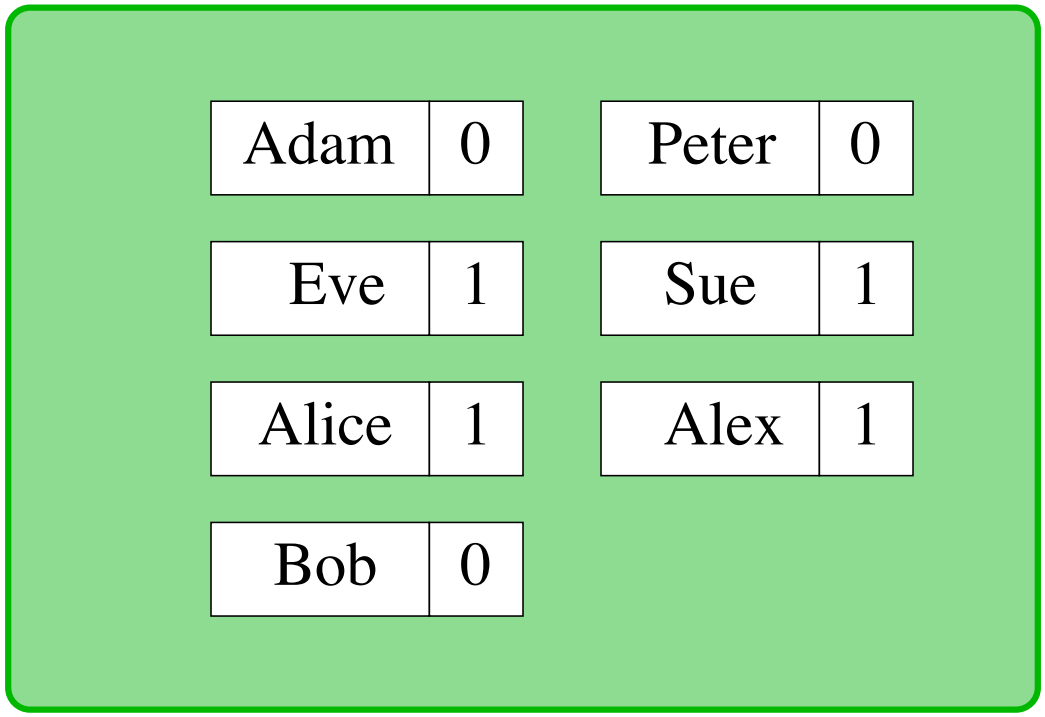
Succinct Data Structures for Retrieval and Approximate Membership

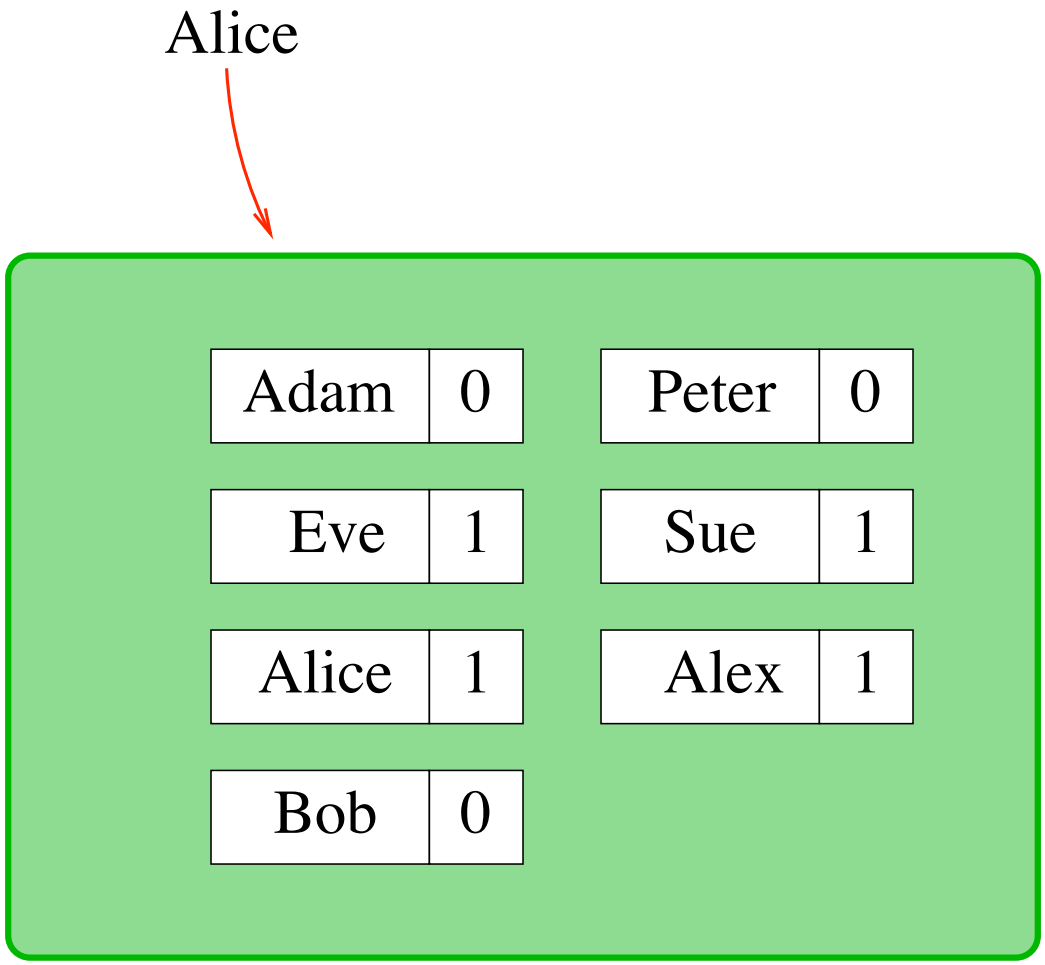
Martin Dietzfelbinger
Technische Universität Ilmenau

