Mixed language project compilation in Eclipse: Java and Groovy

Andy Clement, SpringSource/VMware

Agenda

- Me
- Groovy-Eclipse version 2
  - Quick review of Joint Compilation
  - A new approach in Eclipse … and why
  - The tricky parts
  - What had to change… in groovy and in eclipse

Joint Compilation: What is it?

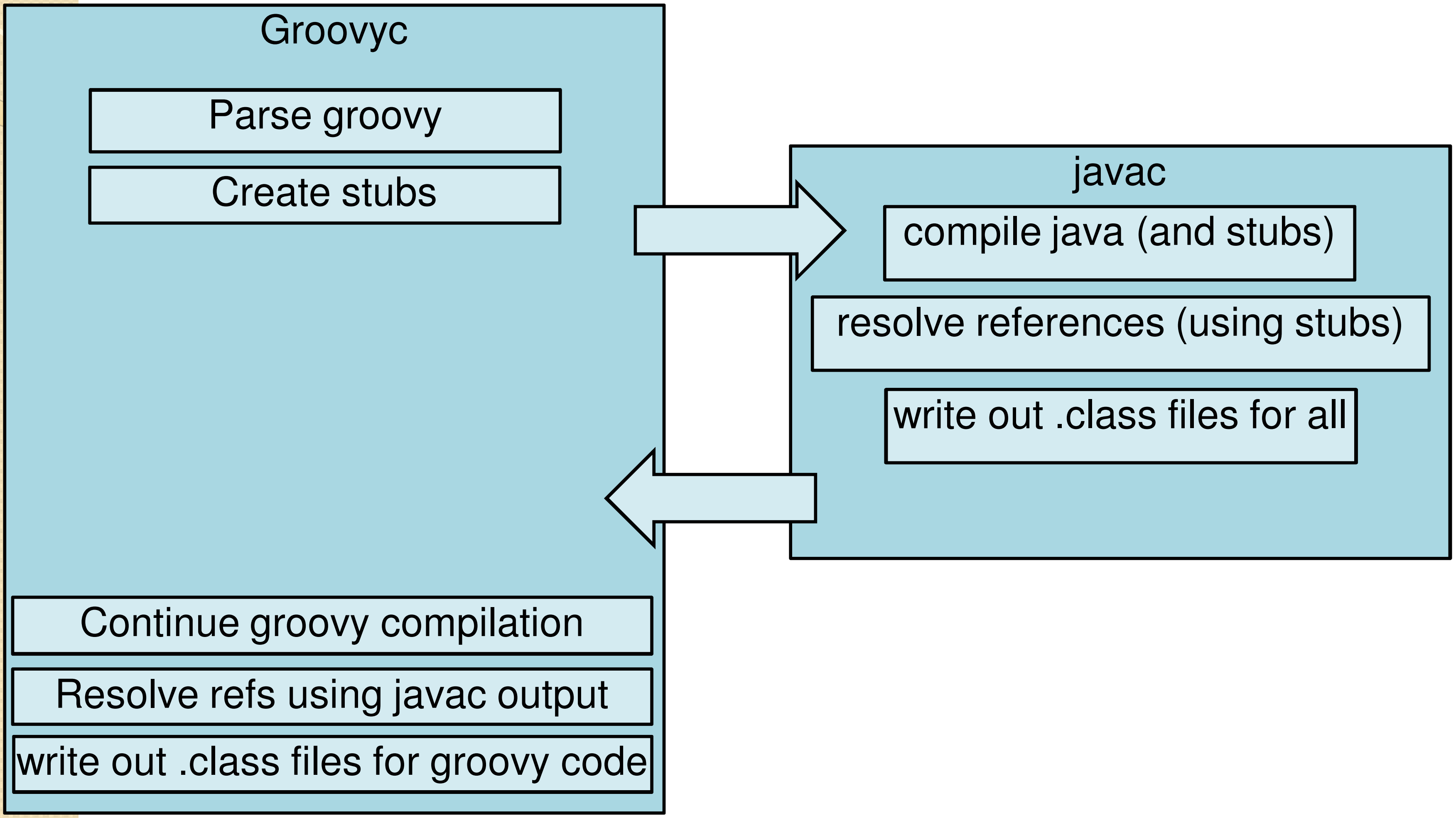
- For compilation of multi-language codebases
  - e.g. java/groovy in my case
  - Multiple compilers working together
  - What makes it an interesting problem?
    - dependencies

```
Top.groovy

class Top {
}
```

```
Middle.java

class Middle extends Top{
}
```

