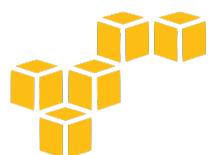
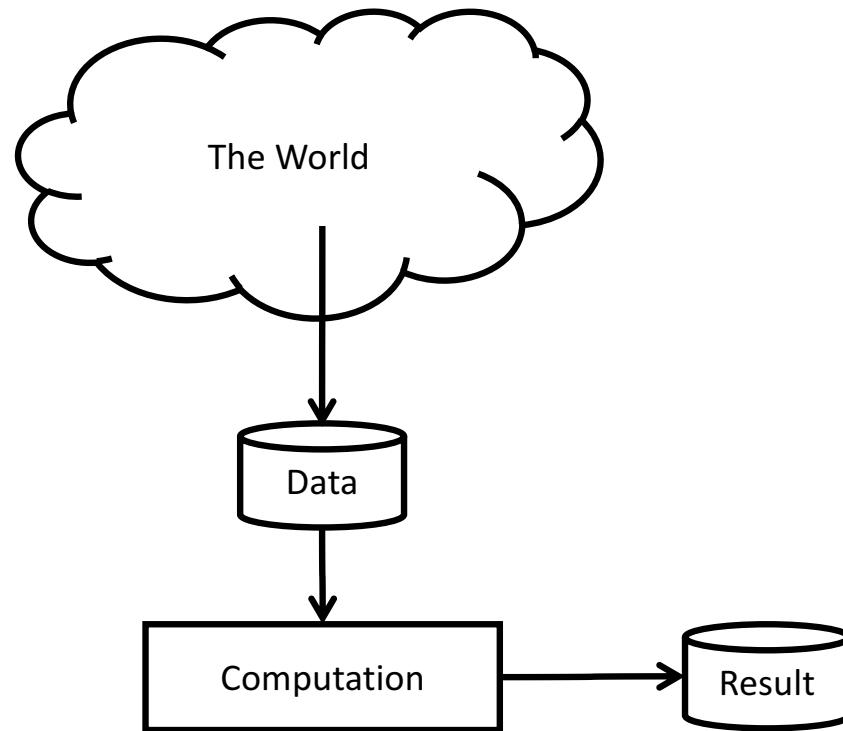


Data Mining Distributed Streams

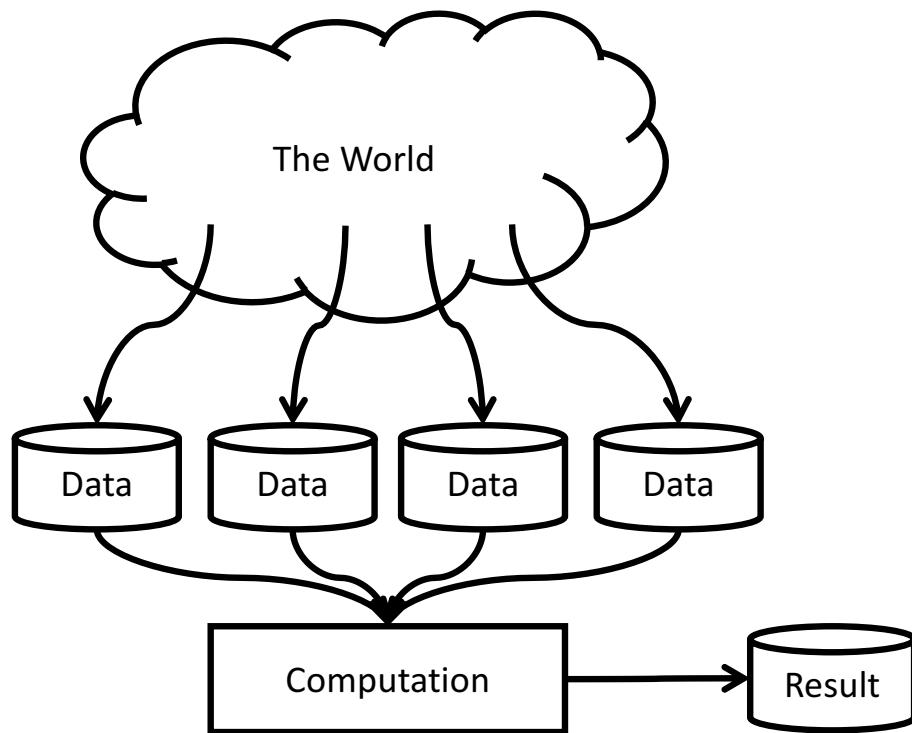
Edo Liberty
Principal Scientist
Amazon Web Services



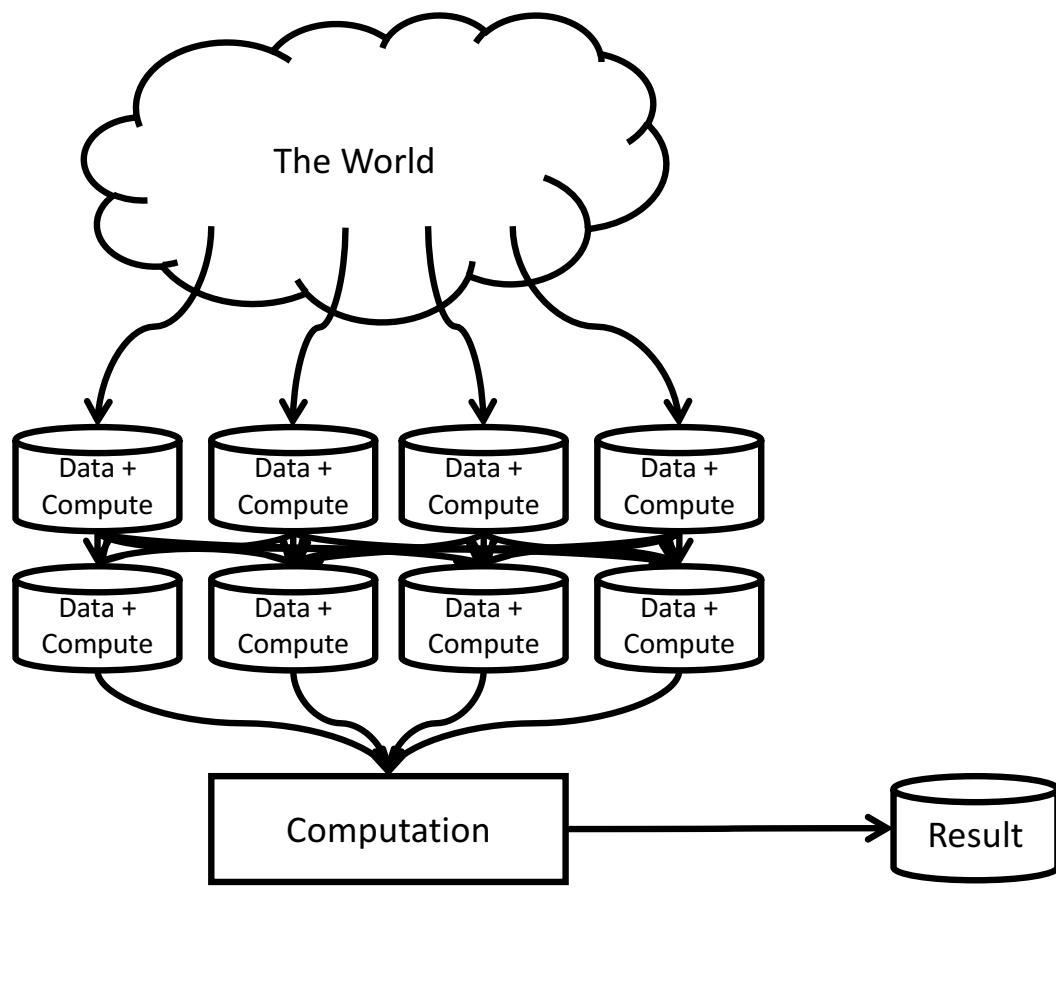
Single machine data processing



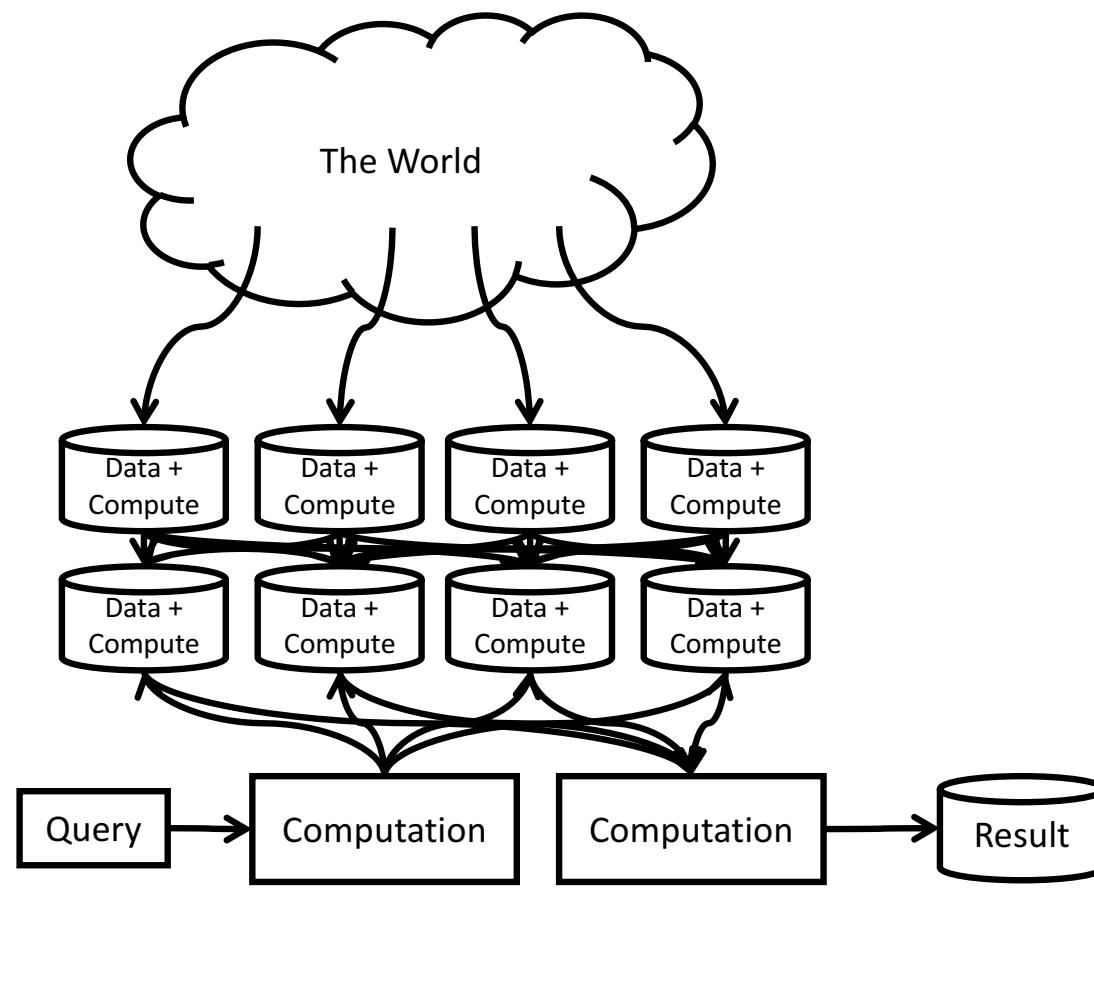
Distributed storage



Distributed compute (map/reduce, MPI, ...)



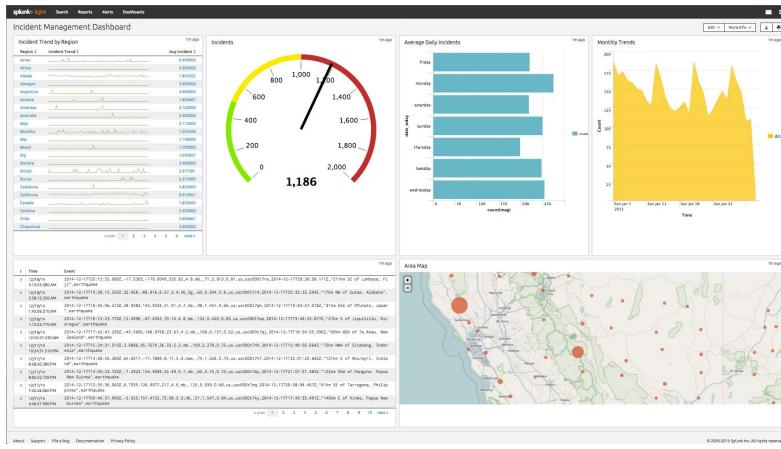
Distributed model (indexes, databases, Spark...)



