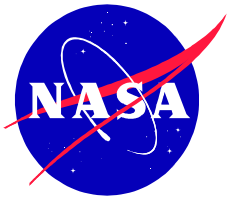


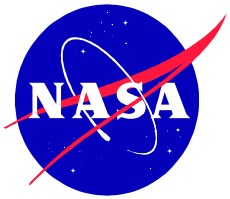
Java Model Checking

Willem Visser, Klaus Havelund, Guillaume Brat,
SeungJoon Park, Flavio Lerda

NASA Ames
Automated Software Engineering Group

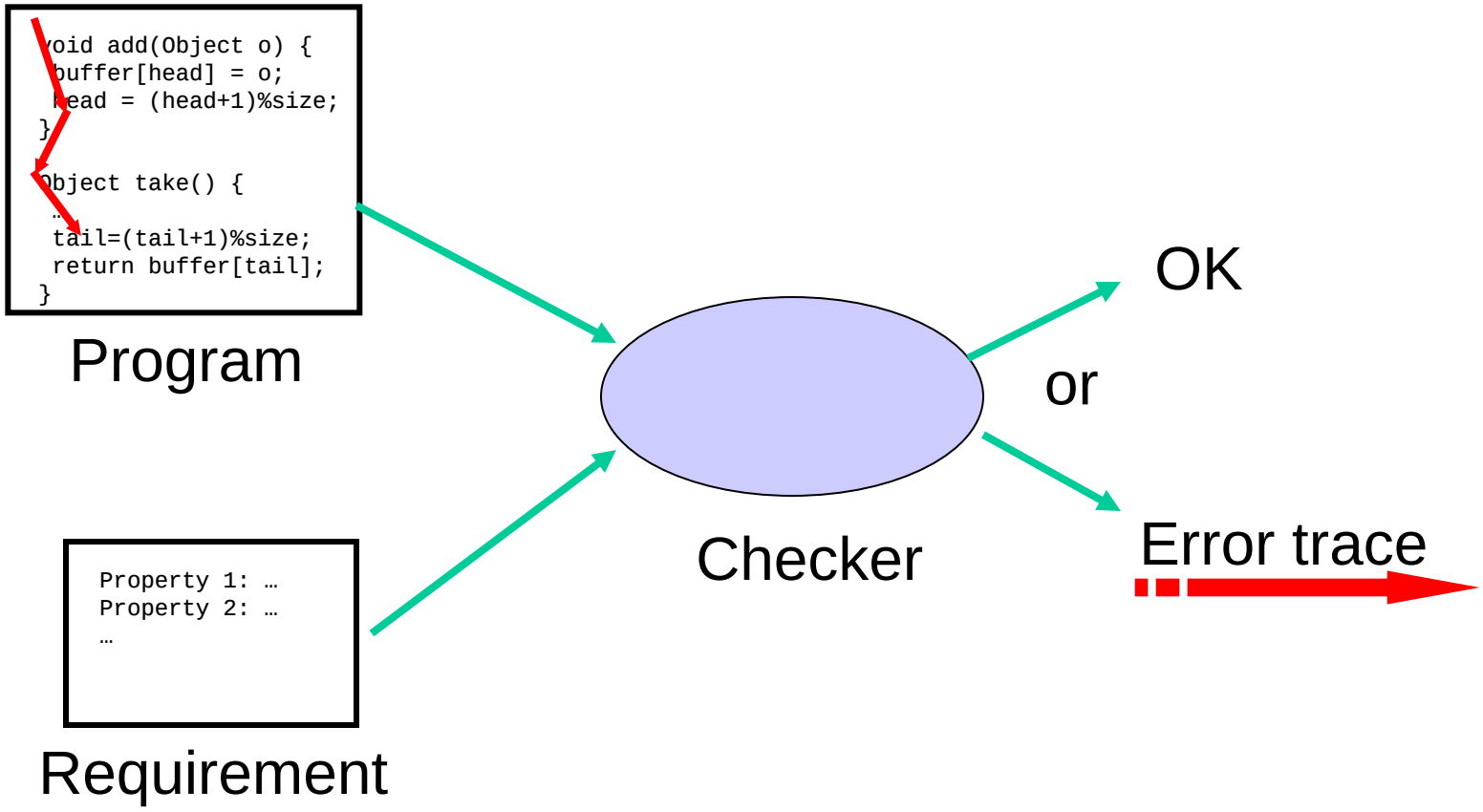
in collaboration with the
Bandera group at Kansas State

Future of Software Verification

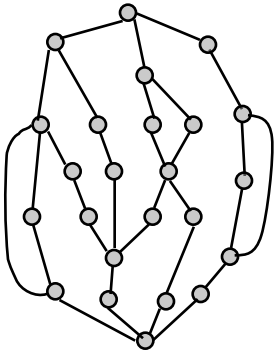


- Software is becoming more complex and expensive to develop.
  - Software failure can cost millions
  - Cost of a software engineer is going up
- CPU cycles are becoming cheaper
- Future: Automated Fault Detection
- *Model Checking Programs?*
- Can model checkers be used by programmers?
  - Only if it takes real programs as input