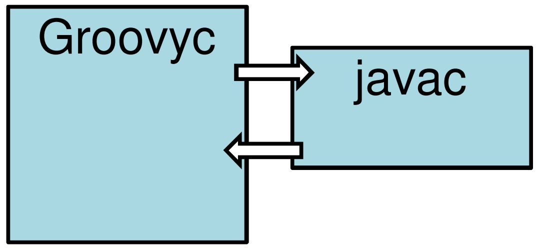
```
Bottom.groovy

class Bottom extends Middle {
}
```

Joint Compilation: Breaking it down


Groovyc
Parse groovy
Create stubs
javac
compile java (and stubs)
resolve references (using stubs)
write out .class files for all
Continue groovy compilation
Resolve refs using javac output
write out .class files for groovy code

Joint compilation: observations

- Simple communication
  - Using text files on disk
    - Java stubs for groovy>javac
    - .class files for javac>groovyc
  - Some wasted processing
    - Javac stub parsing and compilation to .class files
- Can we do better?
  - **Why** would we want to?



Why change the approach?

- Groovy-Eclipse pre v2.0
  - Used joint compilation, but proved quite unreliable
  - Eclipse JDT just didn't understand groovy that well

- Can we make Eclipse more easily understand?
  - Eclipse compiler can be extended (AspectJ does it)
  - Groovy compiler phases well suited to ECJ integration
  - Will IDE functionality spring to life?

