```
foo :: Ord a => [a] -> [a]
foo []     = []
foo (p:xs) =
    (foo lesser)
    ++ [p]
    ++ (foo greater)
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

① cave; de

tiger's lair/ 匪～

acupuncture point; acu

【穴居】 xuejū live in ca

【穴位】 xuéwèi 〈中医〉 a

学* xué ① study; lear

painting/ 把新技术～到手

会了吗？ Have you le

老。Keep on l

mimic;

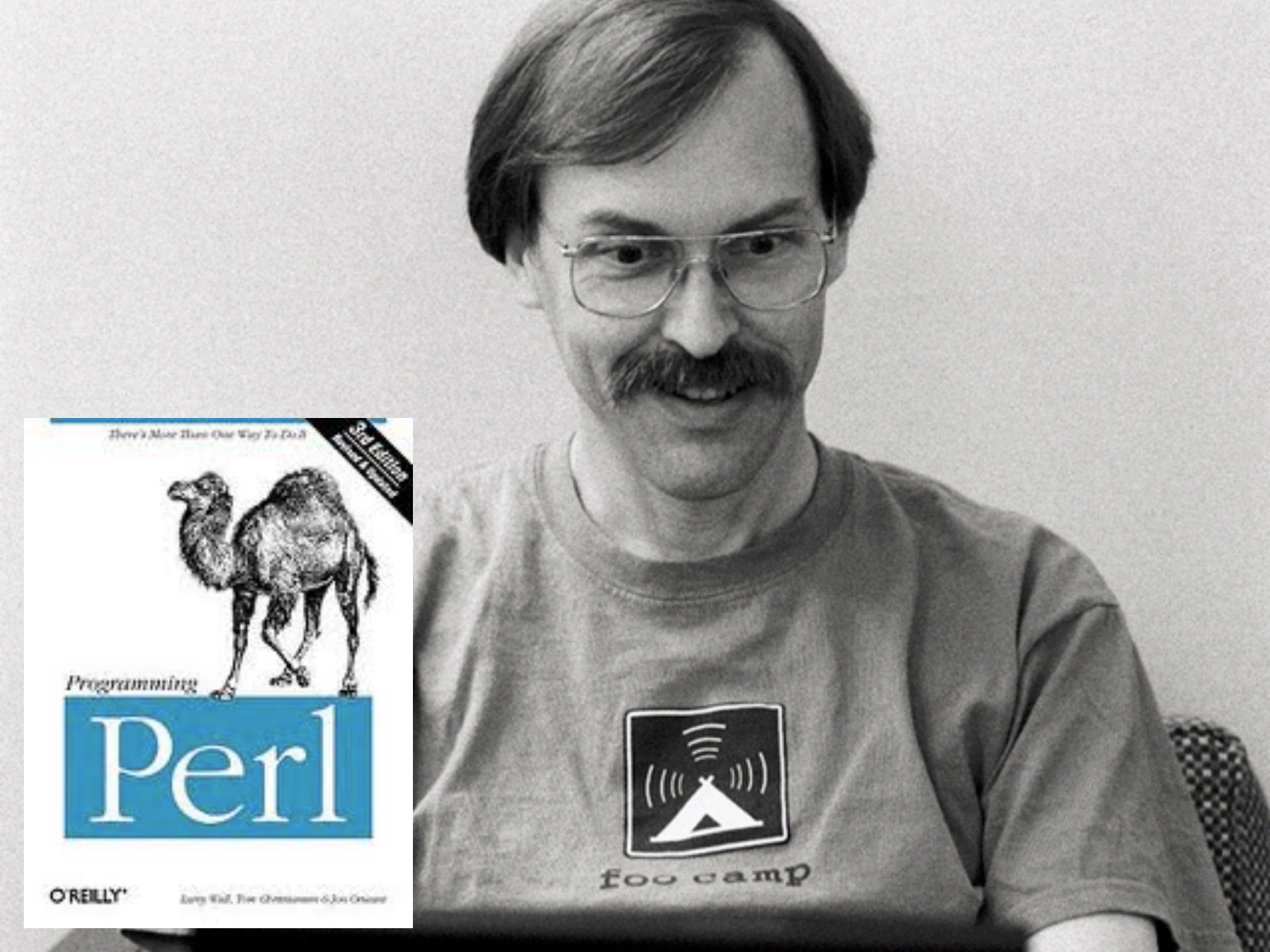