The Dream


void add(Object o) {
buffer[head] = o;
head = (head+1)%size;
}
Object take() {
…
tail=(tail+1)%size;
return buffer[tail];
}
Program
Property 1: …
Property 2: …
…
Requirement
Checker
OK
or
Error trace

Model Checking



Finite-state model

$\Box(\Phi \rightarrow \Diamond\, \Omega)$

Temporal logic formula

Model Checker

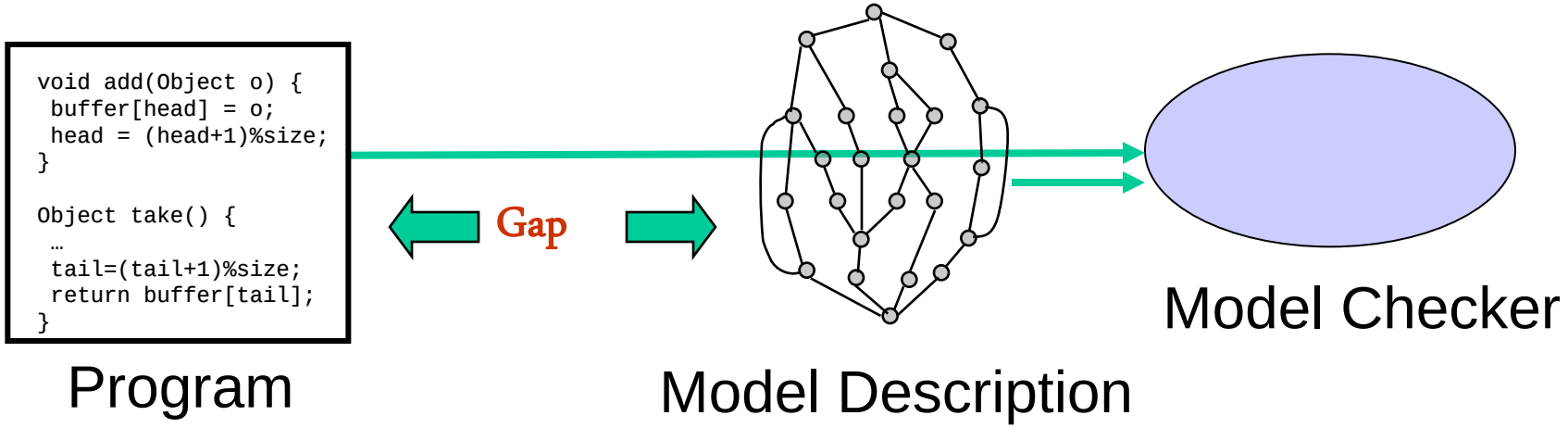
OK

or

Error trace

```
Line 5: …
Line 12: …
Line 15:…
Line 21:…
Line 25:…
Line 27:…
   …
Line 41:…
Line 47:…
```

Model Construction Problem



- **Semantic gap:**

  Programming Languages

  methods, inheritance, dynamic creation, exceptions, etc.

  Model Description Languages

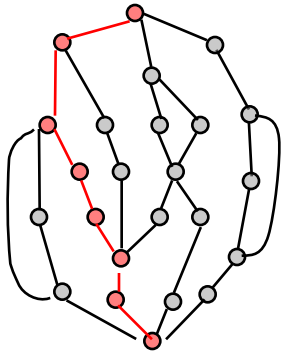
  automata

Java PathFinder

```
void add(Object o) {
 buffer[head] = o;
 head = (head+1)%size;
}

Object take() {
 …
 tail=(tail+1)%size;
 return buffer[tail];
}
```

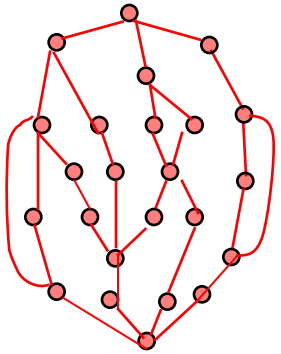
```
0:    iconst_0
1:    istore_2
2:    goto    #39
5:    getstatic
8:    aload_0
9:    iload_2
10:   aaload
```

JVM



Controllable Scheduler

Special JVM ⟷ Model Checker



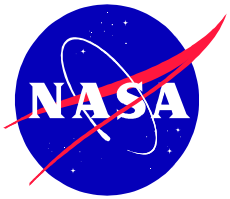
Memory not Speed

Modular Design

Key Points

- Models can be infinite state
  - Depth-first state graph generation (explicit-state MC)
  - Errors are real
  - Verification can be problematic (Abstraction required)
- All of Java is handled except native code
- Nondeterministic Environments
  - JPF traps special nondeterministic methods
- Properties
  - User-defined assertions and deadlock
  - Buchi automata based properties (recent addition)
- Source level error analysis (with Bandera tool)

