

Perfect Hashing in Theory and Practice

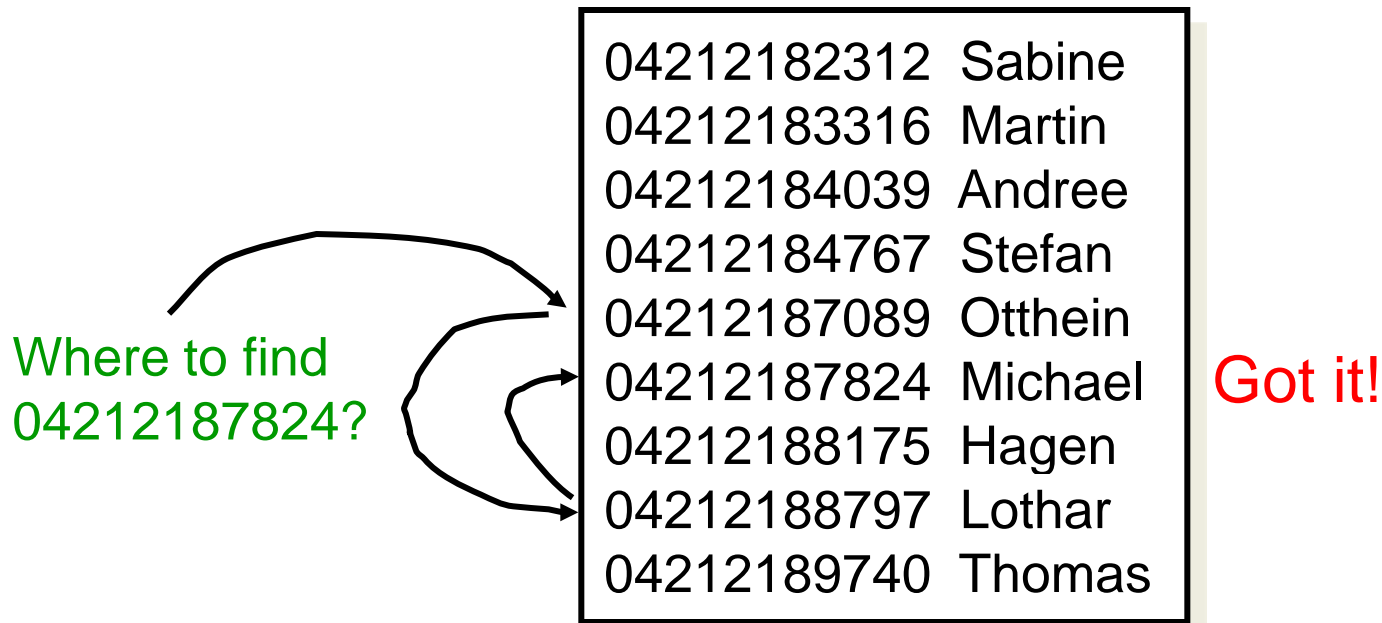
Stefan Edelkamp

- ▶ Motivation Hashing
- ▶ Universal and Perfect Hashing
- ▶ Dynamic Perfect Hashing
- ▶ Perfect Hashing in Permutation Games
- ▶ Perfect Hashing in Selection Games
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

- ▶ **Motivation Hashing**
- ▶ Universal and Perfect Hashing
- ▶ Dynamic Perfect Hashing
- ▶ Perfect Hashing in Permutation Games
- ▶ Perfect Hashing in Selection Games
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

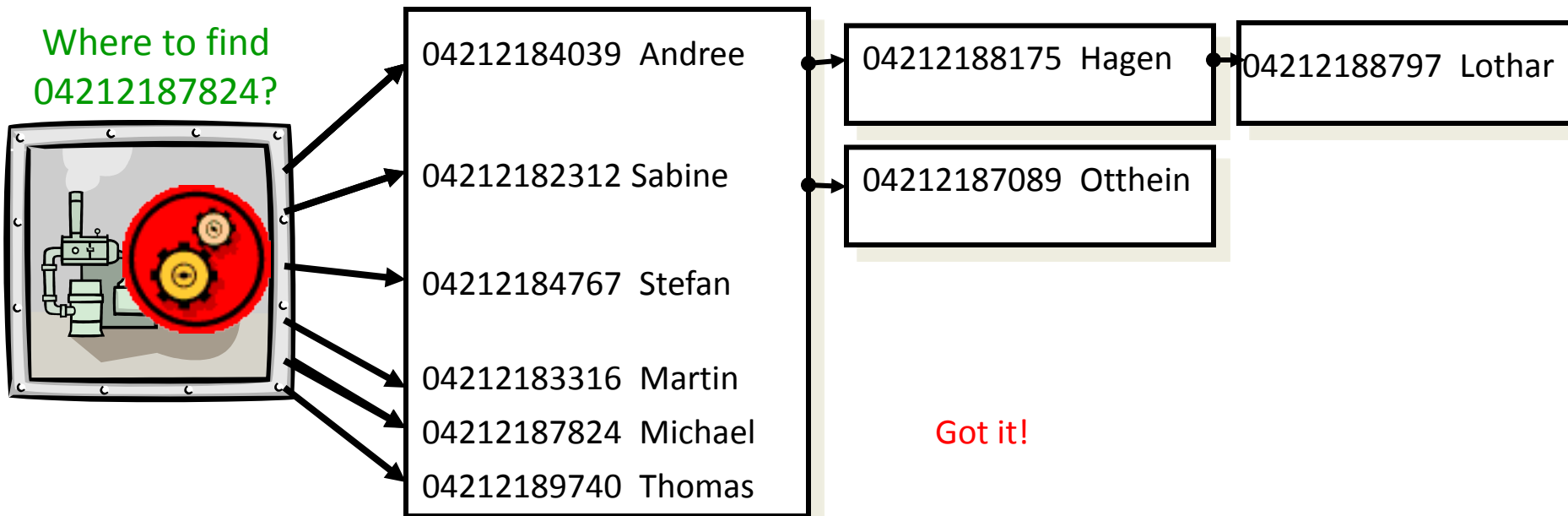
Motivation: Find a Phone Number

a) In a sorted list



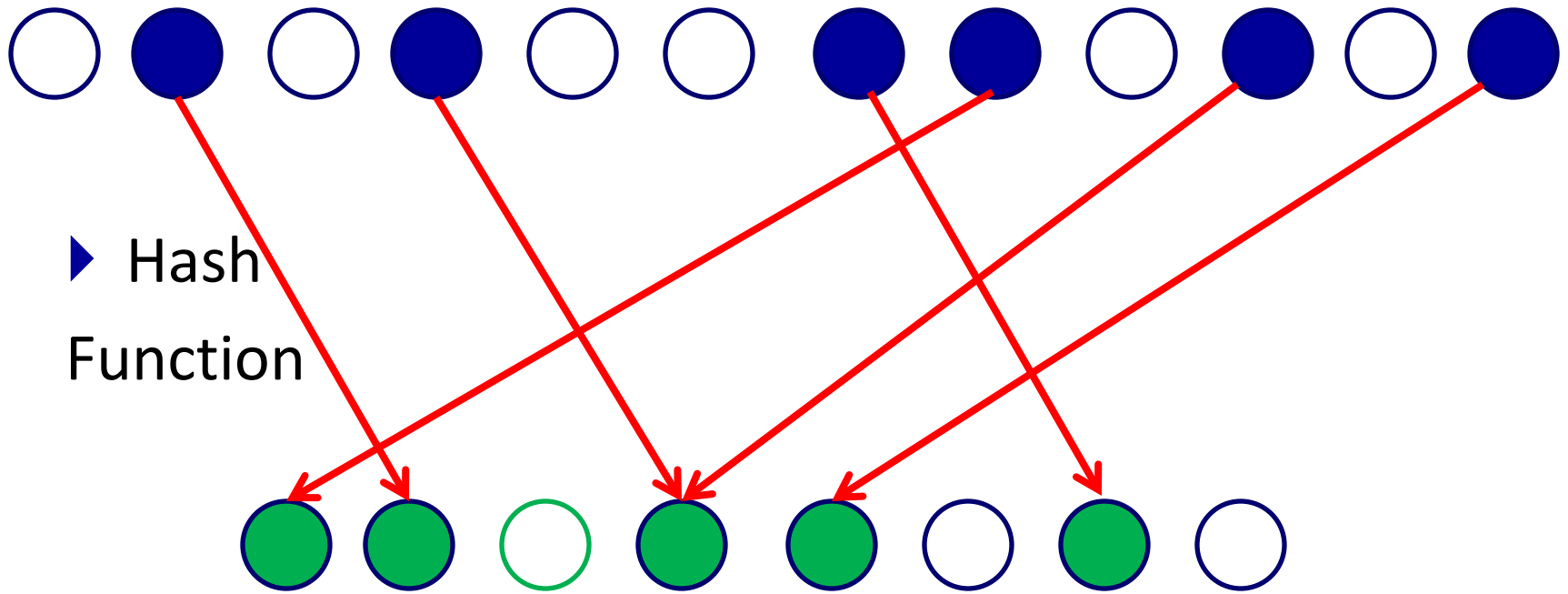
Number of steps *increases* when the amount of information grows.

b) In a Hash Table



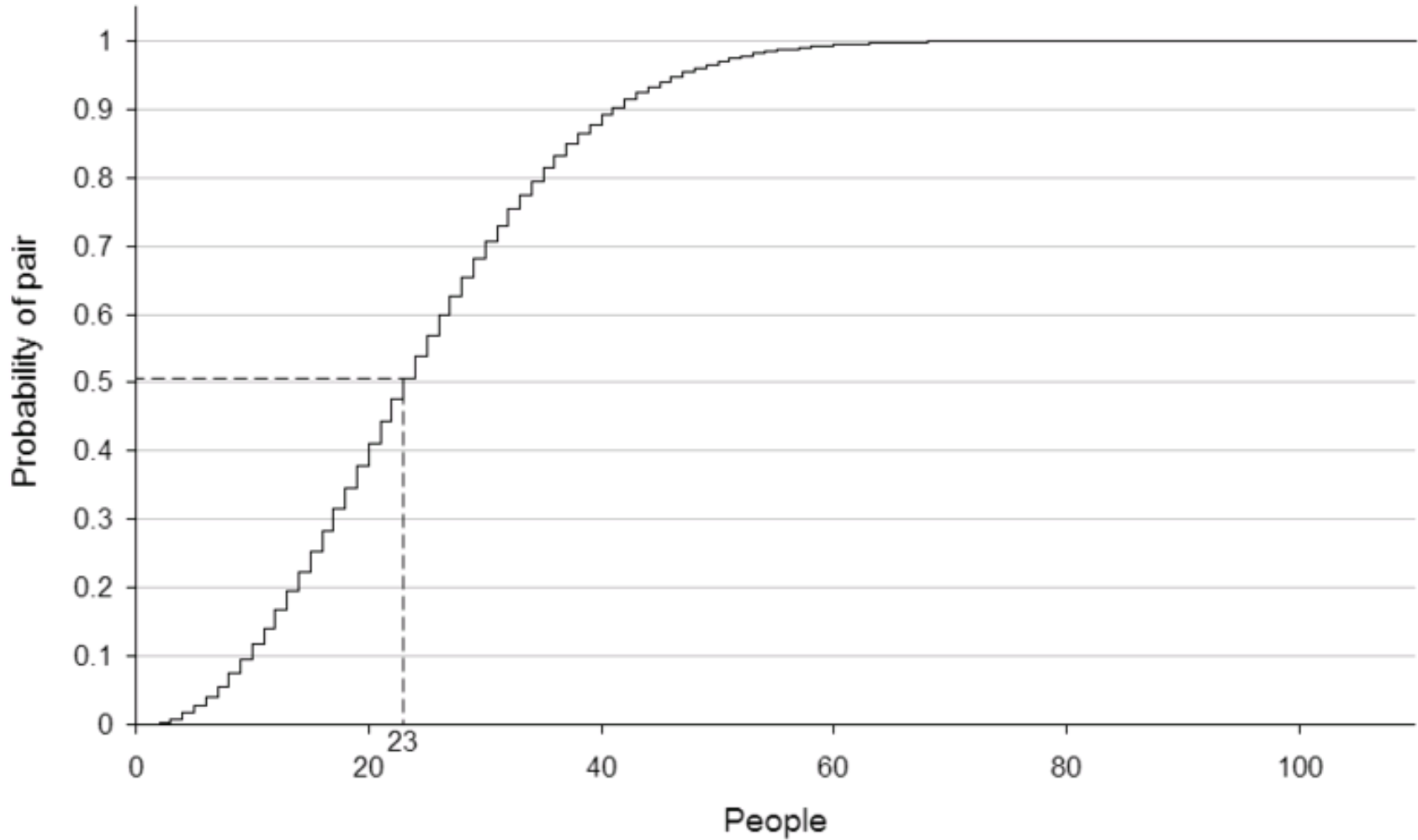
Use a "hash function" to generate and remember random locations.

