

Data Structures and Algorithms

Exam 5 Review and Bloom Filters

CS 225
Brad Solomon

April 24, 2026

hash
review

Was
all
review

lots of
good
questions!

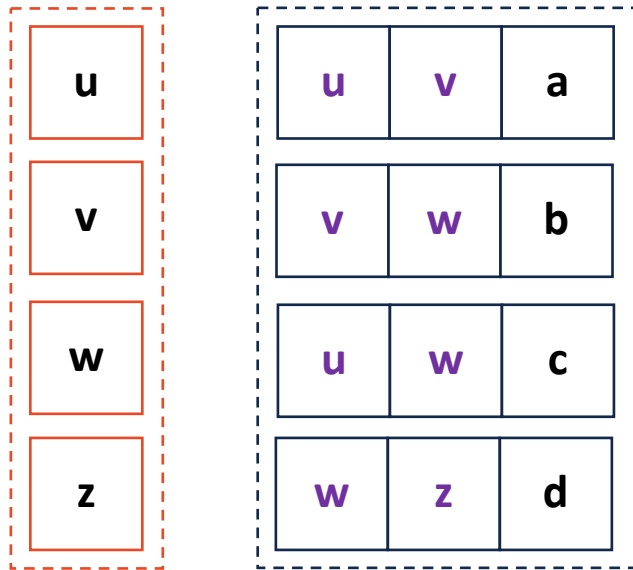
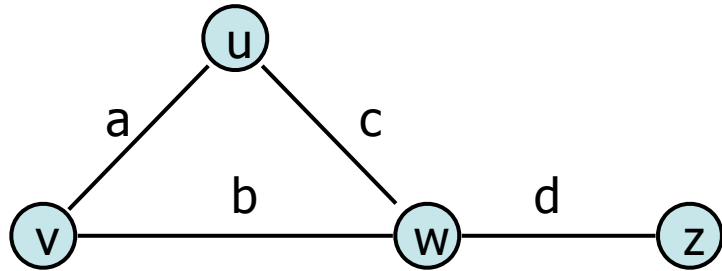


UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Graph Implementation: Edge List $|V| = n, |E| = m$

* The equivalent of an 'unordered' data structure *



Vertex Storage:

An optional list of vertices

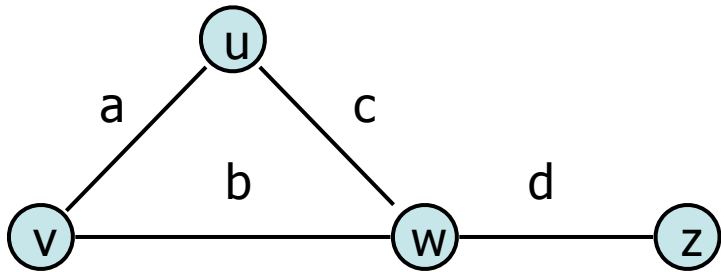
Edge Storage:

A list storing edges as (V1, V2, Weight)

Most graphs are stored as just an edge list!

Graph Implementation: Adjacency Matrix

$$|V| = n, |E| = m$$



Vertex Storage:

A hash table of vertices

Implicitly or explicitly store index

Edge Storage:

A $|V| \times |V|$ matrix of edges

Weight is stored at position (u, v)

$O(n)$

u	0
v	1
w	2
z	3

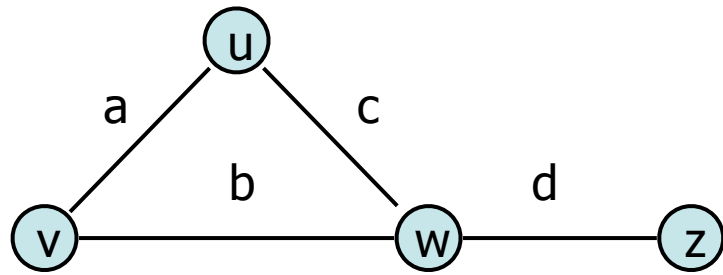
	u	v	w	z
0	-	a	c	0
1		-	b	0
2			-	d
3				-

Taking advantage of array lookup

Big (high storage cost)

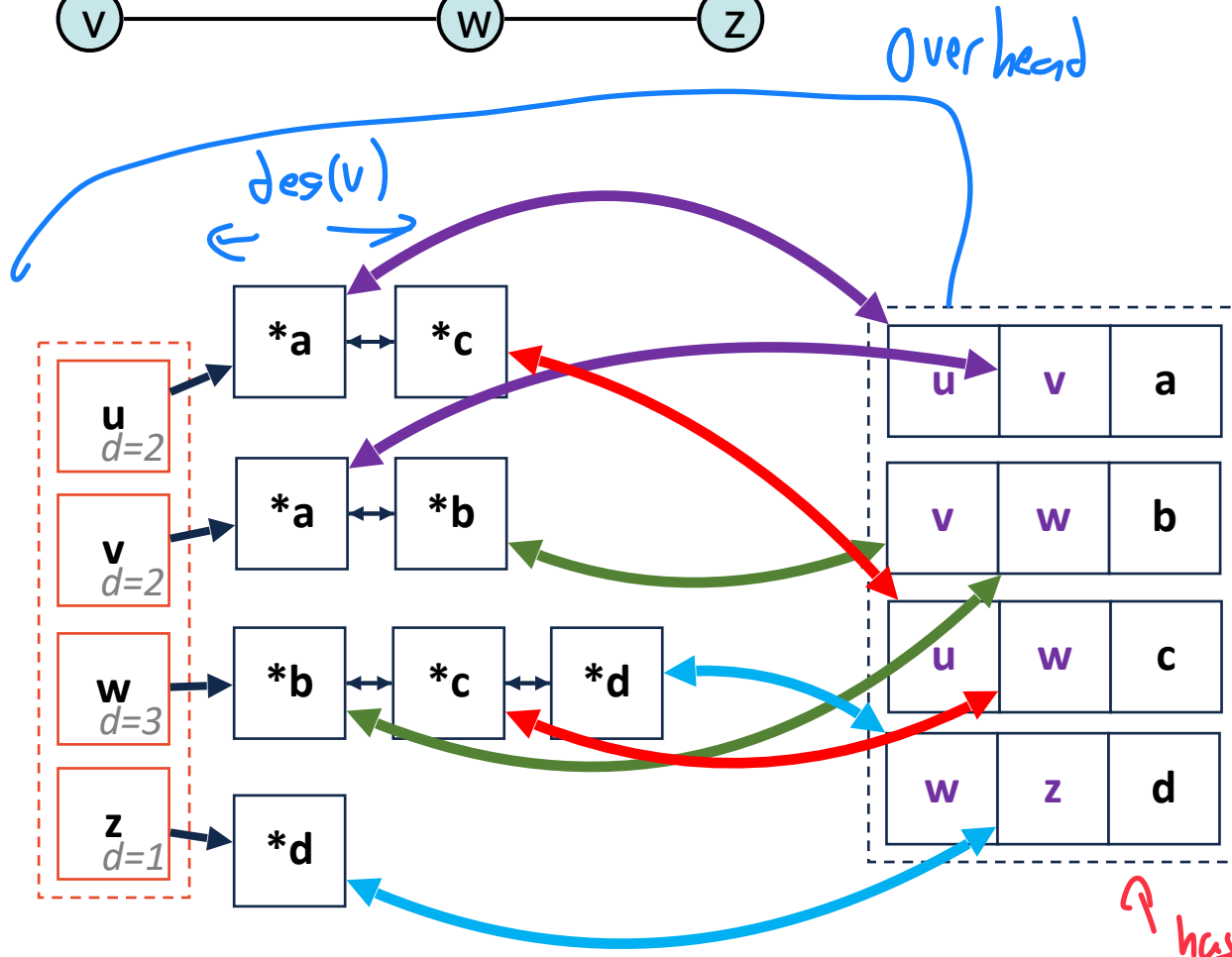
Certain edge ops are fast!

Adjacency List



Vertex Storage:

A bidirectional linked list with size variable
Each node is a pointer to edge in edge list



Edge Storage:

A list of (v1, v2, weight) edges
Also store pointers back to nodes

↑ has edge list

$$|V| = n, |E| = m$$



Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	$n+m$	n^2	$n+m$
insertVertex(v)	1^*	n^*	1^*
removeVertex(v)	$n+m$	n	$\text{deg}(v)$
insertEdge(u, v)	1	1	1^*
removeEdge(u, v)	m	1	$\min(\text{deg}(u), \text{deg}(v))$
incidentEdges(v)	m	n	$\text{deg}(v)$
areAdjacent(u, v)	m	1	$\min(\text{deg}(u), \text{deg}(v))$

Spase

vs

Dense

why

For some points memorize this! → for full credit, understand why

Summary: DFS and BFS

$|V| = n, |E| = m$

Both are $O(n+m)$ traversals! They label every edge and every node

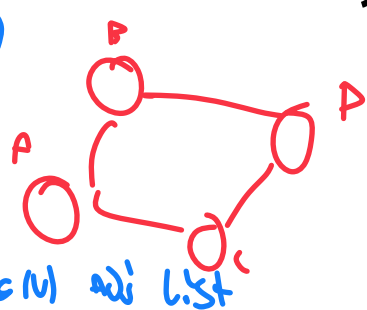
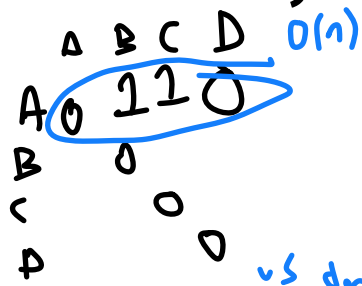
BFS

Solves unweighted MST

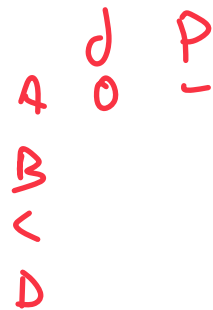
Solves shortest path

Solves cycle detection

Memory bounded by width



vs $O(m)$ adj list

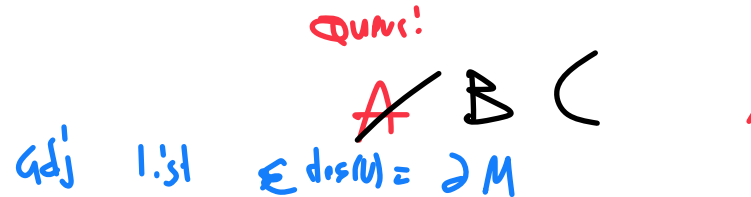


DFS

Solves unweighted MST

Solves cycle detection

Memory bounded by longest path



Querc:



adj list

$O(m) = 2M$

Kruskal's Algorithm

↳ MST problem!

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G.vertices()):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   Q.buildFromGraph(G.edges())
8
9   Graph T = (V, {})
10
11  while |T.edges()| < n-1:
12    Vertex (u, v) = Q.removeMin()
13    if forest.find(u) != forest.find(v):
14      T.addEdge(u, v)
15      forest.union( forest.find(u),
16                  forest.find(v) )
17
18  return T
```

the next smallest edge

Does this connect two things?

Globally greedy / A/c plane!

- 1) Build a **priority queue** on edges
A minheap or *A sorted array*
- 2) Build a **disjoint set** on vertices
All vertices start as their own set
- 3) Loop through min edges
If edge connects two disjoint sets
Union sets and record edge in MST
- 4) Stop when:
N-1 edges recorded
Only a single disjoint set remains

Kruskal's Algorithm

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G.vertices()):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   Q.buildFromGraph(G.edges())
8
9   Graph T = (V, {})
10
11  while |T.edges()| < n-1:
12    Vertex (u, v) = Q.removeMin()
13    if forest.find(u) != forest.find(v):
14      T.addEdge(u, v)
15      forest.union( forest.find(u),
16                  forest.find(v) )
17
18  return T
19
```

$$O(\log(m)) = O(\log(n))$$

$$|V| = n, |E| = m$$

What is the Big O?

