# Using Tabulation to Implement

## The Power of Hashing

# Using Tabulation to Implement The Power of Hashing

Talk surveys results from

- M. Pătraşcu and M. Thorup: The power of simple tabulation hashing. STOC'11 and J. ACM'12.
- M. Pătraşcu and M. Thorup: Twisted Tabulation Hashing. SODA'13
- M. Thorup: Simple Tabulation, Fast Expanders, Double Tabulation, and High Independence. FOCS'13.
- S. Dahlgaard and M. Thorup: Approximately Minwise Independence with Twisted Tabulation. SWAT'14.
- T. Christiani, R. Pagh, and M. Thorup: From Independence to Expansion and Back Again. STOC'15.
- S. Dahlgaard, M.B.T. Knudsen and E. Rotenberg, and M. Thorup: Hashing for Statistics over K-Partitions. FOCS'15.

## Target

- Simple and reliable pseudo-random hashing.

## Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing, yet easy to implement.

## Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing, yet easy to implement.
- Bridging theory (assuming truly random hashing) with practice (needing something implementable).

## Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing, yet easy to implement.
- Bridging theory (assuming truly random hashing) with practice (needing something implementable).
- Many randomized algorithms are very simple and popular in practice, but often implemented with too simple hash functions, so guarantees only for sufficiently random input.

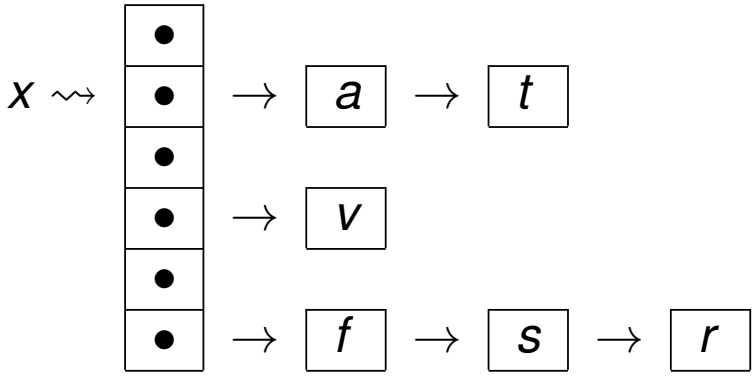
## Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing, yet easy to implement.
- Bridging theory (assuming truly random hashing) with practice (needing something implementable).
- Many randomized algorithms are very simple and popular in practice, but often implemented with too simple hash functions, so guarantees only for sufficiently random input.
- Too simple hash functions may work deceivingly well in random tests, but the real world is full of structured data on which they may fail miserably (as we shall see later).

# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

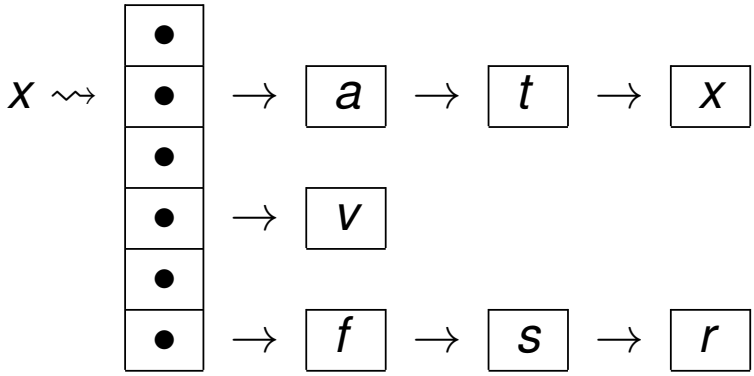
- chaining: follow pointers



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

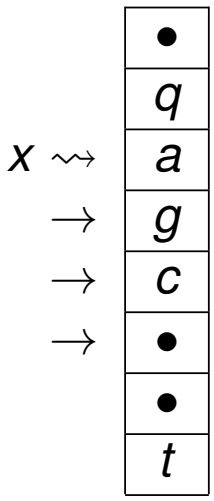
- chaining: follow pointers



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers
- linear probing: sequential search in *one* array



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

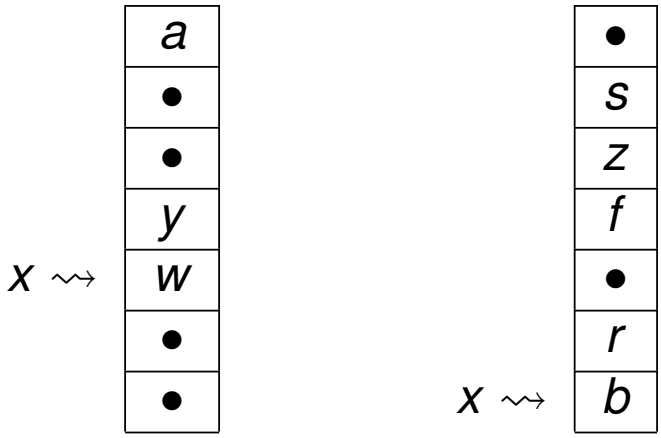
- chaining: follow pointers
- linear probing: sequential search in *one* array

| | |
|---|---|
| | $\bullet$ |
| | $q$ |
| $x \rightsquigarrow$ | $a$ |
| $\rightarrow$ | $g$ |
| $\rightarrow$ | $c$ |
| $\rightarrow$ | $x$ |
| | $\bullet$ |
| | $t$ |

# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

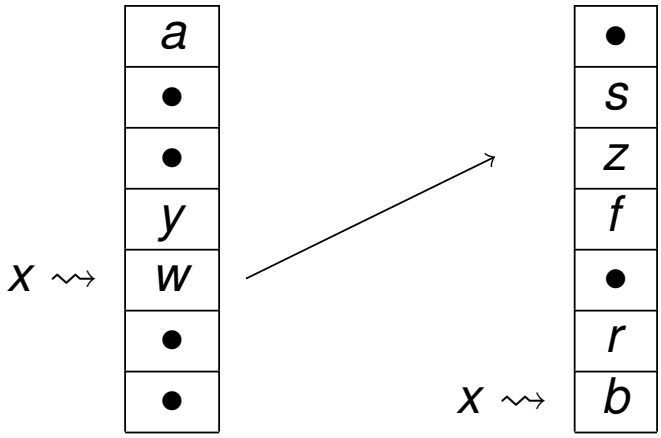
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

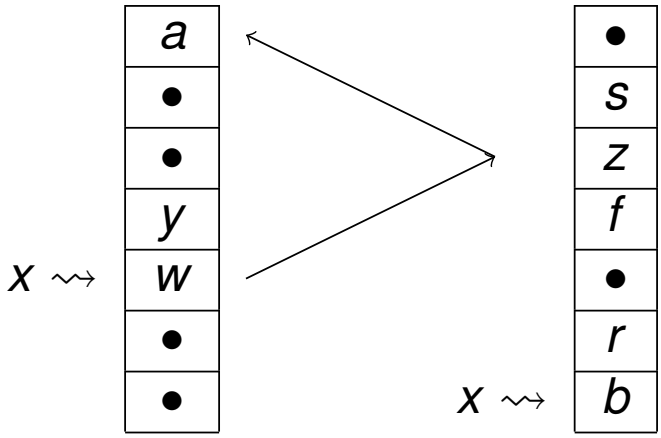
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

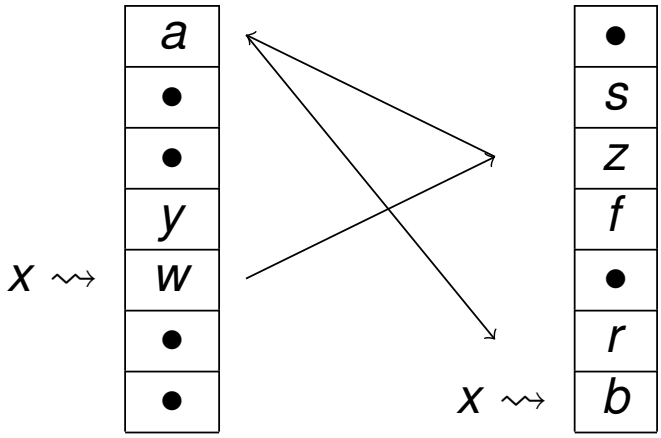
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

