Rank-Indexed Hashing: A Compact Construction of Bloom Filter and its Variants*

Nan Hua[1], Haiquan (Chuck) Zhao[1], Bill Lin[2], Jun (Jim) Xu[1]

[1]College of Computing, Georgia Tech
[2]Dept of ECE, UCSD

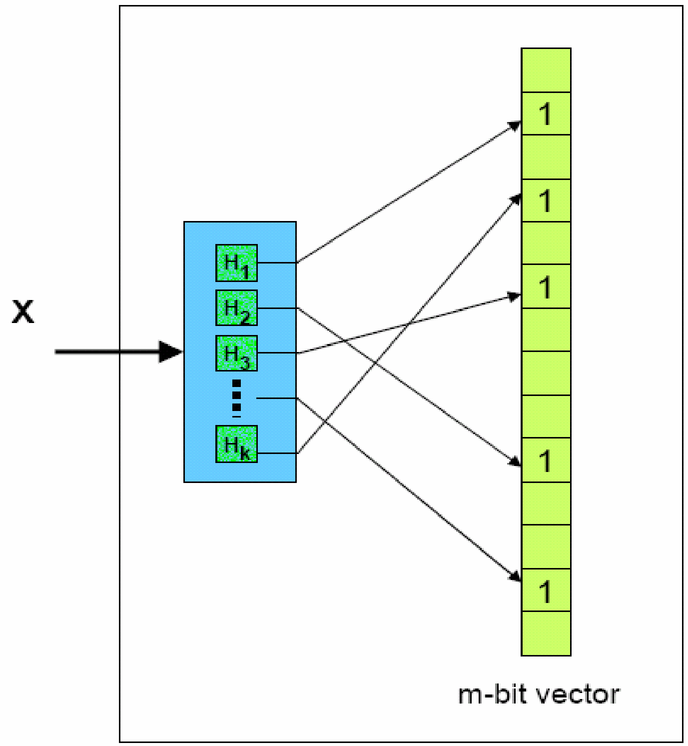
* Supported in part by CNS-0519745, CNS-0626979, CNS-0716423, CAREER Award ANI-0238315, and a gift from CISCO

Index

- What is Bloom Filter?
- Alternative Construction Way: Fingerprint Hash Table
- Scheme of Rank-Indexed Hashing
- Tail Bound Guarantee
- Numerical Results
- Conclusion

Bloom Filter (standard)

- A space-efficient data-structure to approximately represent a set and answer membership query
- *k* hash functions and an m-bit-long vector
- Insert element by setting the bits in *k* hashed locations to 1
- Answer membership query by checking the bits in the *k* hashed locations



Bloom Filter (Standard) (II)

- In one word, only approximately answers "in the set" or not, nothing else.
  - No False Negative, while False Positive is possible and predictable
  - Can't support deletion, since once bit is set, never reset
- Tradeoff of false positive rate *p*, hash functions *k*, and storage per inserted item
  - The best tradeoff for storage is achieved when

    $k=\lg(1/p)=m\ln 2/n$ (p is false positive rate)

    $1.44\lg(1/p)$ bits per item ($1.44=1/\ln 2$; lg means $\log_2$)

Variants on Bloom Filters (I)

- Support Deletion:
  - Counting Bloom Filter (CBF)
- Support Query of Associated Values
  - Associated value: such as counts, state, address, etc.
  - Spectral Bloom Filter (SBF)
  - Bloomier Filter
  - Approximate Concurrent State Machines (ACSM) (based on d-left Counting Bloom Filter, dlCBF)

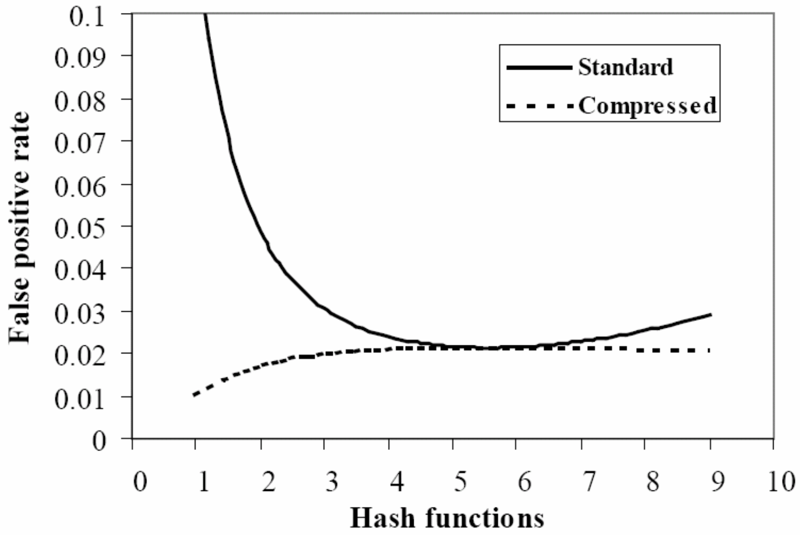
Variants on Bloom Filters (II)