Joint work with Rasmus Pagh
February 18, 2008

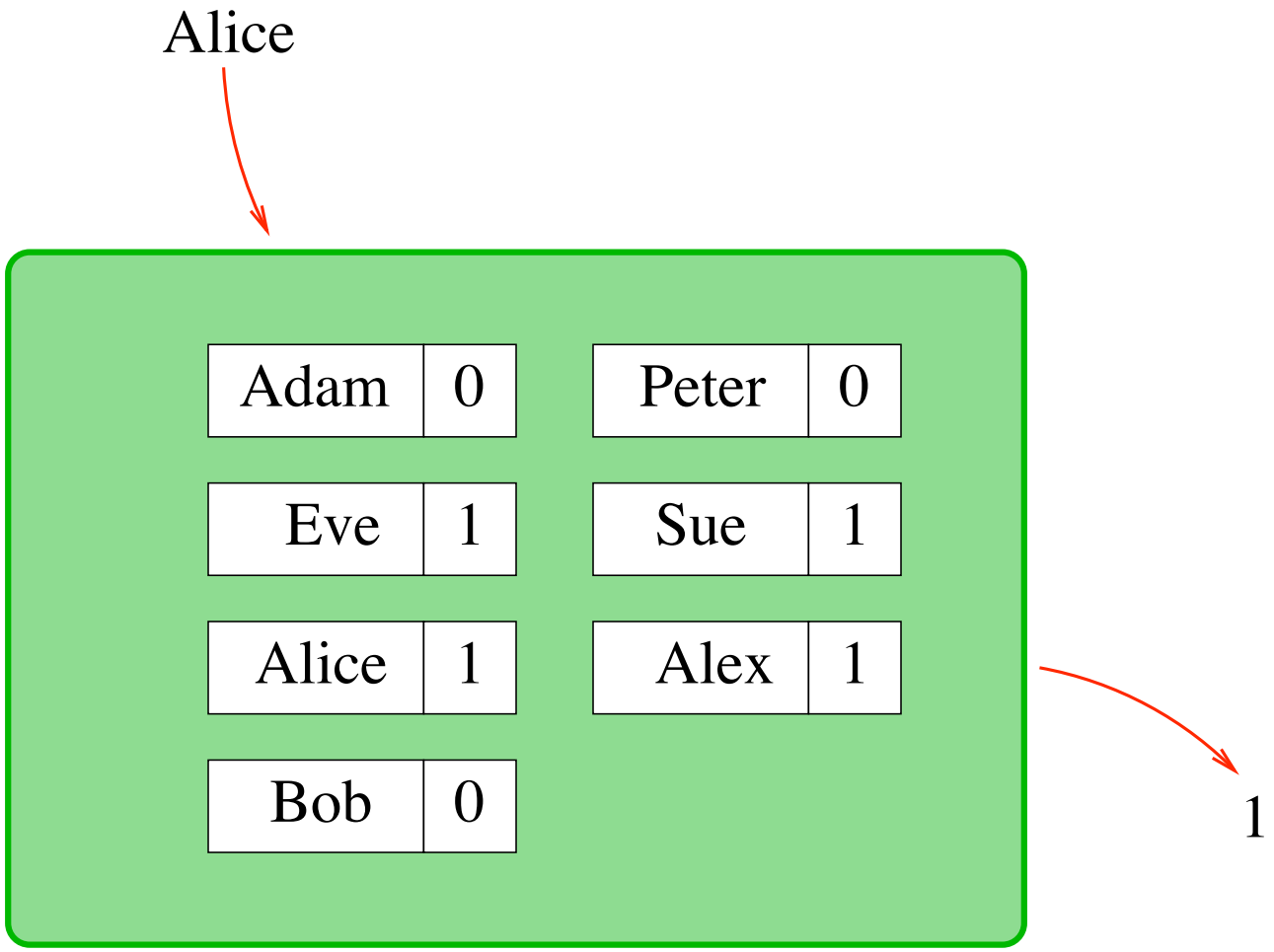
Retrieval



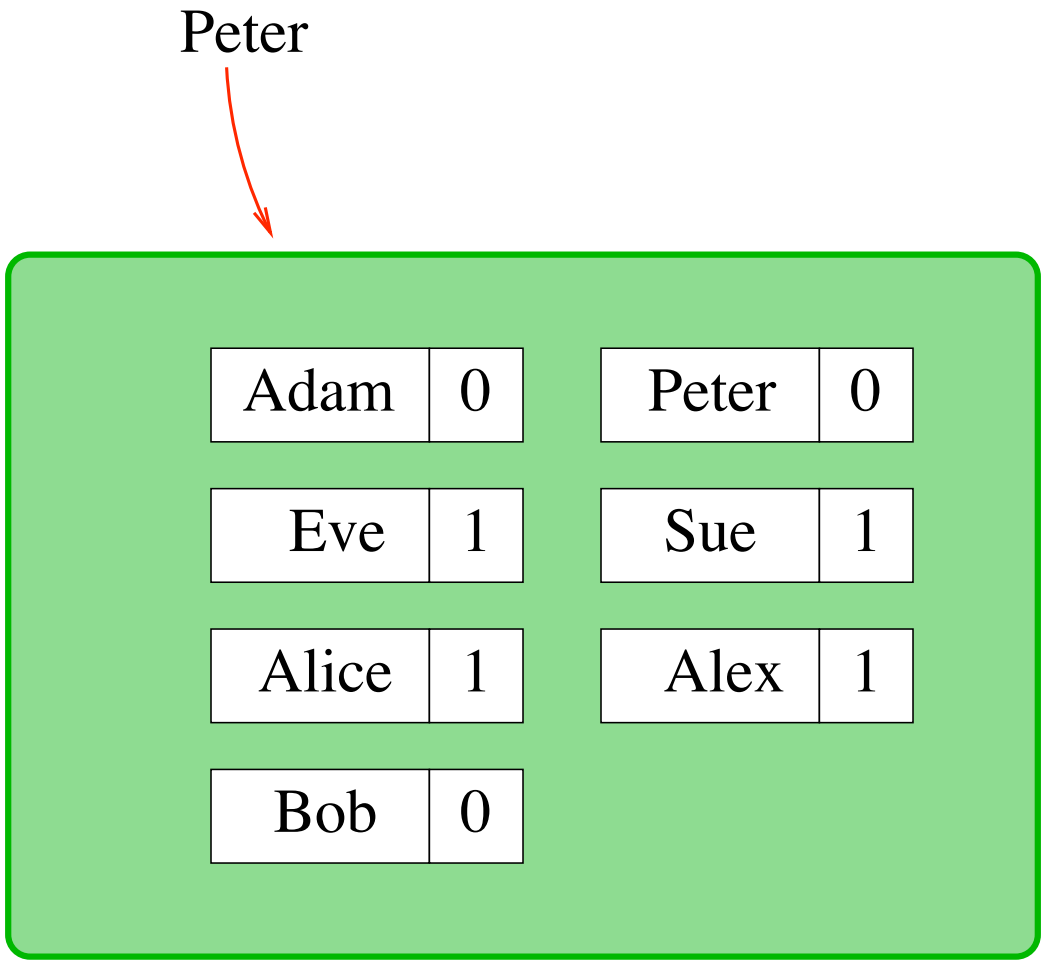
Retrieval



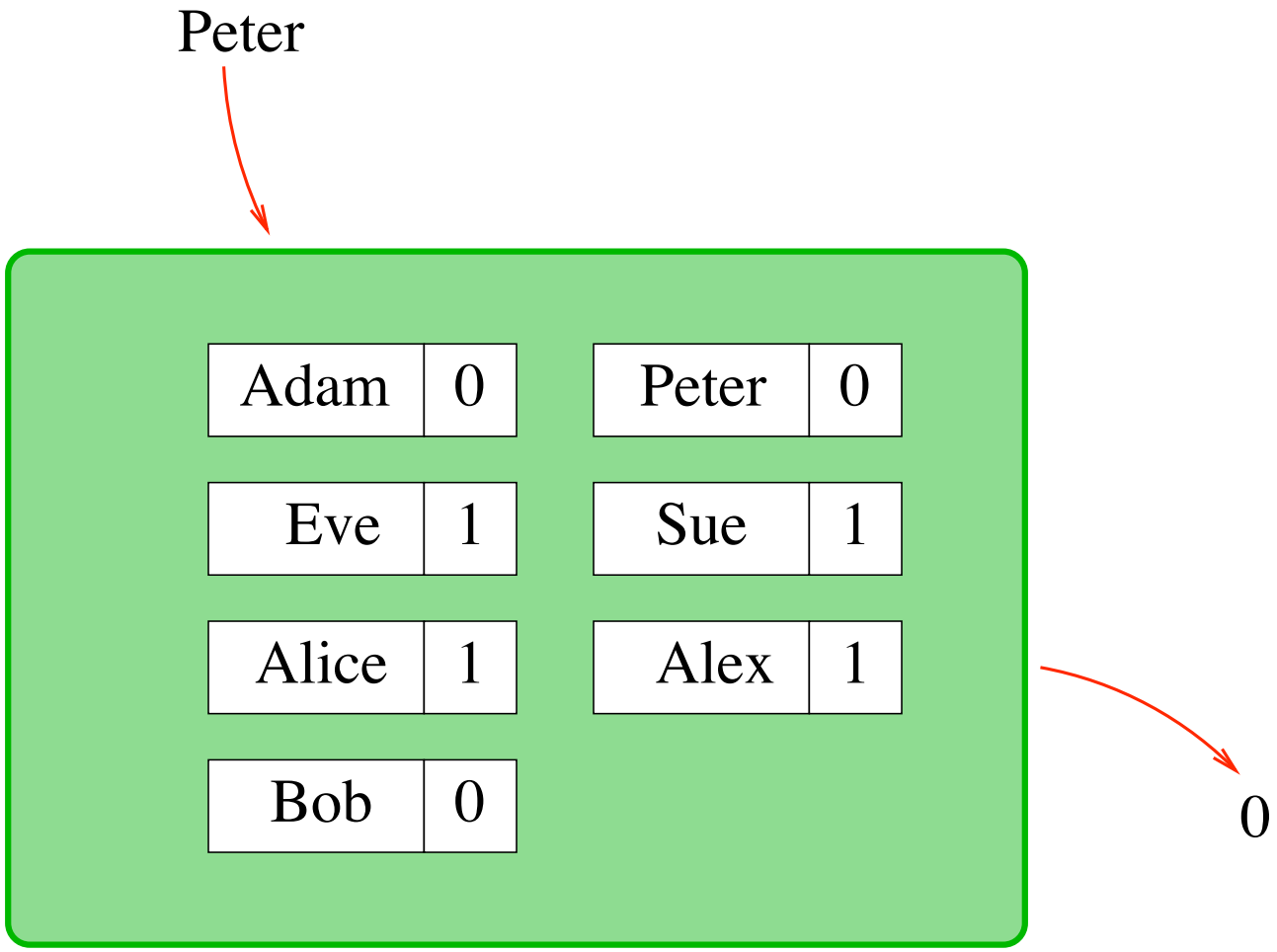
Retrieval



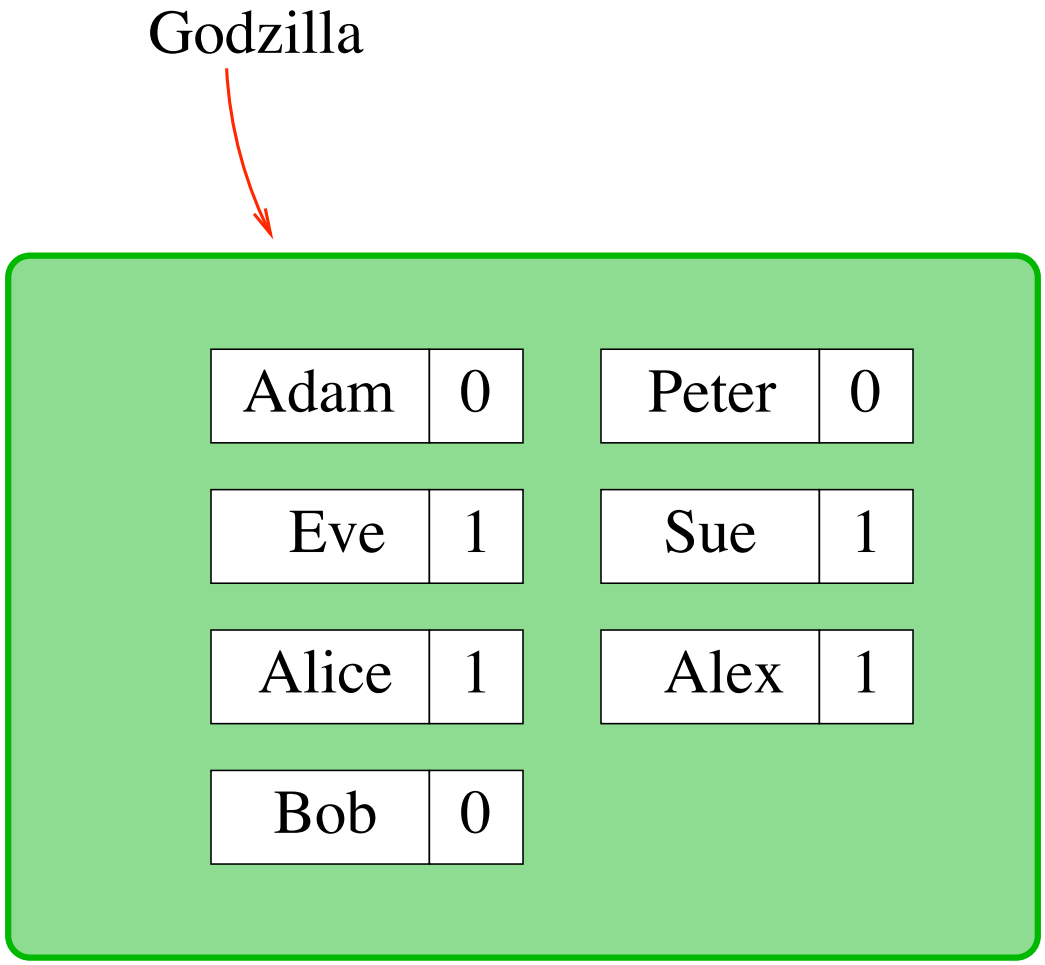
Retrieval



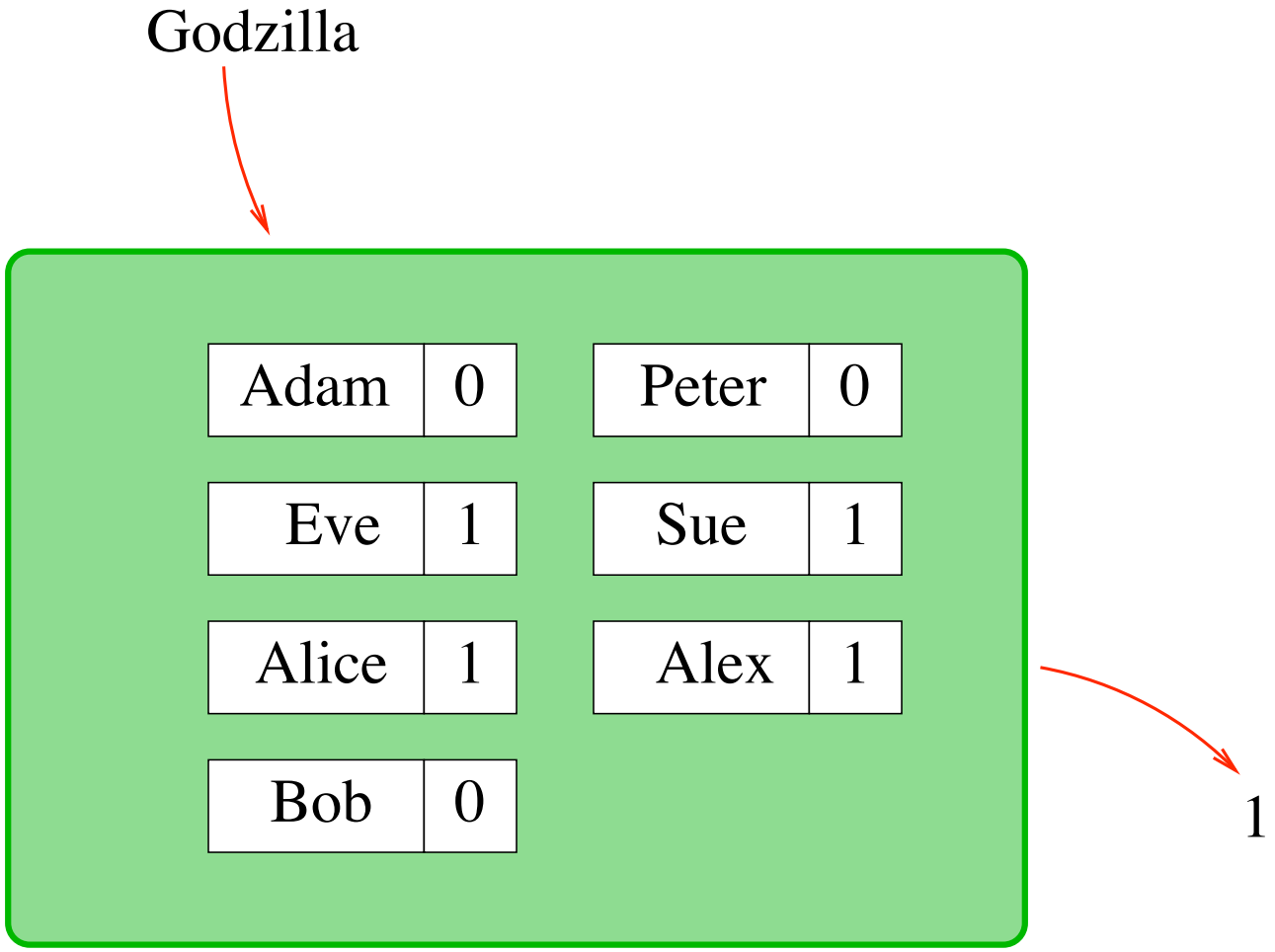
Retrieval



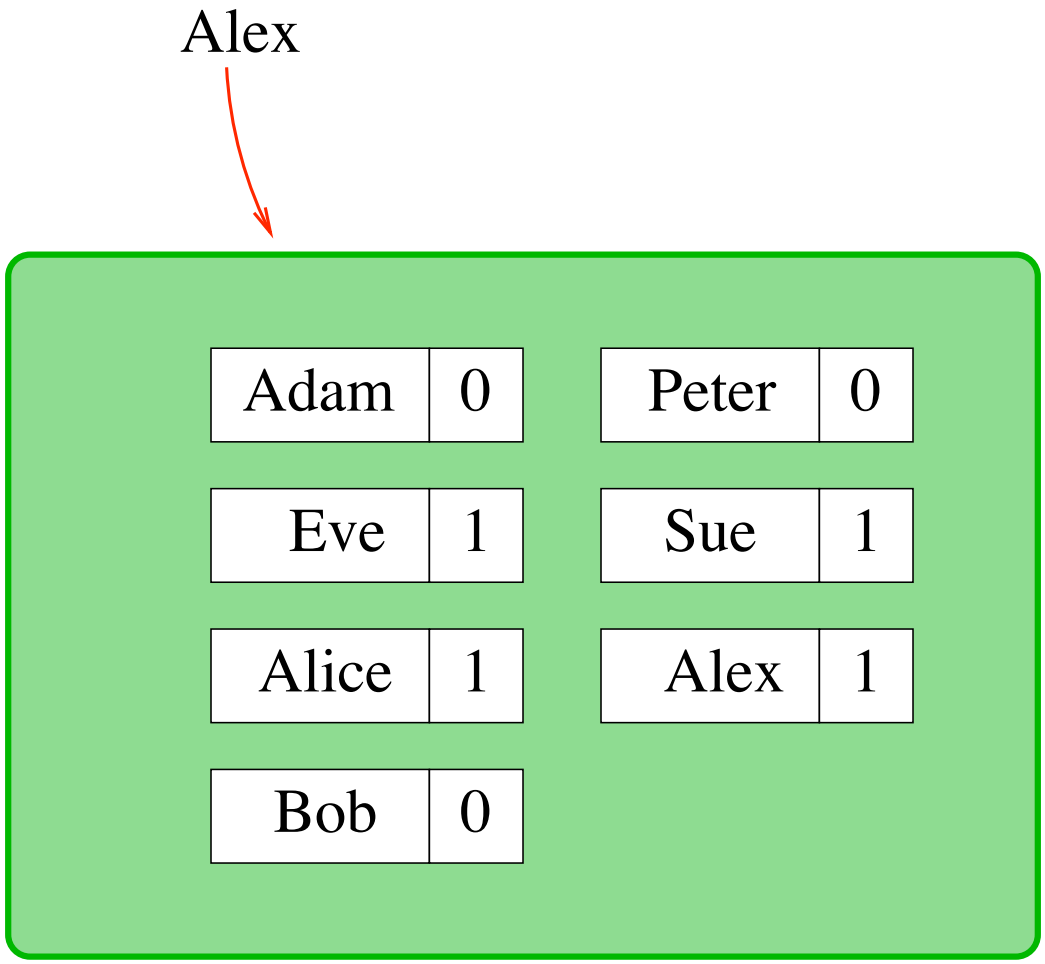
Retrieval



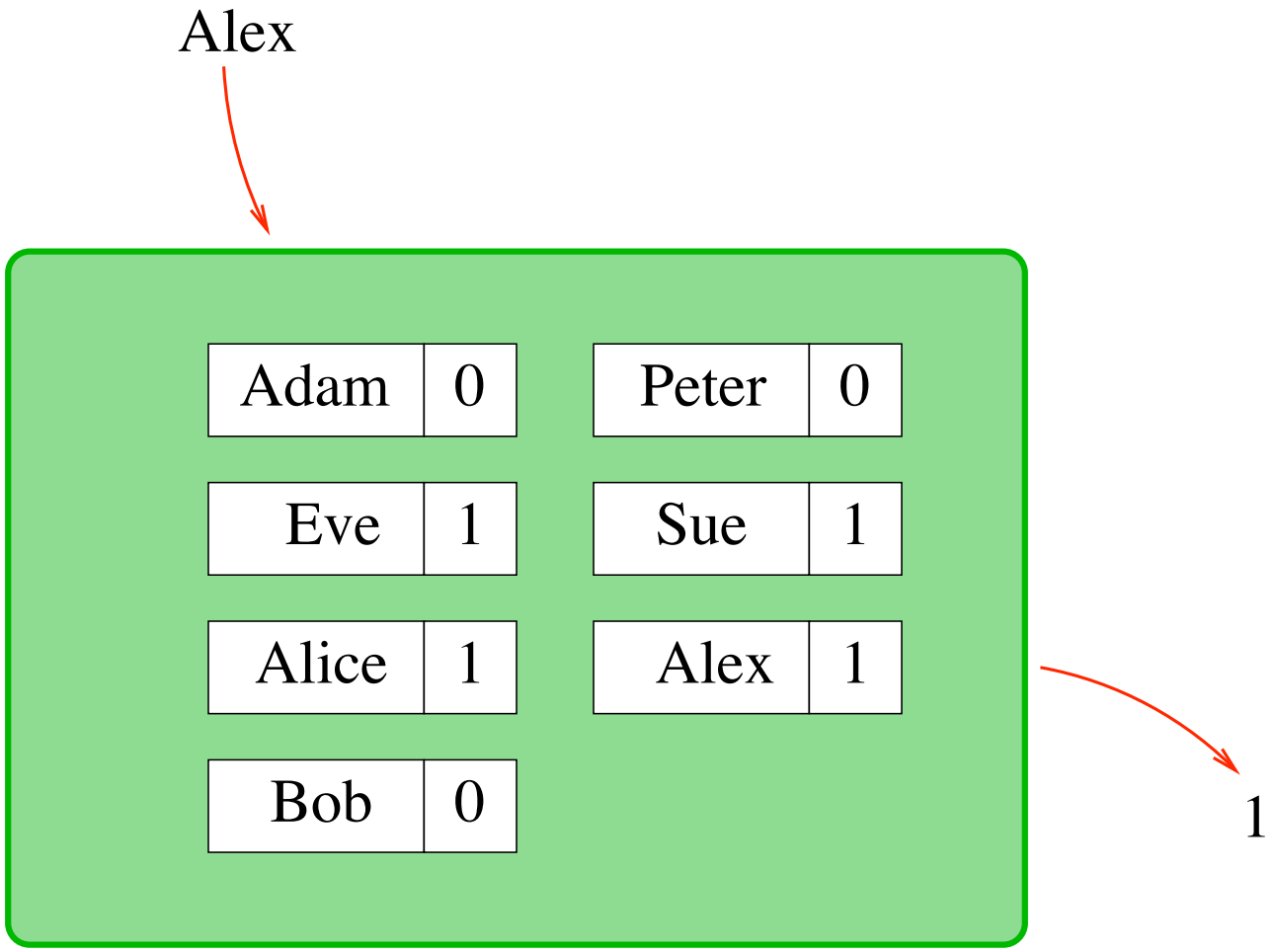
Retrieval



Retrieval



Retrieval

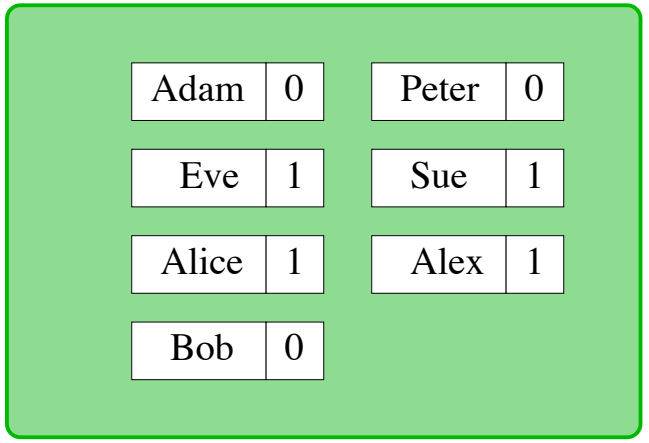


Retrieval

$S \subseteq U$.

Store $f : S \to R$, where w.l.o.g. $R = \{0,1\}^r$,

in data structure D_{retr}.

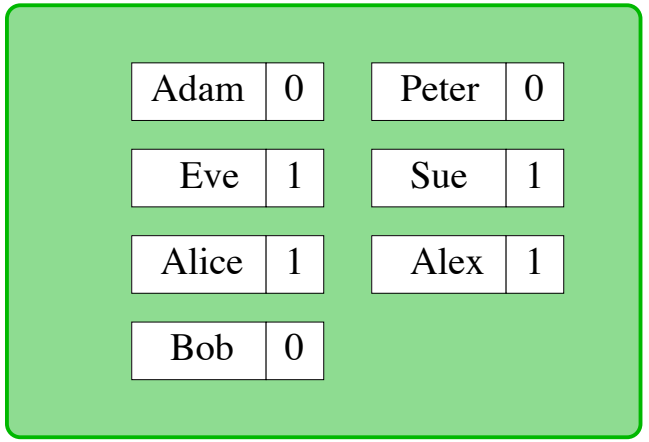


Retrieval

$S \subseteq U$.

Store $f : S \to R$, where w.l.o.g. $R = \{0,1\}^r$,

in data structure D_{retr}.



On $x \in S$, return $f(x)$.

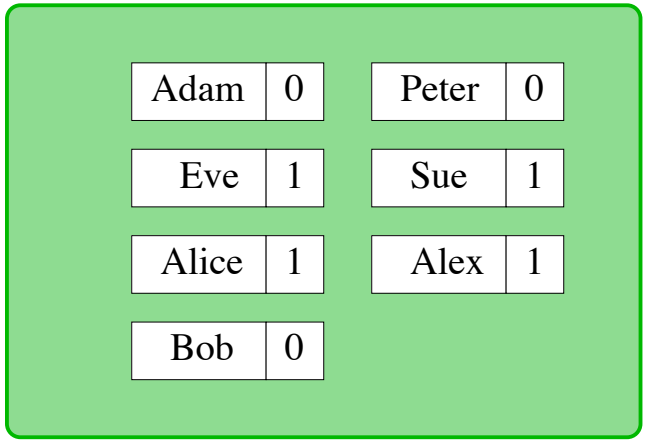
On $x \notin S$ return **any element of R**.

Retrieval

$S \subseteq U$.

Store $f : S \to R$, where w.l.o.g. $R = \{0,1\}^r$,

in data structure D_{retr}.



On $x \in S$, return $f(x)$.

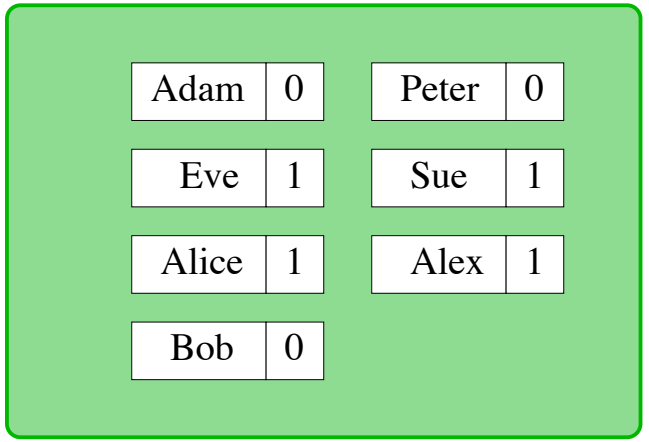
On $x \notin S$ return **any element of R**.

Issues:

Retrieval

$S \subseteq U$.

Store $f : S \to R$, where w.l.o.g. $R = \{0,1\}^r$,



in data structure D_{retr}.

On $x \in S$, return $f(x)$.

On $x \notin S$ return **any element of R**.

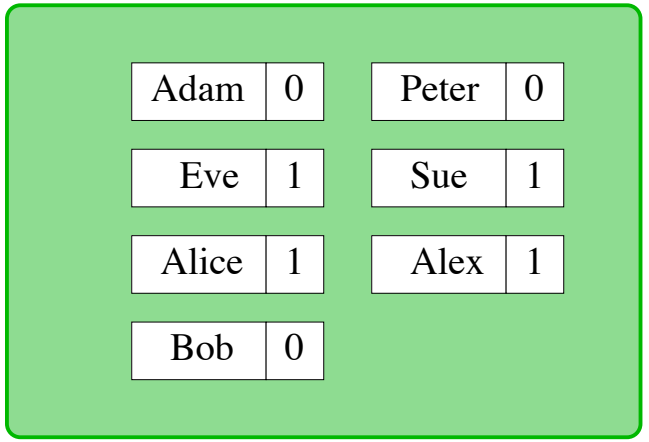
Issues: • Space for D_{retr}.

Retrieval

$S \subseteq U$.

Store $f : S \to R$, where w.l.o.g. $R = \{0,1\}^r$,

in data structure D_{retr}.



On $x \in S$, return $f(x)$.

On $x \notin S$ return **any element of R**.

Issues:
- Space for D_{retr}.
- Evaluation time $O(1)$ (unless said otherwise).

Simple solutions – Our results

Simple solutions – Our results

Simple:

Simple solutions – Our results

Simple:

- **Dictionaries**: nr bits **plus** space for dictionary.
  Solve in addition the problem "is $x \in S$?".

Simple solutions – Our results

Simple:

- **Dictionaries**: nr bits **plus** space for dictionary.
  Solve in addition the problem "is $x \in S$?".
- **Minimal perfect hash function** for S with range $[n]$:
  nr bits **plus** $\Theta(n)$ bits for hash function: Best possible: $n/\ln 2 \approx 1.44n$. [Hagerup, Tholey 2001].

Simple solutions – Our results

Simple:

- **Dictionaries**: nr bits **plus** space for dictionary.
  Solve in addition the problem "is $x \in S$?".
- **Minimal perfect hash function** for S with range $[n]$:
  nr bits **plus** $\Theta(n)$ bits for hash function: Best possible: $n/\ln 2 \approx 1.44n$. [Hagerup, Tholey 2001].

Our results:

- Space $(1 + e^{-k})nr$, evaluation time $O(k)$.

Simple solutions – Our results

Simple:

- **Dictionaries**: nr bits **plus** space for dictionary.
  Solve in addition the problem "is $x \in S$?".
- **Minimal perfect hash function** for S with range $[n]$:
  nr bits **plus** $\Theta(n)$ bits for hash function: Best possible: $n/\ln 2 \approx 1.44n$. [Hagerup, Tholey 2001].

Our results:

- Space $(1 + e^{-k})nr$, evaluation time $O(k)$.
- (Optimal) Space $n(r + o(1))$, evaluation time $O(\log n)$.

Simple solutions – Our results

Simple:

- **Dictionaries**: nr bits **plus** space for dictionary.
  Solve in addition the problem "is $x \in S$?".
- **Minimal perfect hash function** for S with range $[n]$:
  nr bits **plus** $\Theta(n)$ bits for hash function: Best possible: $n/\ln 2 \approx 1.44n$. [Hagerup, Tholey 2001].

Our results:

- Space $(1 + e^{-k})nr$, evaluation time $O(k)$.
- (Optimal) Space $n(r + o(1))$, evaluation time $O(\log n)$.

Assumption: Fully random hash functions for free.

Overview

- Introduction, Overview
- Basic approach: Hash-read-add
- Previous work
- Results
- Calkin's theorem on sparse random matrices
- Retrieval structure I
- Approximate membership
- Retrieval structure II: Construction in linear time
- Multiple-choice hashing and retrieval structures
- Summary

Basic approach: Hash-read-add

Basic approach: Hash-read-add

$$A : U \ni x \mapsto A(x) = \{h_1(x), \ldots, h_k(x)\} \subseteq [m] \text{ (distinct)}$$

Basic approach: Hash-read-add

$$A : U \ni x \mapsto A(x) = \{h_1(x), \ldots, h_k(x)\} \subseteq [m] \text{ (distinct)}$$

Assumption: $A(x)$ **fully random on S**.

Justification:
"Split-and-share" ($O(n^{1-\delta})$ extra space, not discussed here).

