



# Cardinality Estimation

**Robert Sedgewick**  
**Princeton University**

with special thanks to Jérémie Lumbroso

*Philippe Flajolet, mathematician and computer scientist extraordinaire*



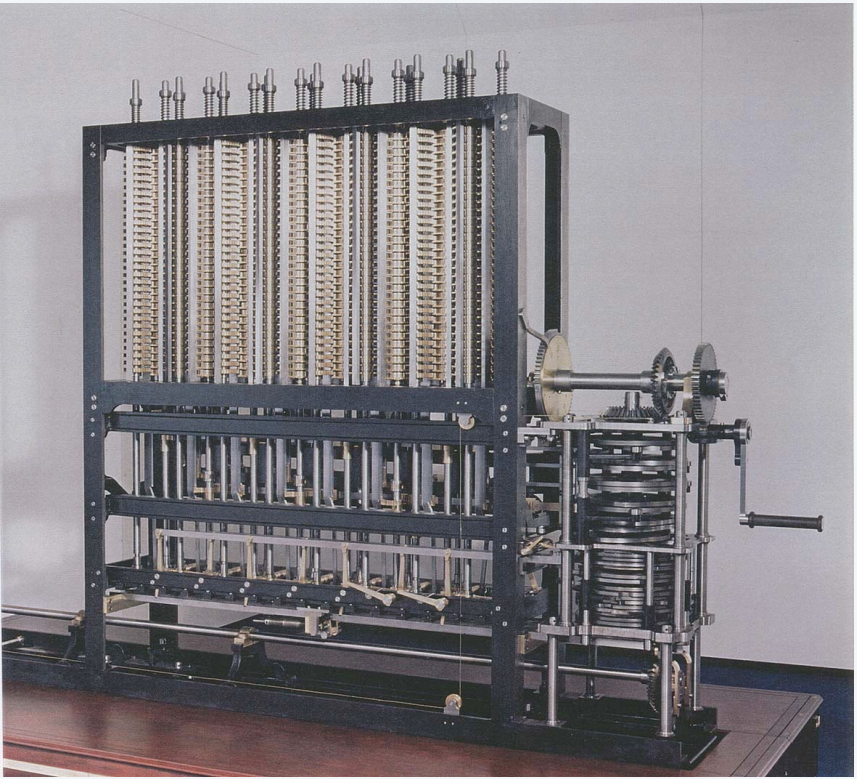
Philippe Flajolet 1948–2011



# Don Knuth's legacy: Analysis of Algorithms (AofA)

Understood since Babbage:

- Computational resources are limited.
- Method (algorithm) used matters.



Analytic Engine

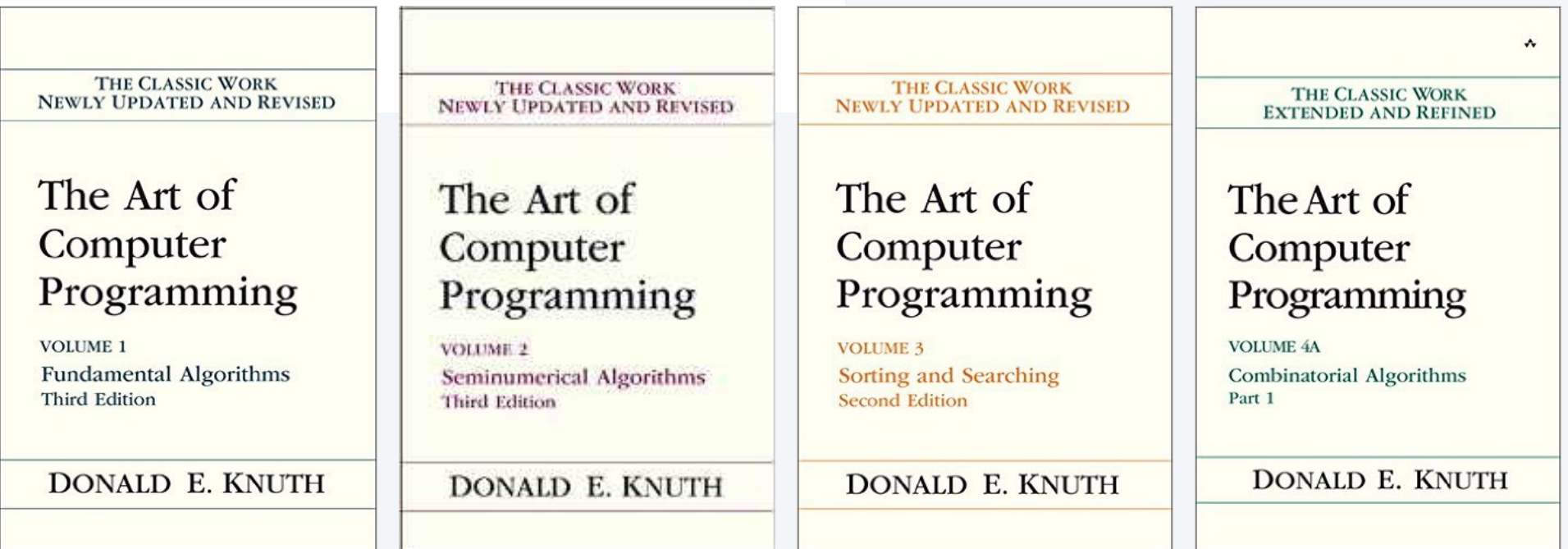
*how many times do we  
have to turn the crank?*



Knuth's insight: AofA is a *scientific* endeavor.

- Start with a working program (algorithm implementation).
- Develop mathematical model of its behavior.
- Use the *model* to formulate hypotheses on resource usage.
- Use the *program* to validate hypotheses.
- Iterate on basis of insights gained.

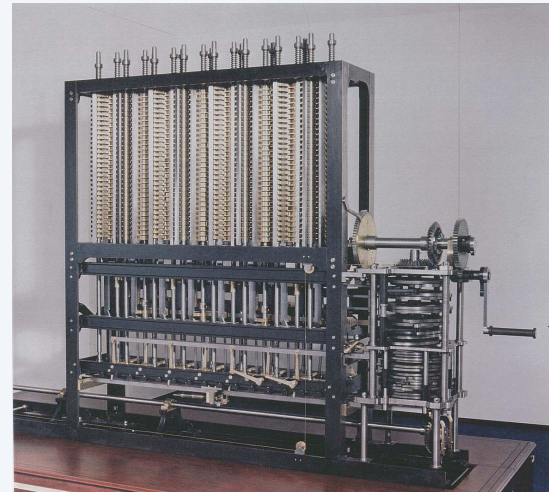
Difficult to overstate the significance of this insight.





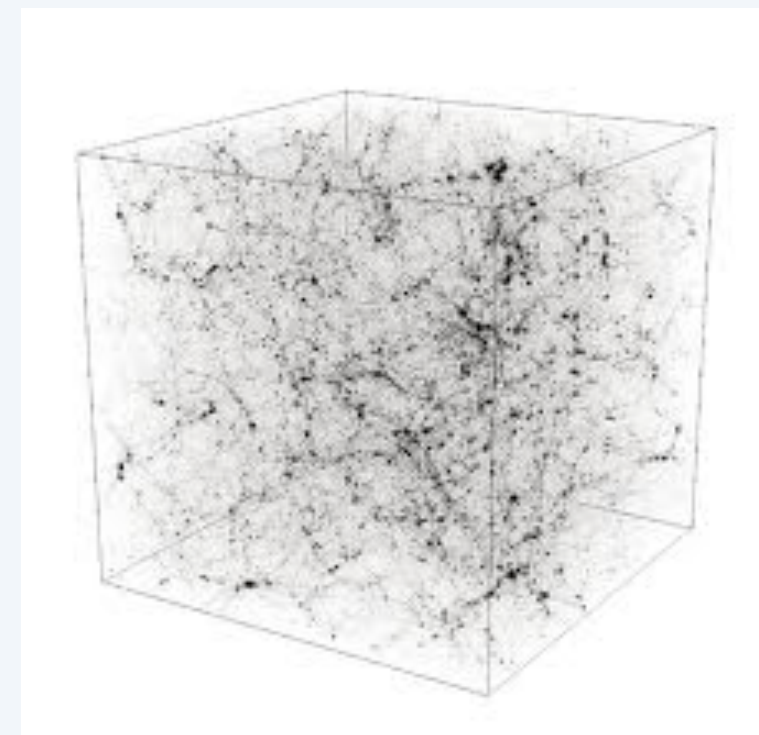
# AofA has played a critical role

in the development of our computational infrastructure *and the advance of scientific knowledge*

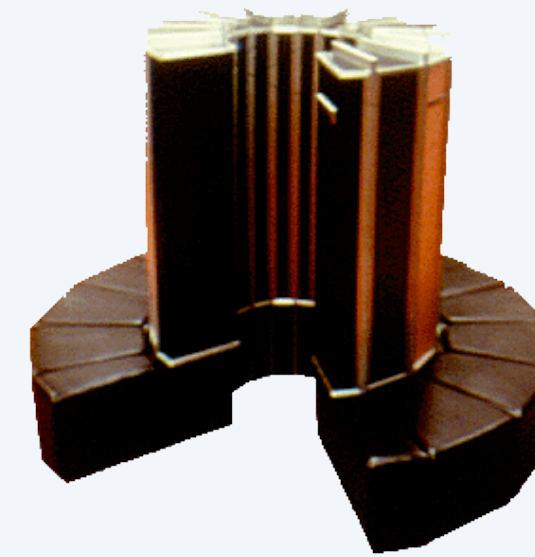


how many times  
to turn the crank?

how long to compile  
my program?

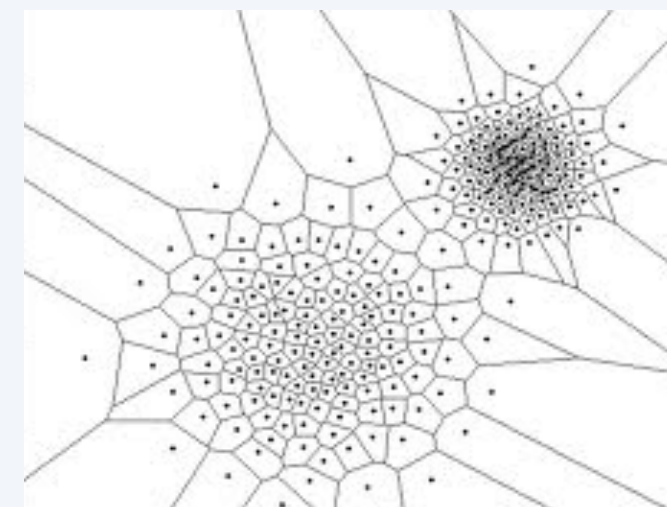
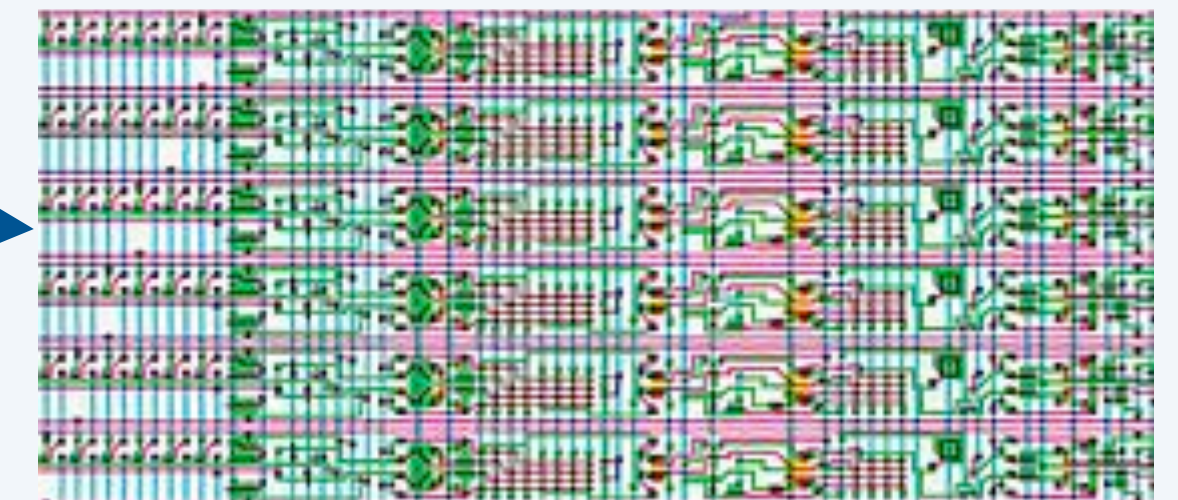


how many bodies  
in motion can I  
simulate?



how long to sort random data for  
cryptanalysis preprocessing?

how long to check  
that my VLSI circuit  
follows the rules?



how quickly can I find clusters?

