Compiling a lisp into a lisp using a lisp

with Jan Stępień

@janstepien

affiliations

cljmuc

INNOQ

Hatenke

jetzt

Halunke

by Lucas Dohmen

IS A BEAUTIFUL OBJECT ORIENTED PROGRAMMING LANGUAGE

Halunke

by Lucas Dohmen

IS A BEAUTIFUL OBJECT ORIENTED PROGRAMMING LANGUAGE

it's a Lisp!

$$(a > 1)\ (\text{'}a = 1)$$

$$\{(a > 1)\}$$

$$\{\,|\,\text{'}a\,|\,(a > 1)\}$$

immutable!

Lisp!

message passing!

Lucas as the author!

WOW!

SPARKLES!

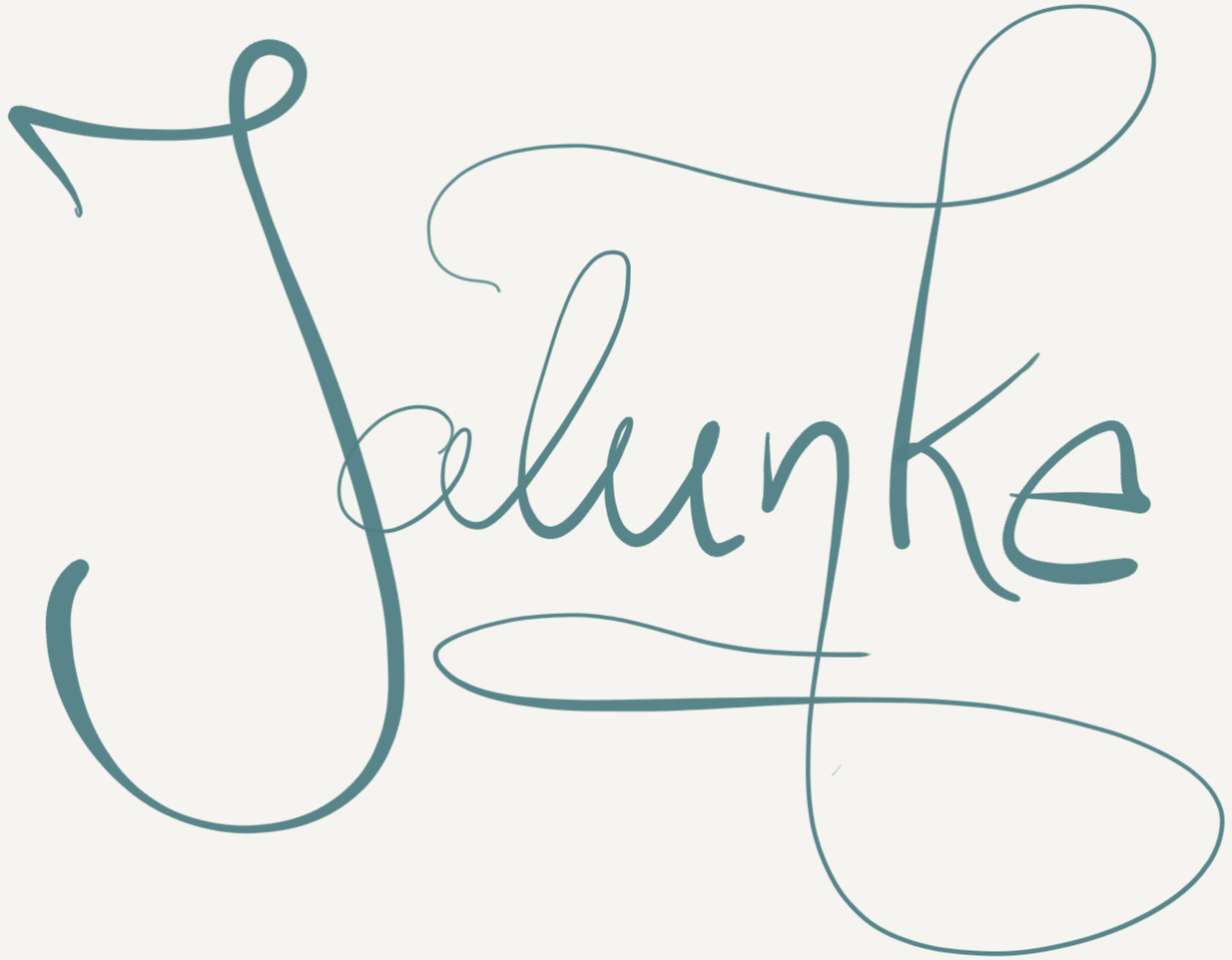
```
(1 + 2)
```

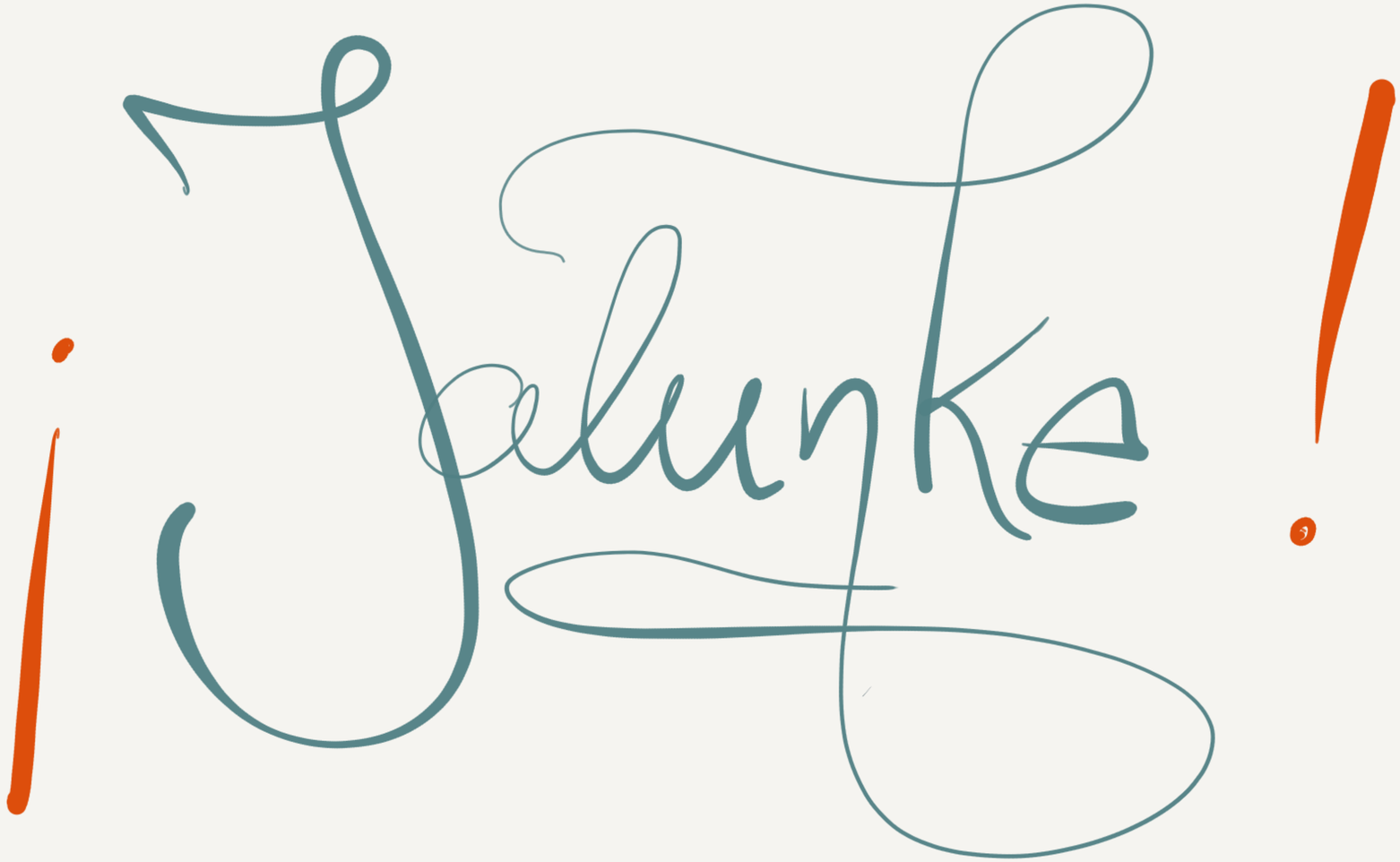
replace with

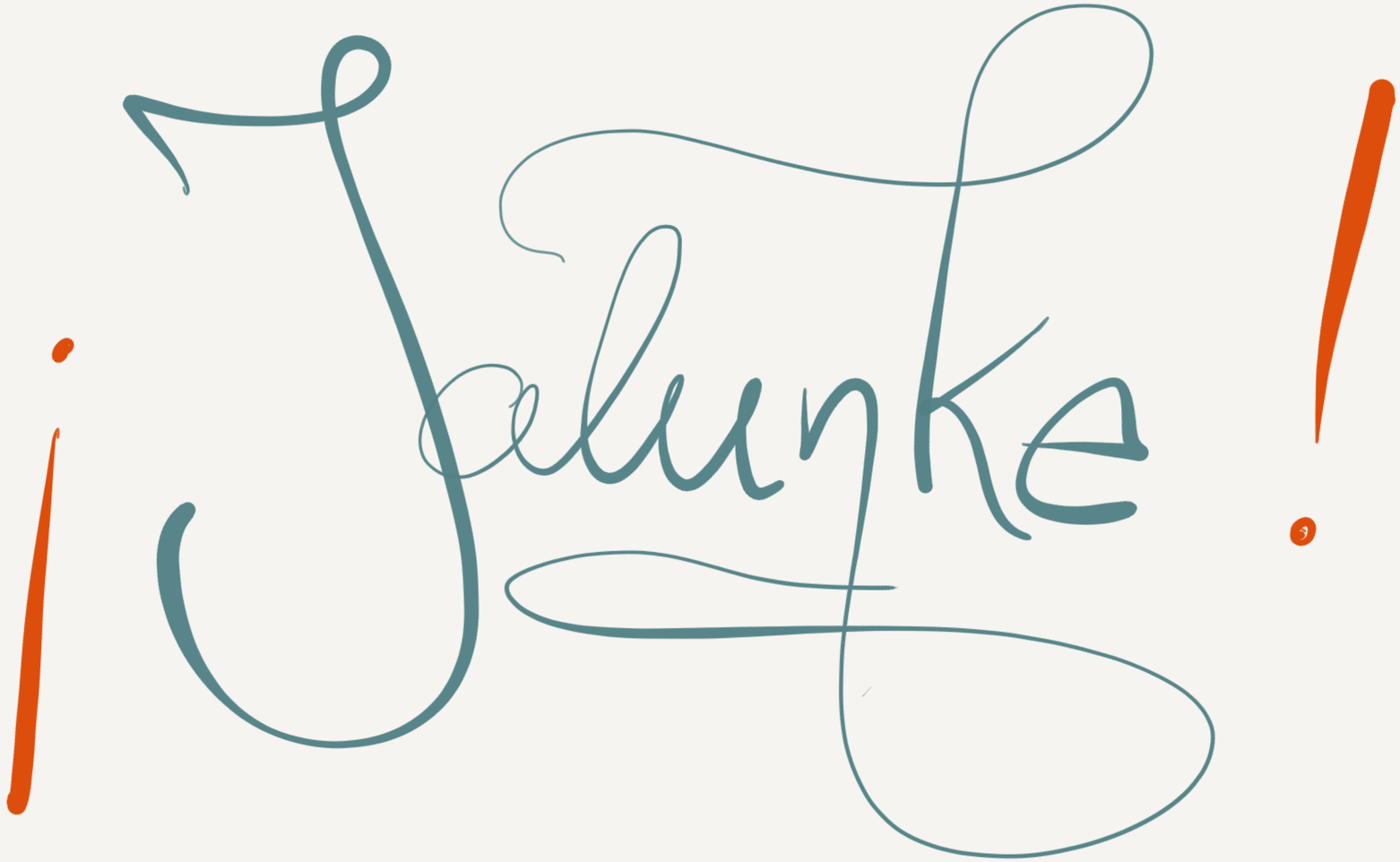
```
("cons" replace " " with "j")
```

```
((1 < 3) then { "yes" }
         else { "no" })
```

