# Why Prismatic Goes Faster With Clojure

# One Slide Summary

1. **F**ine-grained **C**omposable **A**bstractions > Monolithic Frameworks

2. lets you make **FCA**

3. **<3**

# About Prismatic

- We learn about your interests
- Personalized feeds based on interests
- Explore new interests

Live Demo At End

London Apartment with Glor… | Clojure - home | Prismatic | Prismatic | Prismatic

alhost/newsfeed#topic/Interior%20Design/

Explore new interests...

## Interior Design

61

Modern Interiors — 16 hours ago

### London Apartment with Glorious Interior Architecture

Interior Design · Architecture · Lofts · London — 4 people shared this

This London apartment has tons of potential - the interior architecture is glorious. We just had to share it with you. Beams on the ceiling, gothic style arched windows, arched details in the walls, lots of brick, a ladder to the second floor ... it's all just very cool. It seems to beg for an industrial style interior, don't you think? Those naked pendant lights seem so happy here. As it is, it lacks some coziness (and a couch!), but it's nothing some soft textures and warm materials couldn't fix. We often post about beautifully finished interiors, ...

pdd — Anyone think that this apartment interior is stripping it back a bit too far? ^RBH
—@PDDinnovation

Comments on: Mission Bowling Club in action for t… — 10 related stories — 6 hours ago

### Mission Bowling Club in action for the very first time!

Bowling · Interior Design — 7 people shared this

Our BFF Jess Kelso went bowling at a Mission Bowling Club preview party last night, and took a bunch of beautiful photos and told us all about it: Apparently they rolled the scissor lift out 5 minutes before we arrived, but you never would have known it — the space looked amazing. As well as being responsible for a lot of the decor inside (including the curtains and the bowling stripes along the wall), Sommer's mum has also ...


Get a sneak peek at the @MissionBowling Club, opening next Monday By 16th St BART
—@myBART

Here's what a fully operational Mission Bowling Club (including its kitchen's food) looks like
—@missionmission

**Home**

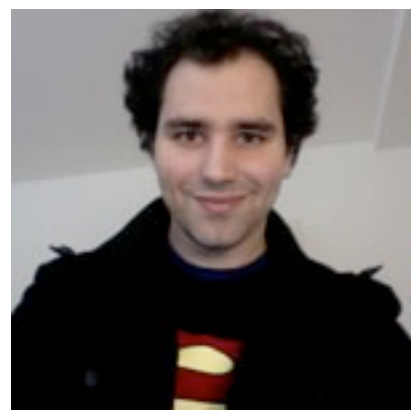
Activity
- Favorites
- Shared
- Read

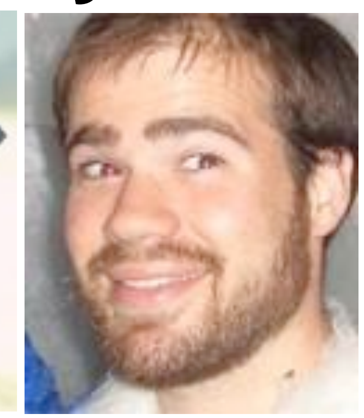
Interests
- Social
- Global
- Dresses
- Pets
- Interior Design
- Chairs
- Kitchens
- Typography
- Graphic Design
- Design
- Posters
- Architecture
- Urban Planning

# Our Backend Team

**Three CS PhDs in AI**

Me | Aria | Jenny | Jason



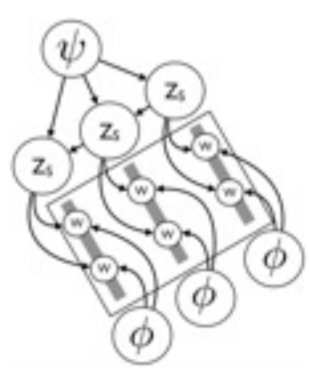


Zero Brogrammers

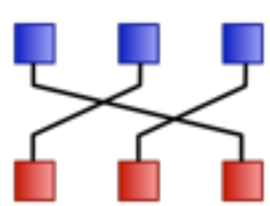
# What We Build

Web crawlers

Social Graph Analysis



Topic Models



Relevance Ranking

# Newsfeeds

- Real-time indexing of social, entity elements
- Online clustering of related stories
- Real-time personalized reranking of feeds
- Must serve requests in under about 200ms

# Our Design Approach

We tend to roll our own.