Example Environments



- Thread making OS calls

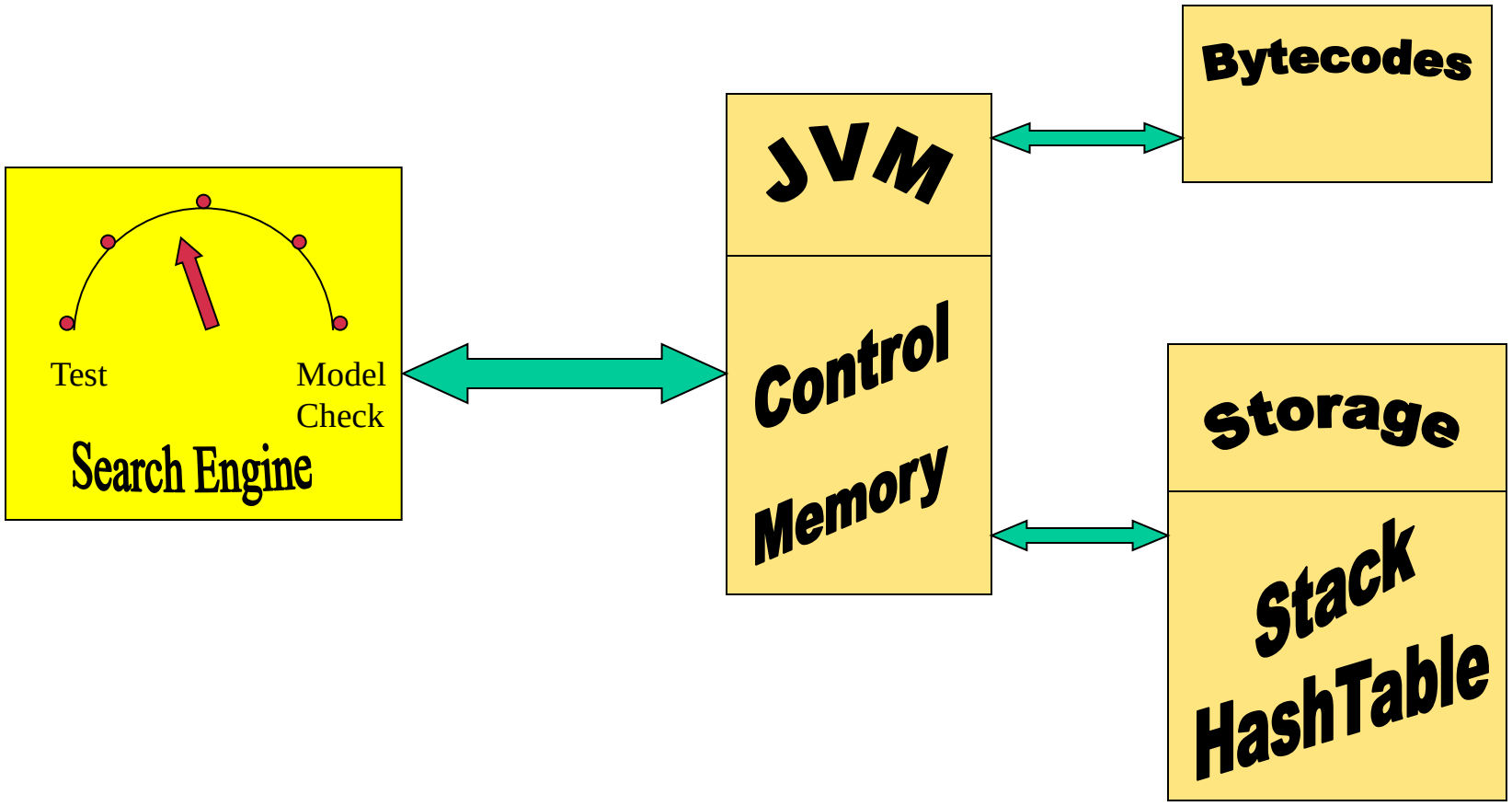
```
If (Verify.randomBool())
  Kernel.yieldCPU();
Else
  Kernel.deleteThread();
```

- Method returning 5 values

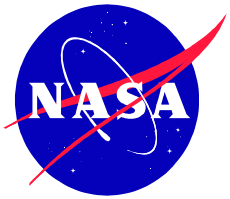
```
Return Verify.random(4);
```

```
public class Signs {
  public static final int NEG  = 0;
  public static final int ZERO = 1;
  public static final int POS  = 2;

  public static int abstract(int n) {
    if (n < 0) return NEG;
    if (n == 0) return ZERO;
    if (n > 0) return POS;
  }

  public static int add(int arg1, int arg2) {
    if (arg1==NEG  && arg2==NEG)  return NEG;
    if (arg1==NEG  && arg2==ZERO) return NEG;
    if (arg1==ZERO && arg2==NEG)  return NEG;
    if (arg1==ZERO && arg2==ZERO) return ZERO;
    if (arg1==ZERO && arg2==POS)  return POS;
    if (arg1==POS  && arg2==ZERO) return POS;
    if (arg1==POS  && arg2==POS)  return POS;
    return Verify.random(2);
    /* case (POS,NEG), (NEG,POS) */
  }
}
```

