a first look at

# Strings in Pharo

Damien Pollet — Inria Lille

International Workshop on Smalltalk Technology — ESUG 2015, Brescia

# A First Analysis of String APIs: the Case of Pharo

Damien Pollet  Stéphane Ducasse

RMoD — Inria & Université Lille 1

damien.pollet@inria.fr

## Abstract

Most programming languages natively provide an abstraction of character strings. However, it is difficult to assess the design or the API of a string library. There is no comprehensive analysis of the needed operations and their different variations. There are no real guidelines about the different forces in presence and how they structure the design space of string manipulation. In this article, we harvest and structure a set of criteria to describe a string API. We propose an analysis of the Pharo 4 String library as a first experience on the topic.

***Keywords*** Strings, API, Library, Design, Style

case of st s are particularly hard to re nstraints.

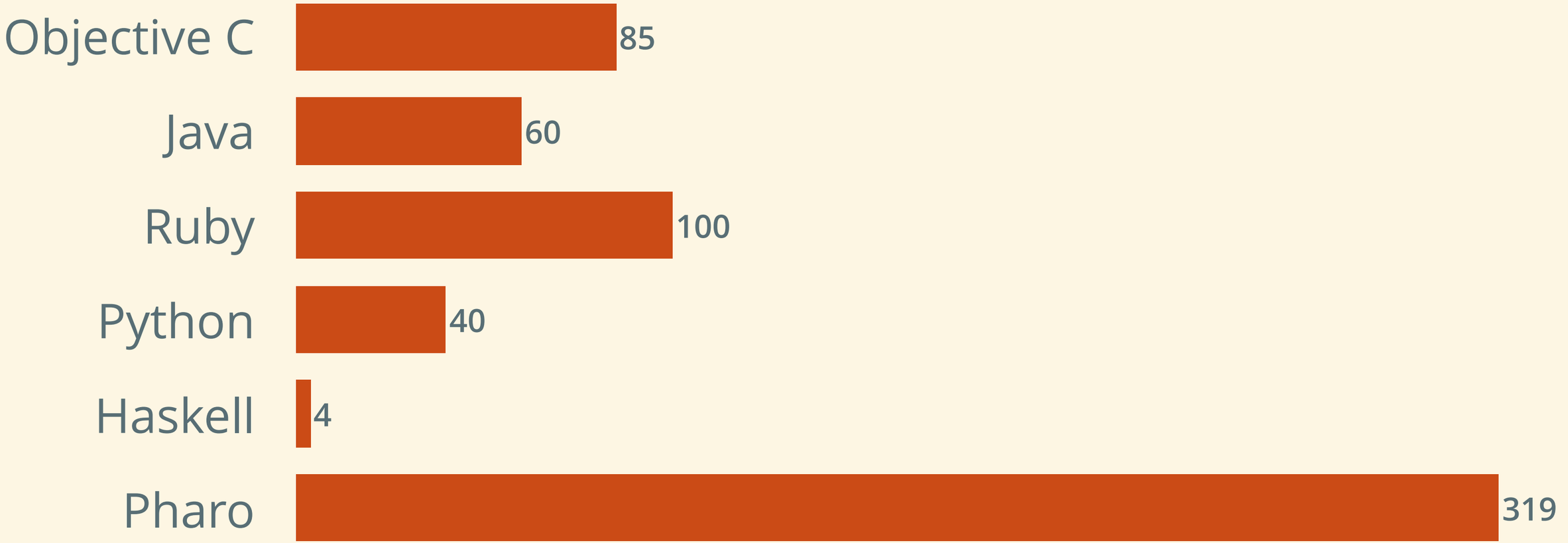
For a ave a large API: in Ruby an 100 methods, in Java n und 40. In Pharo[1], the String c stinct messages, not counting inh a large API is not always a problem *pe* hat strings have many use cases, from concatenation and printing to search-and-replace, parsing, natural or domain-specific languages. Unfortunately, strings are often abused to eschew proper modeling of structured data, resulting in inadequate serialized representations

Using strings feels

***T E D I O U S***...

Why?

