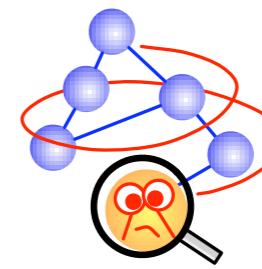
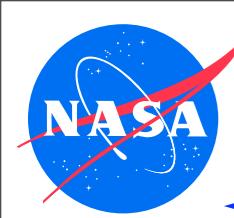


JavaPathfinder Tutorial

02/2010



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



Objectives and Roadmap



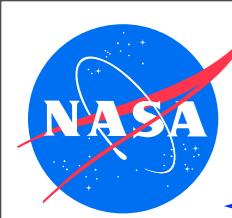
◆ Objectives

- get basic understanding of what JPF is
- learn by example what (current) JPF can be used for
- learn what it takes to install, configure and run JPF
- learn how JPF can be extended

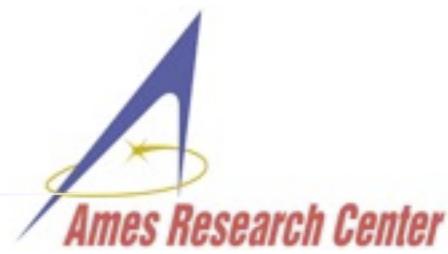
learn
↓
use
↓
extend

◆ Roadmap

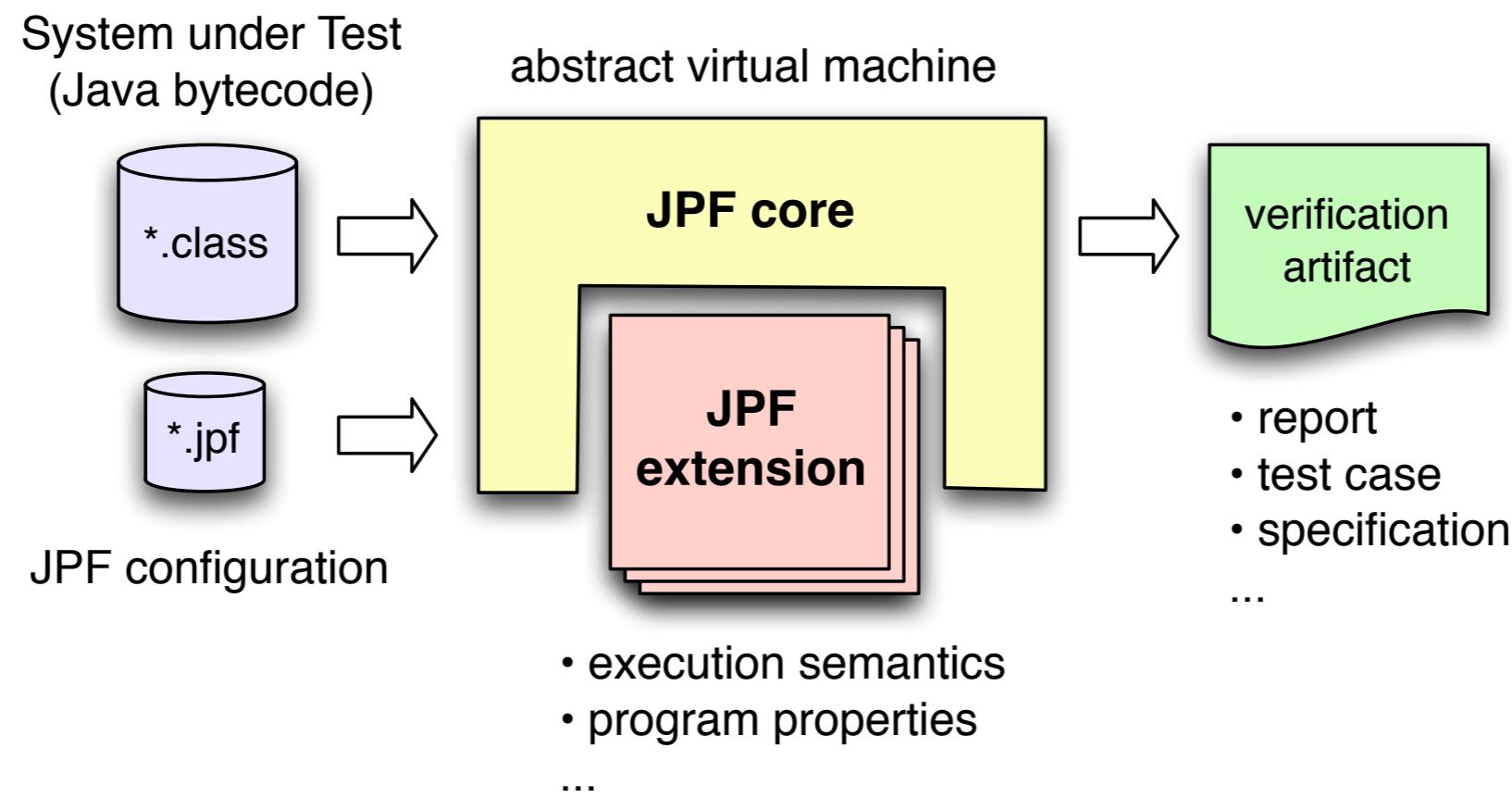
| | |
|--|------|
| (1) Introduction | 1.0h |
| (2) Application Examples | 2.0h |
| (3) Installation, Configuration and Invocation | 1.0h |
| (4) Extension Mechanisms | 1.5h |
| (5) Discussion | |



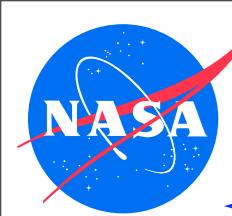
Introduction - What Is JPF?



- ◆ 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)



Example - What has JPF been used for ?

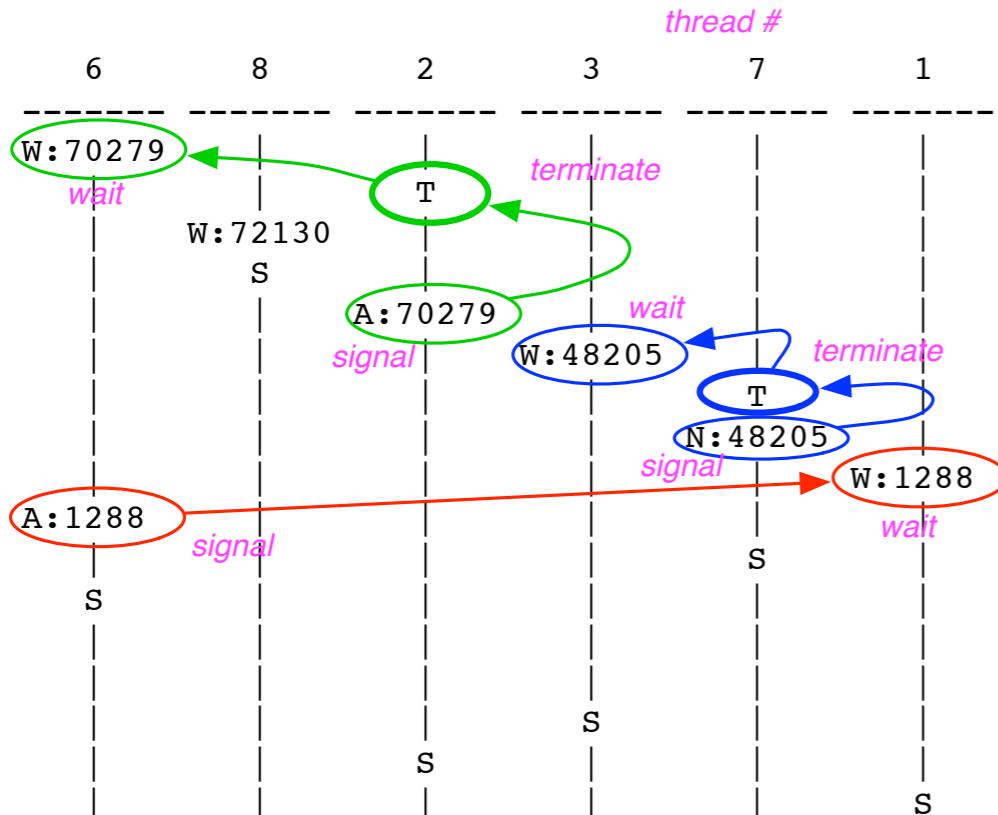


Objective: reproduce - and analyze - spurious deadlock in large web application

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

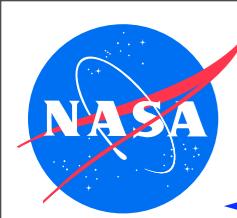
| trans | insn | loc |
|--------|---|-----|
| 123009 | invokevirt awtMutex.java:38 : try {wait();} | |
| 123008 | return MessageAnalyzer.java:341 : | |
| 122597 | invokevirt MessageAnalyzer.java:96 : wai | |
| 122593 | | |
| 122572 | invokevirt awtMutex.java:48 : notifyAll(); | |
| 122541 | invokevirt 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 | | |
| 29 | invokevirt awtQueue.java:182 : wait(); | |
| 9 | | |
| 7 | | |
| 2 | | |
| 0 | | |

...

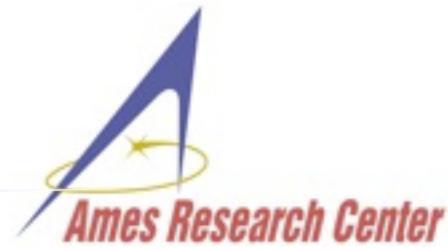
```
===== statistics
```

```
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
```

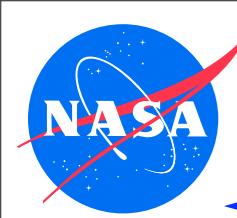
just one success story
for the type of deep
defects targeted by JPF



Who is using JPF ?



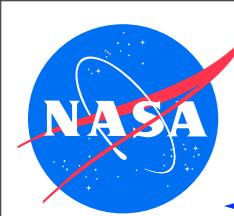
- ◆ major user group is academic research - collaborations with >20 universities worldwide (uiuc.edu, unl.edu, byu.edu, umn.edu, Stellenbosch Za, Waterloo Ca, AIST Jp, Charles University Prague Cz, ..)
- ◆ companies not so outspoken (exception Fujitsu - see press releases, e.g. <http://www.fujitsu.com/global/news/pr/archives/month/2010/20100112-02.html>) , 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



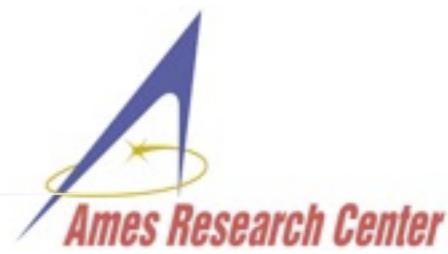
History



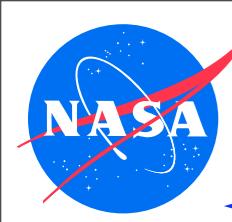
- ◆ not a new project: around since 10 years and continuously developed:
 - 1999 - project started as front end for Spin model checker
 - 2000 - reimplemented as concrete virtual machine for software model checking (concurrency defects)
 - 2003 - introduction of extension interfaces
 - 2005 - open sourced on Sourceforge
 - 2008 - participation in Google Summer of Code
 - 2009 - moved to own server, hosting extension projects and Wiki



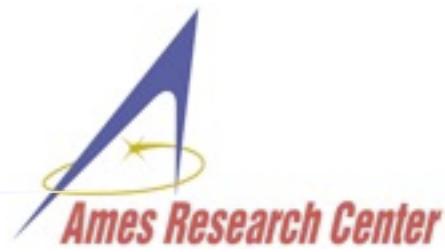
Awards



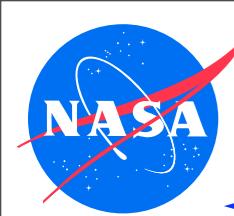
- ◆ widely recognized, awards for JPF in general and for related work, team and individuals
 - 2002 - ACM Sigsoft Distinguished Paper award
 - 2003 - “Turning Goals into Reality” (TGIR) Engineering Innovation Award from the Office of AeroSpace Technology
 - 2004, 2005 - Ames Contractor Council Awards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



Caveat emptor - Costs of JPF application



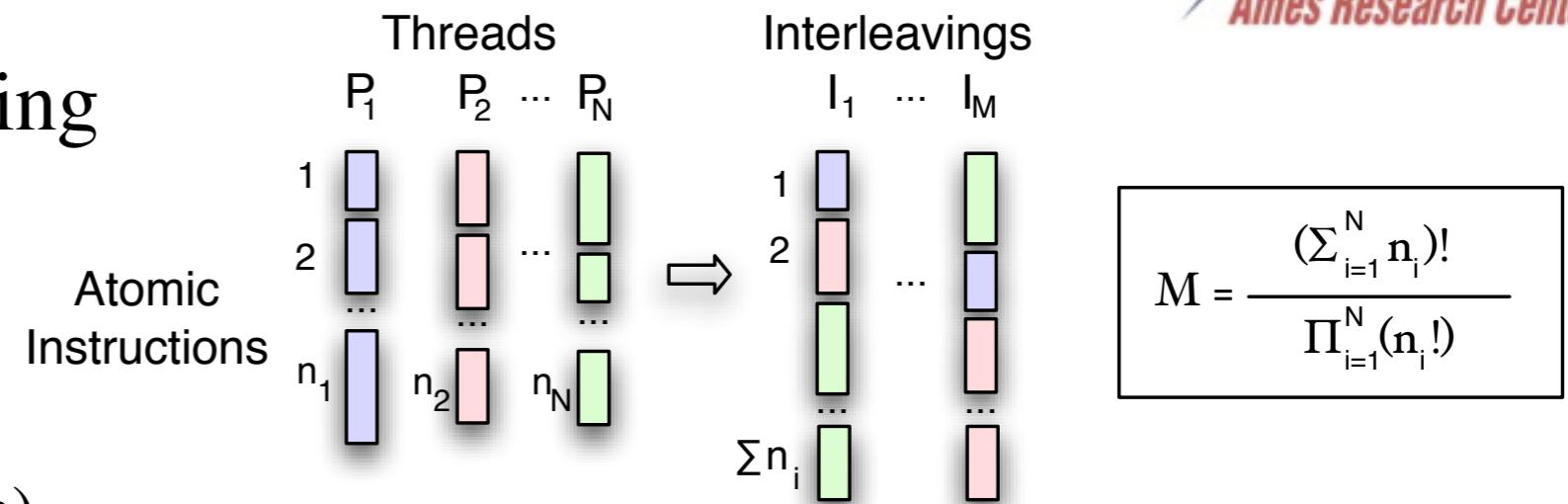
- ◆ you need to learn
 - JPF is not a lightweight tool
 - flexibility has its price - configuration can be intimidating
 - might require extension for your SUT (properties, libraries)
- ◆ you will encounter unimplemented/missing parts (e.g. UnsatisfiedLinkError)
 - usually easy to implement
 - exception: state-relevant native libraries (java.io, java.net)
 - can be either modeled or stubbed
- ◆ you need suitable test drivers for concrete inspection & SMC



JPF's Challenges

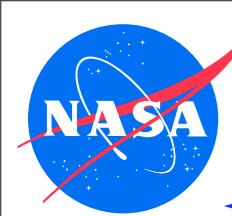


- ◆ **theoretical:** model checking scalability
⇒ optimization, adaptation (config), research (collaboration)



- ◆ **technical:**
 - Java VM + libraries HUGE a lot of ground to cover
⇒ reuse (libs, where possible)
 - JPF is research platform *and* production tool at the same time
⇒ open sourcing (but no free lunch)
- ◆ **organizational:**
 - small inhouse developer team
⇒ relies on external collaboration

| Java | files | loc | MB |
|---------|--------|----------|-----|
| OpenJDK | 12,000 | 3.10E+06 | 118 |
| JPF | 1,600 | 2.04E+05 | 6 |



Where to learn more - the JPF-Wiki



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

The screenshot shows the JPF Wiki homepage with several features highlighted:

- Public access:** A red circle points to the URL in the browser's address bar: <http://babelfish.arc.nasa.gov/trac/jpf/wiki>.
- Ticketing:** A red circle highlights the "View Tickets" and "New Ticket" buttons in the top navigation bar.
- Bug tracking:** A red circle points to the "Trac ticket system" link in the sidebar.
- Project blog:** A red circle points to the "Blog" button in the top navigation bar.
- Hierarchical menu:** A red circle highlights the "JPFWiki – Welcome Page" sidebar menu, which includes links like Introduction, Installing JPF, User Guide, Developer Guide, Projects, Change(B)log, About, Papers, FAQ, Playground, and Table of Context.
- Latest news:** A red circle points to the "Latest JPF News" section, which lists recent events with dates and links.

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

bug tracking

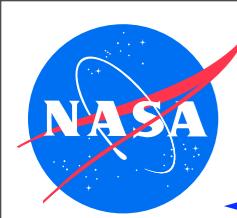
- Trac ticket system

project blog

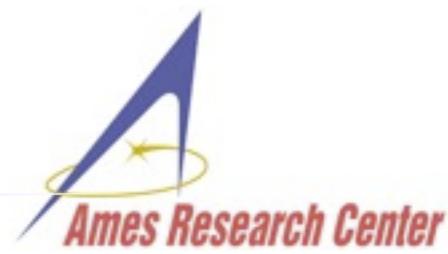
- announcements
- important changes

hierarchical
navigation menu

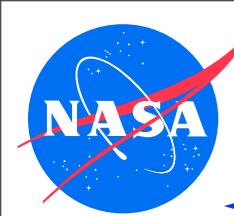
- intro
- installation
- user docu
- developer docu
- extension projects



Introduction: Key Points



- ◆ JPF is research platform *and* production tool (basis)
- ◆ JPF is designed for extensibility
- ◆ JPF is open source
- ◆ JPF is an ongoing collaborative development project
- ◆ JPF cannot find all bugs
 - but as of today -
some of the most expensive bugs only JPF can find
- ◆ JPF is moderately sized system (~200ksloc core + extensions)
- ◆ JPF represents >20 man year development effort
- ◆ JPF is pure Java application (platform independent)



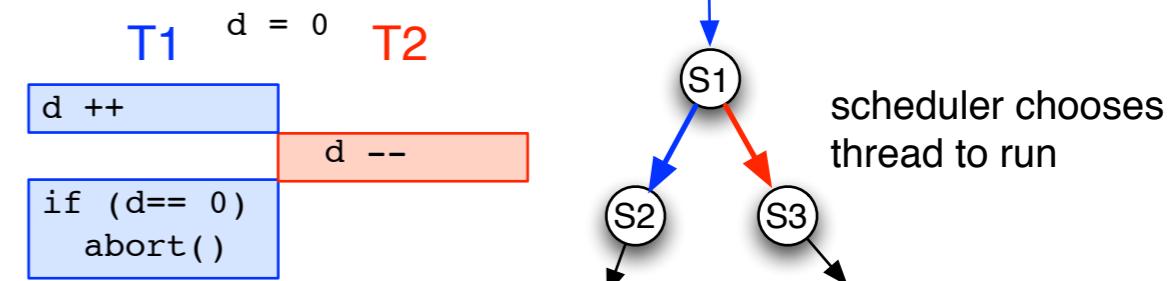
Examples



goal is to show gamut of possible applications, not detailed defect understanding

- ◆ software model checking (SMC) of production code

- data acquisition (random, user input)
 - concurrency (deadlock, races)



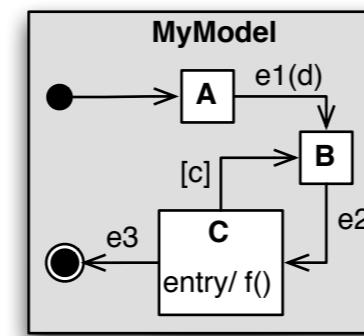
- ◆ deep inspection of production code

- property annotations (Const, PbC,..)
 - numeric verification (overflow, cancellation)

```
@Const  
int dontChangeMe() {...}  
double x = (y - z) * c  
numeric error of x?
```

- ◆ model verification

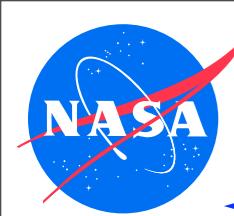
- UML statecharts



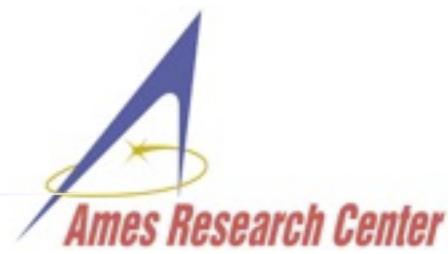
does model work?

- ◆ test case generation

- ◆ verification of distributed application



Application Types



constraints

*JPF unaware
programs*

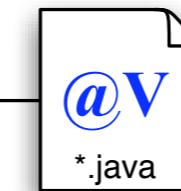
runs on any
JVM



- runtime costs**
 - order of magnitude slower
 - state storage memory
- standard library support**
 - java.net, javax.swing, ..
(needs abstraction models)
- functional property impl. costs**
 - listeners, MJI knowledge
- restricted choice types**
 - scheduling sequences
 - java.util.Random

*JPF enabled
programs*

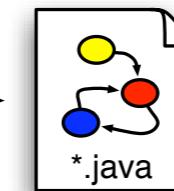
"sweet spot"



- annotate program**
 - requirements
 - sequences (UML)
 - contracts (PbC)
 - tests
 - ...
- analyze program**
 - symbolic exec
 - test data
 - thread safety / races

*JPF dependent
programs*

runs only
under JPF

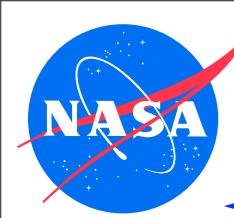


- restricted application models**
 - UML statemachines
 - does not run w/o JPF libraries
- initial domain impl. costs**
 - domain libs can be tricky

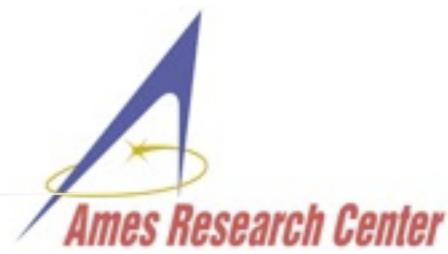
benefits

- non-functional properties**
 - unhandled exceptions
(incl. AssertionError)
 - deadlocks
 - races
- improved inspection**
 - coverage statistics
 - exact object counts
 - execution costs

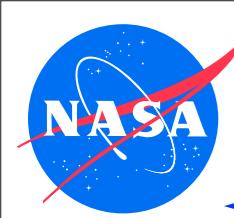
- low modeling costs**
 - statemachine w/o layout hassle,...
- functional (domain) properties**
 - built-in into JPF libraries
- flexible state space**
 - domain specific choices
(e.g. UML "enabling events")
- runtime costs & library support**
 - usually not a problem, domain
libs can control state space



Examples: Software Model Checking



- ◆ focus is on *production code*, i.e. SUT applications are not specifically written for JPF
- ◆ might include abstracted/modeled libraries, but no modified SUT behavior
- ◆ objective: check execution paths that are hard or impossible to test reliably (scheduling sequences, large input spaces, mix between both)
- ◆ classic application: detect concurrency defects



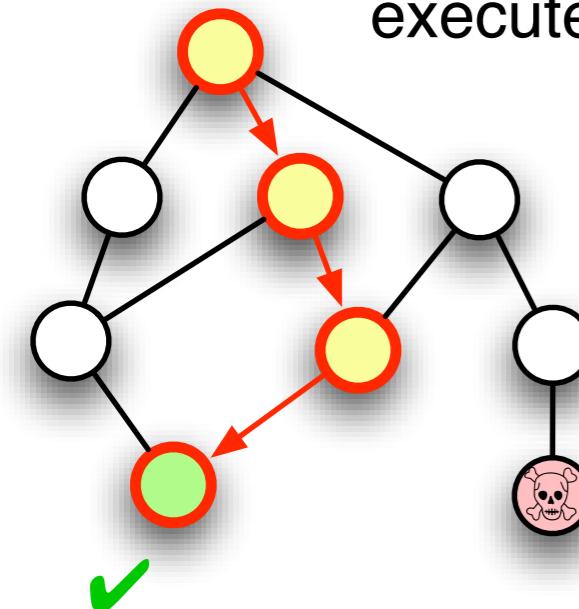
Model Checking vs. Testing



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

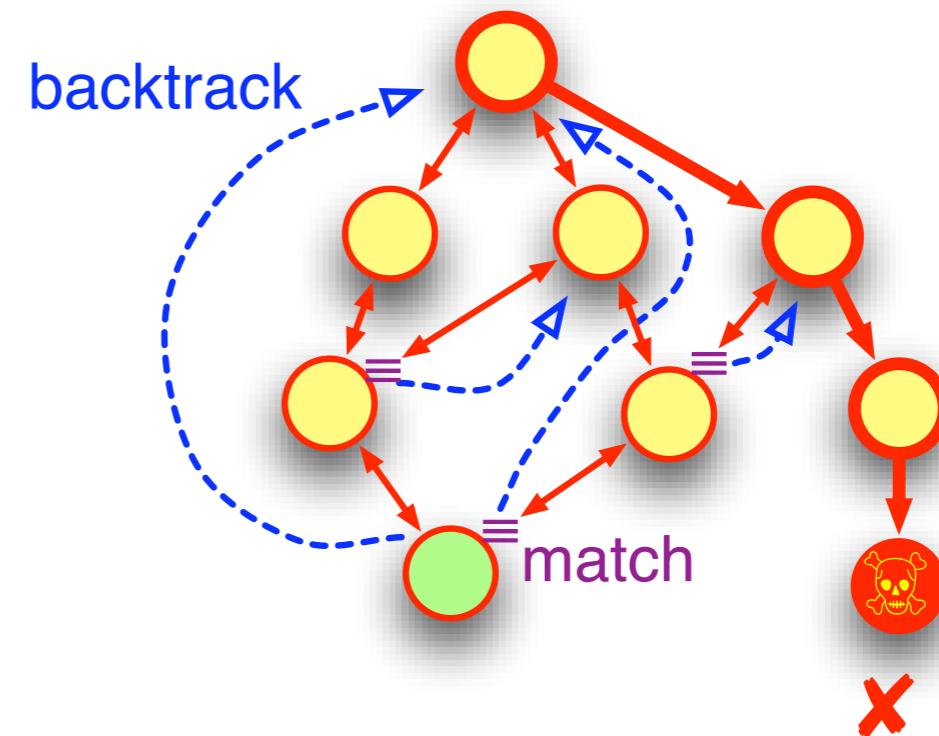
testing:

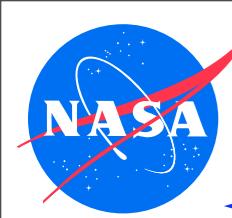
based on input set $\{d\}$
only one **path**
executed at a time



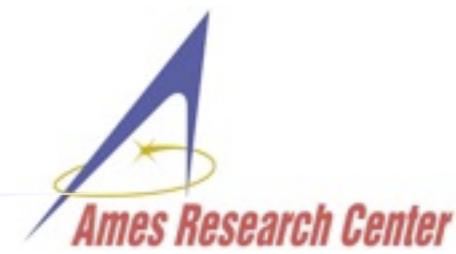
model checking:

all program states are explored
until none left or defect found





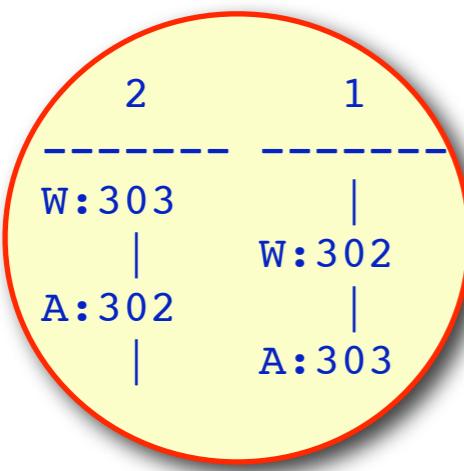
Model Checking vs. Debugging

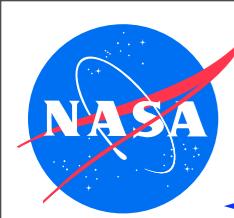


- ◆ JPF automatically *finds* defects to debug
- ◆ program traces - JPF *remembers* whole execution history that lead to defect
- ◆ automatic trace analysis capability *explains* defects

```
deadlock encountered:  
  thread index=1,name=Thread-0,status=WAITING,...  
  thread index=2,name=Thread-1,status=WAITING,...  
  ...  
----- transition #32 thread: 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  
  oldclassic.java:103  : event1.wait_for_event();  
  oldclassic.java:79   : wait();  
  ...  
  
trans  loc  
-----  
35  oldclassic.java:79  : wait();  
34  oldclassic.java:79  : wait();  
32  oldclassic.java:74  : notifyAll();  
28  oldclassic.java:74  : notifyAll();
```

Trace Analysis





Example 1: Nondeterministic Data (1)



◆ Source

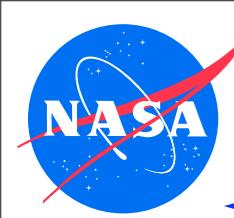
```
..    public static void main (String[ ] args) {  
..        ...  
7:        Random random = new Random(42);           // (1)  
8:        int a = random.nextInt(2);                // (2)  
..        ...  
13:        int b = random.nextInt(3);                // (3)  
..        ...  
16:        int c = a/(b+a -2);                     // (4)  
..        ...  
..    }
```

◆ Config

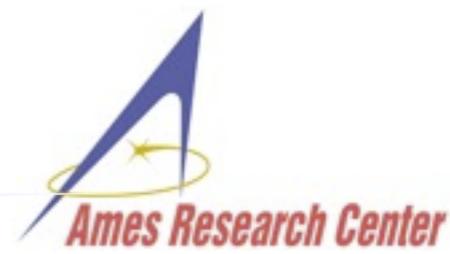
```
cg.enumerate_random = true
```

◆ Output

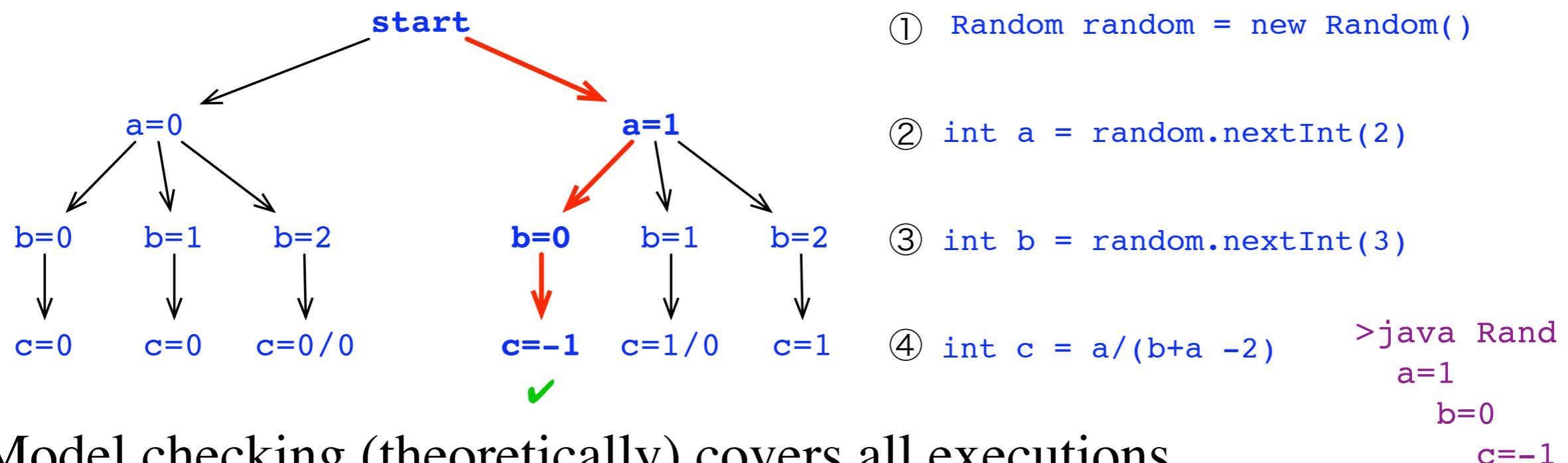
```
...  
===== search started: 2/10/10 6:08 PM  
computing c = a/(b+a - 2)..  
a=0  
    b=0      ,a=0  
=>  c=0      ,a=0,b=0  
    b=1      ,a=0  
=>  c=0      ,a=0,b=1  
    b=2      ,a=0  
  
===== error #1  
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty  
java.lang.ArithmetricException: division by zero  
at Rand.main(Rand.java:16)
```



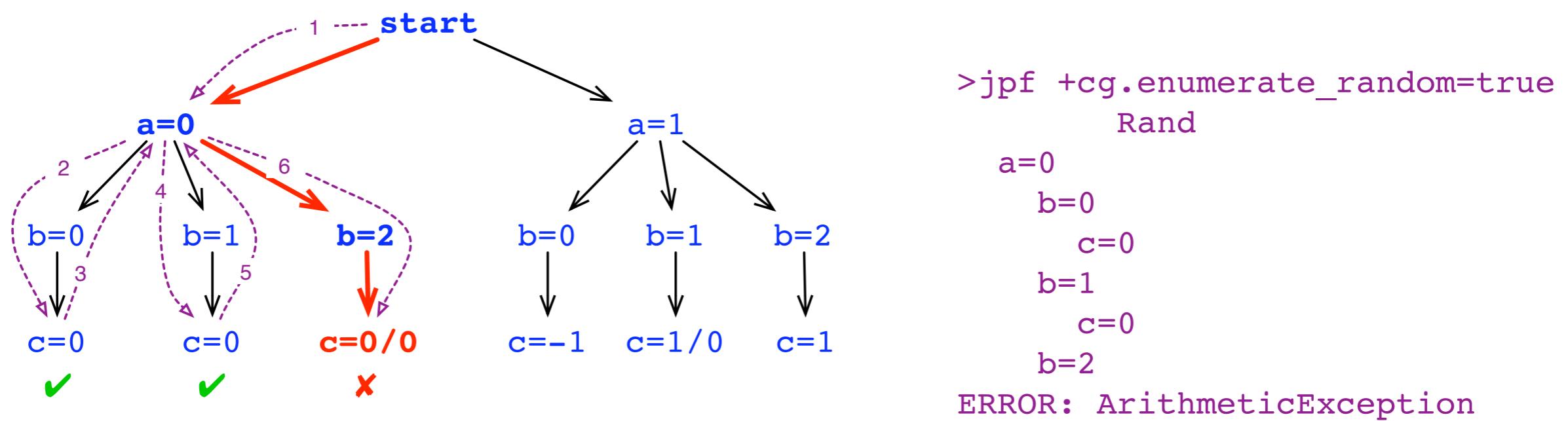
Example 1: Nondeterministic Data (2)

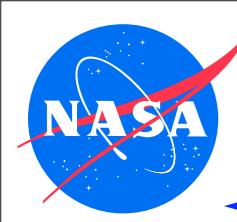


- ◆ Testing only covers one execution

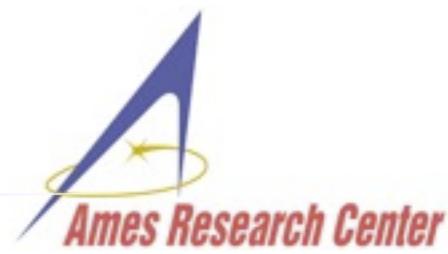


- ◆ Model checking (theoretically) covers all executions





Example 2: Data Race (1)



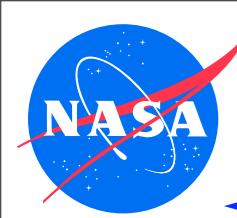
◆ Source

```
int d = 42;

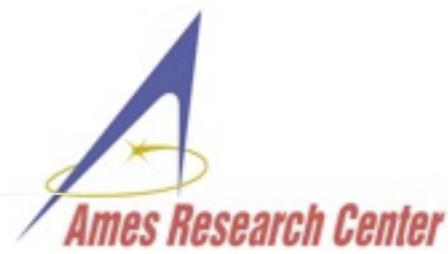
public void run () {
    doSomething(1001);                                // (1)
    d = 0;                                            // (2)
}

public static void main (String[ ] args){
    Racer racer = new Racer();
    Thread t = new Thread(racer);
    t.start();

    doSomething(1000);                                // (3)
    int c = 420 / racer.d;                            // (4)
    System.out.println(c);
}
```



Example 2: Data Race (1)



◆ Source

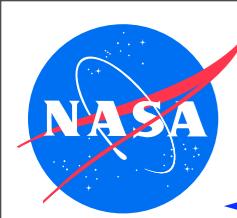
```
int d = 42;

public void run () {
    doSomething(1001); // (1)
    d = 0; // (2)
}

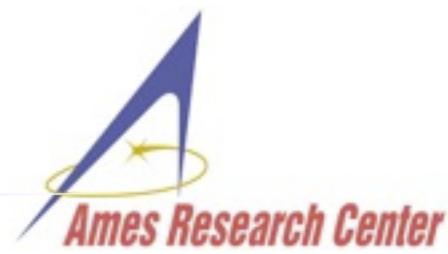
public static void main (String[ ] args){
    Racer racer = new Racer();
    Thread t = new Thread(racer);
    t.start();

    doSomething(1000); // (3)
    int c = 420 / racer.d; // (4)
    System.out.println(c);
}
```

A red oval highlights the assignment statement `d = 0;` in the `run()` method. A red arrow points from this oval to a red callout bubble containing the text "data race". Another red oval highlights the division operation `420 / racer.d;` in the `main()` method, with a red arrow pointing from it to the same "data race" callout bubble.



Example 2: Data Race (2)

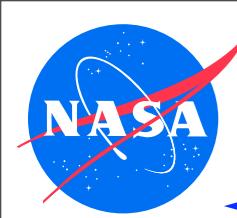


◆ Config

```
listener = gov.nasa.jpf.listener.PreciseRaceDetector  
report.console.propertyViolation = error,trace
```

◆ Output

```
...  
===== search started: 2/10/10 6:32 PM  
10  
10  
===== error #1  
gov.nasa.jpf.listener.PreciseRaceDetector  
race for field Racer@d  
    main at Racer.main(Racer.java:16)  
        "int c = 420 / racer.d;"                      " : getfield  
    Thread-0 at Racer.run(Racer.java:7)  
        "d = 0;"                                         " : putfield  
===== trace #1  
----- transition #0 thread: 0  
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}  
    [2843 insn w/o sources]  
...  
    Racer.java:16                                     : int c = 420 / racer.d;  
----- transition #4 thread: 1  
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0}  
...  
    Racer.java:7                                     : d = 0;  
----- transition #5 thread: 0  
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0}  
    Racer.java:16                                     : int c = 420 / racer.d;
```



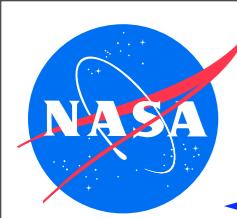
Example 3: Deadlock (1)



```
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 Event {  
    int count = 0;  
  
    public synchronized void signal_event () {  
        count++; notifyAll();  
    }  
    public synchronized void wait_for_event () {  
        .. wait(); ..  
    }  
}
```

```
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;  
        }  
    }  
}
```



Example 3: Deadlock (2)



- ◆ Report - shows what happened, but not easily why

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

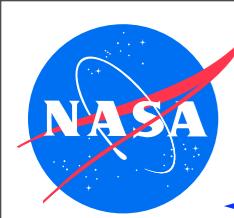
```
===== system under test
application: oldclassic.java
```

```
===== 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 3: Deadlock (3)



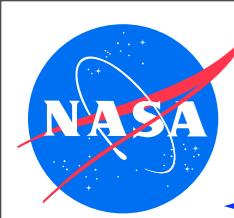
```
...
===== trace #1
----- transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
[2843 insn w/o sources]
oldclassic.java:47          : Event      new_event1 = new Event();

...
----- 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;

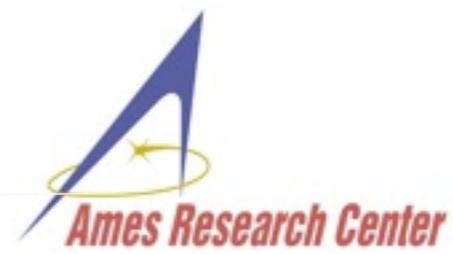
...
oldclassic.java:126         : System.out.println(" 2");
oldclassic.java:127         : event1.signal_event();

...
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();
```



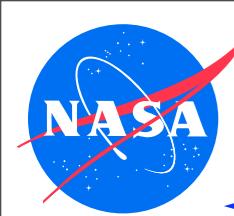
Example 3: Deadlock (3)



```
...
===== 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(); missed signal
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(); missed signal
----- 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();
```



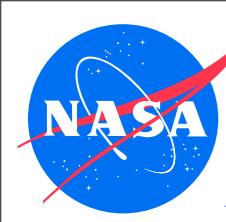
Example 3: Deadlock (4)



```
report.console.propertyViolation = error, snapshot
listener = .listener.DeadlockAnalyzer
===== error #1
gov.nasa.jpf.jvm.NotDeadlockedProperty
deadlock encountered:
...
===== snapshot #1
thread index=1, name=Thread-0, status=WAITING, this=FirstTask@295 ...
  waiting on: Event@290
  call stack:
    at Event.wait_for_event(oldclassic.java:79)
    at FirstTask.run(oldclassic.java:103)

thread index=2, name=Thread-1, status=WAITING, this=SecondTask@322 ...
  waiting on: Event@291
  call stack:
    at Event.wait_for_event(oldclassic.java:79)
    at SecondTask.run(oldclassic.java:130)

===== 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 4: User Input Model Checking (1)



- specify user input in event script with alternatives

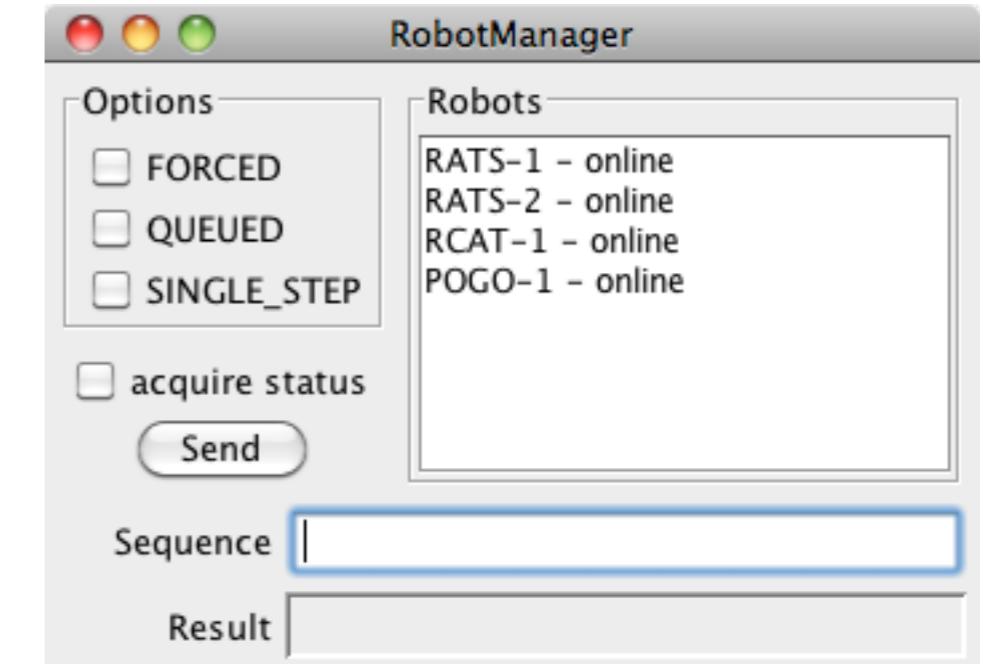
```
// enter command sequence  
$Sequence:input.setText("turn")  
  
// try all Options combinations  
ANY { $<FORCED|QUEUED|SINGLE_STEP>.doClick() }  
ANY { $<FORCED|QUEUED|SINGLE_STEP>.doClick() }  
ANY { $<FORCED|QUEUED|SINGLE_STEP>.doClick() }
```

```
// select robot  
ANY { $Robots:list.setSelectedIndex(0|1|2|3) }
```

```
// send sequence  
$Send.doClick()
```

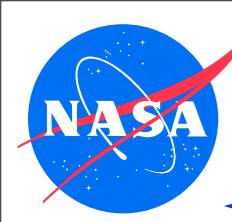
```
input.setText(...)  
FORCED.doClick()  
FORCED.doClick()  
FORCED.doClick()  
list.setSelectedIndex(0)  
Send.doClick()
```

```
input.setText(...)  
QUEUED.doClick()  
...
```



```
...  
input.setText(...)  
SINGLE_STEP.doClick()  
SINGLE_STEP.doClick()  
SINGLE_STEP.doClick()  
list.setSelectedIndex(3)  
Send.doClick()
```

generated event sequences



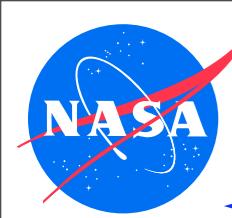
Example 4: User Input Model Checking (2)



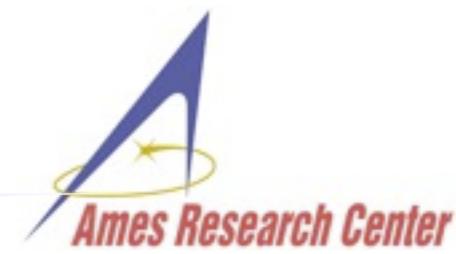
- ◆ Finds input combination that causes exception in user code:

```
===== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.AssertionError: POGOs cannot force queued sequences
    at POGO.processSequence(RobotManager.java:92)
    at RobotManager.sendSequence(RobotManager.java:264)
    ...
    ...

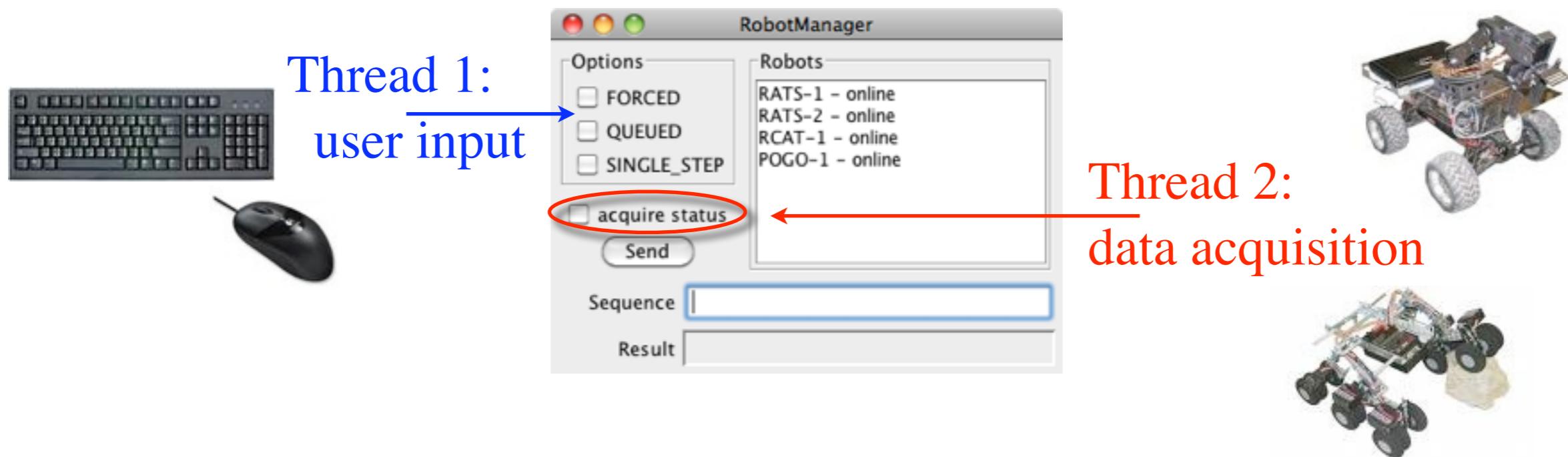
===== choice trace #1
0: gov.nasa.jpf.awt.UIActionSingleChoice[... >$Sequence:input.setText("turn")]
1: gov.nasa.jpf.awt.UIActionFromSet[... >$FORCED.doClick(),
    $QUEUED.doClick(),
    $SINGLE_STEP.doClick()]
2: gov.nasa.jpf.awt.UIActionFromSet[... >$FORCED.doClick(),
    >$QUEUED.doClick(),
    $SINGLE_STEP.doClick()]
3: gov.nasa.jpf.awt.UIActionFromSet[... >$FORCED.doClick(),
    $QUEUED.doClick(),
    >$SINGLE_STEP.doClick()]
4: gov.nasa.jpf.awt.UIActionFromSet[... $Robots:list.setSelectedIndex(0),
    $Robots:list.setSelectedIndex(1),
    $Robots:list.setSelectedIndex(2),
    >$Robots:list.setSelectedIndex(3)]
5: gov.nasa.jpf.awt.UIActionSingleChoice[... >$Send.doClick()]
```

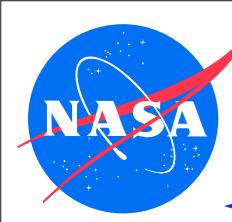


Example 5: Multithreaded GUI (1)



- ◆ 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





Example 5: Multithreaded GUI (2)



- ◆ JPF finds defect that is untestable, but..

general,
low level
defect

The screenshot shows the JavaPathfinder UI Shell interface. The title bar says "UIShell". Below it is a toolbar with icons for "Properties" (green checkmark), "Report" (green arrow), "Test Output" (selected), "Verify Output", "Components", "Trace", and "Script". The main window displays the following text:

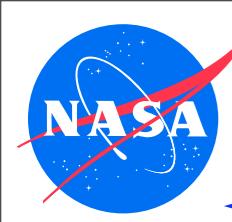
```
JavaPathfinder v5.x - (C) RIACS/NASA Ames Research Center

=====
system under test
application: RobotManager.java
=====
search started: 2/17/10 10:03 AM

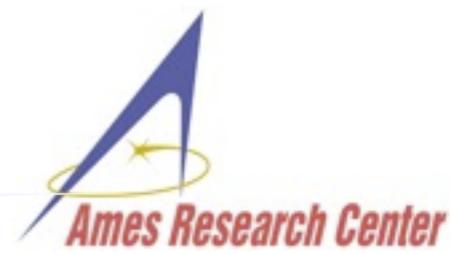
=====
error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.NullPointerException: calling 'processSequence(Ljava/lang/String;)Ljava/lang/String;' o
    at RobotManager.sendSequence(RobotManager.java:266)
    at RobotManagerView.sendSequence(RobotManager.java:538)
    at RobotManagerView$3.actionPerformed(RobotManager.java:339)
    at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:113)
    at javax.swing.AbstractButton$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 5: Multithreaded GUI (3)



- ◆ trace too long to find out what causes it - needs more analysis

UIShell

V ➤

Properties Report Test Output Verify Output Components Trace Script

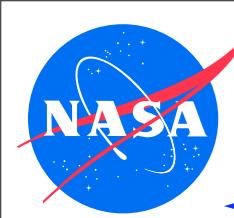
```
RobotManager.java:257      : return onlineRobots.get(robotName);
RobotManager.java:537      : Robot robot = model.getOnlineRobot(selectedRobotName);
RobotManager.java:538      : String result = model.sendSequence(robot, sequence);
----- transition #169 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>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
```

too deep to understand



Example 5: Multithreaded GUI (4)



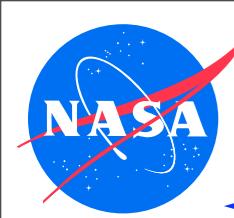
- generic trace analysis can help (e.g. hierarchical method execute/return log with MethodAnalyzer listener)
- still too time consuming in many cases

