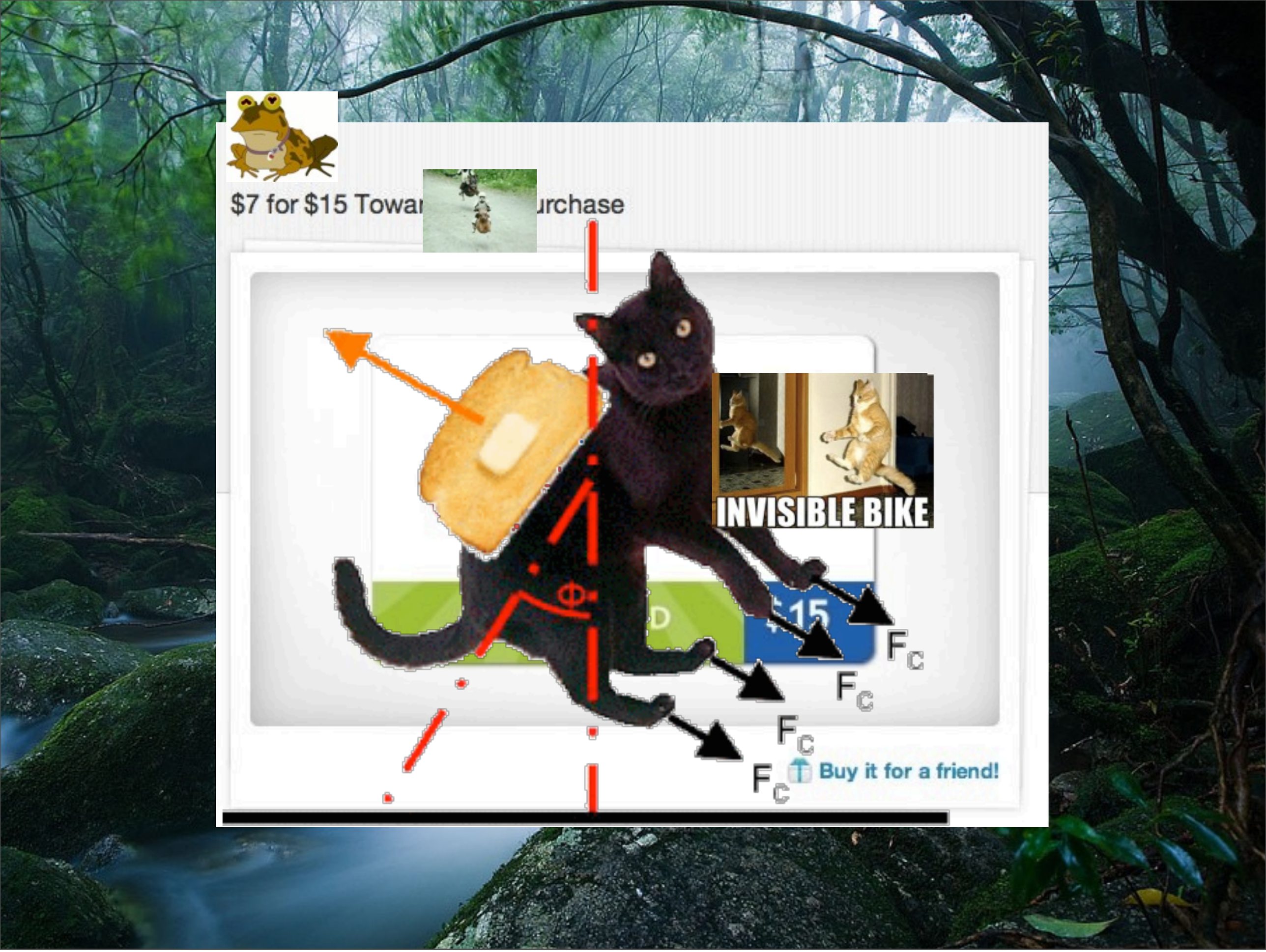
# Bootstrapping Clojure

Tyler Jennings
@tyler_jennings

A Short Story

# The Deal


$7 for $15 Towar
urchase
INVISIBLE BIKE
15
D
Fc
Fc
Fc
Fc
Buy it for a friend!

Free Money!

