Jinline

A Tool for Altering Java Semantics

Eric Tanter and Marc Ségura-Devillechaise

etanter@dcc.uchile.cl, msegura@emn.fr

University of Chile and Ecole des Mines de Nantes

Content

- introduction to alteration and Java
- Jinline
- example
- conclusion

Altering semantics

The need to alter semantics

- component integration
- adaptation, customization
- SOC
- AOP, custom extensions. . .

Approaches to alteration

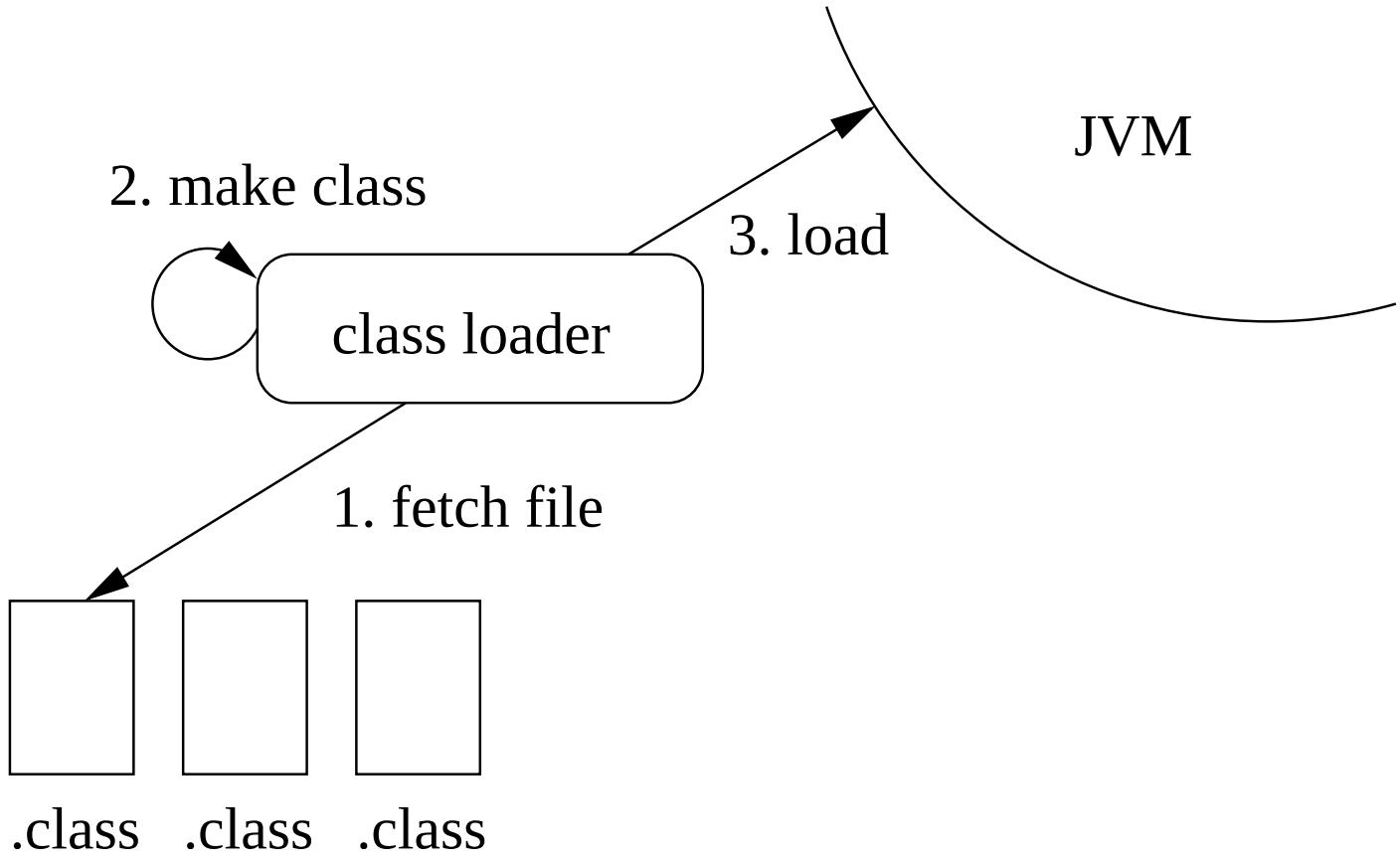
- source editing ;-)
- macro system
- reflection
  - specialization of the class model
  - modification of the interpreter
  - code transformation

