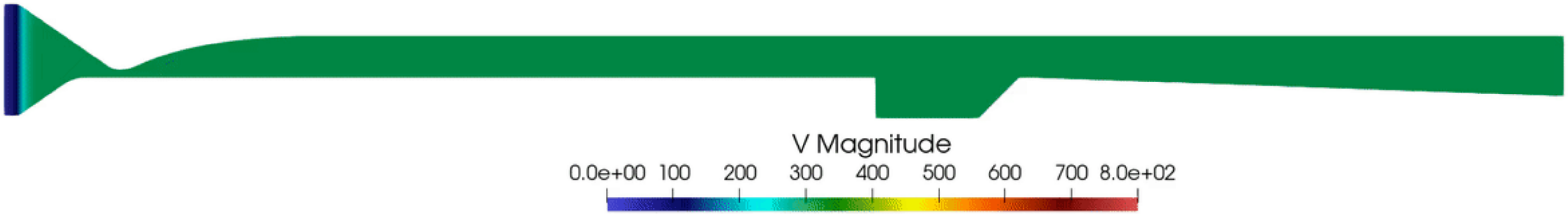
# Data Structures and Algorithms
## Bloom Filters

CS 225
Brad Solomon

April 25, 2022

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science




# Center for Exascale-Enabled Scramjet Design (CEESD)

*Flexible research opportunities in modeling and simulation code development*

| | |
|---|---|
| Summer Research Info: | **go.illinois.edu/ceesd** |
| Project Homepage: | **ceesd.illinois.edu** |
| **Info Contact:** | gcevans@illinois.edu |
| **Summer Dates:** | June 13–August 4 |

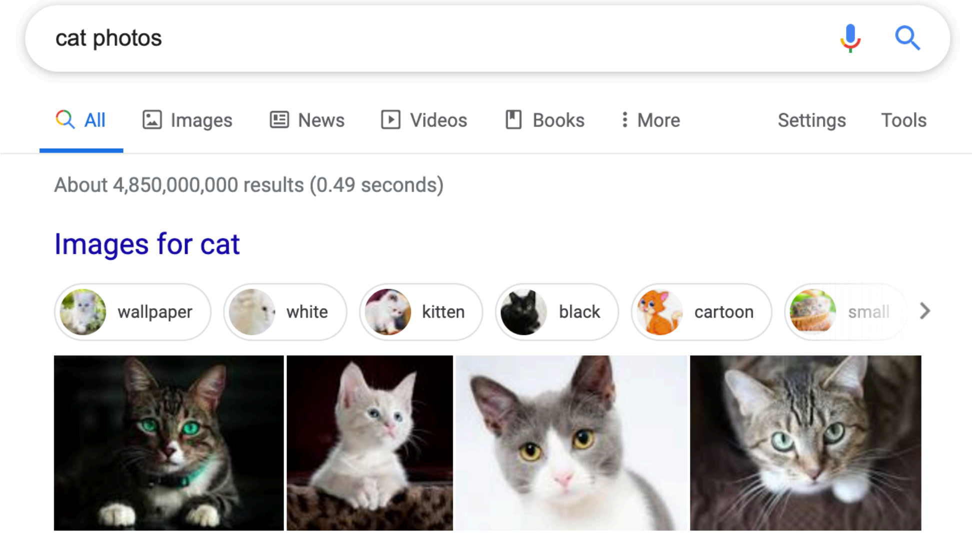
# Data Structures Review

What method would you use to build a search index on a collection of objects?

# 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 $N$)**

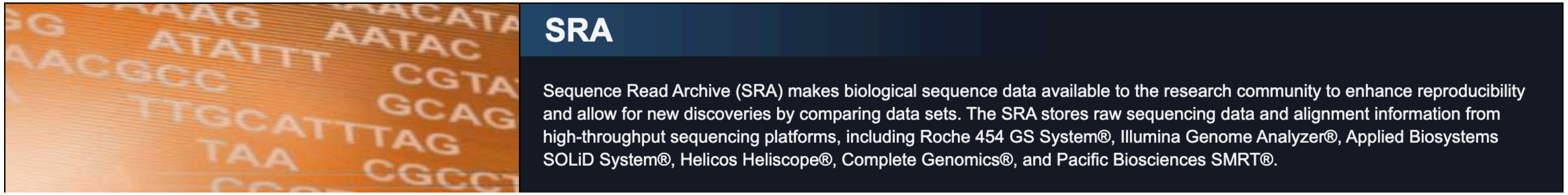
European Nucleotide Archive

1000 Genomes Project

Defining Genetic Variation in People

www.1000genomes.org

GTEx




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 $N$)**

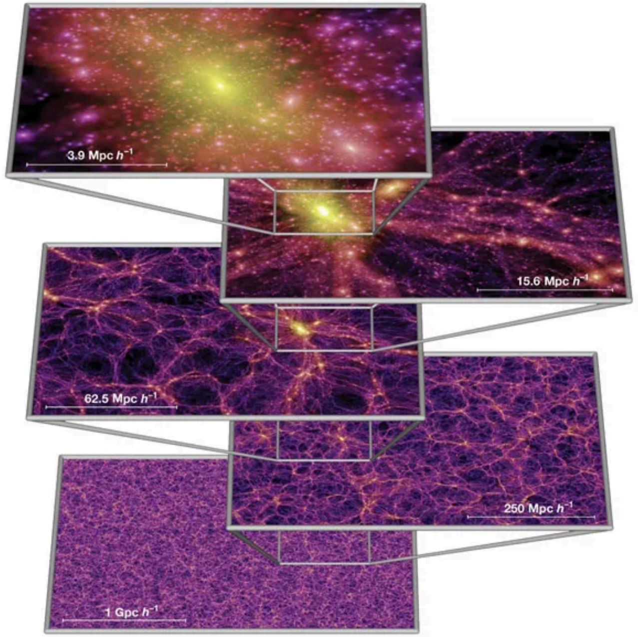



Image: https://doi.org/10.1038/nature03597

| 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 |

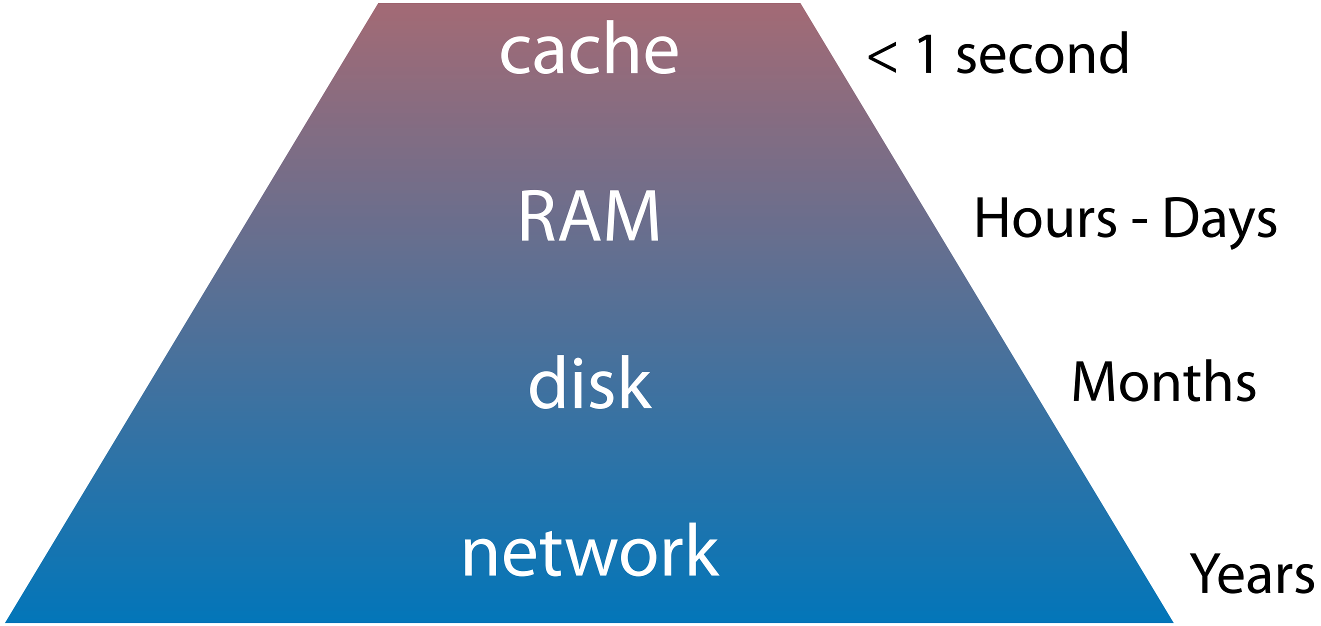
Table: http://doi.org/10.5334/dsj-2015-011

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*?

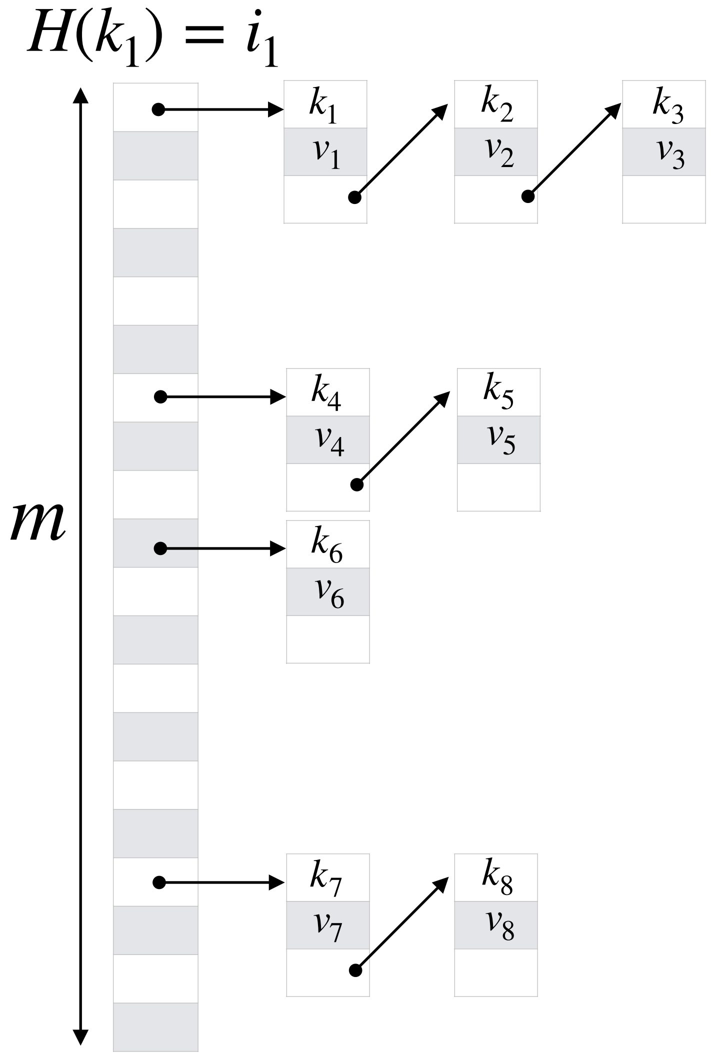
# Reducing storage costs

1)