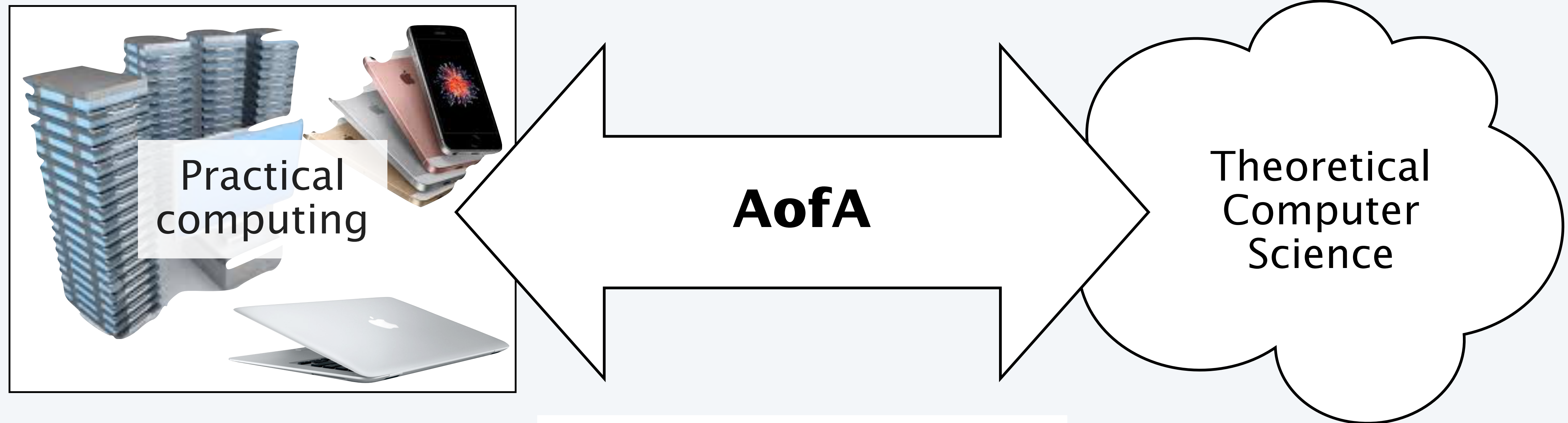
*“PEOPLE WHO ANALYZE ALGORITHMS have double happiness. They experience the sheer beauty of elegant mathematical patterns that surround elegant computational procedures. Then they receive a practical payoff when their theories make it possible to get other jobs done more quickly and more economically.”*

– Don Knuth





# Analysis of Algorithms (present-day context)



## Practical computing

- Real code on real machines
- Thorough validation
- Limited math models

## AofA

- Theorems *and* code
- Precise math models
- Experiment, validate, iterate

## Theoretical computer science

- Theorems
- Abstract math models
- Limited experimentation

# Cardinality Estimation

- **Warmup: exact cardinality count**
- Probabilistic counting
- Stochastic averaging
- Refinements
- Final frontier

# 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***

```
109.108.229.102
pool-71-104-94-246.lsanca.dsl-w.verizon.net
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
```

*6 million strings*

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

***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
```

# Standard “optimal” solution: Use a hash table

## Hashing with linear probing

- Create a table of size  $M$ .
- Transform each value into a “random” table index.
- Move right to find space if value collides.
- Count values new to the table.

**example:** multiply by a prime,  
then take remainder after dividing by  $M$ .

*small example data stream*

P J J E K J L C K O M T P G L J I F K C

*hash values  $(x - 'A') * 97 \% 17$*

15 6 6 14 1 6 13 7 1 15 8 7 15 4 13 6 11 9 1 7

*hash table ( $M = 17$ )*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	K			G		J	C	M			I		L	E	P	
							T		F						O	

*count*

11

**Additional (key) idea.** Keep searches short by doubling table size when it becomes half full.



# Mathematical analysis of exact cardinality count with linear probing

**Theorem.** Expected time and space cost is linear.

**Proof.** Follows from classic Knuth Theorem 6.4.K.

**Theorem K.** The average number of probes needed by Algorithm L, assuming that all  $M^N$  hash sequences (35) are equally likely, is

$$C_N = \frac{1}{2}(1 + Q_0(M, N-1)) \quad (\text{successful search}), \quad (40)$$

$$C'_N = \frac{1}{2}(1 + Q_1(M, N)) \quad (\text{unsuccessful search}), \quad (41)$$

where

$$\begin{aligned} Q_r(M, N) &= \binom{r}{0} + \binom{r+1}{1} \frac{N}{M} + \binom{r+2}{2} \frac{N(N-1)}{M^2} + \dots \\ &= \sum_{k \geq 0} \binom{r+k}{k} \frac{N}{M} \frac{N-1}{M} \dots \frac{N-k+1}{M}. \end{aligned} \quad (42)$$

*Proof.* Details of the calculation are worked out in exercise 27. (For the variance, see exercises 28, 67, and 68.) ■

*“I first formulated [this] derivation in 1962. Since this was the first nontrivial algorithm I had ever analyzed satisfactorily, it had a strong influence on the structure of these books. Ever since that day, the analysis of algorithms has in fact been one of the major themes of my life.”*

– Knuth, TAOCP volume 3

THE CLASSIC WORK  
NEWLY UPDATED AND REVISED

The Art of  
Computer  
Programming

VOLUME 3  
Sorting and Searching  
Second Edition

DONALD E. KNUTH

**Q.** Do the hash functions that we use *uniformly* and *independently* distribute keys in the table?

**A.** Not likely.

# Scientific validation of exact cardinality count with linear probing

**Hypothesis.** Time and space cost is *linear for the hash functions we use and the data we have.*

**Quick experiment.** Doubling the problem size should double the running time.

## ***Driver to read N strings and count distinct values***

*get problem size*  
*initialize input stream*  
*get current time*

*print count*

*print elapsed time*

```
public static void main(String[] args)
{
    int N = Integer.parseInt(args[0]);
    StringStream stream = new StringStream(N);
    long start = System.currentTimeMillis();

    StdOut.println(count(stream));

    long now = System.currentTimeMillis();
    double time = (now - start) / 1000.0;
    StdOut.println(time + " seconds");
}
```

```
% java Hash 2000000 < log.07.f3.txt
483477
```

**3.322** seconds

```
% java Hash 4000000 < log.07.f3.txt
883071
```

**6.55** seconds

```
% java Hash 6000000 < log.07.f3.txt
1097944
```

**9.49** seconds



```
% sort -u log.07.f3 | wc -l
1097944
```

↑  
sort-based method  
takes about 3 minutes

**Q.** Is hashing with linear probing effective?

**A.** Yes. Validated in countless applications for *over half a century.*



# Complexity of exact cardinality count

Q. Does there exist an *optimal* algorithm for this problem?

A. Depends on definition of “optimal”.

*Guaranteed linear-time?* NO. Linearithmic lower bound.

*Guaranteed linearithmic?* YES. Balanced BSTs or mergesort.

*Linear-time with high probability assuming the existence of random bits?*

YES. Dynamic perfect hashing.

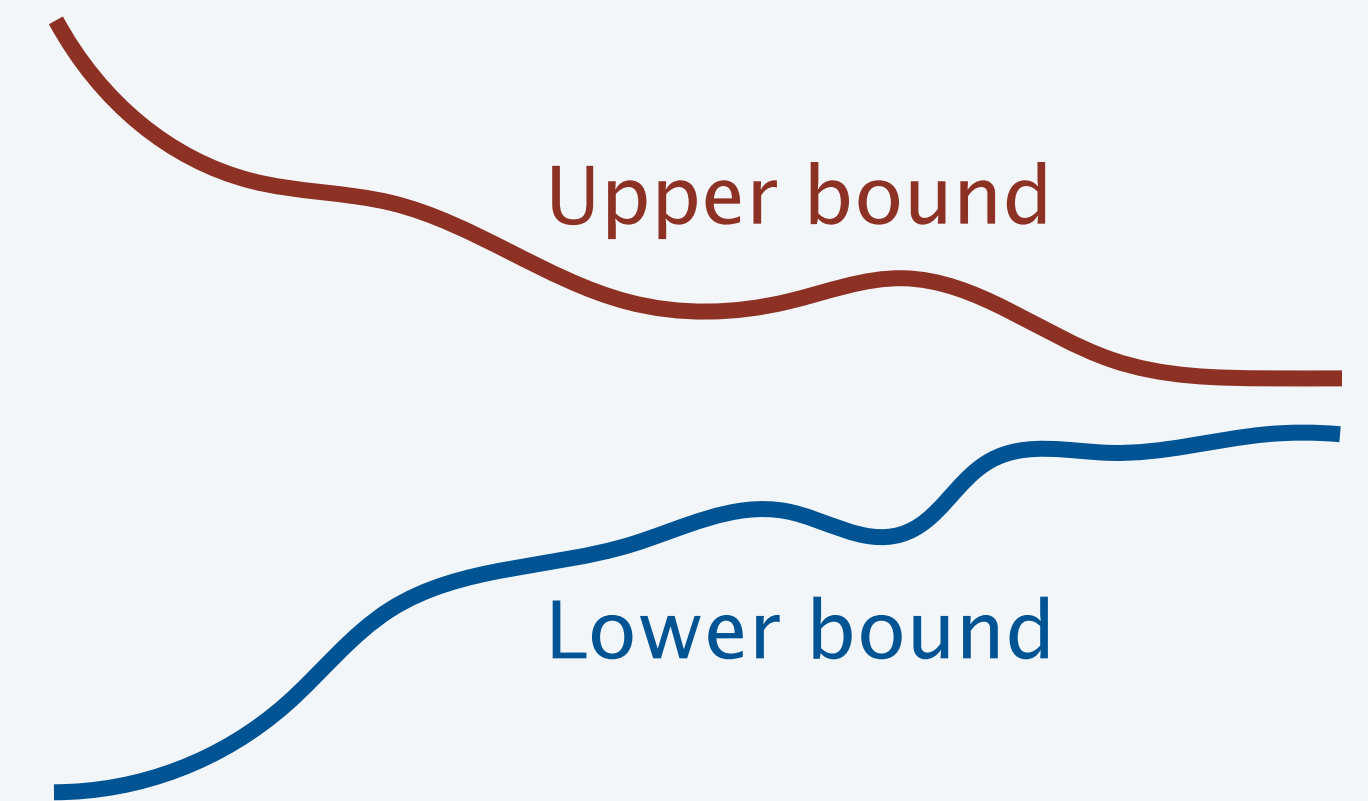
Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert, and Tarjan  
*Dynamic Perfect Hashing: Upper and Lower Bounds*  
SICOMP 1994.

*Linear with a small constant factor in practical situations?*

YES. Hashing with linear probing.

M. Mitzenmacher and S. Vadhan  
*Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream.*  
SODA 2008.

Hypothesis. Hashing with linear probing is “optimal”. ← but TSTs may give a sublinear algorithm



# Exact cardinality count requires linear space

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

**A.** Bad news: You cannot get an exact count.

**A.** (Bloom, 1970) You can get an accurate *estimate* using a few bits per distinct value.