Altering semantics in Java

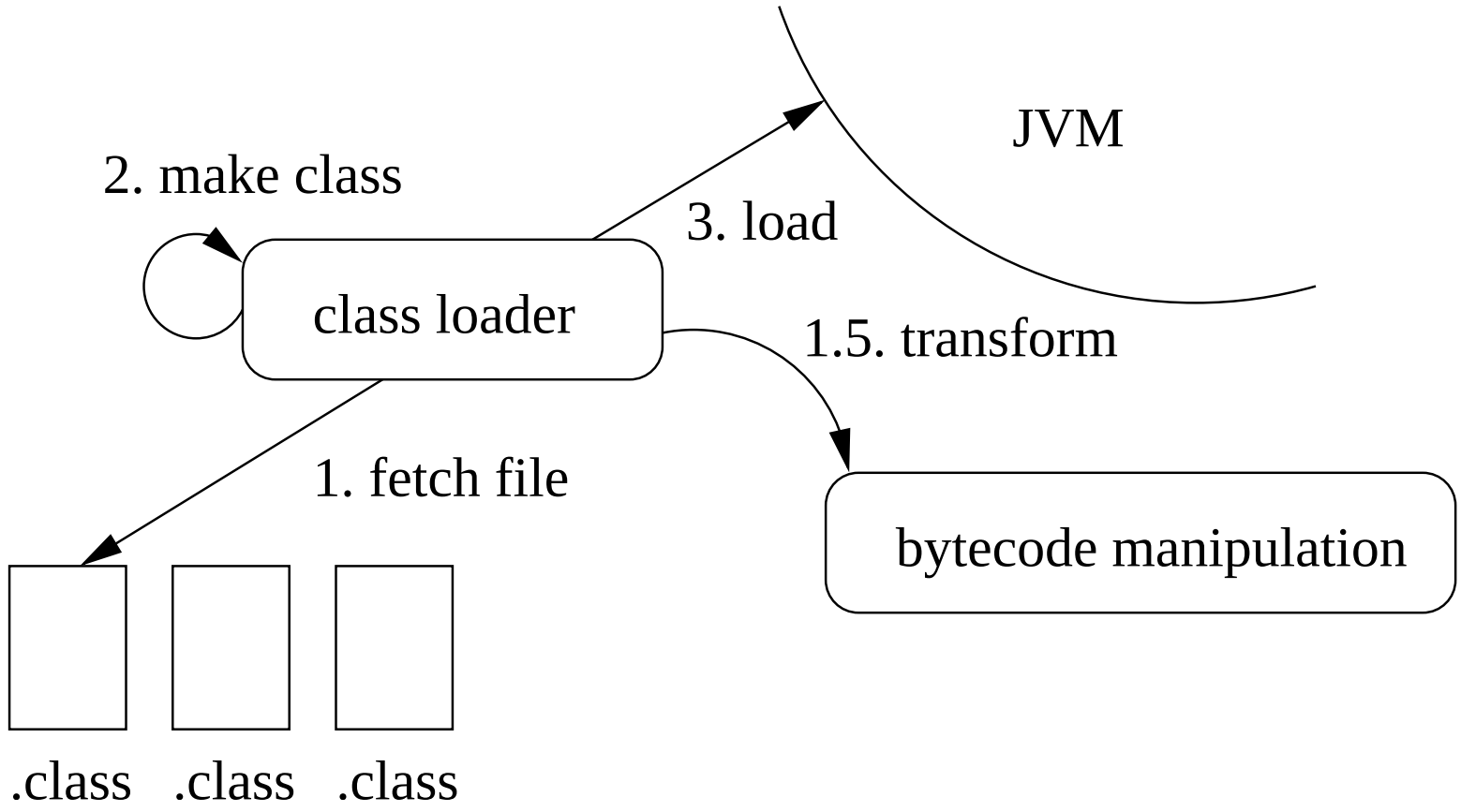
Our context: Java

- class model is closed
- keep portability
- no need for source code
- $\Longrightarrow$ bytecode transformation

Java load time



Load-time transformation



Existing proposals

- bytecode-level abstractions
  - restricted scope proposals: BIT, BCA, . . .
  - general-purpose proposals: BCEL, JikesBT, JOIE
    - intermediate representation
    - powerful but low-level
- source-level abstractions
  - run-time MOPs: Reflex, Kava
  - load-time MOP: Javassist
  - Javassist’s code converter

An example (unsatisfied)

install a factory pattern

- factory method:

```
public Object getInstance(String classname, Object[] args)...
```

- replace:

$$\texttt{new Point}(1,2); \quad \Longrightarrow \quad \texttt{Factory.getInstance}(\text{"Point"}, [1,2]);$$

- code converter of Javassist is not general enough:

$$\texttt{new Point}(1,2); \quad \Longrightarrow \quad \texttt{Factory.getPoint}(1,2);$$