2 Days

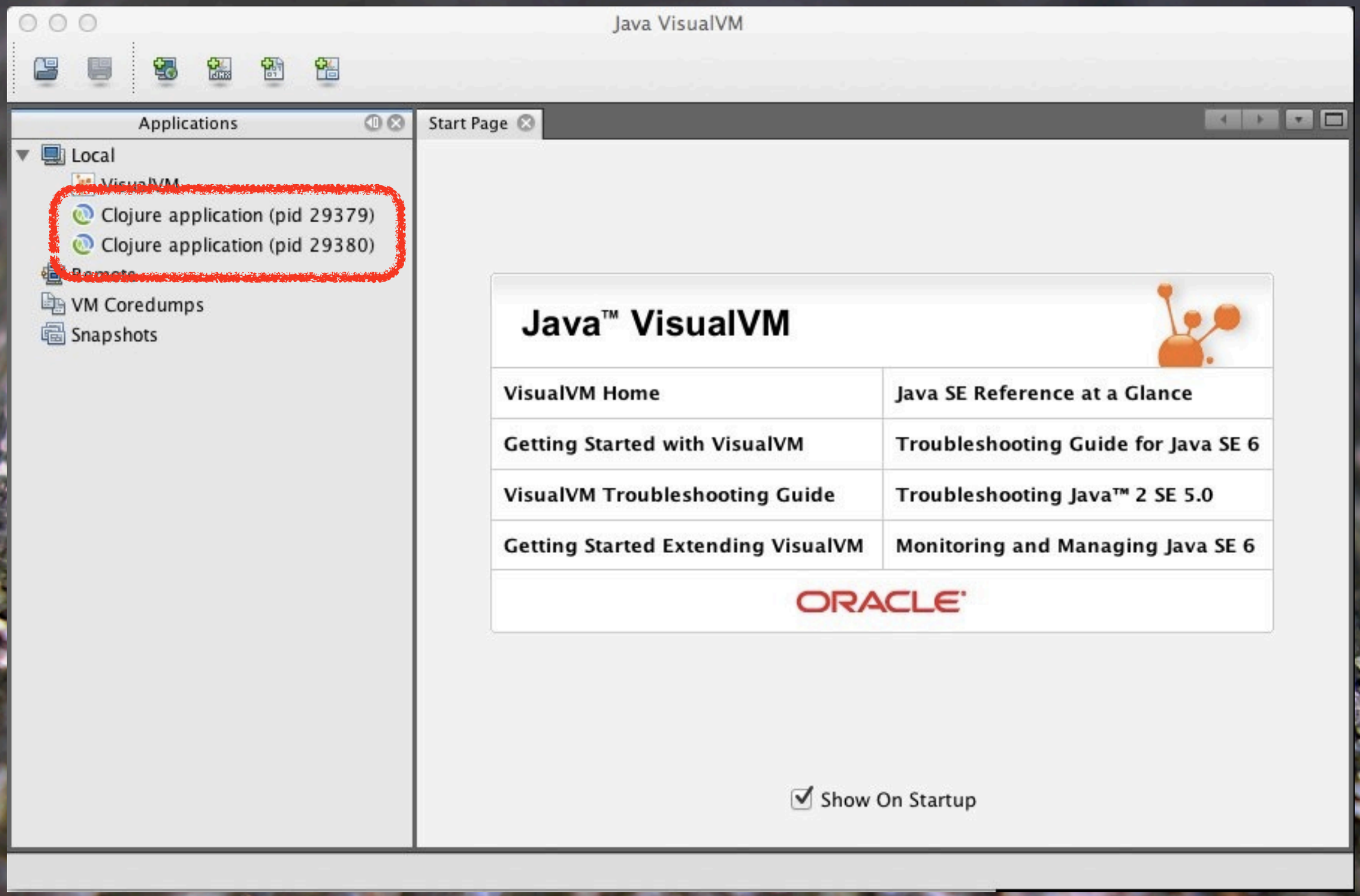
```
"tyler+cheats@gmail.com"  #easy, normalize
"tyler+chea+ts@gmail.com"

"tylercheats1@gmail.com"  #Getting hard
"tylercheats2@gmail.com"  #how do we find these guys?
"tylercheats3@gmail.com"
"tylercheats4@gmail.com"
"tylercheats5@gmail.com"
```

A Solution

# Optimization

jvisualvm
Run This command!