Libraries >> Frameworks

99.9% Clojure, 0.1% Java

# Flop Library

- We do lots of `double[]` processing
- For efficiency, often in-place mutation
- Native Clojure makes this a PITA

```
;; Add (5.0+j) to j-th element of array
(dotimes [j (alength arr)]
  (aset xs j (+ 5.0 j (aget xs j))))
```

# Flop Library

- Even type-hinting **can** yield inefficient code
- In Flop,

```
// Add (5.0+j) to j-th element of array
(afill! [[j v] arr] (+ v 5.0 j))
```

- Succinct and efficient!
- **Can't** yield code with reflection

# Flop Library

- Rare use of macros in our code
- **doarr**: doseq for `double[]`

```
;; print all pairs from two arrays
;; ‘parallel’ looping over two arrs
;; bind value or [idx value]
(doarr [[idx val1] arr1
        val2 arr2]
  (println [idx val1 val2]))
```

# Flop Examples

**Dot Product**

$$w \cdot x = \sum_{i=1}^{n} w_i x_i$$

prediction

$$\arg\max_{\ell} w \cdot x_{\ell}$$

training

$$P(x; w) \propto \exp\{w \cdot x\}$$

Inner loop in machine learning

# Flop Examples

**Dot Product**

$$w \cdot x = \sum_{i=1}^{n} w_i x_i$$

```
(defn dot-product [^doubles ws ^doubles xs]
  (areduce ws idx sum 0.0
   (+ sum (*  (aget ws idx)
              (aget xs idx)))))
```

# Flop Examples

**Dot Product**

$$w \cdot x = \sum_{i=1}^{n} w_i x_i$$

```
double dotProd(double[] ws, double[] xs) {
  double sum = 0.0;
  for (int i=0; i < xs.length; ++i) {
    sum += ws[i] * xs[i];
  }
  return sum;
}
```

# Flop Examples

**Dot Product**

$$w \cdot x = \sum_{i=1}^{n} w_i x_i$$

```
(defn dot-product [ws xs]
  (flop/asum [w ws x xs] (* w x)))
```

# Flop Examples

**Expected**

**Log Probs**

$$\psi_i = \mathbb{E}_\theta(\lg \theta_i | \alpha)$$

$$\theta \sim \texttt{Dirichlet}(\alpha)$$

$$\psi_i = \gamma(\alpha_i) - \gamma\left(\sum_{i=1}^{n} \alpha_i\right)$$

Inner loop in topic modeling

$\gamma(x)$ Digamma Function

expensive + gnarly

Taylor approximation

# Flop Examples

**Expected**

**Log Probs**

$$\psi_i = \mathbb{E}_\theta(\lg \theta_i | \alpha)$$

$$\theta \sim \texttt{Dirichlet}(\alpha)$$

```
(defn exp-log-probs [alphas]
  (let [log-z (digamma (asum alphas))]
   (flop/amap [a alphas]
     (- (digamma a) log-z))))
```

# Flop Library

- Comparable performance to tuned Java
  - State-of-the-art numerical optimization in < 180 lines
  - LDA-style topic modeling with variational inference < 180 lines

# Store Library

- Storage and aggregation abstractions
- Key-value protocol over Memory, File system, S3, BDB, Mongo, SQL
- implementations use specific features of underlying

# Store Library

- Key-value protocol: bucket/get, bucket/put
- the big deal: bucket/update
- can reify IMergeBucket: bucket/merge
- IWriteBucket has bucket/sync

# Store Library

- Automatic hosting for any store. f.ex. HTTP handlers for GET, PUT, MERGE ops for store & bucket.
- Easily test services by swapping persistent stores with memory stores
- Abstract over buffer & flush policies

# Store Library

```
;; MERGE 1: index bigrams
(def bigrams
  (bucket/new
   {:type :mem
    :merge (partial merge-with +)}))

;; For each word, count following words
(doseq [[before after]
         (partition-all 2 words)]
   (bucket/merge bigrams
     before {after 1}))
```