- not trivial with low-level tools

Our proposal: Jinline

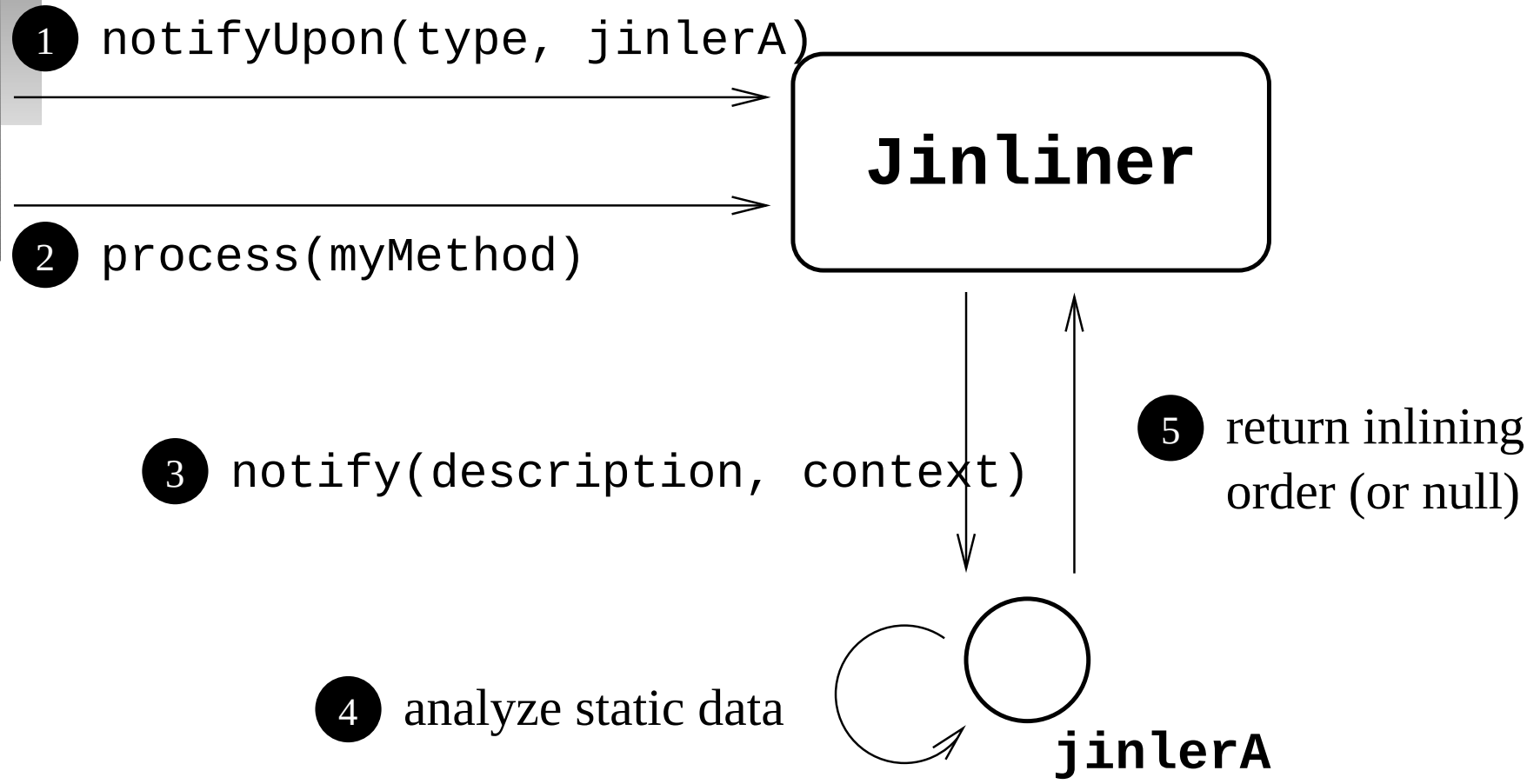
Overview

- unit of transformation: a method
- triggers: occurrences of language mechanisms
- transformation: inline a piece of code before/after/instead
- “piece of code” = a compiled method body (Javassist)
- information provided:
  - static info about the occurrence (drive inlining process)
  - run-time info is packed and passed to the inlined code

Language mechanisms

- message send, constructor send
- field access (read, write)
- array (create, length, access read & write)
- local var access (read, write)
- type identification (casts, RTTI)
- termination (return, throw)
- synchronization (lock grab, release)
- enter method, try, catch

