European Symposium on Algorithms (ESA 23)

Learned Monotone Minimal Perfect Hashing

Paolo Ferragina[1], Hans-Peter Lehmann[2], Peter Sanders[2], and **Giorgio Vinciguerra**[1]

1 Università di Pisa

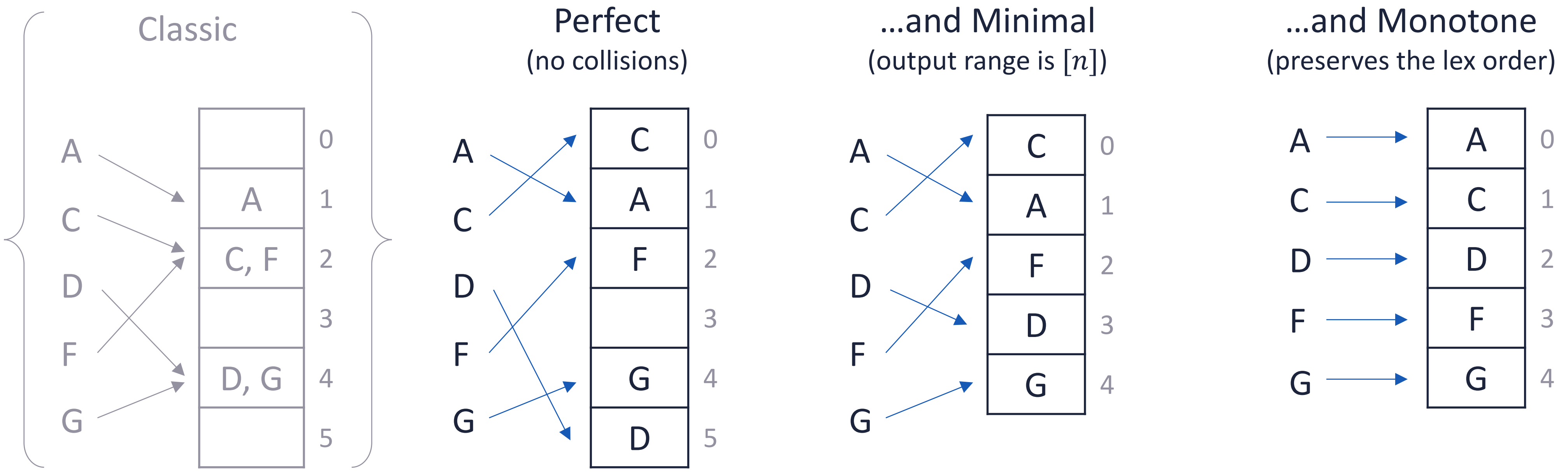
2 KIT Karlsruher Institut für Technologie

European Research Council
Established by the European Commission

Monotone Minimal Perfect Hash Functions (MMPHFs)

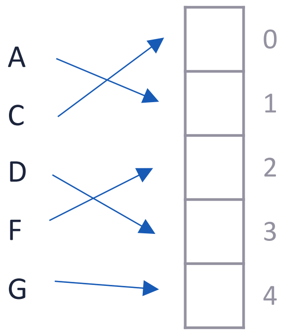
Given a set S of n keys from a universe $[u] = \{0, \ldots, u-1\}$

Construct a hash function that maps keys $\in S$ to their rank, and keys $\notin S$ to an arbitrary value



Why are MMPHFs interesting?

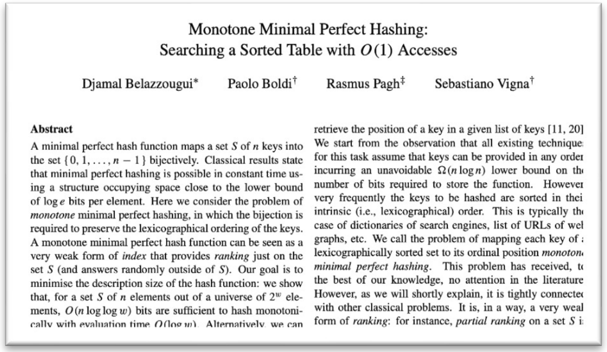
Minimal Perfect Hash Functions (MPHF)



