Large Data and Clojure

The Middle Ground Between RAM and EC2

Mission: Probable

- 20k Random Sample
- Population size: ~400 Million

DBeee!

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


HELLO
my name is
Run Time

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]

- **Take Sample in One Pass over the Population**
- **Disk IO is 1x**
- **Disk Usage is on the Order of Sample Size**
- **Sample Size Guaranteed**

Next Lurking Issue?

HELLO
my name is

Default

Dummy Values

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

Naive Counting

```
(defn find-dupes-naieve [inp-seq]
  (reduce (fn [res item]
            (assoc res item (inc (get res item 0))))
          {}
          (map #(second (.split %1 "\t")) inp-seq)))
```


HELLO
my name is
OOM

Only Way?

Bloom Filter

There's Some Math...

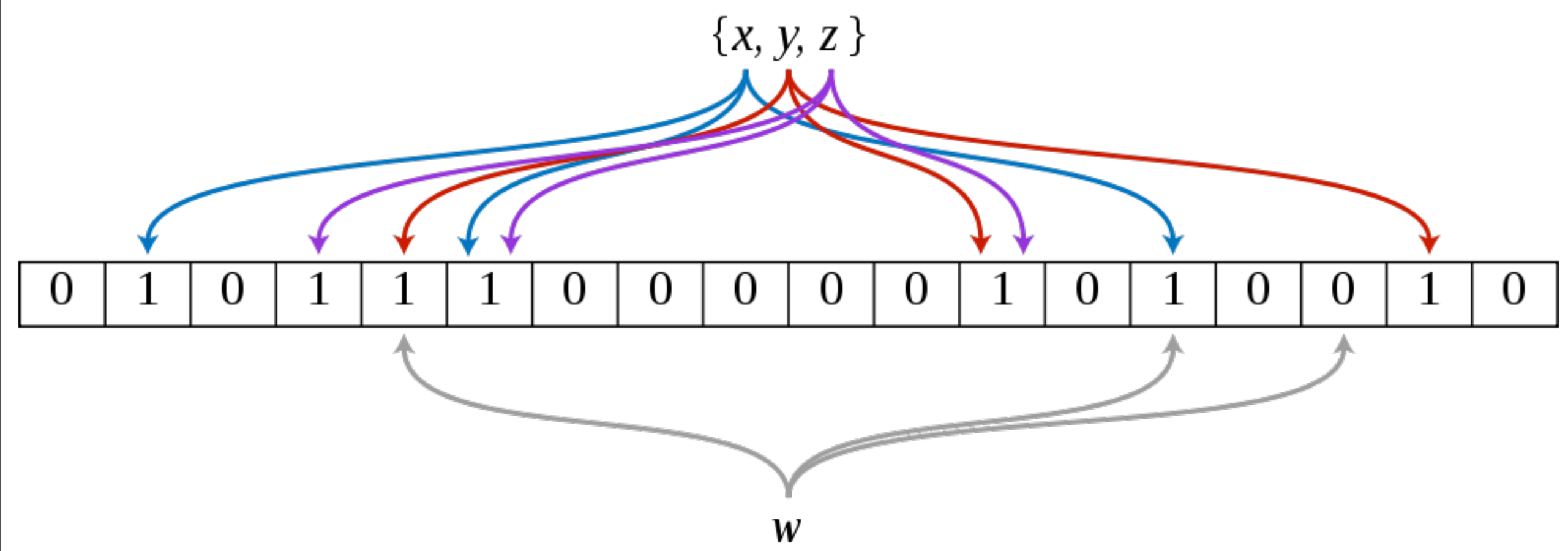
$$\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

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

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

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

Probabilistic Set



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

Naive Counting

```
(defn find-dupes-with-bloom-filter [inp-seq expected-size fp-prob]
  (let [flt (bloom/make-optimal-filter expected-size 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)))))
            {}
            (map #(second (.split %1 "\t")) inp-seq))))
```