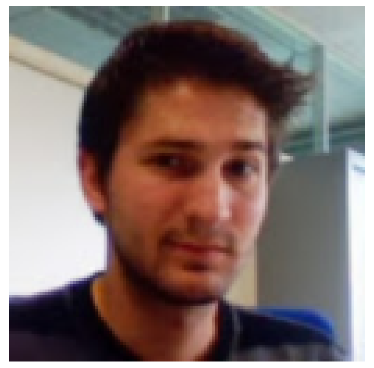
Nicolas Comet
n.comet@lectra.com

`Byte Buddy` : bytecode gen made *easy* !

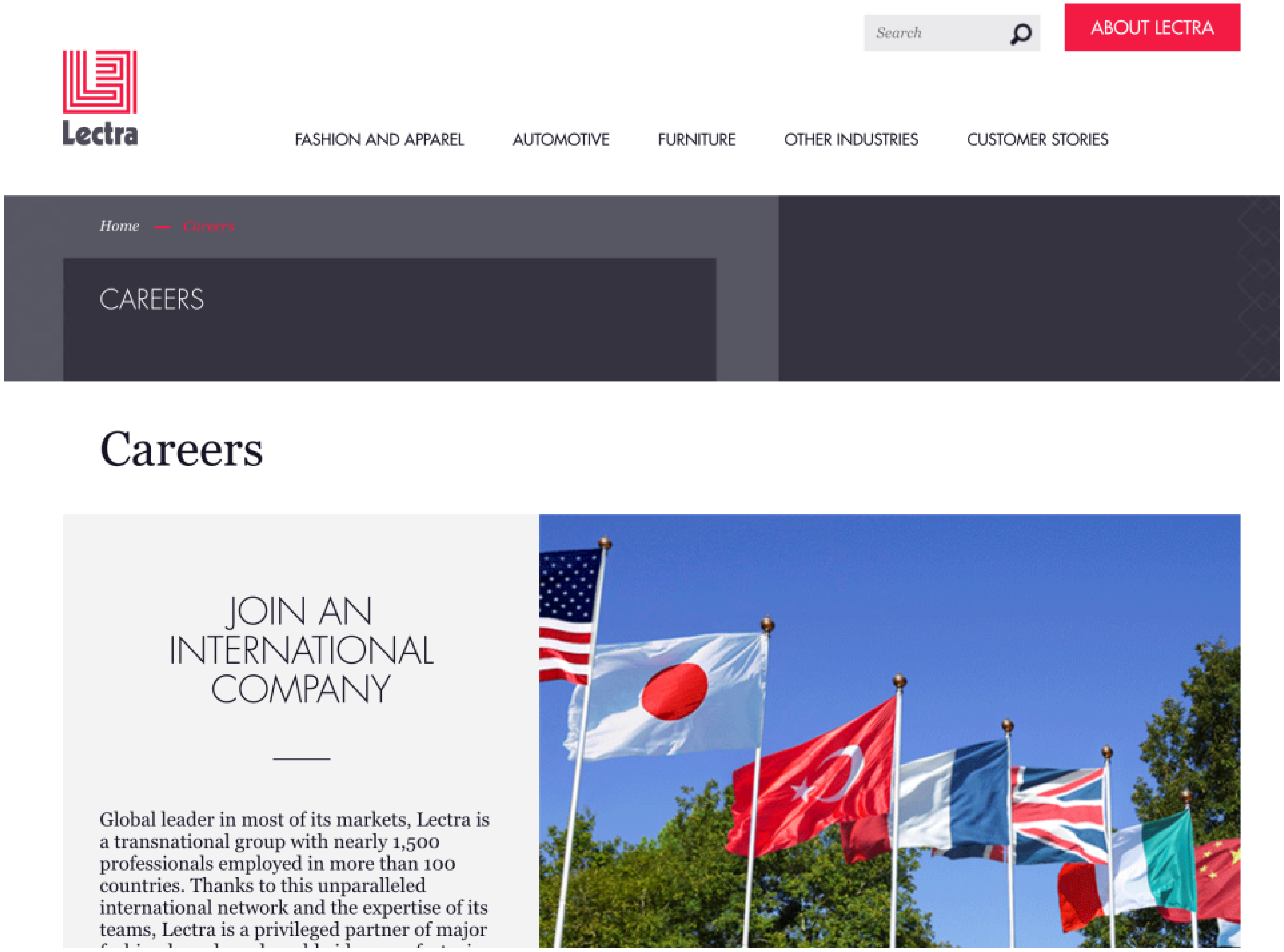
About me

@NicolasComet
https://github.com/ncomet



- Developer since 7 years
- Software R&D Engineer @Lectra, worldwide leader in integrated softwares, machines and cloud services for soft material industries
- Speaker & Member of Bordeaux JUG

