Large Data and Clojure

The Middle Ground Between RAM and EC2

Please ask questions as we go, I like interaction.

Mission: Probable

20k Random Sample

Population size: ~400 Million

I know I'll use the Database...

```
SELECT *
FROM some_huge_table
ORDER BY RANDOM()
LIMIT 20000;
```


HELLO
my name is
Run Time

Flip a Coin?

Streaming Sampling

Pr(e) = N / M

N = Sample Set Size

M = Remaining Population Size

Streaming Sampling

N = 20,000

M = 400,000,000

Pr(e) = 0.000005

Streaming Sampling

Decrement N when a sample is taken

Decrement M for every element

Streaming Sampling

- 1 N=2 M=7 Pr(e)=0.285 MISS
- 2 N=2 M=6 Pr(e)=0.333 HIT
- 3 N=1 M=5 Pr(e)=0.20 MISS
- 4 N=1 M=4 Pr(e)=0.25 MISS
- 5 N=1 M=3 Pr(e)=0..33 MISS
- 6 N=1 M=2 Pr(e)=0.50 MISS
- 7 N=1 M=1 Pr(e)=1.0 HIT [DONE]

Streaming Sampling

```
(sequences/random-sample-seq
  (ds/read-lines "phone-nums.txt")
                 2000000
                 200)
```

- Sample in One Pass
- Disk IO is 1x
- Avoid OutOfMemoryException

HELLO
my name is

Win

Next Lurking Issue?

