Università
della
Svizzera
italiana

**Faculty
of Informatics**

Introduction to **DiSL**

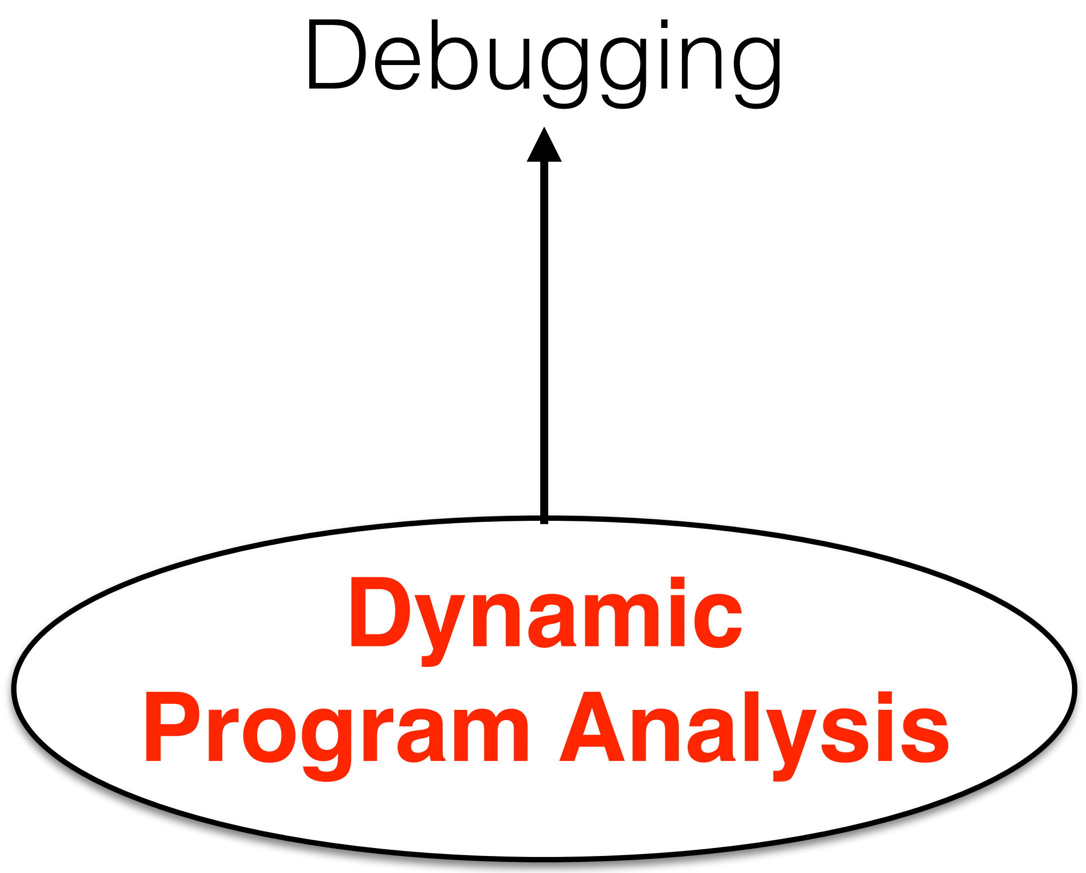
Yudi Zheng

Domain **S**pecific **L**anguage for bytecode **i**nstrumentation

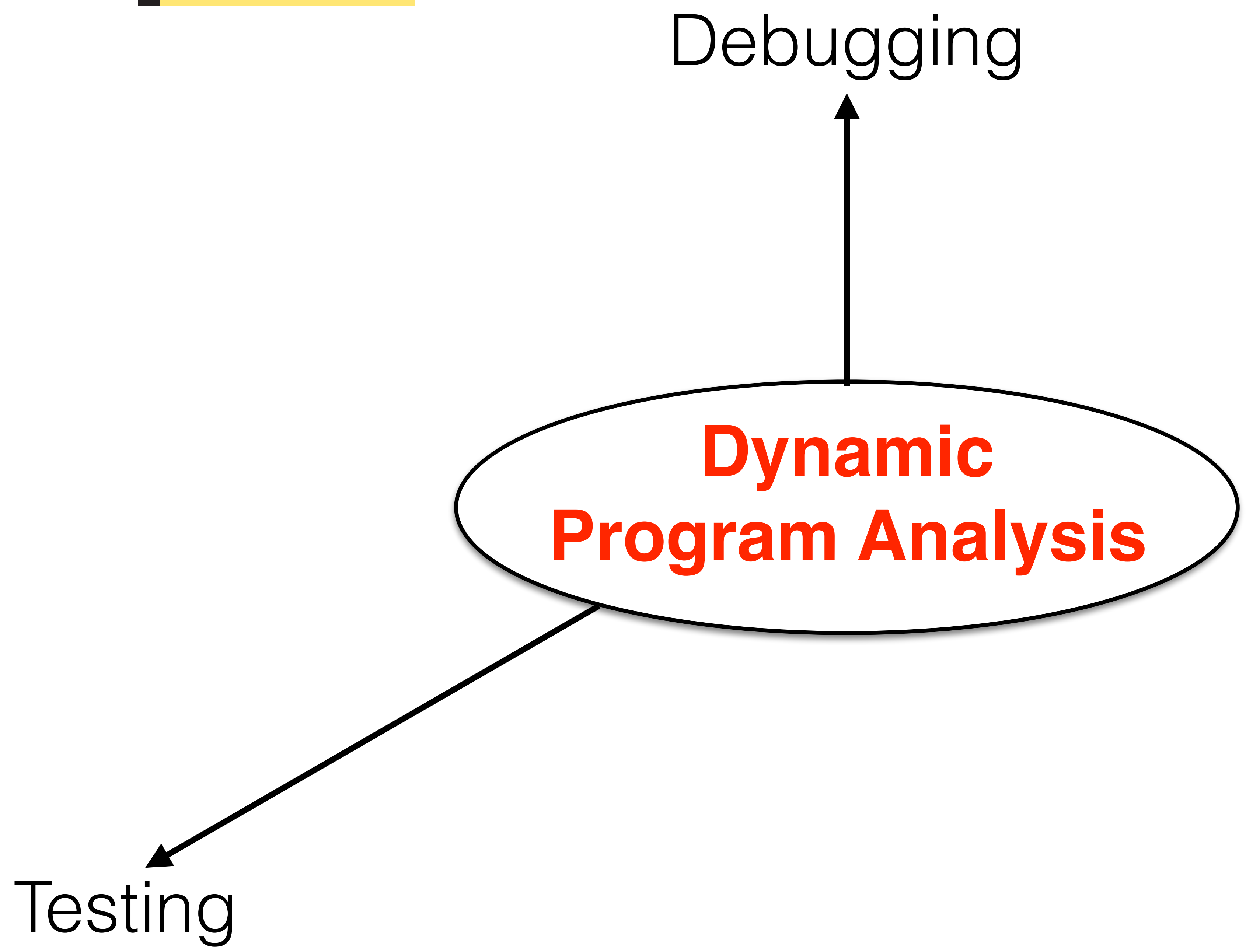
DiSL targets ...

**Dynamic
Program Analysis**

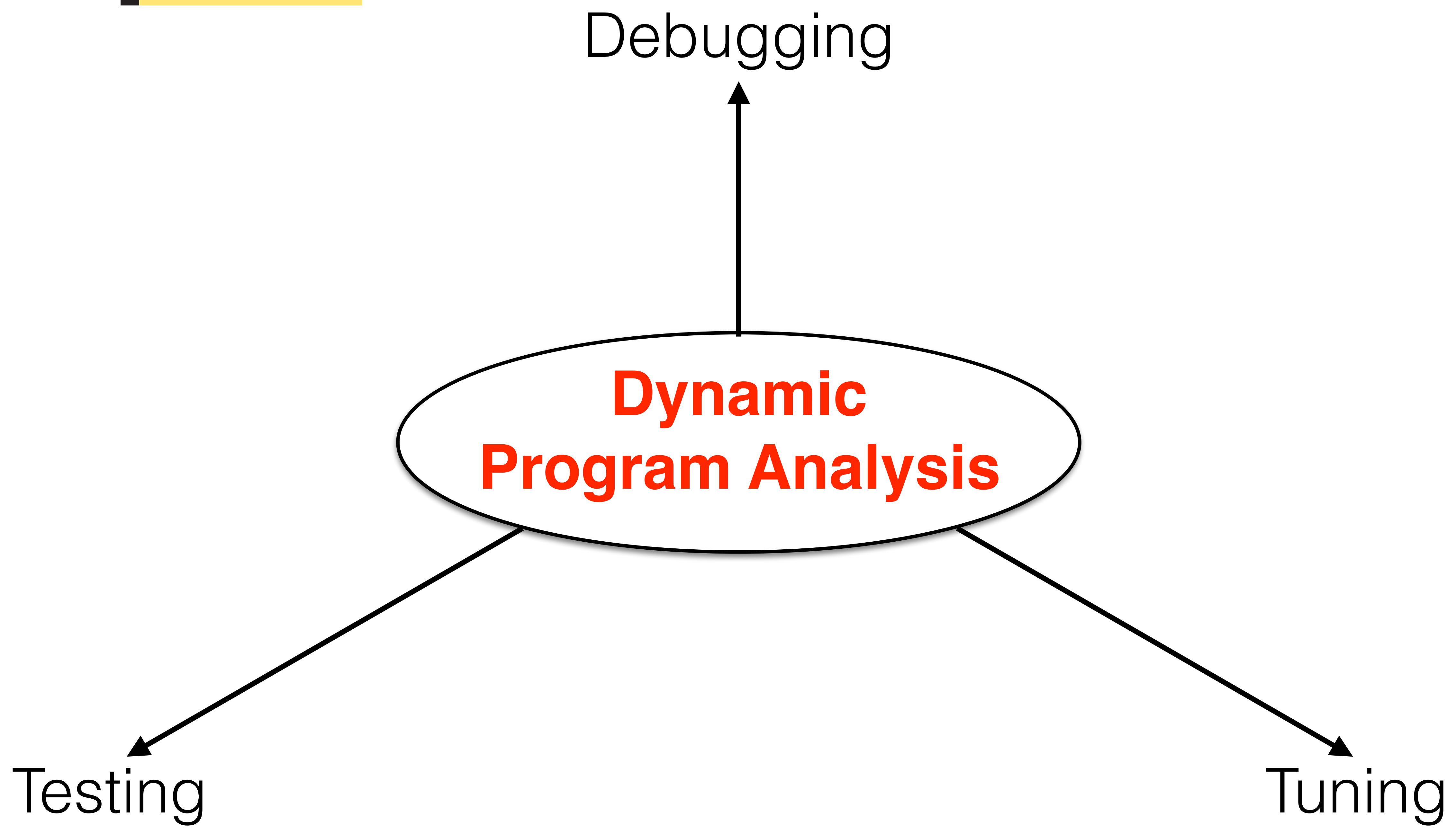
DiSL targets ...



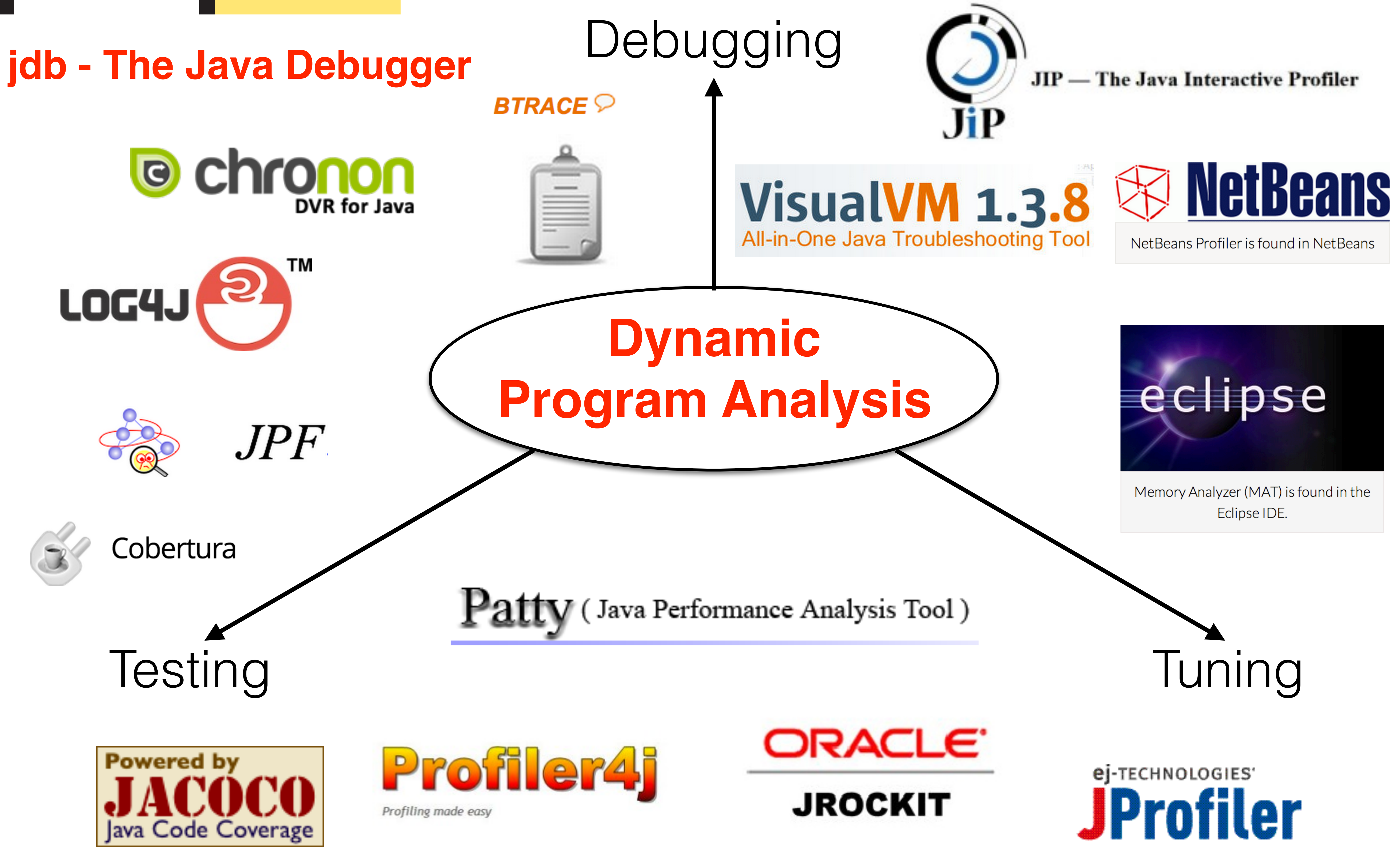
DiSL targets ...



DiSL targets ...



DiSL targets ...


jdb - The Java Debugger
BTRACE
Debugging
JIP — The Java Interactive Profiler
JiP
chronon DVR for Java
VisualVM 1.3.8
All-in-One Java Troubleshooting Tool
NetBeans
NetBeans Profiler is found in NetBeans
LOG4J
Dynamic Program Analysis
eclipse
Memory Analyzer (MAT) is found in the Eclipse IDE.
JPF
Cobertura
Patty (Java Performance Analysis Tool)
Testing
Tuning
Powered by JACOCO Java Code Coverage
Profiler4j Profiling made easy
ORACLE JROCKIT
ej-TECHNOLOGIES JProfiler

TALK IS CHEAP.
SHOW ME THE CODE.

DiSL code

DiSL code

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  System.out.println("Hello, DiSL");
}
```

DiSL code

Annotation: describes ***where*** *to instrument*

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  System.out.println("Hello, DiSL");
}
```

DiSL code

*Annotation: describes **where** to instrument*

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  System.out.println("Hello, DiSL");
}
```

*Method Body: describes **what** to instrument*

DiSL code

Annotation: describes ***where*** *to instrument*

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  System.out.println("Hello, DiSL");
}
```

Method Body: describes ***what*** *to instrument*

>_
Demo

How **DiSL** instruments

How **DiSL** instruments

```
                Helloworld.java

public class Helloworld {
  public static void main(String[] args) {
    System.out.println("Hello, world");
  }
}
```

← Target Class

How DiSL instruments

DiSL
Class →

HelloDiSL.java

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  System.out.println("Hello, DiSL");
}
```

Helloworld.java

```
public class Helloworld {
  public static void main(String[] args) {
    System.out.println("Hello, world");
  }
}
```

← Target
Class

How DiSL instruments

1. Identify the instrumentation locations →

HelloDiSL.java

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  System.out.println("Hello, DiSL");
}
```

Helloworld.java

```
public class Helloworld {
  public static void main(String[] args) {
    System.out.println("Hello, world");
  }
}
```

How DiSL instruments

2. Extract the method body →

```
System.out.println("Hello, DiSL");
```

```
            Helloworld.java

public class Helloworld {
  public static void main(String[] args) {
    System.out.println("Hello, world");
  }
}
```

How **DiSL** instruments

```
                 Helloworld.java

public class Helloworld {
  public static void main(String[] args) {
    System.out.println("Hello, DiSL");
    System.out.println("Hello, world");
  }
}
```

← **3. Instrument**

Approach

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Approach

```
> java -cp bin-target demo1.Helloworld
```

Approach

```
> java -cp bin-target demo1.Helloworld
```

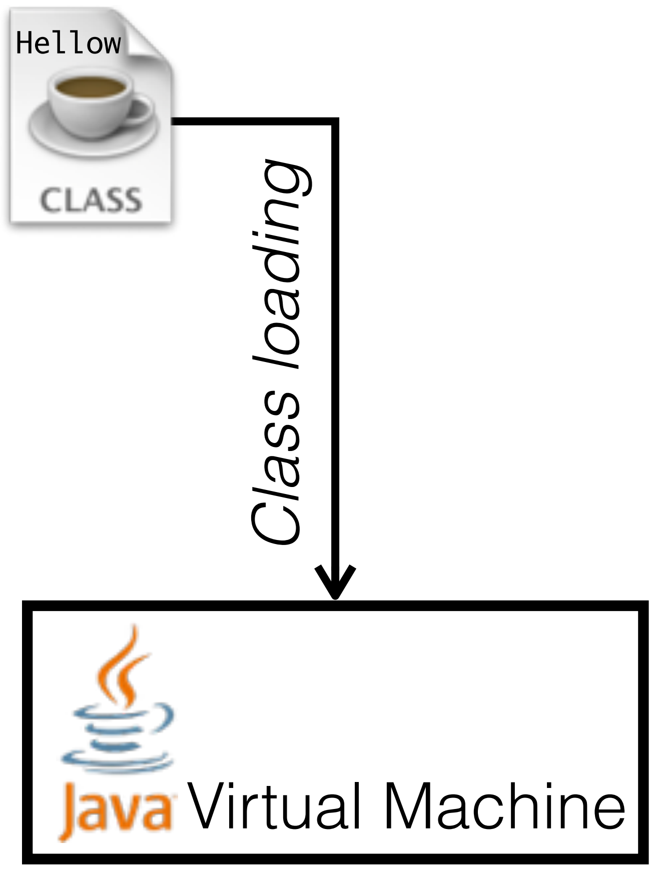
Approach

```
> java -cp bin-target demo1.Helloworld
```