Lectra



CAREERS

Careers

JOIN AN INTERNATIONAL COMPANY

Global leader in most of its markets, Lectra is a transnational group with nearly 1,500 professionals employed in more than 100 countries. Thanks to this unparalleled international network and the expertise of its teams, Lectra is a privileged partner of major

How it all began

"*Making Java more dynamic*" @Devoxx France 2015

https://youtu.be/vjv4idwQL7k

Rafael Winterhalter

Cross Cutting Concerns

- Non business/domain related
- Orthogonal preoccupations

Cross Cutting Concerns

- Non business/domain related
- Orthogonal preoccupations

Examples

- Logging
- Caching
- Monitoring
- Data validation
- Real time constraints
- Persistence
- Transactions
- ...

Adding behavior

- Transparent
- Non intrusive
- Strongly typed
- Regardless of type
- Lightweight

(Some) solutions

- Reflection
- Aspect Oriented Programming (AOP)
- Bytecode generation

Reflection

`java.lang.reflect`

Reading Type metamodel at runtime

Calling constructors, methods, access attributes (sometimes unsafely)

Reflection

`java.lang.reflect`

Reading Type metamodel at runtime

Calling constructors, methods, access attributes (sometimes unsafely)

Mostly about *Introspection*

Reflection

`java.lang.reflect`

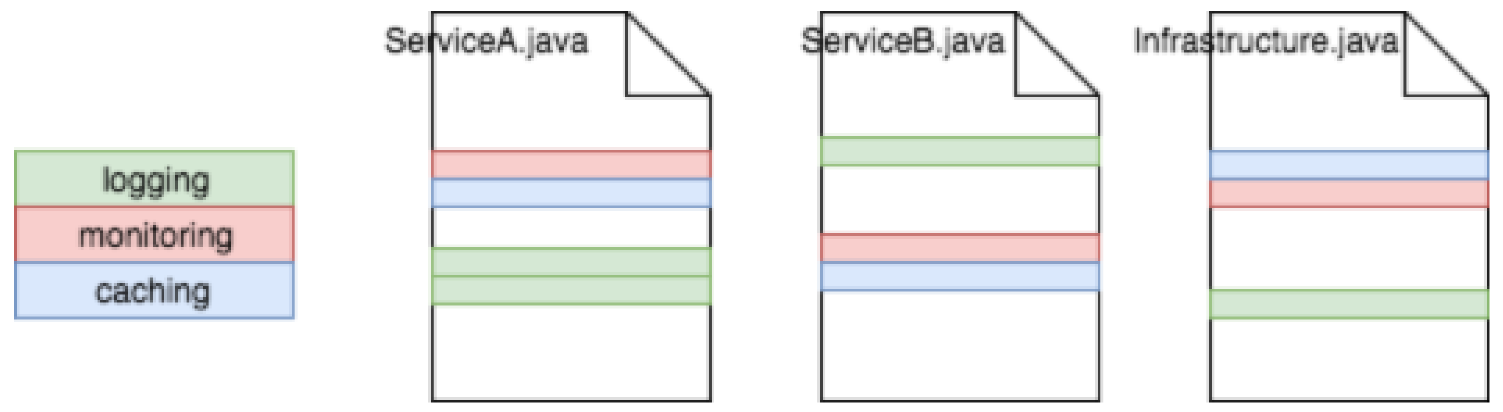
Reading Type metamodel at runtime

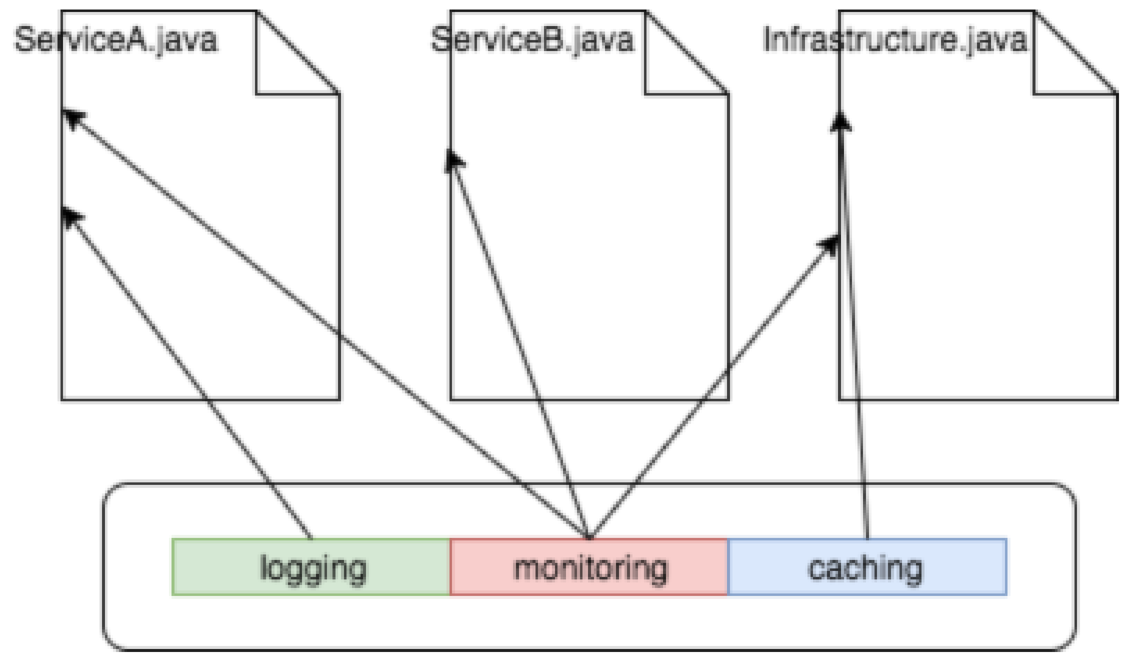
Calling constructors, methods, access attributes (sometimes unsafely)

Mostly about *Introspection*

It has a cost (JIT is useless)

