Real World Cascalog

Federico Brubacher
@fbru02

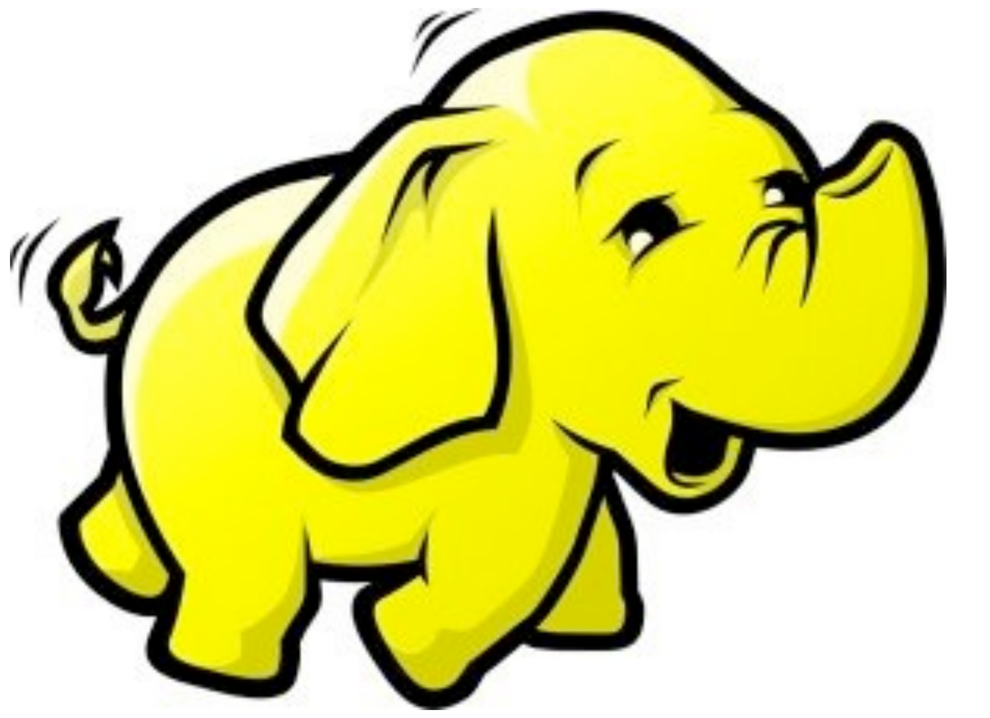
What is the object of this talk?

We (in the Clojure community) have seen what Cascalog is, now I want to share real world experiences.

Personal introduction

Hadoop is complected.

BUT....



Hadoop is proven
Hadoop is robust

Why Cascalog?

Raw Data

(unstructured)

Raw Data

(unstructured)

(just tuples)

Structured Data

- Using Thirft or any other serialization framework

Structured Data

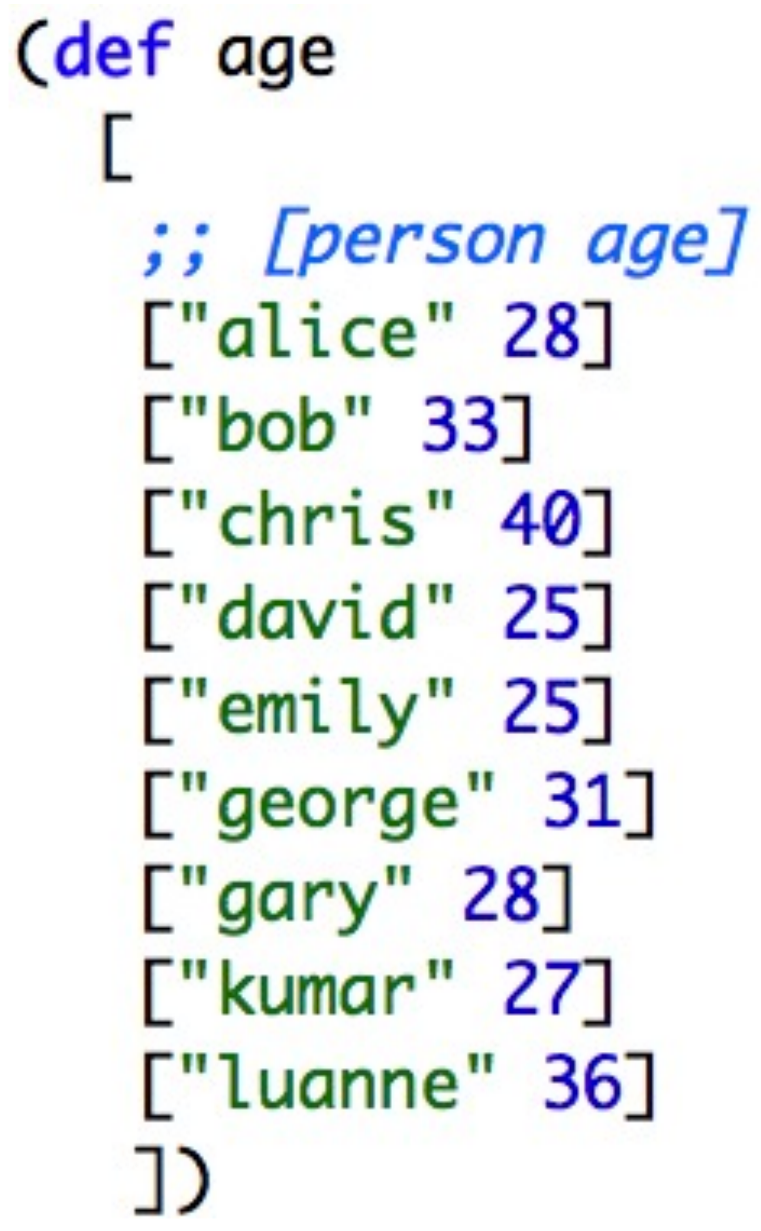
- Using Thirft or any other serialization framework
- Unstructured -> Structured (Transform it using a Cascalog job)

What can you do with Cascalog?

- Big Data systems
- Taps to a bunch of technologies

Basic Cascalog

The age dataset



```
(def age
  [
   ;; [person age]
   ["alice" 28]
   ["bob" 33]
   ["chris" 40]
   ["david" 25]
   ["emily" 25]
   ["george" 31]
   ["gary" 28]
   ["kumar" 27]
   ["luanne" 36]
   ])
```