- Compressed Bloom Filter
  - optimized for best compressed size
  - The optimal point of standard Bloom Filter is also the worst point of compressed Bloom Filter:

    1.44lg(1/p) bits per item
  - The limit is of compressed Bloom Filter

    lg(1/p) bits per item

    (also the information theory limit)
  - **Compressed Bloom Filter trade original (uncompressed) size for compressed size.**



Application of Bloom Filter and Variants

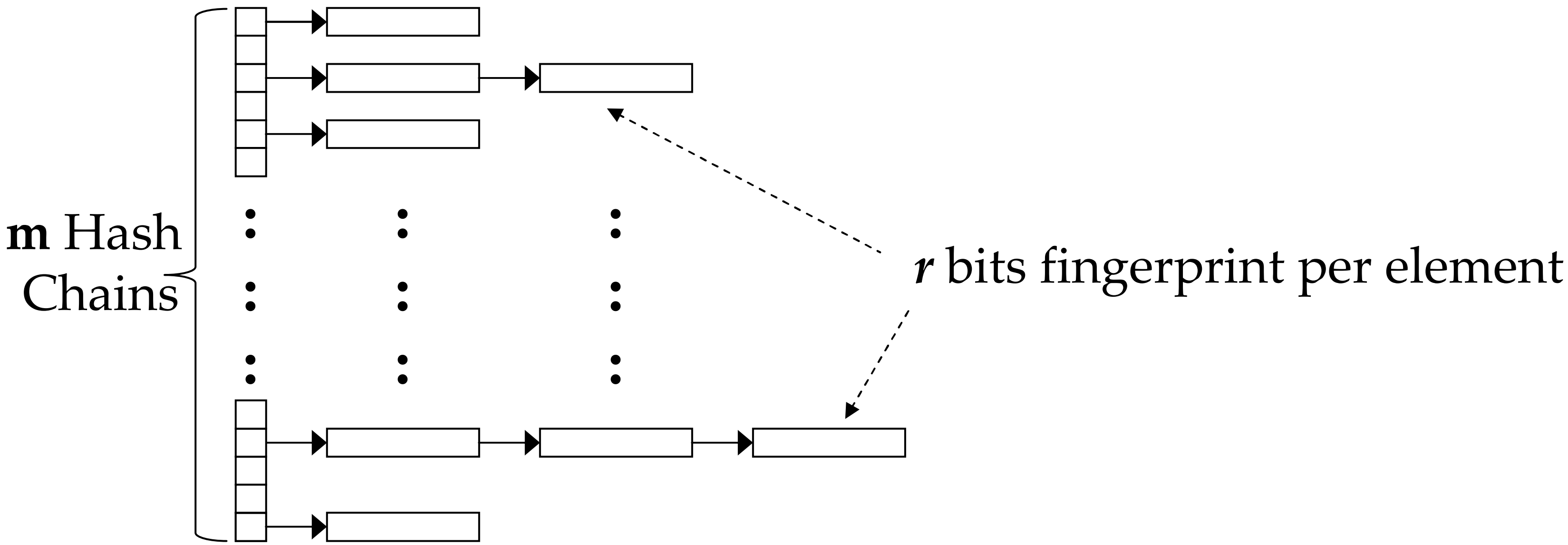
- Although discovered in database area, widely used in networking
  - IP Lookup & Packet Classification
  - Hash table acceleration
  - Pattern Matching & Deep Packet Inspection
  - P2P File Sharing
  - Routing Protocols Design
  - …
- Design consideration and optimization targets
  - Space (bloom filters are always small and on-chipp-able)
  - Query speed (HW/SW? Need deterministic query time bound or flexible?)
  - Update speed (static, quasi-static or fast)
  - Support of extra functions

An alternative way to build "Bloom Filter": Fingerprint Hash Table (I)

- The weakness of standard Bloom Filter:
  - the best storage of Bloom Filter is 1.44lg(1/p) bits per element, still far from the limit, lg(1/p).
  - Not easy to support deletions and associated values
- We could revisit "Bloom filter" problem, i.e. approx. representation of a set, by another way: fingerprint hash table
  - Make fingerprint for each element (by hash) and store it in a hash table

