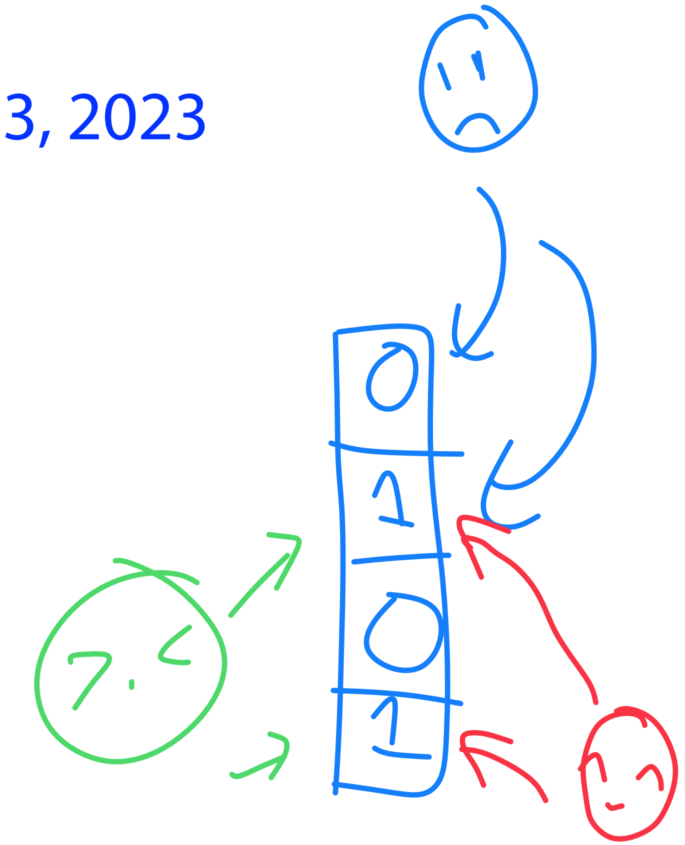
Data Structures and Algorithms

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

next 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

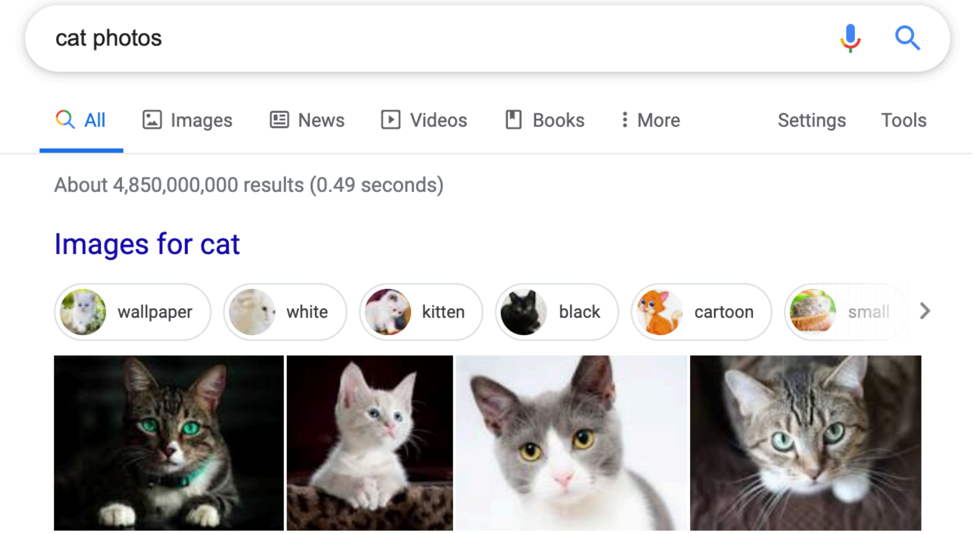
Formalize the math behind the bloom filter

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

H(x) (one or more)

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

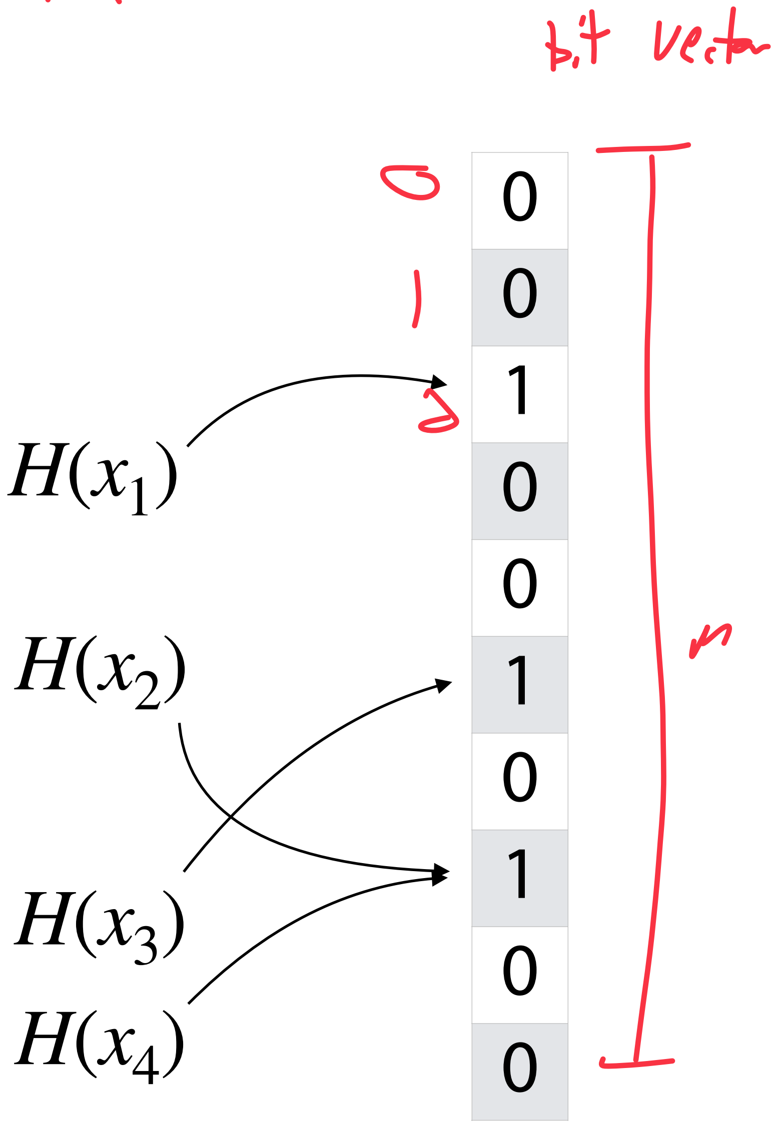
If the bit was already one, it stays 1

↳ No collision management!

Data Point: x_1

Hash x_1: $H(x_1) = 2$ (hash value index)

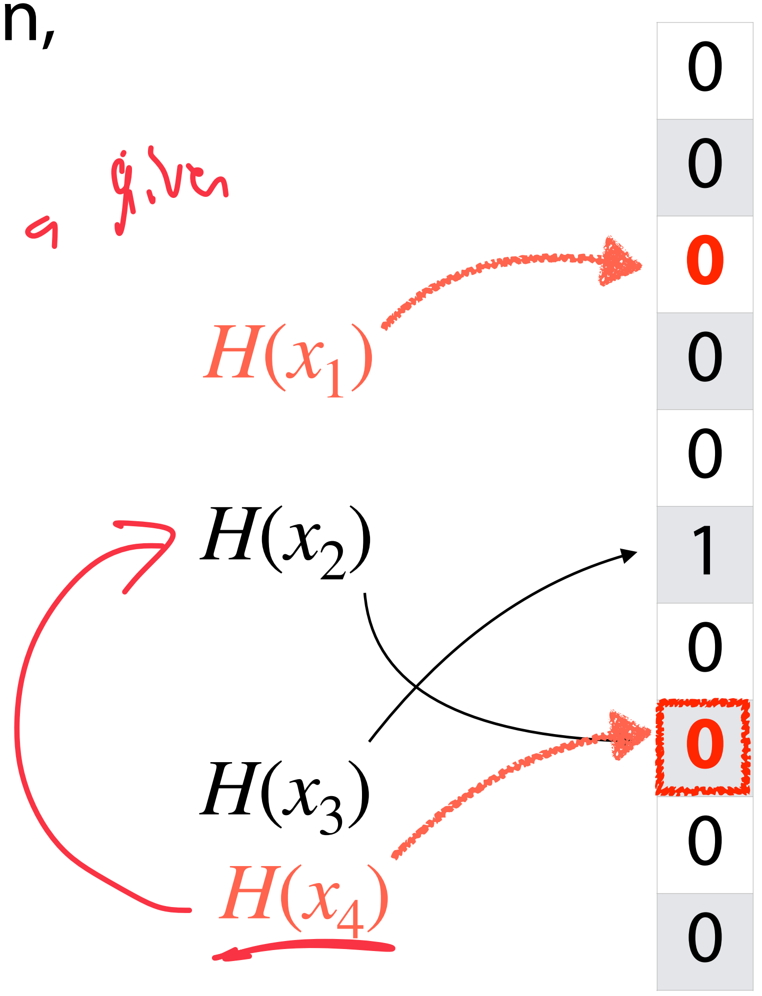
↳ Set bit at index $H(x_1)$ to 1



Bloom Filter: Deletion

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

No idea # of references to a given index.



