2023 ESA Test-of-Time Award

LogLog Counting of Large Cardinalities

Marianne Durand and Philippe Flajolet

presented by

Robert Sedgewick
Princeton University

Passing the test of time

Why does log-log counting pass the test of time?

It solves a **fundamental problem** in *data science.*

It is a **simple, elegant** and **efficient** solution.

It is a poster child for **analytic combinatorics.**

It is a poster child for **algorithm science**.

It is **broadly applicable** and **widely used.**

LogLog Counting of Large Cardinalities

- A fundamental problem in data science
- A simple, elegant and efficient solution
- A poster child for algorithm science
- A poster child for analytic combinatorics
- Widely applicable and still relevant

Cardinality counting

Q. In a given stream of data values, how many *different* values are present?

Reference application. How many unique visitors in a web log ?

log.07.f3.txt

```
117.222.48.163
pool-71-104-94-246.lsanca.dsl-w.verizon.net
1.23.193.58
188.134.45.71
1.23.193.58
gsearch.CS.Princeton.EDU
pool-71-104-94-246.lsanca.dsl-w.verizon.net
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
CPE-121-218-151-176.lnse3.cht.bigpond.net.au
117.211.88.36
```

6 million strings

UNIX (1970s-present)

```
% sort -u log.07.f3.txt | wc -l
1112365
```

"unique"

SQL (1970s-present)

```
SELECT
DATE_TRUNC('day',event_time),
COUNT(DISTINCT user_id),
COUNT(DISTINCT url)
FROM weblog
```

State of the art in the wild for decades. Sort, then count.

"Optimal" solution. Use a hash table. ← order of magnitude faster than sort-based solution

Q. I can't use a hash table. The stream is ***much too big*** to fit all values in memory. Now what?

Cardinality *estimation*

A. Look for a way to *estimate* the value of ***N***, the number of distinct values in the stream.

Practical cardinality estimation problem

- Make *one pass* through the stream.
- Use *as few operations per value* as possible
- Use *as little memory* as possible.
- Produce *as accurate an estimate* as possible.

typical applications where exact count is not really necessary

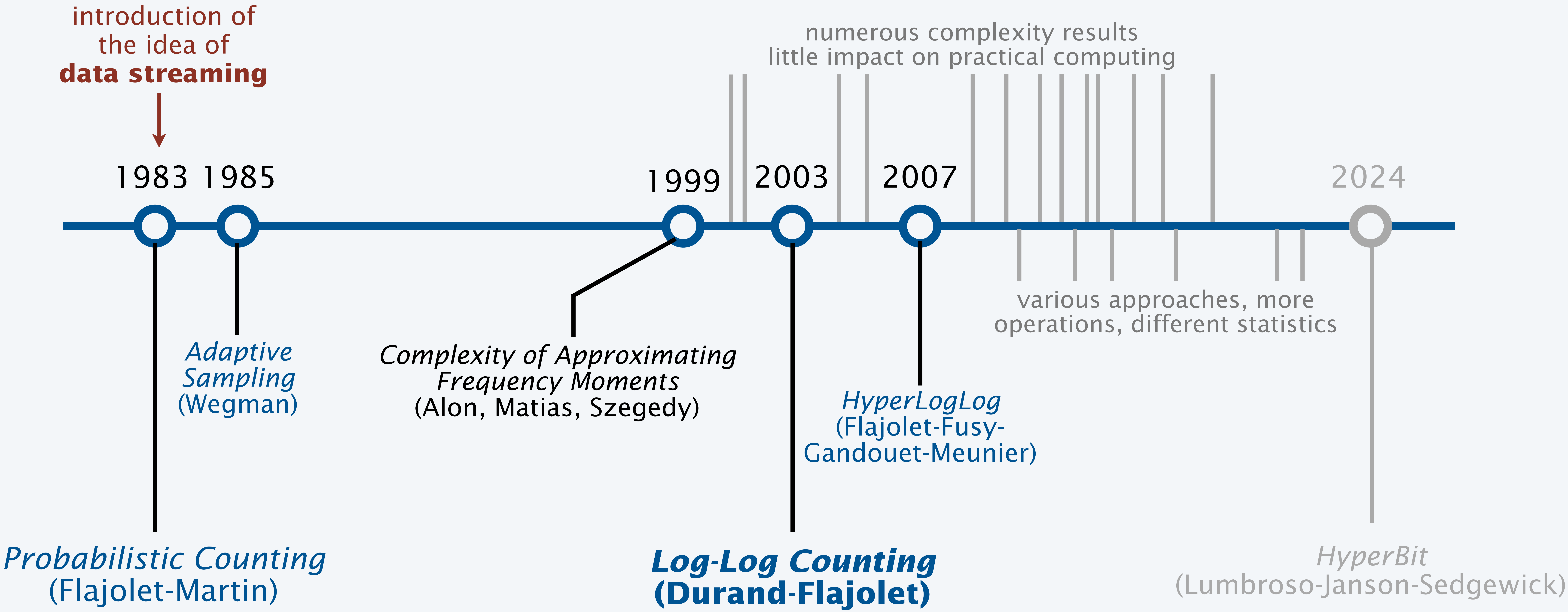
- How many unique visitors to my website?
- How many different cars passed here this year?
- How many different IP addresses hit this node?
- How many different values for a database join?

To fix ideas on scope (202x): Think of *billions* of streams each having *trillions* of values.

Q. How much memory is needed to estimate ***N*** to within, say, 10% accuracy?

A. Much less than you might think!

Timeline of milestones in cardinality estimation



For details, see "*The Story of HyperLogLog: How Flajolet Processed Streams with Coin Flips*" J. Lumbroso, 2013.

LogLog Counting of Large Cardinalities

- A fundamental problem in data science
- A simple, elegant and efficient solution
- A poster child for algorithm science
- A poster child for analytic combinatorics
- Widely applicable and still relevant

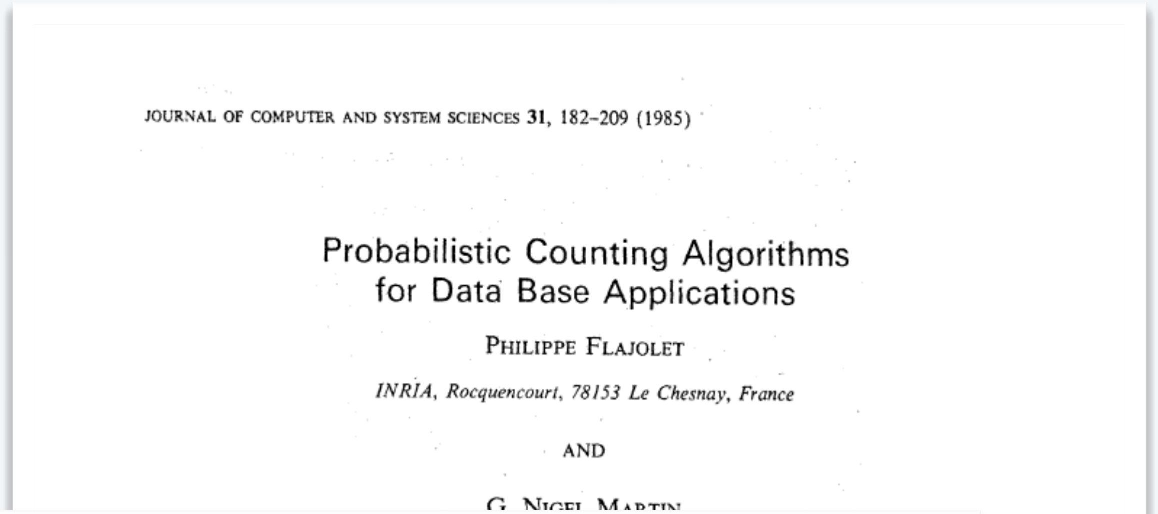
Simple, elegant, and efficient solutions

