

Opening our brains rethinking our language dogmas

<http://stephane.ducasse.free.fr>



The only way to convince
myself to accept to do this
keynote was to it
unconventional



Revisiting Dynamic Language Infrastructure

La perfection est atteinte, non pas lorsqu'il n'y a plus rien à ajouter, mais lorsqu'il n'y a plus rien à retirer. St-Exupery

Memory object swapper, First class references

Classboxes: Modules for open-classes, OOPAL: OOP + APL
Generalizing message passing

Language symbiosis (Jour. Program)

Encapsulation for dynamic languages, Reusable behavior: Traits

Impacts

Traits used by Perl-6, PHP5.4, Squeak/Pharo, Slate,
Dr-Scheme variants: Fortress (SUN Microsystems),
Scala (EPFL), Multiple type systems

Pharo <http://www.pharo-project.org>

Objective 3: Ecosystem around Pharo



- Pure object-oriented language
- Fully reflective with an agile and powerful IDE
- Running on mac, linux, windows, android, iOS
- Active and growing community: 40 active committers
- 30 or more companies
- 2 releases/year, 100 fixes/month
- Couple hundred of libraries
- Actively supported by Inria

compiler, core classes (stream, collections, unicode), IDE (editor, inspector, debugger, code versioning,...), Versioning model, code model, UI frameworks (widgets, theme), FFI, JIT and on the fly assembly generation, Graphics (soon opengl), Object serializer, Network, HTTP, Proxy, Logging, Web frameworks: seaside, iliad, HTTP2 XML, HTML scraping, Zodiac, HTTPS, WebSocket, Graphical frameworks: Roassal, Mondrian, EyeSee, Athens, Tool builder: Glamour, MetaTools: Moose, Databases: DBXTalk, Mongo, Riak, CouchDB, Parser: Petit Parser, SmaCC, Units (Aconcagua, Units),

You are young

You are smart

...probably :)

but sometimes you are old
inside

Teaching for years,
I saw so many syntactic
people

{ }; versus ()

end. vs)

is it not the same?

a.max(b)

vs.

a max: b

Java is not close to C++
even if they both have {}
syntax

Java is closer to Smalltalk
than C++

Java is a Smalltalk with type
problems

Do not confuse form and
contents

This is not because this is
saved in XML that you will
be able to do anything with
it!

<?xml version="1.0"?>

<foo>

16089151

</foo>

What is the impact of our
education on our choices?

Why can't we work on
something else than
mainstream?

Is there any interest in all of us working with the same constraints?

Between 98 and 2002
we could only work on Java

“Language engineering in post Java era” workshop was the first revolt against this mindset

Javascript/scala will be the
new dogma :)

until the next one :)

Some Examples

- ✦ Traits
- ✦ Path execution based conditional halt
- ✦ Seaside
- ✦ Smalltalk in a nutshell

The story behind Traits

- ✦ Developed in Smalltalk
- ✦ Now in Php 5.4, Perl.6, Squeak, Pharo
- ✦ Somehow Scala, Fortress

Designing traits with Java
would have led to just
another type system

Units of *composable* *behavior*

multiple *implementation* inheritance

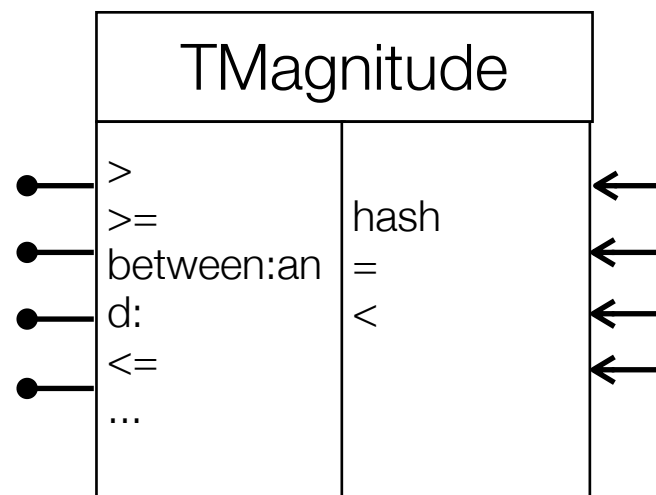
composer is in *control*

resolve conflicts via ignore / alias

backward compatible

Traits are parameterized behaviors:

- **provide** a set of methods
- **require** a set of methods
- purely behavioral (no state)