Java VisualVM
Applications
Start Page
Local
VisualVM
Clojure application (pid 29379)
Clojure application (pid 29380)
Remote
VM Coredumps
Snapshots
Java™ VisualVM
VisualVM Home
Java SE Reference at a Glance
Getting Started with VisualVM
Troubleshooting Guide for Java SE 6
VisualVM Troubleshooting Guide
Troubleshooting Java™ 2 SE 5.0
Getting Started Extending VisualVM
Monitoring and Managing Java SE 6
ORACLE
Show On Startup

# BK Tree

"Some Approaches to Best-Match File Searching" by W. A. Burkhard and R. M. Keller 1972

# Metric Space

A metric space is a set where a notion of distance (called a metric) between elements of the set is defined.

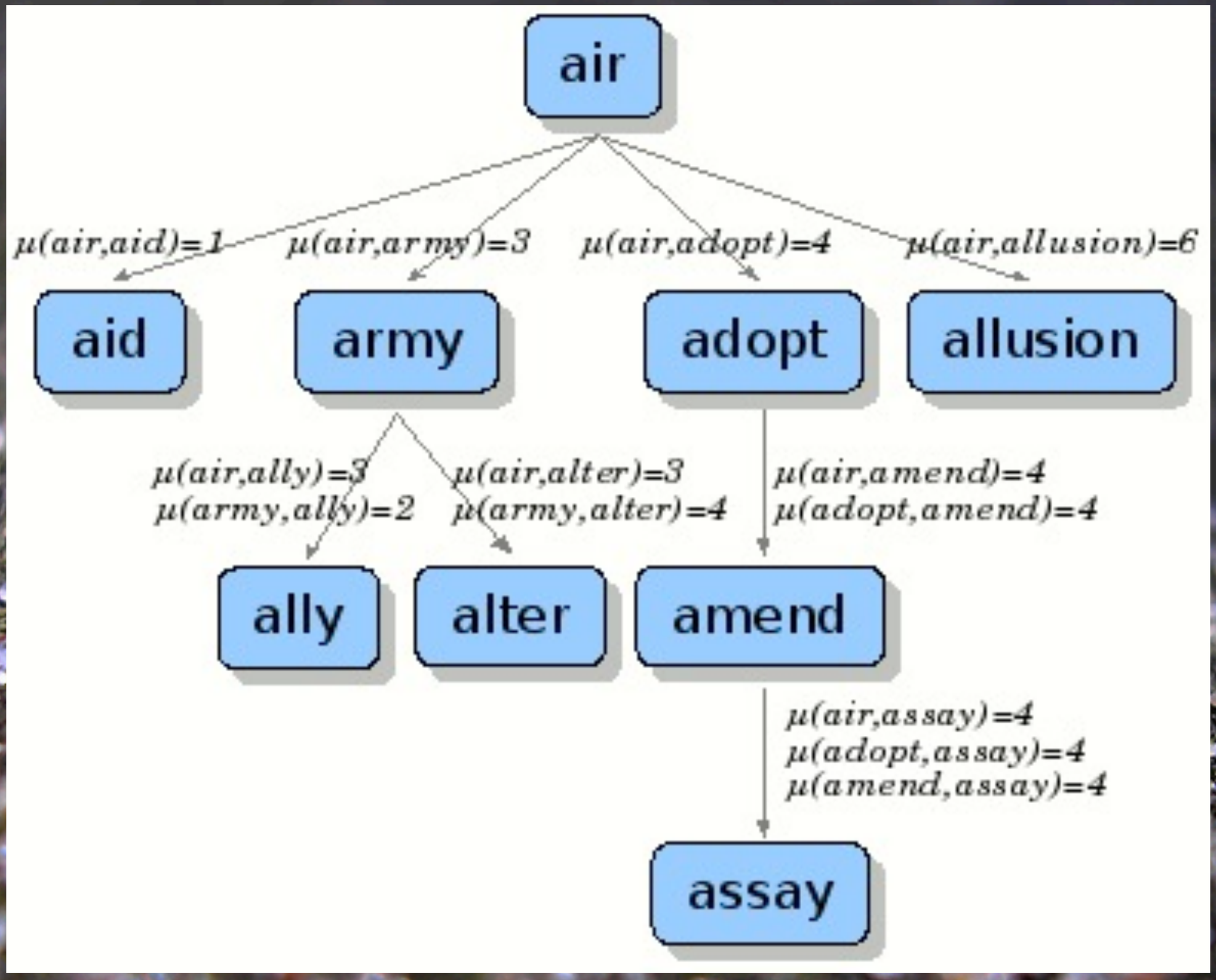
Such that for any x, y, z:

$d(x,y) >= 0$

$d(x,y) = 0$ IF $x = y$

$d(x,y) = d(y, x)$

$d(x,z) <= d(x,y) + d(y,z)$


air
μ(air,aid)=1
μ(air,army)=3
μ(air,adopt)=4
μ(air,allusion)=6
aid
army
adopt
allusion
μ(air,ally)=3
μ(army,ally)=2
μ(air,alter)=3
μ(army,alter)=4
μ(air,amend)=4
μ(adopt,amend)=4
ally
alter
amend
μ(air,assay)=4
μ(adopt,assay)=4
μ(amend,assay)=4
assay

| DB Size (words) | Threshold (distance) | Scanned Node Percentage |
|---|---|---|
| 500,000 | 1 | 0.0027 |
| | 2 | 0.0029 |
| | 3 | 0.0039 |
| | 4 | 0.0012 |
| | 5 | 0.3444 |
| | 6 | 0.4244 |
| | 7 | 13.4834 |
| | 8 | 30.3728 |

```clojure
(defn insert
  ([element node]
    (insert element node (:distance-fn node)))

  ([element node distance-fn]
    (if (empty-node? node)
      (assoc node :term element)
      (let [dist (distance-fn (:term node) element)
            children (:children node)
            child (get children dist)]
          (if (= 0 dist)
            (merge-with conj node {:matches element})
            (if child
              (assoc-in node [:children dist]
                        (insert element child distance-fn))
              (assoc-in node [:children dist]
                        (new-node element)))))))
```

```clojure
(defn query
  ([term threshold node]
    (flatten (query term threshold node (:distance-fn node) [])))

  ([term threshold node distance-fn matches]
    (let [dist (distance-fn (:term node) term)
          query-range (range (- dist threshold)
                             (+ 1 (+ dist threshold)))
          to-query (remove nil?
                           (map (fn [score] (get (:children node) score))
                                query-range))]

        (concat (if (<= dist threshold) [node])
                matches
                (map (fn [n] (query term threshold n distance-fn matches)) to-query)))))
```

```clojure
(defn treemap [a-fn root]
  (if (:term root)
    (map a-fn (tree-seq (fn is-branch? [node] (seq (:children node)))
                        (fn children [node] (vals (:children node)))
                        root))
    `()))
```

# Parallelization

```clojure
(defn json-to-records [coll]
  (sort-records
    (map (comp create from-json)
         coll)))
```

# Elapsed time: 74.2 sec

```clojure
(defn json-to-records [coll]
  (sort-records
    (pmap (comp create from-json)
          coll)))
```

# Elapsed time: 45.7 sec

38%

Separate!
IS SLOW!

```
46
47 (separate even? (range 0 2000000))
48
```

```
user=> (separate even? (range 0 20))
[(0 2 4 6 8 10 12 14 16 18) (1 3 5 7 9 11 13 15 17 19)]
```

```clojure
(defn separate
  "Returns a vector:
   [ (filter f s), (filter (complement f) s) ]"
  [f s]
  [(doall (filter f s))
   (doall (filter (complement f) s))])
```

# Elapsed time: 1.5 seconds

```clojure
(defn separate [pred coll]
  (reduce (fn [[t f] ele]
            (if (pred ele)
              [(conj t ele) f]
              [t (conj f ele)]))
          [[][]]
          coll))
```

# Elapsed time: 1.05 seconds

```clojure
(defn separate-fast-recur [pred coll]
  (loop [true-elements (transient [])
         false-elements (transient [])
         my-coll coll]
    (if (not (empty? my-coll))
      (let [curr (first my-coll)
            tail (rest my-coll)]
        (if (pred curr)
          (recur (conj! true-elements curr) false-elements tail)
          (recur true-elements (conj! false-elements curr) tail)))
      [(persistent! true-elements) (persistent! false-elements)])))
```

# Elapsed time: 1.1 seconds

```clojure
(defn separate-fast-doseq [pred coll]
  (let [true-elements (transient [])
        false-elements (transient [])]
    (doseq [curr coll]
      (if (pred curr)
        (conj! true-elements curr)
        (conj! false-elements curr)))
    [(persistent! true-elements) (persistent! false-elements)]))
```