Hellow
CLASS
Java Virtual Machine

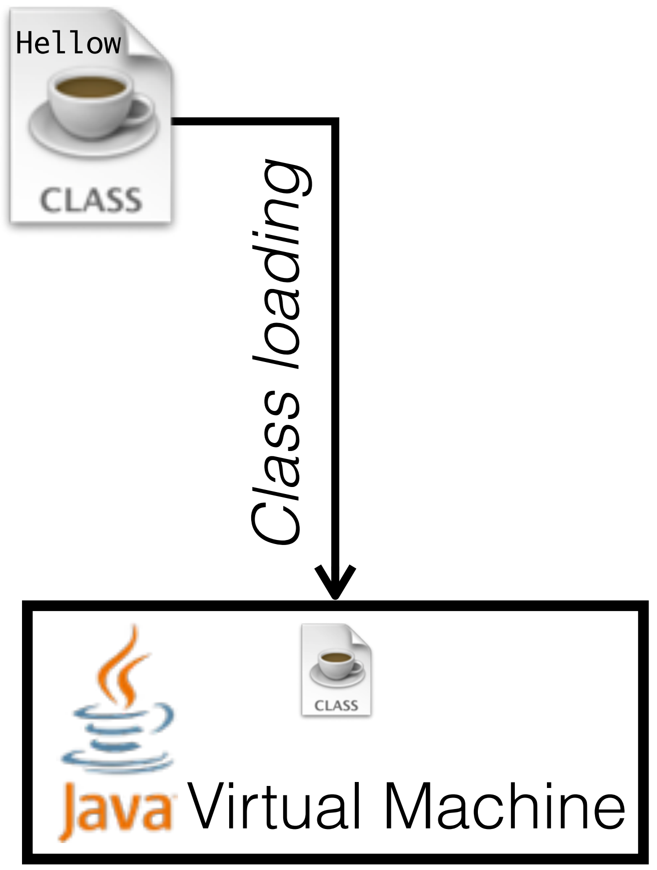
Approach

```
> java -cp bin-target demo1.Helloworld
```



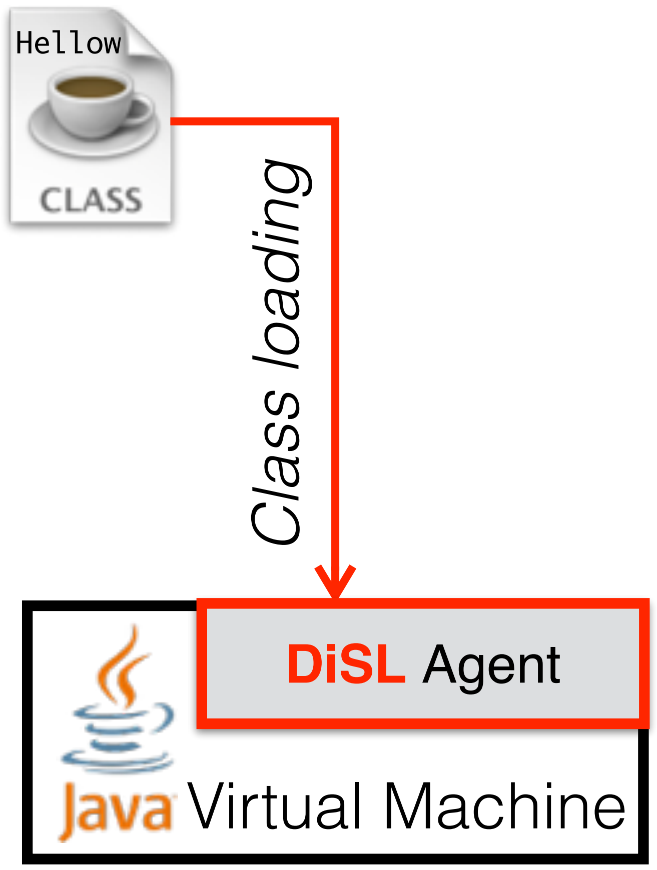
Approach

```
> java -cp bin-target demo1.Helloworld
```



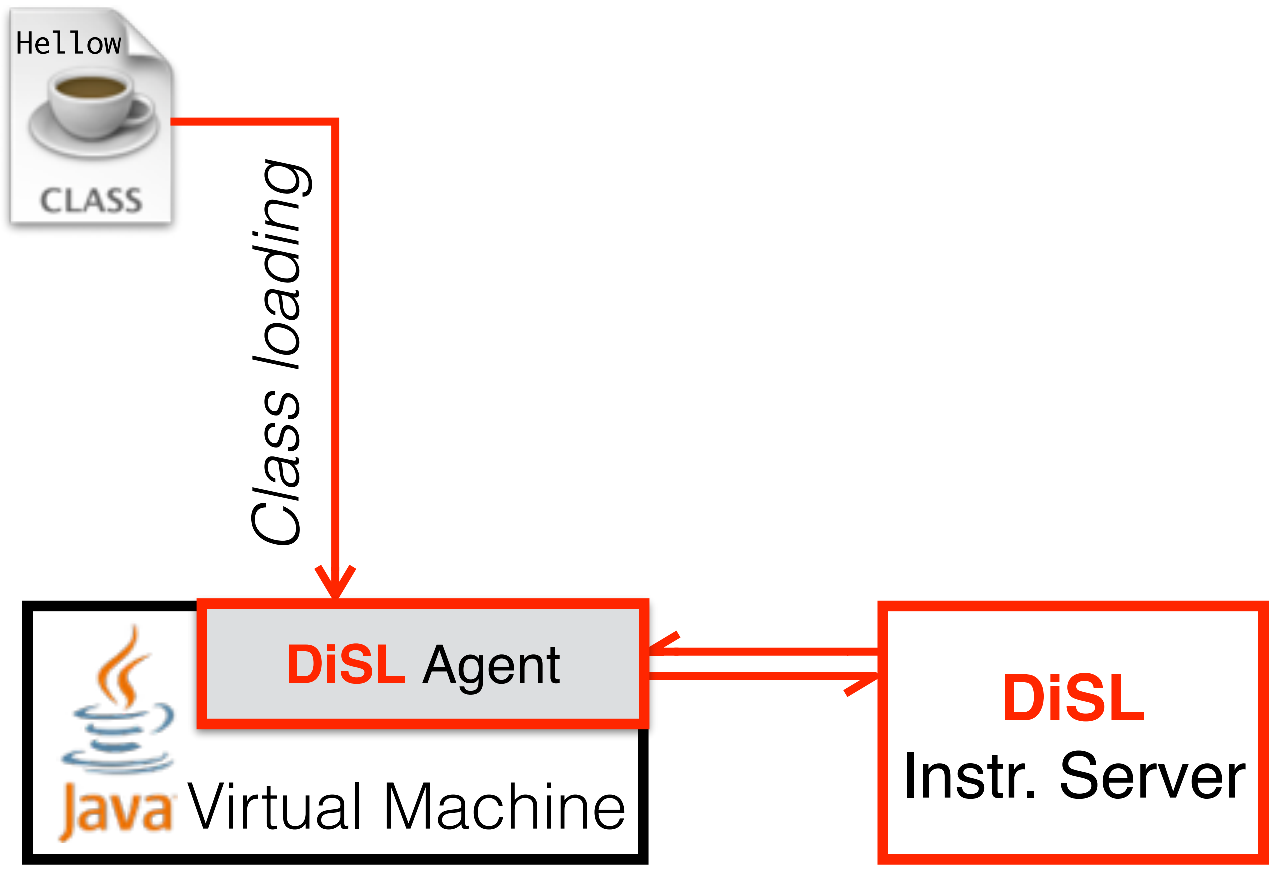
Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



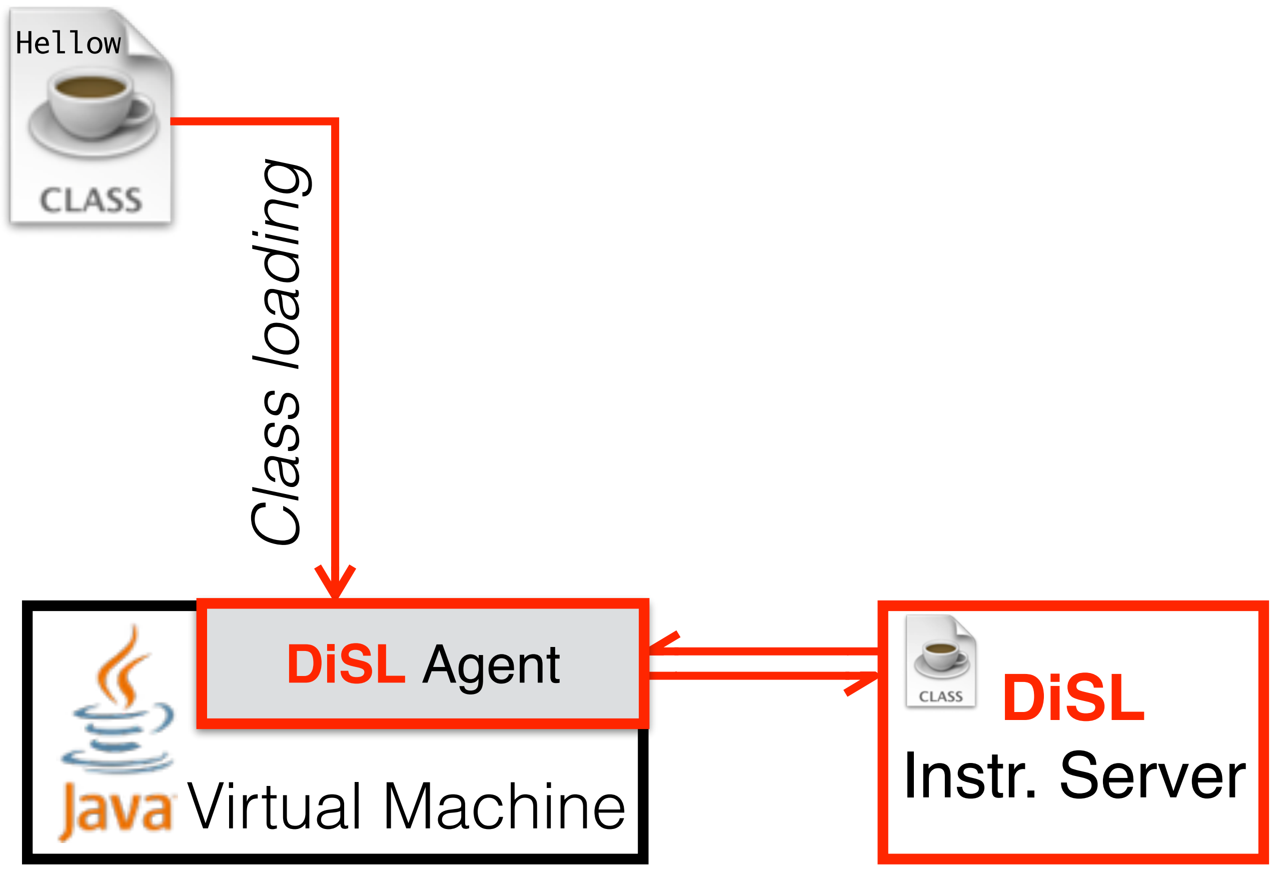
Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



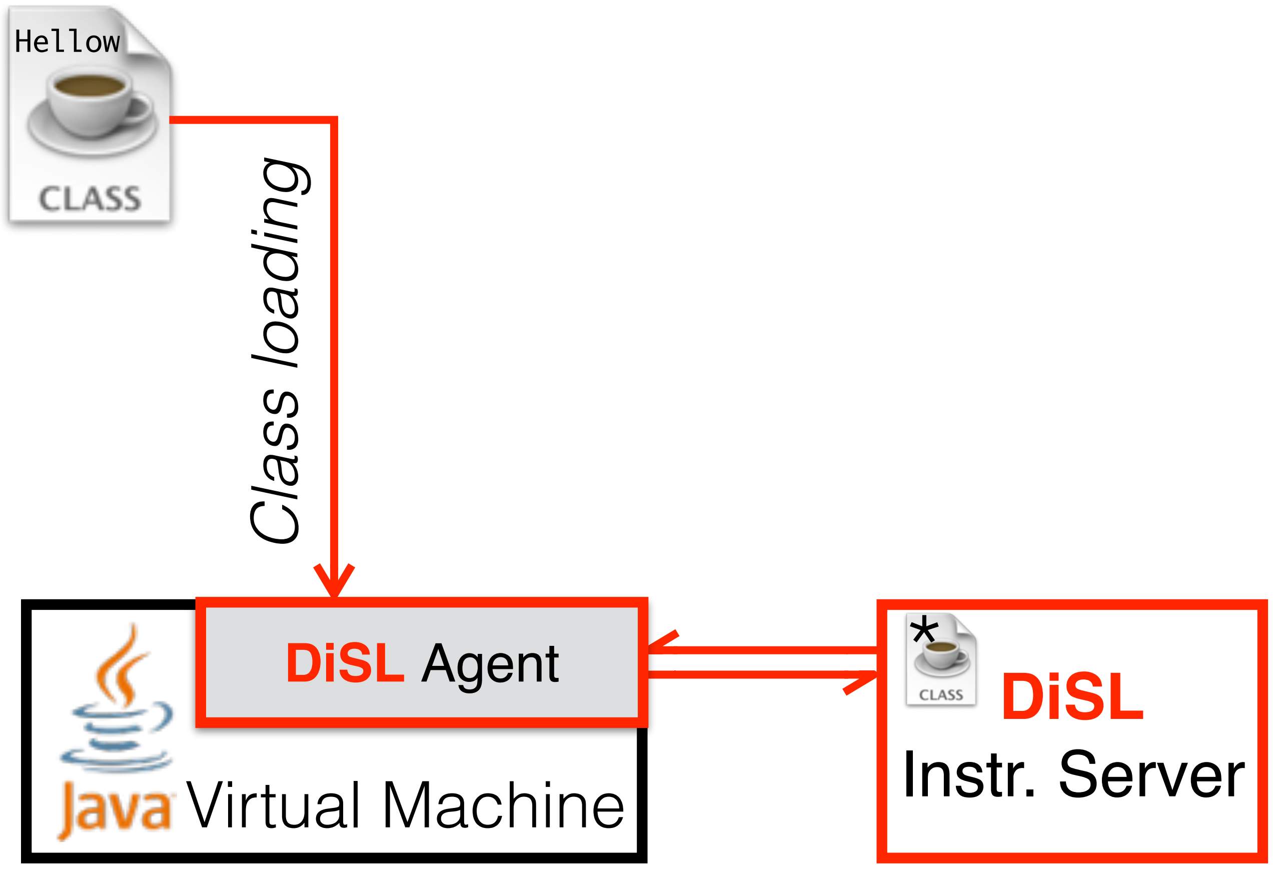
Approach

```
> java -agentpath:lib/libdislagent.jnilib
         -Xbootclasspath/a:<...>
         -cp bin-target demo1.Helloworld
```



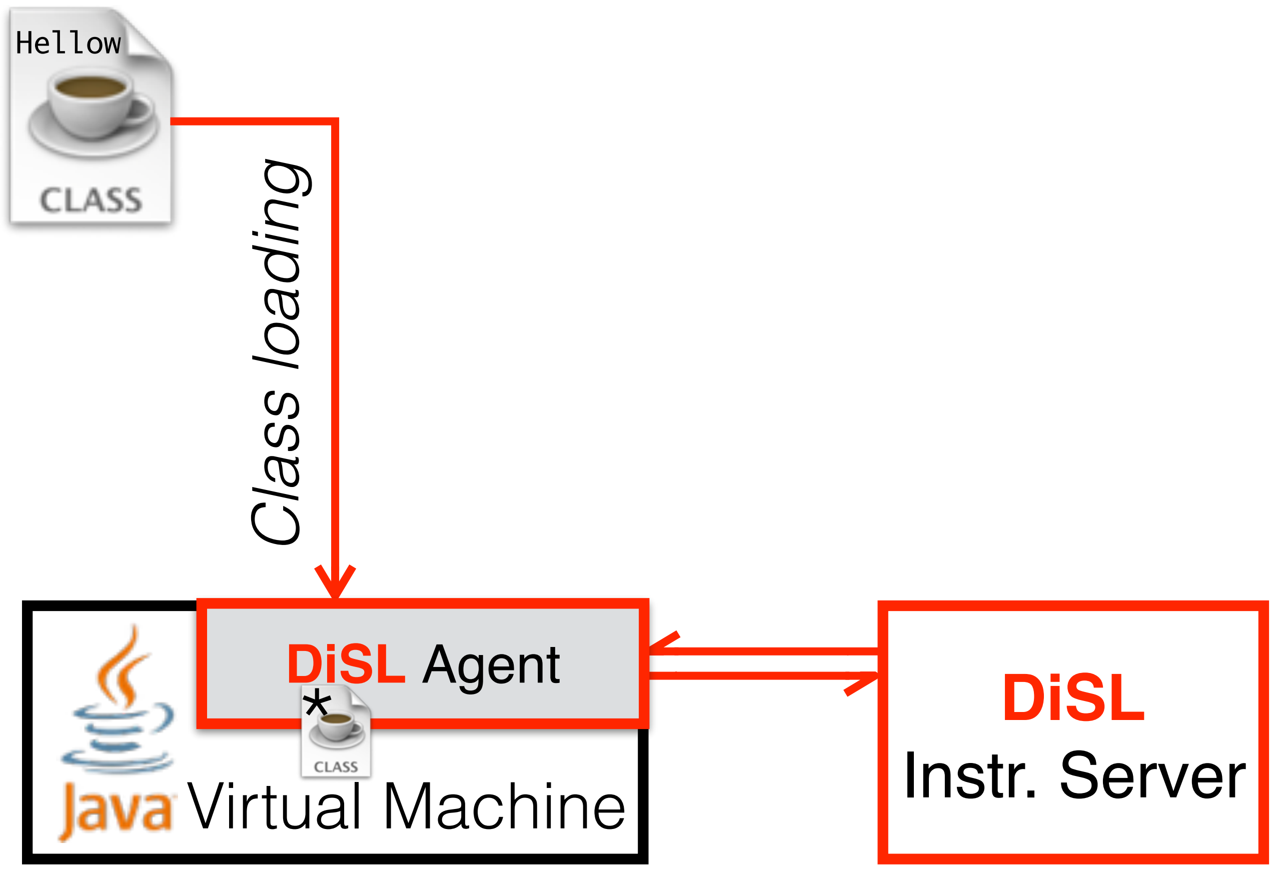
Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



