Bibliotech - smart cafe

Programming Languages: Some news for the last N years.

Ruslan Shevchenko
<ruslan@shevchenko.kiev.ua>

@rssh1
https://github.com/rssh

Programming languages:

- Industry / Academic
  - what's hot now:
    - academic research from 70-th
    - 'freaky' things from 2000

Where is progress ?

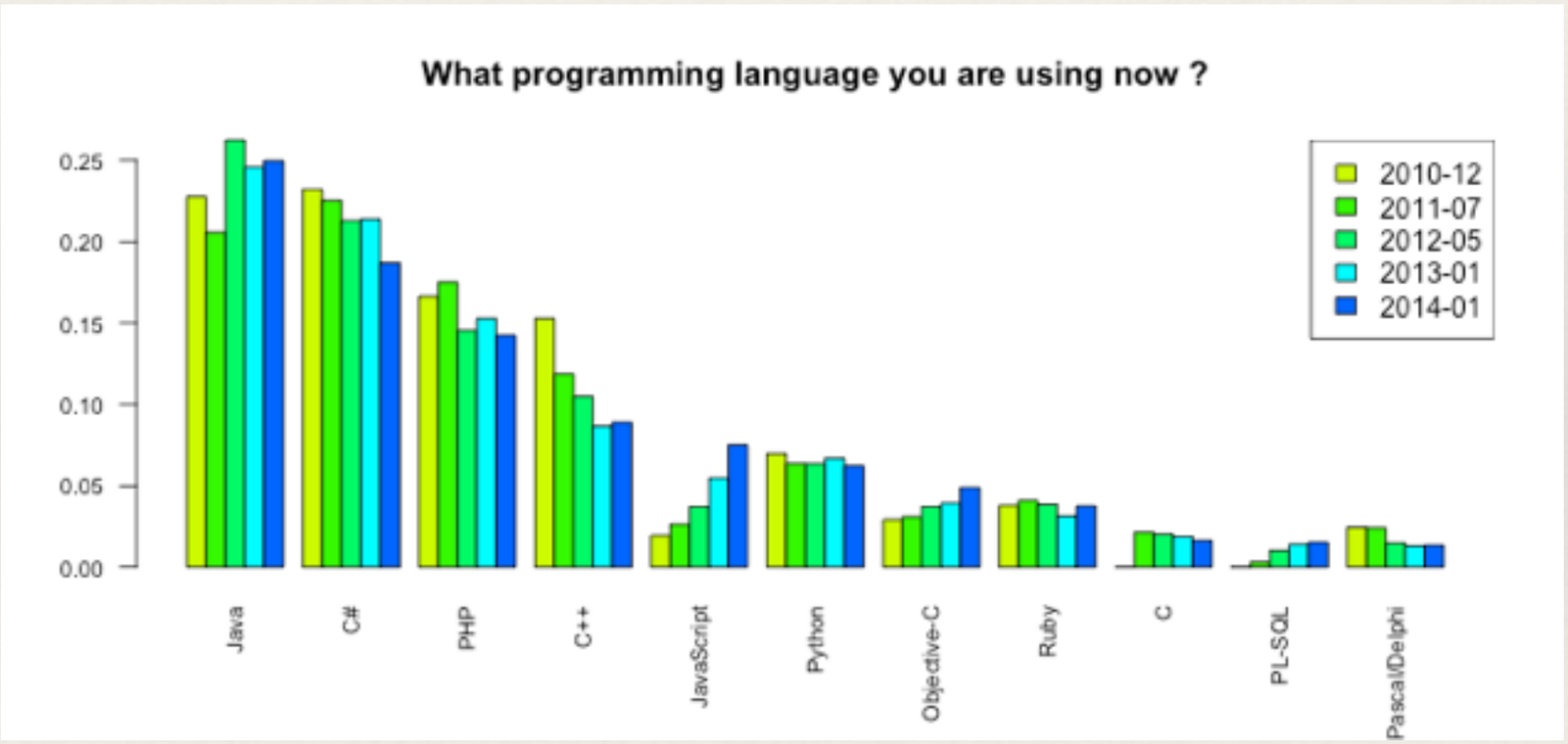
What we will see in industry from today-s academic ?

from today-s freaks ?