Class

=

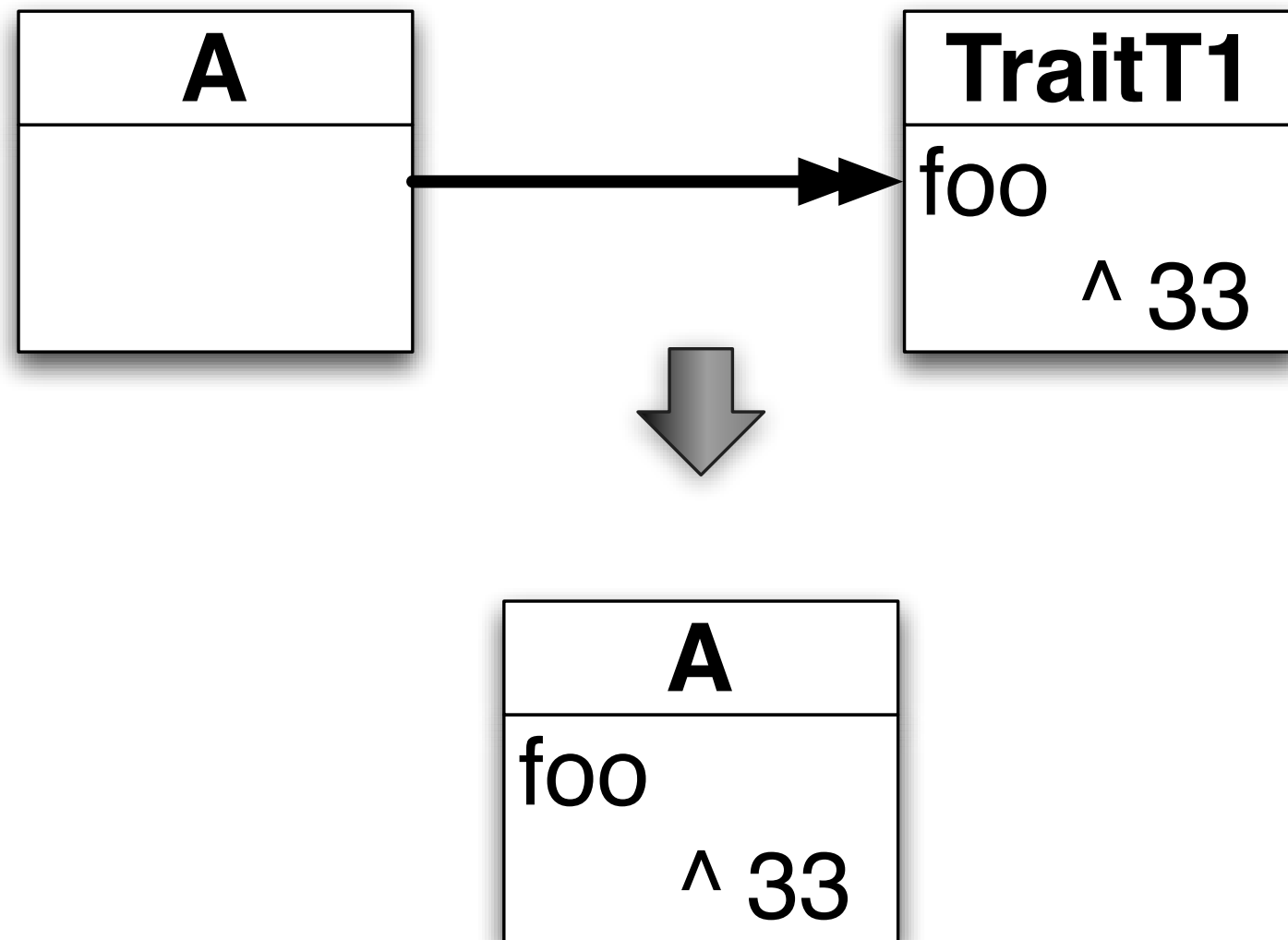
Superclass

+ State

+ Traits

+ Methods

Using T I



Binary method problems

T1 defined and applied on Point, Book....
so as what is the type of this?

We need an anchor based type system

Path based conditional halt

Would be good if we could say:

“Stop method bar ***only if you are*** invoked it from testBar”

bar

...

self haltIf: #testBar...