1983

Flajolet and Martin
Probabilistic Counting Algorithms for Data Base Applications

2003

Durand and Flajolet
LogLog Counting of Large Cardinalities



JOURNAL OF COMPUTER AND SYSTEM SCIENCES 31, 182–209 (1985)

Probabilistic Counting Algorithms
for Data Base Applications

PHILIPPE FLAJOLET

INRIA, Rocquencourt, 78153 Le Chesnay, France

AND

G. NIGEL MARTIN

LOGLOG COUNTING OF LARGE CARDINALITIES

MARIANNE DURAND AND PHILIPPE FLAJOLET

ABSTRACT. Using an auxiliary memory smaller than the size of this abstract, the LOGLOG algorithm makes it possible to estimate in a single pass and within a few percents the number of different words in the whole of Shakespeare's works. In general the LOGLOG algorithm makes use of m "small bytes" of auxiliary memory in order to estimate in a single pass the number of distinct elements (the "*cardinality*") in a file, and it does so with an accuracy that is of the order of $1/\sqrt{m}$. The "small bytes" to be used in order to count cardinalities till $N_{\max}$ comprise about $\log\log N_{\max}$ bits, so that cardinalities well in the range of billions can be determined using one or two kilobytes of memory only. The basic version of the LOGLOG algorithm is validated by a complete analysis. An optimized version, super–LOGLOG, is also engineered and tested on real-life data. The algorithm parallelizes optimally.

1. INTRODUCTION

The problem addressed in this note is that of determining the number of *distinct* elements, also called the *cardinality*, of a large file. This problem arises in several areas of data-mining, database query optimization, and the analysis of traffic in routers. In such contexts, the data may be either too large to fit at once in core memory or even too massive to be stored, being a huge continuous flow of data packets. For instance, Estan *et al.* [3] report traces of packet headers, produced at a rate of 0.5GB per hour of compressed data (!), which were collected while trying to trace a "worm" (Code Red, August 1 to 12, 2001), and on which it was necessary to count the number of *distinct* sources passing through the link. We propose here the LOGLOG algorithm that estimates cardinalities using only a very small amount of auxiliary memory, namely m memory units, where a memory unit, a "*small byte*", comprises close to $\log\log N_{\max}$ bits, with $N_{\max}$ an *a priori* upperbound on cardinalities. The estimate is (in the sense of mean values) asymptotically *unbiased*; the relative accuracy of the estimate (measured by a standard deviation) is close to $1.05/\sqrt{m}$ for our best version of the algorithm, Super–LOGLOG. For instance, estimating cardinalities till $N_{\max} = 2^{27}$ (a hundred million different records) can be achieved with $m = 2048$ memory units of 5 bits each, which corresponds to 1.28 kilobytes of auxiliary storage in total, the error observed being typically less than 2.5%. Since the algorithm operates incrementally and in a single pass it can be applied to data flows for which it provides on-line estimates available at any given time. Advantage can be taken of the low memory consumption in order to gather *simultaneously* a *very large number of statistics* on huge heterogeneous data sets. The LOGLOG algorithm can also be *fully distributed* or *parallelized*, with *optimum* speed-up and *minimal* interprocess communication. Finally, an embedded hardware design would involve strictly minimal resources.

Motivations. A traditional application of cardinality estimates is database query optimization. There, a complex query typically involves a variety of set-theoretic operations as well as projections, joints, and so on. In this context, knowing "for free" cardinalities of associated sets provides a valuable guide for selecting an efficient processing strategy best suited to the data at hand. Even a

Date: April 1, 2003. Submitted to the *European Symposium on Algorithms*, ESA'2003.

1

Key steps

- **Hash** each item so as to work with "random" values.
- Develop a **sketch** that enables cardinality estimation.
- **Split** stream into ***M*** substreams and average their estimates.
- Precisely **analyze** the bias.

Flajolet-Martin (probabilistic counting) uses ***M*** ***64-bit*** sketches.

Durand-Flajolet (loglog counting) uses M 6-bit sketches.

21st century values

both deserve test-of-time awards

First step: Hash the values

Transform value to a “random” computer word.

- Compute a *hash function* that transforms data value into a 32- or 64-bit value.
- Cardinality count is unaffected (with high probability).
- Built-in capability in modern systems.
- *Allows use of fast machine-code operations.*

20th century: use 32 bits (millions of values)
21st century: use 64 bits (quintillions of values)

State-of-the-art-"Mersenne twister" uses only a few machine-code instruictions.

Bottom line: Do cardinality estimation on streams of (binary) integers, not arbitrary value types.

```
01111000100111110111000111001000
01111000100111110111000111001000
01110101010110110000000011011010
00110100010001111100010100111010
00010000111001101000111010010011
00001001011011100000010010010111
00001001011011100000010010010111
00111000101001001011010101001100
00111000101001001011010101001100
01101001001000011100110100110011
00001000011101100110110010100101
```

“Random” *except* for the fact that some values are equal.

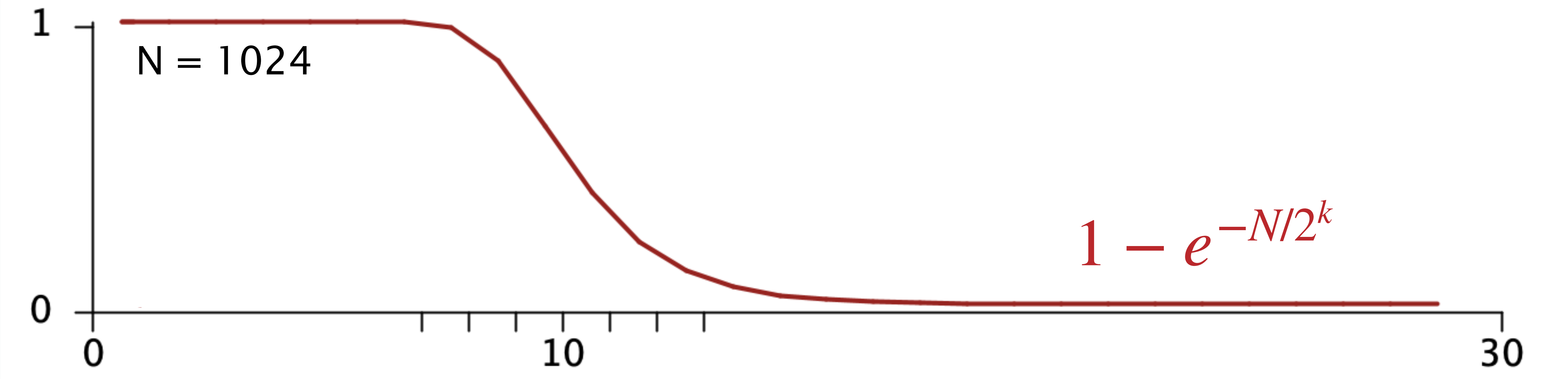
Second step: Focus on the leading 0s

Let **X** be the max number of leading 0s in a random stream of random distinct binary values.

