A Framework for Optimizing Java Using Attributes

Patrice Pominville

Feng Qian

Raja Vallée-Rai

Laurie Hendren

McGill University

Clark Verbrugge

IBM Toronto Lab

www.sable.mcgill.ca

Outline

- Class File Attributes and Soot
  - What are attributes in Java Class Files?
  - What other information can be conveyed in Attributes?
  - An introduction to the Soot framework
  - Attributes in Soot
- Case Study: Array Bounds Check Elimination
  - Analyses
  - Attributes
  - Modifying JVM to be aware of attributes
  - Experimental Results
- Conclusion

Java `.class` files

A Java `class` file contains *fields*, *methods* and *attributes*.

Fields: instance variables or class variables.

Methods: contain Java bytecode

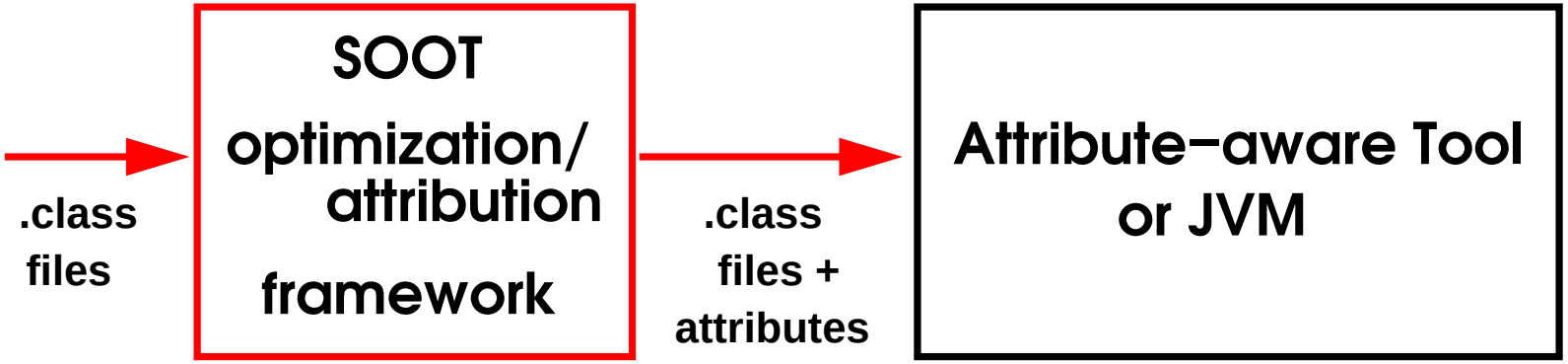
```
// Java source
int cc (int x, int y)
{
  int z;
  z = x * y;
  return z;
}
```

```
// Bytecode
Method int cc (int, int)
   0 iload 1
   1 iload 2
   2 imul
   3 istore 3
   4 iload 3
   5 ireturn
```

Classes, methods and fields also have *attributes*.

Why Use Attributes?

- bytecode is relatively high-level and limited expressiveness for low-level optimizations;
- classfiles have **attributes** that provide space for arbitrary data;
- we can convey information to a JVM or other tool using attributes.