then else

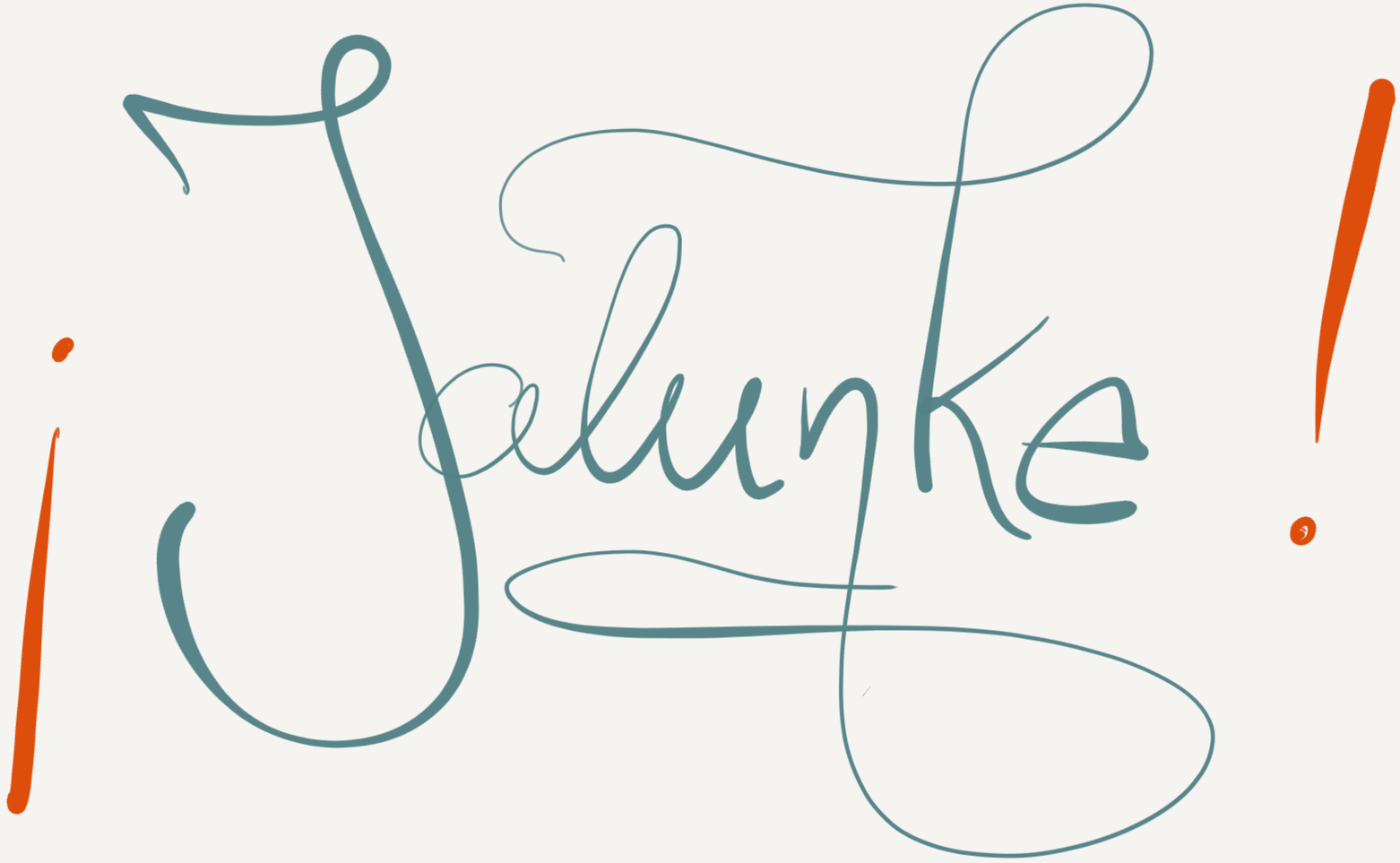

Jalunke


¡Salute!



Pure Java Halunke

clojure.jar



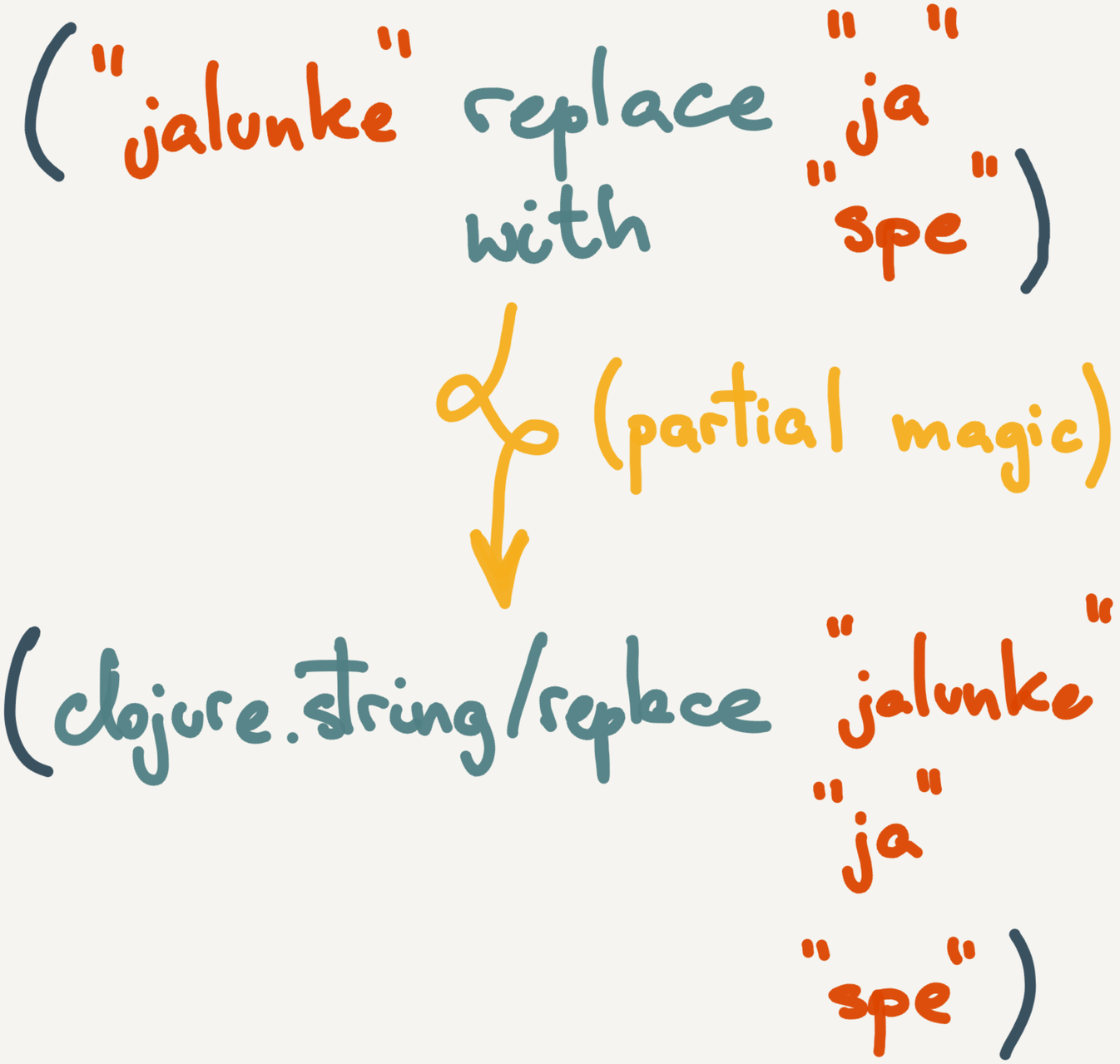
Pure Java Halunke

(give or take Clojure)

