Università della Svizzera italiana | Faculty of Informatics

Java Bytecode Instrumentation - Reconciling Developer Productivity

Aibek Sarimbekov, Yudi Zheng, Danilo Ansaloni, Walter Binder
University of Lugano, Lugano, Switzerland

Lubomir Bulej, Lukas Marek, Petr Tuma
Charles University, Prague, Czech Republic

Zhengwei Qi
Shanghai Jiao Tong University, Shanghai, China

March 23rd 2013

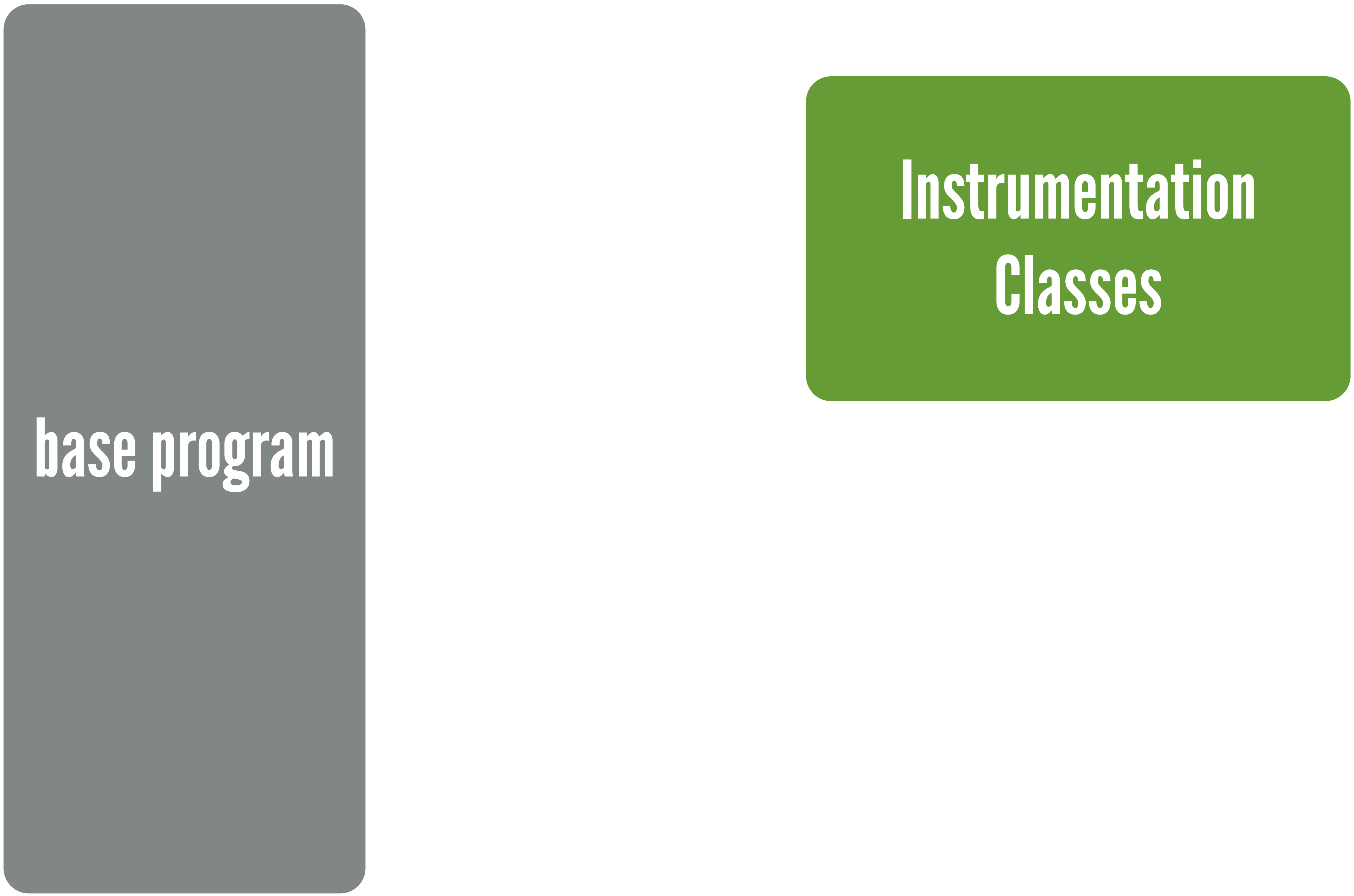
Dynamic Program Analysis Tools

Tools that observe relevant activities of running programs.

