Data Structures and Algorithms
Bloom Filters 2

CS 225
Brad Solomon

November 21, 2025

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

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)

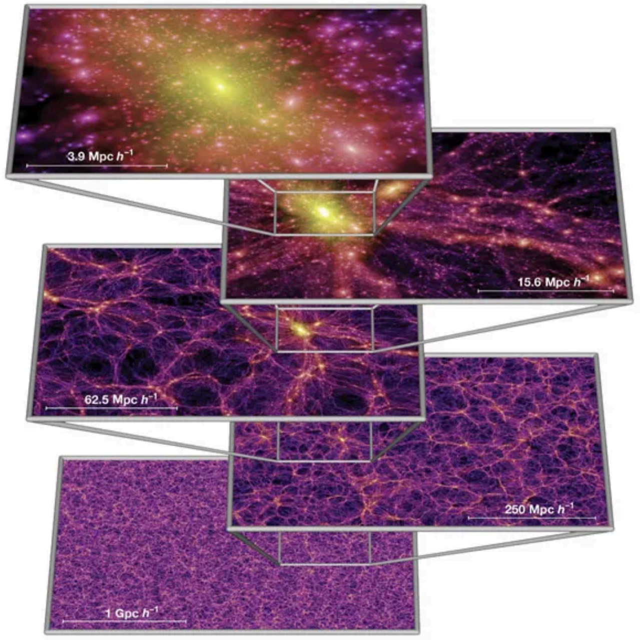


Image: https://doi.org/10.1038/nature03597

Sky Survey Projects	Data Volume
DPOSS (The Palomar Digital Sky Survey)	3 TB
2MASS (The Two Micron All-Sky Survey)	10 TB
GBT (Green Bank Telescope)	20 PB
GALEX (The Galaxy Evolution Explorer)	30 TB
SDSS (The Sloan Digital Sky Survey)	40 TB
SkyMapper Southern Sky Survey	500 TB
PanSTARRS (The Panoramic Survey Telescope and Rapid Response System)	~ 40 PB expected
LSST (The Large Synoptic Survey Telescope)	~ 200 PB expected
SKA (The Square Kilometer Array)	~ 4.6 EB expected

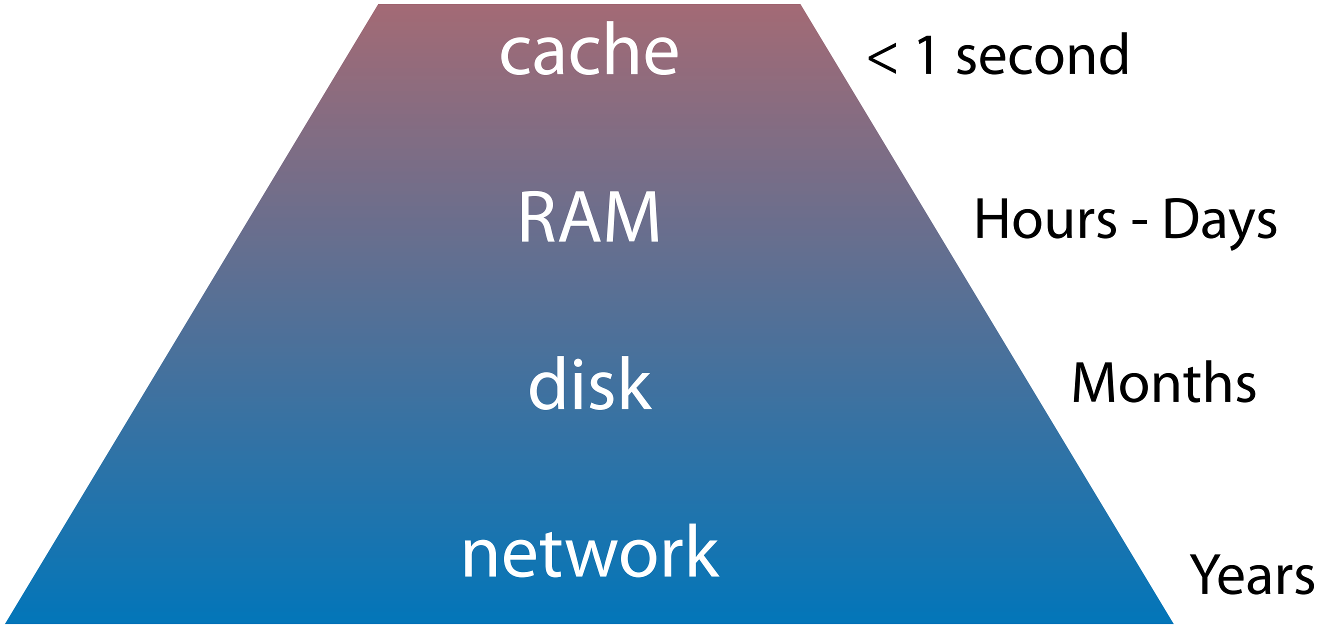
Table: http://doi.org/10.5334/dsj-2015-011

Estimated total volume of one array: 4.6 EB

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 resource limitations



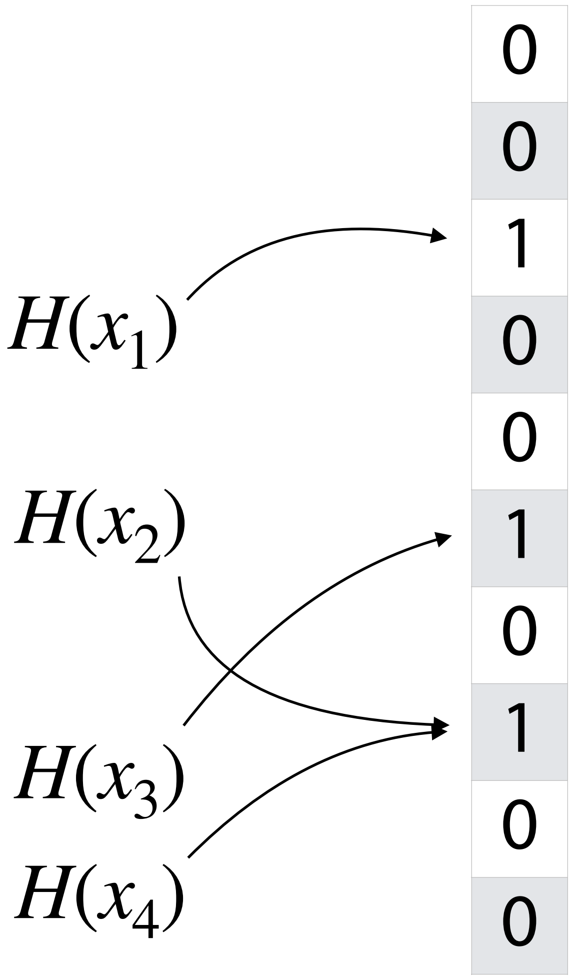
(Estimates are Time x 1 billion courtesy of https://gist.github.com/hellerbarde/2843375)

Bloom Filter: Insertion

1) Hash the input key to get its **hash value**

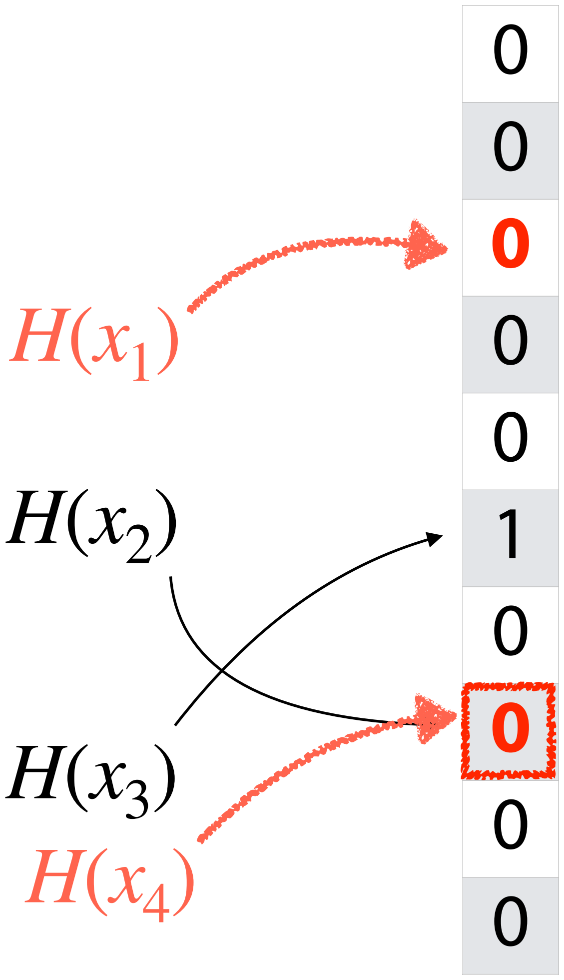
2) Set the bit at the hash value address to 1