Statement from 80-th:

Full change of language and tools during 2-3 years

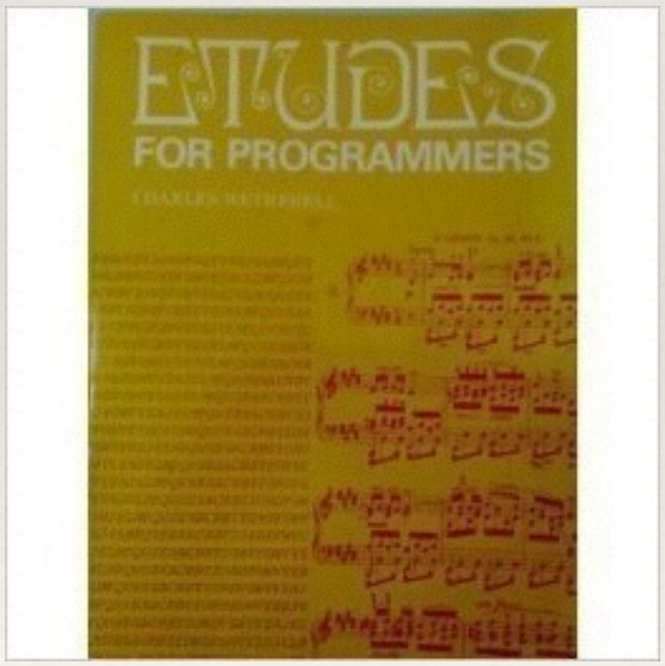
http://dou.ua



Actual situation in 2014:

Top 11 languages are the same: 2010 - 2014

Can we track progress in PL ?



- Estimations
- Code size / project duration

Evolution is slow

If we think about ideal language ?

Make all good

What is:

- all ?
- good ?
- make ?

- Object and processes from real world,
  represented in computer
- Operations with this representation.
- Mapping of result back to real world

Syntax [parsing/ast manipulation]

Semantics [type systems]

"New" functionality

Syntax

Traditional approach: context free grammars

Syntax structure is fixed

Now in practice:

Context-dependent keywords (C++)

Multi-language programming

Extensible syntax

Seed 7: Syntax definitions of operations with priority

```
syntax expr: .(). + .()   is -> 7;
```

```
syntax expr: .loop.().until.().do.().end.loop   is -> 25;
```

Nemerle: macroses

```
macro for (init, cond, change, body)
syntax ("for", "(", init, ";", cond, ";", change, ")", body)
```

XL Compiler plugins

Extensible syntax

// Idris

```
boolCase : (x:Bool) -> Lazy a -> Lazy a -> a;
boolCase True t e = t;
boolCase False t e = e;

syntax "if" [test] "then" [t] "else" [e] = boolCase test t e;
```

```
syntax "for" {x} "in" [xs] ":" [body] = forLoop xs (\x => body)
```

// introducing new binding

```
// DSL — map object to Idrix language constructions
```

XL: Concept programming

Active library = Library + compiler plugin

Concept in real life. In computer - representation

Metrics:

Syntax noise / signal what part of code is about business

Bandwidth what part of concept is represented

Complexity source

Articulate programming

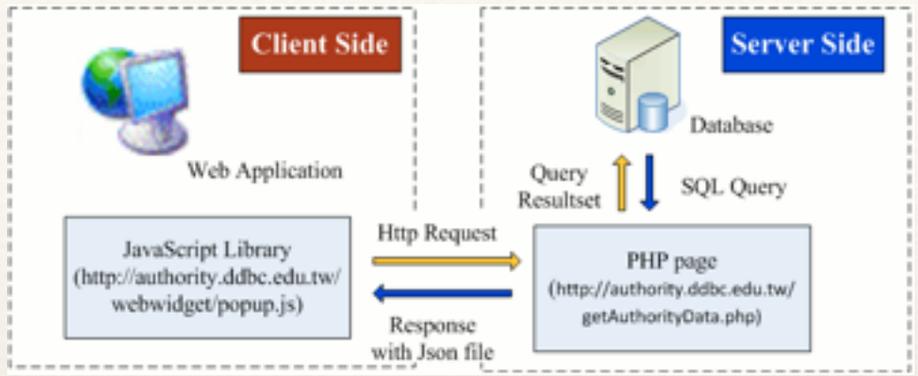
Avail: SmallTalk-based, static typing

Try to deduce ‘needed’ value from context.