AOP concepts



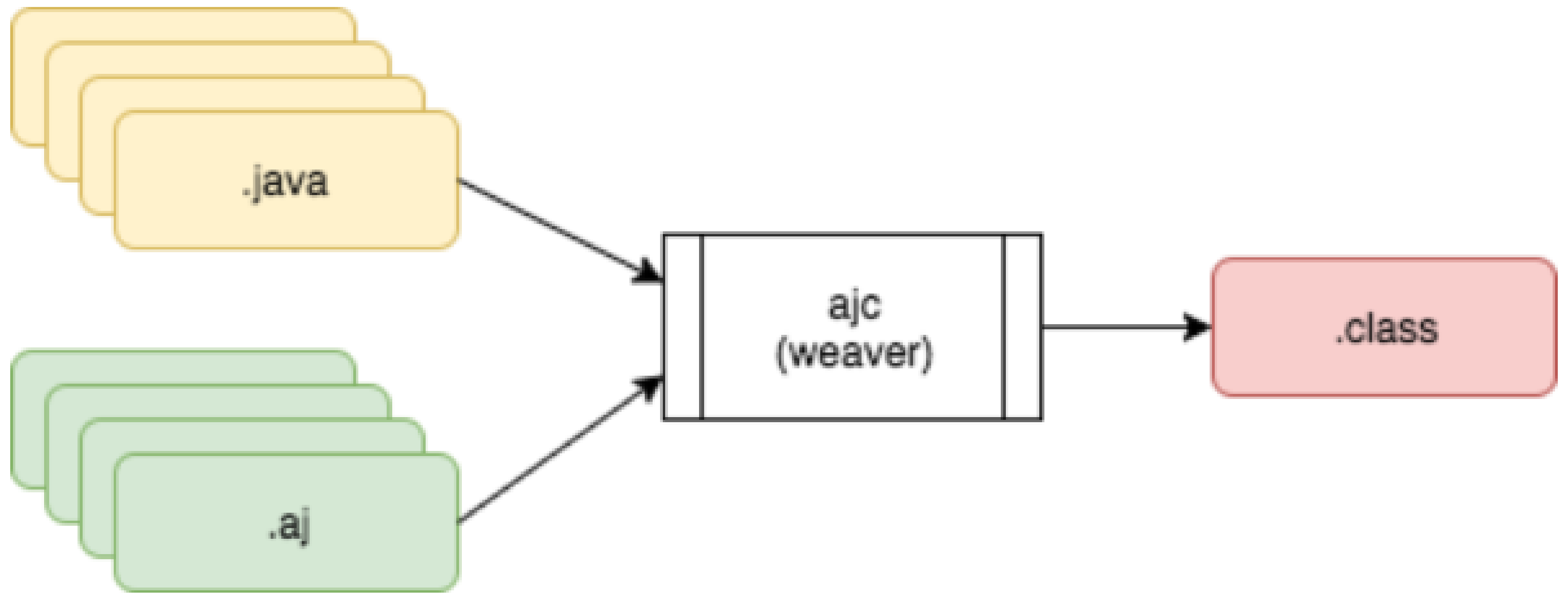


AOP

- **Aspect** Description of a cross cutting concern
- **Join point** A point during the execution of code (method execution, attribute access)
- **Advice** Action taken by an aspect at a particular join point
- **Pointcut** A regular expression that matches join points.

AOP

- **Aspect** Description of a cross cutting concern
- **Join point** A point during the execution of code (method execution, attribute access)
- **Advice** Action taken by an aspect at a particular join point
- **Pointcut** A regular expression that matches join points.

