Data Structures and Algorithms

Bloom Filters 2

CS 225
Brad Solomon

November 3, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Extra Credit Project Submissions

~110 teams submitted extra credit projects.

Drafted TAs to do a first pass grading of some of the major topics

Each TA-graded project is graded by two TAs for fairness

Mentors will (hopefully) be assigned sometime next week

Quick announcements on MPs

MP_Traversal had the lowest plagiarism rate of any assignment!

MP_mazes is due next week

The next MP will NOT be released next Monday

Quick announcements on Exams

Next exam is next Monday

Look at topic list / do practice exam

Make sure you thoroughly understand the coding question.

Learning Objectives

Review conceptual understanding of bloom filter

Review probabilistic data structures and explore one-sided error

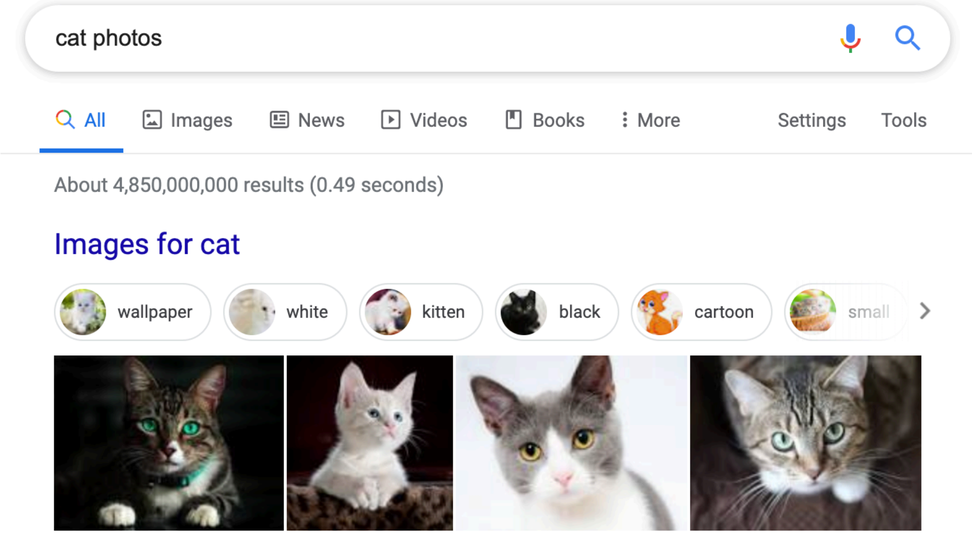
Formalize the math behind the bloom filter

Discuss bit vector operations and potential extensions to bloom filters

Memory-Constrained Data Structures

What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

Constrained by Big Data (Large N)



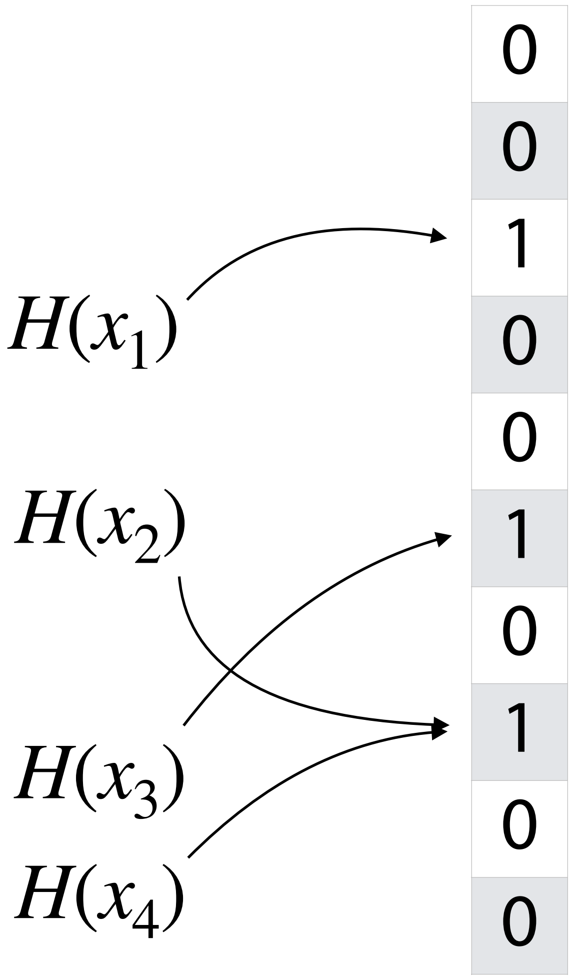
Google Index Estimate: >60 billion webpages

Google Universe Estimate (2013): >130 trillion webpages

Bloom Filter: Insertion

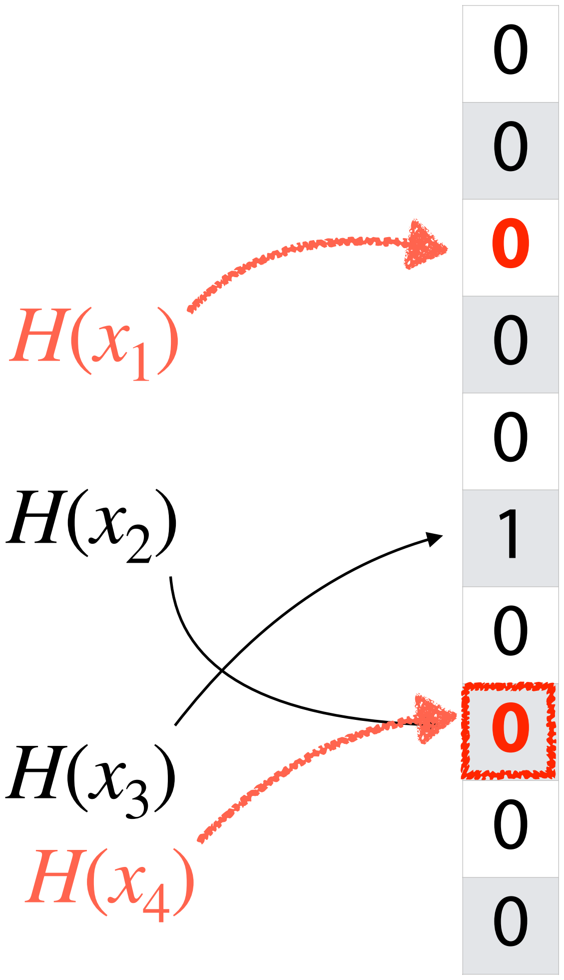
An item is inserted into a bloom filter by hashing and then setting the hash-valued bit to 1

If the bit was already one, it stays 1



Bloom Filter: Deletion

Due to hash collisions and lack of information, items cannot be deleted!



Bloom Filter: Search

S = { 16, 8, 4, 13, 29, 11, 22 }

h(k) = k % 7

0	0
1	1
2	1
3	0
4	1
5	0
6	1

```
_find(16)
```

```
_find(20)
```

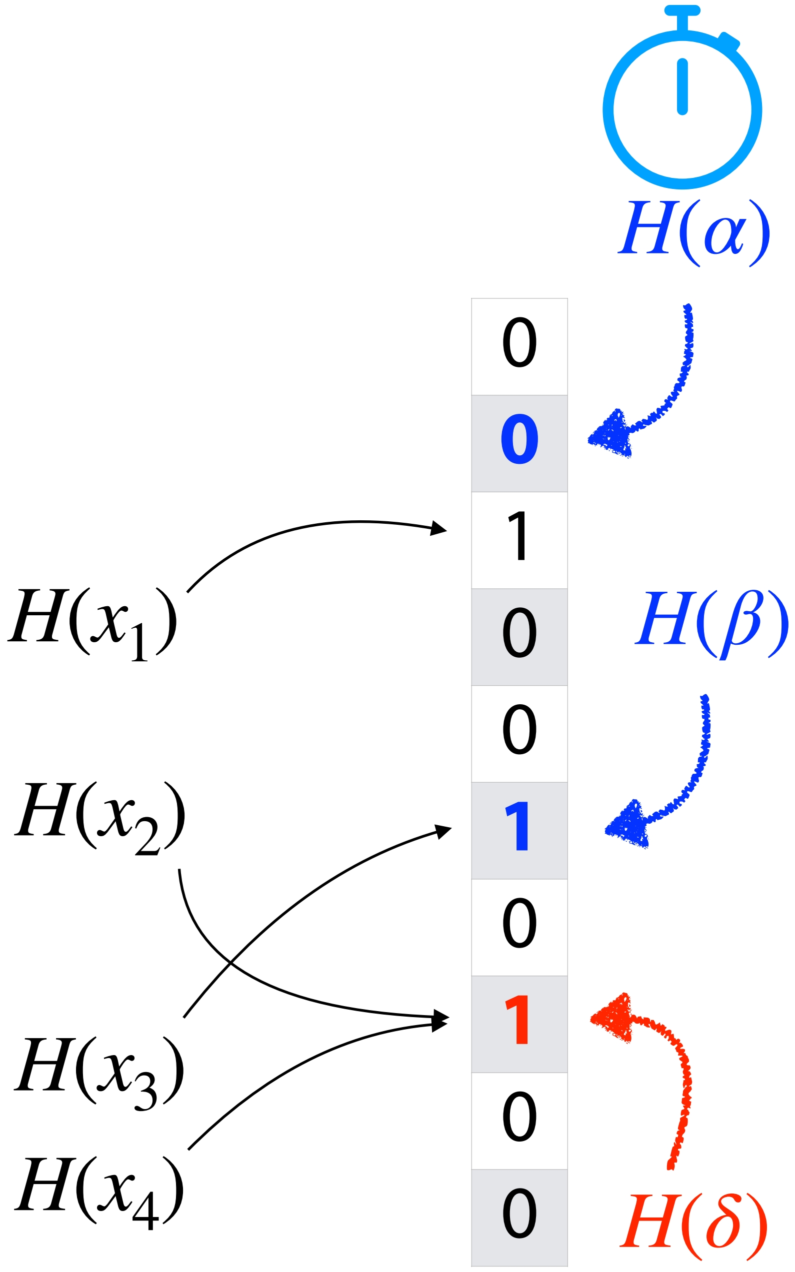
```
_find(3)
```

Bloom Filter: Search

The bloom filter is a *probabilistic* data structure!

