JPF'08
Tales from all Corners of the Realm

Peter C. Mehlitz

PSGS / NASA Ames Research Center

<Peter.C.Mehlitz@nasa.gov>

Overview

✦ Modeling Framework Example: Statecharts

*.java

- Motivation
- Implementation Components

✦ Assorted Core Additions

*.class @V *.java *.java

- BytecodeFactory
- Value Attributes
- Reporter/Publisher

✦ Annotations and JPF

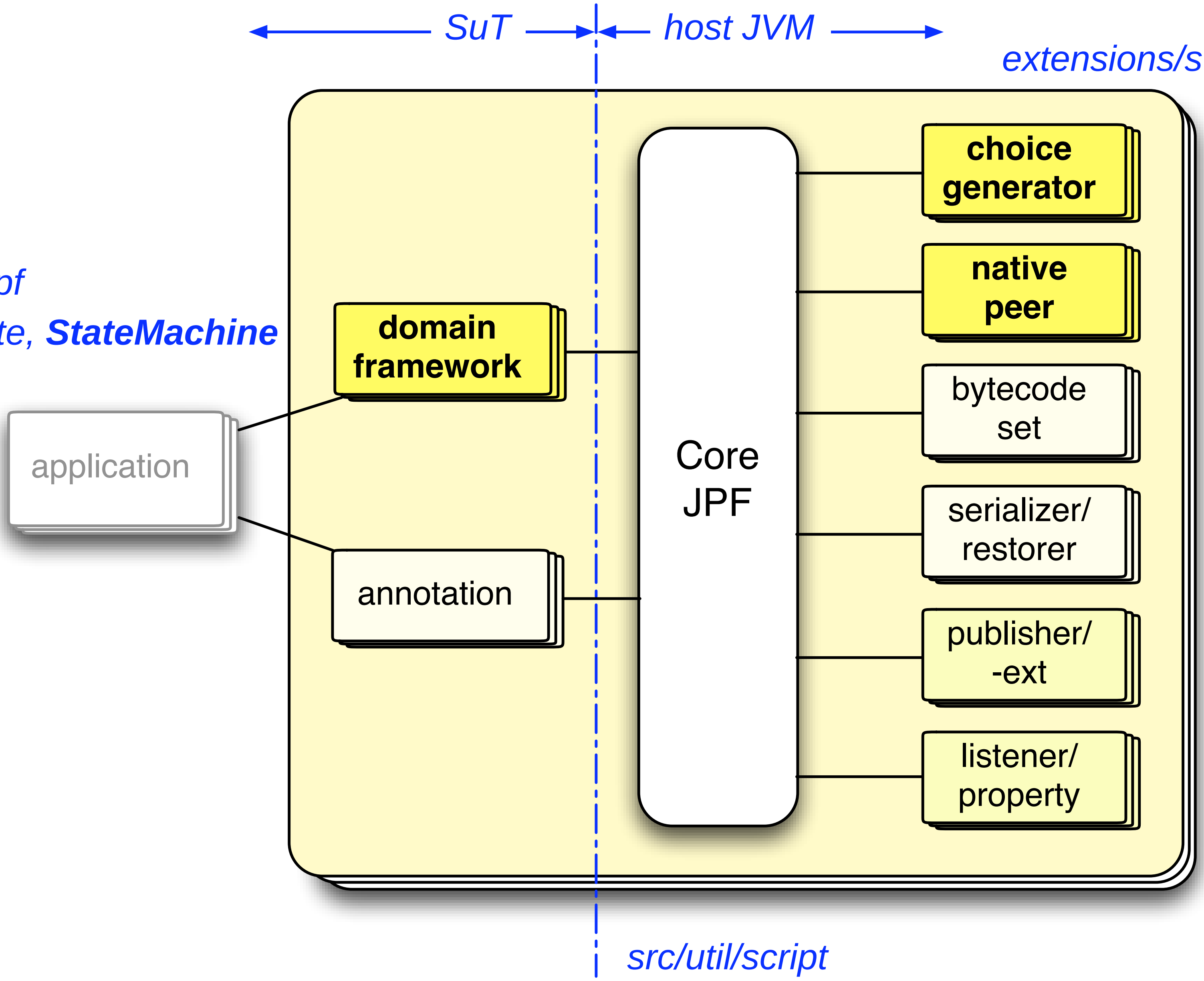
@V *.java

- Requirements Coverage
- Sequences
- Programming-by-Contract
- Tests

Model Frameworks: Statecharts

- example of domain specific modeling with Java

```
> jpf gov.nasa.jpf.sc.StateMachine MyStateMachine [guidance-script]
```



extensions/statechart

.../src
SCEvent, SCEventGenerator, ..

.../env/jvm
JPF_.._State, **NativeStateMachine**

.../env/jpf
State, **StateMachine**

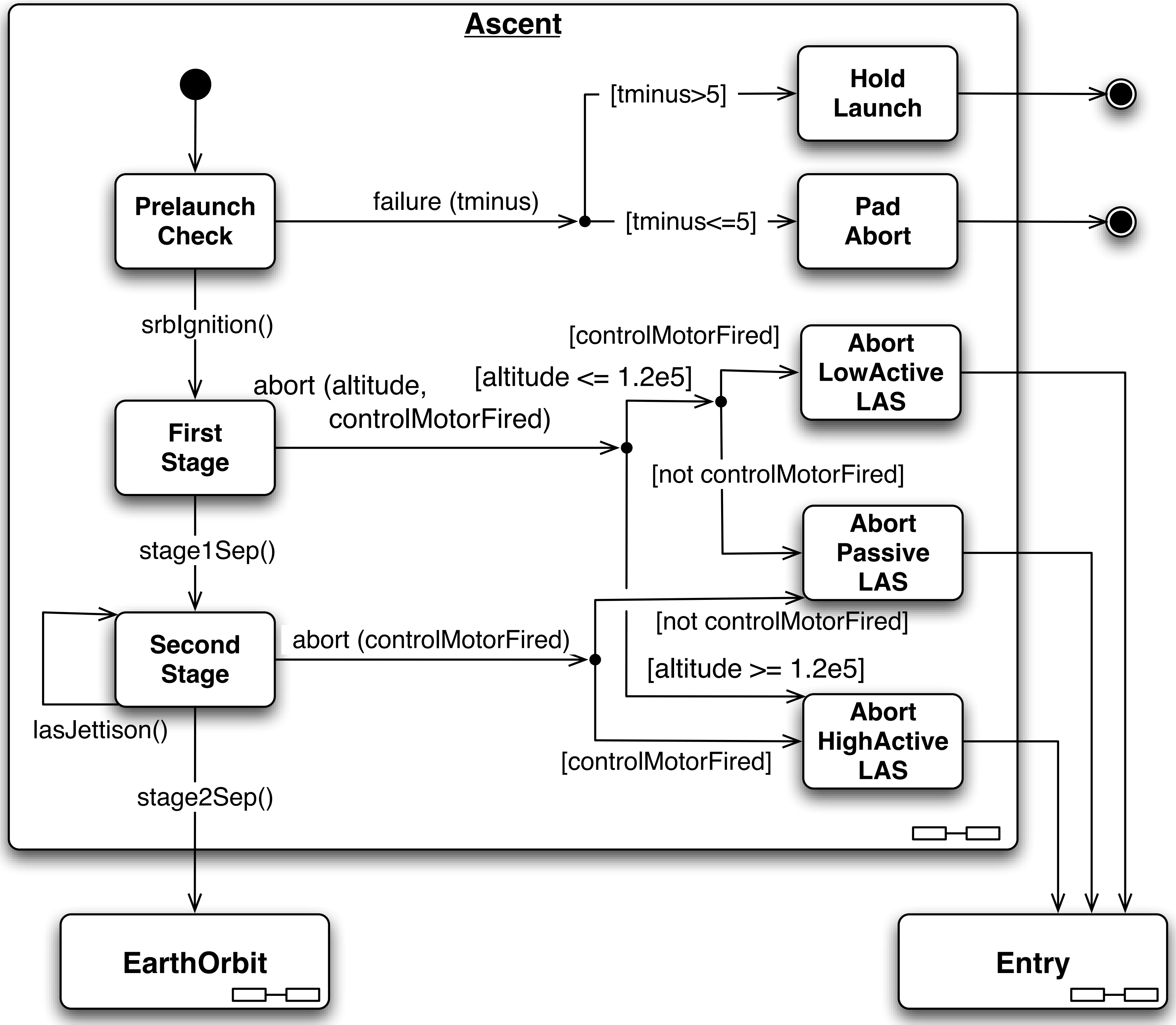
.../src/tools
Coverage, SimStateMachine

src/util/script

ScriptEnvironment, ESParser, Event, ..

