Groovy Domain-Specific Languages

Paul King

Groovy Core Developer

ASERT

@paulk_asert

Guillaume Laforge

Groovy Project Manager

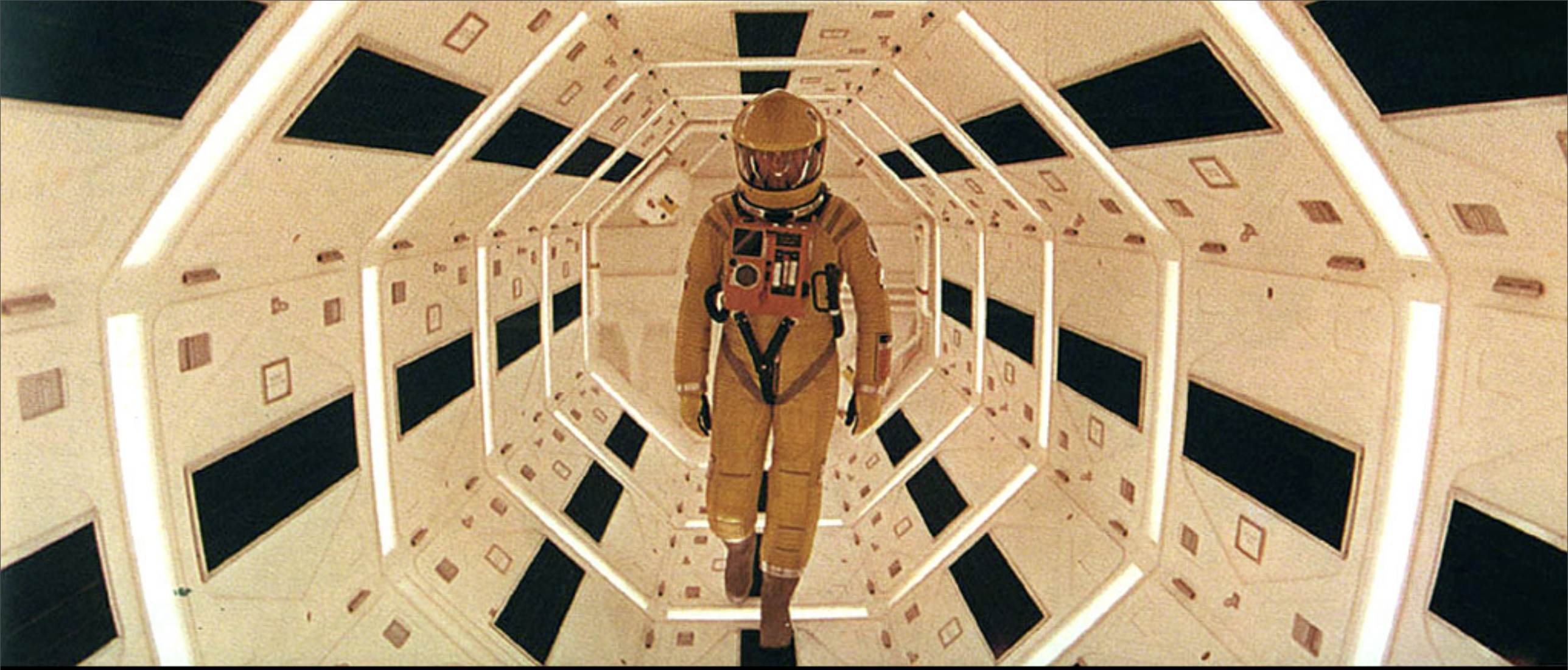
SpringSource / VMware

@glaforge

Let's get started!

Groovy provides…

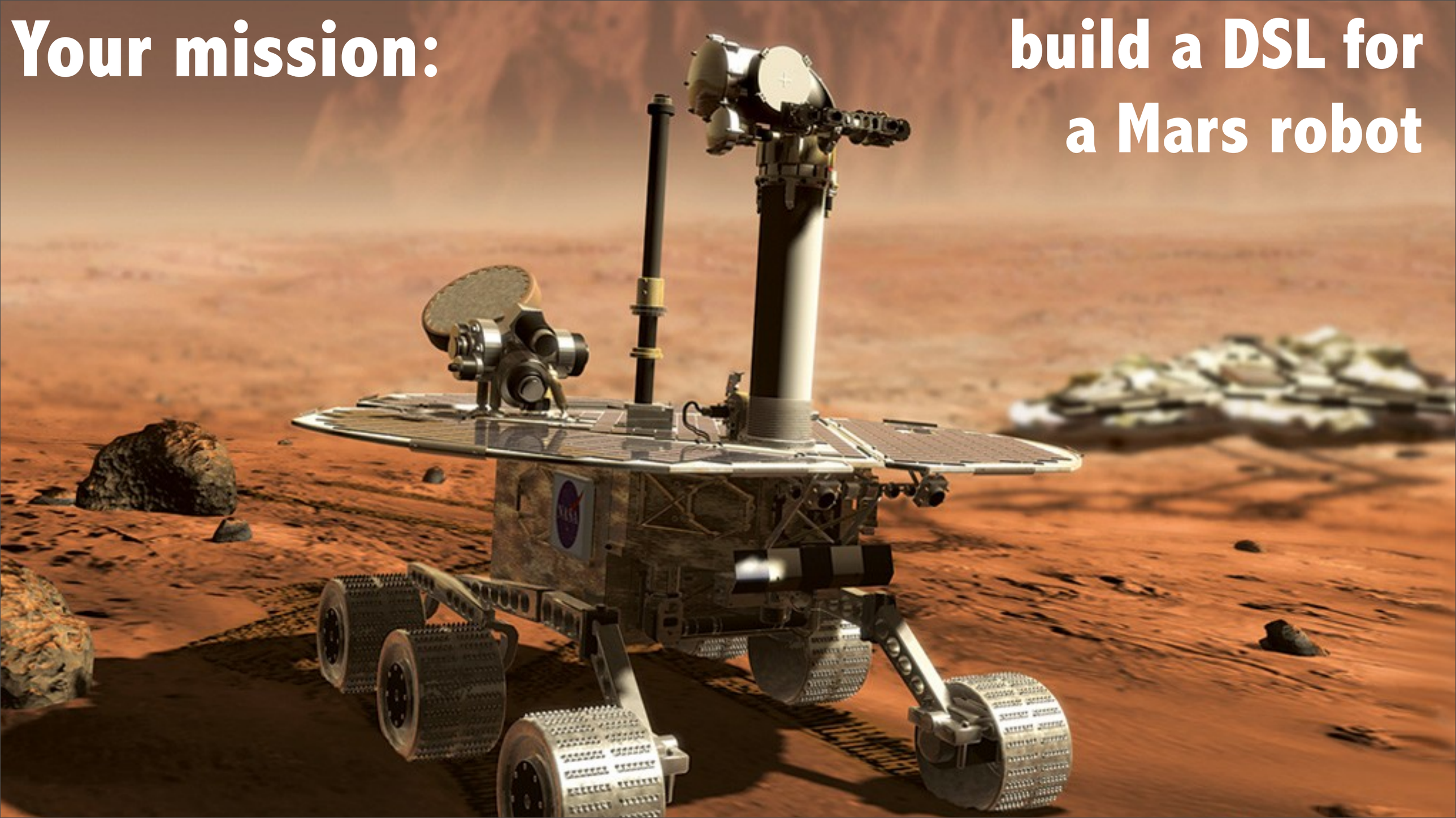
- A flexible and **malleable syntax**
  - scripts, native syntax constructs (list, map, ranges), closures, less punctuation...
- Compile-time and runtime **meta-programming**
  - metaclasses, AST transformations
  - also operator overloading
- The ability to **easily integrate** into Java / Spring apps
  - also security and safety



2001: A SPACE ODYSSEY

Your mission:

build a DSL for a Mars robot



We need a robot!

```
package mars

class Robot {}
```

It should move…

```
package mars

class Robot {
    void move() {}
}
```

..in a direction!

```
package mars

class Robot {
    void move(String dir) {}
}
```

More explicit direction

```
package mars

class Robot {
    void move(Direction dir) {}
}
```

```
package mars

enum Direction {
    left, right, forward, backward
}
```

Now how can we control it?

```
import static mars.Direction.*;
import mars.Robot;

public class Command {
    public static void main(String[] args) {
        Robot robot = new Robot();
        robot.move(left);
    }
}
```

Now how can we control it?

```
import static mars.Direction.*;
import mars.Robot;

public class Command {
    public static void main(String[] args) {
        Robot robot = new Robot();
        robot.move(left);
    }
}
```

Syntactical noise!

Now how can we control it?

```
import static mars.Direction.*;
import mars.Robot;

public class Command {
    public static void main(String[] args) {
        Robot robot = new Robot();
        robot.move(left);
    }
}
```

Syntactical noise!

Optional semicolons & parentheses / Scripts vs classes

```
import static mars.Direction.*
import mars.Robot



        def   robot = new Robot()
        robot.move left
```

Optional semicolons & parentheses / Scripts vs classes

```
import static mars.Direction.*
import mars.Robot

def    robot = new Robot()
robot.move left
```

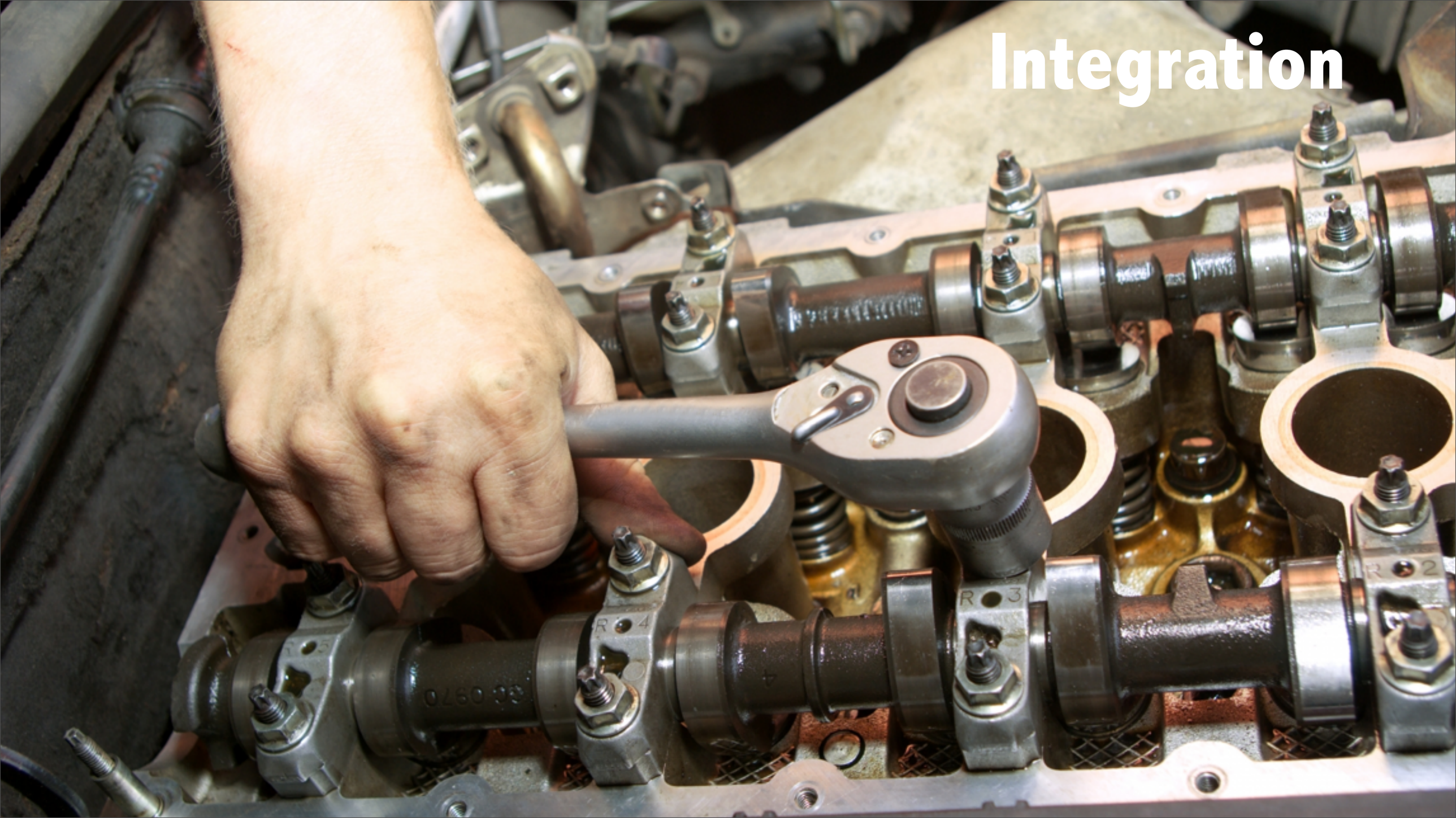
Optional typing

Optional semicolons & parentheses / Scripts vs classes

```
import static mars.Direction.*
import mars.Robot

def    robot = new Robot()
robot.move left
```

Optional typing

But I don't want to compile a script for every command!


Integration

GroovyShell to the rescue

GroovyShell to the rescue

```
def shell = new GroovyShell()
shell.evaluate(
    new File("command.groovy")
)
```

GroovyShell to the rescue

integration.groovy

```
def shell = new GroovyShell()
shell.evaluate(
    new File("command.groovy")
)
```

GroovyShell to the rescue

integration.groovy

```
def shell = new GroovyShell()
shell.evaluate(
    new File("command.groovy")
)
```

```
import static mars.Direction.*
import mars.Robot

def robot = new Robot()
robot.move left
```

GroovyShell to the rescue

integration.groovy

```groovy
def shell = new GroovyShell()
shell.evaluate(
    new File("command.groovy")
)
```

```groovy
import static mars.Direction.*
import mars.Robot

def robot = new Robot()
robot.move left
```

command.groovy

Integration mechanisms

- Different solutions available:
  - **Groovy's own mechanisms**
    - GroovyScriptEngine, GroovyShell, GroovyClassLoader, Eval
  - Java 6: javax.script.* / **JSR-223**
    - Groovy provides a JSR-223 implementation
  - **Spring**'s lang namespace

- Groovy provides the highest level of flexibility and customization, but JSR-223 is a standard...

Integration mechanisms

- Different solutions available:
  - **Groovy's own mechanisms**
    - GroovyScriptEngine, GroovyShell, GroovyClassLoader, Eval
  - Java 6: javax.script.* / **JSR-223**
    - Groovy provides a JSR-223 implementation
  - **Spring**'s lang namespace

- Groovy provides the highest level of flexibility and customization, but JSR-223 is a standard...

What's wrong with our DSL?

```
import static mars.Direction.*
import mars.Robot

def robot = new Robot()
robot.move left
```

What's wrong with our DSL?

Can't we hide those imports?

```
import static mars.Direction.*
import mars.Robot

def robot = new Robot()
robot.move left
```

What's wrong with our DSL?

Can't we hide those imports?

```
import static mars.Direction.*
import mars.Robot

def robot = new Robot()
robot.move left
```

Can't we inject the robot?

