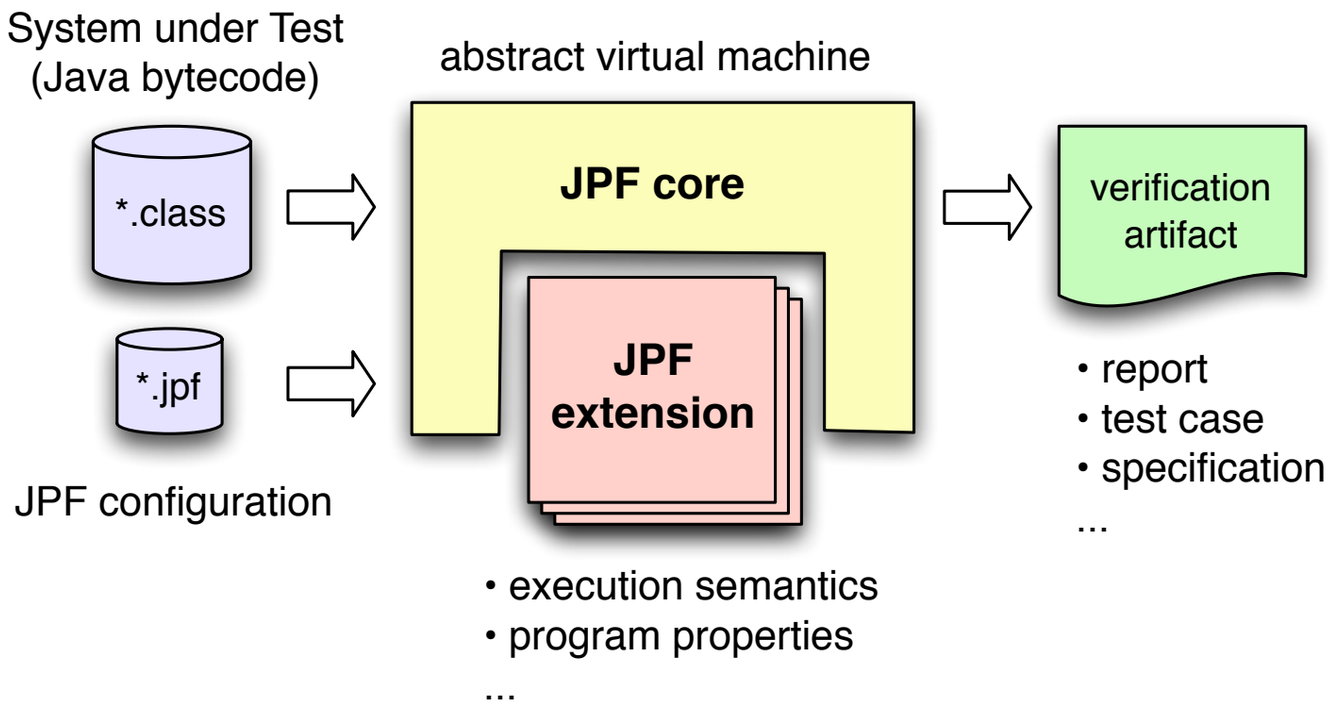
Java Pathfinder Session 3:
Under the Hood - Building *Your* JPF

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

Recap: JPF - What is it?

- a model checker? a JVM? ..



- and the answer is: both, and more - it depends on you
- one thing it is not: a monolithic, black box tool

> No “one size fits all” - Extensibility is Paramount

- the quest of today: learn what is in the toolbox to find out how you can adapt JPF to *your* needs

Roadmap

- Basics (focused on writing extensions)
  - main stumbling block: VM inside VM
  - key design components & classes

- Organization:
  - runtime configuration scheme
  - where are the sources: directory structure

- Examples
  - listeners
  - native peer classes
  - instruction factories

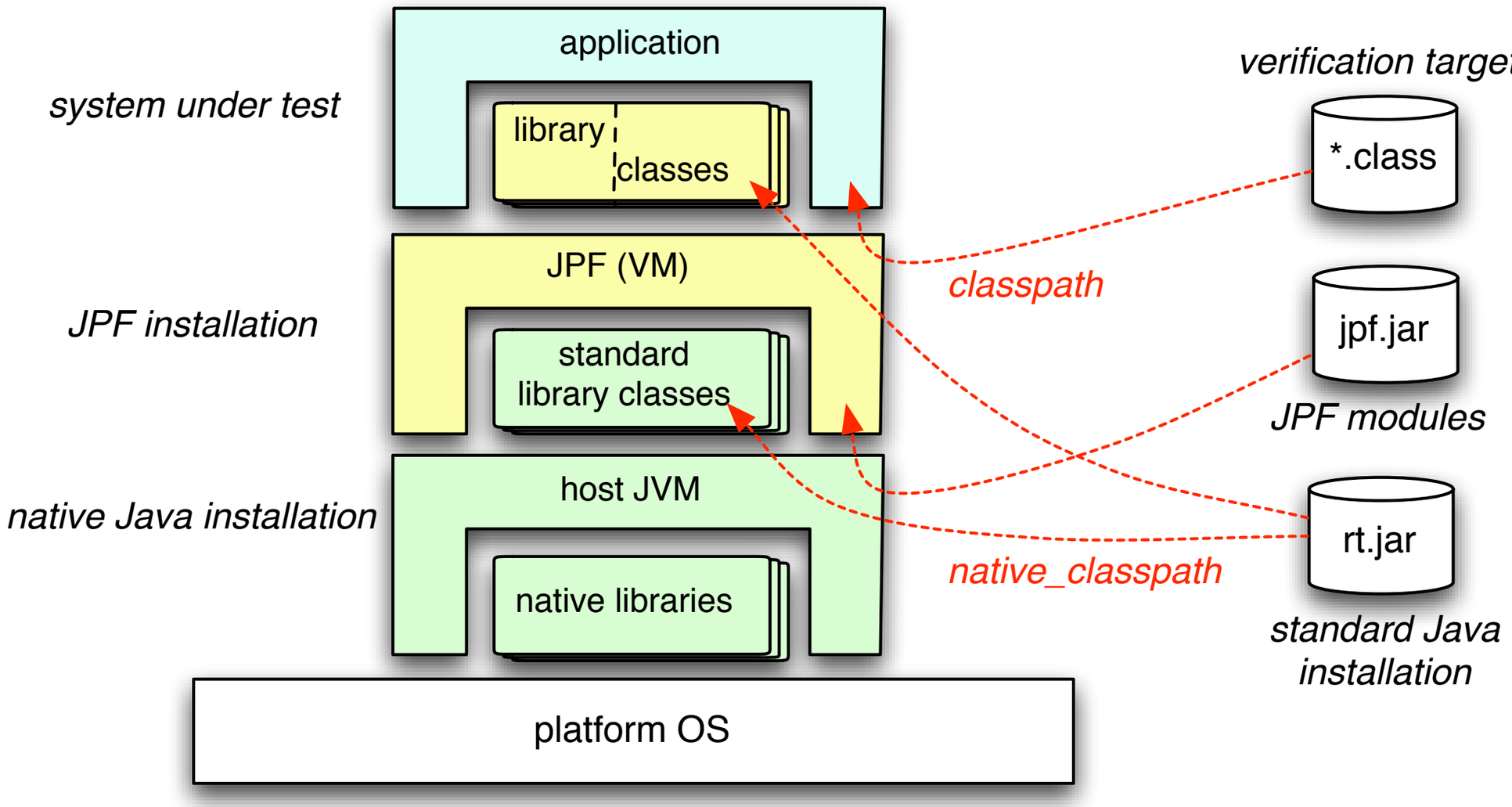
1. Basics

- main stumbling block: recursive nature of JPF

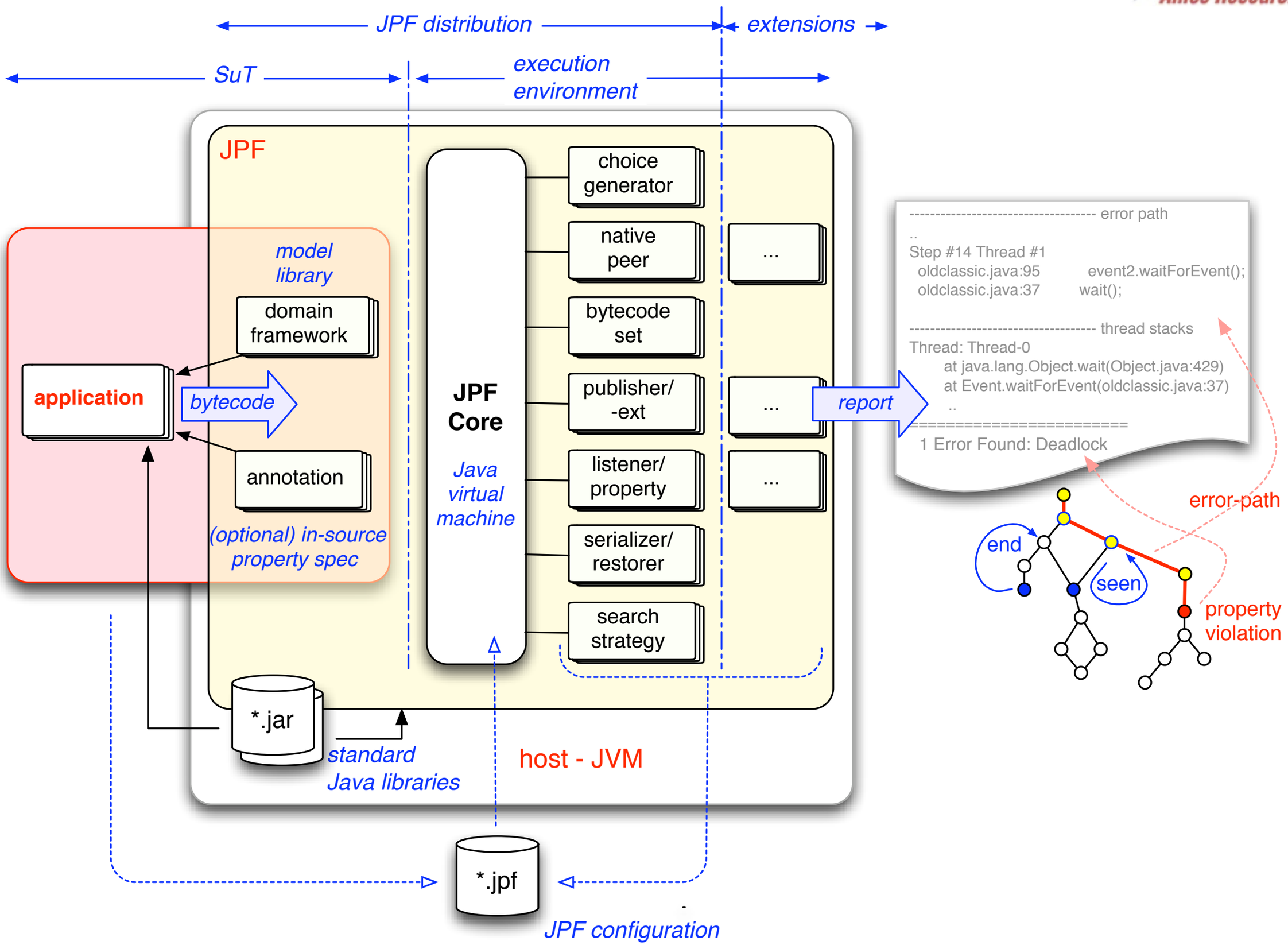
- key design components and classes (as needed for extensions)
  - Search and VM
  - ChoiceGenerators
  - Instruction Factories
  - Attributes
  - Listeners
  - native peers: Model-Java-Interface (MJI)

