ICDE 2018

Adaptive Adaptive Indexing

Felix Martin Schuhknecht

Jens Dittrich

Laurent Linden

Big Data Analytics Group

bigdata.uni-saarland.de

Saarland University

2007

Database Cracking

Stratos Idreos
CWI Amsterdam
The Netherlands
Stratos.Idreos@cwi.nl

Martin L. Kersten
CWI Amsterdam
The Netherlands
Martin.Kersten@cwi.nl

Stefan Manegold
CWI Amsterdam
The Netherlands
Stefan.Manegold@cwi.nl

ABSTRACT

Database indices provide a non-discriminative navigational infrastructure to localize tuples of interest. Their maintenance cost is taken during database updates. In this paper, we study the complementary approach, addressing index maintenance as part of query processing using continuous physical reorganization, i.e., *cracking* the database into manageable pieces. The motivation is that by automatically organizing data the way users request it, we can achieve fast access and the much desired self-organized behavior.

We present the first mature cracking architecture and report on our implementation of cracking in the context of a full fledged relational system. It led to a minor enhancement to its relational algebra kernel, such that cracking could be piggy-backed without incurring too much processing overhead. Furthermore, we illustrate the ripple effect of dynamic reorganization on the query plans derived by the SQL optimizer. The experiences and results obtained are indicative of a significant reduction in system complexity. We show that the resulting system is able to self-organize based on incoming requests with clear performance benefits. This behavior is visible even when the user focus is randomly shifting to different parts of the data.

1. INTRODUCTION

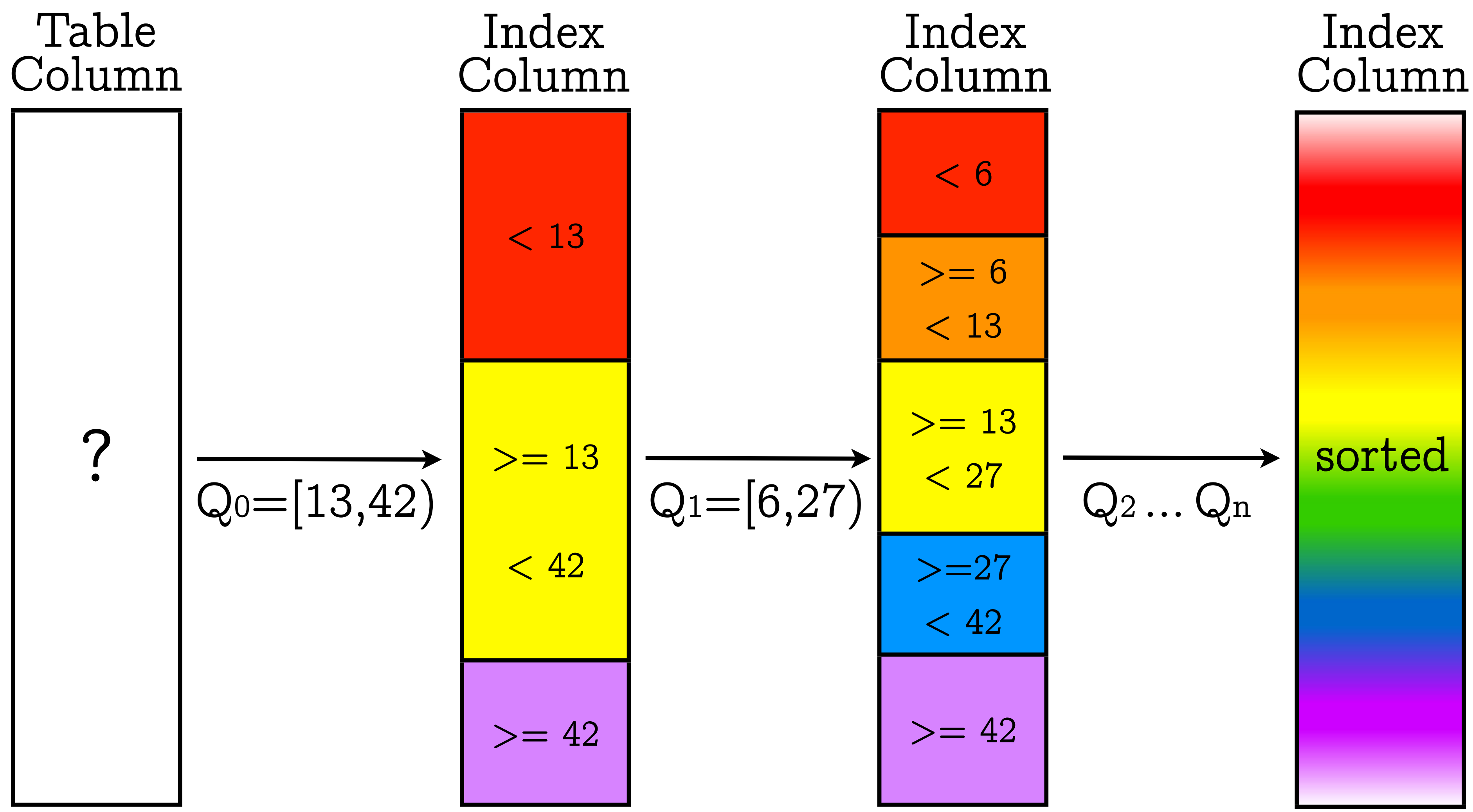
Nowadays, the challenge for database architecture design is not in achieving ultra high performance but to design systems that are *simple* and *flexible*. A database system should be able to handle *huge* sets of data and *self-organize* according to the environment, e.g., the workload, available resources, etc. A nice discussion on such issues can be found in [6]. In addition, the trend towards distributed environments to speed up computation calls for new architecture designs. The same holds for multi-core CPU architectures that are starting to dominate the market and open new possibilities and challenges for data management. Some notable departures from the usual paths in database architecture design include [2, 3, 9, 14].

In this paper, we explore a radically new approach in database architecture, called *database cracking*. The cracking approach is based on the hypothesis that index maintenance should be a byproduct of query processing, not of updates. Each query is interpreted not only as a request for a particular result set, but also as an *advice* to crack the physical database store into smaller pieces. Each piece is described by a query, all of which are assembled in a *cracker index* to speedup future search. The cracker index replaces the non-discriminative indices (e.g., B-trees and hash tables) with a discriminative index. Only database portions of past interest are easily localized. The remainder is unexplored territory and remains non-indexed until a query becomes interested. Continuously reacting on query requests brings the powerful property of self-organization. The cracker index is built dynamically while queries are processed and adapts to changing query workloads.

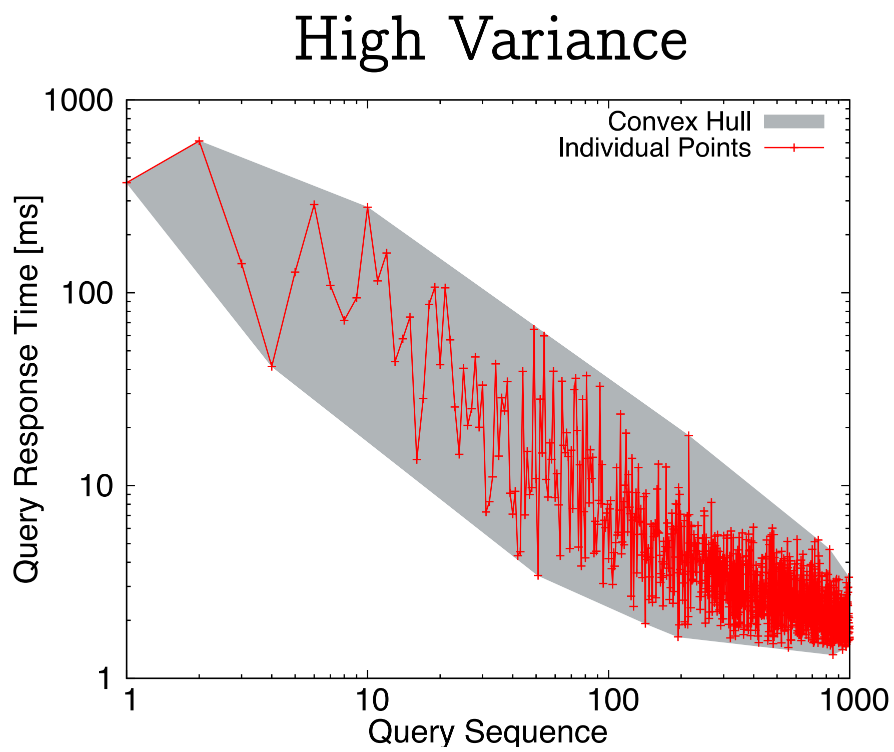
The cracking technique naturally provides a promising basis to attack the challenges described in the beginning of this section. With cracking, the way data is physically stored self-organizes according to query workload. Even with a huge data set, only tuples of interest are touched, leading to significant gains in query performance. In case the focus shifts to a different part of the data, the cracker index automatically adjusts to that. In addition, cracking the database into pieces gives us disjoint sets of our data targeted by specific queries. This information can be nicely used as a basis for high-speed distributed and multi-core query processing.

