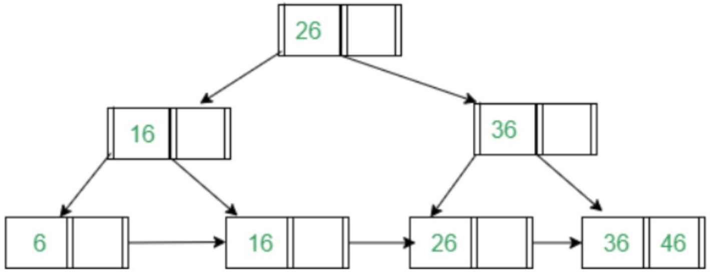
Adaptive Adaptive Indexing

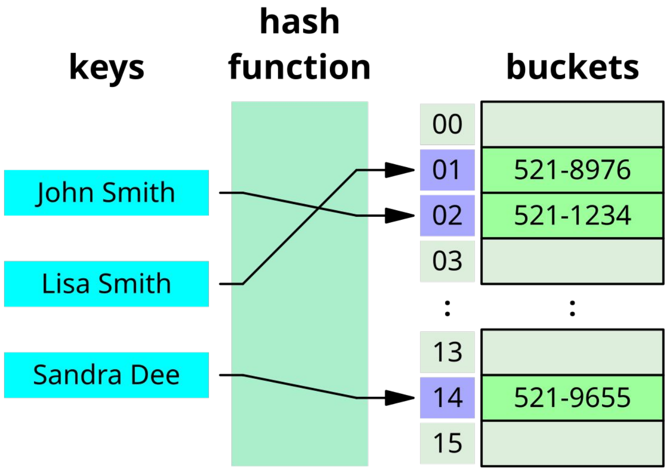
Felix Martin Schuhknecht, Jens Dittrich, Laurent Linden

Arun Shrestha, Parthiv Ganguly, Binyamin Friedman

Indexing

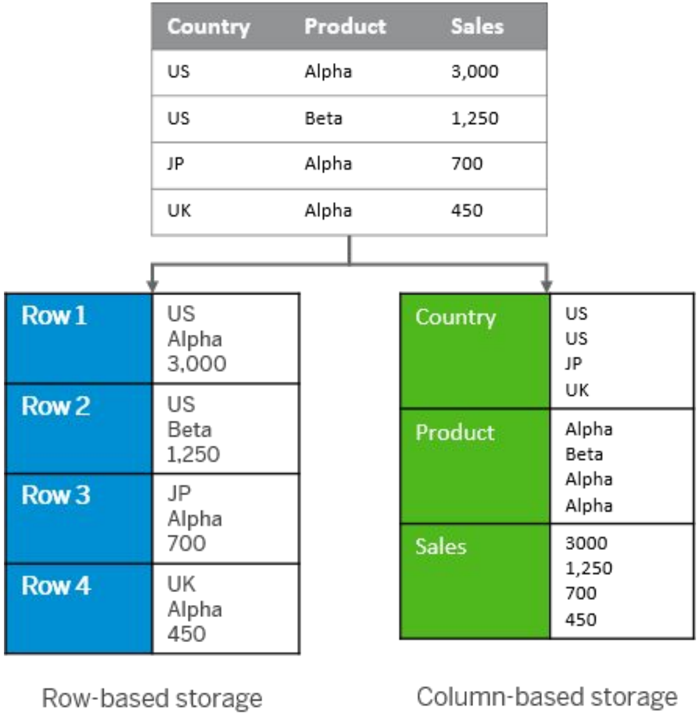


Identifier	Gender	Bitmaps	
		F	M
1	Female	1	0
2	Female	1	1
3	Male	0	1
4	Male	0	1
5	Female	1	0
6	Male	0	1



What do these indexing techniques have in common?

The Need for Adaptive Indexing



- Adaptive Radix Tree (ART)
- Adaptive Merging
- Cracking
- Hybrid Cracking
- Predictive Adaptive Indexing