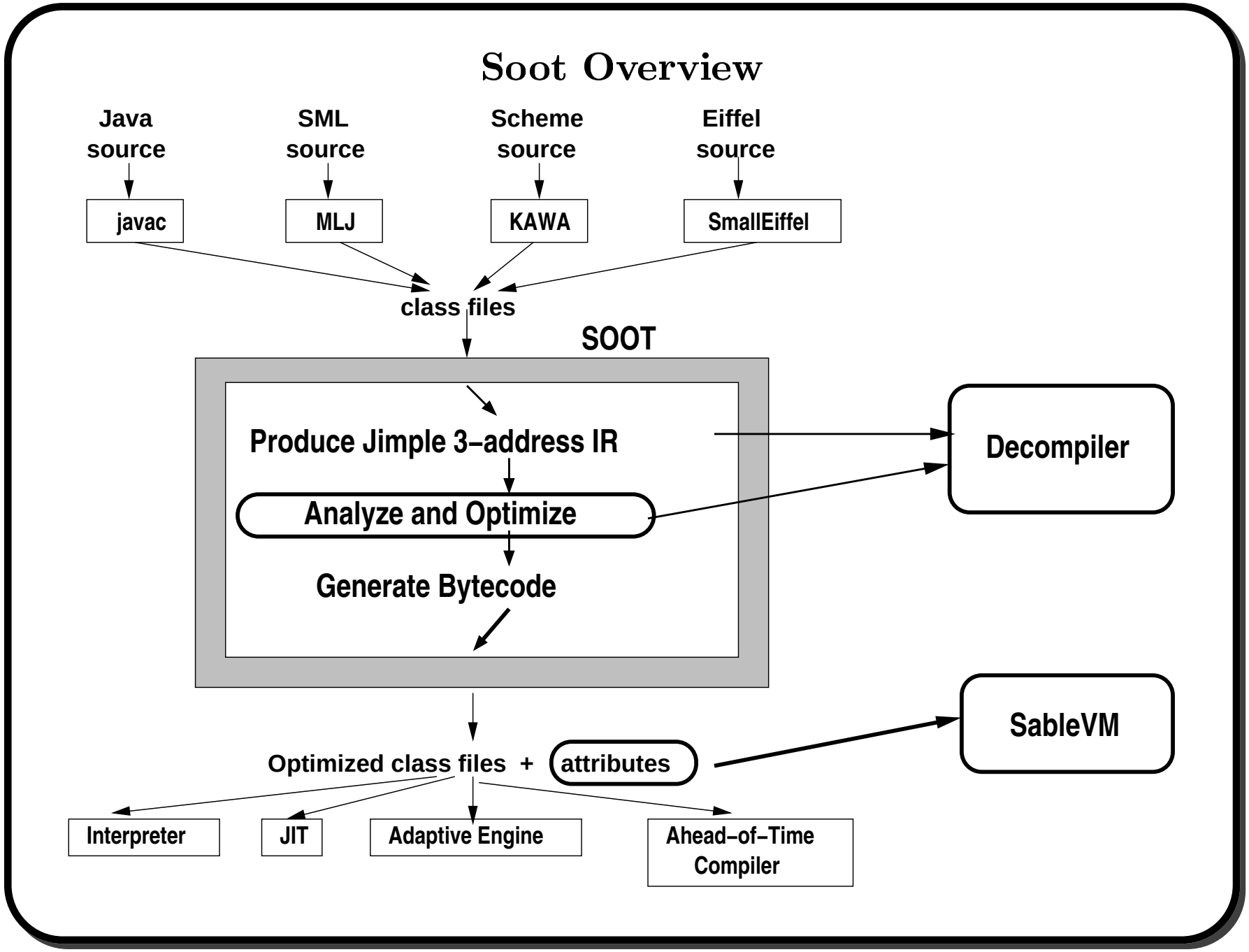
Uses of Attributes

- Attributes to convey static program analysis:
  - array bounds check elimination
  - register allocation
  - stack allocation of objects
- Attributes to convey hints or profiling information:
  - hot methods
  - lifetime of objects
  - branch prediction annotation

Class File Attributes

- attributes for `ClassFile`,`field_info`, `method_info` and `Code_attribute` structures;
- code is actually an attribute of a method;
- standard attributes include: `SourceFile`, `ConstantValue`, `Exceptions`, `LineNumberTable` and `LocalVariableTable`.
