

Magic Literals In Pharo

Julien Delplanque, Stéphane Ducasse, and Oleksandr Zaitsev

{firstname}.{lastname}@inria.fr



```

method name parameter
exampleWithNumber: x

<syntaxOn: #postcard> ] pragma
"A ""complete"" Pharo syntax" ] comment
| y | local variable
true & false not & (nil isNil) ] block
  boolean literals nil literal unary message
  iffFalse: [ self perform: #add: with: x ].
  assignment pseudo variables keyword message
  y := thisContext stack size + super size.
byteArray := #[2 2r100 8r20 16rFF].
  instance variable integer literals byte array
  array generated at runtime literal array
  { -42 . #($a #a #'I'm' 'a' 1.0 1.23e2 3.14s2 1) }
  do: [ :each |
    local block variable block parameter global variable
    | var |
    var := Transcript
    show: each class name; ] cascade
    show: each printString ].
  keyword message
^ x < y
  return instruction

```

other method definition examples:
 unary
 + binaryMessageArgument
 keyword: arg
 keyword: arg1 withTwo: arg2

<https://www.pharo.org>

PLACE
STAMP
HERE

.....

.....

.....

.....

Literals in Smalltalk



```

method name parameter
exampleWithNumber: x

<syntaxOn: #postcard> ]
"A ""complete"" Pharo syntax" ]
| y |
true & false not & (nil isNil)
  iffFalse: [ self perform: #add: with: x ].
y := thisContext stack size + super size.
byteArray := #[2 2r100 8r20 16rFF].
{ -42 . #($a #a #'I'm' 'a' 1.0 1.23e2 3.14s2 1) }
do: [ :each |
  | var |
  var := Transcript
  show: each class name;
  show: each printString ].
^ x < y

```

Annotations in the code block include: method name, parameter, pragma, comment, local variable, binary message, unary message, boolean literals, nil literal, block, keyword message, assignment, pseudo variables, instance variable, integer literals, byte array, array generated at runtime, literal array, symbols, character, string, floating point, scaled decimal, local block variable, block parameter, global variable, cascade, keyword message, and return instruction.

other method definition examples:
 unary
 + binaryMessageArgument
 keyword: arg
 keyword: arg1 withTwo: arg2

<https://www.pharo.org>

PLACE
STAMP
HERE

.....

.....

.....

.....

Literals in Smalltalk

Literals in Smalltalk

- **true** and **false**
- **nil**
- Numbers: **42**, **-42**, **4.2**, **4s2**, **4e2**, ...
- Individual characters: **\$a**
- Strings of characters: **'foo'**
- Symbols: **#foo:bar:**
- Arrays of other literal constants: **#(1 2 4)** and **#[255 0]**

What are Magic Literals?

Magic Literals

Literals are called **Magic** if their purpose is not well explained in the source code and is not clear from the context.



Exploring Literals in Pharo



We have identified

169,133

literals in Pharo 7

RQ1: What are the most common data types of literals?

Symbol?

String?

Integer?

Float?

UndefinedObject?

Character?

Array?

Boolean?

