PhD defense

October 11, 2002

Hashing, randomness and dictionaries

Rasmus Pagh

Overview of presentation

1. Introduction to searching a computer's memory using ”hashing”.
2. New ideas behind some of the results in the thesis.
3. Overview of results in the thesis.

PART I

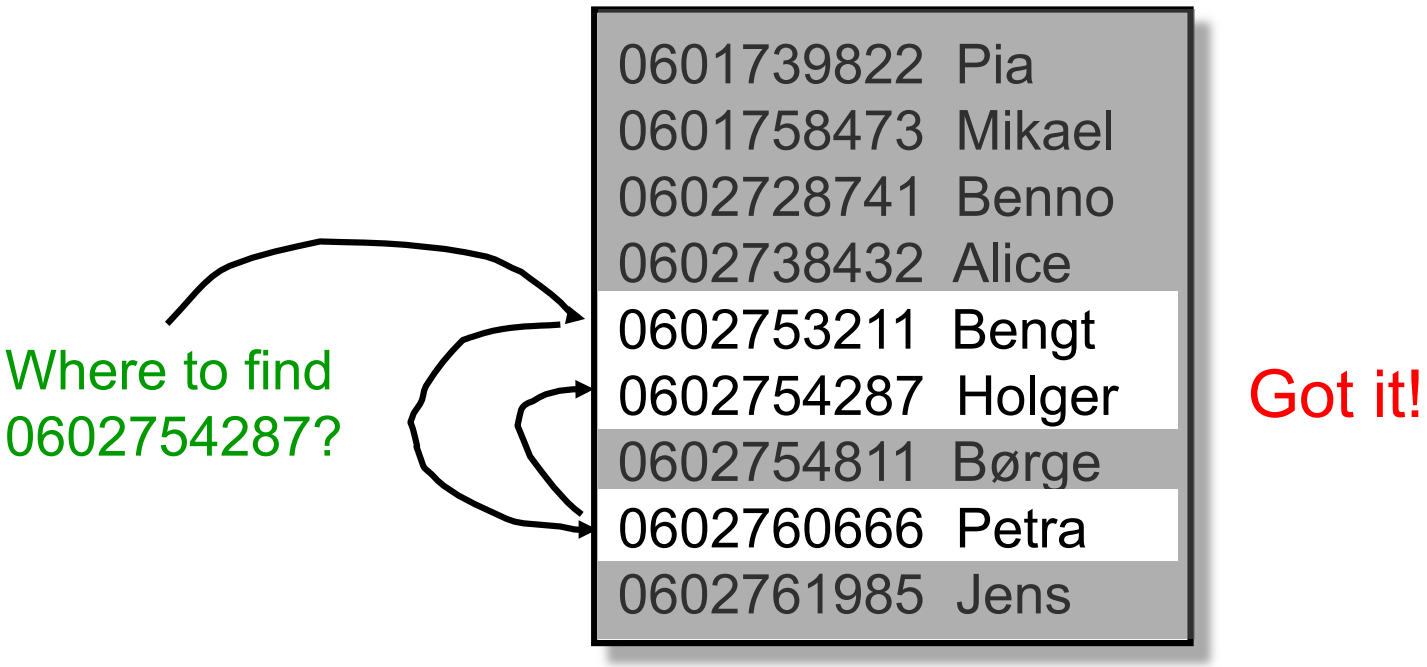
Introduction to searching a computer's memory using "hashing".

Searching a computer's memory

- A basic task in computer science is to *search* for a certain piece of information (a ”key”) in the memory of a computer.
- This is often called the *dictionary* problem.
- For example, search for information related to ”060275-4287”.

The sorting approach

- Keep a *sorted* list of all information.