Take ≥ 1.44 bits/key
But no ranks

MMPHFs [SODA 09]

Exploit lex order



Monotone Minimal Perfect Hashing:
Searching a Sorted Table with $O(1)$ Accesses

Djamal Belazzougui* Paolo Boldi† Rasmus Pagh‡ Sebastiano Vigna†

Abstract

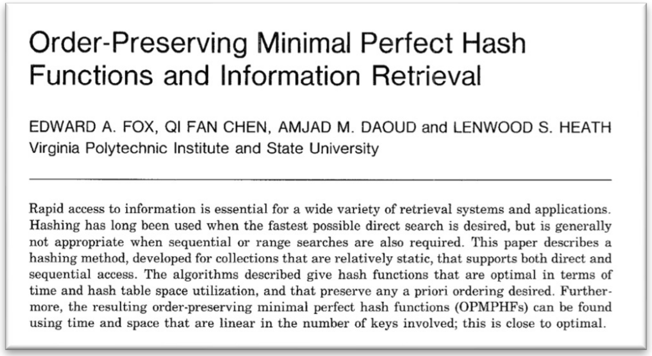
A minimal perfect hash function maps a set S of n keys into the set $\{0, 1, \ldots, n-1\}$ bijectively. Classical results state that minimal perfect hashing is possible in constant time using a structure occupying space close to the lower bound of $\log e$ bits per element. Here we consider the problem of *monotone* minimal perfect hashing, in which the bijection is required to preserve the lexicographical ordering of the keys. A monotone minimal perfect hash function can be seen as a very weak form of *index* that provides *ranking* just on the set S (and answers randomly outside of S). Our goal is to minimise the description size of the hash function: we show that, for a set S of n elements out of a universe of 2^w elements, $O(n \log\log w)$ bits are sufficient to hash monotonically with evaluation time $O(\log w)$. Alternatively, we can

retrieve the position of a key in a given list of keys [11, 20] We start from the observation that all existing technique for this task assume that keys can be provided in any order incurring an unavoidable $\Omega(n \log n)$ lower bound on the number of bits required to store the function. However very frequently the keys to be hashed are sorted in thei intrinsic (i.e., lexicographical) order. This is typically the case of dictionaries of search engines, list of URLs of wel graphs, etc. We call the problem of mapping each key of a lexicographically sorted set to its ordinal position *monoton minimal perfect hashing*. This problem has received, to the best of our knowledge, no attention in the literature However, as we will shortly explain, it is tightly connecte with other classical problems. It is, in a way, a very weal form of *ranking*: for instance, *partial ranking* on a set S i

Return ranks of keys in S
in $\mathcal{O}(\log\log\log u)$ bits/key

Order-Preserving MPHFs [TOIS 91]

Any order



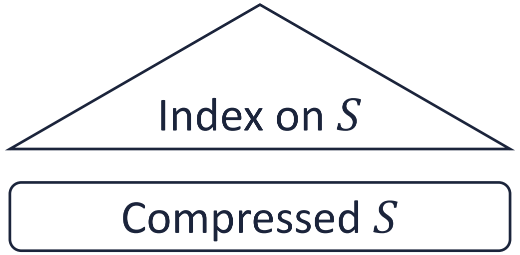
Order-Preserving Minimal Perfect Hash Functions and Information Retrieval

EDWARD A. FOX, QI FAN CHEN, AMJAD M. DAOUD and LENWOOD S. HEATH
Virginia Polytechnic Institute and State University

Rapid access to information is essential for a wide variety of retrieval systems and applications. Hashing has long been used when the fastest possible direct search is desired, but is generally not appropriate when sequential or range searches are also required. This paper describes a hashing method, developed for collections that are relatively static, that supports both direct and sequential access. The algorithms described give hash functions that are optimal in terms of time and hash table space utilization, and that preserve any a priori ordering desired. Furthermore, the resulting order-preserving minimal perfect hash functions (OPMPHFs) can be found using time and space that are linear in the number of keys involved; this is close to optimal.