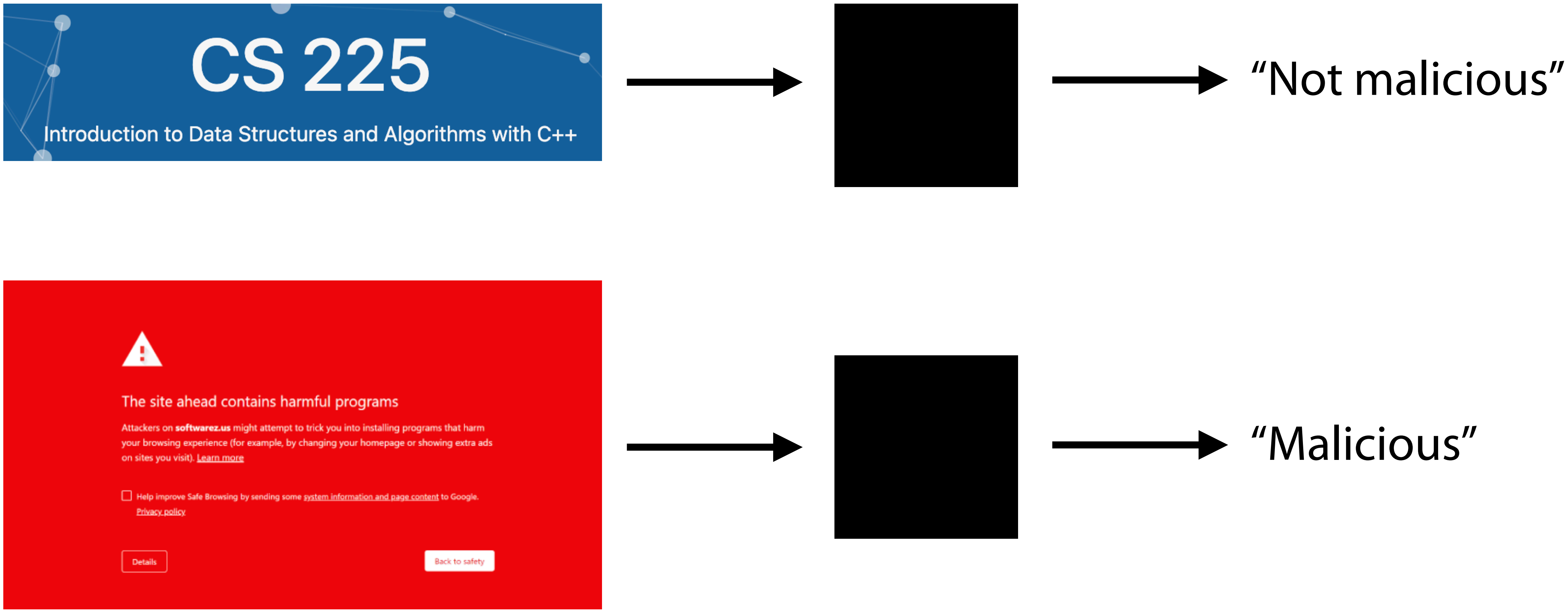
If the value in the BF is 0:

If the value in the BF is 1:



Probabilistic Accuracy: Malicious Websites

Imagine we have a detection oracle that identifies if a site is malicious



Probabilistic Accuracy: Malicious Websites

Imagine we have a detection oracle that identifies if a site is malicious

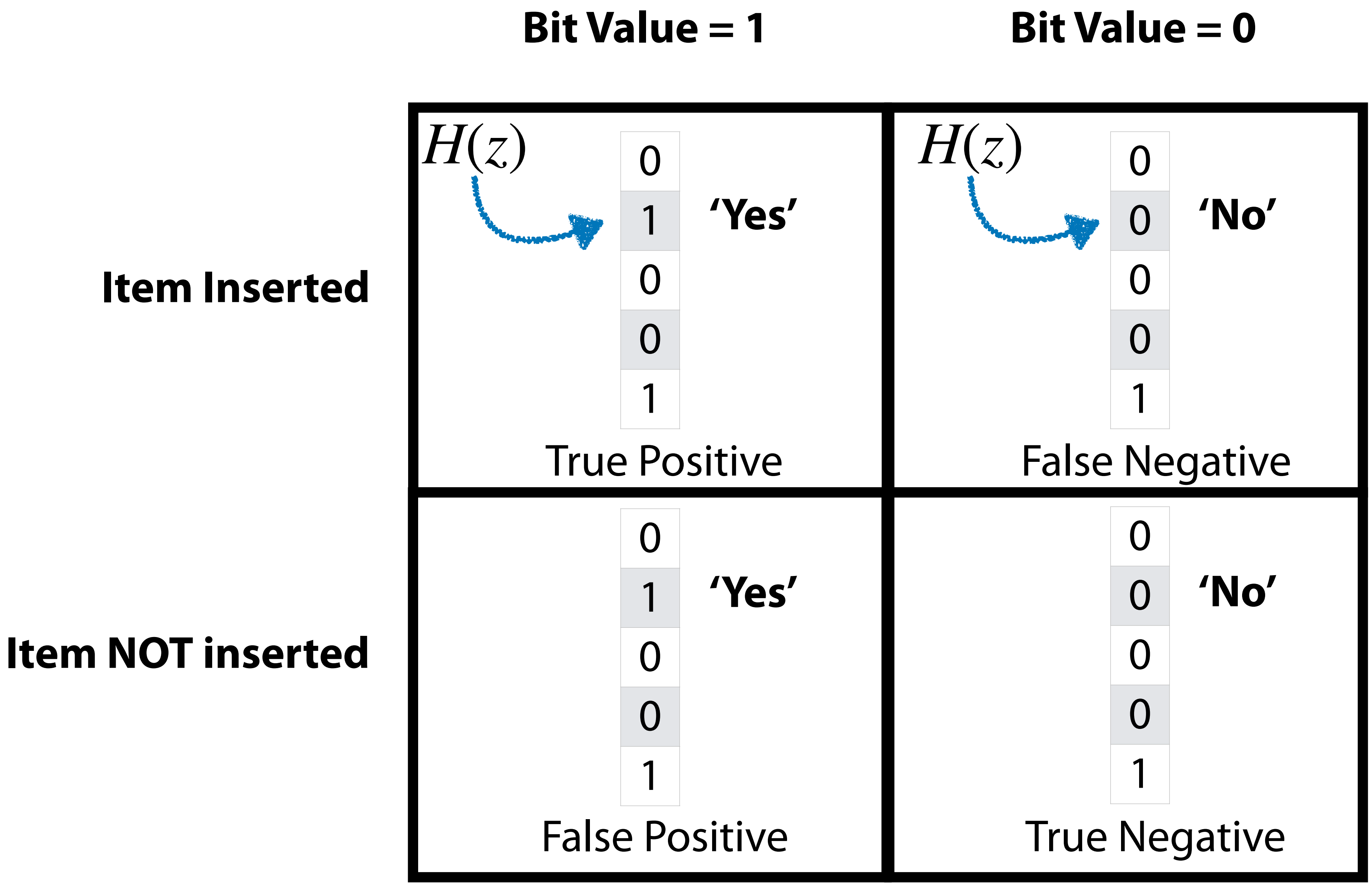
True Positive:

False Positive:

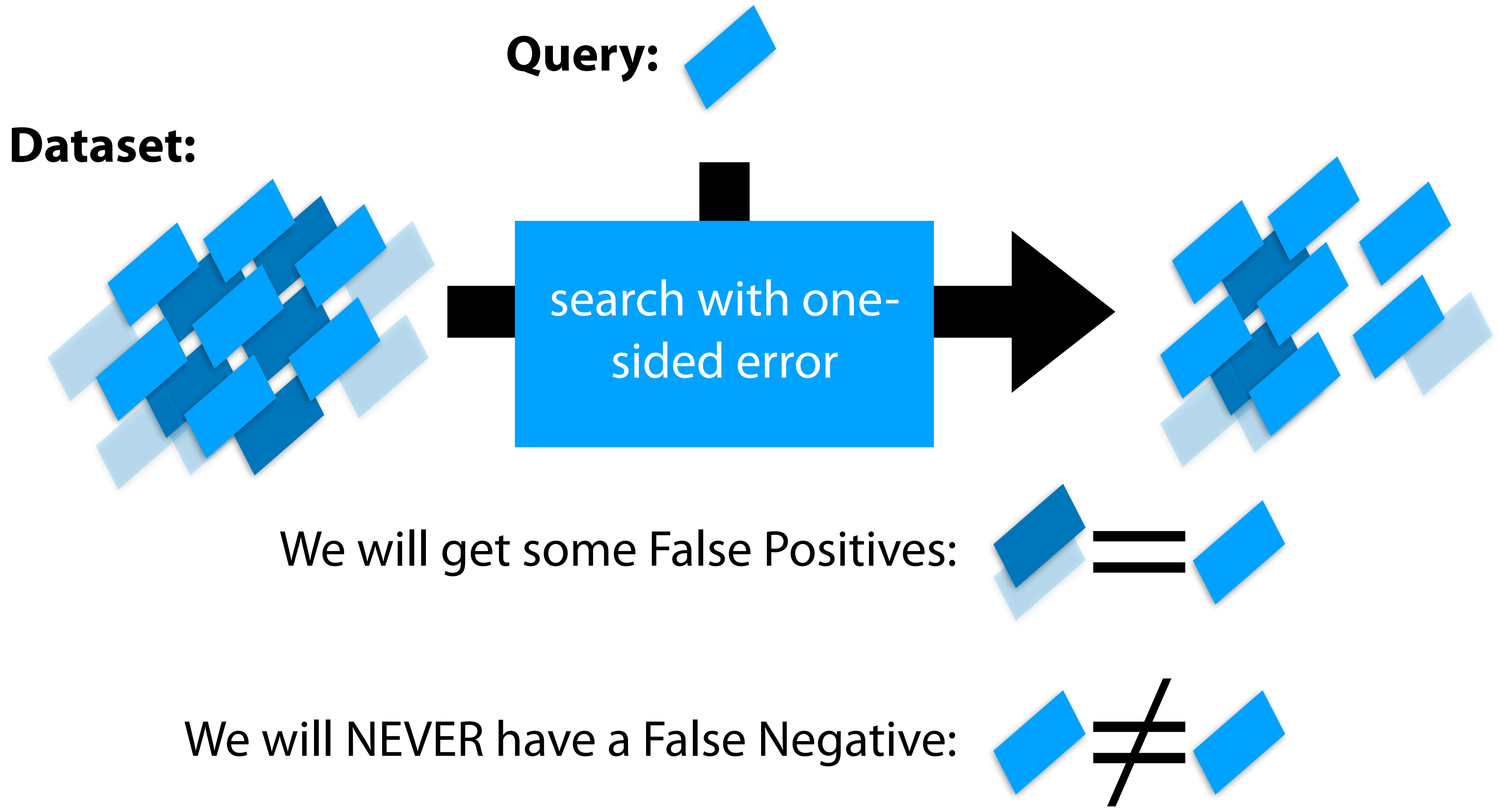
False Negative:

True Negative:

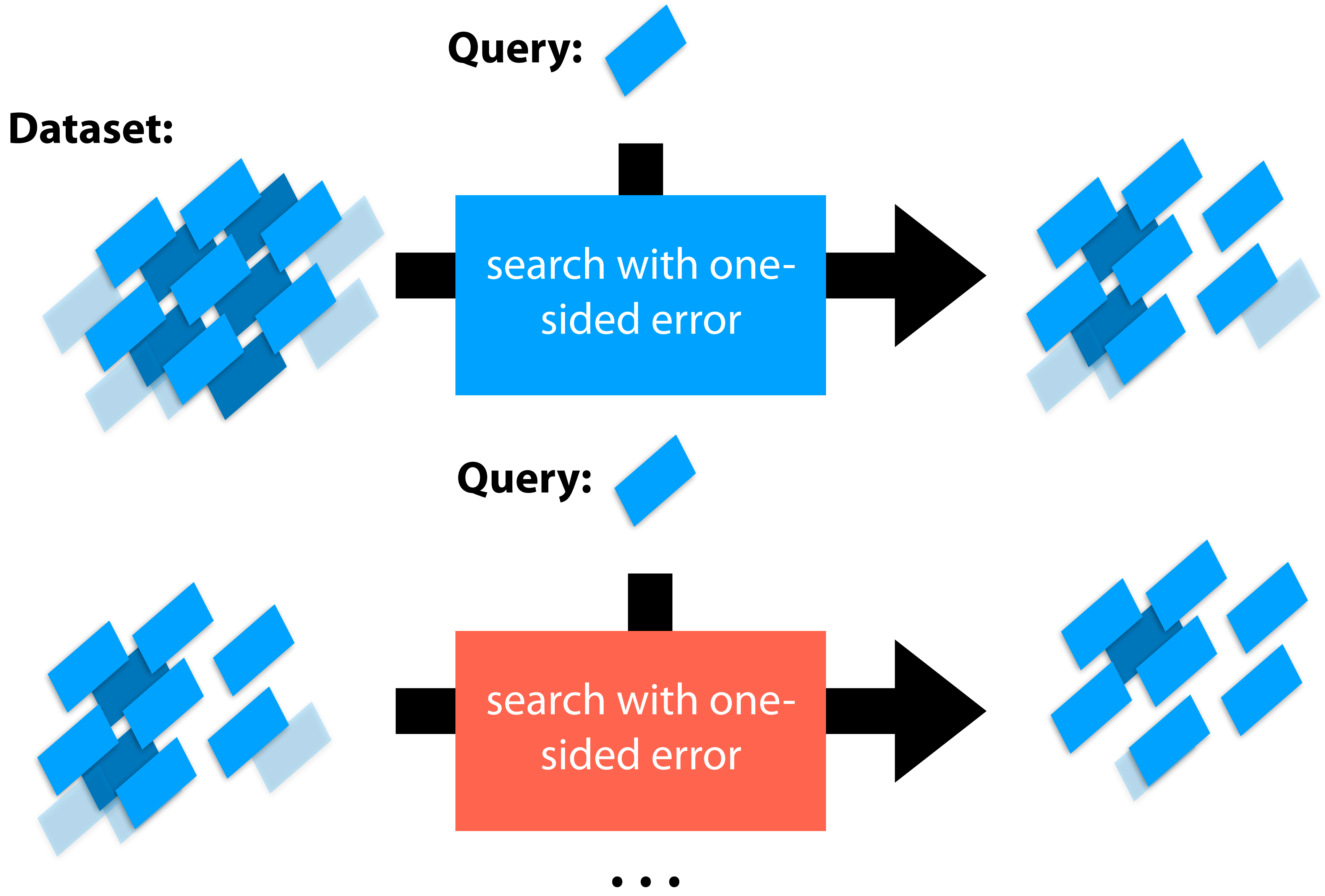
Imagine we have a **bloom filter** that **stores malicious sites…**



Probabilistic Accuracy: One-sided error



Probabilistic Accuracy: One-sided error

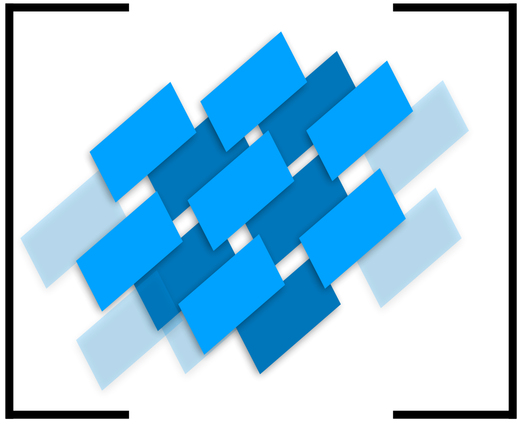


Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

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

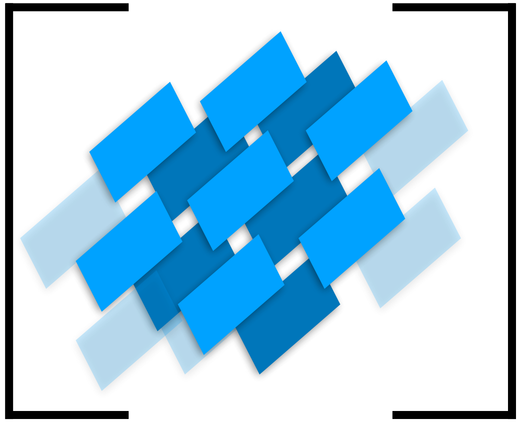
h_1



Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

h_1	h_2
0	0
1	0
0	0
1	**1**
0	0
0	0
0	0
1	**1**
0	0
1	**1**
1	0
0	0
1	**1**
0	**1**
1	**1**
1	0
0	0
1	**1**
0	0
1	0