Basic approach: Hash-read-add

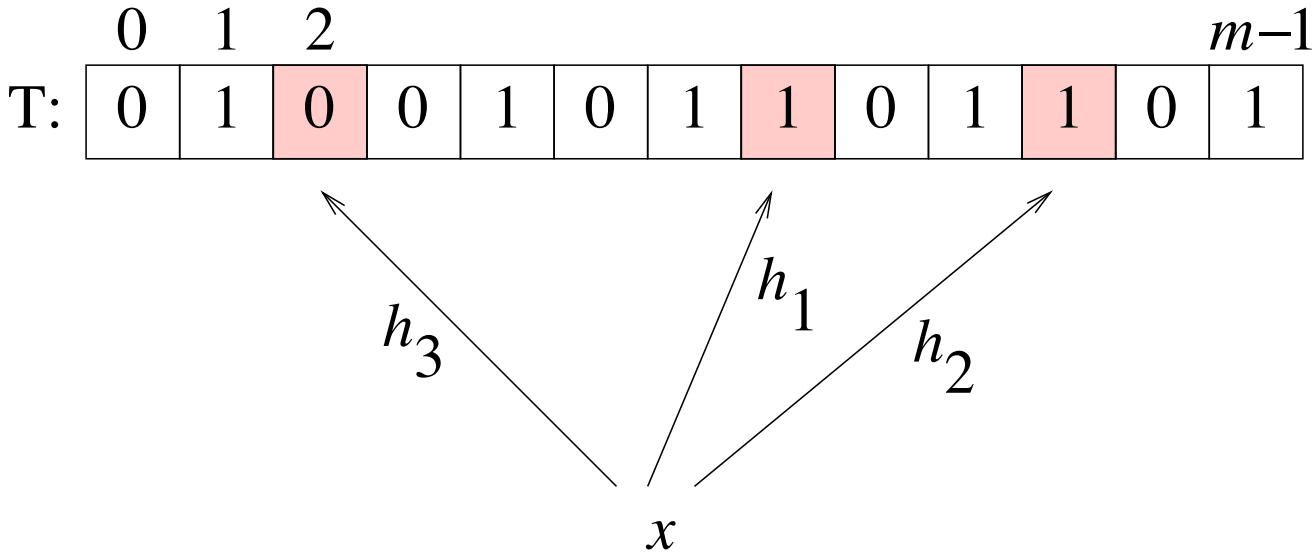
$A : U \ni x \mapsto A(x) = \{h_1(x), \ldots, h_k(x)\} \subseteq [m]$ (distinct)

Assumption: $A(x)$ **fully random on S**.

Justification:
"Split-and-share" ($O(n^{1-\delta})$ extra space, not discussed here).

Data structure: Array $T[0\,..\,m-1]$, entries from $R = \{0,1\}^r$.

Basic approach: Hash-read-add



$$f(x) = \bigoplus_{1 \le \ell \le k} T[h_\ell(x)]$$

Evaluation time: $O(k)$. Nonadaptive reads!

Basic approach: Hash-read-add

Sufficient condition for this to work:

Matrix $M_A = (p_{ij})_{\substack{1 \le i \le n \\ 0 \le j < m}} =$ with $p_{ij} = \begin{cases} 1 & \text{, if } j \in A_{x_i}, \\ 0 & \text{, otherwise,} \end{cases}$

Basic approach: Hash-read-add

Sufficient condition for this to work:

Matrix $M_A = (p_{ij})_{\substack{1 \le i \le n \\ 0 \le j < m}}$ = with $p_{ij} = \begin{cases} 1 & \text{, if } j \in A_{x_i}, \\ 0 & \text{, otherwise,} \end{cases}$

	$[m]$							
	0	1	2	3	4	5	6	7
$A(x_1)$:	0	1	0	0	1	0	0	1
$A(x_2)$:	1	0	1	0	0	0	1	0
$A(x_3)$:	0	1	0	0	1	0	1	0
$A(x_4)$:	0	1	1	1	0	0	0	0
$A(x_5)$:	1	0	1	0	0	1	0	0

Basic approach: Hash-read-add

Sufficient condition for this to work:

Matrix $M_A = (p_{ij})_{\substack{1 \le i \le n \\ 0 \le j < m}}$ = with $p_{ij} = \begin{cases} 1 & \text{, if } j \in A_{x_i}, \\ 0 & \text{, otherwise,} \end{cases}$

	$[m]$							
	0	1	2	3	4	5	6	7
$A(x_1)$:	0	1	0	0	1	0	0	1
$A(x_2)$:	1	0	1	0	0	0	1	0
$A(x_3)$:	0	1	0	0	1	0	1	0
$A(x_4)$:	0	1	1	1	0	0	0	0
$A(x_5)$:	1	0	1	0	0	1	0	0

has **full row rank**.

Basic approach: Hash-read-add

If M_A has **full row rank** . . . then

Basic approach: Hash-read-add

If M_A has **full row rank** . . . then the system

$$M_A \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

Basic approach: Hash-read-add

If M_A has **full row rank** . . . then the system

$$M_A \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

has a solution $(a_0, \ldots, a_{m-1})^{\mathsf{T}}$ (goes into $T[0\,..\,m-1]$).

Basic approach: Hash-read-add

If M_A has **full row rank** . . . then the system

$$M_A \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

has a solution $(a_0, \ldots, a_{m-1})^{\mathsf{T}}$ (goes into $T[0\,..\,m-1]$).

If $r = 1$: Linear algebra in $\mathbb{Z}_2$.

Basic approach: Hash-read-add

If M_A has **full row rank** . . . then the system

$$M_A \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

has a solution $(a_0, \ldots, a_{m-1})^{\mathsf{T}}$ (goes into $T[0\,..\,m-1]$).

If $r = 1$: Linear algebra in $\mathbb{Z}_2$.

If $r \geq 2$: Work in parallel in the components.

Basic approach: Hash-read-add

If M_A has **full row rank** . . . then the system

$$M_A \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

has a solution $(a_0, \ldots, a_{m-1})^{\mathsf{T}}$ (goes into $T[0\,..\,m-1]$).

If $r = 1$: Linear algebra in $\mathbb{Z}_2$.

If $r \geq 2$: Work in parallel in the components.

Alternative:

$R \subseteq \mathrm{GF}(q)$, any finite field (like $\mathbb{Z}_p$), calculate in $\mathrm{GF}(q)$.

Previous work

Previous work

[1] [Seiden, Hirschberg 1994]

"Ordered perfect hashing" (special case with $f(x_i) = i - 1$).

Propose hash-read-add-scheme, experiments, no analysis.

Previous work

[1] [Seiden, Hirschberg 1994]

"Ordered perfect hashing" (special case with $f(x_i) = i - 1$).

Propose hash-read-add-scheme, experiments, no analysis.

[2] [Majewski, Wormald, Havas, Czech 1996]

(Ordered) Perfect hashing, $r \geq 1$, $m = O(n)$.

Analysis via random (hyper)graph theory.

Previous work

[1] [Seiden, Hirschberg 1994]

"Ordered perfect hashing" (special case with $f(x_i) = i - 1$).

Propose hash-read-add-scheme, experiments, no analysis.

[2] [Majewski, Wormald, Havas, Czech 1996]

(Ordered) Perfect hashing, $r \geq 1$, $m = O(n)$.

Analysis via random (hyper)graph theory.

[3] [Chazelle, Kilian, Rubinfeld, Tal 2004]

Implicit in work on "Bloomier filter", $r \geq 1$, $m = O(n)$.

Ad-hoc analysis.

Previous work

In [2] + [3]:

Previous work

In [2] + [3]:

Use **sufficient** condition

> "**hypergraph G_A** (with hyperedges $A(x_i)$) is **acyclic**".

Previous work

In [2] + [3]:

Use **sufficient** condition

"**hypergraph G_A** (with hyperedges $A(x_i)$) is **acyclic**".

Equivalent:
Each nonempty subset of the $A(x_i)$'s covers at least one node only once.

Previous work

In [2] + [3]:

Use **sufficient** condition

"**hypergraph G_A** (with hyperedges $A(x_i)$) is **acyclic**".

Equivalent:
Each nonempty subset of the $A(x_i)$'s covers at least one node only once.

Equivalent:
M_A can be brought into **echelon form** by row and column **exchanges**.

Previous work

	$[m]$							
	7	3	1	6	0	2	4	5
$A(x_1)$:	1	0	1	0	0	0	1	0
$A(x_4)$:	0	1	1	0	0	1	0	0
$A(x_3)$:	0	0	1	1	0	0	1	0
$A(x_2)$:	0	0	0	1	1	1	0	0
$A(x_5)$:	0	0	0	0	1	1	0	1

Previous work

	$[m]$							
	7	3	1	6	0	2	4	5
$A(x_1)$:	1	0	1	0	0	0	1	0
$A(x_4)$:	0	1	1	0	0	1	0	0
$A(x_3)$:	0	0	1	1	0	0	1	0
$A(x_2)$:	0	0	0	1	1	1	0	0
$A(x_5)$:	0	0	0	0	1	1	0	1

Thresholds: (Acyclicity whp **if** $m \geq (1+\gamma)n > \gamma_k n$):

k	2	3	4	5	6	asympt.
γ_k	2	1.222	1.295	1.425	1.570	$\ln k$

Previous work

	$[m]$							
	7	3	1	6	0	2	4	5
$A(x_1)$:	1	0	1	0	0	0	1	0
$A(x_4)$:	0	1	1	0	0	1	0	0
$A(x_3)$:	0	0	1	1	0	0	1	0
$A(x_2)$:	0	0	0	1	1	1	0	0
$A(x_5)$:	0	0	0	0	1	1	0	1

Thresholds: (Acyclicity whp **if** $m \geq (1+\gamma)n > \gamma_k n$):

k	2	3	4	5	6	asympt.
γ_k	2	1.222	1.295	1.425	1.570	$\ln k$

Advantage: Solve linear system in time $O(nk)$.

New in this context: Calkin's theorem

New in this context: Calkin's theorem

Theorem [Calkin 1997] There are constants $\beta_k < 1$, $k = 3, 4, \ldots$, with the following properties:

- If $(1+\gamma) > \beta_k^{-1}$ and $m \geq (1+\gamma)n$ and M_A is as before, then M_A has full row rank with probability $1 - 1/n^{\varepsilon}$.

New in this context: Calkin's theorem

Theorem [Calkin 1997] There are constants $\beta_k < 1$, $k = 3, 4, \ldots$, with the following properties:

- If $(1+\gamma) > \beta_k^{-1}$ and $m \geq (1+\gamma)n$ and M_A is as before, then M_A has full row rank with probability $1 - 1/n^{\varepsilon}$.
- $\beta_k^{-1} \approx 1 + e^{-k}/\ln 2$, for growing k.

New in this context: Calkin's theorem

Theorem [Calkin 1997] There are constants $\beta_k < 1$, $k = 3, 4, \ldots$, with the following properties:

- If $(1+\gamma) > \beta_k^{-1}$ and $m \geq (1+\gamma)n$ and M_A is as before, then M_A has full row rank with probability $1 - 1/n^{\varepsilon}$.
- $\beta_k^{-1} \approx 1 + e^{-k}/\ln 2$, for growing k.

Thresholds:

k	2	3	4	5	6	asympt.
β_k^{-1}	2	1.1243	1.034	1.011	1.0038	$1 + e^{-k}/\ln 2$
γ_k	2	1.222	1.295	1.425	1.570	$\ln k$

New retrieval structures

New retrieval structures

Theorem 1

A retrieval data structure using the hash-read-add scheme with k accesses for a query and space $(1+\gamma)rn$ bits can be built if $1+\gamma > \beta_k^{-1} \approx 1 + e^{-k}/\ln 2$.
Construction time: $O(n^3)$.

New retrieval structures

Theorem 1

A retrieval data structure using the hash-read-add scheme with k accesses for a query and space $(1+\gamma)rn$ bits can be built if $1+\gamma > \beta_k^{-1} \approx 1 + e^{-k}/\ln 2$.
Construction time: $O(n^3)$.

Proof: Existence: Calkin. Construction: Solve a linear system.

New retrieval structures

Theorem 1

A retrieval data structure using the hash-read-add scheme with k accesses for a query and space $(1+\gamma)rn$ bits can be built if $1+\gamma > \beta_k^{-1} \approx 1 + e^{-k}/\ln 2$.
Construction time: $O(n^3)$.

Proof: Existence: Calkin. Construction: Solve a linear system.

Theorem 2

. . . same . . .
Construction time $O(n^{1+\delta})$.

New retrieval structures

Theorem 1

A retrieval data structure using the hash-read-add scheme with k accesses for a query and space $(1+\gamma)rn$ bits can be built if $1+\gamma > \beta_k^{-1} \approx 1 + e^{-k}/\ln 2$.
Construction time: $O(n^3)$.

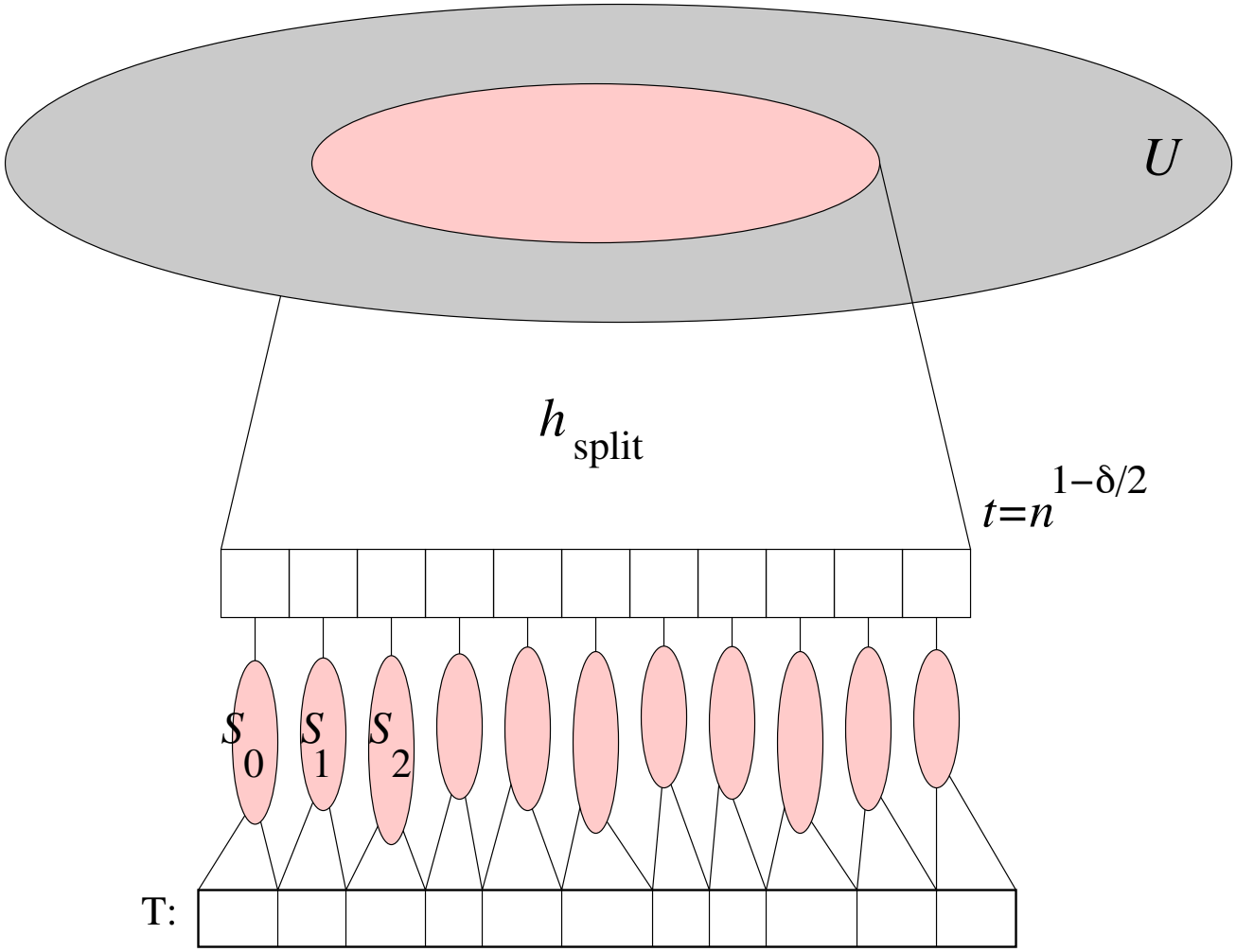
Proof: Existence: Calkin. Construction: Solve a linear system.

