

Algorithms and Data Structures for Data Science

Bloom Filters

CS 277

Brad Solomon

April 22, 2024



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Learning Objectives

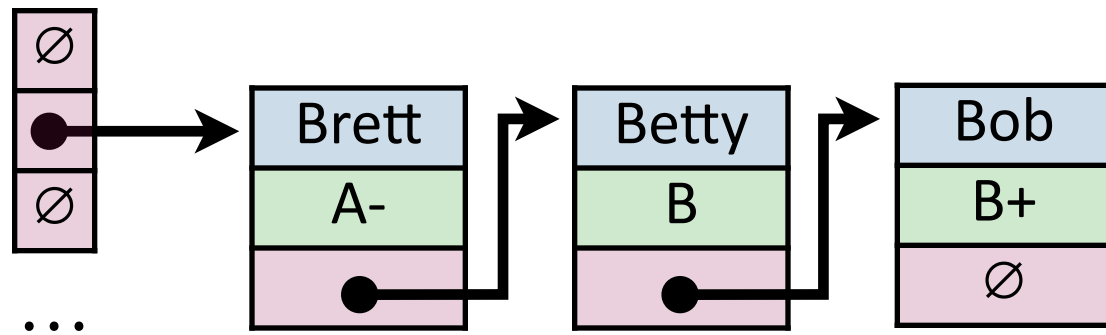
Review fundamentals of hash tables

Introduce probabilistic data structure with bloom filters

Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing:** store k, v pairs externally



- **Closed Hashing:** store k, v pairs in the hash table

K, V
K, V
K, V

Collision Handling: Double Hashing

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

$$h_1(k) = k \% 7$$

$$h_2(k) = 5 - (k \% 5)$$

$$|S| = n$$

$$|\text{Array}| = m$$

0	22
1	8
2	16
3	29
4	4
5	11
6	13

$$h(k, i) = (h_1(k) + i * h_2(k)) \% 7$$

Try $h(k) = (k + 0 * h_2(k)) \% 7$, if full...

Try $h(k) = (k + 1 * h_2(k)) \% 7$, if full...

Try $h(k) = (k + 2 * h_2(k)) \% 7$, if full...

Try ...

Running Times *(Don't memorize these equations, no need.)*

The expected number of probes for find(key) under SUHA

Linear Probing:

- Successful: $\frac{1}{2}(1 + 1/(1-\alpha))$
- Unsuccessful: $\frac{1}{2}(1 + 1/(1-\alpha))^2$

Double Hashing:

- Successful: $1/\alpha * \ln(1/(1-\alpha))$
- Unsuccessful: $1/(1-\alpha)$

Separate Chaining:

- Successful: $1 + \alpha/2$
- Unsuccessful: $1 + \alpha$

Instead, observe:

- **As α increases:**

- **If α is constant:**

Running Times

The expected number of probes for find(key) under SUHA

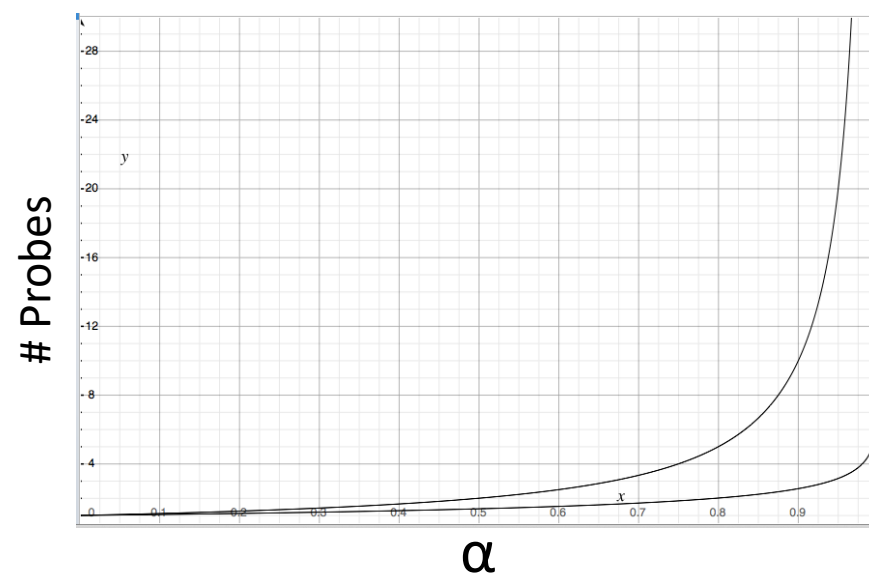
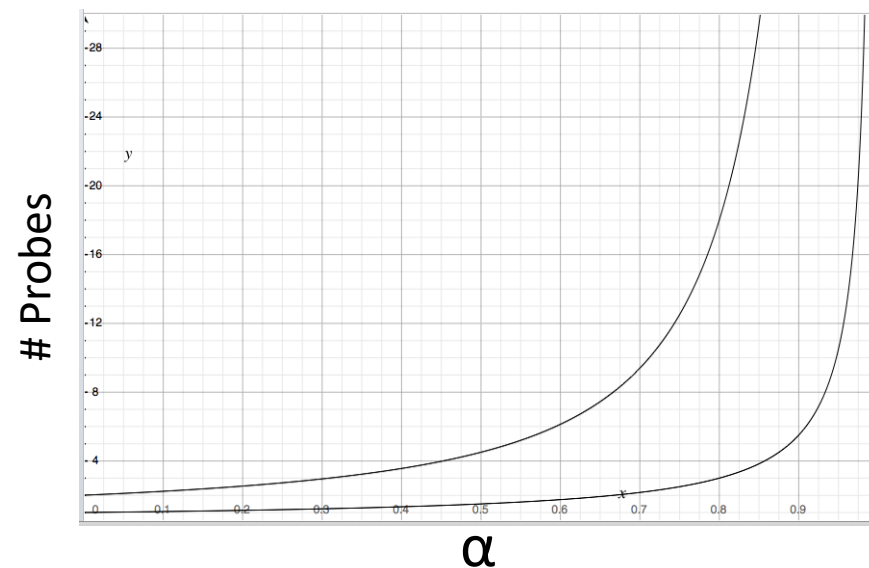
Linear Probing:

- Successful: $\frac{1}{2}(1 + 1/(1-\alpha))$
- Unsuccessful: $\frac{1}{2}(1 + 1/(1-\alpha))^2$

Double Hashing:

- Successful: $1/\alpha * \ln(1/(1-\alpha))$
- Unsuccessful: $1/(1-\alpha)$

When do we resize?



Running Times

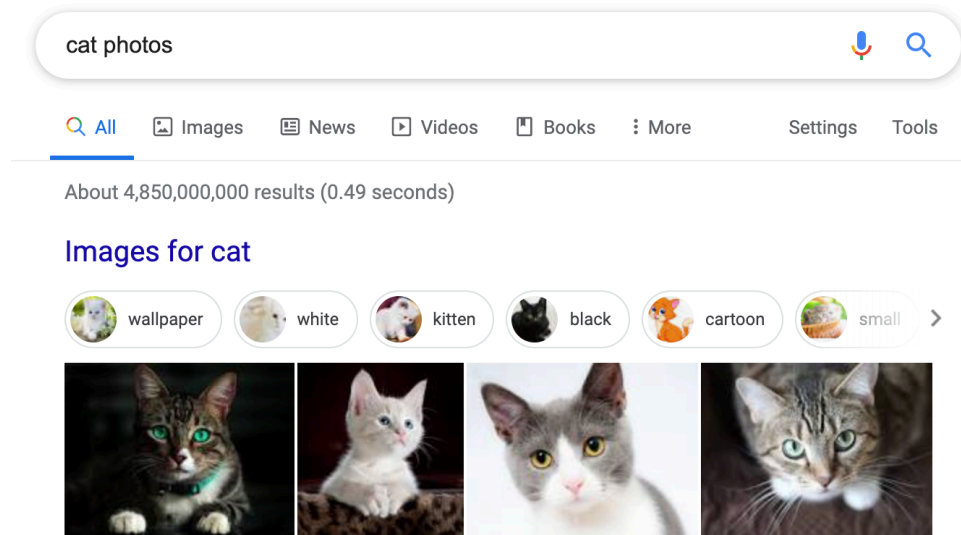


	Hash Table	AVL	Linked List
Find	Expectation*: Worst Case:		
Insert	Expectation*: Worst Case:		
Storage Space			

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)



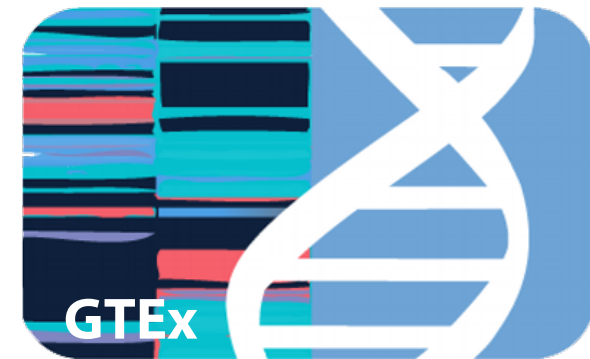
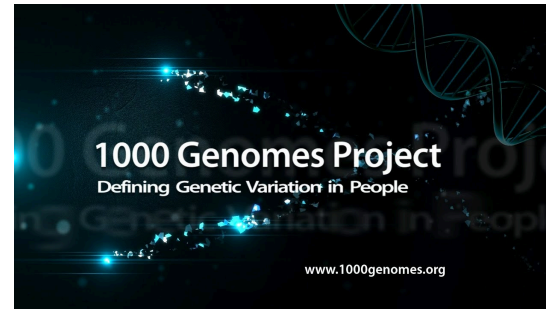
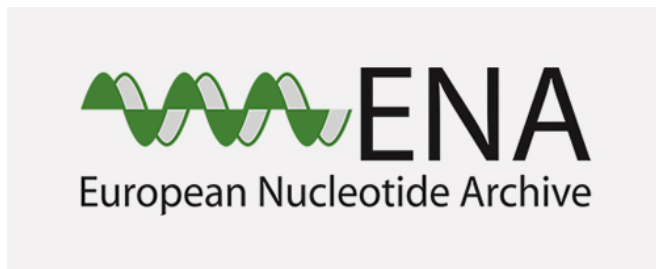
Google Index Estimate: >60 billion webpages

Google Universe Estimate (2013): >130 trillion webpages

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)



SRA

Sequence Read Archive (SRA) makes biological sequence data available to the research community to enhance reproducibility and allow for new discoveries by comparing data sets. The SRA stores raw sequencing data and alignment information from high-throughput sequencing platforms, including Roche 454 GS System®, Illumina Genome Analyzer®, Applied Biosystems SOLiD System®, Helicos Heliscope®, Complete Genomics®, and Pacific Biosciences SMRT®.

Sequence Read Archive Size: >60 petabases (10^{15})

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)

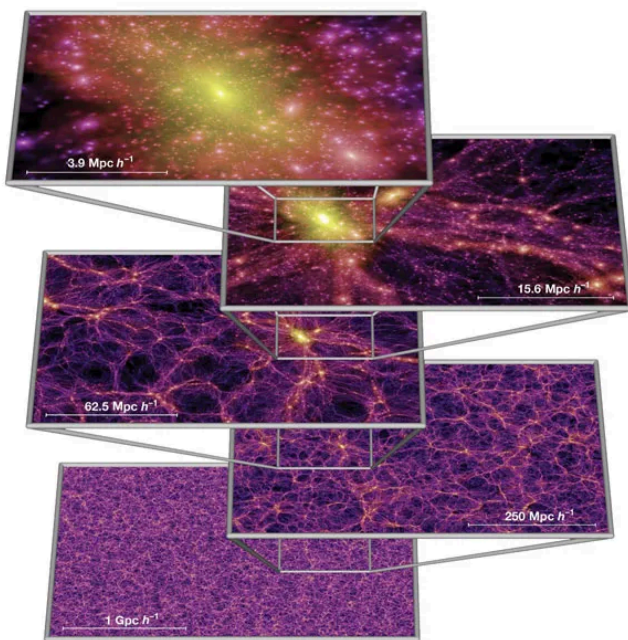


Image: <https://doi.org/10.1038/nature03597>

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

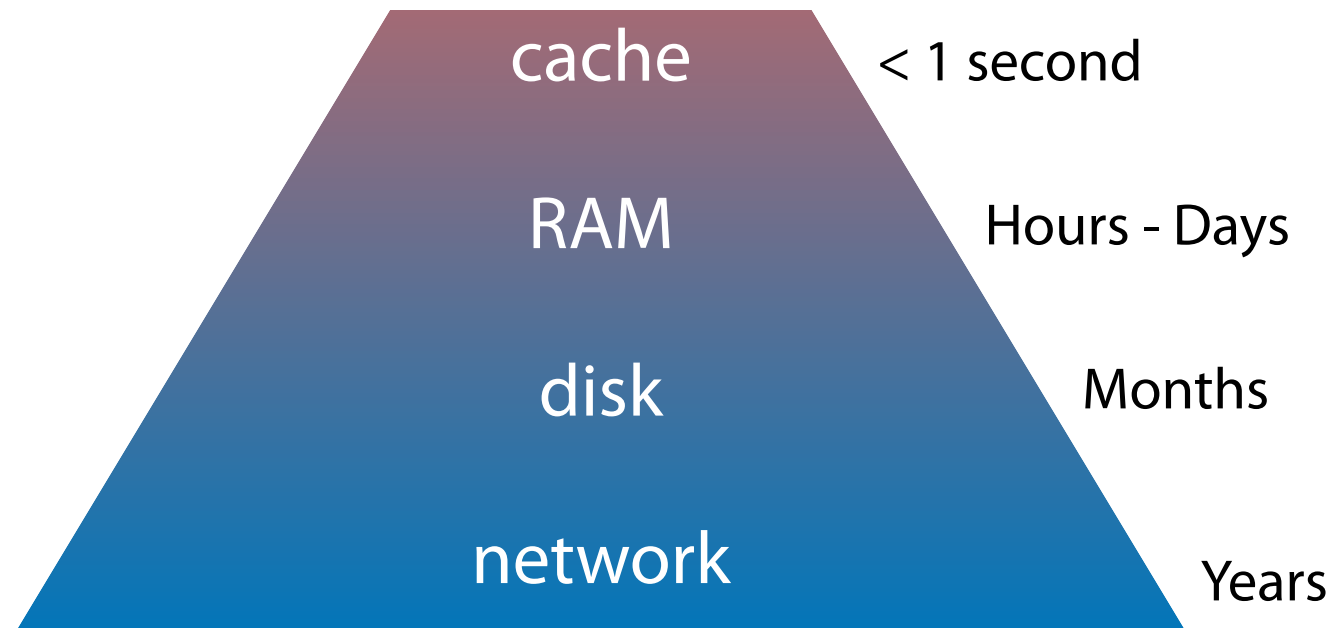
Table: <http://doi.org/10.5334/dsj-2015-011>