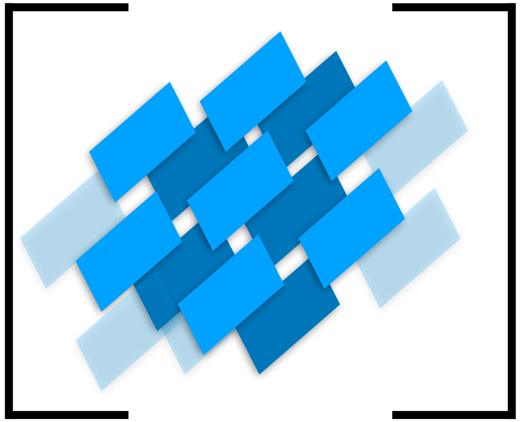


Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

h_1	h_2	
0	0	0
1	0	**1**
0	0	**1**
1	1	**1**
0	0	0
0	0	0
0	0	**1**
1	1	**1**
0	0	0
1	1	**1**
1	0	**1**
0	0	0
1	1	**1**
0	1	0
1	1	**1**
1	0	**1**
0	0	0
1	1	**1**
0	0	0
1	0	**1**

h_3



Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

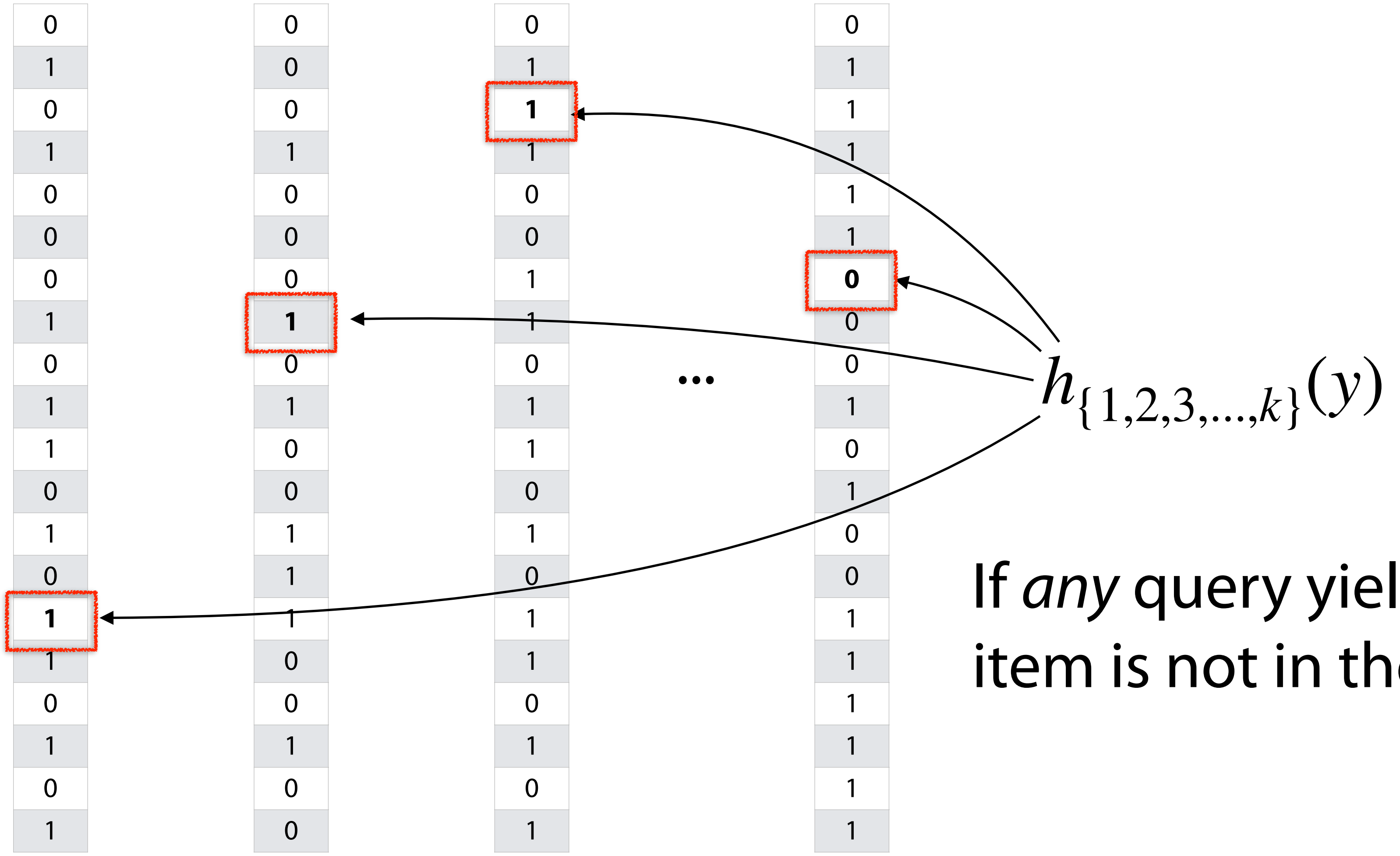
h_1	h_2	h_3	…	h_k
0	0	0		0
1	0	1		1
0	0	1		1
1	1	1		1
0	0	0		1
0	0	0		1
0	0	1		0
1	1	1		0
0	0	0		0
1	1	1		1
1	0	1		0
0	0	0		1
1	1	1		0
0	1	0		0
1	1	1		1
1	0	1		1
0	0	0		1
1	1	1		1
0	0	0		1
1	0	1		1

Bloom Filter: Repeated Trials

0	0	0		0
1	0	1		1
0	0	1		1
1	1	1		1
0	0	0		1
0	0	0		1
0	0	1		0
1	1	1		0
0	0	0	…	0
1	1	1		1
1	0	1		0
0	0	0		1
1	1	1		0
0	1	0		0
1	1	1		1
1	0	1		1
0	0	0		1
1	1	1		1
0	0	0		1
1	0	1		1

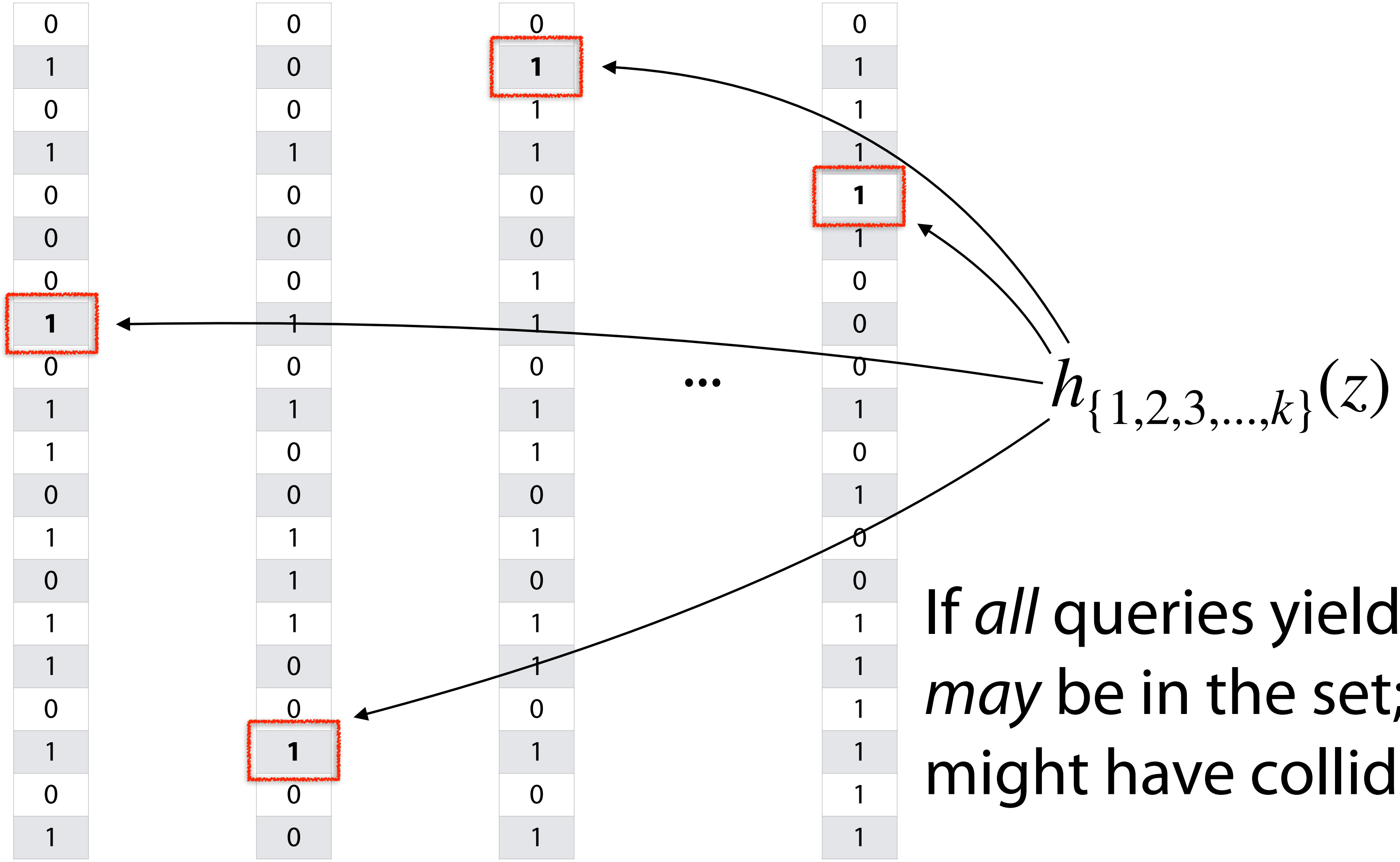
$$h_{\{1,2,3,\ldots,k\}}(y)$$

Bloom Filter: Repeated Trials



If *any* query yields 0, item is not in the set

Bloom Filter: Repeated Trials



If *all* queries yield 1, item *may* be in the set; or we might have collided *k* times

Bloom Filter: Repeated Trials

Using repeated trials, even a very bad filter can still have a very low FPR!