like scala implicit, but *all* is implicit

Syntax

Multi-language programming



- Do nothing (mess)
- Embed all in one language [lifted evaluation]
  - (like LINQ or scala slick or idris DSL)
- Define and use ‘part of grammar’ with type.

Wyvern: https://www.cs.cmu.edu/~aldrich/wyvern/

```
                                        //Type-specific tokens

    let imageBase : URL = <images.example.com>
    let bgImage : URL = <%imageBase%/background.png>
    new : SearchServer
        def resultsFor(searchQuery, page)
            serve(~) (* serve : HTML -> Unit *)
>html
  >head
    >title Search Results
  >style ~
    body { background-image: url(%bgImage%) }
    #search { background-color: ..
```

Wyvern: https://www.cs.cmu.edu/~aldrich/wyvern/

```
                                        //Type-specific tokens
let imageBase : URL = <images.example.com>
let bgImage : URL = <%imageBase% /background.png>
new : SearchServer
    def resultsFor(searchQuery, page)
        serve(~) (* serve : HTML -> Unit *)
          >html
           >head
             >title Search Results
          >style ~
            body { background-image: url(%bgImage%) }
            #search { background-color: ..
```

Wyvern: https://www.cs.cmu.edu/~aldrich/wyvern/

```
                                                    //Type-specific tokens
let imageBase : URL = <images.example.com>
let bgImage : URL = <%imageBase% /background.png>
new : SearchServer
    def resultsFor(searchQuery, page)
        serve(~) (* serve : HTML -> Unit *)
            >html
              >head
                >title Search Results
            >style ~
                body { background-image: url(%bgImage%) }
                #search { background-color: ..
```

Type Systems

Static/Dynamic —-> Gradual

Dynamic:

- clojure -> core.typed
- python -> type annotations
- groovy -> type annotations

Static:

- Dynamic type

fromDynamic[T]: D => Option[T]
toDynamic[T](x: T) => Dynamic

needed for reflective typing

Clojure: core/typed.

```
(ann clojure.core/first
  (All [x]
    (Fn [(Option (EmptySeqable x)) -> nil]
        [(NonEmptySeqable x) -> x]
        [(Option (Seqable x)) -> (Option x)]))
)
```