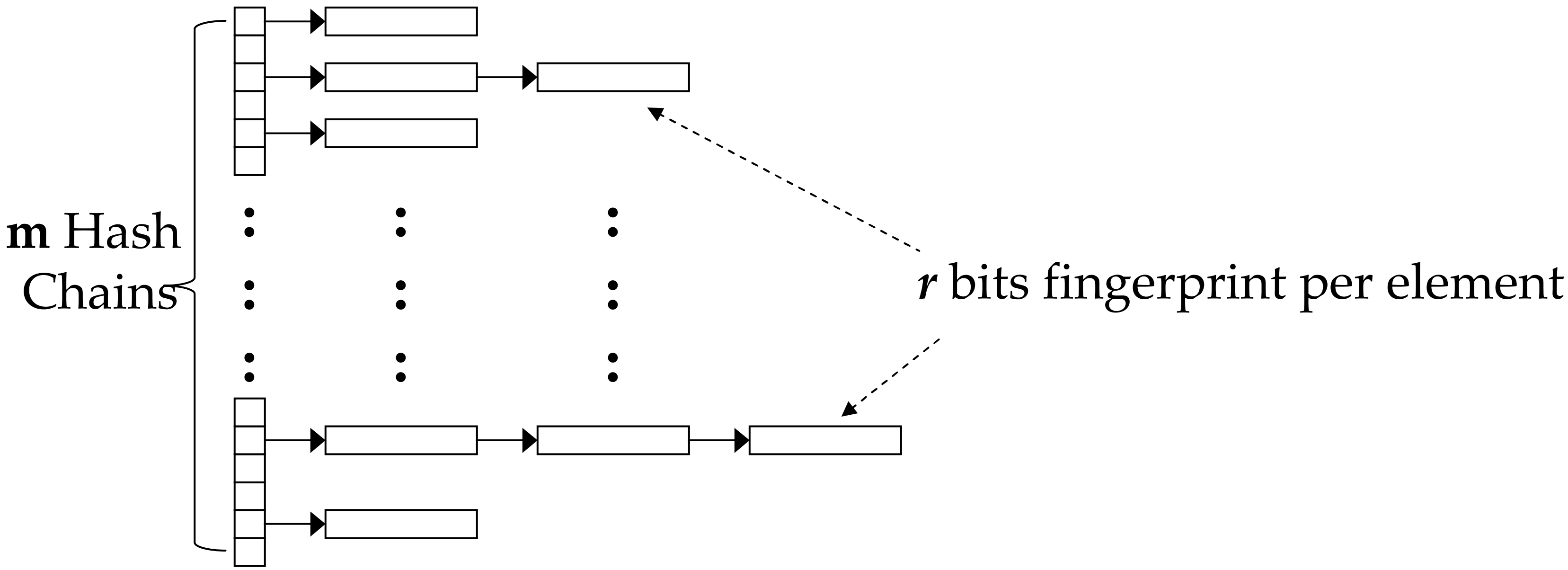
An alternative way to build Bloom Filter: Fingerprint Hash Table (II)

- For each incoming element x, get location d(x) and r-bit-long fingerprint r(x) ([0,2^r-1]) by hash function.
- Store the r-bit-long fingerprint r(x) in the d(x)th hash chain.
- To test whether x is in the set, simply check all fingerprints in the d(x)th hash chain to find r(x)
- Naturally support deletion and query of associated values



An alternative way to build Bloom Filter: Fingerprint Hash Table (III)

- Suppose n items inserted, false positive rate p would be only $(n/m)2^{-r}$
- When n=m, if only counting the bits for fingerprint, only lg(1/p) bits per inserted item is needed, which is the lower bound by information theory!
- We haven't included the chaining cost…

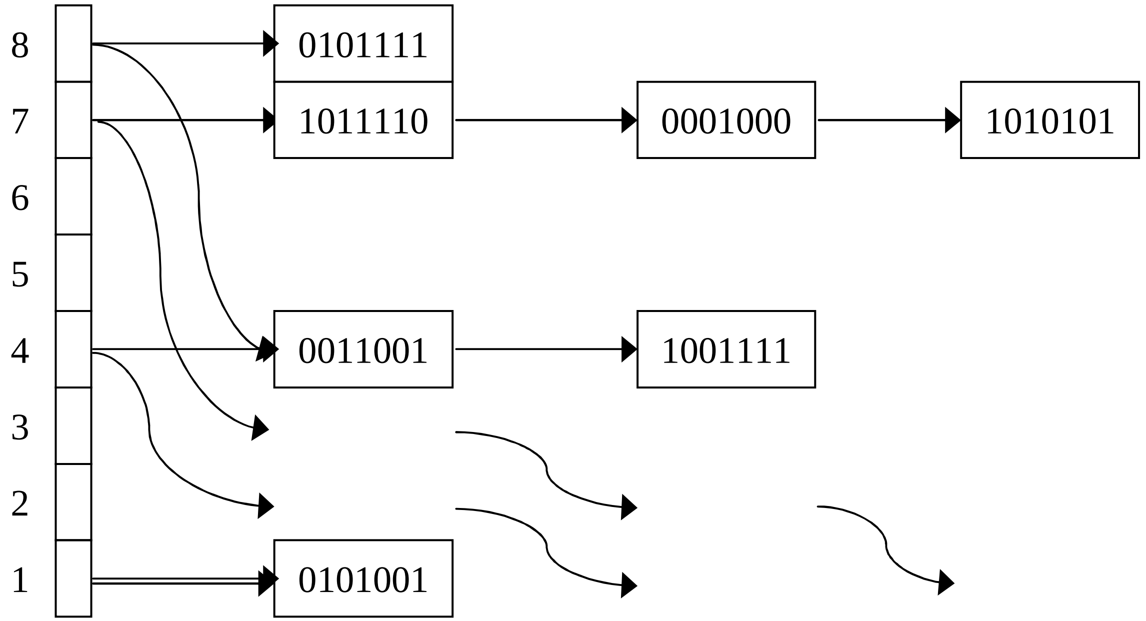


An alternative way to build Bloom Filter: Fingerprint Hash Table (III)