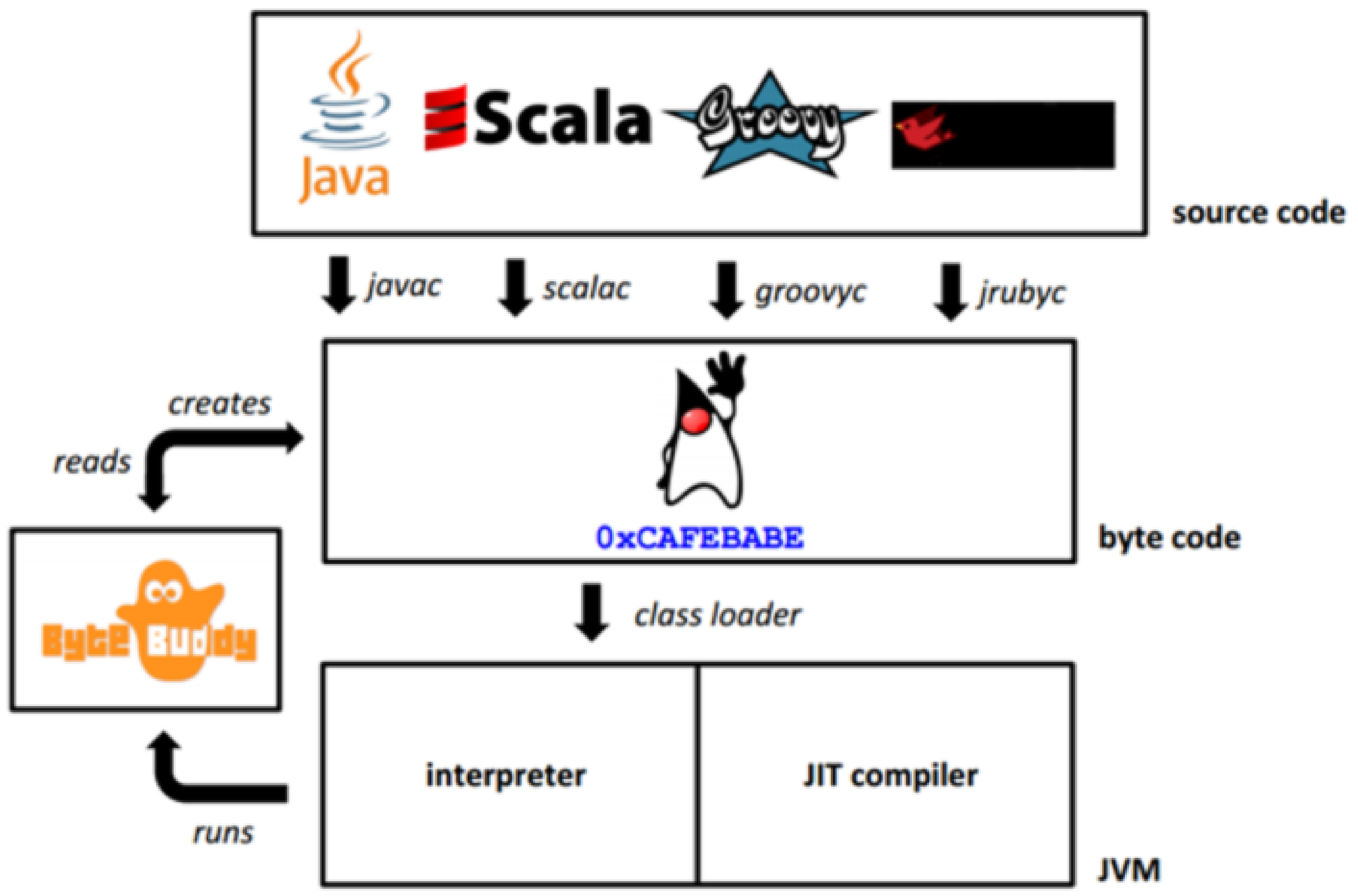


Java Bytecode

```
// access flags 0x1
public getName()Ljava/lang/String;
 L0
  LINENUMBER 16 L0
  ALOAD 0
  GETFIELD domain/Cat.name : Ljava/lang/String;
  ARETURN
 L1
  LOCALVARIABLE this Ldomain/Cat; L0 L1 0
  MAXSTACK = 1
  MAXLOCALS = 1

// access flags 0x1
public setName(Ljava/lang/String;)V
 L0
  LINENUMBER 20 L0
  ALOAD 0
  ALOAD 1
  PUTFIELD domain/Cat.name : Ljava/lang/String;
 L1
  LINENUMBER 21 L1
  RETURN
 L2
  LOCALVARIABLE this Ldomain/Cat; L0 L2 0
  LOCALVARIABLE name Ljava/lang/String; L0 L2 1
  MAXSTACK = 2
  MAXLOCALS = 2
```

Bytecode generation



JavaOne™

Frameworks

spring
HIBERNATE
mockito
Guice
eclipselink
play
Clover
OpenEJB
APACHE WICKET
APACHE SHIRO
OpenJDK
GRAILS