fath

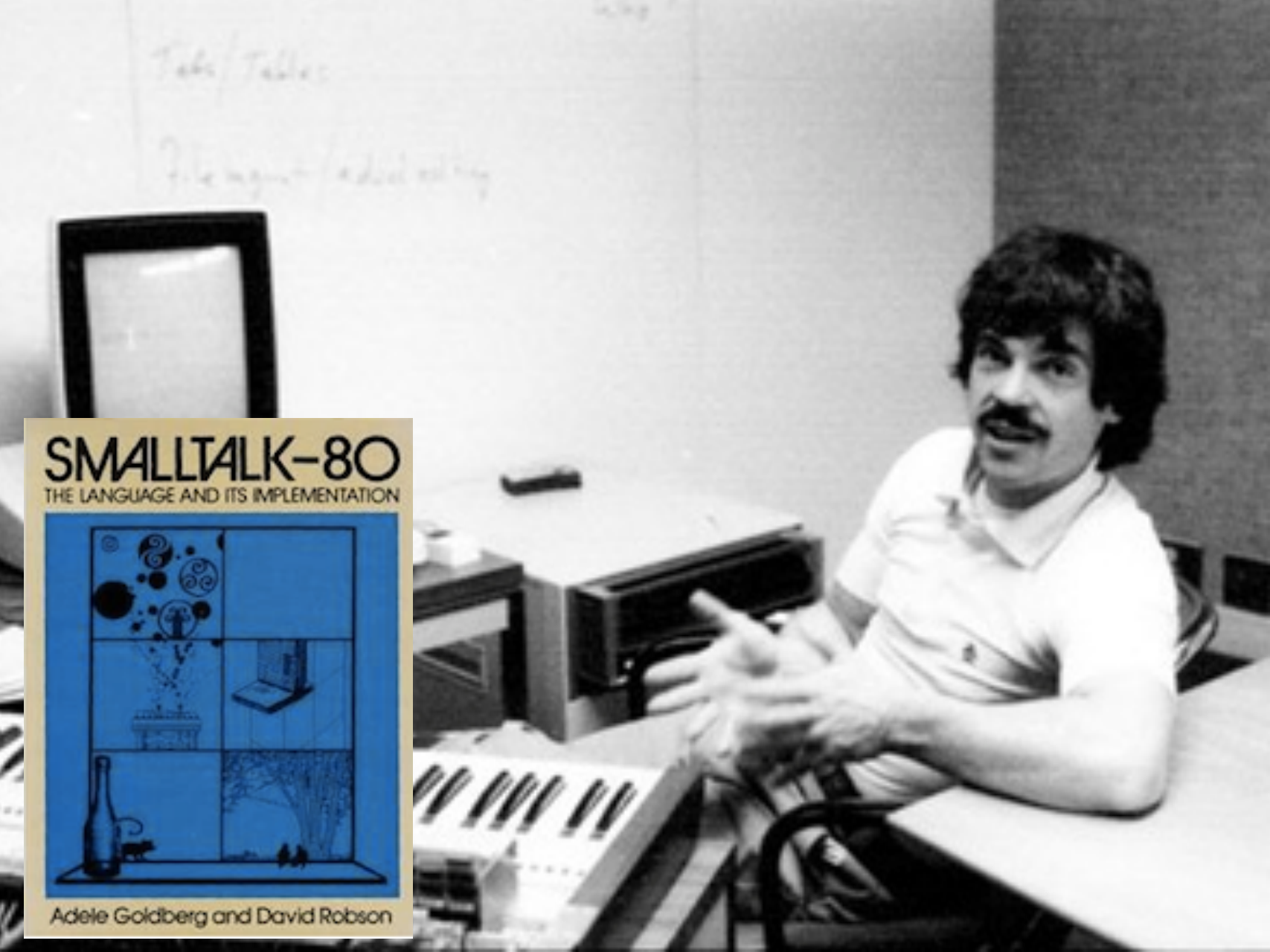
Ruby is a language designed in the following steps:

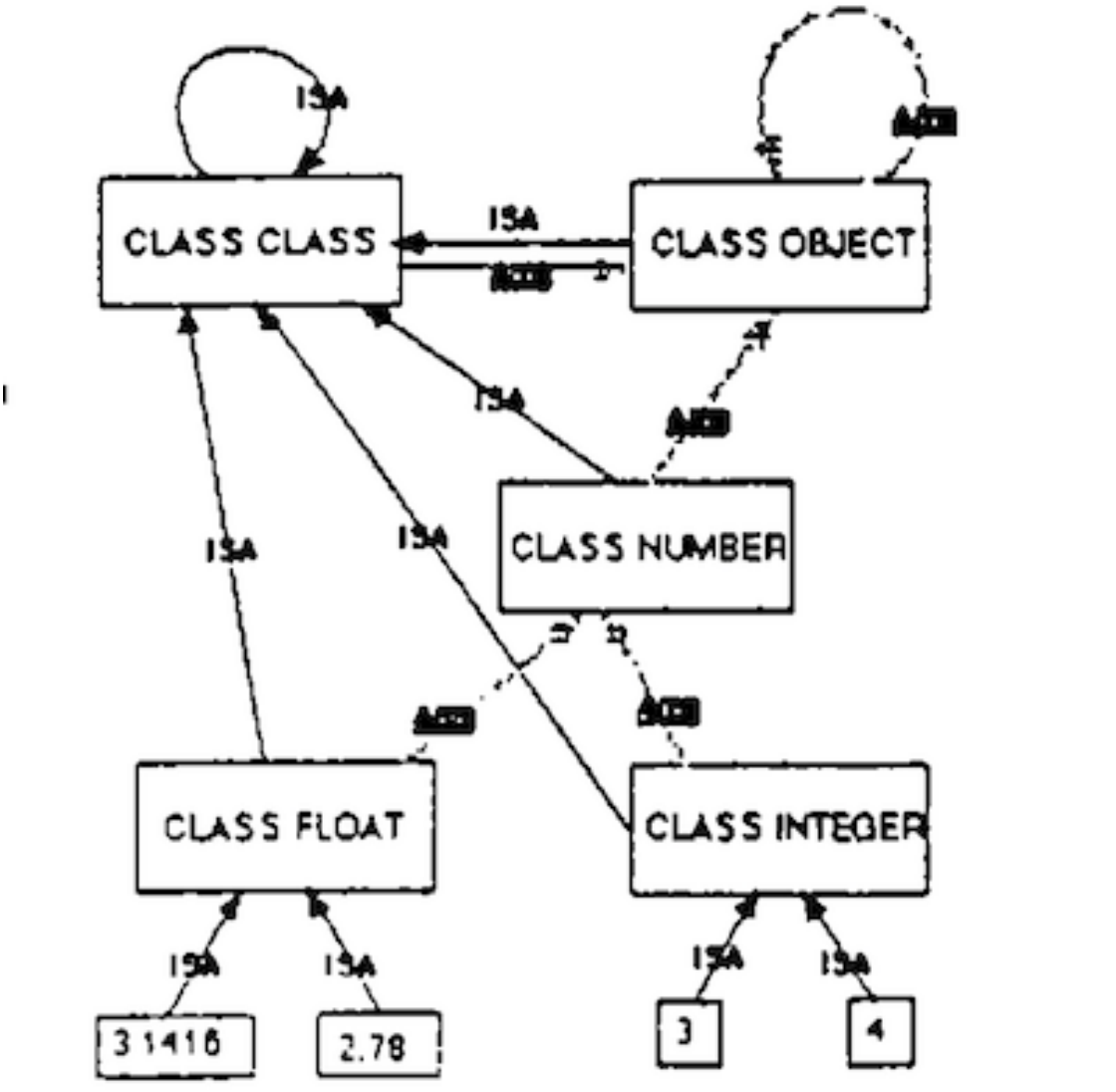
* take a simple lisp language
* add blocks, inspired by higher order functions
* add methods found in Smalltalk
* add functionality found in Perl

So, Ruby was a Lisp originally, in theory. Let's call it MatzLisp from now on. ;-)