   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

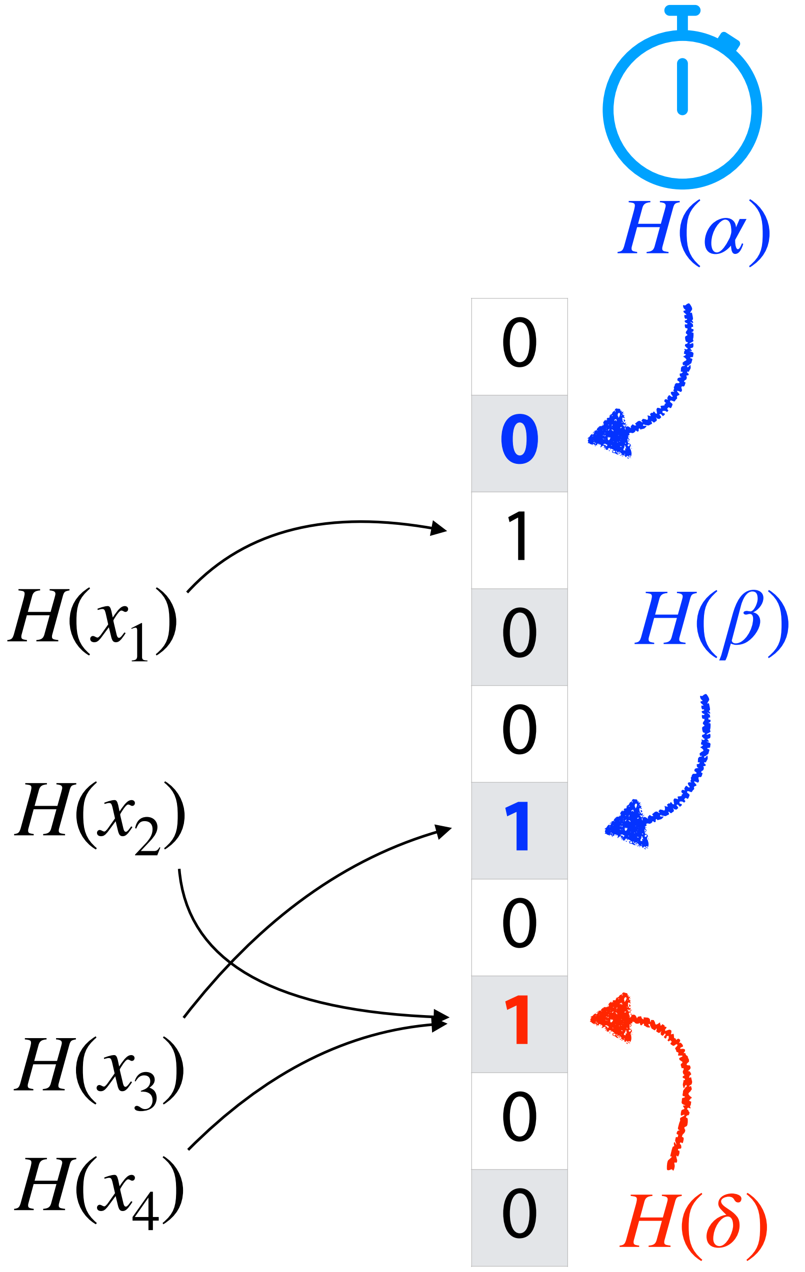
The bloom filter is a *probabilistic* data structure!

If the value in the BF is 0:

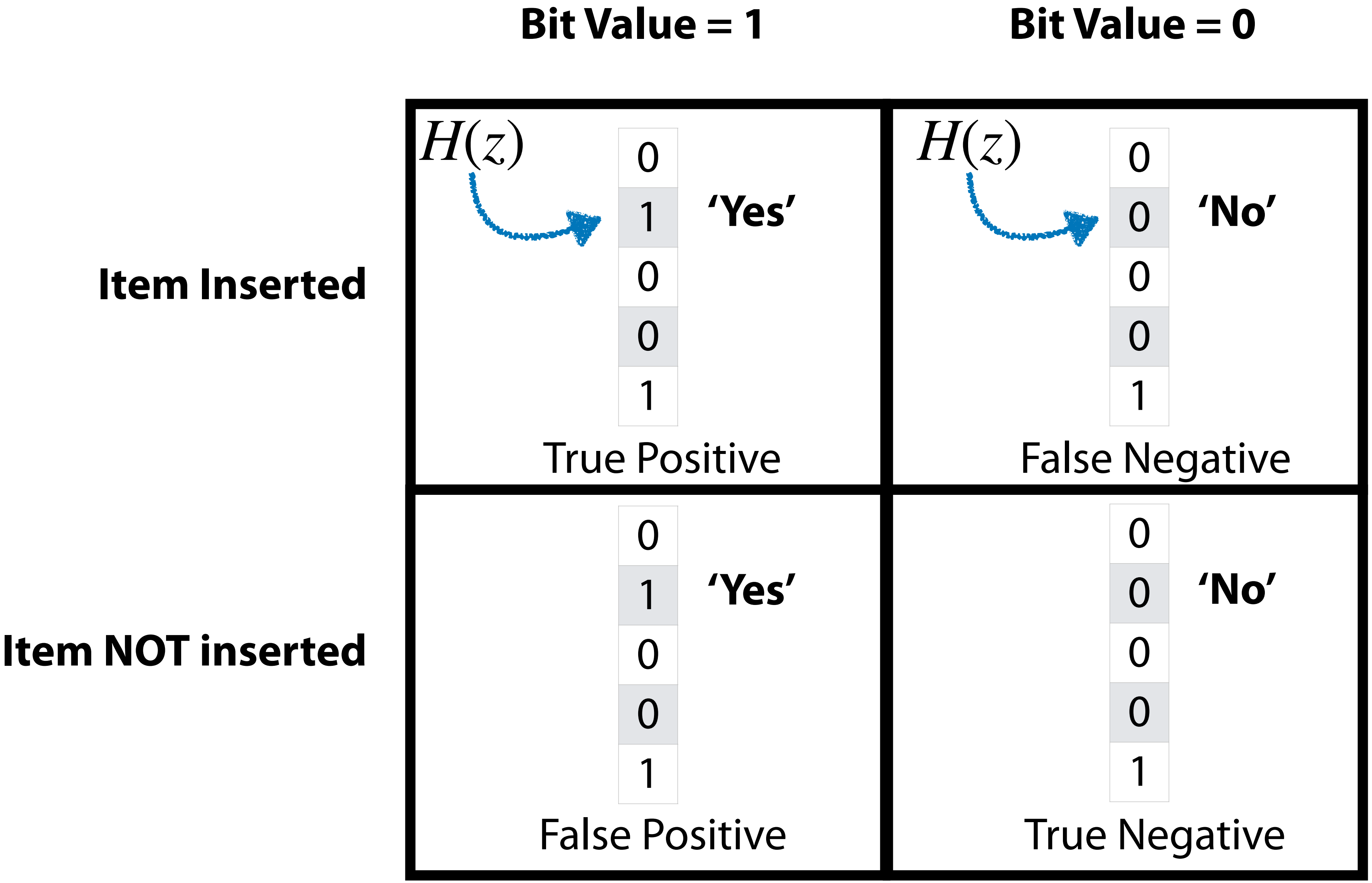
100% of time, we know it is not present

If the value in the BF is 1:

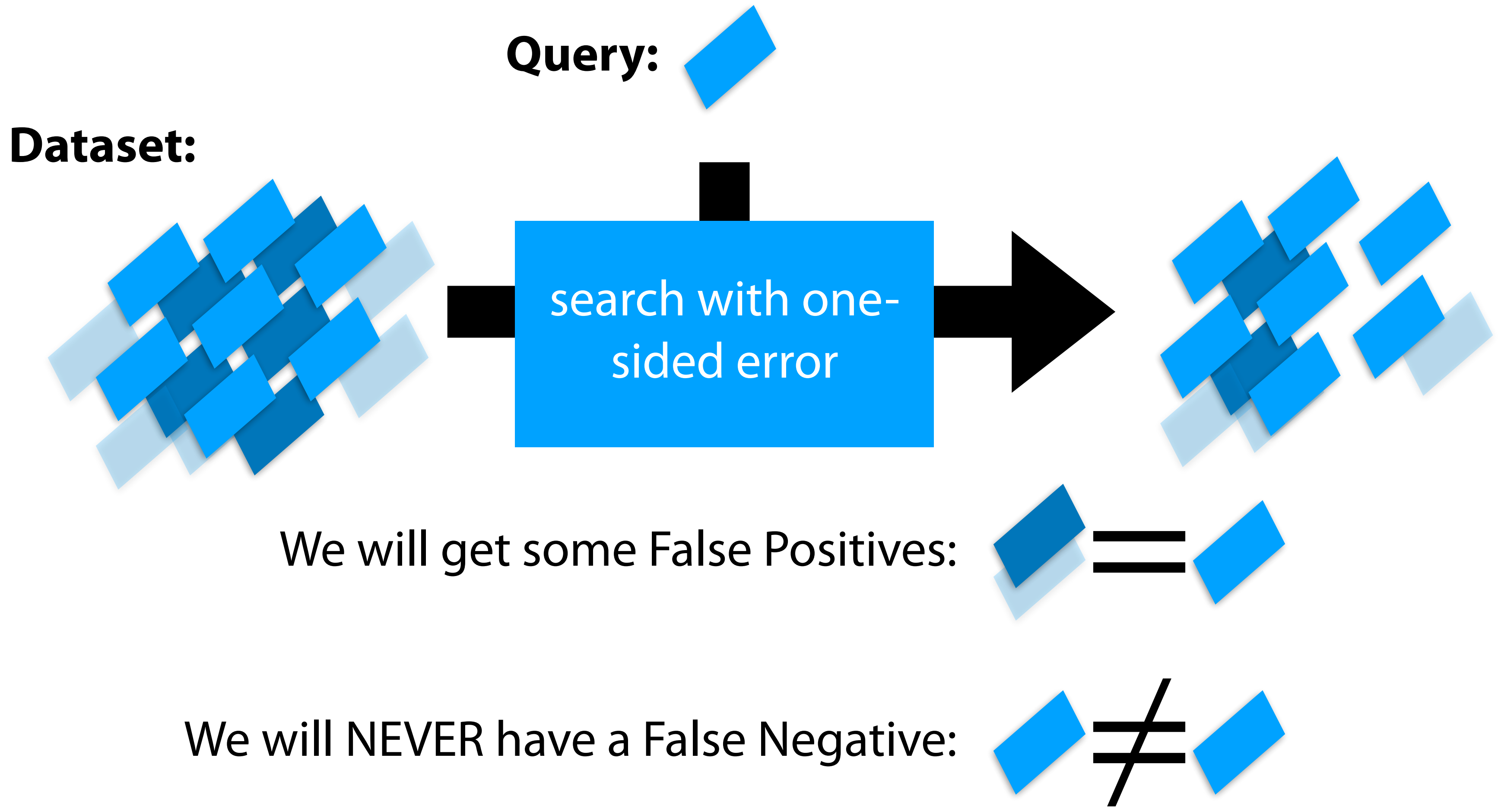
It **may** be present or it may be a hash collision



