Near Space-Optimal Perfect Hashing Algorithms

Nivio Ziviani

Fabiano C. Botelho

Department of Computer Science
Federal University of Minas Gerais, Brazil

23rd Annual ACM Symposium on Applied Computing
Fortaleza, Brazil, March 19, 2008

Objective of the Presentation

Present two perfect hashing algorithms:

- Internal memory based algorithm
- External memory based algorithm

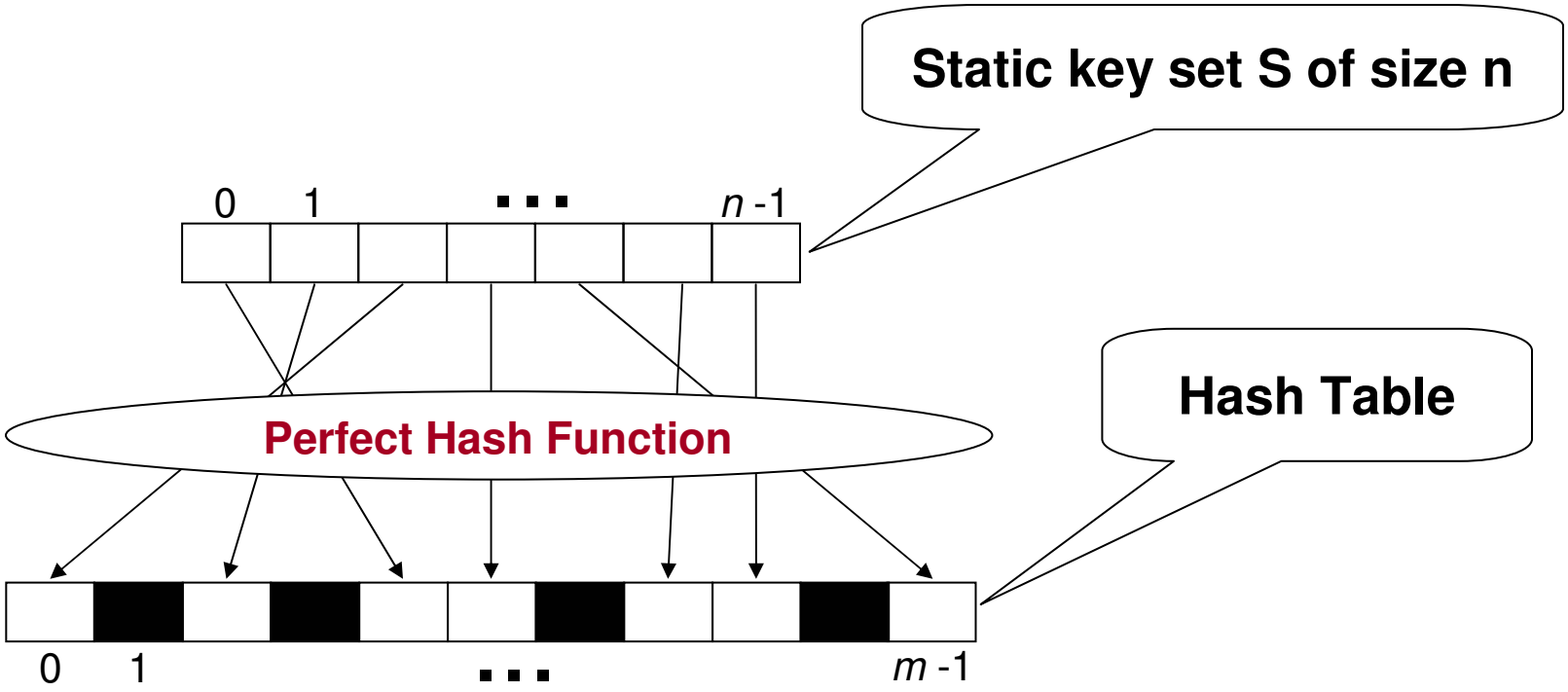
Objective of the Presentation

Present two perfect hashing algorithms:

- Internal memory based algorithm
- External memory based algorithm

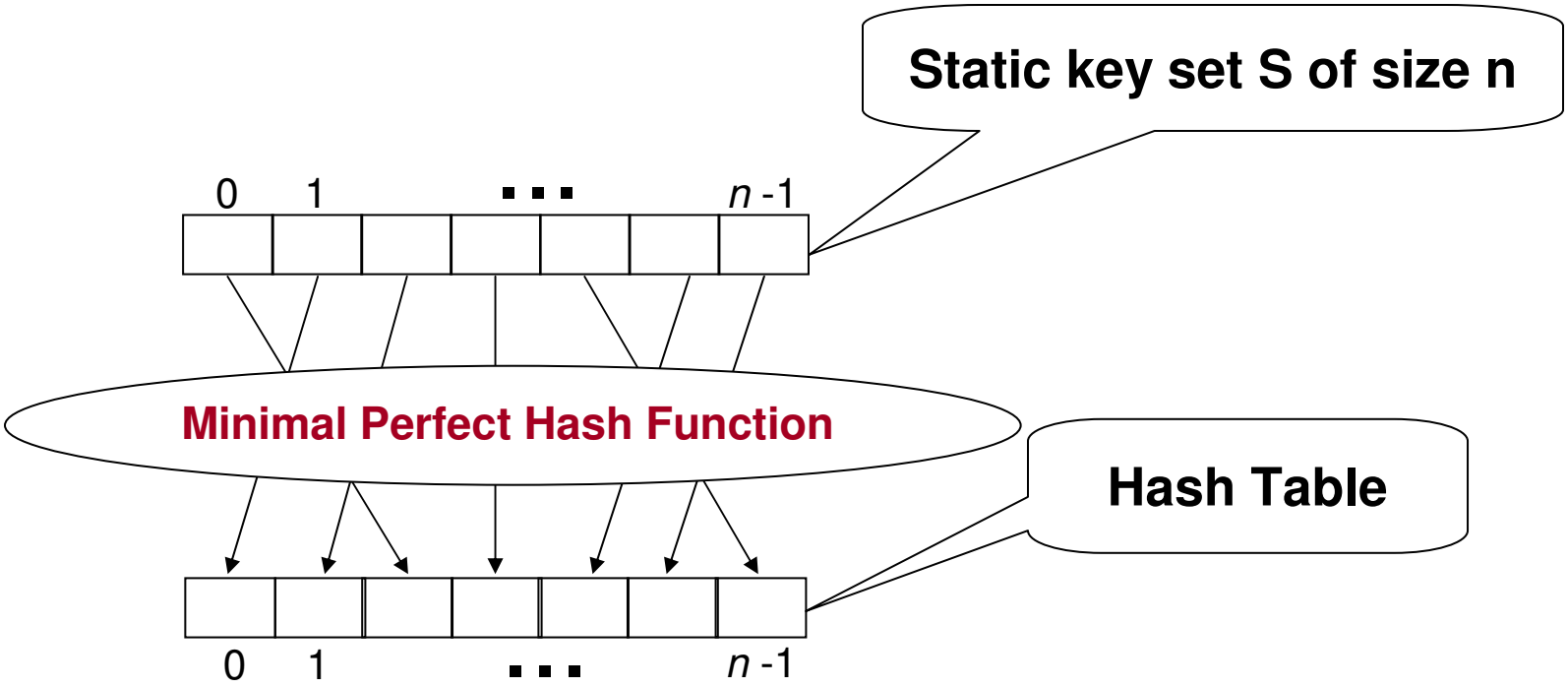
Both algorithms are practical, time efficient and near space-optimal