- Format of attributes:

```
attribute_info {
  u2 attribute_name_index;
  u4 attribute_length;
  u1 info[attribute_length];
}
```


Soot Overview
Java source
SML source
Scheme source
Eiffel source
javac
MLJ
KAWA
SmallEiffel
class files
SOOT
Produce Jimple 3–address IR
Analyze and Optimize
Generate Bytecode
Decompiler
SableVM
Optimized class files +
attributes
Interpreter
JIT
Adaptive Engine
Ahead–of–Time Compiler

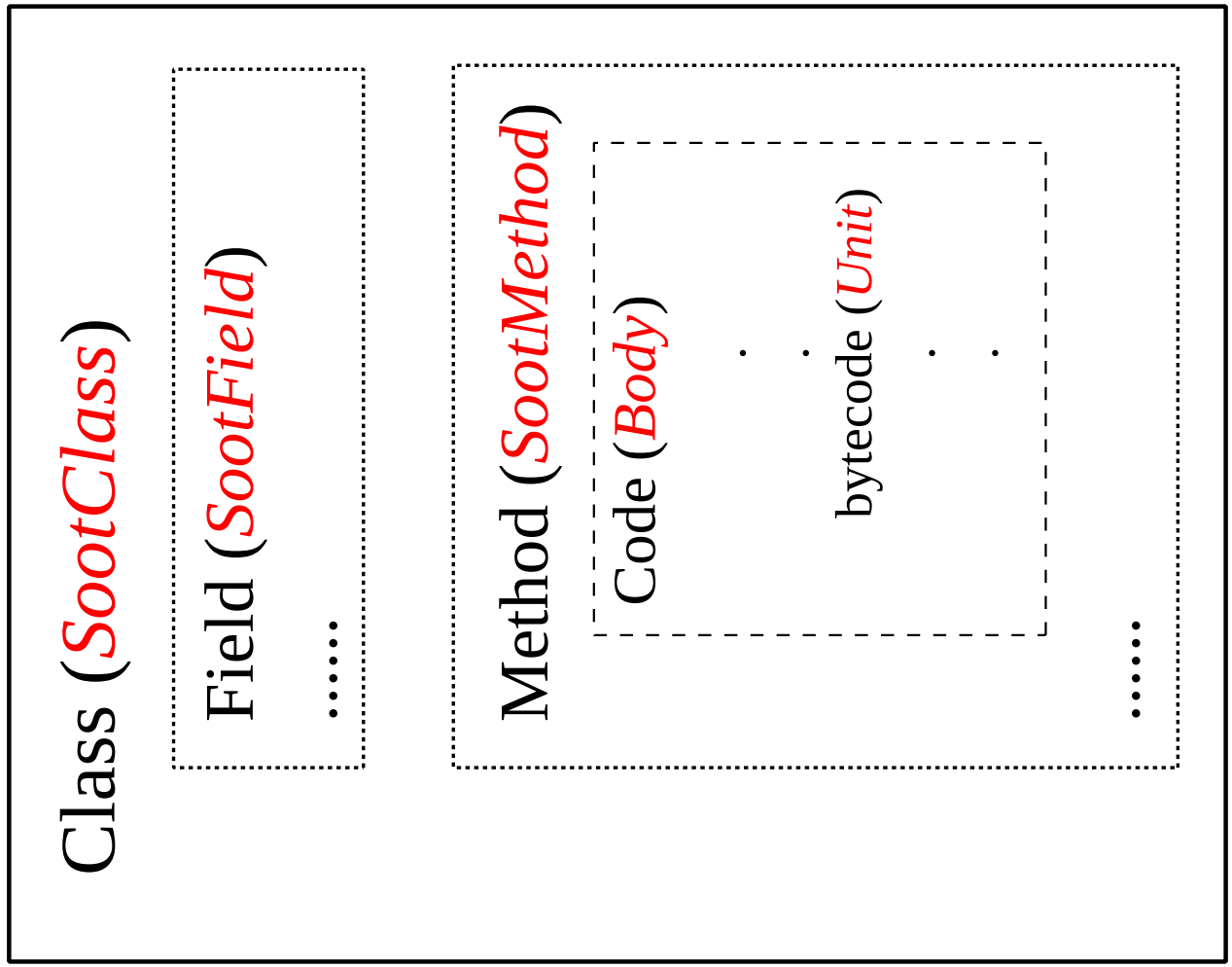
Why classfile optimization/annotation?