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

2

`_find(16)`

↳ h(16) = 2

Bit is one so yes!

`_find(20)`

↳ h(20) → 6

↳ Yes!

This is a false positive

`_find(3)`

↳ h(3)

↳ No

Bloom Filter: Search

Thrown out information
tradeoff is loss in accuracy

The bloom filter is a *probabilistic* data structure!

If the value in the BF is 0:

↳ 100% of the time, item is not present

↳ No false negatives

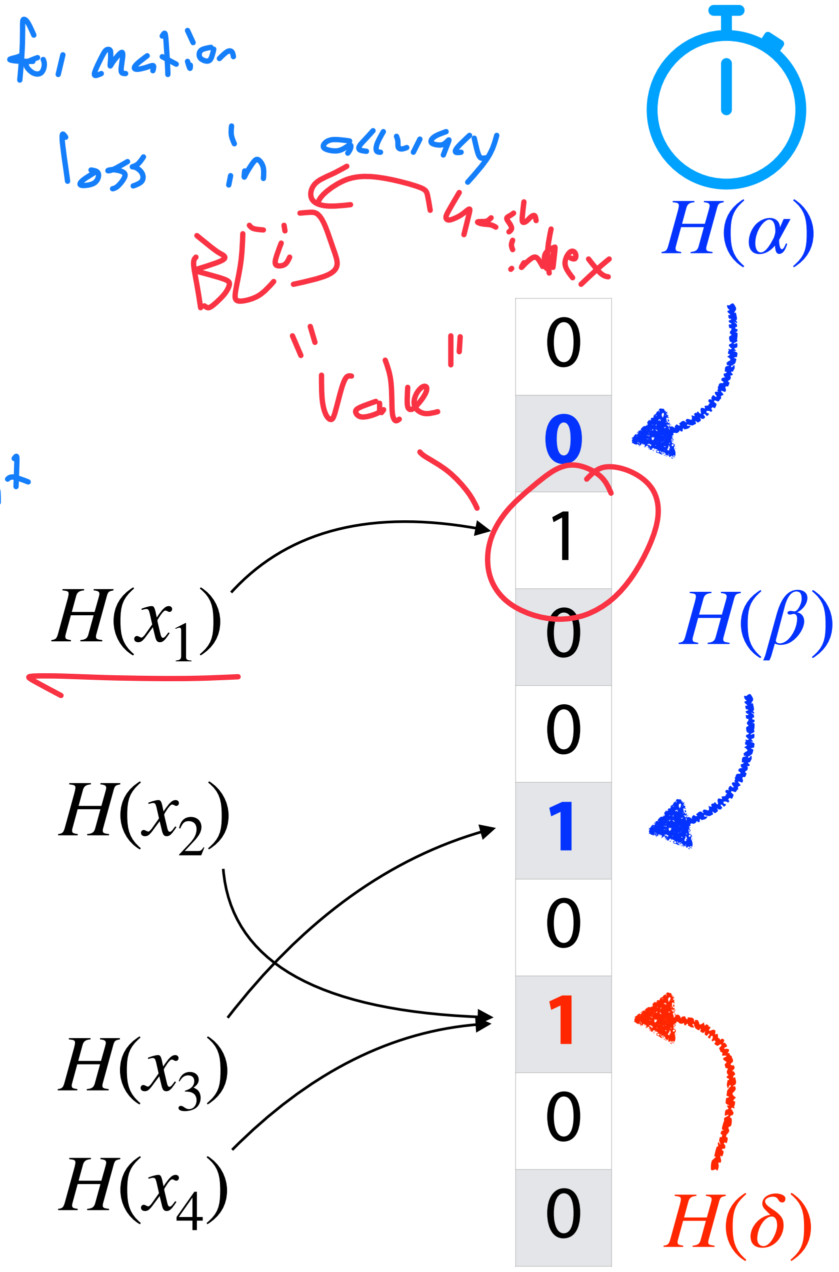
If the value in the BF is 1:

↳ The object might be present

either I insert

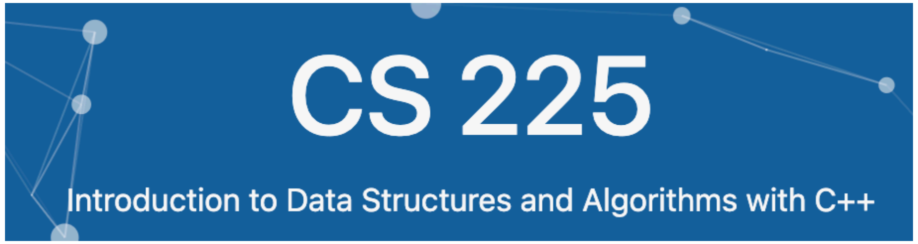
or

I hash collided

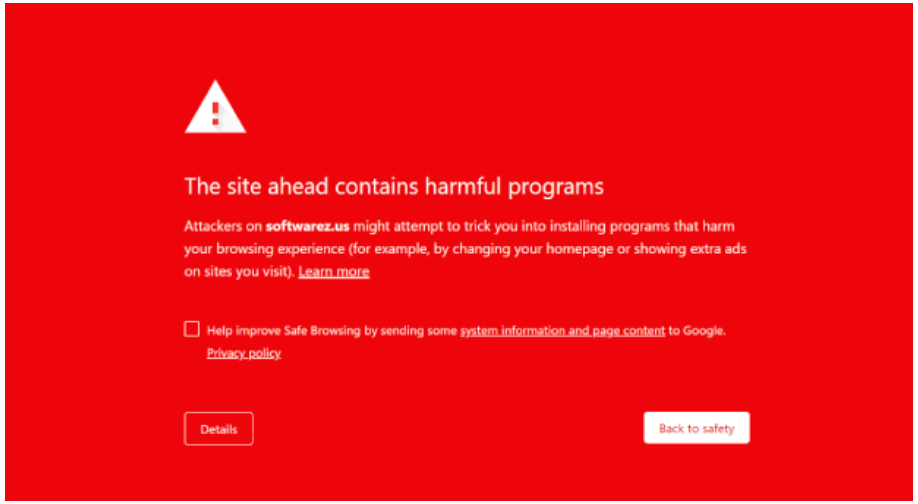


Probabilistic Accuracy: Malicious Websites

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: Oracle says Malicious | Prediction says malicious