Perfect Hash Function



$$S \subseteq U, \text{where} |U| = u$$

Minimal Perfect Hash Function

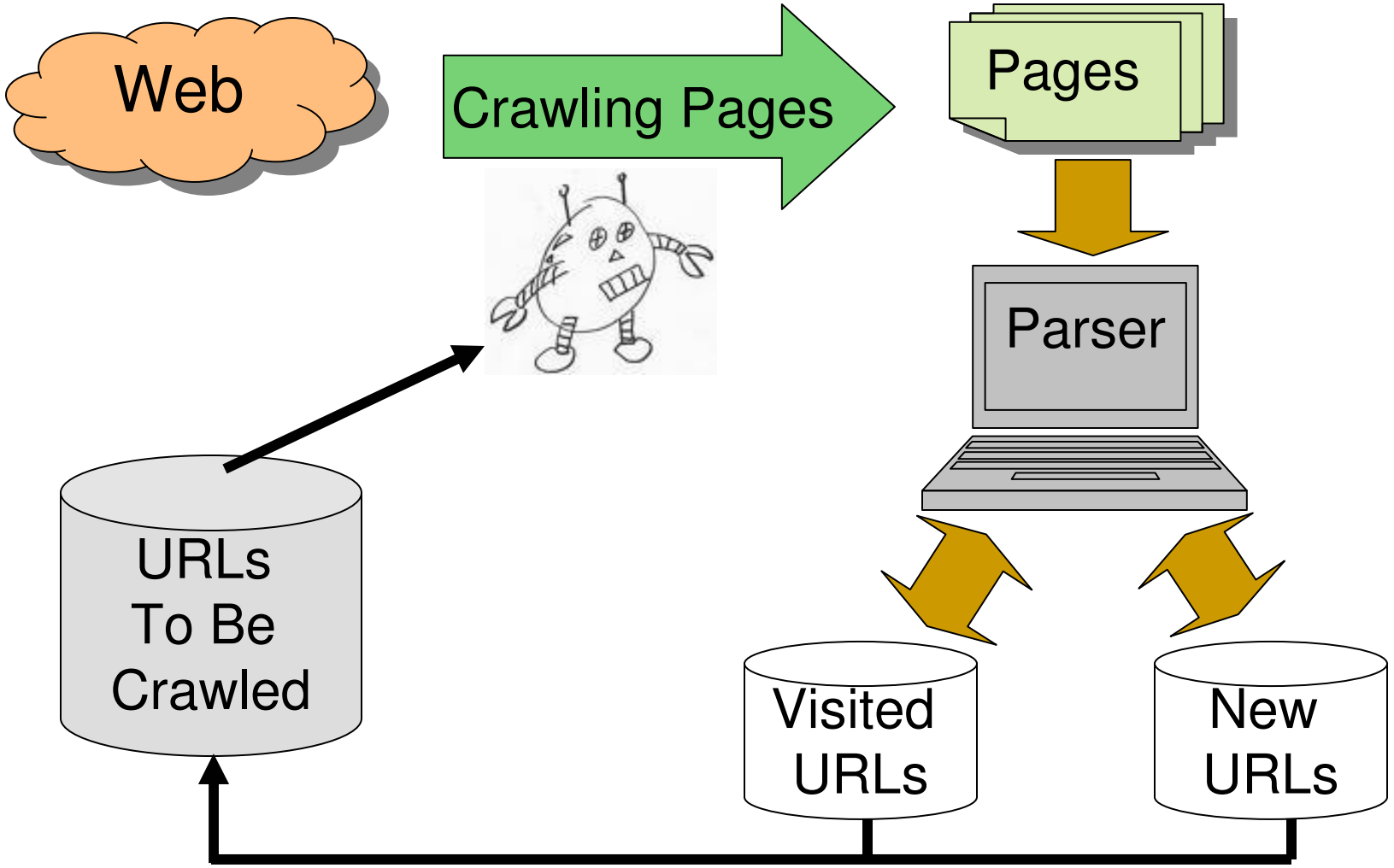


$$S \subseteq U, \text{ where} |U| = u$$

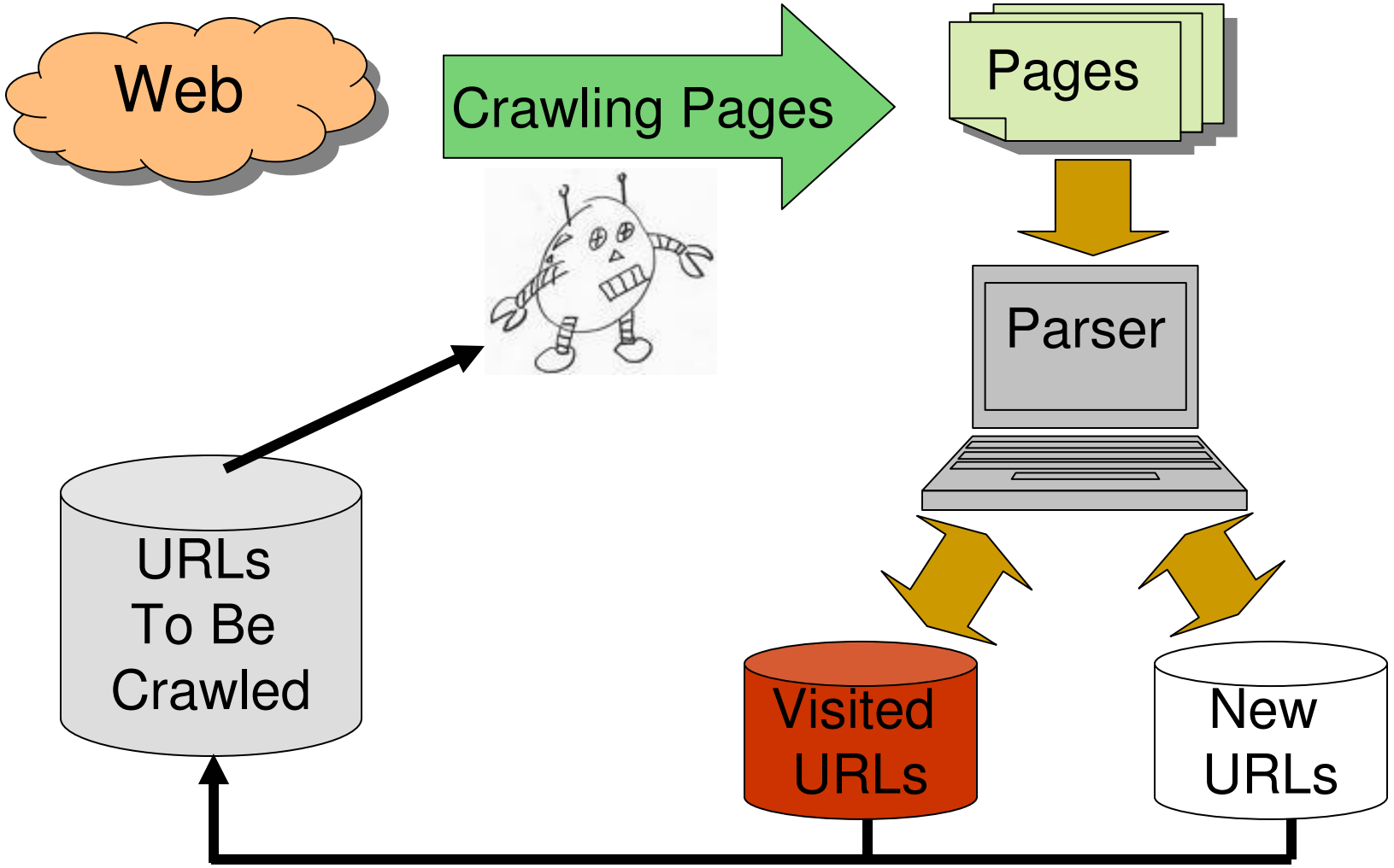
Where to use a PHF or a MPHF?

- Ubiquitous in Computer Science: access items based on the value of a key
- Work with huge static item sets:
  - In data warehousing applications:
    - On-Line Analytical Processing (OLAP) applications
  - In Web search engines:
    - Represent the set of visited URLs when the Web is beeing crawled
    - Large vocabularies
    - Map long URLs in smaller integer numbers that are used as IDs

Crawling


Web
Crawling Pages
Pages
Parser
URLs
To Be
Crawled
Visited
URLs
New
URLs

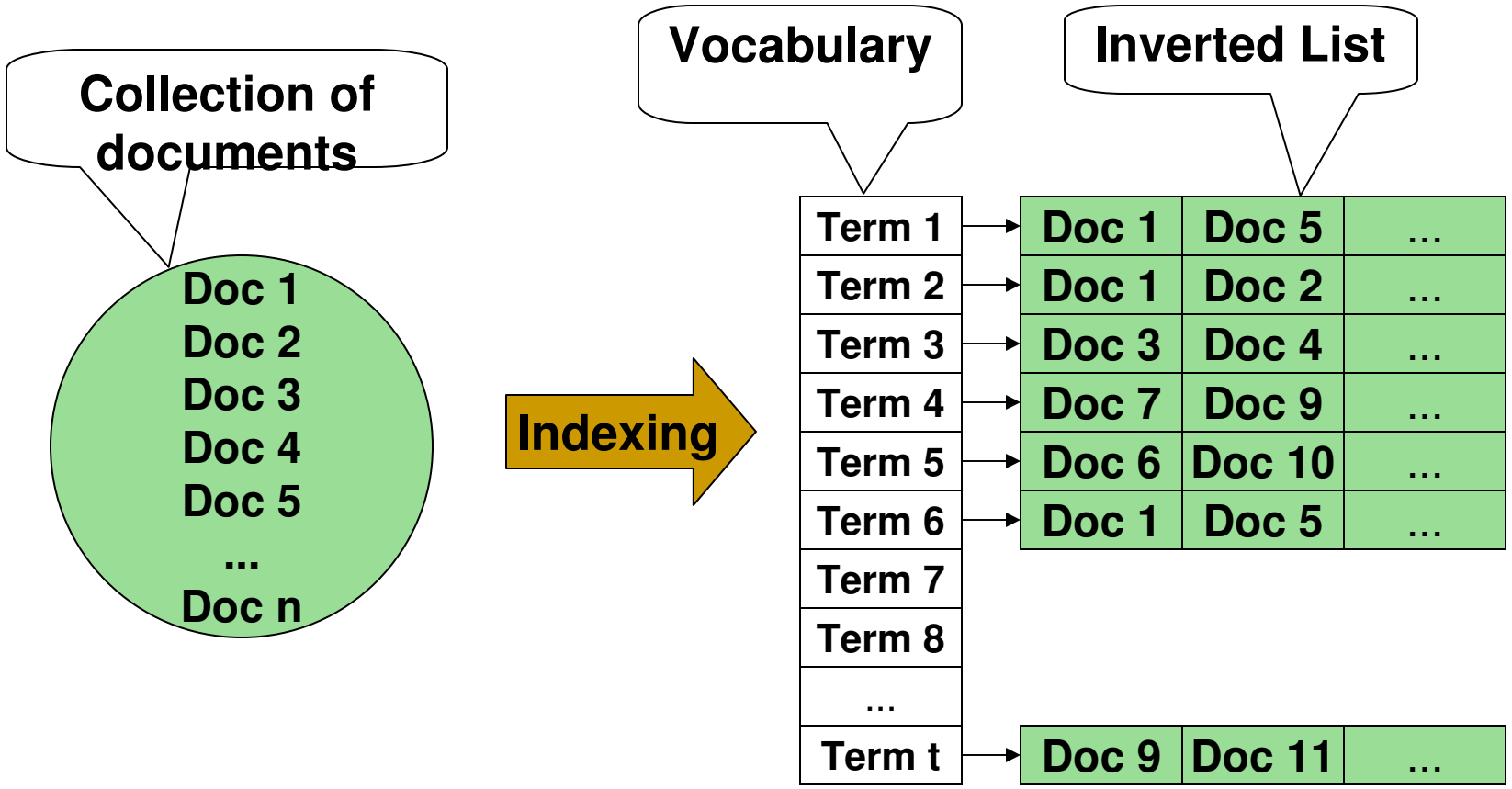
Crawling


Web
Crawling Pages
Pages
Parser
URLs
To Be
Crawled
Visited
URLs
New
URLs

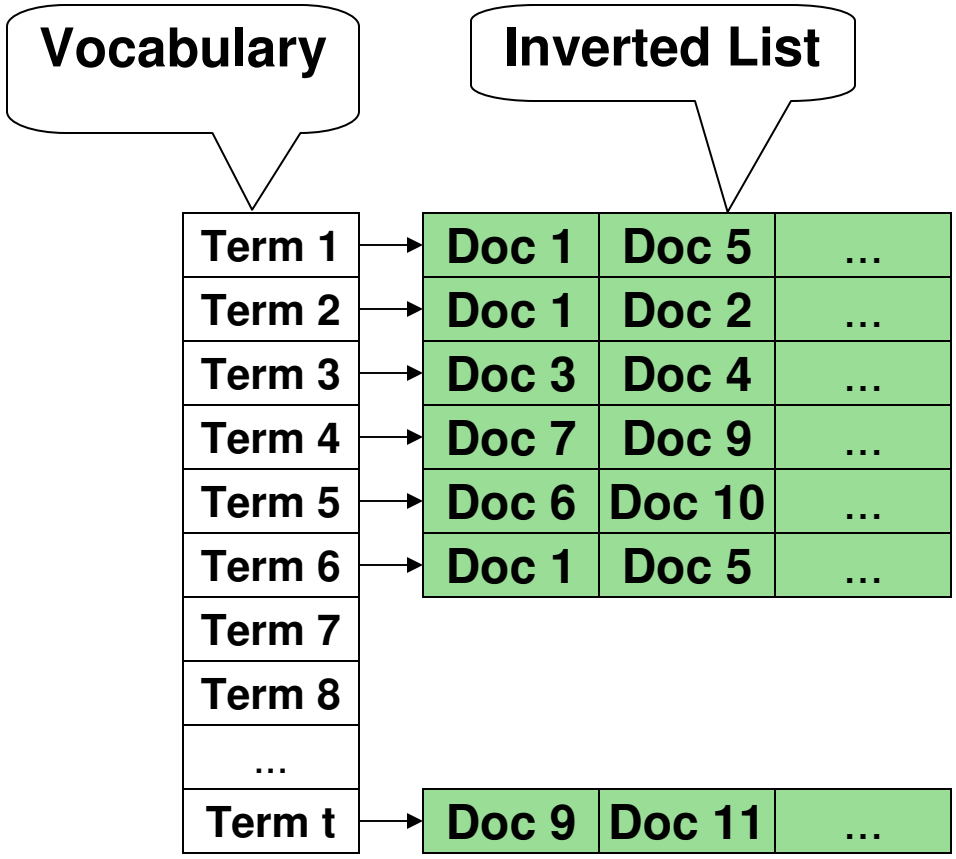
Representing Visited URLS

- MPHFs is the most compact way of representing the set of visited URLs
- Enable to keep much more URLs in main memory of each machine
- When the set of new URLs becomes large, a new MPHF is generated for the whole set of URLs

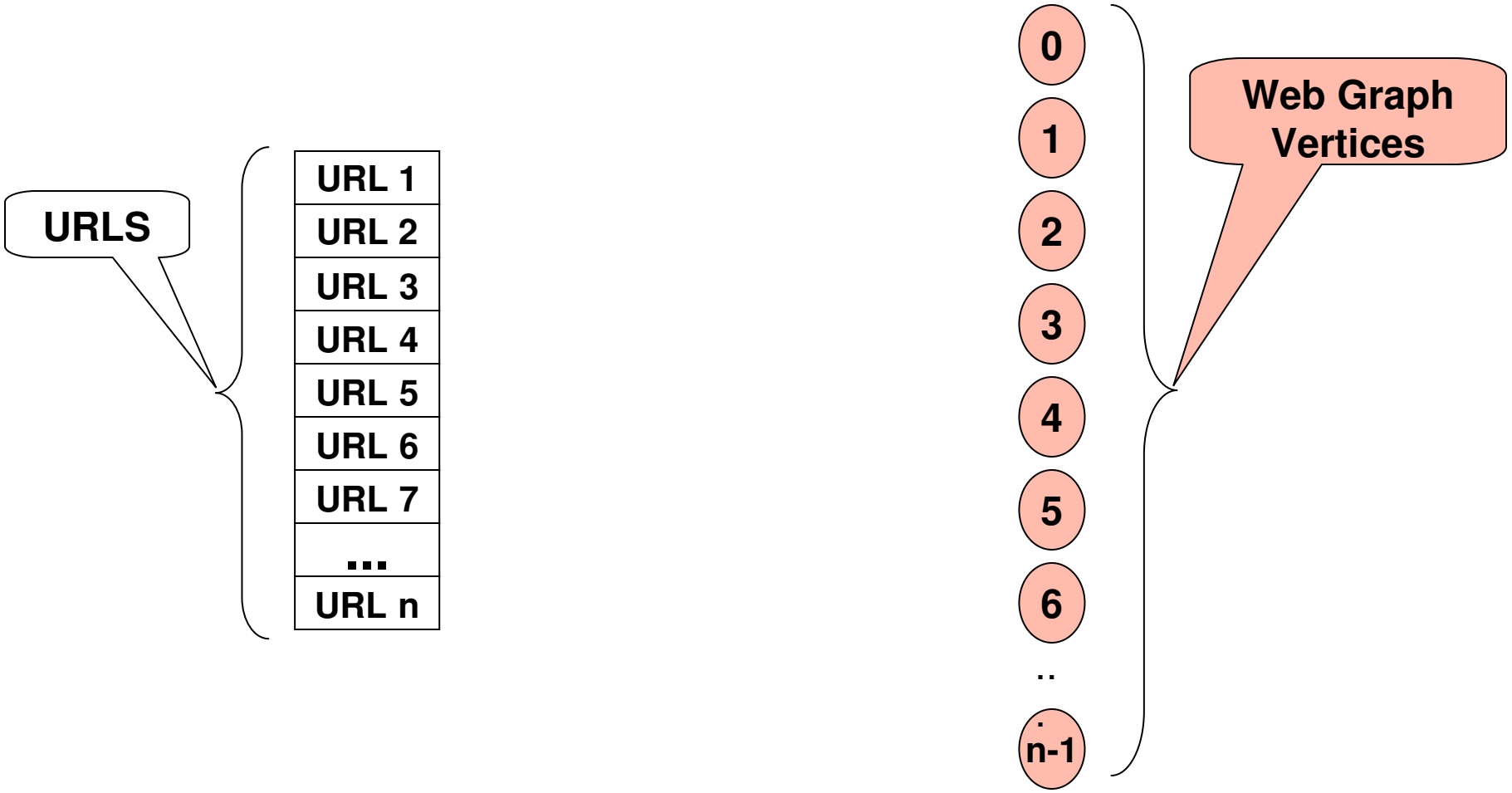
Indexing


Collection of documents
Doc 1
Doc 2
Doc 3
Doc 4
Doc 5
...
Doc n
Indexing
Vocabulary
Inverted List
Term 1
Term 2
Term 3
Term 4
Term 5
Term 6
Term 7
Term 8
...
Term t
Doc 1 Doc 5 ...
Doc 1 Doc 2 ...
Doc 3 Doc 4 ...
Doc 7 Doc 9 ...
Doc 6 Doc 10 ...
Doc 1 Doc 5 ...
Doc 9 Doc 11 ...

