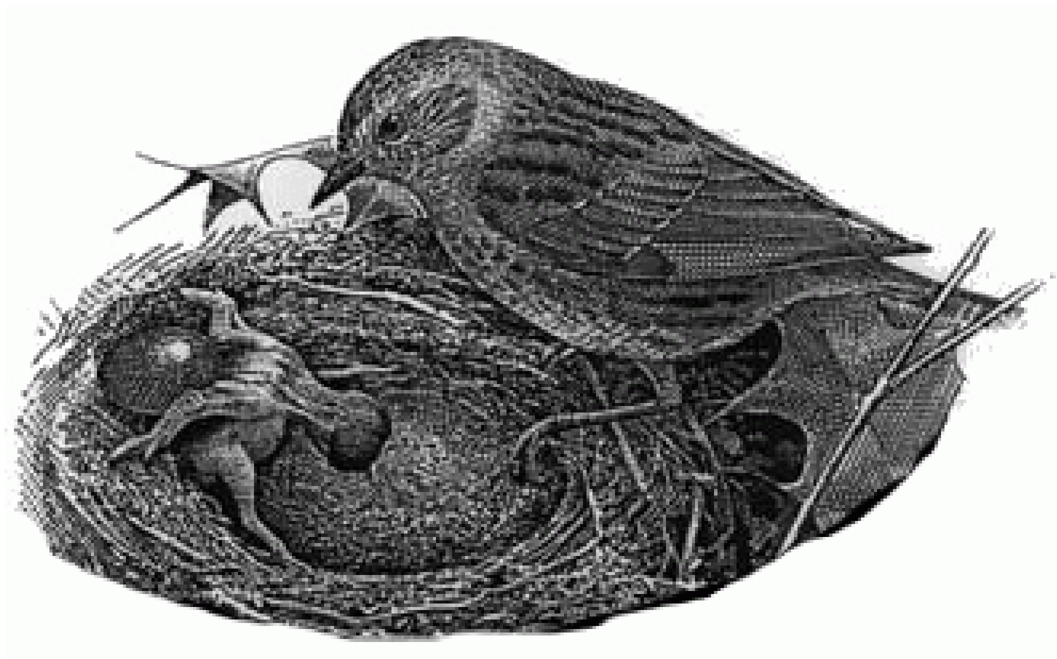
Cuckoo Hashing



Flemming Friche Rodler and Rasmus Pagh (speaking)
BRICS, University of Aarhus, Denmark

ESA, Aarhus, August 28, 2001

The dynamic dictionary problem

Store a set of keys $S \subseteq U$ under the following operations:

- Lookup of keys, and retrieval of satellite information.
- Insertion of a key, with satellite information.
- Deletion of a key.

Complexity expressed in terms of $n = |S|$.

Model of computation:

- Unit cost RAM with word size w and a standard instruction set.
- $U = \{0, 1\}^w$.

Space constraint: Use $O(n)$ words.

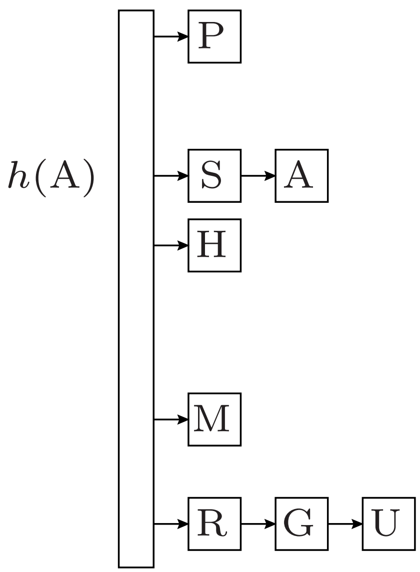
“Classic” hashing schemes

Idea (Dumey ’56 and others):

One or more “random” functions determine where to look for keys.

Collision resolution schemes:

- Separate chaining
- Coalesced chaining
- Linear probing
- Double hashing
- Uniform probing
- ...



Expected *constant* time for all operations. Simple. Widely used.

Modern developments

[Carter & Wegman '77]:

Universal hash functions suffice for hashing with separate chaining.

Theory and practice meet!

[Fredman et al. '82], [Dietzfelbinger et al. '88]:

Worst case constant time lookups, expected constant time updates.

