JPF Tutorial
ASE 2013

Neha Rungta
SGT / NASA Ames Research Center
<neha.s.rungta@nasa.gov>

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

Willem Visser
University of Stellenbosch
<wvisser@cs.sun.ac.za>

JPF-Tutorial: Where Are the Slides?

- for now: http://babelfish.arc.nasa.gov/trac/jpf/raw-attachment/wiki/presentations/start/ASE13-Tutorial.pdf

- or go to http://babelfish.arc.nasa.gov/trac/jpf/wiki/presentations/start, scroll down to the "Attachments" section, expand and *click on the download icon trailing the filename*

- might get its own "tutorial" wiki section in the future

JPF Tutorial: Roadmap

- Basics
  - what is JPF
  - core concepts and extension mechanisms
  - where to look for more info

2h

- Install and run JPF
  - step-by-step guided tour of installing jpf-core
  - running core examples
  - [extension examples - what else can be done with JPF]

2h

~ lunch break ~

- Extending JPF
  - RangeChecker - safety property *listener*
  - [StopWatch - control flow variation *listener* using data value attributes]
  - symbolic execution

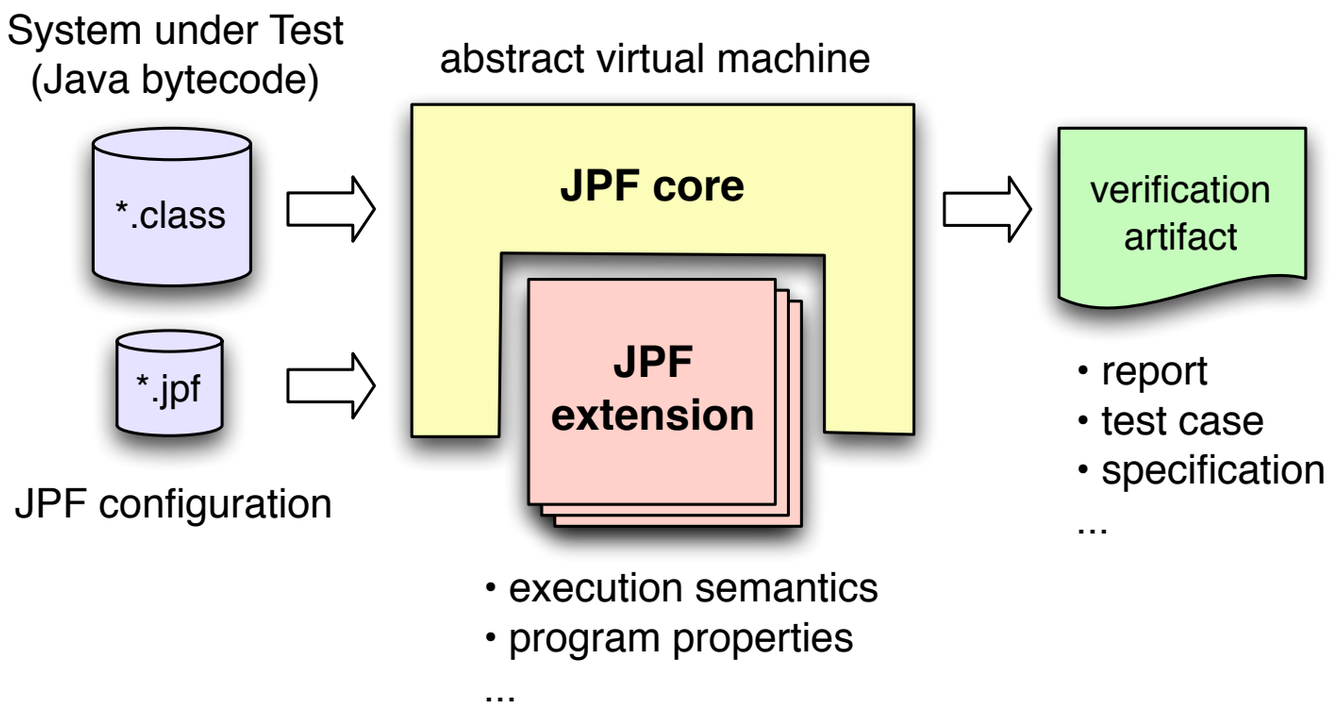
3h

Tutorial Part 1: Basics

- the inevitable “What is JPF”
- toplevel design
- main concepts and extension points:
  - ChoiceGenerators
  - Listeners
  - Attributes
  - MJI and NativePeers
  - InstructionFactories
  - Publishers
- JPF configuration
- JPF project layout
- Where to get help

Basics: JPF - What is it?

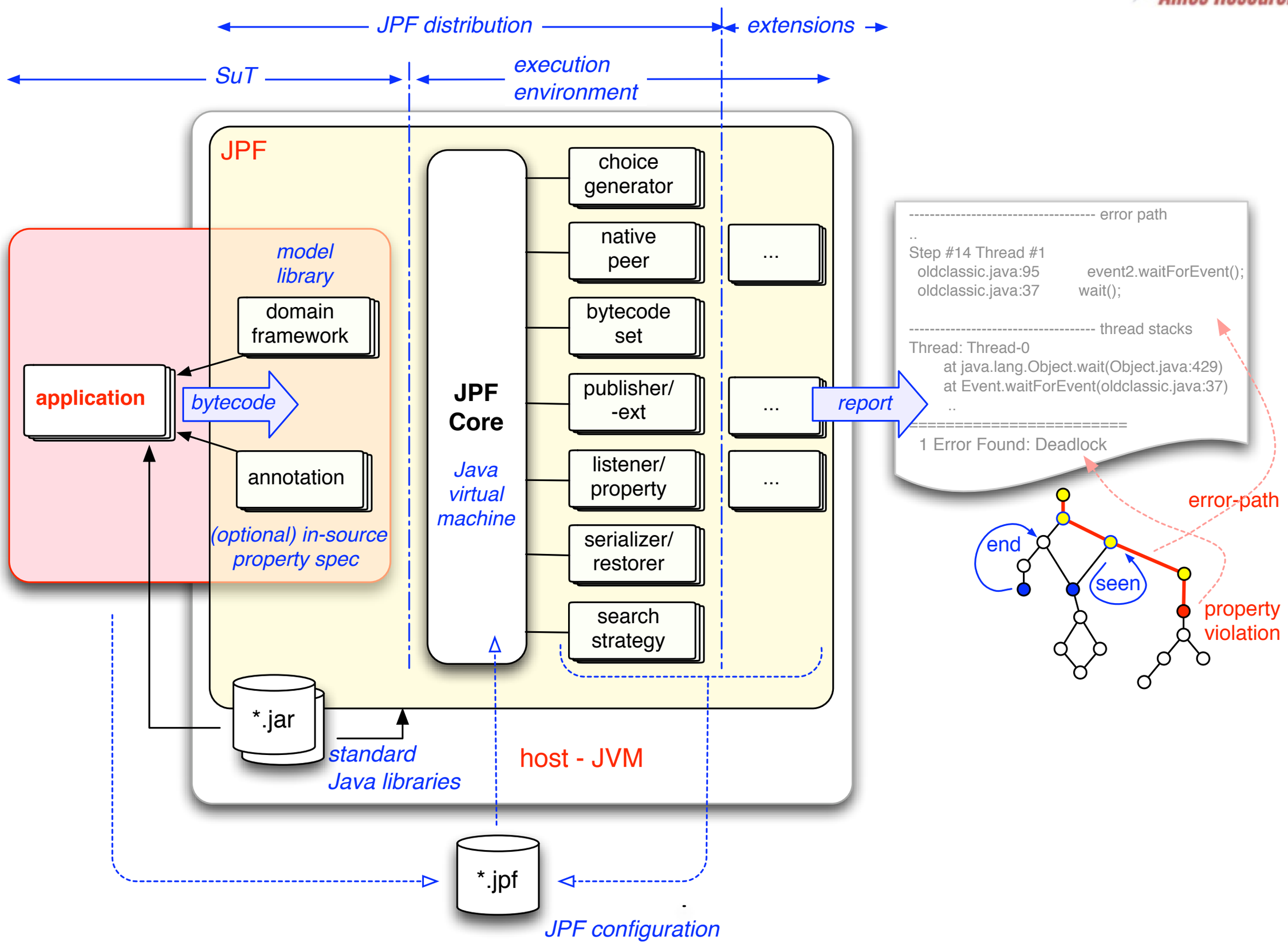
- a model checker? a virtual machine? ..



- ..and the answer is: both, and more - it depends on configuration
- *not* a monolithic, black box tool

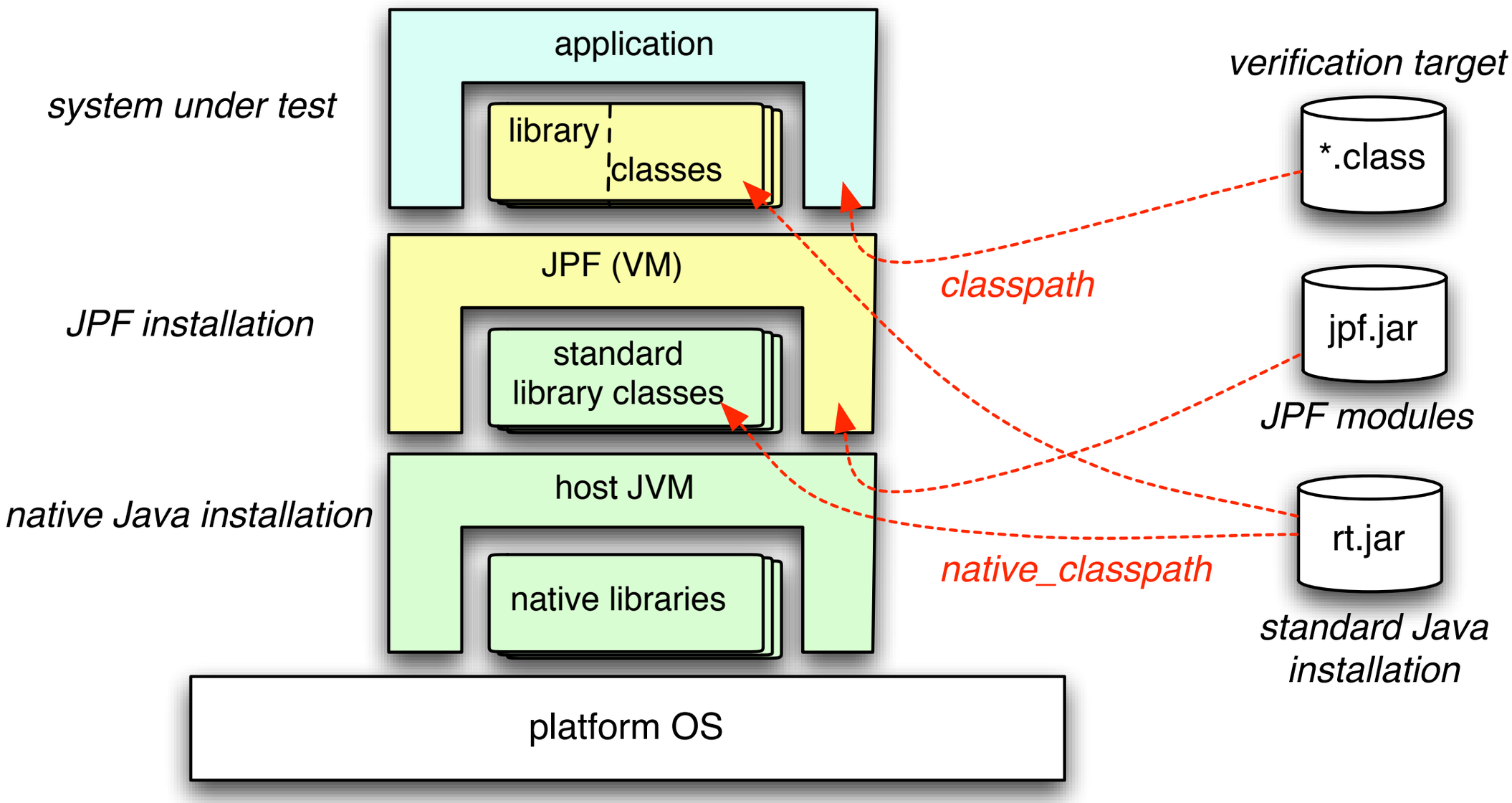
No “one size fits all” - Extensibility is Paramount

Basics: JPF Components

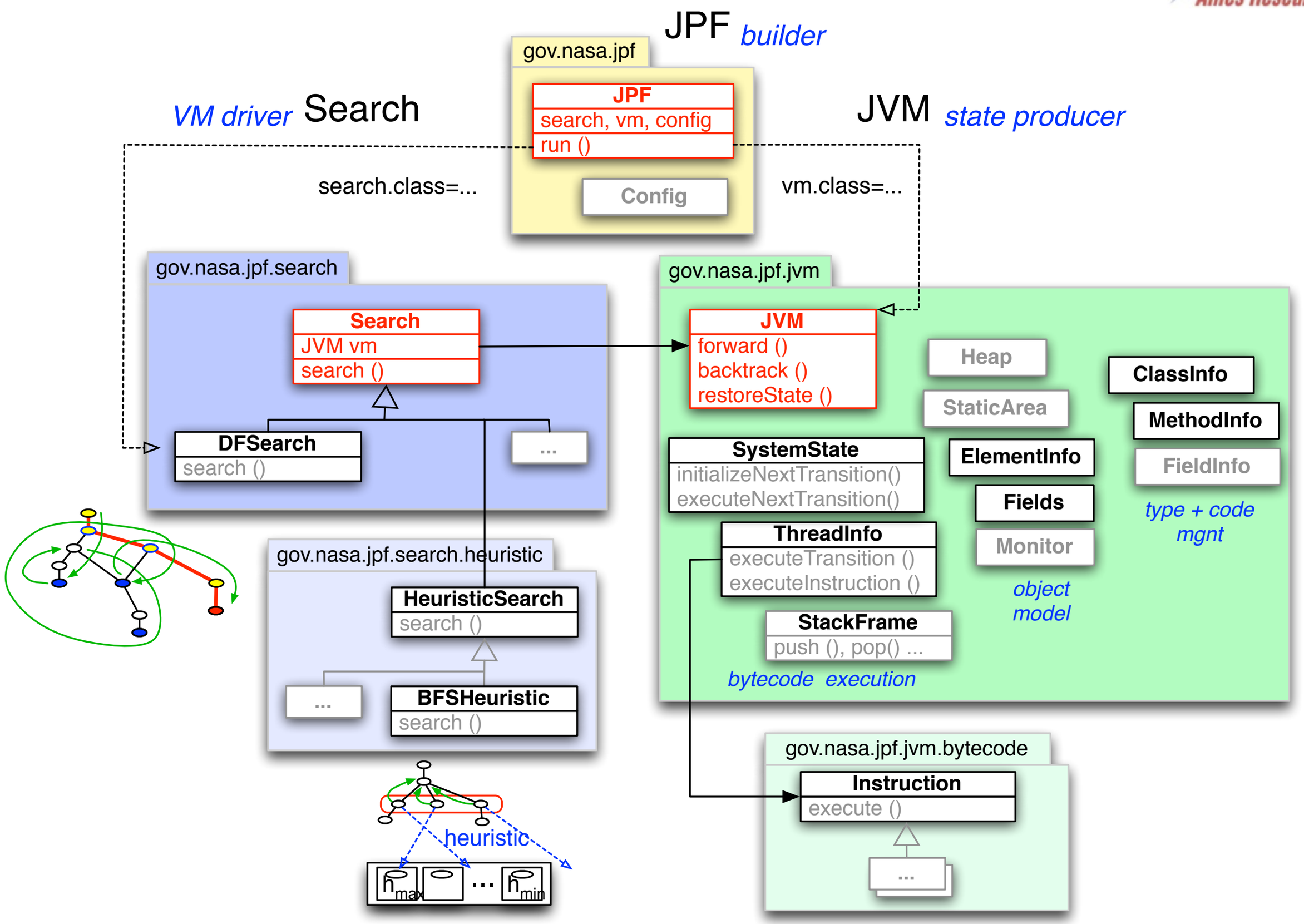


Basics: a VM running inside a VM

- main stumbling block is recursive nature of JPF
- verified Java program is executed by JPF, which is a virtual machine implemented in Java, i.e. runs on top of a host JVM
- need for two separate classpaths (host VM and JPF)



Basics: JPF Toplevel Design

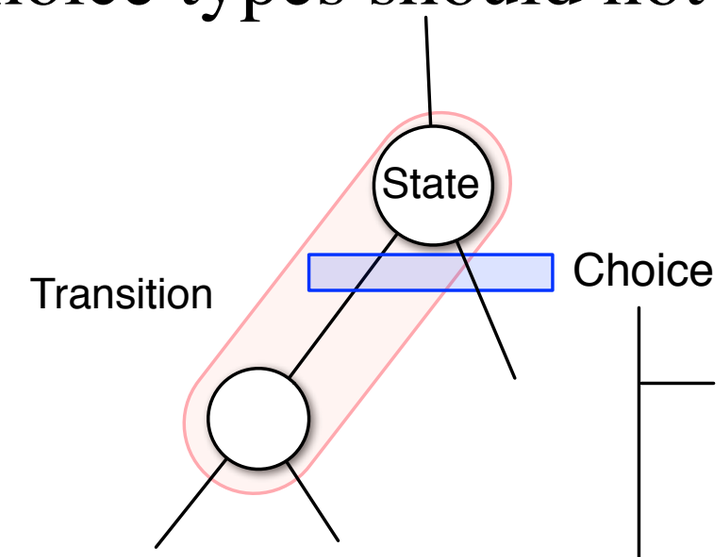


Basics: Concepts and Extension Mechanisms

- ChoiceGenerators - Transitions and States
- VM and Search Listeners - the JPF "plugins"
- Object Attributes - data flow
- Native Peers - host VM delegation
- Instruction Factories - execution semantics
- Loggers and Publishers - output generation
- ... and many more (Searches, Serializers etc.)