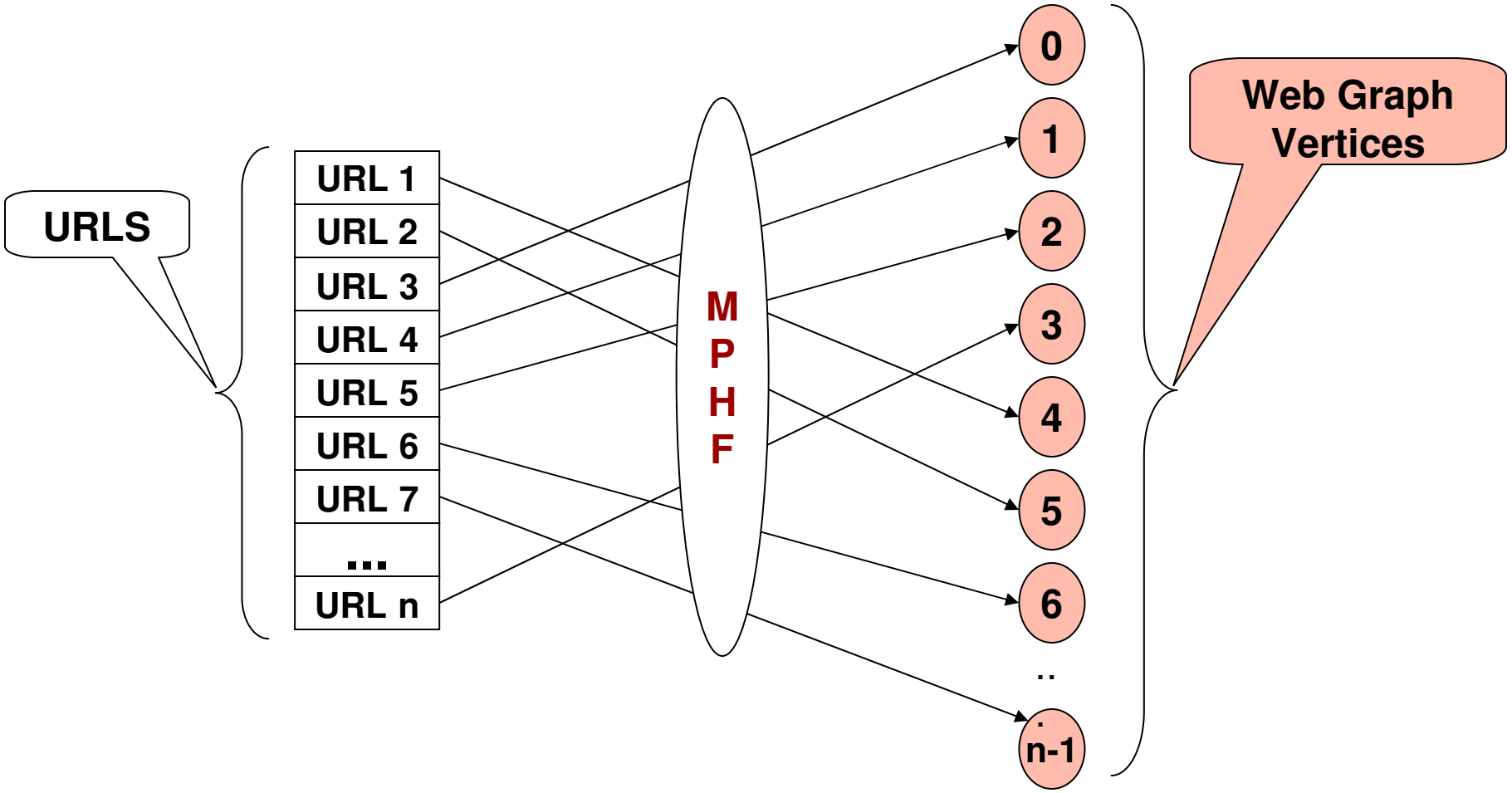
Representing the Vocabulary



Mapping URLs to Web Graph Vertices



Mapping URLs to Web Graph Vertices



Information Theoretical Lower Bounds for Storage Space

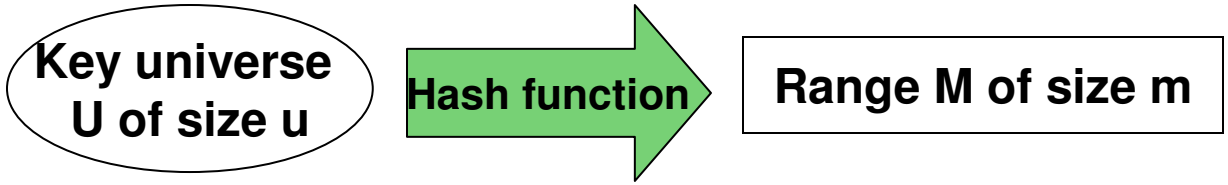
- PHFs (m ≈ n): $\text{Storage Space} \geq \frac{n^2}{m} \log e$

- MPHFs (m = n): $\text{Storage Space} \geq n \log e$

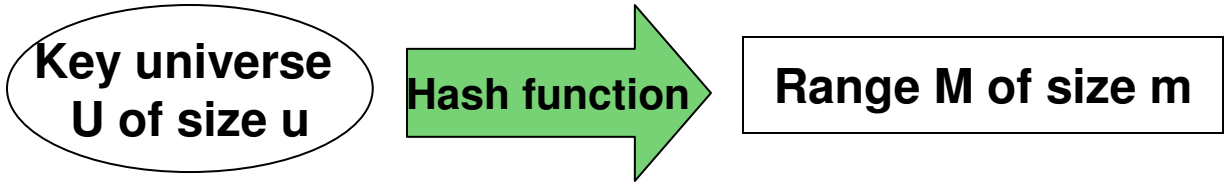
m < 3n

$\log e \approx 1.4427$

Uniform Hashing Versus Universal Hashing



Uniform Hashing Versus Universal Hashing



Uniform hashing

- # of functions from U to M?

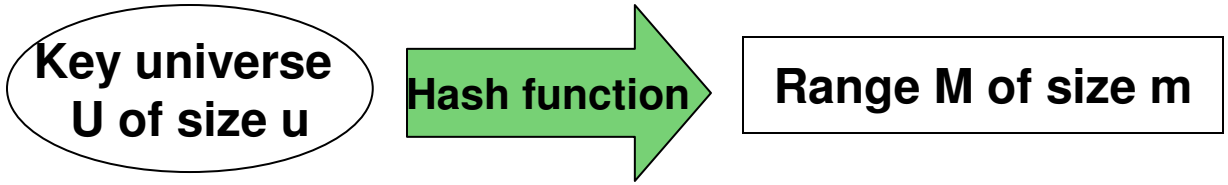
$$m^u$$

- # of bits to encode each fucntion

$$u \log m$$

- Independent functions with values uniformly distributed

Uniform Hashing Versus Universal Hashing



Uniform hashing

- # of functions from U to M?

$$m^u$$

- # of bits to encode each fucntion

$$u \log m$$

- Independent functions with values uniformly distributed

Universal hashing

- A family of hash functions $\mathcal{H}$ is universal if:
  - for any pair of distinct keys (x_1, x_2) from U and
  - a hash function **h** chosen uniformly from $\mathcal{H}$ then:

$$\Pr(h(x_1) = h(x_2)) \leq \frac{1}{m}$$

Uniform Hashing Versus Universal Hashing

- Intuition behind universal hashing
  - We often lose relatively little compared to using a completely random map (uniform hashing)
- If S of size n is hashed to n^2 buckets, with probability more than ½, no collisions occur
  - Even with complete randomness, we do not expect little $o(n^2)$ buckets to suffice (the birthday paradox)
  - So nothing is lost by using a universal family instead!

Related Work

- Theoretical Results
  (use uniform hashing)
- Practical Results
  (assume uniform hashing for free)
- Heuristics

Theoretical Results

Work	Gen. Time	Eval. Time	Size (bits)
Mehlhorn (1984)	Expon.	Expon.	O(n)
Schmidt and Siegel (1990)	Not analyzed	O(1)	O(n)
Hagerup and Tholey (2001)	O(n+log log u)	O(1)	O(n)

Theoretical Results

Work	Gen. Time	Eval. Time	Size (bits)
Mehlhorn (1984)	Expon.	Expon.	O(n)
Schmidt & Siegel (1990)	Not analyzed	O(1)	O(n)
Hagerup & Tholey (2001)	O(n+log log u)	O(1)	O(n)
Botelho & Ziviani (CIKM 2007)	**O(n)**	**O(1)**	**O(n)**

Practical Results – Assume Uniform Hashing For Free

Work	Gen. Time	Eval. Time	Size (bits)
Czech, Havas & Majewski (1992)	**O(n)**	**O(1)**	**O(n log n)**
Majewski, Wormald, Havas & Czech (1996)	**O(n)**	**O(1)**	**O(n log n)**
Pagh (1999)	**O(n)**	**O(1)**	**O(n log n)**
Botelho, Kohayakawa, & Ziviani (2005)	**O(n)**	**O(1)**	**O(n log n)**

Practical Results – Assume Uniform Hashing For Free

Work	Gen. Time	Eval. Time	Size (bits)
Czech, Havas & Majewski (1992)	**O(n)**	**O(1)**	**O(n log n)**
Majewski, Wormald, Havas & Czech (1996)	**O(n)**	**O(1)**	**O(n log n)**
Pagh (1999)	**O(n)**	**O(1)**	**O(n log n)**
Botelho, Kohayakawa, & Ziviani (2005)	**O(n)**	**O(1)**	**O(n log n)**
Botelho, Pagh, Ziviani (WADs 2007)	**O(n)**	**O(1)**	**O(n)**

Practical Results – Assume Uniform Hashing For Free

Work	Gen. Time	Eval. Time	Size (bits)
Czech, Havas & Majewski (1992)	**O(n)**	**O(1)**	**O(n log n)**
Majewski, Wormald, Havas & Czech (1996)	**O(n)**	**O(1)**	**O(n log n)**
Pagh (1999)	**O(n)**	**O(1)**	**O(n log n)**
Botelho, Kohayakawa, & Ziviani (2005)	**O(n)**	**O(1)**	**O(n log n)**
Botelho, Pagh, Ziviani (WADs 2007)	**O(n)**	**O(1)**	**O(n)**
Botelho & Ziviani (CIKM 2007)	**O(n)**	**O(1)**	**O(n)**

Empirical Results

Work	Application	Gen. Time	Eval. Time	Size (bits)
Fox, Chen & Heath (1992)	**Index data in CD-ROM**	**Exp.**	**O(1)**	**O(n)**
Lefebvre & Hoppe (2006)	**Sparse spatial data**	**O(n)**	**O(1)**	**O(n)**
Chang, Lin & Chou (2005, 2006)	**Data mining**	**O(n)**	**O(1)**	**Not analyzed**