- However, the indexing cost for chaining is very high, if using conventional hash table techniques
  - Suppose using pointers, at least another lg(n) bits per item are needed.
  - Memory organization is hard
  - Some well-known hash table constructions are unsuitable here, such as Linear Probe (cann't know which chain is in during query, while the information of d(x) is also important for fingerprint hash table)
- In nature, dl-CBF is using multiple-choices hash table to lower the chaining cost
- We use Rank-Indexing technique to lower chaining cost

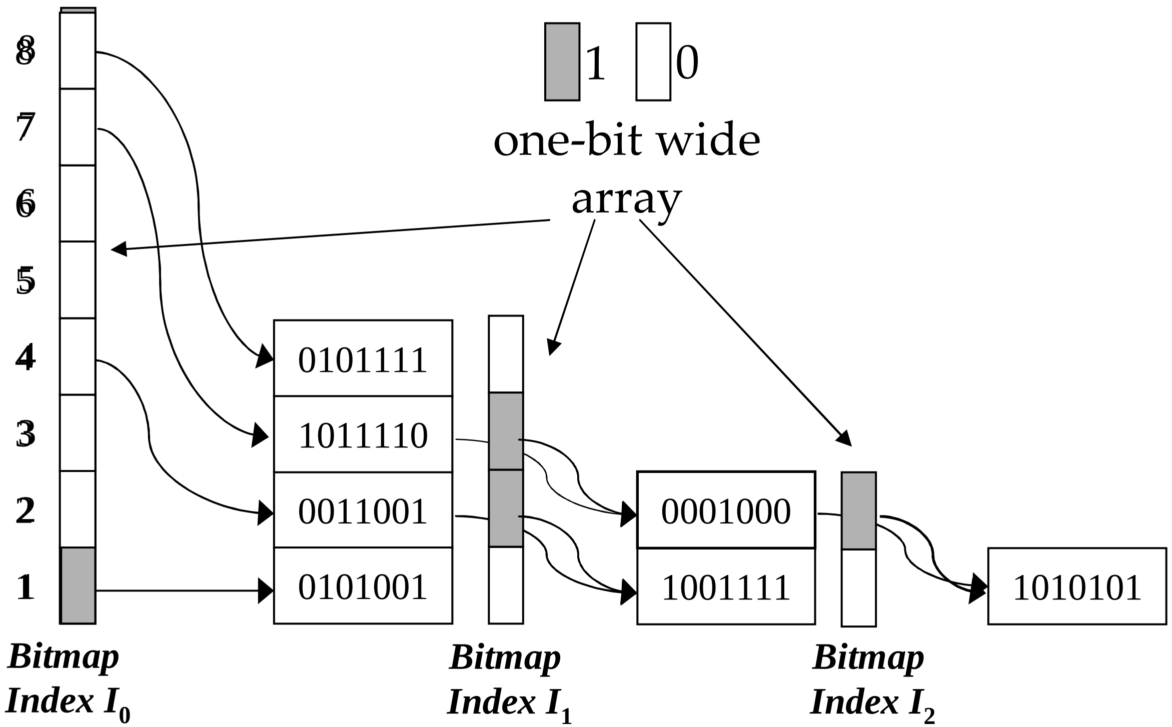
Our approach– Rank Indexing

From a horizontal view, hash chain is very fluctuating.
However, we could pack them together from a vertical view.
Then, how to link??!!



