Java Runtime Error Detection Through Static Data Flow Analysis

...avoiding runtime errors at development time.

Dr. Fridtjof Siebert
CTO, aicas
30. May 2007

Goals of this DFA

Correctness Analysis

- Proof absence of runtime errors
- Gain confidence in code
- Provide input for other tools

Not a goal are

- Compiler optimizations:
  - omit runtime checks
  - reduce application size
  - **But: very dangerous!**

Data Flow Analysis Basics

Idea

Start at main() method and follow all assignments in an iterative loop.

Result is

- Set of calls
- Set of values for each variable

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)
- variables = { } (set of variables x values)

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)
- variables = { } (set of variables x values)

Iterate

- do {

} **until fix point is reached**;

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)
- variables= { } (set of variables x values)

Iterate

- do {

  calls = calls ∪

} until fix point is reached;

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)
- variables = { } (set of variables x values)

Iterate

- do {

$$calls = calls \cup \{c \mid \exists x: x \in calls \wedge \text{"}\mathbf{c()}\text{"} \in x\}$$

} until fix point is reached;

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)
- variables = { } (set of variables x values)

Iterate

- do {

  calls = calls $\cup$ {c | $\exists$ x: x $\in$ calls $\wedge$ "c()" $\in$ x }

  variables = variables $\cup$

  } until fix point is reached;

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)
- variables = { } (set of variables x values)

Iterate

- do {

  calls = calls $\cup$ {c | $\exists$ x: x $\in$ calls $\wedge$ "c()" $\in$ x }

  variables = variables $\cup$

  { (a,X) : $\exists$ x: x $\in$ calls $\wedge$ "**a = new X()**" $\in$ x) } $\cup$

  } until fix point is reached;

Data Flow Analysis Basics

Start with the following

- calls = { main } (set of calls)
- variables = { } (set of variables x values)

Iterate

- do {

  calls = calls ∪ {c | ∃ x: x ∈ calls ∧ "c()" ∈ x }

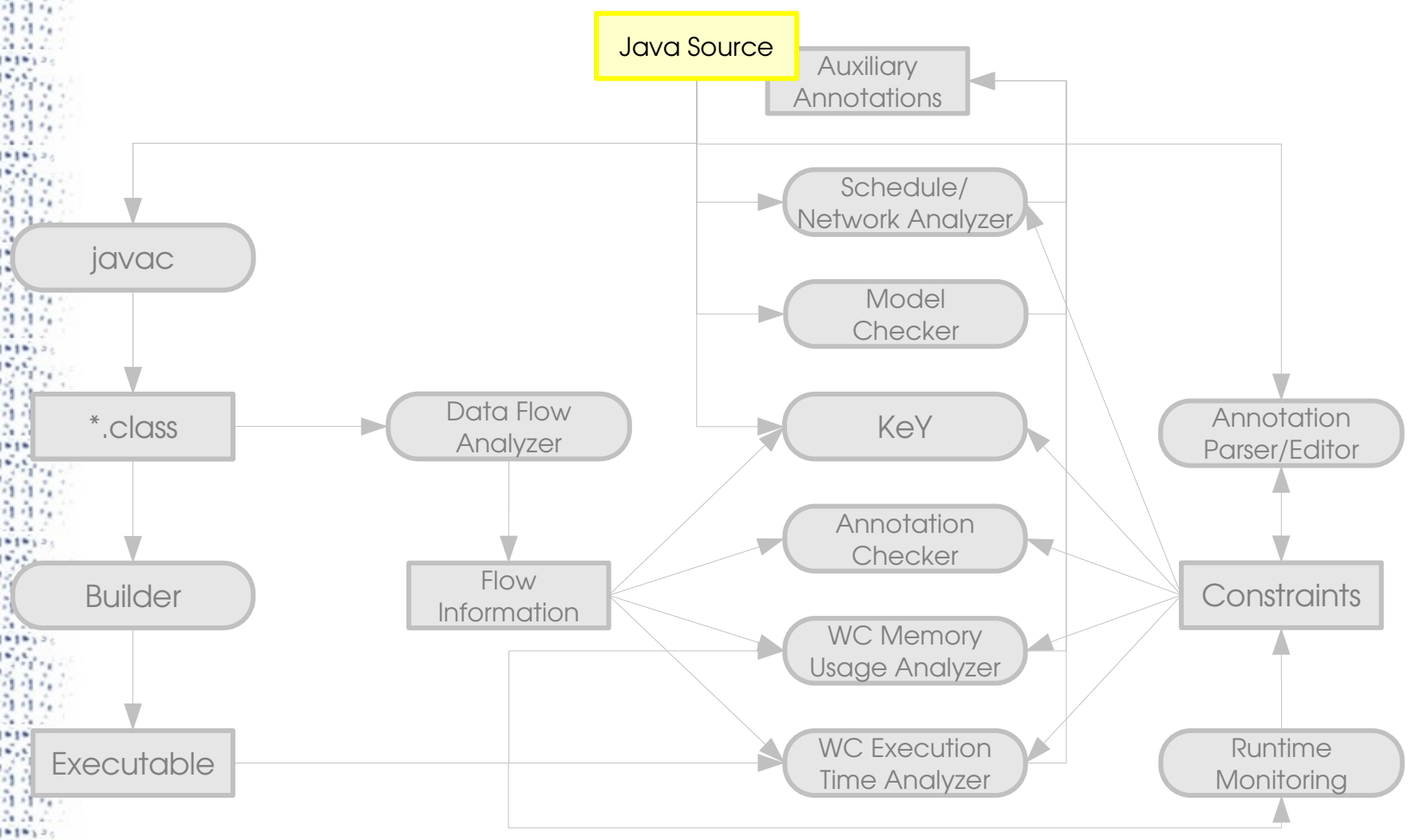
  variables = variables ∪

  { (a,X) : ∃ x: x ∈ calls ∧ "a = new X()" ∈ x) } ∪

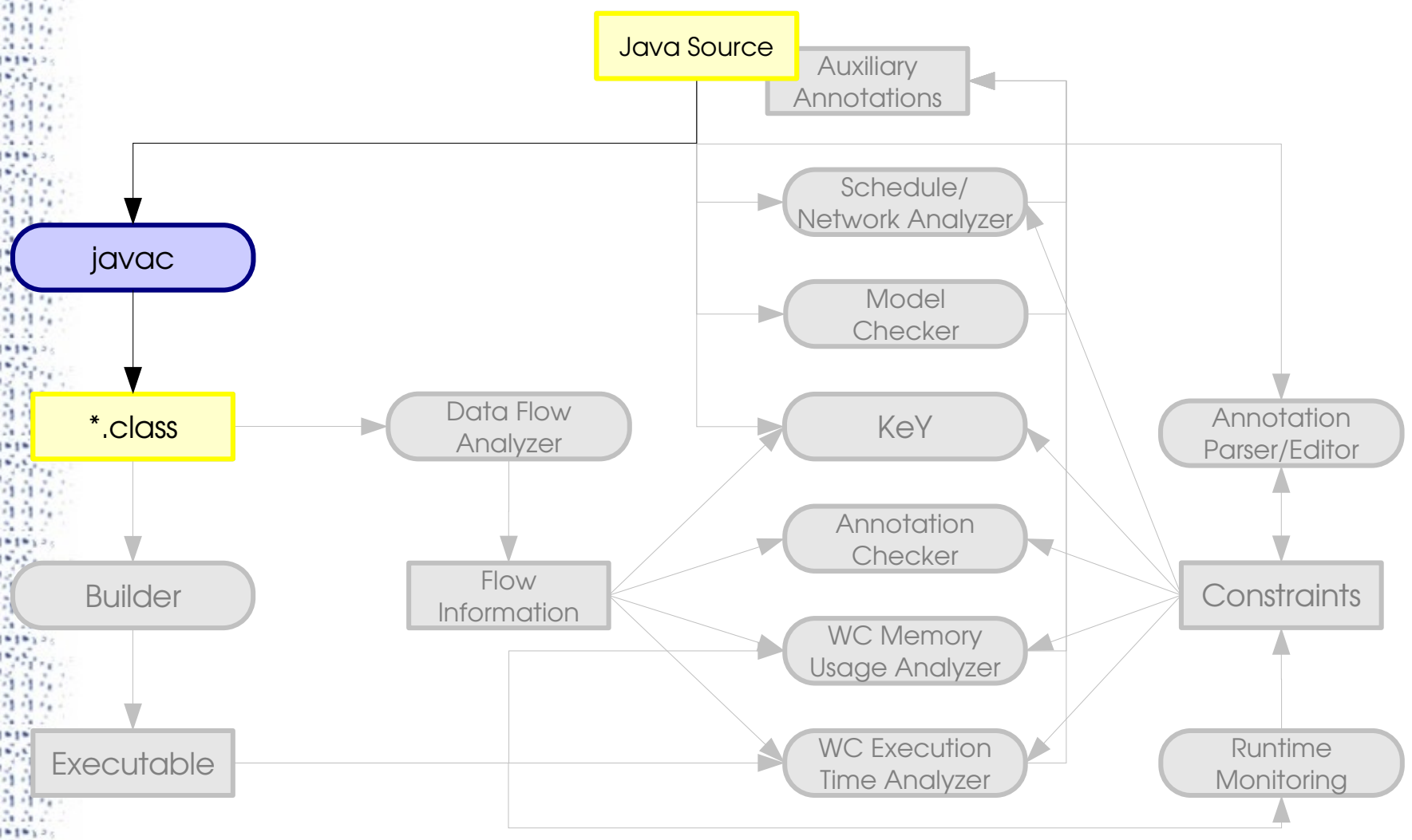
  { (a,v) : ∃ x: x ∈ calls ∧ "**a = b**" ∈ x ∧ (b,v) ∈ values }