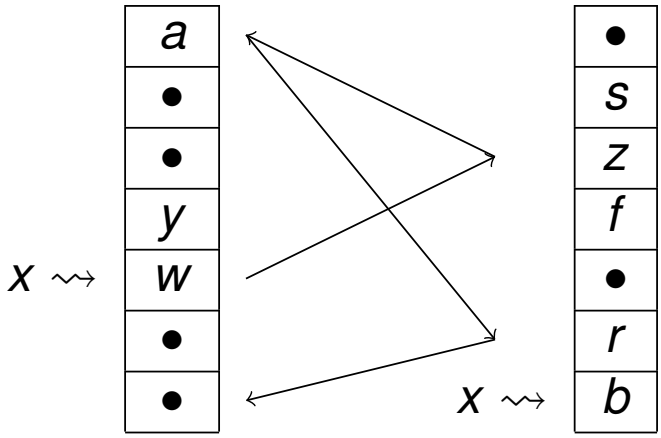
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

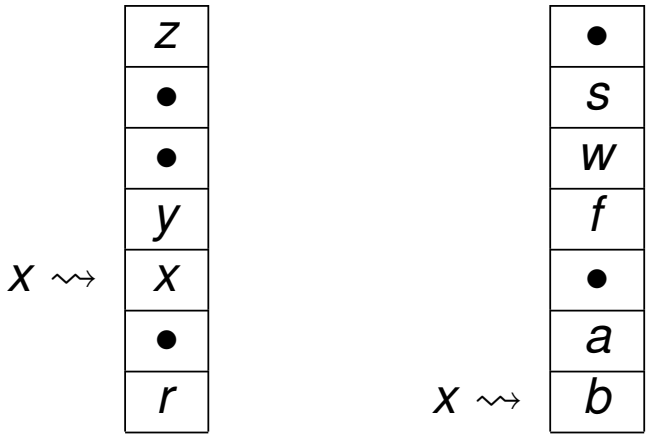
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates



# Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates



## Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers.
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates

## Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers.
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates

Sketching, streaming, and sampling:

- second moment estimation: $F_2(\bar{x}) = \sum_i x_i^2$

## Applications of Hashing $\rightsquigarrow$

Hash tables ($n$ keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers.
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq 2$ locations, complex updates

Sketching, streaming, and sampling:

- second moment estimation: $F_2(\bar{x}) = \sum_i x_i^2$
- sketch $A$ and $B$ to later find $|A \cap B|/|A \cup B|$

$$|A \cap B|/|A \cup B| = \Pr_h[\min h(A) = \min h(B)]$$

We need $h$ to be $\varepsilon$-minwise independent:

$$x \in S: \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

## Wegman & Carter [FOCS'77]

We do not have space for truly random hash functions, but

Family $\mathcal{H} = \{h : [u] \to [b]\}$ $k$-independent iff for random $h \in \mathcal{H}$:

- $(\forall) x \in [u]$, $h(x)$ is uniform in $[b]$;
- $(\forall) x_1, \ldots, x_k \in [u]$, $h(x_1), \ldots, h(x_k)$ are independent.

## Wegman & Carter [FOCS'77]

We do not have space for truly random hash functions, but

Family $\mathcal{H} = \{h : [u] \to [b]\}$ $k$-independent iff for random $h \in \mathcal{H}$:

- $(\forall) x \in [u]$, $h(x)$ is uniform in $[b]$;
- $(\forall) x_1, \ldots, x_k \in [u]$, $h(x_1), \ldots, h(x_k)$ are independent.

Prototypical example: degree $k-1$ polynomial

- $u = b$ prime;
- choose $a_0, a_1, \ldots, a_{k-1}$ randomly in $[u]$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod u$.

## Wegman & Carter [FOCS'77]

We do not have space for truly random hash functions, but

Family $\mathcal{H} = \{h : [u] \to [b]\}$ $k$-independent iff for random $h \in \mathcal{H}$:

- $(\forall) x \in [u]$, $h(x)$ is uniform in $[b]$;
- $(\forall) x_1, \ldots, x_k \in [u]$, $h(x_1), \ldots, h(x_k)$ are independent.

Prototypical example: degree $k-1$ polynomial

- $u = b$ prime;
- choose $a_0, a_1, \ldots, a_{k-1}$ randomly in $[u]$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod u$.

Many solutions for $k$-independent hashing proposed, but generally slow for $k > 3$ and too slow for $k > 5$.

# How much independence needed?

| | | |
|---|---|---|
| Chaining $\mathbf{E}[t] = O(1)$<br>$\mathbf{E}[t^k] = O(1)$<br>$\max t = O(\lg n / \lg\lg n)$ | $2$<br>$2k+1$<br>$\Theta(\lg n / \lg\lg n)$ | |
| Linear probing | $\leq 5$ [Pagh$^2$, Ružić'07] | $\geq 5$ **[PT ICALP'10]** |
| Cuckoo hashing | $O(\lg n)$ | $\geq 6$ [Cohen, Kane'05] |
| $F_2$ estimation | $4$ [Alon, Mathias, Szegedy'99] | |
| $\varepsilon$-minwise indep. | $O\left(\lg \frac{1}{\varepsilon}\right)$ [Indyk'99] | $\Omega\left(\lg \frac{1}{\varepsilon}\right)$ **[PT ICALP'10]** |

# How much independence needed?

| | | |
|---|---|---|
| Chaining $\mathbf{E}[t] = O(1)$<br>$\mathbf{E}[t^k] = O(1)$<br>$\max t = O(\lg n / \lg\lg n)$ | 2<br>$2k+1$<br>$\Theta(\lg n / \lg\lg n)$ | |
| Linear probing | $\leq 5$ [Pagh$^2$, Ružić'07] | $\geq 5$ **[PT ICALP'10]** |
| Cuckoo hashing | $O(\lg n)$ | $\geq 6$ [Cohen, Kane'05] |
| $F_2$ estimation | 4 [Alon, Mathias, Szegedy'99] | |
| $\varepsilon$-minwise indep. | $O\left(\lg \frac{1}{\varepsilon}\right)$ [Indyk'99] | $\Omega\left(\lg \frac{1}{\varepsilon}\right)$ **[PT ICALP'10]** |

Independence has been the ruling measure for quality of hash functions for 30+ years, but is it right?

## Simple Tabulation Hashing [Zobrist'70 chess]

- Key $x$ divided into $c = O(1)$ characters $x_1, ..., x_c$, e.g., 32-bit key as $4 \times 8$-bit characters.

## Simple Tabulation Hashing [Zobrist'70 chess]

- Key $x$ divided into $c = O(1)$ characters $x_1, ..., x_c$, e.g., 32-bit key as $4 \times 8$-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i$ : char $\rightarrow$ hash values (bit strings)

## Simple Tabulation Hashing [Zobrist'70 chess]

- Key $x$ divided into $c = O(1)$ characters $x_1, ..., x_c$, e.g., 32-bit key as $4 \times 8$-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i$ : char $\rightarrow$ hash values (bit strings)
- Hash value

$$h(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

## Simple Tabulation Hashing [Zobrist'70 chess]

- Key $x$ divided into $c = O(1)$ characters $x_1, ..., x_c$, e.g., 32-bit key as $4 \times 8$-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i$ : char $\rightarrow$ hash values (bit strings)
- Hash value

$$h(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

- Space $cN^{1/c}$ and time $O(c)$. With 8-bit characters, each $R_i$ has 256 entries and fit in L1 cache.

## Simple Tabulation Hashing [Zobrist'70 chess]

- Key $x$ divided into $c = O(1)$ characters $x_1, ..., x_c$, e.g., 32-bit key as $4 \times 8$-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i$ : char $\rightarrow$ hash values (bit strings)
- Hash value

$$h(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

- Space $cN^{1/c}$ and time $O(c)$. With 8-bit characters, each $R_i$ has 256 entries and fit in L1 cache.
- Simple tabulation is the fastest 3-independent hashing scheme. Speed like 2 multiplications.

