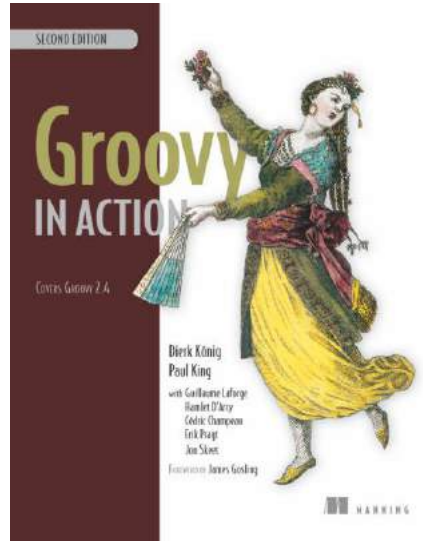
Groovy 2.5, 3, 4 Roadmap

Presented by
Dr Paul King

Dr Paul King

OBJECT COMPUTING

OCI Groovy Lead
V.P. and PMC Chair Apache Groovy



Author:
https://www.manning.com/books/groovy-in-action-second-edition

Slides:
https://speakerdeck.com/paulk/groovy-roadmap

Examples repo:
https://github.com/paulk-asert/upcoming-groovy

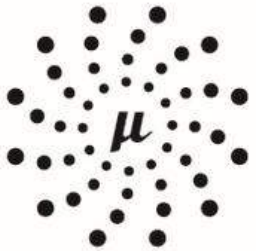
@paulk_asert

WE ARE SOFTWARE ENGINEERS.

OBJECT COMPUTING

We deliver mission-critical software solutions that accelerate innovation within your organization and stand up to the evolving demands of your business.

Grails



MICRONAUT

WE ARE SOFTWARE ENGINEERS.

OBJECT COMPUTING

We deliver mission-critical software solutions that accelerate innovation within your organization and stand up to the evolving demands of your business.

AREAS OF EXPERTISE

INDUSTRIAL IoT

We equip industrial environments with seamless connectivity and real-time analytics that reduce operating costs and deliver on customer demands.

MACHINE LEARNING

We can modernize your legacy applications and enable scalable, integrated AI capabilities tailored to your unique sets of data.

BLOCKCHAIN CONSULTING

We are at the forefront of blockchain technology, and we have practical, real-world experience with its implementation.

CLOUD SOLUTIONS

We combine software engineering expertise with cloud-native architecture to accelerate innovation within your organization.

THE WORLD'S LARGEST OPEN SOURCE FOUNDATION

- 200M+ lines of code in stewardship
- 1,058,321,099 lines of code committed
- 3,022,836 code commits
- 730 individual ASF Members
- 7,000 Apache code committers
- All volunteer community

- 350+ Projects and Initiatives
- 300+ Top-Level Projects
- 52 podlings in the Apache Incubator

$20B+ worth of Apache Open Source software products are made available to the public-at-large at 100% no cost, and benefit billions of users around the world.

Source: apache.org and apachecon.com websites

Groovy Open Collective

Thank you to our Silver Sponsors:

Become a Sponsor

Thank you to our Bronze Sponsors:

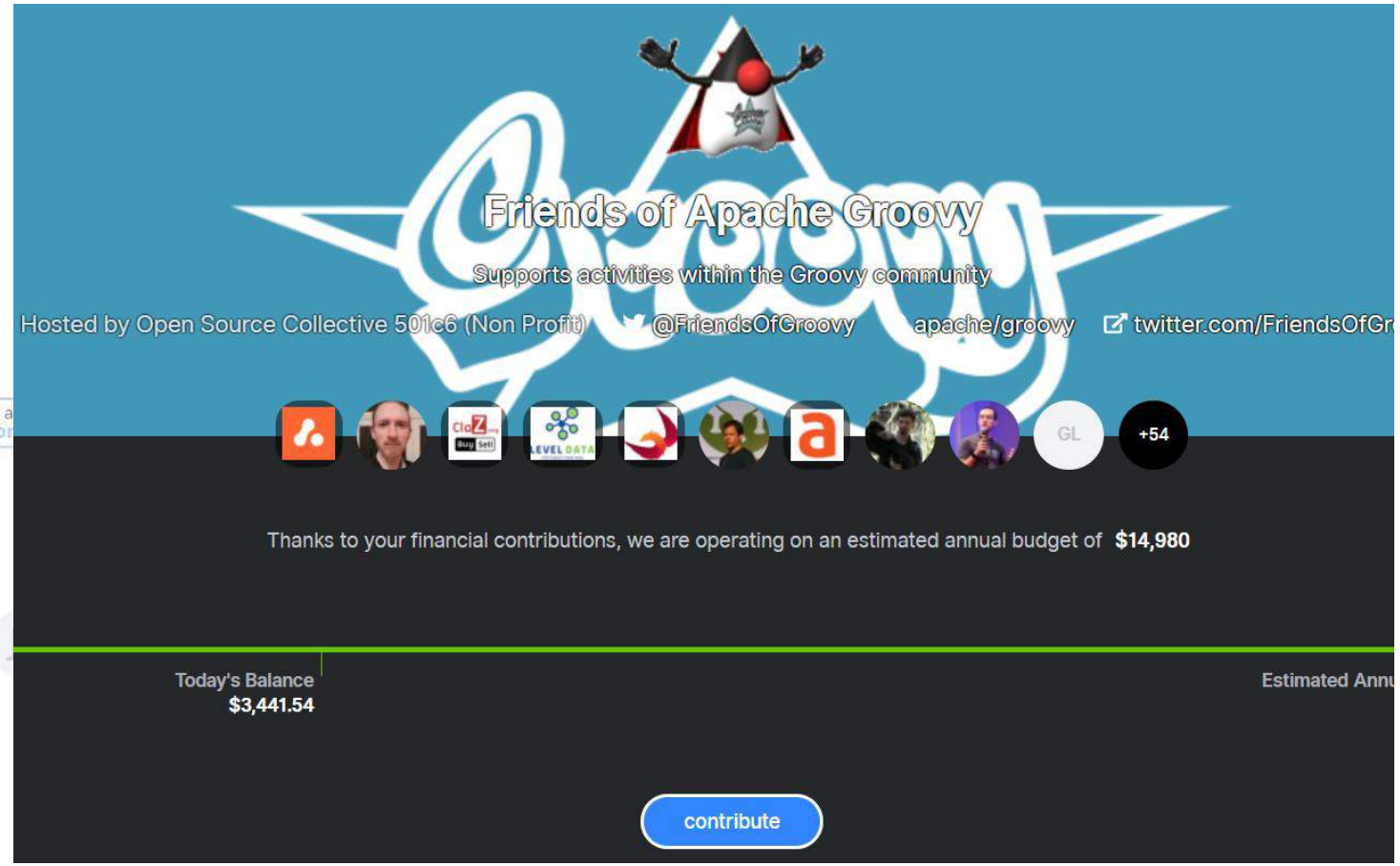
Become a Sponsor

Thank you to our backers(donating monthly):

Become a Backer

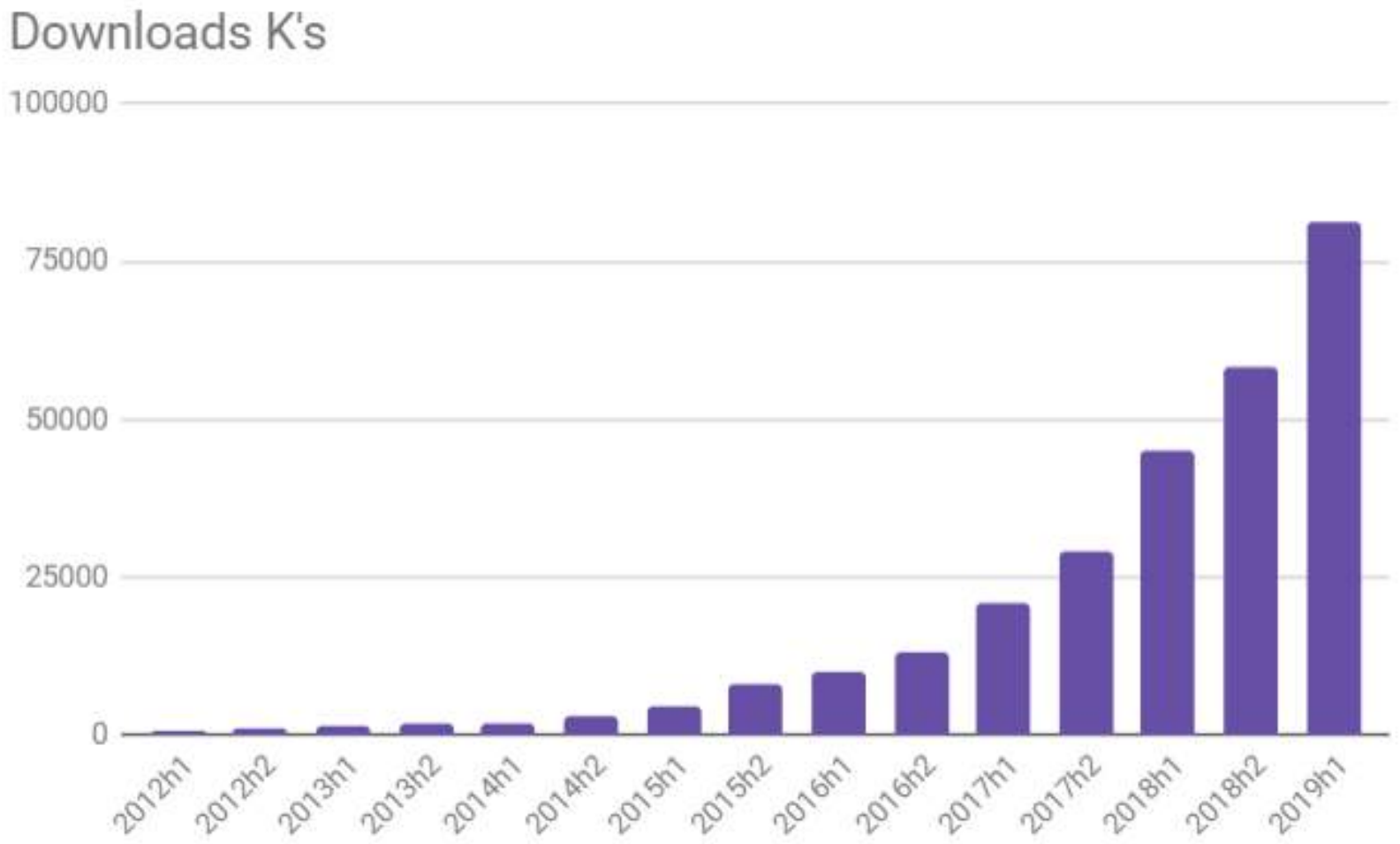
Thank you to all our backers:

Become a Backer



Groovy by the numbers: **Downloads**

- Popular & growing
  2016: 23M
  2017: 50M
  2018: 103M
  currently: approx.
  16M+ per month

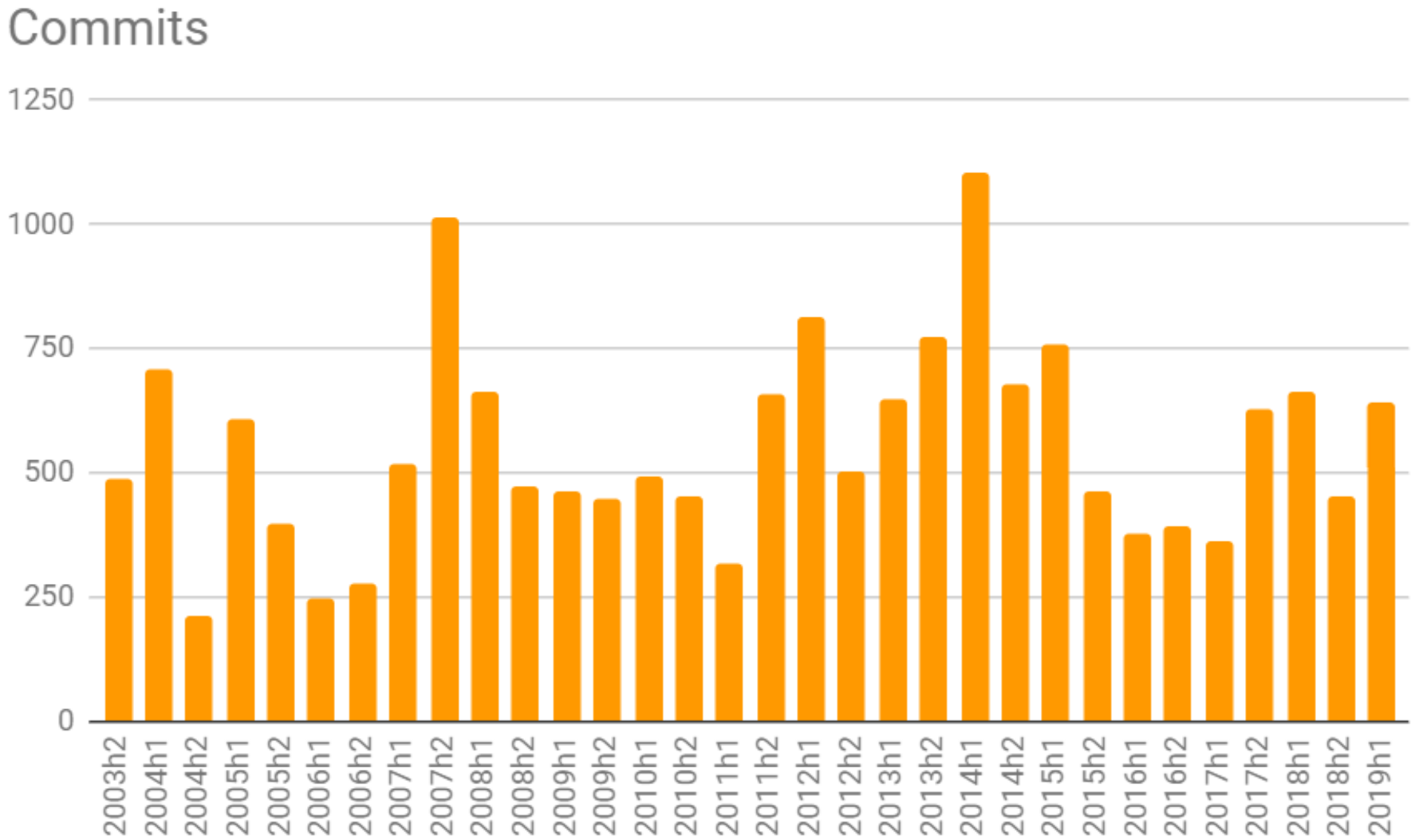


ApacheGroovy @ApacheGroovy · May 6

Did you know that @ApacheGroovy has been downloaded more than 240 million times since its inception in 2003?