```
109.108.229.102
pool-71-104-94-246.lsanca.dsl-w.verizon.net
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
msnbot-131-253-46-251.search.msn.com
msnbot-131-253-46-251.search.msn.com
```



**A.** Much better news: *You can get an accurate estimate using only a handful of bits* (stay tuned).



# Cardinality Estimation

- Warmup: exact cardinality count
- **Probabilistic counting**
- Stochastic averaging
- Refinements
- Final frontier

OF

# Cardinality *estimation*

is a fundamental problem with many applications *where memory is limited*.

Q. *About* how many different values appear in a given stream?

## Constraints

- 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*

How many unique  
visitors to my website?

Which sites are the  
most/least popular?

How many different websites  
visited by each customer?

How many different values  
for a database join?

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



# Probabilistic counting with stochastic averaging (PCSA)

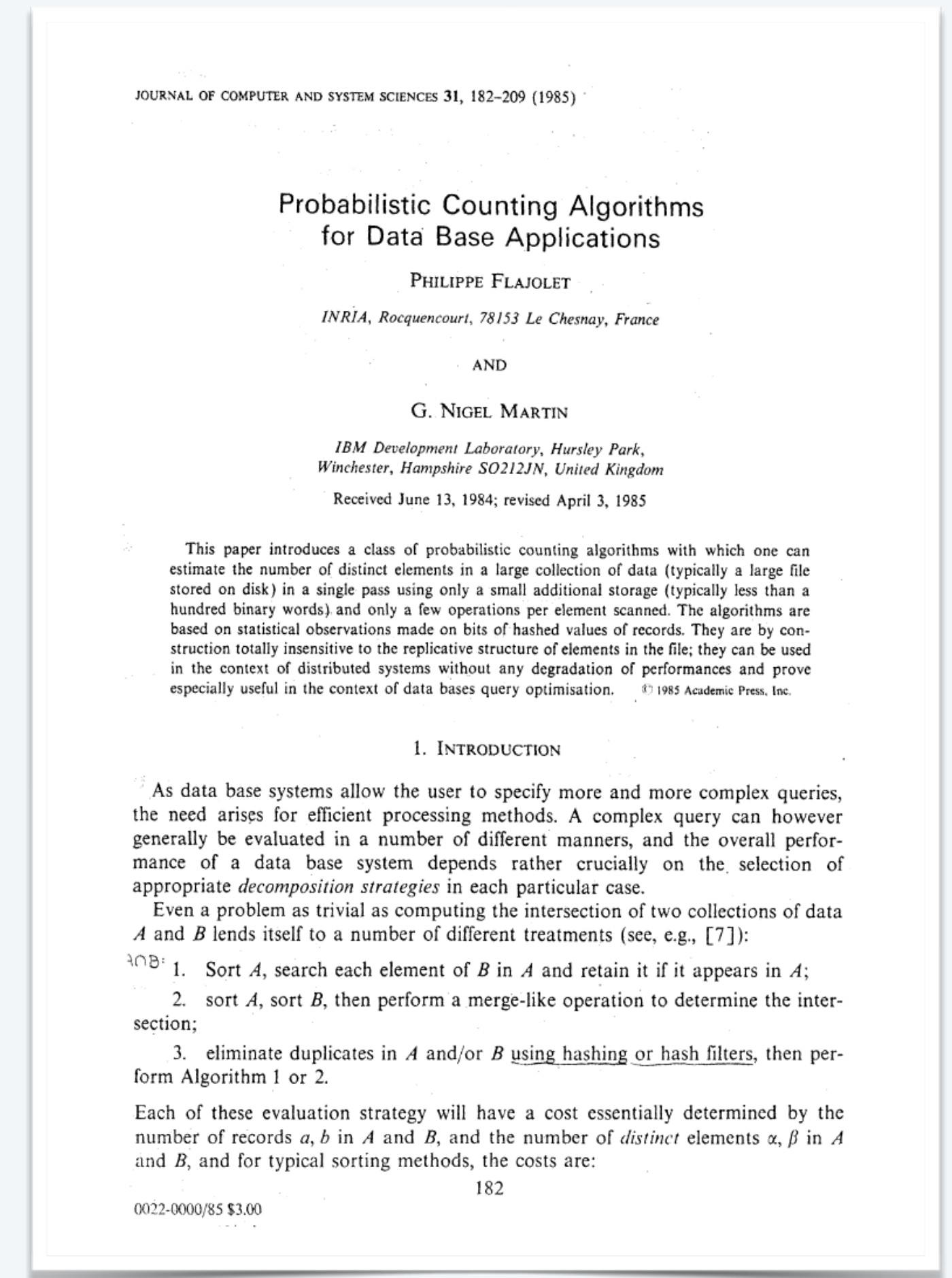
Flajolet and Martin, *Probabilistic Counting Algorithms for Data Base Applications* FOCS 1983, JCSS 1985.



**Philippe Flajolet 1948–2011**

## Contributions

- Introduced problem
- Idea of *streaming algorithm*
- Idea of “small” *sketch* of “big” data
- Detailed analysis that yields tight bounds on accuracy
- Full validation of mathematical results with experimentation
- Practical algorithm that has remained effective for decades



**Bottom line:** Quintessential example of the effectiveness of scientific approach to algorithm design.

# PCSA first step: Use hashing

## 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 (quadrillions of values)

### Example: Java

- All data types implement a `hashCode()` method (though we often override the default).
- String data type stores value (computed once).

```
String value = "gsearch.CS.Princeton.EDU"  
int x = value.hashCode();
```

current Java default  
is 32-bit int value

**Bottom line:** Do cardinality estimation on streams of (binary) integers.

```
01111000100111110111000111001000  
01111000100111110111000111001000  
01110101010110110000000011011010  
00110100010001111100010100111010  
00010000111001101000111010010011  
00001001011011100000010010010111  
00001001011011100000010010010111
```

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



# Initial hypothesis

---

**Hypothesis.** Uniform hashing assumption is reasonable in this context.

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

No problem!

- AofA is a scientific endeavor (we always validate hypotheses).
- 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 AofA.

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



# Probabilistic counting starting point: three integer functions

Definition.  $p(x)$  is the **number of 1s** in the binary representation of  $x$ .

Definition.  $r(x)$  is the **number of trailing 1s** in the binary representation of  $x$ .  *position of rightmost 0*

Definition.  $R(x) = 2^{r(x)}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	$p(x)$	$r(x)$	$R(x)$	$R(x)_2$
1	0	1	1	1	1	0	1	1	1	1	1	0	1	0	1	12	1	2	10
1	0	1	0	1	0	1	0	1	0	0	0	1	1	1	0	8	0	1	1
0	1	1	0	1	0	0	1	0	1	0	1	1	1	1	1	10	5	32	100000

*Bit-whacking magic:*

$R(x)$  is easy to compute.

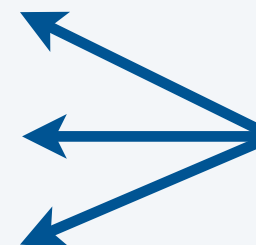
0	1	1	0	1	0	0	1	0	1	0	1	1	1	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	0	0	0
0	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

$x$

$\sim x$

$x + 1$

$\sim x \& (x + 1)$

 *3 instructions  
on a typical  
computer*

Exercise: Compute  $p(x)$  as easily.

Note:  $r(x) = p(R(x) - 1)$ .

Beeler, Gosper, and Schroepel

HAKMEM item 169, MIT AI Laboratory AIM 239, 1972

<http://www.inwap.com/pdp10/hbaker/hakmem/hakmem.html>

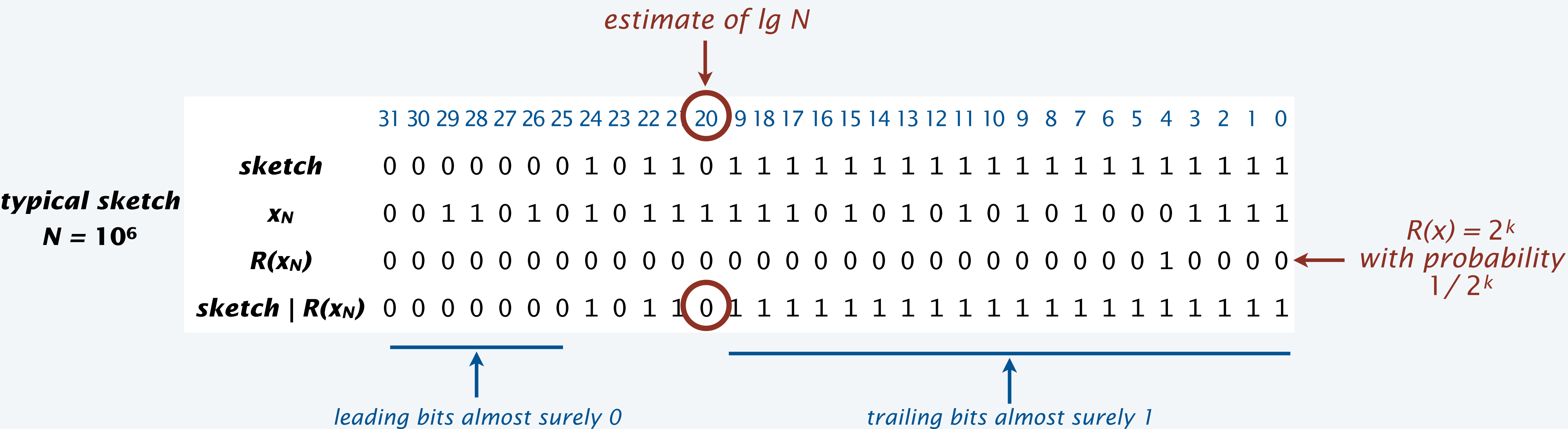
 *see also Knuth volume 4A*

Bottom line:  $p(x)$ ,  $r(x)$ , and  $R(x)$  all can be computed with *just a few machine instructions*.

# Probabilistic counting (Flajolet and Martin, 1983)

Maintain a single-word *sketch* that summarizes a data stream  $x_0, x_1, \dots, x_N, \dots$

- For each  $x_N$  in the stream, update sketch by *bitwise or* with  $R(x_N)$ .
- Use *position of rightmost 0* (with slight correction factor) to estimate  $\lg N$ .



Rough estimate of  $\lg N$  is  $r(\text{sketch})$ .

Rough estimate of  $N$  is  $R(\text{sketch})$ .

← *correction factor needed (stay tuned)*



# Probabilistic counting trace

<i>x</i>	<i>r(x)</i>	<i>R(x)</i>	<i>sketch</i>
011000100110001110100111101110 <b>11</b>	2	100	00000000000000000000000000000000 <b>100</b>
0110011100100011000111110000010 <b>1</b>	1	10	00000000000000000000000000000000 <b>110</b>
000100010001110001101101101100 <b>11</b>	2	100	00000000000000000000000000000000 <b>110</b>
010001000111011100000001110 <b>1111</b>	5	100000	00000000000000000000000000000000 <b>100110</b>
01101000001011000101110001000100	0	1	0000000000000000000000000000000010011 <b>1</b>
0011011110110000000010100101010 <b>1</b>	1	10	000000000000000000000000000000001001 <b>11</b>
00110100011000111010101111111100	0	1	000000000000000000000000000000001001 <b>11</b>
00011000010000100001011100110 <b>111</b>	3	1000	0000000000000000000000000000000010 <b>1111</b>
00011001100110011110010000 <b>111111</b>	6	1000000	00000000000000000000000000000000 <b>1101111</b>
01000101110001001010110011111100	0	1	00000000000000000000000000000000110 <b>1111</b>

$R(\text{sketch}) = 10000_2$   
 $= 16$

# Probabilistic counting (Flajolet and Martin, 1983)