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>)

Memory-Constrained Data Structures



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

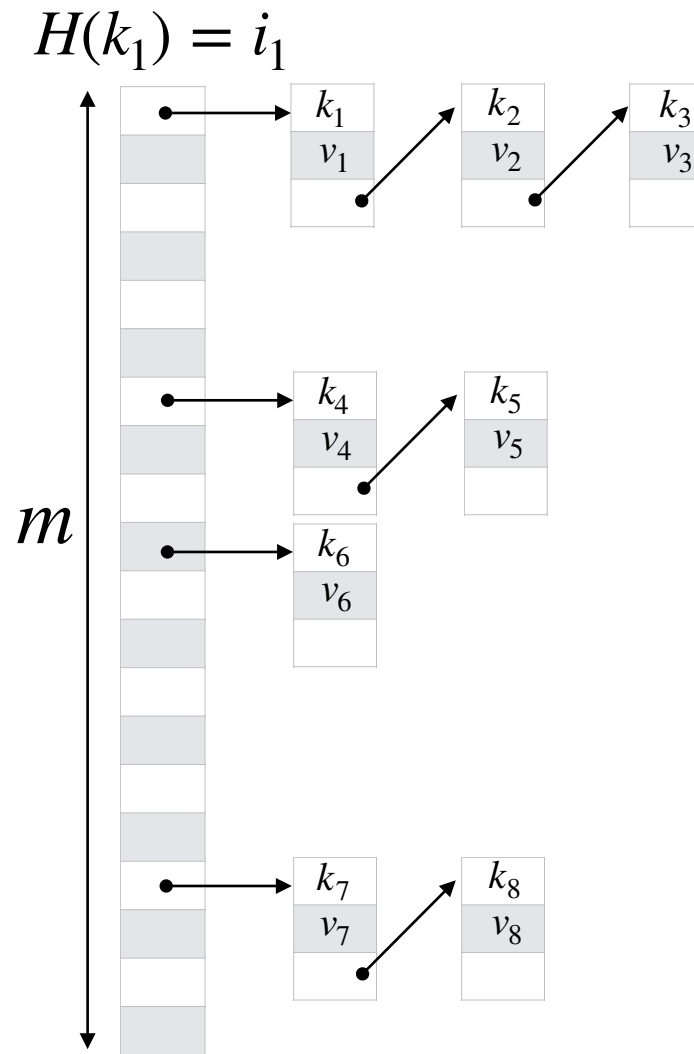
Reducing storage costs

1) Throw out information that isn't needed

2) Compress the dataset

Reducing a hash table

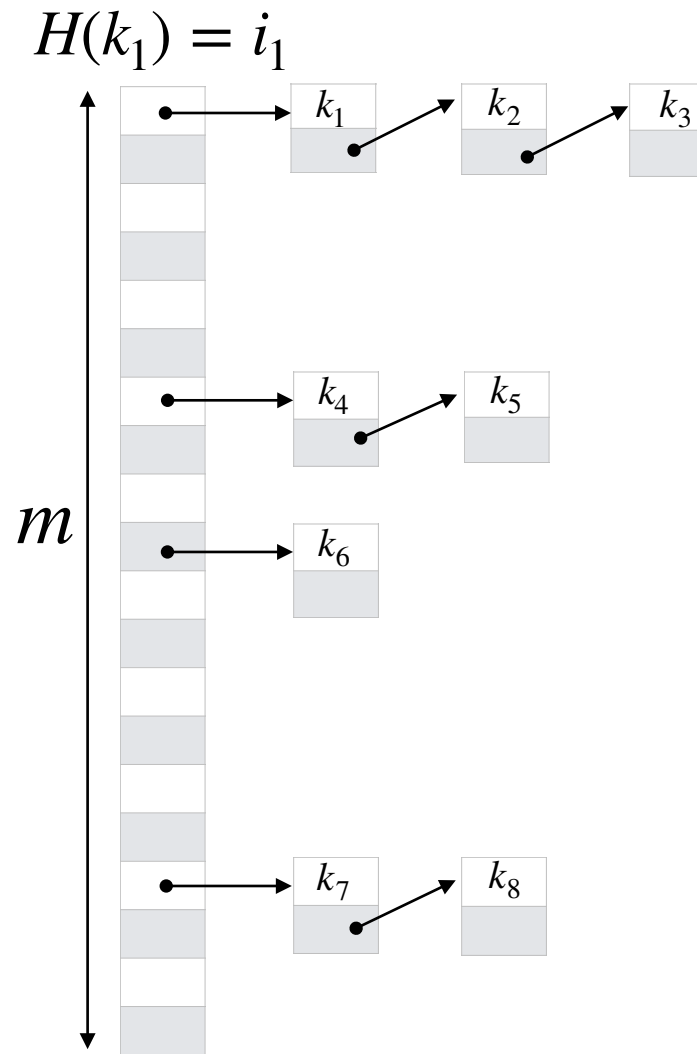
What can we remove from a hash table?



Reducing a hash table

What can we remove from a hash table?

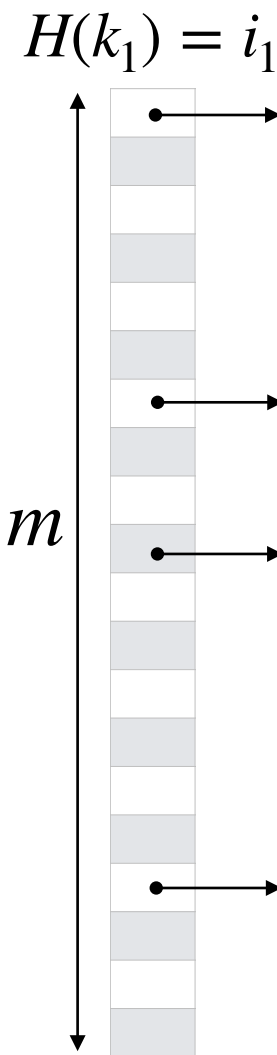
Take away values



Reducing a hash table

What can we remove from a hash table?

Take away values and keys



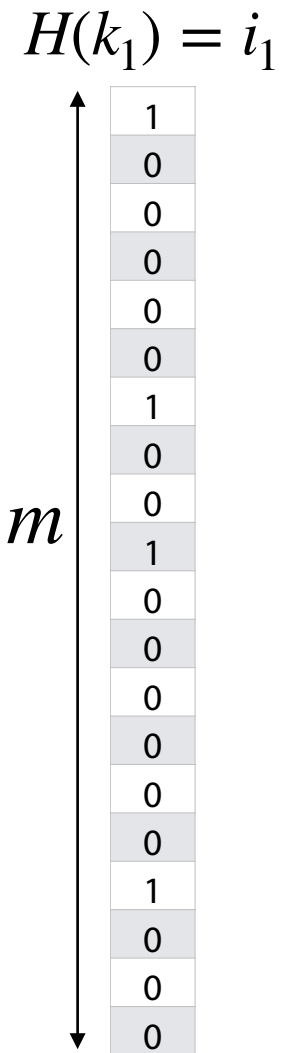


Reducing a hash table

What can we remove from a hash table?

