Data Structures and Algorithms

Bloom Filters 2

(Mostly review plus analys.'s)

CS 225
Brad Solomon

November 20, 2024

UNIVERSITY OF ILLINOIS
URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Throwing out information

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*?
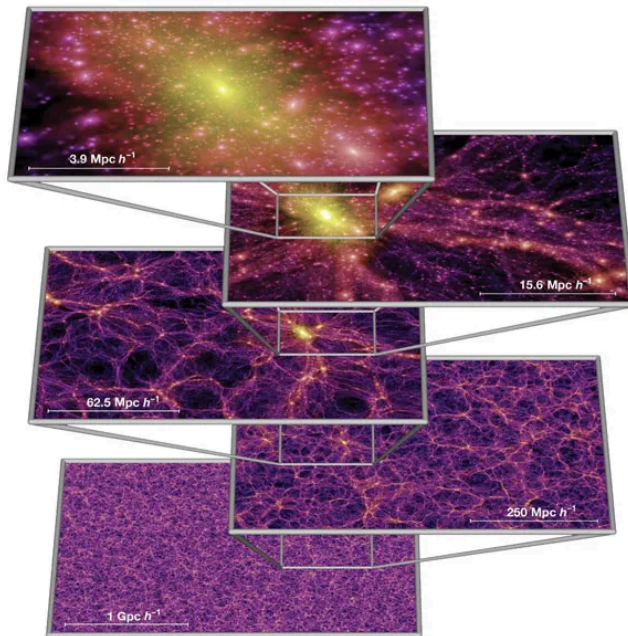
Constrained by Big Data (Large N)

| 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

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

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

cache	< 1 second
RAM	Hours - Days
disk	Months
network	Years
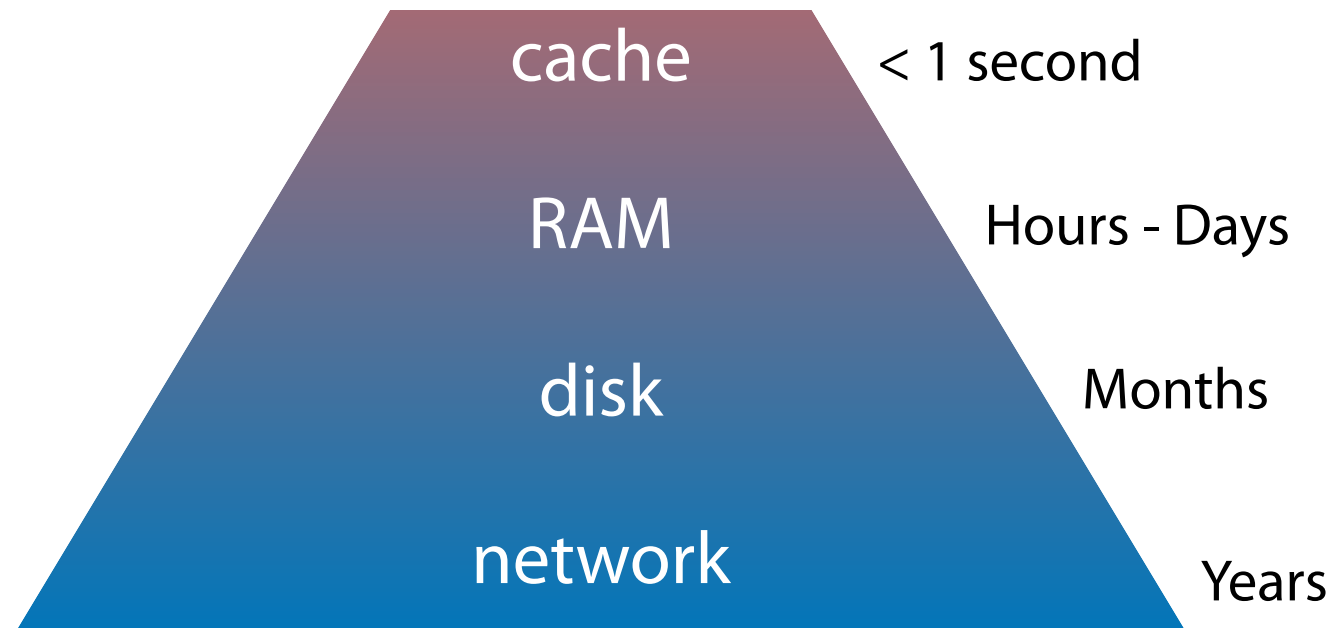
(Estimates are Time x 1 billion courtesy of https://gist.github.com/hellerbarde/2843375)
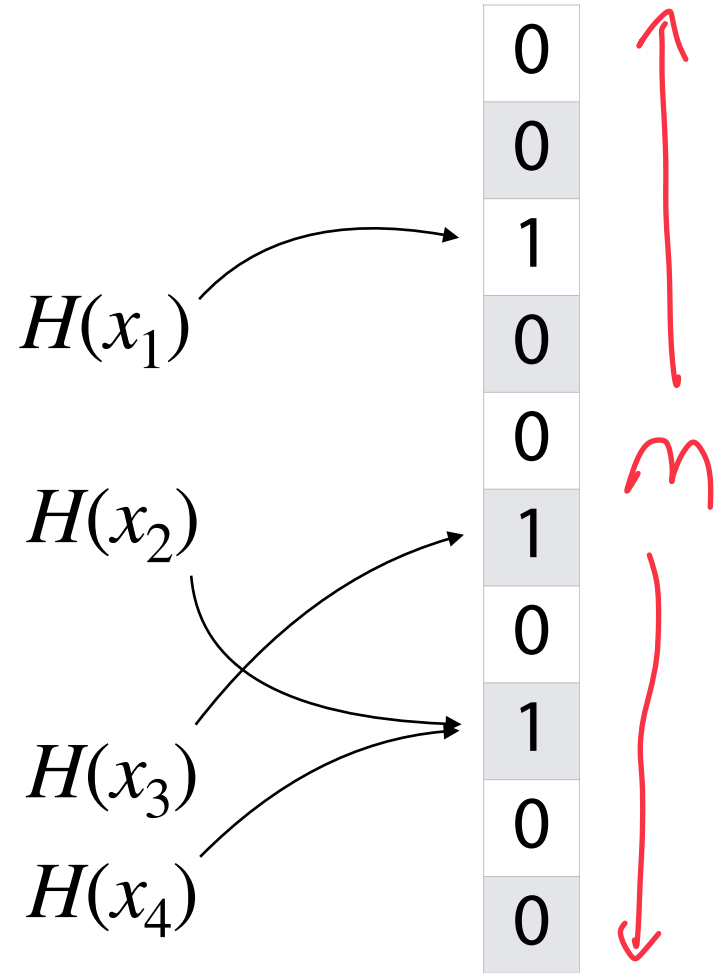
Bloom Filter: Insertion

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

$O(1)$

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

   If the bit was already one, it stays 1

$H(x_1)$

$H(x_2)$

$H(x_3)$
$H(x_4)$

0
0
1
0
0
1
0
1
0
0

Bloom Filter: Deletion

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

$H(x_1)$

$H(x_2)$

$H(x_3)$

$H(x_4)$

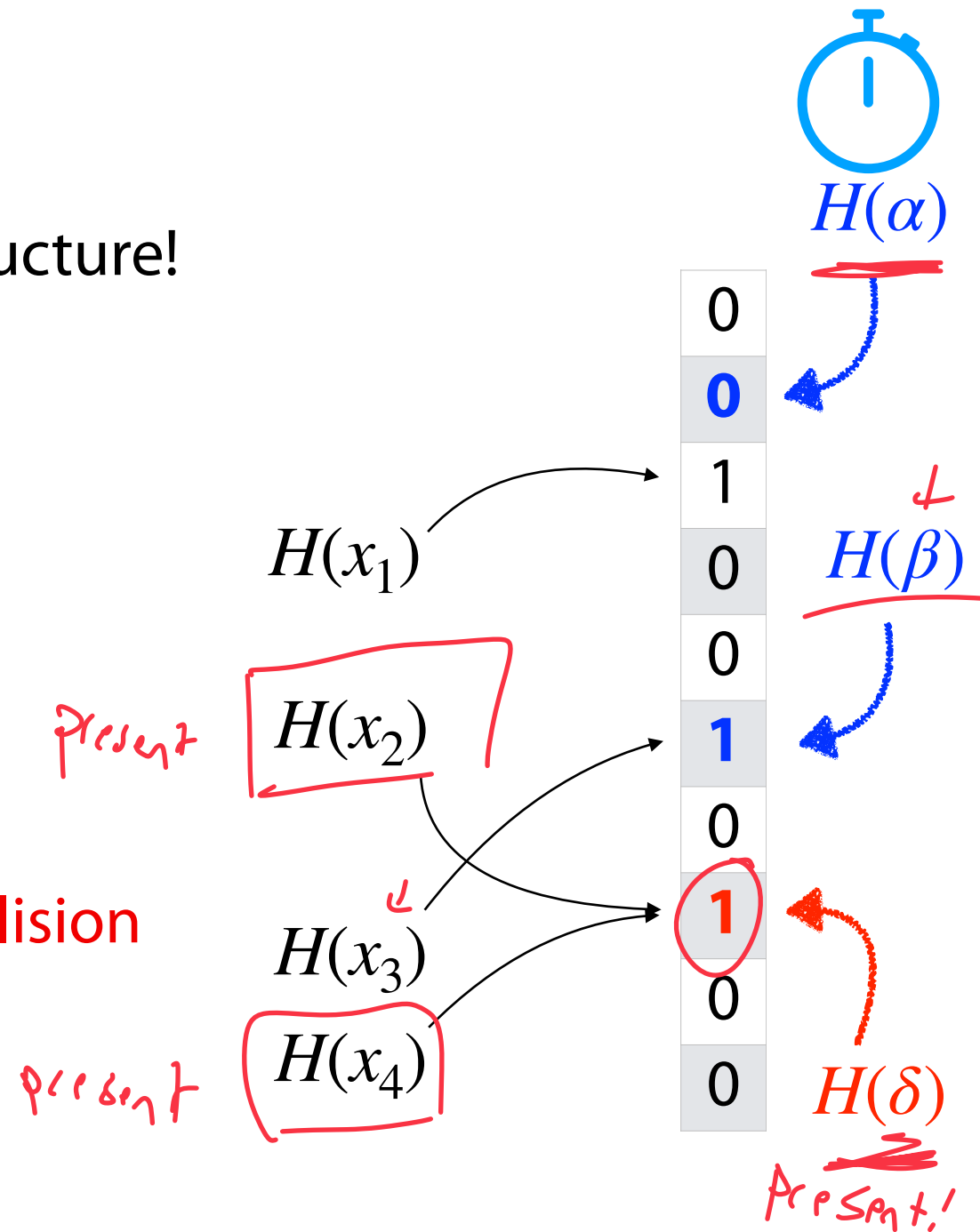| |
|---|
| 0 |
| 0 |
| **0** |
| 0 |
| 0 |
| 1 |
| 0 |
| **0** |
| 0 |
| 0 |
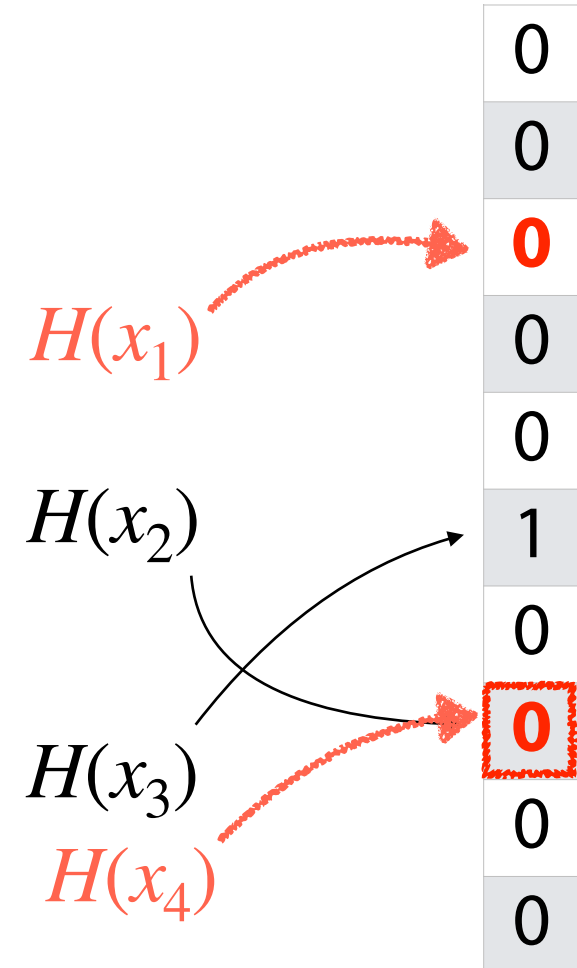
Bloom Filter: Search

