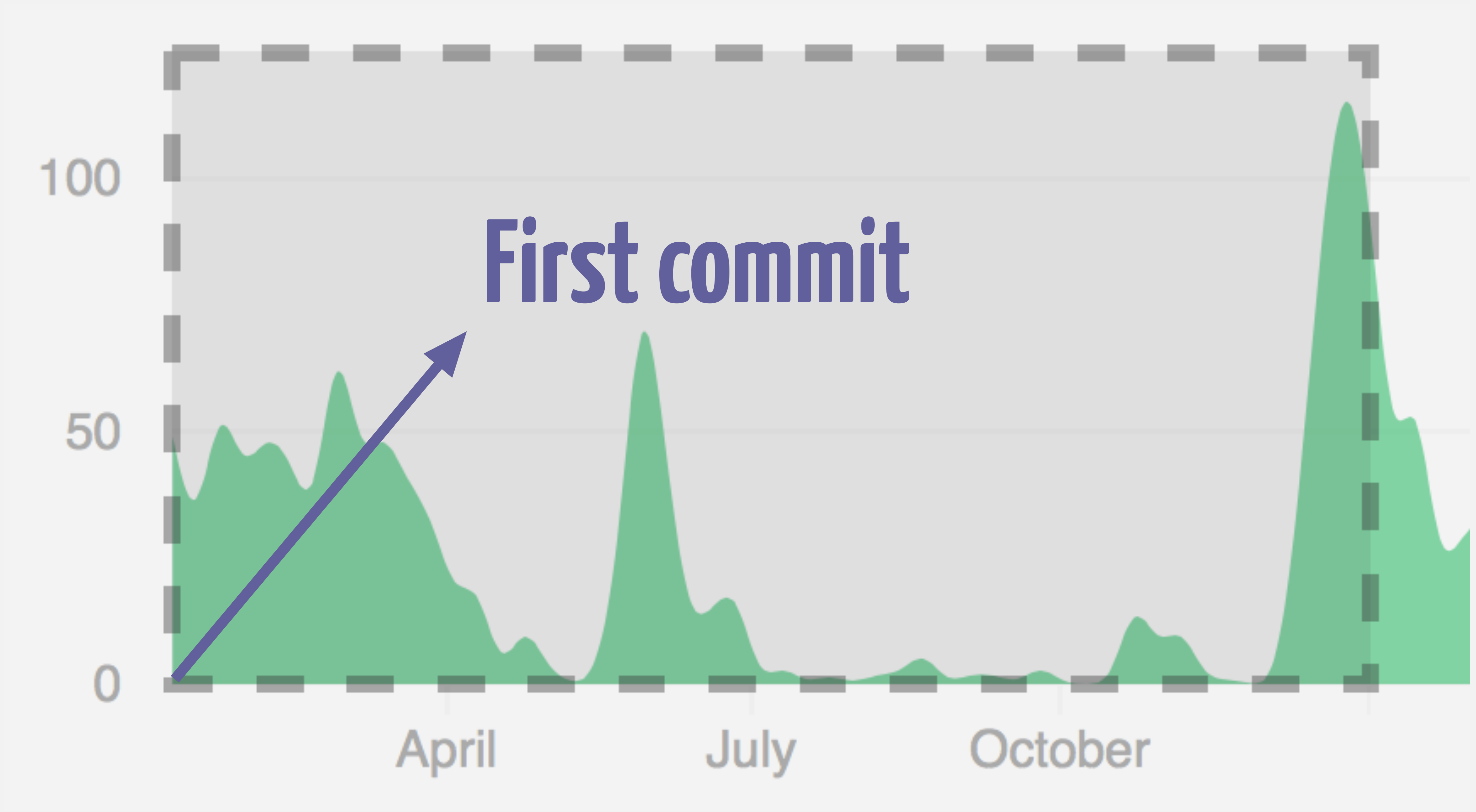
elixir

Past & Future

Timeline: 2011



Timeline: <2011

- 2005 - The Free Lunch is Over
  Herb Sutter

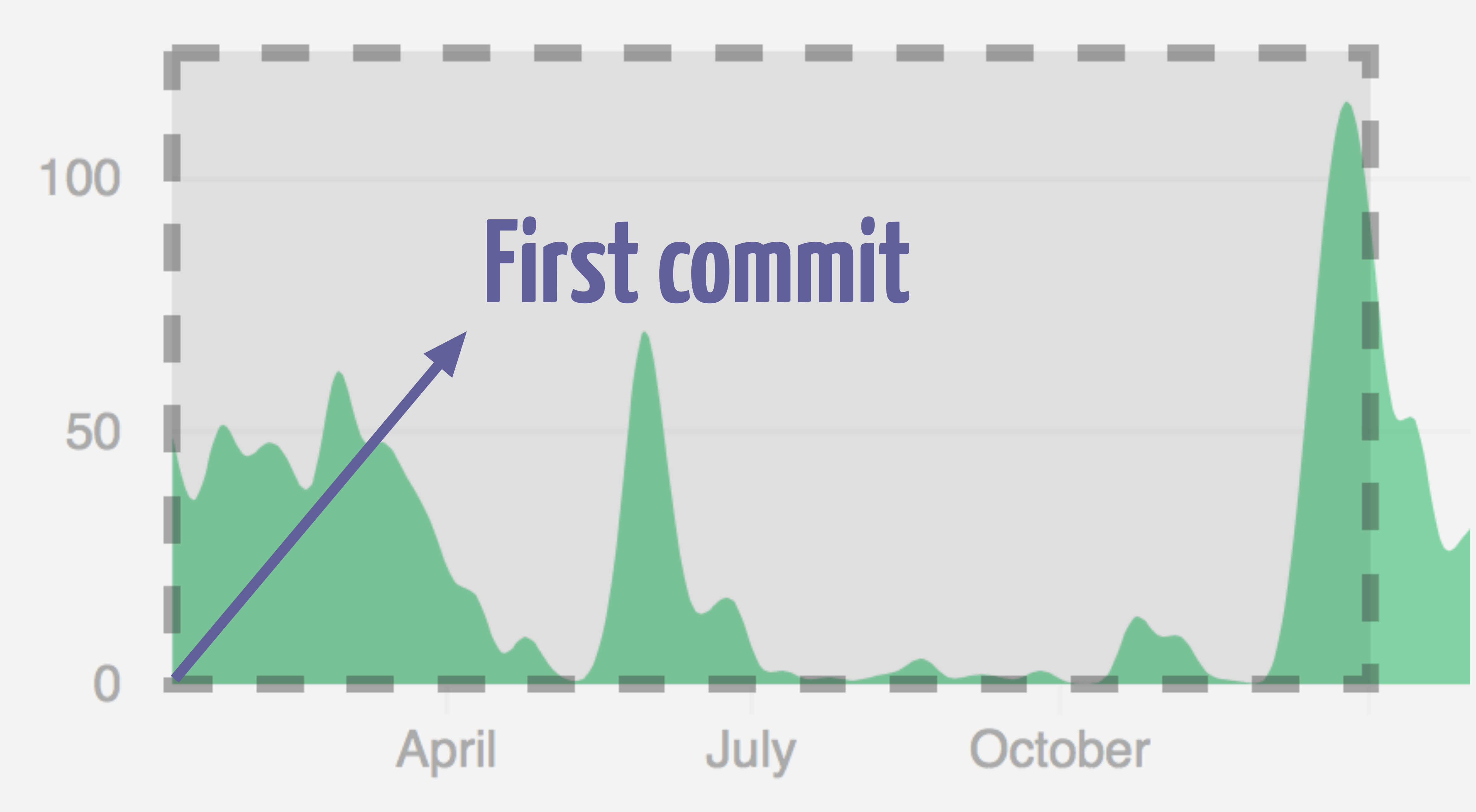
- 2007 - Programming Erlang
  Joe Armstrong