In 5 lines

```
Object>>haltIf: aSelector
```

```
    | cntxt |
```

```
    cntxt := thisContext.
```

```
    [cntxt sender isNil] whileFalse: [
```

```
        cntxt := cntxt sender.
```

```
        (cntxt selector = aSelector)
```

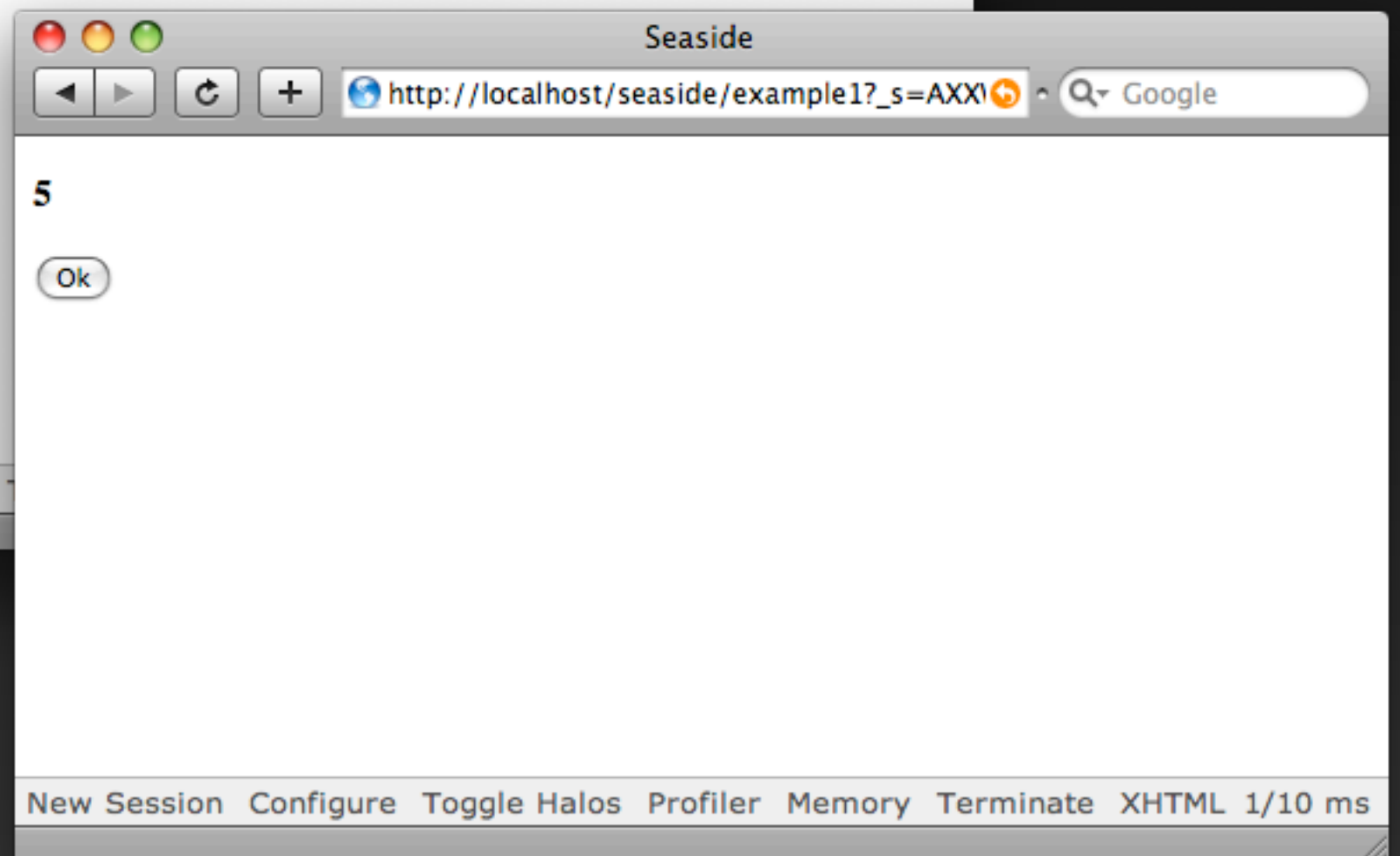
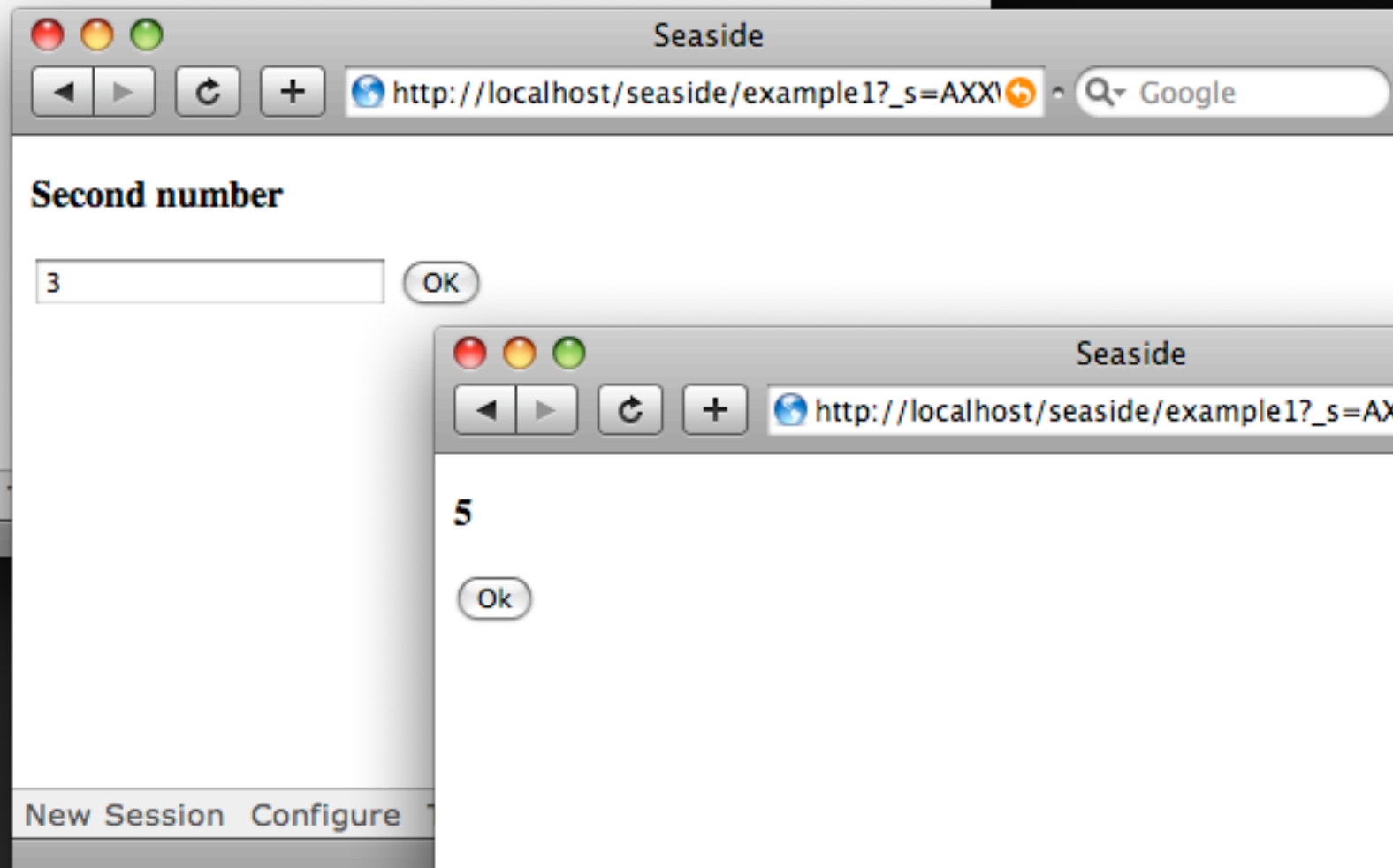
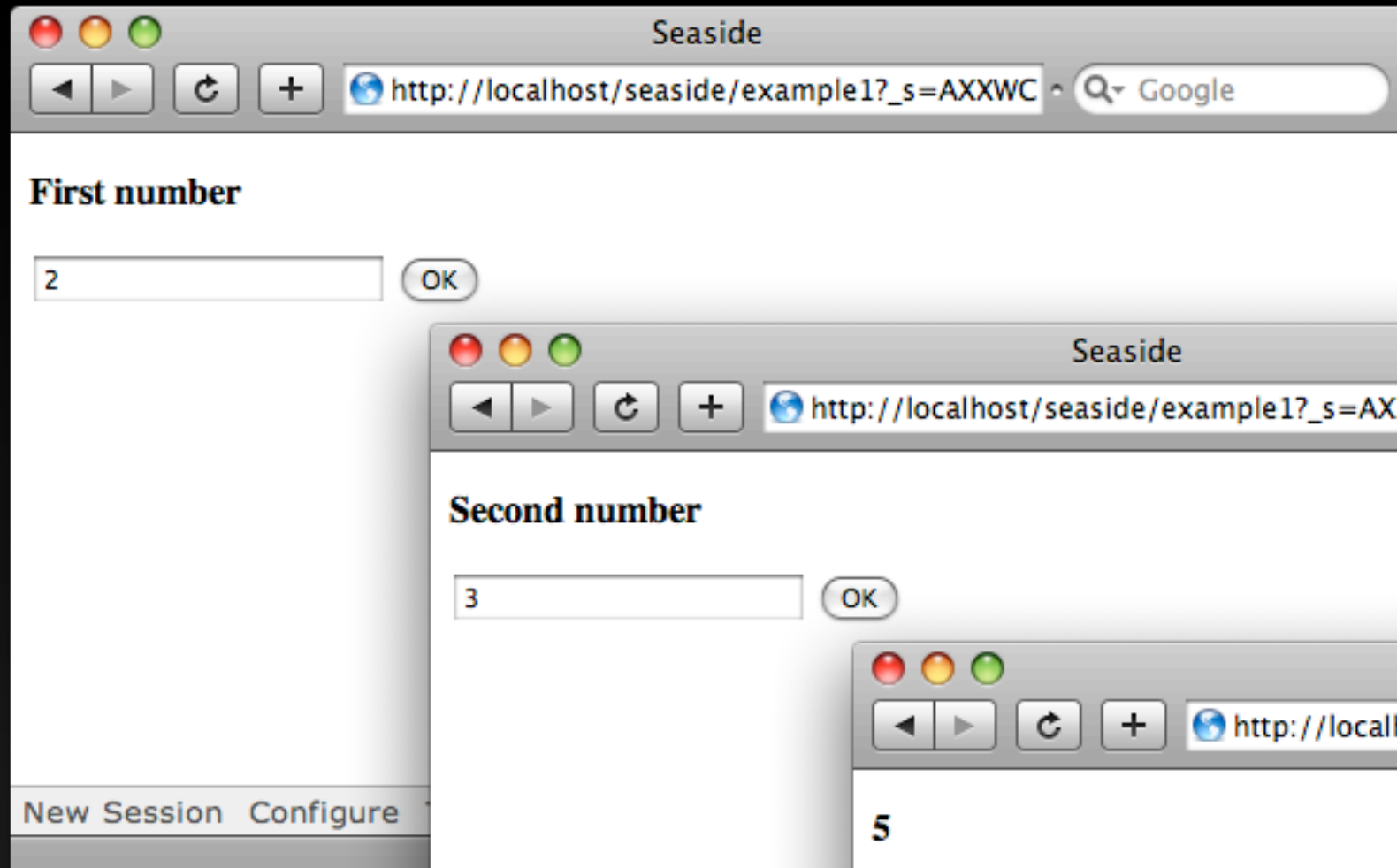
```
            ifTrue: [ Halt signal].
```

```
    ].
```