Take away values and keys

This is a ***bloom filter***



Bloom Filter ADT

Constructor

Insert

Find

Bloom Filter: Insertion

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

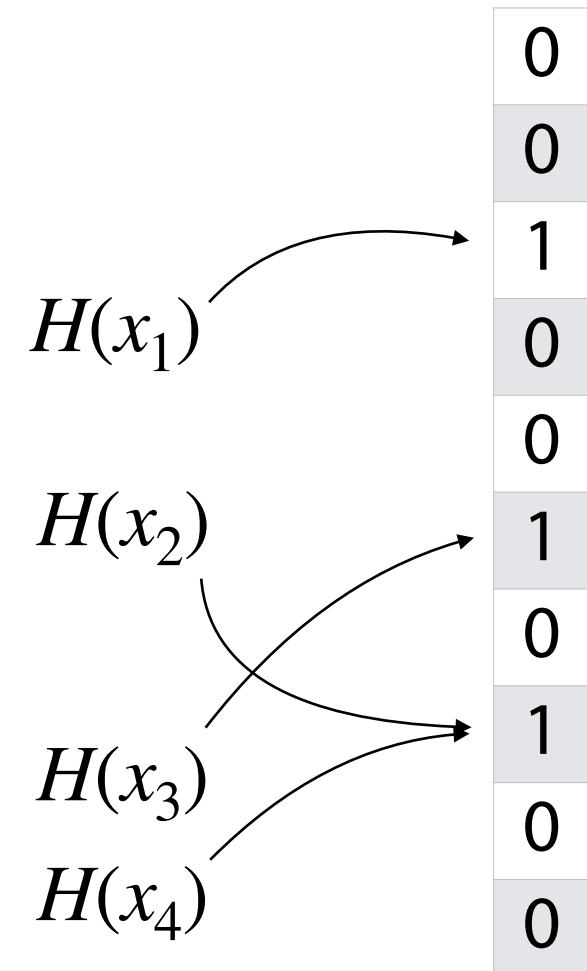
$h(k) = k \% 7$

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

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

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

$h(k) = k \% 7$

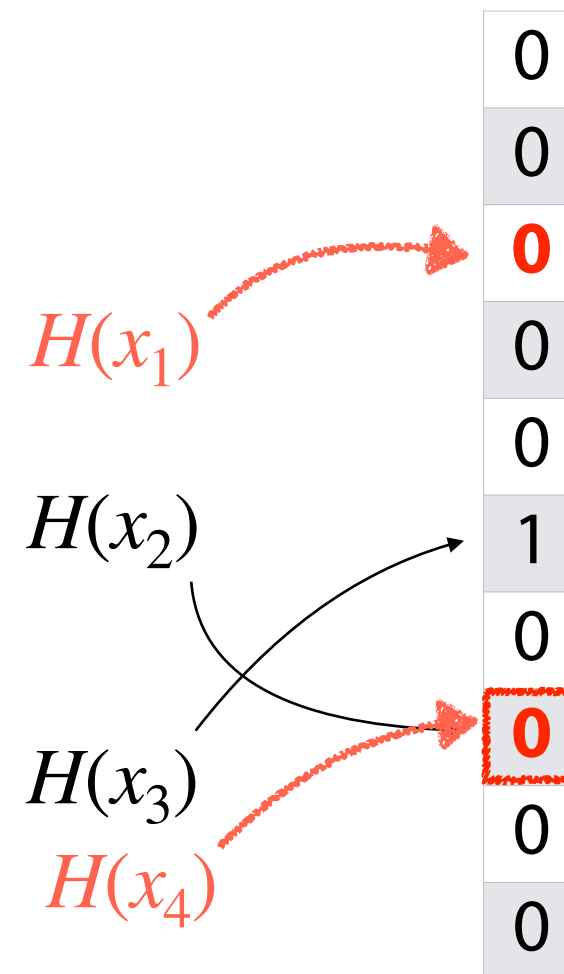
0	0
1	1
2	1
3	0
4	1
5	0
6	1

`_delete(13)`

`_delete(29)`

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)`

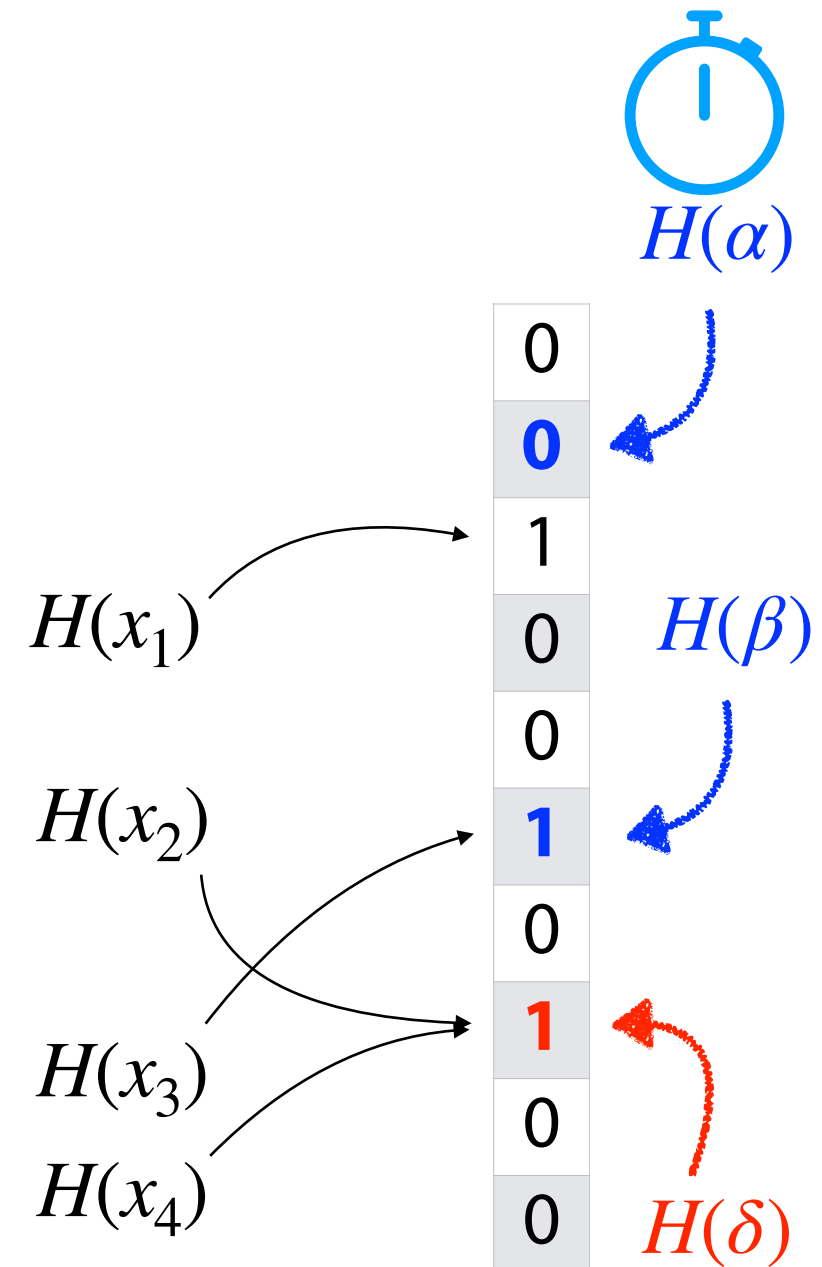
`_find(3)`

Bloom Filter: Search

The bloom filter is a *probabilistic* data structure!

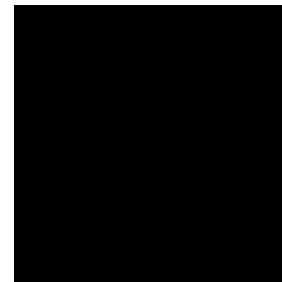
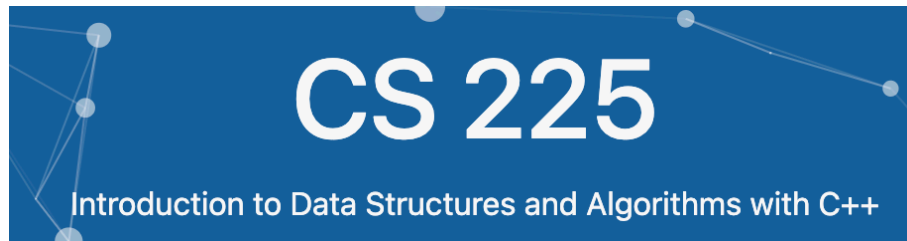
If the value in the BF is 0:

If the value in the BF is 1:

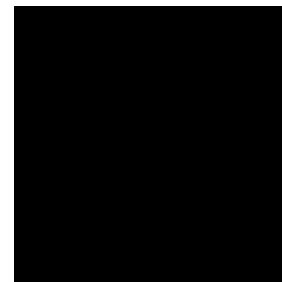
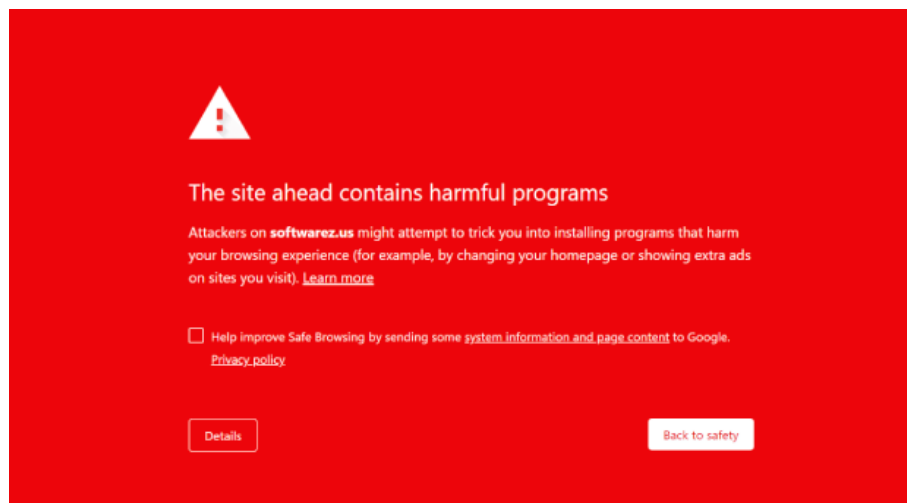


Probabilistic Accuracy: Malicious Websites

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



"Not malicious"



"Malicious"

Probabilistic Accuracy: Malicious Websites

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

True Positive:

False Positive:

False Negative:

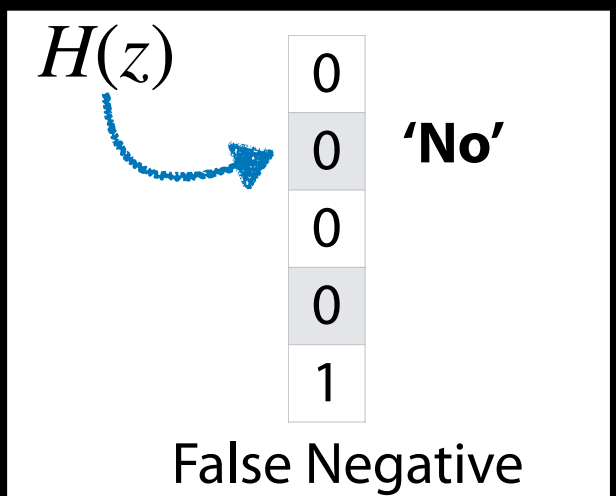
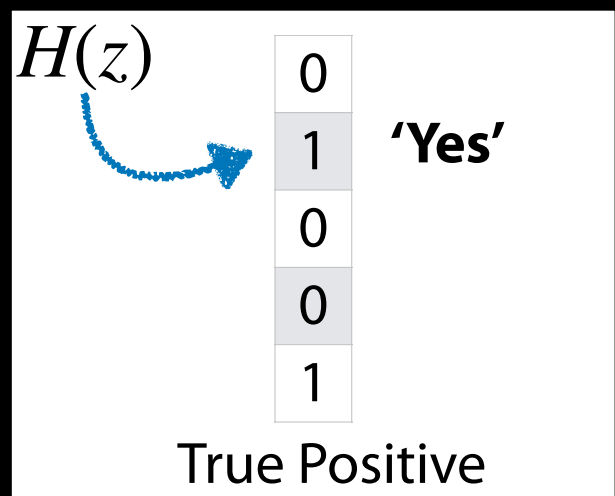
True Negative:

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

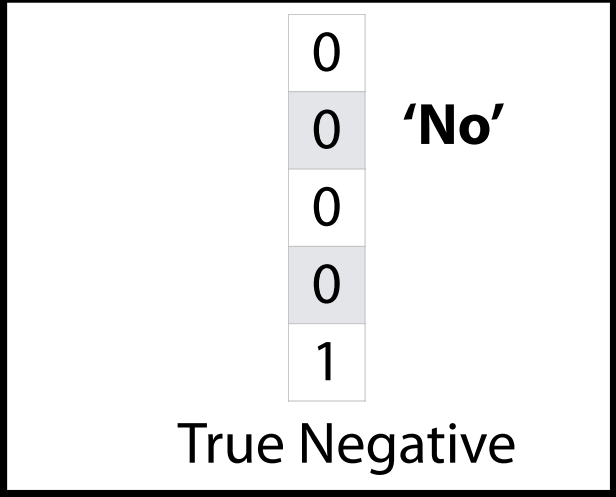
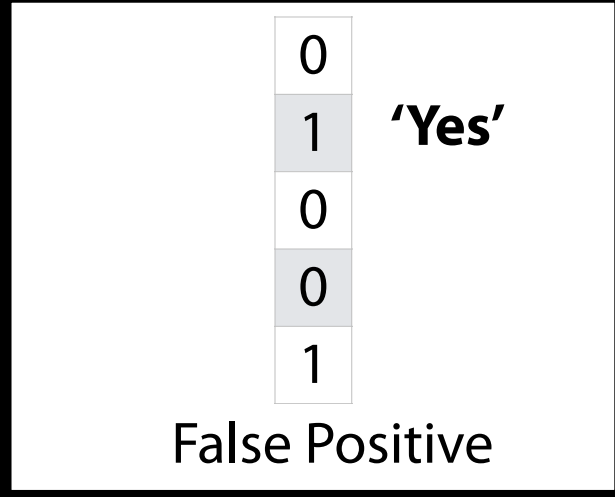
Bit Value = 1