History of Adaptive Indexing

2000s

Column-store research and Commercial **column-stores** took off

2007

S. Idreos,
M. L. Kersten,
S. Manegold
“Database cracking”
CIDR

2013

Adaptive Radix Tree was presented at ICDE

2018

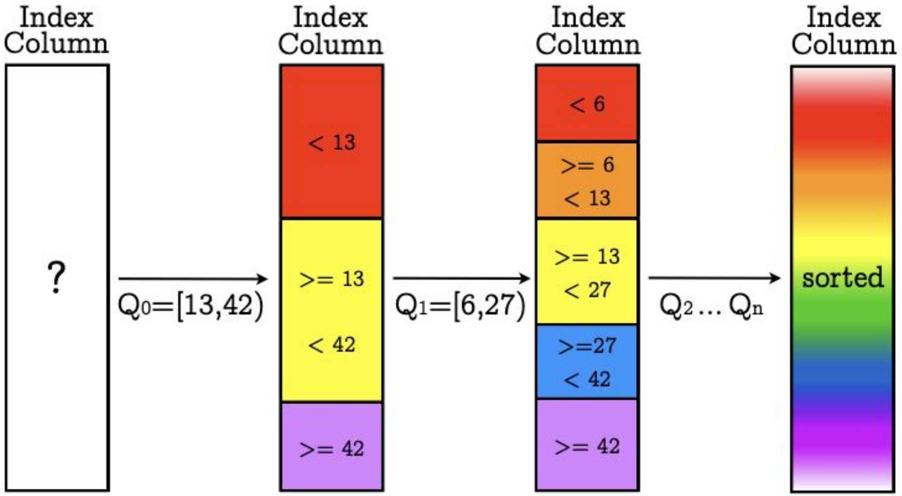
Adaptive Adaptive Indexing was presented and published at ICDE

What is cracking?