Return ranks of keys in S
in $\Omega(\log n)$ bits/key

Rank data structures



Return rank of **any** key
in $\Omega\left(\log\frac{u}{n}\right)$ bits/key

More space*

Less powerful rank

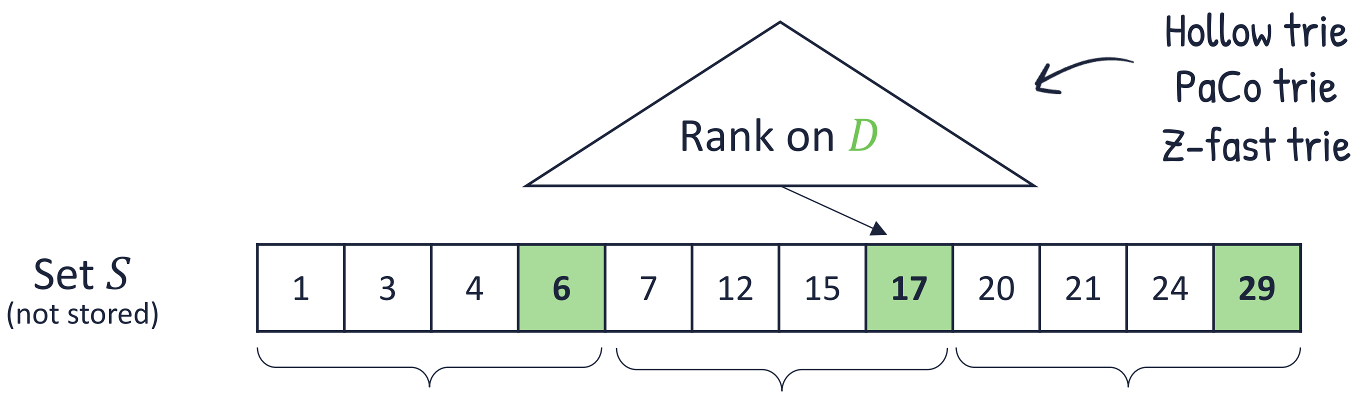
Applications of MMPHFs in databases, pattern matching, and search engines

Key tool: Retrieval data structures

- Associate given r-bit values to keys in S, and retrieve them in $\mathcal{O}(1)$ time

- Take rn bits + small overhead
  - → $o(n)$ bits in theory Dietzfelbinger and Pagh [ICALP 08], Porat [CSR 09]
  - → < 0.01 rn bits in practice with BuRR Dillinger et al. [SEA 22]

Known approaches for MMPHFs

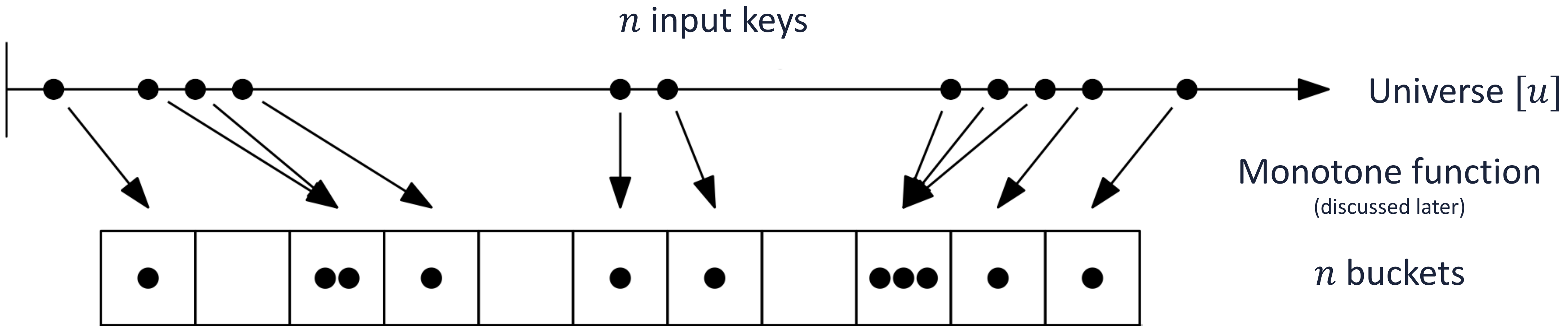


1. Form equal-size buckets and store local ranks with a retrieval data structure
2. Build a (relative) rank data structure on the bucket delimiters D to route keys in S to buckets