Groovy by the numbers: **Commits**

- Steady activity across its lifespan



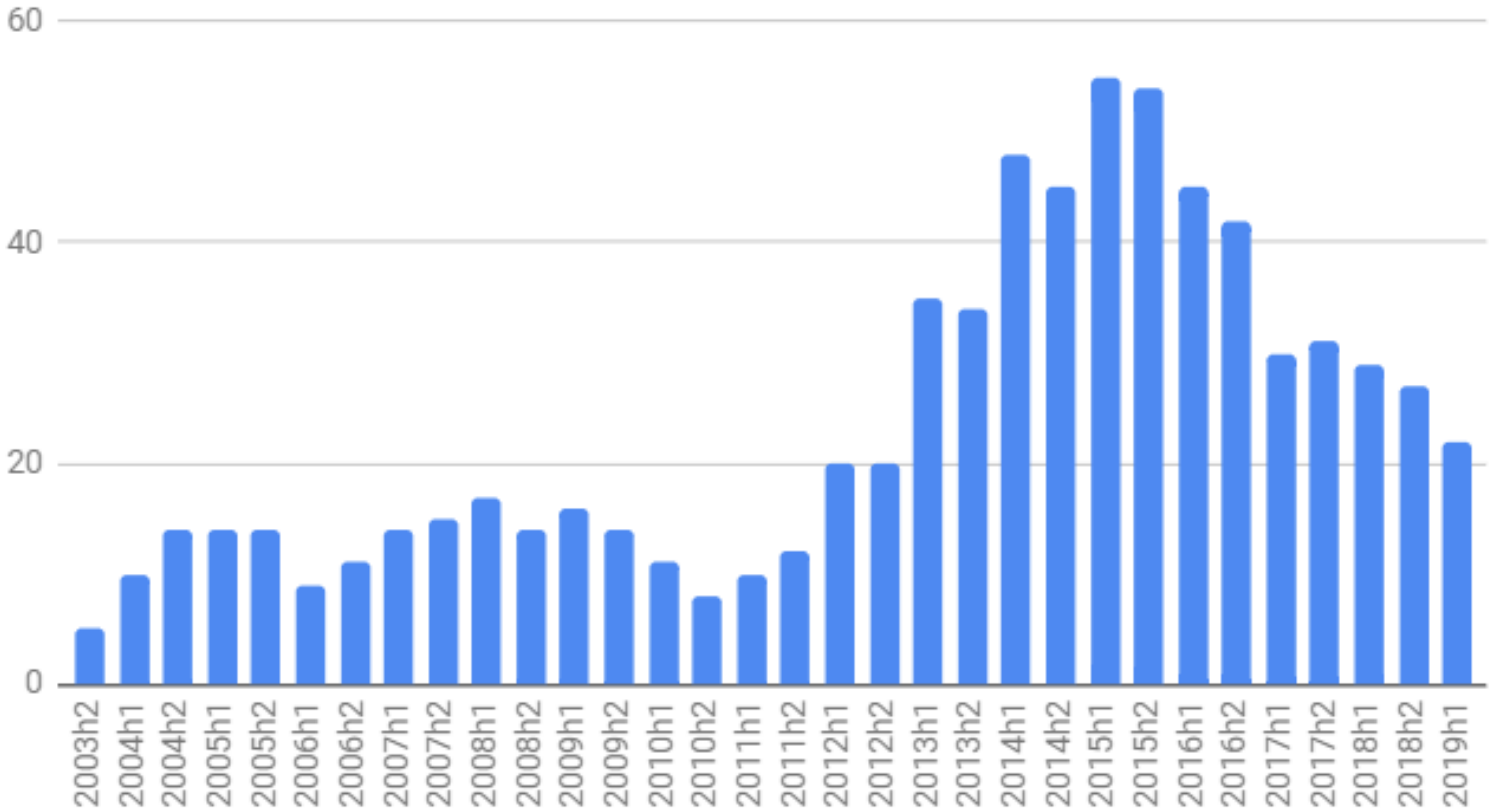
Groovy by the numbers: **Contributors**

Hosting/governance:

CODEHAUS

THE APACHE SOFTWARE FOUNDATION

Contributors:



Sponsorship:

G2One

Spring-Source

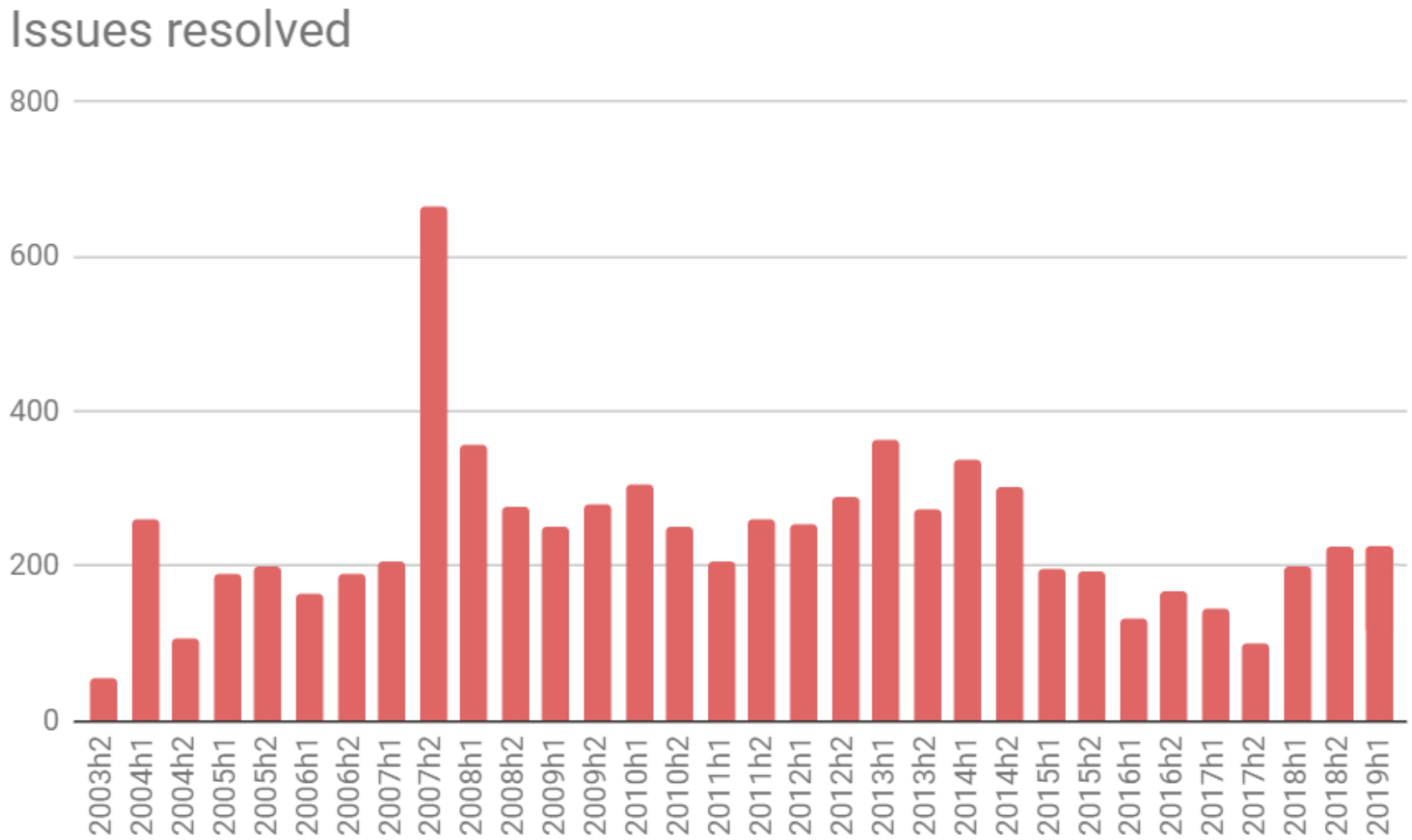
VMware

Pivotal

OCI

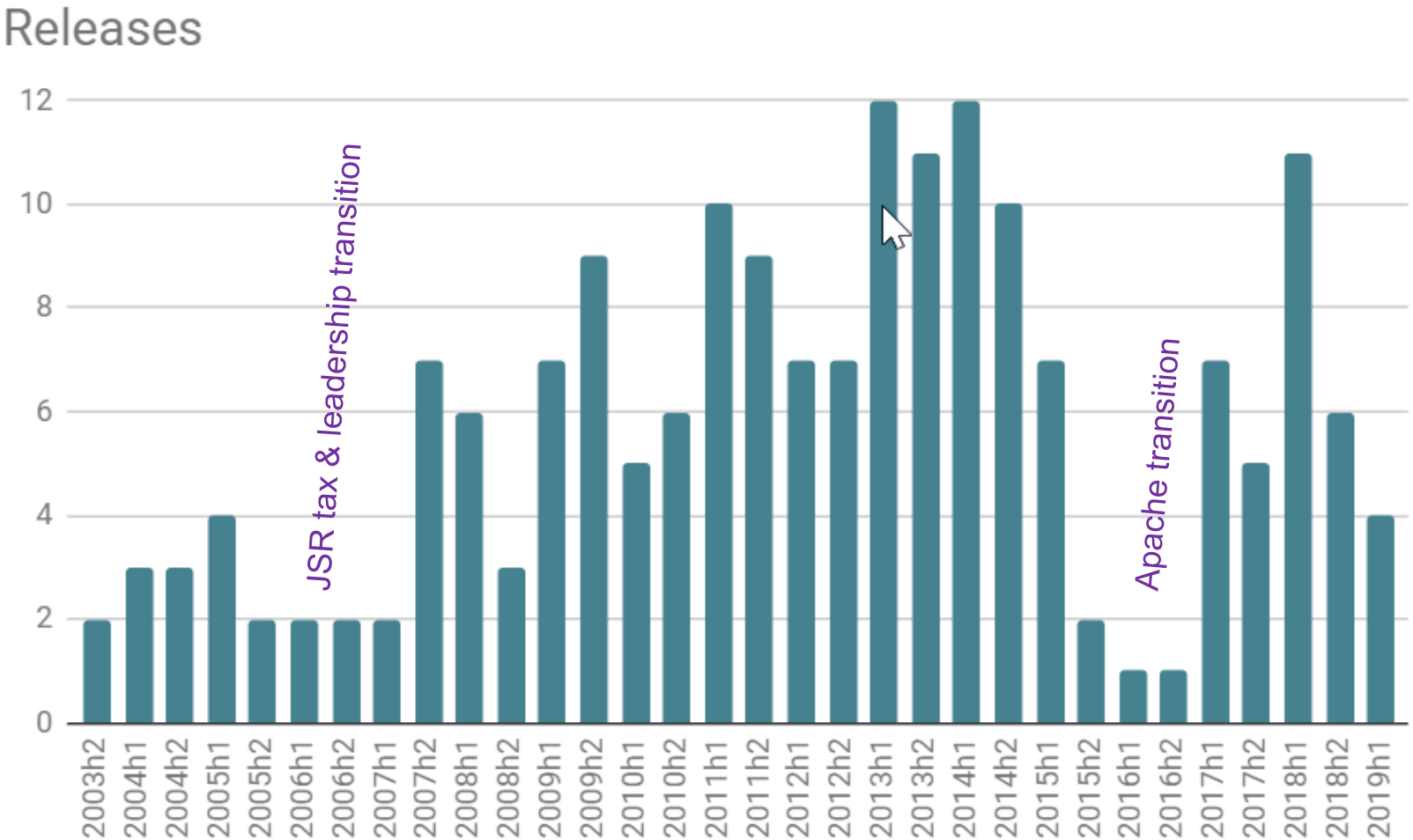
Groovy by the numbers: **Issues resolved**