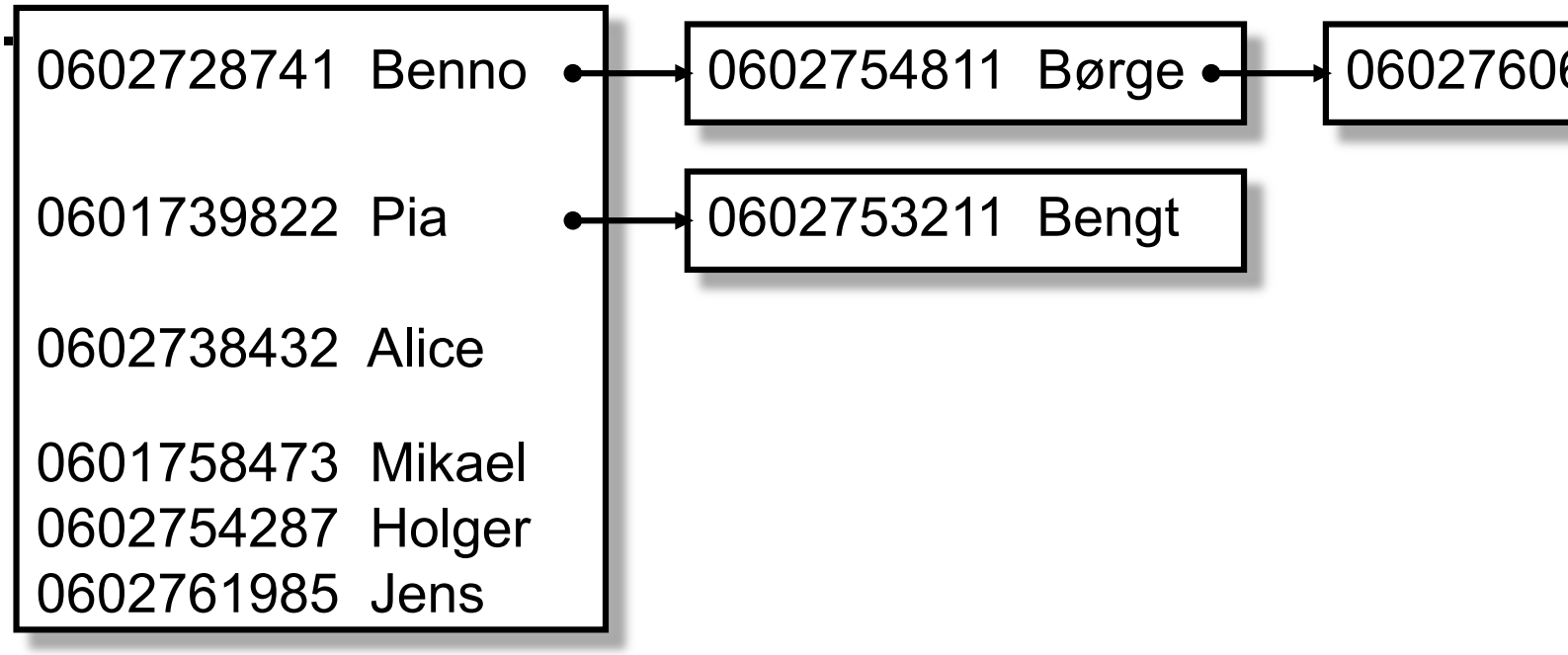
- Number of search steps *increases* when the amount of information grows.

The need for speed

- Many applications perform millions, billions, or even trillions of searches.
- Users do not want to wait for answers.
- The amount of data is rapidly increasing – solutions that remain fast for large data set are needed.

The hashing approach

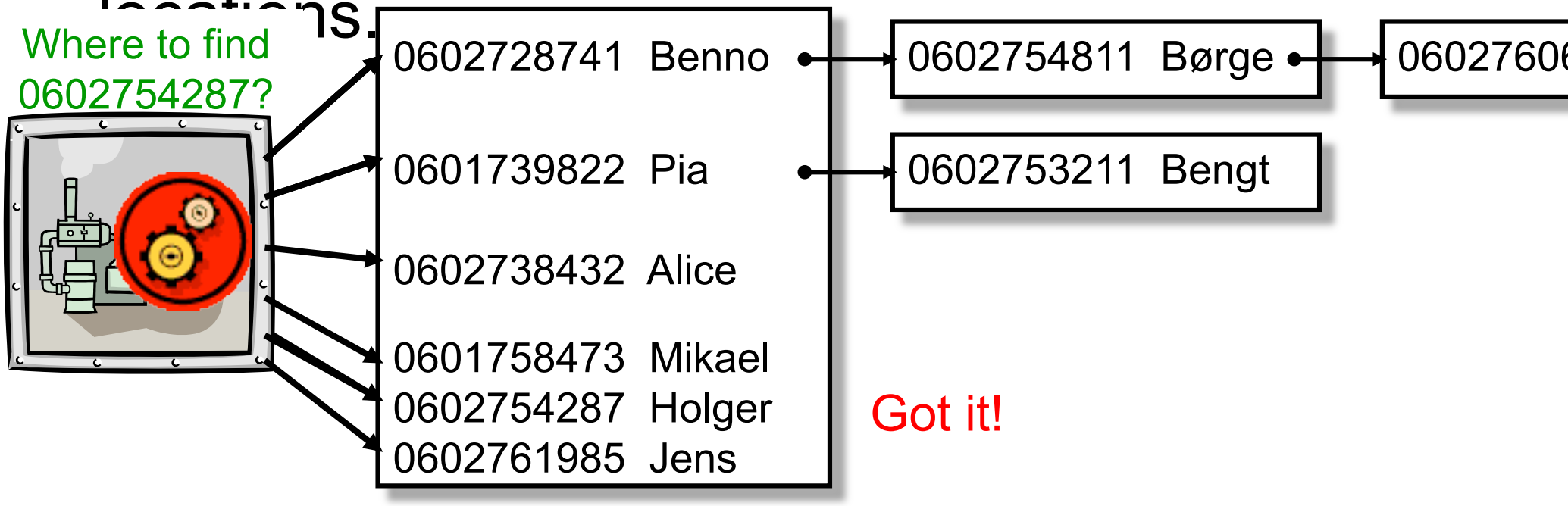
- Idea: Store information in *random* locations.



- Use a ”hash function” to generate and remember random locations.

The hashing approach

- Idea: Store information in *random* locations.



- Use a ”hash function” to generate and remember random locations.

Hashing in the real world

- Search time varies, but on average it is great – no matter how much information!
- Udi Manber (chief scientist, YAHOO!®): ”The most important techniques behind Yahoo! are: hashing, hashing, and hashing”.
- Lots of other critical applications in databases, search engines, algorithms, etc.