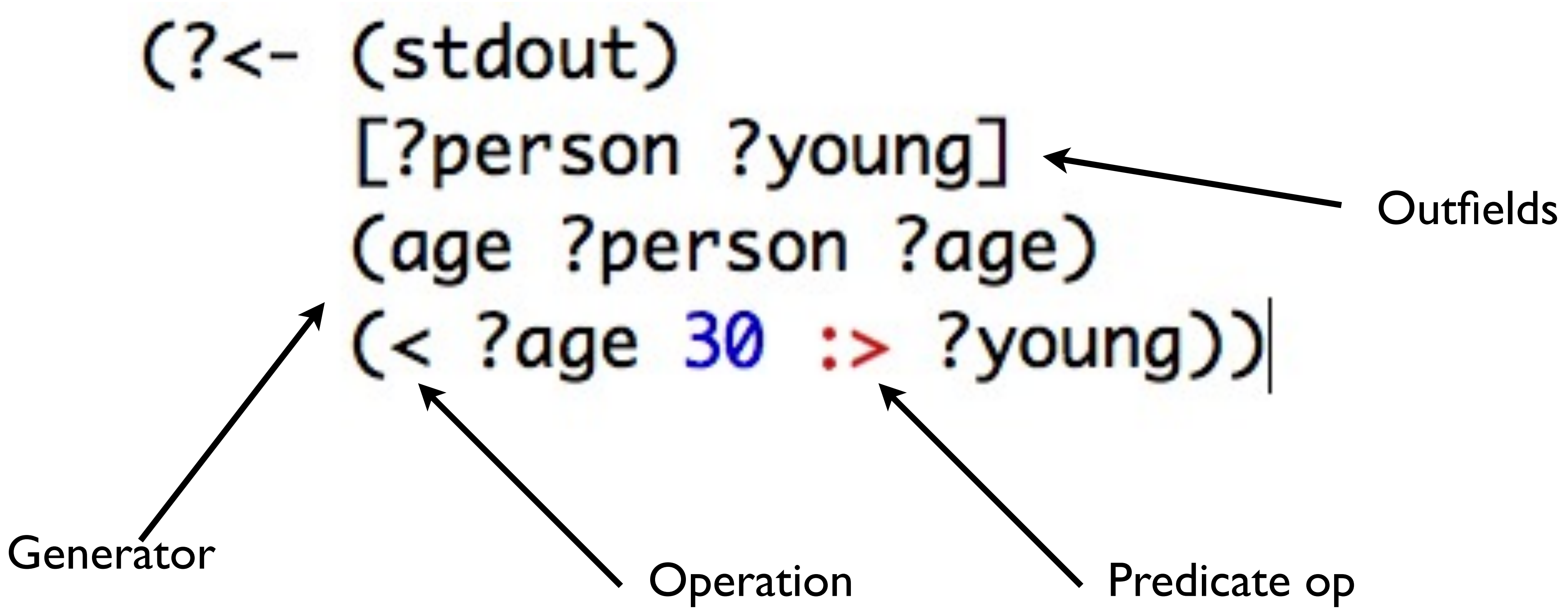
How do we do queries?

How do we do queries?

1. Pre-aggregation
2. Aggregation
3. Post-aggregation

How do we do queries?

Source(Tap) -> Transformation -> Sink(Tap)



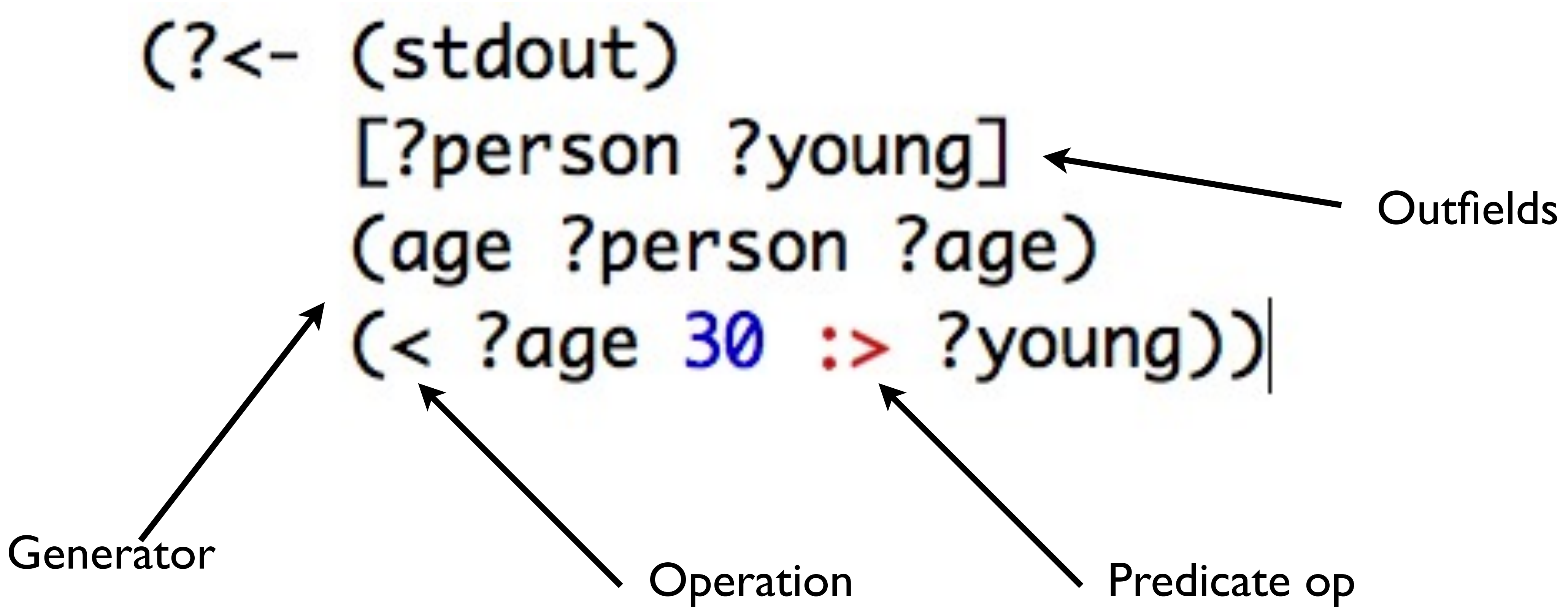
```
(?<- (stdout)
     [?person ?young]
     (age ?person ?age)
     (< ?age 30 :> ?young))
```

Rationale

- We build queries by chaining together operations

How do we do queries?

Source(Tap) -> Transformation -> Sink(Tap)



```
(?<- (stdout)
     [?person ?young]
     (age ?person ?age)
     (< ?age 30 :> ?young))
```

Competition

Hive vs Cascalog

Hive vs Cascalog 2

Editing, compiling, testing

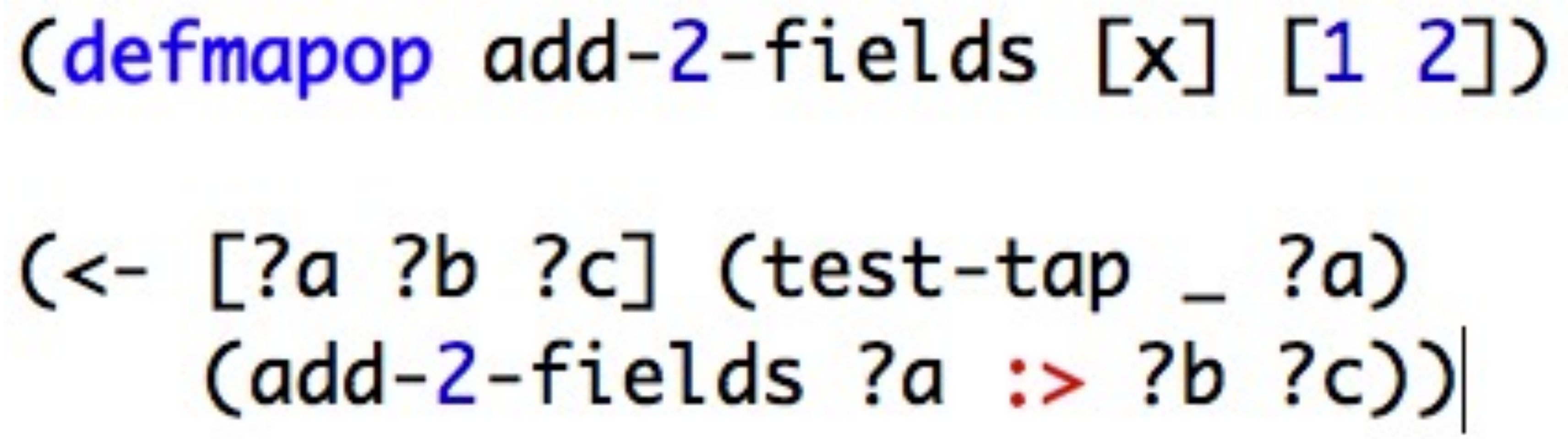
vs

REPL

Hive vs Cascalog 3

Hive enforces a static model of data
vs
Cascalog let's the developer decide how he wants to couple data.