## Simple Tabulation Hashing [Zobrist'70 chess]

- Key $x$ divided into $c = O(1)$ characters $x_1, ..., x_c$, e.g., 32-bit key as $4 \times 8$-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i$ : char $\rightarrow$ hash values (bit strings)
- Hash value

$$h(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

- Space $cN^{1/c}$ and time $O(c)$. With 8-bit characters, each $R_i$ has 256 entries and fit in L1 cache.
- Simple tabulation is the fastest 3-independent hashing scheme. Speed like 2 multiplications.
- Not 4-independent: $h(a_1 a_2) \oplus h(a_1 b_2) \oplus h(b_1 a_2) \oplus h(b_1 b_2)$

$$\begin{aligned} = \quad & (R_1[a_1] \oplus R_2[a_2]) \oplus (R_1[a_1] \oplus R_2[b_2]) \oplus \\ & (R_1[b_1] \oplus R_2[a_2]) \oplus (R_1[b_1] \oplus R_2[b_2]) \ = \ 0. \end{aligned}$$

# How much independence needed? Wrong question

| | | |
|---|---|---|
| Chaining $\mathbf{E}[t] = O(1)$<br>$\mathbf{E}[t^k] = O(1)$<br>$\max t = O(\lg n / \lg\lg n)$ | $2$<br>$2k+1$<br>$\Theta(\lg n / \lg\lg n)$ | |
| Linear probing | $\leq 5$ [Pagh$^2$, Ružić'07] | $\geq 5$ **[PT ICALP'10]** |
| Cuckoo hashing | $O(\lg n)$ | $\geq 6$ [Cohen, Kane'05] |
| $F_2$ estimation | $4$ [Alon, Mathias, Szegedy'99] | |
| $\varepsilon$-minwise indep. | $O(\lg \frac{1}{\varepsilon})$ [Indyk'99] | $\Omega(\lg \frac{1}{\varepsilon})$ **[PT ICALP'10]** |

# How much independence needed? Wrong question

| | | |
|---|---|---|
| Chaining $\mathbf{E}[t] = O(1)$<br>$\mathbf{E}[t^k] = O(1)$<br>$\max t = O(\lg n / \lg\lg n)$ | 2<br>$2k+1$<br>$\Theta(\lg n / \lg\lg n)$ | |
| Linear probing | $\leq 5$ [Pagh$^2$, Ružić'07] | $\geq 5$ **[PT ICALP'10]** |
| Cuckoo hashing | $O(\lg n)$ | $\geq 6$ [Cohen, Kane'05] |
| $F_2$ estimation | 4 [Alon, Mathias, Szegedy'99] | |
| $\varepsilon$-minwise indep. | $O(\lg \frac{1}{\varepsilon})$ [Indyk'99] | $\Omega(\lg \frac{1}{\varepsilon})$ **[PT ICALP'10]** |

PT'11: Despite its 4-dependence, simple tabulation suffices for all the above applications:

> *One simple and fast hashing scheme for almost all your needs.*

# How much independence needed? Wrong question

| | | |
|---|---|---|
| Chaining $\mathbf{E}[t] = O(1)$<br>$\mathbf{E}[t^k] = O(1)$<br>$\max t = O(\lg n / \lg\lg n)$ | 2<br>$2k+1$<br>$\Theta(\lg n / \lg\lg n)$ | |
| Linear probing | $\leq 5$ [Pagh$^2$, Ružić'07] | $\geq 5$ **[PT ICALP'10]** |
| Cuckoo hashing | $O(\lg n)$ | $\geq 6$ [Cohen, Kane'05] |
| $F_2$ estimation | 4 [Alon, Mathias, Szegedy'99] | |
| $\varepsilon$-minwise indep. | $O(\lg \frac{1}{\varepsilon})$ [Indyk'99] | $\Omega(\lg \frac{1}{\varepsilon})$ **[PT ICALP'10]** |

PT'11: Despite its 4-dependence, simple tabulation suffices for all the above applications:

> *One simple and fast hashing scheme for almost all your needs.*

Knuth recommends simple tabulation but cites only 3-independence as mathematical quality.

# How much independence needed? Wrong question

| | | |
|---|---|---|
| Chaining $\mathbf{E}[t] = O(1)$<br>$\mathbf{E}[t^k] = O(1)$<br>$\max t = O(\lg n / \lg\lg n)$ | 2<br>$2k+1$<br>$\Theta(\lg n / \lg\lg n)$ | |
| Linear probing | $\leq 5$ [Pagh$^2$, Ružić'07] | $\geq 5$ **[PT ICALP'10]** |
| Cuckoo hashing | $O(\lg n)$ | $\geq 6$ [Cohen, Kane'05] |
| $F_2$ estimation | 4 [Alon, Mathias, Szegedy'99] | |
| $\varepsilon$-minwise indep. | $O(\lg \frac{1}{\varepsilon})$ [Indyk'99] | $\Omega(\lg \frac{1}{\varepsilon})$ **[PT ICALP'10]** |

PT'11: Despite its 4-dependence, simple tabulation suffices for all the above applications:

> *One simple and fast hashing scheme for almost all your needs.*

Knuth recommends simple tabulation but cites only 3-independence as mathematical quality.
Need to prove, for worst-case input, that the dependence of simple tabulation is not harmful in any of the above applications.

## Chaining/hashing into bins

Theorem Consider hashing $n$ balls into $m \geq n^{1-1/(2c)}$ bins by simple tabulation. Let $q$ be an additional *query ball*, and define $X_q$ as the number of regular balls hashing to the same bin as $q$. Let $\mu = \mathbf{E}[X_q] = \frac{n}{m}$. The following probability bounds hold for any constant $\gamma$:

$$\Pr[X_q \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\Omega(\mu)} + m^{-\gamma}$$

$$\Pr[X_q \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\Omega(\mu)} + m^{-\gamma}$$

## Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

## Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Nothing like this lemma holds if we instead of simple tabulation assumed $k$-independent hashing with $k = O(1)$.

## Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently.

- Let $i$ be character position where keys in $T$ differ.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently.

- Let $i$ be character position where keys in $T$ differ.
- Let $a$ be least common character in position $i$ and pick $x \in T$ with $x_i = a$

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently.

- Let $i$ be character position where keys in $T$ differ.
- Let $a$ be least common character in position $i$ and pick $x \in T$ with $x_i = a$
- Reduce $T$ to $T'$ removing all keys $y$ from $T$ with $y_i = a$.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently.

- Let $i$ be character position where keys in $T$ differ.
- Let $a$ be least common character in position $i$ and pick $x \in T$ with $x_i = a$
- Reduce $T$ to $T'$ removing all keys $y$ from $T$ with $y_i = a$.
- The hash of $x$ is independent of the hash of $T'$ as only $h(x)$ depends on $R_i[a]$.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently.

- Let $i$ be character position where keys in $T$ differ.
- Let $a$ be least common character in position $i$ and pick $x \in T$ with $x_i = a$
- Reduce $T$ to $T'$ removing all keys $y$ from $T$ with $y_i = a$.
- The hash of $x$ is independent of the hash of $T'$ as only $h(x)$ depends on $R_i[a]$.
- Return $\{x\} \cup U'$ where $U'$ independent subset of $T'$.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

- There are $\binom{n}{u} < n^u$ sets $U$ of $u$ keys.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

- There are $\binom{n}{u} < n^u$ sets $U$ of $u$ keys.
- If keys in $U$ hash independently,
  they land in *same* bin with probability $1/m^{u-1}$.

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

- There are $\binom{n}{u} < n^u$ sets $U$ of $u$ keys.
- If keys in $U$ hash independently,
  they land in *same* bin with probability $1/m^{u-1}$.
- Probability bound over all independently hashed $U$ is

$$n^u/m^{u-1} \leq m^{(1-\varepsilon)u+1-u} = m^{1-\varepsilon u} = m^{-\gamma}.$$

# Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.$\square$

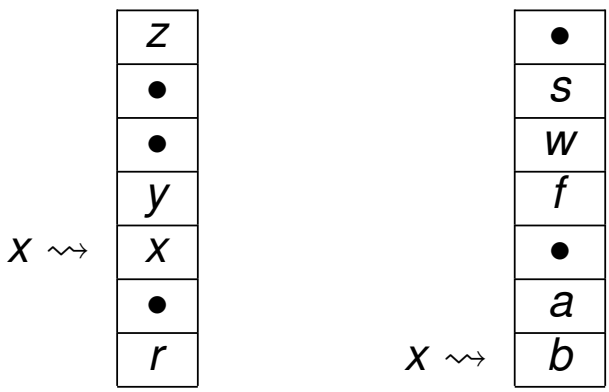
## Hashing into many bins

Lemma If we hash $n$ keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants $\varepsilon, \gamma$, if we hash $n$ keys into $m$ bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.
Claim 1 Any set $T$ contains a subset $U$ of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$
Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.$\square$

Fundamental point Simple tabulation is only 3-independent, but inside any set $T$ of size $\omega(1)$, there is a subset $S \subseteq T$ of size $\omega(1)$ where all keys in $S$ are hashed independently.

# Cuckoo hashing

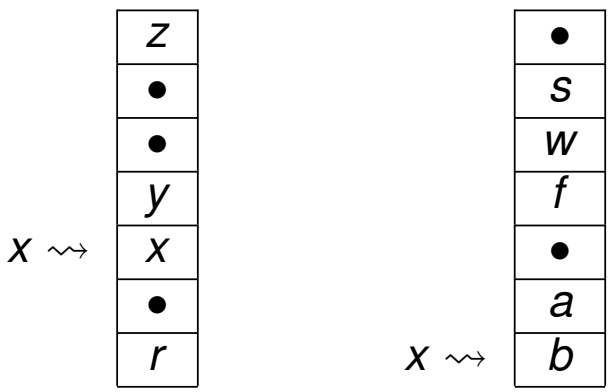
Each key placed in one of two hash locations.