Google™ ORACLE CISCO SYSTEMS EMPOWERING THE INTERNET GENERATION™ CEDA

PART II

New ideas behind some of the results in the thesis.

A PROBLEM

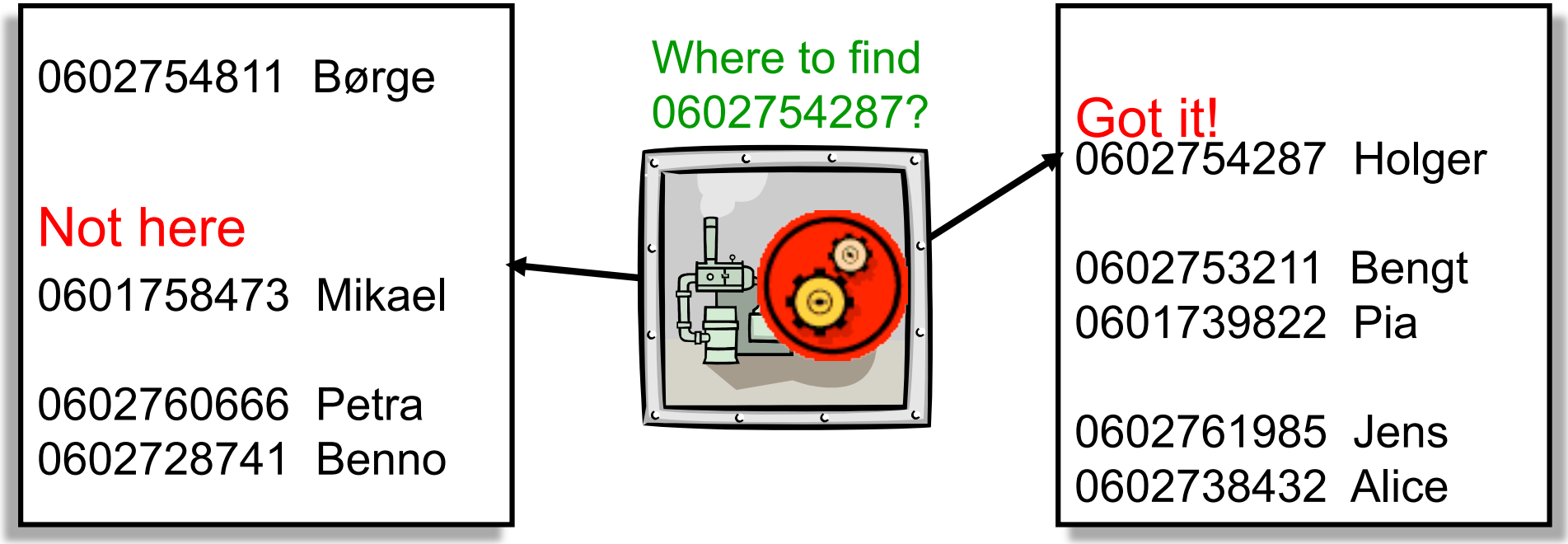
Search time guarantee

- The time for searching is often more critical than the time for updating.
- Sometimes *worst case* bounds are important.
- Pioneering work on dictionaries with worst case search time by Fredman et al. (1982) and Dietzfelbinger et al. (1988).