If we have k bloom filter, each with a FPR p, what is the likelihood that ***all*** filters return the value '1' for an item we didn't insert?

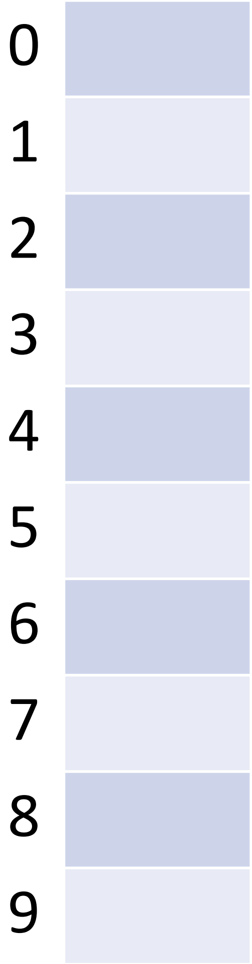
Bloom Filter: Repeated Trials

But doesn't this hurt our storage costs by storing k separate filters?

h_1	h_2	h_3	...	h_k
0	0	0		0
1	0	1		1
0	0	1		1
1	1	1		1
0	0	0		1
0	0	0		1
0	0	1		0
1	1	1		0
0	0	0		0
1	1	1		1
1	0	1		0
0	0	0		1
1	1	1		0
0	1	0		0
1	1	1		1
1	0	1		1
0	0	0		1
1	1	1		1
0	0	0		1
1	0	1		1

Bloom Filter: Repeated Trials

Rather than use a new filter for each hash, one filter can use k hashes



S = { 6, 8, 4 }

$h_1(x) = x \% 10$ **$h_2(x) = 2x \% 10$** **$h_3(x) = (5+3x) \% 10$**

Bloom Filter: Repeated Trials

Rather than use a new filter for each hash, one filter can use k hashes

0	0
1	0
2	1
3	1
4	1
5	0
6	1
7	1
8	1
9	1

$h_1(x) = x \% 10$ **$h_2(x) = 2x \% 10$** **$h_3(x) = (5+3x) \% 10$**

`_find(1)`

`_find(16)`

Bloom Filter

$H = \{h_1, h_2, \ldots, h_k\}$

A probabilistic data structure storing a set of values

Built from a bit vector of length m and k hash functions

Insert / Find runs in: ____________________

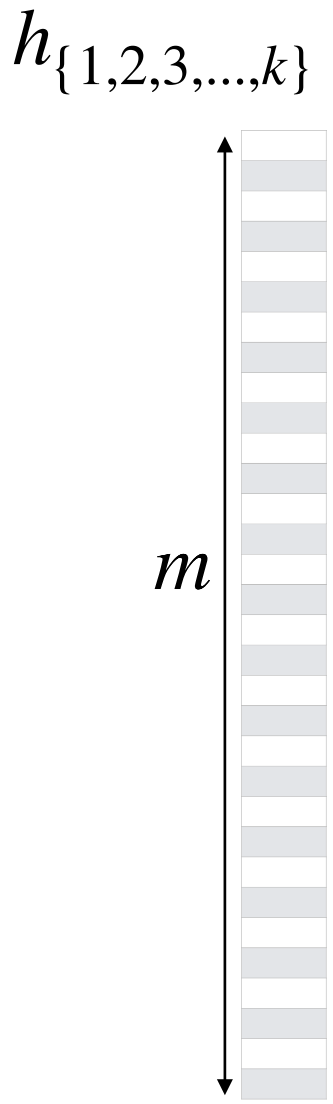
Delete is not possible (yet)!

0
0
1
0
0
1
0
1
0
0

Bloom Filter: Error Rate

Given bit vector of size m and k SUHA hash function

What is our expected FPR after n objects are inserted?

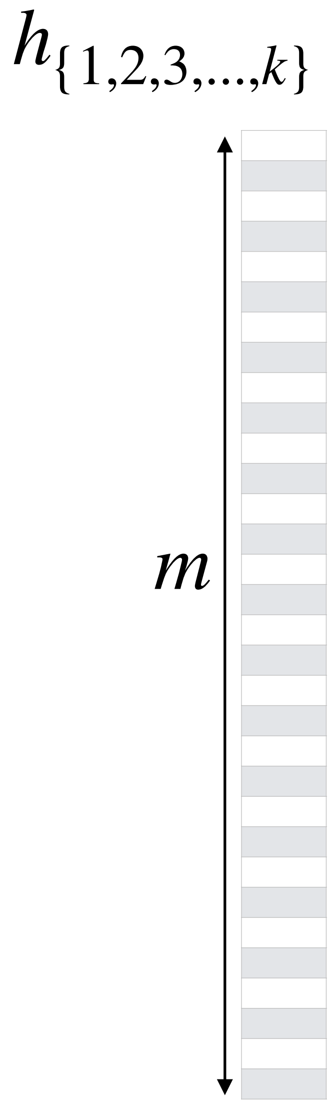


Bloom Filter: Error Rate

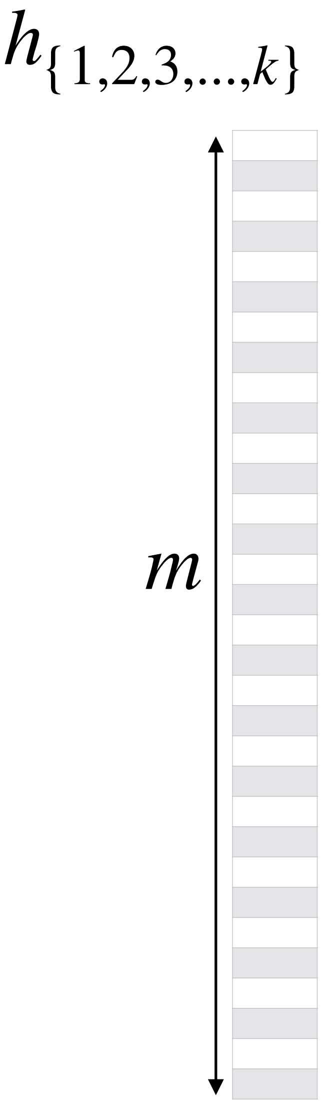
Given bit vector of size m and 1 SUHA hash function

What's the probability a specific bucket is 1 after one object is inserted?

Same probability given k SUHA hash function?



Bloom Filter: Error Rate



Given bit vector of size m and k SUHA hash function

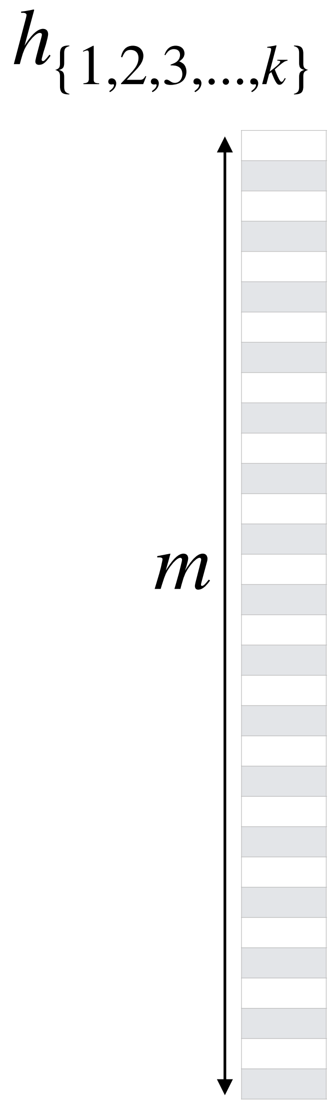
Probability a specific bucket is 0 after one object is inserted?

After n objects are inserted?

Bloom Filter: Error Rate

Given bit vector of size m and k SUHA hash function

What's the probability a specific bucket is 1 after n objects are inserted?



Bloom Filter: Error Rate

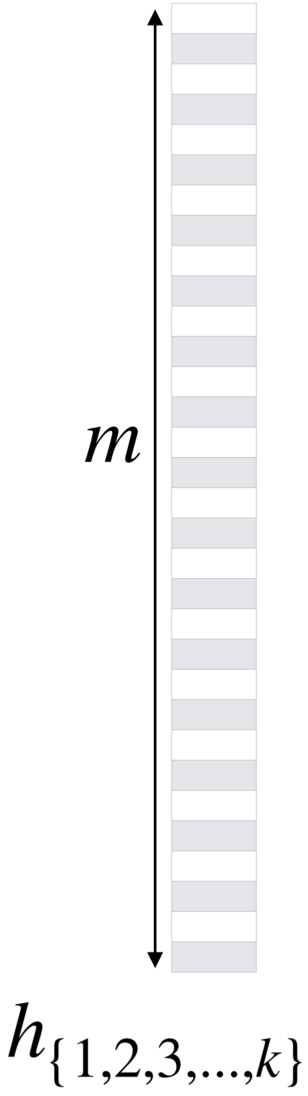
Given bit vector of size m and k SUHA hash function

What is our expected FPR after n objects are inserted?

