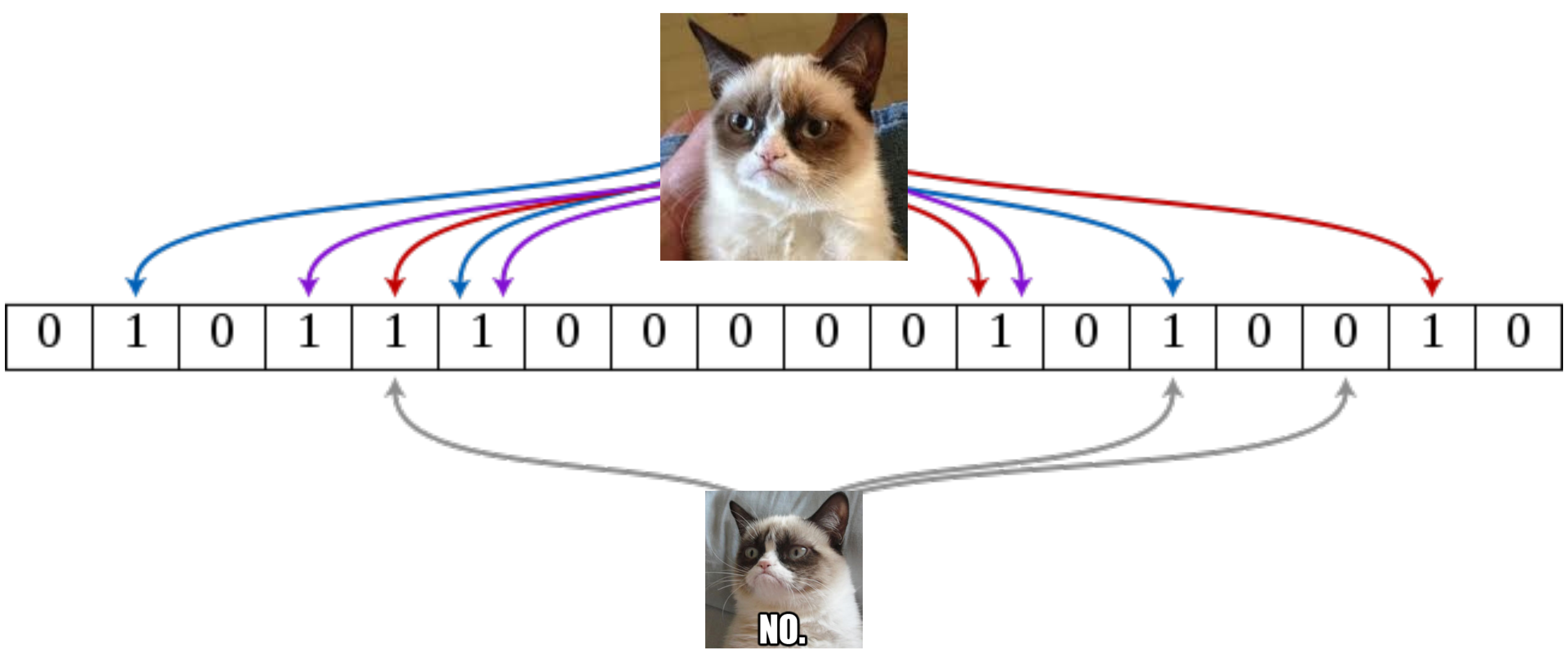
Research Students' Lecture Series 2015

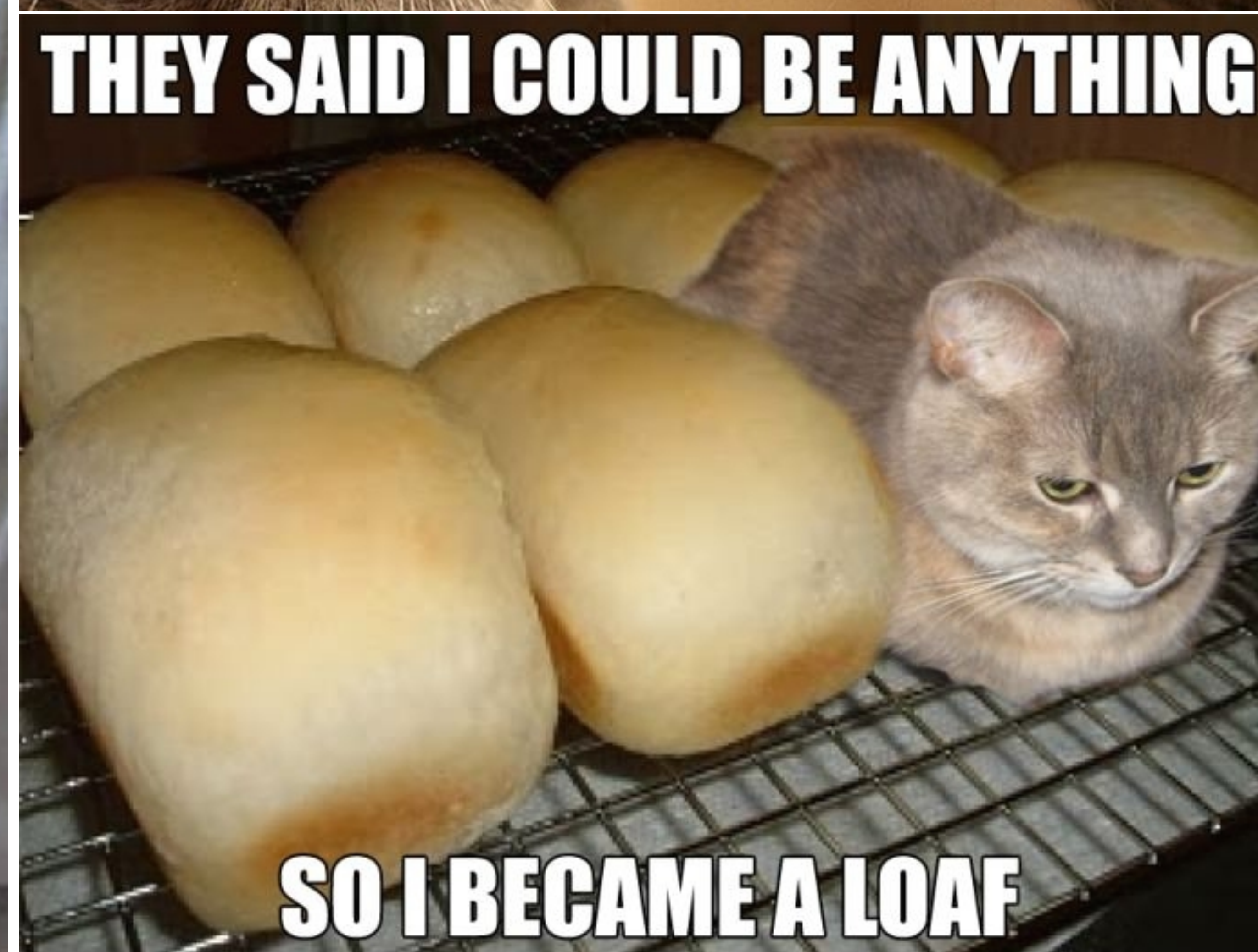
Analyse your big data with this one weird probabilistic approach!