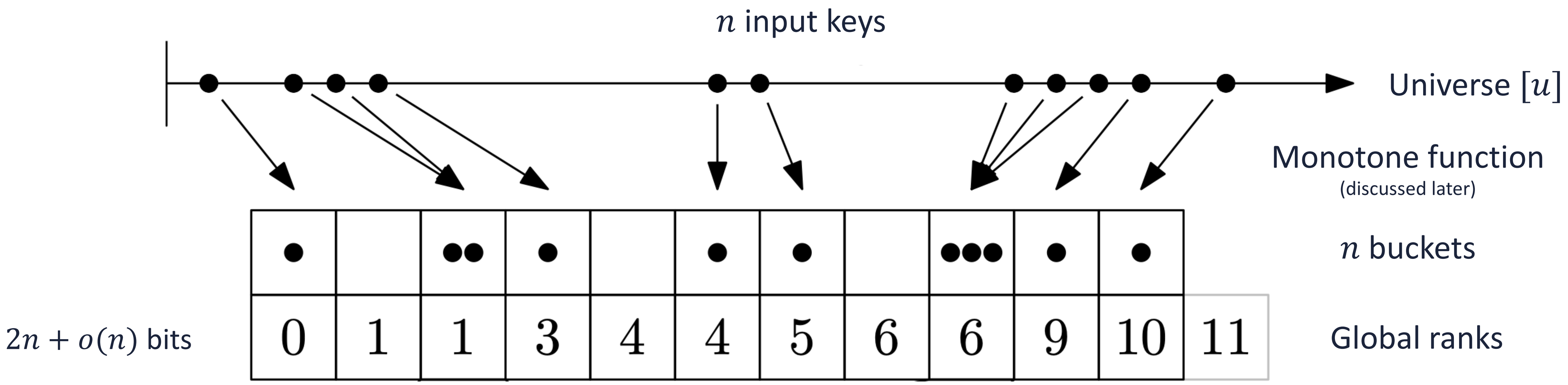
This is optimal Assadi et al. [SODA 23]

Space: $\mathcal{O}(\log\log\log u)$ bits/key
Queries: $\mathcal{O}(\log\log u)$ time

