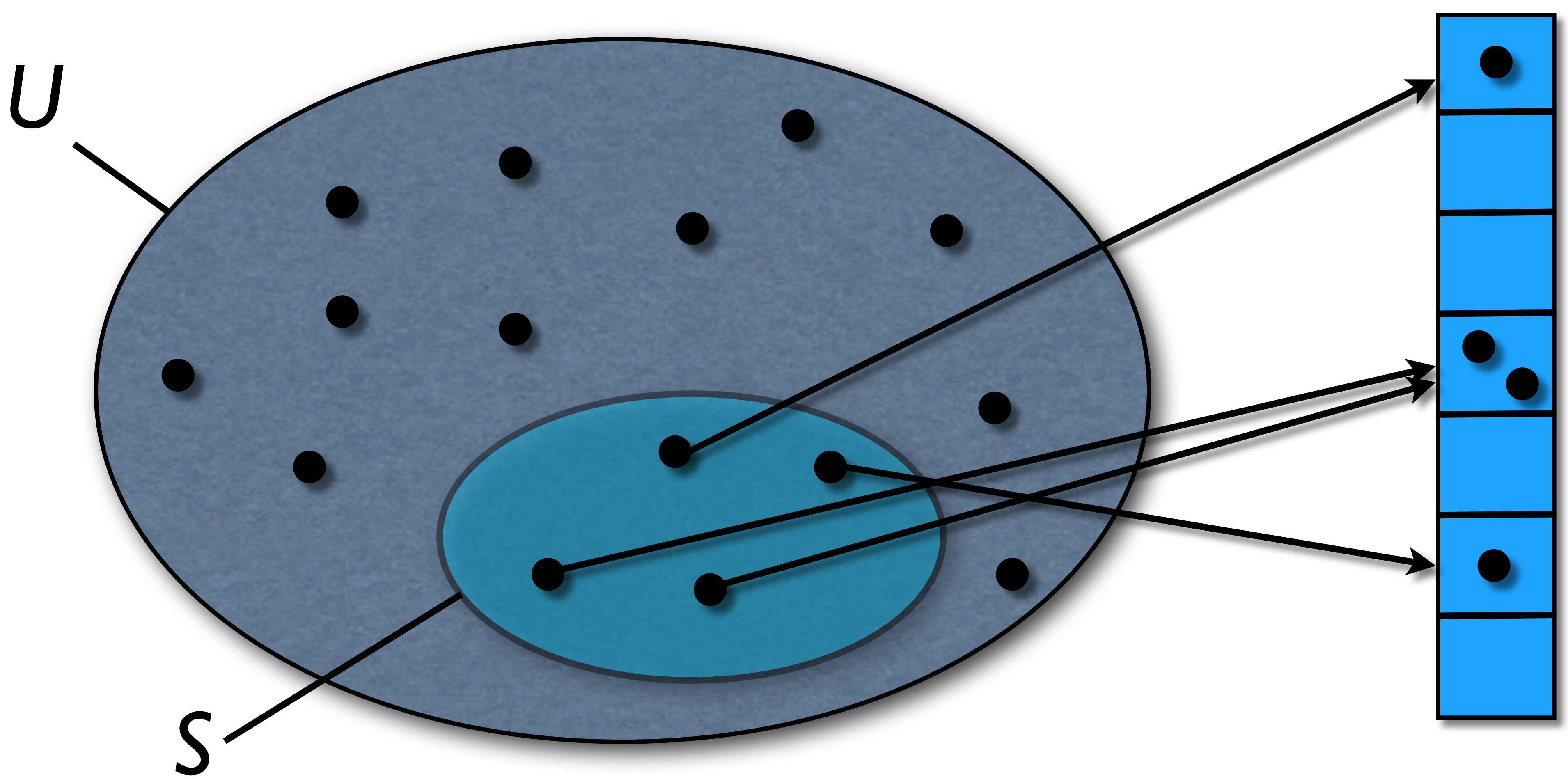
Lecture 2: Hashing

Johannes Fischer

Hashing

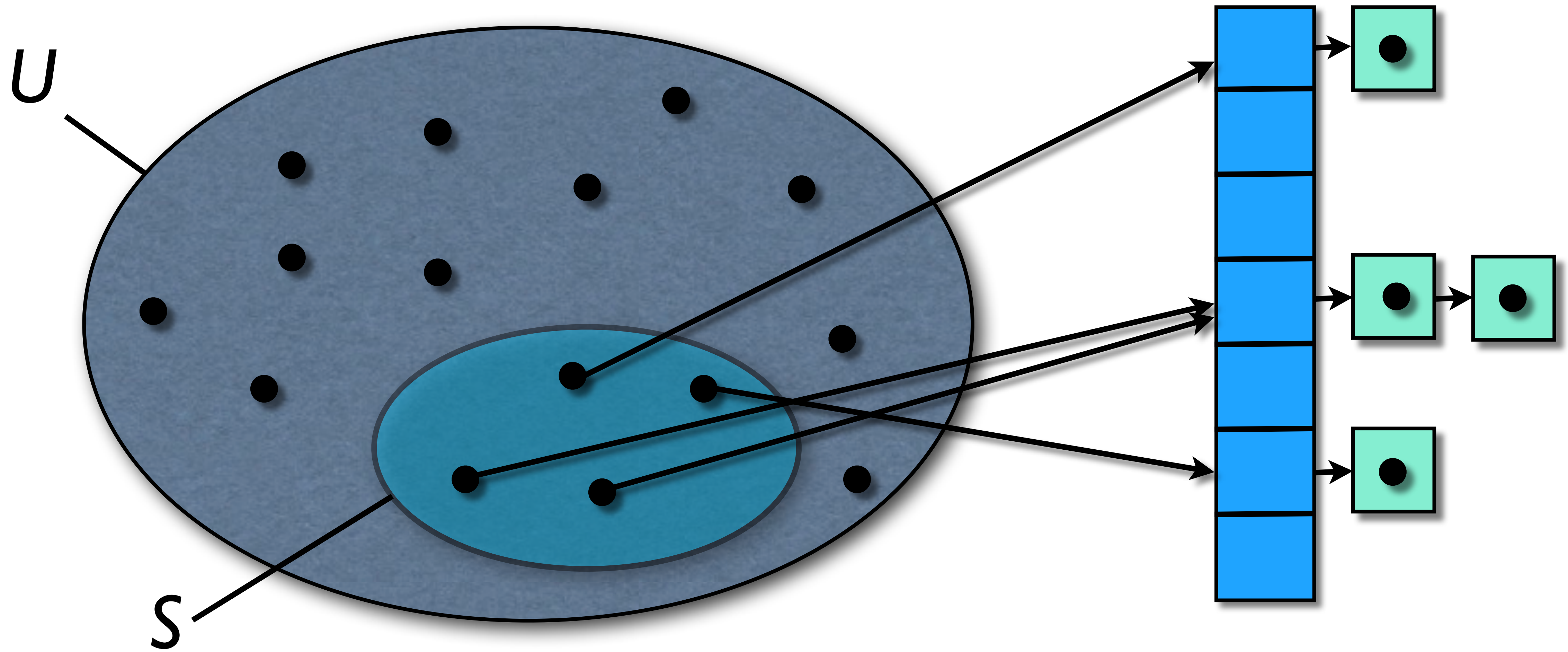
- set S of n objects from universe U=[0,u-1]
- operations: insert/delete/search



Baseline Algorithms

	array $A[0, u-1]$	**linked list**	**balanced search tree**
search	$O(1)$	$O(n)$	$O(\lg n)$
insert	$O(1)$	$O(1)$	$O(\lg n)$
delete	$O(1)$	$O(n)$	$O(\lg n)$
space	$O(u)$	$O(n)$	$O(n)$

Hashing with Chaining



hashing with chaining	
search	$O(1)$ expected
insert	$O(1)$ amortized
delete	$O(1)$ expected, amortized
space	$O(n)$

