

Static meta-programming (AST transformations)



Václav Pech

NPRG014 2025/2026

<http://www.vaclavpech.eu>

@vaclav_pech

Last time agenda



Dynamic method call dispatch

Dynamic object creation

Builders

Traits

Categories

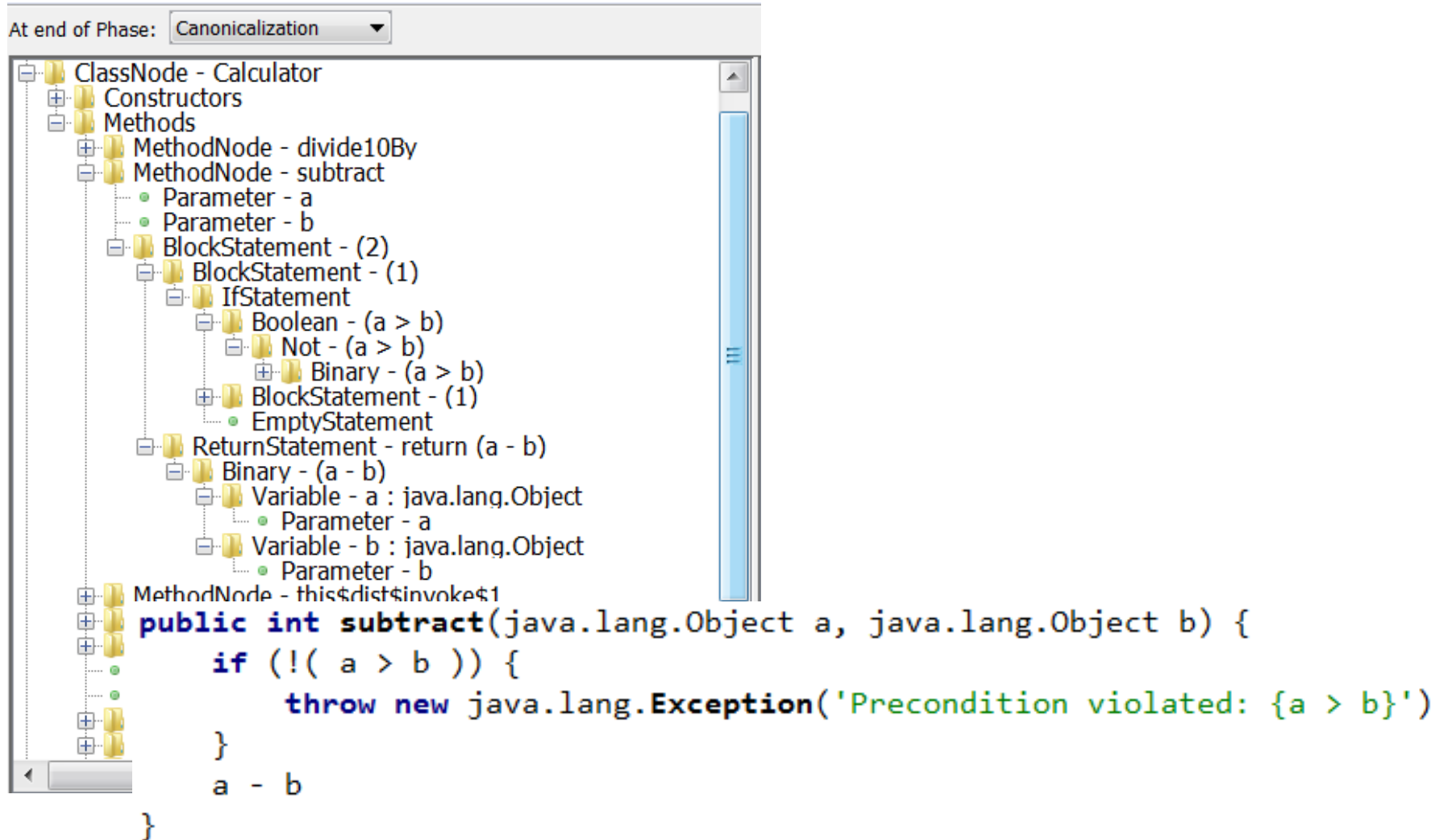
Internal DSLs

Part 5

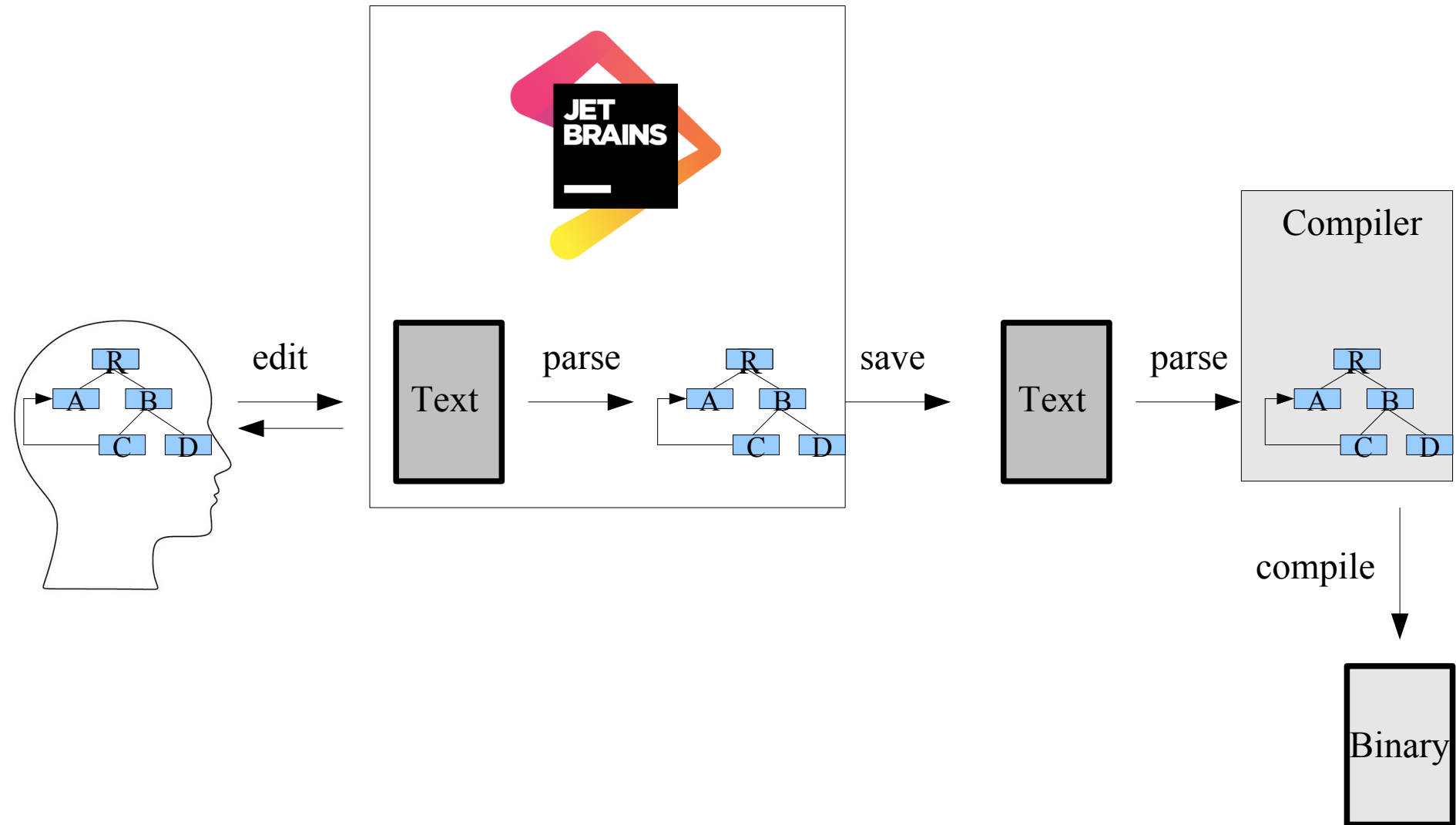
Static meta-programming

(program modifications at compile-time)