$Q_0, Q_1, Q_2, ..., Q_n$

$Q=[low, high)$

Standard cracking



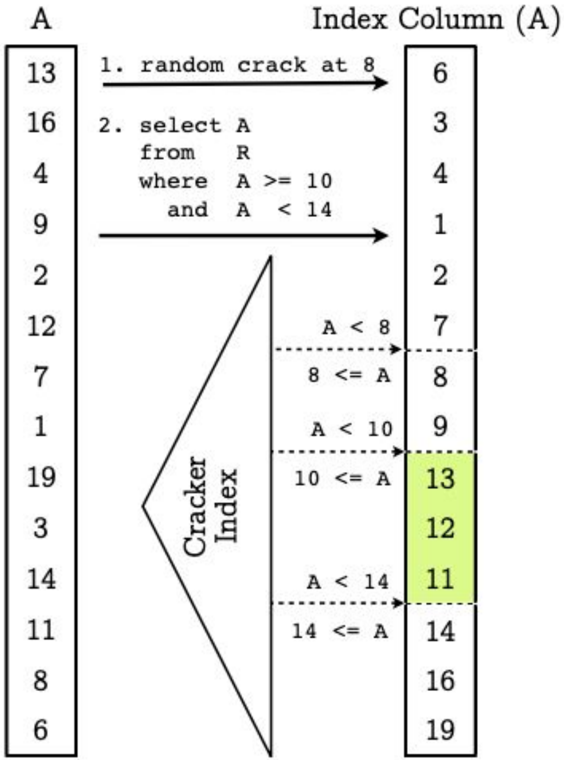
Limitations

Variable query performance

Slow convergence speed

Weak robustness

Stochastic cracking

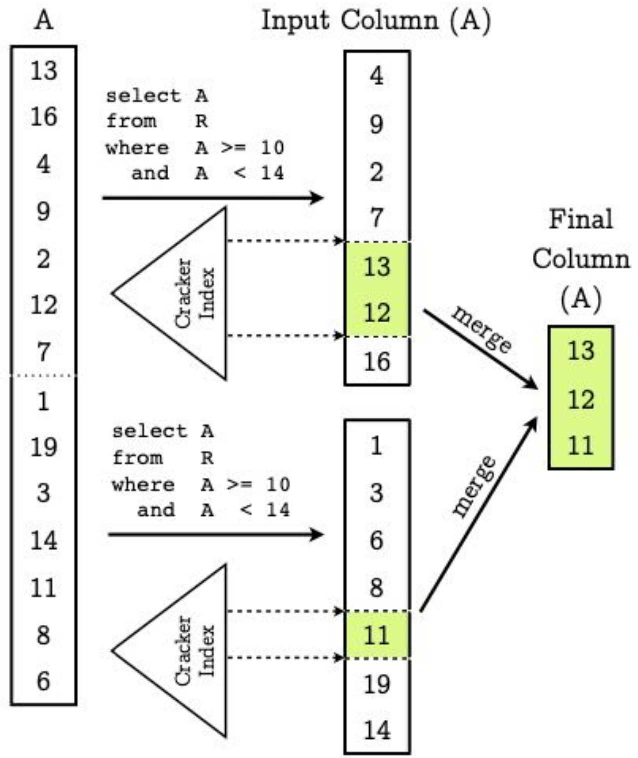


Variable query performance

Slightly better convergence speed

Strong robustness

Hybrid Cracking