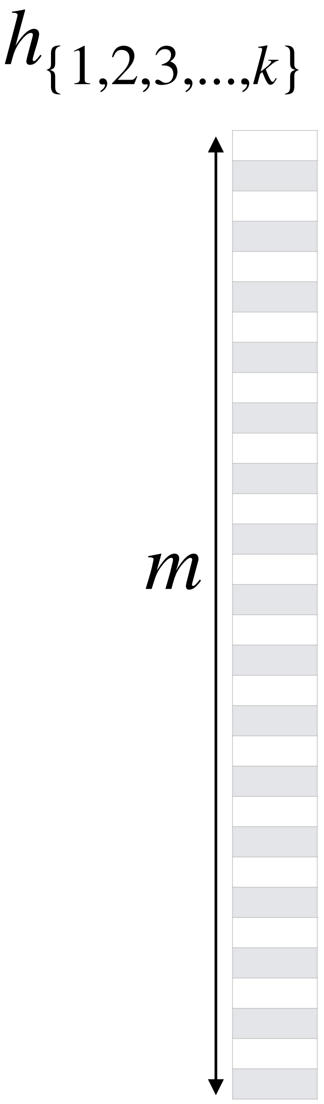
The probability my bit is 1 after n objects inserted

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

The number of [assumed independent] trials



Bloom Filter: Error Rate



Vector of size m, k SUHA hash function, and n objects

To minimize the FPR, do we prefer…

(A) large k **(B) small** k

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

Bloom Filter: Error Rate

Vector of size m, k SUHA hash function, and n objects

(A) large k

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

As k increases, this gets smaller!

(B) small k

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

As k decreases, this gets smaller!

Bloom Filter: Optimal Error Rate

To build the optimal hash function, fix **m** and **n**!

Claim: The optimal hash function is when $k* = ln\ 2 \cdot \frac{m}{n}$

(1) $$\left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1 - e^{\frac{-nk}{m}}\right)^{k}$$

(2) $$\frac{d}{dk}\left(1 - e^{\frac{-nk}{m}}\right)^{k} \approx \frac{d}{dk}\left(k\ \ ln(1 - e^{\frac{-nk}{m}})\right)$$

Bloom Filter: Optimal Error Rate

Claim 1: $$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1-e^{\frac{-nk}{m}}\right)^{k}$$

$$\left(1-\frac{1}{m}\right)^{nk} = e^{ln\left[\left(1-\frac{1}{m}\right)^{nk}\right]}$$

$$= e^{ln\left[\left(1-\frac{1}{m}\right)\right]nk}$$

$$\approx e^{\frac{-nk}{m}}$$

Bloom Filter: Optimal Error Rate

Claim 2: $$\frac{d}{dk}\left(1 - e^{\frac{-nk}{m}}\right)^k \approx \frac{d}{dk}\left(k \;\; ln(1 - e^{\frac{-nk}{m}})\right)$$

Fact: $$\frac{d}{dx}\ln\; f(x) = \frac{1}{f(x)}\frac{df(x)}{dx}$$

TL;DR: $min\left[f(x)\right] = min\left[\ln\; f(x)\right]$

Derivative is zero when $k* = \ln 2 \cdot \frac{m}{n}$

Bloom Filter: Error Rate

$$\left(1 - e^{\frac{-nk}{m}}\right)^k$$

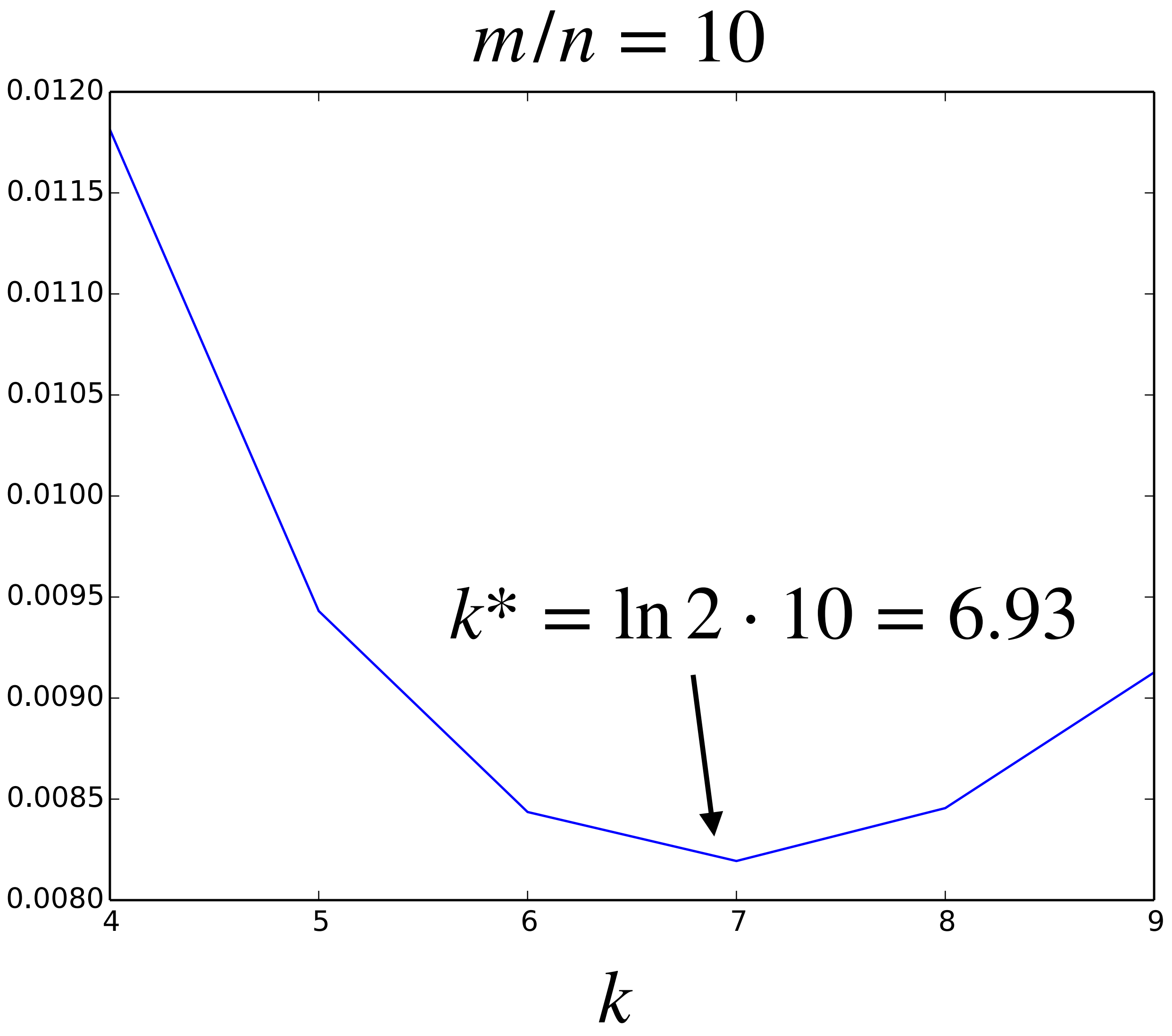


Figure by Ben Langmead

Bloom Filter: Optimal Parameters

$$k^* = \ln 2 \cdot \frac{m}{n}$$

Given any two values, we can optimize the third

$n = 100$ items $\quad$ $k = 3$ hashes $\quad$ $m =$

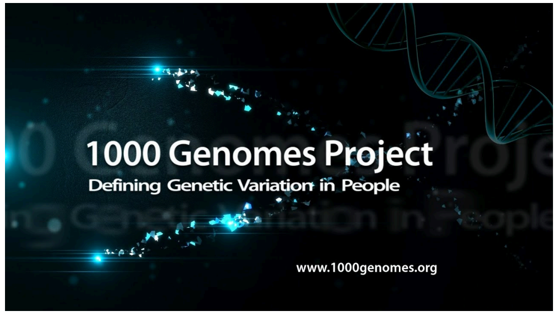
$m = 100$ bits $\quad$ $n = 20$ items $\quad$ $k =$

$m = 100$ bits $\quad$ $k = 2$ items $\quad$ $n =$

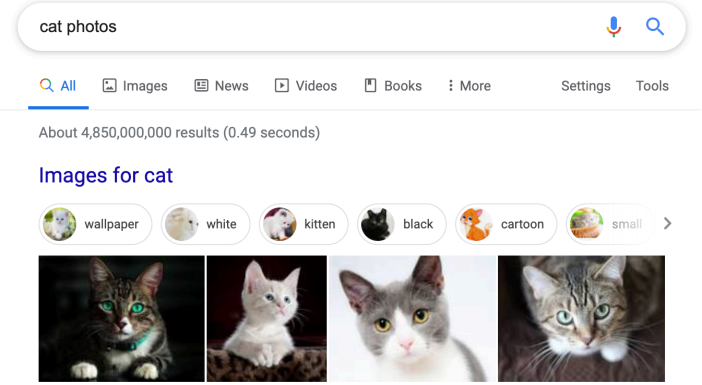
Bloom Filter: Optimal Parameters

$$m = \frac{nk}{\ln 2} \approx 1.44 \cdot nk$$

Optimal hash function is still O(m)!