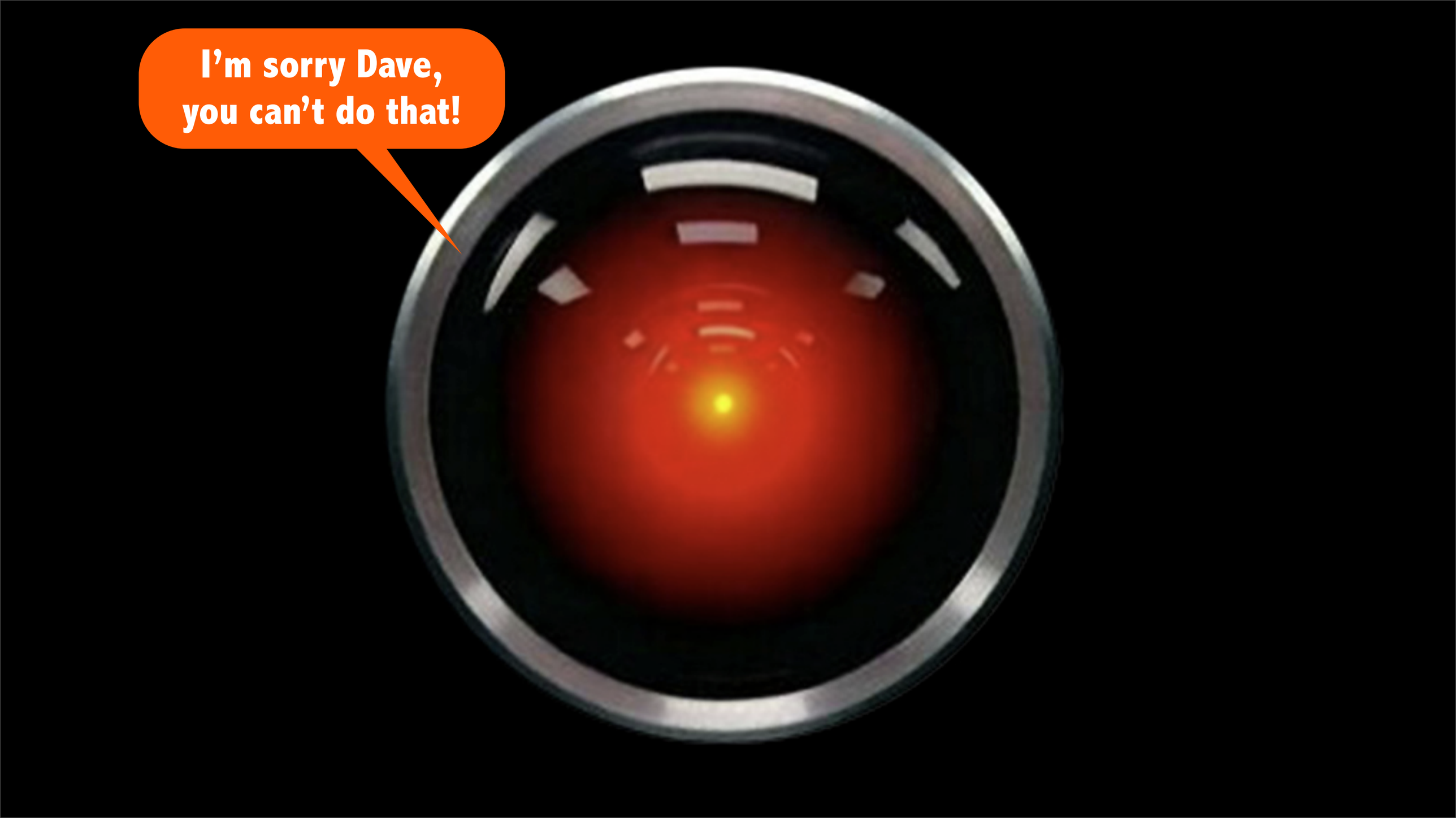
What's wrong with our DSL?

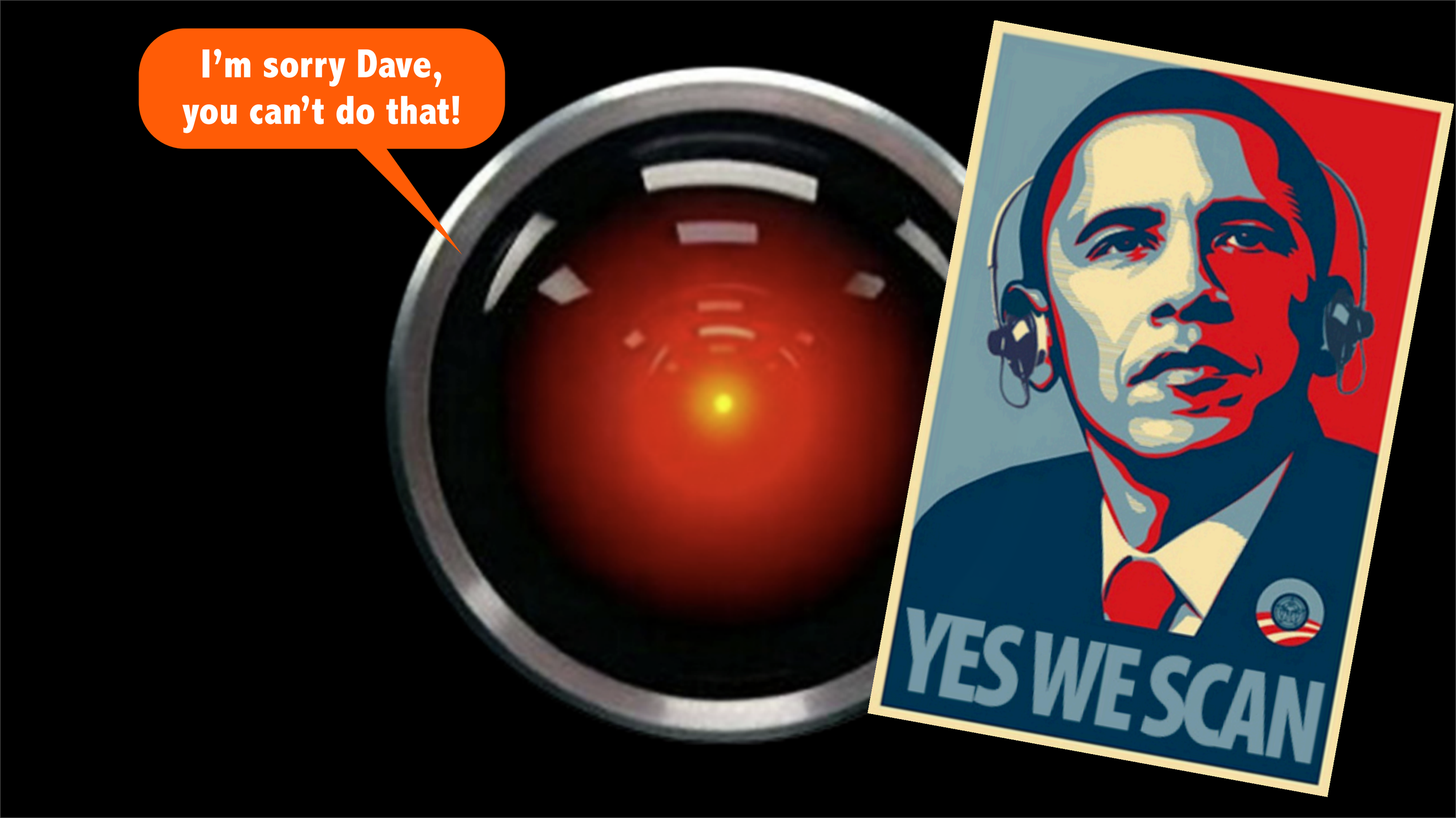
Can't we hide those imports?

```
import static mars.Direction.*
import mars.Robot

def robot = new Robot()
robot.move left
```

Do we really need to repeat 'robot'?

Can't we inject the robot?


I’m sorry Dave,
you can’t do that!


I'm sorry Dave,
you can't do that!
YES WE SCAN

What we really want is...

```
move left
```

Let's inject a robot!

- We can pass data in / out of scripts through the **Binding**
  - basically **just a map** of variable name keys and their associated values

Let's inject a robot!

- We can pass data in / out of scripts through the **Binding**
  - basically **just a map** of variable name keys and their associated values

```
def binding = new Binding([
    robot: new mars.Robot()
])
def shell = new GroovyShell(binding)
shell.evaluate(
    new File("command.groovy")
)
```

Let's inject a robot!

- We can pass data in / out of scripts through the **Binding**
  - basically **just a map** of variable name keys and their associated values

integration.groovy

```
def binding = new Binding([
    robot: new mars.Robot()
])
def shell = new GroovyShell(binding)
shell.evaluate(
    new File("command.groovy")
)
```

Better?

```
import static mars.Direction.*



robot.move left
```

Better?

Robot import removed

```
import static mars.Direction.*



robot.move left
```

Better?

Robot import removed

```
import static mars.Direction.*



robot.move left
```

Robot injected, no ‘new’ needed

How to inject the direction?

- Using the binding...

```
import mars.*

def binding = new Binding([
    robot: new Robot(),
    left:      Direction.left,
    right:     Direction.right,
    backward: Direction.backward,
    forward:  Direction.forward
])
def shell = new GroovyShell(binding)
shell.evaluate(
    new File("command.groovy")
)
```

How to inject the direction?

- Using the binding...

```
import mars.*

def binding = new Binding([
    robot: new Robot(),
    left:      Direction.left,
    right:     Direction.right,
    backward: Direction.backward,
    forward:   Direction.forward
])
def shell = new GroovyShell(binding)
shell.evaluate(
    new File("command.groovy")
)
```

Fragile in case of new directions!

How to inject the direction?

- Using the binding...

Spread map operator

```
import mars.*

def binding = new Binding([
    robot: new Robot(),
    *: Direction.values()
            .collectEntries {
                [(it.name()): it]
            }
])
def shell = new GroovyShell(binding)
shell.evaluate(
    new File("command.groovy")
)
```

How to inject the direction?

- Using string concatenation?

- Using **compiler customizers**

String concatenation? Bad idea!

```
new GroovyShell(new Binding([robot: new mars.Robot()]))
    .evaluate("import static mars.Direction.*\n" +
              "robot.move left")
```

String concatenation? Bad idea!

Cheat with string concatenation? Bad!

```
new GroovyShell(new Binding([robot: new mars.Robot()]))
    .evaluate("import static mars.Direction.*\n" +
              "robot.move left")
```

String concatenation? Bad idea!

```
new GroovyShell(new Binding([robot: new mars.Robot()]))
    .evaluate("import static mars.Direction.*\n" +
              "robot.move left")
```

String concatenation? Bad idea!

Line #1 becomes Line #2

```
new GroovyShell(new Binding([robot: new mars.Robot()]))
    .evaluate("import static mars.Direction.*\n" +
              "robot.move left")
```

String concatenation? Bad idea!

```
new GroovyShell(new Binding([robot: new mars.Robot()]))
    .evaluate("import static mars.Direction.*\n" +
              "robot.move left")
```

Compilation customizers

- Ability to apply some customization to the Groovy compilation process

- Three available customizers **Groovy 1.8**
  - **ImportCustomizer**: add transparent imports
  - **ASTTransformationCustomizer**: injects an AST transform
  - **SecureASTCustomizer**: restrict the groovy language to an allowed subset

- But you can implement your own

Imports customizer

```
def configuration = new CompilerConfiguration()

def imports = new ImportCustomizer()
imports.addStaticStar(mars.Direction.name)
configuration.addCompilationCustomizers(imports)



new GroovyShell(new Binding([robot: new mars.Robot()]),
                configuration)
    .evaluate("robot.move left")
```

AST transformation customizer

```
def configuration = new CompilerConfiguration()

def imports = new ImportCustomizer()
imports.addStaticStar(mars.Direction.name)
configuration.addCompilationCustomizers(imports,
                new ASTTransformationCustomizer(Log))

new GroovyShell(new Binding([robot: new mars.Robot()]),
                    configuration)
    .evaluate("robot.move left" + "\n"
                  "log.info ‘Robot moved’")
```

AST transformation customizer

```
def configuration = new CompilerConfiguration()

def imports = new ImportCustomizer()
imports.addStaticStar(mars.Direction.name)
configuration.addCompilationCustomizers(imports,
                new ASTTransformationCustomizer(Log))

new GroovyShell(new Binding([robot: new mars.Robot()]),
                    configuration)
    .evaluate("robot.move left" + "\n"
                  "log.info ‘Robot moved’")
```

@Log injects a logger in scripts and classes

Secure the onboard trajectory calculator

Secure AST customizer

- Let's set up our environment
  - an import customizer to import java.lang.Math.*
  - prepare a secure AST customizer

```
def imports = new ImportCustomizer()
                      .addStaticStars('java.lang.Math')
def secure = new SecureASTCustomizer()
```

Secure AST customizer

Idea: secure the rocket's onboard trajectory calculation system by allowing only math expressions to be evaluated by the calculator

- Let's set up our environment
  - an import customizer to import java.lang.Math.*
  - prepare a secure AST customizer

```
def imports = new ImportCustomizer()
                      .addStaticStars('java.lang.Math')
def secure = new SecureASTCustomizer()
```

Secure AST customizer

```
...
secure.with {
   // disallow closure creation
   closuresAllowed = false
   // disallow method definitions
   methodDefinitionAllowed = false

   // empty white list => forbid imports
   importsWhitelist = []
   staticImportsWhitelist = []
   // only allow the java.lang.Math.* static import
   staticStarImportsWhitelist = ['java.lang.Math']
...
```

Secure AST customizer

Disallow closures and methods

```
...
secure.with {
   // disallow closure creation
   closuresAllowed = false
   // disallow method definitions
   methodDefinitionAllowed = false

   // empty white list => forbid imports
   importsWhitelist = []
   staticImportsWhitelist = []
   // only allow the java.lang.Math.* static import
   staticStarImportsWhitelist = ['java.lang.Math']
...
```

Secure AST customizer

```
...
secure.with {
   // disallow closure creation
   closuresAllowed = false
   // disallow method definitions
   methodDefinitionAllowed = false

   // empty white list => forbid imports
   importsWhitelist = []
   staticImportsWhitelist = []
   // only allow the java.lang.Math.* static import
   staticStarImportsWhitelist = ['java.lang.Math']
...
```

Disallow closures and methods

Black / white list imports

Secure AST customizer

```
...
   // language tokens allowed
   tokensWhitelist = [
      PLUS, MINUS, MULTIPLY, DIVIDE, MOD, POWER, PLUS_PLUS, MINUS_MINUS,
      COMPARE_EQUAL, COMPARE_NOT_EQUAL, COMPARE_LESS_THAN, COMPARE_LESS_THAN_EQUAL,
      COMPARE_GREATER_THAN, COMPARE_GREATER_THAN_EQUAL
   ]

   // types allowed to be used (including primitive types)
   constantTypesClassesWhiteList = [
      Integer, Float, Long, Double, BigDecimal,
      Integer.TYPE, Long.TYPE, Float.TYPE, Double.TYPE
   ]

   // classes who are allowed to be receivers of method calls
   receiversClassesWhiteList = [
      Math, Integer, Float, Double, Long, BigDecimal ]
}
...
```

Secure AST customizer

You can build a subset of the Groovy syntax!

```
...
   // language tokens allowed
   tokensWhitelist = [
      PLUS, MINUS, MULTIPLY, DIVIDE, MOD, POWER, PLUS_PLUS, MINUS_MINUS,
      COMPARE_EQUAL, COMPARE_NOT_EQUAL, COMPARE_LESS_THAN, COMPARE_LESS_THAN_EQUAL,
      COMPARE_GREATER_THAN, COMPARE_GREATER_THAN_EQUAL
   ]

   // types allowed to be used (including primitive types)
   constantTypesClassesWhiteList = [
      Integer, Float, Long, Double, BigDecimal,
      Integer.TYPE, Long.TYPE, Float.TYPE, Double.TYPE
   ]

   // classes who are allowed to be receivers of method calls
   receiversClassesWhiteList = [
      Math, Integer, Float, Double, Long, BigDecimal ]
}
...
```

Secure AST customizer

You can build a subset of the Groovy syntax!

```
...
   // language tokens allowed
   tokensWhitelist = [
      PLUS, MINUS, MULTIPLY, DIVIDE, MOD, POWER, PLUS_PLUS, MINUS_MINUS,
      COMPARE_EQUAL, COMPARE_NOT_EQUAL, COMPARE_LESS_THAN, COMPARE_LESS_THAN_EQUAL,
      COMPARE_GREATER_THAN, COMPARE_GREATER_THAN_EQUAL
   ]

   // types allowed to be used (including primitive types)
   constantTypesClassesWhiteList = [
      Integer, Float, Long, Double, BigDecimal,
      Integer.TYPE, Long.TYPE, Float.TYPE, Double.TYPE
   ]

   // classes who are allowed to be receivers of method calls
   receiversClassesWhiteList = [
      Math, Integer, Float, Double, Long, BigDecimal ]
}
...
```