- Reasonably responsive most of the time



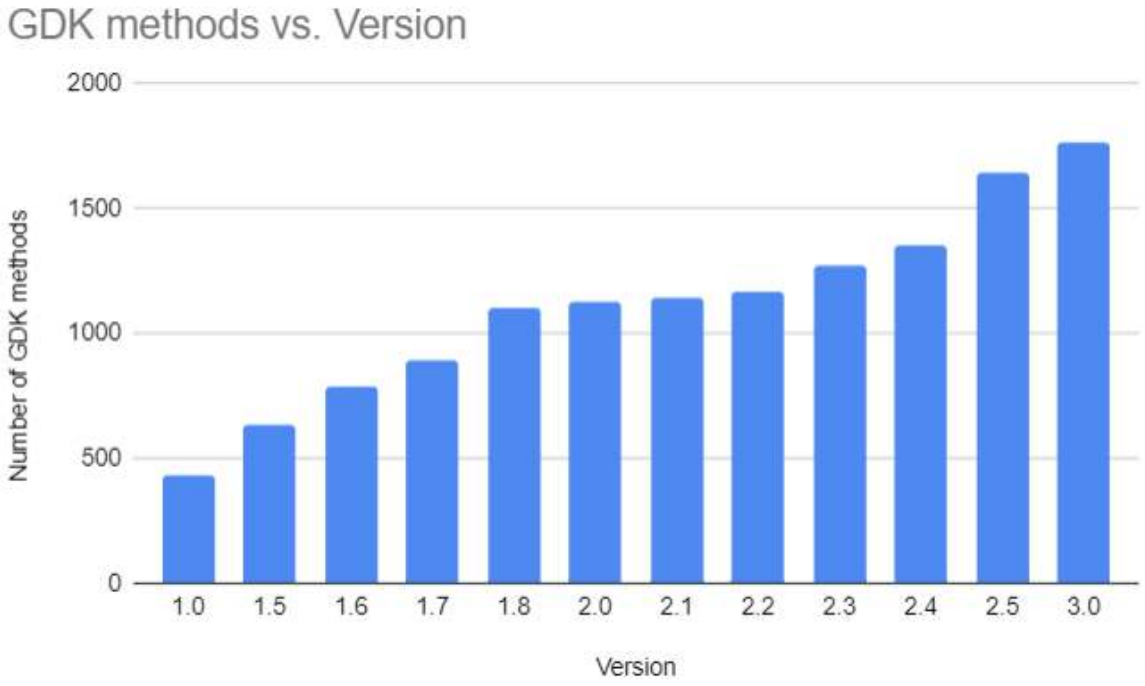
Groovy by the numbers: **Releases**

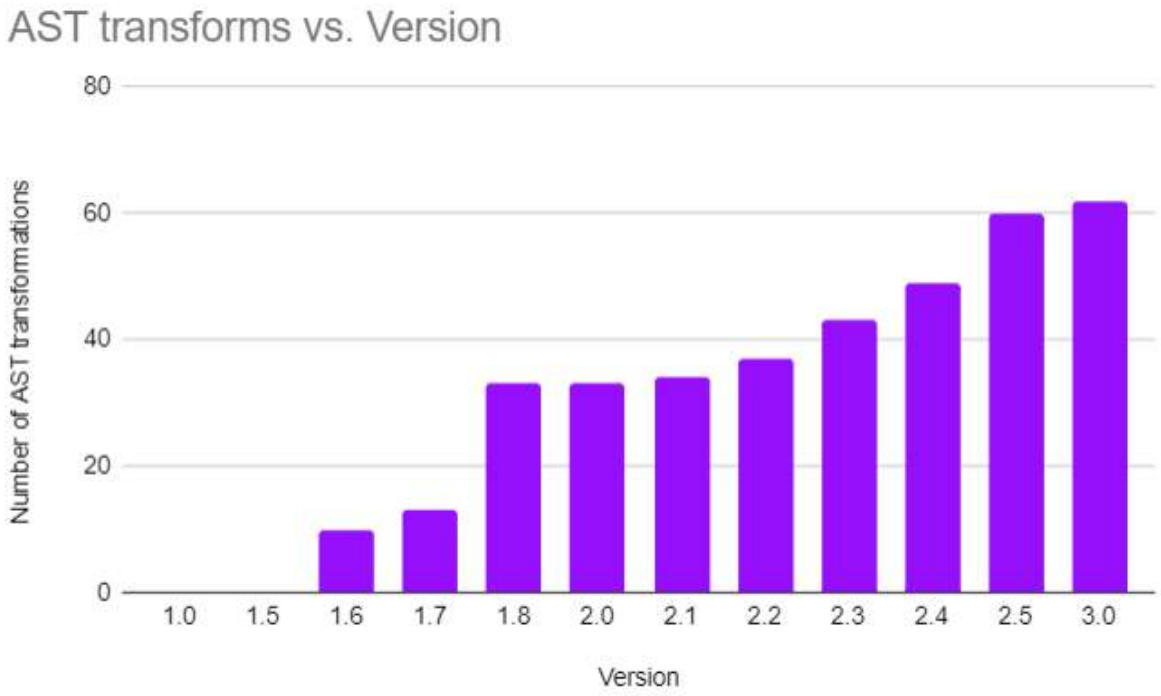
- Healthy cadence for most of its lifetime



Groovy by the numbers: **Enhancements**

- On-going innovation





Groovy Roadmap

- **Groovy 2.5**
  - 2.5.8 released
  - Macros, AST transformation improvements, various misc. features
  - JDK 7 minimum, runs on JDK 9/10/11 with warnings
- **Groovy 3.0**
  - Beta-3 out now, RC-1 in the next week or two, GA later in 2019
  - Parrot parser, various misc. features
  - JDK 8 minimum, address most JDK 9/10/11 issues
- **Groovy 4.0**
  - Alphas out soon
  - Further module support, split packaging
  - Indy-only jar


2.5

What is Groovy?

Groovy = Java

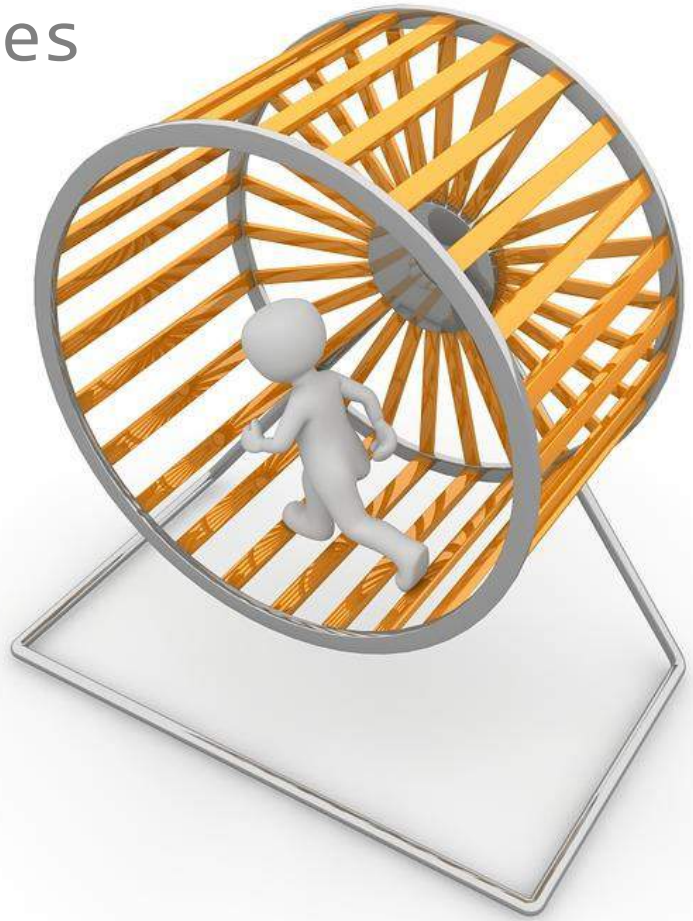
- boiler plate code
+ extensible dynamic and static natures
+ better functional programming
+ better OO programming
+ runtime metaprogramming
+ compile-time metaprogramming
+ operator overloading
+ additional tools
+ additional productivity libraries
+ scripts & flexible language grammar

What is Groovy?

Groovy = **Java**

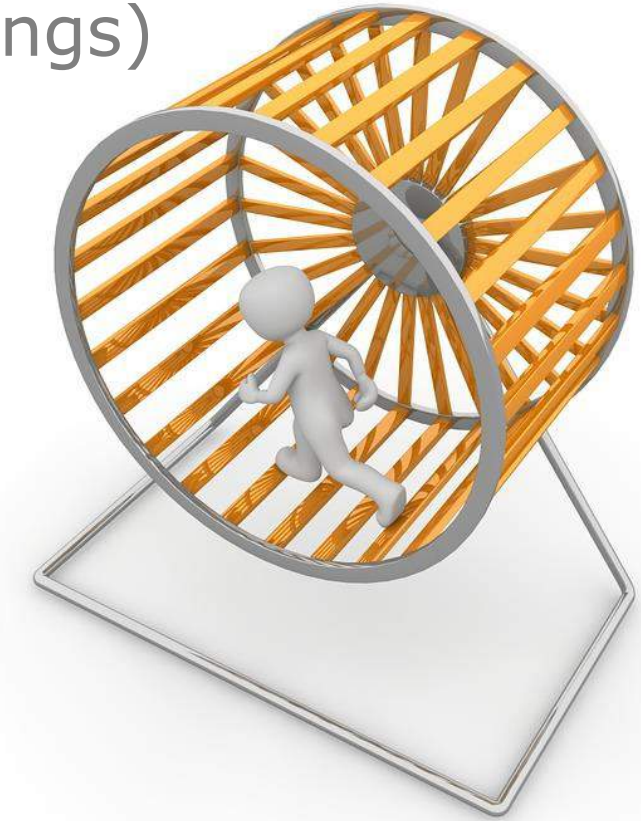
- – boiler plate code
- + extensible [illegible]amic and static natures
- + better f[illegible]ing
- + better O[illegible]
- + runtime [illegible]
- + compile-time metaprogramming
- + operator overloading
- + additional tools
- + additional productivity libraries
- + scripts & flexible language grammar

As Java evolves,
Groovy must evolve too!



Groovy 2.5 Java compatibility enhancements

- Repeated annotations (JEP 120 JDK8)
- Method parameter names (JEP 118 JDK8)
- GDK methods for java.time package (JSR 310 JDK8)
- Annotations in more places (JSR 308)
- Runs on JDK 9/10/11 (currently with warnings)
- Repackaging towards JPMS



Modules – Groovy 2.4, Groovy 2.5, Groovy 3.0

groovy		groovy-json	
groovy-all	*fat jar -> fat pom*	groovy-json-direct	*optional removed*
groovy-ant		groovy-jsr223	
groovy-bsf	*optional*	groovy-macro	
groovy-cli-commons	*optional*	groovy-nio	
groovy-cli-picocli		groovy-servlet	
groovy-console		groovy-sql	
groovy-datetime		groovy-swing	
groovy-dateutil	*groovy -> optional*	groovy-templates	
groovy-docgenerator		groovy-test	
groovy-groovydoc		groovy-test-junit5	
groovy-groovysh		groovy-testng	
groovy-jaxb	*optional*	groovy-xml	
groovy-jmx		groovy-yaml	*optional*

What is Groovy?

Groovy = Java

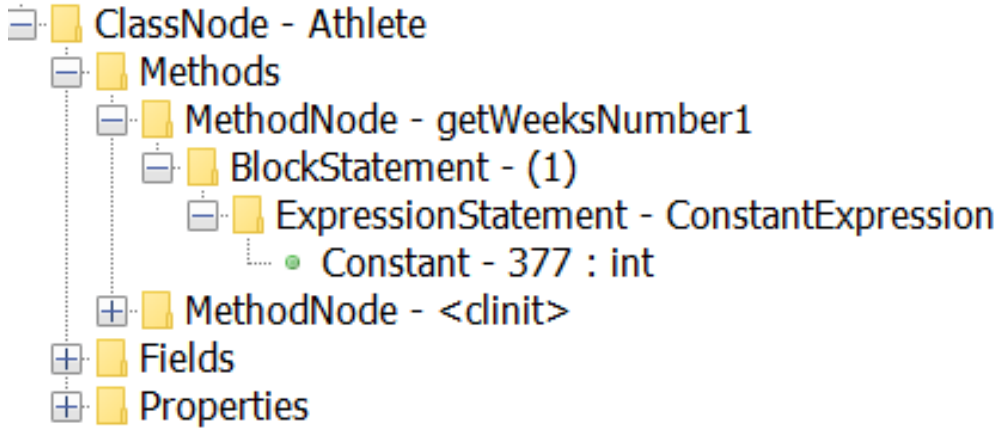
- boiler plate code
+ extensible dynamic and static natures
+ better functional programming
+ better OO programming
+ runtime metaprogramming
+ compile-time metaprogramming
  (**AST transforms**, macros, extension methods)
+ operator overloading
+ additional
+ additiona ries
+ scripts & rammar

Some additions and consistency improvements!