False Positive: Oracle says safe | Prediction says malicious

False Negative: Actual Malicious | Predict Safe

True Negative: Safe | predict Safe

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

predicted

Actual truth

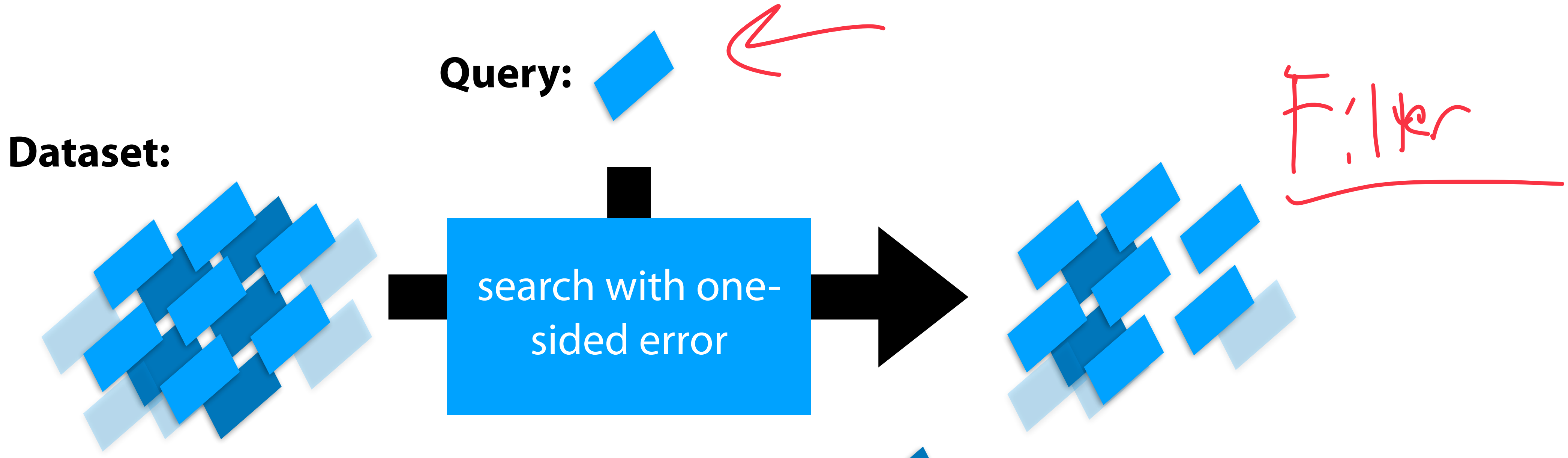
	Bit Value = 1 ↳ Yes	**Bit Value = 0** ↳ No
Item Inserted	$H(z)$ 0 1 0 0 1 **'Yes'** True Positive	$H(z)$ 0 0 0 0 1 **'No'** False Negative
Item NOT inserted	H(x) H(y) 0 1 0 0 1 **'Yes'** False Positive	0 0 0 0 1 **'No'** True Negative

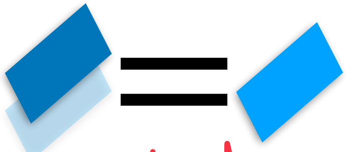
cannot

Impossible happen

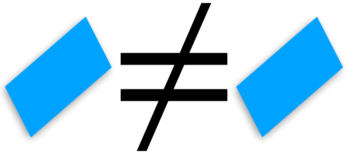
One sided error

Probabilistic Accuracy: One-sided error



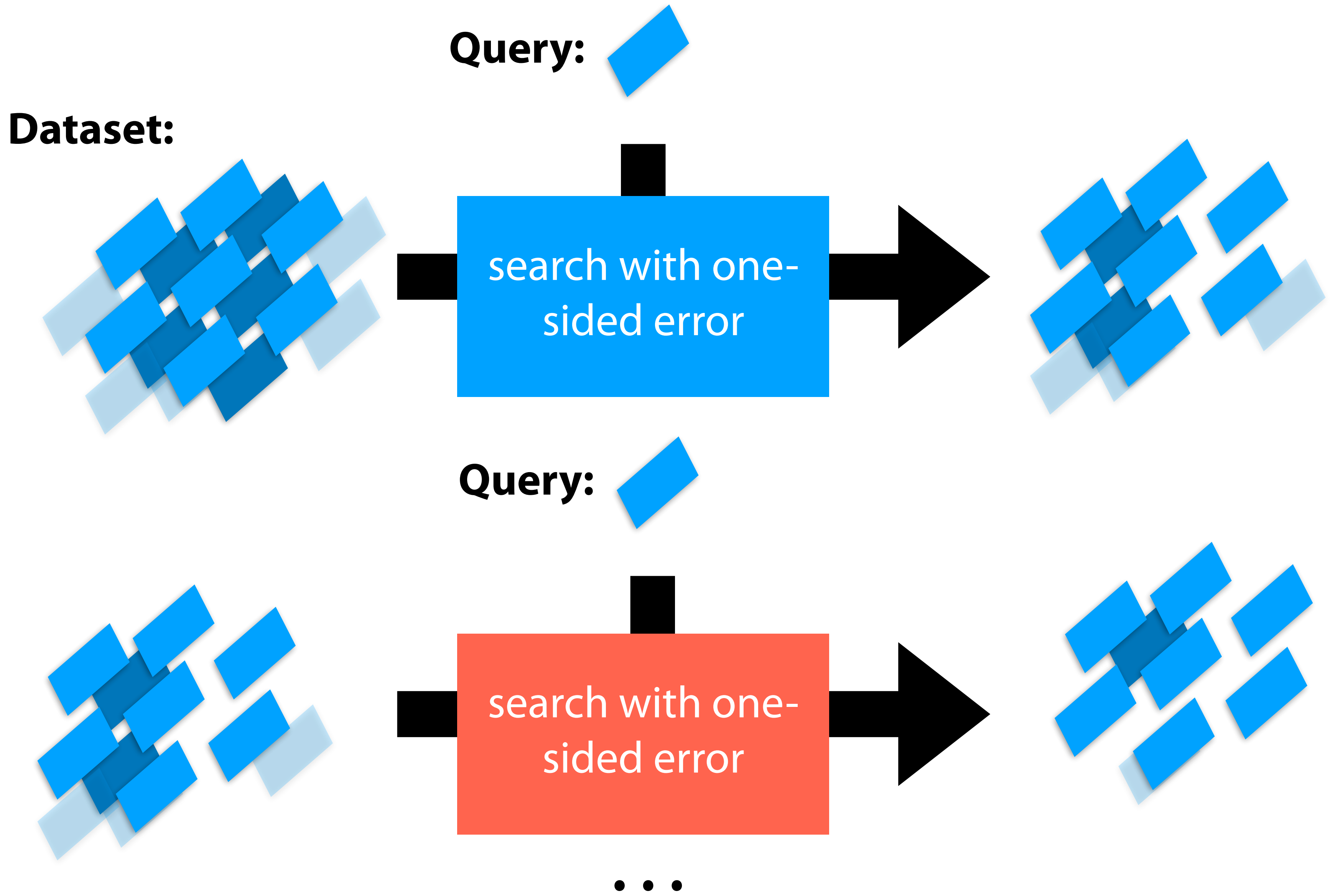
We will get some False Positives: 

↳ Get some bad

We will NEVER have a False Negative: 

↳ Can't throw out good

Probabilistic Accuracy: One-sided error

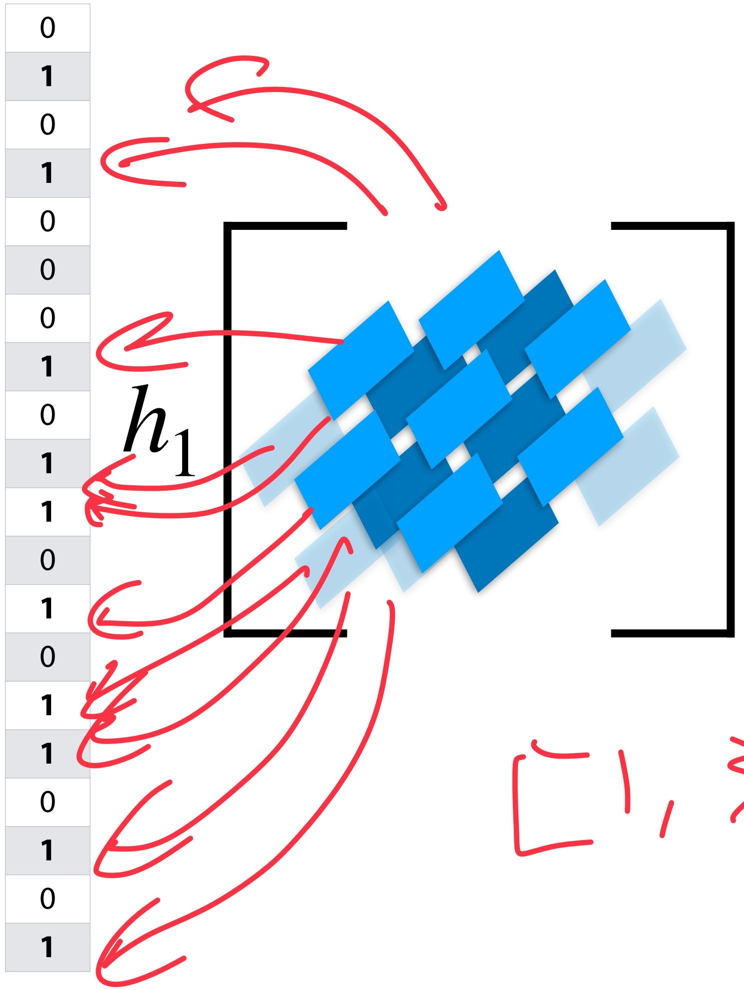


Bloom Filter: Repeated Trials

Talking present
absence

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

Does it exist
or
not



Does 2 exist?

