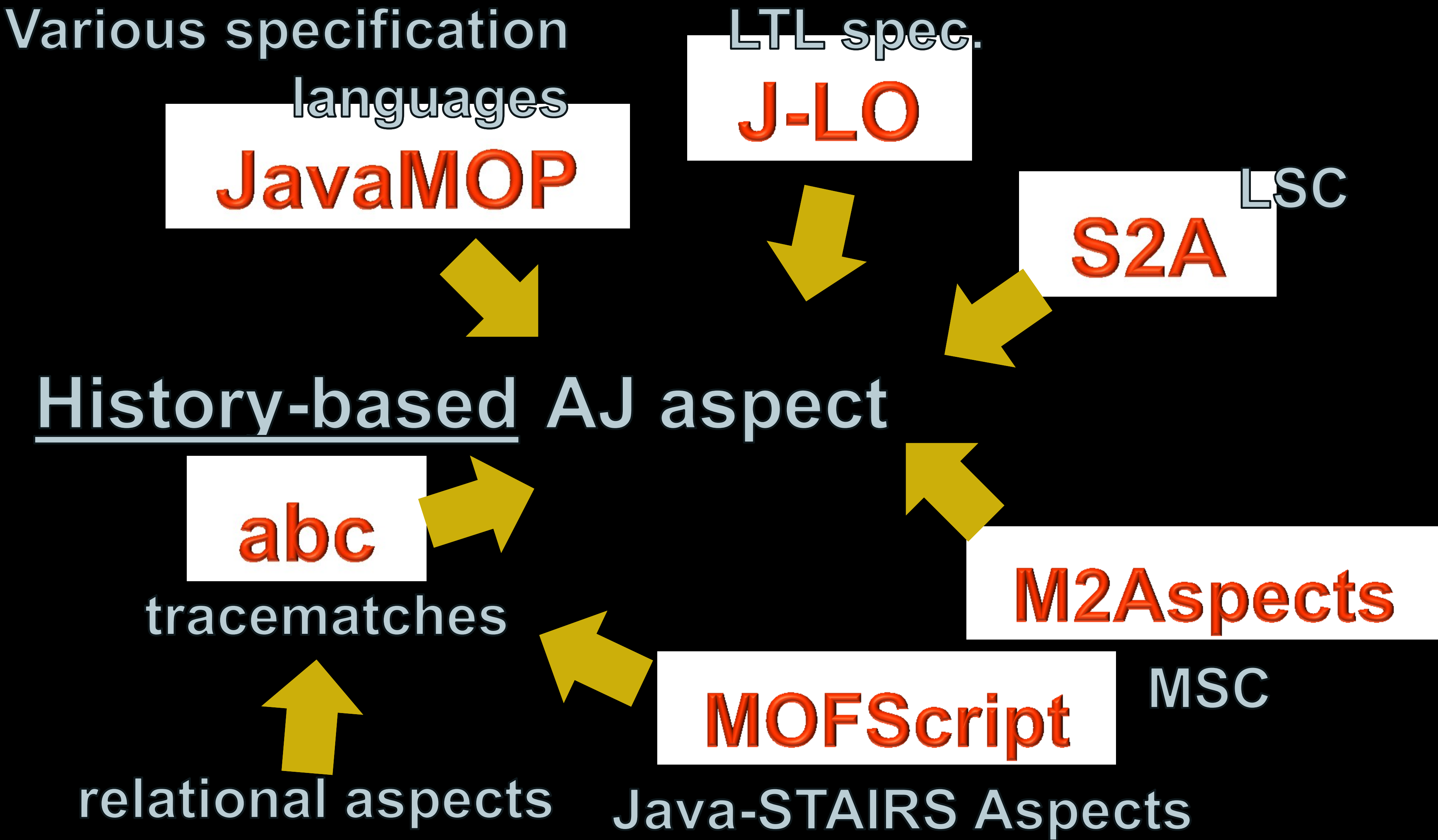
McGill University | Eric Bodden

University of Illinois at Urbana Champaign | Feng Chen
Grigore Roşu

Dependent Advice

A General Approach to Optimizing History-based Aspects

AspectJ as an intermediate language



Various specification languages

JavaMOP

LTL spec.

J-LO

LSC

S2A

History-based AJ aspect

abc

tracematches

M2Aspects

MSC

MOFScript

relational aspects

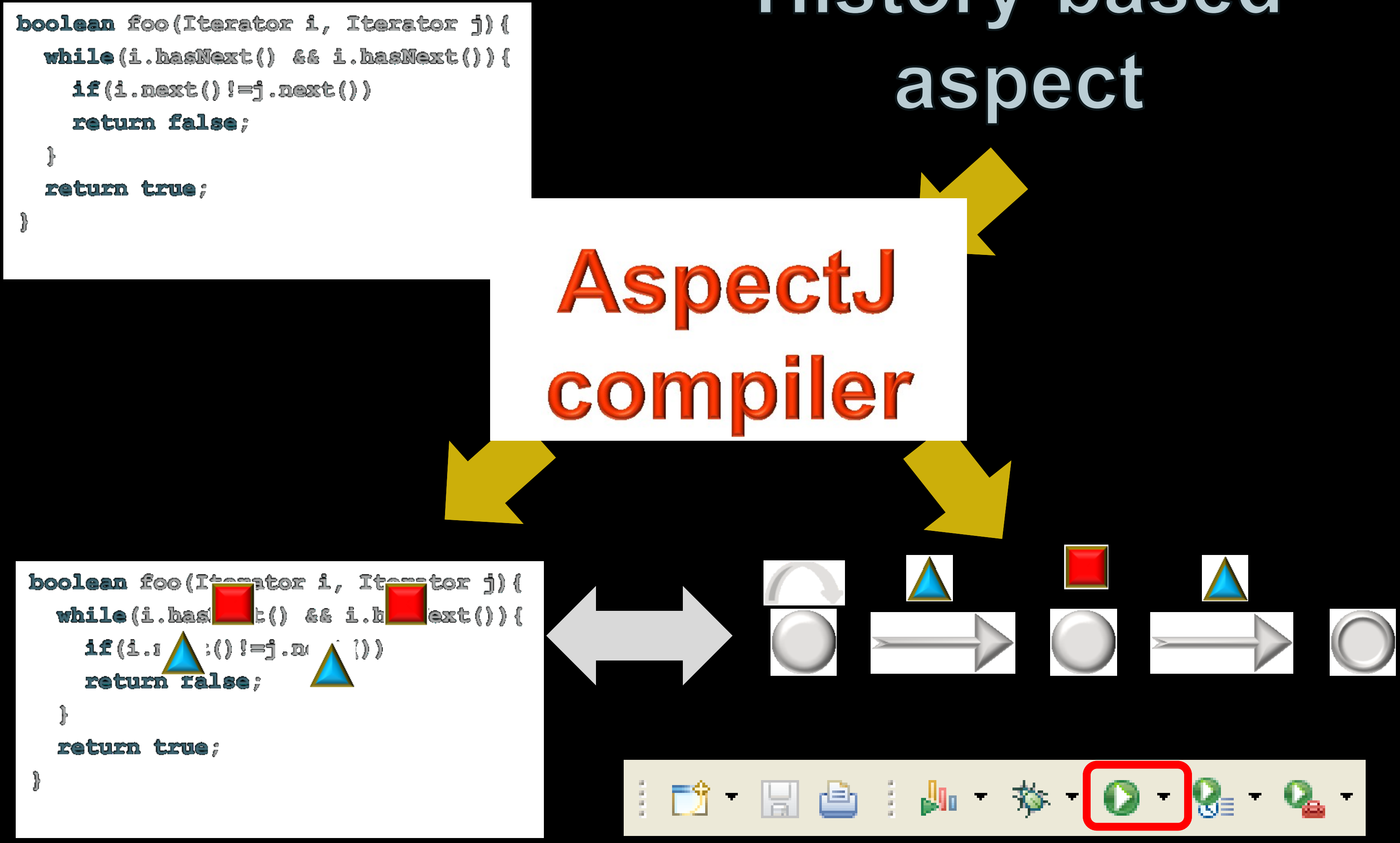
Java-STAIRS Aspects

Example concern

Do not write to a disconnected connection.

```
aspect ConnectionClosed {
   Set closed = new WeakIdentityHashSet();

   after /*disconn*/ (Connection c) returning:
      call(* Connection.disconnect()) && target(c) {
      closed.add(c);
   }

   after /*reconn*/ (Connection c) returning:
      call(* Connection.reconnect()) && target(c) {
      closed.remove(c);
   }

   before /*write*/ (Connection c) :
      call(* Connection.write (..)) && target(c) {
      if (closed.contains(c))
             error(c+″ is closed !″);
   }
}
```