JPF Structure

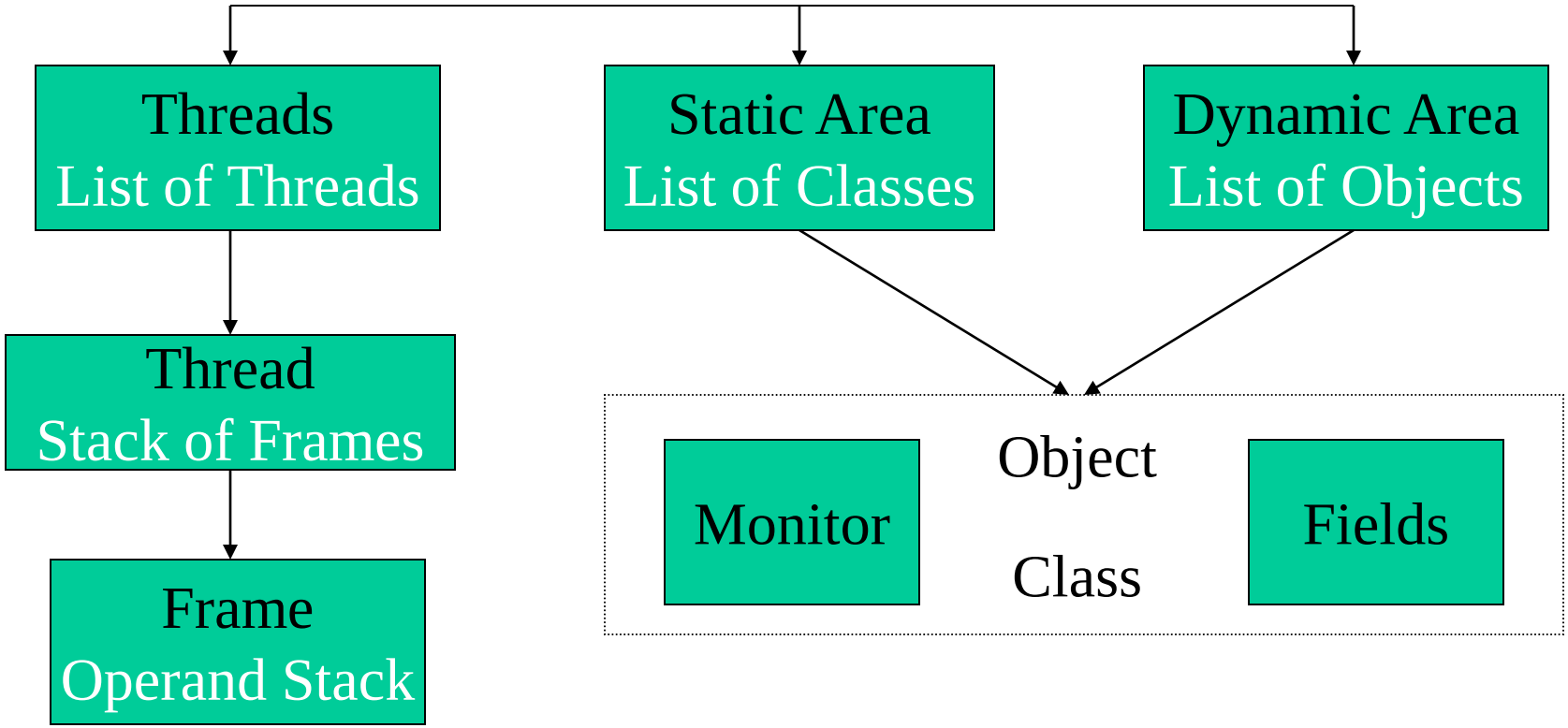


Our JVM

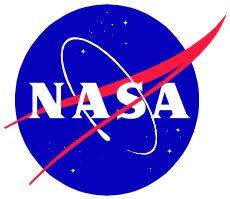


- Written in Java
  - *Java-in-Java* at SUN and *Rivet* at MIT
  - Exploit Java in Java at every opportunity
- Initial implementation took < 3 man-months!
  - Use *JavaClass* package
    - Class loading, Internal class file structure, etc.
- State encoded in complex data-structures

JVM State


Threads
List of Threads
Static Area
List of Classes
Dynamic Area
List of Objects
Thread
Stack of Frames
Frame
Operand Stack
Object
Monitor
Class
Fields

FDIV Bytecode



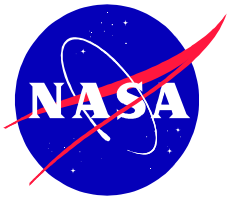
- If either v1 or v2 is NaN then the result is NaN
- Division of infinity by infinity results in NaN
- Division of 0 by 0 results in NaN; division of zero by any other finite value results in a signed zero…
- …

```
public class FDIV {
  public InstructionHandle execute(SystemState s) {
    KernelState ks = s.getKernelState();
    ThreadInfo th =
      s.getScheduler().locateRunningThread(ks);

    StackFrame sf = th.top();
    Float v1 = (Float)sf.pop();
    Float v2 = (Float)sf.pop();
    sf.push(new Float(v2.floatValue() /
                      v1.floatValue()));
    return th.currentPC().getNext();
  }
}
```

