ExtendJ

Extensible Java Compiler

Jesper Öqvist
PhD Student
Lund University, Sweden

What is ExtendJ?

Extensible Java compiler.

Java 4-6	by **Torbjörn Ekman**
Java 7	by **Jesper Öqvist** (Master’s Thesis)
Java 8	by **Erik Hogeman** (Master’s Thesis)

Current maintainer: Jesper Öqvist (me)

Open Source: Modified BSD License

Goals

ExtendJ should be easy to extend with

- Analyses
- New language constructs
- Metrics

Used for

research,

PL experiments based on Java,

compiler course projects.

String Equality Check

Student Project

Bug pattern detection:

```
String food;
if (food == “beer”) { …
```

Ella Eriksson & Zimon Kuhs

String Equality Check

Student Project

Bug pattern detection:

```
String food;
if (food == “beer”) { …
```

`=> Suggestion: replace == with .equals()`

Ella Eriksson & Zimon Kuhs

String Equality Check

Implementation (JastAdd code):

```
EQExpr contributes "Suggestion: ..."
    when badStringEq()
    to Program.errors();



syn boolean EQExpr.badStringEq() =
    getLeft().type().isString() &&
    getRight().type().isString();
```

String Equality Check

Implementation (JastAdd code):

```
EQExpr contributes "Suggestion: ..."
    when badStringEq()
    to Program.errors();


syn boolean EQExpr.badStringEq() =
    getLeft().type().isString() &&
    getRight().type().isString();
```

```
food == “beer”
```

Ella Eriksson & Zimon Kuhs

Spread Operator

Student Project

Type analysis for the Spread Operator (*.)

```
List<Person>  people;
List<String>  names = people*.name;
```

(Working but not complete implementation)

Hans Bjerndell & Linus Lexfors

Multiplicities Extension

Developed in collaboration with prof. Friedrich Steimann:

```
@any Person people;
people += alice;        // += → .add()
people += bob;
people.work();          // Call .work() on alice and bob.
```

Demo

Demo: multiplicities.