```
listener=.listener.MethodAnalyzer
method.include=*Robot*
...
...
1: .....R RobotManager.isRobotOnline(..)Z, RobotManager@287
-----
...
2: ..X RobotManager.isRobotOnline(..)Z, RobotManager@287
-----
1: .....X RobotManager.getOnlineRobot(..)LRobot;, RobotManager@287
-----
...
2: ..R RobotManager.isRobotOnline(..)Z, RobotManager@287
2: ..X RobotManager.setRobotOnline(LRobot;Z)V, RobotManager@287
...
-----
1: .....R RobotManager.getOnlineRobot(..)LRobot;, RobotManager@287
-----
...
2: ..R RobotManager.setRobotOnline(LRobot;Z)V, RobotManager@287
-----
1: .....X RobotManager.sendSequence(LRobot;..), RobotManager@287
```

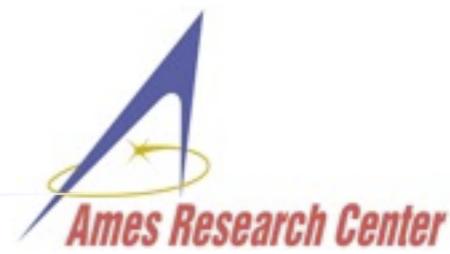


thread 1

thread 2



Example 5: Multithreaded GUI (5)



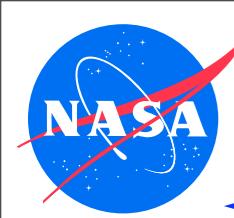
- ◆ better approach - precise properties from annotations (user docu)
- ◆ self explaining error message (no need for further analysis)
- ◆ much more efficient to check

```
// means: RobotManager instances are not thread-safe, don't use them concurrently
@NonShared
public class RobotManager {
    ...
}

=====
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.AssertionError: NonShared object: RobotManager@287
    accessed in live thread cycle: AWT-EventQueue-0,Thread-1,AWT-EventQueue-0,main
        at RobotManager$ListModel.getSize(RobotManager.java:170)
    ...
===== statistics
elapsed time:          0:00:01
states:                 new=76, visited=0, backtracked=0, end=0
search:                 maxDepth=75, constraints=0
choice generators:      thread=67, data=9
heap:                   gc=94, new=2668, free=212
instructions:          72884
...
```



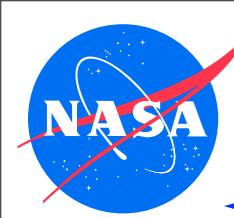
self-explaining,
doesn't even need
trace



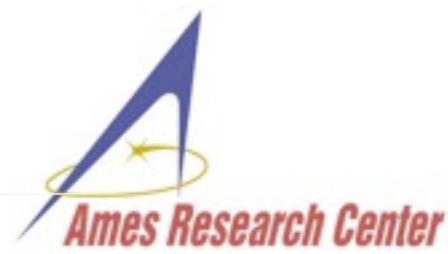
Examples: Deep Inspection



- ◆ focus is primarily on properties that require or benefit from VM level evaluation (execution history and details)
- ◆ checks do not modify SUT behavior (performed at VM level)
- ◆ orthogonal to path exploration (makes model checking more useful)
- ◆ can be based on “hints” (annotations) in SUT (JPF-aware application)
- ◆ can be just based on binary code (JPF unaware application)



Example 6: Const Annotation



- ◆ transitive properties (holds for certain dynamic scope)
- ◆ checked by dedicated listeners (runtime monitoring)
- ◆ does not require extensive annotation (const/var/unknown) of static analysis tools to avoid false positives

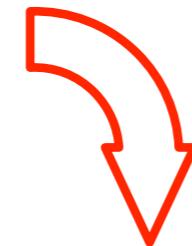
```
import gov.nasa.jpf.annotation.Const;

public class ConstViolation {
    int d;

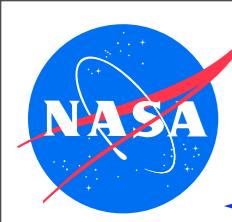
    @Const
    public void dontDoThis() {
        foo();
    }

    void foo () {
        d = 42;
    }
    ...
===== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.AssertionError: instance field write within const context: int ConstViolation.d
    at ConstViolation.foo(ConstViolation.java:15)
    at ConstViolation.dontDoThis(ConstViolation.java:11)
    ...

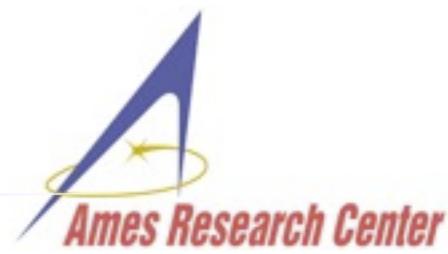
```



```
listenerautoload=gov.nasa.jpf.annotation.Const, ...
listener.gov.nasa.jpf.annotation.Const=
    .aprop.listener.ConstChecker
```



Example 7: SandBox Annotation



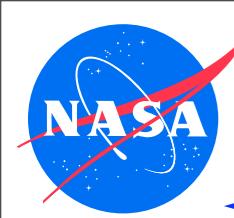
- ◆ more sophisticated, instance aware scope property
- ◆ instances only allowed to write “owned” fields (incl. fields of objects stored in reference fields)
- ◆ mechanism suitable for security related properties

listener=.aprop.listener.ConstChecker

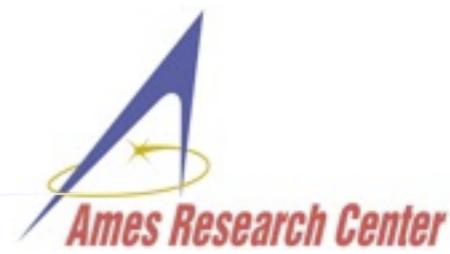


```
class X {  
    int d;  
    void doSomethingUnsuspectious () {  
        d++; // nothing bad in and of itself  
    }  
    ...  
}  
@SandBox  
class Y {  
    X myOwnX = ...  
    void foo (X x){  
        x.doSomethingUnsuspectious();  
    }  
    void bar () { myOwnX.d = 42; }  
    ...  
    Y y = new Y();  
    y.bar(); // fine  
    y.foo(new X()); // not so fine
```

```
===== error #1  
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty  
java.lang.AssertionError: write to non-owned object: SandBoxViolation$X@290  
                                from sandbox: SandBoxViolation$Y@294  
at SandBoxViolation$X.doSomethingUnsuspectious(SandBoxViolation.java:30)  
at SandBoxViolation$Y.foo(SandBoxViolation.java:37)
```

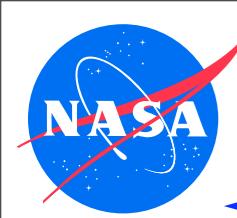


Example 8: Programming-by-Contract

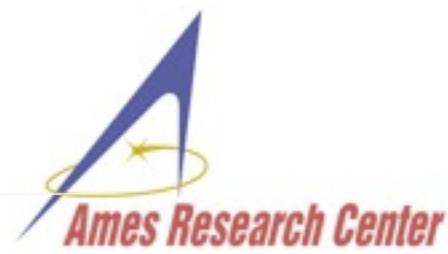


- ◆ property annotations can be parameterized with expressions
- ◆ can be a lot more complex
- ◆ inheritance aware
- ◆ expr evaluation has no SUT side-effects

```
class Base {  
    @Ensures("Result > 0")  
    public int compute (int c){ return c -1; }  
    ...  
    @Invariant({ "d within 40 +- 5", "a > 0" })  
    public class Derived extends Base {  
        double d;  int a; // not checked until return from ctor  
  
        ContractViolation() { a = 42;  d = 42; }  
  
        @Requires("c > 0")  
        @Ensures("Result >= 0")  
        public int compute (int c){  
            return c - 3;  
        }  
  
        public void doSomething (int n){  
            for (int i=0; i<n; i++){  
                d += 1.0;  
            }  
        }  
        ...  
        Derived t = new Derived();  
        //int n = t.compute(3); // would violate strengthening  
                           // base postcondition  
        t.doSomething(10); // violates 'd' invariant
```



Example 9: Overflow



- ◆ Java does not throw exceptions on overflow
- ◆ tedious to check in SUT ⇒ usually ignored

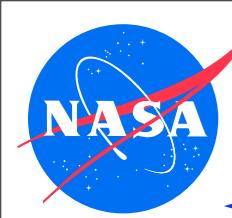
JPF configuration

```
vmInsnFactory.class =
    .numeric.NumericInstructionFactory
...
2103933699
2123371282
2142808865
...
=====
error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.ArithmaticException: integer overflow: 2142808865+19437583 = -2132720848
    at Overflow.notSoObvious(Overflow.java:18)
...
```



```
void notSoObvious(int x){
    int a = x*50;
    int b = 19437583;
    int c = a;

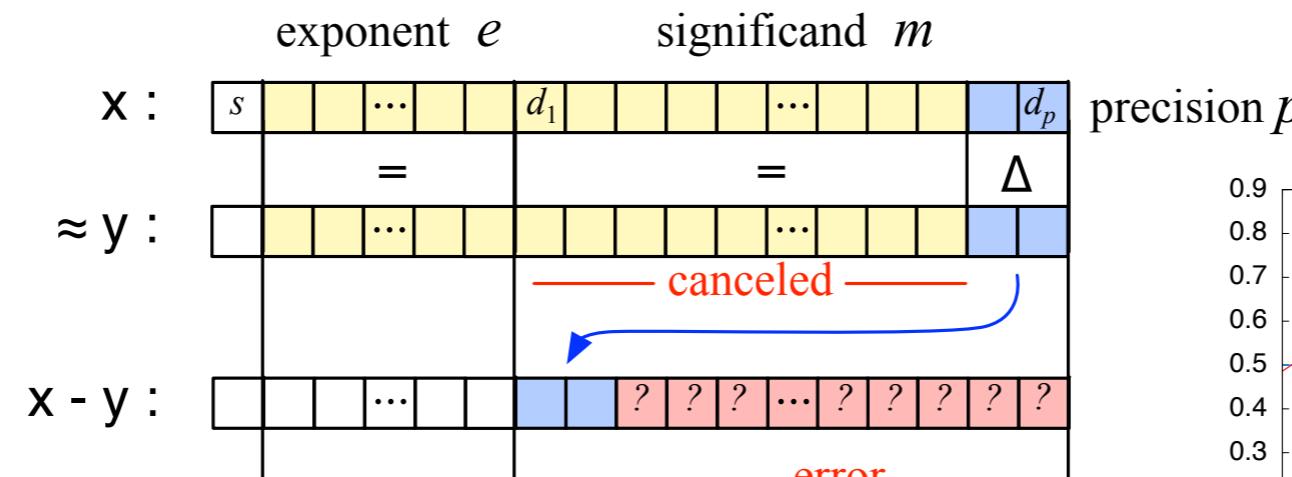
    for (int k=0; k<100; k++){
        c += b;
        System.out.println(c);
    }
    ...
    notSoObvious( 21474836);
```



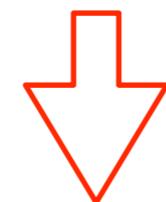
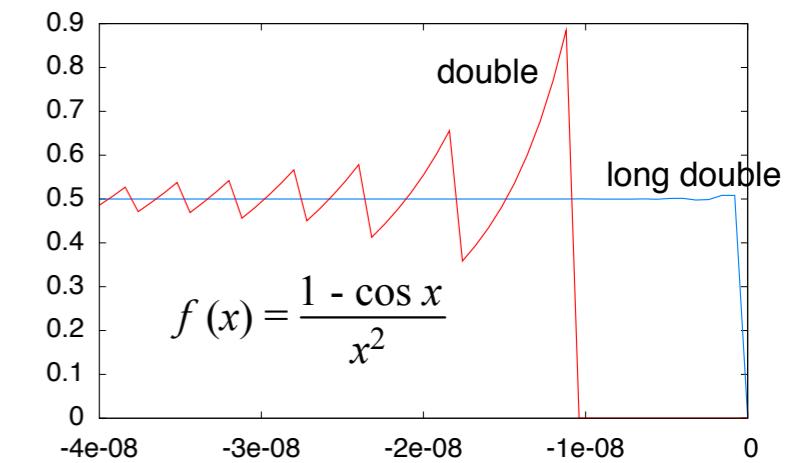
Example 10: Catastrophic Cancellation



- errors and required inspection can be a lot more intricate
- e.g.: amplification of previous operand errors due to cancellation of identical leading bits in the significands, followed by normalization of the result



```
double a = 77617.0;  
double b = 33096.0;  
res = 333.75*pow(b,6) + pow(a,2)*(11*pow(a,2)*pow(b,2) -  
    pow(b,6) - 121*pow(b,4) - 2) + 5.5*pow(b,8) + a/(2*b);
```



```
vmInsn_factory.class =  
.numeric.NumericInstructionFactory
```

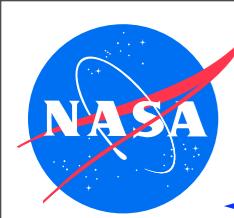
```
res=-1.1805916207174113e21 (should be -0.827396...)
```

```
...
```

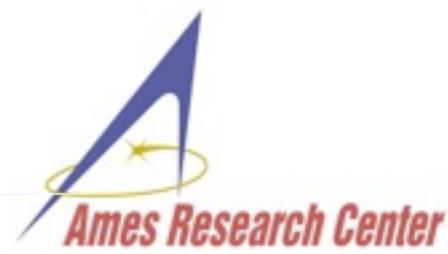
```
===== error #1
```

```
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
```

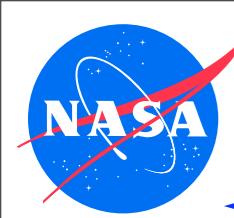
```
java.lang.ArithException: cancellation of:  
-7.917111340668963E36 + 7.917111340668962E36  
= -1.1805916207174113E21
```



Examples: Model Verification



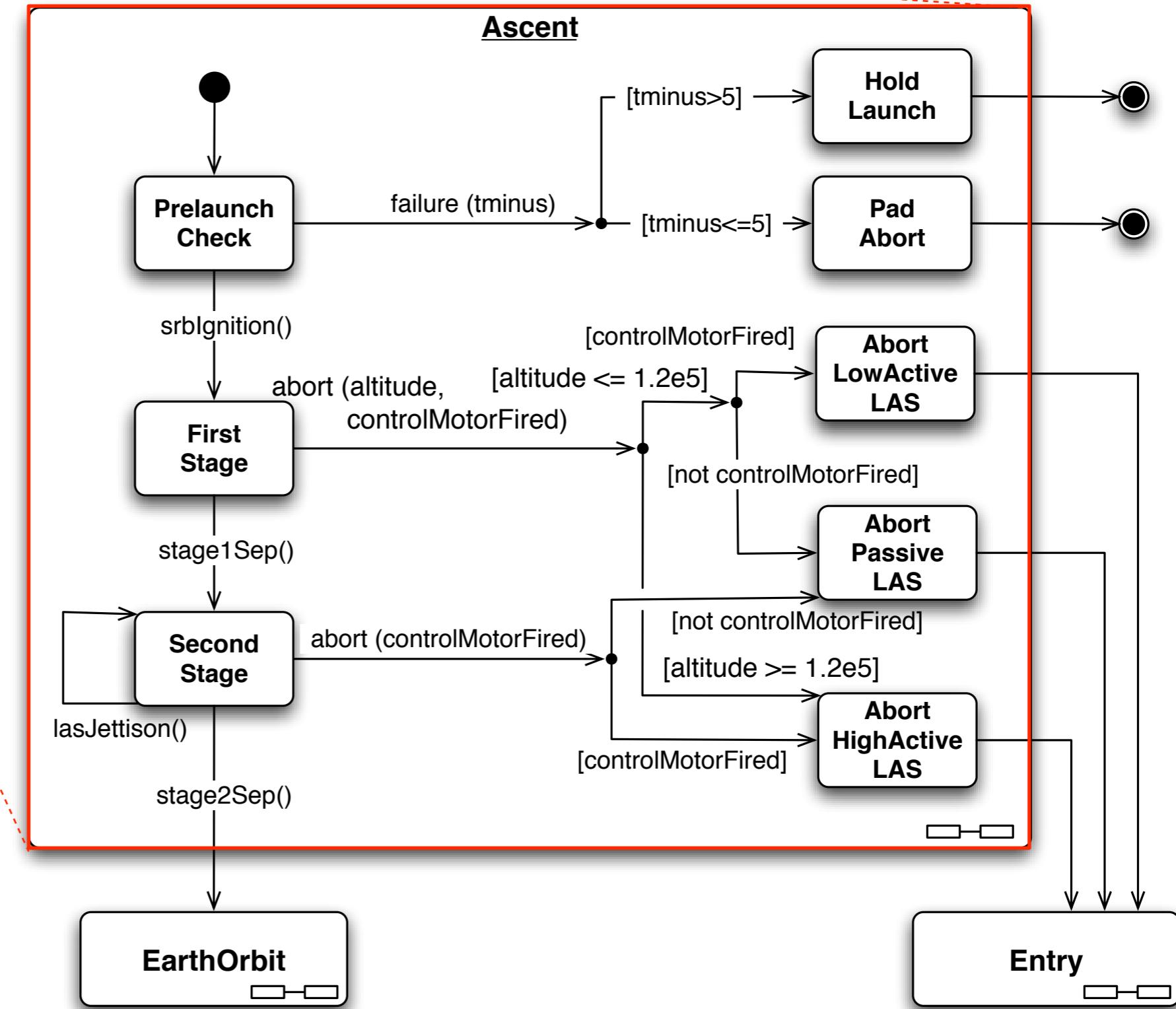
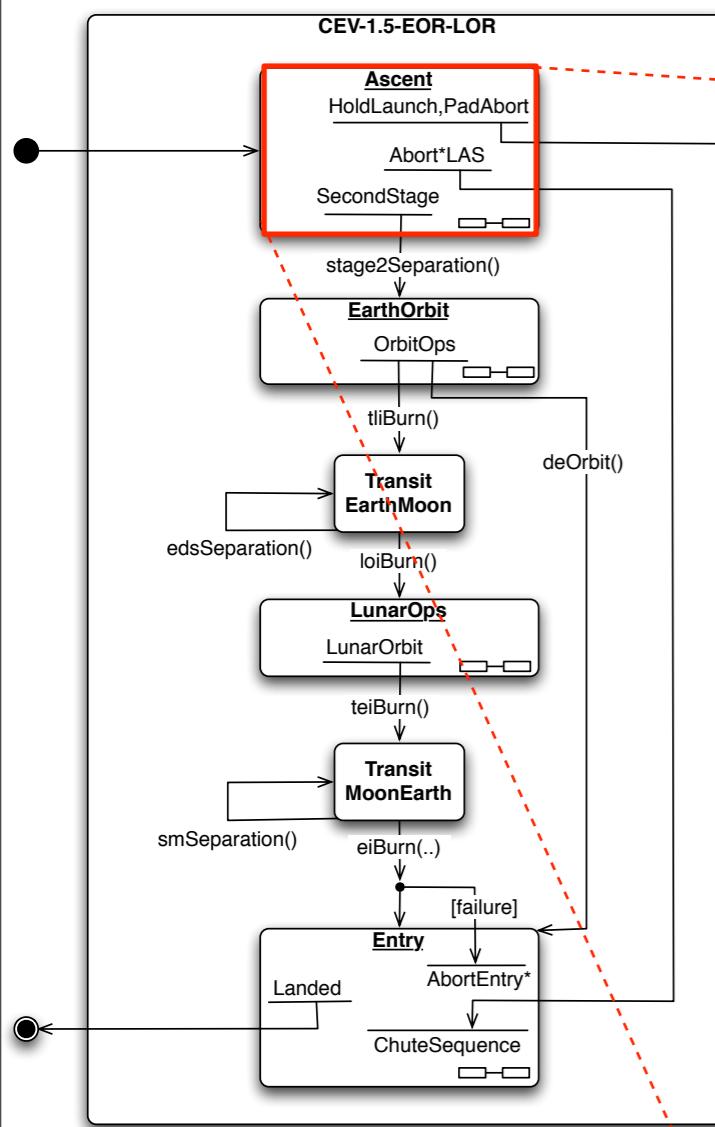
- ◆ motivation: model driven development
- ◆ as models get more complex, risk of model inconsistencies increases
- ◆ classic example: state machine verification
- ◆ usually requires translation into JPF specific model representation
(application that is JPF dependent)



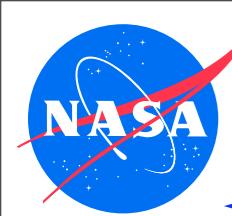
UML Model Checking (1)



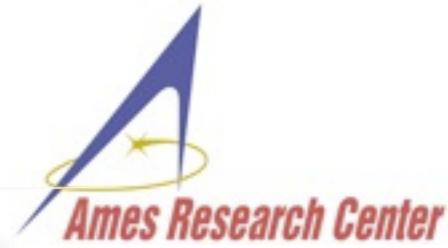
model is understandable ..



.. but does it work?
(actions, guards,
reachability etc.)

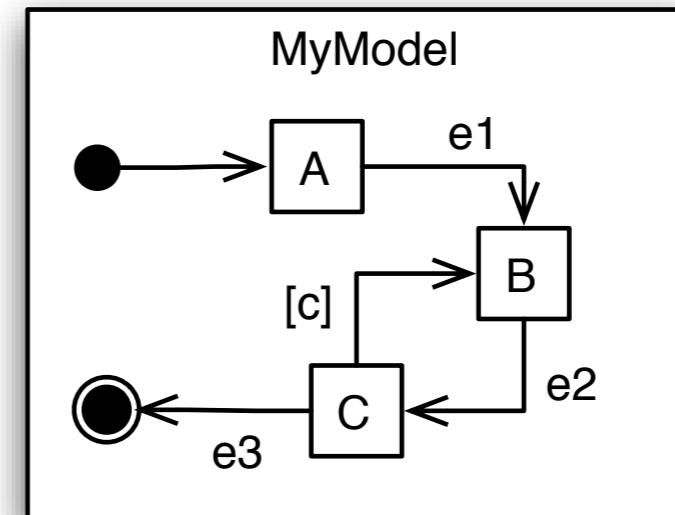


UML Model Checking (2)



- ◆ to find out:
 - make model executable
(give it formal semantics)
⇒ translation
 - execute it with JPF
(systematically checking
all enabling events and
parameter combinations)

UML
Statechart

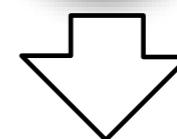
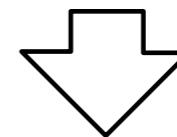
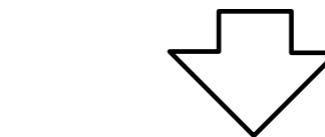
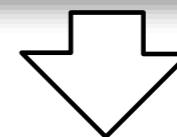


Java
Program

```
class MyModel extends State {  
    class A extends State {...}  
    class B extends State {...}  
    class C extends State {...}  
}
```

Software
Model Checker

Verification
Report

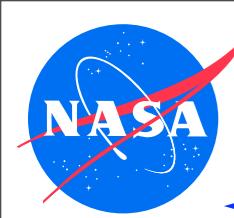


[guidance script]

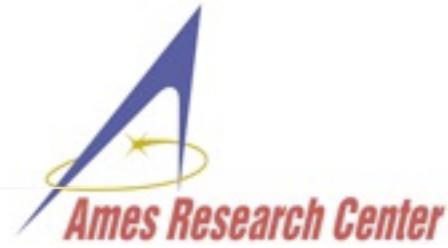
```
..  
e1()  
e2()  
ANY{*}
```