History-based aspect



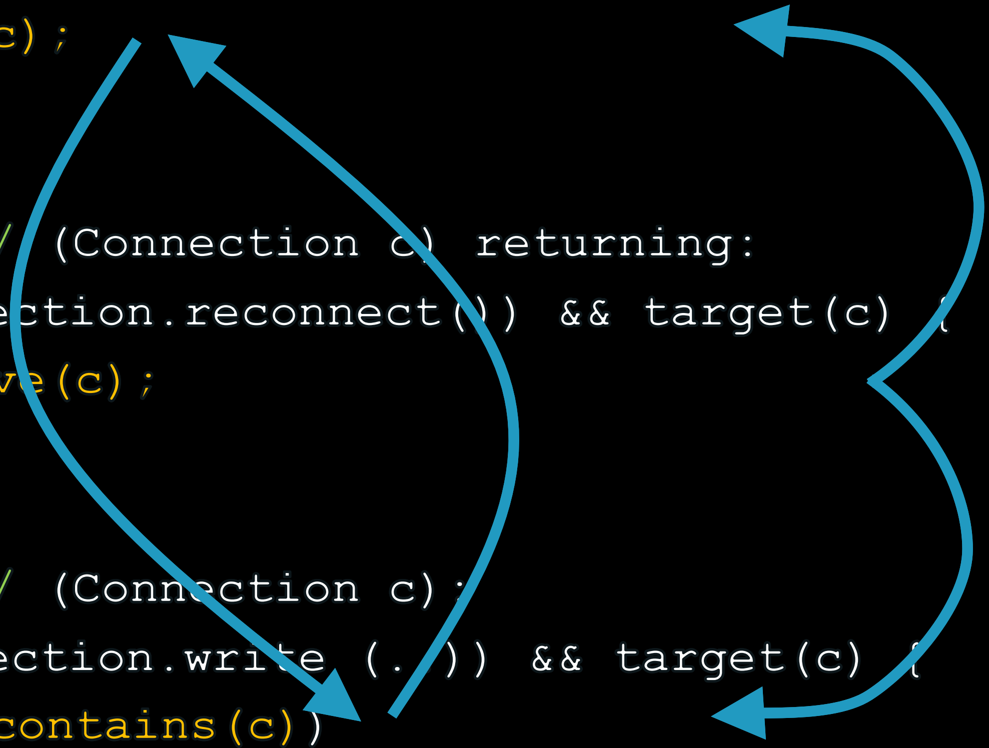
```
boolean foo(Iterator i, Iterator j){
  while(i.hasNext() && i.hasNext()){
    if(i.next()!=j.next())
    return false;
  }
  return true;
}
```

AspectJ compiler

```
boolean foo(Iterator i, Iterator j){
  while(i.has■t() && i.h■ext()){
    if(i.▲:()!=j.n▲())
    return false;
  }
  return true;
}
```

Problem:

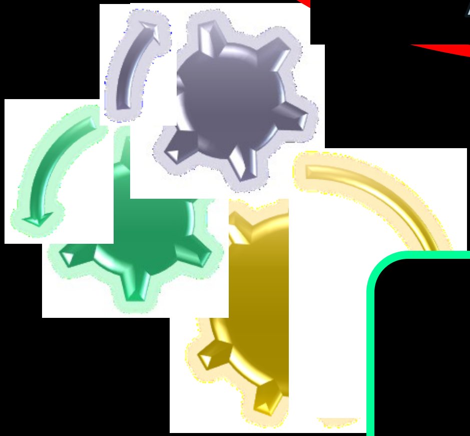
Potentially large runtime overhead



```
aspect ConnectionClosed {
   Set closed = new WeakIdentityHashSet();

   after /*disconn*/ (Connection c) returning:
       call(* Connection.disconnect()) && target(c) {
       closed.add(c);
   }

   after /*reconn*/ (Connection c) returning:
       call(* Connection.reconnect()) && target(c) {
       closed.remove(c);
   }

   before /*write*/ (Connection c) :
       call(* Connection.write (..)) && target(c) {
       if (closed.contains(c))
              error(c+″ is closed !″);
   }
}
```

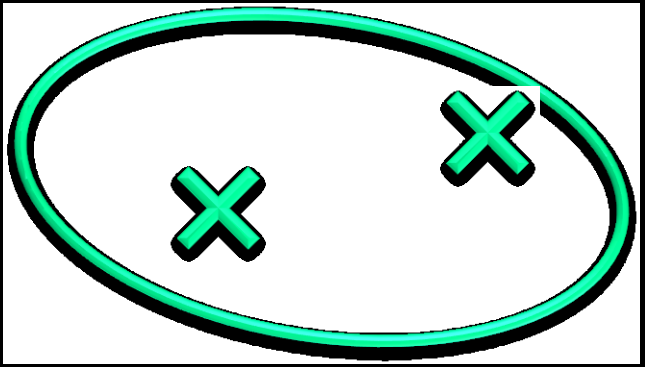
Now: general case

History-based
AJ aspects



Optimized
Runtime

Static
program analysis



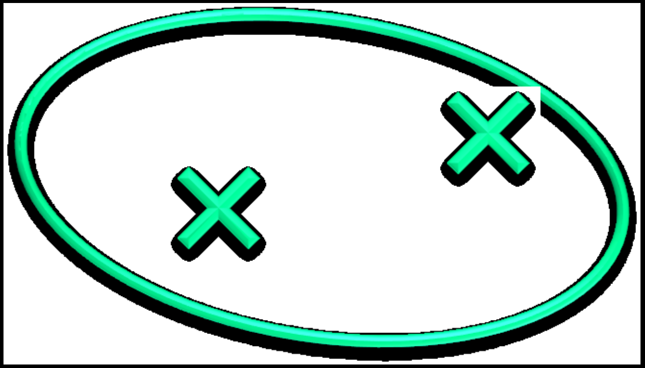
Analyzing history-based aspects?

```
/* Monitor aspect for HasNext+*/
import java.util.*;
import org.apache.commons.collections.map.*;
class HasNextMonitorPM {
    Vector monitorList = new Vector();
    synchronized public void hasNext(Iterator i) {
        HashSet monitorSet = new HashSet();
        Iterator it = monitorList.iterator();
        while (it.hasNext()){
            HasNextMonitor monitor = (HasNextMonitor)it.next();
            monitor.hasNext(i);
            if (monitorSet.contains(monitor) || monitor.failed())
            it.remove();
            else {
                monitorSet.add(monitor);
                if (monitor.suceeded()){
```

