Java Pathfinder (JPF)

all info from:
http://babelfish.arc.nasa.gov/trac/jpf

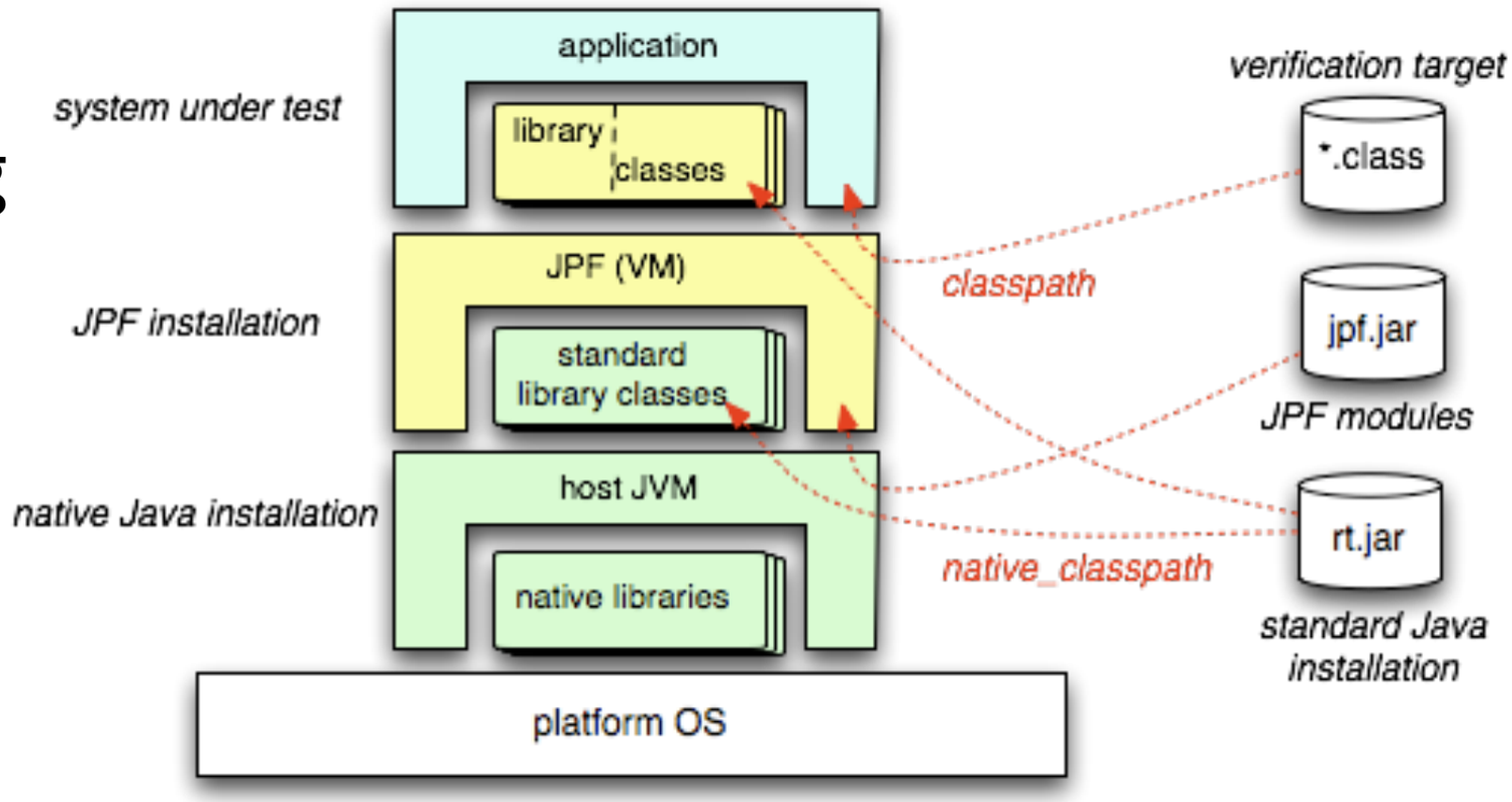
Volker Stolz

INF 2140