Variable query performance

Fast convergence speed

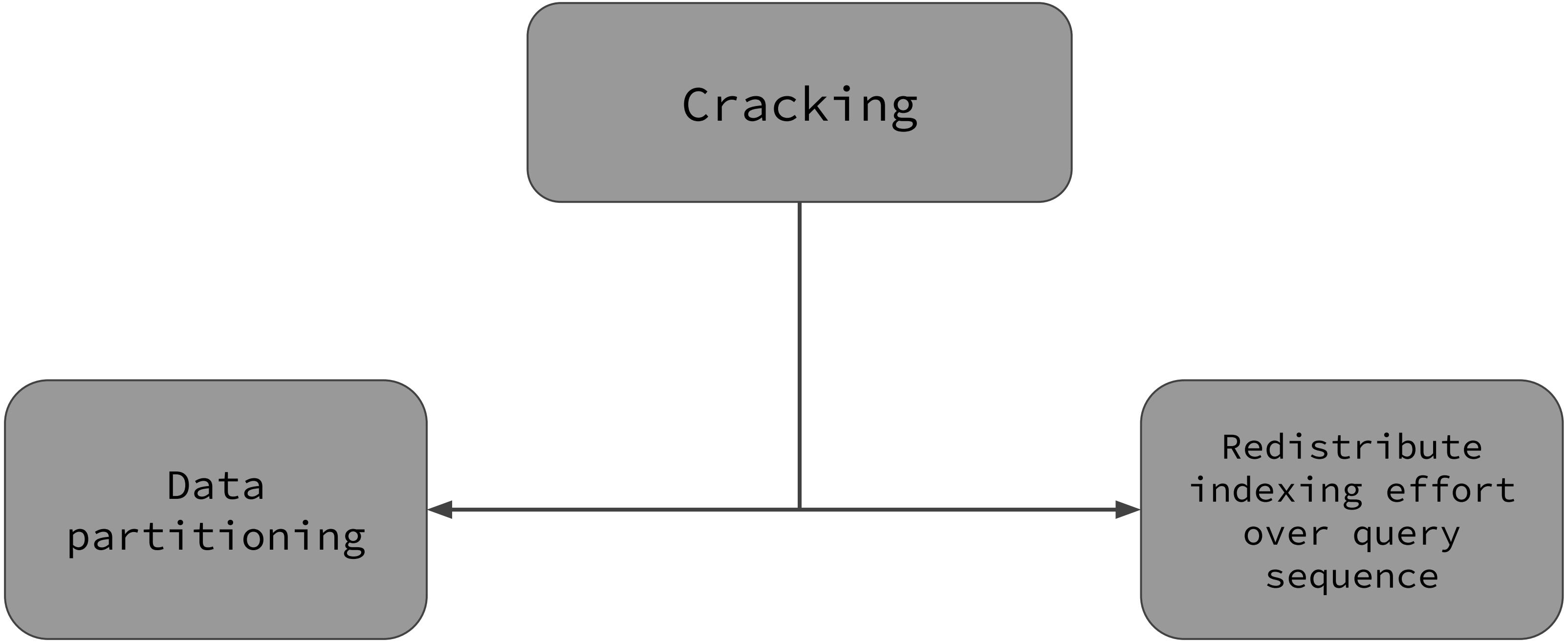
Decent robustness

Cracking Overview

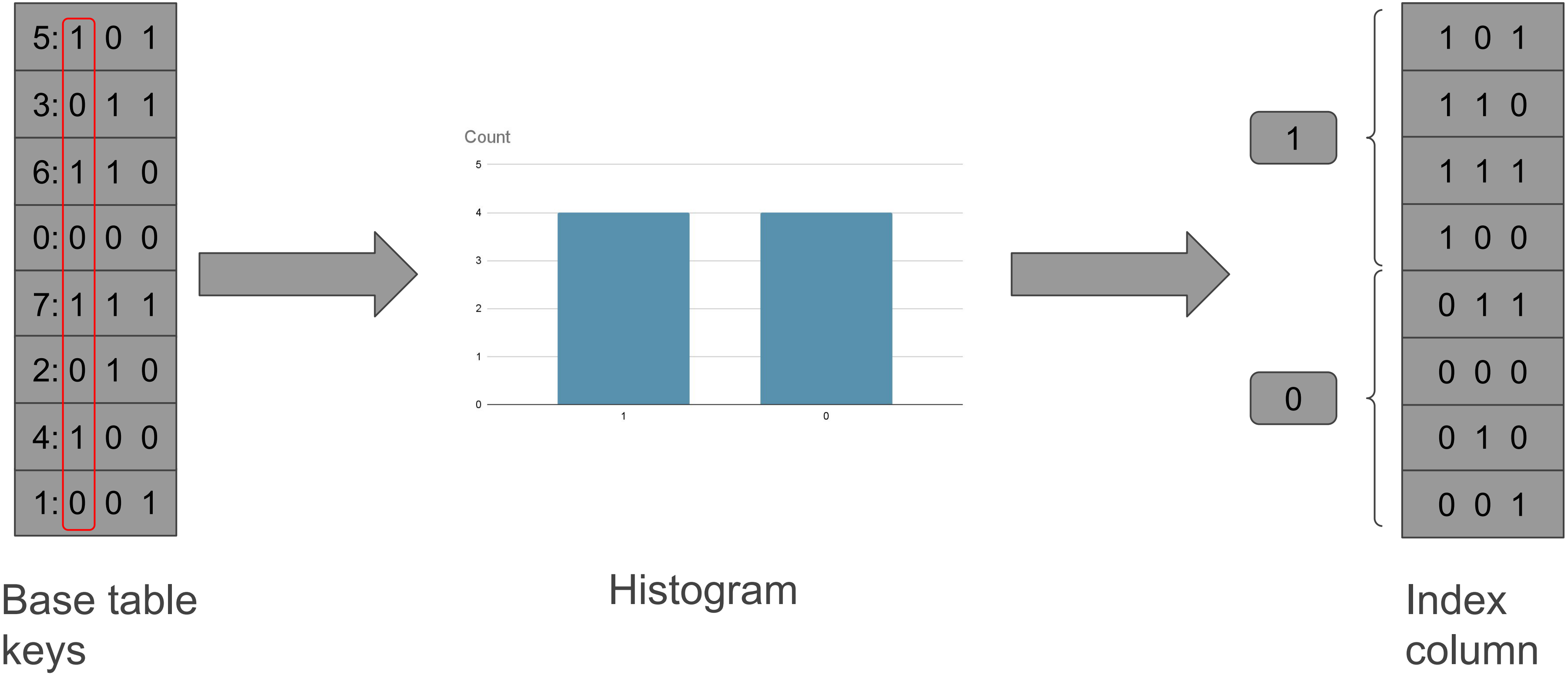
Cracking	Variance in Query Performance	Convergence Speed	Robustness
Standard	High	Slow	Weak
Stochastic	High	Medium	Strong
Hybrid	High	Fast	Medium

Can we get all the benefits in 1 cracking algorithm?