Pr { no value has k leading 0s } $= \left(1 - \frac{1}{2^k}\right)^N \sim e^{-N/2^k}$ = Pr { **X** <= k }

Pr { **X** > k } $\sim 1 - e^{-N/2^k}$ ← ~1 when k is small, ~0 when k is large

$$\mathbf{E(X)} \sim \sum_{k\geq 0} \left(1 - e^{-N/2^k}\right)$$



$\sum_{k\geq 0} (1 - e^{-N/2^k})$ = 1+1+1+1+1+1+1+1+1+1+1 + . . . + 0+0+0+0+0+0+0+0+0+...

~lg N terms are ~1 | a few are not close to 0 or 1 | the rest are all ~0

111100111111110010...
111100010100111010...
011100110100110011...
011100110100110011...
011100110100110011...
011000011101001101...
011100110100110011 ..
110000000011011010...
011100110100110011...
011100110100110011...
001001110010100000...
111100010100111010...
111101010110110001...
000111000111001000...
000111000111001000...
110000000011011010...
111100010100111010...
011000111010010011...
100000010010010111...
100000010010010111...
001011010101001100...

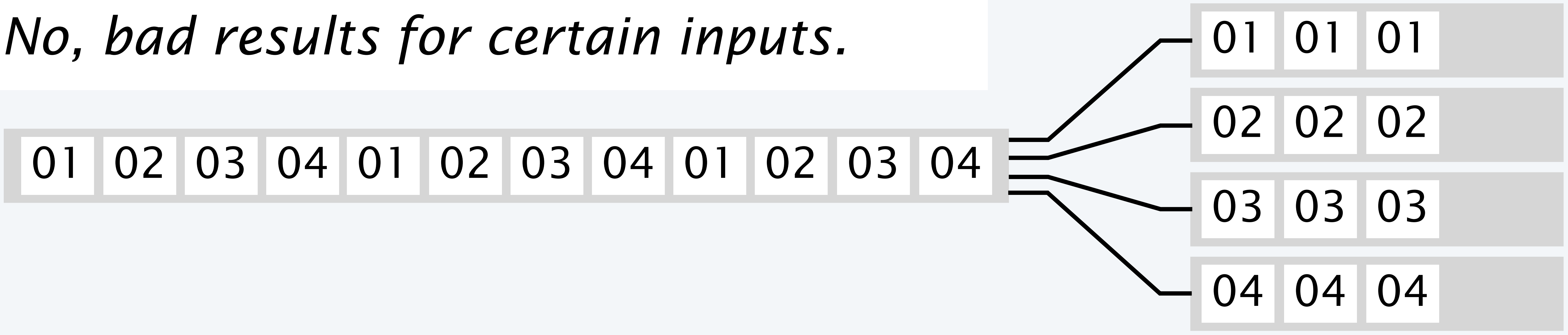
Takeaway. **E(X)** is slightly larger than lg N

Third step: split the stream into substreams and average results

Goal: Perform M independent experiments and average results.

Alternative 1: M independent hash functions? *No, too expensive.*

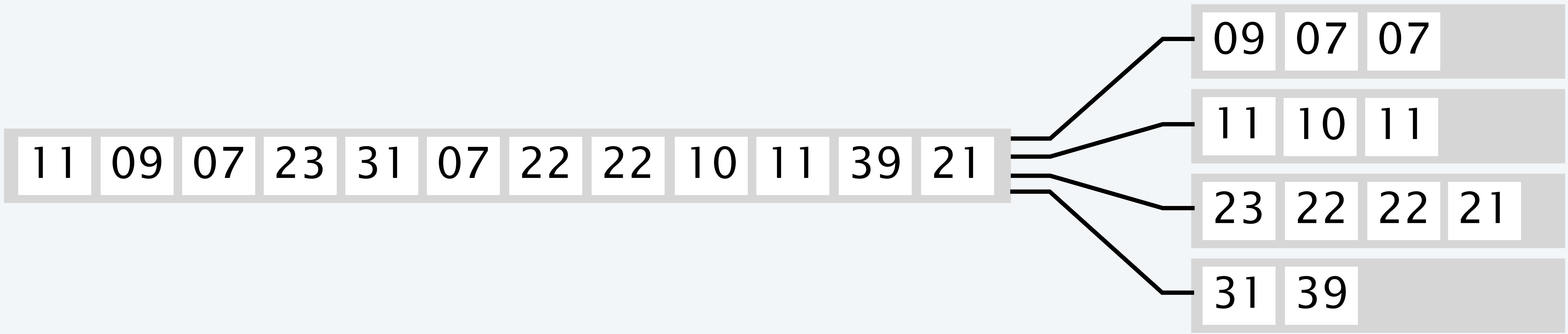
Alternative 2: M-way alternation? *No, bad results for certain inputs.*



Alternative 3: **Stochastic averaging**

- Use second hash to divide stream into M independent streams
- Compute max number of leading 0 bits in each stream,
- Average these values.

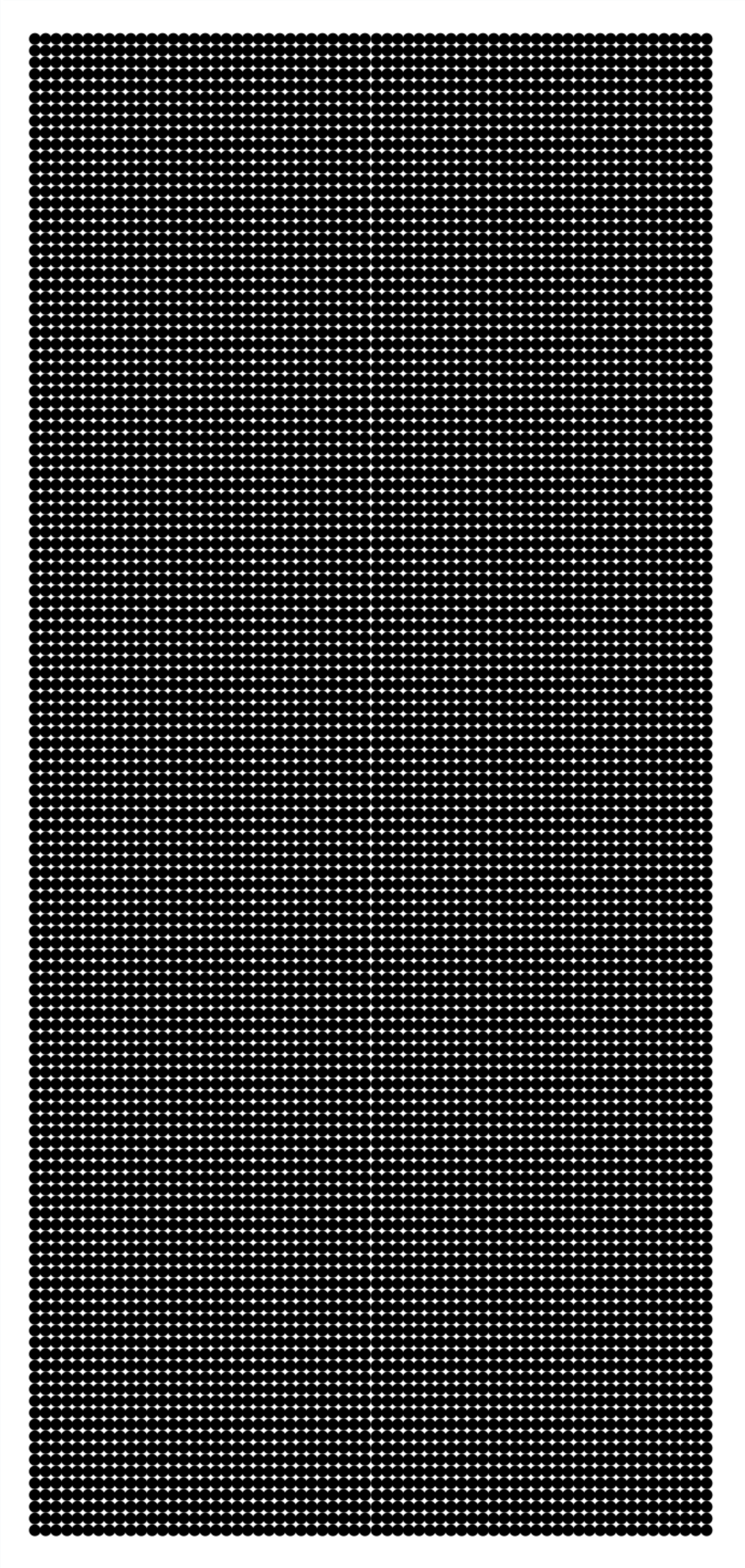
key point: equal values all go to the same stream