```
public long R(long x)
{ return ~x & (x+1); }

public long estimate(Iterable<String> stream)
{
    long sketch;
    for (s : stream)
        sketch = sketch | R(s.hashCode());
    return R(sketch) /.77351;
}
```

Maintain a *sketch* of the data

- A single word
- OR of all values of  $R(x)$  in the stream
- Return smallest value not seen

*with correction for bias*

Early example of “a simple algorithm whose analysis isn’t”

**Q.** (Martin) Estimate seems a bit low. How much?

**A.** (unsatisfying) Obtain correction factor empirically.

**A.** (Flajolet) Without the analysis, there is no algorithm!

Magic is  
something  
you make.

# Mathematical analysis of probabilistic counting

**Theorem.** *The expected number of trailing 1s in the PC sketch is*

$$\lg(\phi N) + P(\lg N) + o(1) \quad \text{where } \phi \doteq .77351$$

*and  $P$  is an oscillating function of  $\lg N$  of very small amplitude.*

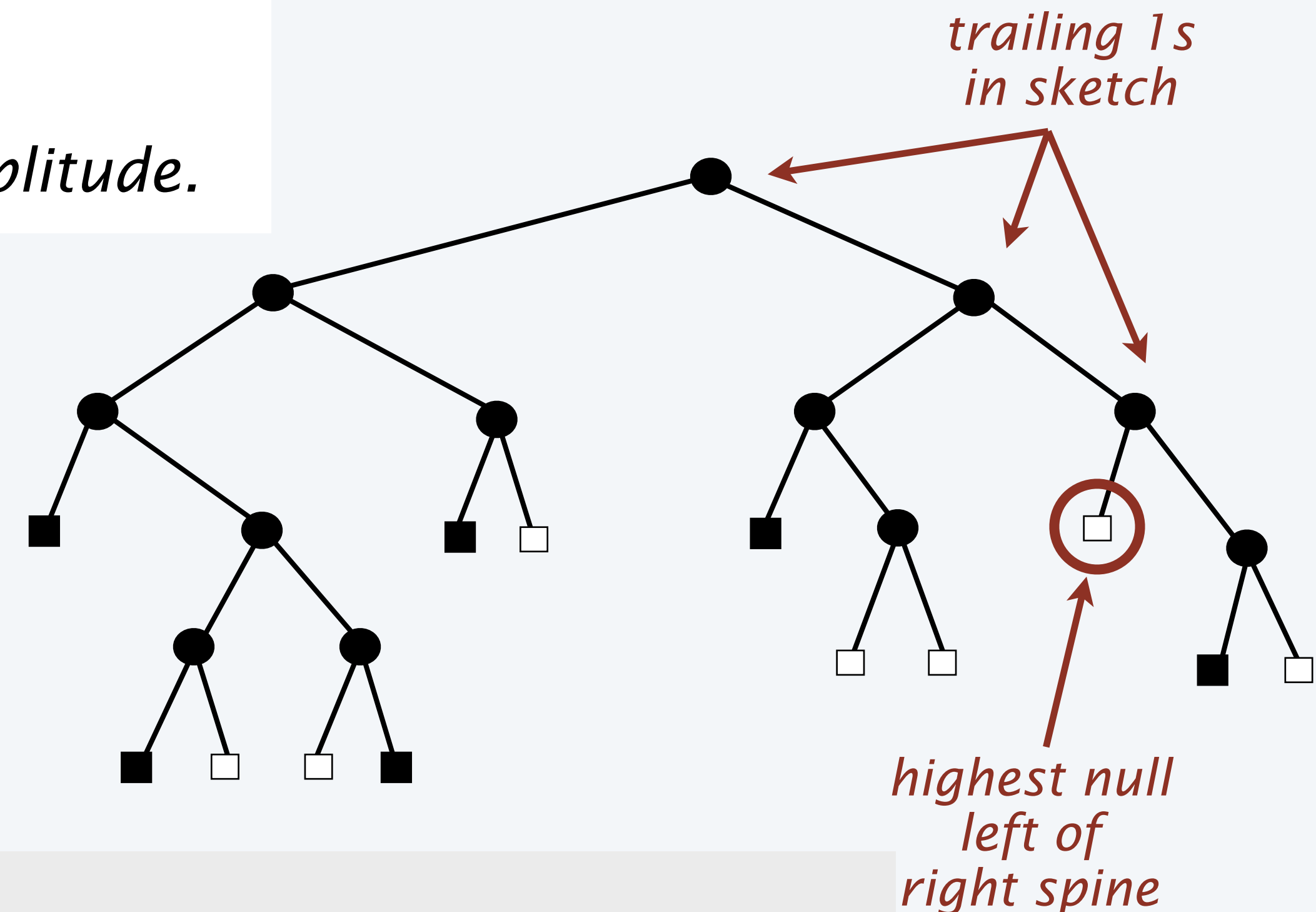
*Proof (omitted).*

1980s: Flajolet *tour de force*

1990s: trie parameter

21st century: standard AC

*stay tuned for  
Szpankowski talk*



Kirschenhofer, Prodinger, and Szpankowski

*Analysis of a splitting process arising in probabilistic counting and other related algorithms, ICALP 1992.*

Jacquet and Szpankowski

*Analytical depoissonization and its applications, TCS 1998.*

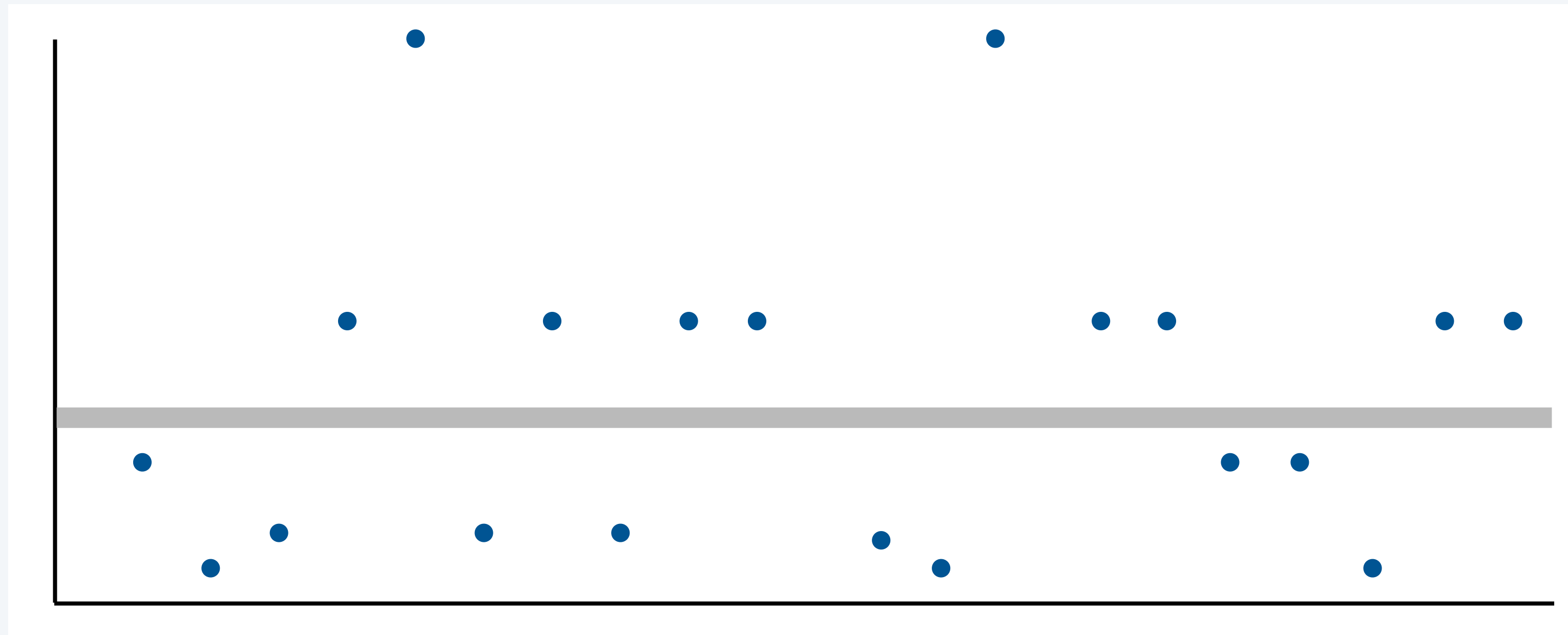
**In other words.** In PC code,  $R(\text{sketch}) / .77351$  is an *unbiased statistical estimator* of  $N$ .



# Validation of probabilistic counting

**Hypothesis.** Expected value returned is  $N$  *for random values from a large range.*

**Quick experiment.** 100,000 31-bit random values (20 trials)



Flajolet and Martin: Result is “typically one binary order of magnitude off.”

**Of course!** (Always returns a power of 2 divided by .77351.)

**Need to incorporate more experiments for more accuracy.**

$16384 / .77351 = 21181$   
 $32768 / .77351 = 42362$   
 $65536 / .77351 = 84725$

...

# Cardinality Estimation

- Rules of the game
- Probabilistic counting
- **Stochastic averaging**
- Refinements
- Final frontier

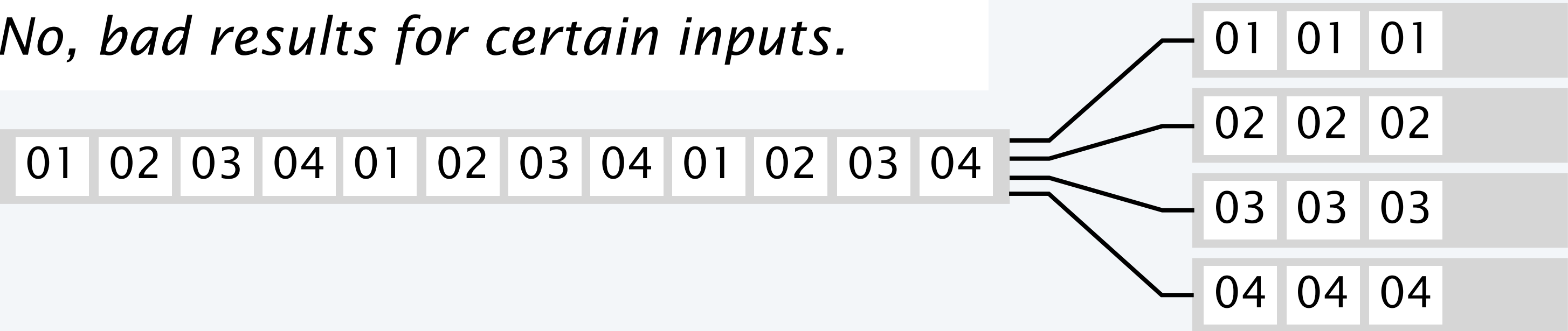
OF

# Stochastic averaging

Goal: Perform  $M$  independent PC experiments and average results.

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

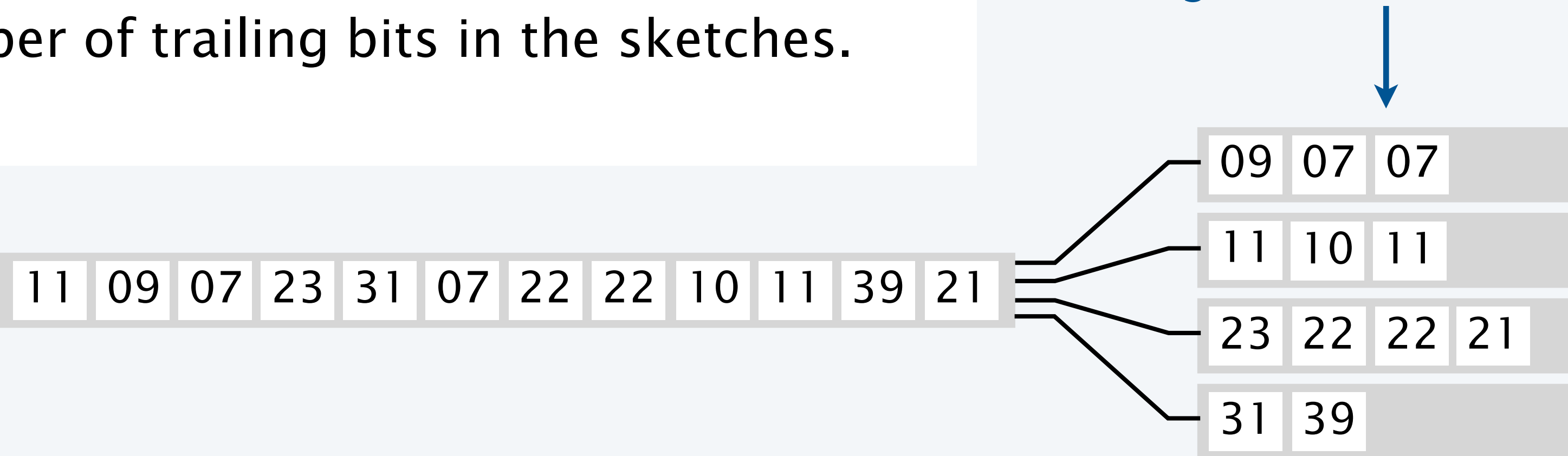
Alternative 2:  $M$ -way alternation? *No, bad results for certain inputs.*



Alternative 3: *Stochastic averaging*

- Use second hash to divide stream into  $2^m$  independent streams
- Use PC on each stream, yielding  $2^m$  sketches .
- Compute *mean* = average number of trailing bits in the sketches.
- Return  $2^{\text{mean}}/.77531$ .

*key point: equal values  
all go to the same stream*





PCSA trace

use initial  $m$  bits  
for second hash

$M = 4$

$x$	$R(x)$	$sketch[0]$	$sketch[1]$	$sketch[2]$	$sketch[3]$
<b>10</b> 100111101110 <b>11</b>	100	0000000000000000	0000000000000000	0000000000000 <b>100</b>	0000000000000000
<b>00</b> 0111110000010 <b>1</b>	10	000000000000000 <b>10</b>	0000000000000000	0000000000000100	0000000000000000
<b>01</b> 101101101100 <b>11</b>	100	00000000000000010	0000000000000 <b>100</b>	0000000000000100	0000000000000000
<b>00</b> 000001110 <b>11111</b>	100000	0000000000 <b>100010</b>	0000000000000100	0000000000000100	0000000000000000
<b>01</b> 101110001000100	1	0000000000100010	000000000000010 <b>1</b>	0000000000000100	0000000000000000
<b>00</b> 0010100101010 <b>1</b>	10	0000000000100010	0000000000000101	0000000000000100	0000000000000000
<b>10</b> 10101111111100	1	0000000000100010	0000000000000101	000000000000010 <b>1</b>	0000000000000000
<b>00</b> 01011100110 <b>111</b>	1000	000000000010 <b>1010</b>	0000000000000101	0000000000000101	0000000000000000
<b>11</b> 10010000 <b>111111</b>	1000000	0000000000101010	0000000000000101	0000000000000101	000000000 <b>1000000</b>
<b>10</b> 1011001111110 <b>1</b>	10	0000000000101010	0000000000000101	00000000000001 <b>11</b>	0000000001000000
<b>00</b> 01110100110100	1	000000000010101 <b>1</b>	0000000000000101	0000000000000111	
$r(sketch[ \ ]) $		00000000001010 <b><u>11</u></b>	0000000000000010 <b><u>1</u></b>	00000000000000 <b><u>111</u></b>	0000000001000000
		2	1	3	0

# Probabilistic counting with stochastic averaging in Java

