Data Structures and Algorithms
Bloom Filters 2

CS 225
Brad Solomon

November 21, 2025

Have a great fall break! :)

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)

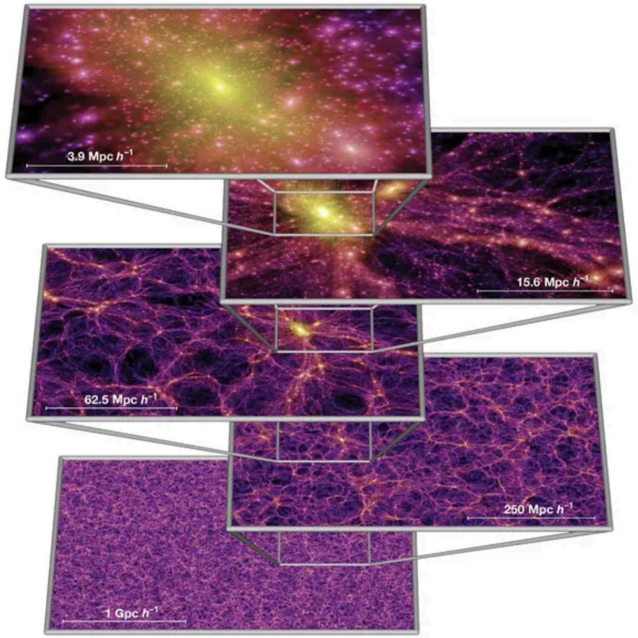


Image: https://doi.org/10.1038/nature03597

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

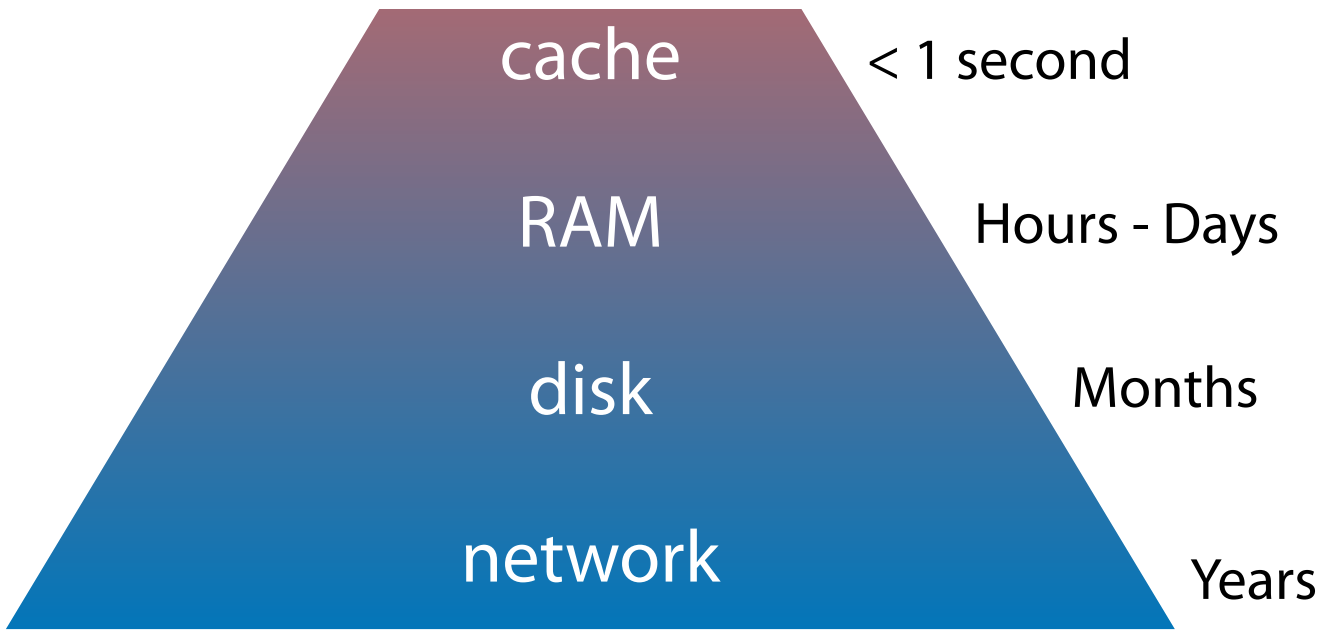
Table: http://doi.org/10.5334/dsj-2015-011

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)

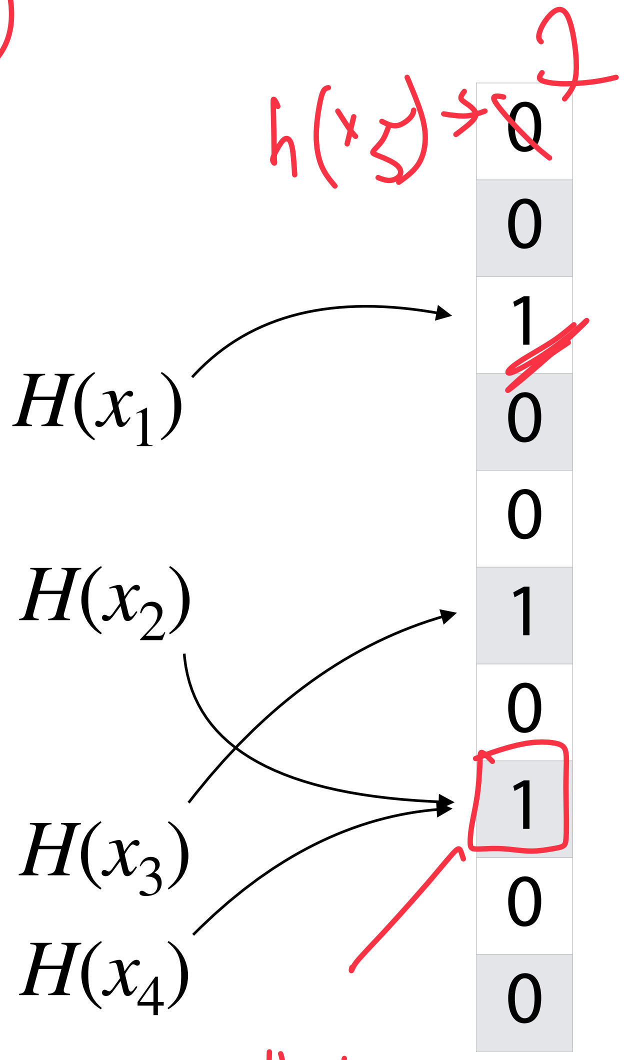
Bloom Filter: Insertion

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

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

If the bit was already one, it stays 1

1 bit per register



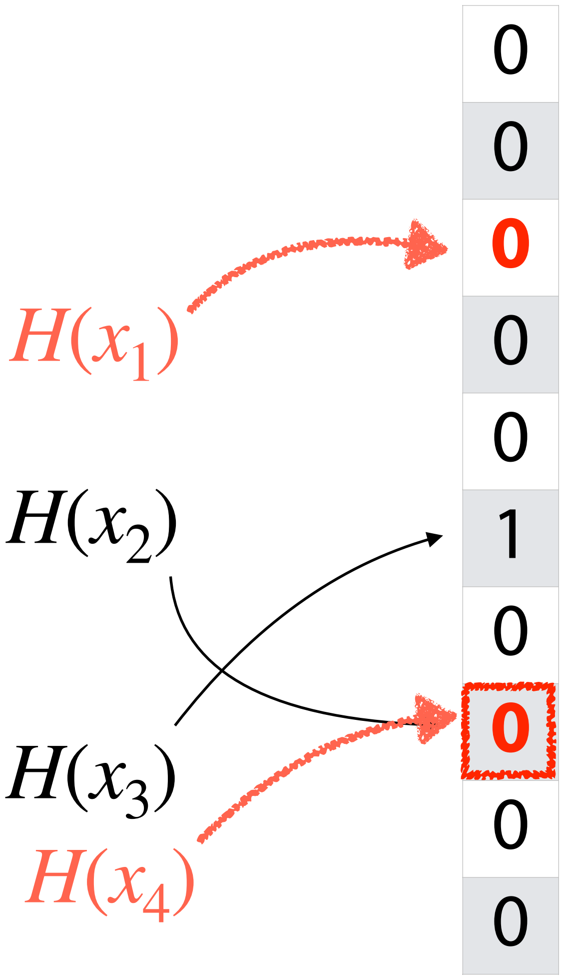
could be x2, x4, ...

