

# PROGRAMACION FUNCIONAL

Por qué debería  
interesarte?





Antes de arrancar...

# MOTIVACION DE *ESTA CHARLA*



Antes de arrancar...

# DEFINICION SEMI-FORMAL

# AGENDA

- Que hace a FP relevante?
- Features de FP (algunas en JS)
- CODIGO!



**FP ES RELEVANTE  
POR 3 RAZONES**

# HIPSTER



# **HIPSTER**

Closures

Generics

Garbage collection

Code as data

Monads

Lazy evaluation

Pattern Matching

Lexical Scope



# **DESMITIFICA**



# **DESMITIFICA**

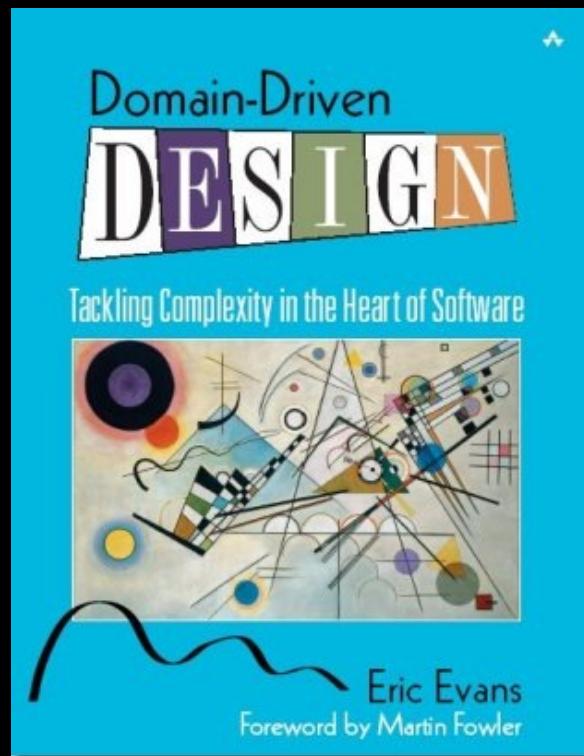


Recursión



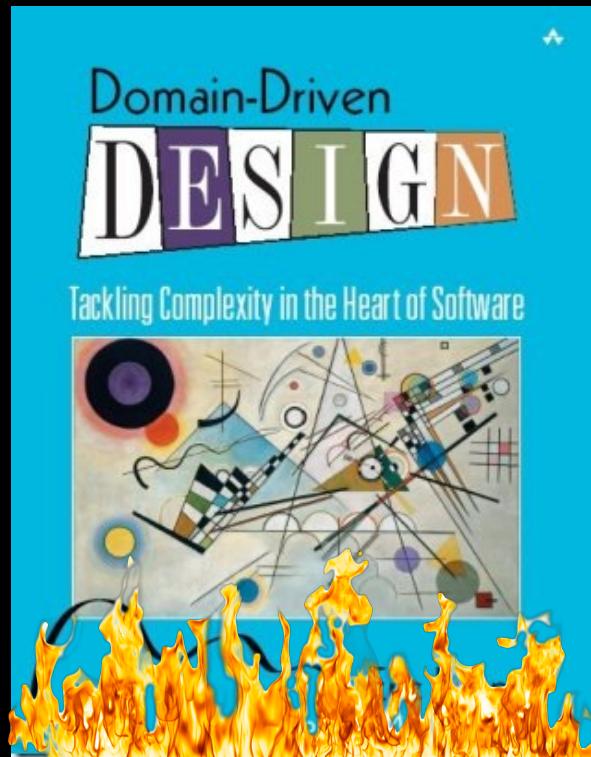
Mutabilidad

# **DESMITIFICA**



# **DESMITIFICA**

Rick Hickey  
(clojure)



