.g. blocked sync)
                EXECUTE (" - "),               // method entered method after transition break
                CALL_EXECUTE ("-> "),          // call & enter within same transition
                RETURN (" <"),                 // method returned
                EXEC_RETURN ("-<"),            // execute & return in consecutive ops
                CALL_EXEC_RETURN ("><");       // call & execute & return in consecutive ops
    String code;
    OpType (String code){ this.code = code; }
  }

  static class MethodOp {
    OpType type;

    ThreadInfo ti;
    ElementInfo ei;
    Instruction insn; // the caller
    MethodInfo mi;    // the callee
    int stackDepth;

    // this is used to keep our own trace
    int stateId = Integer.MIN_VALUE;
    MethodOp prevTransition;
    MethodOp p;   // prev during execution

    MethodOp (OpType type, MethodInfo mi, ThreadInfo ti, ElementInfo ei, int stackDepth){
      this.type = type;
      this.ti = ti;
      this.mi = mi;
      this.ei = ei;
      this.stackDepth = stackDepth;
    }

    MethodOp clone (OpType newType){
      MethodOp op = new MethodOp(newType, mi, ti, ei, stackDepth);
      op.p = p;
      return op;
    }

    boolean isMethodEnter() {
      return (type == OpType.CALL_EXECUTE) || (type == OpType.EXECUTE);
    }

    boolean isSameMethod(MethodOp op) {
      return (mi == op.mi) && (ti == op.ti) && (ei == op.ei) && (stackDepth == op.stackDepth);
    }

    void printOn(PrintWriter pw, MethodAnalyzer analyzer) {
      pw.print(ti.getIndex());
      pw.print(": ");

      pw.print(type.code);
      pw.print(' ');

      if (analyzer.showDepth){
        for (int i = 0; i < stackDepth; i++) {
          pw.print('.');
        }
        pw.print(' ');
      }

      if (!mi.isStatic()){
        if (ei.getClassInfo() != mi.getClassInfo()){ // method is in superclass
          pw.print(mi.getClassName());
          pw.print('.');
          pw.print(ei);
          pw.print('>');
        } else { // method is in concrete object class
          pw.print(ei);
        }
      } else {
        pw.print(mi.getClassName());
      }

      pw.print('.');
      pw.print(mi.getUniqueName());
    }

    public String toString() {
      return "Op {" + ti.getName() + ',' + type.code +
                    ',' + mi.getFullName() + ',' + ei + '}';
    }
  }

  // report options

  StringSetMatcher includes = null; //  means all
  StringSetMatcher excludes = null; //  means none

  int maxHistory;
  String format;
  boolean skipInit;
  boolean showDepth;
  boolean showTransition;
  boolean showCompleted;

  // execution environment

  JVM vm;
  Search search;

  OpType opType;

  // this is used to keep our own trace
  MethodOp lastOp;
  MethodOp lastTransition;
  boolean isFirstTransition = true;

  // this is set after we call revertAndFlatten during reporting
  // (we can't call revertAndFlatten twice since it is destructive, but
  // we might have to report several times in case we have several publishers)
  MethodOp firstOp = null;

  // for HeuristicSearches. Ok, that's braindead but at least no need for cloning
  HashMap<Integer,MethodOp> storedTransition = new HashMap<Integer,MethodOp>();


  public MethodAnalyzer (Config config, JPF jpf){
    jpf.addPublisherExtension(ConsolePublisher.class, this);

    maxHistory = config.getInt("method.max_history", Integer.MAX_VALUE);
    format = config.getString("method.format", "raw");
    skipInit = config.getBoolean("method.skip_init", true);
    showDepth = config.getBoolean("method.show_depth", false);
    showTransition = config.getBoolean("method.show_transition", true);

    includes = StringSetMatcher.getNonEmpty(config.getStringArray("method.include"));
    excludes = StringSetMatcher.getNonEmpty(config.getStringArray("method.exclude"));