Bit Value = 0

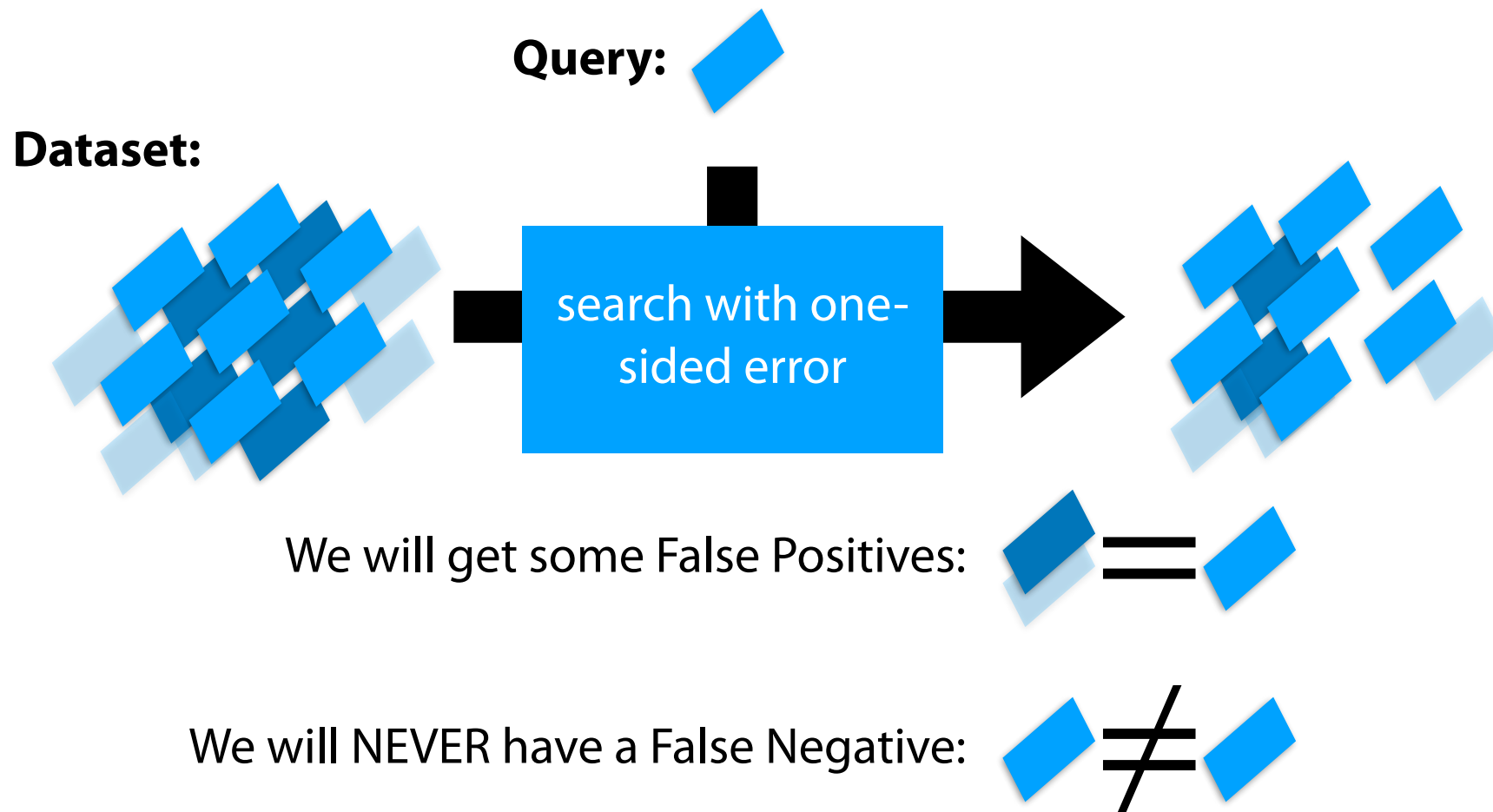
Item Inserted



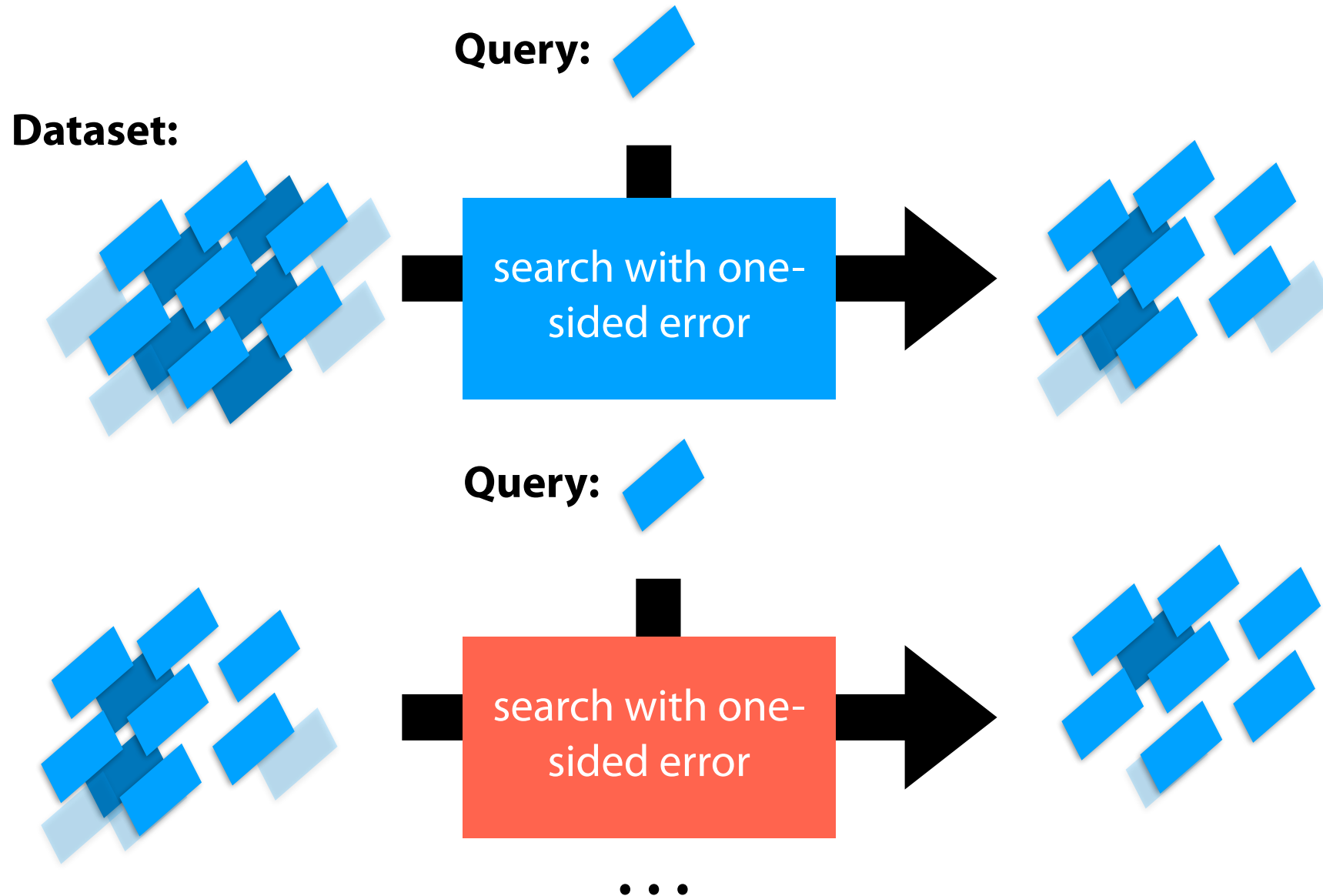
Item NOT inserted



Probabilistic Accuracy: One-sided error

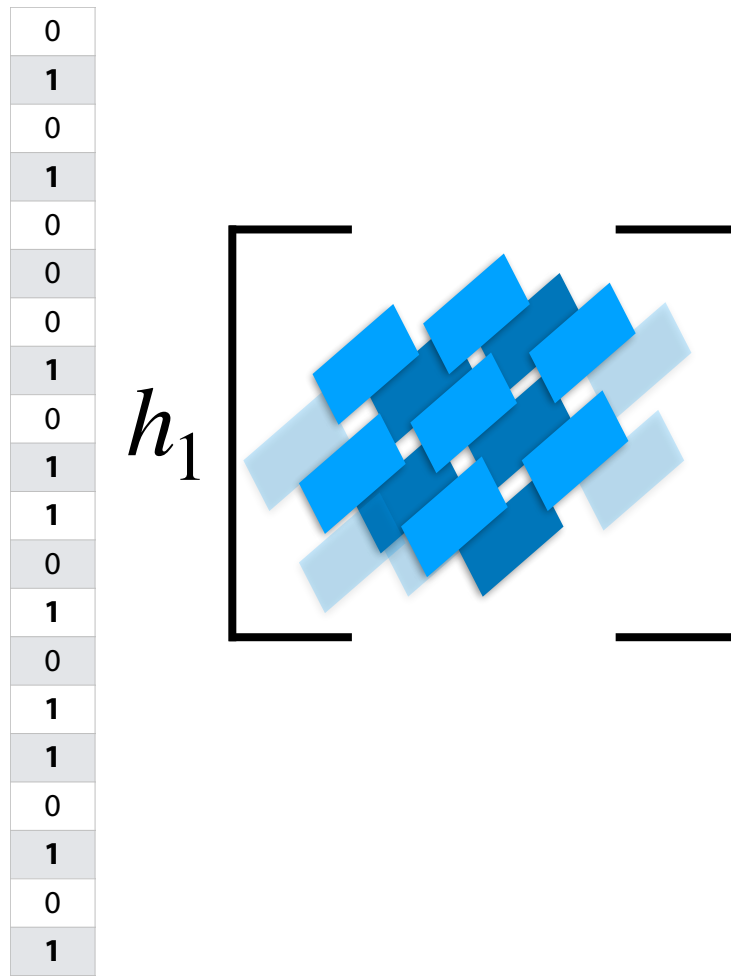


Probabilistic Accuracy: One-sided error



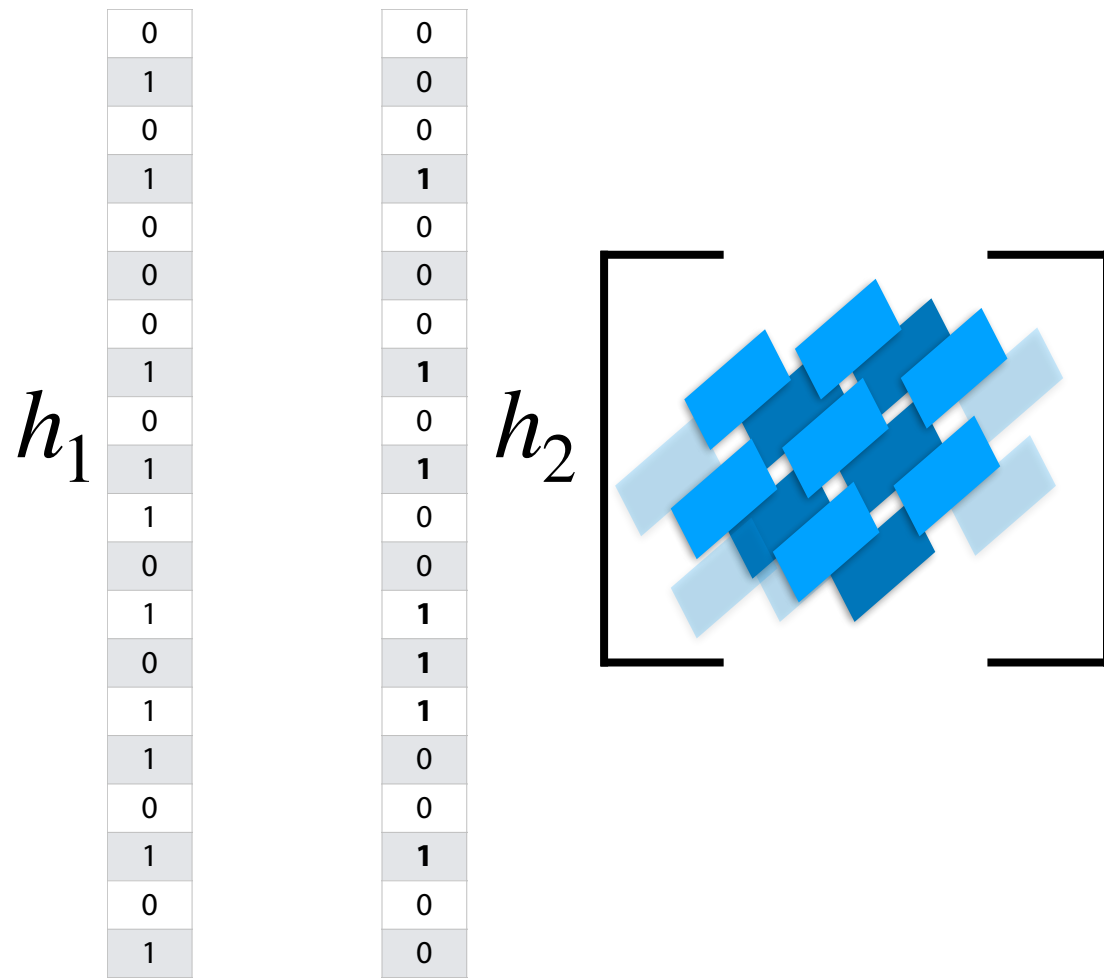
Bloom Filter: Repeated Trials

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



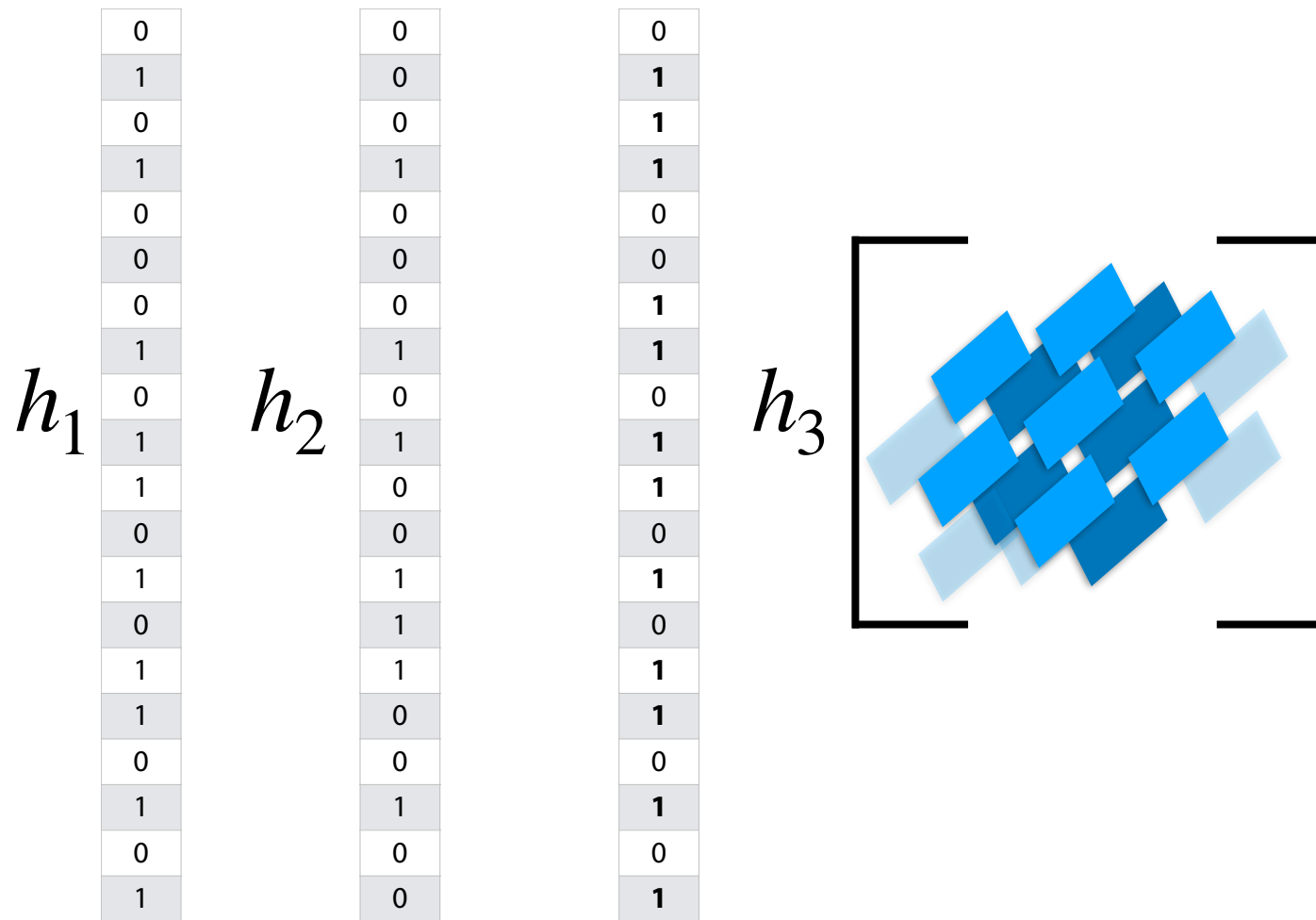
Bloom Filter: Repeated Trials

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



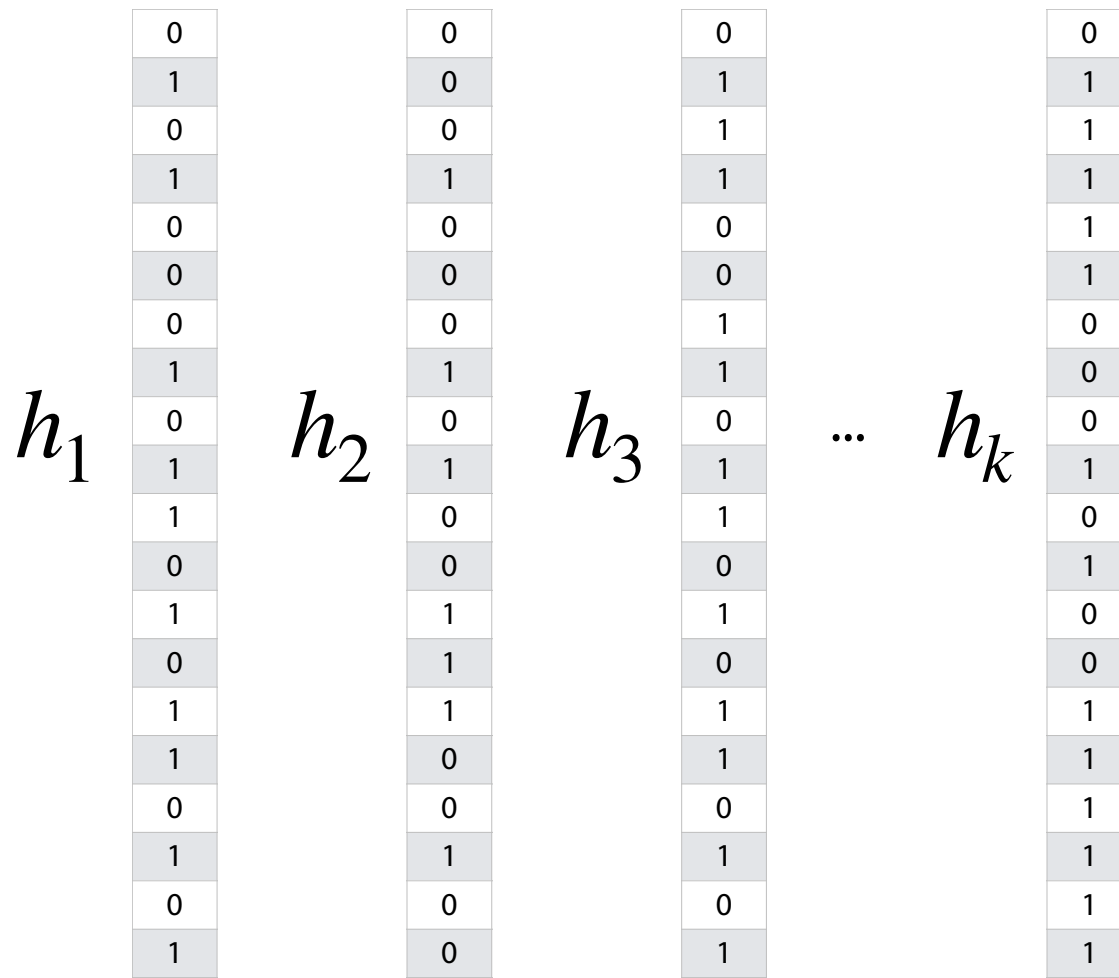
Bloom Filter: Repeated Trials

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

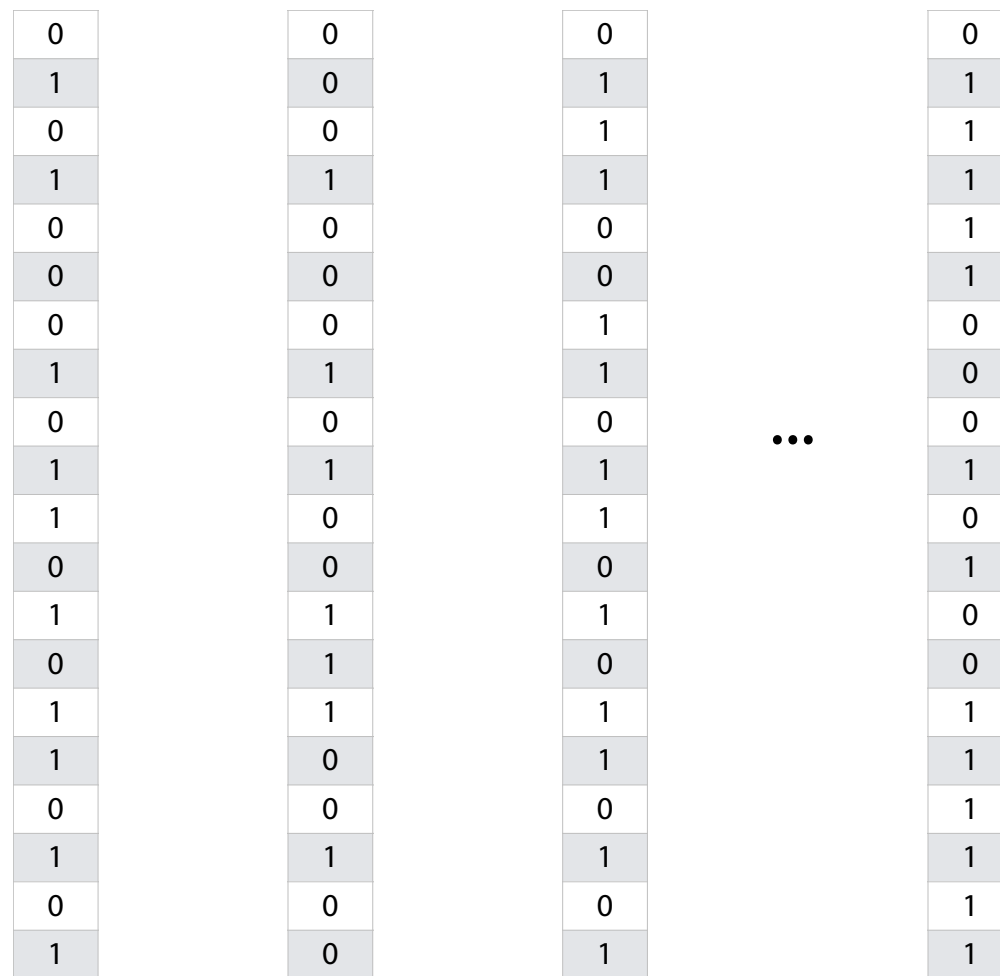


Bloom Filter: Repeated Trials

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

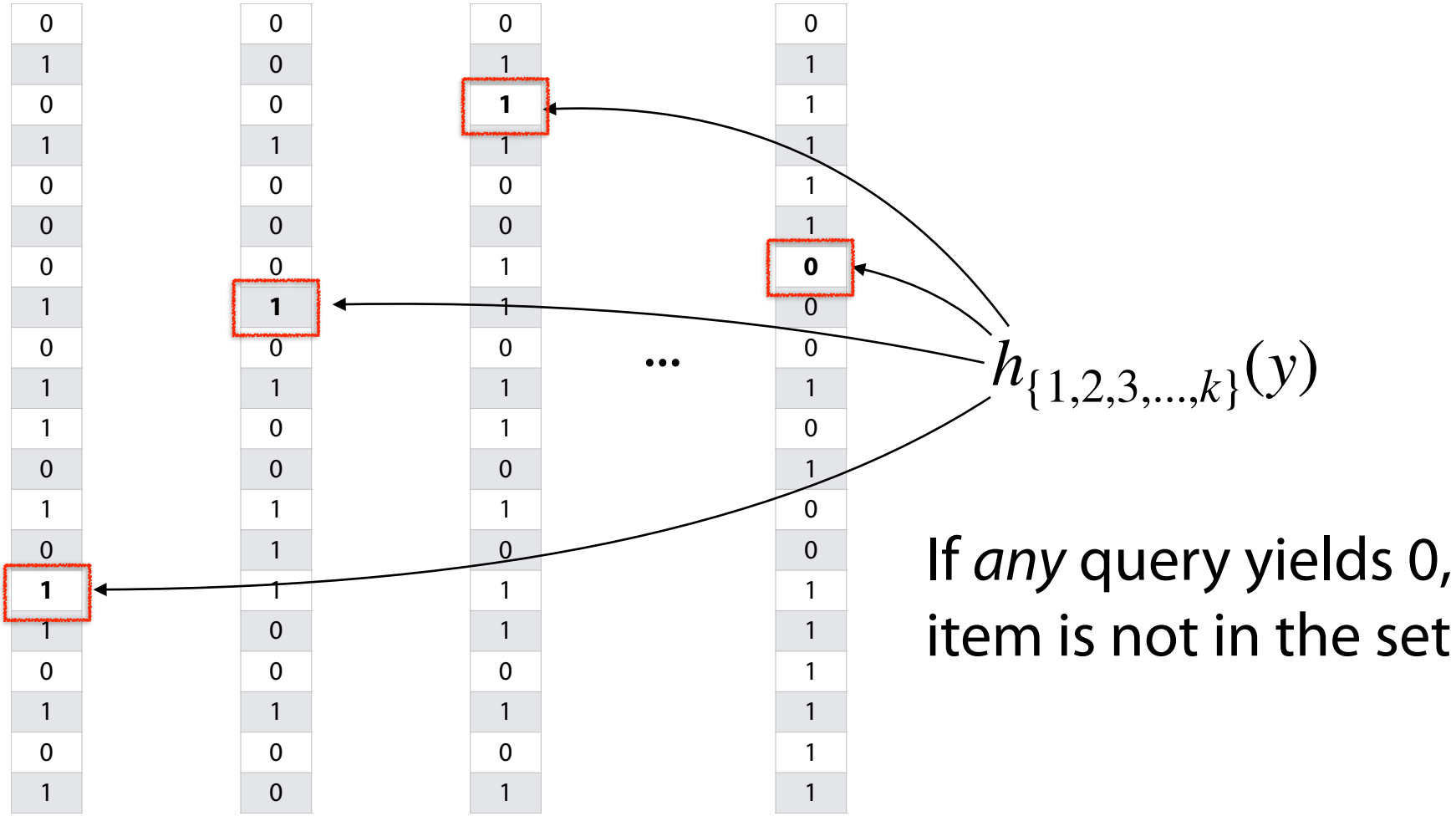


Bloom Filter: Repeated Trials

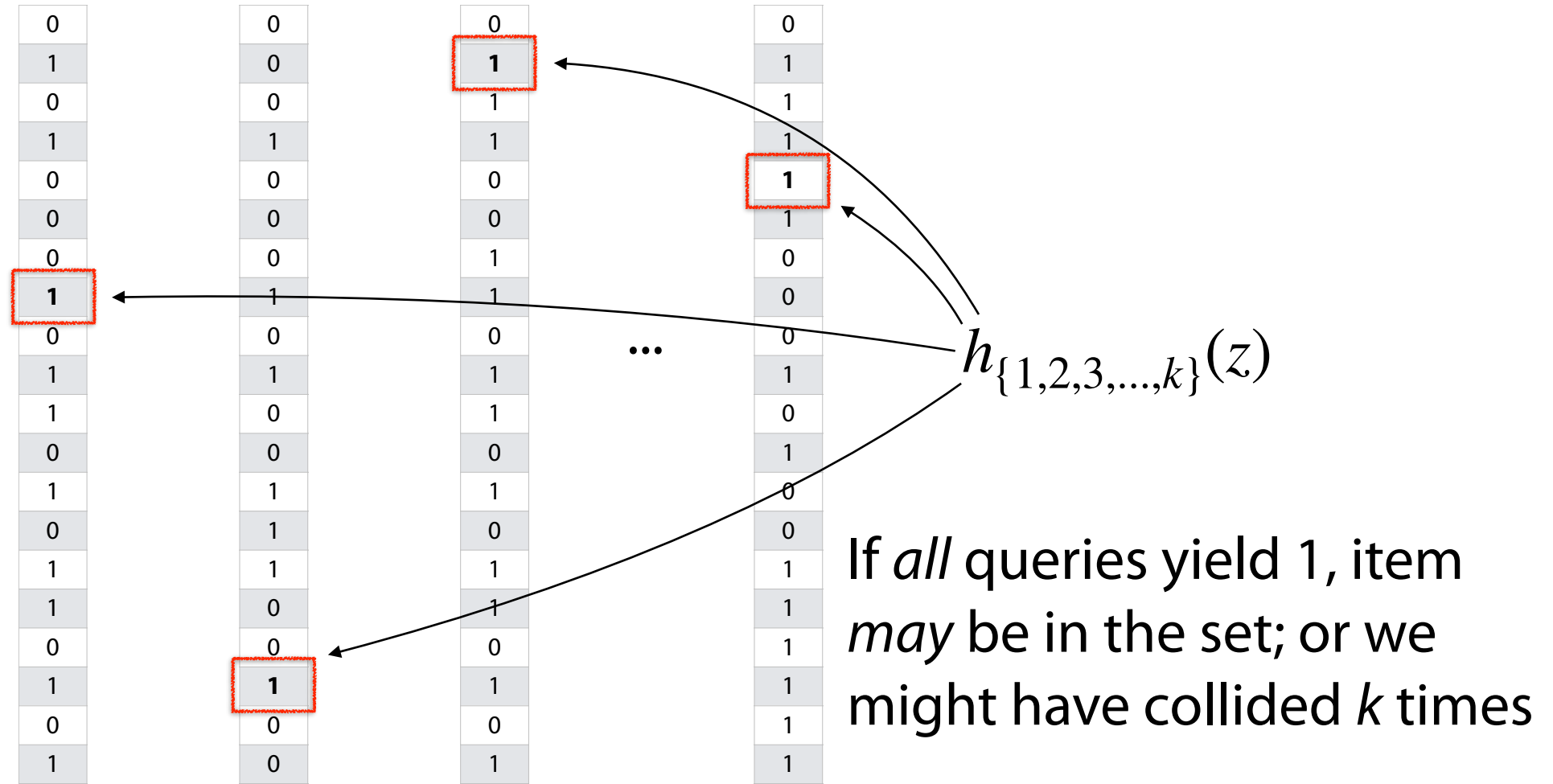


$$h_{\{1,2,3,\dots,k\}}(y)$$

Bloom Filter: Repeated Trials



Bloom Filter: Repeated Trials



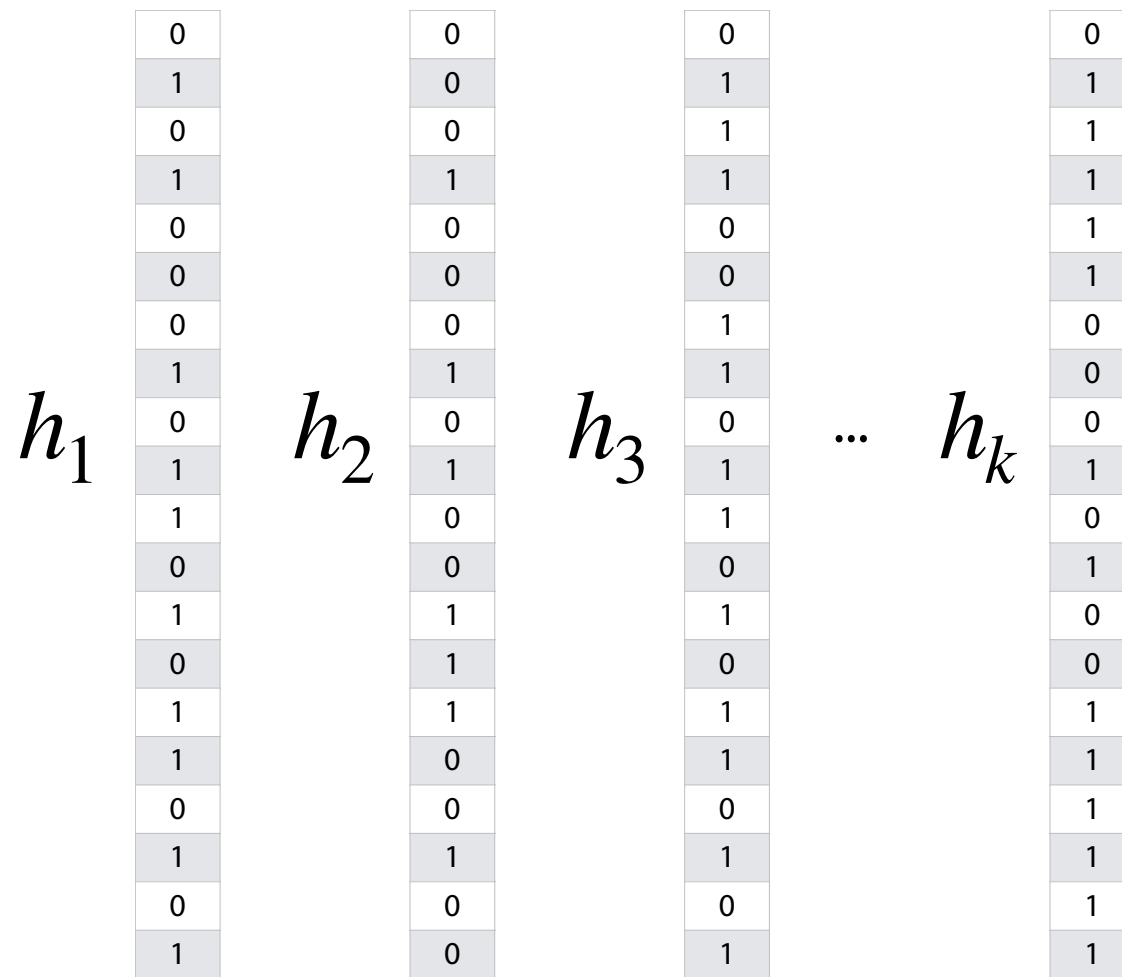
Bloom Filter: Repeated Trials

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?



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	$h_1(x) = x \% 10$	$h_2(x) = 2x \% 10$	$h_3(x) = (5+3x) \% 10$
1	0			
2	1	<code><u>find</u>(1)</code>		
3	1			
4	1			
5	0			
6	1	<code><u>find</u>(16)</code>		
7	1			
8	1			
9	1			

Bloom Filter