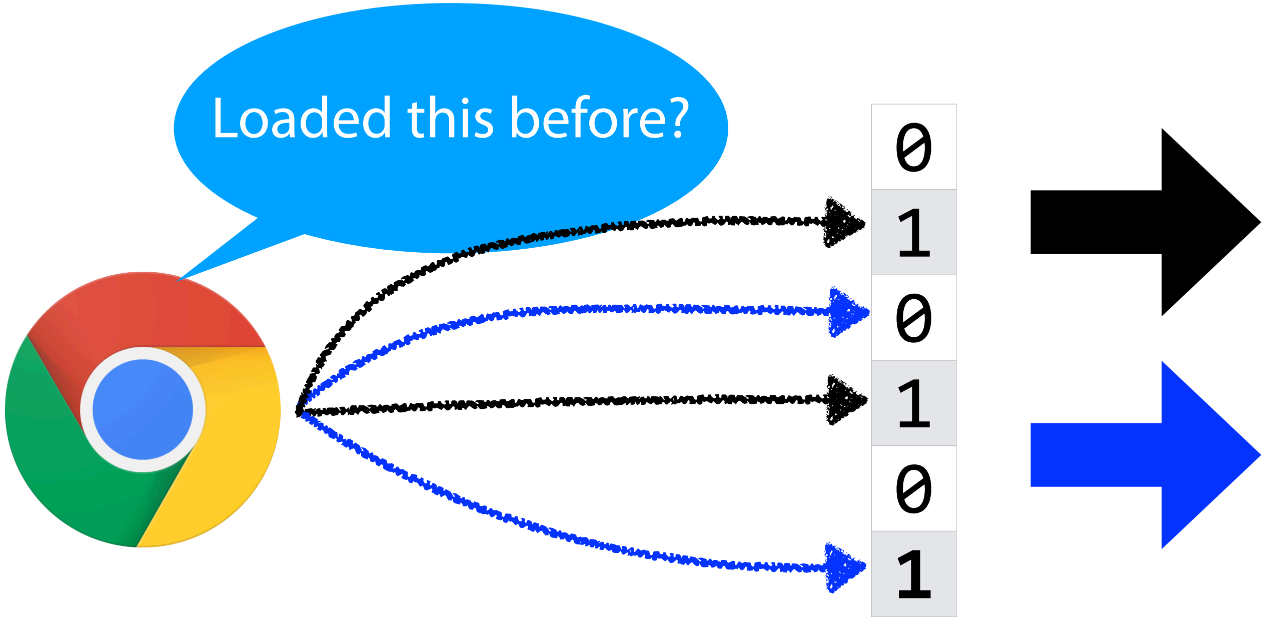


n = 250,000 files vs ~10^{15} nucleotides vs 260 TB



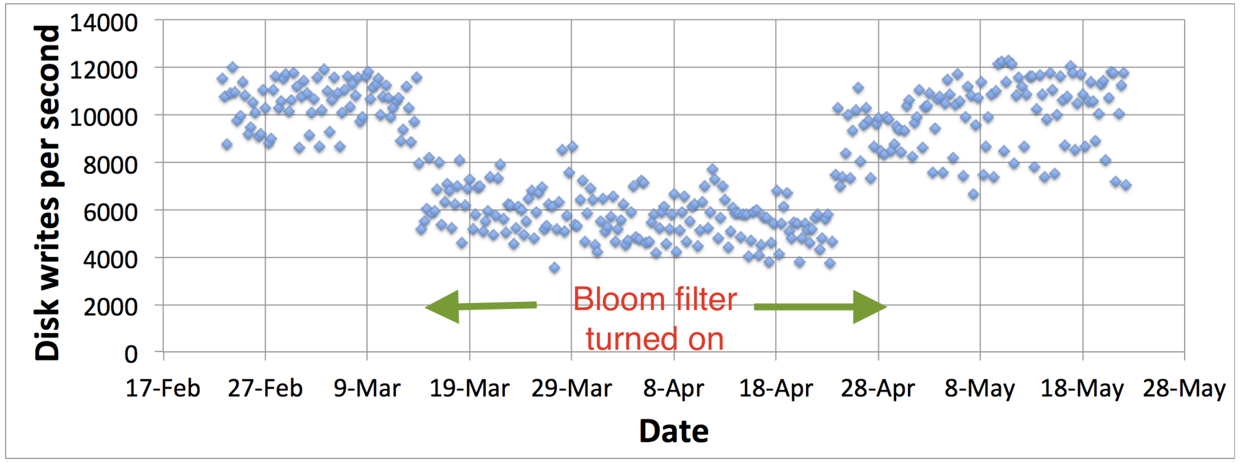
n = 60 billion — 130 trillion

Bloom Filter: Website Caching



Cache this page!

Add to filter (but don't cache!)



Maggs, Bruce M., and Ramesh K. Sitaraman. **Algorithmic nuggets in content delivery.** *ACM SIGCOMM Computer Communication Review* 45.3 (2015): 52-66.

Bitwise Operators in C++

Traditionally, bit vectors are read from RIGHT to LEFT

Warning: Lab_Bloom won't do this but MP_Sketching will!

0	0	0	0	1	1	1

1	0	0	1	0	1	0

Bitwise Operators in C++

Let **A = 10110** Let **B = 01110**

```
   ~B:
A & B:
A | B:
A >> 2:
B << 2:
```

Bit Vectors: Unioning

Bit Vectors can be trivially merged using bit-wise union.

0	1		0	0		0	
1	0		1	1		1	
2	1		2	1		2	
3	1		3	0		3	
4	0	$\cup$	4	0	=	4	
5	0		5	0		5	
6	1		6	1		6	
7	0		7	1		7	
8	0		8	1		8	
9	1		9	1		9	

Bit Vectors: Intersection

Bit Vectors can be trivially merged using bit-wise intersection.

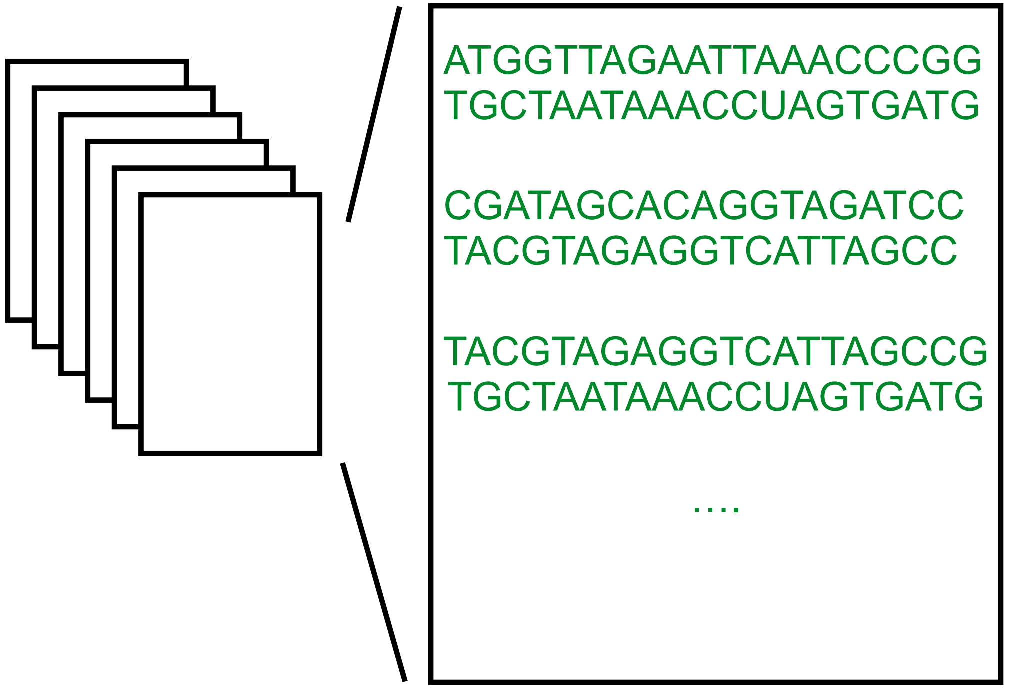
0	1		0	0		0	
1	0		1	1		1	
2	1		2	1		2	
3	1		3	0		3	
4	0	U	4	0	=	4	
5	0		5	0		5	
6	1		6	1		6	
7	0		7	1		7	
8	0		8	1		8	
9	1		9	1		9	

Bit Vector Merging

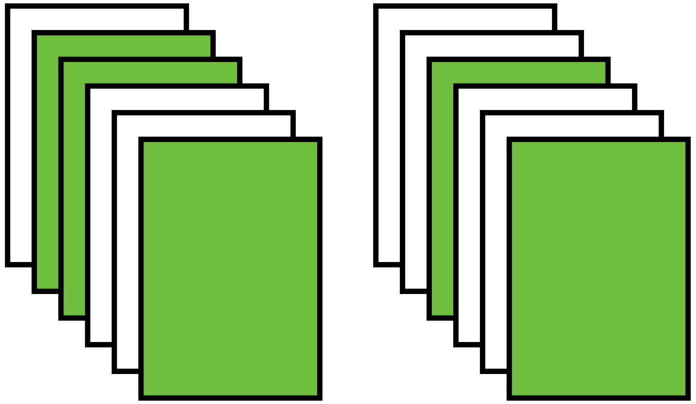
What is the conceptual meaning behind **union** and **intersection**?