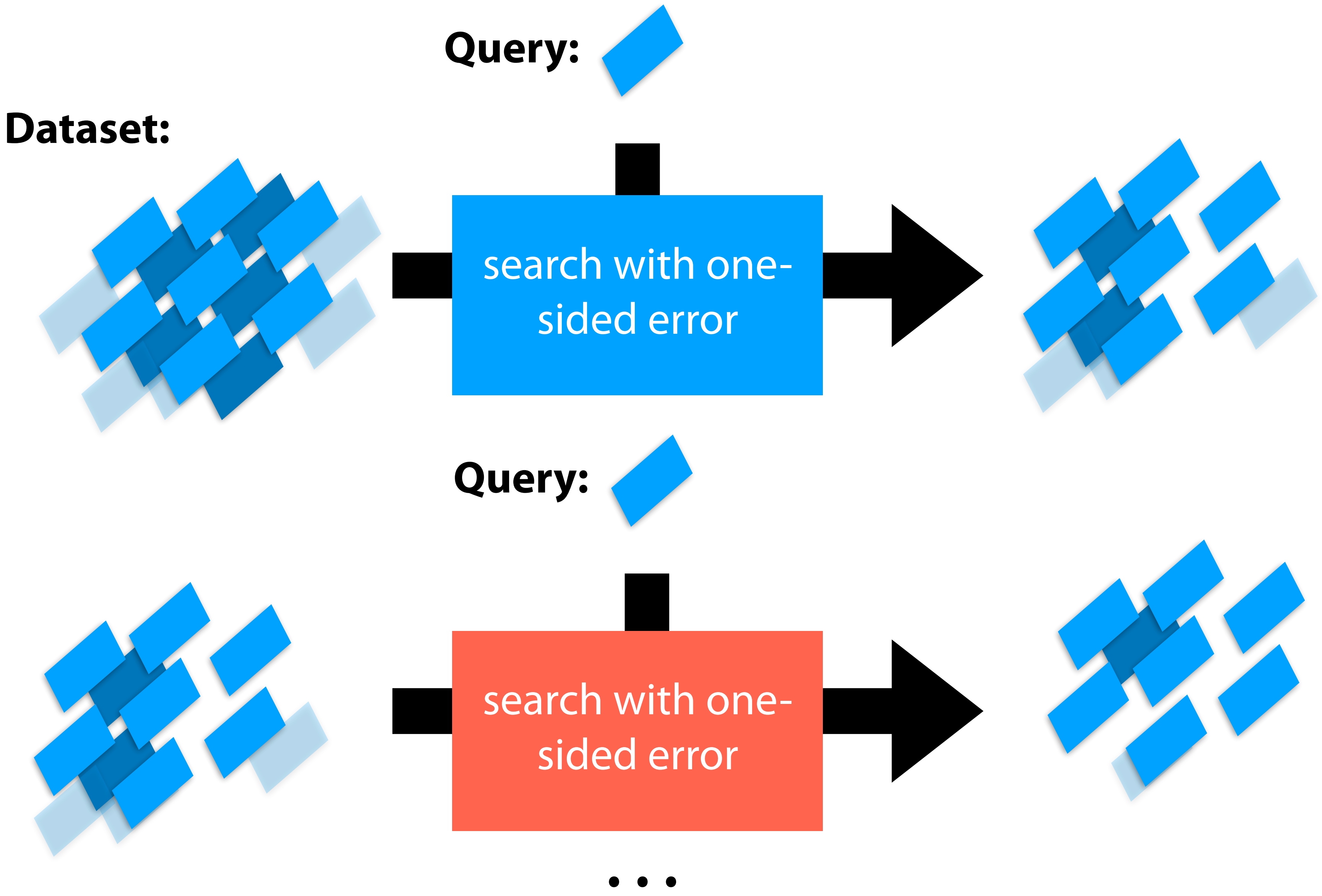
Probabilistic Accuracy in a Bloom Filter



Probabilistic Accuracy: One-sided error

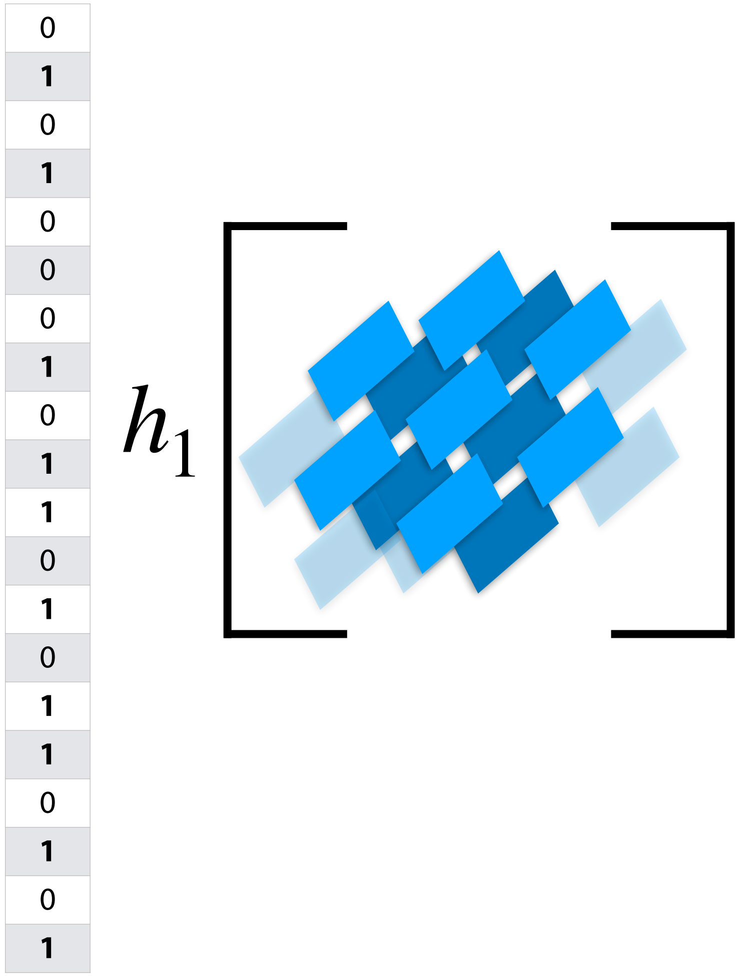


Probabilistic Accuracy: One-sided error



Bloom Filter: Repeated Trials

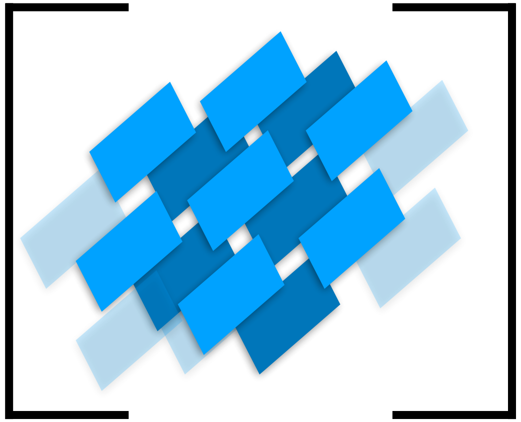
Improve accuracy by using multiple hash functions as a ‘filter’



Bloom Filter: Repeated Trials

Improve accuracy by using multiple hash functions as a 'filter'

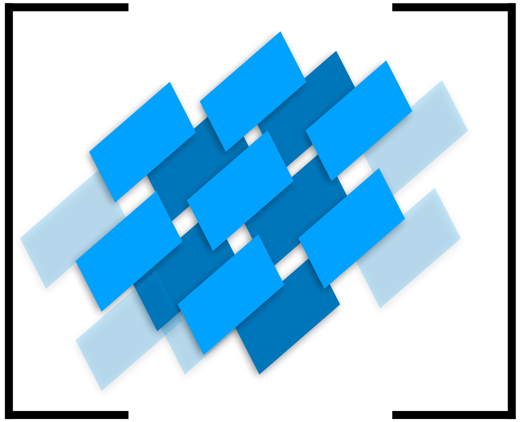
h_1	h_2
0	0
1	0
0	0
1	**1**
0	0
0	0
0	0
1	**1**
0	0
1	**1**
1	0
0	0
1	**1**
0	**1**
1	**1**
1	0
0	0
1	**1**
0	0
1	0



Bloom Filter: Repeated Trials

Improve accuracy by using multiple hash functions as a ‘filter’

h_1	h_2	h_3
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**



Bloom Filter: Repeated Trials

Each of these k Bloom Filters is a repeated trial — improved accuracy!