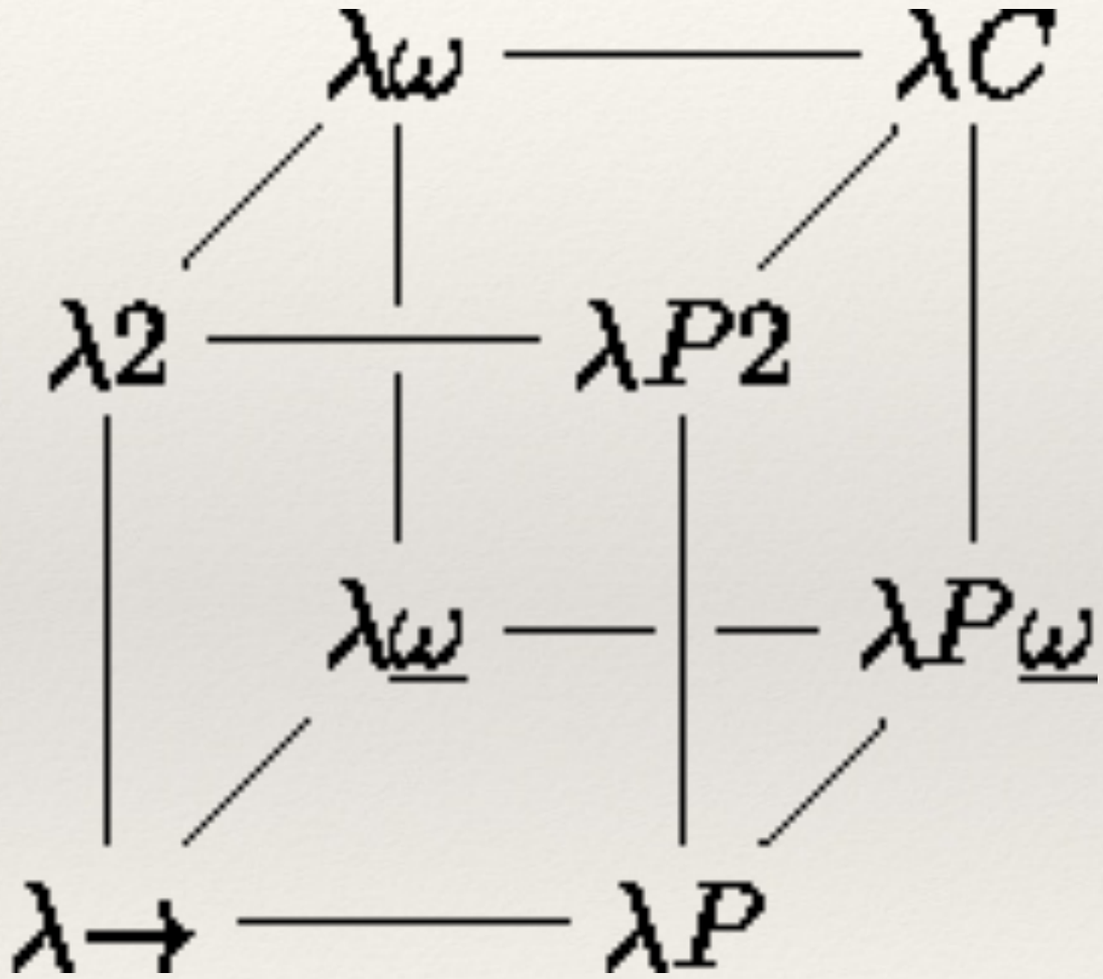
- static empty -> nil
- static non-empty -> not-nil
- unknown -> x or nil,