... about 10 more pages

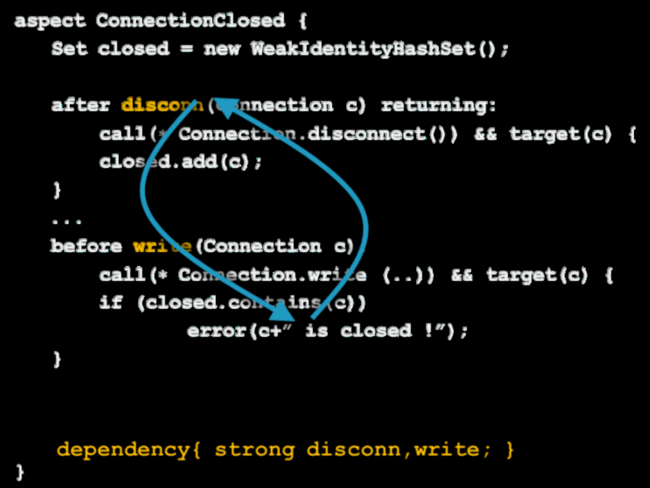
+Dependency annotations

Static program analysis



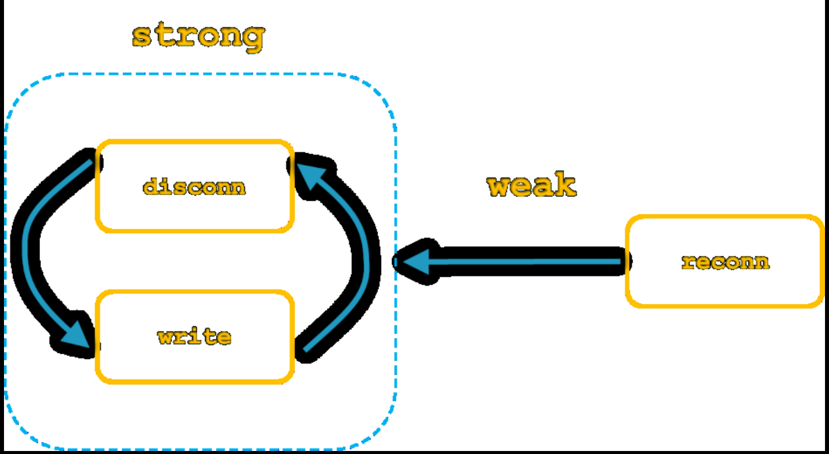
Contributions and Outline

Syntax of dep. advice

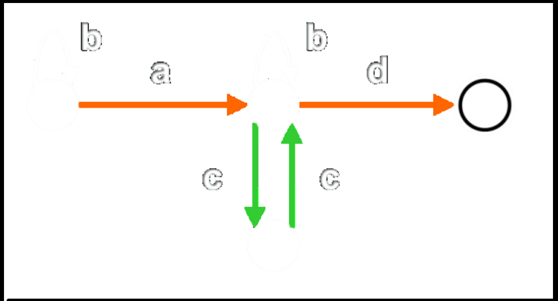


```
aspect ConnectionClosed {
   Set closed = new WeakIdentityHashSet();

   after disconn(Connection c) returning:
       call(* Connection.disconnect()) && target(c) {
       closed.add(c);
   }
   ...
   before write(Connection c)
       call(* Connection.write (..)) && target(c) {
       if (closed.contains(c))
              error(c+" is closed !");
   }


    dependency{ strong disconn,write; }
}
```

Semantics of dep. advice

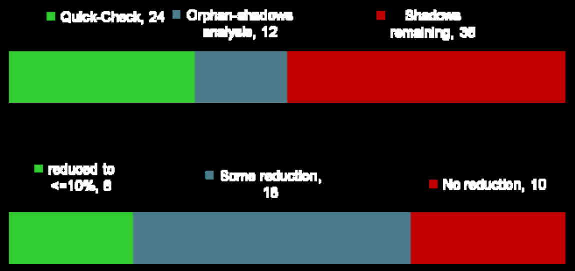


Generating dep. advice

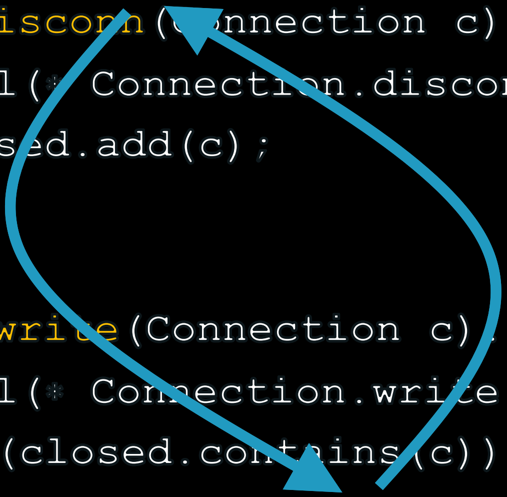


```
dependency{  strong a,d; weak c; }
dependency{  strong a,c,d; weak b; }
```

Experimental results



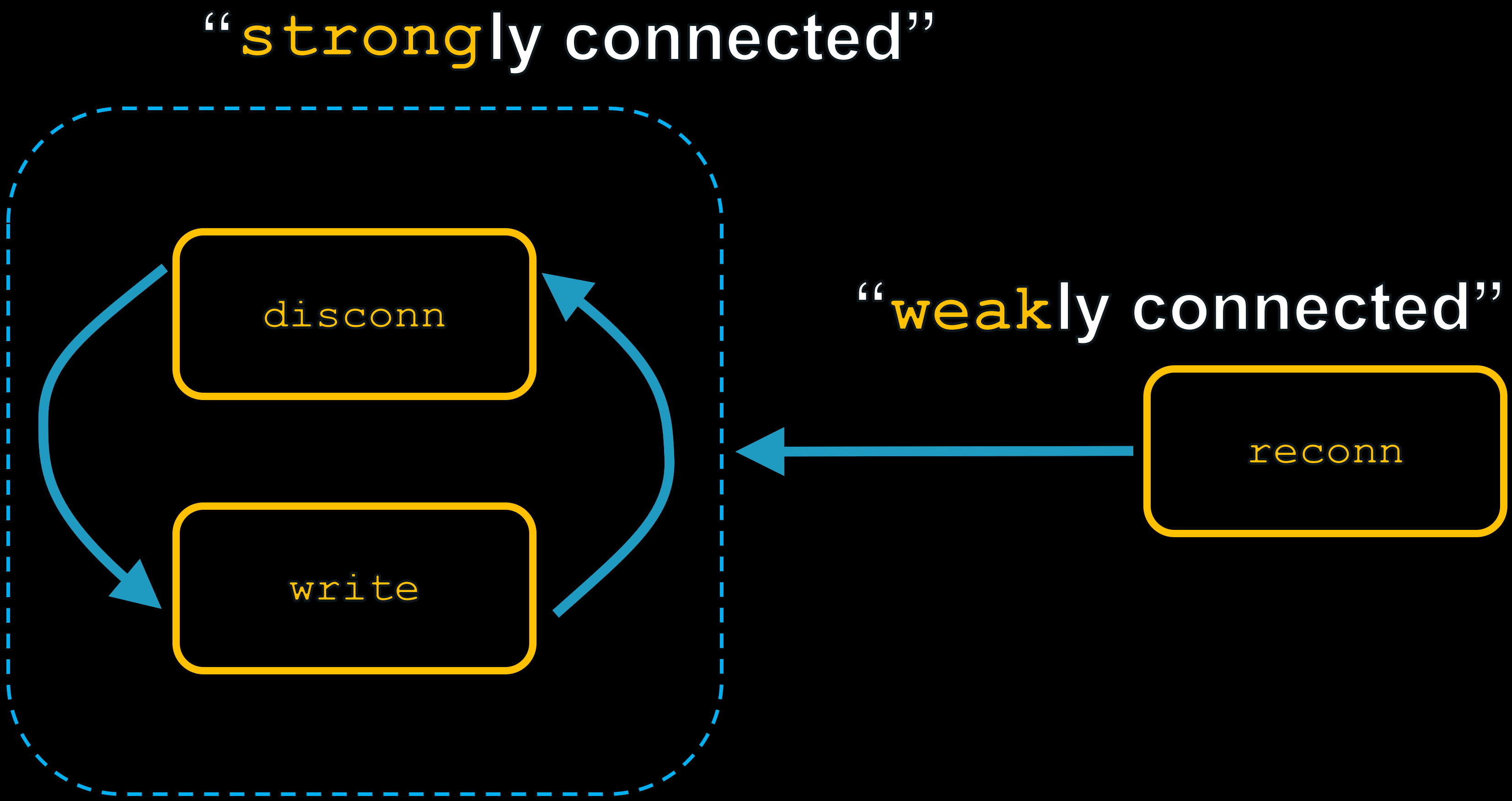
Dependent advice



```
aspect ConnectionClosed {
    Set closed = new WeakIdentityHashSet();

    after disconn(Connection c) returning:
        call(* Connection.disconnect()) && target(c) {
        closed.add(c);
    }
    ...
    before write(Connection c):
        call(* Connection.write (..)) && target(c) {
        if (closed.contains(c))
               error(c+" is closed !");
    }



    dependency{ strong disconn,write; }
}
```

Dependent advice

```
aspect ConnectionClosed {
   Set closed = new WeakIdentityHashSet();

   after disconn(Connection c) returning:
       call(* Connection.disconnect()) && target(c) {
       closed.add(c);
   }

   after reconn(Connection c) returning:
       call(* Connection.reconnect()) && target(c) {
       closed.remove(c);
   }

   before write(Connection c):
   ...
    dependency{ strong disconn,write; weak reconn; }

}
```



“strongly connected”

disconn

write

“weakly connected”

reconn

```
dependency{ strong disconn,write; weak reconn; }
```

Verbose syntax

```
dependency{
  strong disconn,write;
  weak reconn;
}
```

… is a shorth

```
after reconn(Connection c)
```

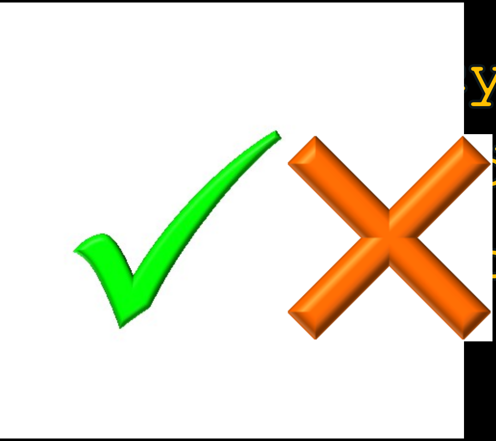
```
dependency{
  strong disconn(c),write(c);
  weak reconn(c);
}
```

When is a dependency fulfilled?

```
dependency{
   strong disconn(c),write(c);
   weak reconn(c);
}
```

Dependency is fulfilled for Connection `c` if both `disconn(c)` and `write(c)` do execute on `c` at some point in time.