Theorem With simple tabulation Cuckoo hashing works with probability $1 - \tilde{\Theta}(n^{-1/3})$.

# Cuckoo hashing

Each key placed in one of two hash locations.

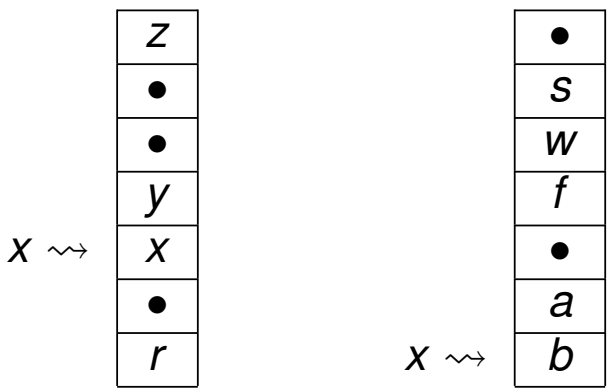



Theorem With simple tabulation Cuckoo hashing works with probability $1 - \tilde{\Theta}(n^{-1/3})$.

- For chaining and linear probing, we did not care about a constant loss, but obstructions to cuckoo hashing may be of just constant size, e.g., 3 keys sharing same two hash locations.

# Cuckoo hashing

Each key placed in one of two hash locations.




Theorem With simple tabulation Cuckoo hashing works with probability $1 - \tilde{\Theta}(n^{-1/3})$.

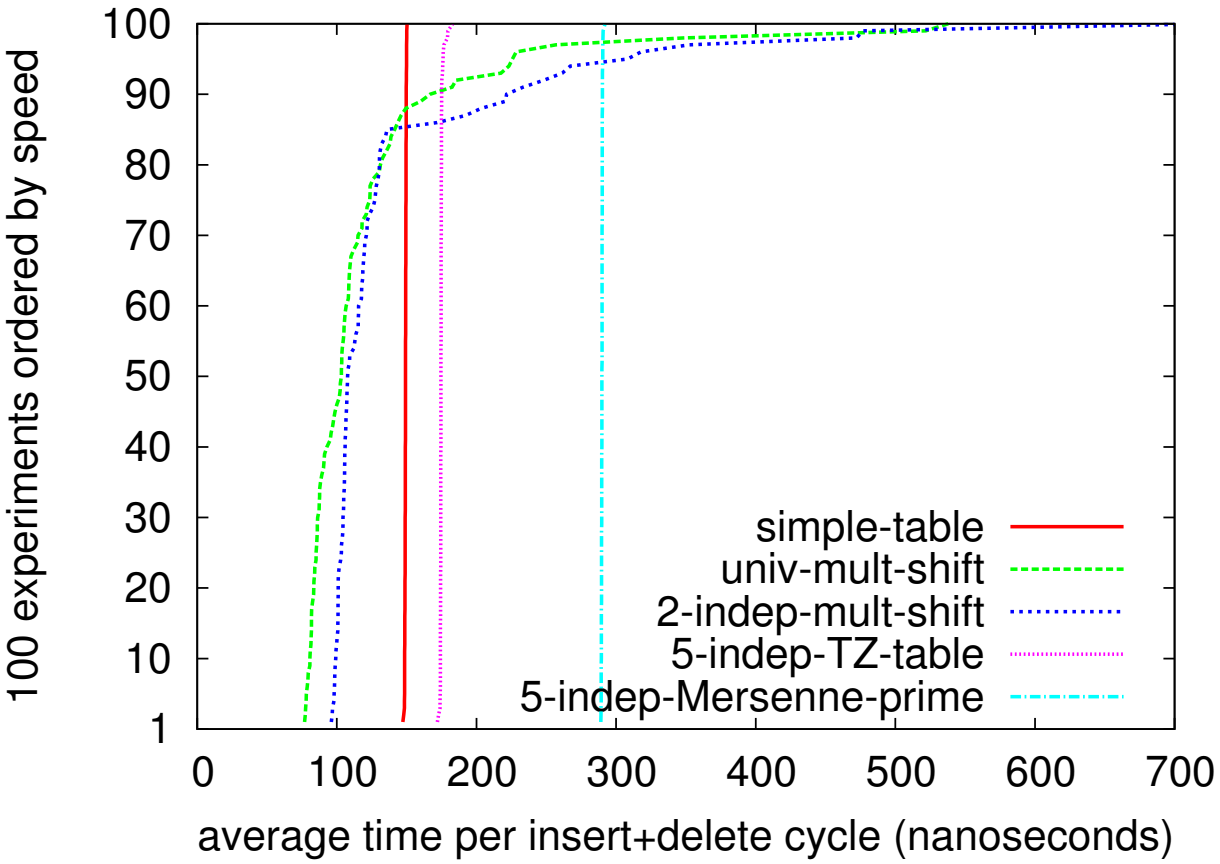
- For chaining and linear probing, we did not care about a constant loss, but obstructions to cuckoo hashing may be of just constant size, e.g., 3 keys sharing same two hash locations.
- Very delicate proof showing that obstruction can be used to code random tables $R_i$ with few bits.

# Speed

| Hashing random keys | | 32-bit computer | 64-bit computer |
|---|---|---|---|
| bits | hashing scheme | hashing time (ns) | |
| 32 | univ-mult-shift `(a*x)>>s` | 1.87 | 2.33 |
| 32 | 2-indep-mult-shift | 5.78 | 2.88 |
| 32 | 5-indep-Mersenne-prime | 99.70 | 45.06 |
| 32 | 5-indep-TZ-table | 10.12 | 12.66 |
| 32 | simple-table | 4.98 | 4.61 |
| 64 | univ-mult-shift | 7.05 | 3.14 |
| 64 | 2-indep-mult-shift | 22.91 | 5.90 |
| 64 | 5-indep-Mersenne-prime | 241.99 | 68.67 |
| 64 | 5-indep-TZ-table | 75.81 | 59.84 |
| 64 | simple-table | 15.54 | 11.40 |

Experiments with help from Yin Zhang.

# Robustness in linear probing for dense interval



## Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].

## Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.

## Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.
- Linear probing had gotten a reputation for being fastest in practice, but sometimes unreliable needing special protection against bad cases.

## Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.
- Linear probing had gotten a reputation for being fastest in practice, but sometimes unreliable needing special protection against bad cases.
- Here we proved linear probing safe with good probabilistic performance for all input if we use simple tabulation.

## Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.
- Linear probing had gotten a reputation for being fastest in practice, but sometimes unreliable needing special protection against bad cases.
- Here we proved linear probing safe with good probabilistic performance for all input if we use simple tabulation.
- Simple tabulation also powerful for chaining, cuckoo hashing, and min-wise hashing:

  *one simple and fast scheme for (almost) all your needs.*

## Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.

## Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.

## Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!

## Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!
- Simple tabulation does not achieve this: may often spend $\tilde{\Omega}(\log^2 n)$ time on $\log n$ consecutive operations.

## Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!
- Simple tabulation does not achieve this: may often spend $\tilde{\Omega}(\log^2 n)$ time on $\log n$ consecutive operations.
- Twisted tabulation:

$$\begin{aligned}(h, \alpha) &= R_1[x_1] \oplus \cdots \oplus R_{c-1}[x_{c-1}]; \\ h(x) &= h \oplus R_c[\alpha \oplus x_c]\end{aligned}$$

## Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!
- Simple tabulation does not achieve this: may often spend $\tilde{\Omega}(\log^2 n)$ time on $\log n$ consecutive operations.
- Twisted tabulation:

$$\begin{aligned} (h, \alpha) &= R_1[x_1] \oplus \cdots \oplus R_{c-1}[x_{c-1}]; \\ h(x) &= h \oplus R_c[\alpha \oplus x_c] \end{aligned}$$

- With twisted tabulation, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.

## C-code for twisted tabulation (32-bit key, 8-bit char)

```
INT32 TwistedTab32(INT32 x, INT64[4][256] H) {
  INT32 i; INT64 h=0; INT8 c;
  for (i=0;i<c-1;i++) {
    c=x;
    h^=H[i][c];
    x = x>> 8;
  }
  c=x^h;        //twisted character
  h^=H[i][c];
  h>>=8;       //dropping twister from hash
  return ((INT32) h);
}
```

## General Chernoff bounds with twisted tabulation

- 0-1 variables $X_i$, $X = \sum_i X_i$, $\mu = \mathbf{E}[X]$.
- E.g., with hashing into $[0, 1]$, set $X_i = 1$ if $h(i) < p_i$.
- With bounded independence only polynomial concentration.
- With twisted tabulation: for any constant $\gamma$,

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\Omega(\mu)} + |U|^{-\gamma}$$

$$\Pr[X \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\Omega(\mu)} + |U|^{-\gamma}$$

- With simple tabulation, additive term was $m^{-\gamma}$ with $m$ bins, but $m = 2$ for unbiased coins.