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

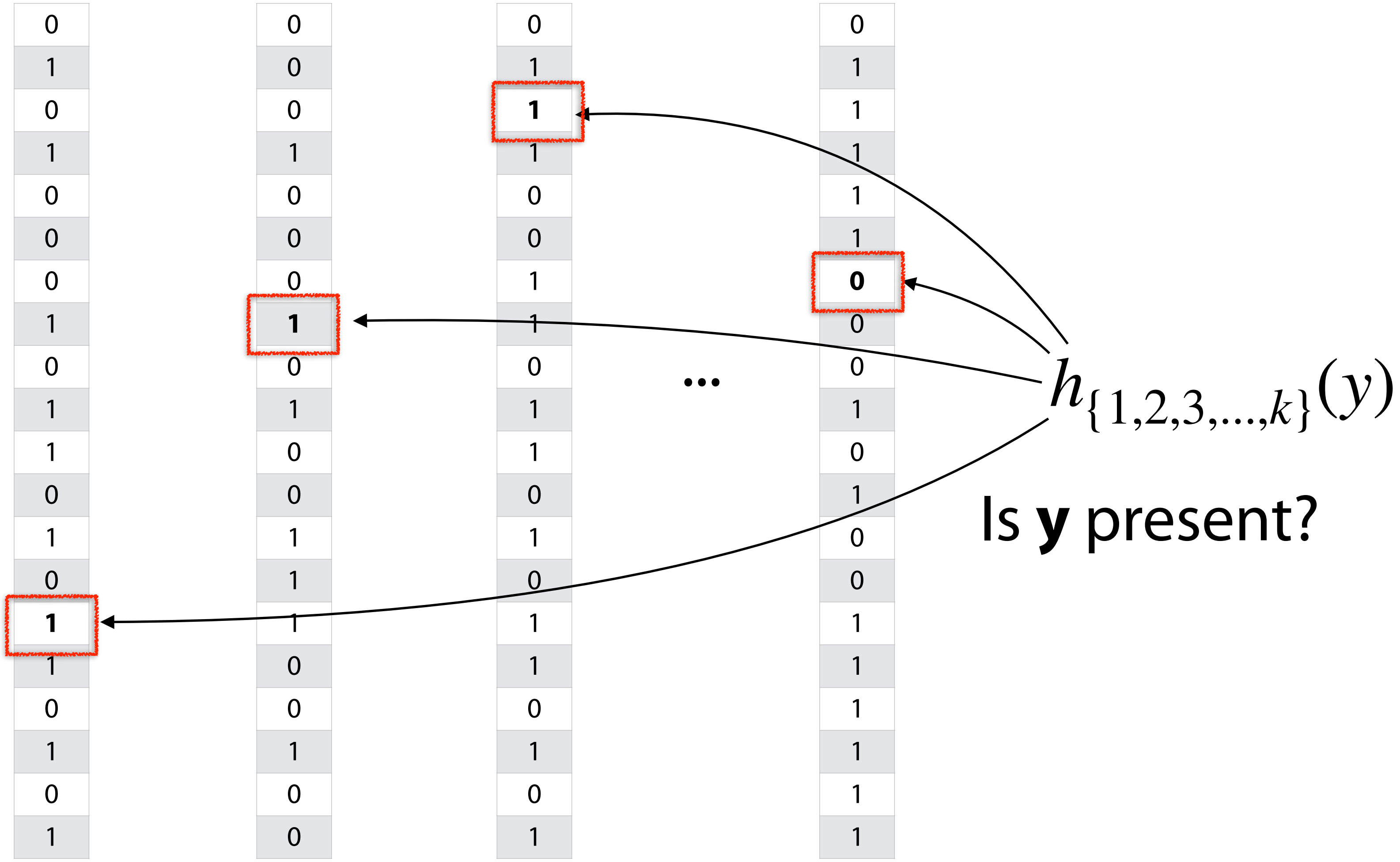
Each of these k Bloom Filters is a repeated trial — improved accuracy!

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

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

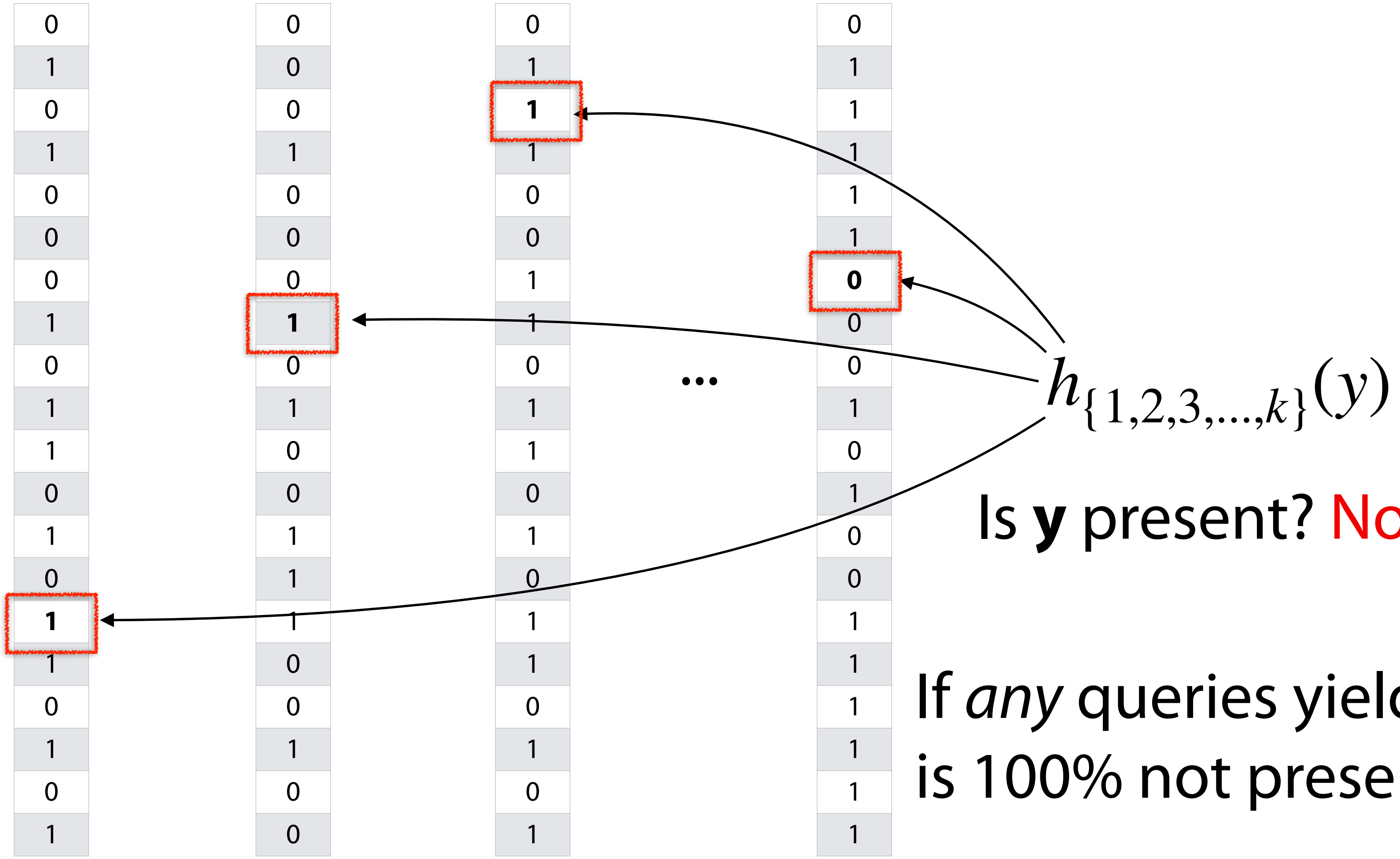
Bloom Filter: Repeated Trials

Each of these k Bloom Filters is a repeated trial — improved accuracy!



Bloom Filter: Repeated Trials

Each of these k Bloom Filters is a repeated trial — improved accuracy!