Memory use for cardinality estimation algorithms with M-way stochastic averaging

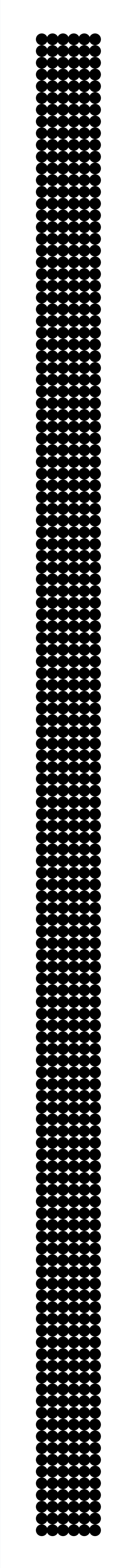
Probabilistic Counting (1983)

M 64-bit words



LogLog Counting (2003)

***M* 6-bit bytes**



Cardinality N is less than 2^{64}

lg $N < 64$ so 64-bit hash values suffice

PC uses a 64-bit sketch

lg lg $N < 6$ so 6-bit counters suffice

Pictured: $M = 128$

LogLog counting Java implementation: simple, elegant, and effective

```
public static long estimate(Iterable<Long> stream, int M)
{
   byte[] sketch = new byte[M];
   for (String s : stream)
   {
      int x = hash1(s);
      int k = hash2(s, M);
      if Bits.r(x) > sketch[k]) sketch[k] = Bits.r(x);
   }
   double sum = 0.0;
   for (int i = 0; i < M; i++)
      sum += sketch[i];
   double mean = sum / M;
   return (int) (.794 * M * Math.pow(2, mean));
}
```

code to maintain
6-bit bytes omitted

Ideas.

- Use M sketches (6 bits each).
- Hash each item.
- Use second hash to split into M independent streams.
- Compute max # leading 0 bits in each stream (`Bits.r()` computes # leading 0s)
- Compute *mean* = average of the sketch values.
- Return .794 * 2^{mean}

magic constant

Critical questions

- Formula for the "magic constant"?
- How does the accuracy improve as M increases?

"Without the analysis there is no algorithm"

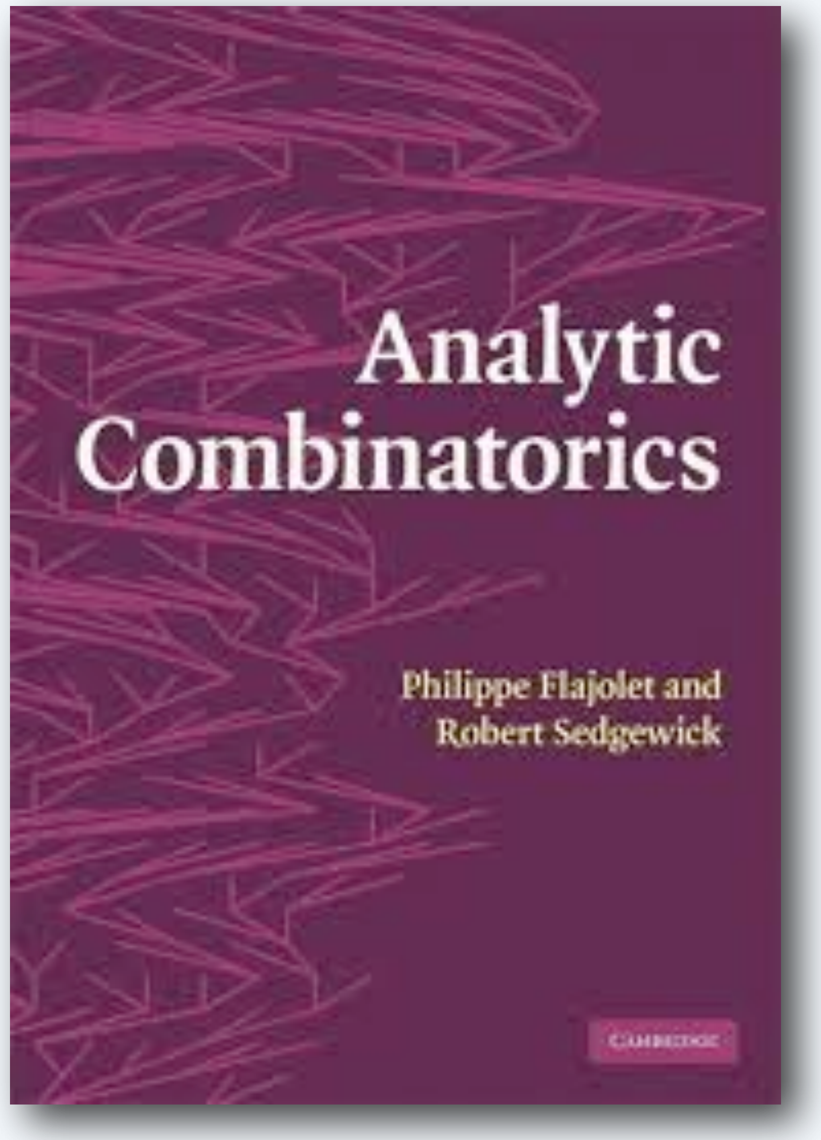
—-attributed to Flajolet (folklore)

LogLog Counting of Large Cardinalities

- A fundamental problem in data science
- A simple, elegant and practical solution
- **A poster child for analytic combinatorics**
- A poster child for algorithm science
- Broadly applicable and widely used

Analytic combinatorics

is a calculus enabling precise analysis of properties of discrete structures



Short history

- Foundations in classical mathematics, dating back to 18th century.
- Several "classical" examples found in Knuth's books.
- Developed by Flajolet and many coauthors in 1980s and 1990s.
- Defined in 2008 book by Flajolet and Sedgewick.
- *Applies to many sciences, not just computer science.*

Basic ideas

- Generating functions (GFs) are the central object of study.
- Formal methods transfer specifications into equations on GFs.
- Analysis of properties of GFs as complex functions gives asymptotic estimates of coefficients.
- *Universal laws that suppress detail in the analysis are often available.*

Analysis of loglog counting: overview

Theorem. (Durand and Flajolet) When loglog counting is used to process a stream of N distinct values using M substreams, the mean of the max # 0s in the streams has

expected value: $\lg N + \dfrac{\gamma}{\ln 2} + \dfrac{1}{2}$ (so the magic constant is $e^{-\gamma}\sqrt{2} \doteq .794028$)

standard error: $\sim c_M/\sqrt{M}$ where $c_M \sim \sqrt{(\ln 2)^2/12 + \pi^2/6} \approx 1.30$

(ignoring small oscillating functions of magnitude $< 10^{-6}$).