Timeline: <2011

- 2009 - Rails is "threadsafe"
  Rails Core Team

- 2010 - 7 Languages in 7 Weeks
  Bruce Tate

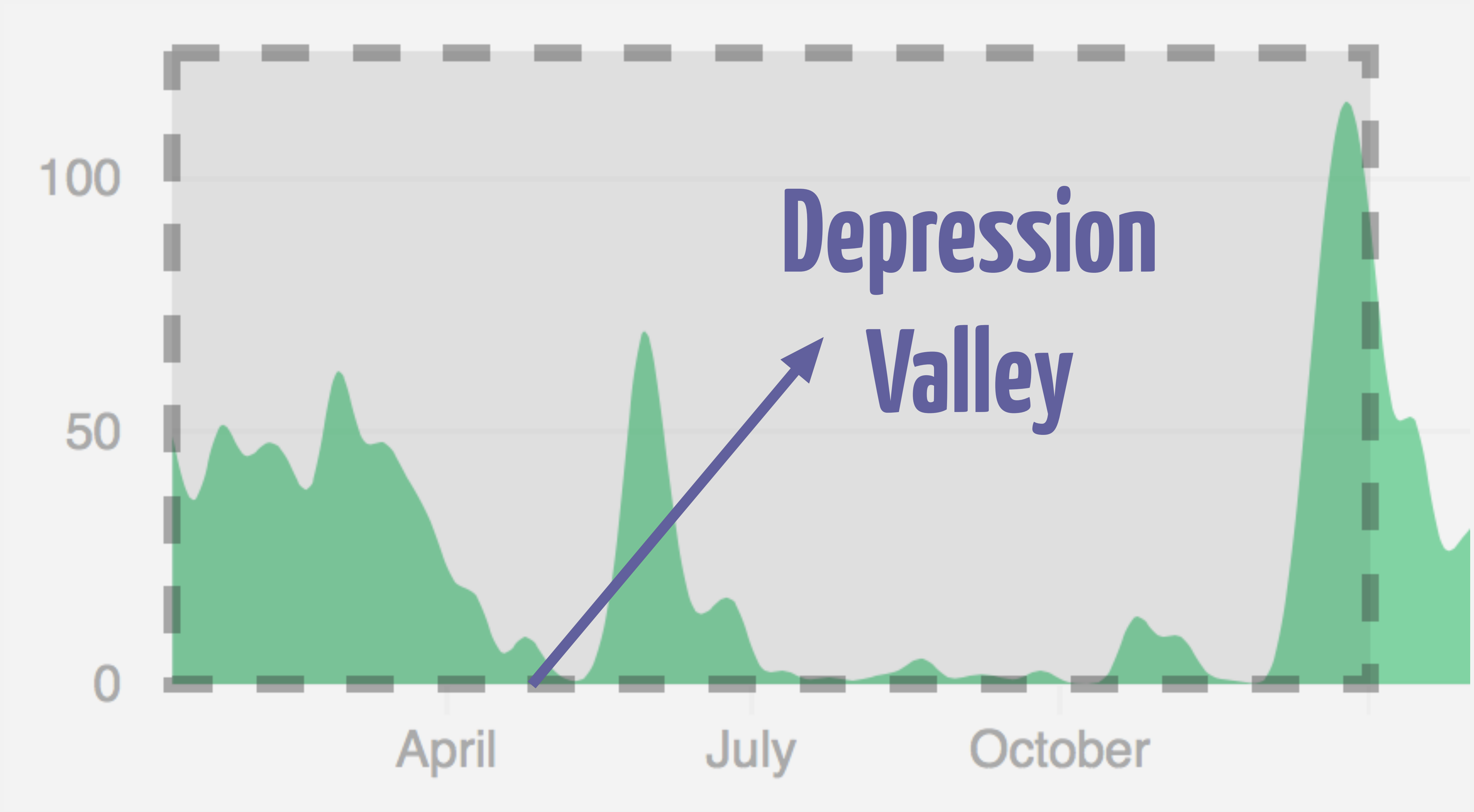
Timeline: 2011



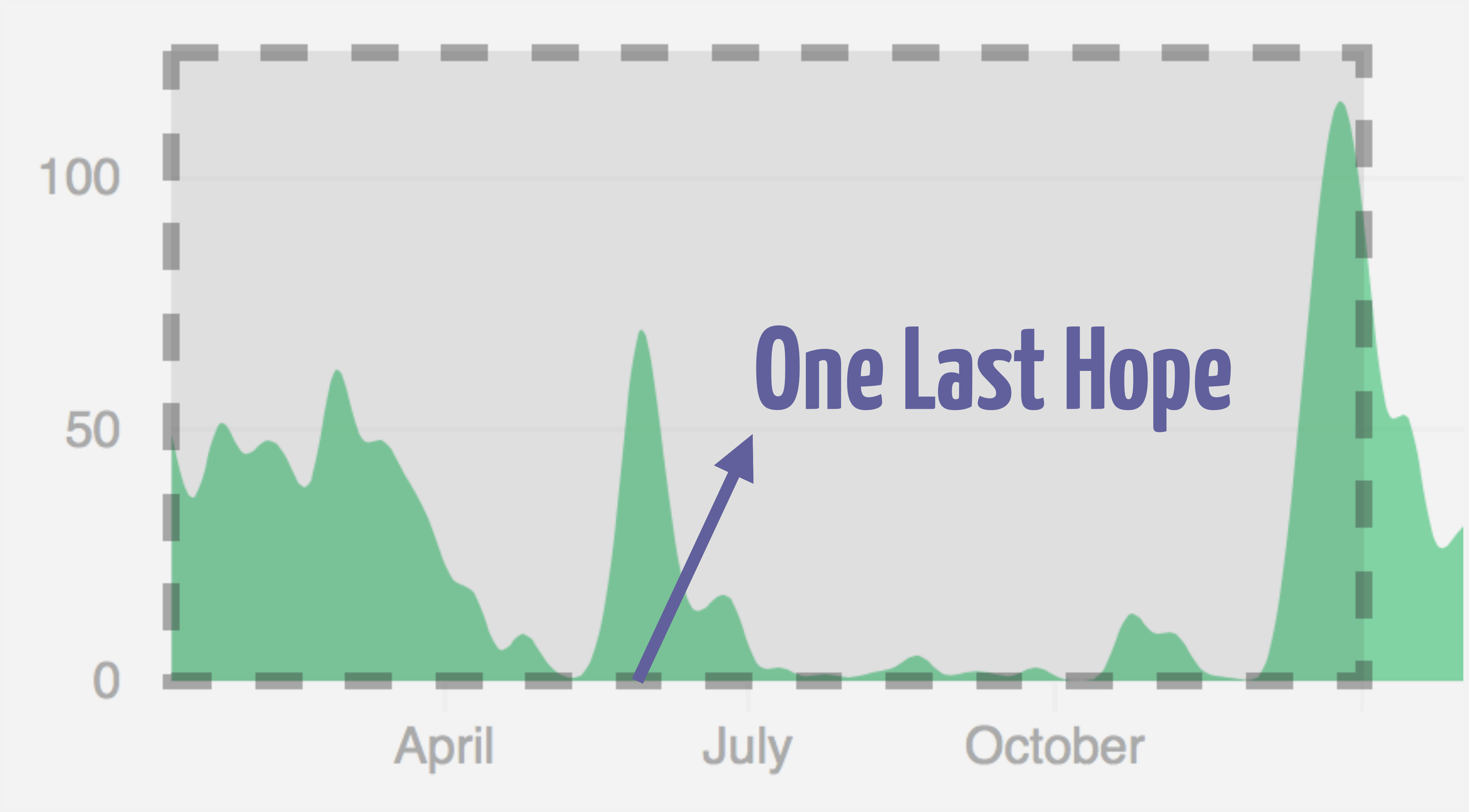
"Elixir" as of Apr/2011

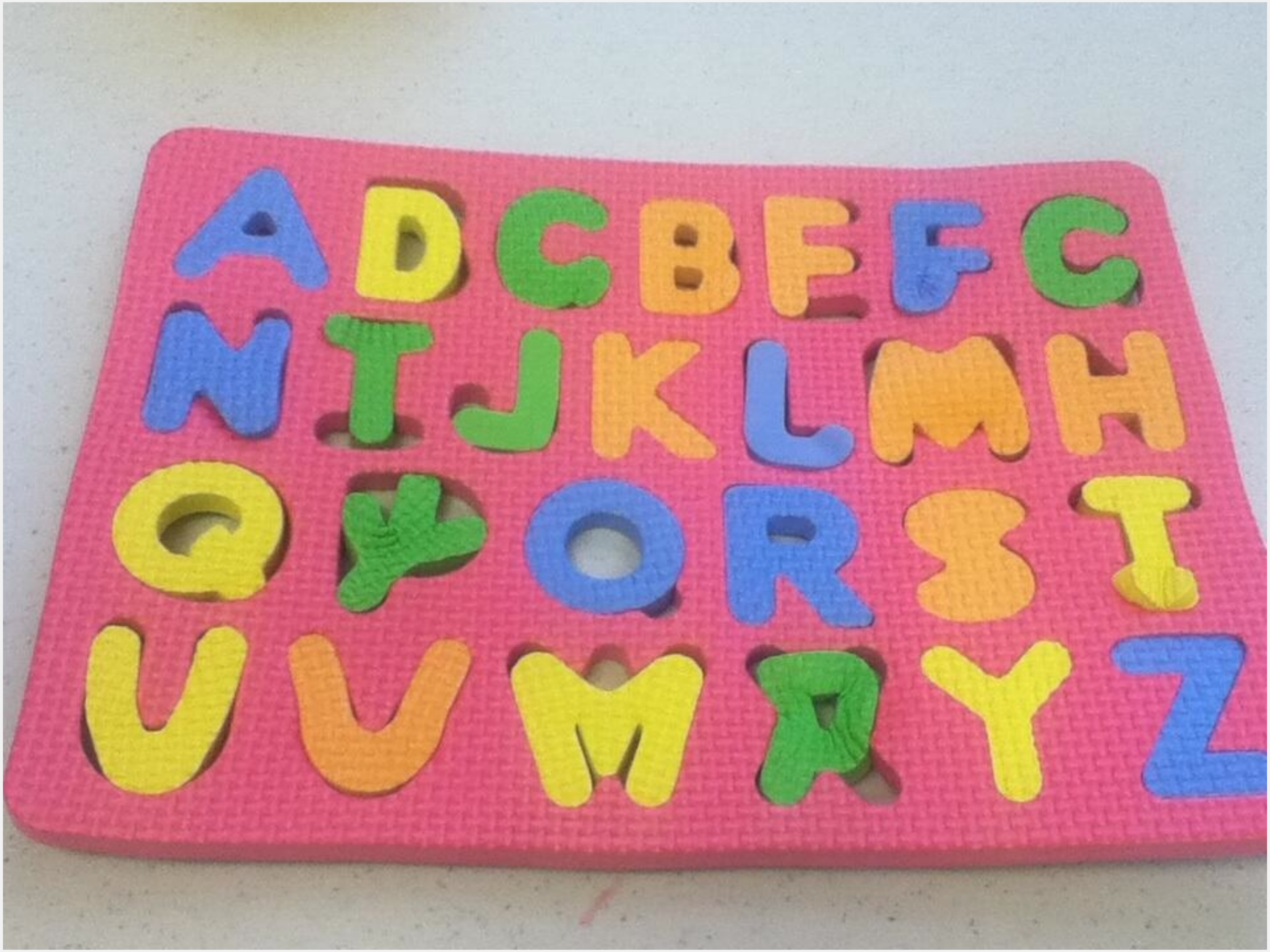
- defobject to define “objects"
- prototype object-model
- eval everywhere   (evil evalware)
- slow, extremely slow
- it broke Erlang’s hot code swapping