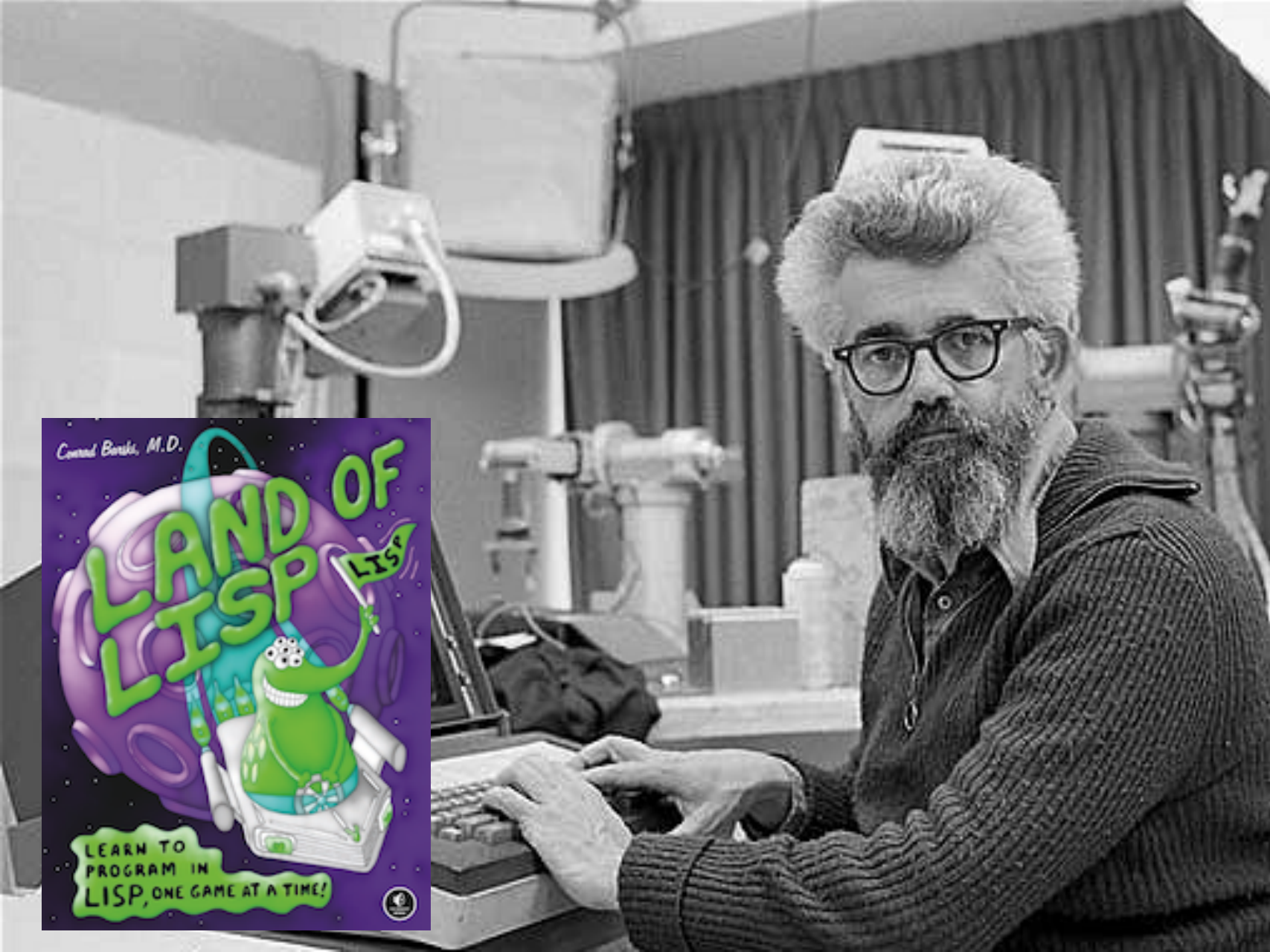
matz.


There's More Than One Way To Do It
3rd Edition
Programming
Perl
O'REILLY
foo camp


SMALLTALK-80
THE LANGUAGE AND ITS IMPLEMENTATION
Adele Goldberg and David Robson



Smalltalk-76 Metaphysics


Conrad Barski, M.D.
LAND OF LISP
LISP
LEARN TO PROGRAM IN LISP, ONE GAME AT A TIME!

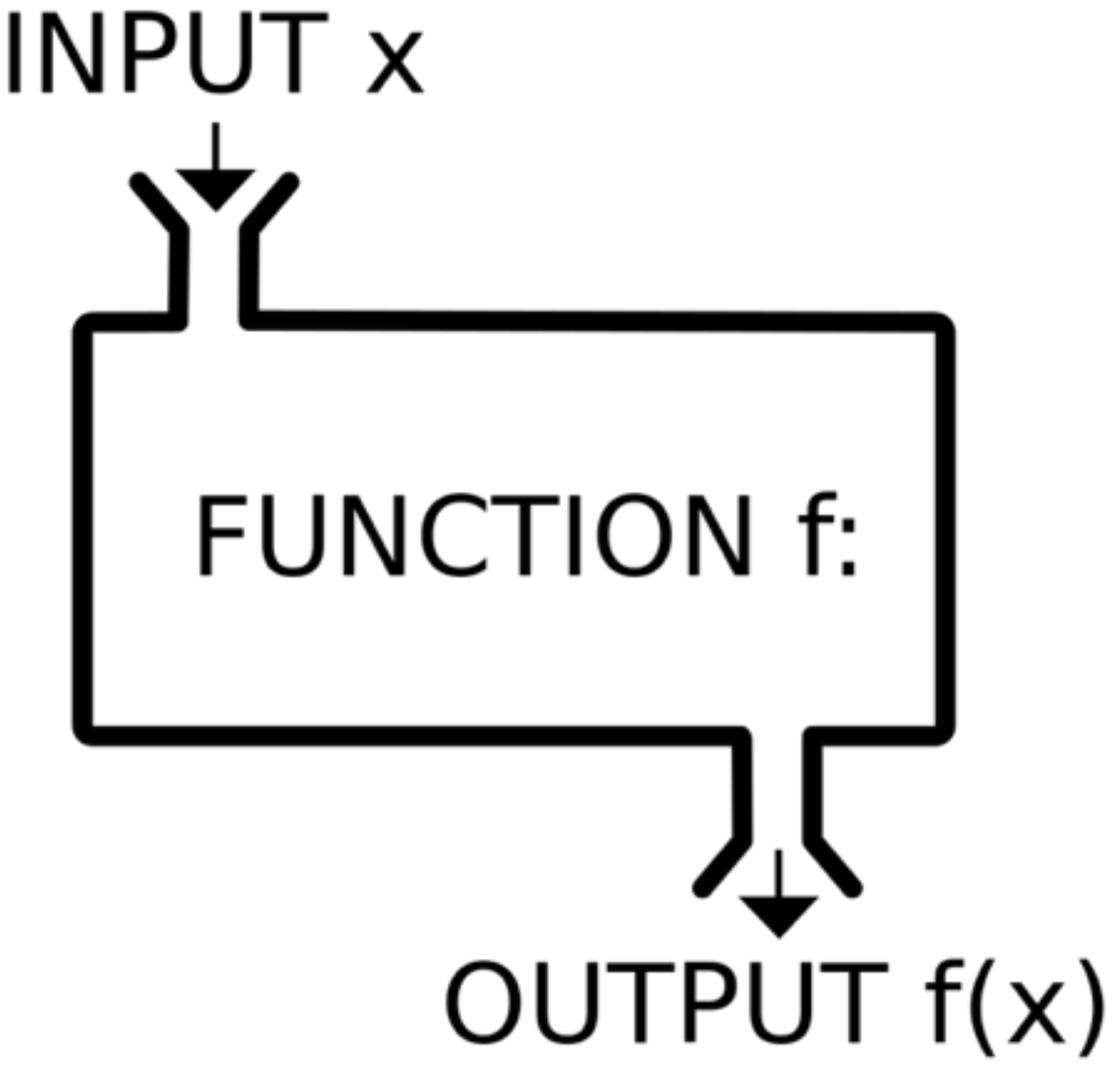
Haskell... is a *polymorphically statically typed*, *lazy*, *purely functional* language, quite different from most other programming languages. The language is named for Haskell Brooks Curry, ...

- what is “functional programming?”

- higher order functions

- lazy evaluation

- memoization


INPUT x
FUNCTION f:
OUTPUT f(x)

higher order functions