Groovy compilation process

- Multiple phases
- Skeletal AST

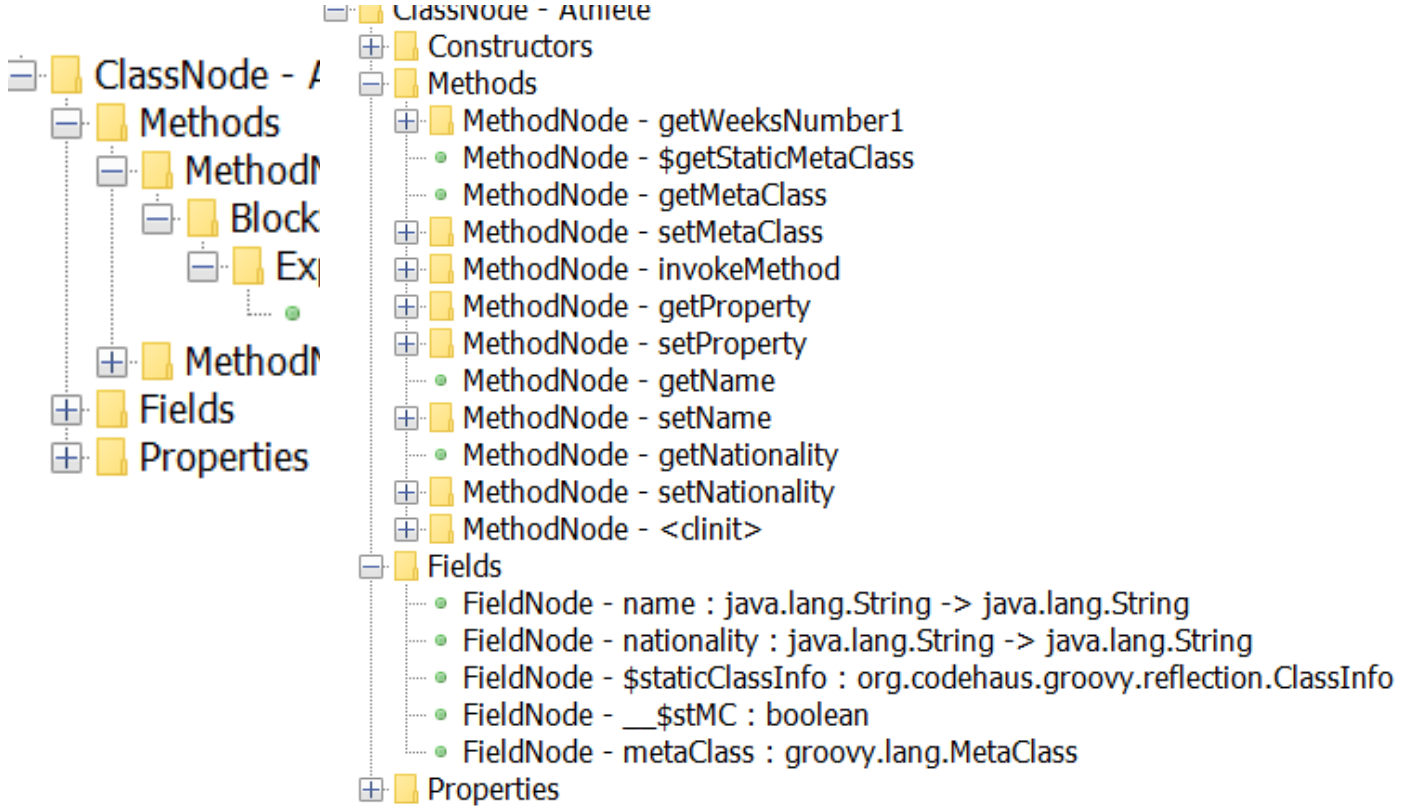
```groovy
class Athlete {
    String name, nationality
    int getWeeksNumber1() {
        377
    }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```



Groovy compilation process

- Multiple phases
- Skeletal AST => Completely resolved enriched AST
- Output bytecode

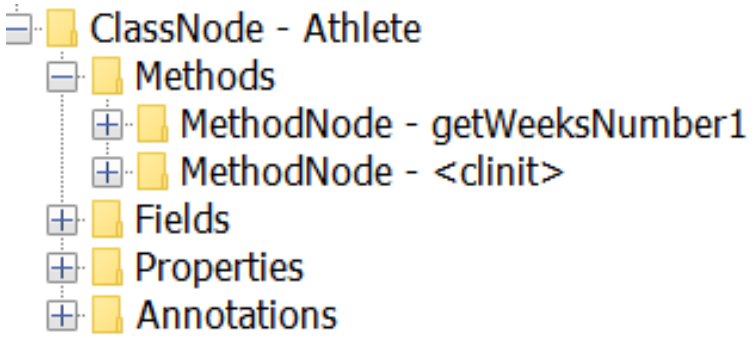
```
class Athlete {
    String name, nationality
    int getWeeksNumber1() {
        377
    }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```



Compile-time metaprogramming: AST transformations

- Global transforms
  - run for all source files
- Local transforms
  - annotations target where transform will be applied
- Manipulate the AST

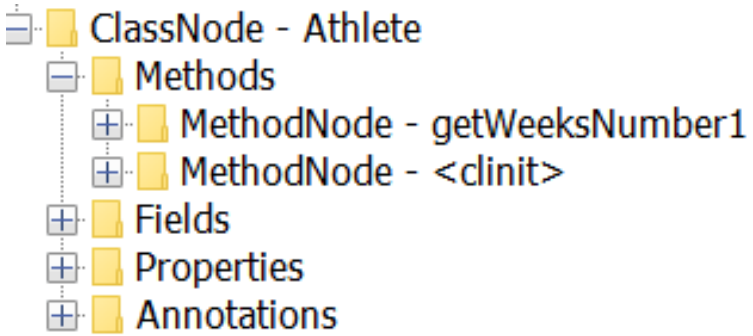
```
@ToString
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```

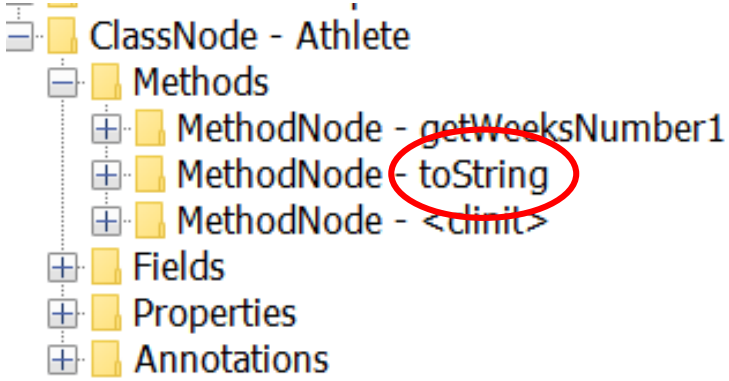


Compile-time metaprogramming: AST transformations

- Global transforms
  - run for all source files
- Local transforms
  - annotations target where transform will be applied
- Manipulate the AST

```
@ToString
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```

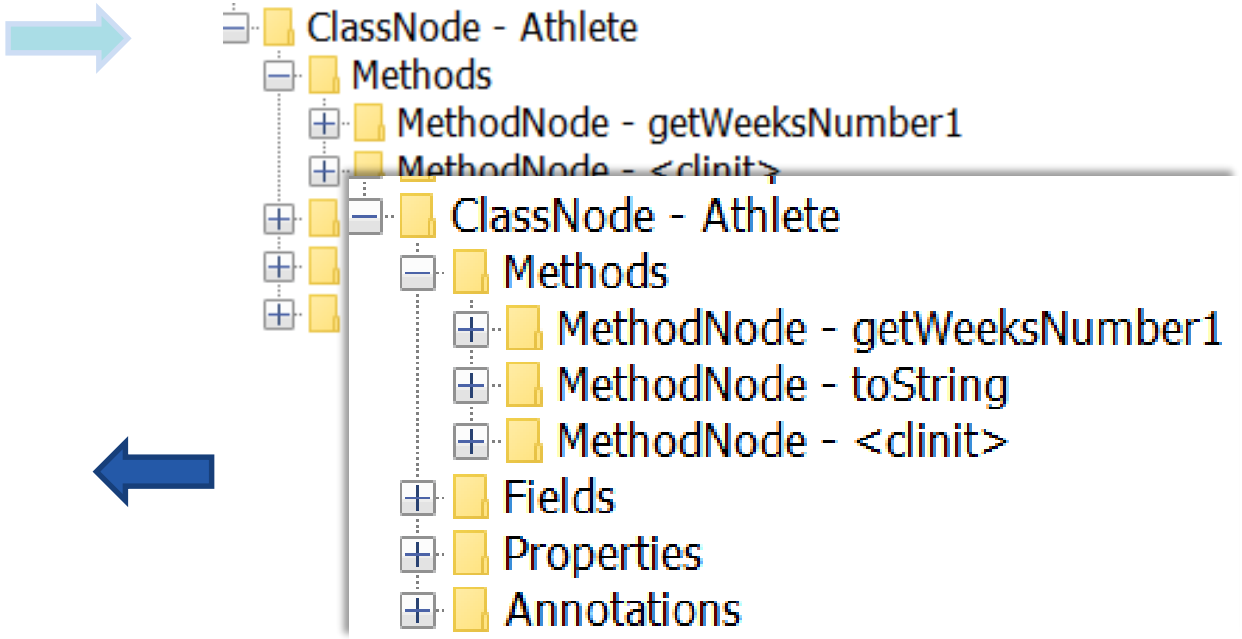




Compile-time metaprogramming: AST transformations

```
@ToString
class Athlete {
    Stri
    int g
}

new At
      n
```

```
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
    String toString() {
        def sb = new StringBuilder()
        sb << 'Athlete('
        sb << name
        sb << ', '
        sb << nationality
        sb << ')'
        return sb.toString()
    }
}
```



AST Transformations – Groovy 2.4, Groovy 2.5, Groovy 3.0

@ASTTest
@AutoClone
@AutoExternalize
@BaseScript
@Bindable
@Builder
@Canonical
@Category
@CompileDynamic
@CompileStatic
@ConditionalInterrupt
@Delegate
@EqualsAndHashCode
@ExternalizeMethods
@ExternalizeVerifier
@Field

@Grab
- @GrabConfig
- @GrabResolver
- @GrabExclude

@Grapes
@Immutable
@IndexedProperty
@InheritConstructors
@Lazy

Logging:
- @Commons
- @Log
- @Log4j
- @Log4j2
- @Slf4j

@ListenerList
@~~Mixin~~

@Newify
@NotYetImplemented
@PackageScope
@Singleton
@Sortable
@SourceURI
@Synchronized
@TailRecursive
@ThreadInterrupt
@TimedInterrupt
@ToString
@Trait *
@TupleConstructor
@TypeChecked
@Vetoable
@WithReadLock
@WithWriteLock

@AutoFinal
@AutoImplement
@ImmutableBase
@ImmutableOptions
@MapConstructor
@NamedDelegate
@NamedParam
@NamedParams
@NamedVariant
@PropertyOptions
@VisibilityOptions

@GroovyDoc *
@NullCheck

* Not normally used directly

AST Transformations – Groovy 2.4, Groovy 2.5, Groovy 3.0

@ASTTest
@AutoClone
@AutoExternalize
@BaseScript
@Bindable
@Builder
@Canonical
@Category
@CompileDynamic
@CompileStatic
@ConditionalInterrupt
@Delegate
@EqualsAndHashCode
@ExternalizeMethods
@ExternalizeVerifier
@Field

@Grab
- @GrabConfig
- @GrabResolver
- @GrabExclude

@Grapes
@Immutable
@IndexedProperty
@InheritConstructors
@Lazy
Logging:
- @Commons
- @Log
- @Log4j
- @Log4j2
- @Slf4j

@ListenerList
@~~Mixin~~

@Newify
@NotYetImplemented
@PackageScope
@Singleton
@Sortable
@SourceURI
@Synchronized
@TailRecursive
@ThreadInterrupt
@TimedInterrupt
@ToString
@Trait
@TupleConstructor
@TypeChecked
@Vetoable
@WithReadLock
@WithWriteLock

@AutoFinal
@AutoImplement
@ImmutableBase
@ImmutableOptions
@MapConstructor
@NamedDelegate
@NamedParam
@NamedParams
@NamedVariant
@PropertyOptions
@VisibilityOptions

@GroovyDoc
@NullCheck

* Improved in 2.5

AST Transformations – Groovy 2.4, Groovy 2.5

Numerous annotations have additional annotation attributes, e.g @TupleConstructor

```
String[] excludes() default {}
String[] includes() default {Undefined.STRING}
boolean includeProperties() default true
boolean includeFields() default false
boolean includeSuperProperties() default false
boolean includeSuperFields() default false
boolean callSuper() default false
boolean force() default false

boolean defaults() default true
boolean useSetters() default false
boolean allNames() default false
boolean allProperties() default false
String visibilityId() default Undefined.STRING
Class pre() default Undefined.CLASS
Class post() default Undefined.CLASS
```

AST Transformations – Groovy 2.5

Some existing annotations totally reworked:

@Canonical and @Immutable are now meta-annotations (annotation collectors)

AST Transforms: @Canonical becomes meta-annotation

@Canonical =>

@ToString, @TupleConstructor, @EqualsAndHashCode

AST Transforms: @Canonical becomes meta-annotation

@Canonical =>

@ToString, @TupleConstructor, @EqualsAndHashCode

```
@AnnotationCollector(
  value=[ToString, TupleConstructor, EqualsAndHashCode],
  mode=AnnotationCollectorMode.PREFER_EXPLICIT_MERGED)
public @interface Canonical { }
```

@Canonical

```
@Canonical(cache = true,
           useSetters = true,
           includeNames = true)
class Point {
    int x, y
}
```

@Canonical

```
@Canonical(cache = true,
           useSetters = true,
           includeNames = true)
class Point {
    int x, y
}
```