```
error:  
unreachable end state...  
trace:  
e1(), e2() ...
```

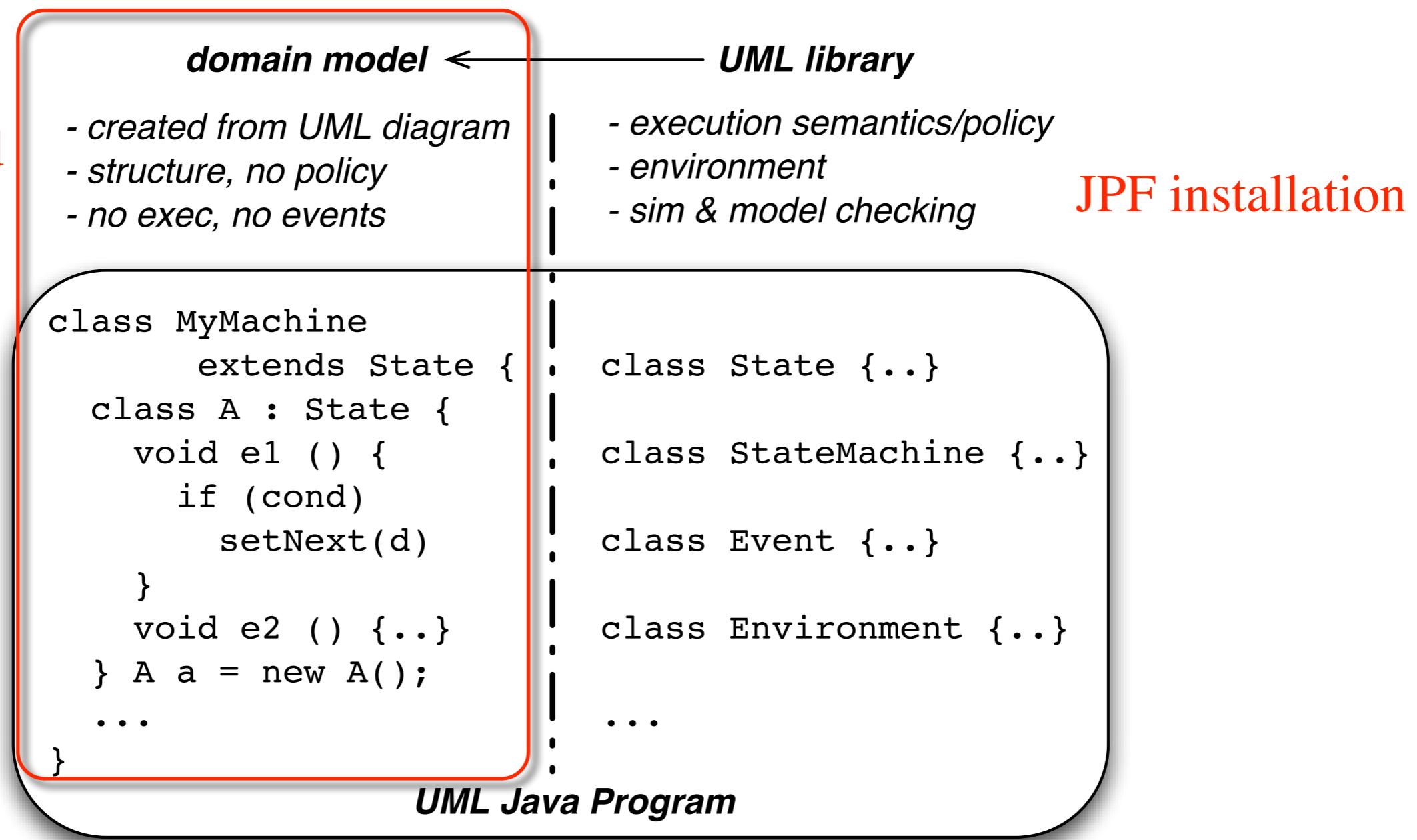


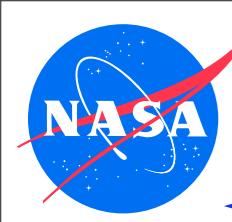
UML Model Checking (3)



- ◆ layer 1: domain model (code translated from UML diagram)
- ◆ layer 2: UML modeling library (part of JPF distribution)
- ◆ goals: model readability, UML/program state space alignment

translated



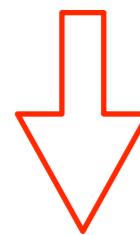


Example 11: UML Model Checking

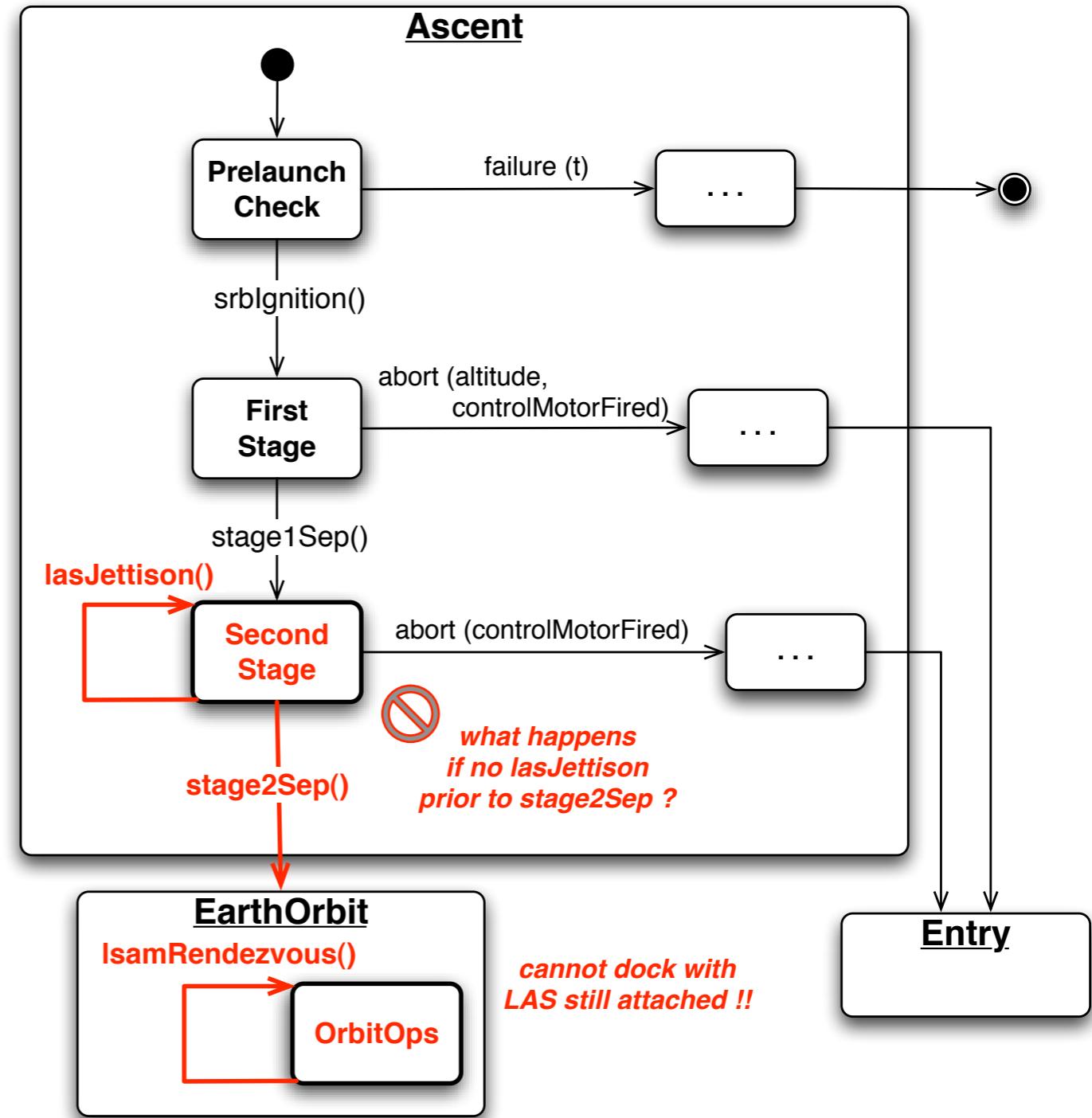


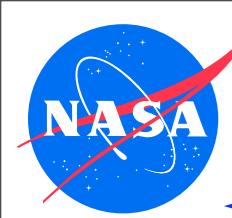
- ◆ property: “no unhandled exceptions”

```
class OrbitOps {...
void lsamRendezvous(){...
    assert !spacecraft contains(LAS):
        "lsamRendezvous with LAS attached"
    ...
}
}
```



```
...
===== error #1
NoUncaughtExceptionsProperty
AssertionError:
    lsamRendezvous with LAS attached
    at ...
=====
choice trace #1
srbIgnition()
stage1Separation()
stage2Separation()
lsamRendezvous()}
```





Example 12: UML Model Checking



- ◆ property: “*all states of the model have to be reachable*”

```
class EarthOrbit {..  
void entry_checkSensors() {..  
if (!checkEarthSensor()) earthSensorFailed = true;
```

autom. entry action

```
class Insertion {..  
void entry_setMajorMode() {..resetSensors();..}
```

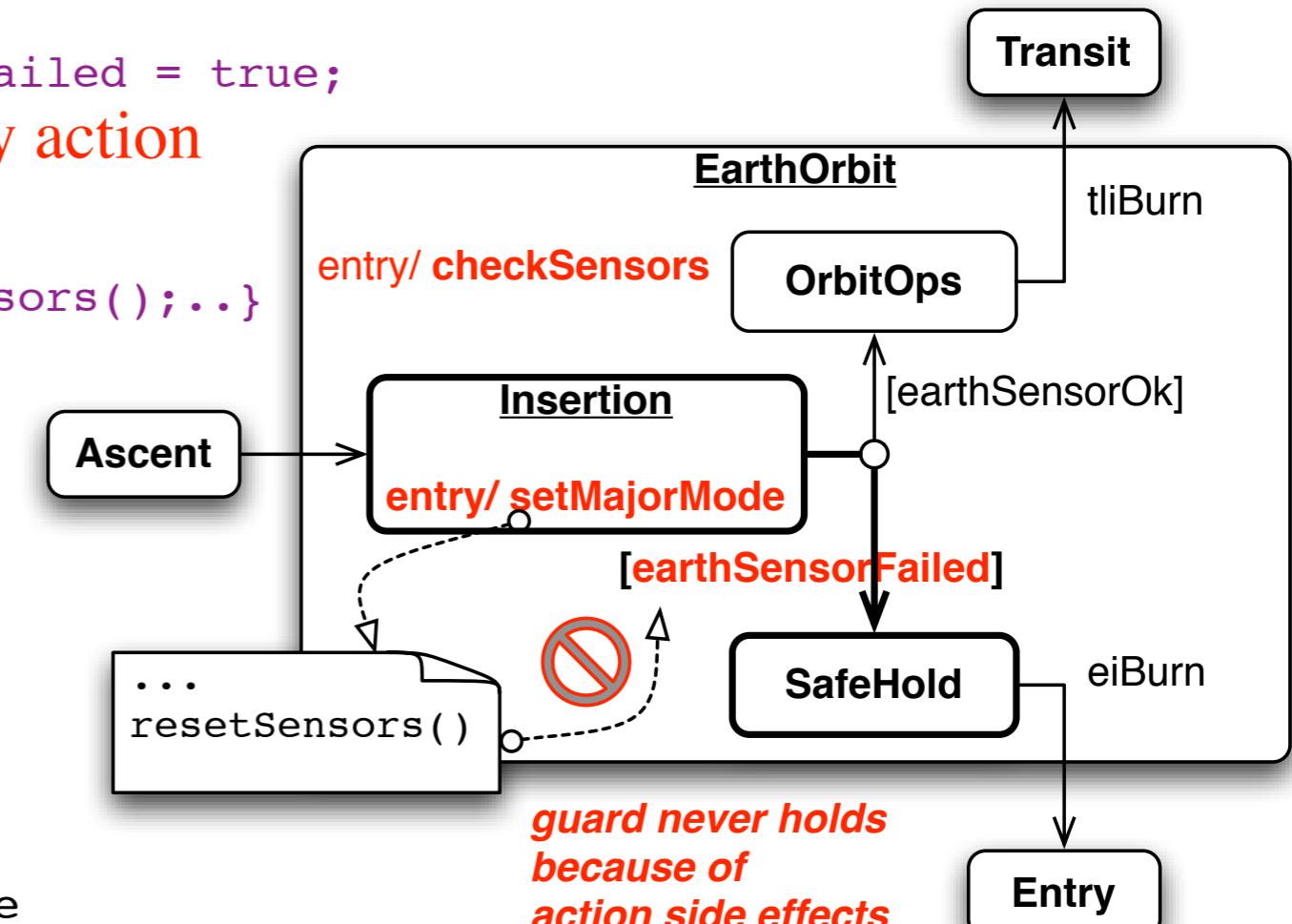
```
void resetSensors() {..  
earthSensorFailed = false; }
```

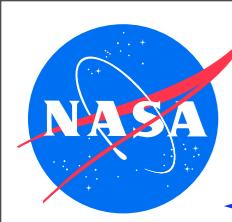
```
void completion() {  
if (earthSensorFailed)  
setNextState(safeHold);  
..
```

```
}  
}  
  
listener=.tools.sc.Coverage  
sc.required=earthOrbit
```

===== error #1

gov.nasa.jpf.tools.sc.Coverage
required earthOrbit.safeHold NOT COVERED





Example 13: UML Model Checking



- ◆ property: “*no ambiguous transitions*”
- ◆ example: overlapping guard conditions

```
@Params("5000|120000|200000,...")
void abort(altitude,...){...
    if (altitude <= 1.2e5)
        setNextState(abortLowActive)
...
    if (altitude >= 1.2e5)
        setNextState(abortHighActive)
```



===== error #1 ...

AssertionError:

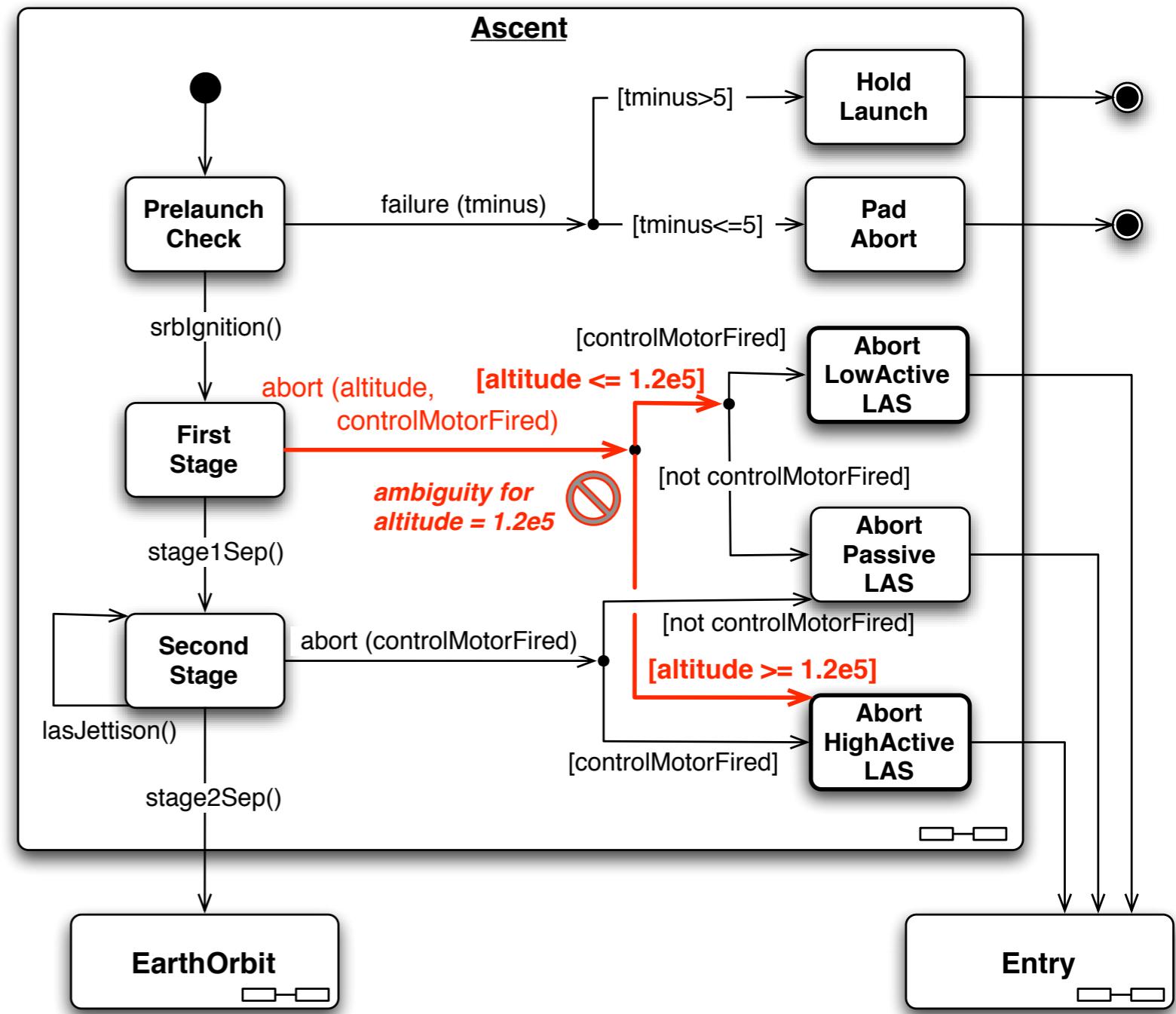
ambiguous transitions in:

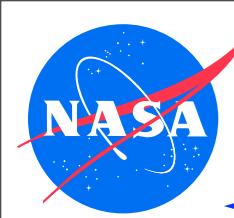
ascent.firstStage

processing event: abort(120000,true)
state 1: ascent.abortHighActiveLAS
state 2: ascent.abortLowActiveLAS

...

===== choice trace #1
srbIgnition()
abort(120000,true)

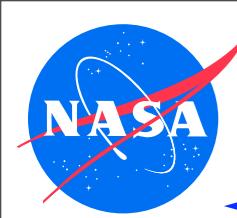




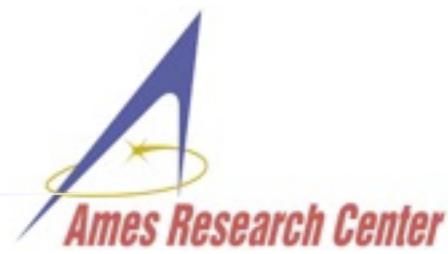
Examples: Test Case Generation



- ◆ test creation is very expensive - how many tests do we need (instruction coverage, branch coverage, MCDC, path coverage ..)
- ◆ the application for *symbolic execution*
- ◆ “normal” verification - use data to find out which paths to explore
- ◆ problem with huge input spaces (what test data is interesting?)
- ◆ symbolic execution uses opposite direction - use program structure to deduce which data values need to be tested to reach interesting program locations (exceptions, coverage requirements)



Test Case Generation (1)

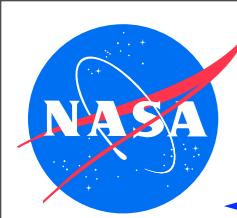


- ◆ how do we identify interesting parameter values if they are not known a priori (example 13 @PARAMS annotation)?

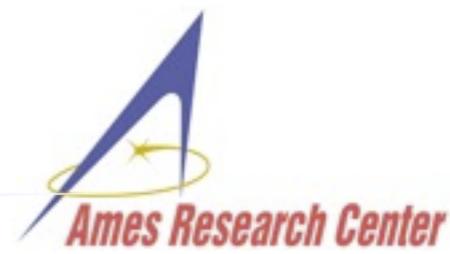
~~@PARAMS(5000|120000|200000)~~

```
void abort(altitude){  
    ...  
    if (altitude <= 1.2e5)  
        setNextState(abortLowActive)  
    ...  
    if (altitude >= 1.2e5)  
        setNextState(abortHighActive)
```

- ◆ answer: test case generation with *Symbolic Execution*



Test Case Generation (2)



Program Control Structure

```
if (x > C) {  
    ...  
    if (y < D) {  
        ...  
        ...  
        if (x >= y) {  
            ...  
        } else {  
            ...  
        }  
    }  
    ...  
}
```

Path Conditions

$$\xrightarrow{\text{Symbolic Execution}} PC_1: (x > C)$$

$$\xrightarrow{\text{Symbolic Execution}} PC_2: (y < D)$$

⋮

$$\xrightarrow{\text{Symbolic Execution}} PC_n: \neg(x \geq y)$$

$$PC_1 \wedge PC_2 \wedge \dots \wedge PC_n = \text{true}$$

constraint solver

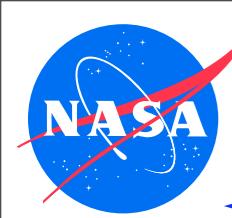
[$x=.., y=..$]

concrete test vector(s)

unreachable code

?

program logic
too complex ?



Example 14: Symbolic Execution



- ◆ look for instruction that constitutes property violation and compute data that leads to it

```
vmInsnFactory.class =  
gov.nasa.jpf.symbc.SymbolicInstructionFactory  
symbolic.method = ..abort(sym#con)
```

...

symbolic.dp=choco

Symbolic Execution Mode

...

***Execute symbolic INVOKEVIRTUAL: abort(IZ)V
(altitude_1_SYMINT, controlMotorFired_CONCRETE)

...

Property Violated: PC is # = 2

altitude_1_SYMINT[120000] >= CONST_120000 &&
altitude_1_SYMINT[120000] <= CONST_120000

...

Property Violated: result is "java.lang.AssertionError: ambiguous transitions
in: ascent.firstStage..."

...

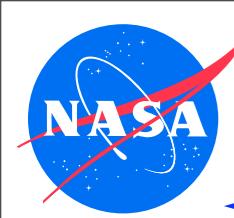
===== Method Summaries

Symbolic values: altitude_1_SYMINT

abort(120000,true) --> "java.lang.AssertionError: ambiguous transitions in:
ascent.firstStage..."

```
void abort(altitude,controlMotorFired){  
    ...  
    if (altitude <= 1.2e5)  
        setNextState(abortLowActive)  
    ...  
    if (altitude >= 1.2e5)  
        setNextState(abortHighActive)}
```





Example 15: Test Case Generation



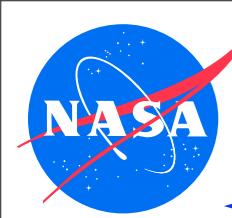
- ◆ Symbolic Execution especially suitable to generate test suite to achieve path coverage

```
vmInsn_factory.class =  
    .symbc.SymbolicInstructionFactory  
symbolic.method= TestPaths.testMe(sym#con)  
listener = .symbc.SymbolicListener
```

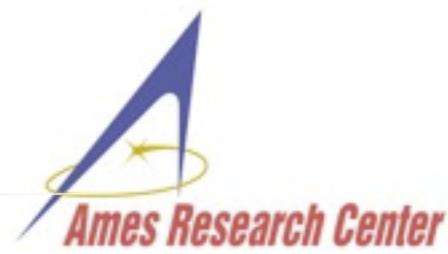
```
...  
***Execute symbolic INVOKESTATIC: testMe(IZ)V  ( x_1_SYMINT, b_CONCRETE )  
..  
PC # = 2  
x_1_SYMINT[1200] >= CONST_1200 &&  
x_1_SYMINT[1200] <= CONST_1200  
...  
x_1_SYMINT[-1000000] < CONST_1200 &&  
x_1_SYMINT[-1000000] <= CONST_1200  
...  
===== Method Summaries  
...  
testMe(1200,true)      BLOCK-1, BLOCK-2  
testMe(-1000000,true)  BLOCK-1  
testMe(1201,true)       BLOCK-2
```