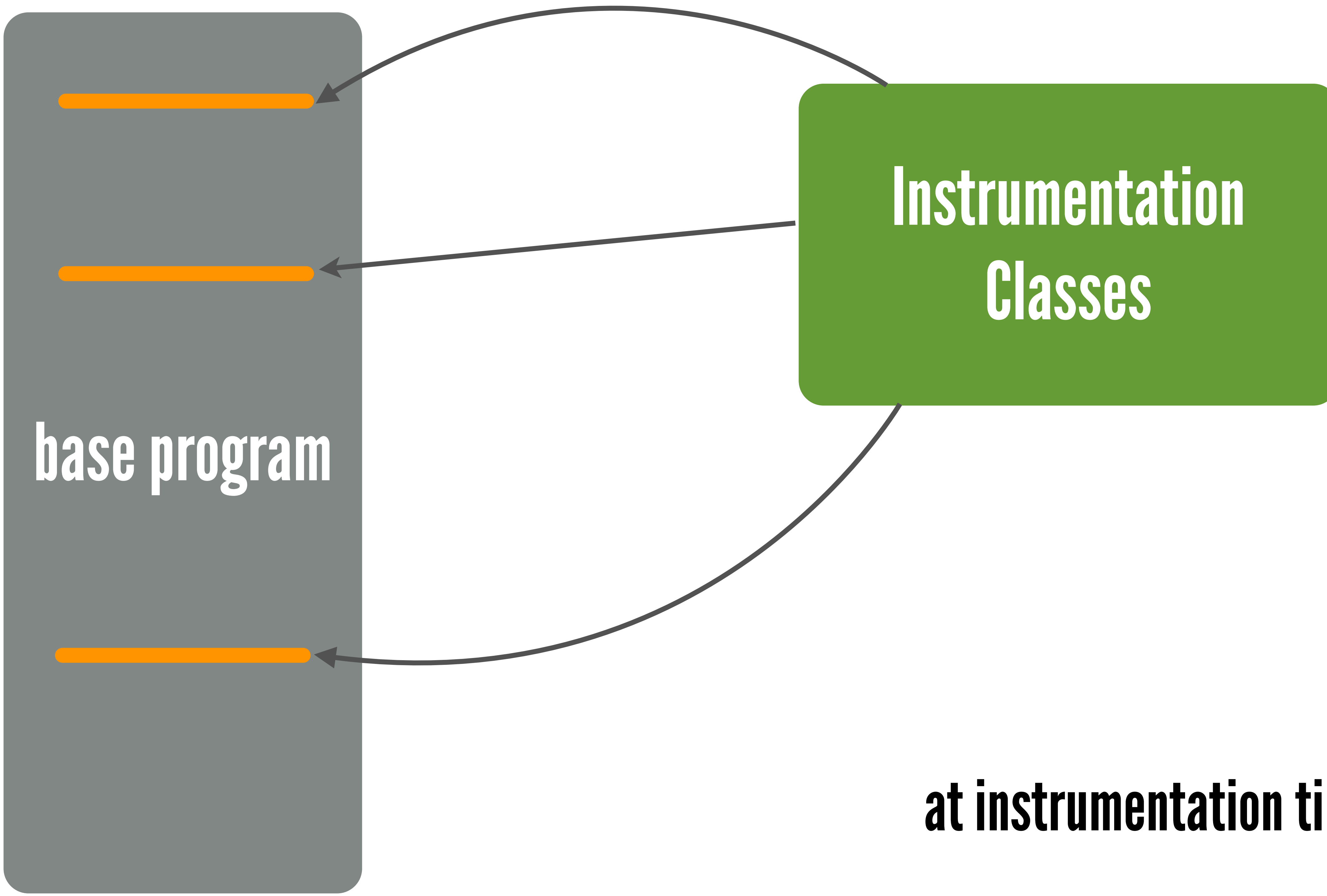
Examples:

- profiling
- debugging
- testing
- reverse engineering
- program comprehension

Instrumentation Classes vs. Analysis Classes

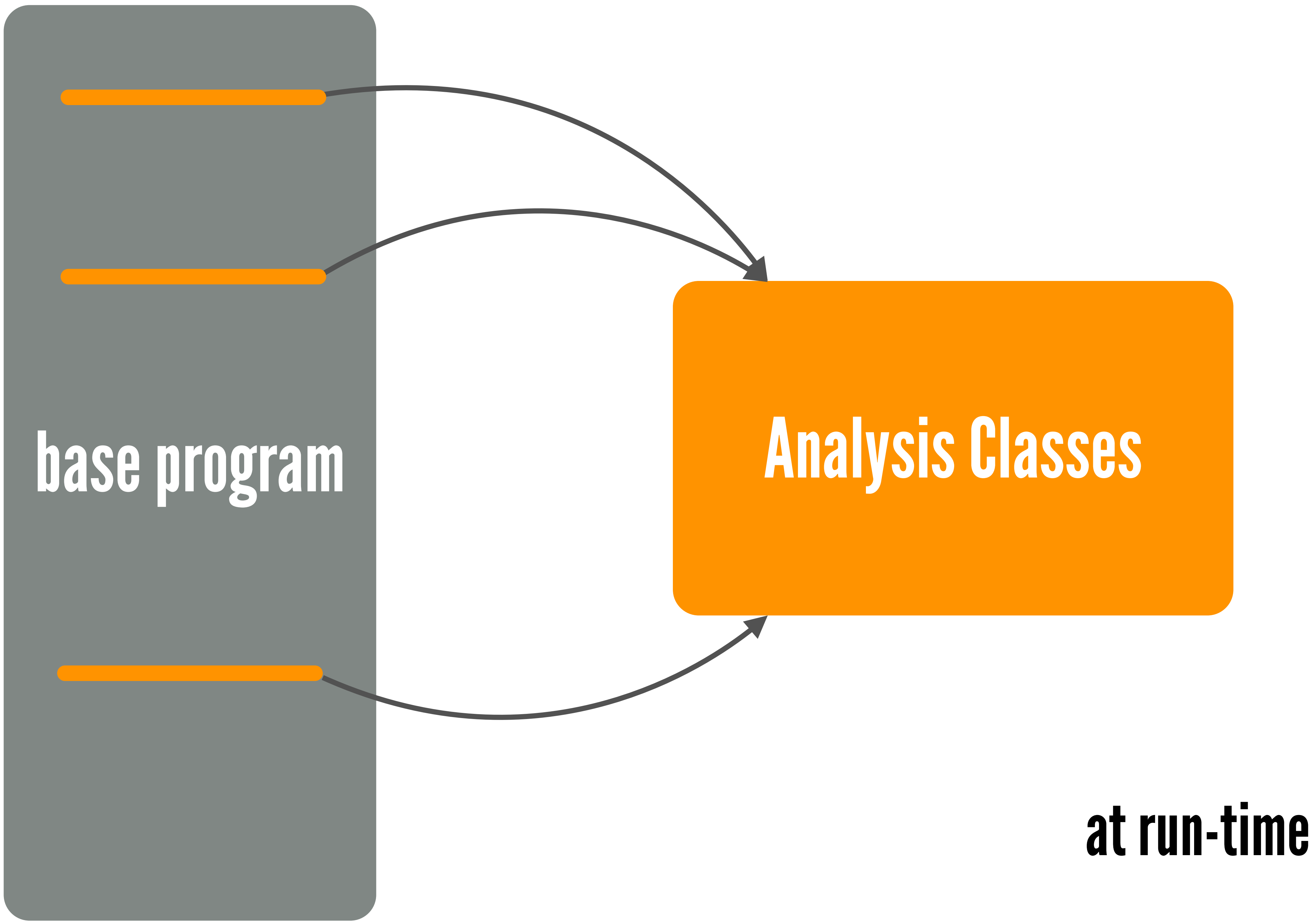


Instrumentation Classes vs. Analysis Classes



at instrumentation time

Instrumentation Classes vs. Analysis Classes



Bytecode Instrumentation

BCEL

Javassist

WALA
T. J. WATSON LIBRARIES FOR ANALYSIS

DiSL

Soot

ASM

jChord

Bytecode Instrumentation

BCEL

Javassist

WALA

T. J. WATSON LIBRARIES FOR ANALYSIS

DiSL

jChord

Soot

ASM

Which to use?

Research Question

Can we raise the abstraction level for writing dynamic analysis tools, allowing software engineers to rapidly develop custom analysis tools, impairing neither expressiveness nor tool performance?


DiSL
ASM
VS.

Empirical Study

- Controlled Experiment
- Recasts of 10 open-source DPA Tools

DiSL: a DSL for Instrumentations

- Provide higher abstraction layer to write instrumentations
- Reduce development effort for writing instrumentations
- Avoid shortcomings of AOP-based approach
- Do not impair the performance of the resulting tools

DiSL at a Glance

Language constructs

- markers and snippets
- static and dynamic context
- scope and guards
- synthetic and thread local variables

DiSL Markers and Snippets Example

```java
public class DiSL {

   @Before(marker = BasicBlockMarker.class)
   public static void onBB() {
      Profile.profileBB(); // count number of executed basic blocks of code
   }

   @AfterReturning(marker = BytecodeMarker.class, args = "new")
    public static void onAlloc() {
      Profile.profileAlloc(); // count the number of allocated objects
    }

}
```

DiSL Static Context Information Example

```
public class DiSL {

   @Before(marker = BasicBlockMarker.class)
   public static void onBB(MethodStaticContext msc,
                  UniqueMethodId uid,
                  BasicBlockStaticContext bbsc) {
      Profile.profileBB(
            msc.thisMethodFullName(), // full method name
            uid.get(),                // unique method ID (int value)
            bbsc.getBBindex(),        // basic block index (int value)
            bbsc.getBBSize()          // bytecodes in the BB (int value)
      );
   }
}
```