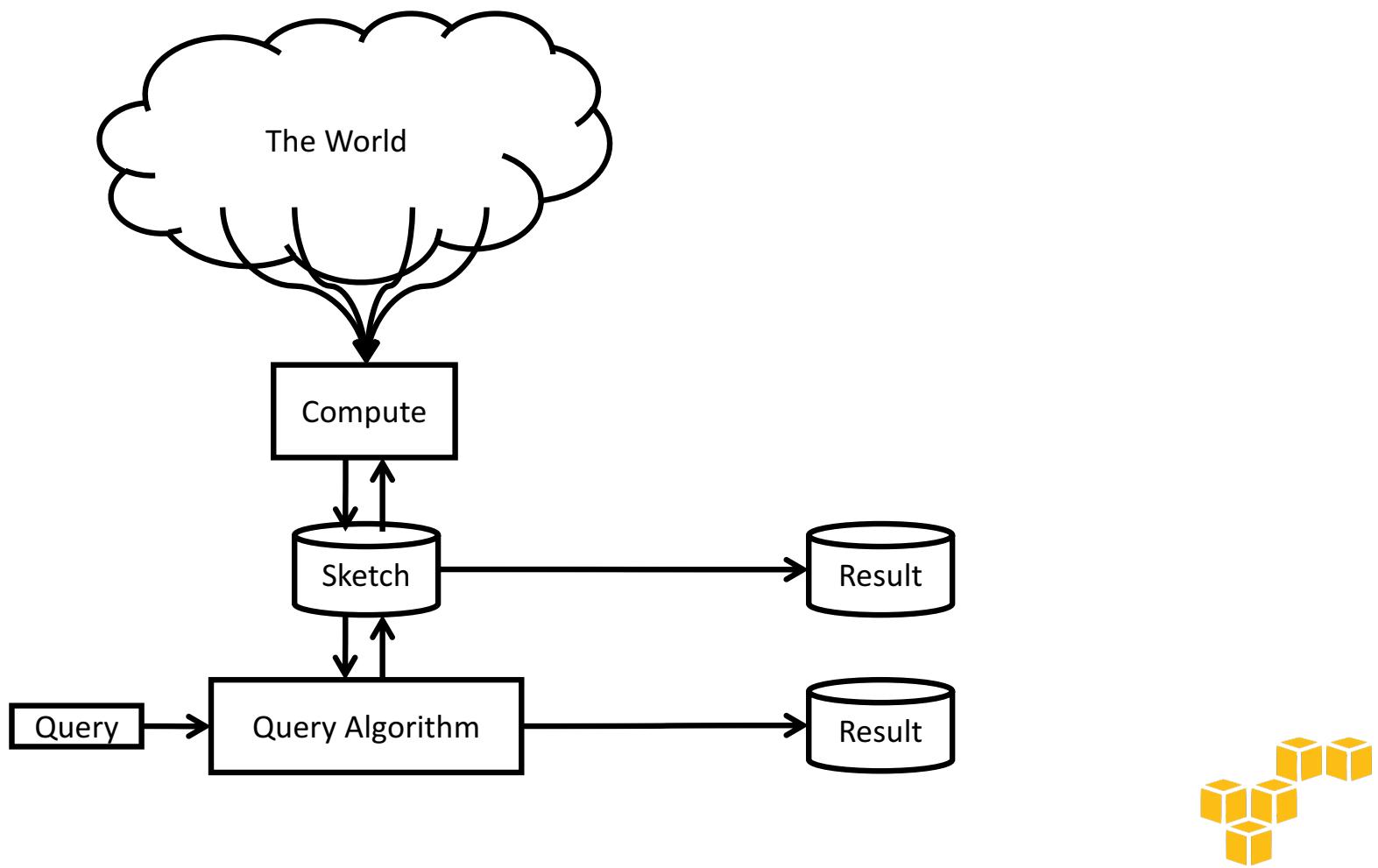
207 big-data infographics (a meta infographic)



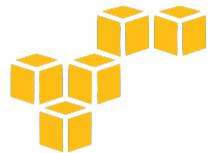
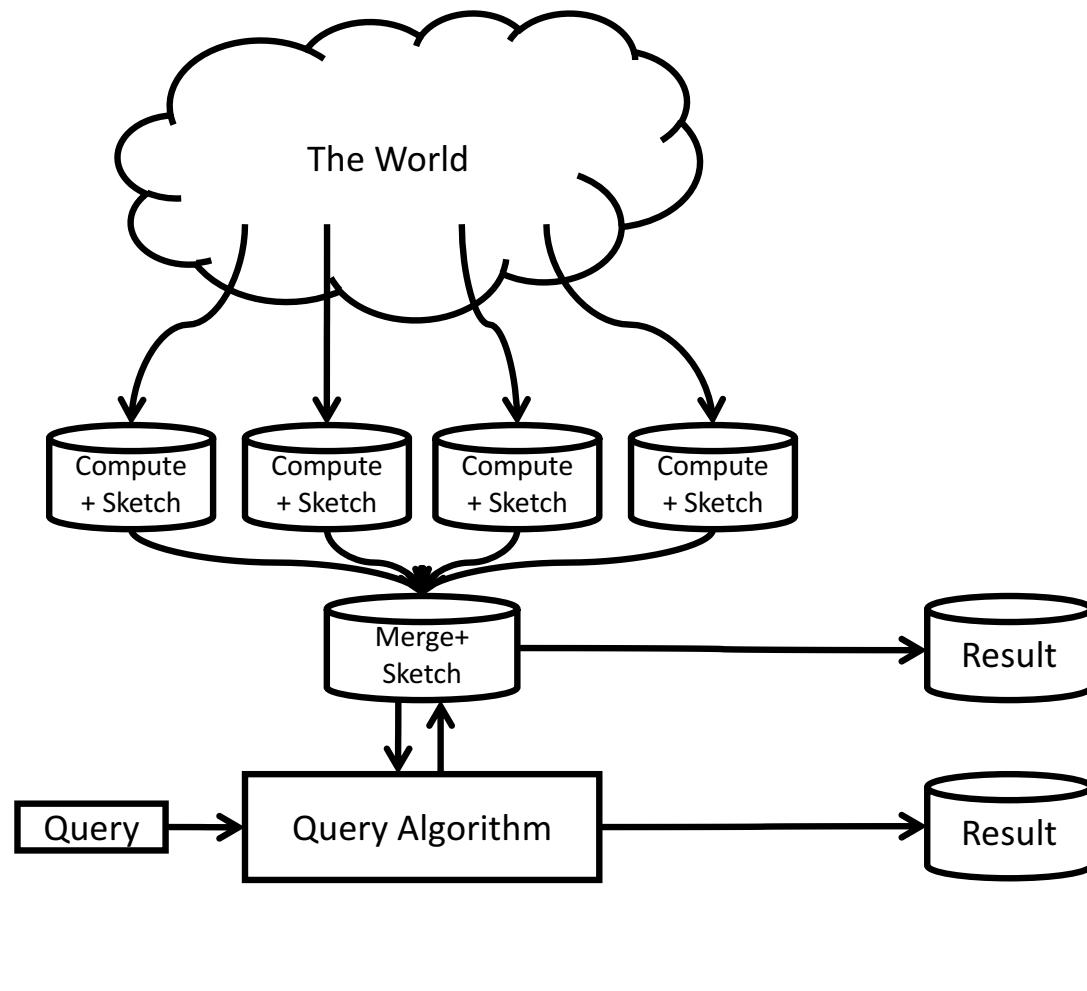
Amazon Kinesis Analytics



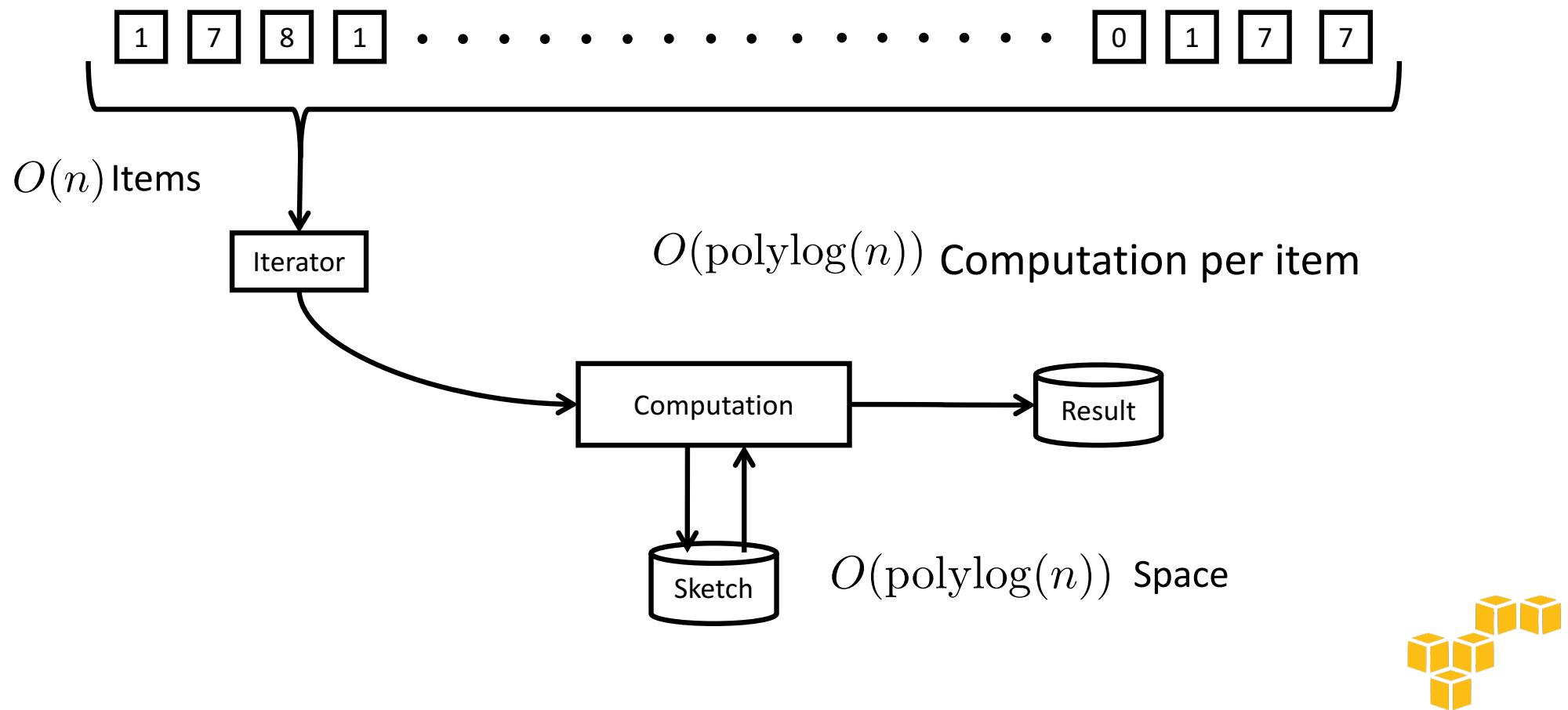
The streaming model



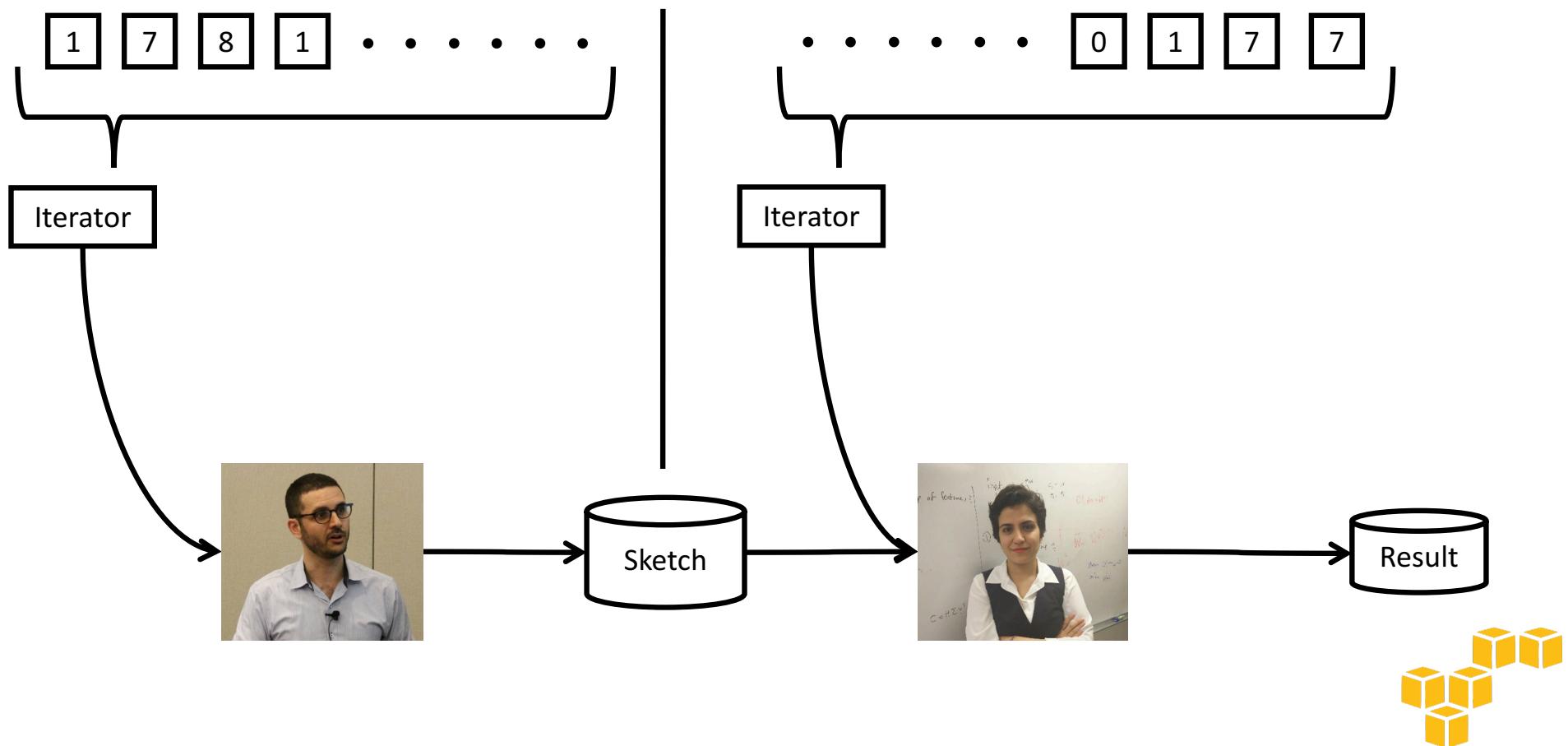
The distributed streaming model



The streaming model (more accurately)



Communication complexity



What Can we do in this model?

Items

(words, IP-addresses, events, clicks,...)

- Item frequencies
- Approximate Quantiles
- Counting distinct elements
- Moment and entropy estimation
- Approximate set operations
- Sampling

Vectors

(text documents, images, example features,...)