A probabilistic data structure storing a set of values

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

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

Insert / Find runs in: _____

Delete is not possible (yet)!

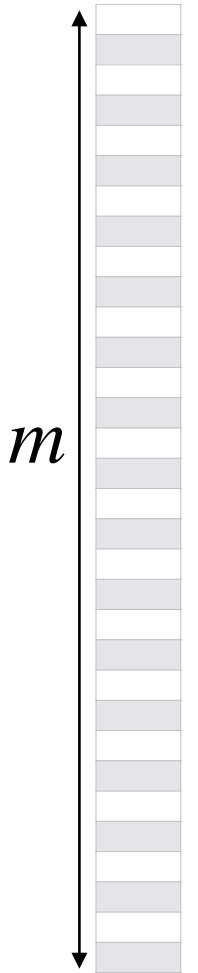
0
0
1
0
0
1
0
1
0
0

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?

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



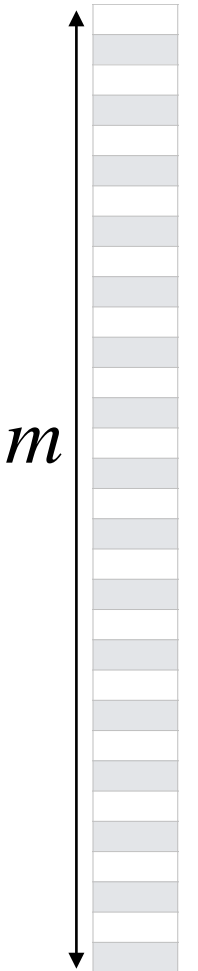
Bloom Filter: Error Rate

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?

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



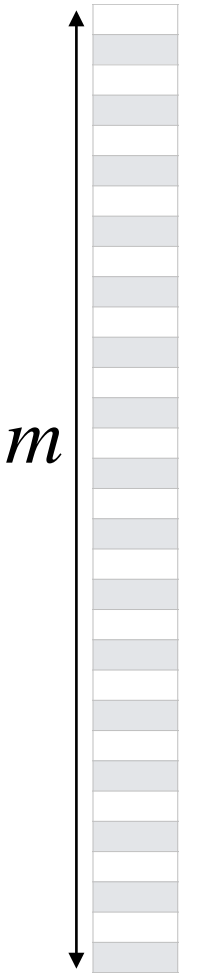
Bloom Filter: Error Rate

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? Using k hashes?

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

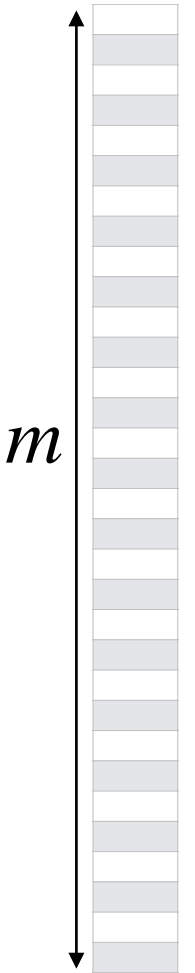


Bloom Filter: Error Rate

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? How about k buckets?

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



Bloom Filter: Error Rate

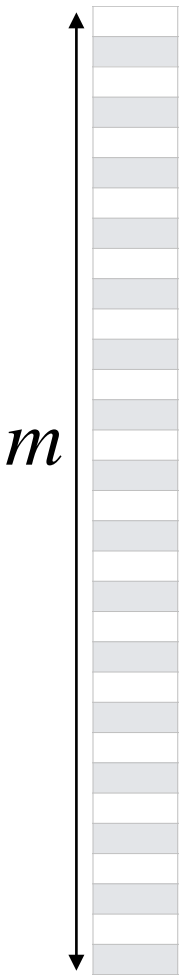
Given bit vector of size m and k SUHA hash function

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



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

Bloom Filter: Error Rate

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

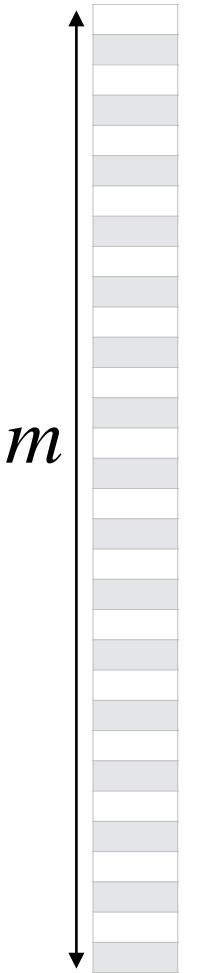
To minimize the FPR, do we prefer...

(A) large k