Integer

String

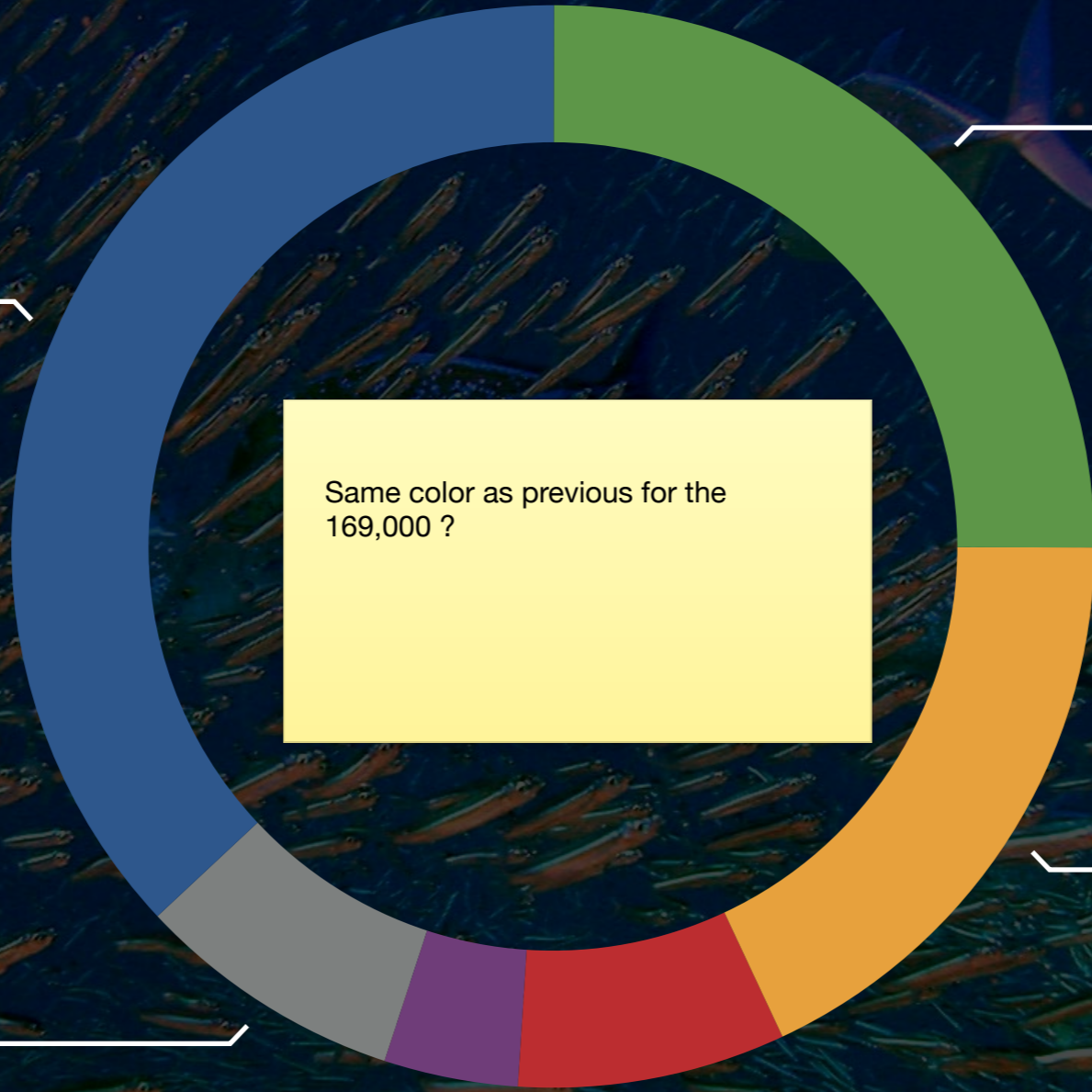
Same color as previous for the 169,000 ?

Symbol

Other

Boolean

Array



**RQ2: In which part of the AST do those
literals appear?**

Assignment node?

Message node?

Return node?

Pragma node?

Argument node?

Sequence node?

Receiver node?