- Optimizing class files allows our tool to be independent of front-end compiler and back-end JVM.
- Java bytecode contains enough high-level information so that we can retrieve all important information for program analysis.
- Adding attributes to classfiles gives us a mechanism of transmitting program analysis information to a JVM or another tool.

Why Soot?

- Provide an API for general research use, including support for important intermediate representations.
- Goal is have an infrastructure in which competing analyses can be implemented and evaluated.

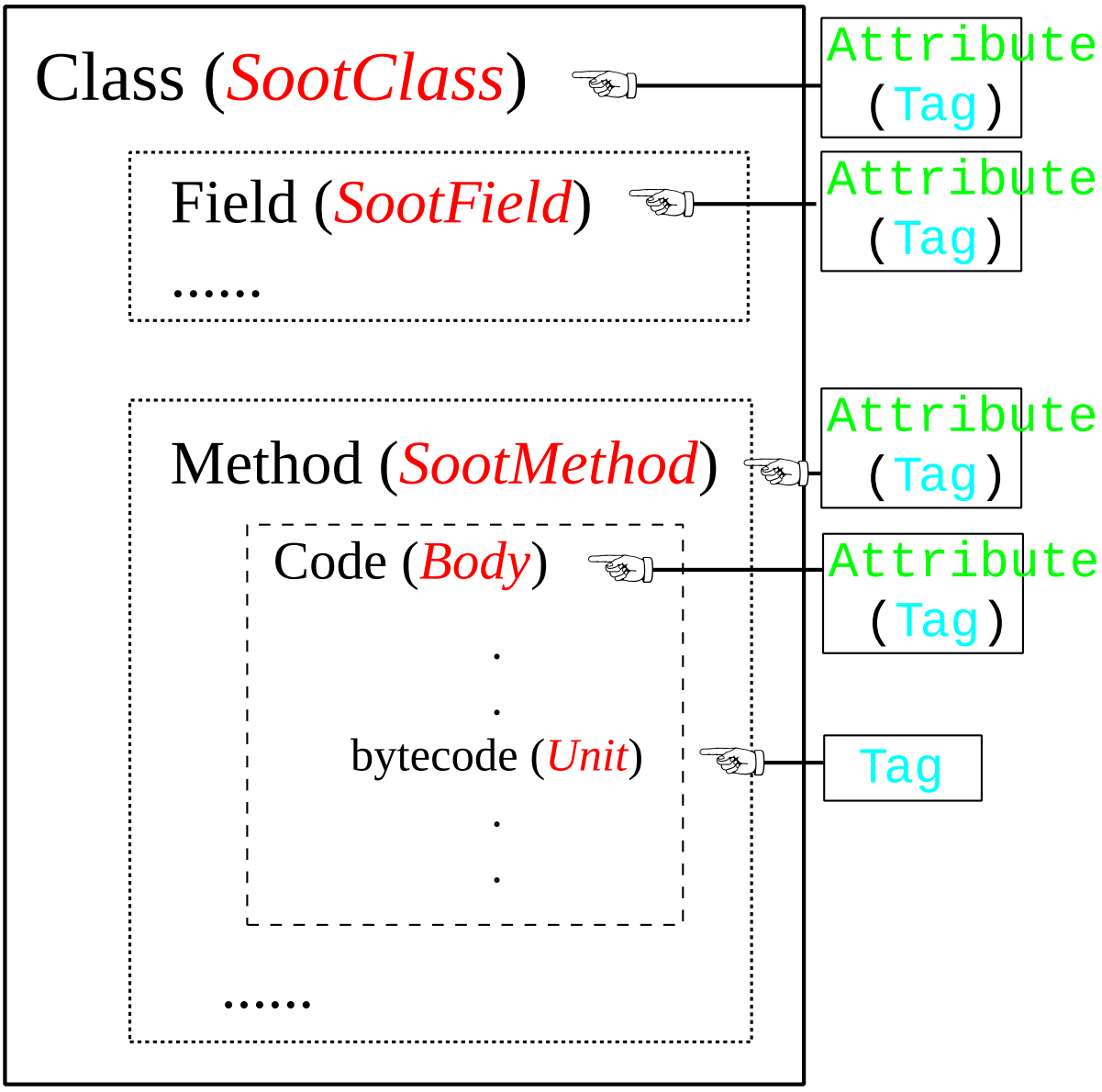
Soot Structures



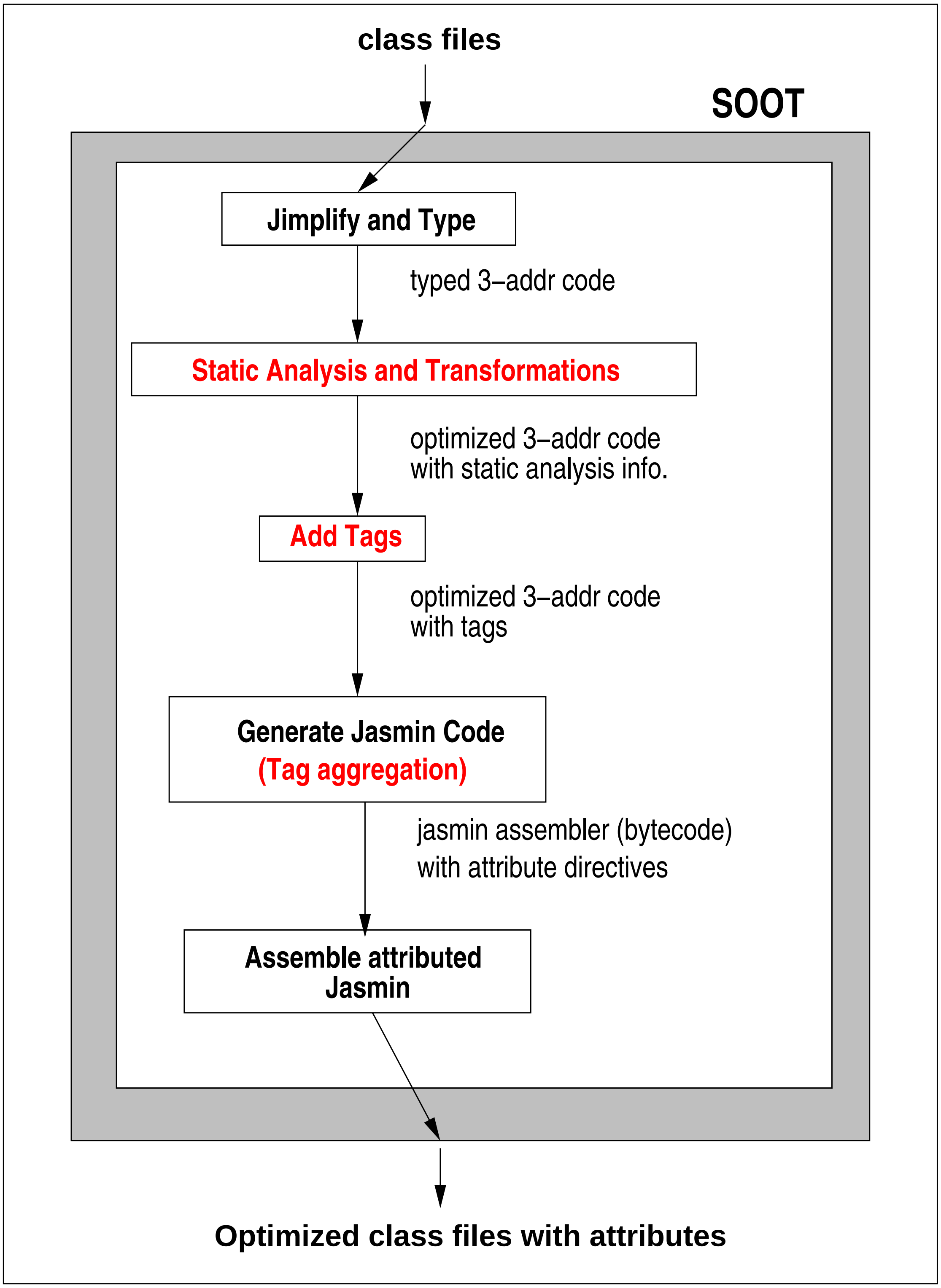
```
public interface Host {
  /* gets list of tags associated with the host.*/
  public List getTags();
  /* gets a tag by name. */
  public Tag getTag(String aName);
  /* adds a tag to the host. */
  public void addTag(Tag t);
  /* removes a tag by name. */
  public void removeTag(String name);
  /* checks if a tag exists.*/
  public boolean hasTag(String aName);
}
public interface  Tag {
  public String getName();
  public byte[] getValue();
}
```