(B) small k

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

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



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

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

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

Bloom Filter: Optimal Error Rate

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

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

Bloom Filter: Optimal Error Rate

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

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

$$\ln(x^y) = y \ln(x)$$

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

Bloom Filter: Optimal Error Rate

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

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

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots$$

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

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

Bloom Filter: Optimal Error Rate

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

$$\min [f(x)] = \min [\ln f(x)]$$

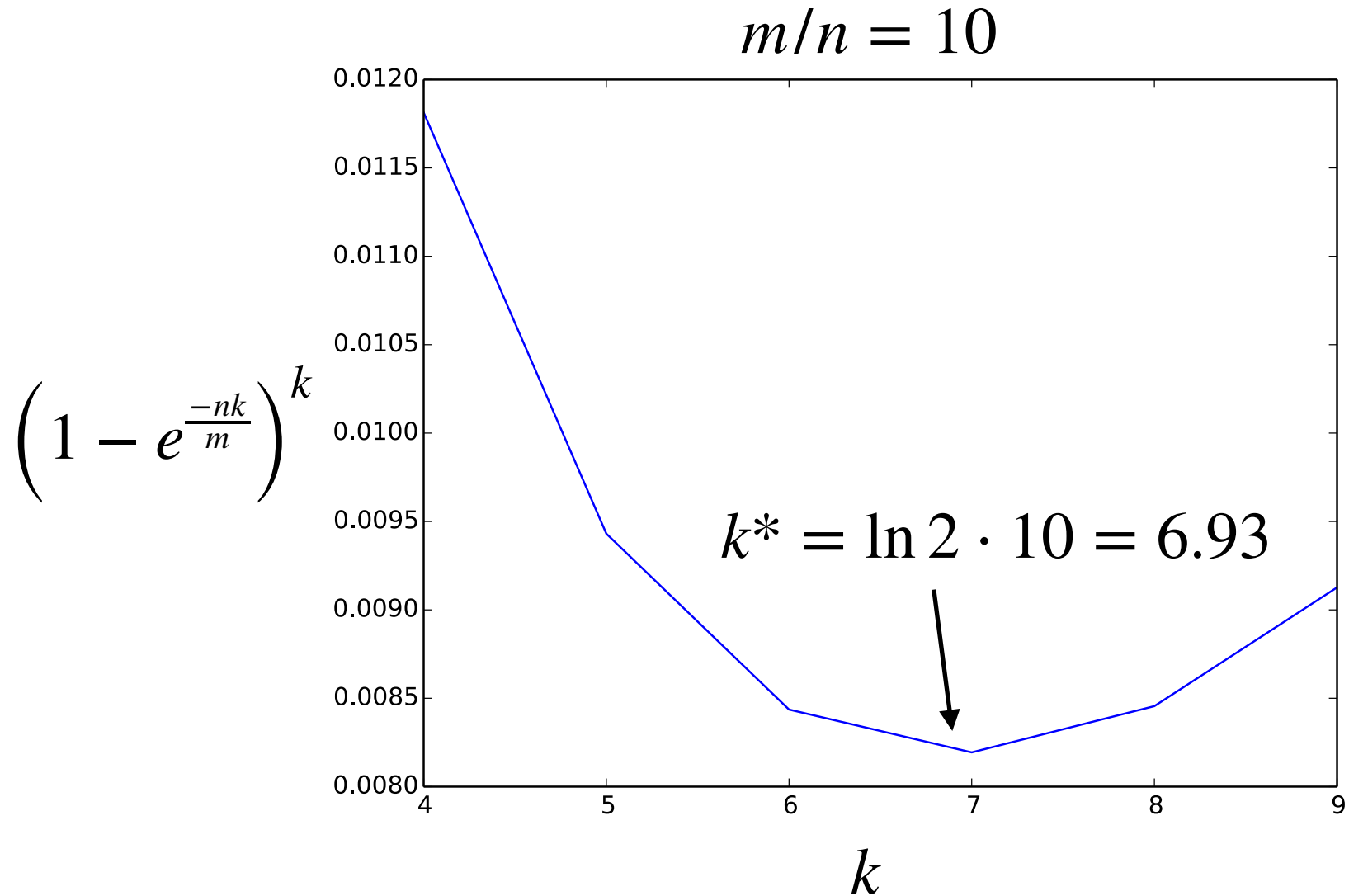
Bloom Filter: Optimal Error Rate

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

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

$$\frac{d}{dx} \ln f(x) = \frac{1}{f(x)} \frac{df(x)}{dx} \quad \dots \text{and math!}$$

