Mixing Source and Bytecode

A Case for Compilation by Normalization

OOPSLA'08, Nashville, Tennessee

Lennart Kats, TU Delft
Martin Bravenboer, UMass
Eelco Visser, TU Delft

October 21, 2008

Software Engineering Research Group

TU Delft
Delft University of Technology

Language Extensions

DSLs and Language Extensions

Domain-specific

- Database queries
- Regular expressions
- XML processing
- Matrices
- Complex numbers
- …

```
void browse() {
    List<Book> books =
      <| SELECT *
         FROM books
         WHERE price < 100.00
       |>;

    …

    for (int i = 0; i < books.size(); i++)
        books.get(i).title =~ s/^The //;
}
```

DSLs and Language Extensions

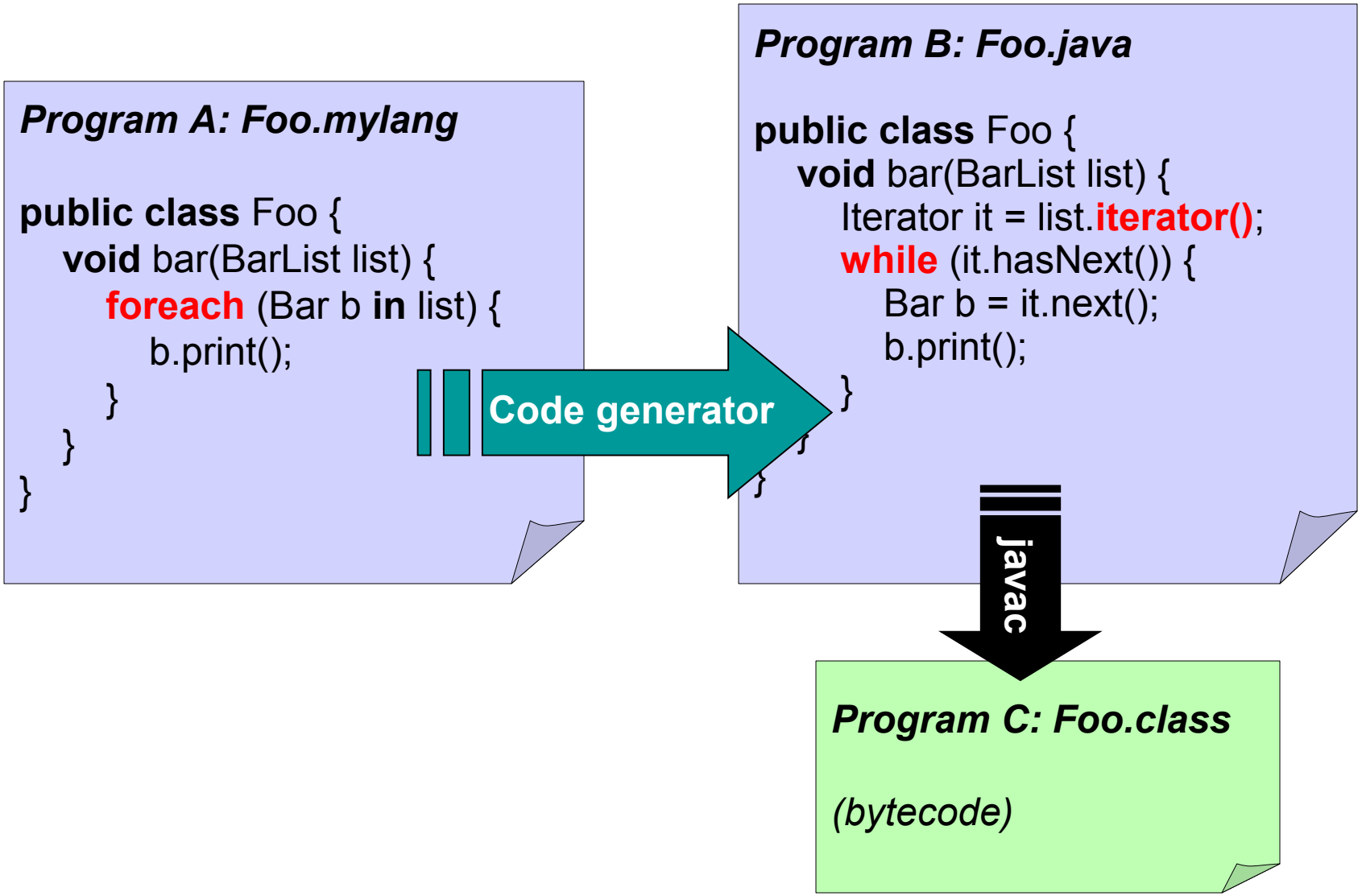
Domain-specific

- Database queries
- Regular expressions
- XML processing
- Matrices
- Complex numbers
- ...