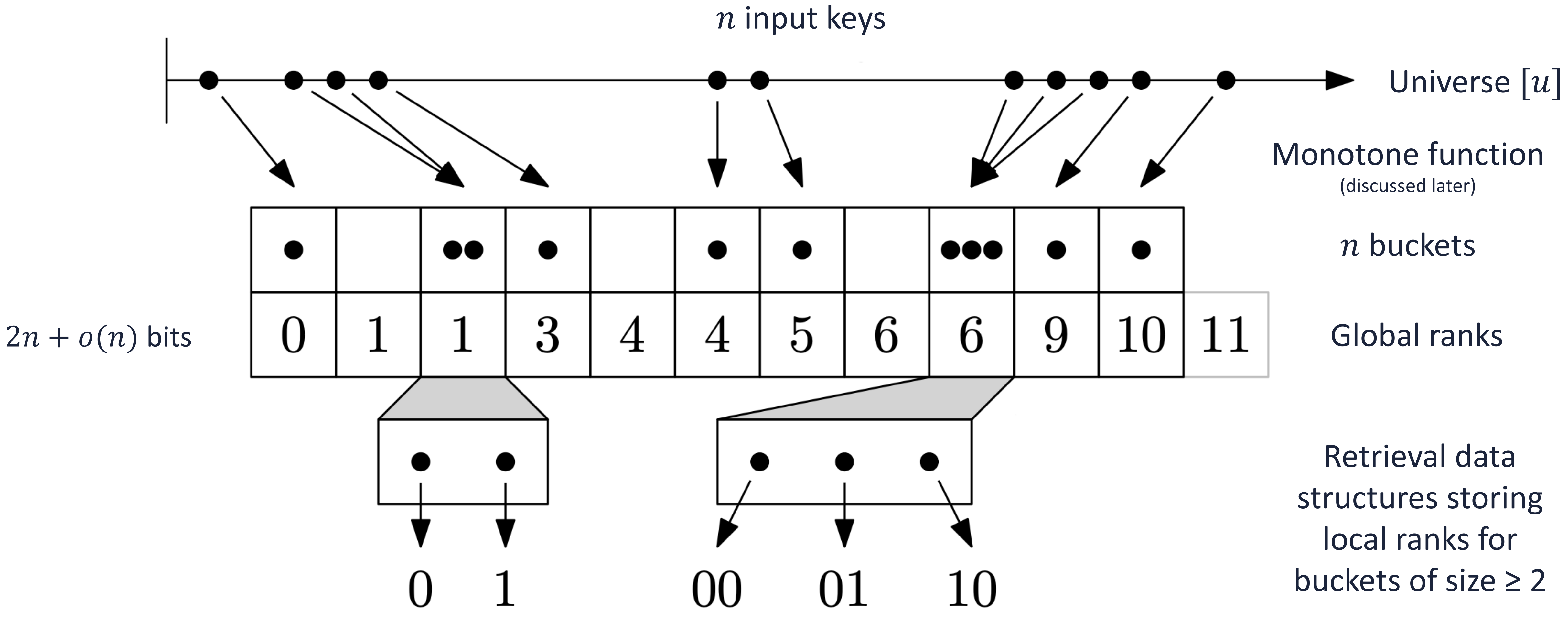
Our MMPHF


n input keys
Universe $[u]$
Monotone function
(discussed later)
n buckets

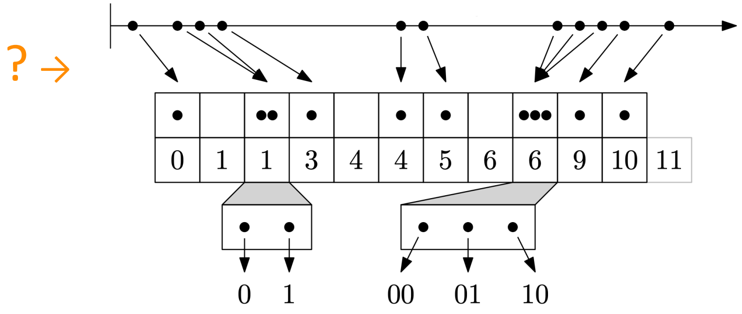
Our MMPHF


n input keys
Universe $[u]$
Monotone function
(discussed later)
n buckets
$2n + o(n)$ bits
0 1 1 3 4 4 5 6 6 9 10 11
Global ranks

Our MMPHF



n input keys
Universe $[u]$
Monotone function
(discussed later)
n buckets
$2n + o(n)$ bits
0
1
1
3
4
4
5
6
6
9
10
11
Global ranks
0
1
00
01
10
Retrieval data structures storing local ranks for buckets of size ≥ 2