Basics: a VM running inside JVM

- 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
  ⇒ easy to get confused about who executes what



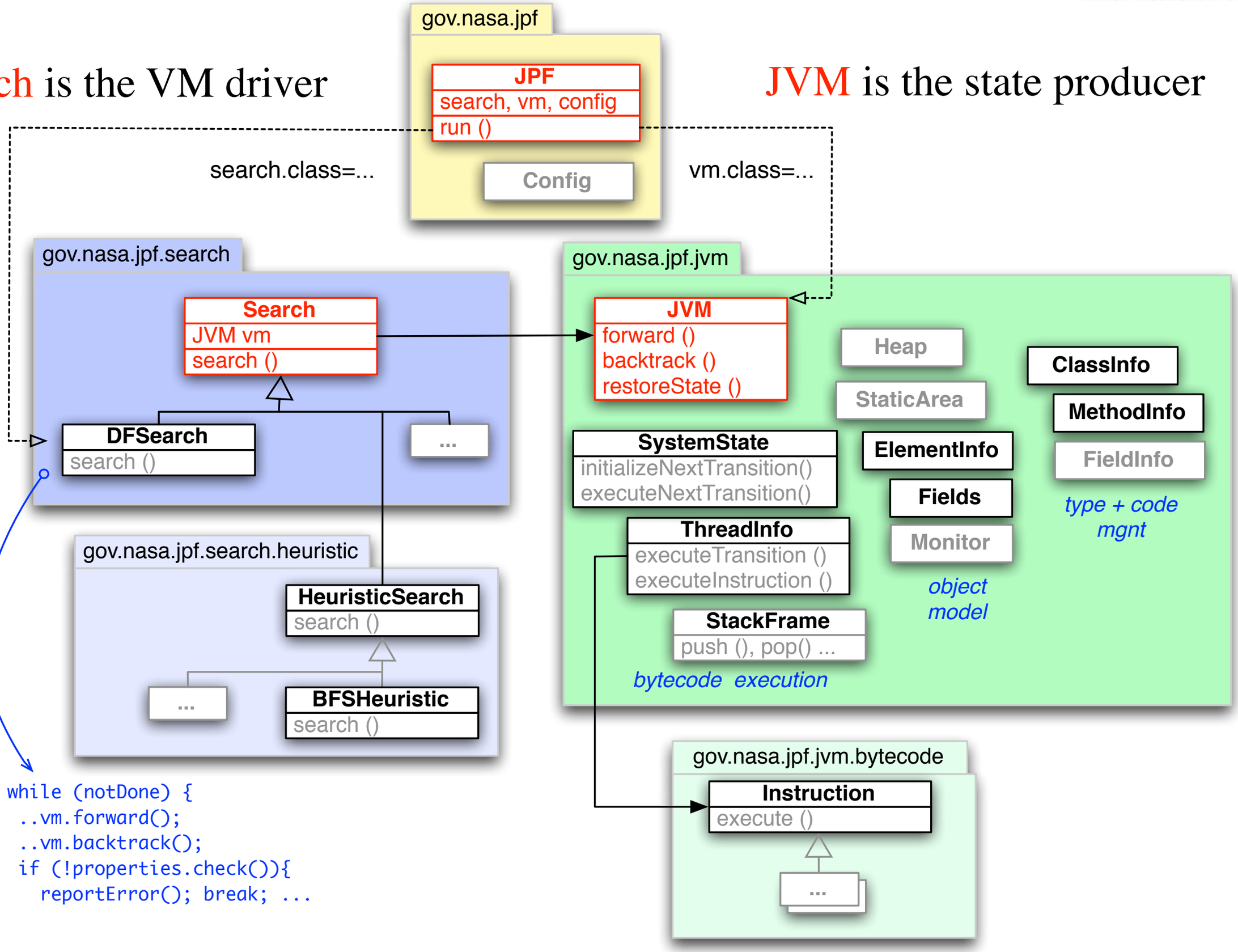
Basics: JPF Components



JPF Toplevel Structure

Search is the VM driver

JVM is the state producer

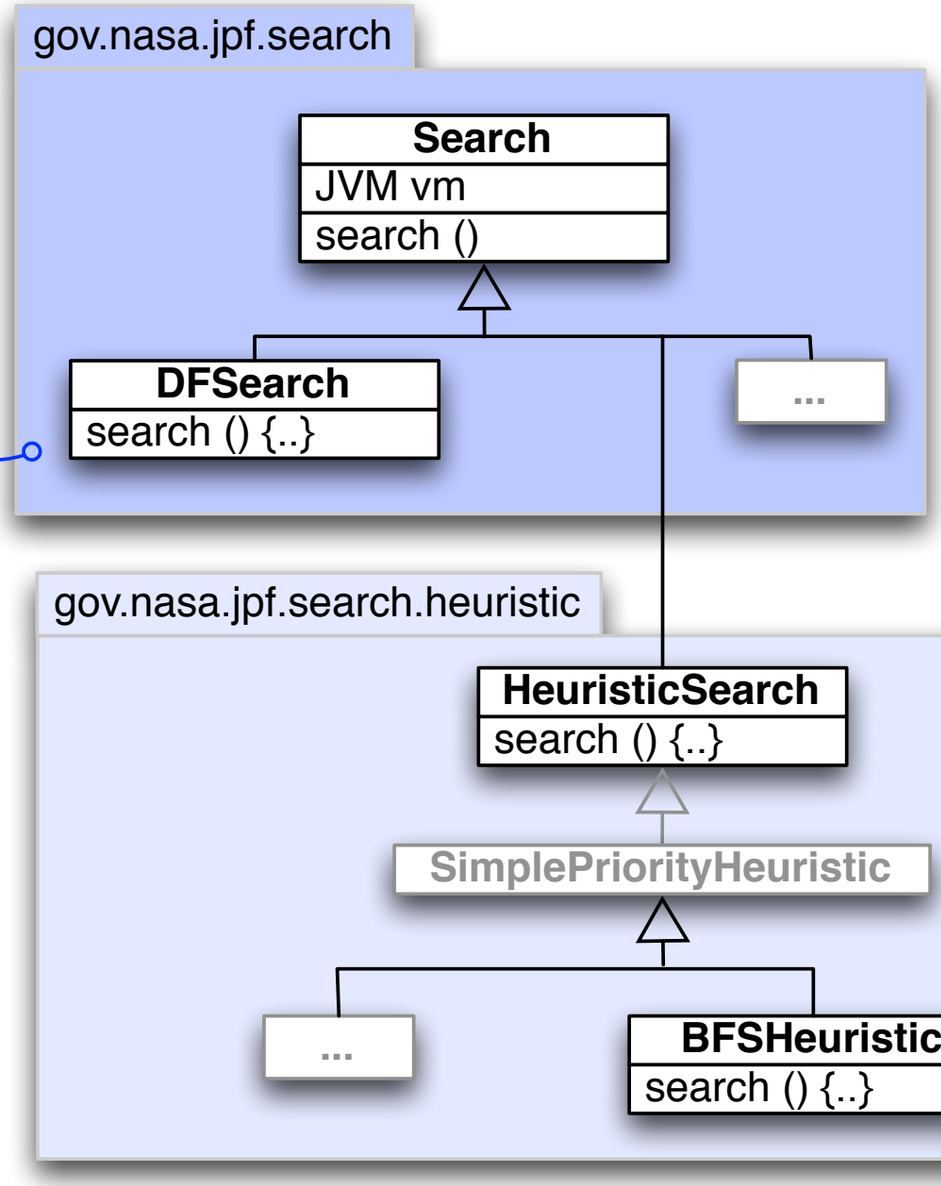


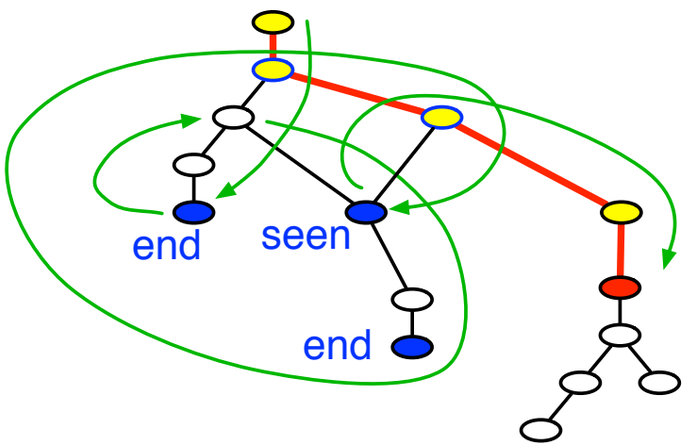

```
while (notDone) {
 ..vm.forward();
 ..vm.backtrack();
 if (!properties.check()){
   reportError(); break; ...
```

Basics: Search Policies

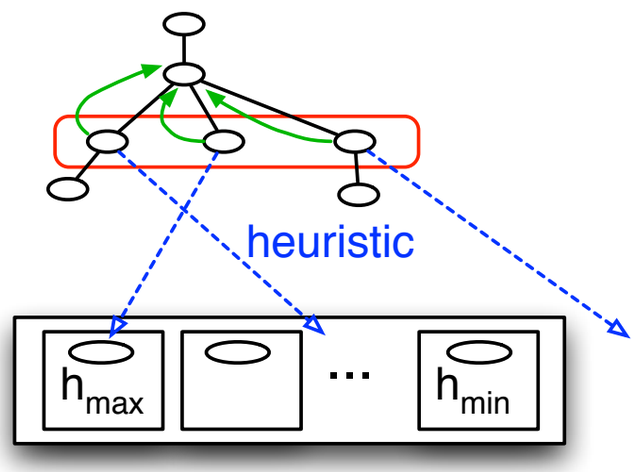
- state explosion mitigation: search the interesting state space part first ("get to the bug early, before running out of memory")
- Search instances encapsulate (configurable) search policies