Example



```
         y{
          disco✗(c),write(c);
          conn(c);
```

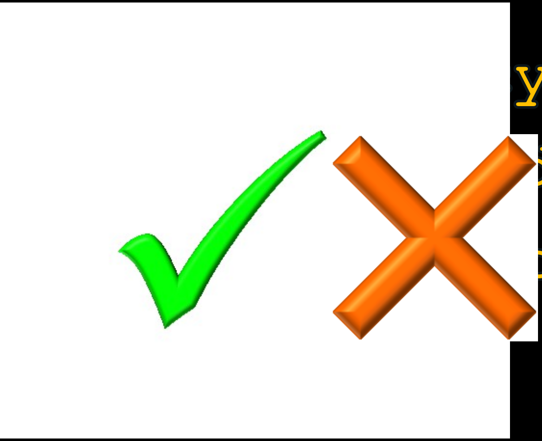
```
Connection c1 = new Connection();
Connection c2 = new Connection();
c1.disconnect();
c2.write("foo");
c1.reconnect();
c1.write("bar");
```

→ dependency fulfilled for c1,
but not fulfilled for c2

When does a Dep. Adv. execute?

Dependent advice *a* ***must*** execute at a joinpoint *j* on objects *o* if there exists a dependency *d* that references *a* and is fulfilled for objects *o*.

Example



```
y{
disconn(c),write(c);
conn(c);
```

```
Connection c1 = new Connection();
Connection c2 = new Connection();
```

✓ `c1.disconnect();`

? `c2.write(“foo”);`

✓ `c1.reconnect();`

✓ `c1.write(“bar”);`

→ `disconn`/`write`/`reconn` *will* execute on `c1`, do not have to execute (but *may*) on `c2`

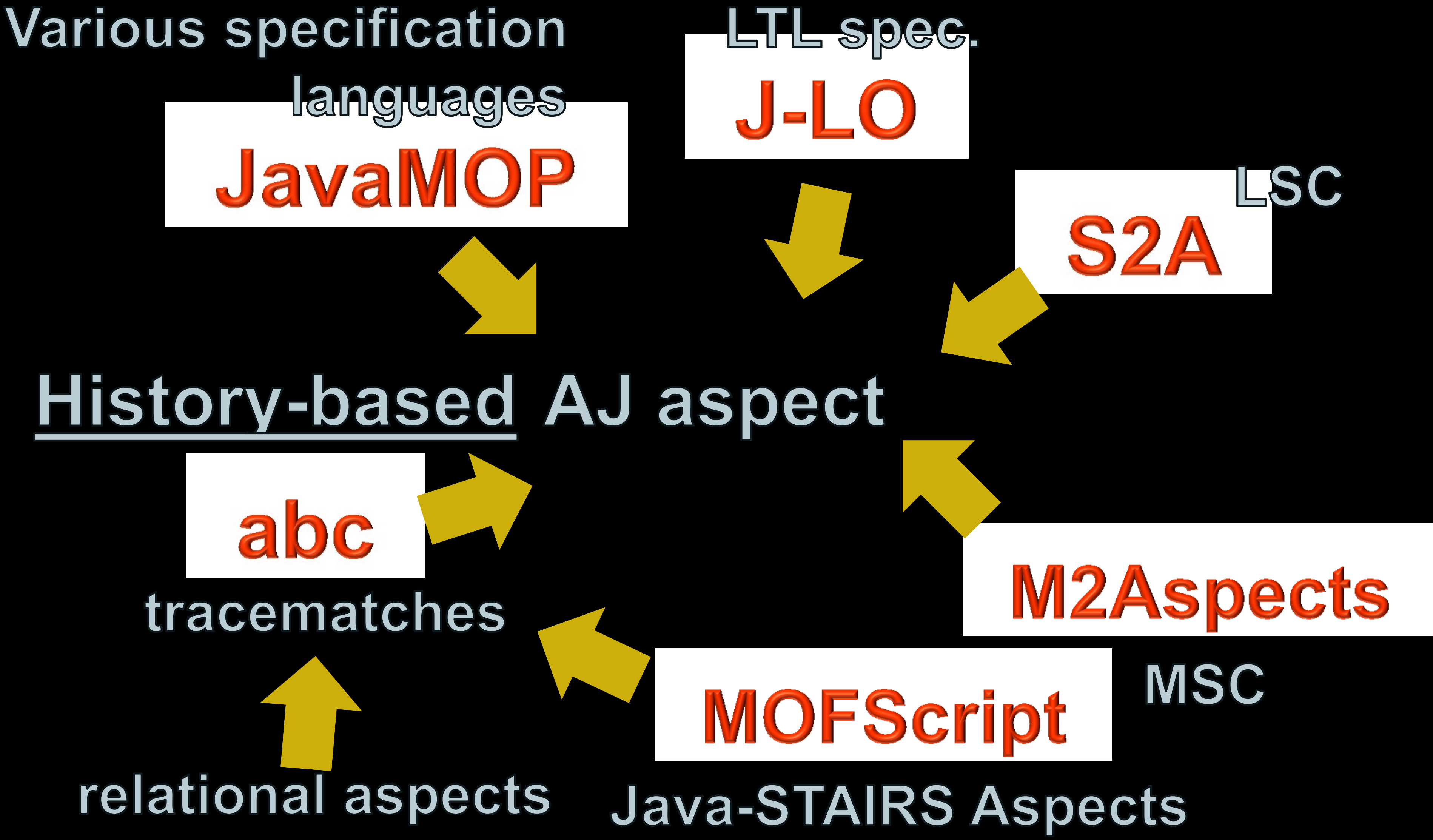
Optimizing Dependent Advice

Motivated by tracematch-based analysis, Bodden, Hendren & Lhotak (ECOOP 2007)

Two analysis stages:

- Quick check
  - syntactic
- Flow-insensitive Orphan-shadows analysis
  - uses context-sensitive points-to information

Auto-generating dependent advice



Various specification languages

JavaMOP

LTL spec.