Major Issues

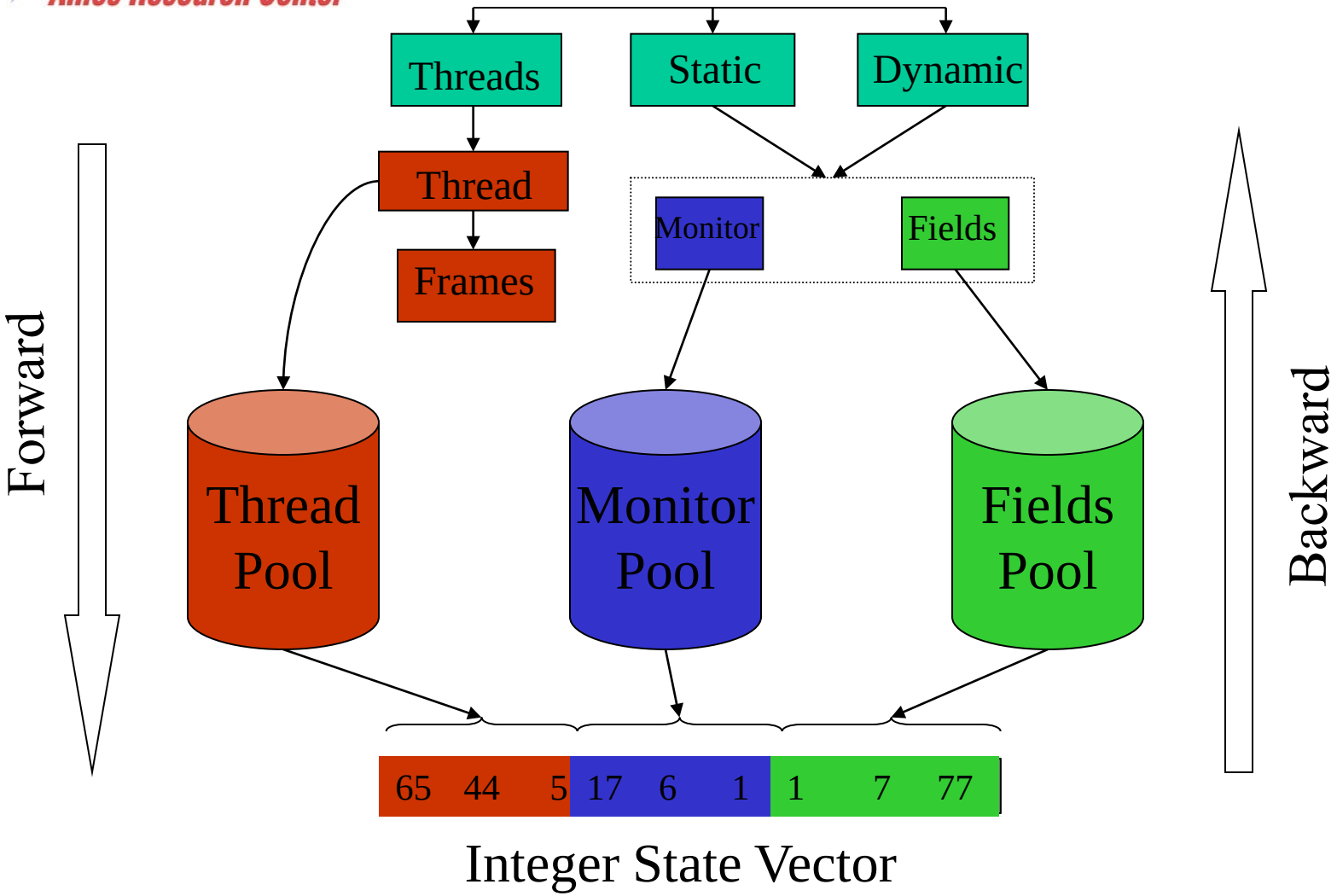
- States are too complex
  - Collapse to integer vector
- State-space too large
  - Symmetry reductions
  - Garbage collection
  - Partial-order reductions
  - Abstraction