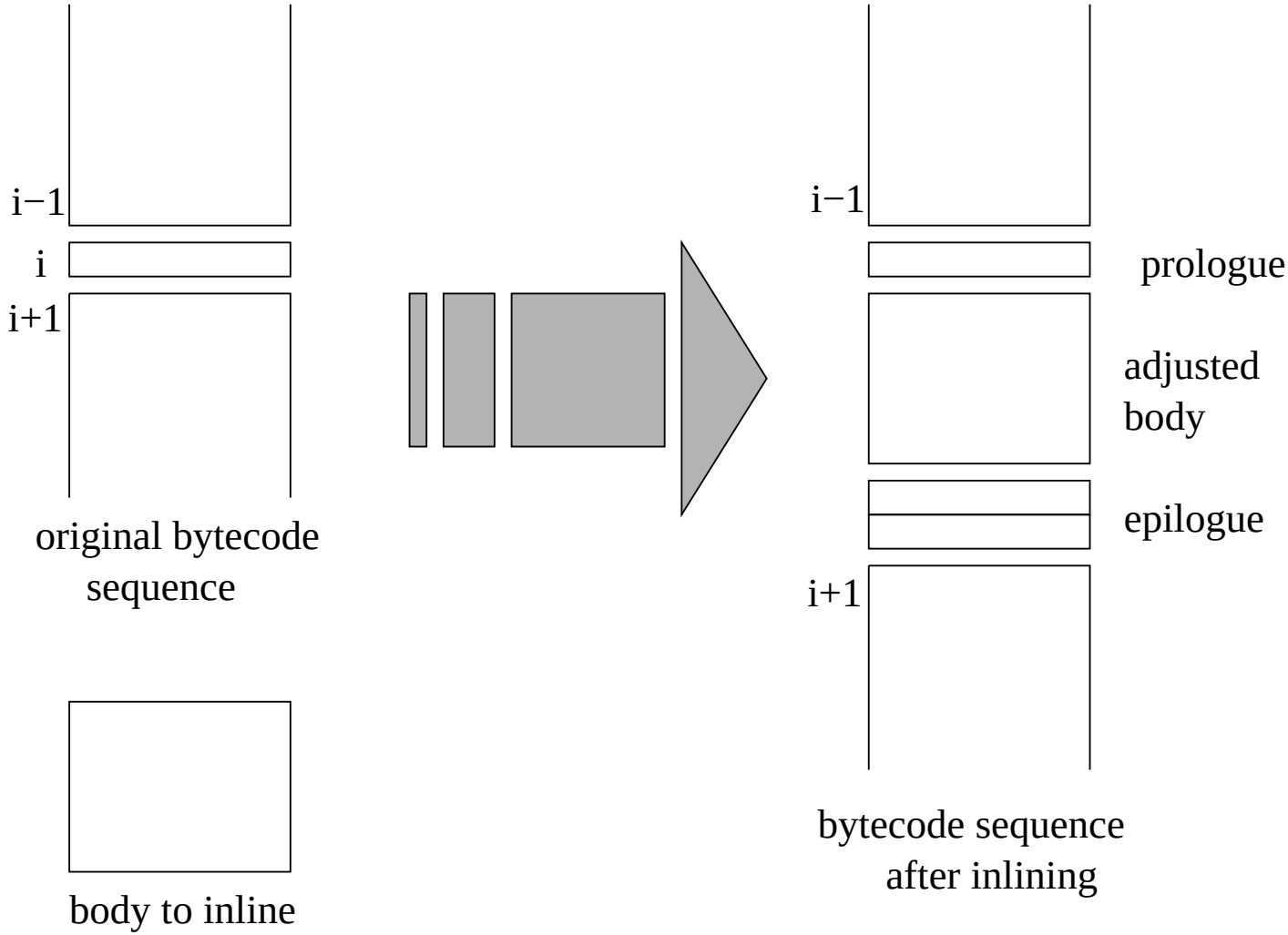
The process


1 notifyUpon(type, jinlerA)
Jinliner
2 process(myMethod)
3 notify(description, context)
5 return inlining order (or null)
4 analyze static data
jinlerA

Implementation

- reification of bytecode: `BytecodeSequence`
- connection with the Javassist framework
- process:
  - `MethodParser` notifies upon occurrences
  - user returns a `ToInline` object
  - `MethodInliner` performs the transformation

Illustration



Back to the example

Code to inline

```
 public class FactorySample {
   public Object newMethod(Object[] jinArgs){
(1)  String classname = (String) jinArgs[2];
(2)  Object[] args = (Object[]) jinArgs[3];
(3)  return Factory.getInstance(classname, args);
   }
 }
```

The Jinler

```
public class FactoryJinler implements Jinler {
  CtMethod newMethod;

  FactoryJinler(){
(1)   newMethod =
        ClassPool.getDefault()
                 .get("FactorySample")
                 .getDeclaredMethod("newMethod");
  }

  public ToInline notify(Description desc, Context cor
(2)   if(desc instanceof ConstructorSend)
(3)     return new ToInline(newMethod);
(4)   return null;
  }}
```

Connecting to Javassist

```
public class FactoryTranslator
      implements javassist.Translator {
  Jinliner inliner = new Jinliner();
  Jinler jinler = new FactoryJinler();

  public void start(ClassPool pool){
(1)  inliner.notifyUpon(ConstructorSend.class, jinler);
  }

  public void onWrite(ClassPool pool, String classname)
(2)  CtClass clazz = pool.get(classname);
(3)  inliner.process(clazz);
  }
}
```

Conclusion

Achievements

- fine-grained alterations
- high-level abstractions
- simple and powerful
- complements Javassist

Future work

- code explosion problem
  - optimizing generated code
  - use of subroutines
  - straightforward optimization
- application
  - library of transformers for Reflex