Theory and practice don't quite meet:

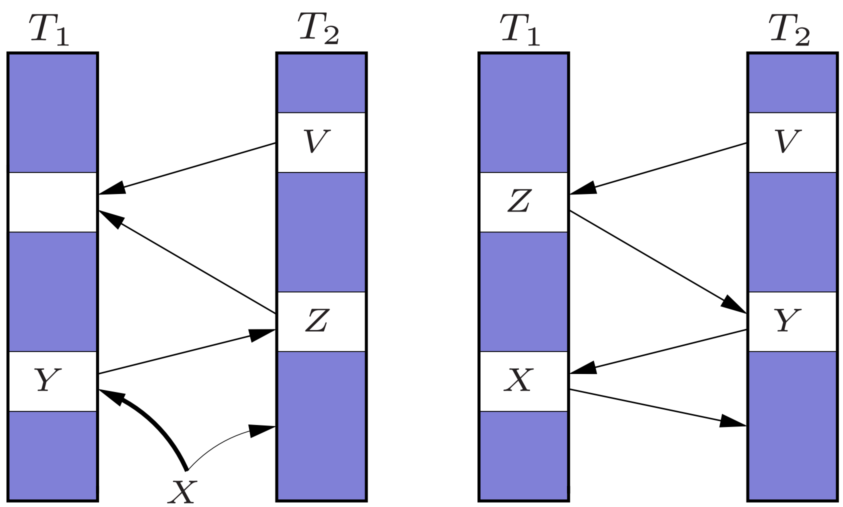
Not so simple. High space usage. Mostly slower than classic schemes. (Efficiency can be improved at the cost of complicating the scheme.)

This talk: Cuckoo Hashing.

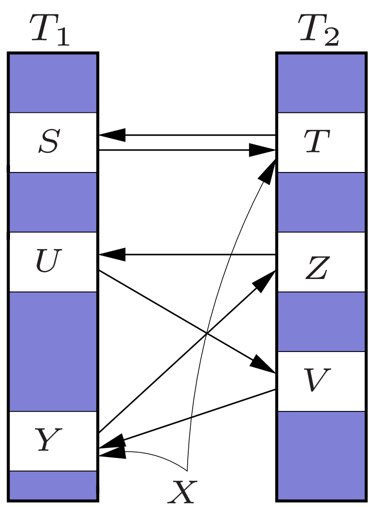
A simple and practical scheme with worst case constant lookup time.

Algorithmic idea

- Use tables T_1, T_2 and hash functions h_1, h_2.
- Store x in one of $T_1[h_1(x)]$ and $T_2[h_2(x)]$.
- Insert(x):
  - Greedily insert in table.
  - Repeat in other table with the previous occupant, if any.