Non-reversible
(Trade off)

Bloom Filter: Deletion

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

Tradeoff



Bloom Filter: Search

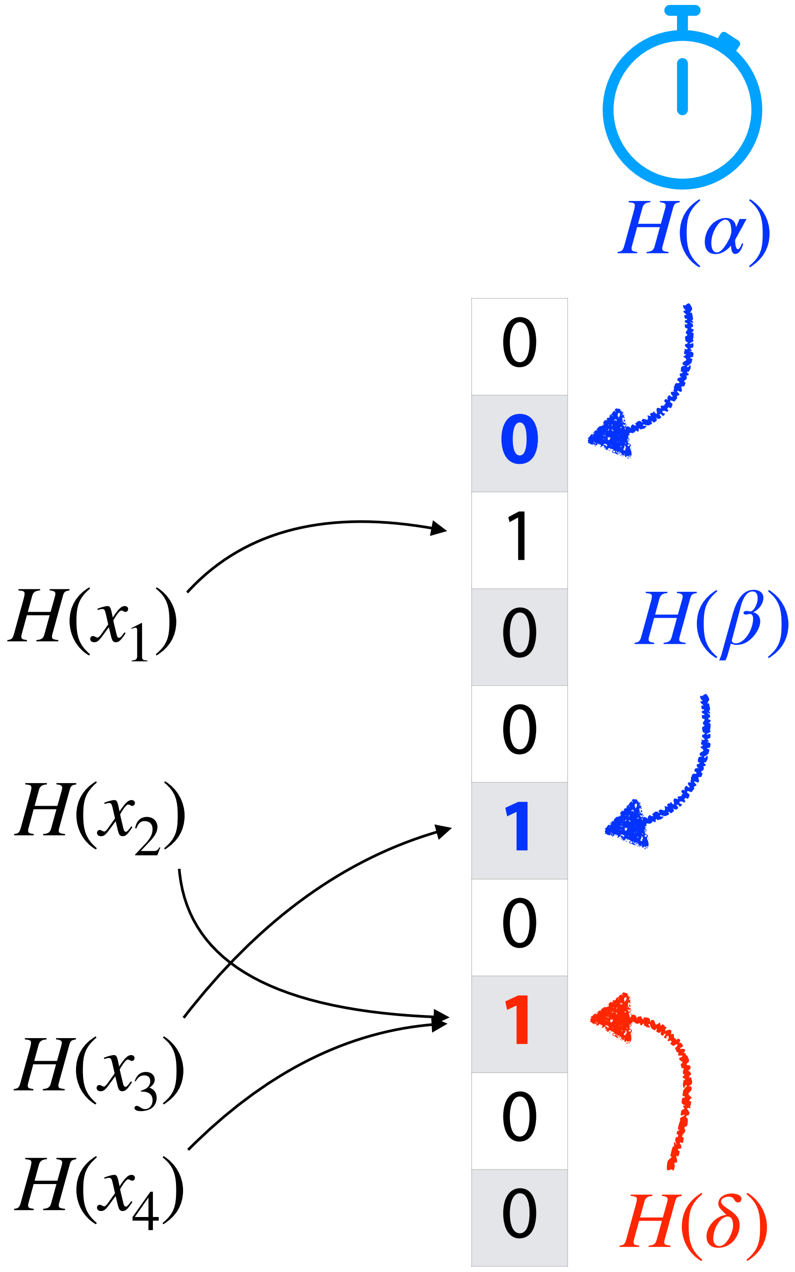
The bloom filter is a *probabilistic* data structure!

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



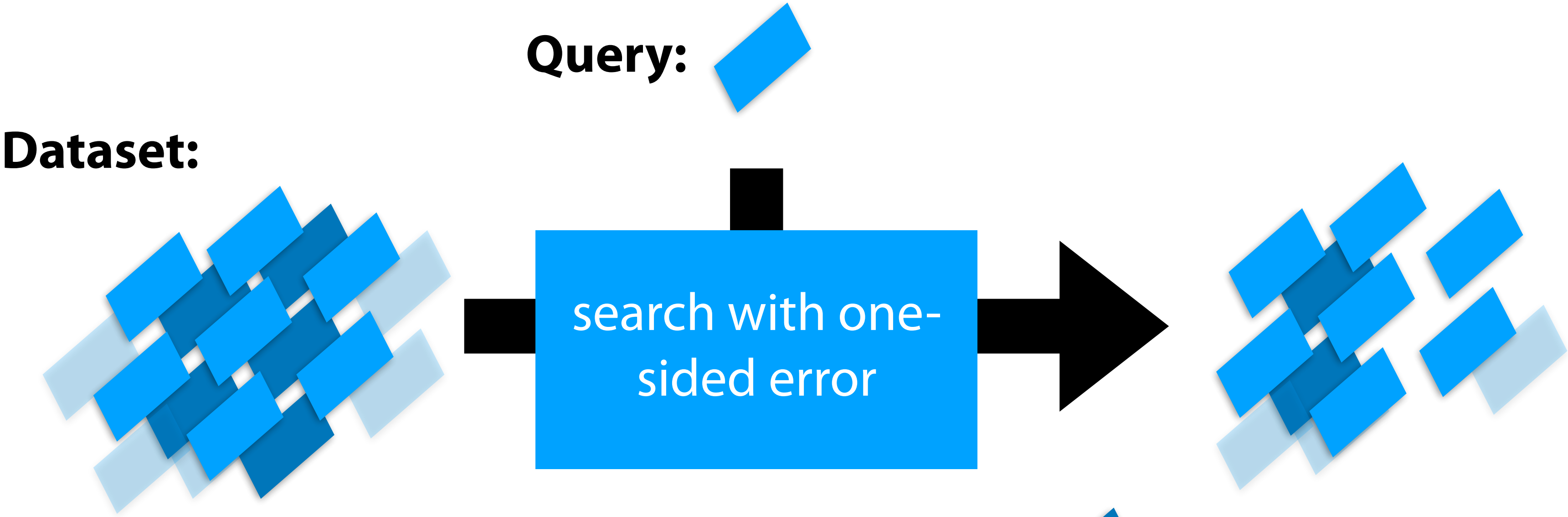
Probabilistic Accuracy in a Bloom Filter

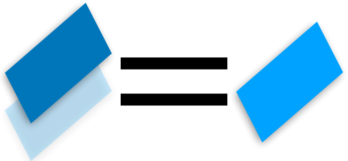
	given Bit Value = 1	Bit Value = 0
Item Inserted 100%	$H(z)$ 0 1 0 0 1 'Yes' True Positive	$H(z)$ 0 0 0 0 1 'No' False Negative
FPR Item NOT inserted ~~Assume~~ given	0 1 0 0 1 'Yes' False Positive	0 0 0 0 1 'No' True Negative