J-LO

S2A

LSC

History-based AJ aspect

abc

tracematches

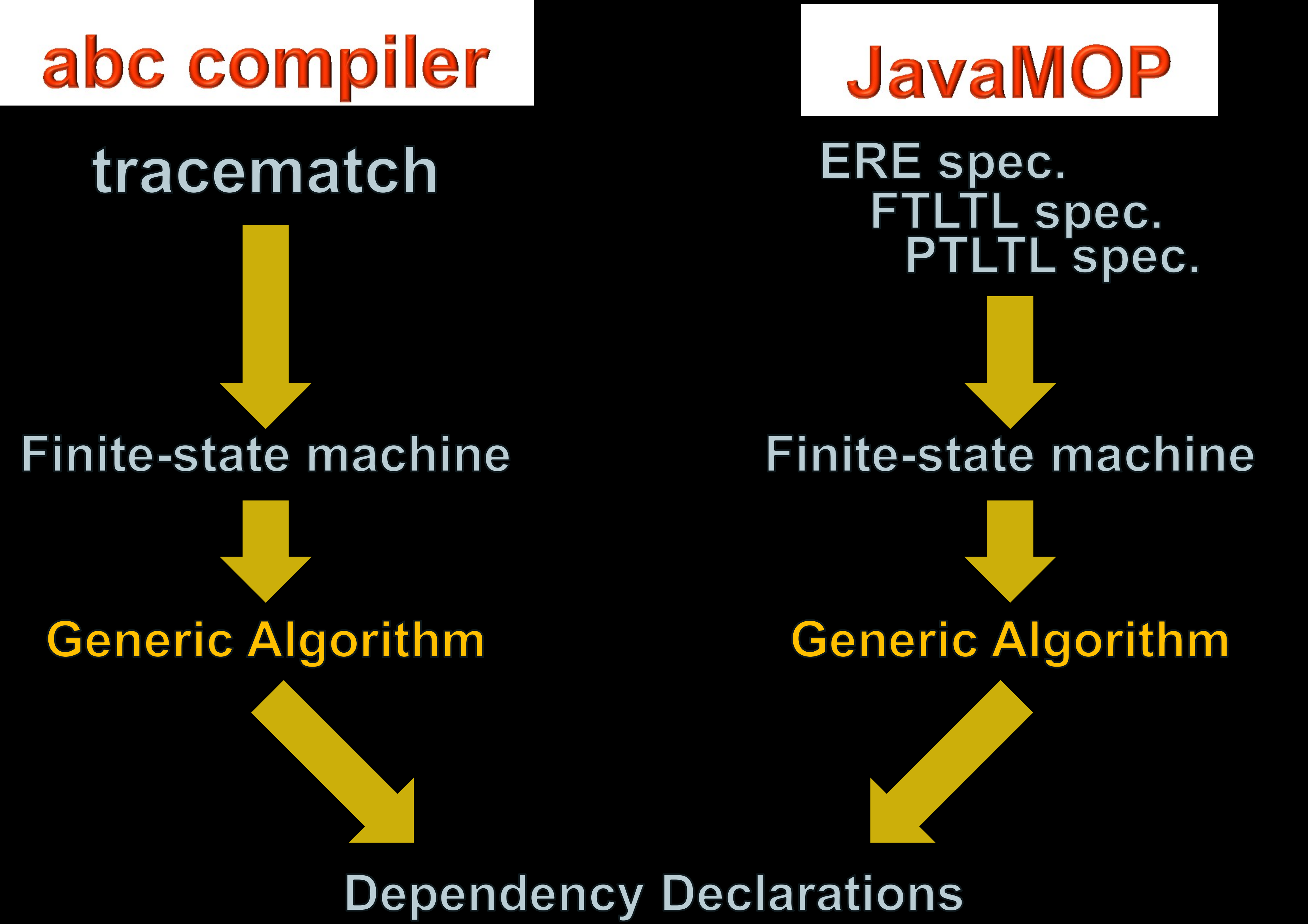
relational aspects

M2Aspects

MOFScript

MSC

Java-STAIRS Aspects



abc compiler

JavaMOP

tracematch

ERE spec.
FTLTL spec.
PTLTL spec.

Finite-state machine

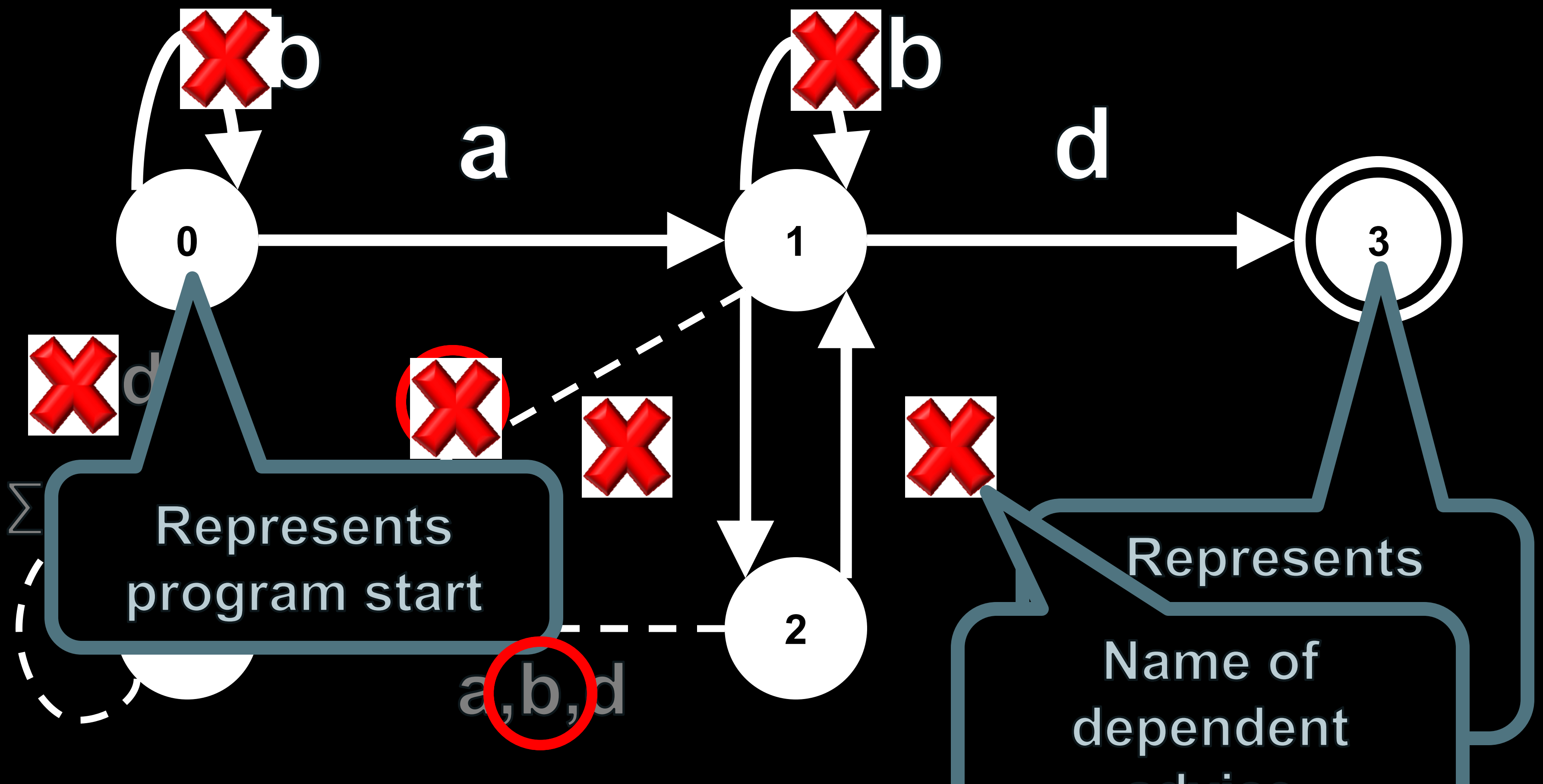
Finite-state machine

Generic Algorithm

Generic Algorithm

Dependency Declarations

FSM → dependency declarations



Represents program start

Represents

Name of dependent advice

```
dependency{  strong a,d; we
dependency{  strong a d; weak b; }
```

Proven: Algorithm is "stable"

Equivalent automata yield equivalent dependency declarations

Benchmarks - Properties

- ASyncIter
- ASyncIterM
- FailSafeEnum
- FailSafeEnumHT
- FailSafeIter
- FailSafeIterM
- HasNext
- LeakingSync
- Reader
- Writer

Benchmarks - Properties

For each of the ten properties:

- Hand-coded AspectJ aspect & annotations
- Tracematch

Where possible:

- JavaMOP specification in ERE

For three specifications also:

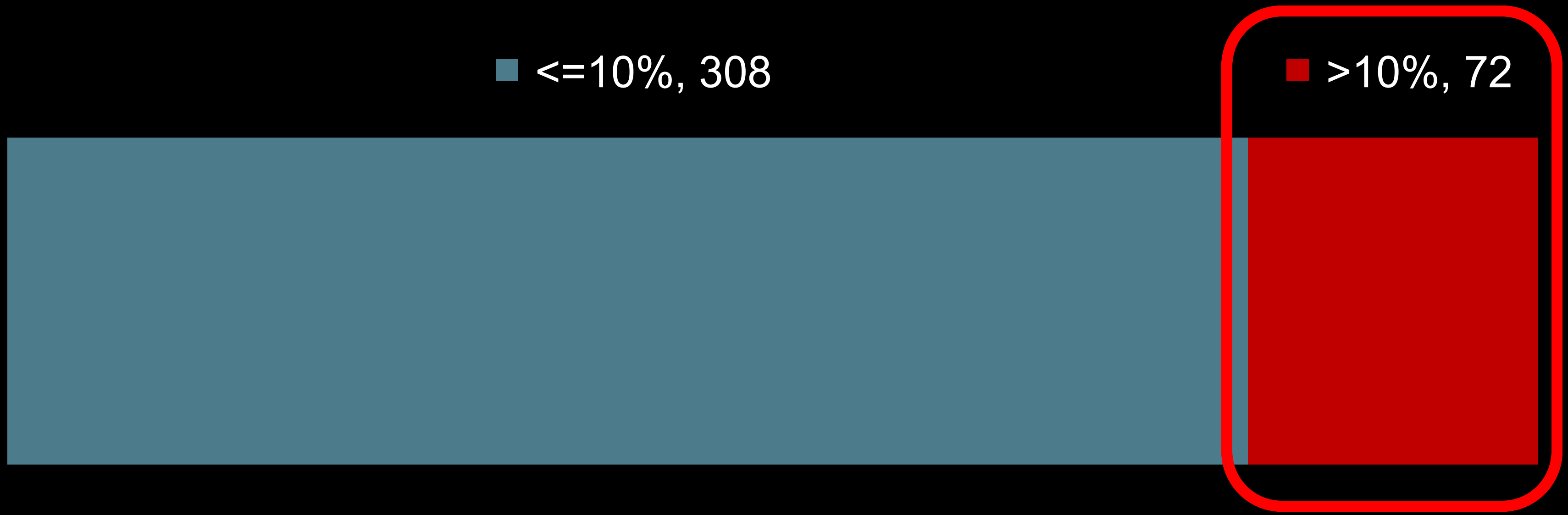
- JavaMOP specification in FTLTL
- JavaMOP specification in PTLTL