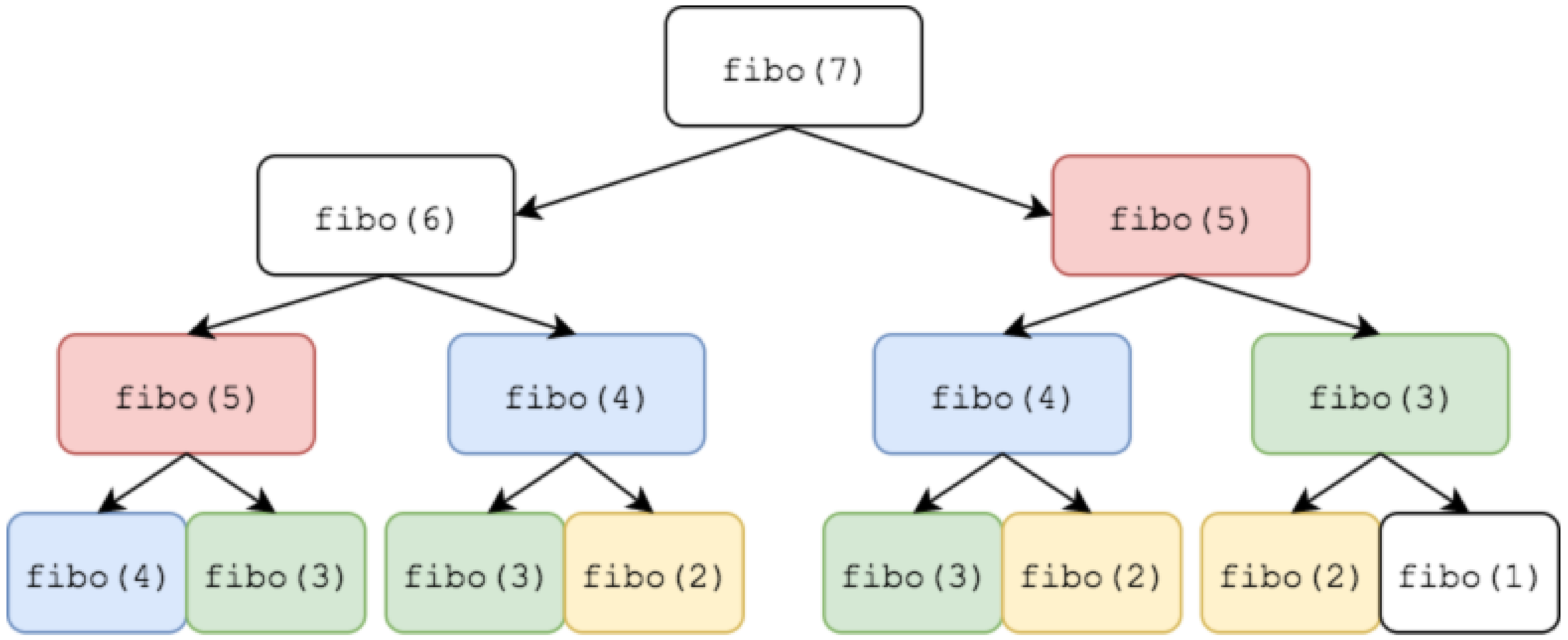
The famous case

- $n \in \mathbb{N}$

$$f(n) = \begin{cases} 0 & if \quad n = 0 \\ 1 & if \quad n = 1 \\ f(n-1) + f(n-2) & if \quad n > 1 \end{cases}$$