[1, 3, 5, 7]

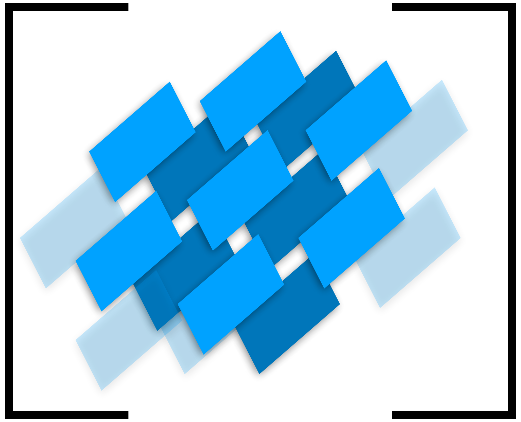
Bloom Filter: Repeated Trials

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

h_1

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

h_2

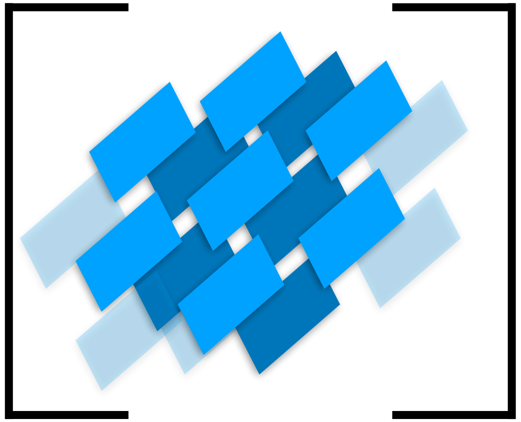


Bloom Filter: Repeated Trials

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

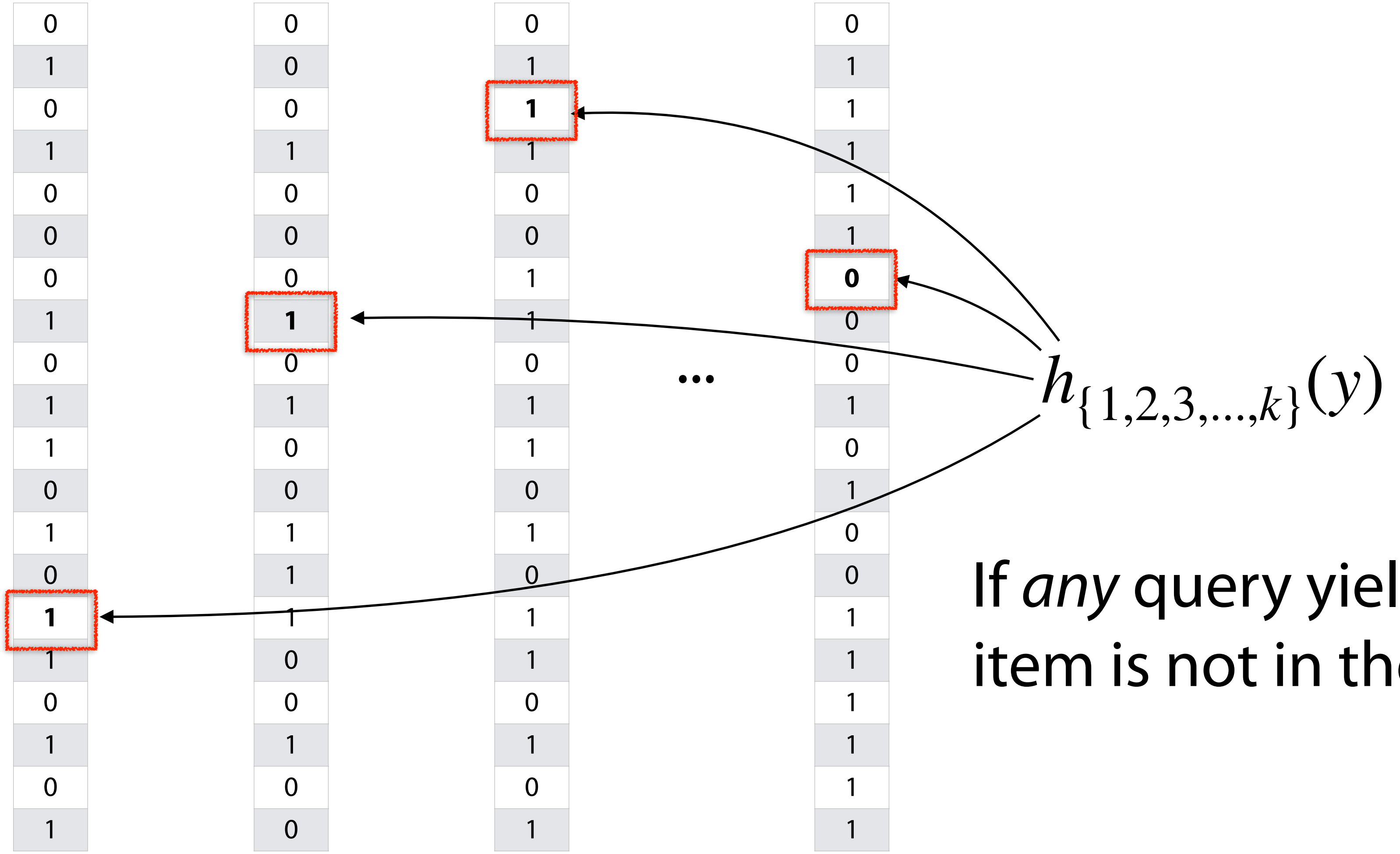
k different hashes

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

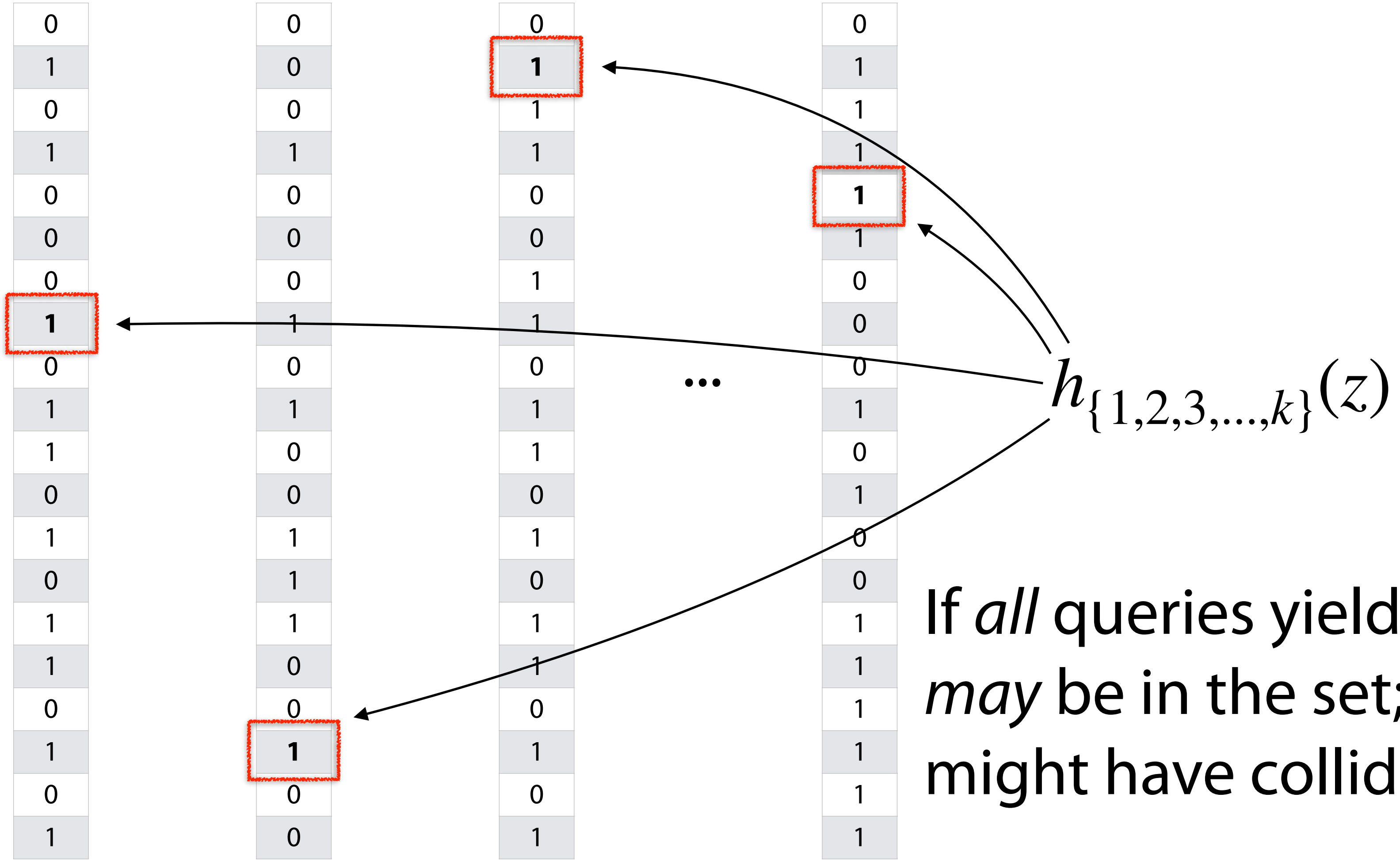
$$h_{\{1,2,3,\ldots,k\}}(y)$$

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

* Same k's as BF FPR

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?

$FPR = \frac{1}{2}$ $\quad k=10$

$$\left(\frac{1}{2}\right)^k \qquad \left(\frac{1}{2}\right)^{10} = 0.000976$$

Power of repeated trials ↔ One-sided error

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

New BF

different k

10 trillion hash functions

picked k of them

↳ One Bloom filter

Bloom Filter: Repeated Trials

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

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

S = { 6, 8, 4 }

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

Insert all hashes always

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)` → Not present TN

1 2 8

`_find(16)` → Exists (was not in data set) FP

6 2 3

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: $O(k) \sim O(1)$

In real world possible

Delete is not possible (yet)!

↳ One sided error b/c we don't delete

0
0
1
0
0
1
0
1
0
0

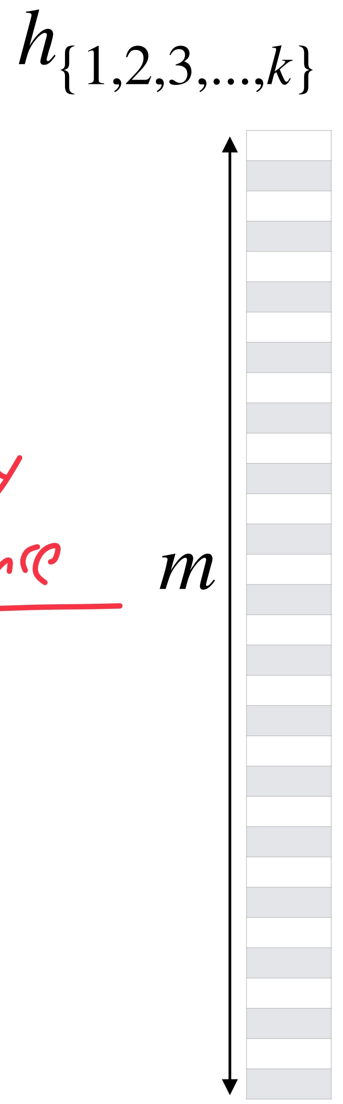