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:

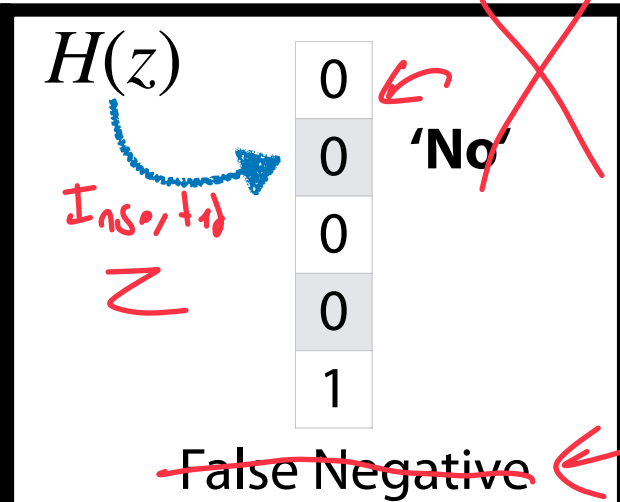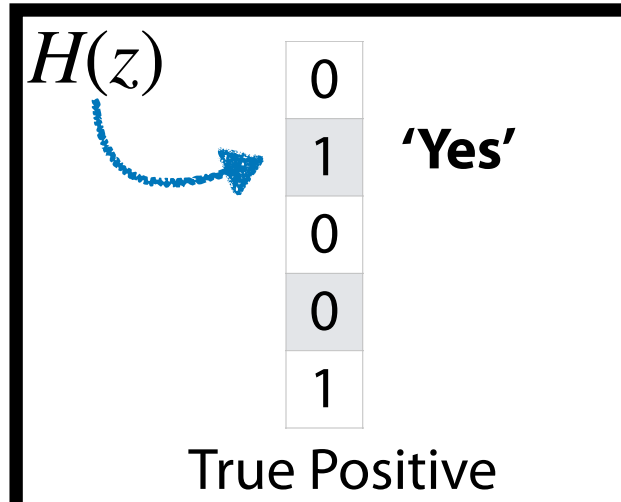It **may** be present or it may be a hash collision

$H(\alpha)$

$H(\beta)$

$H(x_1)$

$H(x_2)$ Present

$H(x_3)$

$H(x_4)$ present

$H(\delta)$ Present!

0
0
1
0
0
1
0
1
0
0

Probabilistic Accuracy in a Bloom Filter

Probabilistic Accuracy: One-sided error

Query:

50% accurate

Dataset:
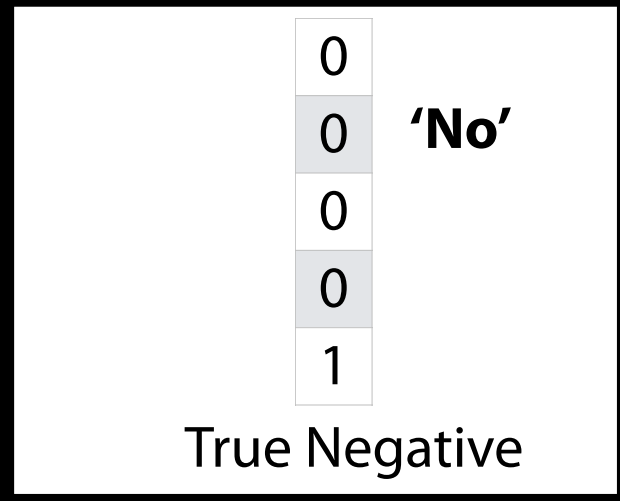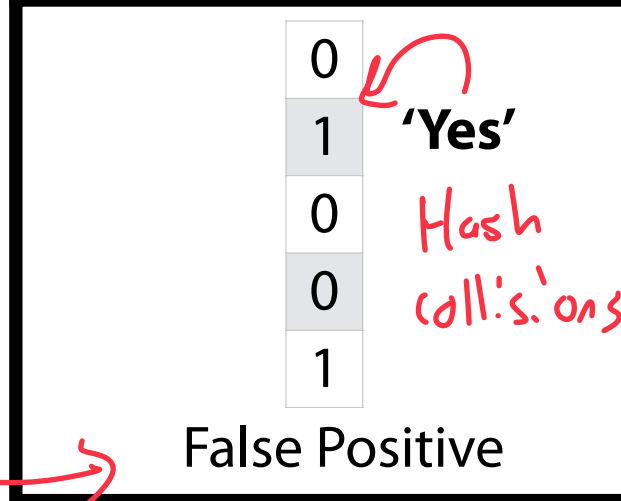
search with one-sided error
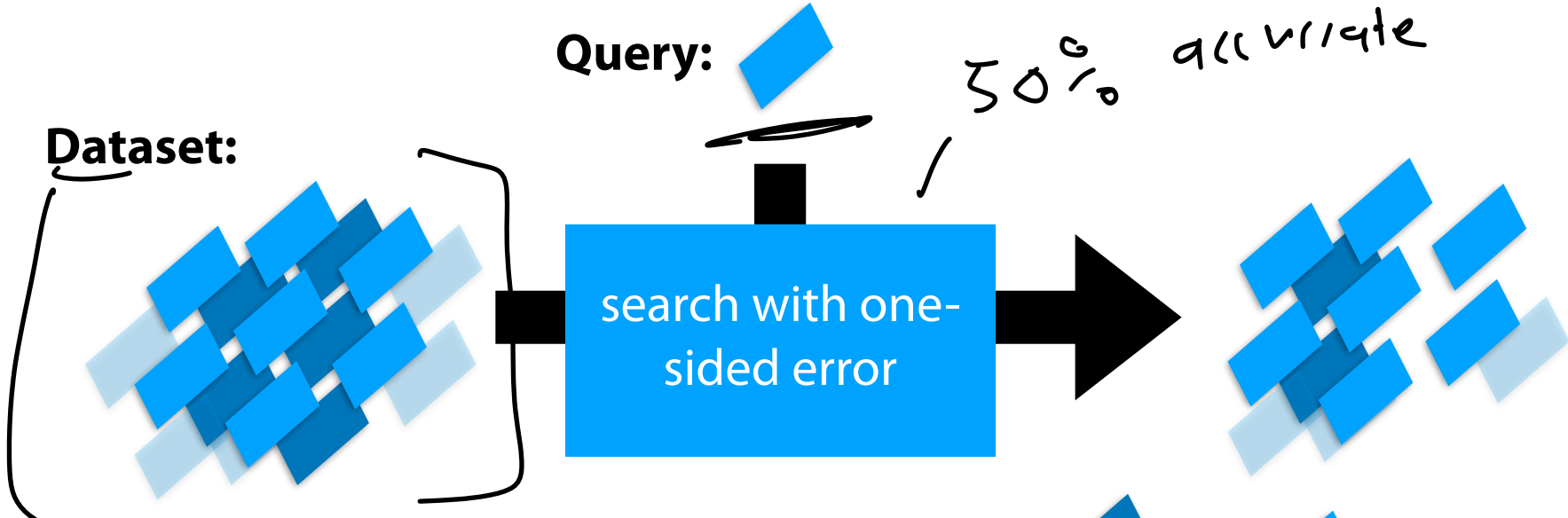
We will get some False Positives:

↳ Sometimes incorrect passes
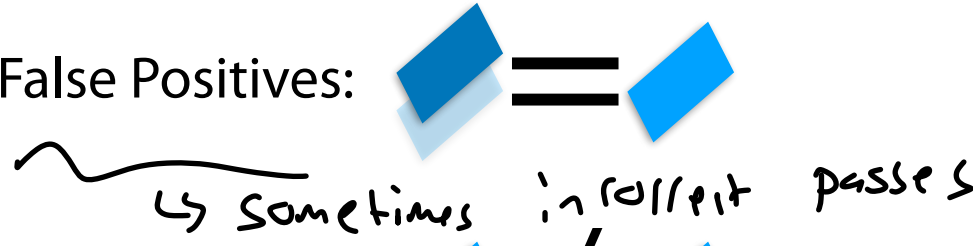
We will NEVER have a False Negative:

↳ Never treat correct as incorrect

Probabilistic Accuracy: One-sided error

Query:

Dataset:

~50%

Output from round 1

search with one-sided error

Input in round 2

Query:

~50%

search with 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

Bad → 0

O

Filtering out bad
(Didn't collide)