# Not enough methods, maybe?



# Concatenation

Objective C
```
[@"Hello" stringByAppendingString: @"_world"]
```

Java
```
"Hello" + "_world"
```

Ruby
```
"Hello" + "_world"
```

Pharo
```
'Hello' , '_world'
```

just to pick on

~~a dynamicallyTypedObjectiveCProgrammingLanguage~~

```
stringByAddingPercentEncodingWithAllowedCharacters:
```

(yes, this is a **single-keyword** message)

# Extraction

Objective C
```
[@"abcdef" substringWithRange: NSMakeRange(2, 4)]
```

Java
```
"abcdef".substring(2, 4)
```

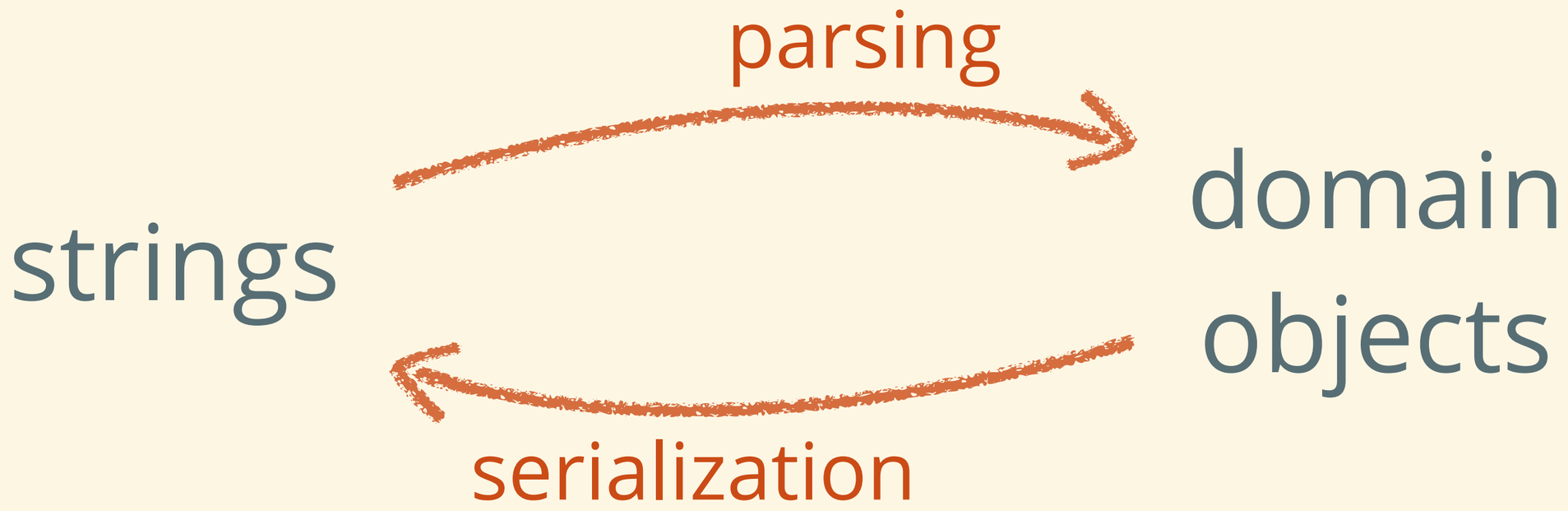
Ruby
```
"abcdef"[2, 4]
```

Pharo
```
'abcdef' copyFrom: 3 to: 5
```


http://www.confidentruby.com

*well, aren't strings just...*

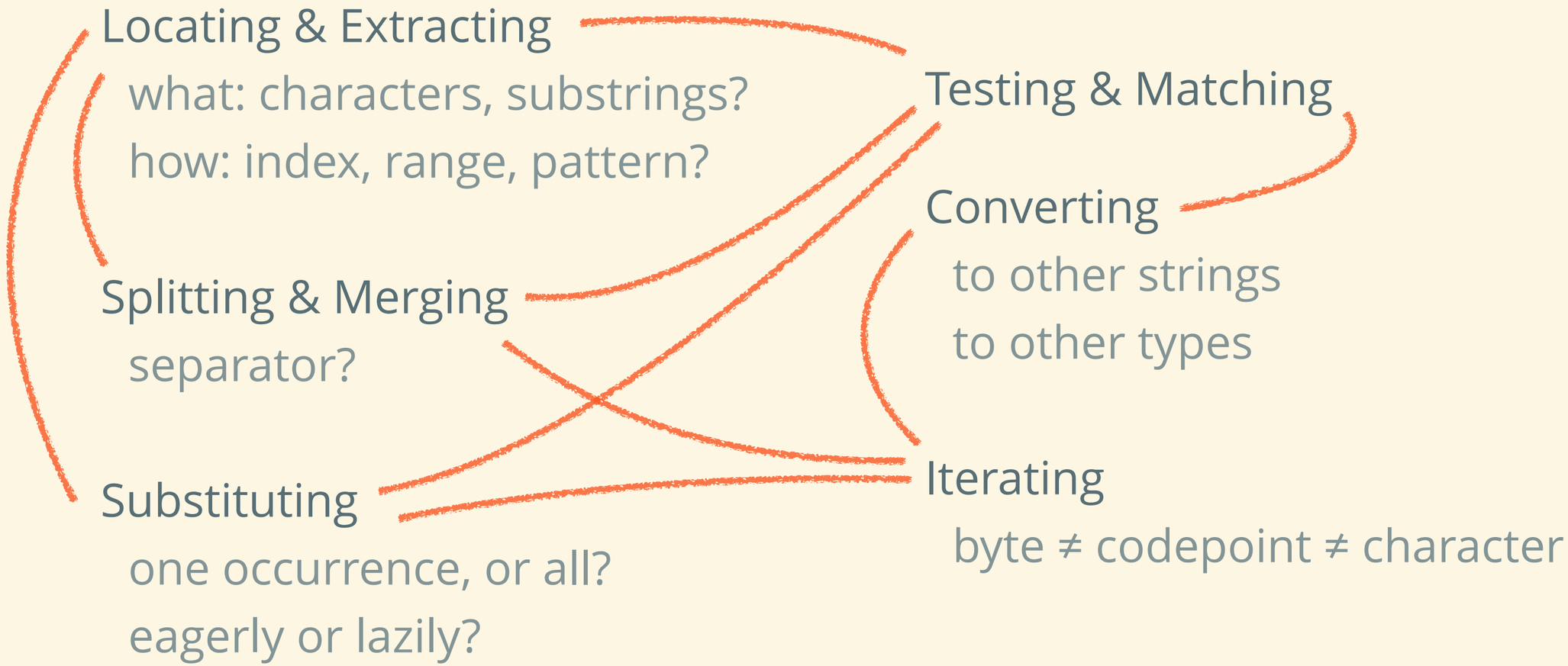
objects?


parsing
strings
domain
objects
serialization

*well, aren't strings just...*

collections?