```
public static long estimate(Iterable<Long> stream, int M)
{
    long[] sketch = new long[M];
    for (long x : stream)
    {
        int k = hash2(x, M);
        sketch[k] = sketch[k] | R(x);
    }
    int sum = 0;
    for (int k = 0; k < M; k++)
        sum += r(sketch[k]);
    double mean = 1.0 * sum / M;
    return (int) (M * Math.pow(2, mean)/.77351);
}
```

## Idea. *Stochastic averaging*

- Use second hash to split into  $M = 2^m$  independent streams
- Use PC on each stream, yielding  $2^m$  sketches .
- Compute *mean* = average # trailing 1 bits in the sketches.
- Return  $2^{\text{mean}}/.77351$ .

## **Flajolet-Martin 1983**

Q. Accuracy improves as M increases.

Q. How much?

**Theorem** (paraphrased to fit context of this talk).

*Under the uniform hashing assumption, PCSA*

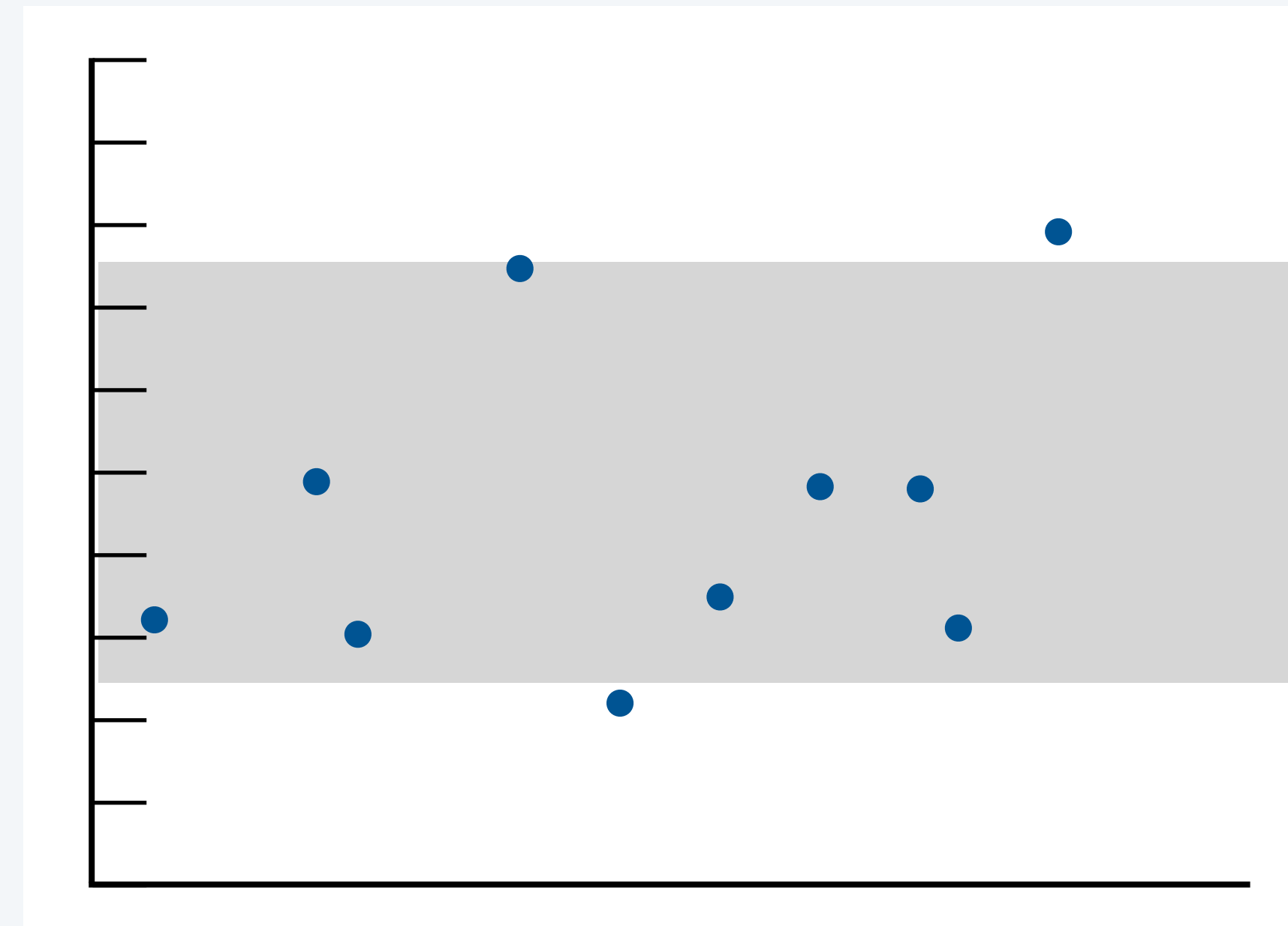
- *Uses 64M bits.*
- *Produces estimate with a relative accuracy close to  $0.78/\sqrt{M}$ .*

# Validation of PCSA analysis

**Hypothesis.** Value returned is accurate to  $0.78/\sqrt{M}$  *for random values from a large range.*

**Experiment.** 1,000,000 31-bit random values,  $M = 1024$  (10 trials)

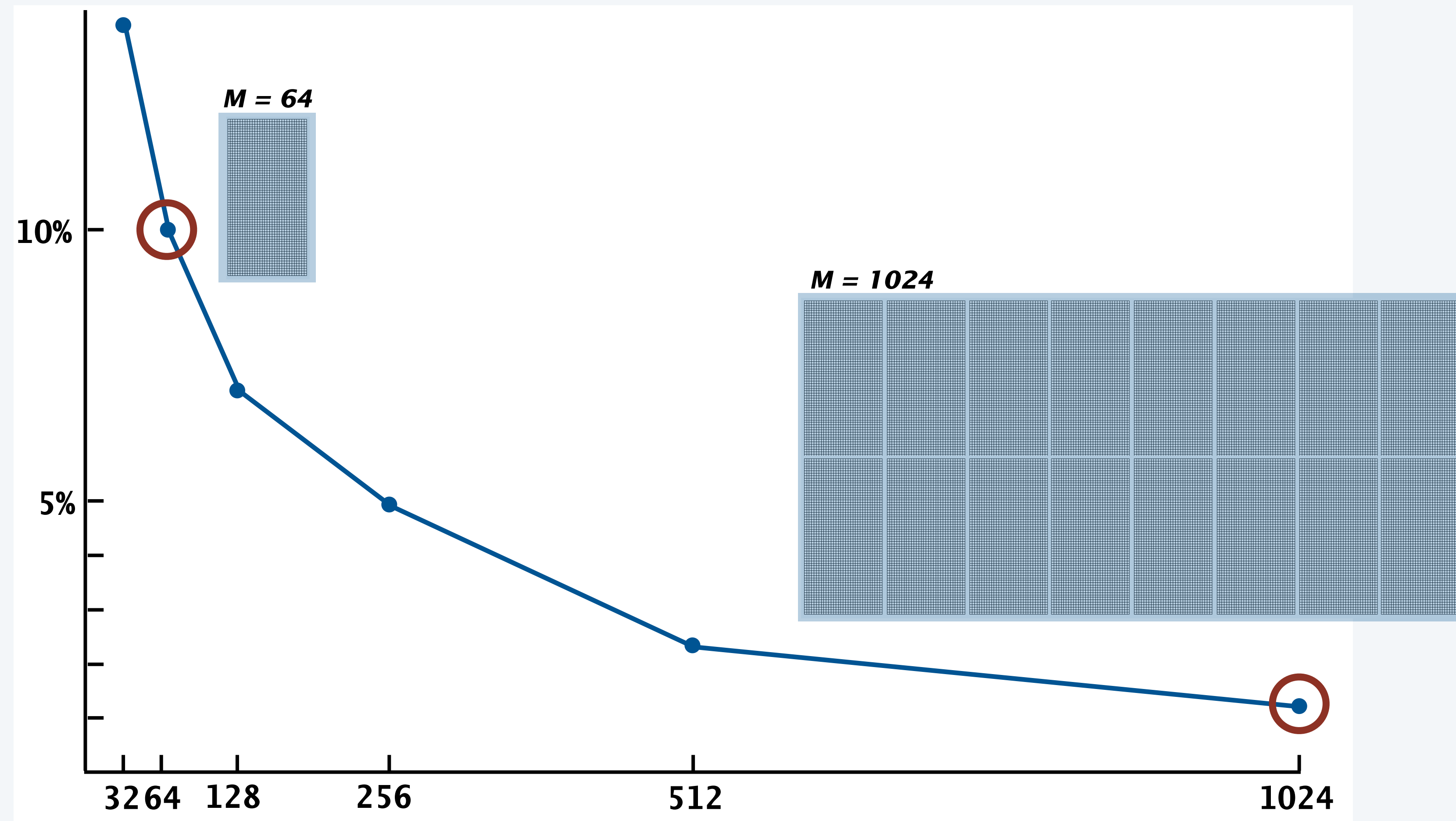
```
% java PCSA 1000000 31 1024 10
964416
997616
959857
1024303
972940
985534
998291
996266
959208
1015329
```





# Space-accuracy tradeoff for probabilistic counting with stochastic averaging

Relative accuracy:  $\frac{0.78}{\sqrt{M}}$



## Bottom line.

- Attain 10% relative accuracy with a sketch consisting of 64 words.
- Attain 2.4% relative accuracy with a sketch consisting of 1024 words.

# Scientific validation of PCSA

**Hypothesis.** Accuracy is as specified *for the hash functions we use and the data we have.*

**Validation (Flajolet and Martin, 1985).** Extensive reproducible scientific experiments (!)

**Validation (RS, this morning).**

```
% java PCSA 6000000 1024 < log.07.f3.txt
1106474
```

**<1% larger than actual value**

***log.07.f3.txt***