Theorem 2

. . . same . . .
Construction time $O(n^{1+\delta})$.

Proof: Theorem 1 plus "splitting".

Construction time $O(n^{1+\delta})$



Construction time $O(n^{1+\delta})$

Construction time $O(n^{1+\delta})$, for $\delta > 0$ constant:

"Split" .

Construction time $O(n^{1+\delta})$

Construction time $O(n^{1+\delta})$, for $\delta > 0$ constant:

"Split" .

Introduce extra first level of hashing, using hash function $h_{\text{split}} : U \to [t]$.

Splits S into $t = n^{1-\delta/2}$ chunks $S_0, \ldots, S_{t-1}$ of size $O(n^{\delta/2})$.

Construction time $O(n^{1+\delta})$

Construction time $O(n^{1+\delta})$, for $\delta > 0$ constant:

"Split" .

Introduce extra first level of hashing, using hash function $h_{\text{split}} : U \to [t]$.

Splits S into $t = n^{1-\delta/2}$ chunks $S_0, \ldots, S_{t-1}$ of size $O(n^{\delta/2})$.

Construct separate retrieval data structure for each of the chunks: construction time

$$t \cdot O((n^{\delta/2})^3) = O(n^{1+\delta}).$$

Extra space: $o(n)$.

Construction time $O(n^{1+\delta})$

Construction time $O(n^{1+\delta})$, for $\delta > 0$ constant:

"Split" .

Introduce extra first level of hashing, using hash function $h_{\text{split}} : U \to [t]$.

Splits S into $t = n^{1-\delta/2}$ chunks $S_0, \ldots, S_{t-1}$ of size $O(n^{\delta/2})$.

Construct separate retrieval data structure for each of the chunks: construction time

$$t \cdot O((n^{\delta/2})^3) = O(n^{1+\delta}).$$

Extra space: $o(n)$.
Retrieval for y: access retrieval data structure for $S_{h_{\text{split}}(y)}$.

Approximate membership

Approximate membership

$x \in S \;\rightarrow\;$ Answer "yes"

$x \notin S \;\rightarrow\;$ **Pr**(Answer "no") $\geq 1 - 2^{-r}$

Approximate membership

$x \in S \;\rightarrow$ Answer "yes"

$x \notin S \;\rightarrow$ **Pr**(Answer "no") $\geq 1 - 2^{-r}$

Standard implementation: **Bloom filters**. Space $\approx nr / \ln 2$.

Approximate membership

$x \in S \;\rightarrow$ Answer "yes"

$x \notin S \;\rightarrow$ **Pr**(Answer "no") $\geq 1 - 2^{-r}$

Standard implementation: **Bloom filters**. Space $\approx nr / \ln 2$.

Can show ([Carter *et al.* 1978]): Need $\geq nr - O(1)$ bits.

Approximate membership

$x \in S \;\rightarrow$ Answer "yes"

$x \notin S \;\rightarrow$ **Pr**(Answer "no") $\geq 1 - 2^{-r}$

Standard implementation: **Bloom filters**. Space $\approx nr / \ln 2$.

Can show ([Carter *et al.* 1978]): Need $\geq nr - O(1)$ bits.

Or (folklore):

Use (minimal) **perfect hashing** to store an r-bit **fingerprint** for $x \in S$.

Approximate membership

$x \in S \;\rightarrow$ Answer "yes"

$x \notin S \;\rightarrow$ **Pr**(Answer "no") $\geq 1 - 2^{-r}$

Standard implementation: **Bloom filters**. Space $\approx nr / \ln 2$.

Can show ([Carter *et al.* 1978]): Need $\geq nr - O(1)$ bits.

Or (folklore):

Use (minimal) **perfect hashing** to store an r-bit **fingerprint** for $x \in S$.

Space: $nr + O(n)$ bits.

Approximate membership

General construction:

Approximate membership

General construction:

Assume **any** algorithm for retrieval structure for range $R = \{0,1\}^r$

Approximate membership

General construction:

Assume **any** algorithm for retrieval structure for range $R = \{0,1\}^r$ **plus** one fully random hash function $q\colon U \to R$ (fingerprint).

Approximate membership

General construction:

Assume **any** algorithm for retrieval structure for range $R = \{0,1\}^r$ **plus** one fully random hash function $q\colon U \to R$ (fingerprint).

Given S, build retrieval structure D_{retr} for $f = q|_S$.

Approximate membership

General construction:

Assume **any** algorithm for retrieval structure for range $R = \{0,1\}^r$ **plus** one fully random hash function $q\colon U \to R$ (fingerprint).

Given S, build retrieval structure D_{retr} for $f = q|_S$.

On query y:

> Retrieve $s = D_{\text{retr}}(y)$;
> answer "yes" if $q(y) = s$, "no" otherwise.

Approximate membership

General construction:

Assume **any** algorithm for retrieval structure for range $R = \{0,1\}^r$ **plus** one fully random hash function $q\colon U \to R$ (fingerprint).

Given S, build retrieval structure D_{retr} for $f = q|_S$.

On query y:

> Retrieve $s = D_{\text{retr}}(y)$;
> answer "yes" if $q(y) = s$, "no" otherwise.

Performance: Error probability $\leq 2^{-r}$.

Approximate membership

General construction:

Assume **any** algorithm for retrieval structure for range $R = \{0,1\}^r$ **plus** one fully random hash function $q\colon U \to R$ (fingerprint).

Given S, build retrieval structure D_{retr} for $f = q|_S$.

On query y:

> Retrieve $s = D_{\text{retr}}(y)$;
> answer "yes" if $q(y) = s$, "no" otherwise.

Performance: Error probability $\leq 2^{-r}$.

New: Space $(1 + e^{-k})nr$ bits, evaluation time $O(k)$.

Approximate membership

General construction:

Assume **any** algorithm for retrieval structure for range $R = \{0,1\}^r$ **plus** one fully random hash function $q\colon U \to R$ (fingerprint).

Given S, build retrieval structure D_{retr} for $f = q|_S$.

On query y:

> Retrieve $s = D_{\text{retr}}(y)$;
> answer "yes" if $q(y) = s$, "no" otherwise.

Performance: Error probability $\leq 2^{-r}$.

New: Space $(1 + e^{-k})nr$ bits, evaluation time $O(k)$.

Construction time $O(n^3)$ resp. $O(n^{1+\delta})$.

Retrieval: Construction in linear time

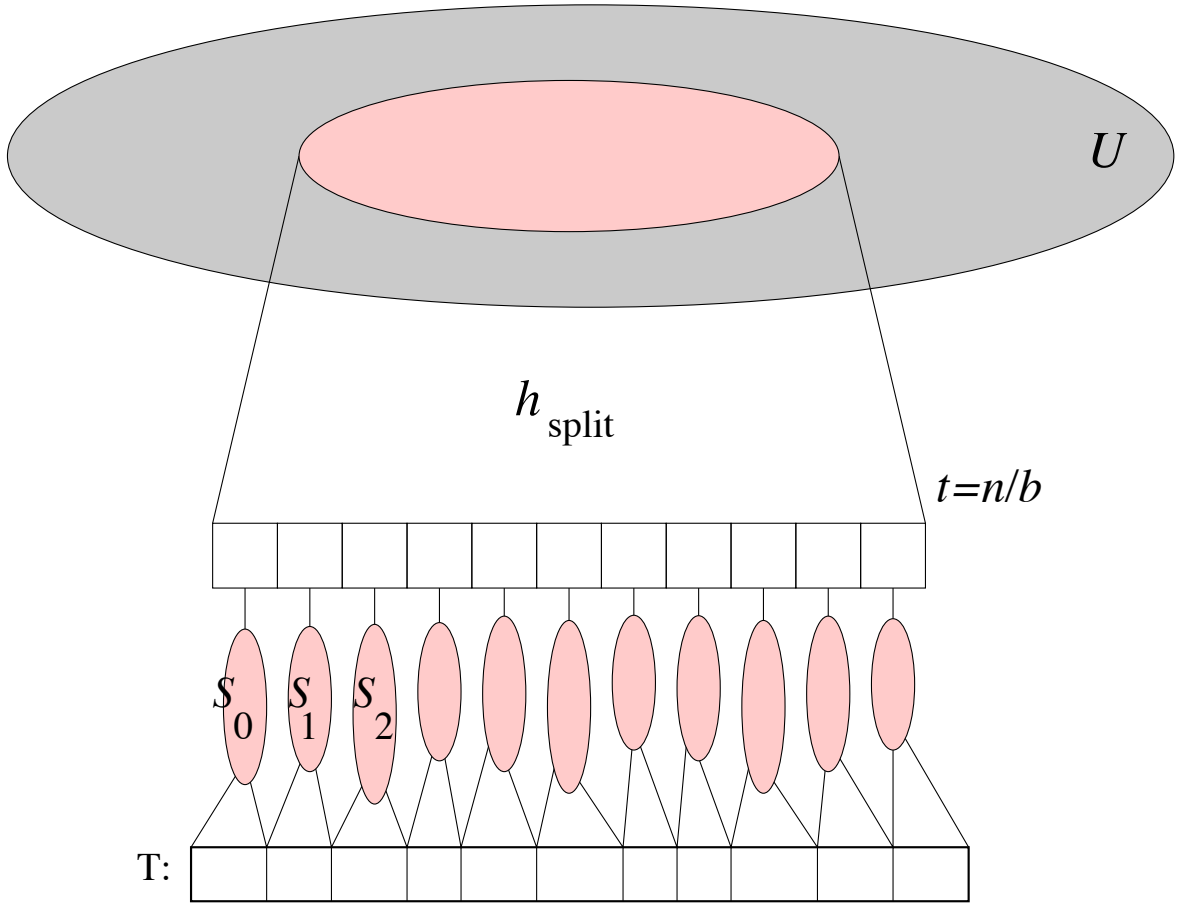
Retrieval: Construction in linear time

Theoretical construction, works for very large n.

Retrieval: Construction in linear time

Theoretical construction, works for very large n.

Extra level of hashing: $h_{\text{split}} : U \to [t]$.



Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construct separate retrieval data structure for each chunk:
Allocate space $b' = (1+\gamma)b$ for each chunk, $1 + \gamma > \beta_k^{-1}$.

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construct separate retrieval data structure for each chunk:
Allocate space $b' = (1+\gamma)b$ for each chunk, $1+\gamma > \beta_k^{-1}$.

Construction time $\boldsymbol{O(b)}$ **!**

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construct separate retrieval data structure for each chunk:
Allocate space $b' = (1+\gamma)b$ for each chunk, $1 + \gamma > \beta_k^{-1}$.

Construction time $\boldsymbol{O(b)}$ **!**

Why?

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construct separate retrieval data structure for each chunk:
Allocate space $b' = (1+\gamma)b$ for each chunk, $1+\gamma > \beta_k^{-1}$.

Construction time $\boldsymbol{O(b)}$ **!**

Why? Can arrange that matrix size $((1+\gamma)b)^2$ is $< \frac{1}{2}\log n$.
Use table-lookup for the linear algebra.

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construct separate retrieval data structure for each chunk:
Allocate space $b' = (1+\gamma)b$ for each chunk, $1+\gamma > \beta_k^{-1}$.

Construction time $\boldsymbol{O(b)}$ **!**

Why? Can arrange that matrix size $((1+\gamma)b)^2$ is $< \frac{1}{2}\log n$.
Use table-lookup for the linear algebra.

Total construction time: $O(n)$.

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construct separate retrieval data structure for each chunk: Allocate space $b' = (1+\gamma)b$ for each chunk, $1+\gamma > \beta_k^{-1}$.

Construction time $\boldsymbol{O(b)}$ **!**

Why? Can arrange that matrix size $((1+\gamma)b)^2$ is $< \frac{1}{2}\log n$. Use table-lookup for the linear algebra.

Total construction time: $O(n)$. Total space $(1+\gamma)nr + o(n)$ bits.

Construction in linear time

Chunk size: $b = \frac{1}{2}\sqrt{\log n}$.