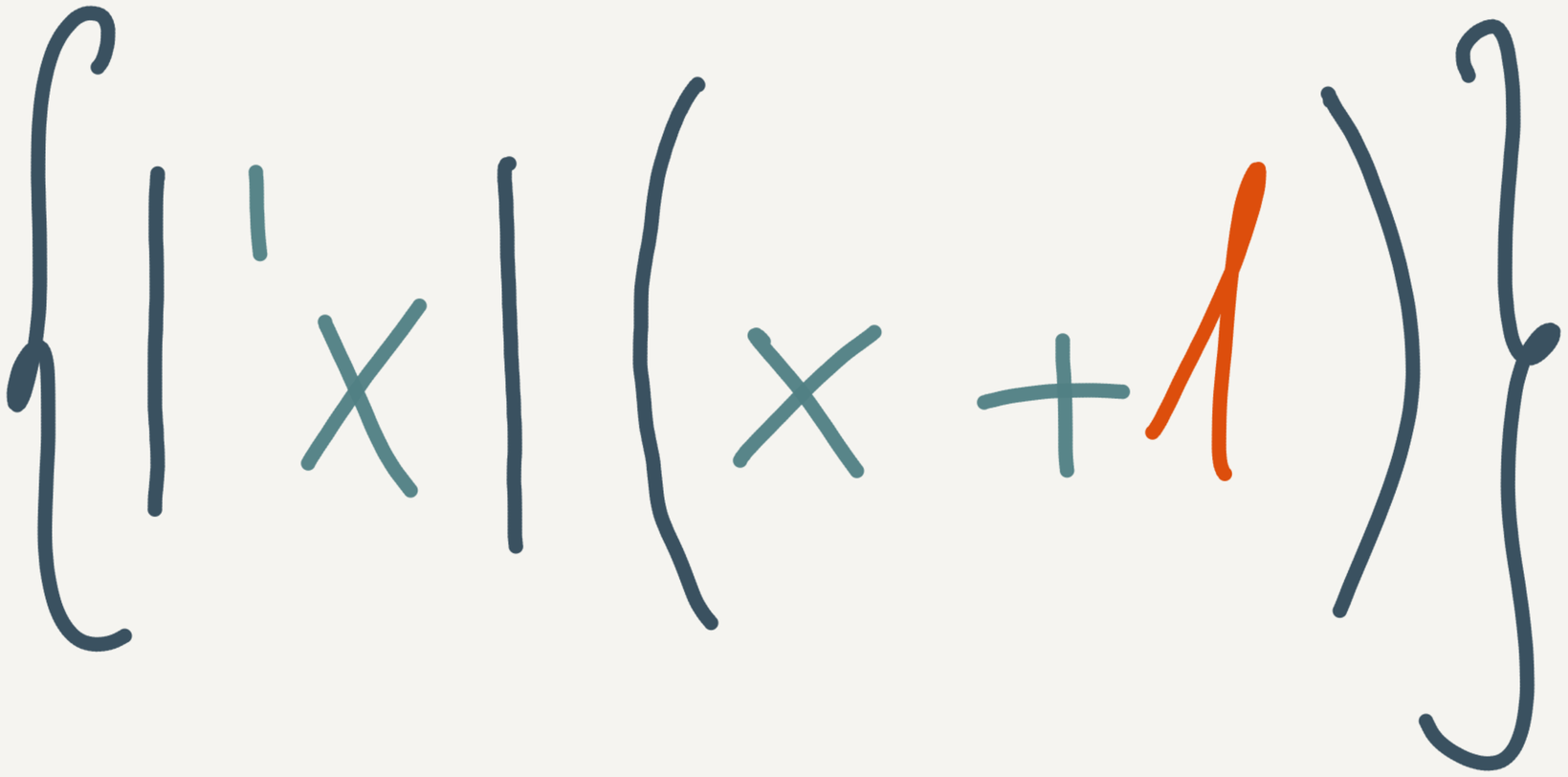
Halunke ~~~> Clojure

Simple?

Easy?


("jalunke" replace with "ja" "spe")
(partial magic)
(clojure.string/replace "jalunke" "ja" "spe")


clojure.core
read-string



(partial misunderstanding)

Halunke ~~~> Clojure

LEXING	stream of tokens
PARSING	syntax tree
CODEGEN	resulting code

"{|'a|(a+3)}"

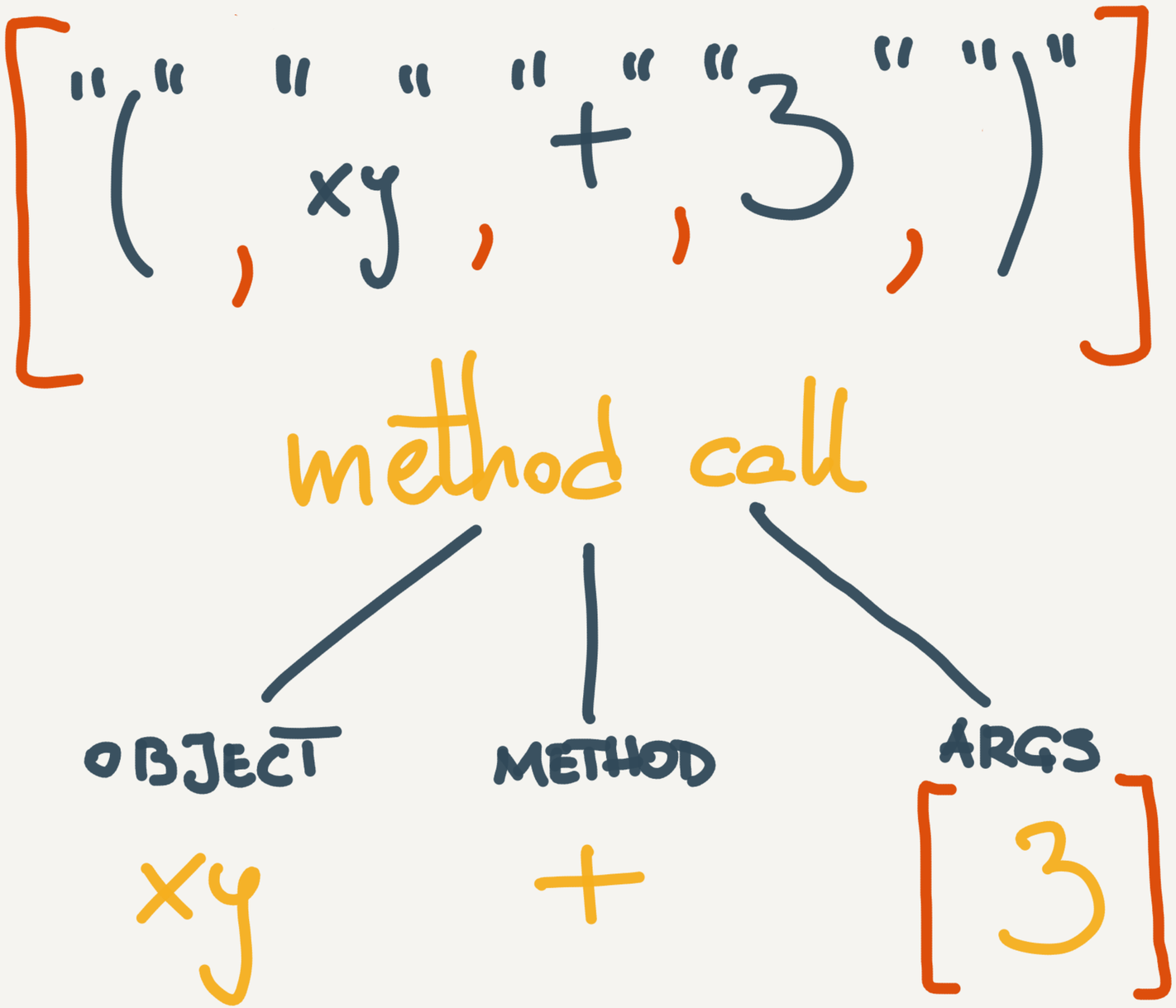
["{", "|", "'", "a", "|", "(",
"a", "+", "3", ")", "}"]