```
109.108.229.102
pool-71-104-94-246.lsanca.dsl-w.verizon.net
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
```

**Q.** Is PCSA effective?

**A.** ABSOLUTELY!

# Summary: PCSA (Flajolet-Martin, 1983)

is a *demonstrably* effective approach to cardinality estimation

Q. *About* how many different values are present in a given stream?

## PCSA

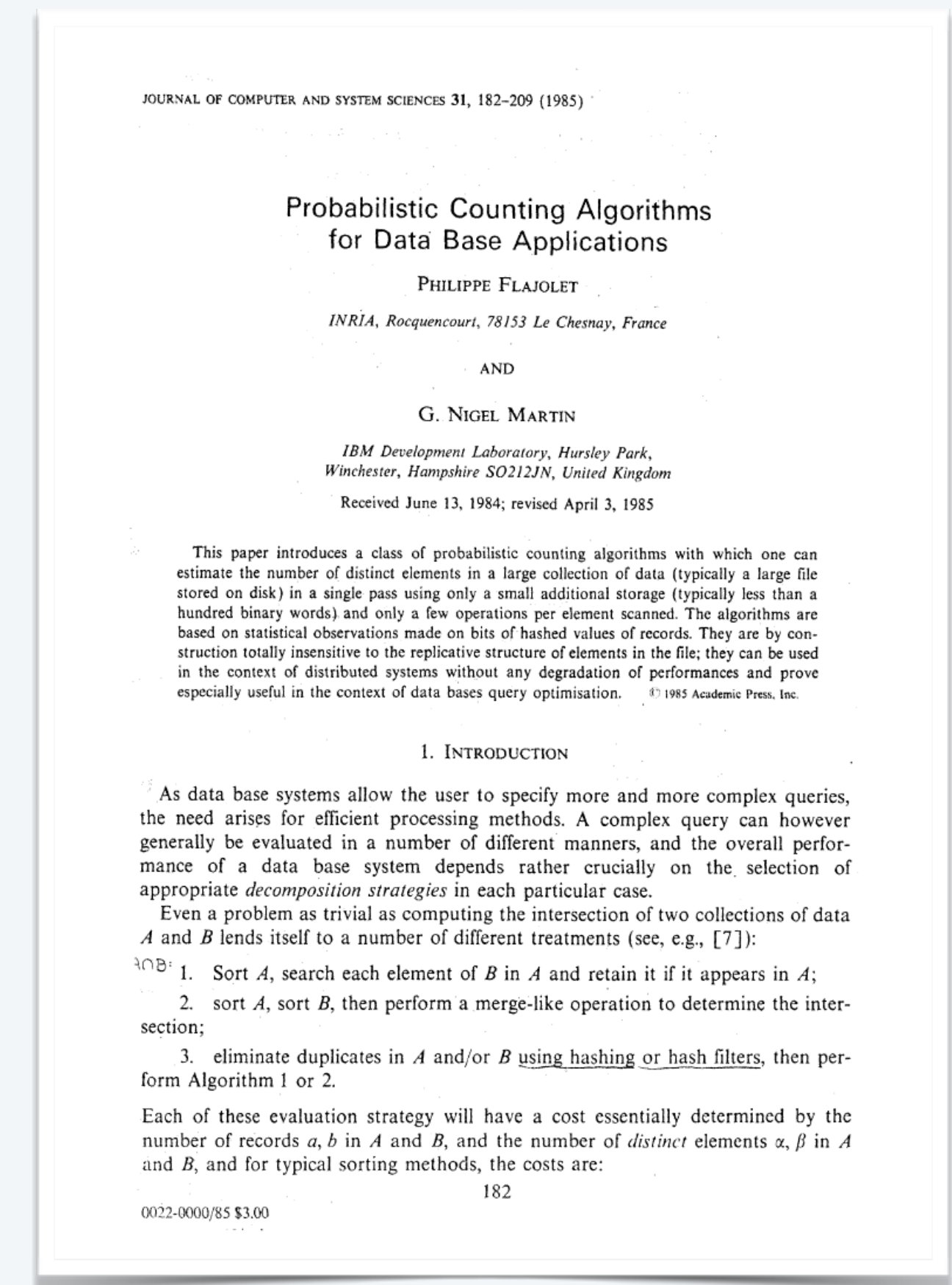
- Makes *one pass* through the stream.
- Uses *a few machine instructions per value*
- Uses *M* words to achieve relative accuracy  $0.78/\sqrt{M}$



Results validated through extensive experimentation.

## Open questions

- Better space-accuracy tradeoffs?
- Support other operations?



A poster child for AofA/AC

*"IT IS QUITE CLEAR that other observable regularities on hashed values of records could have been used..."*

— Flajolet and Martin



# Small sample of work on related problems

---

1970	Bloom	set membership
1984	Wegman	unbiased sampling estimate
1996–	many authors	refinements (stay tuned)
2000	Indyk	L1 norm
2004	Cormode– Muthukrishnan	frequency estimation deletion and other operations
2005	Giroire	fast stream processing
2012	Lumbroso	full range, asymptotically unbiased
2014	Helmi–Lumbroso– Martinez–Viola	uses neither sampling nor hashing



# Cardinality Estimation

- Rules of the game
- Probabilistic counting
- Stochastic averaging
- **Refinements**
- Final frontier

OF



# We can do better (in theory)

## Alon, Matias, and Szegedy

*The Space Complexity of Approximating the Frequency Moments*  
*STOC 1996; JCSS 1999.*

### Contributions

- Studied problem of estimating higher moments
- Formalized idea of *randomized* streaming algorithms
- Won Gödel Prize in 2005 for “foundational contribution”

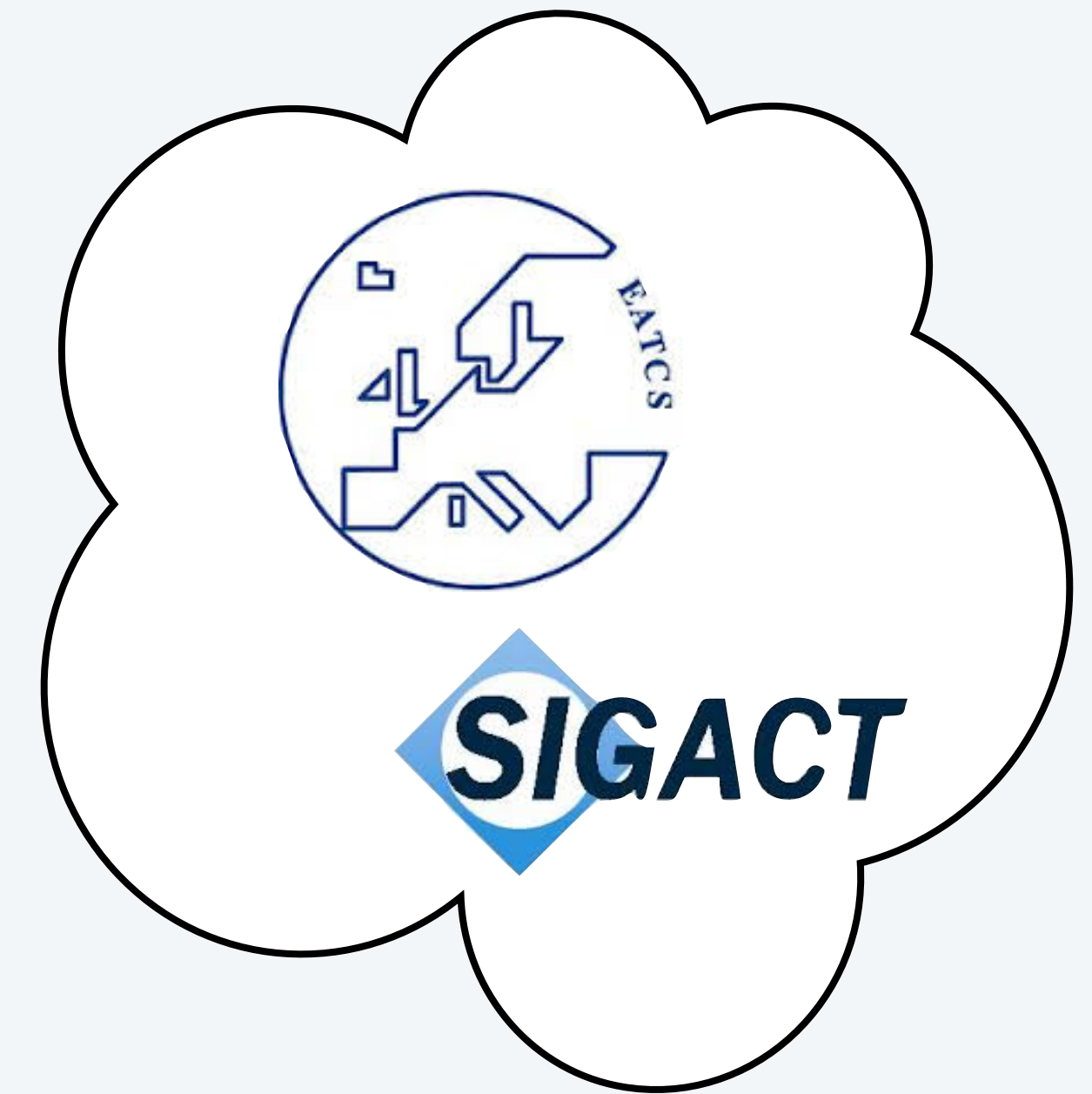
**Theorem** (paraphrased to fit context of this talk).

*With strongly universal hashing, PC, for any  $c > 2$ ,*

- *Uses  $O(\log N)$  bits.*
- *Is accurate to a factor of  $c$ , with probability at least  $2/c$ .*

BUT, no impact on cardinality estimation in practice

- “Algorithm” just changes hash function for PC
- Accuracy estimate is too weak to be useful
- No validation



Replaces “uniform hashing” assumption  
with “random bit existence” assumption

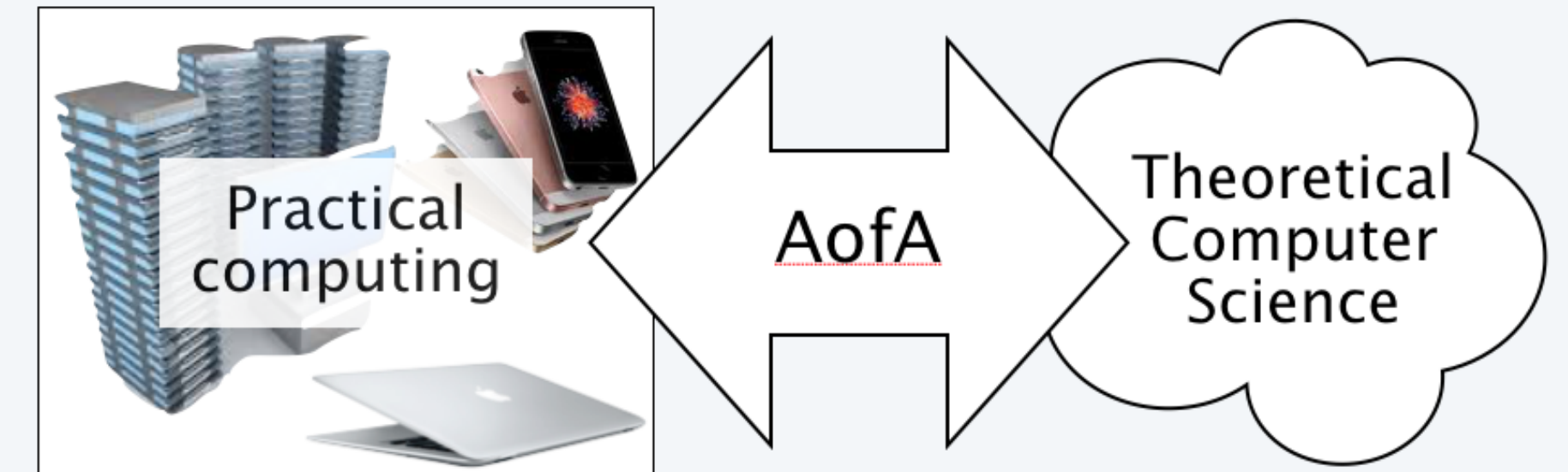




## Interesting quote

*“ Flajolet and Martin [assume] that one may use in the algorithm an explicit family of hash functions which exhibits some ideal random properties. Since we are not aware of the existence of such a family of hash functions ...”*

*– Alon, Matias, and Szegedy*



No! They hypothesized that practical hash functions would be as effective as random ones.  
***They then validated that hypothesis by proving tight bounds that match experimental results.***

### Points of view re *hashing*

- **Theoretical computer science.** Uniform hashing assumption is not proved.
- **Practical computing.** Hashing works for many common data types.
- **AofA.** *Extensive experiments have validated precise analytic models.*

### Points of view re *random bits*

- **Theoretical computer science.** Axiomatic that random bits exist.
- **Practical computing.** No, they don't ! And randomized algorithms are inconvenient, btw.
- **AofA.** *More effective path forward is to validate precise analysis even if stronger assumptions are needed.*

# logs and loglogs

---

To improve space-time tradeoffs, we need to *carefully count bits*.

## Relevant quantities

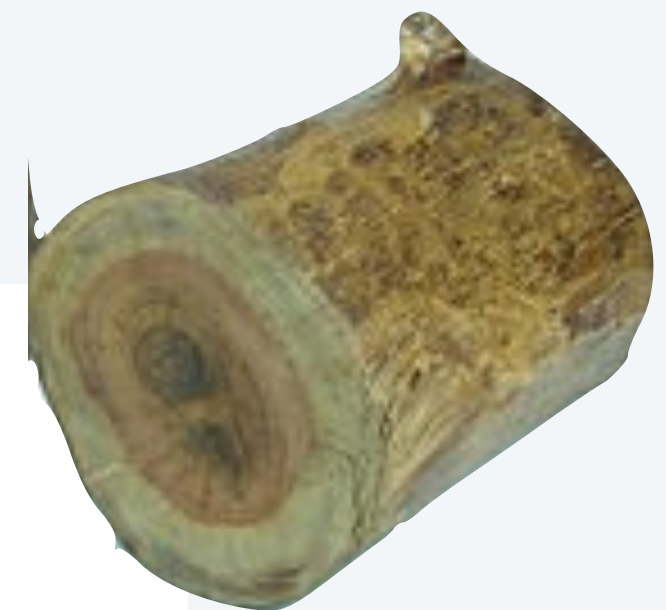
- $N$  is the number of items in the data stream.
- $\lg N$  is the number of bits needed to represent numbers less than  $N$  in binary.
- $\lg \lg N$  is the number of bits needed to represent numbers less than  $\lg N$  in binary.

## For real-world applications

- $N$  is less than  $2^{64}$ .
- $\lg N$  is less than 64.
- $\lg \lg N$  is less than 8.

## Typical PCSA implementations

- Could use  $M \lg N$  bits, in theory.
- Use 64-bit words to take advantage of machine-language efficiencies.
- Use (therefore)  $64 * \mathbf{64} = 4096$  bits with  $M = 64$  (for 10% accuracy with  $N < 2^{64}$ ).



# We can do better (in theory)

**Bar-Yossef, Jayram, Kumar, Sivakumar, and Trevisan**

*Counting Distinct Elements in a Data Stream*  
*RANDOM 2002.*

## Contribution

Improves space-accuracy tradeoff at extra stream-processing expense.



**Theorem** (paraphrased to fit context of this talk).

*With strongly universal hashing, there exists an algorithm that*

- *Uses  $O(M \log \log N)$  bits.*  $\longleftarrow$  *PCSA uses  $M \lg N$  bits*
- *Achieves relative accuracy  $O(1/\sqrt{M})$ .*

STILL no impact on cardinality estimation in practice

- Infeasible because of high stream-processing expense.
- Big constants hidden in O-notation
- No validation

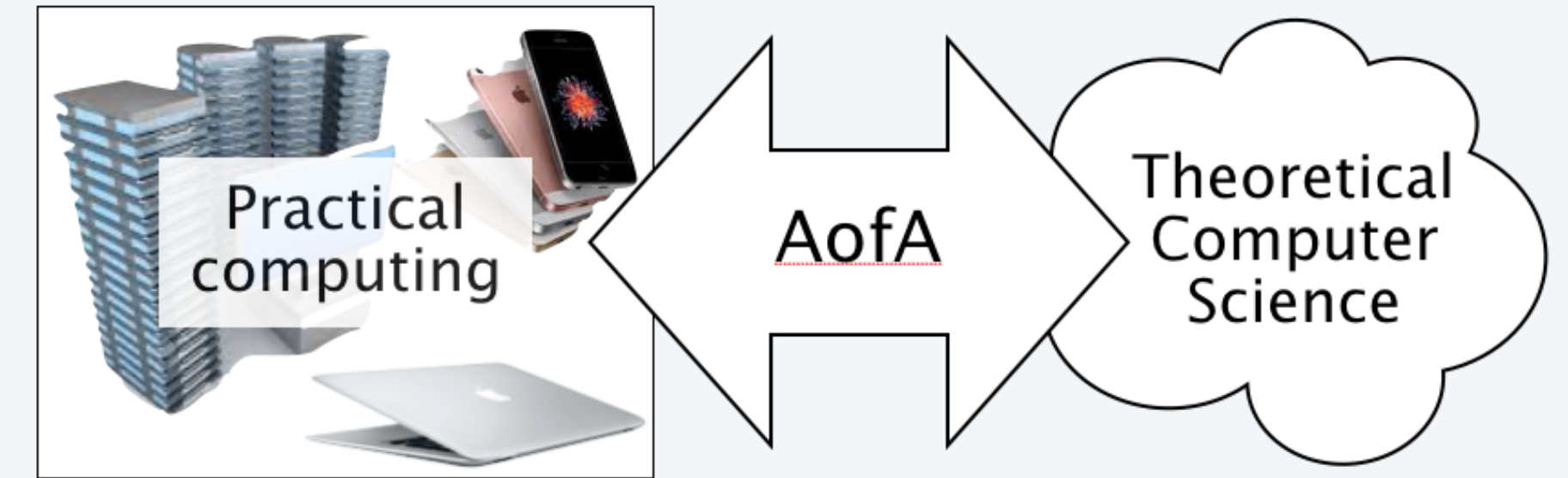




# We can do better (in theory *and* in practice)

# Durand and Flajolet

*LogLog Counting of Large Cardinalities*  
*ESA 2003; LNCS volume 2832.*



## Contributions (independent of BYJKST)

- Presents **LogLog** algorithm, an easy variant of PCSA
- Improves space-accuracy tradeoff *without* extra expense per value
- Full analysis, fully validated with experimentation

**Theorem** (paraphrased to fit context of this talk).

*Under the uniform hashing assumption, **LogLog***

- Uses  $M \lg \lg N$  bits.
- Achieves relative accuracy close to  $1.30/\sqrt{M}$ .

*PCSA saves sketches (lg N bits each)*

00000000000000000000000000000000**110111**

*LogLog saves  $r()$  values ( $\lg \lg N$  bits each)*

00100 ( = 4)

I like PCSA

Not much impact on cardinality estimation in practice *only because*

- PCSA was effectively deployed in practical systems
- Idea led to a better algorithm a few years later (stay tuned)



# We can do better (in theory and in practice): HyperLogLog algorithm (2007)