Type Systems

typing => static analysis at compile-time

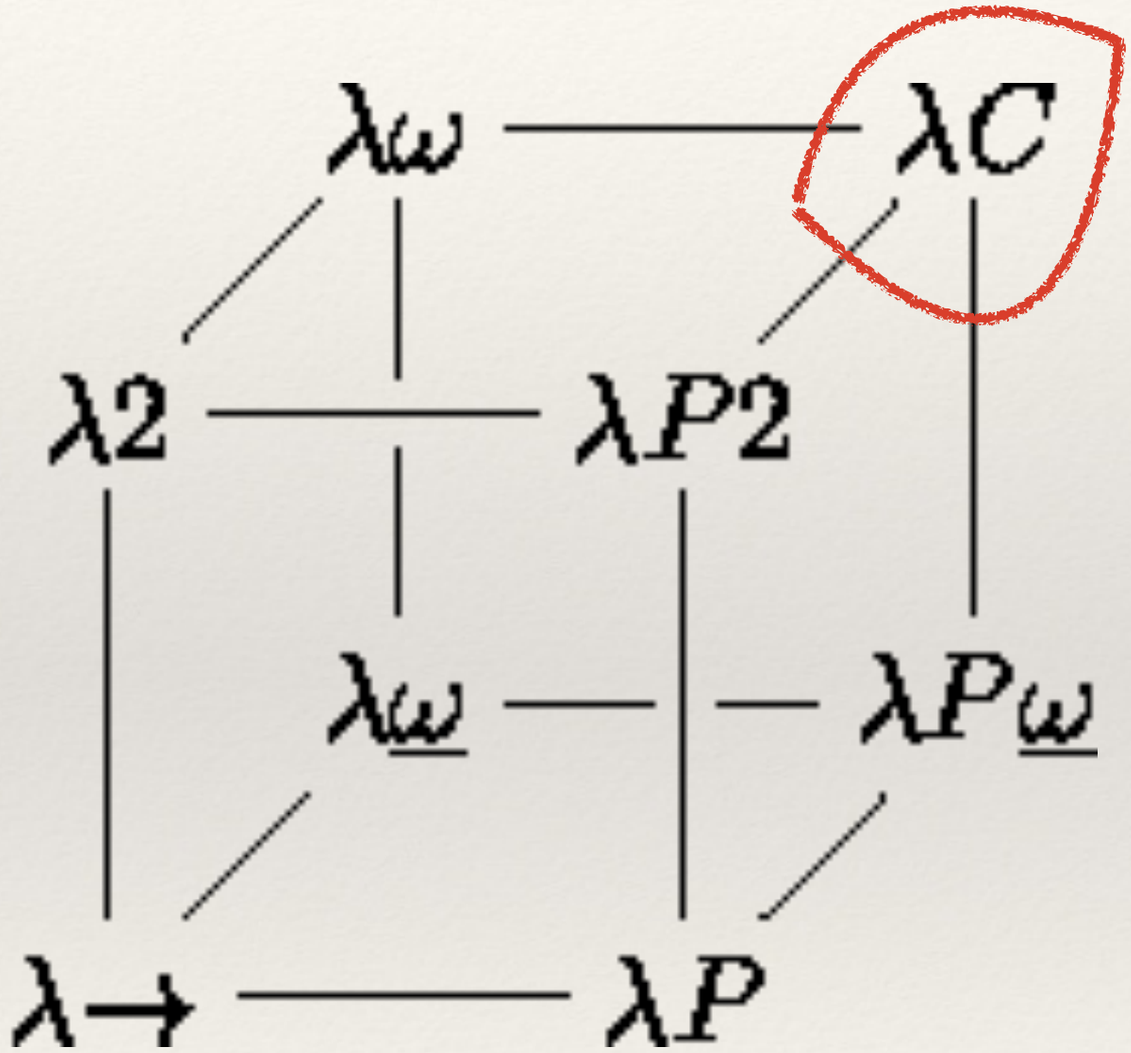
compilation => optimization of interpretation

Type Systems



Type Systems

Dependent types



Non-dependent types example

```
trait CInt  { type Succ <: CInt }
trait CPos extends CInt                    // church numeration

trait CSucc[P <: CPos] extends CInt  {
    type Succ = CSucc[CSucc[Int]]
}

final class _0 extends CPos {
    type Succ = CSucc[0]
}

type _1 = _0#Succ
type _2 = _1#Succ
```

Non-dependent types example

```
trait LimitedVector[+A, N<:CInt]
{

  ..............

  def  append[B <: A, NX <:CInt](x: LimitedVector[B,NX]):
                                          LimitedVector[B,Plus[N,NX]]

  ..............
}
```

// shapeless full of such tricks

Dependent types example

```
trait LimitedVector[+A, n:Int]
{
   ..............
  def  append[B <: A, nx:Int](x:  LimitedVector[B,nx]):
                                                  LimitedVector[B, n+nx]
   ..................
}
```

// warning: pseudocode, impossible in scala

Dependent types example

```
data Vect : Nat -> Type -> Type                    // Idris
     where
          Nil : Vect 0 a
          (::) : a -> Vect k a -> Vect (k+1) a
```

Calculus of constructions: Idris, Coq, Agda

Dependent types:

Track properties, which can be proven by structural induction

Function, which call SQL inside no more than 3 times.

Resource with balanced acquire/release for all execution paths.

