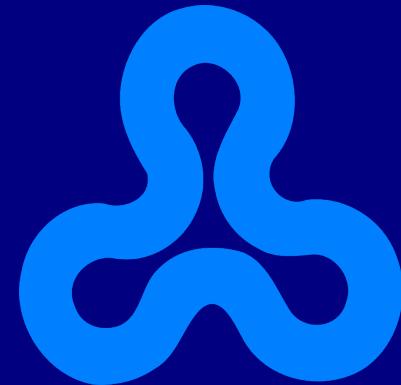


# Demanding First-Class Equality for Domain Specific Aspect Languages

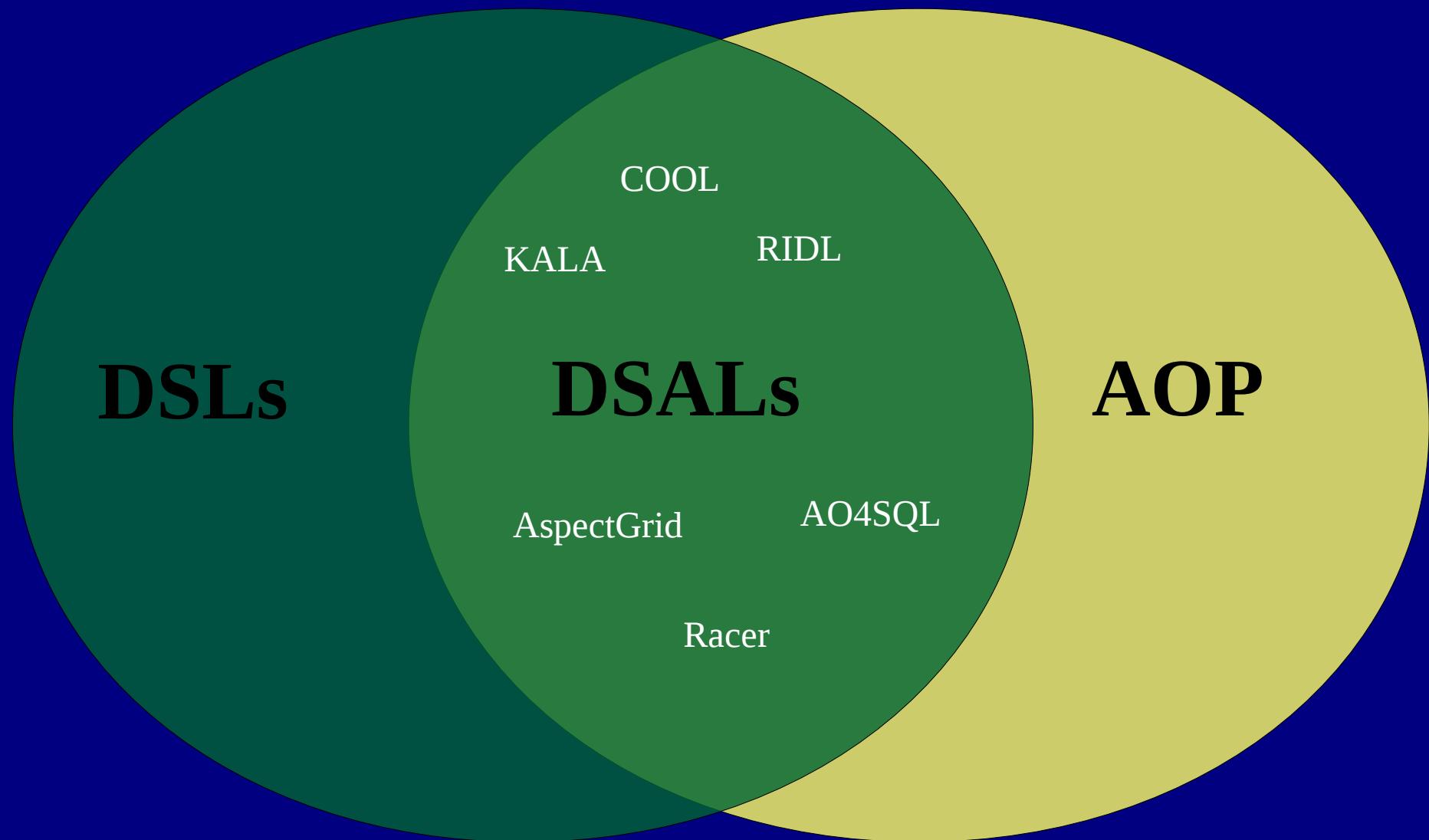
Arik Hadas

Dept. of Mathematics and Computer Science  
The Open University of Israel



Joint Work With:  
**David H. Lorenz**

# Domain Specific Aspect Languages



# **DSALs are Second-class**

- **Second-class DSLs**
  - Not as easy to develop and to use
- **Second-class AOP languages**
  - Incompatible with AOP tools

# Background

2005

DSL

**Language  
workbenches: The  
killer-app for  
domain specific  
languages**

DSAL

**abc: an  
extensible  
AspectJ  
compiler**

# Background

2005

DSL

**Language workbenches: The killer-app for domain specific languages**

DSAL

**abc: an extensible AspectJ compiler**

# Background

	2005	Since 2005
DSL	<p><b>Language workbenches:</b> The killer-app for domain specific languages</p>	<p><b>Language workbenches (LW)</b></p> <ul style="list-style-type: none"><li>– <b>xtext, MPS, Spoofax</b></li></ul>
DSAL	<p><b>abc:</b> an extensible AspectJ compiler</p>	<p><b>AOP composition frameworks (CF)</b></p> <ul style="list-style-type: none"><li>– <b>Reflex, XAspects, AWESOME</b></li></ul>