2)

# Reducing a hash table

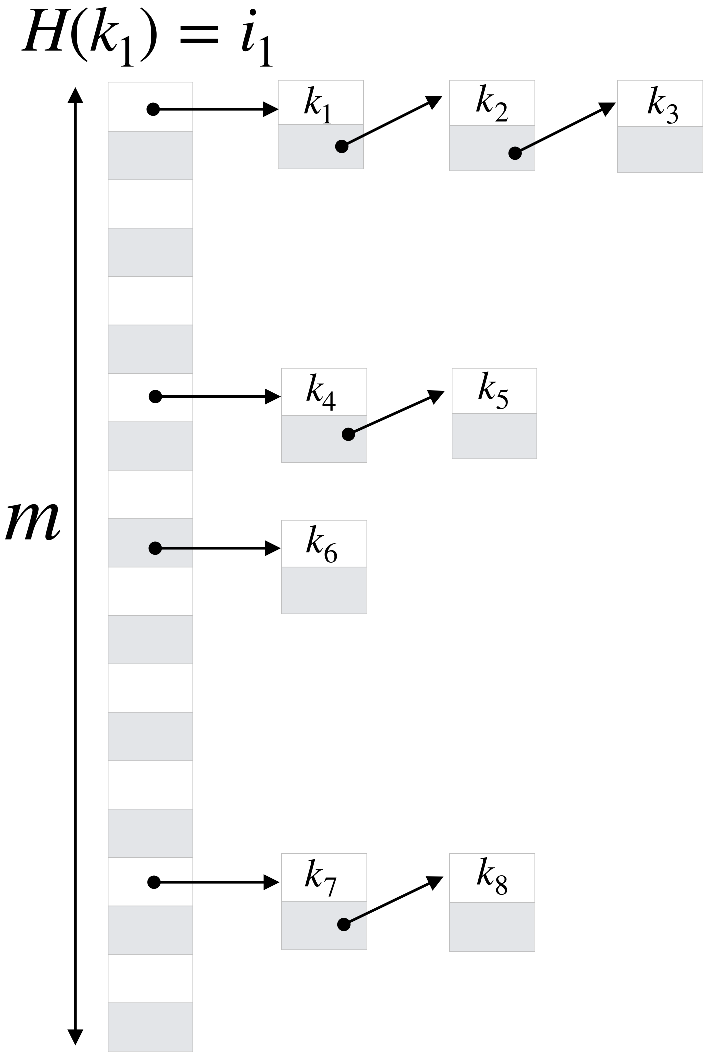
What can we remove from a hash table?



# Reducing a hash table

What can we remove from a hash table?

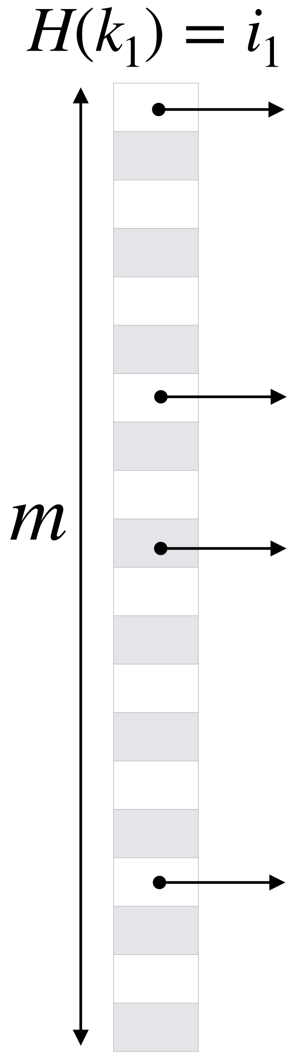
Take away values



# Reducing a hash table

What can we remove from a hash table?

Take away values and keys

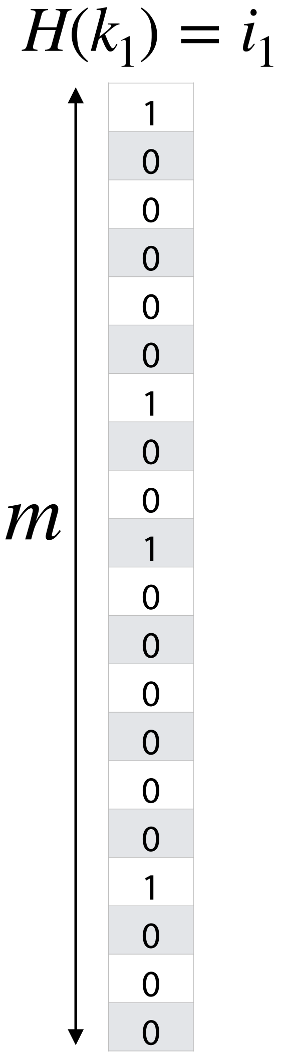


# Reducing a hash table

What can we remove from a hash table?

Take away values and keys

This is a ***bloom filter***



# Learning Objectives

Build a conceptual understanding of a bloom filter

Discuss probabilistic data structures and one-sided error

Formalize the math behind the bloom filter

Introduce extensions to the bloom filter

# Bloom Filter: Insertion

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

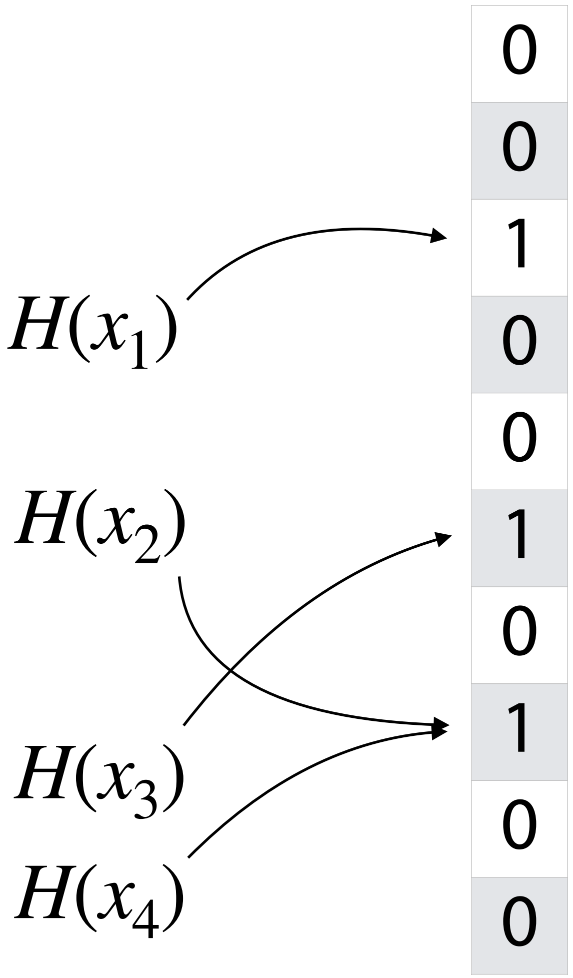
**h(k) = k % 7**

| | |
|---|---|
| 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 }**

`_delete(13)`

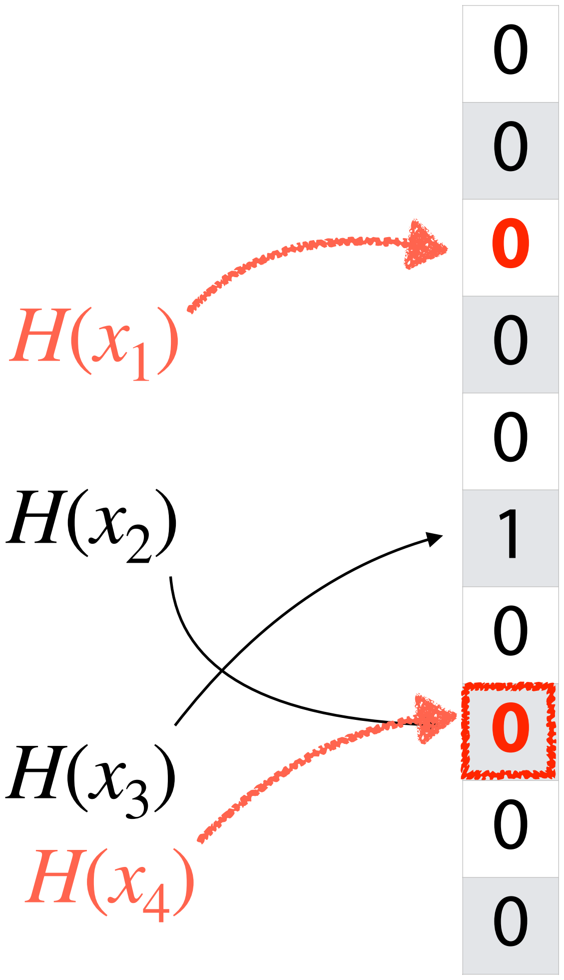
**h(k) = k % 7**

`_delete(29)`

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 0 |
| 4 | 1 |
| 5 | 0 |
| 6 | 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)`