Basics: Choice Generators

- model checker needs choices to explore state space
- there are many potential types of choices (scheduling, data, ..)
- some types can't be automatically enumerated (e.g. double)
- choice types should not be hardwired in model checker



Scheduling Choice

```
synchronized (..) {..}
wait (..)
x = mySharedObject
..
```

Data Choice

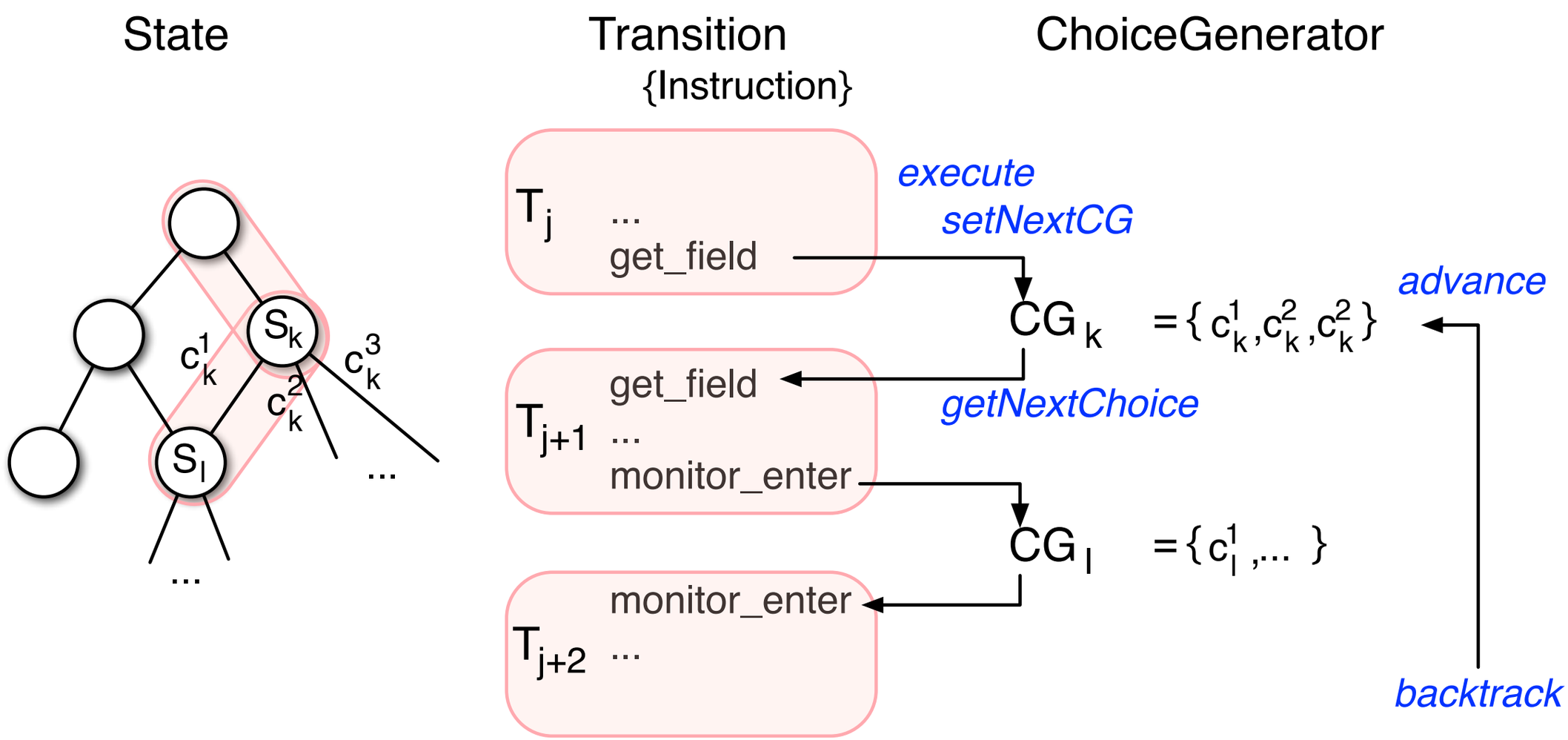
```
boolean b = Verify.getBoolean();
double d = Verify.getDouble("MyHeuristic");
..
```

Control Choice

```
if (<cond>) ..
INVOKECG.setInvocations(..)
..
```

Basics: ChoiceGenerators, States, Transitions

- 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



Basics: ChoiceGenerator Interface

- enumerator object for type specific choice values

```
public class DoubleThresholdGenerator extends DoubleChoiceGenerator {
  ...

  public boolean hasMoreChoices () {
    // are we done yet
    ...
  }

  public void advance () {
    // position on next choice
    ...
  }

  public Double getNextChoice () {
    // return choice value at current position
  }

  ...
```

Basics: Configurable ChoiceGenerators

`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

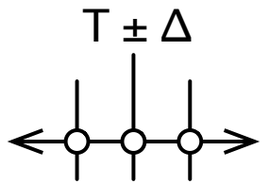
xChoiceGenerator
choiceSet: {x}
hasMoreChoices() advance() getNextChoice() → x

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



C = { T-Δ, T, T+ Δ }

application code (test driver)

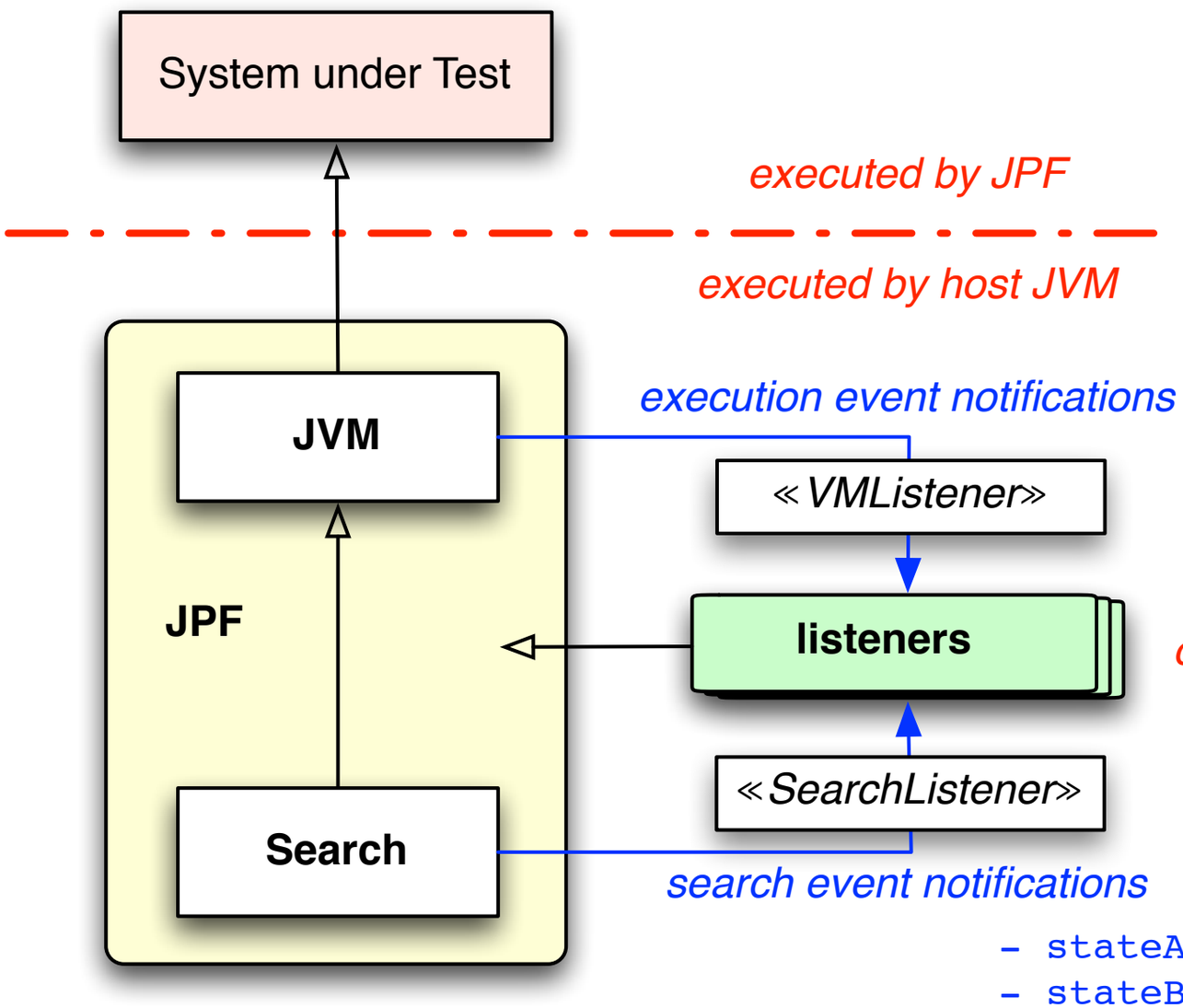
```
..
double v = Verify.getDouble("velocity");
..
```

configuration (e.g. mode property file)

```
velocity.class = gov.nasa.jpf.jvm.choice.DoubleThresholdGenerator
velocity.threshold = 13250
velocity.delta = 500
```

Basics: Listeners - the JPF plugins

- executed at host VM level, can access all exported JPF features
- includes Search and JVM events, both high- and low-level
- dynamically configured at runtime



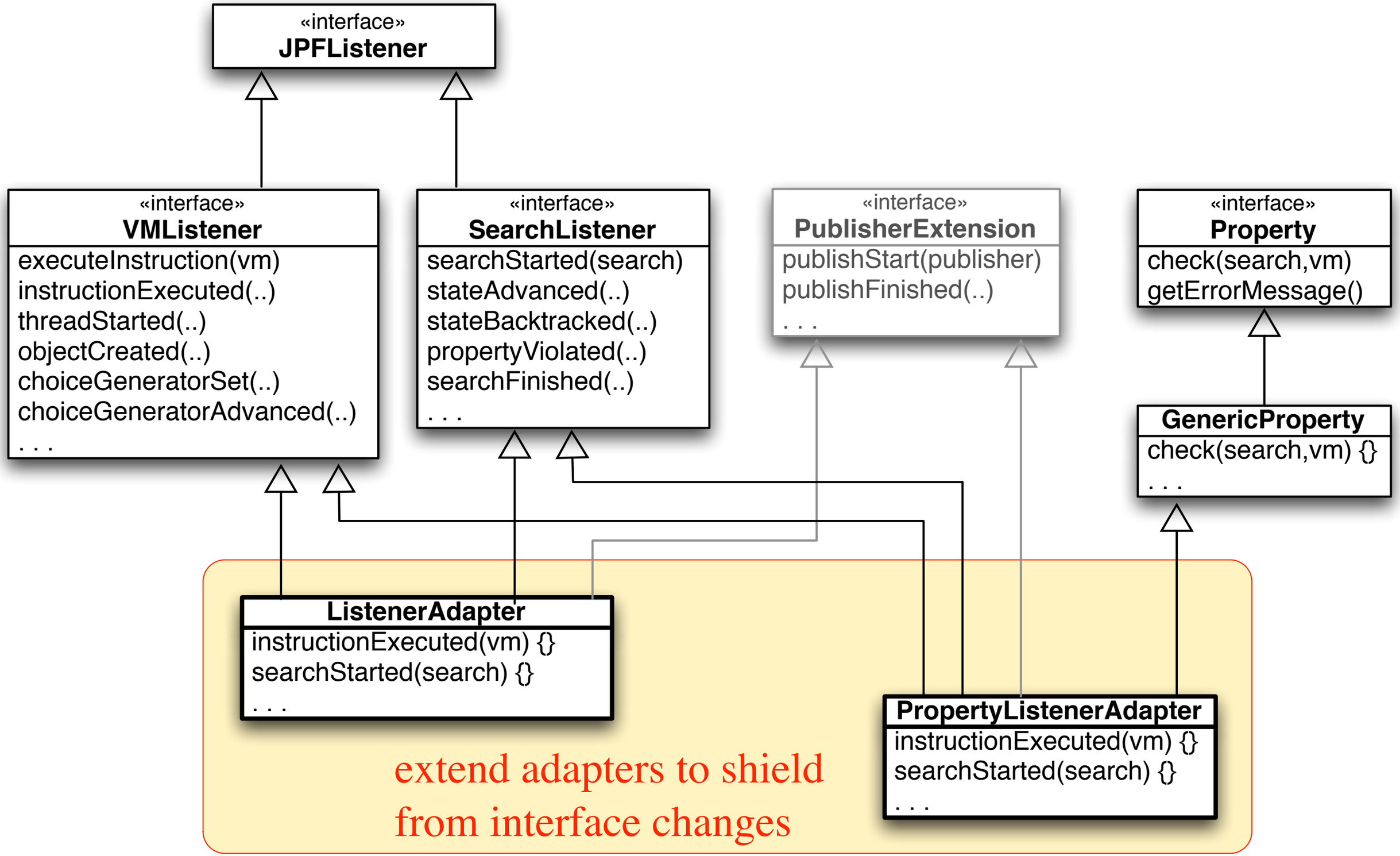

```
- classLoaded (vm)
- threadScheduled (vm)
- threadNotified (vm)
  ...
- executeInstruction (vm)
- instructionExecuted (vm)
  ...
- objectCreated (vm)
  ...
- exceptionThrown(vm)
  ...
- choiceGeneratorAdvanced (vm)
  ...
```

configured

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

```
- stateAdvanced (search)
- stateBacktracked (search)
- propertyViolated (search)
- searchFinished (search)
  ...
```

Basics: Listener Infrastructure



Basics: Listener Code

- extend `gov.nasa.jpf.ListenerAdapter` instead of directly implementing `..Listener`
- override the notifications you are interested in

```
public class NonnullChecker extends ListenerAdapter {

  ...
  @Override
  public void executeInstruction (VM vm,
      ThreadInfo curThread, Instruction insnToExecute){
  ...
```

override the notifications you want to handle

JPF context as arguments

Basics: Listener Example

- what to do in a listener? Anything that is visible to the listener package
- example: detect null value returns from `@Nonnull` annotated methods (that are supposed to return objects)

```
public class NonnullChecker extends ListenerAdapter {
  ...
  @Override
  public void executeInstruction (JVM vm, ThreadInfo ti, Instruction insn) {

    if (insn instanceof ARETURN) {
      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;
        }
      }
    ...
```

Basics: Listener Patterns

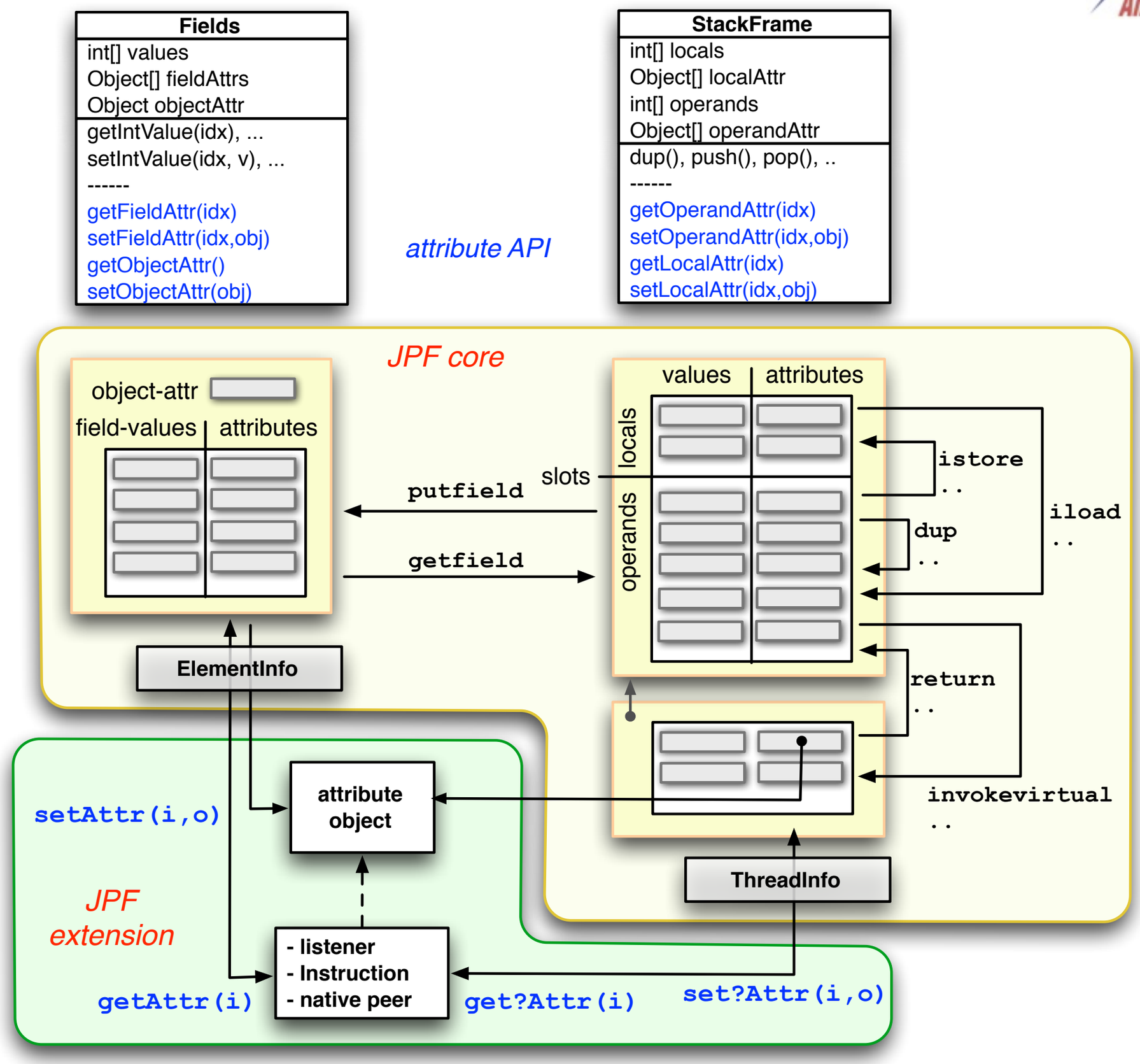
- cascaded `instanceof` bad if many alternatives, use Visitor pattern

```
public class MyListener extends gov.nasa.jpf.ListenerAdapter {
  class InsnDispatcher
     extends gov.nasa.jpf.jvm.bytecode.InstructionVisitorAdapter {
    ...
    void visit (ARETURN insn){
      .. if (areturn.getReturnValue(ti) ..
    }
  }
  ...
  VM vm;
  ThreadInfo ti;
  InsnDispatcher dispatcher;

  public MyListener(Config conf){ .. dispatcher = new InsnDispatcher(); ..}

  public void executeInstruction (VM vm, ThreadInfo ti, Instruction insn){
    this.vm = vm;
    this.ti = ti;
    vm.getLastInstruction().accept(dispatcher);
  }
}
```

Basics: Object Attributes

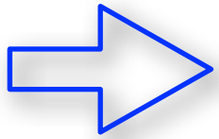
- user defined (host-VM) objects that can be attached to JPF values
- attributes travel automatically with values (stack↔stack, stack↔heap)
- set/processed by extensions (listeners, native peers)
- good for data flow & data quality properties
- supports multiple attributes for objects, fields, operands and local vars

Basics: Attribute Propagation



Attributes - how to set and access

- various APIs that are part of the *Info classes
  - ThreadInfo, **StackFrame**
    - hasOperandAttr(type), hasLocalAttr(idx,type)
    - getOperandAttr(type), getLocalAttr(idx,type)
    - operandAttrIterator(), localAttrIterator(idx)
    - addOperandAttr(attr), addLocalAttr(idx, attr)
    - setOperandAttr(attr), setLocalAttr(idx, attr)
    - ...
  - **ElementInfo**, Fields
    - hasObjectAttr(type), hasFieldAttr(type)
    - getObjectAttr(type), getFieldAttr(field,type)
    - addObjectAttr(attr), addFieldAttr(field, attr)
    - setObjectAttr(attr), setFieldAttr(field, attr)
    - ...
  - ClassInfo, MethodInfo
    - hasAttr(type)
    - getAttr(type)
    - addAttr(attr)
    - setAttr(attr)
    - ...



gov.nasa.jpf.vm.InfoObject,
gov.nasa.jpf.util.ObjectList

Basics: Attribute Example

```
class StopWatchFuzzer extends ListenerAdapter {
  static class TimeValAttr {} // just a tag
  static TimeValAttr timeValAttr = .. // we need only one
  ...
                                        long t1 = System.currentTimeMillis();
  @Override
  public void instructionExecuted(VM vm, ThreadInfo ti, Instruction nextInsn,
                                        Instruction executedInsn){
    if (executedInsn instanceof NATIVERETURN){
        if (executedInsn.getMethodInfo().getName().equals("currentTimeMillis")){
          // the two top operand slots hold the 'long' time value
          StackFrame frame = ti.getModifiableTopFrame();
          frame.addLongOperandAttr( timeValAttr);
    ...
                                        if ((t2 - t1) > MAX_DURATION) explode();
  @Override
  public void executeInstruction(VM vm, ThreadInfo ti, Instruction insnToExecute){
    if (insnToExecute instanceof LCMP){
      // create and set CG if operand has TimeVal attr
      ...
      StackFrame frame = ti.getTopFrame();
      if (frame.hasOperandAttr(1, TimeVal.class) ||
                                frame.hasOperandAttr(3, TimeVal.class)){
          IntChoiceFromSet cg = new IntChoiceFromSet( CG_ID, -1, 0, 1);
          vm.setNextChoiceGenerator(cg);
        ...
```

Basics: Native Peers

- what to do with (host-VM) native methods?
  - state lives outside of Java (files, network, windows, ..)
  - how can we backtrack? `System.currentTimeMillis()`
- solution: intercept native method calls and replace with calls to *NativePeer* methods that are executed by host VM
- association done at class load time

- native peer methods can be used for more than native code:
  - atomic (no transition breaks unless explicitly created)
  - not automatically state tracked
  - can directly interact with JPF (ChoiceGenerator creation etc.)
- can be venerable optimization
- main challenge is to translate between different object models: MJI