# Background

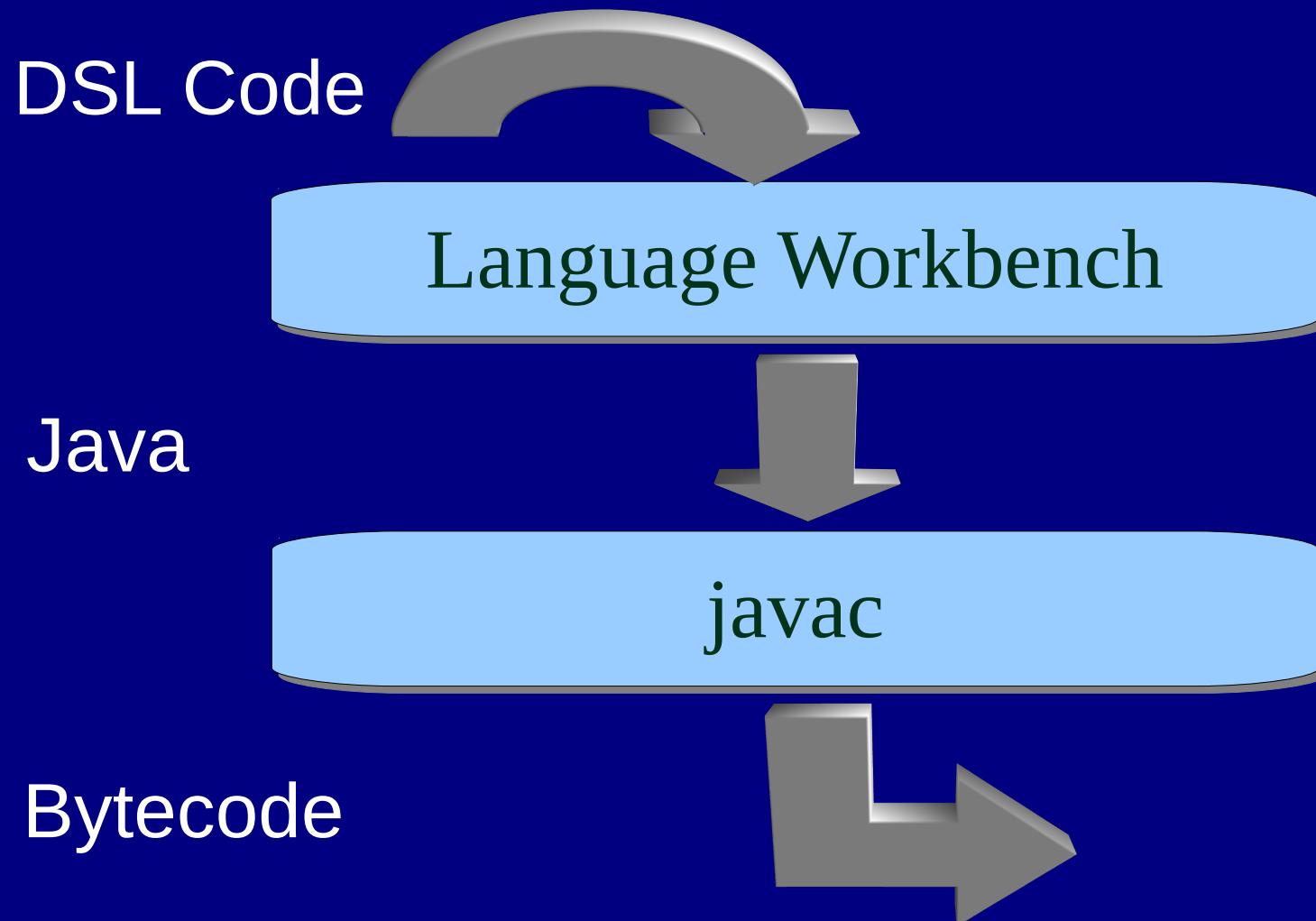
	2005	Since 2005
DSL	<p><b>Language workbenches:</b> The killer-app for domain specific languages</p>	<p><b>Language workbenches (LW)</b></p> <ul style="list-style-type: none"><li>– <b>xtext, MPS, Spoofax</b></li></ul>
DSAL	<p><b>abc:</b> an extensible AspectJ compiler</p>	<p><b>AOP composition frameworks (CF)</b></p> <ul style="list-style-type: none"><li>– <b>Reflex, XAspects, AWESOME</b></li></ul>

# Background

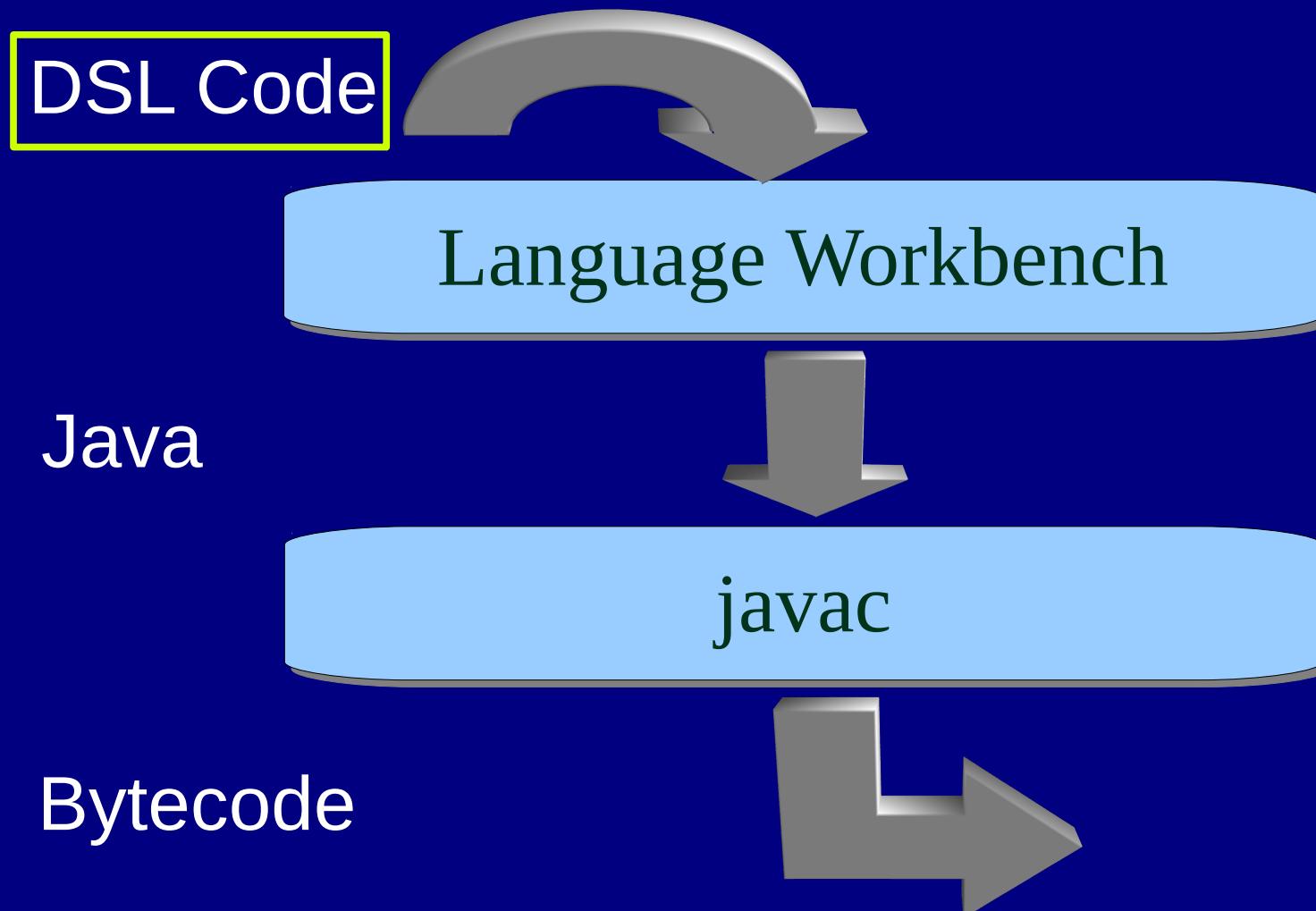
	2005	Since 2005
DSL	<b>Language workbenches:</b> The killer-app for domain specific languages	<b>Language workbenches (LW)</b> <ul style="list-style-type: none"><li>–.xtext, MPS, Spoofax</li></ul>
DSAL	<b>abc:</b> an extensible AspectJ compiler	<b>AOP composition frameworks (CF)</b> <ul style="list-style-type: none"><li>– Reflex, XAspects, AWESOME</li></ul>