Benchmark programs

DaCapo benchmark suite:

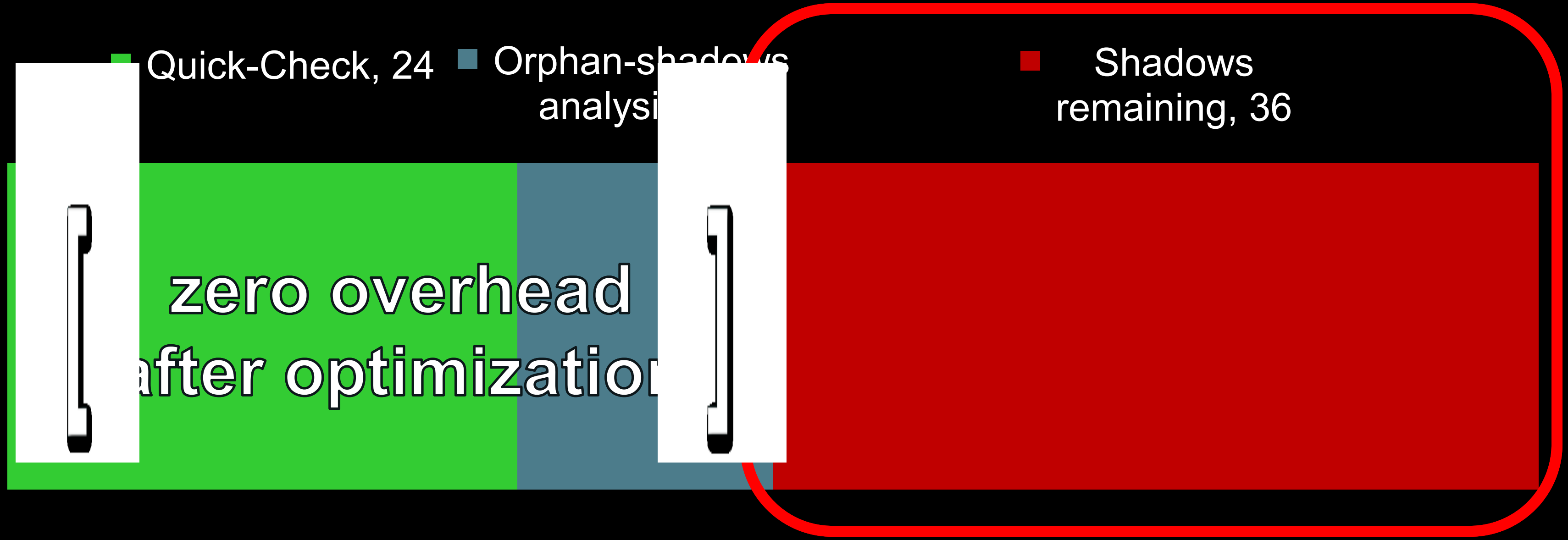
antlr	hsqldb
bloat	jython
chart	lucene
eclipse	pmd
fop	xalan