m

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?

↳ FP is when I pick k positions all 1 by chance



Bloom Filter: Error Rate

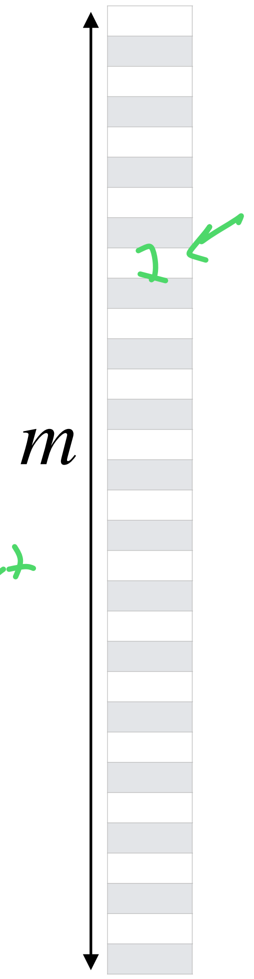
$h_{\{1,2,3,\ldots,k\}}$

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?

$$P(\text{bucket}=1) = \frac{1}{m}$$

After 1 insert prob changes from 1/m on next insert



Same probability given k SUHA hash function?

$$\left(\frac{1}{m}\right)^k \xrightarrow{\text{approaches}} 0$$

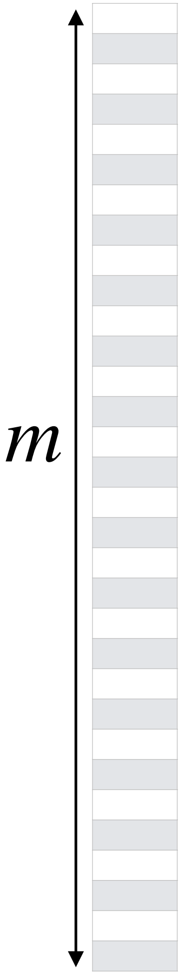
Not true!

This is hard to write!

Bloom Filter: Error Rate

$Pr(\text{Bucket} = 1) = 1 - Pr(\text{Bucket} = 0)$

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



0/1

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

4 (Rel)

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

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

Insert n items

k hashes each time

After n objects are inserted?

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

likelihood that nk indep uniform trials all missed this space

This takes into account decreasing likelihood as insert nk times

Bloom Filter: Error Rate

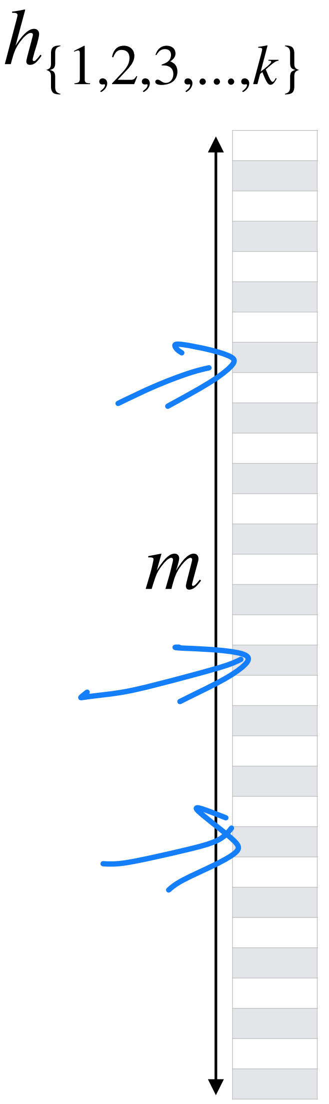
k buckets b/c
k random trials

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?

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

k



Bloom Filter: Error Rate

$P = \left(\frac{1}{2}\right)^k$ $k \to$ small

Given bit vector of size m and k SUHA hash function

FPR | truth is false

What is our expected FPR after n objects are inserted?