Attributes in Soot

Hosts, Tags, Attributes



Attributes inside Soot



Outline

- Class File Attributes and Soot
  - What are attributes in Java Class Files?
  - What other information can be conveyed in Attributes?
  - An introduction to the Soot framework
  - Attributes in Soot
- Case Study: Array Bounds Check Elimination
  - Analyses
  - Attributes
  - Modifying JVM to be aware of attributes
  - Experimental Results
- Conclusion

Step 1: Develop Analyses for Array Bounds Check Elimination

- Intraprocedural flow-sensitive analysis to build an inequality constraint graph for each program point.
- Intraprocedural flow-sensitive analysis to determine nullness of object references.
- Analysis of each class to find fields with constant length.
- Flow-insensitive whole program analysis to find rectangular arrays.

Note: all analyses implemented on the Jimple IR in Soot.

Step 2: Develop Tags and Aggregators

1. Create an `ArrayCheckTag` class that implements the `Tag` interface.
2. (a) For each array reference, create an `ArrayCheckTag` object; and
   (b) attach the `ArrayCheckTag` to the Jimple statement which is acting as the `Host`.
3. Create a `ArrayCheckTagAggregator` class which implements the `TagAggregator` iterface.
4. Register the aggregator with the `CodeAttributeGeneratorClass` and specify this aggregator as `active`.

Soot produces attributed Jasmin assembler code

```
public void sum(int[] a) {
  int total=0;
  int i=0;
  for (i=0; i<a.length; i++)
    total += a[i];
  int c = a[i];
}
```

(a) Java Source

```
.method public sum([I)V
   ...
label2:
    iaload
    ...
label3:
    iaload
    return

.code_attribute
      ArrayNullCheckAttribute
     "%label2%AA==%label3%Aw=="
.end method
```