Parallel Programming
Vår 2012

JPF-*"the swiss army knife of Java verification"*

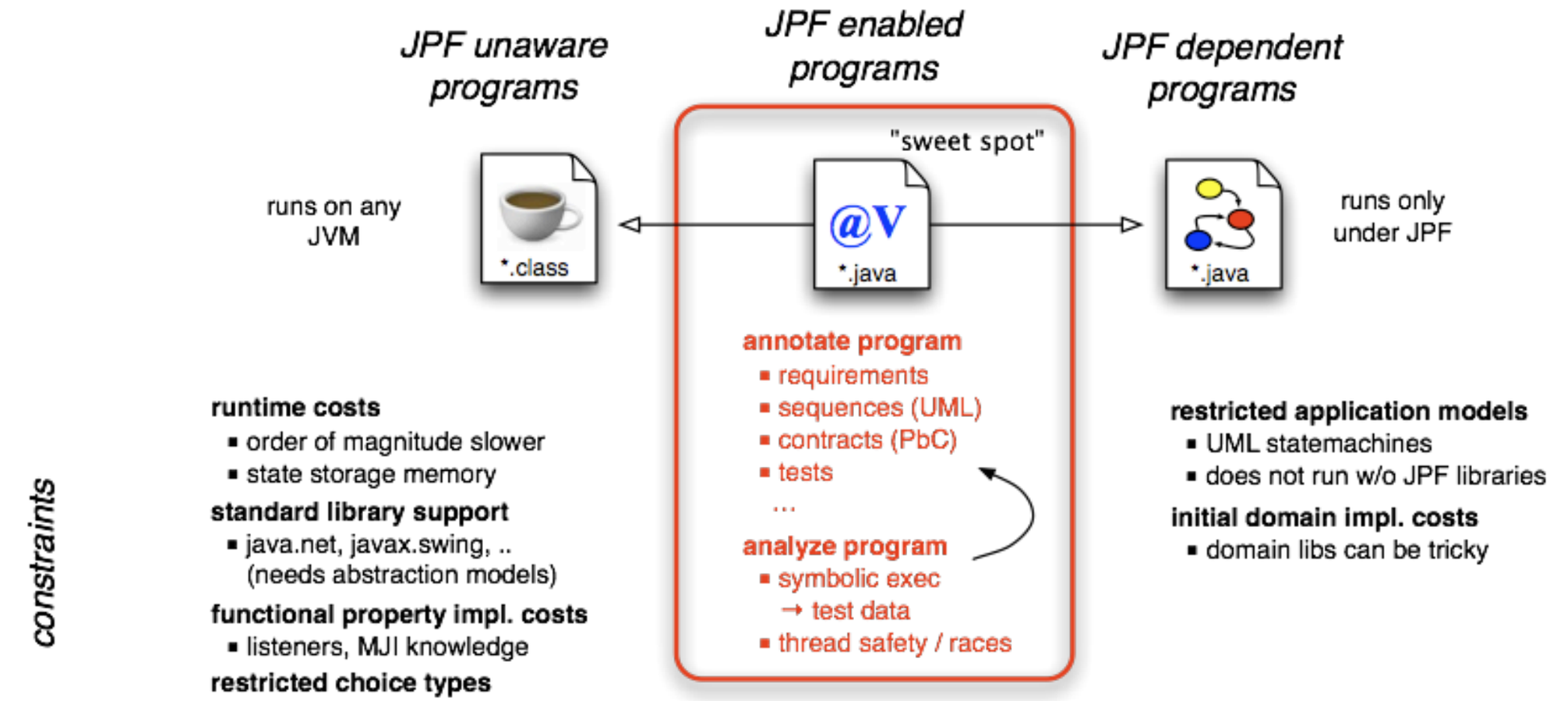
Useful for:

- software model checking
- testing
- highly customisable (value generators etc.)
- *highly confusing :-)*



JPF: Java VM running on top of JVM

Application Types



Race Detection

```
public class RaceCondition {
  private static class Pair {
    String x = "x";
    String y = "y";

    public void update() {
      x = x + y + x;
    }
  }

private static class RC extends Thread
  {
  Pair p;

  public void run() {
    p.update();
  }
}
```

```
 public static void main(String[] args)
    throws Exception
{
    Pair p = new Pair();
    RC rc1 = new RC();
    RC rc2 = new RC();

    rc1.p = p;
    rc2.p = p;

    rc1.start();
    rc2.start();
    rc1.join();
    rc2.join();
    System.out.println("x: " + p.x);
  }
}
```

Deadlock Detection

```
public class MyRaceCondition {
  private static class Pair {
    String x = "x";

    String y = "y";
    String z = "";

    public void update() {
      x = x + y + x;
    }}

private static class RC1 extends Thread {
  Pair p;

  public void run() {
    synchronized (p.x) {
      synchronized (p.y) {
        p.update();
      }}}}

private static class RC2 extends Thread {
  Pair p;

  public void run() {
    synchronized (p.y) {
      synchronized (p.x) {
        p.update();
      }}}}
```

```
public static void main(String[] args)
  throws Exception
{
  Pair p = new Pair();
  RC1 rc1 = new RC1();
  RC2 rc2 = new RC2();

  rc1.p = p;
  rc2.p = p;

  rc1.start();
  rc2.start();
  rc1.join();
  rc2.join();
  System.out.println("x: " + p.x);
}}
```