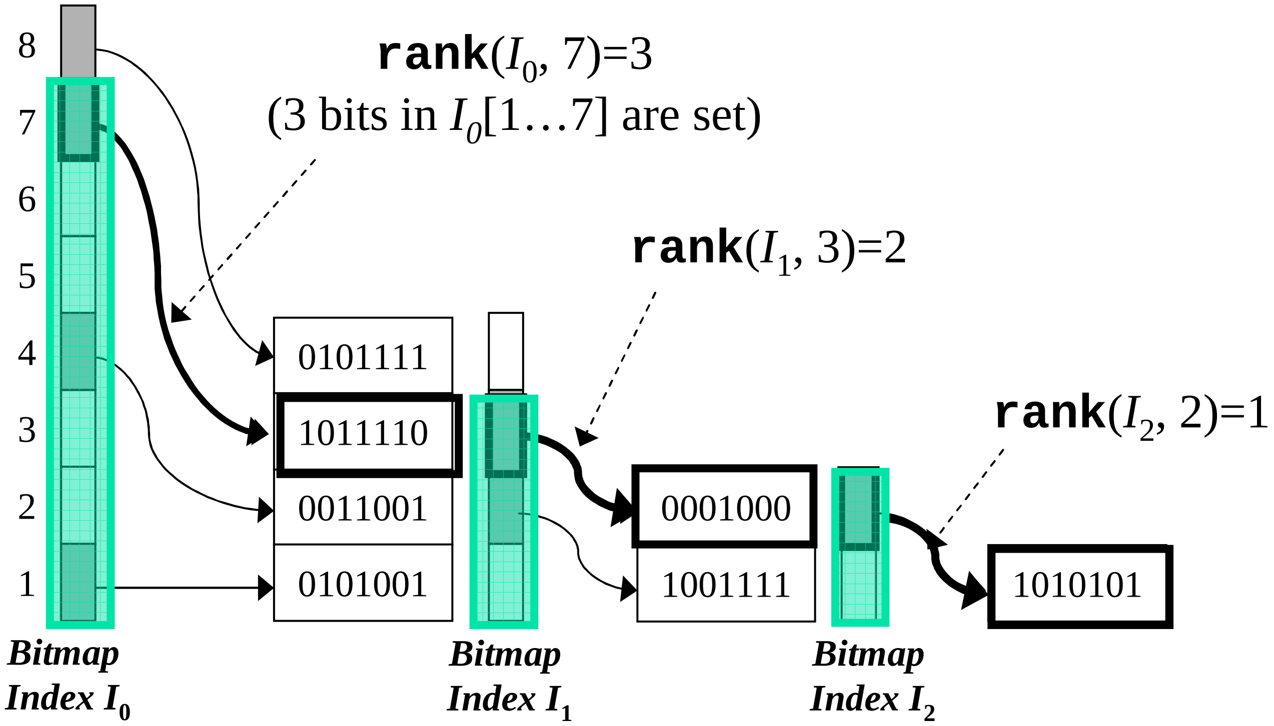
Rank Indexing (II)

- The link is established by "rank indexing"
  - Named by "Dictionaries using variable-length keys and data, with applications" [SODA05]
  - Also called bitmap technique in [Varghese04]



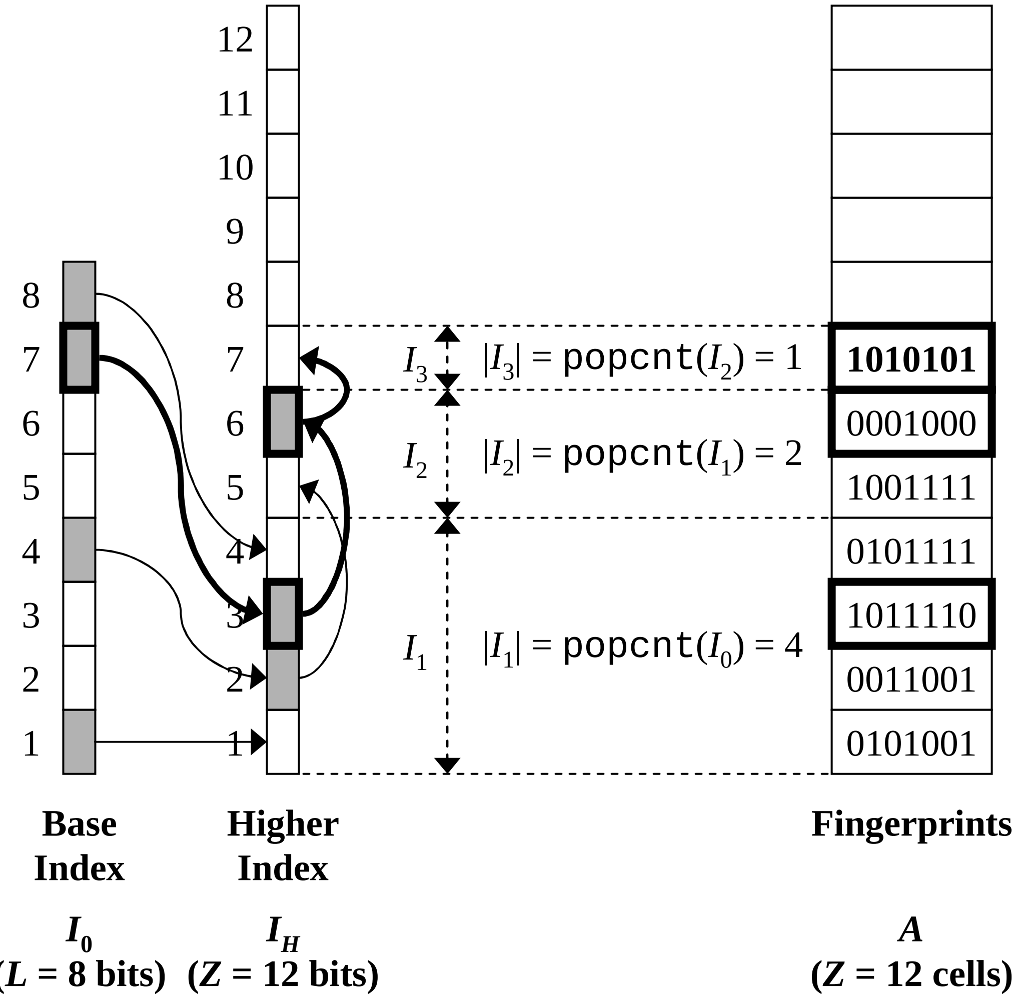
Rank Indexing (III)