```
public class TestPaths {  
    ...  
    // what tests do we need to cover all paths?  
    public static void testMe (int x, boolean b) {  
        if (x <= 1200){  
            // BLOCK-1  
        }  
        if(x >= 1200){  
            // BLOCK-2  
        }  
    }  
}
```

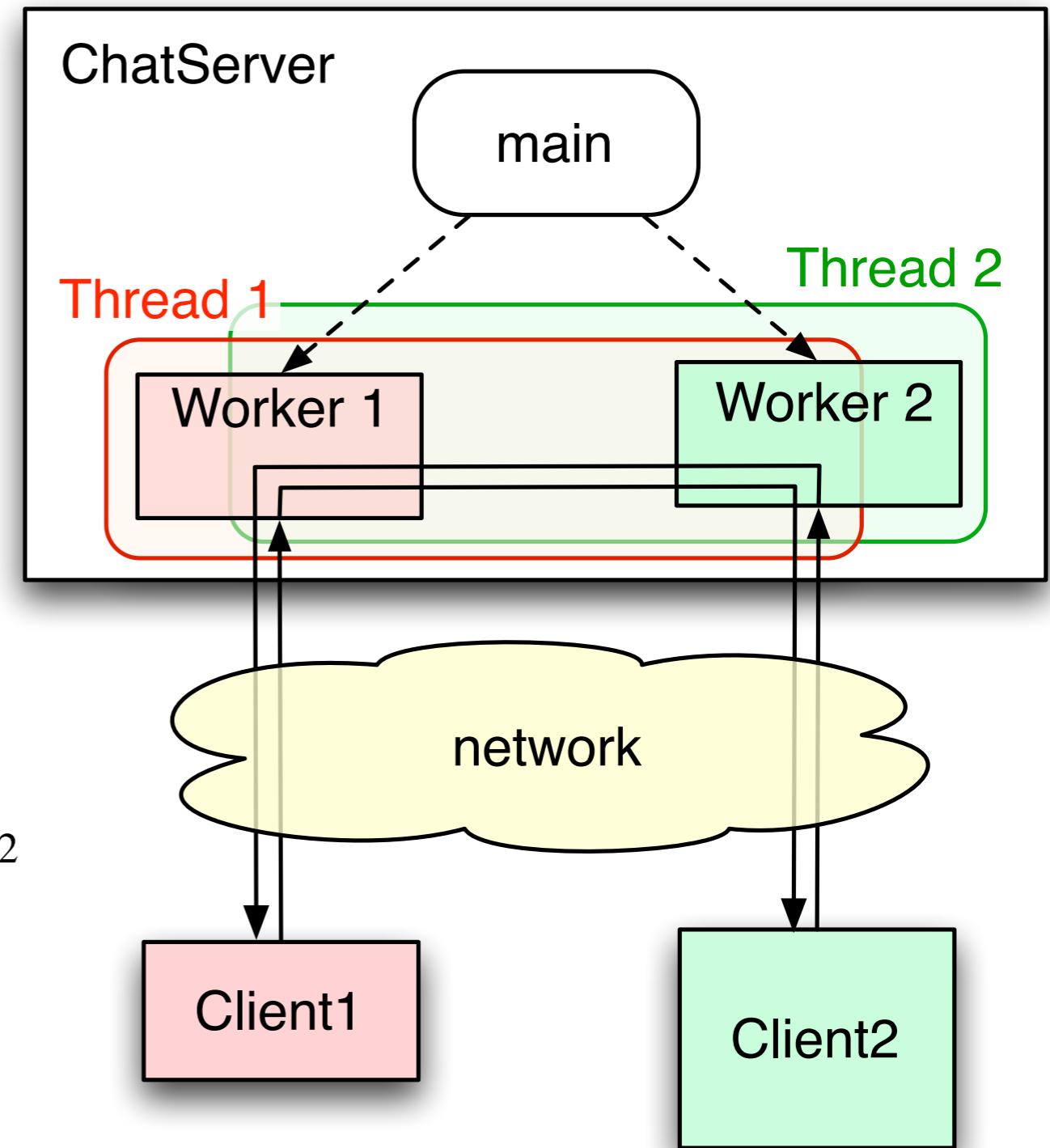


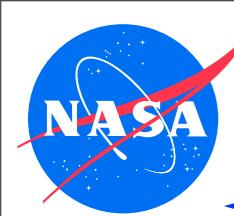
Examples: Distributed Applications



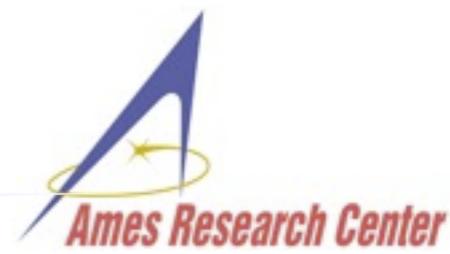
- ◆ example: ChatServer
- ◆ 1 ChatServer process
- ◆ N Client processes
- ◆ server creates one worker object/thread per connection

- ◆ model checking problem:
 - state explosion for known clients:
$$S_{\text{total}} = S_{\text{server}} \times S_{\text{client1}} \times S_{\text{client2}}$$
 - what to do if clients unknown ?

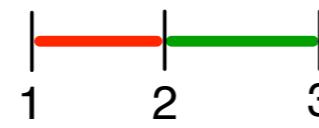
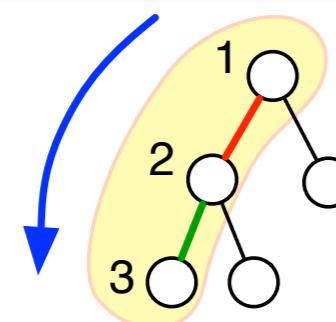
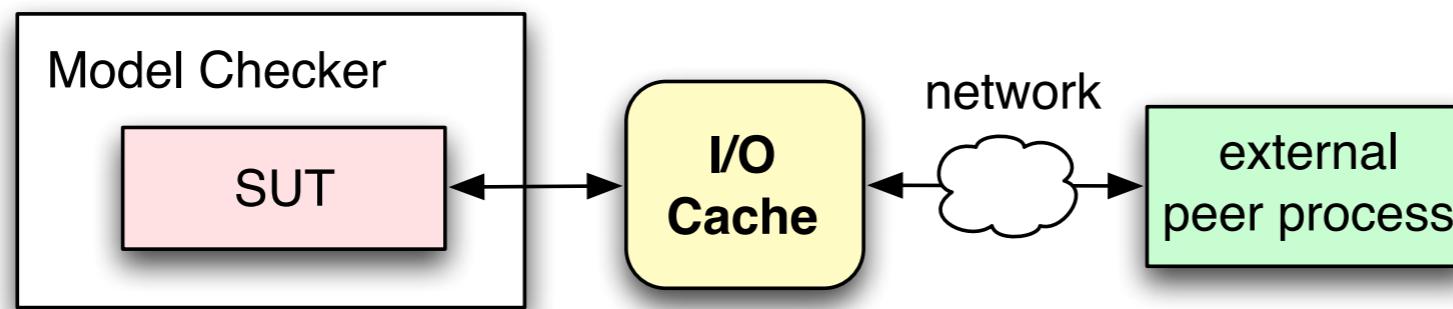




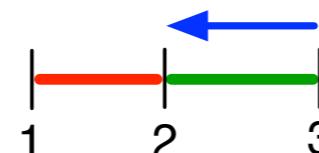
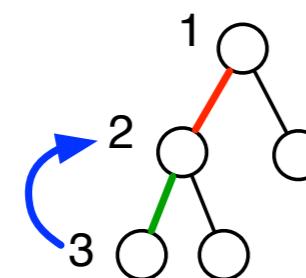
IO Cache



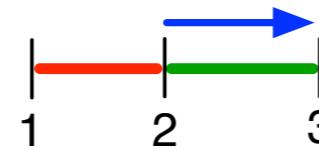
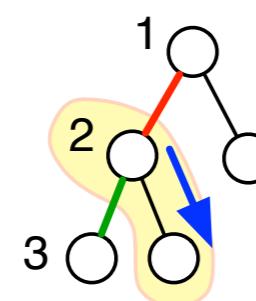
- ◆ no stubs required
- ◆ environment executes normally
- ◆ protocol structure preserved



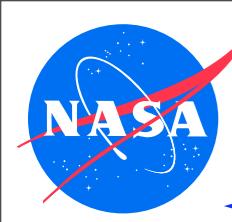
new state: store I/O data streams "globally"
cache I/O data & message size
map program state to stream position



backtracking: restore I/O state "locally"
reset streams back to old state



continue: replay previous I/O
duplicate sends: ignore
duplicate reads: previous peer response



Example 16: ChatServer Deadlock



- ◆ jpf-net-iocache finds server defect (deadlock) without analyzing clients

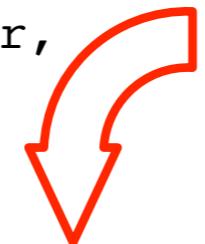
```
listener=.network.listener.CacheNotifier,  
    .network.listener.CacheLogger
```

```
...  
[SEVERE] unprotected field access  
of ChatServer.workers  
in thread: Thread-1..
```

```
...  
Client quit, 1 client(s) left.  
Client quit, 1 client(s) left.
```

```
===== error #1  
gov.nasa.jpf.jvm.NotDeadlockedProperty  
deadlock encountered:  
  thread index=0,name=main,status=WAITING..  
  thread index=1,name=Thread-0,status=TERMINATED..  
  thread index=2,name=Thread-1,status=TERMINATED..
```

```
===== snapshot #1  
thread index=0,name=main,status=WAITING...  
  waiting on: gov.nasa.jpf.network.chat.ChatServer@294  
call stack:  
  at gov.nasa.jpf.network.chat.ChatServer.accept(ChatServer.java:85)  
  ...
```



```
class Worker implements Runnable {
```

```
  ...  
  public void run() {  
    ..server.remove(id)  
  ...
```

```
class ChatServer {
```

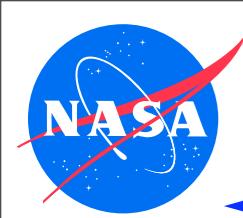
```
  ...  
  void remove(int n) {  
    workers[n] = null;  
    activeClients--;  
    synchronized (this){  
      notify();  
    }  
  ...
```



```
void accept(int maxServ, int port) {
```

```
  ...  
  synchronized (this) {  
    while (activeClients != 0) {  
      try {  
        wait();  
      ...
```

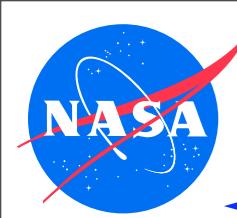




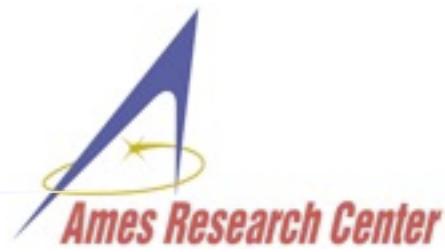
Using JPF



- ◆ obtaining JPF
- ◆ installing, building and testing JPF
- ◆ JPF configuration
- ◆ running JPF
- ◆ JPF and NetBeans
- ◆ JPF and Eclipse



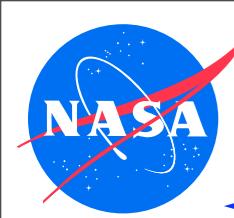
Obtaining JPF



- ◆ Mercurial repositories on
<http://babelfish.arc.nasa.gov/hg/jpf/{jpf-core,jpf-aprop,...}>
- ◆ Steps
 - (1) obtain and install Mercurial (<http://mercurial.selenic.com>)
 - (2) clone required JPF repositories

```
>hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-core
>hg clone .../jpf/jpf-numeric
...

```
- ◆ don't clone whole .../hg/jpf directory !
- ◆ don't use https://.. (only required for push)
- ◆ alternative: download binary distributions from wiki project page attachments (e.g. <http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/jpf-core>) - not recommended at this time



Installing, Building and Testing JPF



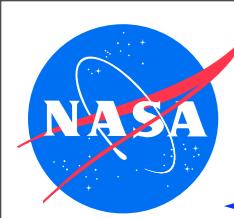
- ◆ create `site.properties` file in `${user.home}/jpf` directory
(see <http://babelfish.arc.nasa.gov/trac/jpf/wiki/install/site-properties>, required so that JPF can find jpf-core and installed projects without extensive manual classpath configuration)

```
jpf-core = ${user.home}/projects/jpf/jpf-core
...
jpf-numeric = ${user.home}/projects/jpf/jpf-numeric
extensions+=,${jpf-numeric}
...
```

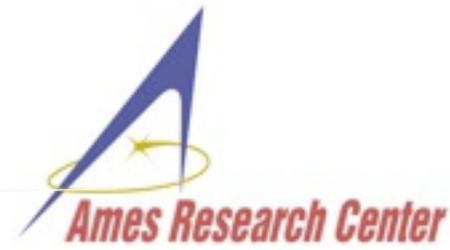
- ◆ build & test jpf-core and required extensions
(with included Ant builder)

```
>cd jpf-core
>bin/ant clean test
...
>cd ../jpf-numeric
>bin/ant clean test
...
```

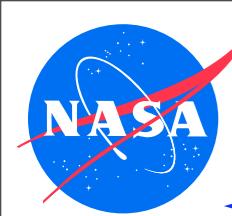
```
$ bin/ant clean test
Buildfile: .../jpf-core/build.xml
clean:
    [delete] Deleting directory .../jpf-core/build
-init:
    [mkdir] Created dir: .../jpf-core/build
...
build:
    [jar] Building jar: .../jpf-core/build/jpf.jar
test:
    [junit] Running TypeNameTest
    [junit] Tests run: 1, Failures: 0, Errors: 0...
    ...
BUILD SUCCESSFUL
Total time: 1 minute 30 seconds
```



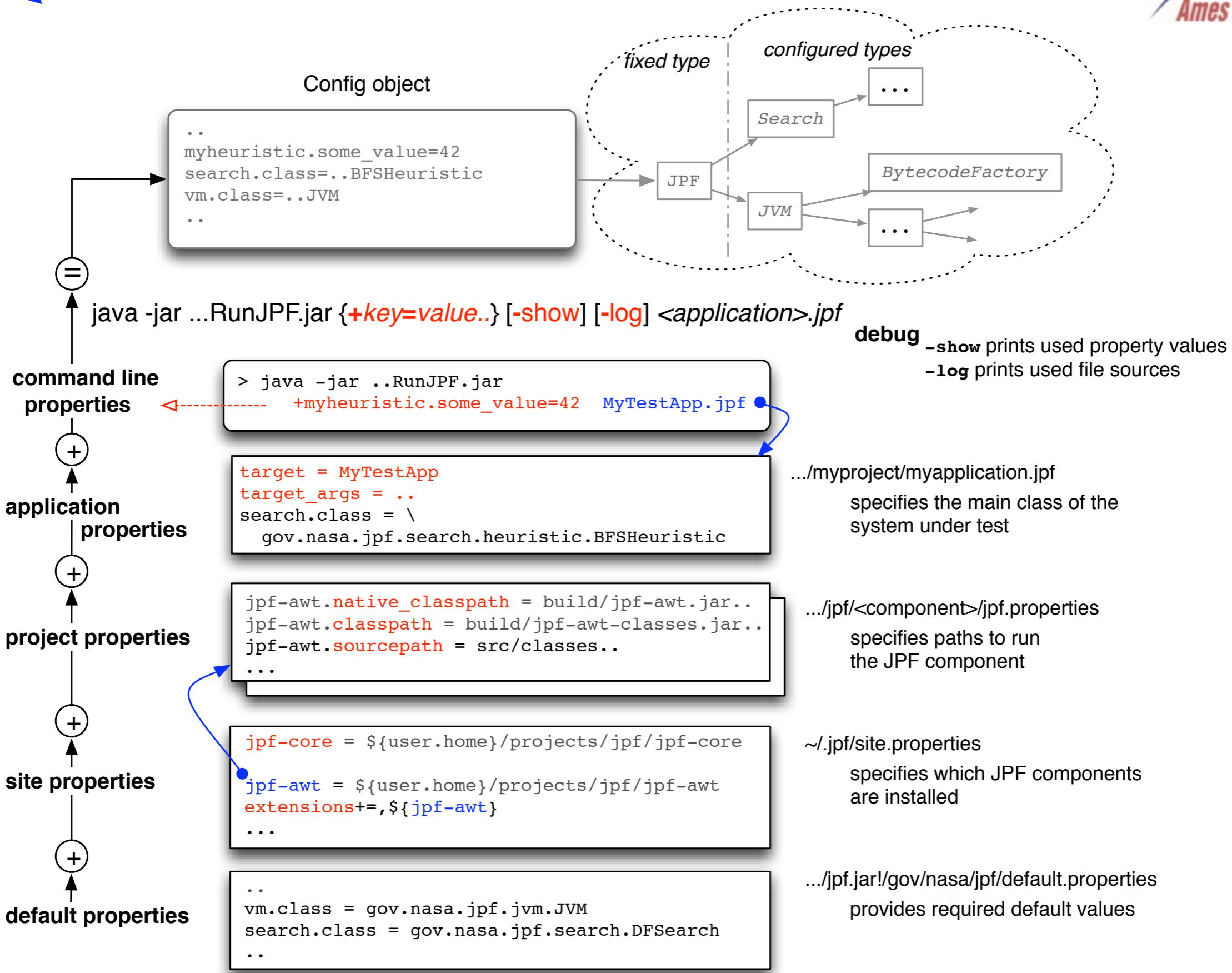
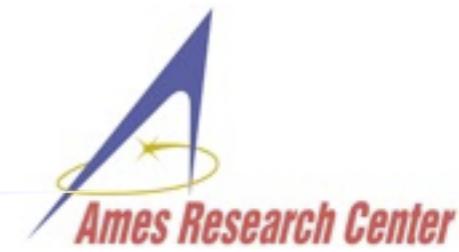
JPF Configuration (1)

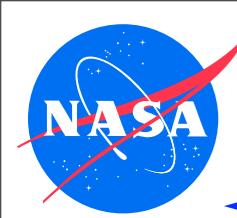


- ◆ almost nothing in JPF is hardwired ⇒ great flexibility but config can be intimidating
- ◆ all of JPFs configuration is done through Java properties (but with some extended property file format)
 - keyword expansion `jpf-root = ${user.home}/jpf`
 - ▶ previously defined properties
 - ▶ system properties
 - append `extensions+=,jpf-aprop` no space between key and '+' !
 - prepend `+peer_packages=jpf-symbc/build/peers,`
 - directives
 - ▶ dependencies `@requires jpf-awt`
 - ▶ recursive loading `@include ../jpf-symbc/jpf.properties`
- ◆ hierarchical process
 - system defaults (from jpf.jar)
 - site.properties
 - project properties from all site configured projects (<project-dir>/jpf.properties)
 - current project properties (.jpf.properties)
 - selected application properties file (*.jpf)
 - command line args (e.g. `bin/jpf +listener=.listeners.ExecTracker ...`)

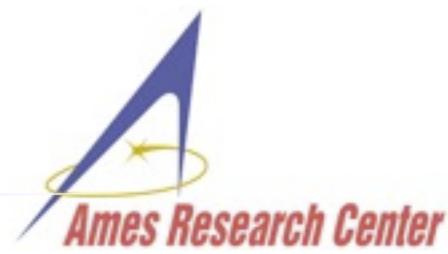


JPF Configuration (2)

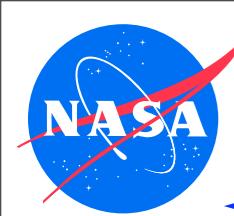




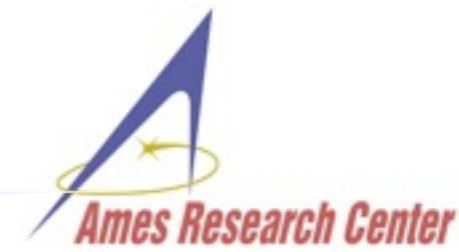
Running JPF



- ◆ for purists (tedious, do only if you have to)
 - setting up classpaths >`export CLASSPATH=...jpf-core/build/jpf.jar...`
 - invoking JVM >`java gov.nasa.jpf.JPF +listener=... x.y.MySUT`
- ◆ using site config and starter jars (much easier and portable)
 - explicitly >`java -jar tools/RunJPF.jar MySUT-verify.jpf`
 - using scripts >`bin/jpf MySUT-verify.jpf`
- ◆ running JPF from within JUnit
- ◆ running JPF from your program (tools using JPF)
- ◆ using NetBeans or Eclipse plugins
 - “Verify..” context menu item for selected *.jpf application property file
 - using provided launch configs (Eclipse) or run targets (NetBeans)



Running JPF: from JUnit

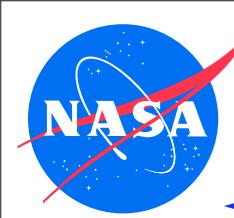


- ◆ derive your test cases from `gov.nasa.jpf.util.test.TestJPF`
- ◆ run normally under JUnit or from Ant `<junit ...>` task
- ◆ be aware of that test case is run by JVM *and* JPF

```
public class ConstTest extends TestJPF {  
  
    static final String[] JPF_ARGS = { "+listener=.aprop.listener.ConstChecker" };  
  
    //--- standard driver to execute single test methods  
    public static void main(String[] args) {  
        runTestsOfThisClass(args);  
    }  
  
    //--- the test methods  
    @Test  
    public void testStaticConstOk () {  
        if (verifyNoPropertyViolation(JPF_ARGS)){  
            ConstTest.checkThis();  
        }  
    }  
    ...  
}
```

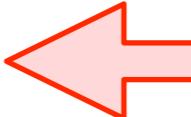
Verification goal

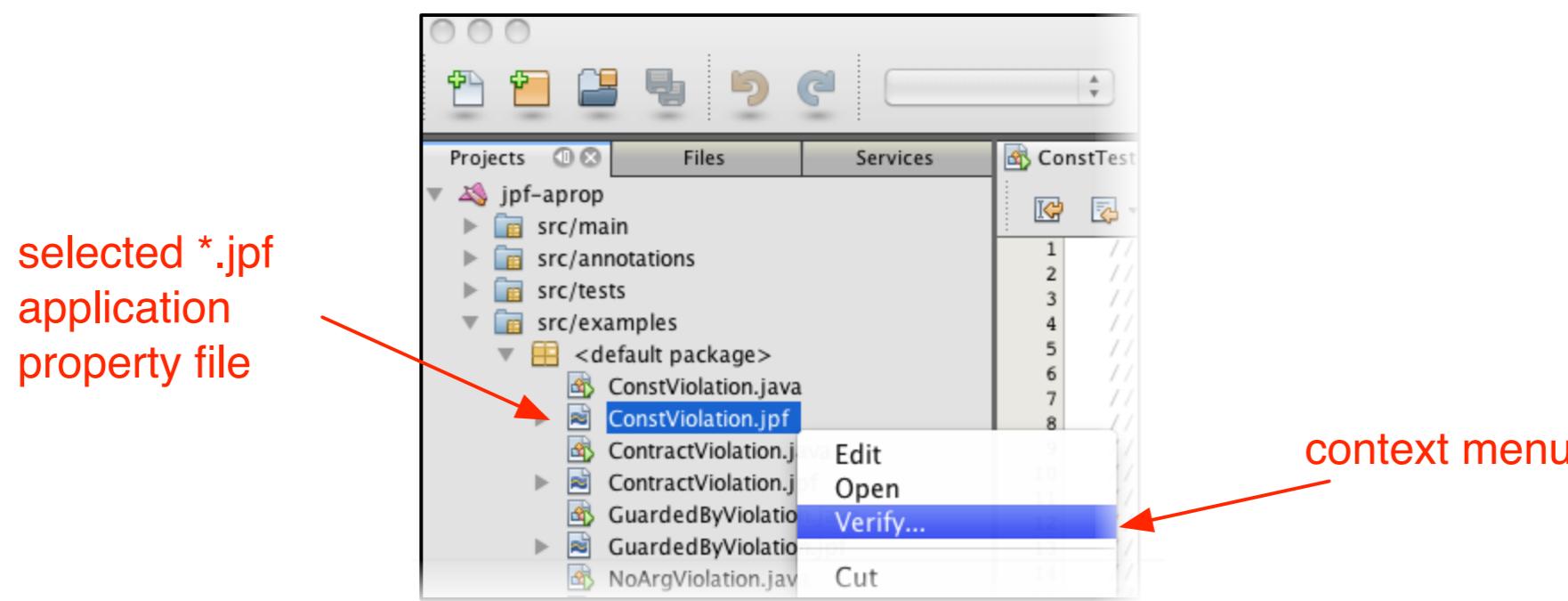
code checked by JPF

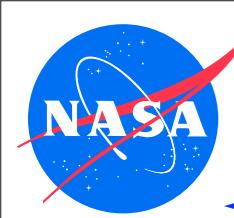


Running JPF from NetBeans



- ◆ use project provided run/debug tasks (requires `nbproject/ide-file-targets.xml` in project)
 - select `*.jpf` file in projects view
 - invoke **Run→Run File** from menubar (not in context menu)
 - results in Output view
- ◆ use NetBeans JPF plugin 
from <http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/netbeans-jpf>
 - download & install attached `*.nbm` if you don't want to build
 - optionally install jpf-shell extension if you want JPF to run in own window
 - launch JPF by selecting `*.jpf` file and invoking “**Verify..**“ context menu item



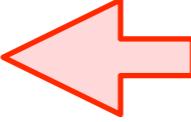


Running JPF from Eclipse

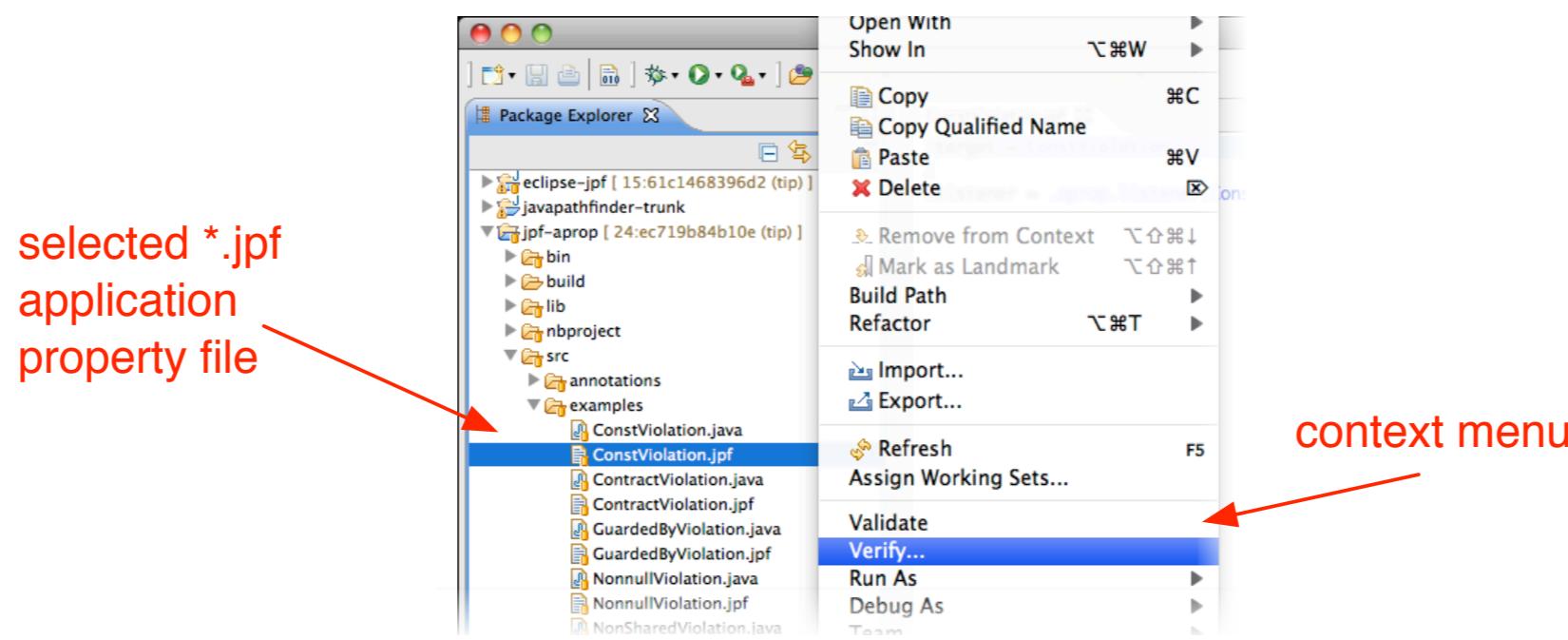


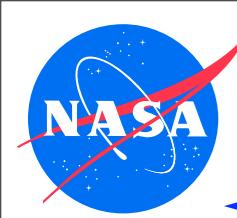
- ❖ use project provided launch configuration (requires [eclipse/run-JPF.launch](#) in project)

- select *.jpf file in projects view
- invoke **Run As**→**Run Configurations**→**run-JPF** from context menu
- results in Output view

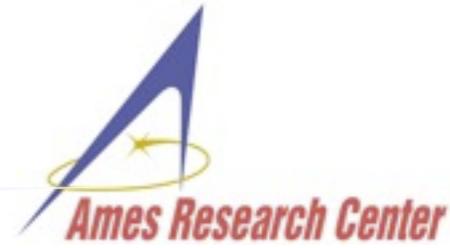
- ❖ use Eclipse JPF plugin  from <http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/eclipse-jpf>

- install from update site if you don't want to rebuild
<http://babelfish.arc.nasa.gov/trac/jpf/raw-attachment/wiki/install/eclipse-plugin/update/>
- optionally install jpf-shell extension if you want JPF to run in own window
- launch JPF by selecting *.jpf file and invoking “**Verify..**” context menu item



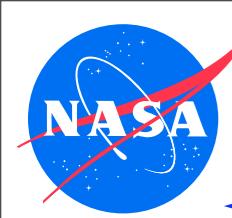


Using JPF Shell



- ◆ sometimes JPF reports too long for IDE Output view
- ◆ some applications / properties might require specific visualization
- ◆ solution: configured JPF shell
- ◆ specify shell class and views in your application property file (*.jpf)
- ◆ launch normally through “Verify” context menu
- ◆ JPF shell shows as separate window
- ◆ sources can be shown in IDE editors by clicking links in JPF shell window