Number of chunks: $t = n/b$.

h_{split} splits S into chunks $S_0, \ldots, S_{t-1}$ of expected size b.

Construct separate retrieval data structure for each chunk: Allocate space $b' = (1+\gamma)b$ for each chunk, $1+\gamma > \beta_k^{-1}$.

Construction time $\boldsymbol{O(b)}$ **!**

Why? Can arrange that matrix size $((1+\gamma)b)^2$ is $< \frac{1}{2}\log n$. Use table-lookup for the linear algebra.

Total construction time: $O(n)$. Total space $(1+\gamma)nr + o(n)$ bits.

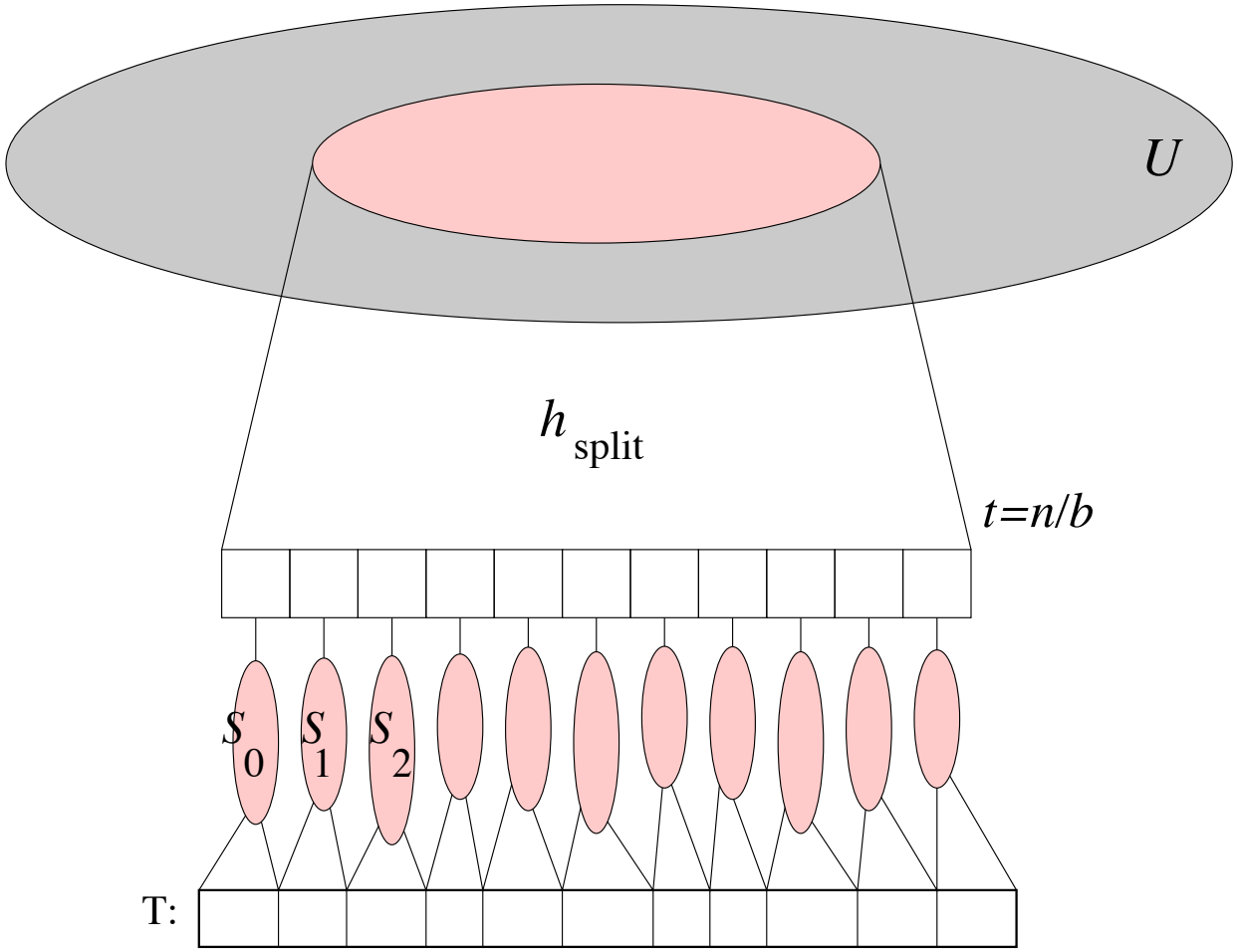
Done?

Construction in linear time

No!

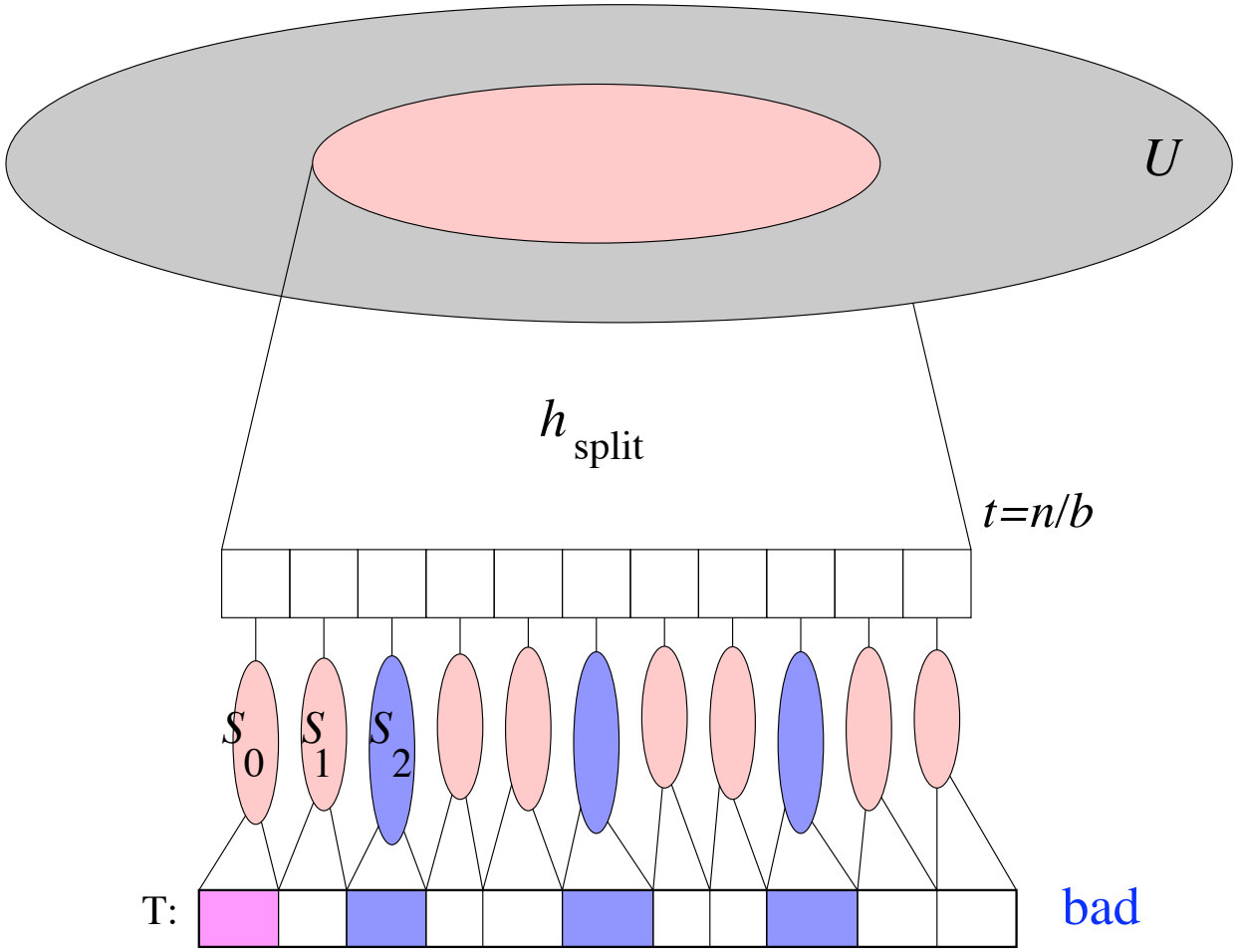
Construction in linear time

No! "Bad chunks":



Construction in linear time

No! "Bad chunks": (1) Overflow (2) Construction fails.



Construction in linear time

Nice: #(keys in bad chunk) $= o(n)$.

Construction in linear time

Nice: #(keys in bad chunk) $= o(n)$.

The (keys from) bad chunks are accommodated in a secondary structure with table $T'[0..o(n)]$, hash functions $h'_1, \ldots, h'_3$, with $o(n)$ construction time (e.g. [Chazelle *et al.* 2004]).

Construction in linear time

Nice: #(keys in bad chunk) $= o(n)$.

The (keys from) bad chunks are accommodated in a secondary structure with table $T'[0..o(n)]$, hash functions $h'_1, \ldots, h'_3$, with $o(n)$ construction time (e.g. [Chazelle *et al.* 2004]).

Flag: $v[i] = 0$ if chunk i is bad and $v[i] = 1$ otherwise.

Construction in linear time

Segment for chunk S_i in T: $T[d_i \,..\, d_{i+1} - 1]$.

Construction in linear time

Segment for chunk S_i in T: $T[d_i \mathinner{..} d_{i+1} - 1]$.

Lookup operation: $i \leftarrow h_{\text{split}}(x)$, then . . .

$$f(x) = v[i] \cdot \bigoplus_{1 \le \ell \le k} T[h_\ell(x) + d_i] \;\oplus\; \overline{v[i]} \cdot \bigoplus_{1 \le \ell \le 3} T'[h'_\ell(x)]$$

Construction in linear time

Segment for chunk S_i in T: $T[d_i \mathinner{..} d_{i+1} - 1]$.

Lookup operation: $i \leftarrow h_{\text{split}}(x)$, then . . .

$$\boldsymbol{f(x) = v[i] \cdot \bigoplus_{1 \le \ell \le k} T[h_\ell(x) + d_i] \;\oplus\; \overline{v[i]} \cdot \bigoplus_{1 \le \ell \le 3} T'[h'_\ell(x)]}$$

Time $O(k)$. Nonadaptive reads!

Almost optimal space, logarithmic evaluation time

Almost optimal space, logarithmic evaluation time

Use random sets $A(x) = \{h_1(x), \ldots, h_{k(x)}(x)\} \subseteq [\boldsymbol{n}]$,
with $\mathbf{E}(k(x)) = \Theta(\log n)$, binomially distributed.

Almost optimal space, logarithmic evaluation time

Use random sets $A(x) = \{h_1(x), \ldots, h_{k(x)}(x)\} \subseteq [n]$,
with $\mathbf{E}(k(x)) = \Theta(\log n)$, binomially distributed.

[Cooper, 1999] $\Rightarrow$

$$\mathbf{Pr}(M_A \text{ is regular}) \geq 0.28.$$

Almost optimal space, logarithmic evaluation time

Use random sets $A(x) = \{h_1(x), \ldots, h_{k(x)}(x)\} \subseteq [n]$,
with $\mathbf{E}(k(x)) = \Theta(\log n)$, binomially distributed.

[Cooper, 1999] $\Rightarrow$

$$\mathbf{Pr}(M_A \text{ is regular}) \geq 0.28.$$

$O(\log n)$ trials will find good A with probability $1 - 1/n^{O(1)}$.

Almost optimal space, logarithmic evaluation time

Use random sets $A(x) = \{h_1(x), \ldots, h_{k(x)}(x)\} \subseteq [n]$,
with $\mathbf{E}(k(x)) = \Theta(\log n)$, binomially distributed.

[Cooper, 1999] $\Rightarrow$

$$\mathbf{Pr}(M_A \text{ is regular}) \geq 0.28.$$

$O(\log n)$ trials will find good A with probability $1 - 1/n^{O(1)}$.

(Must store index of this A, $O(\log \log n)$ additional bits.)

Almost optimal space, logarithmic evaluation time

Use random sets $A(x) = \{h_1(x), \ldots, h_{k(x)}(x)\} \subseteq [n]$,
with $\mathbf{E}(k(x)) = \Theta(\log n)$, binomially distributed.

[Cooper, 1999] $\Rightarrow$

$$\mathbf{Pr}(M_A \text{ is regular}) \geq 0.28.$$

$O(\log n)$ trials will find good A with probability $1 - 1/n^{O(1)}$.

(Must store index of this A, $O(\log \log n)$ additional bits.)

Space for table T: nr **bits** **(optimal)**.

Almost optimal space, logarithmic evaluation time

Use random sets $A(x) = \{h_1(x), \ldots, h_{k(x)}(x)\} \subseteq [n]$,
with $\mathbf{E}(k(x)) = \Theta(\log n)$, binomially distributed.

[Cooper, 1999] $\Rightarrow$

$$\mathbf{Pr}(M_A \text{ is regular}) \geq 0.28.$$

$O(\log n)$ trials will find good A with probability $1 - 1/n^{O(1)}$.

(Must store index of this A, $O(\log \log n)$ additional bits.)