Timeline: 2011



Timeline: 2011

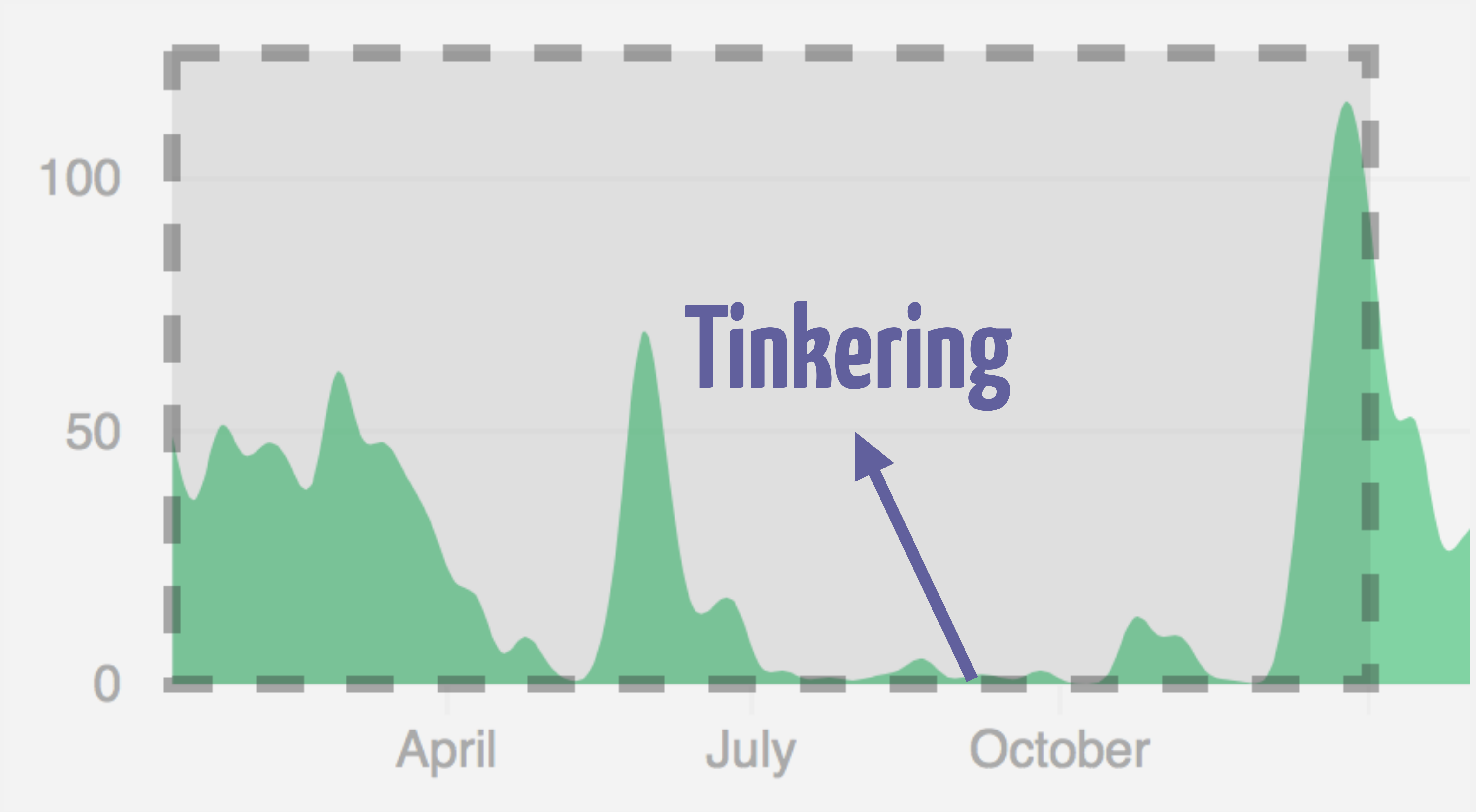



A D C B F F C
N I J K L M H
Q Y O R S I
U U M A Y Z

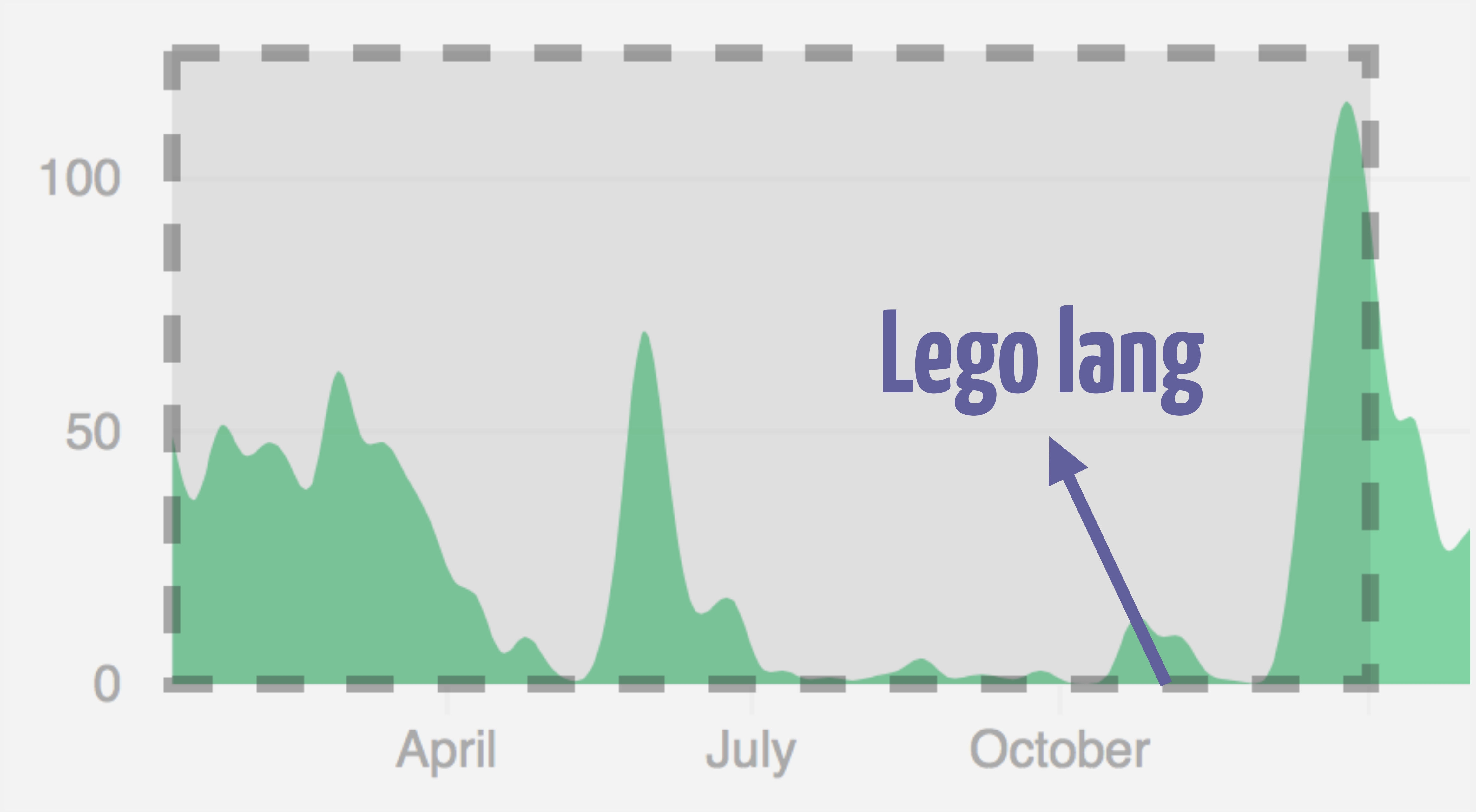
Elixir Goals

- Productivity
    - Meta-programming
- Extensibility
    - Polymorphism
- Compatibility

Timeline: 2011



Timeline: 2011



Meta-programming

- Macros are flexible
- How to combine:
    - Lisp-macros
    - Natural syntax?
- How to guarantee explicitness?

Meta-programming

```
add(1, 2)
{:add, [], [1, 2]}

add 1, 2
{:add, [], [1, 2]}

1 + 2
{:+, [], [1, 2]}
```