Black / white list usage of classes

Secure AST customizer

- Ready to evaluate our flight equations!

```
def config = new CompilerConfiguration()
config.addCompilationCustomizers(imports, secure)
def shell = new GroovyShell(config)

shell.evaluate 'cos PI/3'
```

- But the following would have failed:

```
shell.evaluate 'System.exit(0)'
```

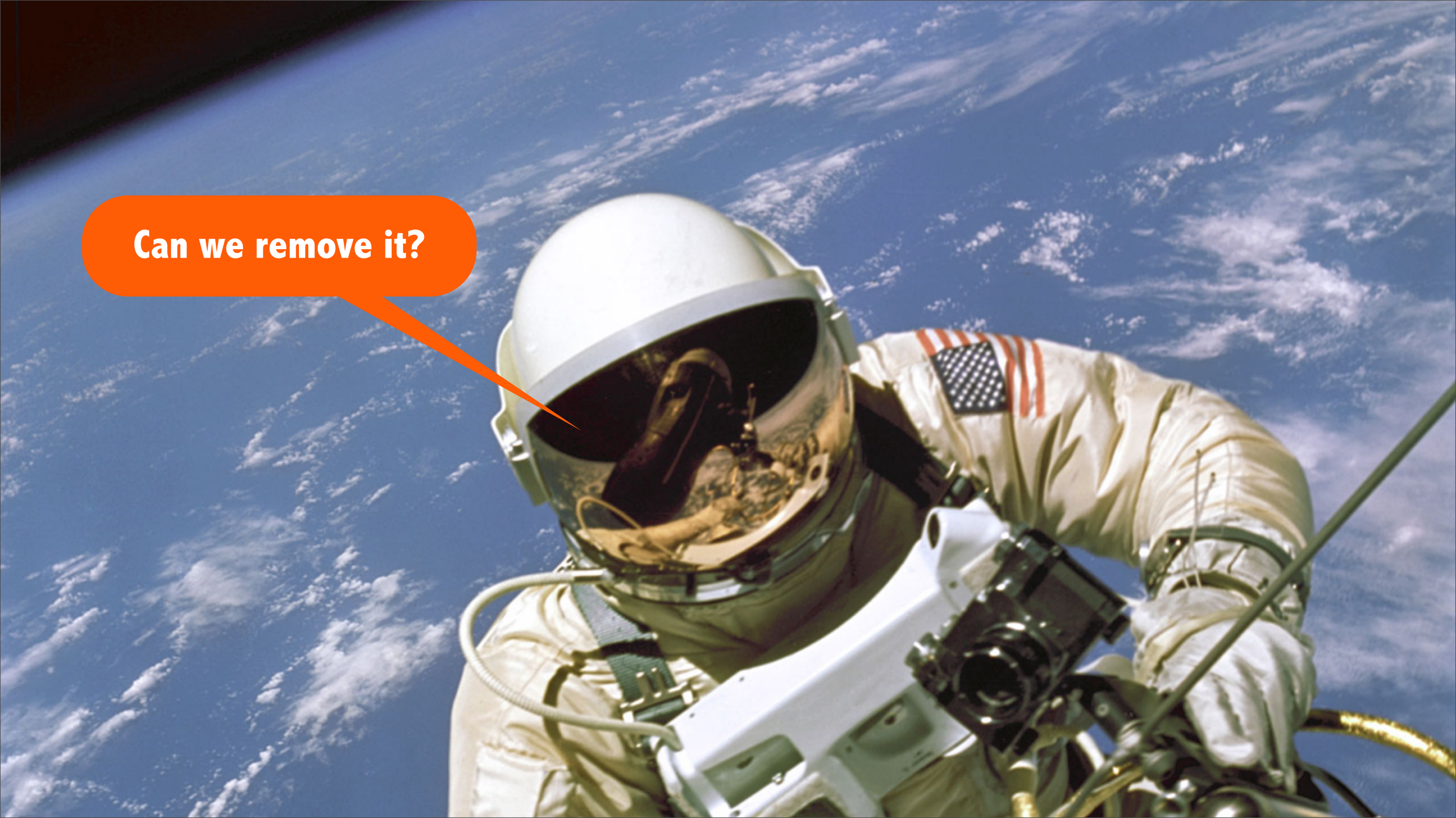
Back to our robot...

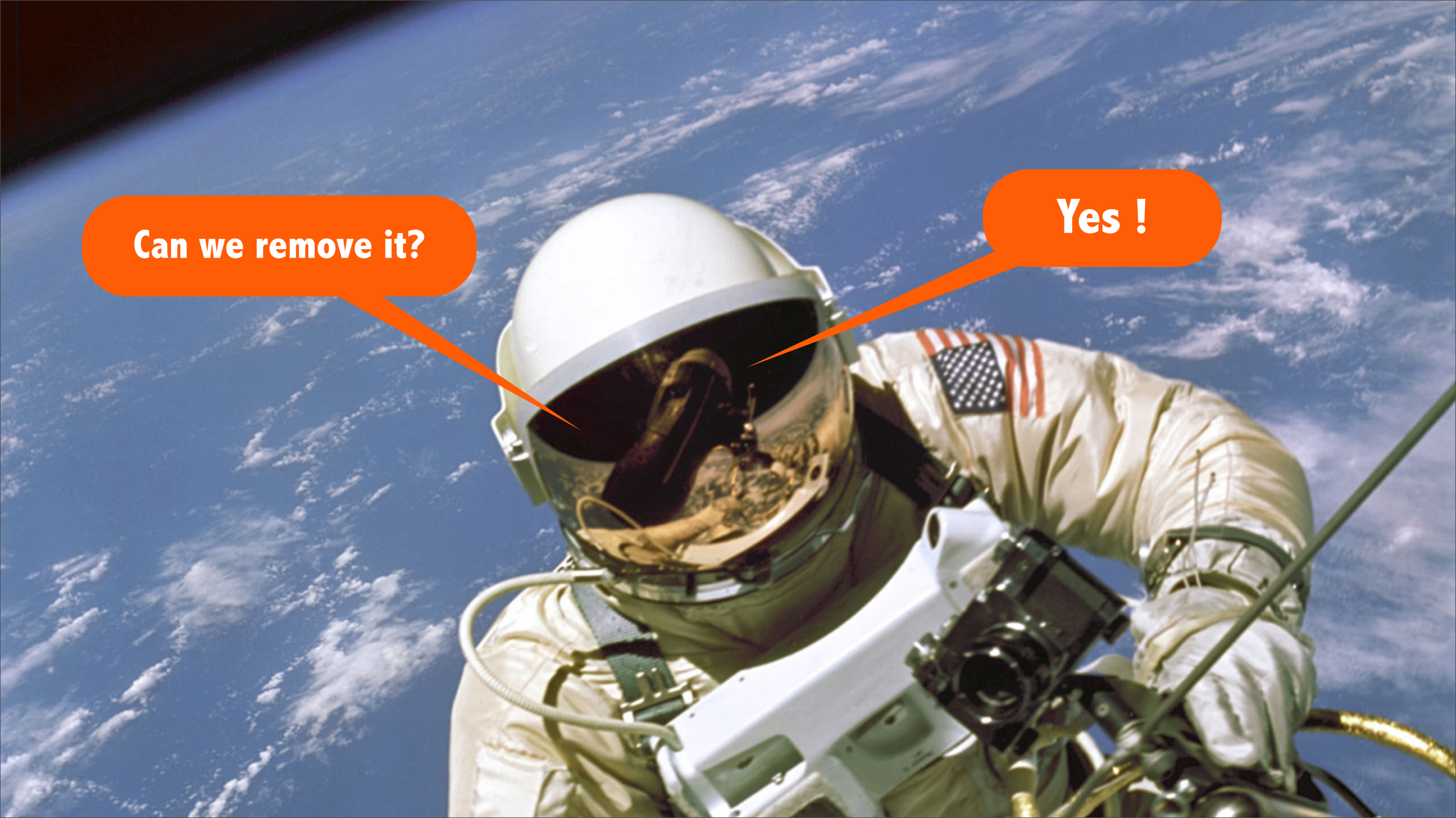
```
robot.move left
```

Back to our robot...

Still need to get rid of the robot prefix!

```
robot.move left
```


Can we remove it?


Can we remove it?
Yes !

How to get rid of the ‘robot’?

- Instead of calling the `move()` method on the robot instance, we should be able to call the `move()` method directly from within the script

- Two approaches
  - Inject a ‘move’ closure in the binding with a method pointer
  - Use a base script class with a ‘move’ method delegating to the robot

Inject a closure in the binding

```
def robot = new mars.Robot()
binding = new Binding([
    robot: robot,
    *: Direction.values()
            .collectEntries {
                [(it.name()): it]
            },

    move: robot.&move
])
```

Inject a closure in the binding

```
def robot = new mars.Robot()
binding = new Binding([
    robot: robot,
    *: Direction.values()
            .collectEntries {
                [(it.name()): it]
            },

    move: robot.&move
])
```

Method pointer (a closure) on robot's move instance method

Define a base script class

```
abstract class RobotBaseScriptClass extends Script {
    void move(Direction dir) {
        def robot = this.binding.robot
        robot.move dir
    }
}
```

Define a base script class

```
abstract class RobotBaseScriptClass extends Script {
    void move(Direction dir) {
        def robot = this.binding.robot
        robot.move dir
    }
}
```

The move() method is now at the script level

Define a base script class

```
abstract class RobotBaseScriptClass extends Script {
    void move(Direction dir) {
        def robot = this.binding.robot
        robot.move dir
    }
}
```

The move() method is now at the script level

Access the robot through the script's binding

Configure the base script class

```
def conf = new CompilerConfiguration()
conf.scriptBaseClass = RobotBaseScriptClass
```

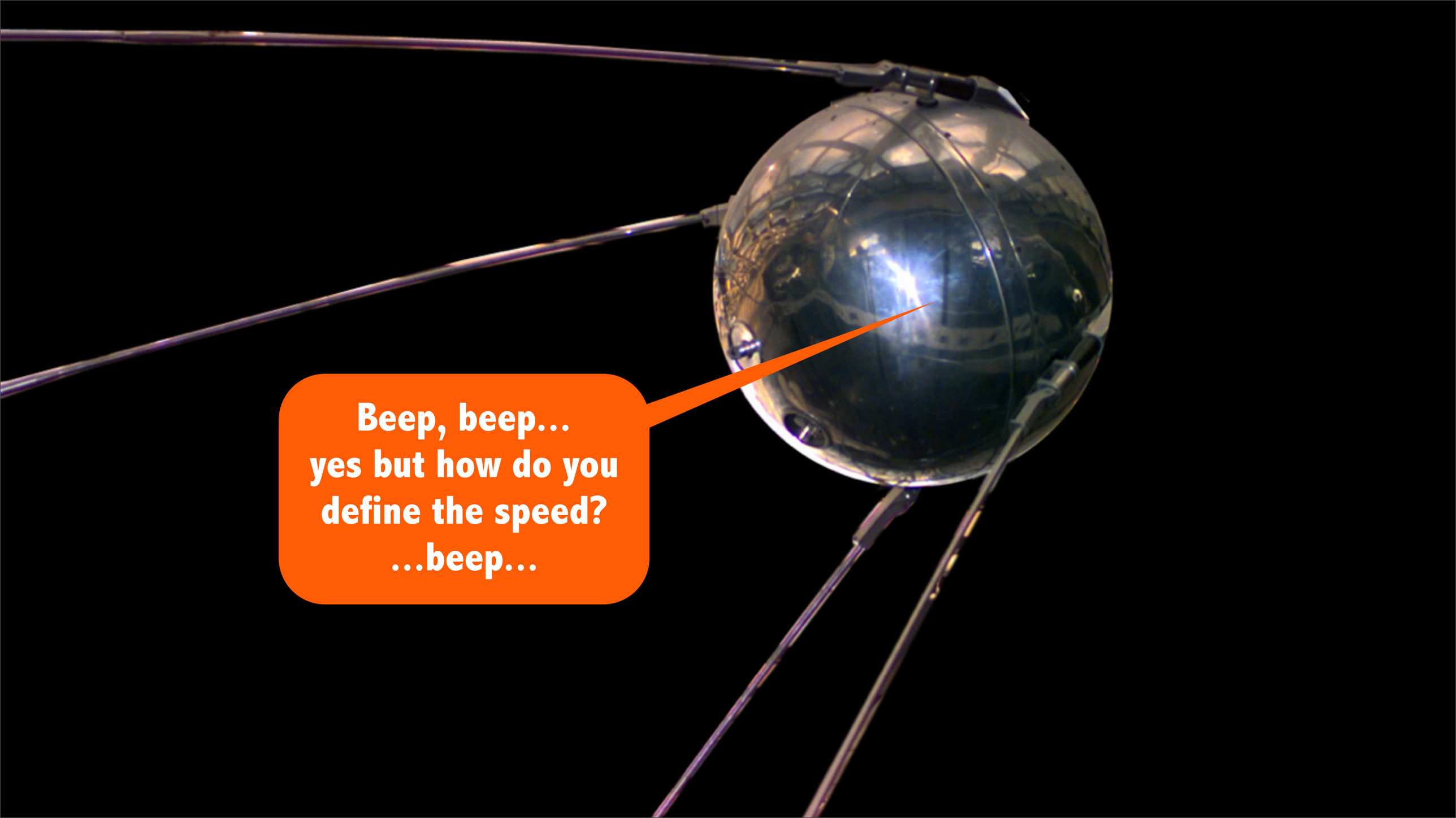
Configure the base script class

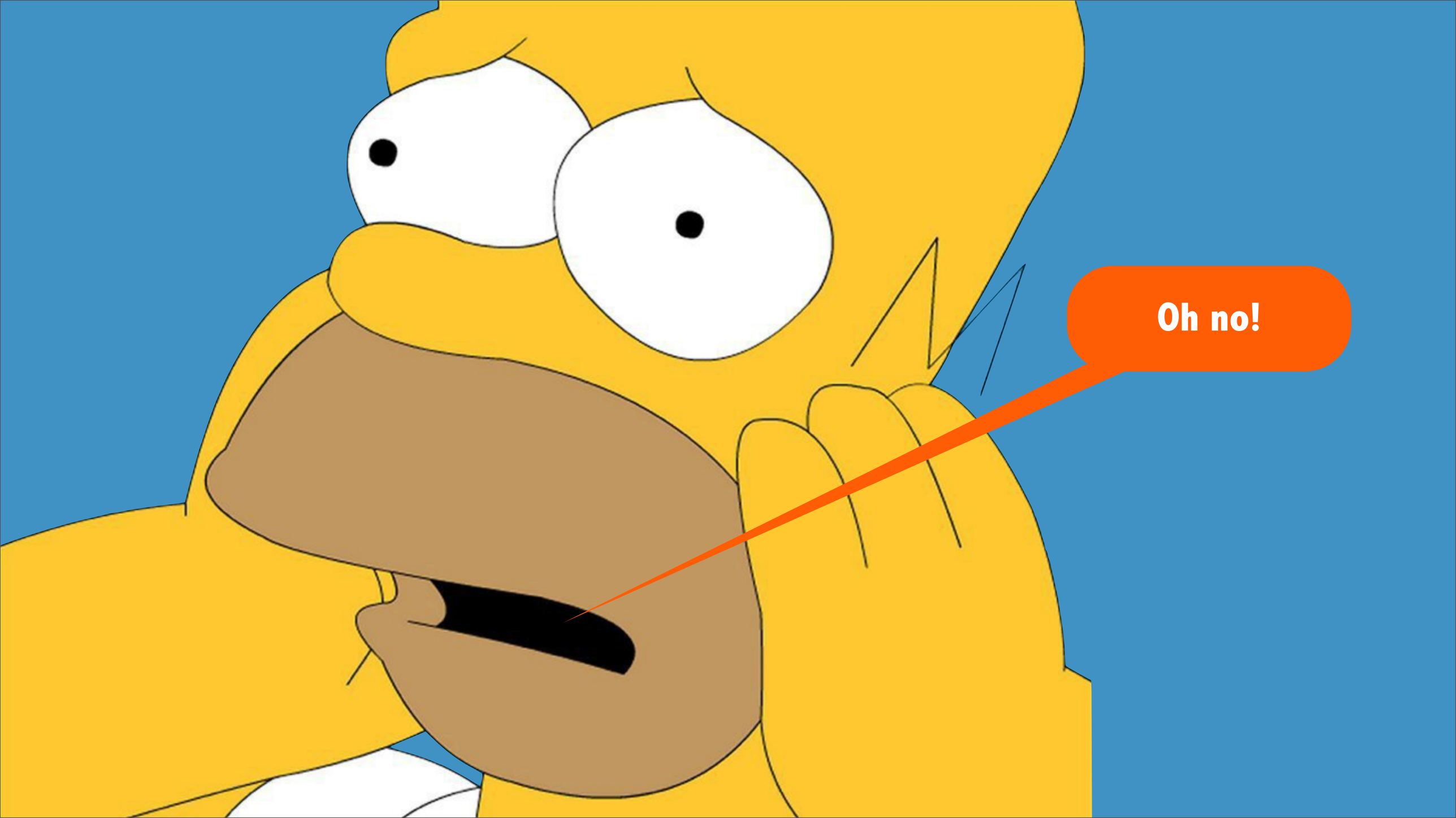
```
def conf = new CompilerConfiguration()
conf.scriptBaseClass = RobotBaseScriptClass
```

Scripts evaluated with this configuration will inherit from that class

Ready for lift off!

```
move left
```


Beep, beep…
yes but how do you
define the speed?
…beep…


Oh no!