# Elapsed time: 0.8 seconds

46%

# Implementation

# Test-Driven

```
building the model
- stores each record keyed by its e-mail address
- adds identical records to the same hash key
- adds records with an edit distance of 1 to the same hash key
- normalizes e-mail addresses

detecting fraud by edit distance
- detects all 4 records as fraud
- assigns the same group id to each record

calc-lead-ip-key
- joins the lead id and IP address with a colon
- strips white space

detect-dupe-leads-and-ip
- ignores records without a lead_id

pw-hash-and-ip-key
- simply joins the two strings
- strips white space

a detector that finds duplicate password and ip pairs
  when there is fraud
  - finds fraud
  when ips are missing
  - ignores records that have no ip address
  when passwords are missing
  - ignores records that have no password

reporting
  chosing a formatter
  - picks the computer format by default
  - picks the human format when passed the --human-readable switch
  human-readable-output
  - should print the summary header
  - should print the summary detail
  - should print the detail header
  - should print the detail row

Finished in 1.58497 seconds
96 examples, 0 failures
```

# Rspec-Like output & Autorunner Via Speclj

# End-to-End Functional Tests

```
10  (describe "processing an empty file"
11    (with input-data "")
12    (with return
13          (sh "./bin/process" :in @input-data :return-map true))
14    (with response (.split (:out @return) "\n"))
15
16    (it "should put EOF at end of file"
17      (should= "EOF" (second @response)))
18    (it "should not write anything to error"
19      (should= "" (:err @return))))
```

# Generative Fixtures (via re-rand)

```
:user_email (re-rand #"[a-z]{5}@[a-z]{5}\.com")
:user_first_name (re-rand #"[a-z]{4}")
:user_last_name (re-rand #"[a-z]{4}")
:user_created_at (re-rand #"2[0-9]{3}-\d{2}-\d{2} \d{2}:\d{2}")
```