Halunke ~~~> Clojure

LEXING — stream of tokens

PARSING — syntax tree

CODEGEN — resulting code


["(" "xy" "+" "3" ")"]
method call
OBJECT
METHOD
ARGS
xy
+
[3]

"Insight: it's a lot of fun to start writing a parser, it's a lot less fun to actually finish a parser"

— @plexus

21.3.2018

Why parse properly if there's

```
(s/def ::program
  (s/+ ::expr))

;; s being clojure.spec
```

```
(s/def ::array
  (s/cat ::open  ::open-bracket
         ::elems (s/* ::expr)
         ::close ::close-bracket))

(s/def ::open-bracket
  #{[:open-bracket "["]})
```

```
(let [prog (s/conform ::program
                      tokens)]
  (if (s/invalid? prog)
    (s/explain ::program
               tokens)
    {:program prog}))
```

Halunke ~~~> Clojure

- [x] LEXING — stream of tokens
- [x] PARSING — syntax tree
- [ ] CODEGEN — resulting code

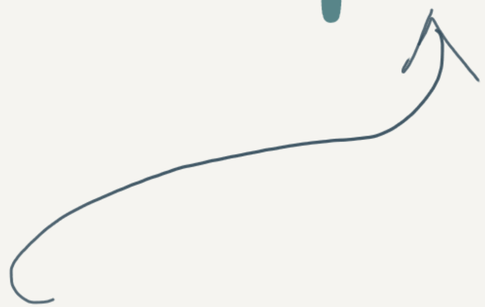
Code Generation

STARRING

Multi Methods

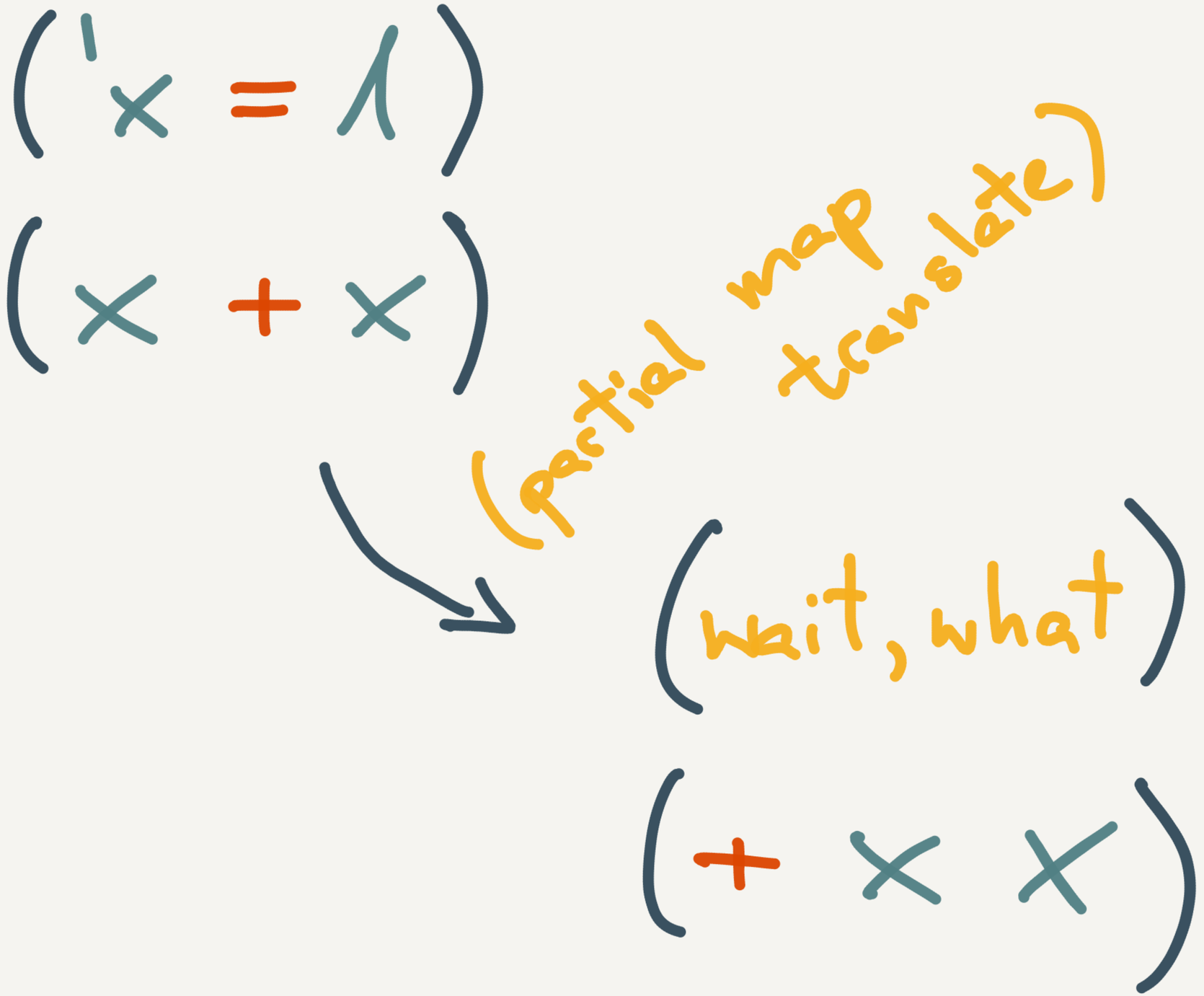
```
(defmulti translate first)

(map translate program)
```

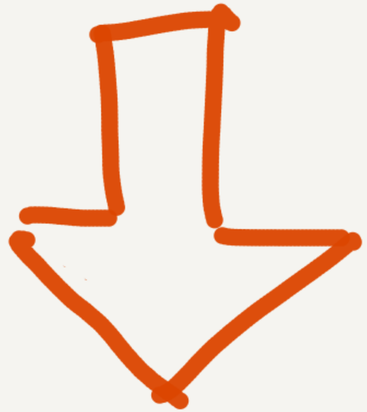


a seq of expressions


SUDDENLY
assignment?
now what?