Or: applied probabilistic algorithms in 5 easy pieces



Advait Sarkar
advait.sarkar@cl.cam.ac.uk
Research Students' Lecture Series Michaelmas 2015


"Owning a cat may increase the chance of suicide."
globalpost
GOOD


THEY SAID I COULD BE ANYTHING
SO I BECAME A LOAF

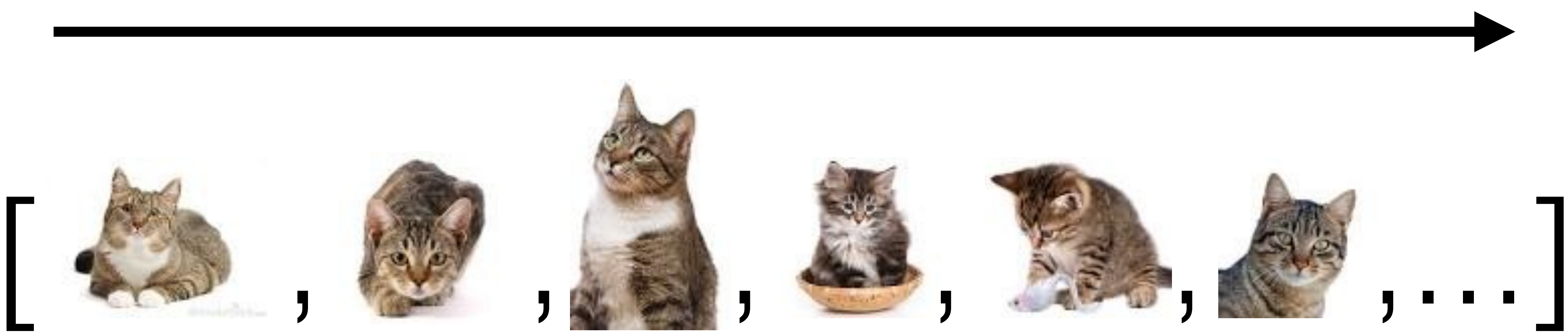
Problem

- MildlyInappropriateCatAppreciationSociety.com receives thousands of cat pictures a minute
- Strict quality control requirements mean that repeated submissions are frowned upon
- How do we quickly detect if a picture already exists in the database?

One might say we are looking for *duplicats*.

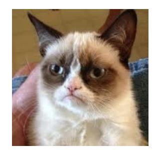
Solution 1: use the catabase

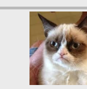
- Equivalent to time complexity of lookup operation
- E.g. O(log(n)) for binary searching a sorted list



- Problem: slow
- Problem: large data transfer overhead

Solution 2: hash table

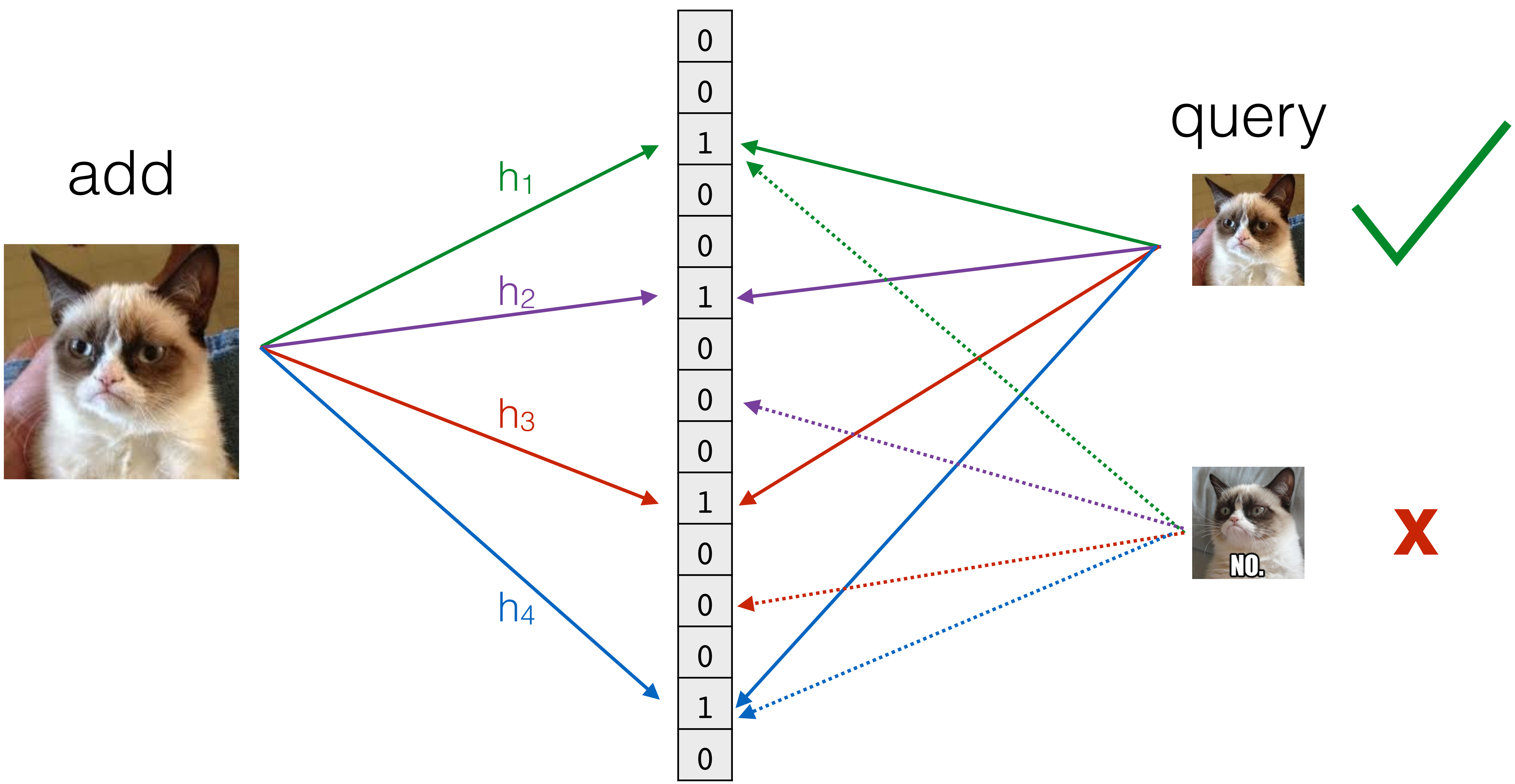
h()

00000000	0
00000001	0
…	…
01100101	
…	…
11111111	0

- O(1) lookup
- Still 2^n space for n-bit hash function
- Exact solution — can we do better?