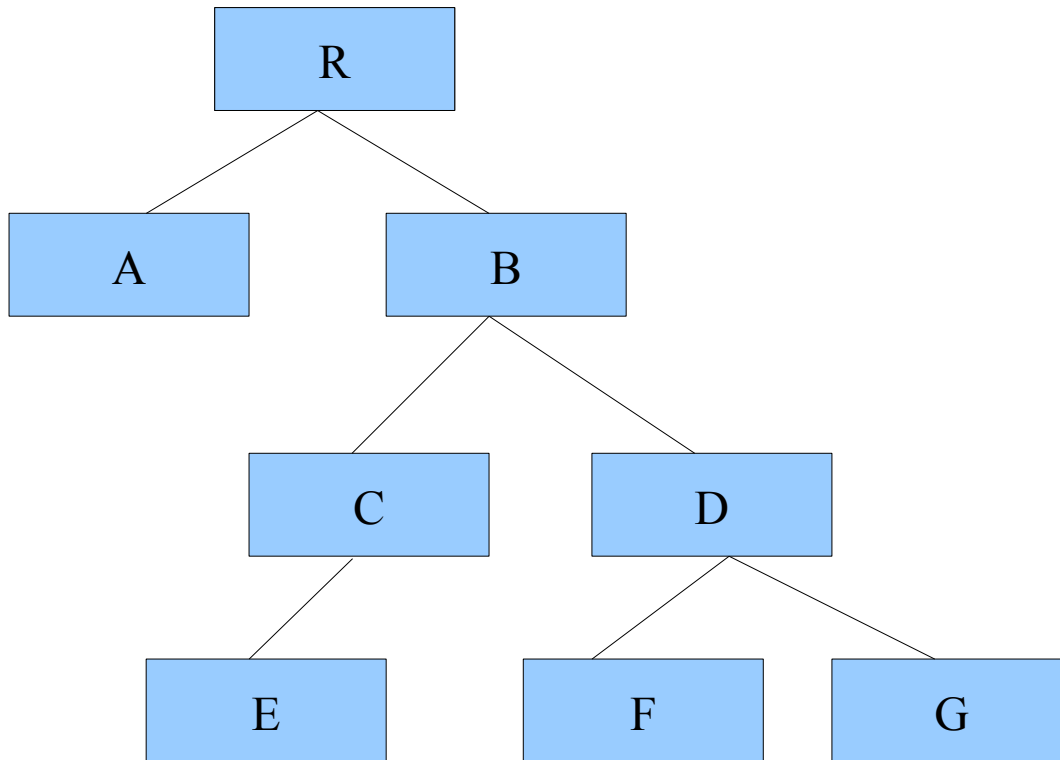
AST



Programming

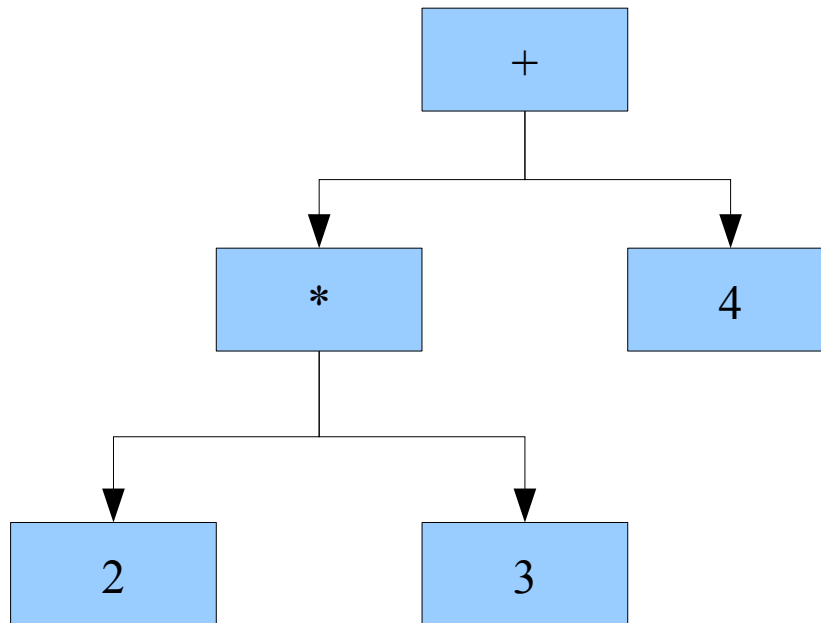


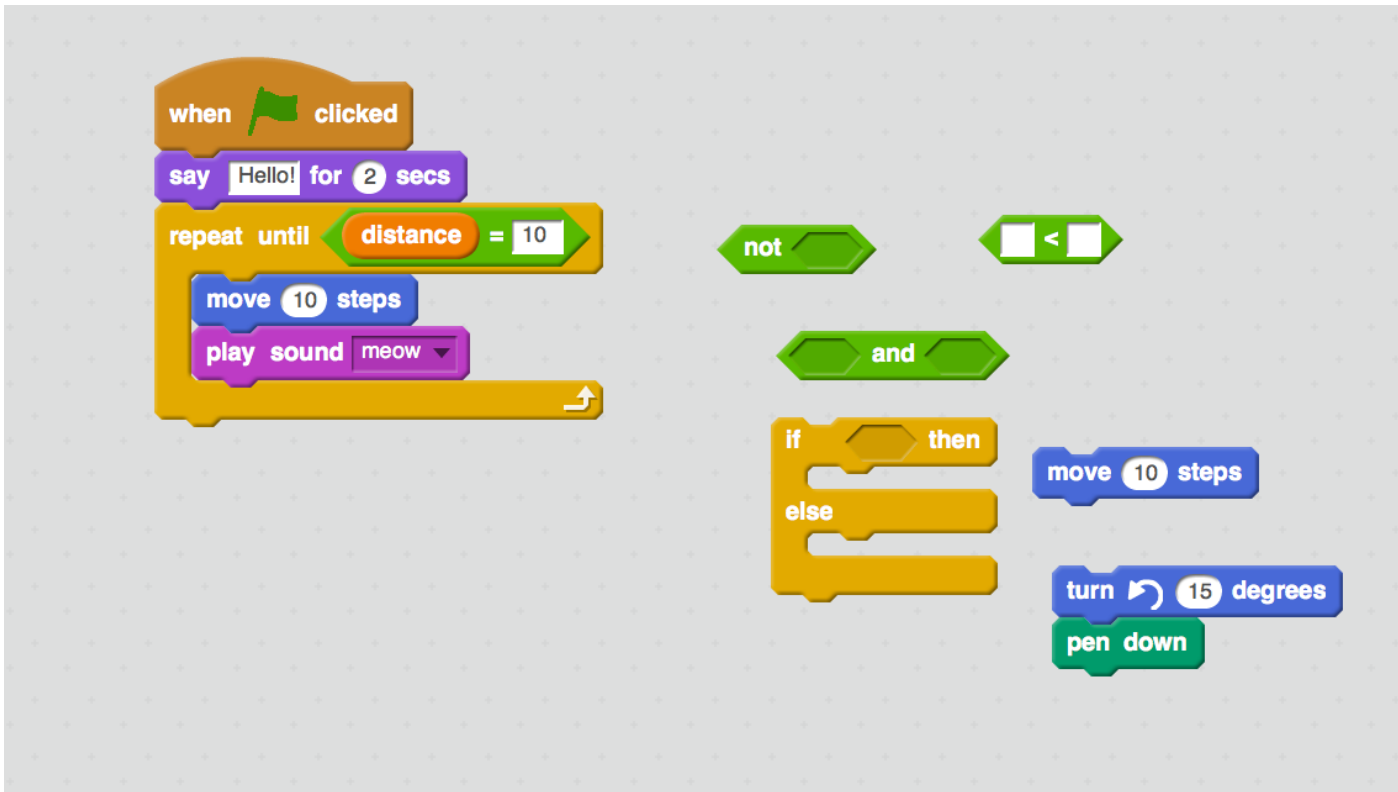
Programs are trees



Lisp (1958)

(+ (* 2 3) 4)






```
public class Demo {
```

```
    private static void foo() {  
        System.out.println("Foo called");  
    }
```

```
    public static void main(String[] args) {
```

```
        System.out.println("Application started");
```



```
        if (args.length > 0) {  
            System.out.println("Supplied arguments");  
            for (String value : args) {  
                System.out.println("Argument: " + value);  
            }  
        } else {  
            System.out.println("No arguments provided");  
        }
```

```
        foo();  
        System.out.println("Application completed");  
    }
```

```
}
```

▼ node

▼ (n) if

📁 Concept = `jetbrains.mps.baseLanguage.structure.IfStatement`

▼ n ifTrue : StatementList

📁 Concept = `jetbrains.mps.baseLanguage.structure.StatementList`

► **n** statement : ExpressionStatement

▼ **n statement : for**

📁 Concept = `jetbrains.mps.baseLanguage.structure.ForeachStatement`

▼ n **body** : StatementList

📁 Concept = `jetbrains.mps.baseLanguage.structure.StatementList`

▼ **n statement : ExpressionStatement**


📁 Concept = `jetbrains.mps.baseLanguage.structure.ExpressionStatement`


▼ **n** expression : DotExpression {void}


📁 Concept = `jetbrains.mps.baseLanguage.structure.DotExpression`


- operand : .<static field> {java.io.PrintStream}

```
▶ n operation : println(String):void {void}
```


