OOP & FP

Lev Walkin
@levwalkin

Myself

- Founder, Chief Editor of a peer-reviewed «Practice of Functional Programming» journal http://fprog.ru/
- CTO Echo, a venture backed company http://aboutecho.com/
- Proficient in N languages, M of them FP

Echo

- 50,000+ HTTP requests per second
- 500,000+ simultaneous users on site
- 1,527,721,774 search queries last month
- JavaScript, Clojure, Haskell, C, LAMP, RoR, OCaml, Python, Perl, ~~Visual Basic~~, C++

In this talk, we...

- Test a few myths about OOP & FP
- Distill OOP & FP to their core concepts
- Show that concepts & idioms matter
- Show that bells and wistles matter

For this talk, we...

- Avoid IDE concerns
- Disregard library availability issues
- Assume LISP = Clojure, Scheme, Common Lisp, Scheme, et cetera
- Minimize trolling 

PHP vs Haskell

Business issues

- Business succeeds or fails because of people & markets, not programming paradigms
- On a business and systems architecture level the differences are slim if any
- Turing-complete formalisms are enough

Why argue FP?

- Because people do not know what FP is
- Because people don’t even know what OOP is
- Because arguing FP incites to learn a new mental tool
- Mental tools are good: they are *cheap* to maintain and they may save lots of time

http://lionet.livejournal.com/38971.html

But first... OOP

- Encapsulation
- Inheritance
- Polymorphism

But first... OOP

- Encapsulation
- Inheritance
- Polymorphism

Quote

- “[Goal] was to find a better module scheme for complex systems involving hiding of details, and [...] a more flexible version of assignment, and then to try to eliminate it altogether.” (c) Alan Kay

Huh?..

http://worrydream.com/EarlyHistoryOfSmalltalk/

Quote

Encapsulation?

Function call?

- “OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things.” (c) Alan Kay

WTF?

http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en

OOP

- Where's inheritance?
- What languages conform to that? C++? Java?
- Erlang — a language typically associated with FP

http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en

Smalltalk'71

```
to 'factorial' 0 is 1
to 'factorial' :n do 'n*factorial n-1'
```

```
to :e 'is-member-of' [] do False
to :e 'is-member-of' :group
   do'if e = firstof group
      then True
      else e is-member-of rest of group'
```

Haskell'98

```
factorial 0 = 1
factorial n = n * factorial (n-1)
```

```
e `isMemberOf` [] = False
e `isMemberOf` group =
   if e = hd group
      then True
      else e `isMemberOf` (tl group)
```

Confused already?

http://en.wikipedia.org/wiki/Object-oriented_programming

History quirk

- The first standardized object-oriented language was… LISP
(ANSI CL's CLOS + Meta Object Protocol)

http://en.wikipedia.org/wiki/Object-oriented_programming

Quote

- “Scheme has wonderful lambdas, I use all the time. I am very fond of Scheme, by the way. The first variant of STL was written in Scheme, if you didn’t know.” (c) Alexander Stepanov

http://lionet.livejournal.com/59475.html

FP Myths

- LISP is FP
- Language X has first class functions, therefore it supports FP
- I frequently use Lambdas, therefore I FP

More FP Myths

- If you want to be on the forefront, use LISP
- Fact: language research has moved on to Haskell and Scala (to a lesser degree). LISP is playing catch-up (Clojure).