(b) attributed Jasmin

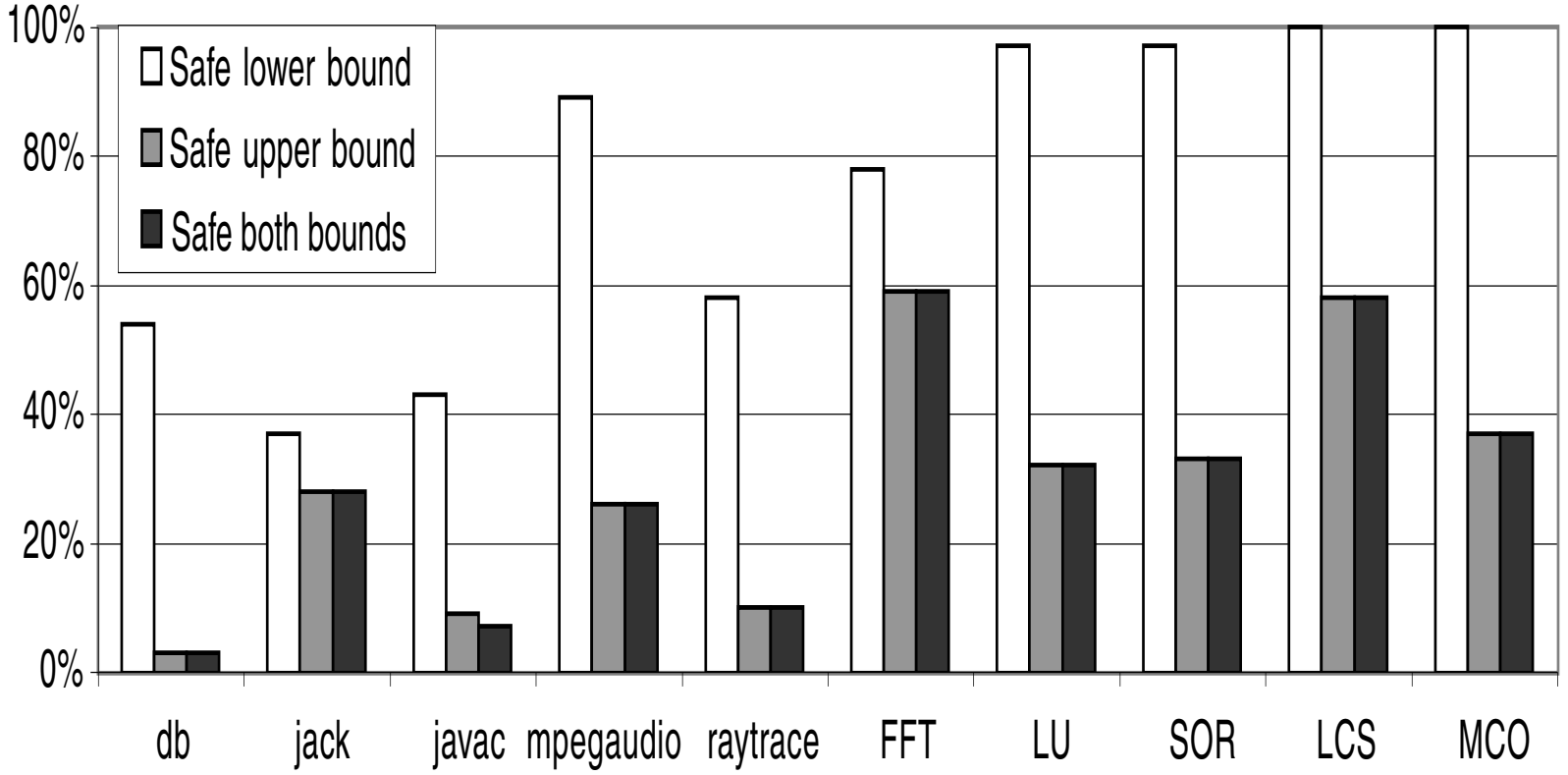
Step 3: Make a JVM aware of attributes

- read attribute table (entries sorted by PC);
- modify JVM
  - Kaffe VM 1.0.5 (JIT engine 3)
  - IBM HPCJ
  - *SableVM*