```
[1..10]
```

```
(1..10).to_a
```

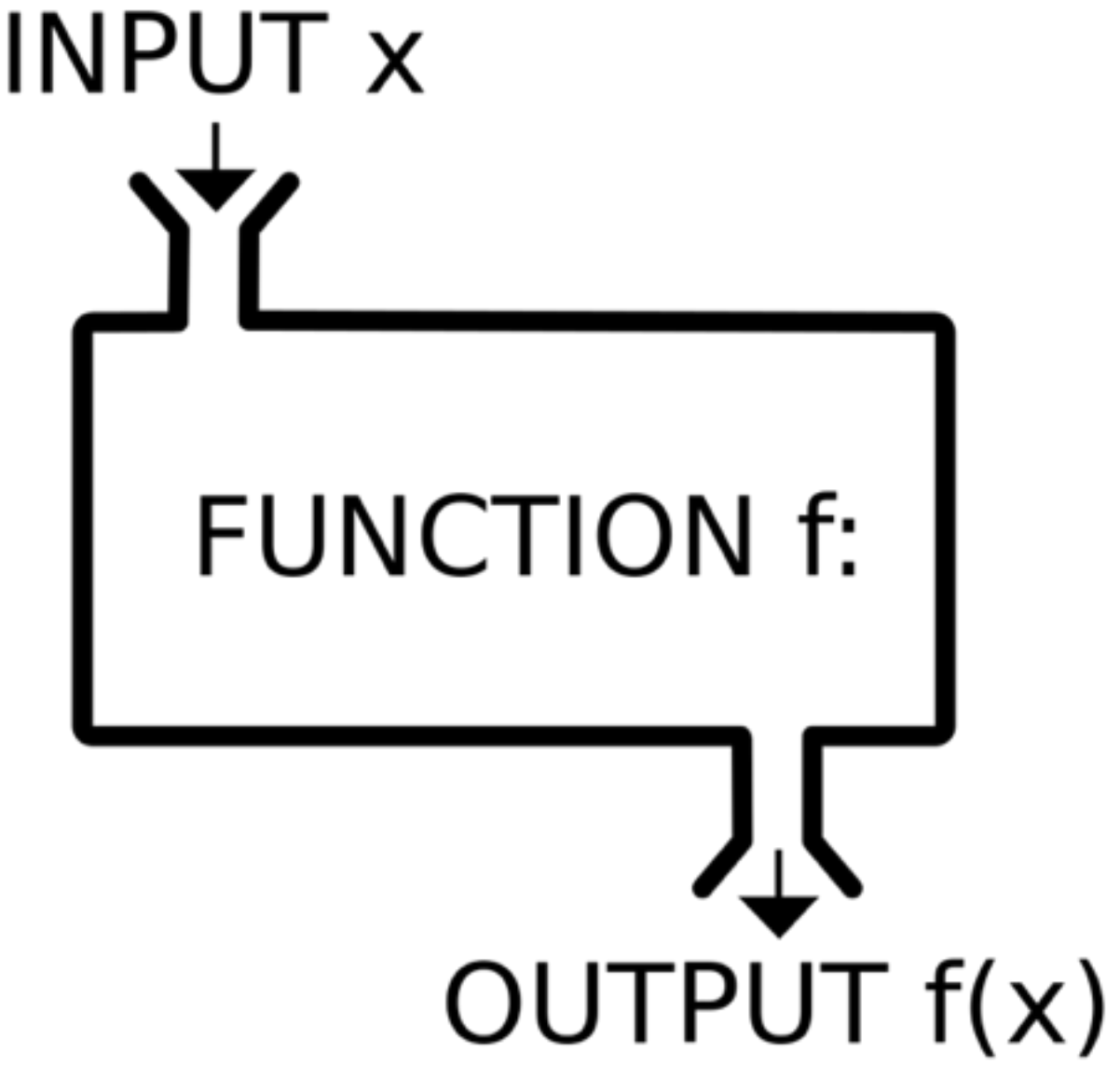
```
=>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
[ x*x | x <- [1..10]]
```

```
(1..10).collect { |x| x*x }
```

```
(1..10).map { |x| x*x }
```

```
=>[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```


INPUT x
FUNCTION f:
OUTPUT f(x)

```
map (\x -> x*x) [1..10]
```

```
(1..10).map &lambda { |x| x*x }
```

```
(1..10).map &(->(x) { x*x })
```

```
=>[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

lazy evaluation

```
[1..]
```

```
=>[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,
29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,
55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,
81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,1
05,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,

etc...
```

```
take 10 [1..]
```

```
=>[1,2,3,4,5,6,7,8,9,10]
```

```
take 10
   [ x+1 | x <-
      [ x*x | x <- [1..]]]
```

```
=>[2,5,10,17,26,37,50,65,82,101]
```

```
(1..Float::INFINITY)
   .lazy
   .collect { |x| x*x }
   .collect { |x| x+1 }
   .take(10).force
```

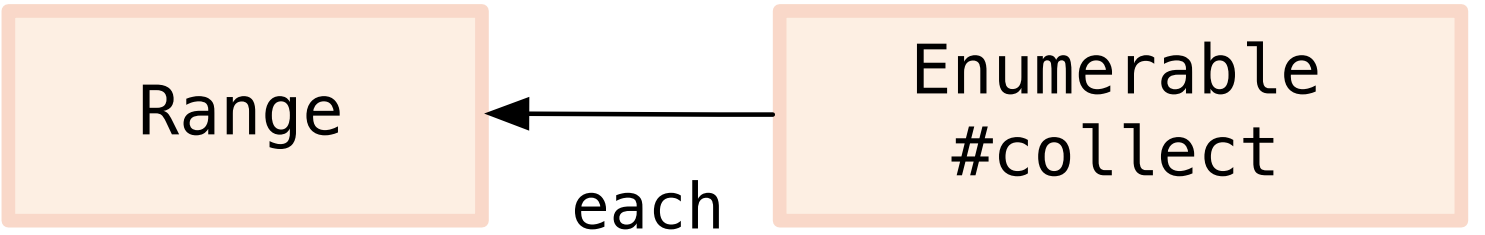
```
=>[2,5,10,17,26,37,50,65,82,101]
```

```
(1..Float::INFINITY)
   .lazy
   .collect { |x| x*x }
   .collect { |x| x+1 }
   .first(10)
```