**But no workbench solution for DSALs**

# Language Workbench (LW)



# Language Workbench (LW)

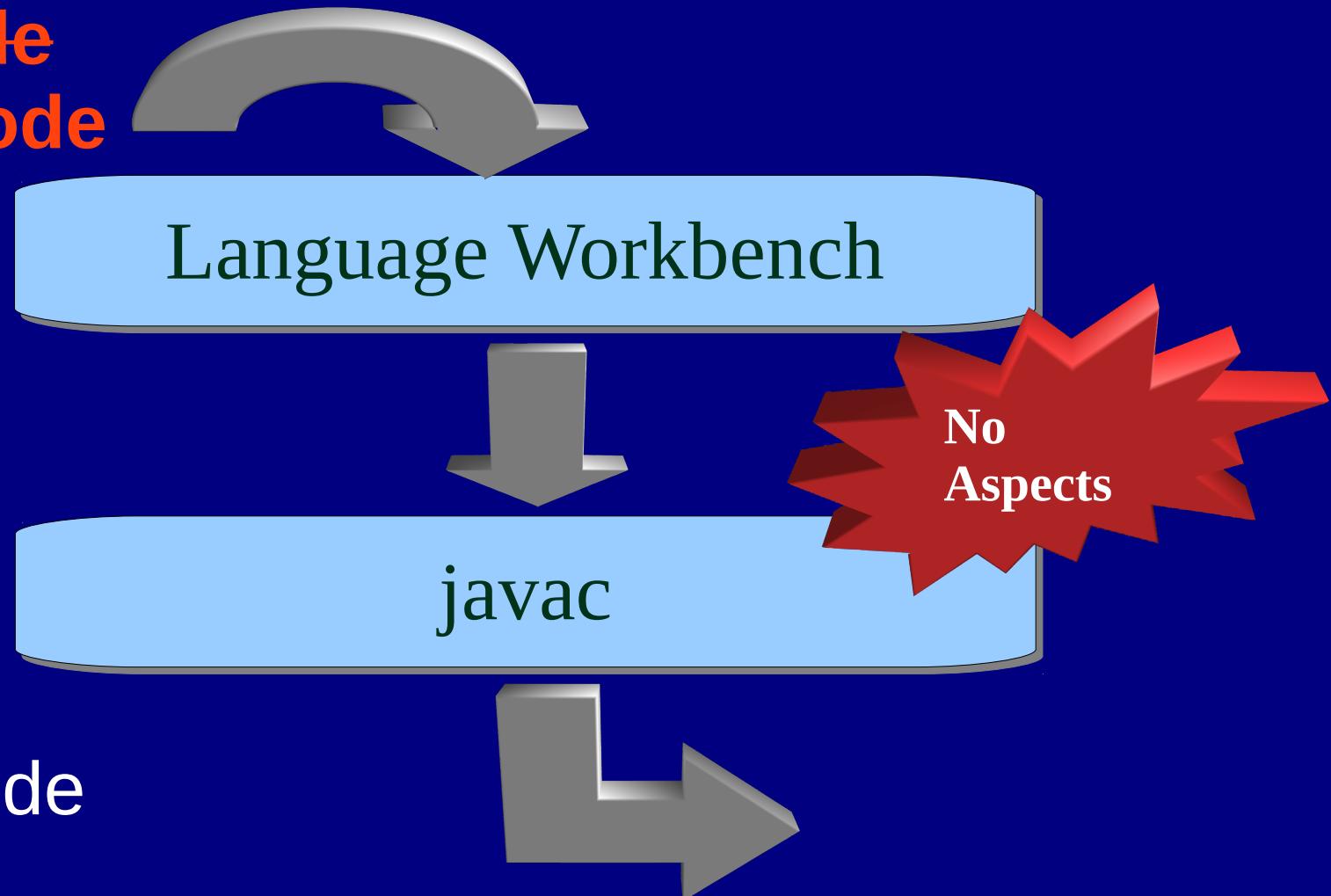


# Using LW for DSAL

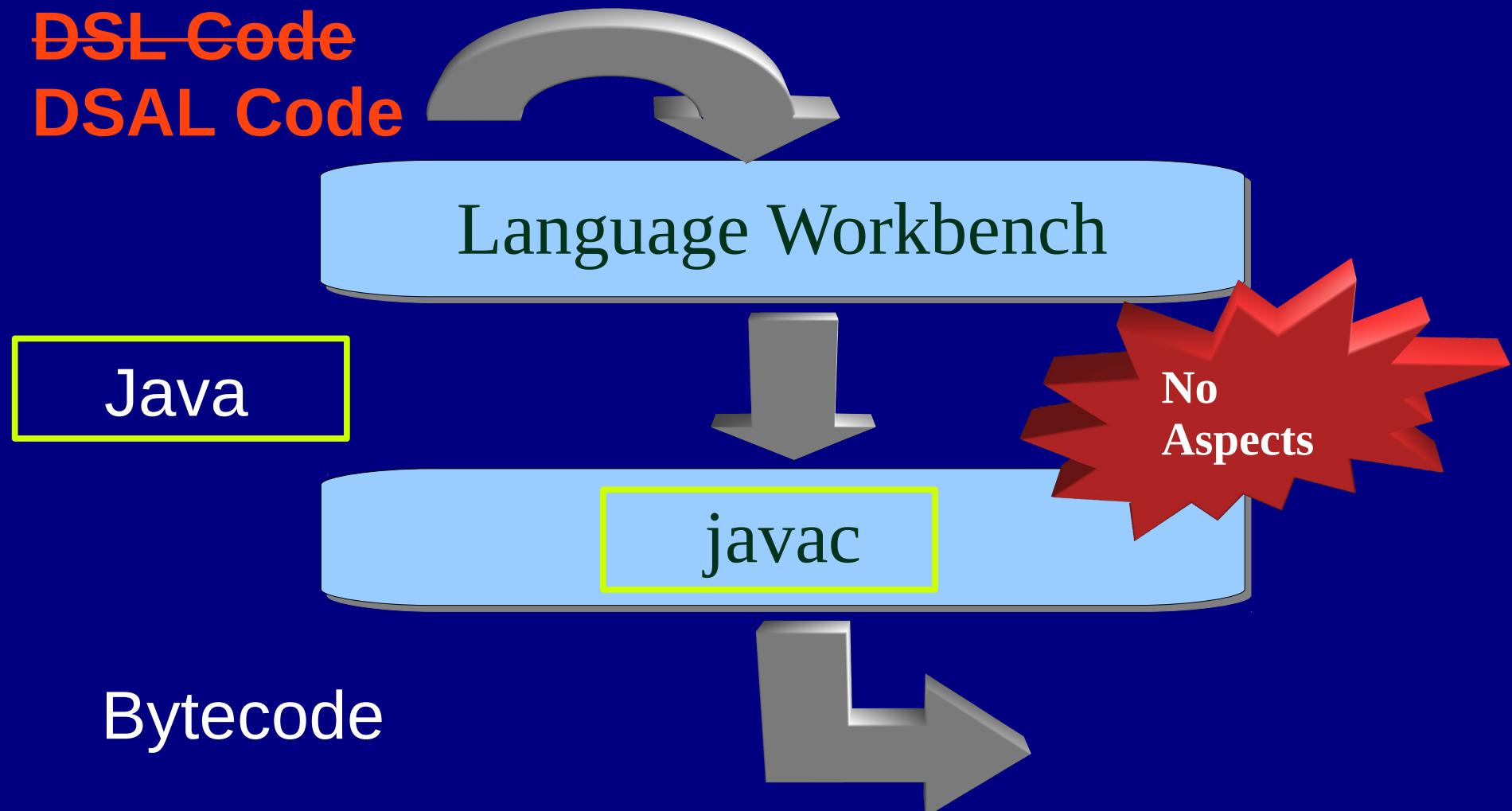
~~DSL Code~~  