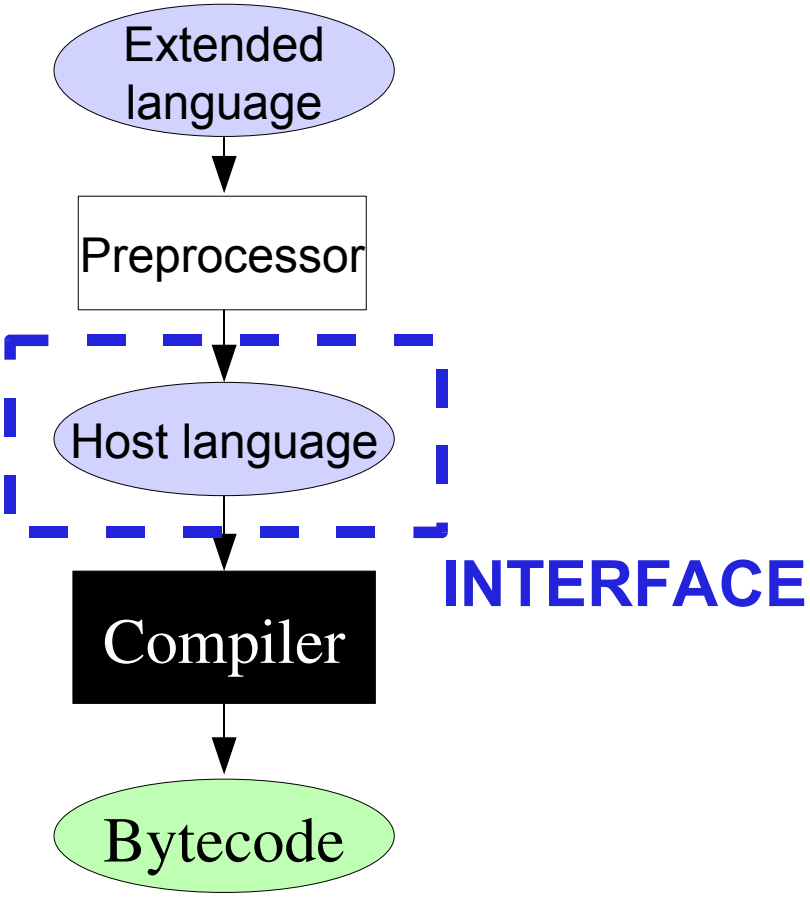
General-purpose

- AOP
- Traits
- Enums
- Iterators
- Case classes
- Properties/accessors
- ...

Example: The Foreach Statement

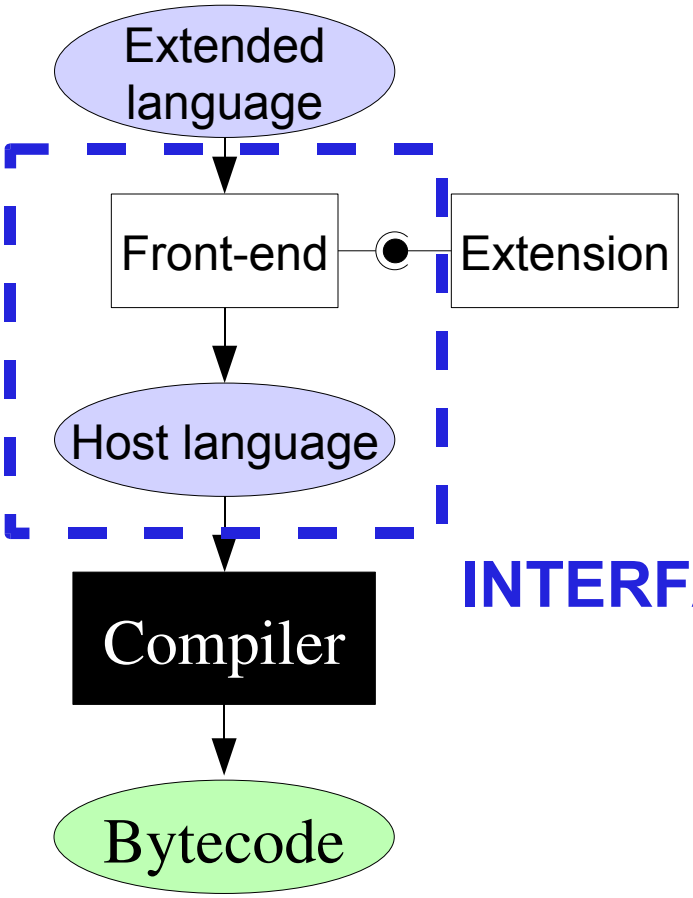


The Preprocessor



+ loosely coupled
+ small, modular extension
+ no need to know compiler internals
– no parser
– semantic analysis, errors?

