Visi

David Pollak
January 18th, 2012

Visi: What?

Visi: What?

Beautiful & Literate Code

Visi: What?

Beauti

Code

Radically Improve Programming:
Make it Beautiful and
Understandable

Visi: What?

Beau Code

Radically Im e Programming:
Make it B tiful and
Underst able

Pure, Lazy Functional Language

Visi: What?

Pure, Lazy F e Programming:

Bea iful and

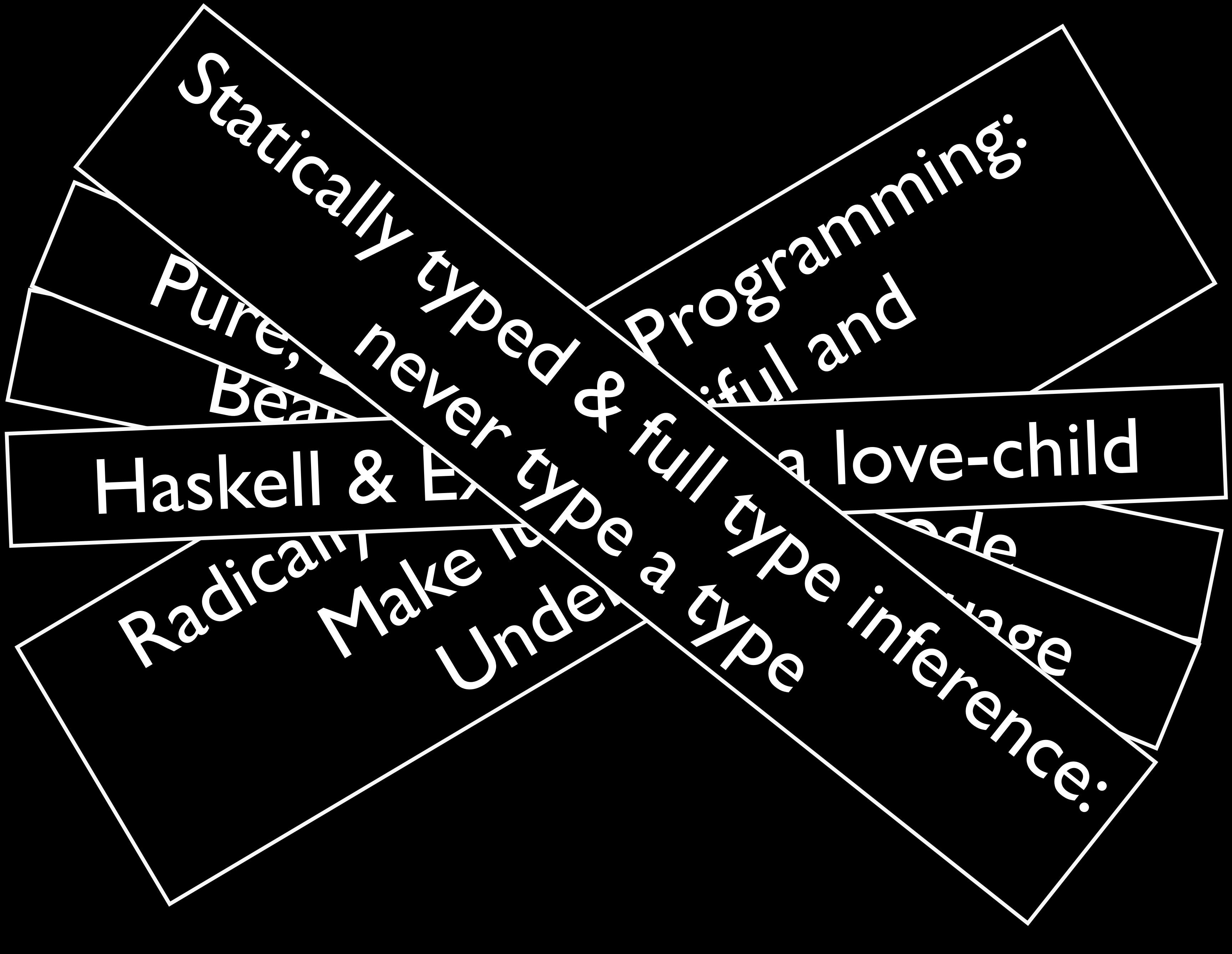
Haskell & Excel have a love-child

Radicall

Make it ode

Underst al Language

Visi: What?