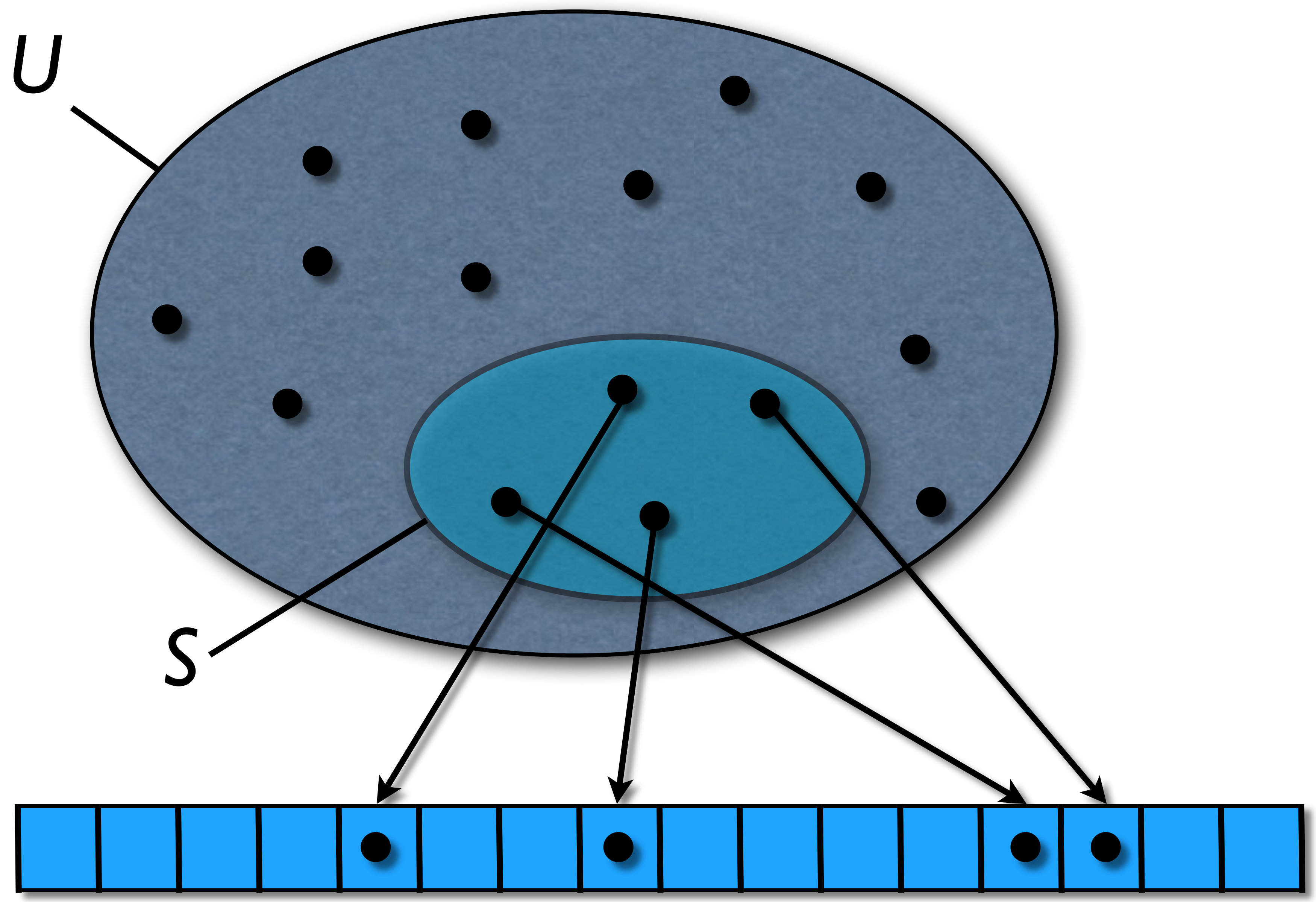
Perfect Hashing

- *S* static (insert all items at beginning)
  - can achieve $O(1)$ **worst case** search
- Fredman, Komlós, Szemerédi [J. ACM 1984]

perfect hashing	
search	$O(1)$ w.c.
construction	$O(n)$ exp.
space	$O(n)$ w.c.

Idea

- Table of size n^2 ⇒ collisions **unlikely**



Idea

- Formally:
  - storing n keys
  - table of size $m=n^2$
  - universal hash function h ("truly random")

  ➡ Prob[single collision] < $^1/_2$

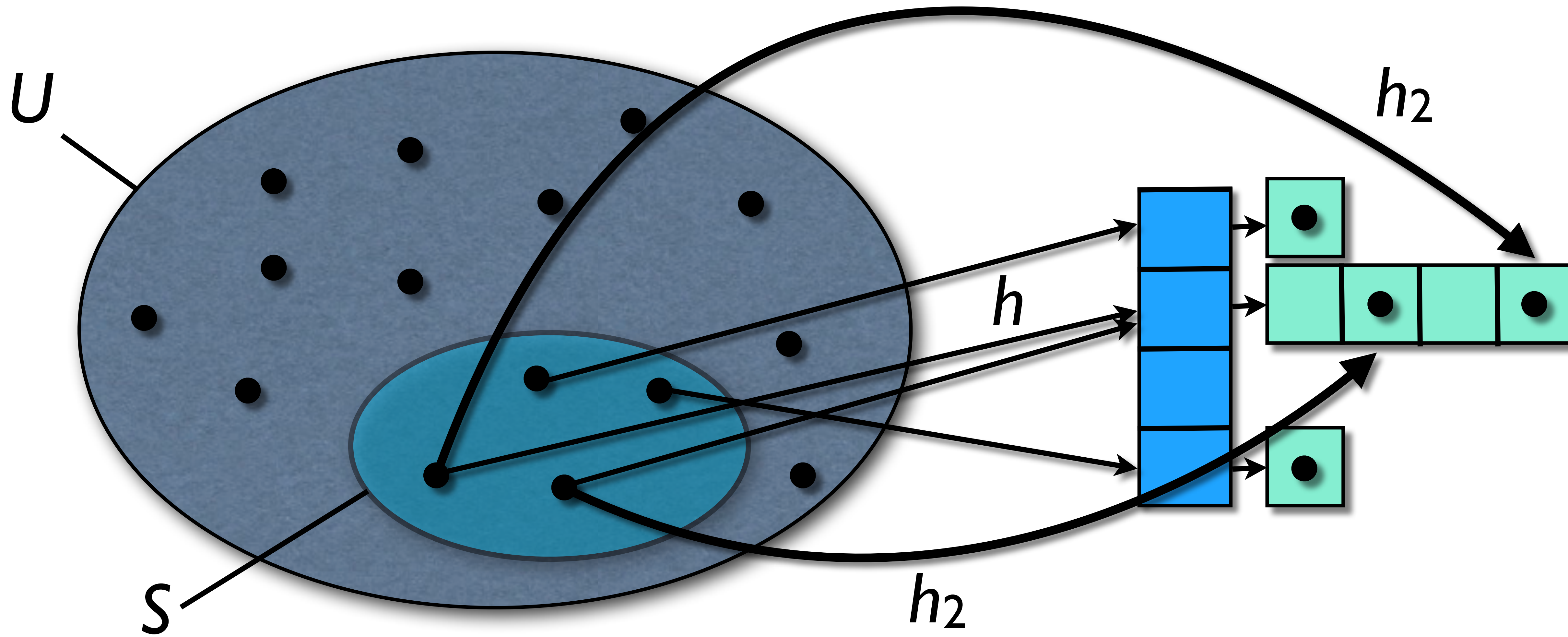
- **Proof**: Prob[$h(x)=h(y)$] = $^1/_m$ for $x \neq y$
  ⇒ Exp[#collisions] = #pairs · Prob[$h(x)=h(y)$]

  $= (n^2 - n)/2 \cdot {}^1/_m = {}^1/_2 - 1/2n < {}^1/_2$

  ⇒ Prob[#collisions≥1] ≤ Exp[#collisions] < $^1/_2$ ∎

Using the n^2-idea

- start as in hashing with chaining (m=n)
- bucket i with n_i items: n^2-idea with $m_i = n_i^2$
  - need n additional hash functions h_i



Space Usage

- Claim: $\text{Exp}[\Sigma_{1 \le i \le m} n_i^2] \le 2n$
- **Proof**: $\sum n_i^2 = \sum (2\binom{n_i}{2} + n_i)$

$$= 2\sum \binom{n_i}{2} + \sum n_i$$

$$= 2\sum \binom{n_i}{2} + n$$

$$\Rightarrow \text{Exp}[\sum n_i^2] = n + 2\text{Exp}[\sum \binom{n_i}{2}]$$

$$\text{Exp}[\sum \binom{n_i}{2}] = \binom{n}{2} \cdot \frac{1}{m} = \frac{n-1}{2}$$

$$\Rightarrow \text{Exp}[\sum n_i^2] = n + 2\frac{n-1}{2} = 2n - 1 < 2n \ \blacksquare$$