The Front-end Extensible Compiler



+ front-end reuse

– tied to the front-end internals

INTERFACE

Polyglot

MetaBorg

ableJ

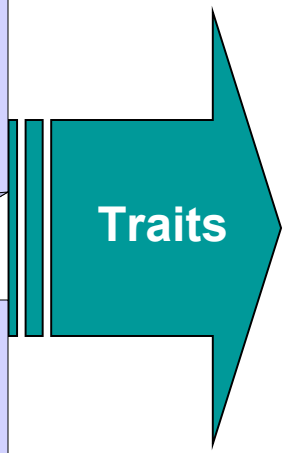
OJ

Traits: Composable Units of Behavior

[Schärli, Ducasse, Nierstrasz, Black 2003]

```
public trait TDrawing {
    void draw() {
        …
    }

    require Vertices getVertices();
}
```

```
public class Shape with TDrawing {
    …

    Vertices getVertices() { … }
}
```

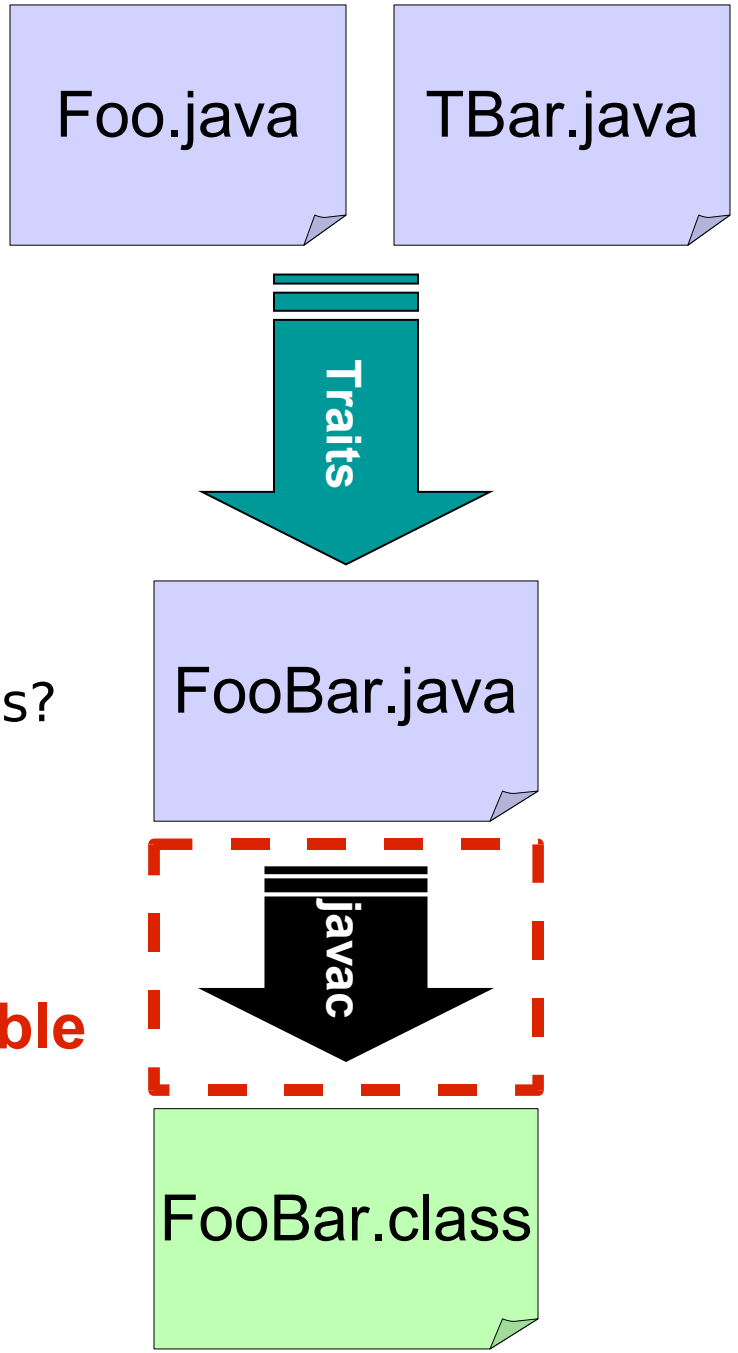



```
public class Shape {
    void draw() { … }

    Vertices getVertices() { … }
}
```