Runtime overhead



380 woven programs

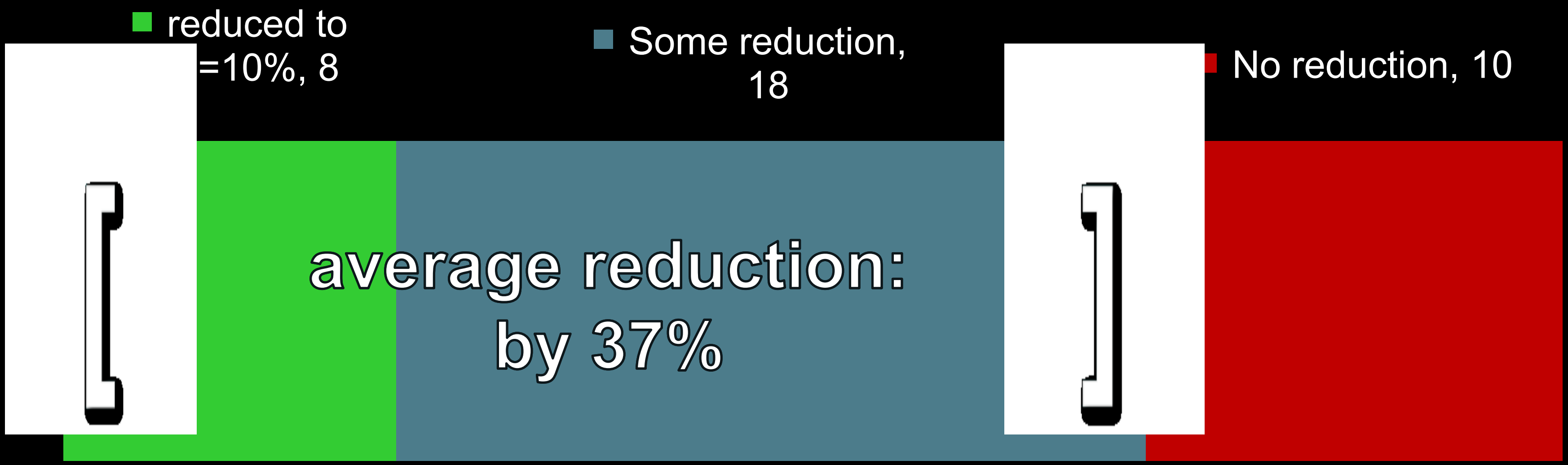
Elimination of all shadows



zero overhead after optimization

72 programs with overhead >10%

Reduction of runtime overhead



average reduction:
by 37%

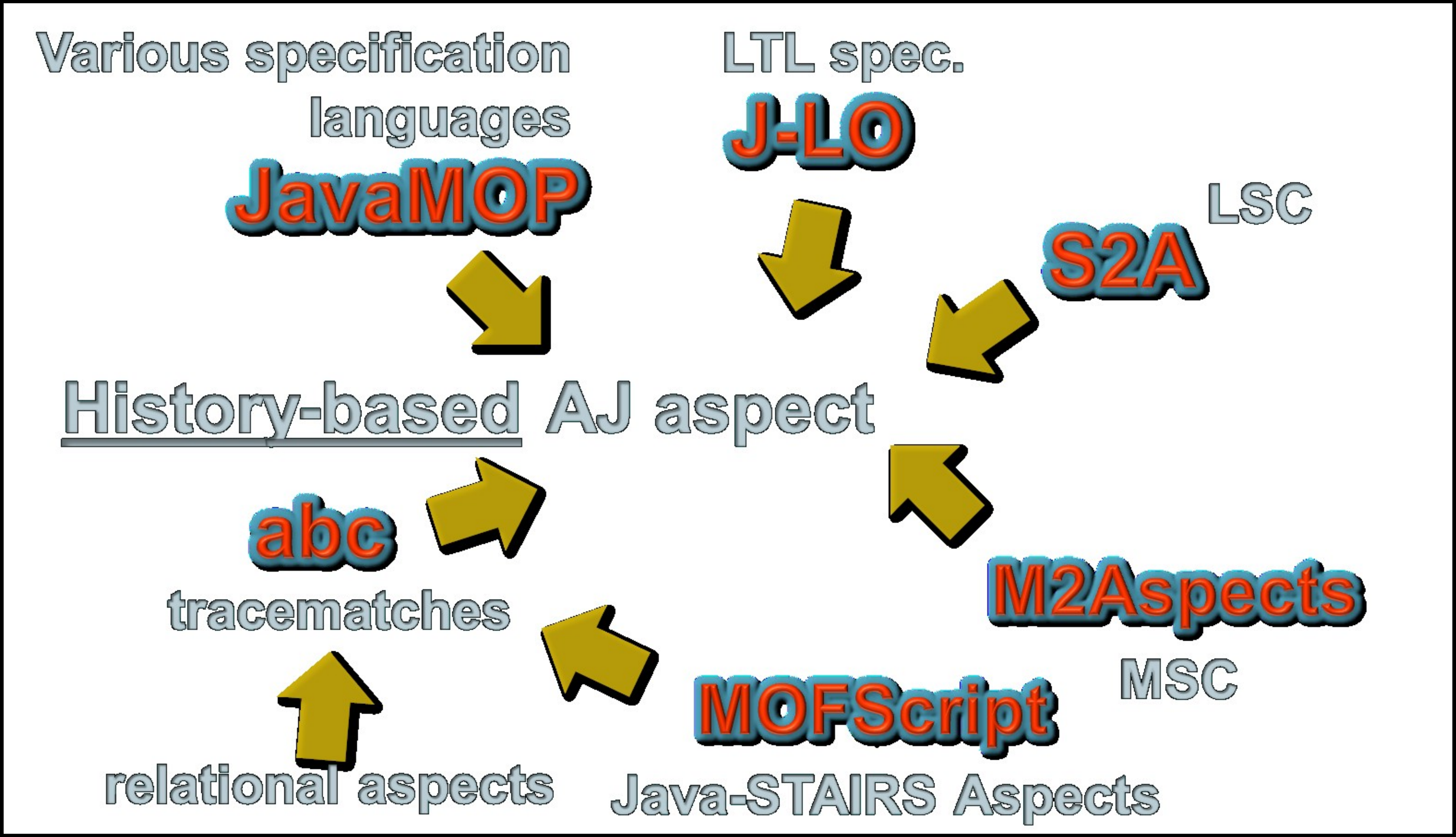
36 programs with overhead >10%
and shadows remaining

Limitations

Law-of-Demeter Checker (Lieberherr et al.)

```
after() returning(Object o):IgnoreTargets() {
  ignoredTargets.put(o,o);
}

after(Object thiz,Object targt):
     Any.MethodCall(thiz,targt) && !IgnoreCalls() {
     if (!ignoredTargets.containsKey(targt) &&
         !Pertarget.aspectOf(thiz).contains(targt)){
           objectVolations.put(tjSP, tjSP);
     }
}
```

Related work



Various specification languages

JavaMOP

LTL spec.

J-LO

LSC

S2A

History-based AJ aspect

abc

tracematches

relational aspects

M2Aspects

MSC

MOFScript

Java-STAIRS Aspects

Related work

Already mentioned:

- S2A (Maoz & Harel, FSE 2006)
- M2Aspects (Krüger et al., SCESM 2006)
- Java-STAIRS Aspects (Oldevik & Haugen, *5pm*)
- J-LO (Bodden, Diploma Thesis)

Other possible clients of dependent advice:

- Association aspects (Sakurai et al., AOSD 2004)
- LogicAJ (Kniesel et al., RAM-SE 2004)
- Dataflow pointcuts (Masuhara & Kawauchi, APLAS 2003; and *tomorrow, 14:30*)
- Conditional compilation (Adams et al., *Friday*)

Related work

Optimizations for tracematches:

- Bodden, Hendren & Lhotak, ECOOP 2007
- Bodden, Lam & Hendren, FSE 2008
- Naeem & Lhotak, OOPSLA 2008

Optimizations of the Runtime Monitor:

- Avgustinov et al., OOPSLA 2007
- Chen & Rosu, TACAS 2009

Important conclusion

Approach hard to formalize without AOP

- History-based aspect modularizes instrumentation
- Hence can use modular dependency annotation

Acknowledgements

co-workers:

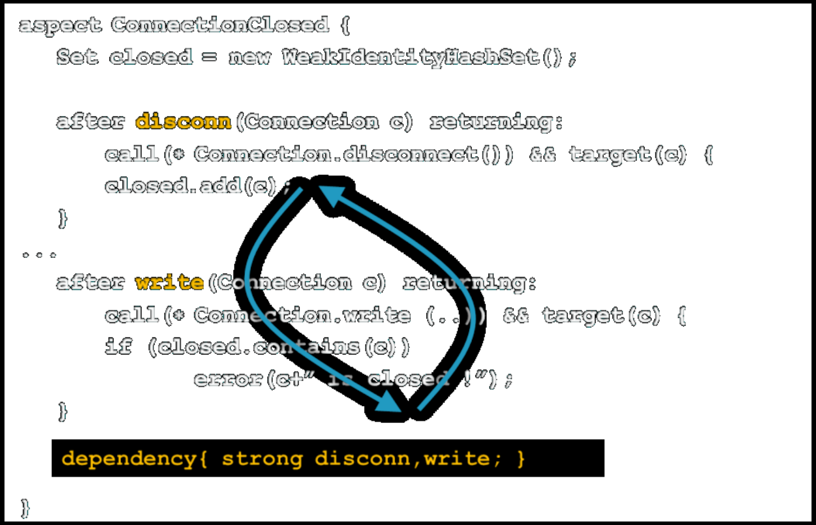
- Laurie Hendren
- Patrick Lam

developer of S2A:

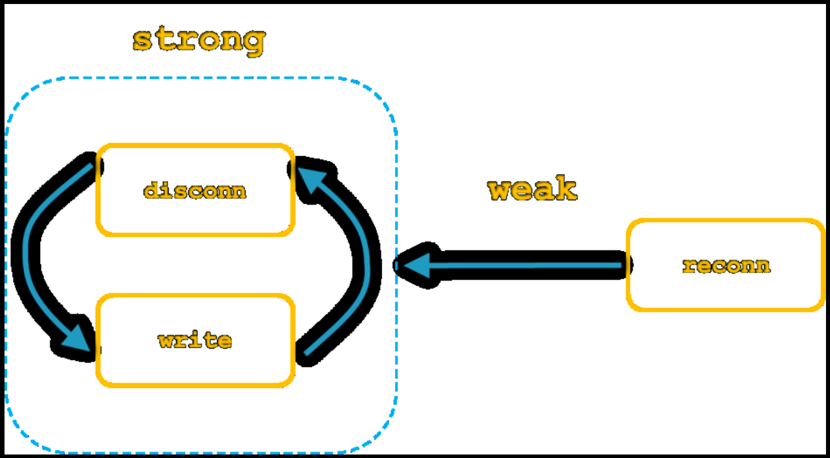
- Shahar Maoz

abc/tracematch maintainers:

- Pavel Avgustinov
- Julian Tibble

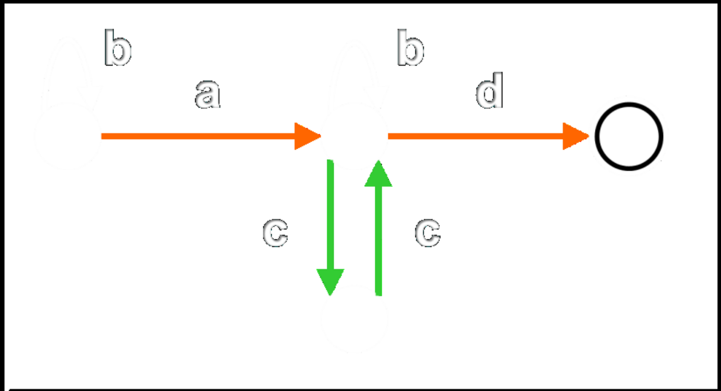


```
aspect ConnectionClosed {
  Set closed = new WeakIdentityHashSet();

  after disconn(Connection c) returning:
      call(* Connection.disconnect()) && target(c) {
      closed.add(c);
  }
...
  after write(Connection c) returning:
      call(* Connection.write (..)) && target(c) {
      if (closed.contains(c))
            error(c+" is closed !");
  }
  dependency{ strong disconn,write; }
}
```