Building language IDE support is extremely expensive

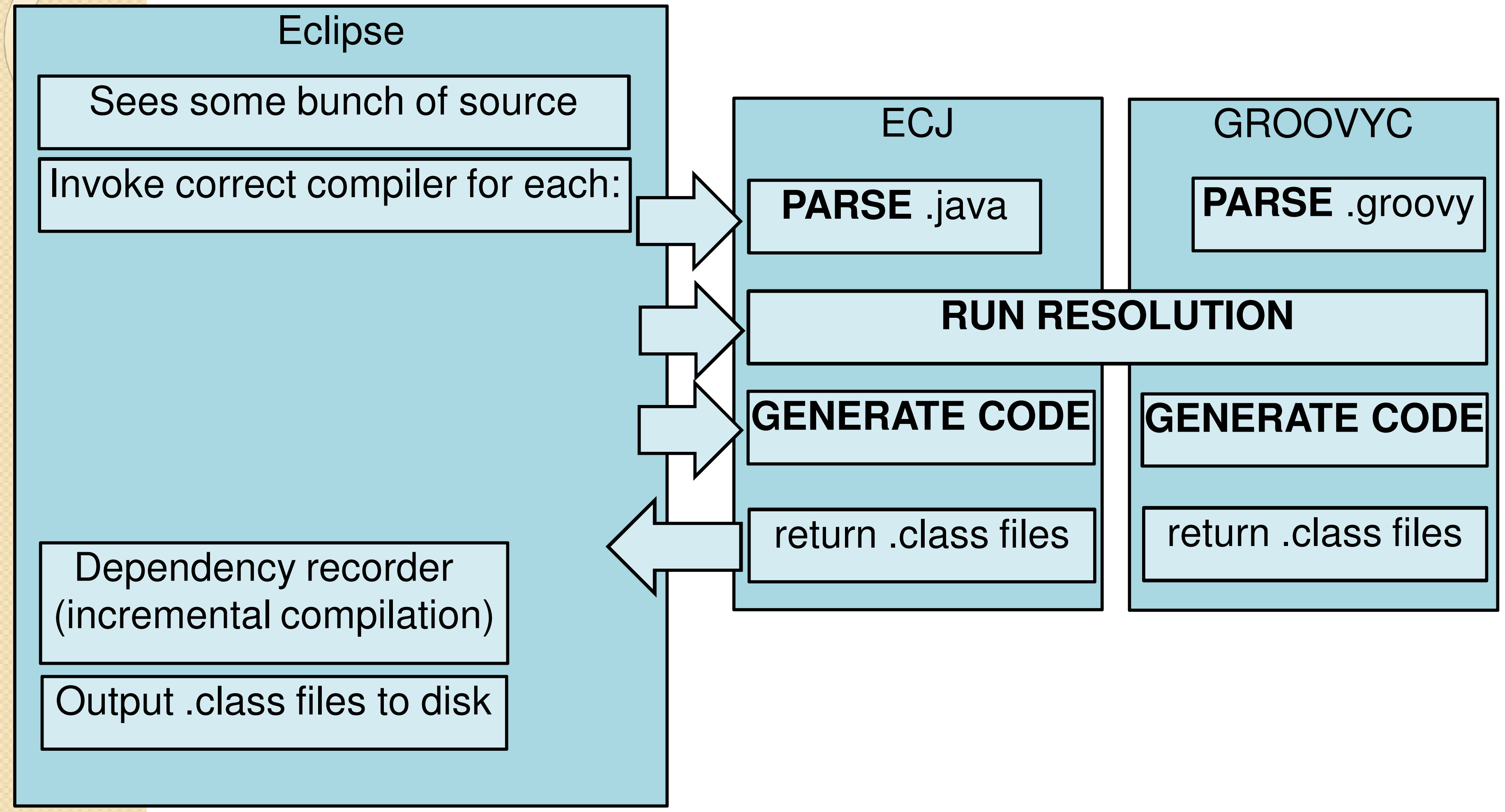
Groovy Eclipse v2

- The Plan
  - Continue with principle of joint compilation
    - Compilers working together
      - Understand each others data structures to a degree
  - Reduce unnecessary processing
  - Modify ECJ in a language independent way
    - (more likely to get changes into Eclipse base)
- Measures of success
  - Do some Eclipse JDT features ‘just work’ for groovy

Compiler integration: all the way down

- Both compilers integrate at each of these stages:
  - **Parsing** of the text into an internal representation
  - **Resolution** of the references using the rules of that language
  - **Code generation**

In Groovy Eclipse v2.0


Eclipse
Sees some bunch of source
Invoke correct compiler for each:
Dependency recorder (incremental compilation)
Output .class files to disk
ECJ
PARSE .java
GENERATE CODE
return .class files
GROOVYC
PARSE .groovy
RUN RESOLUTION
GENERATE CODE
return .class files

Data structures

- No new meta-model
  - We're in Eclipse, use the eclipse structures
    - Languages that can be interacted with from Java will have an eclipse compiler representation
    - **Eager** mapping from groovy to ECJ
    - **Lazy** mapping from ECJ to groovy

Demo

- Groovy-Eclipse

The benefits

- No disk communication
  - No stub creation
  - Some translation between groovy and eclipse structures where necessary
- Incremental compilation just works (!)
  - And across eclipse restarts
- Usable as a batch compiler
  - ECJ can be used from Ant or command line
- Unexpected benefits
  - ECJ checks some groovy structures,
    - e.g. was checking generics

Of course I'm lying…

- It wasn't all straightforward
  - Reconciling
- Not all UI functionality is free
  - Syntax highlighting, inferencing, navigation, code assist
  - But built more rapidly because of the underlying architecture

Of course I’m lying…

- Ongoing problems handling 1.6/1.7/1.8 of groovy

- I am a now a patch monkey ☹
  - Changes required to the eclipse compiler
    - Versions 3.4/3.5/3.6 all different
  - Changes required to the groovy compiler
    - Versions 1.6/1.7 and shortly 1.8

- Really want to get these patches into those base compilers – what kinds of change?