- ▶ Set of Keys S , Universum of all possible Keys U



- ▶ Hash Function

- ▶ Hash Table $T=[0,\dots,m-1]$



$P(A')$ = probability of not being any two people having the same birthday

$$= 365/365 \times 364/365 \times 363/365 \times \dots \times 343/365$$

$$= 365!/342! \times (1/365)^{23} = 0.49270276$$

$\Rightarrow P(A)$ = probability of two people having the same birthday

$$\Rightarrow = 1 - 0.49270276 = 0.507297 \text{ (50.729\%)}$$

State space of $n = 2^{30}$ elements uniformly hashed to the $m = 2^{64}$ possible words \rightarrow

$$\begin{aligned}P(A') &= m! / (m^n (m - n)!) = m \cdot \dots \cdot (m - n + 1) / m^n \\&= m/m \cdot \dots \cdot (m - n + 1)/m \\&\geq ((m - n + 1)/m)^n \geq (1 - n/m)^n\end{aligned}$$

$$\begin{aligned}(1 - 2^{-34}) 2^{30} &\geq (.9999999999994179233909)^{1073741824} = \\&(.9999999999994179233909)^{(10737)^{1000000} + 41824} \\&= 93.94\%\end{aligned}$$

Game Players do it anyway, e.g. Checkers is „solved“

- ▶ Motivation Hashing
- ▶ **Universal and Perfect Hashing**
- ▶ Dynamic Perfect Hashing
- ▶ Perfect Hashing in Permutation Games
- ▶ Perfect Hashing in Selection Games
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

Idea: draw hash function randomly from class

$$H: \{h \mid h: U \rightarrow [0..m-1]\}$$

Def. H is **universal** if for any x, y in U we have

$$|\{h \text{ in } H \mid h(x)=h(y)\}| \leq |H|/m$$

$$\rightarrow P(h(x)=h(y)) \leq 1/m$$

Example: $H_m = \{h = ((ax+b) \bmod p) \bmod m \mid$
 $a \text{ in } [1..p-1], b \text{ in } [0..p-1]\}$

$$x = 1, y = 4, m = 3, p = 5 \rightarrow$$

$|H| = 20, a \text{ in } [0..3], b \text{ in } [0..4] \rightarrow$ Collisions:

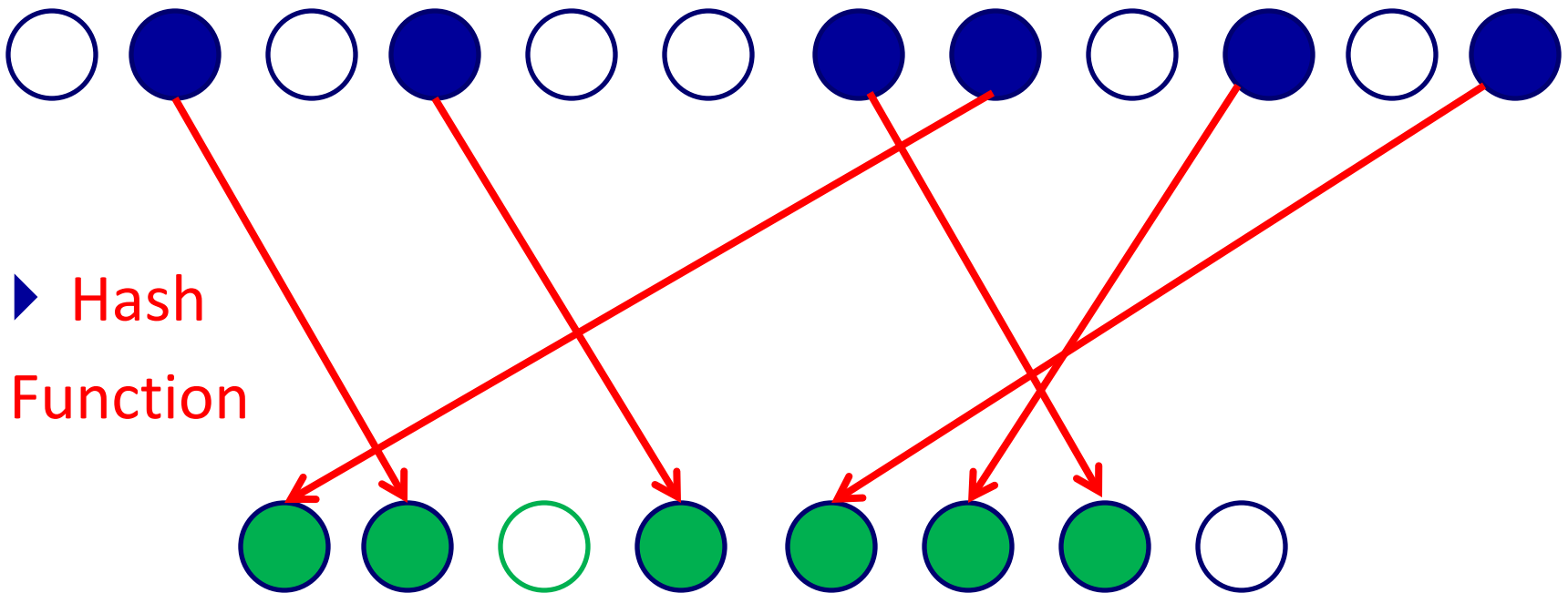
$$(1*1 + 0) \bmod 5 \bmod 3 = 1 = (1*4+0) \bmod 5 \bmod 3$$

$$(1*1 + 4) \bmod 5 \bmod 3 = 0 = (1*4+4) \bmod 5 \bmod 3$$

$$(4*1 + 0) \bmod 5 \bmod 3 = 1 = (4*4+0) \bmod 5 \bmod 3$$

$$(4*1 + 4) \bmod 5 \bmod 3 = 0 = (4*4+4) \bmod 5 \bmod 3$$

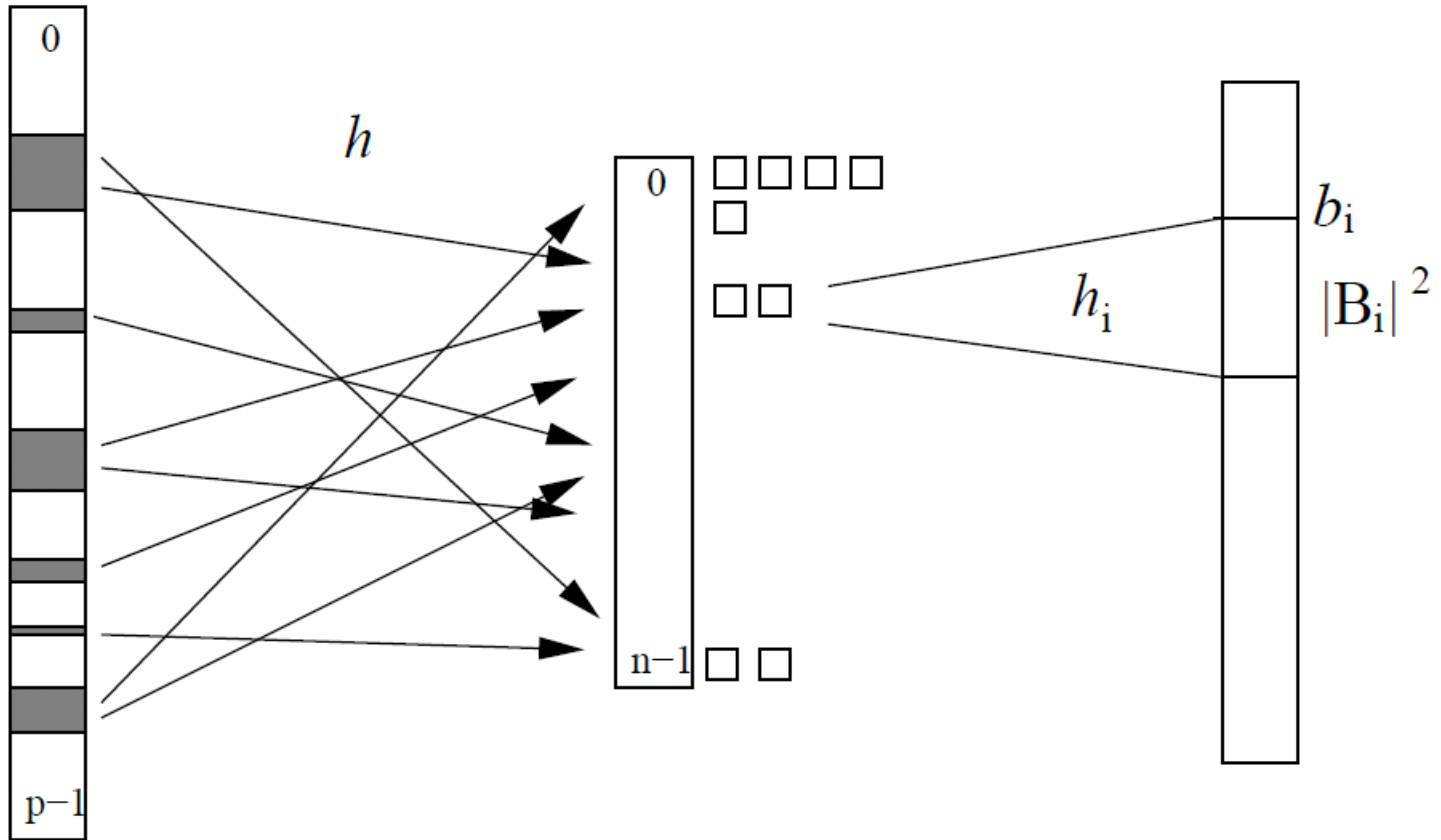
- ▶ Set of **Keys S**, **Universum** of all possible Keys **U**



- ▶ Hash Function

- ▶ Hash Table $T=[0,\dots,m-1]$

Fredman, Komlós and Szemerédi (82)