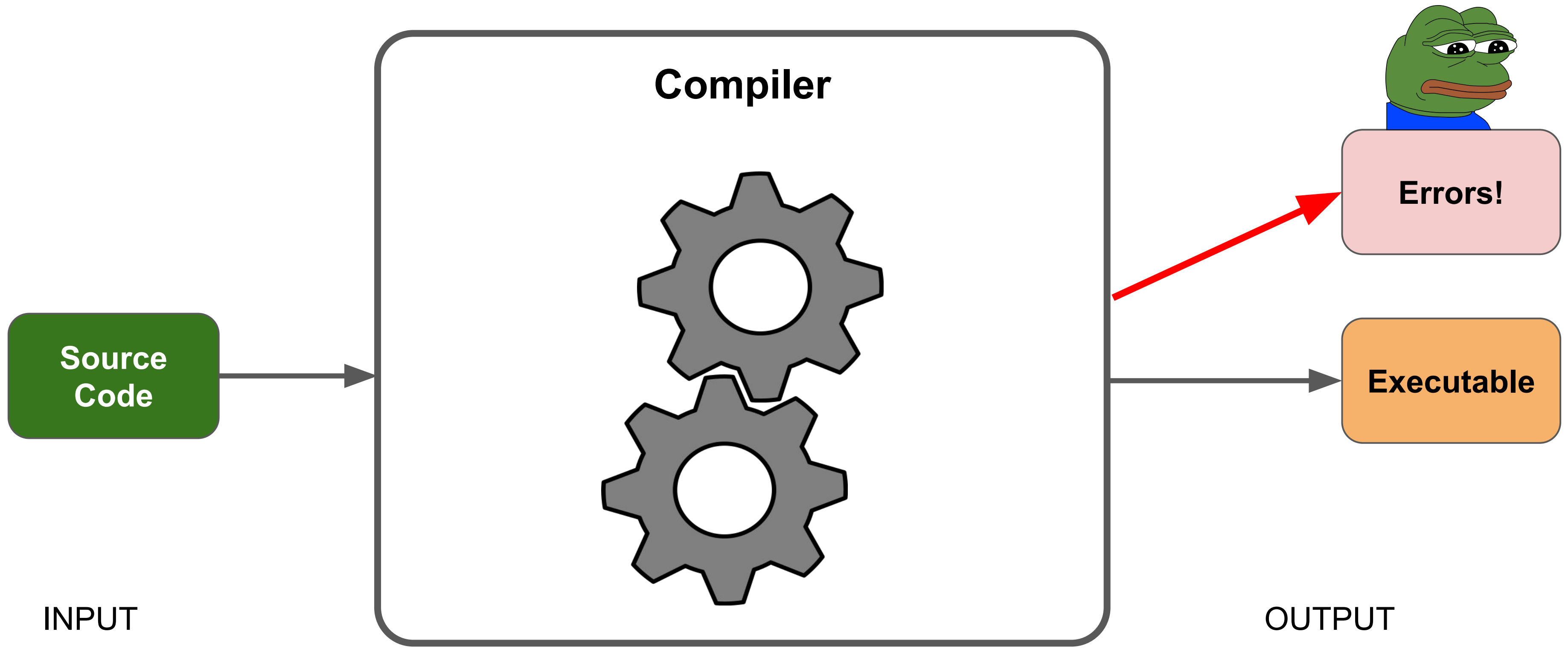
extendj.org

ExtendJ Internals

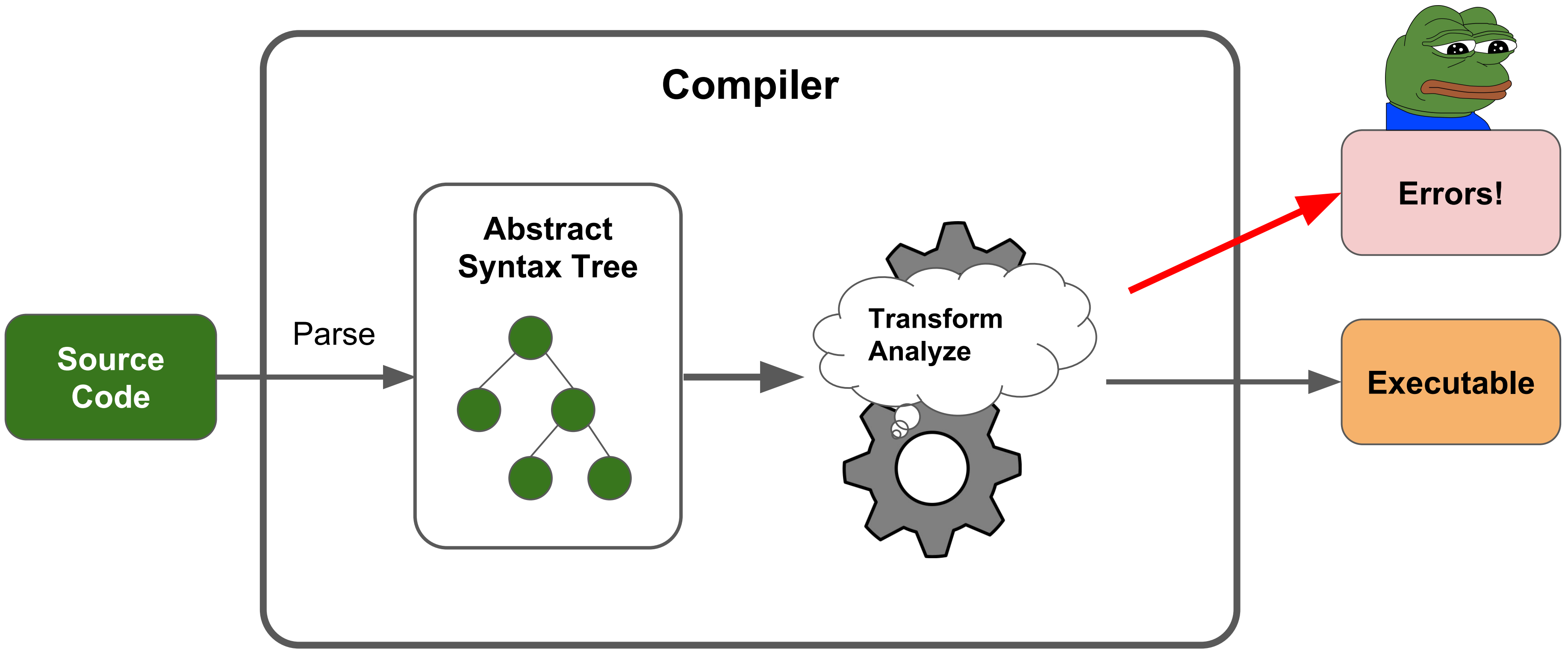
How does ExtendJ work?

- Let’s look at the internals...

Compiler Internals



Compiler Internals



Compiler Passes?

Conventional compilers are structured around **passes**.

One or multiple passes.

Compiler Passes?

Conventional compilers are structured around **passes**.

One or multiple passes.

One pass: all translation done while parsing.

Not all languages can be compiled in a single pass!

(C is a one-pass language. That’s why declaration order matters!)