Operations a la carte



```
(defmapop add-2-fields [x] [1 2])

(<- [?a ?b ?c] (test-tap _ ?a)
    (add-2-fields ?a :> ?b ?c))
```

Cascading

Cascading

- Productivity

Cascading

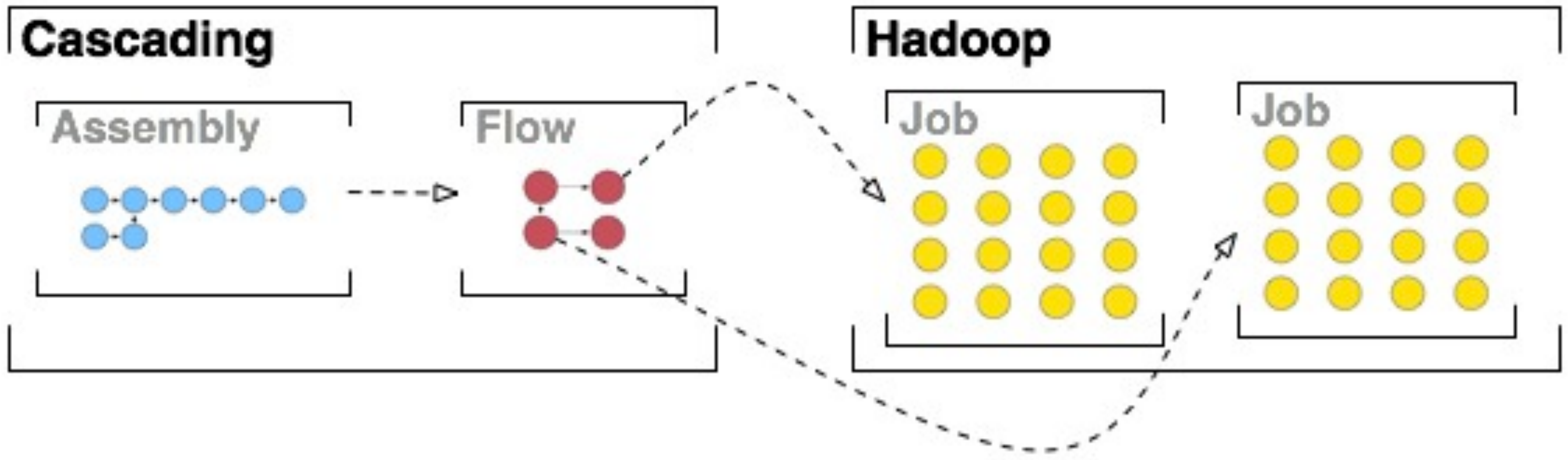
- Productivity
- Chaining (Composability)

Cascading

- Productivity
- Chaining (Composability)
- Failing fast

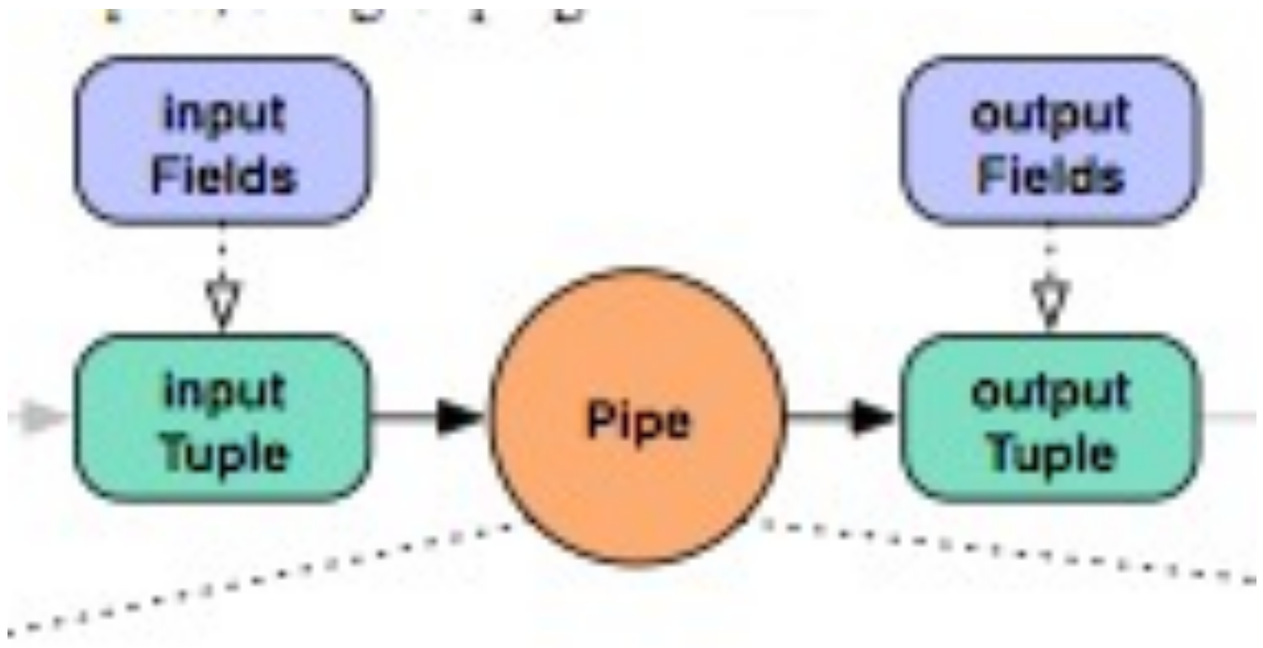
Optimizer

Query planner



Cascading taps

Cascading



Cascading 2

- We build a Cascade chaining up Flows

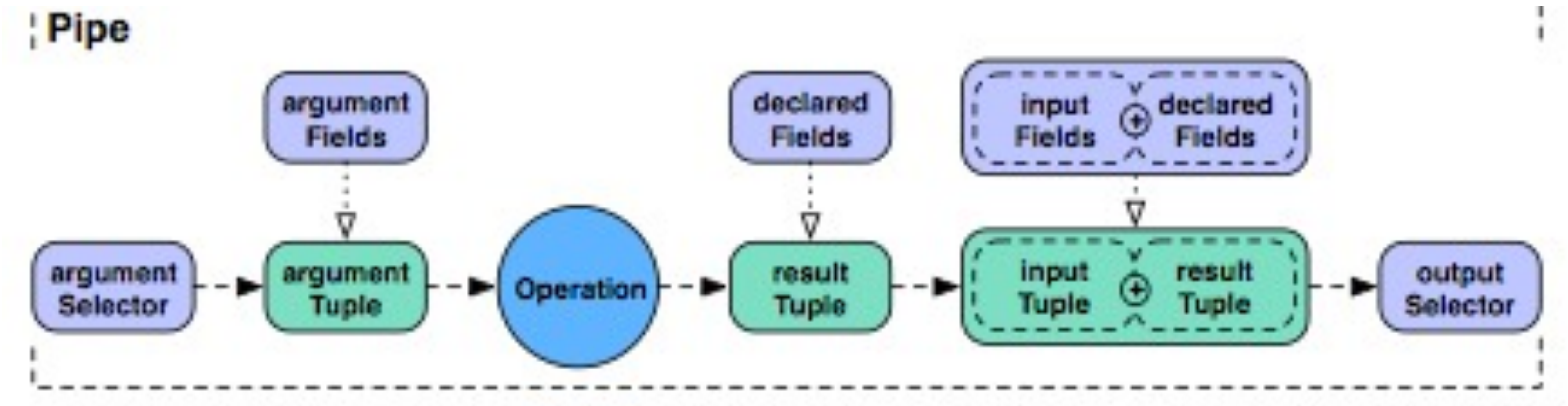
Cascading 2

- We build a Cascade chaining up Flows
- We build Flows chaining up Pipes

Cascading 2

- We build a Cascade chaining up Flows
- We build Flows chaining up Pipes
- And Pipes are made of operations