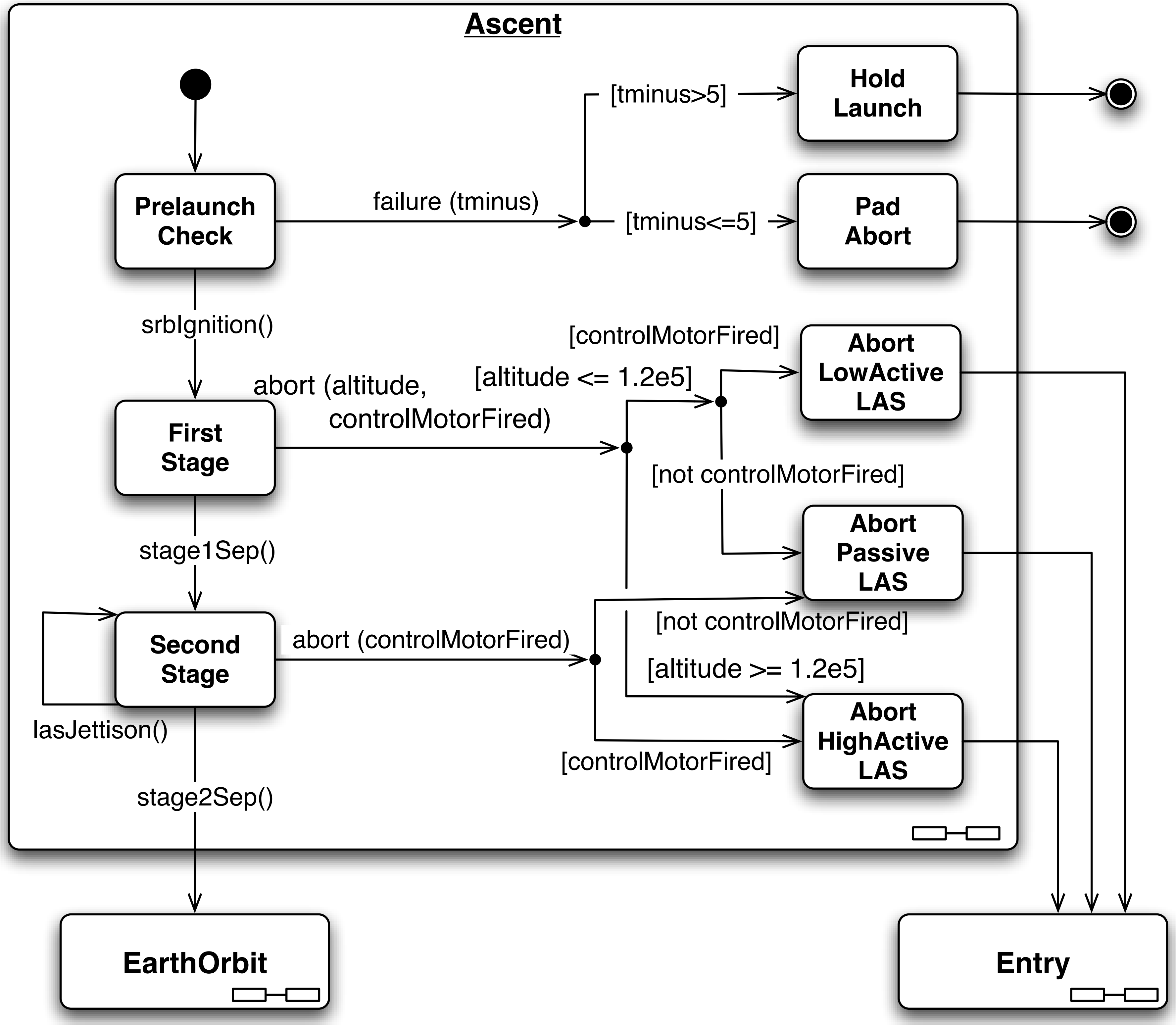
Statecharts: Why (1)

- why? domain specific properties



Statecharts: Why (1)

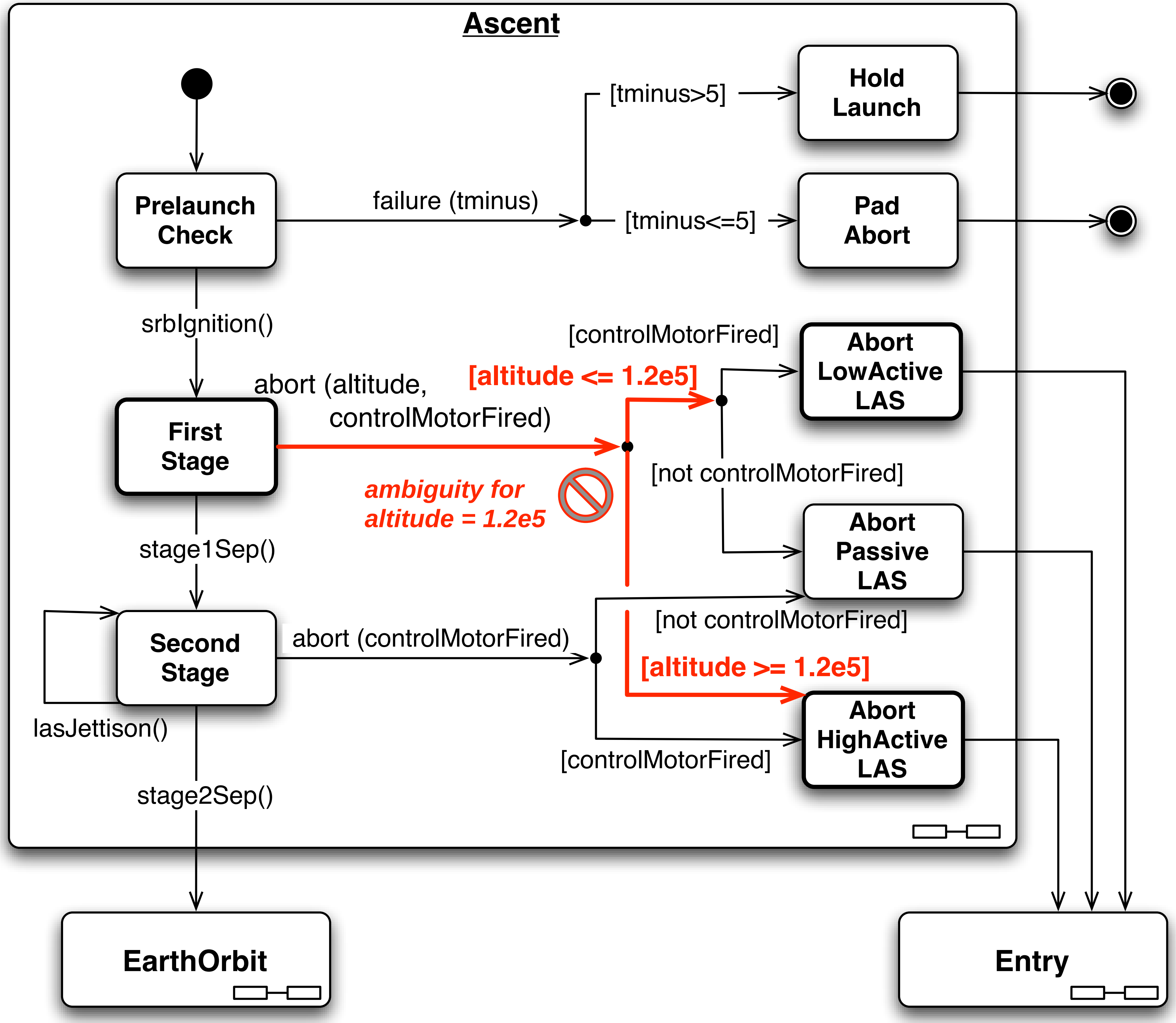
- why? domain specific properties



text consistent with diagram?

Statecharts: Why (2)

- make model executable (strict execution semantics)



Statecharts: Execution Semantics

- loop as long as *active* state set is not empty
- get enabling events
- loop over *active* state set
- try event on active state by executing trigger method
- if trigger fires, add target state to *next* set, otherwise add the currently processed state again
- *next* set becomes new *active* set
- continue with next step

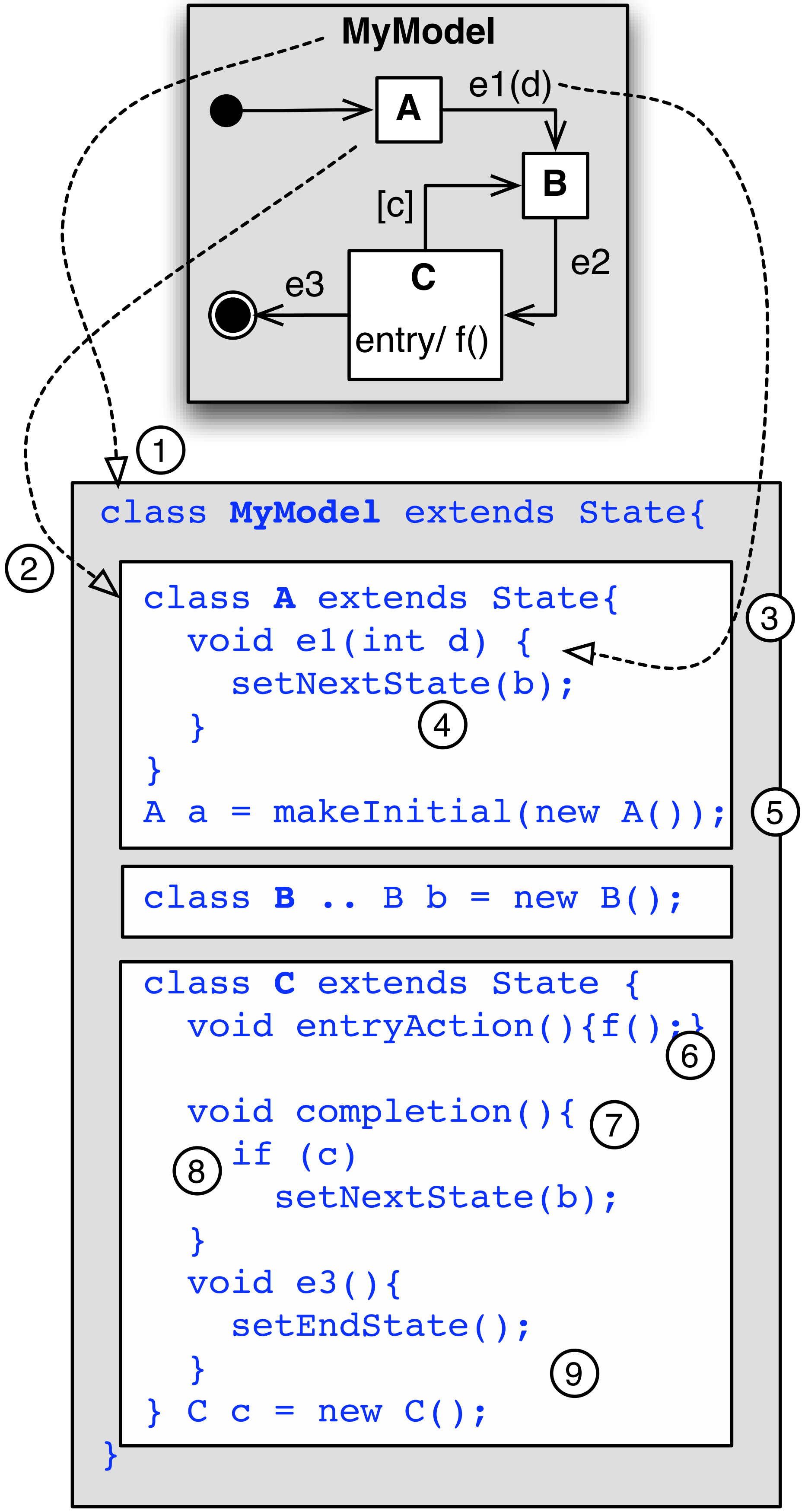
```
instantiate state machine
compute $S_{active}$ (start states)

while $S_{active} \neq \varnothing$
    get $E_{enable}$ ($S_{active}$)
    if ($E_{enable} \neq \varnothing$)
                                        model checking loop
        foreach $event \in E_{enable}$
            $S_{next} = \varnothing$
            foreach $state \in S_{active}$
                reset $nextState$
                get $triggerMethod$ ($event$)
                if $triggerMethod$ found
                    call $triggerMethod$

                if $nextState$ set
                    add $nextState$ to $S_{next}$
                else
                    add $state$ to $S_{next}$
        $S_{active} = S_{next}$
```