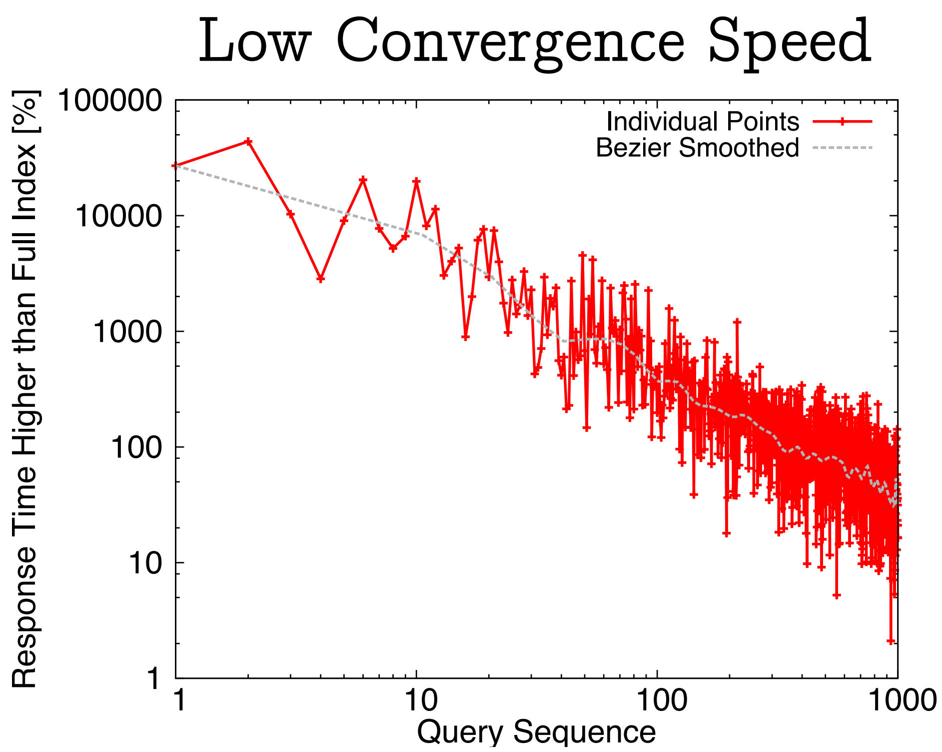
The idea of physically reorganizing the database based on incoming queries has first been proposed in [10]. The contributions of this paper are the following. We present the first mature cracking architecture (a complete cracking software stack) in the context of column oriented databases. We report on our implementation of cracking on top of MonetDB/SQL, a column oriented database system, showing that cracking is easy to implement and may lead to further system simplification. We present the cracking algorithms that physically reorganize the datastore and the new cracking operators to enable cracking in MonetDB. Using SQL micro-benchmarks, we assess the efficiency and effectiveness of the system at the operator level. Additionally, we perform experiments that use the complete software stack, demonstrating that cracker-aware query optimizers can successfully generate query plans that deploy our new cracking operators and thus exploit the benefits of database cracking. Furthermore, we evaluate our current implementation and discuss some promising results. We clearly demonstrate that the resulting system can self-organize according to query

Database Cracking / Standard Cracking

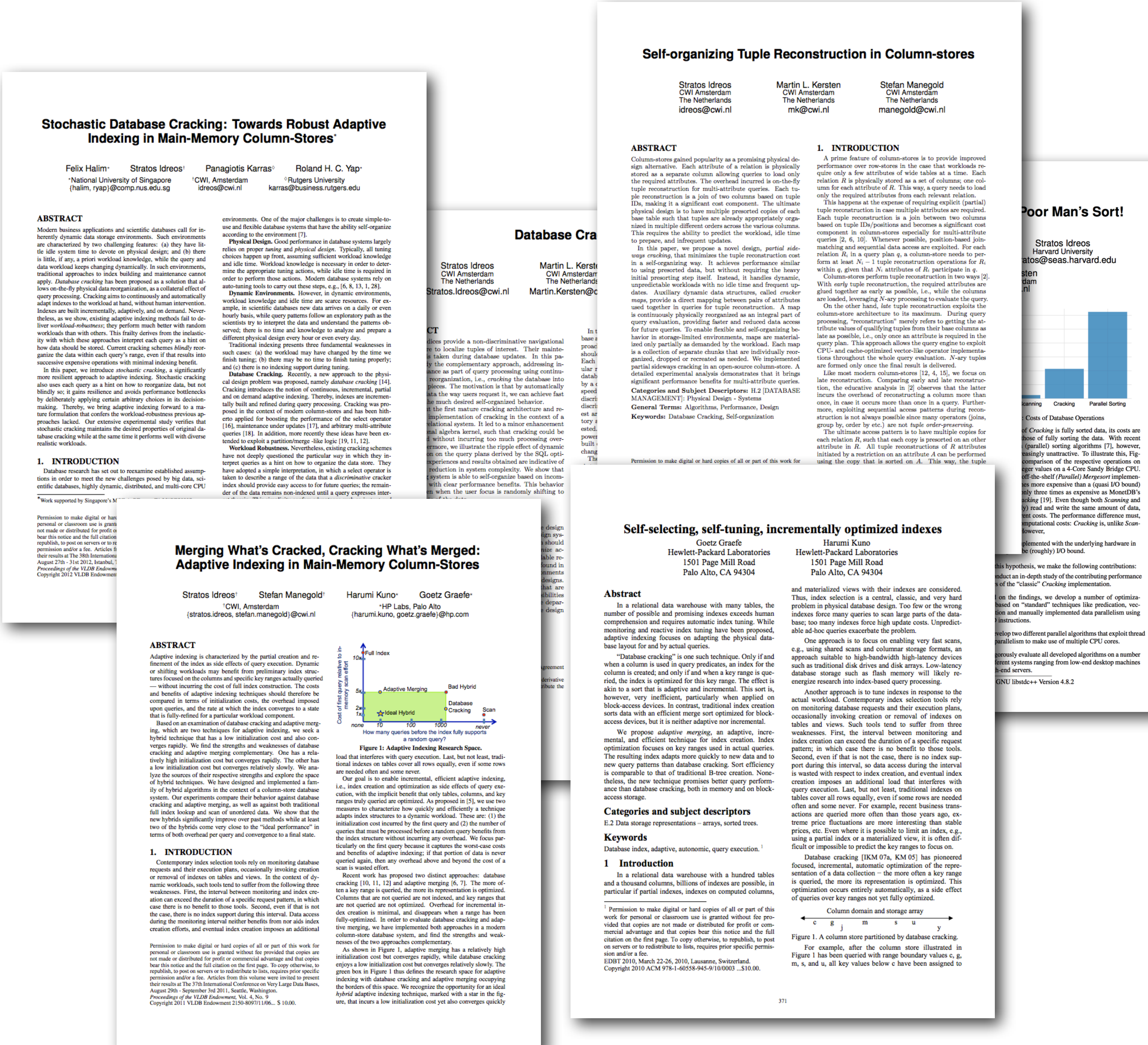


Problems?