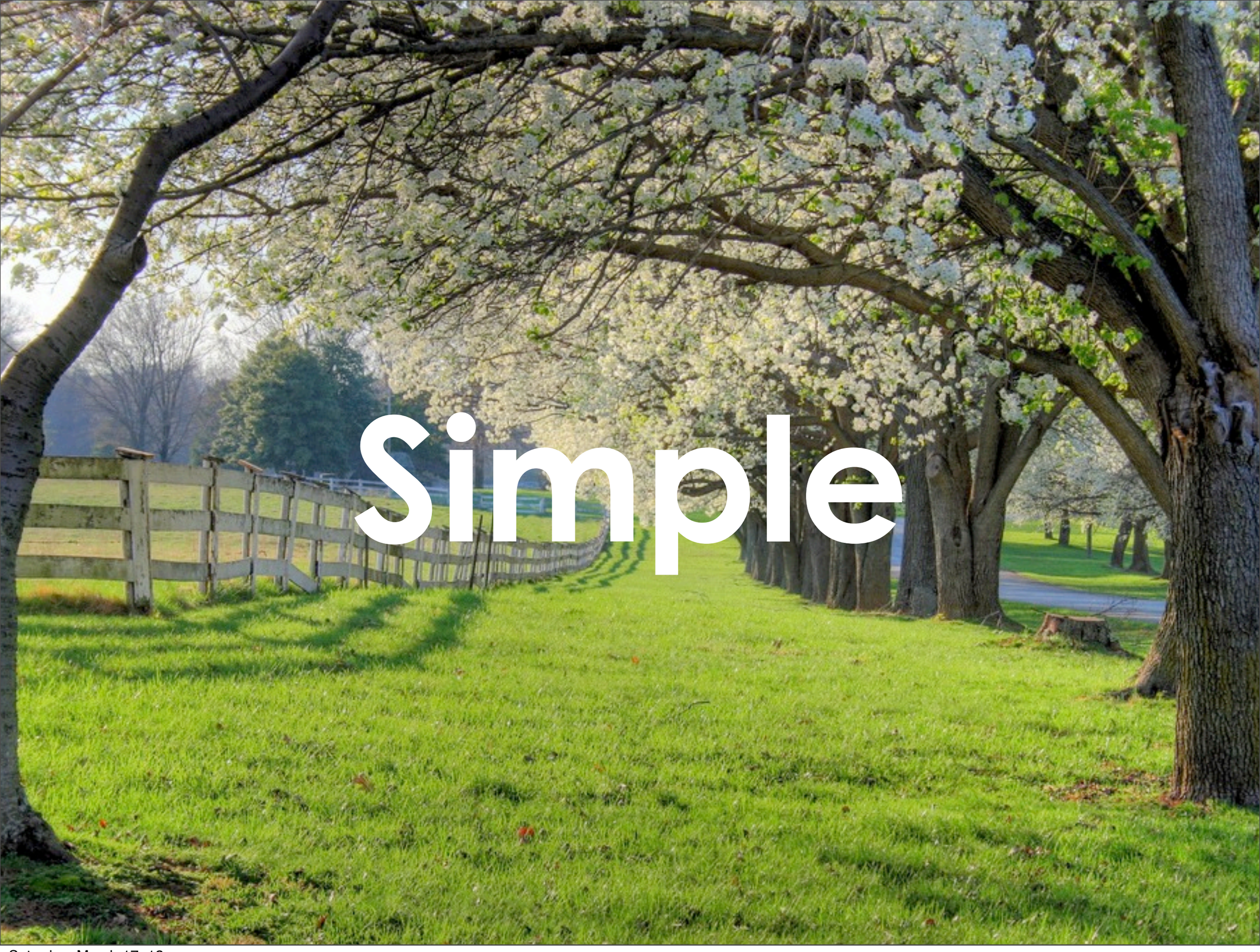
Extensible

# Easy to add detectors

```
36 (def detectors
37   [detect-duplicate-emails
38    detect-duplicate-emails-by-distance-bktree ])
```

```
10 (defn calculate-key [record]
11   (normalize-email (:user_email (:raw record))))
12
13 (defn detect-duplicate-emails-by-distance-bktree [state]
14   (edit-distance-detector
15     state
16     {:detector-name "duplicate/similar e-mail address"
17      :key-fn calculate-key
18      :filter-fns [cancelled? in-email-whitelist]
19      :max-distance 1
20      :min-occurances (:min-fraud-threshold *config*)}))
```

# Lessons


Simple

# Refactoring

```clojure
(defn calculate-key [record]
  (normalize-email (:user_email (:raw record))))

(defn distance-fn [a b]
  (let [email-a (calculate-key a)
        email-b (calculate-key b)
        dist (hamming-distance email-a email-b)]
    (if (<= dist 1)
      0
      dist)))

(defn build-tree [records]
  (reduce (fn [tree record] (bk/insert record tree))
          (bk/root distance-fn)
          records))

(defn- prepare-fraud [fraud]
  (assoc-all (mapcat values-with-group fraud)
             :status "fraud" :detected_by "duplicate/similar e-mail address"))

(defn has-non-empty-key? [record]
  (not (empty? (calculate-key record))))

(defn has-sufficient-occurences? [values]
  (> (count values) (:min-fraud-threshold *config*)))

(defn groups-sufficient-for-fraud [[record values]]
  (and (has-non-empty-key? record) (has-sufficient-occurences? values)))

(defn detect-duplicate-emails-by-distance-bktree [state]
  (let [fraudulent (:fraudulent state)
        tree (build-tree (:unknown state))
        nodes (bk/treemap (fn [node] [(:term node) (cons (:term node) (:matches node))]) tree)
        [fraud unknown] (separate-fast groups-sufficient-for-fraud nodes)]

    (assoc state :fraudulent (concat fraudulent (prepare-fraud fraud))
                 :unknown (mapcat values-with-group unknown))))
```