Cascading 3

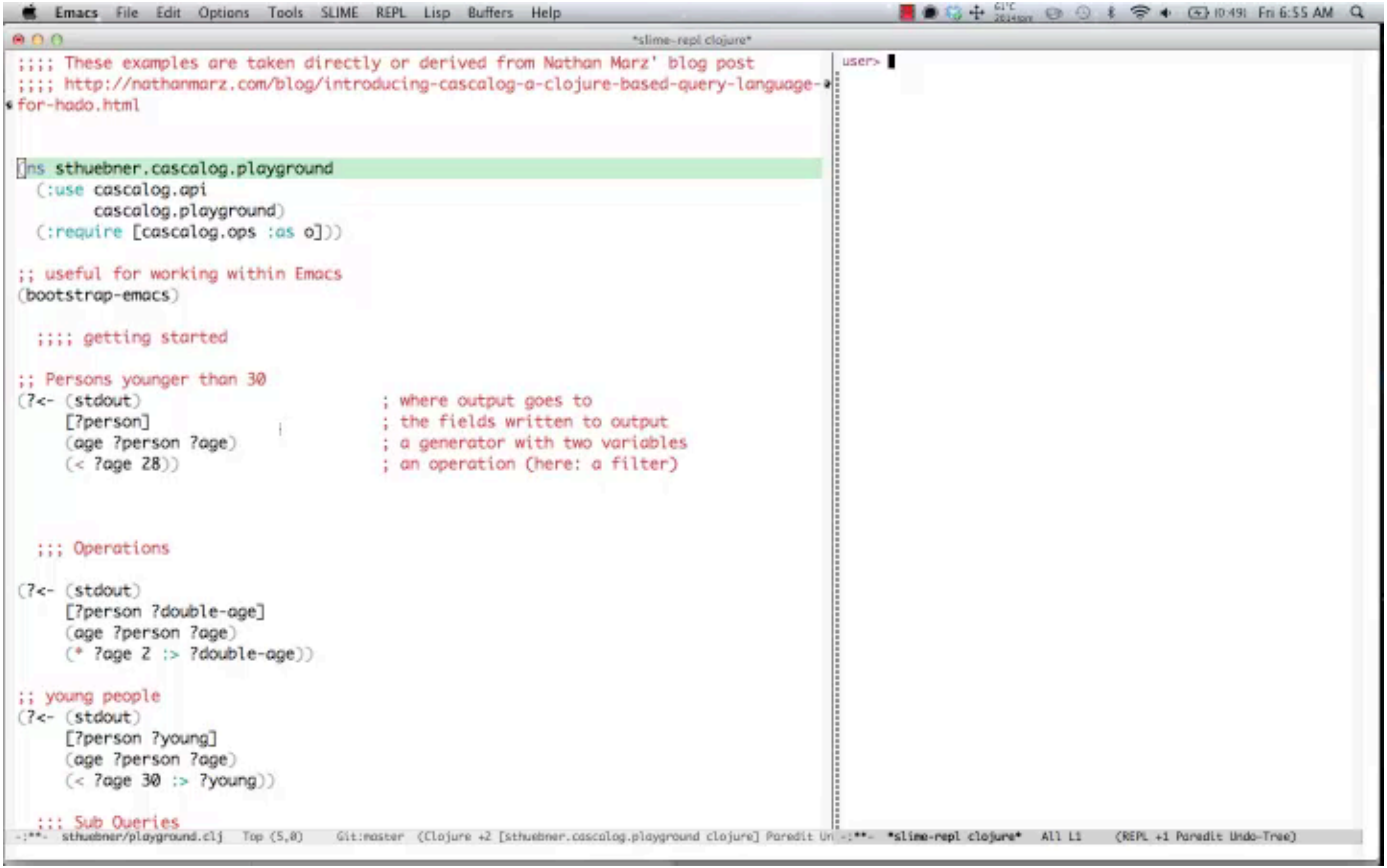

Pipe
argument Fields
declared Fields
input Fields ⊕ declared Fields
argument Selector
argument Tuple
Operation
result Tuple
input Tuple ⊕ result Tuple
output Selector

Cascading world

- 2 basic kind of pipes
- Each and Every
- Co-group and Group-by

Leaky Abstraction ?

Demo



```
;;;; These examples are taken directly or derived from Nathan Marz' blog post
;;;; http://nathanmarz.com/blog/introducing-cascalog-a-clojure-based-query-language-
for-hado.html


(ns sthuebner.cascalog.playground
  (:use cascalog.api
        cascalog.playground)
  (:require [cascalog.ops :as o]))

;; useful for working within Emacs
(bootstrap-emacs)

  ;;;; getting started

;; Persons younger than 30
(?<- (stdout)                         ; where output goes to
     [?person]                        ; the fields written to output
     (age ?person ?age)               ; a generator with two variables
     (< ?age 28))                     ; an operation (here: a filter)



  ;;; Operations

(?<- (stdout)
     [?person ?double-age]
     (age ?person ?age)
     (* ?age 2 :> ?double-age))

;; young people
(?<- (stdout)
     [?person ?young]
     (age ?person ?age)
     (< ?age 30 :> ?young))

  ;;; Sub Queries
```

```
user>
```

Emacs File Edit Options Tools SLIME REPL Lisp Buffers Help
Fri 6:55 AM
slime-repl clojure
-:**- sthuebner/playground.clj Top (5,0) Git:master (Clojure +2 [sthuebner.cascalog.playground clojure] Paredit Un
-:**- *slime-repl clojure* All L1 (REPL +1 Paredit Undo-Tree)

Real world applications

- Analytics

Real world applications

- Analytics
- Data transformation

Real world applications

- Analytics
- Data transformation
- Data Crunching (Aggregation)