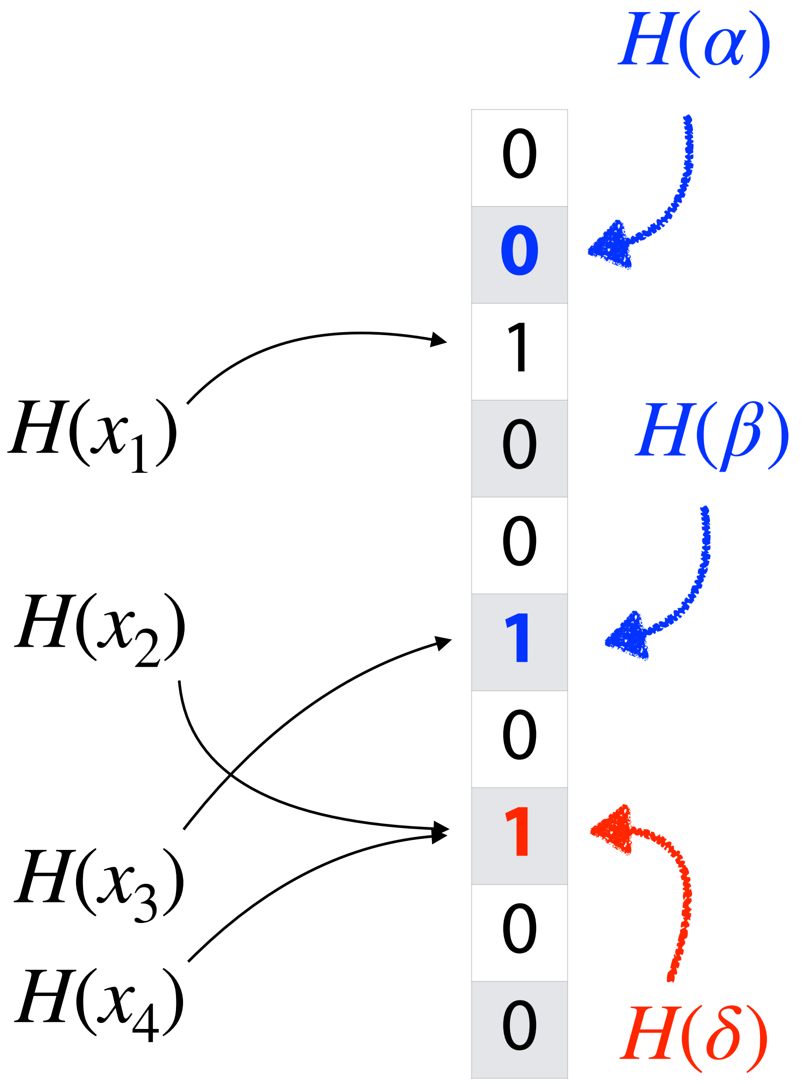
`_find(3)`

# Bloom Filter: Search

The bloom filter is a *probabilistic* data structure!

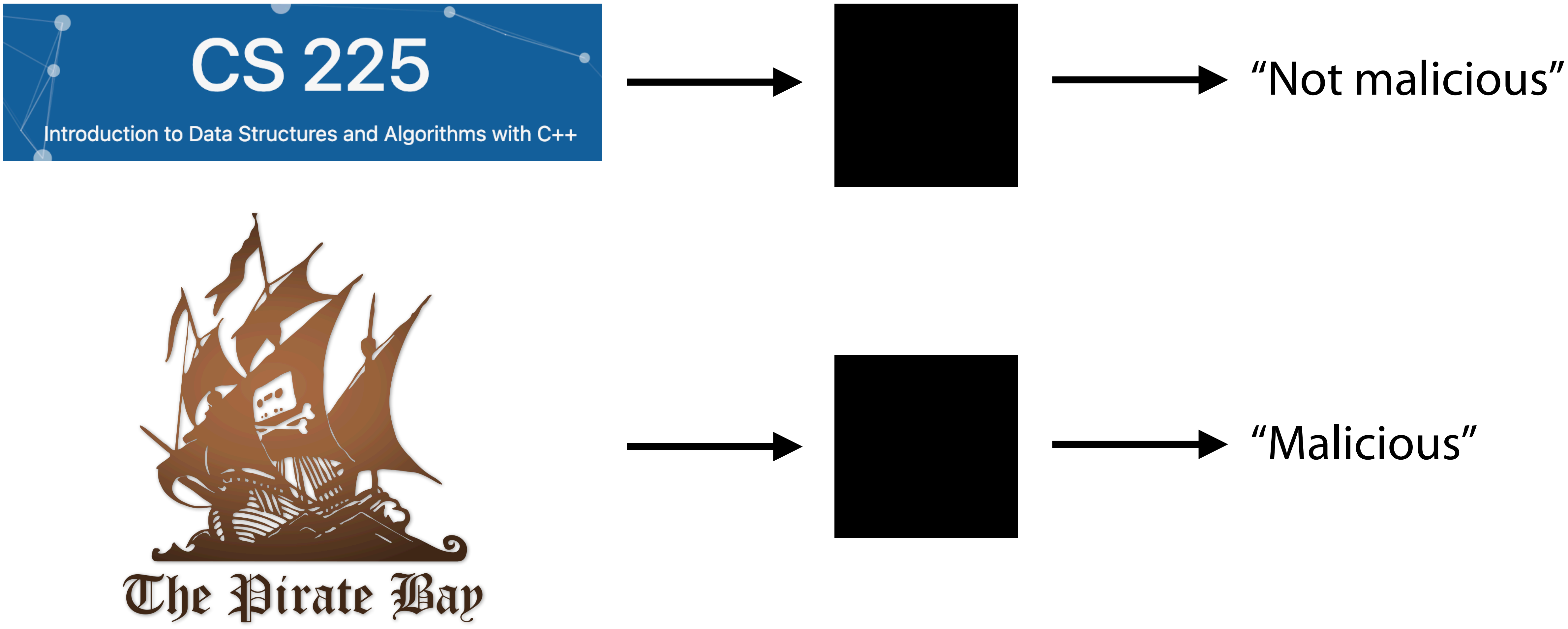
If the value in the BF is 0:

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

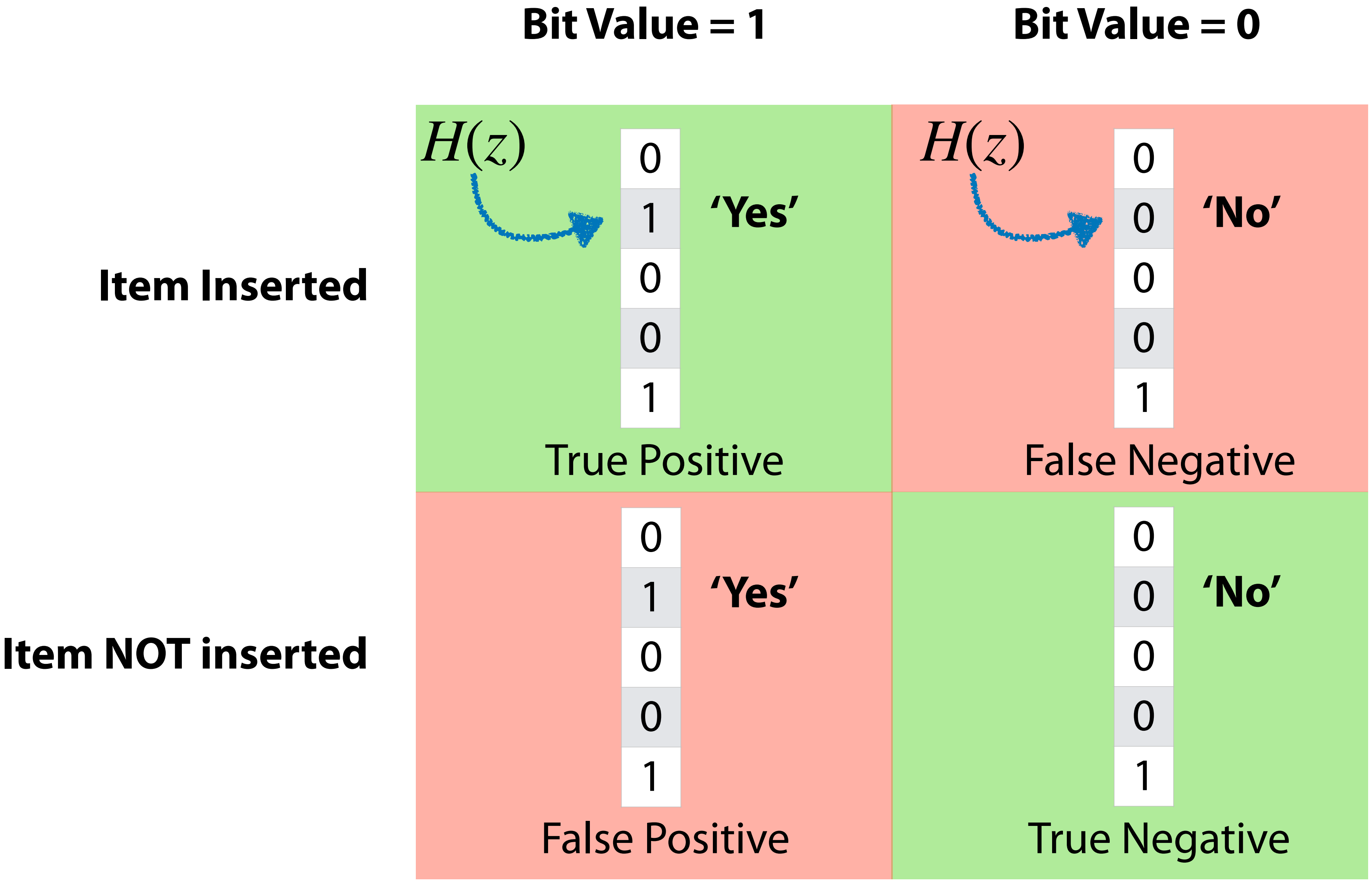
| | |
|---|---|
| True Positive | False Negative |
| False Positive | True Negative |

# Probabilistic Accuracy: Bloom Filters

Are all outcomes possible for the bloom filter?