1. **draw** h in universal class H
2. **for** all x in S compute $i=h(x)$ and B_i
3. **if** $(\sum_i |B_i|^2 \geq 5n)$ **goto** 1
4. **for** i in $[0..m-1]$
 - a) **draw** h_i in universal class H_i
 - b) **If not** (h_i injective) **goto** a)

→ $O(n)$ time, $O(n)$ space

1. $i := h(x)$
2. $T' := T[i]$
3. **extract** (bi , $|Bi|$, hi)
4. **return** $x = T'[bi+hi(x)]$

→ $O(1)$ time

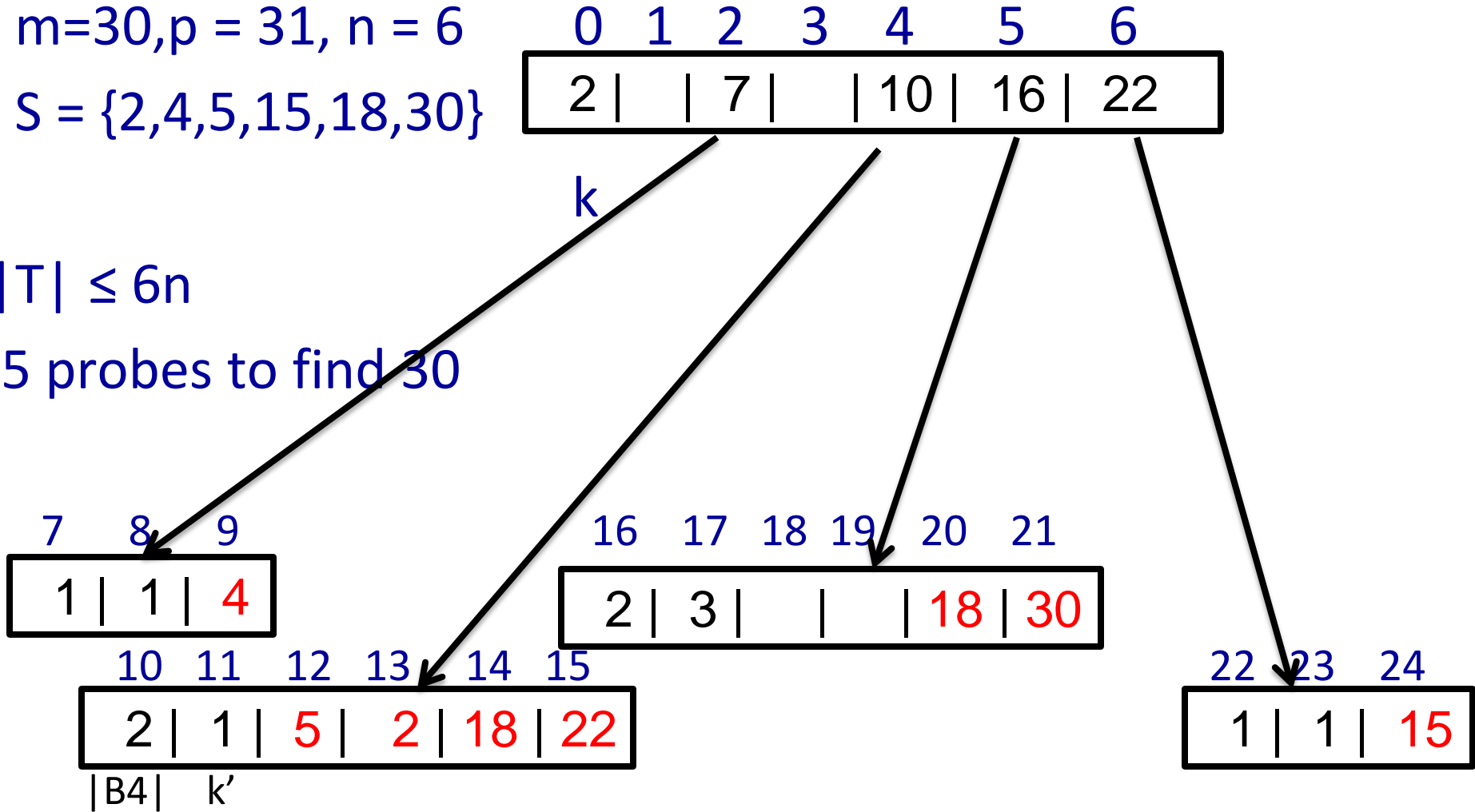
Tzi Example

$m=30, p = 31, n = 6$

$S = \{2,4,5,15,18,30\}$

$|T| \leq 6n$

5 probes to find 30



$H(k,p,s) = \{ h = (kx \bmod p) \bmod s \mid k \text{ in } [1..p-1] \}$ (univ. class)

Thm: $B_i := B(s,S,k,i) = \{ x \text{ in } S \mid h \text{ in } H(k,p,s): h(x)=i \}$

→ **Exists** $k \text{ in } U, s \geq n : \sum_i |B_i| (|B_i|-1)/2 < n^2/s.$

Proof: For pair $(k, \{x,y\})$: $kx \bmod p \bmod s = ky \bmod p \bmod s \rightarrow$

$k*(x-y) \bmod p \bmod s \text{ in } \{s, 2s, 3s, \dots, p-s, p-2s, p-3s\}$

k mult. Inverses mod $p \rightarrow \# k's \leq 2(p-1)/s$

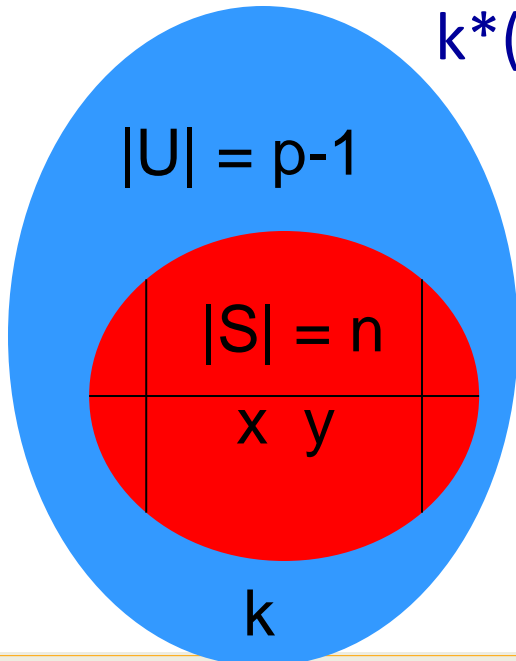
summing over all $(n(n-1)/2)$ pairs $(x,y) \rightarrow$

$\#$ pairs $(k, \{x,y\})$ with

$(kx \bmod p \bmod s = ky \bmod p \bmod s) < (p-1)n^2/s \rightarrow$

$\rightarrow \sum_k \sum_i |B_i| (|B_i|-1)/2 < (p-1)n^2/s \rightarrow$

$\sum_i |B_i| (|B_i|-1)/2 < n^2/s.$

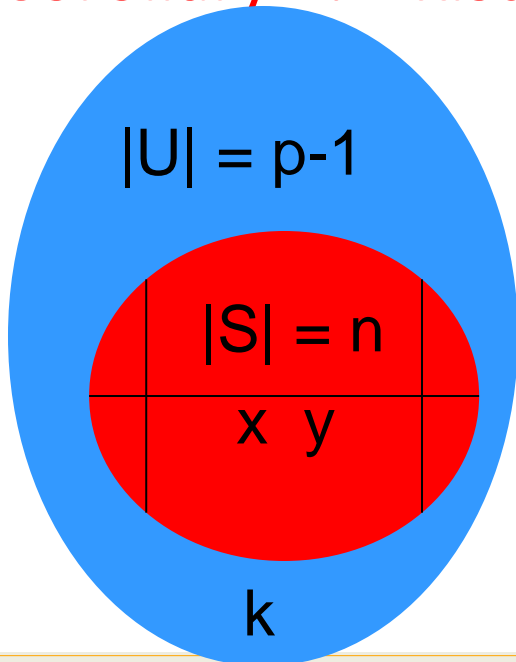


Corollary 1: Exists k in U : $\sum_i |B_i|^2 < 3n$

Proof: $\sum_i |B_i|^2 = 2 * \sum_i |B_i| (|B_i|-1)/2 + 2 * \sum_i |B_i|/2$
 $< 2n^2/s + n \leq 3n$

Corollary 2: Exists k' in U for all i : $|B'_i = B(n^2, S, k', i)| \leq 1$

Proof: $B'_i = \{ x \text{ in } S \mid h = (k'x \text{ mod } p) \text{ mod } n^2: h(x) = i \}$



\rightarrow (Thm) $\rightarrow \sum_i |B'_i| (|B'_i|-1)/2 \leq 1$

$\rightarrow |B'_i| \leq 1$ for all i

$\rightarrow h$ is one-to-one on S

TZi Increasing Probability to Construct FKS

One k in U not sufficient for fast construction, need $\geq |U|/2$

$H = \{h = (kx \bmod p) \bmod n \mid k \in [1..p-1]\}$ univ. class \rightarrow

Corollary 3: For **at least half** of all possible k in U

$$\sum |B_i|^2 < 5n \quad [\text{Stage 1 Hash}]$$

Proof: Most a half values smaller than 2^* average \rightarrow

$$(\text{Thm 1}) \rightarrow \sum_i |B_i| (|B_i| - 1) / 2 < 2n.$$

Corollary 4: For **at least half** of all possible k' in U

$$|B'_i| \leq 1 \quad [\text{Stage 2 Hash}]$$

Proof: Analogous to Corollary 2

Corollary 5: Space can be reduced to $n + o(n)$.

Proof: Substitute n with $g(n)$ given $\lim g(n)/n = 0$

Further Results on (Minimum) Perfect Hashing

Lower bound: At least $\log e = 1.44$ bits per element

(Proof: e.g., Mehlhorn (82)+ Dietzfelbinger et al. (09))

Mehlhorn (82): At least $\theta(n + \log \log |U|)$ bits.