h_1

Bad →

Not filtered

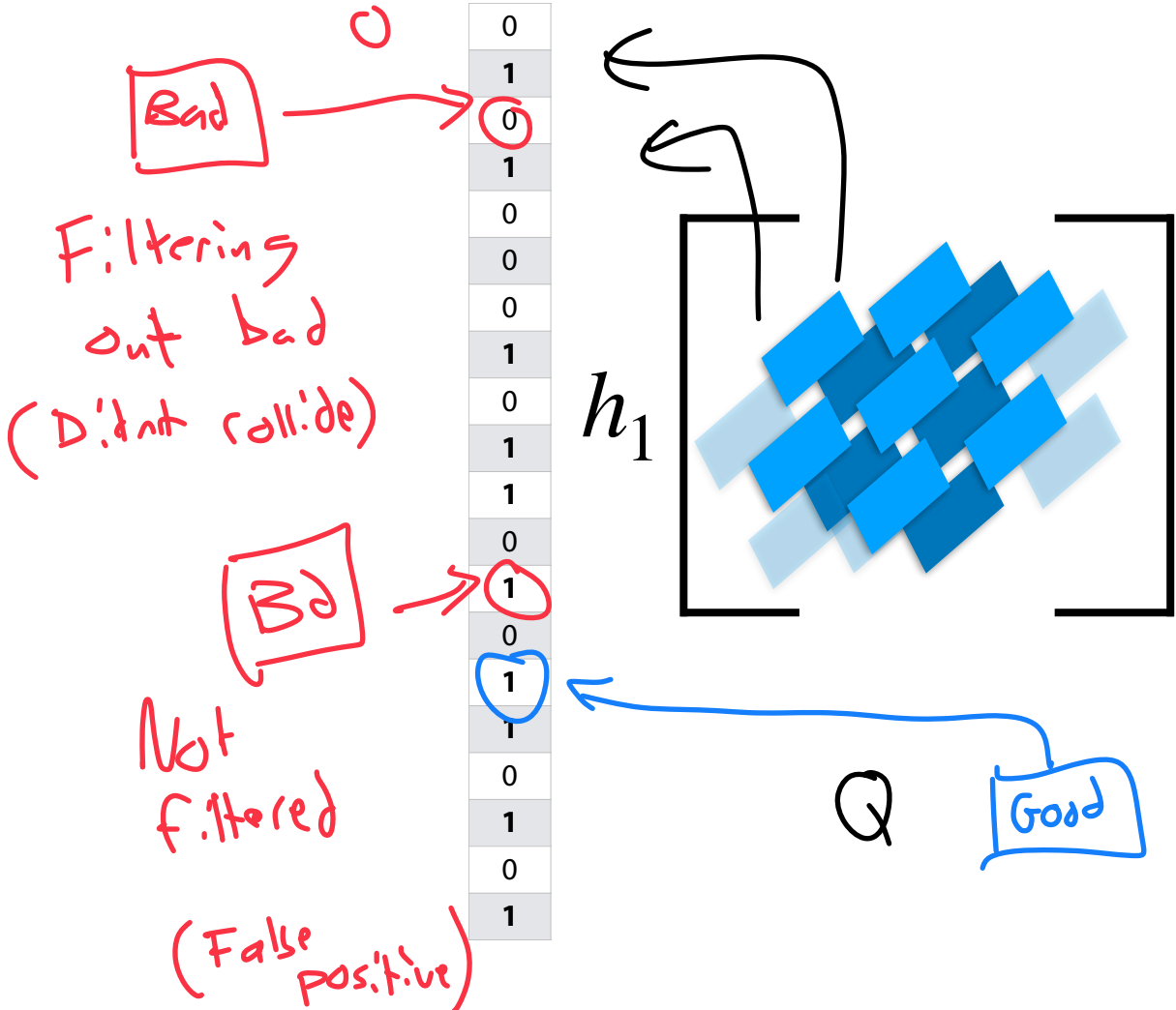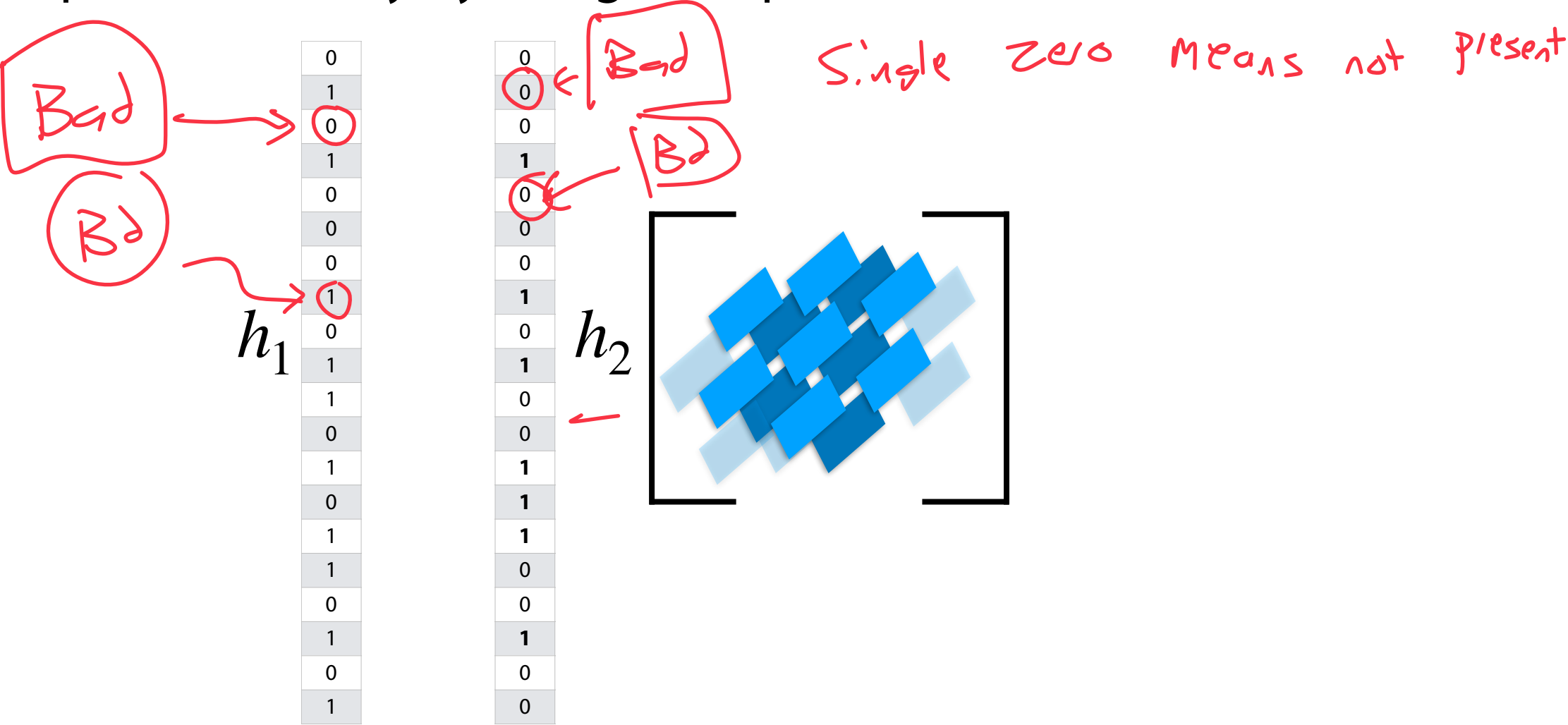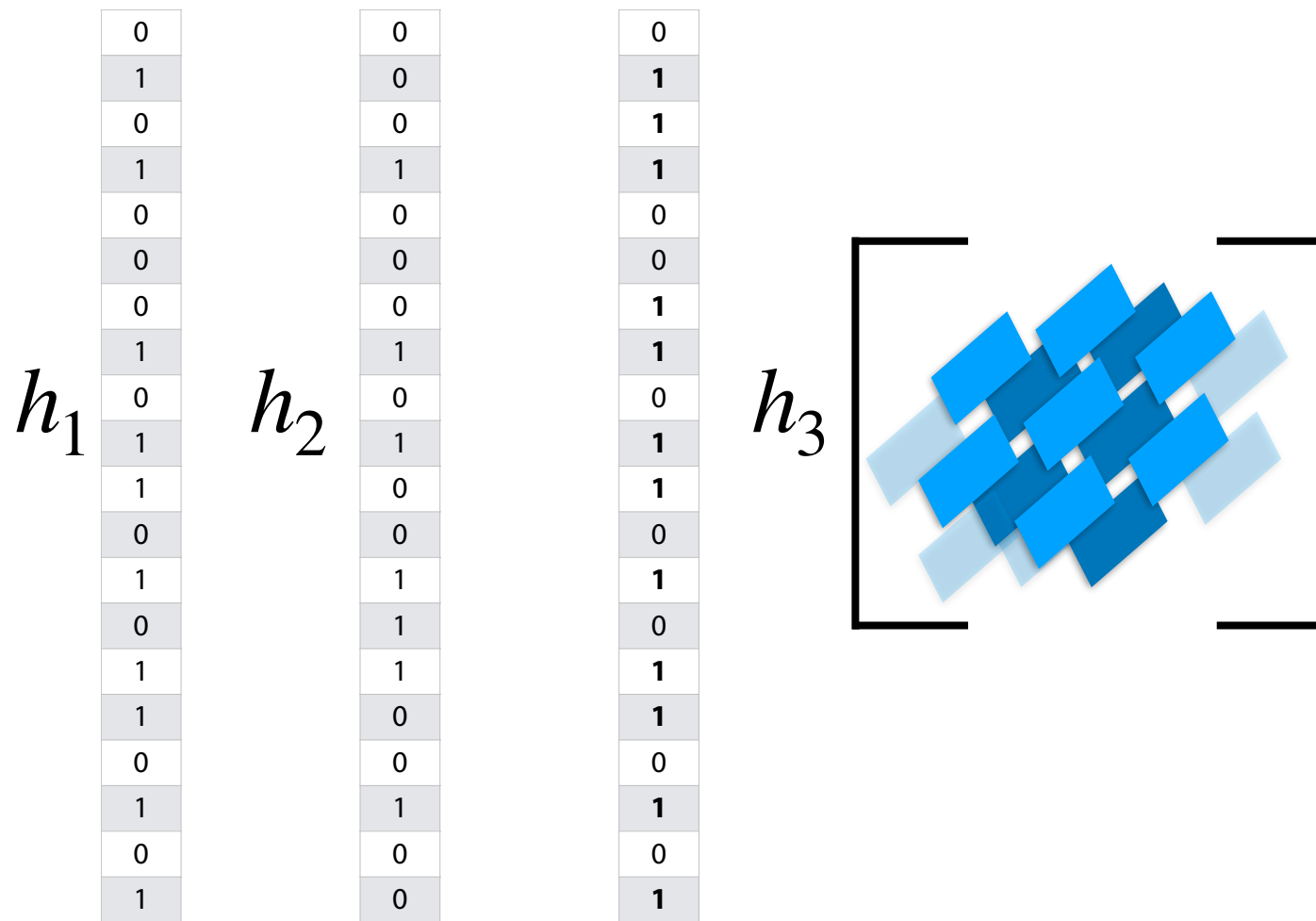(False positive)

Q   Good

Bloom Filter: Repeated Trials

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

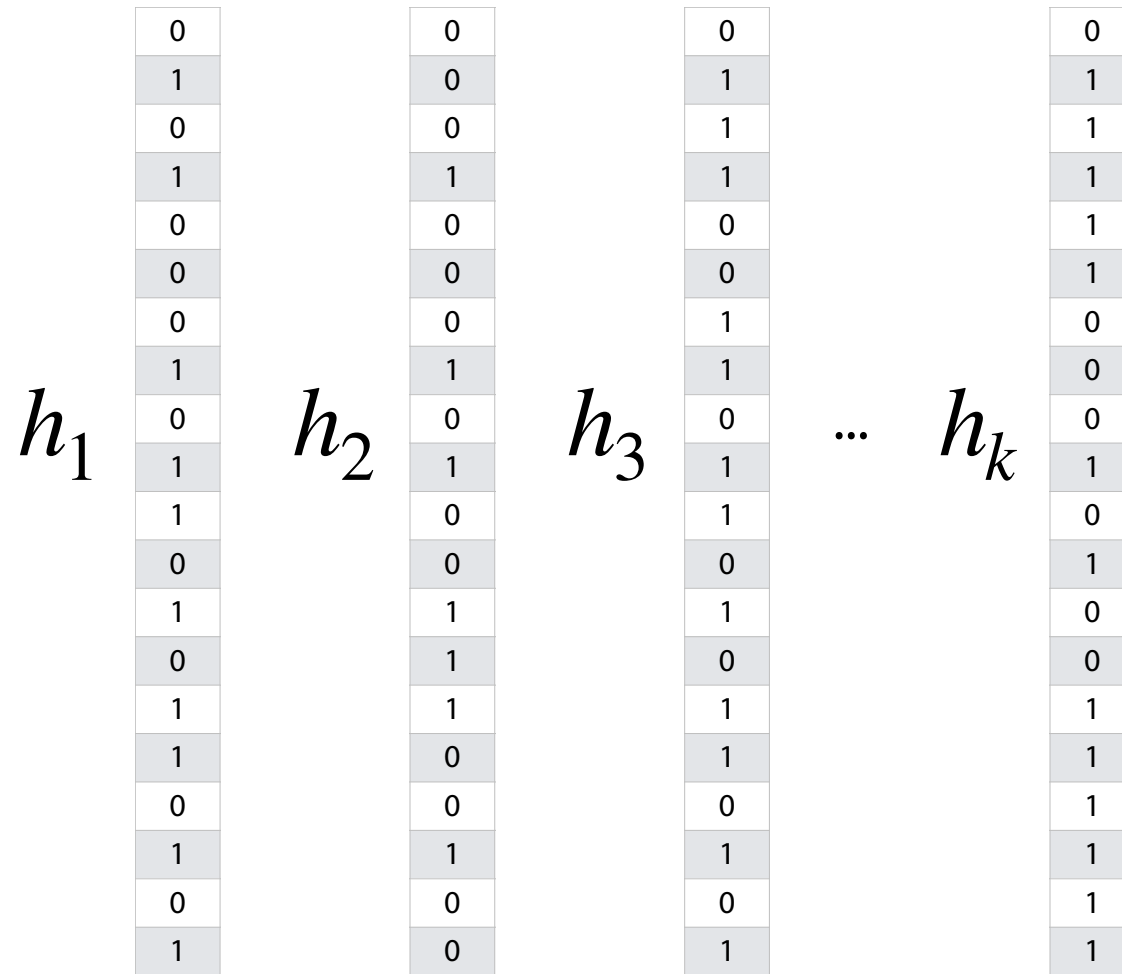# Bloom Filter: Repeated Trials

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

Bloom Filter: Repeated Trials

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

h_1 h_2 h_3 ... h_k

Bloom Filter: Repeated Trials

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

We insert y K times

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

Bloom Filter: Repeated Trials

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

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

Is **y** present?

Bloom Filter: Repeated Trials

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

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

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!

$h_{\{1,2,3,...,k\}}(z)$

Is **z** present?