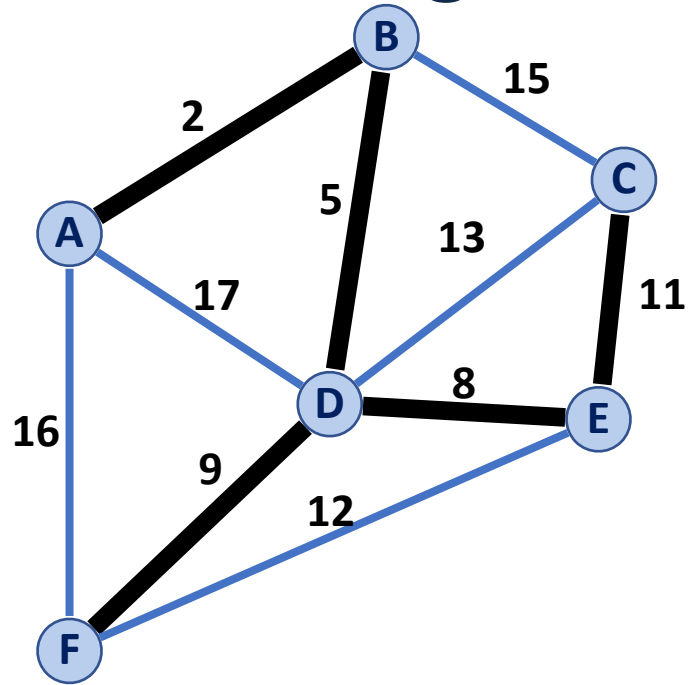
2 — 4: $O(n)$ *Init*
* 6 — 7: **Heap: $O(m)$** *!!*
* Sorted List: $O(m \log m)$

11: $m \times \langle 12-17 \rangle$ *m log m*
12 — 17: **Sorted List: $O(1)$** *!!*

$O(n + m + m \log m)$ *o/n*

Simplified: $O(n + m \log n)$

Prim's Algorithm



A	B	C	D	E	F
0, —	2, A	11, E	5, B	8, D	9, D

Locally greedy

```

1 PrimMST(G, s):
2   Input: G, Graph;
3         s, vertex in G, starting vertex
4   Output: T, a minimum spanning tree (MST) of G
5
6   foreach (Vertex v : G.vertices()):
7     d[v] = +inf
8     p[v] = NULL
9   d[s] = 0
10
11  PriorityQueue Q // min distance, defined by d[v]
12  Q.buildHeap(G.vertices())
13  Graph T // "labeled set"
14
15  repeat n times:
16    Vertex m = Q.removeMin()
17    T.add(m)
18    foreach (Vertex v : neighbors of m not in T):
19      if cost(v, m) < d[v]:
20        * d[v] = cost(v, m) *
21        p[v] = m
22
23  return T
  
```

← 2

] find smallest weight edge

← changing a distance in a priority queue!

Prim's Big O

7 — 9: $O(n)$

12—14:

MinHeap: $O(n)$

Unsorted Array: $O(1)$

16—22: Complicated!



$$|V| = n, |E| = m$$

```
6 PrimMST(G, s):
7   foreach (Vertex v : G.vertices()):
8     d[v] = +inf
9     p[v] = NULL
10    d[s] = 0
11
12    PriorityQueue Q // min distance, defined by d[v]
13    Q.buildHeap(G.vertices())
14    Graph T // "labeled set"
15
16    repeat n times:
17      Vertex m = Q.removeMin()
18      T.add(m)
19      foreach (Vertex v : neighbors of m not in T):
20        if cost(v, m) < d[v]:
21          d[v] = cost(v, m)
22          p[v] = m
23
```

Problem: minheap $O(\log n)$

Seems like it should be $O(1)$

unsorted array in organized array

Depends on choice of **PriorityQueue** (MinHeap vs Unsorted Array)

Depends on choice of **Graph** (Adjacency Matrix vs Adjacency List)

Order
best
has
tr's

Prim's Algorithm

Sparse Graph: ($m \sim n$)

Dense Graph: ($m \sim n^2$)

```

6  PrimMST(G, s):
7  foreach (Vertex v : G.vertices()):
8      d[v] = +inf
9      p[v] = NULL
10 d[s] = 0
11
12 PriorityQueue Q // min distance, defined by d[v]
13 Q.buildHeap(G.vertices())
14 Graph T // "labeled set"
15
16 repeat n times:
17     Vertex m = Q.removeMin()
18     T.add(m)
19     foreach (Vertex v : neighbors of m not in T):
20         if cost(v, m) < d[v]:
21             d[v] = cost(v, m)
22             p[v] = m
23

```

Fib heap $\geq O(n^2 + m)$

This is $O(1)^*$

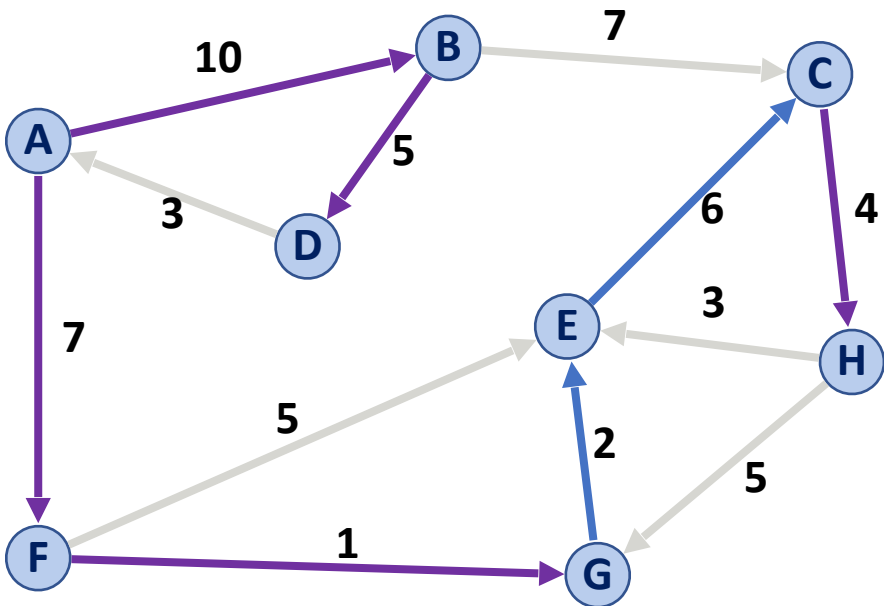
Lines 7 — 14 are $O(n)$ [at most]

matrix less efficient here

	Adj. Matrix	Adj. List
Heap	$O(n^2 + m \lg(n))$	$O(n \lg(n) + m \lg(n))$ $O(nbsn + m)$
Unsorted Array	$O(n^2)$	$O(n^2)$

Dijkstra's Algorithm (SSSP)

Start or source



```

DijkstraSSSP(G, s):
6  foreach (Vertex v : G.vertices()):
7      d[v] = +inf
8      p[v] = NULL
9      d[s] = 0
10
11  PriorityQueue Q // min distance, defined by d[v]
12  Q.buildHeap(G.vertices())
13  Graph T          // "labeled set"
14
15  repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19          if cost(u, v) + d[u] < d[v]:
20              d[v] = cost(u, v) + d[u]
21              p[v] = u
    
```

Prim but weight is path not edge!

A	B	C	D	E	F	G	H
--	A	E	B	G	A	F	C
0	10	16	15	10	7	8	20

Floyd-Warshall Algorithm

Floyd-Warshall's Algorithm is an alternative to Dijkstra in the presence of negative-weight edges (not negative weight cycles).

```
1 FloydWarshall(G):
2   Let d be a adj. matrix initialized to +inf
3   foreach (Vertex v : G):
4     d[v][v] = 0
5   foreach (Edge (u, v) : G):
6     d[u][v] = cost(u, v)
7
8   foreach (Vertex u : G):
9     foreach (Vertex v : G):
10      foreach (Vertex w : G):
11        if (d[u, v] > d[u, w] + d[w, v])
12          d[u, v] = d[u, w] + d[w, v]
```



A Hash Table based Dictionary

User Code (is a map):

```
1 Dictionary<KeyType, ValueType> d;  
2 d[k] = v;
```

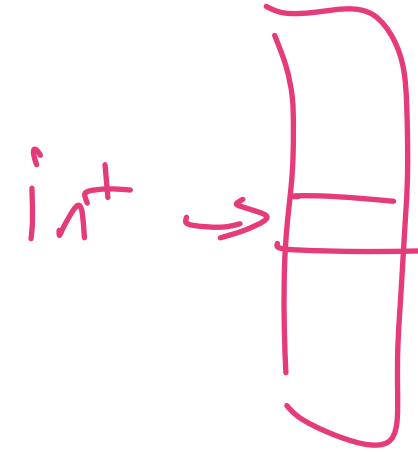
A **Hash Table** consists of three things:

1. A hash function

$h(k) \rightarrow \text{int}$

2. A data storage structure

\rightarrow
store at

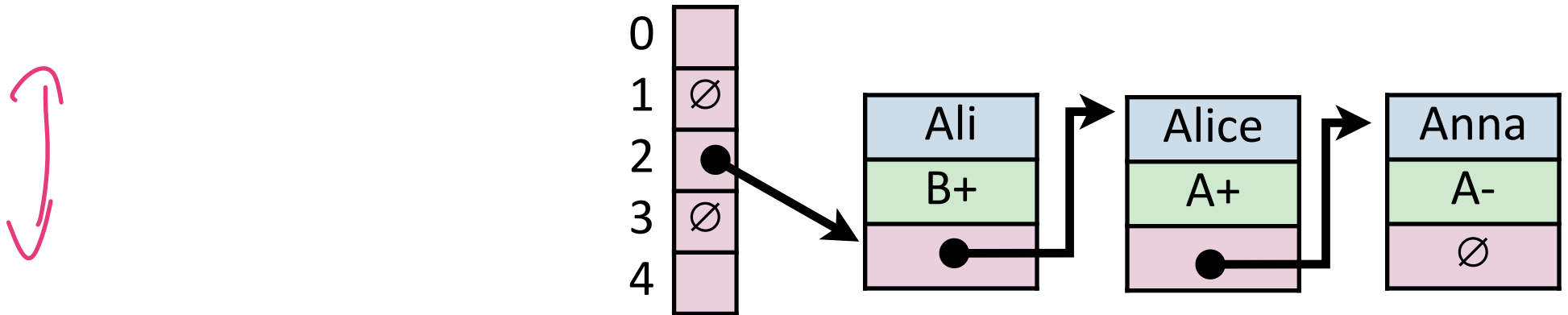


3. A method of addressing *hash collisions*

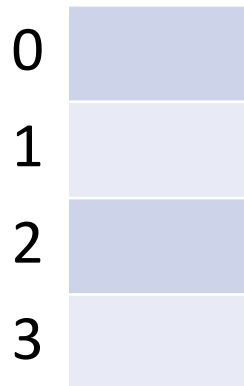
Open vs Closed Hashing

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



Separate Chaining Under SUHA

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$ **Table Size: m**

α_j = expected # of items hashing to position j **Num objects: n**

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} 1 & \text{if item } i \text{ hashes to } j \\ 0 & \text{otherwise} \end{cases}$$

$$E[\alpha_j] = E\left[\sum_i H_{i,j}\right]$$

$$Pr[H_{i,j} = 1] = \frac{1}{m}$$

$$E[\alpha_j] = n * Pr(H_{i,j} = 1)$$

$$\mathbf{E}[\alpha_j] = \frac{\mathbf{n}}{\mathbf{m}}$$

load factor ☺

Separate Chaining Under SUHA

Under SUHA, a hash table of size m and n elements:

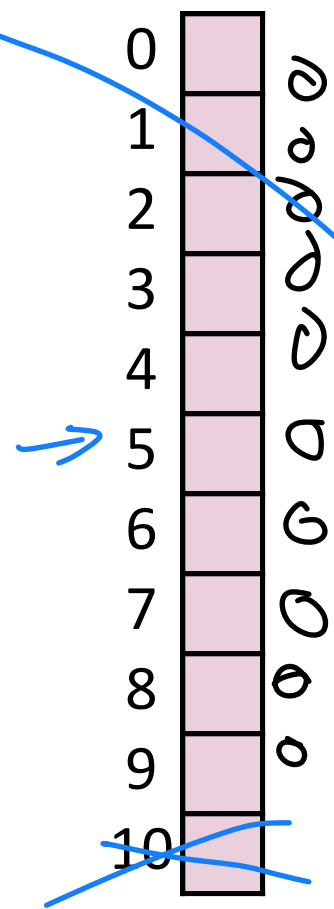
Find runs in: $O(1 + \alpha)$

Insert runs in: $O(1)$

Remove runs in: $O(1 + \alpha)$



$$h(k) = k \% 10$$



$$0 \leq \alpha < \infty$$

Suha says blue went happen

α is length / load factor

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:

Runtime approaches infinity!

- If α is constant:

Runtime is a constant!

in expectation under SUHA

Resizing a hash table

$$O \leq \alpha < 1$$

When and how do you resize?

|
When we reach $0.7 - 0.9 = \alpha$

How

1) Double size of array ...

if double hashing in general good to do
to closest prime ~~#~~
next largest

2) Re compute all hashes (redo all inserts)

std::map in C++

Red-Black tree

```
T& map<K, V>::operator[]
```

```
pair<iterator, bool> map<K, V>::insert()
```

```
iterator map<K, V>::erase()
```

```
iterator map<K, V>::lower_bound( const K & );
```

```
iterator map<K, V>::upper_bound( const K & );
```

Why does this mean no hash table?

↓
Giving me closest value to k

hash table cannot do this!

std::unordered_map in C++

```
T& unordered_map<K, V>::operator[]
```

```
pair<iterator, bool> unordered_map<K, V>::insert()
```

```
iterator unordered_map<K, V>::erase()
```

```
iterator map<K, V>::lower_bound( const K & );
```

```
iterator map<K, V>::upper_bound( const K & );
```

```
float unordered_map<K, V>::load_factor();
```

```
void unordered_map<K, V>::max_load_factor(float m);
```


Running Times

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

Running Times (Tradeoff Highlights)



	Hash Table	AVL	Linked List
Find	Expectation*: $O(1)$ *** Worst Case: $O(n)$	$O(\log n)$	$O(n)$
Insert	Expectation*: $O(1)$ *** Worst Case: $O(n)$	$O(\log n)$	$O(1)$
Storage Space	$O(n)$	$O(n)$	$O(n)$



Any (review) questions?

Learning Objectives

Review when you would prefer different data structures

Build a conceptual understanding of a bloom filter

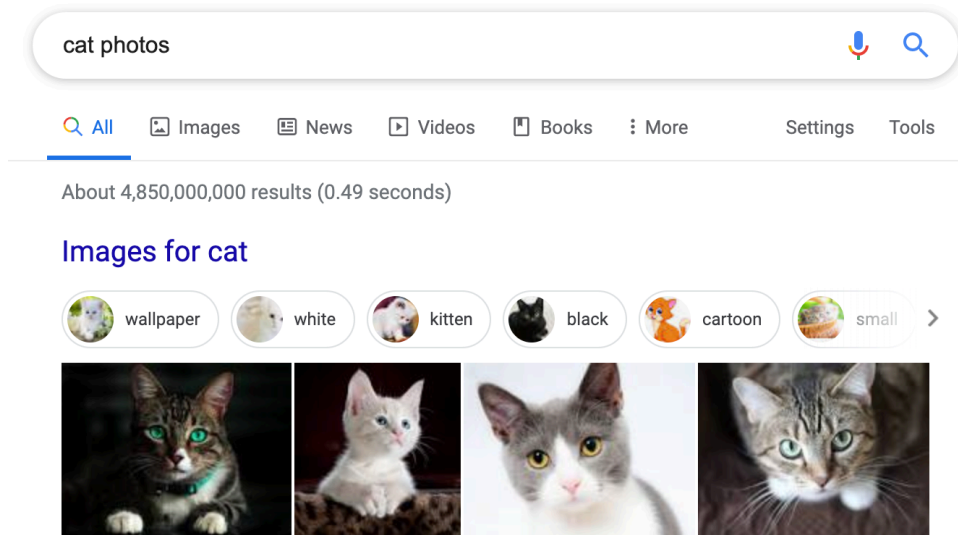
Review probabilistic data structures and one-sided error

Formalize the math behind the bloom filter

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)



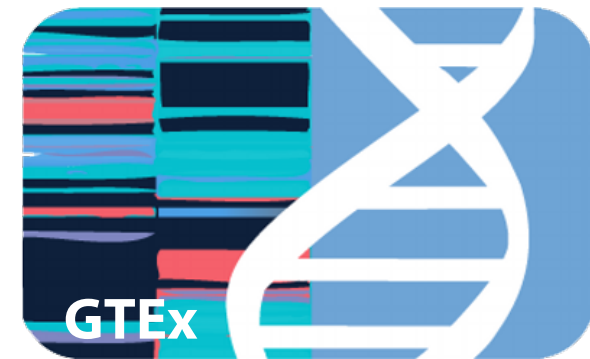
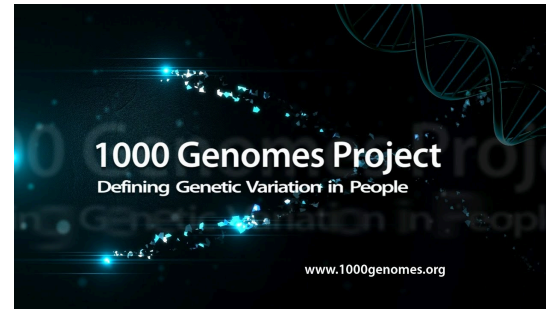
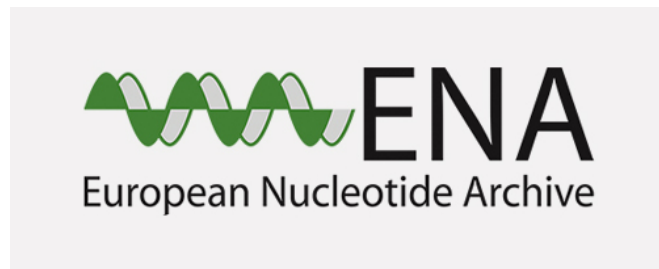
Google Index Estimate: >60 billion webpages

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)

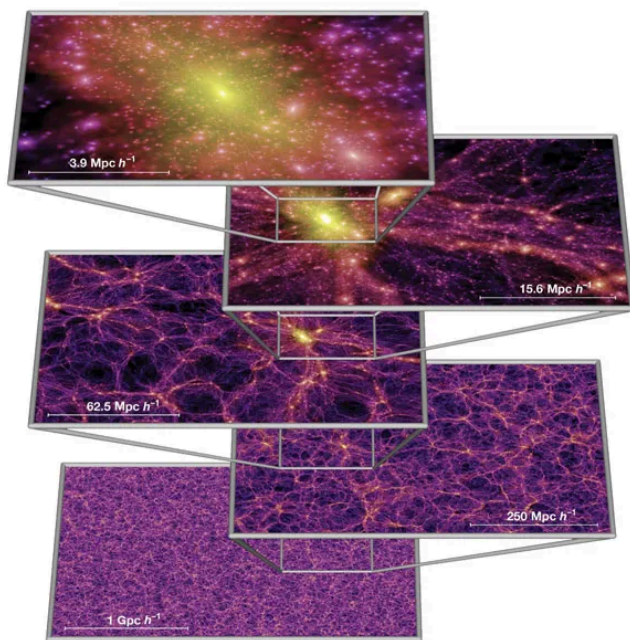


Image: <https://doi.org/10.1038/nature03597>

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

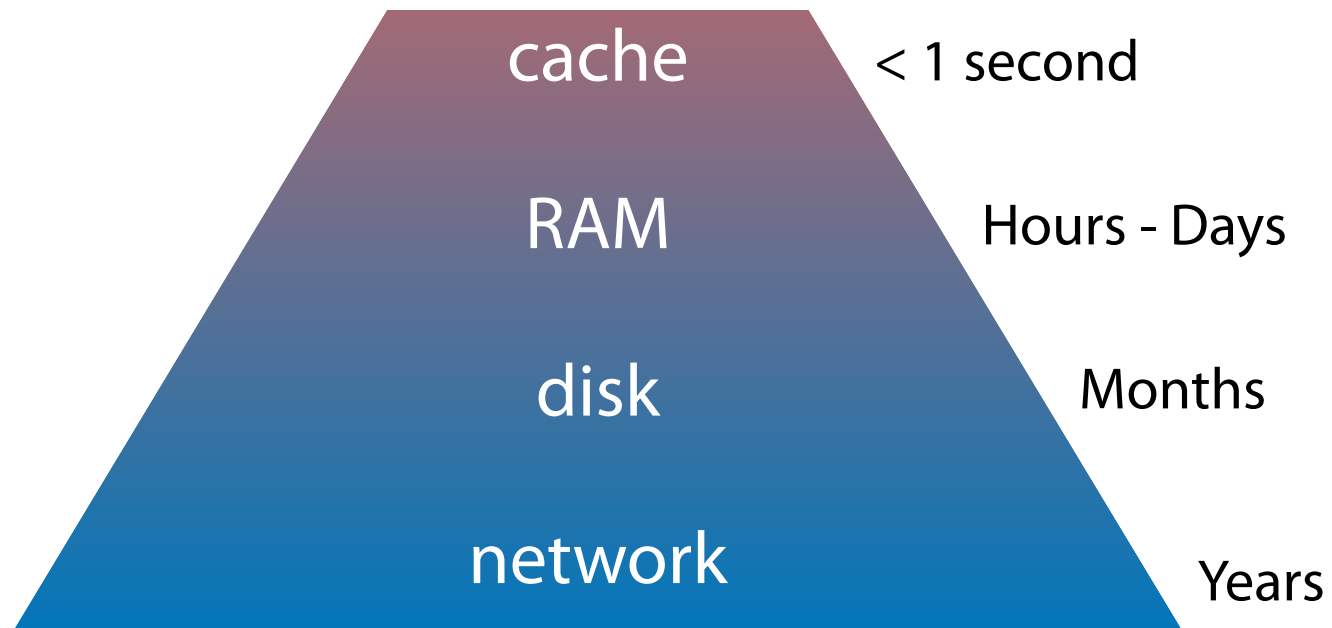
Table: <http://doi.org/10.5334/dsj-2015-011>

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*?