Argument —



Receiver

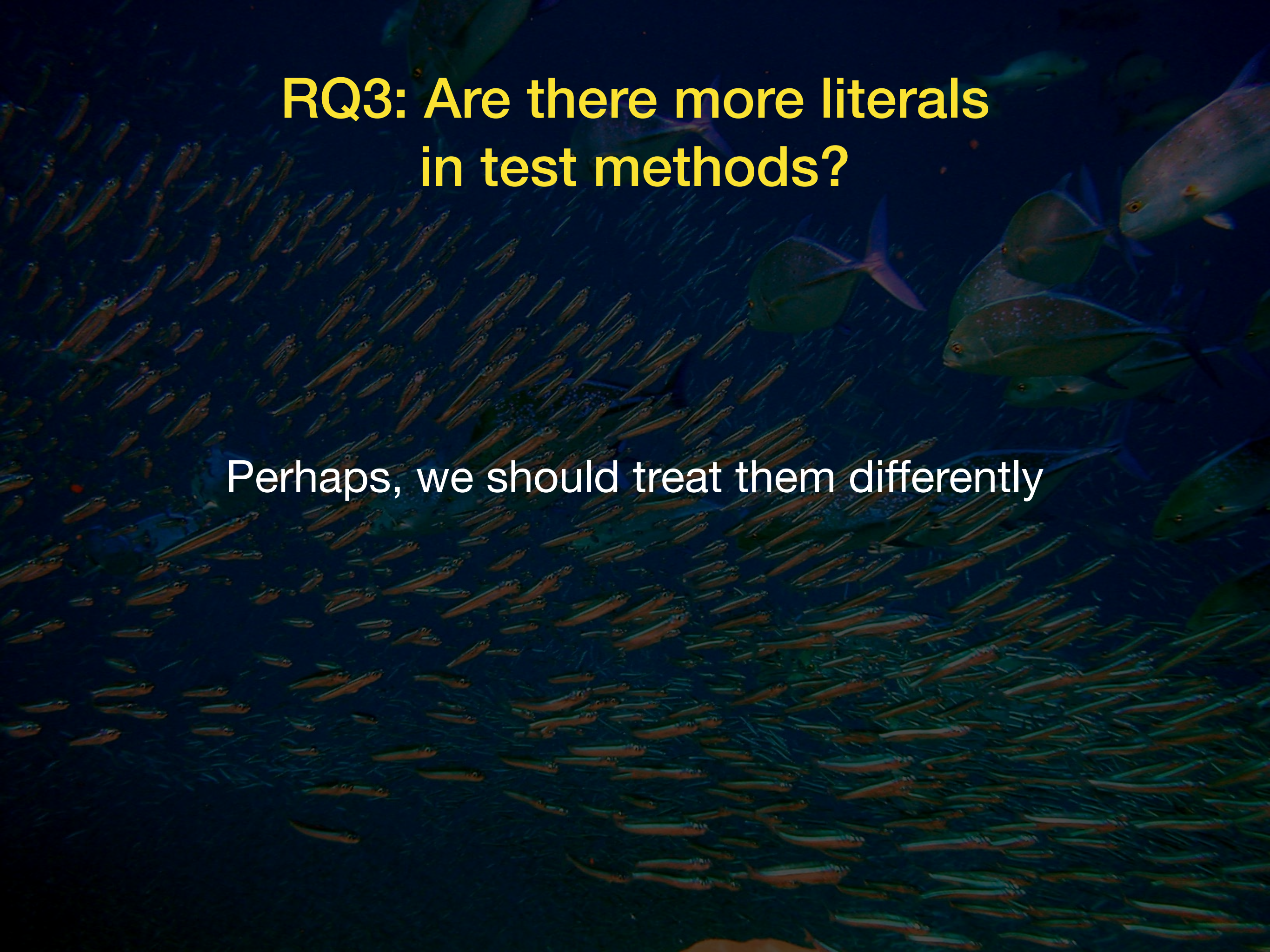
Return

Assignment

Sequence

Pragma

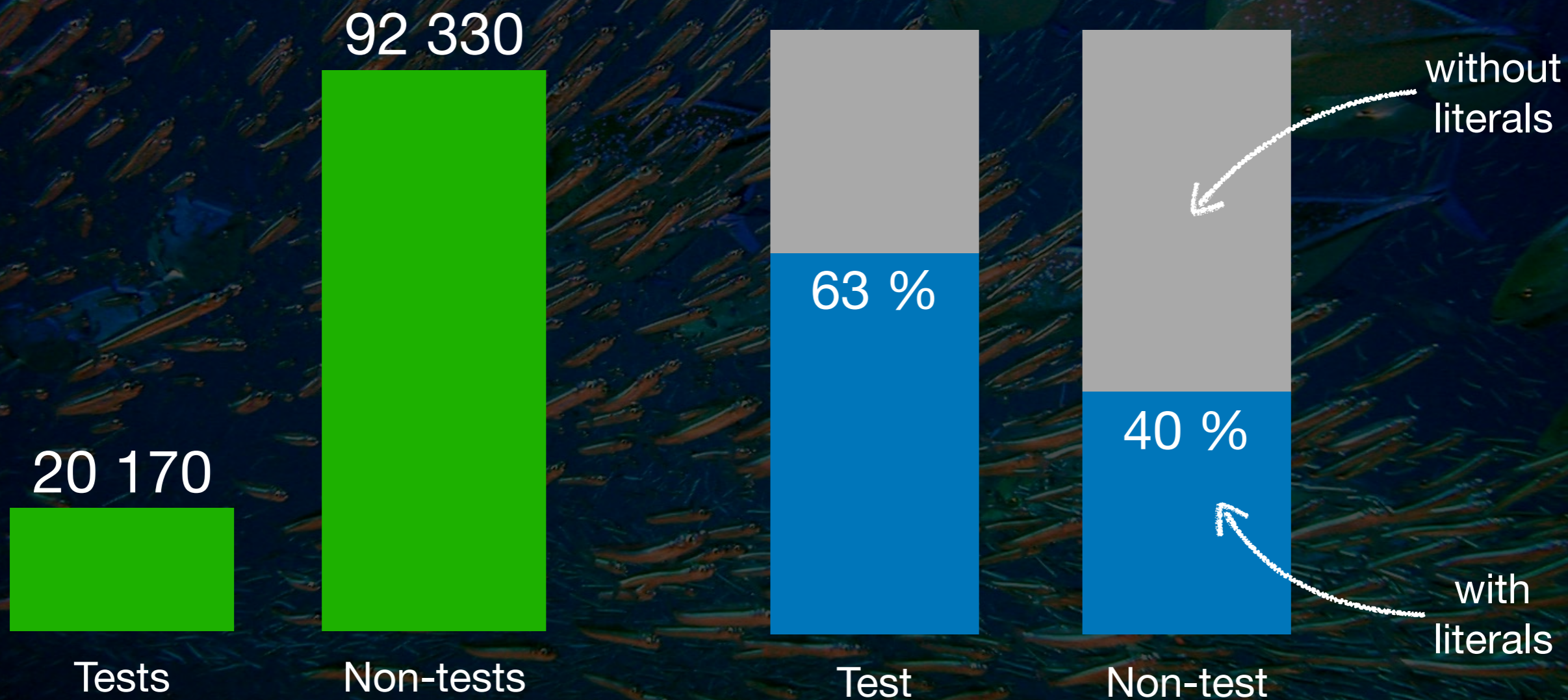
Array

The background of the slide is a photograph of a school of fish in dark blue water. On the left side, there is a dense school of many small, silver fish swimming in a similar direction. On the right side, there are several larger, more prominent fish, possibly groupers, with a mottled pattern on their bodies. The overall scene is underwater and somewhat dimly lit.

RQ3: Are there more literals in test methods?

Perhaps, we should treat them differently

Distribution Over Test and Non-Test



Acceptable Literals



Acceptable literals

Literals self-describing their semantics

- **true**

- **false**

- **nil**

- **#()**

- **#[]**

- **{}**

- **“”**

Acceptable literals

Literals that are part of API

'abcdedfgh' copyFrom: 1 to: 5

Acceptable literals

**Literals directly assigned to a variable /
returned by a method**

EventSensorConstants class » initializeEventTypeConstants

"Types of events"

EventTypeNone := 0.

[...]

JPEGReadWriter class » typicalFileExtensions

"Answer a collection of file extensions (lowercase)
which files that I can read might commonly have"

^#('jpg' 'jpeg')

Acceptable literals

Literals located in a method annotation arguments

Form » gtInspectorFormIn: composite

<gtInspectorPresentationOrder: 90>

^ composite morph

title: **'Morph'**;

display: [self asMorph]

Acceptable literals

Literals located in a test and example methods

Form » testNewWithSize

|array|

array := **Array** new: **5**.

self assert: array size = **5**.