Bloom Filter: Repeated Trials

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

$h_{\{1,2,3,...,k\}}(z)$

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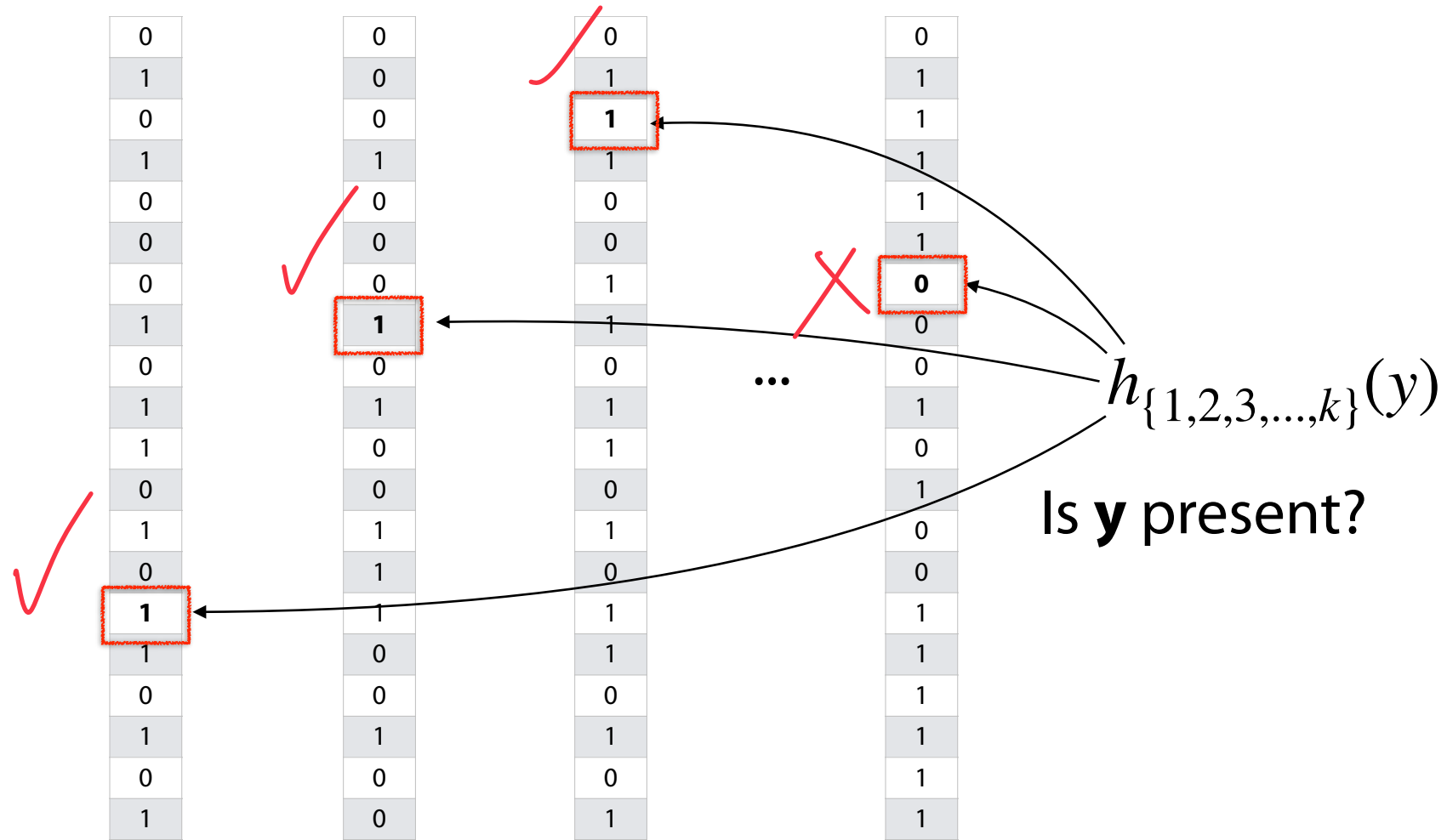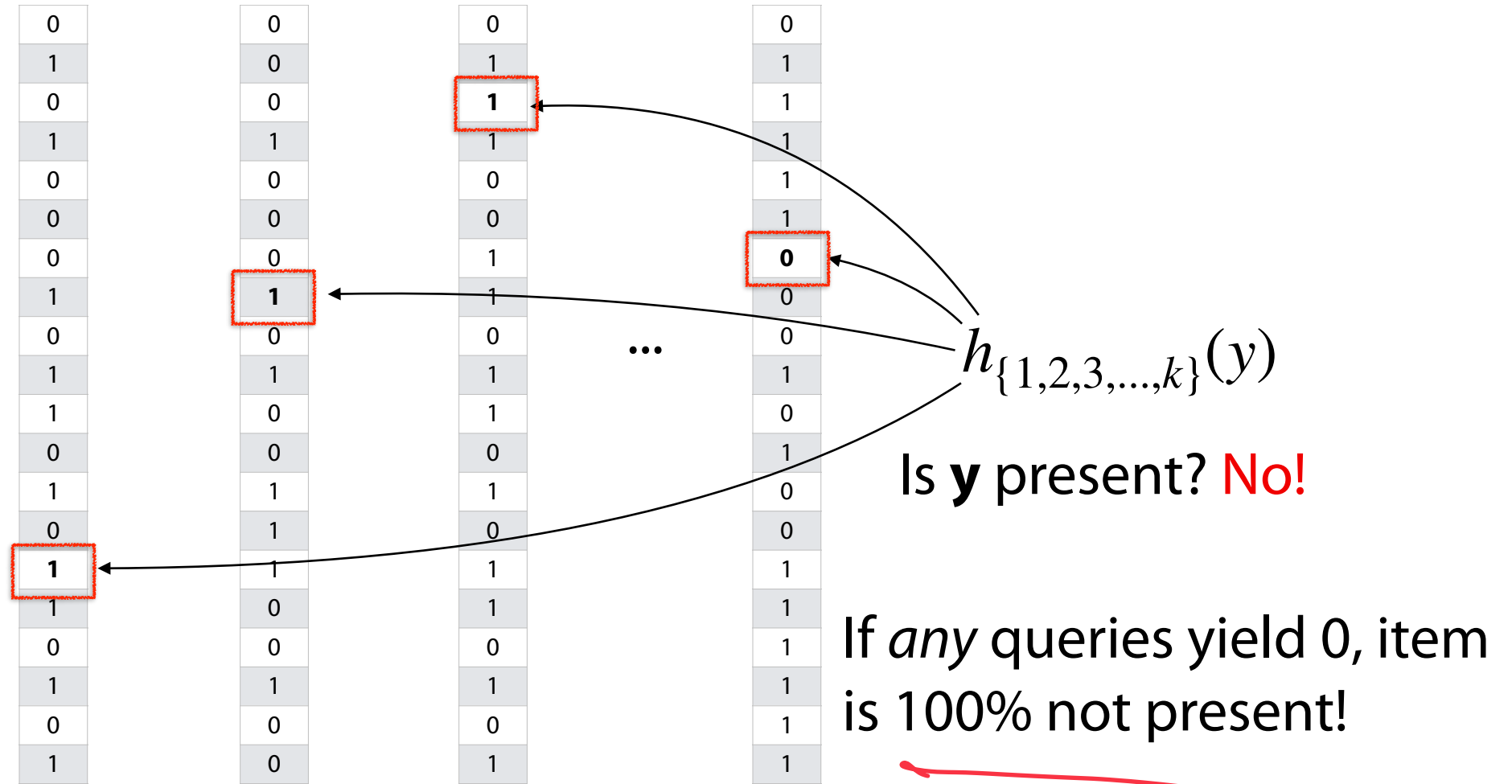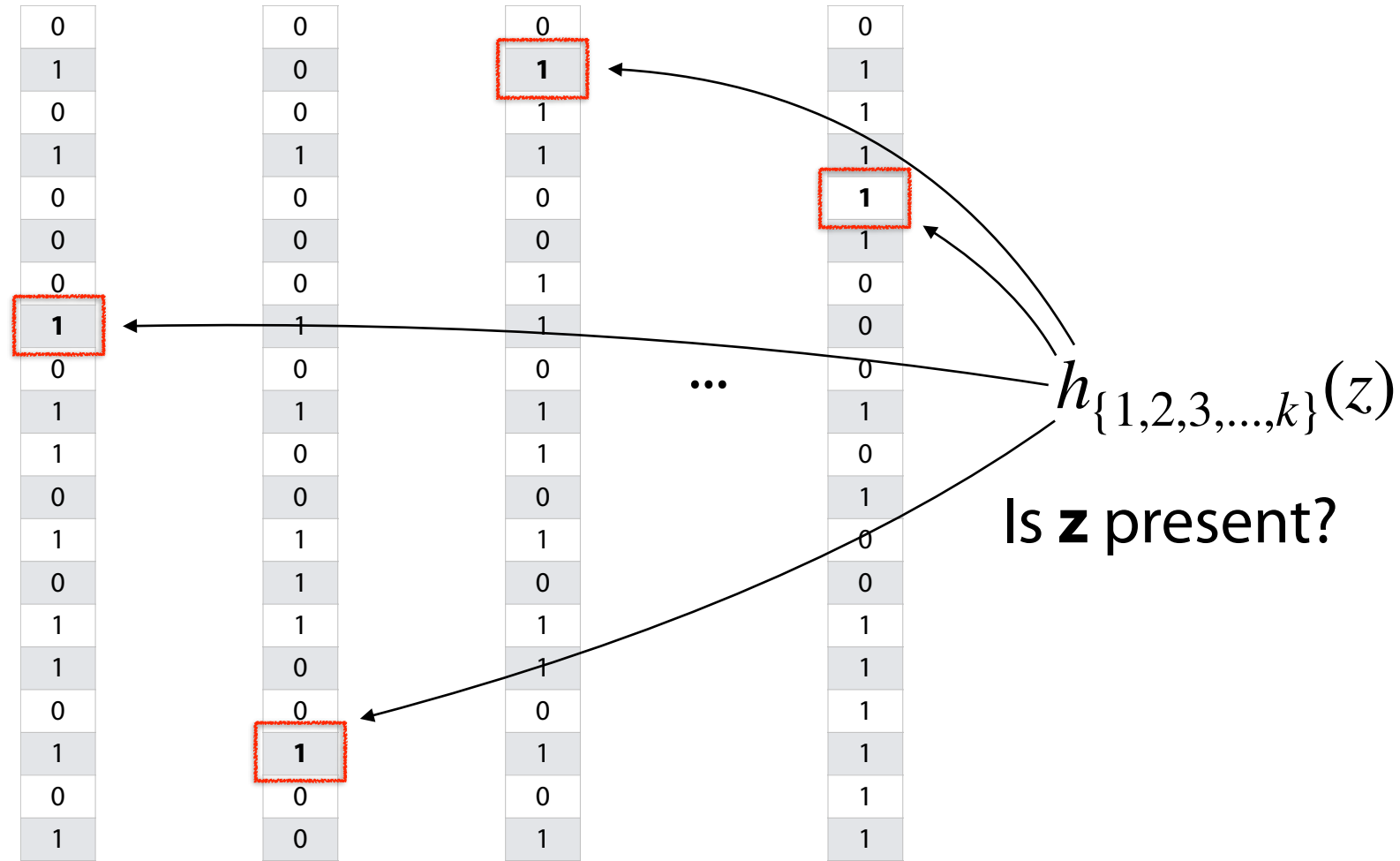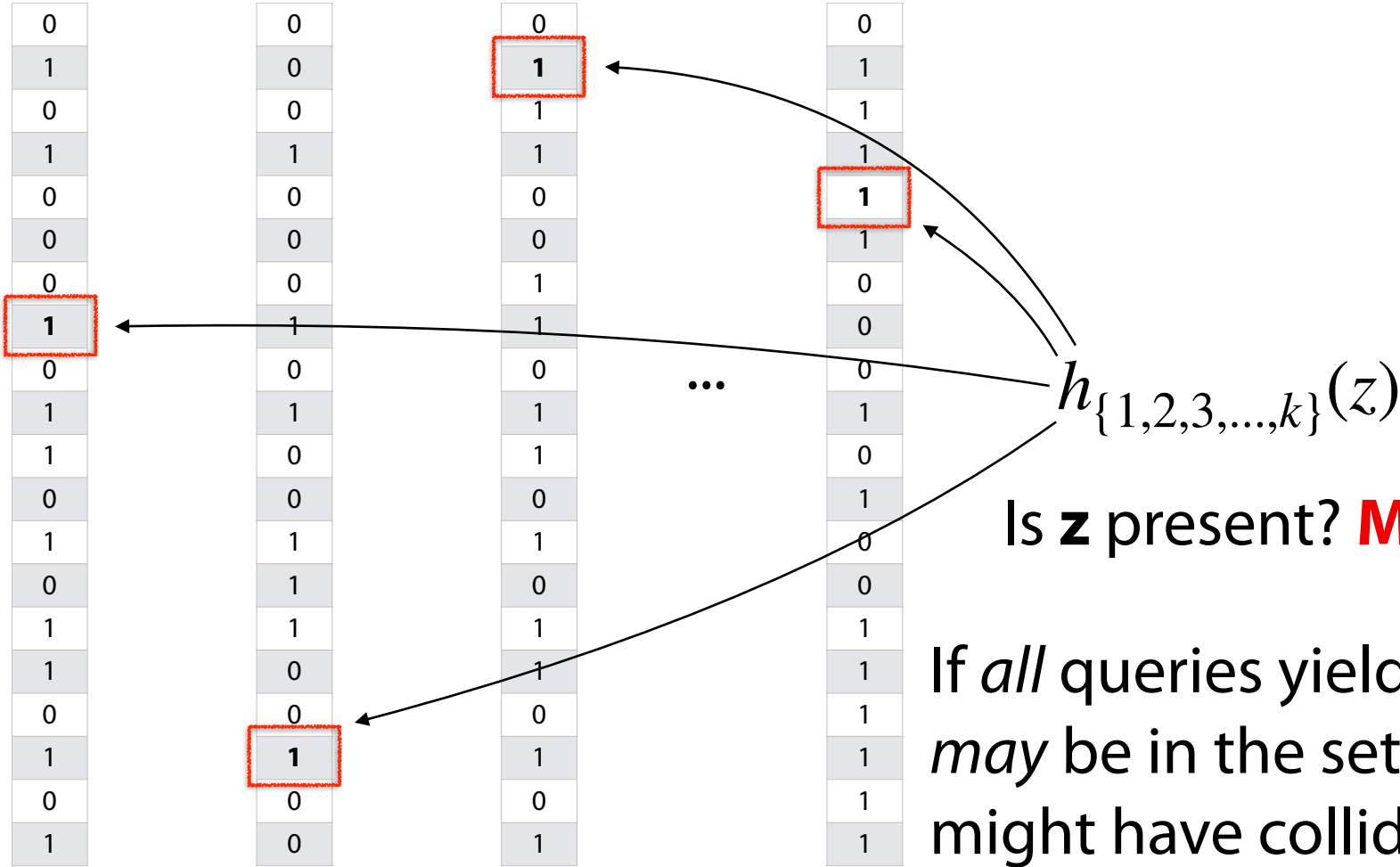
Is **z** present? **Maybe!**