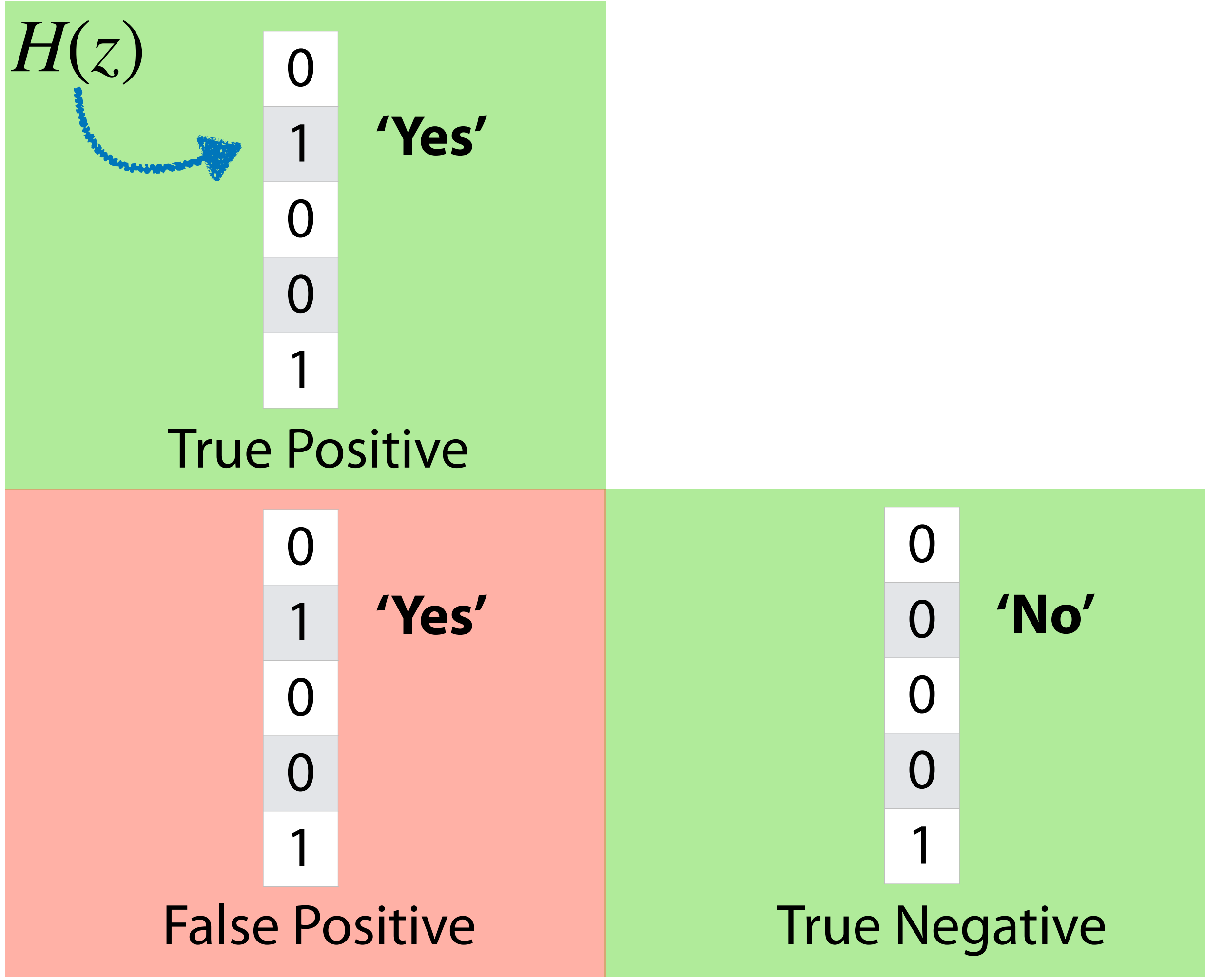
| | |
|---|---|
| True Positive | False Negative |
| False Positive | True Negative |

# Probabilistic Accuracy: Bloom Filters

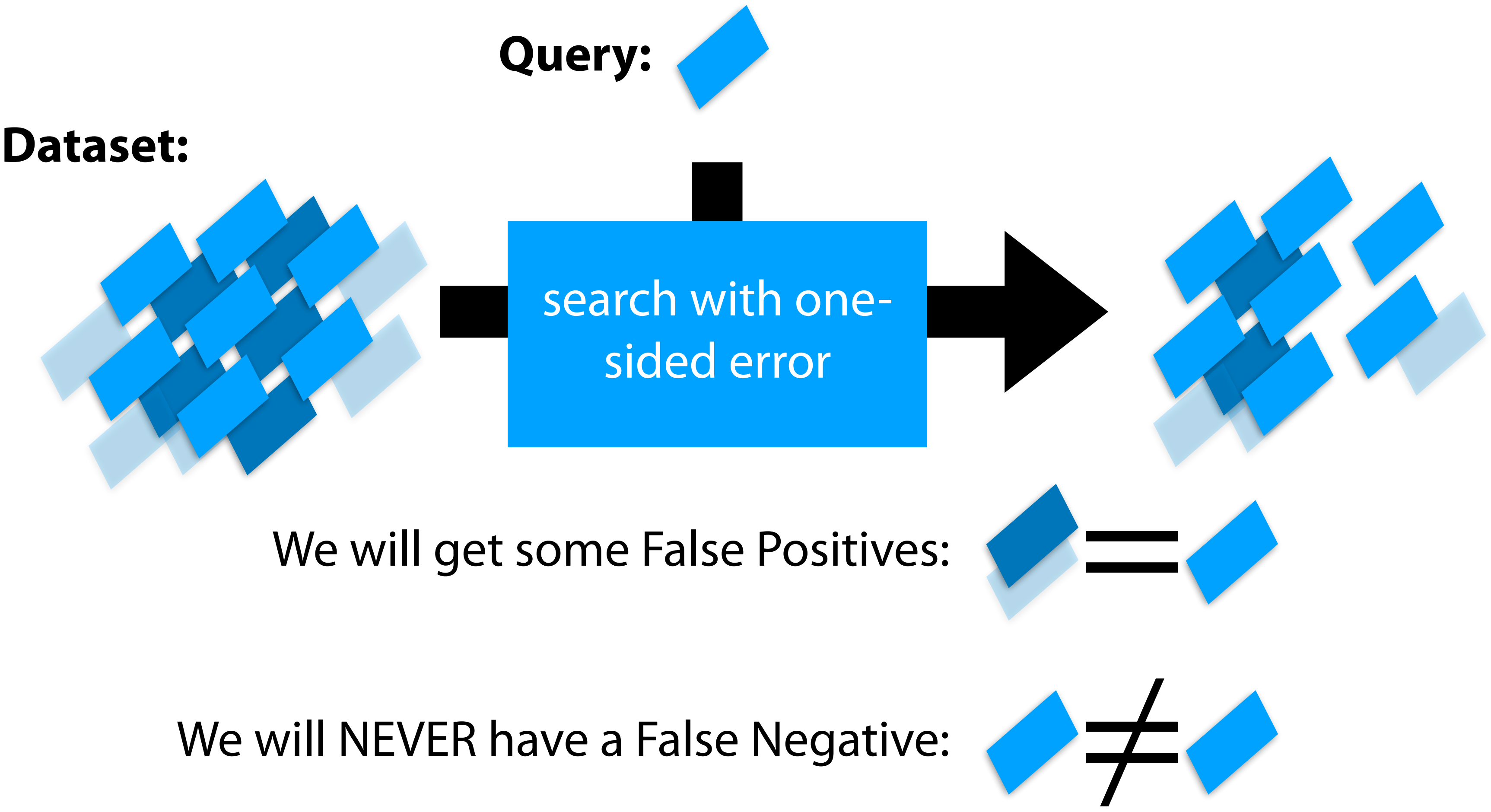


# Probabilistic Accuracy: Bloom Filter

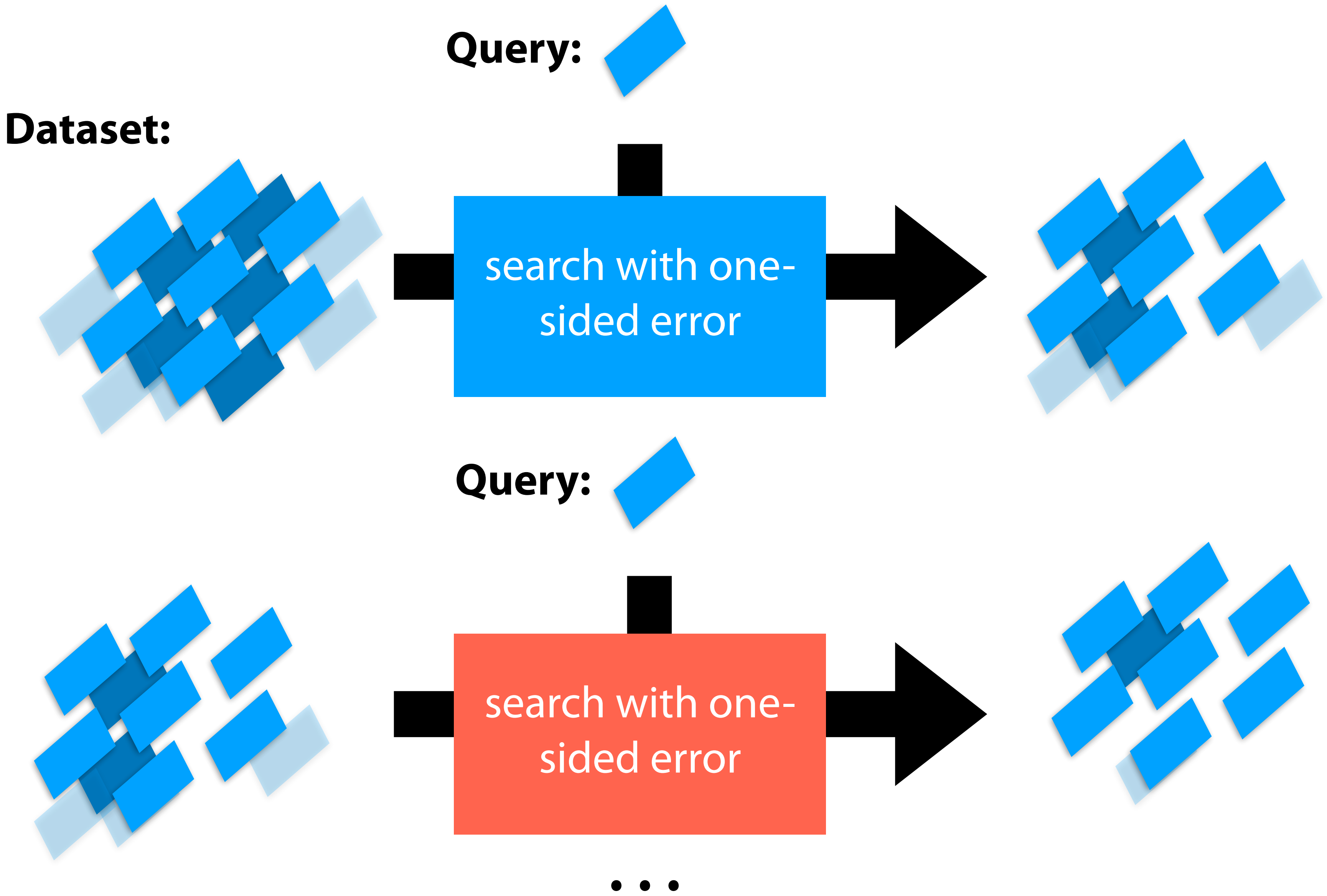
The bloom filter error is 'one-sided' — only false positives are possible!