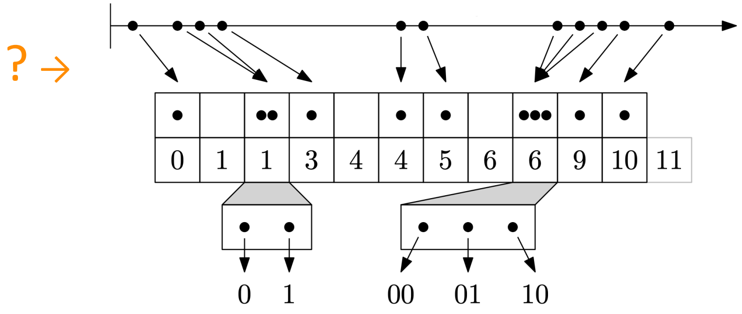
How to map to buckets

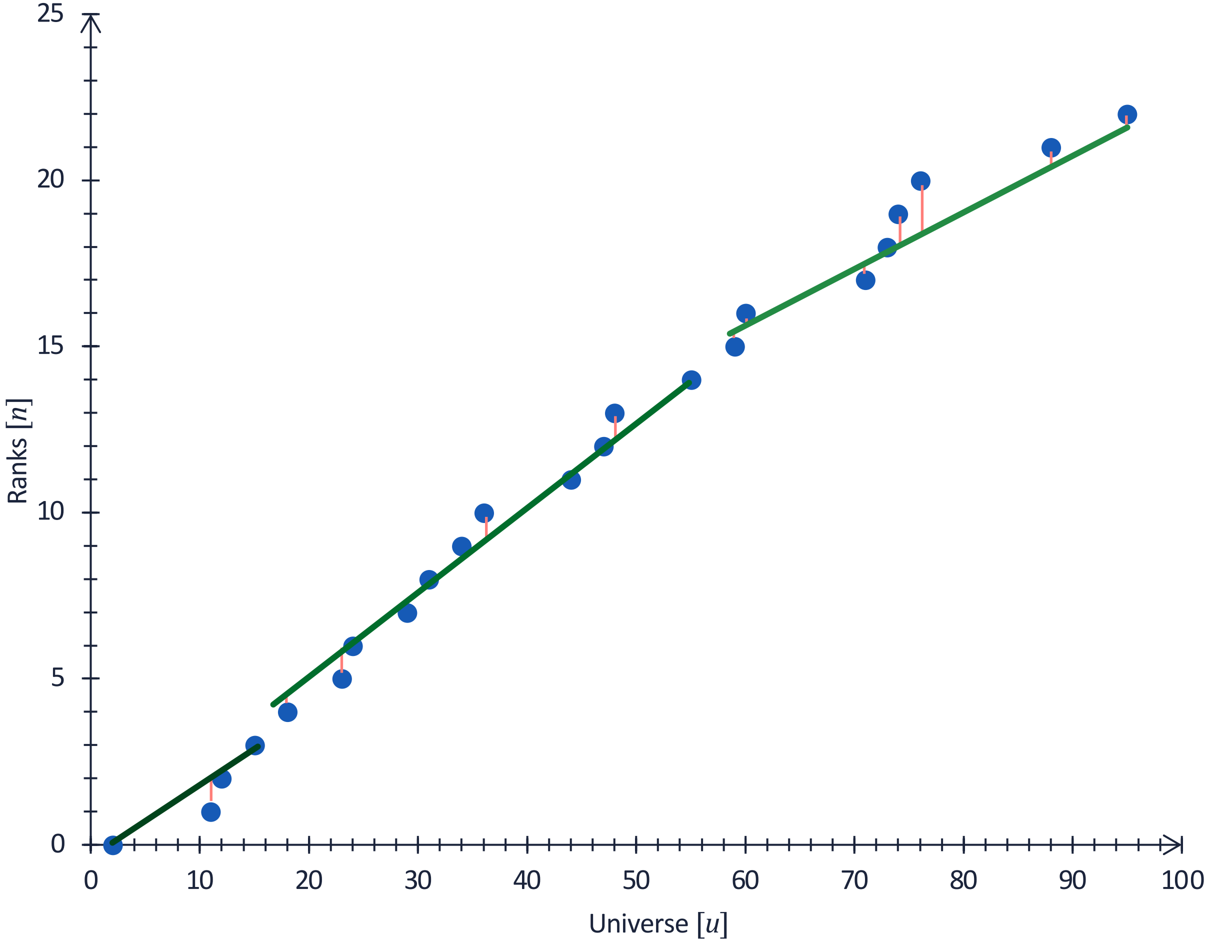
- Suppose the input integers are **uniform**
- Map x to bucket number $\left\lfloor \frac{x}{u} n \right\rfloor$, i.e. a linear mapping

Theorem 1.
Our MMPHF on **uniform** integers needs $n(2.915 + o(1))$ bits on average and can be queried in $\mathcal{O}(1)$ time.

- This breaks the lower bound of $\Omega(n \log \log \log u)$ bits for a MMPHF
- Learning and leveraging the input data smoothness: **LeMonHash**