DSAL Code

Java

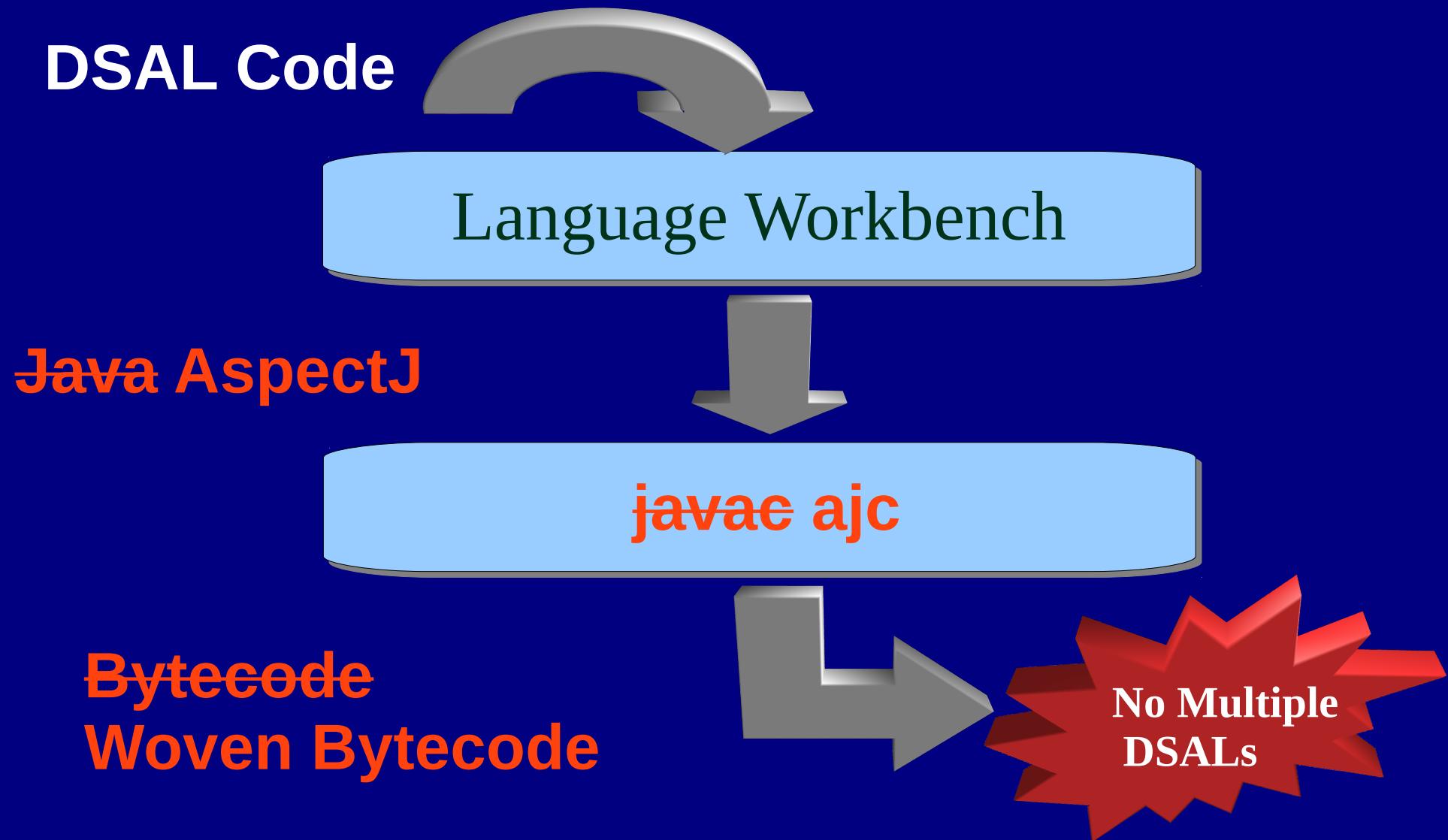
Bytecode



# Using LW for DSAL

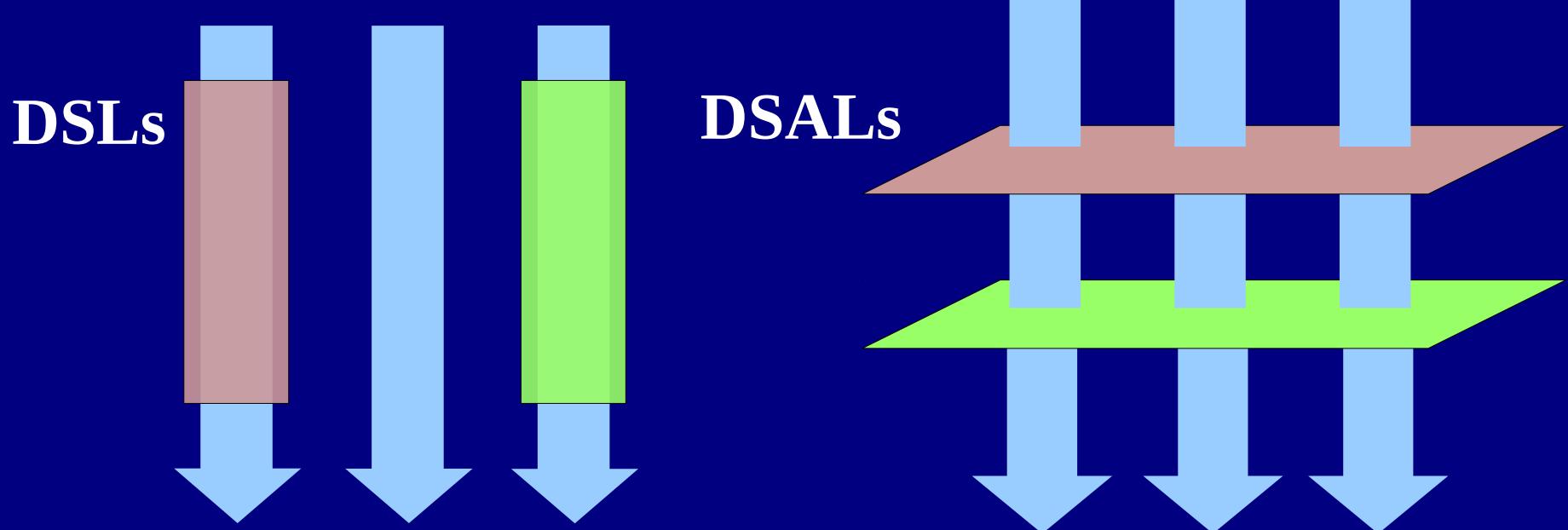


# LW for AspectJ



# DSAL Complexity

- Unlike DSL, DSAL tackles crosscutting concerns