A NEW SOLUTION

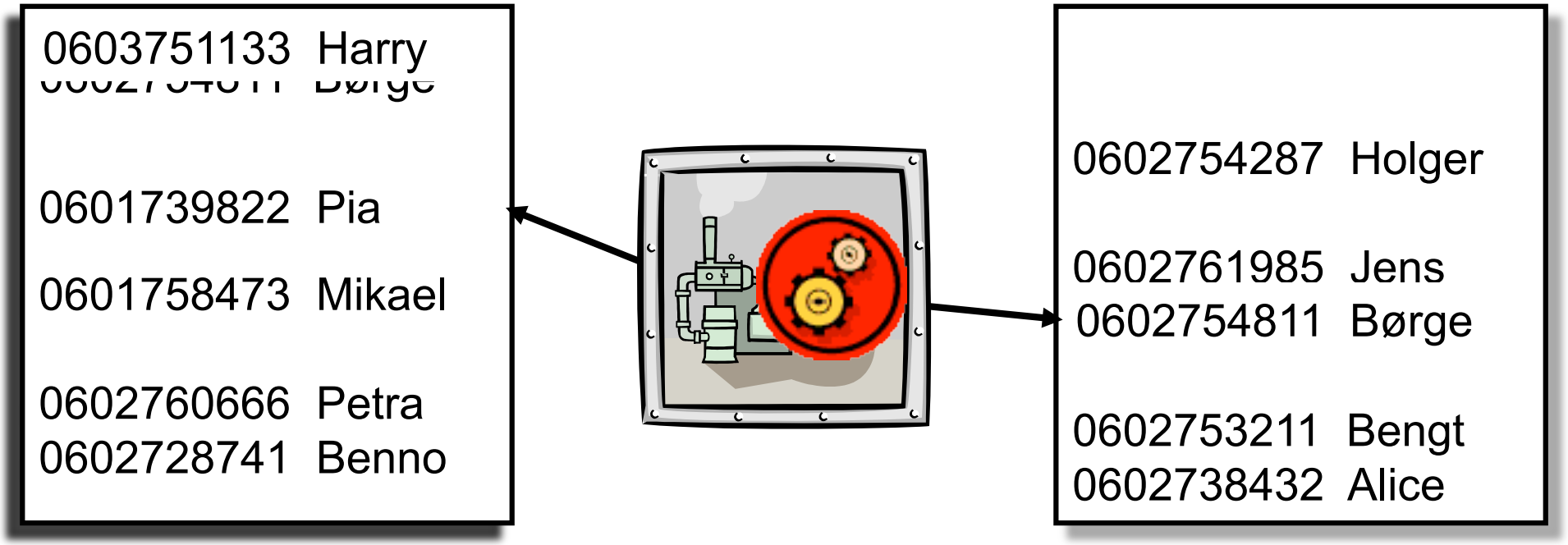
Cuckoo hashing

- Idea: The hash function provides *two* possible locations.



Cuckoo insertion

- New information is inserted by, if necessary, kicking out old information.

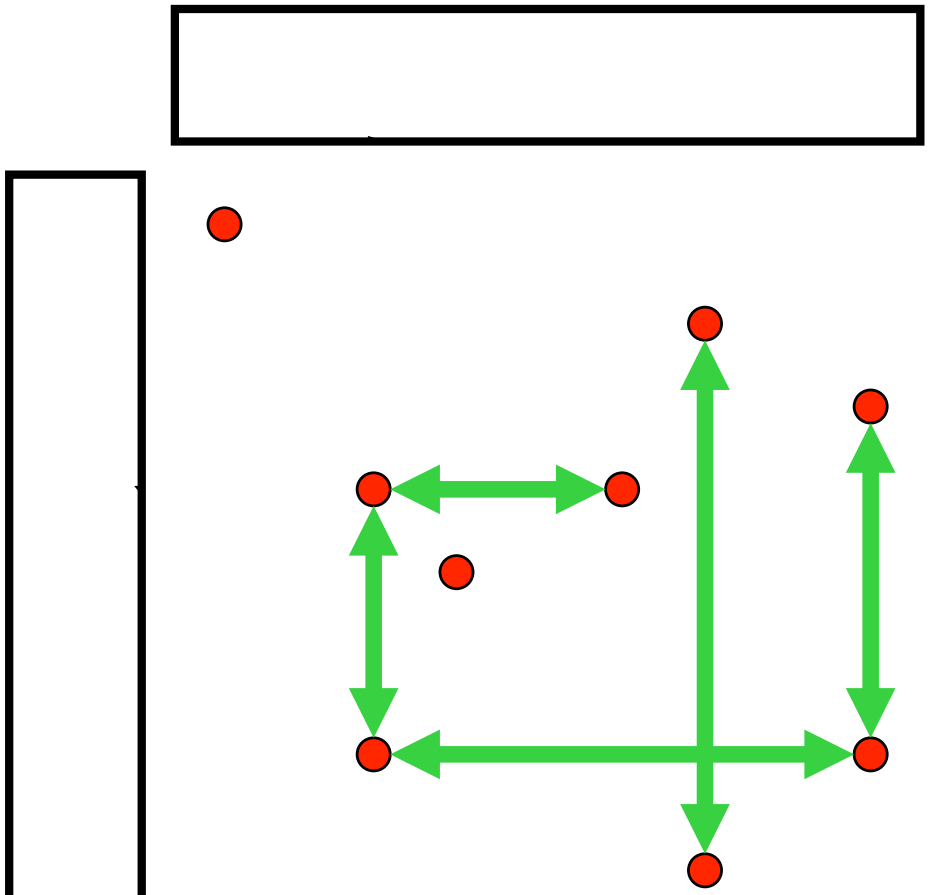


A PROBLEM

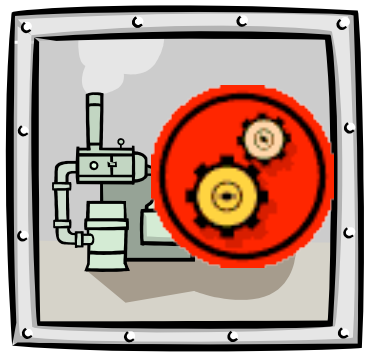
Perfect hashing

- ”Perfect hashing” is hashing without collisions.
- Some memory to store such a function is necessary.
- Function description may be stored in fast, expensive memory and the table in cheaper and slower memory.