Successful insertion



Rehash needed

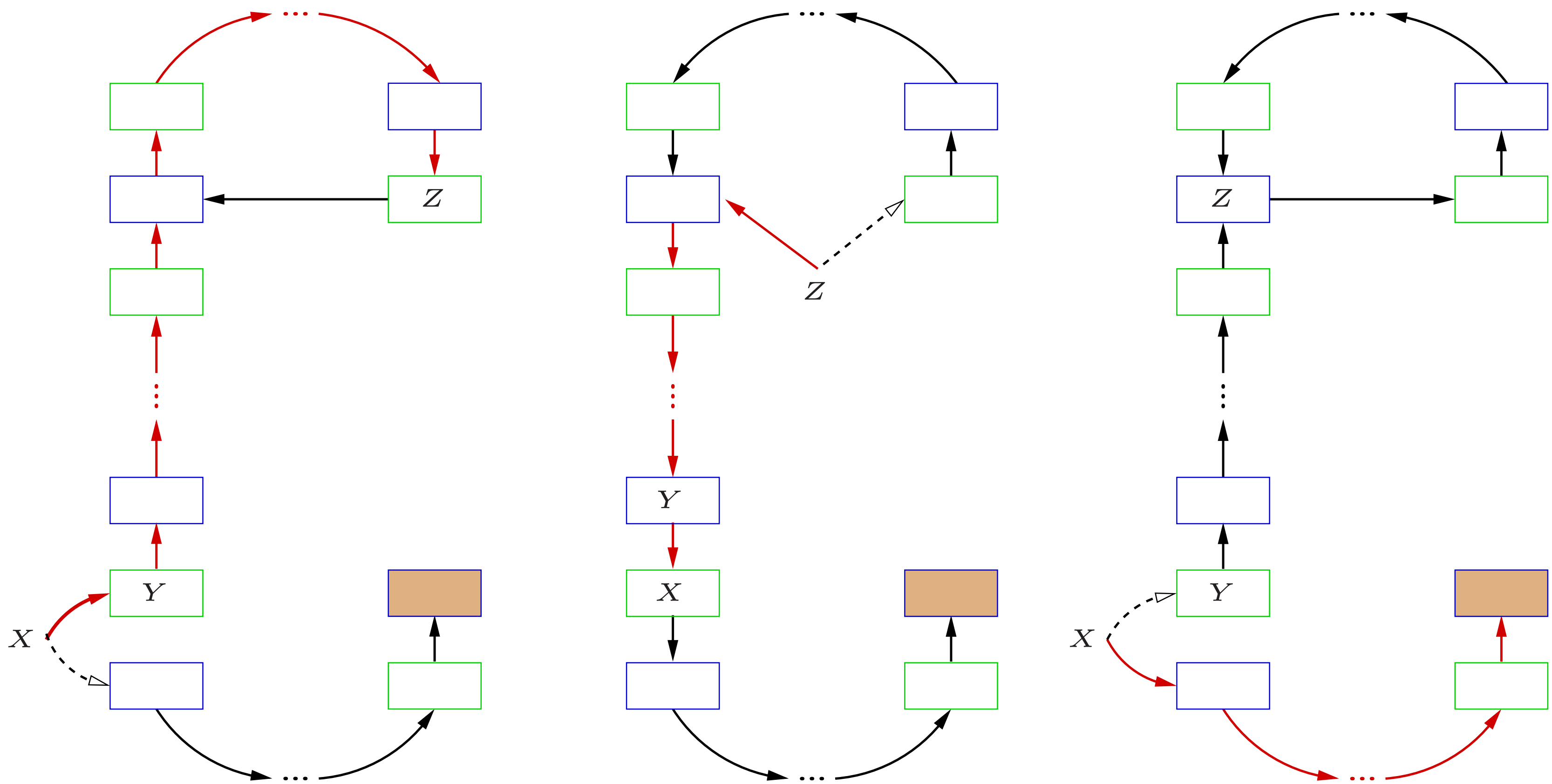
Insertion procedure

procedure insert(x)
  if lookup(x) **then return**
  loop MaxLoop **times**
    if $T_1[h_1(x)] = \perp$ **then** $\{$ $T_1[h_1(x)] \leftarrow x$; **return** $\}$
    $x \leftrightarrow T_1[h_1(x)]$
    if $T_2[h_2(x)] = \perp$ **then** $\{$ $T_2[h_2(x)] \leftarrow x$; **return** $\}$
    $x \leftrightarrow T_2[h_2(x)]$
  end loop
  rehash(); insert(x)
end

(Assumes size of each hash table is bounded away from n.)

Stages of an insertion

Sequence of pushes through T_1 and T_2:



Analysis

Cases:

- Brown cell was seen before — insertion impossible.
  Probability $O(1/n)$.
- Path has length $> 2\,\text{MaxLoop} = \Theta(\log(n))$.
  Probability $o(1/n)$.
- Brown cell is empty — successful insertion.
  Expected length of path $O(1)$.

Randomness assumption:

- $O(\log n)$-wise independence.
- $O(1)$ time evaluation possible [Siegel '89].
  Unfortunately not practical ...