Initial stipulations about randomness omitted (stay tuned)

Proof sketch (standard analytic combinatorics, now).

Generating-function formulation

Poisson approximation

Mellin transform

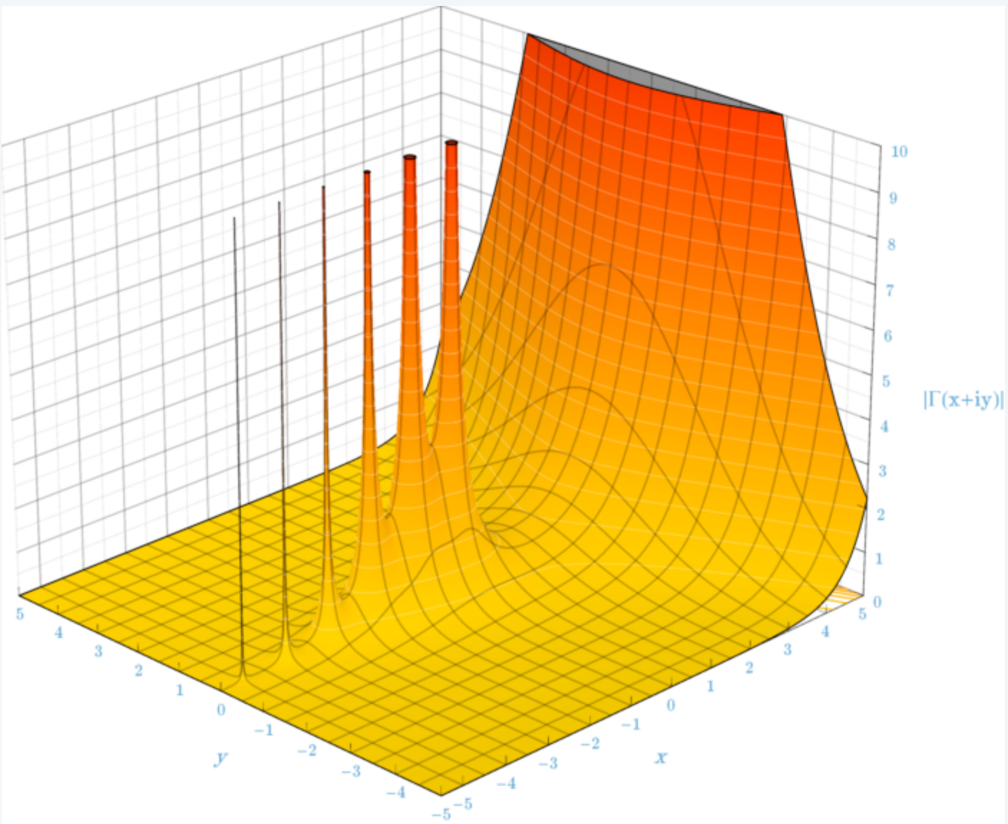
Depoissonize

Analysis of loglog counting: the magic constant

news flash: it's not a constant

Fact: To know the magic constant, one must study the Gamma function in the complex plane (!).

$$\Gamma(s) = \int_0^\infty e^x x^{s-1} dx$$



$$\Gamma(z) = \frac{1}{z} - \gamma + O(z) \text{ near } z = 0.$$

Euler's constant $\doteq .57721$

Q. For M=1, what is the mean of the max # leading 0s in a stream with *N* distinct values?

A. The sum of the residues of $\dfrac{\Gamma(z)N^{-z}}{1-2^{-z}}$ at the poles $z = \dfrac{2\pi i k}{\ln 2}$ $(k = 0, \pm 1, \pm 2, \ \ldots \ )$.

A. $\lg N + \dfrac{\gamma}{\ln 2} + \dfrac{1}{2} + \delta_N$ where $\delta_N = \dfrac{2}{\ln 2}\sum_{k\geq 1}\Gamma(-\dfrac{2k\pi i}{\ln 2})e^{2k\pi i \lg N}$

"Without the Gamma function there is no analysis"

—-RS

A. $\lg N + \dfrac{\gamma}{\ln 2} + \dfrac{1}{2}$, ignoring small oscillating functions of magnitude $< 10^{-6}$).

Analysis of loglog counting: the distribution

In 1973, Knuth and deBruijn developed the Γ-function method to analyze tries.

identified as the Mellin transform by Sedgewick in 1978

In 1985, Flajolet, Sedgewick, and Regnier generalized the method and identified applications.

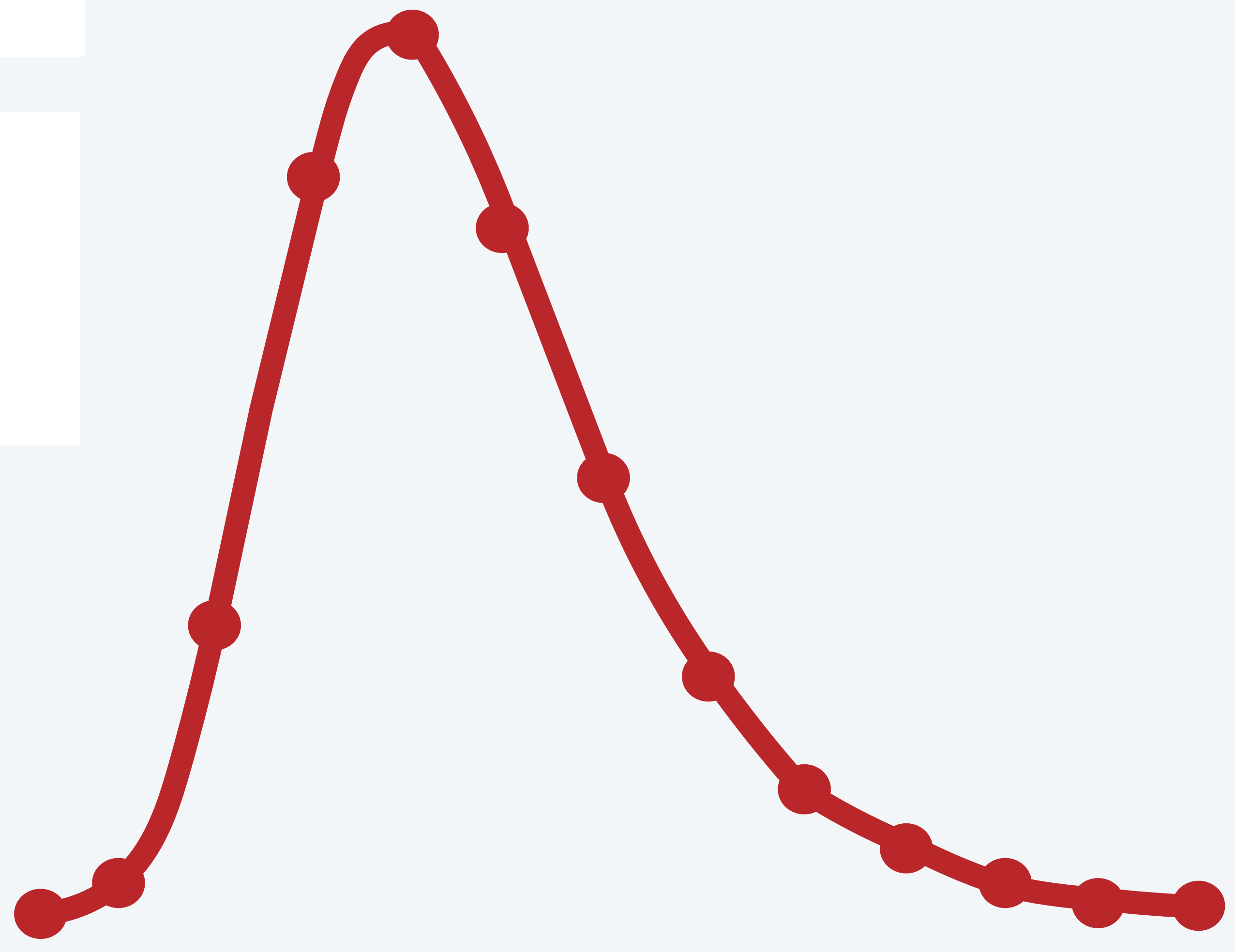
In 1998, Jacquet and Szpankowski introduced "poissonization"