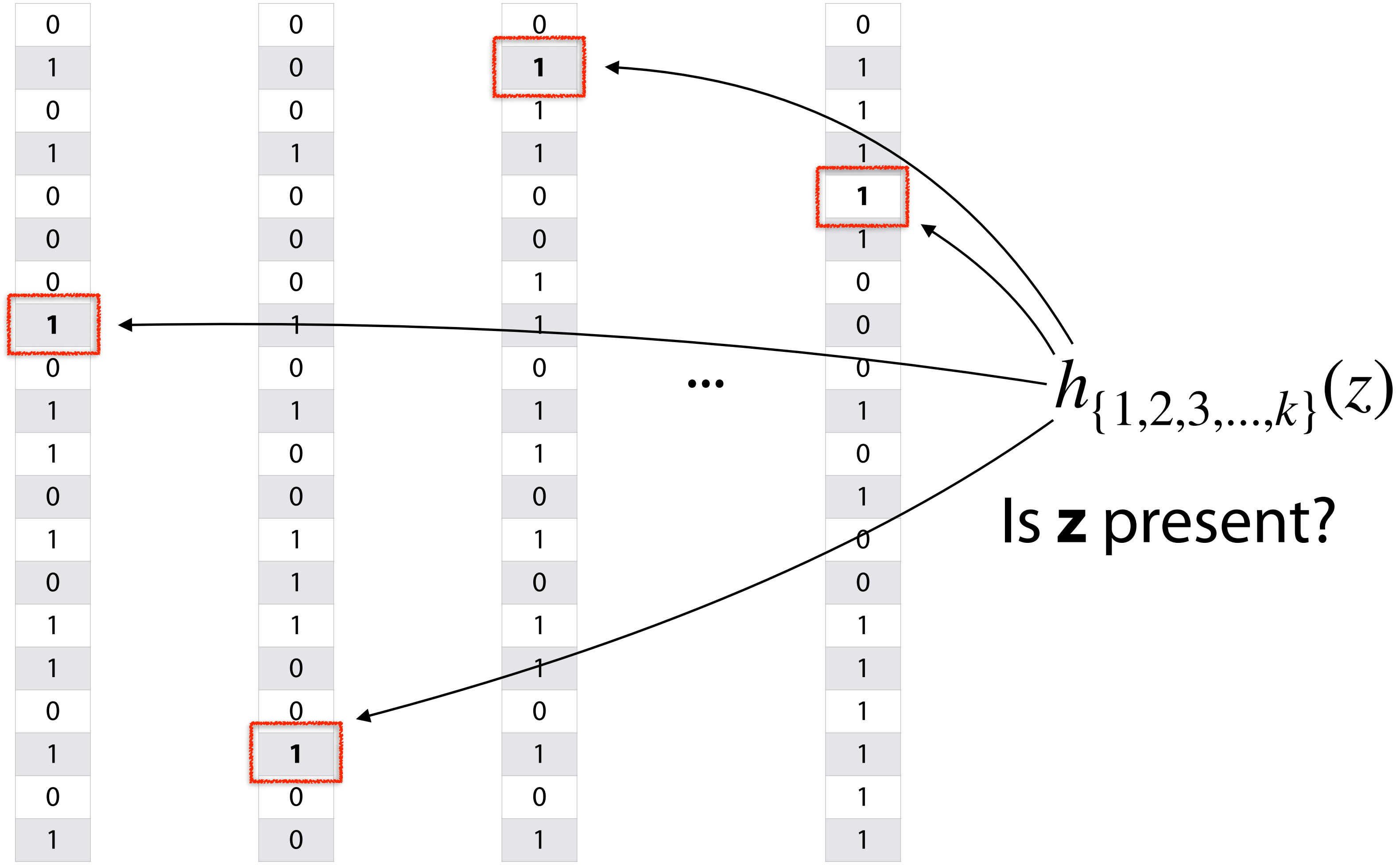


Is **y** present? No!

If *any* queries yield 0, item is 100% not present!

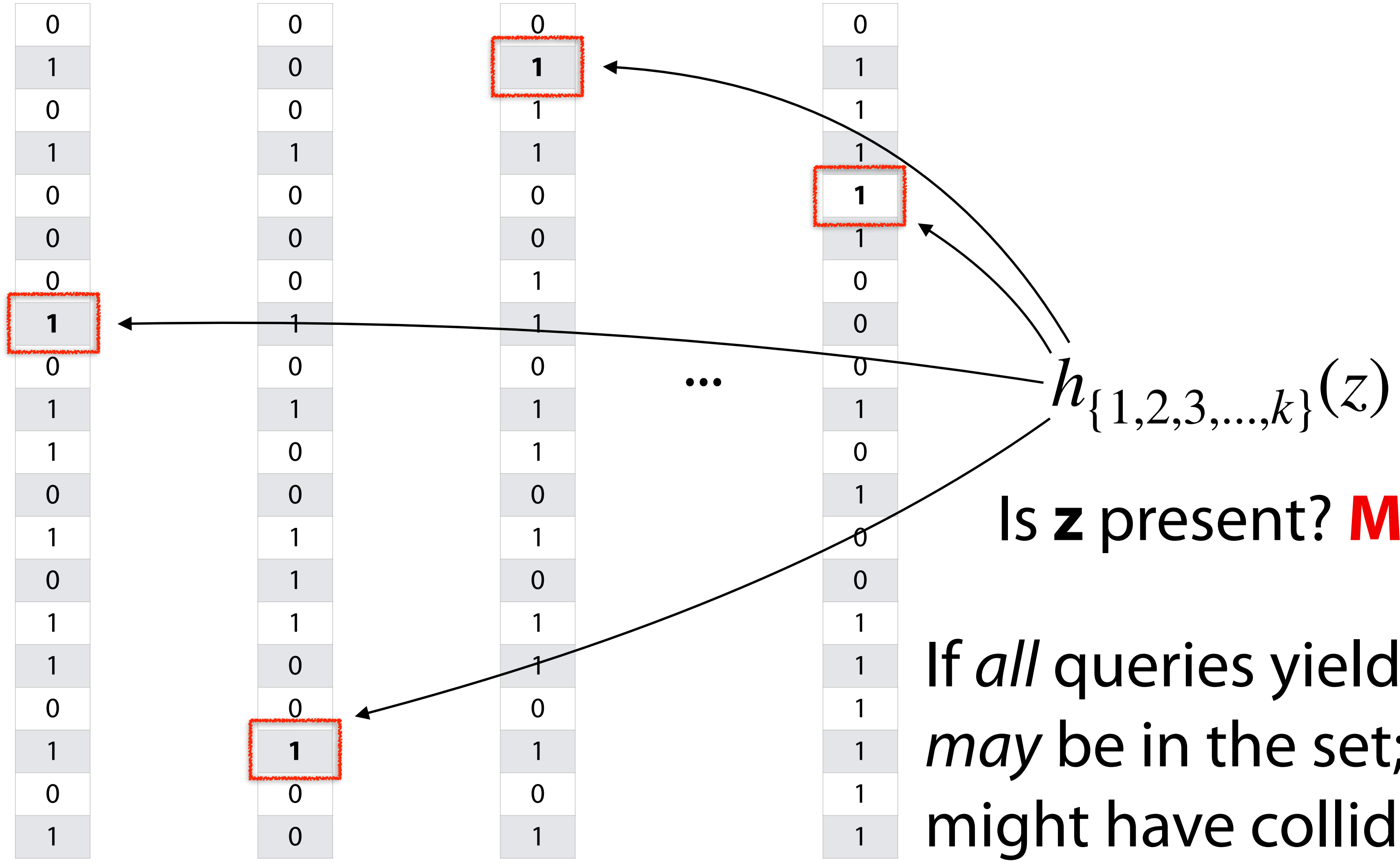
Bloom Filter: Repeated Trials

Each of these k Bloom Filters is a repeated trial — improved accuracy!



Bloom Filter: Repeated Trials

Each of these k Bloom Filters is a repeated trial — improved accuracy!



Is **z** present? **Maybe!**

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?

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	

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

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

S = { 6, 8, 4 }

$h_1(x) = x \% 10$	$h_2(x) = 2x \% 10$	$h_3(x) = (5+3x) \% 10$
6	2	3
8	6	9
4	8	7

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: ___________________

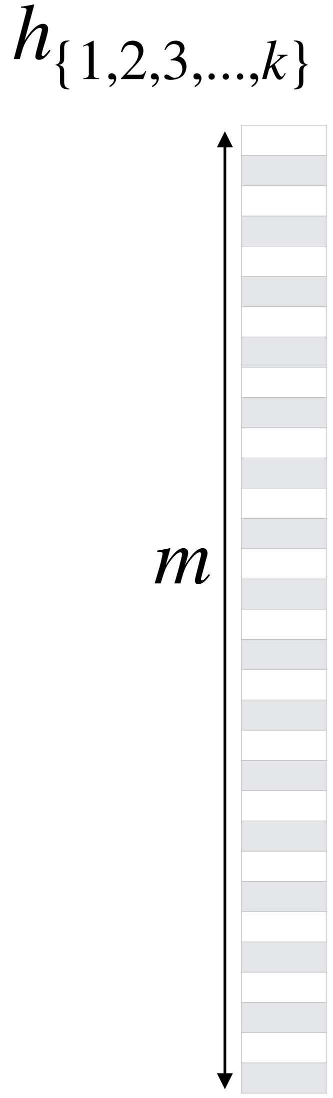
Delete is not possible (yet)!

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?

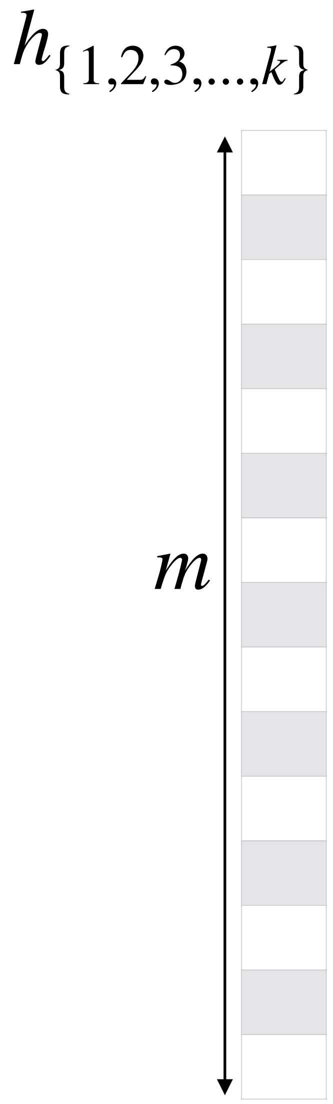


Bloom Filter: Error Rate

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?