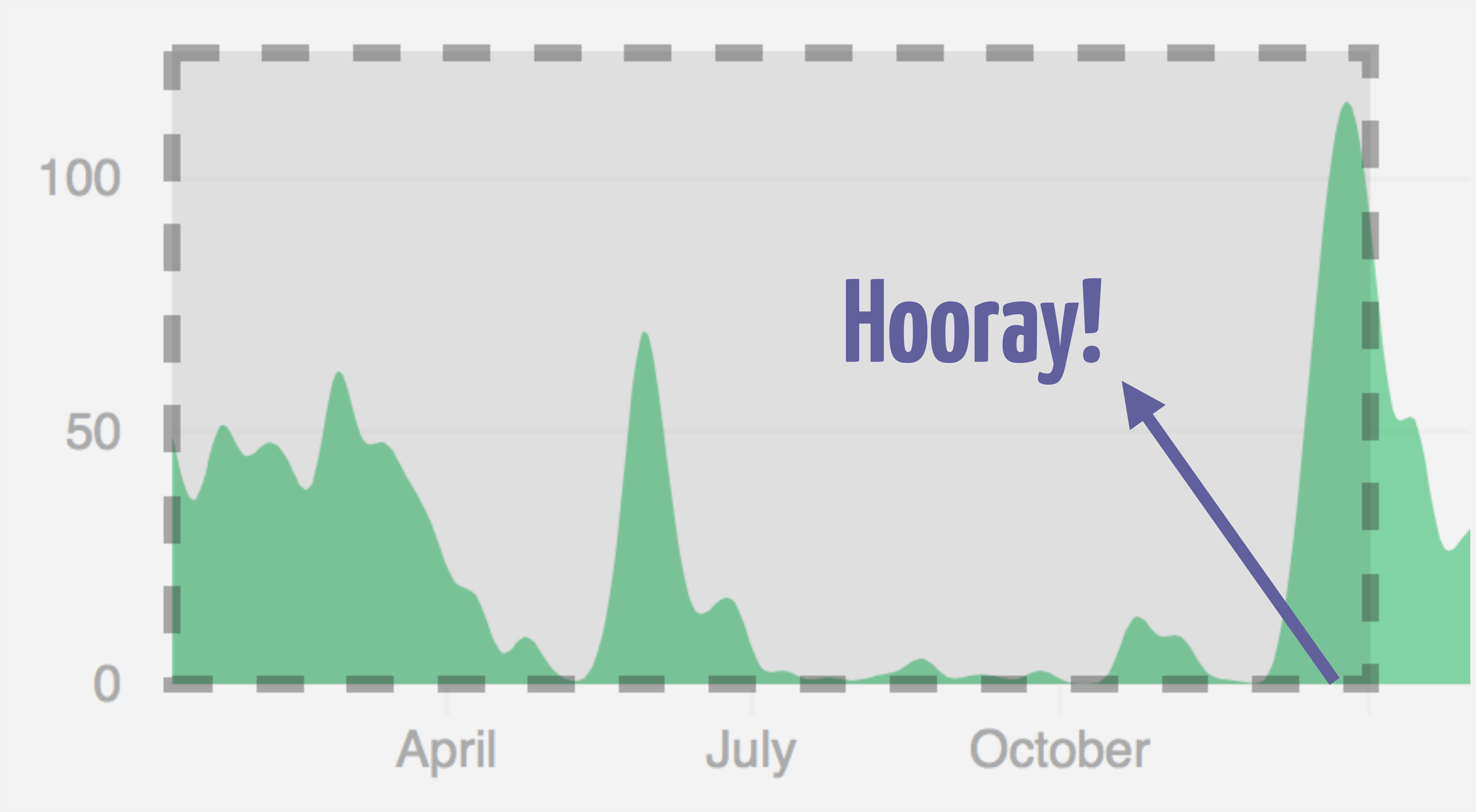
Meta-programming

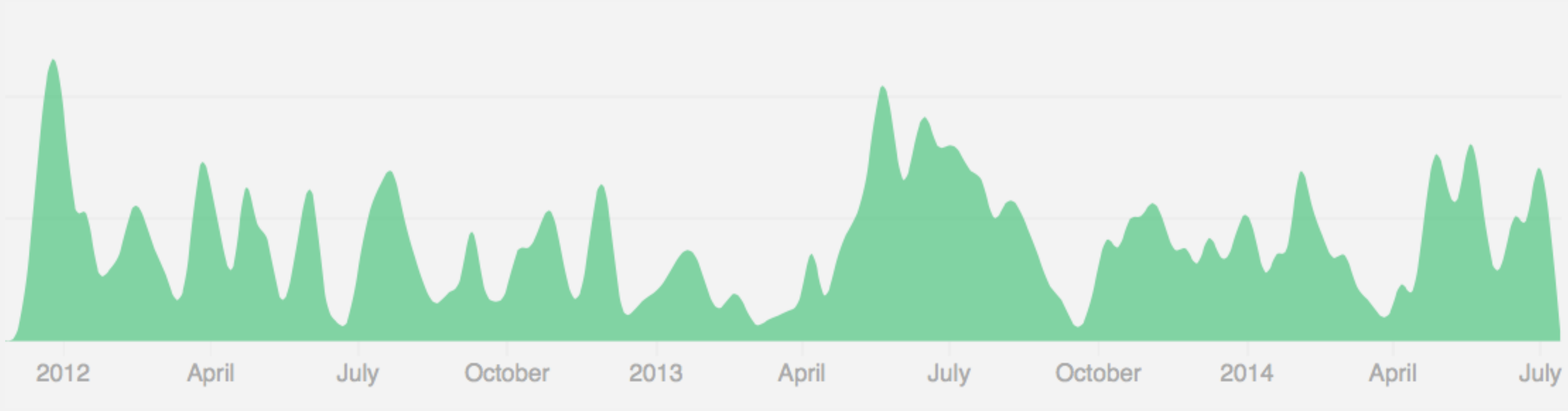
```
quote do
  def hello() do
    unquote(value)
  end
end
```

Meta-programming

```
require MyMacros
```

Timeline: 2011




2012
April
July
October
2013
April
July
October
2014
April
July

Timeline: 2012

- Jan/2012
  plataformatec's blessing
  tecnologia e engenharia de software

- Feb/2012
  Logo and website launched

Timeline: 2012

- May/2012
  Elixir v0.5 launched

- Sep/2012
  First Elixir presentation at
  Emerging Languages Camp

Timeline: 2013

- May/2013
  "Programming Elixir" announced

- Jun/2013
  "Introducing Elixir" announced

Critical mass!

Elixir Goals

- Productivity
  - First-class documentation
  - Tooling (Mix, ExUnit, IEx)
  - Hex packages

Elixir Goals

- Extensibility
  - Macros
  - Structs & Protocols (polymorphism)

Elixir Goals

- Compatibility
    - Concurrency
    - Distribution
    - Embrace & extend

Today

- v0.14.3 - no more planned backwards incompatibilities
- v0.15.0 - Logger and fix <= 6 pending issues
- v1.0.0!