Traits

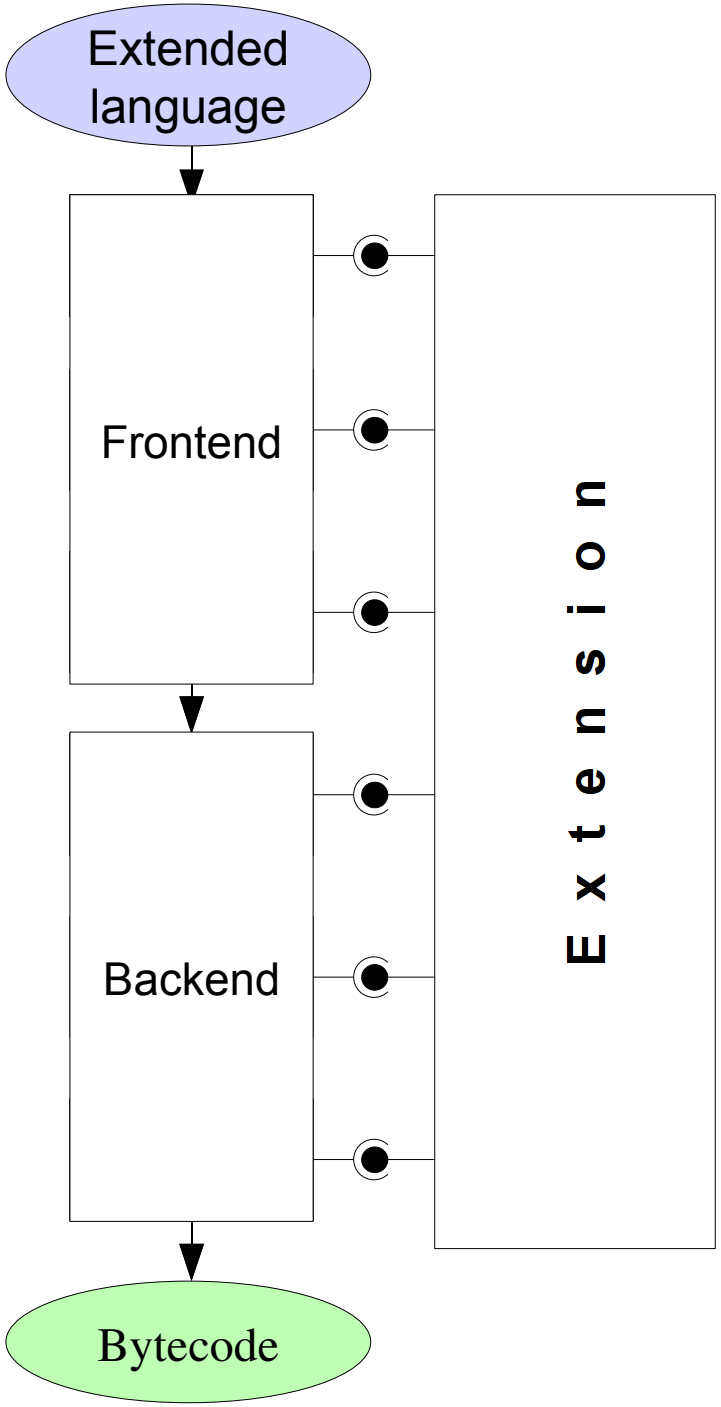
- Simple idea
- Most libraries are distributed as *compiled* code

→ Why not do this for traits?



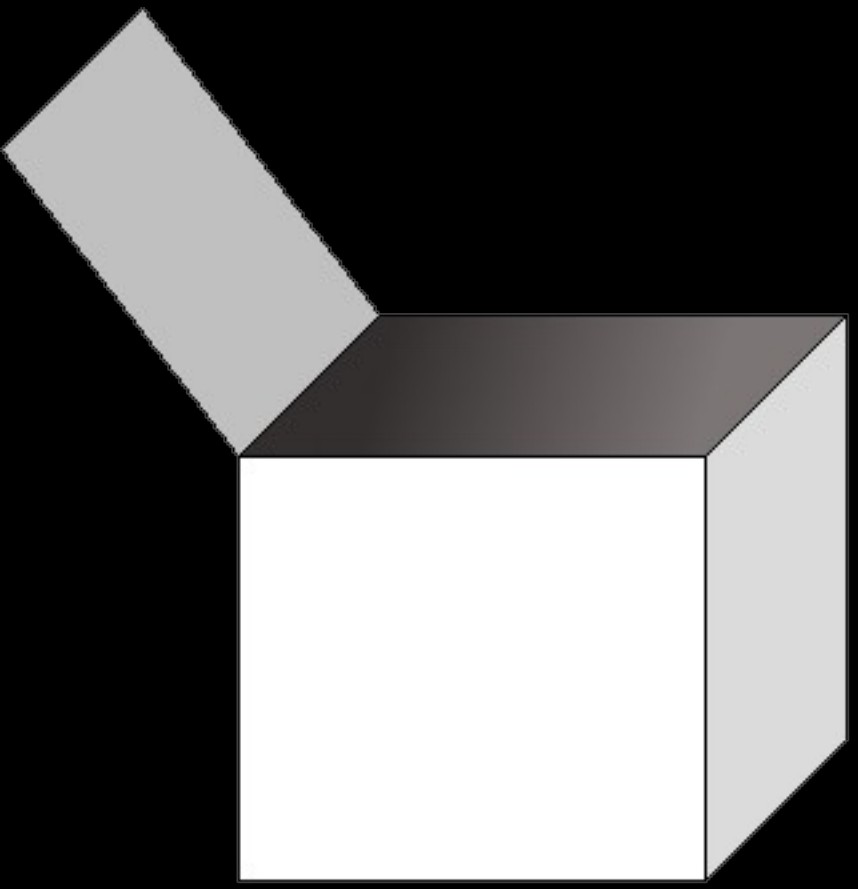
Not extensible

The Fully Extensible Compiler?



The Fully Extensible Compiler?