Seaside

- ✧ HTTP considering harmful
- ✧ Web framework for the heretics
- ✧ Invented by Avi Briant and J. Fitzell
- ✧ Enhanced by L. Renggli, P. Marshall, J. Fitzell
- ✧ <http://www.seaside.st>
- ✧ <http://book.seaside.st>

Natural Flow




```
<form action="second.html">  
  <input type="text" name="value1">  
  <input type="submit" value="OK">  
</form>
```



```
<form action="second.html">  
  <input type="text" name="value1">  
  <input type="submit" value="OK">  
</form>
```

```
<form action="result.html">  
  <input type="text" name="value2">  
  <input type="submit" value="OK">  
</form>
```



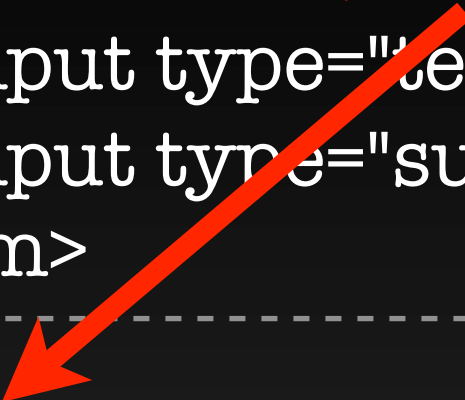
```
<form action="second.html">  
  <input type="text" name="value1">  
  <input type="submit" value="OK">  
</form>
```

```
<form action="result.html">  
  <input type="text" name="value2">  
  <input type="submit" value="OK">  
</form>
```

```
<p>  
  <% value1 + value2 %>  
</p>
```



```
<form action="second.html">  
  <input type="text" name="value1">  
  <input type="submit" value="OK">  
</form>
```

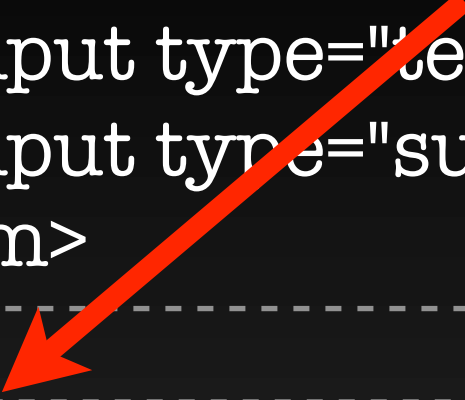


```
<form action="result.html">  
  <input type="text" name="value2">  
  <input type="submit" value="OK">  
</form>
```