Bloom filter

- An extension of the hash table idea
- Multiple hashes index into a bit vector



Bloom filter properties

- P(false positive) = $\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$
  m-bit vector, *k* hashes, *n* elements

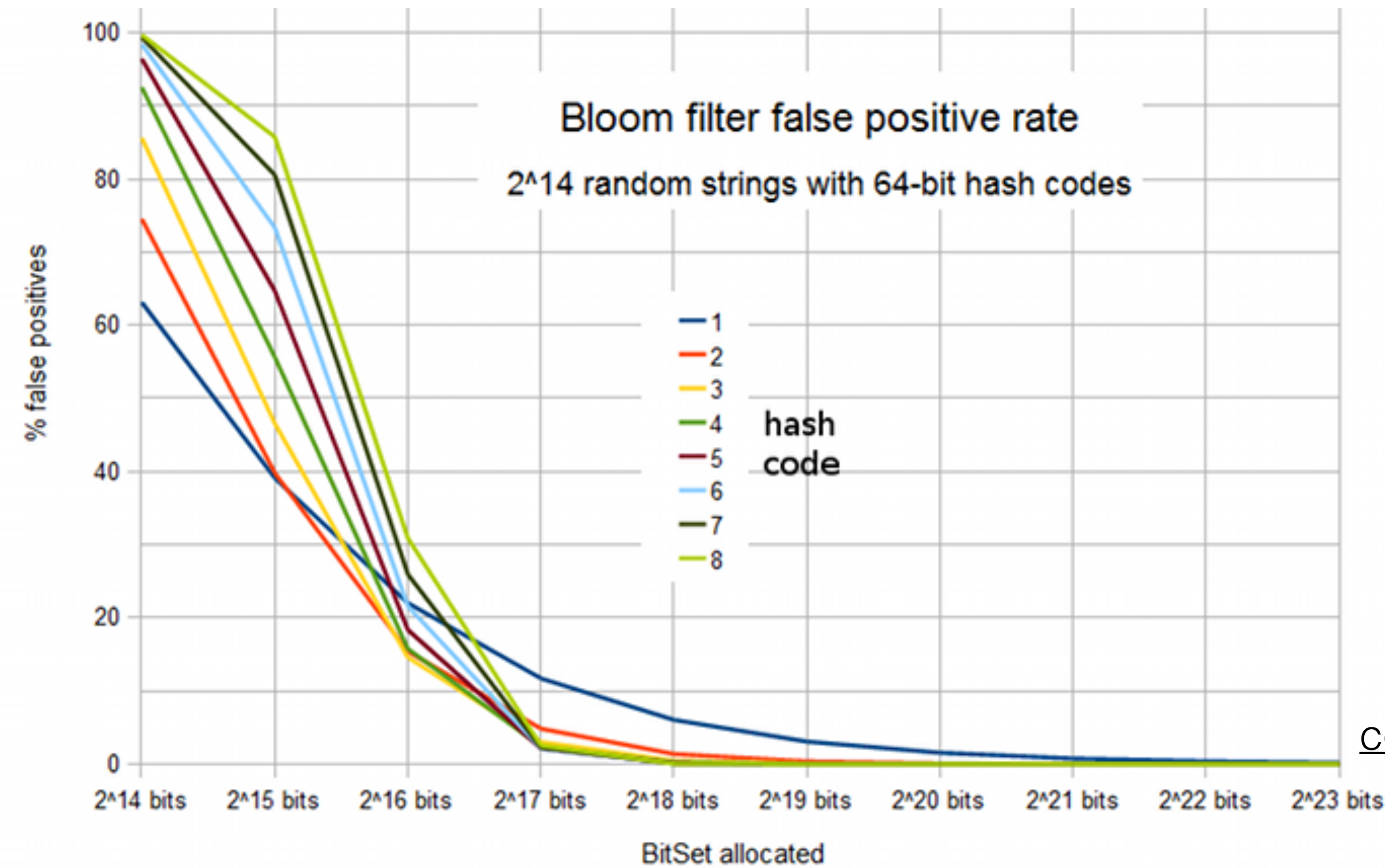


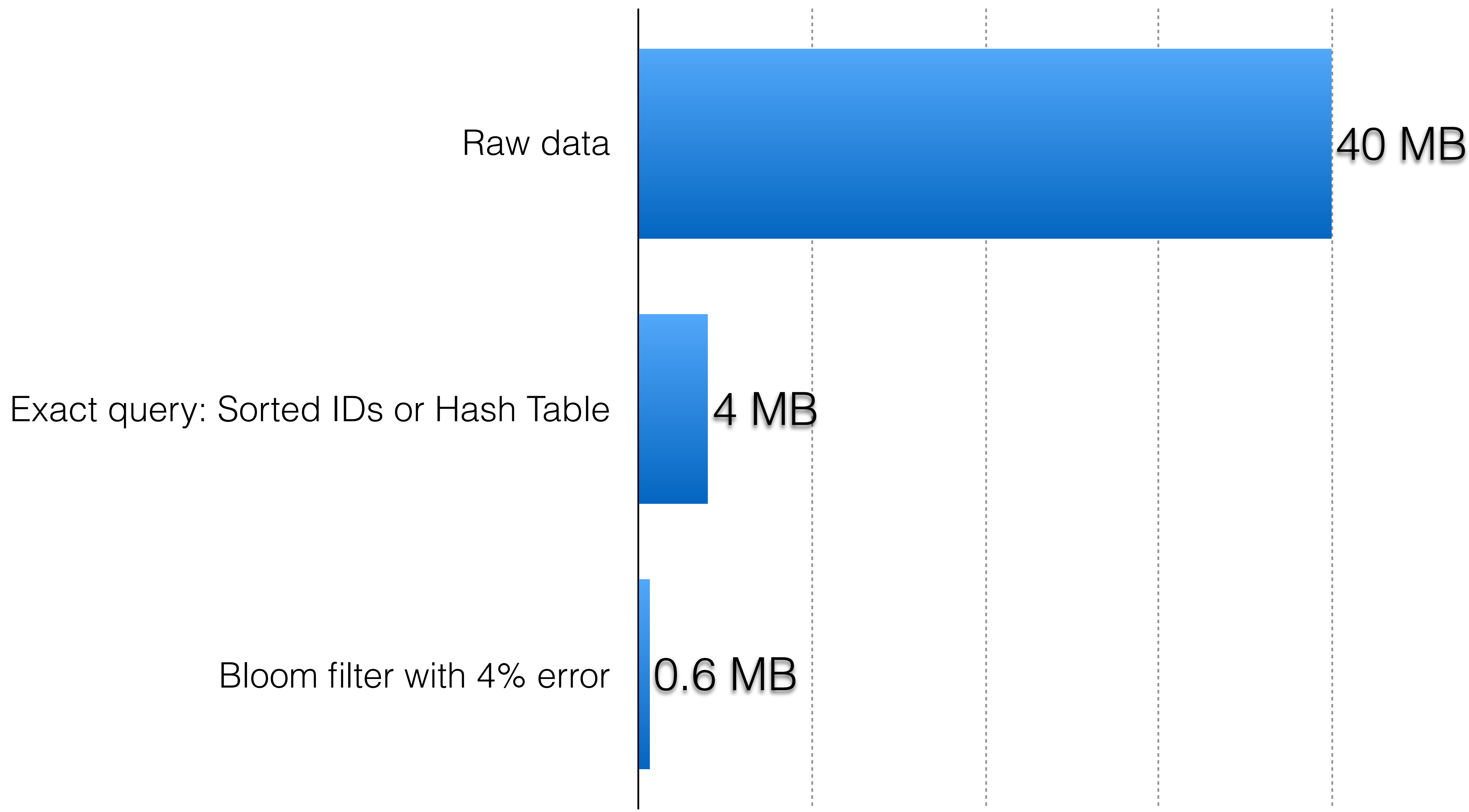
image credit: http://www.javamex.com/tutorials/collections/bloom_filter_false_positives_graph.shtml

- no false negatives

- can be *much* smaller & faster than hash table

Cardinality estimation / membership query

Dataset of 32-bit integers with 10^7 elements, 10^6 distinct values



Quiz!

index:	1	2	3	4	5	6	7	8	9	10	11	12
	1	1	0	1	0	0	0	1	1	0	0	1

This bloom filter contains a list of English words.