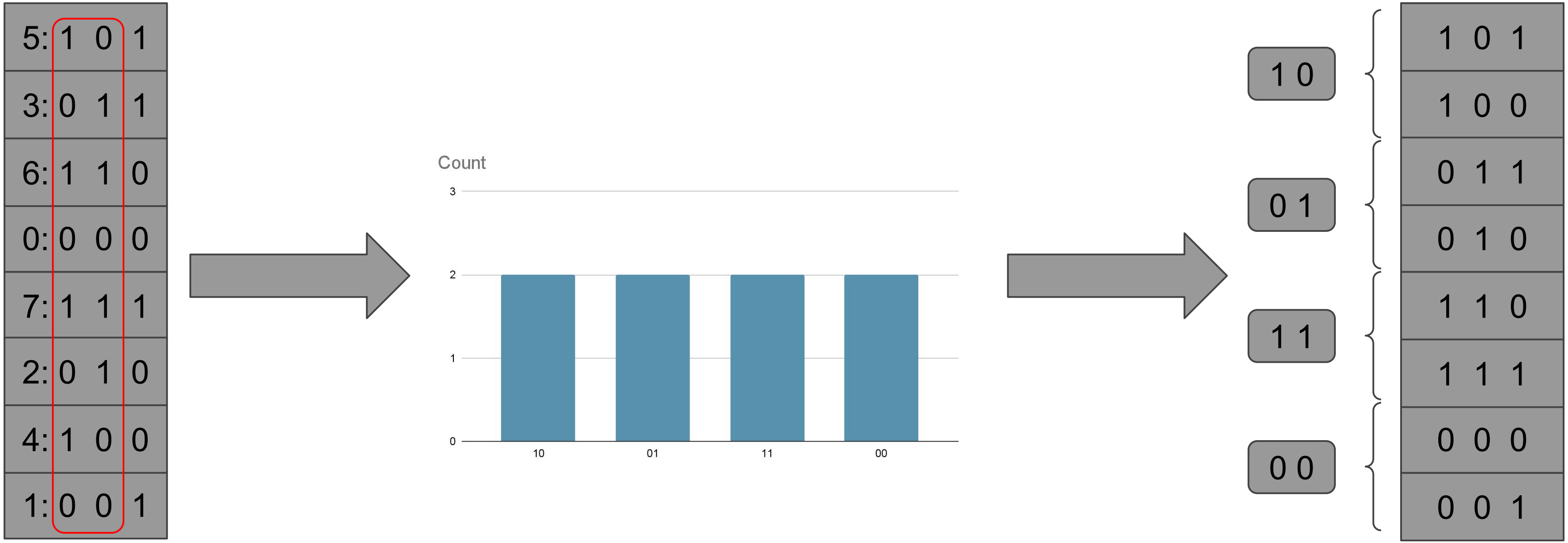
Design Principles of Cracking



Radix based partitioning (1 bit)



Radix based partitioning (2 bits)



Base table keys

Histogram

Index column

Fanout (partition-in-k)

Is there a pattern?

Meta-Adaptive Indexing Strategy

What is Meta-Adaptivity?

Classical Adaptivity	→	Choose k before starting, and every time the partitioning algorithm is used, create k more partitions
Meta Adaptivity	→	Adjust k based on the size of the input partition

How do we adjust k?

For the first query: Set k to a high number and reduce the partition size drastically

How do we adjust k?

For the first query: Set k to a high number and reduce the partition size drastically

For subsequent queries: With a decrease in input partition size, increase the fanout k. If the input partition is small enough, just sort the partition

Issue with Radix Partitioning

Cracking splits the column according to the query predicates, while radix uses the bits of the key.

What issue can this cause when searching for keys in a ranged query using radix?