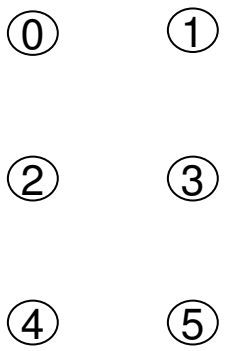
Internal and External Memory Based Algorithms

- Near space optimal
- Evaluation in constant time
- Function generation in linear time
- Simple to describe and implement
- Known algorithms with near-optimal space either:
  - Require exponential time for construction and evaluation, or
  - Use near-optimal space only asymptotically, for large *n*
- Acyclic random hypergraphs
  - Used before by Majewski et all (1996): O(n log n) bits
- We proceed differently: O(n) bits

  (we changed space complexity, close to theoretical lower bound)

Random Hypergraphs (r-graphs)

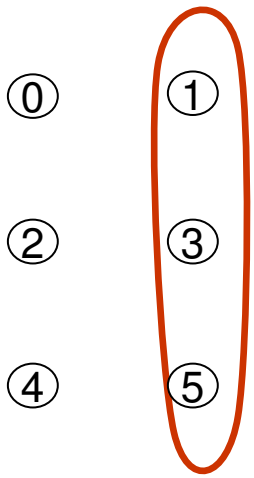
- 3-graph:



- 3-graph is induced by three uniform hash functions

Random Hypergraphs (r-graphs)

- 3-graph:

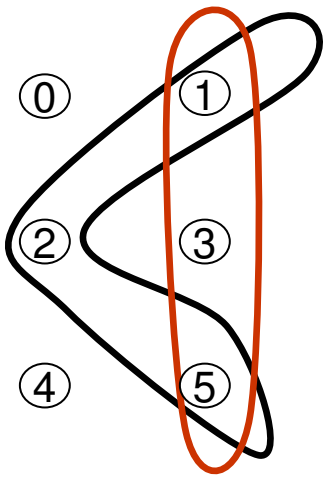


$h_0(jan) = 1 \quad h_1(jan) = 3 \quad h_2(jan) = 5$

- 3-graph is induced by three uniform hash functions

Random Hypergraphs (r-graphs)

- 3-graph:



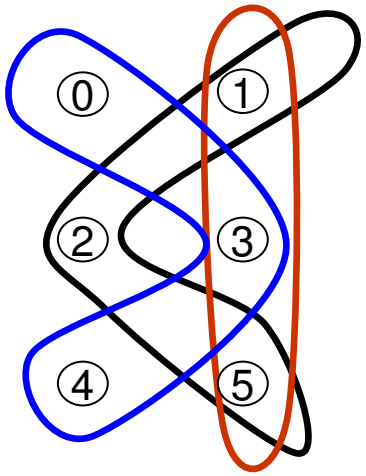
$h_0(jan) = 1 \quad h_1(jan) = 3 \quad h_2(jan) = 5$

$h_0(feb) = 1 \quad h_1(feb) = 2 \quad h_2(feb) = 5$

- 3-graph is induced by three uniform hash functions

Random Hypergraphs (r-graphs)

- 3-graph:



$h_0(jan) = 1 \quad h_1(jan) = 3 \quad h_2(jan) = 5$

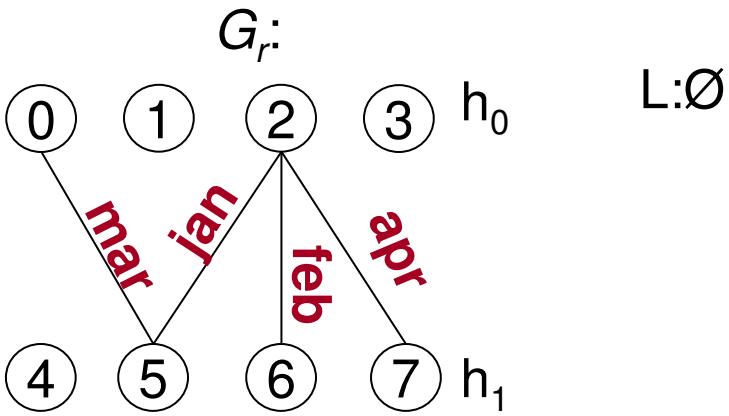
$h_0(feb) = 1 \quad h_1(feb) = 2 \quad h_2(feb) = 5$

$h_0(mar) = 0 \quad h_1(mar) = 3 \quad h_2(mar) = 4$

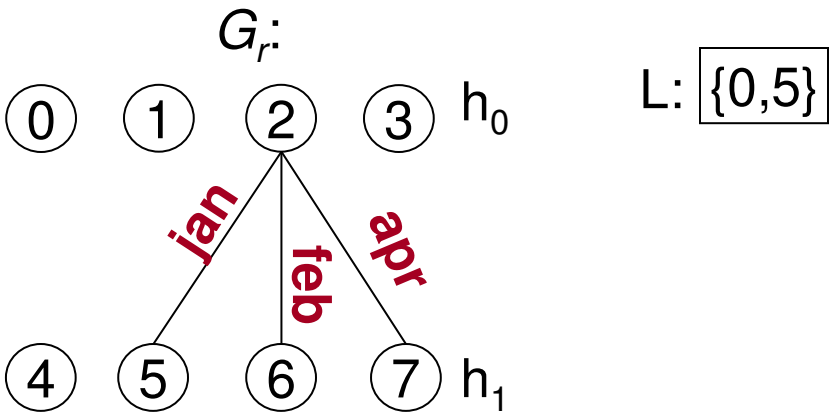
- 3-graph is induced by three uniform hash functions
- Our best result uses 3-graphs

The Internal memory based algorithm ...

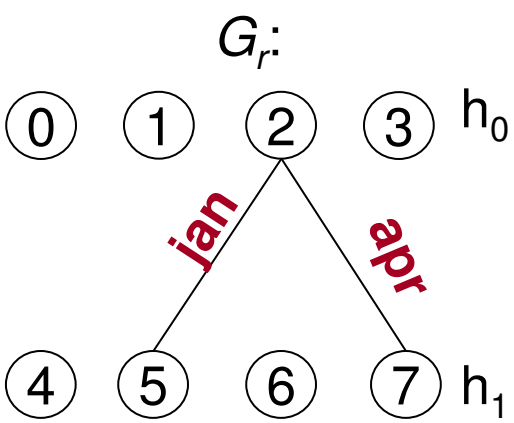
Acyclic 2-graph



Acyclic 2-graph

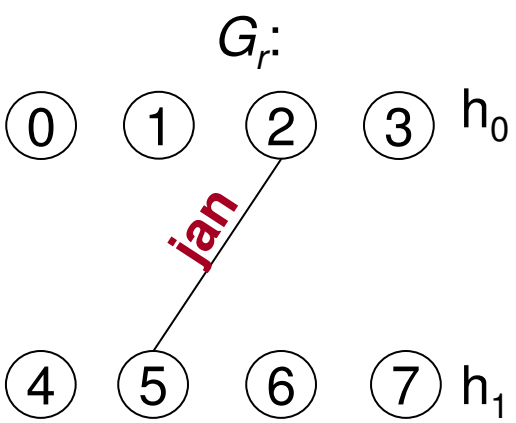


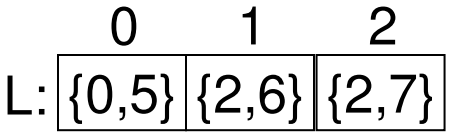
Acyclic 2-graph



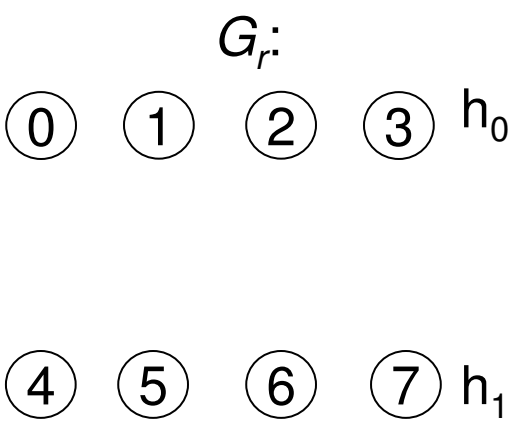
	0	1
L:	{0,5}	{2,6}

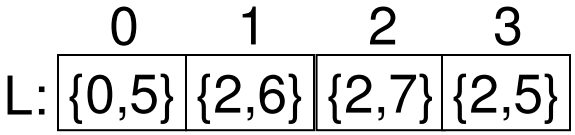
Acyclic 2-graph





Acyclic 2-graph





G_r is acyclic

The Family of Algorithms (r = 2)