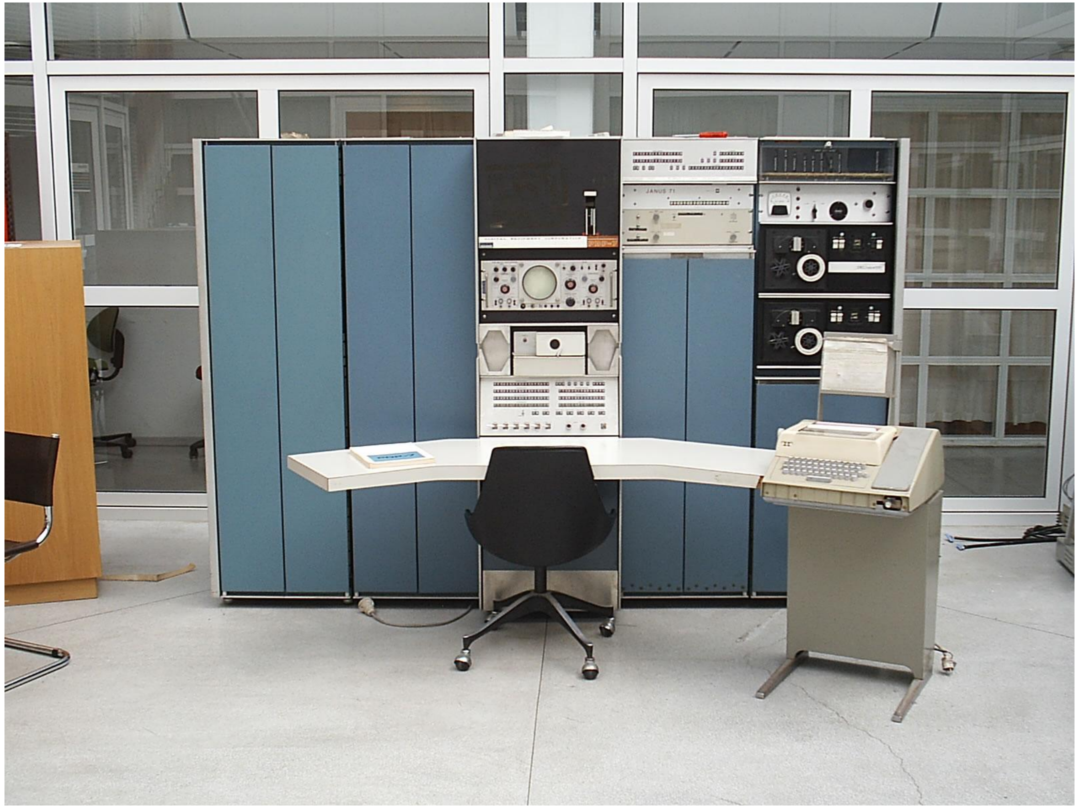
Single Pass

Storage limitations on the B compiler demanded a one-pass technique in which output was generated as soon as possible, and the syntactic redesign that made this possible was carried forward into C.

- Dennis M. Ritchie, The Development of the C Language

(The B compiler ran on PDP-7 with 8K of 18-bit words!)

DEC PDP-7



Compiler Passes?

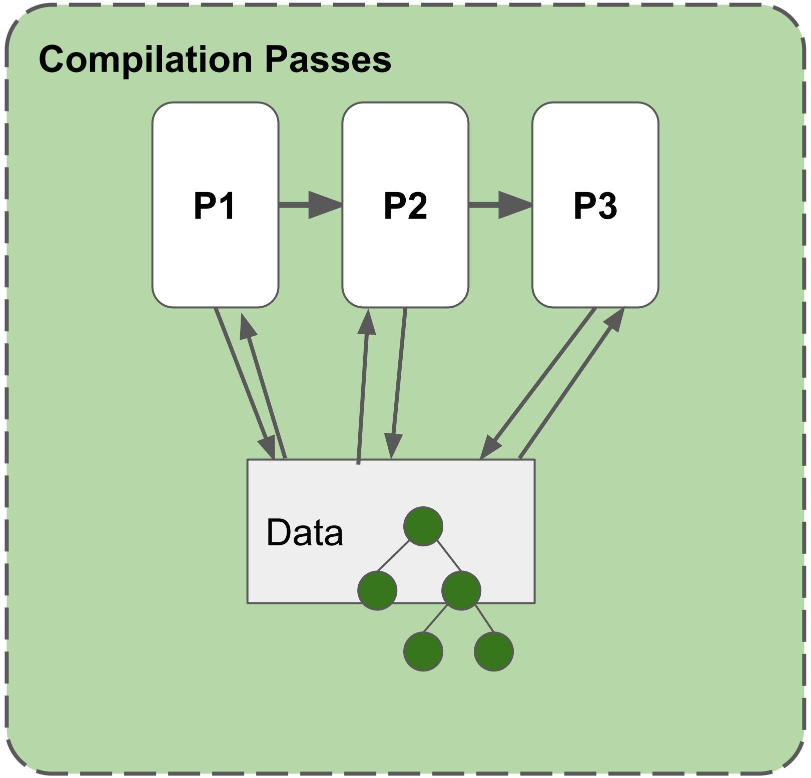
Conventional compilers are structured around **passes**.