# Looking for a DSAL Workbench

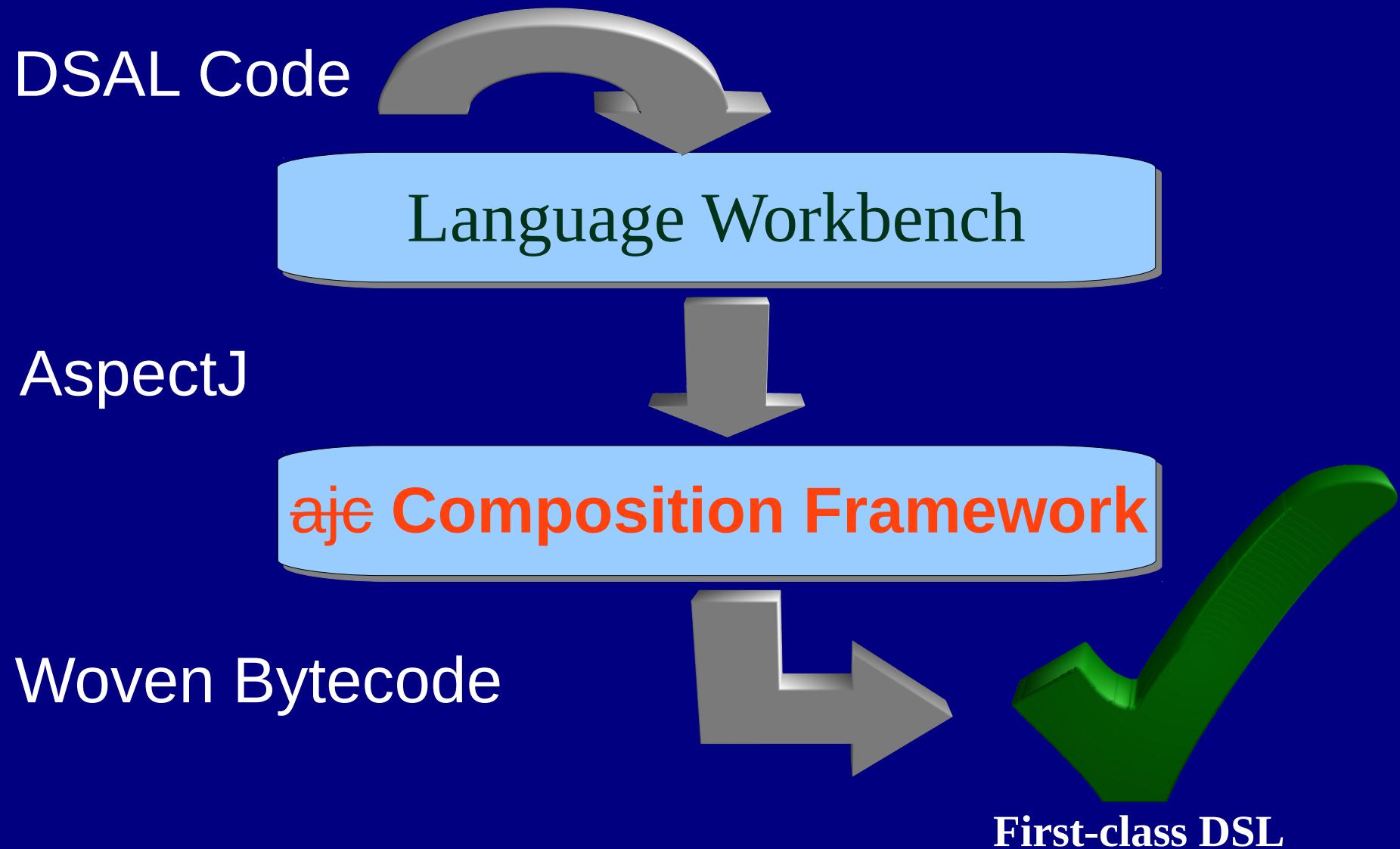
	Language Workbench	AOP Composition Framework
Tools for creation & usage of languages		
Defining weaving semantics needed for DSALs		