The Final Picture

- try first hash function until $\Sigma\ n_i^2 = \Theta(n)$
- with Markov's inequality:

$$\text{Prob}[\Sigma\ n_i^2 \geq 4n] \leq {}^1/_2$$

➡on **average** less than 2 trials needed

- same true for the n secondary hash functions until collision free

➡**expected** running time $O(n)$

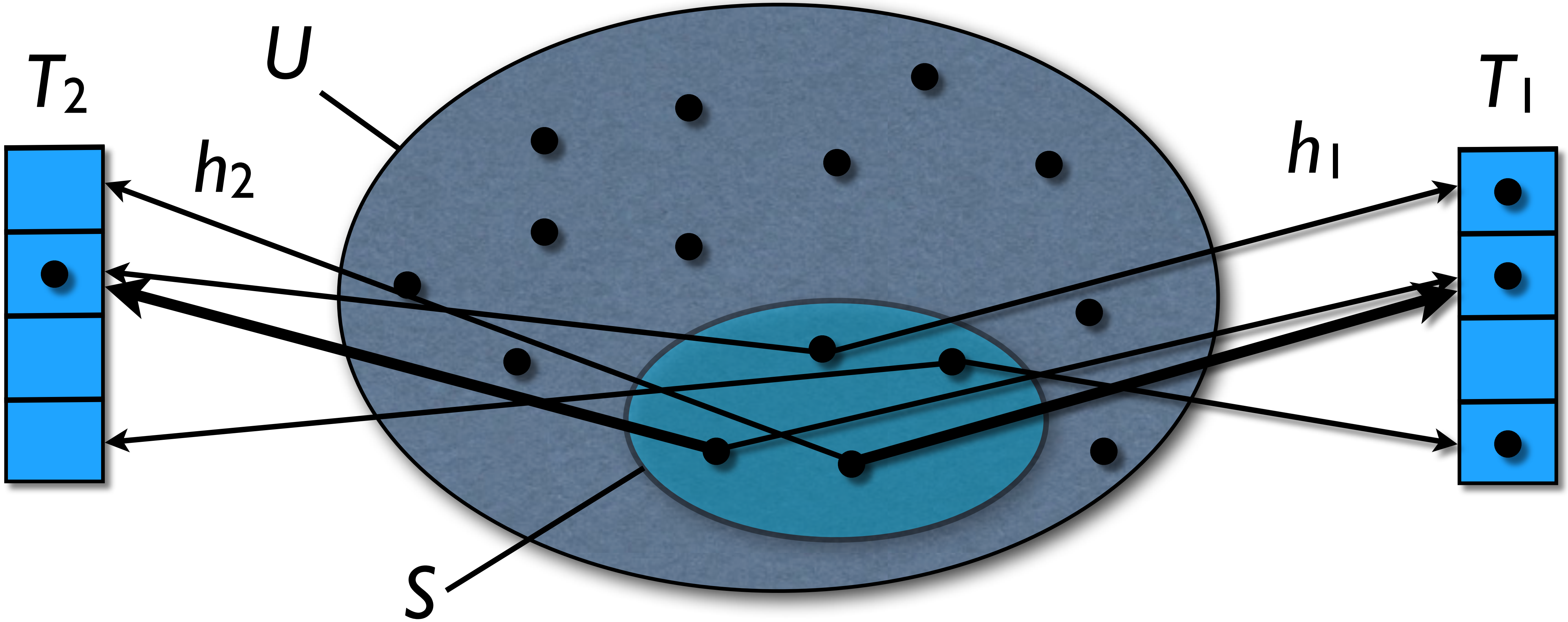
Cuckoo Hashing

- Performance of perfect hashing, but **dynamic**
- Pagh, Rodler [J. Alg. 2004]
- current state of the art!

cuckoo hashing	
search	$O(1)$ w.c.
insert	$O(1)$ exp., amort.
delete	$O(1)$ exp., amort.
space	$O(n)$ w.c.

Idea

- 2 tables T_1 and T_2 (both of size $\Theta(n)$)
- 2 hash functions h_1 and h_2
  - x either at $T_1[h_1(x)]$ or $T_2[h_2(x)]$



Simple Procedures

- table sizes $m \approx 2n$

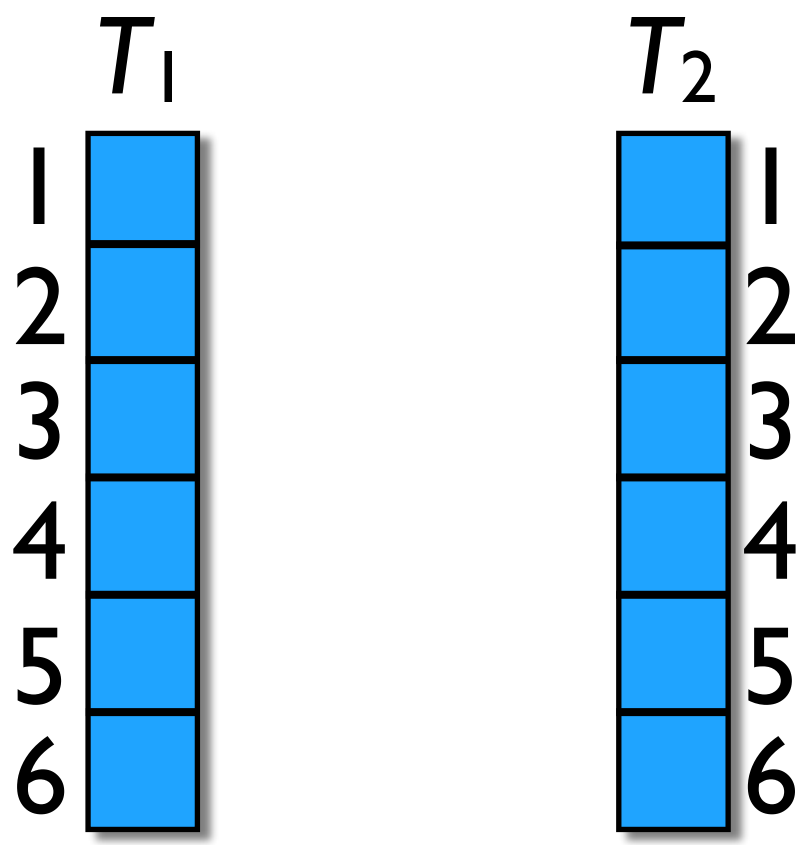
function search(x):
  if ($T_1[h_1(x)]=x$ **or** $T_2[h_2(x)]=x$) **return** true
  otherwise return false

function delete(x):
  if ($T_1[h_1(x)]=x$) $T_1[h_1(x)] \leftarrow \perp$; n--
  if ($T_2[h_2(x)]=x$) $T_2[h_2(x)] \leftarrow \perp$; n--
  if ($n < m/8$) rehash($m/2$)

Insertion

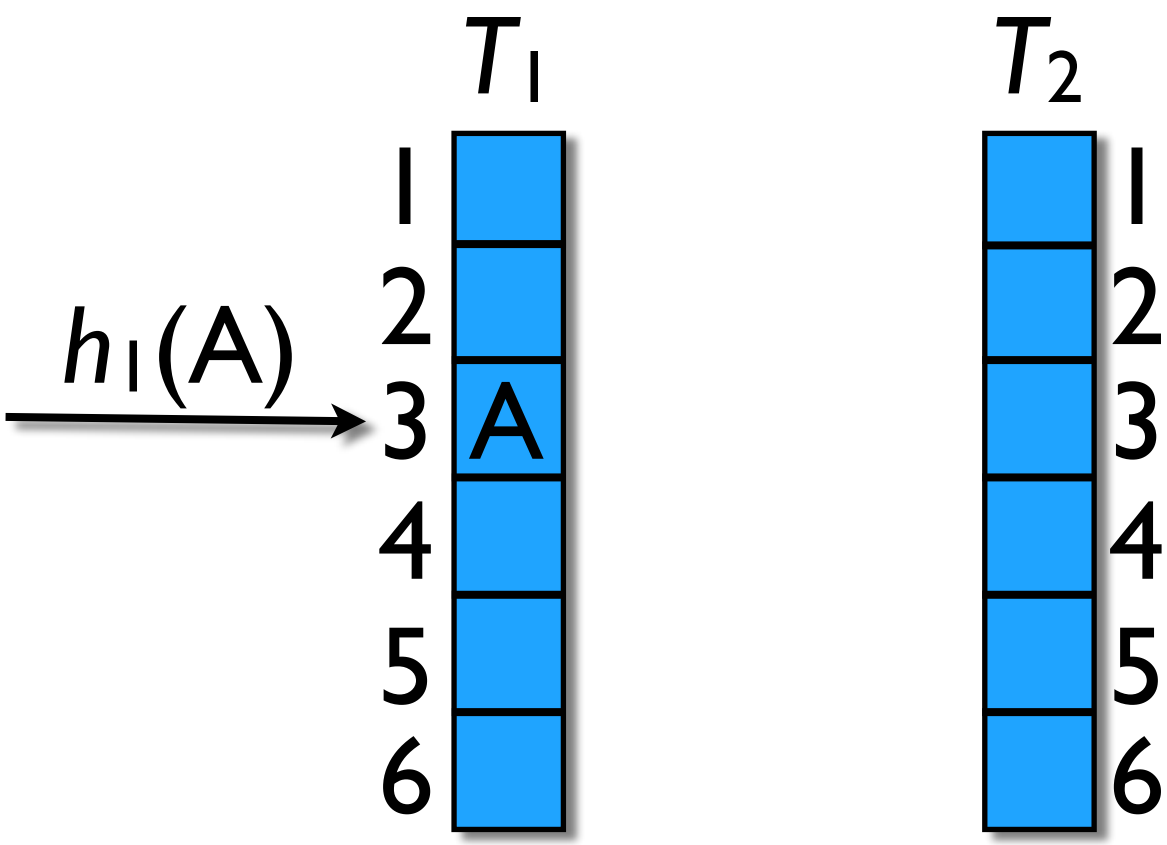
```
function insert(x):
    if (search(x)) return
    k ← 1
    repeat maxLoop times:
        swap x with T_k[h_k(x)]
        if (x = ⊥)
            n++; if (n > m/2) rehash(2m)
            return
        k ← 3 - k
    rehash(m); insert(x)
```