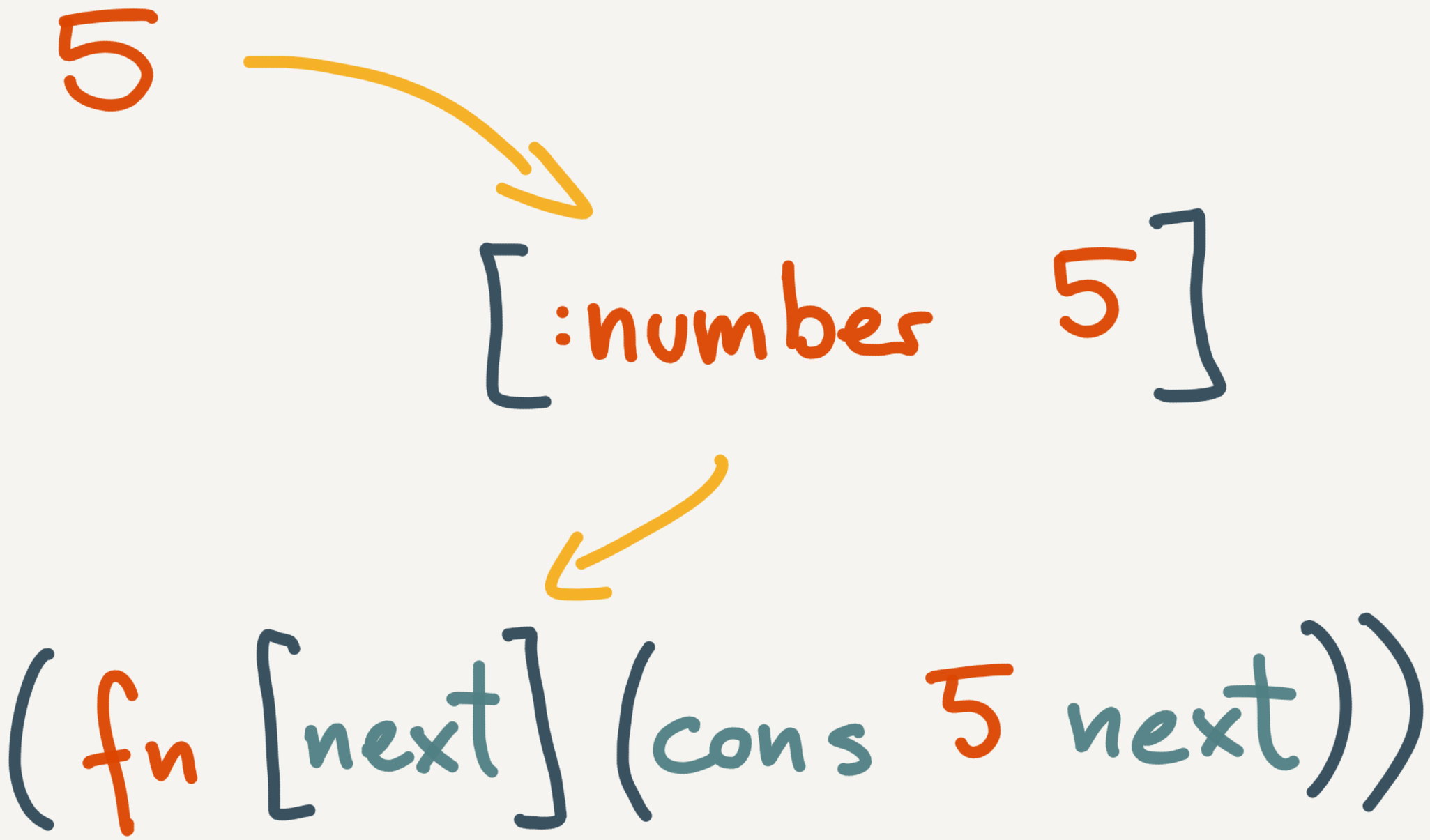

('x = 1)
(x + x)
(partial map translate)
(hit, what)
(+ x x)