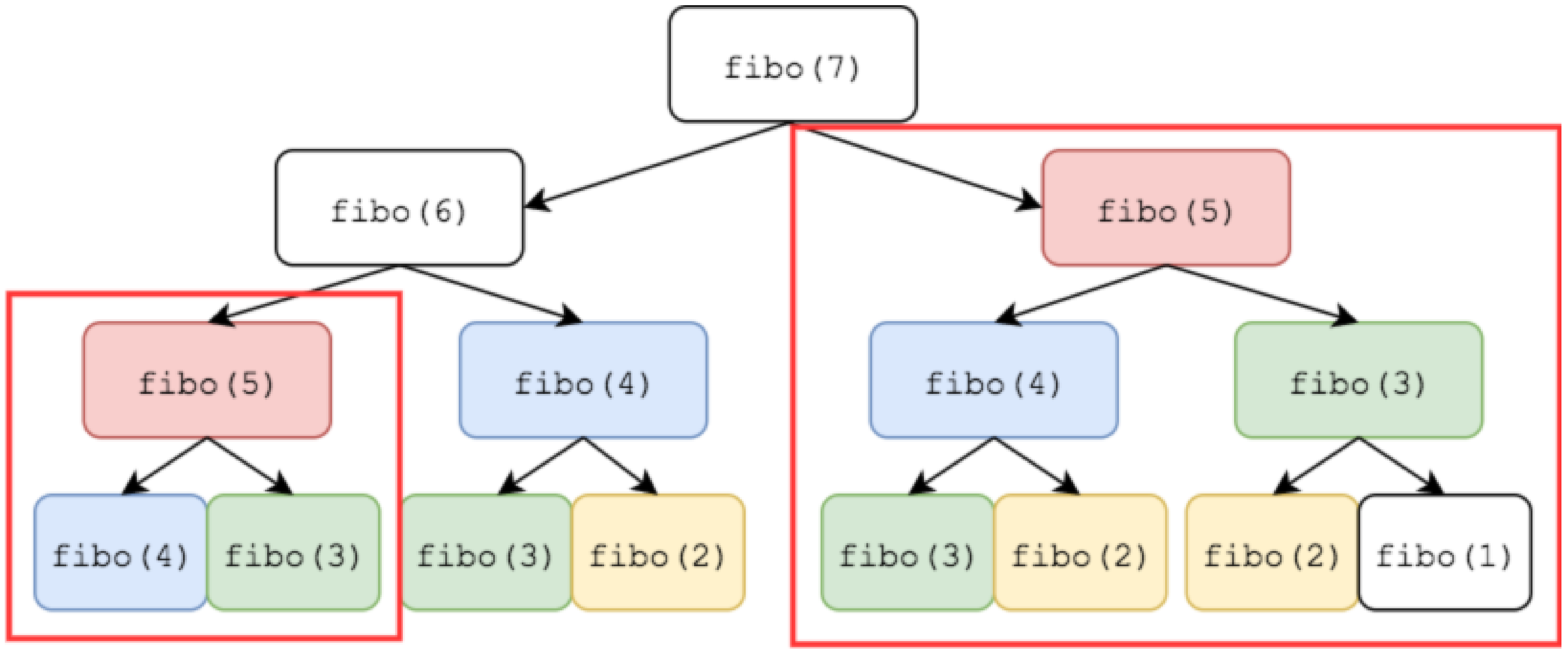
The famous case

- Call tree



Caching

- Memoization



Some code !

Perf comparison

Calling `fibonacci(42)` (average results)

Version	Time
Raw Fibonacci	`1123.658` ms
AspectJ (compile time)	`0.013` ms
Byte Buddy (runtime)	`0.689` ms
Spring AOP	`2123` ms (first time, then instant)

Under the hood

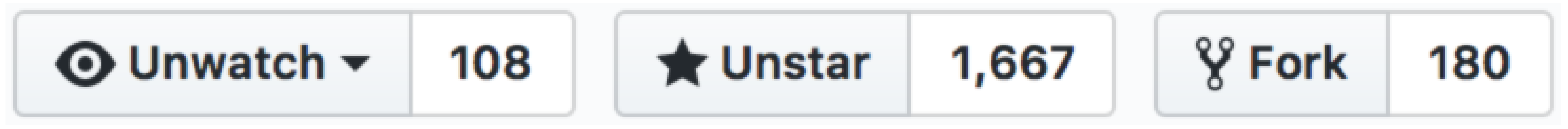
- AspectJ
  - compile time weaving (`ajc`)
  - post-compile weaving (on classes and jars)
  - load time weaving (agent)
  - intercept everything
- Spring AOP
  - proxy-based
    - Interface → Java dynamic proxy
    - else CGLIB bytecode generated proxy
  - good AspectJ integration if you need more

Pros & Cons

- AspectJ
  - ⊖ Setup
  - ⊖ DSL to learn
  - ⊕ Performance
  - ⊕ Non intrusive
  - ⊕ Span
- Spring AOP
  - ⊖ Not really AOP
  - ⊖ Component's public methods only
  - ⊖ / ⊕ Framework
  - ⊕ Spring integration
  - ⊕ Migration to AspectJ
- Byte Buddy
  - ⊖ / ⊕ No compile time
  - ⊕ Library
  - ⊕ Java DSL API
  - ⊕ Performance
  - ⊕ Agent writing help

Byte Buddy

Open Source (license Apache), used by `Mockito`, `Hibernate`, `Google Bazle`, and others