  } until fix point is reached;

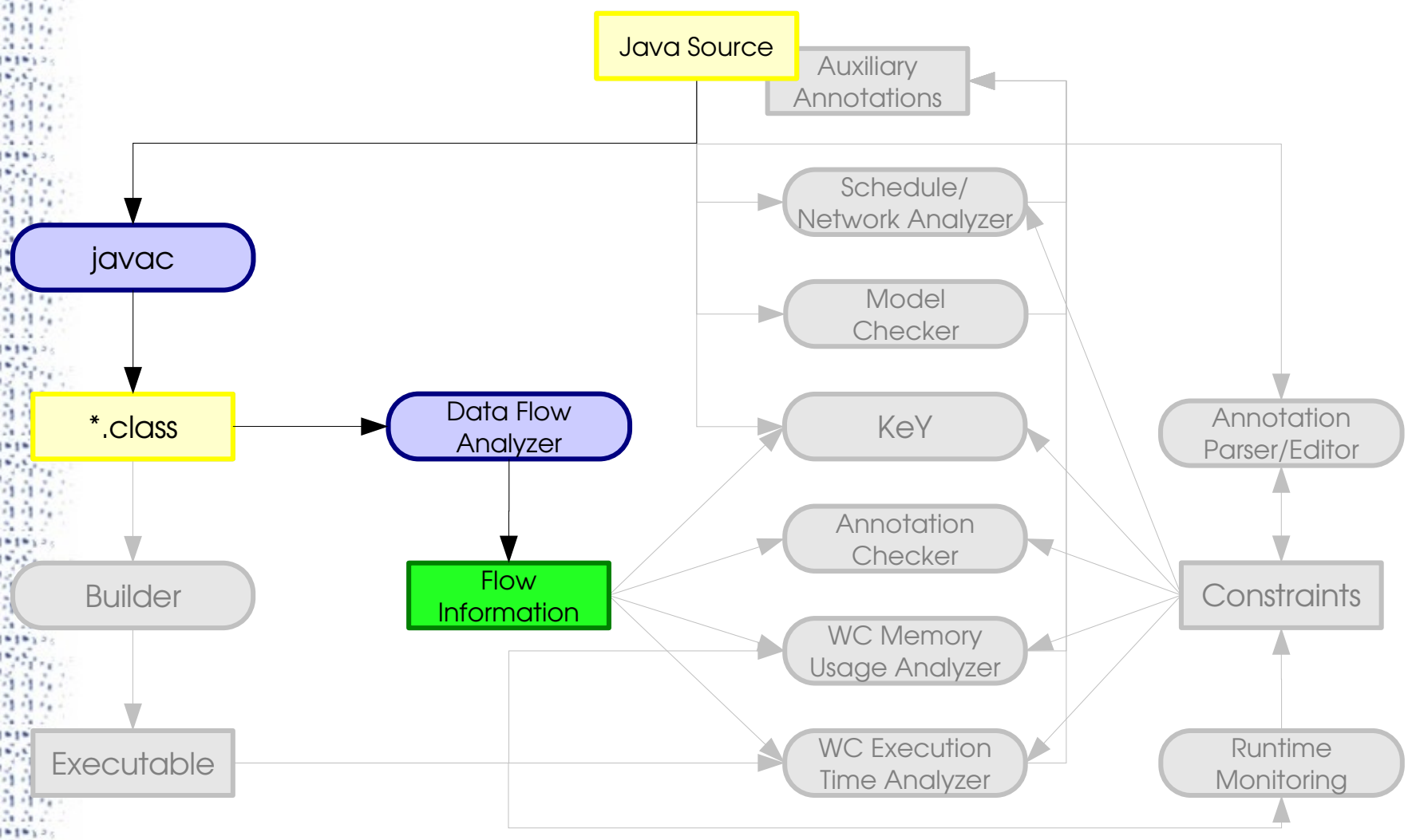
DFA as part of larger Tool Chain



DFA as part of larger Tool Chain



DFA as part of larger Tool Chain



Application-wide Data-Flow Analysis

Problems

Identification of objects by type not sufficient

Granularity: Methods and Fields are not enough

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.put(new A());
v2.put(new B());
x = v1.get();
A a = (A) x;  /* how to proof cast is ok? */
```

we need **context information!**

Main Problem

Accuracy of context information

- Only type information not sufficient
- Complete call chain causes exponential explosion

A useful solution must lie somewhere in between

- must trace types accurately enough
- must run in reasonable time and space

Object-Context

For values

- type (`new X()`)
- allocation site in source (**X.java:235**)
- object-context: type of **this**

Context information

In calls

- Java **method**
- value sets of all **arguments**
- **thread** executing method
- current set of **locks** held
- etc.

Result

Detailed information on values

v1.get() --> **A** (Test.java: line 14)

v2.get() --> **B** (Test.java: line 15)

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;


  int value = s.reading();

  ...
}
...
```

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;


  int value = s.reading();

  ...
}
...
```

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException

ClassCastException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException

ClassCastException

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException

ClassCastException

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException

ClassCastException

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException ✔

ClassCastException

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException

values(MyDevice.sensor)
contains only MySensor

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException

values(MyDevice.sensor)
contains only MySensor

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

values(MyDevice.sensor)
contains only MySensor

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException ✔

Runtime Errors

```
...
if (device instanceof MyDevice) {

  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

NullPointerException ✔

Deadlock detection

```
...
new Thread() {
  public void run() {
    synchronized (A) {
      synchronized (B) {
      }
    }
  }
}.start();
new Thread() {
  public void run() {
    synchronized (B) {
      synchronized (A) {
      }
    }
  }
}.start();
...
```

Deadlock detection

```
...
new Thread() {
  public void run() {
    synchronized (A) {
      synchronized (B) {
      }
    }
  }
}.start();
new Thread() {
  public void run() {
    synchronized (B) {
      synchronized (A) {
      }
    }
  }
}.start();
...
```

Lock Order:

Thread 1: A → B

Deadlock detection

```
...
new Thread() {
  public void run() {
    synchronized (A) {
      synchronized (B) {
      }
    }
  }
}.start();
new Thread() {
  public void run() {
    synchronized (B) {
      synchronized (A) {
      }
    }
  }
}.start();
...
```

Lock Order:

Thread 1: A → B

Thread 2: B → A

Deadlock detection

```
...
new Thread() {
  public void run() {
    synchronized (A) {
      synchronized (B) {
      }
    }
  }
}.start();
new Thread() {
  public void run() {
    synchronized (B) {
      synchronized (A) {
      }
    }
  }
}.start();
...
```