- Dimensionality reduction
- Clustering (k-means, k-median,...)
- Linear Regression
- Machine learning (some of it at least)

Matrices

(text corpora, recommendations, ...)

- Covariance estimation matrix
- Low rank approximation
- Sparsification

Graphs*

(social networks, communications, ...)

- Connectivity
- Cut Sparsification
- Weighted Matching



What Can we do in this model?

Items

(words, IP-addresses, events, clicks,...)

- Item frequencies ←
- Approximate Quantiles ←
- Counting distinct elements ←
- Moment and entropy estimation
- Approximate set operations
- Sampling

Vectors

(text documents, images, example features,...)

- Dimensionality reduction
- Clustering (k-means, k-median,...)
- Linear Regression
- Machine learning (some of it at least)

Matrices

(text corpora, recommendations, ...)

- Covariance estimation matrix
- Low rank approximation
- Sparsification

Graphs*

(social networks, communications, ...)

- Connectivity
- Cut Sparsification
- Weighted Matching



Frequency Counting

Misra, Gries. Finding repeated elements, 1982.

Demaine, Lopez-Ortiz, Munro. Frequency estimation of internet packet streams with limited space, 2002

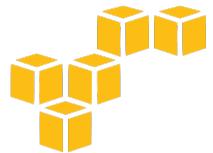
Karp, Shenker, Papadimitriou. A simple algorithm for finding frequent elements in streams and bags, 2003

The name "Lossy Counting" was used for a different algorithm by Manku and Motwani, 2002

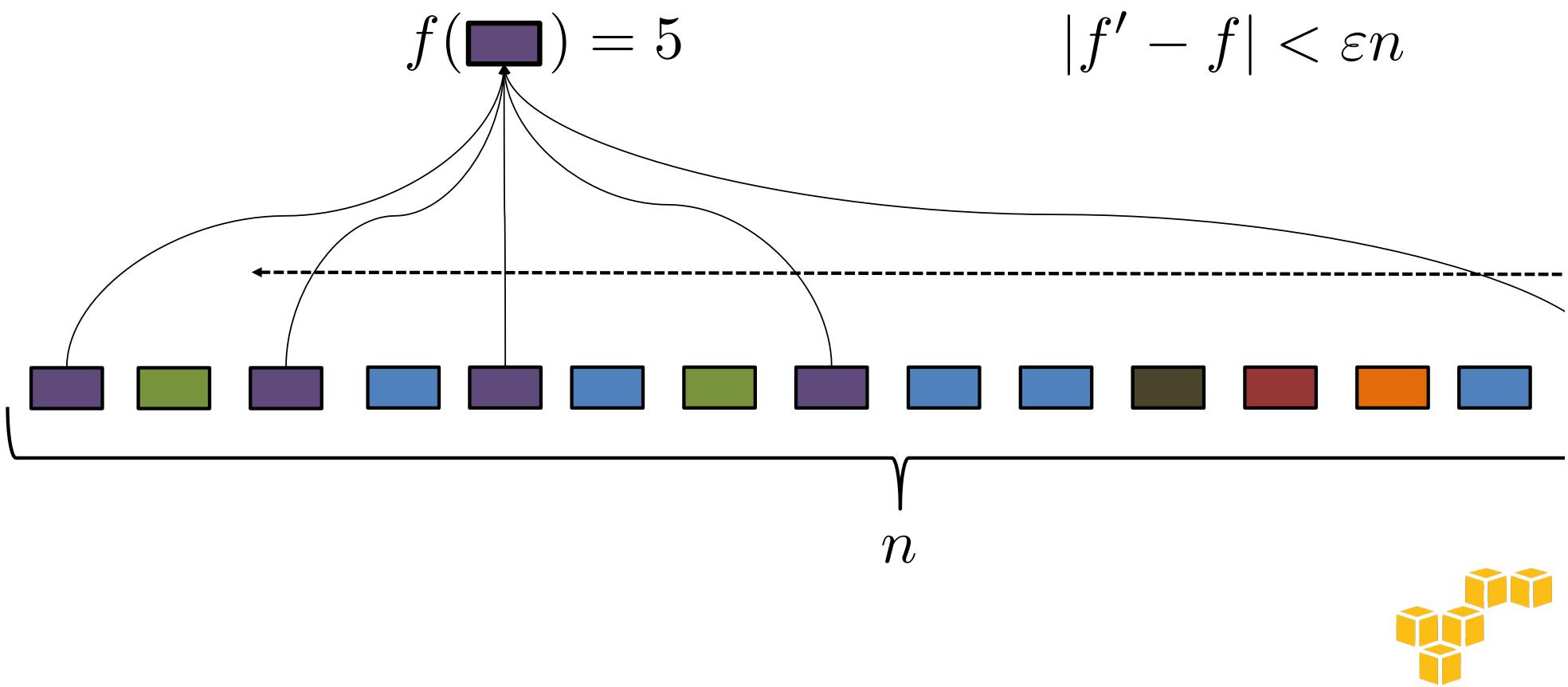
Metwally, Agrawal, Abbadi, Efficient Computation of Frequent and Top-k Elements in Data Streams, 2006

Charikar, Chen, Farach-Colton, Finding frequent items in data streams, 2002

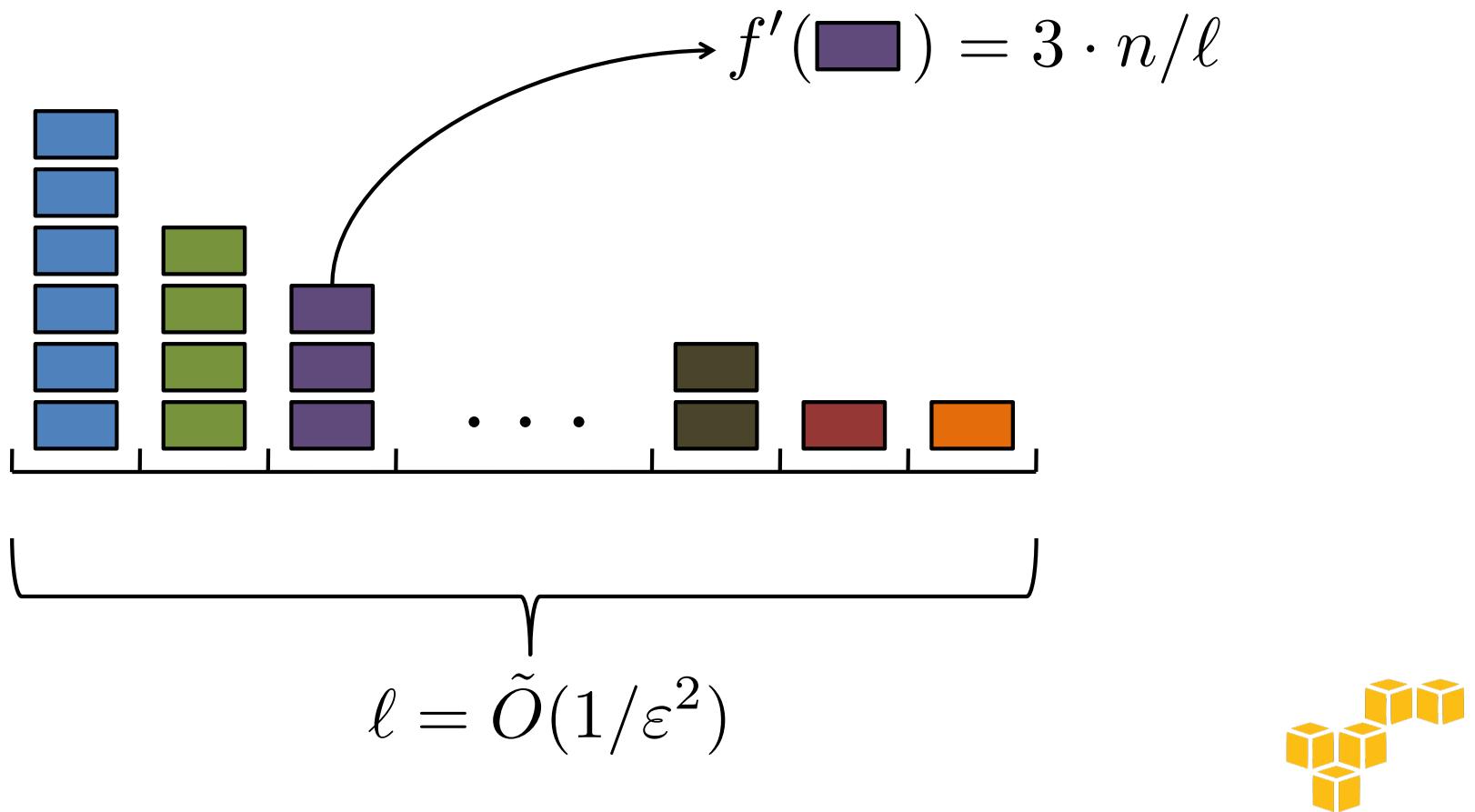
Cormode, Muthukrishnan, An Improved Data Stream Summary: The Count-Min Sketch and its Applications.

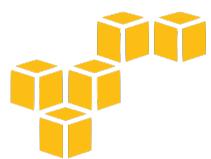
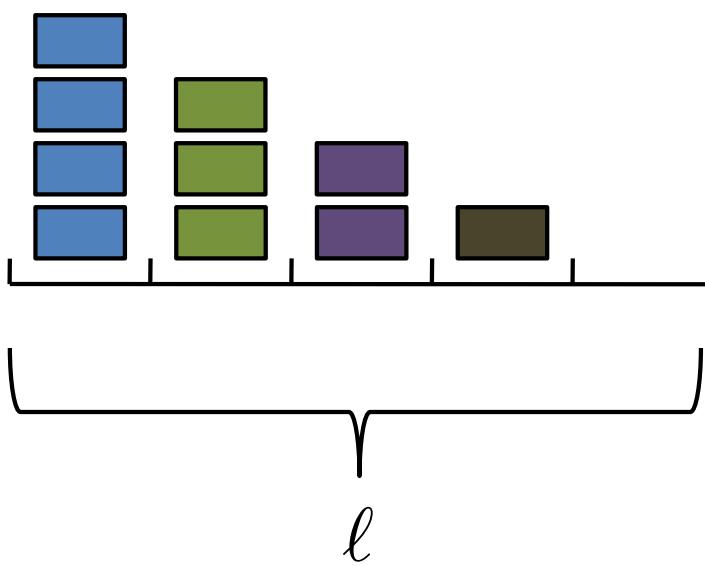


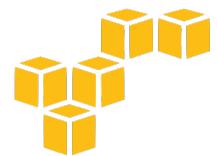
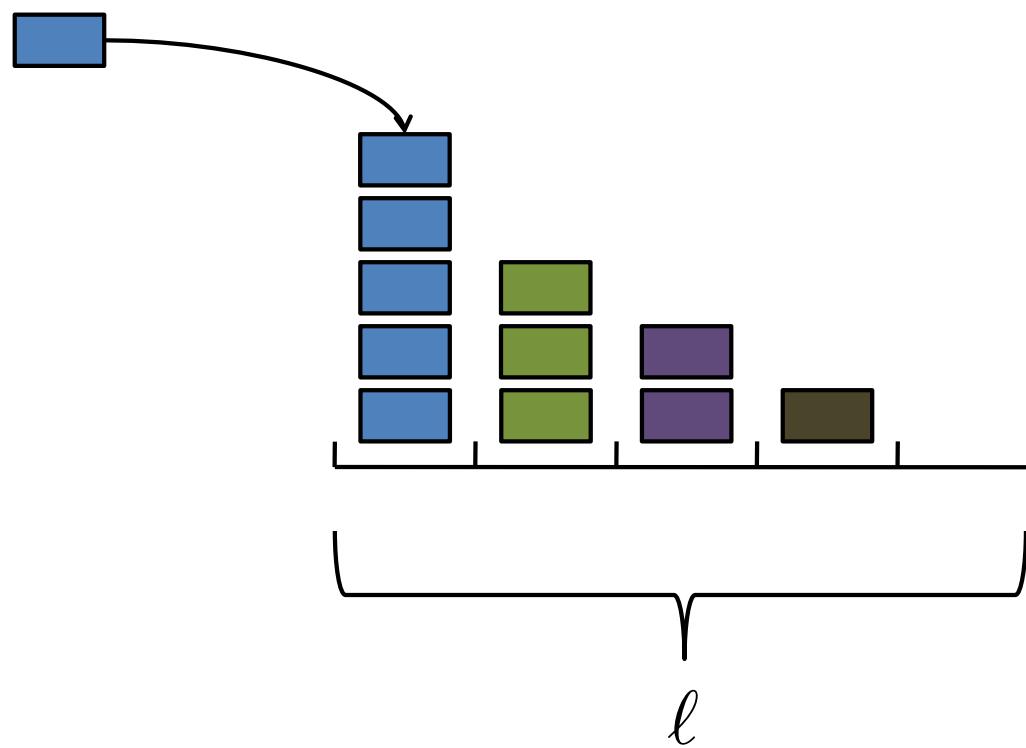
Problem Definition