The (rather terrible) hashes are:

- h1 = number of letters in the word
- h2 = number of vowels in the word

Is the word "`ailurophilia`" present in the list?

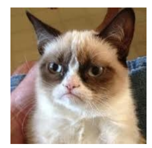
Fed up with the tyranny of MildlyInappropriateCatAppreciationSociety.com, rebels set up their own website which allows reposts.

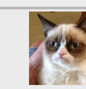
The website grows rapidly in popularity, but MICAS claims that their website still has more unique content.

Problem:

How can the rebels keep a fast, live count of their distinct cats and prove MICAS wrong?

Solution 1: hash table

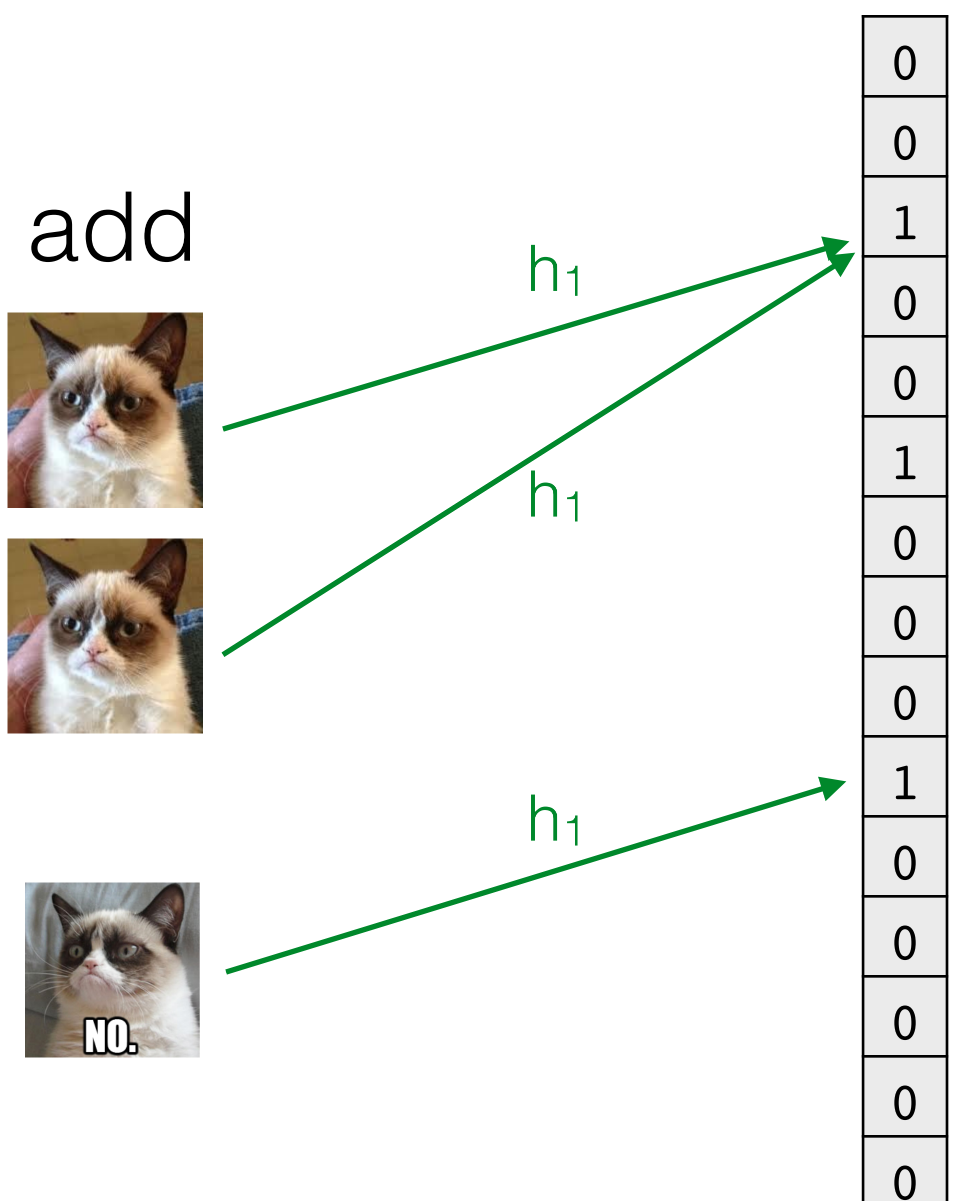
h()

00000000	0
00000001	0
…	…
01100101	 1
…	…
11111111	0

- 2^n space for n-bit hash function
- Exact solution — can we do better?