- How Rank Indexing works?
  - The location of the linked element is calculated by "rank" operation
  - **rank**(A, b) "popcount" the bits in A[1…b]



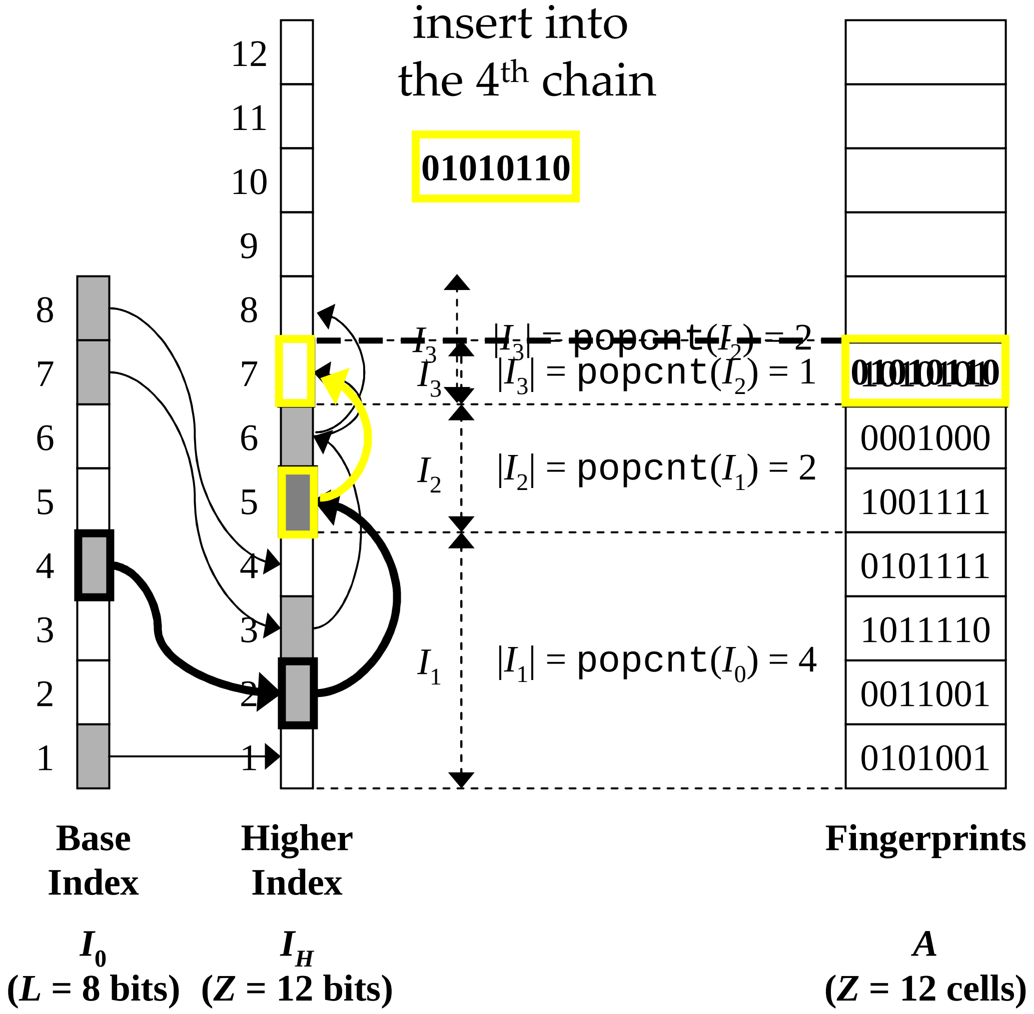
Rank Indexing (IV)

- We could even pack these I_1, I_2, I_3… together, since the size of I_1, I_2, I_3, etc, are calculable.