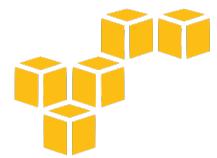
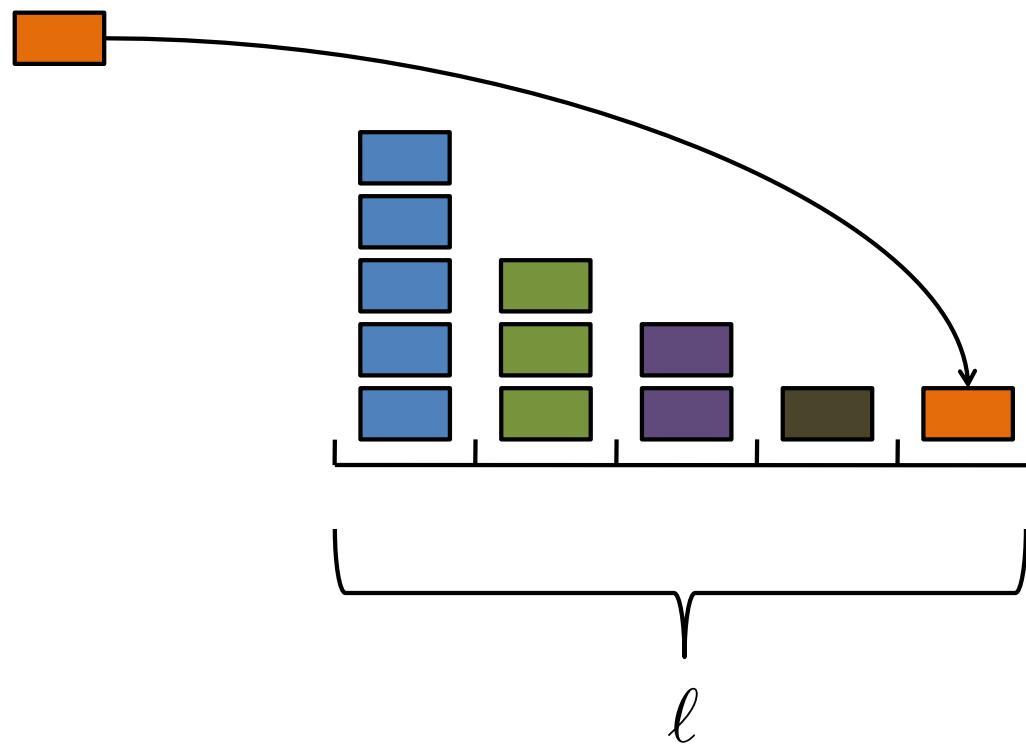


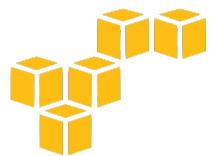
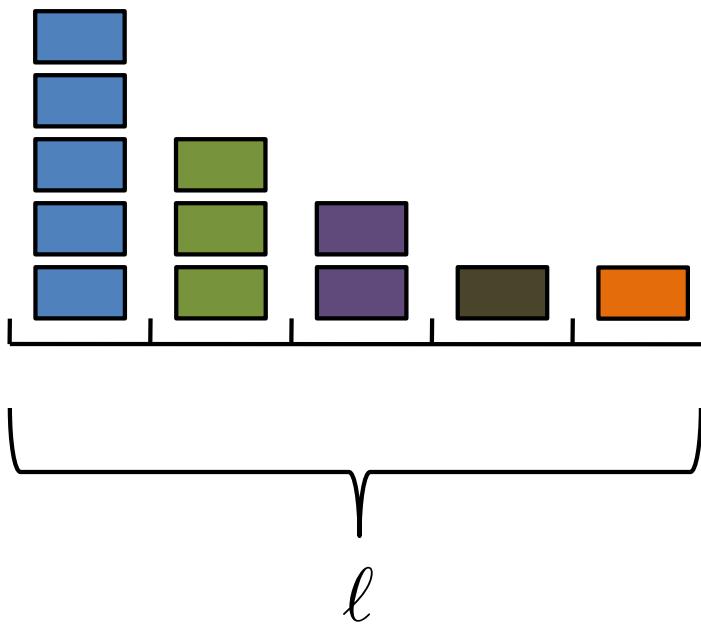
Can we do better than sampling?

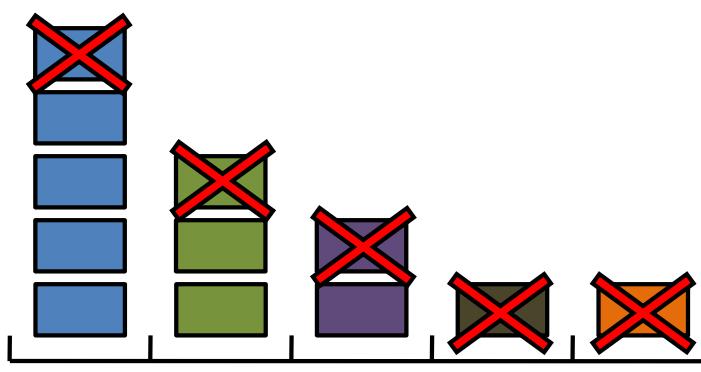






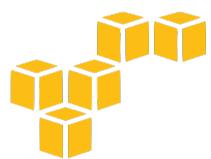


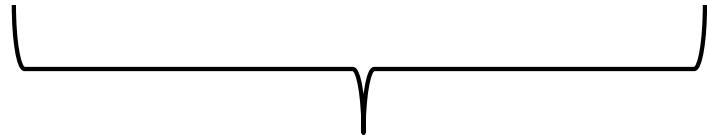
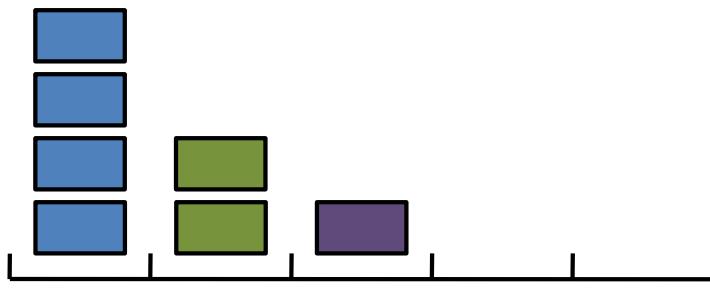




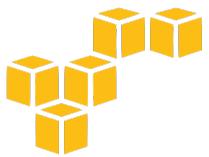
A large curly brace is positioned below the horizontal line, spanning the width of the five bars above it.

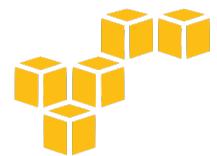
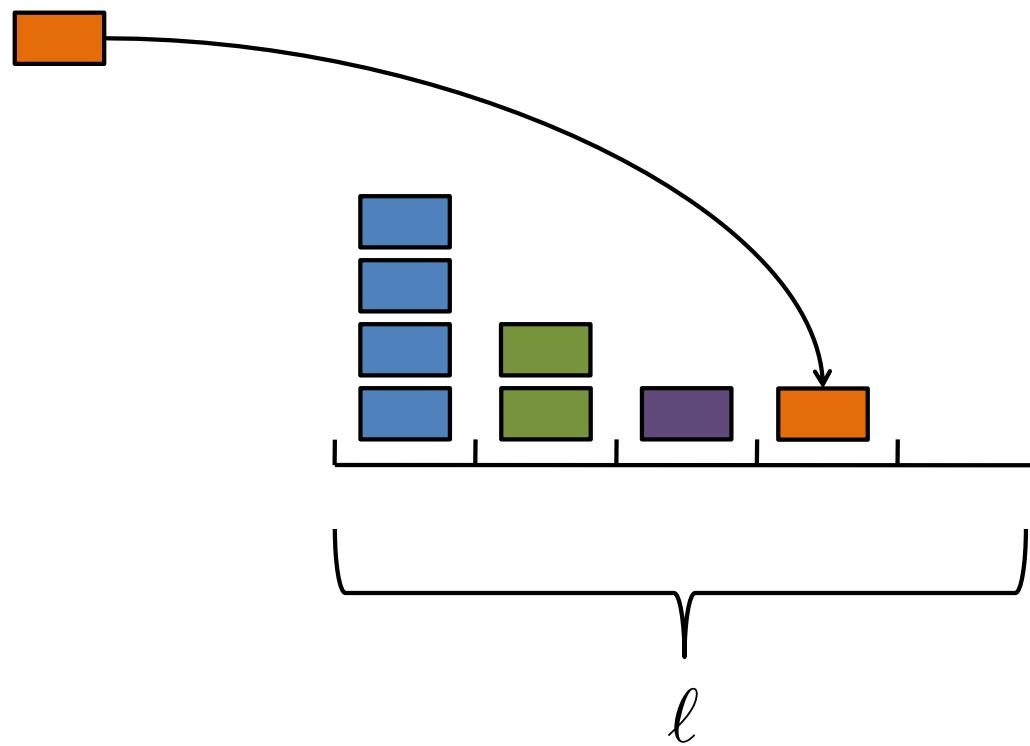
ℓ

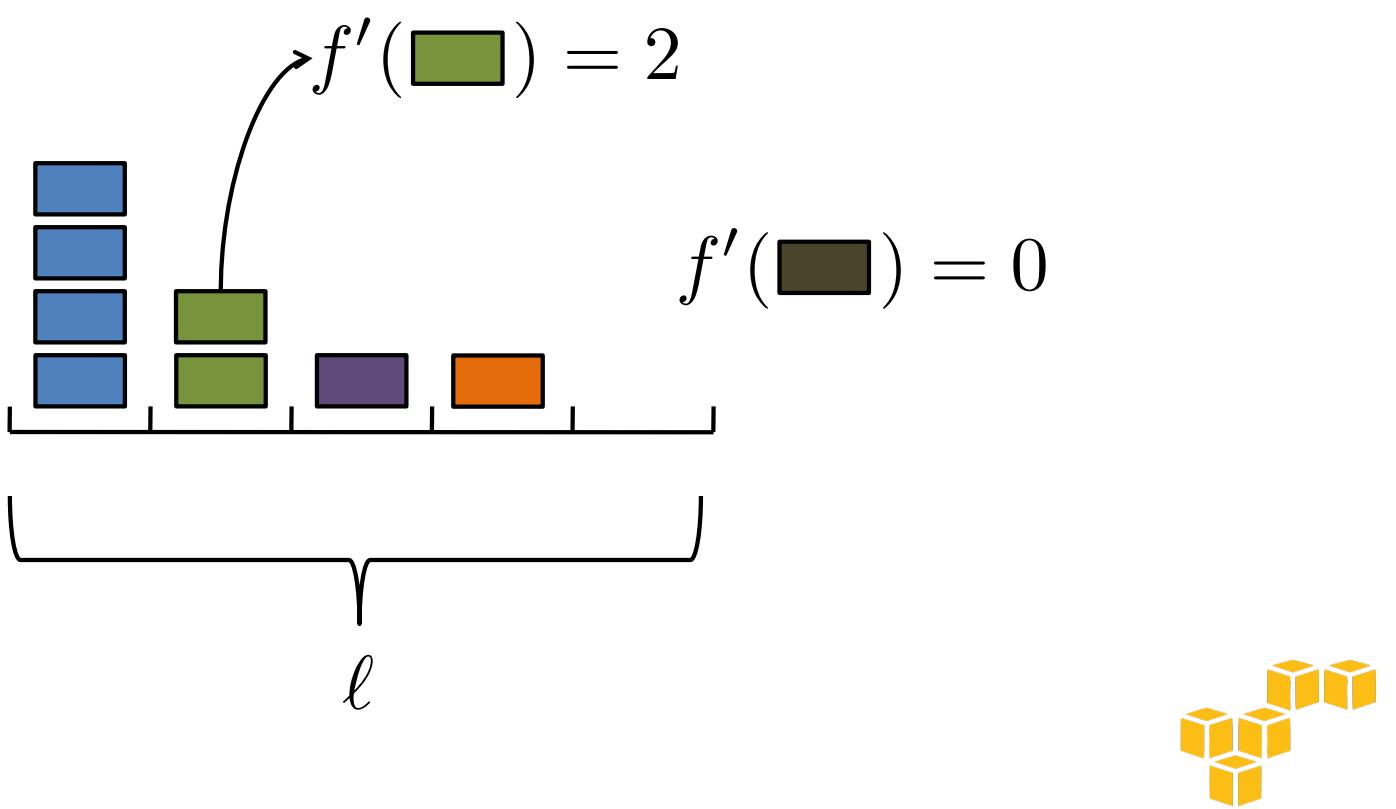




ℓ

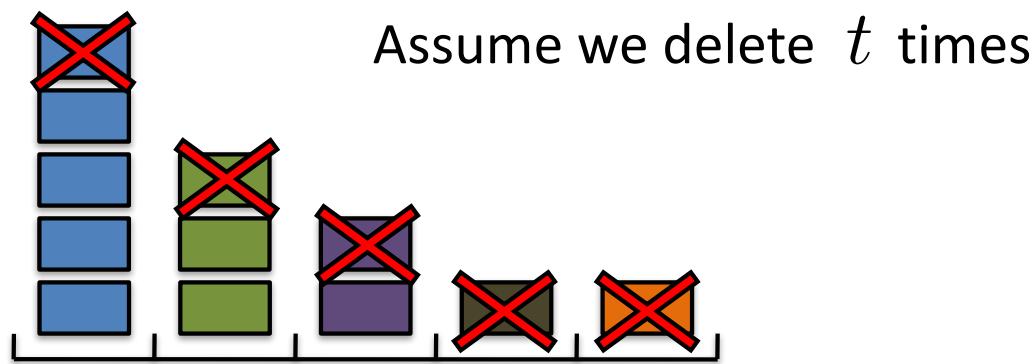






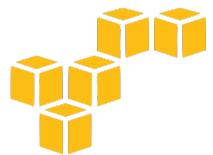
Analysis

First fact: $f'(x) \leq f(x)$



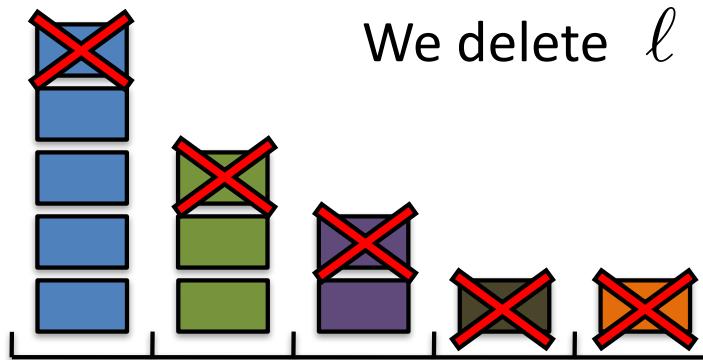
Second fact: $f'(x) \geq f(x) - t$

Therefore: $|f'(x) - f(x)| \leq t$



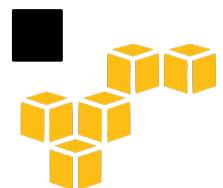
Analysis

Third fact: $t \leq n/\ell$



We get that: $|f'(x) - f(x)| < \varepsilon n$

When: $\ell = 1/\varepsilon$ (much better than sampling!)