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





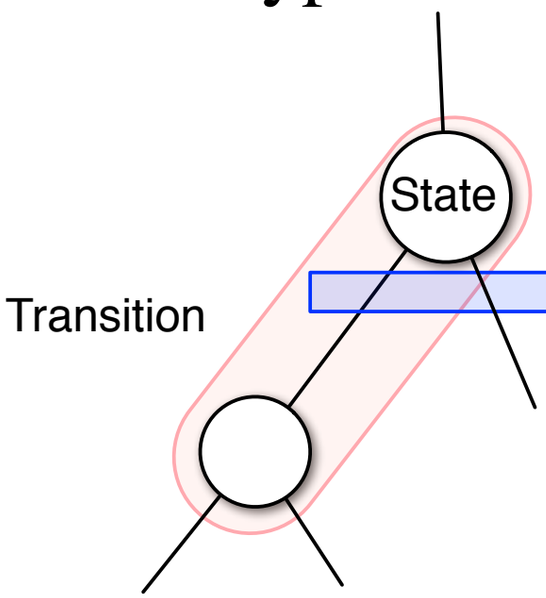
depth first traversal
optional state matching



sorted state queue (bounded)

Basics: Choice Generators

- 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



Scheduling Choice

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

Data Choice

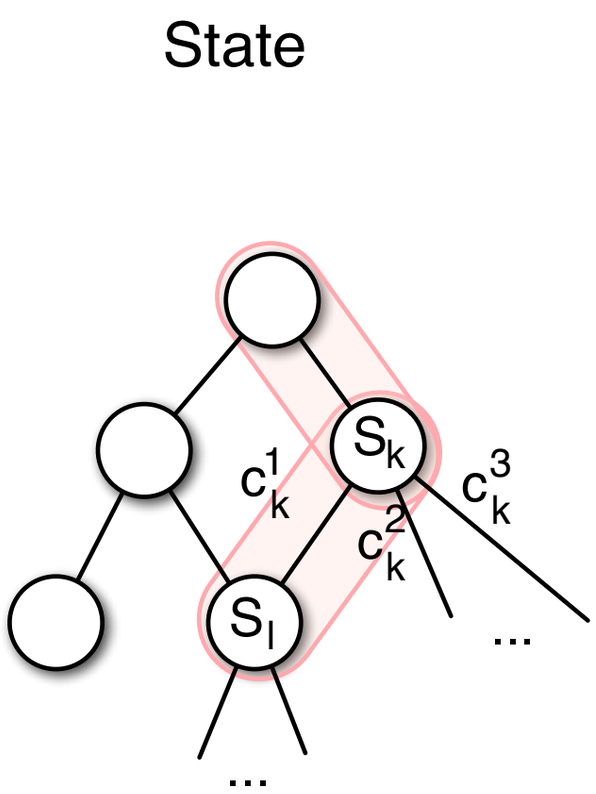
```
boolean b = Verify.getBoolean();
double d = Verify.getDouble("MyHeuristic");
..
```

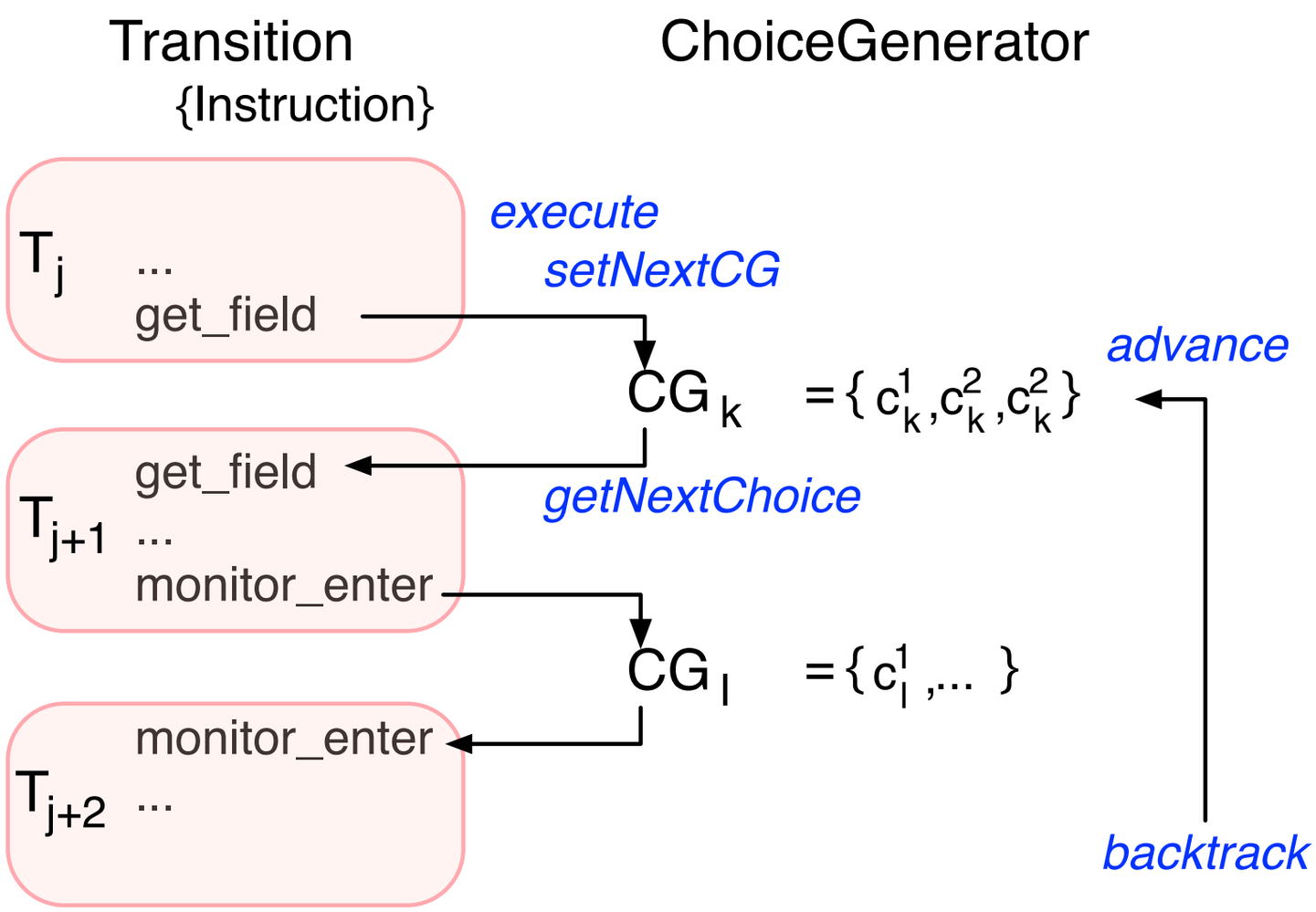
Control Choice

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

Basics: ChoiceGenerators & 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 Implementation

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

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

SystemState
nextCg
curCg
initNextTransition()
breakTransition()
setNextChoiceGenerator()

ChoiceGenerator
hasMoreChoices()
advance()

ThreadCG
threadSet
getNextChoice()

DoubleCG
doubleSet
getNextChoice()

...

...

scheduling choices

data acquisition choices

ThreadInfo
{pc, nextPc}
executeTransition()
executeInstruction()
isFirstStepInsn()

① ② ③

Instruction
execute()

MonitorEnter

...

Invoke..

MethodInfo

NativePeer

e.g. JPF_gov_nasa_jpf_jvm_Verify.getBoolean(env)

```
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);
```

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

***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*

top half

bottom half

Basics: Instruction Factories

- JVM is (mostly) agnostic to what `Instruction.execute()` does
- concrete `Instruction` class hierarchy represents execution semantics
- can be configured at startup time and replaced at runtime (`MethodInfo` keeps code as replaceable Instruction array)
- JVM 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
- overflow example:

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);
```

JPF configuration

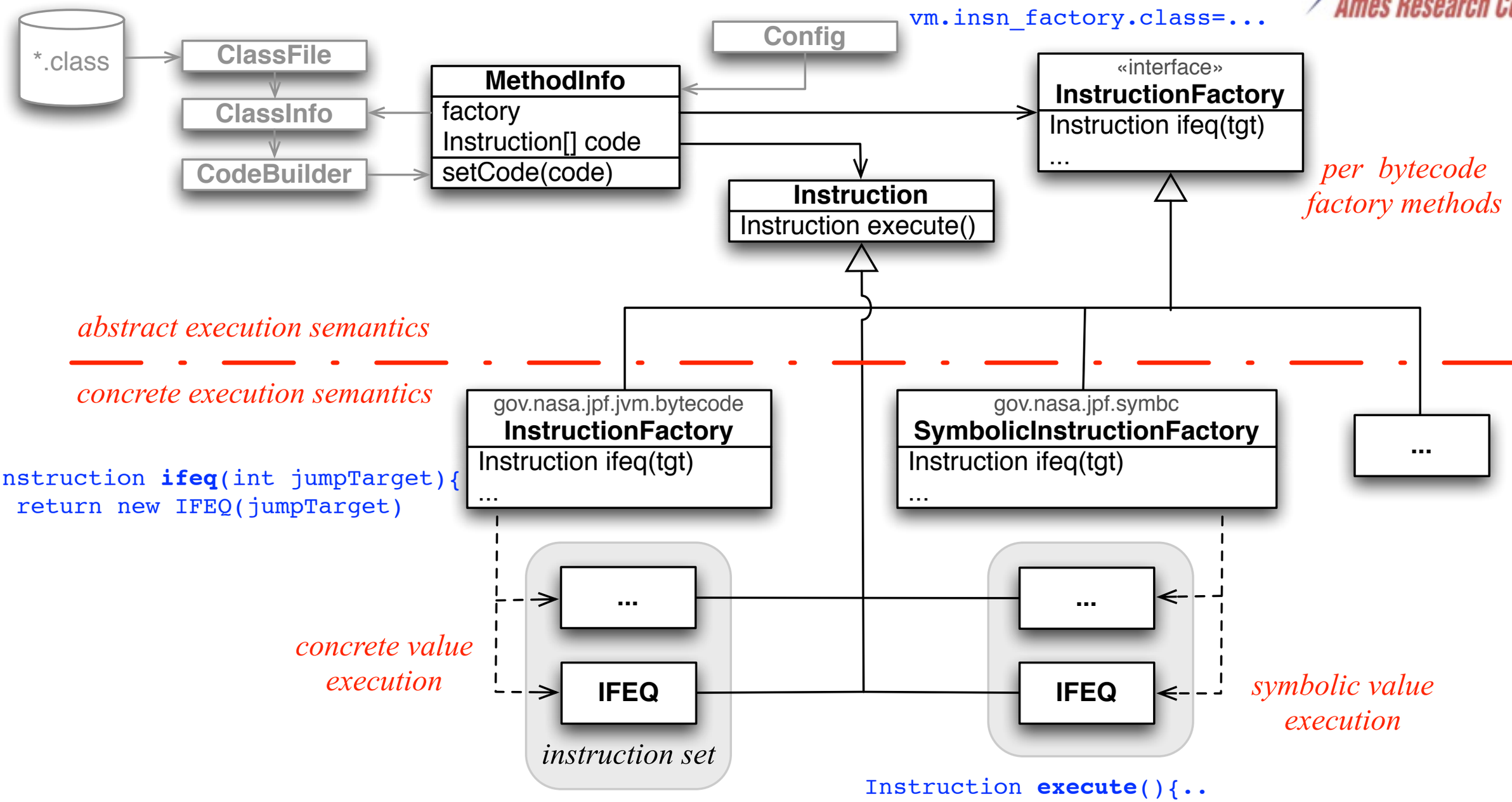
```
vm.insn_factory.class =
    .numeric.NumericInstructionFactory
```

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: Instruction Factory Implementation




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

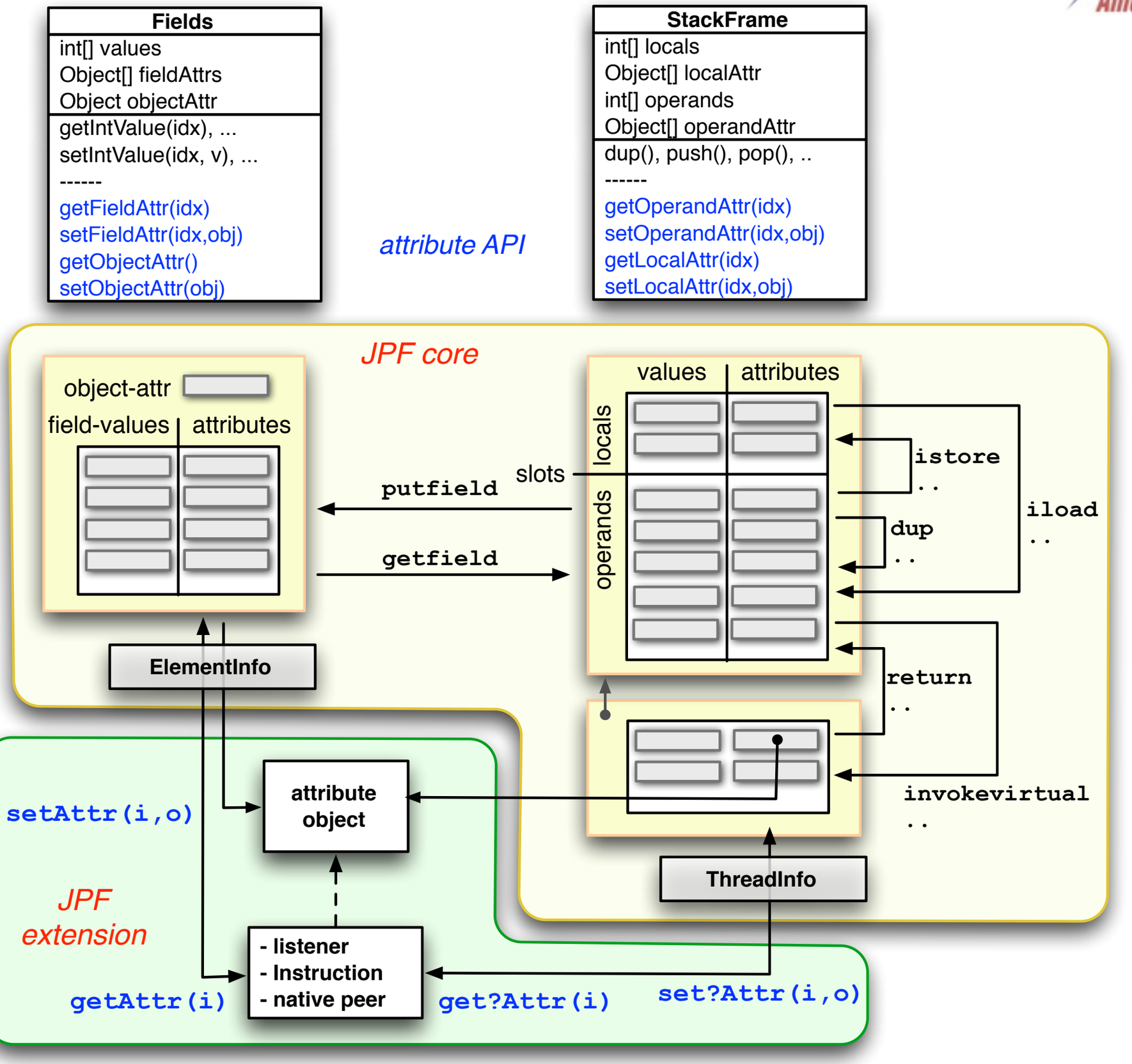
```
Instruction execute (){..
  cond = popCondition()
  if (cond)
    return jumpTarget
  else
    return getNextInsn()
```

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

Basics: 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

Basics: Attribute Implementation



Basics: Native Peers

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

x.y.MyClass $_{JPF}$ → peer.package. JPF_x_y_MyClass $_{hostVM}$

- native peer methods can be used for more than native code:
  - atomic
  - 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

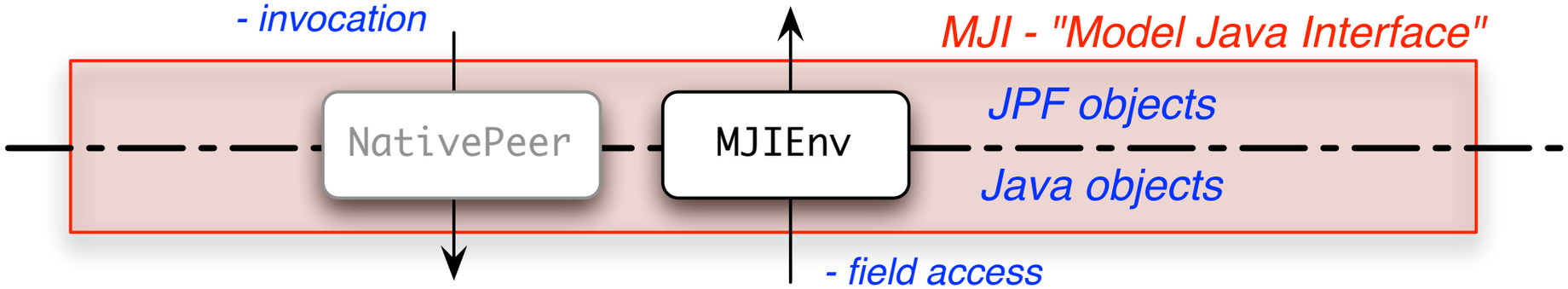
- execution lowering from JPF executed code into JVM executed code

"Model" Class

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

JPF Class

- *method lookup*
- *parameter conversion*
- *invocation*



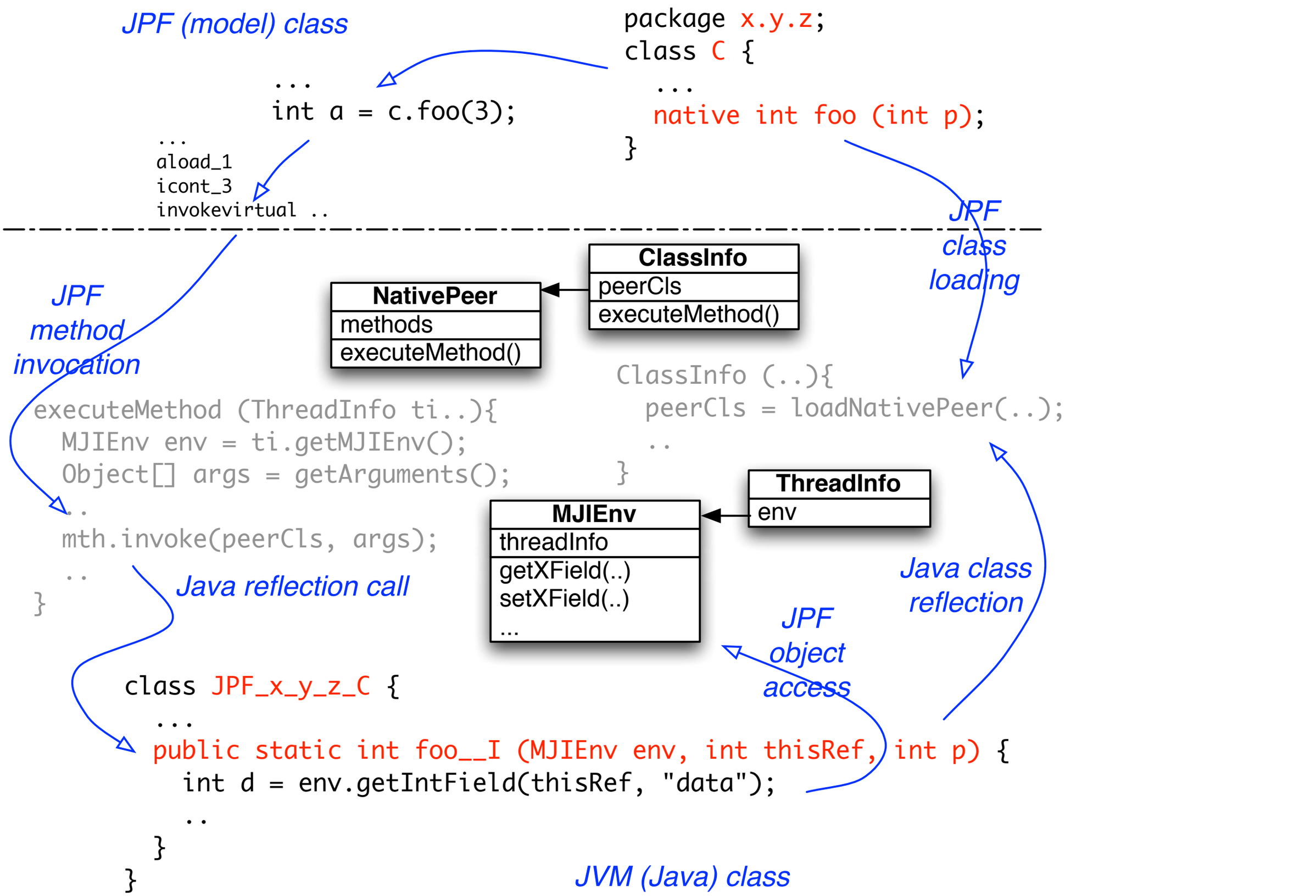
- *field access*
- *object conversion*
- *JPF intrinsics access*