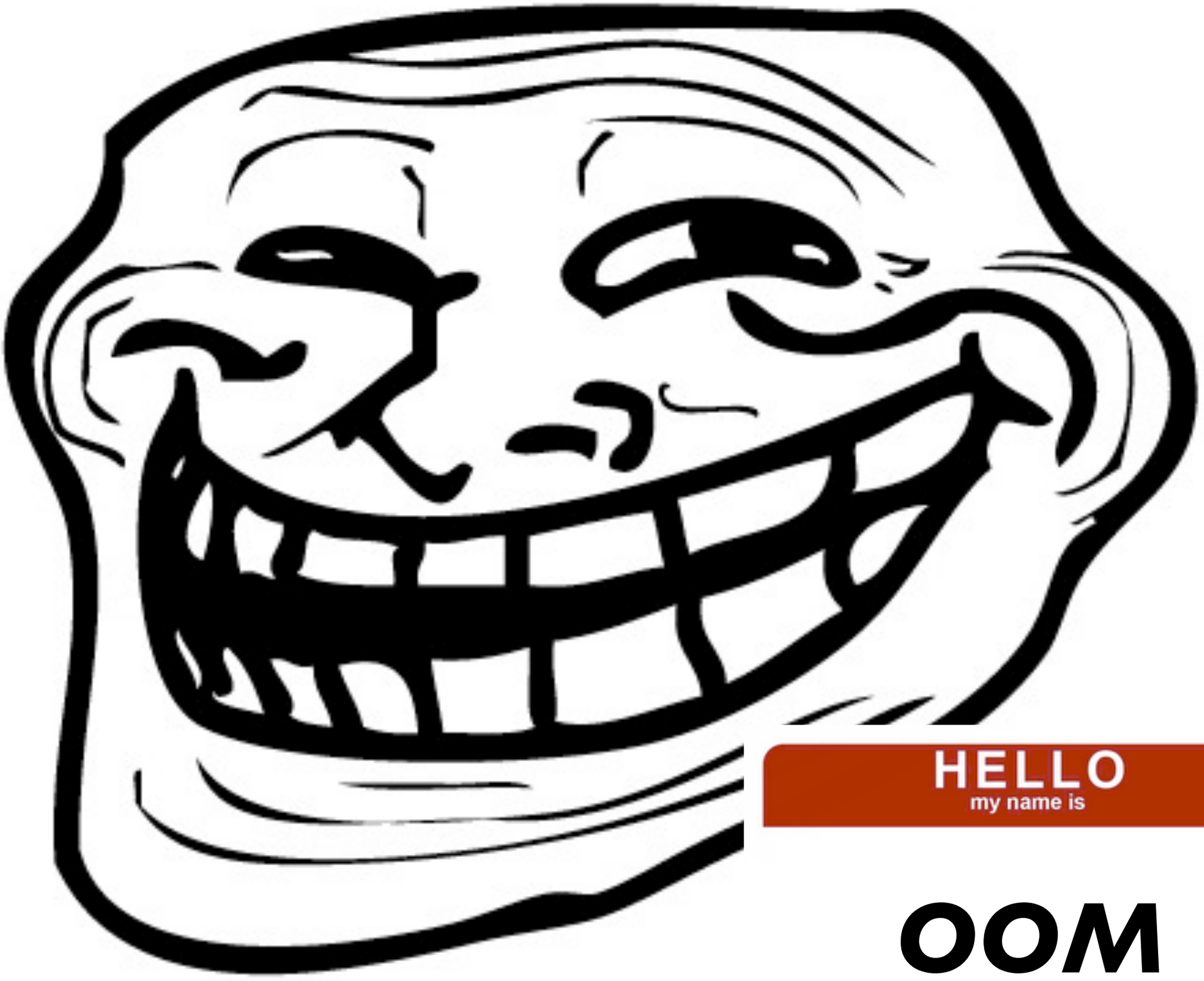

HELLO
my name is
Default

Dummy :: Default

- ‘NULL’ / ‘N/A’
- (000) 000-0000
- nobody@nobody.com
- 123 Main St.
- John Q. Public

Naive Counting

```
(defn find-dupes-naive [inp-seq]
  (reduce (fn [res item]
            (assoc res
                   item
                   (inc (get res item 0))))
          {}
          inp-seq))
```


HELLO
my name is
OOM

Is This The Only Way?

Bloom Filter

There's Some Math...

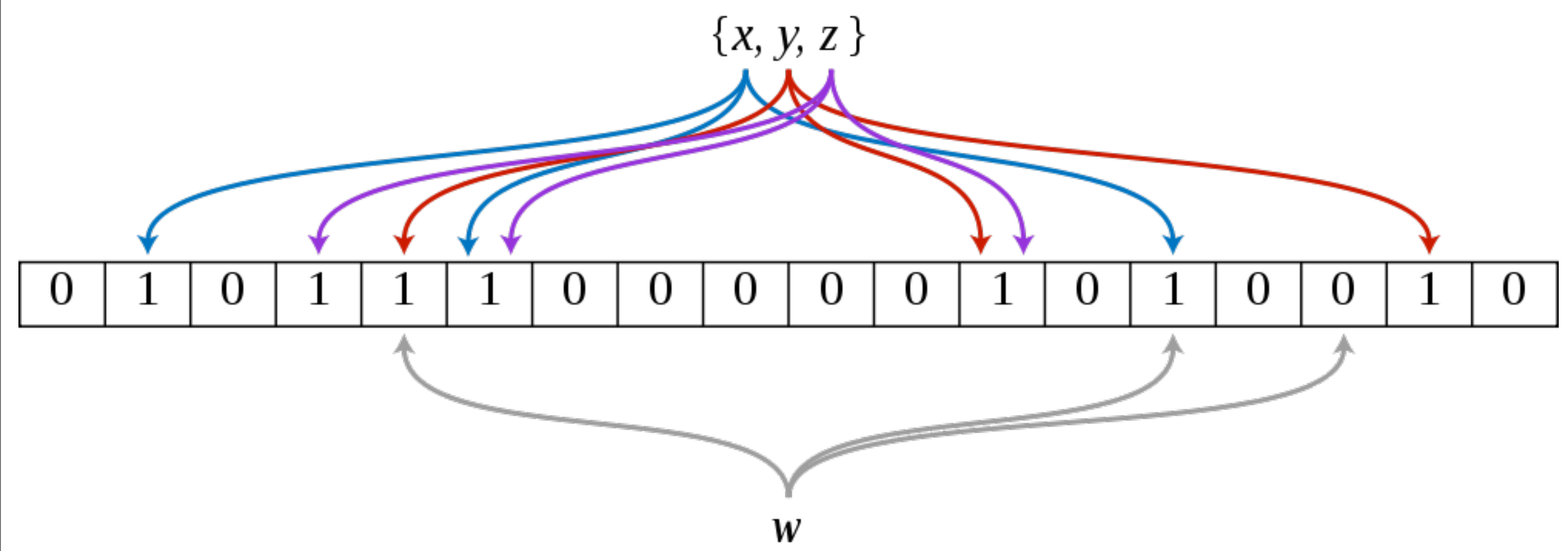
$$\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