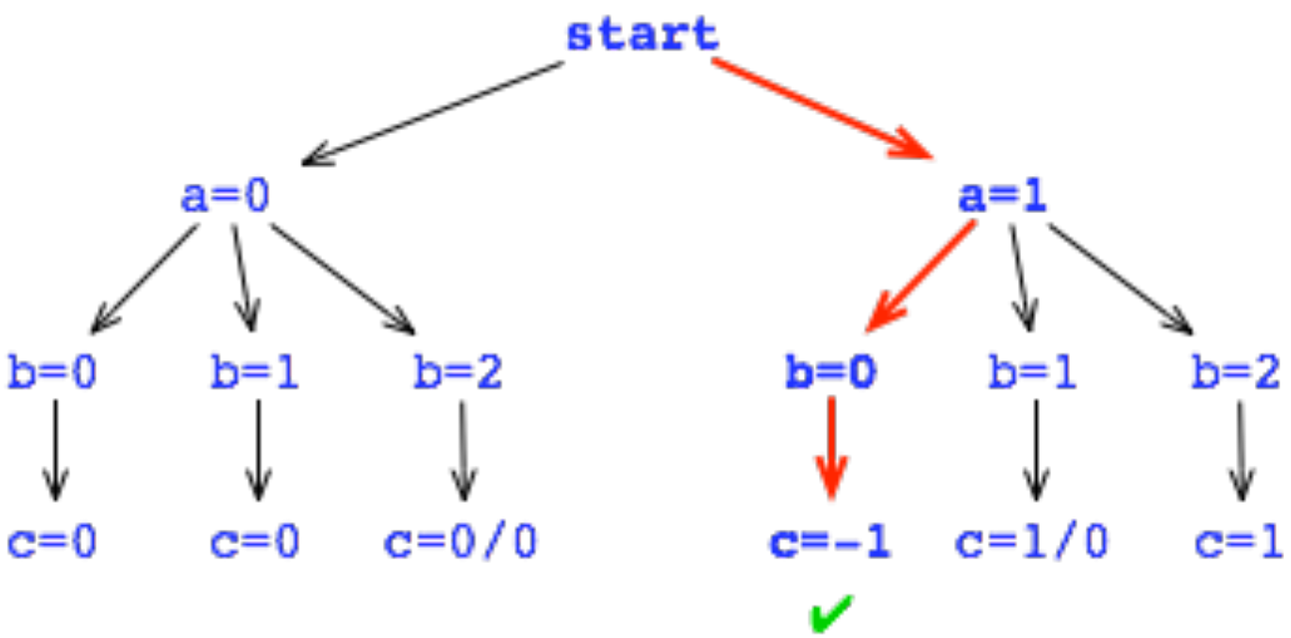
D.Bushnell/JPF Workshop 2008

Verification with Data (1)

```
> bin/jpf +cg.enumerate_random=true Rand
JavaPathfinder v4.1 - (C) 1999-2007 RIACS/NASA Ames Research Center
====================================================== system under test
application: /Users/pcmehlitz/tmp/Rand.java

====================================================== search started: 5/23/07 11:49 PM
a=0
  b=0
    c=0
  b=1
    c=0
  b=2

====================================================== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.ArithmeticException: division by zero
        at Rand.main(Rand.java:15)
....
>
```



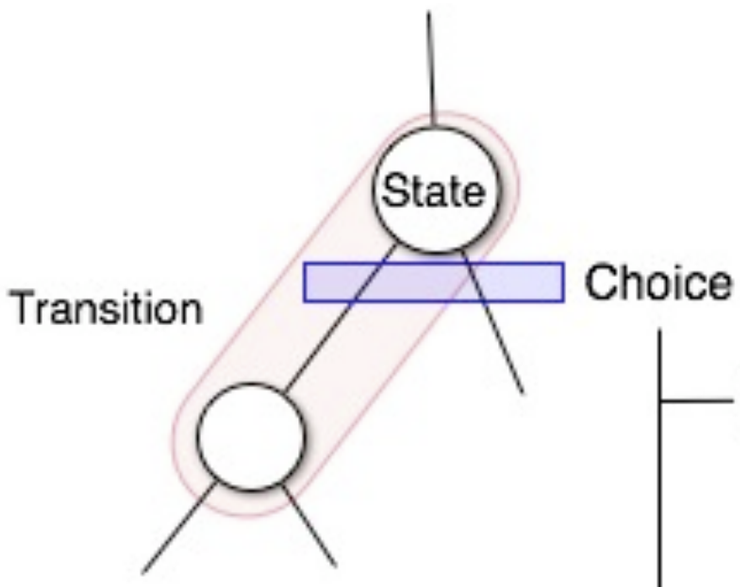
① `Random random = new Random()`

② `int a = random.nextInt(2)`

③ `int b = random.nextInt(3)`

④ `int c = a/(b+a -2)`

Verification with Data (2)



Scheduling Choice

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

Data Choice

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

Verification with Data (3)

`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

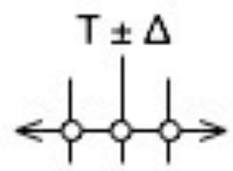
xChoiceGenerator
choiceSet: {x}
hasMoreChoices() advance() getNextChoice() → x

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");
..
```

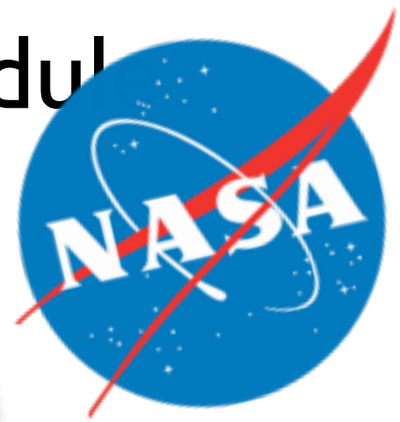
configuration (e.g. mode property file)

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

JPF Limitations

- "Usual" state space explosion problem
  ⇒ use partial-order-reduction modul
- Some assembly re
- R ing, GUIs like AWT/SWING

Try it out yourself:
http://babelfish.arc.nasa.gov/trac/jpf



Workaround:

- Run JPF on simplified program:
  "model" or units