```
class JPF_x_y_z_MyClass {
  public static
      int foo__ILjava_lang_String__2 (MJIEnv env, int objRef,
                                      int i, int sRef) {
    String s = env.getStringObject(sRef);
    ..
    int ref = env.newString(..);
    return ref;
  }
}
```

Java Class

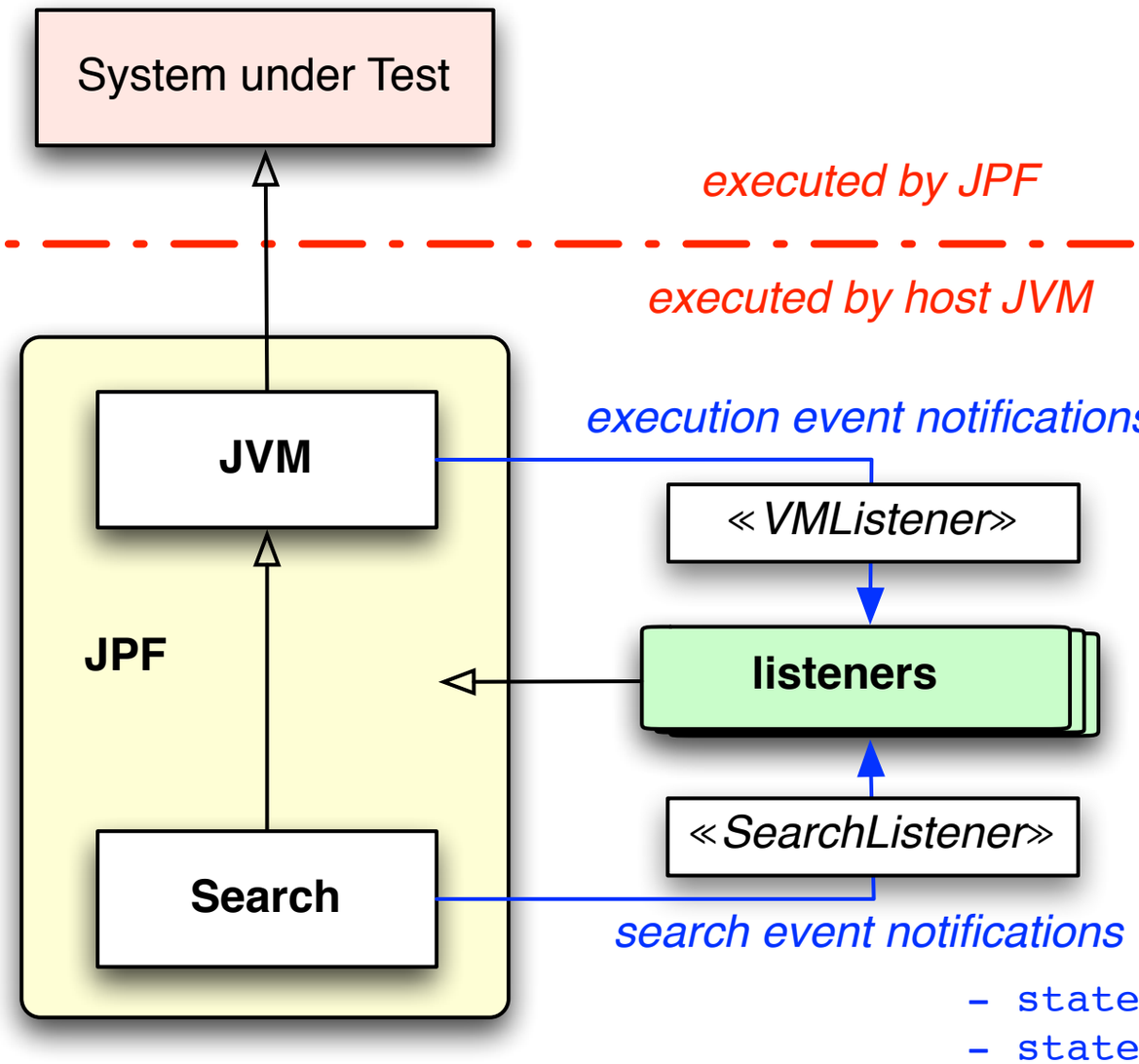
"NativePeer" Class

Basics: MJI - Implementation



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



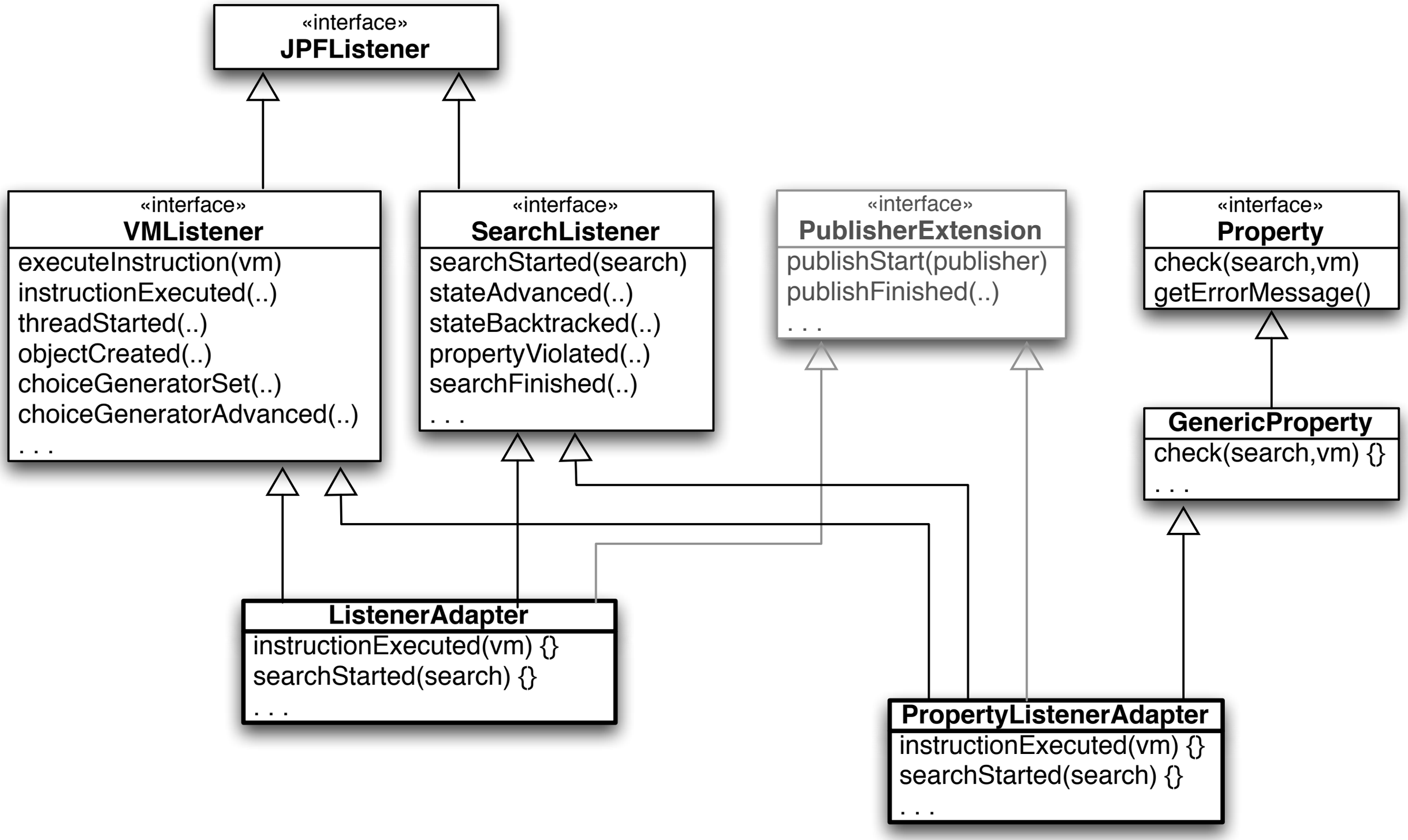

```
- classLoaded (vm)
- threadScheduled (vm)
- threadNotified (vm)
  ...
- executeInstruction (vm)
- instructionExecuted (vm)
  ...
- objectCreated (vm)
  ...
- exceptionThrown(vm)
  ...
- choiceGeneratorAdvanced (vm)
  ...
```

configured

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

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

Basics: Listener Implementation



2. Organization

- JPF configuration system - how can I customize JPF?
- project layout & directory structure - where do I find files?

- [and more that is outside this scope: test infrastructure, documentation etc.]

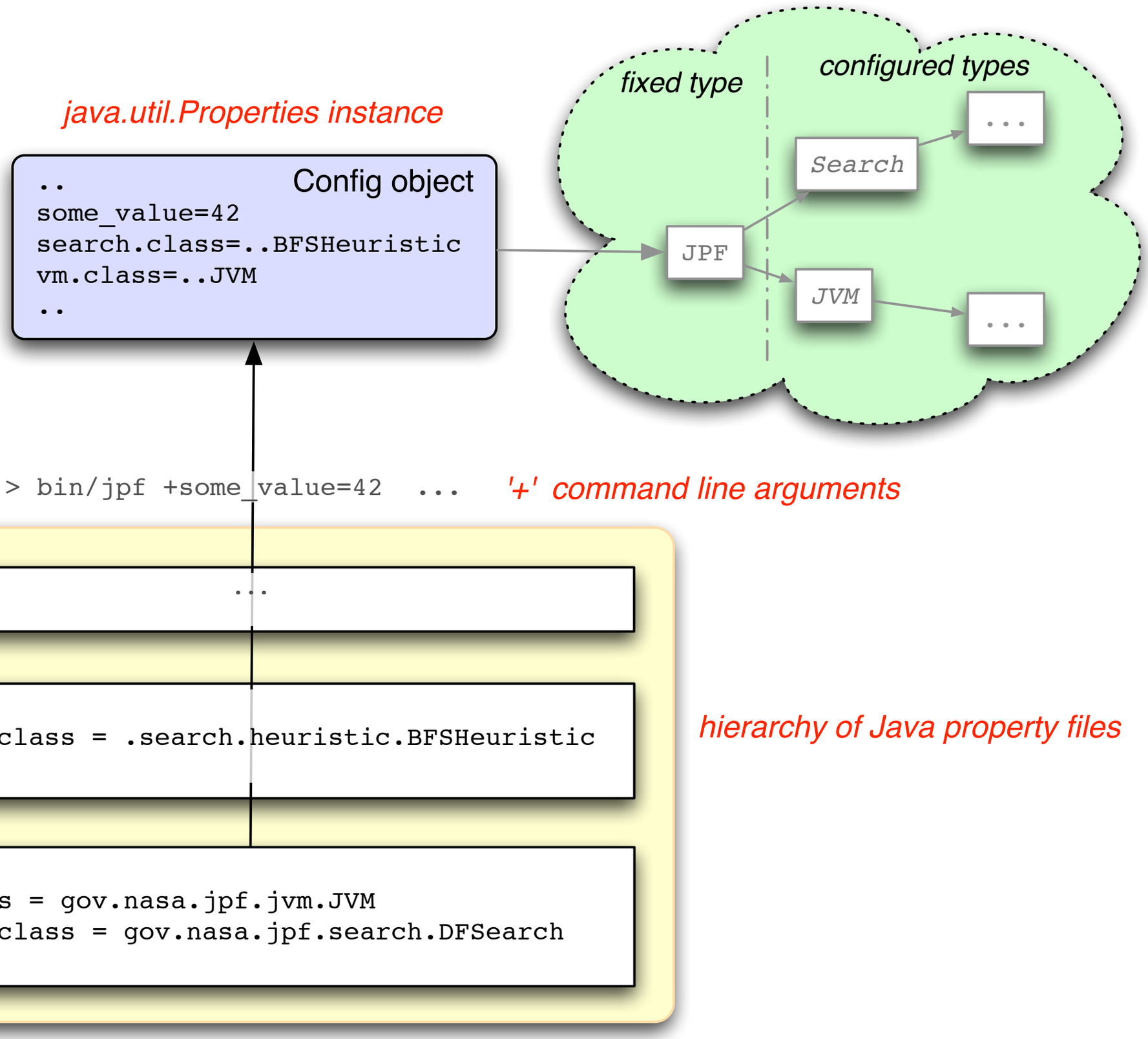
Org: JPF Configuration

- JPF is a dynamically configured open system
- many options for core and extensions
- ⇒ need for an open configuration mechanism

- based on a hierarchical set of normal Java properties files
- four levels:
  - site
  - project
  - application
  - command line

- no central place that describes all options for all extensions
- can be a bit intimidating to master
- help is on the way (GSoC'11 options collector project)

Org: Configuration - General Mechanism



Org: Configuration Levels

command line property arguments

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

```
target = RobotManager
target_args = ...