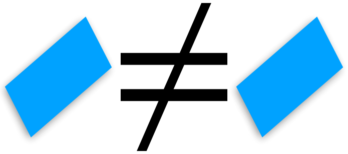
No FN in BF

100%

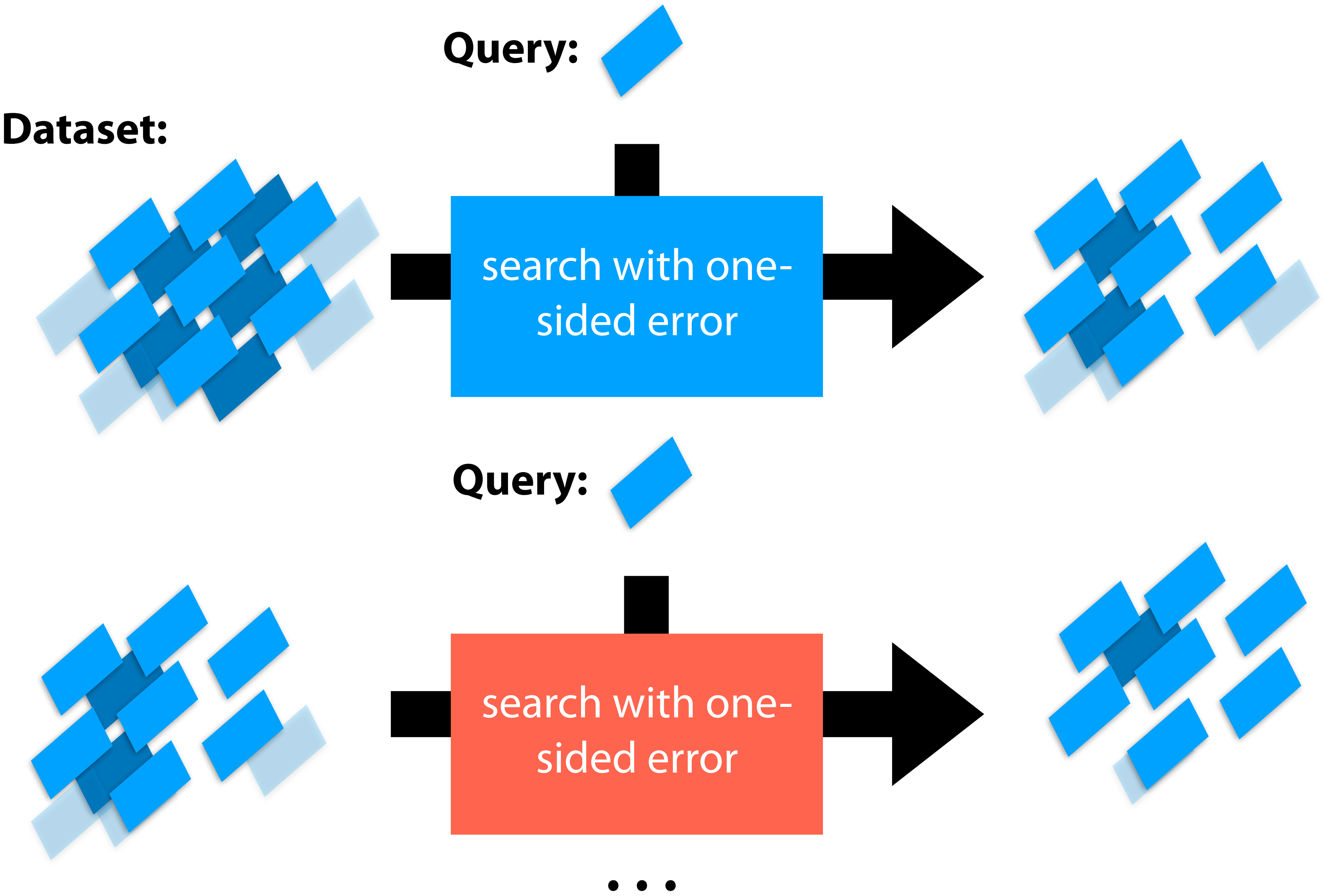
Probabilistic Accuracy: One-sided error



We will get some False Positives: 

We will NEVER have a False Negative: 

Probabilistic Accuracy: One-sided error

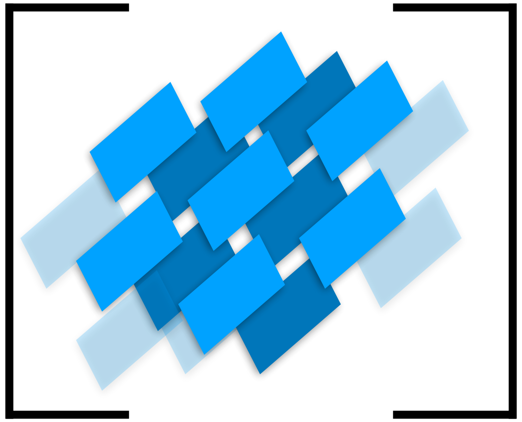


Bloom Filter: Repeated Trials

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

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

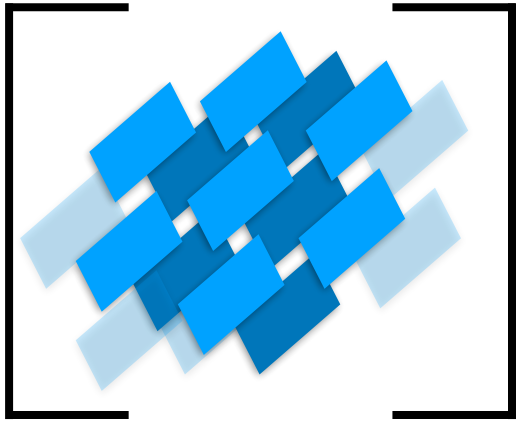
h_1



Bloom Filter: Repeated Trials

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

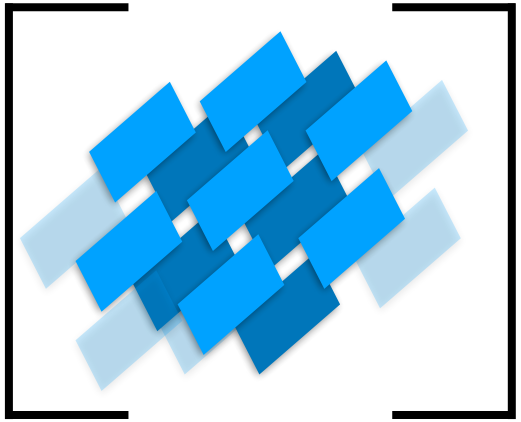
h_1	h_2
0	0
1	0
0	0
1	**1**
0	0
0	0
0	0
1	**1**
0	0
1	**1**
1	0
0	0
1	**1**
0	**1**
1	**1**
1	0
0	0
1	**1**
0	0
1	0



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

k Different hash functions

Bloom Filter: Repeated Trials

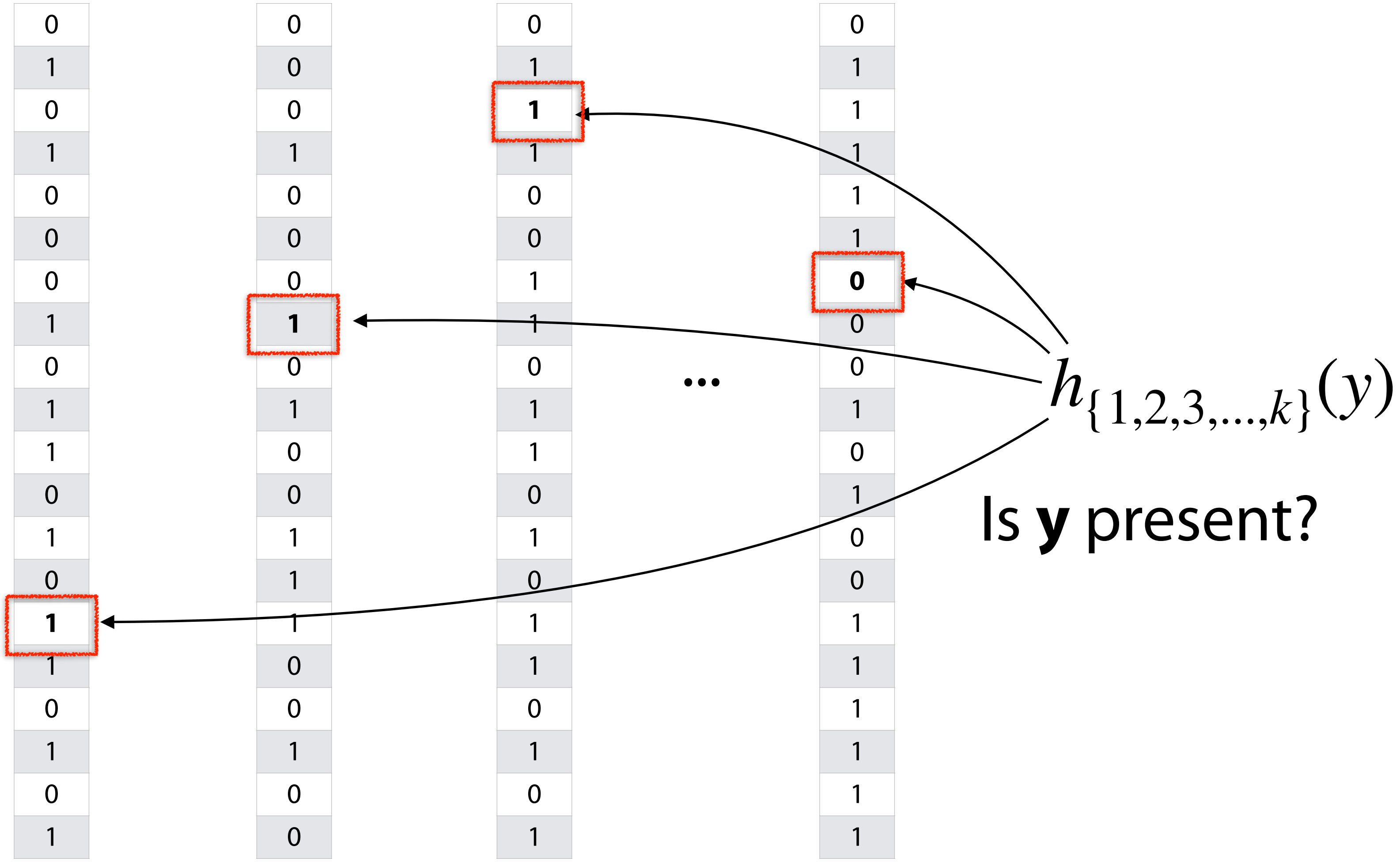
Each of these k Bloom Filters is a repeated trial — improved accuracy!