Linear counter

- An extension of the hash table idea
- A single hash indexes into a bit vector



Max likelihood cardinality estimate is given by

$$\hat{n} = -m \ln \left(\frac{\#\ \text{zero entries}}{m} \right)$$

Linear counter properties

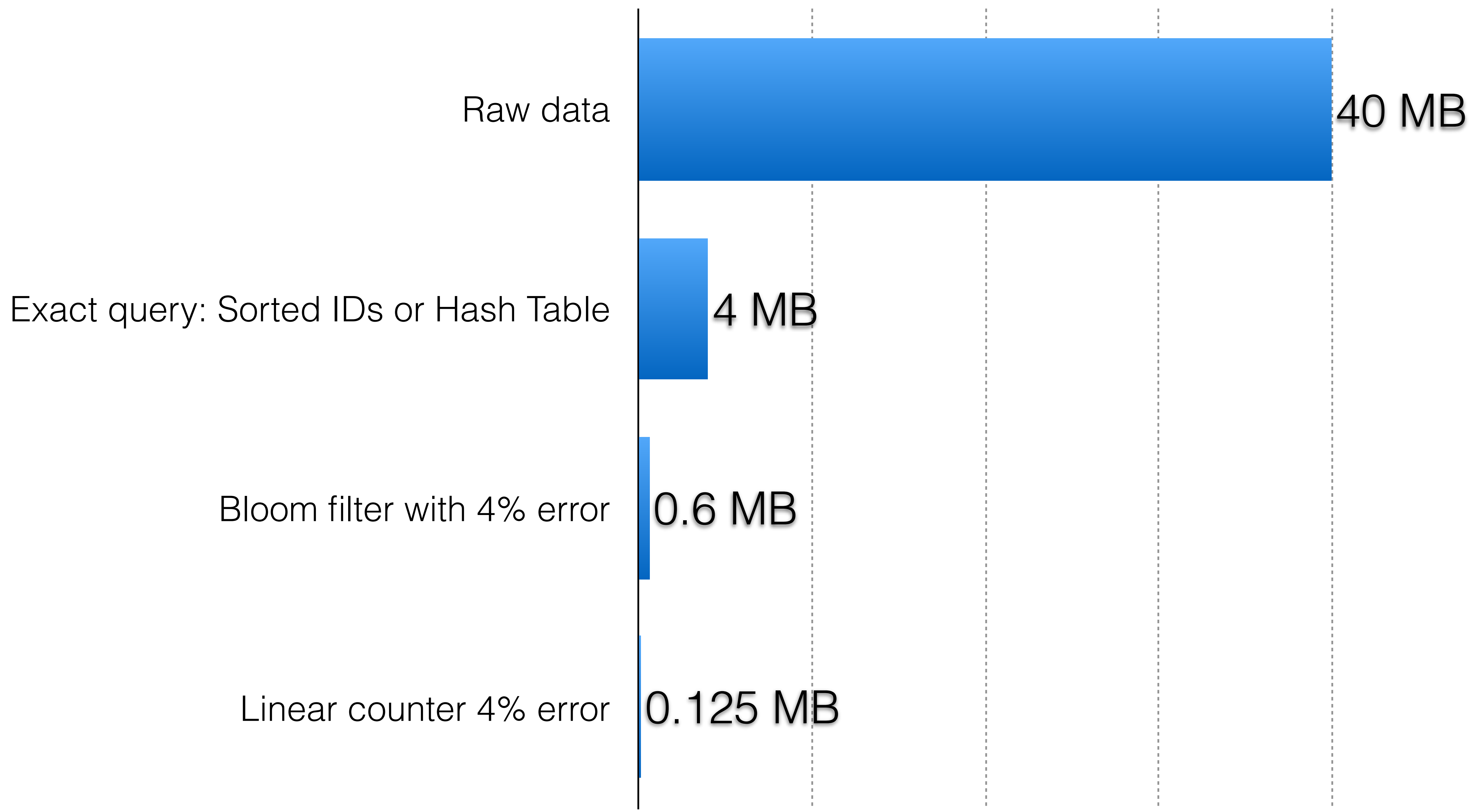
$$m > \frac{e^t - t - 1}{(\epsilon t)^2}$$

m bits limits standard error to **ϵ**
t = n/m, where n is true cardinality

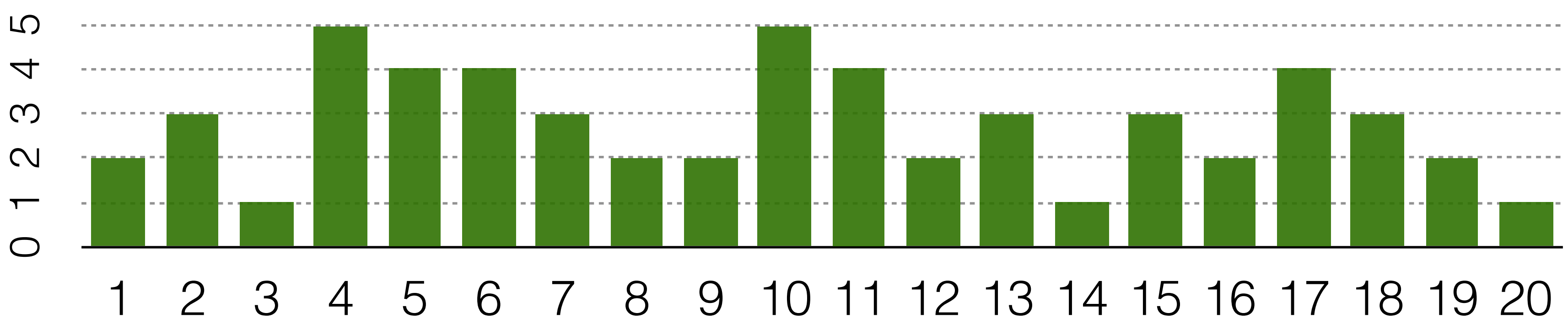
- this can be much smaller than a hash table!
- easy to merge distributed counters
- can we do better?

Cardinality estimation

Dataset of 32-bit integers with 10^7 elements, 10^6 distinct values



A different approach



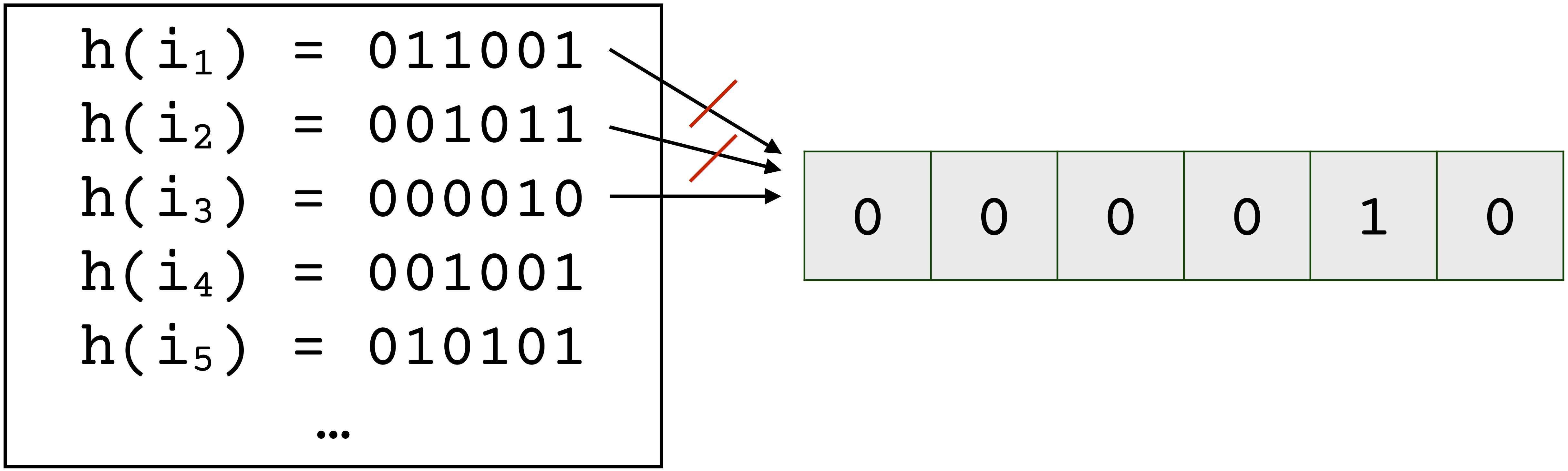
20÷1 = 20

- Want to count distinct integers
- If integers are uniformly distributed, can estimate cardinality as max÷min
- Super cheap, super fast, super rough