Statecharts: Model Code Structure (1)

- (1) set of diagrams → one toplevel class
- (2) each substate → nested class
- (3) each trigger → Java method
- (4) transitions → `setNextState(state)`
- (5) initial states / orthogonal regions
  → `makeInitial(..)`
- (6) entry/ exit/ actions → corresponding
  `entry/exitAction()` methods
- (7) completion triggers
  → `completion()` method
- (8) guards → boolean java expressions
- (9) end states → `setEndState(..)` calls

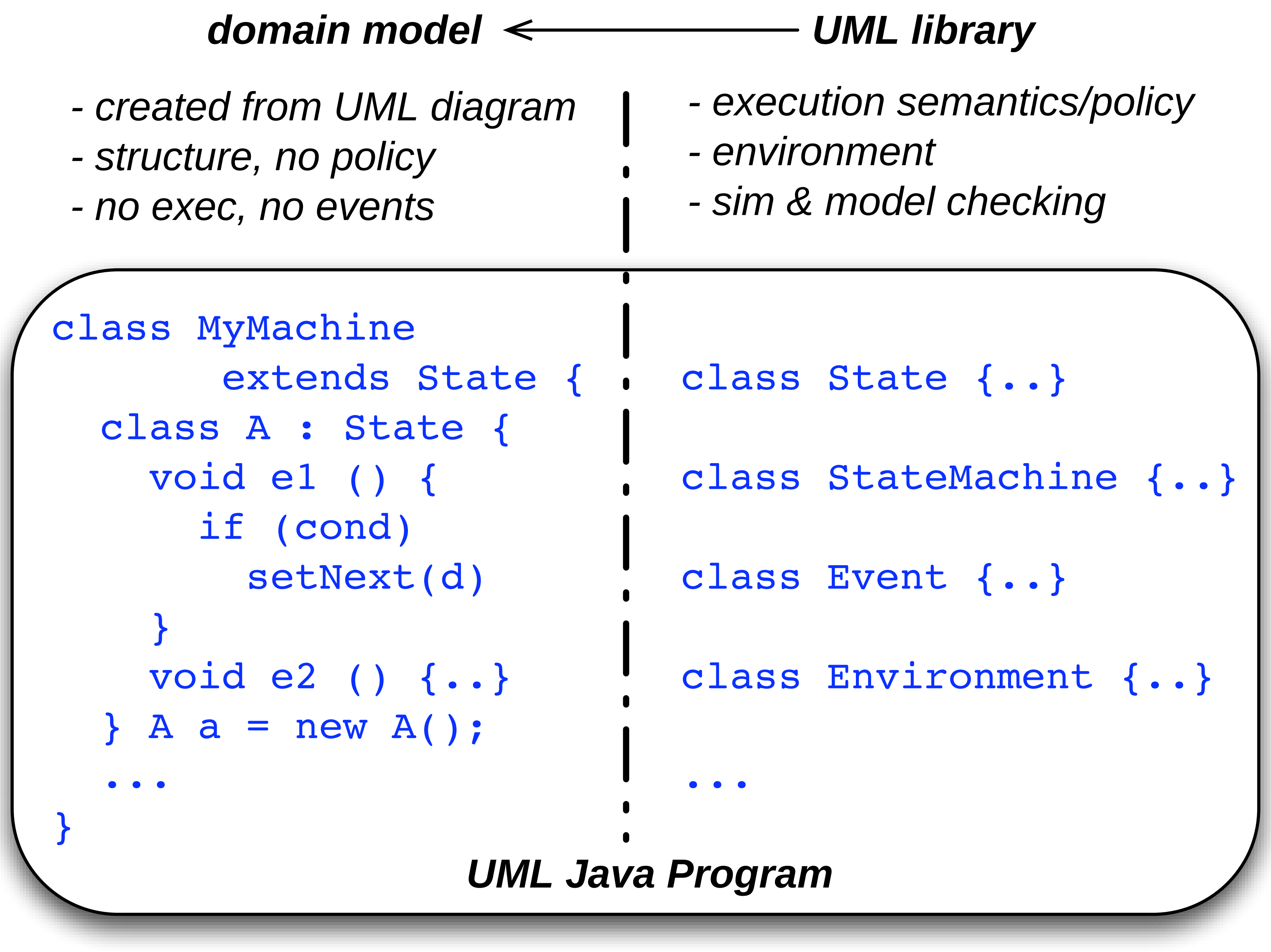


```
class MyModel extends State{
  class A extends State{
    void e1(int d) {
      setNextState(b);
    }
  }
  A a = makeInitial(new A());
  class B .. B b = new B();
  class C extends State {
    void entryAction(){f();}

    void completion(){
      if (c)
        setNextState(b);
    }
    void e3(){
      setEndState();
    }
  } C c = new C();
}
```

Statecharts: Model Code Structure (2)

- Layers: make modeling easy, push complexity into (hidden) library
- domain library is the real development effort



domain model ← ***UML library***

- created from UML diagram
- structure, no policy
- no exec, no events

- execution semantics/policy
- environment
- sim & model checking

```
class MyMachine
       extends State {
  class A : State {
    void e1 () {
      if (cond)
        setNext(d)
    }
    void e2 () {..}
  } A a = new A();
  ...
}
```

```
class State {..}

class StateMachine {..}

class Event {..}

class Environment {..}

...
```

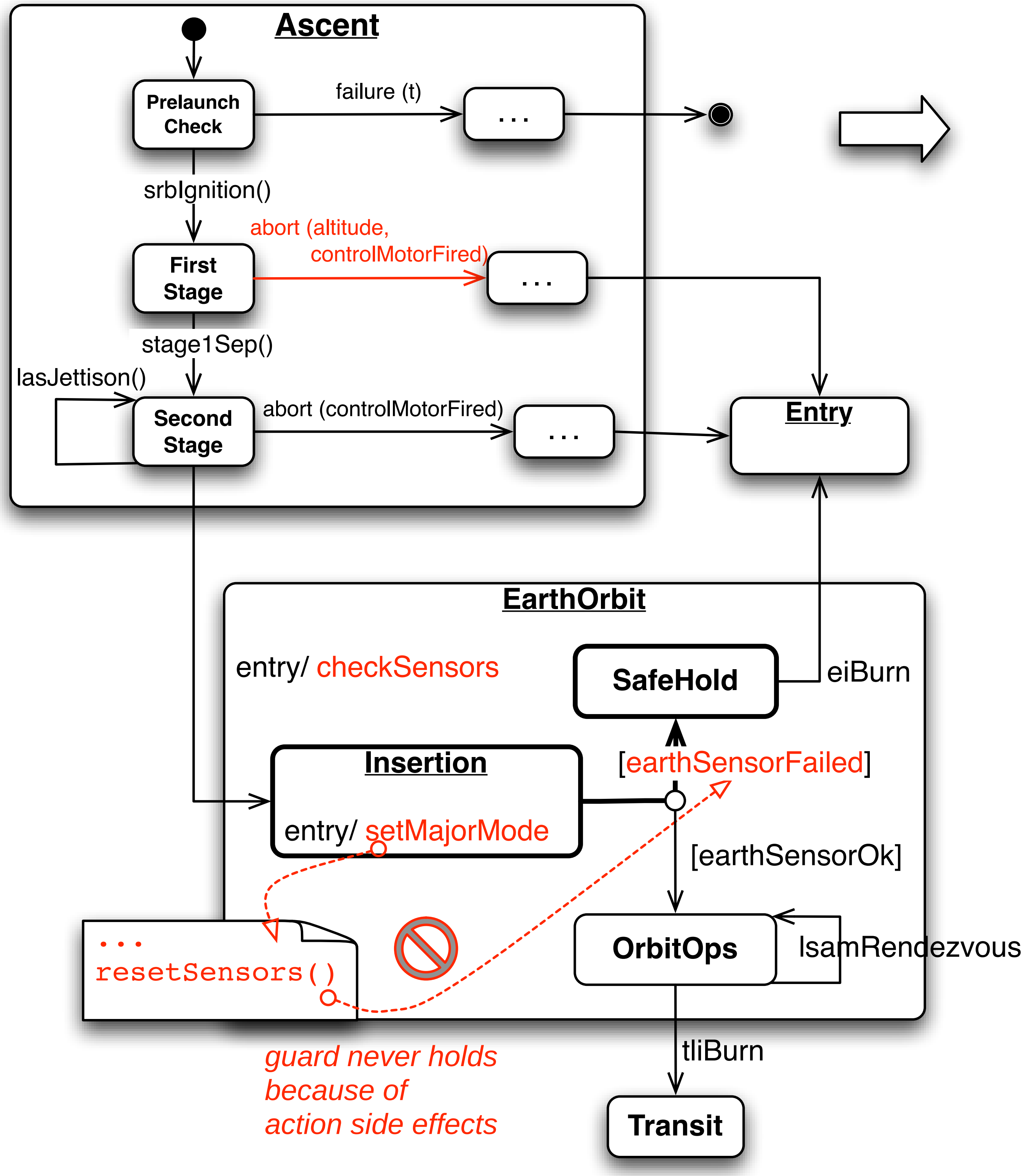
UML Java Program

Statecharts: Guidance Scripts (1)