# Looking for a DSAL Workbench

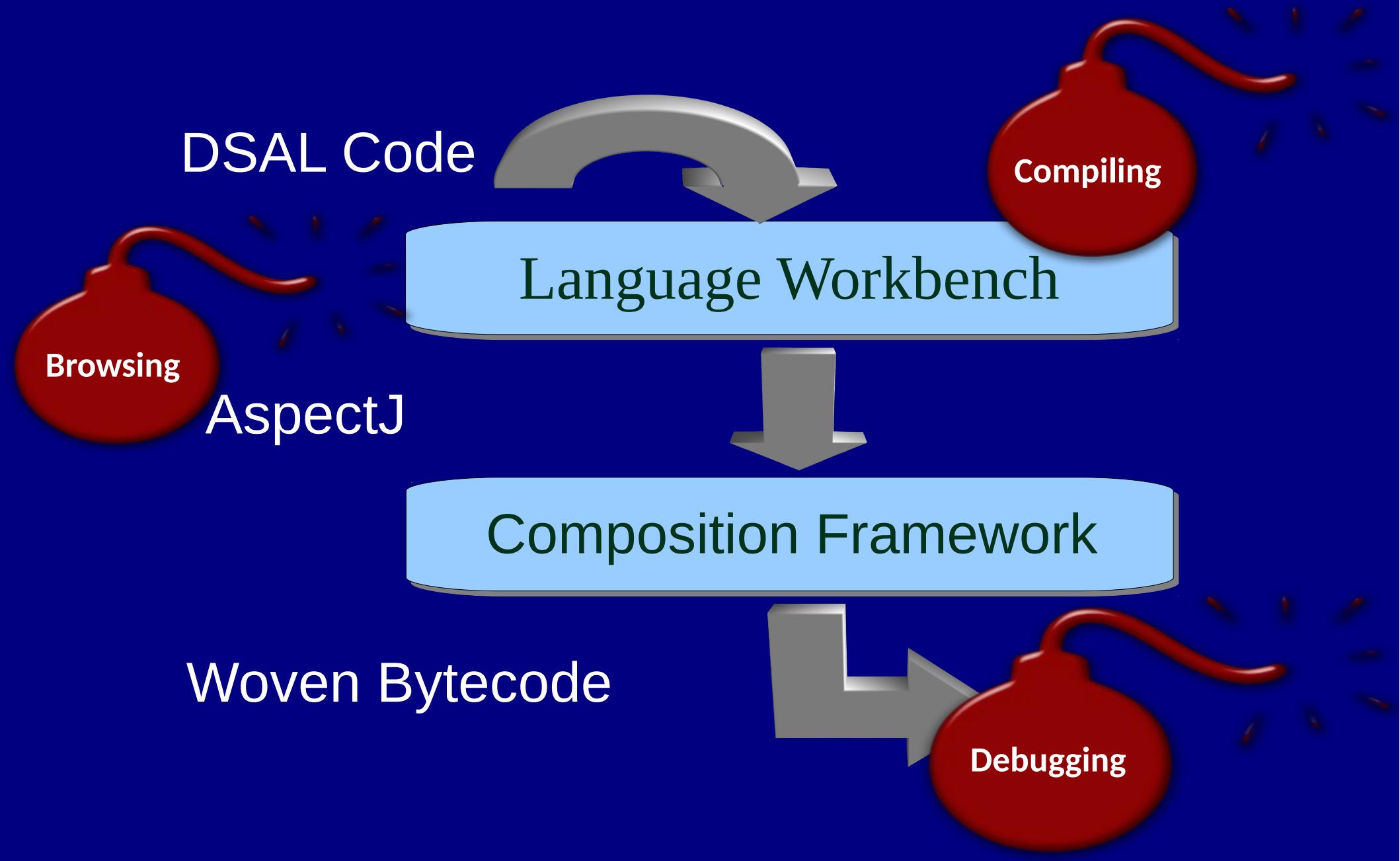
	Language Workbench	AOP Composition Framework
Tools for creation & usage of languages		
Defining weaving semantics needed for DSALs		