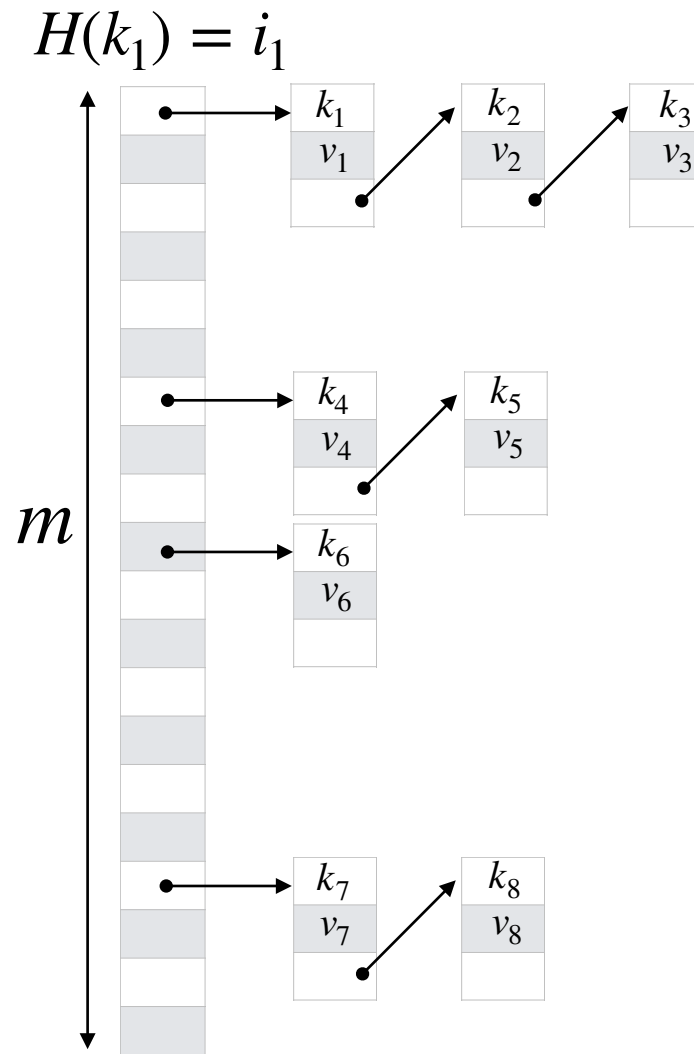
Reducing storage costs

1) Throw out information that isn't needed

2) Compress the dataset

Reducing a hash table

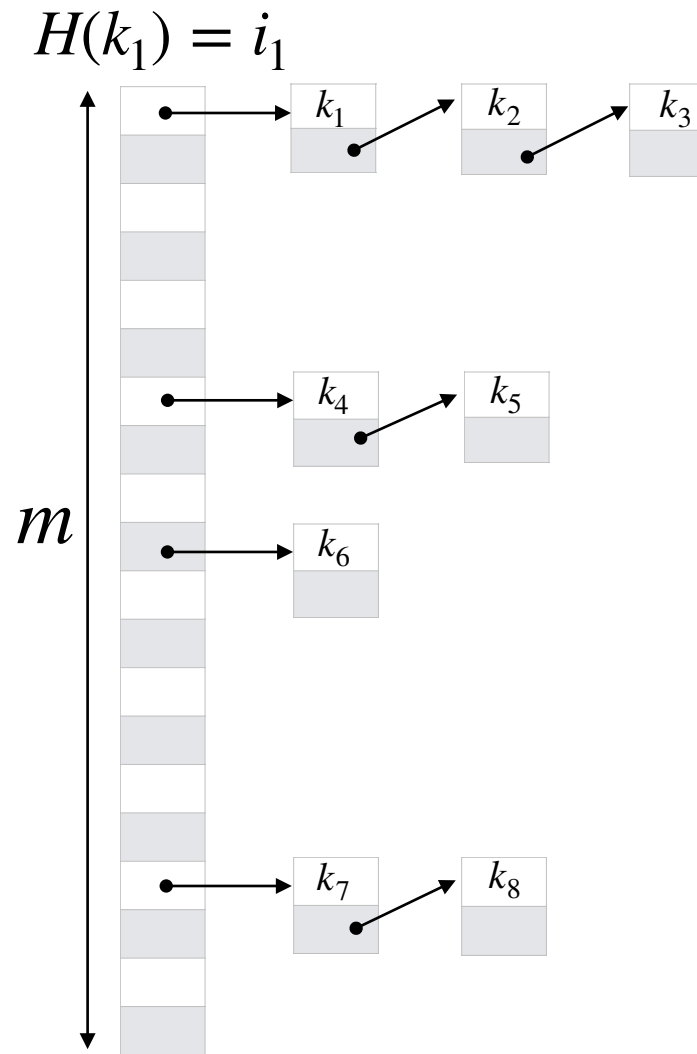
What can we remove from a hash table?



Reducing a hash table

What can we remove from a hash table?

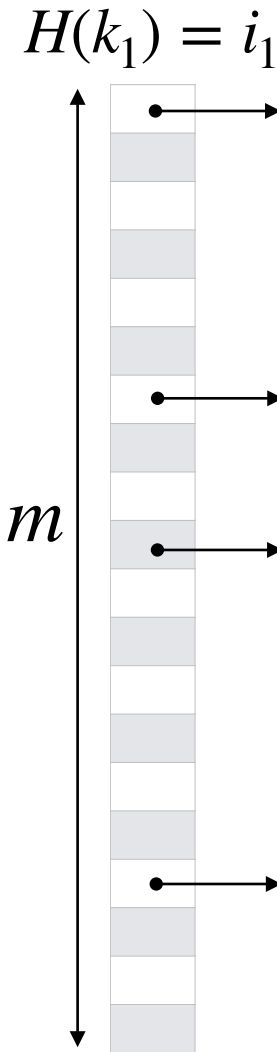
Take away values



Reducing a hash table

What can we remove from a hash table?