+ reuse *everything*
+ access to back-end
–– tied to *all* compiler internals
– complexity
– performance trade-offs



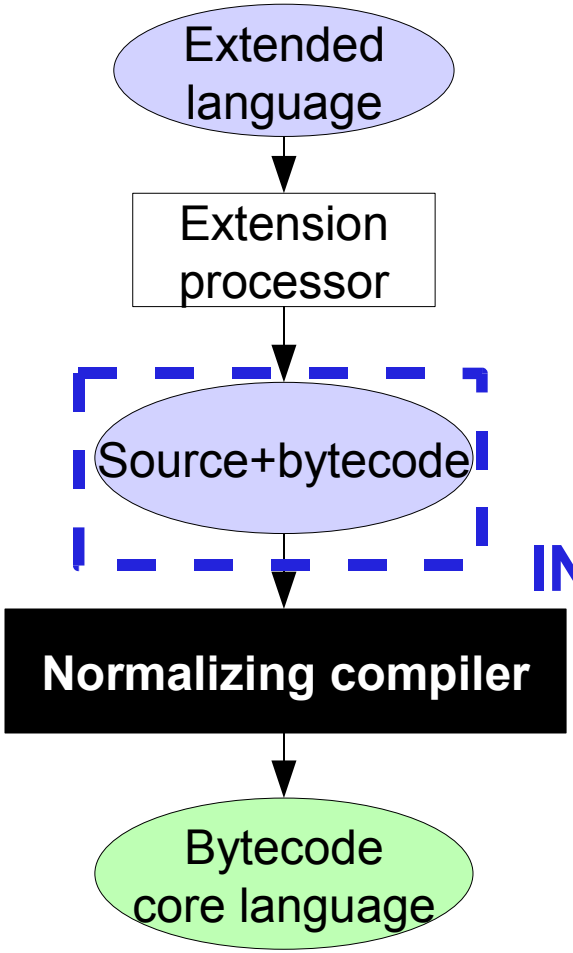
The White Box Model?

–– tied to *all* compiler internals

– complexity

– performance trade-offs

The Normalizing Compiler

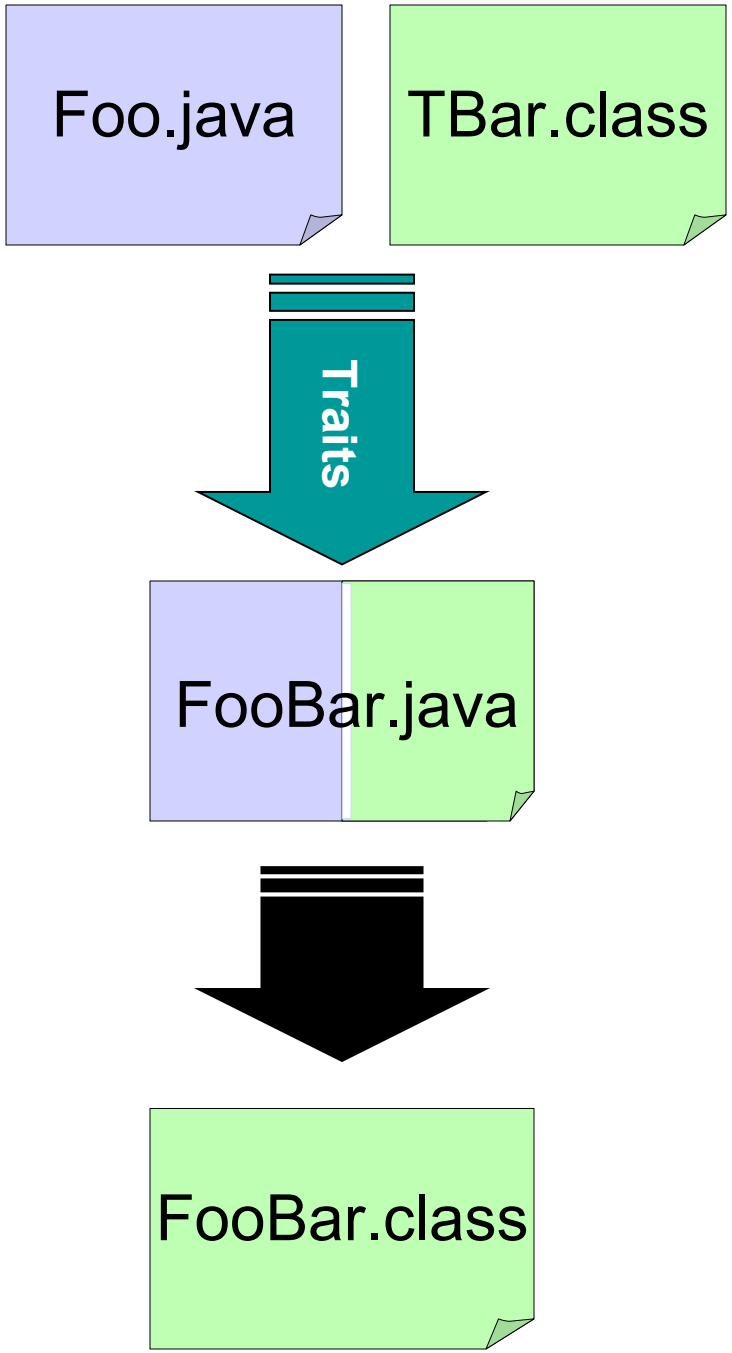


\+ simple interface

\+ access to back-end

++ not tied to compiler internals

Traits, Revisited



Java, and Bytecode, and Backticks, Oh, My!

```
class FooBar {
```

Java (Foo.java)

```
  void hello1() {
      System.out.println("Hello Java world");
  }
```