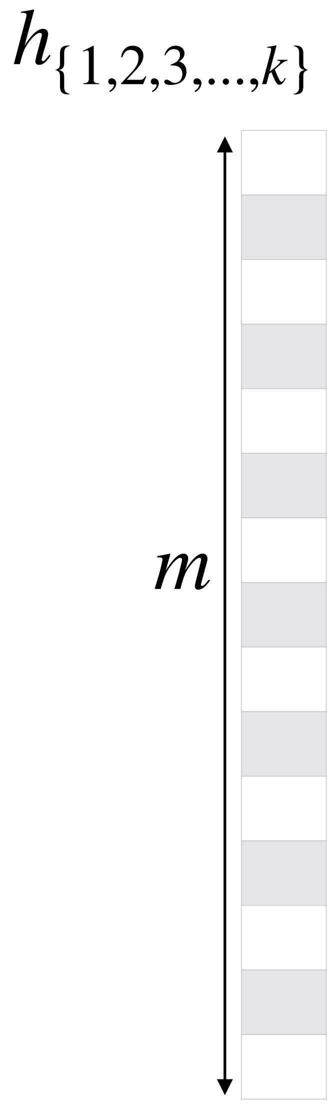


Bloom Filter: Error Rate

Given bit vector of size m and 1 SUHA hash function

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

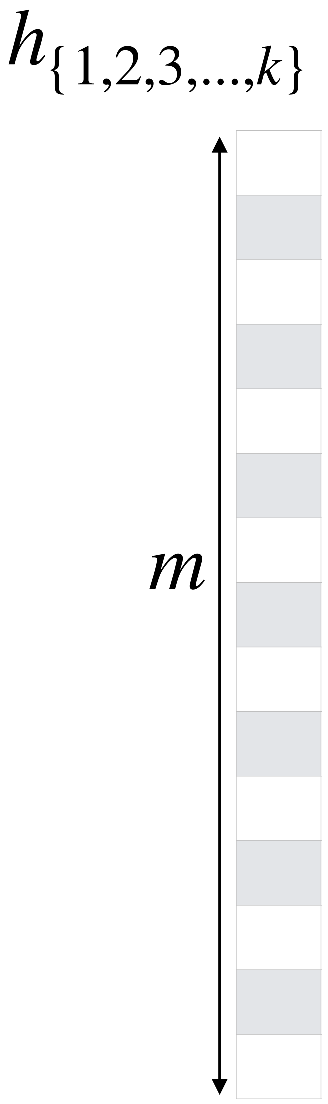
After n objects are inserted?



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

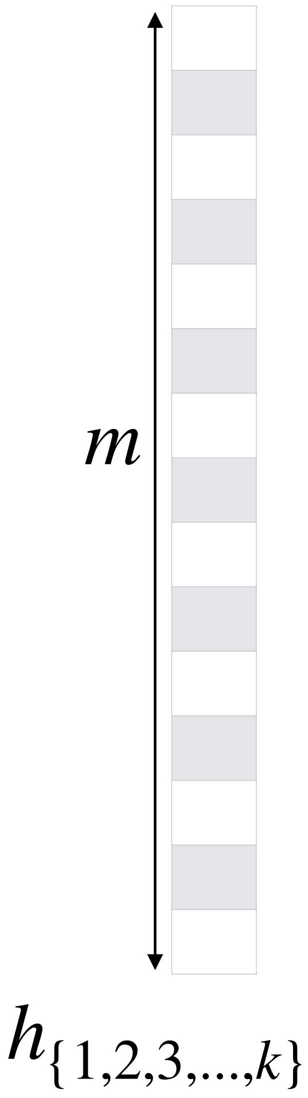
Given bit vector of size m and k SUHA hash function

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



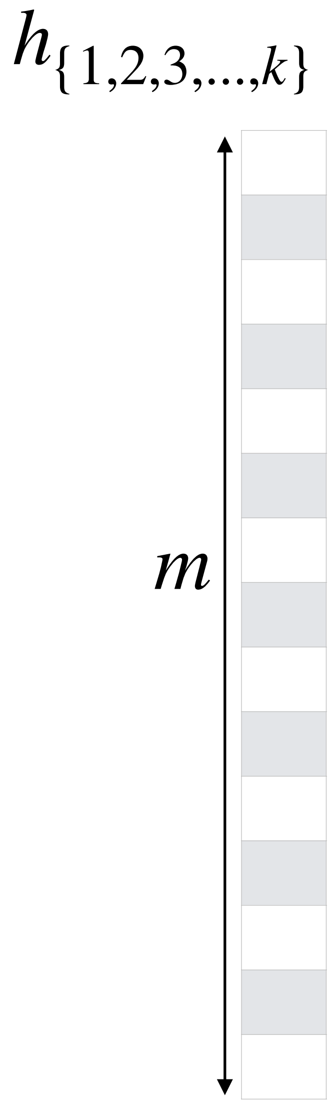
Bloom Filter: Error Rate

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

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

Bloom Filter: Optimal Error Rate

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

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

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

Bloom Filter: Optimal Error Rate

Taylors expansion of $ln(1+x)$: $$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots$$

"Mercator Series"

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

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

Bloom Filter: Error Rate

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

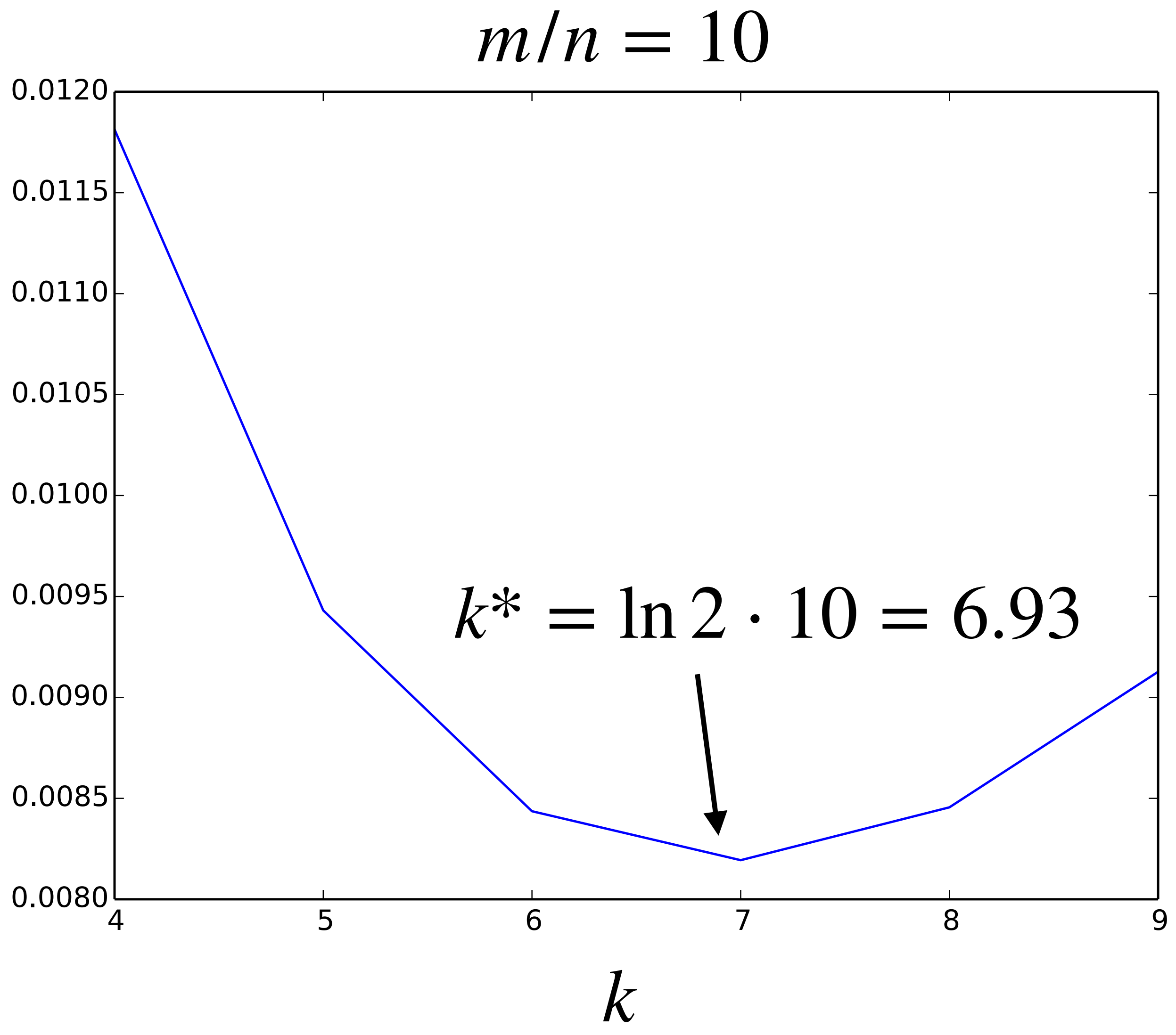


Figure by Ben Langmead

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 $\quad m =$

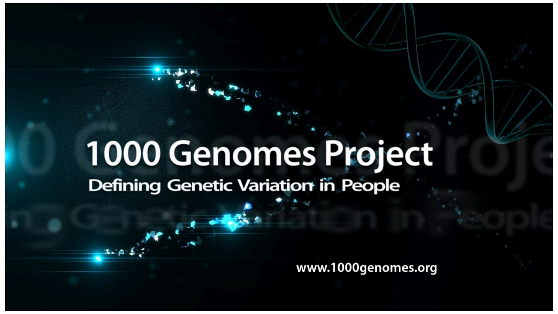
$m = 100$ bits $\quad n = 20$ items $\quad k =$