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

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

application properties

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
...
```

all jpf.properties in order of *extensions*

```
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}
```

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

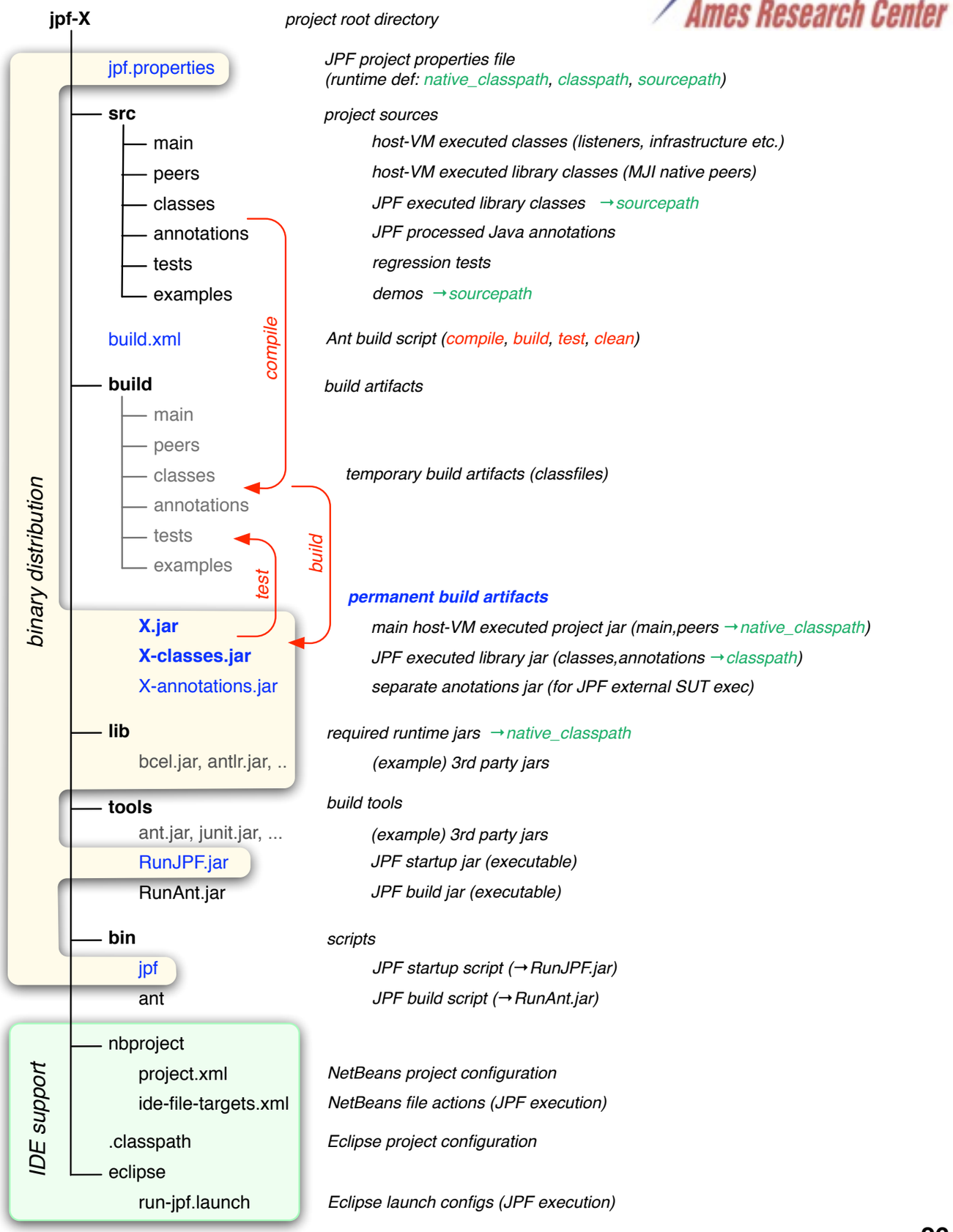
1. site properties

   ~/.jpf/site.properties

   - project locations
   - pre-loaded projects

Org: JPF Directory Structure

- 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)
- all projects have examples and regression test suites (eventually ☹)
- projects have out-of-the-box IDE configuration (NB,Eclipse)



3. Examples

- Listener
- Native Peer
- Instruction Factory

Examples: Listeners

- most frequently used JPF extension mechanism
  - non-invasive
  - orthogonal (can coexist with other extensions)

- executed by host VM ⇒ make sure listener is in classpath

- can be configured in a variety of ways:
  - from properties files (<app>.jpf or jpf.properties)

    ```
    .. listener+=,x.y.MyListener ..
    ```
  - from command line (debugging)

    ```
    bin/jpf +listener=.listener.ExecTracker ..
    ```
  - on demand via class annotation

    ```
    .. listener.autoload+=,<annotation> ..
    ```

- avoid dependencies between listeners (esp. order)

Examples: Listener Code

- extend ListenerAdapter instead of implementing Listener (interface might be extended)

```
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;
        }
      }
...
```

example from http://babelfish.arc.nasa.gov/hg/jpf/jpf-aprop

Examples: Listener Instruction Dispatch

- 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){
      ThreadInfo ti = vm.getLastThreadInfo();
      .. if (areturn.getReturnValue(ti) ..
    }
  }
  ...
  VM vm;
  InsnDispatcher dispatcher;

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

  public void executeInstruction (JVM vm){
    this.vm = vm;
    vm.getLastInstruction().accept(dispatcher);
  }
}
```

Examples: Native Peers

- primary purpose: model native methods
- can do more:
  - performance - long computation in non-relevant code such as String.matches()
  - state optimization
    - native methods are executed atomic
    - can directly interact with VM (lockless waits)
  - function - adding domain specific ChoiceGenerators
- challenge are different object models of JPF and host VM
- MJIEnv is used as a facade to alleviate conversion

Examples: Native Peer

```
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);
  }
```

Example: InstructionFactory

```
package gov.nasa.jpf.numeric.bytecode;

import gov.nasa.jpf.jvm.bytecode.Instruction;

public class InstructionFactory extends gov.nasa.jpf.jvm.bytecode.InstructionFactory {

  @Override public Instruction iadd() {
    return new IADD();
  }
  ...
}

public class IADD extends gov.nasa.jpf.jvm.bytecode.IADD {

  @Override public Instruction execute (SystemState ss, KernelState ks, ThreadInfo ti) {
    int v1 = ti.pop();
    int v2 = ti.pop();

    int res = v1 + v2;

    if ((v1>0 && v2>0 && res<=0) || (v1<0 && v2<0 && res>=0)){
      return th.createAndThrowException("java.lang.ArithmeticException",
                                        "integer overflow: " + v2 + "+" + v1 + "=" + res);
    }

    th.push(res, false);

    return getNext(th);
  }
}
```

Example: Attributes

```
public class NumericAttr {
  Apfloat 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!

Backup Slides

Ames Research Center

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

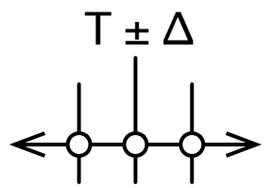
xChoiceGenerator
choiceSet: {x}
hasMoreChoices() advance() getNextChoice() → x

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



C = { T-Δ, T, T+ Δ }

application code (test driver)

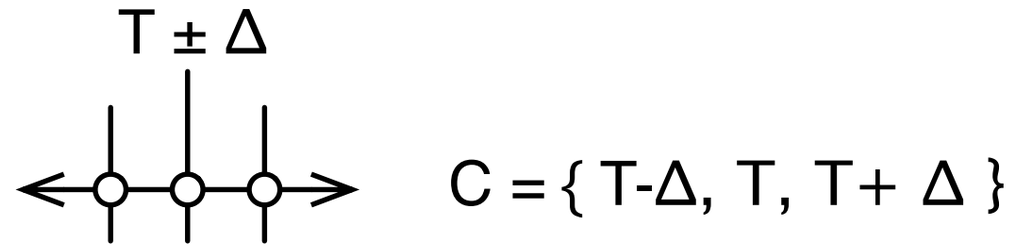
```
..
double v = Verify.getDouble("velocity");
..
```

configuration (e.g. mode property file)

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

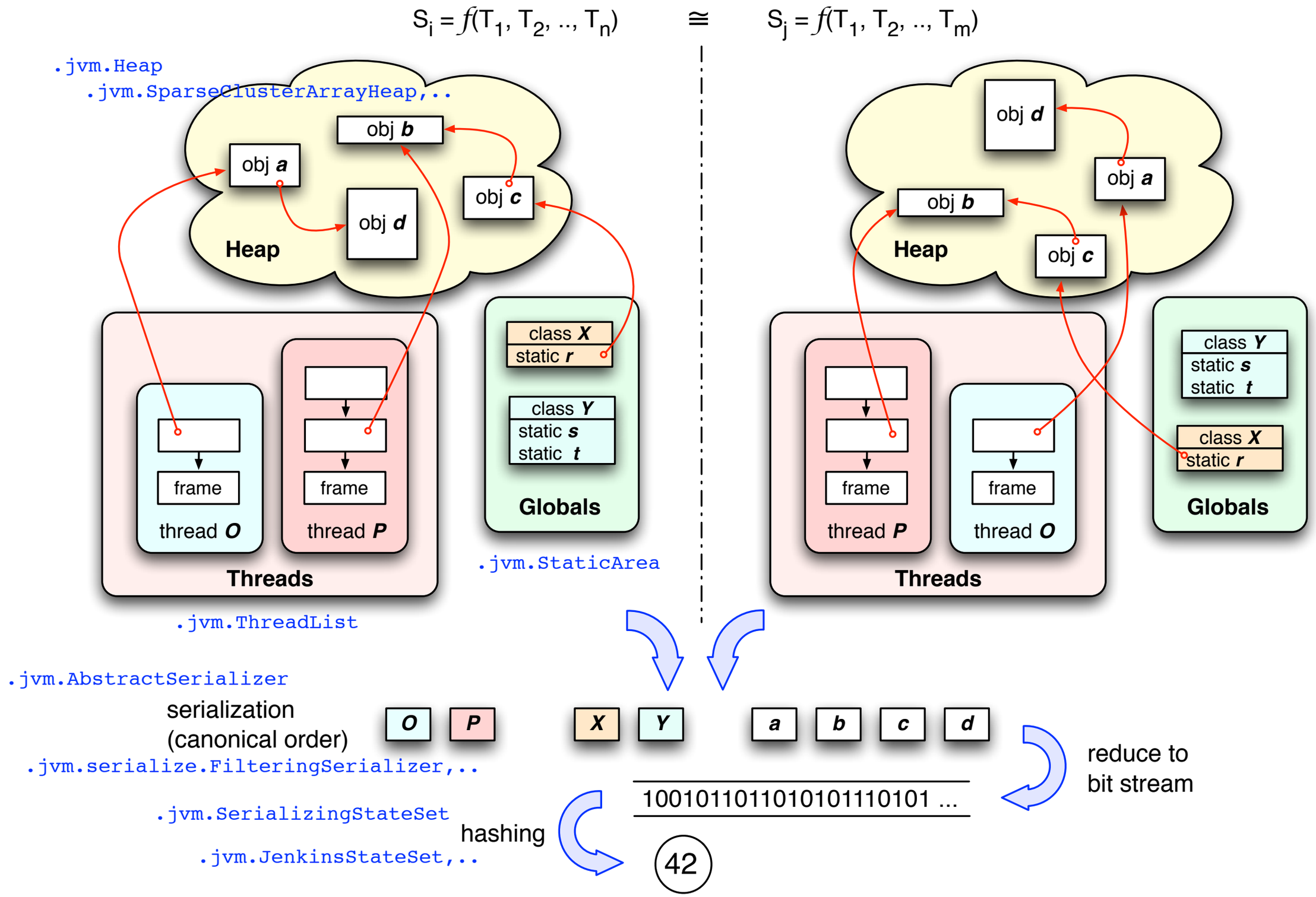
Example: ChoiceGenerator