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

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

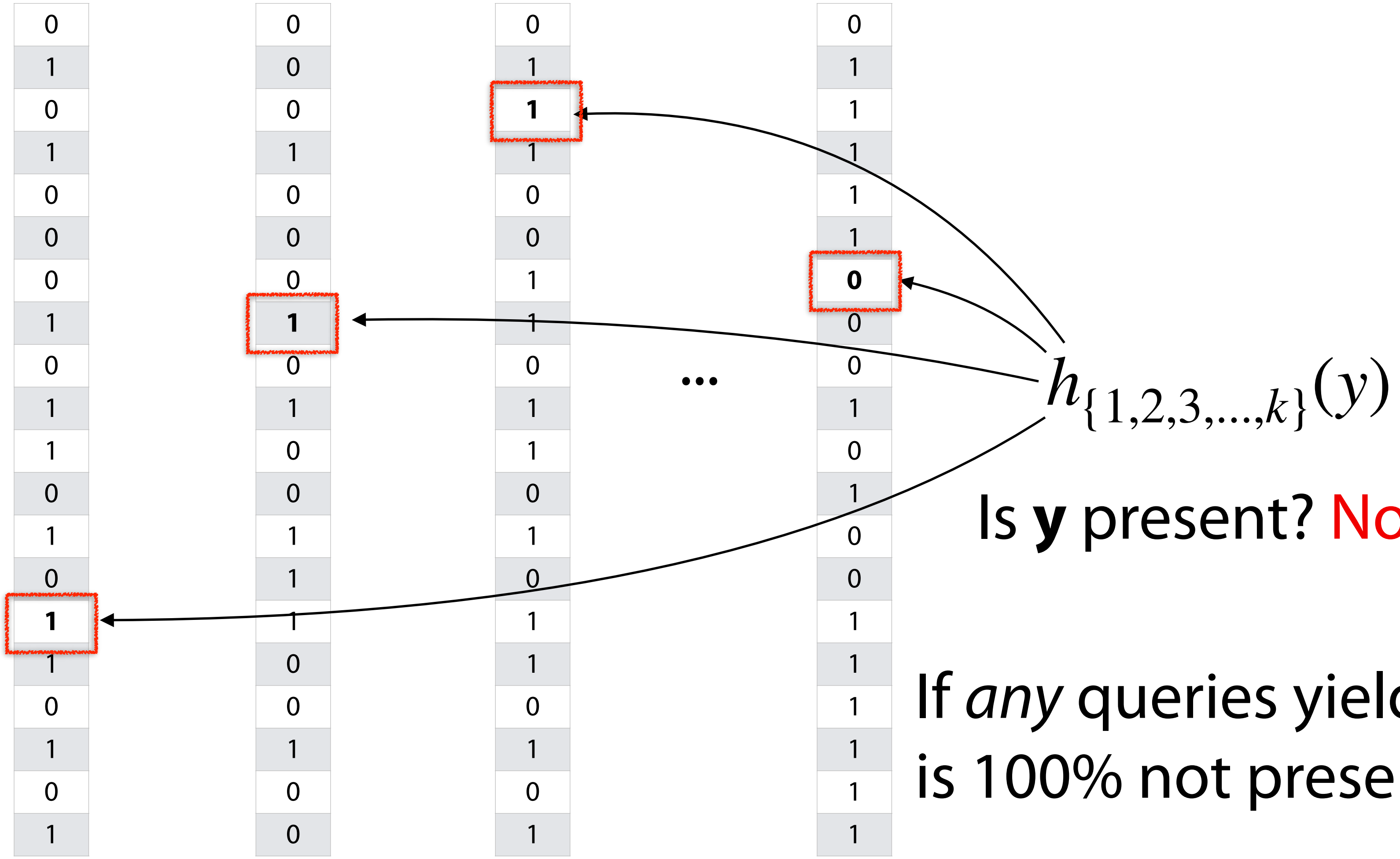
Bloom Filter: Repeated Trials

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!

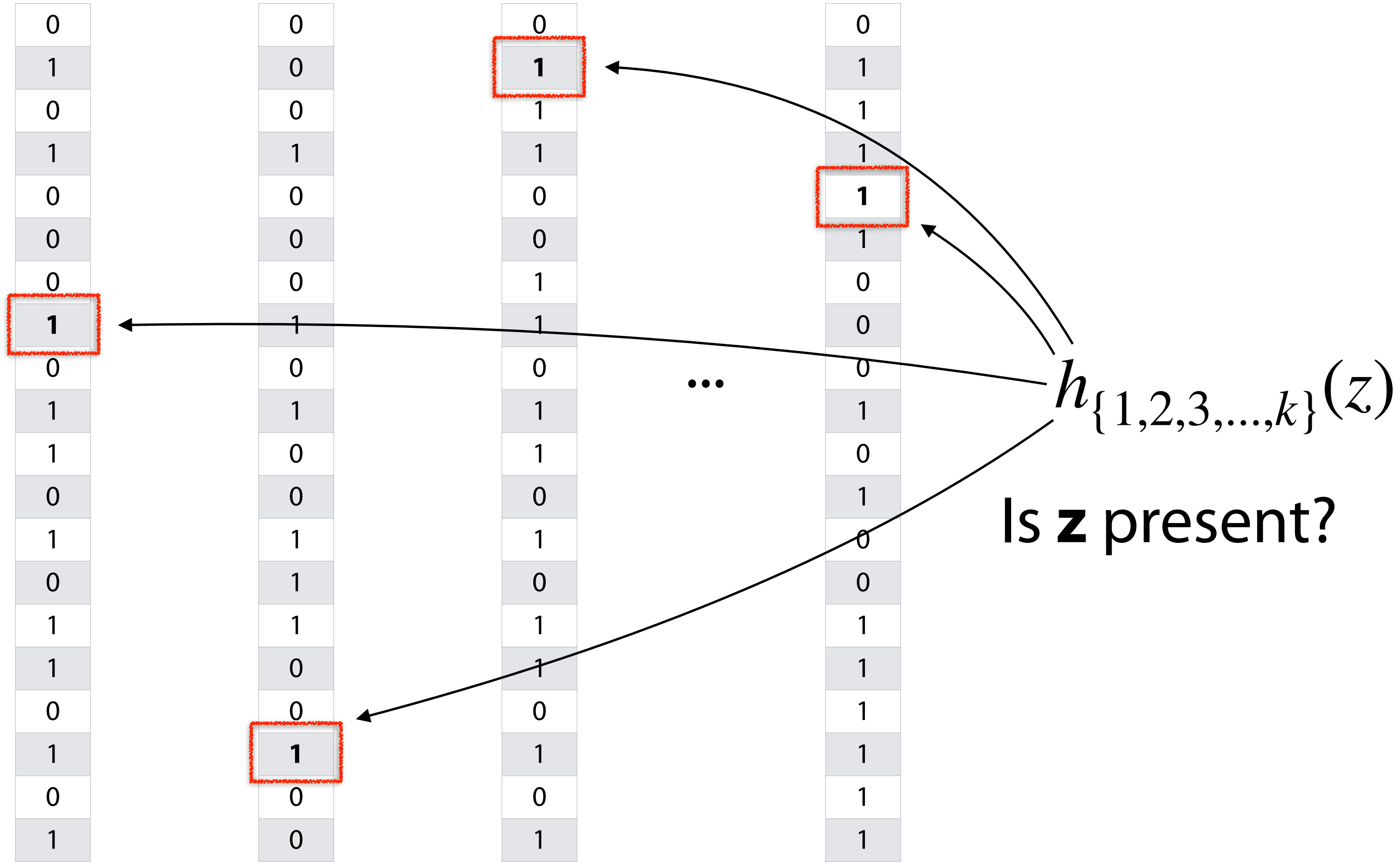


Is **y** present? No!