LogLog counting

- A well-designed hash function can transform any dataset into a uniformly distributed one
- Some estimators work better than others
- In particular, sequences of leading 0s
- If k is the maximum number of leading 0s observed so far, 2^k is a reasonable cardinality estimate

LogLog counter



- Hash each new item as it comes in
- If the hash has more leading zeros (higher k) than current maximum, store it
- Compute cardinality as 2^k

LogLog counting with Stochastic Averaging

- Use *m* “buckets”, hash each item into a single bucket, and average *k* for a better estimate:

  $\alpha m 2^{\bar{k}}$ (α ≅0.8, reduces bias)

- Get better estimate by removing outliers

LogLog counter properties

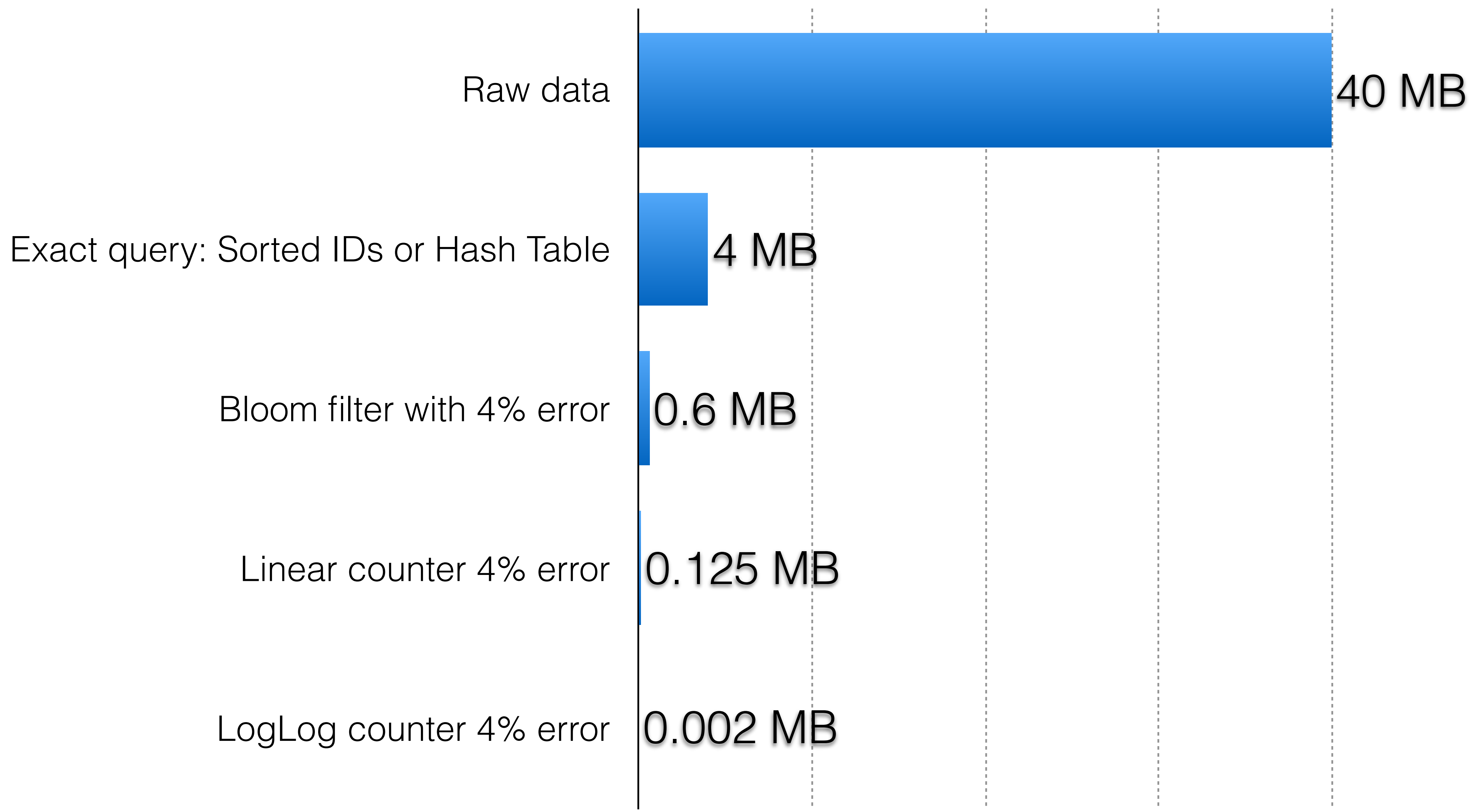
$$\epsilon \approx \frac{1.3}{\sqrt{m}}$$

ϵ is standard error
m buckets, each at least size log(log(*n*))
n is true cardinality