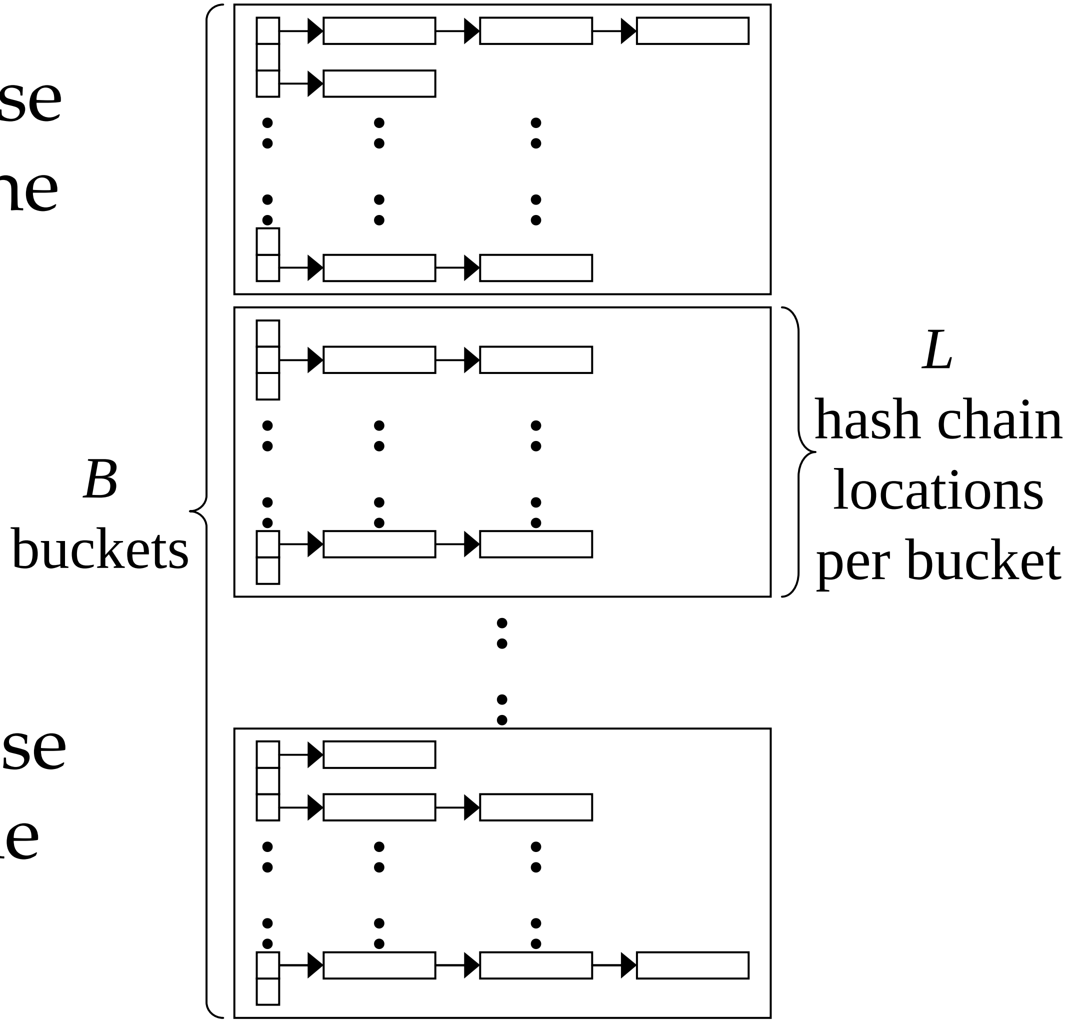
Rank Indexing (V)

- To insert one fingerprint,
  - Bit shift of the bitmap index
  - Bit shift of the fingerprints



Bucket is needed

- We can't directly use rank indexing on the whole fingerprint hash table
  - Too large bitmap
- We could group hash chains into buckets and only use rank indexing in the "local" bucket.

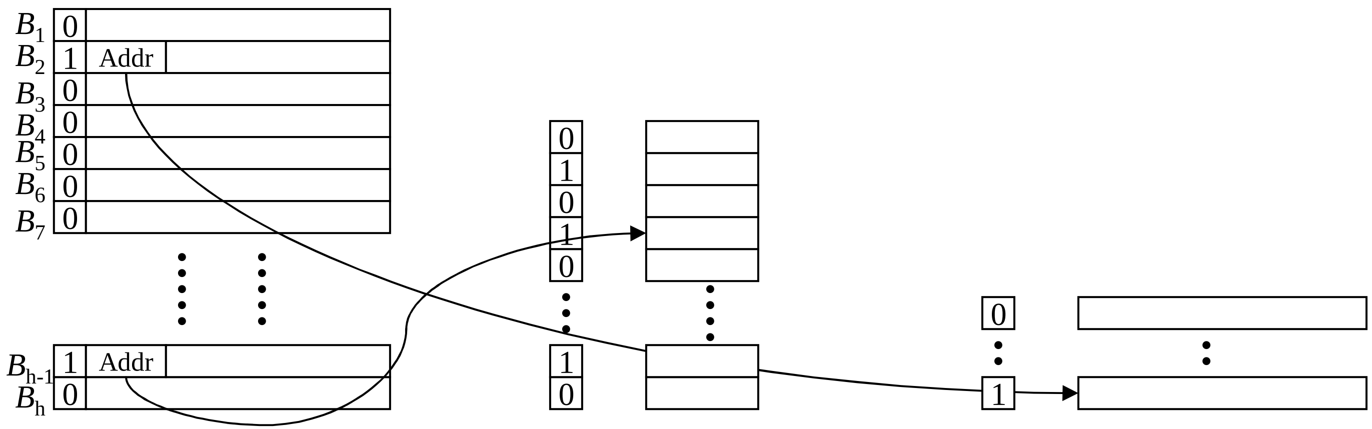


Bucket Overflow

- There is some probability that bucket might overflow!
  - X: fingerprints inserted into bucket
  - Z: the number of entries pre-allocated in each bucket

Threshold Z	Overflow (X>Z) Probability (When E[X]=56)
Z=64	0.13
Z=74	0.01
Z=128	8e-17

Handle Bucket Overflow



Array of Normal Buckets (B Buckets)

Array of 2[nd] Level Bucket Extensions (say, around 0.1B buckets)

Array of 3[rd] Level Bucket Extensions (very few, say, 10)

- The number of 2nd-level and 3nd-level bucket extensions are defined as J_2 and J_3.