►  properties


▶  **referents**

►  properties

▶  **referents**


► properties


▶  **referents**

- ▶  `variable : value {string}`

- ▶ **n** iterable : args {string[]}

► properties


▶  **referents**


►  properties

▶ **referents**

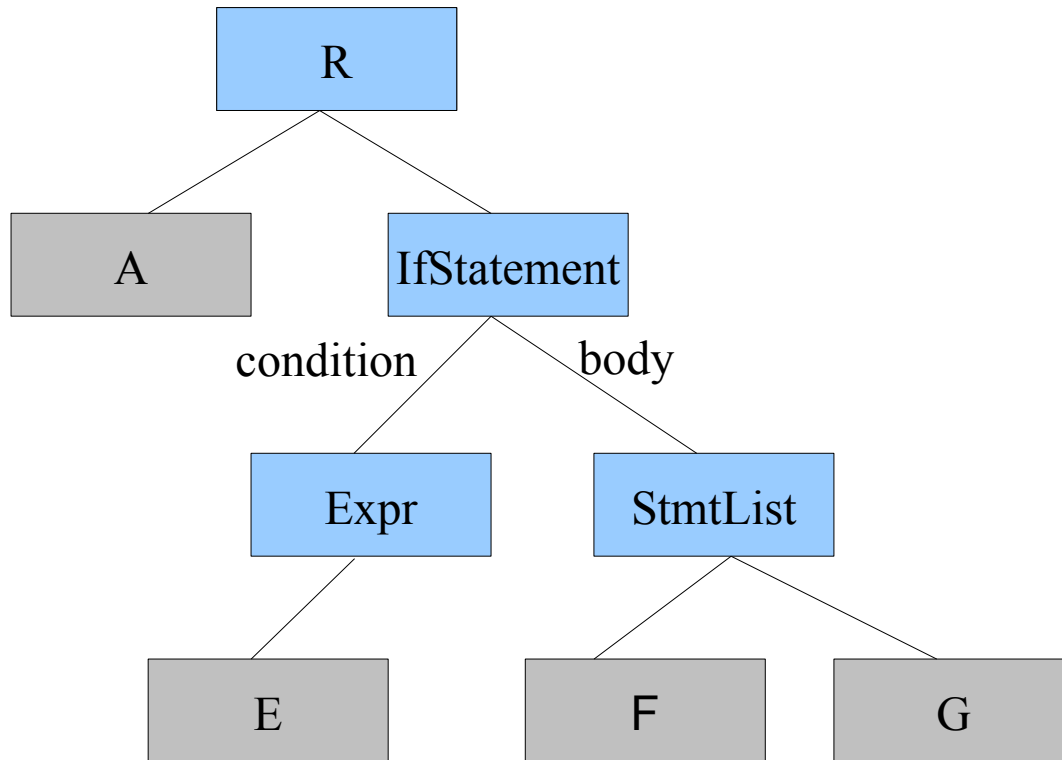
- ▶ **n** condition : > {boolean}

```
▶ n ifFalseStatement : {
```