$\left(\frac{1}{2}\right) \equiv \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$. $\left(\frac{1}{4}\right) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

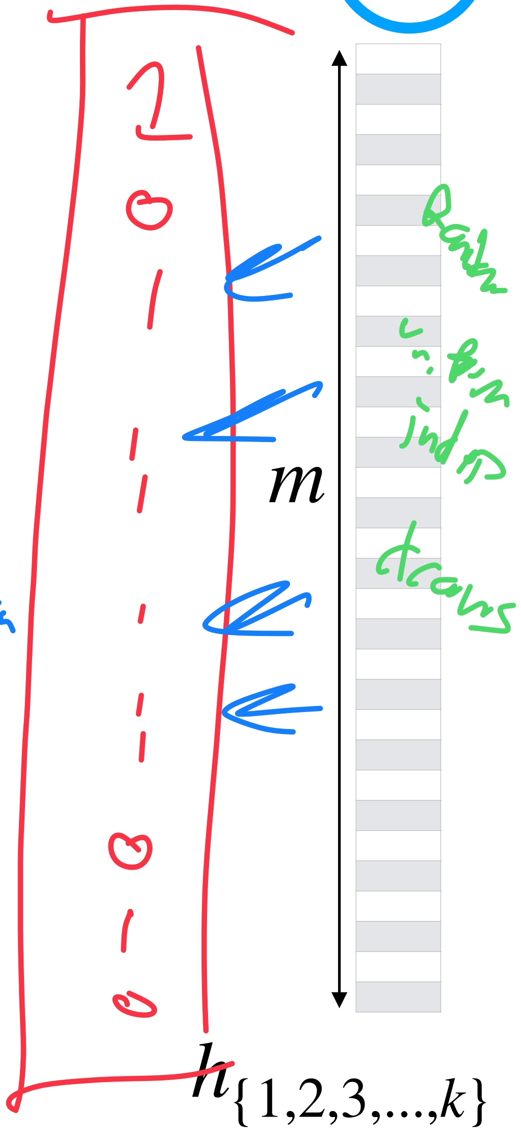
after n inserts

The probability my bit is 1 after n objects inserted

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

The number of [assumed independent] trials

P^K



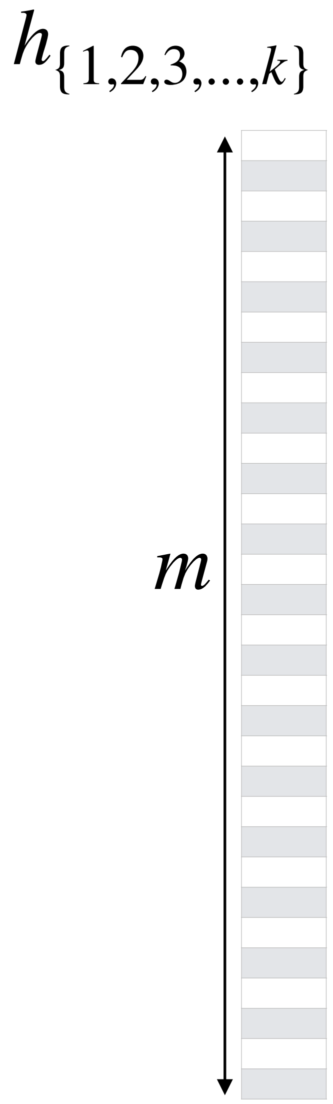
Bloom Filter: Error Rate

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

To minimize the FPR, do we prefer…

(A) large k **(B) small** k Both right and wrong!

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



Bloom Filter: Error Rate

$\frac{1}{2}$ $\left(\frac{1}{2}\right)^{10}$ ← much better

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

random trials

(A) large k

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

(B) small k

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

As k increases, this gets smaller!

As k decreases, this gets smaller!

P = fraction of bits in BF larger

↳ Want random trials

Too many hashes / inserts & filter saturates

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

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

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

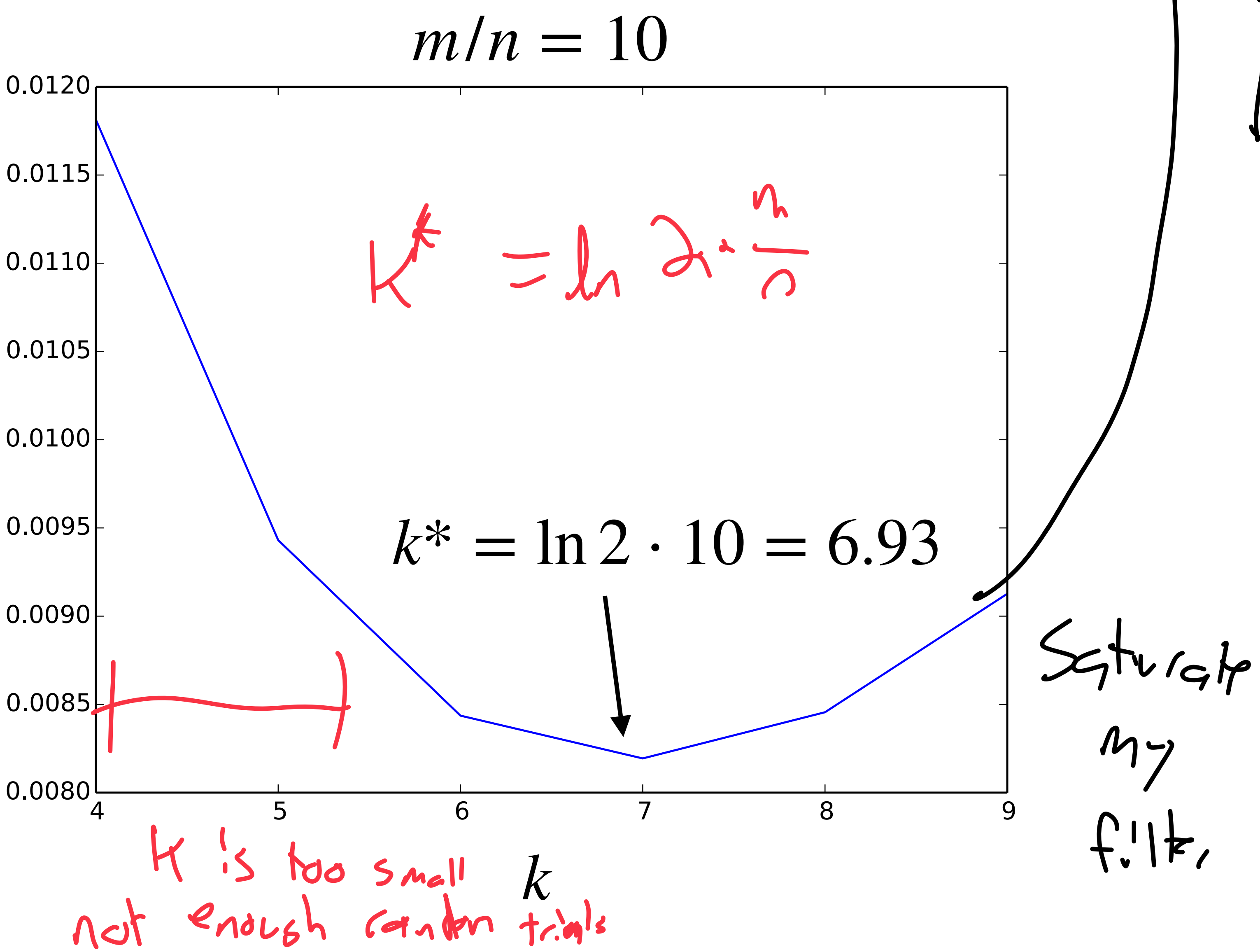


Figure by Ben Langmead

Bloom Filter: Optimal Parameters → Lab_Bloom

☺

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

Given any two values, we can optimize the third

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

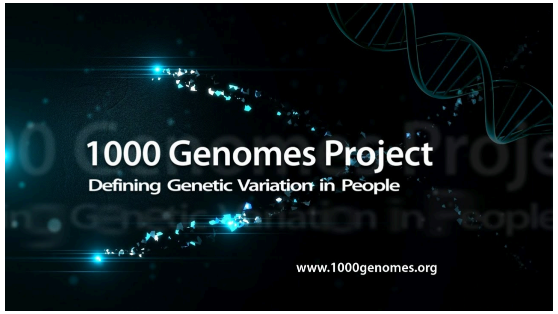
$m = 100$ bits $\quad n = 20$ items $\quad k =$

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

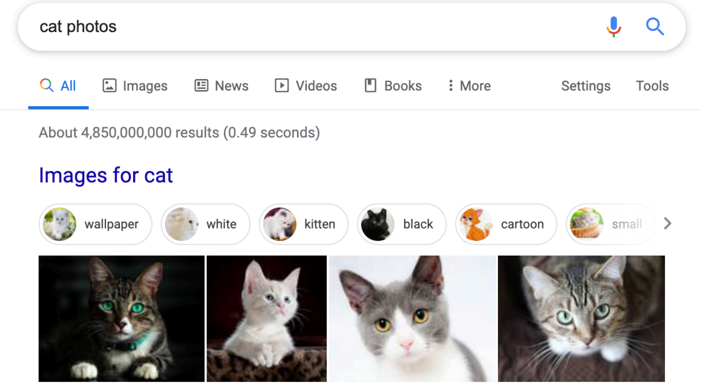
Bloom Filter: Optimal Parameters

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

Optimal hash function is still O(m)!