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

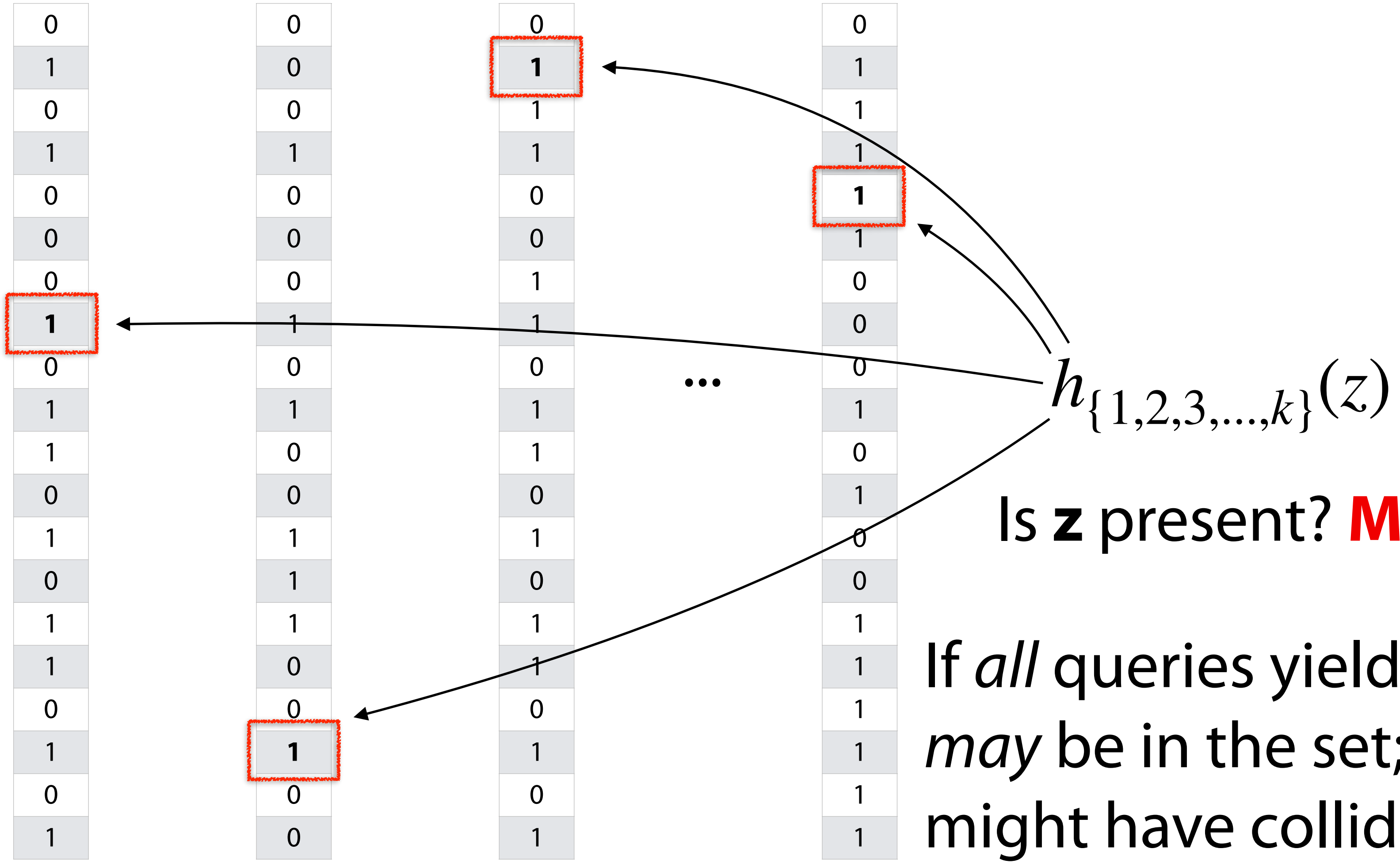
Bloom Filter: Repeated Trials

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?

X is not present

h_1

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	

Likelihood of collisions by chance is p^K

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

Don't need K filters!

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 }

Hash K times

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

what is the tradeoff here?

Larger FPR $\longleftrightarrow$ space savings

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)` (Hash k times, do k lookups)

1 2 8

B/c 0 @ index 1, 100% not present

`_find(16)`

6 2 3

B/c 1s at all positions, probably exists!

[This is false positive]

Bloom Filter

A probabilistic data structure storing a set of values

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

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

insert n items

0
0
1
0
0
1
0
1
0
0

Insert / Find runs in: $O(1) / O(1)$

$O(k) / O(k)$

More correct

Delete is not possible (yet)!

Bloom Filter: Error Rate

$h_{\{1,2,3,\ldots,k\}}$

Given bit vector of size m and k SUHA hash function

What is our expected FPR after n objects are inserted?

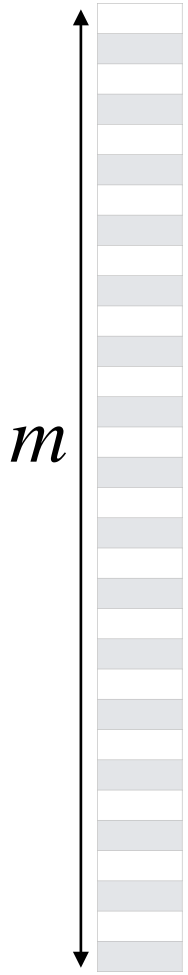
$\left(\text{Probability that a bit is 1 after } n \text{ insertions}\right)^{K}$

hard to write!

K

FPR

K random independent trials



Bloom Filter: Error Rate

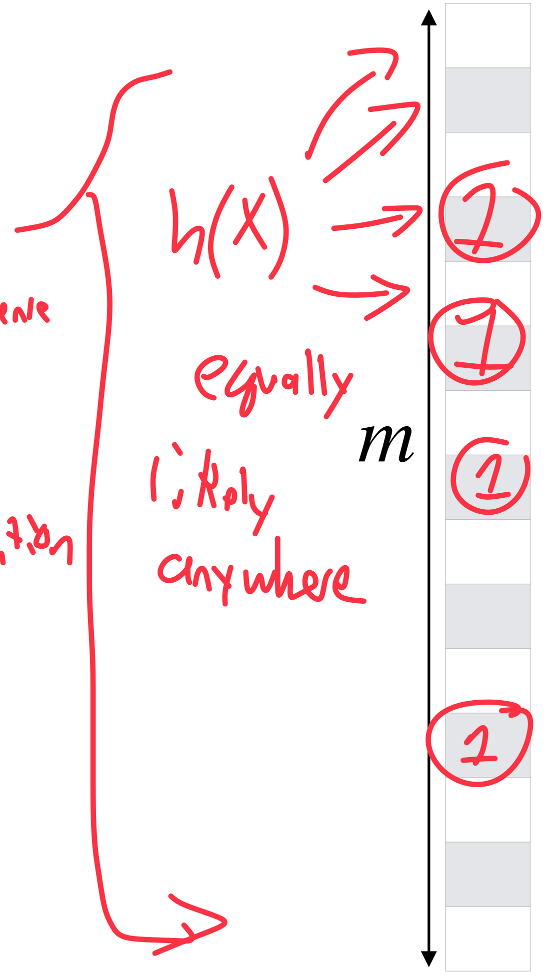
$h_{\{1,2,3,\ldots,k\}}$

Given bit vector of size m and 1 SUHA hash function

SUHA
↳ uniform
↳ independence

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

$1/m$ from uniform distribution



Same probability given k SUHA hash function?

$(1/m)^k$ ← No! Bad! Incorrect!

Tip/Trick: If writing equation difficult, invert it!

Bloom Filter: Error Rate

$h_{\{1,2,3,\ldots,k\}}$

Given bit vector of size m and ~~1~~ k SUHA hash function

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

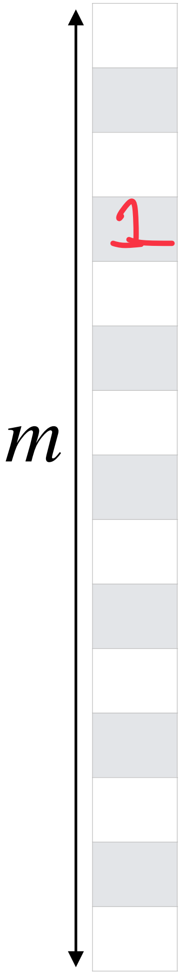
$$1 - \frac{1}{m}$$

After n objects are inserted?

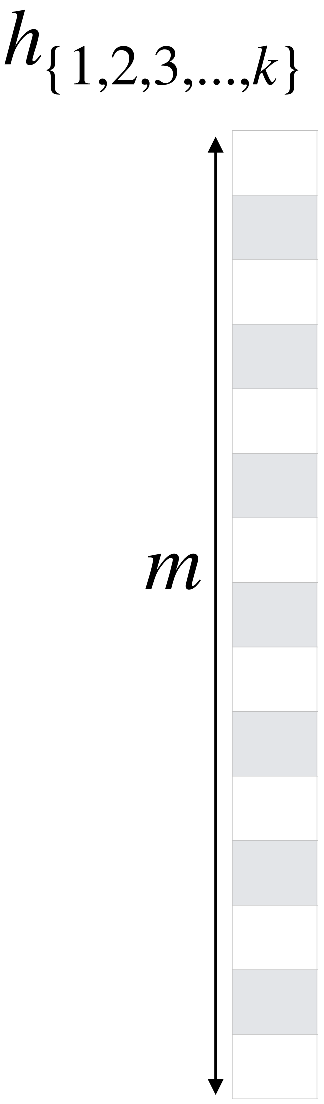
$$\left(1 - \frac{1}{m}\right)^{nk}$$

$n \cdot k$

$n \cdot k$ independent uniform random trials



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?

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

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?