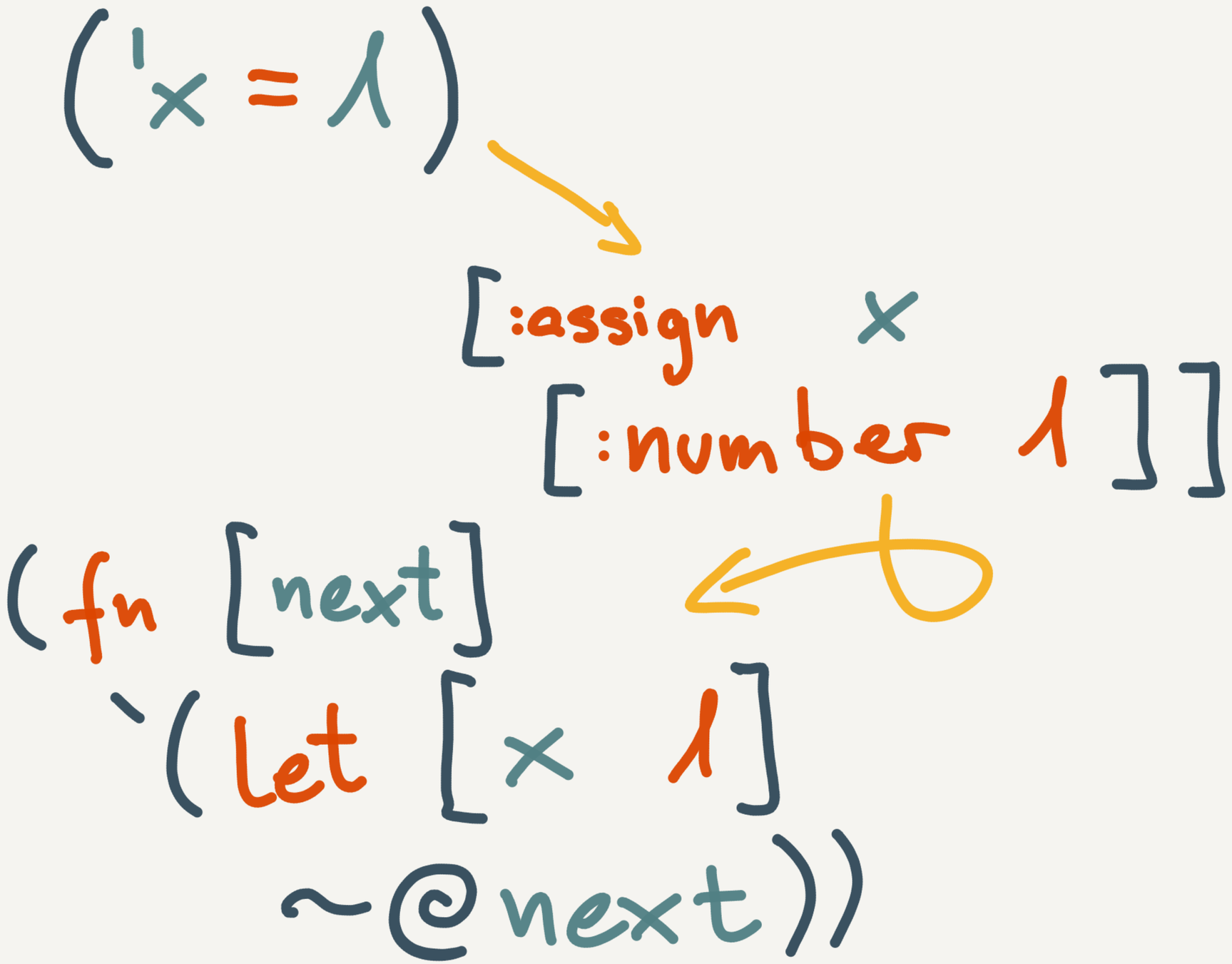
Halunke sexpr

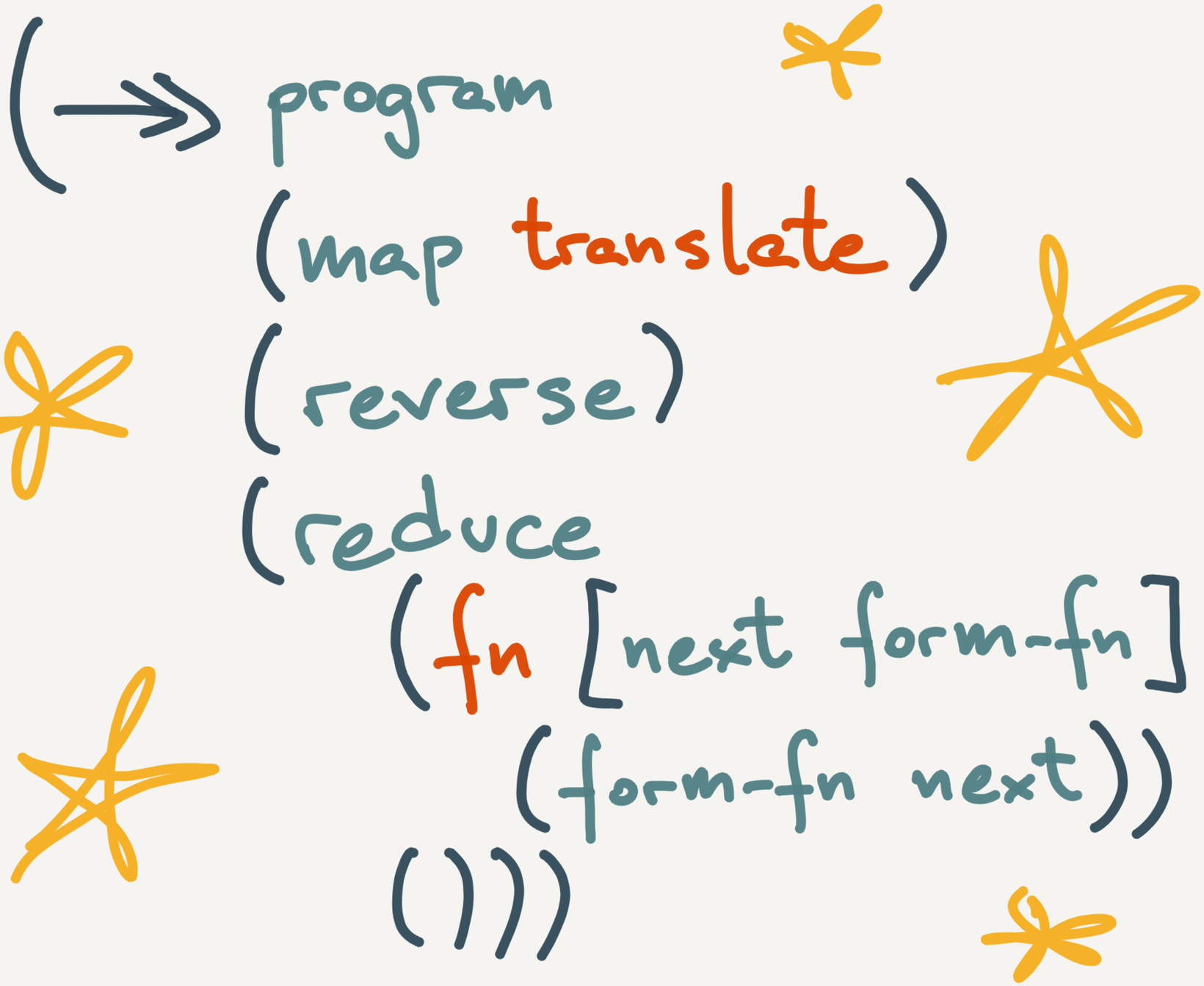


translate

```
(fn [next-clojure-expr]
  (emit-clojure-expr ...))
```


5
[:numbers 5]
(fn [next] (cons 5 next))


'x = 1
[:assign x
[:number 1]]
(fn [next]
`(let [x 1]
~@next))


(->> program
(map translate)
(reverse)
(reduce
(fn [next form-fn]
(form-fn next))
()))

Halunke ~~~> Clojure

- [x] LEXING — stream of tokens
- [x] PARSING — syntax tree
- [x] CODEGEN — resulting code

```
('foo = 3)
((foo > 0) then {"positive"}
           else {"zero or less"})
```

```
(let [foo 3]
  (then-else (> foo 0)
    (fn [] "positive")
    (fn [] "zero or less")))
```