Sequence Bloom Trees

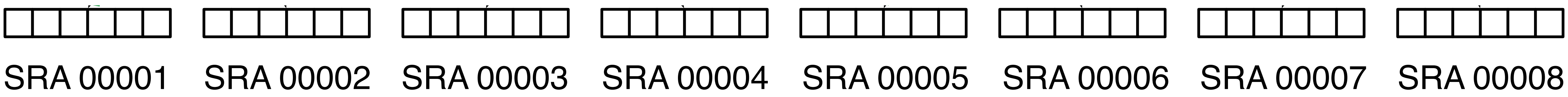
Imagine we have a large collection of text…



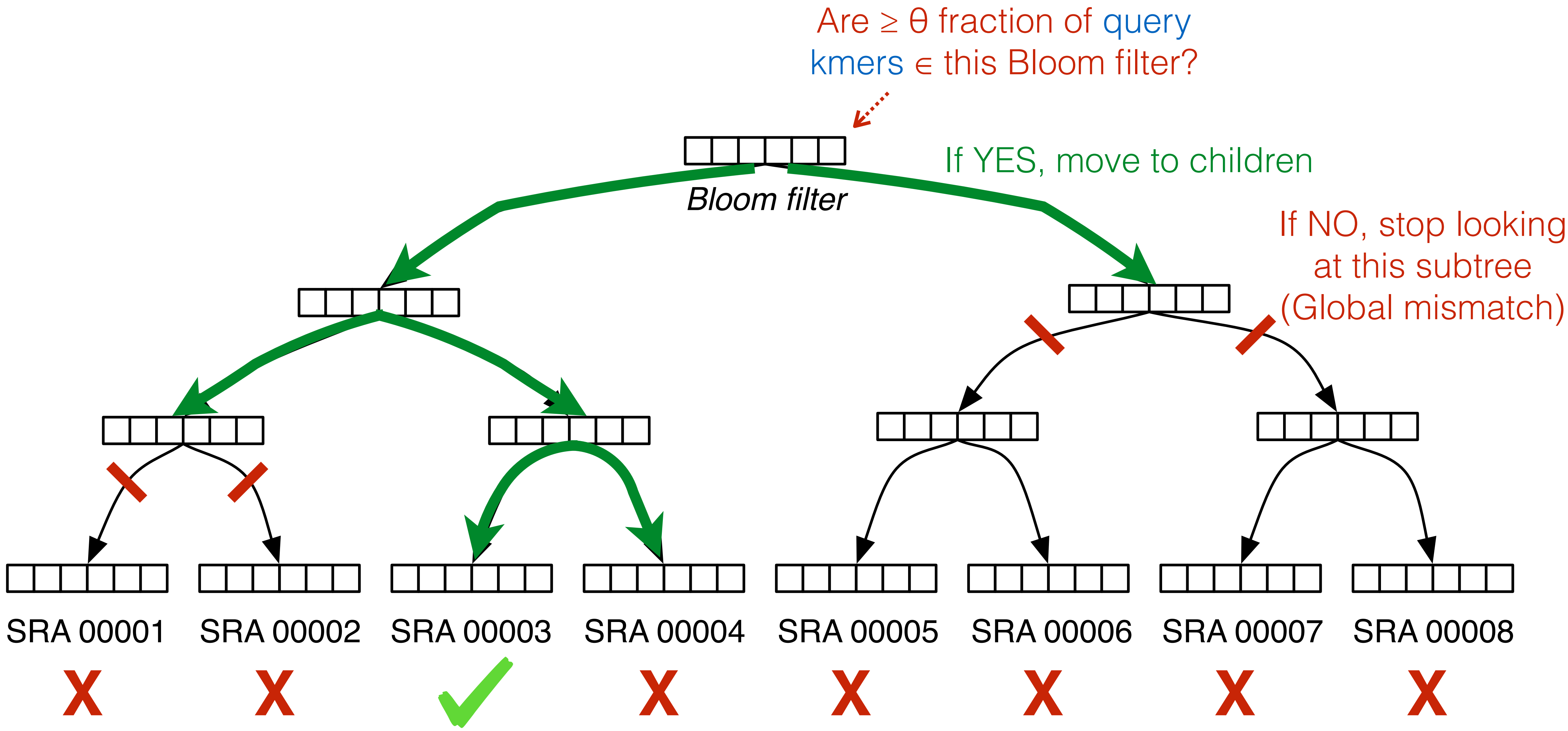
And our goal is to search these files for a query of interest…



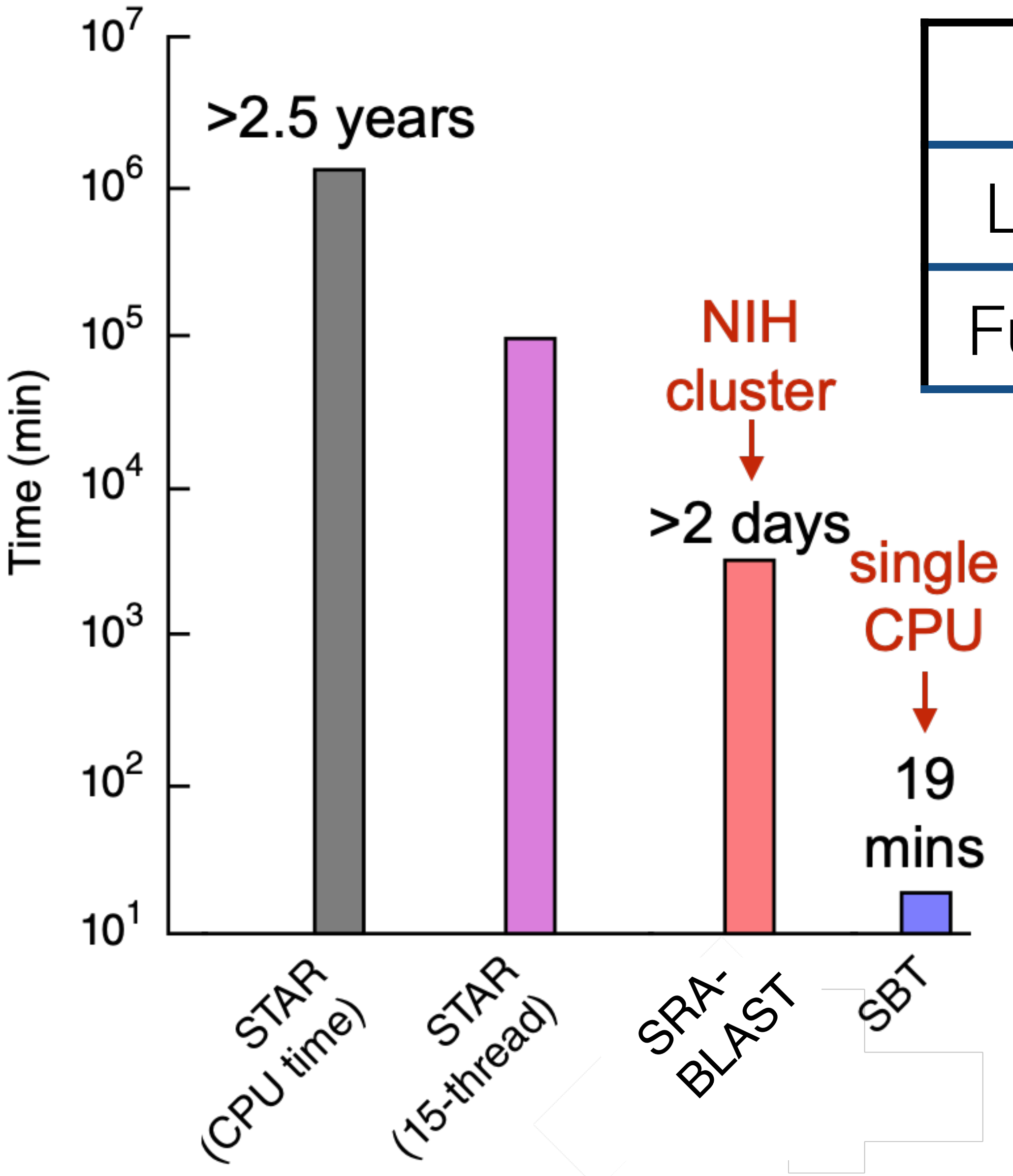
Sequence Bloom Trees



Sequence Bloom Trees



Sequence Bloom Trees



	SRA	FASTA.gz	SBT
Leaves	4966 GB	2692 GB	63 GB
Full Tree	-	-	200 GB

Solomon, Brad, and Carl Kingsford. "Fast search of thousands of short-read sequencing experiments." *Nature biotechnology* 34.3 (2016): 300-302.

Solomon, Brad, and Carl Kingsford. "Improved search of large transcriptomic sequencing databases using split sequence bloom trees." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2017.

Sun, Chen, et al. "Allsome sequence bloom trees." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2017.

Harris, Robert S., and Paul Medvedev. "Improved representation of sequence bloom trees." *Bioinformatics* 36.3 (2020): 721-727.

Bloom Filters: Tip of the Iceberg

Cohen, Saar, and Yossi Matias. "Spectral bloom filters." *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003.

Fan, Bin, et al. "Cuckoo filter: Practically better than bloom." *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014.

Nayak, Sabuzima, and Ripon Patgiri. "countBF: A General-purpose High Accuracy and Space Efficient Counting Bloom Filter." *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021.

Mitzenmacher, Michael. "Compressed bloom filters." *IEEE/ACM transactions on networking* 10.5 (2002): 604-612.

Crainiceanu, Adina, and Daniel Lemire. "Bloofi: Multidimensional bloom filters." *Information Systems* 54 (2015): 311-324.

Chazelle, Bernard, et al. "The bloomier filter: an efficient data structure for static support lookup tables." *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. 2004.

There are many more than shown here…