What we could do now is...

```
move left, at: 3.km/h
```

What we could do now is...

Mix of named and normal parameters

```
move left, at: 3.km/h
```

What we could do now is...

Mix of named and normal parameters

```
move left, at: 3.km/h
```

How to support this speed notation?

Supporting the speed notation

- We need to:
  - define units of distance, time and speed
    - DistanceUnit and Distance
    - TimeUnit and Duration
    - Speed
  - have a nice notation for them by **adding properties to numbers**
  - be able to define speed thanks to **operator overloading**

Distance unit enum and distance

```
enum DistanceUnit {
    centimeter ('cm',    0.01),
    meter      ( 'm',    1   ),
    kilometer  ('km', 1000   )

    String abbreviation
    double multiplier

    DistanceUnit(String abbr, double mult) {
        this.abbreviation = abbr
        this.multiplier = mult
    }

    String toString() { abbreviation }
}
```

Distance unit enum and distance

```
enum DistanceUnit {
    centimeter ('cm',    0.01),
    meter      ( 'm',    1   ),
    kilometer  ('km', 1000   )

    String abbreviation
    double multiplier

    DistanceUnit(String abbr, double mult) {
        this.abbreviation = abbr
        this.multiplier = mult
    }

    String toString() { abbreviation }
}
```

```
@TupleConstructor
class Distance {
    double amount
    DistanceUnit unit

    String toString() {
        "$amount $unit"
    }
}
```

Time unit enum and duration

```
enum TimeUnit {
    hour      (  'h', 3600),
    minute    ('min',   60),
    second    (  's',    1)

    String abbreviation
    double multiplier

    TimeUnit(String abbr, double mult) {
        this.abbreviation = abbr
        this.multiplier = mult
    }

    String toString() { abbreviation }
}
```

Time unit enum and duration

```
enum TimeUnit {
    hour      (  'h', 3600),
    minute    ('min',   60),
    second    (  's',    1)

    String abbreviation
    double multiplier

    TimeUnit(String abbr, double mult) {
        this.abbreviation = abbr
        this.multiplier = mult
    }

    String toString() { abbreviation }
}
```

```
@TupleConstructor
class Duration {
    double amount
    TimeUnit unit

    String toString() {
        "$amount $unit"
    }
}
```

Now at (light!) speed

$$speed = \frac{distance}{duration}$$

```
@TupleConstructor
class Speed {
    Distance distance
    Duration dur

    String toString() {
        "$distance/$dur"
    }
}
```

First, we need the distance notation

- We add a dynamic property to numbers by adding a getter to them and use the property notation shortcut:

```
2.km
```

```
2.getKm()
```

Techniques to add properties to numbers

- To add dynamic methods or properties, there are several approaches at your disposal:
  - ExpandoMetaClass
  - custom MetaClass
  - Categories
  - Extension modules **Groovy 2**
- Let's have a look at the **ExpandoMetaClass**

Using ExpandoMetaClass

```
Number.metaClass.getCm = { ->
    new Distance(delegate, Unit.centimeter)
}
Number.metaClass.getM = { ->
    new Distance(delegate, Unit.meter)
}
Number.metaClass.getKm = { ->
    new Distance(delegate, Unit.kilometer)
}
```

Using ExpandoMetaClass

Add that to integration.groovy

```
Number.metaClass.getCm = { ->
    new Distance(delegate, Unit.centimeter)
}
Number.metaClass.getM = { ->
    new Distance(delegate, Unit.meter)
}
Number.metaClass.getKm = { ->
    new Distance(delegate, Unit.kilometer)
}
```

Using ExpandoMetaClass

Add that to integration.groovy

```
Number.metaClass.getCm = { ->
    new Distance(delegate, Unit.centimeter)
}
Number.metaClass.getM = { ->
    new Distance(delegate, Unit.meter)
}
Number.metaClass.getKm = { ->
    new Distance(delegate, Unit.kilometer)
}
```

'delegate' is the current number

Using ExpandoMetaClass

Add that to integration.groovy

```
Number.metaClass.getCm = { ->
    new Distance(delegate, Unit.centimeter)
}
Number.metaClass.getM = { ->
    new Distance(delegate, Unit.meter)
}
Number.metaClass.getKm = { ->
    new Distance(delegate, Unit.kilometer)
}
```

'delegate' is the current number

Usage in your DSLs

```
40.cm
3.5.m
4.km
```

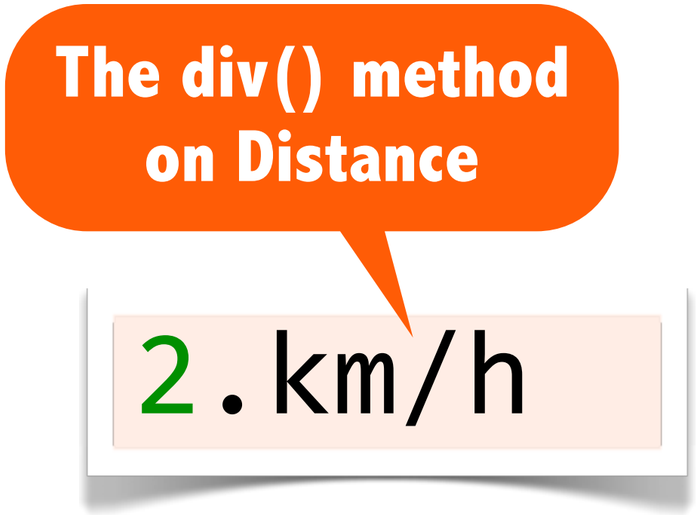
Distance okay, but speed?

- For distance, we just added a property access after the number, but we now need to divide (‘div’) by the time

```
2.km/h
```

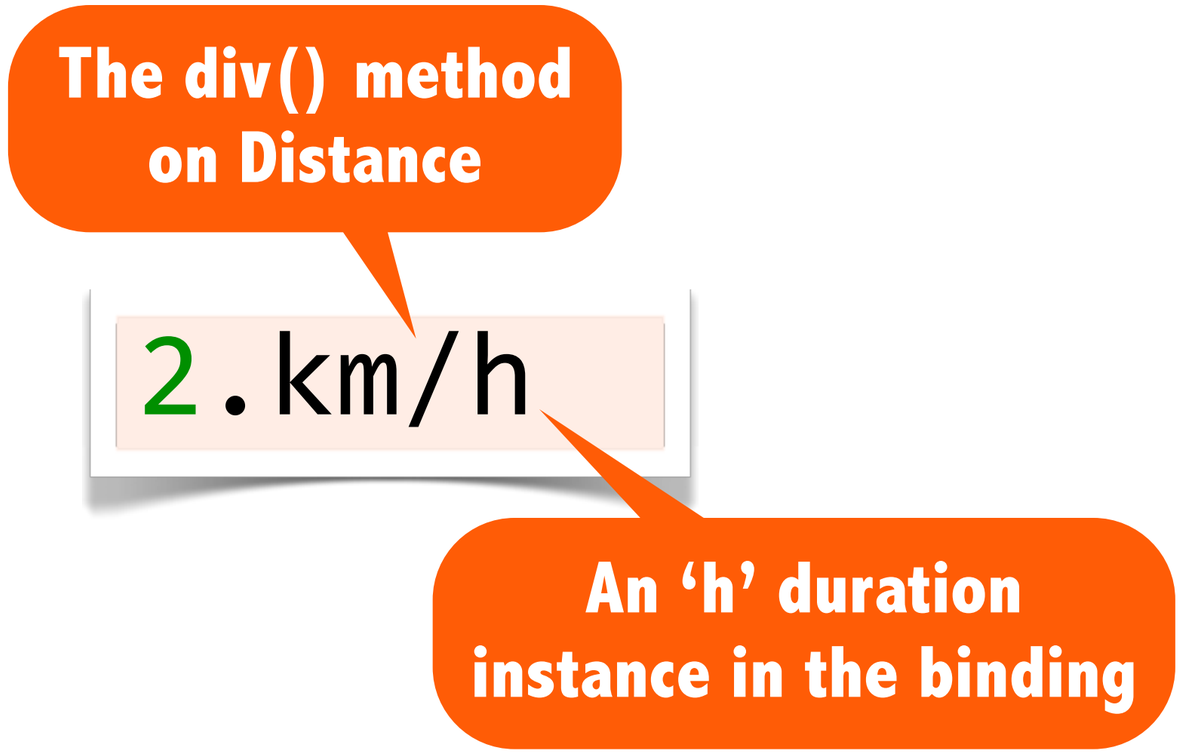
Distance okay, but speed?

- For distance, we just added a property access after the number, but we now need to divide (‘div’) by the time



Distance okay, but speed?

- For distance, we just added a property access after the number, but we now need to divide (‘div’) by the time



Inject the ‘h’ hour constant in the binding

```
def binding = new Binding([
    robot: new Robot(),
    *: Direction.values()
            .collectEntries {
                [(it.name()): it]
            },

    h: new Duration(1, TimeUnit.hour)
])
```

Inject the ‘h’ hour constant in the binding

An ‘h’ duration added to the binding

```
def binding = new Binding([
    robot: new Robot(),
    *: Direction.values()
            .collectEntries {
                [(it.name()): it]
            },

    h: new Duration(1, TimeUnit.hour)
])
```

Operator overloading

```
a + b    // a.plus(b)
a - b    // a.minus(b)
a * b    // a.multiply(b)
a / b    // a.div(b)
a % b    // a.modulo(b)
a ** b   // a.power(b)
a | b    // a.or(b)
a & b    // a.and(b)
a ^ b    // a.xor(b)
a[b]     // a.getAt(b)
a << b   // a.leftShift(b)
a >> b   // a.rightShift(b)
a >>> b // a.rightShiftUnsigned(b)
+a       // a.unaryPlus()
-a       // a.unaryMinus()
~a       // a.bitwiseNegate()
```

- Currency amounts
  - `15.euros + 10.dollars`
- Distance handling
  - `10.km - 10.m`
- Workflow, concurrency
  - `taskA | taskB & taskC`
- Credit an account
  - `account << 10.dollars`
    `account += 10.dollars`
    `account.credit 10.dollars`

Operator overloading

- Update the Distance class with a div() method following the naming convention for operators

```
class Distance {
    ...
    Speed div(Duration t) {
        new Speed(this, t)
    }
    ...
}
```

Operator overloading

- Update the Distance class with a div() method following the naming convention for operators

```
class Distance {
    ...
    Speed div(Duration t) {
        new Speed(this, t)
    }
    ...
}
```

Optional return

Equivalence of notation

- Those two notations are actually equivalent:

```
2.km/h
```

```
2.getKm().div(h)
```

Equivalence of notation

- Those two notations are actually equivalent:

```
2.km/h
```

```
2.getKm().div(h)
```

This one might be slightly more verbose!

Named parameters usage

```
move left, at: 3.km/h
```

Named parameters usage

```
move left, at: 3.km/h
```

Named parameters usage

```
move left, at: 3.km/h
```

Named parameters usage

```
move left, at: 3.km/h
```

Will call:

```
def move(Map m, Direction q)
```

Named parameters usage

```
move left, at: 3.km/h
```

Normal parameter

Named parameter

Will call:

```
def move(Map m, Direction q)
```

All named parameters go into the map argument

Named parameters usage

```
move left, at: 3.km/h
```

Normal parameter

Named parameter

Will call:

```
def move(Map m, Direction q)
```

All named parameters go into the map argument

Positional parameters come afterwards

Named parameters usage

```
move left, at: 3.km/h
```

Named parameters usage

```
move left, at: 3.km/h
```

Can we get rid of the comma?

Named parameters usage

```
move left, at: 3.km/h
```

Can we get rid of the comma?

What about the colon too?

Command chains

Groovy 1.8

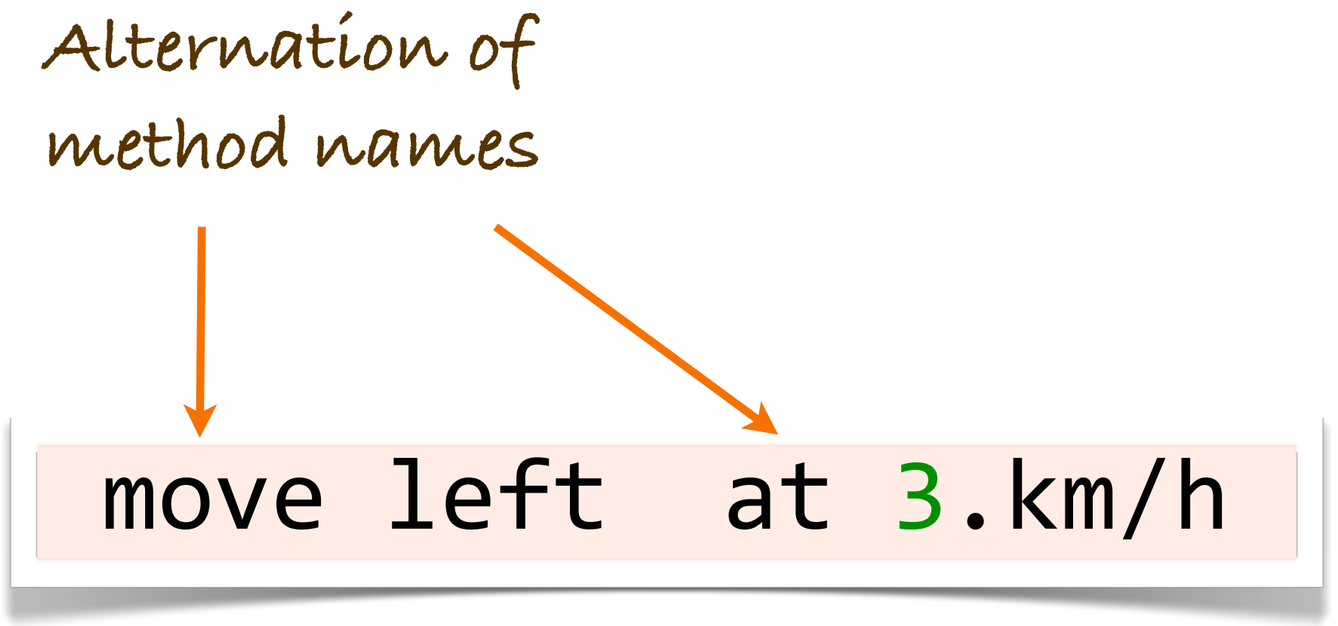
- A grammar improvement allowing you to **drop dots** & **parens** when **chaining method calls**
  - an extended version of top-level statements like println