Fredman and Komlós (84): $n \log e + \log \log |U| + \theta(n)$ bits

Schmidt and Siegel (90): Existence of $n + \log \log |U|$ bits for $O(1)$ minimum perfect hash function, no construction

Dietzfelbinger, Gil, Matias, Pipping (92): Generalization of FKS to dynamic setting (via **polynomial hash functions**)

Majewski, Wormald, Havas, Czech (96): $O(1)$ perfect hashing with $O(n \log n)$ bits (using **hypergraph theory**)

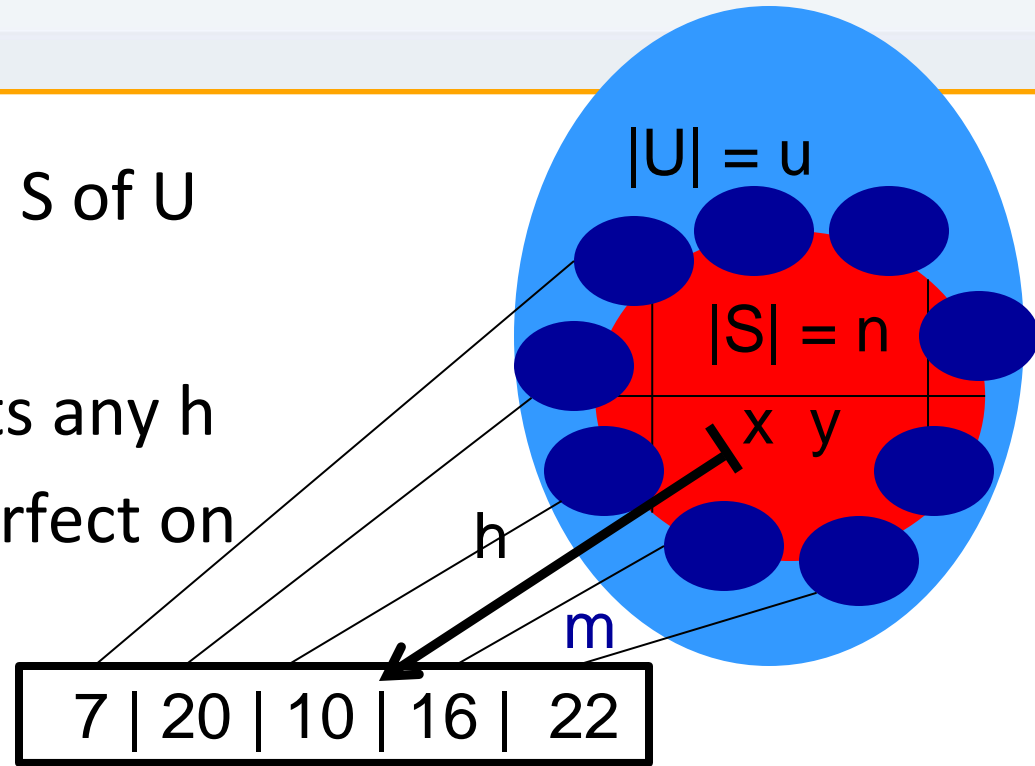
Hagerup/Tholey (01): $O(1)$ minimal perfect hashing, $n \log e + \log \log |U| + O(n(\log \log n)^2 / \log n + \log \log \log |U|)$ (01)

Edelkamp/Meyer (01): Suffix-Lists with space close to lower bound

$$|H| \geq \frac{\binom{u}{n}}{\binom{u}{m}^n \binom{m}{n}}$$

subsets S of U

sets any h is perfect on



$$\log |H| \geq \sum \log(1 - i/u) - \sum \log(1 - i/m) \rightarrow \text{NR-1} \rightarrow$$

$$\log |H| \geq (m-n+1) \log (m-n+1/m) - (n-u) \log (1-n/u)$$

$u \gg n \rightarrow \text{NR-2} \rightarrow m = 1.23n$ requires ≥ 0.89 bits per key

$\rightarrow m = n$ requires ≥ 1.44 bits per key

Botelho, Pagh and Ziviani (07) Implementation of memory-based hash function based FKS + hypergraph theory

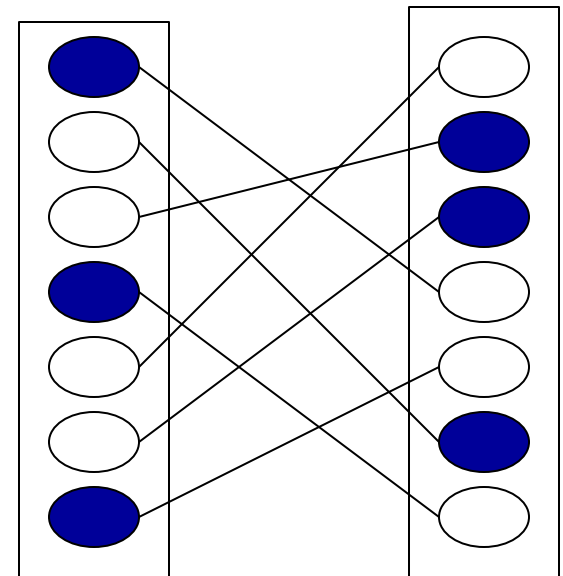
- r -uniform random hypergraphs with r hash functions on the keys of S ; $r=2$: Two tables of size $(2+\epsilon)m$, divide in $m=255$
- For $r = 3$ they obtained a space usage of approximately $2.62n$ bits for a minimum perfect hash function

Features:

- + Hash value computed efficiently
- + Constant access to identifier
- All keys need to be known beforehand

Botelho and Ziviani (07)

Works well for data stored on **disk**



cmph: C Library for Minimum Perfect Hash Functions

<http://cmph.sourceforge.net/>

Currently best: HDC based on „Hash Displace and Compress“ by Belazzougui, Botelho and Dietzfelbinger (09)

(...following Hash & Displace by Pagh (99) and Tarjan/Yao (79))

$O(n)$ construction $O(1)$ evaluation, close to $1.44n$ bits, e.g

- For $m = n$ it has 2.07 bits per key,
- For $m = 2n$ it has 0.67 bits per key,
- For $m = 1.23n$ it has 1.4 bits per key

SUX4J: Succinct Data Structures Umbrella (<http://Sux4j.dsi.unimi.it>)

Java based **minimal perfect hashing** using ~ 2.65 bits per element + implementations of **monotone minimal perfect hashing**...

Known: Given that keys can be in any order → order-preserving hash requires $\theta(n \log n)$ bits

Belazzougui, Boldi, Pagh, Vigna (09) show (& implement)

Thm: Given keys in lexicographic order

- a) $O(n \log \log \log |U|)$ space and $O(\log \log |U|)$ search time
- b) $O(n \log \log |U|)$ space and $O(1)$ search time

Use $O(1)$ **rank** and **select**

- $\text{rank}(p, 0001001010) = \# 1 \text{ till } p$
- $\text{select}(r, 0001001010) = \text{position of } r \text{ th } 1$

Bucketing a) longest common prefixes b) relative ranking

- ▶ Motivation Hashing
- ▶ Universal and Perfect Hashing
- ▶ **Dynamic Perfect Hashing**
- ▶ Perfect Hashing in Permutation Games
- ▶ Perfect Hashing in Selection Games
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

Pagh & Rodler (03)

2 Tables of size $n + \epsilon$, 2 univ. hash functions

- ▶ $O(1)$ look-up
- ▶ Fast insert



Search: The hash function provides *two* possible locations.

04212188175 Hagen

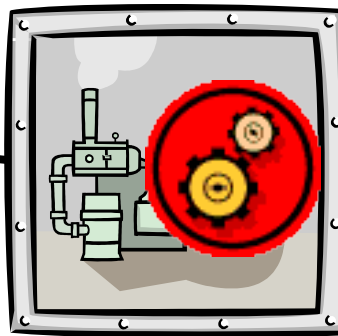
Not here

04212183316 Martin

04212188797 Lothar

04212184039 Andree

Where to find
04212187824?



Got it!

04212187824 Michael

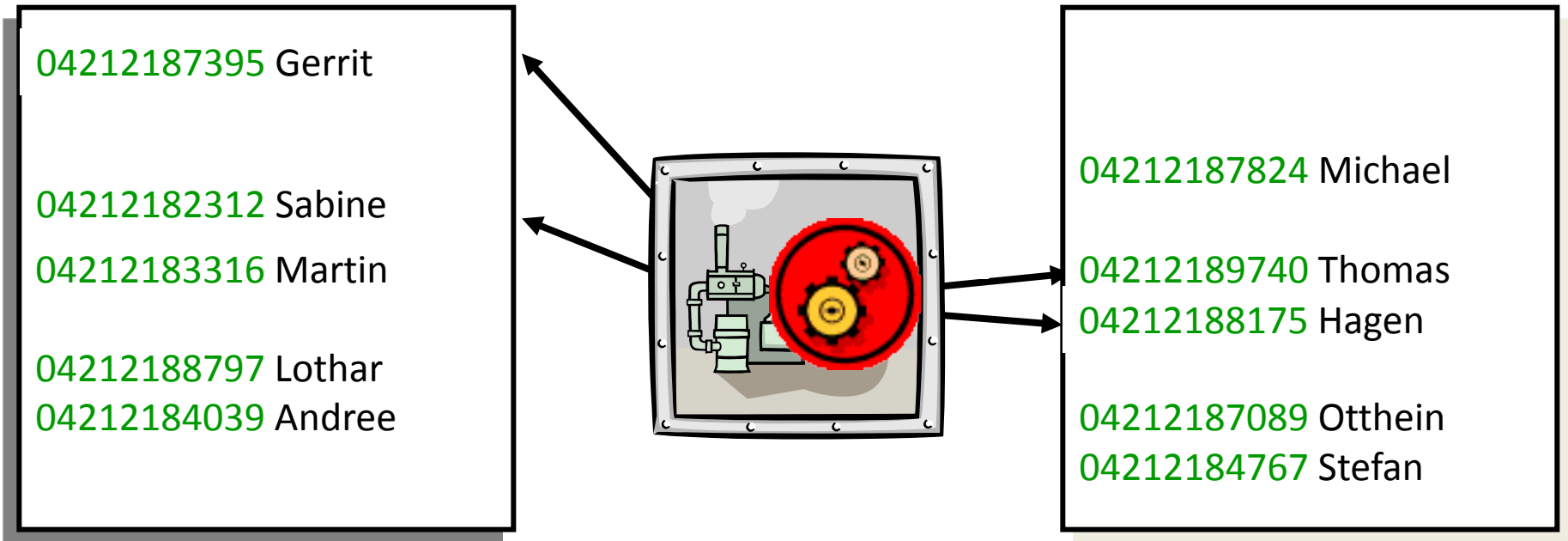
04212187089 Otthein

04212182312 Sabine

04212189740 Thomas

04212184767 Stefan

Insert: New information is inserted; if necessary, kick out old information.



1. Compute $h_1(x)$
2. If $T[h_1(x)]$ empty, $T[h_1(x)] := x$
else $y := T[h_1(x)]$ and $T[h_1(x)] := x$
3. Look at $T[h_1(y)]$ and $T[h_2(y)]$ that is not occupied by x . If empty, insert y . If not, put y there and evict z . Set $x := y$ and $y := z$
4. Goto 3 $O(\log n)$ times until an empty spot is found. Otherwise, pick a new pair of hash functions and rehash.

Bad case 1: inserted element runs into cycles.

Bad case 2: inserted element has very long path before insertion completes (Could be on a long cycle).

Observation: Bad cases occur with small probability when load is sufficiently low

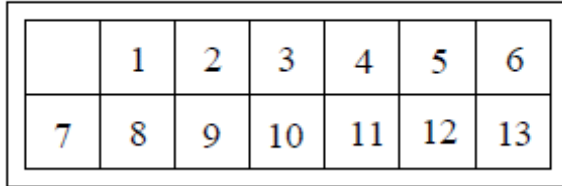
Solution: re-hash everything if a failure occurs.