Take away values and keys



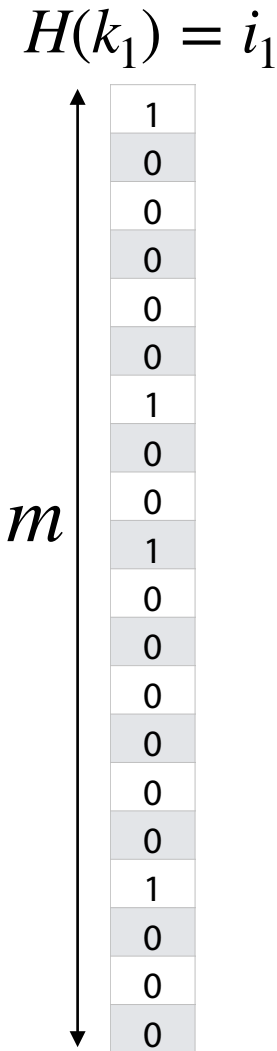


Reducing a hash table

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

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

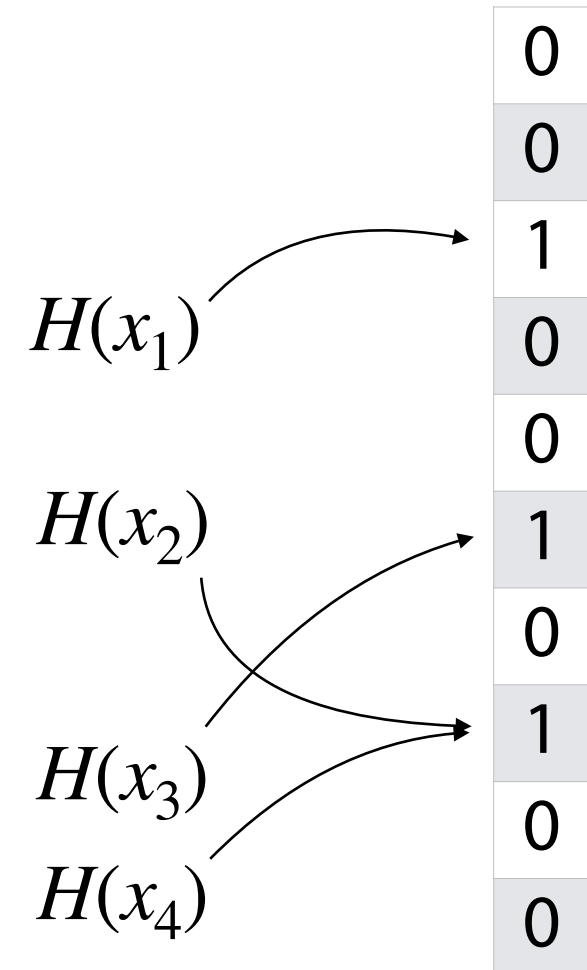
$h(k) = k \% 7$

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

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

$h(k) = k \% 7$

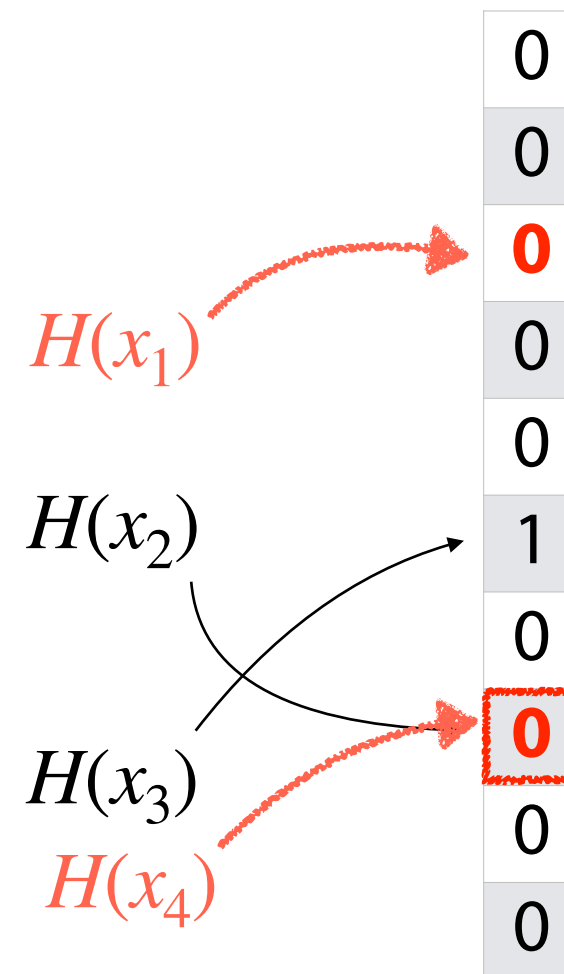
0	0
1	1
2	1
3	0
4	1
5	0
6	1

`_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)`

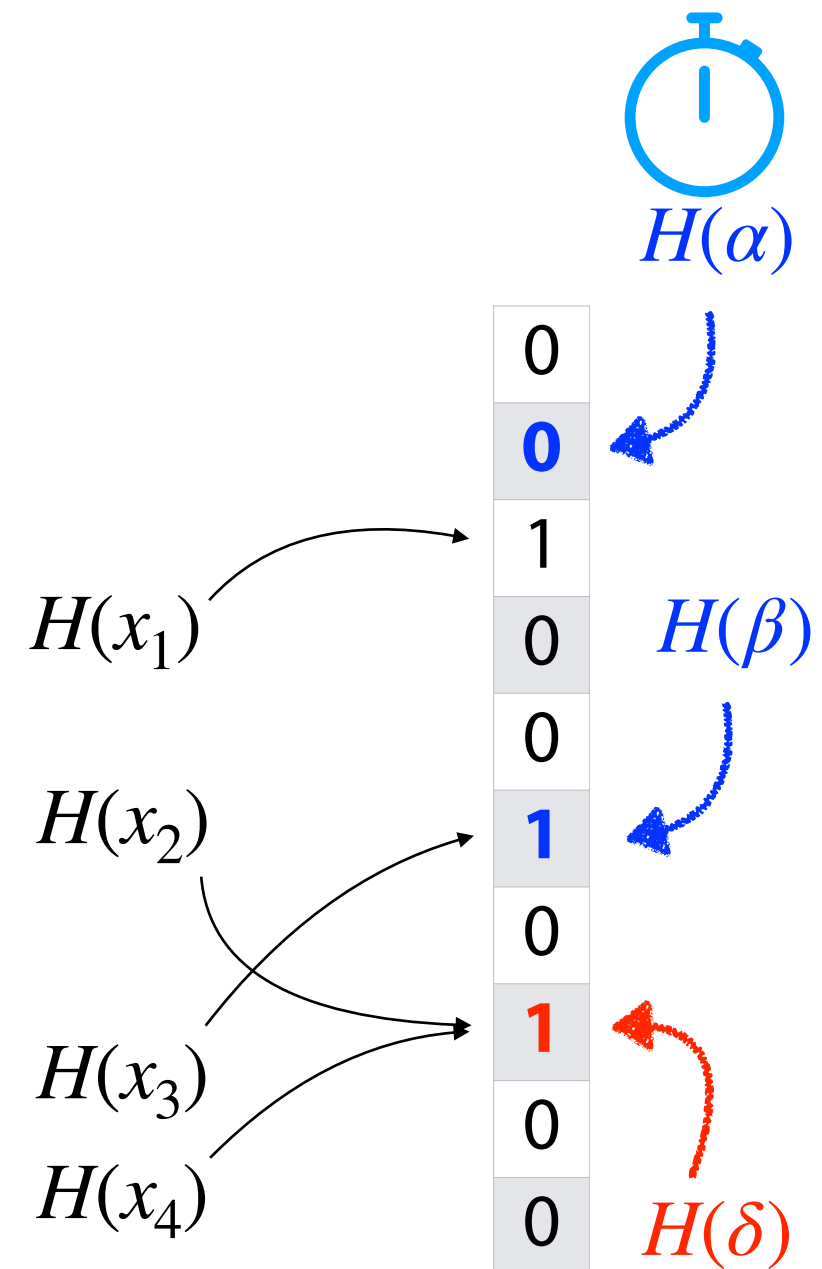
`_find(3)`

Bloom Filter: Search

The bloom filter is a *probabilistic* data structure!

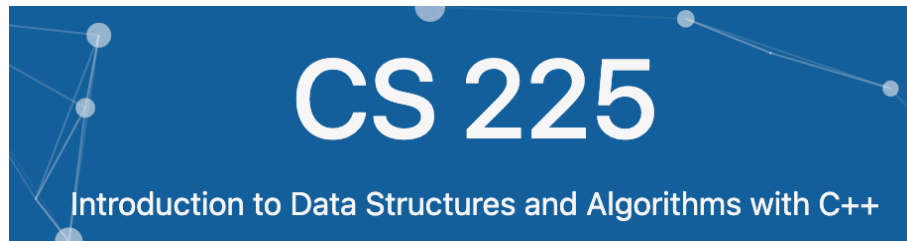
If the value in the BF is 0:

If the value in the BF is 1:

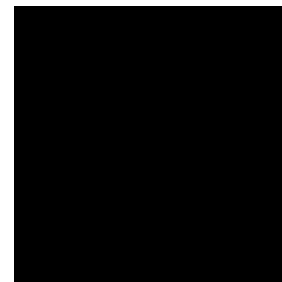
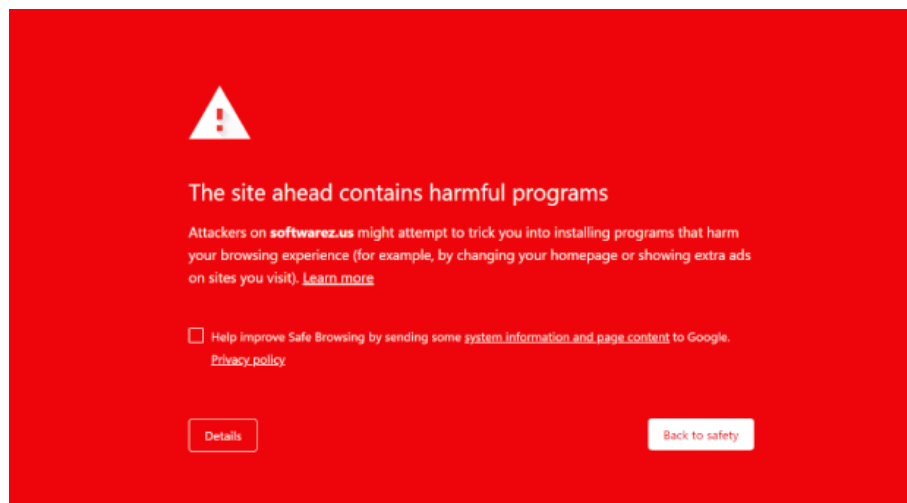


Probabilistic Accuracy: Malicious Websites

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

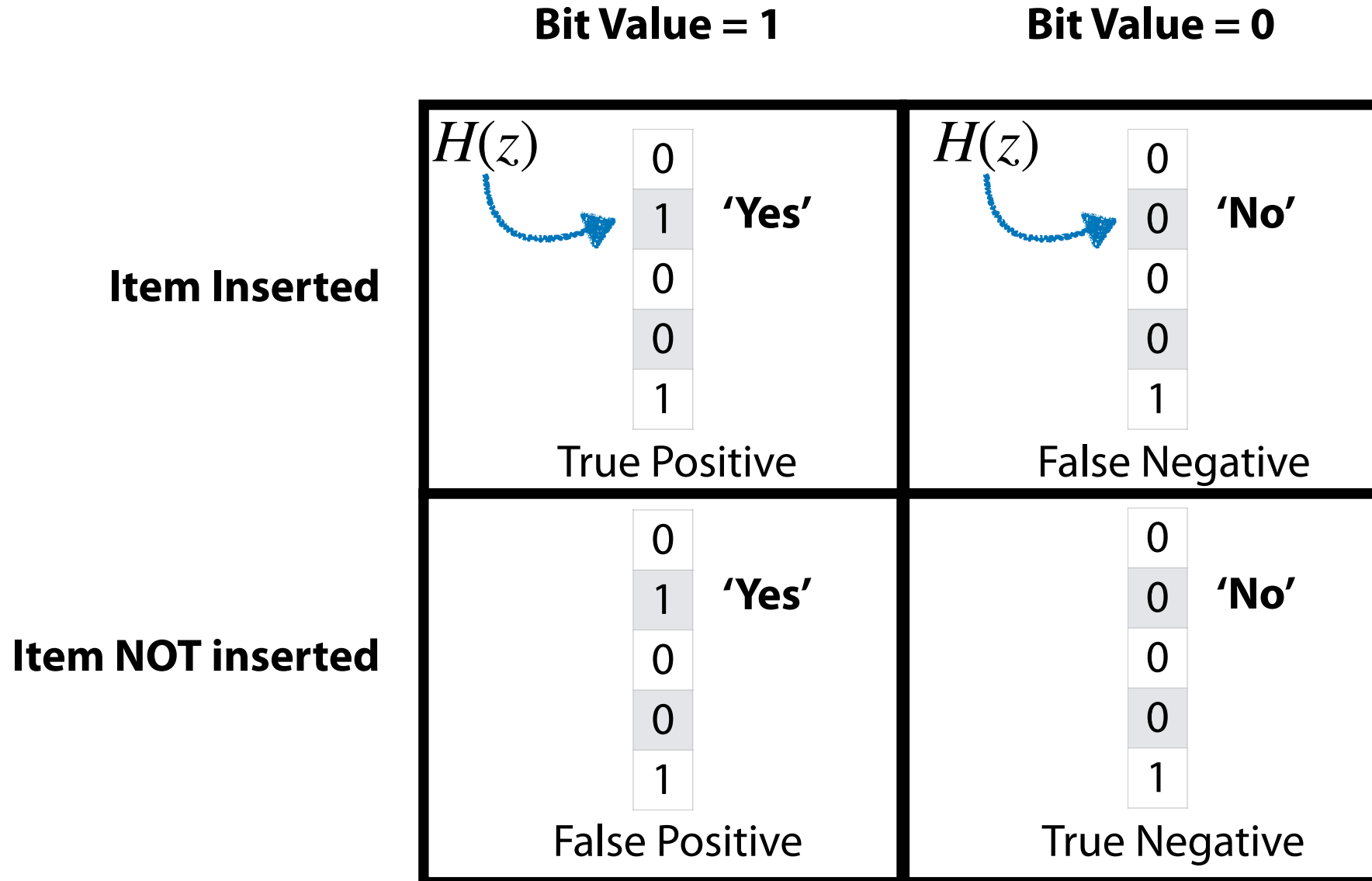
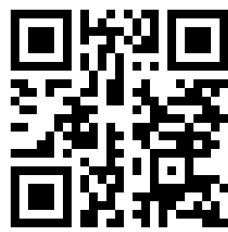
True Positive:

False Positive:

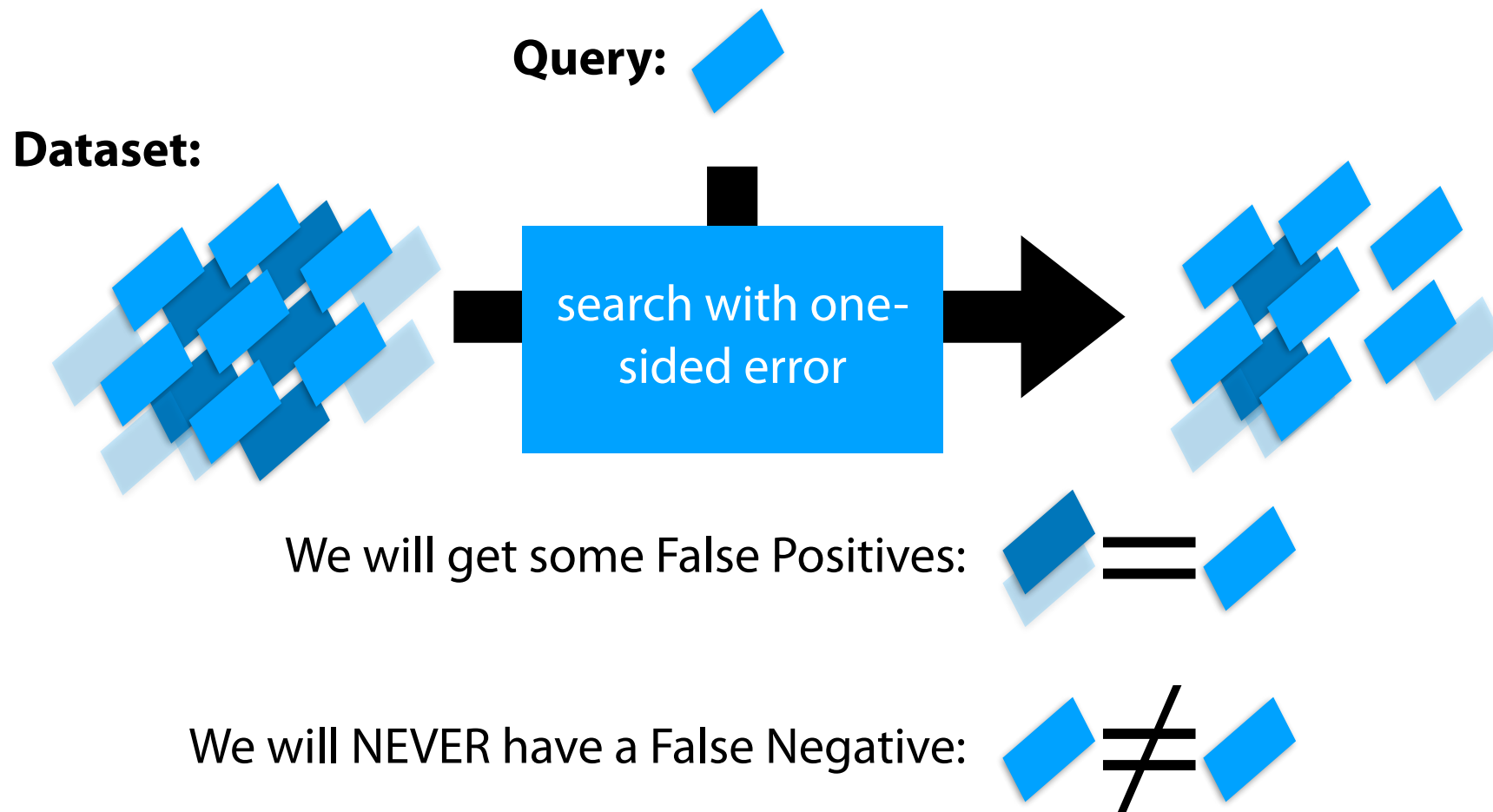
False Negative:

True Negative:

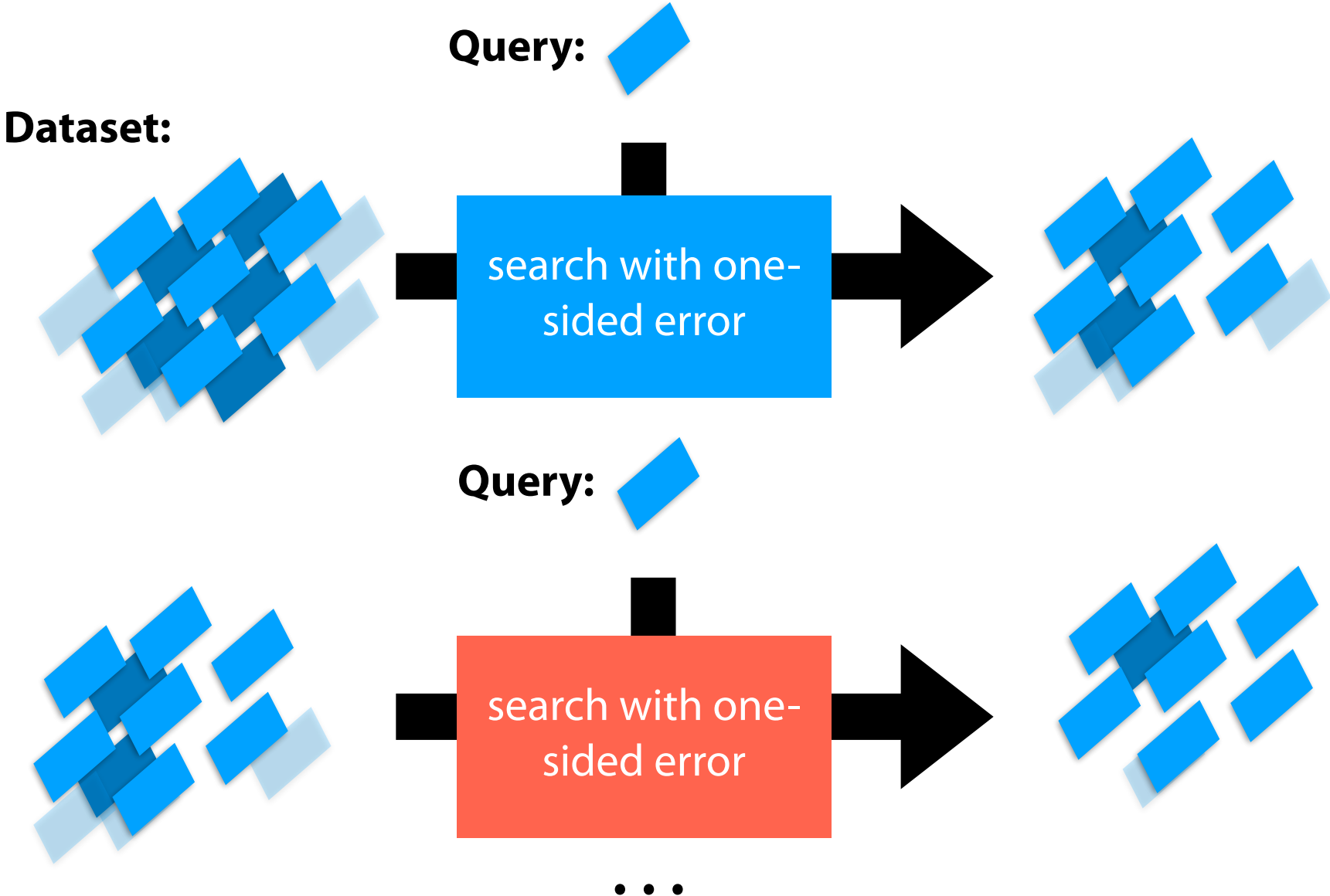
Imagine we have a **bloom filter** that **stores malicious sites...**