[Felix Schuhknecht, Alekh Jindal, Jens Dittrich: The Uncracked Pieces in Database Cracking, PVLDB Vol. 7, No. 2, Best Paper Award]


Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores
Felix Halim Stratos Idreos Panagiotis Karras Roland H. C. Yap
Database Cracking
Stratos Idreos Martin L. Kersten
Self-organizing Tuple Reconstruction in Column-stores
Stratos Idreos Martin L. Kersten Stefan Manegold
CWI Amsterdam The Netherlands
Poor Man's Sort!
Stratos Idreos Harvard University
Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores
Stratos Idreos Stefan Manegold Harumi Kuno Goetz Graefe
Self-selecting, self-tuning, incrementally optimized indexes
Goetz Graefe Hewlett-Packard Laboratories 1501 Page Mill Road Palo Alto, CA 94304
Harumi Kuno Hewlett-Packard Laboratories 1501 Page Mill Road Palo Alto, CA 94304

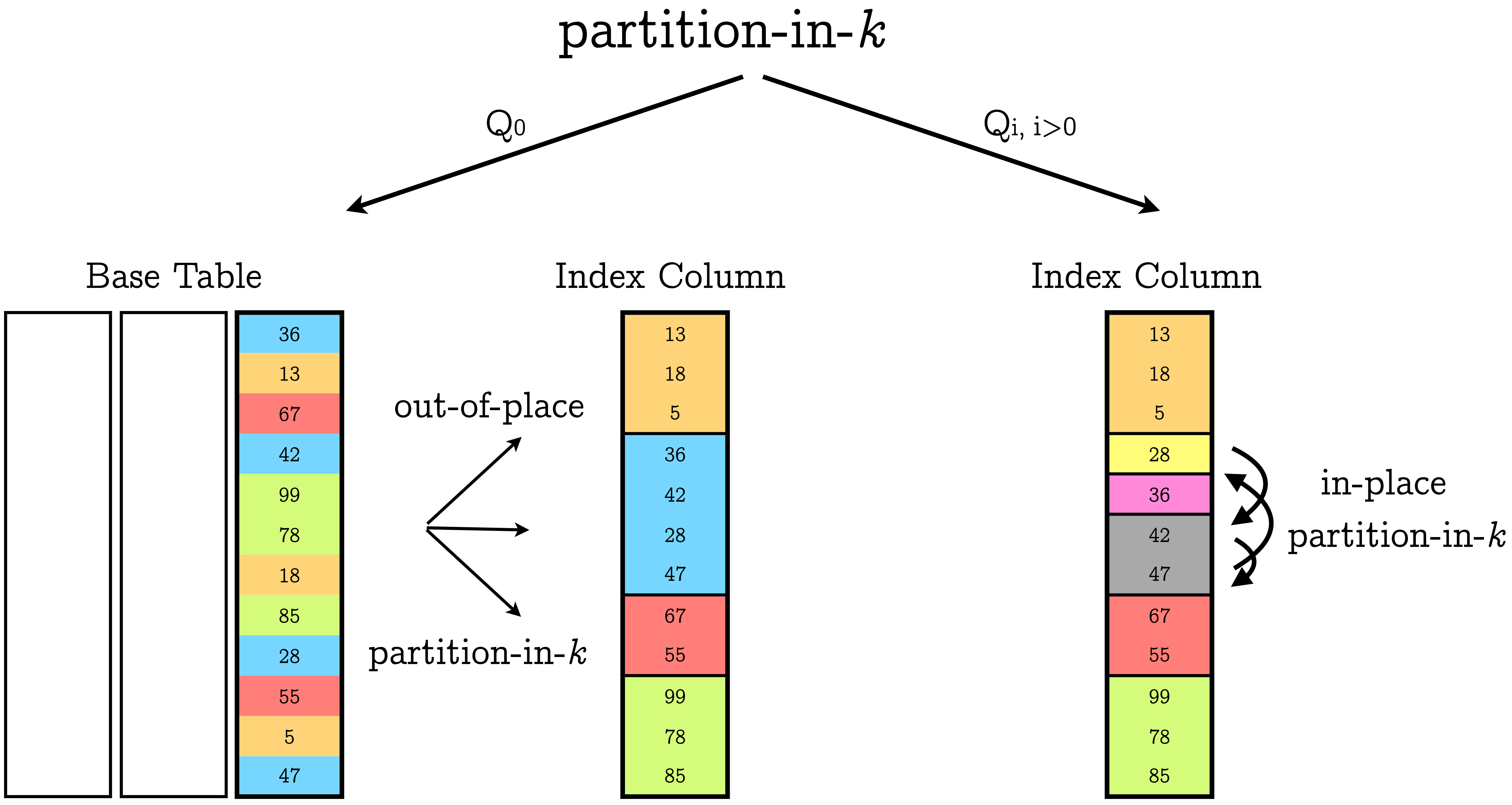
All-in-one?

An Adaptive Adaptive Index?

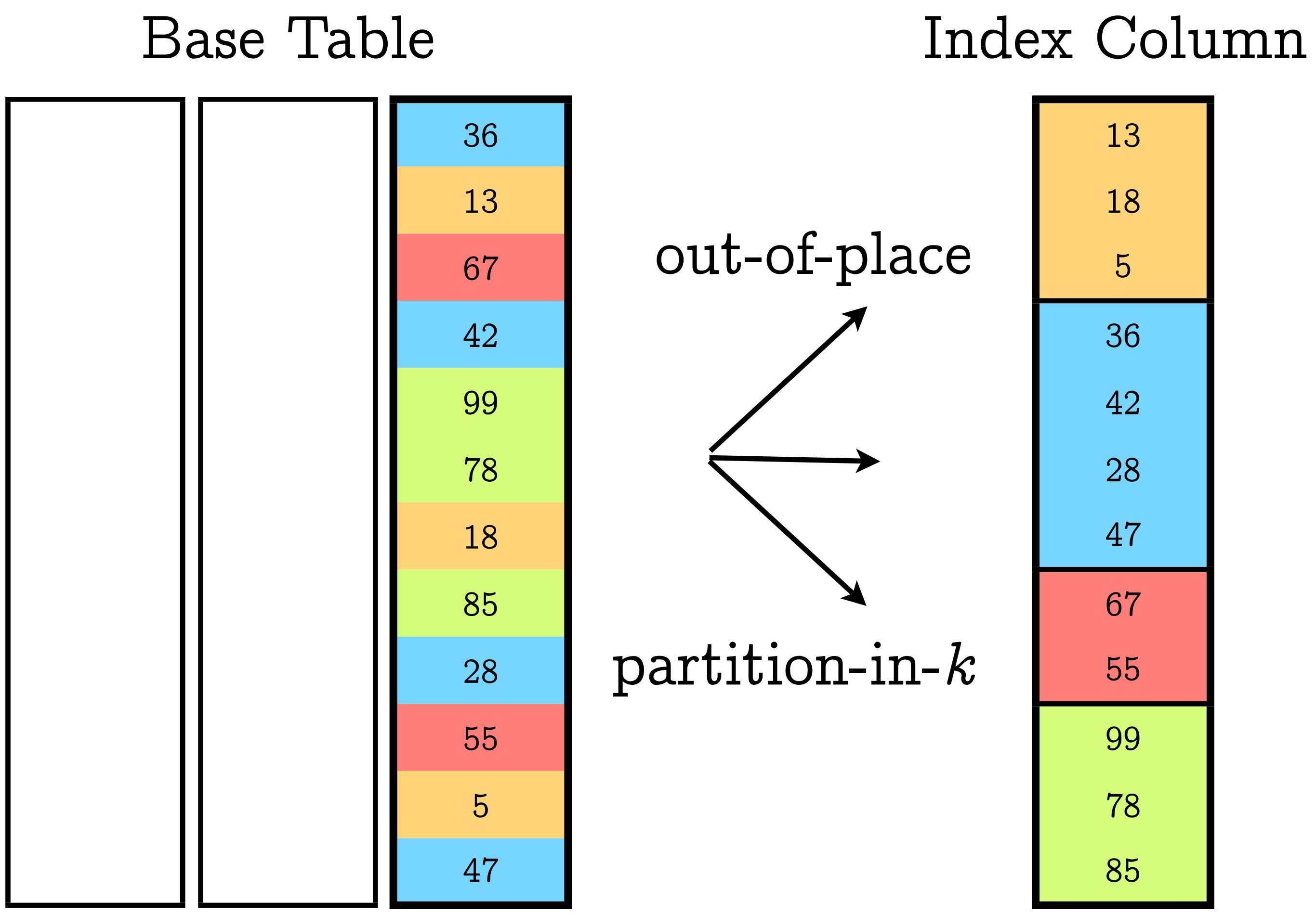
Design rules:

1. Generalize way of refinement
2. Adapt refinement effort
3. Awareness of key distributions

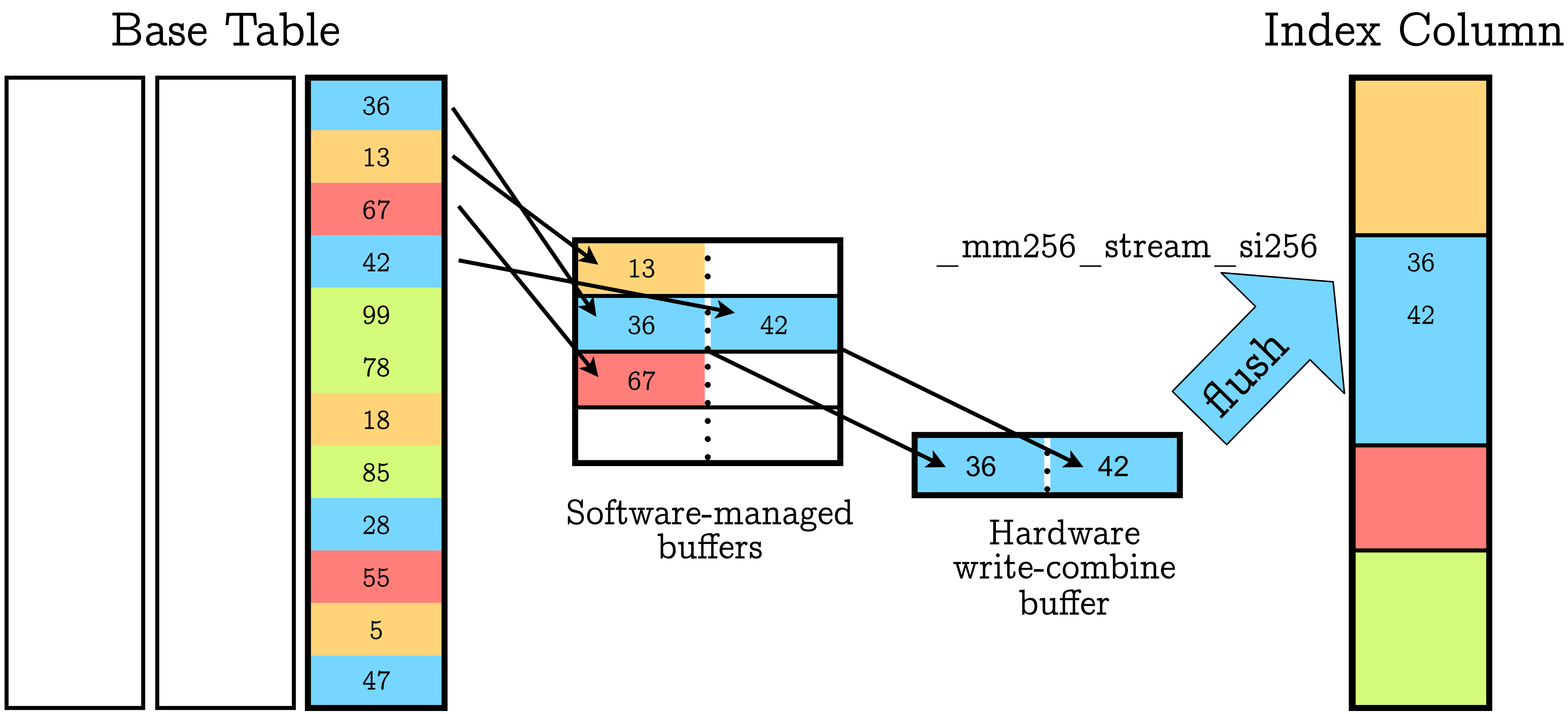
1. Generalize way of refinement:



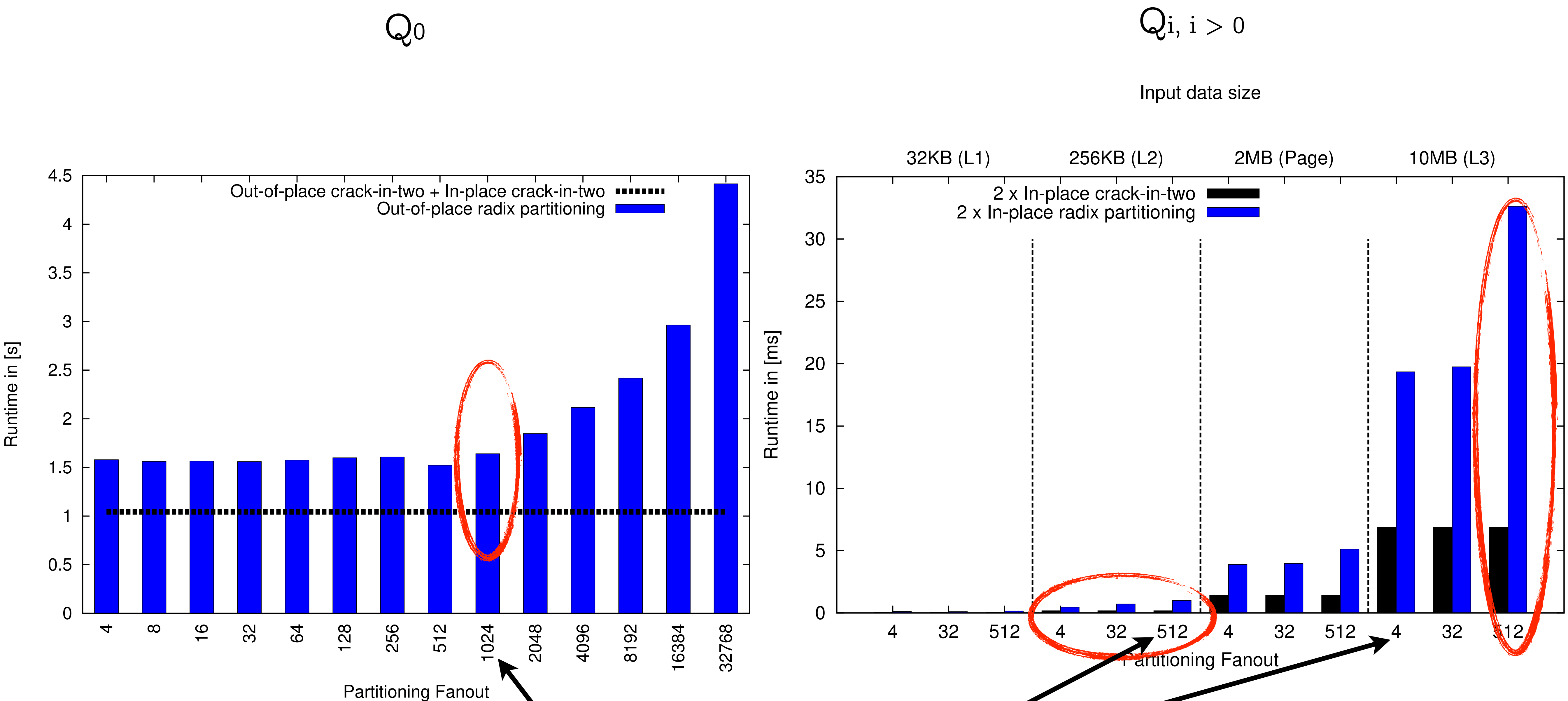
1. Generalize way of refinement:



1. Generalize way of refinement:



2. Adapt refinement effort

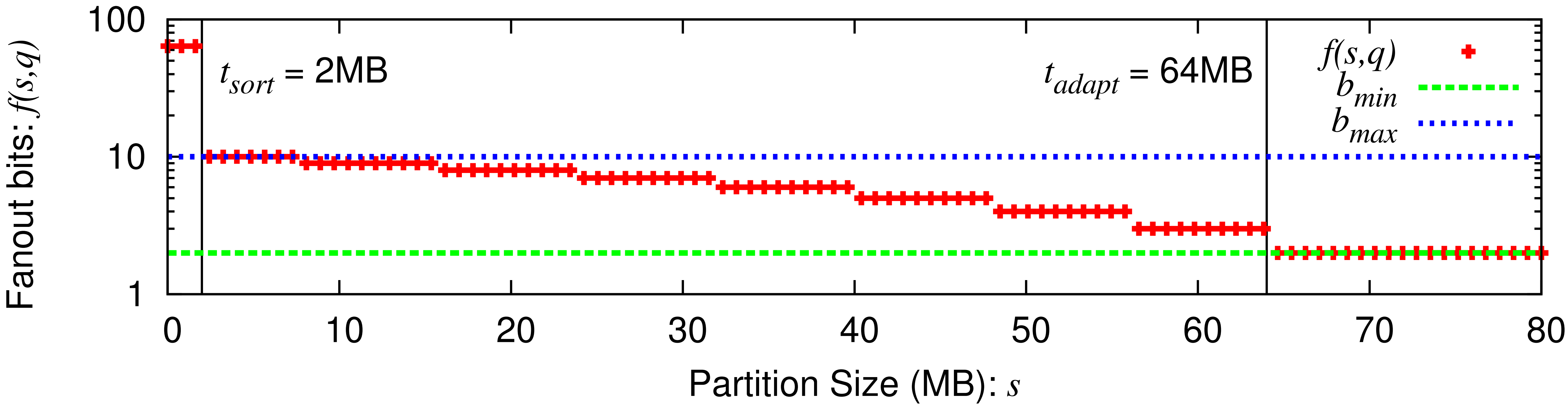


$$f(s,q) = \begin{cases} b_{first} & \text{if } q = 0 \\ b_{min} & \text{else if } s > t_{adapt} \\ b_{min} + \left\lceil (b_{max} - b_{min}) \cdot \left(1 - \frac{s}{t_{adapt}}\right) \right\rceil & \text{else if } s > t_{sort} \\ b_{sort} & \text{else.} \end{cases}$$