The Unknown Future

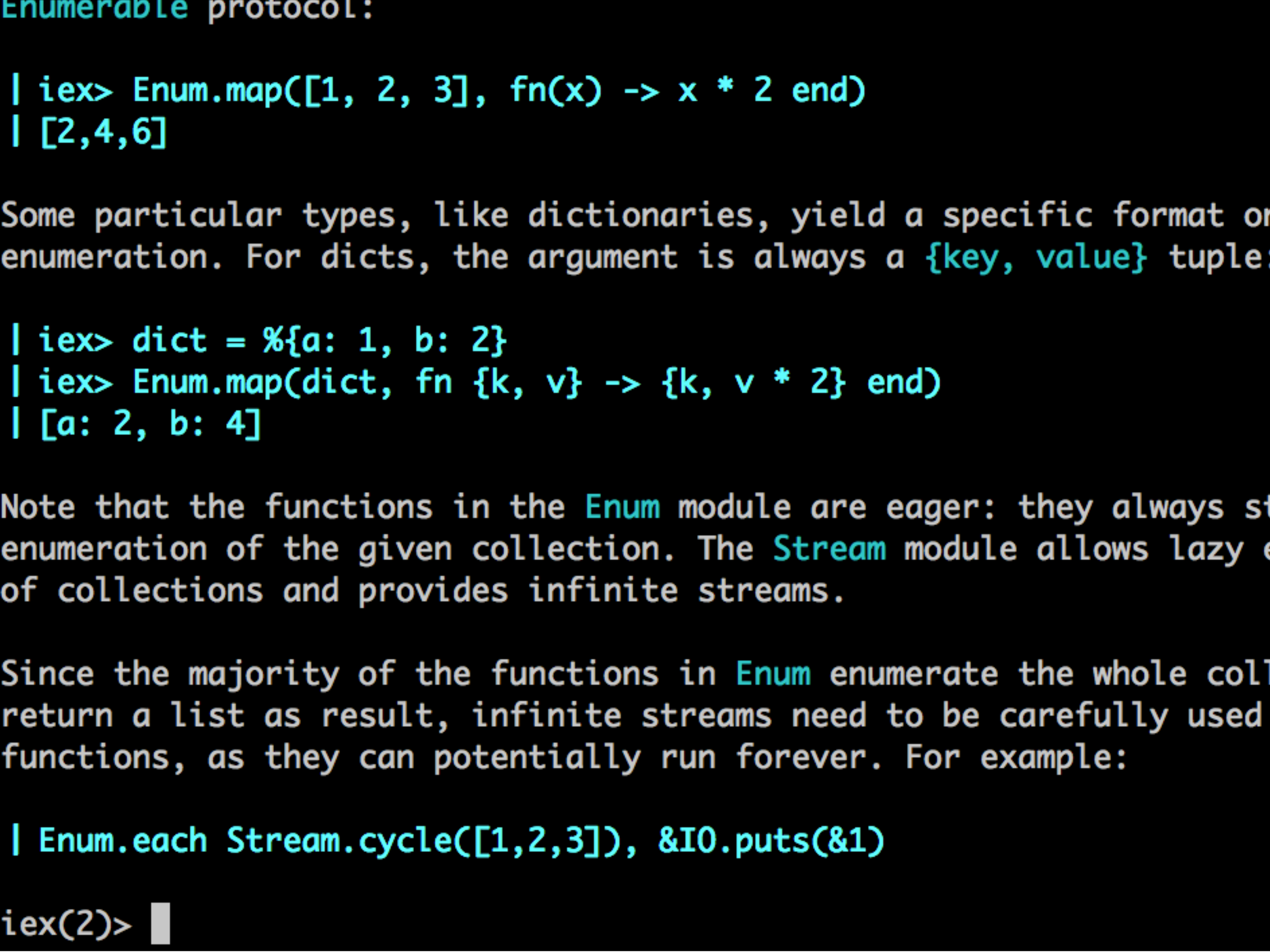
Erlang

Tracing

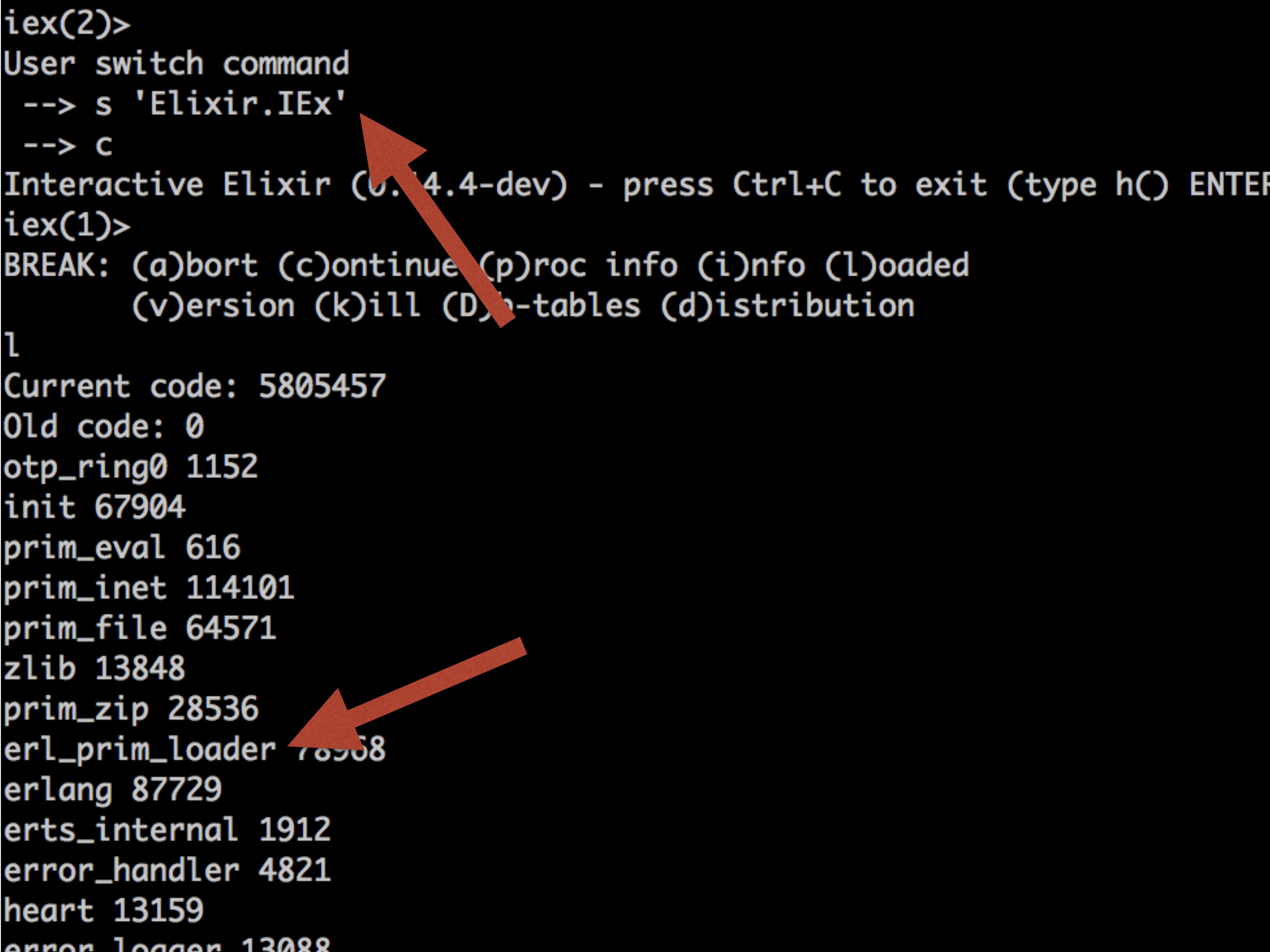
- erlang:trace/3 and erlang:trace_pattern/3
- Can trace function calls, process lifecycle, process interactions and more
- https://github.com/fishcakez/dbg

IEx

- Interactive Elixir shell
- Fantastic helpers, remote shell, pry
- Emacs-mapping, poorly customizable

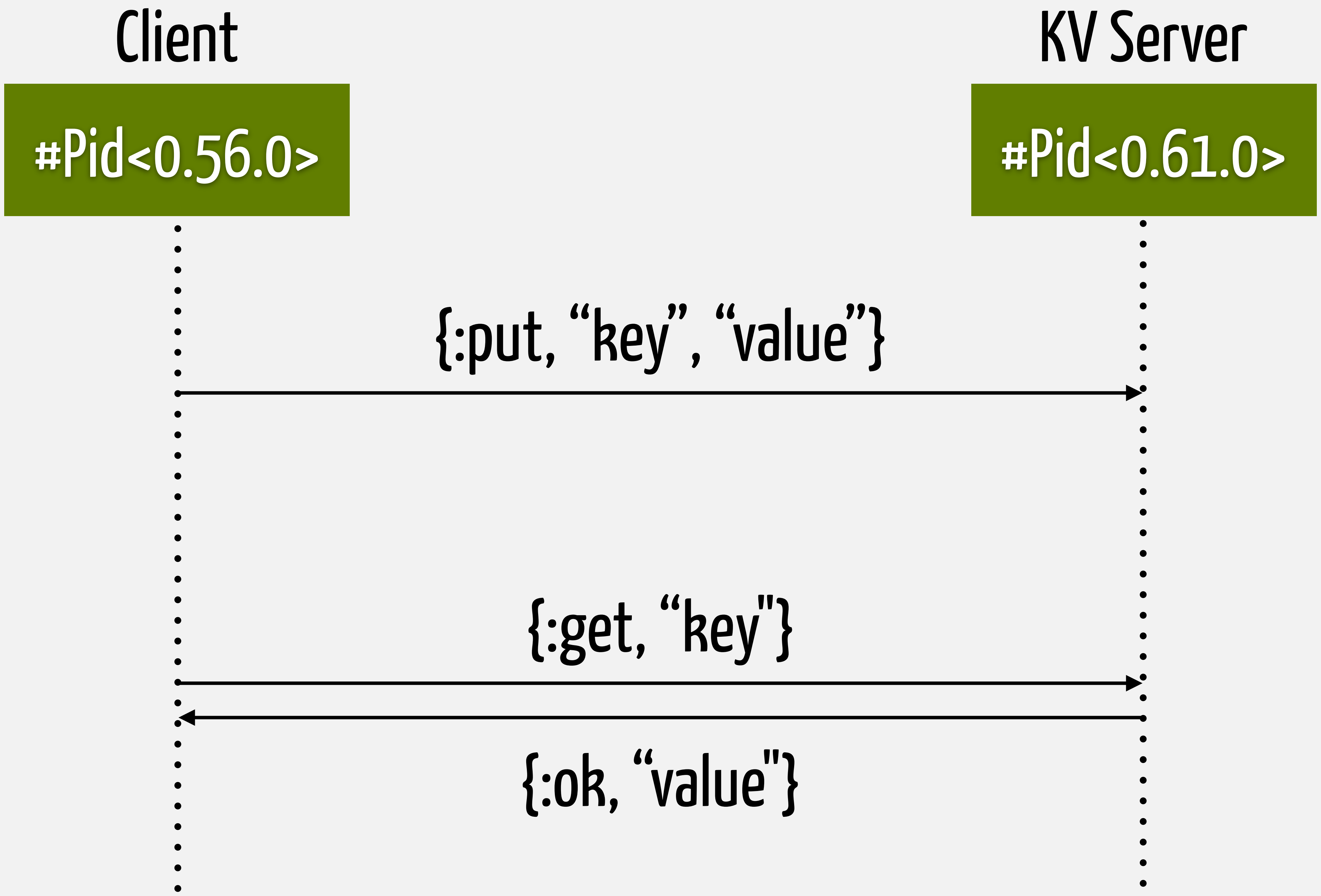


```
Enumerable protocol:

 | iex> Enum.map([1, 2, 3], fn(x) -> x * 2 end)
 | [2,4,6]

Some particular types, like dictionaries, yield a specific format or
enumeration. For dicts, the argument is always a {key, value} tuple

 | iex> dict = %{a: 1, b: 2}
 | iex> Enum.map(dict, fn {k, v} -> {k, v * 2} end)
 | [a: 2, b: 4]

Note that the functions in the Enum module are eager: they always st
enumeration of the given collection. The Stream module allows lazy e
of collections and provides infinite streams.

Since the majority of the functions in Enum enumerate the whole coll
return a list as result, infinite streams need to be carefully used
functions, as they can potentially run forever. For example:

 | Enum.each Stream.cycle([1,2,3]), &IO.puts(&1)

iex(2)> █
```