# Store Example

```clojure
(defn map-reduce [map-fn reduce-fn n xs]
 (let [bspec {:type :mem :merge reduce-fn}
       bs (repeatedly n #(bucket/new bspec))
       work (fn [b x]
              (doseq [[k v] (map-fn x)]
                (bucket/merge b k v)))
       workers (map #(partial work %) bs)]
     ;; workers process xs in par, blocking
     (do-work workers xs)
     ;; merge all bucket users
     (reduce bucket/merge-all bs)))
```

# Store Example

```
;; MERGE 2: map reduce
(defn map-reduce [map-fn reduce-fn num-workers
input]
 (let [pool (workers num-workers))
       agg-bucket #(bucket/new
         {:type :mem :merge reduce-fn})
       res (agg-bucket)
       in-queue (queue/new {:type :mem})
       sentinel (java.util.UUID/randomUUID)]
   (future (do (doseq [x input]
     (queue/offer in-queue x))
         (queue/offer in-queue sentinel)))
```

# Store Example

```
terminal-latch (CountDownLatch. 1)
mapper-latch (CountDownLatch. num-workers)
terminator (fn [x]
             (if (= x sentinel)
               (.countDown terminal-latch)
               (map-fn x)))
defaults {:f terminator
          :in #(queue/poll in-queue)}
buckets (repeatedly num-workers agg-bucket)
```

# Store Example

```
(doseq [b buckets]
 (exec/submit-to pool
   (let [b (agg-bucket)]
     #(if (= 0 (.getCount terminal-latch))
        (do (try
      (bucket/merge-to! b res)
      (finally (.countDown mapper-latch)))
     :done)
      (assoc defaults :out
        (fn [kvs]
     (doseq [[k v] kvs]
      (bucket/merge b k v)))))))))
```

# Store Example

```
;;block on mapper encountering the sentinel
value
(.await terminal-latch)
;;other mappers could still be processing
tasks, ensure they finish.
(.await mapper-latch)
;;ensure all reducers are merged
(doseq [b buckets]
  (bucket/merge-to! b res))
(exec/shutdown-now pool)
res))
```

# Store Library

- wrapper policies
- caching & checkpointing
- buffering & flushing
- checkpoint & drain seqs: coming in store + graph example

# Graph Library

- Stream graph computation model
- Separate specification from execution plan
- Optimized for system throughput

```
 ;; Count entities in documents
(->> (graph)
     (gmap :doc-fetch (juxt :id get-text))
     (gmapcat :ent-tag
      (fn [[id text]]
       (map (fn [ent] [id ent])
        (nlp/extract-entities text))))
    ;; Branch output to both nodes
    (>> (gmap :bmerge
          (fn [[id ent]]
            (bucket/merge ent-counts ent 1)))
        (gmap :pub
          (publisher :topic “entities”))))
```

# Graph Flexibility

- Graph input and outputs play nicely with Store and PubSub libraries
- Execution policies
  - ‘compile’ to a single fn
  - each node it’s own machine/thread-pool
- Real win: monitoring and visibility

# Graph Monitoring

- Each node monitors performance: cpu, exceptions, throughput, etc.

| node | times | throughput | % cpu | % loss |
|---|---|---|---|---|
| :doc-fetch | 450 | 1.5 | 0.10 | 0.02 |
| :ent-tag | 450 | 5.0 | 0.88 | 0.00 |
| :bmerge | 5,400 | 70.0 | 0.01 | 0.0 |
| :pub | 5,400 | 1,500 | 0.01 | 0.01 |

# Graph + Store

- Use graph to compute and monitor
- Store as terminal aggregation node
- Quickly craft systems for problems

# Graph + Store Example

- Online learning over streaming user events
- Collect statistics over time, periodically flush statistics to update existing ranking parameters
- Updating parameters is expensive so trigger batch updates after collecting 'enough' new user events

```clojure
(def params ...)
(def suff-stats (bucket/new ...)
(->> (graph)
     (gmapcat :feature-extract
       (partial event-feats params))
     (gmap :feature-accum
       (fn [[event feat val]]
        (bucket/merge suff-stats
            event {feat val}))))
(cron-job
  #(update-params! params
    (bucket/flush suff-stats))
  [60 :minutes])
```

# Demo