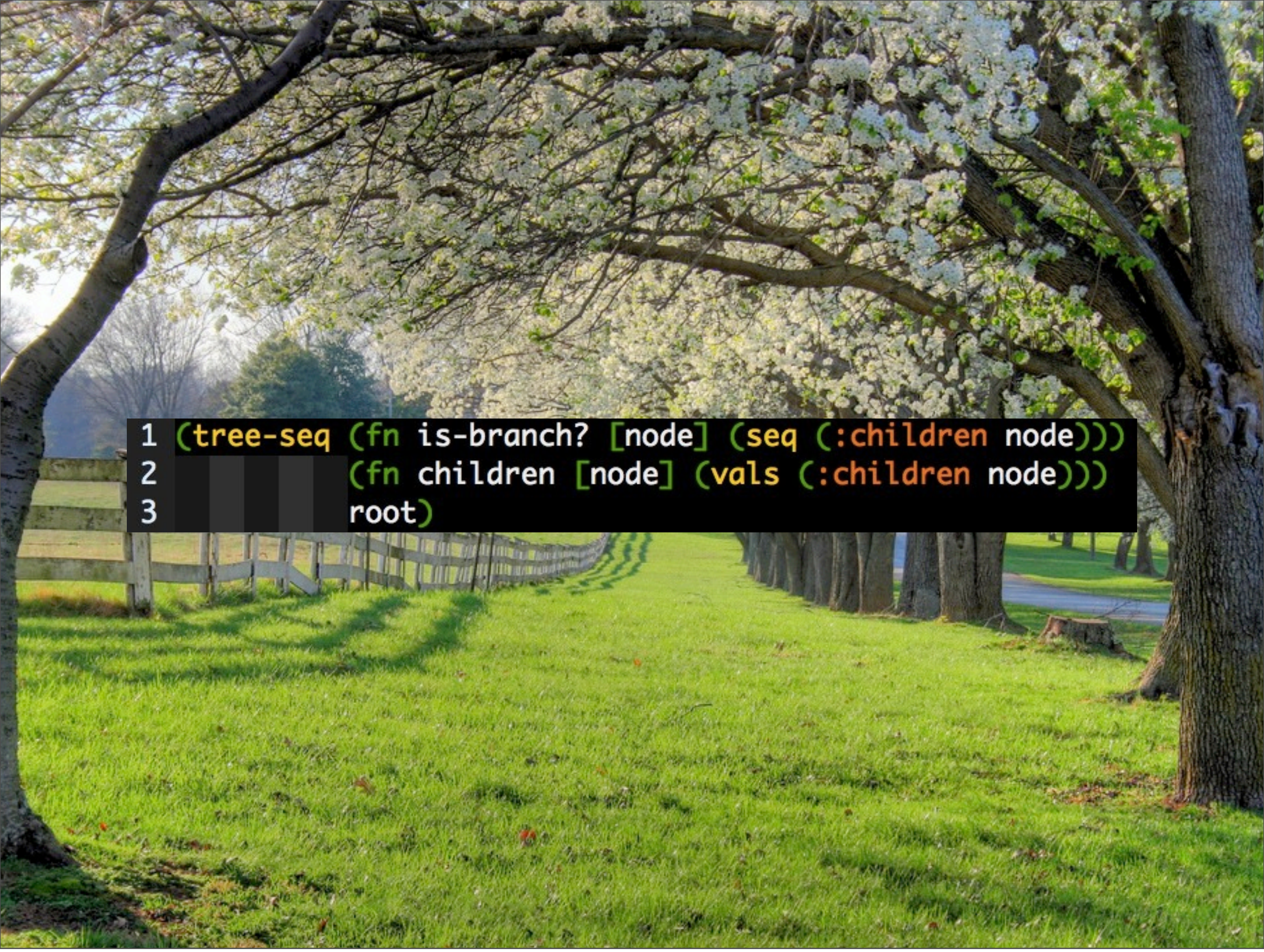
```clojure
(defn- calculate-key [record]
  (calculate-address-key record))

(defn detect-duplicate-address-by-distance-bktree [state]
  (edit-distance-detector
    state
    {:detector-name "duplicate/similar billing address"
     :filter-fns [cancelled? avs-success?]
     :key-fn calculate-key
     :max-distance 1
     :min-occurances (:min-fraud-threshold *config*)}))
```

# Be Declarative

```clojure
(defn- calculate-key [record]
  (calculate-address-key record))

(defn detect-duplicate-address-by-distance-bktree [state]
  (edit-distance-detector
    state
    {:detector-name "duplicate/similar billing address"
     :filter-fns [cancelled? avs-success?]
     :key-fn calculate-key
     :max-distance 1
     :min-occurances (:min-fraud-threshold *config*)}))
```



```
(tree-seq (fn is-branch? [node] (seq (:children node)))
          (fn children [node] (vals (:children node)))
          root)
```

# Encapsulate Data

Steep Curve

# How did we do it?

Be Prepared

# Demonstrate Value

# Play to Clojure's Strengths

No Job too Small

# Thank You!

- http://www.flickr.com/photos/bigluzer/2290456524/sizes/l/in/photostream/
- http://www.flickr.com/photos/anotherphotograph/2276607037/sizes/m/in/photostream/
- http://www.flickr.com/photos/kmtucker/3355551036/sizes/l/in/photostream/
- http://www.flickr.com/photos/tropicaliving/4379503242/
- http://www.flickr.com/photos/madcow/146030923/sizes/l/in/photostream/

# Test Deck

- This is a test deck
- Another line
- dsdad
  - sdsds
  - sdsds
  - sdsd
- Testing