DiSL Dynamic Context Information Example

```java
public class DiSL {

   @AfterReturning(marker = BytecodeMarker.class, args = "new")
   public static void onBB(DynamicContext dc) {
      // access allocated object
      Object allocObj = dc.getStackValue(0, Object.class);
      Profile.profileAlloc(allocObj);
   }
}
```

DiSL Scope and Guards Example

```
public class DiSL {

   @Before(marker = BasicBlockMarker.class,
         scope = "TargetClass.*", // constrain instrumentation
         guard = LoopGuard.class) // constrain instrumentation
   public static void onBB(BasicBlockStaticContext bbsc) {
      Profile.profileBB(bbsc.getBBSize());
   }
}

public class LoopGuard {
   @GuardMethod
   public static boolean isApplicable(BasicBlockStaticContext bbsc) {
      return bbsc.isFirstOfLoop(); // instrument only first BBs of loops
   }
}
```

DiSL Synthetic Local Variables Example

```
public class DiSL {

    @SyntheticLocal
    static int bbsSL;
    @SyntheticLocal
    static long sizeSL;

    @Before(marker = BasicBlockMarker.class)
    public static void onBB(BasicBlockStaticContext bbsc) {
        bbsSL++;
        sizeSL += bbsc.getBBSize();
    }

    @After(marker = BodyMarker.class)
    public static void onMethodCompletion() {
        Profile.profileExecBytecodes(bbsSL, sizeSL);
    }
}
```

DiSL Thread Local Variables Example

```
public class DiSL {

   @ThreadLocal
   static Profile profileTL;

   @Before(marker = BodyMarker.class, order = 1)
   public static void onMethodEntry() {
      if (profileTl == null)
         profileTL = new Profile();
   }

   @Before(marker = BasicBlockMarker.class, order = 0)
   public static void onBB(BasicBlockStaticContext bbsc) {
      profileTL.profileBB(bbsc.getBBSize());
   }
}
```

Empirical Study

- Controlled Experiment
- Recasts of 10 open-source DPA Tools

Controlled Experiment

1. Goal: to identify a framework that allows rapid development of correct DPA tools
2. Subjects: students from Shanghai Jiao Tong University
3. Variables
   - independent: use of ASM or DiSL
   - dependent: time, correctness
4. Tasks: 6 typical instrumentations

Controlled Experiment: Procedure

- Self-assessment Questionnaire
- Tutorial on DPA
- Distribution of tasks
- Q&A Session
- Experiment
- Debriefing Questionnaire

Controlled Experiment: Procedure

Università della Svizzera italiana

Thank you for participating in our experiment. We very much appreciate your help. Please fill in the following questionnaire.

Participant's details

1. Full name: ____________________
2. Contact e-mail address: ____________________
3. Age: __________
4. Gender: ○ male ○ female
5. Affiliation: ____________________
6. Current highest academic degree:
   - ○ Bachelor student
   - ○ Master student
   - ○ Ph.D. student
   - ○ Professor
   - ○ Other: __________
7. Experience level in:

	None	Beginner	Knowledgeable	Advanced	Expert
OOP	○	○	○	○	○
Java	○	○	○	○	○
Using Eclipse IDE	○	○	○	○	○
Dynamic Program Analysis	○	○	○	○	○
AspectJ	○	○	○	○	○
ASM	○	○	○	○	○
Linux	○	○	○	○	○
JVM and Java bytecode	○	○	○	○	○

Università della Svizzera italiana

Task 2

Implement the instrumentation that injects a call to `onArrayAllocation(int arrayLength)`, before each array allocation, where `arrayLength` is the length of the allocated array.

Note: your instrumentation should intercept only these bytecodes: NEWARRAY, ANEWARRAY

Time needed for task completion: _______ minutes

Time Pressure:

- ○ Too much time pressure.
- ○ Fair amount of pressure.
- ○ Not so much time pressure.
- ○ Very little time pressure.
- ○ No time pressure at all.