# Feature overlap



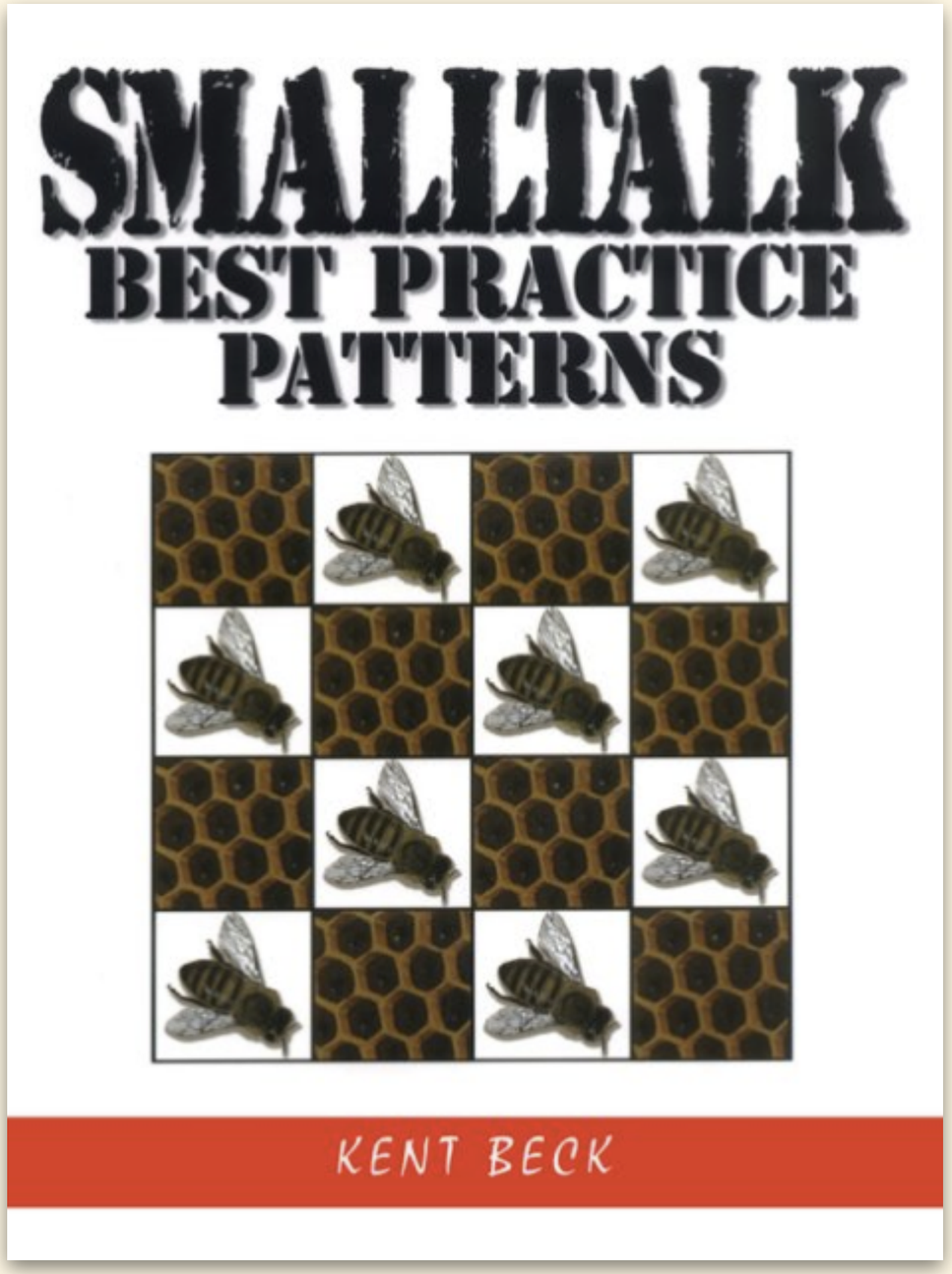
# More than indices

Ruby's indexing operator (square brackets):