Experimental results

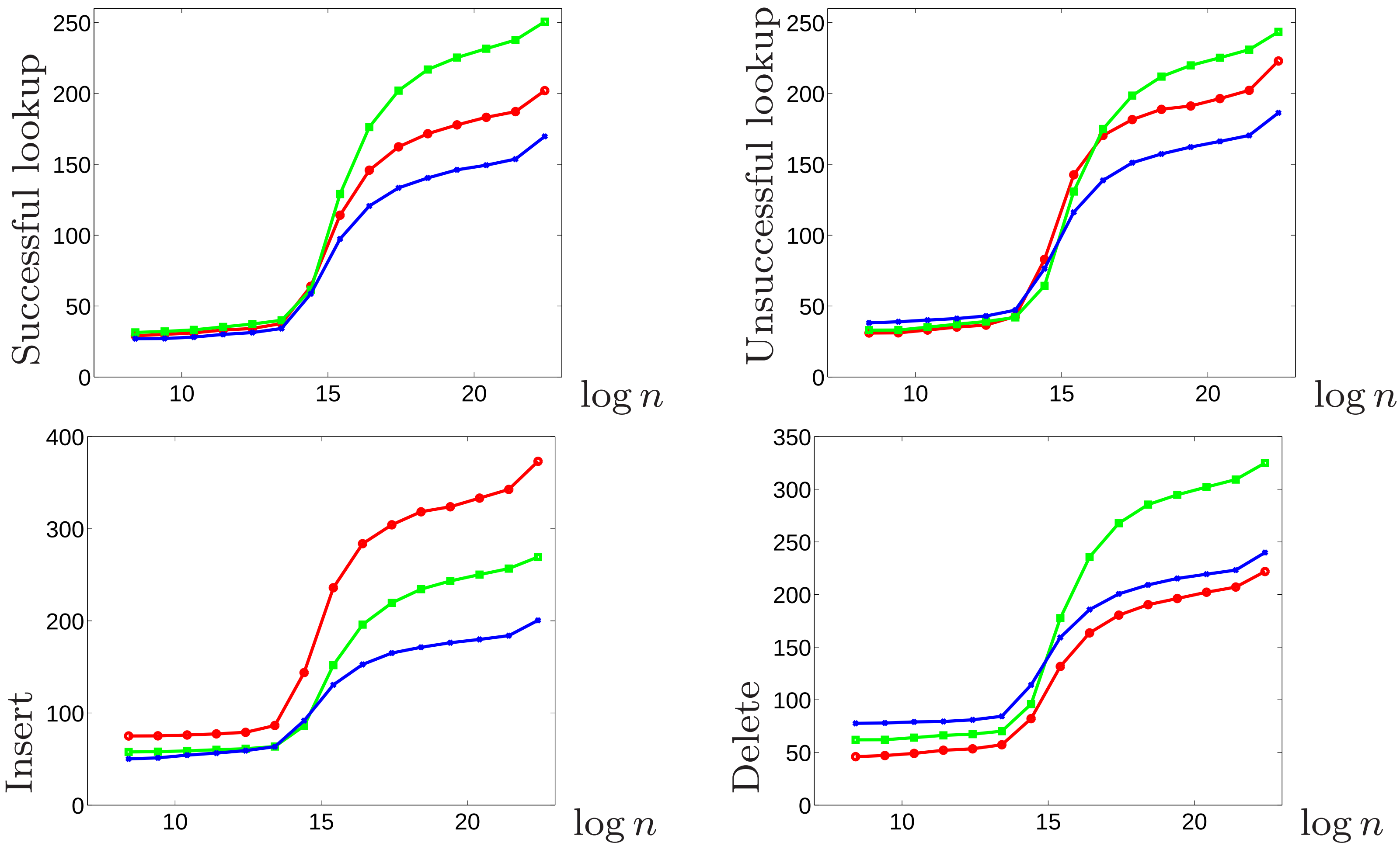
Experimenting with the scheme using weaker hash functions ...

DIMACS test	Joyce		Eddington	
Linear P.	42 - 45	[.35]	26 - 27	[.40]
Chained H.	49 - 52	[.31]	36 - 38	[.28]
A.Cuckoo	47 - 50	[.33]	37 - 39	[.32]

DIMACS test	3.11-Q-1		Smalltalk-2		3.2-Y-1	
Linear P.	99 - 103	[.30]	68 - 72	[.29]	85 - 88	[.32]
Chained H.	113 - 121	[.30]	78 - 82	[.29]	90 - 93	[.31]
A.Cuckoo	166 - 168	[.29]	87 - 95	[.29]	95 - 96	[.32]

Clock cycles per operation [load factor].