Type depends from value <=> Type must be available as value

```
data Parity : Nat -> Type where
     even :  Parity (n + n)
     odd :   Parity (S (n + n))
```

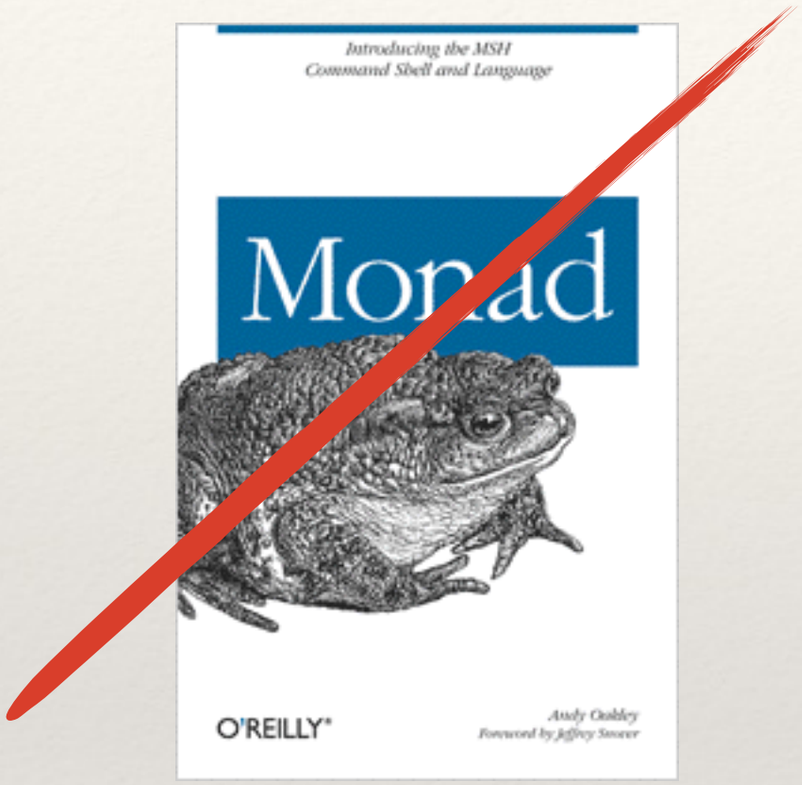
Dependent types:

intuitionistic logic

absence (p || ~p)

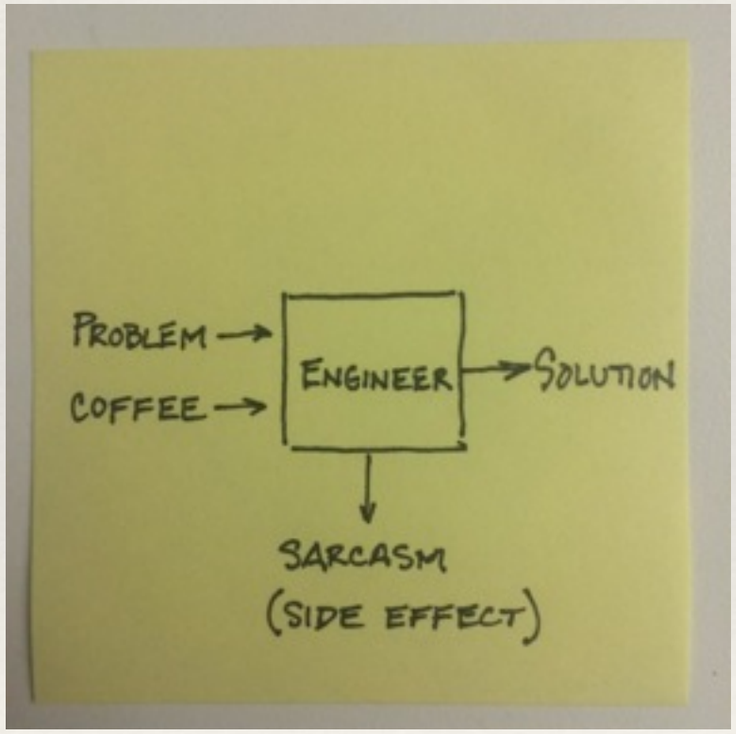
Type:Type - logical bomb

Effects tracking:



No more IO monads !!!!