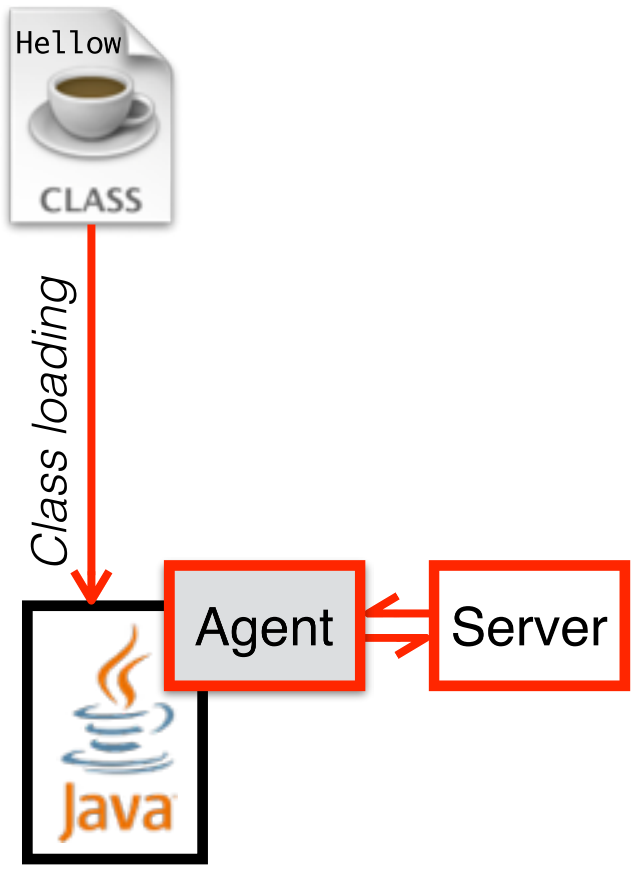
Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



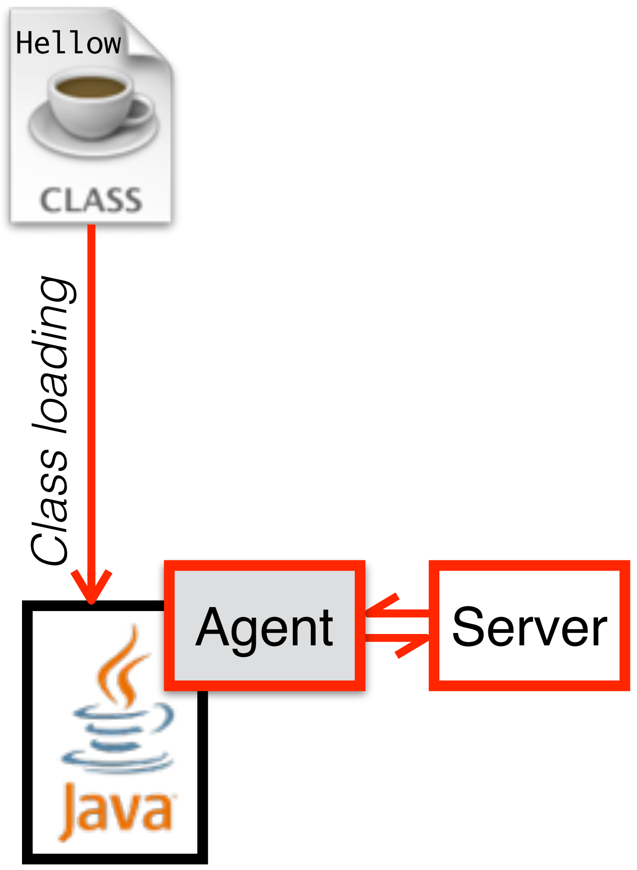
Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



Approach

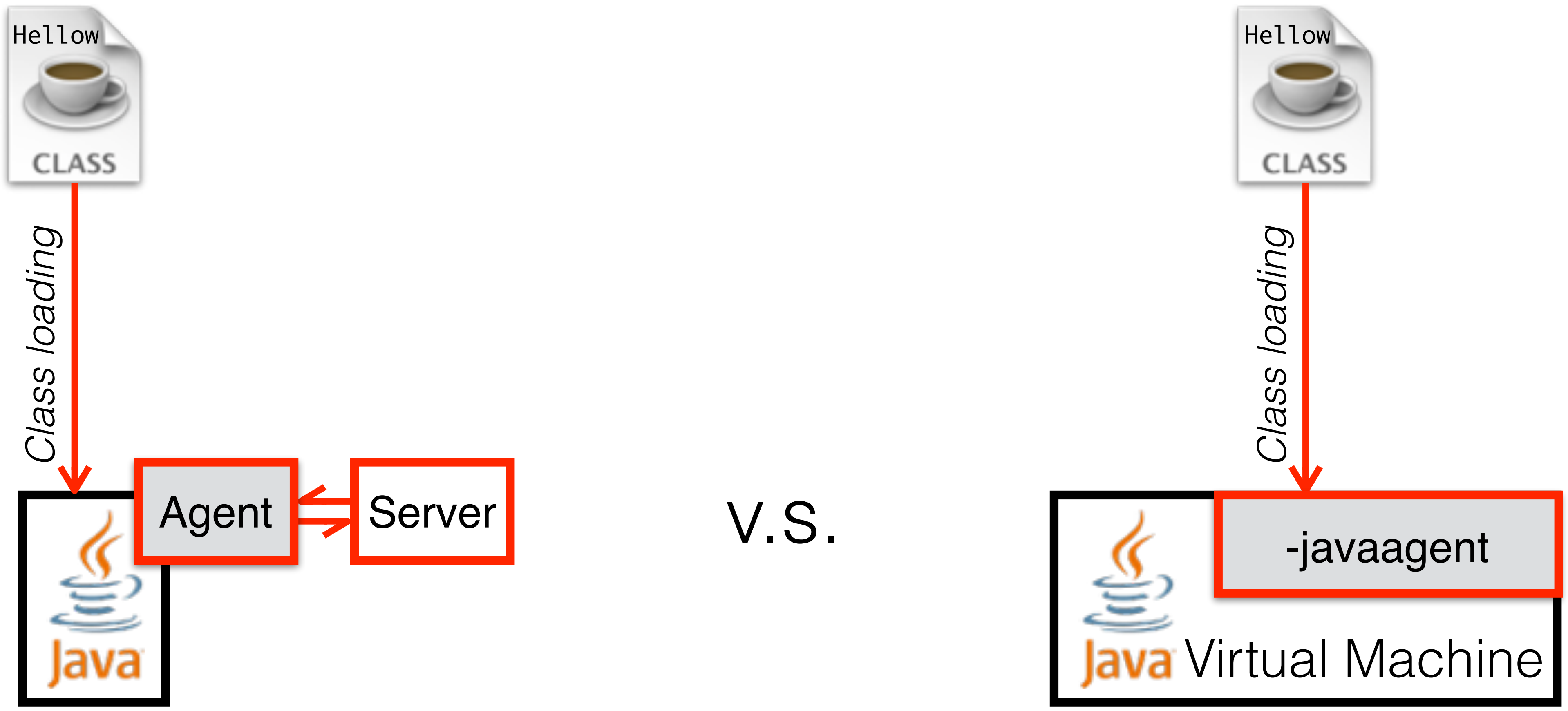
```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



V.S.

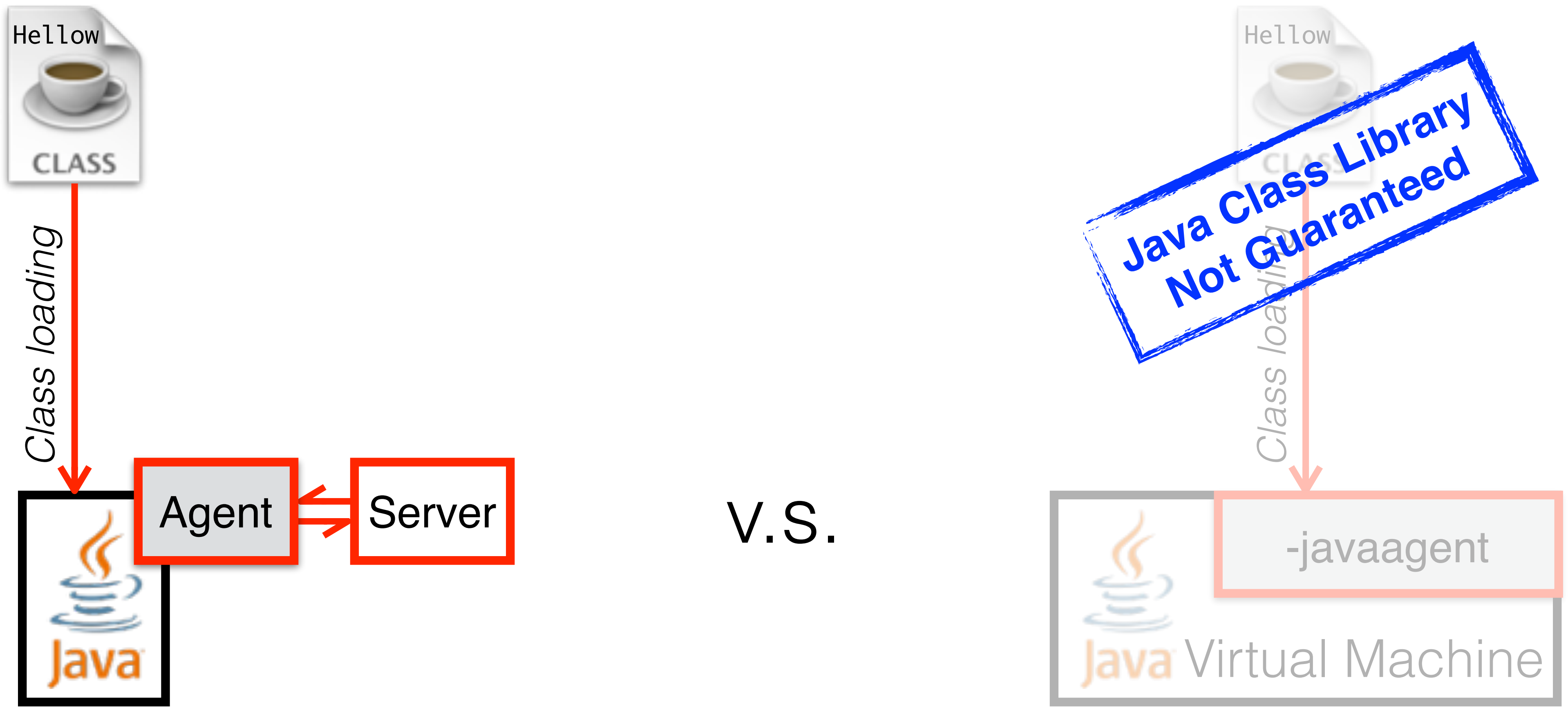
Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



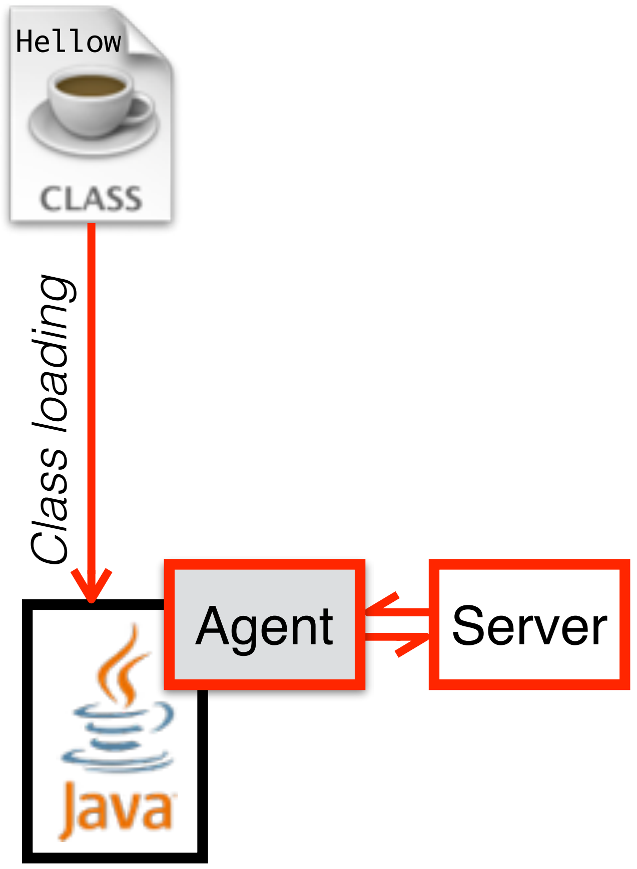
Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```

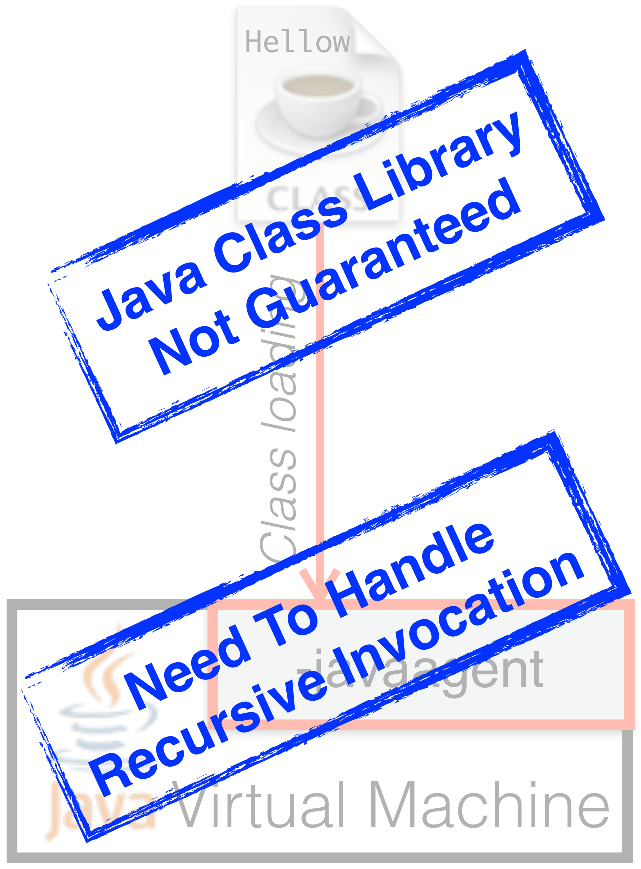


Approach

```
> java -agentpath:lib/libdislagent.jnilib
       -Xbootclasspath/a:<...>
       -cp bin-target demo1.Helloworld
```



V.S.


Hellow
CLASS
Class loading
javaagent
Java Virtual Machine
Java Class Library Not Guaranteed
Need To Handle Recursive Invocation

Recursive Invocation

Recursive Invocation

```
@Before(marker = BodyMarker.class,
        scope = "PrintStream.println")
static void premain() {
   System.out.println("Hello, DiSL");
}
```

Recursive Invocation

```
@Before(marker = BodyMarker.class,
        scope = "PrintStream.println")
static void premain() {
   System.out.println("Hello, DiSL");
}
```

Hint: System.out is an instance of PrintStream

Recursive Invocation

```
@Before(marker = BodyMarker.class,
        scope = "PrintStream.println")
static void premain() {
   System.out.println("Hello, DiSL");
}
```

Hint: System.out is an instance of PrintStream

DiSL addresses the issue automatically!

Recursive Invocation