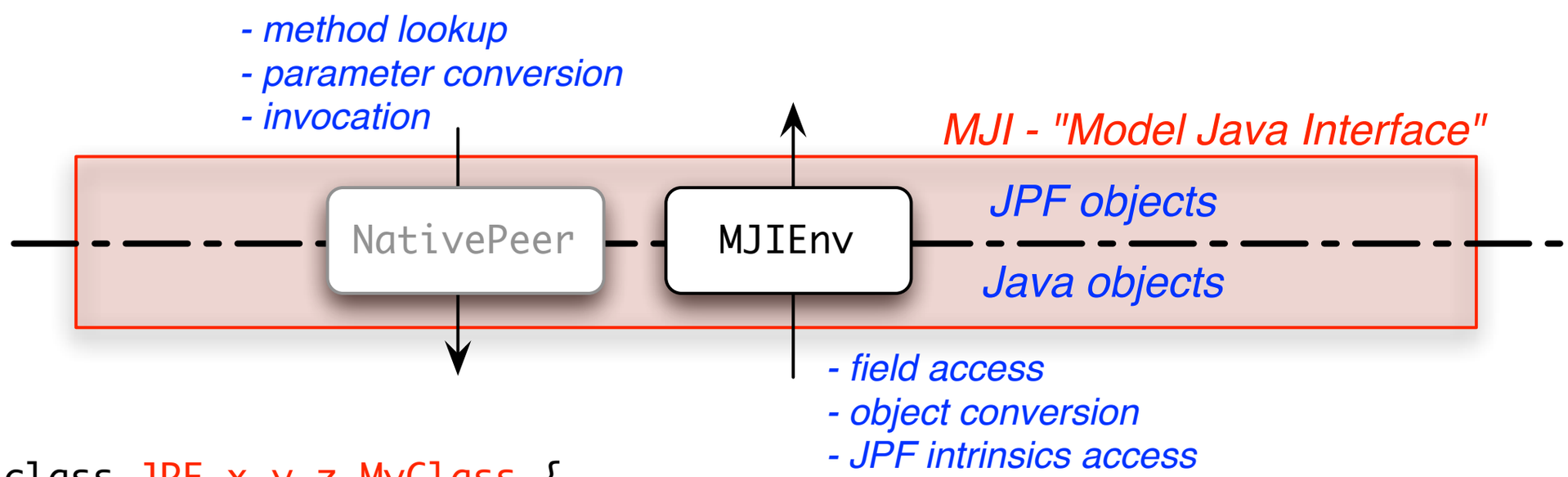
Basics: MJI - Model Java Interface

- transfer from JPF executed code into JVM executed code

"Model" Class

JPF executed

```
package x.y.z;
class MyClass {
  ..
  native String foo (int i, String s);
}
```




```
class JPF_x_y_z_MyClass {
  public static int
      foo__ILjava_lang_String_2__Ljava_lang_String_2 (MJIEnv env, int objRef,
                                                      int i, int sRef) {
    String s = env.getStringObject(sRef);
    ..
    int ref = env.newString(..);
    return ref;
  }
}
```

host VM executed

"NativePeer" Class

Basics: Native Peer 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;
  }
```

loops
(performance, would clutter traces)

```
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);
}
```

delegation
(complex but “un-interesting” functions)

Basics: Instruction Factories

- what NativePeers are at the class level, Instruction instances are at the bytecode level
- VM is (mostly) agnostic to what `Instruction.execute()` does
- concrete `Instruction` class hierarchy implements execution semantics
- can be configured at startup time and replaced at runtime (`MethodInfo` keeps code as replaceable Instruction array)
- VM uses a configured `InstructionFactory` class to delegate instantiation of instruction objects

Basics: InstructionFactory Motivation

- provide alternative Instruction classes for relevant bytecodes
- create & configure InstructionFactory that instantiates them
- integer overflow example:

```
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);
```

compiler

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

JPF configuration

```
jvm.insn_factory.class =
    .numeric.NumericInstructionFactory
```

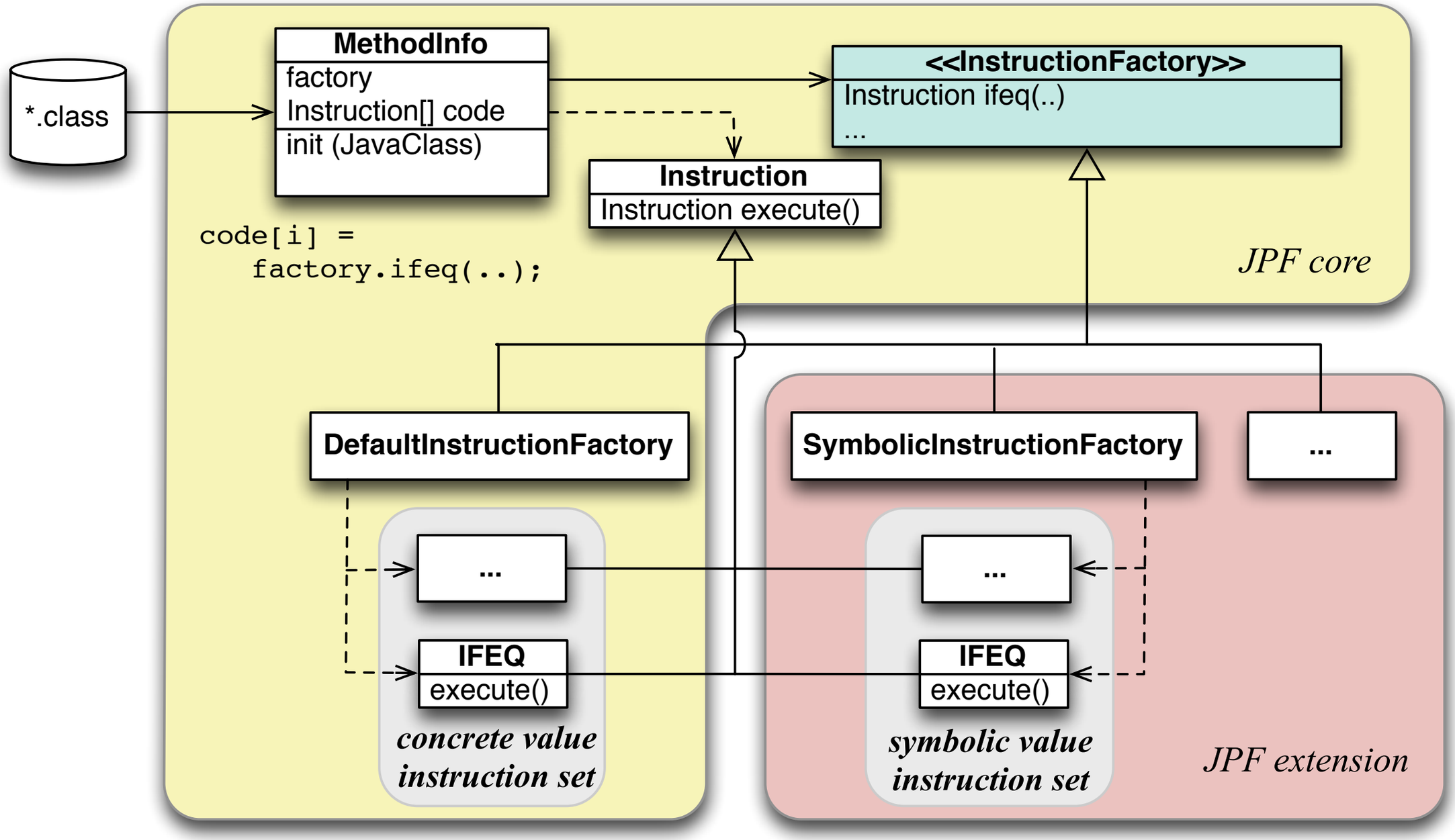
class loading

```
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..
```

code execution
(by JPF)

Basics: InstructionFactory

- Instruction objects get created during class loading by configurable InstructionFactory



Basics: JPF Output

- output at JPF level
  - logging/debugging
    - JPFLogger - named, leveled loggers to be used from JPF code
  - reporting
    - Publisher - configurable, event-driven, format specific generator
    - PublisherExtension - extension point for specific listener output etc.

- output at system under test level
  - normal `System.out.print..()`
  - configurable to be path or tree output

Basics: Logging

- for debugging and JPF error reporting
- piggybacks on `java.util.Logging`
- loggers identified by name, can be shared among classes/packages

```
.. static JPFLogger logger = JPF.getLogger("class");
```

- supports standard log levels: severe, warning, info, fine, finer

```
.. logger.warning("classpath element ", cpe, " not found");
.. logger.finer("resolving interface: ", ifcName, " of ", name);
```

use parameter APIs, avoid string concatenation for inactive levels

- default and logger specific levels can be configured

`log.level=warning` default level: site/project properties

`+log.finer=class` level for specific logger: command line

Basics: Publishers

- used to create verification reports
- event driven "topics":
  start, transition, constraint, property_violation, finished
- configurable, format specific publishers (console, XML, HTML etc.)
- publishers support configurable "items": error, trace, statistics etc.

```
report.publisher = console

report.console.class = gov.nasa.jpf.report.ConsolePublisher
report.console.start = jpf,sut
...
report.console.property_violation = error,trace,snapshot
report.console.finished = result,statistics
```

default in jpf-core/jpf.properties

Basics: PublisherExtensions

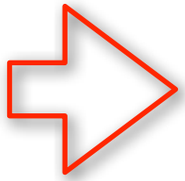
- can be used to add additional output/items from your listeners etc.

```
class DeadlockAnalyzer extends ListenerAdapter {
  ...
  public DeadlockAnalyzer (Config config, JPF jpf){
    .. jpf.addPublisherExtension(ConsolePublisher.class, this);

  @Override
  public void publishPropertyViolation (Publisher publisher) {
    PrintWriter pw = publisher.getOut();
    publisher.publishTopicStart("thread ops");
    printThreadOps( pw);
  ...
```

Basics: SUT output

- just normal System.out.print..(..);
- default "tree" mode produces output when executed (can be hard to read)

```
main(..){
   System.out.println("started");
   ... random.nextInt(42);
   System.out.println("finished");
}
```



```
started
..
finished
finished
finished
..
```

- "path" mode preserves output integrity for each SUT execution path

```
listener=.listener.PathOutputMonitor

report.console.property_violation=...,output
```

```
~~~~~~~
started..
finished
~~~~~~~
started..
finished
```

Basics: Testing Infrastructure

- not just for testing JPF itself
- convenient way to write JPF test drivers for SUT test suites

```
public class MyTest extends TestJPF {
  ..
  @Test
  public void testSomething(){
    if (verifyNoPropertyViolation()){
      saveTheWorld(..);
    }
```

test fails if JPF detects property violation

code verified by JPF

- caveat: test classes need to be in host VM and JPF classpath (executed by both
- can be executed stand-alone or through Ant/JUnit

```
> bin/test test-class [method,..]
> bin/ant test
```

Basics: JPF Project Layout

- all JPF projects share uniform directory layout
- use ANT based build system
- use same configuration scheme
- binary distributions are slices of source distributions (interchangeable)
- 3rd party tools & libraries can be included (self-contained)
- projects should have examples and regression test suites
- projects should have out-of-the-box IDE config (NB,Eclipse)

jpf-proj	***project root directory***
jpf.properties	*JPF project properties file (runtime def: native_classpath, classpath, sourcepath)*
src	***project sources***
main	*host-VM executed classes (listeners, infrastructure etc.)*
peers	*host-VM executed library classes (MJI native peers)*
classes	*JPF executed library classes →sourcepath*
annotations	*JPF processed Java annotations*
tests	*regression tests*
examples	*demos →sourcepath*
build.xml	*Ant build script (compile, build, test, clean)*
build	***build artifacts***
main	*temporary build artifacts (classfiles)*
...	
	permanent build artifacts
jpf-proj.jar	*host-VM executed jar (main,peers →native_classpath)*
jpf-proj-classes.jar	*JPF executed library jar (classes,annotations →classpath)*
jpf-proj-annotations.jar	*separate anotations jar (for JPF external SUT exec)*
lib	***required runtime jars*** *→native_classpath*
tools	***build & run tools***
RunJPF.jar	*JPF startup jar (executable)*
RunTest.jar	*JPF testing jar (executable)*
RunAnt.jar	*JPF build jar (executable)*
bin	***scripts***
jpf	*JPF startup script (→RunJPF.jar)*
test	*JPF build script (→RunTest.jar)*
ant	*JPF build script (→RunAnt.jar)*

...

Directory layout: jpf-core sources

- jpf-core/src
  - main
    - gov.nasa.jpf — JPF itself, Config, adapter classes
      - .vm — ClassInfo, ThreadInfo, MethodInfo,...
      - .jvm
        - .bytecode — default JVM instruction set
      - .listener
      - .util

main jpf-core implementation

  - peers — peers for essential system classes
  - classes — replacements (models) for essential system classes
  - tests — regression tests
  - examples

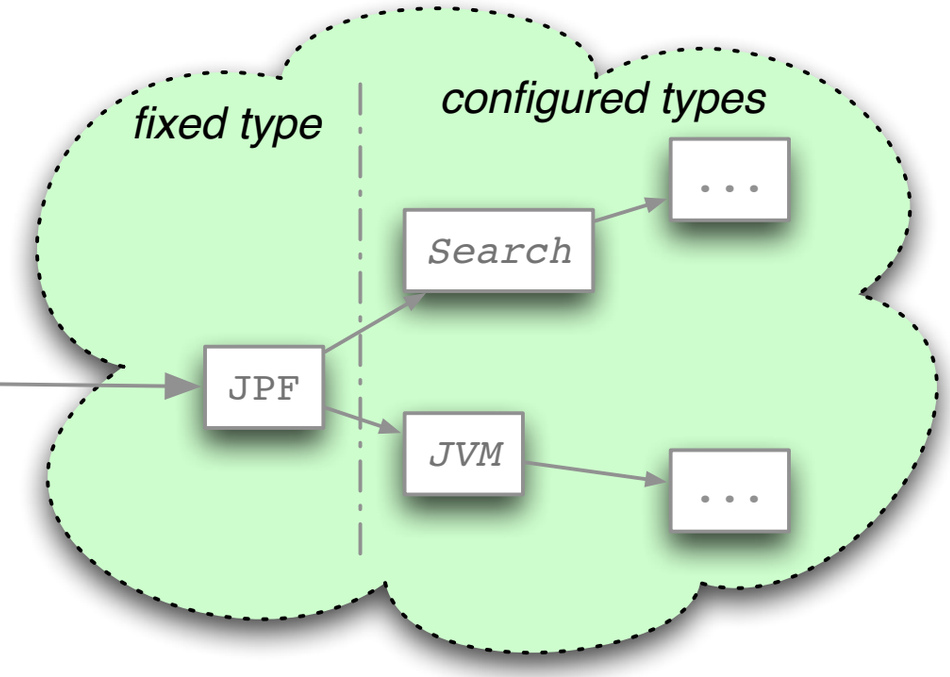
Basics: Configuration

- open - easy to extend, easy to get lost

java.util.Properties instance

Config object

```
..
some_value=42
search.class=..BFSHeuristic
vm.class=..JVM
..
```




```
> bin/jpf +some_value=42  ...
```

'+' command line arguments

```
...
```

```
..
search.class = .search.heuristic.BFSHeuristic
..
```

```
..
vm.class = gov.nasa.jpf.jvm.JVM
search.class = gov.nasa.jpf.search.DFSearch
..
```

hierarchy of Java property files

depends on installed JPF (extension) projects

Basics: Configuration Files

- just glorified `java.util.Properties` (don't confuse with program properties to verify)

- 3 main configuration file types
  - site properties - which project configuration files should be read
  - project properties - where are the classes for this project, what factories need to be set
  - application properties - how to run a specific application, what to check

- if you get lost, run JPF with
  - `-log` - shows which configuration files are loaded in which order
  - `-show` - displays final property values after parsing

- all property files support variable expansion: `${property-key}`

Configuration: Site Properties

- tells JPF where to find projects, and which project properties should be loaded by default
- not essential as long as you work within jpf-core (current directory properties are loaded by default)
- absolutely essential if working outside jpf-core - most common cause of installation problems
- can be in ~/.jpf/ or anywhere upwards in your current directory

```
jpf/
    site.properties
    jpf-core/
    <your-project>/
```

project name

project location

```
jpf-core = /Users/pcmehlitz/projects/jpf/jpf-core
jpf-numeric = ...
...

extensions = ${jpf-core}
```

which project properties to load automatically

Configuration: Project Properties

- fixed name, fixed location: `<your-project>/jpf.properties`
- tells JPF and host VM where to find the classes for this project (used to construct classpaths)
- has to set required factories (e.g. InstructionFactory)

```
jpf-numeric = ${config_path}

jpf-numeric.native_classpath=\
   build/jpf-numeric.jar;\
   lib/apfloat.jar

jpf-numeric.classpath=\
   build/jpf-numeric-classes.jar

jpf-numeric.test_classpath=\
  build/tests

jpf-numeric.sourcepath=...

jvm.insn_factory.class = \
   gov.nasa.jpf.numeric.bytecode.InstructionFactory
```

classes used by host VM

classes used by JPFs VM

classes used by both VMs
for running JPF tests

e.g. instruction factory to use by JPF

Configuration: Application Properties

- `<application>.jpf`
- can be specified in lieu of main class from command line, e.g.
  `bin/jpf src/examples/StopWatch.jpf`
- specifies application entry point (main class) to run by JPF
- configures listeners to use
- convention: keep close to main class,
  e.g. `src/examples/`
  `StopWatch.java`
  `StopWatch.jpf`

```
target = StopWatch
target.args = ...

listener = .listener.StopWatchFuzzer
```

← mandatory: application main class to run by JPF

e.g. listener configuration

Configuration: Initialization Sequence

command line property arguments

```
> bin/jpf [-log][-show] {+log.info=..} .../RobotManager.jpf
```

application properties

```
target = RobotManager
target_args = ...

@using = jpf-aprop
@import = ./my.properties

shell = .shell.basicshell.BasicShell
listener = .aprop.listener.SharedChecker
...
```

jpf.properties in current directory

```
jpf-core = ${config_path}

jpf-core.native_classpath=\
  ${jpf-core}/build/jpf.jar;\
  ...
  ${jpf-core}/lib/bcel.jar;

jpf-core.classpath=\
  build/jpf-classes.jar

jpf-core.test_classpath=\
  build/tests

jpf-core.sourcepath=\
  src/classes
...
```

```
jpf-awt-shell = ${config_path}

@using = jpf-awt

jpf-awt-shell.native_classpath=...
jpf-awt-shell.classpath=...
...
```

site properties