Bloom Filter: Error Rate



Bloom Filter: Optimal Parameters

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

Given any two values, we can optimize the third

$$n = 100 \text{ items} \quad k = 3 \text{ hashes} \quad m =$$

$$m = 100 \text{ bits} \quad n = 20 \text{ items} \quad k =$$

$$m = 100 \text{ bits} \quad k = 2 \text{ items} \quad n =$$

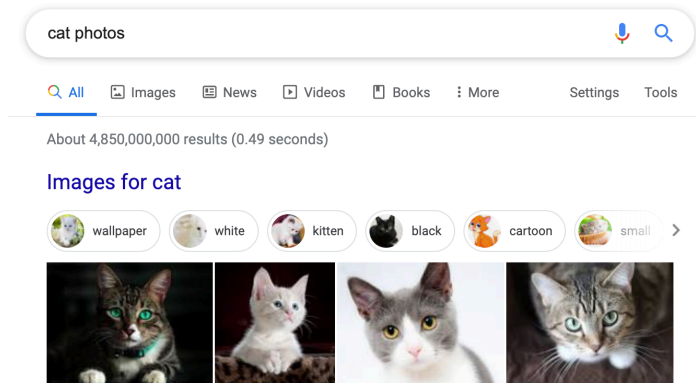
Bloom Filter: Optimal Parameters

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

Optimal hash function is still $O(m)$!

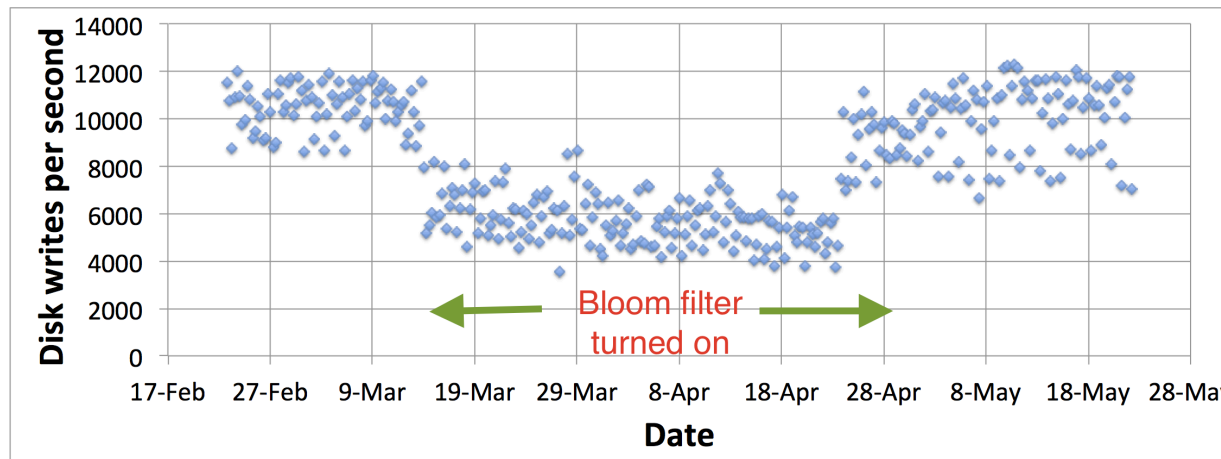
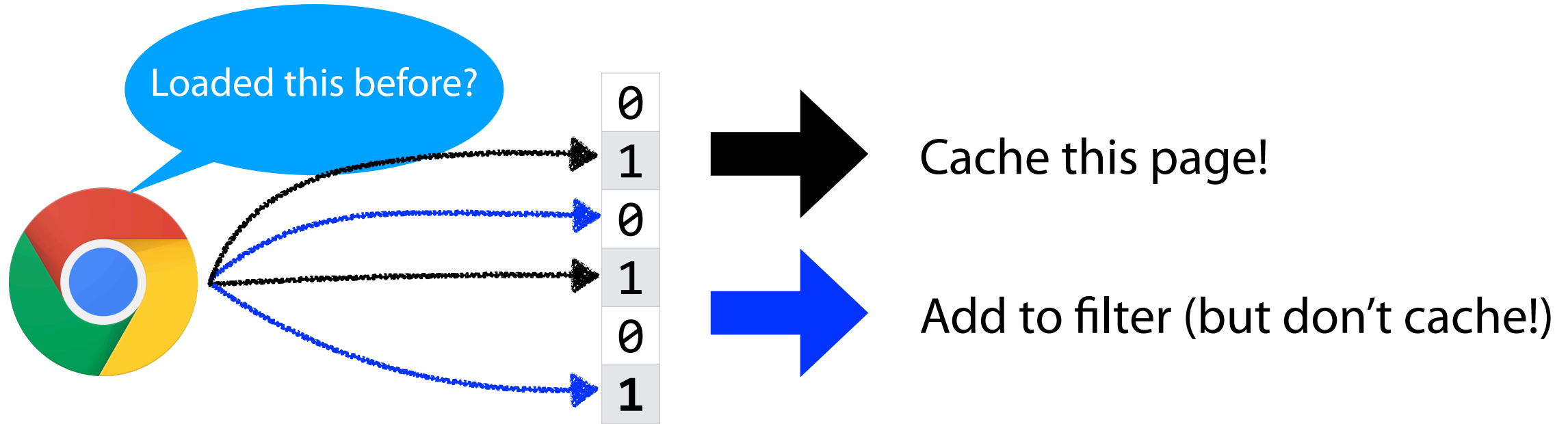


$n = 250,000$ files vs $\sim 10^{15}$ nucleotides vs 260 TB



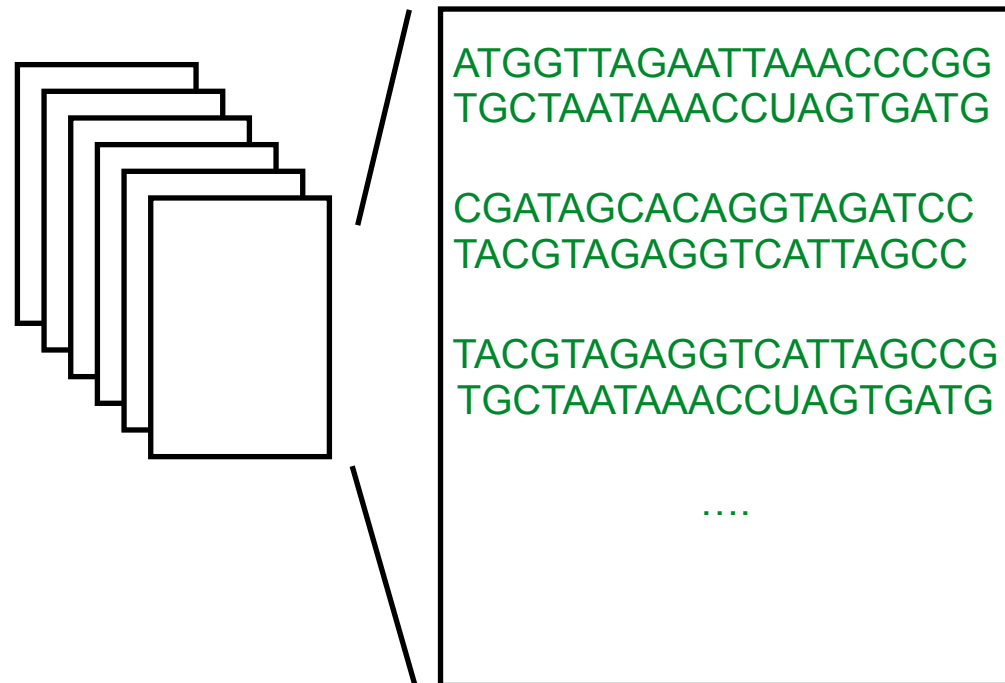
$n = 60$ billion — 130 trillion

Bloom Filter: Website Caching

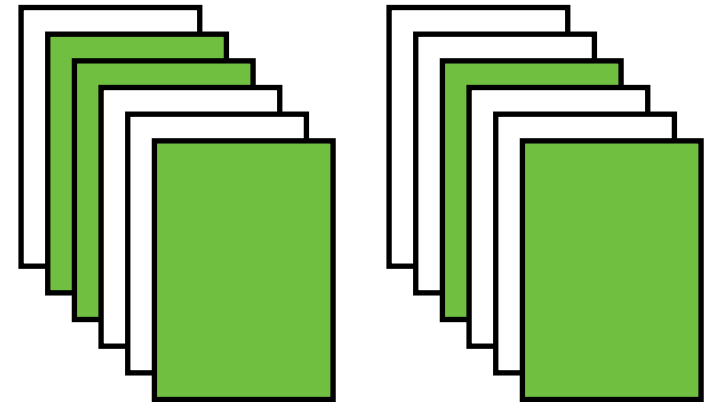


Sequence Bloom Trees

Imagine we have a large collection of text...

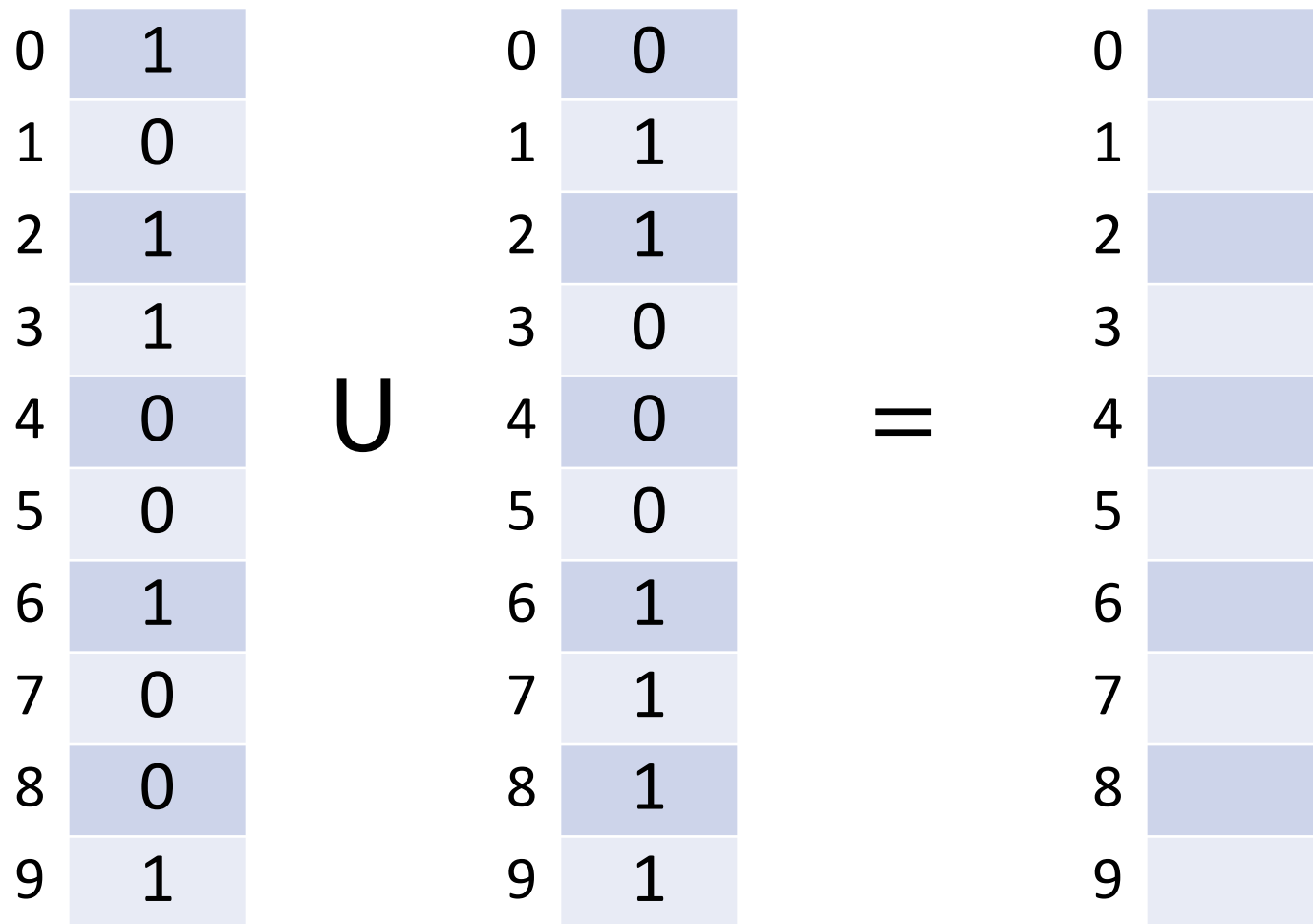


And our goal is to search these files for a query of interest...