# Probabilistic Accuracy: One-sided error



# Probabilistic Accuracy: One-sided error

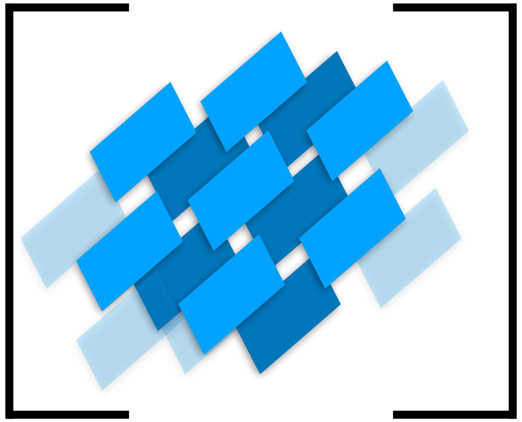


# Bloom Filter: Repeated Trials

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 |

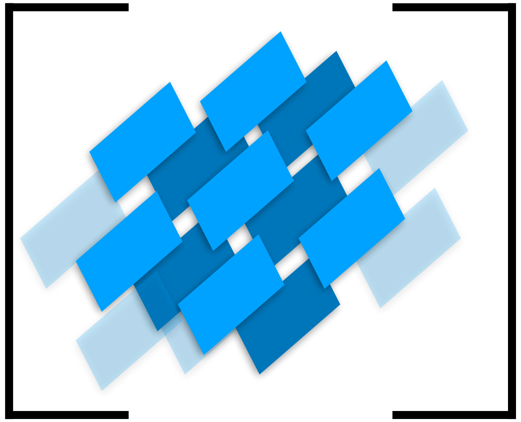
$h_1$



# 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 |

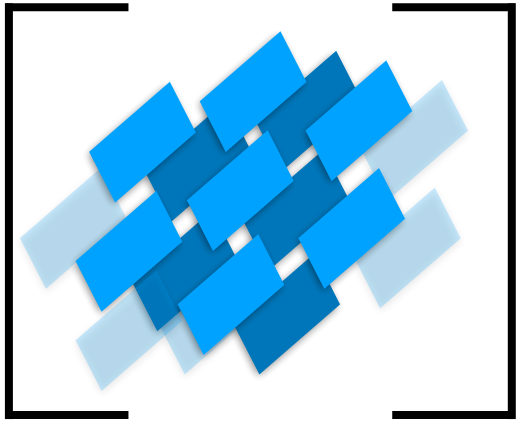


# Bloom Filter: Repeated Trials

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

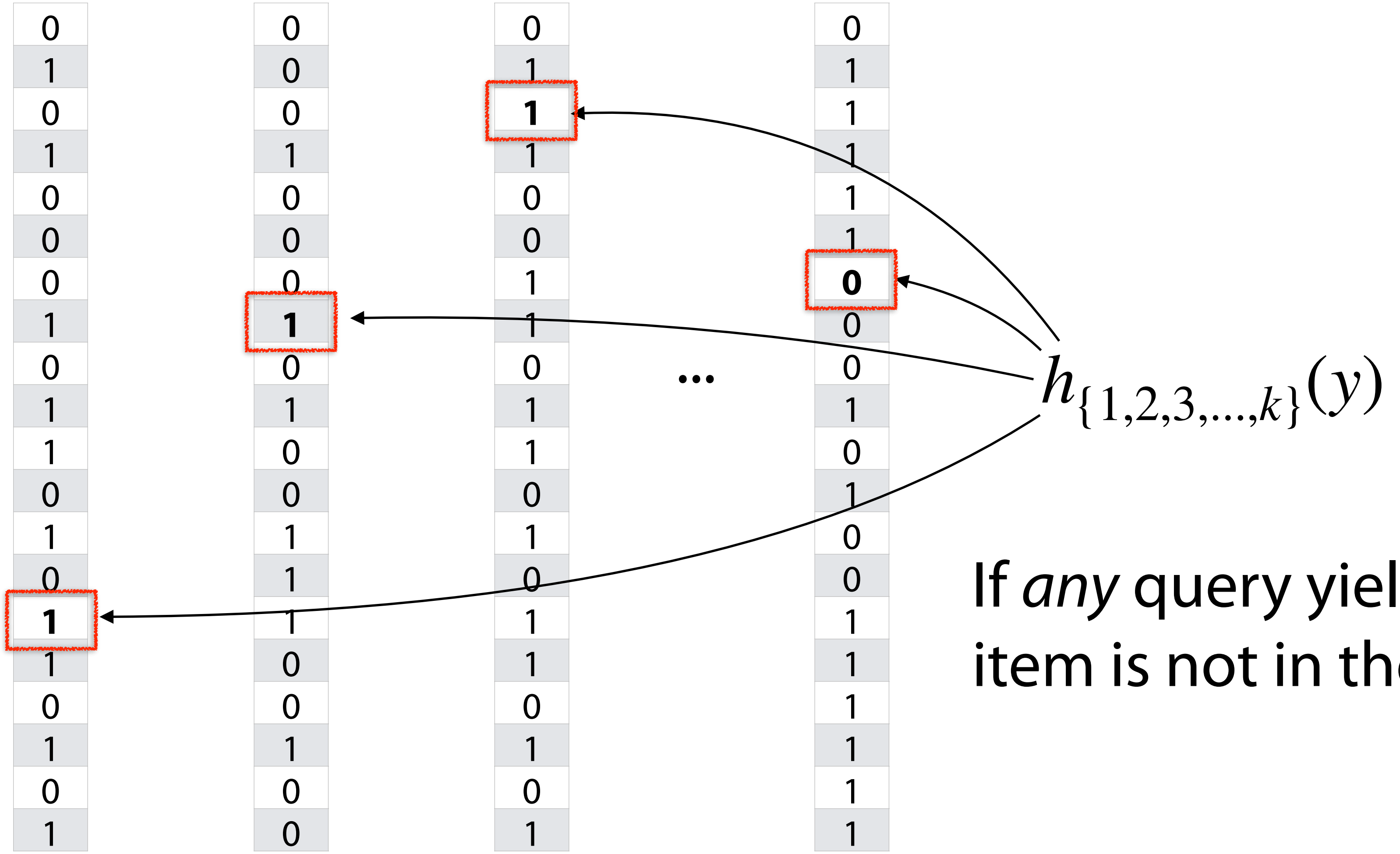
| $h_1$ | $h_2$ | $h_3$ | ... | $h_k$ |
|---|---|---|---|---|
| 0 | 0 | 0 | | 0 |
| 1 | 0 | 1 | | 1 |
| 0 | 0 | 1 | | 1 |
| 1 | 1 | 1 | | 1 |
| 0 | 0 | 0 | | 1 |
| 0 | 0 | 0 | | 1 |
| 0 | 0 | 1 | | 0 |
| 1 | 1 | 1 | | 0 |
| 0 | 0 | 0 | | 0 |
| 1 | 1 | 1 | | 1 |
| 1 | 0 | 1 | | 0 |
| 0 | 0 | 0 | | 1 |
| 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 |