Analysis

Items' exact probability $p(x) = f(x)/n$

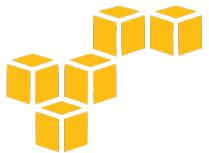
Approximate probability $p'(x) = f'(x)/n$

We get:

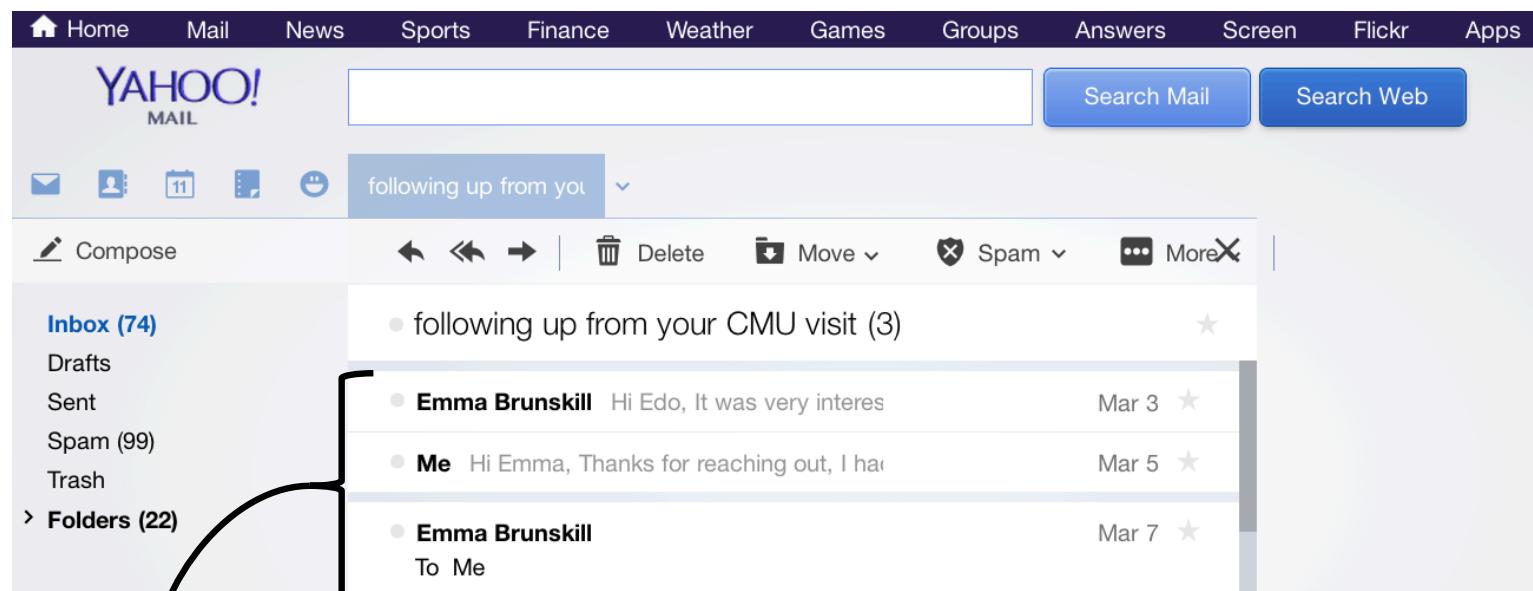
$$|p'(x) - p(x)| \leq 1/\ell$$

If $\ell = 10,000$ we get only a 0.01% error in our estimations.

We would need 10 billion samples to get the same accuracy!



Email threads



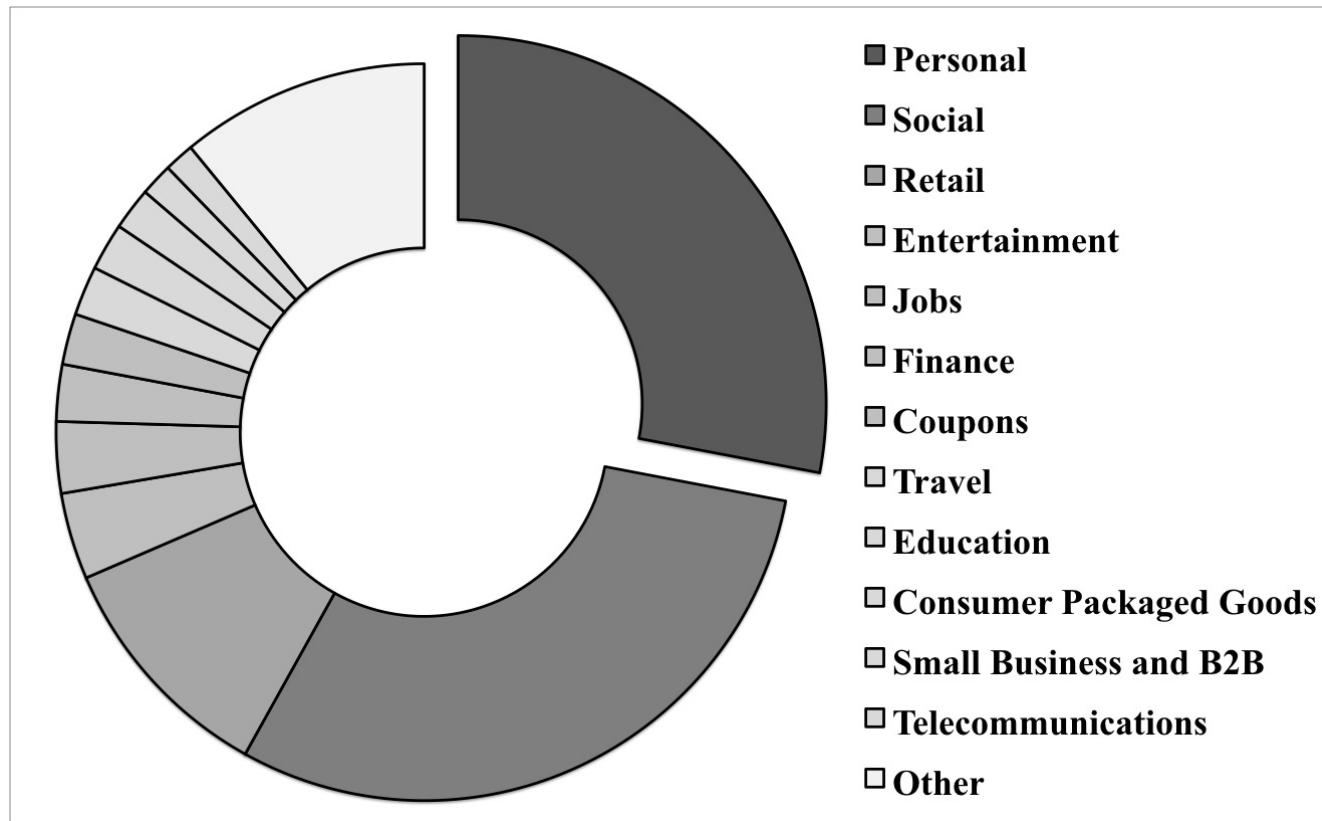
The screenshot shows the Yahoo! Mail inbox interface. The top navigation bar includes links for Home, Mail, News, Sports, Finance, Weather, Games, Groups, Answers, Screen, Flickr, and Apps. The main area features the Yahoo! logo and search bars for Mail and Web. Below the search bar is a toolbar with icons forCompose, Reply, Forward, Delete, Move, Spam, and More. The inbox list is titled "Inbox (74)". It displays three messages from "following up from your CMU visit": one from "Emma Brunskill" on March 3, one from "Me" on March 5, and another from "Emma Brunskill" on March 7. A bracket on the left side of the screen groups these three messages under the heading "Folders (22)". A large black arrow points downwards from this bracket towards the explanatory text below.

Date	From	Subject
Mar 3	Emma Brunskill	Hi Edo, It was very interes
Mar 5	Me	Hi Emma, Thanks for reaching out, I ha
Mar 7	Emma Brunskill	To Me

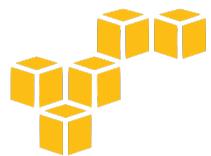
A simple email thread (that's not very hard to do...)