## Min-wise hashing

- Min-wise hashing $h$ of keys in $U$ with bias $\varepsilon$:

$$x \in S \subseteq U : \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

## Min-wise hashing

- Min-wise hashing $h$ of keys in $U$ with bias $\varepsilon$:

$$x \in S \subseteq U : \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

- With simple tabulation, we get $\varepsilon = \tilde{O}(1/|S|^{1/c})$ —good only for large sets $S$.

## Min-wise hashing

- Min-wise hashing $h$ of keys in $U$ with bias $\varepsilon$:

$$x \in S \subseteq U: \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

- With simple tabulation, we get $\varepsilon = \tilde{O}(1/|S|^{1/c})$
  —good only for large sets $S$.
- With twisted tabulation, we get $\varepsilon = \tilde{O}(1/|U|^{1/c})$.
  —good for sets $S$ of any size.

# Speed of different schemes

| Hashing scheme | time (ns) |
|---|---|
| 2-indep: multiplication-shift [Dietzfelbinger'96] | 1.27 |
| 3-indep: Polynomial with Mersenne prime $2^{61} - 1$ | 7.72 |
| 5-indep: Polynomial with Mersenne prime $2^{61} - 1$ | 14.09 |
| $k$-indep: Polynomial with Mersenne prime $2^{61} - 1$ | $\approx 3.5(k-1)$ |
| simple tabulation | 2.26 |
| twisted tabulation | 2.86 |

# Speed of different schemes

| Hashing scheme | time (ns) |
|---|---|
| 2-indep: multiplication-shift [Dietzfelbinger'96] | 1.27 |
| 3-indep: Polynomial with Mersenne prime $2^{61} - 1$ | 7.72 |
| 5-indep: Polynomial with Mersenne prime $2^{61} - 1$ | 14.09 |
| $k$-indep: Polynomial with Mersenne prime $2^{61} - 1$ | $\approx 3.5(k-1)$ |
| simple tabulation | 2.26 |
| twisted tabulation | 2.86 |
| **Pseudo-random number generator** | **time (ns)** |
| random() from C-libraray | 6.99 |
| twisted-PRG() | 1.18 |

## Double Tabulation

- Recall that simple tabulation is only 3-independent (and so is twisted).
- If you want high independence, then just apply simple tabulation twice...

## Double Tabulation

- Recall that simple tabulation is only 3-independent (and so is twisted).
- If you want high independence, then just apply simple tabulation twice...
- Still many applications only known to work with $\Theta(\log n)$-independence, and errors in min-wise hashing and Chernoff bounds fall exponentially in the independence.

## $k$-independence [Wegman Carter FOCS'79]

Hashing universe $U$ of keys into range $R$ of hash values.

Random hash function $h : U \to R$ is $k$-independent iff:

- $(\forall) x \in U$, $h(x)$ is (almost) uniform in $R$;
- $(\forall) x_1, \ldots, x_k \in U$, $h(x_1), \ldots, h(x_k)$ are independent.

Prototypical example: Polynomial over prime field $\mathbb{Z}_p$:

- choose $a_0, a_1, \ldots, a_{k-1}$ randomly in $\mathbb{Z}_p$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod p$.
- then $h : \mathbb{Z}_p \to \mathbb{Z}_p$ is $k$-independent with exact uniformity.

Ideal when it comes to space and amount of randomness:

- store $k$ random elements from $\mathbb{Z}_p$,

## $k$-independence [Wegman Carter FOCS'79]

Hashing universe $U$ of keys into range $R$ of hash values.

Random hash function $h : U \to R$ is $k$-independent iff:

- $(\forall) x \in U$, $h(x)$ is (almost) uniform in $R$;
- $(\forall) x_1, \ldots, x_k \in U$, $h(x_1), \ldots, h(x_k)$ are independent.

Prototypical example: Polynomial over prime field $\mathbb{Z}_p$:

- choose $a_0, a_1, \ldots, a_{k-1}$ randomly in $\mathbb{Z}_p$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod p$.
- then $h : \mathbb{Z}_p \to \mathbb{Z}_p$ is $k$-independent with exact uniformity.

Ideal when it comes to space and amount of randomness:

- store $k$ random elements from $\mathbb{Z}_p$,
- but evaluating $h(x)$ takes $O(k)$ time.

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

Question Can we get $k$-independent hashing in $t < k$ time?

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

**Siegel's formulation** With $t, c = O(1)$:

**Negative** $\omega(1)$-independence in $O(1)$ time requires $|U|^{\Omega(1)}$ space.

# Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

**Siegel's formulation** With $t, c = O(1)$:

**Negative** $\omega(1)$-independence in $O(1)$ time requires $|U|^{\Omega(1)}$ space.

**Positive** $|U|^{\Omega(1)}$-independence in $O(1)$ time using $|U|^{\varepsilon}$ space for any $\varepsilon = \Omega(1)$.

# Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

**Siegel's formulation** With $t, c = O(1)$:

**Negative** $\omega(1)$-independence in $O(1)$ time requires $|U|^{\Omega(1)}$ space.

**Positive** $|U|^{\Omega(1)}$-independence in $O(1)$ time using $|U|^{\varepsilon}$ space for any $\varepsilon = \Omega(1)$.

Better than the $\Theta(\log |U|)$-independence needed for many applications. Gives error bounds like $2^{-|U|^{\Omega(1)}}$ in Chernoff bounds and min-wise hashing.

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

**Siegel's formulation** With $t, c = O(1)$:

**Negative** $\omega(1)$-independence in $O(1)$ time requires $|U|^{\Omega(1)}$ space.

**Positive** $|U|^{\Omega(1)}$-independence in $O(1)$ time using $|U|^{\varepsilon}$ space for any $\varepsilon = \Omega(1)$.

Better than the $\Theta(\log |U|)$-independence needed for many applications. Gives error bounds like $2^{-|U|^{\Omega(1)}}$ in Chernoff bounds and min-wise hashing.

But "far too slow for any practical application".

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

"far too slow for any practical application".

New result with double tabulation

$|U|^{\Omega(1/c^2)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space.

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

"far too slow for any practical application".

New result with double tabulation

$|U|^{\Omega(1/c^2)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space.

Scheme is simple but analysis is not.

## Siegel on highly independence [FOCS'89]

Polynomials: $k$-independent hashing in $k$ time using $k$ space

**Question** Can we get $k$-independent hashing in $t < k$ time?

**Negative** $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

**Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

"far too slow for any practical application".

New result with double tabulation

$|U|^{\Omega(1/c^2)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space.

Scheme is simple but analysis is not.

More convoluted with Christiani and Pagh STOC'15

$|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space.

## Simple tabulation

- Key $x = (x_0, ..., x_{c-1}) \in \Phi^c = U$, $c = O(1)$.
- For $i = 0, , ..., c-1$, truly random character hash table:
  $h_i : \Phi \to R = b$-bit strings.
- Hash function $h : U \to R$ defined by

$$h(x_0, ..., x_{c-1}) = h_0[x_0] \oplus \cdots \oplus h_{c-1}[x_{c-1}]$$

- We saw not 4-independent, yet powerful enough for many algorithmic contexts otherwise requiring higher independence.

## Simple tabulation

- Key $x = (x_0, ..., x_{c-1}) \in \Phi^c = U$, $c = O(1)$.
- For $i = 0, , ..., c-1$, truly random character hash table:
  $h_i : \Phi \to R = b$-bit strings.
- Hash function $h : U \to R$ defined by

$$h(x_0, ..., x_{c-1}) = h_0[x_0] \oplus \cdots \oplus h_{c-1}[x_{c-1}]$$

- We saw not 4-independent, yet powerful enough for many algorithmic contexts otherwise requiring higher independence.

We now claim that simple tabulation is expected to yield:

- Very fast unbalanced constant degree expanders.