```
@Before(marker = BodyMarker.class,
        scope = "PrintStream.println")
static void premain() {
   System.out.println("Hello, DiSL");
}
```

Hint: System.out is an instance of PrintStream

DiSL addresses the issue automatically!

>_
Demo

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Example: mvn

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Example: mvn

- Background: in a recent project, we try to measure the execution of the maven unit tests.

Example: mvn

- Background: in a recent project, we try to measure the execution of the maven unit tests.

- Funny facts:

Example: mvn

- Background: in a recent project, we try to measure the execution of the maven unit tests.
- Funny facts:
  - `mvn` runs on the Java Virtual Machine

Example: mvn

- Background: in a recent project, we try to measure the execution of the maven unit tests.
- Funny facts:
  - `mvn` runs on the Java Virtual Machine
    - ✓ we can apply **DiSL**

Example: mvn

- Background: in a recent project, we try to measure the execution of the maven unit tests.
- Funny facts:
  - `mvn` runs on the Java Virtual Machine
    - ✓ we can apply **DiSL**
  - `mvn test` runs on yet another JVM

Example: mvn

- Background: in a recent project, we try to measure the execution of the maven unit tests.

- Funny facts:

  - `mvn` runs on the Java Virtual Machine

    ✓ we can apply **DiSL**

  - `mvn test` runs on yet another JVM

    Q: how do I know that?

Example: mvn

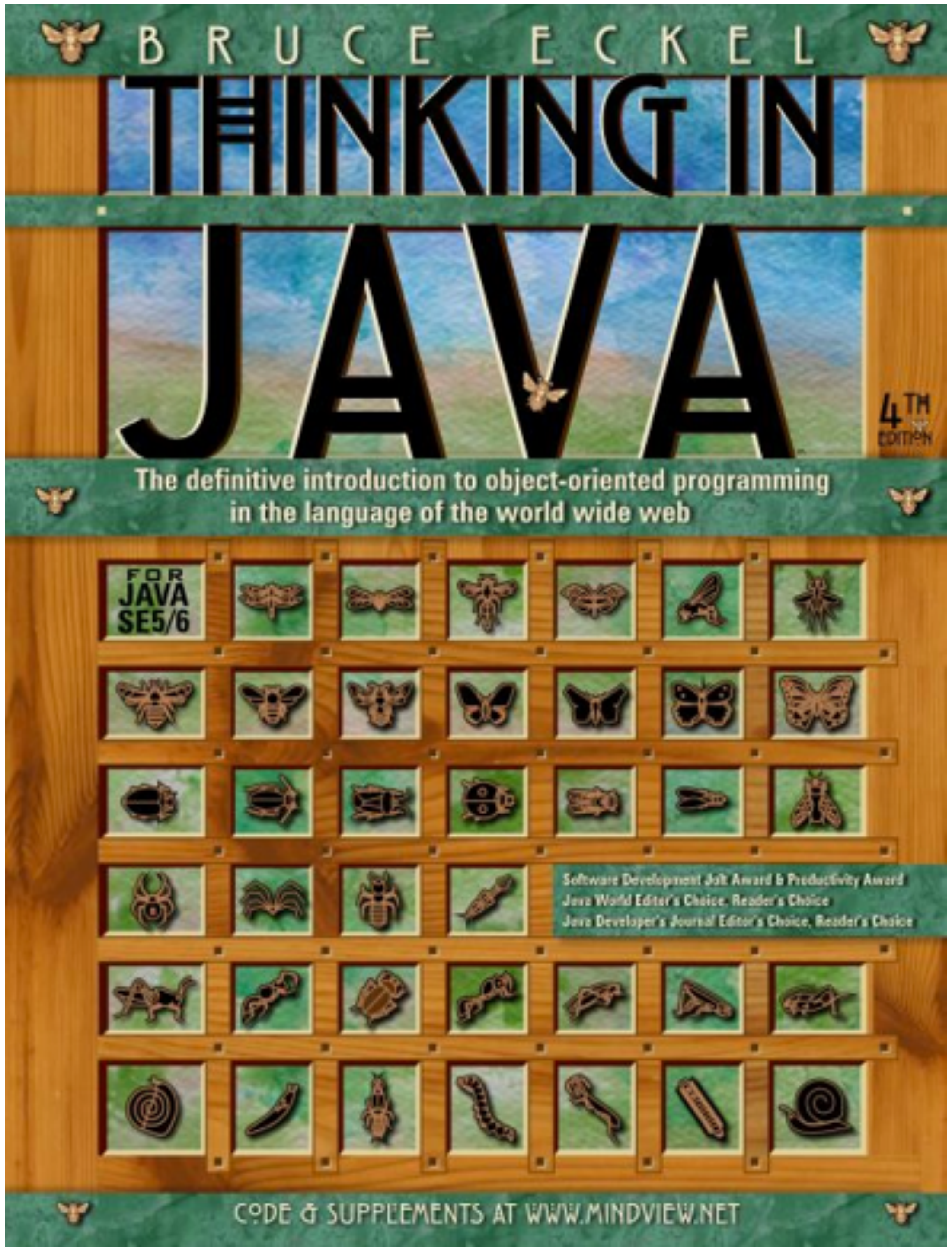
- Background: in a recent project, we try to measure the execution of the maven unit tests.

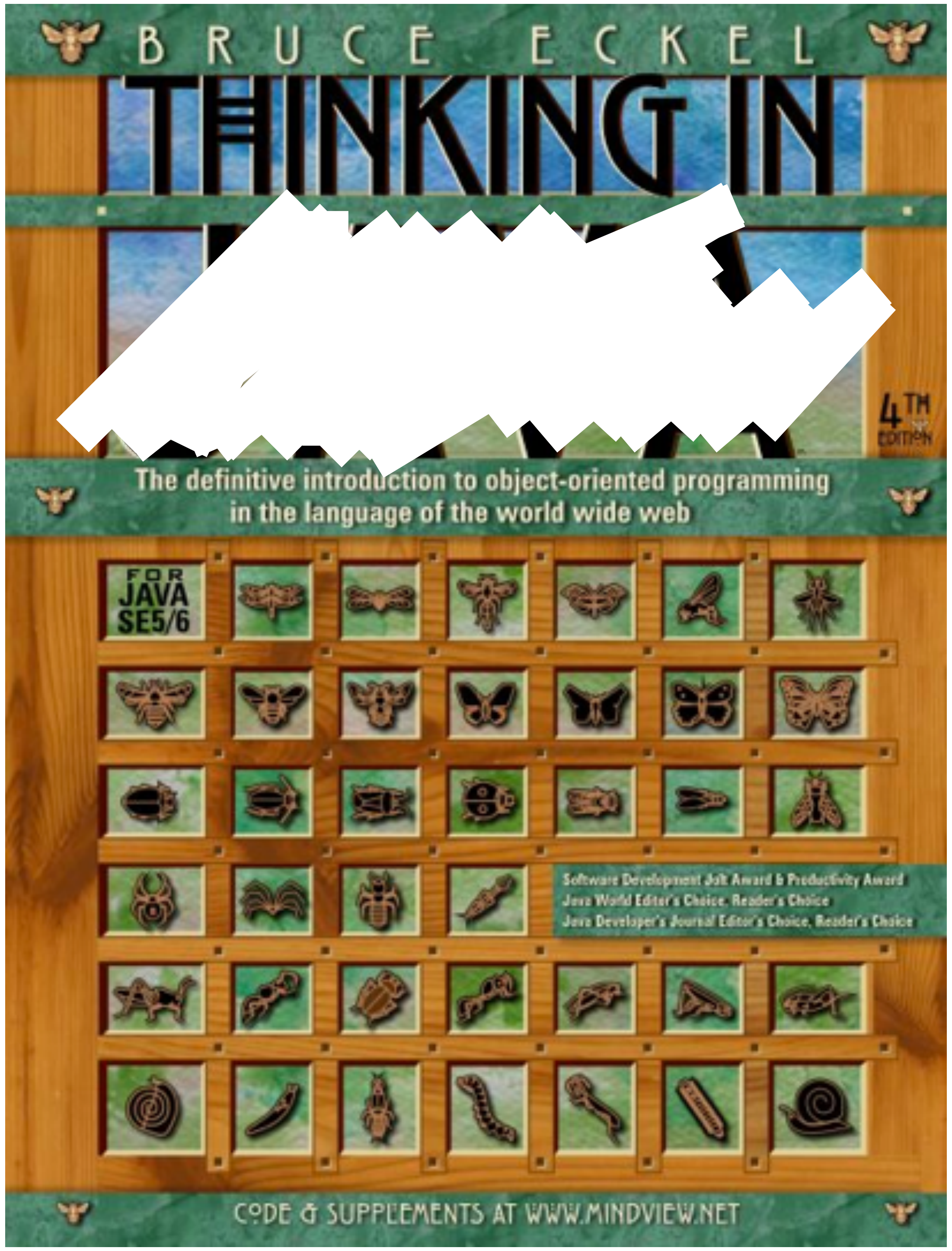
- Funny facts:
  - `mvn` runs on the Java Virtual Machine
    - ✓ we can apply **DiSL**
  - `mvn test` runs on yet another JVM
    - Q: how do I know that?
    - Q: how do I confirm that?

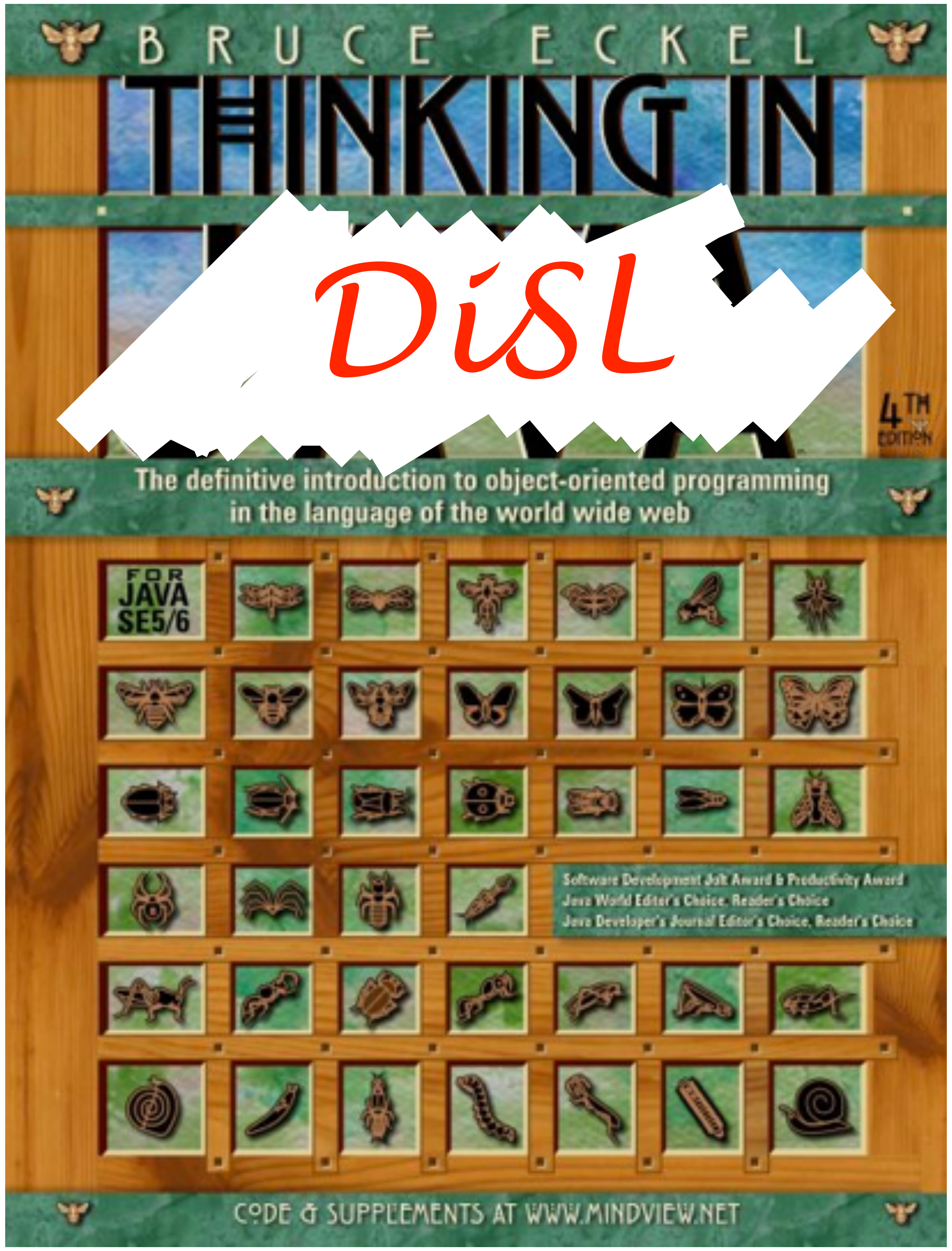
Example: mvn

- Background: in a recent project, we try to measure the execution of the maven unit tests.
- Funny facts:
  - `mvn` runs on the Java Virtual Machine
    - ✓ we can apply **DiSL**
  - `mvn test` runs on yet another JVM
    - Q: how do I know that?
    - Q: how do I confirm that?

>_ Demo


BRUCE ECKEL
THINKING IN JAVA
4TH EDITION
The definitive introduction to object-oriented programming in the language of the world wide web
FOR JAVA SE5/6
Software Development Jolt Award & Productivity Award
Java World Editor's Choice, Reader's Choice
Java Developer's Journal Editor's Choice, Reader's Choice
CODE & SUPPLEMENTS AT WWW.MINDVIEW.NET


BRUCE ECKEL
THINKING IN
4TH EDITION
The definitive introduction to object-oriented programming in the language of the world wide web
FOR JAVA SE5/6
Software Development Jolt Award & Productivity Award
Java World Editor's Choice, Reader's Choice
Java Developer's Journal Editor's Choice, Reader's Choice
CODE & SUPPLEMENTS AT WWW.MINDVIEW.NET


BRUCE ECKEL
THINKING IN
DiSL
4TH EDITION
The definitive introduction to object-oriented programming in the language of the world wide web
FOR JAVA SE5/6
Software Development Jolt Award & Productivity Award
Java World Editor's Choice, Reader's Choice
Java Developer's Journal Editor's Choice, Reader's Choice
CODE & SUPPLEMENTS AT WWW.MINDVIEW.NET

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Thinking in **DiSL**

Thinking in DiSL

1. **Event Producer/Consumer**

1. Event Producer/Consumer

Target Code

```
public static void main(String[] args) {
  System.out.println("Hello, world");
}
```

Thinking in DiSL

1. Event Producer/Consumer

Target Code

```
public static void main(String[] args) {
  System.out.println("Hello, world");
}
```

DiSL Code

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  System.out.println("Hello, DiSL");
}
```

Thinking in DiSL

1. **Event Producer/Consumer**

Target Code

```
public static void main(String[] args) {
  System.out.println("Hello, world");
}
```

DiSL Code

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  EventConsumer.firePremainEvent();
}
```

Thinking in **DiSL**

1. Event Producer/Consumer

Target Code

DiSL Code

```
@Before(marker = BodyMarker.class,
        scope = "Helloworld.main")
static void premain() {
  EventConsumer.firePremainEvent();
}
```

Runtime Analysis

```
public class EventConsumer {
  public static void
           firePremainEvent() {
      ...
   }
}
```

1. Event Producer/Consumer

Target Code

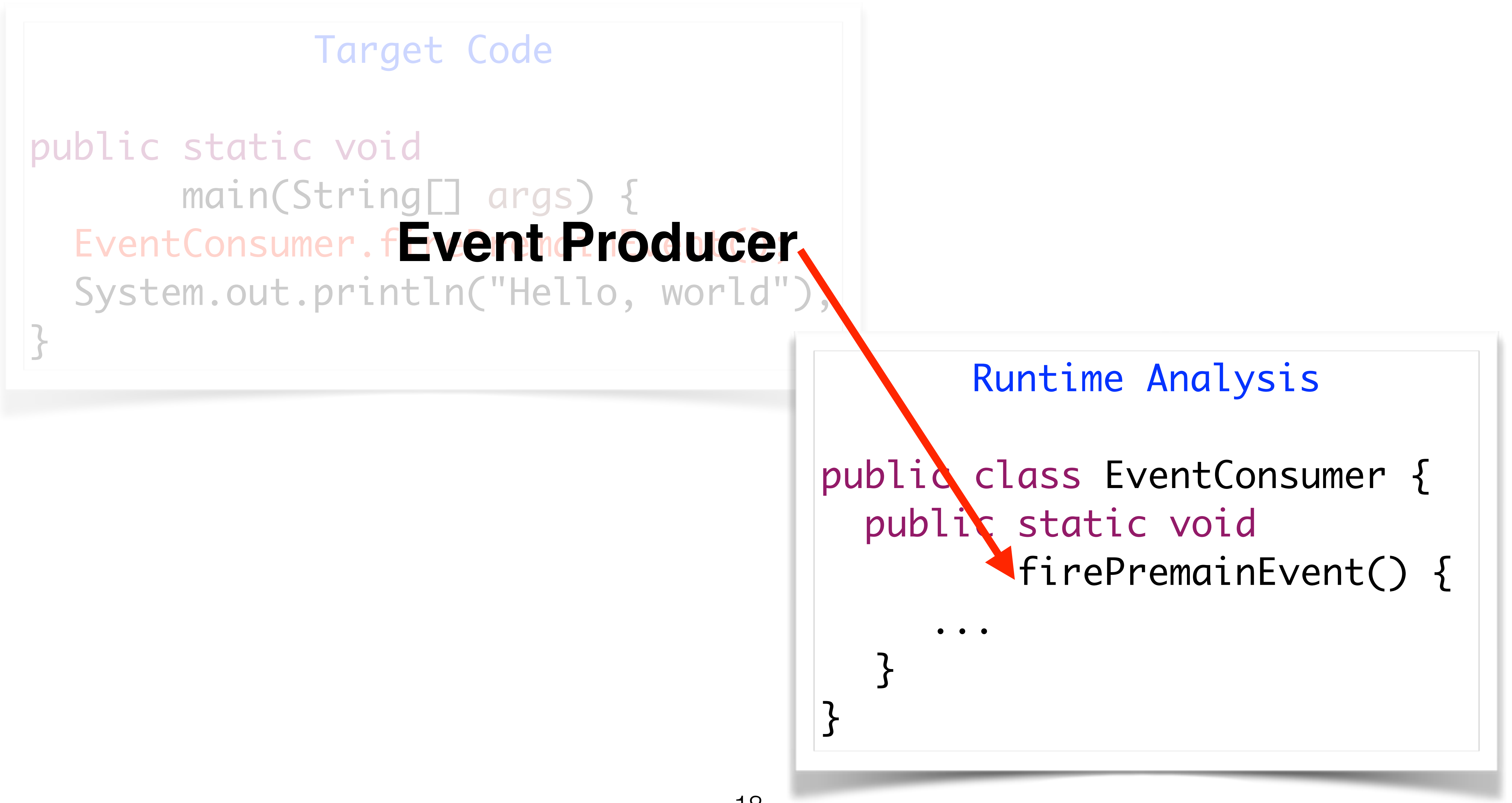
```
public static void
       main(String[] args) {
  EventConsumer.firePremainEvent();
  System.out.println("Hello, world");
}
```

Runtime Analysis

```
public class EventConsumer {
  public static void
         firePremainEvent() {
     ...
  }
}
```

Thinking in **DiSL**

1. **Event Producer/Consumer**



Target Code