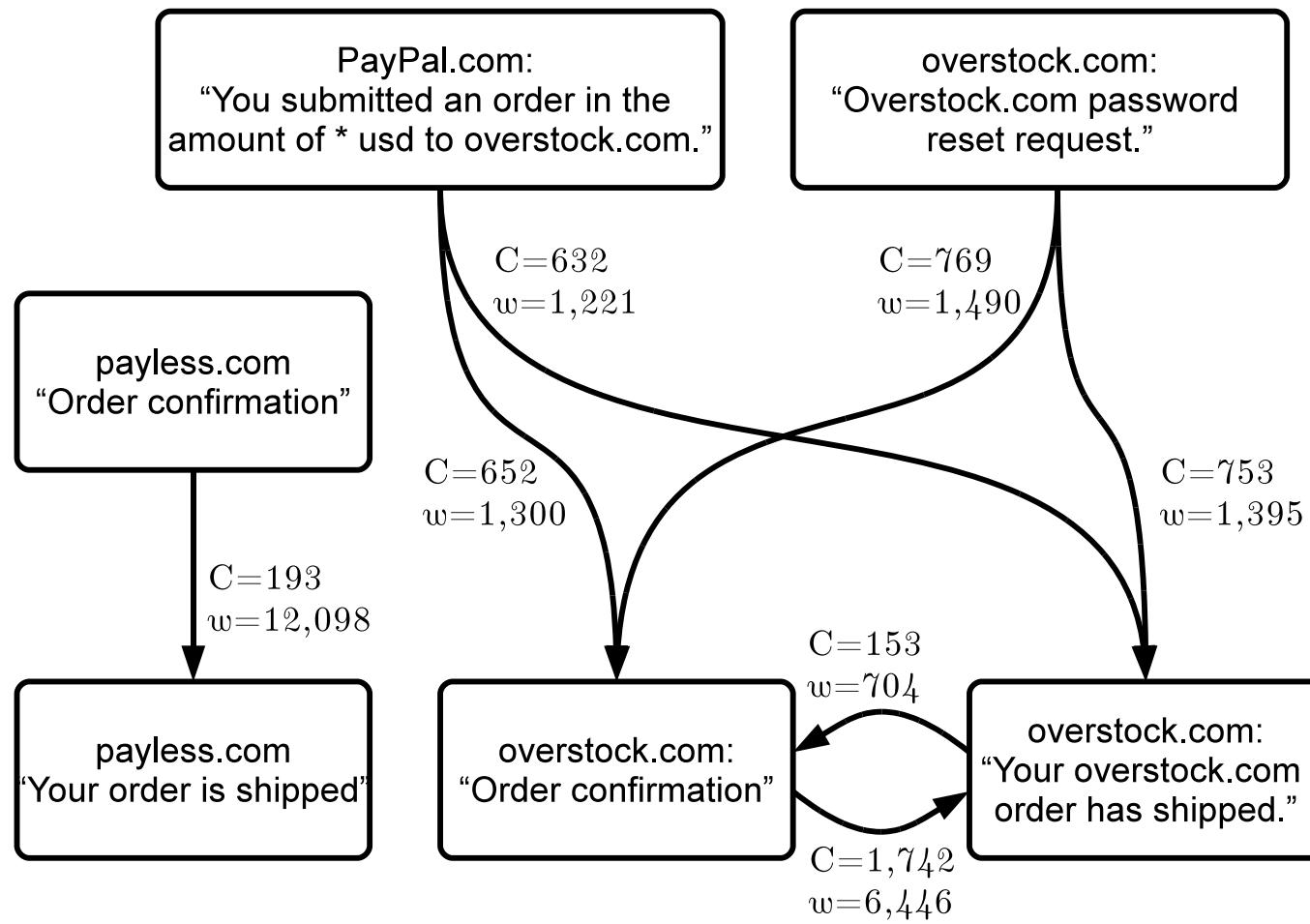
Threading Machine Generated Email



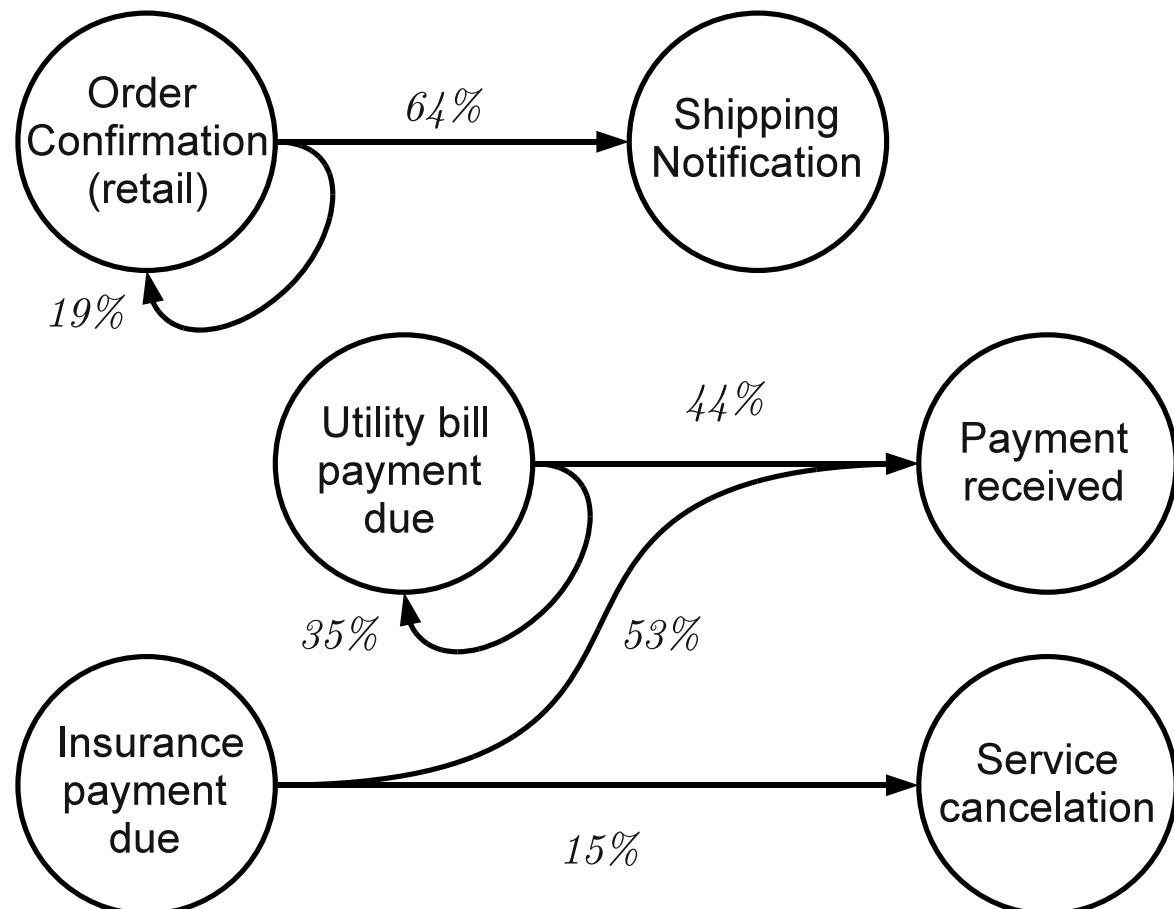
Ailon, Karnin, Maarek, Liberty, Threading Machine Generated Email, WSDM 2013



Threading Machine Generated Email



Threading Machine Generated Email



Streaming quantiles

Manku, Rajagopalan, Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets.

Munro, Paterson. Selection and sorting with limited storage.

Greenwald, Khanna. Space-efficient online computation of quantile summaries.

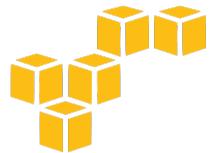
Wang, Luo, Yi, Cormode. Quantiles over data streams: An experimental study.

Greenwald, Khanna. Quantiles and equidepth histograms over streams.

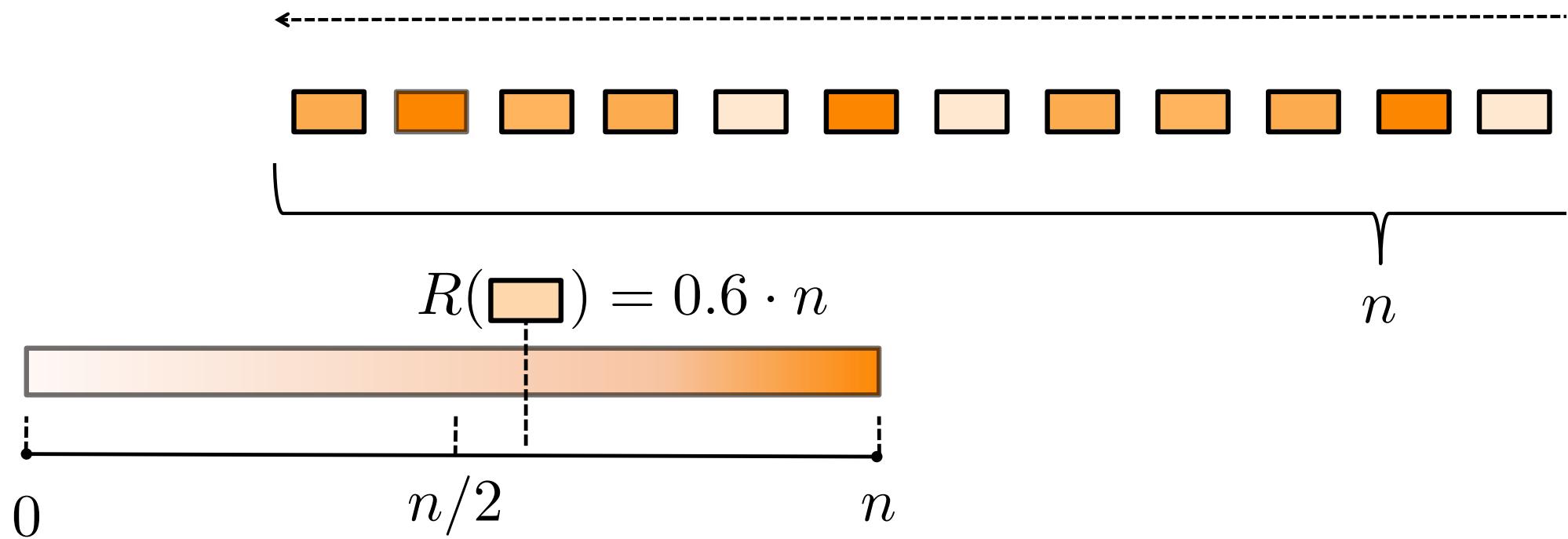
Agarwal, Cormode, Huang, Phillips, Wei, Yi. Mergeable summaries.

Felber, Ostrovsky. A randomized online quantile summary in $O((1/\varepsilon) \log(1/\varepsilon))$ words.

Lang, Karnin, Liberty, Optimal Quantile Approximation in Streams.



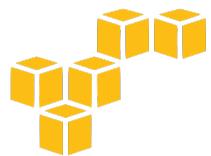
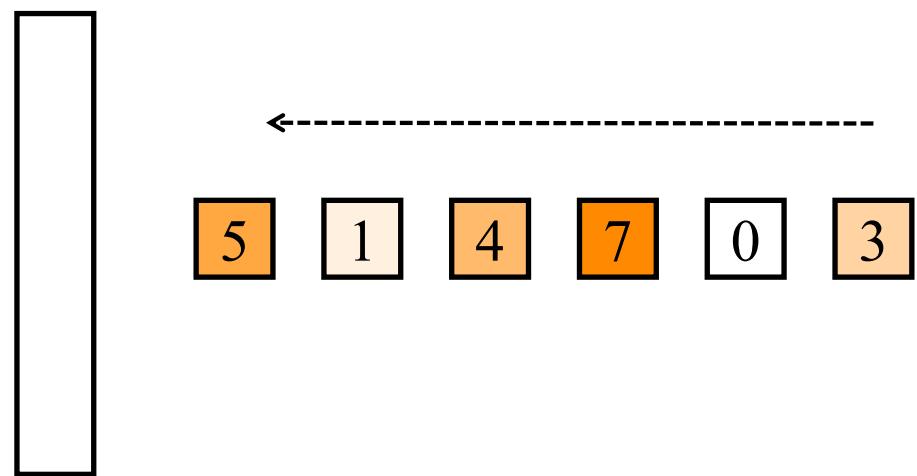
Problem Definition



Sampling $\tilde{O}(1/\varepsilon^2)$ values gives $|R' - R| < \varepsilon n$ can we do better? 

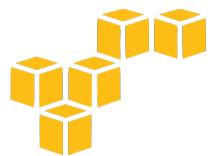
The basic buffer idea

Buffer of size k



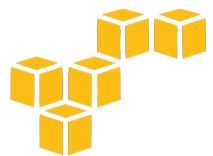
The basic buffer idea

Stores k stream entries



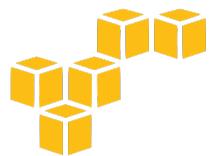
The basic buffer idea

The buffer sorts k stream entries



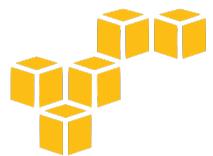
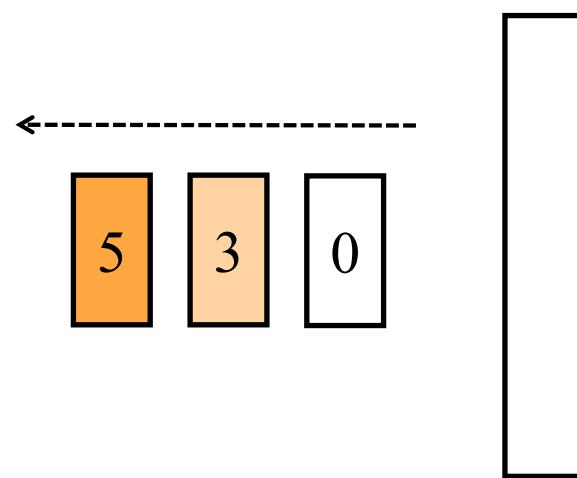
The basic buffer idea