List of effects as part of type.



- Idris
- Eff

Knowledge-based programming

Let's add structured information about problem domain.

Wolfram Alpha:

Mathematics domain. Near 5000 built-in functions.

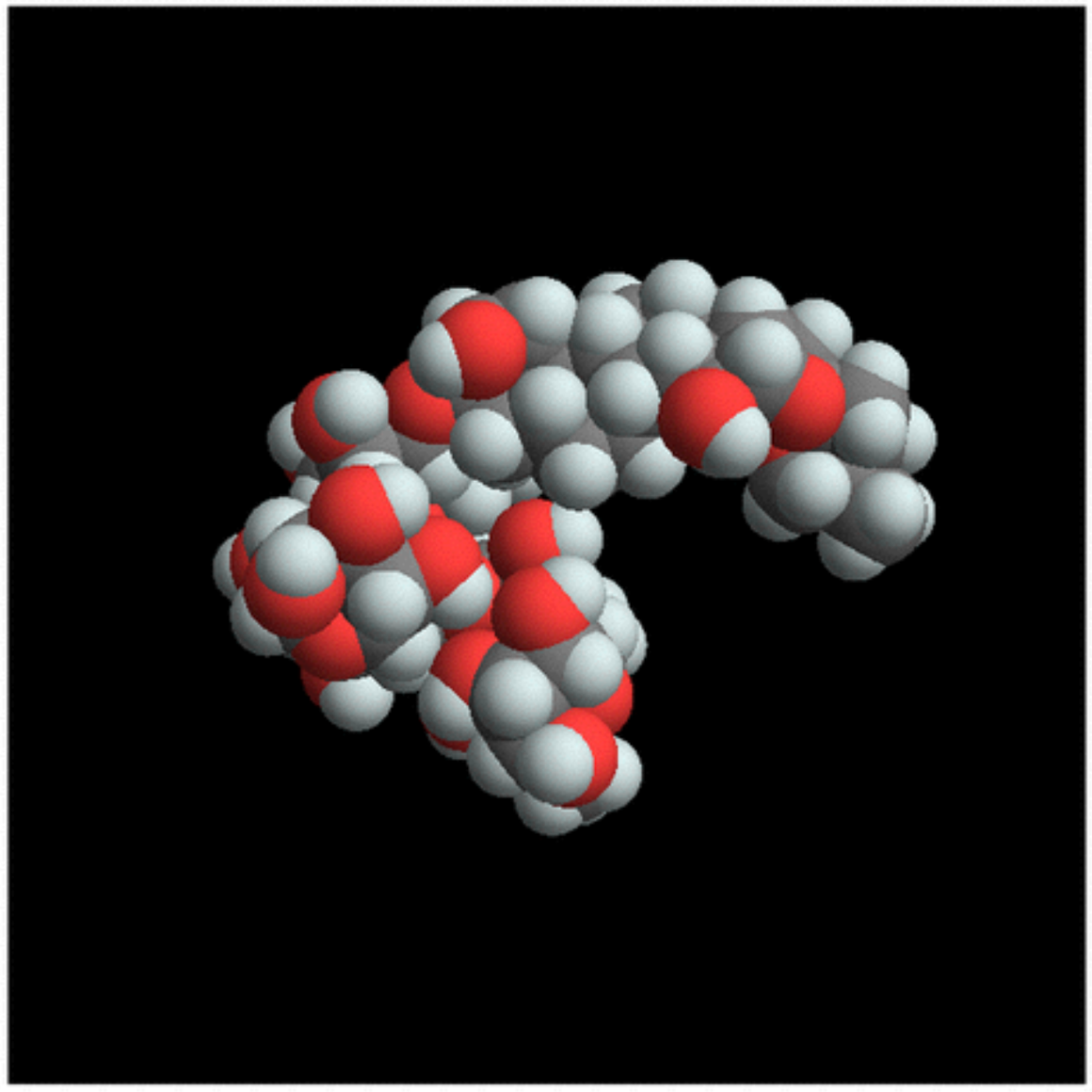
Household lexicon: approx 4000

Differ from community repository: all interfaces are consistent

Wolfram:

Tweet-a-program

```
Show[ChemicalData["Digitonin", "SpaceFillingMoleculePlot"],
Background -> Black]
```



Wolfram Language | Tweet–a–Program @wolframtap

Wolfram:

Tweet-a-program

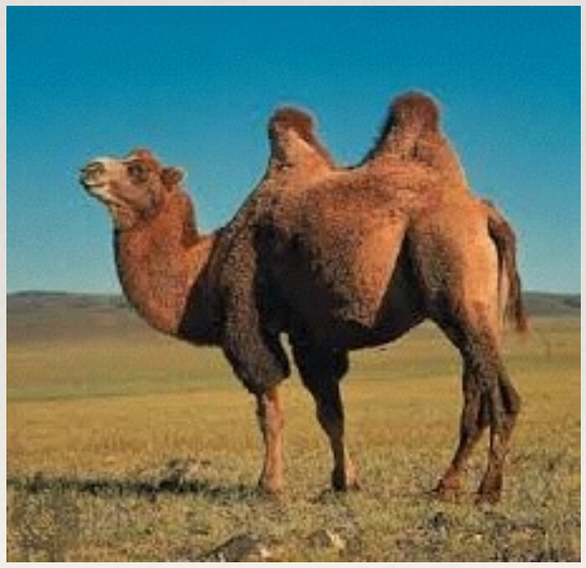
```
ImageCollage[Colorize[#, ColorFunction -> ColorData["Pastel"]] & /@
 #[RandomSample[#["Pokemon", "Entities"], 25], "Image"] & @
  EntityValue]
```

Wolfram Language | Tweet–a–Program @wolframtap

Make all good

- Syntax, defined in problem-domain specific library
- Vocabulary simular to one in natural language
- Advanced type theory with refinement specs.

Implementation



Implementation

Compilation/Interpretation [both - hard]

VM/System both - impossible without divergence

JVM no continuations (?)

LLVM non-trivial gc (?)

Complexity.

Let's rewrite all with callbacks, because it's more easy than fix JVM/OS

CORBA story:

Let's do remote and local call undistinguished (1996)

Local calls: fast

Remote calls: slow

Common antipattern: build fine-grained object model in RPC

Akka:

Let's do remote and local call undistinguished (2014)

// hope, now things will move forward more accurate

So, we must

choose one level of abstraction

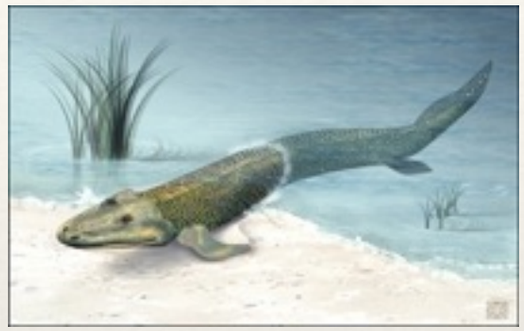
or

build family of systems

otherwise — abstraction leaks

Ok, assume we build ideal language. (in academia or as freaks)

When it will be in industry ?



- Success in sense of adoption => evolution slowdown

Haskell:

- avoid success of all costs => be open for evolution,
  live in a water

Scala: from research to engineering

(LMS, attempt to remove skolem types, … etc)

Sociology of programming languages

Leo Meyerovich

Average ‘incubation’ period before wider adoption: 12 years

before - specialized in some niche

Adoption => social, marketing ….., not technical.

What will be in industry tomorrow: landscape now.

General purpose: need some time for current adoption

JVM land: scala, clojure

Trends: integrating other paradigms

System: D / Rust instead C++ (?)

Application/ Scripting: [place is vacant]

// JavaScript successor [?]

What will be **not in industry** tomorrow:

academic/OSS

Better VM-s

New paradigms

Depends from you

// niche as social issue

Thanks.

ruslan@shevchenko.kiev.ua

https://github.com/rssh

@rssh1

If you build something interesting - ping me.