## Simple tabulation

- Key $x = (x_0, ..., x_{c-1}) \in \Phi^c = U$, $c = O(1)$.
- For $i = 0, , ..., c-1$, truly random character hash table:
  $h_i : \Phi \to R = b$-bit strings.
- Hash function $h : U \to R$ defined by

$$h(x_0, ..., x_{c-1}) = h_0[x_0] \oplus \cdots \oplus h_{c-1}[x_{c-1}]$$

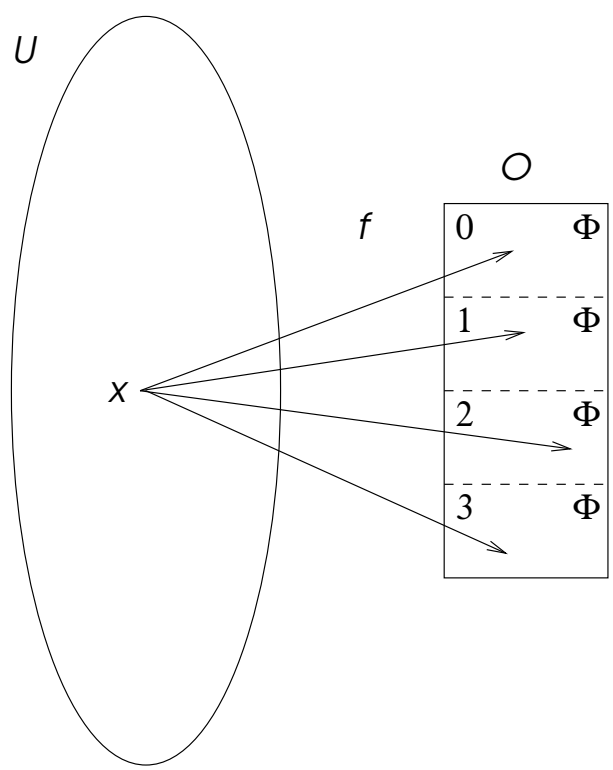
- We saw not 4-independent, yet powerful enough for many algorithmic contexts otherwise requiring higher independence.

We now claim that simple tabulation is expected to yield:

- Very fast unbalanced constant degree expanders.
- Applied twice yields highly independent hashing.

# Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys $U$ and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.



## Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys $U$ and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.

## Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys $U$ and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.

Thm Let $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 12c$. Then with probability $1 - o(1/|\Phi|)$,

$$\forall S \subseteq U, |S| \leq |\Phi|^{1/(5c)} : |N_h(S)| > d|S|/2.$$

## Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys $U$ and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.

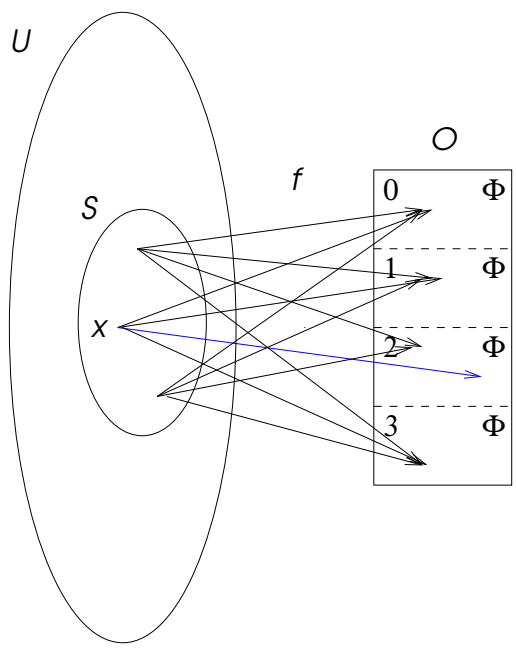
Thm Let $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 12c$. Then with probability $1 - o(1/|\Phi|)$,

$$\forall S \subseteq U, |S| \leq |\Phi|^{1/(5c)} : |N_h(S)| > d|S|/2.$$

Note Best explicit unbalanced expanders [Guruswami et al. JACM'09] have logarithmic degrees, and would be orders of magnitude slower to compute.

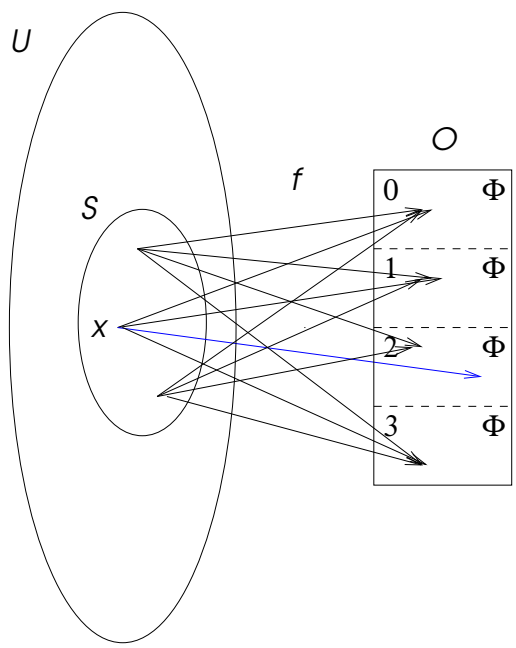
# Unique output position character

Set $S \subseteq U$ has unique output position character if there is output position character neighbor to exactly one key in $S$.



# Unique output position character

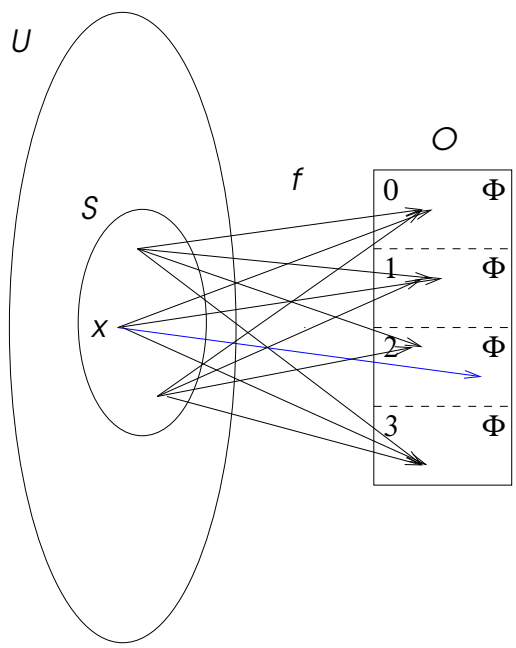
Set $S \subseteq U$ has unique output position character if there is output position character neighbor to exactly one key in $S$.




Obs $S$ has a unique neighbor if $|N(S)| > d|S|/2$.

# Unique output position character

Set $S \subseteq U$ has unique output position character if there is output position character neighbor to exactly one key in $S$.




Obs $S$ has a unique neighbor if $|N(S)| > d|S|/2$.

Thm Let $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then with probability $1 - o(1/|\Phi|)$, $\forall S \subseteq U, |S| \leq |\Phi|^{1/(5c)}$ : $S$ has a unique output position character.

## Unique output position character

A function $f : U \to \Phi^d$ is $k$-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character .}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then $h$ is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

## Unique output position character

A function $f : U \to \Phi^d$ is $k$-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character .}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then $h$ is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is $k$-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is $k$-independent.

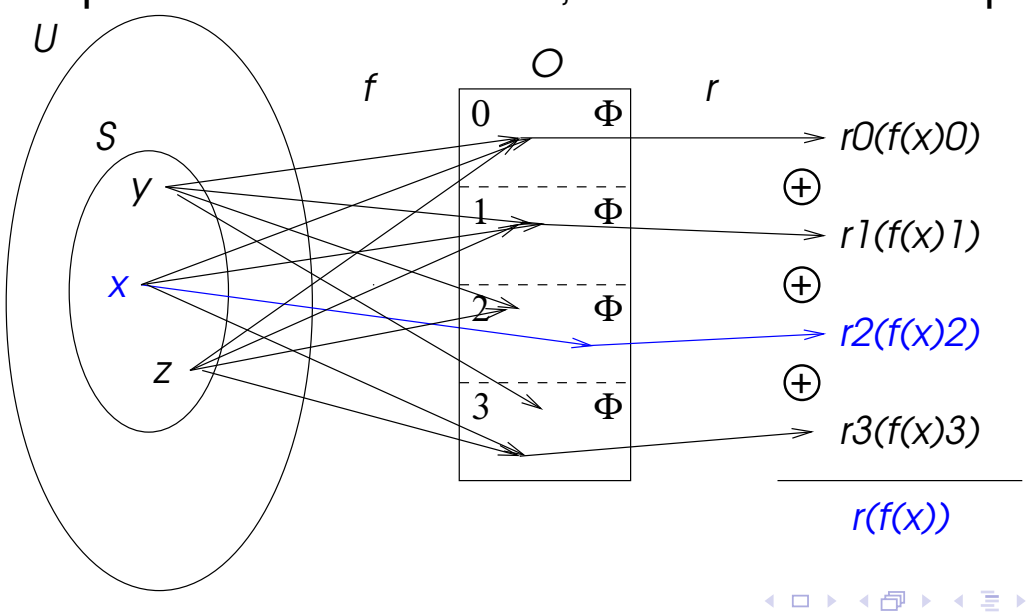
# Unique output position character

A function $f : U \to \Phi^d$ is $k$-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character .}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then $h$ is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is $k$-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is $k$-independent.