  - indented
  - indented

Cool Things
Willenbrock
Tarnstedt

# Cascalog

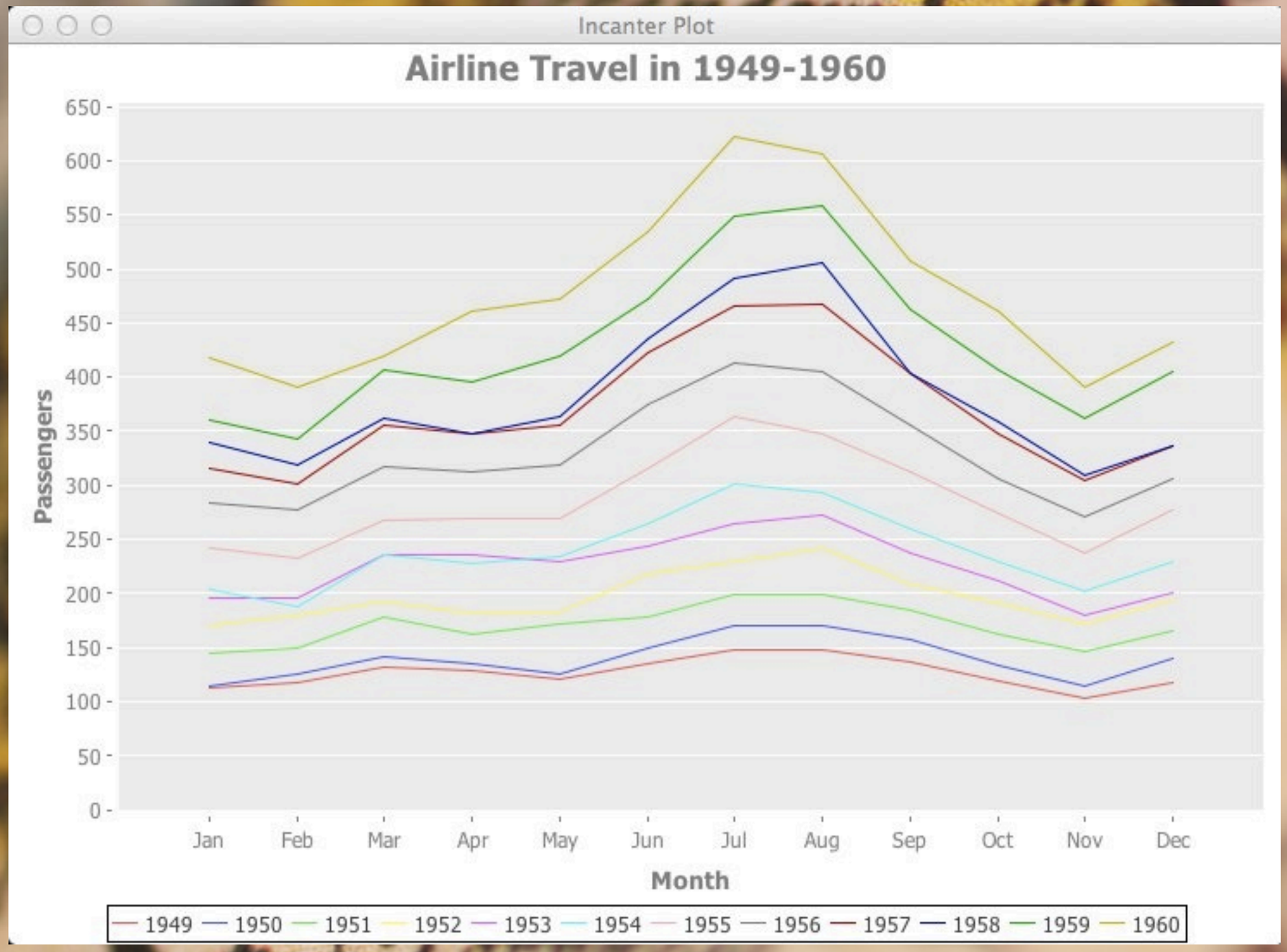
a Clojure-based query language for Hadoop

# Storm

Storm is a distributed realtime computation system.

# Incanter

R-like platform for statistical computing and graphics.


Incanter Plot
Airline Travel in 1949-1960
Passengers
650
600
550
500
450
400
350
300
250
200
150
100
50
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960

```clojure
(use '(incanter core charts datasets))

(def data (get-dataset :airline-passengers))

(view (line-chart (sel data :cols 2)
                  (sel data :cols 1)
                  :group-by (sel data :cols 0)
                  :title "Airline Travel in 1949-1960"
                  :legend true
                  :y-label "Passengers"
                  :x-label "Month"))
```

# Gaeshi

Rails like dev/publishing env
for google app engine.

# Joodo

Rails like dev/publishing env
for heroku.