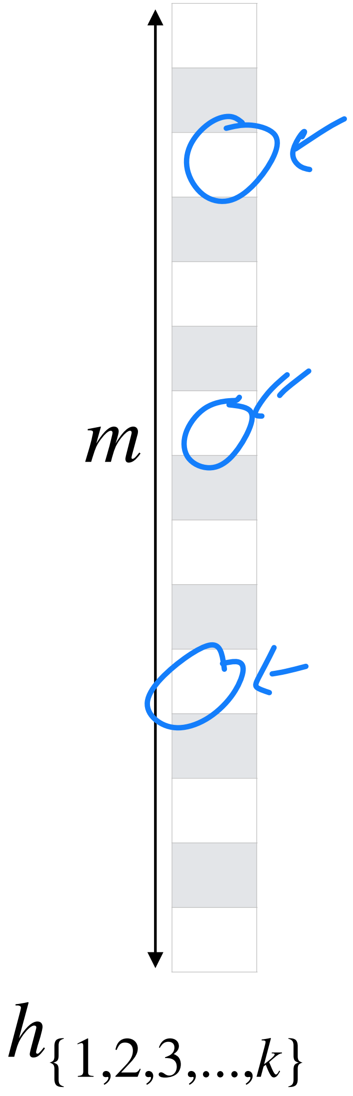
FPR for single register

The probability my bit is 1 after n objects inserted

we look up k positions

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

The number of [assumed independent] trials



Bloom Filter: Error Rate

$h_{\{1,2,3,\ldots,k\}}$

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

To minimize the FPR, do we prefer…

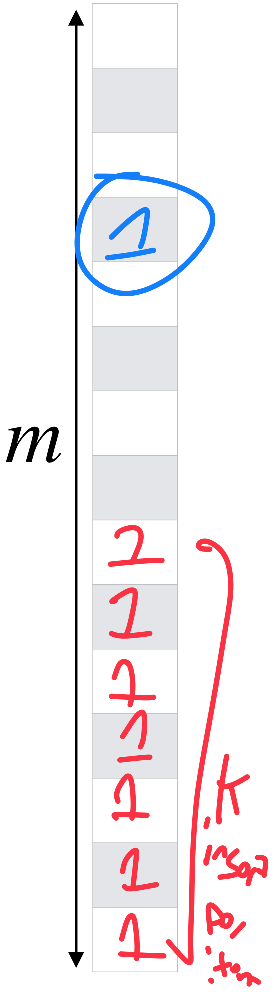
(A) large k
70%

(B) small k
30%

$$\left(1-\underbrace{\left(1-\frac{1}{m}\right)}_{\text{This is c1}}{}^{nk}\right)^{k}$$

repeated random trials

Taking that value to $(x)^{nk}$ reduces # 1s in my filter!



Bloom Filter: Error Rate

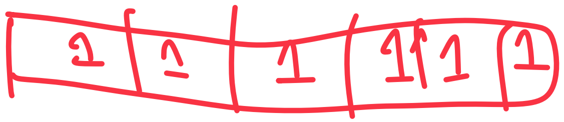
Vector of size m, k SUHA hash function, and n objects

(A) large k

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

As k increases, this gets smaller!

The # of 1s in filter gets larger



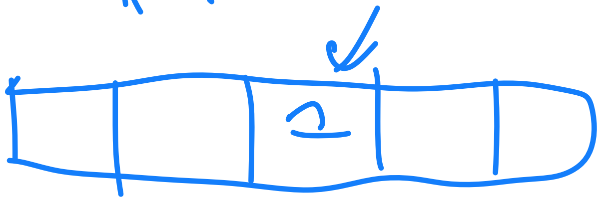
(B) small k

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

As k decreases, this gets smaller!

↳ Less random trials ☹

$k=1$



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

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

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

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"

$\ln\left(1 - \frac{1}{m}\right) \rightarrow x = -\frac{1}{m}$

$-\frac{1}{m} + \frac{\frac{1}{m}^2}{2} - \frac{\frac{1}{m}^3}{3} + \ldots$

Small & gets smaller !!

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

Bloom Filter: Optimal Error Rate

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

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

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

$$-\frac{1}{m}\cdot nk$$

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

Bloom Filter: Optimal Error Rate

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

Math here not important

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

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

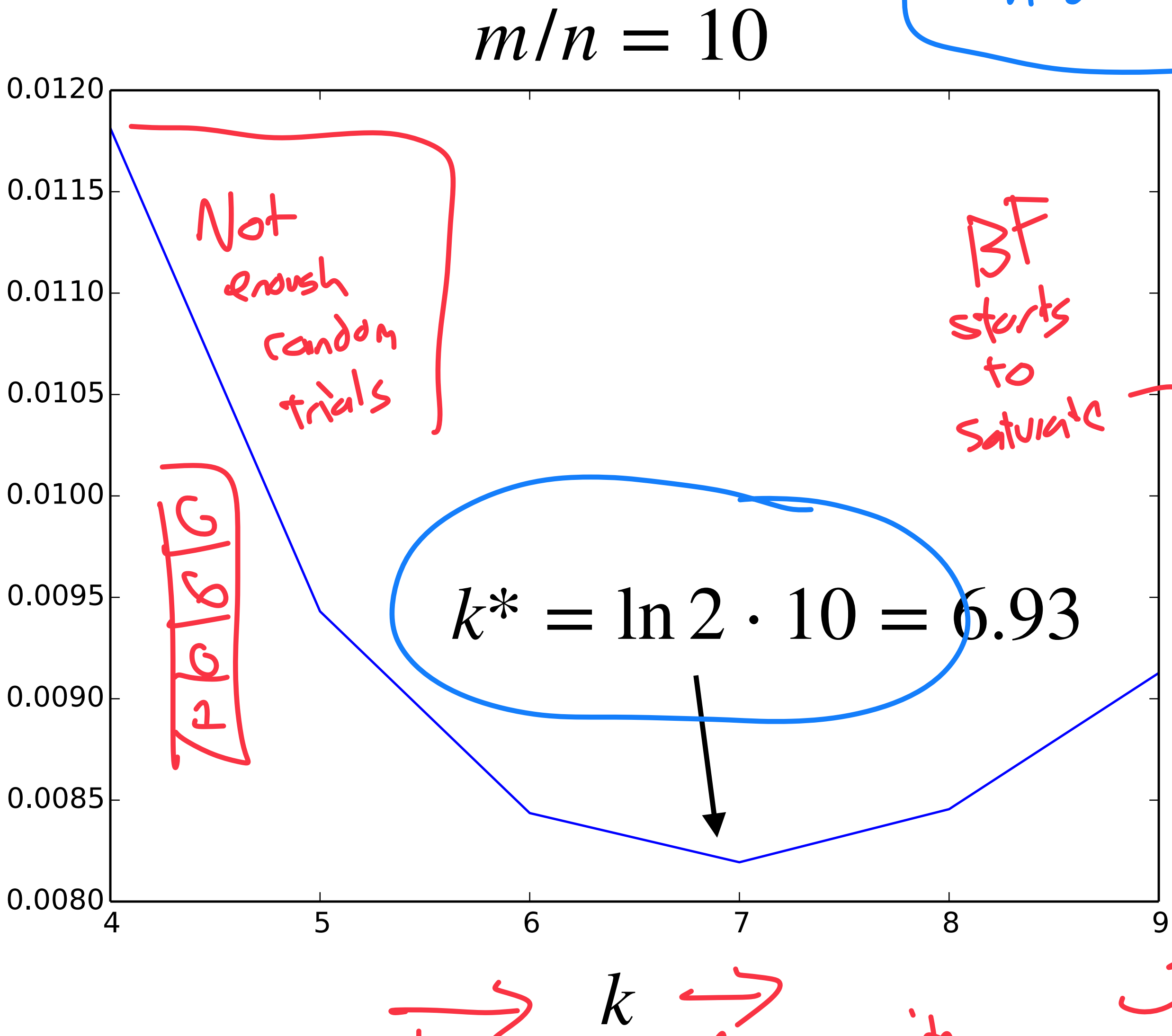
Derivative is zero when $k^* = \ln 2 \cdot \dfrac{m}{n}$

Bloom Filter: Error Rate

k^* = optimal # hashes

$\ln 2 \cdot \frac{m}{n}$

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



inserts more 1s per item

results more registers

Figure by Ben Langmead

Bloom Filter: Optimal Parameters

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

Given any two values, we can optimize the third

$n = 100$ items $\qquad k = 3$ hashes $\qquad m = \sim 433$ bits

$m = 100$ bits $\qquad n = 20$ items $\qquad k = \sim 3.47 = 4$ hash

3

$m = 100$ bits $\qquad k = 2$ items $\qquad n = 34.7$ items ≈ 34 items

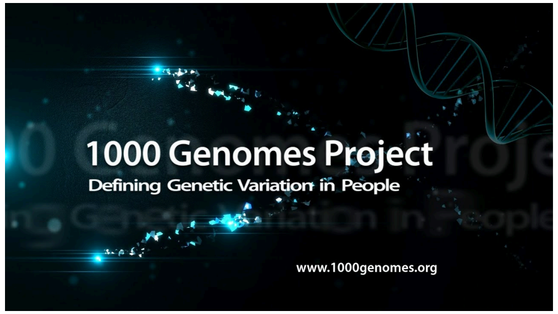
35

Bloom Filter: Optimal Parameters

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

bits per item

Optimal hash function is still O(m)!