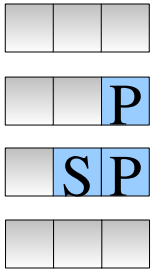
Bytecode (TBar.class)

```
  `hello2 (void) [
      getstatic java.lang.System.out : java.io.PrintStream;
      ldc "Hello bytecode world";
      invokevirtual java.io.PrintStream.println(java.lang.String : void);
  ]

}
```

Java, and Bytecode, and Backticks: Fine-grained Mixing

```
class FooBar {

    void hello3() {
        System.out.println(`ldc "Hello mixed world");
    }

    `hello4 (void) [
        getstatic System.out : java.io.PrintStream;
        push `"Hello " + "Java".concat("bytecode world");
        invokevirtual java.io.PrintStream.println (java.lang.String : void);
    ]

}
```

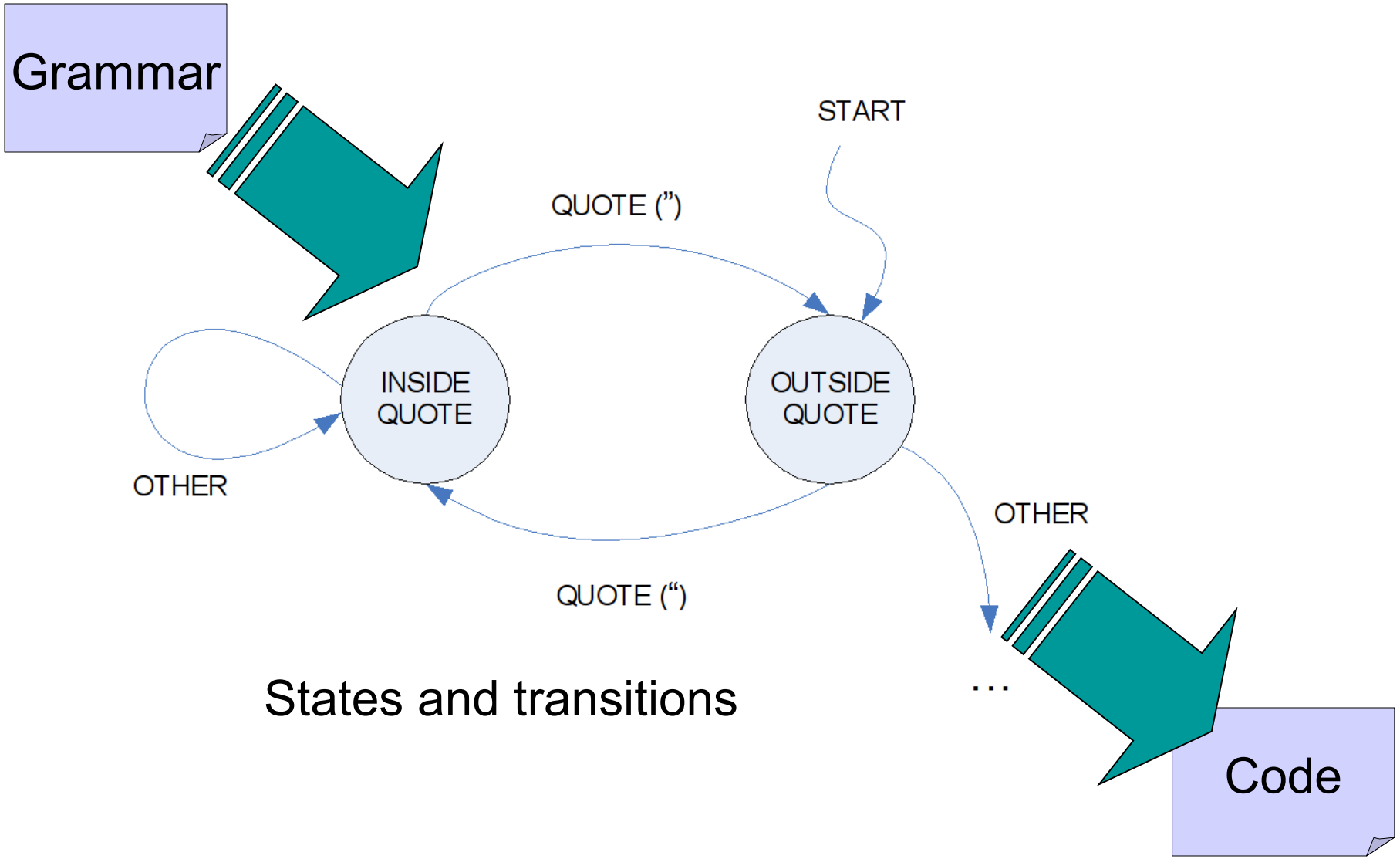
Typechecker



Typical Applications

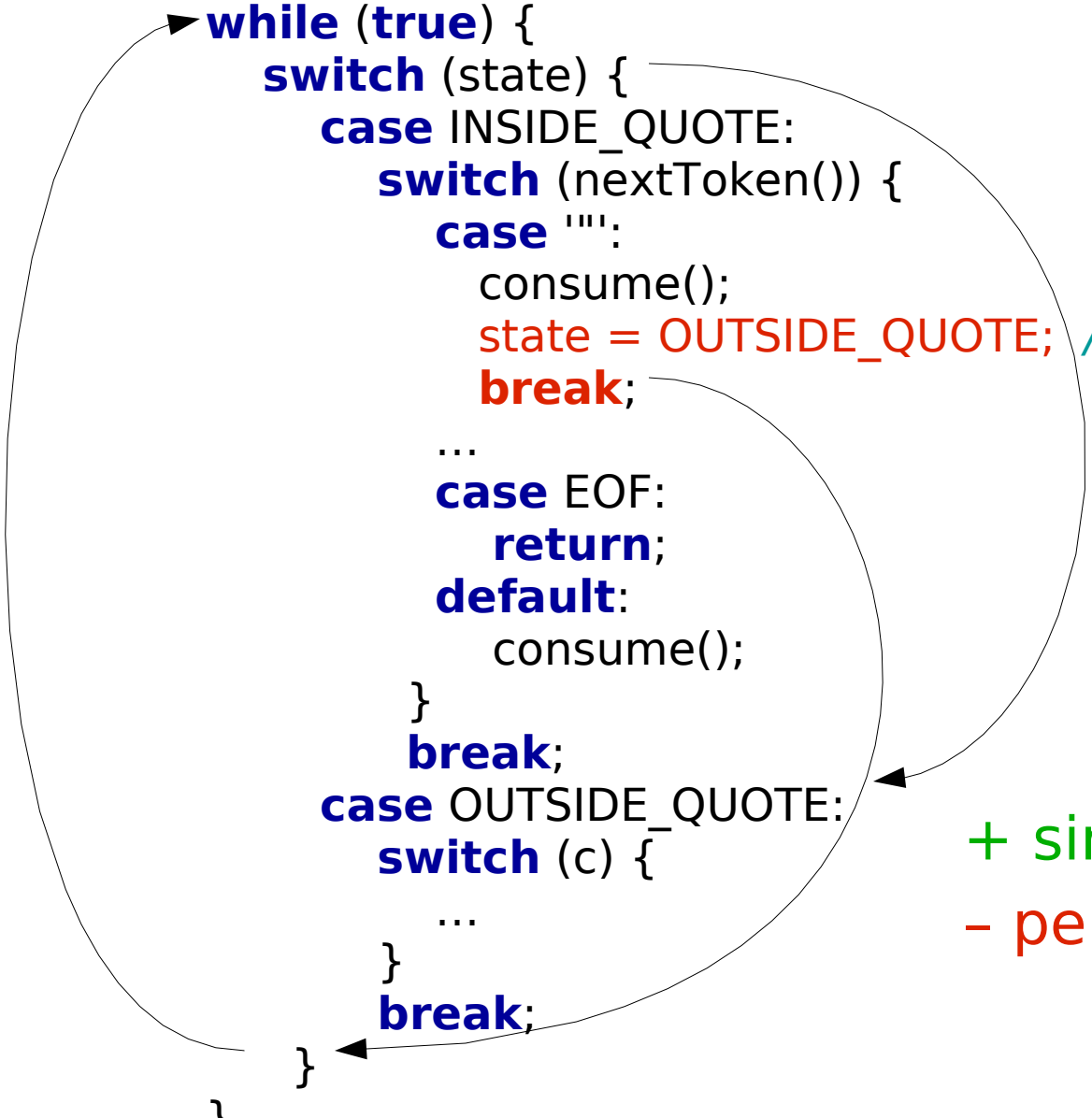
	Class-method	Statement	Expression level
Traits	X		
Partial/open classes	X		
Iterator generators		X	
Finite State Automata		X	
Expression-based extensions			X

Example: Finite State Automata (DFAs)



States and transitions

Finite State Automata: Generated Java code