Another view of cuckoo hashing



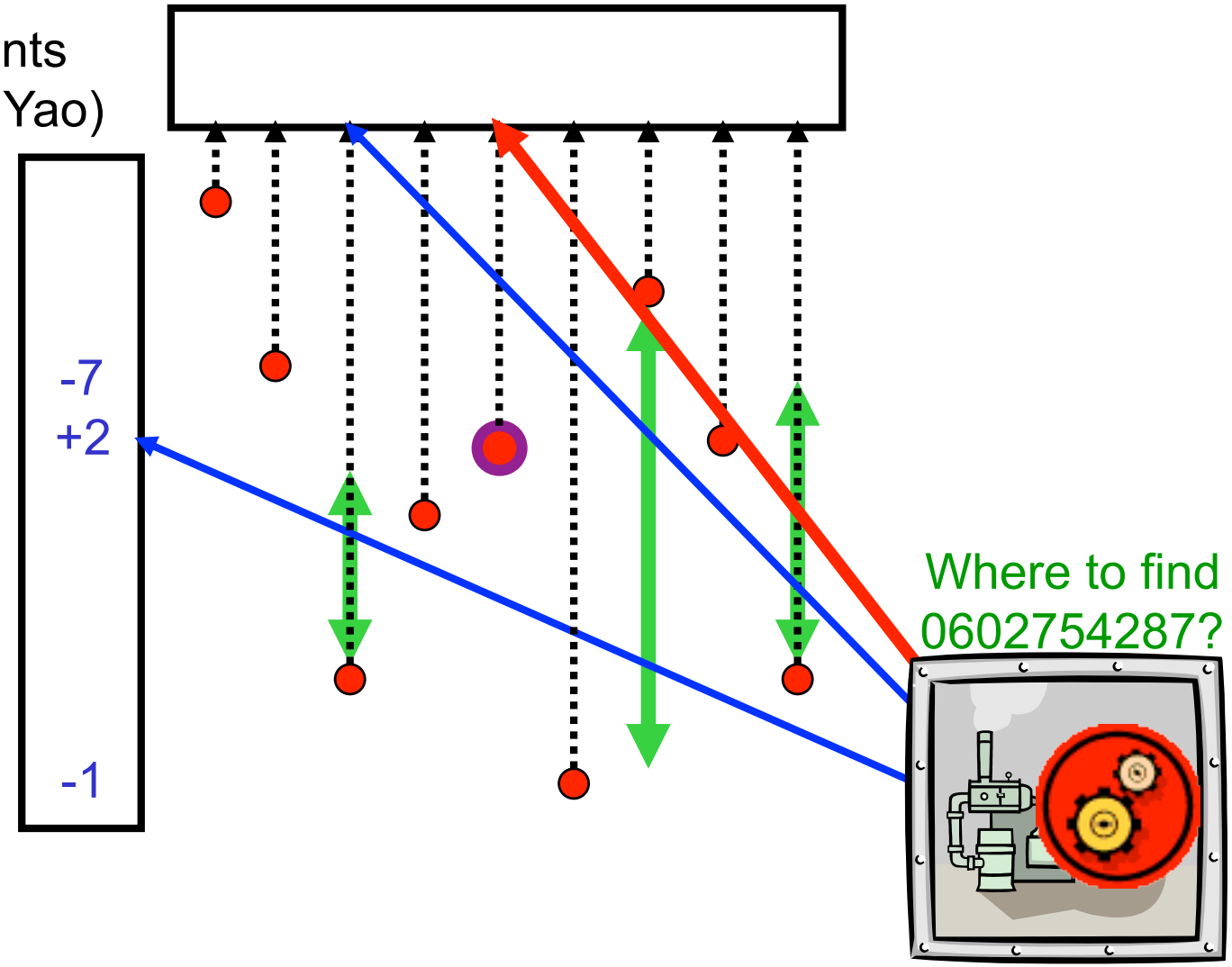
0602754287



A NEW SOLUTION

Hash and displace

Idea:
Displacements
(Tarjan and Yao)