Ad-network

Advanced Cascalog

- Predicate operators

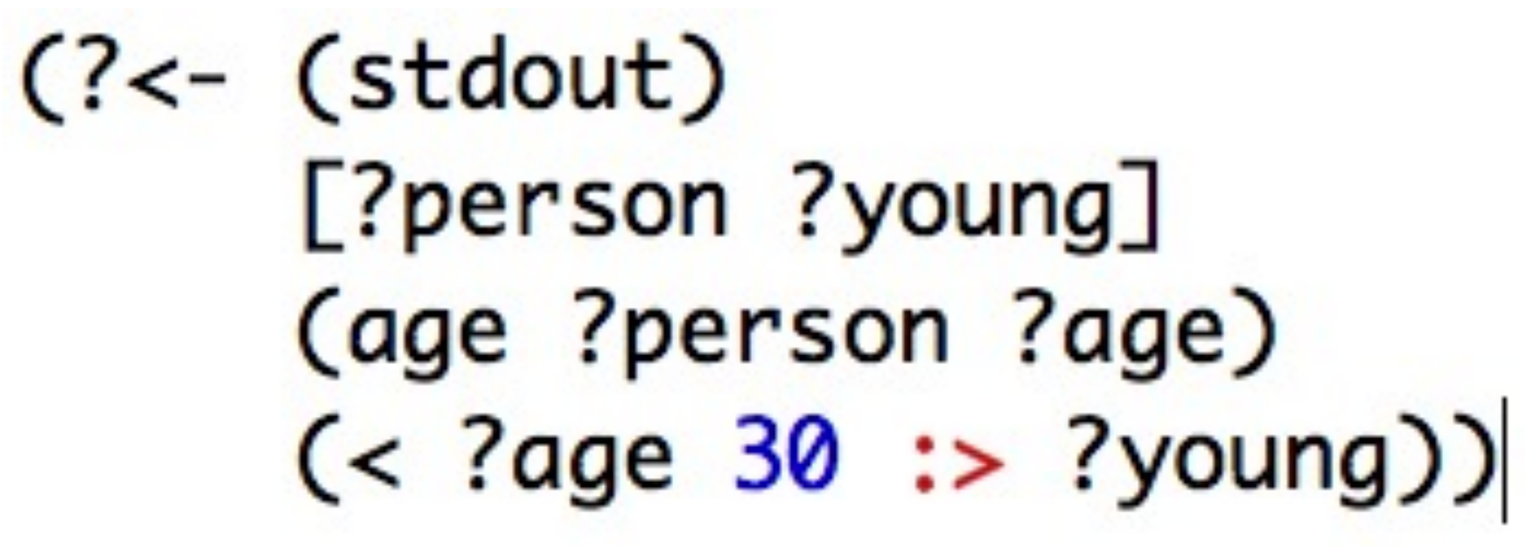
- Predicate operators
- Operations

- Predicate operators
- Operations
- Parametric Ops

- Predicate operators
- Operations
- Parametric Ops
- Stateful Ops

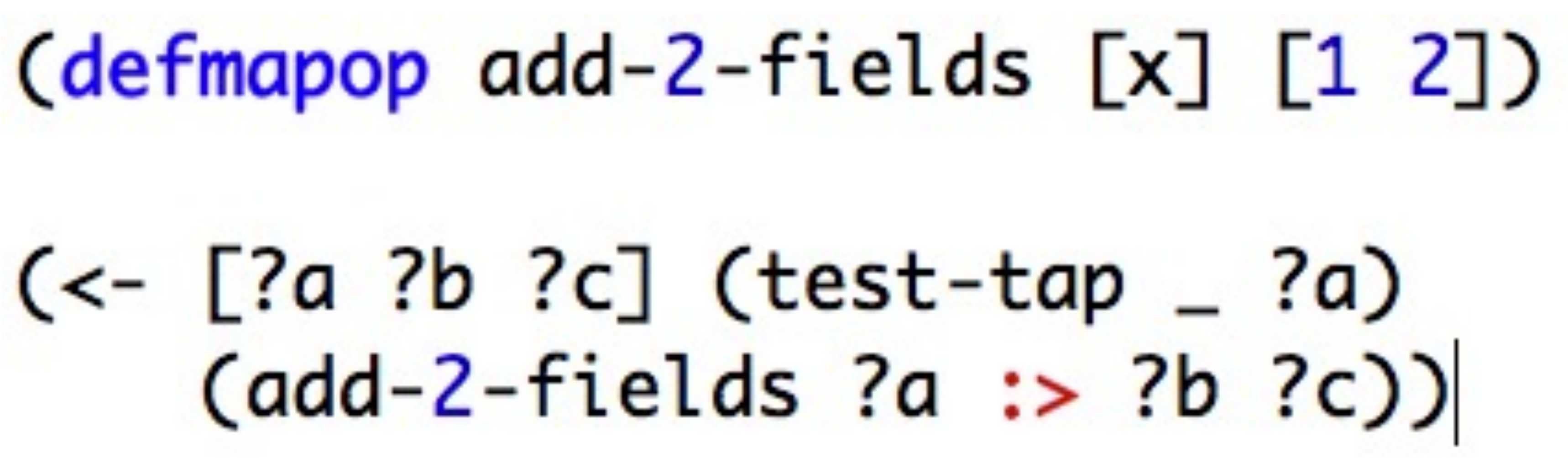
- Predicate operators
- Operations
- Parametric Ops
- Stateful Ops
- Predicate Macros

Predicate operators



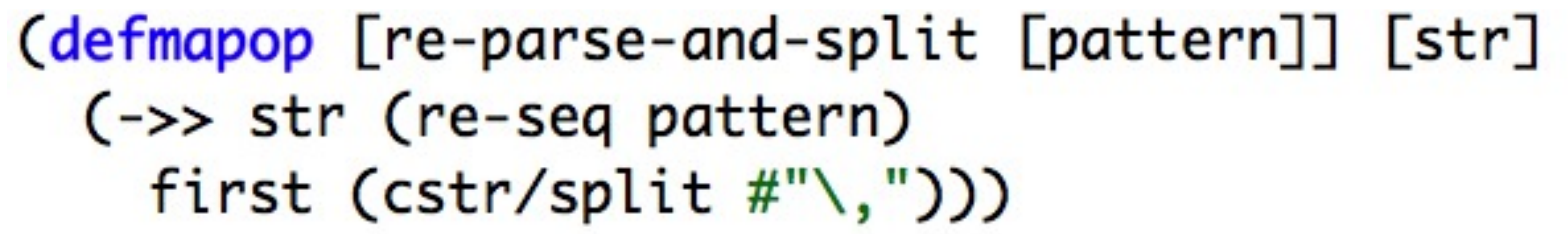
```
(?<- (stdout)
     [?person ?young]
     (age ?person ?age)
     (< ?age 30 :> ?young))
```

Ops



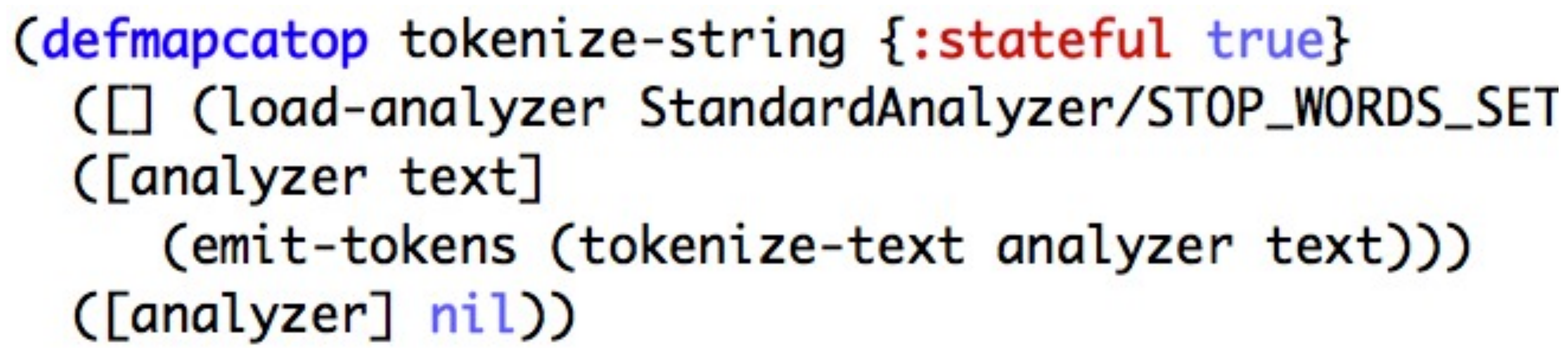
```
(defmapop add-2-fields [x] [1 2])

(<- [?a ?b ?c] (test-tap _ ?a)
    (add-2-fields ?a :> ?b ?c))
```