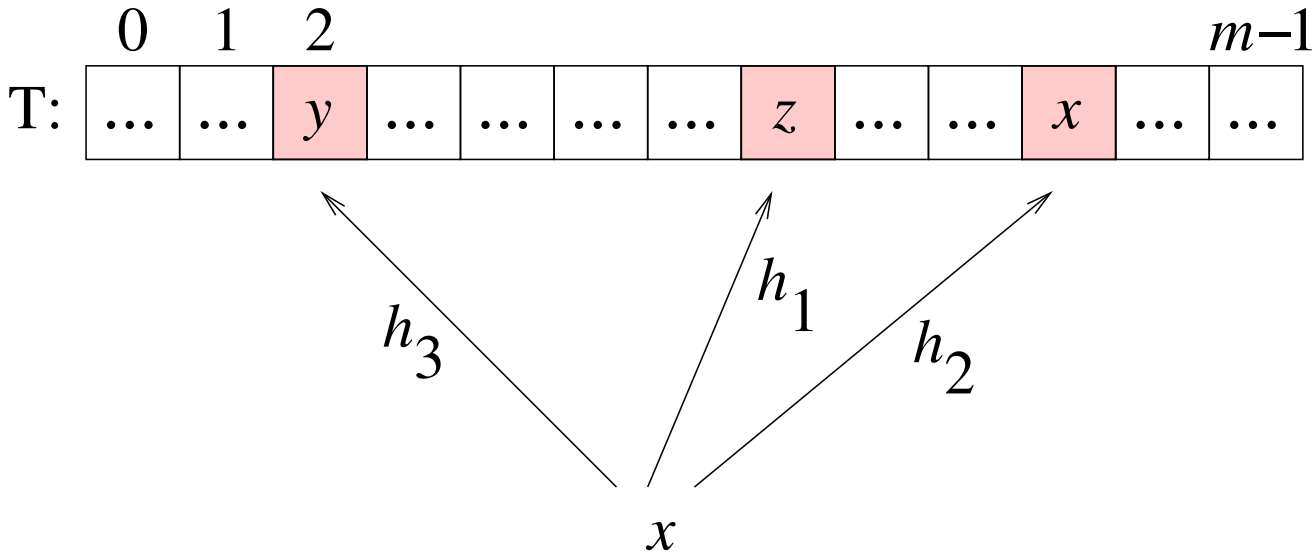
Space for table T: nr **bits** **(optimal)**.

Evaluation time: $O(\log n)$.

Multiple-choice hashing and retrieval structures

Multiple-choice hashing and retrieval structures

Dictionary structure with multiple choices and retrieval:

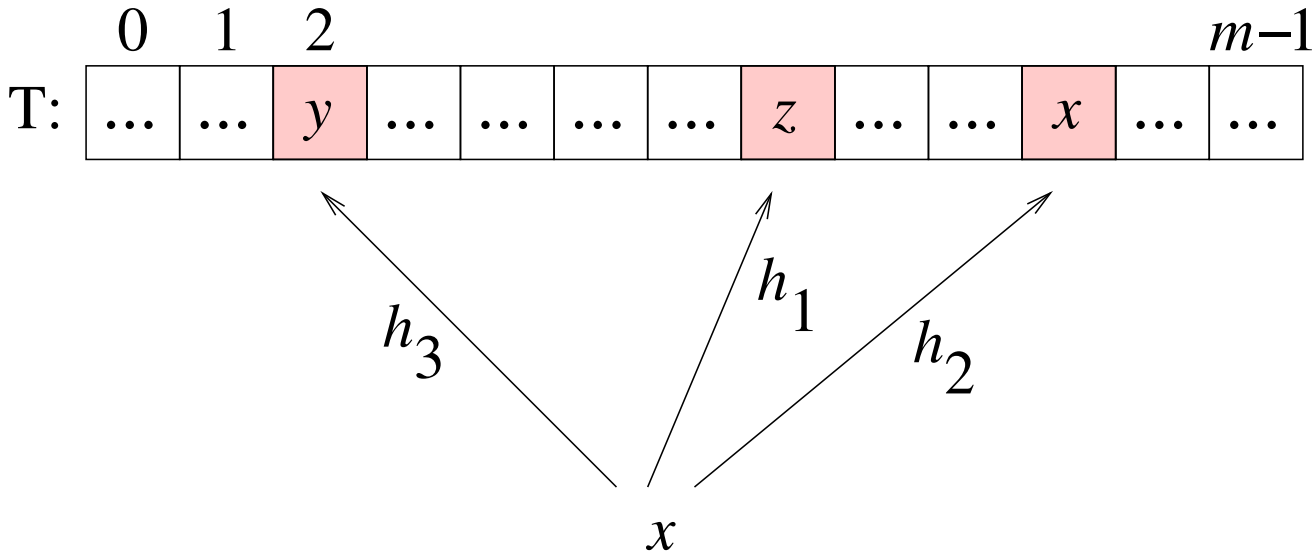


$A(x) = \{h_1(x), h_2(x), h_3(x)\}$.

x in $T[h_1(x)]$ or $T[h_2(x)]$ or $T[h_3(x)]$.

Multiple-choice hashing and retrieval structures

Dictionary structure with multiple choices and retrieval:



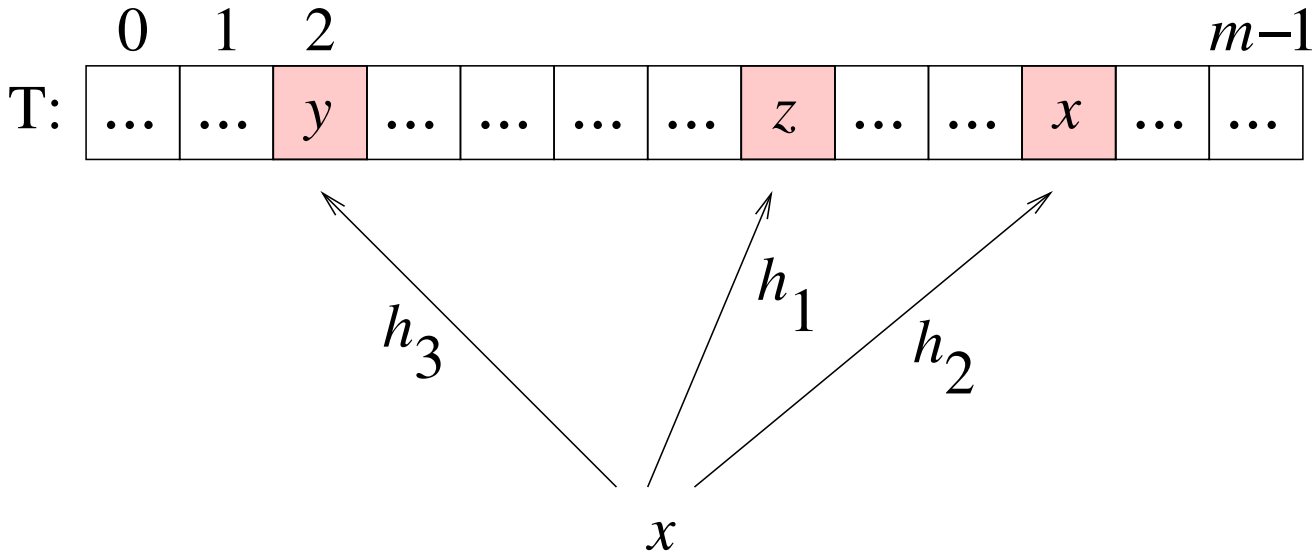
$A(x) = \{h_1(x), h_2(x), h_3(x)\}$.

x in $T[h_1(x)]$ or $T[h_2(x)]$ or $T[h_3(x)]$.

Similarity to retrieval structure!

Multiple-choice hashing and retrieval structures

Dictionary structure with multiple choices and retrieval:



$A(x) = \{h_1(x), h_2(x), h_3(x)\}$.

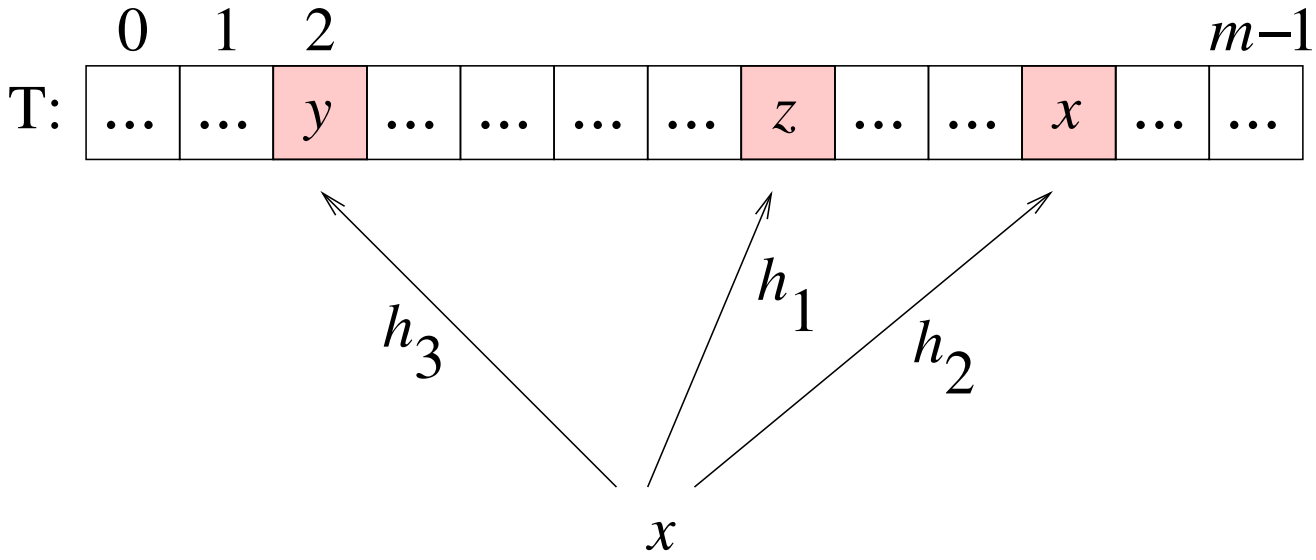
x in $T[h_1(x)]$ or $T[h_2(x)]$ or $T[h_3(x)]$.

Similarity to retrieval structure!

Many variants: ". . . cuckoo hashing", "balanced allocation".

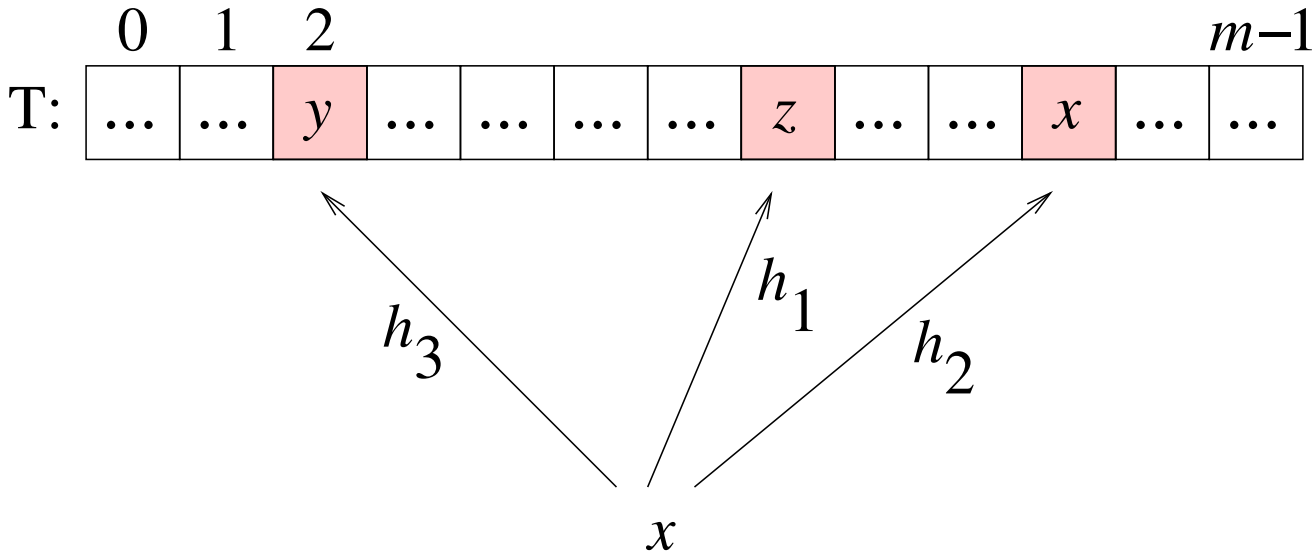
Multiple-choice hashing and retrieval structures

Dictionary structure with multiple choices and retrieval:



Multiple-choice hashing and retrieval structures

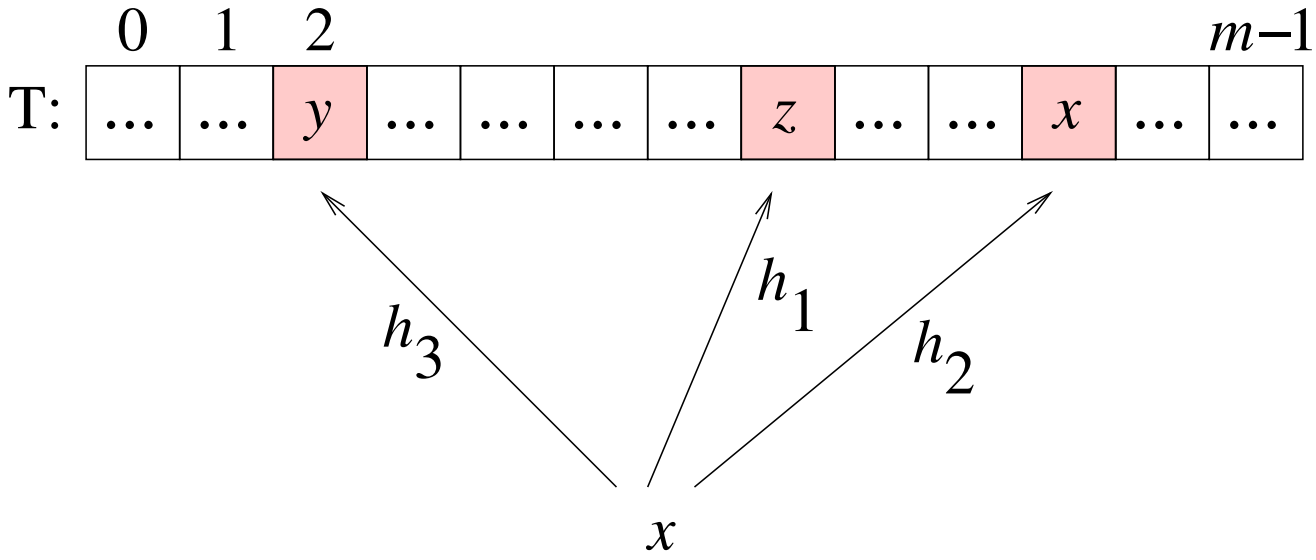
Dictionary structure with multiple choices and retrieval:



Definition: A is **suitable for** $S = \{x_1, \ldots, x_n\}$
if there is a mapping σ: $\xrightarrow{1-1}$ $[m]$ with $\sigma(i) \in A(x_i)$.