Example



x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

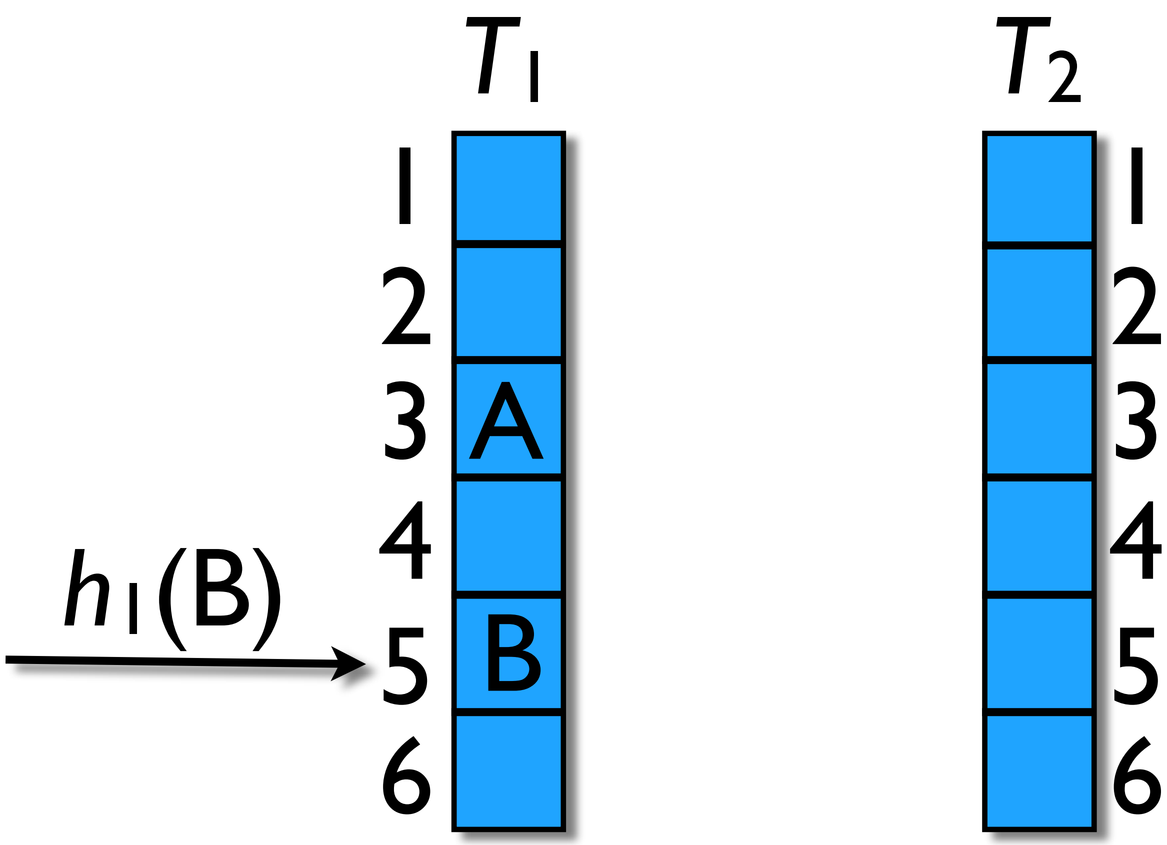
Example



x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

insert(A)

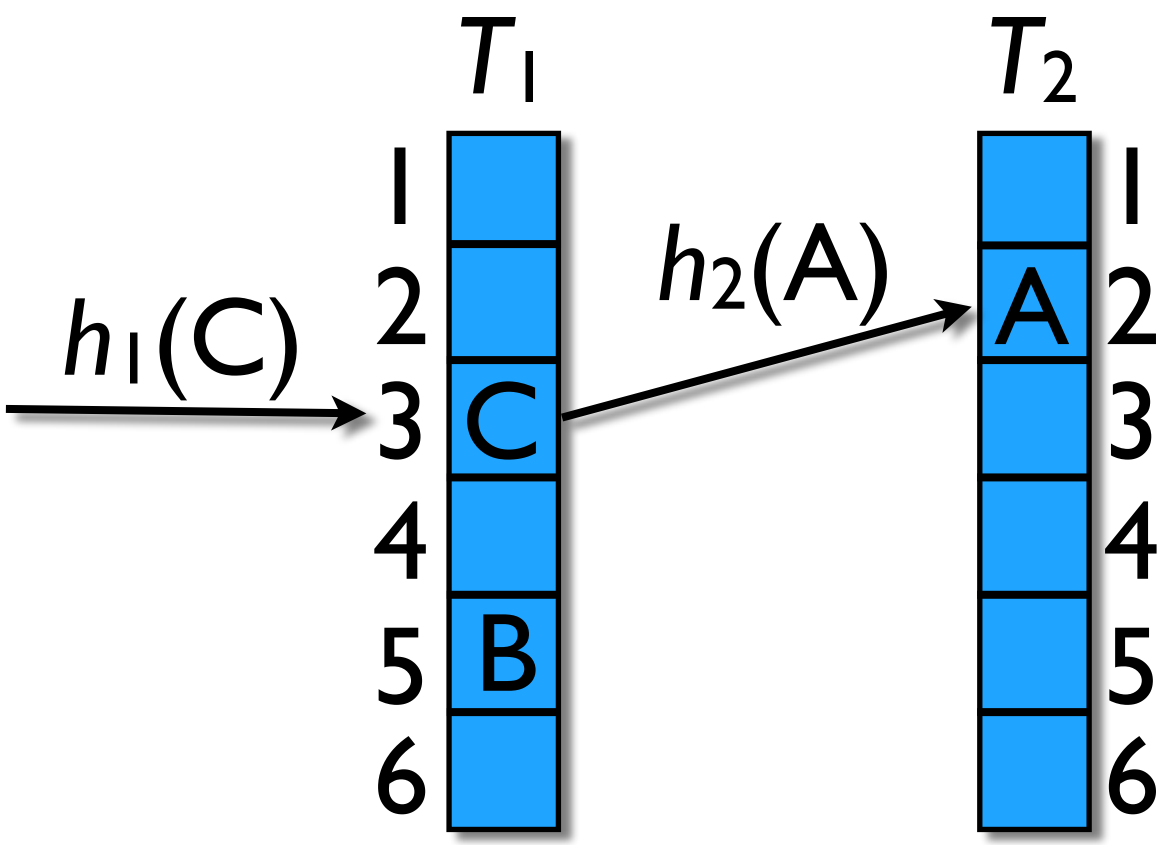
Example



x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

insert(B)

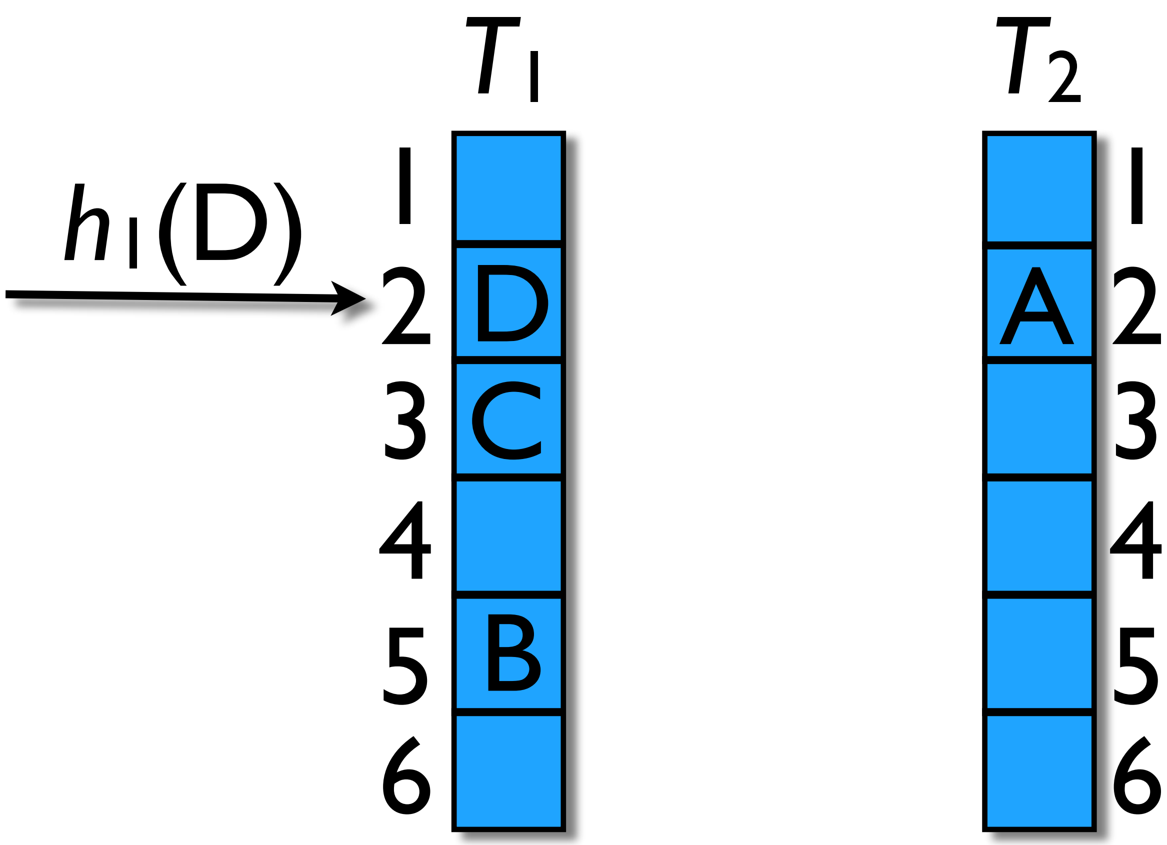
Example



x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

insert(C)