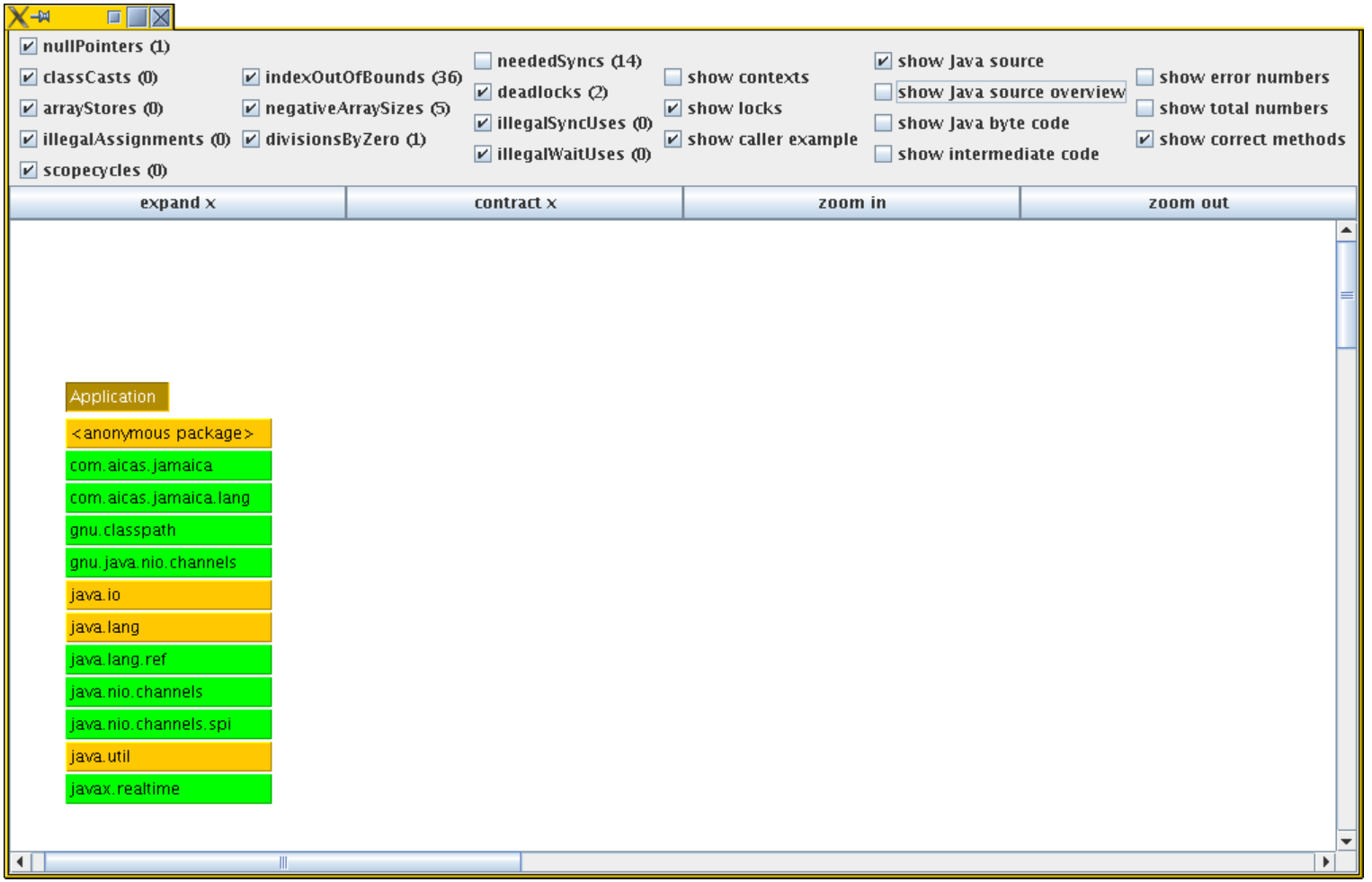
Lock Order:

Thread 1: A → B

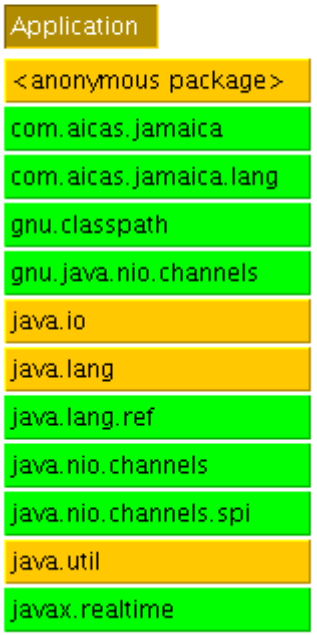
potential deadlock!