    vm = jpf.getVM();
    search = jpf.getSearch();
  }


  void addOp (JVM vm, OpType opType, MethodInfo mi, ThreadInfo ti, ElementInfo ei, int stackDepth){
    if (!(skipInit && isFirstTransition)) {
      MethodOp op = new MethodOp(opType, mi, ti, ei, stackDepth);
      if (lastOp == null){
        lastOp = op;
      } else {
        op.p = lastOp;
        lastOp = op;
      }
    }
  }

  boolean isAnalysedMethod (MethodInfo mi){
    String mthName = mi.getFullName();
    return StringSetMatcher.isMatch(mthName, includes, excludes);
  }

  void printOn (PrintWriter pw) {
    MethodOp start = firstOp;
    int lastStateId  = Integer.MIN_VALUE;
    int transition = skipInit ? 1 : 0;
    int lastTid = start.ti.getIndex();

    for (MethodOp op = start; op != null; op = op.p) {

      if (showTransition) {
        if (op.stateId != lastStateId) {
          lastStateId = op.stateId;
          pw.print("---------------------------------------- #");
          pw.println(transition++);
        }
      } else {
        int tid = op.ti.getIndex();
        if (tid != lastTid) {
          lastTid = tid;
          pw.println("----------------------------------------");
        }
      }

      op.printOn(pw, this);
      pw.println();
    }
  }

  // warning - this rotates pointers in situ, i.e. destroys the original structure
  MethodOp revertAndFlatten (MethodOp start) {

    MethodOp last = null;
    MethodOp prevTransition = start.prevTransition;

    for (MethodOp op = start; op != null;) {
      op.p = last;

      if (opp == null) {
        if (prevTransition == null) {
          return op;
        } else {
          last = op;
          op = prevTransition;
          prevTransition = op.prevTransition;
        }
      } else {
        last = op;
        op = opp;
      }
    }

    return null;
  }
```

~200 loc

JPF overhead

```
//--- SearchListener interface
  // <2do> this is the same as DeadlockAnalyzer, except of xnOp type -> refactor

  public void stateAdvanced (Search search){

    if (search.isNewState() && (lastOp != null)) {
      int stateId = search.getStateNumber();

      for (MethodOp op=lastOp; op != null; op=op.p) {
        op.stateId = stateId;
      }

      lastOp.prevTransition = lastTransition;
      lastTransition = lastOp;
    }

    lastOp = null;
    isFirstTransition = false;
  }

  public void stateBacktracked (Search search){
    int stateId = search.getStateNumber();
    while ((lastTransition != null) && (lastTransition.stateId > stateId)){
      lastTransition = lastTransition.prevTransition;
    }
    lastOp = null;
  }

  public void stateStored (Search search) {
    // always called after stateAdvanced
    storedTransition.put(search.getStateNumber(), lastTransition);
  }

  public void stateRestored (Search search) {
    int stateId = search.getStateNumber();
    MethodOp op = storedTransition.get(stateId);
    if (op != null) {
      lastTransition = op;
      storedTransition.remove(stateId);  // not strictly required, but we don't come back
    }
  }

  //--- VMListener interface

  public void instructionExecuted (JVM vm) {
    Instruction insn = vm.getLastInstruction();
    ThreadInfo ti;
    MethodInfo mi;
    ElementInfo ei = null;

    if (insn instanceof InvokeInstruction) {
      InvokeInstruction call = (InvokeInstruction)insn;
      ti = vm.getLastThreadInfo();
      mi = call.getInvokedMethod(ti);

      if (!isAnalysedMethod(mi)) {
        OpType type;

        // check if this was actually executed, or is a blocked sync call
        if (ti.getNextPC() == call) { // re-executed -> blocked or overlayed
          type = OpType.CALL;

        } else { // executed
          if (ti.isFirstStepInsn()) {
            type = OpType.EXECUTE;
          } else {
            type = OpType.CALL_EXECUTE;
          }
        }

        if (call instanceof InstanceInvocation) {
          ei = ((InstanceInvocation)call).getThisElementInfo(ti);
        }

        addOp(vm,type,mi,ti,ei, ti.getStackDepth());
      }

    } else if (insn instanceof ReturnInstruction) {
      ReturnInstruction ret = (ReturnInstruction)insn;
      ti = vm.getLastThreadInfo();
      StackFrame frame = ret.getReturnFrame();
      mi = frame.getMethodInfo();

      if (isAnalysedMethod(mi)) {
        if (!mi.isStatic()) {
          int ref = frame.getThis();
          if (ref != -1) {
            ei = ti.getElementInfo(ref);
          }
        }

        addOp(vm,OpType.RETURN,mi,ti,ei, ti.getStackDepth()+1); // postExec-> frame already popped
      }
    }
  }