Statically typed & full type inference:

never type a type

Pure,

Bea

Programming:

iful and

Haskell & E

a love-child

ode

uage

Radicall

Make lu

Unde

Visi: What?

Simple for Excel power users and PHP developers

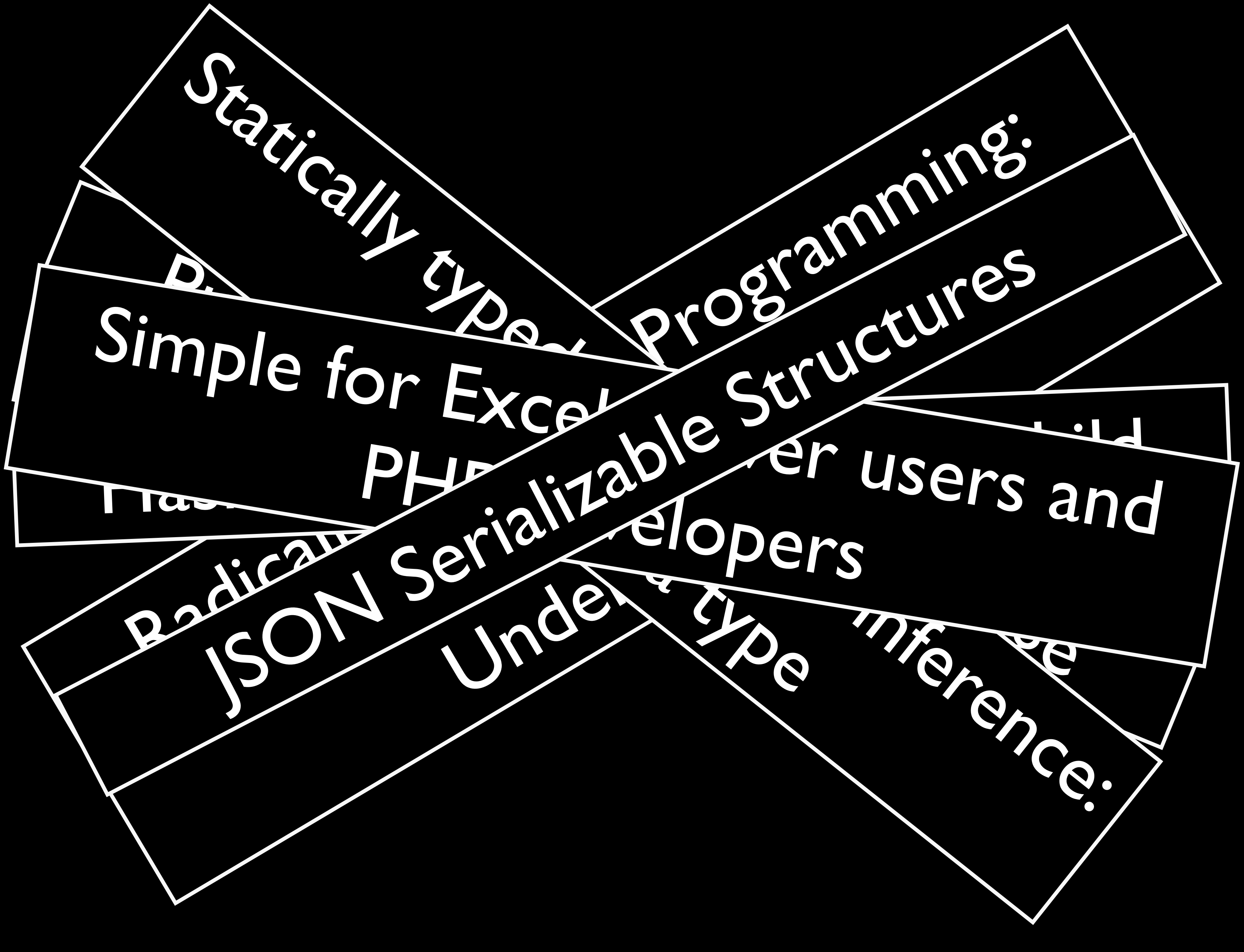
Statically typed

Programming:
ful and

Radically
Make
Unde
type
inference:

Has

Visi: What?