- a quintessential result in analytic combinatorics.
- suppresses details in calculations for a broad class of problems.
- *allows approximation of the full distribution.*

In 2003, Durand and Flajolet analyzed the loglog counting *distribution*,
a necessary step to properly characterize memory-accuracy trade-offs

Q. What are the essential characteristics of the the distribution?

A. It is ***approximately normal*** with standard error about $1.30/\sqrt{M}$.



LogLog Counting of Large Cardinalities

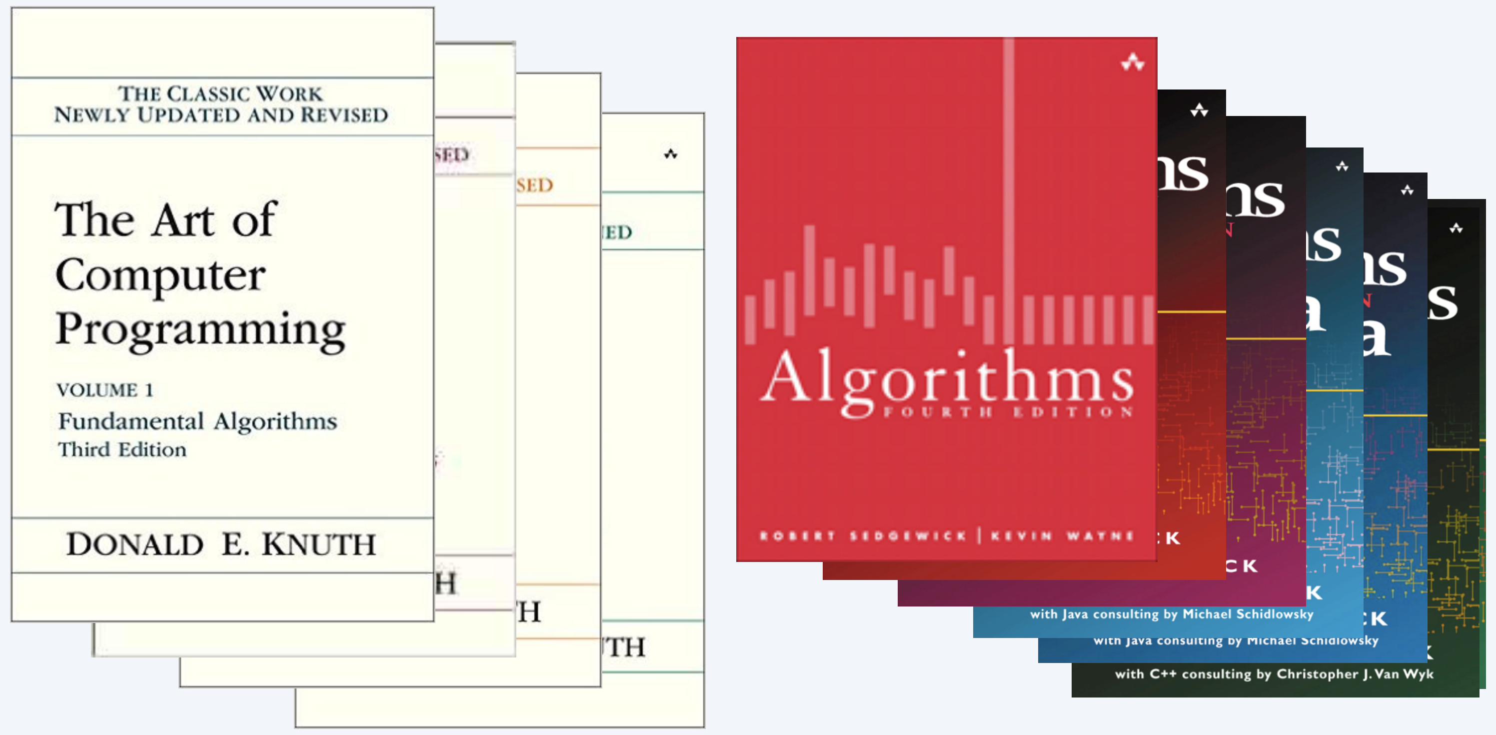
- A fundamental problem
- A simple, elegant and practical solution
- A poster child for analytic combinatorics
- **A poster child for algorithm science**
- Broadly applicable and widely used

Algorithm science

is an approach to studying algorithms that embraces the scientific method.

Short history

- Practiced by Turing, von Neumann, Hoare, and many others.
- Developed by Knuth in the 1970s and 1980s.
- Popularized by Sedgewick in the in 1990s and 2000s.
- *Enabled development of our computational infrastructure.*
- *Renewed focus likely as Moore's Law wanes.*



Basic ideas

- Start with real programs and real data.
- Develop a precise mathematical model of critical characteristics.
- Use model to formulate hypotheses about performance and tune parameter settings.
- Test hypthotheses with real-world experiments.
- Refine and iterate.

Initial hypothesis

Fact. Hash values are *not* random.

Hypothesis. Hash values are "sufficiently" random.

Implication. Need to run experiments to validate *any* hypotheses about performance.

No problem!

- We *always* validate hypotheses in algorithm science.
- End goal is development of algorithms that are useful in practice.
- It is the responsibility of the *designer* to validate utility before claiming it.
- After decades of experience, discovering a performance problem due to a bad hash function would be a significant research result.

Unspoken bedrock principle of algorithm science.