Probabilistic Accuracy: One-sided error

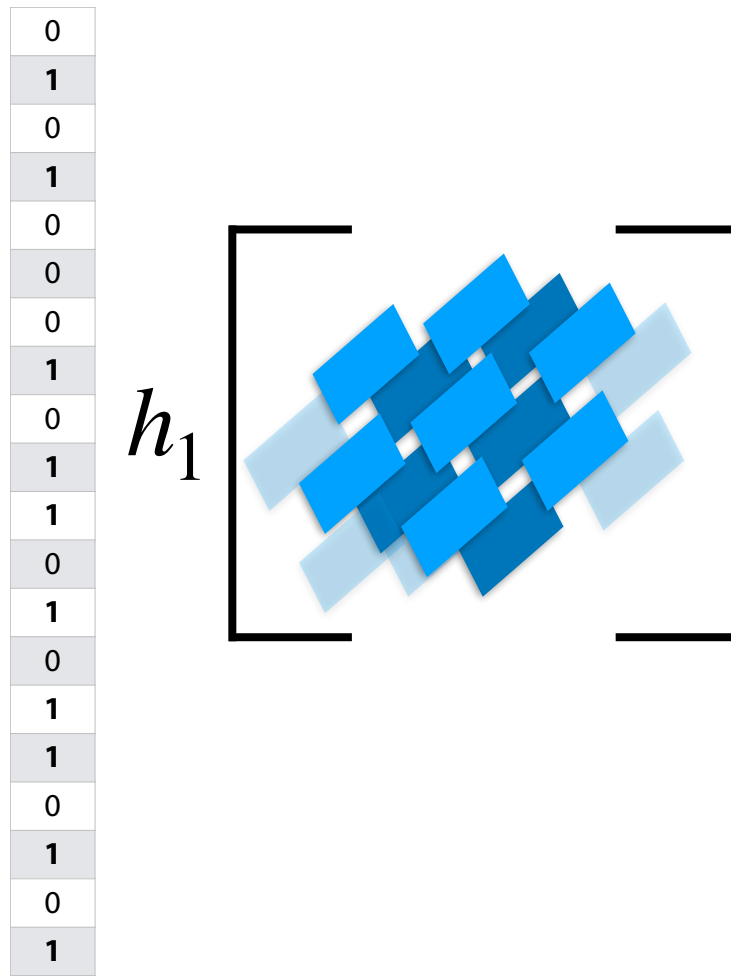


Probabilistic Accuracy: One-sided error



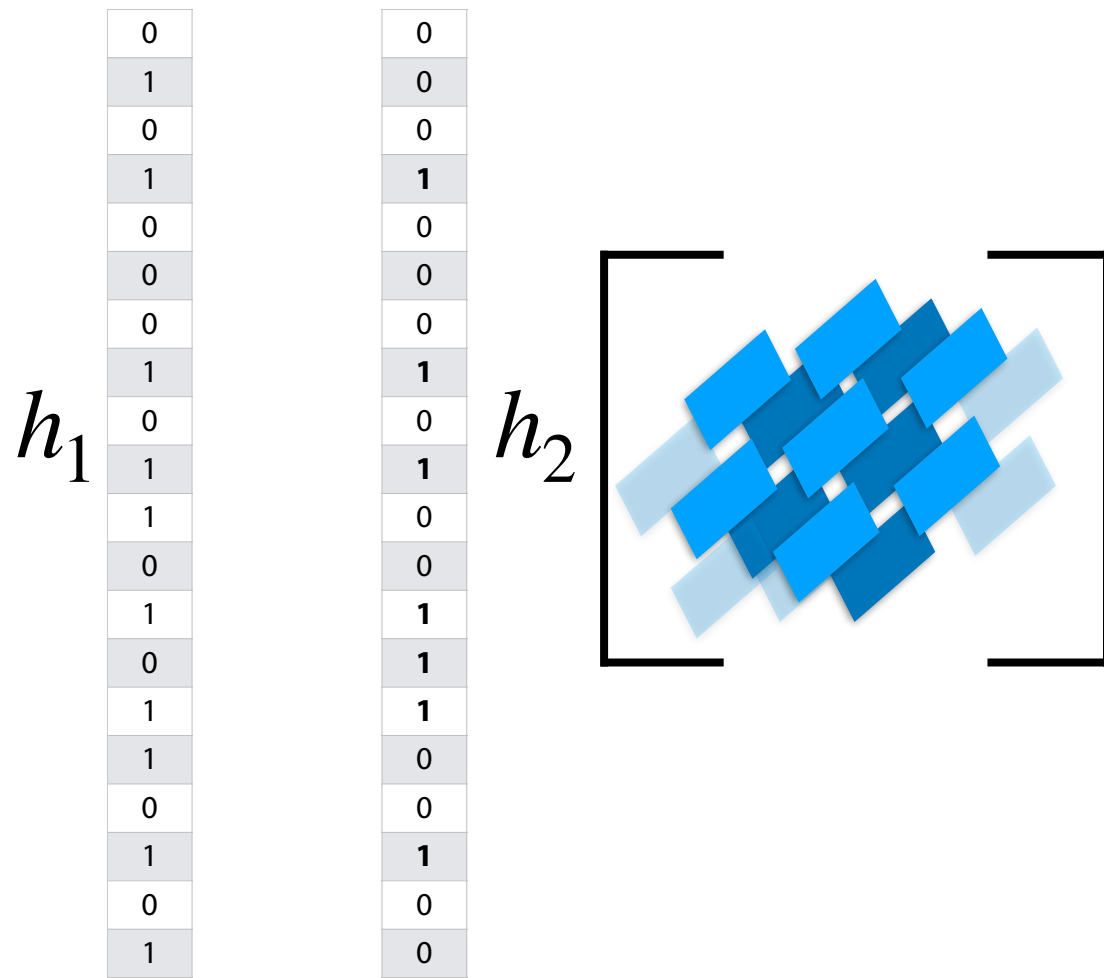
Bloom Filter: Repeated Trials

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



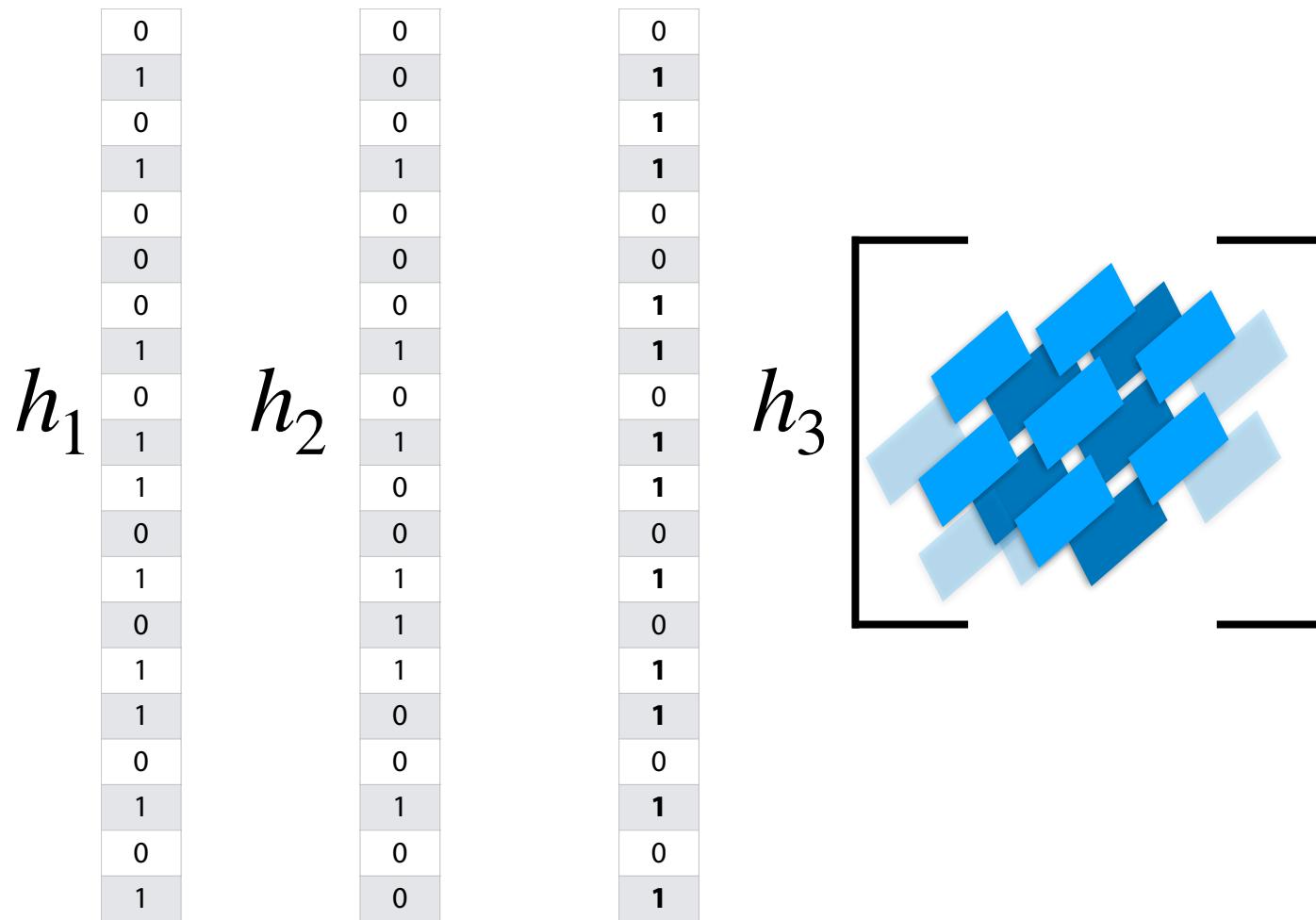
Bloom Filter: Repeated Trials

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



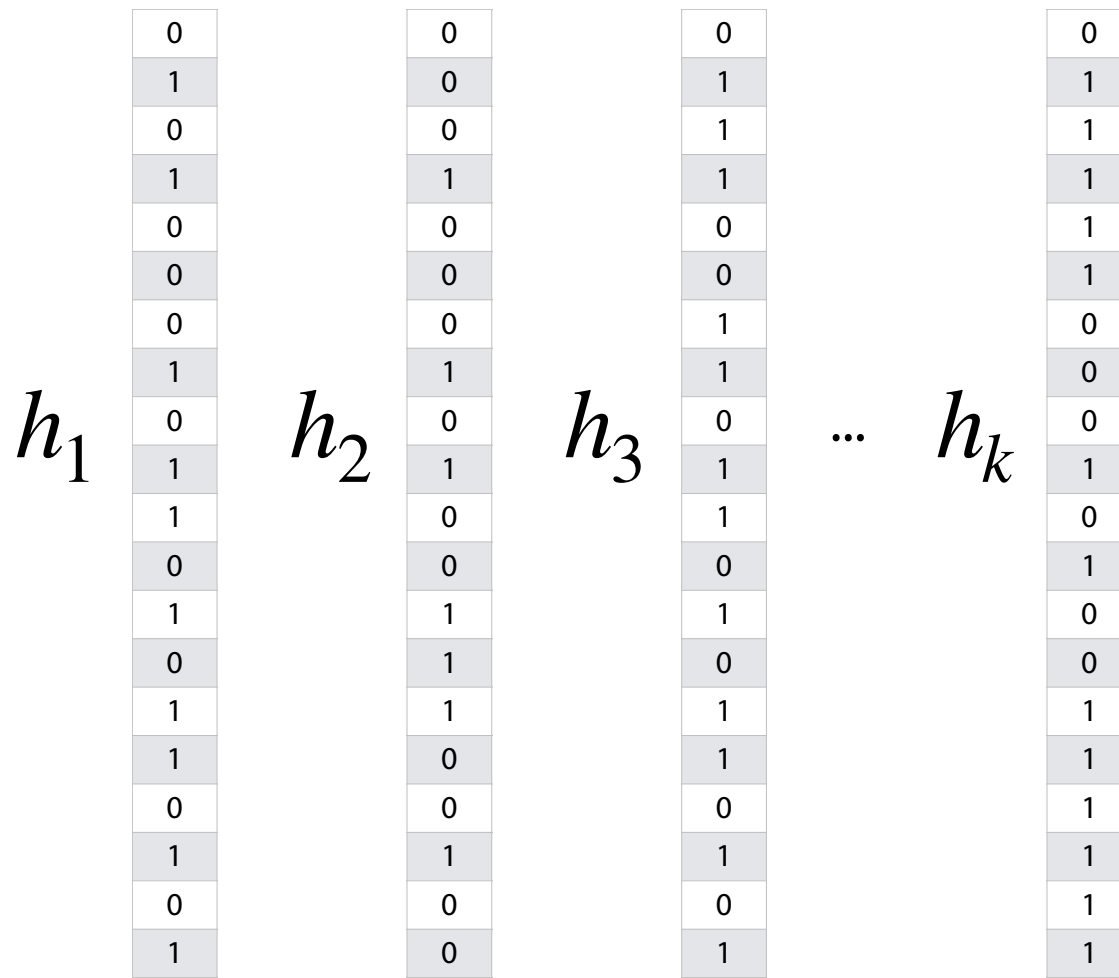
Bloom Filter: Repeated Trials

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