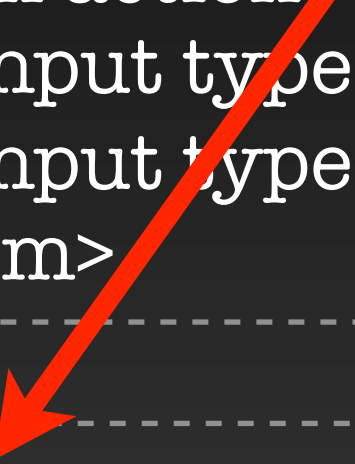
```
<p>  
  <% value1 + value2 %>  
</p>
```



```
<form action="second.html">
  <input type="text" name="value1">
  <input type="submit" value="OK">
</form>
```



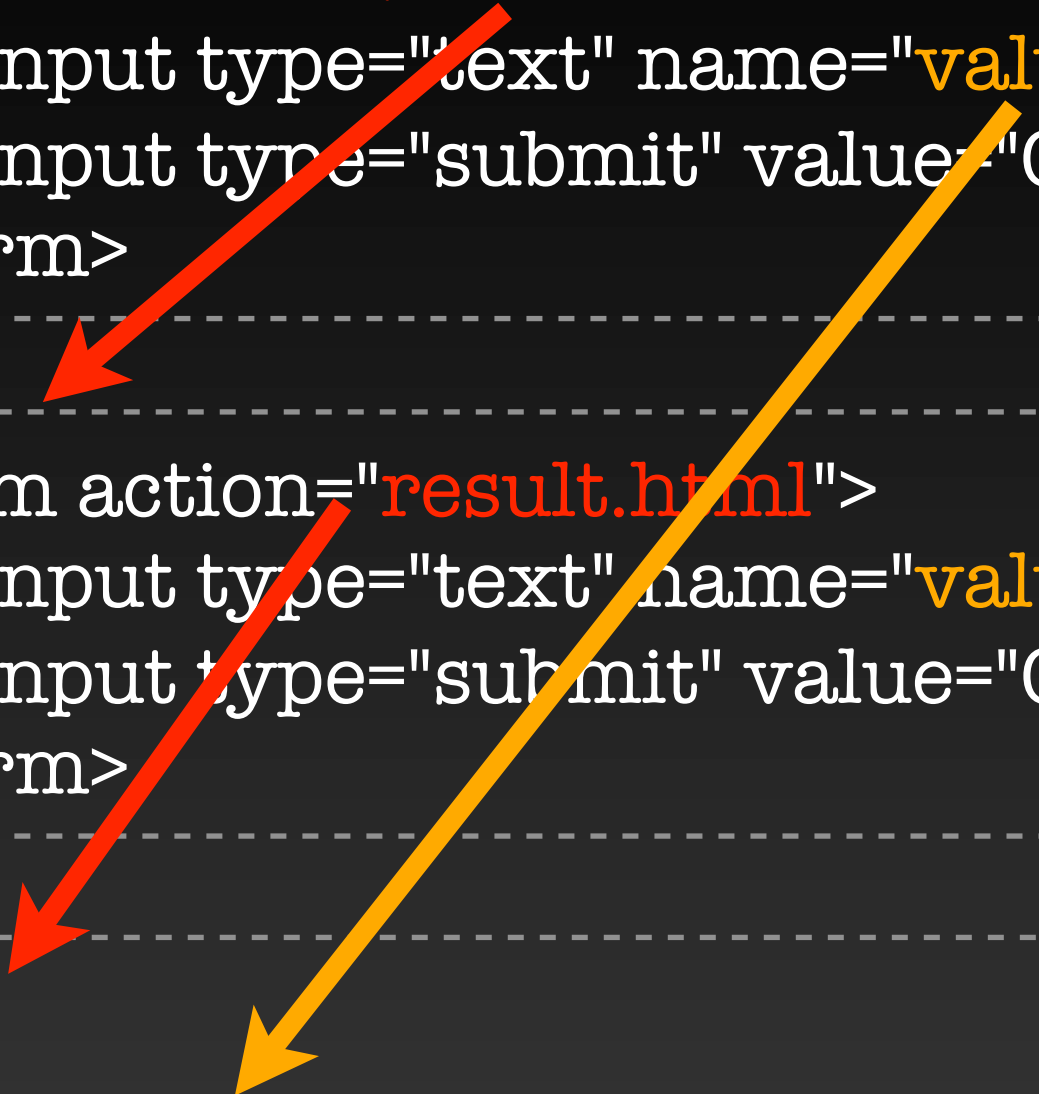
```
<form action="result.html">
  <input type="text" name="value2">
  <input type="submit" value="OK">
</form>
```



```
<p>
  <% value1 + value2 %>
</p>
```



```
<form action="second.html">  
  <input type="text" name="value1">  
  <input type="submit" value="OK">  
</form>
```