Parametric Ops



```
(defmapop [re-parse-and-split [pattern]] [str]
  (->> str (re-seq pattern)
    first (cstr/split #"\,")))
```

Stateful Ops



```
(defmapcatop tokenize-string {:stateful true}
  ([] (load-analyzer StandardAnalyzer/STOP_WORDS_SET
  ([analyzer text]
     (emit-tokens (tokenize-text analyzer text)))
  ([analyzer] nil))
```

Kind of Operations

Map side	Reduce side
defmapop	defbufferop
deffilterop	defaggregateop
defmapcatop	
Vanilla Clojure functions	

defparallelagg

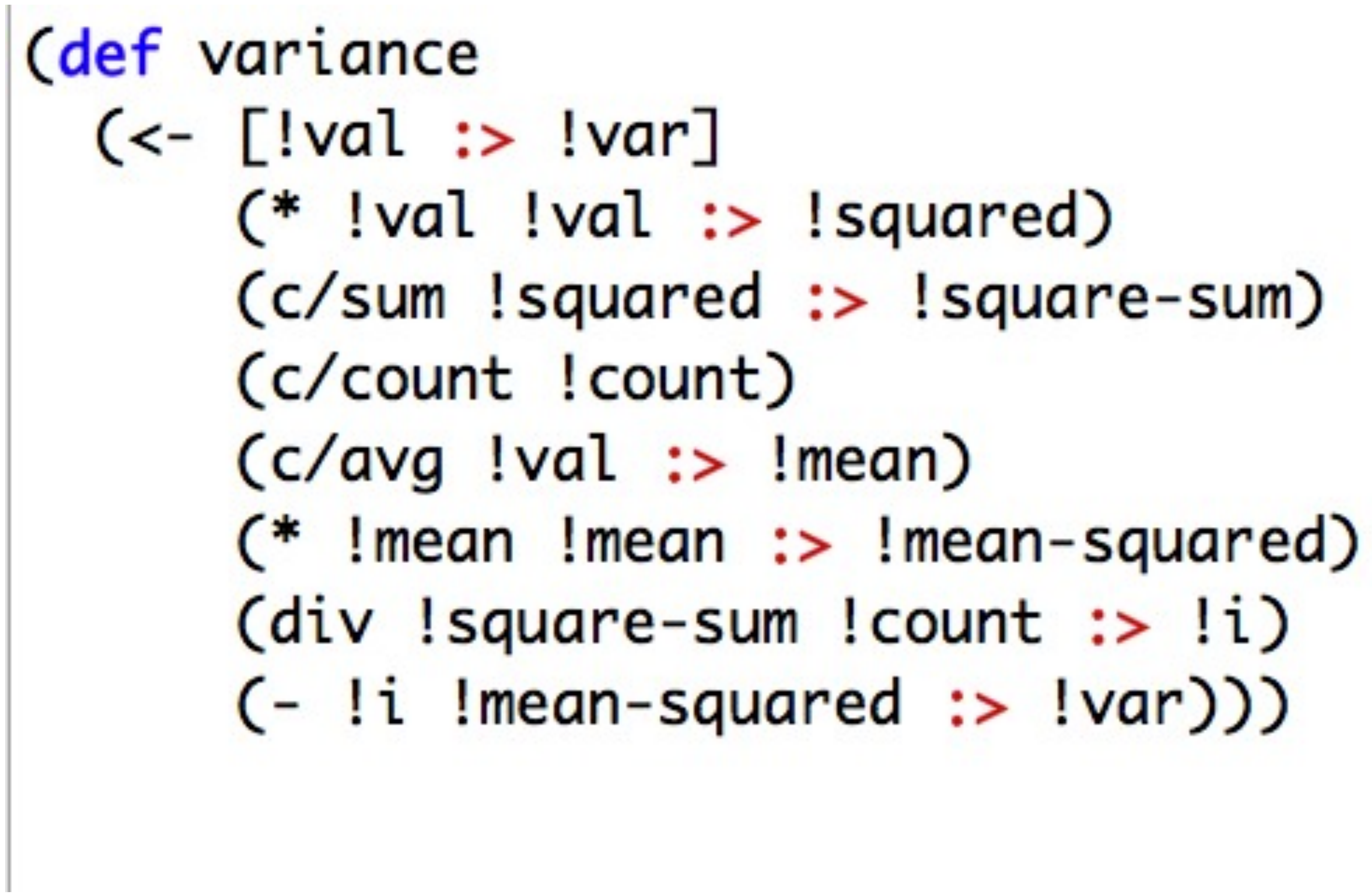
Predicate Macros

- Arbitrarily compose predicates together

Predicate Macros

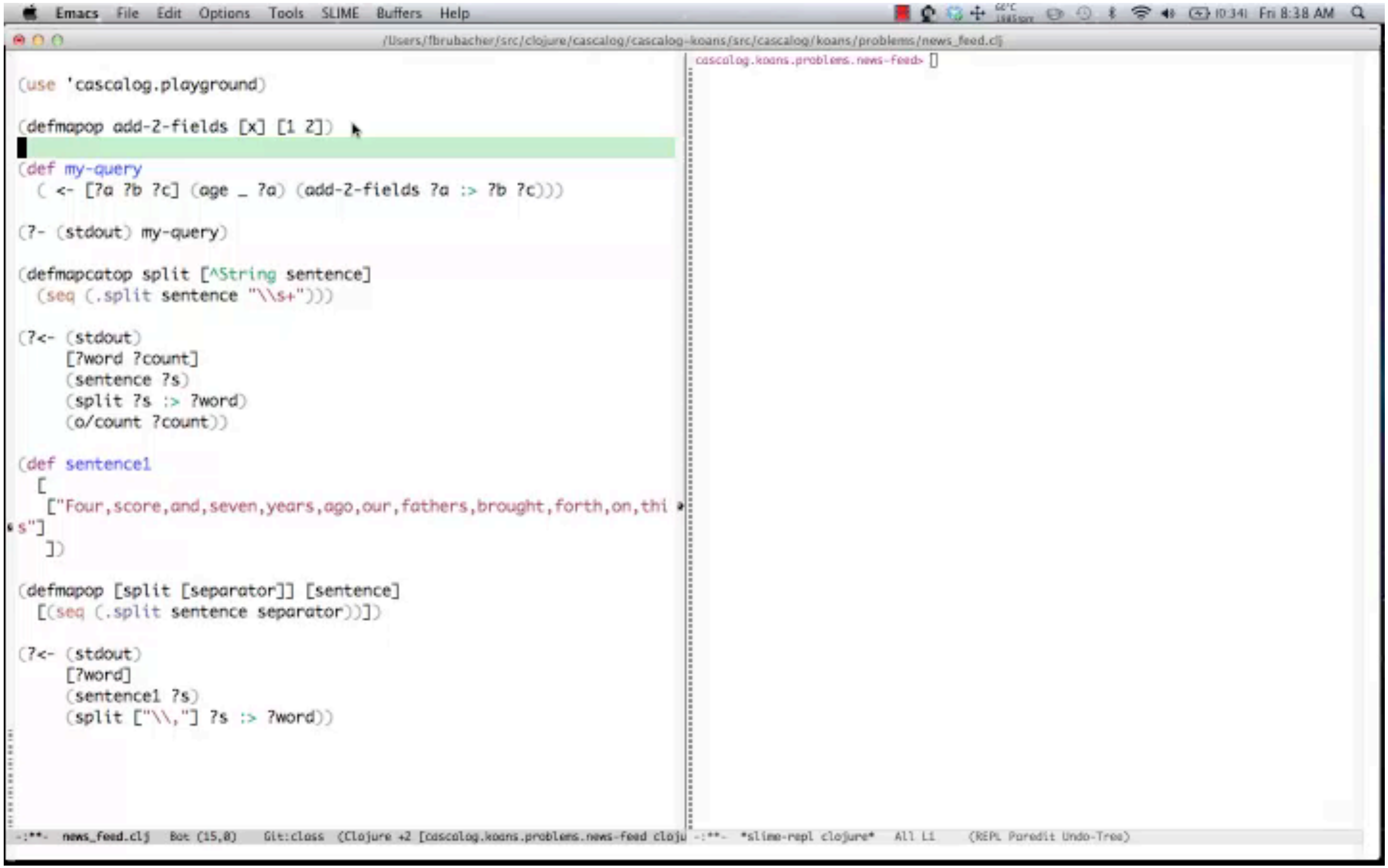
- Arbitrarily compose predicates together
- Useful for abstracting things away from our queries