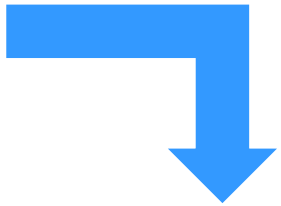
```
@ToString(cache = true, includeNames = true)
@TupleConstructor(useSetters = true)
@EqualsAndHashCode(cache = true)
class Point {
    int x, y
}
```

AST Transforms: @Immutable becomes meta-annotation

```
@Immutable
class Point {
    int x, y
}
```

AST Transforms: @Immutable becomes meta-annotation

```
@Immutable
class Point {
    int x, y
}
```



```
@ToString(includeSuperProperties = true, cache = true)
@EqualsAndHashCode(cache = true)
@ImmutableBase
@ImmutableOptions
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
@TupleConstructor(defaults = false)
@MapConstructor(noArg = true, includeSuperProperties = true, includeFields = true)
@KnownImmutable
class Point {
    int x, y
}
```

AST Transforms: @Immutable enhancements

An immutable class with one constructor making it dependency injection friendly

```
import groovy.transform.*
import groovy.transform.options.*

@ImmutableBase
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
@Canonical(defaults = false)
class Shopper {
    String first, last
    Date born
    List items
}

println new Shopper('John', 'Smith', new Date(), [])
```

AST Transforms: @TupleConstructor pre/post

```groovy
import groovy.transform.ToString
import groovy.transform.TupleConstructor

import static groovy.test.GroovyAssert.shouldFail

@ToString
@TupleConstructor(
    pre = { first = first?.toLowerCase(); assert last },
    post = { this.last = first?.toUpperCase() }
)
class Actor {
    String first, last
}

assert new Actor('Johnny', 'Depp').toString() == 'Actor(johnny, JOHNNY)'
shouldFail(AssertionError) {
    println new Actor('Johnny')
}
```

AST Transforms: @TupleConstructor enhancements

Visibility can be specified, also works with MapConstructor and NamedVariant

```
import groovy.transform.*
import static groovy.transform.options.Visibility.PRIVATE

@TupleConstructor
@VisibilityOptions(PRIVATE)
class Person {
    String name
    static makePerson(String first, String last) {
        new Person("$first $last")
    }
}

assert 'Jane Eyre' == Person.makePerson('Jane', 'Eyre').name
def publicCons = Person.constructors
assert publicCons.size() == 0
```

AST Transforms: @MapConstructor

```
import groovy.transform.MapConstructor
import groovy.transform.ToString

@ToString(includeNames = true)
@MapConstructor
class Conference {
    String name
    String city
    Date start
}

println new Conference(
    name: 'Gr8confUS', city: 'Minneapolis', start: new Date() - 2)
println Conference.constructors
```

```
Conference(name:Gr8confUS, city:Minneapolis, start:Wed Jul 26 ...)
[public Conference(java.util.Map)]
```

AST Transforms: @AutoImplement

Designed to complement Groovy's dynamic creation of "dummy" objects

```
def testEmptyIterator(Iterator it) {
    assert it.toList() == []
}

def emptyIterator = [hasNext: {false}] as Iterator

testEmptyIterator(emptyIterator)

assert emptyIterator.class.name.contains('Proxy')
```

AST Transforms: @AutoImplement

```
@AutoImplement
class MyClass extends AbstractList<String>
                implements Closeable, Iterator<String> { }
```

AST Transforms: @AutoImplement

```
class MyClass extends AbstractList<String> implements Closeable, Iterator<String> {
    String get(int param0) {
        return null
    }

    String next() {
        return null
    }

    boolean hasNext() {
        return false
    }

    void close() throws Exception {
    }

    int size() {
        return 0
    }
}
```

```
@AutoImplement
class MyClass extends AbstractList<String>
                implements Closeable, Iterator<String> { }
```

AST Transforms: @AutoImplement

```
class MyClass extends AbstractList<String> implements Closeable, Iterator<String> {
    String get(int param0) {
        return null
    }

    String next() {
        return null
    }

    boolean hasNext() {
        return false
    }

    void close() throws Exception {
    }

    int size() {
        return 0
    }
}
```

```
@AutoImplement
class MyClass extends AbstractList<String>
                implements Closeable, Iterator<String> { }
```

```
def myClass = new MyClass()

testEmptyIterator(myClass)

assert myClass instanceof MyClass
assert Modifier.isAbstract(Iterator.getDeclaredMethod('hasNext').modifiers)
assert !Modifier.isAbstract(MyClass.getDeclaredMethod('hasNext').modifiers)
```

AST Transforms: @AutoImplement

```
@AutoImplement(exception = UncheckedIOException)
class MyWriter extends Writer { }
```

```
@AutoImplement(exception = UnsupportedOperationException,
        message = 'Not supported by MyIterator')
class MyIterator implements Iterator<String> { }
```

```
@AutoImplement(code = { throw new UnsupportedOperationException(
        'Should never be called but was called on ' + new Date()) })
class EmptyIterator implements Iterator<String> {
    boolean hasNext() { false }
}
```

Built-in AST Transformations @AutoFinal

Automatically adds final modifier to constructor and method parameters

```groovy
import groovy.transform.AutoFinal

@AutoFinal
class Animal {
    private String type
    private Date lastFed

    Animal(String type) {
        this.type = type.toUpperCase()
    }

    def feed(String food) {
        lastFed = new Date()
    }
}
```

```groovy
class Zoo {
    private animals = []
    @AutoFinal
    def createZoo(String animal) {
        animals << new Animal(animal)
    }

    def feedAll(String food) {
        animals.each{ it.feed(food) }
    }
}

new Zoo()
```

Built-in AST Transformations @AutoFinal

Automatically adds final modifier to constructor and method parameters

```
import groovy.transform.AutoFinal

@AutoFinal
class Animal {
    private String type
    private Date lastFed

    Animal(final String type) {
        this.type = type.toUpperCase()
    }

    def feed(final String food) {
        lastFed = new Date()
    }
}
```

```
class Zoo {
    private animals = []
    @AutoFinal
    def createZoo(final String animal) {
        animals << new Animal(animal)
    }

    def feedAll(String food) {
        animals.each{ it.feed(food) }
    }
}

new Zoo()
```

@NamedVariant

@CompileStatic/Java friendly named args

```
import ...

def chooseBox(Color c, Integer size) {
    boolean special = RED.equals(c)
    boolean large = size > 12
    String desc = "${(large ? 'large ' : '')}" +
            "${special ? 'special' : 'standard'}"
    println "You chose a $desc box"
}

chooseBox(RED, 20) //You chose a large special box
chooseBox(BLUE, 10) //You chose a standard box
```

@NamedVariant

@CompileStatic/Java friendly named args

```
import ...

def chooseBox(Map map) {
    boolean special = RED.equals(map.color)
    boolean large = (Integer) map.size > 12
    String desc = "${(large ? 'large ' : '')}" +
            "${special ? 'special' : 'standard'}"
    println "You chose a $desc box"
}

chooseBox(color: RED, size: 20)
chooseBox(size: 10, color: BLUE)
```

@NamedVariant

```
import ...

@TypeChecked
def chooseBox(Map map) {
    boolean special = RED.equals(map.colour) // Silent fail!
    boolean large = (Integer) map.size > 12
    String desc = "${(large ? 'large ' : '')}" +
            "${special ? 'special' : 'standard'}"
    println "You chose a $desc box"
}

@TypeChecked
def method() {
    chooseBox(color: "red", size: 20)
    chooseBox(colour: BLUE, size: '10') // GroovyCastException at runtime!
}

method()
```

If we have parameters of different types the best we can use is: `Map<String, Object>` which doesn't give much information to the type checker. Impacts method writer and user.

@NamedVariant

```
import ...

@TypeChecked
@NamedVariant
def chooseBox(Color color, Integer size) {
    boolean special = RED.equals(color)
    boolean large = size > 12
    String desc = "${(large ? 'large ' : '')}" +
            "${special ? 'special' : 'standard'}"
    println "You chose a $desc box"
}

@TypeChecked
def method() {
    chooseBox(color: "red", size: 20)
    chooseBox(colour: BLUE, size: '10')
}

method()
```

@NamedVariant

```groovy
import ...

@TypeChecked
@NamedVariant
def chooseBox(Color color, Integer size) {
    boolean special = RED.equals(color)
    boolean large = size > 12
    String desc = "${(large ? 'large ' : '')}" +
            "${special ? 'special' : 'standard'}"
    println "You chose a $desc box"
}

@TypeChecked
def method() {
    chooseBox(color: "red", size: 20)
    chooseBox(colour: BLUE, size: '10')
}

method()
```

```groovy
def chooseBox(@NamedParams([
        @NamedParam(value = 'color', type = Color),
        @NamedParam(value = 'size', type = Integer)])
        Map __namedArgs) {
    chooseBox(__namedArgs.color, __namedArgs.size)
}
```

@NamedVariant

```
import ...

@TypeChecked
@NamedVariant
def chooseBox(Color color, Integer size) {
    boolean special = RED.equals(color)
    boolean large = size > 12
    println "special=$special, large=$large"
}

@TypeChecked
def method() {
    chooseBox(color: "red", size: 20)
    chooseBox(colour: BLUE, size: '10')
}

method()
```

```
def chooseBox(@NamedParams([
        @NamedParam(value = 'color', type = Color),
        @NamedParam(value = 'size', type = Integer)])
        Map __namedArgs) {
    chooseBox( __namedArgs.color, __namedArgs.size)
}
```

```
[Static type checking] - parameter for named arg 'size' has type
'java.lang.String' but expected 'java.lang.Integer'
[Static type checking] - parameter for named arg 'color' has type
'java.lang.String' but expected 'java.awt.Color'
```

@NamedVariant

```
def chooseBox(@NamedParams([
        @NamedParam(value = 'color', type = Color),
        @NamedParam(value = 'size', type = Integer)])
        Map __namedArgs) {
    chooseBox( __namedArgs.color, __namedArgs.size)
}
```

```
import ...

@TypeChecked
@NamedVariant
def chooseBox(Color color, Integer size) {
    boolean special = RED.equals(color)
    boolean large = size > 12
    String desc = "${(large ? 'large ' : '')}" +
            "${special ? 'special' : 'standard'}"
    println "You chose a $desc box"
}

@TypeChecked
def method() {
    chooseBox(color: "red", size: 20)
    chooseBox(colour: BLUE, size: '10')
}

method()
```

Type checker and IDE support still in its infancy.

What is Groovy?

Groovy = Java

- boiler plate code
+ extensible dynamic and static natures
+ better functional programming
+ better OO programming
+ runtime metaprogramming
+ compile-time metaprogramming
  (AST transforms, **macros**, extension methods)
+ operator overloading
+ additional tools
+ additional productivity libraries
+ scripts & flexible language grammar

Macros

- macro, MacroClass
- AST matcher
- Macro methods (custom macros)

Without Macros

```
import org.codehaus.groovy.ast.*
import org.codehaus.groovy.ast.stmt.*
import org.codehaus.groovy.ast.expr.*

def ast = new ReturnStatement(
    new ConstructorCallExpression(
            ClassHelper.make(Date),
            ArgumentListExpression.EMPTY_ARGUMENTS
    )
)
```

```
def ast = macro {
    return new Date()
}
```

With Macros

```
import org.codehaus.groovy.ast.*
import org.codehaus.groovy.ast.stmt.*
import org.codehaus.groovy.ast.expr.*

def ast = new ReturnStatement(
    new ConstructorCallExpression(
            ClassHelper.make(Date),
            ArgumentListExpression.EMPTY_ARGUMENTS
    )
)
```

```
def ast = macro {
    return new Date()
}
```

Macros

- Variations:
  - Expressions, Statements, Classes
  - Supports variable substitution, specifying compilation phase

```
def varX = new VariableExpression('x')
def varY = new VariableExpression('y')

def pythagoras = macro {
  return Math.sqrt($v{varX} ** 2 + $v{varY} ** 2).intValue()
}
```

Macros

- Variations:
  - Expressions, Statements, Classes
  - Supports variable substitution, specifying compilation phase

```
@Statistics
class Person {
    Integer age
    String name
}

def p = new Person(age: 12,
                   name: 'john')

assert p.methodCount == 0
assert p.fieldCount  == 2
```

```
ClassNode buildTemplateClass(ClassNode reference) {
    def methodCount = constX(reference.methods.size())
    def fieldCount = constX(reference.fields.size())

    return new MacroClass() {
        class Statistics {
            java.lang.Integer getMethodCount() {
                return $v { methodCount }
            }

            java.lang.Integer getFieldCount() {
                return $v { fieldCount }
            }
        }
    }
}
```

AST Matching

- AST Matching:
  - Selective transformations, filtering, testing
  - Supports placeholders

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

    if (ASTMatcher.matches(ref, exp)) {
        return macro { 2 }
    }

    return super.transform(exp)
}
```

Macro method examples: match

```
def fact(num) {
    return match(num) {
        when String then fact(num.toInteger())
        when(0 | 1) then 1
        when 2 then 2
        orElse num * fact(num - 1)
    }
}

assert fact("5") == 120
```

See: https://github.com/touchez-du-bois/akatsuki

Macro method examples: doWithData

Spock inspired

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class MathSpec extends Specification {
    def "maximum of two numbers"(int a, int b, int c) {
        expect:
        Math.max(a, b) == c

        where:
        a | b | c
        1 | 3 | 3
        7 | 4 | 7
        0 | 0 | 0
    }
}
```

Macro method examples: doWithData

```
doWithData {
    dowith:
        assert a + b == c
    where:
        a | b || c
        1 | 2 || 3
        4 | 5 || 9
        7 | 8 || 15
}
```

See: https://github.com/touchez-du-bois/akatsuki

What is Groovy?