```
public static long estimate(Iterable<Long> stream, int M)
{
    int[] bytes = new int[M];
    for (long x : stream)
    {
        int k = hash2(x, M);
        if (bytes[k] < Bits.r(x)) bytes[k] = Bits.r(x);
    }
    double sum = 0.0;
    for (int k = 0; k < M; k++)
        sum += Math.pow(2, -1.0 - bytes[k]);
    return (int) (alpha * M * M / sum);
}
```

*8-bit bytes (code to pack into M loglogN bits omitted)*

*about .709 for M = 64*

**Flajolet-Fusy-Gandouet-Meunier 2007**

**Idea.** *Harmonic mean of  $r()$  values*

- Use stochastic splitting
- Keep track of  $\min(r(x))$  for each stream
- Return *harmonic mean*.

Flajolet, Fusy, Gandouet, and Meunier  
*HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm*  
AofA 2007; DMTCS 2007.

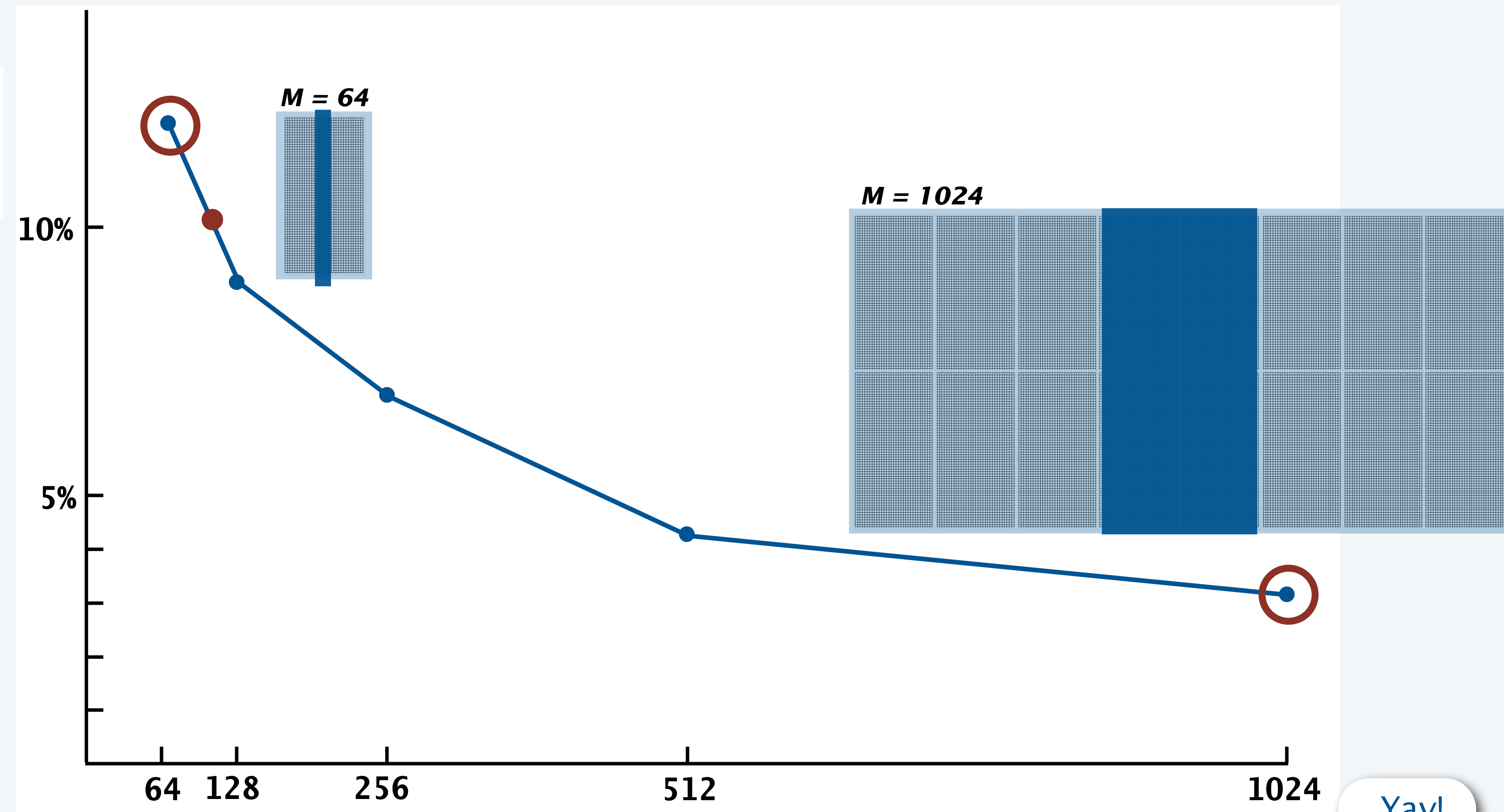
**Theorem** (paraphrased to fit context of this talk).

*Under the uniform hashing assumption, **HyperLogLog***

- *Uses  $M \log \log N$  bits.*
- *Achieves relative accuracy close to  $1.02/\sqrt{M}$ .*

# Space-accuracy tradeoff for HyperLogLog

Relative accuracy:  $\frac{1.02}{\sqrt{M}}$



Bottom line (for  $N < 2^{64}$ ).

- Attain 10% relative accuracy with a sketch consisting of  $108 \times 6 = 648$  bits.
- Attain 3.1% relative accuracy with a sketch consisting of  $1024 \times 6 = 6144$  bits.





# PCSA vs Hyperloglog

---



## Typical PCSA implementations

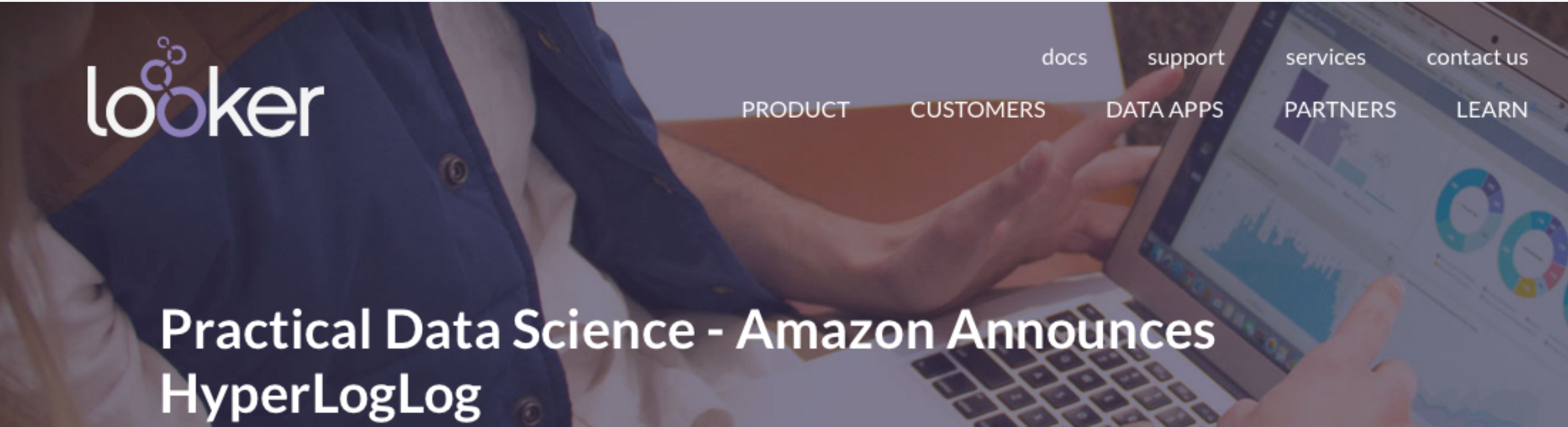
- Could use  $M \lg N$  bits, in theory.
- Use 64-bit words to take advantage of machine-language efficiencies.
- Use (therefore)  $64 * \mathbf{64} = 4096$  bits with  $M = 64$  (for 10% accuracy with  $N < 2^{64}$ ).



## Typical Hyperloglog implementations

- Could use  $M \lg \lg N$  bits, in theory.
- Use 8-bit bytes to take advantage of machine-language efficiencies.
- Use (therefore)  $108 * \mathbf{8} = 864$  bits with  $M = 108$  (for 10% accuracy with  $N < 2^{64}$ ).

# Validation of Hyperloglog



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.*



Philippe Flajolet, mathematician, **data scientist**, and computer scientist extraordinaire



# Cardinality Estimation

- Rules of the game
- Probabilistic counting
- Stochastic averaging
- Refinements
- **Final frontier**



# We can do a bit better (in theory) *but not much better*

## Indyk and Woodruff

*Tight Lower Bounds for the Distinct Elements Problem, FOCS 2003.*

**Theorem** (paraphrased to fit context of this talk).

*Any algorithm that achieves relative accuracy  $O(1/\sqrt{M})$  must use  $\Omega(M)$  bits*

*loglogN improvement possible* 

Upper bound

Lower bound

## Kane, Nelson, and Woodruff

*Optimal Algorithm for the Distinct Elements Problem, PODS 2010.*

**Theorem** (paraphrased to fit context of this talk).

*With strongly universal hashing there exists an algorithm that*

- *Uses  $O(M)$  bits.*
  - *Achieves relative accuracy  $O(1/\sqrt{M})$ .*
- optimal* 

Unlikely to have impact on cardinality estimation in practice

- Tough to beat HyperLogLog's low stream-processing expense.
- Constants hidden in O-notation not likely to be  $< 6$
- No validation

Theoretical  
Computer  
Science

# Can we beat HyperLogLog in practice?

Necessary characteristics of a better algorithm

- Makes *one pass* through the stream.
- Uses *a few dozen machine instructions per value*
- Uses *a few hundred bits*
- Achieves 10% relative accuracy or better

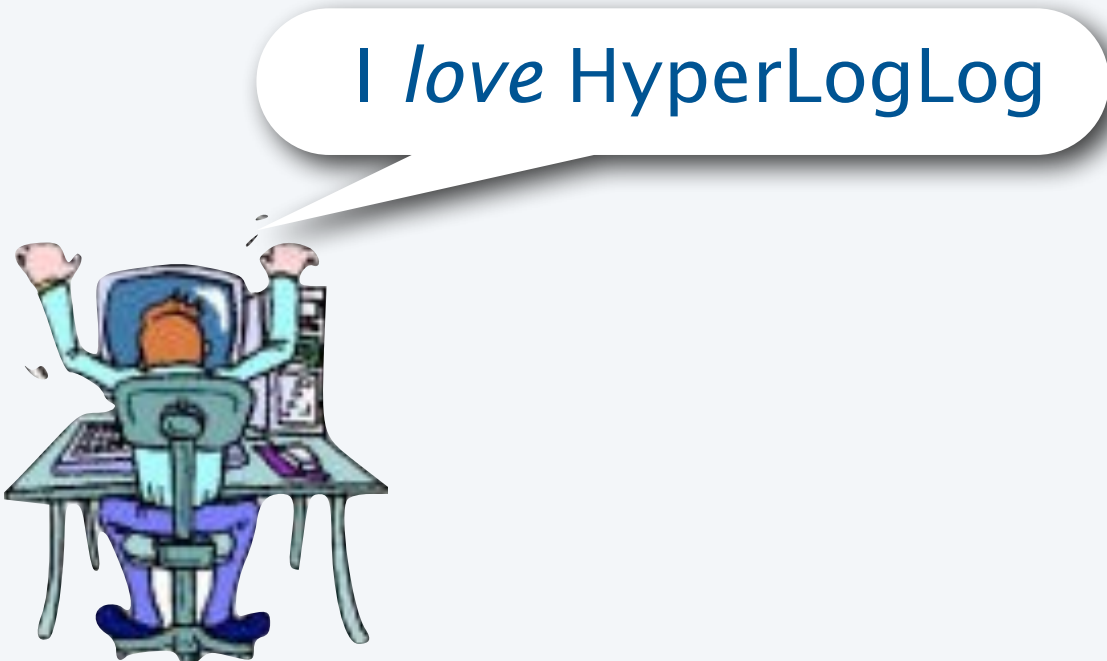


*“I’ve long thought that there should be a simple algorithm that uses a small constant times  $M$  bits...”*

– Jérémie Lumbroso

	<i>machine instructions per stream element</i>	<i>memory bound</i>	<i>memory bound when <math>N &lt; 2^{64}</math></i>	<i># bits for 10% accuracy when <math>N &lt; 2^{64}</math></i>
<b>HyperLogLog</b>	20–30	$M \log \log N$	$6M$	648
<b>BetterAlgorithm</b>	<i>a few dozen</i>			<i>a few hundred</i>

Also, results need to be validated through extensive experimentation.



# A proposal: HyperBitBit (Sedgewick, 2016)