If *all* queries yield 1, item *may* be in the set; or we might have collided k times

False Positive still Possible!

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?

$$(p)^k = (0.5)^k$$

$$\frac{1}{2^n}$$

Bloom Filter: Repeated Trials

But doesn't this hurt our storage costs by storing k separate filters?
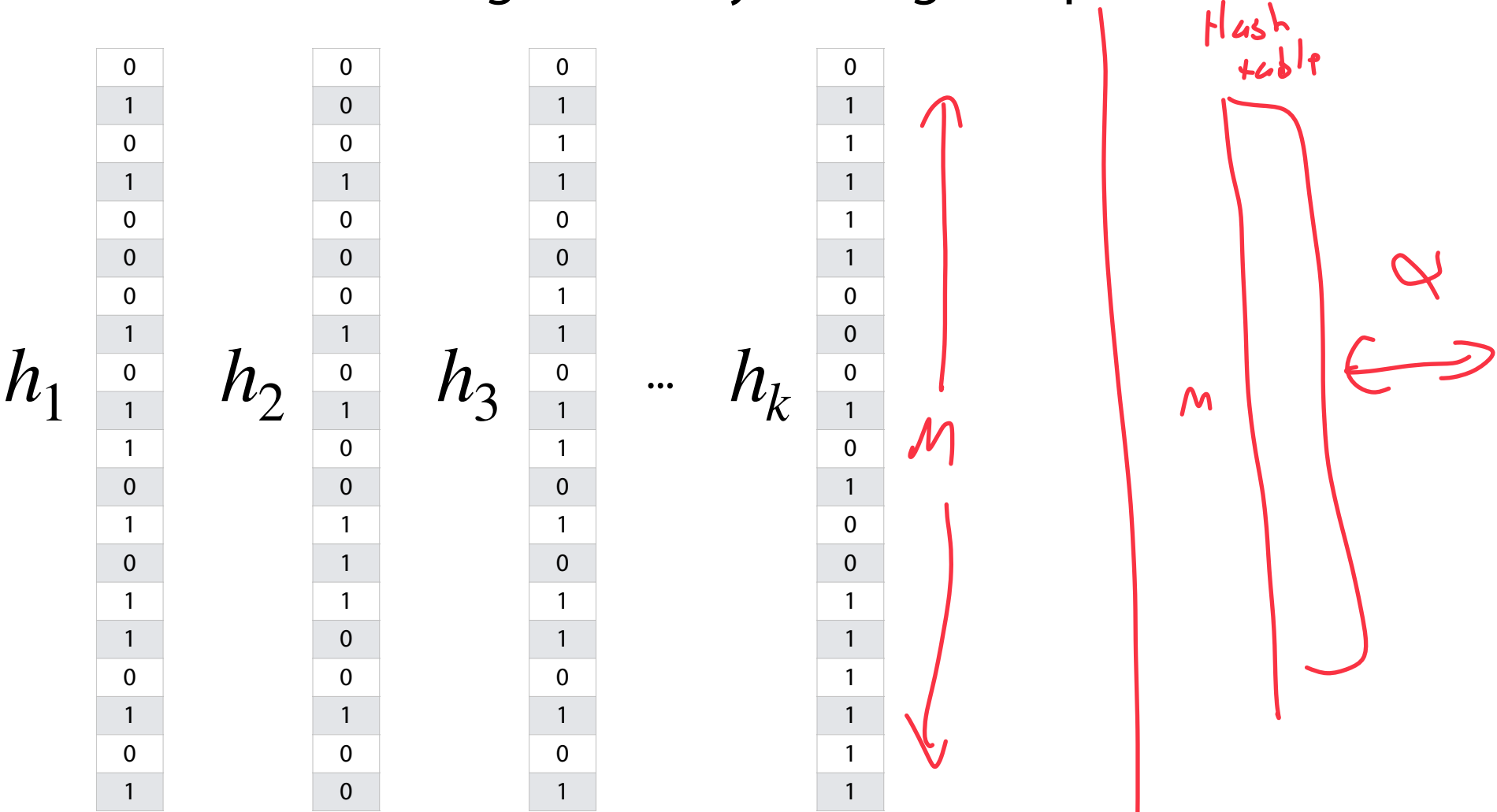
Bloom Filter: Repeated Trials

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

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

S = { 6, 8, 4 }

h₁(x) = x % 10

6
8
4

h₂(x) = 2x % 10

2
6
8

h₃(x) = (5+3x) % 10

3
9
7

New source of collision

Bloom Filter: Repeated Trials

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

M

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) → False b/c M[1] = 0

1   2   8

_find(16) → True! (False pos still possible!)

6   2   3
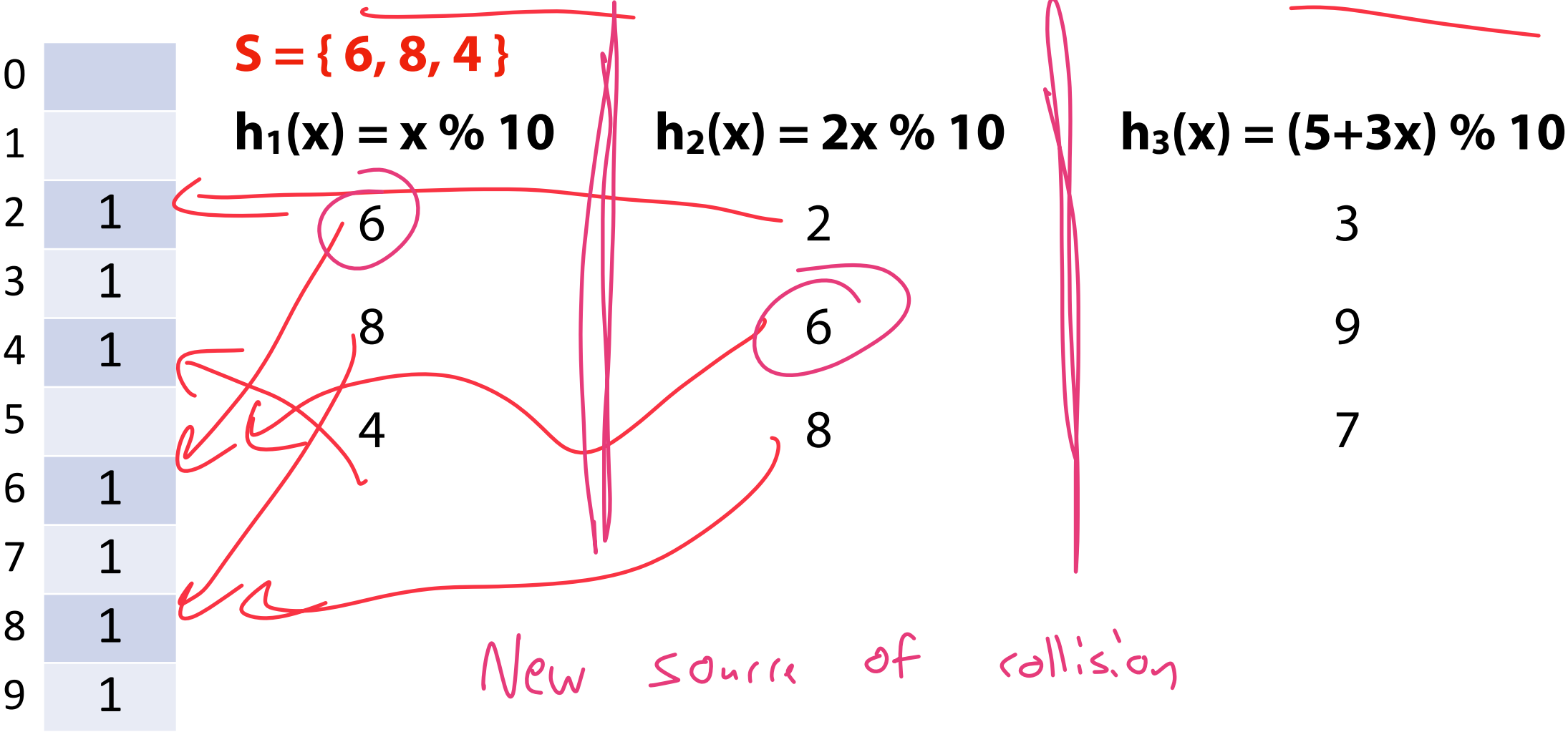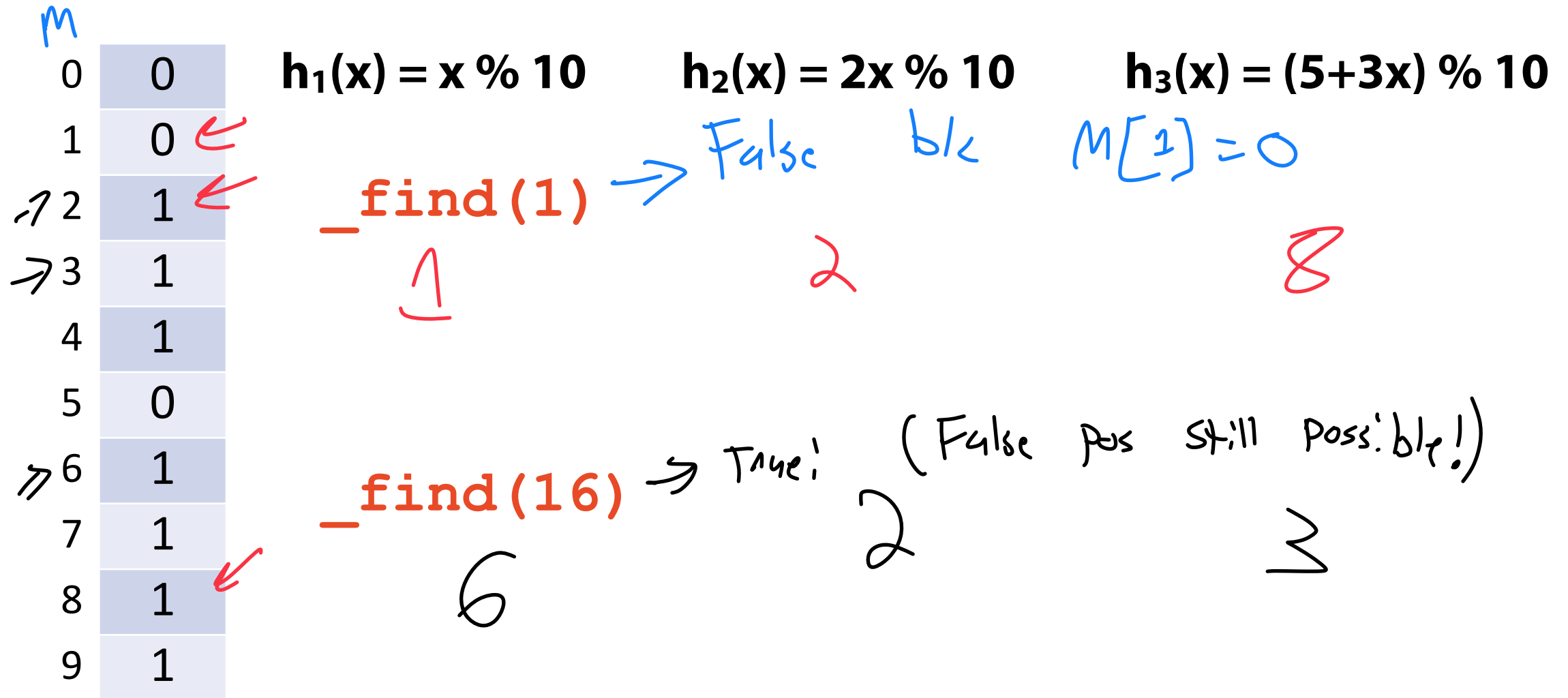
Bloom Filter

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: $O(k)$ / $O(1)$

Delete is not possible (yet)!