Predicate macros 2



```
(def variance
  (<- [!val :> !var]
      (* !val !val :> !squared)
      (c/sum !squared :> !square-sum)
      (c/count !count)
      (c/avg !val :> !mean)
      (* !mean !mean :> !mean-squared)
      (div !square-sum !count :> !i)
      (- !i !mean-squared :> !var)))
```

Demo



/Users/fbrubacher/src/clojure/cascalog/cascalog-koans/src/cascalog/koans/problems/news_feed.clj

```
(use 'cascalog.playground)

(defmapop add-2-fields [x] [1 2])

(def my-query
  ( <- [?a ?b ?c] (age _ ?a) (add-2-fields ?a :> ?b ?c)))

(?- (stdout) my-query)

(defmapcatop split [^String sentence]
  (seq (.split sentence "\\s+")))

(?<- (stdout)
     [?word ?count]
     (sentence ?s)
     (split ?s :> ?word)
     (c/count ?count))

(def sentence1
  [
   ["Four,score,and,seven,years,ago,our,fathers,brought,forth,on,thi
s"]
    ])

(defmapop [split [separator]] [sentence]
  [(seq (.split sentence separator))])

(?<- (stdout)
     [?word]
     (sentence1 ?s)
     (split ["\\,"] ?s :> ?word))
```

```
cascalog.koans.problems.news-feed> 
```

```
-:**-  news_feed.clj   Bot (15,0)    Git:class  (Clojure +2 [cascalog.koans.problems.news-feed cloju
-:**-  *slime-repl clojure*   All L1     (REPL Paredit Undo-Tree)
```

New Cascading (2.0)

- Decouple from Hadoop

New Cascading (2.0)

- Decouple from Hadoop
- Better statistics

New Cascading (2.0)

- Decouple from Hadoop
- Better statistics
- In memory data - queries

New Cascalog

- Support for Cascading 2.0

New Cascalog

- Support for Cascading 2.0
- Support for serializing data using Kryo

New Cascalog

- Support for Cascading 2.0
- Support for serializing data using Kryo
- Support for serializing vars (First order map-reduce)

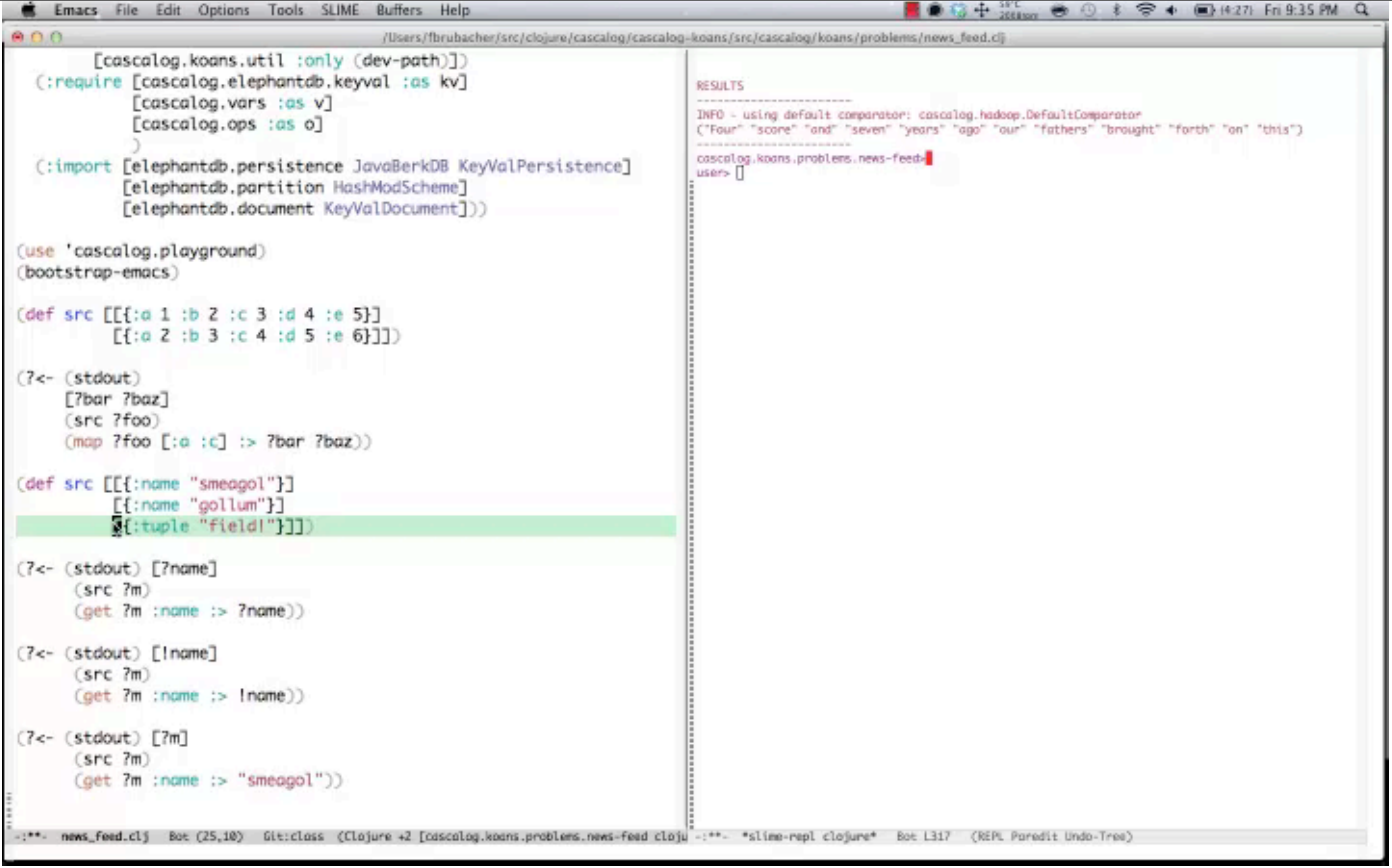
Coming up...

- New operations API

Coming up...