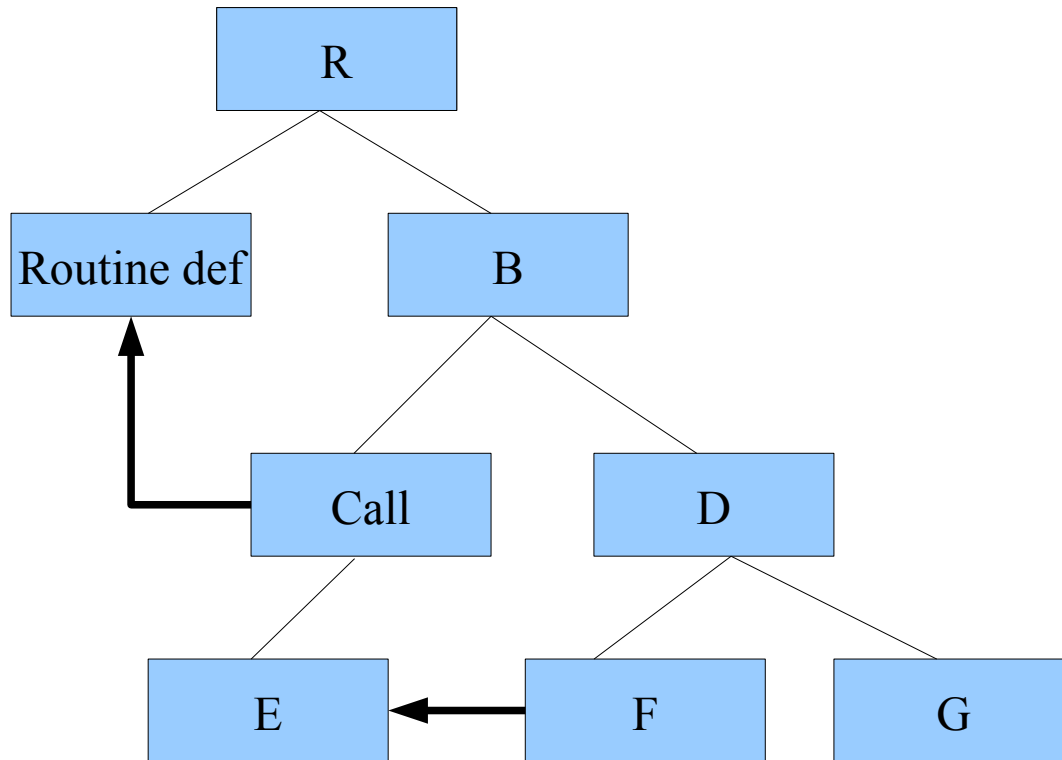
►  properties

▶  **referents**

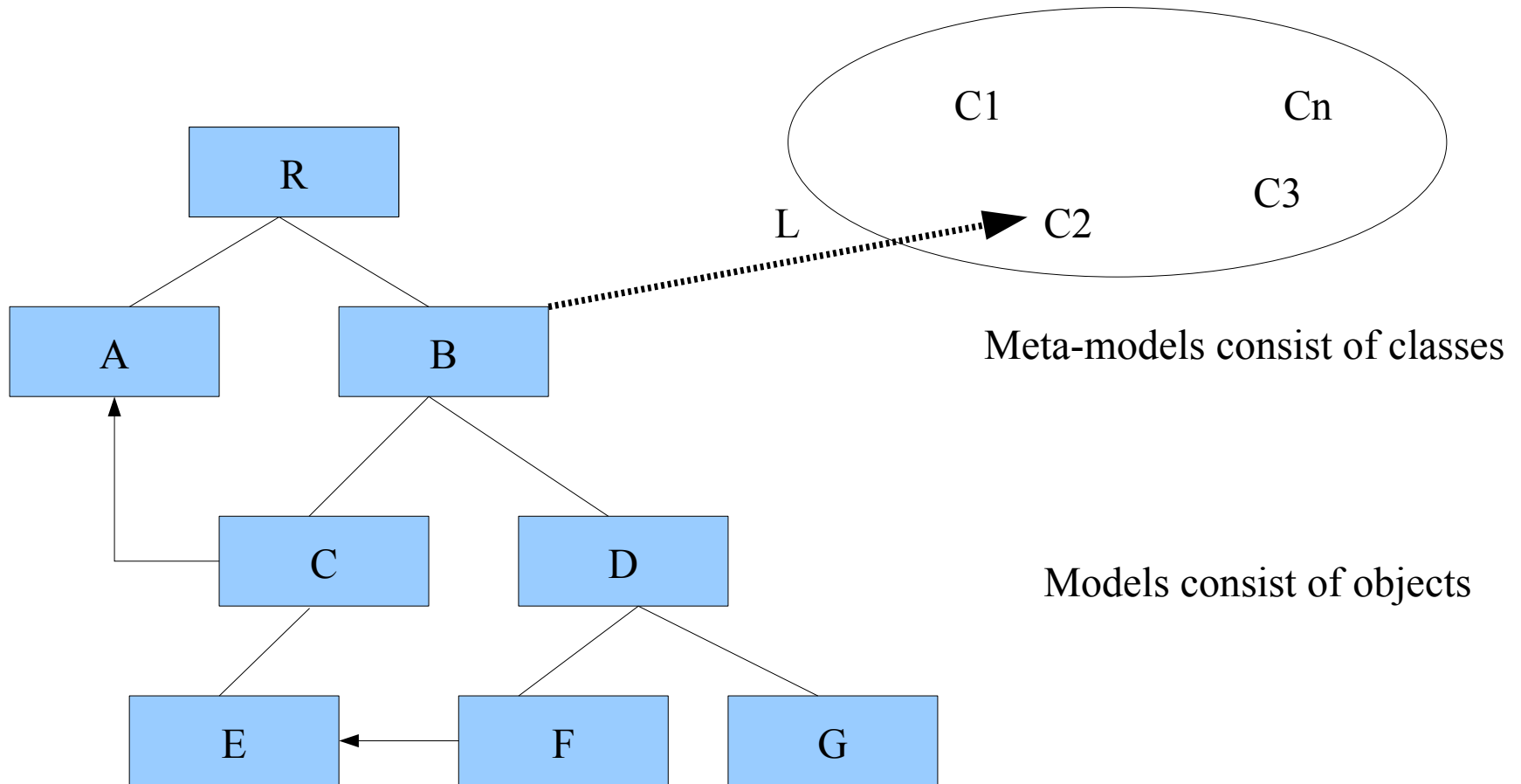
Children have roles



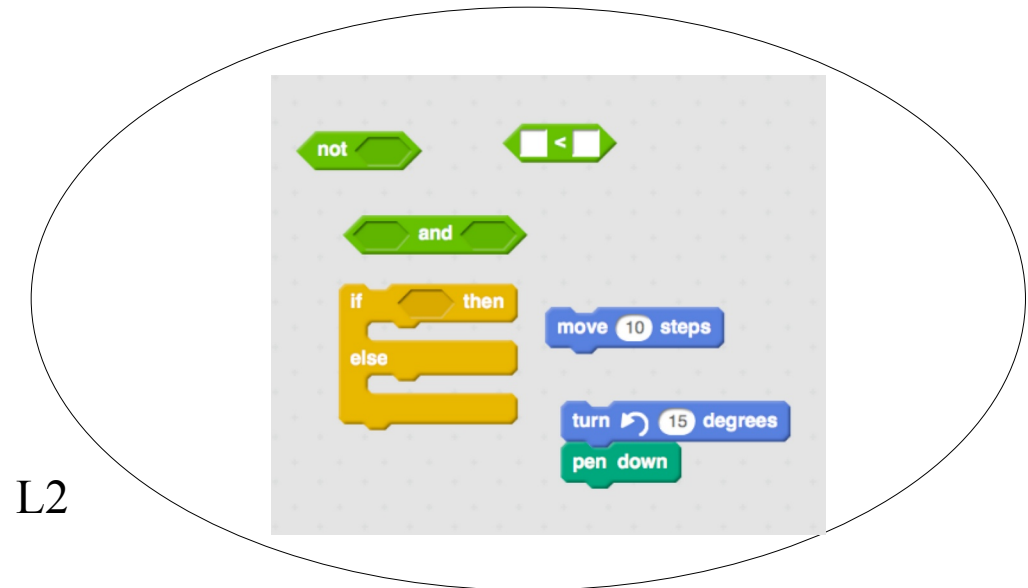
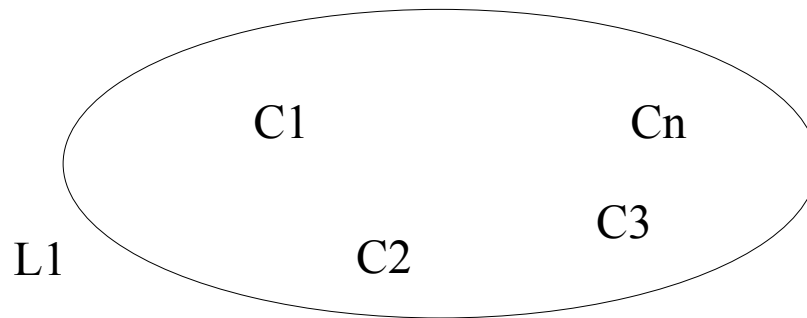
References cross the tree