A PROBLEM

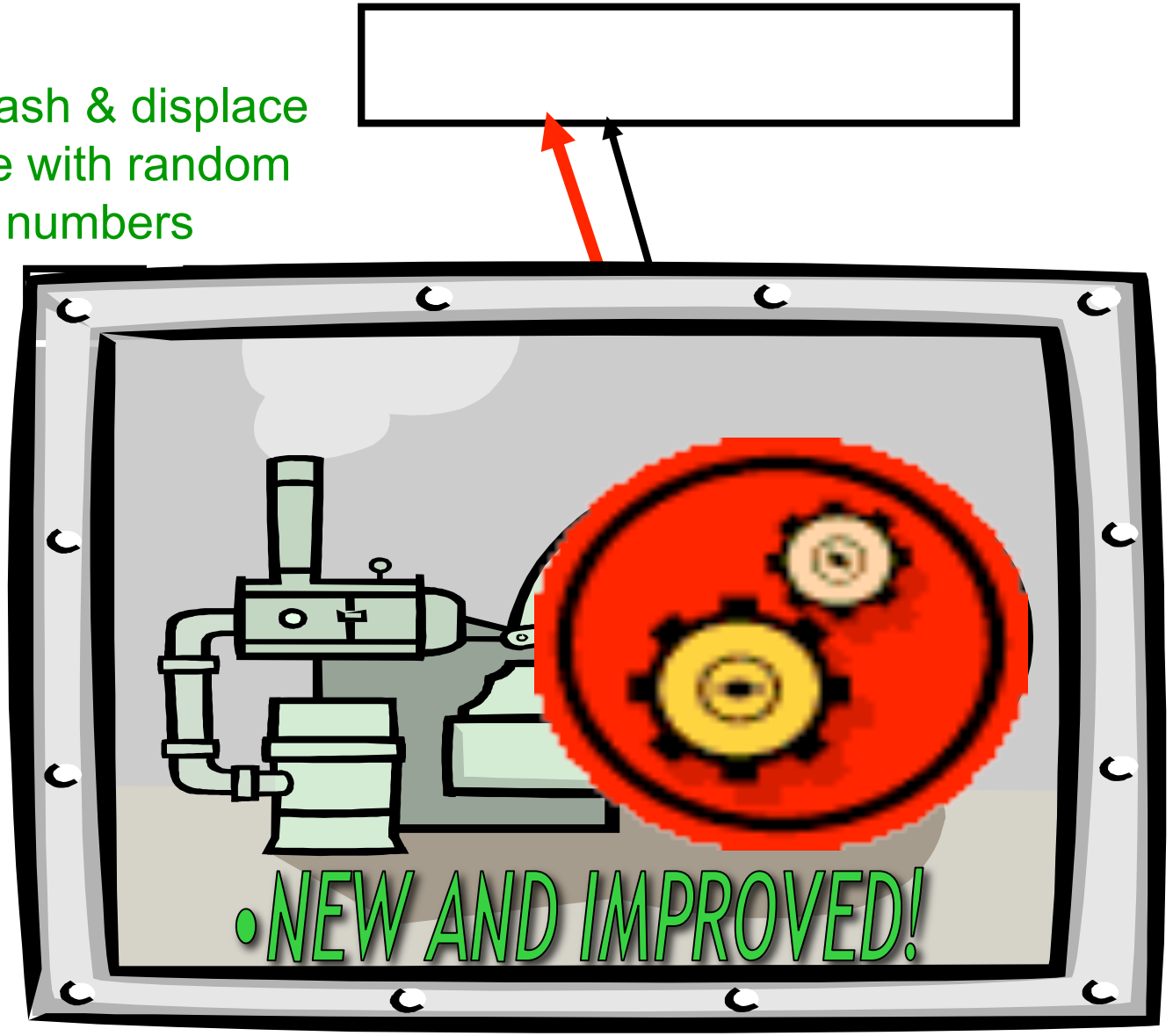
”Simulating” random functions

- Quest for practical constructions of hash functions that ”behave sufficiently similar to random functions” initiated by Carter and Wegman in 1977.
- Many constructions suggested, and shown to work well in certain situations.

A NEW SOLUTION

Uniform hashing

Fill hash & displace table with random numbers



PART III

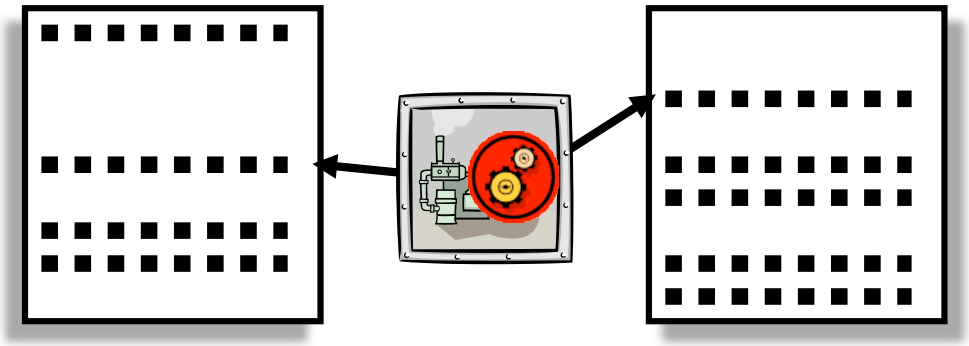
Overview of results in the thesis.

Contributions of the thesis

Mainly *theoretical* results of *mathematical* nature, but also some results that may be practically useful.

- New simple and efficient hashing schemes (cuckoo hashing, hash & displace).
- New kinds of hash functions (uniform hashing, dispersing hash functions), with applications.
- More efficient deterministic hashing algorithms (static and dynamic).
- Detailed investigation of hashing and dictionaries in the ”cell probe” model (upper/lower bounds).
- A dictionary using almost minimal space.

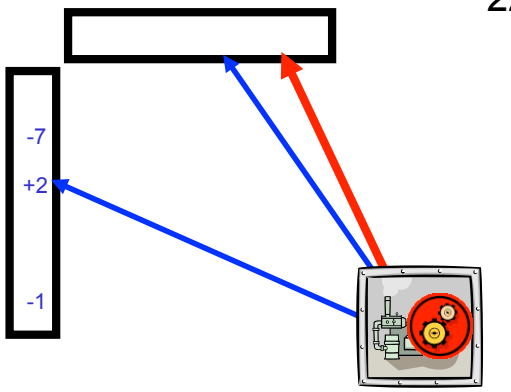
Cuckoo hashing



- Utilizes nearly half of the hash tables.
- Searching uses two memory lookups.
- Insertion takes expected constant time.
- … when using powerful hash functions.
- However, very efficient in practice using weaker hash functions.
- Considerably simpler than other hashing schemes with worst case search time.

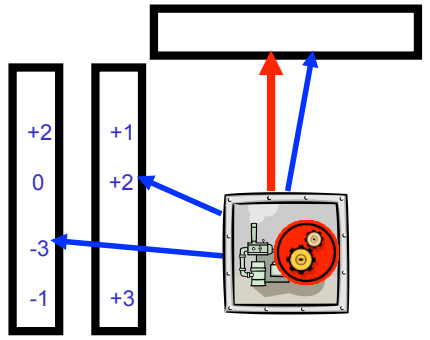
Joint work with Flemming Friche Rodler

Hash and displace



- For a set of *n* keys, the analysis needs a table of $(2+\varepsilon)n$ integers.
- Table with suitable displacements can be computed in expected O(n) time.
- The table containing information has size *n*, i.e., it is completely full.
- ”Universal” hash functions suffice – fast.
- Perhaps the most practical such scheme that is theoretically understood.