Statically typed

Programming:

Simple for Excel er users and

PHP elopers

JSON Serializable Structures

Unde type

nference:

Has

Radical

ild

pe

Visi: What?

Statically typed

Programming:

Open Classes & Structural Typing

Simple

Structures

PH

er users and

ild

Has

Radical

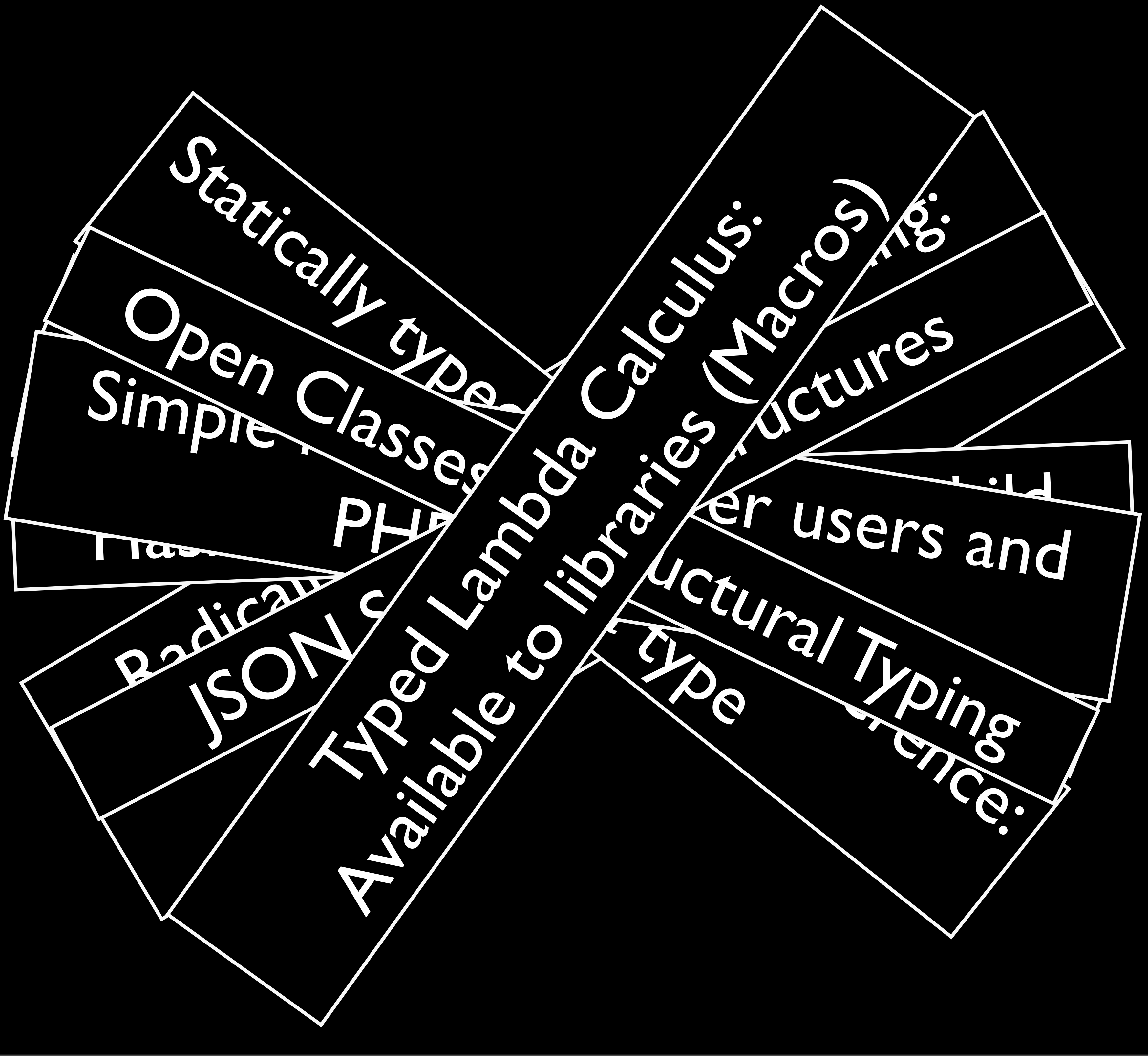
JSON Serializ

Unde

type

rence:

Visi: What?



Typed Lambda Calculus:
Available to libraries (Macros)

Statically type

Open Classes

Simple

PH

Has

Radical

JSON S

ng:

uctures

ild

er users and

uctural Typing

type

rence:

Visi: What?

Statically typed

Open C

Radica

JSON

Typed L
Available to

Calculus:
s (Macros)

ng:

uctures

al Typing

type

rence:

Dynamic dev, compiled for performance, self hosted

Visi: What?

Statically typed

Calculus:

Open Cl

Dynamic dev mpiled for
perform , self hosted

uctures

Migratory Computation Locus

Radica

JSON

Available to type

al Typing

erence:

Visi: What?

Programming
for the
Rest of Us

Visi: Why?

Visi: Why?

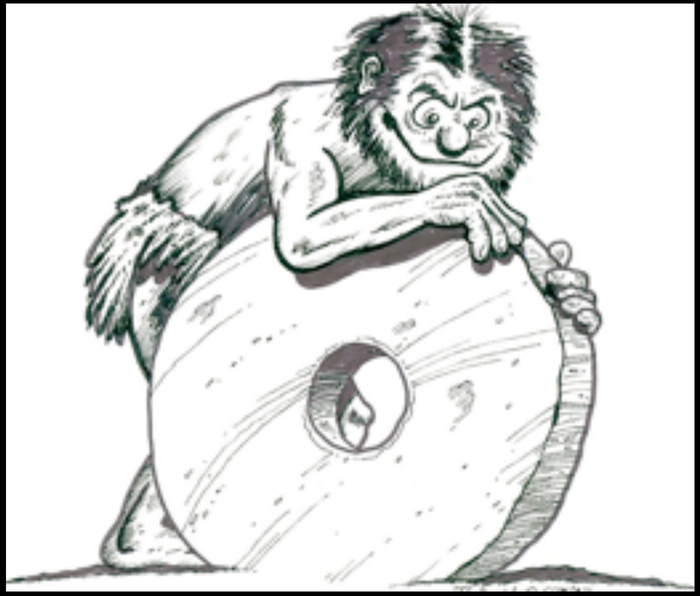
Visi: Why?

- Programing tools have not kept up

Visi: Why?

- Programing tools have not kept up
- iOS development is significant pain

Visi: Why?



- Programing tools have not kept up
- iOS development is significant pain
- Economics of software is broken

DPP: Who?

DPP: Who?

DPP: Who?

DPP: Who?

- Wrote first real-time spreadsheet: Mesa

DPP: Who?

- Wrote first real-time spreadsheet: Mesa
- Founded Lift Web Framework

DPP: Who?

- Wrote first real-time spreadsheet: Mesa
- Founded Lift Web Framework
- Always writing languages: mostly trivial

Visi: When?

Visi: When?

- Announced Nov, 17th, 2011

Visi: When?

- Announced Nov, 17th, 2011
- Work in progress

Visi: When?

- Announced Nov, 17th, 2011
- Work in progress
- Beta in 2012

Visi: Where?

Visi: Where?

- http://visi.io

Visi: Where?

- http://visi.io
- http://blog.visi.io

Visi: Where?

- http://visi.io
- http://blog.visi.io
- http://github.com/visi-lang/visi

Visi: Where?

- http://visi.io
- http://blog.visi.io
- http://github.com/visi-lang/visi
- http://groups.google.com/group/visi-lang

Visi: How?