- Less dots, less parens allow you to
  - write more readable business rules
  - in almost plain English sentences
    - (or any language, of course)

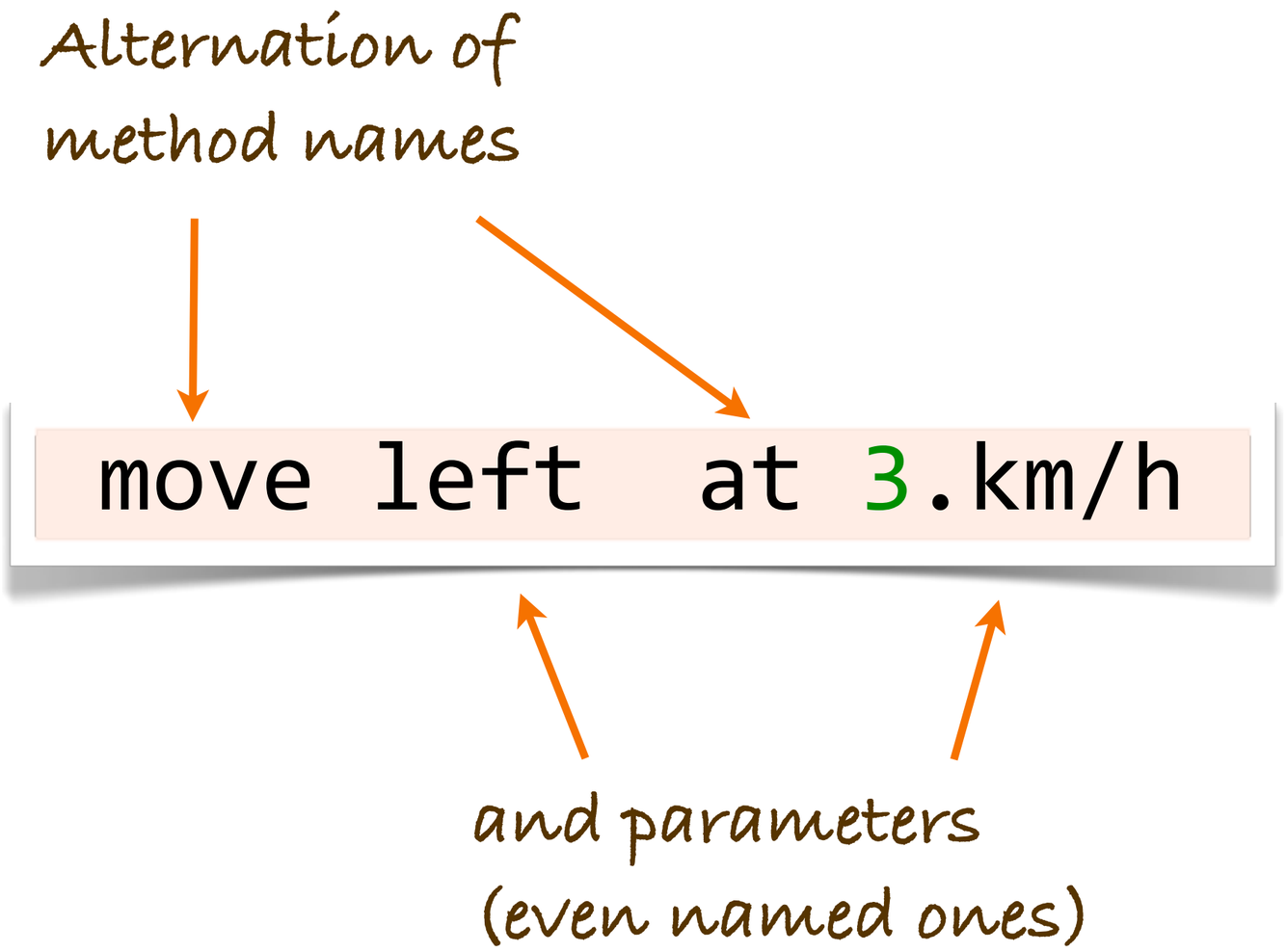
Command chains

```
move left  at 3.km/h
```

Command chains



Command chains



Command chains

```
move left  at 3.km/h
```

Command chains

Equivalent to:

```
move(left).at(3.km/h)
```


Look Ma!
No parens,
no dots!

Command chains

```
// Java fluent API approach
class Robot {
    ...
    def move(Direction dir) {
        this.dir = dir
        return this
    }

    def at(Speed speed) {
        this.speed = speed
        return this
    }
    ...
}
```

Command chains

```
def move(Direction dir) {
    [at: { Speed speed ->
        ...
    }]
}
```

Command chains

```
def move(Direction dir) {
    [at: { Speed speed ->
        ...
    }]
}
```

Nested maps and closures

Command chains

```
def move(Direction dir) {
    [at: { Speed speed ->
        ...
    }]
}
```

Nested maps and closures

Usage in your DSLs

```
move left at 3.km/h
```

Command chains

Command chains

```
// methods with multiple arguments (commas)
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
check that: vodka  tastes good
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
check that: vodka  tastes good

// closure parameters for new control structures
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
check that: vodka  tastes good

// closure parameters for new control structures
given {}  when {}  then {}
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
check that: vodka  tastes good

// closure parameters for new control structures
given {}  when {}  then {}

// zero-arg methods require parens
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
check that: vodka  tastes good

// closure parameters for new control structures
given {}  when {}  then {}

// zero-arg methods require parens
select all  unique() from names
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
check that: vodka  tastes good

// closure parameters for new control structures
given {}  when {}  then {}

// zero-arg methods require parens
select all  unique() from names

// possible with an odd number of terms
```

Command chains

```
// methods with multiple arguments (commas)
take coffee  with sugar, milk  and liquor

// leverage named-args as punctuation
check that: vodka  tastes good

// closure parameters for new control structures
given {}  when {}  then {}

// zero-arg methods require parens
select all  unique() from names

// possible with an odd number of terms
deploy left  arm
```

Command chains

```
// methods with multiple arguments (commas)
take(coffee).with(sugar, milk).and(liquor)

// leverage named-args as punctuation
check that: vodka  tastes good

// closure parameters for new control structures
given {}  when {}  then {}

// zero-arg methods require parens
select all  unique() from names

// possible with an odd number of terms
deploy left  arm
```

Command chains

```
// methods with multiple arguments (commas)
take(coffee).with(sugar, milk).and(liquor)

// leverage named-args as punctuation
check(that: vodka).tastes(good)

// closure parameters for new control structures
given {}  when {}  then {}

// zero-arg methods require parens
select all  unique() from names

// possible with an odd number of terms
deploy left  arm
```

Command chains

```
// methods with multiple arguments (commas)
take(coffee).with(sugar, milk).and(liquor)

// leverage named-args as punctuation
check(that: vodka).tastes(good)

// closure parameters for new control structures
given({}).when({}).then({})

// zero-arg methods require parens
select all  unique() from names

// possible with an odd number of terms
deploy left  arm
```

Command chains

```
// methods with multiple arguments (commas)
take(coffee).with(sugar, milk).and(liquor)

// leverage named-args as punctuation
check(that: vodka).tastes(good)

// closure parameters for new control structures
given({}).when({}).then({})

// zero-arg methods require parens
select(all).unique().from(names)

// possible with an odd number of terms
deploy left  arm
```

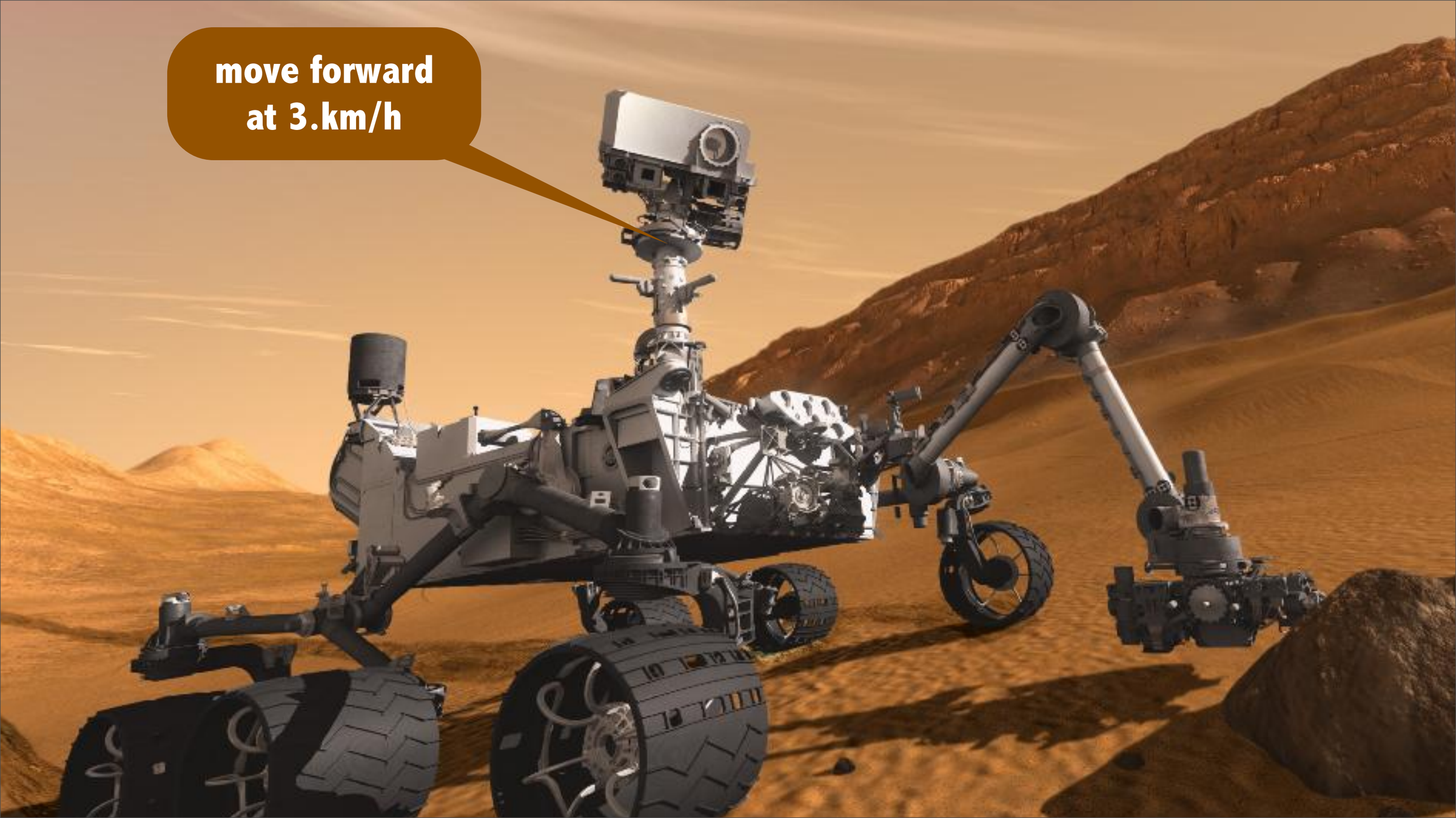
Command chains

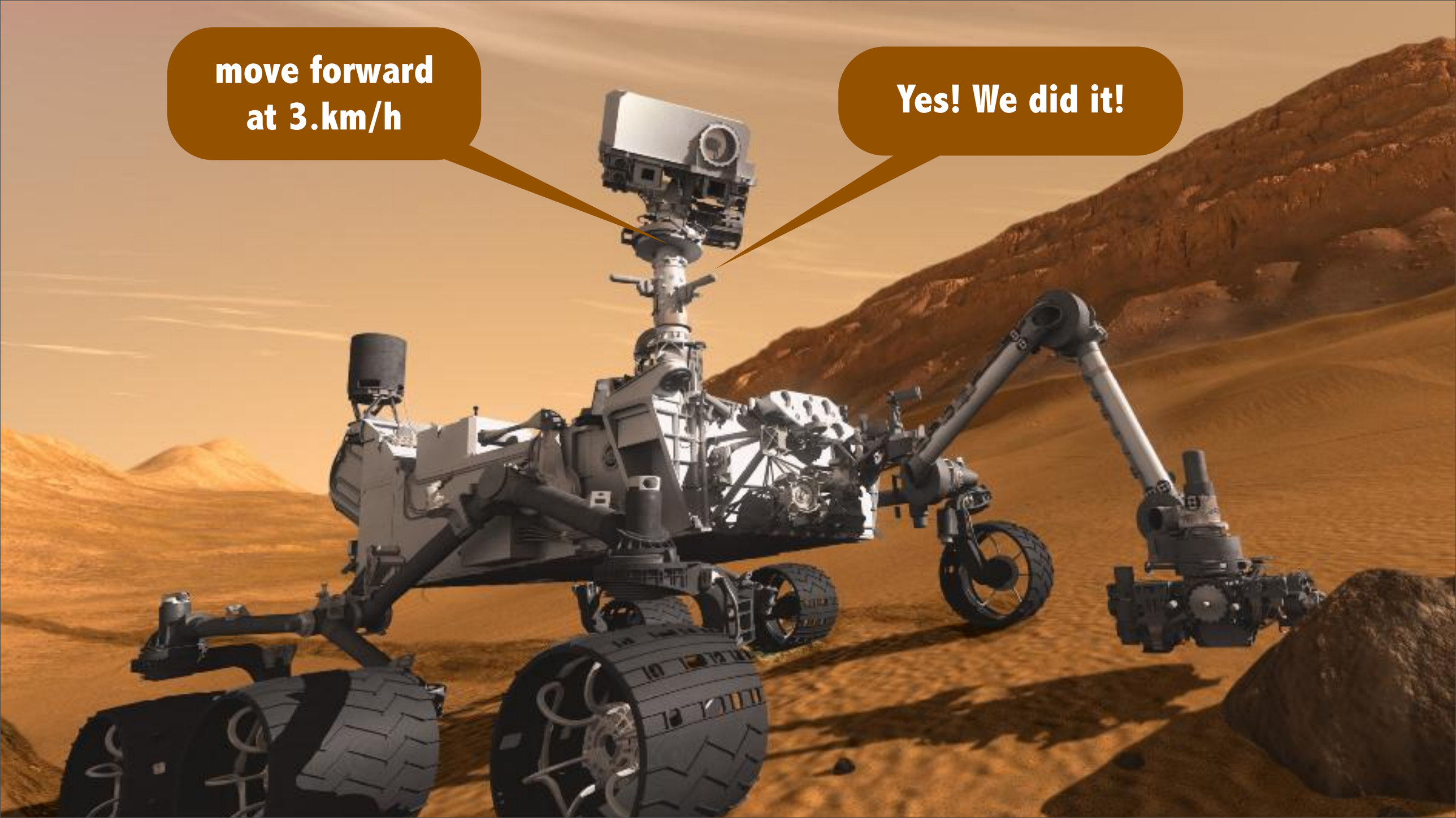
```
// methods with multiple arguments (commas)
take(coffee).with(sugar, milk).and(liquor)

// leverage named-args as punctuation
check(that: vodka).tastes(good)

// closure parameters for new control structures
given({}).when({}).then({})

// zero-arg methods require parens
select(all).unique().from(names)

// possible with an odd number of terms
deploy(left).arm
```

Final result

Final result

```
move forward at 3.km/h
```


move forward
at 3.km/h


move forward
at 3.km/h
Yes! We did it!