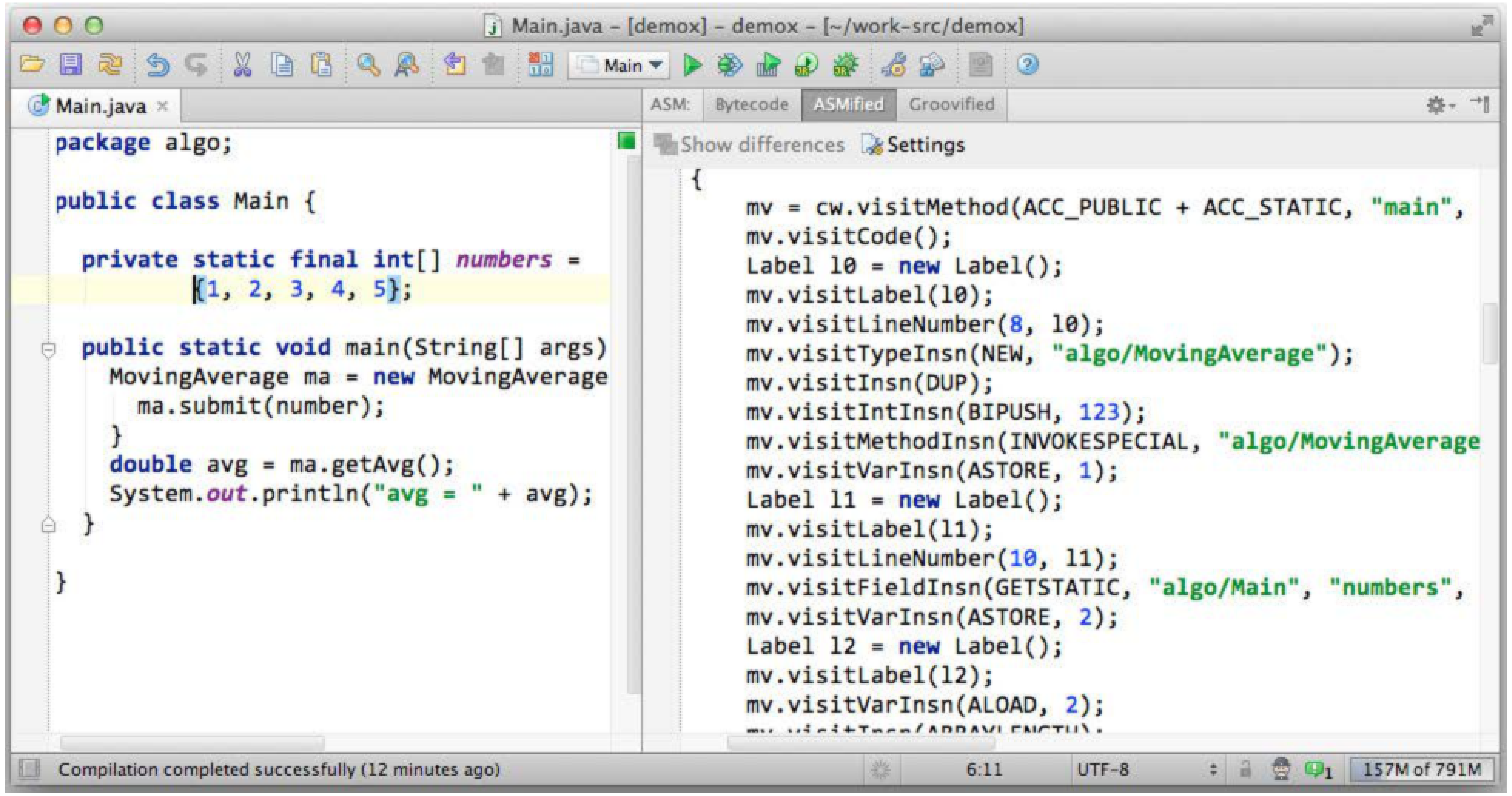


https://github.com/raphw/byte-buddy

http://bytebuddy.net

- Light
- Easy to use (compared to CGLIB, BCEL, ASM)
- Become a library writer

ASM Visitor



```java
package algo;

public class Main {

  private static final int[] numbers =
          {1, 2, 3, 4, 5};

  public static void main(String[] args)
    MovingAverage ma = new MovingAverage
      ma.submit(number);
    }
    double avg = ma.getAvg();
    System.out.println("avg = " + avg);
  }

}
```

```java
{
    mv = cw.visitMethod(ACC_PUBLIC + ACC_STATIC, "main",
    mv.visitCode();
    Label l0 = new Label();
    mv.visitLabel(l0);
    mv.visitLineNumber(8, l0);
    mv.visitTypeInsn(NEW, "algo/MovingAverage");
    mv.visitInsn(DUP);
    mv.visitIntInsn(BIPUSH, 123);
    mv.visitMethodInsn(INVOKESPECIAL, "algo/MovingAverage
    mv.visitVarInsn(ASTORE, 1);
    Label l1 = new Label();
    mv.visitLabel(l1);
    mv.visitLineNumber(10, l1);
    mv.visitFieldInsn(GETSTATIC, "algo/Main", "numbers",
    mv.visitVarInsn(ASTORE, 2);
    Label l2 = new Label();
    mv.visitLabel(l2);
    mv.visitVarInsn(ALOAD, 2);
```

Compilation completed successfully (12 minutes ago)   6:11   UTF-8   157M of 791M

Demo

Slides :

https://ncomet.github.io/javaone2017-bytebuddy/bytebuddy.html

Sources :

https://github.com/ncomet/javaone2017-bytebuddy

Conclusion

- Adding behavior
  - AOP → Implementing multiple cross cutting concerns
  - Byte Buddy → Writing libraries/frameworks or agents
- Discovering at runtime
  - Reflection → Custom serialization, ***nasty*** things (setting private fields…)