Load less than 50%, n elements gives failure rate of $\Theta(1/n)$; maximum insert time $O(\log n)$

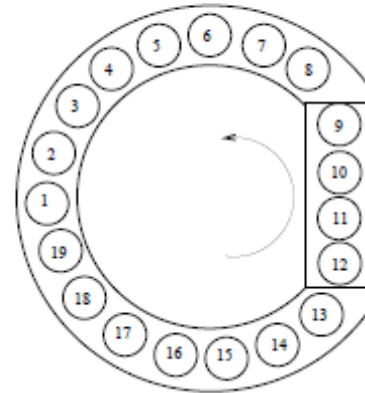
Better Space Utilization: Dietzfelbinger & Weidling (07) generalize cuckoo hashing to **bucketed cuckoo hashing** that uses more than 1 entry per table

- ▶ Motivation Hashing
- ▶ Universal and Perfect Hashing
- ▶ Dynamic Perfect Hashing
- ▶ **Perfect Hashing in Permutation Games**
- ▶ Perfect Hashing in Selection Games
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

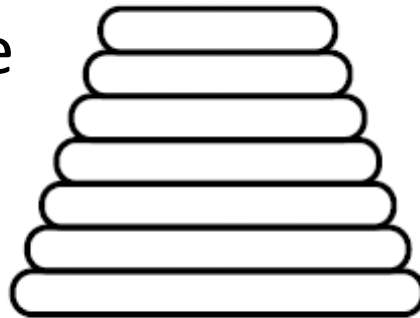
$n \times m$ Sliding Tile



n-TopSpin



n-Pancake



- ▶ Assumes Perfect Hash Function **Rank**
- ▶ Applies to State Space Search, if $m \geq |\text{Reach}(\text{init})|$
- ▶ **Minimal**, if $m = |S|$
- ▶ Inverse **Unrank** needed to reconstruct state
- ▶ Features:
 - **Two-Bit BFS** (Cooperman/Finkelstein 92, Korf 08)
 - **One-Bit Breadth-First Search**
 - **One-Bit Reachability** (Edelkamp/Sulewski 09)

Interpretation: 3=UNSEEN, {0,1,2} = depth mod 3

E: 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3

G: 3 3 3 3 3 3 2 3 3 1 3 3 2 3 3 3 3 3 3 3 3 3 3 3

E: 3 3 3 3 3 3 2 3 3 1 3 3 2 3 3 3 3 3 3 3 3 3 3 3

G: 3 3 3 0 3 3 2 3 3 1 3 3 2 3 3 3 0 3 3 3 3 3 3 3

E: 3 3 3 0 3 3 2 3 3 1 3 3 2 3 3 3 0 3 3 3 3 3 3 3

...

Tzi One-Bit Reachability

E: 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

G: 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

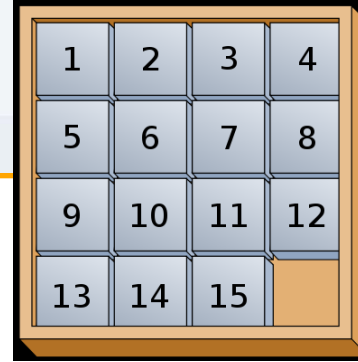
E: 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0

G: 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0

E: 1 0 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0

...

→ **Thm:** $\text{Scan}(i) \leq \text{Depth}(i)$



Solvable states have same **parity** as the goal

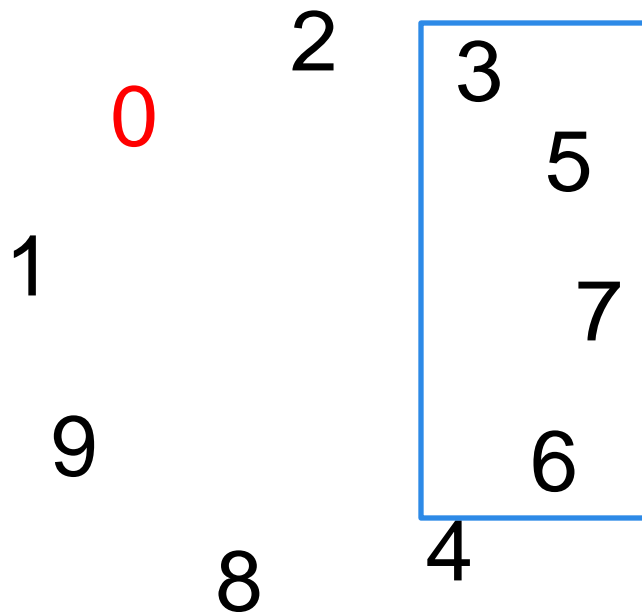
→ **parity** is key concept to rank state to $[0..(nm)!/2-1]$

Caution: Swapping a tile with the blank is move

→ partition state space wrt. „blank“ position

Obs: Moving blank left from B_i to B_{i-1} or right from B_i to B_{i+1} does not change the rank

Proof: Relative order of tiles does not change



Normalization

▶ 0 2 3 5 7 6 4 8 9 1

Thm: If n even, k odd then **parity** remains unchanged

→ Compression to
 $[0..(n-1)!/2-1]$

Lexicographic

- Permutations with opposite **parity** next to each other → compression
- **No** linear time and space algorithm known

Myrvold Ruskey (05)

- **linear time and space**
- News (Edelkamp & Sulewski 09): including **parity** computation

Algorithm $rank(n, \pi, \pi^{-1})$

- 1: **for all** i **in** $\{1, \dots, n - 1\}$ **do**
 - 2: $l \leftarrow \pi_{n-i}$
 - 3: $swap(\pi_{n-i}, \pi_{\pi_{n-i}^{-1}})$
 - 4: $swap(\pi_l^{-1}, \pi_{n-i}^{-1})$
 - 5: $rank_i \leftarrow l$
 - 6: **return** $\prod_{i=1}^{n-1} (rank_{n-i+1} + i)$
-

(1,2,3,0)	0	(2,1,3,0)	1
(3,2,0,1)	0	(2,3,0,1)	1
(1,3,0,2)	0	(3,1,0,2)	1
(1,2,0,3)	1	(2,1,0,3)	0
(2,3,1,0)	0	(3,2,1,0)	1
(2,0,3,1)	0	(0,2,3,1)	1
(3,0,1,2)	0	(0,3,1,2)	1
(2,0,1,3)	1	(0,2,1,3)	0
(1,3,2,0)	1	(3,1,2,0)	0
(3,0,2,1)	1	(0,3,2,1)	0
(1,0,3,2)	1	(0,1,3,2)	0
(1,0,2,3)	0	(0,1,2,3)	1

(1,2,3,0)	0	(2,1,3,0)	1
(3,2,0,1)	0	(2,3,0,1)	1
(1,3,0,2)	0	(3,1,0,2)	1
(1,2,0,3)	1	(2,1,0,3)	0
(2,3,1,0)	0	(3,2,1,0)	1
(2,0,3,1)	0	(0,2,3,1)	1
(3,0,1,2)	0	(0,3,1,2)	1
(2,0,1,3)	1	(0,2,1,3)	0
(1,3,2,0)	1	(3,1,2,0)	0
(3,0,2,1)	1	(0,3,2,1)	0
(1,0,3,2)	1	(0,1,3,2)	0
(1,0,2,3)	0	(0,1,2,3)	1

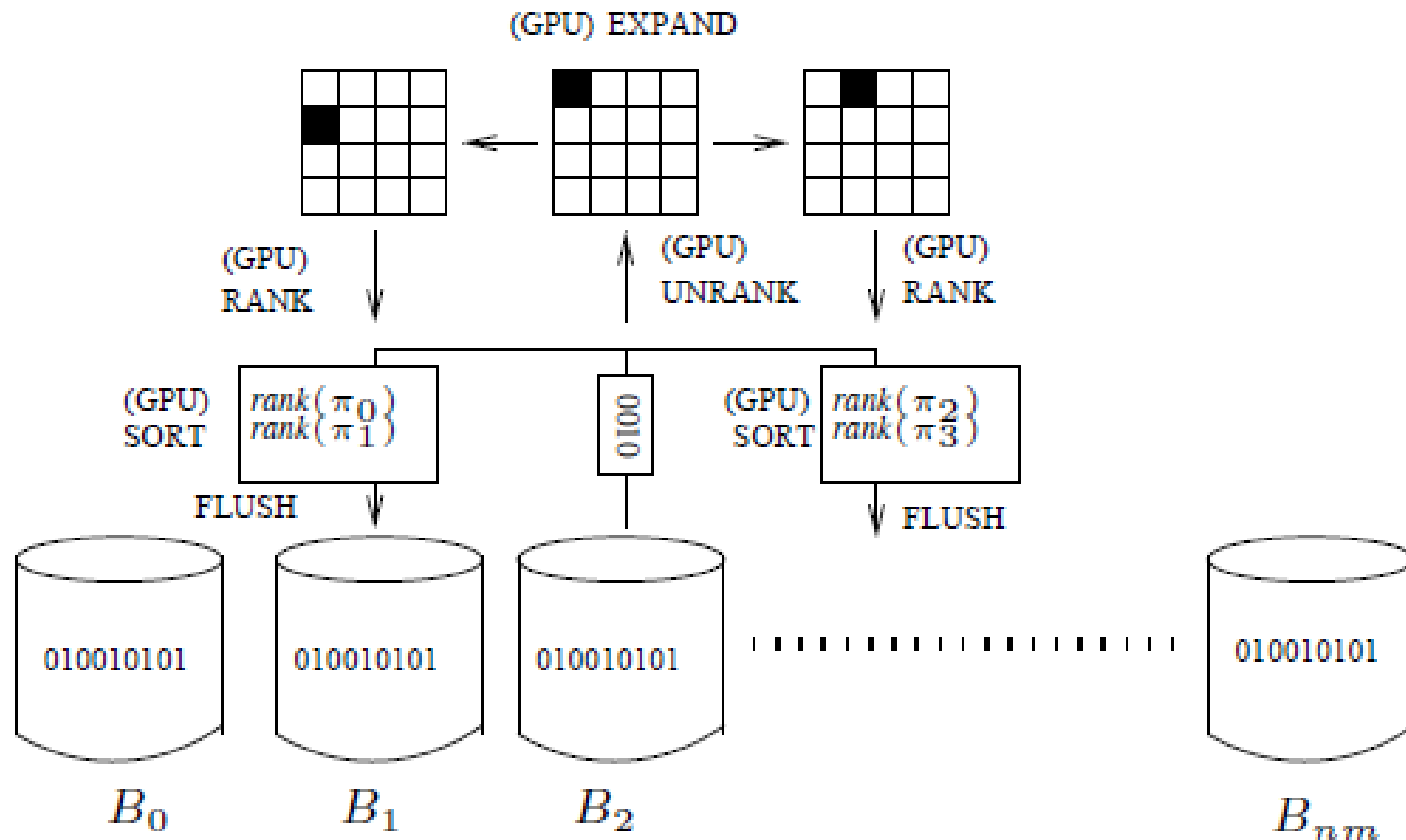
(1,2,3,0)	0	(2,1,3,0)	1
(3,2,0,1)	0	(2,3,0,1)	1
(1,3,0,2)	0	(3,1,0,2)	1
(1,2,0,3)	1	(2,1,0,3)	0
(2,3,1,0)	0	(3,2,1,0)	1
(2,0,3,1)	0	(0,2,3,1)	1
(3,0,1,2)	0	(0,3,1,2)	1
(2,0,1,3)	1	(0,2,1,3)	0
(1,3,2,0)	1	(3,1,2,0)	0
(3,0,2,1)	1	(0,3,2,1)	0
(1,0,3,2)	1	(0,1,3,2)	0
(1,0,2,3)	0	(0,1,2,3)	1

Algorithm *unrank*(r)

```
1:  $\pi := id$ 
2: parity := false
3: while  $n > 0$  do
4:    $i := n - 1$ 
5:    $j := r \bmod n$ 
6:   if  $i \neq j$  then
7:     parity :=  $\neg$ parity
8:     swap( $\pi_i, \pi_j$ )
9:      $r := r \bdiv n$ 
10:   $n := n - 1$ 
11: return (parity,  $\pi$ )
```

Parallelism for the „masses“

- ▶ Current CPUs have 2, 4 or 8 cores
 - ▶ Current (GP) GPUs have 540 cores
- ➔ Huge potential to be exploited



Sliding-Tile (One-Bit)

$n \times m = 6 \times 2$

GPU 149s

CPU 1,110s

$n \times m = 7 \times 2$

GPU 13,590s

CPU o.o.t

Pancake (Two Bit)