1 to: **5** do: [:index | self assert: (array at: index) isNil]

Magic Literals




Magic Literals

Any literal that does not fall in one of the acceptable literal category.



But... why are they bad?

- 
1. Readability
 2. Logic duplication
 3. Modularity



Detecting Magic Literals

Heuristic implemented as a CodeCritic rule

The screenshot shows an IDE window titled "WorldMorph>>truncatedMenuLabelFor:". The interface is divided into several panes:

- Left Pane:** A tree view showing a hierarchy of packages under "Worlds". The "WorldMorph" package is selected.
- Middle Pane:** A list of classes and methods. "WorldMorph" is selected, and the "instance side" view is active, showing a list of methods including "truncatedMenuLabelFor:".
- Right Pane:** A list of methods. "truncatedMenuLabelFor:" is selected.
- Bottom Pane:** A code editor showing the implementation of the method: `truncatedMenuLabelFor: aWindowLabel` and `^ aWindowLabel truncateWithElipsisTo: 47`. A yellow warning icon is next to the first line. A blue box highlights the code, and white arrows point to the warning icon and the code.

At the bottom of the IDE, there is a status bar with the text "1/27 [1]" and "Utility methods" and "Magic literals" with red 'X' and '?' icons. The page number "26" is visible at the bottom center.

Evaluation



Evaluation Strategy

Step 1. Select all methods from Pharo 7

Step 2. Run heuristic to find magic literals

Step 3. Select a small sample of those methods

Step 4. Manually evaluate the results

Evaluation. Step 1

We have collected

112,500 methods

from Pharo 7 image

Evaluation. Step 2

Our heuristic reported

Without
magic literals

160 147

Methods

With
magic literals

8 986



Evaluation. Step 2

Our heuristic reported

Non-magic

103 514

Magic

23 292

Literals



Evaluation. Step 3

We randomly selected

100 methods

for manual evaluation

Evaluation. Step 3

According to our heuristic,
those methods contain

243

magic
literals

Evaluation. Step 4

True
Positive

?