```
=>[2,5,10,17,26,37,50,65,82,101]
```

Enumerable#collect

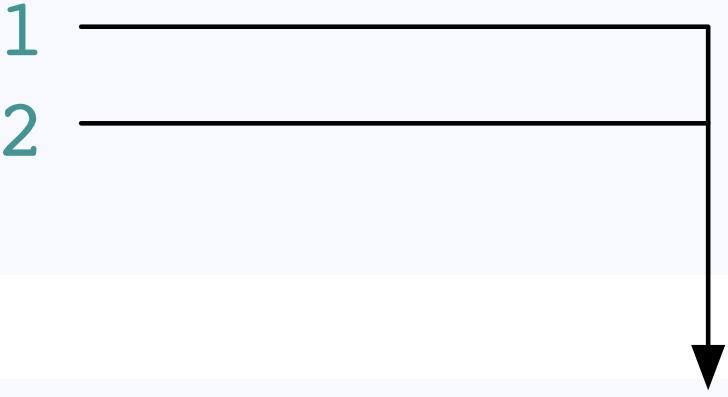
```
(1..10).collect { |x| x*x }
```



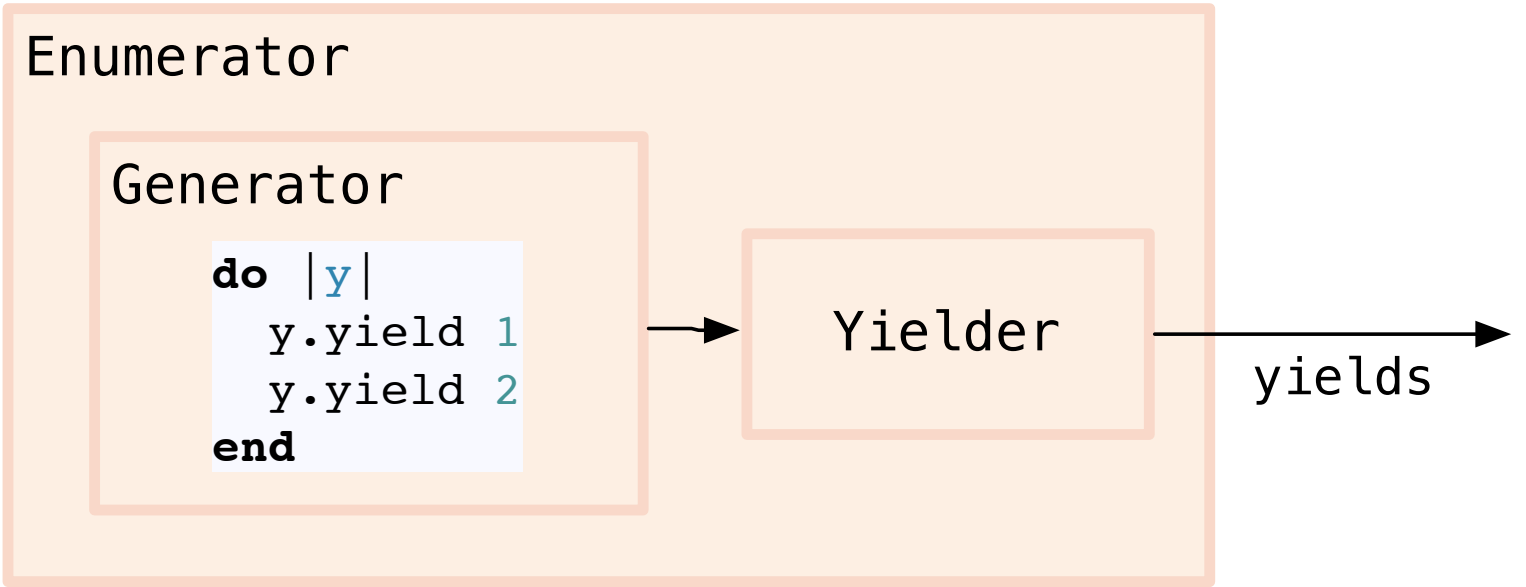
Enumerator

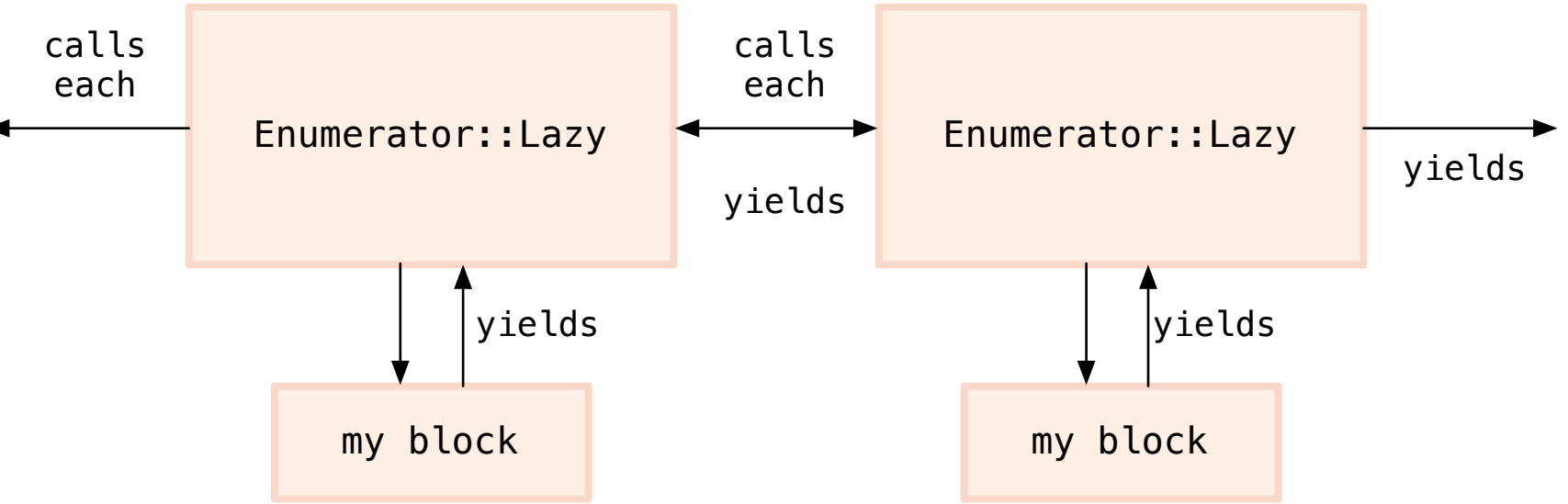
```
enum = Enumerator.new do |y|
  y.yield 1
  y.yield 2
end