  //--- the PubisherExtension part

  public void publishPropertyViolation (Publisher publisher) {

    if (firstOp == null && lastTransition != null){ // do this just once
      firstOp = revertAndFlatten(lastTransition);
    }

    PrintWriter pw = publisher.getOut();
    publisher.publishTopicStart("method ops " + publisher.getLastErrorId());

    printOn(pw);
  }
}
```

~120 loc

OverlappingMethodAnalyzer

(almost) all functional

```
public class OverlappingMethodAnalyzer extends MethodAnalyzer {

  public OverlappingMethodAnalyzer (Config config, JPF jpf){
    super(config,jpf);
  }

  MethodOp getReturnOp (MethodOp op, boolean withinSameThread){
    MethodInfo mi = op.mi;
    int stateId = op.stateId;
    int stackDepth = op.stackDepth;
    ElementInfo ei = op.ei;
    ThreadInfo ti = op.ti;

    for (MethodOp o = op.p; o != null; o = o.p){
      if (withinSameThread && o.ti != ti){
        break;
      }

      if ((o.mi == mi) && (o.ti == ti) && (o.stackDepth == stackDepth) && (o.ei == ei)){
        return o;
      }
    }

    return null;
  }

  // check if there is an open exec from another thread for the same ElementInfo
  boolean isOpenExec (HashMap<ThreadInfo,Deque<MethodOp>> openExecs, MethodOp op){
    ThreadInfo ti = op.ti;
    ElementInfo ei = op.ei;

    for (Map.Entry<ThreadInfo, Deque<MethodOp>> e : openExecs.entrySet()) {
      if (e.getKey() != ti) {
        Deque<MethodOp> s = e.getValue();
        for (Iterator<MethodOp> it = s.descendingIterator(); it.hasNext();) {
          MethodOp o = it.next();
          if (o.ei == ei) {
            return true;
          }
        }
      }
    }

    return false;
  }

  // clean up (if necessary) - both RETURNS and exceptions
  void cleanUpOpenExec (HashMap<ThreadInfo,Deque<MethodOp>> openExecs, MethodOp op){
    ThreadInfo ti = op.ti;
    int stackDepth = op.stackDepth;

    Deque<MethodOp> stack = openExecs.get(ti);
    if (stack != null && !stack.isEmpty()) {
      for (MethodOp o = stack.peek(); o != null && o.stackDepth >= stackDepth; o = stack.peek()) {
        stack.pop();
      }
    }
  }

  void addOpenExec (HashMap<ThreadInfo,Deque<MethodOp>> openExecs, MethodOp op){
    ThreadInfo ti = op.ti;
    Deque<MethodOp> stack = openExecs.get(ti);

    if (stack == null){
      stack = new ArrayDeque<MethodOp>();
      stack.push(op);
      openExecs.put(ti, stack);

    } else {
      stack.push(op);
    }
  }

  void printOn (PrintWriter pw) {
    MethodOp start = firstOp;

    HashMap<ThreadInfo,Deque<MethodOp>> openExecs = new HashMap<ThreadInfo,Deque<MethodOp>>();

    int lastStateId  = Integer.MIN_VALUE;
    int transition = skipInit ? 1 : 0;
    int lastTid = start.ti.getIndex();

    for (MethodOp op = start; op != null; op = op.p) {

      if (showTransition) {
        if (op.stateId != lastStateId) {
          lastStateId = op.stateId;
          pw.print("---------------------------------------- #");
          pw.println(transition++);
        }
      } else {
        int tid = op.ti.getIndex();
        if (tid != lastTid) {
          lastTid = tid;
          pw.println("----------------------------------------");
        }
      }

      cleanUpOpenExec(openExecs, op);