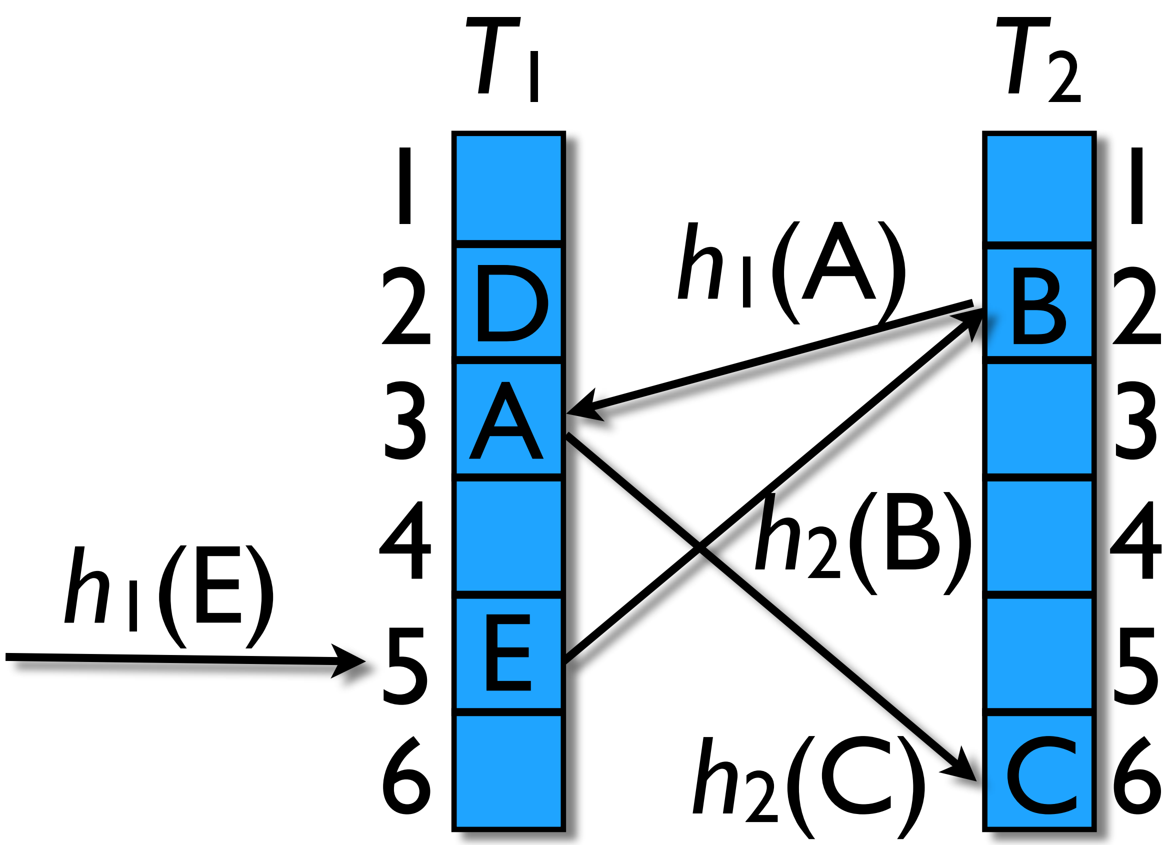
Example



x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

insert(D)

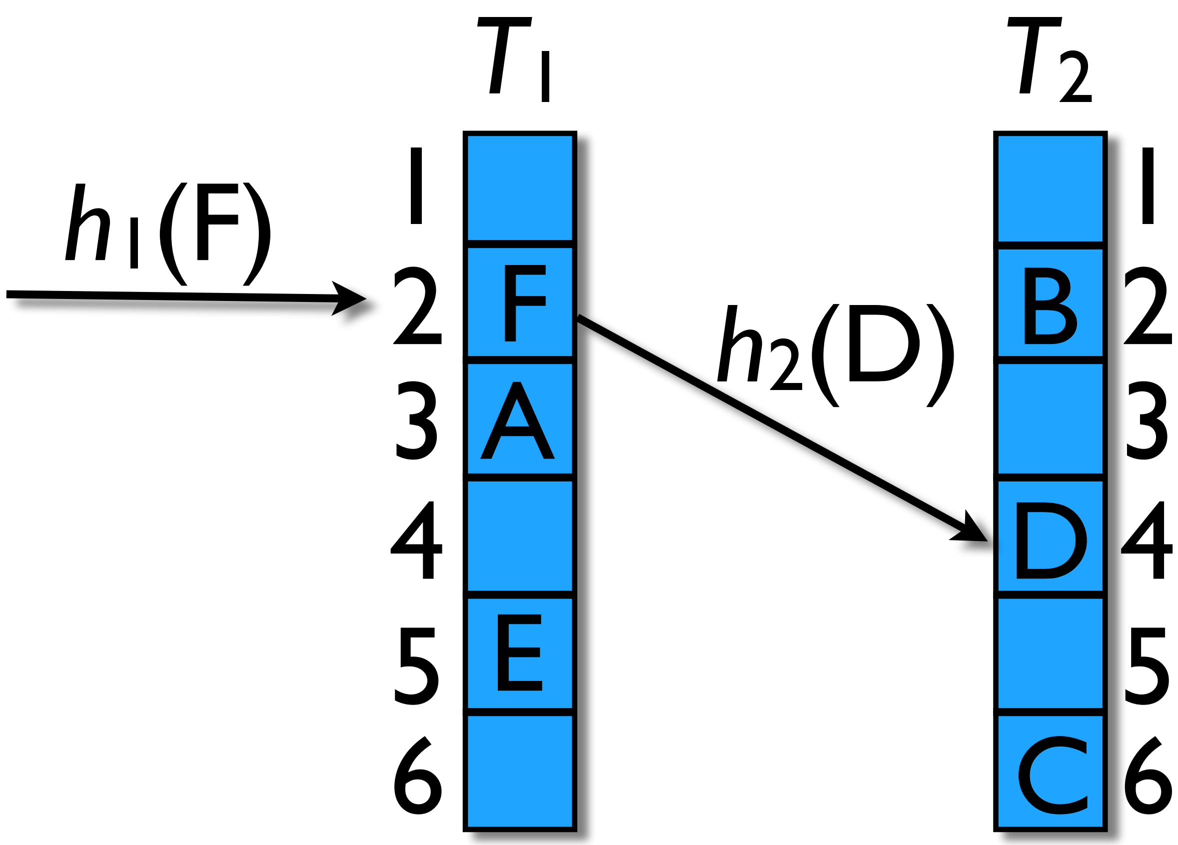
Example



insert(E)