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

0
0
1
0
0
1
0
1
0
0

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?

$\hookrightarrow (\text{Prob a bit is } 1)^{k}$

How do we get
prob bit $= 1$?

$H_2 \rightarrow 3/4$

$H_2 \rightarrow 3/4$

$H_3 \ldots$

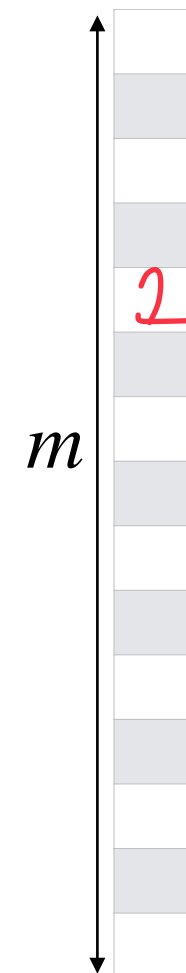|	
1	0 ←
7	1 ←
2	2 ←
0	3

m

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?

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

2

m

Same probability given k SUHA hash function?

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

→ This prob approaches 0 as k increases

Not correct!
↳ This is hard to write

Bloom Filter: Error Rate

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

Given bit vector of size m and 1 SUHA hash function

$\rightarrow \textcircled{k}$

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

$1 - \left(\dfrac{1}{m}\right)$ $\leftarrow$ After 1 insert

$\uparrow$
prob bucket is 1

After n objects are inserted? $\leftarrow$ Am I still 0? (B/c SUHA math is trivial)

likelihood that after n inserts my bucket is still 0

$$\left[\left(1 - \dfrac{1}{m}\right)^{nK}\right]$$

K hashes $\rightarrow$ nk SUHA inserts

n repeated trials
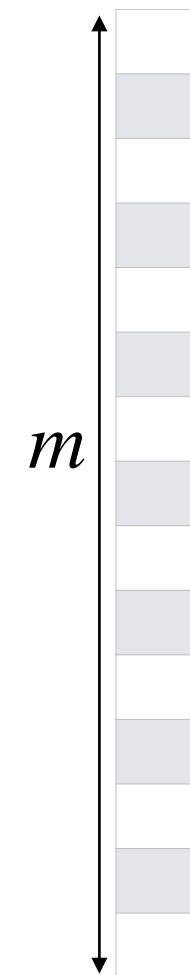
m

1

1
2
2

Bloom Filter: Error Rate

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

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?

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

prob bucket is 0

what is my FPR?

$$\left(\text{prob bucket is 1} \right)^{K}$$

$$m$$
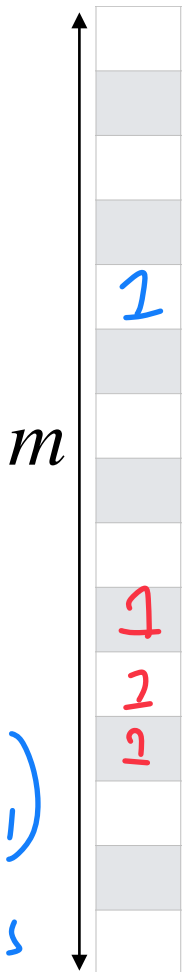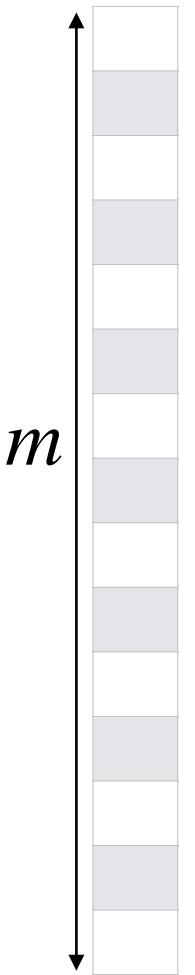
Bloom Filter: Error Rate

Given bit vector of size m and k SUHA hash function

What is our expected FPR after n objects are inserted?

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

m

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

Bloom Filter: Error Rate

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

Vector of size m, k SUHA hash function, and n objects
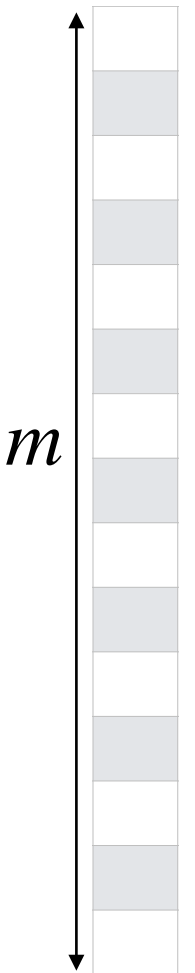
To minimize the FPR, do we prefer…

(A) large k   **(B) small k**

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

m

Both are right!

Bloom Filter: Error Rate

Vector of size m, k SUHA hash function, and n objects
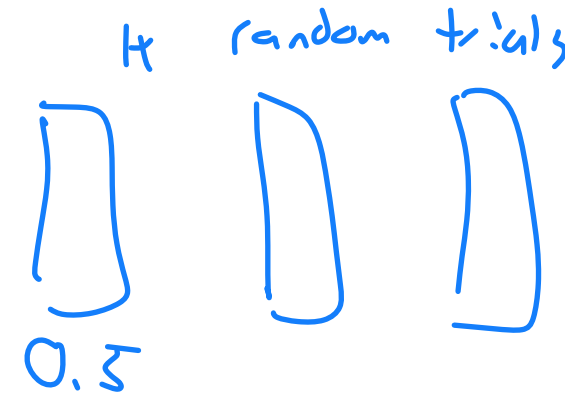
(A) large k

$$\left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k}$$
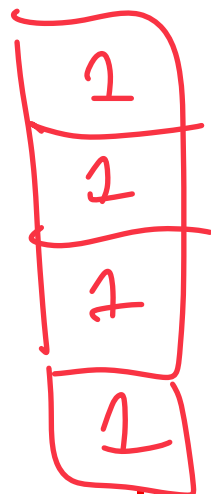
As k increases, this gets smaller!

Prob bit is Zero

↳ k too large Saturates filter

(B) **small** k

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

As k decreases, this gets smaller!

k random trials

0.5

$k = 1 \to FPR = 50\%$

2   23%

3   12.5
        ↓
        1

Bloom Filter: Optimal Error Rate

To build the optimal hash function, fix **m** and **n**!

Claim: The optimal hash function is when $k* = ln\ 2 \cdot \dfrac{m}{n}$

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

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

Bloom Filter: Optimal Error Rate

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

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

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

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

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

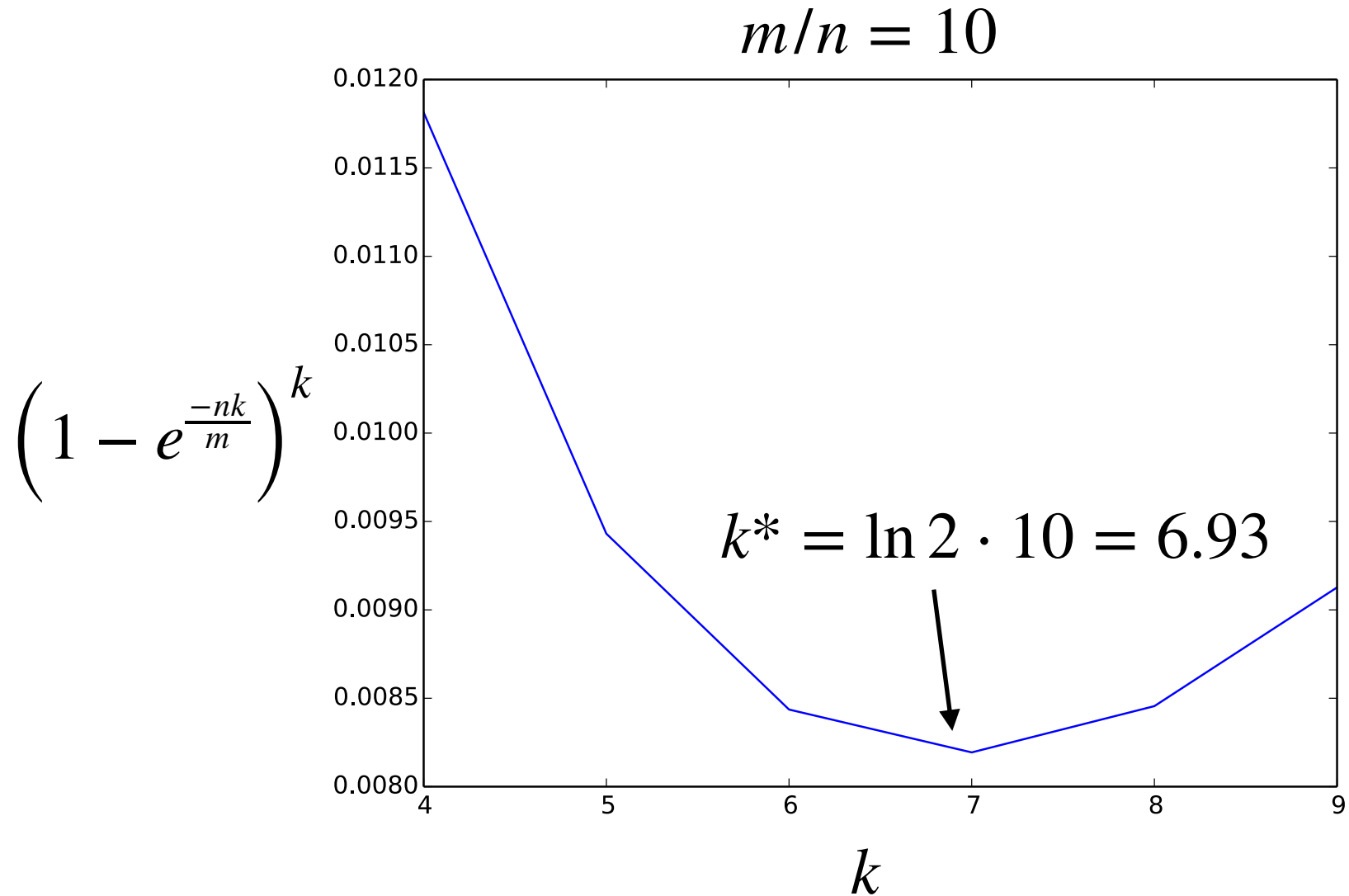$$\approx e^{\frac{-nk}{m}}$$

Bloom Filter: Optimal Error Rate

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

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

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

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

$m/n = 10$

$k* = \ln 2 \cdot 10 = 6.93$

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 $k = 3$ hashes $m =$

$m = 100$ bits $n = 20$ items $k =$
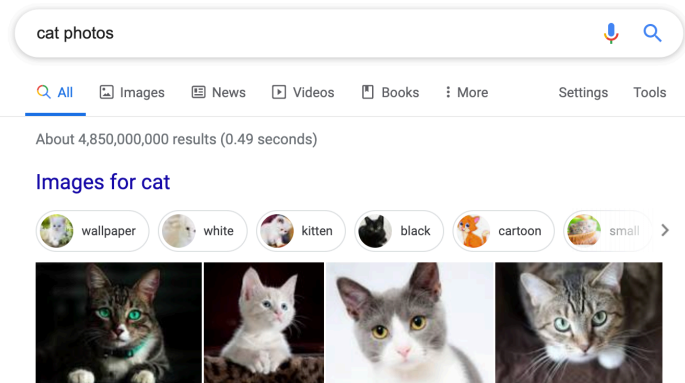
$m = 100$ bits $k = 2$ items $n =$

Bloom Filter: Optimal Parameters

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

Optimal hash function is still O(m)!