Ames Research Center

Model State Space

```
>jpf gov.nasa.jpf.sc.StateMachine CEV_15EOR_LOR SafeHold.es
```



guard never holds because of action side effects

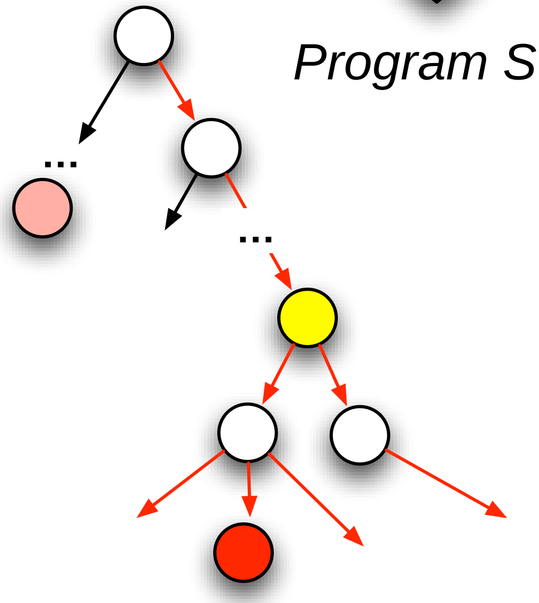
Guidance Script

```
// just get off the ground and into orbit
srbIgnition
stage1Separation
lasJettison
stage2Separation


//--- check all of EarthOrbit
SECTION earthOrbit {
  // covers Insertion and SafeHold
  ANY {*}
}

SECTION earthOrbit.orbitOps {
  lsamRendezvous
  tliBurn
}
```

Program State Space



Statecharts: Guidance Scripts (2)

simple event sequence (no search)

- good for testing nominal sequences

```
srbIgnition
stage1Separation
...
```

choices

- explicit list of event/parameter combinations
- lexical patterns
- ‘*’ choices: all handled events

```
ANY { abort(100), abort(120) }
```

or

```
ANY { abort(1[024]0) }
```

or

```
ANY { * }
```

iterations

- bounded and unbounded (can cause infinite loops if statespace is not closed)

```
REPEAT 5 { fireThruster }
```

sections

- event sequences to be processed when a UML state becomes active
- hierarchical (lookup upwards from concrete active state and all it’s parent states until matching section is found)

```
SECTION ascent {
  srbIgnition
  stage1Separation
  lasJettison
  stage2Separation
}
SECTION earthOrbit {
  ANY {*}
}
SECTION earthOrbit.orbitOps {
  lsamRendezvous
  tliBurn
}
```

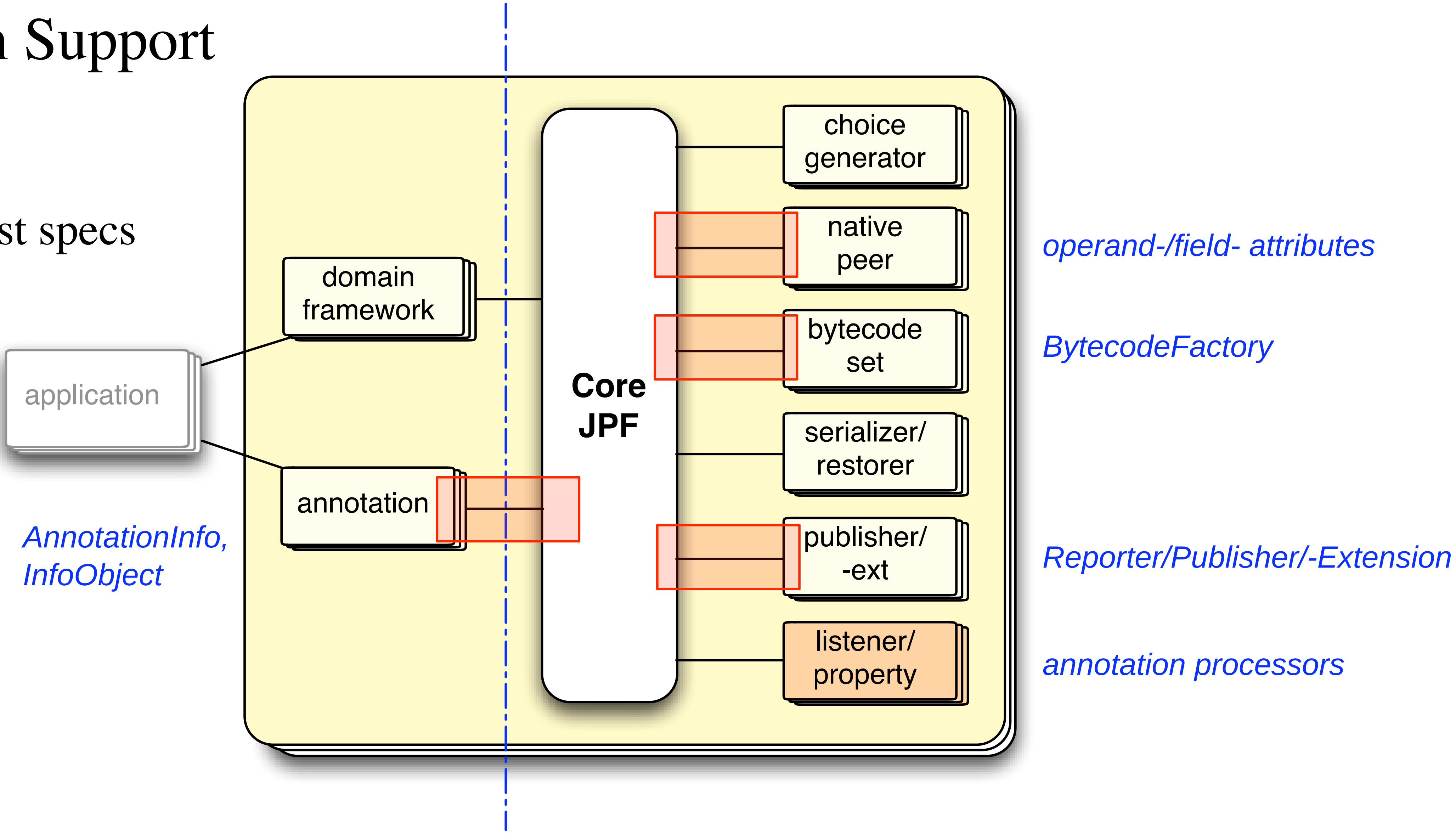
Core Additions

✦ 3 new Extension Mechanisms

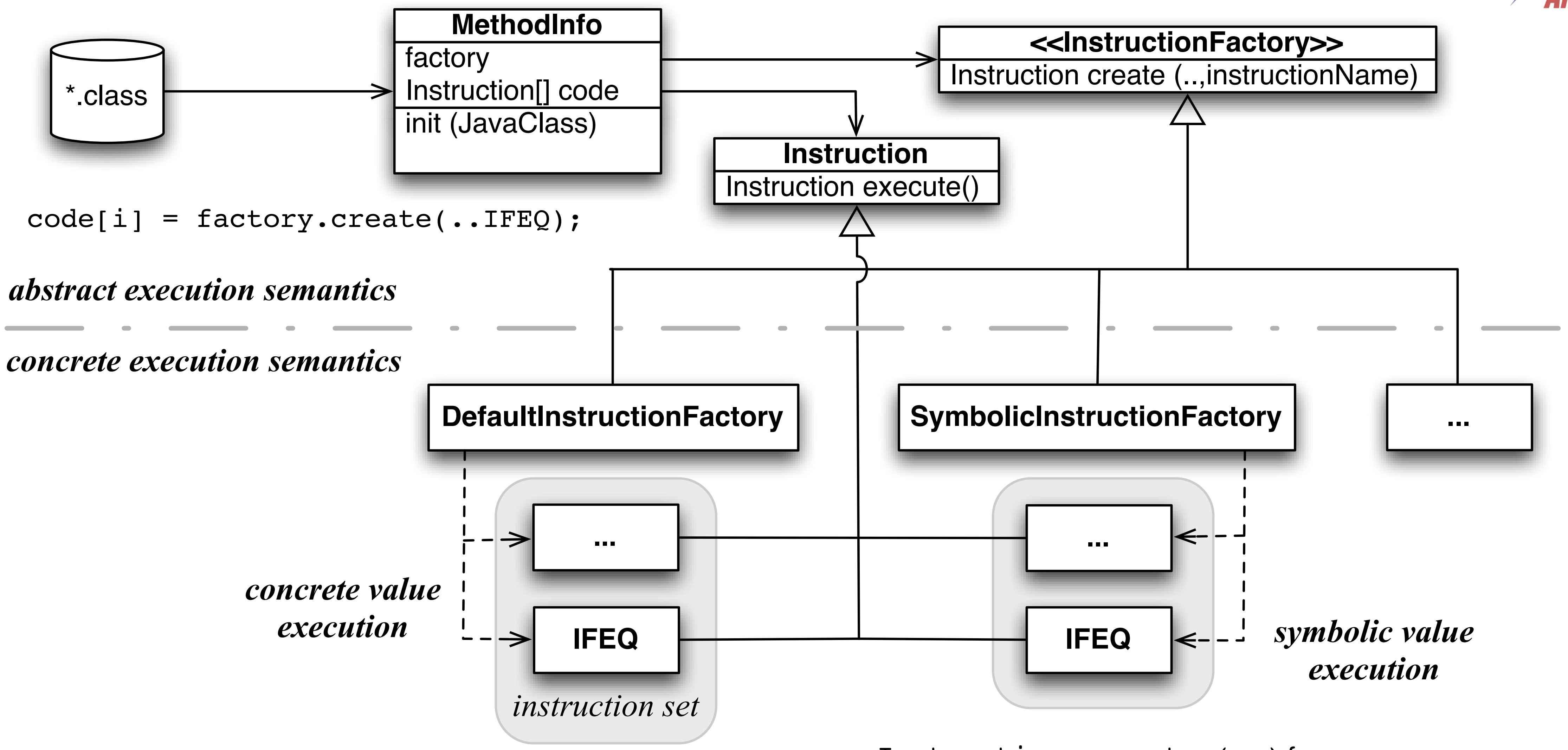
- BytecodeFactory
- operand/field attributes
- Reporter/Publisher