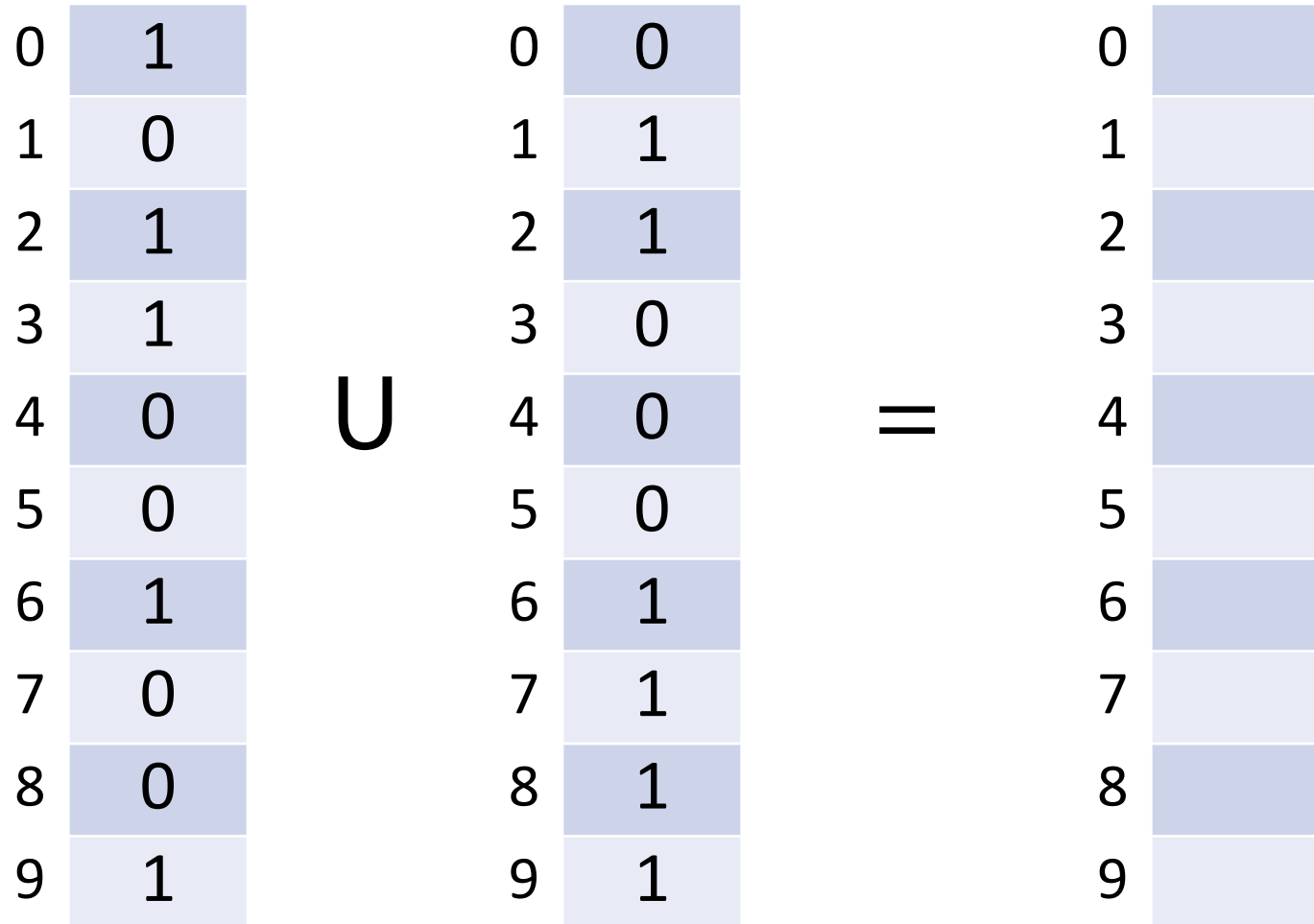
Bloom Filters: Unioning

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



Bloom Filters: Intersection

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



Bit Vector Merging

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



SRA 00001



SRA 00002



SRA 00003



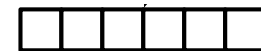
SRA 00004



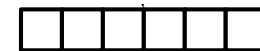
SRA 00005



SRA 00006



SRA 00007

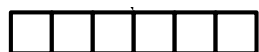


SRA 00008

Sequence Bloom Trees



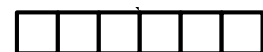
SRA 00001



SRA 00002



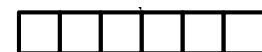
SRA 00003



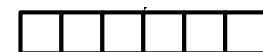
SRA 00004



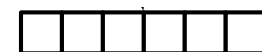
SRA 00005



SRA 00006

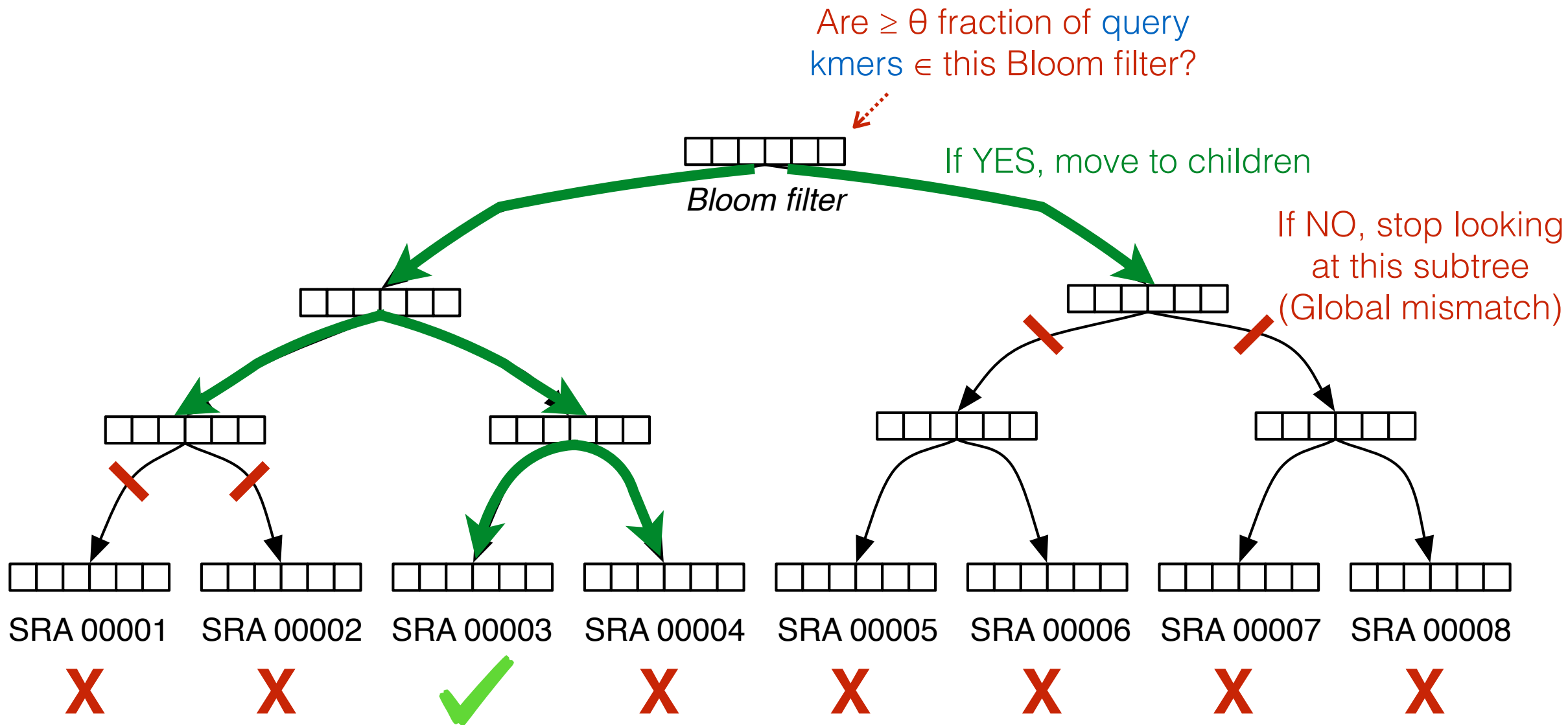


SRA 00007

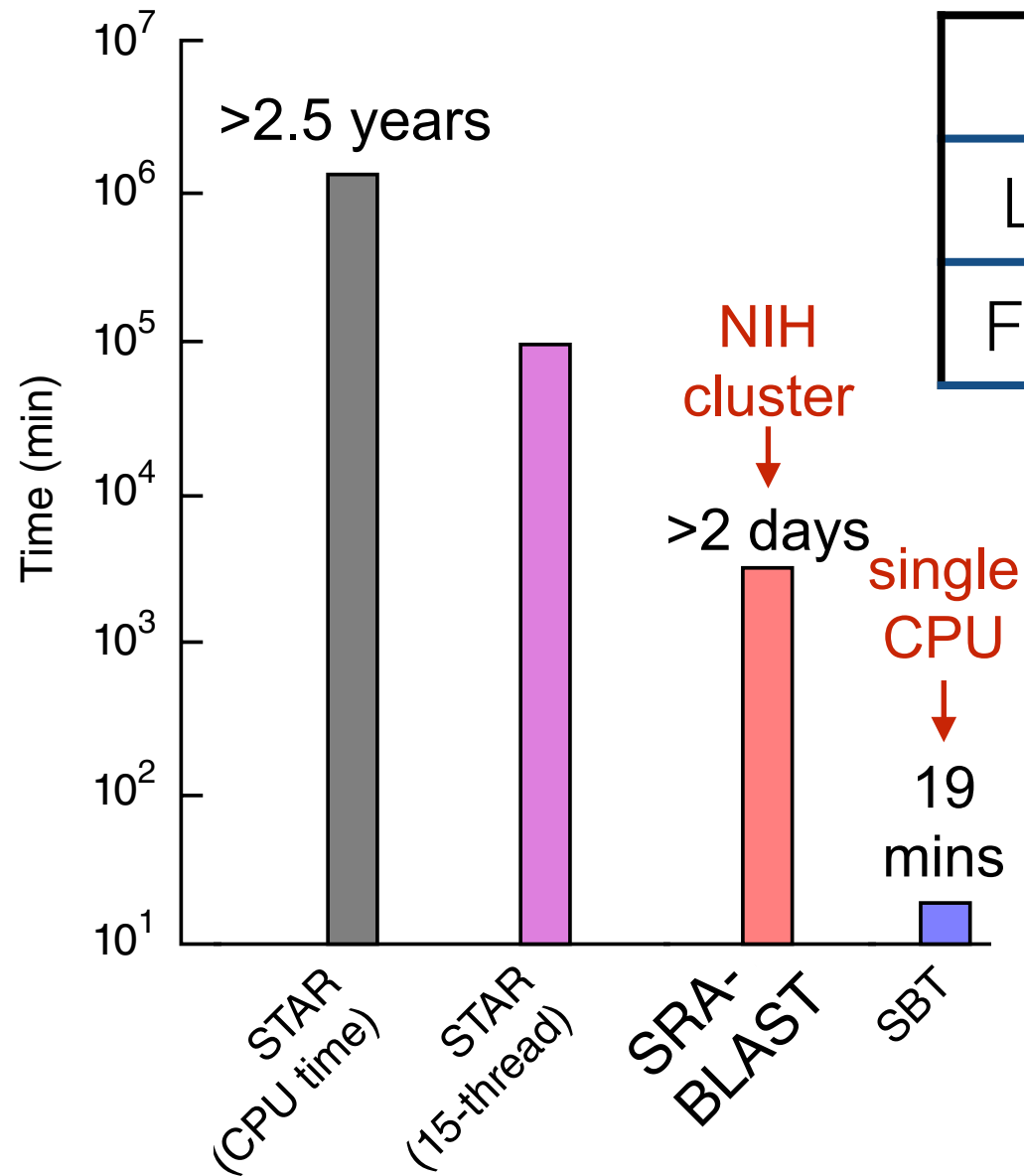


SRA 00008

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...