# **SCALABILIDAD**

B   M   H

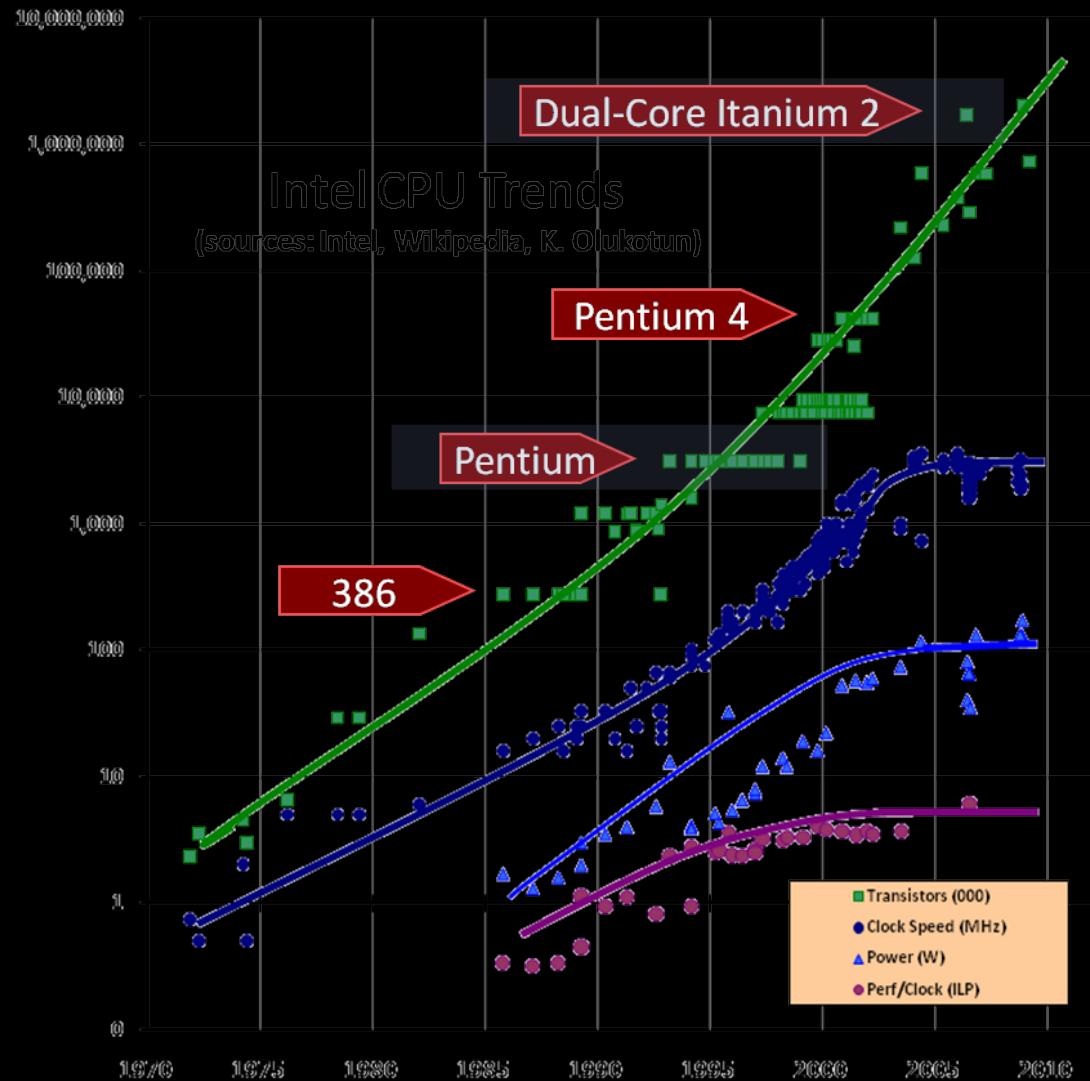


# **SCALABILIDAD**

Buy More Hardware



# SCALABILIDAD



# **SCALABILIDAD**

Launch More Instances



*FP EN ESTADO PURO  
[YTB EN JAVASCRIPT]*

# LEXICAL SCOPE

```
(* Takes a date and returns a string of the
form January 20, 2013 (for example). *)
fun date_to_string (date: int * int * int) =
  let
    val month_names = ["January", "February", "March",
                       "April", "May", "June", "July", "August",
                       "September", "October", "November", "December"]
    val month = get_nth(month_names, #2 date)
    val day = Int.toString (#3 date)
    val year = Int.toString (#1 date)
  in
    month ^ " " ^ day ^ ", " ^ year
  end
```



# LEXICAL SCOPE

```
(* Takes a date and returns a string of the
form January 20, 2013 (for example). *)
fun date_to_string (date: int * int * int) =
  let
    val month_names = ["January", "February", "March",
                       "April", "May", "June", "July", "August",
                       "September", "October", "November", "December"]
    val month = get_nth(month_names, #2 date)
    val day = Int.toString (#3 date)
    val year = Int.toString (#1 date)
  in
    month ^ " " ^ day ^ ", " ^ year
  end
```

```
var x = 2, y = 3;
function someAdd() {
  var x = 4;
  return x + y;
}
someAdd();
```



# CLOSURES

```
(* Takes a string list and returns a string list that has only  
   the strings in the argument that start with an uppercase letter. *)  
val only_capitals =  
  List.filter (fn x => Char.isUpper(String.sub(x, 0)))
```