```
# JPF application property file using generic JPF shell
target=oldclassic
...
shell=.shell.basicshell.BasicShell
shell.panels+=,searchgraph
...
...
```



Why Using JPF Shell ?



run SUT (test) run JPF (verify) browse complex reports

browse report topics

configurable view tabs
shell=.shell.basicshell.BasicShell
shell.panels=...

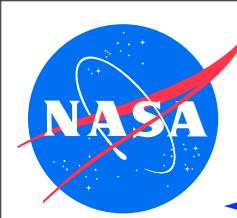
clickable source links

inspect JPF run

.. shell.panels+=,searchgraph

application/domain specific views

shell=.shell.awt.UIShell
shell.panels+=,component,trace,script



Extending JPF

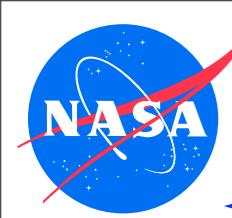


◆ Basics

- a VM running inside a JVM
- JPF Components
- JPF directory structure
- JPF runtime structure

◆ Extension Mechanisms

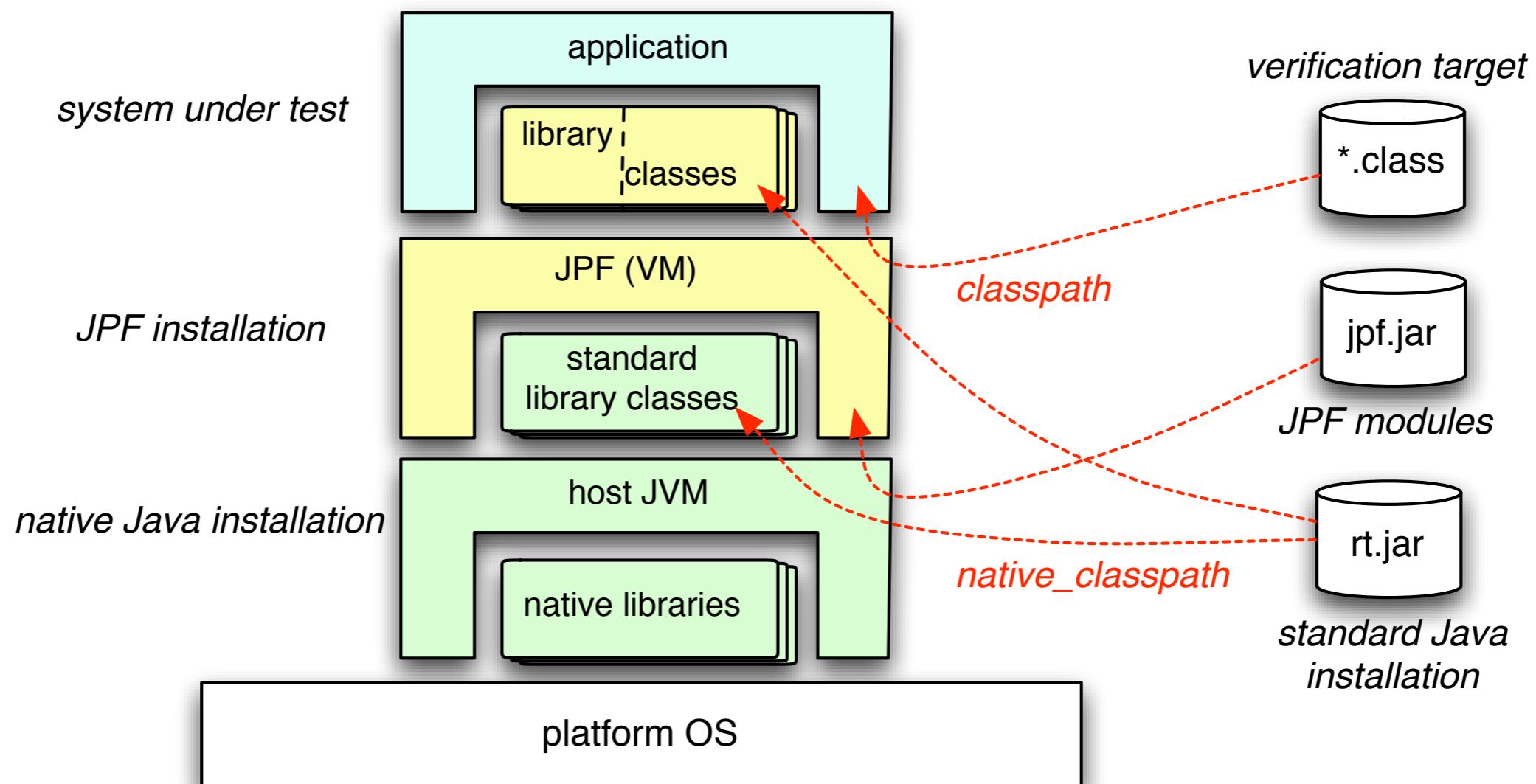
- Search Policies
- ChoiceGenerators
- Listeners
- Bytecode Factories
- Model-Java-Interface and Native Peers
- Attributes

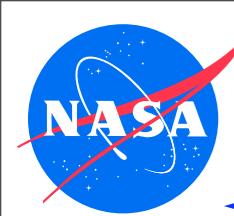


Basics: a VM running inside JVM

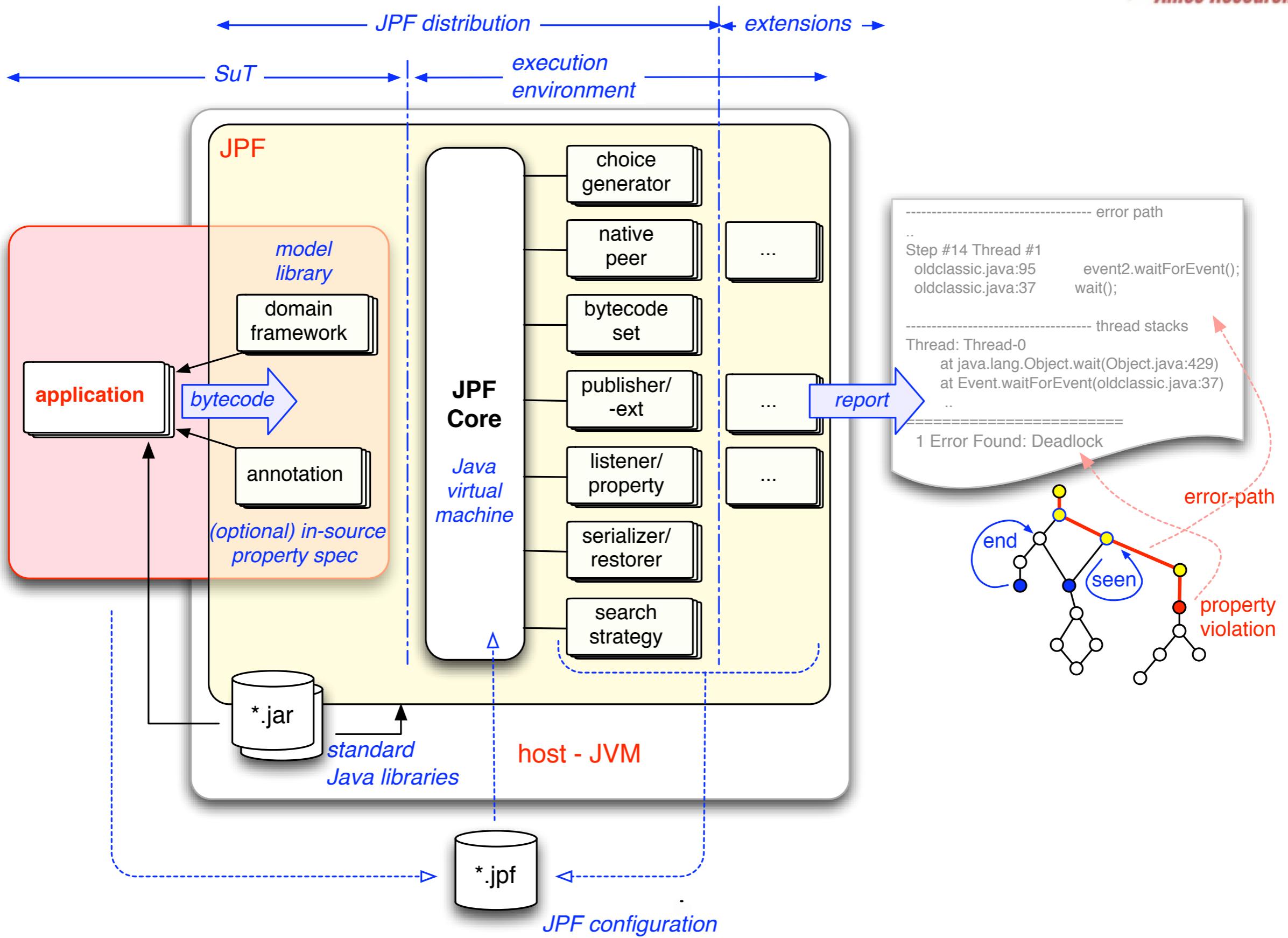
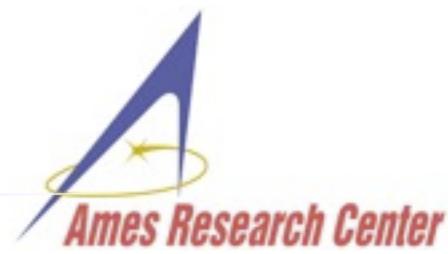


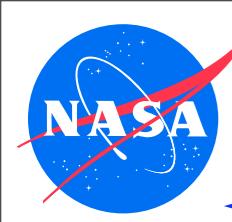
- verified Java program is executed by JPF, which is a virtual machine implemented in Java, i.e. runs on top of a host JVM
⇒ easy to get confused about who executes what





Basics: JPF Components

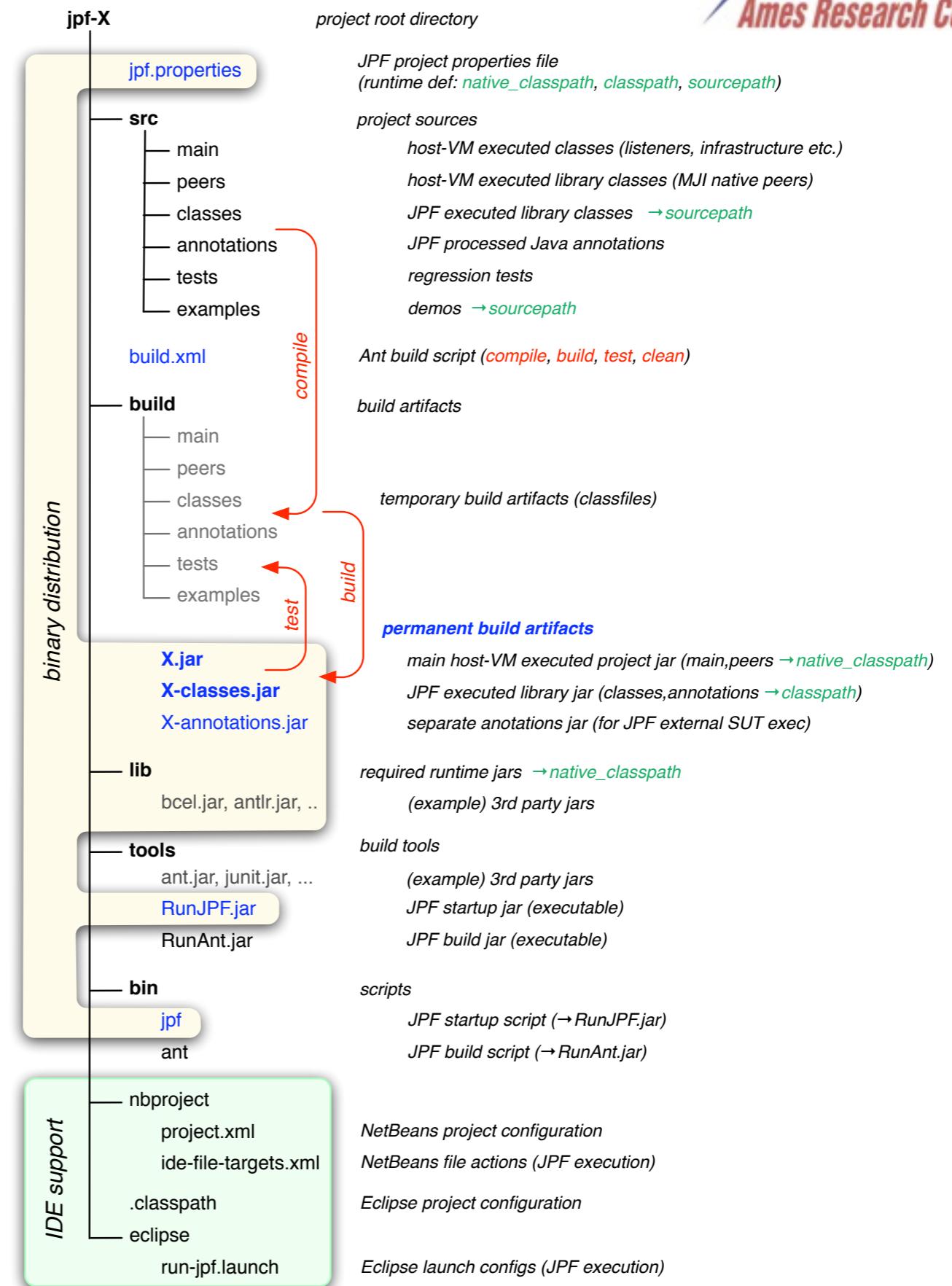


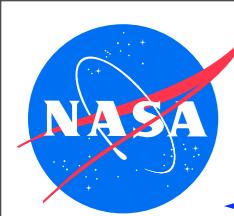


Basics: JPF Module/Project Structure



- ◆ all JPF projects share uniform directory layout
- ◆ binary distributions are slices of source distributions (interchangeable)
- ◆ 3rd party tools & libraries can be included (self-contained)
- ◆ all projects have examples and regression test suites (eventually ☺)
- ◆ projects have out-of-the-box IDE configuration (NB,Eclipse)

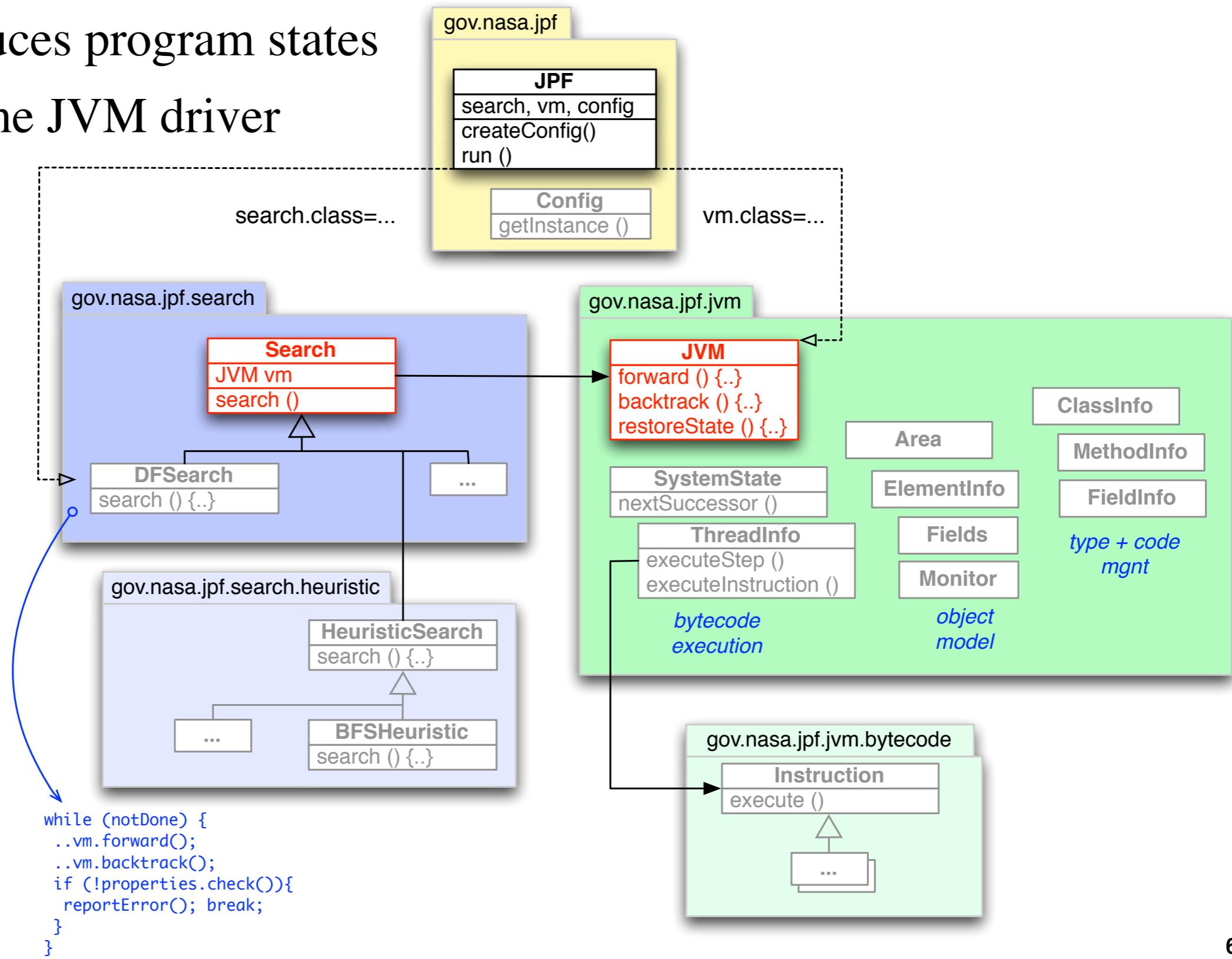


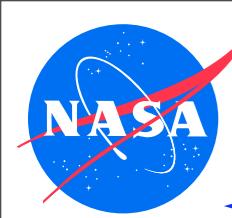


JPF Toplevel Structure



- ◆ two major constructs: **Search** and **JVM**
- ◆ JVM produces program states
- ◆ Search is the JVM driver



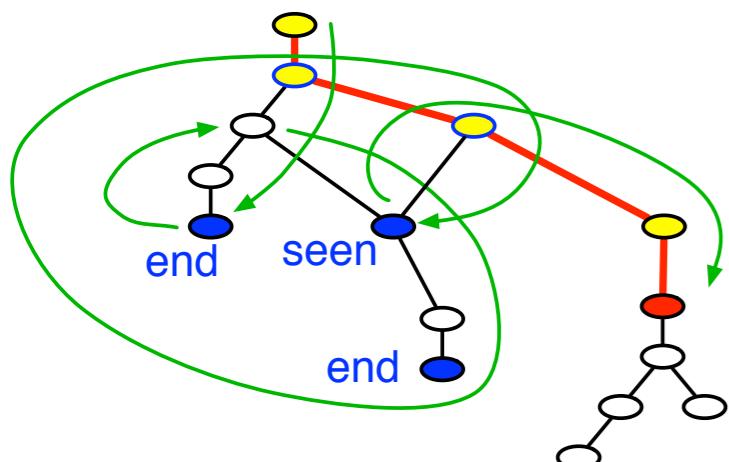


Search Policies

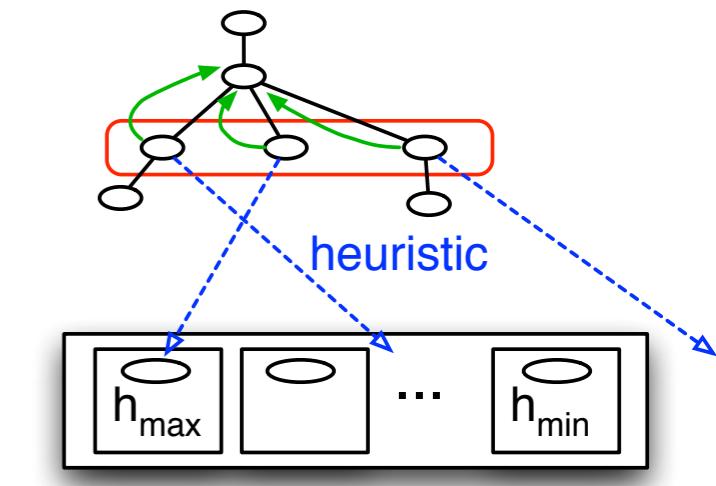
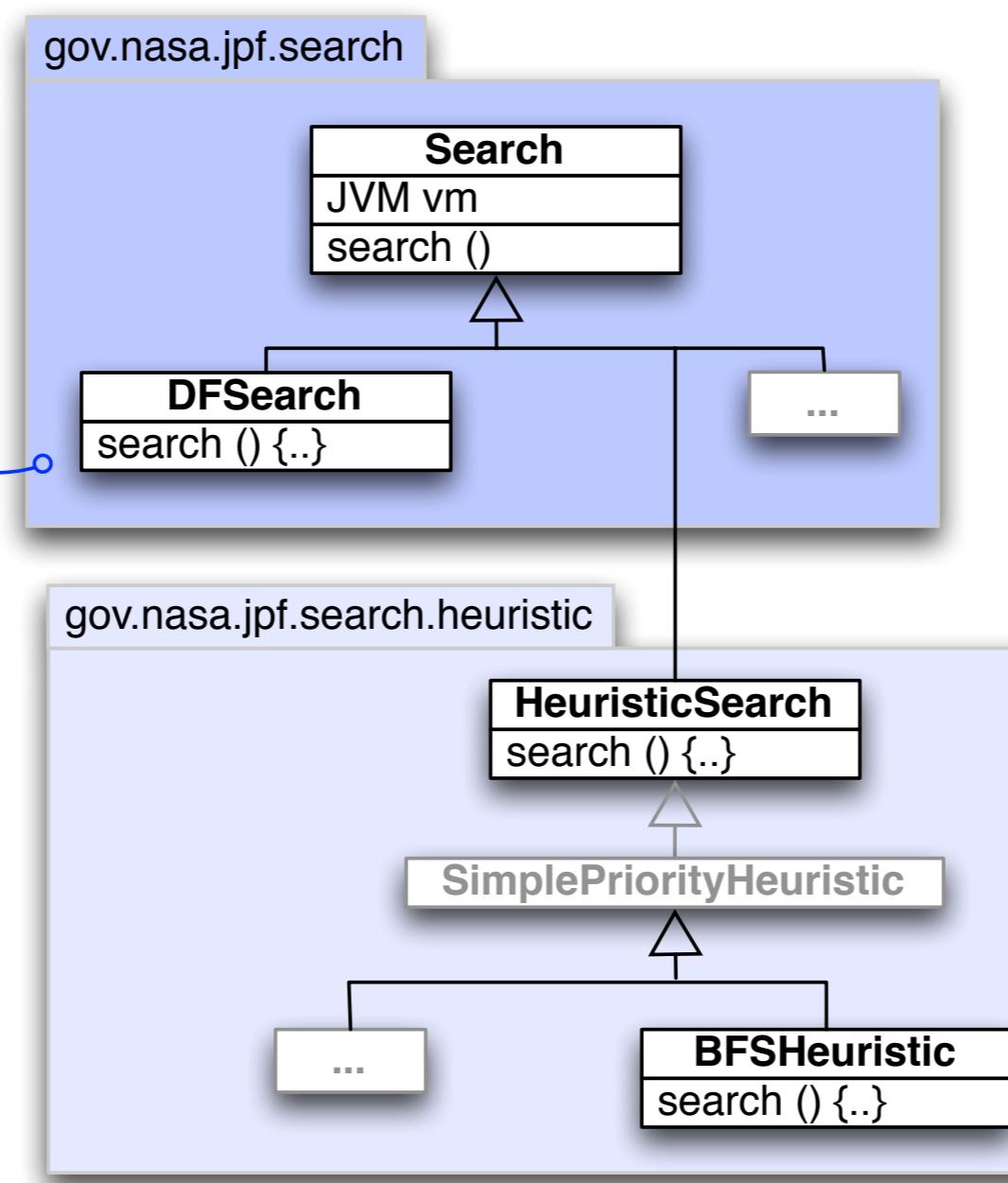


- ◆ state explosion mitigation: search the interesting state space part first (“get to the bug early, before running out of memory”)
- ◆ Search instances encapsulate (configurable) search policies

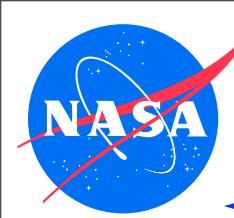
```
while (notDone) {  
    ...vm.forward();  
    ...vm.backtrack();  
    if (!properties.check()) {  
        reportError(); break;  
    }  
}
```



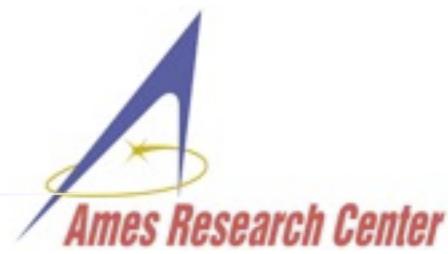
depth first traversal
optional state matching



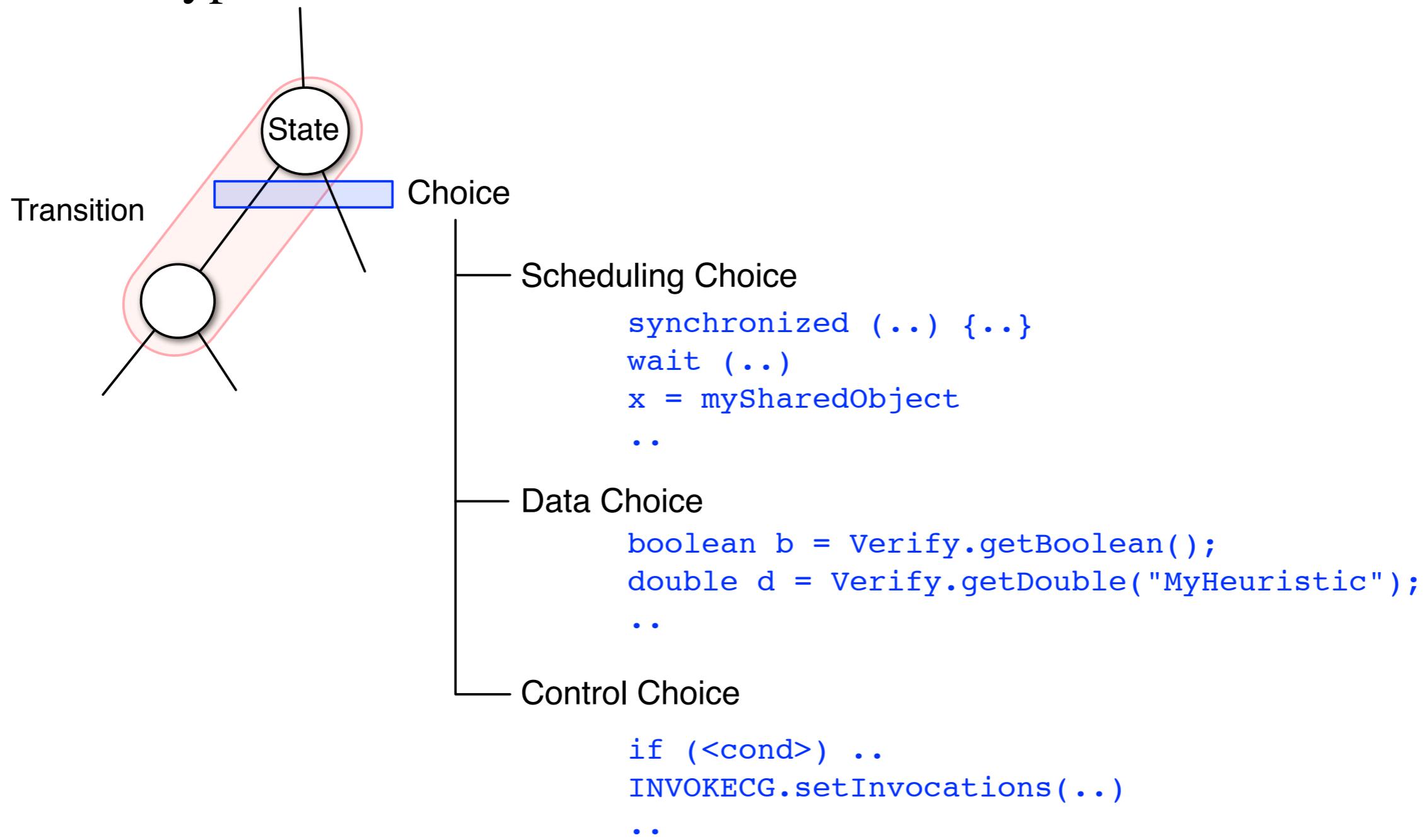
sorted state queue (bounded)

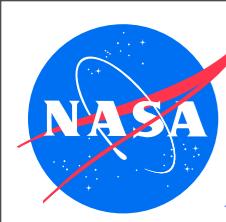


Choice Generators - Terminology



- ◆ model checker needs choices to explore state space
- ◆ there are many potential types of choices (scheduling, data, ..)
- ◆ choice types should not be hardwired in model checker

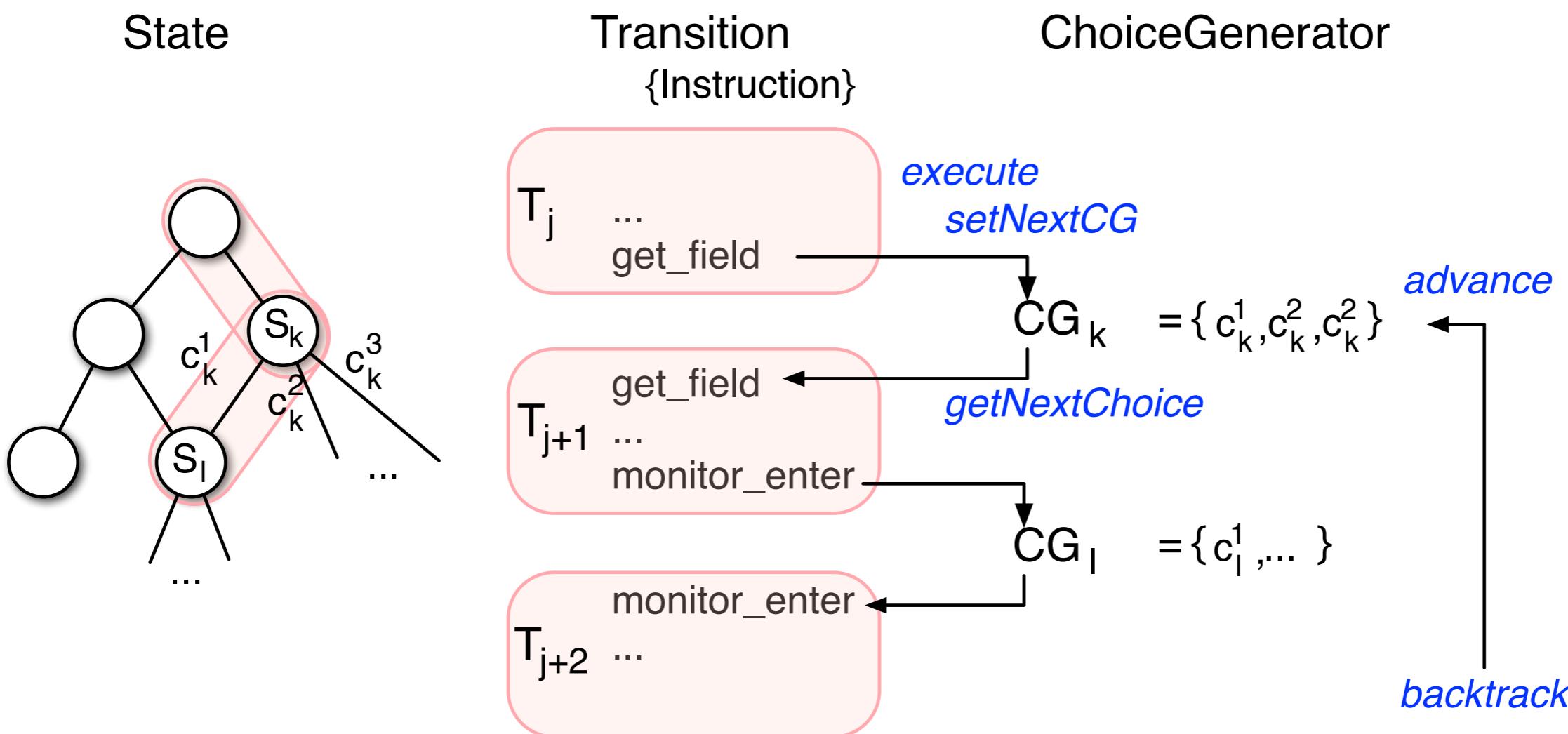


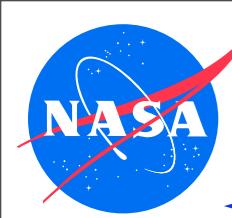


ChoiceGenerators - Transition Boundaries

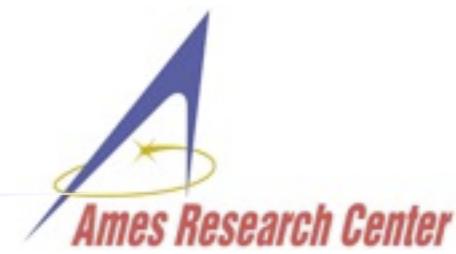


- ◆ transitions begin with a choice and extend until the next ChoiceGenerator (CG) is set (by instruction, native peer or listener)
- ◆ ‘advance’ positions the CG on the next unprocessed choice (if any)
- ◆ ‘backtrack’ goes up to the next CG with unprocessed choices



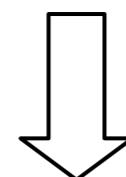


Configurable ChoiceGenerators - Why?



| | | |
|----------------------------|-----------------------|--|
| Verify.getBoolean() | C = { true, false } | ✓ |
| Verify.getInt(0, 4) | C = { 0, 1, 2, 3, 4 } | ? potentially large sets with lots of uninteresting values |
| Verify.getDouble(1.0, 1.5) | C = { ∞ } | ?? no finite value set without heuristics |