```
public static void
      main(String[] args) {
  EventConsumer.firePremainEvent();
  System.out.println("Hello, world");
}
```

Runtime Analysis

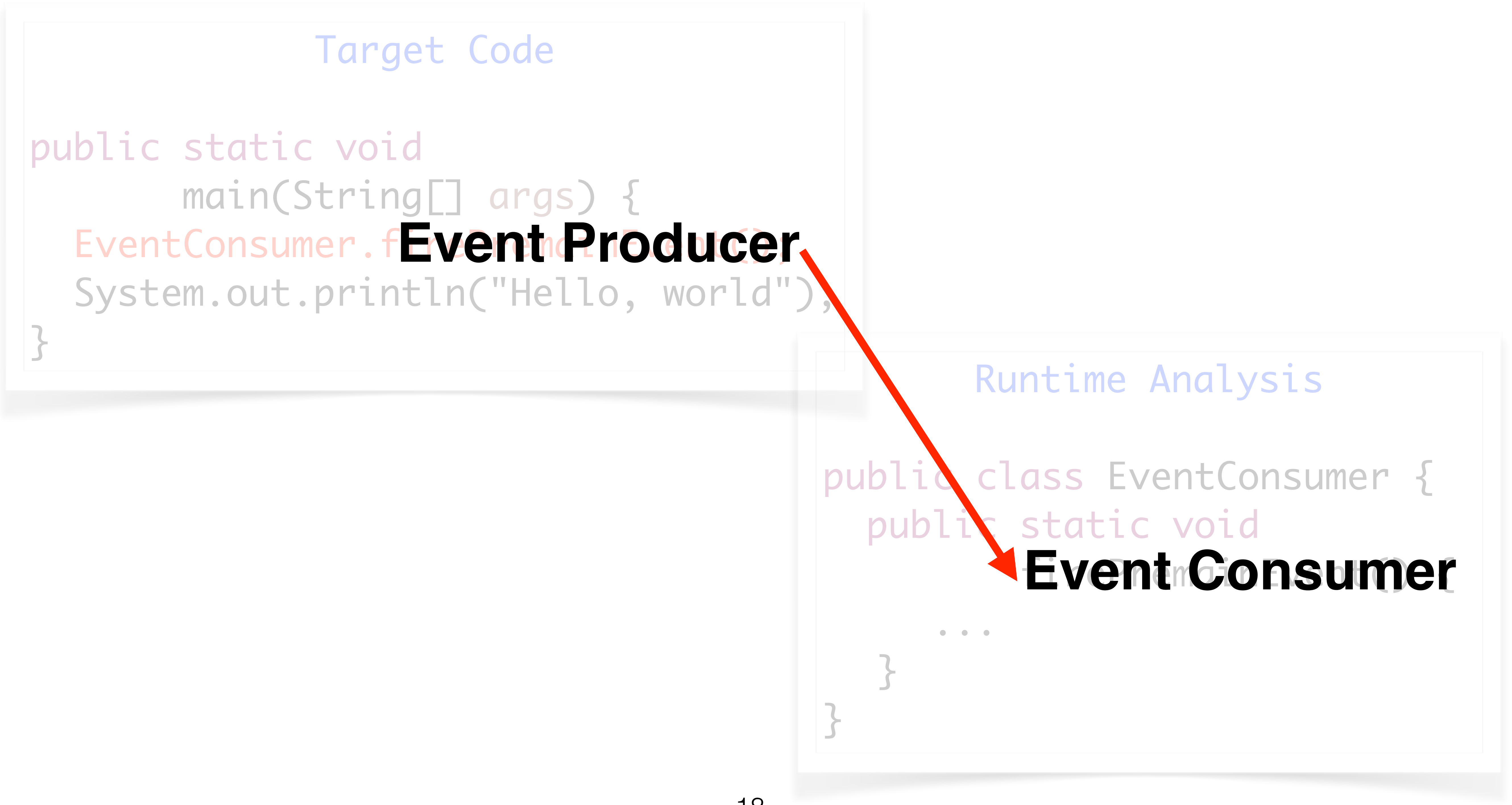
```
public class EventConsumer {
  public static void
         firePremainEvent() {
      ...
   }
}
```

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Thinking in DiSL

1. Event Producer/Consumer

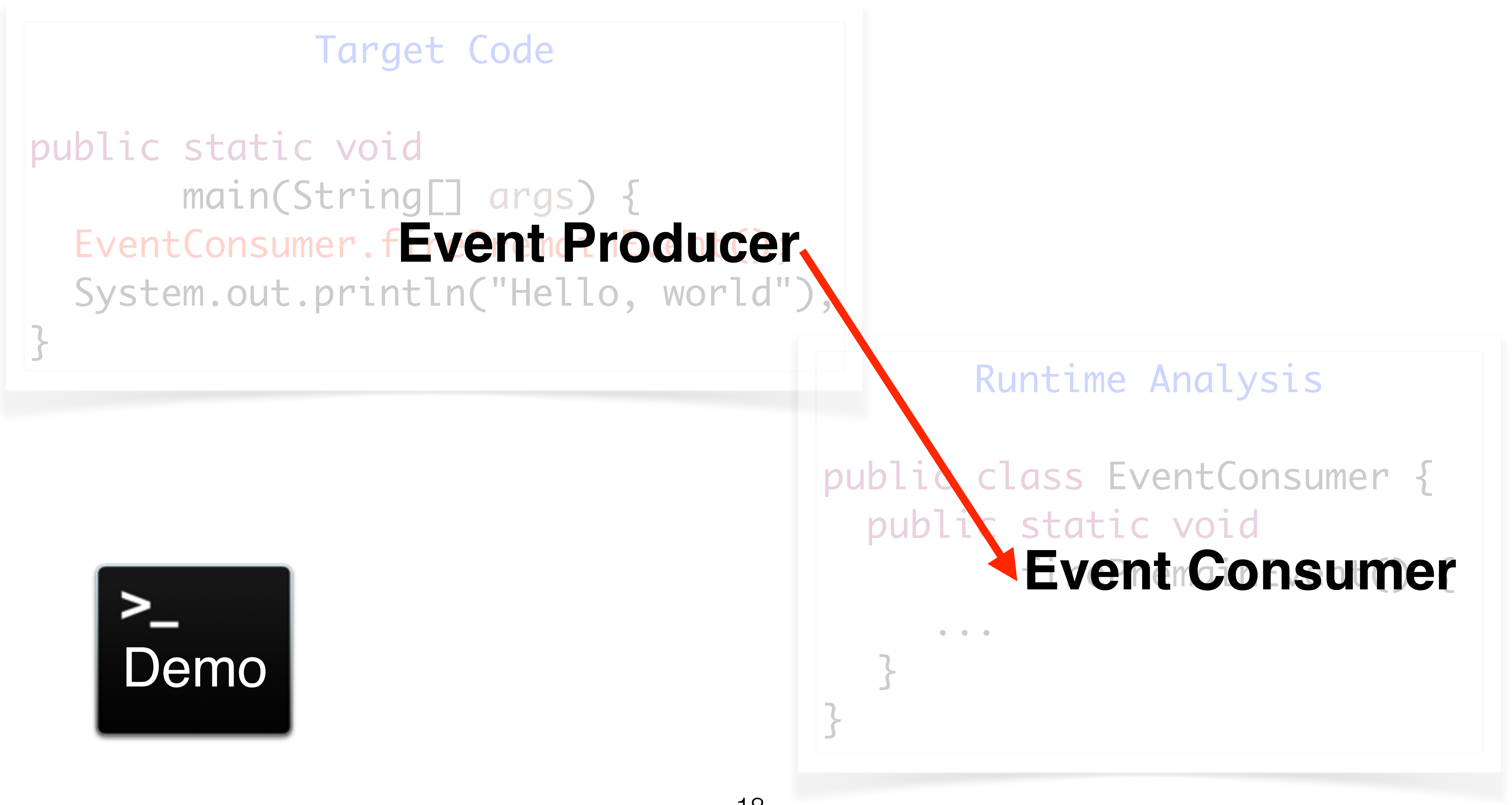


```
public static void
      main(String[] args) {
  EventConsumer.fireDomainEvent();
  System.out.println("Hello, world");
}
```

```
public class EventConsumer {
  public static void
        fireDomainEvent() {
     ...
  }
}
```

Thinking in **DiSL**

1. Event Producer/Consumer



Target Code

```
public static void
      main(String[] args) {
  EventConsumer.firePreMainEvent();
  System.out.println("Hello, world");
}
```

Runtime Analysis

```
public class EventConsumer {
  public static void
          firePreMainEvent() {
     ...
  }
}
```

>_
Demo

Università della Svizzera italiana

Faculty of Informatics

Example: Allocation Profiler

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)
  - **[Producer]** right after each **new** bytecode, emit an event indicating the occurrence of the allocation

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)
    - **[Producer]** right after each **new** bytecode, emit an event indicating the occurrence of the allocation
    - **[Consumer]** maintain a counter, increment it when receiving events

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)
  - **[Producer]** right after each **new** bytecode, emit an event indicating the occurrence of the allocation
  - **[Consumer]** maintain a counter, increment it when receiving events

>_
Demo

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)
  - **[Producer]** right after each **new** bytecode, emit an event indicating the occurrence of the allocation
  - **[Consumer]** maintain a counter, increment it when receiving events
- Benefits

>_
Demo

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)
  - **[Producer]** right after each **new** bytecode, emit an event indicating the occurrence of the allocation
  - **[Consumer]** maintain a counter, increment it when receiving events
- Benefits
  - Keep the target code less inflated (JIT-related)

>_ Demo

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)
  - **[Producer]** right after each **new** bytecode, emit an event indicating the occurrence of the allocation
  - **[Consumer]** maintain a counter, increment it when receiving events
- Benefits
  - Keep the target code less inflated (JIT-related)
  - Separation of concerns

>_
Demo

Example: Allocation Profiler

- Count the number of allocations (**new** bytecode)
  - **[Producer]** right after each **new** bytecode, emit an event indicating the occurrence of the allocation
  - **[Consumer]** maintain a counter, increment it when receiving events
- Benefits
  - Keep the target code less inflated (JIT-related)
  - Separation of concerns
    - *Concurrent Event Handling*

>_
Demo

2. Concurrent Event Handling

2. Concurrent Event Handling

Target Code

```
private static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent();
  }
}
```

2. Concurrent Event Handling

Target Code

```
private static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent();
  }
}
```

Runtime Analysis

```
public class Profiler {
  public static void
         fireEvent() {
      ...
  }
}
```

2. Concurrent Event Handling

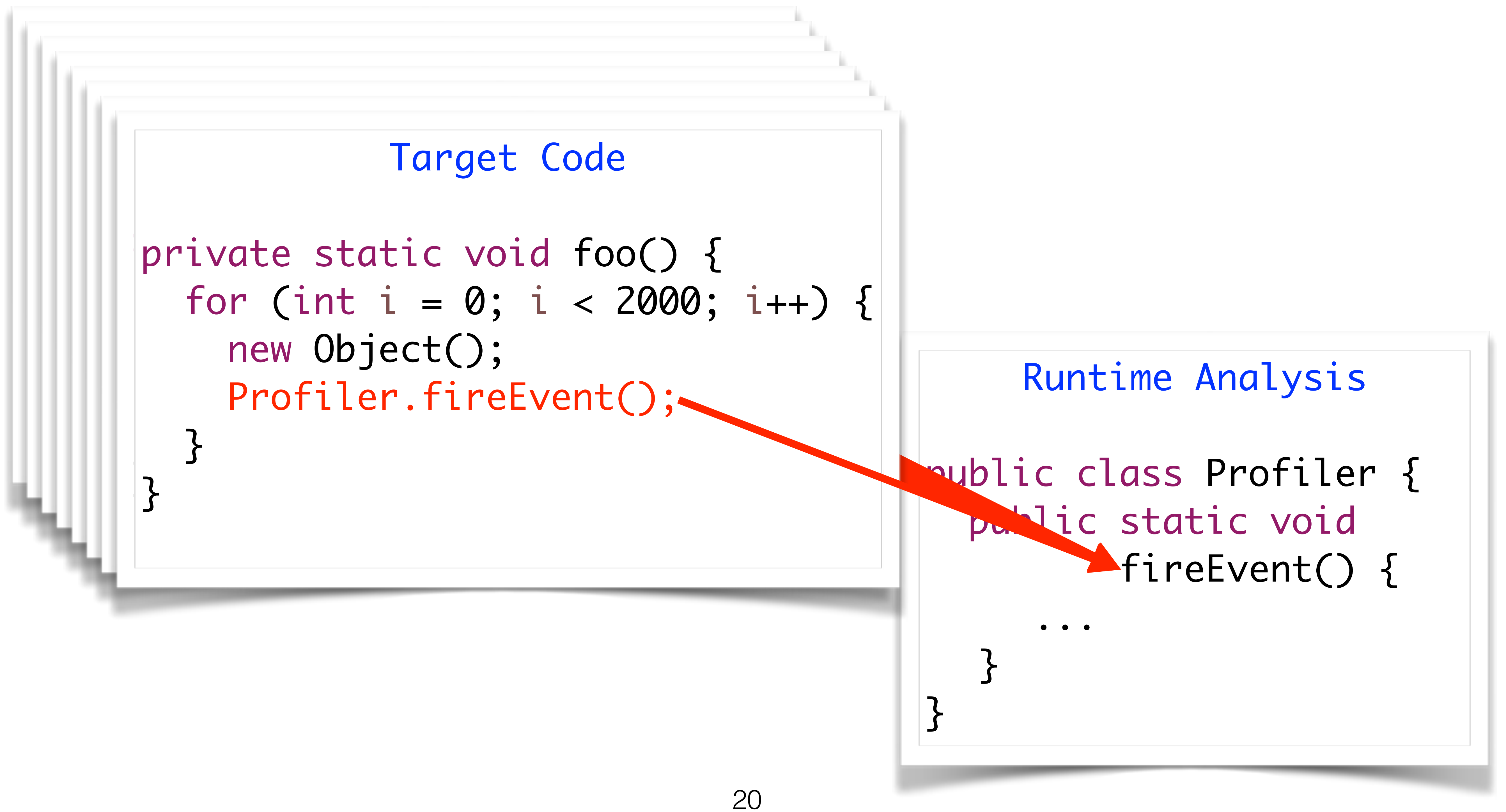
```
           Target Code

private static void foo() {
   for (int i = 0; i < 2000; i++) {
      new Object();
      Profiler.fireEvent();
   }
}
```

```
       Runtime Analysis

public class Profiler {
   public static void
               fireEvent() {
         ...
    }
}
```

2. Concurrent Event Handling



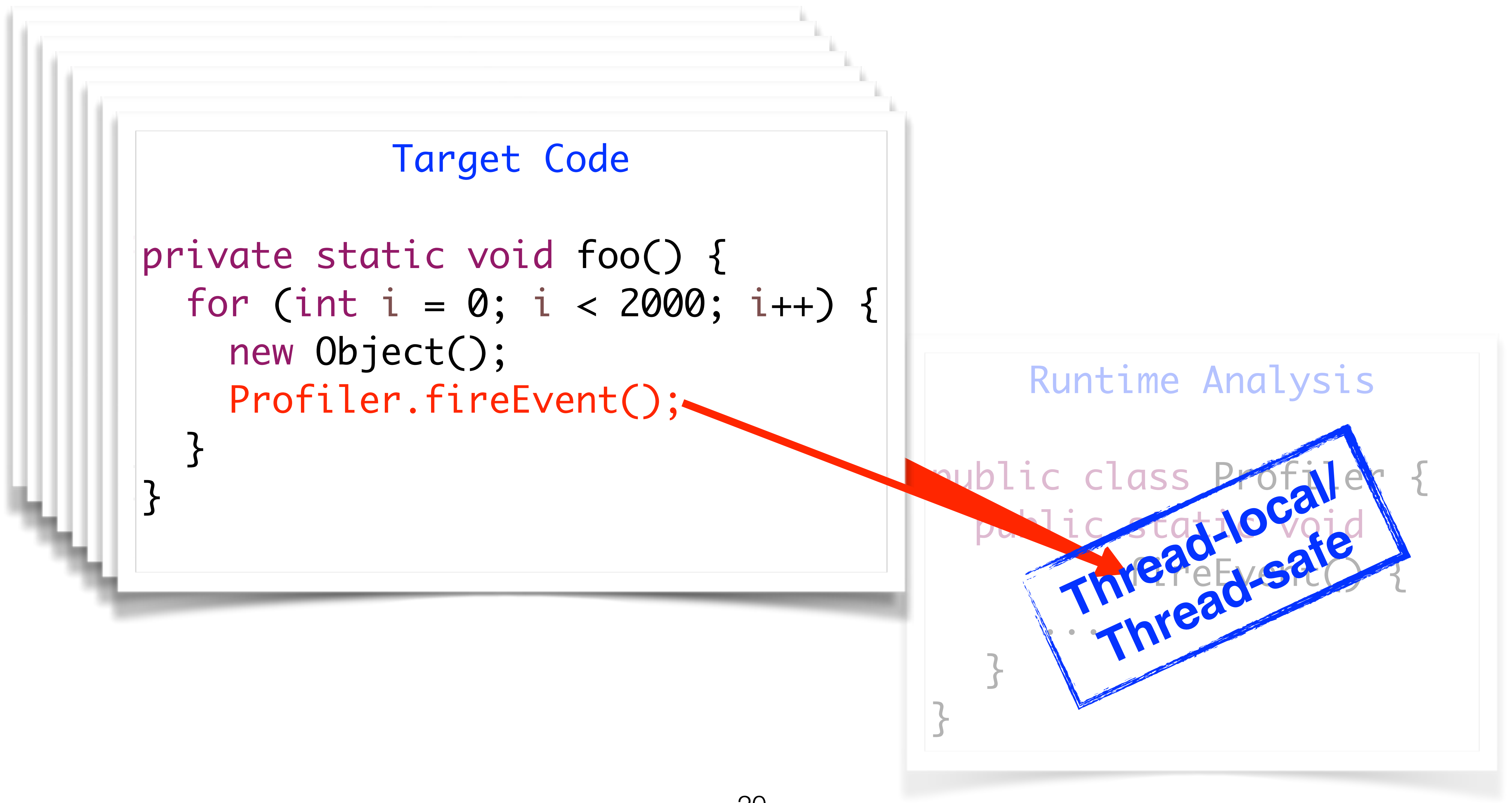
Target Code

```
private static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent();
  }
}
```

Runtime Analysis

```
public class Profiler {
  public static void
            fireEvent() {
       ...
  }
}
```

2. Concurrent Event Handling



Target Code

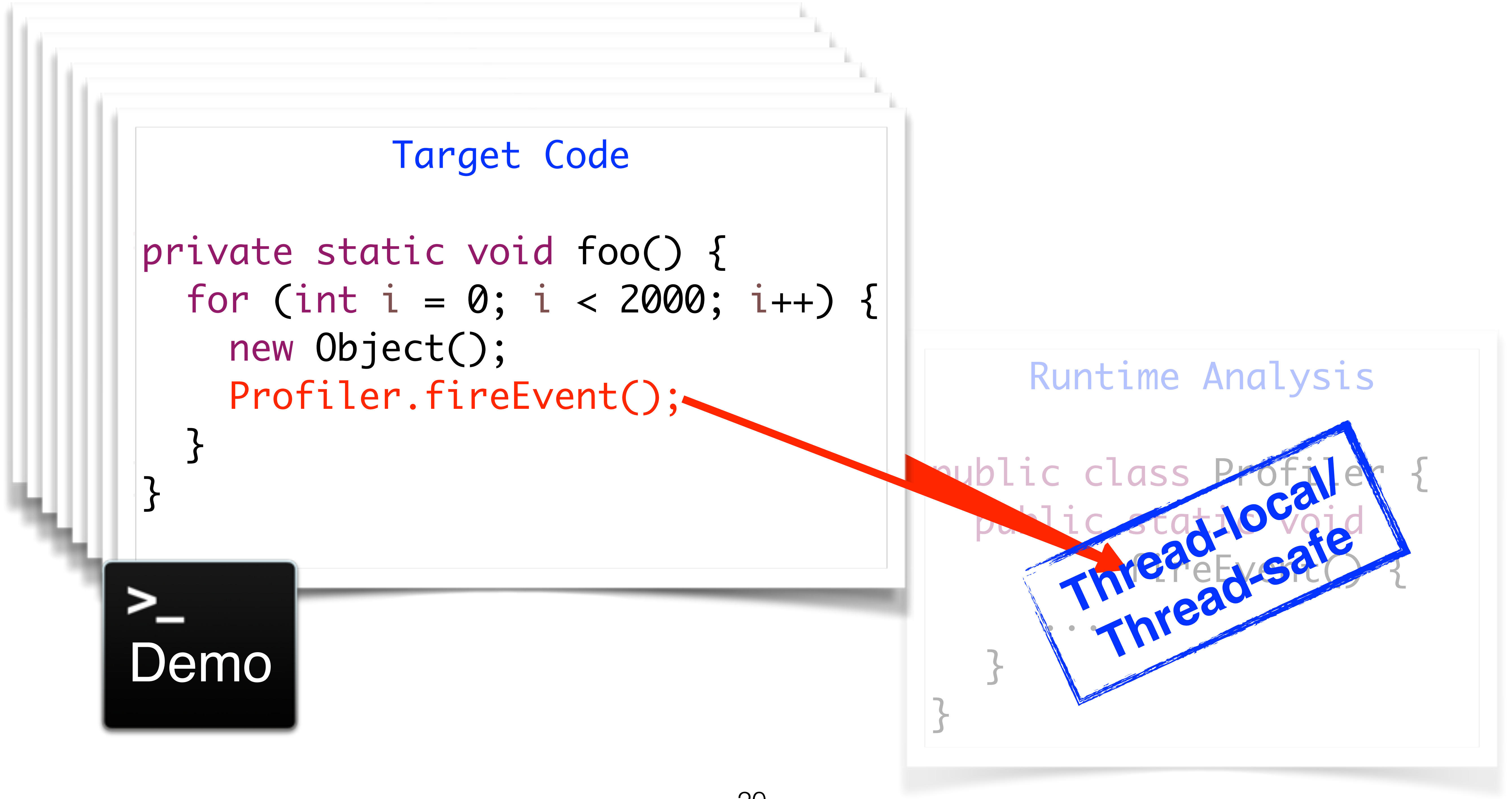
```
private static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent();
  }
}
```

Runtime Analysis

```
public class Profiler {
  public static void
    fireEvent() {
      ...
  }
}
```

Thinking in DiSL

2. Concurrent Event Handling



Target Code

```
private static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent();
  }
}
```

Runtime Analysis

```
public class Profiler {
  public static void
    fireEvent() {
      ...
  }
}
```

>_ Demo

2. Concurrent Event Handling