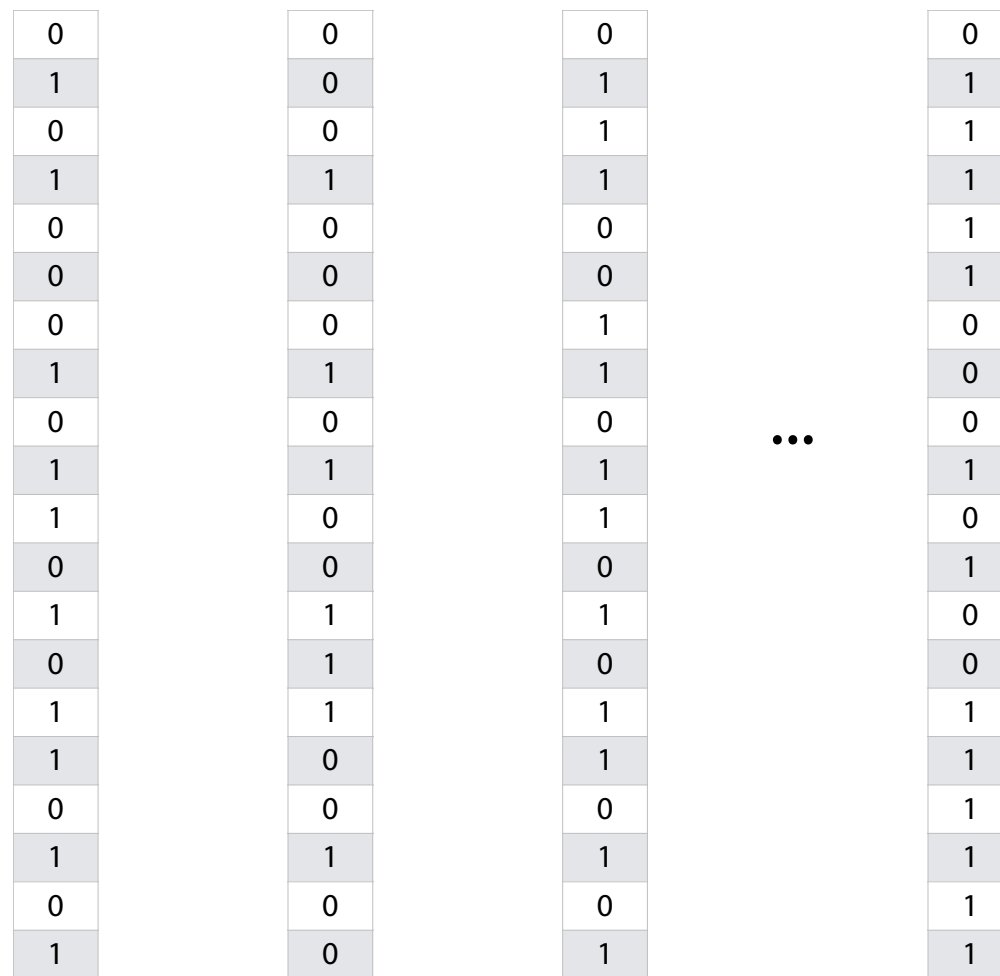


Bloom Filter: Repeated Trials

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

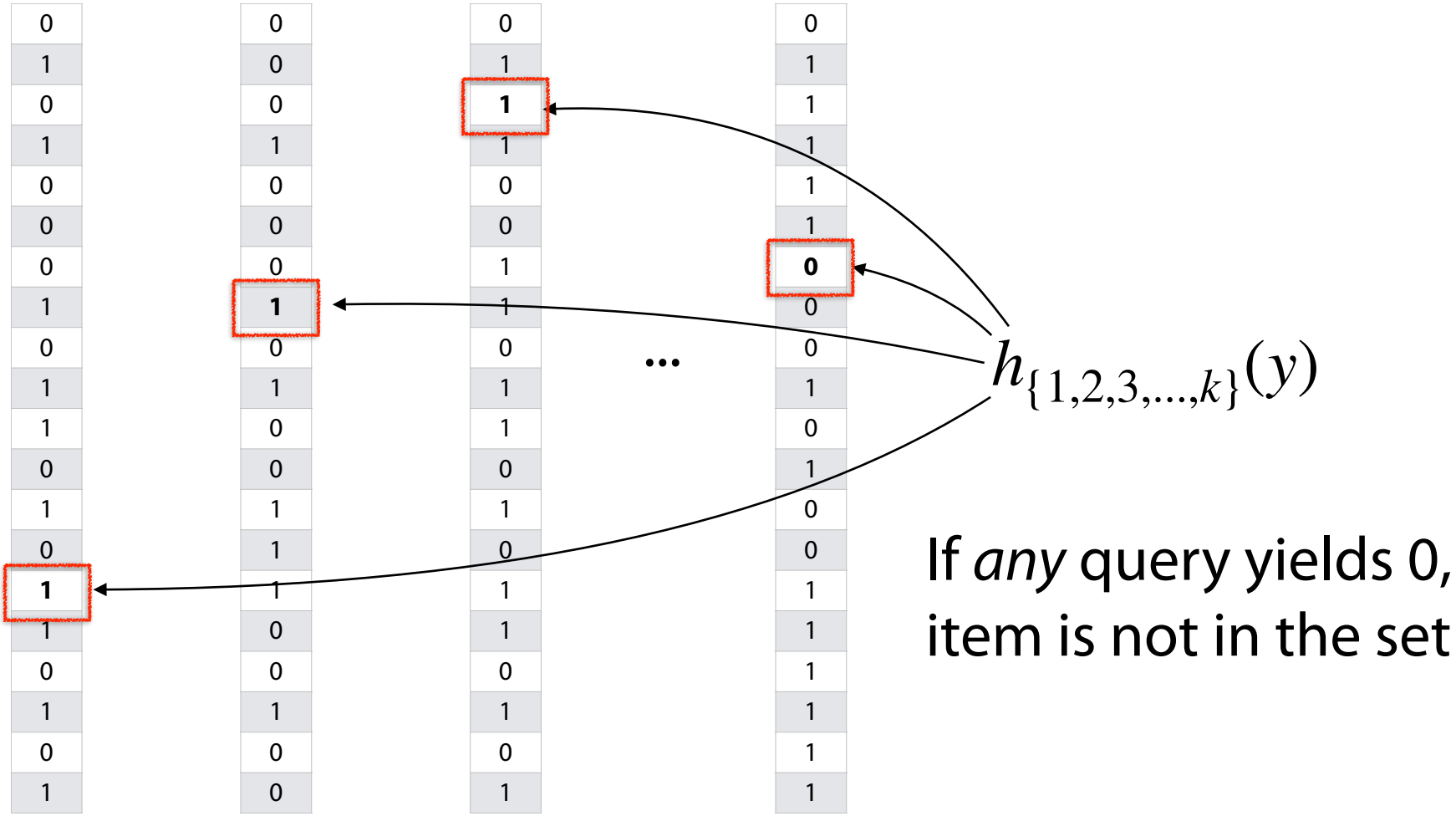


Bloom Filter: Repeated Trials

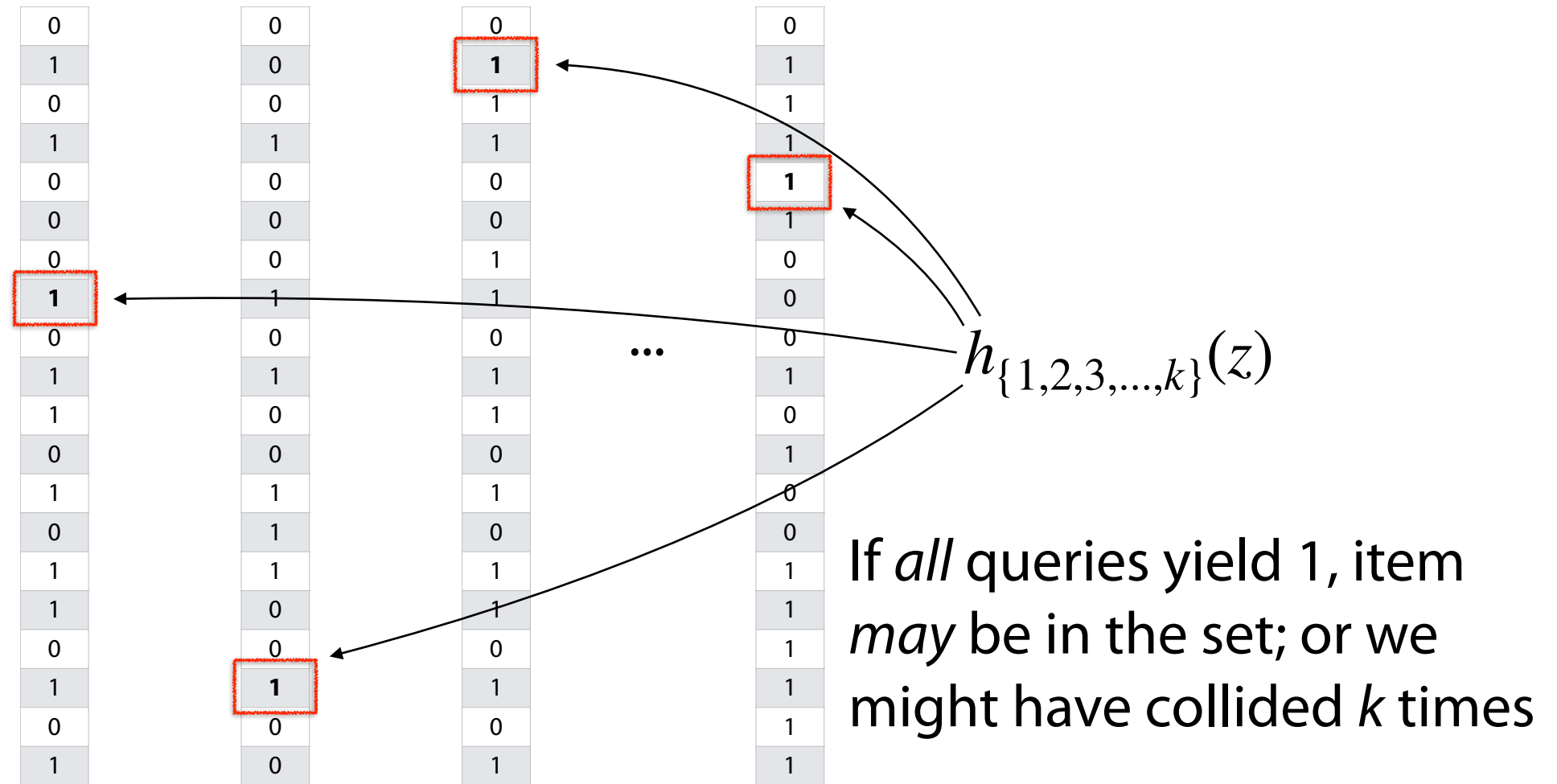


$$h_{\{1,2,3,\dots,k\}}(y)$$

Bloom Filter: Repeated Trials



Bloom Filter: Repeated Trials



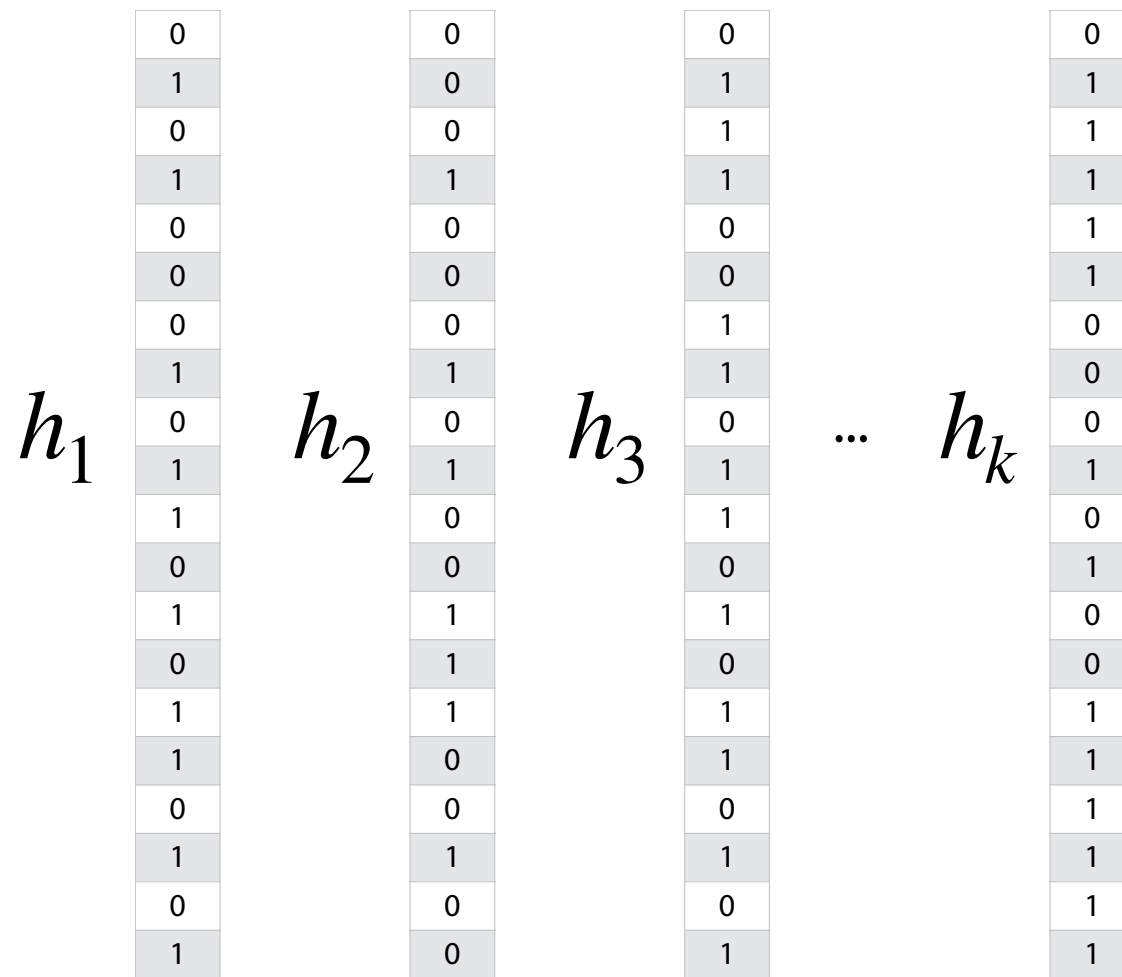
Bloom Filter: Repeated Trials

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