x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

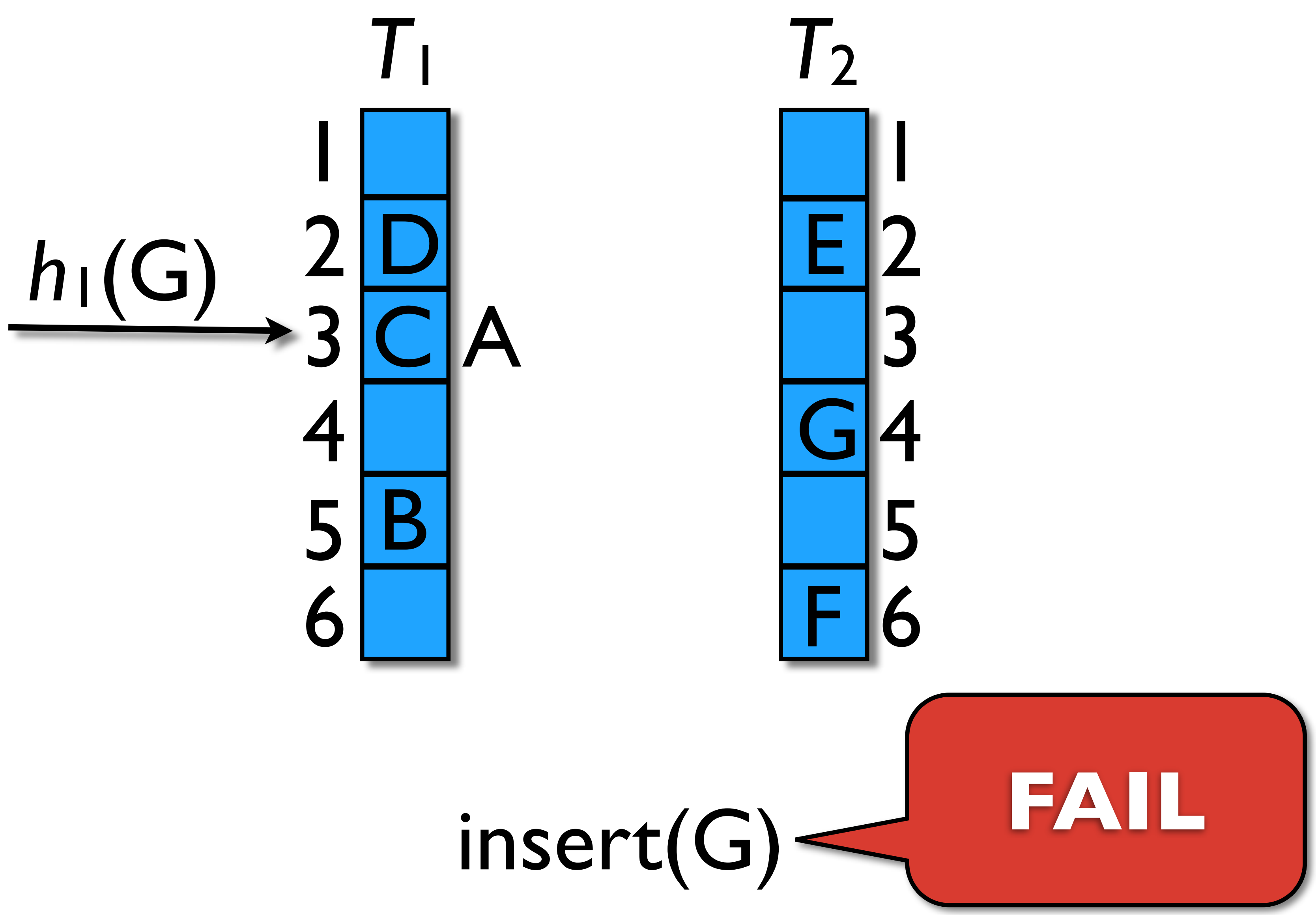
Example



x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

insert(F)

Example



x	*h*₁(*x*)	*h*₂(*x*)
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

(c,k)-Universal Hash Functions

- recall **universal** h.f. h: $[1,u] \rightarrow [1,m]$

  $\text{Prob}[h(x)=h(y)] < {}^1/_m$ for $x \neq y$

- **(c,k)-universal** h.f. h: $[1,u] \rightarrow [1,m]$

  $\text{Prob}[h(x_1)=y_1,\ldots, h(x_k)=y_k] \leq c/m^k$

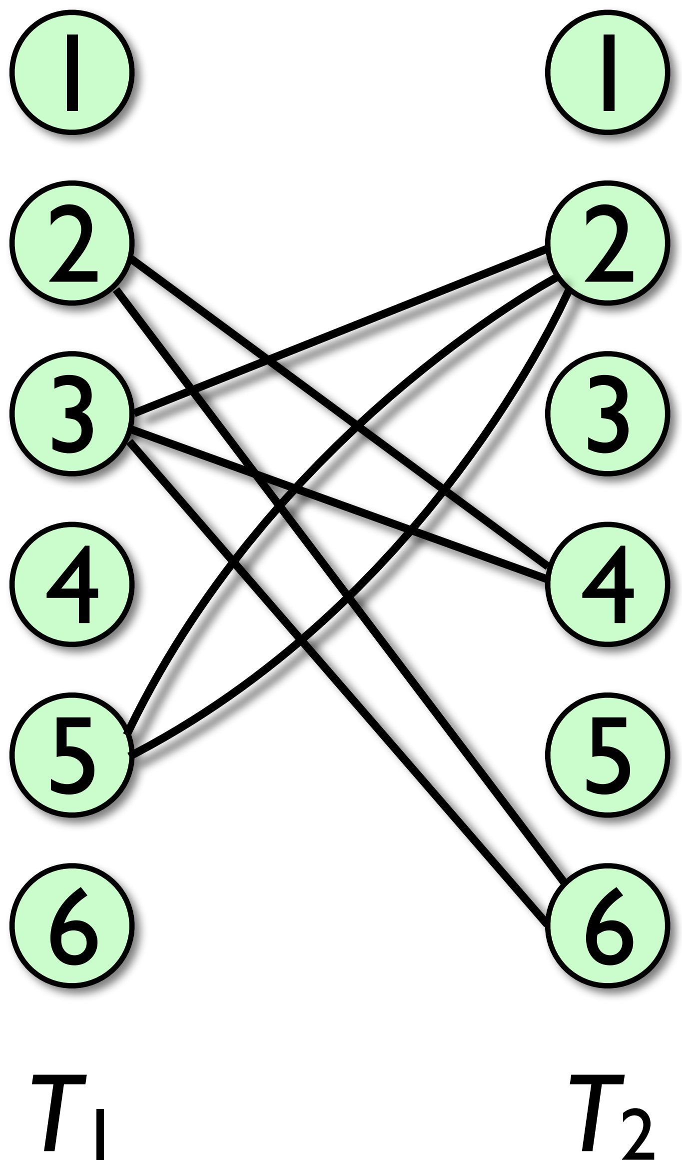
  for fixed $x_i \neq x_j$, fixed $y_1,\ldots,y_k$, and random h

- cuckoo hashing:
  - two $(1,\lg n)$-universal h.f.
  - exist and can be computed efficiently

Cuckoo Graph

- bipartite graph
  - vertices ≜ cells of hash tables
  - edges ≜ possibilities where elements hash
- Formally:
  - $V = \{T_k[x] : 1 \leq x \leq 2n, 1 \leq k \leq 2\} \Rightarrow |V| = 4n$
  - $E = \{(T_1[h_1(x)], T_2[h_2(x)]) : x \in S\} \Rightarrow |E| = n$