# CLOSURES

```
(* Takes a string list and returns a string list that has only  
| the strings in the argument that start with an uppercase letter. *)  
val only_capitals =  
  List.filter (fn x => Char.isUpper(String.sub(x, 0)))
```

```
http.get("www.google.com", function (res) {  
  ....  
  res.on('data', function(chunk) {  
    ...  
  }).on('end', function() {  
    ....  
  });
```



# FIRST ORDER FUNCTIONS

```
applyTwice :: (a -> a) -> a -> a
```

```
applyTwice f x = f (f x)
```

```
map :: (a -> b) -> [a] -> [b]
```

```
map _ [] = []
```

```
map f (x:xs) = f x : map f xs
```

```
ghci> map (+3) [1,5,3,1,6]
```

```
[4,8,6,4,9]
```

```
ghci> map (negate . sum . tail) [[1..5],[3..6],[1..7]]
```

```
[-14,-15,-27]
```



# FIRST ORDER FUNCTIONS

```
applyTwice :: (a -> a) -> a -> a
```

```
applyTwice f x = f (f x)
```

```
map :: (a -> b) -> [a] -> [b]
```

```
map _ [] = []
```

```
map f (x:xs) = f x : map f xs
```

```
ghci> map (+3) [1,5,3,1,6]
```

```
[4,8,6,4,9]
```

```
ghci> map (negate . sum . tail) [[1..5],[3..6],[1..7]]
```

```
[-14,-15,-27]
```

fn.apply

\_ . map...

\_ . compose



# CURRYING

```
multThree :: (Num a) => a -> a -> a -> a
multThree x y z = x * y * z
```

```
ghci> let multTwoWithNine = multThree 9
ghci> multTwoWithNine 2 3
54
ghci> let multWithEighteen = multTwoWithNine 2
ghci> multWithEighteen 10
180
```



# CURRYING

```
multThree :: (Num a) => a -> a -> a -> a  
multThree x y z = x * y * z
```

```
ghci> let multTwoWithNine = multThree 9  
ghci> multTwoWithNine 2 3  
54  
ghci> let multWithEighteen = multTwoWithNine 2  
ghci> multWithEighteen 10  
180
```

\_.partial...



# ***CODE AS DATA***

```
(define (sequence low high stride)
  (if (> low high)
      null
      (cons low (sequence (+ low stride) high stride))))
```

```
; sequence test
(check-equal? (sequence 0 5 1) (list 0 1 2 3 4 5) "Sequence test")
```



# **CODE AS DATA**

```
(define (sequence low high stride)
  (if (> low high)
    null
    (cons low (sequence (+ low stride) high stride))))
```

```
; sequence test
(check-equal? (sequence 0 5 1) (list 0 1 2 3 4 5) "Sequence test")
```