partition size

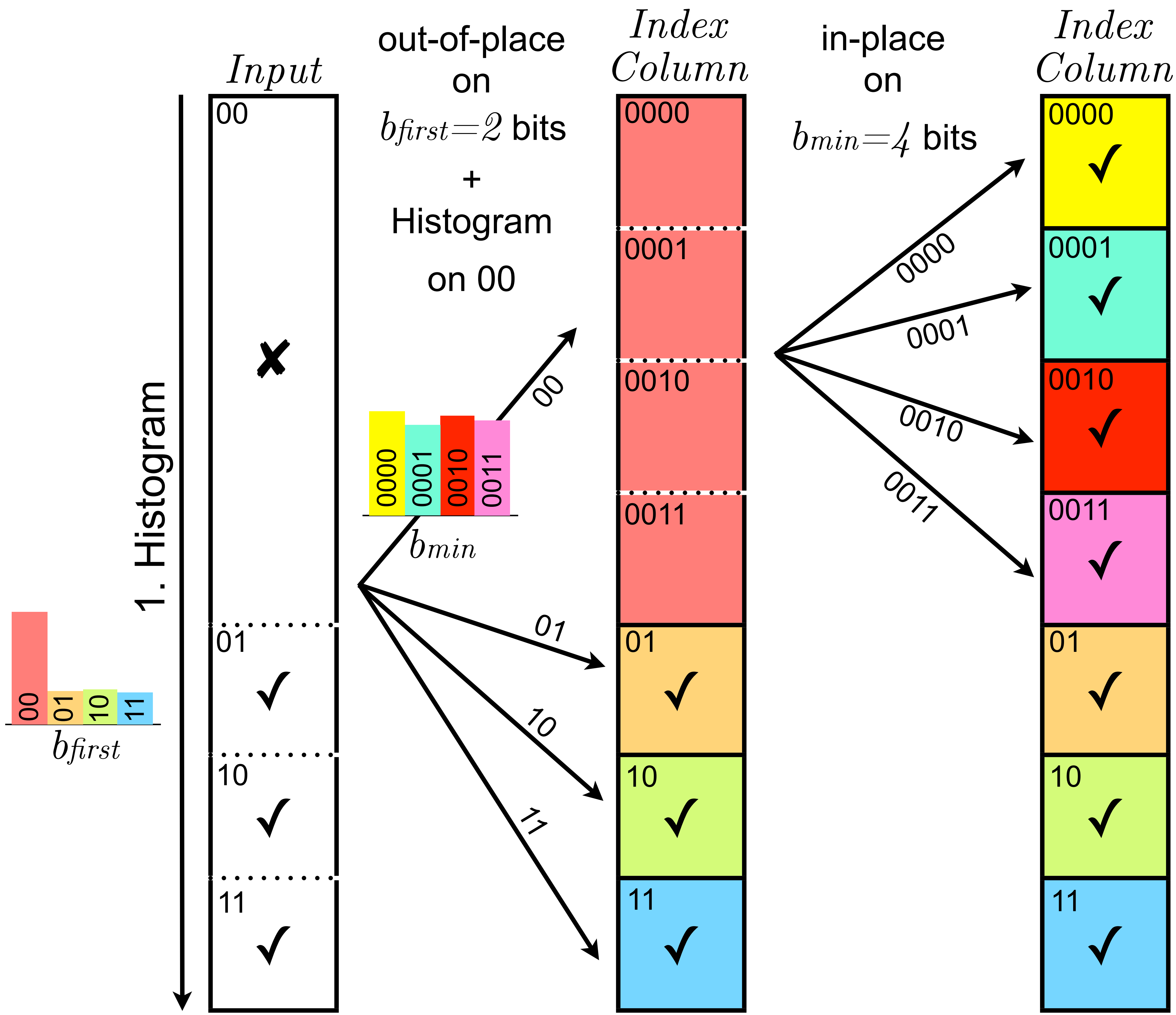
query sequence number

2. Adapt refinement effort

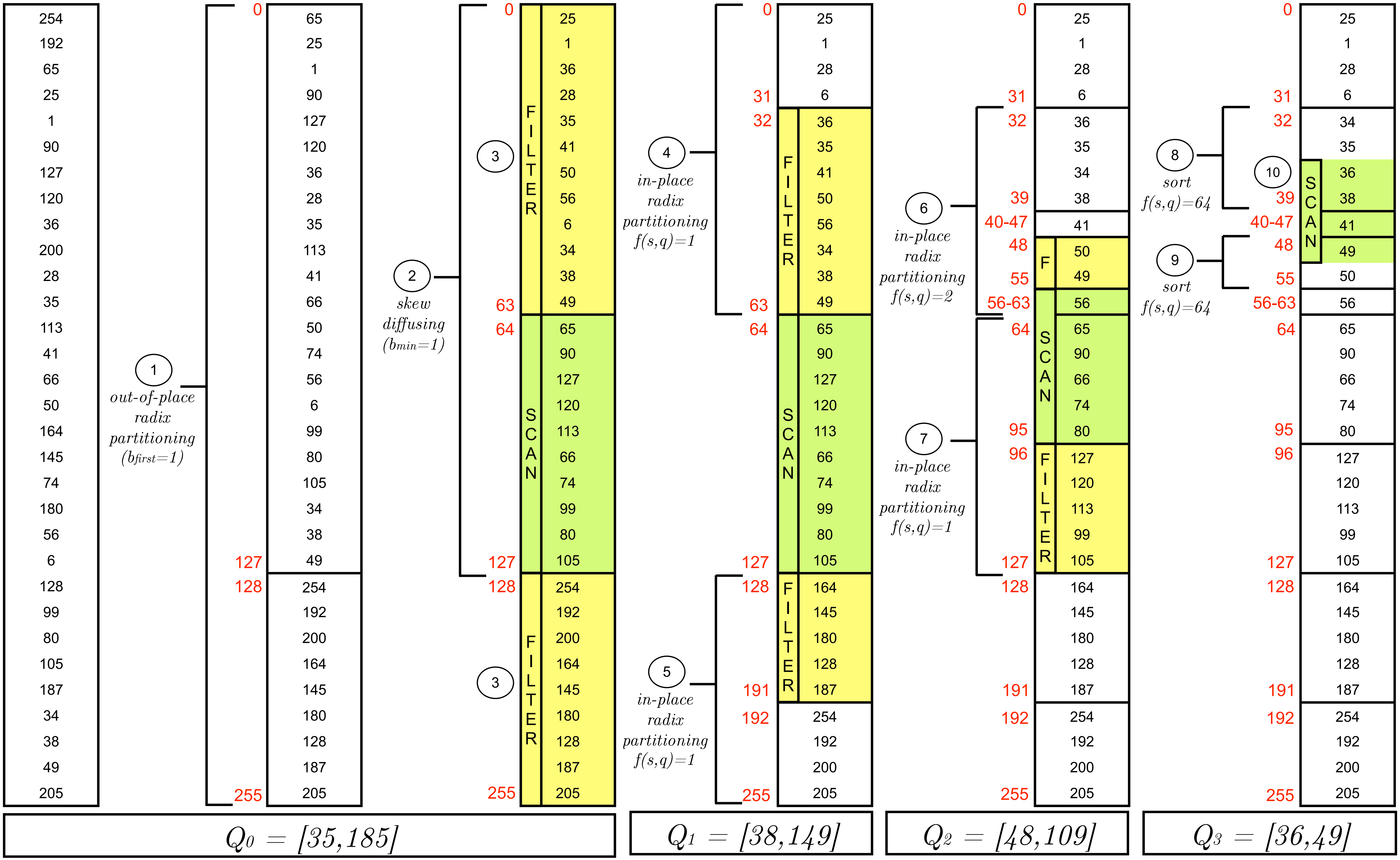


$$f(s,q) = \begin{cases} b_{first} & \text{if } q = 0 \\ b_{min} & \text{else if } s > t_{adapt} \\ b_{min} + \left\lceil (b_{max} - b_{min}) \cdot \left(1 - \frac{s}{t_{adapt}}\right) \right\rceil & \text{else if } s > t_{sort} \\ b_{sort} & \text{else.} \end{cases}$$

3. Awareness of key distributions: skew?

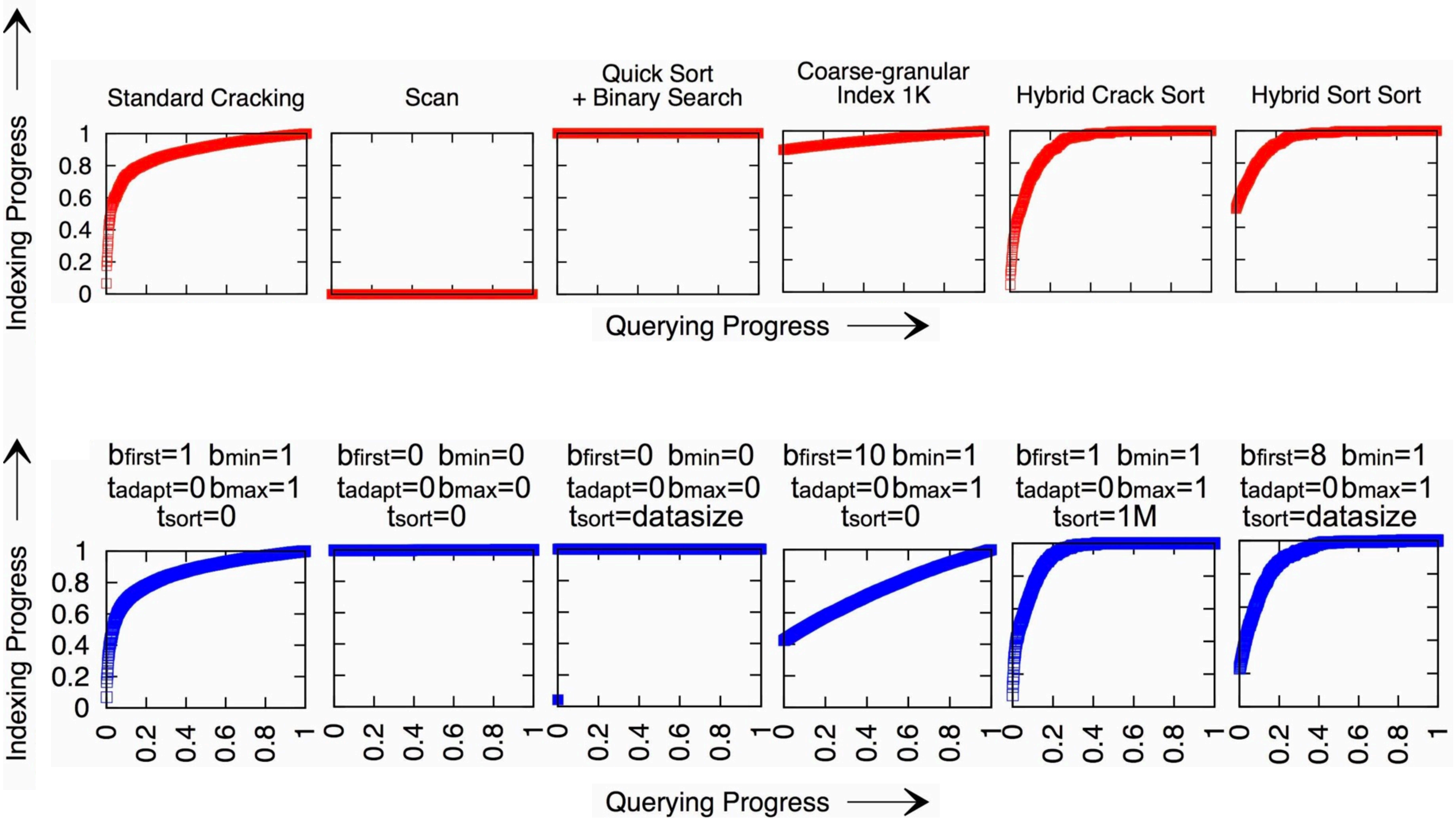


Putting it all together

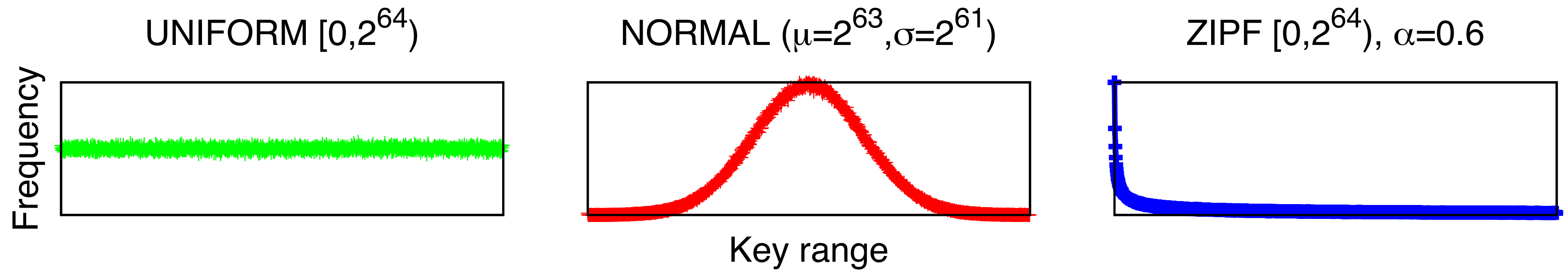


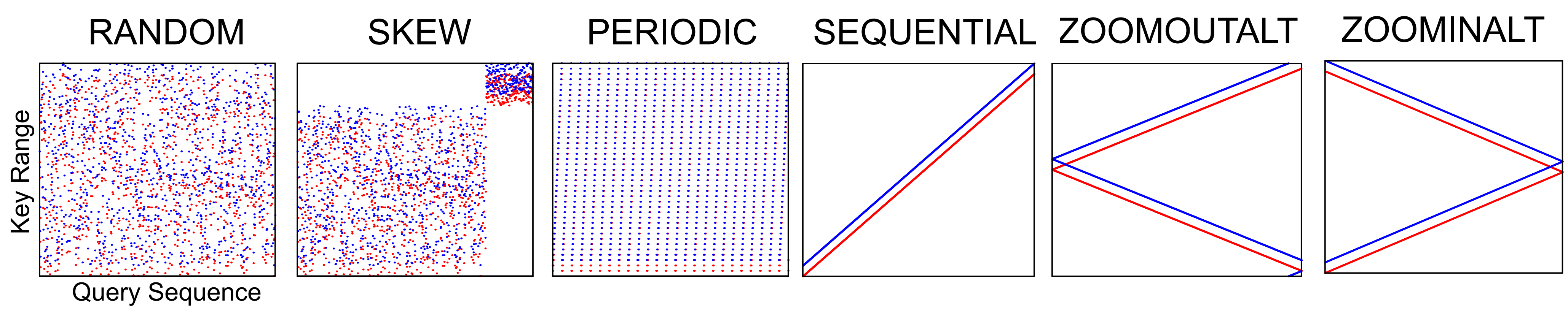
Emulation

[Felix Martin Schuhknecht, Alekh Jindal, Jens Dittrich: The Uncracked Pieces in Database Cracking, PVLDB Vol. 7, No. 2]