Thread 2: B → A

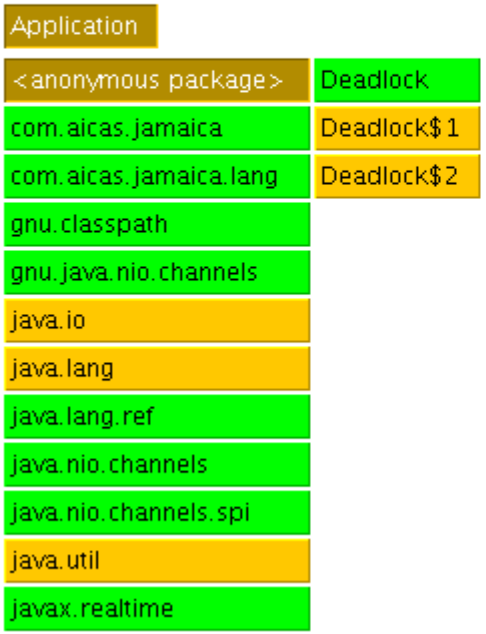
Graphical Display of Results



Graphical Display of Results

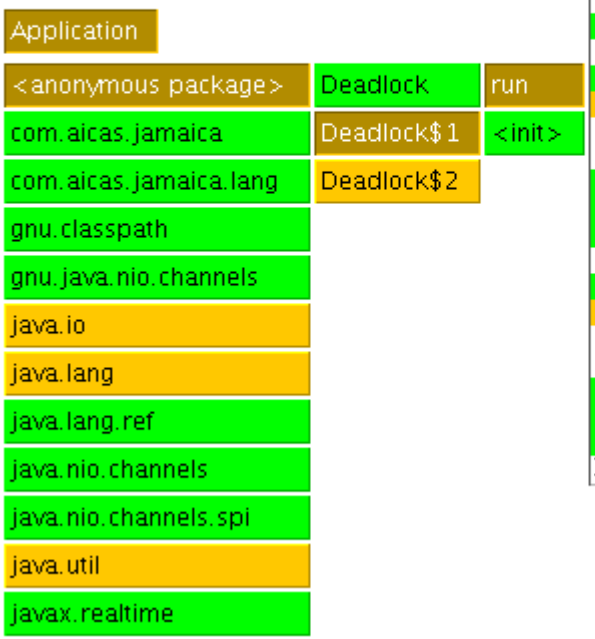


Graphical Display of Results



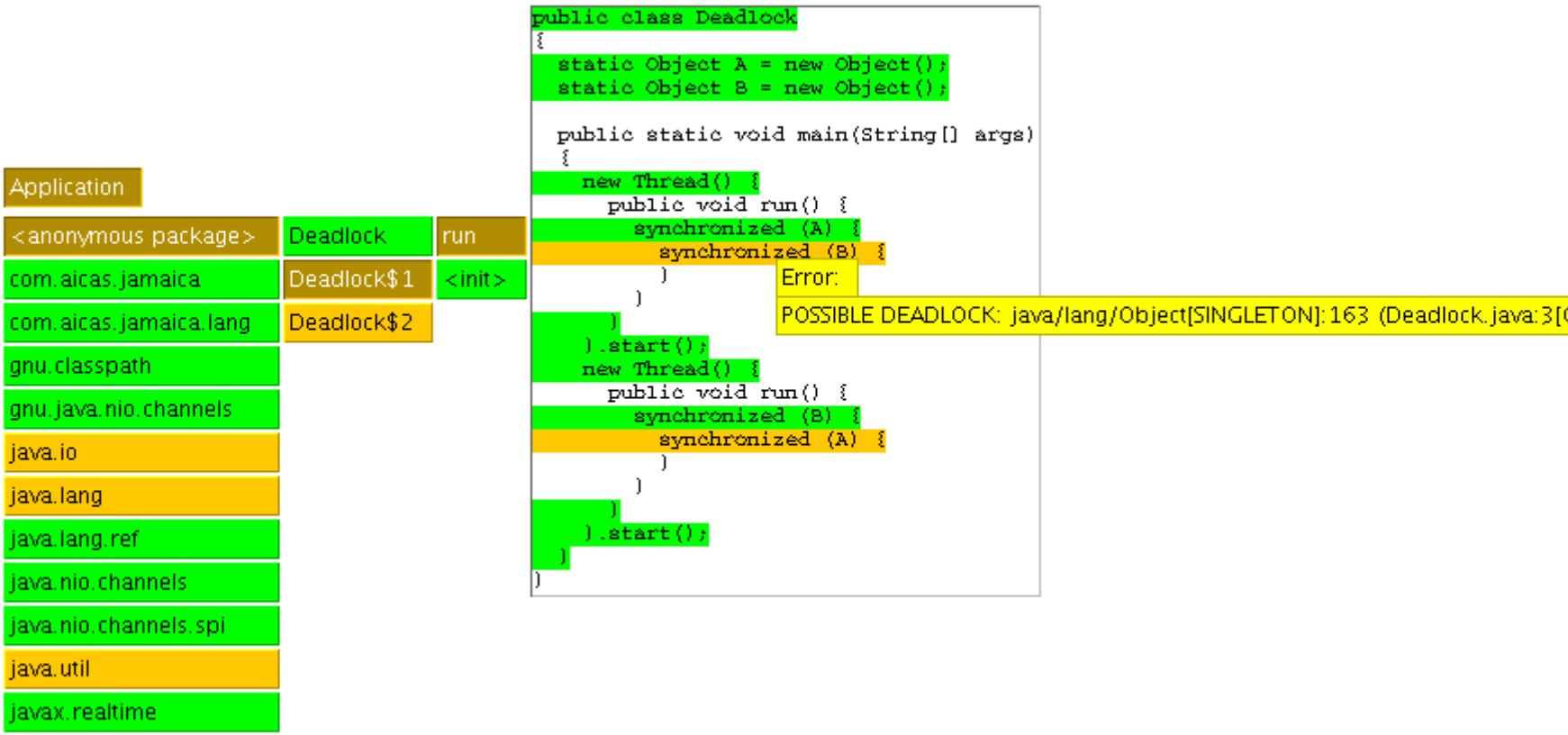
Graphical Display of Results

Graphical Display of Results



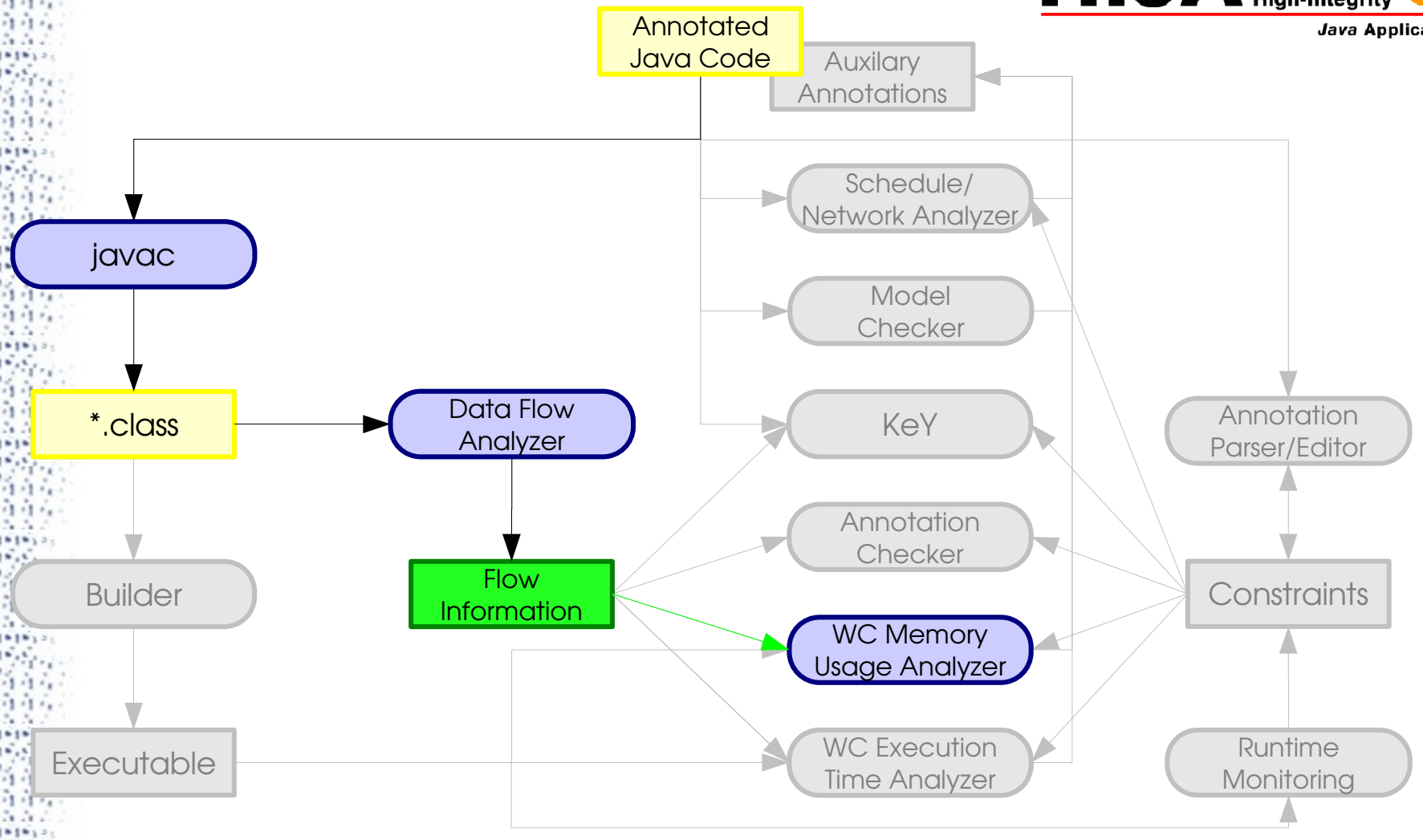

```
public class Deadlock
{
  static Object A = new Object();
  static Object B = new Object();

  public static void main(String[] args)
  {
    new Thread() {
      public void run() {
        synchronized (A) {
          synchronized (B) {
          }
        }
      }
    }.start();
    new Thread() {
      public void run() {
        synchronized (B) {
          synchronized (A) {
          }
        }
      }
    }.start();
  }
}
```

Graphical Display of Results



WC Memory Usage

HIJA High-Integrity Java Application



Memory Analysis

HIJA High-Integrity Java Application

Heap Use Analysis

- Traverse call graph for each thread
- Sum up all allocations

Memory Analysis

HIJA High-Integrity Java Application

Heap Use Analysis

- Traverse call graph for each thread
- Sum up all allocations

```
> jamaica test -dfa
...
HEAP USE:  39680 FOR THREAD: java/lang/FinalizerThread
HEAP USE:      0 FOR THREAD: test$1: (test.java:6)
HEAP USE:      0 FOR THREAD: test$2: (test.java:14)
HEAP USE: 851200 FOR THREAD: INITIAL THREAD
...
DFA DONE: 6431ms.
```