What kinds of change: groovyc

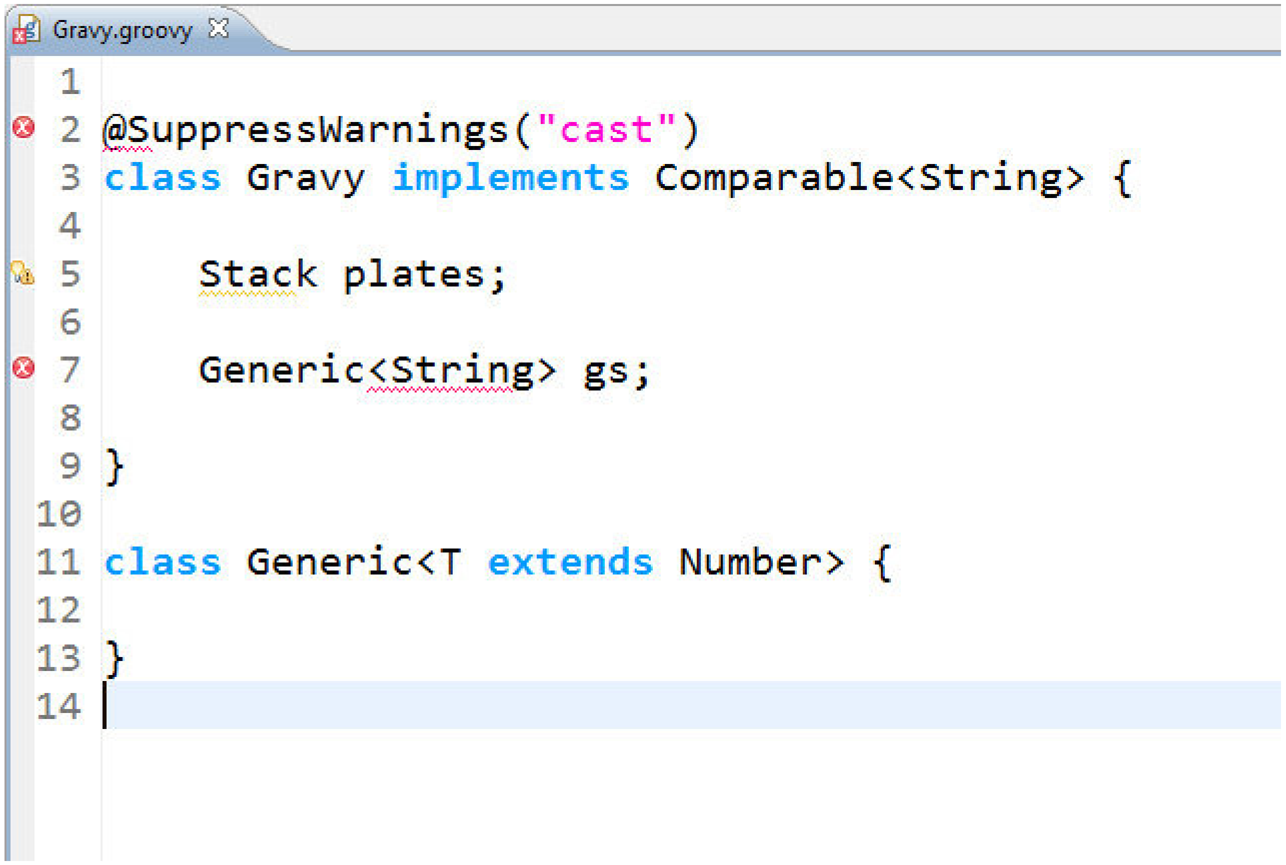
- Needed to improve parser recovery
  - In the IDE, the compiler is usually seeing broken code

```
class Foo {
  void moo() {
      new String().
  }
}
```

  - Comment recording
  - AST node positioning
- Resolving and ClassNodes
  - More than two kinds of resolved ClassNodes