$n=12$

States 479,001,600

GPU 290s

CPU 9,287s

Top-Spin (Two-Bit)

$n=12$

States 39,916,800

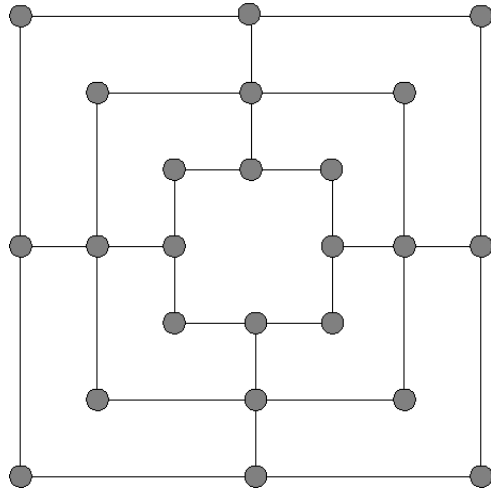
GPU 27s

CPU 920s

- ▶ Motivation Hashing
- ▶ Universal and Perfect Hashing
- ▶ Dynamic Perfect Hashing
- ▶ Perfect Hashing in Permutation Games
- ▶ **Perfect Hashing in Selection Games**
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

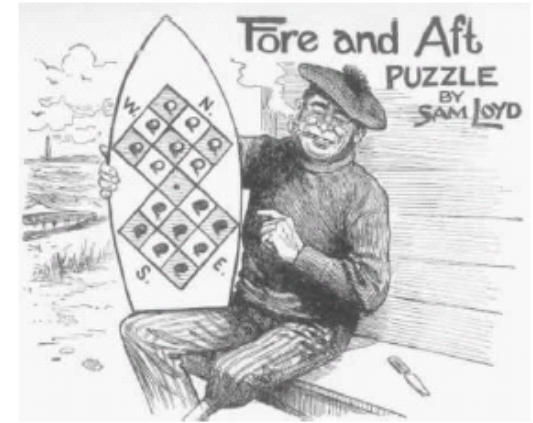
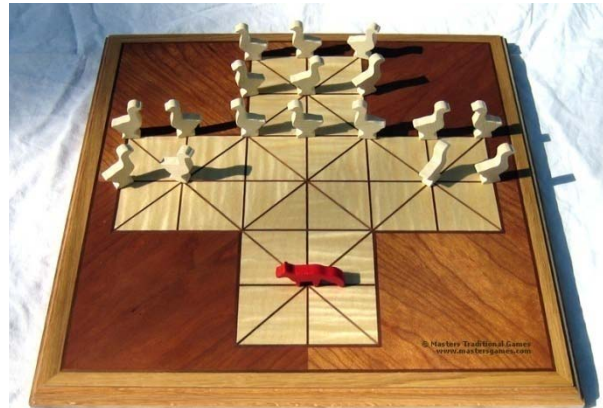
Peg-Solitaire

Nine-
Men-
Morris



Frogs and Toads

Fox and
Geese



Binomial Coefficients

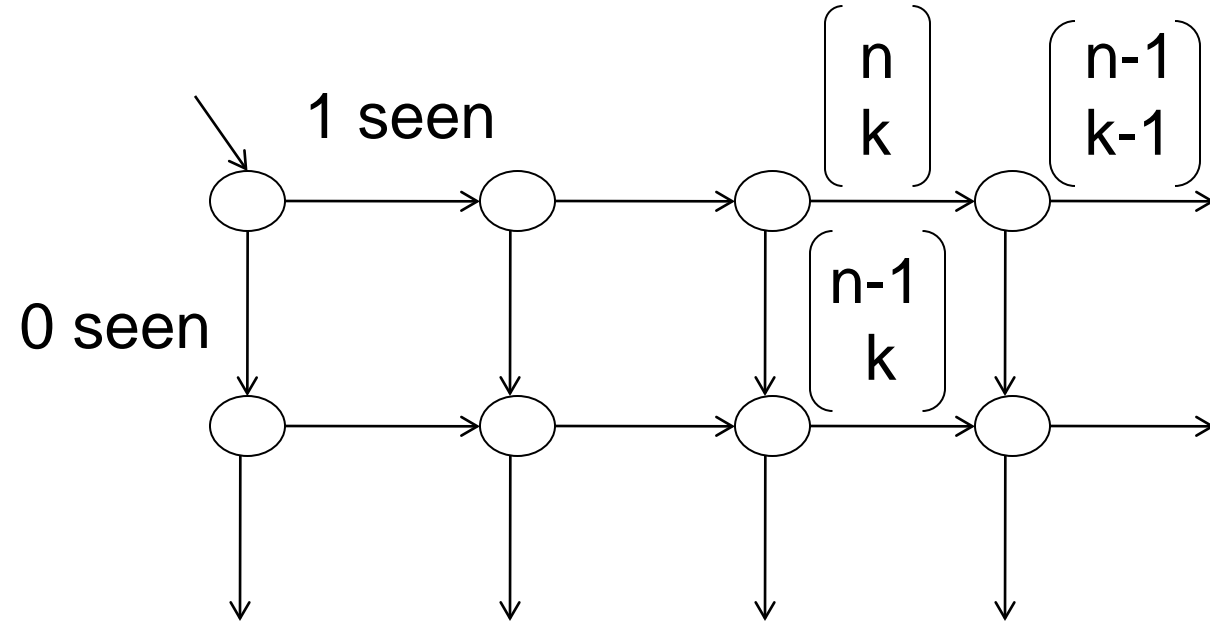
```

11110000
11101000
11100100
11100010
11100001
11011000
...
10000111
01111000
01110100
...
01000111
...
00001111
  
```

Found „0“ →

```

01110000
01100000
...
01000110
01000011
00111000
...
00001111
  
```



Edelkamp, Messerschmidt & Sulewski (09)

Binomial: Peg-Solitaire, Fox and Geese, Frogs & Toads

Binomial Table:

 $O(n+n^2+kn)$ Time $O(n^2)$ Space

Factorial Table:

 $O(n+kn)$ Time $O(n)$ Space**Multinomial:** Nine-Men-Morris

Multinomial Table:

 $O(n+n^3+kn)$ Time $O(n^3)$ Space

Factorial Table:

 $O(n+kn)$ Time $O(n)$ Space

Solved = Solvability Status of all Reachable States
Known

- ▶ **Peg-Solitaire: Solved**, 12m CPU, 1m GPU
- ▶ **Frogs and Toads (4x4): Solved**, 30min GPU
- ▶ **Fox and Geese: Solved** (outcome depending on the number of geese), 1 month on 8 CPU Cores
- ▶ **Nine-Men Morris: Solved** (draw) few days on GPU

- ▶ Motivation Hashing
- ▶ Universal and Perfect Hashing
- ▶ Dynamic Perfect Hashing
- ▶ Perfect Hashing in Permutation Games
- ▶ Perfect Hashing in Selection Games
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

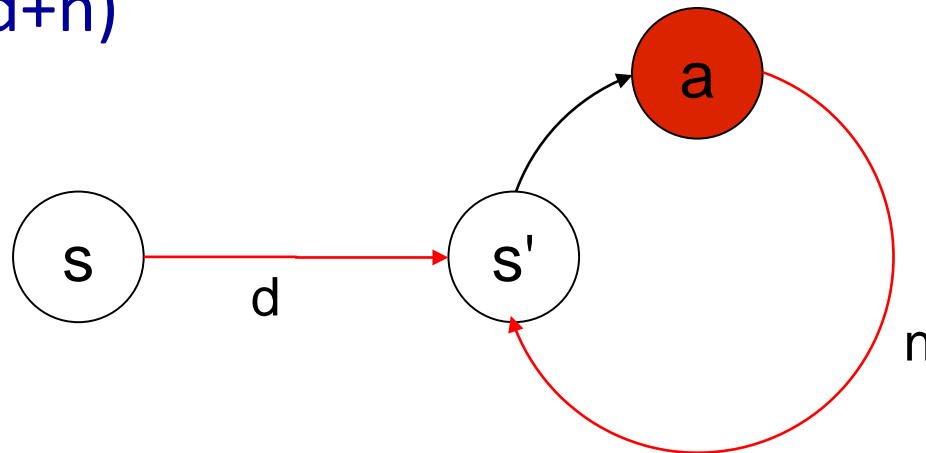
Automata-based (LTL) Model Checking $M \models \phi$:

- ▶ Model M given in some formal spec.
- ▶ Property ϕ to check given in some formal spec.
- ▶ Algorithm detects accepting cycles in (Büchi) Automaton constructed wrt. M and ϕ

Search **Accepting Cycles** / Lasso

Non-Optimal Algorithm: Nested/Double DFS by Courcuoubetis, Vardi, Wolper, Yannakakis (92)

Optimal Algorithm: Search Minimum Accepting Cycles, one with $\min(d+n)$



Model	Number of Vertices	v_{max}	ϵ_s	MPHF Size (bits/vertex)
Elev.2(16),P4	173,916,122	30 bytes	94	4.941
Lamport(5),P4	74,413,141	24 bytes	99	4.941
MCS(5),P4	119,663,657	28 bytes	91	4.941
Peterson(5),P4	284,942,015	32 bytes	177	4.941
Phils(16,1),P3	61,230,206	50 bytes	47	4.941
Ret.(16,8,4),P2	31,087,573	91 bytes	553	4.941
Szyman.(5),P4	419,183,762	32 bytes	223	4.941

Tzi Flash Memory (Solid State Disks)

Expecting falling prices

→ Decreasing discrepancy to RAM (?!)

Increasing Capacity

Low power consumption

→ SSD are replacing HDD (at least in mobile devices)

Faster random read access time than HDD

Same random write access time than HDD

→ Good for static dictionaries (like perfect hash functions)

Gastin and Moro (06)

First stage

- create whole state space with **BFS**
- collect all accepting states

Second stage

- **BFS** at each accepting state checks for a cycle

Third stage

- **BFS** to find the shortest lasso

Edelkamp & Sulewski (08)

Semi-External Flash-Memory MC to generate Minimal-Counter Example

- ▶ Duplicate Detection via External Minimum Perfect Hash Function
- ▶ Minimum Perfect Hash Function stored on SSD

Brim, Edelkamp, Simecek and Sulewski (08)

On-the-Fly Flash-Memory Model Checking

- ▶ Early Duplicate Detection feasible for external memory model checking (CPU usage > 70%)

Experiment	StateSpace	PFH in RAM		PHF on external device		
		RAM	Time	RAM	Time SSD	Time HDD
Szymanski P3-2	1.5 MB	28.7 KB	0:06	2 KB	0:50	0:37
Szymanski P3-3	65 MB	0.99 MB	2:58	68.9 KB	39:02	26:11
Lifts P4 - 7	351 MB	4.5 MB	4:27	0.22 MB	68:56	48:17
Lifts P2 – 8	1559 MB	20.2 MB	19:44	0.99 MB	377:22	o.o.t

- ▶ Motivation Hashing
- ▶ Universal and Perfect Hashing
- ▶ Dynamic Perfect Hashing
- ▶ Perfect Hashing in Permutation Games
- ▶ Perfect Hashing in Selection Games
- ▶ Perfect Hashing for Model Checking
- ▶ Perfect Hashing with BDDs