Experimenting to validate hypotheses is **WHAT WE DO!**

LogLog performance hypotheses

Hypothesis. (Durand and Flajolet) When loglog counting is used to process a stream of *N* distinct values using *M* substreams, the mean of the max # 0s in the streams has

expected value: $\sim \lg N/\alpha_M$ where $\alpha_M \sim e^{-\gamma}\sqrt{2} \approx .794$

standard error: $\sim c_M/\sqrt{M}$ where $c_M \sim \sqrt{(\ln 2)^2/12 + \pi^2/6} \approx 1.30$

(ignoring small oscillating functions of magnitude $< 10^{-6}$).

Also, the distribution is ***approximately normal*** .

Not a Theorem because it rests on unproven assumptions about the existence of truly random hash functions (***as do all programs that use hashing***)

Example. Estimate ***N*** to within 10% accuracy 95% of the time using thousands of bits of memory.

Hypothesis. *Loglog counting does so with 6144 bits.*

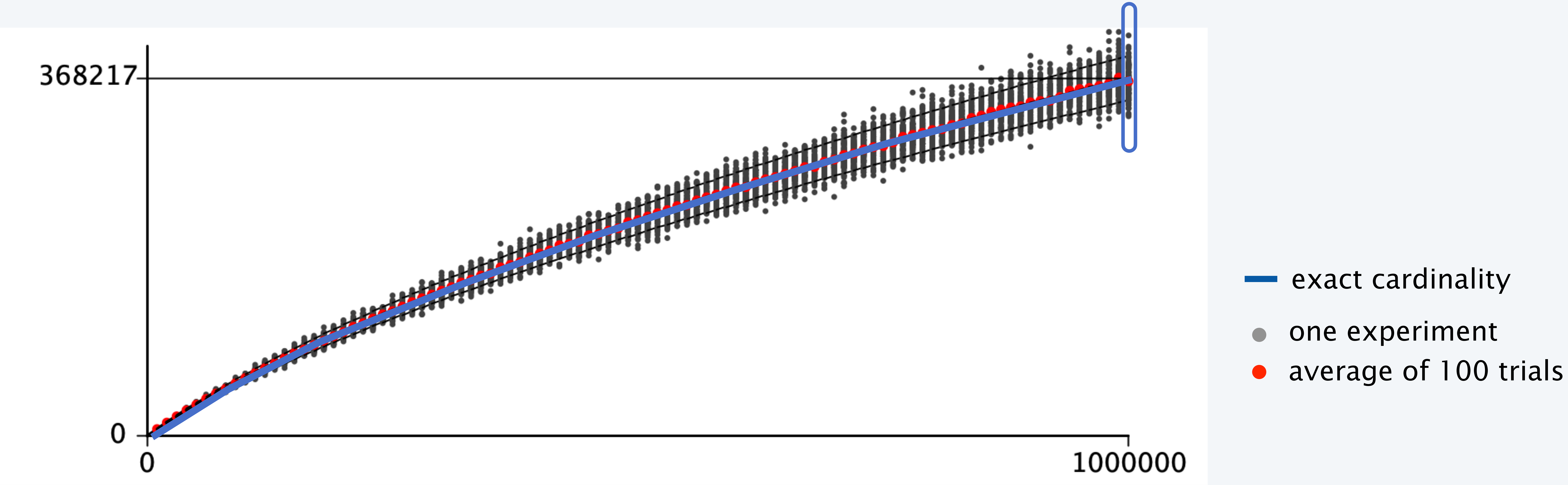
$$M = 1024$$

$$\sigma = 1.30/32 \doteq .04$$

within 2σ 95% of the time in a normal distribution

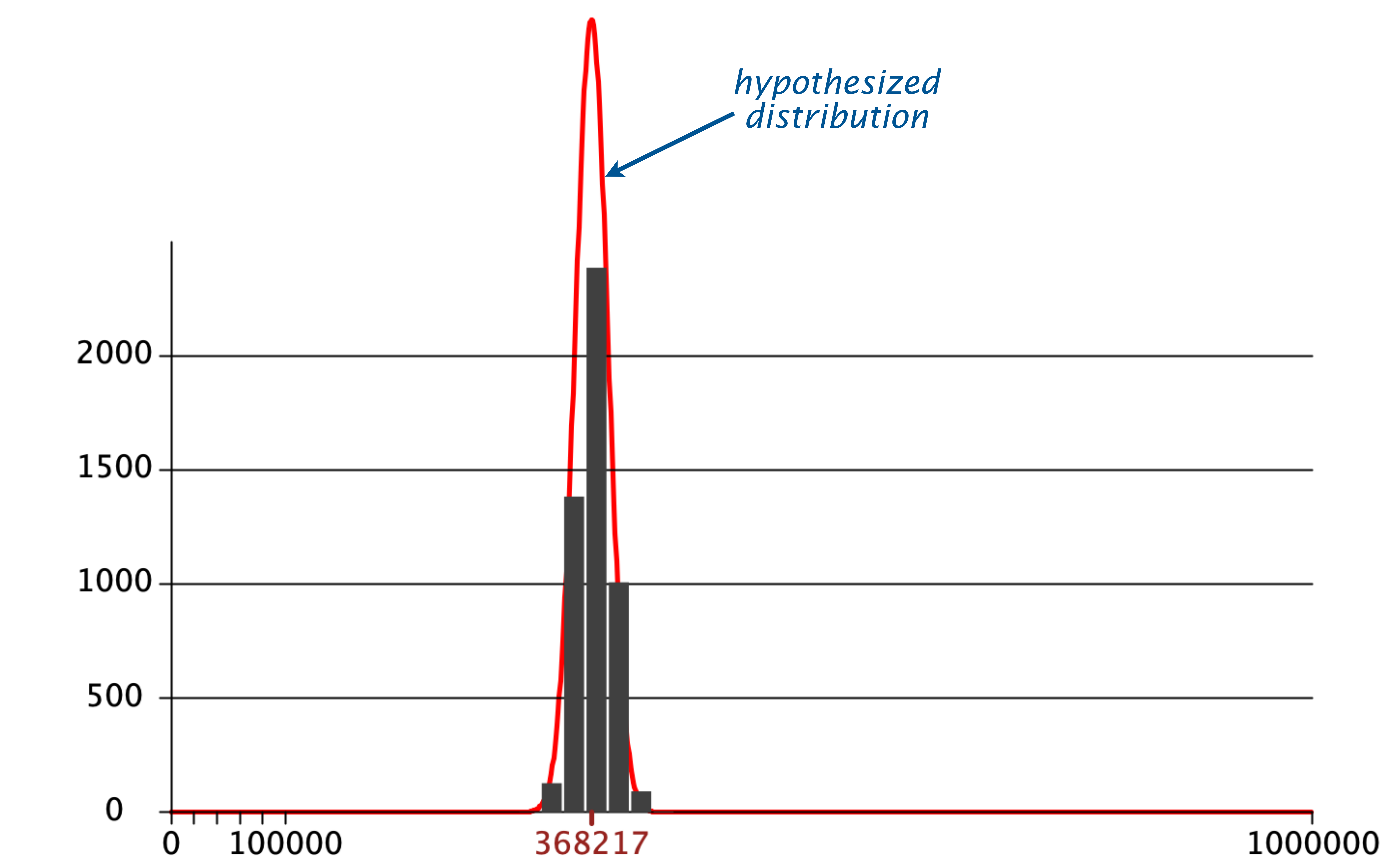
LogLog validation I (RS, 2023)

Experiment. 100 trials for x*10000 inputs for x from 1 to 100 (10000 trials) with $M = 1024$



LogLog validation II (RS, 2023)

Experiment. 10000 trials for 1 million inputs with $M = 1024$



*Histogram of number of estimates between x*2000 and (x+1)*2000*

Refine and iterate!