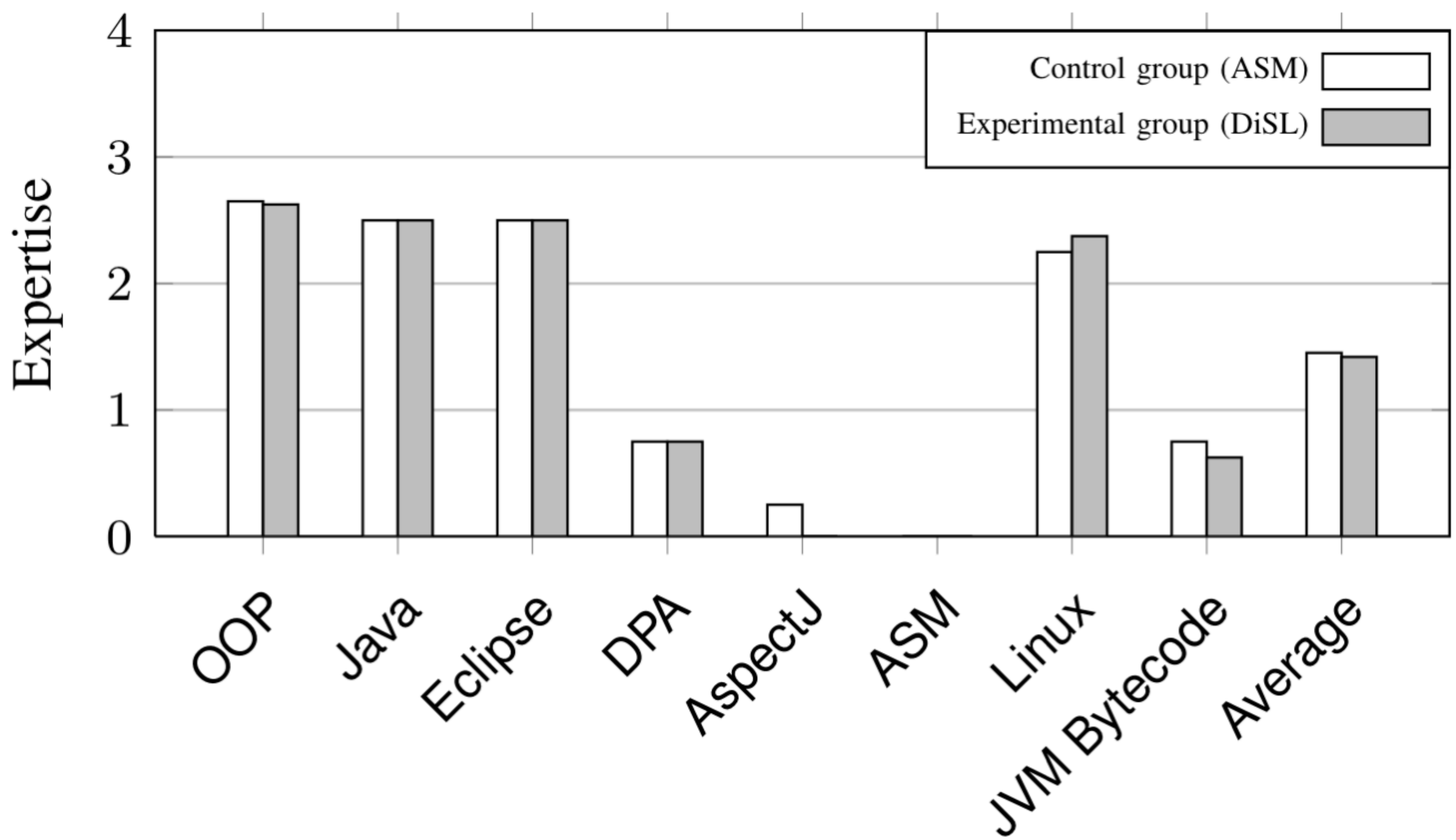
Difficulty:

- ○ trivial
- ○ simple
- ○ intermediate
- ○ difficult
- ○ impossible

Comments:

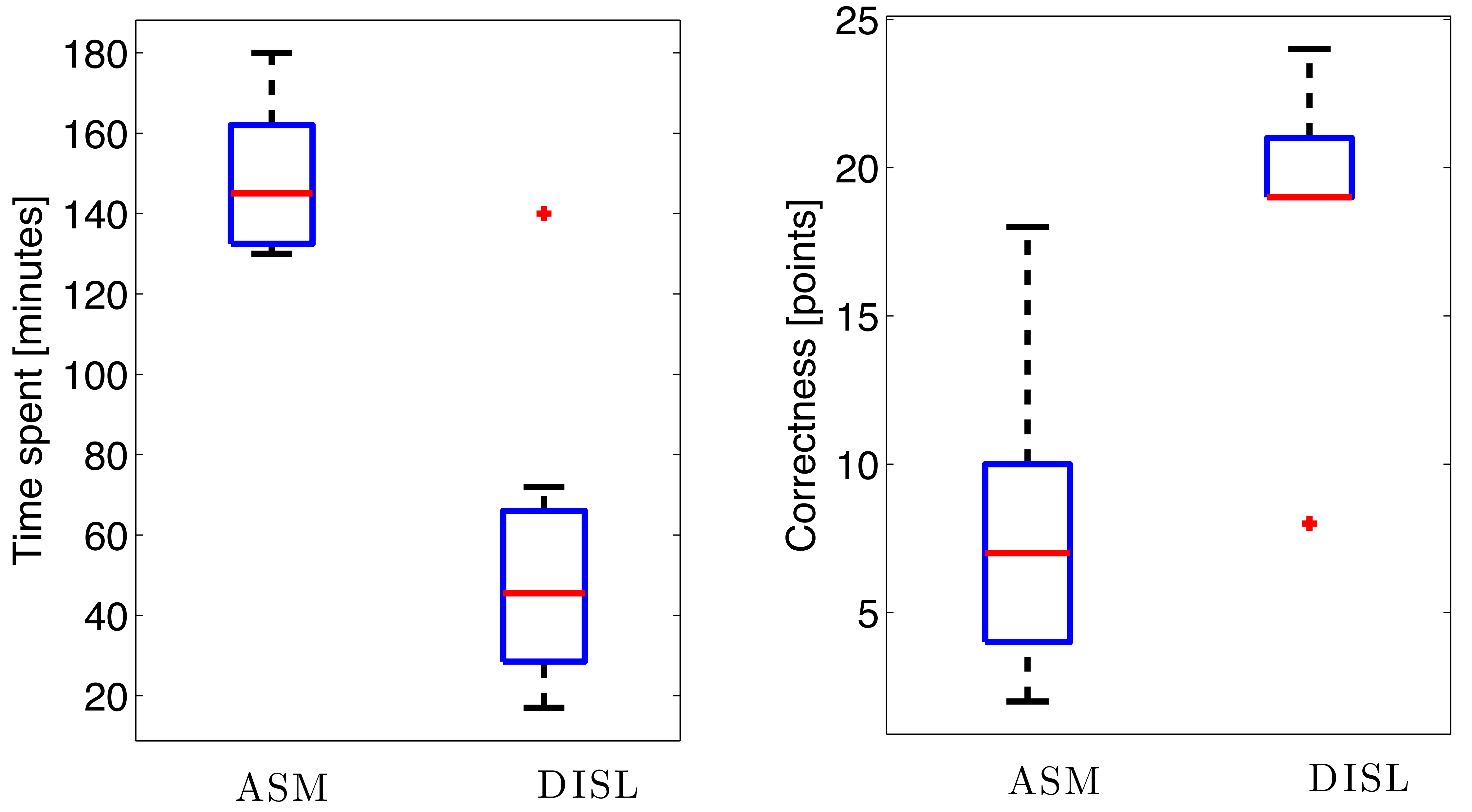
Controlled Experiment: Procedure

0 - none 1 - beginner 2 - knowledgeable 3 - advanced 4 - expert



In total: 16 BSc, MSc, and PhD students

Controlled Experiment: Results



Controlled Experiment: Results

	Time [minutes]		Correctness [points]	
	ASM	DiSL	ASM	DiSL
Summary statistics				
mean	148.62	54.62	8.75	18.75
difference	-63.2%		+46.6%	
min	130	17	2	8
max	180	140	18	24
median	145	45.5	7	19
stdev.	18.73	38.96	5.92	4.68
Assumption checks				
Kolmogorov-Smirnov Z	0.267	0.203	0.241	0.396
Levene F	1.291		1.939	
One-tailed Student's t-test				
df	14		14	
t	6.150		-3.746	
p-value	<0.001		0.002	

descriptive statistics

Controlled Experiment: Summary

DiSL improves developer productivity compared to ASM. Results are both practically and statistically significant.