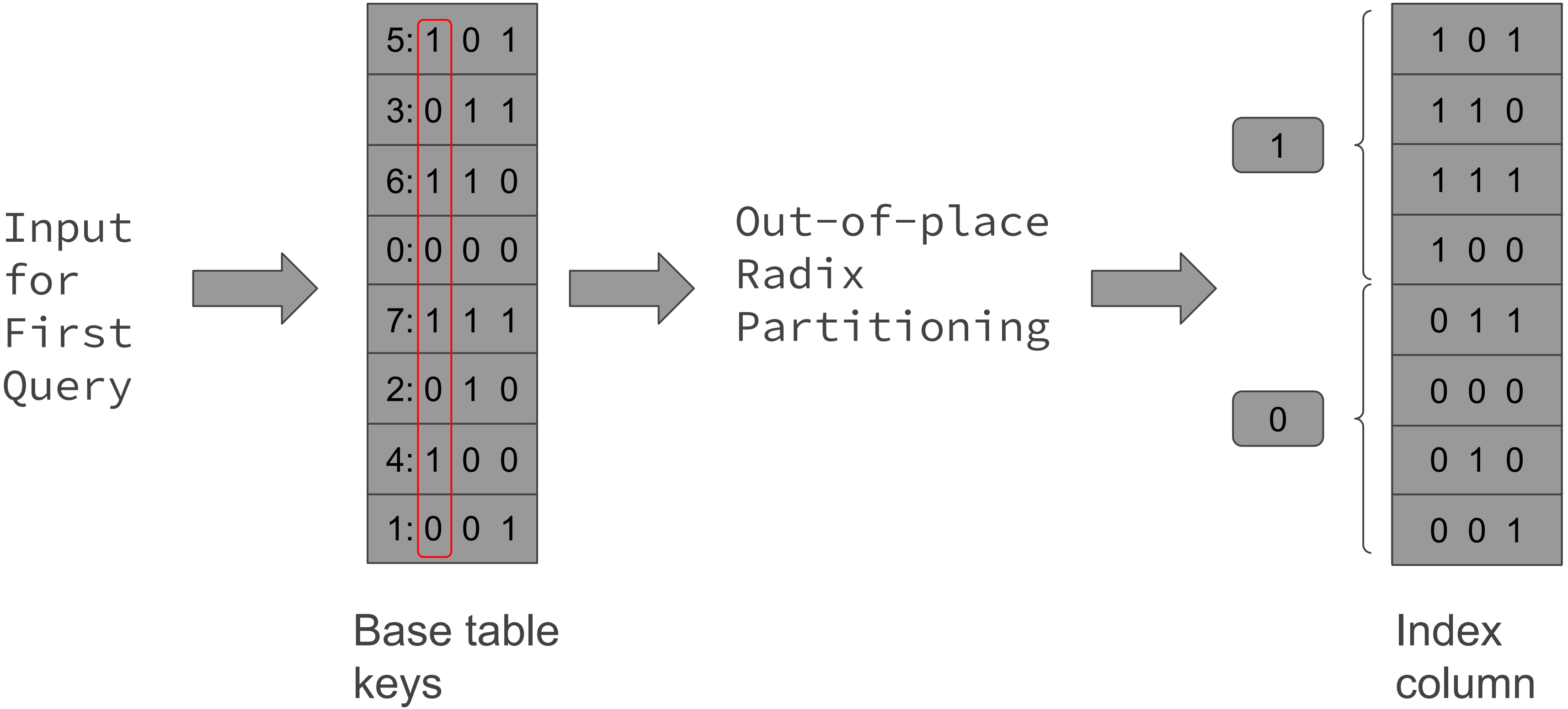
Issue with Radix Partitioning

Cracking splits the column according to the query predicates, while radix uses the bits of the key.

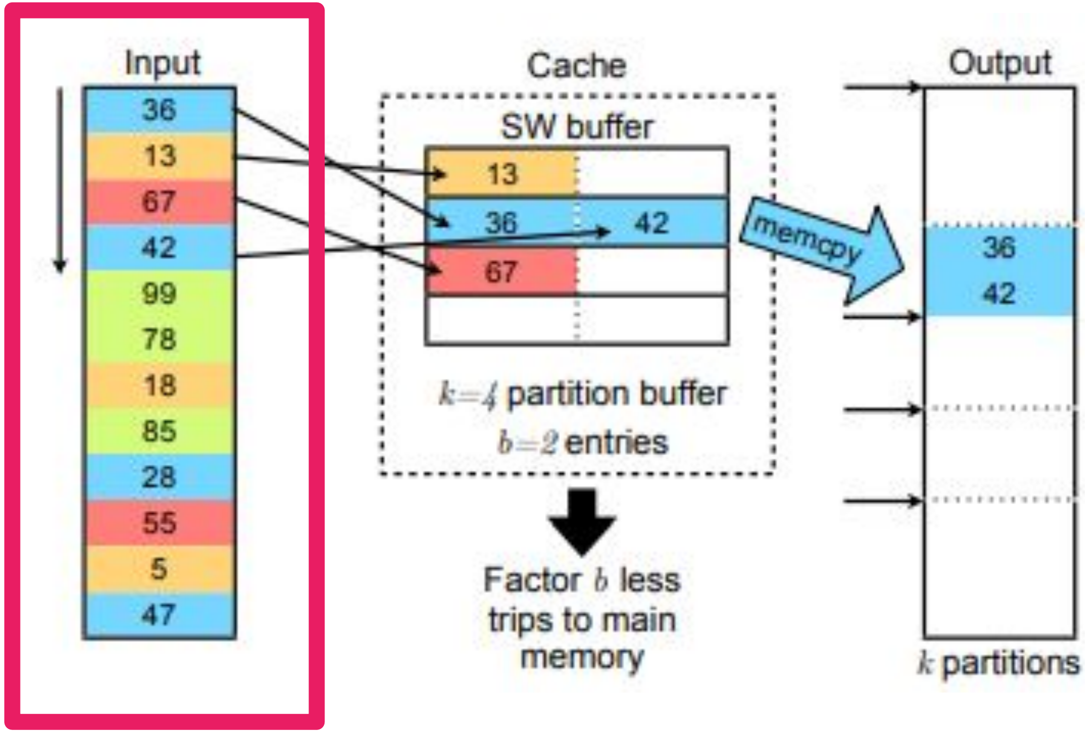
What issue can this cause when searching for keys in a ranged query using radix?