✦ Annotation Support

- events
- PbC
- in-source test specs
- .. and more



Core: Bytecode Sets/Factories

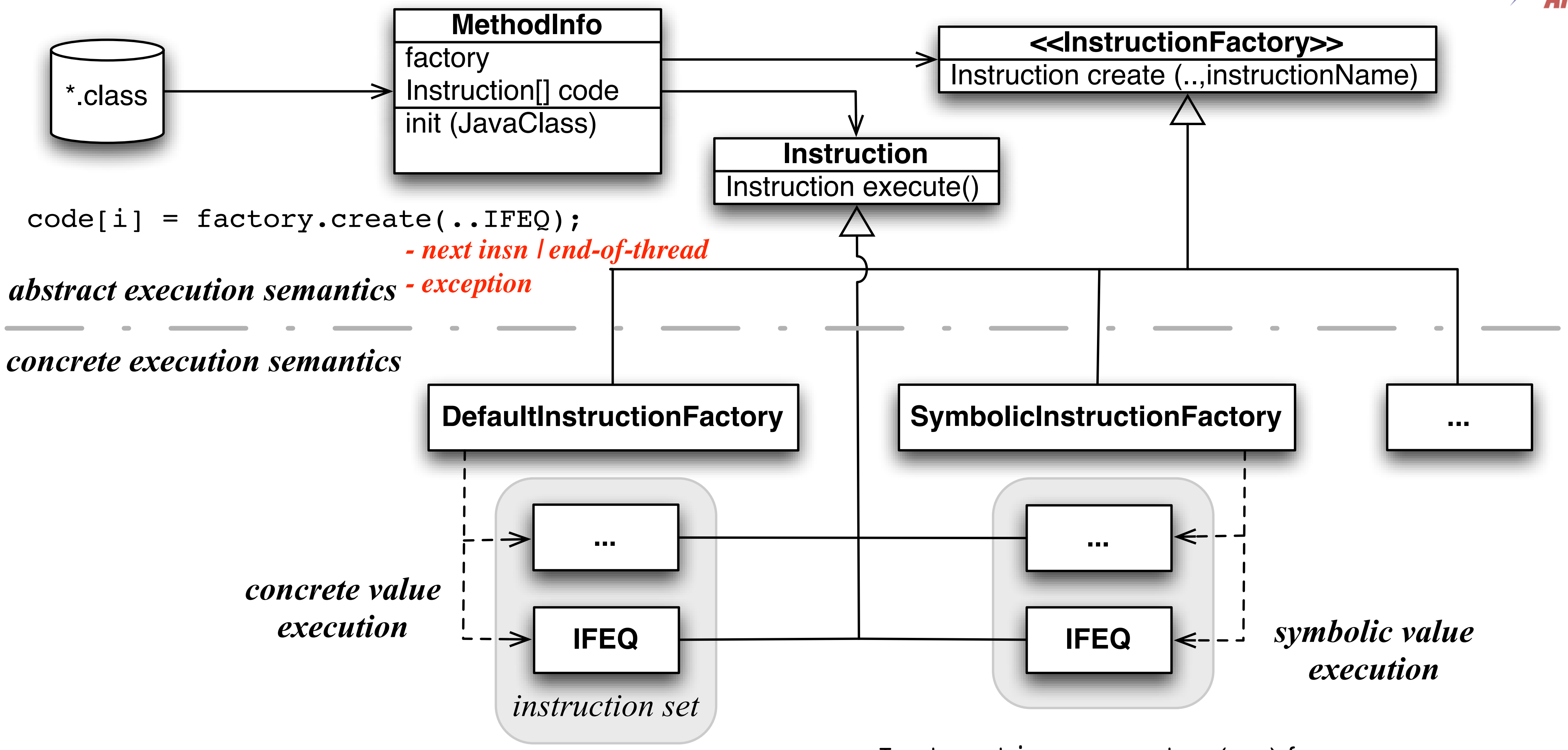


```
code[i] = factory.create(..IFEQ);
```

```
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){...
    updatePathCondition(.., EQ);
    return jumpTarget;
  } else {...
    updatePathCondition(.., NE);
    return getNextInsn();
  }
}
```

Core: Bytecode Sets/Factories



*.class

MethodInfo
factory
Instruction[] code
init (JavaClass)

<<InstructionFactory>>
Instruction create (..,instructionName)

Instruction
Instruction execute()

```
code[i] = factory.create(..IFEQ);
```

- *next insn / end-of-thread*
- *exception*

abstract execution semantics

concrete execution semantics

DefaultInstructionFactory

SymbolicInstructionFactory

...

...

IFEQ

instruction set

...

IFEQ

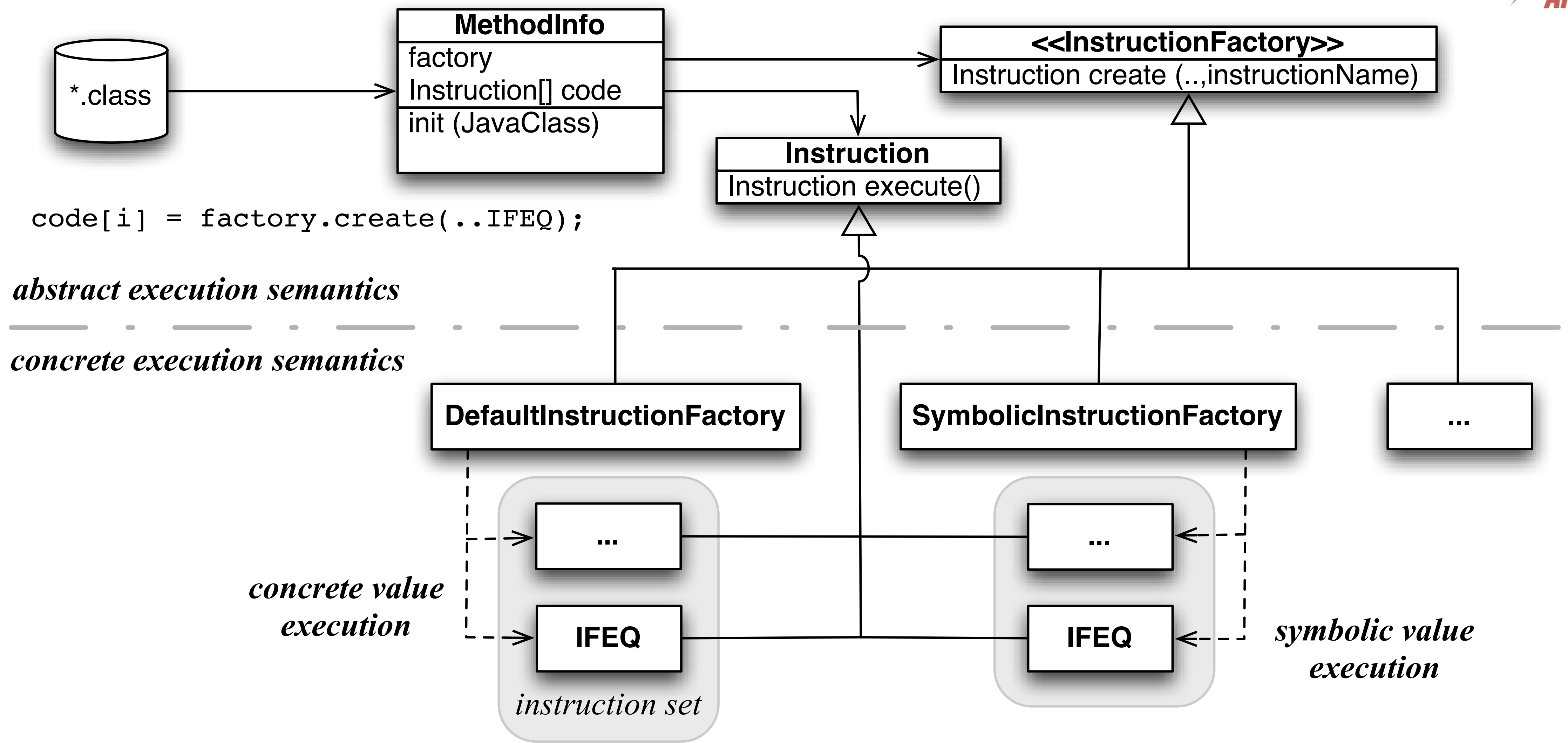
concrete value execution

symbolic value execution

```
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){...
    updatePathCondition(.., EQ);
    return jumpTarget;
  } else {...
    updatePathCondition(.., NE);
    return getNextInsn();
  }
}
```