p enum.collect { |x| x*x }
=> [1, 4]
```

```
enum = Enumerator.new do |y|
  y.yield 1
  y.yield 2
end
```



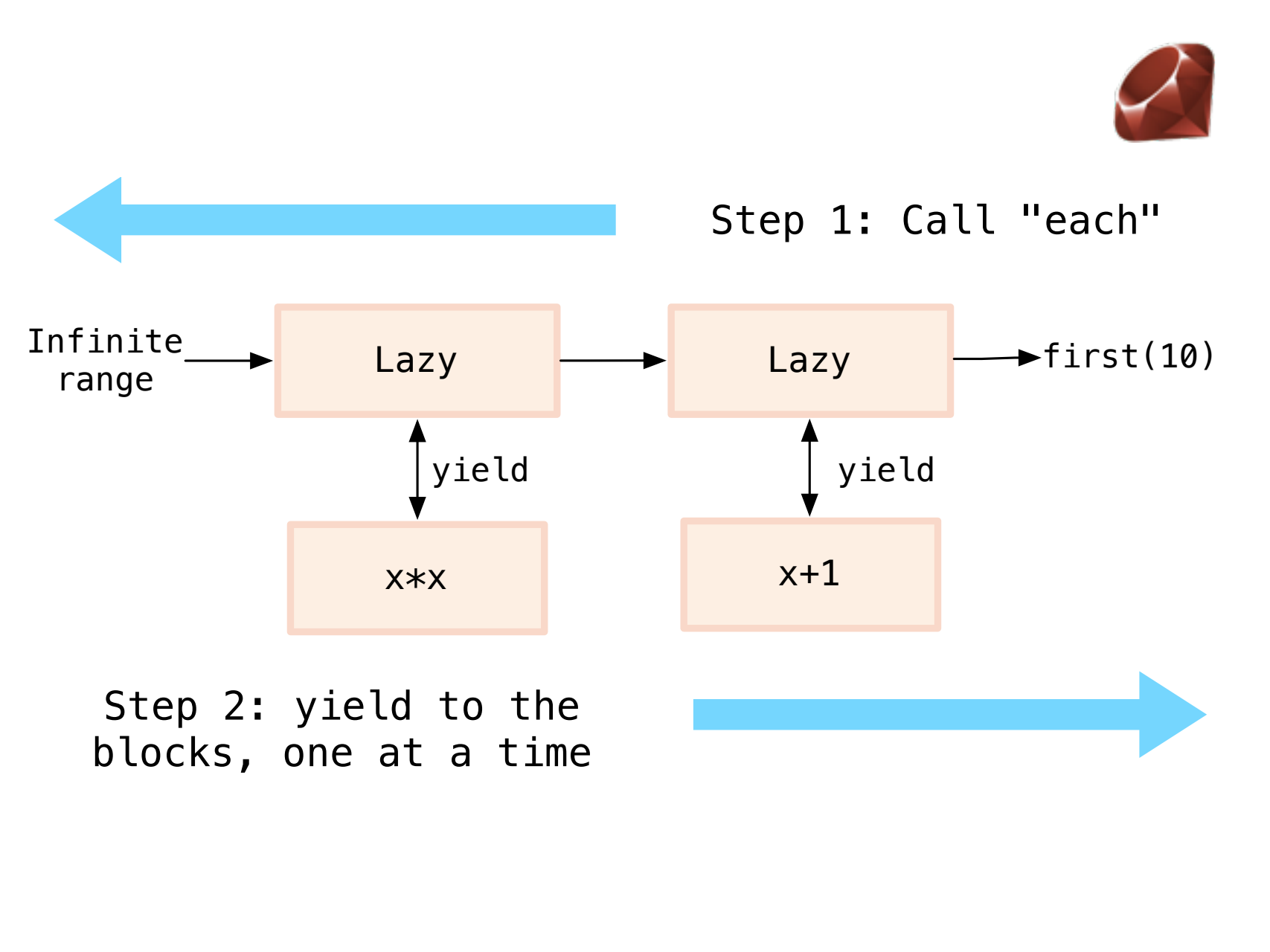
```
enum.collect do |x|
  x*x
end
```


Enumerator
Generator
do |y|
y.yield 1
y.yield 2
end
Yielder
yields


calls
each
Enumerator::Lazy
calls
each
yields
Enumerator::Lazy
yields
yields
my block
yields
my block