```
jpf.home = ${user.home}/projects/jpf

jpf-core = ${jpf.home}/jpf-core
jpf-awt = ${jpf.home}/awt
jpf-shell = ${jpf.home}/jpf-shell
jpf-aprop = ...
...
extensions = ${jpf-core},${jpf-shell}
```

all jpf.properties in order of *extensions*

4. command line
   - debugging

3. application properties

   `<project>/.../*.jpf`

   - system-under-test
   - listeners, shells

2. project properties

   `<project>/jpf.properties`

   - project class paths
   - project dependencies

1. site properties

   `~/.jpf/site.properties`

   - project locations
   - pre-loaded projects

Basics: Where to get Help - Mailing List

- http://groups.google.com/group/java-pathfinder
- anyone can join & read, member only post
- moderate volume (~50 messages/month)
- subscribe on http://groups.google.com/group/java-pathfinder/subscribe

Basics: Where to get Help - Wiki

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

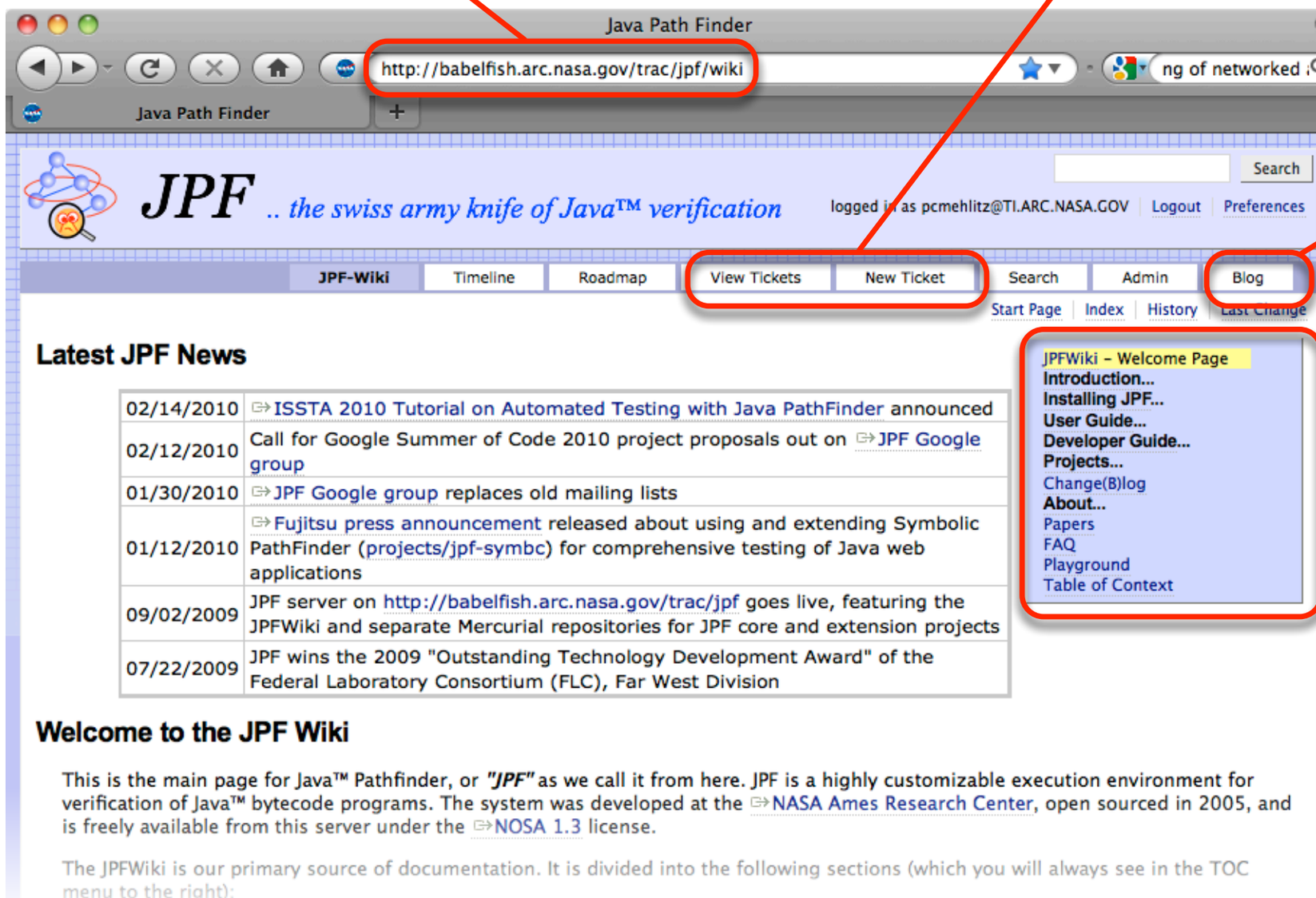
- public read access
- edit for account holders (also non-NASA)

bug tracking

- Trac ticket system

project blog

- announcements
- important changes



hierarchical navigation menu

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

Basics: Recap

- the core of JPF is a VM running on top of a Java VM
  - JPF executes the bytecode of the test application

- JPF is not a monolithic tool, it is highly configurable / extensible
  - ChoiceGenerators
  - Listeners
  - Attribute Objects
  - ...

- extensions should follow JPF conventions
  - directory layout
  - configuration mechanism

Tutorial Part 2: Install & Run JPF

- install (all JPF projects have same procedure)
  - start with `jpf-core`
    - download
    - build
    - test
    - configure
  - repeat for JPF extension projects (`jpf-symbc, jpf-numeric, ...`)
  - for available projects, see "Projects" navigation bar on wiki

- running JPF
  - several alternative ways to run JPF
    - `java` from command line
    - using bin/* JPF scripts (or RunJPF.jar, RunTest.jar)
    - using IDEs (Eclipse, NetBeans)

  - examples

Install: Prerequisites

- JDK7 (make sure JDK, *not* JRE is installed/used - check by running javac from command line)
  http://www.oracle.com/technetwork/java/javase/downloads
  (JDK 6 works too but requires special "java-1.6" JPF branch)

- Mercurial (Version Control System, uses Python - check by running hg from command line):
  http://mercurial.selenic.com/wiki/Download

- optional IDEs:
  - Eclipse: http://www.eclipse.org
  - NetBeans: http://www.netbeans.org
  - IntelliJ IDEA: http://www.jetbrains.com/idea/download

Install: Tutorial Directory Structure

- only convention, can be configured differently
- for now we only need jpf-core, but later-on we will create our own jpf-tutorial project
- suggestion:

```
jpf/                        (1) create manually:  mkdir jpf
  site.properties           (3) create automatically:  bin/jpf -addproject
  jpf-core/                 (2) clone with mercurial:  hg clone http://...
  jpf-tutorial/
```

Install: Steps

1. create jpf/ directory as common root for JPF projects `> mkdir jpf`
2. clone jpf-core with Mercurial
3. build jpf-core from cloned directory
4. test jpf-core build
5. create site.properties with JPF

Install: Step 2 - Getting JPF

- get jpf-core sources from Mercurial repository

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

  recommended if you work continuously with JPF

- fallback: get *.zip snapshot attachment from http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/jpf-core (infrequently updated)

- last resort - memory stick

Install: Step 3 - Building JPF

- jpf-core comes with all the required build tools (except javac)
- build from *within* downloaded jpf-core directory

```
> cd jpf-core
> bin/ant build
```

```
Buildfile: /Users/pcmehlitz/projects/jpf/jpf-core/build.xml
...
-init:
    [mkdir] Created dir: /Users/pcmehlitz/projects/jpf/jpf-core/build
...
build:
      [jar] Building jar: /Users/pcmehlitz/projects/jpf/jpf-core/build/jpf.jar
      [jar] Building jar: /Users/pcmehlitz/projects/jpf/jpf-core/build/jpf-classes.jar
      ...
BUILD SUCCESSFUL
```

- same build procedure for all JPF projects

Install: Step 4 - Test JPF build

- JPF comes with regression test suite
- test from within downloaded jpf-core directory
  `bin/ant test`

```
Buildfile: /Users/pcmehlitz/projects/jpf/jpf-core/build.xml
...
test:
    [junit] Running TypeNameTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.385 sec
    ...
    [junit] Running gov.nasa.jpf.util.script.ScriptEnvironmentTest
    [junit] Tests run: 3, Failures: 0, Errors: 0, Time elapsed: 0.028 sec

BUILD SUCCESSFUL
Total time: 1 minute 44 seconds
```

- same procedure for all projects (if they have tests)

Install: Step 5 - Create site.properties

- can be done manually (see Basics/Configuration)
- site.properties can also be kept in global location (~/.jpf/site.properties)
- in this tutorial, we keep it within our directory tree and create it automatically

- run from *within* jpf-core

```
> bin/jpf -addproject ../site.properties
```

- check created jpf/site.properties file:

```
# auto-generated JPF site properties
jpf-core = ${user.home}/projects/jpf/jpf-core
extensions = ${jpf-core}
```

Running JPF - Alternatives

- 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 MyProgram.jpf`
  - using provided scripts: `>bin/jpf ...MyProgram.jpf`

- running JPF from within JUnit
- running JPF from your program (tools using JPF)

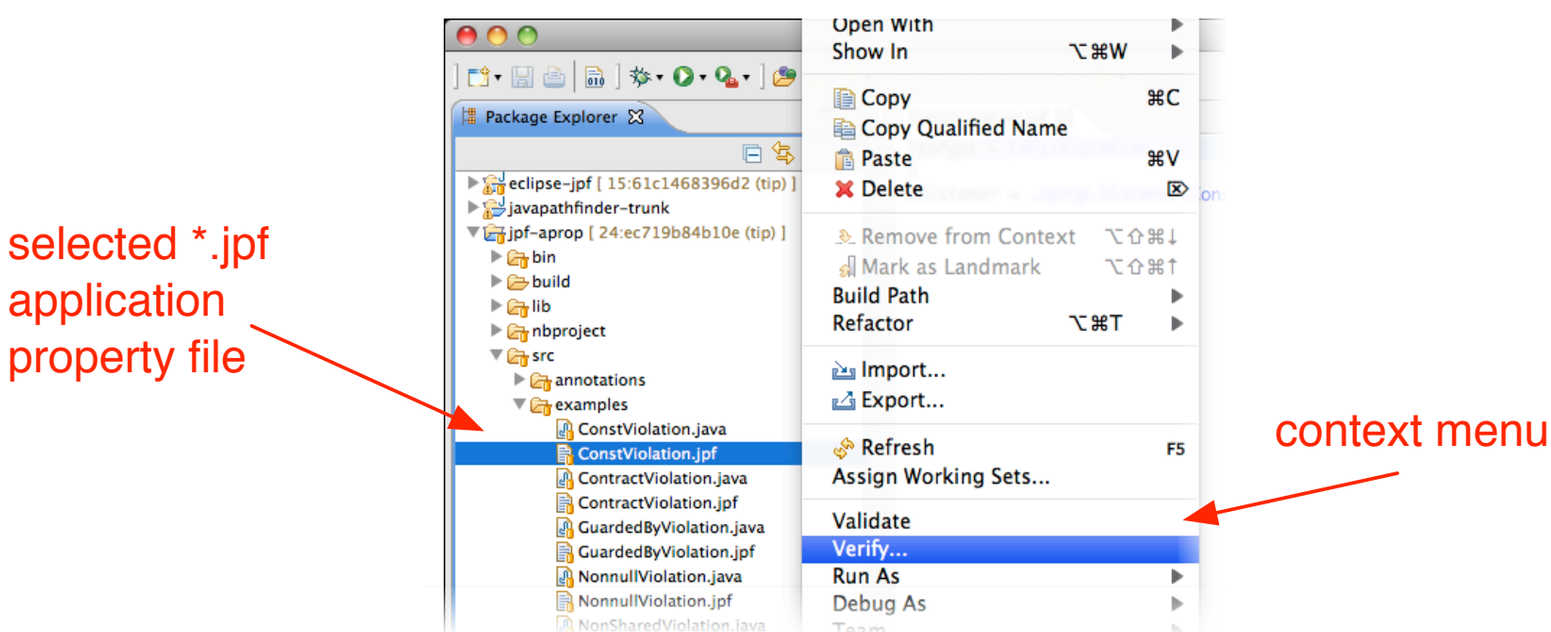
- using NetBeans or Eclipse plugins
  - "Verify.." context menu item for selected *.jpf application property file
  - using provided launch configs (Eclipse) or run targets (NetBeans)

Runing 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

debugging

- use Eclipse JPF plugin
  - clone http://babelfish.arc.nasa.gov/hg/jpf/eclipse-jpf
  - or download snapshot from http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/eclipse-jpf
  - or install from update site http://babelfish.arc.nasa.gov/trac/jpf/raw-attachment/wiki/install/eclipse-plugin/update/
  - launch JPF by selecting *.jpf file and invoking "Verify.." context menu item



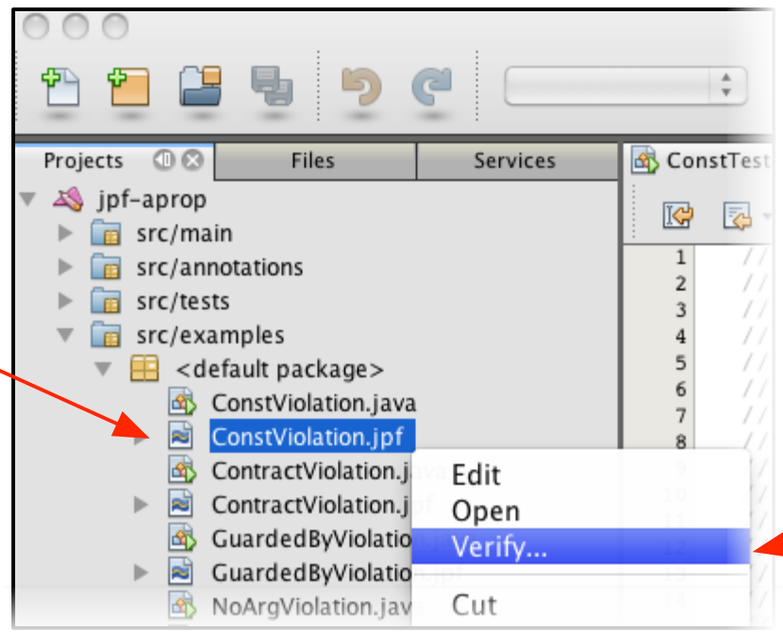
Running JPF: from NetBeans

- 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

debugging

- 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
  - launch JPF by selecting *.jpf file and invoking "Verify.." context menu item

selected *.jpf application property file



context menu

Running JPF - Basic jpf-core Examples

- we use jpf-core examples to familiarize ourselves with running JPF and play with configurations

  - HelloWorld - simple JPF invocation
  - Rand - model checking random number choices, basic configuration
  - Racer - scheduling choices leading to race condition, listener configuration

- .. switch over to jpf-core now

```
> cd ../jpf-core
```

Running JPF: HelloWorld Example

- just our first explicit JPF invocation
- src/examples/HelloWorld.java

```
public static void main(String[] args){
    // guess what it does..
}
```

- src/examples/HelloWorld.jpf

```
target = HelloWorld
```

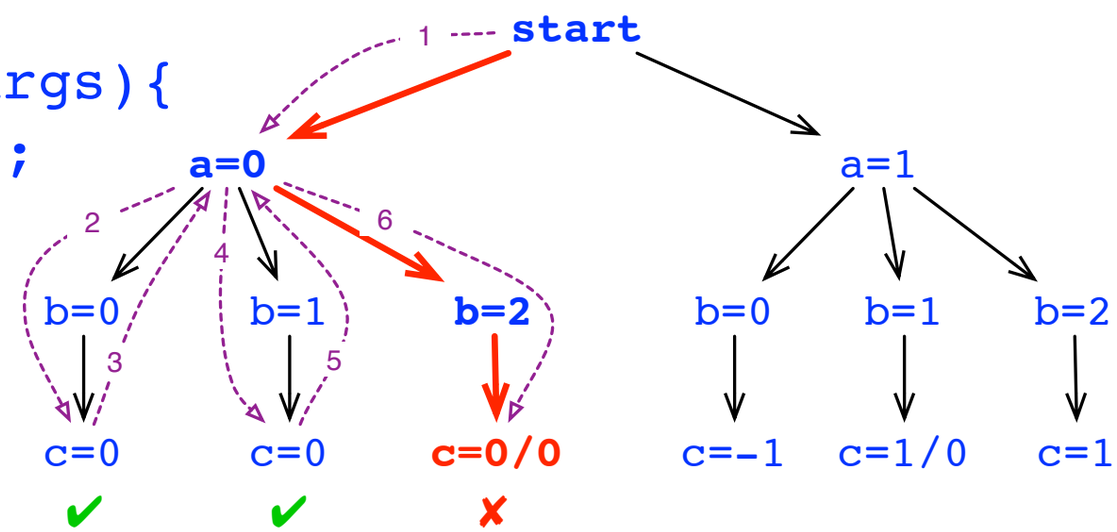
- execute: `>bin/jpf src/examples/HelloWorld.jpf`

```
JavaPathfinder v7.0 - (C) RIACS/NASA Ames Research Center
===================================== system under test
application: HelloWorld.java
...
===================================== results
no errors detected

===================================== statistics
elapsed time:       00:00:00
states:             new=1, visited=0, backtracked=1, end=1
...
```

Running JPF: Random Data Example

- src/examples/Rand.java

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



- certain combination of random values can cause division by zero
- application property file src/examples/Rand.jpf

```
target = Rand
cg.enumerate_random = true
```

?.. =false ?

```
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
```

- execute: `>bin/jpf src/examples/Rand.jpf`

```
JavaPathfinder v7.0 - (C) RIACS/NASA Ames Research Center
===================================== system under test
application: Rand.java
...
===================================== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.ArithmeticException: division by zero
        at Rand.main(Rand.java:16)
```

Running JPF: Data Race

- src/examples/Racer.java

```
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);
}
```

- src/examples/Racer.jpf

```
target = Racer
listener = gov.nasa.jpf.listener.PreciseRaceDetector
```

Running JPF: Data Race

- src/examples/Racer.java

```
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);
}
```

data race

- src/examples/Racer.jpf

```
target = Racer
listener = gov.nasa.jpf.listener.PreciseRaceDetector
```

Running JPF: Data Race Example (cont.)

- run `>bin/jpf src/examples/Racer.jpf`

```
============================= system under test
application: Racer.java
============================= search started: 5/23/11 11:35 AM
...
============================= error #1
gov.nasa.jpf.listener.PreciseRaceDetector
race for field Racer@13d.d
  main at Racer.main(Racer.java:16)
                    "int c = 420 / racer.d;
  Thread-0 at Racer.run(Racer.java:7)
                    "d = 0;

============================= trace #1
----------------------------- transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="root",isCascaded:false,{>main}]
      [2894 insn w/o sources]
  Racer.java:11                    : Racer racer = new Racer();
  Racer.java:1                     : public class Racer implements Runnable {
      [1 insn w/o sources]
  Racer.java:3                     : int d = 42;
...
----------------------------- transition #5 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{>main,Thread-0}]
  Racer.java:16                    : int c = 420 / racer.d;                // (4)
...
```

- what happens if we run without listener?

  `>bin/jpf +listener=  src/examples/Racer.jpf`

Running JPF: Data Race Example (cont.)

- to see what is *really* going on, run with additional listener:

```
> bin/jpf +listener+=,.listener.ExecTracker src/examples/Racer.jpf
```

```
                # choice: ThreadChoiceFromSet[id="root",isCascaded:false,{>main}]
        Racer.java:11                   : Racer racer = new Racer();
0 : [0] new Racer@317
0 : [1] dup
0 : [2] invokespecial Racer.<init>()V
        Racer.java:1                    : public class Racer implements Runnable {
0 : [0] aload_0
...
------------------------------------ [1] forward: 0 new
                # choice: ThreadChoiceFromSet[id="start",isCascaded:false,{>main,Thread-0}]
0 : [0] executenative JPF_java_lang_Thread.start____V
0 : [1] nativereturn java.lang.Thread.start()V
        Racer.java:15                   : doSomething(1000);                    // (3)
...
------------------------------------ [4] forward: 3 new end
------------------------------------ [3] backtrack: 2
------------------------------------ [3] done: 2
------------------------------------ [2] backtrack: 1
                # choice: ThreadChoiceFromSet[id="sleep",isCascaded:false,{main,>Thread-0}]
        Racer.java:6                    : doSomething(1001);                    // (1)
1 : [-1] runstart
1 : [0] sipush
1 : [1] invokestatic Racer.doSomething(I)V
...
```

Running JPF: Fun with extensions

- JPF can do more than model checking concurrent applications

- slightly more interesting examples to show gamut of what can be done with extension points:
  - jpf-numeric extension - detecting numeric errors
  - jpf-awt extension - model checking graphical user interfaces
  - jpf-statechart - verifying DSL models
  - jpf-shell - if output gets too long..
  - ...

- symbolic execution jpf-symbc subject to own tutorial

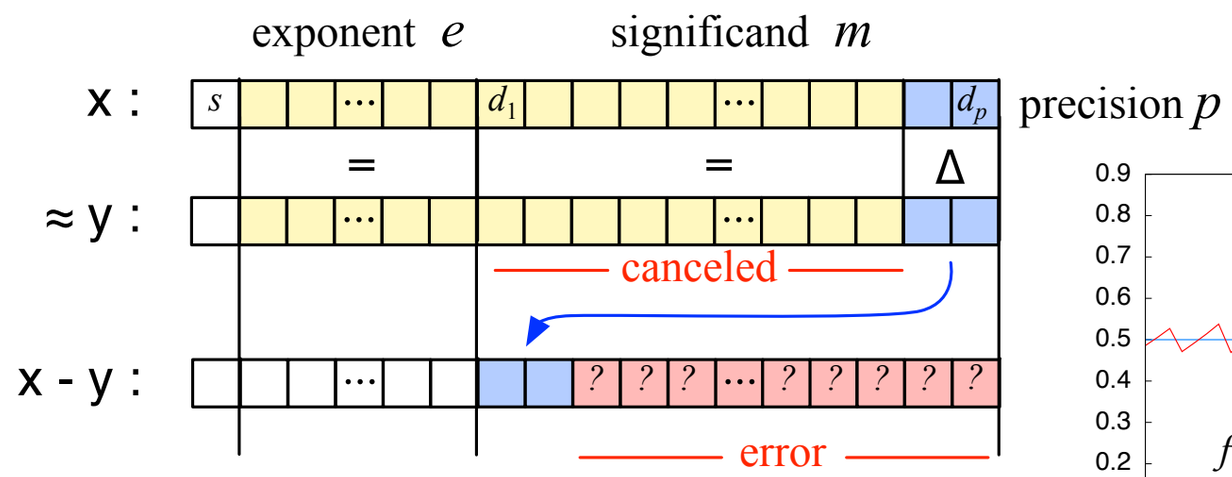
Examples: Numeric Extension

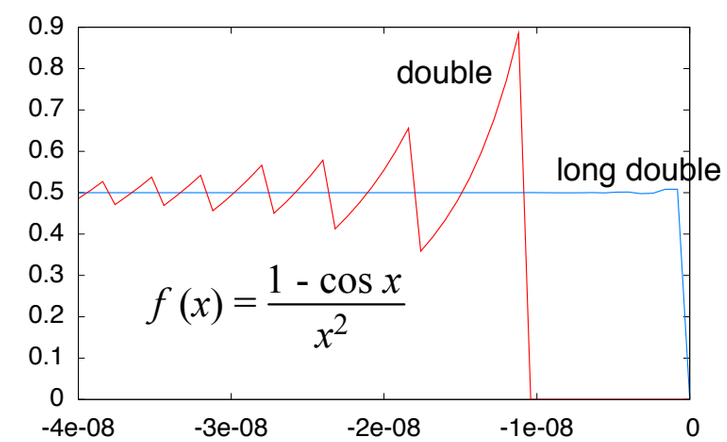
- get jpf-numeric extension
  `hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-numeric`
- build it
  `bin/ant`
- run example (from inside jpf-numeric dir)
  `bin/jpf src/examples/CatastrophicCanellation.jpf`
- try to find this with testing..