```
     Runtime Analysis

public class Profiler {
  public static void
         fireEvent() {
    ...
  }
}
```

2. Concurrent Event Handling

Runtime Analysis

```
public class Profiler {
  public static void
          fireEvent() {
     ...
  }
}
```

- Synchronized event handler

2. Concurrent Event Handling

Runtime Analysis

```
public class Profiler {
  public static void
            fireEvent() {
      ...
  }
}
```

- Synchronized event handler
- Thread-local data structure

2. Concurrent Event Handling

```
     Runtime Analysis

public class Profiler {
  public static void
         fireEvent() {
      ...
  }
}
```

- Synchronized event handler
- Thread-local data structure
- Concurrent data structure

Example: Allocation Profiler

Example: Allocation Profiler

- Re-implement using thread-local data structure

Example: Allocation Profiler

- Re-implement using thread-local data structure
  - No locking during event handling (except initialization)

Example: Allocation Profiler

- Re-implement using thread-local data structure
  - No locking during event handling (except initialization)

>_ Demo

Example: Allocation Profiler

- Re-implement using thread-local data structure
  - No locking during event handling (except initialization)
- Re-implement using concurrent data structure

>_
Demo

Example: Allocation Profiler

- Re-implement using thread-local data structure
  - No locking during event handling (except initialization)
- Re-implement using concurrent data structure
  - Lock-free data structure

>_
Demo

Example: Allocation Profiler

- Re-implement using thread-local data structure
  - No locking during event handling (except initialization)
- Re-implement using concurrent data structure
  - Lock-free data structure

>_ Demo

Example: Allocation Profiler

- Re-implement using thread-local data structure
  - No locking during event handling (except initialization)
- Re-implement using concurrent data structure
  - Lock-free data structure
  - Test with multi-threaded program running **different** methods

>_
Demo

Example: Allocation Profiler

- Re-implement using thread-local data structure
  - No locking during event handling (except initialization)
- Re-implement using concurrent data structure
  - Lock-free data structure
  - Test with multi-threaded program running **different** methods
- How do we express the instrumentation?

>_ Demo

Thinking in **DiSL**

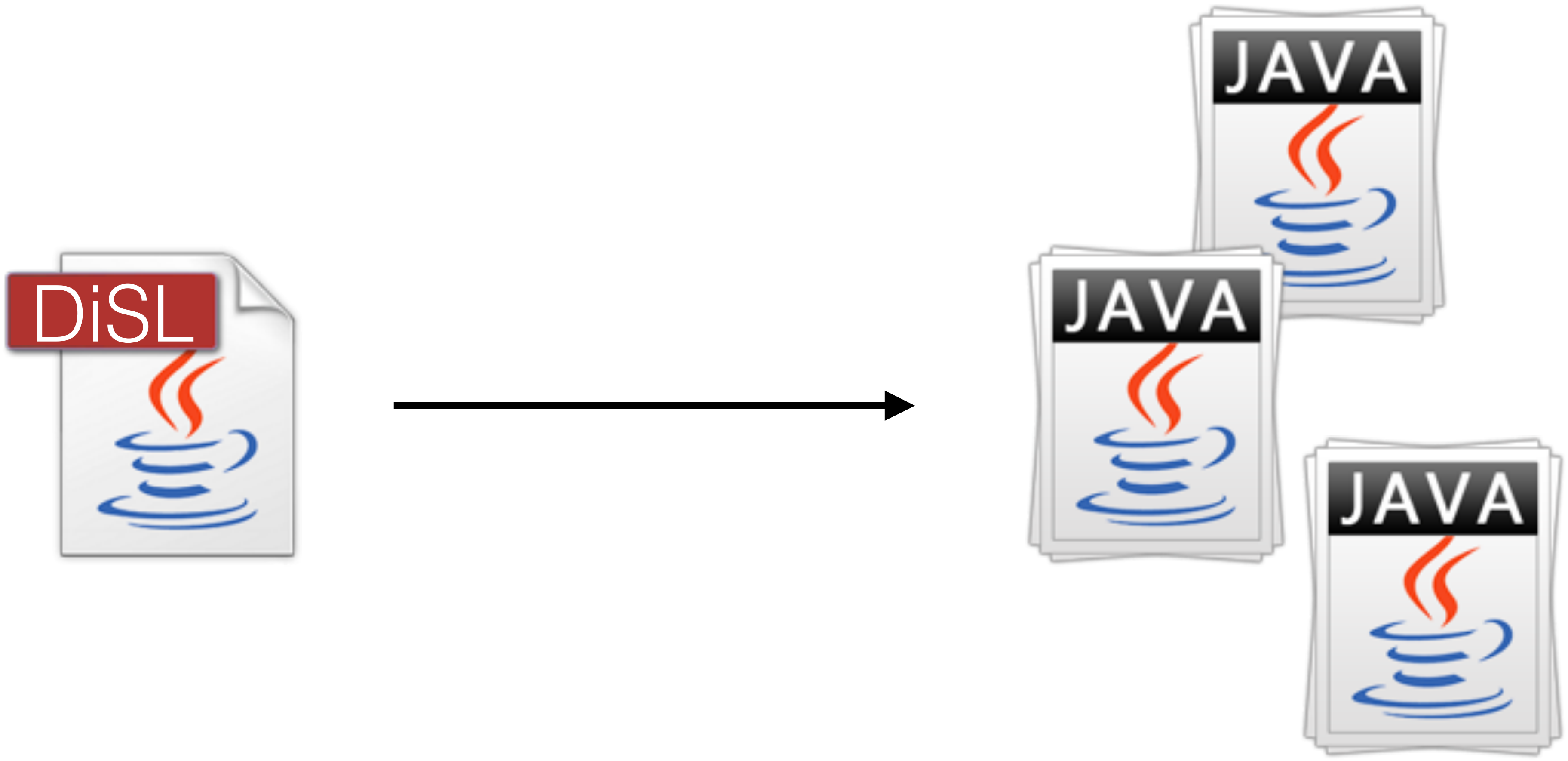
3. Aspect-Oriented

Thinking in **DiSL**

3. **Aspect-Oriented**

Thinking in DiSL

3. **Aspect-Oriented**



Thinking in **DiSL**

*Annotation: describes **where** to instrument*

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

*Method Body: describes **what** to instrument*

Thinking in DiSL

Marks the interested program behavior

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in DiSL

Marks the interested program behavior

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in DiSL

Marks the interested program behavior

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
new java.lang.Object
invokespecial java.lang.Object()
```

Thinking in DiSL

Marks the interested program behavior

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in DiSL

Marks the interested program behavior

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new"  |- BodyMakrer.class
                scope = "A|- BasicBlockMarker.class
static void profile() {   |- TryClauseMarker.class
  Profiler.fireEvent();   |- ExceptionHandlerMarker.class
}
```

Thinking in **DiSL**

Marks the interested program behavior

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new"|- BodyMakrer.class
                scope = "All|- BasicBlockMarker.class
static void profile() {  |- TryClauseMarker.class
  Profiler.fireEvent();  |- ExceptionHandlerMarker.class
}                        |
                         |- <T extends Marker>
                            (Requires ASM knowledge)
```

Thinking in **DiSL**

Defines the relevant position of the instrumentation

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in DiSL

Defines the relevant position of the instrumentation

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
new java.lang.Object
invokespecial java.lang.Object()
```

Thinking in **DiSL**

Defines the relevant position of the instrumentation

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
new java.lang.Object
invokespecial java.lang.Object()
```

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
new java.lang.Object
Profiler.fireEvent();
invokespecial java.lang.Object()
```

Thinking in **DiSL**

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in **DiSL**

```
@AfterReturning
|- @Before
|- @AfterThrowing
|- @After
```

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                 scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in **DiSL**

```
|- @Before
|- @AfterThrowing
|- @After
```

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
|- @Before      args = "new",
|- @AfterThrowing scope = "AllocationTest.foo")
|- @After
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
new java.lang.Object
invokespecial java.lang.Object()
```

Università
della
Svizzera
italiana

Faculty of Informatics

Thinking in **DiSL**

```
@AfterReturning(marker = BytecodeMarker.class,
|- @Before      args = "new",
|- @AfterThrowingscope = "AllocationTest.foo")
|- @After
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
new java.lang.Object
invokespecial java.lang.Object()
```

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
|- @Before      args = "new",
|- @AfterThrowing scope = "AllocationTest.foo")
|- @After
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
Profiler.fireEvent();
new java.lang.Object
invokespecial java.lang.Object()
```

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
|- @Before      args = "new",
|- @AfterThrowing scope = "AllocationTest.foo")
|- @After
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
new java.lang.Object
invokespecial java.lang.Object()
```

Thinking in **DiSL**

```
@AfterReturning(marker = BytecodeMarker.class,
|- @Before      args = "new",
|- @AfterThrowing scope = "AllocationTest.foo")
|- @After
static void profile() {
  Profiler.fireEvent();
}
```

```
goto LOOP_START
try {
new java.lang.Object
} catch (Throwable e) {
  Profiler.fireEvent();
}
invokespecial java.lang.Object()
```

Thinking in **DiSL**

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

```
|- @Before
|- @AfterThrowing
|- @After
```

```
goto LOOP_START
new java.lang.Object
invokespecial java.lang.Object()
```

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Thinking in **DiSL**

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

```
|- @Before
|- @AfterThrowing
|- @After
```

```
goto LOOP_START
try {
new java.lang.Object
} finally {
  Profiler.fireEvent();
}
invokespecial java.lang.Object()
```

Thinking in **DiSL**

Selects methods to instrument

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in **DiSL**

Selects methods to instrument

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                scope = "*.*"AllocationTest.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                guard = SomeGuard.class.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Alternative way of selection.

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                guard = SomeGuard.class.foo")
static void profile() {
  Profiler.fireEvent();
}
```

Alternative way of selection.

1. contains an @GuardMethod returning a boolean.

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                guard = SomeGuard.class
static void profile() {
  Profiler.fireEvent();
}
```

Alternative way of selection.

1. contains an @GuardMethod returning a boolean.
2. has access to static contextual information.

Università della Svizzera italiana

Faculty of Informatics

Thinking in **DiSL**

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new",
                guard = SomeGuard.class
static void profile() {
  Profiler.fireEvent();
}
```

Alternative way of selection.

1. contains an @GuardMethod returning a boolean.
2. has access to static contextual information.

(demo later)

Still With Me?

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile() {
  Profiler.fireEvent();
}
```

Thinking in **DiSL**

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile() {
  Profiler.fireEvent();
}
```

Will be inserted everywhere.

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile() {
  Profiler.fireEvent();
}
```

Will be inserted everywhere.

But where exactly?

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile() {
  Profiler.fireEvent();
}
```

Will be inserted everywhere.

But where exactly?

Answer: depends on the instrumentation location.

Thinking in DiSL

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile() {
  Profiler.fireEvent();
}
```

Will be inserted everywhere.

But where exactly?

Answer: depends on the instrumentation location.

Solution: synthetic method call that represents

Contextual Information

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Contextual Information

Contextual Information

- Dynamic contextual information

Contextual Information

- Dynamic contextual information
    - cannot be resolved until execution.

Università della Svizzera italiana

Faculty of Informatics

Contextual Information

- Dynamic contextual information
    - cannot be resolved until execution.
    - e.g., receiver of an invocation, argument to a method

Contextual Information

- Dynamic contextual information
  - cannot be resolved until execution.
  - e.g., receiver of an invocation, argument to a method

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                 dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}
```

Contextual Information

- Dynamic contextual information

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                 dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}
```

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Contextual Information

- Dynamic contextual information

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                    dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}
```

Contextual Information

- Dynamic contextual information

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                     dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}


final class ProcessImpl {
  static Process start(String[] cmdarray,
                                 ...) throws IOException {
    assert cmdarray != null && cmdarray.length > 0;
```

Contextual Information

- Dynamic contextual information

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                 dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}


final class ProcessImpl {
  static Process start(String[] cmdarray,
                     ...) throws IOException {
    assert cmdarray != null && cmdarray.length > 0;
```

Contextual Information

- Dynamic contextual information

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                 dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}
```

```
final class ProcessImpl {
  static Process start(String[] cmdarray,
                       ...) throws IOException {
    String[] cmd = (String[])
                   dc.getMethodArgumentValue(0, Object.class);
    System.out.println(Arrays.toString(cmd));
    assert cmdarray != null && cmdarray.length > 0;
```

Università della Svizzera italiana | Faculty of Informatics

Contextual Information

- Dynamic contextual information

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                 dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}
```

```
final class ProcessImpl {
  static Process start(String[] cmdarray,
                       ...) throws IOException {
    String[] cmd = (String[])
                   dc.getMethodArgumentValue(0, Object.class);
    System.out.println(Arrays.toString(cmd));
    assert cmdarray != null && cmdarray.length > 0;
```

Contextual Information

- Dynamic contextual information

```
@Before(marker = BodyMarker.class,
        scope = "java.lang.ProcessImpl.start")
static void onNewProcess(DynamicContext dc) {
  String[] cmd = (String[])
                dc.getMethodArgumentValue(0, Object.class);
  System.out.println(Arrays.toString(cmd));
}
```

```
final class ProcessImpl {
  static Process start(String[] cmdarray,
                       ...) throws IOException {
    String[] cmd = (String[]) cmdarray;
    System.out.println(Arrays.toString(cmd));
    assert cmdarray != null && cmdarray.length > 0;
```

Contextual Information

- Dynamic contextual information

Contextual Information

- Dynamic contextual information
  - **[clarify]** DiSL targets Java bytecode, where the variable name may not be available; it uses numeric index and relies on the user for the correct input.

Contextual Information

- Dynamic contextual information
  - **[clarify]** DiSL targets Java bytecode, where the variable name may not be available; it uses numeric index and relies on the user for the correct input.
  - Advanced interface methods require deep understanding of Java Virtual Machine, e.g, `DynamicContext.getStackValue()`

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Contextual Information

Università
della
Svizzera
italiana

**Faculty
of Informatics**

Contextual Information

- Static contextual information

Contextual Information

- Static contextual information
  - can be resolved to either primitive or string constant before execution

Contextual Information

- Static contextual information
  - can be resolved to either primitive or string constant before execution
  - e.g., method name, source line number

Contextual Information

- Static contextual information
  - can be resolved to either primitive or string constant before execution
  - e.g., method name, source line number
  - customizable (requires ASM knowledge)

Contextual Information

- Static contextual information
  - can be resolved to either primitive or string constant before execution
  - e.g., method name, source line number
  - customizable (requires ASM knowledge)
  - can be used in @GuardMethod

Example: Allocation Profiler

Example: Allocation Profiler

- Group allocation profile by method name

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

```
static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
  }
}
```

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

```
static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
  }
}
```

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

```
static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent(msc.thisMethodName());
  }
}
```

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

```
static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent(msc.thisMethodName());
  }
}
```

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

```
static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent("foo");
  }
}
```

Example: Allocation Profiler

- Group allocation profile by method name

```
@AfterReturning(marker = BytecodeMarker.class,
                args = "new")
static void profile(MethodStaticContext msc) {
  Profiler.fireEvent(msc.thisMethodName());
}
```

```
static void foo() {
  for (int i = 0; i < 2000; i++) {
    new Object();
    Profiler.fireEvent("foo");
  }
}
```

>_
Demo

Example: mvn

- Instrument method annotated by @Test

```
public class GuardUnitTest {
  @GuardMethod
  public static boolean
              isApplicable(AnnotationContext context) {
    return context.annotatedByTest();
  }
}

public class AnnotationContext
              extends MethodStaticContext {
  public boolean annotatedByTest() {
    ... // code using ASM
  }
}
```

Example: mvn

- Instrument method annotated by @Test

```
public class GuardUnitTest {
  @GuardMethod
  public static boolean
         isApplicable(AnnotationContext context) {
    return context.annotatedByTest();
  }
}