Groovy = Java

- boiler plate code
+ extensible dynamic and static natures
+ better functional programming
+ better OO programming
+ runtime metaprogramming
+ compile-time metaprogramming
+ operator overloading
+ additional tools **(groovydoc, groovyConsole, groovysh)**
+ additional productivity libraries
+ scripts & flexible language grammar

Junit 5 support via groovy and groovyConsole

```
class MyTest {
    @Test
    void streamSum() {
        assert Stream.of(1, 2, 3).mapToInt{ i -> i }.sum() > 5
    }

    @RepeatedTest(value=2, name = "{displayName} {currentRepetition}/{totalRepetitions}")
    void streamSumRepeated() {
        assert Stream.of(1, 2, 3).mapToInt{i -> i}.sum() == 6
    }

    private boolean isPalindrome(s) { s == s.reverse()  }

    @ParameterizedTest // requires org.junit.jupiter:junit-jupiter-params
    @ValueSource(strings = [ "racecar", "radar", "able was I ere I saw elba" ])
    void palindromes(String candidate) {
        assert isPalindrome(candidate)
    }

    @TestFactory
    def dynamicTestCollection() {[
            dynamicTest("Add test") { -> assert 1 + 1 == 2 },
            dynamicTest("Multiply Test") { -> assert 2 * 3 == 6 }
    ]}
}
```

JUnit5 launcher: passed=8, failed=0, skipped=0, time=246ms

Includes JUnit 5 native annotations and JUnit frameworks

```groovy
@Grab('net.jqwik:jqwik:1.1.3')
import net.jqwik.api.*

class StringConcatenationTests {
    @Property
    void 'size of concatenated string is sum of size of each'(
            @ForAll String s1, @ForAll String s2
    ) {
        String conc = s1 + s2
        assert conc.size() == s1.size() + s2.size()
    }
}
```

2.5.8 and later
3.0.0-beta-2 and later

```groovy
import spock.lang.*
class Spock2Spec extends Specification {
    def "my test"() {
        expect:
        true
    }
}
```

:grab in groovysh

```
Groovy Shell (3.0.0-SNAPSHOT, JVM: 1.8.0_161)
Type ':help' or ':h' for help.
---------------------------------------------------------------
groovy:000> :grab 'com.google.guava:guava:24.1-jre'
groovy:000> import com.google.common.collect.ImmutableBiMap
===> com.google.common.collect.ImmutableBiMap
groovy:000> m = ImmutableBiMap.of('foo', 'bar')
===> [foo:bar]
groovy:000> m.inverse()
===> [bar:foo]
groovy:000>
```

groovyConsole improvements

- Horizontal layout
- Log to file
- Context menus
- JUnit 5
- Potential loop mode?
- Additional context menus?
- Cleaner config?

What is Groovy?

Groovy = Java

- boiler plate code
+ extensible dynamic and static natures
+ better functional programming
+ better OO programming
+ runtime metaprogramming **(extension methods)**
+ compile-time metaprogramming **(extension methods)**
+ operator overloading
+ additional tools
+ additional productivity libraries
+ scripts & flexible language grammar

Nearly 300 additional DGM methods

Examples:

```
assert ('a'..'h').chop(2, 4) == [['a', 'b'], ['c', 'd', 'e', 'f']]

assert 'xyzzy'.replace(xy:'he', z:'l', y: 'o') == 'hello'

def map = [ant:1, bee:2, cat:3]
map.removeAll { k,v -> k == 'bee' }
assert map == [ant:1, cat:3]

assert [1, 2, 3, 4].tails() == [[1, 2, 3, 4], [2, 3, 4], [3, 4], [4], []]

assert 'abcd'.startsWithAny('ab', 'AB')

assert 'abc123'.md5() == 'e99a18c428cb38d5f260853678922e03'
```

With vs Tap

```groovy
@TupleConstructor(includes='first, last')
class Person {
    String first, last
    boolean known = false
}

String greet(String name) {
    "Hello $name"
}

assert 'Hello John Smith' ==
        greet(new Person('John', 'Smith').with{ "$first $last" })

String welcome(Person p) {
    "Hello ${p.known ? 'Friend' : 'Stranger'}"
}

assert 'Hello Stranger' == welcome(new Person('John', 'Smith'))
assert 'Hello Friend' == welcome(new Person('Betty', 'Boo').tap{ known = true })
```

With vs Tap

```
@TupleConstructor(includes='first, last')
class Person {
    String first, last
    boolean known = false
}

String greet(String name) {
    "Hello $name"
}

assert 'Hello John Smith' ==
        greet(new Person('John', 'Smith').with{ "$first $last" })

String welcome(Person p) {
    "Hello ${p.known ? 'Friend' : 'Stranger'}"
}

assert 'Hello Stranger' == welcome(new Person('John', 'Smith'))
assert 'Hello Friend' == welcome(new Person('Betty', 'Boo').tap{ known = true })
```

With vs Tap

```
@TupleConstructor(includes='first, last')
class Person {
    String first, last
    boolean known = false
}

String greet(String name) {
    "Hello $name"
}

assert 'Hello John Smith' ==
        greet(new Person('John', 'Smith').with{ "$first $last" })

String welcome(Person p) {
    "Hello ${p.known ? 'Friend' : 'Stranger'}"
}

assert 'Hello Stranger' == welcome(new Person('John', 'Smith'))
assert 'Hello Friend' == welcome(new Person('Betty', 'Boo').tap{ known = true })
```

What is Groovy?

Groovy = Java
- boiler plate code
+ extensible dynamic and static natures
+ better functional programming
+ better OO programming
+ runtime metaprogramming
+ compile-time metaprogramming
+ operator overloading
+ additional tools
+ additional productivity libraries
  (Regex, XML, SQL, **JAXB**, **JSON**, YAML, **CLI**, builders)
+ scripts & flexible language grammar

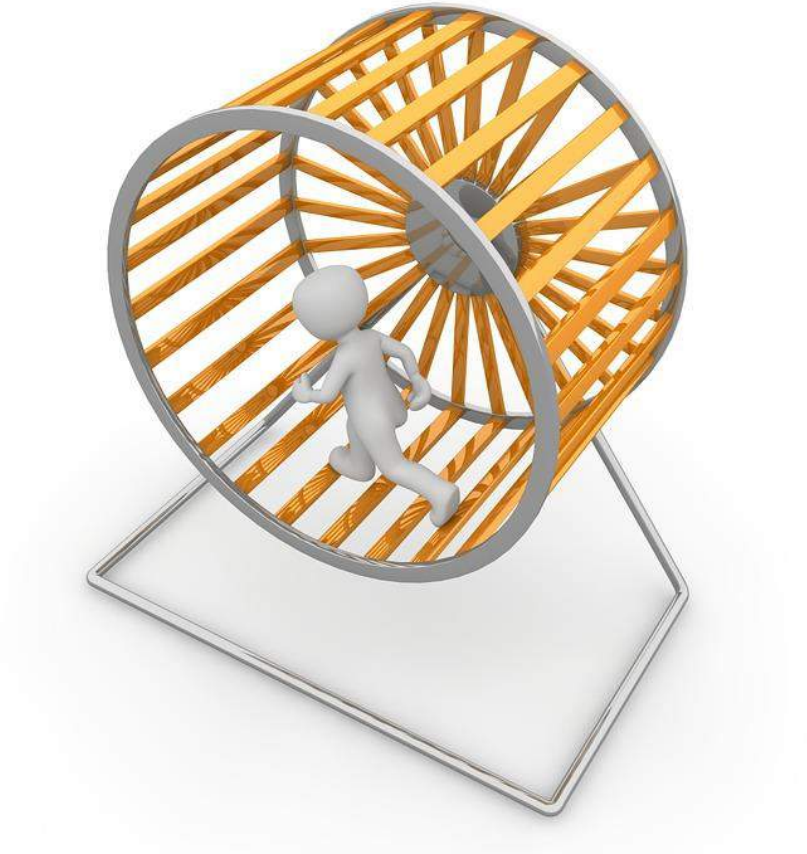

3.0

Groovy 3.0 Themes

- **Parrot parser**
  - **Improved copy/paste with Java**
  - **New syntax/operators**
- JDK8 minimum and better JDK 9/10 JPMS support
- Additional documentation options

Groovy 3.0 Themes

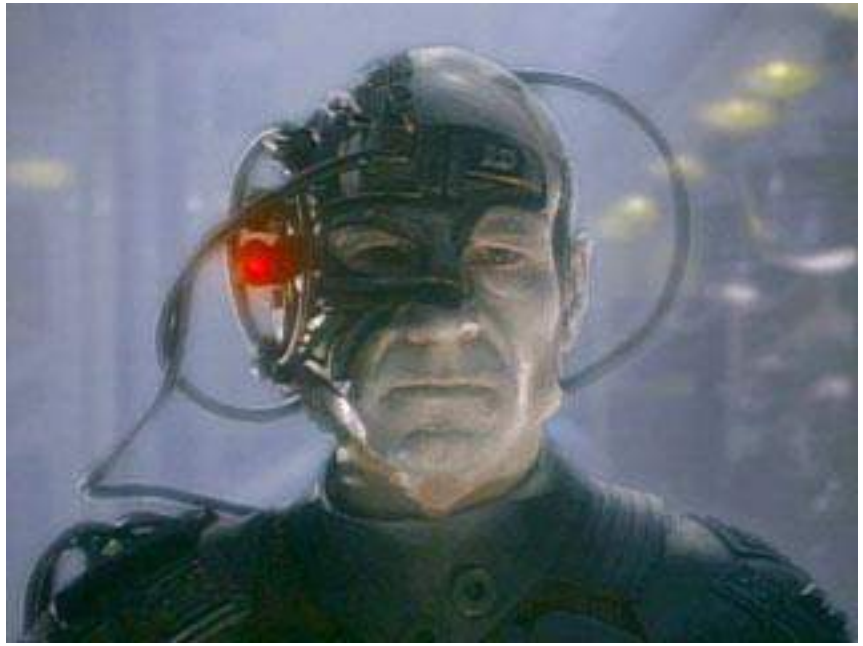
- **Parrot parser**
  - **Improved copy/paste with Java**
  - New syntax/operators
- JDK8 minimum and better JDK 9/10 JPMS support
- Additional documentation options



Catch up with Java

Groovy 3.0 Themes

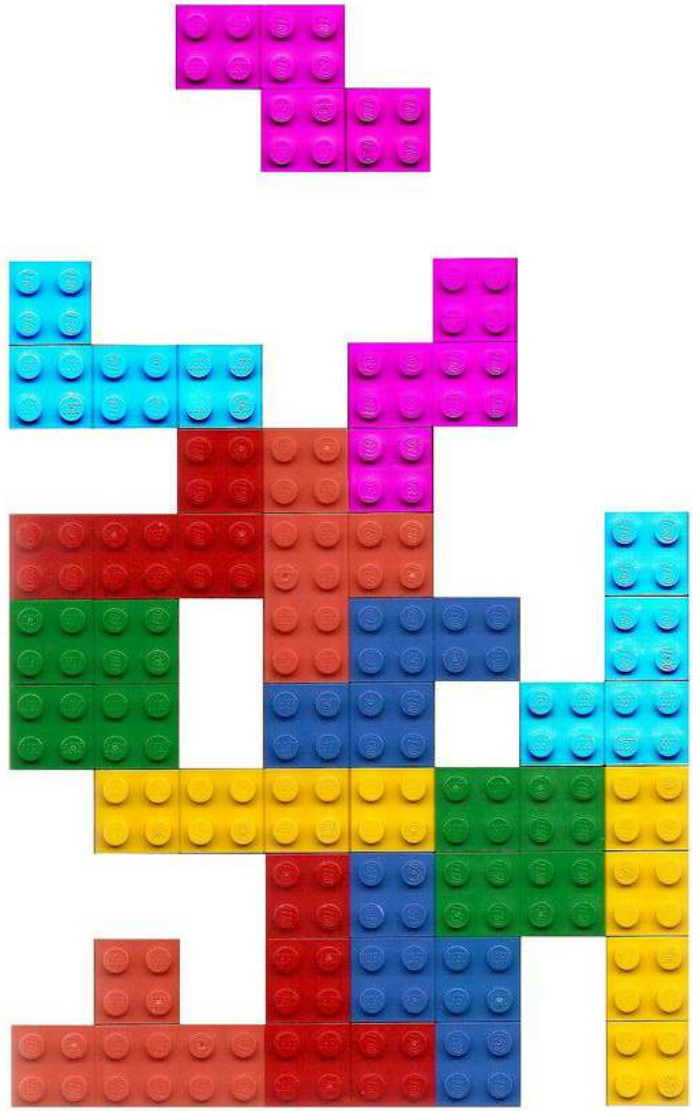
- **Parrot parser**
  - **Improved copy/paste with Java**
  - New syntax/operators
- JDK8 minimum and better JDK 9/10 JPMS support
- Additional documentation options