Uniform hashing



- When hashing a set of n keys, one needs tables of O(n) integers.
- With probability 1-O(n^{-c}) the new hash function computes *independent* and *uniform* values on the set.
- … when based on powerful hash functions.
- Previous results required O($n^{1+\varepsilon}$) integers.
- Gives theoretical justification for the widespread ”uniform hashing” assumption.

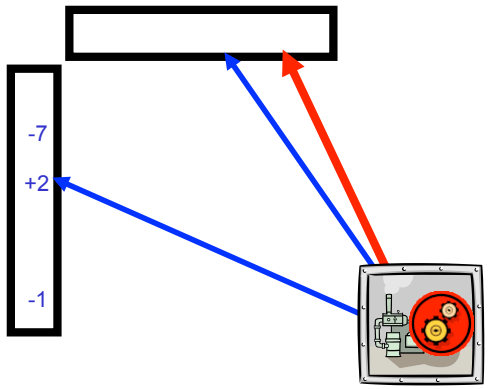
Joint work with Anna Östlin

Dispersing hash functions

- Goal: Small use of random bits when hashing.
- Uniform hashing uses O(n log n) random bits.
- Universal hashing uses O(log n + log log u) random bits.
- *Dispersing* hash functions, introduced in the thesis, may use only O(log log u) random bits.
- Suffice for, e.g., relational join and element distinctness in expected linear time.
- No explicit construction – shown to be as hard as finding good explicit extractors.

Deterministic dictionary construction

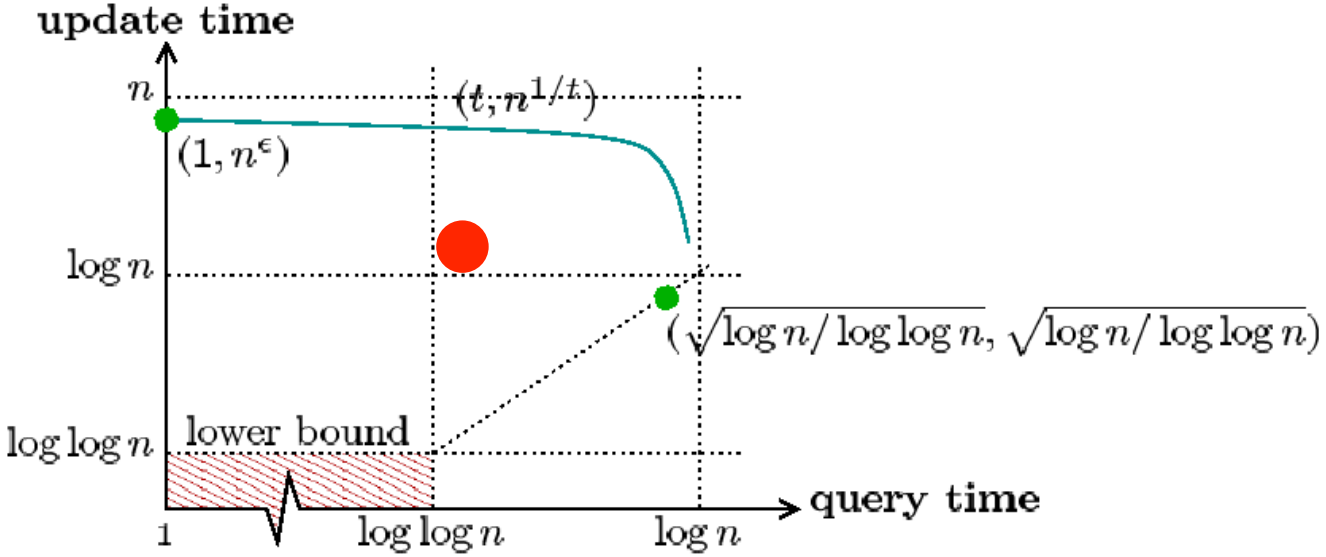
- What if one allows *no* random bits?
- We want O(1) query time and linear space.
- Miltersen (’98) + Hagerup (’99): Reduce in time O(n log n) the general problem to that in a universe of size n^2. Yields time O($n^{1+\varepsilon}$).
- **New**: An O(n log n) algorithm for computing ”good” displacements for the Tarjan-Yao scheme, which handles universe size n^2.



Joint paper with Torben Hagerup and Peter Bro Miltersen

Deterministic dynamic dictionary

- Known deterministic dynamic dictionaries exhibit a *trade-off* between update and query time.