```
[WARNING] cancellation of:
-7.917111340668963E36+7.917111340668962E36=-1.1805916207174113E21
      at CatastrophicCancellation.main(CatastrophicCancellation.java:29)

res=-1.1805916207174113E21 (should be -0.827396...)
```

Examples: Numeric Extension (cont.)

- 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 significands





Annie Cuyt et al. "A remarkable example of catastrophic cancellation unraveled"

```
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);
```

.numeric.bytecode.InstructionFactory

```
res=-1.1805916207174113e21 (should be -0.827396...)
...
========================================================= error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
        java.lang.ArithmeticException: cancellation of:
              -7.917111340668963E36 + 7.917111340668962E36
                    = -1.1805916207174113E21
```

jpf-numeric/src/examples/CatastrophicCancellation.jpf

Examples: Numeric Extension (cont.)

- InstructionFactory with specialized FP instructions

```
public class NumericInstructionFactory extends InstructionFactory {
  ...
  @Override public Instruction dsub() { return new DSUB(); }
}

public class DSUB extends Instruction {...
  @Override public Instruction execute (ThreadInfo ti) {
    StackFrame frame = ti.getModifiableTopFrame();
    long l1 = frame.popLong();
    double d1 = Double.longBitsToDouble(l1);

    long l2 = frame.popLong();
    double d2 = Double.longBitsToDouble(l1);

    double r = d2 - d1;
    frame.pushLong(Double.doubleToLongBits(r);

    if (checkCancellation(l1,l2)){
      reportCancellation(l1,l2);
    }

    return getNextInsn(ti);
  }
```

Examples: GUI verification

- get jpf-awt extension

  `hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-awt`

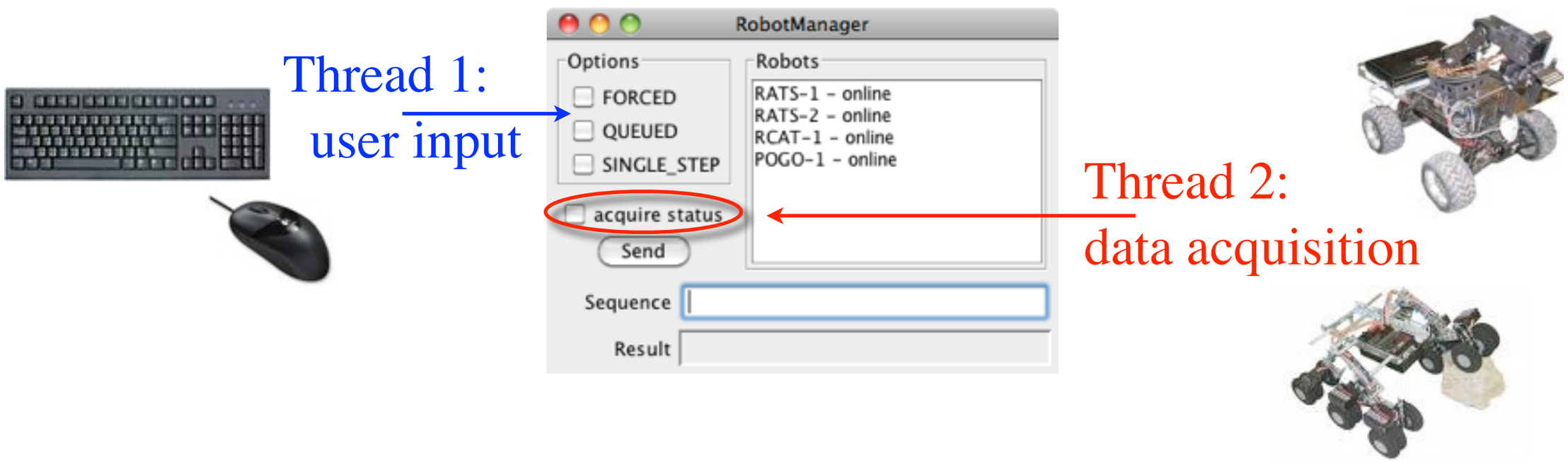
- build it

  `bin/ant`

- run example

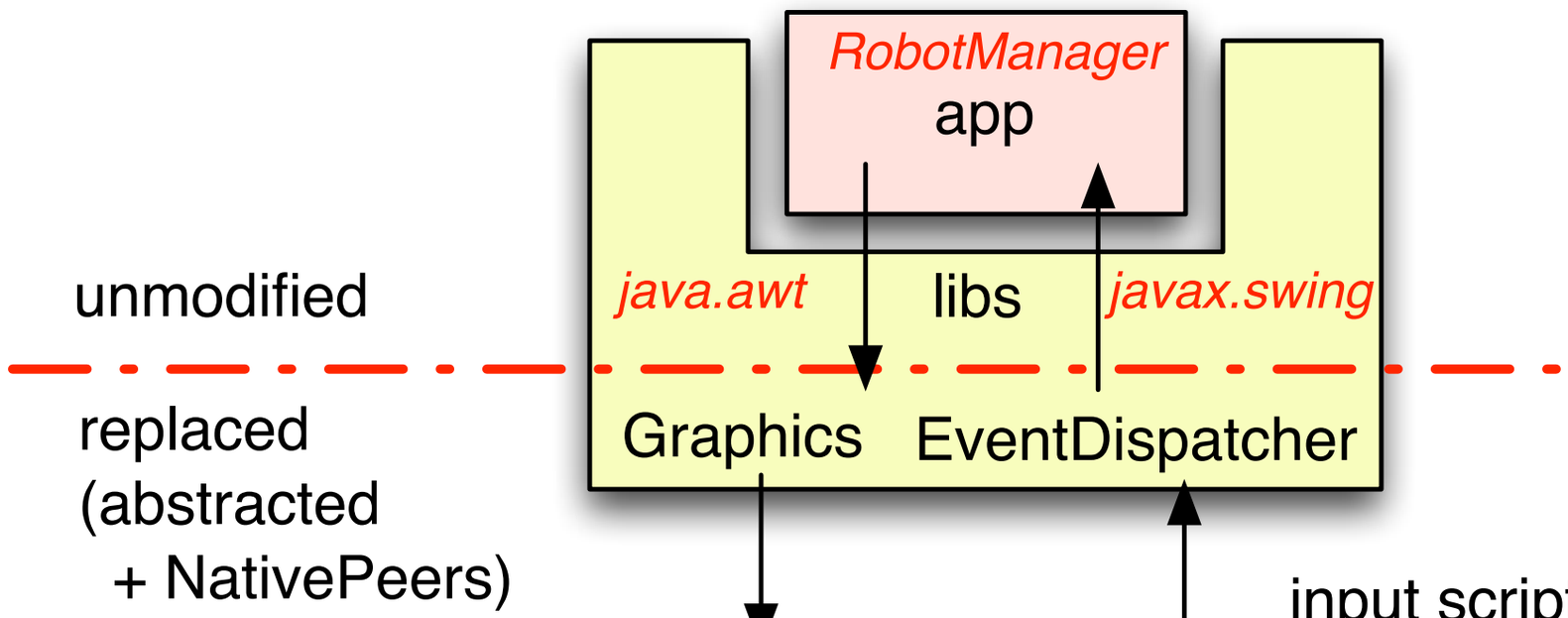
  `bin/jpf src/examples/RobotManager-thread.jpf`

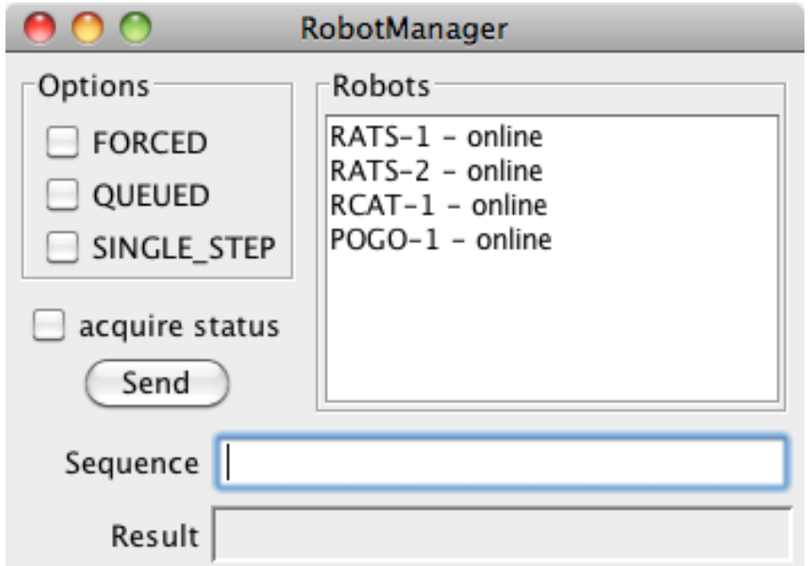
- GUI application that uses concurrent data acquisition:



Examples: GUI verification (cont.)

- modeled libraries + NativePeers
- user simulated with input script supporting alternatives





input script (user model)

```
ANY { $<FORCED|QUEUED>.doClick() }
$Send.doClick()
...
```

input event generator

```
FORCED.doClick()
Send.doClick()
...
```

```
QUEUED.doClick()
Send.doClick()
...
```

...

Examples: GUI verification (cont.)

- JPF finds NullPointerException that depends on scheduling sequences **and** user input

JPF Shell – RobotManager-thread.jpf

Properties | Report | Test Output | Verify Output | Config View

Showing Streams: STDOUT STDERR STDIN

```
------------------------------------------------------ transition #35 thread: 3
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{AWT-Event
  RobotManager.java:44            : return id;
  RobotManager.java:246           : onlineRobots.remove(robot.getId());
      [98 insn w/o sources]
  RobotManager.java:246           : onlineRobots.remove(robot.getId());
  RobotManager.java:248           : listModel.changed(robot);
------------------------------------------------------ transition #36 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{>AWT-Even
  RobotManager.java:260           : return onlineRobots.get(robotName);
      [47 insn w/o sources]
  RobotManager.java:260           : return onlineRobots.get(robotName);
  RobotManager.java:566           : Robot robot = model.getOnlineRobot(selectedRobotName)
  RobotManager.java:567           : String result = model.sendSequence(robot, sequence);
  RobotManager.java:269           : return robot.processSequence(sequence);

====================================================== statistics
elapsed time:       0:00:03
states:             new=638, visited=207, backtracked=808, end=0
search:             maxDepth=68, constraints hit=0
choice generators:  thread=41 (signal=0, lock=10, shared ref=28), data=672
heap:               new=11358, released=5263, max live=3333, gc-cycles=843
instructions:       879891
max memory:         81MB
loaded code:        classes=404, methods=4959
```

Type here to send text to the program's STDIN

~1500 insn/transition,
error at depth 36
⇒ trace too long

Examples: GUI verification (cont.)

- error too general

```
java.lang.NullPointerException:
   Calling 'processSequence(Ljava/lang/String;)Ljava/lang/String;' on null object
         at RobotManager.sendSequence(RobotManager.java:269)
         at RobotManagerView.sendSequence(RobotManager.java:567)
         at RobotManagerView$3.actionPerformed(RobotManager.java:341)
         at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:120)
         ...
```

- generic trace analysis listener can help
- still too time consuming in many cases

+ OverlappingMethodAnalyzer Listener

```
...
2:    < RobotManager@13d.isRobotOnline(Ljava/lang/String;)Z
2: >-  RobotManager@13d.getOnlineRobot(Ljava/lang/String;)LRobot;
---------------------------------------- #13
3: >-  RobotManager@13d.isRobotOnline(Ljava/lang/String;)Z
---------------------------------------- #14
3:    < RobotManager@13d.isRobotOnline(Ljava/lang/String;)Z
3: >-  RobotManager@13d.setRobotOnline(LRobot;Z)V
---------------------------------------- #15
2:    < RobotManager@13d.getOnlineRobot(Ljava/lang/String;)LRobot;
2: >-  RobotManager@13d.sendSequence(LRobot;Ljava/lang/String;)Ljava/lang/String
```

thread 1

thread 2

Examples: GUI verification (cont.)

- better approach - get specific properties from SUT annotations
- more efficient to check (using jpf-aprop extension)
- **self explaining** - no need for further analysis

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

+ NonSharedChecker Listener

```
========================================================= error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.AssertionError: NonShared object: RobotManager@13d
          accessed in live thread cycle: AWT-EventQueue-0,Thread-2,AWT-EventQueue-0,main
    at RobotManager$ListModel.getSize(RobotManager.java:173)
    at javax.swing.JList.setSelectedIndex(JList.java:280)
    at java.awt.EventDispatchThread.processScriptAction(JPF_java_awt_EventDispatchThread)
    at java.awt.EventDispatchThread.run(EventDispatchThread.java:65)