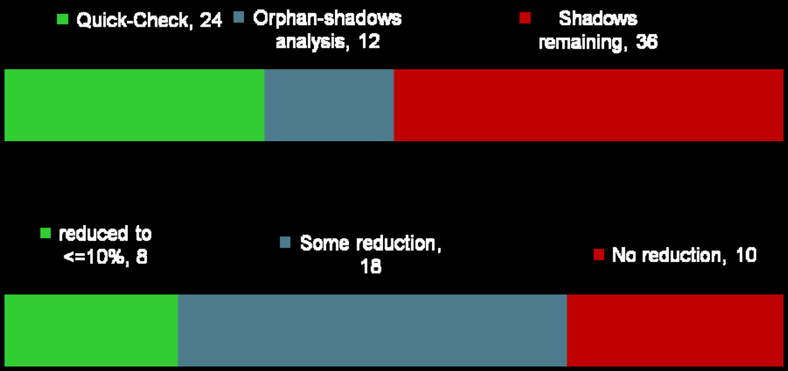


www.aspectbench.org

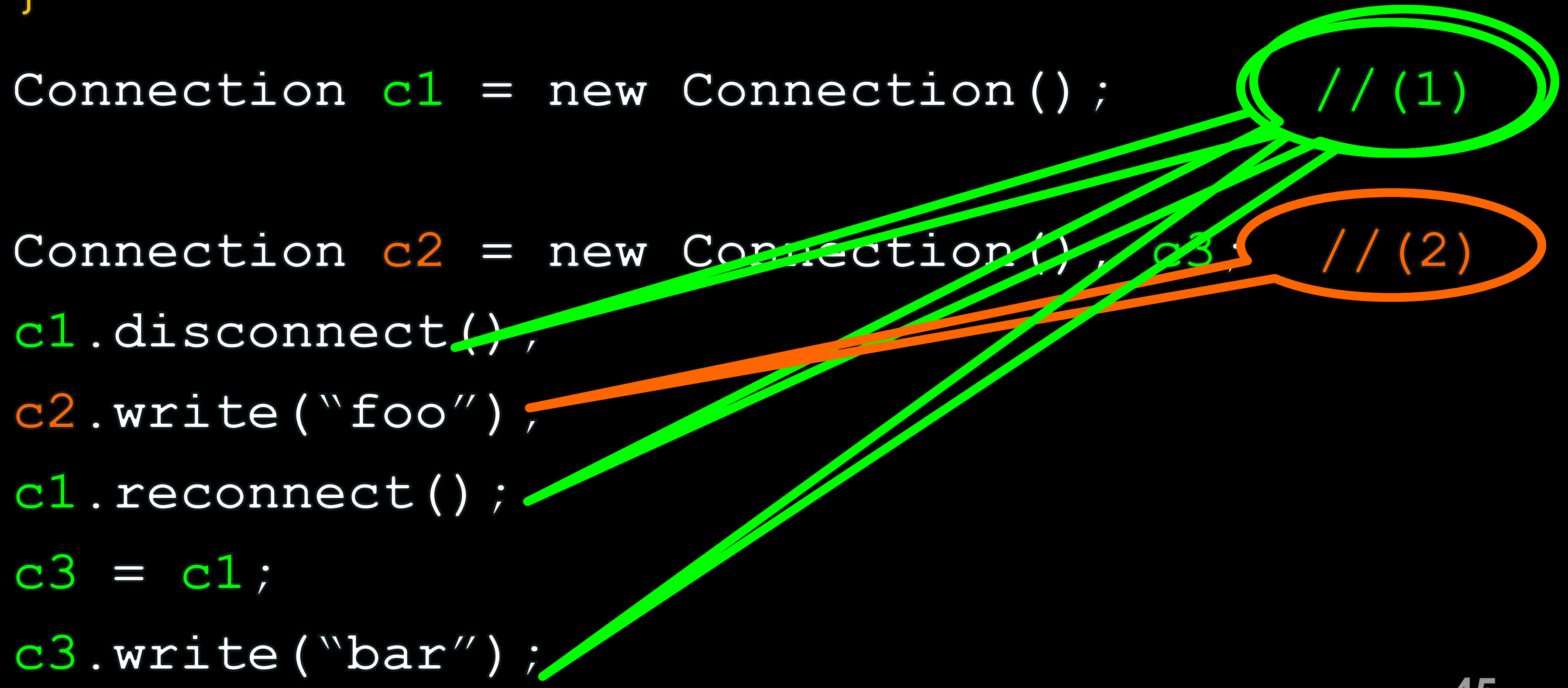
www.bodden.de



```
dependency{  strong a,d;  weak c; }
dependency{  strong a,c,d; weak b;  }
```

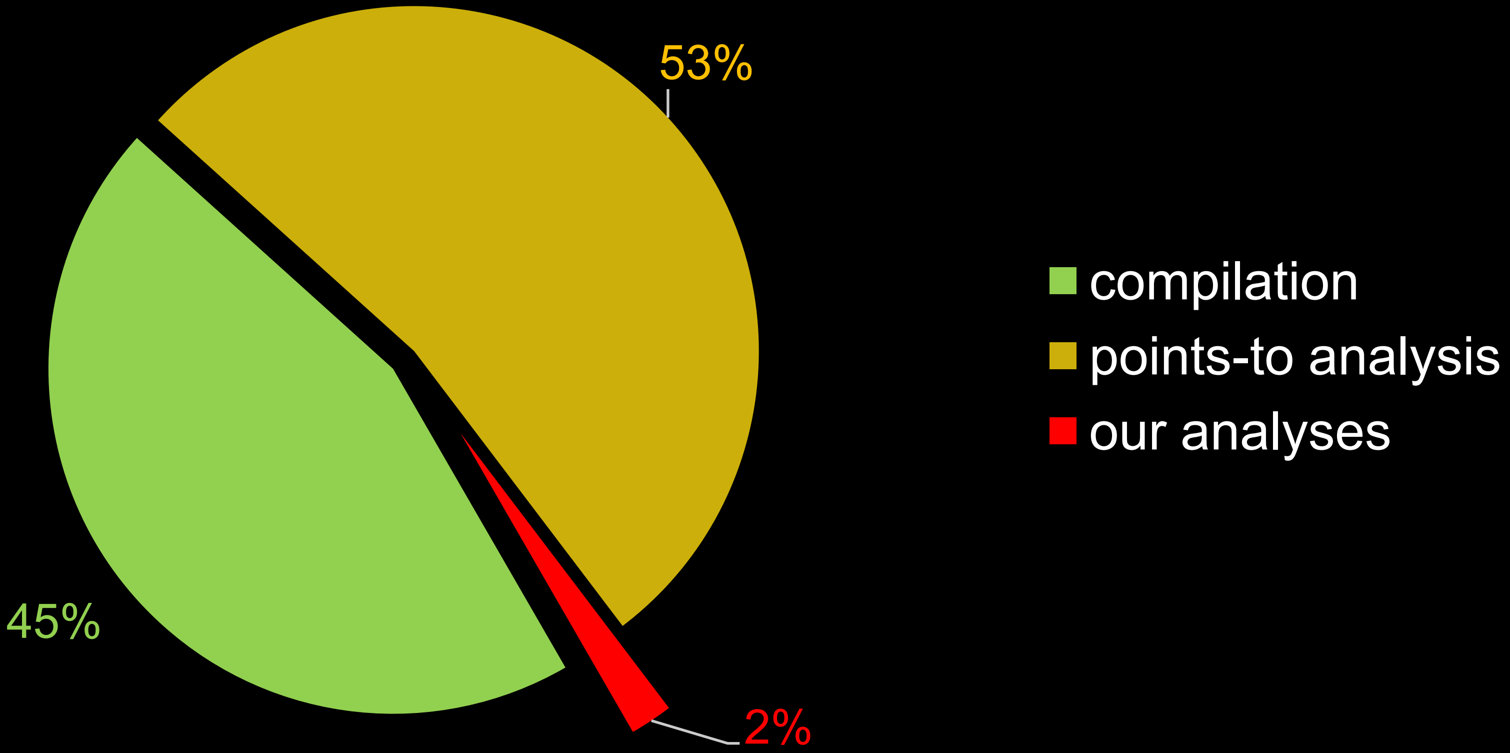


Static analysis



```
dependency{
   strong disconn(c),write(c);
   weak reconn(c);
}
Connection c1 = new Connection();        //(1)

Connection c2 = new Connection(), c3;    //(2)
c1.disconnect();
c2.write(“foo”);
c1.reconnect();
c3 = c1;
c3.write(“bar”);
```

Results – Static-analysis time



Average total: 12 minutes
Max total: 58 minutes