S

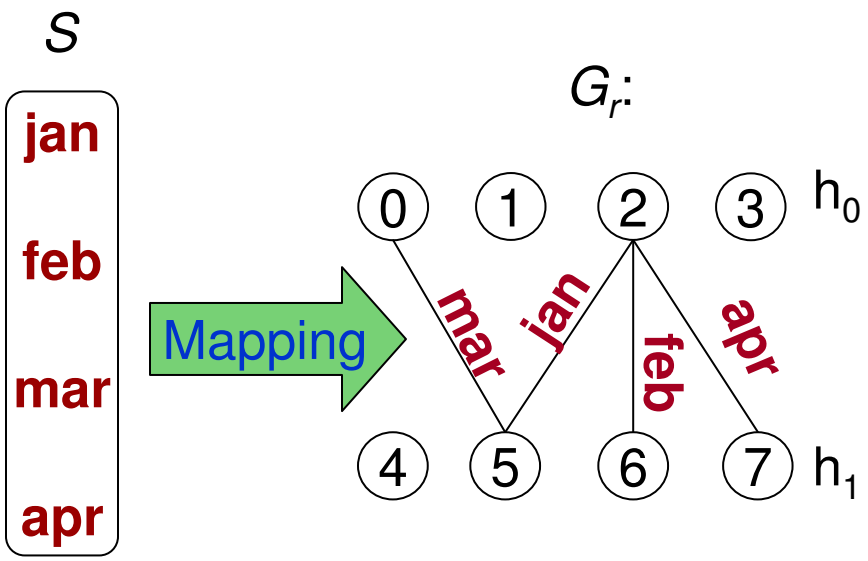
jan

feb

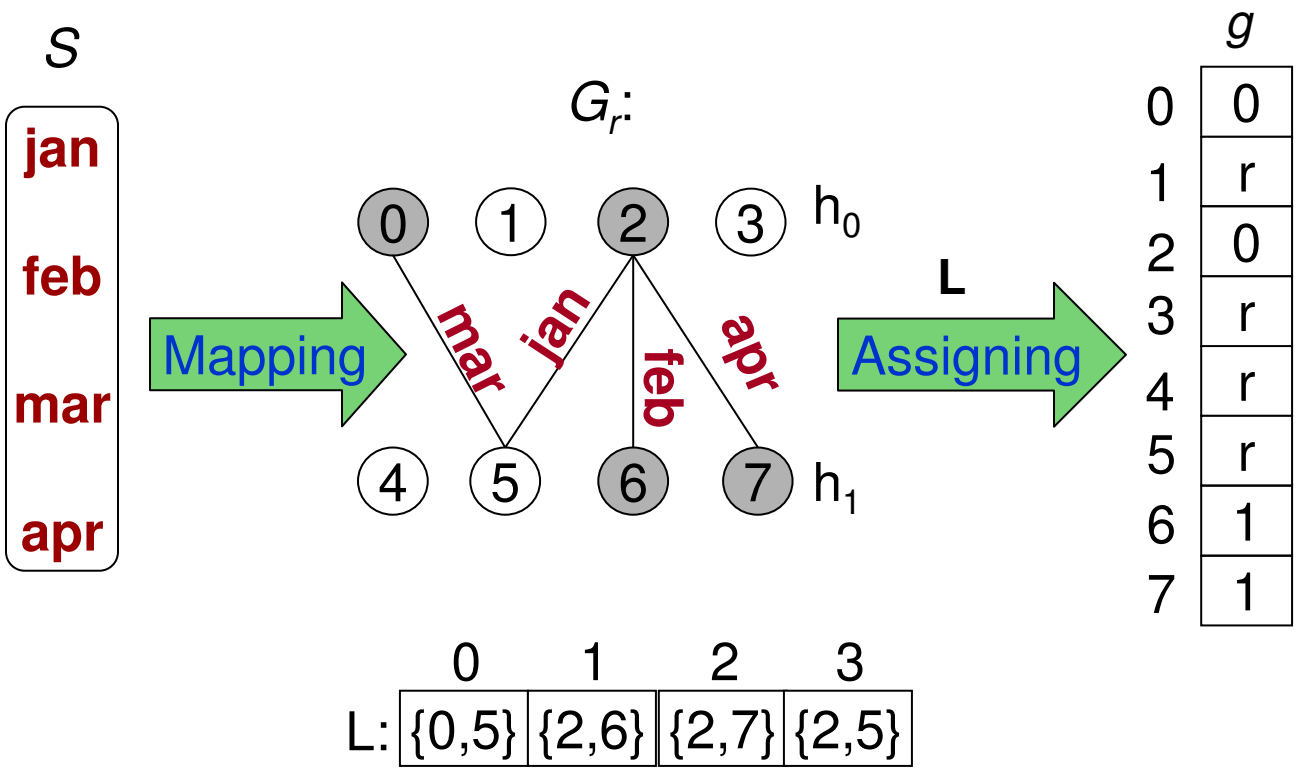
mar

apr

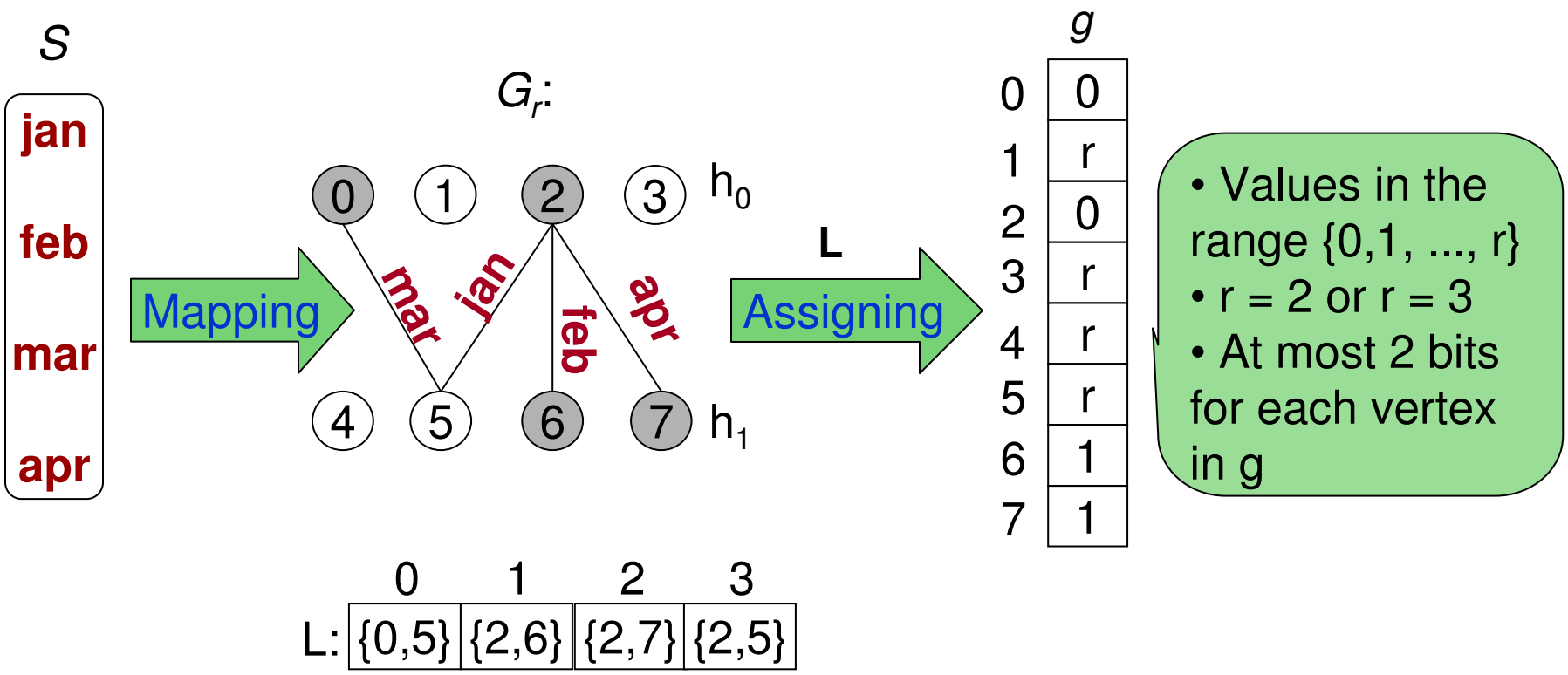
The Family of Algorithms (r = 2)



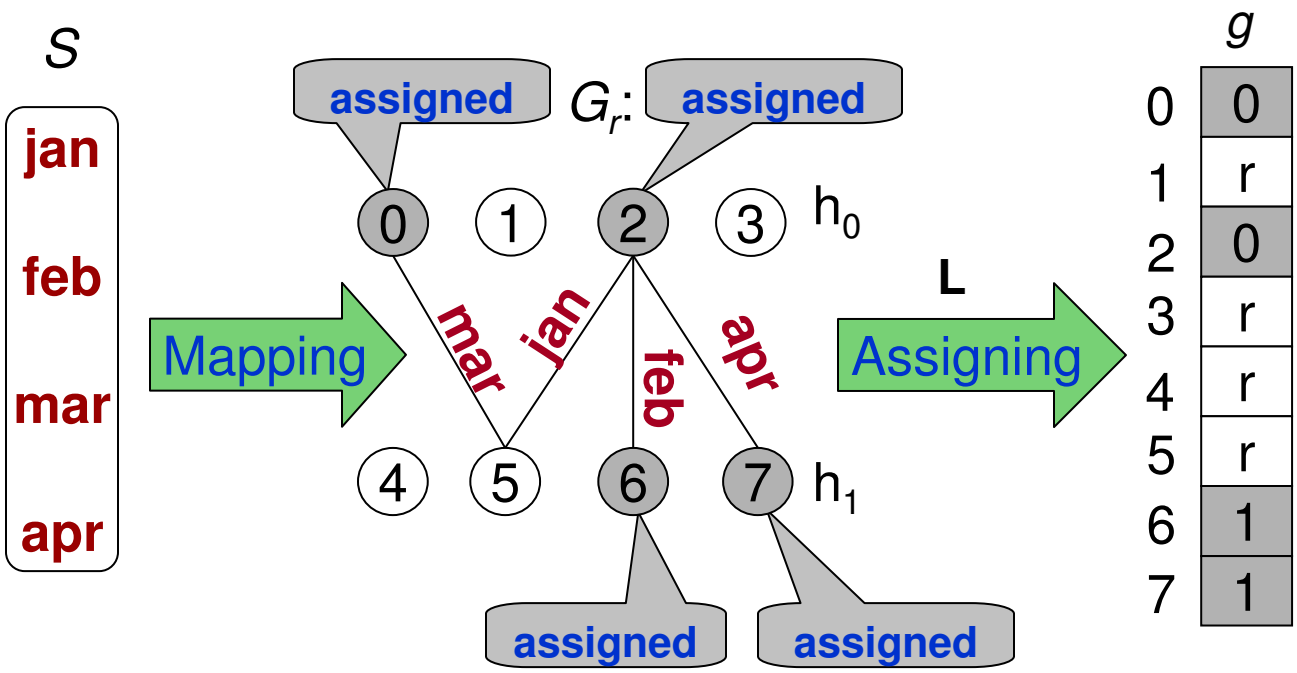
The Family of Algorithms (r = 2)



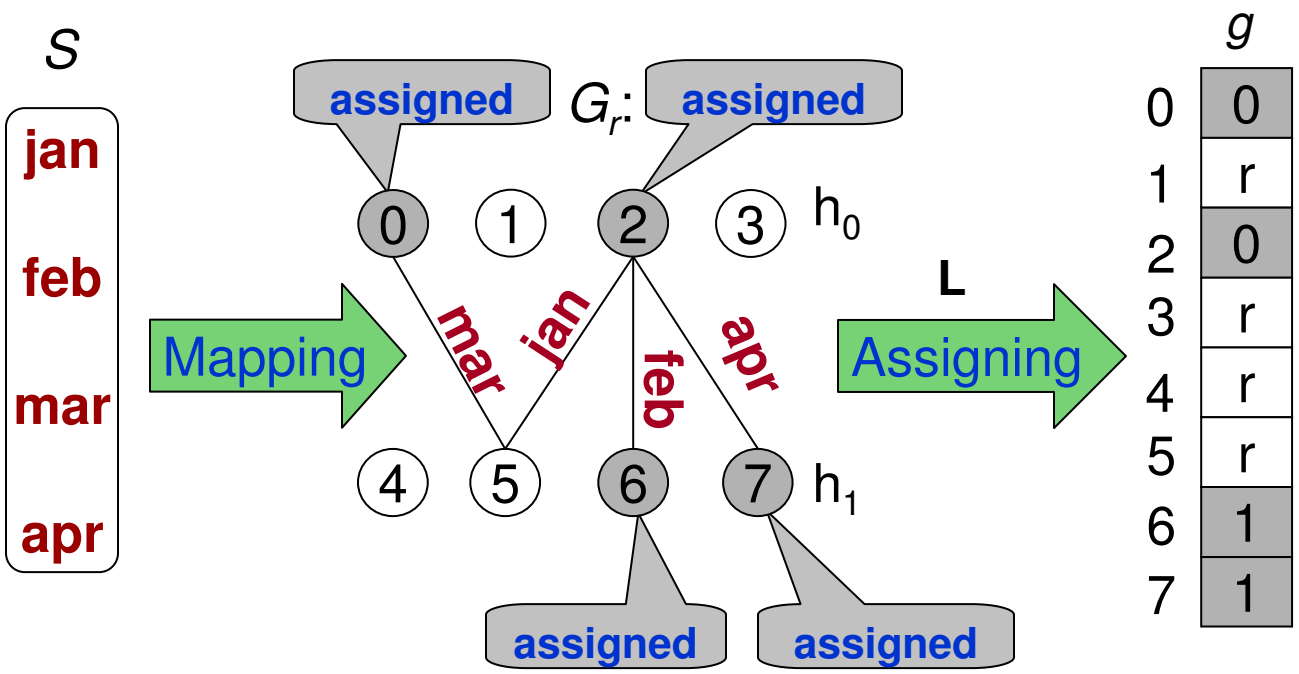
The Family of Algorithms (r = 2)



The Family of Algorithms (r = 2)

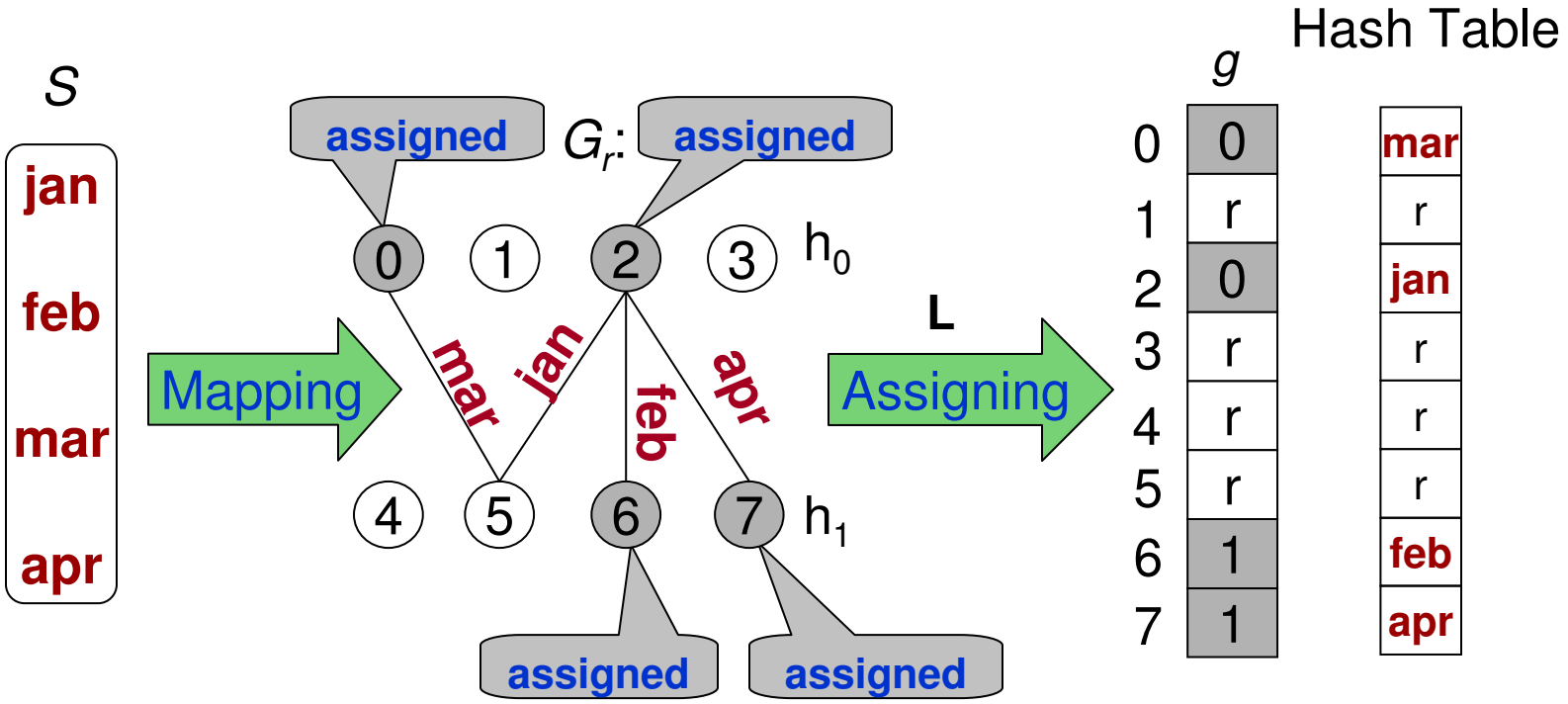


The Family of Algorithms (r = 2)



$$i = (g(h_0(feb)) + g(h_1(feb))) \bmod \mathbf{r} = (g(2) + g(6)) \bmod \mathbf{2} = 1$$