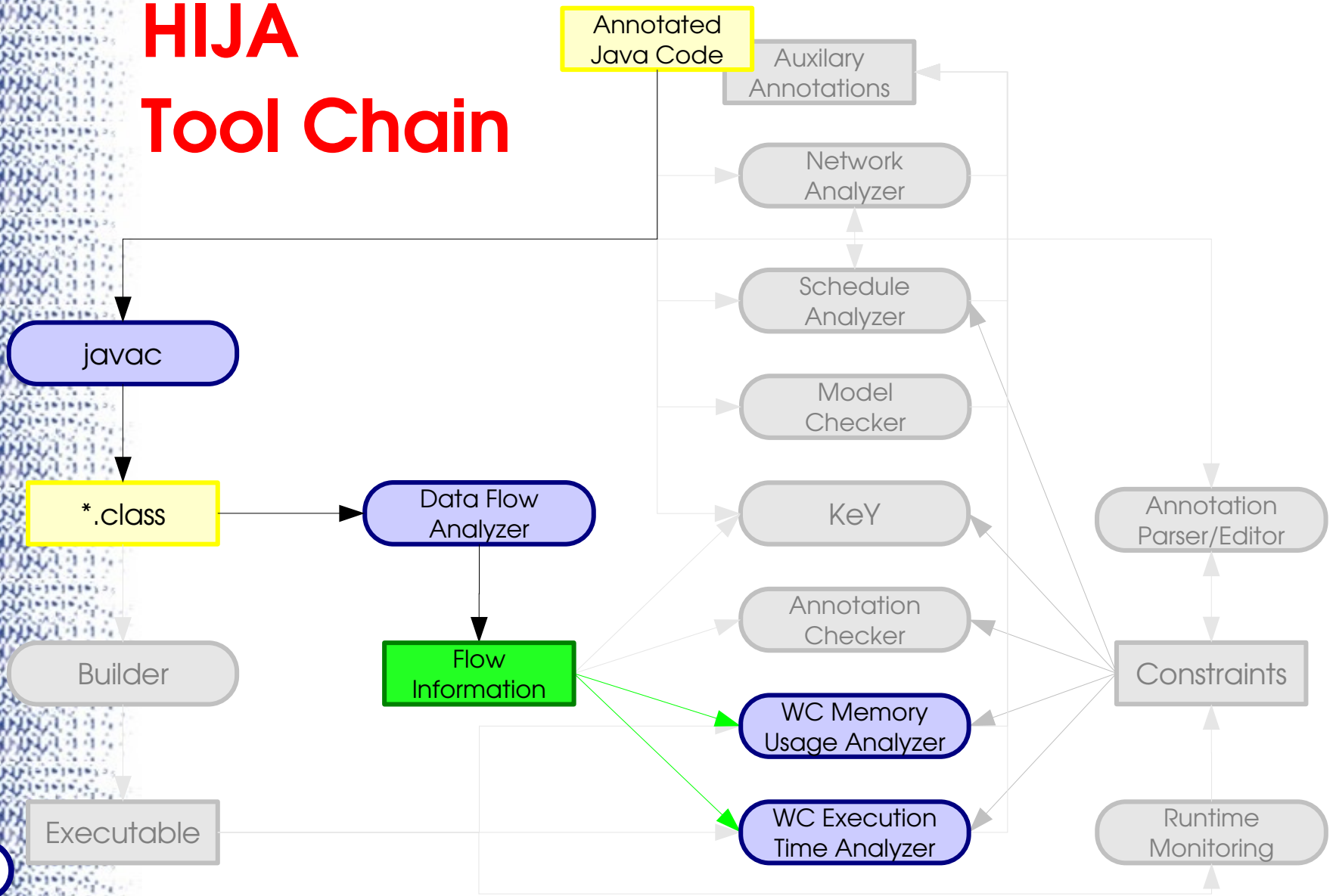
Analysis Limitations:

No information on

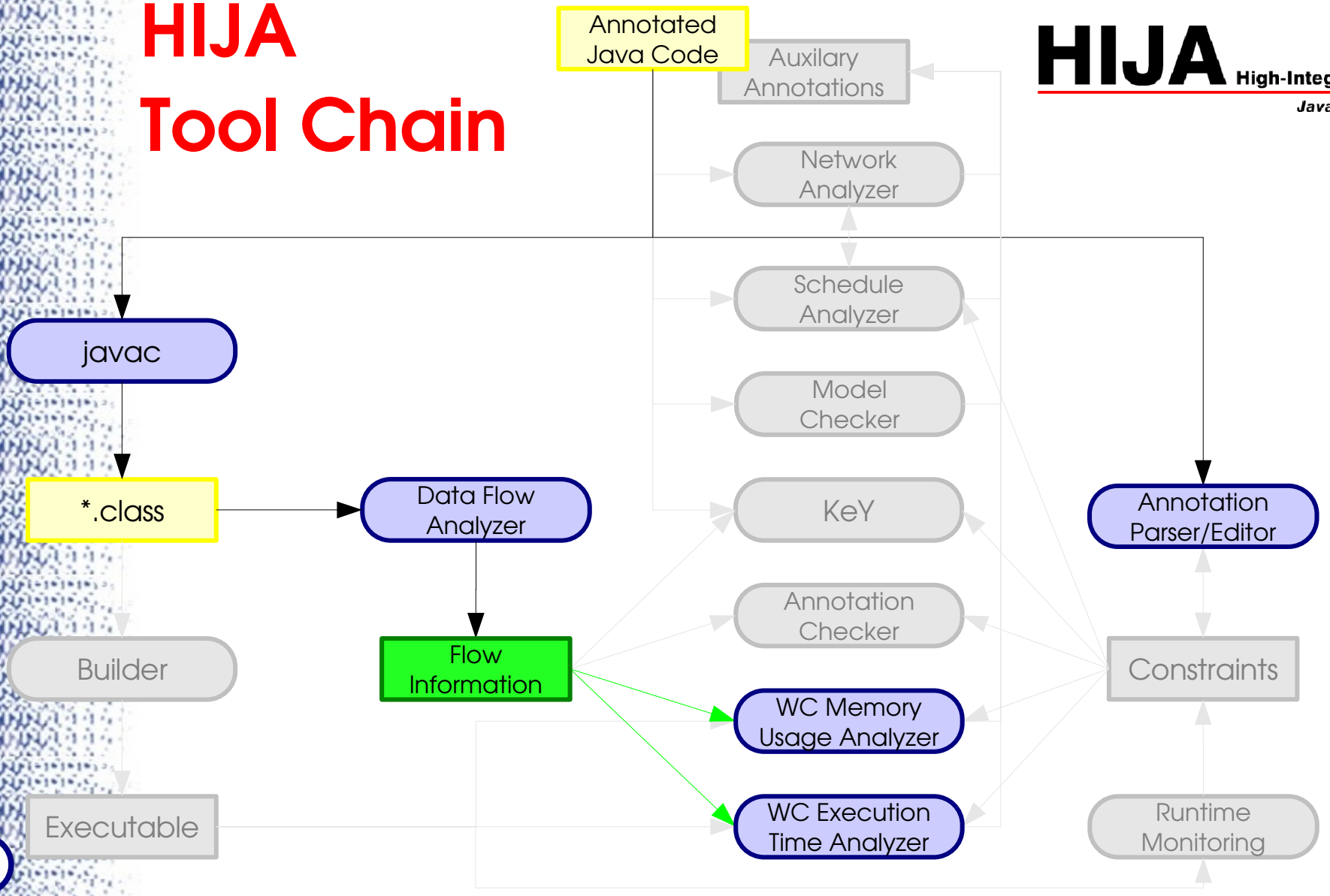
- loop counts
- recursion depths
- array lengths

We need annotations to provide this information!