**Will a naive combination of the two be a proper solution?**

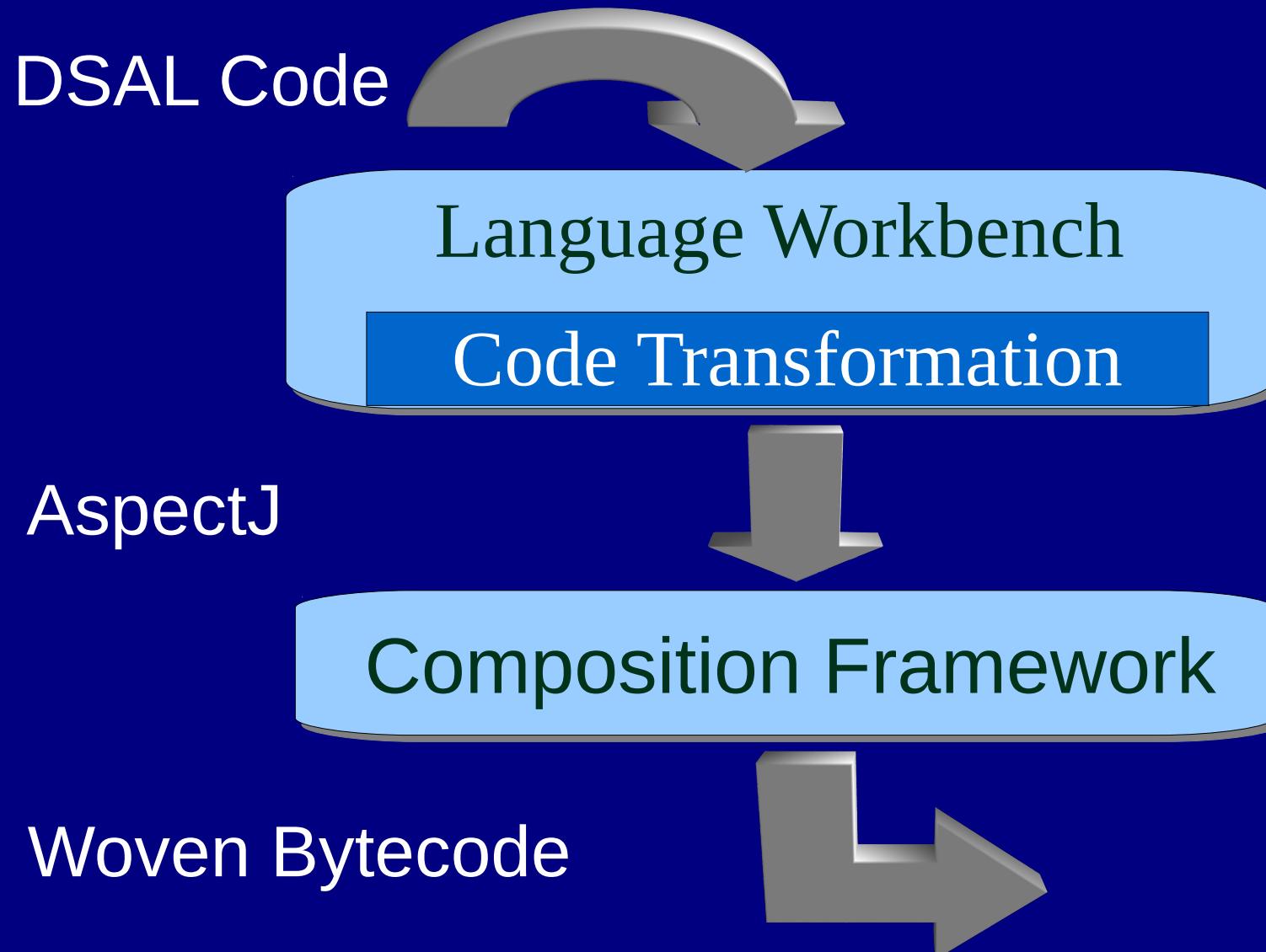
# Naive Combination of LW and CF



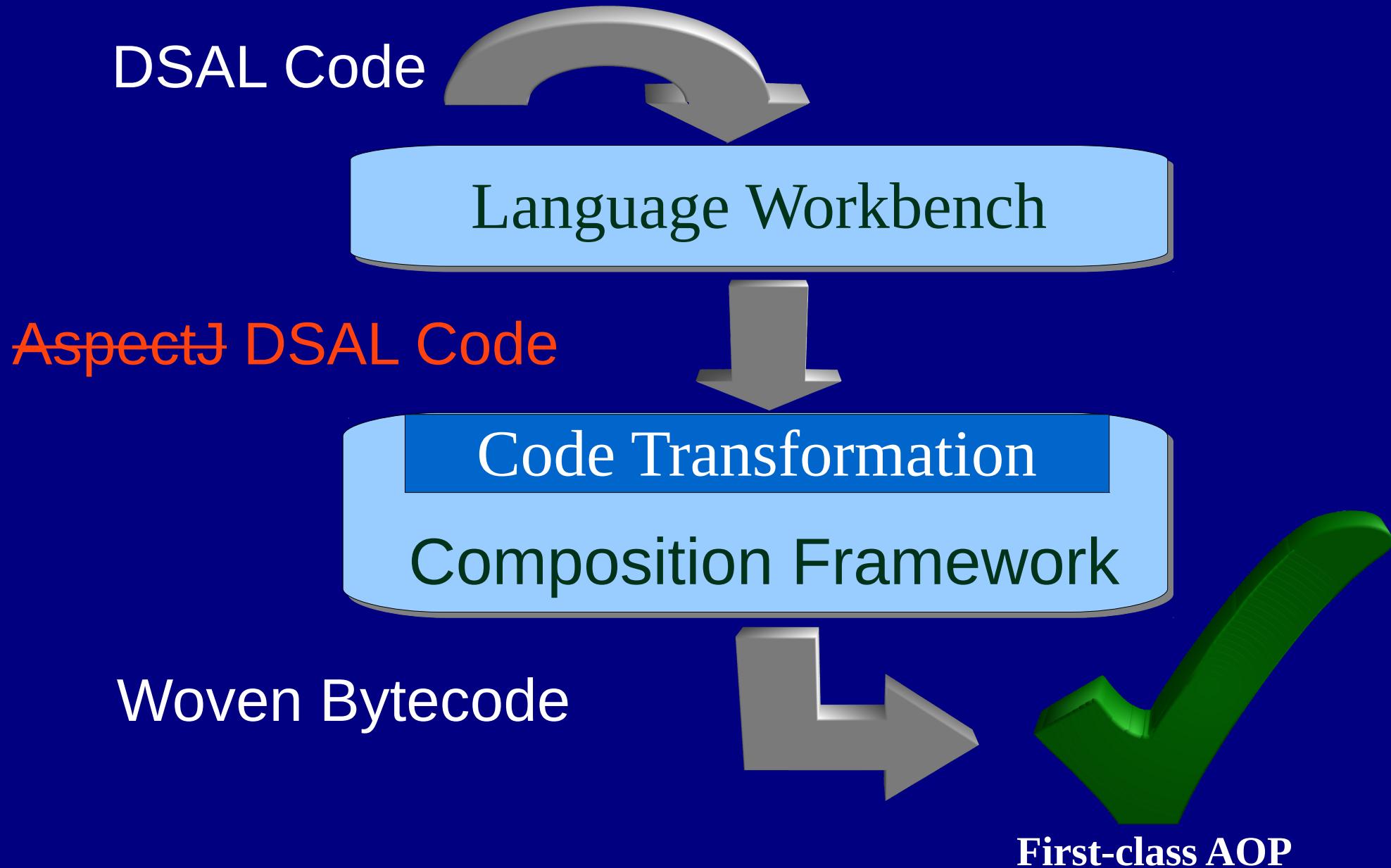
# But Still Second-class AOP



# Traditional LW Architecture



# DSAL Workbench Architecture



# DSAL Workbench Architecture

Standalone  
DSAL compiler

Can generate debugging &  
browsing information

DSAL Code

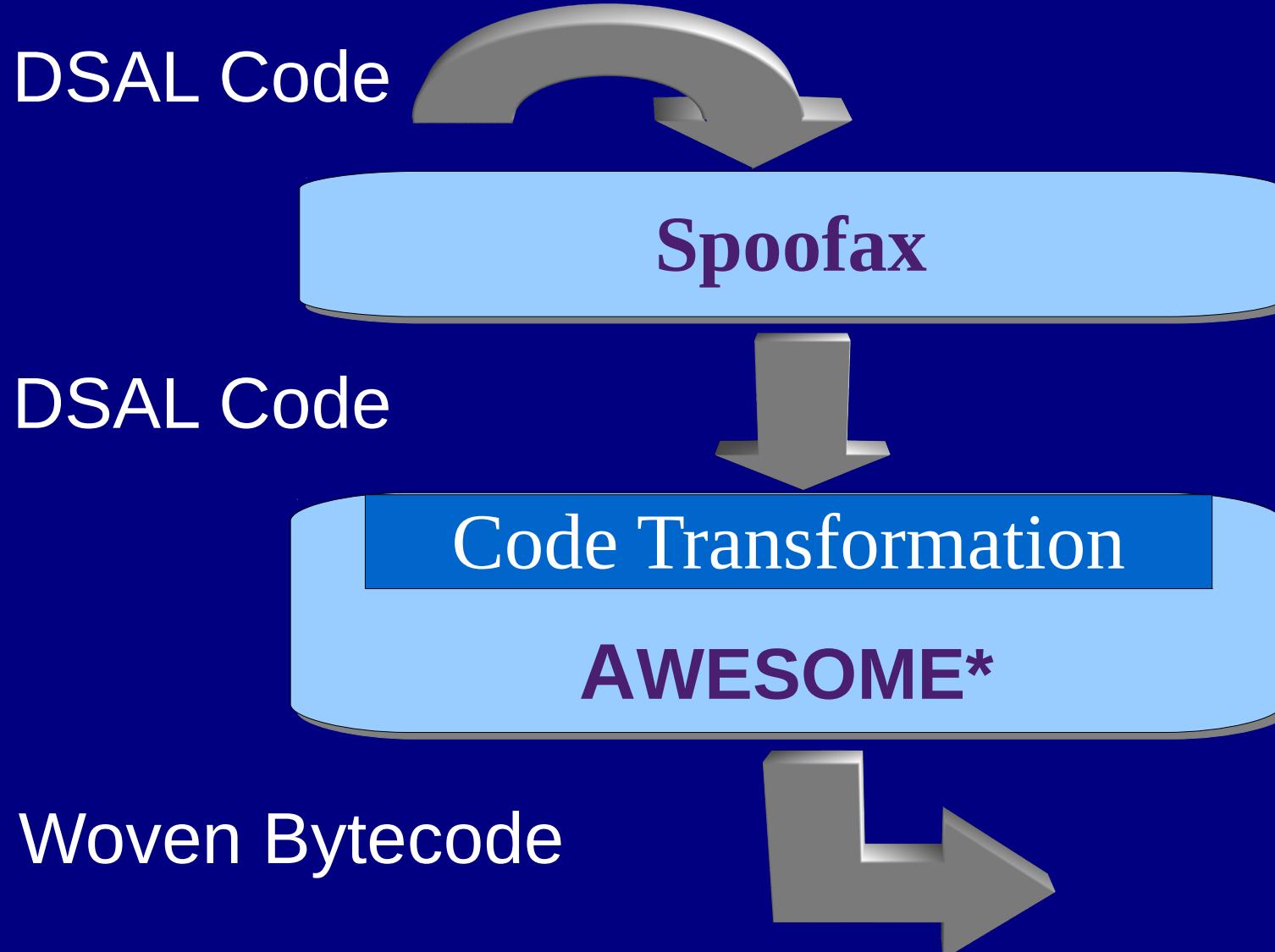
Code Transformation

Composition Framework

Woven Bytecode

First-class AOP

# Implementation



# Example: First Class COOL

The screenshot shows a Java development environment with two code editors. The left editor contains `BoundedStack.java` and the right editor contains `BoundedStackCoord.cool`.

**BoundedStack.java:**

```
package base;

public class BoundedStack implements Stack {
    protected Object[] buffer;
    private int usedSlots = 0;
    public BoundedStack(int capacity) {
        this.buffer = new Object[capacity];
    }
    public Object pop() {
        Object result = buffer[usedSlots - 1];
        usedSlots--;
        buffer[usedSlots] = null;
        return result;
    }
    public void push(Object obj) {
        // Implementation details
    }
}
```

A callout box highlights the `push` method in `BoundedStack.java`, stating "Multiple markers at this line" and listing:

- implements `base.Stack.push`
- advised by injar aspect: `BoundedStackCoord.cool`

**BoundedStackCoord.cool:**

```
package base;

coordinator base.BoundedStack {
    selfex {push(java.lang.Object), pop()};
    mutex {push(java.lang.Object), pop()};

    condition full = false, empty = true;
    int top = 0;

    push(java.lang.Object):
        requires (!full);
        on_entry {top = top + 1;};