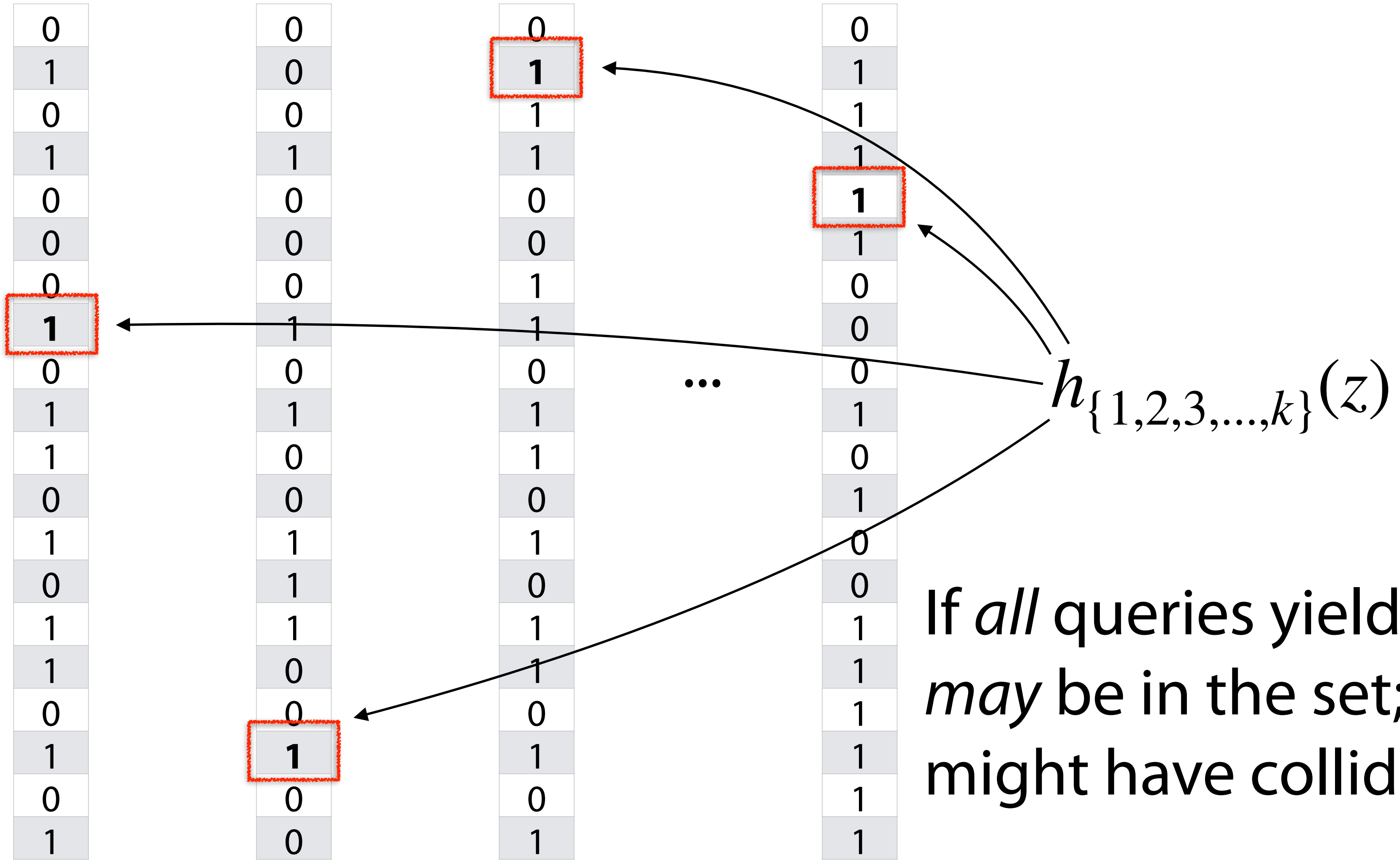
$$h_{\{1,2,3,\ldots,k\}}(y)$$

# 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

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

**$S = \{ 6, 8, 4 \}$**

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

# 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

A probabilistic data structure storing a set of values

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

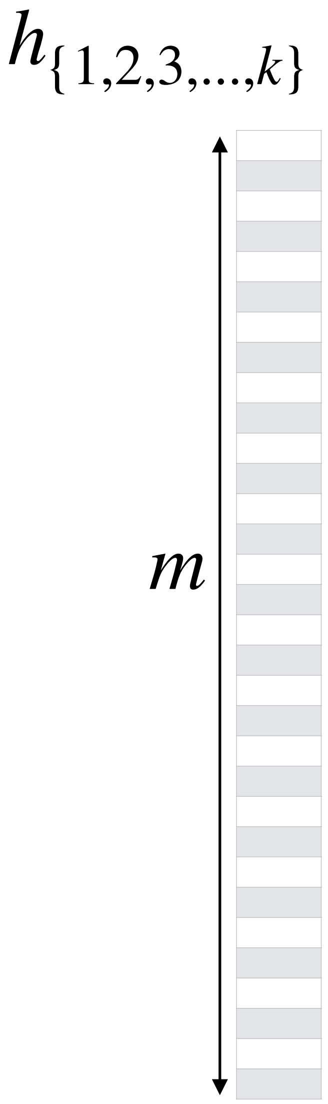
Insert / Find runs in: ________________

| |
|---|
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 1 |
| 0 |
| 0 |

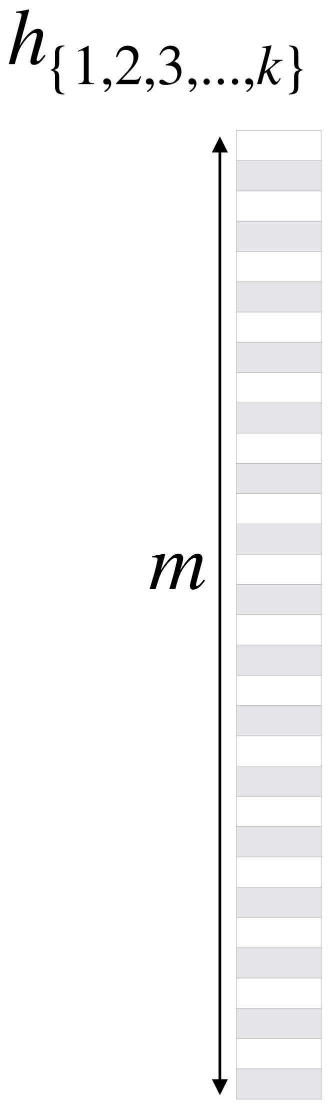
# Bloom Filter: Error Rate

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



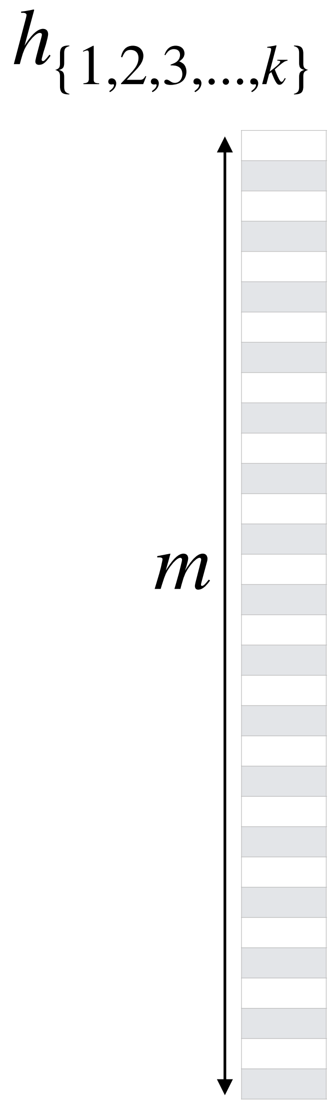

Given bit vector of size $m$ and 1 SUHA hash function

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

# Bloom Filter: Error Rate

Given bit vector of size $m$ and $k$ SUHA hash function

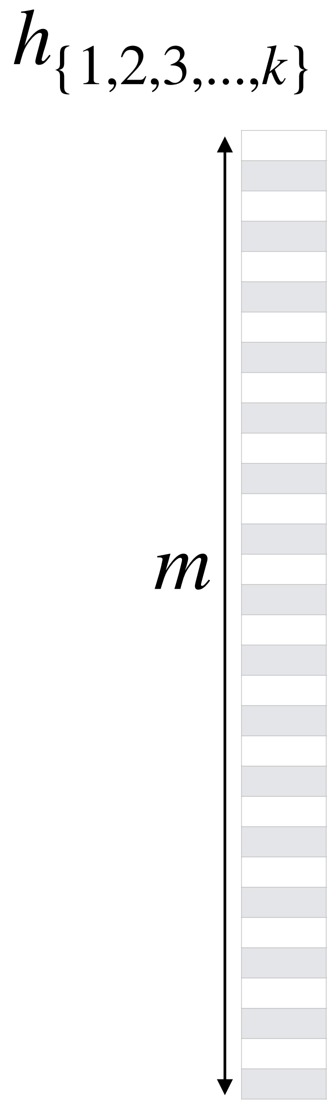
What's the probability a specific bucket is 1 after one object is inserted?



# Bloom Filter: Error Rate

Given bit vector of size $m$ and $k$ SUHA hash function

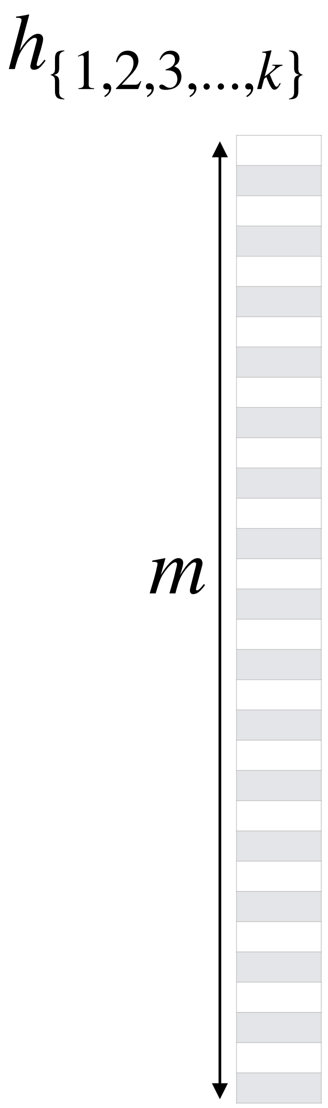
What's the probability a specific bucket is 0 after one object is inserted?



# Bloom Filter: Error Rate

Given bit vector of size $m$ and $k$ SUHA hash function

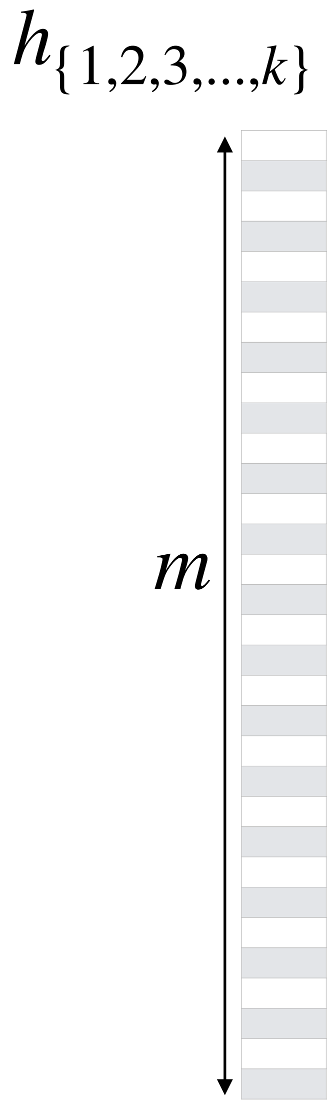
What's the probability a specific bucket is 0 after $n$ objects are inserted?



# 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

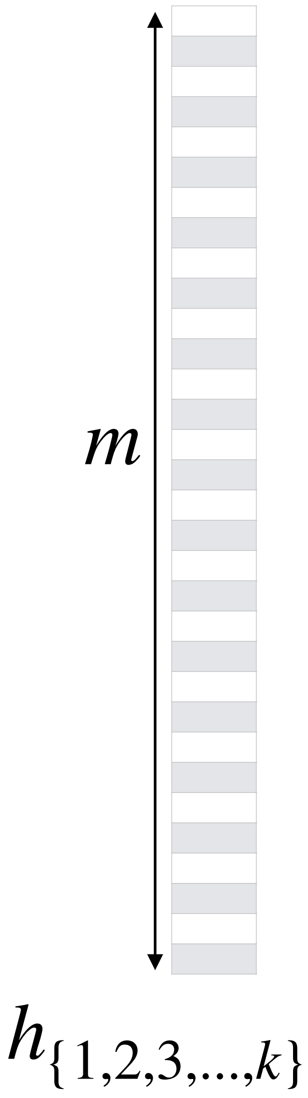
Given bit vector of size $m$ and $k$ SUHA hash function

**What is our expected FPR after $n$ objects are inserted?**

The chance my bit is 1 by chance after $n$ objects inserted

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

The number of [assumed independent] trials



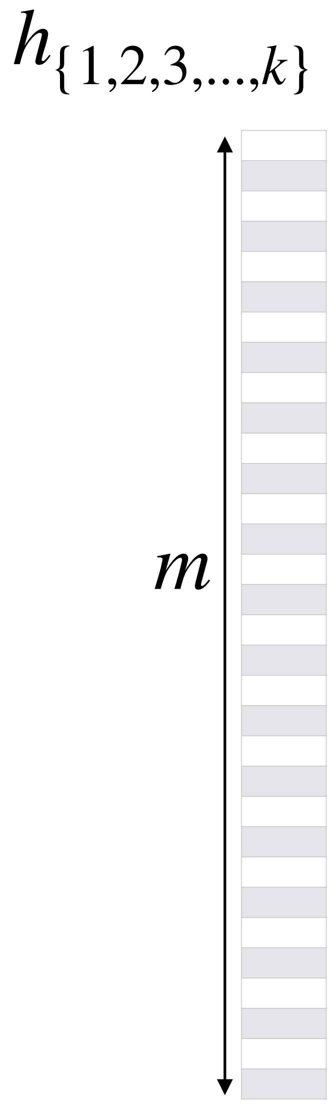
# Bloom Filter: Error Rate

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

So how can we find the minimum error rate?