Core: Bytecode Sets/Factories

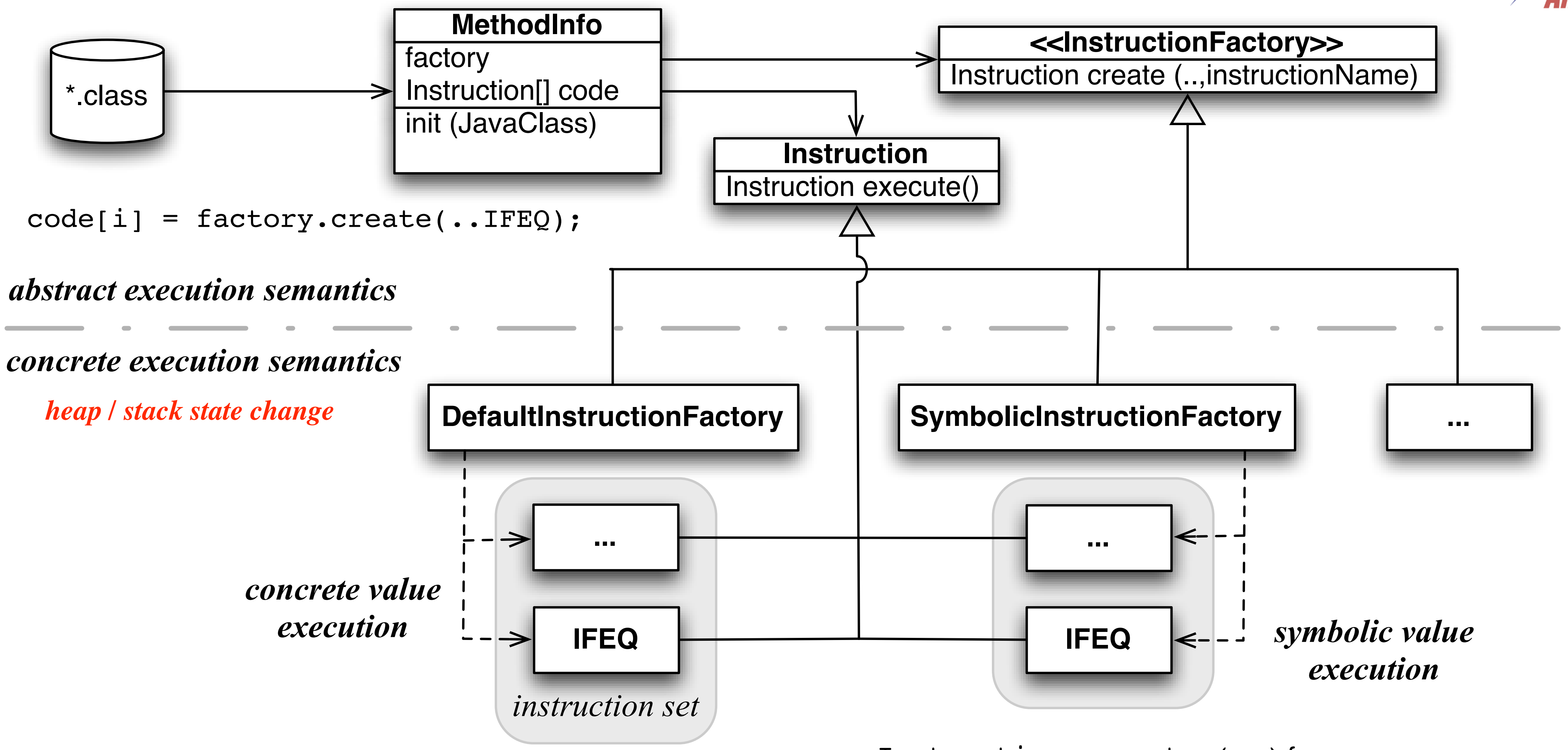


```
code[i] = factory.create(..IFEQ);
```

```
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){...
    updatePathCondition(.., EQ);
    return jumpTarget;
  } else {...
    updatePathCondition(.., NE);
    return getNextInsn();
  }
}
```

Core: Bytecode Sets/Factories



```
code[i] = factory.create(..IFEQ);
```

```
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){...
    updatePathCondition(.., EQ);
    return jumpTarget;
  } else {...
    updatePathCondition(.., NE);
    return getNextInsn();
  }
}
```

Core: Variable Attributes

Fields

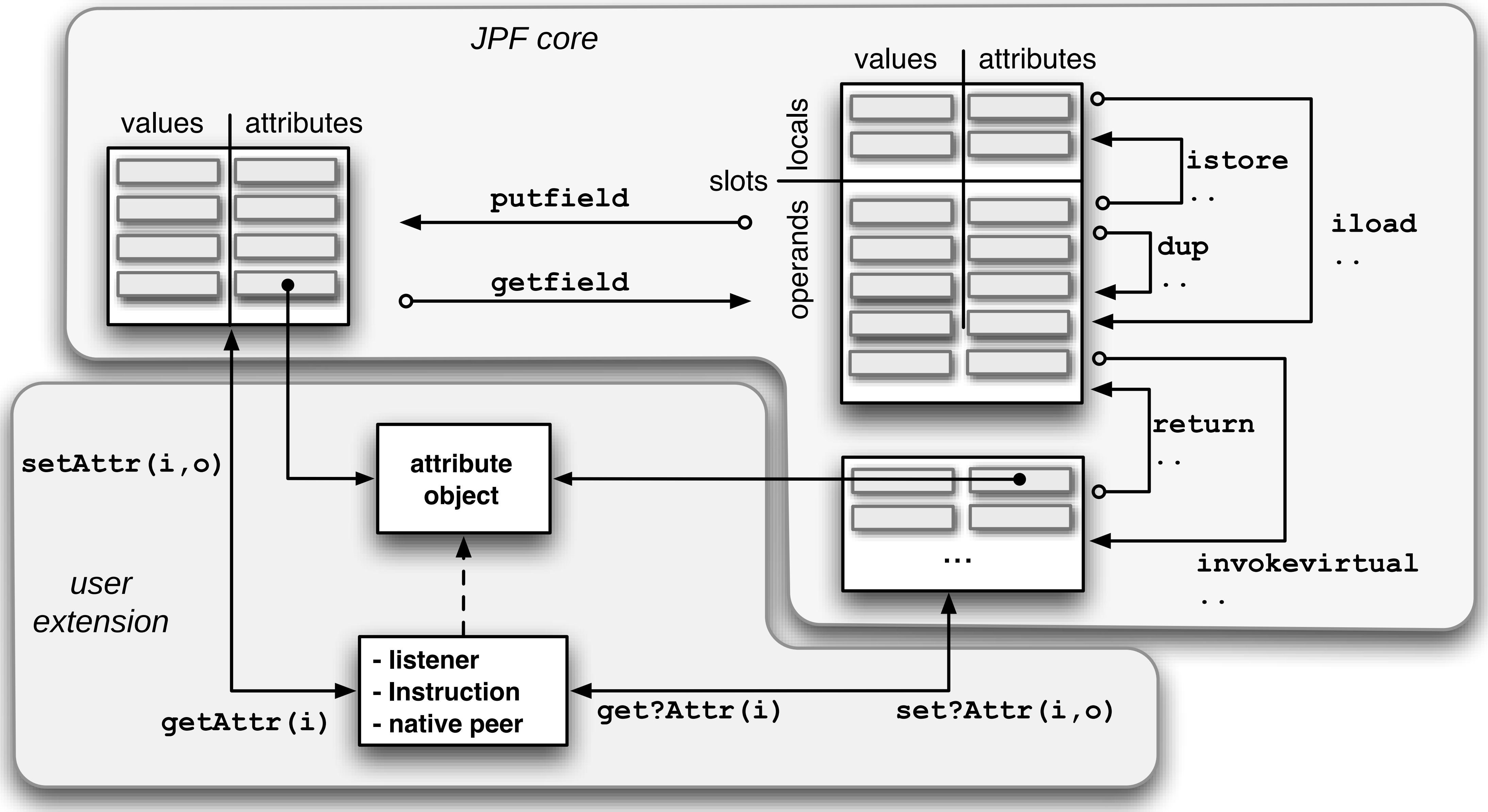
int[] values
Object[] attrs

getIntValue(idx), ...
setIntValue(idx, v), ...
getAttr(idx)
setAttr(idx,obj)