        on_exit {
            empty = false;
            if (top == buffer.length) full = true;
        }
    pop():
        requires (!empty);
        on_entry {top = top - 1;};
        on_exit {
            full = false;
            if (top == 0) empty = true;
        }
}
```

# DSAL “Bill of Rights”

- *Freedom of Expression*
  - *Syntactic*
  - *Semantic*
- *Economic Freedom*
  - *Cost effective Implementation*
  - *Cost effective Usage*
- *Freedom of Assembly*
  - *DSL Interoperability*
- *Equality with domain-specific languages and AOP languages*

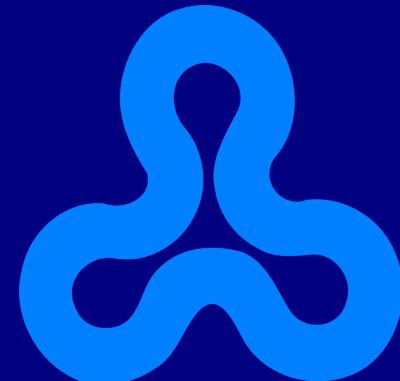
# Related Work

- **Language Workbenches**
  - [Fowler, 2005] Language workbenches: The killer-app for domain specific languages.
  - [Kats and Visser, 2005] The Spoofax language workbench: Rules for declarative specification of languages and IDEs.
  - [Lorenz and Rosenan, 2011] Cedalion: A language for language oriented programming.
- **AOP Composition Frameworks**
  - [Kojarski and Lorenz, 2005] Pluggable AOP: Designing aspect mechanisms for third-party composition.
  - [Lorenz and Kojarski, 2007] Understanding aspect interaction, co-advising and foreign advising.
  - [Kojarski and Lorenz, 2007] Awesome: An aspect co-weaving system for composing multiple aspect-oriented extensions.

# Conclusion

- A novel design for DSAL workbench that produces first-class DSAL
  - First-class DSL
  - First-class AOP language
- Prototype comprising Spooftax and AWESOME\*
- Plug-in for COOL as a first-class DSAL

# Thank You!



**Arik Hadas and David H. Lorenz**  
Dept. of Mathematics and Computer Science  
The Open University of Israel

[arik.hadas@openu.ac.il](mailto:arik.hadas@openu.ac.il)

<https://github.com/OpenUniversity>