```
(1..Float::INFINITY)
   .lazy
   .collect { |x| x*x }
   .collect { |x| x+1 }
   .first(10)
```

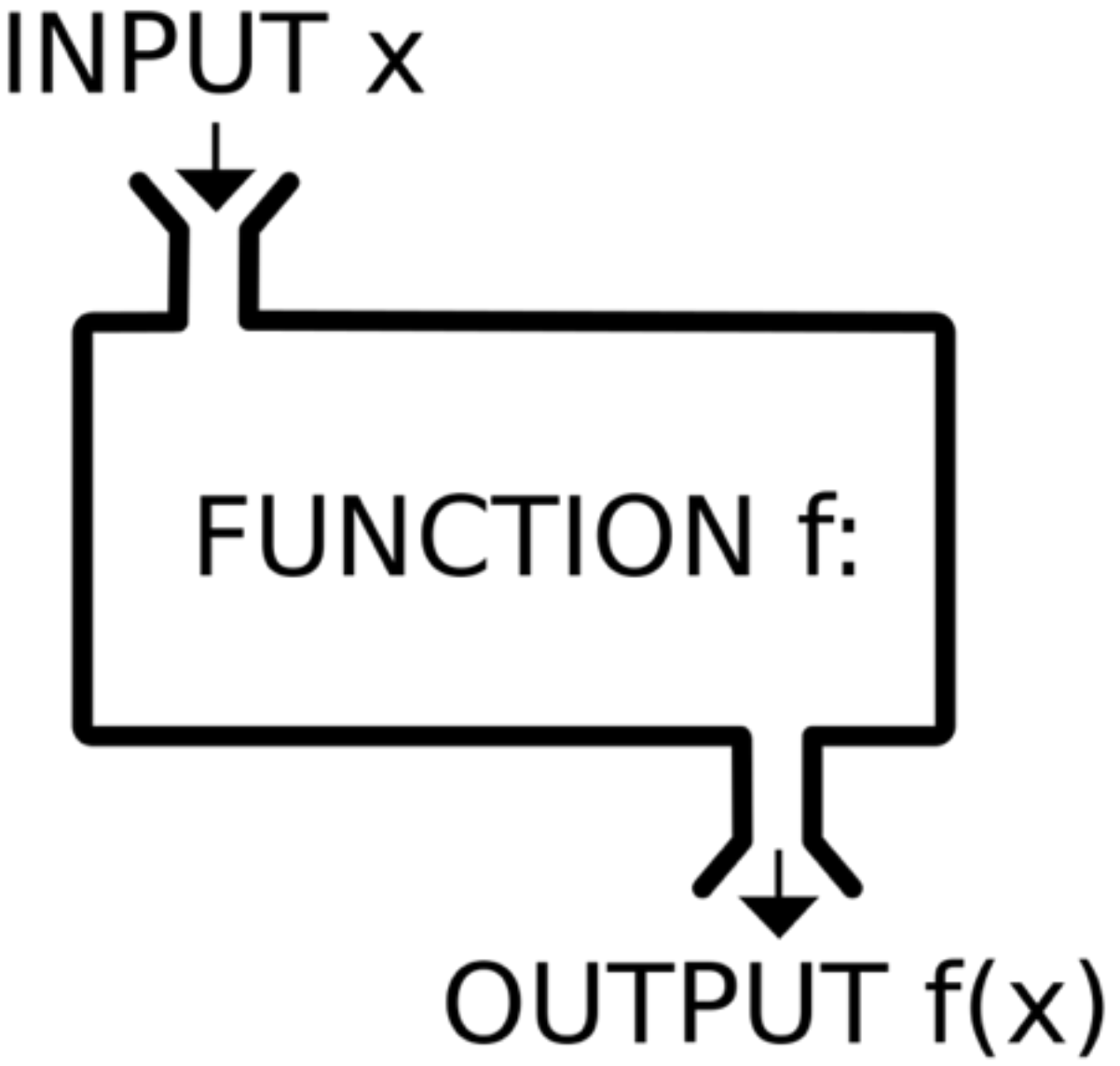
```
=>[2,5,10,17,26,37,50,65,82,101]
```


Step 1: Call "each"
Infinite range
Lazy
Lazy
first(10)
yield
yield
x*x
x+1
Step 2: yield to the blocks, one at a time

memoization

```
slow_fib 0 = 0
slow_fib 1 = 1
slow_fib n = slow_fib (n-2)
           + slow_fib (n-1)

map slow_fib [1..10]
=> [1,1,2,3,5,8,13,21,34,55]
```

http://www.haskell.org/haskellwiki/Memoization


INPUT x
FUNCTION f:
OUTPUT f(x)

Typical Haskell magic!

```
memoized_fib = (map fib [0 ..] !!)
    where fib 0 = 0
          fib 1 = 1
          fib n = memoized_fib (n-2)
                + memoized_fib (n-1)
```

http://www.haskell.org/haskellwiki/Memoization

Infinite, lazy list of return values

```
(map fib [0 ..] !!)
```

A curried function to return the requested fib

```
[0 ..]
```

```
(0..Float::INFINITY)
```

```
map fib [0 ..]
```

```
(0..Float::INFINITY)
  .lazy.map {|x| fib(x) }
```

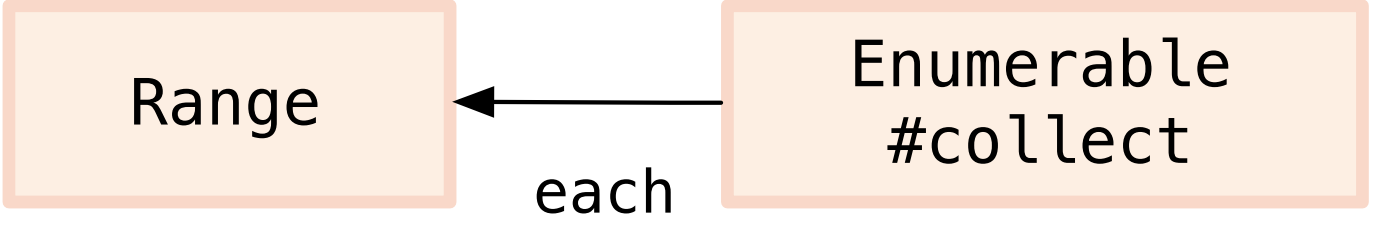
```
(map fib [0 ..] !!)
```

```
cache = (0..Float::INFINITY)
          .lazy.map {|x| fib(x) }

nth_element_from_list =
  lambda { |ary, n| ary[n]}

nth_fib =
  nth_element_from_list.curry[cache]
```

```
map memoized_fib [1..10]
=> [1,1,2,3,5,8,13,21,34,55]
```

```
`block in <main>':
undefined method `[]'
for #<Enumerator::Lazy:
    #<Enumerator::Lazy:
    0..Infinity>:map>
(NoMethodError)
```

```
(0..Float::INFINITY)
     .lazy.map {|x| fib(x) }

nth_element_from_list =
  lambda { |ary, n| ary[n]}
```



```ruby
@cache = {}
@cache[1] = 1
@cache[2] = 1
def memoized_fib(n)
  @cache[n] ||= memoized_fib(n-1)
              + memoized_fib(n-2)
end
```

learn by studying other languages...

and acquire a different perspective on Ruby

Ruby has many functional features, but is not a functional language