Tail Bound (I)

- Random Variable $X_i^{(n)}$ denotes the number of fingerprints inserted in i^{th} bucket (when n fingerprints inserted in total)
- The probability that 2nd-level bucket extensions are insufficient is

$$\Pr[O_2 > J_2],\ where\ O_2 \equiv \sum_{i=1}^{B} 1_{\{X_i^{(n)} > Z_1\}}$$

- The probability P_o that the bucket extensions are insufficient is bound by:

$$P_o \le \sum_{l=2}^{4} Pr[O_l > J_l],\ where\ J_4 \equiv 0, O_l = \sum_{i=1}^{B} 1_{\{X_i^{(n)} > Z_{l-1}\}}$$

Tail Bound (II)

- The hardest point of the proof:
  Random Variables $X_i^{(n)}$, i=1….B, are weakly correlated
- We could construct Random Variables $Y_i^{(n)}$, i=1….B， which is **i.i.d** random variables with distribution *Poisson(n/m)* .
- From [MM05], it could be proved that:

$$E[f(X_1^{(n)},\dots,X_m^{(n)})] \le 2E[f(Y_1^{(n)},\dots,Y_m^{(n)})]$$

where f is an nonnegative and increasing function.

- Then, we could use the following increasing indicator function to get bound for Pr[O>J]

$$f(x_1,\dots,x_B) = 1_{\{\left(\sum_{i=1}^{B} 1_{x_i > W}\right) > J\}}$$

- We could get: $Pr[O > J] \le 2Binotail(B, Poissontail(\frac{n}{B}, Z), J)$

Numerical Results (I) ----Parameter Configuration

- Typical Configuration of Parameters

REPRESENTATIVE RESULTS FOR RANK-INDEXED HASHING UNDER VARIOUS PARAMETER SETTINGS ($P_o = 10^{-10}$, $n = 10^5$).

ϵ	λ	R	L	Z_1	Z_2	Z_3	J_2/B	J_3/B
1%	0.64	6 bits	60	45	8	45	17.9%	2.7%
0.1%	0.92	10 bits	64	63	17	50	36.0%	1.7%
0.01%	0.86	13 bits	61	59	13	48	23.3%	1.8%

Numerical Results (II) ----Storage

- Compared with standard Bloom Filter

ϵ	standard	d-left	Rank	Comparison	
				vs. standard	*vs.* d-left
1%	9.6	15.0	10.6	$+10.1\%$	-29.5%
0.1%	14.6	18.0	14.4	-1.1%	-26.3%
0.01%	19.1	22.2	18.2	-4.4%	-23.2%

- Compared with Counting Bloom Filter

ϵ	standard CBF	d-left CBF	Rank CBF	Comparison	
				vs. standard	*vs.* d-left
1%	38.3	17.6	13.0	-66%	-27%
0.1%	58.4	22.3	16.8	-71%	-24%
0.01%	76.3	26.4	20.6	-73%	-22%

Numerical Results (III) ----for transferring Purpose

- When used for transferring (P2P sharing, etc), no need to transfer the pre-allocated vacant entries
- The bits per item would be only *1+m/n+lg(1/p)*

ϵ	Scheme	Packed Size	Uncompressed Compression BF[1]	Compression ratio[2]
1%	Rank	8.6	30	89%
1%	*d*-left	11.7	$--^3$	122%
0.1%	Rank	12.1	150	83%
0.1%	*d*-left	15.2	$--^3$	104%
0.01%	Rank	15.2	1.3×10^3	79%
0.01%	*d*-left	18.3	27	96%

[1]To achieve the same packed size, how much big original array size is needed by a Compressed Bloom Filter
(Note that Compressed Bloom Filter trade original size for compressed size)

Conclusion

- A compact construction of Bloom Filters and Variants
  - Based on popcnt operation,which is supported by modern processors
  - Very Memory-Efficient
  - Sharing all advantages of fingerprint hash table approaches
    - Support deletions, support query of associated value
  - Also great to substitute Compressed Bloom Filter
- Disadvantages
  - The access time is not deterministic (due to the chaining), although the expected access time is very good
- Related work that have been done
  - Combination with d-left Counting Bloom Filter (Rank-Indexed d-left)
    - the storage performance is even better
    - A little more memory accesses
  - Using similar techniques in statistical counters (ANCS'08)

Q&A

- nanhua@gatech.edu
- Thanks!