      if (op.isMethodEnter()) {  // EXEC or CALL_EXEC
        MethodOp retOp = getReturnOp(op, true);
        if (retOp != null) { // completed, skip
          if (!isOpenExec(openExecs, op)) {
            op = retOp;
            continue;
          }
        } else { // this is an open method exec, record it
          addOpenExec(openExecs, op);
        }
      }

      op = consolidateOp(op);

      op.printOn(pw, this);
      pw.println();
    }
  }

  MethodOp consolidateOp (MethodOp op){
    for (MethodOp o = op.p; o != null; o = o.p){
      if (showTransition && (o.stateId != op.stateId)){
        break;
      }
      if (o.isSameMethod(op)){
        switch (o.type) {
          case RETURN:
            switch (op.type){
              case CALL_EXECUTE:
                op = o.clone(OpType.CALL_EXEC_RETURN); break;
              case EXECUTE:
                op = o.clone(OpType.EXEC_RETURN); break;
            }
            break;
          case EXEC_RETURN:
            switch (op.type){
              case CALL:
                op = o.clone(OpType.CALL_EXEC_RETURN); break;
            }
            break;
          case CALL_EXECUTE:  // simple loop
            switch (op.type){
              case CALL_EXEC_RETURN:
                op = o;
            }
            break;