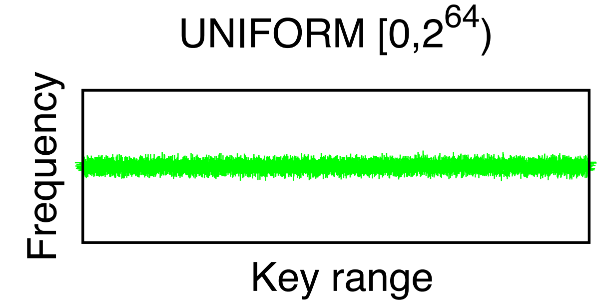
Test Setup

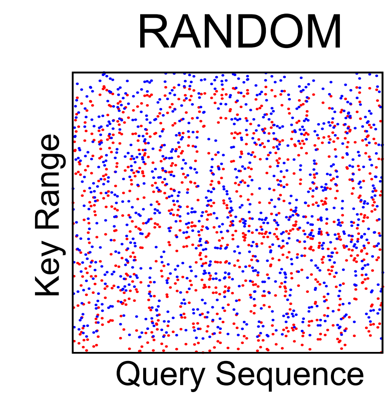


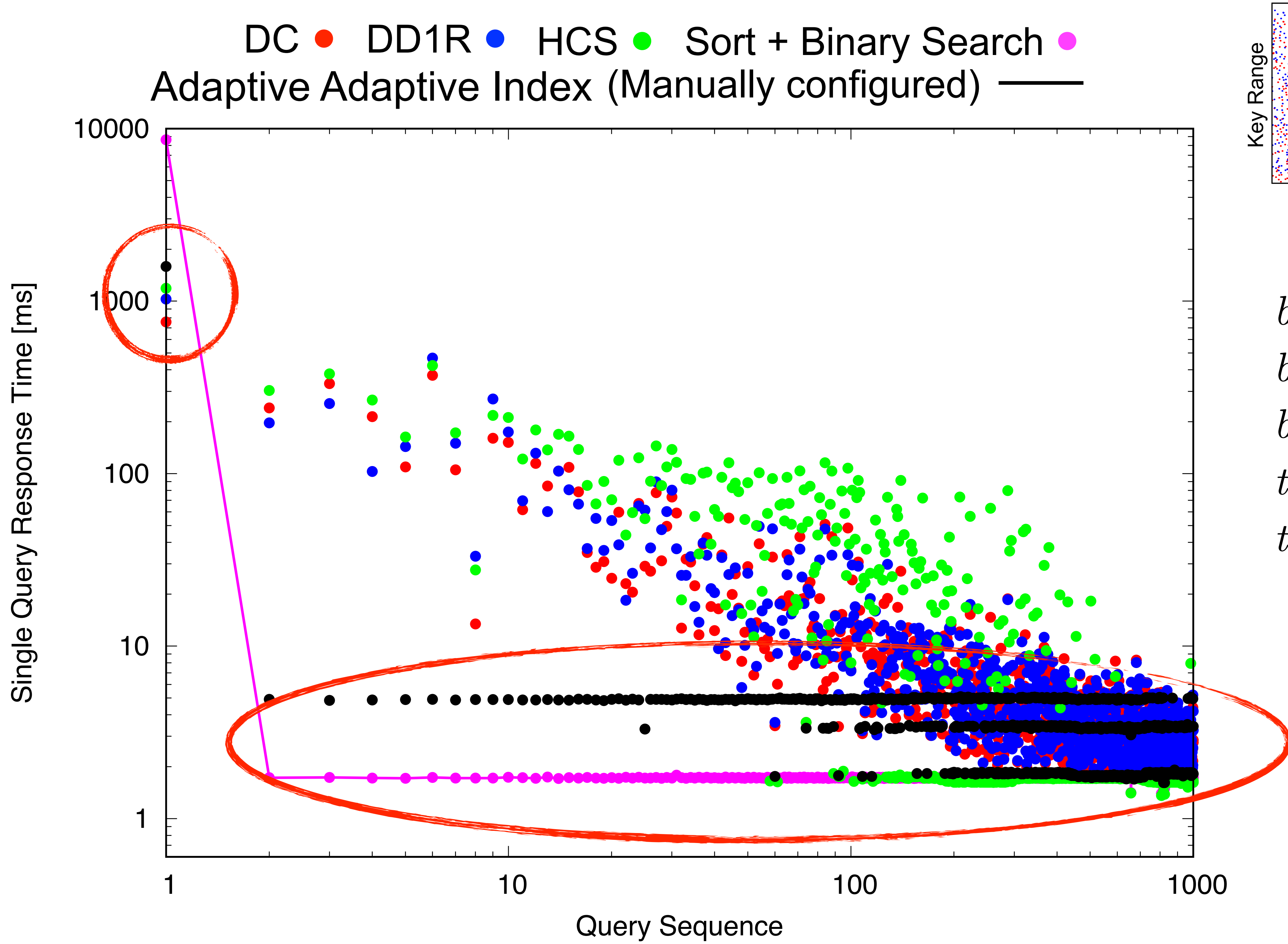


[Felix Halim, Stratos Idreos, Panagiotis Karras, Roland H. C. Yap:
Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores, PVLDB Vol. 5, No. 6]