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

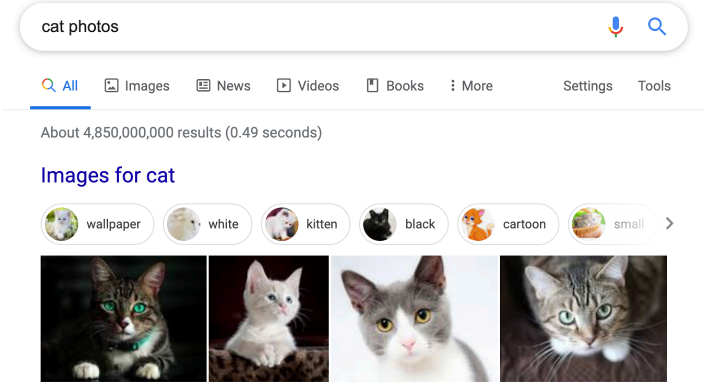
Bloom Filter: Optimal Parameters

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

Optimal hash function is still O(m)!

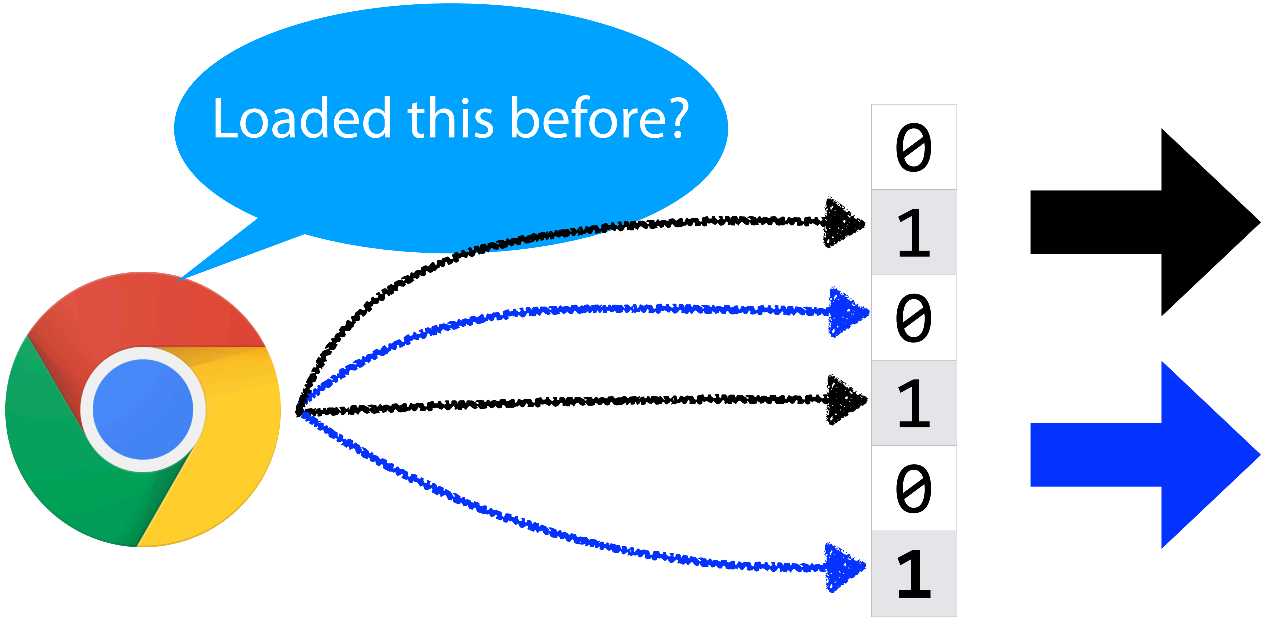


n = 250,000 files vs ~10^{15} nucleotides vs 260 TB



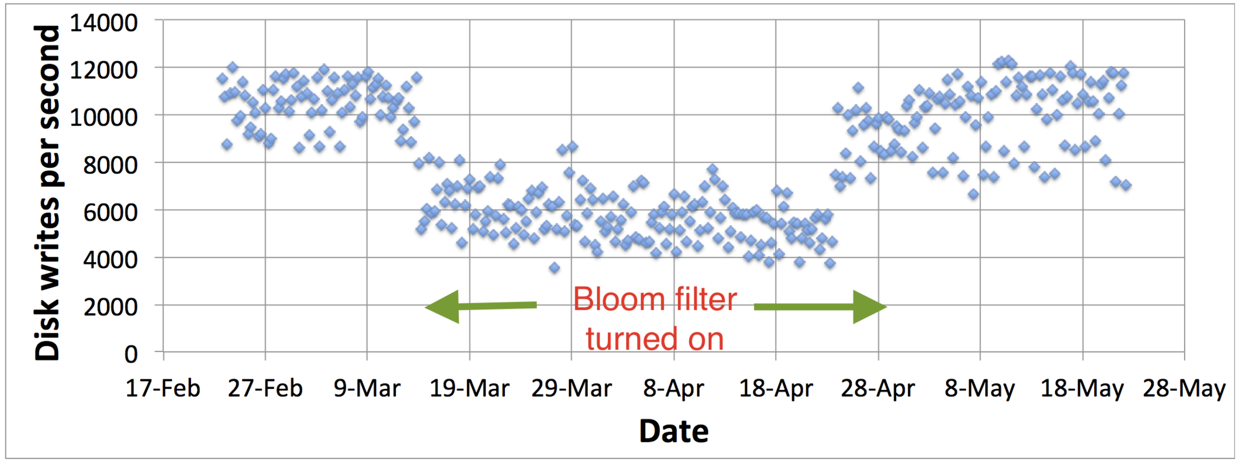
n = 60 billion — 130 trillion

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++

How can we encode a bit vector in C++?

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

Bloom Filters: Unioning

Bloom filters 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	∪	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	

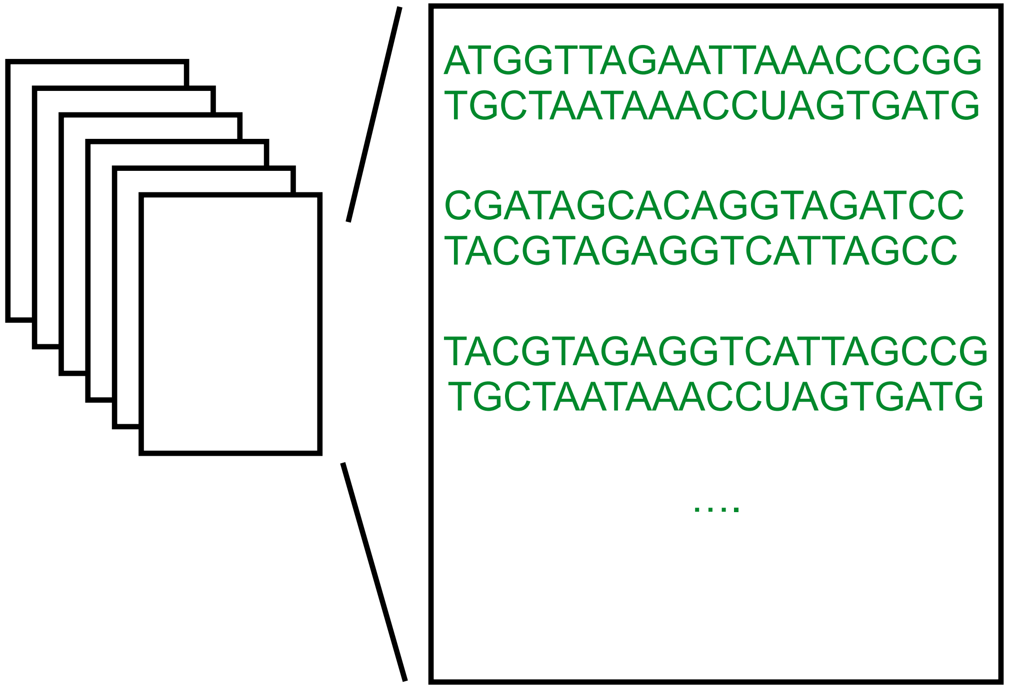
Bloom Filters: Intersection

Bloom filters can be trivially merged using bit-wise intersection.