- Very small, reasonably accurate
- Inherently parallelisable

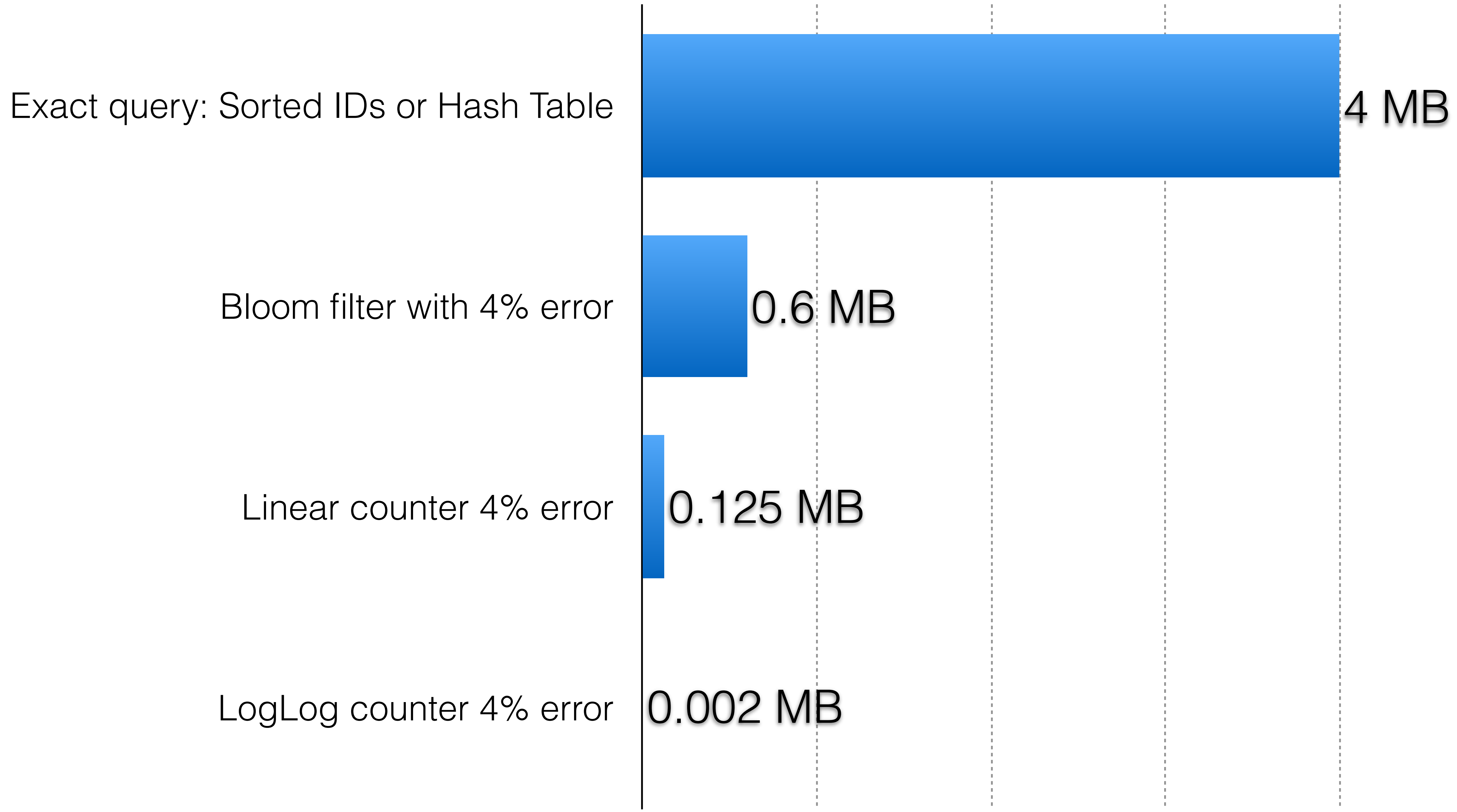
Cardinality estimation

Dataset of 32-bit integers with 10^7 elements, 10^6 distinct values



Cardinality estimation

Dataset of 32-bit integers with 10^7 elements, 10^6 distinct values



Cardinality estimation

Dataset of 32-bit integers with 10^7 elements, 10^6 distinct values

Quiz!

bucket 1	0	0	1	0	1	1
bucket 2	0	0	0	0	1	0

- Approximately how many distinct elements have been recorded by this LogLog counter?
- Use the formula: $m \times 2^{(\text{average } k)}$

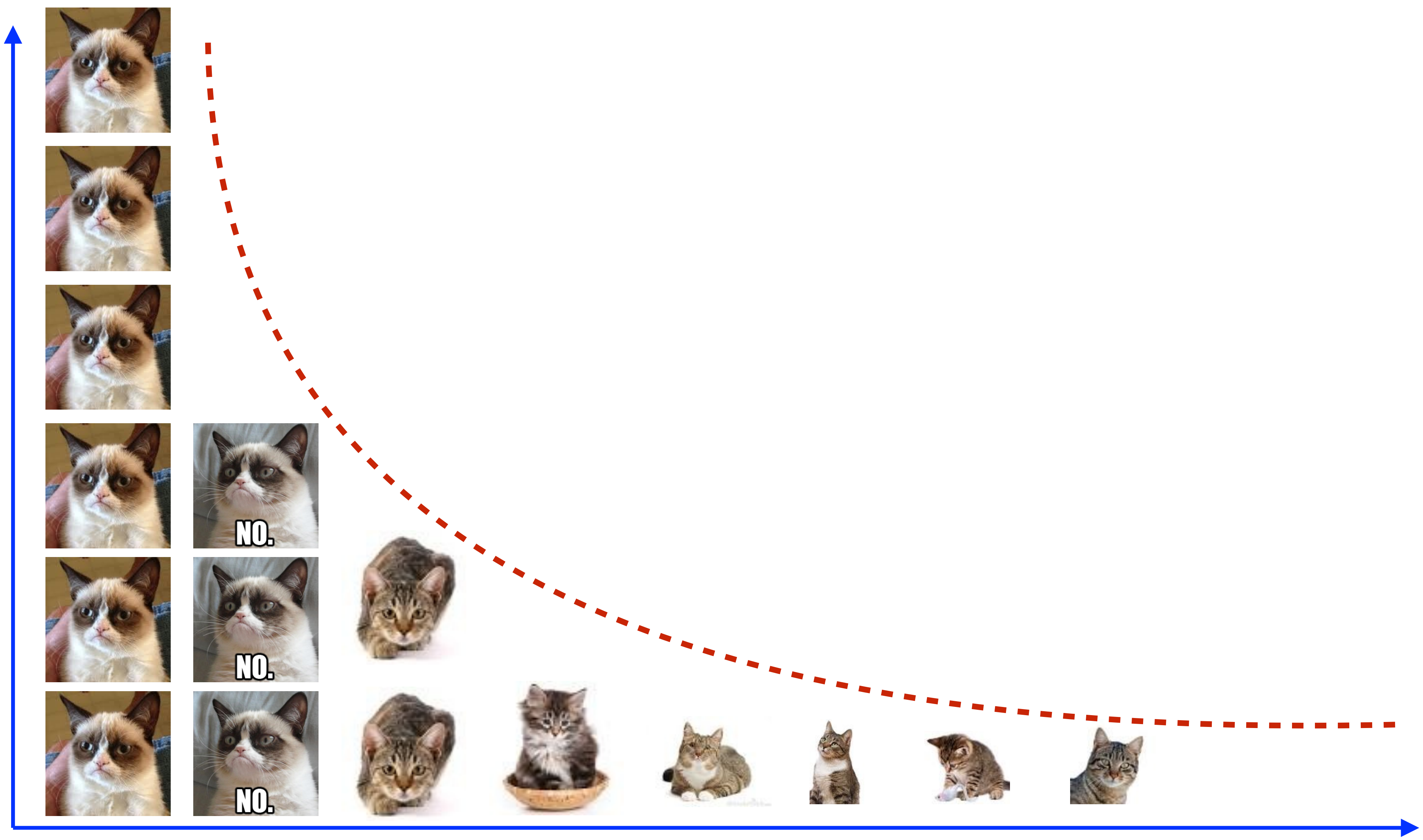
  (m = buckets, k = number of leading 0s)

The rebel website can now quickly establish whether they have more unique cats.

However, they would now like to know which cats are rapidly growing in popularity.

Problem:
How can the rebels keep a fast, live count of the frequencies of their cats?

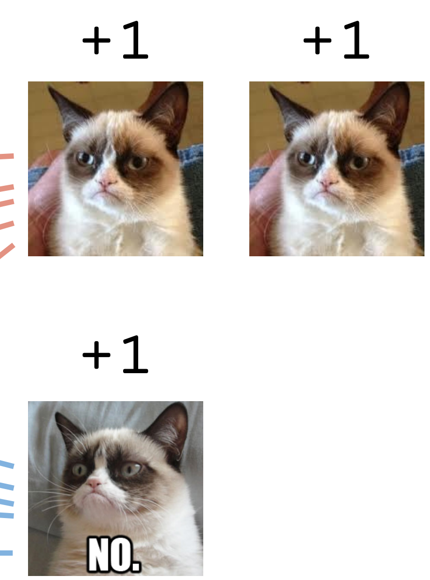
One might say we are trying to graph the *long tail* of cats.