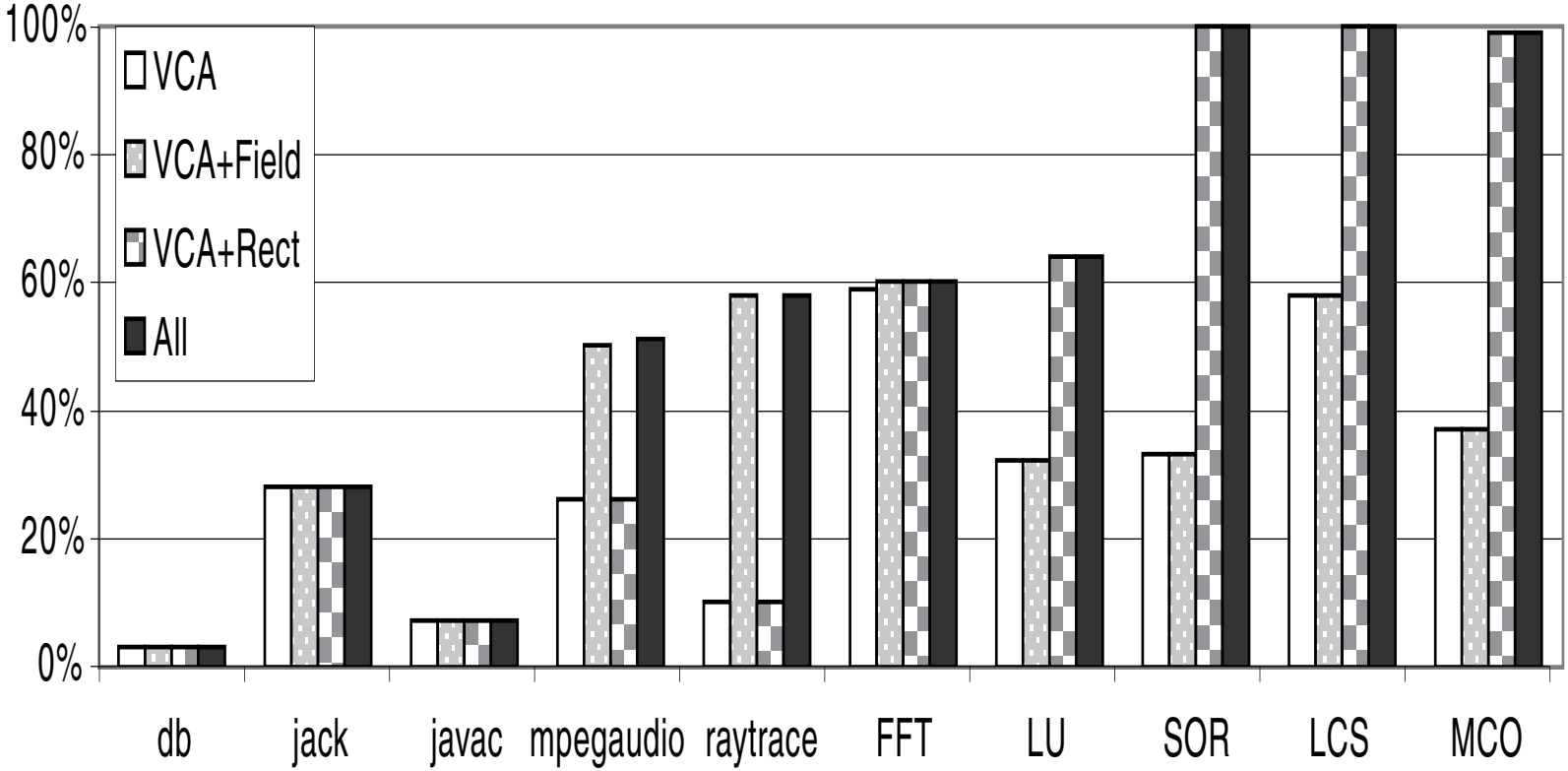
Kaffe Modifications

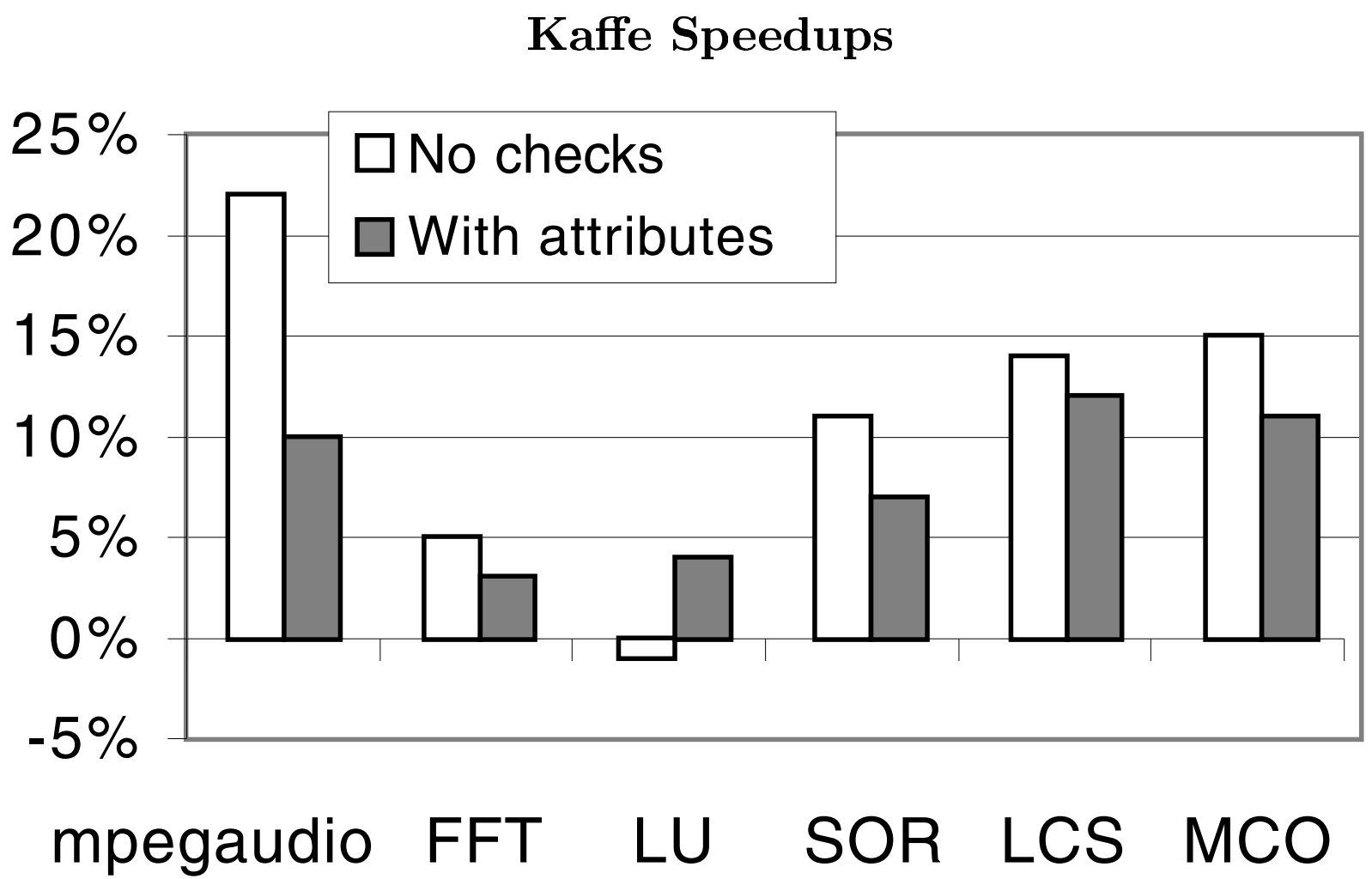
```
idx = 0;
...
case IALOAD:
...
  if (attr_table_size > 0) { /* attributes exist. */
    attr = entries[idx].attribute;
    idx++;
    if (attr & 0x03)  /* generates bounds checks. */
       check_array_index(..);
    else
    if (attr & 0x04) /* gen null pointer check. */
        explicit_check_null(..);
  } else   /* normal path */
    check_array_index(..);
```