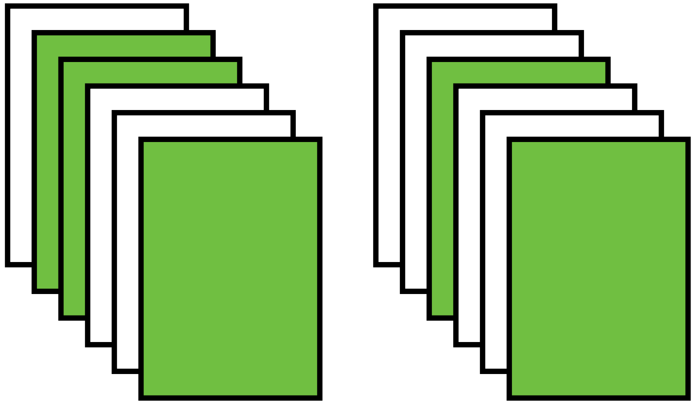
0	1		0	0		0	
1	0		1	1		1	
2	1		2	1		2	
3	1		3	0		3	
4	0	U	4	0	=	4	
5	0		5	0		5	
6	1		6	1		6	
7	0		7	1		7	
8	0		8	1		8	
9	1		9	1		9	

Sequence Bloom Trees

Imagine we have a large collection of text…

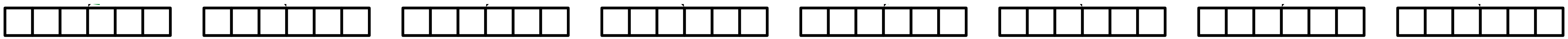


And our goal is to search these files for a query of interest…



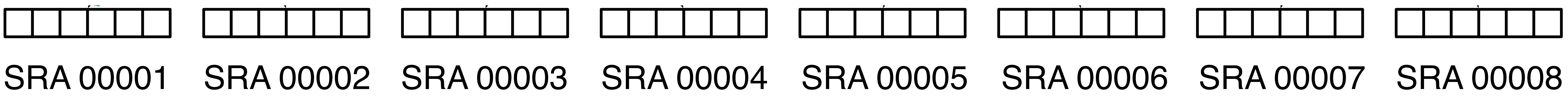
Bit Vector Merging

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

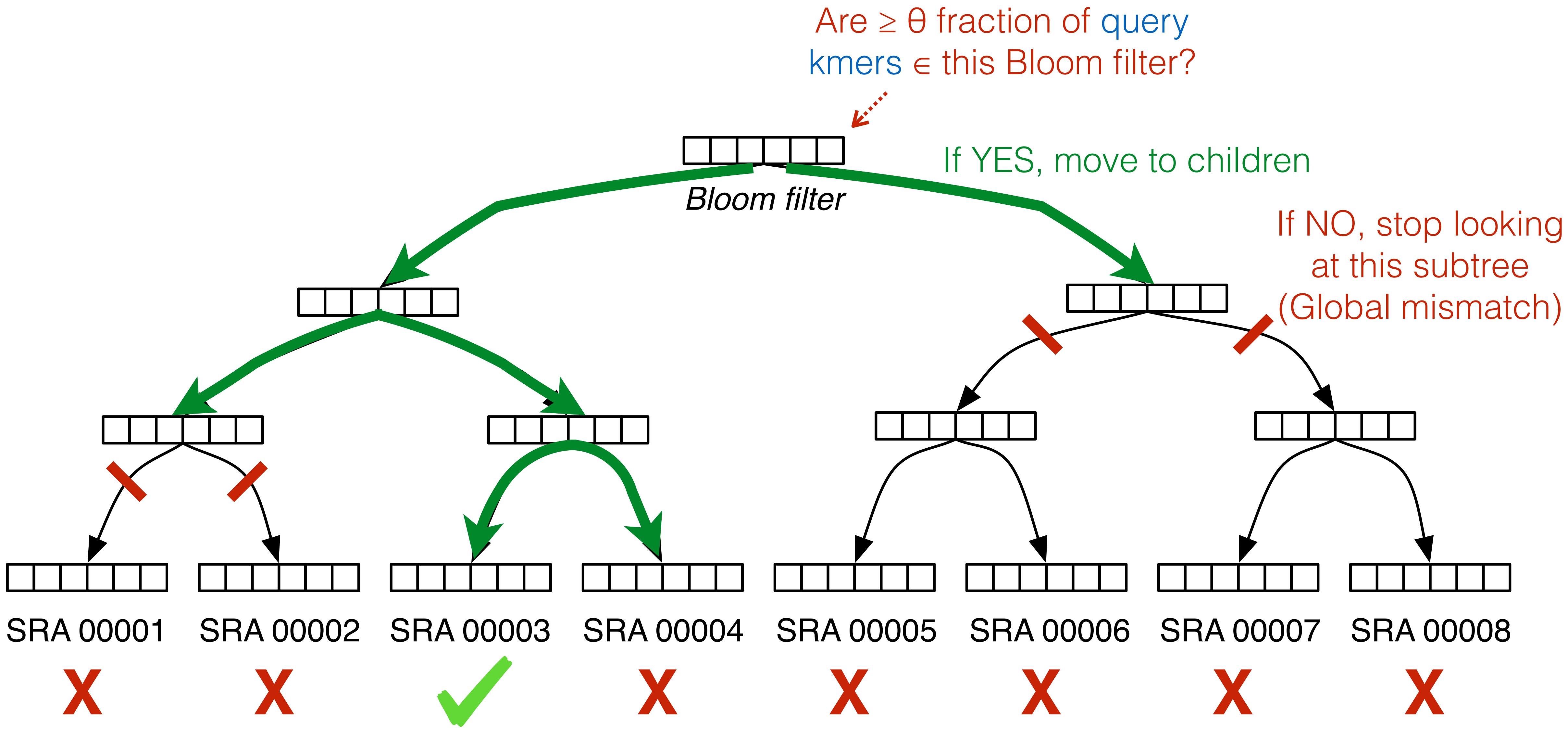


SRA 00001 SRA 00002 SRA 00003 SRA 00004 SRA 00005 SRA 00006 SRA 00007 SRA 00008

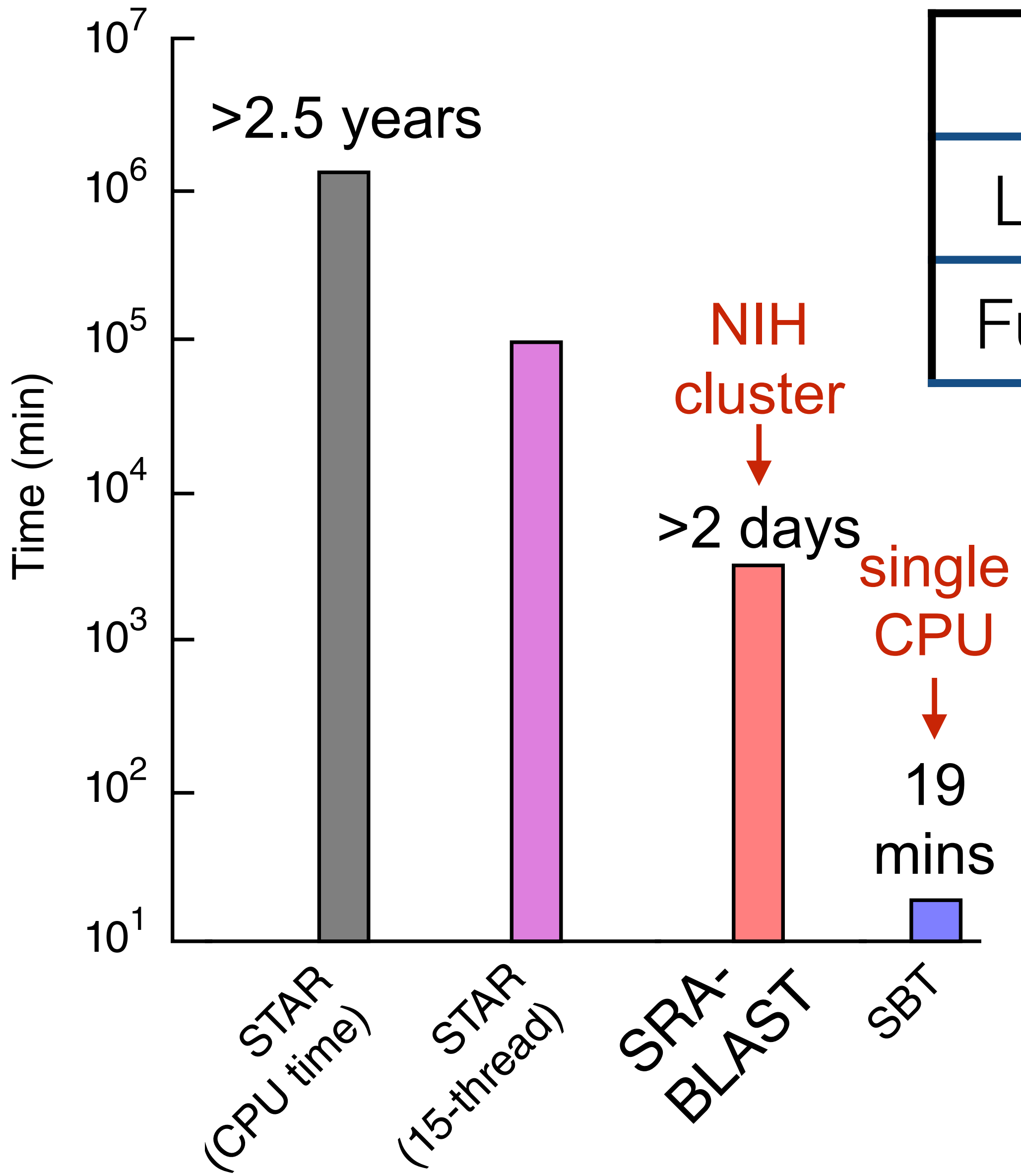
Sequence Bloom Trees



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…