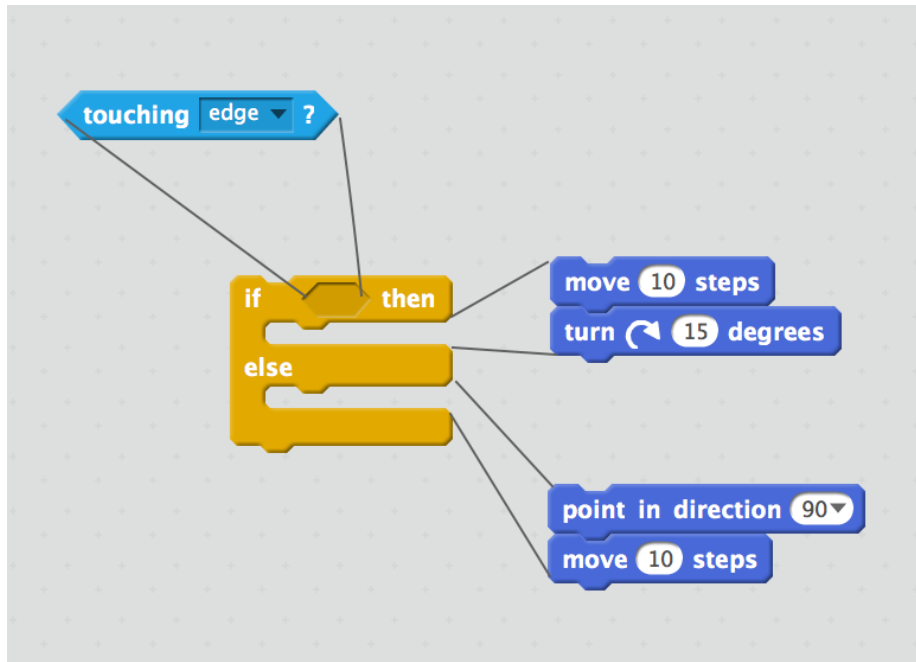


Programs and Languages



Languages are sets of classes





```
us.groovy.ast.stmt;
```

```

* Represents an if (condition) { ... } else { ... } statement in Groovy
*
* @author <a href="mailto:james@coredevelopers.net">James Strachan</a>
* @version $Revision$
*/

```

```
public class IfStatement extends Statement {
```

```

    private BooleanExpression booleanExpression;
    private Statement ifBlock;
    private Statement elseBlock;

```

```

    public IfStatement(BooleanExpression booleanExpression, Statement ifBlock,
        this.booleanExpression = booleanExpression;
        this.ifBlock = ifBlock;
        this.elseBlock = elseBlock;
    }

```

The Node



```
public class ASTNode {
```

```
    private int lineNumber = -1;  
    private int columnNumber = -1;  
    private int lastLineNumber = -1;  
    private int lastColumnNumber = -1;  
    private ListHashMap metaDataMap = null;
```

```
    public void visit(GroovyCodeVisitor visitor) {  
        throw new RuntimeException("No visit() method implemented for class: ")  
    }
```



```
public class TernaryExpression extends Expression {
```

```
    private BooleanExpression booleanExpression;  
    private Expression trueExpression;  
    private Expression falseExpression;
```

```
public class ElvisOperatorExpression extends TernaryExpression {
```

```
    public ElvisOperatorExpression(Expression base, Expression falseExpression) {  
        super(getBool(base), base, falseExpression);  
    }
```

```
public class ForStatement extends Statement implements LoopingStatement {  
    public static final Parameter FOR_LOOP_DUMMY = new Parameter(ClassHelp  
  
    private Parameter variable;  
    private Expression collectionExpression;  
    private Statement loopBlock;  
    private VariableScope scope;
```

```
public class MethodNode extends AnnotatedNode implements Opcodes {

    public static final String SCRIPT_BODY_METHOD_KEY = "org.codehaus.mojo.antlr.lexer.MethodNode";
    private final String name;
    private int modifiers;
    private boolean syntheticPublic;
    private ClassNode returnType;
    private Parameter[] parameters;
    private boolean hasDefaultValue = false;
    private Statement code;
    private boolean dynamicReturnType;
    private VariableScope variableScope;
    private final ClassNode[] exceptions;
    private final boolean staticConstructor;

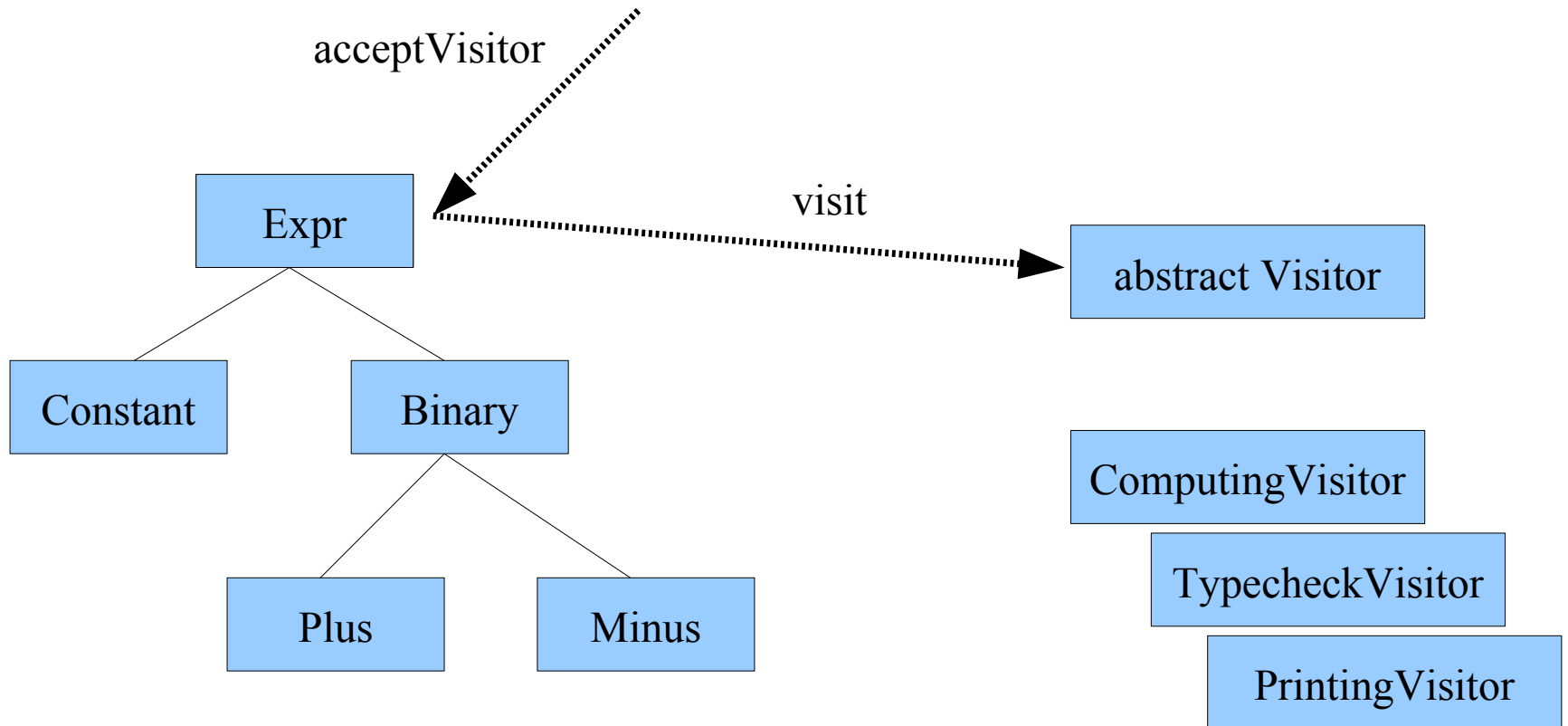
    // type spec for generics
    private GenericsType[] genericsTypes = null;
    private boolean hasDefault;

    // cached data
    String typeDescriptor;
```