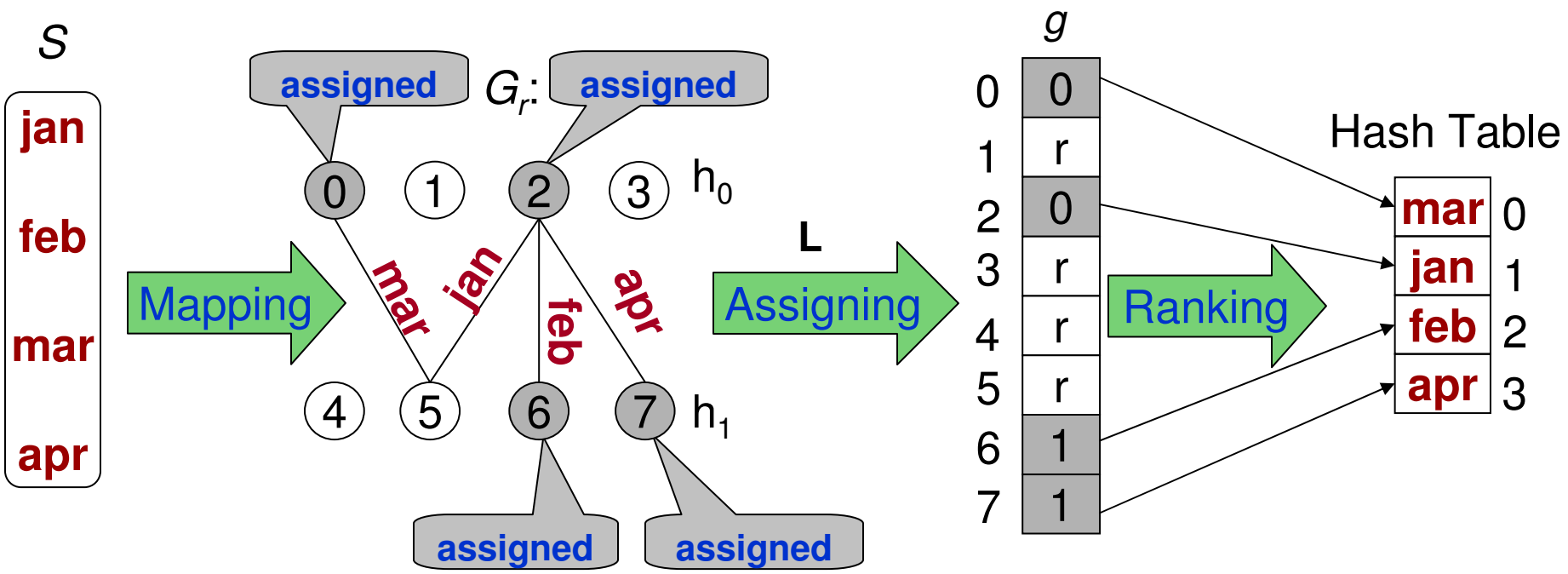
The Family of Algorithms (r = 2)



$i = (g(h_0(feb)) + g(h_1(feb))) \bmod \mathbf{r} = (g(2) + g(6)) \bmod \mathbf{2} = 1$

$phf(feb) = h_{i=1}(feb) = 6$

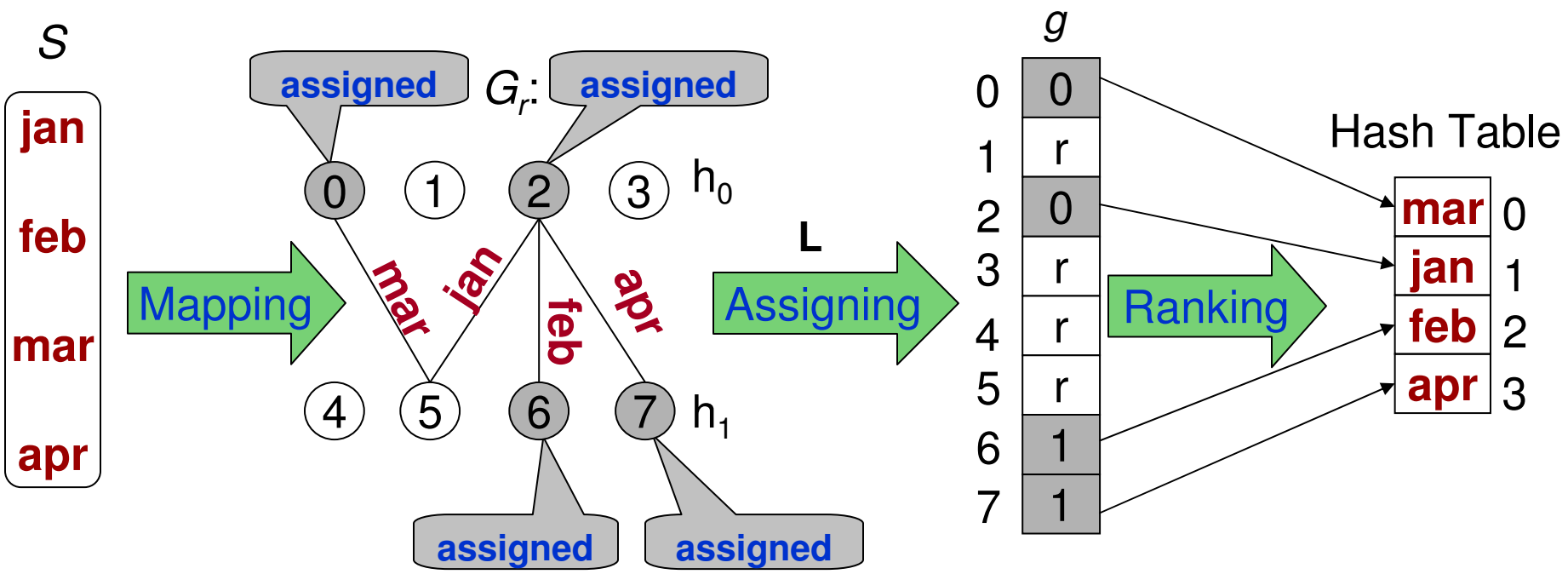
The Family of Algorithms (r = 2)



$i = (g(h_0(feb)) + g(h_1(feb))) \bmod \mathbf{r} = (g(2) + g(6)) \bmod \mathbf{2} = 1$

$phf(feb) = h_{i=1}(feb) = 6$

The Family of Algorithms (r = 2)



$i = (g(h_0(feb)) + g(h_1(feb))) \bmod \mathbf{r} = (g(2) + g(6)) \bmod \mathbf{2} = 1$

$phf(feb) = h_{i=1}(feb) = 6$

$mphf(feb) = rank(phf(feb)) = rank(6) = 2$

Use of Acyclic Random Hypergraphs

- Sufficient condition to work (Majewski et al (1996))
- Repeatedly selects $h_0, h_1 \ldots, h_{r-1}$
- For r = 2, m=cn and c>2,
  - For c = 2.09, $Pr_a = 0.29$ $\quad Pr_a = \sqrt{1-(2/c)^2}$
- For r = 3 and c≥1.23: probability tends to 1
- Number of iterations is $1/Pr_a$:
  - r = 2: 3.5 iterations
  - r = 3: 1.0 iteration

Space to Represent the Functions (r = 3)

- PHFs g: [0,m-1] → {0,1,2}
  - m = cn bits, c = 1.23 → 2.46 n bits
- Packed PHFs (Range of size 3):
  - (log 3) cn bits, c = 1.23 → **1.95 n** bits (arith. coding)
  - Optimal: **1.17n** bits
- MPHFs g: [0,m-1] → {0,1,2,3} (ranking info required)
  - 2m + εm = (2+ ε)cn bits
  - For c = 1.23 and ε = 0.125 → **2.62 n** bits
  - Optimal: **1.4427n** bits.

Experimental Results

- Metrics:
  - Generation time
  - Storage space
  - Evaluation time
- Collection:
  - 3,541,615 millions of URLs collected from the web
  - 64 bytes long on average
- Experiments
  - Commodity PC with a cache of 2 Mbytes
  - 3.2 GHz, 1 GB, Linux, 32 bits achiteture

Related Algorithms

- Botelho, Kohayakawa, Ziviani (2005) - BKZ
- **Botelho, Pagh, Ziviani (2007) - BPZ**
- Fox, Chen and Heath (1992) – FCH
- Majewski, Wormald, Havas and Czech (1996) – MWHC
- Pagh (1999) - PAGH

All algorithms coded in the same framework

Generation Time

Algorithms		Generation Time (sec)
BPZ	r = 2	19.49 ± 3.750
	r = 3	9.80 ± 0.007
BKZ		16.85 ± 1.85
FCH		5901.9 ± 1489.6
MWHC		10. 63 ± 0.09
PAGH		52.55 ± 2.66

n=3,541,615 keys

Storage Space

Algorithms		Storage Space	
		Bits/Key	Size (MB)
BPZ	r = 2	3.60	1.52
	r = 3	2.62	1.11
BKZ		21.76	9.19
FCH		3.66	1.55
MWHC		26.76	11.30
PAGH		44.16	18.65

n=3,541,615 keys

Evaluation Time

Algorithms		Evaluation Time (sec)
BPZ	**r = 2**	2.63
	r = 3	2.73
BKZ		2.81
FCH		2.14
MWHC		2.85
PAGH		2.78

n=3,541,615 keys

Internal Memory Based Algorithm Summary

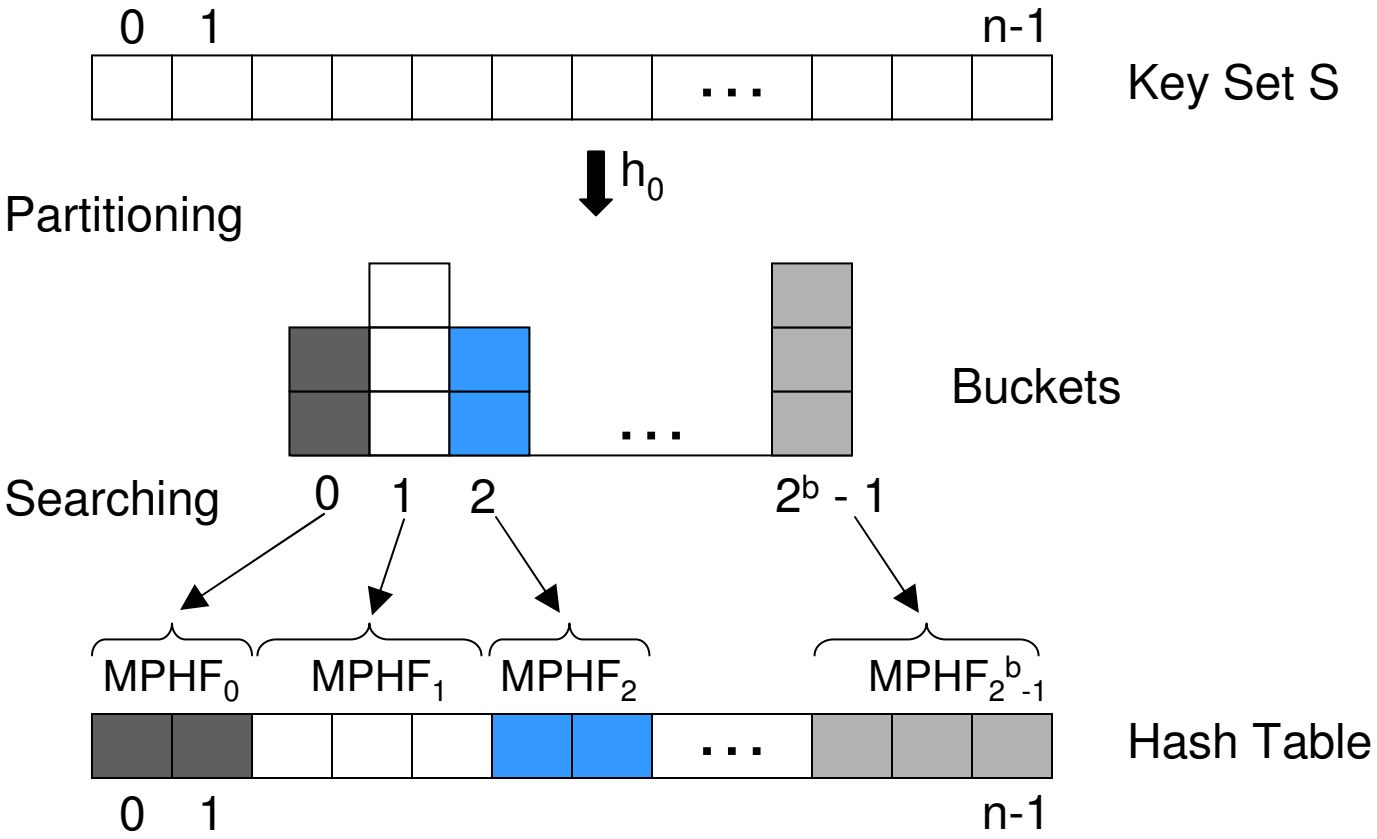
- Presented an efficient family of algorithms
  - Near space-optimal PHFs and MPHFs
  - **Compact functions fit in cache**
- The algorithms are simpler and has much lower constant factors than existing theoretical results
- Outperforms the main practical general purpose algorithms found in the literature considering
  - generation time
  - storage space

The External memory based algorithm ...