What about security and safety?
HAM
RANGER

Security and Safety

JVM Security Managers

SecureASTCustomizer

Sandboxing

Controlling script execution


Play it safe in a sandbox

Playing it safe…

- You have to think carefully about what DSL users are allowed to do with your DSL

- Forbid things which are not allowed
  - leverage the JVM's **Security Managers**
    - this might have an impact on performance
  - use a **Secure AST compilation customizer**
    - not so easy to think about all possible cases
  - avoid long running scripts with ***Interrupt transformations**

Security Managers

- Groovy is just a language leaving on the JVM,
  so you have access to the usual Security Managers mechanism
  - Nothing Groovy specific here
  - Please check the documentation on Security Managers
    and how to design policy files

SecureASTCustomizer

```
def secure = new SecureASTCustomizer()
secure.with {
   // disallow closure creation
   closuresAllowed = false
   // disallow method definitions
   methodDefinitionAllowed = false
   // empty white list => forbid certain imports
   importsWhitelist = [...]
   staticImportsWhitelist = [...]
   // only allow some static import
   staticStarImportsWhitelist = [...]
   // language tokens allowed
    tokensWhitelist = [...]
   // types allowed to be used
   constantTypesClassesWhiteList = [...]
   // classes who are allowed to be receivers of method calls
   receiversClassesWhiteList = [...]
}
def config = new CompilerConfiguration()
config.addCompilationCustomizers(secure)
def shell = new GroovyShell(config)
```

Controlling code execution

- Your application may run user's code
  - what if the code runs in infinite loops or for too long?
  - what if the code consumes too many resources?

- 3 new transforms at your rescue
  - **@ThreadInterrupt**: adds Thread#isInterrupted checks so your executing thread stops when interrupted
  - **@TimedInterrupt**: adds checks in method and closure bodies to verify it's run longer than expected
  - **@ConditionalInterrupt**: adds checks with your own conditional logic to break out from the user code

@ThreadInterrupt

```
@ThreadInterrupt
import groovy.transform.ThreadInterrupt

while (true) {


    // Any extraterestrial around?
}
```

@ThreadInterrupt

```
@ThreadInterrupt
import groovy.transform.ThreadInterrupt

while (true) {
    if (Thread.currentThread().isInterrupted()) {
        throw new InterruptedException()
    // Any extraterestrial around?
}
```

@TimedInterrupt

```
@TimedInterrupt(10)
import groovy.transform.TimedInterrupt

while (true) {
    move left
    // circle forever
}
```

- `InterruptedException` thrown

  when checks indicate code ran longer than desired

@ConditionalInterrupt

- **Specify your own conditions** to be inserted at the start of method and closure bodies
  - check for available resources, number of times run, etc.
- Leverages **closure annotation parameters** Groovy 1.8

```
@ConditionalInterrupt({ battery.level < 0.1 })
import groovy.transform.ConditionalInterrupt

100.times {
    move forward at 10.km/h
}
```

@ConditionalInterrupt

- **Specify your own conditions** to be inserted at the start of method and closure bodies
  - check for available resources, number of times run, etc.
- Leverages **closure annotation parameters** **Groovy 1.8**

```
@ConditionalInterrupt({ battery.level < 0.1 })
import groovy.transform.ConditionalInterrupt

100.times {
    move forward at 10.km/h
}
```

Can we avoid typing the conditional interrupt?

@ConditionalInterrupt

- **Specify your own conditions** to be inserted at the start of method and closure bodies
  - check for available resources, number of times run, etc.
- Leverages **closure annotation parameters** **Groovy 1.8**

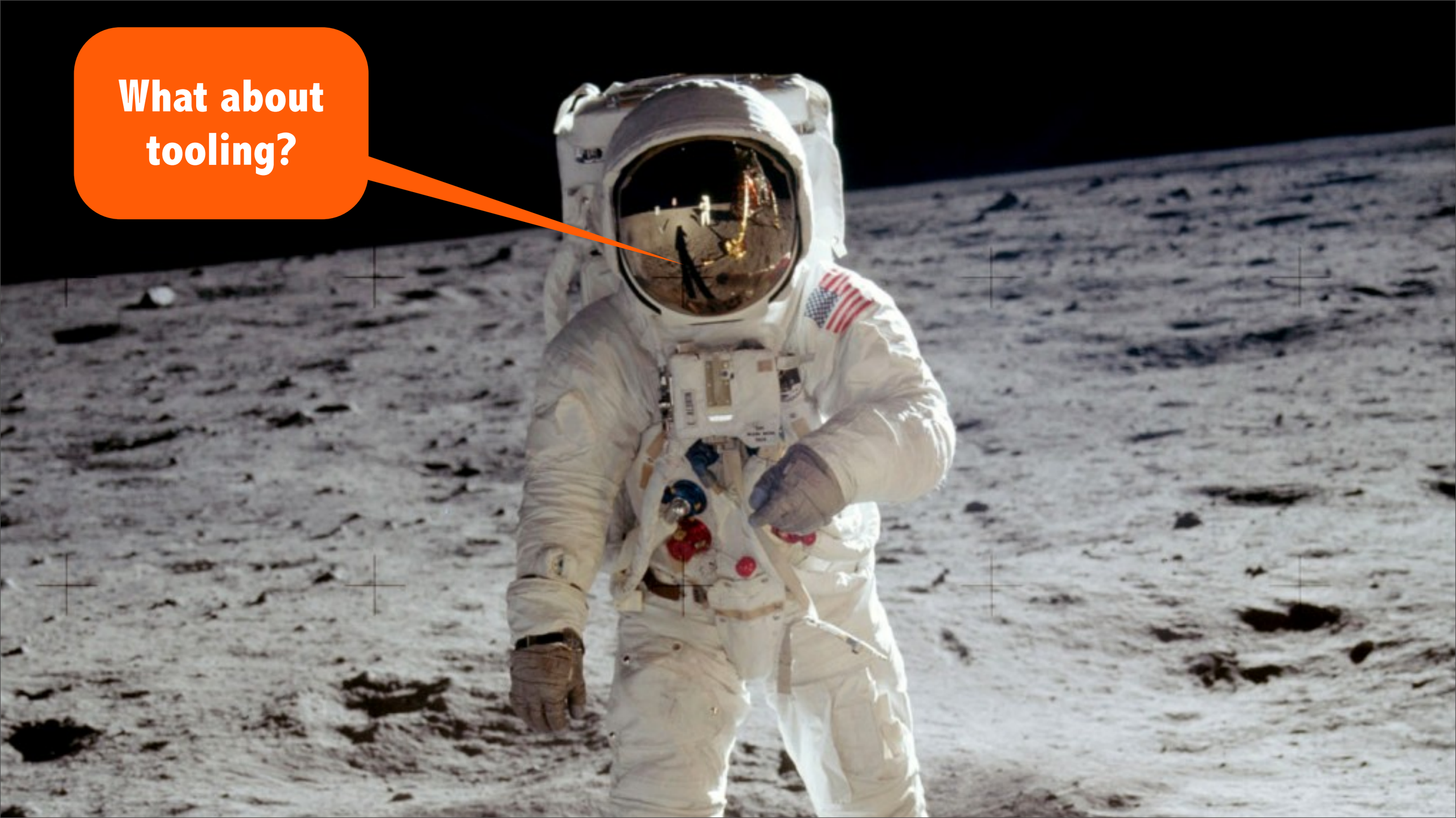
```
100.times {
    move forward at 10.km/h
}
```

Yes! Using compilation customizers

Using compilation customizers

- In our previous examples, the **usage of the interrupts were explicit**, and users had to type them
  - if they want to deplete the battery of your robot, they won't use interrupts, so you have to **impose interrupts yourself**

- With compilation customizers you can **inject those interrupts** thanks to the **AST Transformation Customizer**


What about tooling?

Tooling

Why tooling?

DSL descriptors

Pointcuts and contributions

Packaging DSLDs

Why tooling?

- I know what this language means
  - why do I want anything more?

Why tooling?

- I know what this language means
  - why do I want anything more?

- But, tooling can make things even better
  - syntax checking
  - content assist
  - search
  - inline documentation

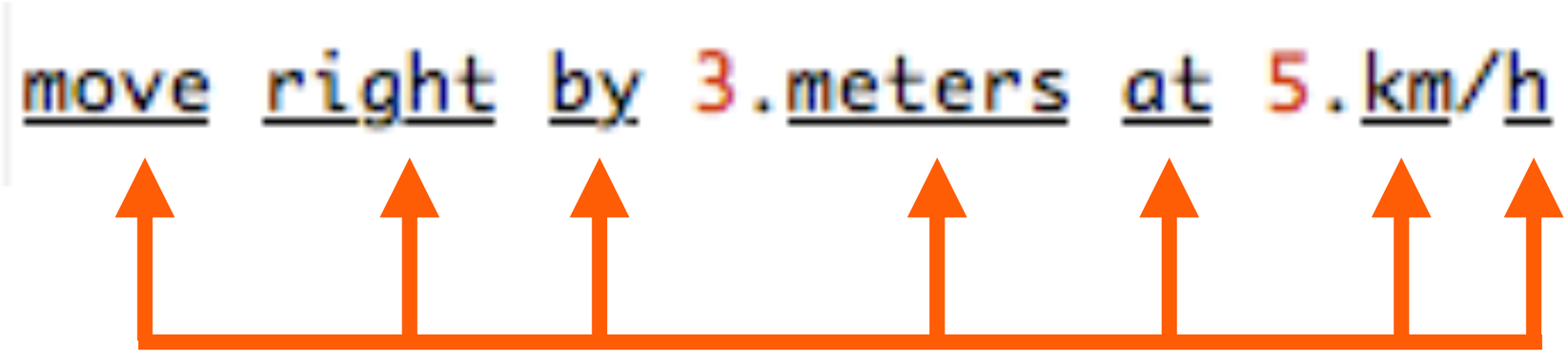
Let's use an IDE

- I hear Groovy-Eclipse is pretty good…

```
move right by 3.meters at 5.km/h
```