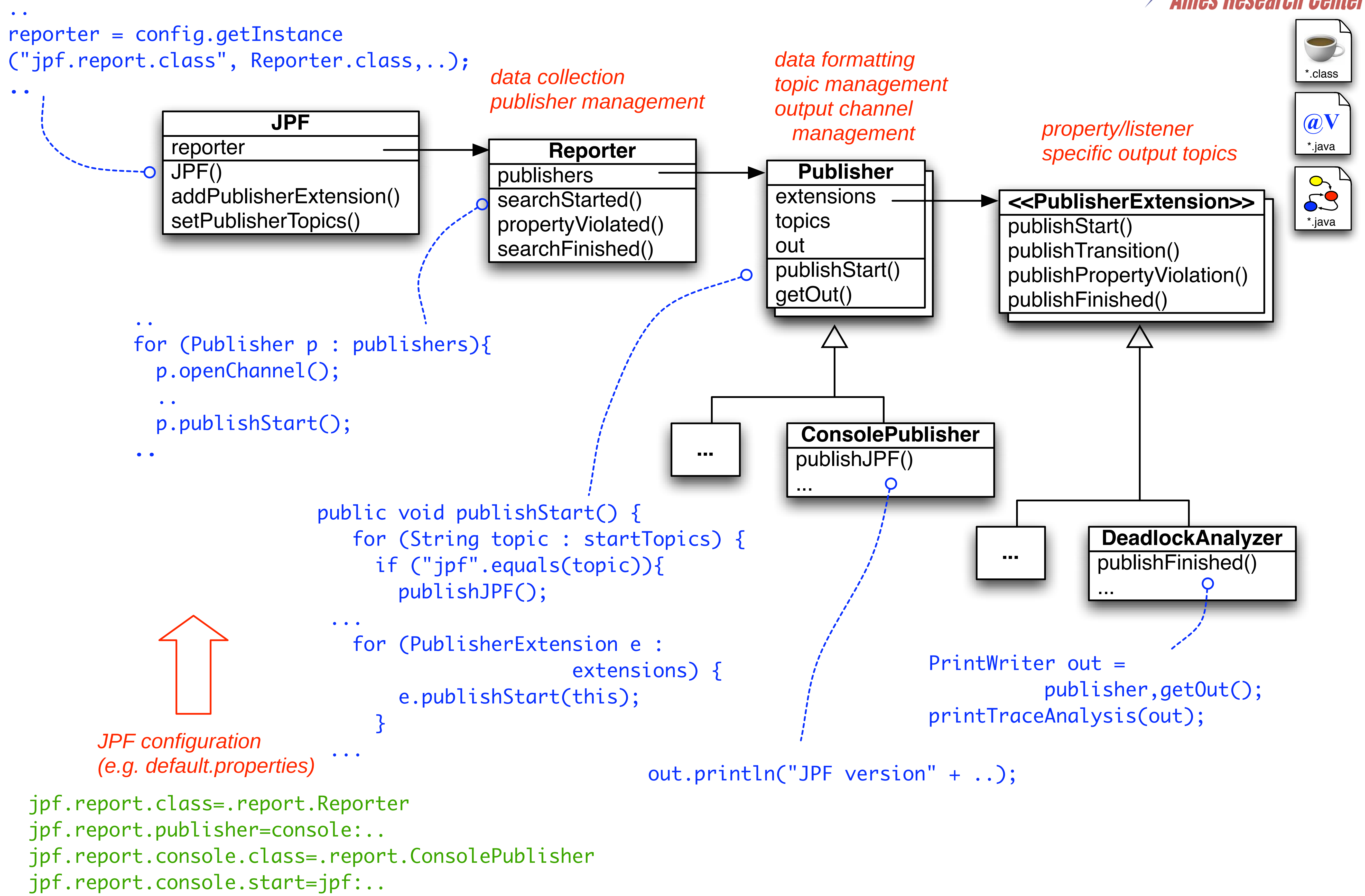
StackFrame

int[] locals
Object[] localAttr
int[] operands
Object[] operandAttr

dup(), push(), pop(), ..
getOperandAttr(idx)
setOperandAttr(idx,obj)
getLocalAttr(idx)
setLocalAttr(idx,obj)



Core: Reports



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

data collection
publisher management

data formatting
topic management
output channel
management

property/listener
specific output topics

```
..
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. default.properties)

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

Annotations: Requirements Coverage

- markup to link to arbitrary documents
- can be used for coverage statistics
- easy, low cost (tool independent), good docu

```
@Requirement("1.1.1")
public double doSomething (double d) {...}
```

```
--------------------------------- requirements coverage -----------------------------------
bytecode            basic-block         branch              methods             requirement
-------------------------------------------------------------------------------------------
...
0.80 (8/10)         0.75 (3/4)          0.00 (0/1)          1.00 (2/2)        "1.1.1"
  0.75 (6/8)          0.67 (2/3)          0.00 (0/1)          ...doSomething(D)D
  ...
------------------------------------------------------------------------------------------
0.80 (8/10)         0.75 (3/4)          0.00 (0/1)          0.67 (2/3)          0.50 (1/2)   total
```

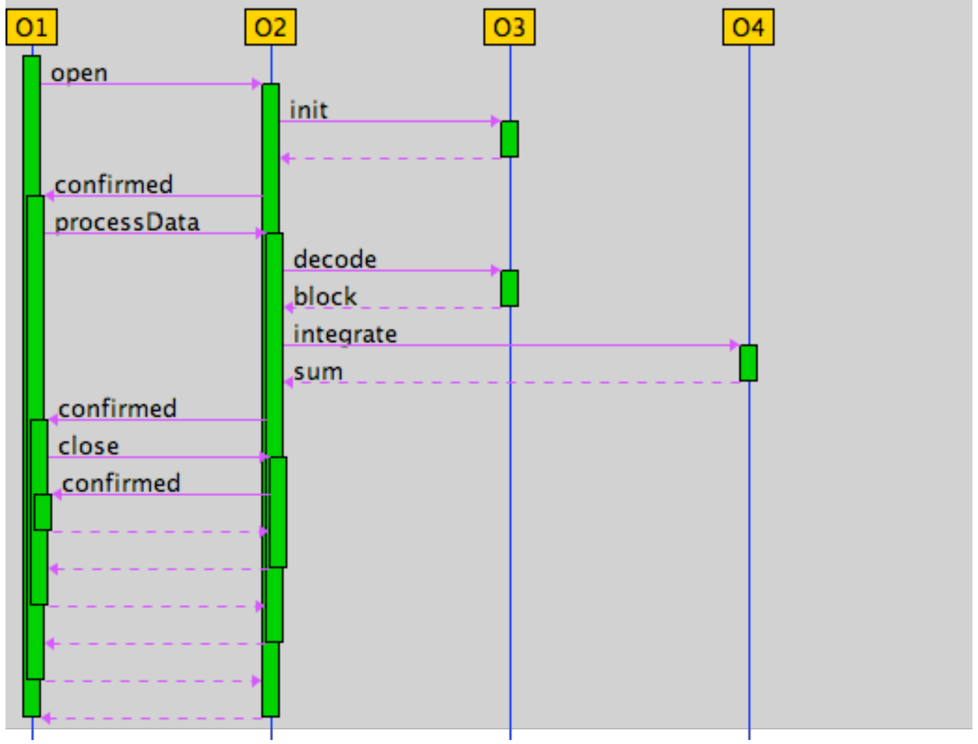
Annotations: Event Sequences

- identify (object aware) events that should be logged/analyzed for temporal properties
- example uses Alex Moffat's *Sequence* editor:

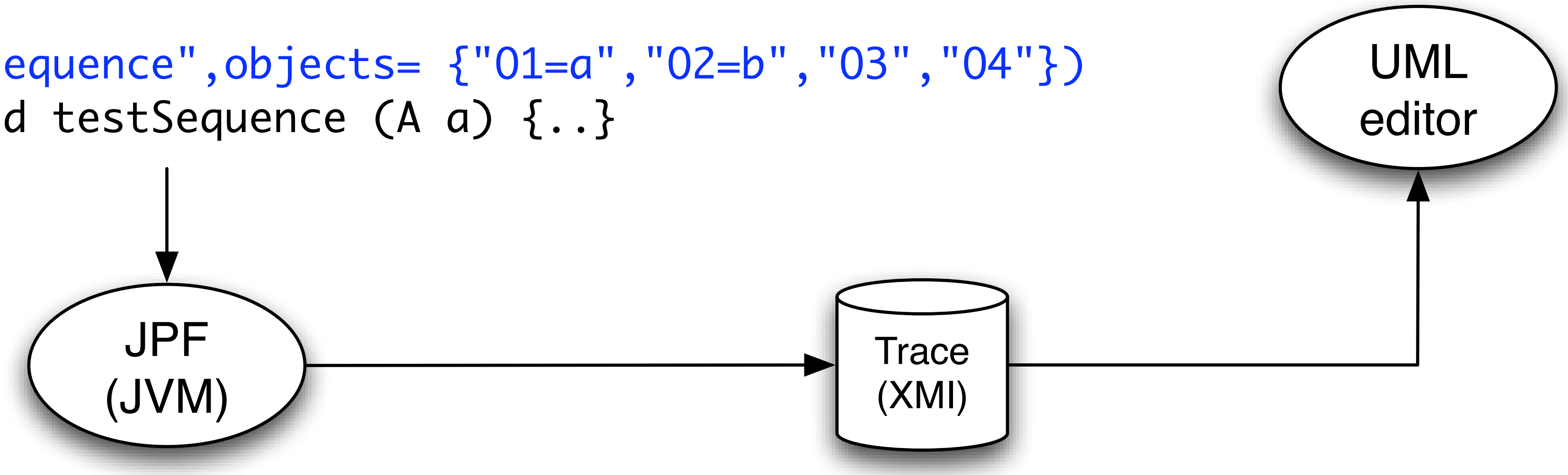
```
class A { ..
  @SequenceMethod(id="MySequence")
  public void initialize(B b) {..} ...
}

class B { ..
  @SequenceObject(id="MySequence", object="O4")
  D d;

  @SequenceMethod(id="MySequence")
  public void open(A client) {..} ...
}
```



```
@Sequence(id="MySequence",objects= {"O1=a","O2=b","O3","O4"})
public static void testSequence (A a) {..}
...
```



Annotations: PbC

- PbC == “assertions on steroids”

```
class TestContractsBase {
  @Ensures("Result < 0")
  int foo (int a){..} ...
}
```

inherited:
weakening preconditions (OR)
strengthening postconditions (AND)

```
@Invariant({"d within 40 +- 5",
            "a > 0"})
class TestContracts extends TestContractsBase {
```

evaluated before and after each public method

```
  double d = 42.1;
  int a = 42;

  @Requires("a within 10,20")
  @Ensures("old(d) >= d")
  int foo (int a){..} ..
}
```

evaluated before entry (callers responsibility)

evaluated after exit (callee responsibility)

⇩

```
...
============================================== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.AssertionError:
         invariant violated: "((d within 40+-5) && (a > 0))", values={d=142.1}
    at gov.nasa.jpf.test.TestContracts.faz(TestContracts.java:48)
    at gov.nasa.jpf.test.TestContracts.main(TestContracts.java:60)
...
```

Annotations: Testing

- *JUnit* good, but:
  - external source (sync)
  - tests can be useful documentation
  - argument variation should be supported (spec efficiency)
- in-source test specs for “simple” unit tests
  - makes developers life easier
  - tests get not lost/out of sync
- (possibly) tool independent
  - can be used with JPF
  - can be used with simple, standalone MethodTester
- annotations could be generated by symbc

test object creation *test method arguments* *test goal*