```
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; }
```

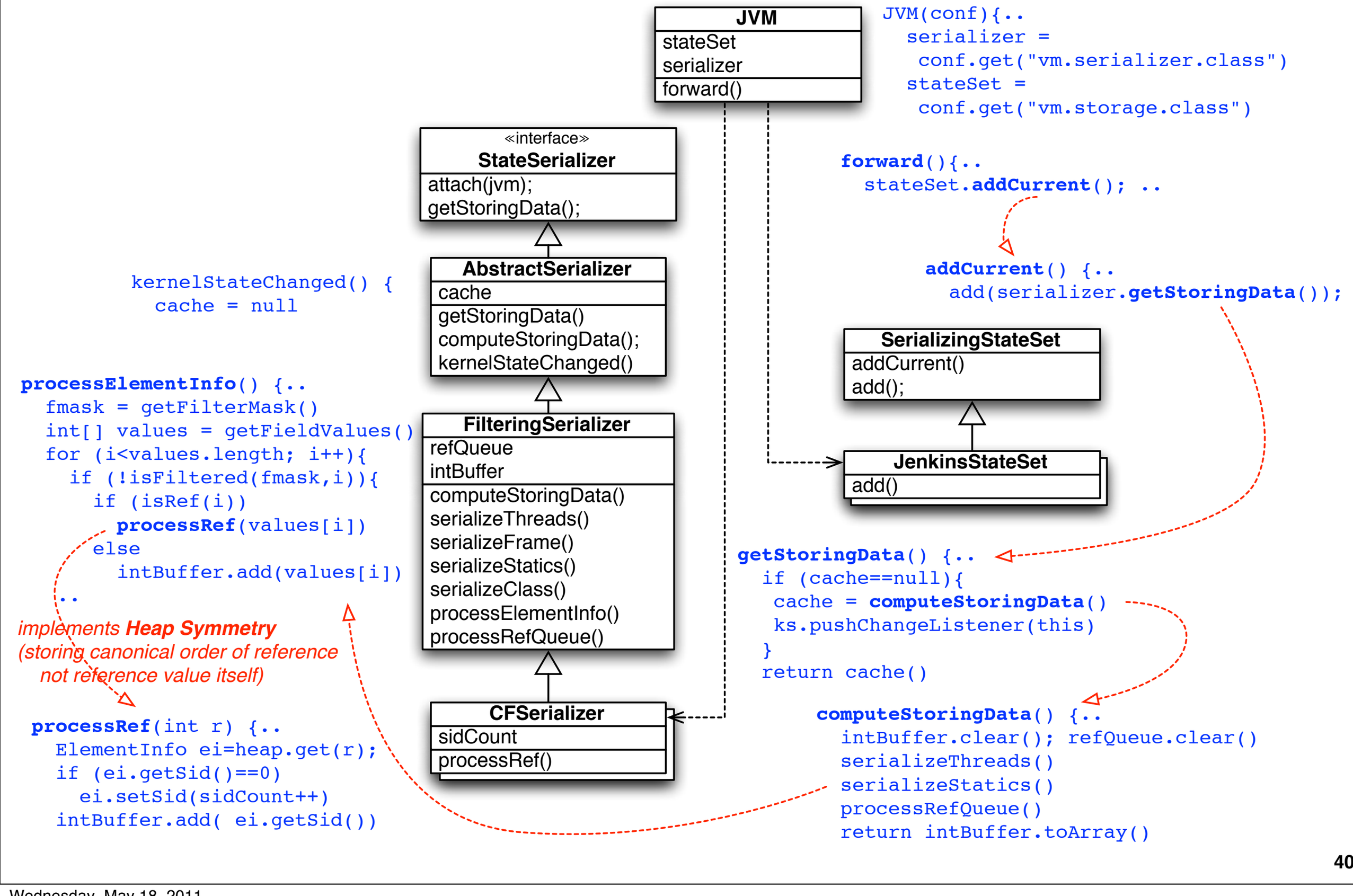


Basics: Serialization

- don't produce different states for property irrelevant permutations

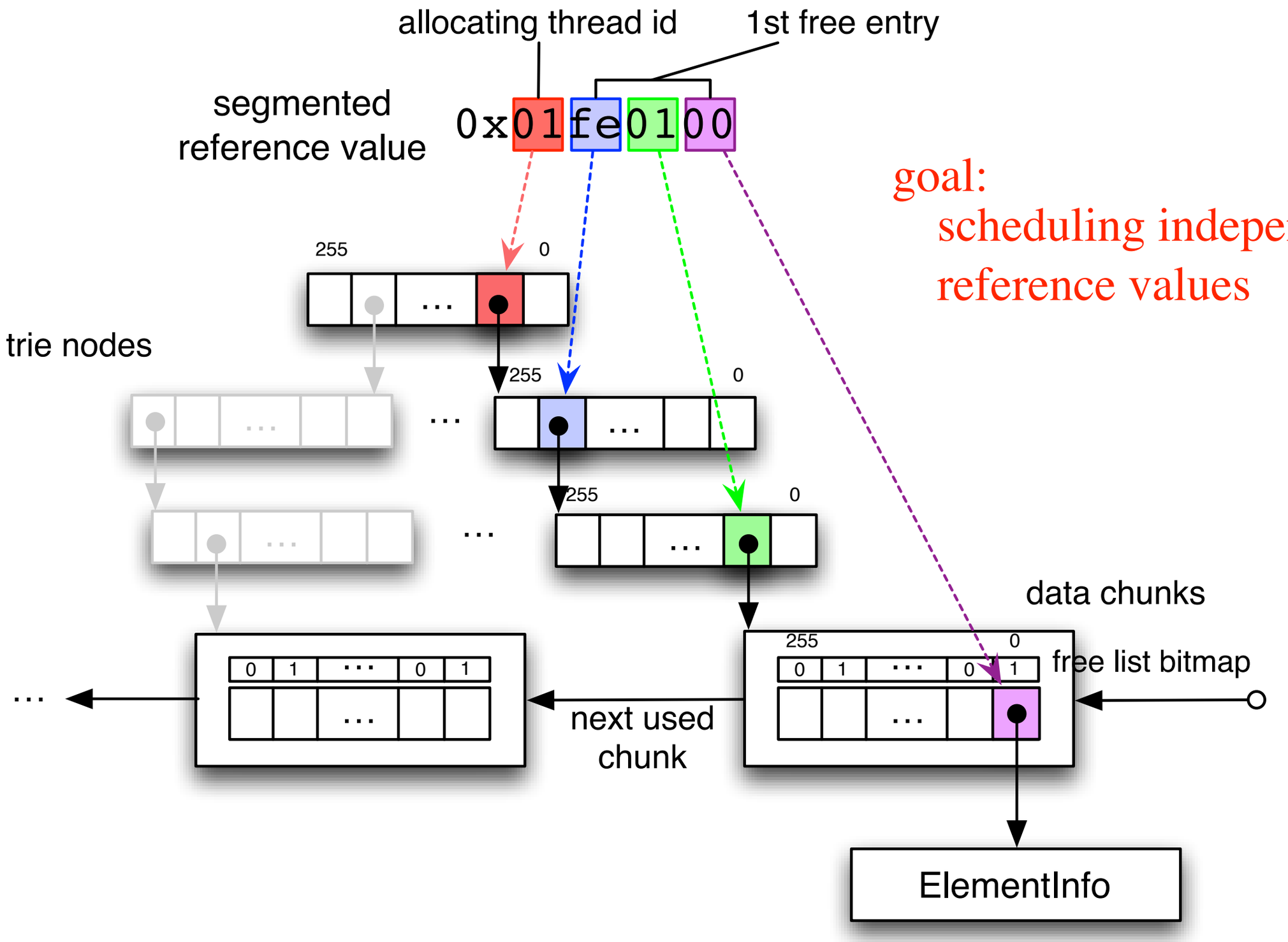


Basics: Serialization Implementation



Basics: Serialization - (Heap Symmetry)

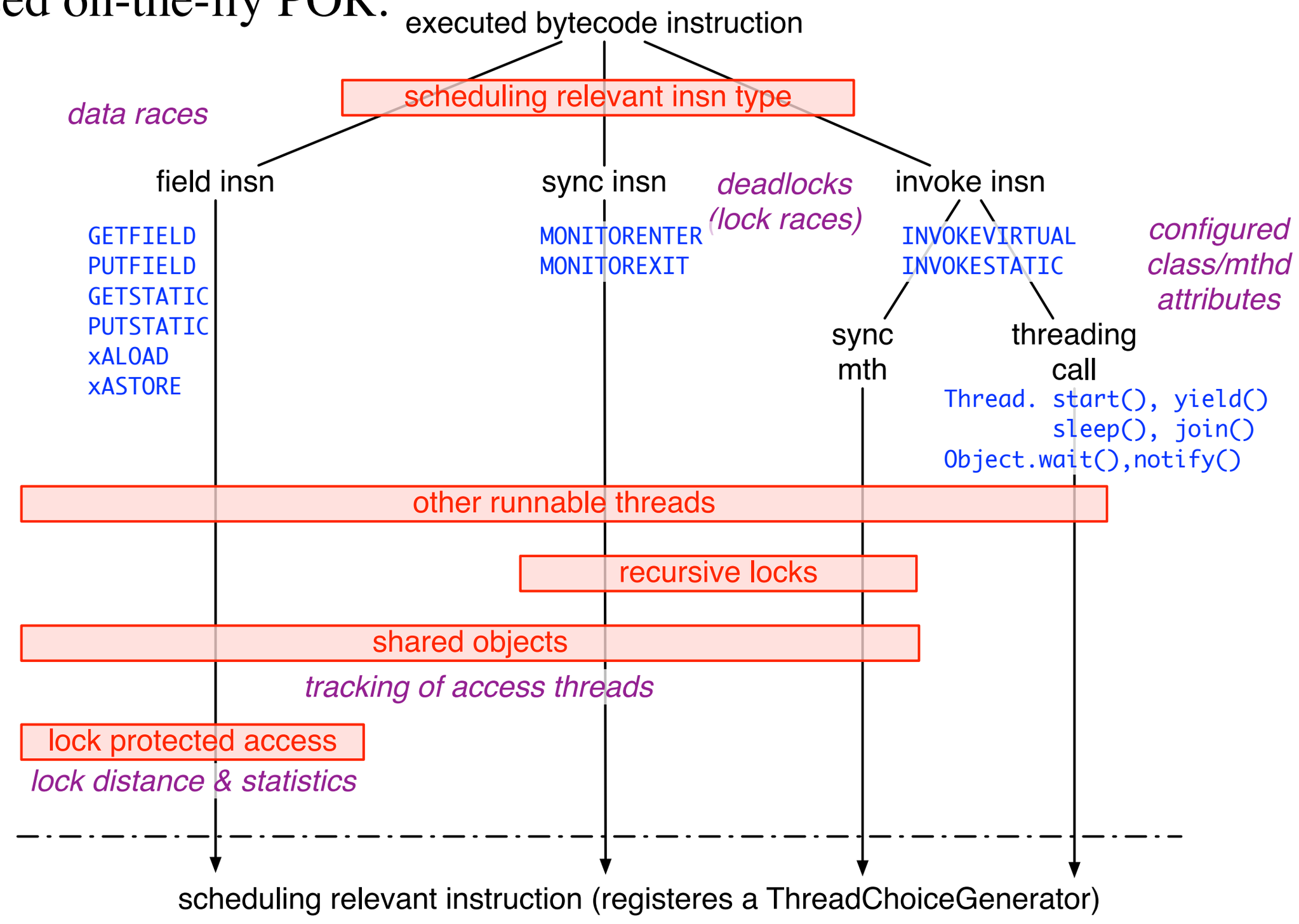
- not really, main mechanism is in CFSerializer



goal:
scheduling independent
reference values

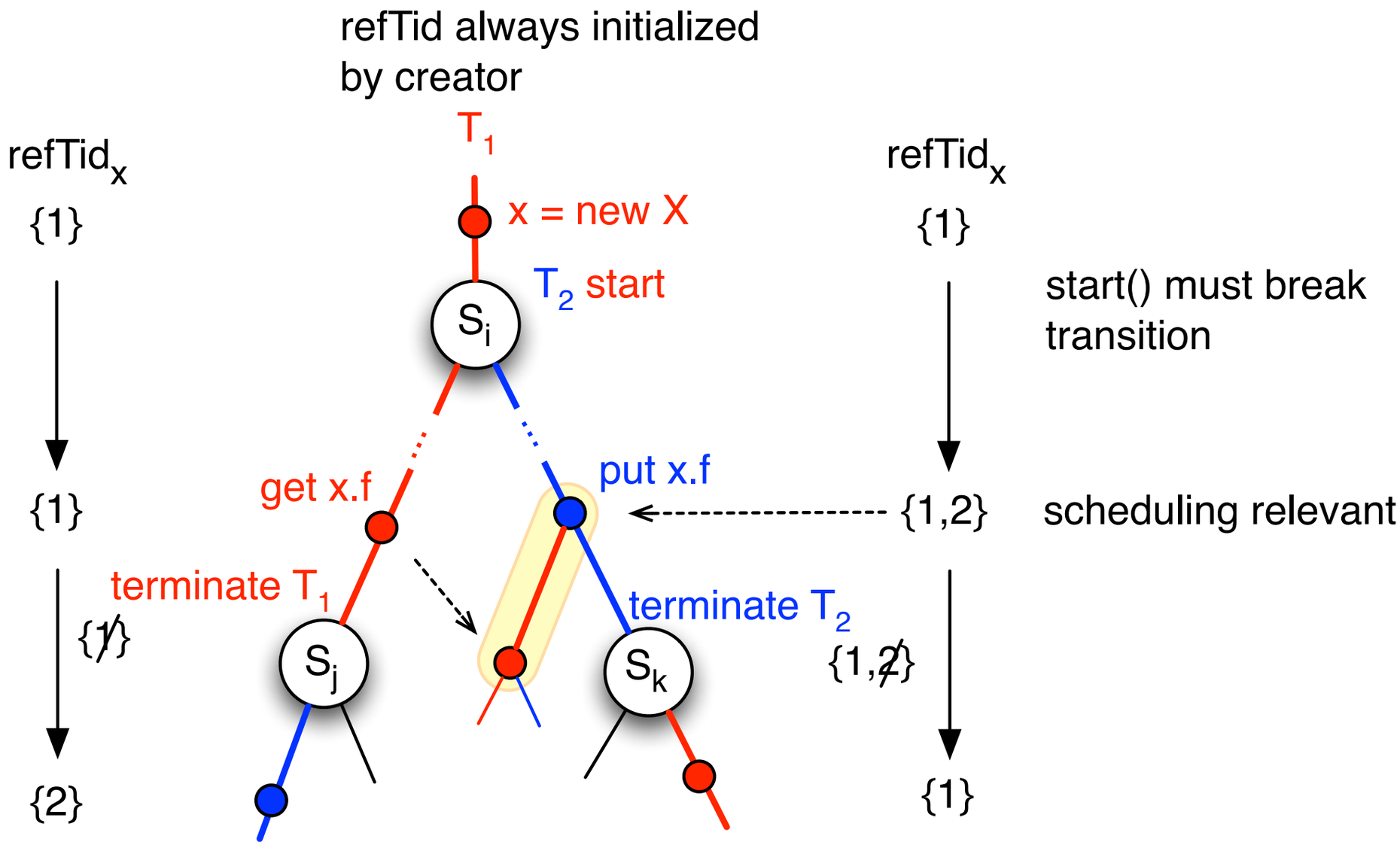
Basics: Partial Order Reduction

- only subset of Java instructions can have inter-thread effects
- type based on-the-fly POR:



Basics: Partial Order Reduction

- new precise thread access tracking requires Thread.start() to break
- interesting things happen on the right branches!



Basics: POR - Thread Access Tracking

- JPF objects keep track of which live threads access them