For radix you have to search multiple partitions that may or may not have the key. However, this cost increase is negligible when compared to the benefits of radix

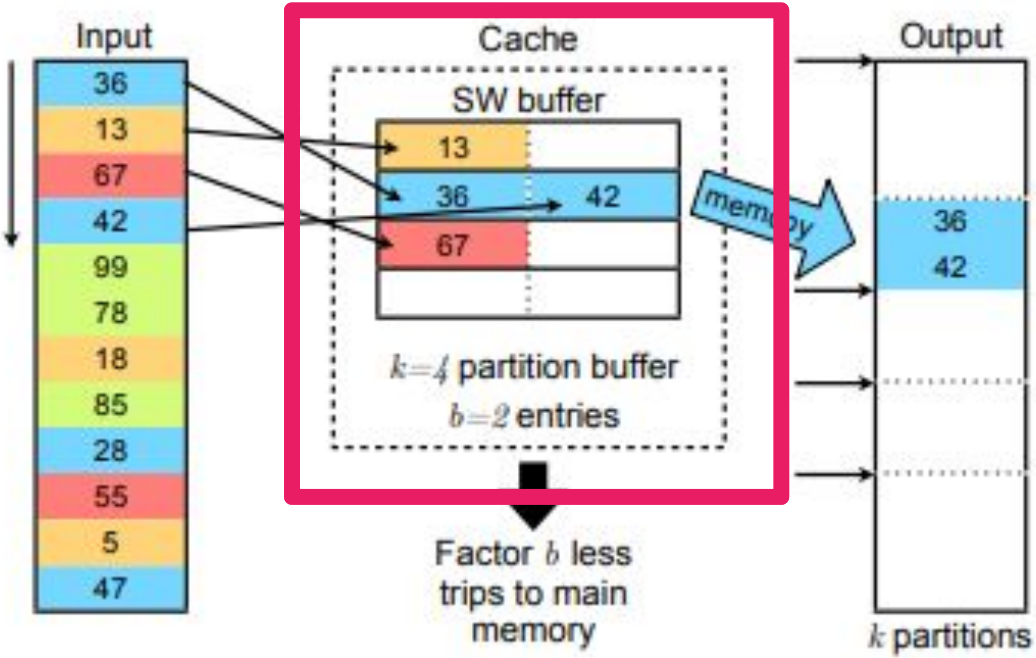
Handling First Query



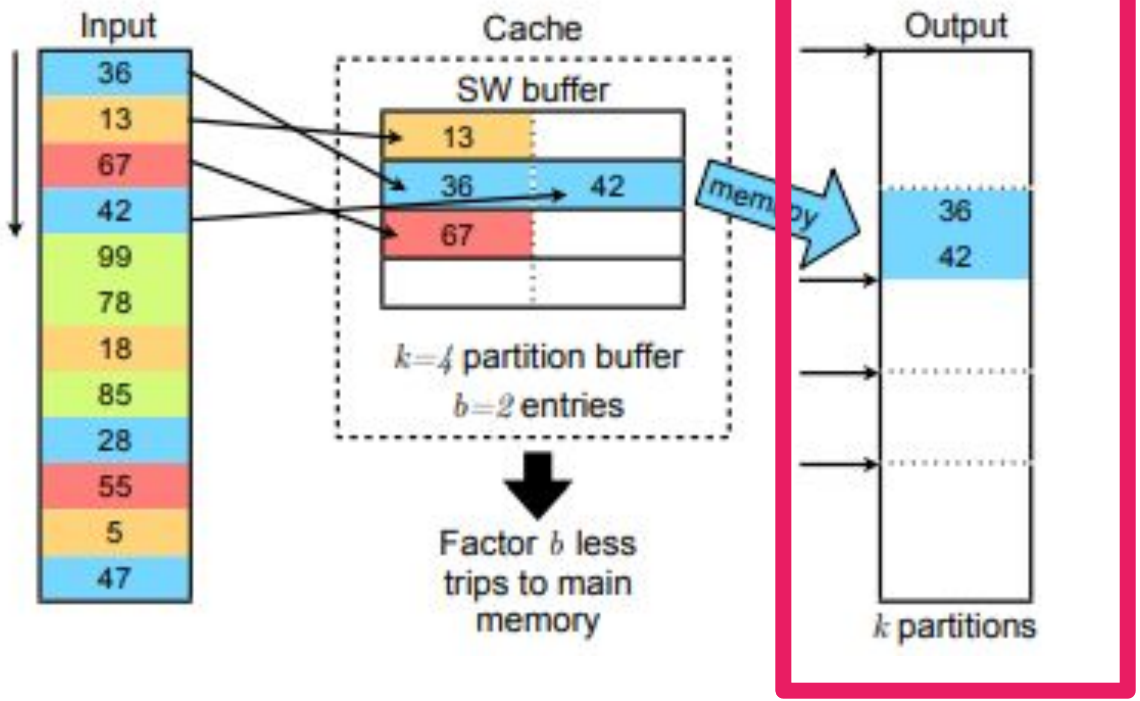
Out-of-place Radix Partitioning w/ SW buffer