(Read Once) Binary Decision Diagram

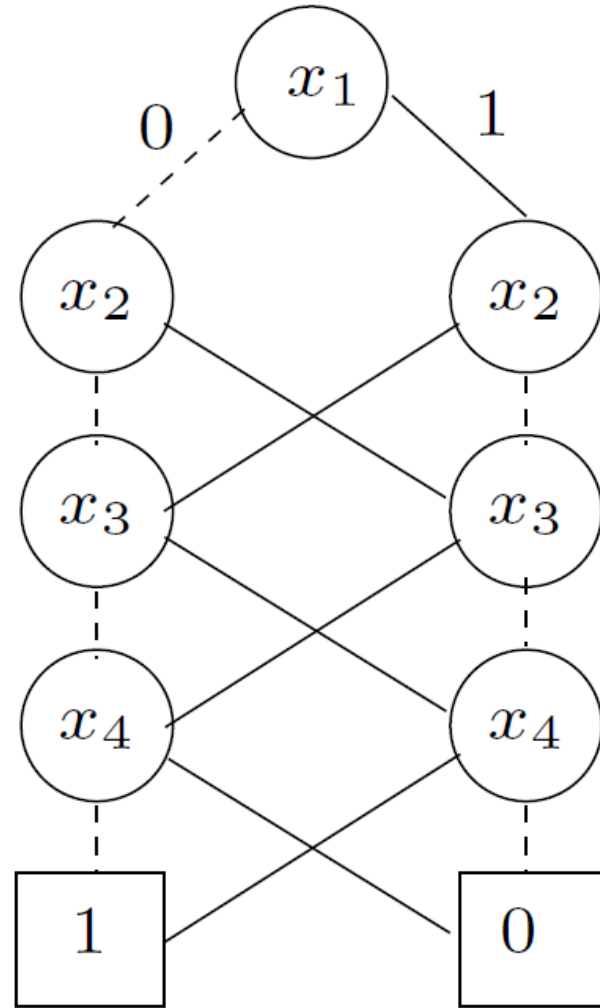
DAG with each variable on every path appearing at most once wrt. fixed variable ordering. Nodes are variables, edges are labeled either 0 or 1, sinks denoted by 0 and 1

→ Unique Representation of Boolean Functions

Quasi-Reduced (RO)BDD

DAG with each variable on every path appearing exactly once [..]

▶ $\text{SatCount}(f) = |\{(a_1, \dots, a_n) \mid f(a_1, \dots, a_n) = 1\}|$



SAT

0000=1

0011=2

0101=3

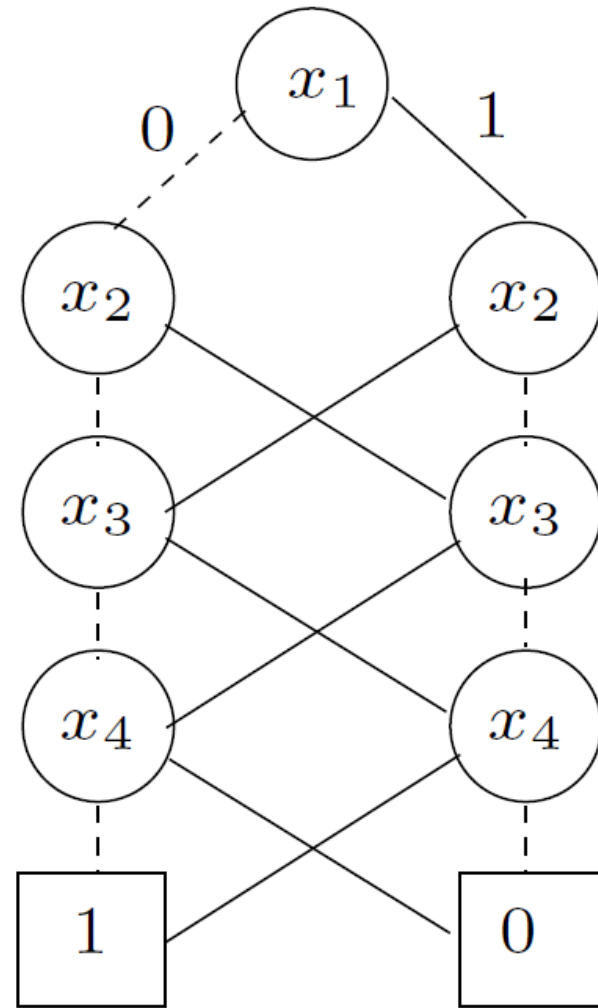
0110=4

1001=5

1010=6

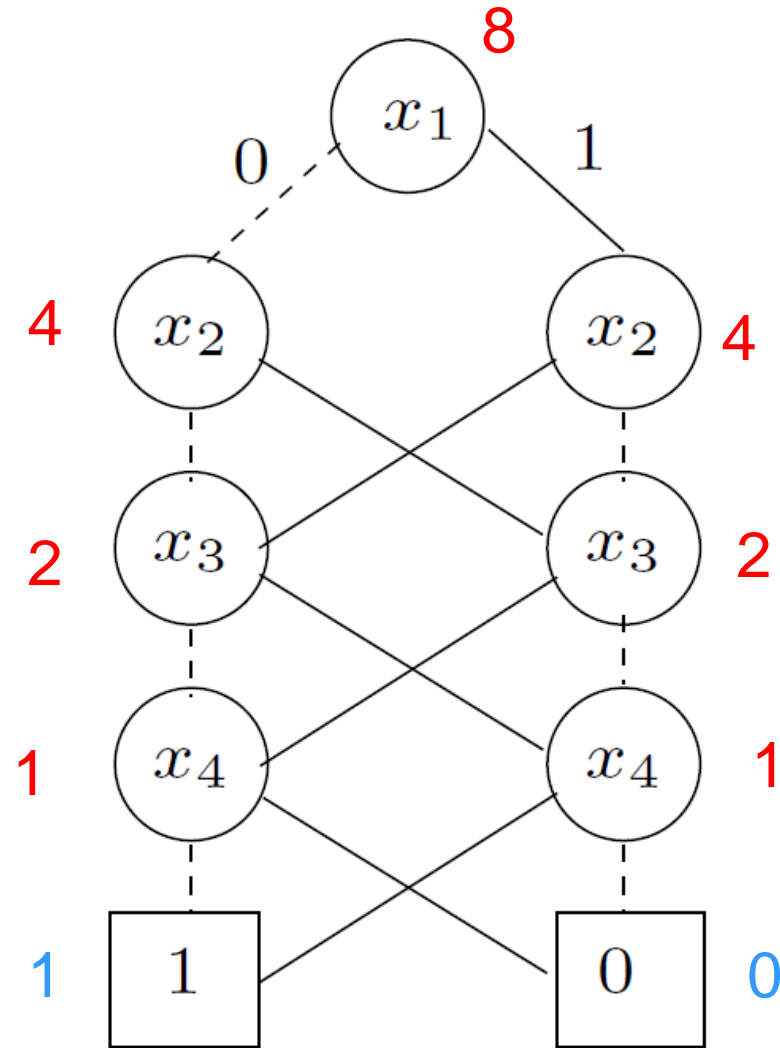
1100=7

1111=8



SATCOUNT

Time $O(|G|)$



Dietzfelbinger & Edelkamp (09)

- ▶ **Perfect Hash Function** (Rank/Unrank) from BDD G for $f(x_1, \dots, x_n)$ to $[1..satcount(f)]$ in
- ▶ $O(|G|)$ Space and Preprocessing Time
- ▶ $O(n)$ Ranking and Unranking Time

lex-rank(G^*, s, v)

if v is 0-sink **return** 0

if v is 1-sink **return** 1

if v is node labeled x_i with 0-succ. u and 1-succ. w

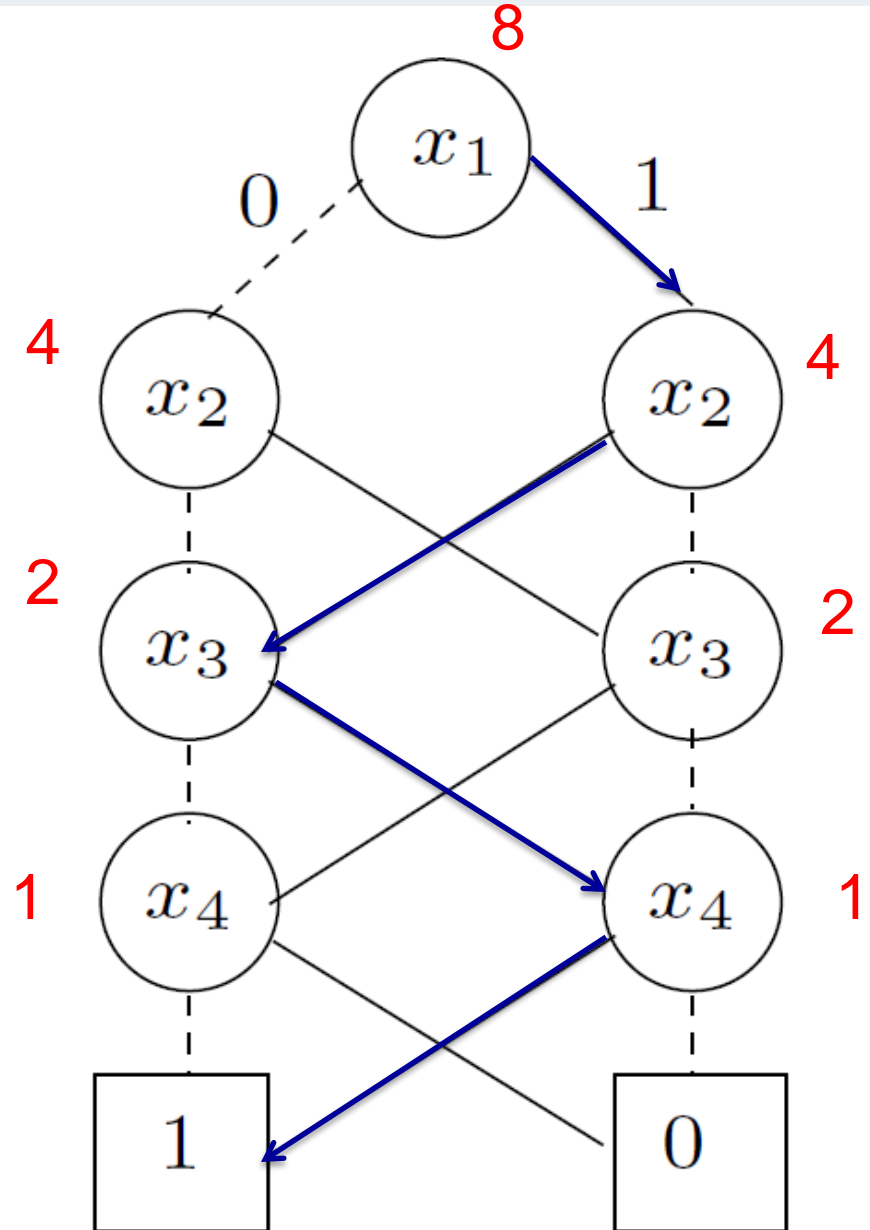
if ($s_i = 1$) **return** **sat-count**(u) + **lex-rank**(G^*, s, w)

if ($s_i = 0$) **return** **lex-rank**(G^*, s, u)