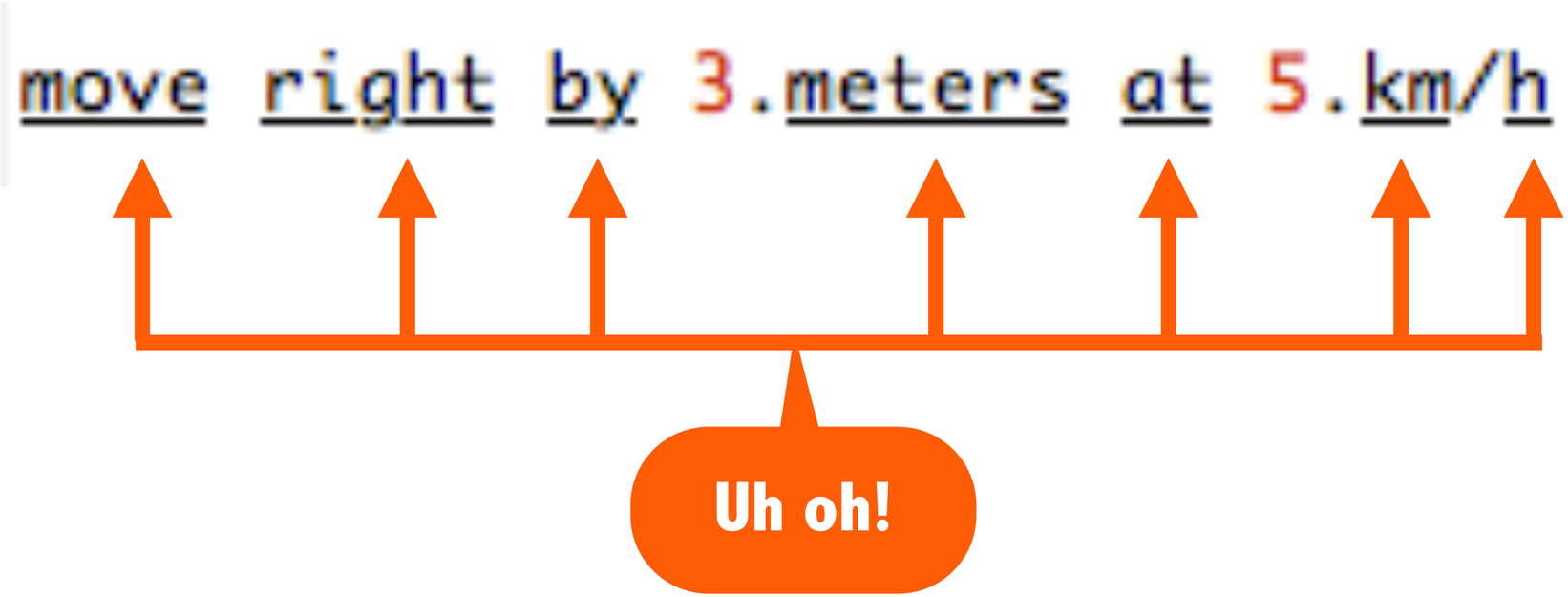
Let's use an IDE

- I hear Groovy-Eclipse is pretty good…



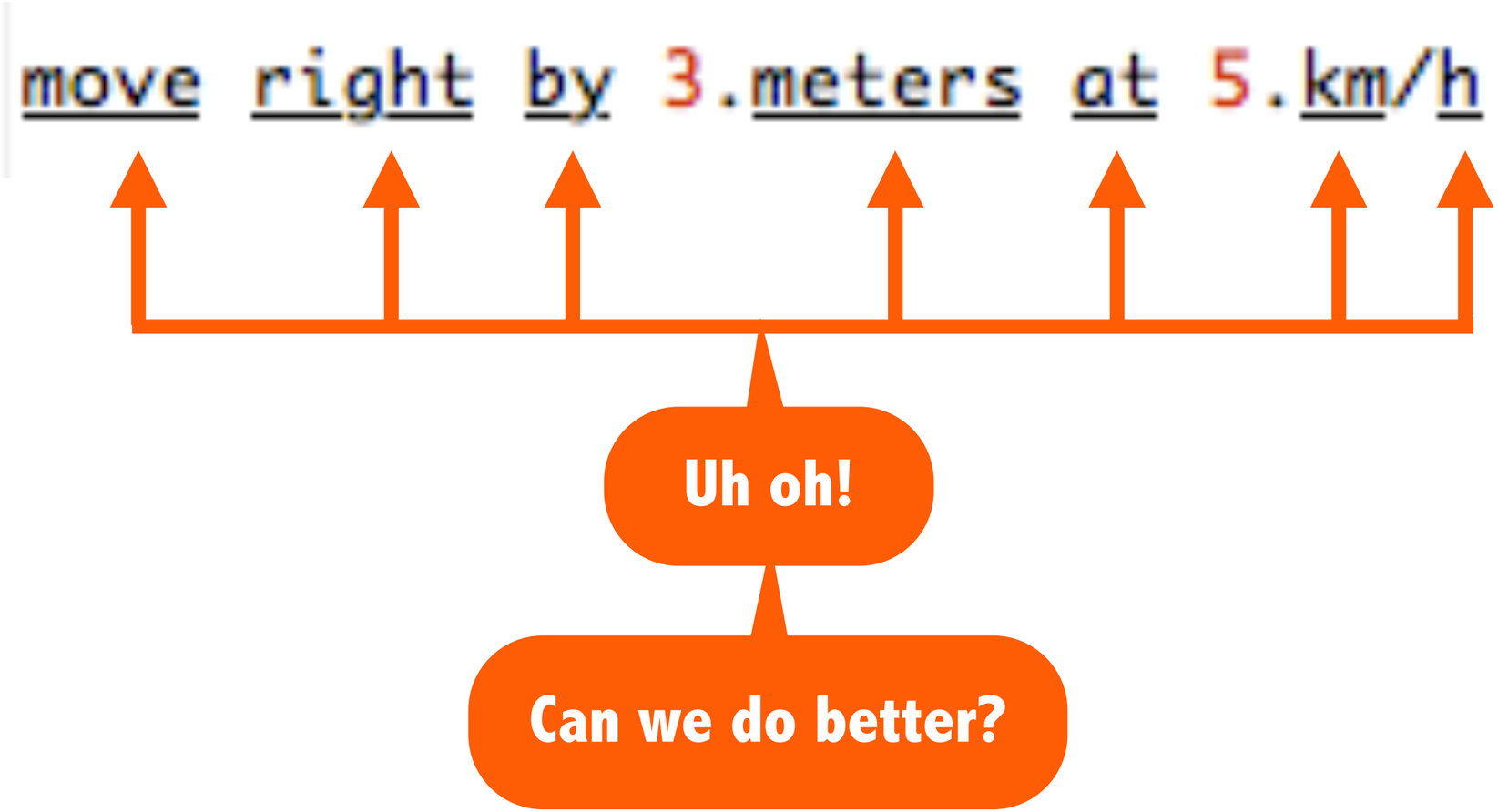
Let's use an IDE

- I hear Groovy-Eclipse is pretty good…



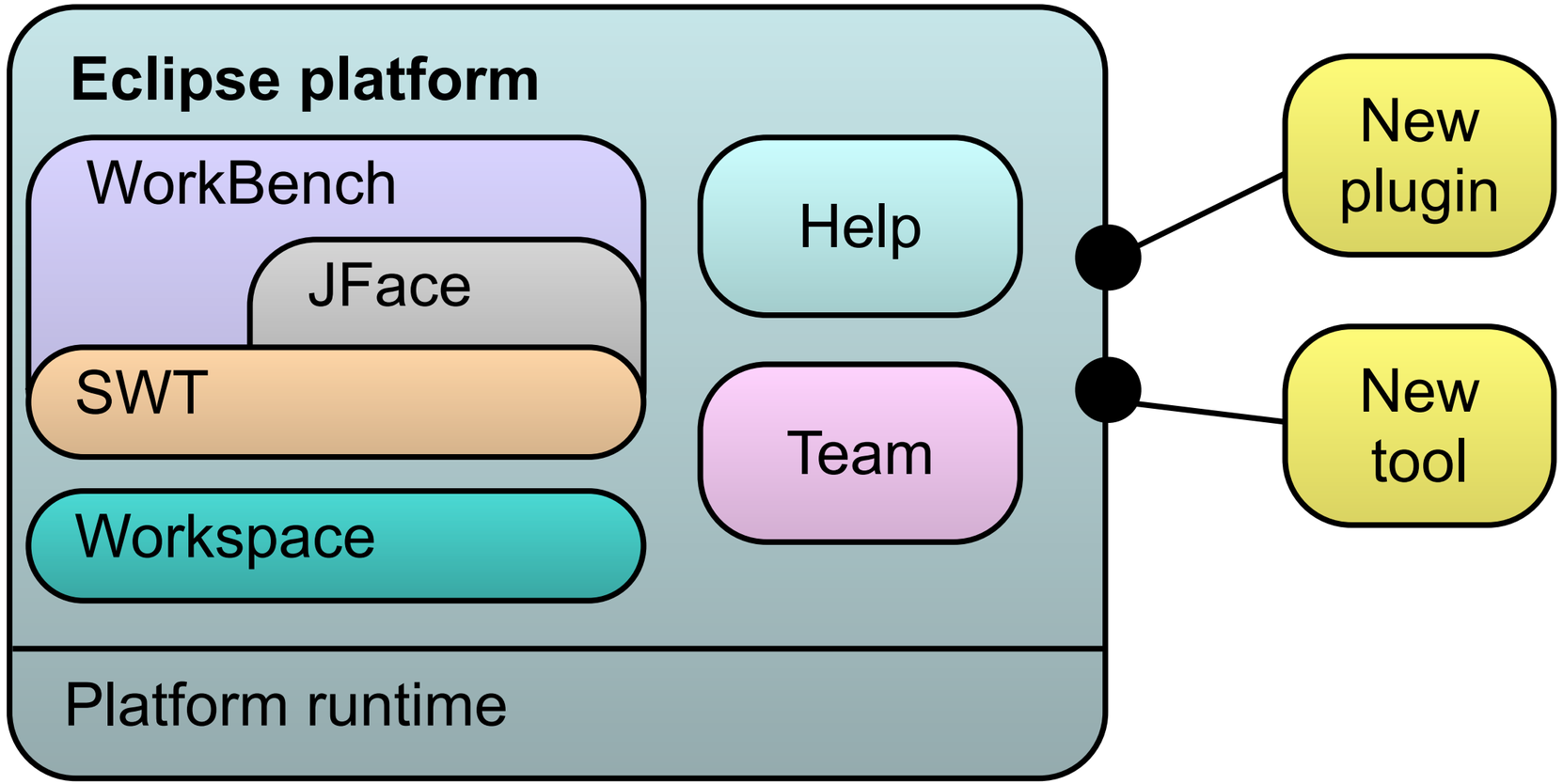
Let's use an IDE

- I hear Groovy-Eclipse is pretty good…



Of course!

- Eclipse is extensible
  - with a plugin architecture



I want my DSL supported in Eclipse

I want my DSL supported in Eclipse

- Let's create a plugin
  - create a plugin project
  - extend an extension point
  - write the code
  - build the plugin
  - host on an update site
  - convince people to install it

I want my DSL supported in Eclipse

- Let's create a plugin
  - create a plugin project
  - extend an extension point
  - write the code
  - build the plugin
  - host on an update site
  - convince people to install it
- Problems
  - I don't want to learn Eclipse APIs
  - I want an easy way for users to install the DSL support
  - I need a specific plugin version for my specific DSL version

I want my DSL supported in Eclipse

- Let's create a plugin
  - create a plugin project
  - extend an extension point
  - write the code
  - build the plugin
  - host on an update site
  - convince people to install it
- Problems
  - I don't want to learn Eclipse APIs
  - I want an easy way for users to install the DSL support
  - I need a specific plugin version for my specific DSL version

I want my DSL supported in Eclipse

- Let's create a plugin
  - create a plugin project
  - extend an extension point
  - write the code
  - build the plugin
  - host on an update site
  - convince people to install it

- Problems
  - I don't want to learn Eclipse APIs
  - I want an easy way for users to install the DSL support
  - I need a specific plugin version for my specific DSL version

Uh oh!

Can we do better?

Of course!

- Groovy is extensible!
  - Meta-Object Protocol
  - Metaprogramming
  - DSLs...

DSL Descriptors

- Teach the IDE about DSLs through a Groovy DSL

```
move right, by: 3.meters, at: 5.km/h
```

Map RobotScript.move(Direction dir)
Moves the robot in a given v11.Direction

DSL Descriptors

- Teach the IDE about DSLs through a Groovy DSL

```
move right, by: 3.meters, at: 5.km/h
```

Map RobotScript.move(Direction dir)
Moves the robot in a given v11.Direction

- **Benefits**
  - Powerful
  - Uses Groovy syntax, semantics, and APIs
  - No knowledge of Eclipse required
  - Can ship with Groovy libraries

DSL Descriptors

- Teach the IDE about DSLs through a Groovy DSL

```
move right, by: 3.meters, at: 5.km/h
```

```
Map RobotScript.move(Direction dir)
Moves the robot in a given v11.Direction
```

- **Benefits**
  - Powerful
  - Uses Groovy syntax, semantics, and APIs
  - No knowledge of Eclipse required
  - Can ship with Groovy libraries

DSL Descriptors

- Teach the IDE about DSLs through a Groovy DSL

```
move right, by: 3.meters, at: 5.km/h
```

```
Map RobotScript.move(Direction dir)
Moves the robot in a given v11.Direction
```

- **Benefits**
  - Powerful
  - Uses Groovy syntax, semantics, and APIs
  - No knowledge of Eclipse required
  - Can ship with Groovy libraries

Let’s start simple

```
move
deploy
h
left
right
forward
backward
```

Let's start simple

```
move
deploy
h
left
right
forward
backward
```

- In English:
  - *When the type is* **`this`**, *add the following properties/methods*
    - **`move`**, **`deploy`**, **`h`**, etc from binding
    - **`Direction`** from import customizer

Let's start simple

```
move
deploy
h
left
right
forward
backward
```

- In English:
  - *When the type is* **`this`**, *add the following properties/methods*
    - **`move`**, **`deploy`**, **`h`**, etc from binding
    - **`Direction`** from import customizer
- In DSLD:
  - *When the type is* **`this`**

```
contribute( isThisType() ) {…}
```

  - …properties/methods…

```
property name: left, type: 'v11.Direction' …
method name: move, type: 'java.util.Map<…>'
```

Let's start simple

```
move
deploy
h
left
right
forward
backward
```

- In English:
  - *When the type is* **`this`**, *add the following properties/methods*
    - **`move`**, **`deploy`**, **`h`**, etc from binding
    - **`Direction`** from import customizer
- In DSLD:
  - *When the type is* **`this`**

```
contribute( isThisType() ) {…}
```

Pointcut

  - …properties/methods…

```
property name: left, type: 'v11.Direction' …
method name: move, type: 'java.util.Map<…>'
```

Let's start simple

```
move
deploy
h
left
right
forward
backward
```

- In English:
  - *When the type is* **`this`**, *add the following properties/methods*
    - **`move`**, **`deploy`**, **`h`**, etc from binding
    - **`Direction`** from import customizer
- In DSLD:
  - *When the type is* **`this`**

```
contribute( isThisType() ) {…}
```

  - …properties/methods…

```
property name: left, type: 'v11.Direction' …
method name: move, type: 'java.util.Map<…>'
```

Pointcut

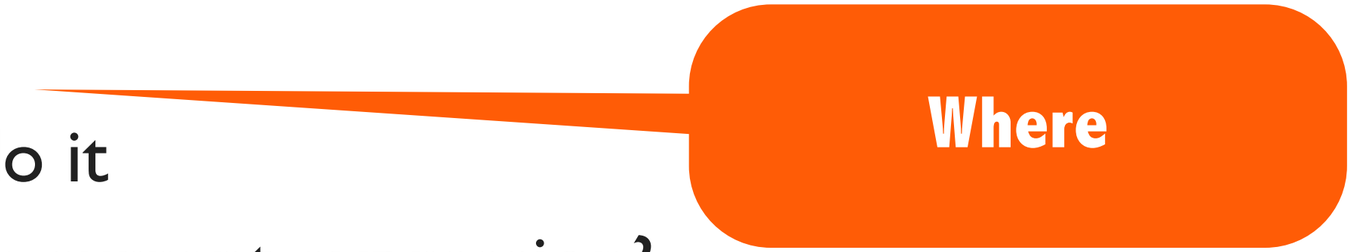
Contribution block

Anatomy of a DSLD script

- **Pointcuts**
  - *Where* to do it
  - What is the current expression?
  - Current type?
  - Enclosing class?
- **Contribution blocks**
  - *What* to do
  - « Add » method
  - « Add » property
  - Delegate to another type

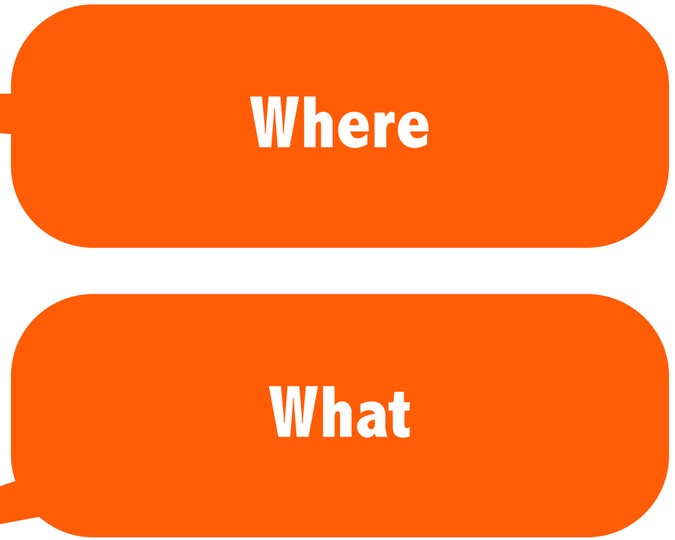
Anatomy of a DSLD script

- **Pointcuts**
  - *Where* to do it
  - What is the current expression?
  - Current type?
  - Enclosing class?
- **Contribution blocks**
  - *What* to do
  - « Add » method
  - « Add » property
  - Delegate to another type



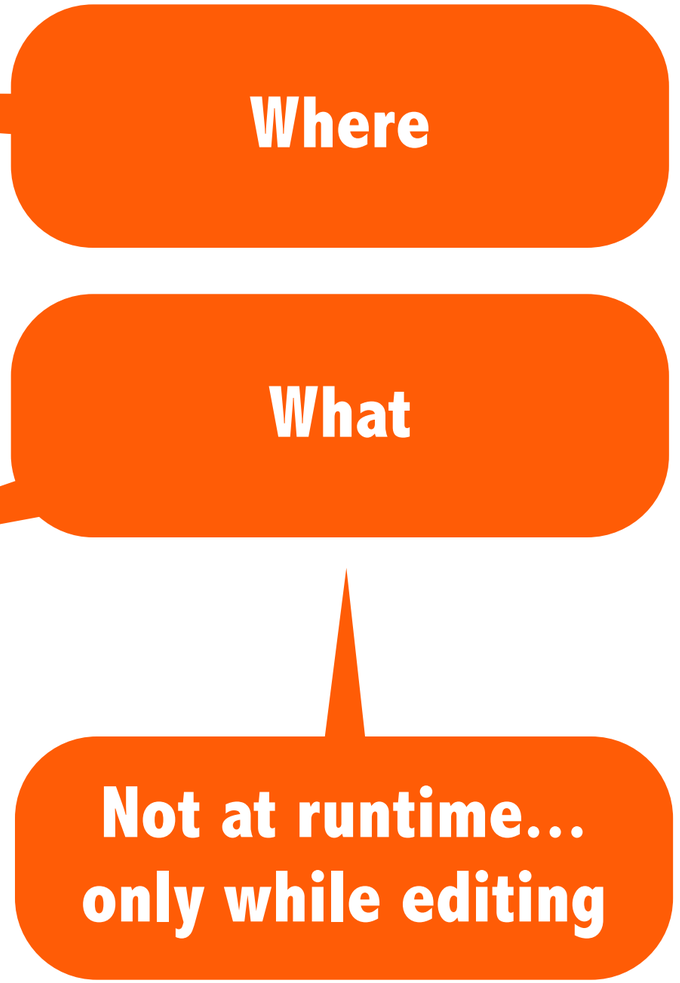
Anatomy of a DSLD script

- **Pointcuts**
  - *Where* to do it
  - What is the current expression?
  - Current type?
  - Enclosing class?
- **Contribution blocks**
  - *What* to do
  - « Add » method
  - « Add » property
  - Delegate to another type



Anatomy of a DSLD script

- **Pointcuts**
  - *Where* to do it
  - What is the current expression?
  - Current type?
  - Enclosing class?
- **Contribution blocks**
  - *What* to do
  - « Add » method
  - « Add » property
  - Delegate to another type



Talking about « x »

```
class Other { }

class Foo {
    def method() {
        def x = new Other()
        x.nuthin
    }
}
```

Talking about « x »

Current type

```
class Other { }

class Foo {
    def method() {
        def x = new Other()
        x.nuthin
    }
}
```

Talking about « x »

Current type

Enclosing class

```
class Other { }

class Foo {
    def method() {
        def x = new Other()
        x.nuthin
    }
}
```

Talking about « x »

Current type

Enclosing class

Enclosing method

```
class Other { }

class Foo {
    def method() {
        def x = new Other()
        x.nuthin
    }
}
```

Pointcuts

Pointcuts

```
currentType()     // matches on current declaring type
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script

currentType("groovy.dsl.Robot")
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script

currentType("groovy.dsl.Robot")
currentType(subType("groovy.dsl.Robot"))
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script

currentType("groovy.dsl.Robot")
currentType(subType("groovy.dsl.Robot"))
currentType(method("move"))
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script

currentType("groovy.dsl.Robot")
currentType(subType("groovy.dsl.Robot"))
currentType(method("move"))
currentType(annotatedBy("groovy.dsl.Robotic"))
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script

currentType("groovy.dsl.Robot")
currentType(subType("groovy.dsl.Robot"))
currentType(method("move"))
currentType(annotatedBy("groovy.dsl.Robotic"))

// combining them, and using the logical and
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script

currentType("groovy.dsl.Robot")
currentType(subType("groovy.dsl.Robot"))
currentType(method("move"))
currentType(annotatedBy("groovy.dsl.Robotic"))

// combining them, and using the logical and
isScript(
    annotatedBy("groovy.dsl.Robotic")
```

Pointcuts

```
currentType()     // matches on current declaring type
isScript()        // matches on the enclosing script

currentType("groovy.dsl.Robot")
currentType(subType("groovy.dsl.Robot"))
currentType(method("move"))
currentType(annotatedBy("groovy.dsl.Robotic"))

// combining them, and using the logical and
isScript(
    annotatedBy("groovy.dsl.Robotic")
) & currentType(method("move"))
```

What goes in a contribution block?