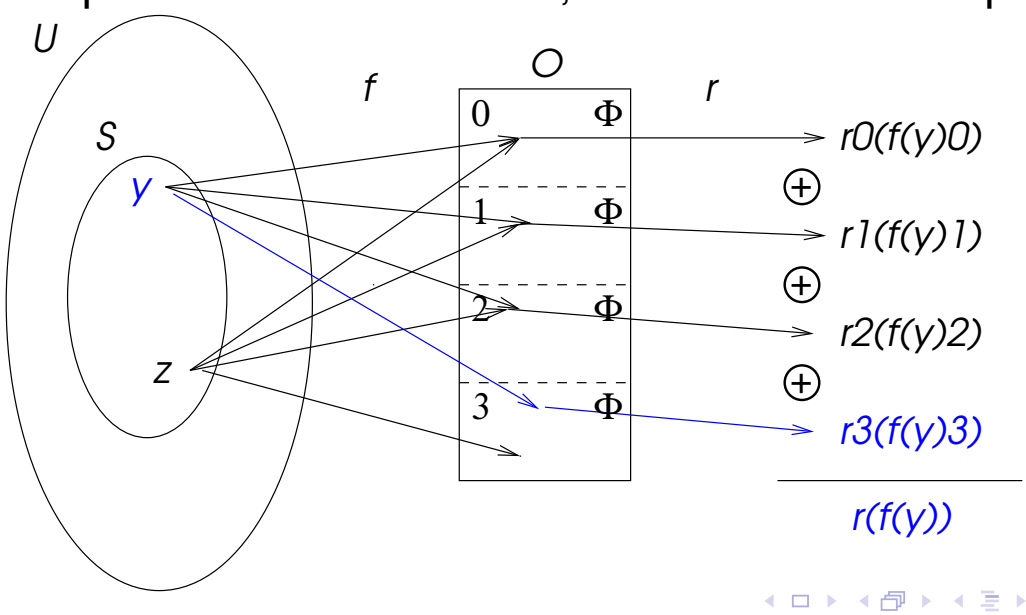
# Unique output position character

A function $f : U \to \Phi^d$ is $k$-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character.}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then $h$ is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is $k$-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is $k$-independent.



## Unique output position character

A function $f : U \to \Phi^d$ is $k$-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character.}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then $h$ is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is $k$-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is $k$-independent.

Cor (Double Tabulation) If $d = 6c$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions, then $r \circ h : U \to R$ is $|\Phi|^{1/(5c)}$-independent with "universal" probability $1 - o(1/|\Phi|)$.

## Unique output position character

A function $f : U \to \Phi^d$ is $k$-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character.}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then $h$ is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is $k$-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is $k$-independent.

Cor (Double Tabulation) If $d = 6c$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions, then $r \circ h : U \to R$ is $|\Phi|^{1/(5c)}$-independent with "universal" probability $1 - o(1/|\Phi|)$.

Note A $k$-unique $h$ can be used as a universal constant that only has to be guessed or constructed once.

## With real constants

Constants in construction pretty good:

Thm For 32-bit keys divided in $c = 2$ characters from $\Phi = [2^{16}]$, if $h : \Phi^2 \to \Phi^{20}$ is random simple tabulation function, then $h$ is 100-unique with probability $1 - 1.5 \times 10^{-42}$.

## With real constants

Constants in construction pretty good:

Thm For 32-bit keys divided in $c = 2$ characters from $\Phi = [2^{16}]$, if $h : \Phi^2 \to \Phi^{20}$ is random simple tabulation function, then $h$ is 100-unique with probability $1 - 1.5 \times 10^{-42}$.

- Claiming that a USB flash drive with random bits represents a universal $k$-unique simple tabulation function, would be very safe hardware.

## With real constants

Constants in construction pretty good:

Thm For 32-bit keys divided in $c = 2$ characters from $\Phi = [2^{16}]$, if $h : \Phi^2 \to \Phi^{20}$ is random simple tabulation function, then $h$ is 100-unique with probability $1 - 1.5 \times 10^{-42}$.

- Claiming that a USB flash drive with random bits represents a universal $k$-unique simple tabulation function, would be very safe hardware.
- 100-indepedent hashing is thus possible and easy to implement.

## With real constants

Constants in construction pretty good:

Thm For 32-bit keys divided in $c = 2$ characters from $\Phi = [2^{16}]$, if $h : \Phi^2 \to \Phi^{20}$ is random simple tabulation function, then $h$ is 100-unique with probability $1 - 1.5 \times 10^{-42}$.

- Claiming that a USB flash drive with random bits represents a universal $k$-unique simple tabulation function, would be very safe hardware.
- 100-indepedent hashing is thus possible and easy to implement.
- Yet simple and twisted tabulation is still going to be at 10-100 times faster.

## Open $k$-independence problem

Current state:

Negative $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).
2. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (recursive tabulation).

## Open $k$-independence problem

Current state:

Negative $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).
2. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (recursive tabulation).

Desired $|U|^{\Omega(1/c)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space?

## Open $k$-independence problem

Current state:

Negative $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).
2. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (recursive tabulation).

Desired $|U|^{\Omega(1/c)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space?

It may be that double tabulation achieves this, but proving it would require different analysis.

## Fully-Random Hashing

It turns out that double tabulation, w.h.p., yields fully-random hashing for given set $S$ using $O(|S|)$ space
(with Dahlgaard, Knudsen, Rotenberg FOCS'15).

Thm If $d = 4$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions and $S \subseteq U$ has size at most $|\Phi|/2$, then $r \circ h : U \to R$ is fully random on $S$ w.h.p.

# Fully-Random Hashing

It turns out that double tabulation, w.h.p., yields fully-random hashing for given set $S$ using $O(|S|)$ space
(with Dahlgaard, Knudsen, Rotenberg FOCS'15).

Thm If $d = 4$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions and $S \subseteq U$ has size at most $|\Phi|/2$, then $r \circ h : U \to R$ is fully random on $S$ w.h.p.

Compare with previous high independence result:

Thm If $d = 6c$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions, then $r \circ h : U \to R$ is $|\Phi|^{1/(5c)}$-independent w.h.p.

Previous fully-random hashing [Pagh and Pagh STOC'03] used high independence as subroutine.

# Fully-Random Hashing

It turns out that double tabulation, w.h.p., yields fully-random hashing for given set $S$ using $O(|S|)$ space
(with Dahlgaard, Knudsen, Rotenberg FOCS'15).

Thm If $d = 4$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions and $S \subseteq U$ has size at most $|\Phi|/2$, then $r \circ h : U \to R$ is fully random on $S$ w.h.p.

Compare with previous high independence result:

Thm If $d = 6c$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions, then $r \circ h : U \to R$ is $|\Phi|^{1/(5c)}$-independent w.h.p.

Previous fully-random hashing [Pagh and Pagh STOC'03] used high independence as subroutine. New technical point:

Thm If $h : U = \Phi^c \to \Phi^d$ is a random simple tabulation function and if $S \subseteq U$ has size $\leq |\Phi|/2$, with probability $1 - O(|\Phi|^{1-\lfloor d/2 \rfloor})$, every $T \subseteq S$ has a unique output character.

.. and then Siegel's peeling argument applies to all of $S$.

## Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.

## Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.

## Invertible Bloom Filters [Goldreich and Mitzenmacher]

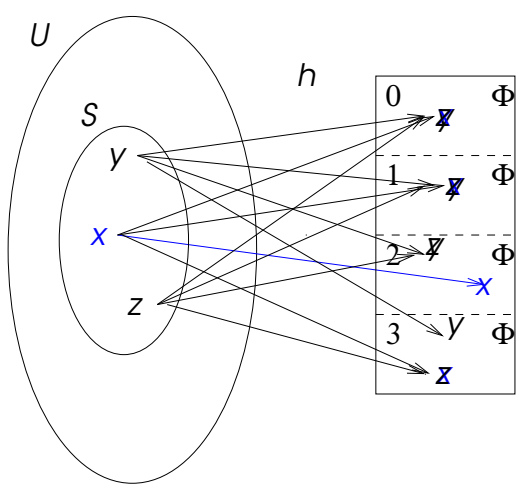
- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.
- Application: Given $IB(S)$ and $IB(T)$, if $|S - T| \leq n$, we can recover $S - T$ from $IB(S) - IB(T)$:
  finding exact difference between similar sets.

## Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.
- Application: Given $IB(S)$ and $IB(T)$, if $|S - T| \leq n$, we can recover $S - T$ from $IB(S) - IB(T)$: finding exact difference between similar sets.
- Let keys $x \in U \subseteq W = \{0, 1\}^b$ include signature, so we can check, w.h.p, if $x \in W$ is in $U$.

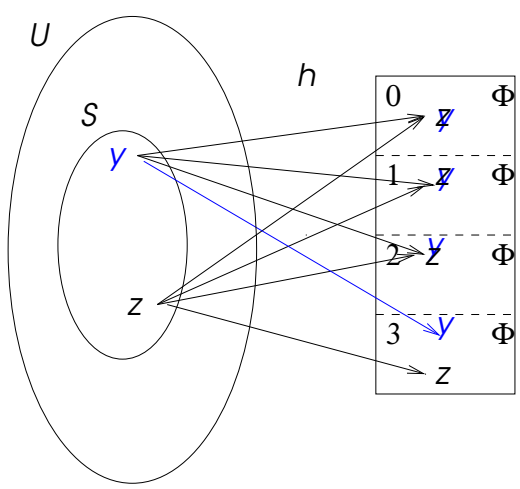
# Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.
- Application: Given $IB(S)$ and $IB(T)$, if $|S - T| \leq n$, we can recover $S - T$ from $IB(S) - IB(T)$: finding exact difference between similar sets.
- Let keys $x \in U \subseteq W = \{0,1\}^b$ include signature, so we can check, w.h.p, if $x \in W$ is in $U$.
- Use random simple tabulation $h : W = \Phi^c \to \Phi^4$, $|\Phi| \geq 2n$.



# Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.
- Application: Given $IB(S)$ and $IB(T)$, if $|S - T| \leq n$, we can recover $S - T$ from $IB(S) - IB(T)$: finding exact difference between similar sets.
- Let keys $x \in U \subseteq W = \{0,1\}^b$ include signature, so we can check, w.h.p, if $x \in W$ is in $U$.
- Use random simple tabulation $h : W = \Phi^c \to \Phi^4$, $|\Phi| \geq 2n$.



# Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.
- Application: Given $IB(S)$ and $IB(T)$, if $|S - T| \leq n$, we can recover $S - T$ from $IB(S) - IB(T)$: finding exact difference between similar sets.
- Let keys $x \in U \subseteq W = \{0,1\}^b$ include signature, so we can check, w.h.p, if $x \in W$ is in $U$.
- Use random simple tabulation $h : W = \Phi^c \to \Phi^4$, $|\Phi| \geq 2n$. Define $IB(S) \in (\mathbb{Z} \times \mathbb{Z})^{[4] \times \Phi}$ such that

$$IB(S)[i,a] = \left( \sum_{x \in S, h(x)[i]=a} S(x), \sum_{x \in S, h(x)[i]=a} x \cdot S(x) \right).$$

# Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.
- Application: Given $IB(S)$ and $IB(T)$, if $|S - T| \leq n$, we can recover $S - T$ from $IB(S) - IB(T)$:
  finding exact difference between similar sets.
- Let keys $x \in U \subseteq W = \{0,1\}^b$ include signature, so we can check, w.h.p, if $x \in W$ is in $U$.
- Use random simple tabulation $h : W = \Phi^c \to \Phi^4$, $|\Phi| \geq 2n$. Define $IB(S) \in (\mathbb{Z} \times \mathbb{Z})^{[4]\times\Phi}$ such that

$$IB(S)[i,a] = \left( \sum_{x \in S, h(x)[i]=a} S(x), \sum_{x \in S, h(x)[i]=a} x \cdot S(x) \right).$$

- If $IB(S)[i,a] = (s,y)$ and $y/s = x \in U$, then $x \in S$, with unique $h(x)[i] = a$ and $S(x) = s$.

# Invertible Bloom Filters [Goldreich and Mitzenmacher]

- Weighted set $S \subseteq U$ is vector from $\mathbb{Z}^U$ support $|S|$.
- Want linear sketch $IB(S)$ of fixed size $O(n)$ so that if $|S| \leq n$, we can recover $S$ from $IB(S)$ w.h.p.
- Application: Given $IB(S)$ and $IB(T)$, if $|S - T| \leq n$, we can recover $S - T$ from $IB(S) - IB(T)$:
  finding exact difference between similar sets.
- Let keys $x \in U \subseteq W = \{0,1\}^b$ include signature, so we can check, w.h.p, if $x \in W$ is in $U$.
- Use random simple tabulation $h : W = \Phi^c \to \Phi^4$, $|\Phi| \geq 2n$. Define $IB(S) \in (\mathbb{Z} \times \mathbb{Z})^{[4]\times\Phi}$ such that

$$IB(S)[i,a] = \left( \sum_{x \in S, h(x)[i]=a} S(x), \sum_{x \in S, h(x)[i]=a} x \cdot S(x) \right).$$

- If $IB(S)[i,a] = (s,y)$ and $y/s = x \in U$, then $x \in S$, with unique $h(x)[i] = a$ and $S(x) = s$.
- Then peel $x$ from $IB(S)$. Recover $S \setminus \{x\}$ from $IB(S \setminus \{x\})$.

## Concluding remarks

- We have shown that tabulation hashing offers much of the power of idealistic fully-random hashing.

## Concluding remarks

- We have shown that tabulation hashing offers much of the power of idealistic fully-random hashing.
- Contrasting classic polynomial hashing, we can have lots of randomness in the tables, but only little is used in each hash computation.

## Concluding remarks

- We have shown that tabulation hashing offers much of the power of idealistic fully-random hashing.
- Contrasting classic polynomial hashing, we can have lots of randomness in the tables, but only little is used in each hash computation.
- For the tables, we only need a pointer to some randomly configured memory without any structure. This could be cheap and fast read-only memory.

## Concluding remarks

- We have shown that tabulation hashing offers much of the power of idealistic fully-random hashing.
- Contrasting classic polynomial hashing, we can have lots of randomness in the tables, but only little is used in each hash computation.
- For the tables, we only need a pointer to some randomly configured memory without any structure. This could be cheap and fast read-only memory.
- In multi-core systems the read-only leaves the cache clean with efficient concurrent processing.

## Concluding remarks

- We have shown that tabulation hashing offers much of the power of idealistic fully-random hashing.
- Contrasting classic polynomial hashing, we can have lots of randomness in the tables, but only little is used in each hash computation.
- For the tables, we only need a pointer to some randomly configured memory without any structure. This could be cheap and fast read-only memory.
- In multi-core systems the read-only leaves the cache clean with efficient concurrent processing.
- With limited randomness, we could fill out tables with a $\Theta(\log n)$-indendent pseudo-random number generator, the point being to gain speed using these pre-computed tables.

## Open problems

- Recall that simple tabulation is much faster than any other 3-independent hashing scheme.

## Open problems

- Recall that simple tabulation is much faster than any other 3-independent hashing scheme.
- The generic open problem is to take *any* problem currently using highly independent hashing, and see if, say, simple or twisted tabulation solves it.

## Open problems

- Recall that simple tabulation is much faster than any other 3-independent hashing scheme.
- The generic open problem is to take *any* problem currently using highly independent hashing, and see if, say, simple or twisted tabulation solves it.
- E.g., recently, with Dahlgaard, Knudsen, Rotenberg SODA'16, we showed that simple tabulation gives "power of choice" in load balancing.

## Open problems

- Recall that simple tabulation is much faster than any other 3-independent hashing scheme.
- The generic open problem is to take *any* problem currently using highly independent hashing, and see if, say, simple or twisted tabulation solves it.
- E.g., recently, with Dahlgaard, Knudsen, Rotenberg SODA'16, we showed that simple tabulation gives "power of choice" in load balancing.
- A fundamental open problem is to get $|U|^{\Omega(1/c)}$-independence in $O(c)$ time using $|U|^{1/c}$ space, matching Siegel's lower bound.

## Open problems

- Recall that simple tabulation is much faster than any other 3-independent hashing scheme.
- The generic open problem is to take *any* problem currently using highly independent hashing, and see if, say, simple or twisted tabulation solves it.
- E.g., recently, with Dahlgaard, Knudsen, Rotenberg SODA'16, we showed that simple tabulation gives "power of choice" in load balancing.
- A fundamental open problem is to get $|U|^{\Omega(1/c)}$-independence in $O(c)$ time using $|U|^{1/c}$ space, matching Siegel's lower bound.
- A wild conjecture is that double tabulation achieves this.