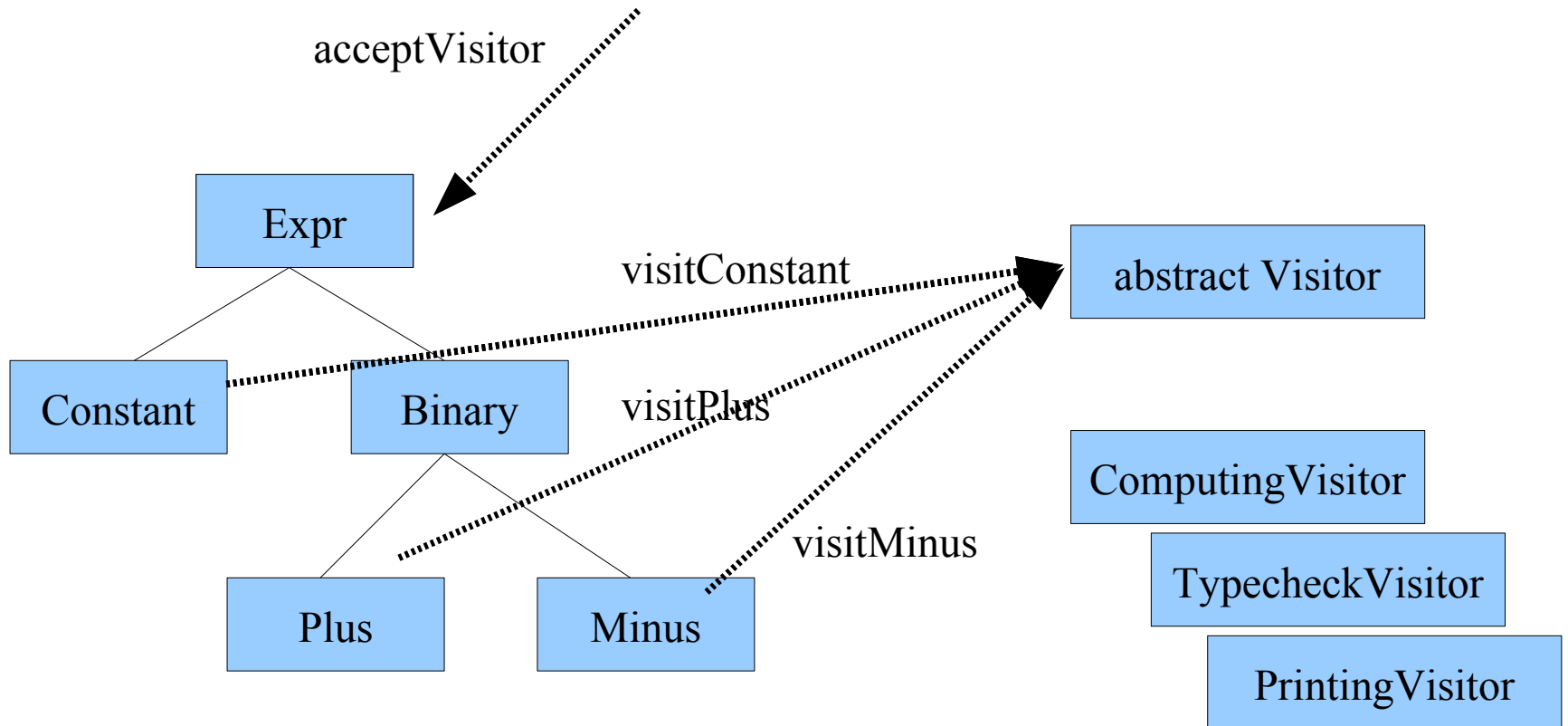
Visitor pattern

Separate class hierarchies from their processing

Visitor pattern



Visitor pattern



There's no life without trees

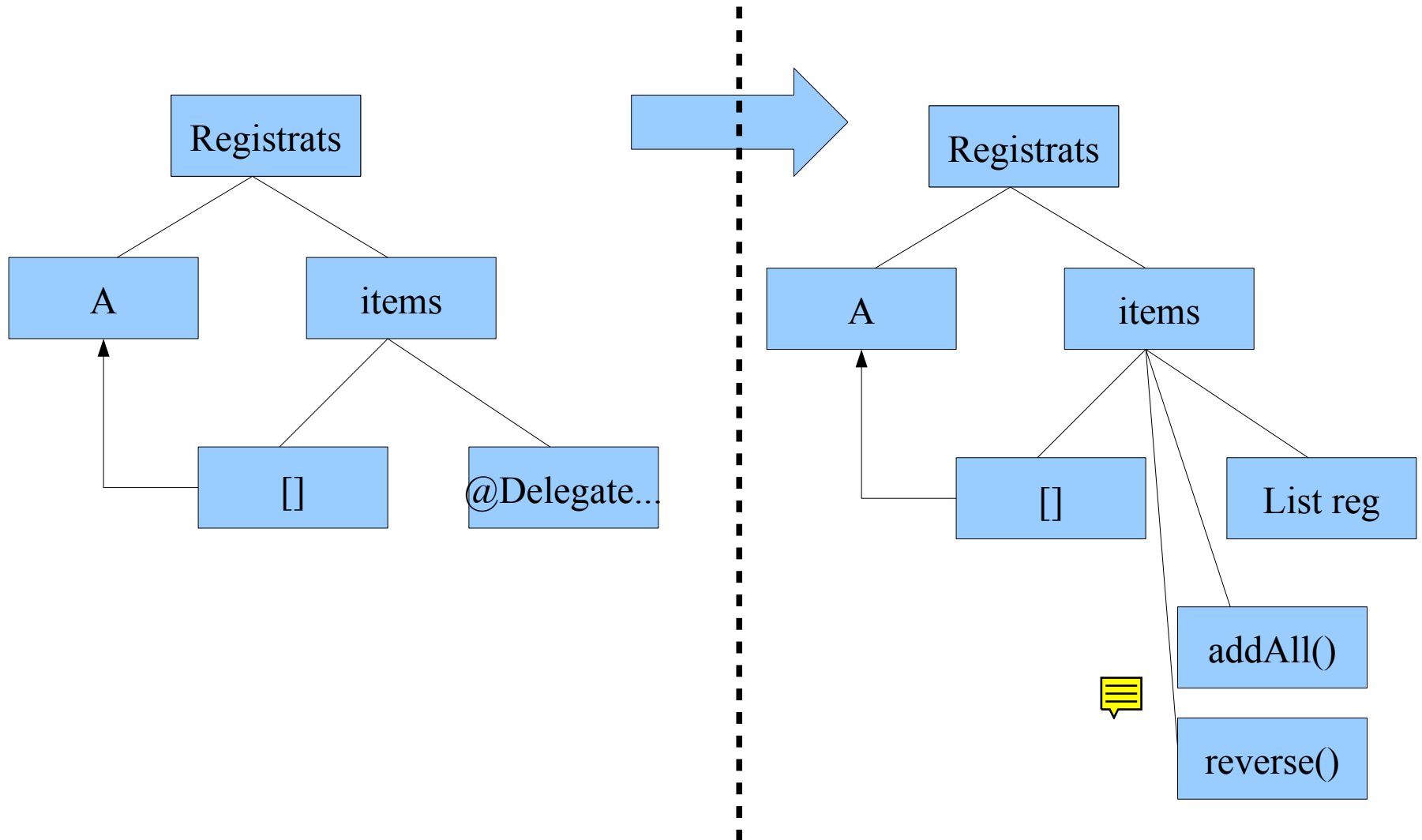
- Extending IDEs
- External DSLs
- Internal DSLs in modern languages
 - Groovy AST transformations
 - Scala macros
 - Lisp, Clojure macros
 - Java Lombok library

AST Transformations

```
class Registrations {  
    @Delegate List items = []  
}
```

```
def regs = new Registrations()  
regs.addAll(["Joe", "Dave"])  
assert ["Dave", "Joe"] == regs.reverse()
```


Ast transformation



Out-of-the-box transforms

@Delegate, @Immutable, @Singleton

@Lazy

@TupleConstructor

@InheritConstructors

@Canonical

@ToString

@EqualsAndHashCode

Out-of-the-box transforms

@Log, @Log4j, @Commons

@Synchronized

@WithReadLock

@WithWriteLock

@AutoClone, @AutoExternalize

... and many more

Type-checking/Static

@TypeChecked, @CompileStatic 

@TypeChecked

```
String test(Object val) {  
    if (val instanceof String) {  
        val.toUpperCase()  
    } else if (val instanceof Number) {  
        val.intValue() * 2  
    }  
}
```

Creating AST Transformations

```
@Retention(RetentionPolicy.SOURCE)
@Target([ElementType.METHOD])
@GroovyASTTransformationClass("RequiresTransformation")
public @interface Requires {
    String value() default "true";
}

@GroovyASTTransformation(phase = SEMANTIC_ANALYSIS)
public class RequiresTransformation implements ASTTransformation {

    public void visit(ASTNode[] astNodes, SourceUnit source) {
        ...
    }
}
```

Using AST classes directly

```
@GroovyASTTransformation(phase = CompilePhase.INSTRUCTION_SELECTION)
class AddMethodASTTransformation extends AbstractASTTransformation {
    @Override
    void visit(ASTNode[] nodes, SourceUnit source) {
        ClassNode classNode = (ClassNode) nodes[1]

        ReturnStatement code =