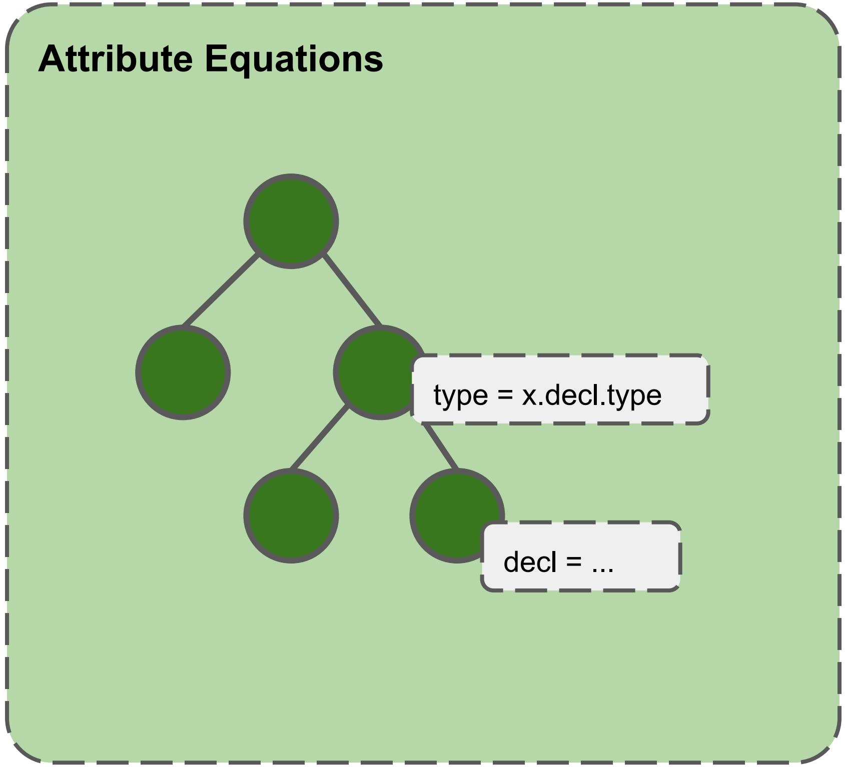
One pass: all translation done while parsing.

Multiple passes improve modularity.

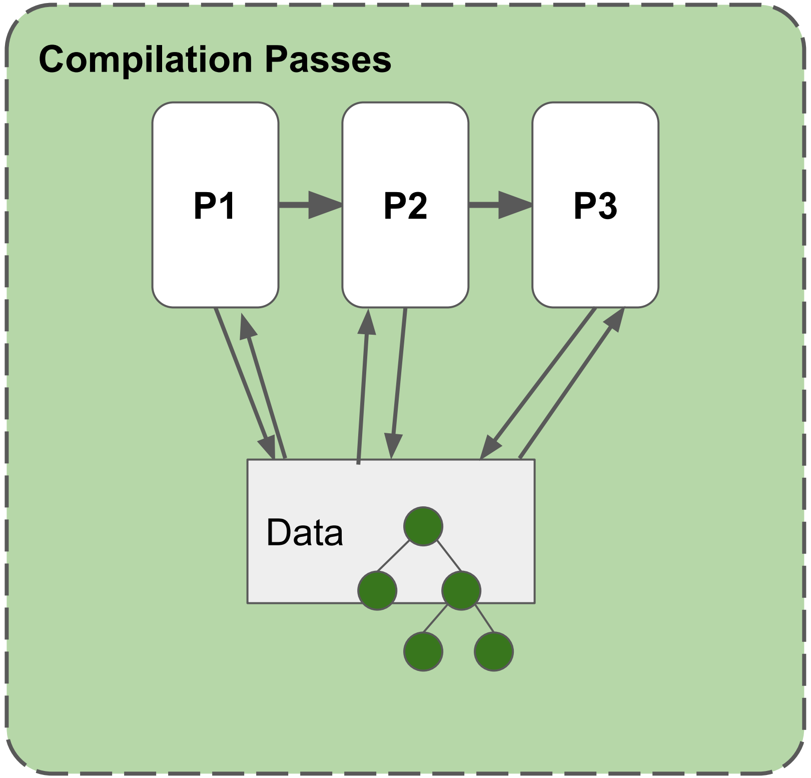
Machine-dependent output in separate pass, easy to swap target machine.