```
public static long estimate(Iterable<String> stream, int M)
{
    int lgN = 5;
    long sketch = 0;
    long sketch2 = 0;
    for (String x : stream)
    {
        long x = hash(s);
        int k = hash2(x, 64);
        if (r(x) > lgN) sketch = sketch | (1L << k);
        if (r(x) > lgN + 1) sketch2 = sketch2 | (1L << k);
        if (p(sketch) > 31)
        { sketch = sketch2; lgN++; sketch2 = 0; }
    }
    return (int) (Math.pow(2, lgN + 5.4 + p(sketch)/32.0));
}
```

*bias factor (determined empirically)*

## Idea.

- $\lg N$  is estimate of  $\lg N$
- sketch is 64 indicators whether to increment  $\lg N$
- sketch2 is 64 indicators whether to increment  $\lg N$  *by 2*
- Update when half the bits in sketch are 1
- correct with  $p(\text{sketch})$  *and bias factor*

Q. Does this even work?



# Initial experiments

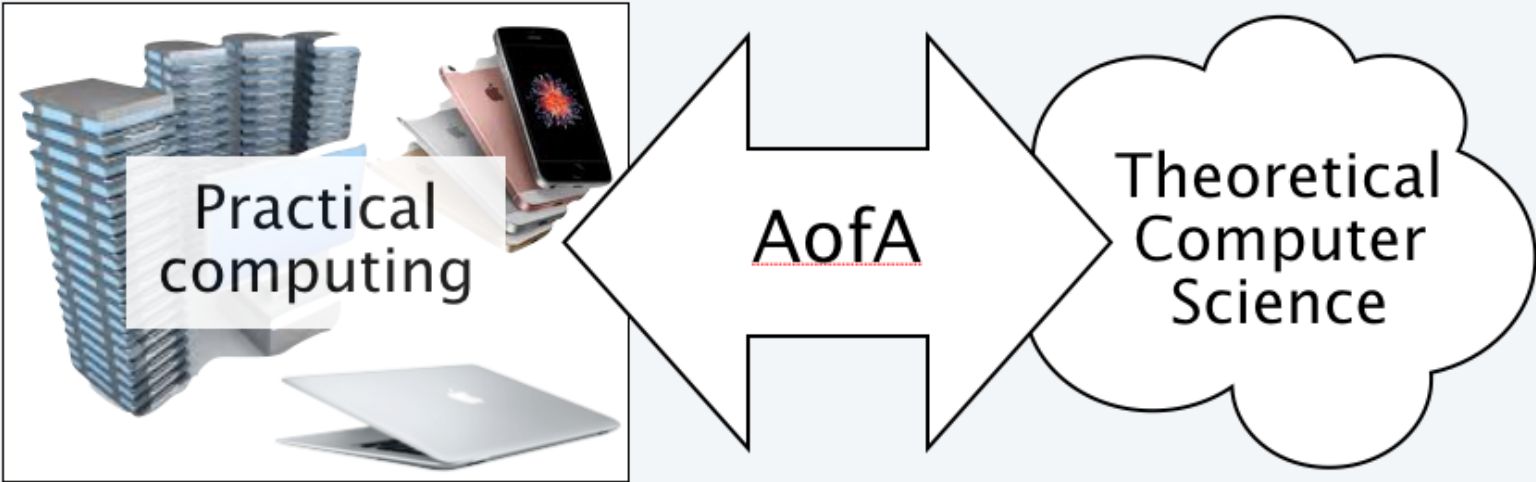
## Exact values for web log example

```
% java Hash 1000000 < log.07.f3.txt
242601
% java Hash 2000000 < log.07.f3.txt
483477
% java Hash 4000000 < log.07.f3.txt
883071
% java Hash 6000000 < log.07.f3.txt
1097944
```

## HyperBitBit estimates

```
% java HyperBitBit 1000000 < log.07.f3.txt
234219
% java HyperBitBit 2000000 < log.07.f3.txt
499889
% java HyperBitBit 4000000 < log.07.f3.txt
916801
% java HyperBitBit 6000000 < log.07.f3.txt
1044043
```

	1,000,000	2,000,000	4,000,000	6,000,000
Exact	242,601	483,477	883,071	1,097,944
HyperBitBit	234,219	499,889	916,801	1,044,043
ratio	1.05	1.03	0.96	1.03



## Next steps.

- Analyze.
- Experiment.
- Iterate

**Conjecture.** On practical data, **HyperBitBit**, for  $N < 2^{64}$ ,

- Uses  $128 + 6$  bits.
- Estimates cardinality within 10% of the actual.

# Summary/timeline for cardinality estimation



			<i>hashing assumption</i>	<i>feasible and validated?</i>	<i>memory bound (bits)</i>	<i>relative accuracy constant</i>	<i># bits for 10% accuracy when <math>N &lt; 2^{64}</math></i>
1970	Bloom	Bloom filter	<i>uniform</i>	✓	$kN$		$> 2^{64}$
1985	Flajolet-Martin	<b>PCSA</b>	<i>uniform</i>	✓	$M \log N$	0.78	4096
1996	Alon-Matias-Szegedy	[ <i>theorem</i> ]	<i>strong universal</i>	✗	$O(M \log N)$	$O(1)$	?
2002	Bar-Yossef-Jayram- Kumar-Sivakumar- Trevisan	[ <i>theorem</i> ]	<i>strong universal</i>	✗	$O(M \log \log N)$	$O(1)$	?
2003	Durand-Flajolet	<b>LogLog</b>	<i>uniform</i>	✓	$M \lg \lg N$	1.30	1536
2007	Flajolet-Fusy- Gandouet-Meunier	<b>HyperLogLog</b>	<i>uniform</i>	✓	$M \lg \lg N$	1.04	648
2010	Kane-Nelson- Woodruff	[ <i>theorem</i> ]	<i>strong universal</i>	✗	$O(M) + \lg \lg N$	$O(1)$	?
2018+	RS-?	<b>HyperBitBit</b>	<i>uniform</i>	✓ (?)	$2M + \lg \lg N$	?	134 (?)

# Happy Birthday, Don!

---



## **ALGORITHMS COMBINATORICS INFORMATION**

**COLLOQUIUM FOR  
DON KNUTH'S  
80TH BIRTHDAY**





# Cardinality Estimation

**Robert Sedgewick**  
**Princeton University**

with special thanks to Jérémie Lumbroso