# Bloom Filter: Error Rate

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

"Mercator Series"

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

# Bloom Filter: Error Rate

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

# Bloom Filter: Error Rate

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

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

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

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

# Bloom Filter: Error Rate

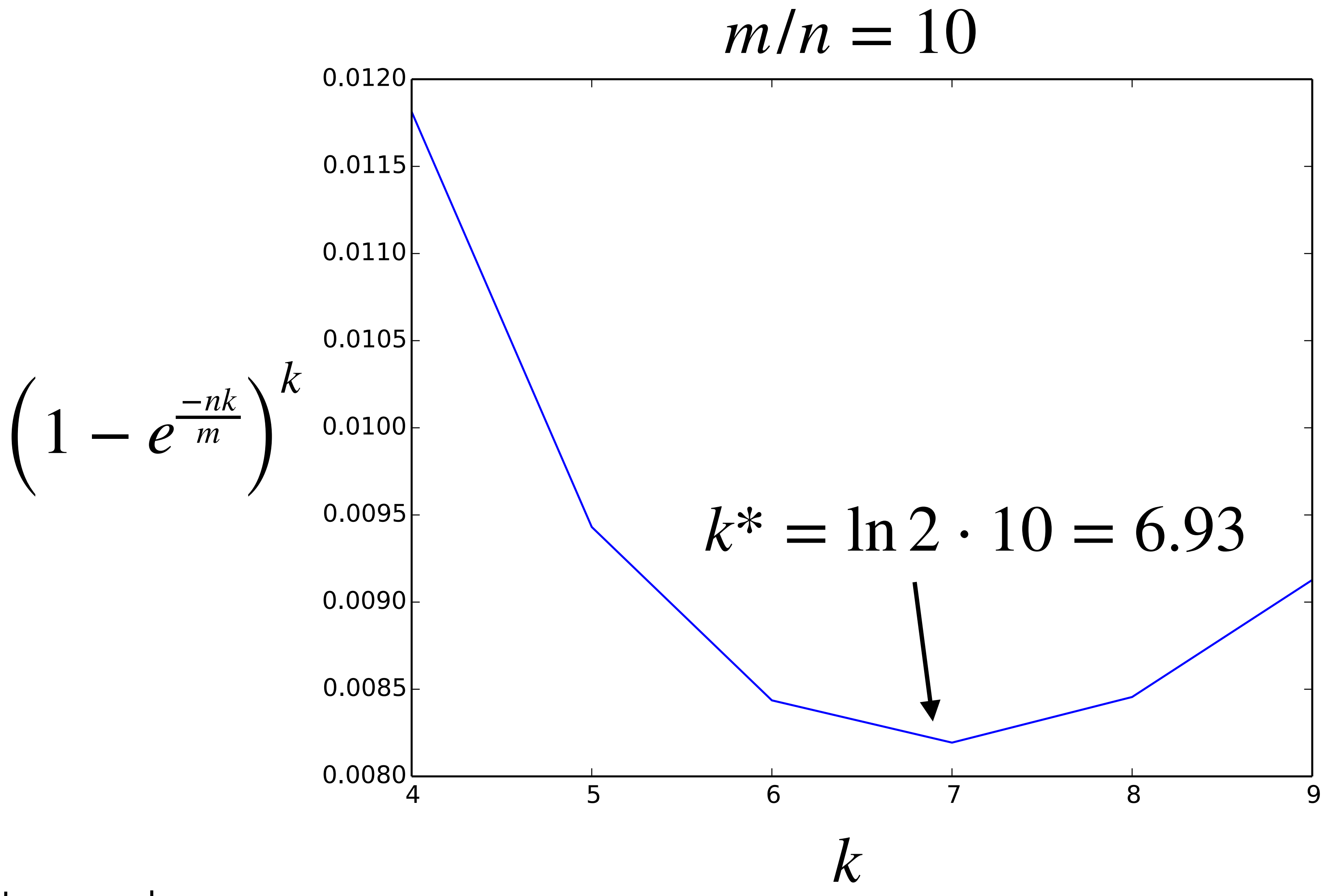



Figure by Ben Langmead

# Bloom Filter: Optimal Parameters

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

**Given any two values, we can optimize the third**

$n = 100$ items $\quad k = 3$ hashes $\quad m = \dfrac{300}{ln(2)} \approx 433$ bits

$m = 100$ bits $\quad n = 20$ items $\quad k = ln(2) * \dfrac{100}{20} \approx 3.47$ hashes

$m = 100$ bits $\quad k = 2$ items $\quad n = ln(2) * \dfrac{100}{2} \approx 34.6$ items

# Bloom Filters

$k$, number of hash functions

$n$, expected number of insertions

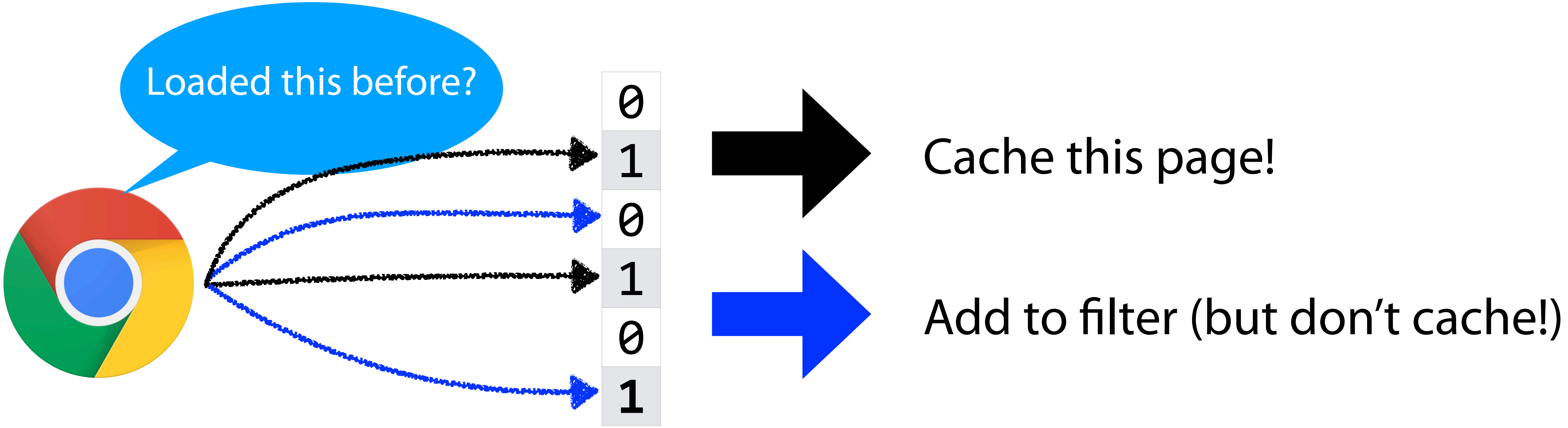
$m$, filter size in bits

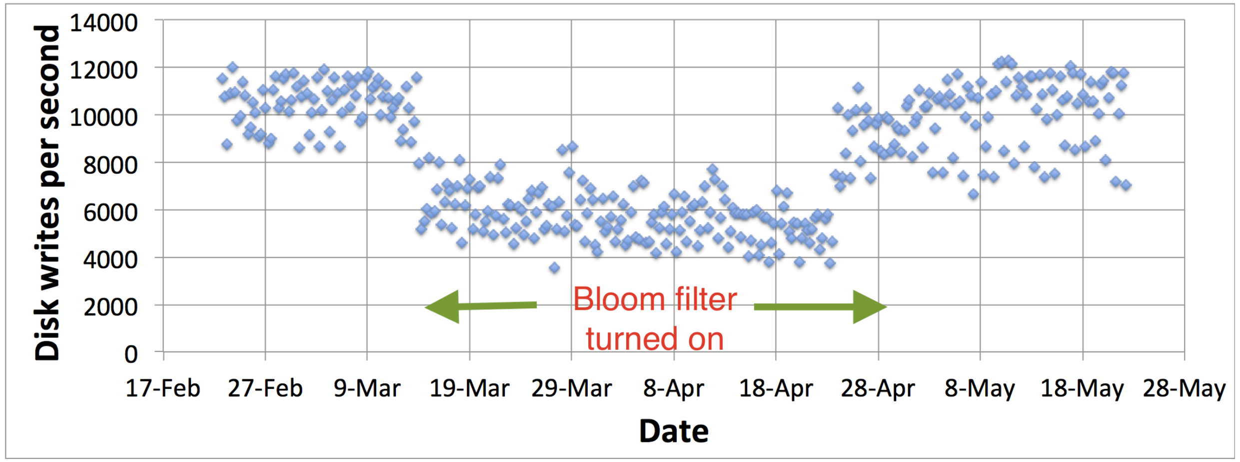
Ideal number of hash functions: $$k^* = \ln 2 \cdot \frac{m}{n}$$

Optimal Bloom Filter does not change asymptotic scaling!