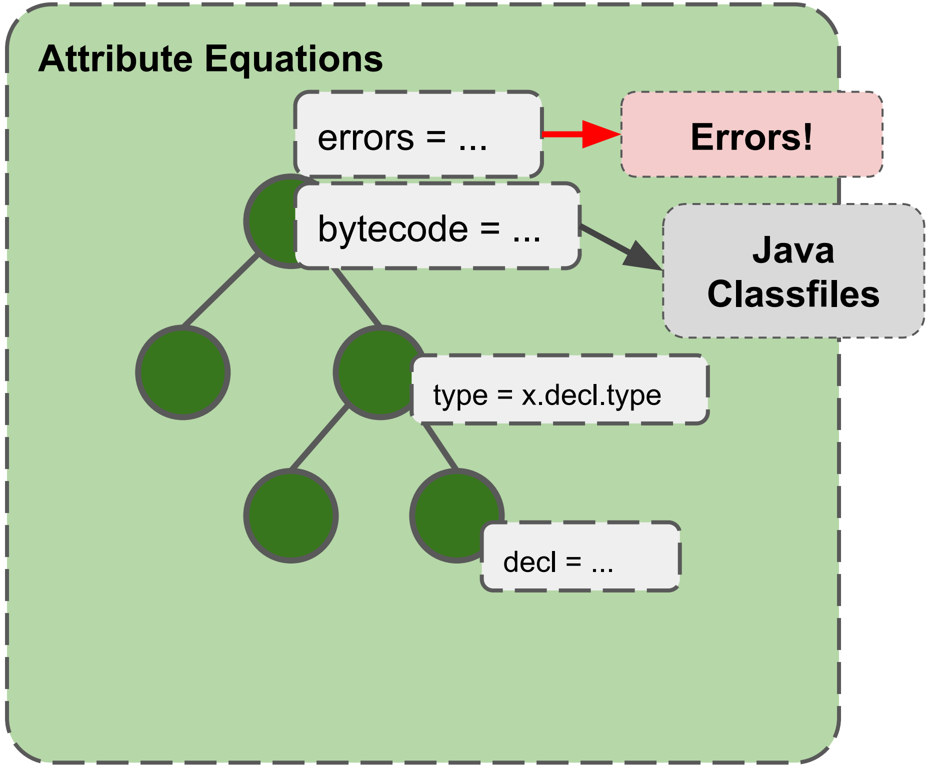
Pass-Oriented vs. Attribute-Oriented





Pass-Oriented vs. Attribute-Oriented





No Passes?

Using attributes:

- Compilation is divided into small computations (attributes).
- Attributes are declarative

  Say **what** should be computed, not **how** to do it.
- Attributes have no side-effects. Order independent!
- Attribute evaluator schedules attributes:
  - Lazy evaluation.
  - Automatic parallelization.

Lazy Evaluation

If an attribute (instance) is not needed, it will not be computed.

Avoids redundant computation for unused features.

Automatic Parallelization

Attributes are observationally pure:

- No side effects.
- Not order-dependent.

Thus, attributes can be parallelized.

Speedup depends on attribute structure.

For ExtendJ, speedup of 2x is possible.

Attribute-Oriented Compiler

How to make a full compiler with attributes?

1. Split computations into meaningful attributes.
2. What should be synthesized/inherited?
3. What should be implicitly generated with higher-order attributes?

ExtendJ Design

Specification divided into modules based on Java version.

All types are represented in AST (user-defined and primitives)

Generic types are represented with higher-order attributes.

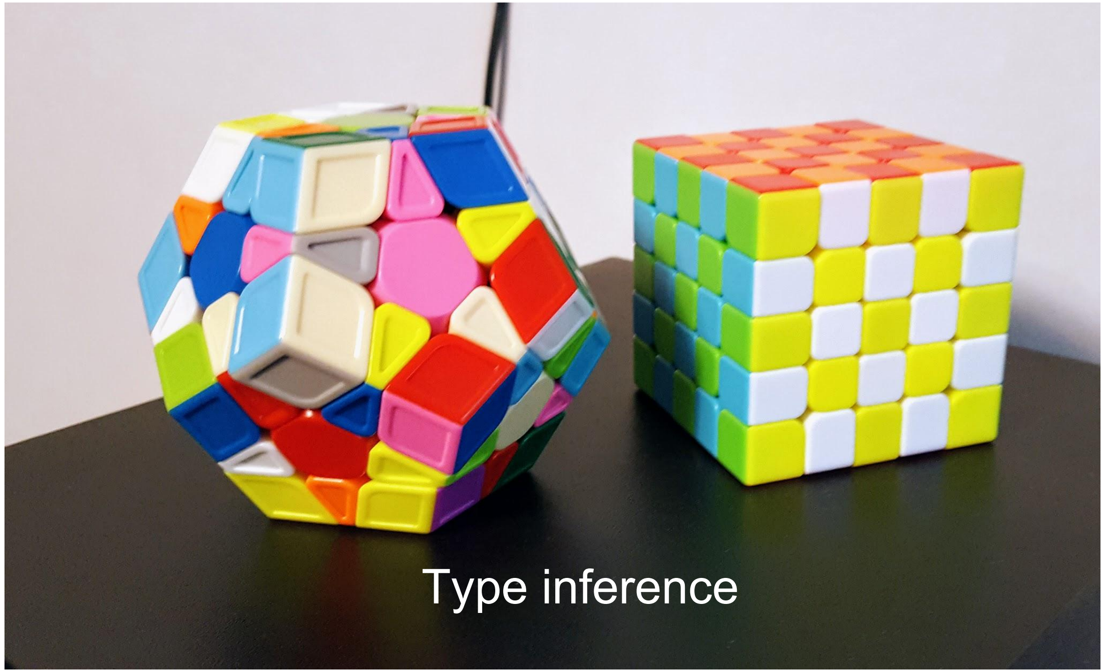
Type and name lookup is demand-driven (no precomputation of symbol tables).

Minimal use of AST transforms (rewrites). Instead, try to use higher-order attrs.

Type inference is least obvious part.

ExtendJ Challenges

Java is a very complicated language.



ExtendJ Challenges

Java is a very complicated language.

The official compliance test suite is proprietary, so we use our own regression tests and regular testing on Open Source projects to find errors.

ExtendJ is not perfectly compliant, but close enough for our needs.