            new ReturnStatement(
                new ConstantExpression("42"))

        MethodNode methodNode =
            new MethodNode(
                "getMessage",
                ACC_PUBLIC,
                ClassHelper.make(String),
                [] as Parameter[],
                [] as ClassNode[],
                code)

        classNode.addMethod(methodNode)
    }
}
```

1

2

3

4

Use factory methods

// Constant expression

GeneralUtils.constX("Hello")

// Assignment

GeneralUtils.assignX(varX("x"), constX(42))

//Method call – *c.m(a)*

GeneralUtils.returnS(callX(c, "m", varX('a')))

Builders

`new AstBuilder()`

`.buildFromString()`

`.buildFromCode()`

`.buildFromSpec()`

Macros

AstSpecificationCompiler

```
.buildFromSpec {  
  method('convertToNumber', ACC_PUBLIC, Integer) {  
    parameters { parameter 'parameter': String.class }  
    exceptions {}  
    block {  
      returnStatement {  
        staticMethodCall(Integer, "parseInt") {  
          argumentList {  
            variable "parameter"  
          }  
        }  
      }  
    }  
  }  
}
```

AstSpecificationCompiler

The delegate for the *buildFromSpec()*
closures

```
.buildFromString ("
    Integer.parseInt(param)
")
```

```
.buildFromCode {  
    Integer.parseInt(param)  
}
```

```
macro {  
    java.lang.Integer.parseInt(param)  
}
```

Macro

```
return macro(CompilePhase.SEMANTIC_ANALYSIS, true) {  
    return java.security.MessageDigest  
        .getInstance('MD5')  
        .digest($v { fieldVar }.getBytes())  
        .encodeHex()  
        .toString()  
}
```

Macro

```
@Override
Expression transform(Expression exp) {
    Expression ref = macro { 1 + 1 }

    if (ASTMatcher.matches(ref, exp)) {
        return macro { 3 }
    }

    return super.transform(exp)
}
```

Summary



AST transformations – use and creation

References

<http://groovy-lang.org>

<http://grails.org>

<https://projectlombok.org/>