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

# Bloom Filter: Website Caching



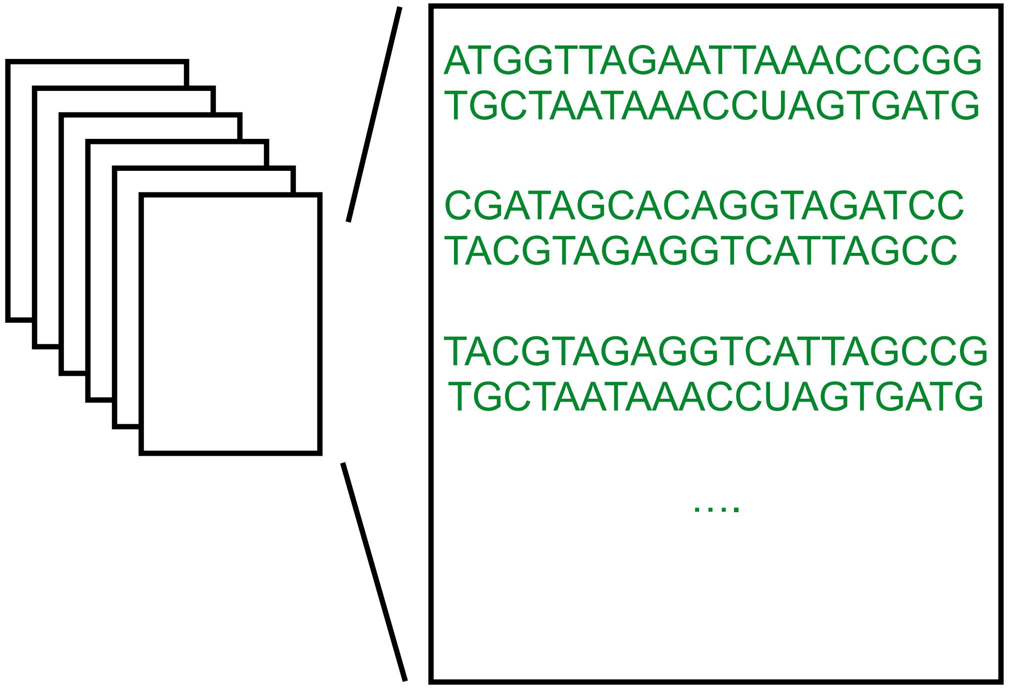





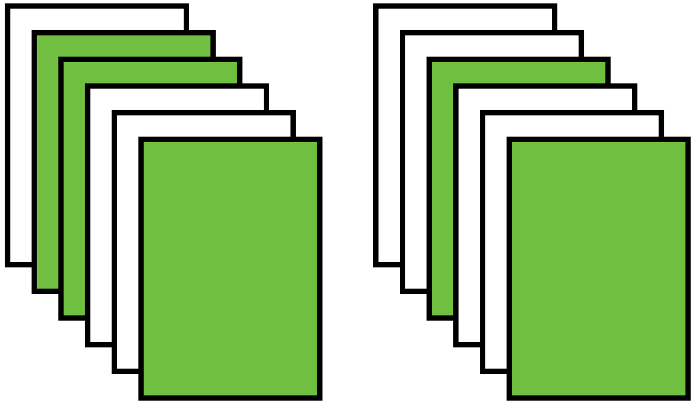
Maggs, Bruce M., and Ramesh K. Sitaraman. **Algorithmic nuggets in content delivery.** *ACM SIGCOMM Computer Communication Review* 45.3 (2015): 52-66.

# Sequence Bloom Trees

Imagine we have a large collection of text…




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

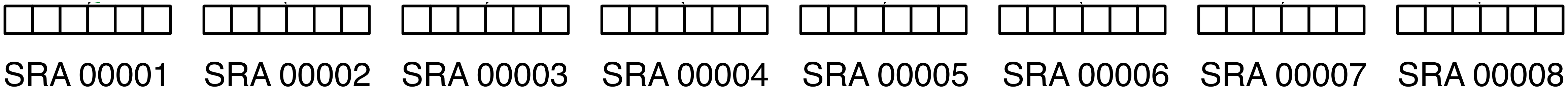


# Bloom Filters: Unioning

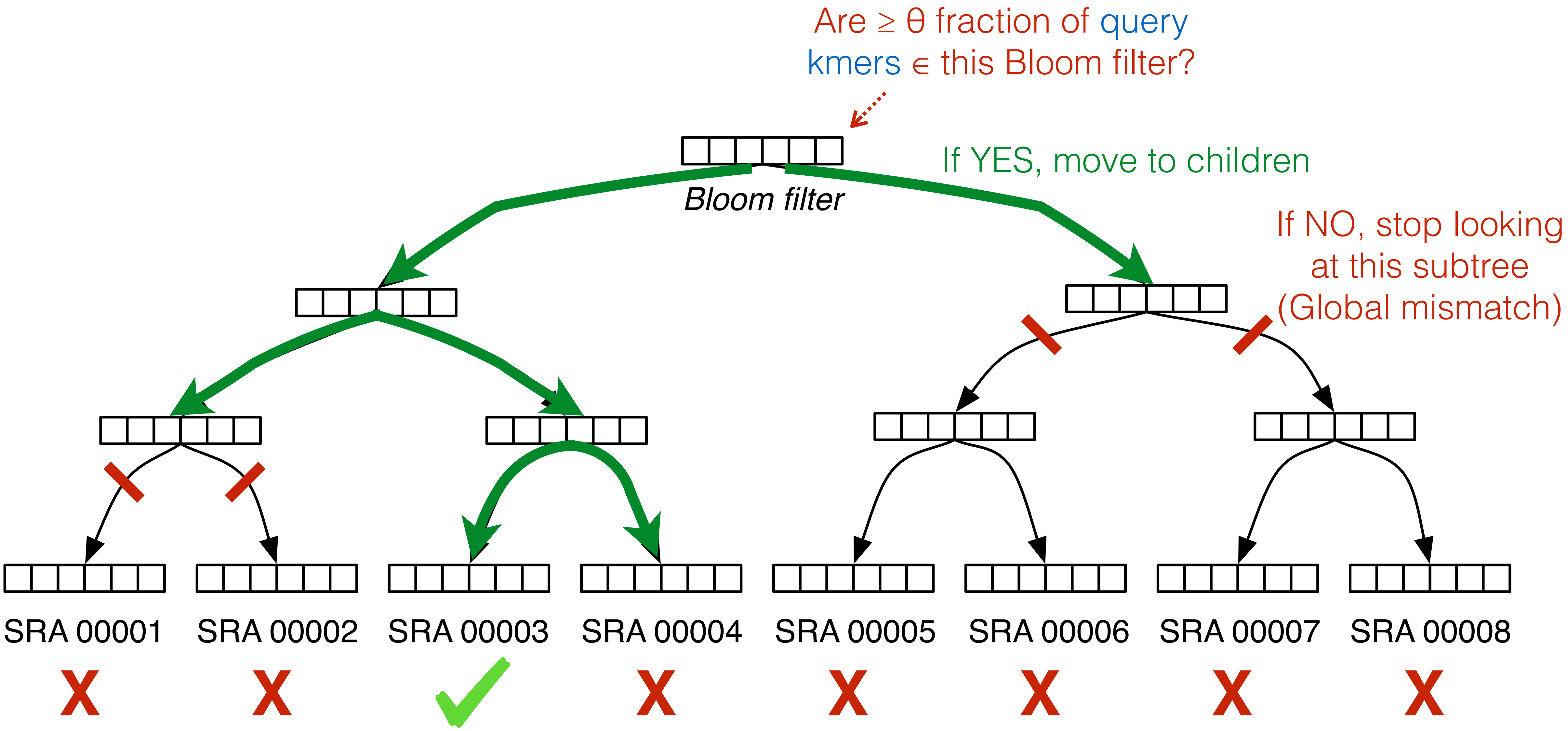
Bloom filters can be trivially merged using bit-wise union.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 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 | |

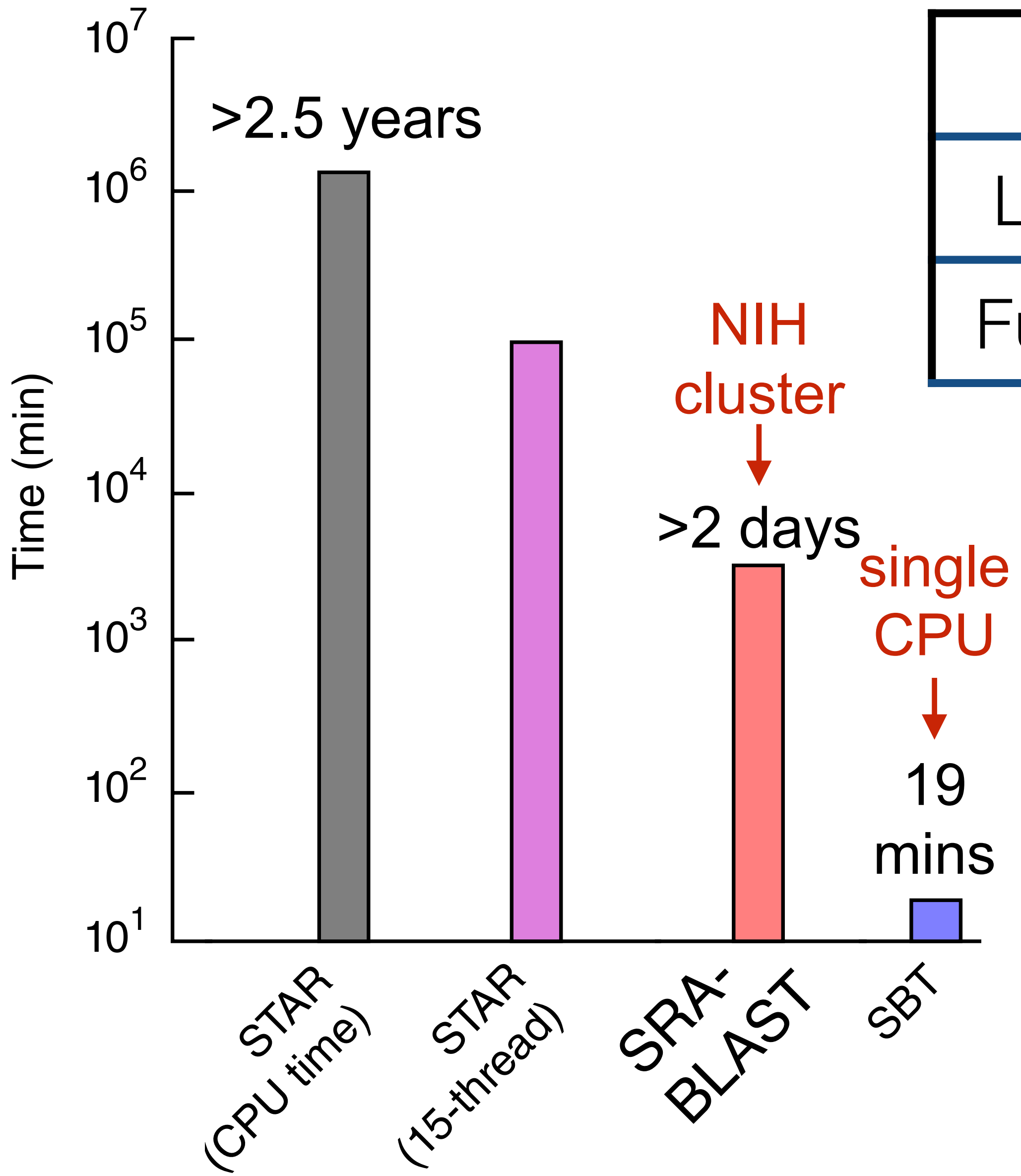
# Sequence Bloom Trees



# 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…