```
while (true) {
    switch (state) {
        case INSIDE_QUOTE:
            switch (nextToken()) {
                case '"':
                    consume();
                    state = OUTSIDE_QUOTE;  // end quote found!
                    break;
                ...
                case EOF:
                    return;
                default:
                    consume();
            }
            break;
        case OUTSIDE_QUOTE:
            switch (c) {
                ...
            }
            break;
    }
}
```

+ simple Java code

– performance trade-off

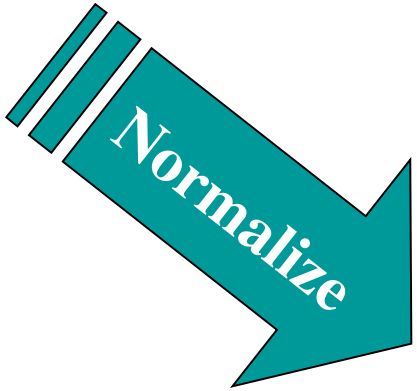
Finite State Automata: Inline Bytecode

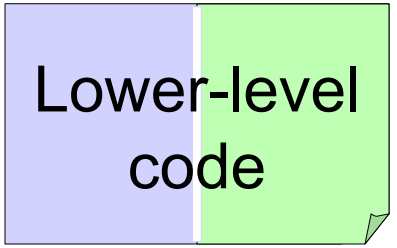
```
InsideQuote:
    switch (nextToken()) {
        case '"':
            consume();
            `goto OutsideQuote;
        case EOF:
            return;
        default:
            consume();
            `goto InsideQuote;
        }
    }
OutsideQuote:
    switch (c) {
        ...
    }
```