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New",
       "onclick": "CreateNewDoc()"}]
```





# **CONCLUSIONES**

I'M NOT SAYING QUIT JS AND GO  
HASKELL...

BUT QUIT JS AND GO  
HASKELL



memegenerator.net



# **LANGUAGES**

Static: ML (primero), Haskell

Dynamic: Racket, Clojure, Scheme (i.e. un Lisp)

Scala???



# **LIBRERIAS**

Q (<https://github.com/kriskowal/q>)

RSVP (<https://github.com/tildeio/rsvp.js>)

FutureJS (<https://github.com/FuturesJS/FuturesJS>)

Promised-io (<https://github.com/kriszyp/promised-io>)

Bacon(<https://github.com/baconjs/bacon.js>)

Fn.js(<http://eliperelman.com/fn.js/>)

Wu (<http://fitzgen.github.io/wu.js/>)

Fnky (<https://github.com/leoasis/fnky>)

Prelude (<http://preludels.com/>)



# **VER Y LEER**

“Hey Underscore, you’re doing it wrong”

<http://www.youtube.com/watch?v=m3svK0dZijA>

“How to survive asynchronous programming in JS”

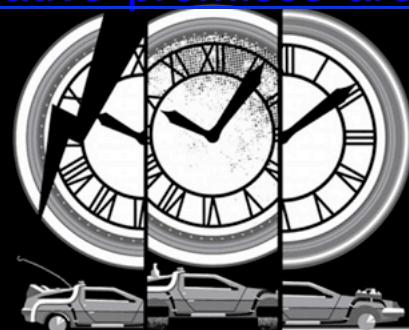
<http://www.infoq.com/articles/surviving-asynchronous-programming-in-javascript>

“Callbacks are imperative, Promises are Functional”

<https://blog.jcoglan.com/2013/03/30/callbacks-are-imperative-promises-are-functional-nodes-biggest-missed-opportunity/>

“Promises/A spec”

<http://wiki.commonjs.org/wiki/Promises/A>



# **TEMAS**

Monads

Currying/Partial/Compose/Point-free

Beneficios de no mutar

Promises/Futures

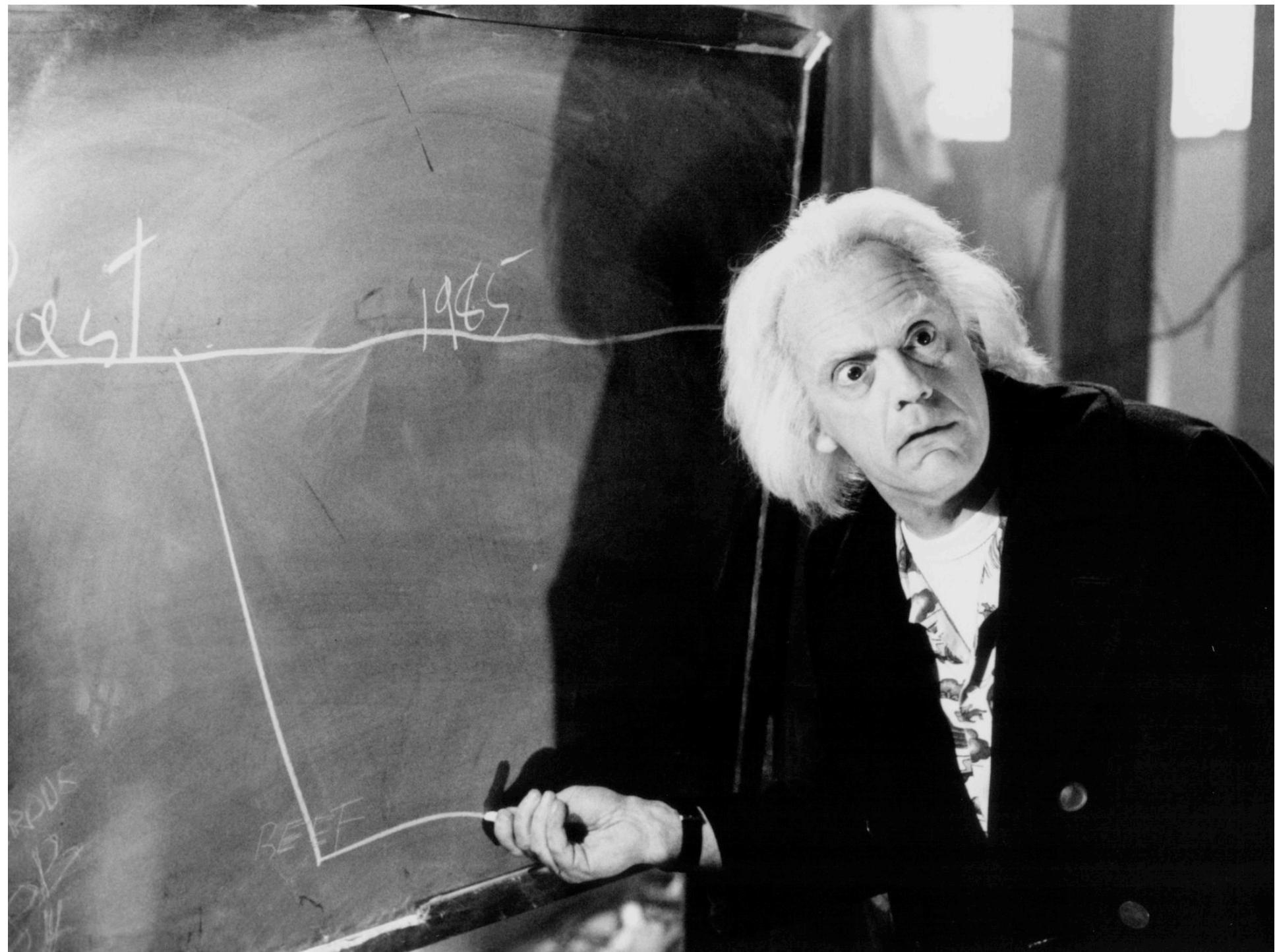
Subtyping/Generics

Lazyness

Librerías

Soporte nativo





[github.com/holden-caulfield/](https://github.com/holden-caulfield/)  
@jpsaraceno