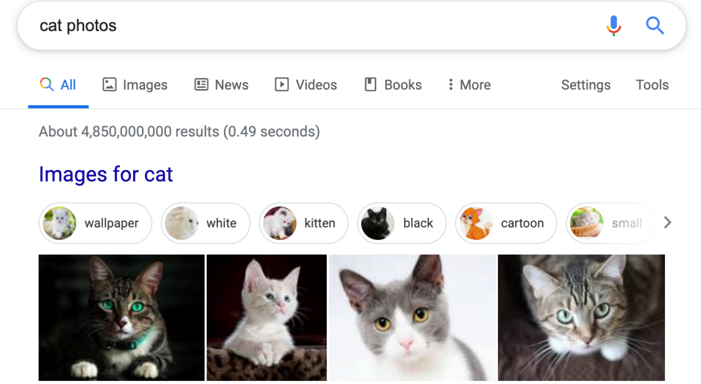
n = 250,000 files vs ~10^{15} nucleotides vs 260 TB

1 BF inserting files

BF inserting strings

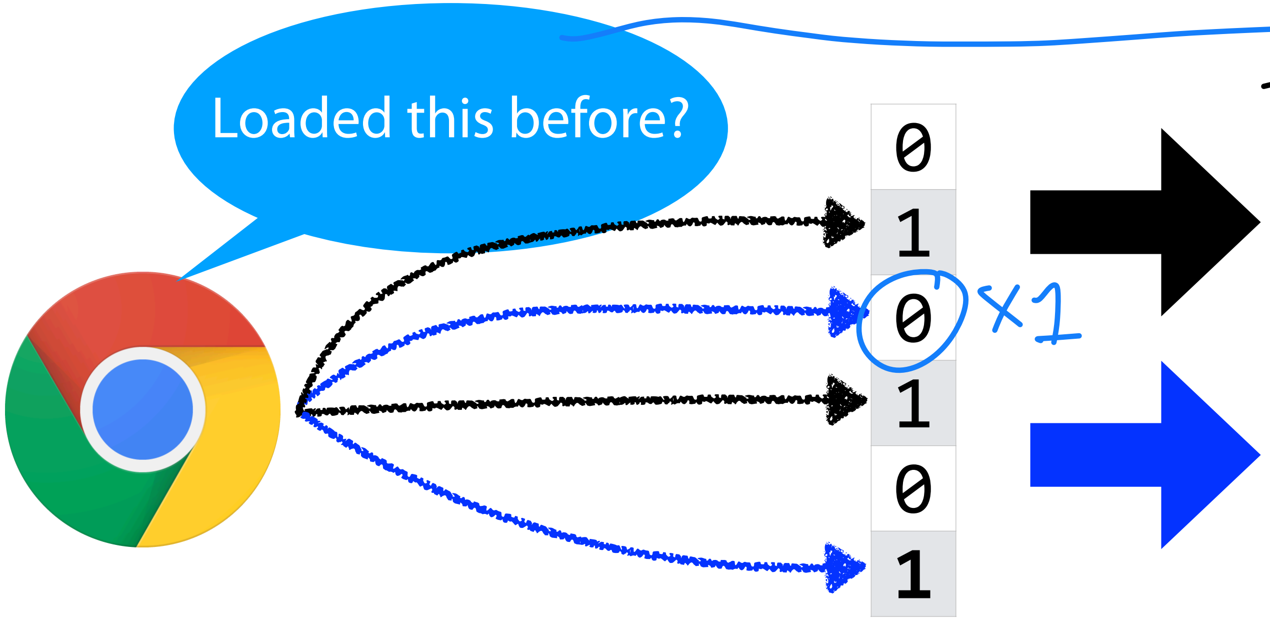
extremely small BF

10^{15} bits



n = 60 billion — 130 trillion

Bloom Filter: Website Caching



If in BF

(cache on and view)

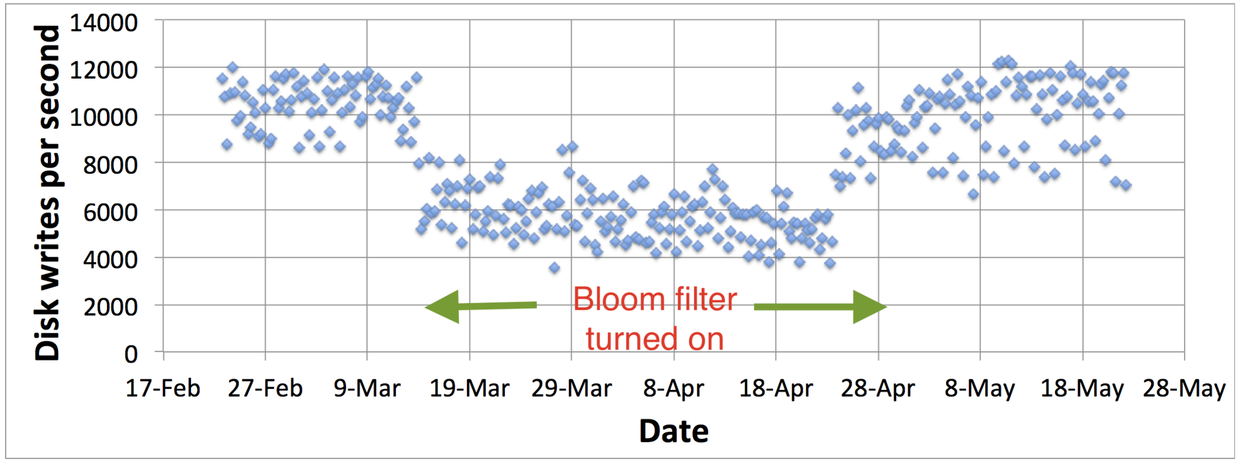
Cache this page!

Add to filter (but don't cache!)

(error)

~~BF anyway~~

is relatively low cost



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.

"or"

0	1		0	0		0	1
1	0		1	1		1	1
2	1		2	1		2	1
3	1		3	0		3	
4	0	∪	4	0	=	4	0
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.

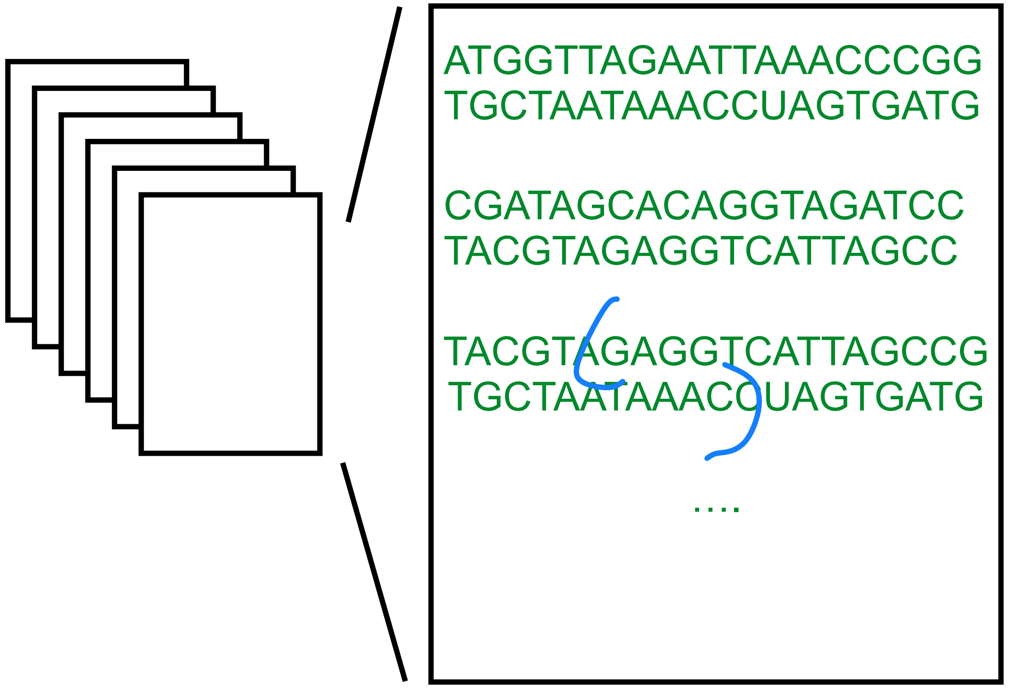
"and"

$\cap$

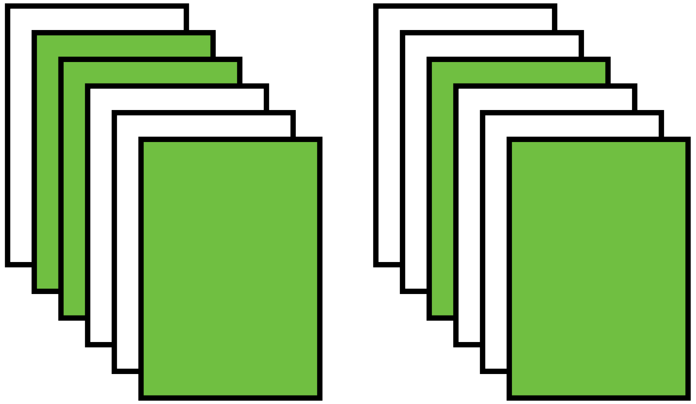
0	1	0	0	0	0
1	0	1	1	1	0
2	1	2	1	2	1
3	1	3	0	3	
4	0	4	0	4	0
5	0	5	0	5	
6	1	6	1	6	
7	0	7	1	7	
8	0	8	1	8	
9	1	9	1	9	