public class AnnotationContext
       extends MethodStaticContext {
  public boolean annotatedByTest() {
    ... // code using ASM
  }
}
```

>_ Demo

Other **DiSL** Features

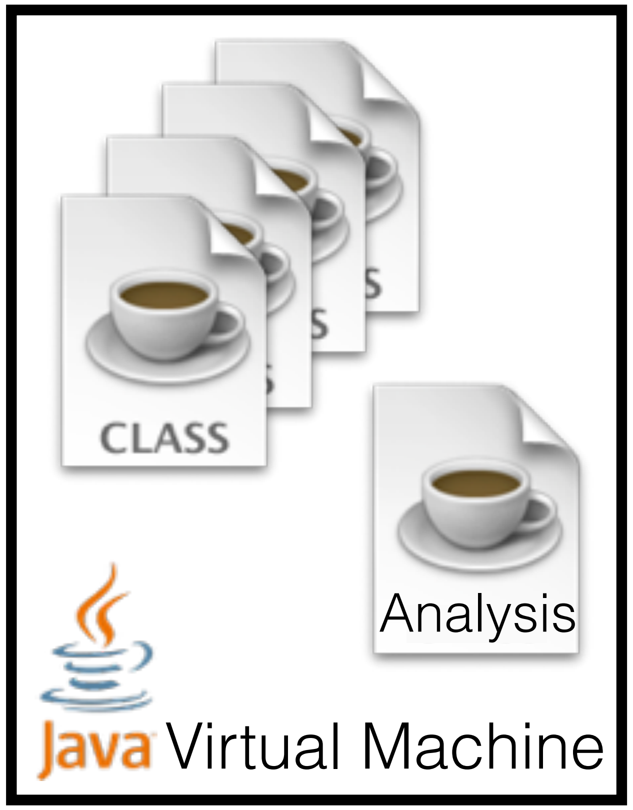
- @SyntheticLocal: share data between different instrumentation at the same method
- @ThreadLocal: append a field in java.lang.Thread
- Argument Processor
  - Single code snippet to process arguments of the same type

Università della Svizzera italiana

Faculty of Informatics

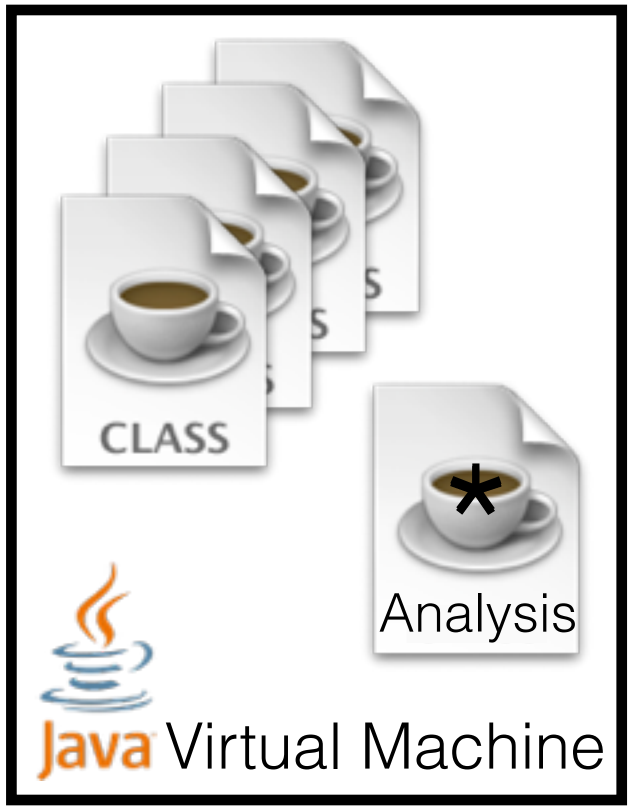
Projects Derived From DiSL

ShadowVM



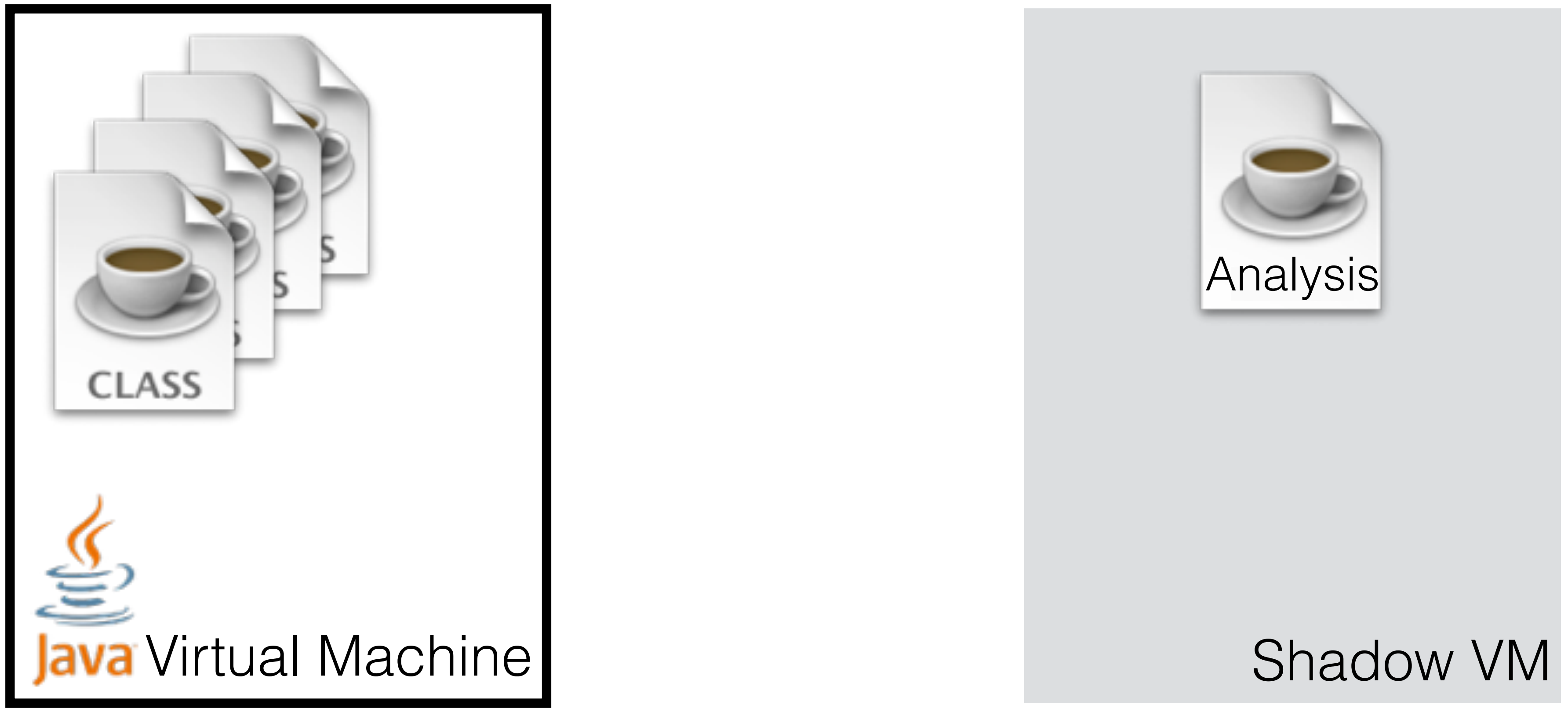
In-process Analysis

ShadowVM



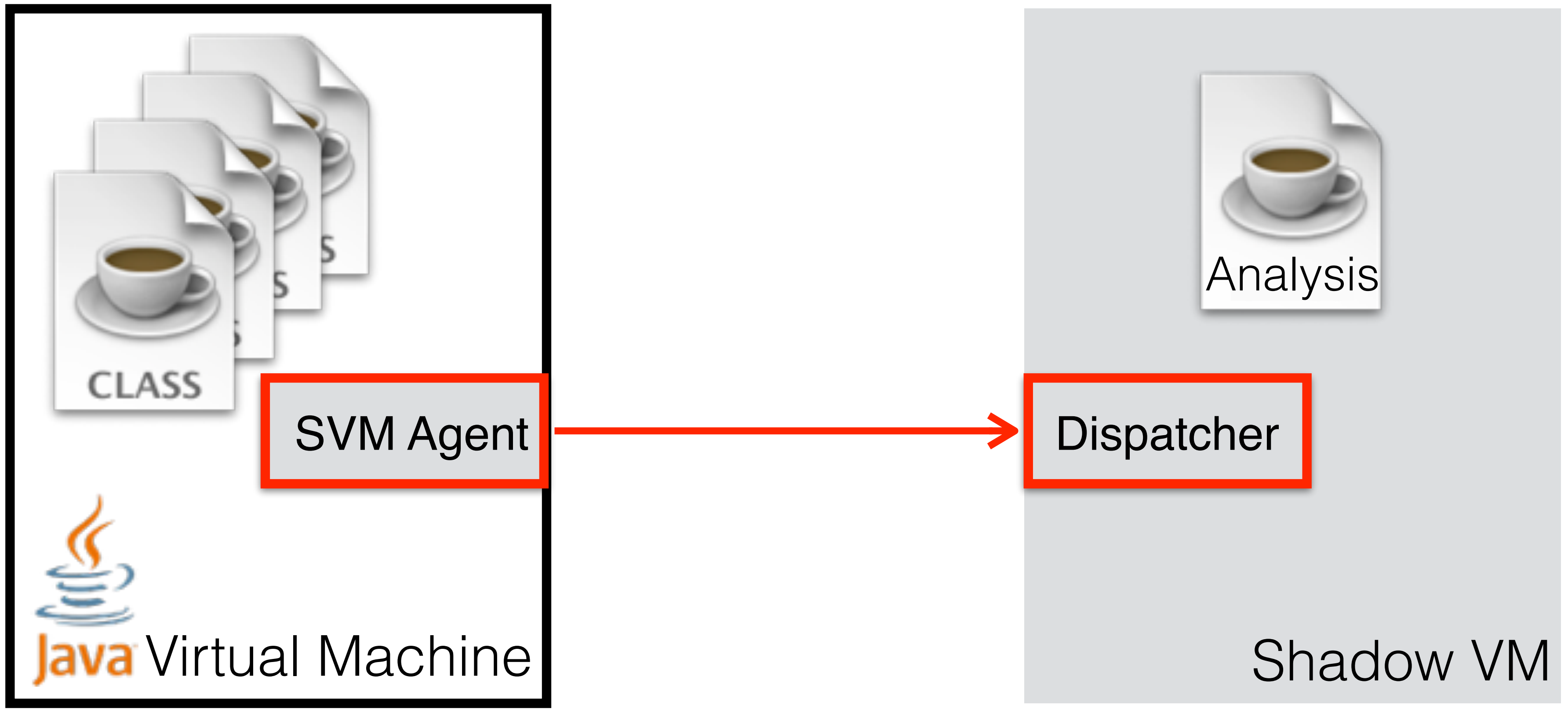
In-process Analysis

ShadowVM



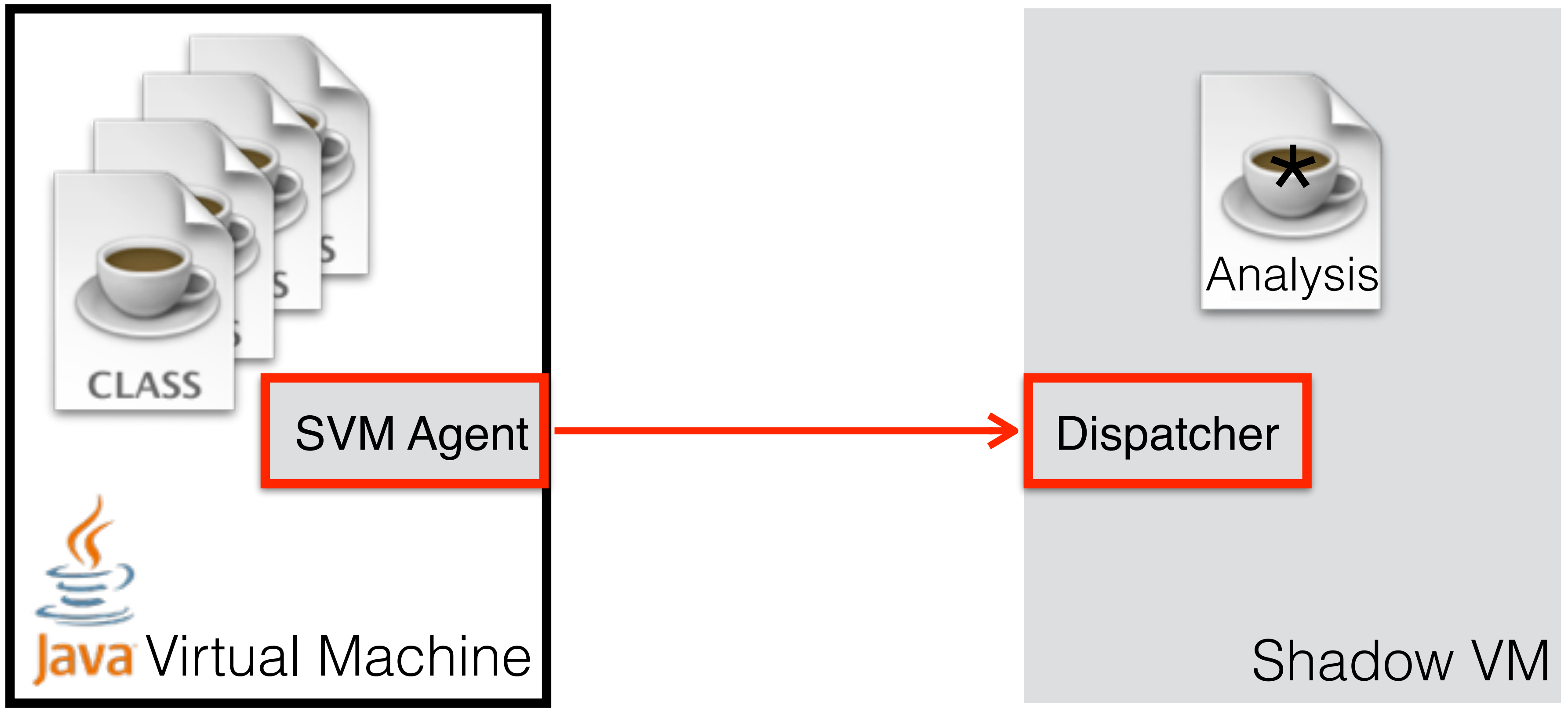
ShadowVM Analysis

ShadowVM



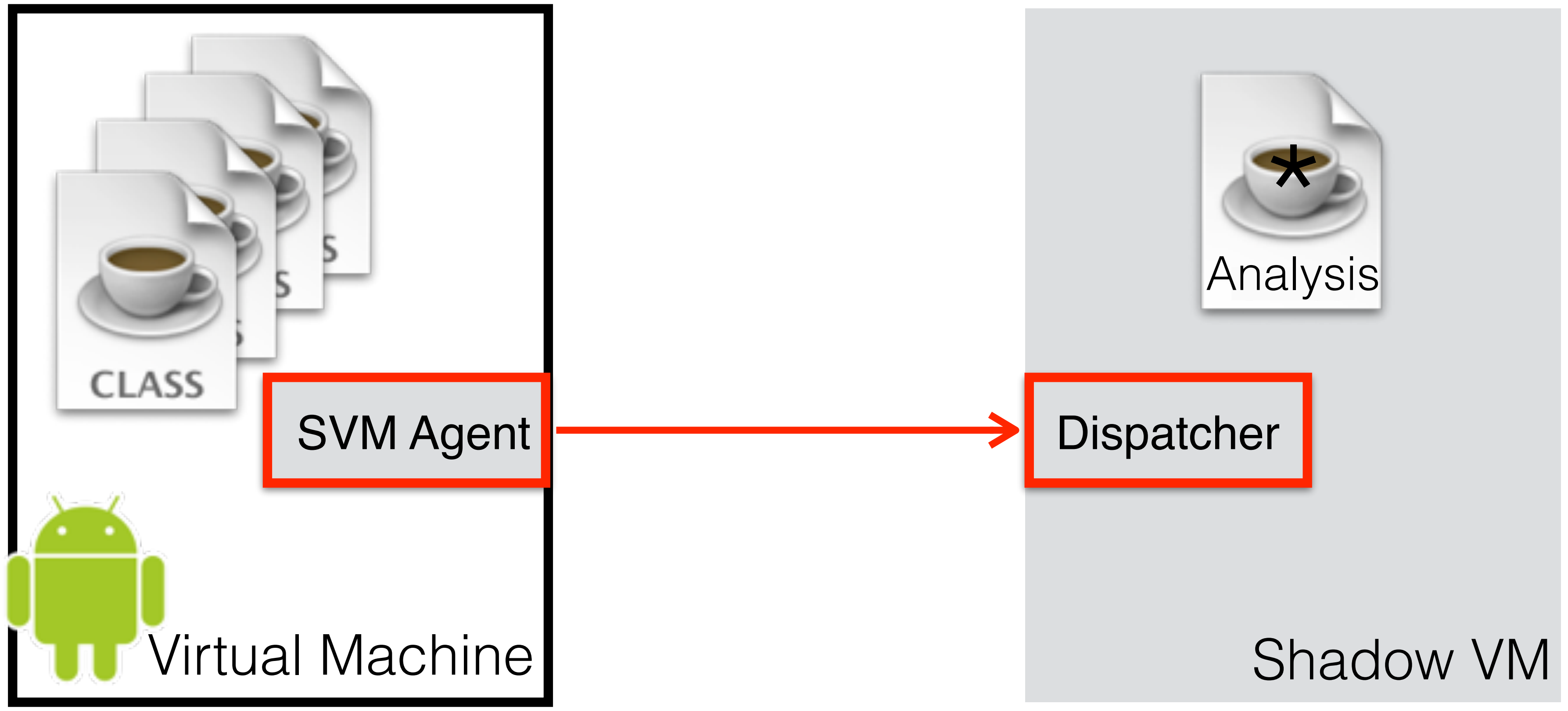
ShadowVM Analysis

ShadowVM



ShadowVM Analysis

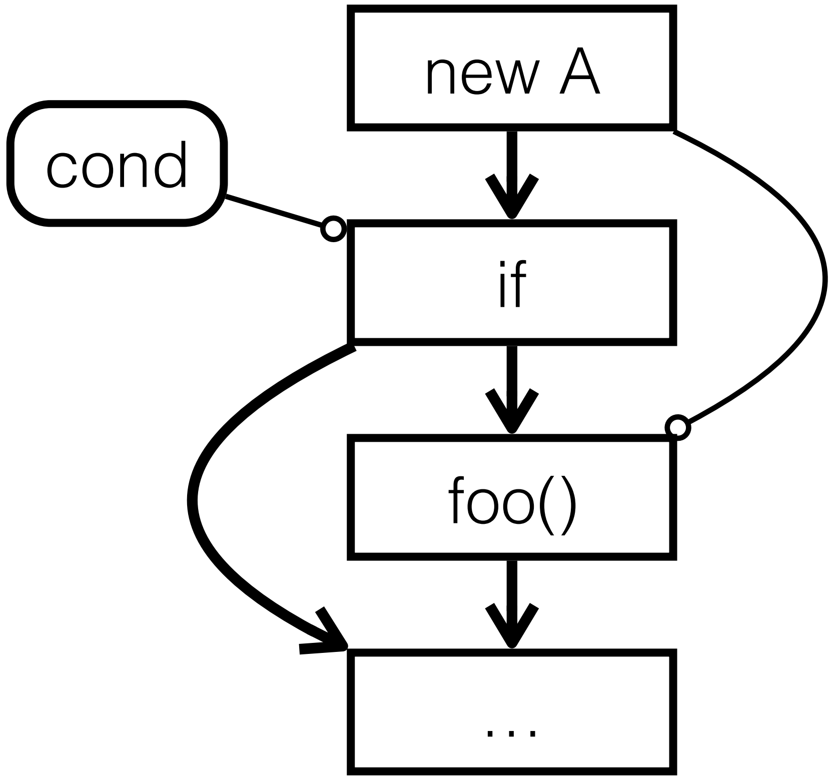
ShadowVM



ShadowVM Analysis

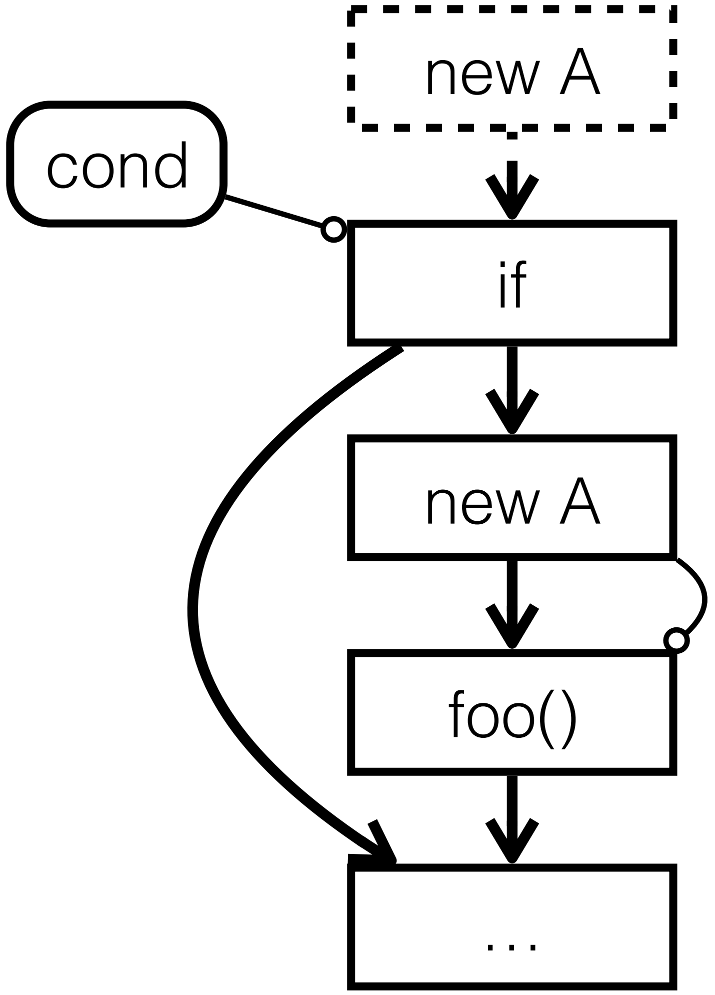
Accurate Profiling - The Problem