Java you will be assimilated: resistance is futile

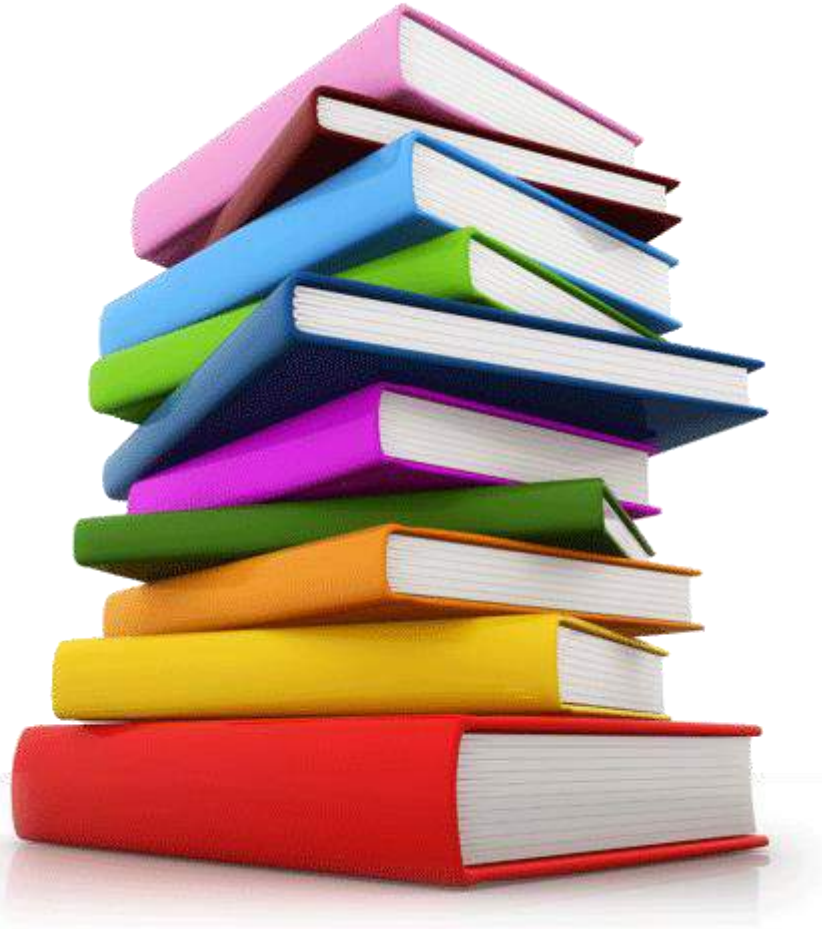
Groovy 3.0 Themes

- Parrot parser
  - Improved copy/paste with Java
  - New syntax/operators
- **JDK8 minimum and better JDK 9/10 JPMS support**
- Additional documentation options



Groovy 3.0 Themes

- Parrot parser
  - Improved copy/paste with Java
  - New syntax/operators
- JDK8 minimum and better JDK 9/10 JPMS support
- **Additional documentation options**



Parrot looping

```
// classic Java-style do..while loop
def count = 5
def fact = 1
do {
    fact *= count--
} while(count > 1)
assert fact == 120
```

Parrot looping

```
// classic for loop but now with extra commas
def facts = []
def count = 5
for (int fact = 1, i = 1; i <= count; i++, fact *= i) {
    facts << fact
}
assert facts == [1, 2, 6, 24, 120]
```

Parrot looping

```
// multi-assignment
def (String x, int y) = ['foo', 42]
assert "$x $y" == 'foo 42'

// multi-assignment goes loopy
def baNums = []
for (def (String u, int v) = ['bar', 42]; v < 45; u++, v++) {
    baNums << "$u $v"
}
assert baNums == ['bar 42', 'bas 43', 'bat 44']
```

Java-style array initialization

```
def primes = new int[] {2, 3, 5, 7, 11}
assert primes.size() == 5 && primes.sum() == 28
assert primes.class.name == '[I'

def pets = new String[] {'cat', 'dog'}
assert pets.size() == 2 && pets.sum() == 'catdog'
assert pets.class.name == '[Ljava.lang.String;'

// traditional Groovy alternative still supported
String[] groovyBooks = [ 'Groovy in Action', 'Making Java Groovy' ]
assert groovyBooks.every{ it.contains('Groovy') }
```

New operators: identity

```groovy
import groovy.transform.EqualsAndHashCode

@EqualsAndHashCode
class Creature { String type }

def cat = new Creature(type: 'cat')
def copyCat = cat
def lion = new Creature(type: 'cat')

assert cat.equals(lion) // Java logical equality
assert cat == lion      // Groovy shorthand operator

assert cat.is(copyCat)  // Groovy identity
assert cat === copyCat  // operator shorthand
assert cat !== lion     // negated operator shorthand
```

New operators: negated variants

```
assert 45 !instanceof Date

assert 4 !in [1, 3, 5, 7]
```

New operators: Elvis assignment

```
import groovy.transform.ToString

@ToString
class Element {
    String name
    int atomicNumber
}
def he = new Element(name: 'Helium')
he.with {
    // name = name != null ? name : 'Hydrogen'   // Java
    name = name ?: 'Hydrogen'   // existing Elvis operator
    atomicNumber ?= 2           // new Elvis assignment shorthand
}
assert he.toString() == 'Element(Helium, 2)'
```

Safe indexing

```
String[] array = ['a', 'b']
assert 'b' == array?[1]      // get using normal array index
array?[1] = 'c'              // set using normal array index
assert 'c' == array?[1]

array = null
assert null == array?[1]     // return null for all index values
array?[1] = 'c'              // quietly ignore attempt to set value
assert array == null
```

Better Java syntax support: try with resources

```
class FromResource extends ByteArrayInputStream {
    @Override
    void close() throws IOException {
        super.close()
        println "FromResource closing"
    }

    FromResource(String input) {
        super(input.toLowerCase().bytes)
    }
}

class ToResource extends ByteArrayOutputStream {
    @Override
    void close() throws IOException {
        super.close()
        println "ToResource closing"
    }
}
```

Better Java syntax support: try with resources

```
def wrestle(s) {
    try (
            FromResource from = new FromResource(s)
            ToResource to = new ToResource()
    ) {
        to << from
        return to.toString()
    }
}

assert wrestle("ARM was here!").contains('arm')
```

ToResource closing
FromResource closing

Better Java syntax support: try with resources

```
// some Groovy friendliness without explicit types
def wrestle(s) {
    try (
            from = new FromResource(s)
            to = new ToResource()
    ) {
        to << from
        return to.toString()
    }
}

assert wrestle("ARM was here!").contains('arm')
```

ToResource closing
FromResource closing

Better Java syntax support: try with resources

```
// some Groovy friendliness without explicit types
def wrestle(s) {
    try (
            from = new FromResource(s)
            to = new ToResource()
    ) {
        to << from
        return to.toString()
    }
}

assert wrestle("ARM was here!").contains('arm')
```

But remember Groovy's IO/Resource extension methods may be better

ToResource closing
FromResource closing

Better Java syntax support: try with resources

```
def wrestle(s) {
    new FromResource(s).withCloseable { from ->
        new ToResource().withCloseable { to ->
            to << from
            to.toString()
        }
    }
}
```

But remember Groovy's IO/Resource extension methods may be better

ToResource closing
FromResource closing

Better Java syntax support: try with resources Java 9

```
def a = 1
def resource1 = new Resource(1)

try (resource1) {
    a = 2
}

assert Resource.closedResourceIds == [1]
assert 2 == a
```

Better Java syntax support: nested blocks

```
{
    def a = 1
    a++
    assert 2 == a
}
try {
    a++ // not defined at this point
} catch(MissingPropertyException ex) {
    println ex.message
}
{
    {
        // inner nesting is another scope
        def a = 'banana'
        assert a.size() == 6
    }
    def a = 1
    assert a == 1
}
```

Better Java syntax support: var (JDK10/11)

- Local variables (JDK10)
- Lambda params (JDK11)

Lambdas

```
import static java.util.stream.Collectors.toList

(1..10).forEach(e -> { println e })

assert (1..10).stream()
        .filter(e -> e % 2 == 0)
        .map(e -> e * 2)
        .collect(toList()) == [4, 8, 12, 16, 20]
```

Lambdas – all the shapes

```
// general form
def add = (int x, int y) -> { def z = y; return x + z }
assert add(3, 4) == 7

// curly braces are optional for a single expression
def sub = (int x, int y) -> x - y
assert sub(4, 3) == 1

// parameter types and
// explicit return are optional
def mult = (x, y) -> { x * y }
assert mult(3, 4) == 12
```

```
// no parentheses required for a single parameter with no type
def isEven = n -> n % 2 == 0
assert isEven(6)
assert !isEven(7)

// no arguments case
def theAnswer = () -> 42
assert theAnswer() == 42

// any statement requires braces
def checkMath = () -> { assert 1 + 1 == 2 }
checkMath()

// example showing default parameter values (no Java equivalent)
def addWithDefault = (int x, int y = 100) -> x + y
assert addWithDefault(1, 200) == 201
assert addWithDefault(1) == 101
```

Method pointers (AKA method closures) up to 2.5

- Mechanism to treat an existing method as a Closure
  - Has the full power of Closures available

	Instance method	*Static method*
Class target	*Not meaningful:* `MissingMethodException`	`def asHex = Integer.&toHexString` `assert asHex(10) == 'a'`

Instance target	`def tenPlus = 10G.&add`
	`assert tenPlus(1G) == 11G`
	`assert tenPlus(2G) == 12G`
		`// generally discouraged`
		`def asOctal = 42.&toOctalString`
		`assert asOctal(10) == '12'`

Method pointers (AKA method closures) 3.0+

- Adds Class/instance method support
  - an extra first ‘instance’ parameter is added to the method
- Allows ‘new’ to be used to reference constructor

	Instance method	*Static method*

Class target	`def file = File.&new`
	`def isHidden = File.&isHidden`
	`assert isHidden(file('.git'))`
		`def asHex = Integer.&toHexString`
		`assert asHex(10) == 'a'`

Instance target	`def tenPlus = 10G.&add`
	`assert tenPlus(1G) == 11G`
	`assert tenPlus(2G) == 12G`
		`// generally discouraged`
		`def asOctal = 42.&toOctalString`
		`assert asOctal(10) == '12'`

Method references (syntax from Java 8+) Groovy 3.0+

- In one sense an equivalent concept
  - But see implementation details later

	Instance method	*Static method*

Class target	`def file = File::new`
	`def isHidden = File::isHidden`
	`assert isHidden(file('.git'))`
		`def asHex = Integer::toHexString`
		`assert asHex(10) == 'a'`

Instance target	`def tenPlus = 10G::add`
	`assert tenPlus(1G) == 11G`
	`assert tenPlus(2G) == 12G`
		`// generally discouraged`
		`def asOctal = 42::toOctalString`
		`assert asOctal(10) == '12'`

More method ref examples: class variants with streams

```
import java.util.stream.Stream
import static java.util.stream.Collectors.toList

// class::staticMethod
assert ['1', '2', '3'] ==
        Stream.of(1, 2, 3)
                .map(String::valueOf)
                .collect(toList())

// class::instanceMethod
assert ['A', 'B', 'C'] ==
        ['a', 'b', 'c'].stream()
                .map(String::toUpperCase)
                .collect(toList())
```

More method ref examples: constructors

```
// normal constructor
def r = Random::new
assert r().nextInt(10) in 0..9

// array constructor is handy when working with various Java libraries, e.g. streams
assert [1, 2, 3].stream().toArray().class.name == '[Ljava.lang.Object;'
assert [1, 2, 3].stream().toArray(Integer[]::new).class.name == '[Ljava.lang.Integer;'

// works with multi-dimensional arrays too
def make2d = String[][]::new
def tictac = make2d(3, 3)
tictac[0] = ['X', 'O', 'X']
tictac[1] = ['X', 'X', 'O']
tictac[2] = ['O', 'X', 'O']
assert tictac*.join().join('\n') == '''
XOX
XXO
OXO
'''.trim()
```

More method ref examples: constructors (cont'd)

```groovy
// also useful for your own classes
import groovy.transform.Canonical
import java.util.stream.Collectors

@Canonical
class Animal {
    String kind
}

def a = Animal::new
assert a('lion').kind == 'lion'

def c = Animal
assert c::new('cat').kind == 'cat'

def pets = ['cat', 'dog'].stream().map(Animal::new)
def names = pets.map(Animal::toString).collect(Collectors.joining( "," ))
assert names == 'Animal(cat),Animal(dog)'
```

Method references: implementation details

- Implemented as method closures for dynamic Groovy

```
def upper = String::toUpperCase
println 'upper = ' + upper
// upper = org.codehaus.groovy.runtime.MethodClosure@7aa9e414
```

Method references: implementation details

- Implemented as method closures for dynamic Groovy

```
def upper = String::toUpperCase
println 'upper = ' + upper
// upper = org.codehaus.groovy.runtime.MethodClosure@7aa9e414
```

```
def plus = BigInteger::add
def fivePlus = plus.curry(5G).memoizeAtLeast(10)
assert fivePlus(3G) == 8G
assert fivePlus << 4G == 9G
```

Method references: implementation details

- Implemented as method closures for dynamic Groovy
- Implemented similar to Java for static Groovy

```
def upper = String::toUpperCase
println 'upper = ' + upper
// upper = org.codehaus.groovy.runtime.MethodClosure@7aa9e414
```

```
@groovy.transform.CompileStatic
def method() {
    Function<String, String> lower = String::toLowerCase
    println 'lower = ' + lower
}
// lower = MethodRefG$$Lambda$135/0x0000000801376440@5411dd90
```