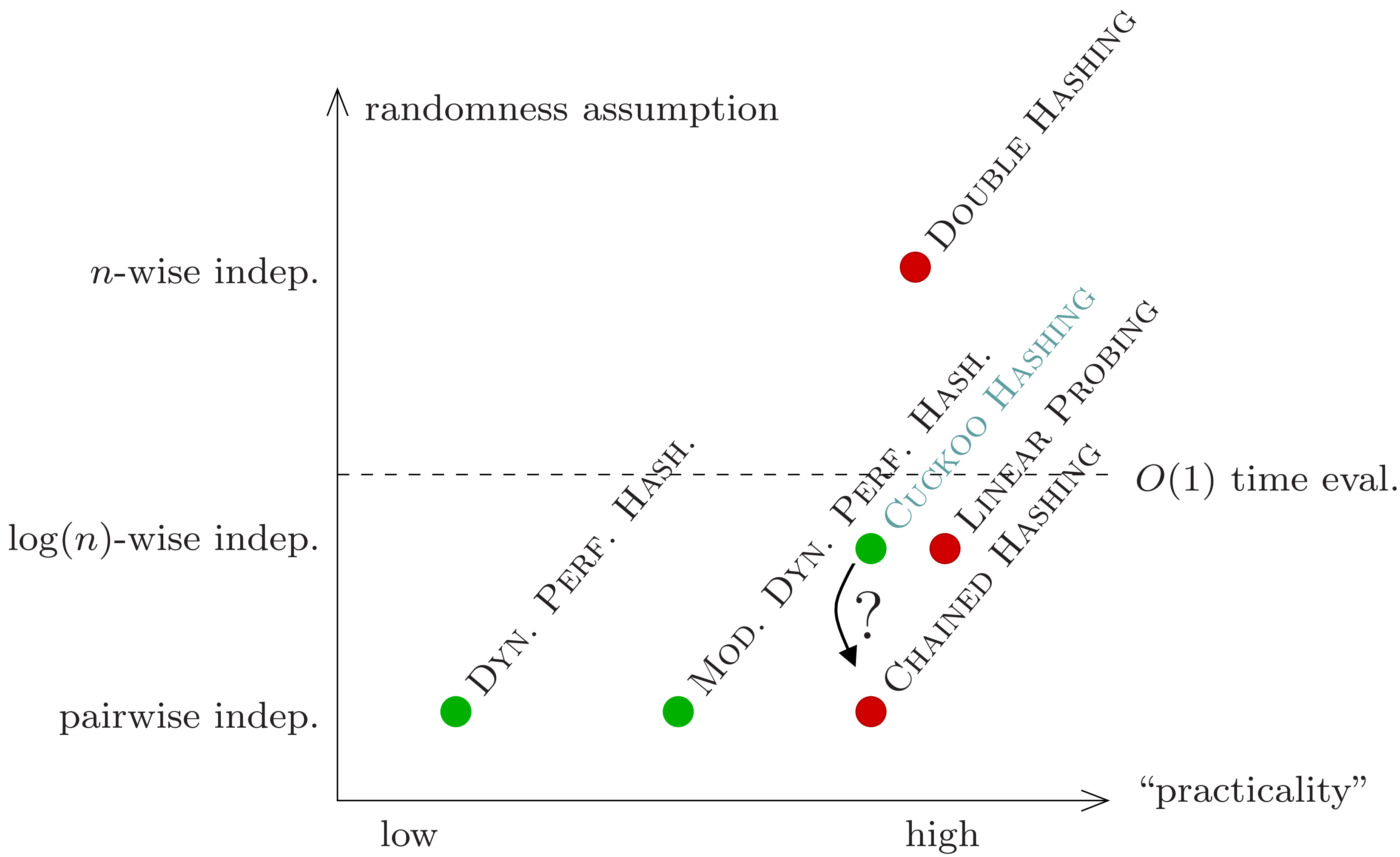
Random input tests



Average time per operation in "equilibrium" for load factor $1/3$.
Cuckoo Hashing, Chained Hashing *and* Linear Probing.

Overview and open problem

Schemes with worst case and average case constant lookup time.



Conclusion

Cuckoo Hashing properties:

+ Simple implementation.

+ Lookups using two probes (optimal).

+ Efficient in the average case.

÷ A practical, provably good hash function family is not known.

Ideas for further work:

- Find a practical, provably good hash function family ...
- A more precise analysis of insertion behavior.