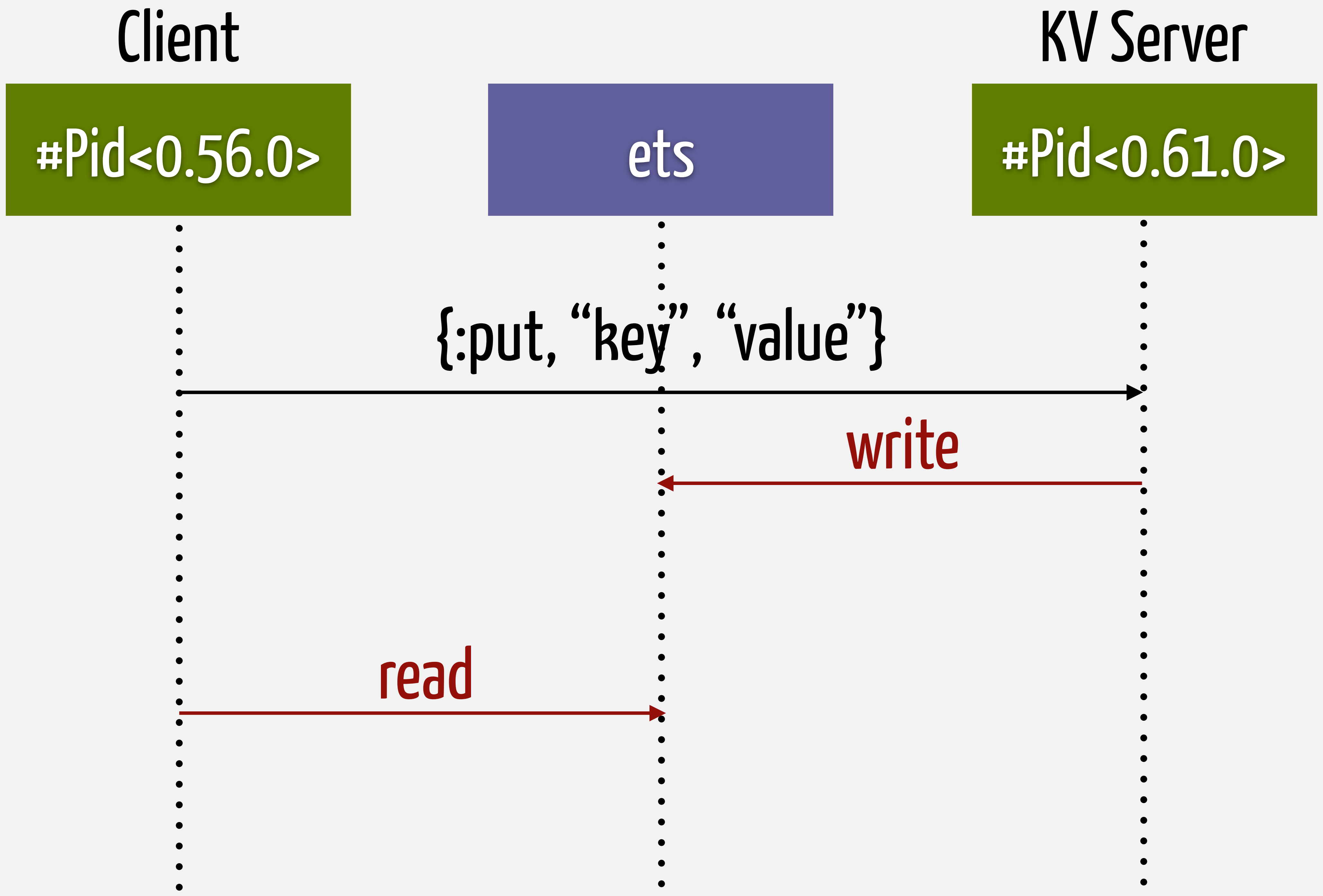


```
iex(2)>
User switch command
 --> s 'Elixir.IEx'
 --> c
Interactive Elixir (0.14.4-dev) - press Ctrl+C to exit (type h() ENTEF
iex(1)>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
l
Current code: 5805457
Old code: 0
otp_ring0 1152
init 67904
prim_eval 616
prim_inet 114101
prim_file 64571
zlib 13848
prim_zip 28536
erl_prim_loader 78968
erlang 87729
erts_internal 1912
error_handler 4821
heart 13159
error_logger 13088
```

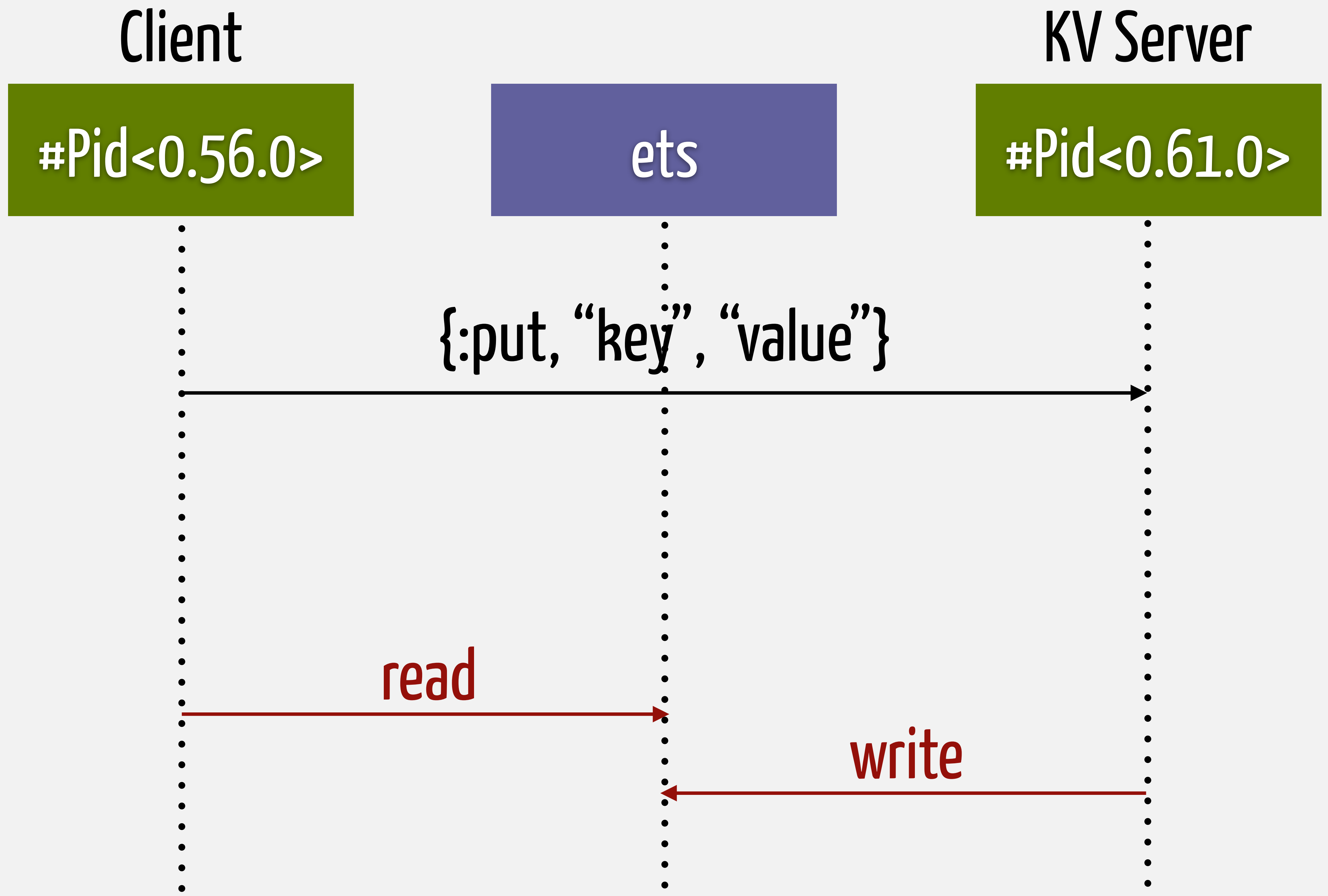
concuerror


Client
KV Server
#Pid<0.56.0>
#Pid<0.61.0>
{:put, “key”, “value”}
{:get, “key"}
{:ok, “value"}

concuerror


Client
KV Server
#Pid<0.56.0>
ets
#Pid<0.61.0>
{:put, “key”, “value”}
write
read

concuerror


Client
KV Server
#Pid<0.56.0>
ets
#Pid<0.61.0>
{:put, “key”, “value”}
read
write

concuerror

- Systematic concurrency testing
- Instruments communication and points with shared state access
- http://concuerror.org/

concuerror in Elixir

- Reports in Elixir terms
- ExUnit integration:

```
@tag :concuerror
test "key-value store" do
  ...
```