```
xChoiceGenerator
choiceSet: {x}
hasMoreChoices()
advance()
getNextChoice() → x
```



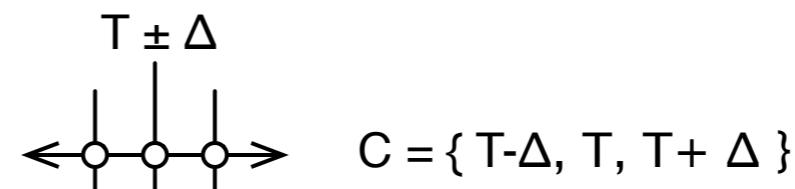
Choice Generators

JPF internal object to store and enumerate a set of choices

Configurable Heuristic Choice Models

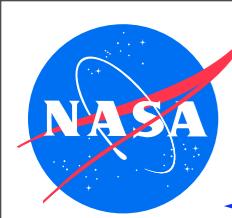
configurable classes to create ChoiceGenerator instances

e.g. "Threshold" heuristic
application code (test driver)



configuration (e.g. mode property file)

```
..  
double v = Verify.getDouble("velocity");  
..  
velocity.class = gov.nasa.jpf.jvm.choice.DoubleThresholdGenerator  
velocity.threshold = 13250  
velocity.delta = 500
```



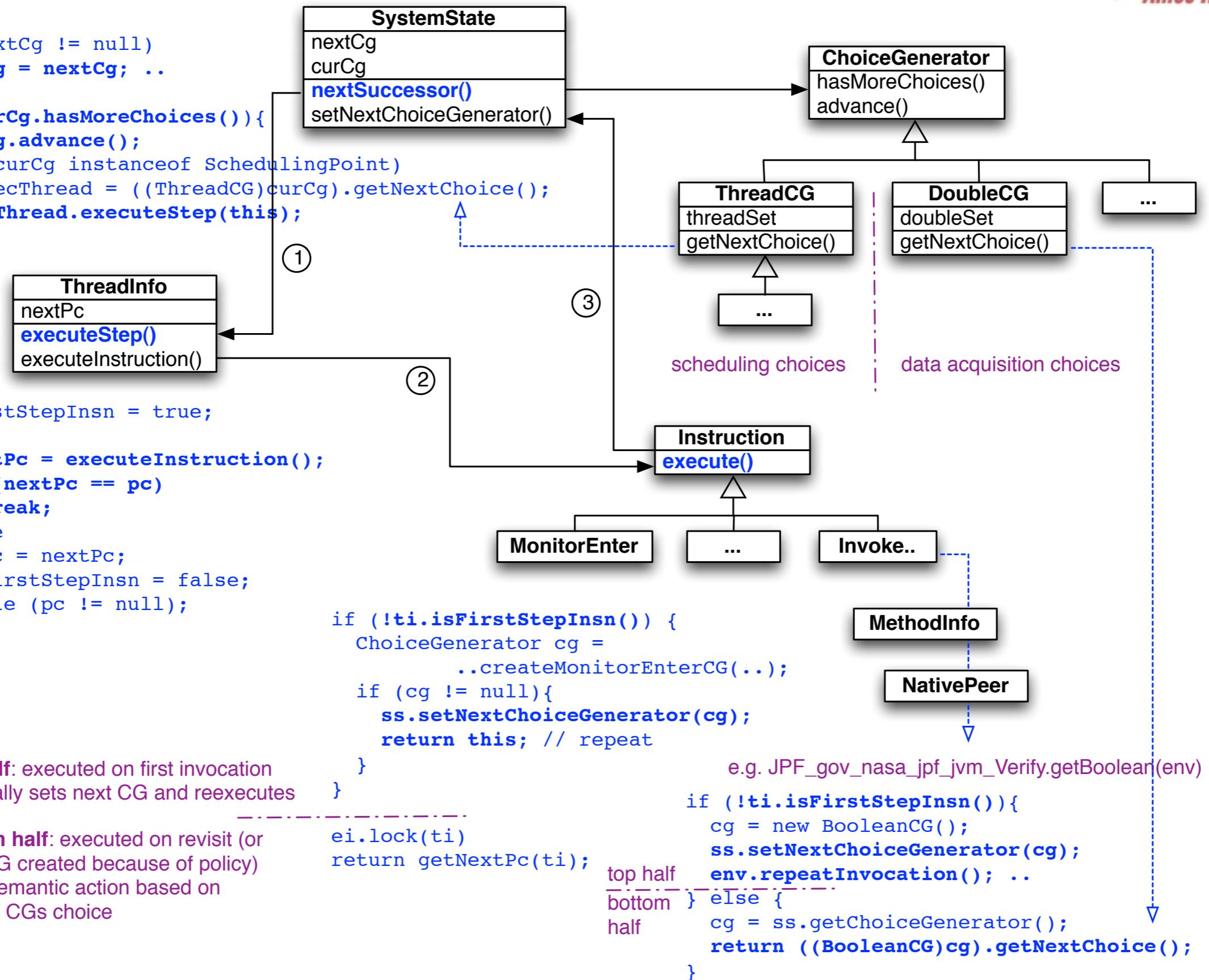
ChoiceGenerators - Implementation



```
if (nextCg != null)
    curCg = nextCg; ..

if (curCg.hasMoreChoices)
    curCg.advance();
    if (curCg instanceof Executable)
        execThread = ((Executable)curCg);
execThread.execute();

}
} ..
```



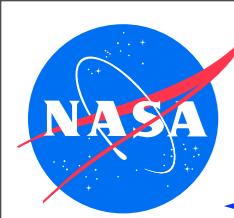
top half: executed on first invocation
optionally sets next CG and reexecutes

bottom half: executed on revisit (only if no CG created because of policy) does semantic action based on current CGs choice

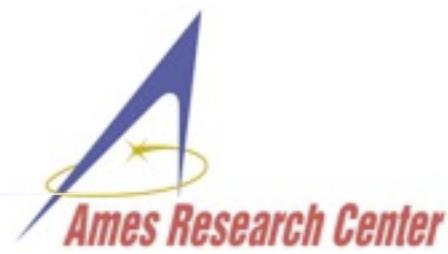
```
if (!ti.isFirstStepInsn()) {  
    ChoiceGenerator cg =  
        ..createMonitorEnterCG(...)  
    if (cg != null){  
        ss.setNextChoiceGenerator(cg);  
        return this; // repeat
```

```
e.g. JPF_gov_nasa_jpf_jvm_Verify.getBoolean(env)

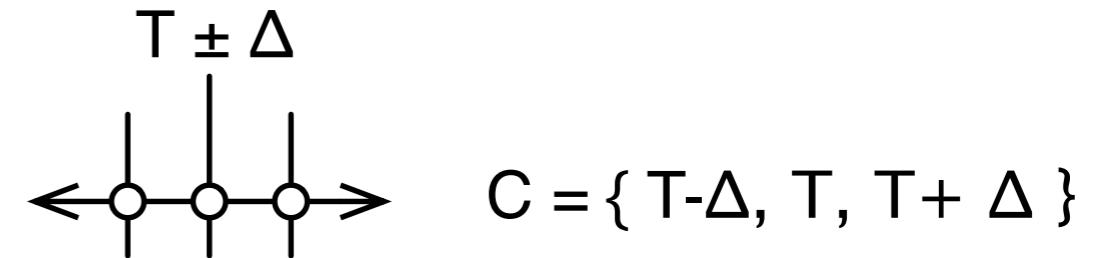
if (!ti.isFirstStepInsn()){
    cg = new BooleanCG();
    ss.setNextChoiceGenerator(cg);
    env.repeatInvocation(); ..
} else {
    cg = ss.getChoiceGenerator();
    return ((BooleanCG)cg).getNextChoice();
}
```

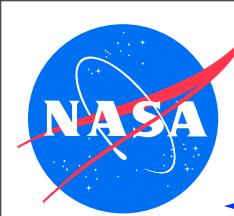


ChoiceGenerator - Example

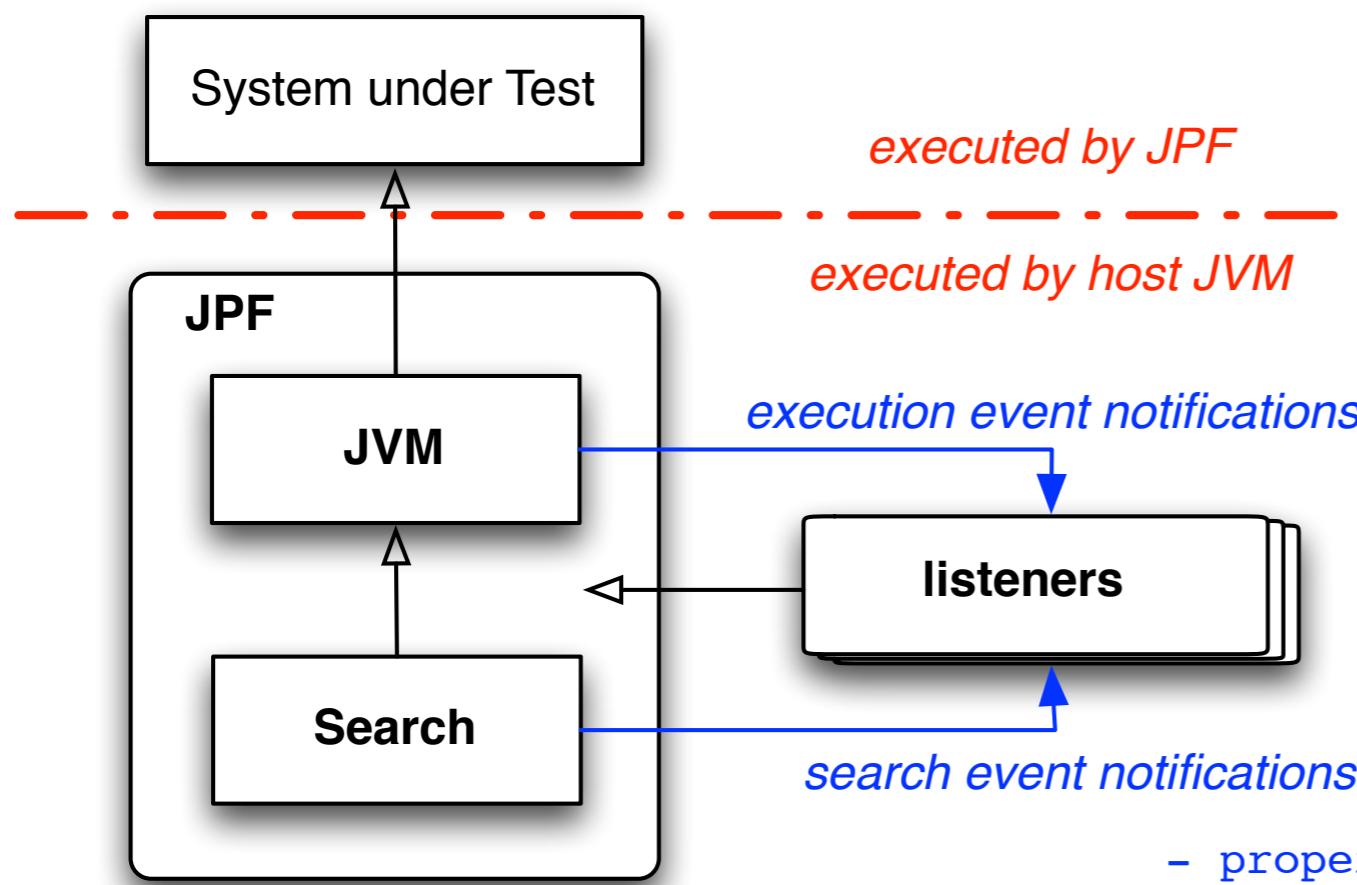
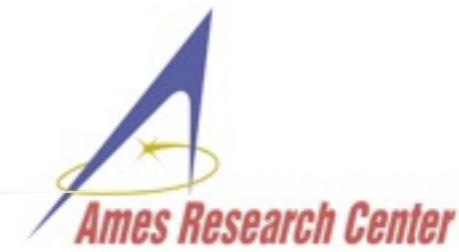