        } else {
          break;
        }
      }
    }

    return op;
  }
}
```

~150 loc

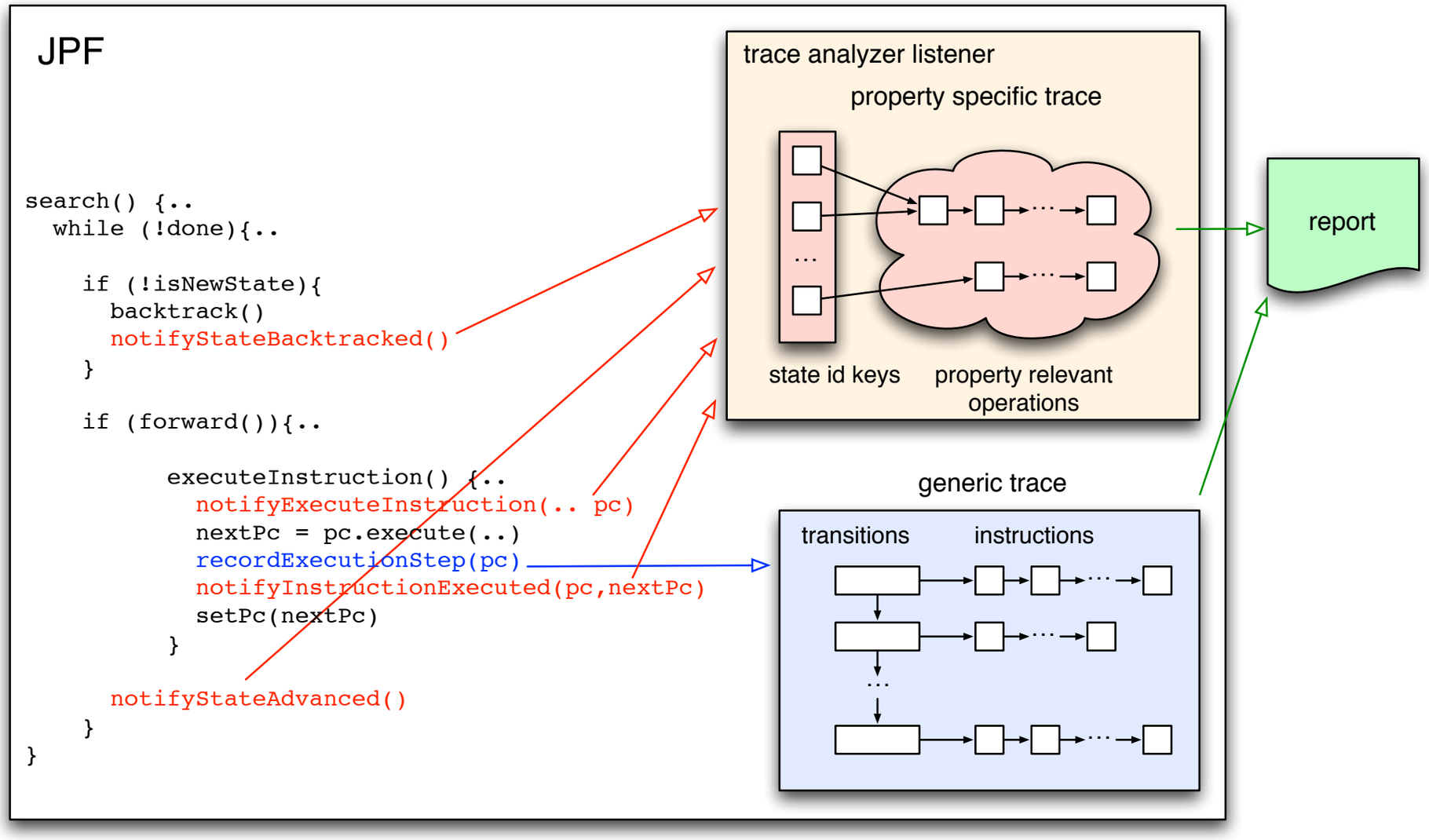
Outlook - Lessons from Example

Ames Research Center

- most explanations are within cross section between
  - non-functional property violations (e.g. defect manifestation through NullPointerException)
  - application/domain specific program structure
- scalability not just a challenge for *finding* but esp. for *explaining* bugs (trace length)
- lesson is *not* "develop the silver bullet listener"
- where possible, trace analysis should not require rerunning JPF
- key is how easy it is to come up with effective domain/property specific analyzers ⇒ agile analysis

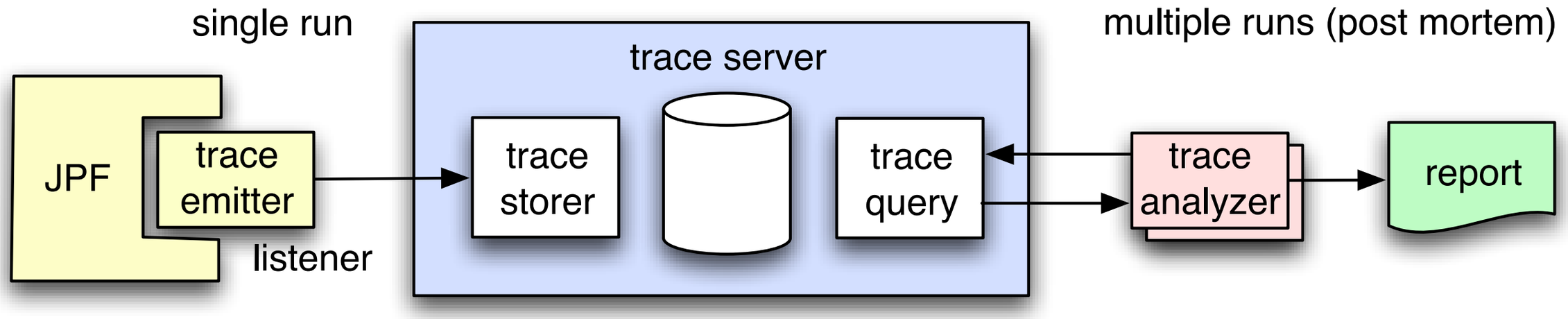
Outlook - Current Deficiencies

- domain/property specific trace generation done by listener outside of generic (instruction) trace generation
- all traces kept in memory, competing with SUT heap and state storage
- does not differentiate between trace generation and analysis/reporting
- listeners have to handle backtracking / state restoration



Outlook - The Trace Server

- where possible, separate property checks, trace storage and post mortem trace analysis (mostly for safety properties)
- keep trace storage outside of JPF (search process memory savings)
- key aspects
  - performance: one-way, no-sync communication JPF → Trace Server
  - usability: simple trace query interface



- execution engine
- property checks
- state matching, storage, backtracking

- transition caching
- trace storage
- query interface (enumeration)

- post mortem trace analysis
- report generation

Outlook - Trace Server Conclusion

- will be implemented as part of our participation in Google Summer of Code 2010 Program (funded by Google)
- directly applicable to ongoing NASA project
- shows how to leverage open source for government

Thank You