How to map non-uniform data





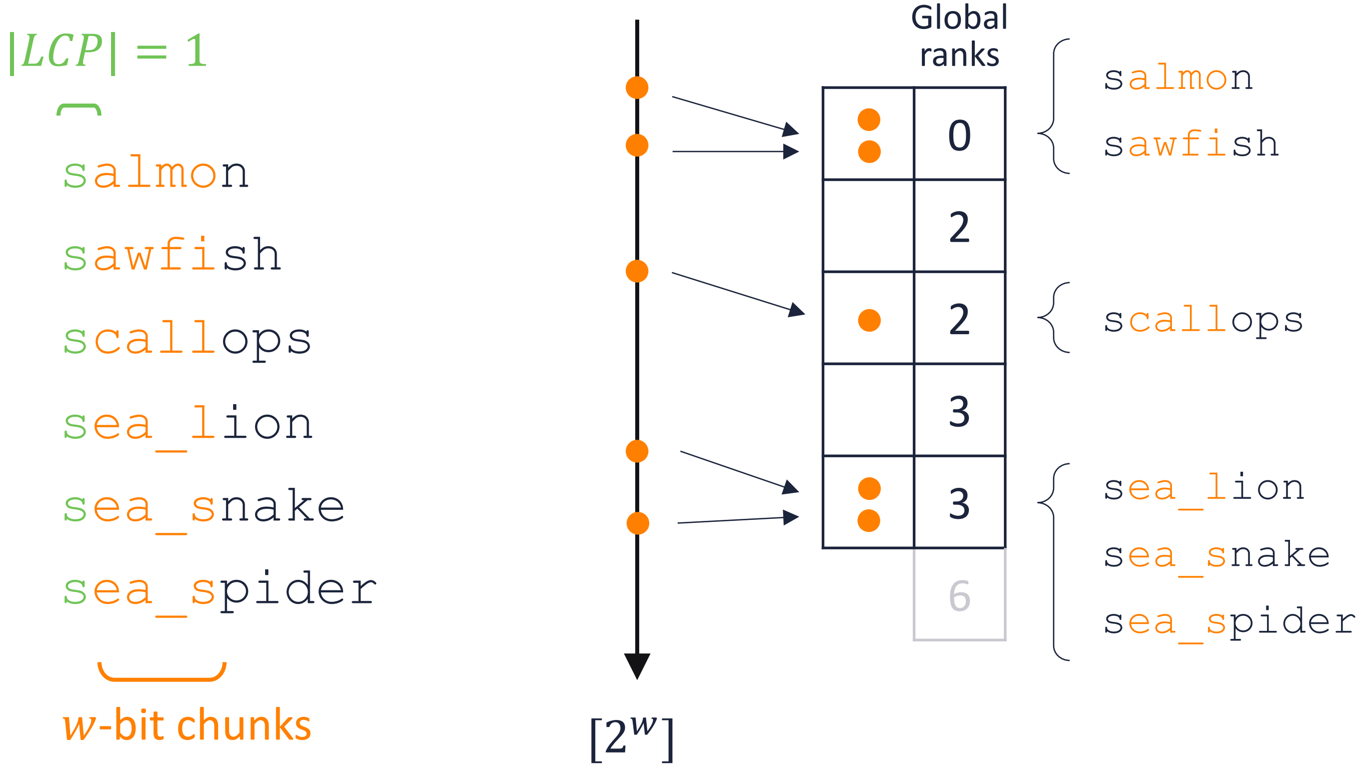
- Learn a **piecewise linear ε-approximation** of the function keys → ranks Ferragina, V. [VLDB 20]
  - |Rank estimate – True rank| $\leq$ a given integer ε
  - The more the data is smooth the smaller is the number of segments
- Rank estimate for a key = bucket index

LeMonHash bounds

Theorem 1.
LeMonHash on *uniform integers* takes $n(2.915 + o(1))$ bits on average and can be queried in $\mathcal{O}(1)$ time.

Theorem 2.
LeMonHash takes $n\big(\underbrace{\log(2\varepsilon + 1)}_{\text{Local ranks}} + \underbrace{2 + o(1)}_{\text{Global ranks}}\big) + \underbrace{\mathcal{O}\left(m \log \frac{u}{m}\right)}_{\text{Piecewise linear } \varepsilon\text{-approx. with } m \text{ segments}}$ bits *in the worst case* and can be queried in $\mathcal{O}(\log \log u)$ time.