External Memory Based Algorithm

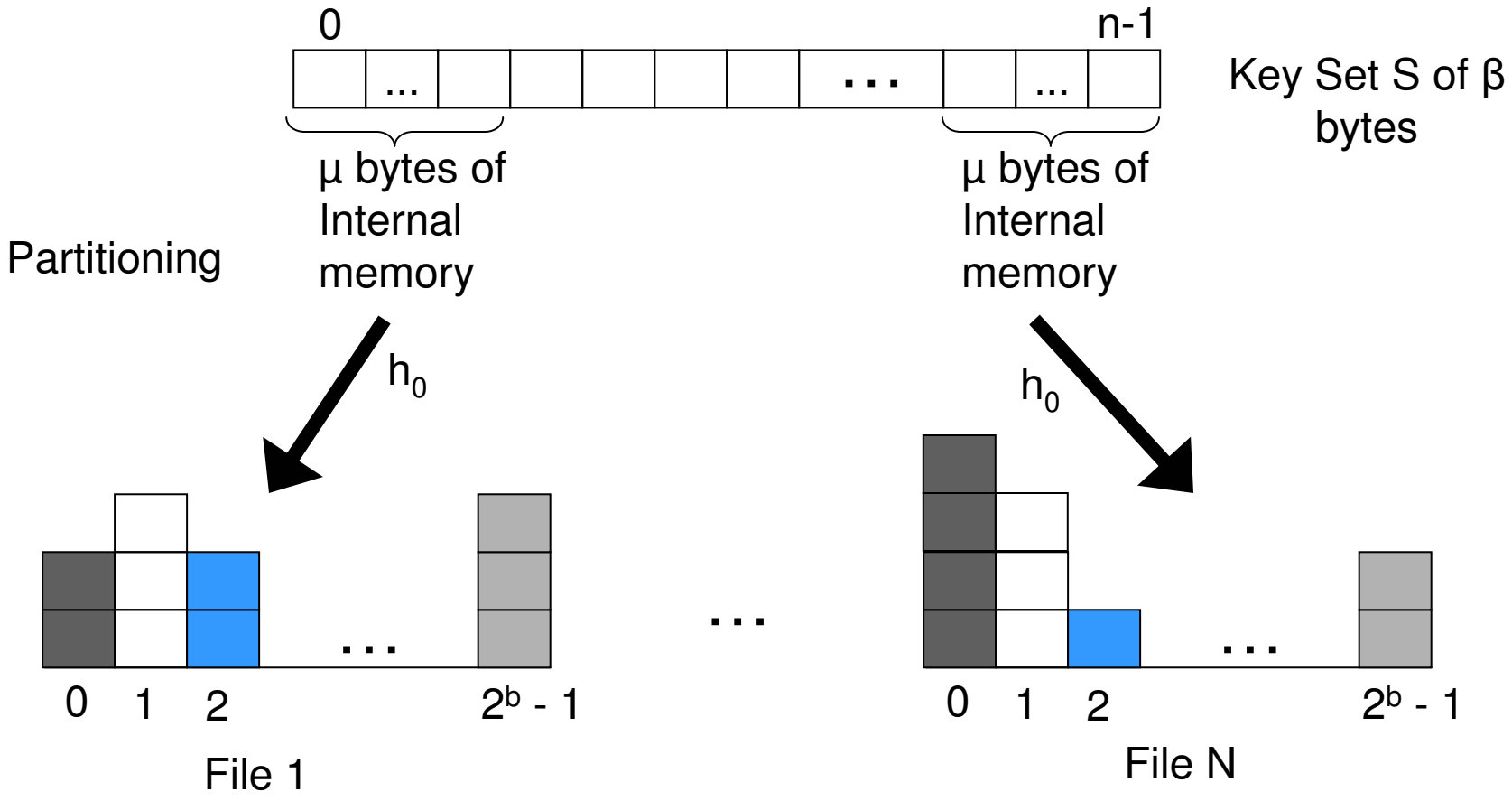
- First MPHF algorithm for very large key sets (in the order of billions of keys)
- This is possible because
  - Deals with external memory efficiently
  - Generates compact functions (near space optimal)
  - Uses a little amount of internal memory to scale
  - Works in linear time
- Two implementations:
  - Theoretical well-founded EPH (uses uniform hashing)
  - Heuristic EPH (uses universal hashing)

The External Perfect Hashing Algorithm (EPH)



$mphf(x) = mphf_i(x) + offset[i];$

Key Set Does Not Fit In Internal Memory

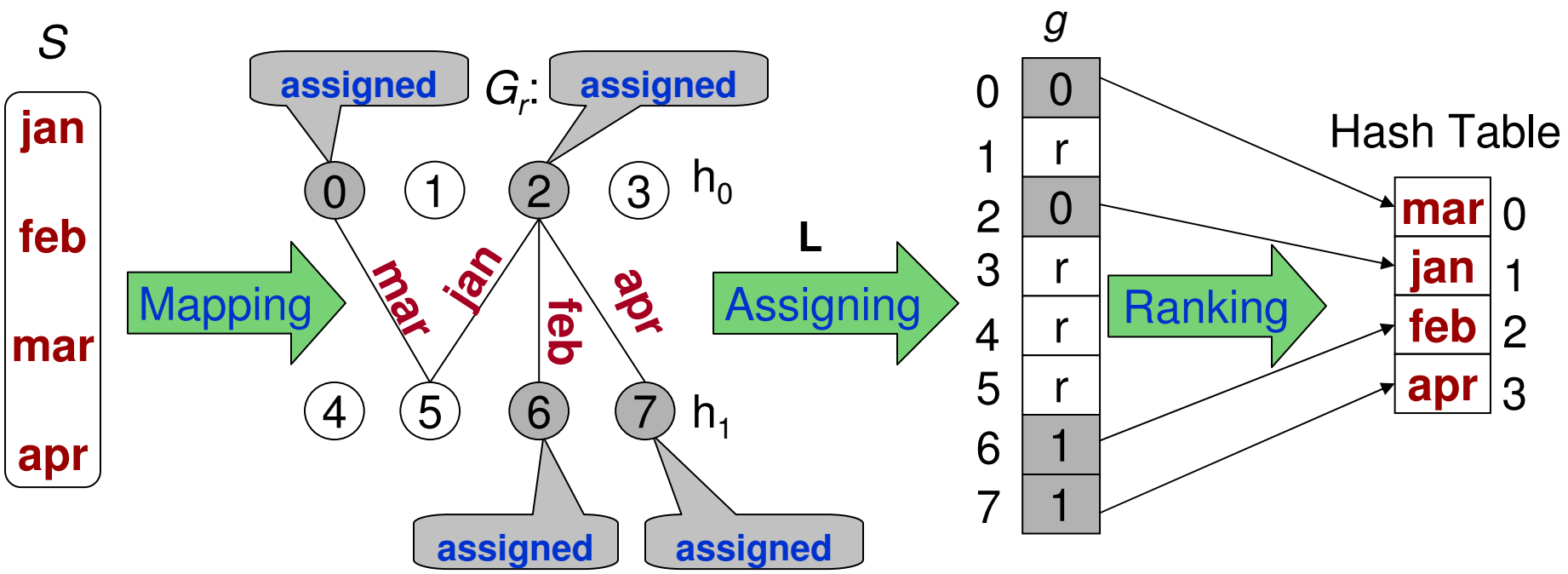


N = β/μ b = Number of bits of each bucket address Each bucket ≤ 256

Important Design Decisions

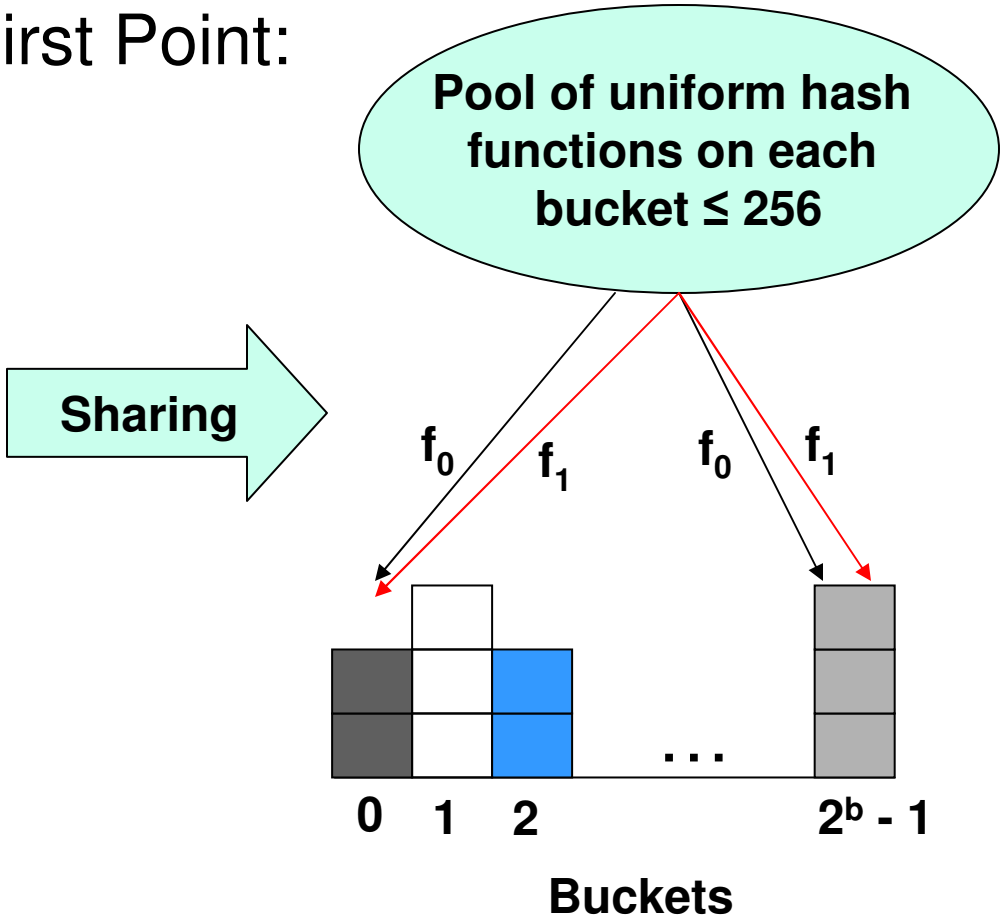
- We map long URLs to smaller keys of fixed size using a hash function
- Choose a linear time and near space-optimal algorithm to generate the MPHF of each bucket
- How do we obtain a linear time complexity?
  - Using internal radix sorting to form the buckets
  - Using a heap of N entries to drive a N-way merge that reads the buckets from disk

Use the Internal Algorithm in Each Bucket



Why the EPH Algorithm is Well-Founded?

First Point:



Why the EPH Algorithm is Well-Founded?