========================================================= statistics
elapsed time:       0:00:02
states:             new=531, visited=0, backtracked=512, end=0
search:             maxDepth=19, constraints hit=0
choice generators:  thread=13 (signal=0, lock=8, shared ref=2), data=604
```

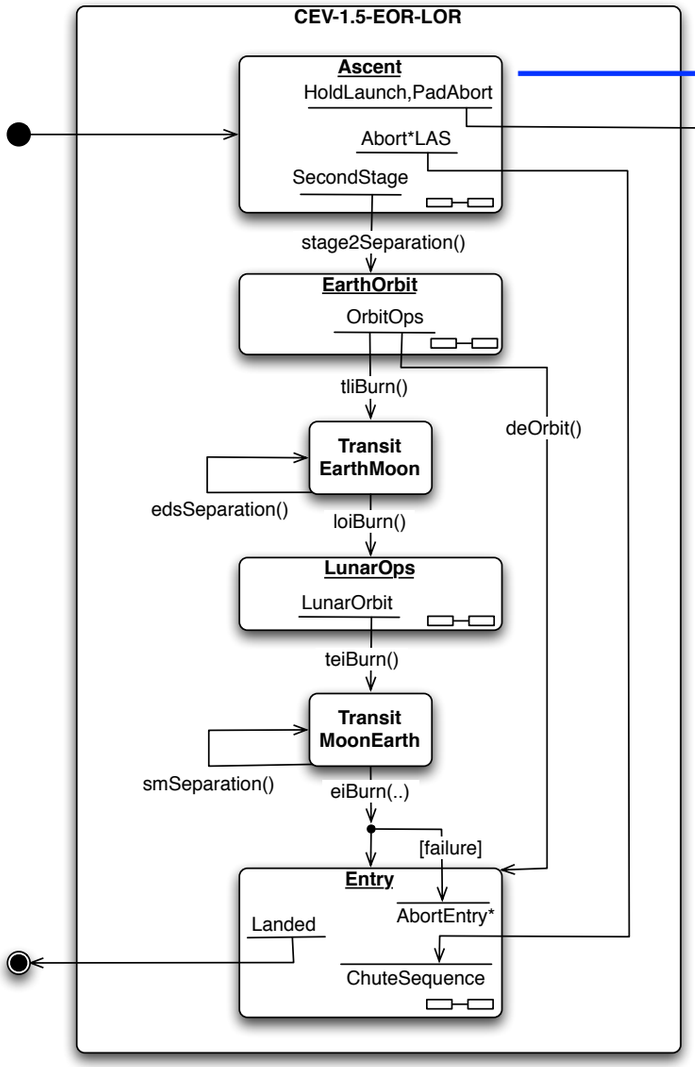
Examples: DSL Models - Statecharts

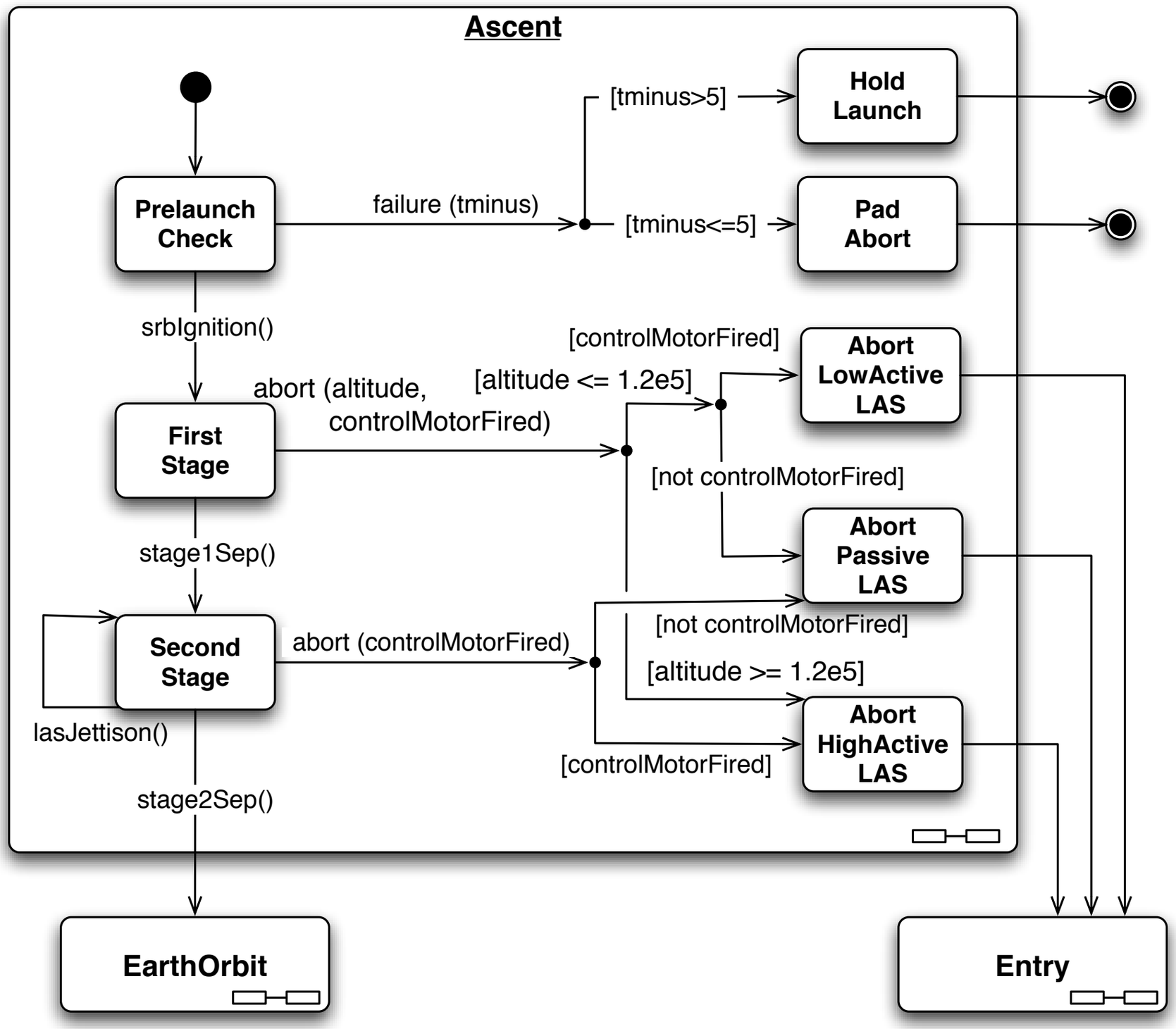
- motivation: model driven development
- as models get more complex, risk of model inconsistencies increases
- classic example: state machine verification

- get jpf-statechart extension
  `hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-statechart`
- build it
  `bin/ant`
- run example (from inside jpf-statechart dir)
  `bin/jpf src/examples/jpfESAS/CEV_15EOR_LOR-ascent-guards.jpf`

Example: Statechart Verification

model is understandable ..

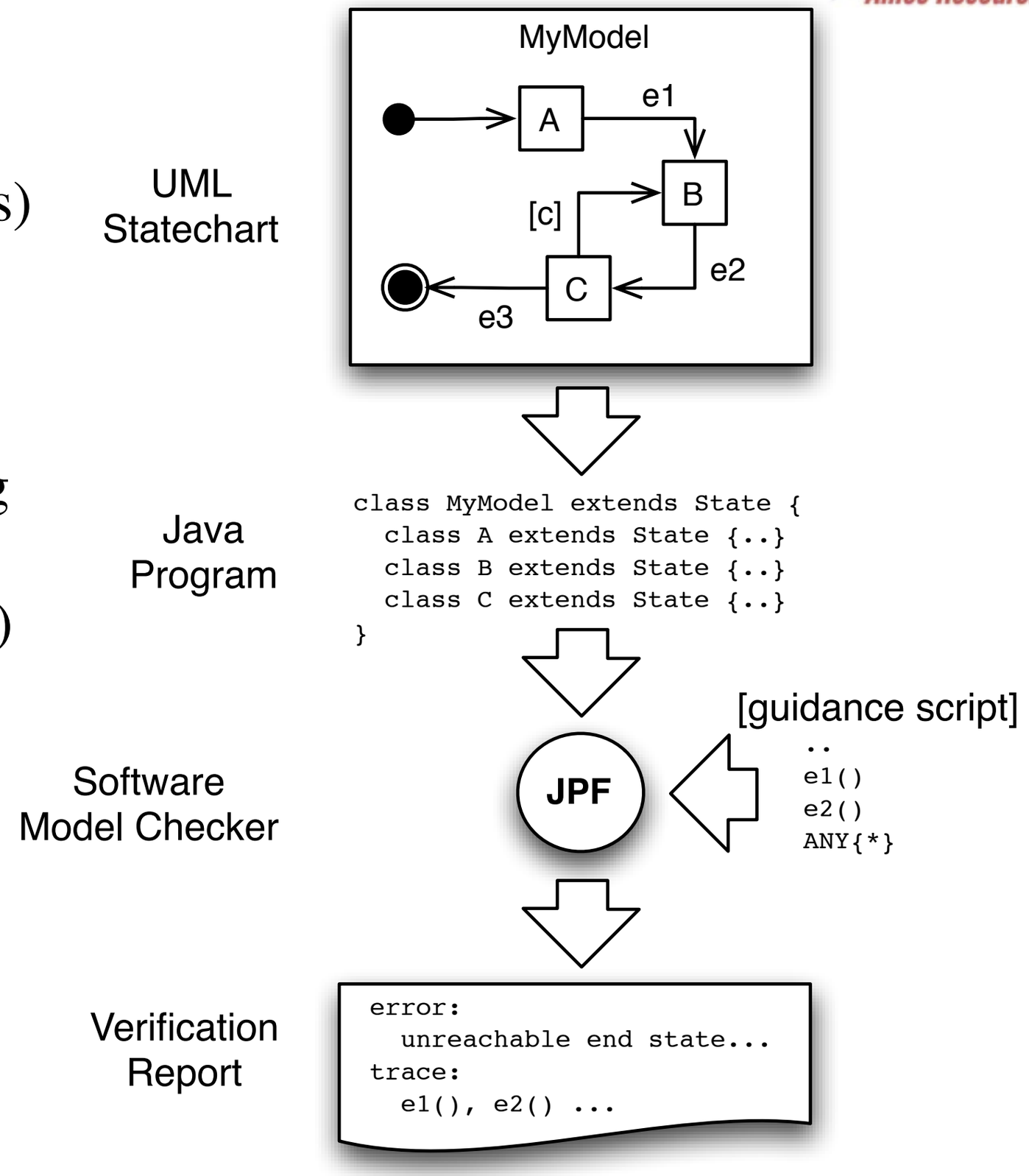




.. **but does it work**?
(actions, guards, reachability etc.)

Example: Statechart Verification (cont.)

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

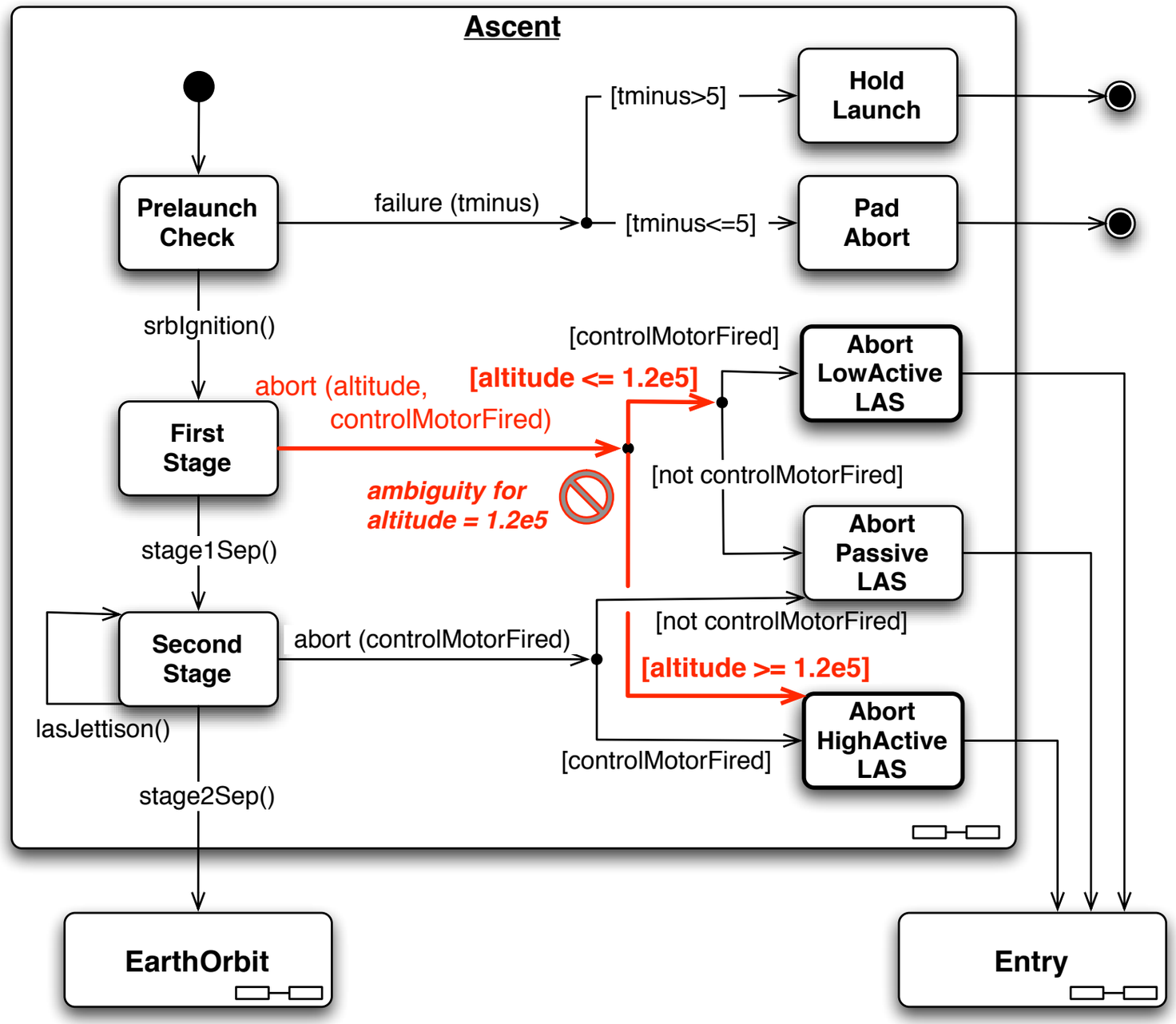


Example: Statechart Verification (cont.)

- 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)
```



jpf-statechart/src/examples/jpfESAS/CEV_15EOR_LOR-ascent-guards.jpf

Running JPF Revisited: The Shell

- sometimes JPF reports too long for IDE Output view or console
- 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
...
```

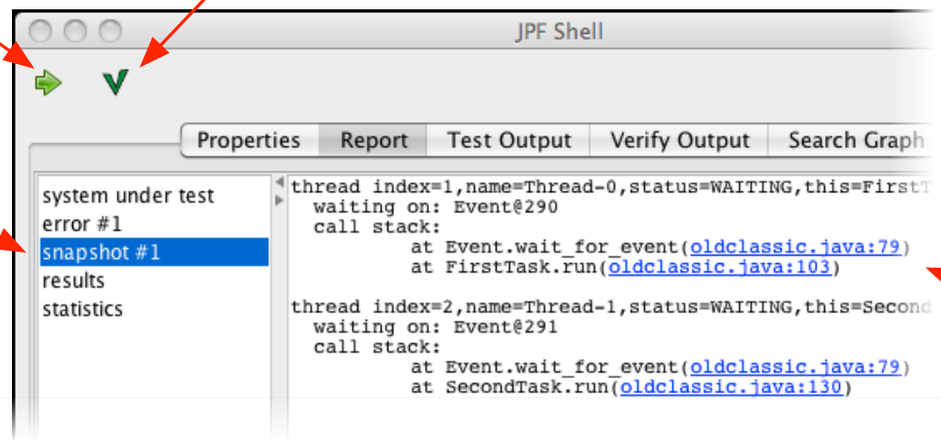
Running JPF Revisited: The Shell (cont.)

browse complex reports

run SUT (test)

run JPF (verify)

browse report topics

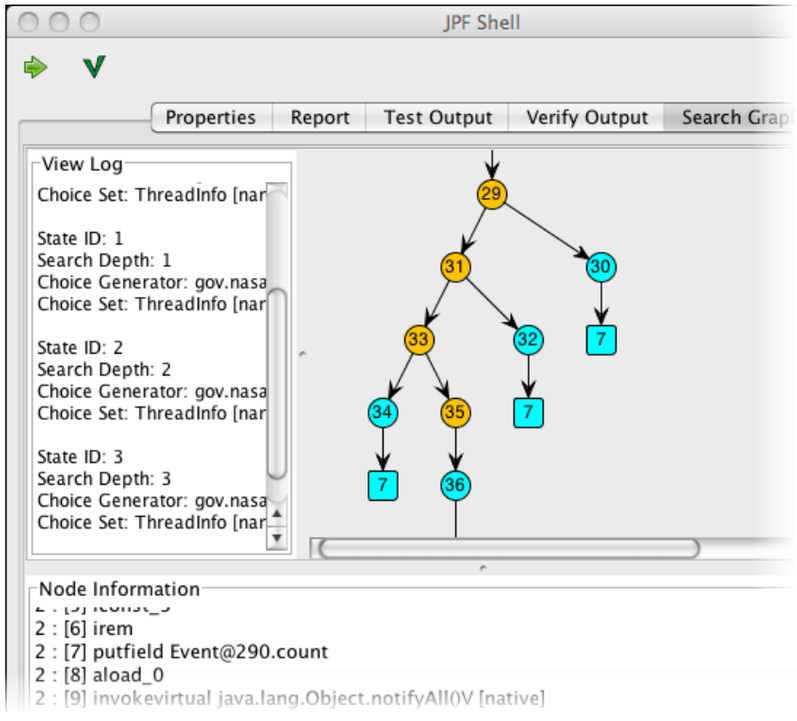


configurable view tabs

```
shell=.shell.basicshell.BasicShell
shell.panels=...
```

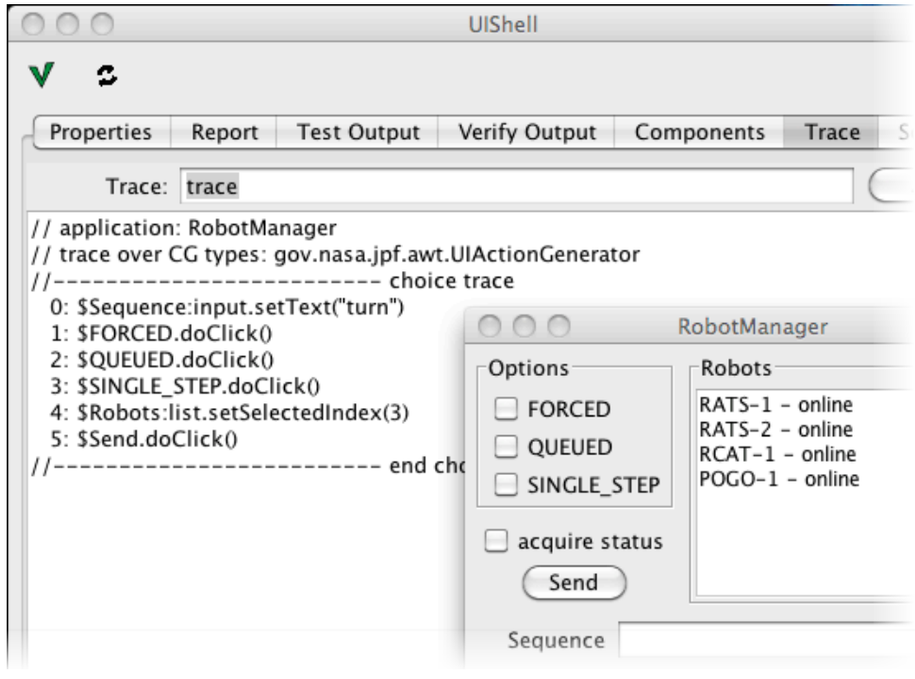
clickable source links

inspect JPF run




```
.. shell.panels+=,searchgraph
```

application/domain specific views




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

Tutorial Part 3: Extending JPF

- hands on exercise - your task is to program JPF listeners
- implemented in new jpf-tutorial project
- Task 1: safety property listener
- Task 2: control flow variation via listener and data attributes

- sources on http://babelfish.arc.nasa.gov/trac/jpf/raw-attachment/wiki/presentations/start/jpf-tutorial-src.zip

Exercises: Create Tutorial Project

- create your sandbox test project (we don't want to modify jpf-core)
- projects can be created manually (see directory structure in basics), but tedious
- we use the jpf-template tool to create our tutorial project (just a convenience tool to create project skeletons)

```
jpf/
  site.properties
  jpf-core/
  jpf-template/
  jpf-tutorial/
```

- (1) get & build jpf-template:

```
> hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-template
> cd jpf-template
> bin/ant
```

- (2) create jpf-tutorial project

```
> bin/create_project ../jpf-tutorial
> cd ../jpf-tutorial
```

```
jpf/
  ...
  jpf-tutorial/
    jpf.properties
    build.xml
    src/
    ...
```

Exercises: Task 1

- goal: create a listener that detects if numeric field values are outside of specified range
- example program

```
public class RangeCheckerTest {
  int data; // should be within [0..42]

  void setData(int d){
    data = d;
  }

  public static void main(String[] args){
    RangeCheckerTest rct = new RangeCheckerTest();
    rct.setData( 42);  // should not trigger violation
    rct.setData(-42);  // should trigger violation
  }
}
```

Exercises: Task 1 - Create Test

- create example/test: src/examples/RangeCheckerTest.java

```
public class RangeCheckerTest {
  int data; // should be within [0..42]

  void setData(int d){
    data = d;
  }

  public static void main(String[] args){
    RangeCheckerTest rct = new RangeCheckerTest();
    rct.setData( 42);  // should not trigger violation
    rct.setData(-42);  // should trigger violation
  }
}
```

Exercises: Task 1 - Listener (1)

- write listener: src/main/tutorial/RangeChecker.java

```
public class RangeChecker
    extends gov.nasa.jpf.PropertyListenerAdapter
```

- initialize from Config

```
FieldSpec fieldSpec;
int min, max;
```

init from RangeCheckerTest.jpf

app properties

rc.field=RangeCheckerTest.data
rc.min = 0
rc.max = 42

```
public RangeChecker (Config conf){
  String spec = conf.getString("rc.field");
  fieldSpec = FieldSpec.createFieldSpec(spec);
  min = conf.getInt("rc.min", Integer.MIN_VALUE);
  max = conf.getInt("rc.max", Integer.MAX_VALUE);
```

Exercises: Task 1 - Listener (2)

- intercept PUTFIELD post execution notification in listener

```
@Override
public void instructionExecuted(VM vm, ThreadInfo ti,
                                Instruction nextInsn,
                                Instruction insn){

  if (insn instanceof PUTFIELD){
    if (isRelevantField(..)){
      if (isValueOutOfRange(..)){
        storeError();
```

Exercises: Task 1 - Listener (3)

- filling in the blanks: checking field and values

```
FieldSpec fieldSpec;
int min, max;

void instructionExecuted(VM vm,ThreadInfo ti,..){..
  if (insn instanceof PUTFIELD){
    PUTFIELD put = (PUTFIELD)insn;
    if (isRelevantField(put)){
      if (isValueOutOfRange(put)){
        storeError(ti, put); ..

boolean isRelevantField(PUTFIELD insn){
  FieldInfo fi = insn.getFieldInfo();
  return fieldSpec.matches(fi);
}

boolean isValueOutOfRange(PUTFIELD insn){
  int v = (int)insn.getLastValue();
  return (v < min) || (v > max);
}
```

Exercises: Task 1 - Listener (4)

- finishing it up: get execution context and report the error

```
String error;

void storeError (ThreadInfo ti, PUTFIELD insn){
  FieldInfo fi = insn.getFieldInfo();
  error = String.format(
    "field %s=%d out of range in thread %s at %s",
      fi.getFullName(), insn.getLastValue(),
      ti.getName(), insn.getSourceLocation());
}

@Override
public boolean check(Search search, VM vm) {
  return (error == null);
}

@Override
public String getErrorMessage(){ return error; }
```

Exercises: Task 1 - Listener Sources

```java
package tutorial;

import gov.nasa.jpf.*;import gov.nasa.jpf.search.*;
import gov.nasa.jpf.vm.*;
import gov.nasa.jpf.jvm.bytecode.*;
import gov.nasa.jpf.util.*;

public class RangeChecker
          extends PropertyListenerAdapter {
  FieldSpec fieldSpec;
  int min, max;
  String error;

  public RangeChecker (Config conf){
    String spec = conf.getString("rc.field");
    fieldSpec = FieldSpec.createFieldSpec(spec);
    min = conf.getInt( "rc.min", Integer.MIN_VALUE);
    max = conf.getInt( "rc.max", Integer.MAX_VALUE);
  }

  protected boolean isRelevantField(PUTFIELD insn){
    return fieldSpec.matches(insn.getFieldInfo());
  }

  protected boolean isValueOutOfRange(PUTFIELD insn){
    int v = (int)insn.getLastValue();
    return (v < min) || (v > max);
  }

  protected void storeError (ThreadInfo ti,
                             PUTFIELD insn){
    FieldInfo fi = insn.getFieldInfo();

    error = String.format(
        "field %s=%d out of range in thread %s at %s",
        fi.getFullName(), insn.getLastValue(),
        ti.getName(), insn.getSourceLocation());
  }

  @Override
  public void instructionExecuted(VM vm,
                ThreadInfo ti, Instruction nextInsn,
                Instruction insn){
    if (insn instanceof PUTFIELD){
      PUTFIELD put = (PUTFIELD)insn;
      if (isRelevantField(put)){
        if (isValueOutOfRange(put)){
          storeError(ti, put);
          vm.breakTransition();
        }
      }
    }
  }

  @Override
  public boolean check(Search search, VM vm) {
    return (error == null);
  }

  @Override
  public String getErrorMessage(){
    return error;
  }
}
```

Exercises: Task 1 - Running

- create src/examples/SUT.jpf

```
target = RangeCheckerTest
listener=tutorial.RangeChecker
rc.field=RangeCheckerTest.data
rc.min=0
rc.max=42
```

- run it: bin/jpf src/examples/SUT.jpf

```
...
============================= error #1
tutorial.RangeChecker
field RangeCheckerTest.data=-42 out of range in thread main
     at RangeCheckerTest.setData(RangeCheckerTest.java:5)
...
```

- Congratulations - Your First Listener!

Exercises: Task 1 - Reference

- see NumericValueChecker listener in jpf-core
- sources in src/main/gov/nasa/jpf/listener/NumericValueChecker.java
- example in src/examples/NumericValueCheck.{java,jpf}
- run with