What goes in a contribution block?

- **property**: “adds” a property
  - name: “myName” (REQUIRED)
  - type: “java.lang.String”
  - declaringType: ”com.foo.Frumble”
  - doc: “Some JavaDoc”

What goes in a contribution block?

- **property**: “adds” a property
  - name: “myName” (REQUIRED)
  - type: “java.lang.String”
  - declaringType: ”com.foo.Frumble”
  - doc: “Some JavaDoc”
- **method**: “adds” a method
  - all arguments above, and
  - params: [firstName:“java.lang.String”, lastName:“java.lang.String”]
  - namedParams, optionalParams

What goes in a contribution block?

- **property**: “adds” a property
  - name: “myName” (REQUIRED)
  - type: “java.lang.String”
  - declaringType: ”com.foo.Frumble”
  - doc: “Some JavaDoc”
- **method**: “adds” a method
  - all arguments above, and
  - params: [firstName:“java.lang.String”, lastName:“java.lang.String”]
  - namedParams, optionalParams
- **delegatesTo**: “delegates” invocations to another type
  - type (REQUIRED)

What goes in a contribution block?

- **property**: "adds" a property
  - name: "myName" (REQUIRED)
  - type: "java.lang.String"
  - declaringType: "com.foo.Frumble"
  - doc: "Some JavaDoc"
- **method**: "adds" a method
  - all arguments above, and
  - params: [firstName:"java.lang.String", lastName:"java.lang.String"]
  - namedParams, optionalParams
- **delegatesTo**: "delegates" invocations to another type
  - type (REQUIRED)

```
contribute(...) {
  property name: "myName"
  method name: "getMyName"
  delegatesTo type:
      "some.other.Type"
}
```

Wait… isn't this Aspect-Oriented Programming?

Wait… isn't this Aspect-Oriented Programming?

- Pointcut
  - Intentionally borrowed from AOP

Wait… isn't this Aspect-Oriented Programming?

- Pointcut
  - Intentionally borrowed from AOP
- AspectJ: pointcuts and advice
  - operates on Java instructions at runtime

Wait… isn't this Aspect-Oriented Programming?

- Pointcut
  - Intentionally borrowed from AOP
- AspectJ: pointcuts and advice
  - operates on Java instructions at runtime
- DSLD: **pointcuts** and **contribution blocks**
  - operates on AST in the editor org.codehaus.groovy.ast.expr.*

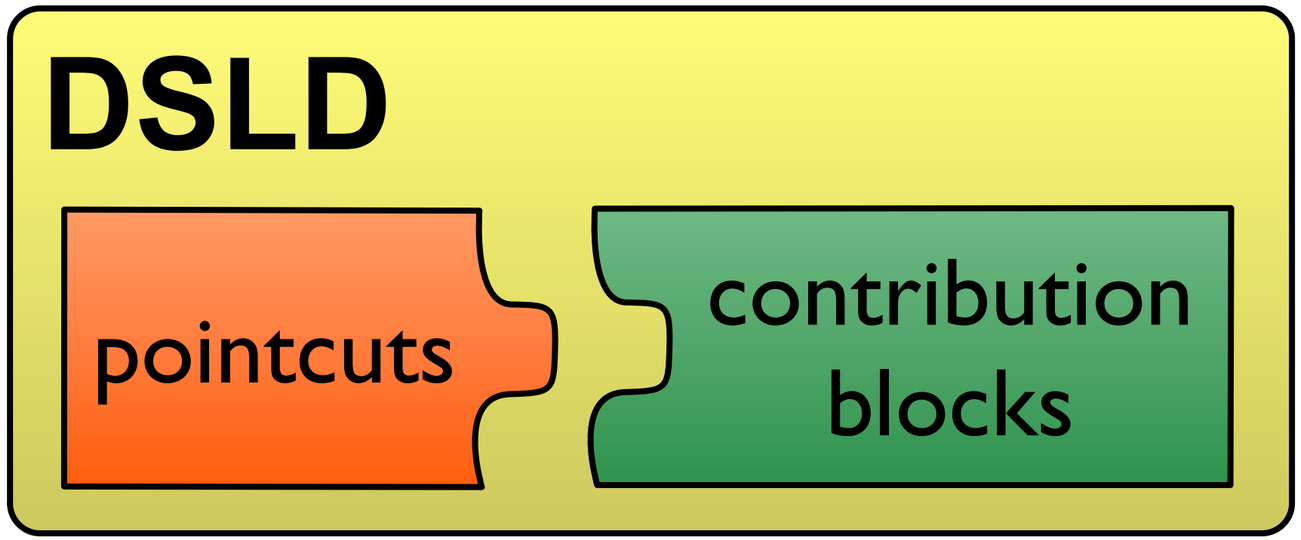
Wait… isn't this Aspect-Oriented Programming?

- Pointcut
  - Intentionally borrowed from AOP
- AspectJ: pointcuts and advice
  - operates on Java instructions at runtime
- DSLD: **pointcuts** and **contribution blocks**
  - operates on AST in the editor org.codehaus.groovy.ast.expr.*
- Join Point Model
  - Join points (e.g., instructions, expressions)
  - Mechanism for quantifying join points (e.g., pointcuts)
  - Means of affect at a join point (e.g., advice, contribution blocks)

How do we ship it?

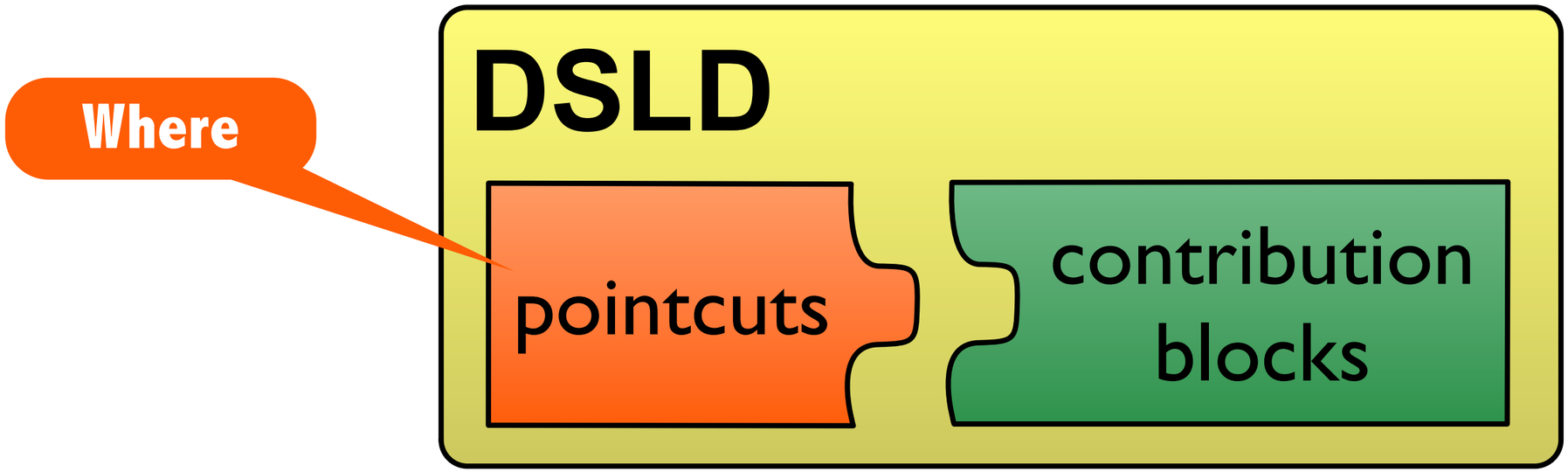
- jar/war file
- DSLD file:
  - as source in `dsld` package
- Hint:
  - Use script folder support in preferences
  - `**/*.dsld` to be copied to bin folder as source
- Can also use maven or gradle

To summarize: Editing support for DSLs



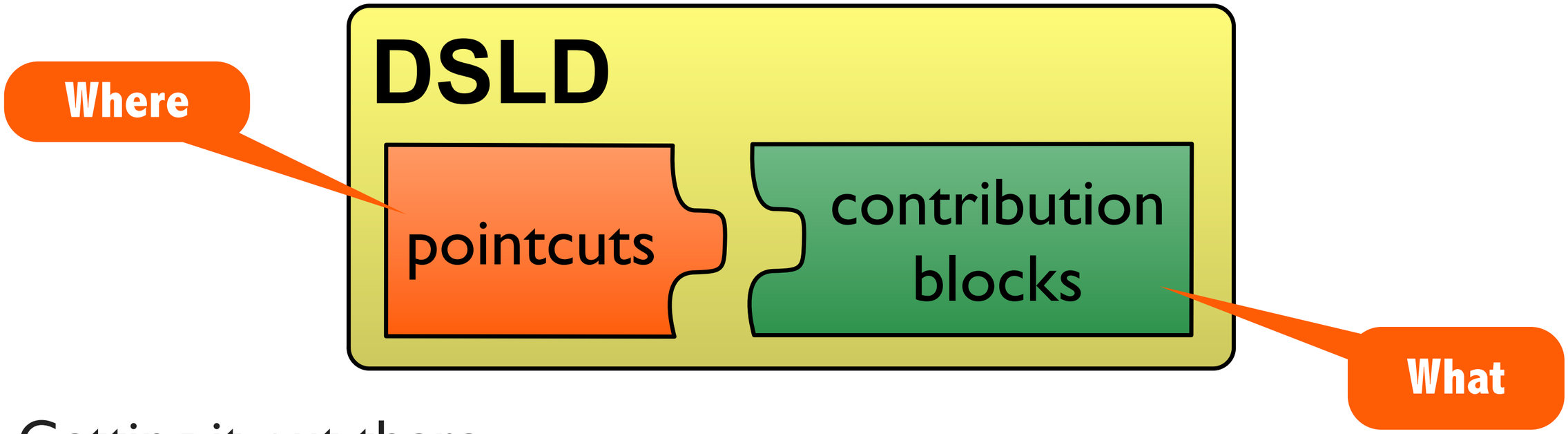
- Getting it out there
  - include a `dsld` package in your JAR
  - add the DSLD for your DSL to the package as source
  - ship it!

To summarize: Editing support for DSLs



- Getting it out there
  - include a `dsld` package in your JAR
  - add the DSLD for your DSL to the package as source
  - ship it!

To summarize: Editing support for DSLs



- Getting it out there
  - include a `dsld` package in your JAR
  - add the DSLD for your DSL to the package as source
  - ship it!

What have we learnt?

Groovy Power!™

- A **flexible** and **malleable syntax**
  - scripts vs classes, optional typing, colons and parens
- Groovy offers useful **dynamic features** for DSLs
  - operator overloading, ExpandoMetaClass
- Can write almost **plain natural language sentences**
  - for readable, concise and expressive DSLs
- Groovy DSLs are **easy to integrate**,
  and can be **secured** to run safely in your own **sandbox**
- Groovy **DSLs can be tooled**
  for **improved authoring capabilities**

Groovy Power!™

Groovy is a great fit for DSLs!

- A **flexible** and **malleable syntax**
  - scripts vs classes, optional typing, colons and parens
- Groovy offers useful **dynamic features** for DSLs
  - operator overloading, ExpandoMetaClass
- Can write almost **plain natural language sentences**
  - for readable, concise and expressive DSLs
- Groovy DSLs are **easy to integrate**,
  and can be **secured** to run safely in your own **sandbox**
- Groovy **DSLs can be tooled**
  for **improved authoring capabilities**

And there's more!

- We haven't dived into...
  - How to implement your **own control structures** with closures
  - How to create Groovy « **builders** »
  - How to define **extension modules**
  - How to hijack the Groovy syntax to develop our own language extensions with **AST Transformations**
  - **Source preprocessing** for custom syntax
  - How to use the **other metaprogramming techniques** available
  - How to improve **error reporting** with customizers

Thank you!

Questions & Answers

Got questions, really?

Questions & Answers

Image credits

- Wikipedia logo: http://www.geekosystem.com/wp-content/uploads/2011/01/wikipedia-logo.png
- Chains: http://2.bp.blogspot.com/-GXDVqUYSCa0/TVdBsON4tdI/AAAAAAAAAW4/EgJOUmAxB28/s1600/breaking-chains5_copy9611.jpg
- Space odissey: http://dearjesus.files.wordpress.com/2010/04/2001_a_space_odyssey_1.jpg
- Yes we scan: http://i.huffpost.com/gen/1218045/thumbs/o-YES-WE-SCAN-facebook.jpg
- HAL red: http://2.bp.blogspot.com/-yjsyPxUFicY/TcazwAltOaI/AAAAAAAAAho/GVT7wGhnrUM/s1600/2001-a-space-odyssey-HAL.jpg
- USSR Space posters: http://www.flickr.com/photos/justinvg
- General: http://www.defense.gov/dodcmsshare/newsphoto/2009-02/hires_090217-D-7203C-004.jpg
- Rocket: http://astro.vision.free.fr/download/fonds/7/saturn5c.jpg
- Star Trek / 747: http://24.media.tumblr.com/tumblr_m35foiJI6a1qzz0iho1_1280.jpg
- Man in space: http://www.nasa.gov/images/content/60130main_image_feature_182_jwfull.jpg
- Sputnik 2: http://launiusr.files.wordpress.com/2010/06/sputnik2.jpg
- Lunakod: http://www.astr.ua.edu/keel/space/lunakhod_moscow.jpg
- Sandbox: http://www.turnbacktogod.com/wp-content/uploads/2008/09/sandbox.jpg
- Repair: http://www.oneangels.com/wp-content/uploads/2012/03/repair1.jpg
- Mars rover: http://wallpapers.free-review.net/wallpapers/49/Mars_rover%2C_Mars_-_03.jpg
- Mars rover 2: http://www.universetoday.com/wp-content/uploads/2011/06/551038main_pia14156-43_946-710.jpg
- Thumb: http://www.wpclipart.com/sign_language/thumbs_up_large.png.html
- Night sky: http://www.aplf-planetariums.info/galeries/ciel_profond/2004-07-01-Voie_Lactee_Scorpion-Jean-Luc_PUGLIESI.jpg
- Obama yes we can: http://www.dessinemoiunboulon.net/wp-content/uploads/2009/01/obama-yes-we-can_04-nov-08.jpg
- Hook: http://winningware.com/blog/wp-content/uploads/2009/12/FishHookXSmall.jpg
- HP 48 GX: http://calculators.torensma.net/files/images/hewlett-packard_hp-48g.jpg
- Omer: http://www.irmin.com/wallpaper/TV/Homer%20Simpson%20Oh%20No.jpg
- Cadenat: http://acsgsecurite.com/upl/site/cadenat.png
- Thanks: http://4.bp.blogspot.com/-hTdT5Ebk5ws/Tu_x2tE4ccI/AAAAAAAAAZc/pxtG8A0w7VE/s1600/thanks-digital-calligraphy-md.png
- Buzz Aldrin: http://2.bp.blogspot.com/-rpV5Oy5N78U/TprVli-2ZdI/AAAAAAAABN8/WiHob4rp2b8/s1600/Astronaut.jpg