Q. How can loglog counting be improved?

A. SuperLogLog: Ignore large counts

- Reduces storage requirement to 5 bits per counter
- Achieves relative accuracy $1.05/\sqrt{M}$

A. HyperLogLog: Use harmonic mean

- Requires new analysis
- Achieves relative accuracy $1.02/\sqrt{M}$

A poster child for algorithm science

- Precise mathematical models guide study of improvements
- Testing infrastructure enables experimental validation

LogLog

SuperLogLog

HyperLogLog

Comparing algorithms and predicting performance: the constants matter

magic constant: Obviously.

standard error: Precisely characterizes memory-accuracy tradeoff.

Example 1.

Algorithm **A** has standard error $\dfrac{0.78}{\sqrt{M}}$ Algorithm **B** has standard error $\dfrac{1.103}{\sqrt{M}}$

B needs ***twice as many substreams*** to achieve the same accuracy as **A** ! $\dfrac{1.103}{\sqrt{2}} \doteq 0.78$

Example 2.

Algorithm **A** uses M_A total bits with 4 bits per substream

Algorithm **B** uses M_B total bits with 256 bits per substream

B is ***eight times less accurate*** if using the same memory as **A** !

$$\frac{1}{\sqrt{M_A/4}} = \frac{2}{\sqrt{M}} \qquad \frac{1}{\sqrt{M_B/256}} = \frac{16}{\sqrt{M}}$$

Algorithm comparisons: memory

Q. How many bits are needed to expect accuracy within $1 \pm x$?

A. Using M_0 bits with b bits per item, the number of streams is M_0/b. Therefore,

when the relative error is $\dfrac{c}{\sqrt{M_0/b}}$, solve for M_0 to get $M_0 = b\left(\dfrac{c}{x}\right)^2$

	memory needed	*variant*	*b*	*c*	*# bits for 2% accuracy*	*# bits for 20% accuracy*
Adaptive Sampling	M *records*		64	1.20	166464	1664
Probabilistic Counting	M *words*		64	0.78	97344	973
LogLog	M *bytes*	basic	6	1.30	25350	253
		super	5	1.05	13871	139
		hyper	5	1.02	13005	130

Algorithm comparisons: accuracy

Q.What accuracy can be expected with a given number of bits?

A. For b bits per item and Mb bits, relative error is $\dfrac{c}{\sqrt{M}}$.

	memory needed	variant	b	c	relative error with 128 bits	relative error with 8K bits
Adaptive Sampling	M *records*		64	1.20	84%	10%
Probabilistic Counting	M *words*		64	0.78	55%	7%
LogLog	M *bytes*	basic	6	1.30	28%	3.5%
		super	5	1.05	21%	2.6%
		hyper	5	1.02	20%	2.1%

LogLog Counting of Large Cardinalities

- A fundamental problem in data science
- A simple, elegant and efficient solution
- A poster child for algorithm science
- A poster child for analytic combinatorics
- Broadly applicable and widely used

Posted on Facebook, 2018

"Computing the count of distinct elements in massive data sets is often necessary but computationally intensive. Say you need to determine the number of distinct people visiting Facebook in the past week using a single machine. With a traditional SQL query on the data sets we use at Facebook this would take days and terabytes of memory.

Posted on Facebook, 2018 (continued)

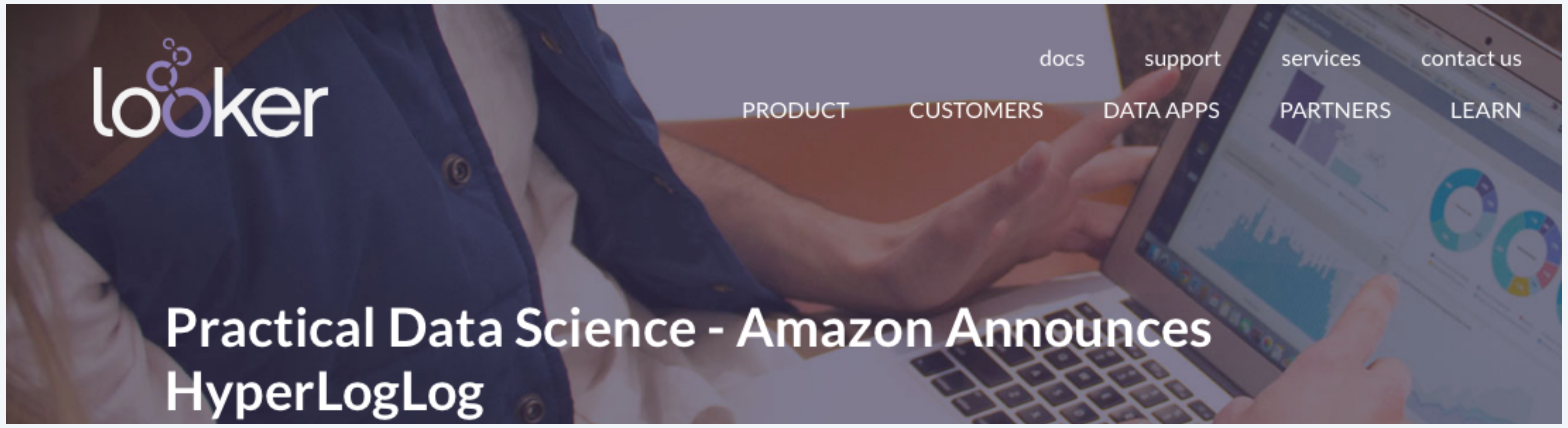
Computing the count of distinct elements in massive data sets is often necessary but computationally intensive. Say you need to determine the number of distinct people visiting Facebook in the past week using a single machine. With a traditional SQL query on the data sets we use at Facebook this would take days and terabytes of memory.

To speed up these queries, we implemented HyperLogLog (HLL) in Presto, a distributed SQL query engine. HLL works by providing an approximate count of distinct elements. With HLL, we can perform the same calculation in 12 hours with ***less than 1 MB of memory****. We have seen great improvements, with some queries being run* ***within minutes****.*

Note: 1 terabyte = 1 million MB

Improving things by a factor of 1 million = a good day for an algorithm scientist!

Hyperloglog validation in the Real World



cradlepoint

redis

Periscope DATA

neustar

Research at Google

S. Heule, M. Nunkesser and A. Hall
HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm.
Extending Database Technology/International Conference on Database Theory 2013.

AMAZON REDSHIFT

BigQuery

facebook

Passing the test of time

Why does log-log counting pass the test of time?

It solves a **fundamental problem** in *data science.*

It is a **simple, elegant** and **efficient** solution.

It is a poster child for **algorithm science**.

It is a poster child for **analytic combinatorics.**

It is **broadly applicable** and **widely used.**

Philippe Flajolet 1948-2011

Marianne Maurel (recent)

2023 ESA Test-of-Time Award

LogLog Counting of Large Cardinalities

Marianne Durand and Philippe Flajolet

presented by

Robert Sedgewick
Princeton University