HIJA Tool Chain

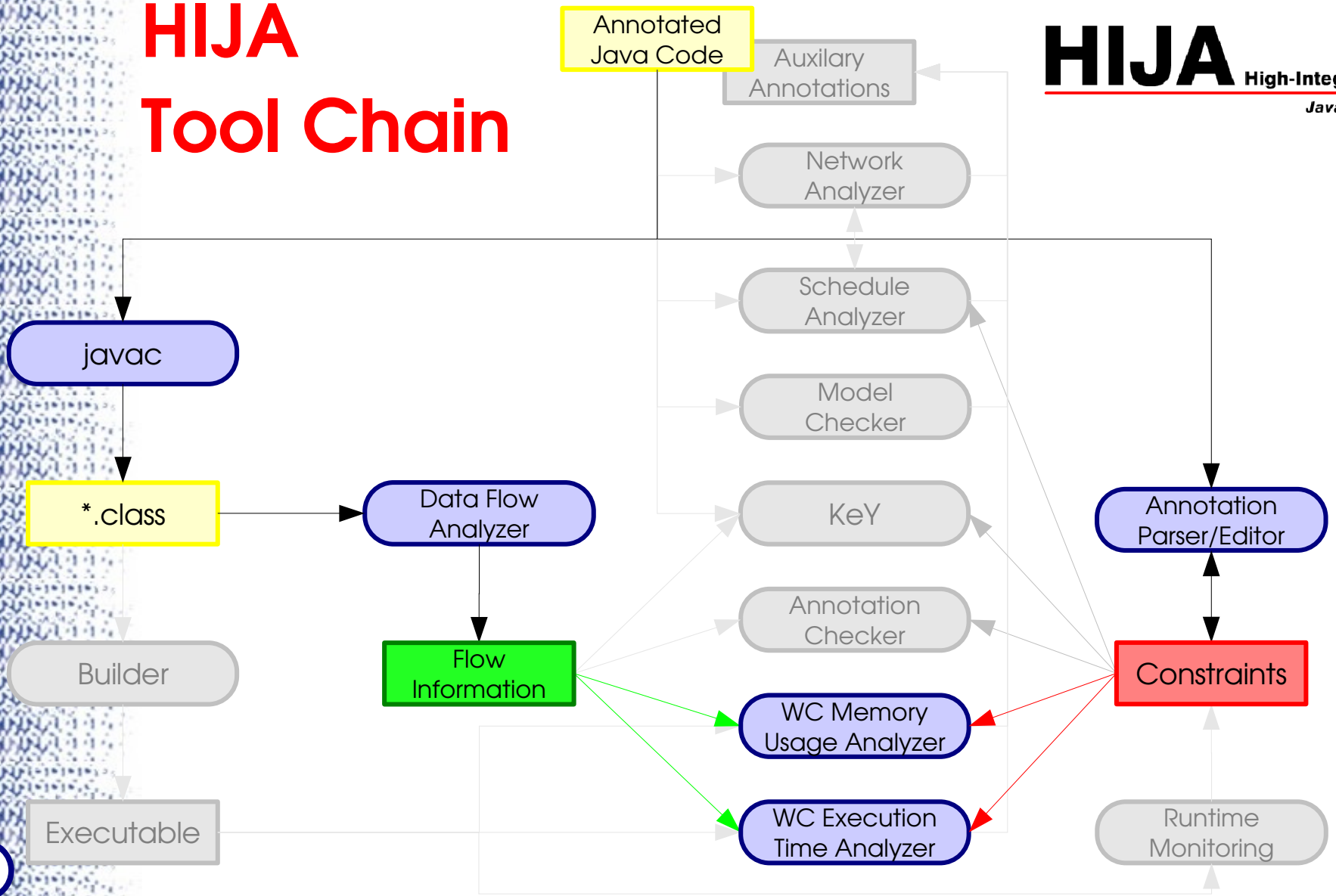


HIJA Tool Chain



HIJA Tool Chain

HIJA High-Integrity Java Application



Current Status:

HIJA High-Integrity Java Application

Industrialisation Phase:

- Non-graphical Prototype tested extensively by users (Thales Avionics, Fiat, Telecom Italia)
- Very positive feedback, but also long list of requests for improvements
- Graphical version available
- Suggested improvements are being implemented

Conclusions

DFA provides a means to prove the absence of important runtime errors and to detect program errors.

The use of this analysis is fully automatic. A very fast learning curve ensures early successes.

However, analysis for large applications using dynamic class loading, reflection, native code, etc. is limited.

DFA Performance

HIJA High-Integrity Java Application

Analysis times for SPECjvm98 benchmark tests

Test	time/sec	Types with reduced acc.
check	4,9	0
compress	12	0
jess	**62,7**	**3**
raytrace	50	1
db	11,2	0
javac	710,5	58
mpegaudio	15	0
mtrt	49,1	1
jack	25,1	1
checkit	14,8	0

DFA Performance

HIJA High-Integrity Java Application

Analysis times for SPECjvm98 benchmark tests

Test	NullPointer			Illegal Casts			ArrayStore			Deadlocks		
	pot. error	analysed	ratio	p.e.	an.	ratio	p.e.	an.	ratio	p.e.	an.	ratio
check	10	1953	1%	3	49	6%	0	45	0%	0	48	0%
compress	33	1813	2%	4	54	7%	1	43	2%	5	43	12%
jess	552	5265	10%	16	120	13%	1	105	1%	5	60	8%
raytrace	452	3398	13%	5	58	9%	2	57	4%	20	60	33%
db	102	2370	4%	5	73	7%	1	39	3%	5	56	9%
javac	1726	9884	17%	213	360	59%	18	457	4%	17	87	20%
mpegaudio	112	9605	1%	9	60	15%	1	944	0%	5	42	12%
mtrt	452	3400	13%	5	58	9%	2	59	3%	20	60	33%
jack	286	5103	6%	8	141	6%	1	99	1%	5	45	11%
hello	0	1012	0%	0	41	0%	0	21	0%	0	35	0%

DFA Performance

HIJA High-Integrity Java Application

Analysis times for SPECjvm98 benchmark tests

Test	NullPointer			Illegal Casts			ArrayStore			Deadlocks		
	pot. error	analysed	ratio	p.e.	an.	ratio	p.e.	an.	ratio	p.e.	an.	ratio
check	10	1953	1%	3	49	6%	0	45	0%	0	48	0%
compress	33	1813	2%	4	54	7%	1	43	2%	5	43	12%
jess	**552**	**5265**	**10%**	**16**	**120**	**13%**	**1**	**105**	**1%**	**5**	**60**	**8%**
raytrace	452	3398	13%	5	58	9%	2	57	4%	20	60	33%
db	102	2370	4%	5	73	7%	1	39	3%	5	56	9%
javac	1726	9884	17%	213	360	59%	18	457	4%	17	87	20%
mpegaudio	112	9605	1%	9	60	15%	1	944	0%	5	42	12%
mtrt	452	3400	13%	5	58	9%	2	59	3%	20	60	33%
jack	286	5103	6%	8	141	6%	1	99	1%	5	45	11%
hello	0	1012	0%	0	41	0%	0	21	0%	0	35	0%

Applying DFA to RTSJ code

Detecting Scope Cycles

- MemoryArea is recorded on every call to *enter()*
- Nesting order tree of memory areas is recorded
- A potential error is flagged if nesting order not cycle-free

Applying DFA to RTSJ code

Detecting Scope Cycles

- MemoryArea is recorded on every call to *enter()*
- Nesting order tree of memory areas is recorded
- A potential error is flagged if nesting order not cycle-free

Results

- all scope cycles are detected
- false positives if different nesting is used at different points in time

Applying DFA to RTSJ code

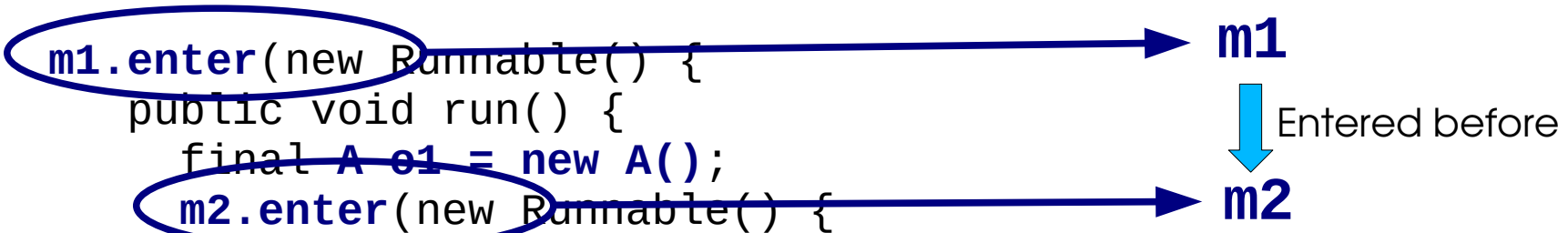
Assignment Checks

```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
   public void run() {
     final A o1 = new A();
     m2.enter(new Runnable() {
       public void run() {
         A o2 = new A();
         o2.f = o1;
         o1.g = o2; // illegal assignment!
       }
     });
   }
});
```

Applying DFA to RTSJ code

Assignment Checks

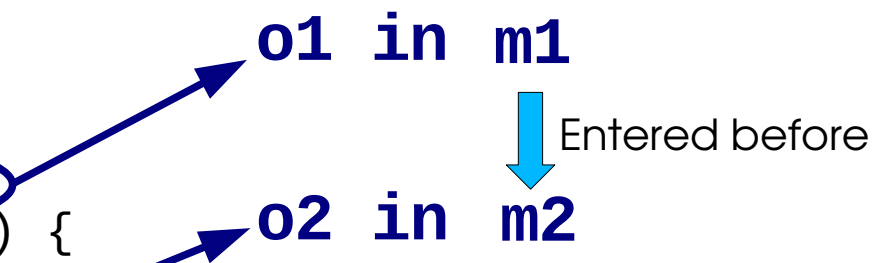
```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
    public void run() {
      final A o1 = new A();
      m2.enter(new Runnable() {
        public void run() {
          A o2 = new A();
          o2.f = o1;
          o1.g = o2; // illegal assignment!
        }
      });
    }
});
```