```
return
  refTid.cardinality() > 1;
```

ElementInfo
BitSet refTid
isShared() checkUpdatedSharedness(thread)

```
if (!refTidChanged)
  refTid = refTid.clone();

int tid = thread.getIndex();
if (!refTid.get(tid)){
  refTidChanged = true;
  refTid.set(thread.getIndex());
}
removeTerminatedThreads(refTid);
return refTid.cardinality() > 1;
```

InstanceFieldInstruction
isSchedulingRelevant(thread,objRef)

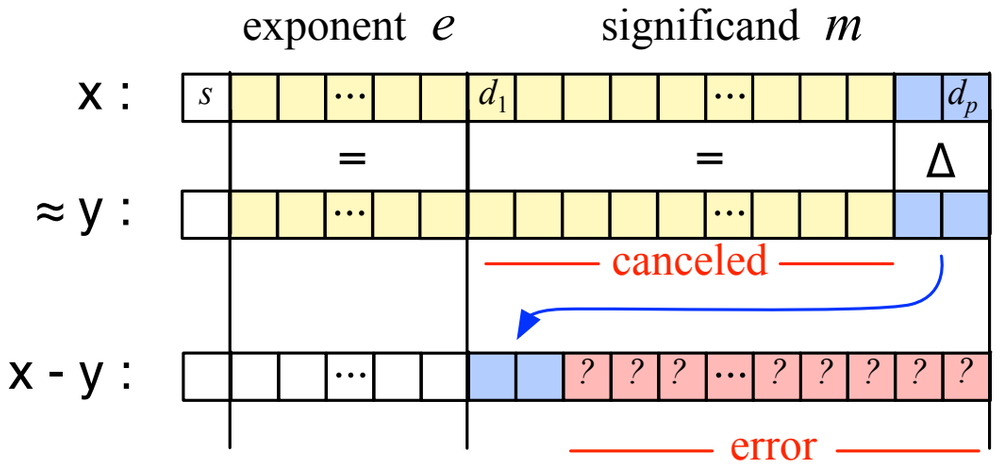
PUT/GETFIELD
execute(thread,..)

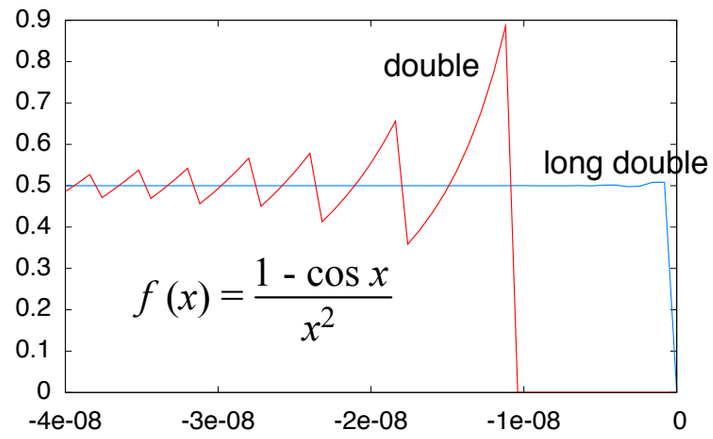
```
..
ElementInfo ei = thread.getElementInfo(objRef);
if (ei.checkUpdatedSharedness(thread))
  return true;
..
```

```
..
objRef = thread.pop();
if (!thread.isFirstStepInsn() &&
      isSchedulingRelevant(thread,objRef)){
  createAndSetFieldCG(..);
  return this; // break transition
} ..
```

Example: 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);
```

```
vm.insn_factory.class =
    .numeric.NumericInstructionFactory
```

```
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