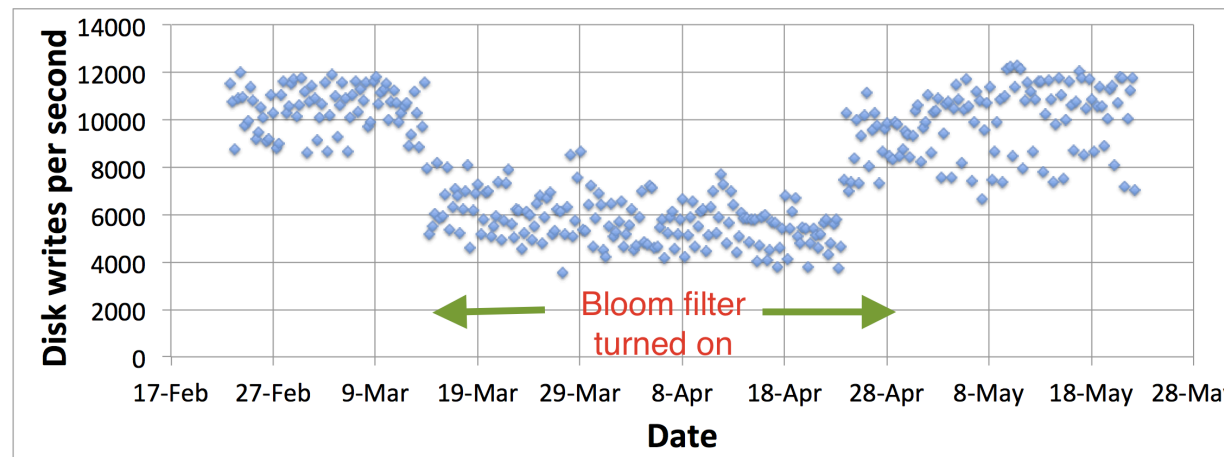
n = 250,000 files vs ~10^{15} nucleotides vs 260 TB

n = 60 billion — 130 trillion

Bloom Filter: Website Caching

Loaded this before?

0
1
0
1
0
1

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.
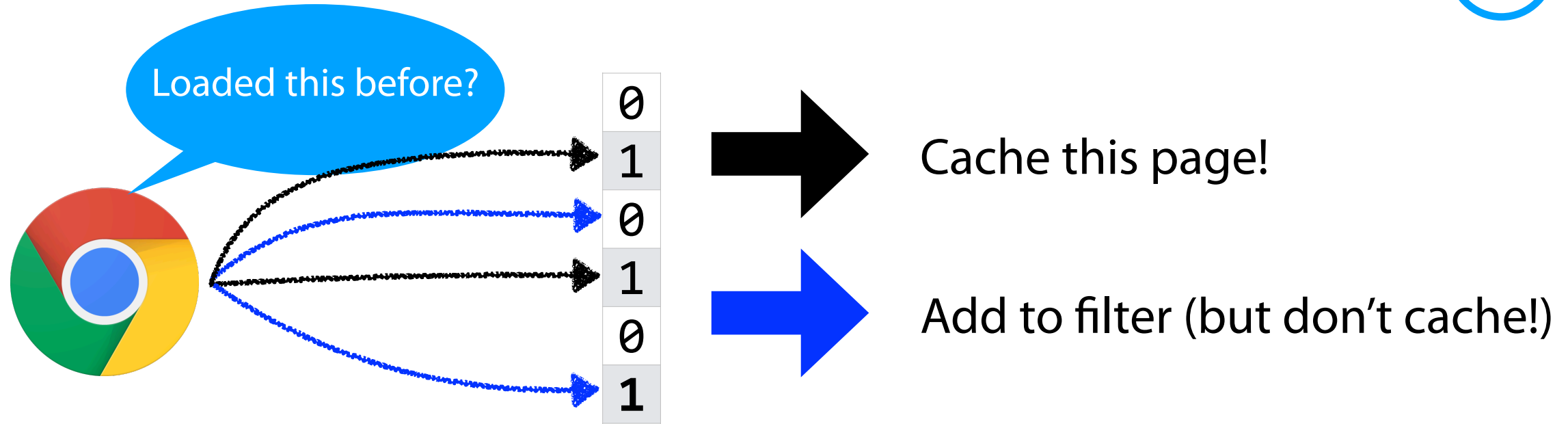
Bitwise Operators in C++

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

Bitwise Operators in C++

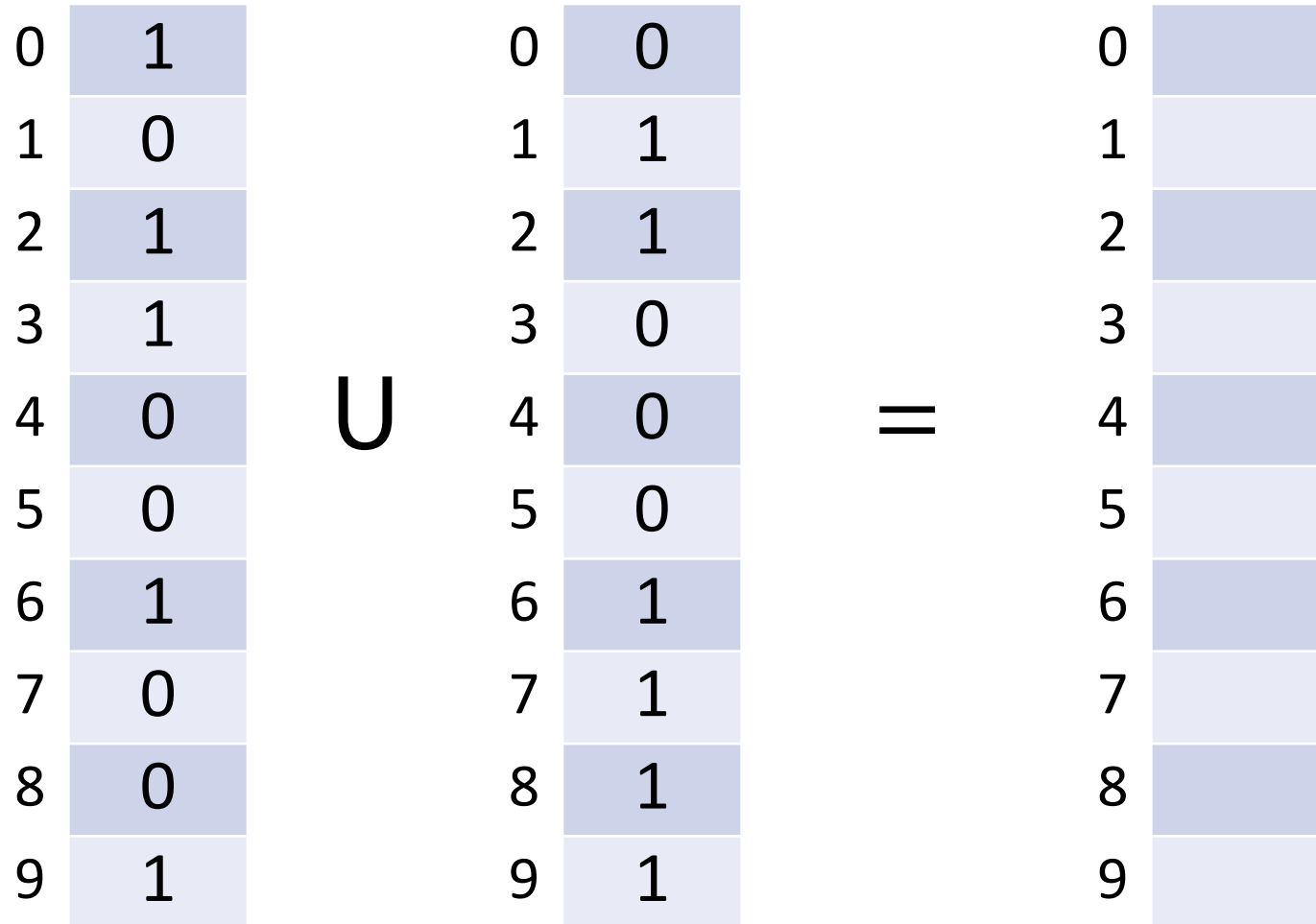Traditionally, bit vectors are read from RIGHT to LEFT

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	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	

Bloom Filters: Intersection

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

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

U

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

=

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
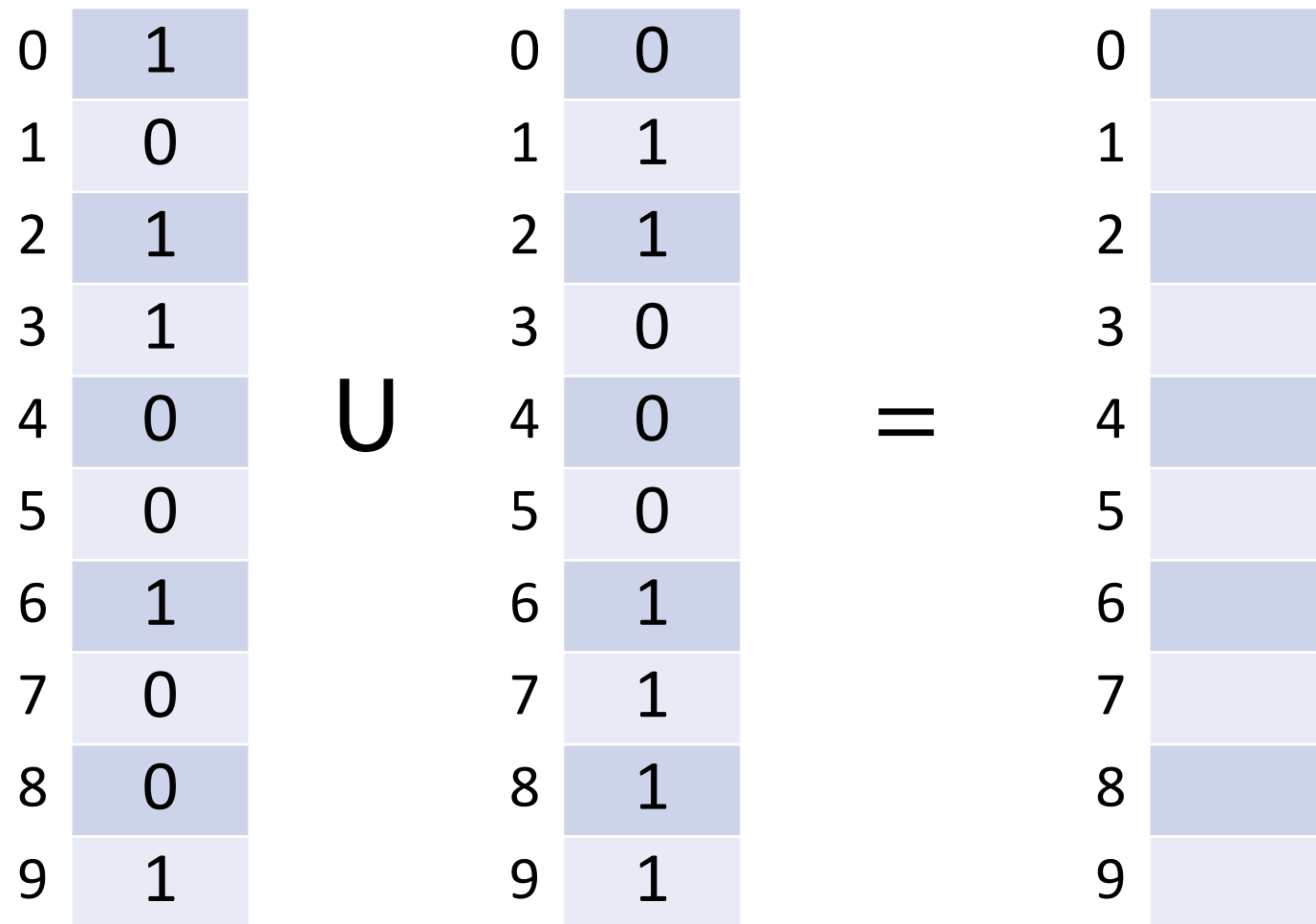
Sequence Bloom Trees