Reducing State Size

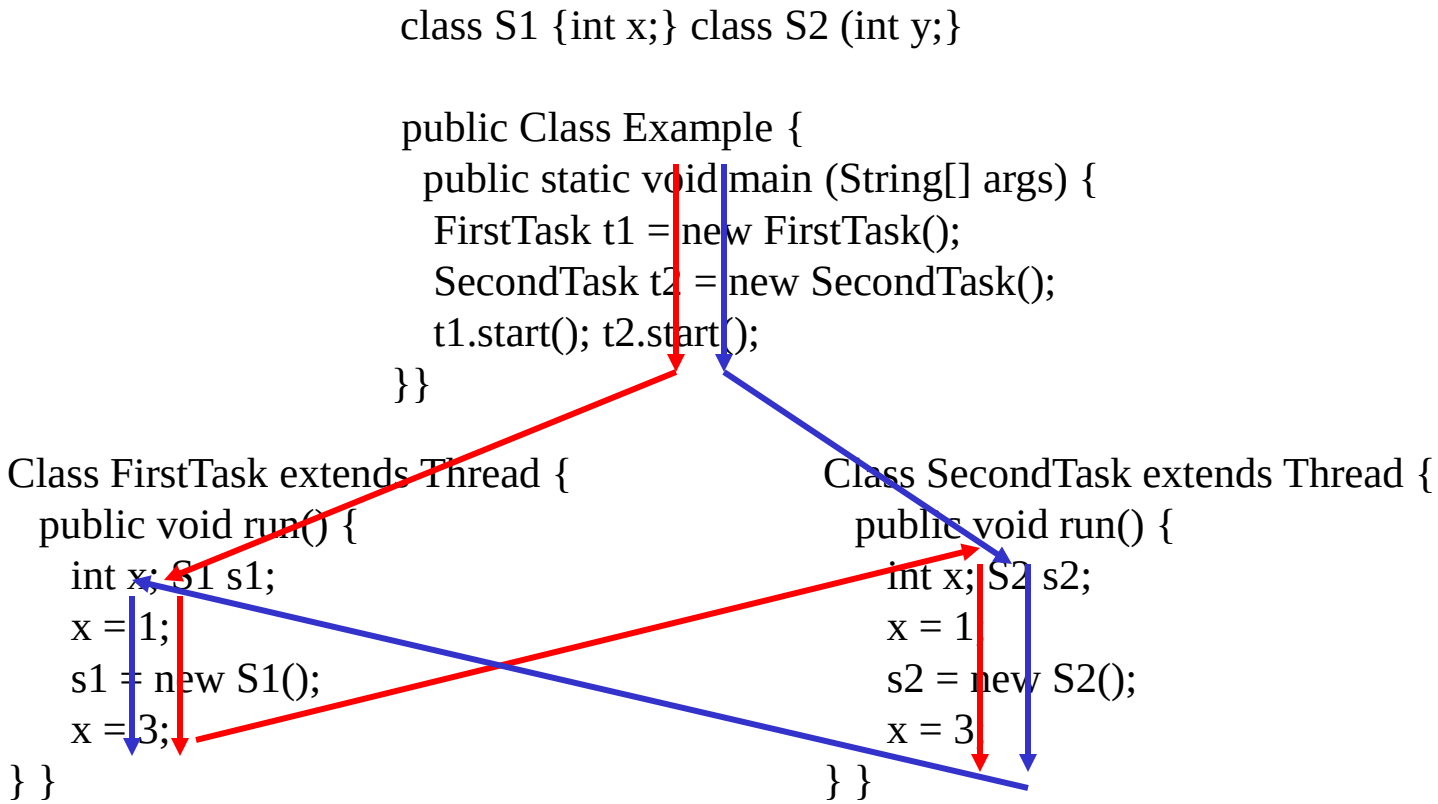


- Use intermediate tables/pools for each major class
  - Threads, Fields, Monitors
- Each time an object of the class must be stored, see if it is in the table, if so return the index, else insert it into next open slot and return the index
- Works well if tables don't become to big
- Optimization
  - Only calculate index when object has changed.
- Backtracking
  - Reverse operation

State Compression



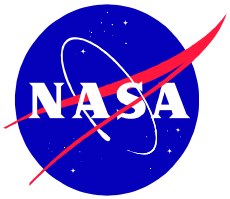
Symmetry and Partial-order Reduction



- jpf Example 155 states
- Symmetry reduction 105 states
- Partial-order reduction 38 states