Deletes every other item

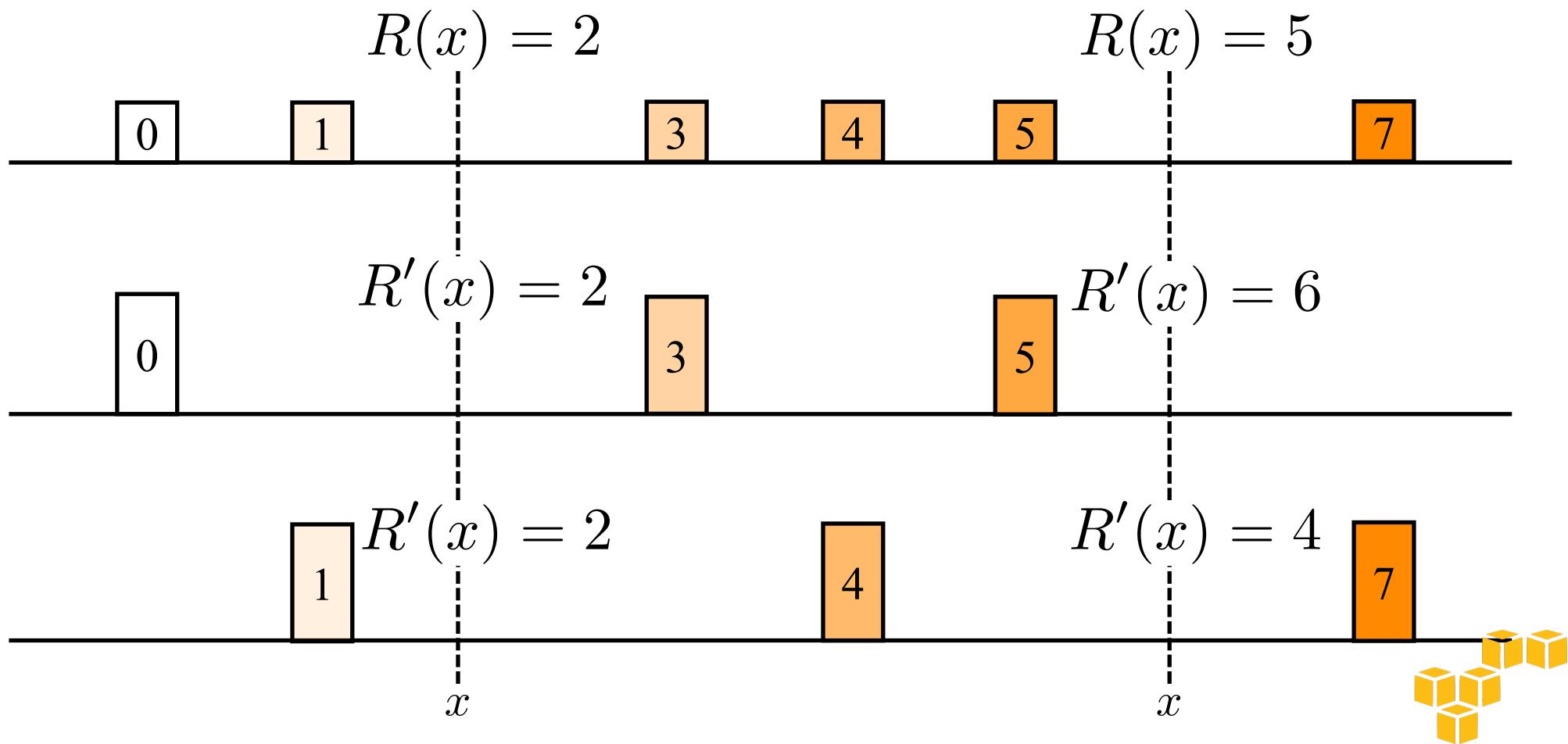


The basic buffer idea

And outputs the rest
with double the weight

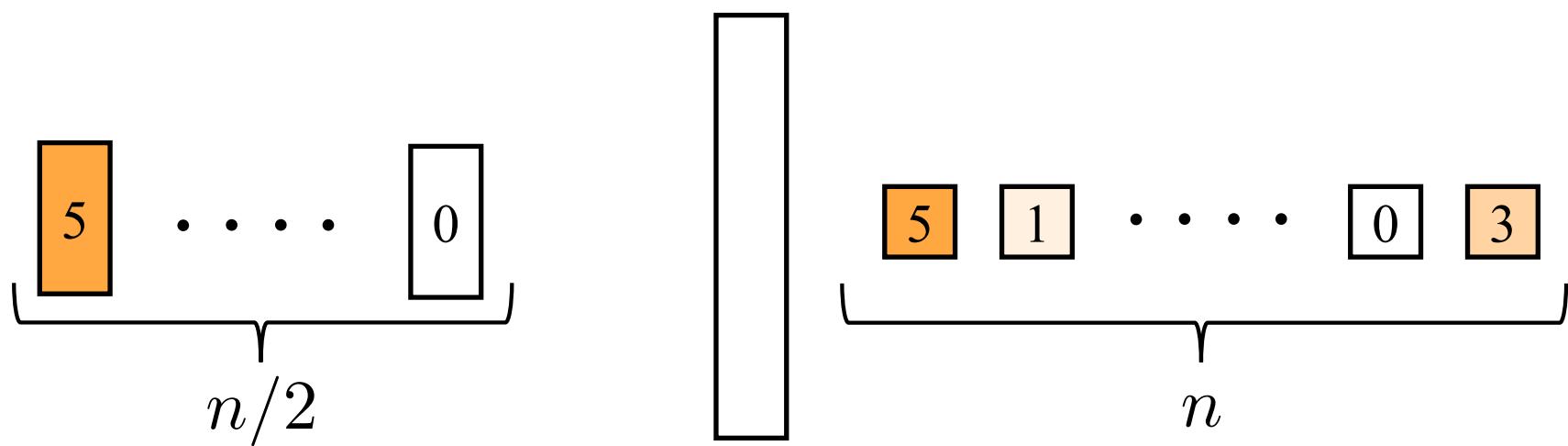


The basic buffer idea

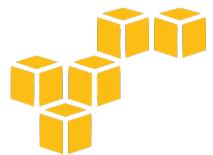


The basic buffer idea

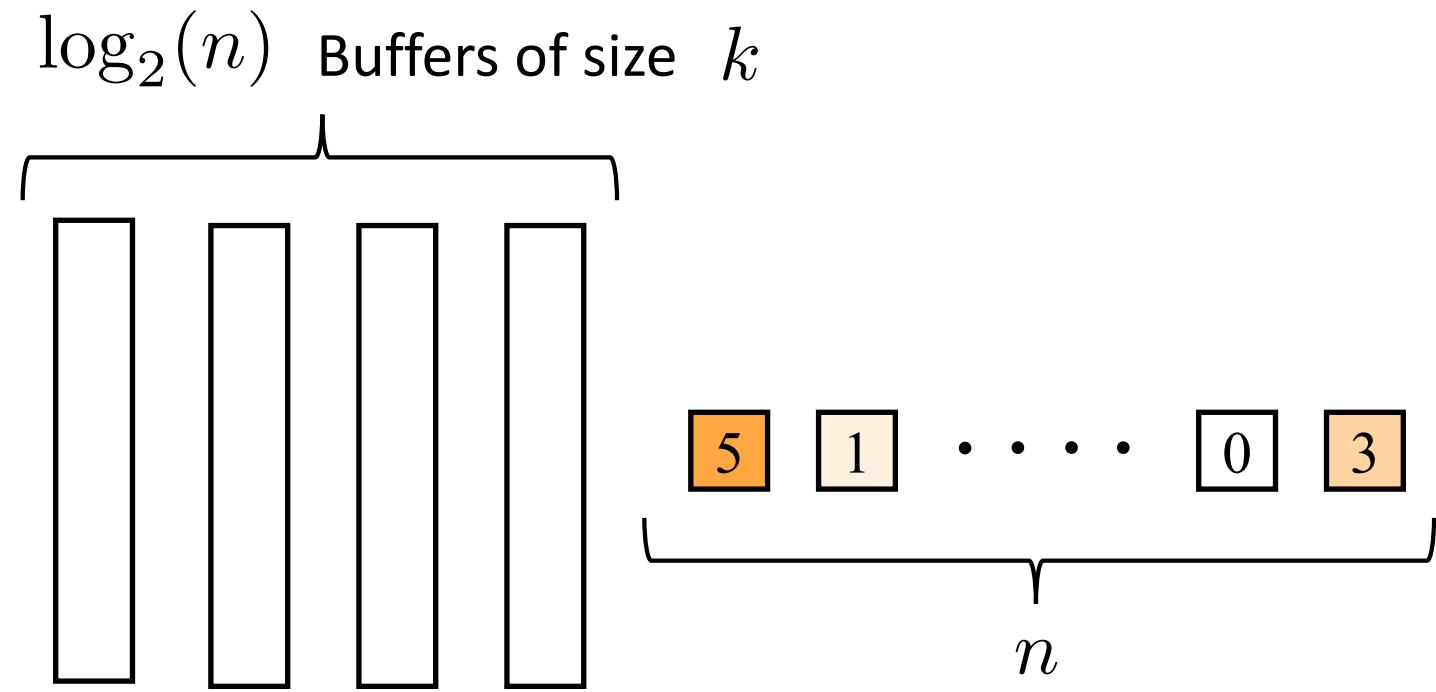
Repeat n/k time until
the end of the stream



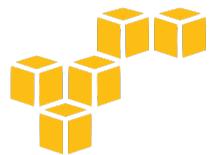
$$|R'(x) - R(x)| < n/k$$



Manku-Rajagopalan-Lindsay (MRL) sketch



$$|R'(x) - R(x)| \leq n \log_2(n)/k$$

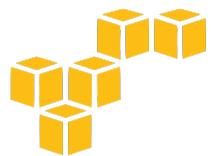


Manku-Rajagopalan-Lindsay (MRL) sketch

If we set $k = \log_2(n)/\varepsilon$

We get $|R'(x) - R(x)| \leq \varepsilon n$

And we maintain only $\log_2^2(n)/\varepsilon$ items from the stream!

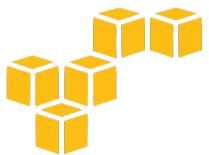


Greenwald-Khanna (GK) sketch

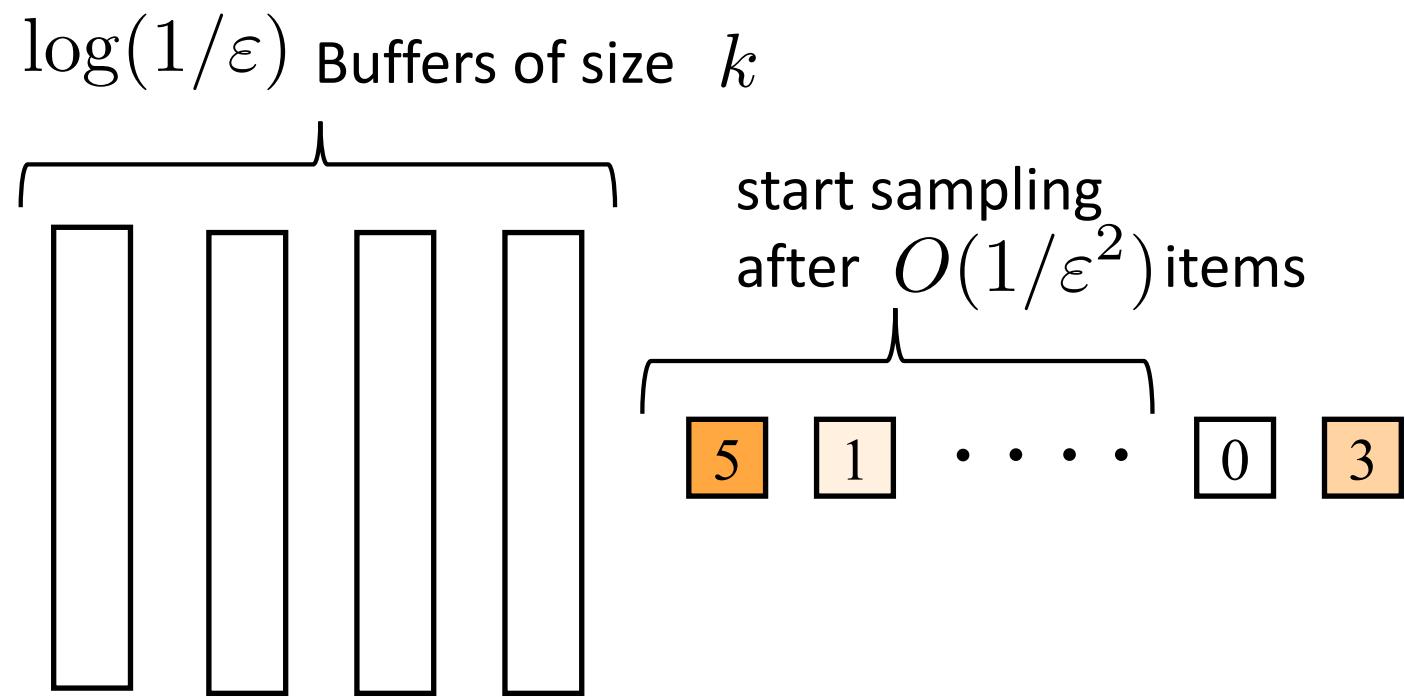
Uses a completely different construction

It gets $|R'(x) - R(x)| \leq \varepsilon n$

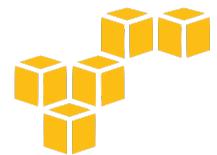
And maintains only $O(\log(n)/\varepsilon)$ items from the stream!



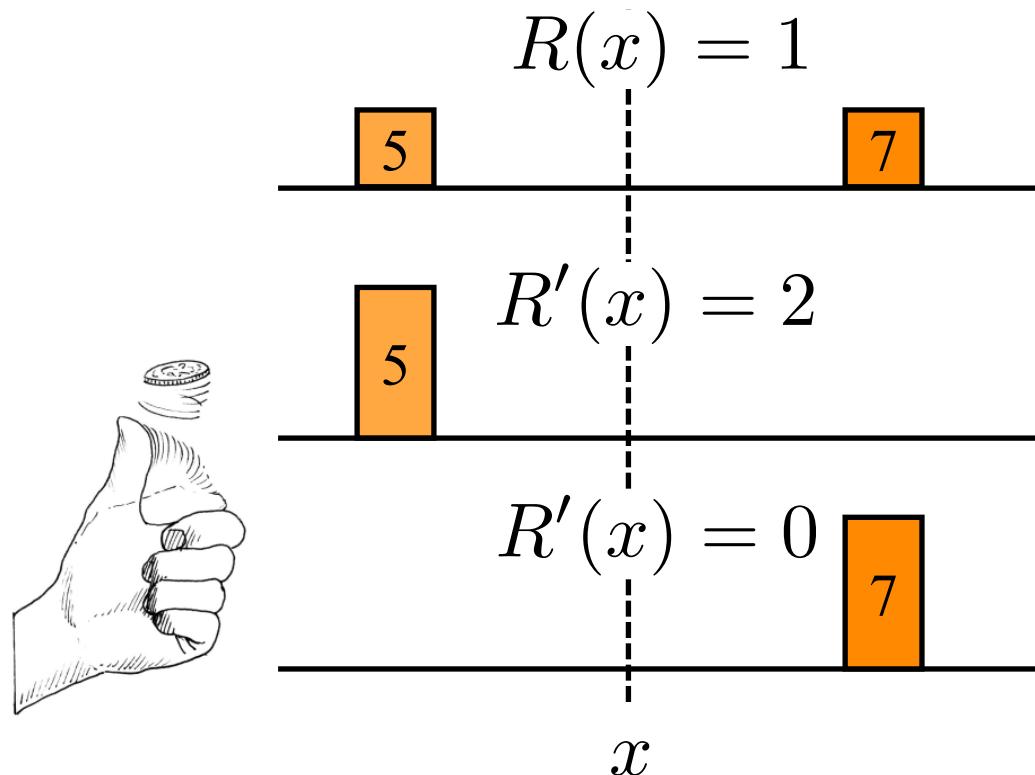
Agarwal, Cormode, Huang, Phillips, Wei, Yi (1)



Reduces space usage to $\log^2(1/\varepsilon)/\varepsilon$ items from the stream.



Agarwal, Cormode, Huang, Phillips, Wei, Yi (2)



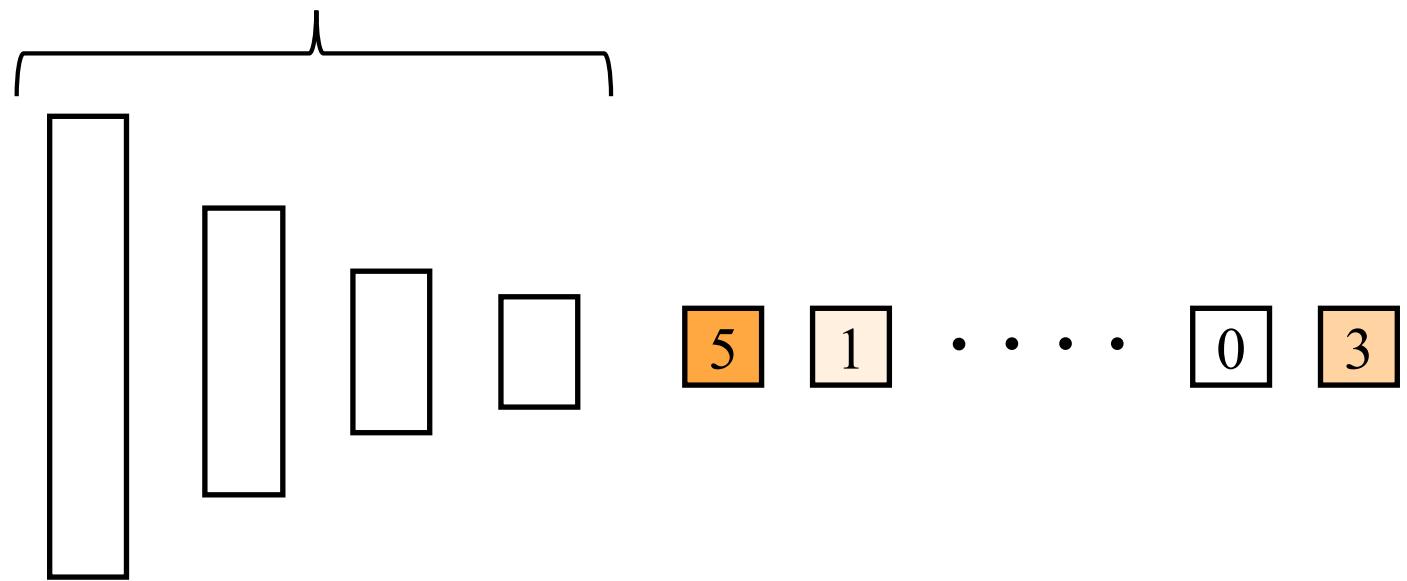
Reduces space usage to $\log^{3/2}(1/\varepsilon)/\varepsilon$ items from the stream.