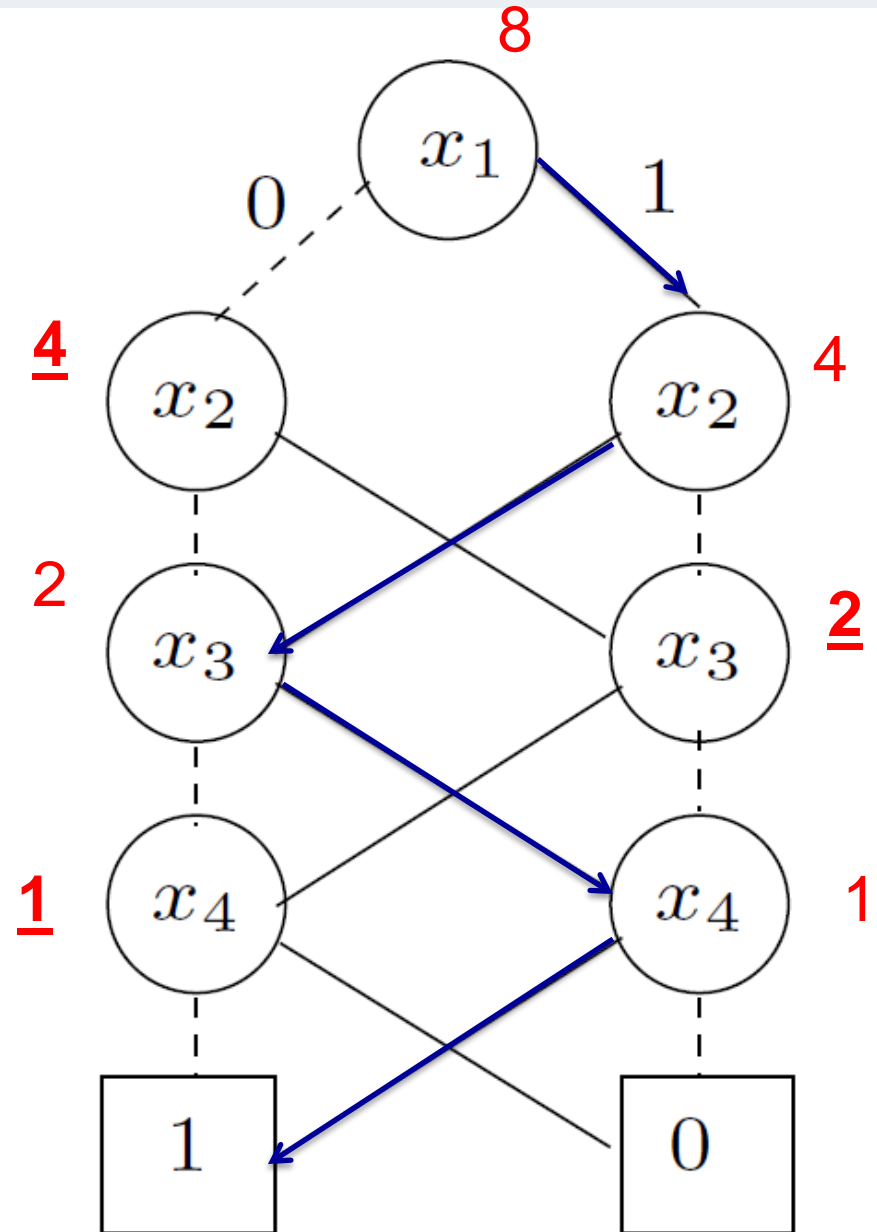
rank(G^*, s)

return **lex-rank**($G^*, s, \text{root of } G^*$)

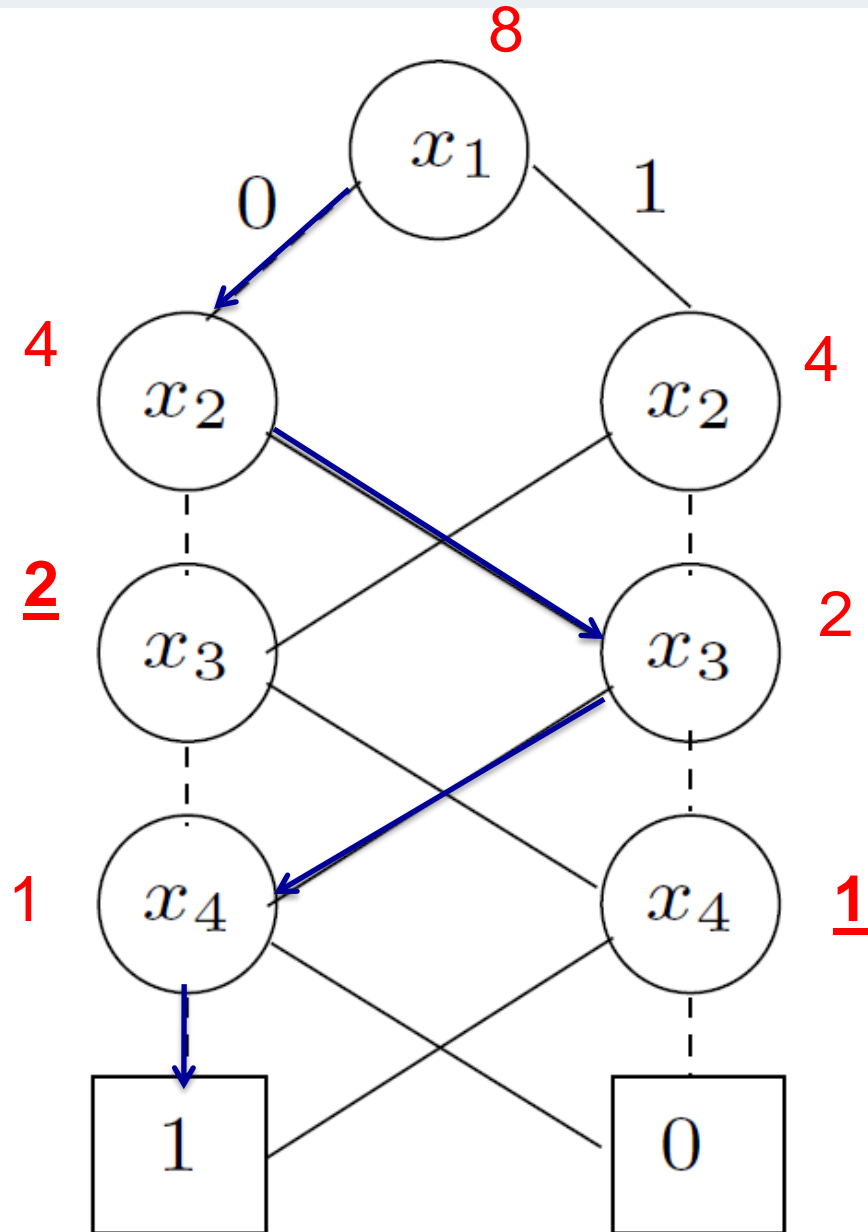
Rank(1111)



$$\text{Rank}(1111) = 4 + 2 + 1 + 1 = 8$$



$$\text{Rank}(0110) = 2 + 1 + 1 = 4$$



unrank(G^*, r)

$i := 1$; start at root of G

while ($i \leq n$)

at node v for x_i with 0-succ. u and 1-succ. w

if ($r > \text{sat-count}(u)$)

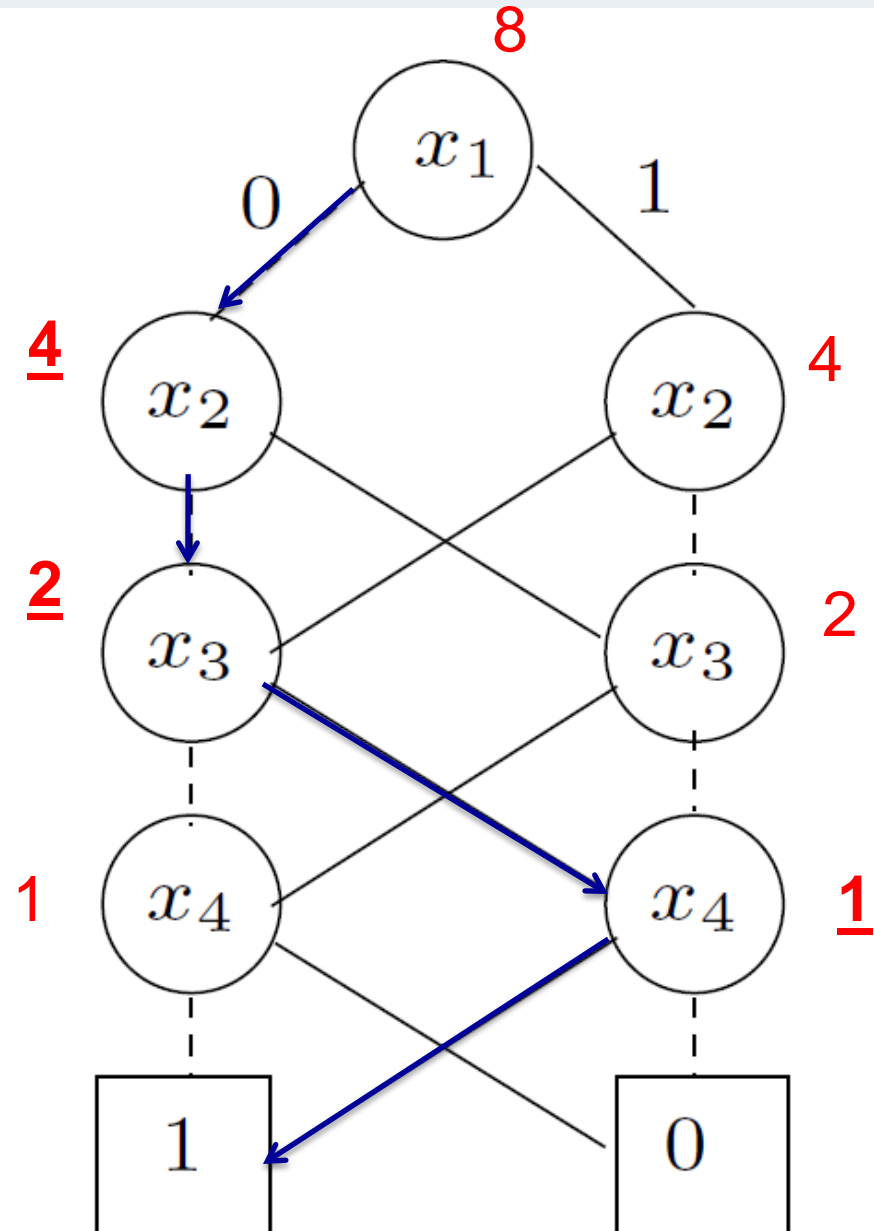
$r := r - \text{sat-count}(u)$

follow 1-edge to w , **record** $s_i := 1$

else follow 0-edge to u , **record** $s_i := 0$

$i := i+1$

Unrank(2)=
 (2>4) ? **No (0)**
 Unrank(1)=
 (2>2) ? **No (0)**
 Unrank(1)=
 (2>1) ? **Yes (1)**
 Unrank(0)=
 (1>0) ? **Yes (1)**



- ▶ **Compression of Data Sets:** Symbolic Equivalent to Explicit (Minimum) Perfect Hash Functions of Botelho et al. (07)
- ▶ **Constant Bitvector Search** based on State Spaces Generated **Symbolically**, e.g. Connect-Four with **4,531,985,218,092** states
- ▶ Uniformly Drawn **Random Satisfying Input** for Boolean Functions

„**HASH**, X. – THERE IS NO DEFINITION
FOR THIS WORD - NOBODY KNOWS
WHAT HASH IS.“

- **AMBROSE BIERCE,**
DEVIL'S DICTIONARY 1906,
- **FOUND IN KNUTH (98), VOL III**

END OF TALK