Second Point:

We have shown how to create that pool based on the linear hash functions proposed by Alon et al (JACM 1999)

$$f(x,s,\Delta) = \left(\sum_{j=1}^{k} t_j[y_j(x) \oplus \Delta] + s \sum_{j=k+1}^{2k} t_j[y_{j-k}(x) \oplus \Delta] \right) \bmod p$$

$$h_{i1}(x) = f(x,s,0) \bmod |B_i|$$

$$h_{i2}(x) = f(x,s,1) \bmod |B_i|$$

Why the EPH Algorithm is Well-Founded?

Second Point:

We have shown how to create that pool based on the linear hash functions proposed by Alon et al (JACM 1999)

$$f(x,s,\Delta) = \left(\sum_{j=1}^{k} t_j [y_j(x) \oplus \Delta] + s \sum_{j=k+1}^{2k} t_j [y_{j-k}(x) \oplus \Delta] \right) \bmod p$$

$$h_{i1}(x) = f(x,s,0) \bmod |B_i|$$

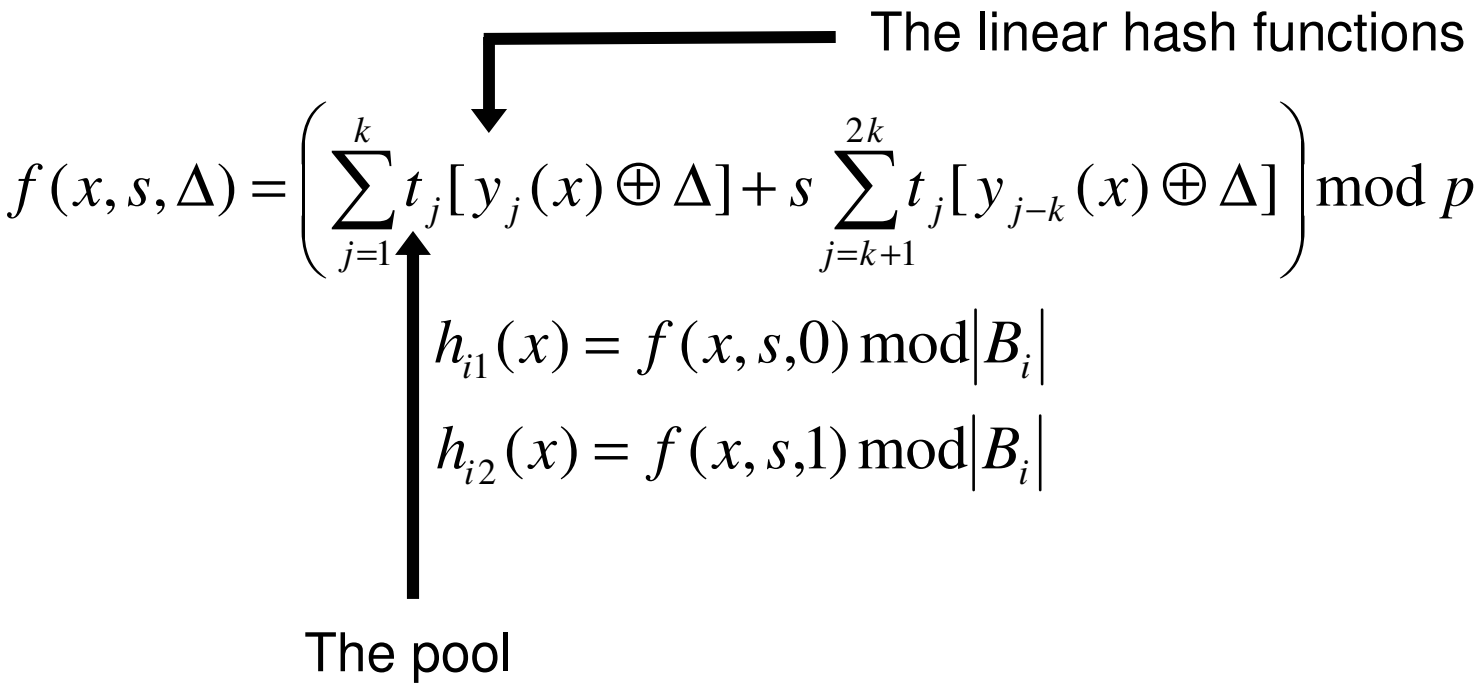
$$h_{i2}(x) = f(x,s,1) \bmod |B_i|$$

The pool

Why the EPH Algorithm is Well-Founded?

Second Point:

We have shown how to create that pool based on the linear hash functions proposed by Alon et al (JACM 1999)



Why the EPH Algorithm is Well-Founded?

Second Point:

Computing 128 bits with the linear hash functions

$h'(x) = 10010101110011011010000111000110$

$h_0(x) = h'(x)[96,127] >> (32-b)$

$y_6(x) = h'(x)[80,95]$

1101110101001001

0011000111000110

0000000111010110

0011111111000111

1111111111000110

0000000000000111

$y_1(x) = h'(x)[0,15]$

Why the EPH Algorithm is Well-Founded?

Third Point:

How to keep maximum bucket size smaller than $l = 256$?

$$b \leq \log(n) - \log(l / \log l) + O(1)$$

$$l \geq \log n \log \log n$$

The Heuristic EPH Algorithm

- Theoretically founded hash functions from the pool:
  - Slower than simpler universal hash functions and pseudo random hash functions
  - Require a lookup table that do not fit in cache
- Heuristic EPH uses a universal pseudo random hash function proposed by Jenkins (1997):
  - Faster hash function
  - Requires just one random integer number as seed to be stored
  - Using universal hashing we often loose relatively little compared to using a complete uniform random map

Experimental Results

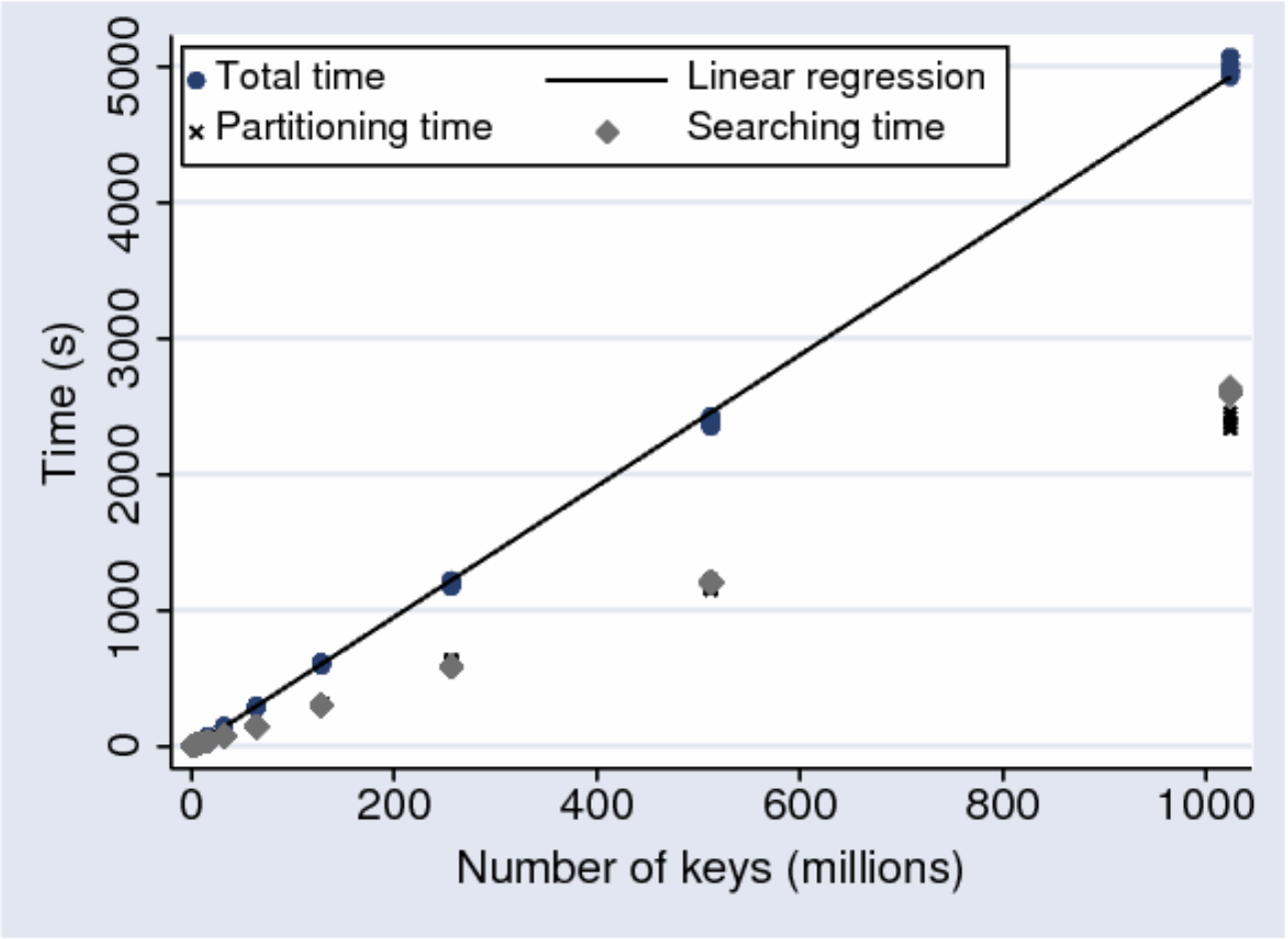
- Metrics:
  - Generation time
  - Storage space
  - Evaluation time
- Collection:
  - 1.024 billions of URLs collected from the web
  - 64 bytes long on average
- Experiments
  - Commodity PC with a cache of 512 Kbytes
  - 1 GHz, 1 GB, Linux, 64 bits achiteture

Generation Time (in Minutes)

n (millions)	32	128	512	1024
EPH	2.5 ± 0.02	10.1 ± 0.1	39.7 ± 0.4	83.0 ± 0.4
Heuristic EPH	1.9 ± 0.05	7.3 ± 0.1	30.9 ± 0.5	64.3 ± 1.1

Time in minutes to construct a MPHF for 32, 128, 512 and 1,024 millions of keys

Generation Time



Related Algorithms

- Botelho, Kohayakawa, Ziviani (2005) - BKZ
- Botelho, Pagh, Ziviani (2007) - BPZ
- **Botelho, Ziviani (2007) - EPH**
- Fox, Chen and Heath (1992) – FCH
- Czech, Havas and Majewski (1992) – CHM
- Pagh (1999) - PAGH

All algorithms coded in the same framework

Generation Time

Algorithms	Generation time (sec)
EPH	6.98 ± 0.01
Heuristic EPH	4.75 ± 0.02
BKZ	8.57 ± 0.31
BPZ	13.94 ± 1.06
CHM	13.80 ± 0.70
FCH	758.66 ± 126.72
PAGH	46.18 ± 1.06

2 million keys

Generation Time and Storage Space

Algorithms	Generation time (sec)	Space (bits/key)
EPH	6.98 ± 0.01	3.85
Heuristic EPH	4.75 ± 0.02	3.7
BKZ	8.57 ± 0.31	32.0
BPZ	13.94 ± 1.06	2.62
CHM	13.80 ± 0.70	66.88
FCH	758.66 ± 126.72	5.84
PAGH	46.18 ± 1.06	64.0

2 million keys

Generation Time, Storage Space and Evaluation Time

Algorithms	Generation time (sec)	Space (bits/key)	Evaluation time (sec)
EPH	6.98 ± 0.01	3.85	4.63
Heuristic EPH	4.75 ± 0.02	3.7	2.11
BKZ	8.57 ± 0.31	32.0	1.48
BPZ (r = 2)	13.94 ± 1.06	3.35	1.73
CHM	13.80 ± 0.70	66.88	1.48
FCH	758.66 ± 126.72	5.84	1.46
PAGH	46.18 ± 1.06	64.0	1.44

2 million keys

Key length = 64 bytes

C Minimal Perfect Hashing Library

- Why to build a library?
  - Lack of similar libraries in the free software community
  - To test the applicability of our algorithms out there
- Feedbacks:
  - 1,790 downloads
  - Incorporated by Debian
- Library address: http://cmph.sourceforge.net

External Memory Based Algorithm Summary

- Two implementations were developed:
  - The Well-founded EPH algorithm
  - The Heuristic EPH algorithm
- They generate near space-optimal functions in linear time
- The functions are computed in time O(1) and are simple to be used in practice
- First theoretically well-founded algorithm that is practical and will work for every key set

Ongoing Work

- Distributed and Parallel Version of the EPH Algorithm

PCs	2	4	8	10
Speedup	1.7	3.3	6.6	7.3

- Representing the set of visited URLs when the web is being crawled