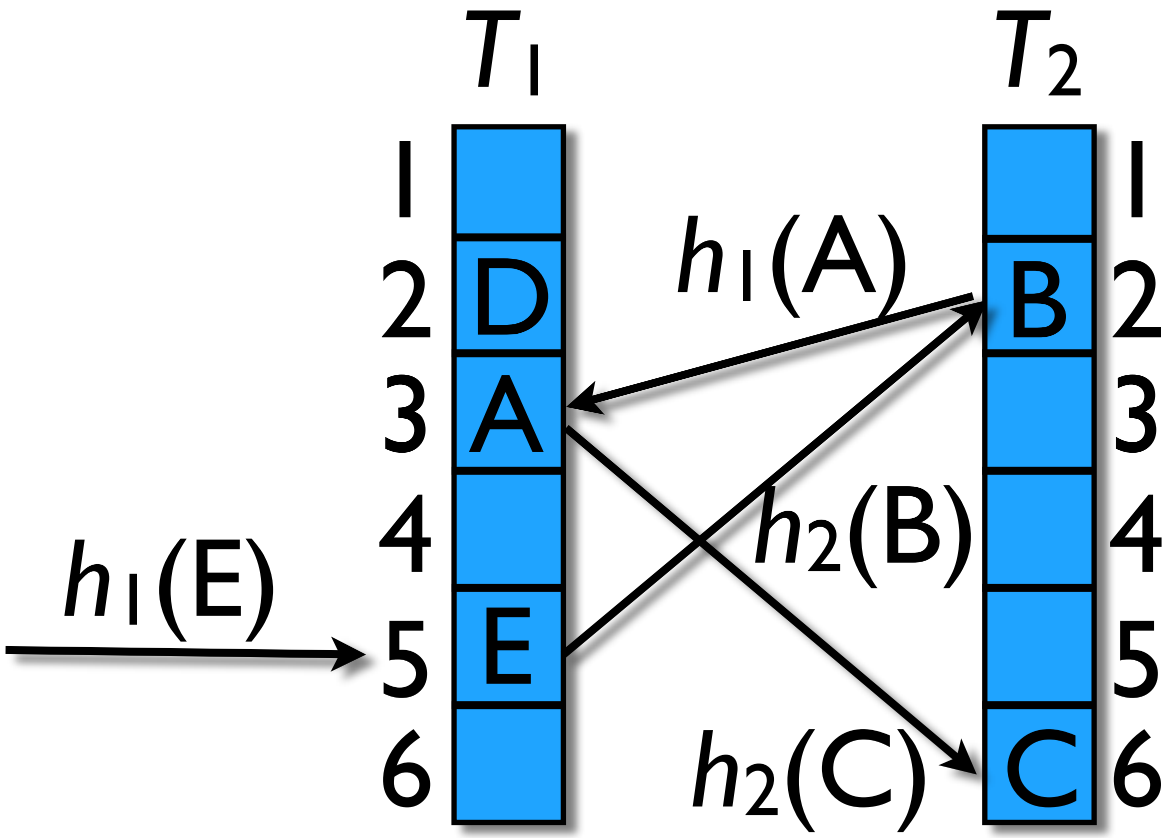
Cuckoo Graph



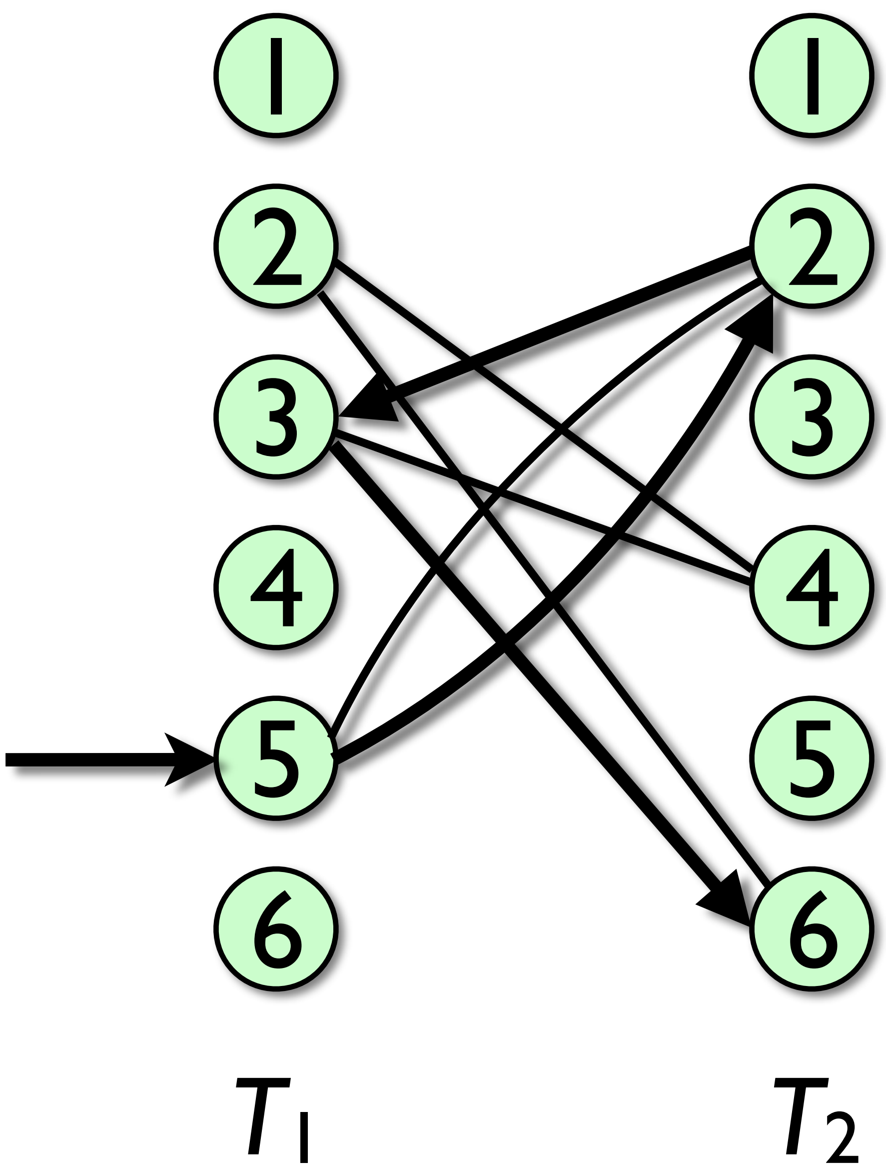
x	$h_1(x)$	$h_2(x)$
A	3	2
B	5	2
C	3	6
D	2	4
E	5	2
F	2	6
G	3	4

- insertions ⇔ walk in cuckoo graph

Example



insert(E)



Analysis

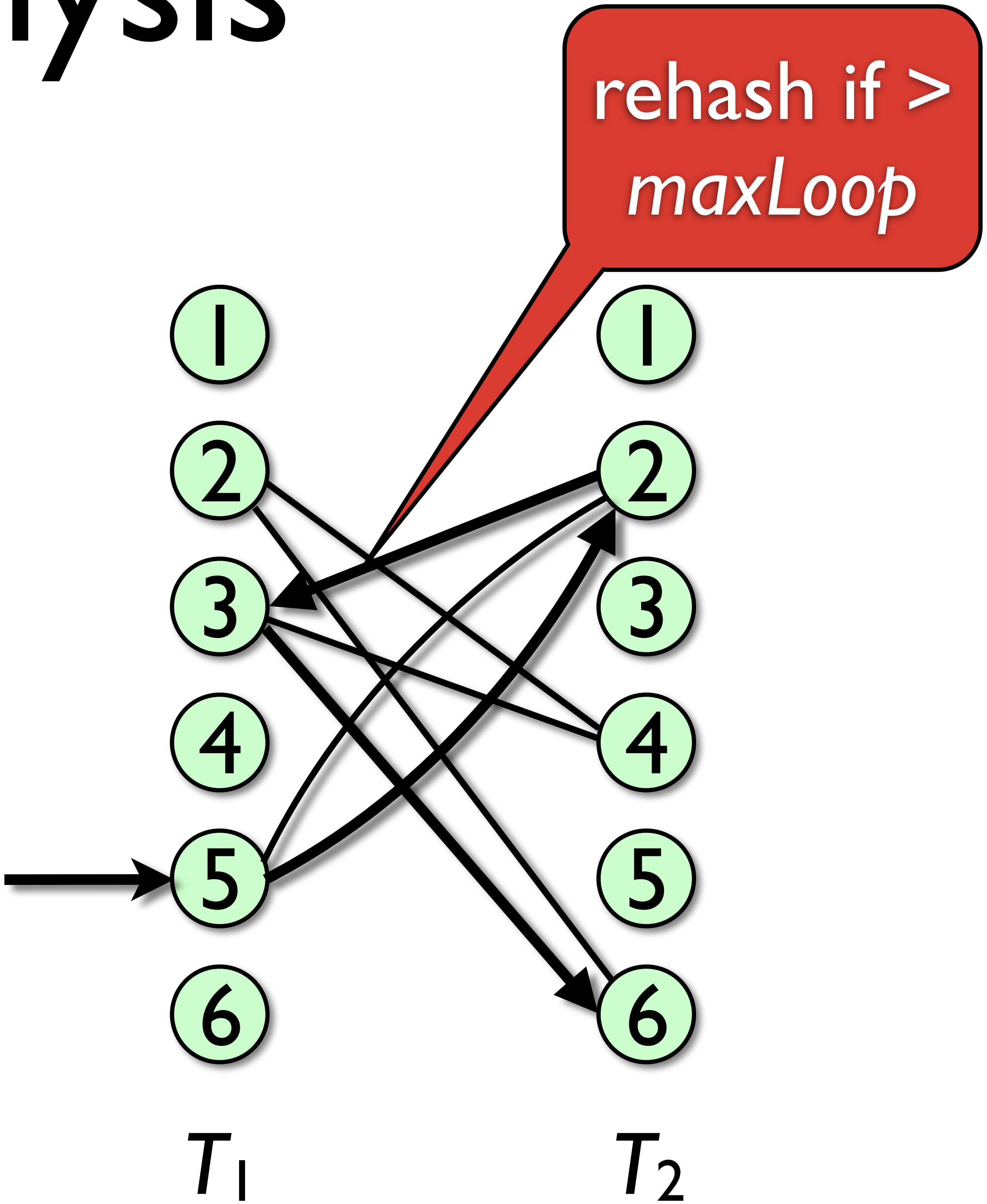
- ⇒ analyze

  probability
  of walks of length
  maxLoop

  - cause rehash!