2.5M to 100K

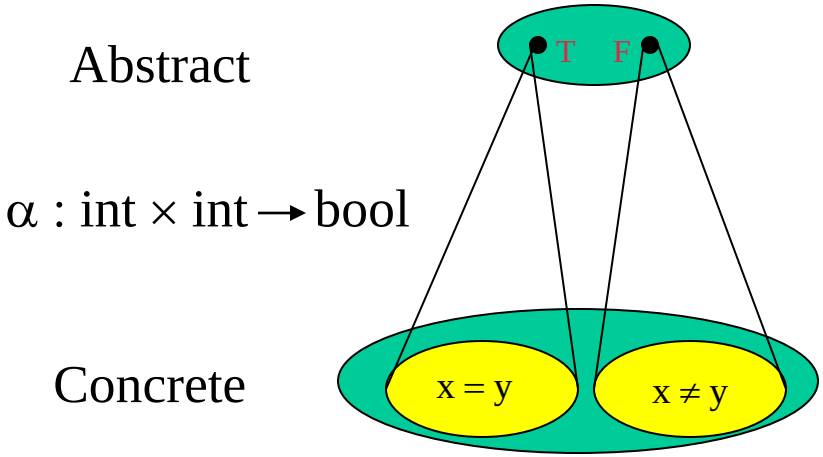
Garbage Collection

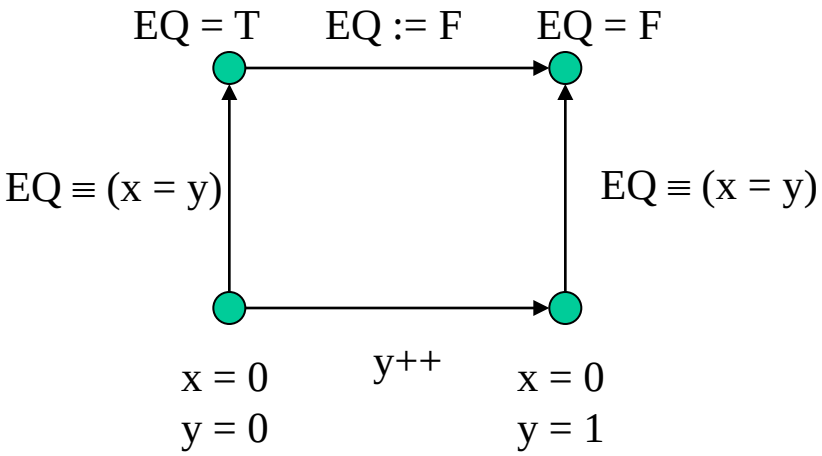



```
class gc {
  public static void main (String[] args) {
    while(true) {
      System.out.println("0");
} } }
```

- Infinite State program
  - Garbage makes the state grow…
- JPF supports
  - Mark-and-Sweep
  - Reference counting

Predicate Abstraction





- Mapping of a concrete system to an abstract system, whose states correspond to truth values of a set of predicate
- Create abstract state-graph during model checking, or,
- Create an abstract transition system before model checking

Example Abstraction

Predicate: $B \equiv (x = y)$

Concrete Statement

$y := y + 1$

Abstract Statement

Step 1: Calculate pre-images

$\{x = y + 1\}\ y := y + 1\ \{x = y\}$

$\{x \neq y + 1\}\ y := y + 1\ \{x \neq y\}$

Step 2: Rewrite in terms of predicates

$\{x = y + 1\}\ y := y + 1\ \{B\}$

$\{B\}\ y := y + 1\ \{\sim B\}$

Step 2a: Use Decision Procedures

$x = y \rightarrow x = y + 1$ $\quad x = y \rightarrow x \neq y + 1$

$x \neq y \rightarrow x = y + 1$ $\quad x \neq y \rightarrow x \neq y + 1$

Step 3: Abstract Code

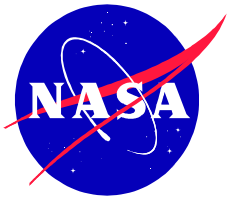
IF B THEN B := false

ELSE B := true | false

JPF's Java Abstraction

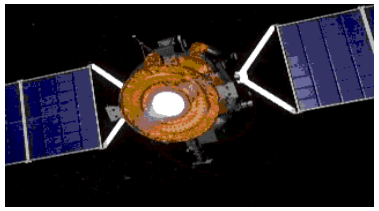
- Annotations used to indicate abstractions
  - `Abstract.remove(x);`
    `Abstract.remove(y);`
    `Abstract.addBoolean("EQ", x==y);`
- Tool generates abstract Java program
  - Using Stanford Validity Checker (SVC)
  - JVM is extended with nondeterminism to handle over approximation
- Abstractions can be local to a class or global across multiple classes
  - `Abstract.addBoolean("EQ", A.x==B.y);`
  - Dynamic predicate abstraction, since it works across instances

Runtime Analysis



- Execute program once, and accumulate information to be analyzed
- Analysis can reveal an error potential although an error did not occur during the run
  - Looking for “footprints” of errors
- Data race violations with Eraser algorithm
- Lock order violations that lead to deadlock
- JPF can use runtime analysis to guide it towards errors

Example Usage



Remote Agent
Deadlocked in Space

Translated offending code from Lisp to Java

Program is infinite state, apply predicate abstraction

Still too large to check with JPF (10^{60} states)

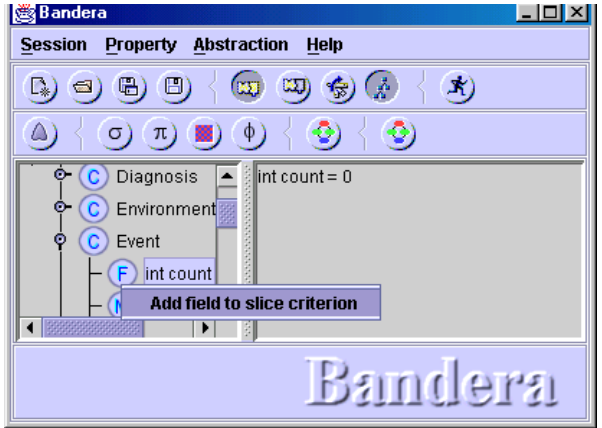
Do run-time analysis with JPF and find that the *count* variable in the *Event* class is accessed unprotected, i.e. there is a race-violation!