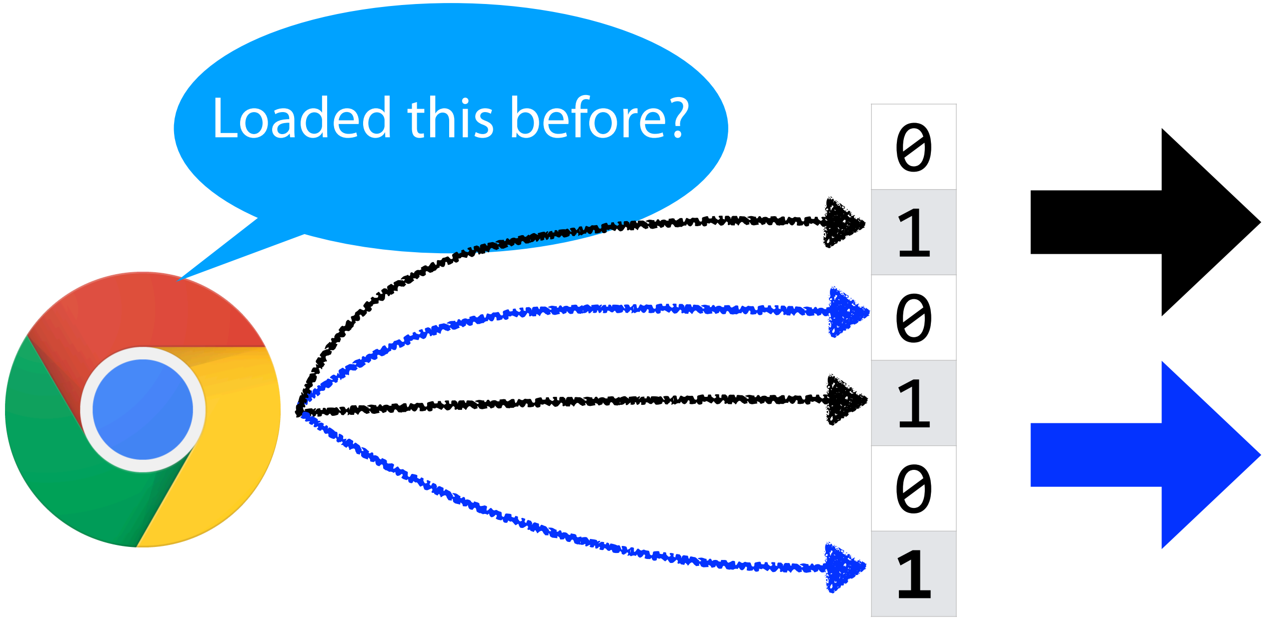


n = 250,000 files vs ~10^{15} nucleotides vs 260 TB



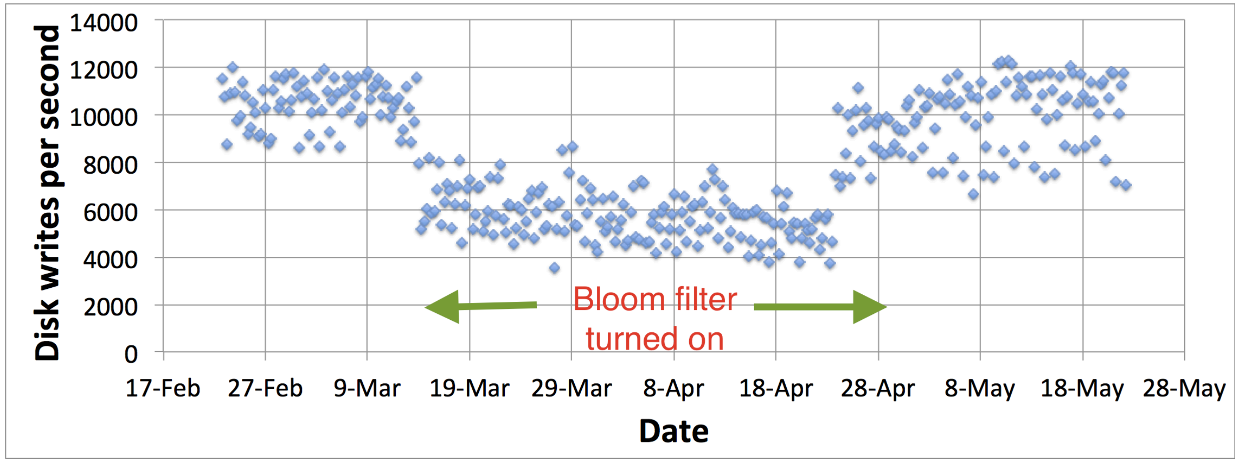
n = 60 billion — 130 trillion

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: Vector<bool> doesn't do this but actual bits do!

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	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 Vectors: Intersection

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

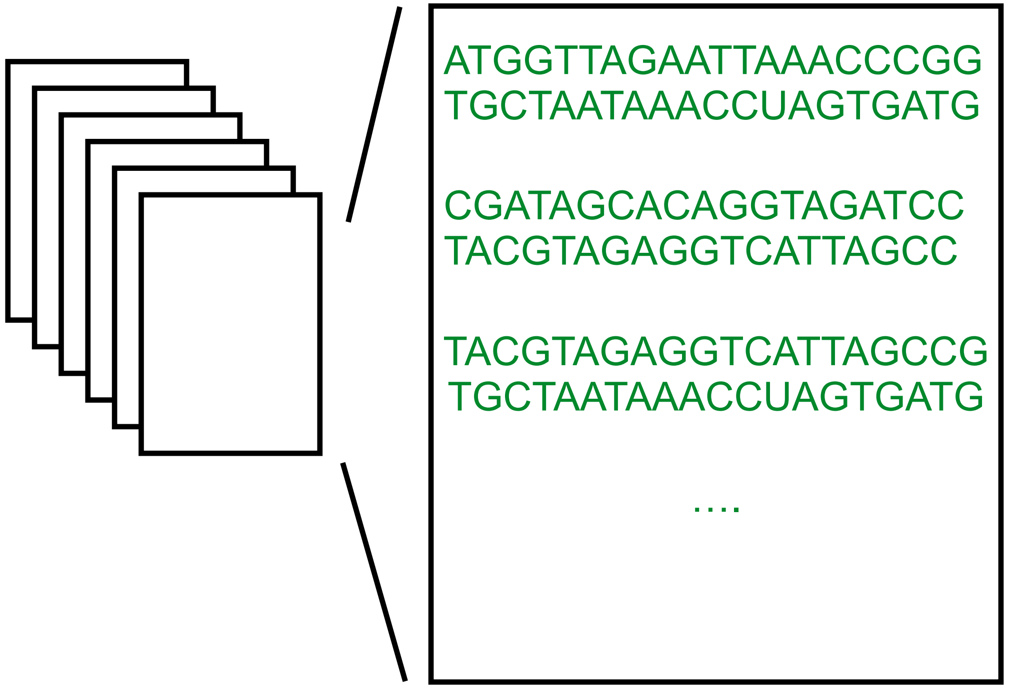
0	1		0	0		0	
1	0		1	1		1	
2	1		2	1		2	
3	1		3	0		3	
4	0	∪	4	0	=	4	
5	0		5	0		5	
6	1		6	1		6	
7	0		7	1		7	
8	0		8	1		8	
9	1		9	1		9	

Bit Vector Merging

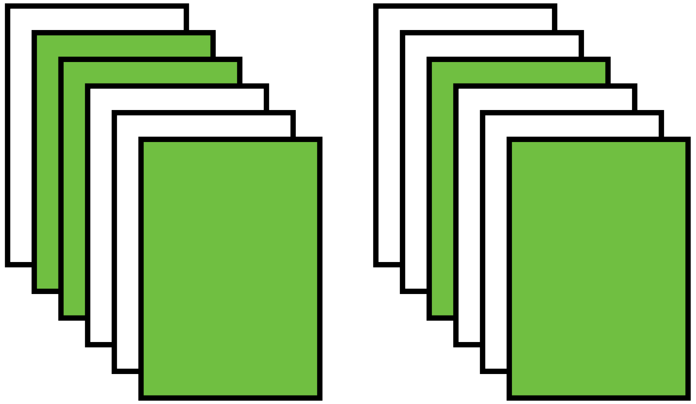
What is the conceptual meaning behind **union** and **intersection**?