```
@Test("this(2)|this(3). (0.[56]e-10) within 0,20")
double func (double d) {..}
...
```

standalone execution (target JVM)

model checking (JPF execution)

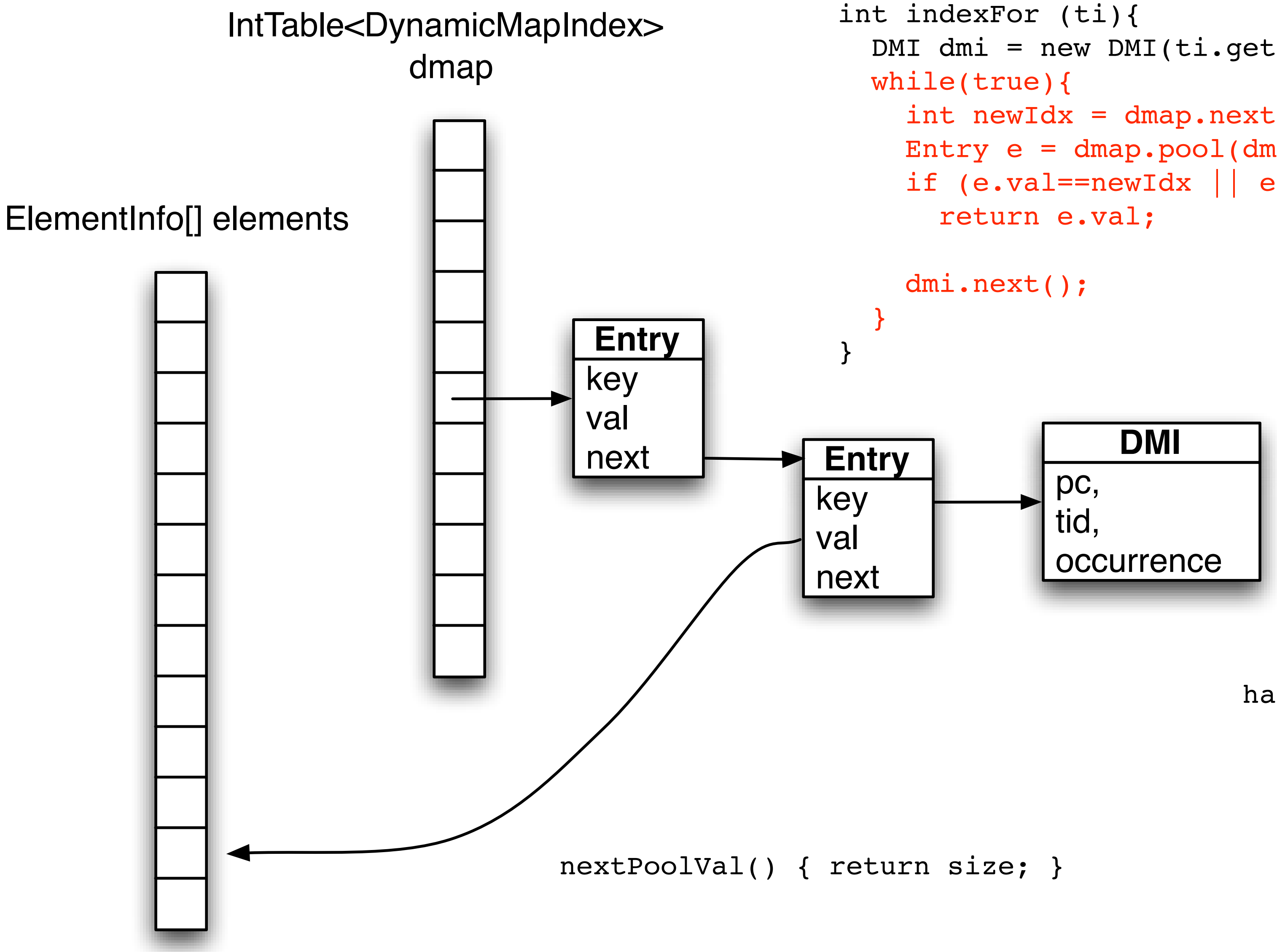
```
...
@ test spec: "this(2)|this(3).(.[56]e-10) within 0,20"
@
@ goal: 0,20
@ execute: TestMethodTest(2).func(5.0E-11)
@ returns: 2.00000000005
@ Ok

@ execute: TestMethodTest(2).func(6.0E-11)
@ returns: 2.00000000006
@ Ok

@ execute: TestMethodTest(3).func(5.0E-11)
@ returns: 3.00000000005
@ Ok

@ execute: TestMethodTest(3).func(6.0E-11)
@ returns: 3.00000000006
@ Ok
...
```

The Big Fix: Heap (1)



```
int indexFor (ti){
  DMI dmi = new DMI(ti.getPC())
  while(true){
    int newIdx = dmap.nextPoolVal();
    Entry e = dmap.pool(dmi);
    if (e.val==newIdx || elements[newIdx]==null)
      return e.val;

    dmi.next();
  }
}
```

```
next() { occurrence++; }
```

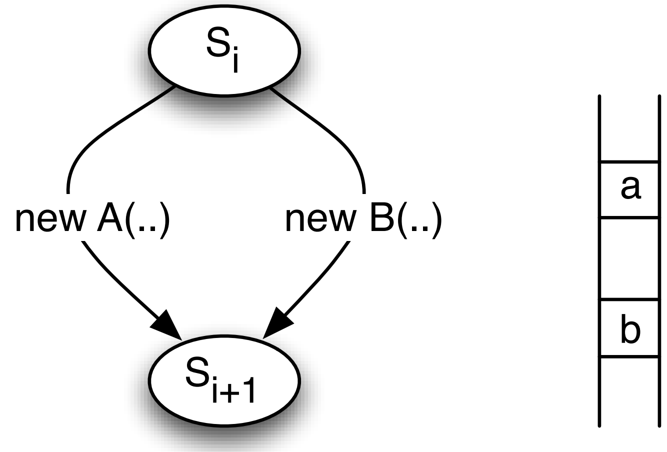
```
hashcode = pc.getPosition()
             + tid + occurrence
```

```
nextPoolVal() { return size; }
```

```
int getTableIndex(E key) {
  return hash(key.hashCode());
}
```

```
Entry lookup(E key,int idx) {
  Entry e=tbl.get(idx);
  while(e!=null){
    if (e.key.equals(key))
      return e;
    cur=cur.next;
  }
  return null; //free index
}
```

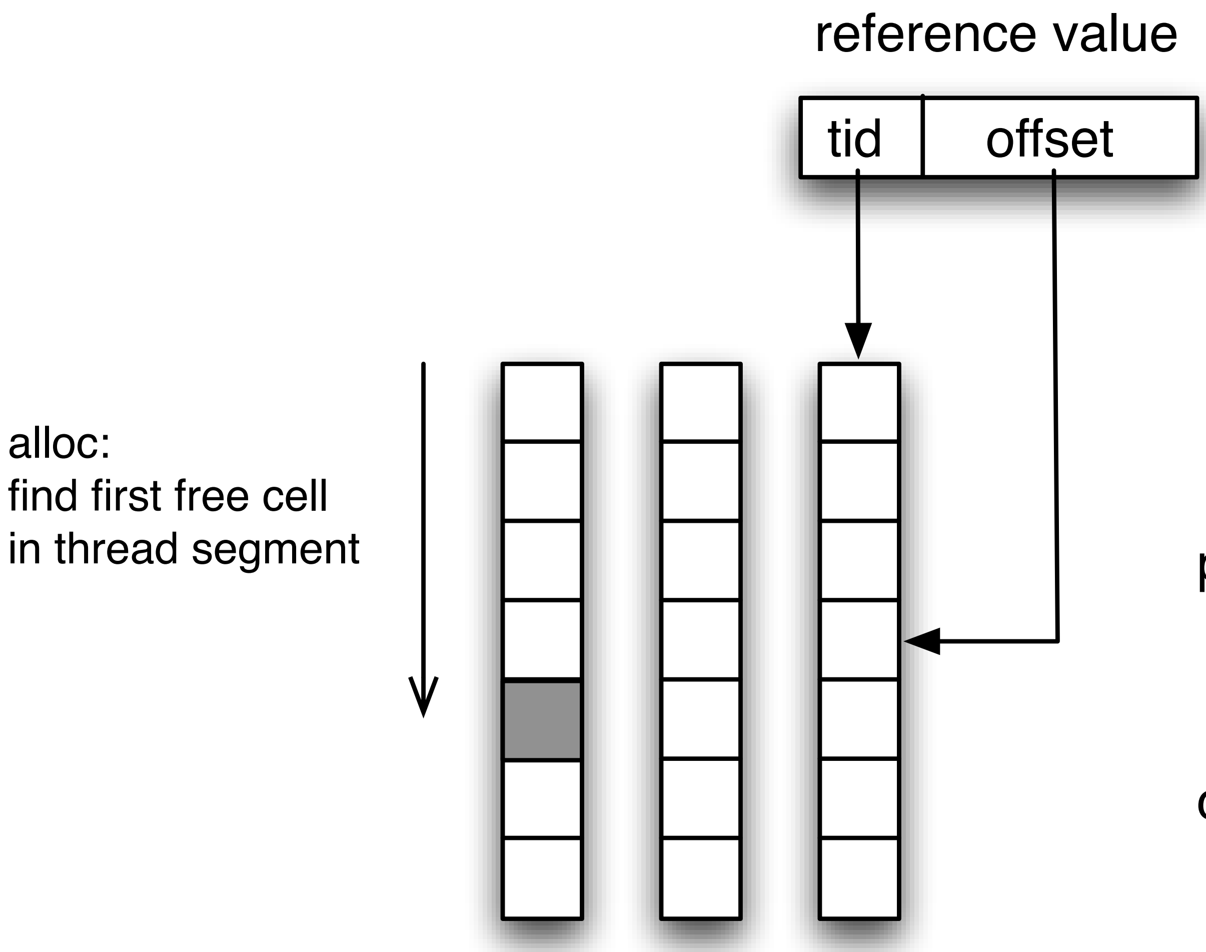
```
Entry pool(E key) {
  int idx = getTableIndex(key)
  Entry e - lookup(key,idx)
  if (e==null){
    e = new Entry(key,size++)
    addFirst(idx, e)
  }
  return e;
}
```



Heap Symmetry =
reference values are scheduling order invariant

The Big Fix: Heap (2)

- we need constant alloc time
- challenge is not better heap data structure, but replacement of DynamicArea dependencies (Serializer/Restorer)
- implementation could be SparseClusteredArray:



pro

fast alloc
efficient data structure (SparseArray w/ in-use bitmap)

con

size limit (>threads, max obj/thread)
heuristic, might not guarantee heap symmetry
(alive creator hands off ref, variable thread start order, ..)