A diagram illustrating data flow between three HTML blocks. A red arrow originates from the 'value1' attribute in the first form and points to the first parameter in the second form's action attribute. A yellow arrow originates from the 'value1' attribute in the first form and points to the 'value1' variable in the output paragraph. Another red arrow originates from the 'value2' attribute in the second form and points to the second parameter in the second form's action attribute. A yellow arrow originates from the 'value2' attribute in the second form and points to the 'value2' variable in the output paragraph.

```
<form action="result.html">  
  <input type="text" name="value2">  
  <input type="submit" value="OK">  
</form>
```

```
<p>  
  <% value1 + value2 %>  
</p>
```



```
<form action="second.html">  
  <input type="text" name="value1">  
  <input type="submit" value="OK">  
</form>
```

```
<form action="result.html">  
  <input type="hidden" name="value1" value="<% value1 %>">  
  <input type="text" name="value2">  
  <input type="submit" value="OK">  
</form>
```

```
<p>  
  <% value1 + value2 %>  
</p>
```



```
<form action="second.html">  
  <input type="text" name="value1">  
  <input type="submit" value="OK">  
</form>
```

```
<form action="result.html">  
  <input type="hidden" name="value1" value="<% value1 %>">  
  <input type="text" name="value2">  
  <input type="submit" value="OK">  
</form>
```

```
<p>  
  <% value1 + value2 %>  
</p>
```


Who cares about
HTTP anyway?





seaside 

is different

Concentrate on
your application ...

... no manual
request parsing

... no XML
configuration files

3 user interactions

3 lines of code


```
value1 := self request: 'First Number'.
```


value1 := self request: 'First Number'.

value2 := self request: 'Second Number'.


```
value1 := self request: 'First Number'.
```

```
value2 := self request: 'Second Number'.
```

```
self inform: value1 + value2.
```


x := **A** call: **B**

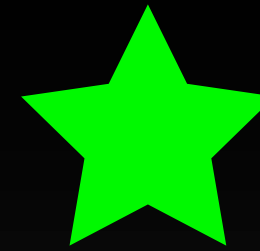
A

x := A call: B






answer:





answer:



X **:=** 



Challenges

- ★ Hide continuations from developers
- ★ Combine objects and continuations
- ★ Multiple control flow on a single page
- ★ Optimize the use of continuations

Fuel

- ✧ Fast object serialization
- ✧ Is part of the PhD of M. Martinez-Peck
- ✧ Can serialize everything: objects (integer, true...)
- ✧ classes (more challenging) methods.
- ✧ stackframes!!!

Now people
serialize a
debugger stack
and load it in
another system :)

Simplicity and purity

Smalltalk OO model

- ✧ ***Everything*** is an object
 - ✧ Only message passing
 - ✧ Only late binding
- ✧ Instance variables are private to the object
- ✧ Methods are public
- ✧ Everything is a pointer
- ✧ Garbage collector
- ✧ Single inheritance between classes

All the syntax on a postcard

example WithNumber: x

“A method that illustrates every part of Smalltalk method syntax except primitives. It has unary, binary, and key word messages, declares arguments and temporaries (but not block temporaries), accesses a global variable (but not an instance variable), uses literals (array, character, symbol, string, integer, float), uses the pseudo variable true false, nil, self, and super, and has sequence, assignment, return and cascade. It has both zero argument and one argument blocks. It doesn't do anything useful, though”

|y|

true & false not & (nil isNil) ifFalse: [self halt].

y := self size + super size.

#\$a #a 'a' 1 1.0)

do: [:each | Transcript

show: (each class name);

show: (each printString);

show: ' '].

^ x < y

Messages and Composition

Three kinds of messages

Unary: Node new, Point new, Browser open

Binary: 1 + 2, 3@4

Keywords: aTomagoshi eat: #cooky furiously: true

Message Priority

(Msg) > unary > binary > keywords

Same Level from left to right

Examples

- (10@0 extent: 10@100) bottomRight

s isNil

ifTrue: [self halt]

```
 #(1 2 3 4 5 6) select: [:each | each odd]  
> #(1 3 5)
```


Numbers...