Imagine we have a large collection of text…

ATGGTTAGAATTAAACCCGG
TGCTAATAAACCUAGTGATG

CGATAGCACAGGTAGATCC
TACGTAGAGGTCATTAGCC

TACGTAGAGGTCATTAGCCG
TGCTAATAAACCUAGTGATG

….
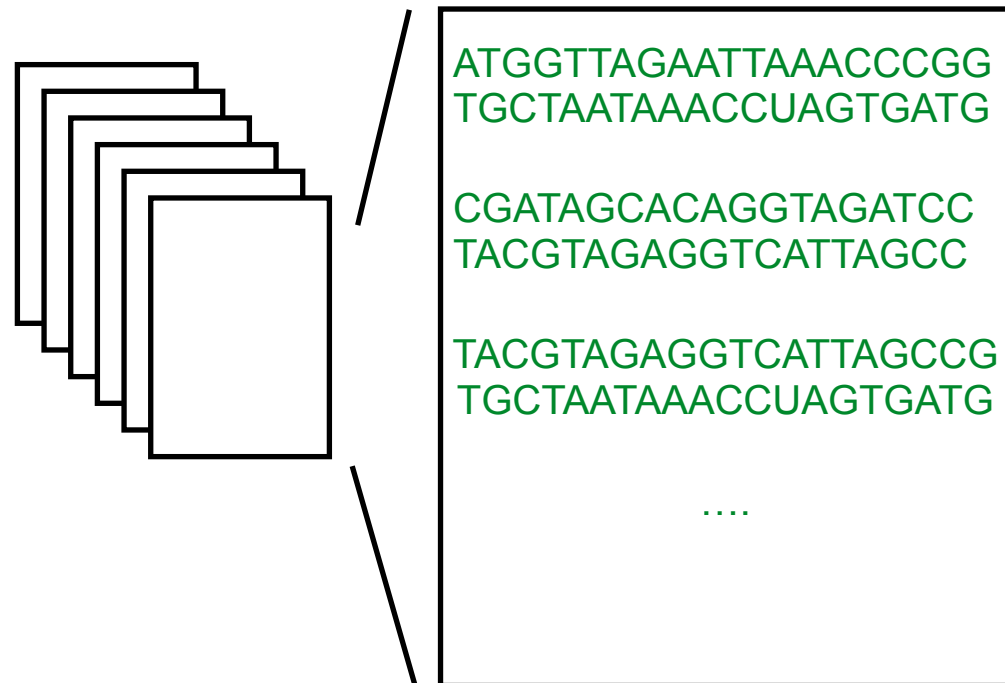
And our goal is to search these files
for a query of interest…
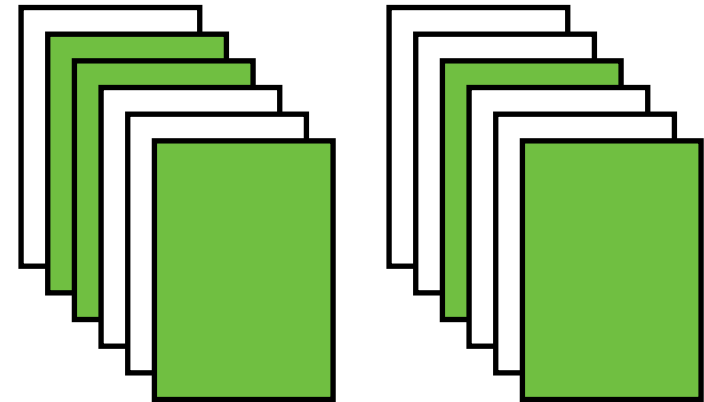
Bit Vector Merging

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

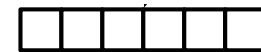|							
SRA 00001	SRA 00002	SRA 00003	SRA 00004	SRA 00005	SRA 00006	SRA 00007	SRA 00008

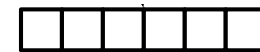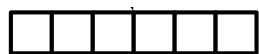# Sequence Bloom Trees

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

Sequence Bloom Trees

Are ≥ θ fraction of query kmers ∈ this Bloom filter?

Bloom filter

If YES, move to children

If NO, stop looking at this subtree (Global mismatch)

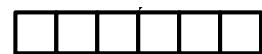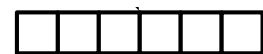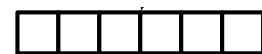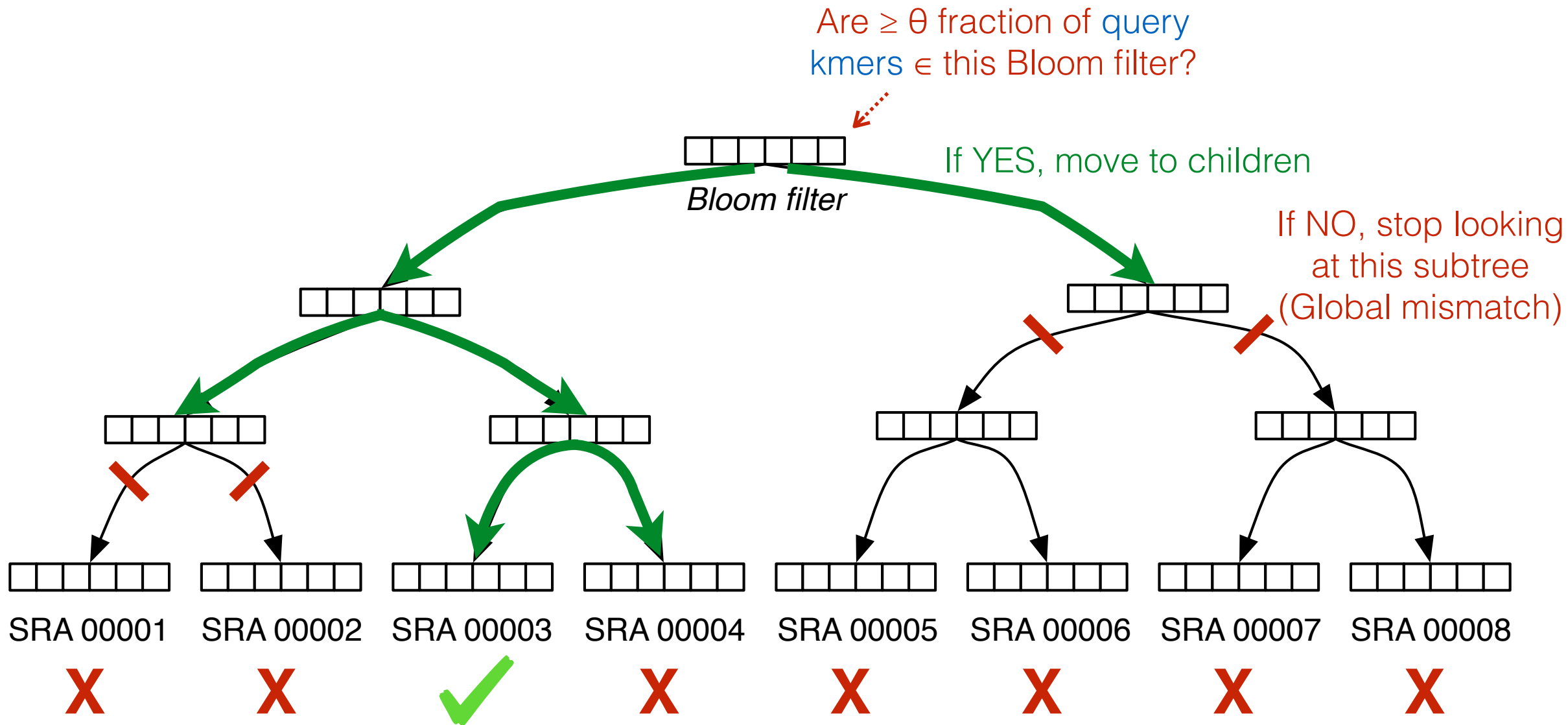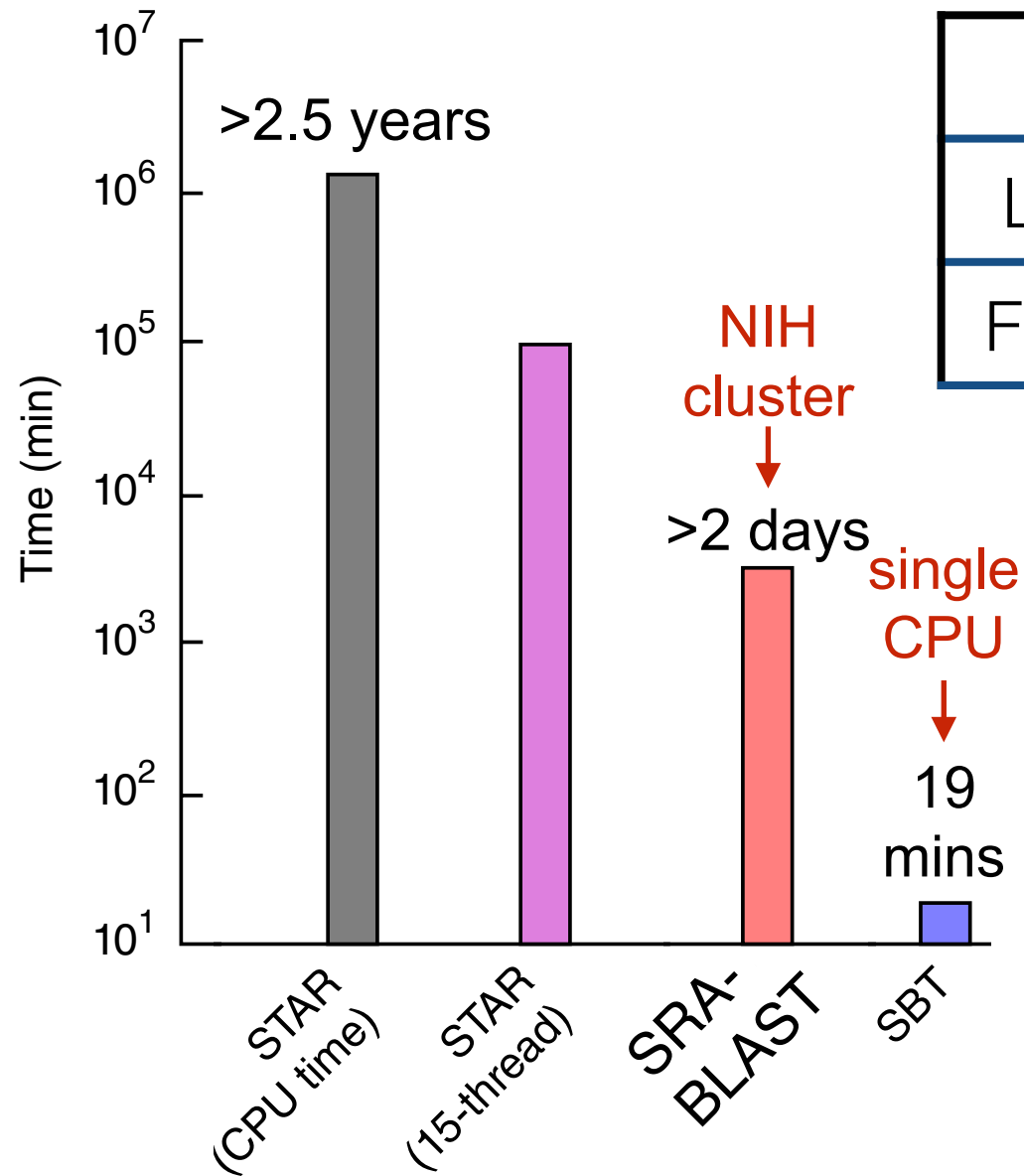SRA 00001   SRA 00002   SRA 00003   SRA 00004   SRA 00005   SRA 00006   SRA 00007   SRA 00008

X   X   ✓   X   X   X   X   X

Sequence Bloom Trees

>2.5 years

NIH cluster

>2 days

single CPU

19 mins

Time (min)

10^7 10^6 10^5 10^4 10^3 10^2 10^1

STAR (CPU time) | STAR (15-thread) | SRA-BLAST | SBT
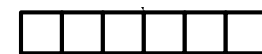
	SRA	FASTA.gz	SBT
Leaves	4966 GB	2692 GB	63 GB
Full Tree	-	-	200 GB

$\theta = 1.0$ $\theta = 1.0$ $\theta = 1.0$

False positive

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…