```
public class DoubleThresholdGenerator extends DoubleChoiceGenerator {  
    double[ ] values = new double[3];  
    int count;  
  
    public DoubleThresholdGenerator(Config conf, String id) {  
        super(id);  
        values[0] = conf.getDouble(id + ".low");  
        values[1] = conf.getDouble(id + ".threshold");  
        values[2] = conf.getDouble(id + ".high");  
        count = -1;  
    }  
  
    public boolean hasMoreChoices () {  
        return !isDone && (count < 2);  
    }  
  
    public Double getNextChoice () {  
        return (count >+ 0) ? new Double(values[count]) : new Double(values[0]);  
    }  
  
    public void advance () {  
        if (count < 2) count++;  
    }  
  
    public int getTotalNumberOfChoices () { return 3; }  
    public int getProcessedNumberOfChoices () { return count + 1; }  
    public void reset () { count = -1; }
```

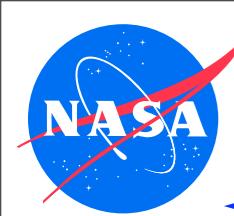




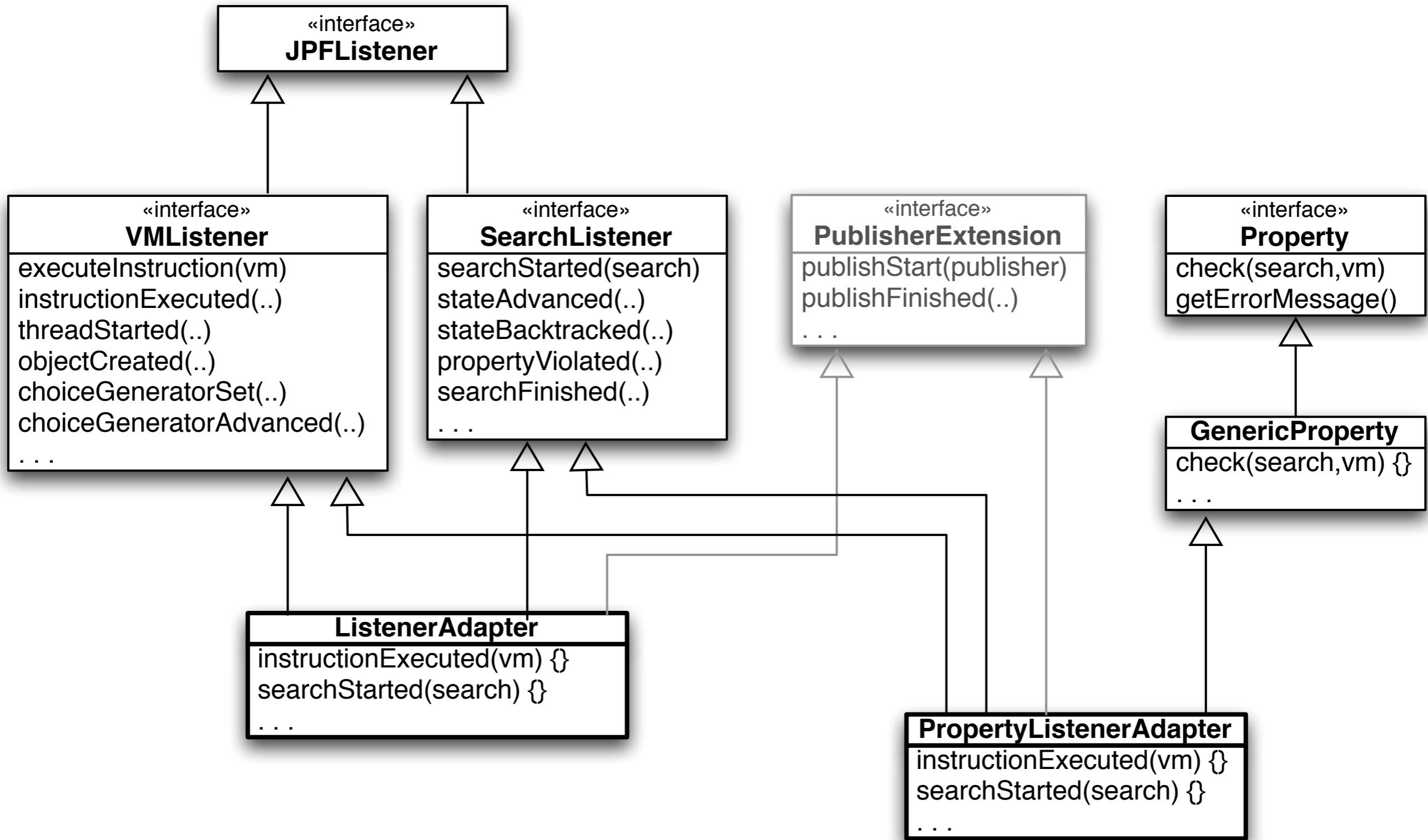
Listeners - the JPF plugins

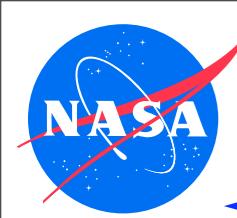


- classLoaded()
 - threadScheduled()
 - threadNotified()
 - ...
 - executeInstruction()
 - instructionExecuted()
 - objectCreated()
 - ...
 - exceptionThrown()
 - ...
 - choiceGeneratorAdvanced()
 - ...
- configured*
- +listener=<listener-class>
 - @JPFCConfig(..)
 - listener.autoload=<annotations>
 - jpf.addListener(..)
 - ...
- propertyViolated()
 - searchConstraintHit
 - searchFinished()
 - ...

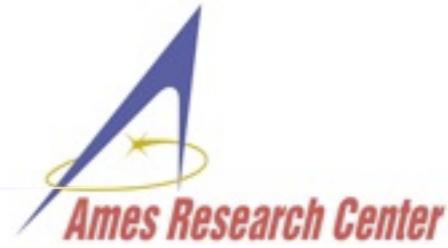


Listeners - Implementation

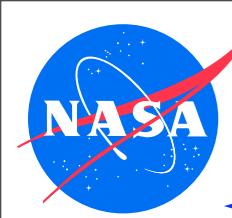




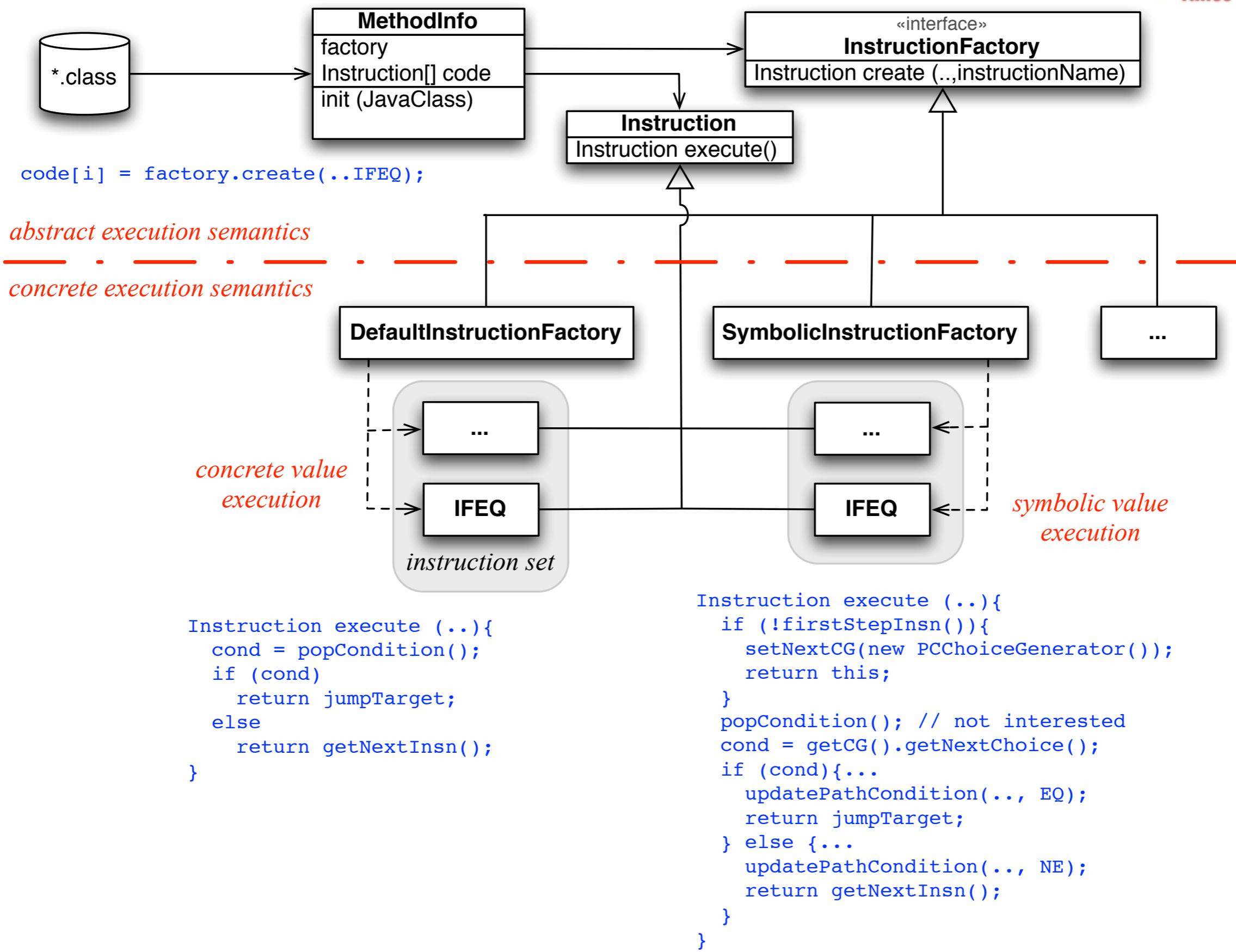
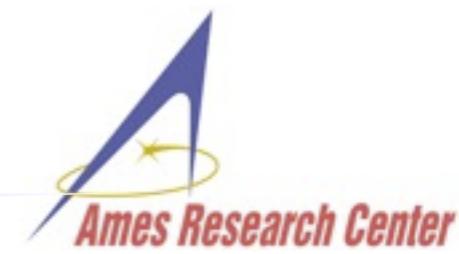
Listeners - Example

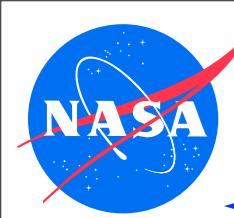


```
public class NonnullChecker extends ListenerAdapter {  
  
    ...  
  
    public void executeInstruction (JVM vm){  
  
        Instruction insn = vm.getLastInstruction();  
        ThreadInfo ti = vm.getLastThreadInfo();  
  
        if (insn instanceof ARETURN) { // check @NonNull method returns  
            ARETURN areturn = (ARETURN)insn;  
            MethodInfo mi = insn.getMethodInfo();  
  
            if (areturn.getReturnValue(ti) == null) {  
                if (mi.getAnnotation("javax.annotation.NonNull") != null) {  
                    Instruction nextPc =  
                        ti.createAndThrowException("java.lang.AssertionError",  
                            "null return from @NonNull method: " + mi.getCompleteName());  
                    ti.setNextPC(nextPc);  
                    return;  
                }  
            }  
        }  
  
        ...  
    }  
}
```

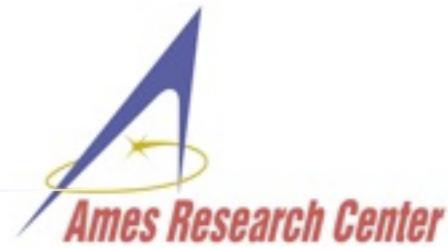


Bytecode Factories - Implementation





Bytecode Factories - Example



- ◆ provide alternative Instruction classes for relevant bytecodes
- ◆ create & configure InstructionFactory that instantiates them

JPF configuration

```
vmInsn_factory.class =  
    numeric.NumericInstructionFactory
```

class loading

```
class IADD extends Instruction {  
    Instruction execute (... ThreadInfo ti) {  
        int v1 = ti.pop();  
        int v2 = ti.pop();  
        int res = v1 + v2;  
  
        if ((v1>0 && v2>0 && res<=0) ...  
            throw ArithmeticException..
```

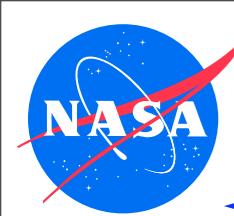


compiler

```
...  
[20] iinc  
[21] goto 10  
[10] iload_4  
[11] bipush  
[12] if_icmpge 22  
[13] iload_3  
[14] iload_2  
[15] iadd  
...
```

```
void notSoObvious(int x){  
    int a = x*50;  
    int b = 19437583;  
    int c = a;  
  
    for (int k=0; k<100; k++){  
        c += b;  
        System.out.println(c);  
    }  
}  
...  
notSoObvious( 21474836);
```

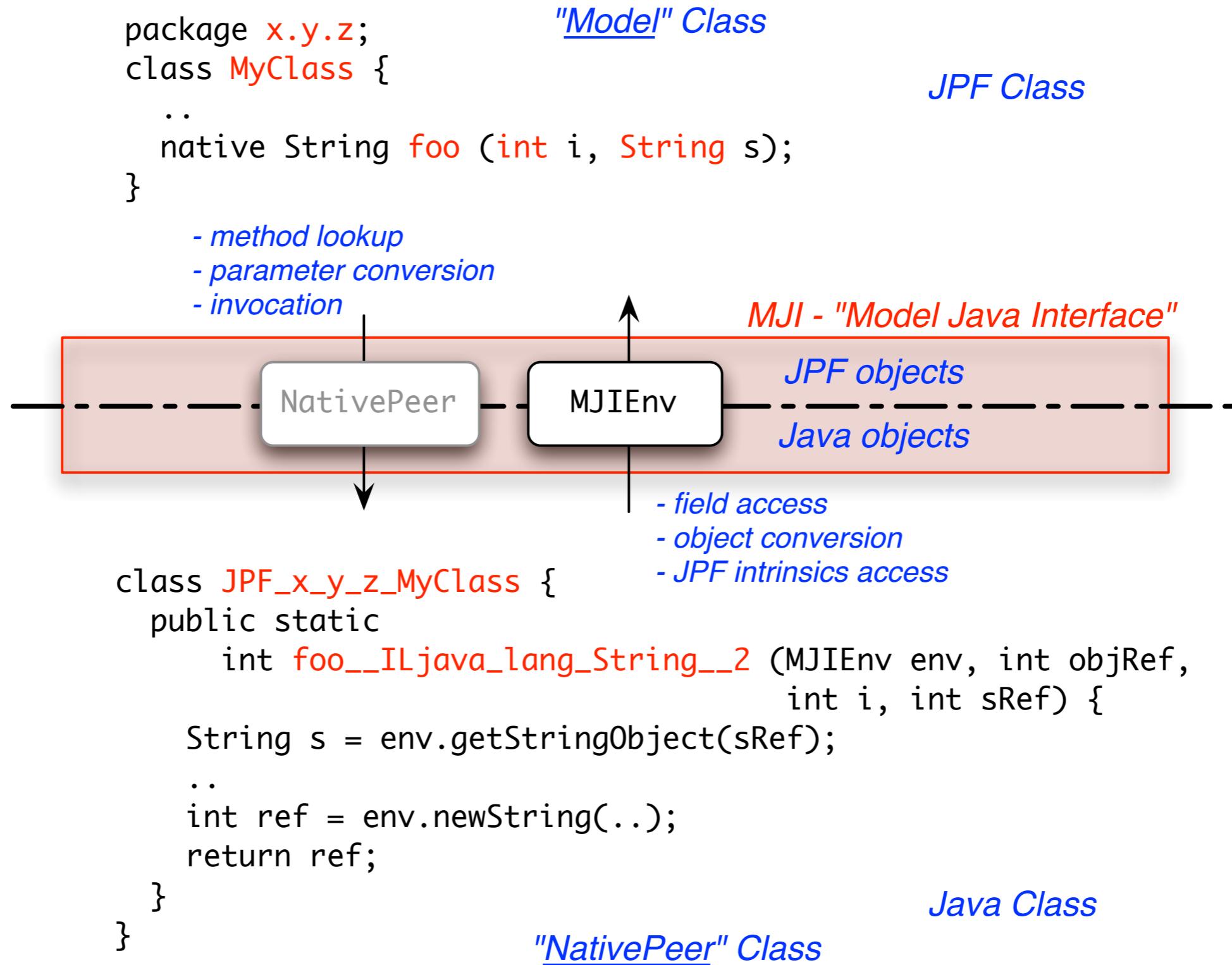
code execution
(by JPF)

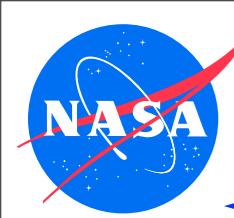


MJI - Model-Java-Interface

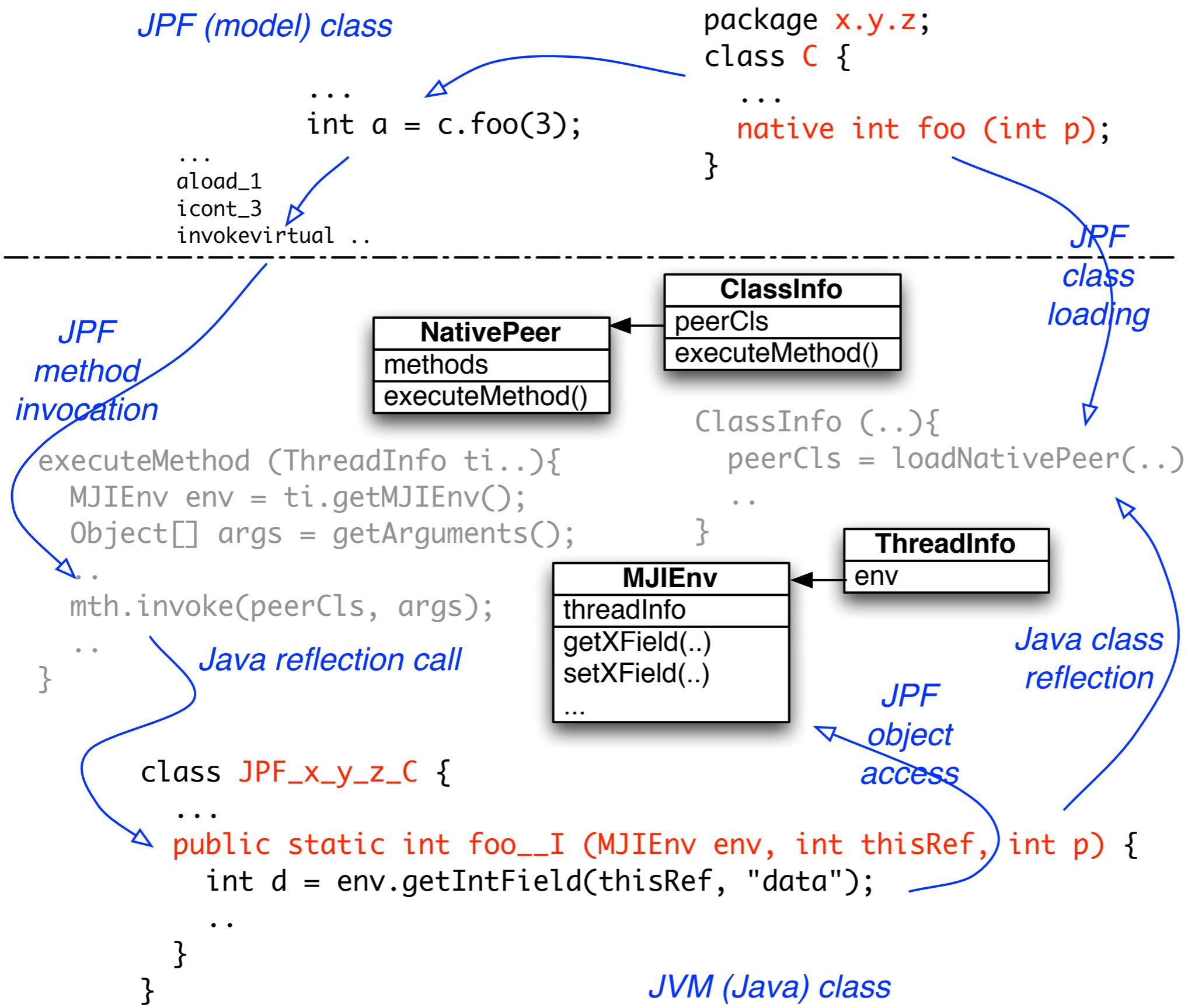


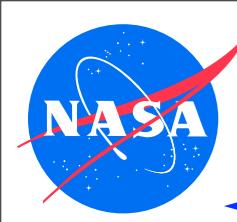
- ◆ execution lowering from JPF executed code into JVM executed code



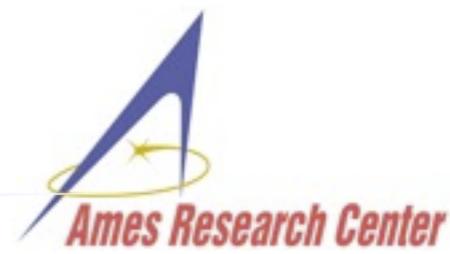


MJI - Implementation

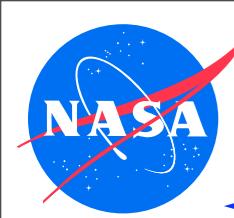




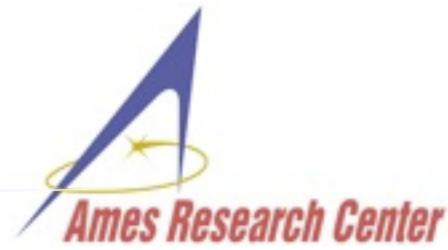
MJI - Example



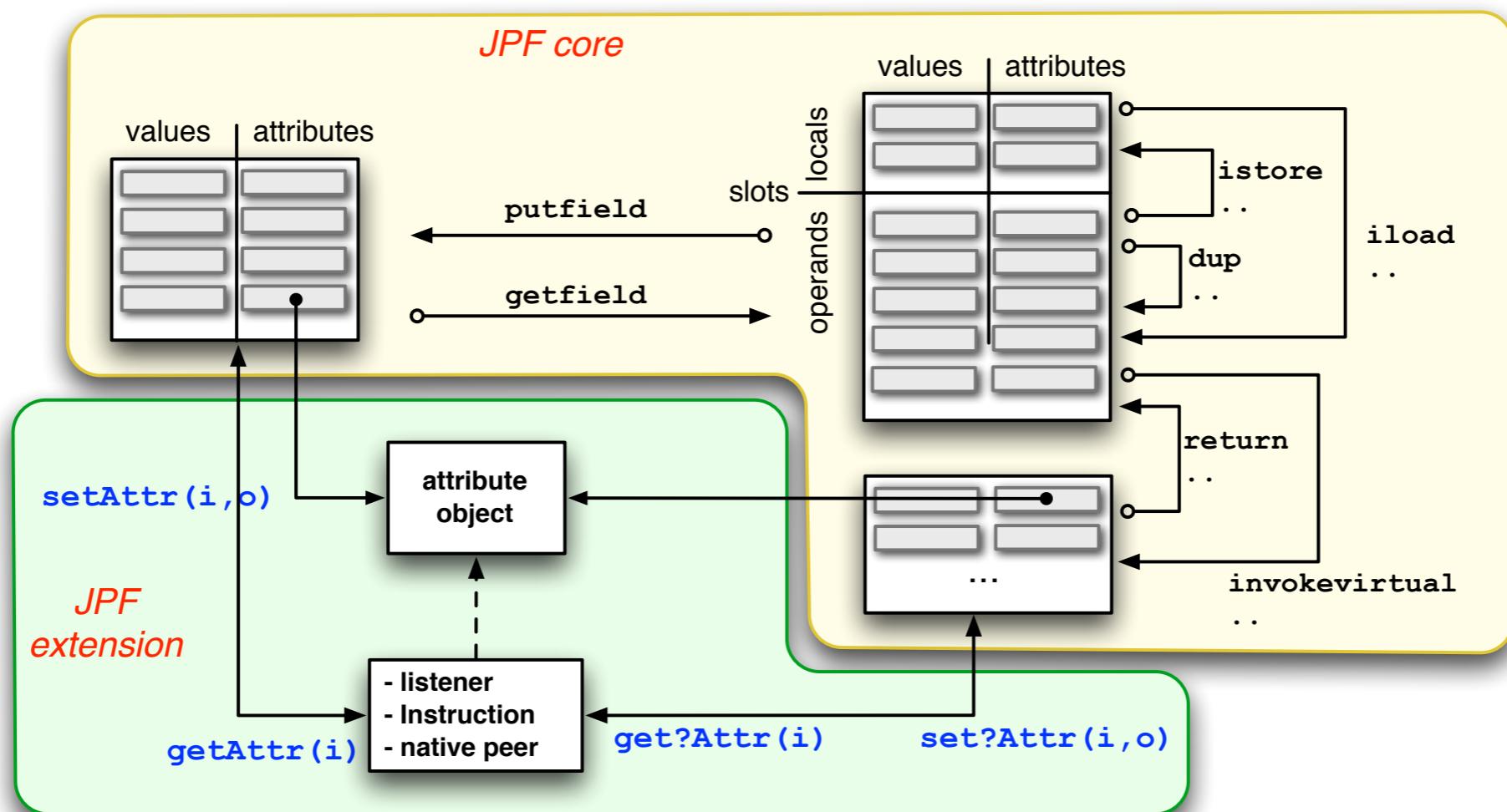
```
public class JPF_java_lang_String {  
    ...  
    public static int indexOf__I__I (MJIEnv env, int objref, int c) {  
        int vref = env.getReferenceField(objref, "value");  
        int off = env.getIntField(objref, "offset");  
        int len = env.getIntField(objref, "count");  
  
        for (int i=0, j=off; i<len; i++, j++) {  
            if ((int)env.getCharArrayElement(vref, j) == c) return i;  
        }  
        return -1;  
    }  
  
    public static int toCharArray____3C (MJIEnv env, int objref){  
        ...  
        int cref = env.newCharArray(len);  
        for (int i=0, j=off; i<len; i++, j++) {  
            env.setCharArrayElement(cref, i, env.getCharArrayElement(vref, j));  
        }  
        return cref;  
    }  
  
    public static boolean matches__Ljava_lang_String_2__Z (MJIEnv env, int objRef,  
                                                       int regexRef){  
        String s = env.getStringObject(objRef);  
        String r = env.getStringObject(regexRef);  
        return s.matches(r);  
    }  
}
```

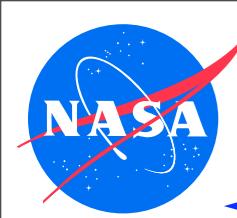


Attributes - Implementation

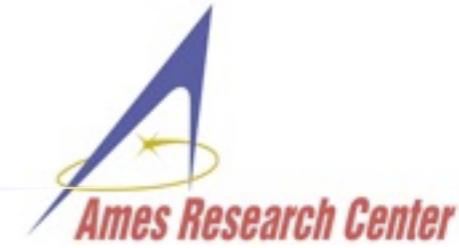


- user defined objects that (automatically) travel with data

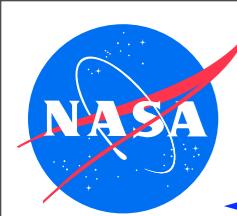




Attributes - Example



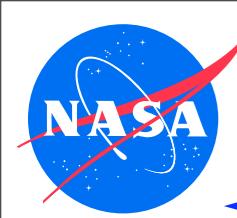
```
public class NumericAttr {  
    Apffloat value;  
  
    public NumericAttr subtract (NumericAttr x){  
        return new NumericAttr( value.subtract( x.value));  
    }  
    ...  
  
    public class DSUB extends gov.nasa.jpf.jvm.bytecode.DSUB {  
        ...  
        public Instruction execute (SystemState ss, KernelState ks, ThreadInfo ti) {  
            NumericAttr a1 = ti.getLongOperandAttr(NumericAttr.class);  
            double v1 = Types.longToDouble(ti.longPop());  
            NumericAttr a2 = ..  
            double v2 = ..  
            double r = v2 - v1;  
            NumericAttr ar = a2.subtract(a1);  
  
            ...  
            if ((error = checkAttr(r,ar)) != null){  
                return throwException(ti,error);  
            }  
  
            ti.longPush(Types.doubleToLong(r));  
            ti.setLongOperandAttrNoClone(ar);  
  
            return getNext(ti);  
        }  
        ...  
    }
```



Conclusions



- ◆ check out <http://babelfish.arc.nasa.gov/trac/jpf>
- ◆ all answers will be there (eventually)
- ◆ .. if not - try <http://groups.google.com/group/java-pathfinder>
- ◆ if not - we are here to help: Peter.C.Mehlitz@nasa.gov



Conclusions



- ◆ check out <http://babelfish.arc.nasa.gov/trac/jpf>
- ◆ all answers will be there (eventually)
- ◆ .. if not - try <http://groups.google.com/group/java-pathfinder>
- ◆ if not - we are here to help: Peter.C.Mehlitz@nasa.gov

Thank You!