Other Erlang initiatives

- http://release-project.eu/
- http://prowessproject.eu/

Elixir

Discriminated Unions

- Imagine you are implementing a calculator:

```
defp calc(op) do
  case op do
    {:+, left, right} -> left + right
    {:-, left, right} -> left - right
    {:*, left, right} -> left * right
    {:/, left, right} -> div(left, right)
  end
end
```

Discriminated Unions

```
defunion Calc.Op do
  def plus(l, r)  = {:+, l, r}
  def minus(l, r) = {:-, l, r}
  def mult(l, r)  = {:*, l, r}
  def div(l, r)   = {:/, l, r}
end
```

Discriminated Unions

```
defp calc(op) do
  Calc.Op.case op do
    plus(left, right)  -> left + right
    minus(left, right) -> left - right
    mult(left, right)  -> left * right
    div(left, right)   -> div(left, right)
  end
end
```

for comprehensions

generator

filter

```
for user <- users,
    user.age >= 18,
    drink <- get_favorite_drinks(user),
    do: {user.name, drink}

#=> [{“Meg”, :tea}, {“Meg”, :coffee},
     {“José”, :coffee}]
```

for + into

```
for user <- users,
    user.age >= 18,
    drink <- get_favorite_drinks(user),
    do: {user.name, drink},
    into: HashSet.new()
```

collectable

```
#=> #HashSet<[{“Meg”, :tea},
              {“Meg”, :coffee},
              {“José”, :coffee}]
```

for + into

```
for user <- users,
    user.age >= 18,
    drink <- get_favorite_drinks(user),
    do: "#{user.name} likes #{drink}",
    into: IO.stream(:stdio, :line)
```

$\longrightarrow$ collectable

```
Meg likes tea
Meg likes coffee
José likes coffee
```

for + ordering

```
for user <- users,
    user.age >= 18,
    drink <- get_favorite_drinks(user),
    order_by: user.age,
    do: {user.name, drink}

#=> [{“José”, :coffee},
     {“Meg”, :tea}, {“Meg”, :coffee}]
```

for + grouping

```
for user <- users,
    user.age >= 18,
    drink <- get_favorite_drinks(user),
    group_by: drink,
    order_by: user.age,
    do: user.name

#=> %{:coffee => [“José”, “Meg”],
       :tea => [“Meg”]}
```

for comprehensions

- Haskell: Comprehensive Comprehensions
- Common Lisp: LOOP macro
- Common Lisp: do+ package

for comprehensions

```
my_for user <- users,
       user.age >= 18,
       drink <- get_favorite_drinks(user),
       group_by: drink,
       order_by: user.age,
       do: user.name
```

for comprehensions

```
my for user <- users,
       user.age >= 18,
       drink <- get_favorite_drinks(user),
       group_by: drink,
       order_by: user.age,
       do: user.name
```

for comprehensions

```
my(for(user <- users,
       user.age >= 18,
       drink <- get_favorite_drinks(user),
       group_by: drink,
       order_by: user.age,
       do: user.name))
```

*-for comprehensions

```
stream for user <- users,
           user.age >= 18,
           do: user.name
```

*-for comprehensions

```
parallel for user <- users,
             user.role == "investor",
             do: fetch_profile(user)
```

Parallel Options

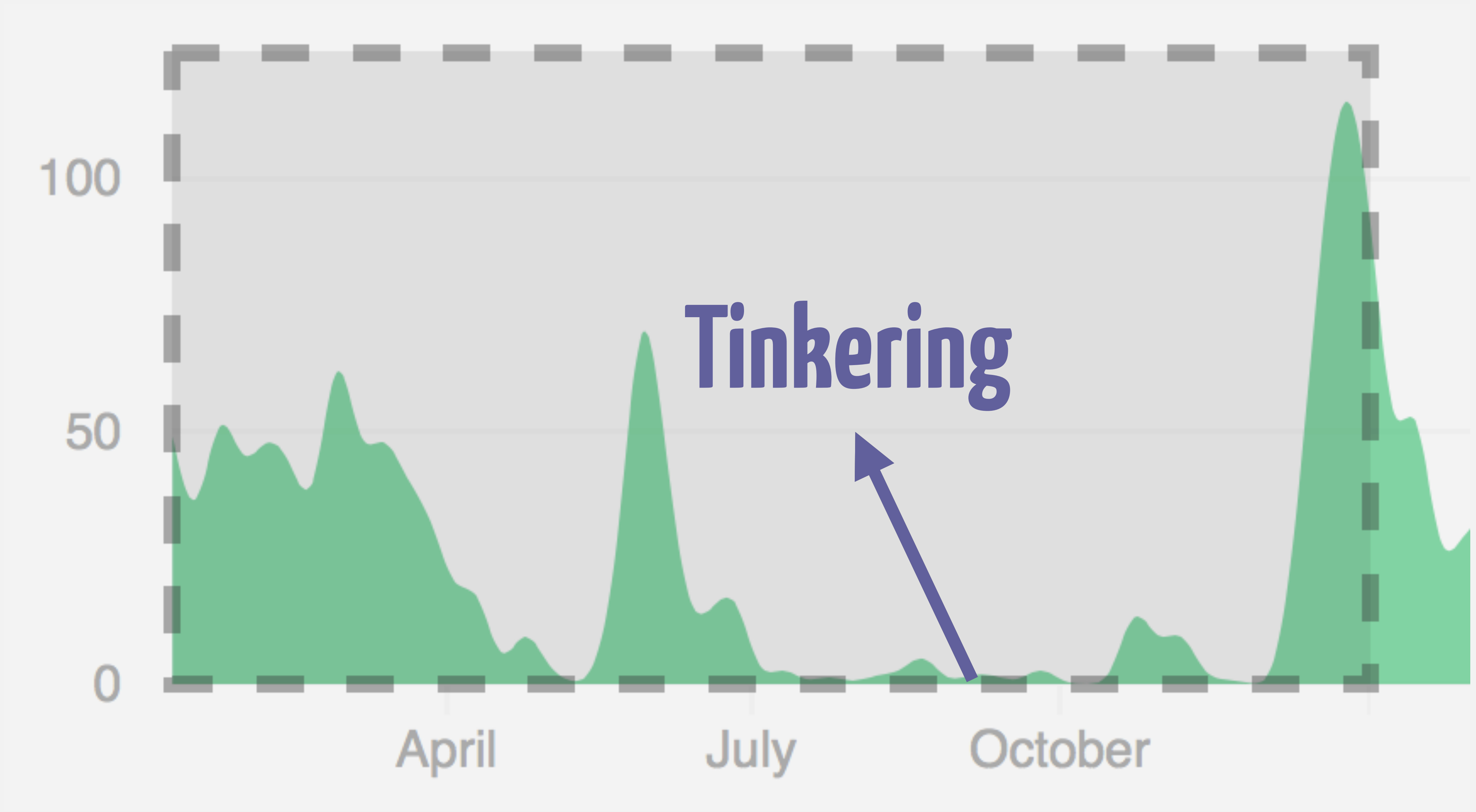
```
parallel for user <- users,
             user.role == "investor",
             do: fetch_profile(user)
```

- Unbound vs Pool
- Pipelines & Feedback

Your ideas!

A D C B F F C
N I J K L M H
Q Y O R S T
U U M A Y Z

Your Timeline



elixir