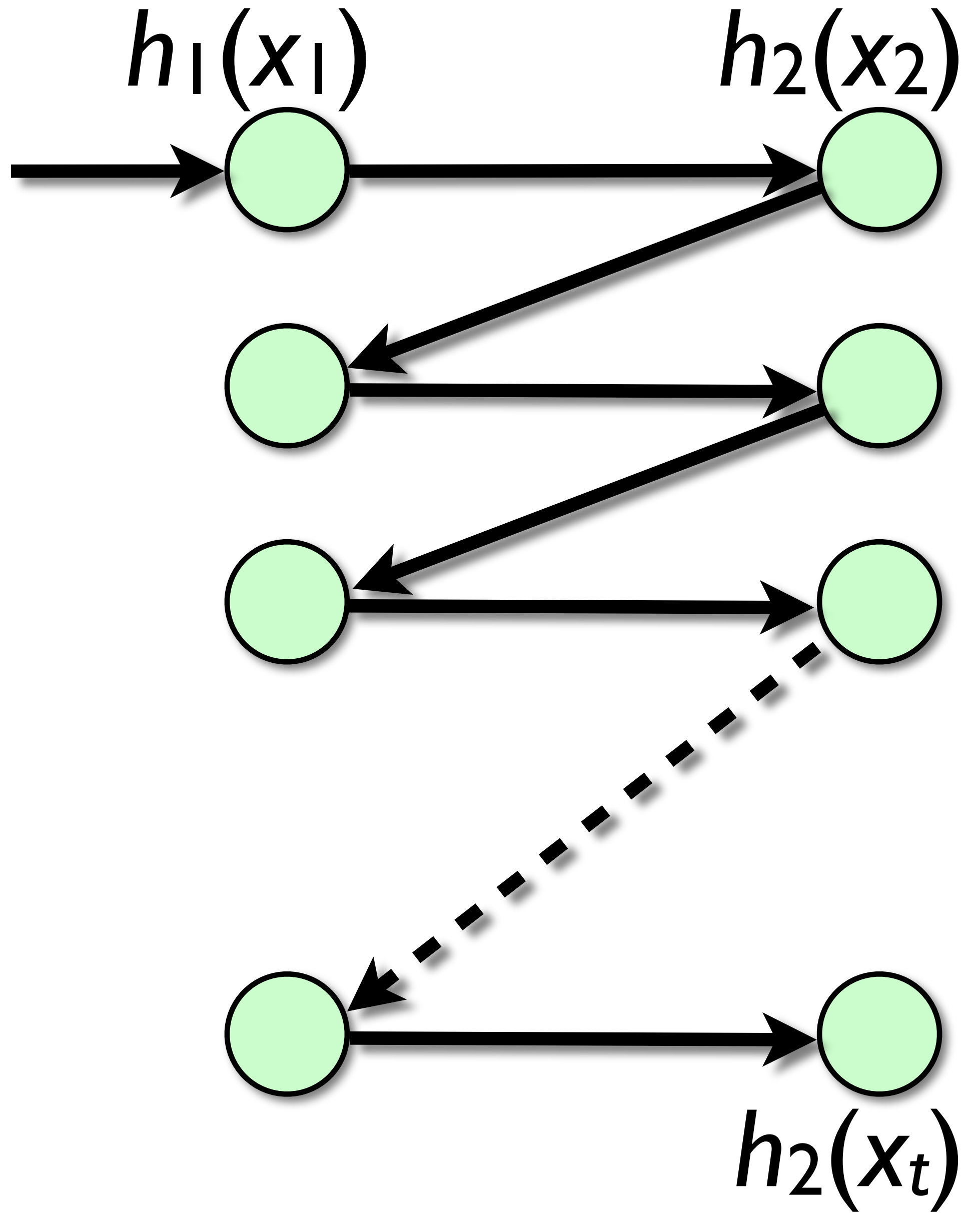
- distinguish 3 cases:
  I. no cycle
  II. 1 cycle
  III. 2 cycles

- fix *maxLoop*=6lg *n*



1: No Cycles

look at walk $h_1(x_1), h_2(x_2), h_1(x_3), \ldots, h_{1/2}(x_t)$
with all vertices **different** ($x_1=x$)



1: No Cycles

$$\text{Prob}[T_1[h_1(x_1)] \text{ occupied}]$$

$$\leq \sum_{y_1 \in S, y_1 \neq x_1} \text{Prob}[h_1(x_1) = h_1(y_1)]$$

$$\leq (n-1)\frac{1}{m} \leq \frac{1}{2}$$

$$\text{Prob}[T_1[h_1(x_1)] \text{ and } T_2[h_2(x_2)] \text{ occupied}]$$

$$\leq \sum_{y_1, y_2 \in S} \text{Prob}[h_1(x_1) = h_1(y_1), h_2(x_2) = h_2(y_2)]$$

$$\leq \sum_{y_1, y_2 \in S} \text{Prob}[h_1(x_1) = h_1(y_1)] \cdot \text{Prob}[h_2(x_2) = h_2(y_2)]$$

$$\leq n^2 \cdot \frac{1}{m^2} = \frac{1}{4}$$

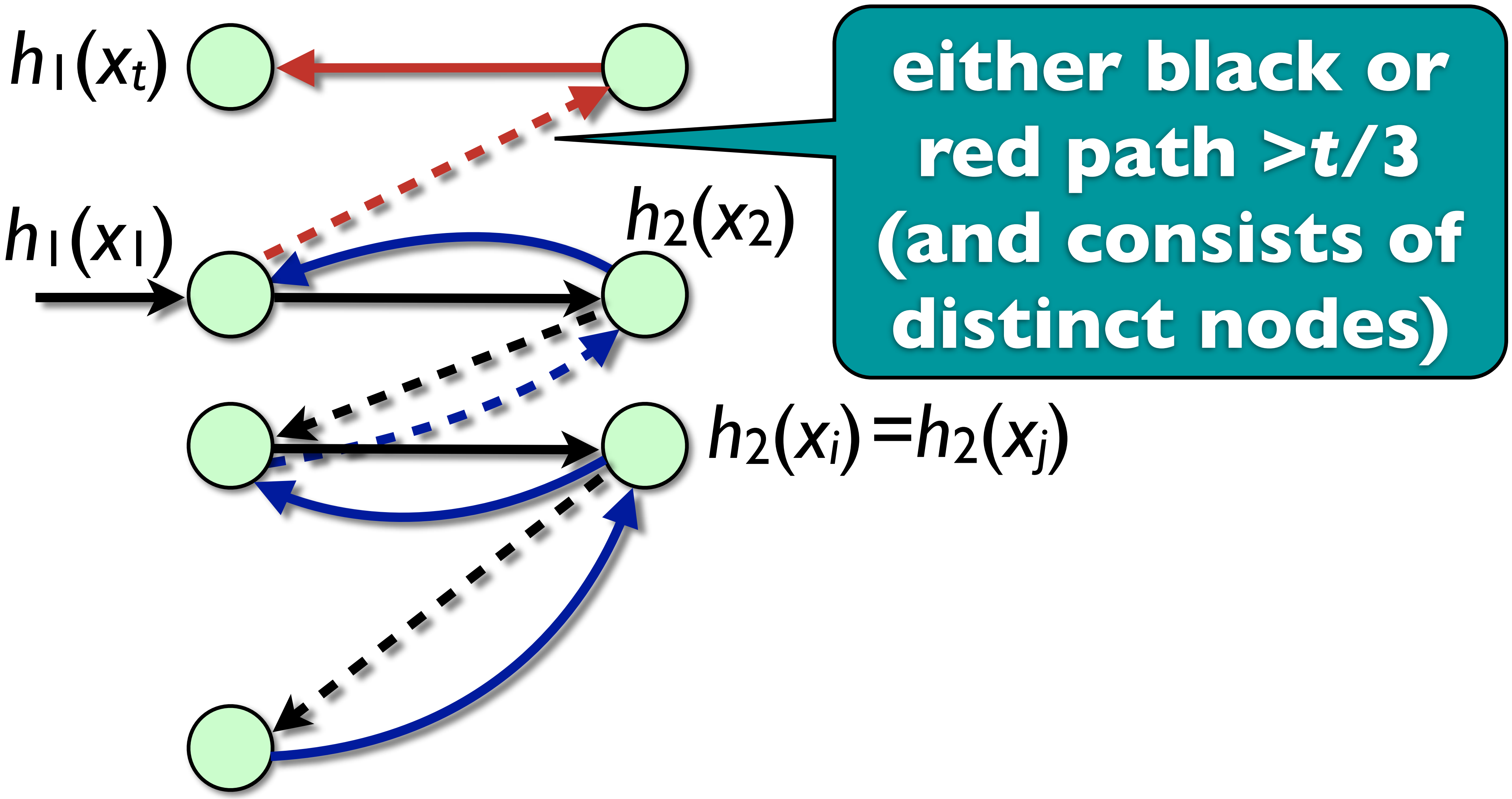
I: No Cycles

- Prob[t cells occupied] $\leq 1/2^t = 1/n^6$
  $\Rightarrow$ rehash occurs with probability $\leq 1/n^2$

- Expected running time:
  Exp[#cells occupied] $\leq \sum_{1 \leq t \leq 6 \lg n} t \cdot 2^{-t}$
  $\leq \sum_{t \geq 1} t \cdot 2^{-t}$
  $(= 2) = O(1)$

II: One Cycle

look at walk $h_1(x_1), h_2(x_2), h_1(x_3), ..., h_{1/2}(x_t)$
with cycle from $h_{1/2}(x_j)$ to $h_{1/2}(x_i)$ with $i<j$



II: One Cycle

- Prob[rehash]
  < Prob[t/3 **different** cells occupied]
  $\leq 2^{-t/3} = 1/n^2$
  $\Rightarrow$ rehash occurs with probability $\leq 1/n^2$
- Expected running time again $O(1)$