Count-min sketch

- “Sketch” because it is an approximate summary
- use vectors of **integer** counters that are themselves hash tables.

hash 1	0	**1**	0	0	0	0	0	0	**2**	0
hash 2	0	0	**1**	0	0	0	**2**	0	0	0
hash 3	0	0	0	0	**2**	**1**	0	0	0	0
hash 4	0	0	0	0	**3**	0	0	0	0	0
hash 5	0	0	0	0	0	0	0	0	**2**	**1**



- Frequency of an item is upper-bounded by the smallest of any of its counters

Count-min sketch properties

$$\varepsilon \leq \frac{2n}{w}$$

$$\text{with probability } \delta = 1 - \left(\frac{1}{2}\right)^d$$

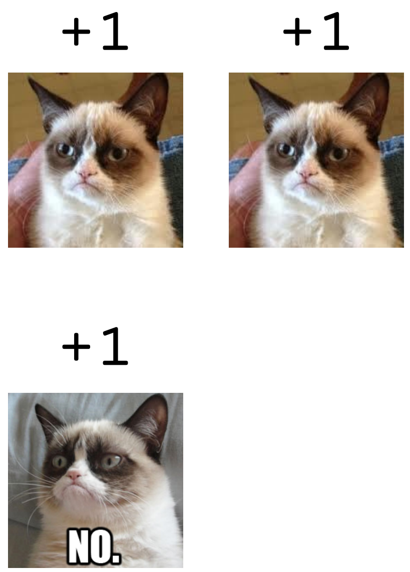
- ε = absolute overestimation (underestimation is impossible)
- For *n* total entries, *d* hashes, each *w* elements wide.

Frequency estimation is a core component of powerful machine learning techniques, e.g. bayes nets, naive bayes, HMMs, etc.

Can we do better?

Count-mean-min sketch

hash 1	0	**1**	0	0	0	0	0	0	**2**	0
hash 2	0	0	**1**	0	0	0	**2**	0	0	0
hash 3	0	0	0	0	**2**	**1**	0	0	0	0
hash 4	0	0	0	0	**3**	0	0	0	0	0
hash 5	0	0	0	0	0	0	0	0	**2**	**1**

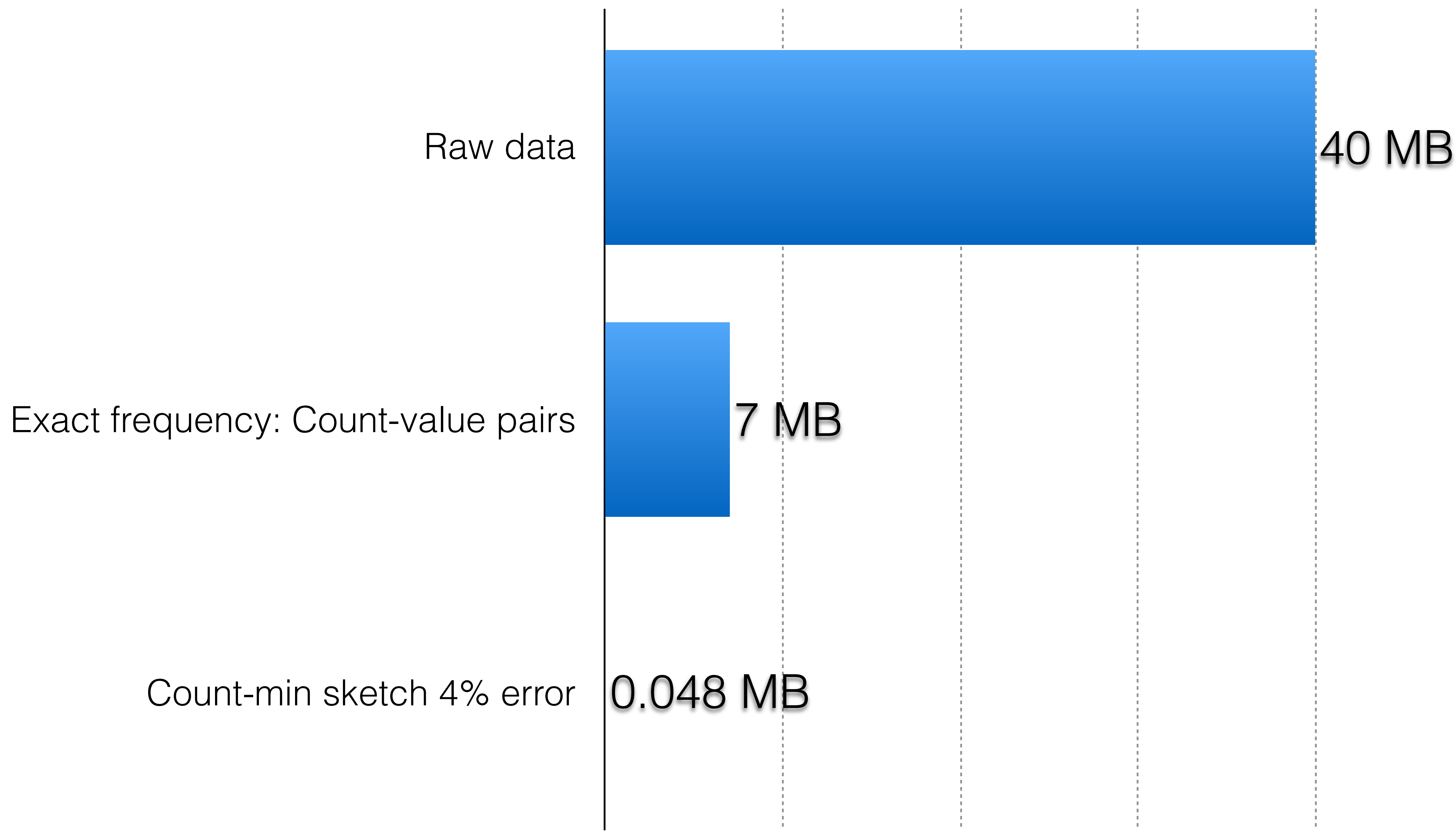


- Can attempt to compensate for collisions
- Estimate "noise" of each row as average of all counters *except* element of interest
- Subtract row noise from each counter
- Return *min*(median denoised counter, min counter)

Performs better in the face of many collisions.

Frequency estimation estimation

Dataset of 32-bit integers with 10^7 elements, 10^6 distinct values



Quiz!

index:	1	2	3	4	5	6	7	8	9	10	11	12
h1	3	0	2	1	0	1	0	3	1	2	0	1
h2	0	1	0	2	0	0	1	0	3	2	1	1

This is a **count-min** sketch of English words.

The (same terrible) hashes are:

- h1 = number of letters in the word
- h2 = number of vowels in the word (y is not a vowel)

At most how many times has "`kittylove`" appeared?

In summary

Technique	Purpose
Bloom filter	membership query
Linear counting	cardinality estimation
LogLog counting	
Count-min sketch	frequency estimation
Count-mean-min sketch	

There is a whole wide world of probabilistic techniques

- Skip lists are fast data structures with O(log n) insert, delete and lookup
- Stream-summary can record top-*k* frequent items in essentially constant space
- Reservoir sampling can pick *n* items out of an infinite stream, with each item having equal probability of being picked
- An array of count-min sketches can be used to calculate range queries (e.g. find all x such that $p < x < q$)
- And many more!