Attributes have a performance cost. ExtendJ is a few times slower than javac.

ExtendJ Overview: AST

Everything is a **declaration** or an **access**:

- TypeDecl
- MethodDecl
- VarAccess
- MethodAccess
- TypeAccess

ExtendJ Overview: AST

```
Program ::= CompilationUnit*;

CompilationUnit ::= TypeDecl*;

abstract TypeDecl ::= BodyDecl*;
ClassDecl : TypeDecl;
InterfaceDecl : TypeDecl;

abstract Stmt;
abstract Expr;
Access : Expr;
```

Source files

Member methods/fields

Statements (if/for/…)

Named type/member use

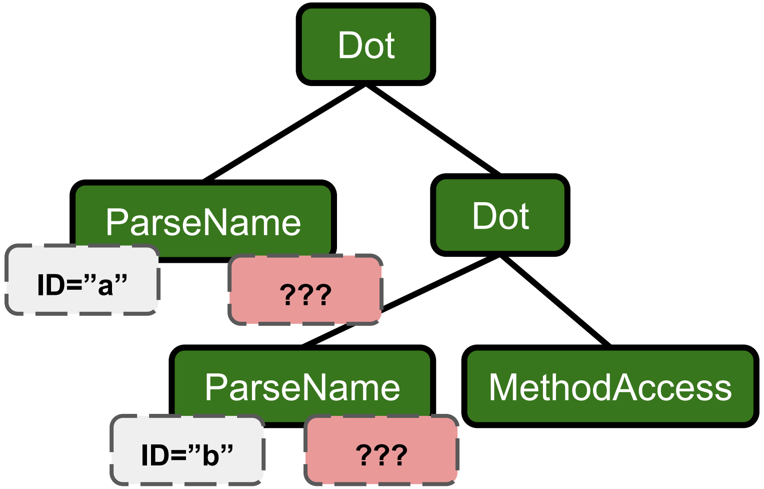
ExtendJ Overview

- Name analysis
  - Classification, lookup
- Type analysis
  - Lookup, subtyping, generics
- Definite assignment
- Normalization
  - Multiple declaration, enhanced for, try-with-resources, lambda
- Implicit code gen
  - Accessors, bridge methods
- Bytecode output

Name analysis: classification

Parsed names must be resolved based on context:

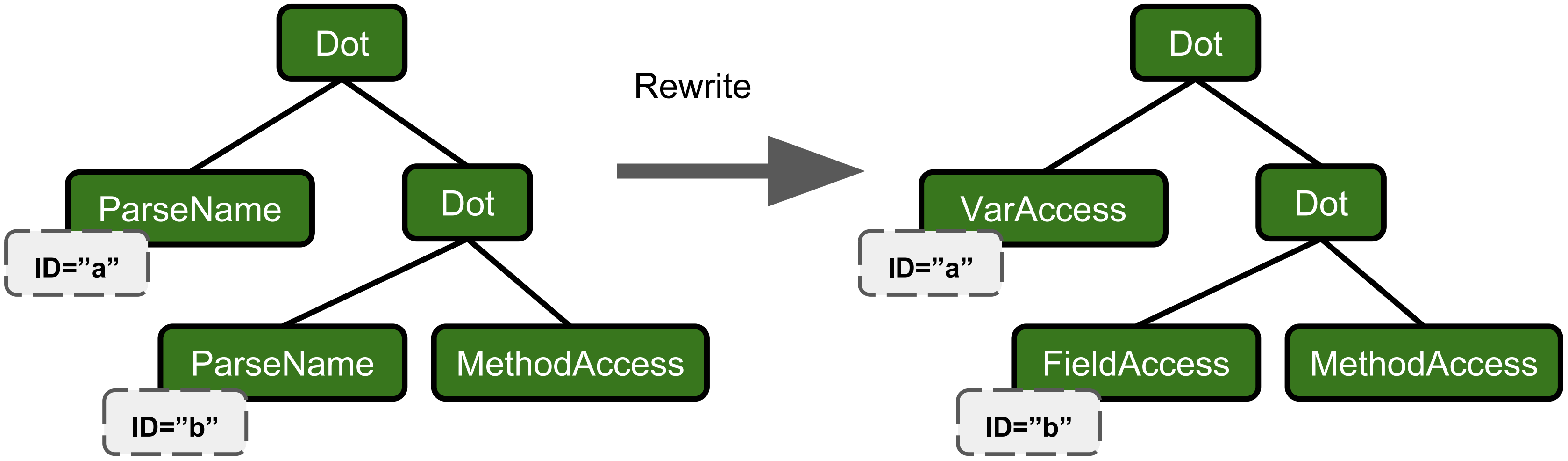
```
a.b.c();
```



Name analysis: classification

Parsed names must be resolved based on context:

```
a.b.c();
```



Name analysis: classification

Parsed names must be resolved based on context:

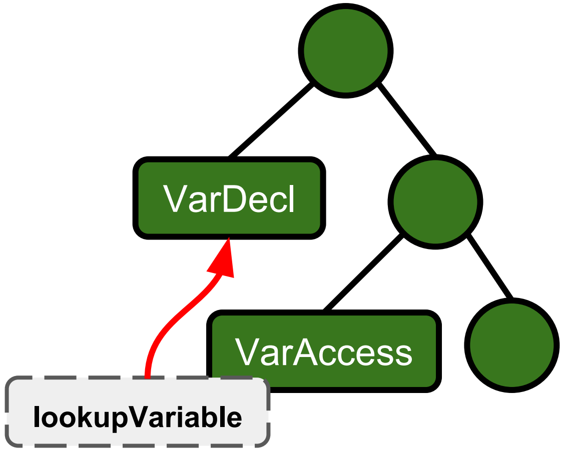
```
a.b.c();
```

Parse name rewrite is done using higher-order attributes!

Name analysis: lookup

Inherited attributes for name lookups:

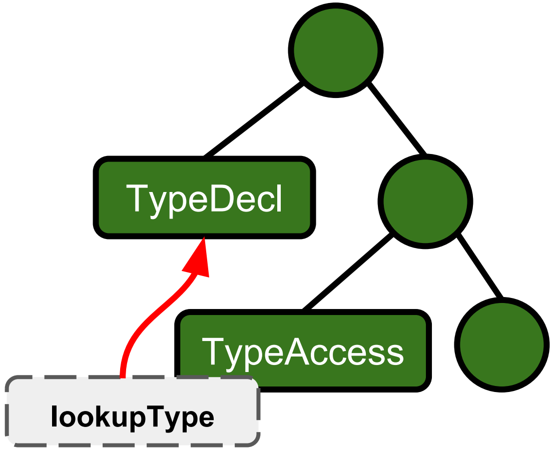
```
lookupVariable(String name)
lookupMethod(String name)
lookupConstructor(String name)
```



Type analysis

Type lookup works like name lookup:

```
lookupType(String pkg, String type)
```



Type Analysis: Subtyping

Double dispatch:

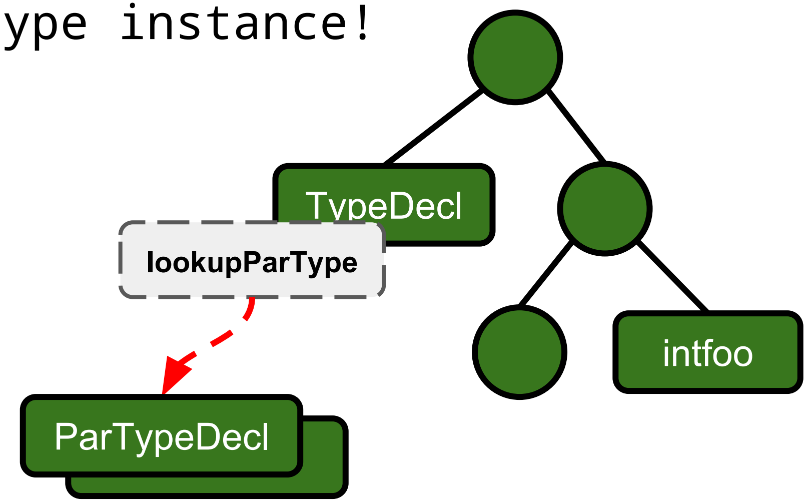
```
syn boolean TypeDecl.subtype(TypeDecl other);
eq ClassDecl.subtype(TypeDecl other) = other.subtypeClassDecl(this);

syn boolean TypeDecl.subtypeClassDecl(ClassDecl other) = false;
eq ClassDecl.subtypeClassDecl(ClassDecl other) = …;
```

Type Analysis: Implicits

Higher-order attributes for implicit types
(primitives, parameterized types)

```
class Foo<T> {}
Foo<Integer> intfoo;  // Need type instance!
```

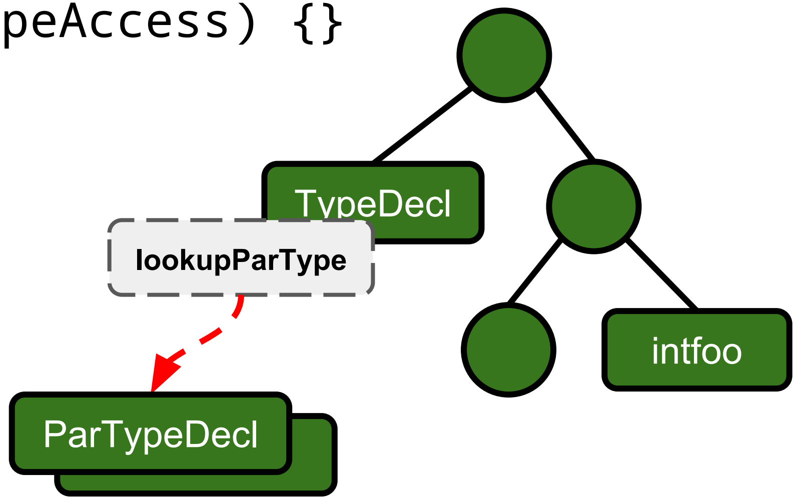


Type Analysis: Implicits

Higher-order attributes for implicit types
(primitives, parameterized types)

```
nta ParTypeDecl
    TypeDecl.lookupParType(ParTypeAccess) {}
```

```
ParClassDecl : ClassDecl ::=
    Parameterization;
```



Extensibility

Extensibility is easy with JastAdd:

- Fine-grained control with **Aspect Oriented Programming** and **Attributes**.
- Extensions can change everything!