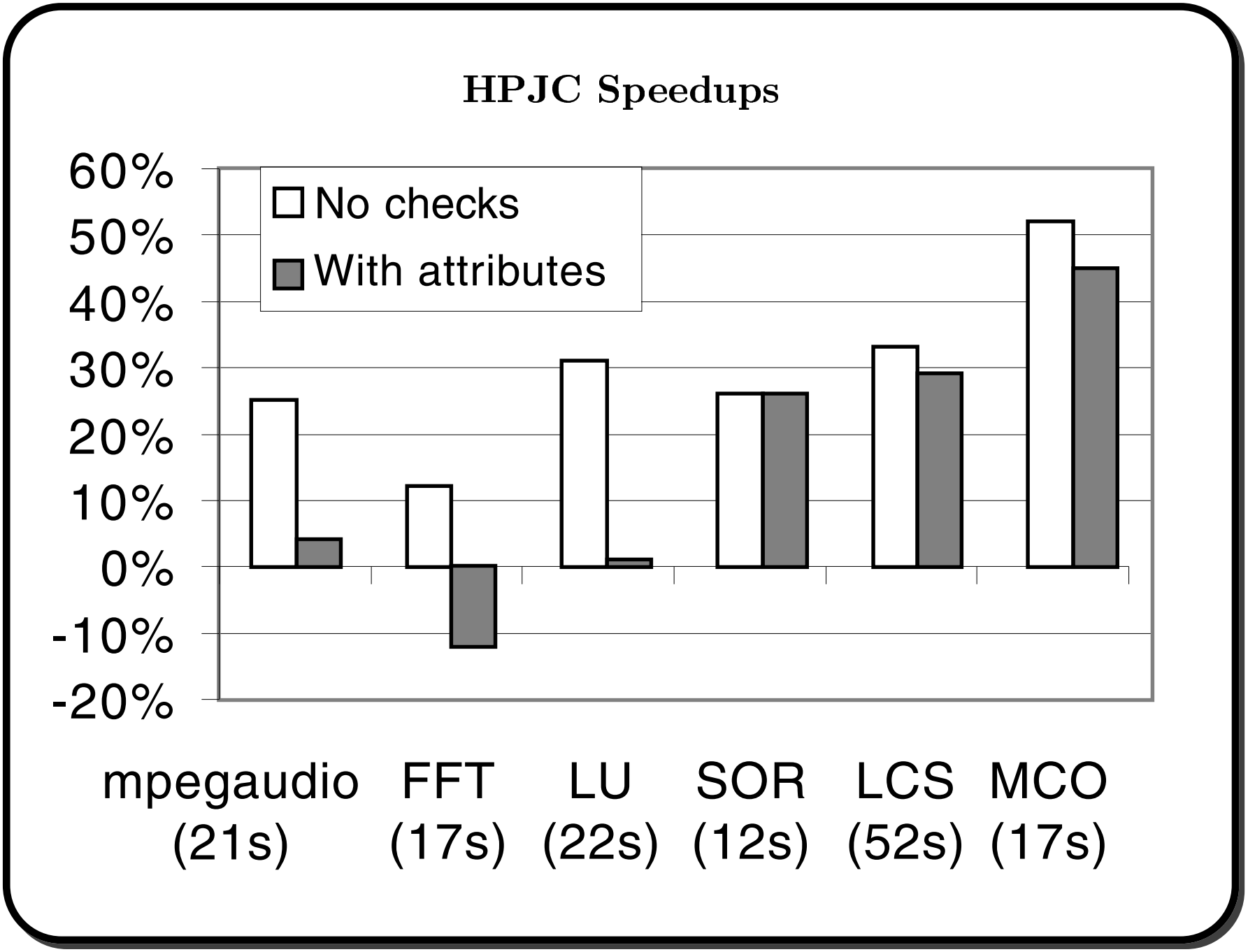
Results of base analysis



Improvements due to field and shape analysis




Kaffe Speedups
25%
20%
15%
10%
5%
0%
-5%
No checks
With attributes
mpegaudio
FFT
LU
SOR
LCS
MCO


HPJC Speedups
60%
50%
40%
30%
20%
10%
0%
-10%
-20%
No checks
With attributes
mpegaudio (21s)
FFT (17s)
LU (22s)
SOR (12s)
LCS (52s)
MCO (17s)

Conclusions

- Have provided a framework for producing attributes for Java class files.
- Can convey static analysis information, thus the analysis can be more expensive than is possible in a JIT.
- Can be used as a way of experimenting with an analysis over many JVMs, or as a way of providing information for other tools.
- Can lead to significant speedups.
- Attributes can be used for other purposes; debugging, watermarking, etc.

Limitation

- For use in secure systems, how can we trust annotated classfiles?

Related Work

- Bytecode optimizers: Cream, Jax, Bloat
- Bytecode manipulation tools: JTrek, Joie, Bit and JavaClass
- Java application packagers: Jax, DashO-Pro, SourceGuard
- Other work on attributes: Jones and Kamin; Hummel, Azevedo, Kolson and Nicolau.

www.sable.mcgill.ca

Download Soot or other Sable Tools

- Soot - a bytecode optimizer/attributer
  www.sable.mcgill.ca/soot
- SableCC - an object-oriented compiler compiler
- SableVM - a portable JVM written in C

Soot and other Sable Group Publications

- papers, reports and theses
  www.sable.mcgill.ca/publications/
- tutorial on attributes
  www.sable.mcgill.ca/soot/tutorial/addattributes/