Particularly satisfying surprise 1

clojure.spec

COMES WITH

test.check

SUPPORT OUT OF THE BOX

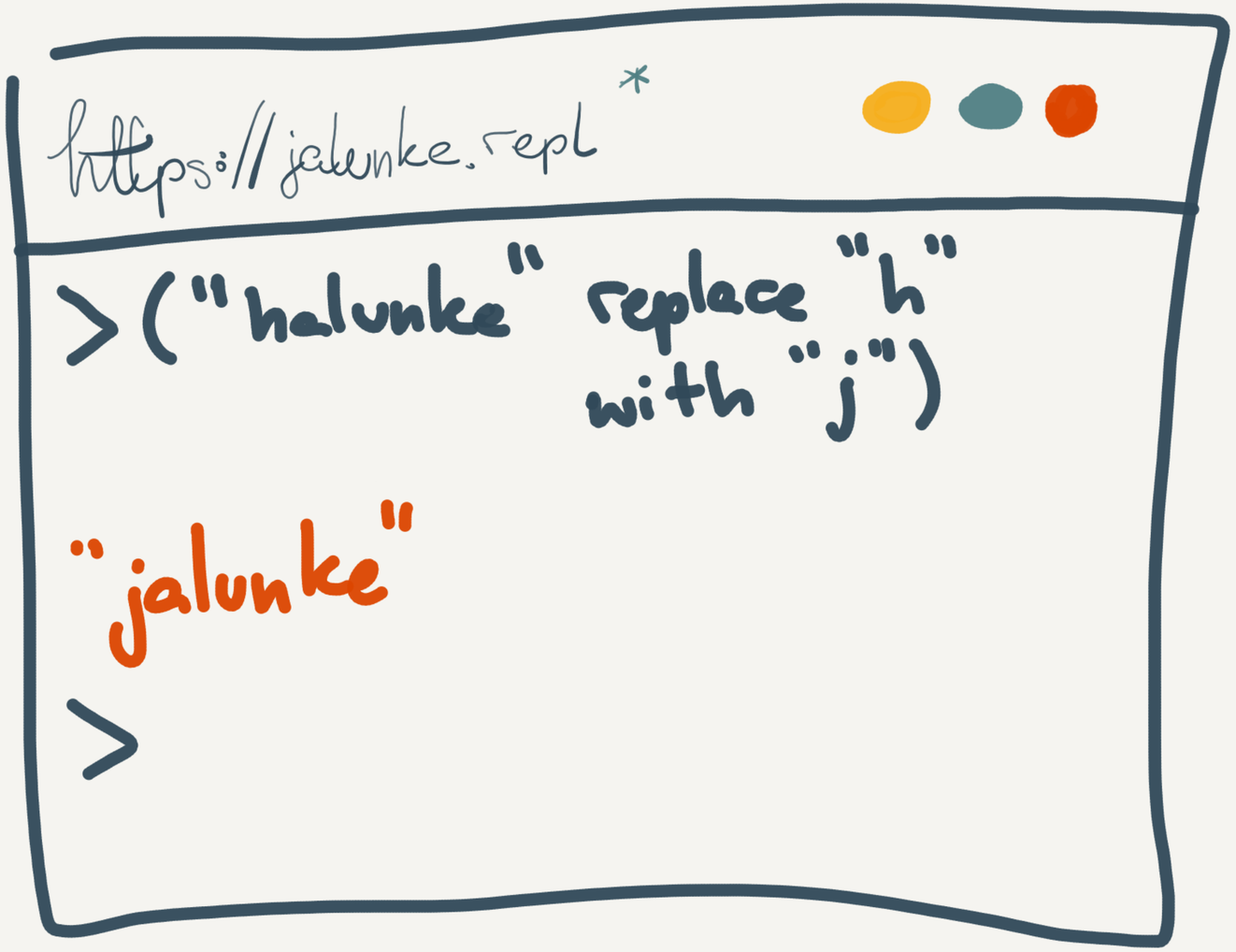
```
(for-all [tokens (s/gen ::program)]
  (parse tokens))

;; ... and hopefully return
;; something reasonable
```

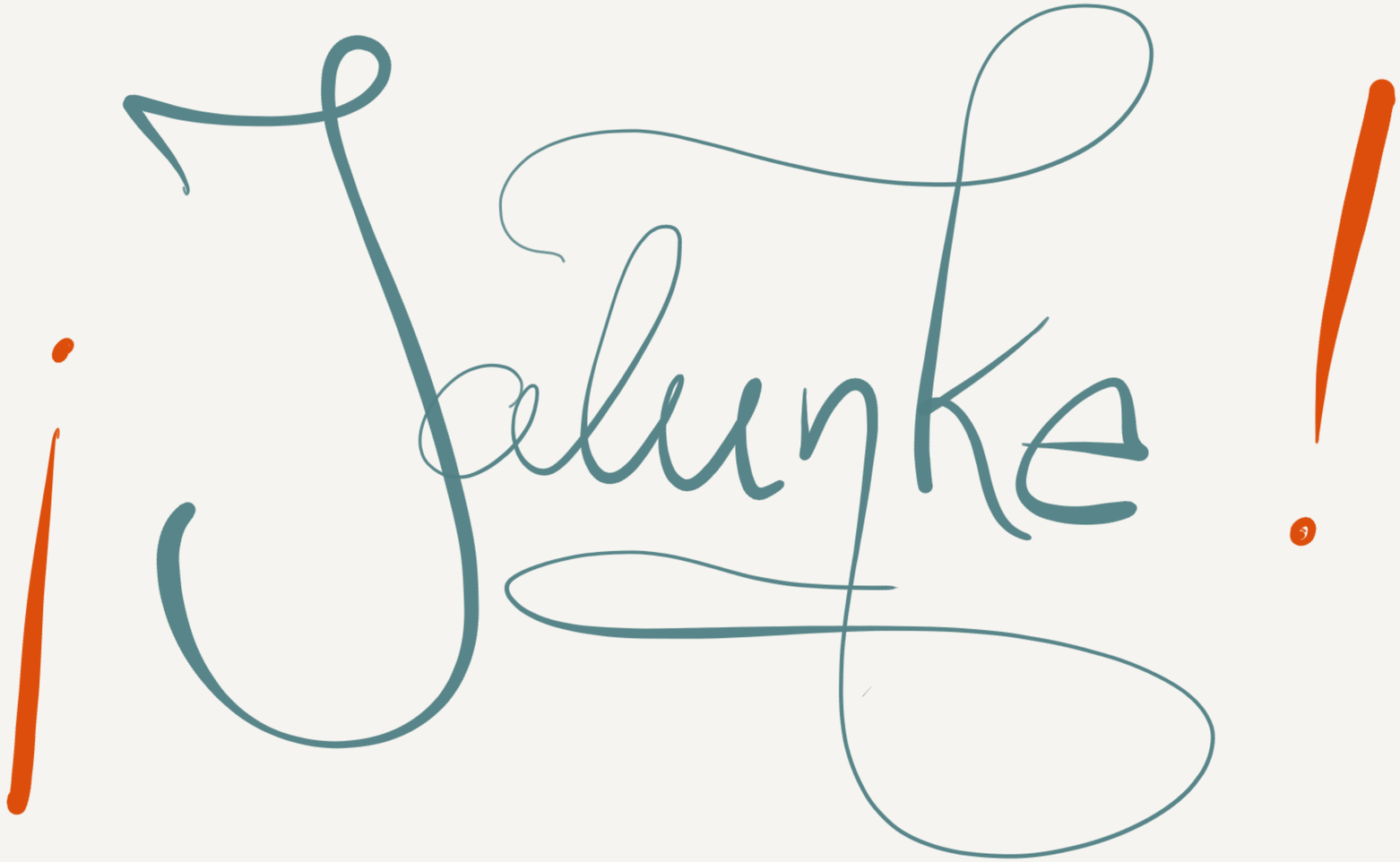
Particularly satisfying surprise 2

RELIES ONLY ON PURE

clojure-1.9.0.jar



* Serviervorschlag.



Pure Java Halunke

There are no

bad questions!

But @janstepien has perfected silly answers...

Compiling a lisp into a lisp using a lisp

with Jan Stępień

@janstepien