Visi: How?

- Haskell core but mostly Visi

Visi: How?

- Haskell core but mostly Visi
- Open source with a business model

Visi: How?

- Haskell core but mostly Visi
- Open source with a business model
- Razor balance: BDFL & Community

Hello, World!

- `“Greeting” = “Hello, World!”`

Sum

-
  ```
  ?number1
  ?number2
  “Sum” = number1 + number2
  ```

Sum & Product

-
  ```
  ?number1
  ?number2
  “Sum” = number1 + number2
  “Product” = number1 * number2
  ```

Factorial

-
  ```
  ?number

  fact n =
    if n == 0 then 1
      else n * fact (n - 1)

  “Factorial” = fact number
  ```

Real World

-
  ```
  ?taxable
  ?nonTaxable
  ?taxRate

  tax = taxable * taxRate
  subtotal = taxable + nonTaxable
  total = subtotal + tax

  “Subtotal” = subtotal
  “Tax” = tax
  “Total” = total
  ```

Keep in mind

- Structural typing like OCaml
- Single level subtyping
- All (almost) immutable data structures
- Open data structures and classes
- No modification of Type Constructors

Data Structures

- `struct Bool2 = True | False`

Data Structures

- `struct Dog(String)`

Data Structures

- `struct Cat(name: String)`

Data Structures

-
  ```
  struct Thing = This(String) |
                 That(when: Date)
  ```

Data Structures

-
  ```
  struct Person(String, age: Int) =
    Kid() |
    Parent(kids: [Person])
  ```

Pattern Matching

- Type Constructor/Extractor
  - `dogsName Dog(name) = name`
  - `kidsName Kid(name, _) = name`

Pattern Matching

- Nominal (anything with the property `name`)
-
  ```
  name (name => theName) = theName
  name2 (name =>) = name
  ```

Pattern Matching

- Positional (anything in the first position of a product type with a single param constructor)

-
  ```
  something (thing) = thing
  // require a String
  someString (str: String) = str
  ```

Pattern Matching

- Tests `name` is “fred”
- Non-exhaustive means `Box`

-
  ```
  fredsAge (name == “fred”,
                => age: Int) =
                   age // Box Int
  ```

Pattern Matching

- Tests `name` is “fred” for a `Person`
- Non-exhaustive means `Box`
-
  ```
  fredsAge2 Person(name == “fred”,
                   => age) =
              age // Box Int
  ```

Functions

- Structural Typing

-
  ```
  anyAge n = n.age
  anyAge2 = #age // curried
  ```

Define Methods

- Methods on a type

-
  ```
  struct Foo(age: Int)
    methods
      old? = self.age > 85
      addToAge n = self.age + n

  testOld n = n.old?
  testOld2 = #old?
  ```

Updators

- How to create a new instance?

  -
    ```
    kid = Kid “Daniel” 7
    birthday = kid.=age 8
    nextYear n =
      curAge = n.age
      n.=age (curAge + 1)
    makeOld = #=age 86
    ```

Updators

- Via function

  -
    ```
    kid = Kid “Daniel” 7

    nextYear kid = kid.>age (+ 1)
    nextYear2 = #>age (+ 1)

    nextYear2 kid // Kid “Daniel” 8
    ```

Precursors

- Mixins with attitude
-
  ```
  precursor TestAge
    data
      old? = olderThan 85
    methods
      olderThan2 n = self.age > n

  enhance Person with TestAge
  ```

Sources & Sinks

- I/O happens here

  -
    ```
    ?age // input the age

    “one year older” = age + 1 // out
    ```

Sources & Sinks

- Accumulation
-
  ```
  ?age // input the age

  allAges = age:allAges // collect
  ageCnt = length allAges

  “age count” = ageCnt
  “average” = (sum allAges)/ageCnt
  ```

References

- Clojure-like
- Computation delineation points
- No syntax or semantics, yet (waves hands)

More unfinished stuff

- Modules/packages/dependency mgt
- Visibility
- Code signing/execution rights
- Library mode (access to types and mutability and stuff)

End

- Questions