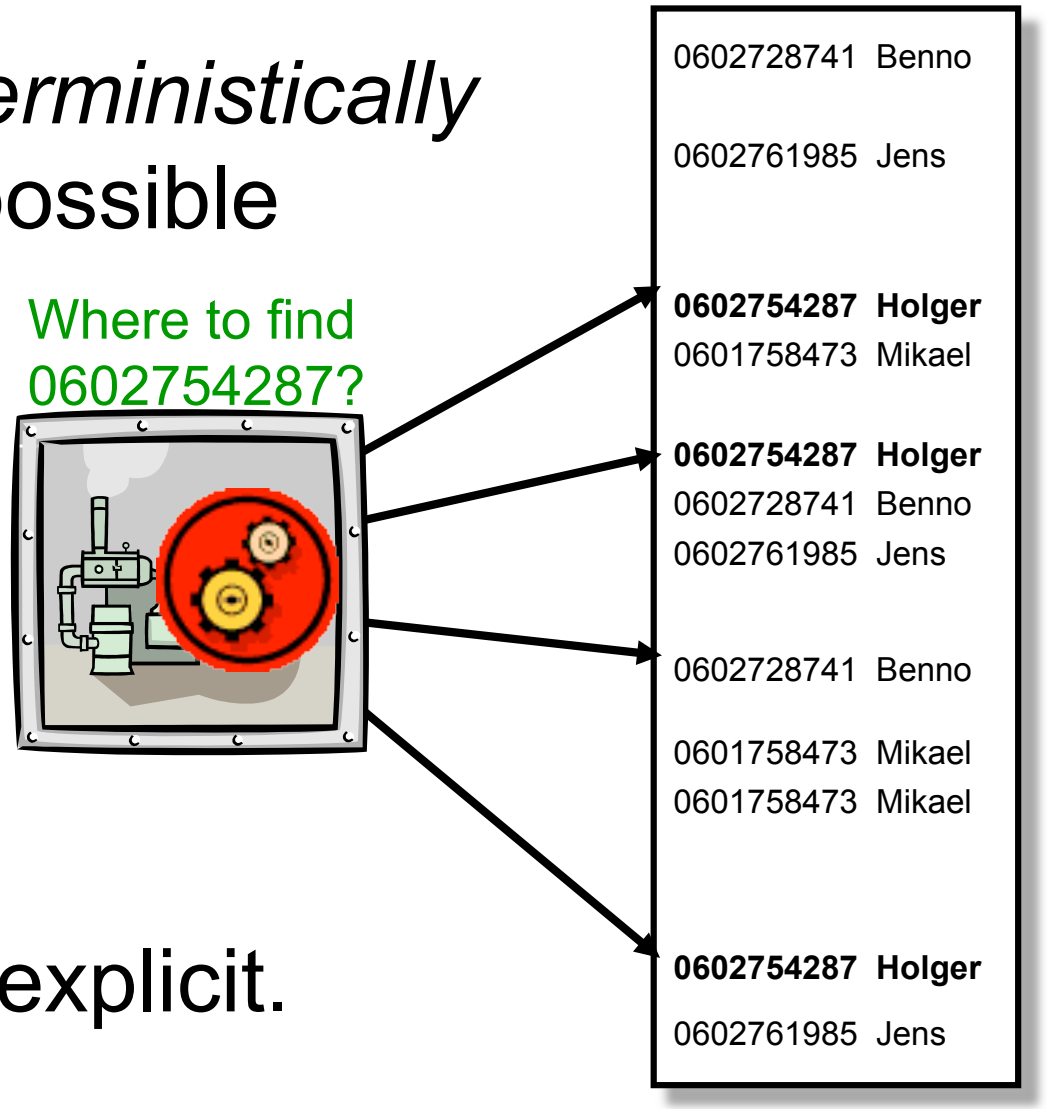
- New trade-off added using the mentioned techniques of Miltersen and Hagerup.

Impossibility results

- There are limits to the efficiency of algorithms.
- Useful to know if this limit has been reached.
- Some results from the thesis:
  - 1 adaptive memory probe (as in hash & displace) is optimal for perfect hashing.
  - 2 random access memory probes (as in cuckoo hashing) is worst-case optimal.

One-probe search

- ”Hash function” *deterministically* produces O(log *u*) possible locations.
- Probing a random location finds the information with probability 1-ε.
- Table size *n* log *u*.
- ”Hash function” not explicit.



Joint work with Anna Östlin

A succinct dictionary

- Simple information theory provides a lower bound **B** on the space usage of a dictionary (say, with no information besides keys).
- **New:** A static dictionary with O(1) lookup time using **B**+o(n)+O(log log u) bits.
- Improves the lower-order term compared to previous results.

Chronology of papers

- **Low Redundancy in Static Dictionaries with Constant Query Time**
  ICALP 1999 and *SIAM Journal on Computing, 2001*
- **Hash and Displace: Efficient Evaluation of Minimal Perfect Hash Functions**
  WADS 1999
- **Deterministic Dictionaries**
  With Torben Hagerup and Peter Bro Miltersen
  SODA 1998-2000 and *Journal of Algorithms, 2001*
- **A Trade-Off for Worst-Case Efficient Dictionaries**
  SWAT 2000 and *Nordic Journal of Computing, 2000*
- **Dispersing Hash Functions**
  RANDOM 2000
- **On the Cell Probe Complexity of Membership and Perfect Hashing**
  STOC 2001
- **Cuckoo Hashing**
  With Flemming Friche Rodler
  ESA 2001
- **One-Probe Search**
  With Anna Östlin
  ICALP 2002
- **Simulating Uniform Hashing in Constant Time and Optimal Space**
  With Anna Östlin
  Unpublished manuscript, 2002