Empirical Study

- Controlled Experiment
- Recasts of 10 open-source DPA Tools

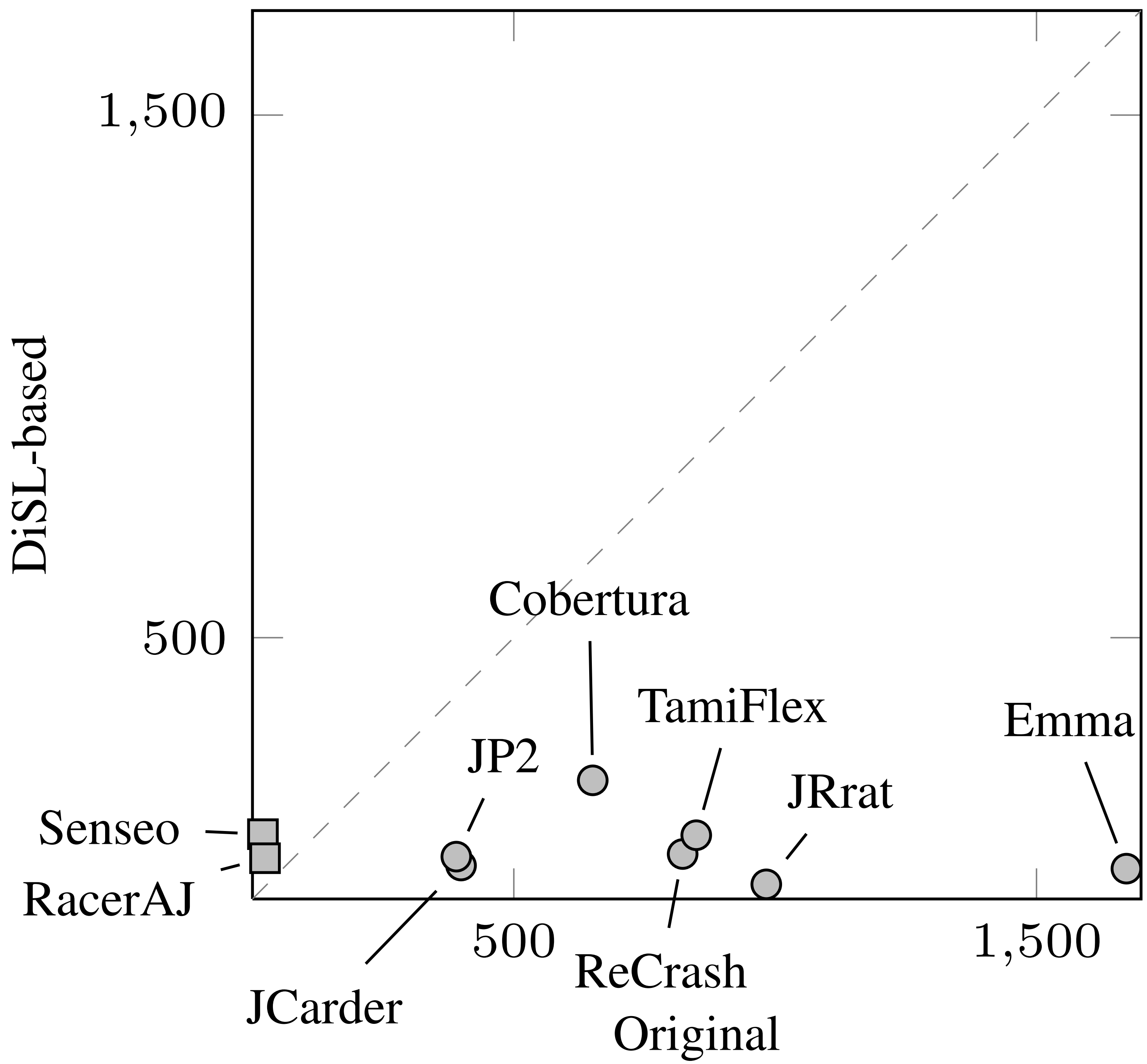
Recasts

10 open-source DPA tools.

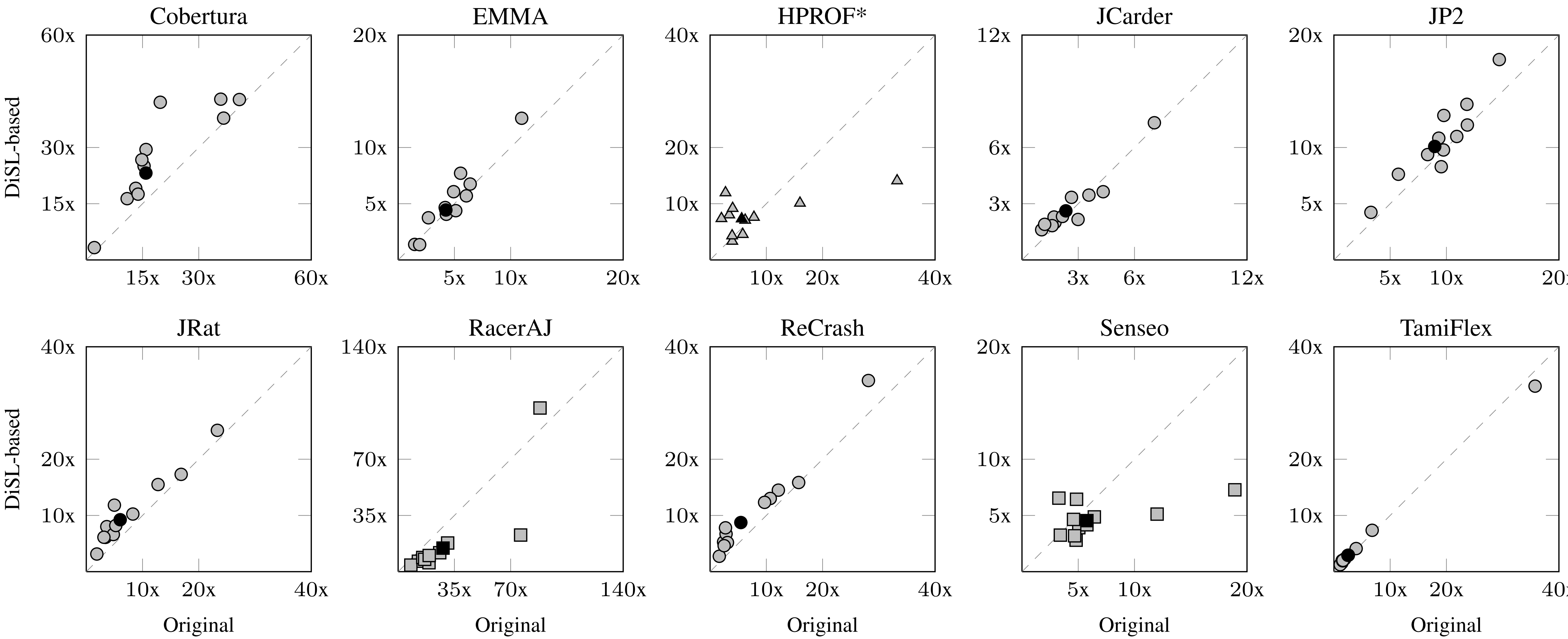
Metrics:

- Logical SLOC
- Performance Overhead

Logical SLOC



Performance Overhead



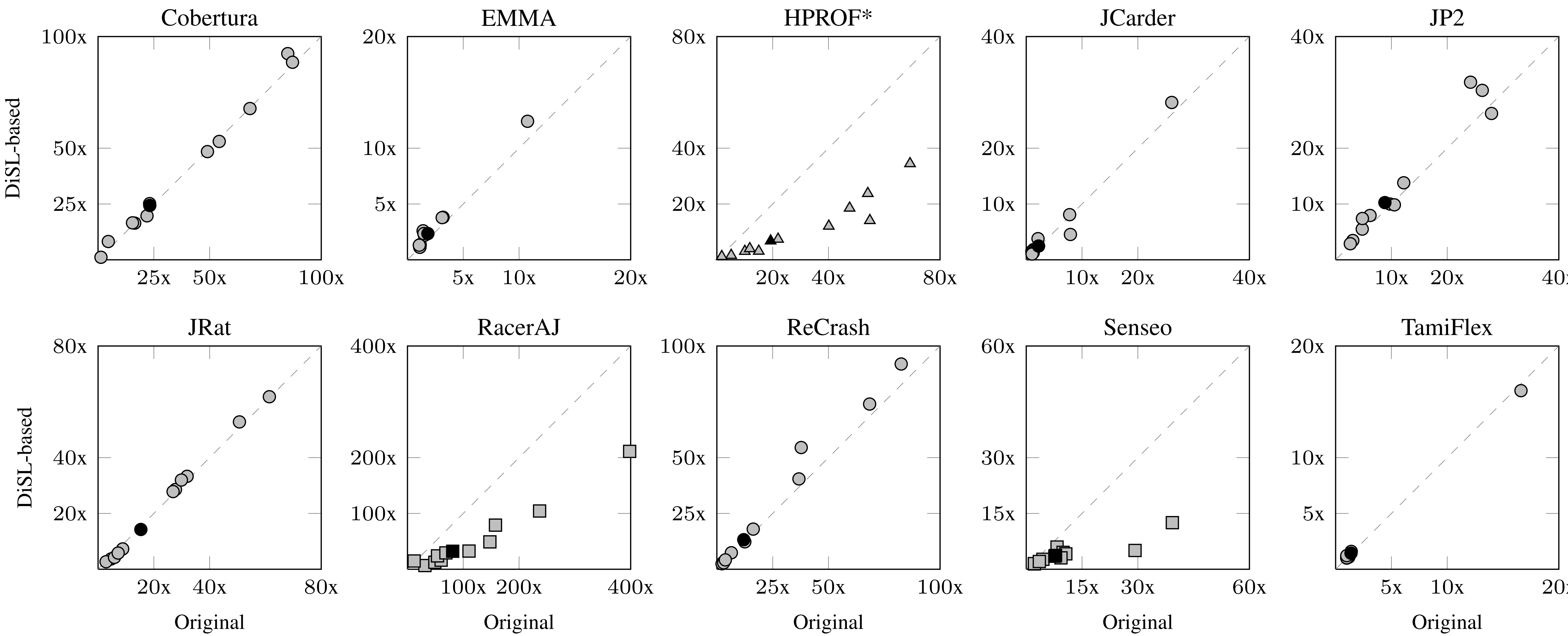
startup

- benchmarks from the Dacapo-9.12 suite

- geometric mean

Four quad-core Intel Xeon CPUs E7340, 2.4 GHz, 16 GB RAM, Ubuntu GNU/Linux11.04 64-bit with kernel 2.6.38, Oracle Java HotSpot 64-bit Server VM 1.6.0_29

Performance Overhead



steady-state

- benchmarks from the Dacapo-9.12 suite
- geometric mean

Four quad-core Intel Xeon CPUs E7340, 2.4 GHz, 16 GB RAM, Ubuntu GNU/Linux11.04 64-bit with kernel 2.6.38, Oracle Java HotSpot 64-bit Server VM 1.6.0_29

Conclusions

RQ: Can we raise the abstraction level for writing dynamic analysis tools, allowing software engineers to rapidly develop custom analysis tools, impairing neither expressiveness nor tool performance?

Conclusions

RQ: Can we raise the abstraction level for writing dynamic analysis tools, allowing software engineers to rapidly develop custom analysis tools, impairing neither expressiveness nor tool performance?

A: yes, use DiSL!