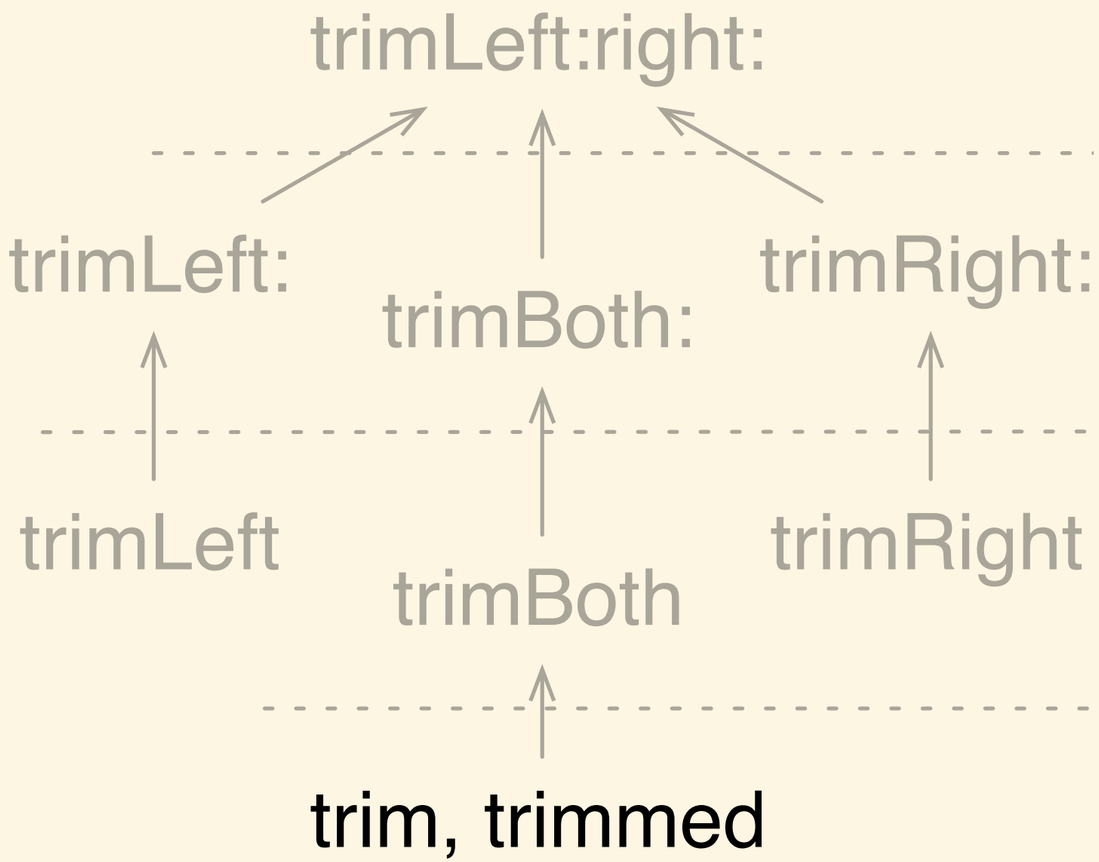
```
my_string[index]
         [-index]
         [from..to]
         [from, length]
         [/reg(exp)+/]
         ['substring']
```

# Idioms

*that I expected to find in…*



# Layers of convenience




QUIZZ!

what's the difference?

# Sentinel values

## Sentinel index

zero?

length + 1

## Depends on use-case...

raise exception, return null object, maybe?

## Pluggable sentinel case

```
indexOf:aCharacter startingAt:index ifAbsent: aBlock
```

# Smells

## Imperative style

indices everywhere — `copyReplaceFrom:to:with:`

## Ad-hoc behavior

`stemAndNumericSuffix` — `endsWithDigit`

## Redundancies

`findAnySubStr:startingAt:` — `findDelimiters:startingAt:`

## Conversion

`asSymbol`, `asInteger`, `asDate` — `asLowercase`, `asHTMLString`

# Mutability

Let's talk about literals:

```
hello
  'hello world' replaceFrom: 7 to: 11 with: 'pharo'.
  ^ 'hello world'

"HelloWho new hello 'hello pharo'"
```

# Where to go from here?

## Idioms more general than strings

how to document & ensure completeness?

lint rules? pragmas? method protocols? *—if only they worked like tags…*

## Improving composability

indices everywhere! imperative style!

iterators, transducers? — rethink collections as well?

## Mutability vs sharing

slices / views, ropes

READ ME!

# A First Analysis of String APIs: the Case of Pharo

Damien Pollet Stéphane Ducasse
RMoD — Inria & Université Lille 1
damien.pollet@inria.fr

## Abstract

Most programming languages natively provide an abstraction of character strings. However, it is difficult to assess the design or the API of a string library. There is no comprehensive analysis of the needed operations and their different variations. There are no real guidelines about the different forces in presence and how they structure the design space of string manipulation. In this article, we harvest and structure a set of criteria to describe a string API. We propose an analysis of the Pharo 4 String library as a first experience on the topic.

case of strings, however, these characteristics are particularly hard to reach, due to the following design constraints.

For a single data type, strings tend to have a large API: in Ruby, the String class provides more than 100 methods, in Java more than 60, and Python's str around 40. In Pharo[1], the String class alone understands 319 distinct messages, not counting inherited methods. While a large API is not always a problem *per se*, it shows that strings have many use cases, from concatenation and printing to search-and-replace, parsing, natural or domain-specific languages. Unfortunately, strings are often abused to eschew proper modeling of struc-