+ performance
+ low-level when necessary
+ minimal changes
+ maintainable

Method-body Extensions

- Operators: x **as** T, x **??** y, ...
- Embedded (external) DSLs: Database queries, XML, ...
- Closures





Stratego Normalization Rules

desugar-foreach:

```
|[ foreach (t x in e) stm ]|
 →
|[ Iterator x_iterator = e.iterator();
   while (x_iterator.hasNext()) {
      t x = x_iterator.next();
      stm;
   }
]|
with x_iterator := <newname> “iterator”
```

Normalizing ??

```
String property = readProperty("x") ?? "default";
System.out.println(property);

// In basic Java:

String temp = readProperty("x");
if (temp == null) temp = "default";
String property = temp;
System.out.println(property);
```

Normalizing ??

“But placing statements in front of the assimilated expressions is easy, isn’t it?”

Requires **syntactic** knowledge:

- for, while, return, throw, …
- other expressions
- other extensions

Normalizing ??

“But placing statements in front of the assimilated expressions is easy, isn’t it? … Right?”

Also requires **semantic knowledge**:

```
Iterator<String> it = ...;



for (String e = e0; it.hasNext(); e = it.next() ?? “default”) {
    ...
}
```

Normalizing ??

“But placing statements in front of the assimilated expressions is easy, isn’t it? … Right?”

Also requires **semantic knowledge**:

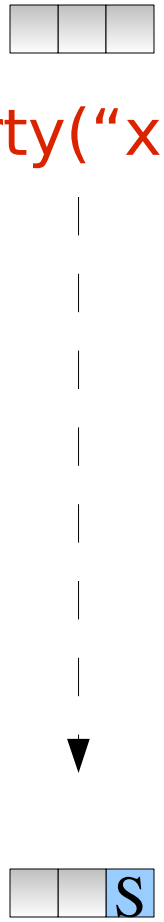
```
Iterator<String> it = ...;

Integer temp = ...;
for (String e = e0; it.hasNext(); e = temp) {
    ...
}
```

Normalizing '??' !!

```
String property = readProperty(“x”) ?? “default”;
System.out.println(property);
```

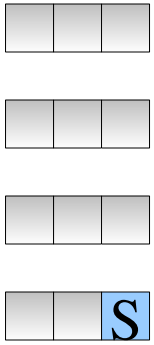
Normalizing '??' !! - with Inline Bytecode

```
String property = `[
    push `readProperty(“x”);
    dup;
    store temp;
    ifnull else;
        push `e2;
        goto end;
    else:
        load temp;
    end:
];
System.out.println(property);
```



Normalizing '??' !! - with Inline Bytecode

```
String property = `[
    `String temp = readProperty(“x”);
    `if (temp == null) temp = “default”;
    push `temp
];
System.out.println(property);
```



Normalizing '??' !! - with an *EBlock*

```
String property = {|
    String temp = readProperty(“x”);
    if (temp == null) temp = “default”;
|   temp
|};
System.out.println(property);
```

desugar-coalescing:

```
|[ e1 ?? e2 ]| →
|[{|
      String x_temp = e1;
      if (x_temp == null) x_temp = e2;
   |  x_temp
   |}
]|
with x_temp := <newname> “temp”
```

'??' in a Normalization Rule

```
desugar-coalescing:
  |[ e1 ?? e2 ]| →
  |[{|
        String x_temp = e1;
        if (x_temp == null) x_temp = e2;
    |   x_temp
    |}
  ]|
  with x_temp := <nev
```

Run-time:
Exception in thread "main"
java.lang.NullPointerException
…
at Generated.getVertices(Generated.java:10)

Position Information In Generated Code

desugar-coalescing:

```
|[ e1 ?? e2 ]| →
|[ trace (e1 ?? e2 @ <File>:<Line>:<Column>) {{|
      String x_temp = e1;
      if (x_temp == null) x_temp = e2;
  |   x_temp
  |}}
]|
with x_temp := <newname> “temp”
```

Run-time:
Exception in thread "main"
java.lang.NullPointerException
…
at Shape.getVertices(Shape.java:10)

Information With Source Tracing

Reflection

	Class-method	Statement	Expression level
Traits	X		
Partial/open classes	X		
Iterator generators		X	
Finite State Automata		X	
Expression-based extensions			X
Aspect weaving	X	X	X

Concluding Remarks

- Extension effort must be proportional to the benefits
- Black Boxes vs. White Boxes
- Don't throw away your primitives
- Normalization rules

Mixing Source and Bytecode. A Case for Compilation by Normalization. Lennart C. L. Kats, Martin Bravenboer, and Eelco Visser. In OOPSLA'08.

http://www.program-transformation.org/Stratego/TheDryadCompiler
http://www.strategoxt.org