Dynamic slicing

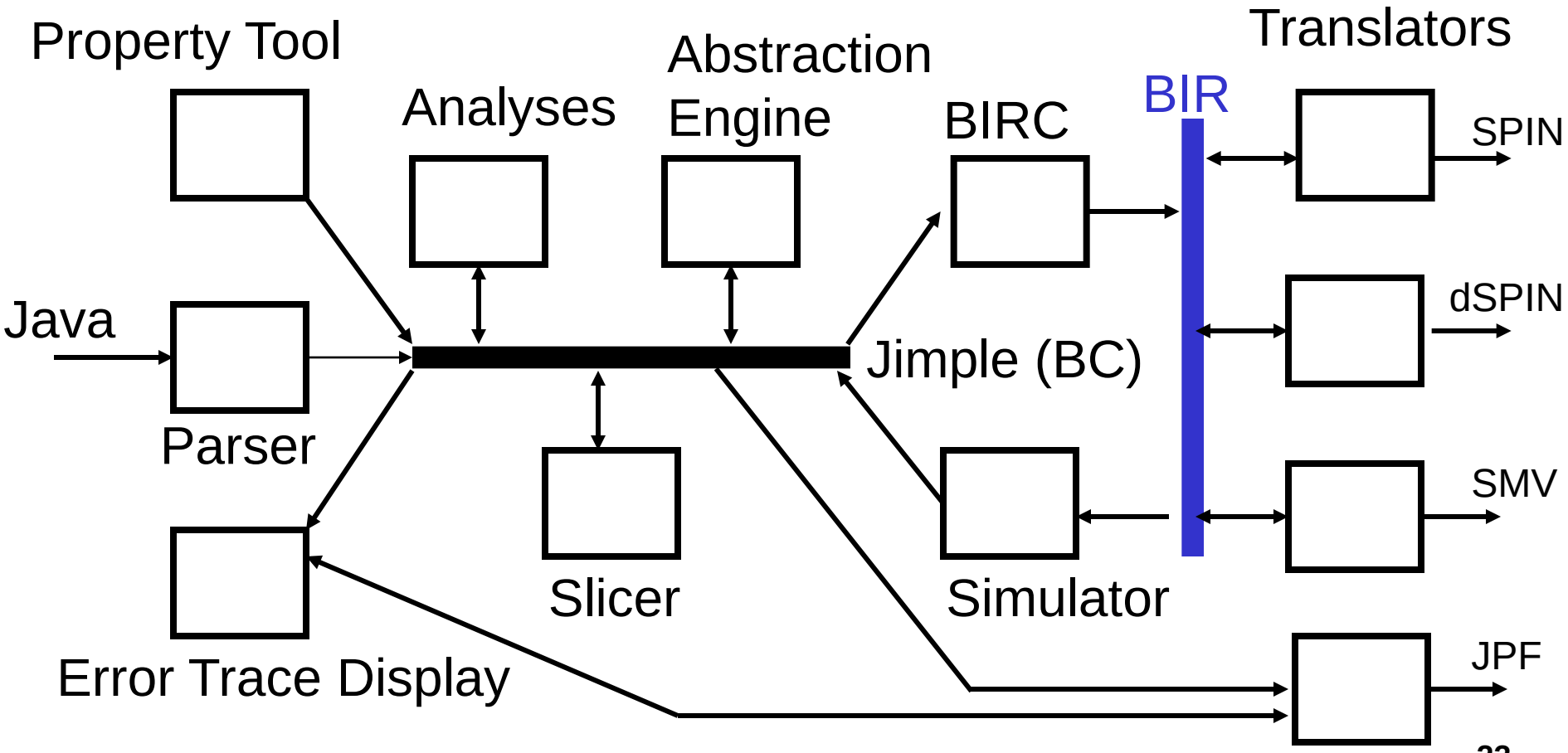
Use runtime analysis as front end to model checking

Finds deadlock within 15 seconds

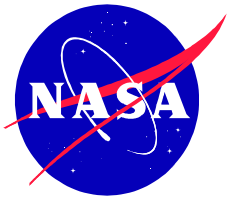
Static slicing



Bandera Architecture

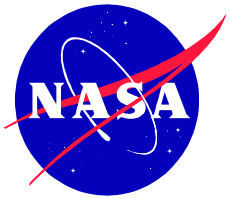


Conclusions



- Low-hanging fruit principle
  - Errors always obvious (in hindsight!)
  - Model checkers are good at finding obvious errors
- Combine many different techniques
  - Abstraction, slicing, runtime analysis, etc.
- Current and Future work
  - Adding property specification language
  - Finding “real” counter-examples during abstraction
  - Defining “environments” for Java programs
  - Test coverage for model checking?
  - Parallel model checking

At Last!



JPF is Available

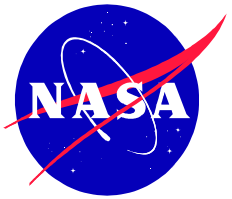
http://ase.arc.nasa.gov/jpf

So is Bandera

The Numbers

- Speed
  - 3000 to 4000 states/second
  - Order of magnitude slower than Spin
- Largest example
  - 1500 lines of DEOS (translated from C++)
  - 6 threads, 22 Classes, 72 objects

Automated Software Engineering Group



- Apply model checking to NASA Software
  - ¼ of the group (5 people)
- 1997
  - Found 5 errors in Deep-Space 1 software
  - Hand translation Lisp to Promela using “meat-axe” treatment
- 1998/1999
  - Rediscovered subtle timing error in DEOS real-time operating system (1500 lines of C++)
  - Hand translation C++ to Promela with 1-to-1 correspondence
- 1998/1999
  - JPF1 model checker translates Java to Promela
  - Completely automated