```
bin/jpf src/examples/NumericValueCheck.jpf
```

- also handles local variables

Exercises: Task 2

- write a listener that detects time based comparisons and creates a ChoiceGenerator that executes <u>both</u> branches

```
long t1 = System.currentTimeMillis();
..
long t2 = System.currentTimeMillis();
if (t2 - t1 > MAX_TIME) {
  ①  println("disaster");
} else {
  ②  println("all fine");
}
```

Exercises: Task 2 - Bytecode Pattern

- approach hinges on bytecode pattern

```
long t1 = System.currentTimeMillis();
..
long t2 = System.currentTimeMillis();
if (t2 - t1 > MAX_TIME) {
  println("disaster");
} else {
  println("all fine");
}
```

obtain time values

compute duration

compare duration

```
0:   invokestatic  #2; // System.currentTimeMillis()
3:   lstore_1
4:   invokestatic  #2; // System.currentTimeMillis()
7:   lstore_3
8:   lload_3
9:   lload_1
10:  lsub
11:  ldc2_w        #3; //long 42l
14:  lcmp
15:  ifle 29
18:  getstatic     #5; // System.out
21:  ldc           #6; // "disaster"
23:  invokevirtual #7; // PrintStream.println()
26:  goto 37
29:  getstatic     #5; // System.out
32:  ldc           #8; // "all fine"
34:  invokevirtual #7; // PrintStream.println()
37:  return
```

Exercises: Task 2 - Create Test

- create example/test: src/examples/StopWatchTest.java

```
public class StopWatchTest {

  public static void main(String[] args){
    long tStart = System.currentTimeMillis();
    System.out.println("some lengthy computation..");
    long tEnd = System.currentTimeMillis();

    if (tEnd - tStart > 1000){
      throw new RuntimeException("it took too long..");
    }

    System.out.println("all fine, finished in time");
  }
```

Exercises: Task 2 - Approach

- write listener: src/main/lab/StopWatch.java

```
public class StopWatch
    extends gov.nasa.jpf.ListenerAdapter
```

- use JPF *value attribute objects* to identify time values and durations
- use JPF *ChoiceGenerator* to force control flow

```
invokestatic System.currentTimeMillis()
...
```
instructionExecuted() → create & set TimeVal operand attribute

```
lsub
...
```
executeInstruction() → propagate TimeVal attr to result operand

```
lcmp
ifle ..
...
```
executeInstruction() → use CG to push result operand

Exercises: Task 2 - Listener (1)

- instructionExecuted() → create & set TimeVal operand attribute

```
static class TimeVal {}
static TimeVal timeValAttr = new TimeVal();
...
@Override
public void instructionExecuted (VM vm, ThreadInfo ti,
                Instruction nextInsn, Instruction insn){

  if (insn instanceof NATIVERETURN){
    MethodInfo mi = ((NATIVERETURN)insn).getMethodInfo();
    if (mi.getUniqueName().equals("currentTimeMillis()J")){
      StackFrame frame = ti.getModifiableTopFrame();
      frame.addLongOperandAttr( timeValAttr);
...
```

not very efficient, there are better ways to do this (see StopWatchFuzzer)

Exercises: Task 2 - Listener (2)

- executeInstruction() → propagate TimeVal attribute to result operand

```
...
@Override
public void executeInstruction (VM vm, ThreadInfo ti,
                                Instruction insn){

  if (insn instanceof LSUB){
    StackFrame frame = ti.getModifiableTopFrame();
    if (frame.hasOperandAttr(1, TimeVal.class) ||
        frame.hasOperandAttr(3, TimeVal.class)){
      long v2 = frame.popLong();
      long v1 = frame.popLong();
      ti.pushLong(v1-v2);

      frame.addLongOperandAttr( timeValAttr);
      ti.skipInstruction( insn.getNext());
...
```

why pre-exec notification?

Exercises: Task 2 - Listener (3)

- executeInstruction() → top half creates CG

```
static final String CG_ID = "LCMP_choice";
...
@Override
public void executeInstruction (VM vm, ThreadInfo ti,
                                Instruction insn){
  ...
  if (insn instanceof LCMP){
    if (!ti.isFirstStepInsn()){      // first time (top half)
      StackFrame frame = ti.getTopFrame();
      if (frame.hasOperandAttr(1, TimeVal.class) ||
          frame.hasOperandAttr(3, TimeVal.class)){
        IntChoiceFromSet cg =
           new IntChoiceFromSet(CG_ID, -1,0,1);
        vm.getSystemState().setNextChoiceGenerator(cg);
        ti.skipInstruction(insn);
      }
    } else { // second time (bottom half)
...
```

Exercises: Task 2 - Listener (4)

- executeInstruction() → bottom half pushes CG choice to be used by subsequent IFLE instruction

```
...
@Override
public void executeInstruction (VM vm, ThreadInfo ti,
                                Instruction insn){
  ...
  if (insn instanceof LCMP){
    if (!ti.isFirstStepInsn()){  // first time (top half)
      ...
    } else { // second time (bottom half)
      IntChoiceFromSet cg =
        vm.getCurrentChoiceGenerator(CG_ID,
                                     IntChoiceFromSet.class);
      if (cg != null){
        StackFrame frame = ti.getModifiableTopFrame();
        frame.popLong(); frame.popLong();
        frame.push( cg.getNextChoice());
        ti.skipInstruction( insn.getNext());
...
```

Exercises: Task 2 - Listener Sources

```java
package tutorial;

import gov.nasa.jpf.*;
import gov.nasa.jpf.vm.*;
import gov.nasa.jpf.vm.choice.*;
import gov.nasa.jpf.jvm.bytecode.*;

public class StopWatch extends ListenerAdapter {
  static class TimeVal {}
  static TimeVal timeValAttr = new TimeVal();

@Override
public void instructionExecuted (VM vm, ThreadInfo ti,
              Instruction nextInsn, Instruction insn){
  if (insn instanceof NATIVERETURN){
    MethodInfo mi =((NATIVERETURN)insn).getMethodInfo();
    if (mi.getUniqueName().equals(
                  "currentTimeMillis()J")){
      StackFrame frame = ti.getModifiableTopFrame();
      frame.addLongOperandAttr( timeValAttr);
    }
  }
}
```

```java
static final String CG_ID = "LCMP_choice";

@Override
public void executeInstruction (VM vm, ThreadInfo ti,
                                Instruction insn){
  if (insn instanceof LSUB){
    StackFrame frame = ti.getModifiableTopFrame();
    if (frame.hasOperandAttr(1, TimeVal.class) ||
        frame.hasOperandAttr(3, TimeVal.class)){
      long v2 = frame.popLong();
      long v1 = frame.popLong();
      frame.pushLong(v1-v2);
      frame.addLongOperandAttr( timeValAttr);
      ti.skipInstruction( insn.getNext());
    }

  } else if (insn instanceof LCMP){
    if (!ti.isFirstStepInsn()){  // first time (top half)
      StackFrame frame = ti.getTopFrame();
      if (frame.hasOperandAttr(1, TimeVal.class) ||
          frame.hasOperandAttr(3, TimeVal.class)){
        IntChoiceFromSet cg =
           new IntChoiceFromSet(CG_ID, -1,0,1);
        vm.setNextChoiceGenerator(cg);
        ti.skipInstruction(insn);
      }
    } else { // second time (bottom half)
      IntChoiceFromSet cg =
        vm.getCurrentChoiceGenerator(CG_ID,
                                     IntChoiceFromSet.class);
      if (cg != null){
        StackFrame frame = ti.getModifiableTopFrame();
        frame.popLong(); frame.popLong();
        frame.push( cg.getNextChoice());
        ti.skipInstruction( insn.getNext());
      }
    }
  }
}
```

Exercises: Task 2 - Run Example

- create src/examples/StopWatchTest.jpf

```
target = StopWatchTest
listener=tutorial.StopWatch
```

- run it: bin/jpf src/examples/StopWatchTest.jpf

```
...
============== search started: 10/23/11 11:28 PM
some lengthy computation..
all fine, finished in time
all fine, finished in time
```

? why repeated ?

```
============== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.RuntimeException: it took too long..
      at StopWatchTest.main(StopWatchTest.java:31)
...
```

Examples: Task 2 - Reference

- see StopWatchFuzzer listener in jpf-core
- sources in src/main/gov/nasa/jpf/listener/StopWatchFuzzer.java
- example in src/examples/StopWatch.{java,jpf}
- run with

```
bin/jpf src/examples/StopWatch.jpf
```

- more efficient way to detect System.currentTimeMillis() (uses classLoaded() notification)
- delegates LSUB execution

- could also be done with a InstructionFactory, avoiding skipping Instructions from listeners

Conclusions

Conclusions

.. confused?

- 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
  - we are here to help

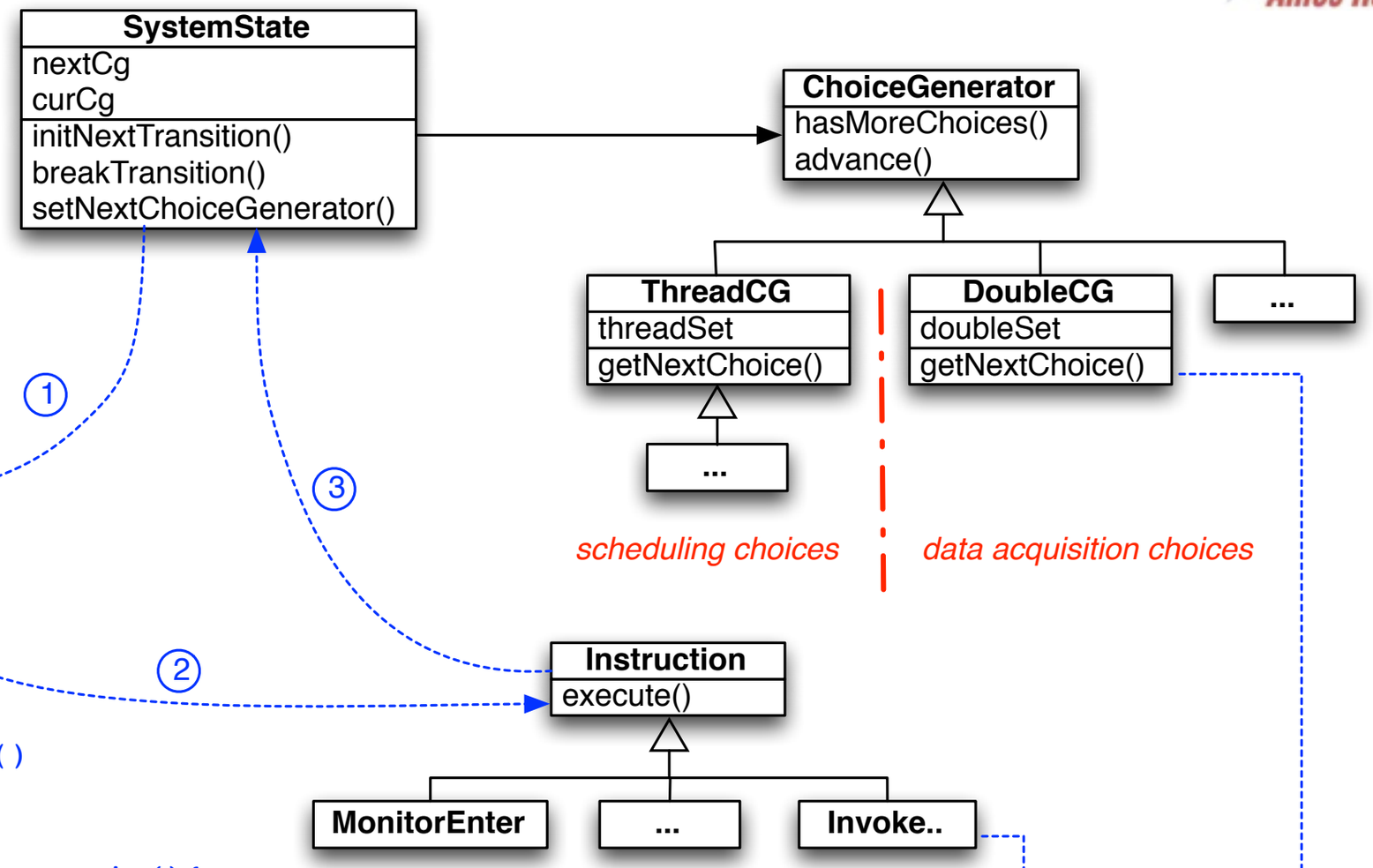
Thank You!

Background Slides

Basics: ChoiceGenerator Implementation

```
initNextTransition(){...
  curCg = nextCg
  nextCg = null
  curCg.advance()
   ..setExecThread()
  ...
```

```
breakTransition(){...
  return nextCg != null
```




```
executeTransition(){..
  isFirstStepInsn = true
  while (pc != null) {
    nextPc = executeInstruction()
    if (ss.breakTransition())
      break
    else
      pc = nextPc
  isFirstStepInsn = false
```

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

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()
}
```

top half

bottom half

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

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

Basics: MJI - Implementation

JPF (model) class

```
...
int a = c.foo(3);
```

```
...
aload_1
icont_3
invokevirtual ..
```

```
package x.y.z;
class C {
  ...
  native int foo (int p);
}
```

JPF class loading

JPF method invocation

NativePeer
methods
executeMethod()

ClassInfo
peerCls
executeMethod()

```
executeMethod (ThreadInfo ti..){
  MJIEnv env = ti.getMJIEnv();
  Object[] args = getArguments();
  ..
  mth.invoke(peerCls, args);
  ..
}
```

Java reflection call

```
ClassInfo (..){
  peerCls = loadNativePeer(..);
  ..
}
```

MJIEnv
threadInfo
getXField(..)
setXField(..)
...

ThreadInfo
env

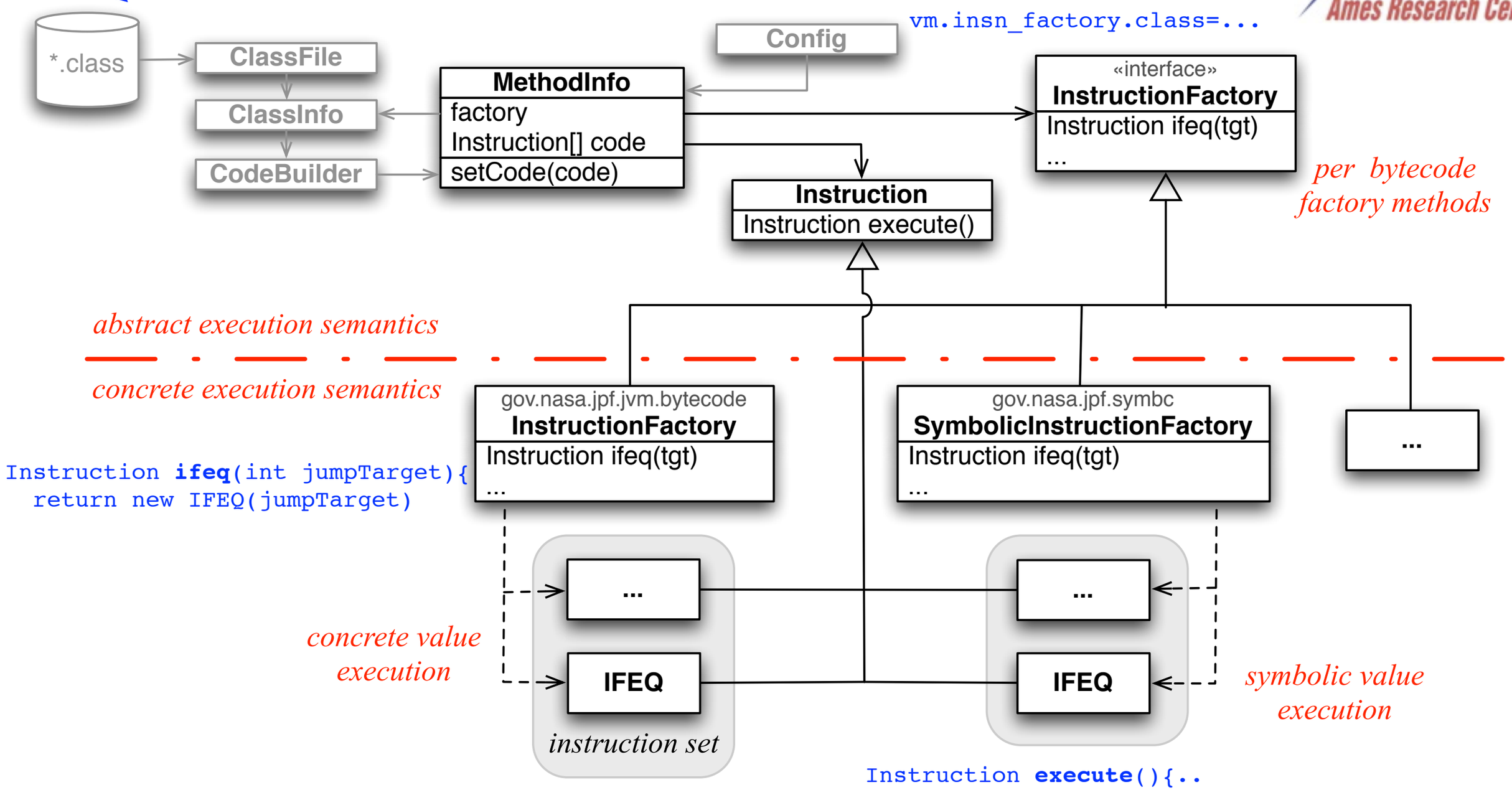
JPF object access

reflection

```
class JPF_x_y_z_C {
  ...
  public static int foo__I__I (MJIEnv env, int thisRef, int p) {
    int d = env.getIntField(thisRef, "data");
    ..
  }
}
```

JVM (Java) class

Basics: Instruction Factory Implementation



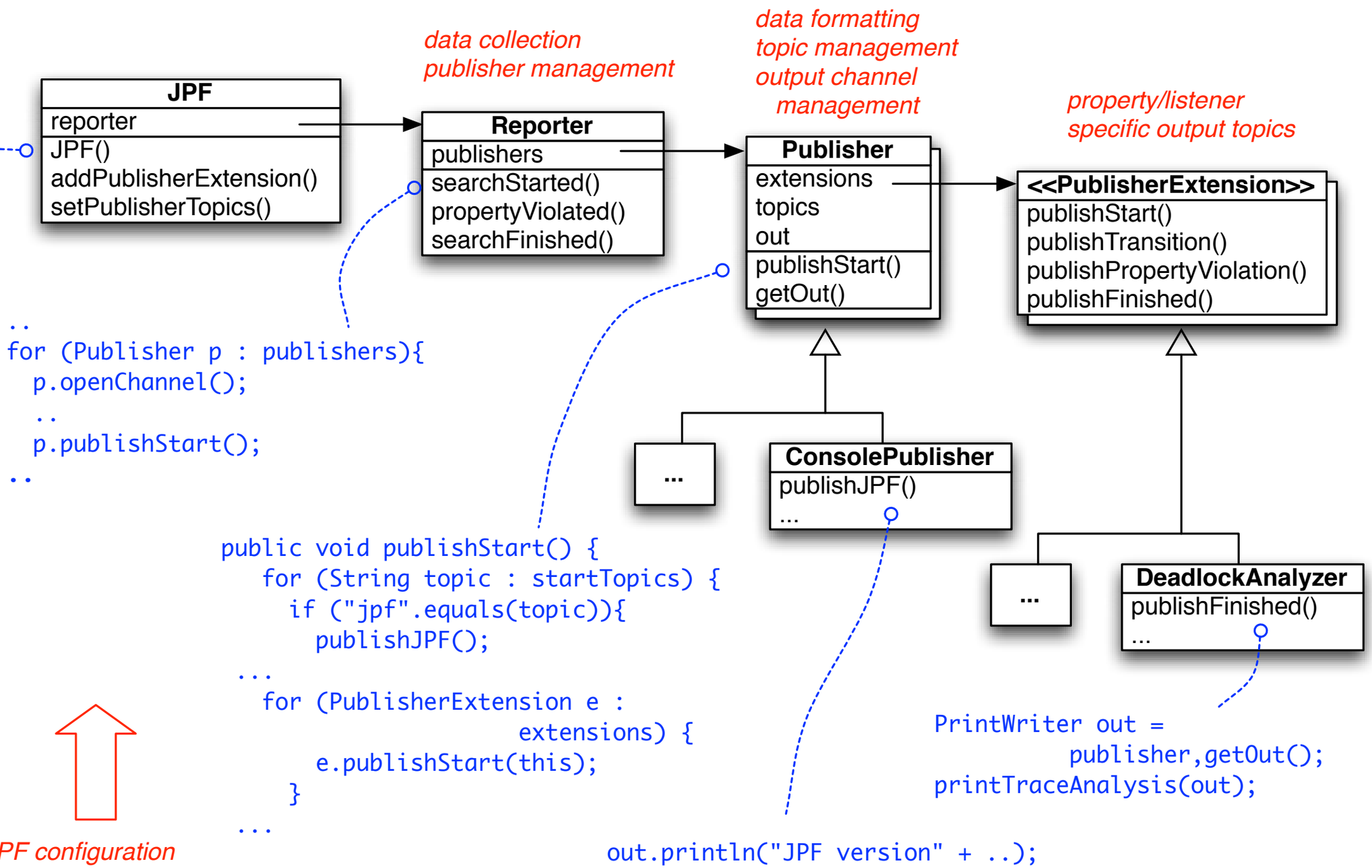

```
Instruction ifeq(int jumpTarget){
  return new IFEQ(jumpTarget)
```

```
Instruction execute (){..
  cond = popCondition()
  if (cond)
    return jumpTarget
  else
    return getNextInsn()
```

```
Instruction execute(){..
  if (!firstStepInsn()){
    setNextCG(new PCChoiceGenerator())
    return this
  }
  popCondition() // not interested
  cond = getCG().getNextChoice()
  if (cond){...
    return jumpTarget
  } else {...
    return getNextInsn()
```