- ✦ 1000 factorial / 999 factorial

Yes ifTrue: is just a message

Weather isRaining

ifTrue: [self takeMyUmbrella]

ifFalse: [self takeMySunglasses]

ifTrue:ifFalse is sent to an object: a boolean!

Of course the compiler optimize it like a mad :)

Yes a l

```
 #(1 2 -4 -86)
```

```
  do: [:each | Transcript show: each abs printString ;cr ]
```

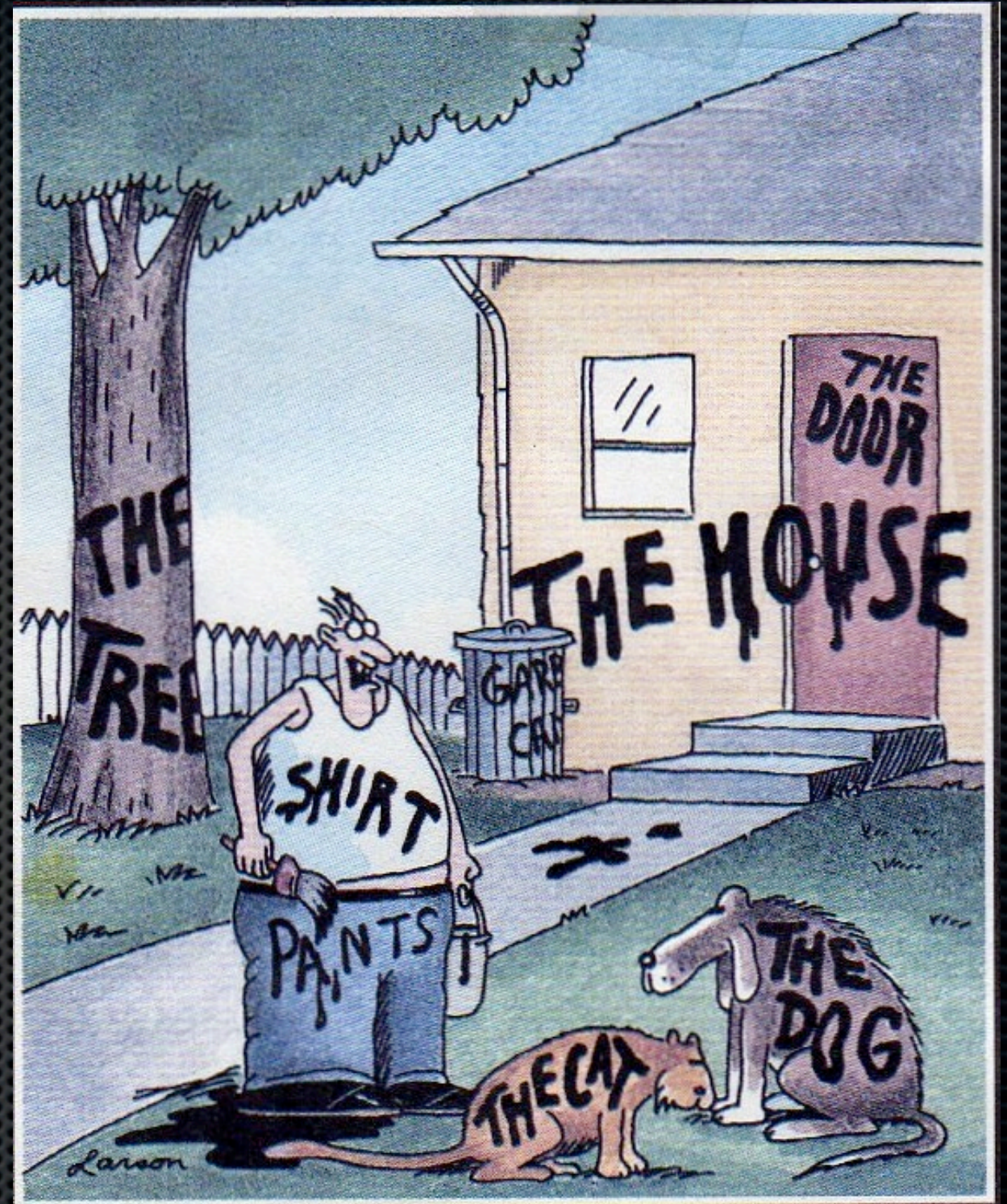
```
> 1
```

```
> 2
```

```
> 4
```

```
> 86
```

Yes we ask the collection object to perform the loop on itself!



"Now! ... That should clear up
a few things around here!"

<http://stephane.ducasse.free.fr>