<demo>

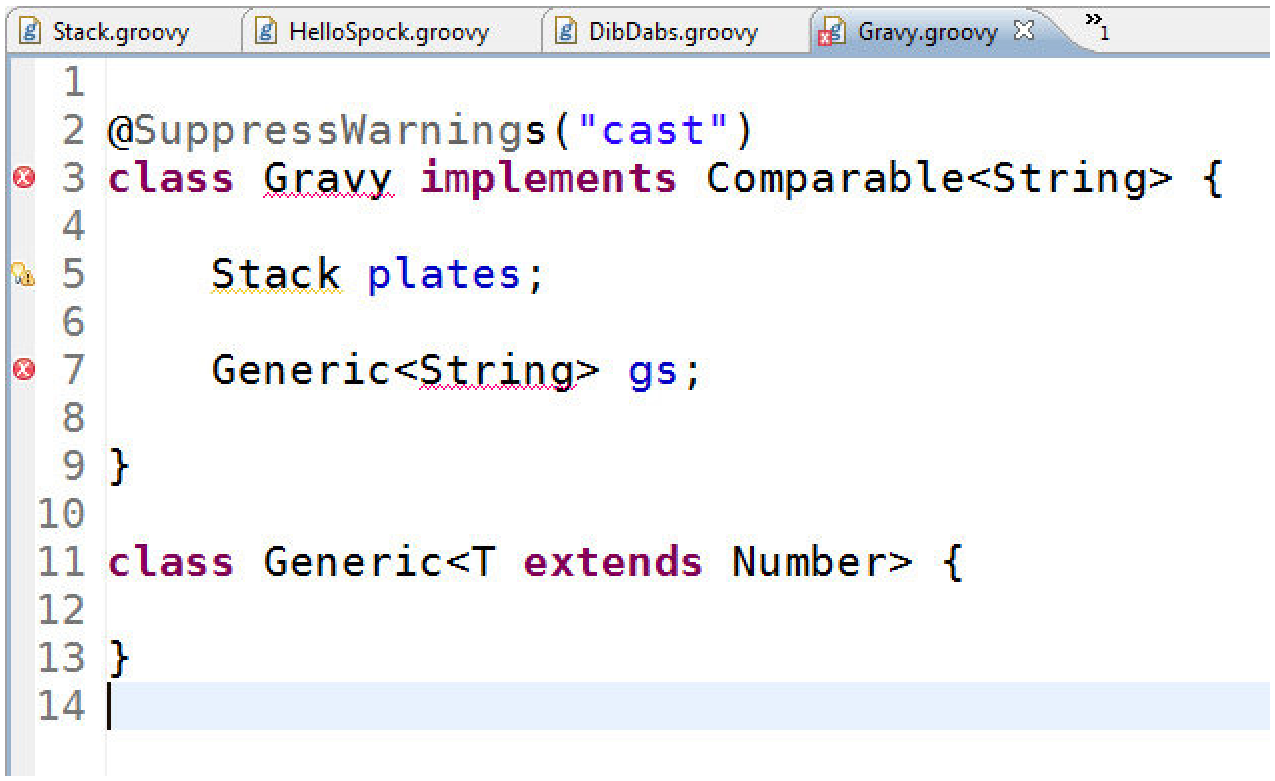
Positions: before



Gravy.groovy

```

@SuppressWarnings("cast")
class Gravy implements Comparable<String> {

    Stack plates;

    Generic<String> gs;

}

class Generic<T extends Number> {

}

```

Positions: after



```

@SuppressWarnings("cast")
class Gravy implements Comparable<String> {

    Stack plates;

    Generic<String> gs;

}

class Generic<T extends Number> {

}

```

What kinds of change: ECJ

- As minimal as possible
  - Keep patch small
  - **Do not damage Java compilation**
- Do not mention 'groovy' anywhere
  - ⇒Although changes are groovy shaped
- All parser creates intercepted/redirected
- Visibility changes to allow subclassing
- Error handling adjustments
  - Defer to groovy

And finally

- *AST transforms*
  - Really complicate things
  - The new architecture actually enables better support for them

<demo>

The end

- Groovy-Eclipse v2 release is based on this architecture
  - a huge improvement over version 1
  - Based on previous experience (AJDT), effort to get to this stage reduced by this approach
- What next?
  - Eclipse debugger modifications
  - Feeding changes back to the eclipse base

Questions?
andy.clement@springsource.com