Method references: implementation details

- Implemented as method closures for dynamic Groovy
- Implemented similar to Java for static Groovy

```
class MethodRefJ {
    public static void main(String[] args) {
        Function<String, String> lower = String::toLowerCase;
        System.out.println("lower = " + lower);
    }
}
// lower = MethodRefJ$$Lambda$14/0x0000000801204840@4e515669
```

```
@groovy.transform.CompileStatic
def method() {
    Function<String, String> lower = String::toLowerCase
    println 'lower = ' + lower
}
// lower = MethodRefG$$Lambda$135/0x0000000801376440@5411dd90
```

Method references: implementation details

- Implemented as method closures for dynamic Groovy
- Implemented similar to Java for static Groovy

Source | Bytecode | ASMifier

```
public method()Ljava/lang/Object;
 L0
  LINENUMBER 6 L0
  INVOKEDYNAMIC apply()Ljava/util/function/Function; [
    // handle kind 0x6 : INVOKESTATIC
    java/lang/invoke/LambdaMetafactory.metafactory(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/
    // arguments:
    (Ljava/lang/Object;)Ljava/lang/Object;,
    // handle kind 0x5 : INVOKEVIRTUAL
    java/lang/String.toLowerCase()Ljava/lang/String;,
    (Ljava/lang/String;)Ljava/lang/String;
  ]
```

```
@groovy.transfo
def method() {
    Function<String, String> lower = String::toLowerCase
    println 'lower = ' + lower
}
// lower = MethodRefG$$Lambda$135/0x0000000801376440@5411dd90
```

Method references: implementation details

- Implemented as method closures for dynamic Groovy
  - Often slower and higher overheads
  - But more concise as less type information needed
  - Full power of Closures:
    - memoize*, n/r/curry, asWriteable, de/rehydrate, left/rightShift
- Implemented similar to Java for static Groovy
  - Can require more type information when type can't be inferred
  - Often faster
  - A few edge cases currently fall back to method closures but will be fixed over time
- If you need @CompileStatic for other reasons but want dynamic behavior, just use method pointers

	dynamic	*@CompileStatic*
.& operator	*Method closure*	*Method closure*
:: operator	*Method closure*	*Native method reference*

Default methods in interfaces


```
interface Greetable {
    String target()

    default String salutation() {
        'Greetings'
    }

    default String greet() {
        "${salutation()}, ${target()}"
    }
}

class Greetee implements Greetable {
    String name
    @Override
    String target() { name }
}

def daniel = new Greetee(name: 'Daniel')
assert 'Greetings, Daniel' == "${daniel.salutation()}, ${daniel.target()}"
assert 'Greetings, Daniel' == daniel.greet()
```

Currently implemented using traits

Illegal Access Warnings


```
$ java -version
java version "12" 2019-03-19
Java(TM) SE Runtime Environment (build 12+33)
Java HotSpot(TM) 64-Bit Server VM (build 12+33, mixed mode, sharing)

$ groovy -version
Groovy Version: 3.0.0-beta-1 JVM: 12 Vendor: Oracle Corporation OS: Win

$ groovy -e "println new Date()"
Sun May 12 22:28:21 AEST 2019

$ groovysh
Groovy Shell (3.0.0-beta-1, JVM: 12)
Type ':help' or ':h' for help.
------------------------------------------------------------------------
groovy:00
Sun May 1
===> null
groovy:00

$ groovy(

$
```

GroovyConsole

File Edit View History Script Help

```
1 println new Date()
```

```
2019
```

1:19

paulk@pop-os: /tmp

```
paulk@pop-os:/tmp$ java -version
openjdk version "12.0.1" 2019-04-16
OpenJDK Runtime Environment (build 12.0.1+12)
OpenJDK 64-Bit Server VM (build 12.0.1+12, mixed mode, sharing)
paulk@pop-os:/tmp$ groovy -version
Groovy Version: 3.0.0-beta-1 JVM: 12.0.1 Vendor: Oracle Corporation OS: Linux
paulk@pop-os:/tmp$ groovy -e "println new Date()"
Sun May 12 22:34:19 AEST 2019
paulk@pop-os:/tmp$ groovyc Test.groovy
paulk@pop-os:/tmp$
```

JDK 9+ improvements: Split packages (from beta-2)

groovy:

groovy.xml.QName → groovy.namespace.QName

groovy-ant:

groovy.util (includes AntBuilder) → groovy.ant

groovy-console:

groovy.ui.ConsoleApplet ✕

groovy.inspect → groovy.console

groovy.inspect.swingui → groovy.console.ui

groovy.ui → groovy.console.ui

groovy-groovysh:

org.codehaus.groovy.tools.shell → org.codehaus.groovy.groovysh.tools

✕ **Deprecated**

→ **Copied and original deprecated**

JDK 9+ improvements (split package changes)

groovy-jmx:

groovy.util.GroovyMBean → groovy.jmx

groovy-nio:

org.codehaus.groovy.runtime.WritablePath → org.apache.groovy.nio.runtime

org.codehaus.groovy.runtime.NioGroovyMethods → org.apache.groovy.nio.extensions.NioExtensions

groovy-swing (SwingBuilder only produces new classes):

org.codehaus.groovy.binding → org.apache.groovy.swing.binding

org.codehaus.groovy.runtime → org.apache.groovy.swing.extensions

groovy.model → groovy.swing.model

groovy.inspect.swingui → org.apache.groovy.swing.table

JDK 9+ improvements (split package changes)

groovy-test:

org.codehaus.groovy.runtime.ScriptTestAdapter → org.apache.groovy.test

groovy.transform.NotYetImplemented → groovy.test

groovy.util (includes GroovyTestCase) → groovy.test

groovy.lang → groovy.test

groovy-xml:

groovy.util (includes XmlParser & XmlSlurper) → groovy.xml

org.codehaus.groovy.tools.xml.DomToGroovy → org.apache.groovy.xml.tools

- Migrate over lifetime of Groovy 3 with some caveats
- Don’t mix and match old and new versions of classes
- Deprecated classes ***removed*** in 4.0 to make jars module friendly

GroovyDoc comments as metadata

```
import org.codehaus.groovy.control.*
import static groovy.lang.groovydoc.GroovydocHolder.DOC_COMMENT

def ast = new CompilationUnit().tap {
    addSource 'myScript.groovy', '''
        /** class doco */
        class MyClass {
            /** method doco */
            def myMethod() {}
        }
    '''
    compile Phases.SEMANTIC_ANALYSIS
}.ast

def classDoc = ast.classes[0].groovydoc
assert classDoc.content.contains('class doco')
def methodDoc = ast.classes[0].methods[0].groovydoc
assert methodDoc.content.contains('method doco')
```

Requires: -Dgroovy.attach.groovydoc=true

Groovydoc comments: runtime embedding

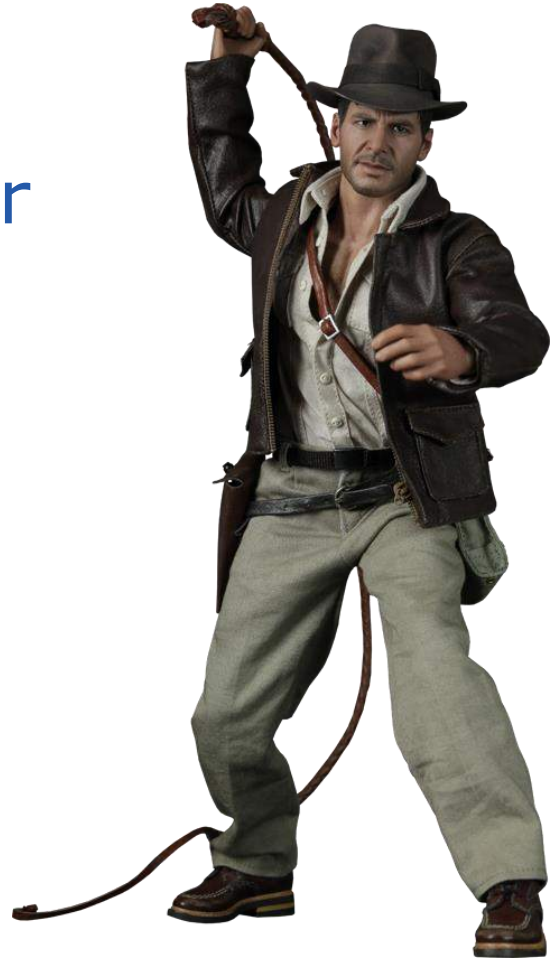
```
class Foo {
    /**@ fo fum */
    def bar() { }
}

Foo.methods.find{ it.name == 'bar' }.groovydoc.content.contains('fo fum')
```

Requires: -Dgroovy.attach.runtime.groovydoc=true


4.0

Groovy 4.0 Themes

- Consolidation:
  - Remove split package duplicates and other old deprecated classes
  - Indy-only bytecode
  - Feature interaction between xforms, traits and joint compiler
- Further JDK 9/10 module support
- Stream based XmlSlurper, GINQ…
- Type checker improvements/TLC
  - Improved built-in extensions @NonNull?
- Pattern matching, improvements for Graal VM, jdk12 switch expressions

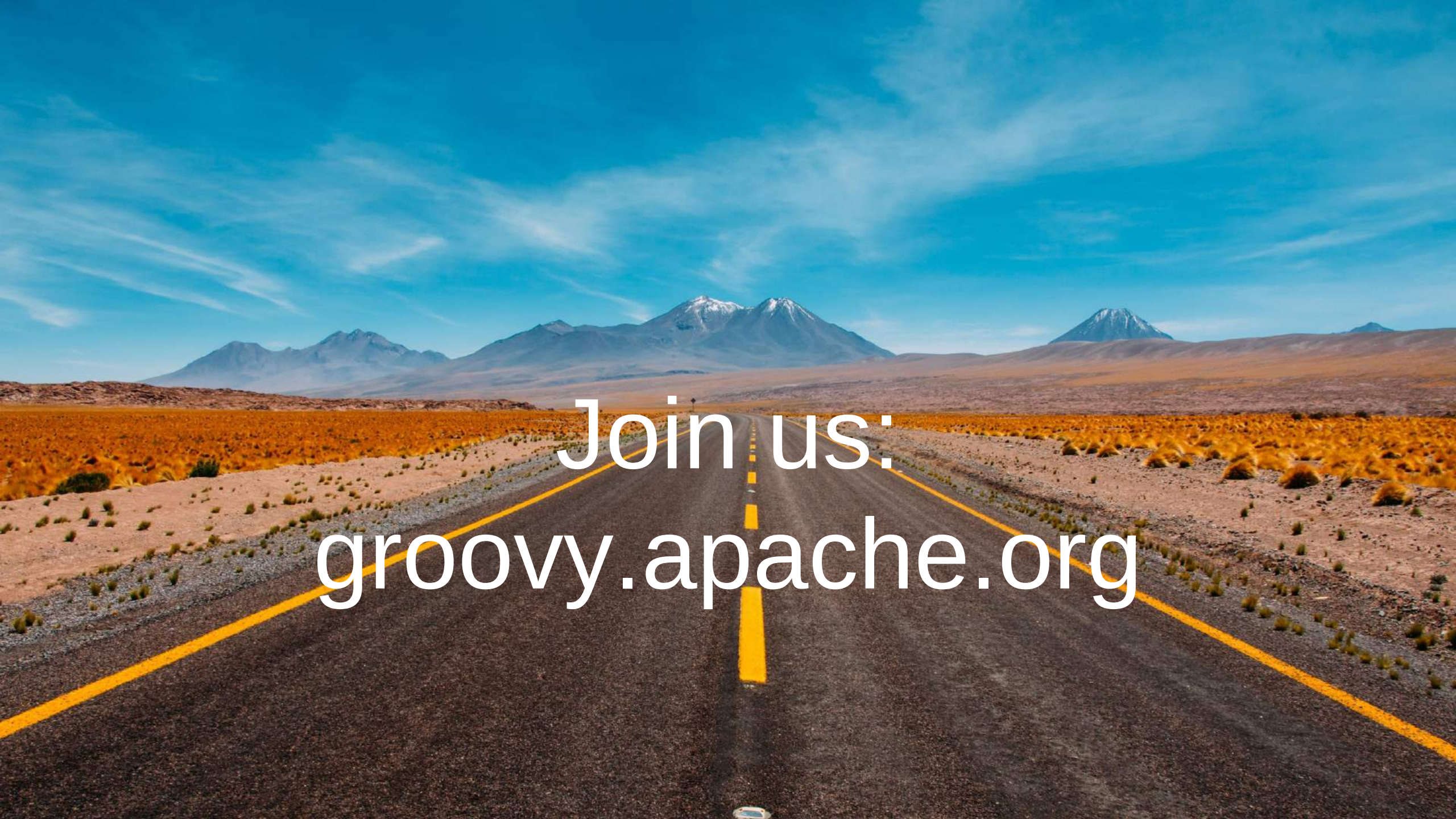


GINQ

- Experimental status

```
from p of persons
leftjoin c of cities
on p.city.name == c.name
select p.name, c.name

from p of persons
groupby p.gender
having p.gender == 'Male'
select p.gender, max(p.age)

from p of persons
orderby p.age desc
thenby p.name asc
select p.name
```


```
// C# LINQ method calls
IQueryable<Product> source = database.Products;
var results = source.Where(product => product.ReorderLevel > 20)
        .Select(product => new
                {
                    ProductName = string.Concat("@", product.ProductName),
                    UnitPrice = product.UnitPrice
                });
```


Join us:
groovy.apache.org

THANK YOU

Find me on twitter
@paulk_asert