Sequence Bloom Trees

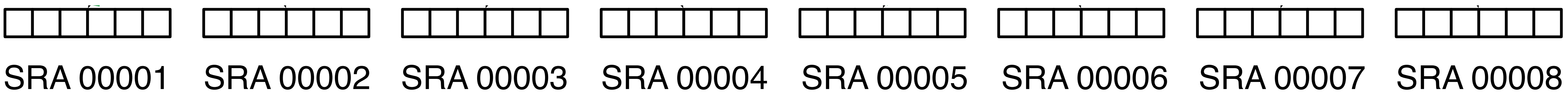
Imagine we have a large collection of text…



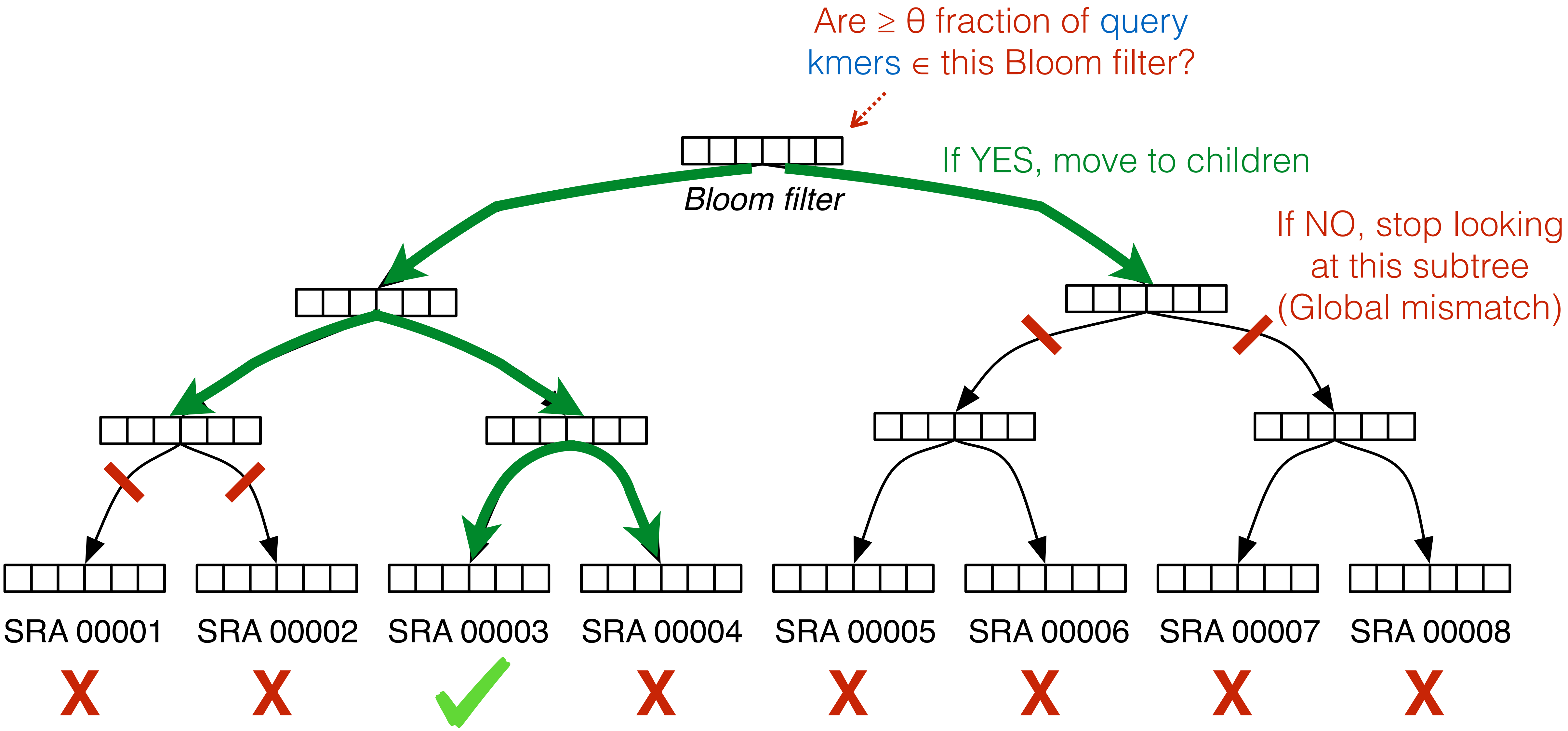
And our goal is to search these files for a query of interest…



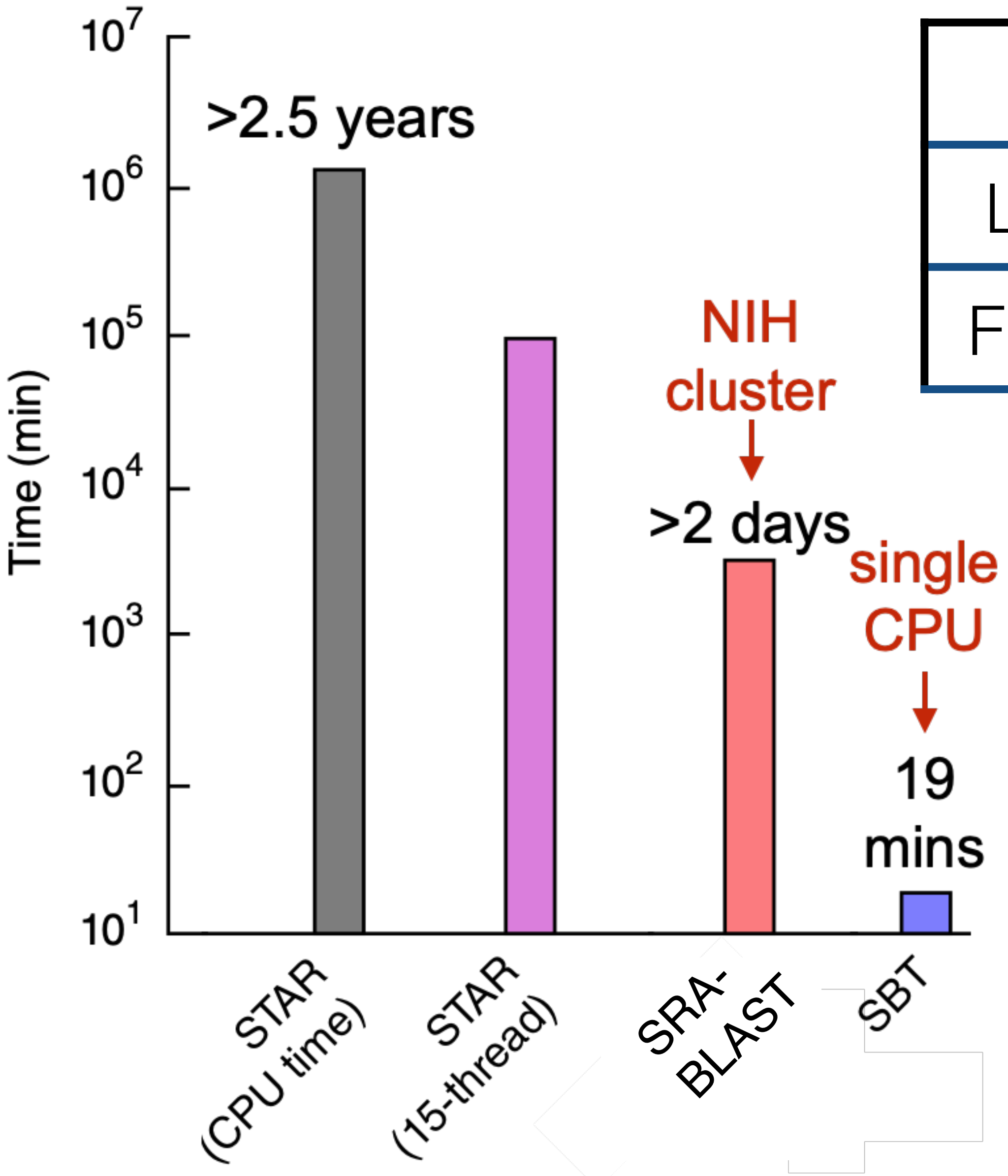
Sequence Bloom Trees



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…