Declarative

Contracts

Model Driven

Domain Driven

Types

eDSL

OOP

[e]DSL

OOP

Structural

Applicative

Imperative

Functional

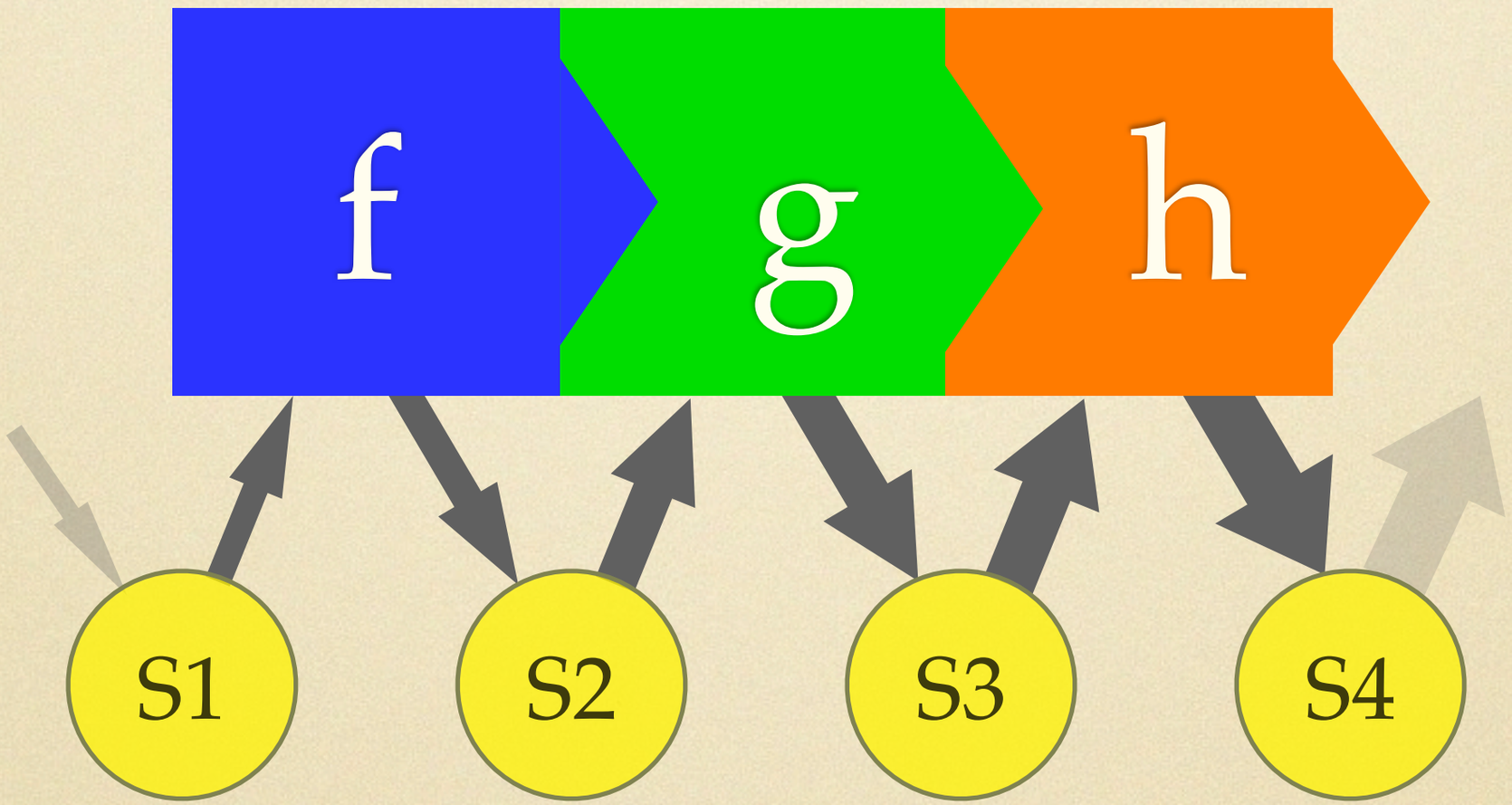
Programming Idioms

- Programming language is an interplay between facilities and available practices
- Practices nicely matching facilities form idioms
- Idioms differ between languages
- Case in point: Monad in C++
  http://ideone.com/cmkvyn

FP Idioms

- Immutability (+ persistent data structures)
- Composability (+ Monads, Arrows, etc)
- Transient state (no object identity)

Composition



Language Research

- Data abstraction for ease of *program modification* (60–70's)

Structured programming

- Separating moving parts for *managing complexity* (70–80's)

Smalltalk, Erlang

- Types and constraints to support *systems' evolution* (2000's–)

Haskell, Scala

False Targets

- Novel purely functional algorithms
- Syntax minification
- Quick to write (!)

True Targets

- Rapid prototyping (LISP, ML)
- Software evolution and maintenance (Erlang, ML)
- Thinking using Models (ML)

Thank you!

Questions?

jobs@aboutecho.com