False
Positive

151

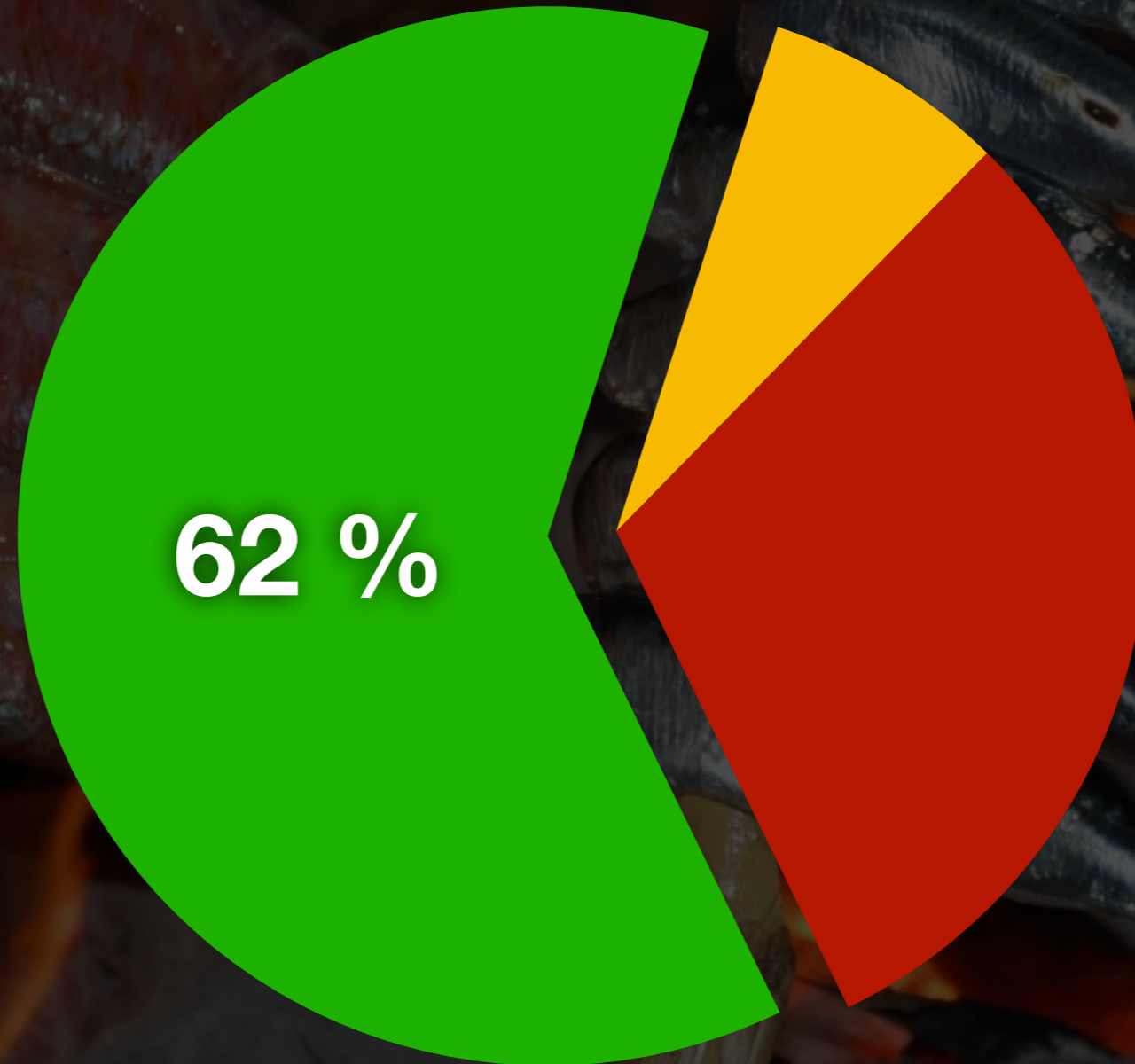
18

74

Same color as previous slide for 243 ?

als
magic

Estimated Precision



Evaluation. Step 4

True Positive example

Socket>>receiveSomeData

"Receive currently available data (if any). Do not wait."

```
| buffer bytesRead |
```

```
buffer := String new: 2000.
```

```
bytesRead := self receiveSomeDataInto: buffer.
```

```
^buffer copyFrom: 1 to: bytesRead
```


Evaluation. Step 4

False Positive example

longAt: index **put:** value **bigEndian:** aBool

"Return a 32bit integer quantity starting from the given byte index"

| b0 b1 b2 b3 |

...

aBool ifTrue:[

self at: index put: b0.

self at: index+1 put: b1.

self at: index+2 put: b2.

self at: index+3 put: b3.

] ifFalse:[

self at: index put: b3.

self at: index+1 put: b2.

self at: index+2 put: b1.

self at: index+3 put: b0.

].

^value

Evaluation. Step 4

Not sure of category example

Put example with number instead

IceGitSshRemote>>httpsUrl

^ **'https://{1}/{2}.git'** format: { self host . self projectPath }

Conclusion



Conclusion

The background image shows a whole roasted piglet on a wooden spit, resting on a burlap placemat. To the left is a small bowl of dipping sauce. To the right is a wooden pepper mill. Several red chilies are scattered around the piglet. The scene is set on a wooden table.

- Exploration of magic literal concept
- Empirical analysis in the context of Pharo
- Implementation of an approach to detect magic literals
- Evaluation of the approach



Future Work

Future Work

- Per project analysis
- Study variety of usages between different domains
- Study the evolution of magic literals across multiple versions of projects
- Improve the heuristic accuracy

Takeaways

- First heuristic for identifying magic literals
- Implemented as code critics (lint) rule
- 62% of reported literals were actually magic (and should be fixed)

Evaluation. Step 4

False Positive example

Put example with number instead

InternetConfiguration >getFTPProxyHost

"Return the FTP proxy host"

"InternetConfiguration getFTPProxyHost"

^self primitiveGetStringKeyedBy: **'FTPProxyHost'**