- New operations API
- Destructuring support for generators

Demo



/Users/fbrubacher/src/clojure/cascalog/cascalog-koans/src/cascalog/koans/problems/news_feed.clj

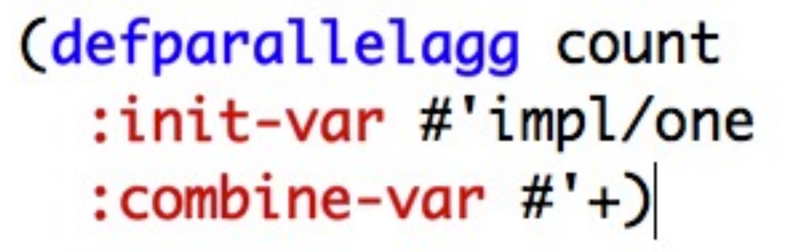
```clojure
          [cascalog.koans.util :only (dev-path)])
  (:require [cascalog.elephantdb.keyval :as kv]
            [cascalog.vars :as v]
            [cascalog.ops :as o]
            )
  (:import [elephantdb.persistence JavaBerkDB KeyValPersistence]
           [elephantdb.partition HashModScheme]
           [elephantdb.document KeyValDocument]))

(use 'cascalog.playground)
(bootstrap-emacs)

(def src [[{:a 1 :b 2 :c 3 :d 4 :e 5}]
          [{:a 2 :b 3 :c 4 :d 5 :e 6}]])

(?<- (stdout)
     [?bar ?baz]
     (src ?foo)
     (map ?foo [:a :c] :> ?bar ?baz))

(def src [[{:name "smeagol"}]
          [{:name "gollum"}]
          [{:tuple "field!"}]])

(?<- (stdout) [?name]
      (src ?m)
      (get ?m :name :> ?name))

(?<- (stdout) [!name]
      (src ?m)
      (get ?m :name :> !name))

(?<- (stdout) [?m]
      (src ?m)
      (get ?m :name :> "smeagol"))
```

```
RESULTS
-----------------------
INFO - using default comparator: cascalog.hadoop.DefaultComparator
("Four" "score" "and" "seven" "years" "ago" "our" "fathers" "brought" "forth" "on" "this")
-----------------------
cascalog.koans.problems.news-feed>
user> 
```

```
-:**-  news_feed.clj   Bot (25,10)   Git:class  (Clojure +2 [cascalog.koans.problems.news-feed cloju
-:**-  *slime-repl clojure*   Bot L317   (REPL Paredit Undo-Tree)
```

Advice for becoming good

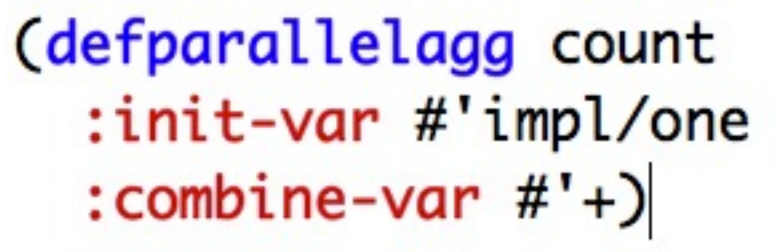
- Skim the code



```
(defparallelagg count
  :init-var #'impl/one
  :combine-var #'+)
```

Advice for becoming good

- Skim the code



```
(defparallelagg count
  :init-var #'impl/one
  :combine-var #'+)
```

- Each and juxt are nice examples

Advice for getting good 2

- Read Cascalog Contrib (and contribute)

Advice for getting good 2

- Read Cascalog Contrib (and contribute)
- Read API and Ops

Advice for getting good 2

- Read Cascalog Contrib (and contribute)
- Read API and Ops
- Ask in the mailing list

Cool uses

- Building the new Mahout (based on Incanter?)

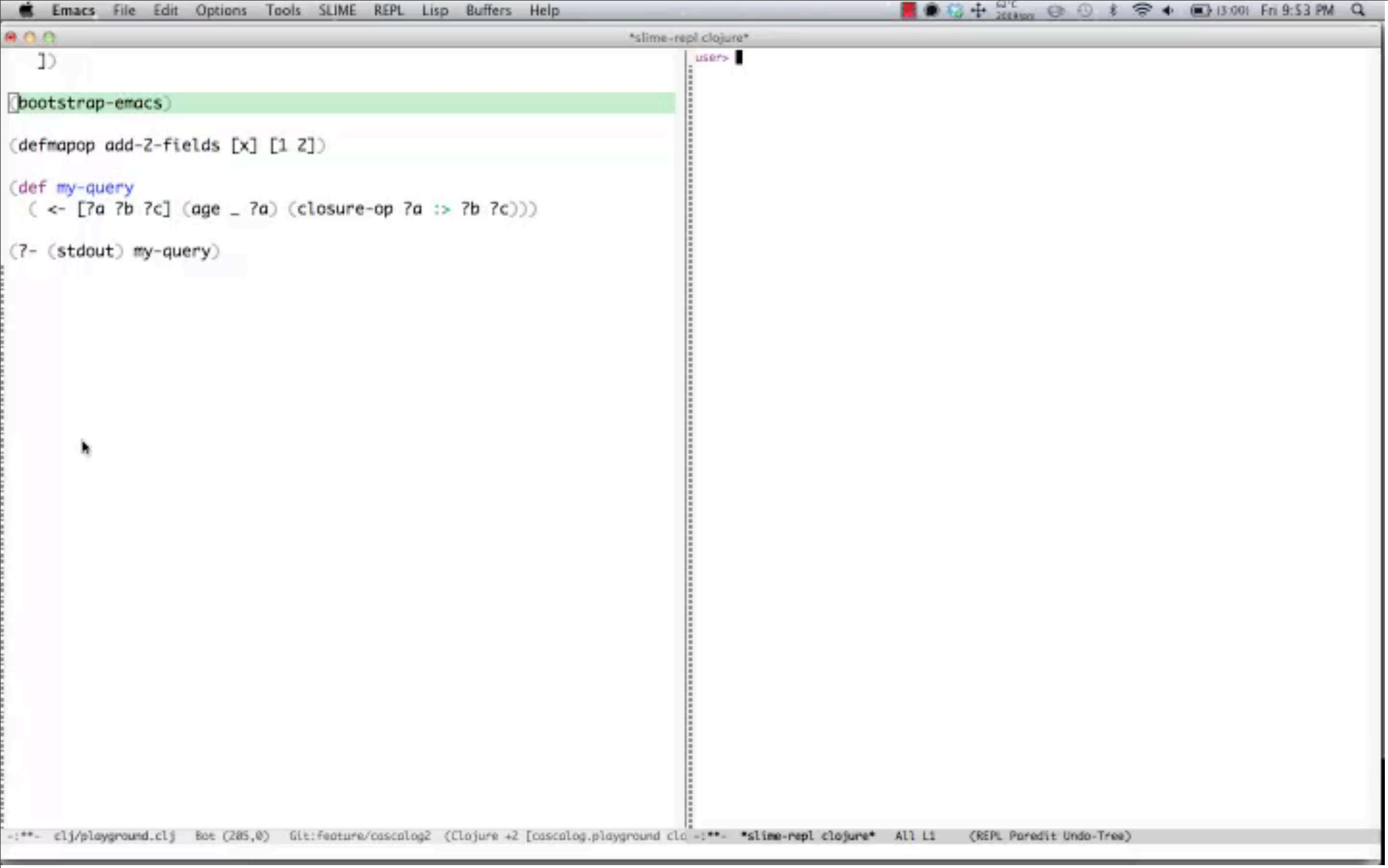
Cool uses

- Building the new Mahout (based on Incanter?)
- Building an open source Data Analytics platform

Cool uses

- Building the new Mahout (based on Incanter?)
- Building an open source Data Analytics platform
- Running queries on live data

Conclusions and questions



```
  ])

(bootstrap-emacs)

(defmapop add-2-fields [x] [1 2])

(def my-query
  ( <- [?a ?b ?c] (age _ ?a) (closure-op ?a :> ?b ?c)))

(?- (stdout) my-query)
```

```
user> 
```

```
-:**-  clj/playground.clj   Bot (205,0)   Git:feature/cascalog2  (Clojure +2 [cascalog.playground clo -:**-  *slime-repl clojure*   All L1    (REPL Paredit Undo-Tree)
```