```
A a = new A();
if (cond) { // 10% taken
   a.foo(); // a escapes
}
```



Accurate Profiling - The Problem

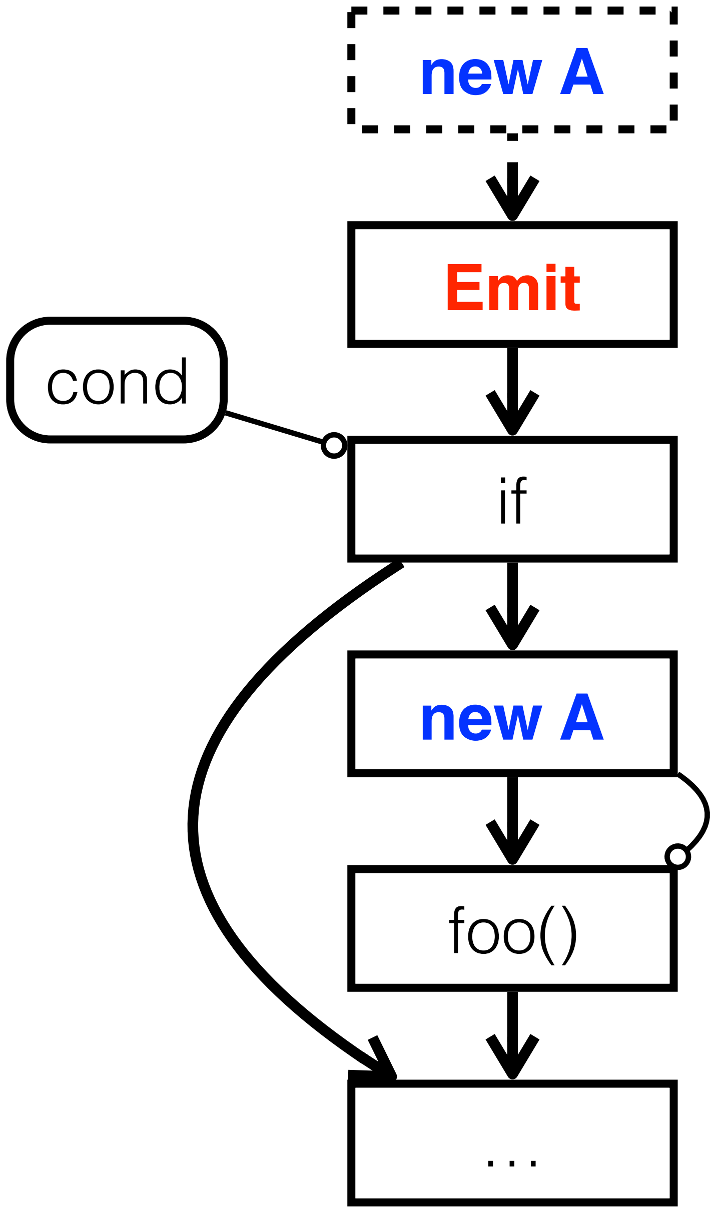
```
A a = new A();
if (cond) { // 10% taken
    a.foo(); // a escapes
}
```



* Lukas Stadler, Thomas Würthinger, Hanspeter Mössenböck. Partial Escape Analysis and Scalar Replacement for Java. CGO '14

Accurate Profiling - The Problem

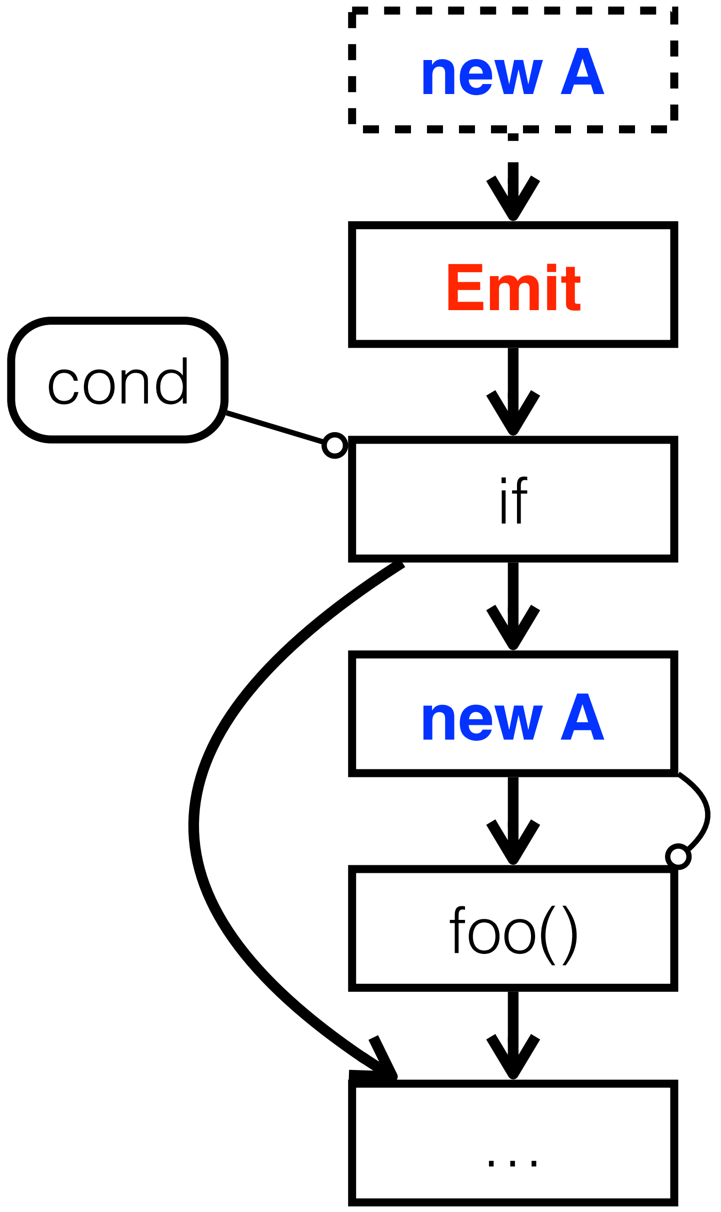
```
A a = new A();
EmitAllocEvent();
if (cond) { // 10% taken
   a.foo(); // a escapes
}
```



Accurate Profiling - The Problem

```
A a = new A();
EmitAllocEvent();
if (cond) { // 10% taken
   a.foo(); // a escapes
}
```

For 1000 executions of the code snippet:
  Actual Allocations: 100
  Intercepted Allocations: 1000
  Without-instrumentation: 100

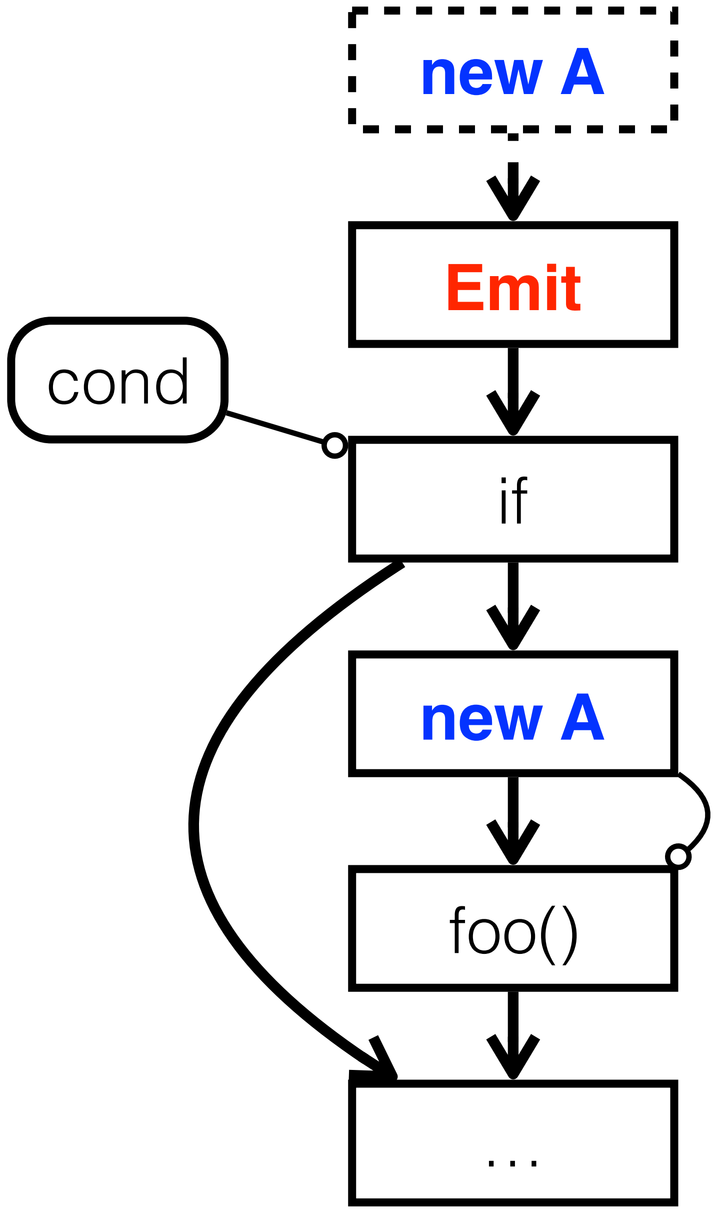


Accurate Profiling - The Problem

```
A a = new A();
EmitAllocEvent();
if (cond) { // 10% taken
   a.foo(); // a escapes
}
```

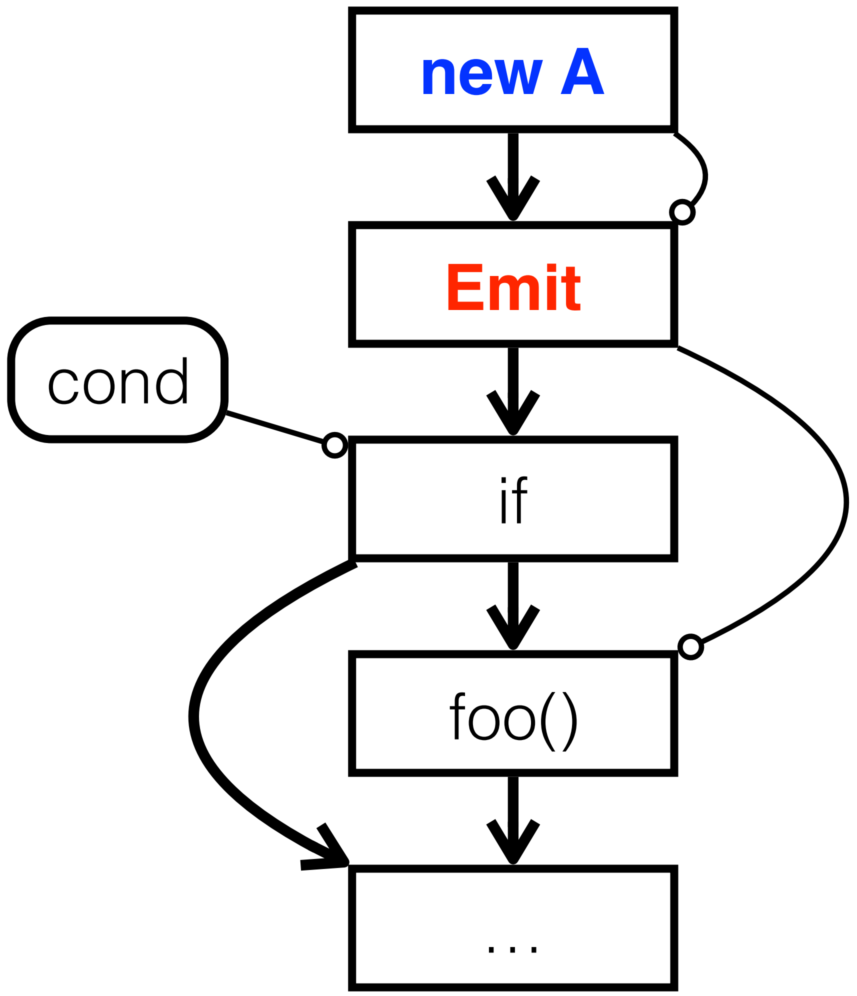
For 1000 executions of the code snippet:
   Actual Allocations: 100
   Intercepted Allocations: 1000
   Without instrumentation: 100

OVER-PROFILED



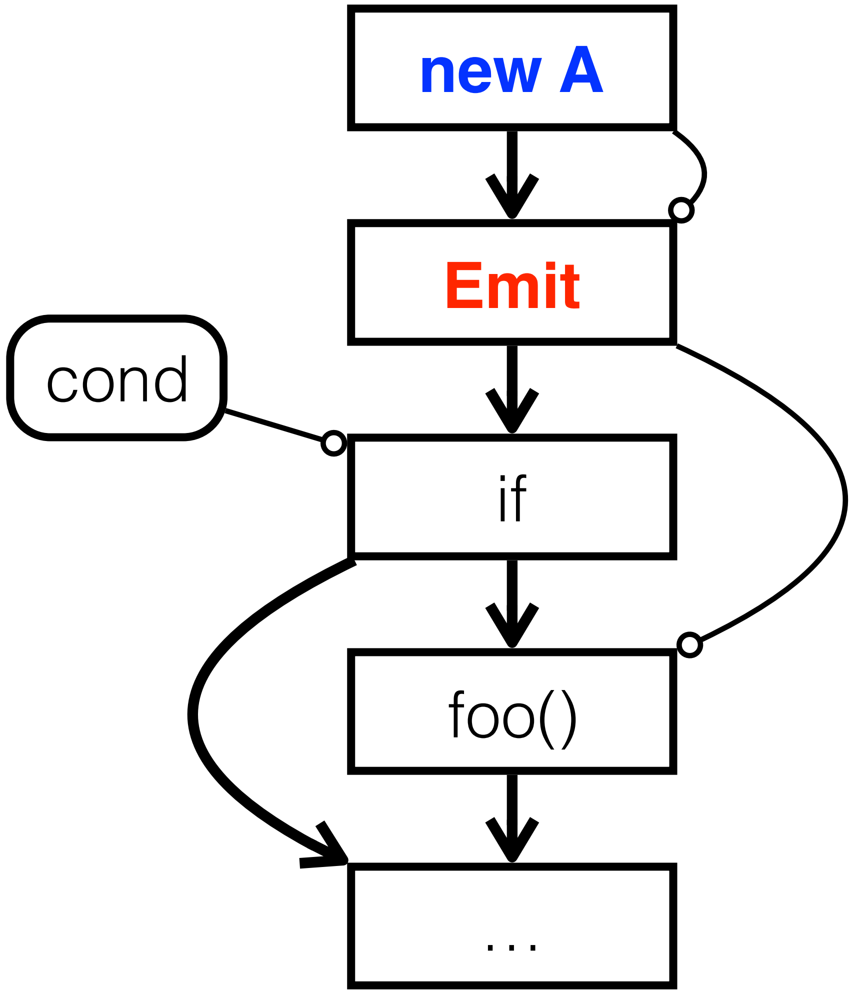
Accurate Profiling - The Problem

```
A a = new A();
EmitAllocEvent(a);
if (cond) { // 10% taken
   a.foo(); // a escapes
}
```



Accurate Profiling - The Problem

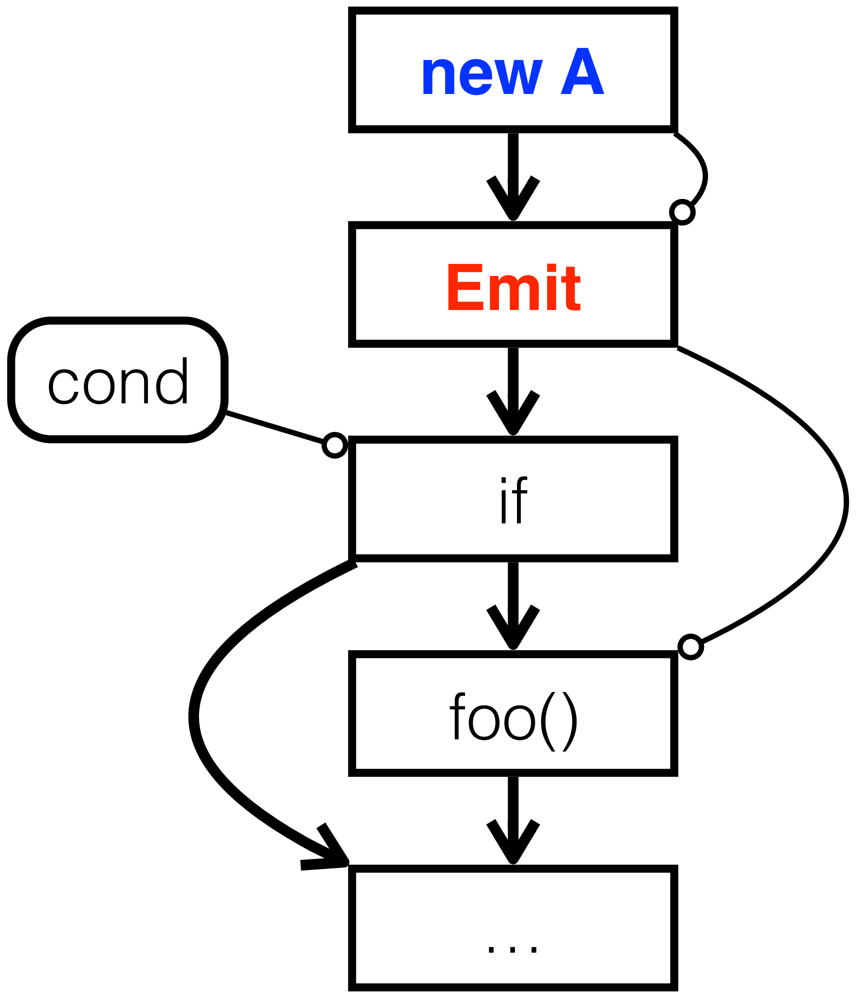
```
A a = new A();
EmitAllocEvent(a);
if (cond) { // 10% taken
   a.foo(); // a escapes
}
```



For 1000 executions of the code snippet:
  Actual Allocations: 1000
  Intercepted Allocations: 1000
  Without-instrumentation: 100

Accurate Profiling - The Problem

```
A a = new A();
EmitAllocEvent(a);
if (cond) { // 10% taken
   a.foo(); // a escapes
}
```



For 1000 executions of the code snippet:
  Actual Allocations: 1000
  Intercepted Allocations: 1000
  Without instrumentation: 100

OVER-PROFILED

Questions?

yudi.zheng@usi.ch
https://github.com/mur47x111/DiSLDemo

1. DiSL: a domain-specific language for bytecode instrumentation. AOSD '12
2. ShadowVM: robust and comprehensive dynamic program analysis for the java platform. GPCE '13
3. A programming model and framework for comprehensive dynamic analysis on Android. MODULARITY '15
4. Accurate profiling in the presence of dynamic compilation. OOPSLA '15