$R'(x)$ is a random variable now and
 $E[R'(x)] = R(x)$

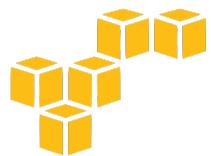


Lang, Karnin, Liberty (1)

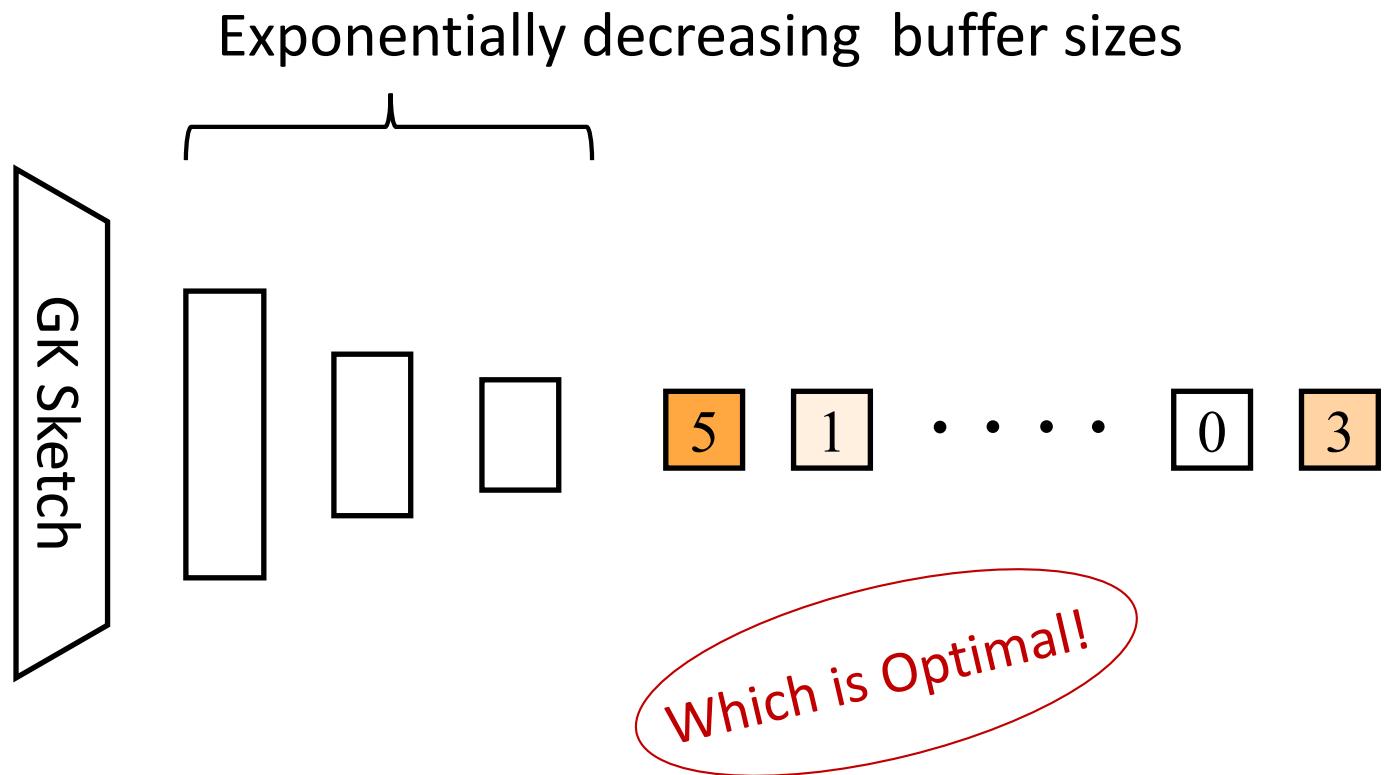
Exponentially shrinking buffers



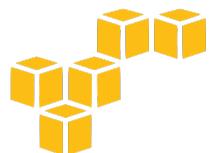
Reduces space usage to $\sqrt{\log(1/\varepsilon)}/\varepsilon$ items from the stream.



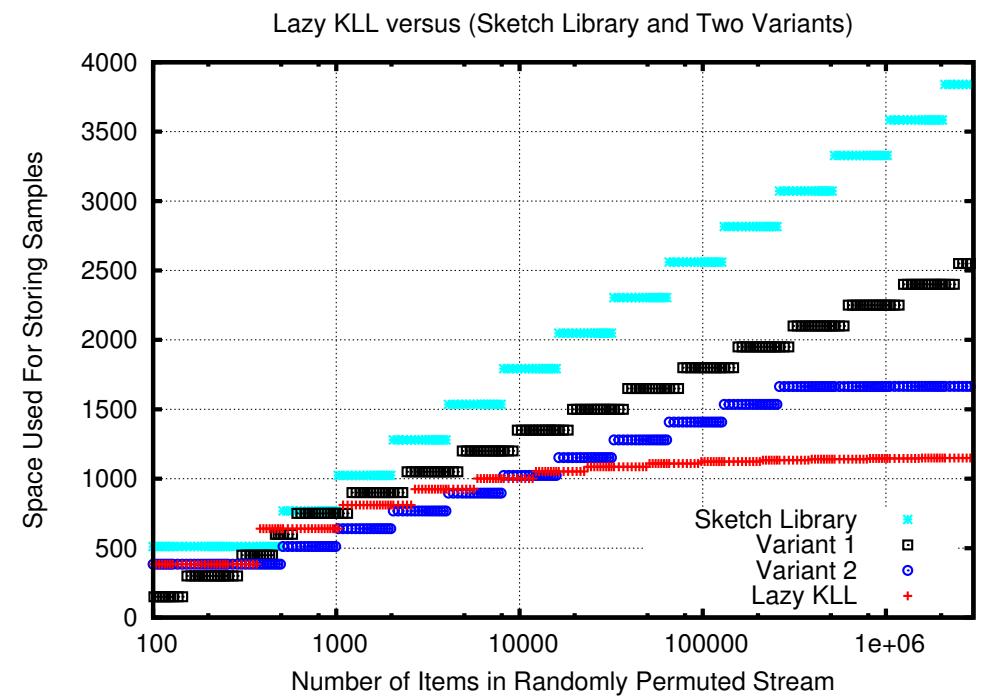
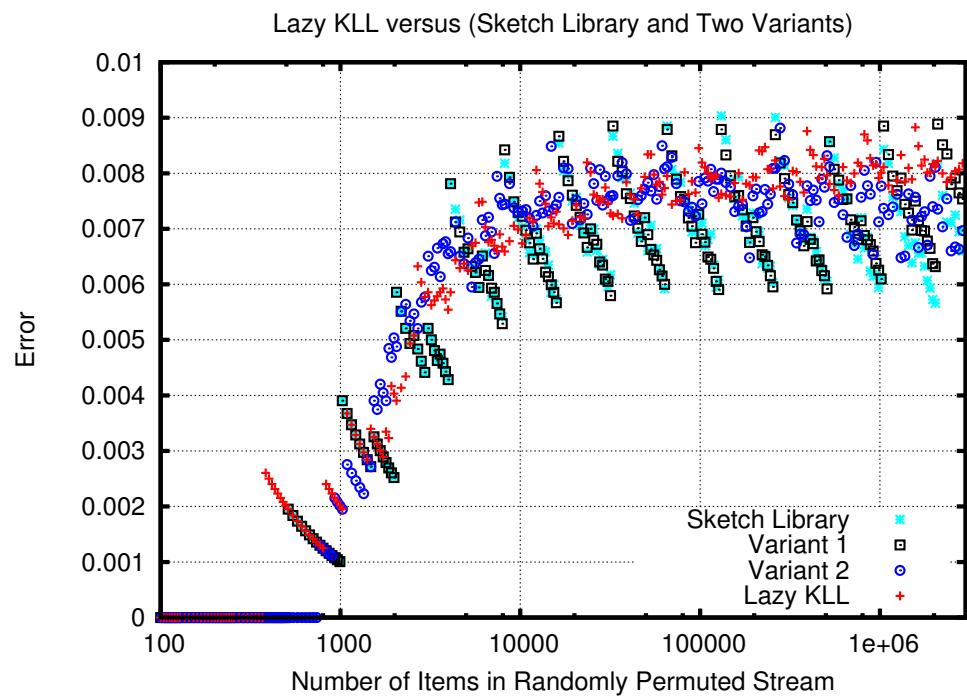
Lang, Karnin, Liberty (2)



Reduces space usage to $\log \log(1/\varepsilon)/\varepsilon$ items from the stream.



Some experimental results



Count Distinct (Demo Only)

 GitHub, Inc. [US] <https://github.com/datasketches>



sketches-core

Core Sketch Library.

 Java  415  119 Updated a day ago

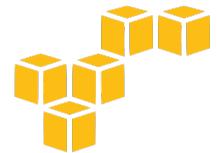


YAHOO!

 druid

 splice
MACHINE

 amazon
web services™

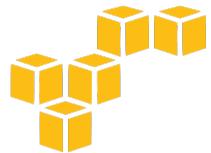


Assume you need to estimate the number of **unique** numbers in a file

```
>>head data.csv
```

```
0  
1  
0  
3  
0  
2  
3  
7  
3  
2
```

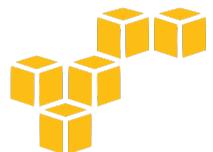
In this one, row i tasks a value from $[0,i]$ uniformly at random.



Some stats: there are 10,000,000 such numbers in this ~76Mb file.

```
>>time wc -lc data.csv  
10000000 76046666 data.csv  
  
real 0m0.101s  
user 0m0.072s  
sys 0m0.021s
```

Reading the file take ~1/10 seconds. We don't foresee IO being an issue.



To count the number of distinct items you might try this:

```
>>sort data.csv | uniq | wc -l
```

However, it is faster to have “uniqify” while sorting.

```
>>sort data.csv -u | wc -l
```

```
>>time sort data.csv -u | wc -l
```

5001233

real 2m37.071s

user 2m36.587s

sys 0m0.376s

Parent Process: [bash_\(11203\)](#) User: libertye (2045342942)

Process Group: cat (12535)

% CPU: 99.50

Recent hangs: 0

Memory

Statistics

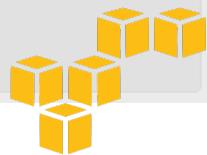
Open Files and Ports

Real Memory Size: 49.9 MB

Virtual Memory Size: 2.38 GB

Shared Memory Size: 224 KB

Private Memory Size: 49.3 MB



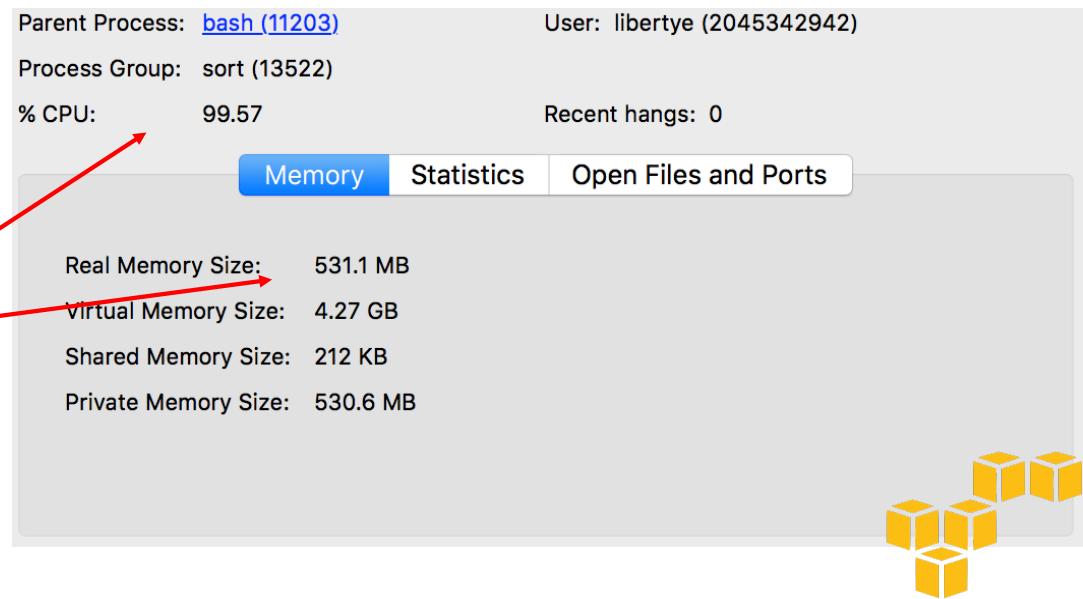
Still, most of the time is spent on comparing strings....

```
>>sort data.csv -u -n -S 100% | wc -l
```

This is much better!

```
>>time sort data.csv -u -n | wc -l  
5001233
```

real 0m11.809s ←
user 0m11.587s
sys 0m0.228s



This is the way to do this with the sketching library

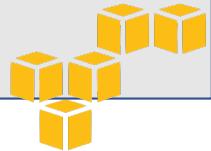
```
>>sketch uniq data.csv
```

```
>>time sketch uniq data.csv  
Estimate      : 4974249  
Upper Bound : 5116569  
Lower Bound : 4835874
```

```
real 0m1.527s ←  
user 0m1.506s  
sys 0m0.152s
```

Too fast to use the system monitor UI...

It uses ~ 32k of memory!



Thank you!