Basics: Publisher Implementation

```
..
reporter = config.getInstance("report.class", Reporter.class,..);
..
```




```
..
for (Publisher p : publishers){
  p.openChannel();
  ..
  p.publishStart();
..
```

```
public void publishStart() {
   for (String topic : startTopics) {
     if ("jpf".equals(topic)){
       publishJPF();
 ...
   for (PublisherExtension e :
                        extensions) {
       e.publishStart(this);
     }
 ...
```

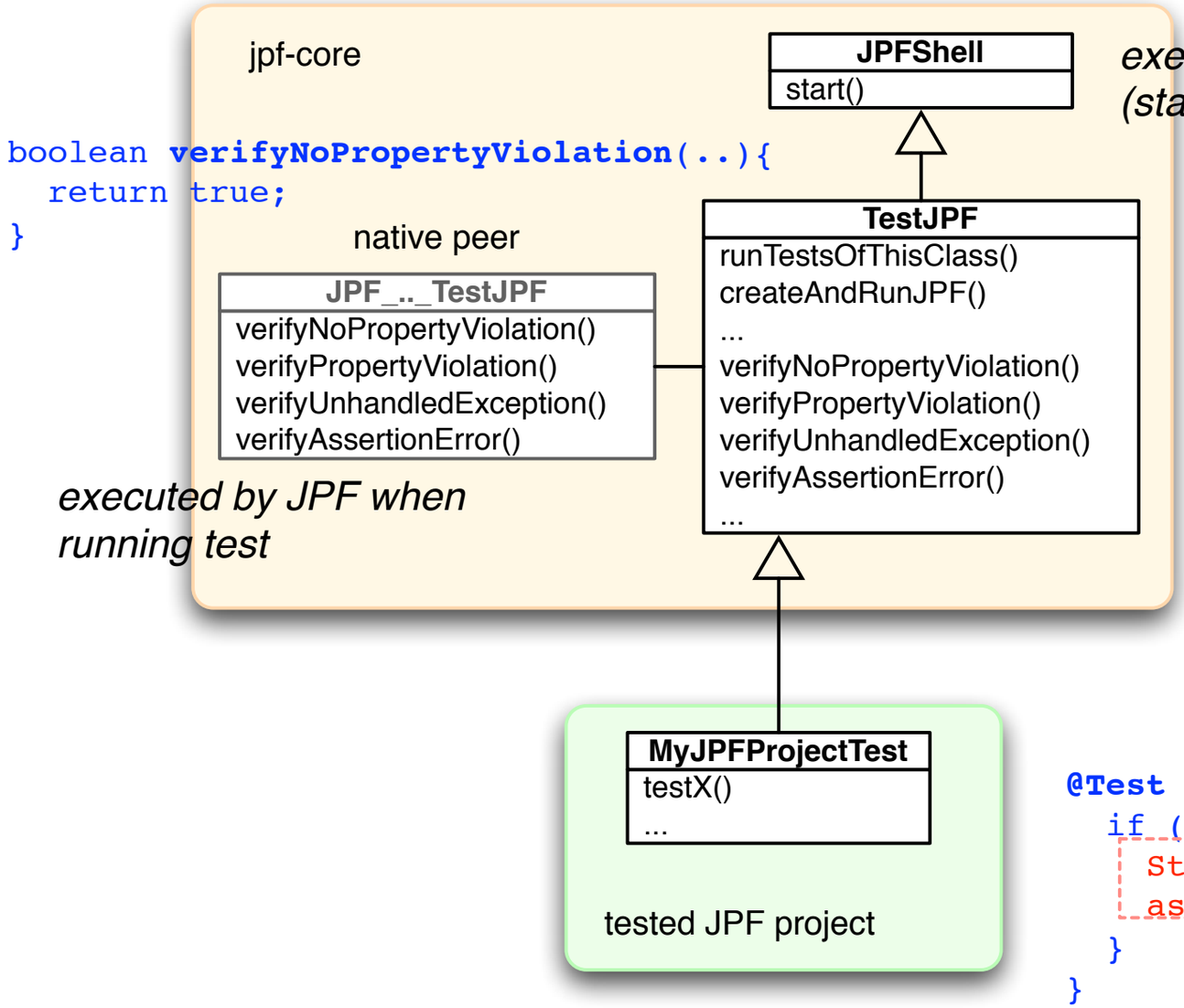
```
PrintWriter out =
          publisher,getOut();
printTraceAnalysis(out);
```

```
out.println("JPF version" + ..);
```

JPF configuration
*(e.g. jpf.properties or *.jpf files)*

```
report.class=.report.Reporter
report.publisher=console,..
report.console.class=.report.ConsolePublisher
report.console.start=jpf,..
```

Basics: TestJPF Implementation




```
boolean verifyNoPropertyViolation (String... jpfArgs) {
  ...
  args = append(jpfArgs, caller.getClassName(),
                         caller.getMethodName());
    ..
    JPF jpf = createAndRunJPF(args);
    ...
    errors = jpf.getSearchErrors();
    if (!errors.isEmpty())
      throw new AssertionError(..);
    ...
  return false; // -> don't execute test block
}
```

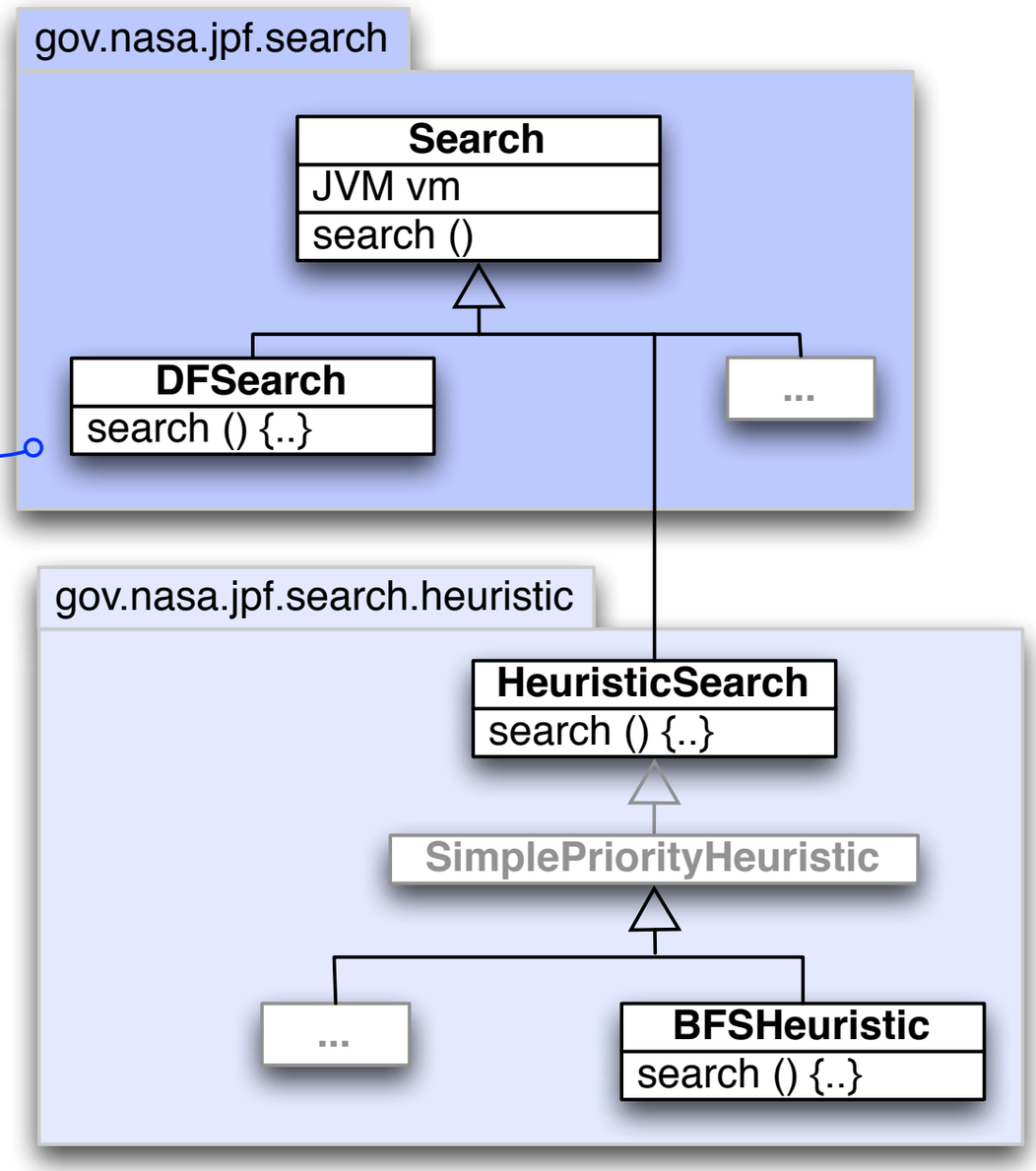
```
@Test public void testX() {
  if (verifyNoPropertyViolation()) {
    String s = "one" + " two";
    assert "one two".equals(s);
  }
}
```

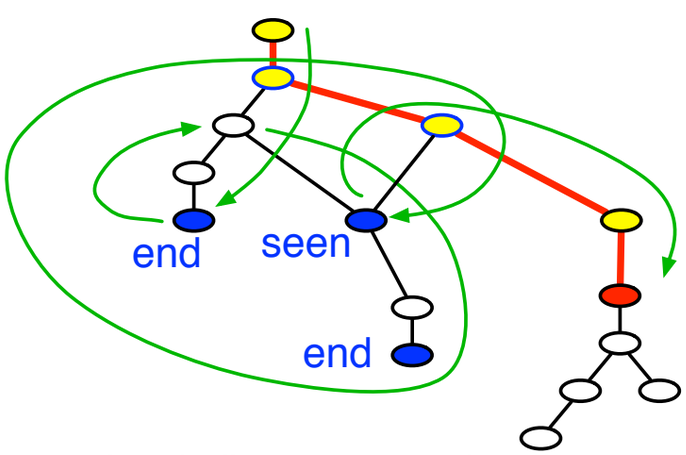
code verified by JPF

Basics: Search Policies

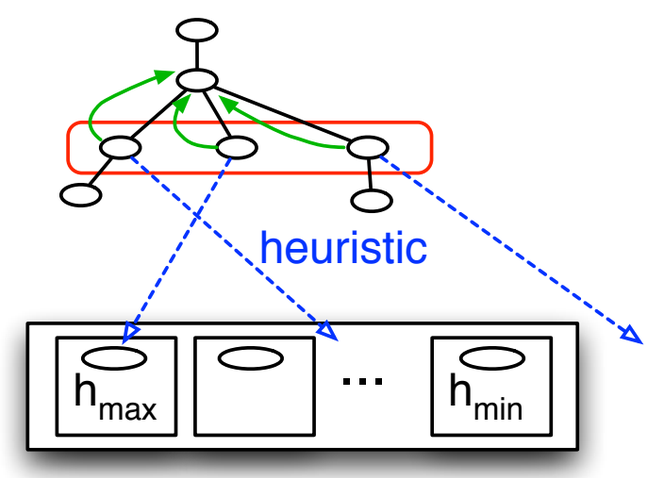
- state explosion mitigation: search the interesting state space part first ("get to the bug early, before running out of memory")
- Search instances encapsulate (configurable) search policies

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





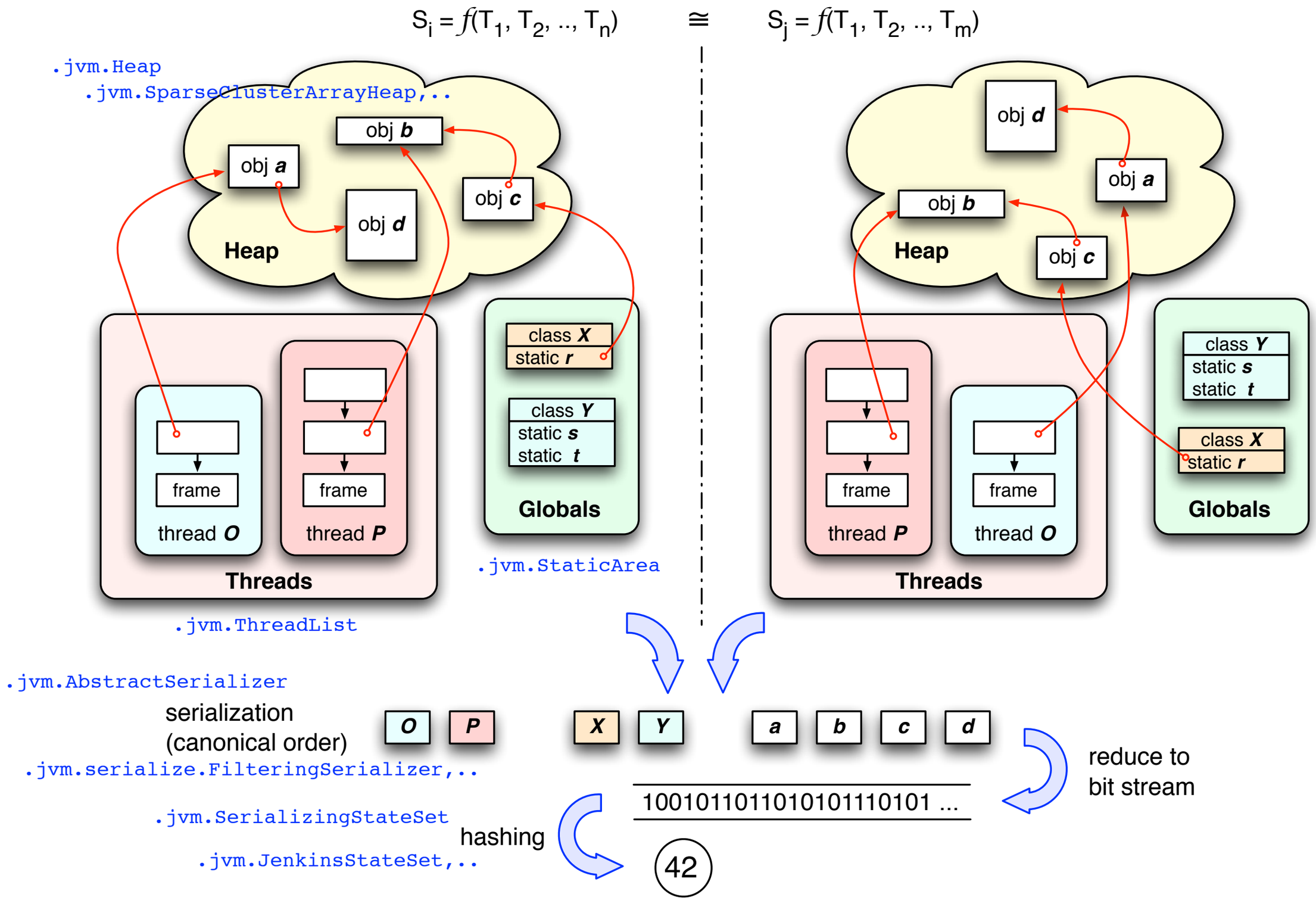
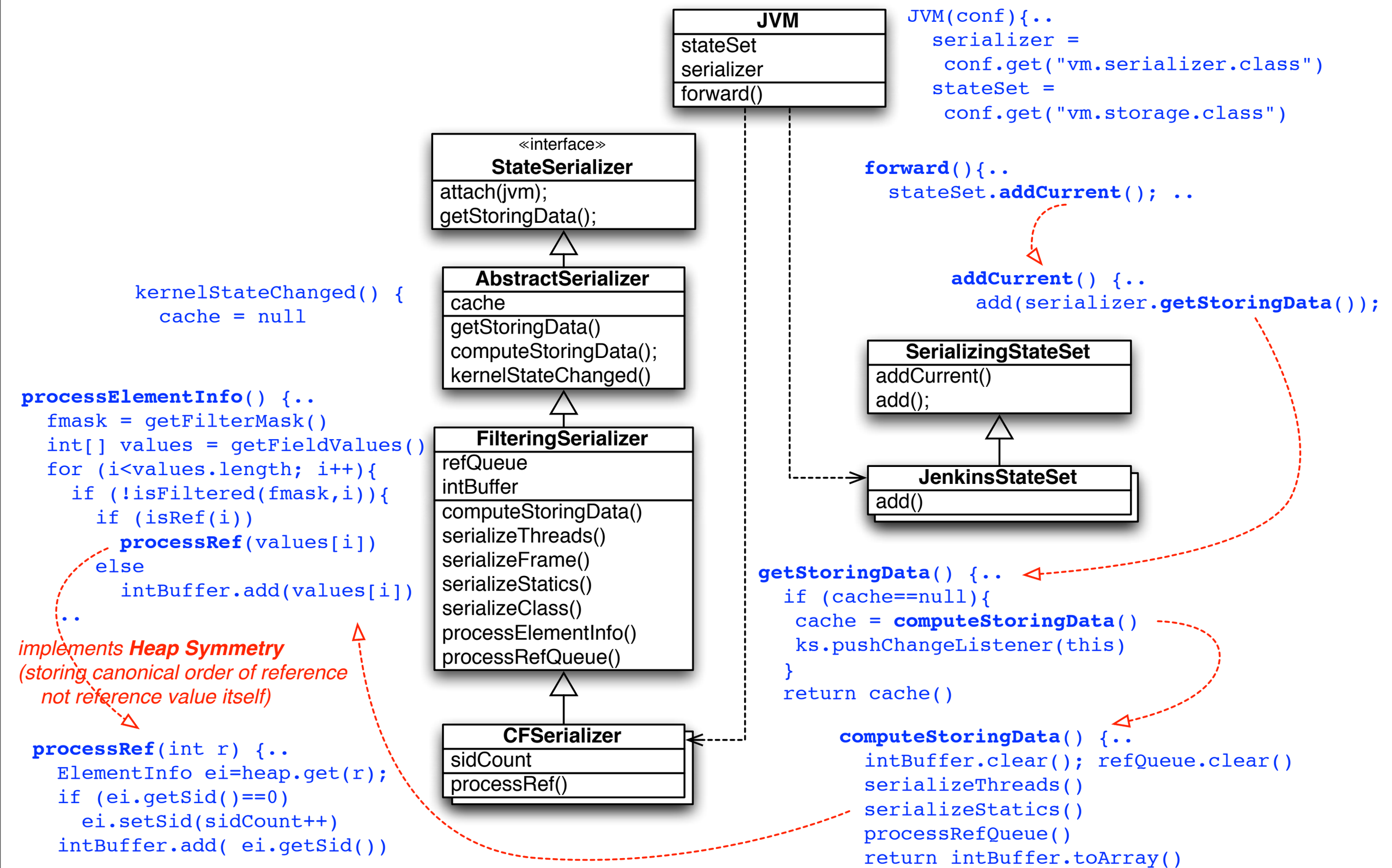
depth first traversal
optional state matching



sorted state queue (bounded)

Basics: Serialization

- don't produce different states for property irrelevant permutations



Basics: Serialization Implementation