Individual Query Response Times







$b_{first} = 10$

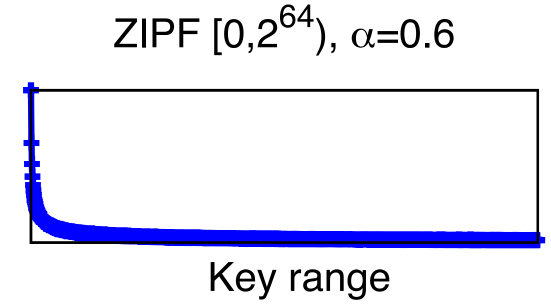
$b_{min} = 3$

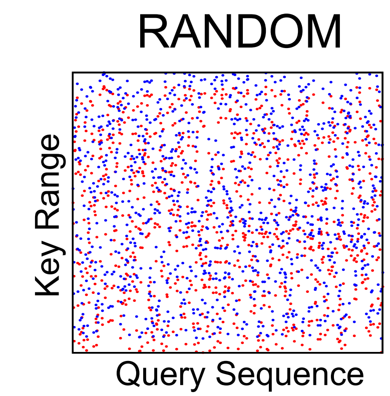
$b_{max} = 6$

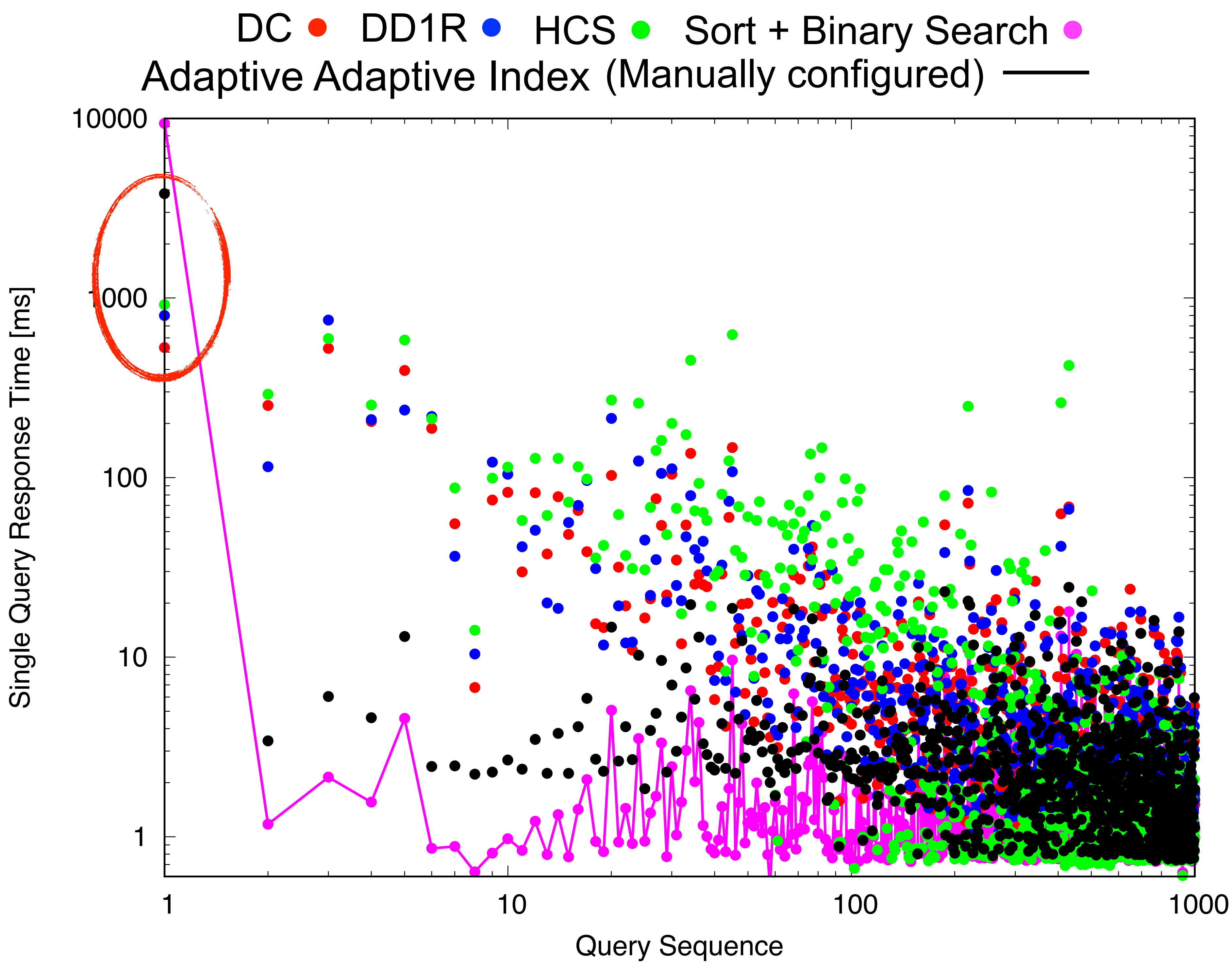
$t_{adapt} = 64MB$

$t_{sort} = 256KB$

Individual Query Response Times







$bfirst=10$

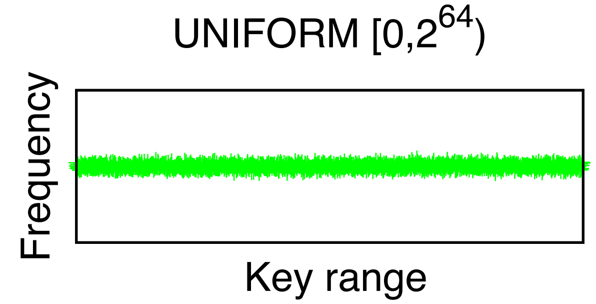
$bmin=3$

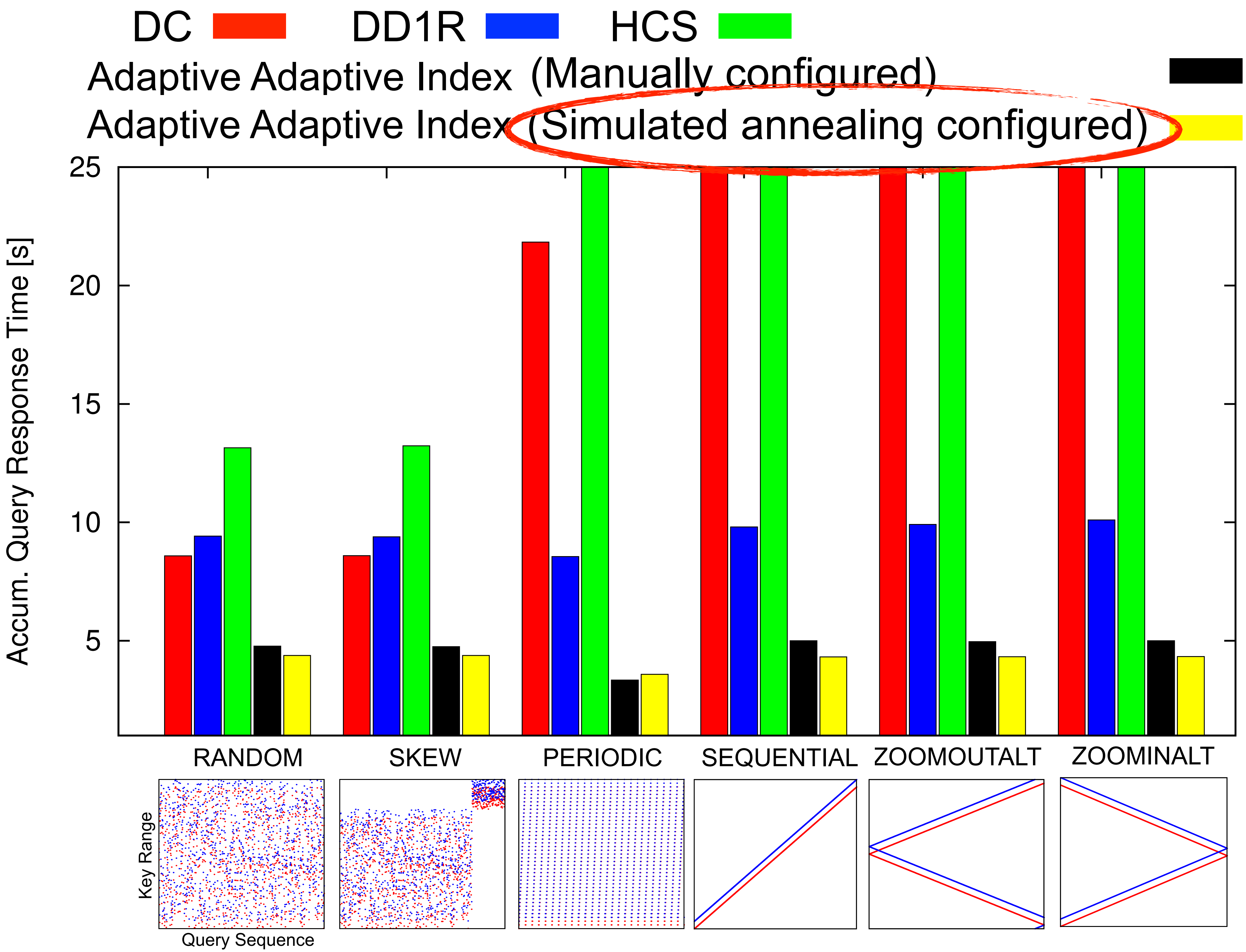
$bmax=6$

$tadapt=64MB$

$tsort=256KB$

Accumulated Query Response Times





$b_{first} = 10$

$b_{min} = 3$

$b_{max} = 6$

$t_{adapt} = 64MB$

$t_{sort} = 256KB$


Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores
Felix Halim, Stratos Idreos, Panagiotis Karras, Roland H. C. Yap
Self-organizing Tuple Reconstruction in Column-stores
Stratos Idreos, Martin L. Kersten, Stefan Manegold
Adaptive Adaptive Indexing
Felix Martin Schuhknecht, Jens Dittrich, Laurent Linden
Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores
Stratos Idreos, Stefan Manegold, Harumi Kuno
Poor Man's Sort!
optimized indexes