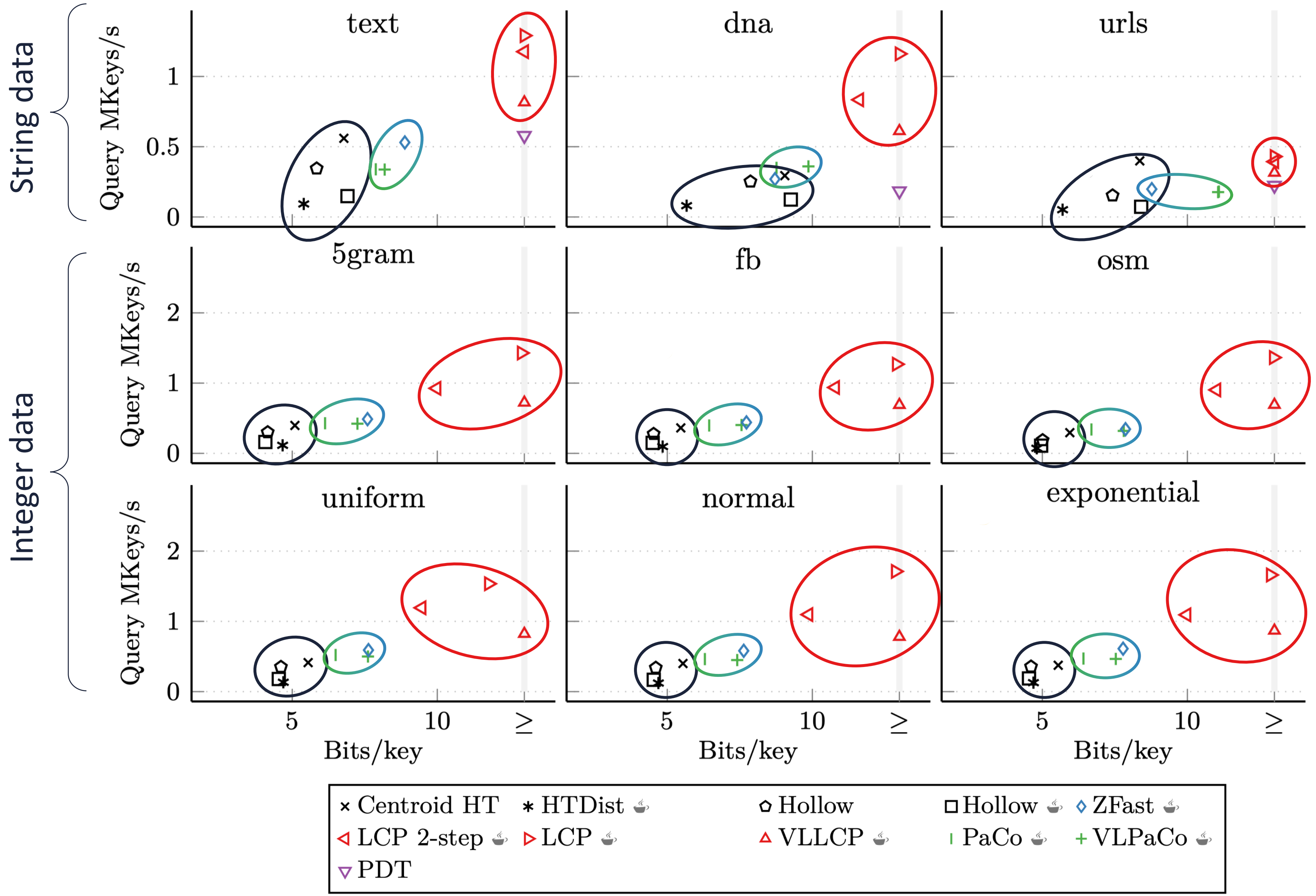
Handling variable-length strings



#strings $\geq t$ $\Rightarrow$ handle recursively thus creating a tree

Many optimisations in the paper

Experiments: space vs query throughput



Existing Pareto front:

- **Hollow-trie** approaches
- **ZFast** / **PaCo**
- **LCP** approaches

LeMonHash:

- On *string data*, space within 13% of the best competitors, and up to 3× faster queries than the larger competitor
- On *integer data*, dominates in space-time all competitors (except for the space on fb)
- Improved construction throughput by up to 2×

Conclusion

LeMonHash: New MMPHF that learns and leverages data smoothness

- Can break the superlinear lower bound on MMPHFs' space
- In practice: on most datasets, dominates all competitors on space usage, query and construction throughput, *simultaneously*

Open problems

1. Strengthen our pessimistic bounds
2. Extend to nonlinear models