Multiple-choice hashing and retrieval structures

Dictionary structure with multiple choices and retrieval:



Definition: A is **suitable for** $S = \{x_1, \ldots, x_n\}$
if there is a mapping $\sigma: \xrightarrow{1-1} [m]$ with $\sigma(i) \in A(x_i)$.
(. . . if and only if one can store x_i in a slot in $A(x_i)$, without collision).

Multiple-choice hashing and retrieval structures

Observation 1:

M_A has full row rank $\Rightarrow$ A is suitable for S.

Multiple-choice hashing and retrieval structures

Observation 1:

M_A has full row rank $\Rightarrow$ A is suitable for S.

Reason: M_A has a regular square submatrix M'_A.

	$[m]$							
	0	1	2	3	4	5	6	7
$A(x_1)$:	0	1	0	0	1	0	0	1
$A(x_2)$:	1	0	1	0	0	0	1	0
$A(x_3)$:	0	1	0	0	1	0	1	0
$A(x_4)$:	0	1	1	1	0	0	0	0
$A(x_5)$:	1	0	1	0	0	1	0	0

$$\det(M'_A) = \sum_{\pi \in S_n} \operatorname{sign}(\pi) \prod_{1 \le i \le n} p_{i\pi(i)} \neq 0.$$

Multiple-choice hashing and retrieval structures

Consequence:

Calkin's result gives sufficient condition for dictionary structures to exist:

Multiple-choice hashing and retrieval structures

Consequence:

Calkin's result gives sufficient condition for dictionary structures to exist:

Space $(1+\gamma)n$, $1+\gamma > \beta_k^{-1}$, k probes.

Construction time $O(n^{1+\delta})$.

Multiple-choice hashing and retrieval structures

Consequence:

Calkin's result gives sufficient condition for dictionary structures to exist:

Space $(1+\gamma)n$, $1+\gamma > \beta_k^{-1}$, k probes.

Construction time $O(n^{1+\delta})$.

(Tiny improvement of space bound of [Fotakis et al., 2005], d-ary cuckoo hashing. — Conjecture: Optimal.)

Multiple-choice hashing and retrieval structures

Observation 2: A suitable for S $\Rightarrow$

	$[m]$							
	0	1	2	3	4	5	6	7
$A(x_1)$:	0	1	0	0	1	0	0	1
$A(x_2)$:	0	1	1	0	0	0	1	0
$A(x_3)$:	0	1	1	0	0	0	1	0
$A(x_4)$:	0	1	1	1	0	0	0	0
$A(x_5)$:	1	0	1	0	0	1	0	0

Multiple-choice hashing and retrieval structures

Observation 2: A suitable for S $\Rightarrow$

	$[m]$							
	0	1	2	3	4	5	6	7
$A(x_1)$:	0	1	0	0	1	0	0	1
$A(x_2)$:	0	1	1	0	0	0	1	0
$A(x_3)$:	0	1	1	0	0	0	1	0
$A(x_4)$:	0	1	1	1	0	0	0	0
$A(x_5)$:	1	0	1	0	0	1	0	0

M_A has a square submatrix M'_A with a nonvanishing term $\mathrm{sign}(\pi) \prod_{1 \le i \le n} p_{i\pi(i)}$ in the determinant . . .

Multiple-choice hashing and retrieval structures

Observation 2: A suitable for S $\Rightarrow$

	$[m]$							
	0	1	2	3	4	5	6	7
$A(x_1)$:	0	8	0	0	4	0	0	5
$A(x_2)$:	0	6	9	0	0	0	2	0
$A(x_3)$:	0	9	8	0	0	0	5	0
$A(x_4)$:	0	7	7	4	0	0	0	0
$A(x_5)$:	5	0	2	0	0	7	0	0

If we replace the 1's in M_A with random elements of $\mathbb{F} - \{0\}$ (finite field $\mathbb{F}$), yielding $M_A(\mathbb{F})$, **then** $M_A(\mathbb{F})$ has full row rank with probability $\geq 1 - \frac{n}{|\mathbb{F}|}$ (Schwartz-Zippel Theorem).

Multiple-choice hashing and retrieval structures

Resulting retrieval function ($g_1(x), \ldots, g_k(x)$ are random hash values, calculate in $\mathbb{F}$):

$$f(x) = \sum_{1 \leq \ell \leq k} g_\ell(x_i) \cdot T[h_\ell(x)]$$

Multiple-choice hashing and retrieval structures

Resulting retrieval function $(g_1(x), \ldots, g_k(x)$ are random hash values, calculate in $\mathbb{F}$):

$$\boldsymbol{f(x) = \sum_{1 \le \ell \le k} g_\ell(x_i) \cdot T[h_\ell(x)]}$$

With splitting trick applicable if $|R| = |\mathbb{F}| \geq n^\delta$.

Multiple-choice hashing and retrieval structures

Resulting retrieval function ($g_1(x), \ldots, g_k(x)$ are random hash values, calculate in $\mathbb{F}$):

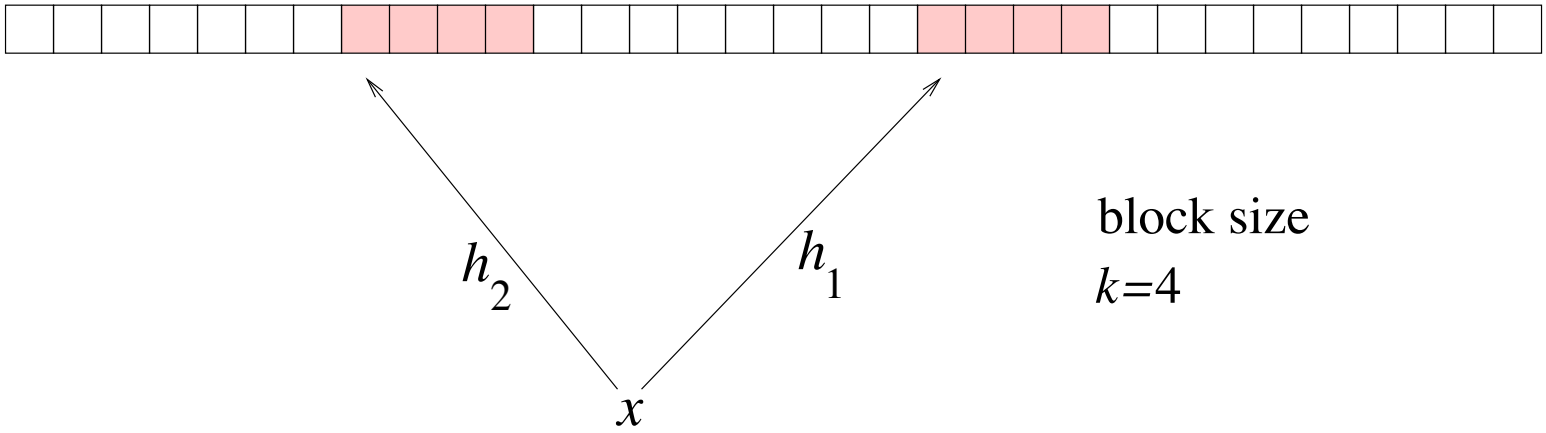
$$\boxed{f(x) = \sum_{1 \le \ell \le k} g_\ell(x_i) \cdot T[h_\ell(x)]}$$

With splitting trick applicable if $|R| = |\mathbb{F}| \geq n^\delta$.

Construction time: $O(n^3)$ or $O(n^{1+\delta})$.

Multiple-choice hashing and retrieval structures

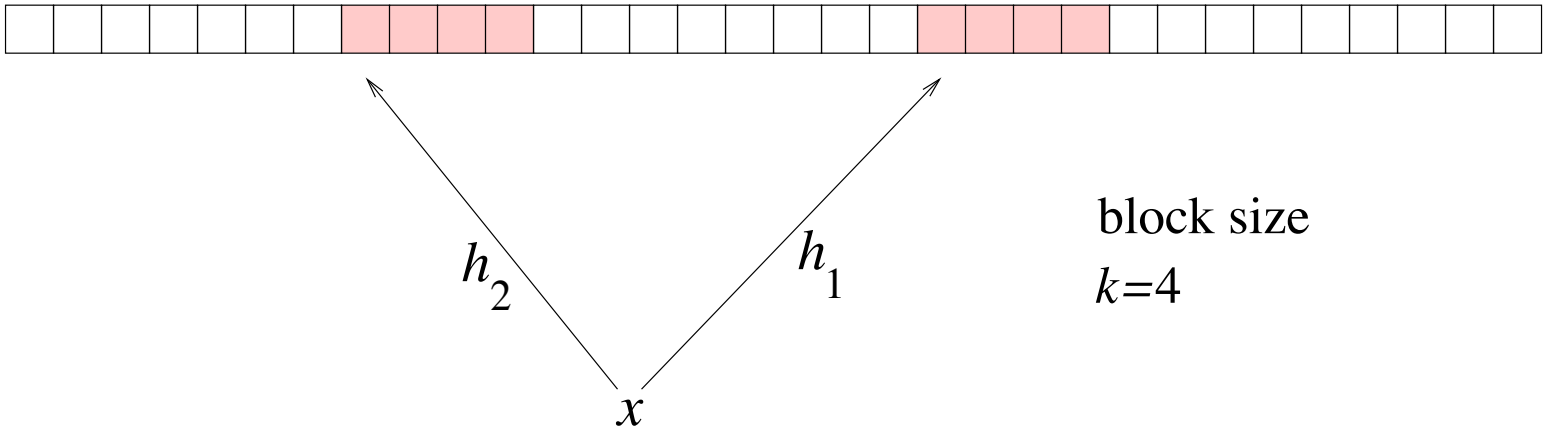
Blocked cuckoo hashing:



Dictionaries with accesses in T only at two intervals.

Multiple-choice hashing and retrieval structures

Blocked cuckoo hashing:

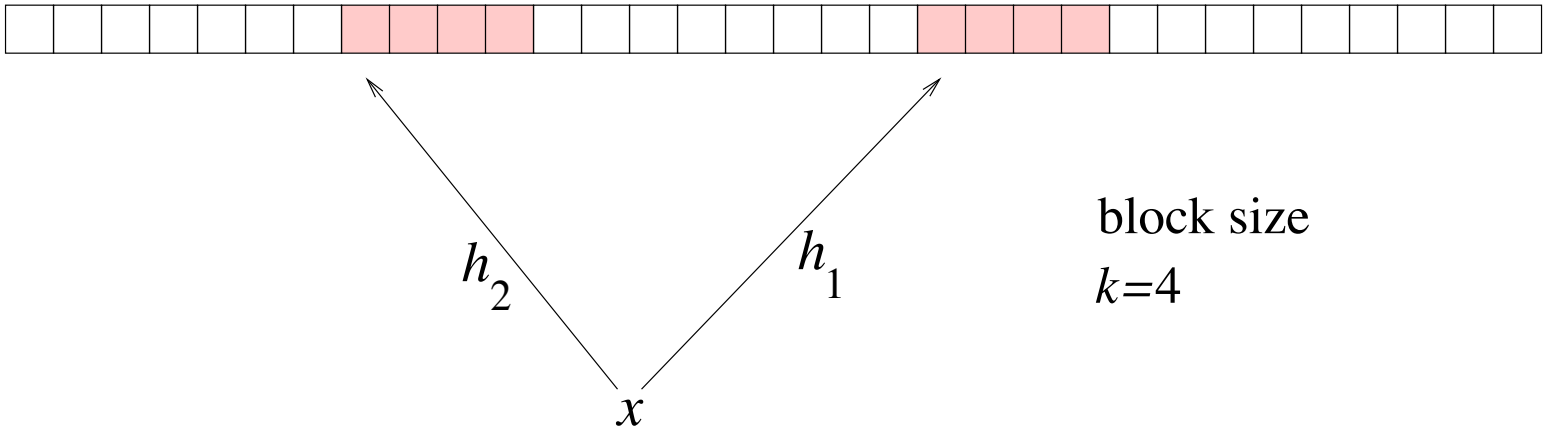


Dictionaries with accesses in T only at two intervals.

Yield **retrieval structures** with **cache-friendly access**.

Multiple-choice hashing and retrieval structures

Blocked cuckoo hashing:



Dictionaries with accesses in T only at two intervals.

Yield **retrieval structures** with **cache-friendly access**.

[D.,Weidling 2007] Read $2 \times k$ locations, space $(1 + e^{-ck})nr$.

[Czumaj *et al.* 2003] Read $2 \times O(\log n)$ locations, space nr.

Summary

- Retrieval in time $O(k)$, space $(1 + e^{-k})nr$.
- Construction time
  $O(n^{1+\delta})$ (fair) or $O(n)$ (contrived, asymptotic)
- Close connection retrieval $\leftrightarrow$ approximate membership
- Close connection retrieval $\leftrightarrow$ hash table à la cuckoo
- Cache-friendly retrieval/approximate membership structures with almost optimal space
  (range size $\geq n^{\delta}$ or error bound $\leq 1/n^{\delta}$.)

Thank you!