Applying DFA to RTSJ code

Assignment Checks

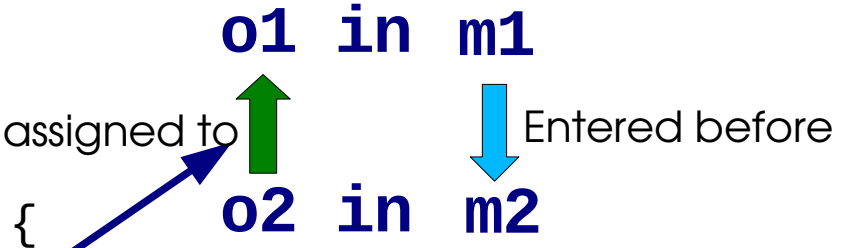
```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
    public void run() {
      final A o1 = new A();
      m2.enter(new Runnable() {
        public void run() {
          A o2 = new A();
          o2.f = o1;
          o1.g = o2; // illegal assignment!
        }
      });
    }
});
```



Applying DFA to RTSJ code

Assignment Checks

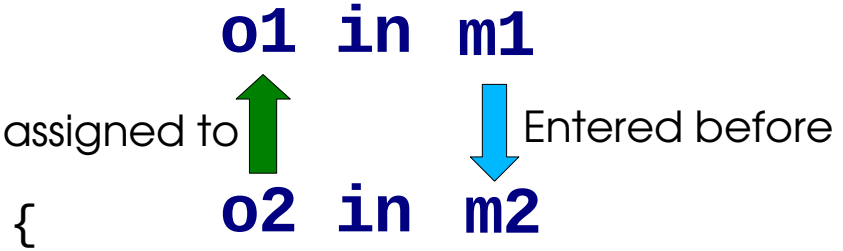
```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
   public void run() {
     final A o1 = new A();
     m2.enter(new Runnable() {
       public void run() {
         A o2 = new A();
         o2.f = o1;
         o1.g = o2; // illegal assignment!
       }
     });
   }
});
```



Applying DFA to RTSJ code

Assignment Checks

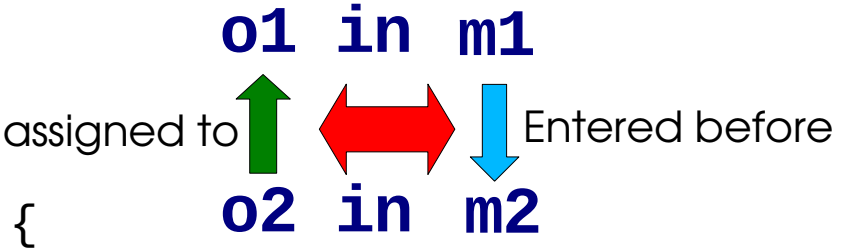
```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
   public void run() {
     final A o1 = new A();
     m2.enter(new Runnable() {
       public void run() {
         A o2 = new A();
         o2.f = o1;
         o1.g = o2; // illegal assignment!
       }
     });
   }
});
```



Applying DFA to RTSJ code

Assignment Checks

```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
   public void run() {
     final A o1 = new A();
     m2.enter(new Runnable() {
       public void run() {
         A o2 = new A();
         o2.f = o1;
         o1.g = o2; // illegal assignment!
       }
     });
   }
});
```



Applying DFA to RTSJ code

Assignment Checks Output

```
POTENTIALLY ILLEGAL ASSIGNEMT:
  to field A.g (test_assign.java:64)
      class: A:114:5 (test_assign.java:35)
        {IN: LTMemory:3 (test_assign.java:23)}
  <== class: A:190:5 (test_assign.java:39)
        {IN: LTMemory:4 (test_assign.java:24)}
```

Applying DFA to RTSJ code

Assignment Checks Output

```
POTENTIALLY ILLEGAL ASSIGNEMT:
  to field A.g (test_assign.java:64)
      class: A:114:5 (test_assign.java:35)
        {IN: LTMemory:3 (test_assign.java:23)}
  <== class: A:190:5 (test_assign.java:39)
        {IN: LTMemory:4 (test_assign.java:24)}
```

Applying DFA to RTSJ code

Assignment Checks Output

```
POTENTIALLY ILLEGAL ASSIGNEMT:
  to field A.g (test_assign.java:64)
      class: A:114:5 (test_assign.java:35)
        {IN: LTMemory:3 (test_assign.java:23)}
  <== class: A:190:5 (test_assign.java:39)
        {IN: LTMemory:4 (test_assign.java:24)}
```

Applying DFA to RTSJ code

Assignment Checks Output

```
POTENTIALLY ILLEGAL ASSIGNEMT:
  to field A.g (test_assign.java:64)
      class: A:114:5 (test_assign.java:35)
        {IN: LTMemory:3 (test_assign.java:23)}
  <== class: A:190:5 (test_assign.java:39)
        {IN: LTMemory:4 (test_assign.java:24)}
```

Analysis Results

HIJA High-Integrity Java Application

```
> jamaica -dfa test
NEEDEDSYNCS                :  29 (  29 locations out of    78)
DEADLOCKS                  : 101 (   9 locations out of    79)
SCOPE CYCLES               :   0 (   0 locations out of     0)
ILLEGAL ASSIGNMENTS        :   0 (   0 locations out of   764)
CLASSCAST EXCEPTIONS       : 128 (  17 locations out of    90)
ARRAY STORE EXCEPTIONS     :   3 (   1 locations out of   310)
NULL POINTER EXCEPTIONS: 503 (139 locations out of 11041)
 + test.dfa_results.summary
 + test.dfa_results
```

# of Source Code Positions with this problem

Conclusion

The RTSJ is the established standard for realtime Java.

The HIJA project extends the domain of Java by safety- and business-critical applications.

Automatic tools for correctness analysis reduces the manual validation & verification work significantly.

Project homepage: **www.hija.info**.

JamaicaVM

Who is aicas GmbH?

- **Founded** in March 2001 in Karlsruhe, Germany
- **Products:** Realtime Java implementations

- **Today:** 20 employees, offices in Karlsruhe (D), Berlin (D) and New Haven/CT (USA)

Business-Critical Profile

HIJA High-Integrity Java Application

Relaxations compared to safety-critical profile:

- dynamic features for non-critical tasks
- Strict Partitioning for critical tasks:
  - Memory: Local memory area for each task
  - CPU utilization
- Tools to proof correctness of critical tasks

HIJA Tools:

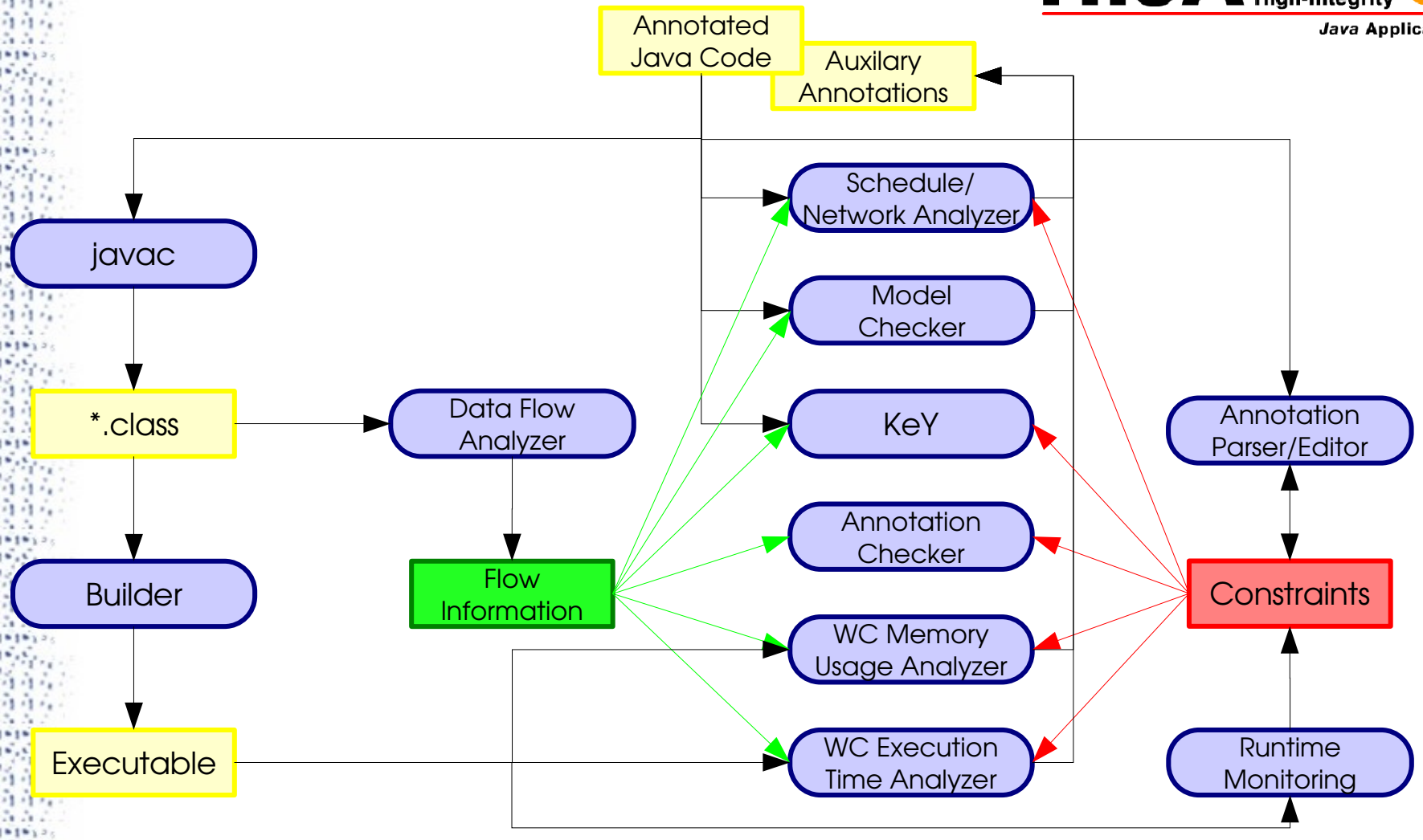
HIJA High-Integrity Java Application

Tools for correctness proof of application:

- DFA analysis to proof absence of runtime errors
- Formal verification using JML annotations (KeY)
- Model-Checker based tools to verify distributed system and multi-threading

DFA in HIJA Tool Chain

HIJA High-Integrity Java Application



Safety-Critical Profile

HIJA High-Integrity Java Application

Guiding Principles:

- Conservative approach
  - no concepts that go too far for SC community
- Do not fragment the market:
  - Base specification on standard Java and RTSJ
  - SC-Java programs should run on any RTSJ JVM
- Use of annotations for off-line checking, not for semantic changes

The HIJA Safety-Critical Java Profile

... building certifiable applications with Java technology

Dr. Fridtjof Siebert
Director of Development, aicas GmbH
Embedded World Conference
Nürnberg, 2006

HIJA-Project: High-Integrity Java

Project overview:

- 6th FP EC-funded project
- started June '04
- duration: 27 months, until Aug'06
- Project coordinator: The Open Group

HIJA-Project

HIJA High-Integrity Java Application

Project partners:

THALES

aicas realtime

Aonix

TELECOM ITALIA

UNIVERSITÄT KARLSRUHE (TH)

bellstream

TRIALOG

FZI

CENTRO RICERCHE FIAT

THE Open GROUP

THE UNIVERSITY of York

Project Overview

HIJA High-Integrity Java Application

Goals

- Identify suitable profiles
- Tools for functional and non-functional analysis
- Reference implementations of profiles

Work based on results of earlier projects:

HIDOORS, AJACS, Expresso, KeY, etc.

JamaicaVM

AERO VM

Project Overview

HIJA High-Integrity Java Application

Three main profiles are defined:

All are subsetting the Java language, standard APIs and Realtime Specification for Java (RTSJ)

- Safety-Critical profile
  - certification up to the highest criticality level (DO178B-A).
- Business-Critical profile
- Flexible Dynamic profile (OSGi, etc.)

Safety-Critical Profile

HIJA High-Integrity Java Application

Severe restrictions on Java and RTSJ:

- Clear distinction: initialization phase, mission phase
- No dynamic loading, thread creation, etc. during mission phase
- No GC
- Strict Partitioning:
  - Memory: Local memory area for each task
  - CPU utilization: RTSJ cost monitoring
- Use of annotations to document correctness of classes

HIJA SC-Java Profile

HIJA High-Integrity Java Application

Concurrency Model:

- Fixed-priority preemptive scheduling, 28 priorities
- FIFO within priority

Schedulable Objects:

- Periodic Threads
- BoundAsyncEventHandler
- all threads and handlers created in initialization phase.

Synchronization:

- Via Java synchronized methods
- Priority Ceiling Emulation Protocol

HIJA SC-Java Profile

Memory Management Phases:

- Initialization phase:
  allocation in Immortal Memory only

- Mission phase:
  allocation in Scoped Memory only
  - each scope local to one task
  - no sharing of objects in scopes between tasks
  - scope is cleared after each release

HIJA SC-Java Profile

HIJA High-Integrity Java Application

Absence of memory related runtime errors:

- Static analysis of worst-case stack
- Static analysis of worst-case heap
- Static analysis of to avoid AssignmentError (no storing of scoped references into objects with longer lifespan)

Use of annotations:

- To document scope-safety of library code

Reference Value Identification

Objects are identified by Allocation Point

- v1 is Vector(Test.java:12)
- v2 is Vector(Test.java:13)

and (recursively) by a list of allocation contexts (instance of „this“ at allocation site)

- v1.data is Object()(Vector.java:54)
  Vector(Test.java:12);
- v2.data is Object()(Vector.java:54)
  Vector(Test.java:13);

Granularity

Do not work on methods, but on Call Points

Call Points are identified by

- Set of Values of „this“ or first argument
- Vector(Test.java:13)

example: 2 call points for Vector.put

- Vector.put(Vector(Test.java:12),
  A(Test.java:14))
- Vector.put(Vector(Test.java:13),
  A(Test.java:15))

Granularity

example: 2 conditional values for data array

- Vector.data(x) is **A**(Test.java: line 14)
  if data is Object()(Vector.java:54)
  Vector(Test.java:**12**);
  Vector.data(x) is **B**(Test.java: line 15)
  if data is Object()(Vector.java:54)
  Vector(Test.java:**13**);

Memory Analysis

HIJA High-Integrity Java Application

Stack Use Analysis

- Traverse call graph for each thread
- Find maximum stack use in each branch

Memory Analysis

HIJA High-Integrity Java Application

Stack Use Analysis

- Traverse call graph for each thread
- Find maximum stack use in each branch

```
> jamaica test -dfa
...
STACK USE:  1264 FOR THREAD: java/lang/FinalizerThread
STACK USE:   104 FOR THREAD: test$1: (test.java:6)
STACK USE:   104 FOR THREAD: test$2: (test.java:14)
STACK USE:  1764 FOR THREAD: INITIAL THREAD
...
DFA DONE: 6431ms.
```

Applying DFA to RTSJ code

Assignment Checks

```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
    public void run() {
      final A o1 = new A();
      m2.enter(new Runnable() {
        public void run() {
          A o2 = new A();
          o2.f = o1;
          o1.g = o2; // illegal assignment!
        }
      });
    }
});
```

Applying DFA to RTSJ code

Assignment Checks

```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
   public void run() {
     final A o1 = new A();
     m2.enter(new Runnable() {
       public void run() {
         A o2 = new A();
         o2.f = o1;
         o1.g = o2; // illegal assignment!
       }
     });
   }
});
```

o1 in m1

Entered before

m2

Applying DFA to RTSJ code

Assignment Checks

```
MemoryArea m1 = new LTMemory(100000);
MemoryArea m2 = new LTMemory(100000);

m1.enter(new Runnable() {
   public void run() {
     final A o1 = new A();
     m2.enter(new Runnable() {
       public void run() {
         A o2 = new A();
         o2.f = o1;
         o1.g = o2; // illegal assignment!
       }
     });
   }
});
```