However, this leads to **fragile** extensions:

- Everything is exposed for extension

  => can not change internals without affecting existing extensions!

Open problem: ExtendJ needs to be refactored to improve, but this hurts existing users.

Parallelizing ExtendJ

I have been working on parallelizing ExtendJ.

Currently, it’s about twice as fast as the sequential version.

Parallelization is done using our recent developments supporting concurrent attribute evaluation.

The concurrent evaluation research was supported by a 2015 Google Faculty Research Award.

Demo

<Demo>: Parallel execution.

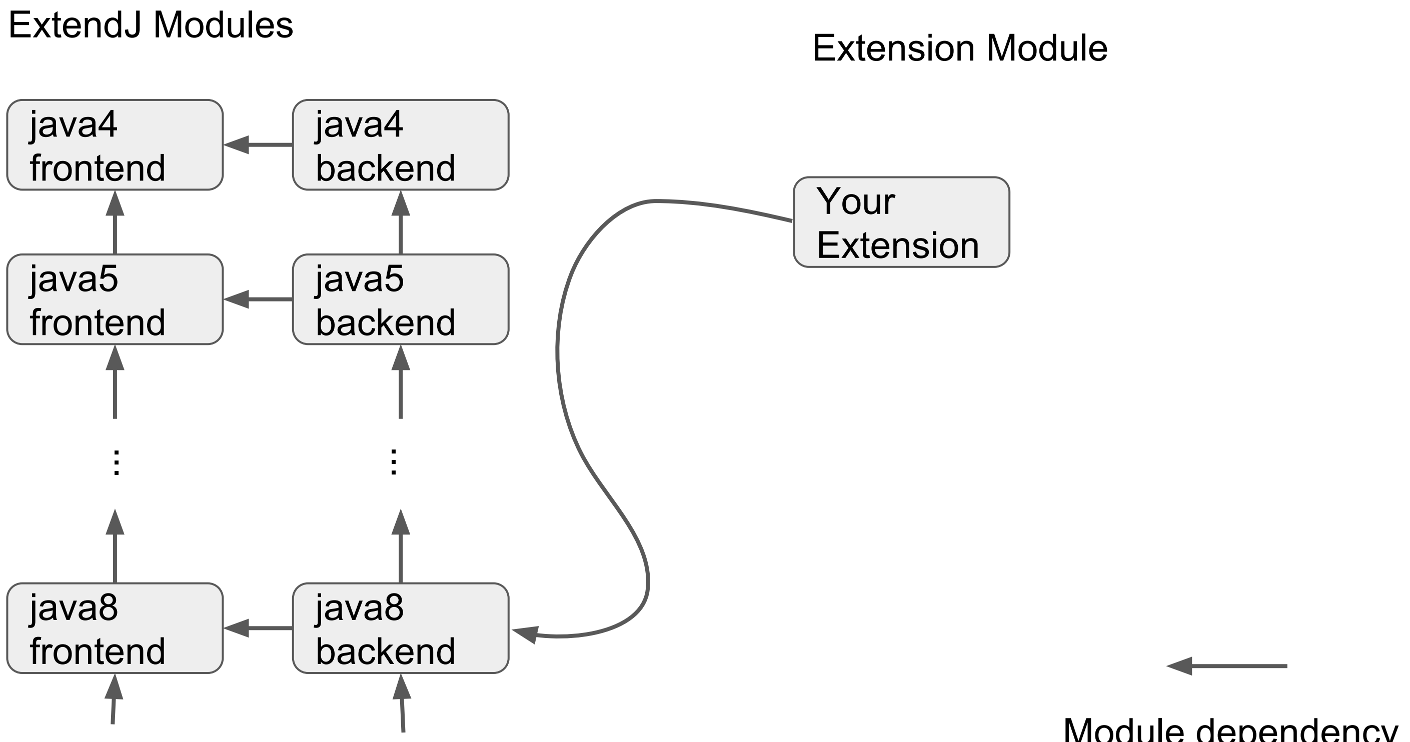
Building Extensions

I developed a Gradle plugin to easily build extensions with ExtendJ!

Gradle plugin: JastAddGradle

The extension is specified in a module specification, which is used to compile together with some base modules from ExtendJ.

Building Extensions



Extensions: Getting Started

A small template project to get started with building an extension:

Extension Base Project: https://bitbucket.org/extendj/extension-base

Demo

<Demo>: Extension Base (StringEq).

Thank You

Thanks for listening!

Learn more at:

extendj.org and **jastadd.org**

Jesper Öqvist

Read my blog https://llbit.se

Upvote my instagram

extendj.org