Sequence Bloom Trees

Imagine we have a large collection of text…



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

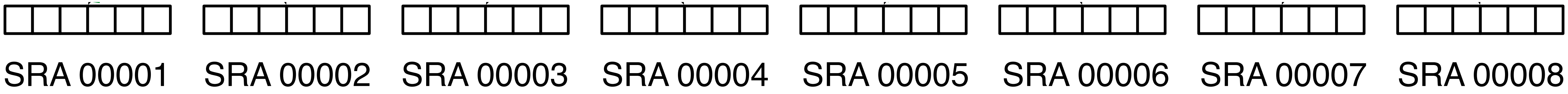


Bit Vector Merging

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

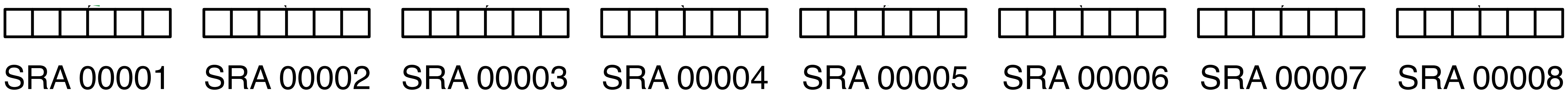
union of 1 & 2

Is all strings in 1 & 2



text content of file 1

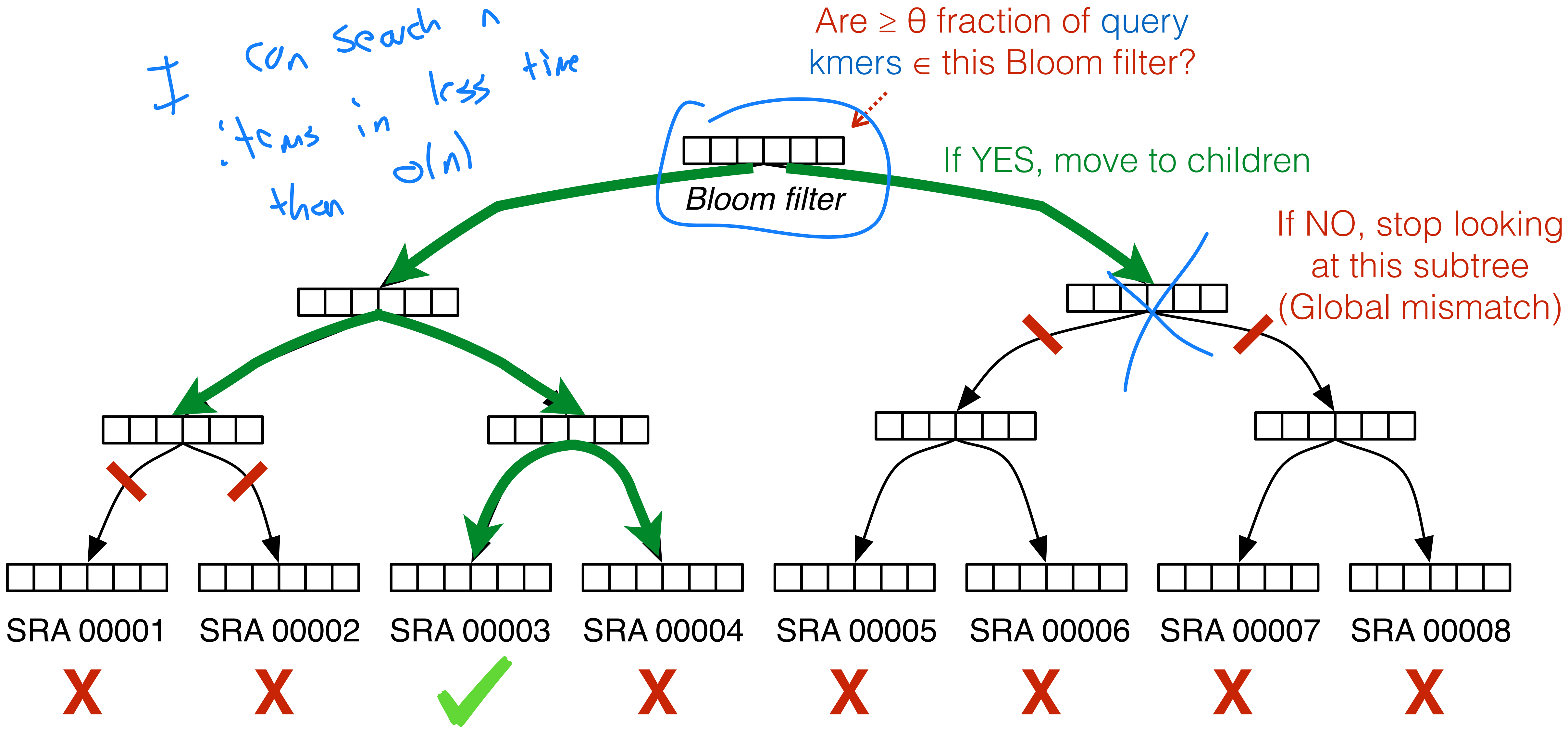
Sequence Bloom Trees



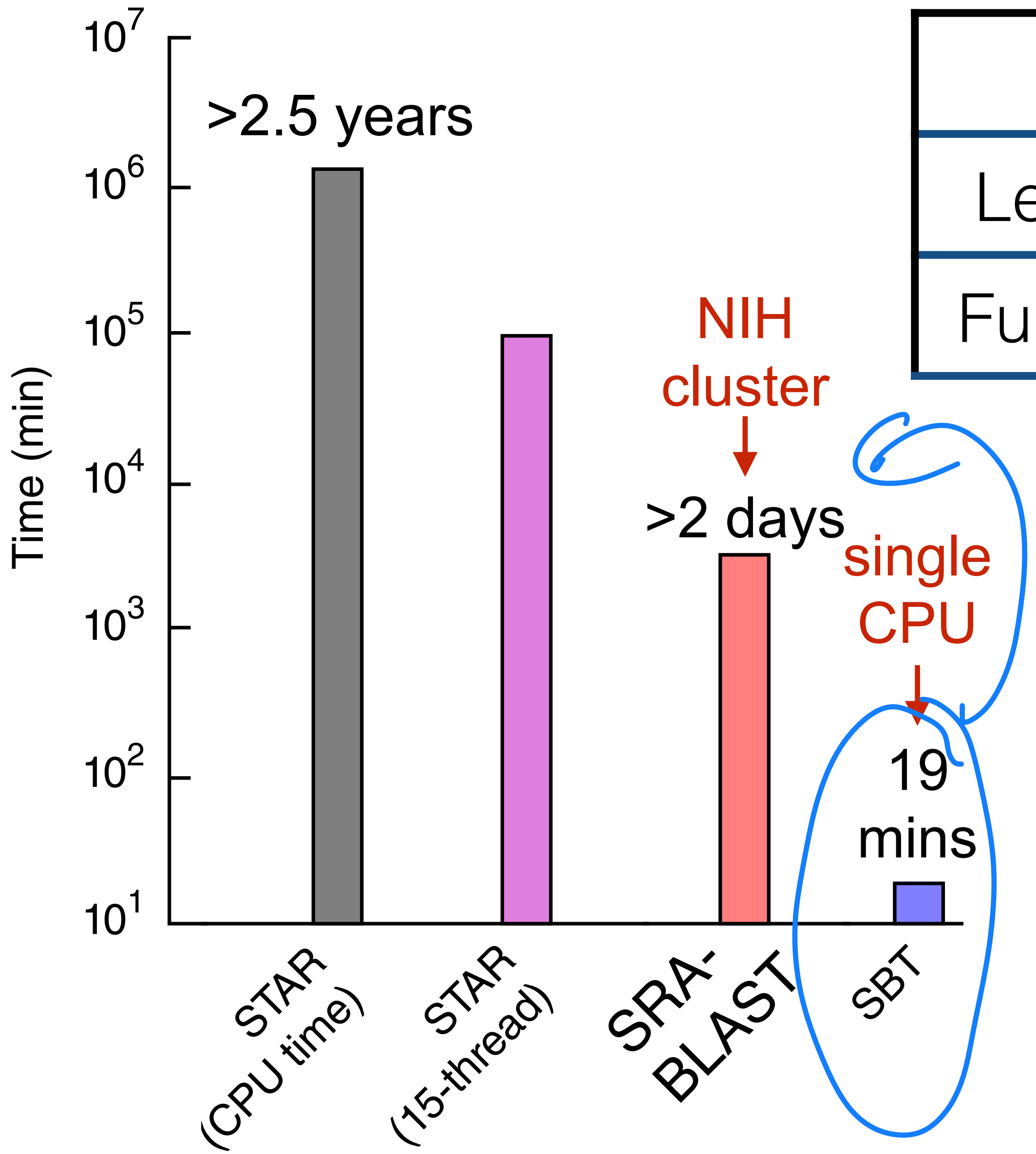
Sequence Bloom Trees

K mers

I can search n items in less time than O(n)



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…