$$m = -\frac{n \ln p}{(\ln 2)^2}.$$

$$\frac{m}{n} \ln 2 \approx \frac{9m}{13n} \approx 0.7\frac{m}{n},$$

$$\left(1 - e^{-k(n+0.5)/(m-1)}\right)^k.$$

Probabilistic Set



"foo" => show-permited-hashes and then the bit numbers...

Itsa Bloomin' Dupe

```
(defn dupes-via-bloom-filter [inp-seq psize fp-prob]
  (let [flt (bloom/make-optimal-filter psize fp-prob)]
    (reduce (fn [res item]
              (if-not (bloom/include? flt item)
                (do
                  (bloom/add! flt item)
                  res)
                (assoc res
                       item
                       (inc (get res item 1)))))))
            {}
            inp-seq)))
```

- **Identify Duplicates in One Pass**
- **Memory Usage is Bloom Filter Size + Elements Tracked**
- **Have to Evict Singletons (FPs)**

Summary Counts

You Know...

NY 14,735

PA 11,234

NJ 8,907

DE 5,191

Smells Like:

Embarrassingly Parallel

Can Haz Multi-Core?

Your machine probably has multiple cores

Maybe even multiple CPUs

FP Langs are Supposed to make it easier to leverage these eh?

(hint: They do)

Divide and Conquer!

```
split -l 100000 \
  phone-nums-with-lfsr-ids.txt \
  working-dir/inp-

wc -l working-dir/inp-a*
  100000 working-dir/inp-aa
  100000 working-dir/inp-ab
  100000 working-dir/inp-ac
  100000 working-dir/inp-ad
  100000 working-dir/inp-ae
  ...
```

Divide and Conquer!

```
(defn count-area-codes [inp-seq]
  (reduce (fn [m line]
            (let [phnum          (second (.split line "\t"))
                  [_ area-code] (first (re-seq #"\((\d+)\)" phnum))]
              (assoc m area-code (inc (get m area-code 0)))))
          {}
          inp-seq))

(apply
 merge-with +
 (map (fn [inp-file]
        (count-area-codes (ds/read-lines inp-file)))
      (map str
           (filter #(.isFile %)
                   (.listFiles (java.io.File. "working-dir/"))))))
```

Divide and Conquer!

Did you notice the letter '**p**' I added right there?

```
(defn cou
  (reduce
                                                    \t"))
                                                    -)\)" phnum))]
                                                     0)))))
          inp-seq))

(appl
 me e-with +
 (pmap (fn [inp-file]
         (count-area-codes (ds/read-lines inp-file)))
       (map str
            (filter #(.isFile %)
                    (.listFiles (java.io.File. "working-dir/"))))))
```

Tricked You

Running ‘split’ wasn’t free

It used up additional Disk Space

No Need to Split

Process ‘blocks’ / ‘chunks’ in parallel

Ensure they’re aligned with line (record) separators

(I wrapped this up into a library for you)

No Split

```
(reduce
   (fn [res counts]
     (merge-with + res counts))
   (pmap (fn [[start end]]
           (count-area-codes
             (io/read-lines-from-file-segment
              inp-file start end)))
         (partition 2 1
           (io/byte-partitions-at-line-boundaries
             inp-file
             (* 1024 1024)))))
```

- **Use Your Cores**

This Slide Left Intentionally Blank

Clojure

Sequence Abstraction

Lazy

Composable

Clojure

Easy Concurrency

pmap

Iterate Faster

Faster Runs let you work out kinks quicker

Reduces Bugs

Thank You

@kyleburton

github.com/kyleburton