Out-of-place Radix Partitioning w/ SW buffer

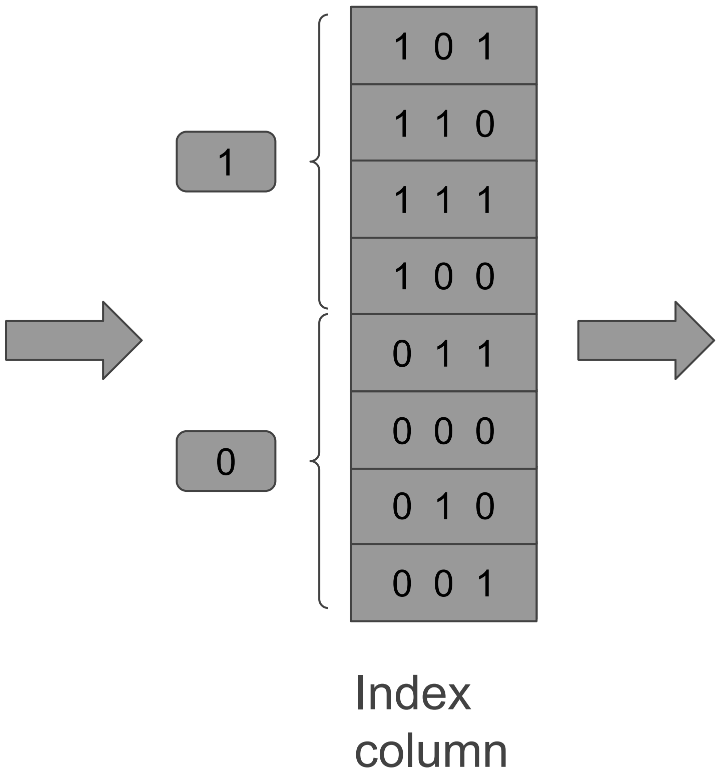


Out-of-place Radix Partitioning w/ SW buffer



Handling Subsequent Queries

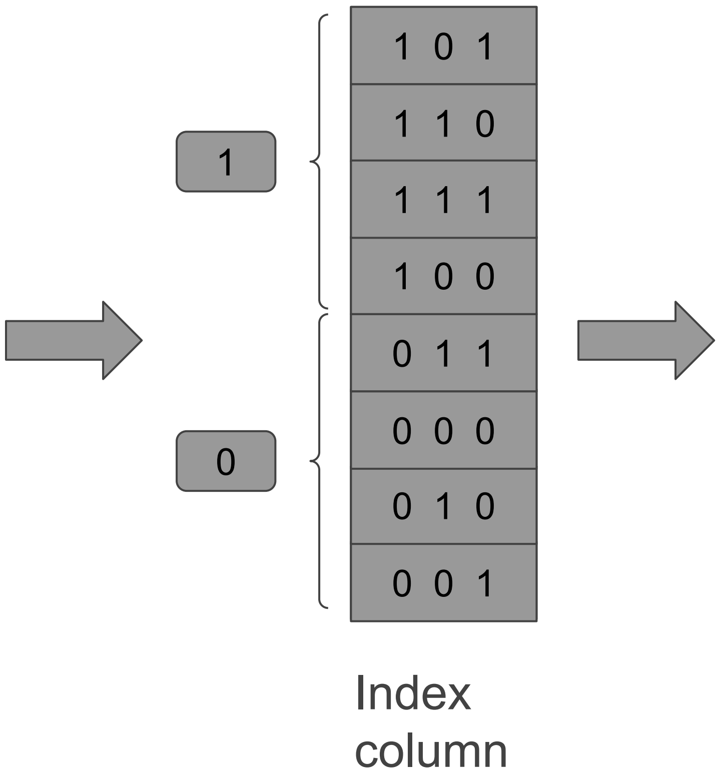
Input for Subsequent Queries



Can we still do out-of-place? If not, why?

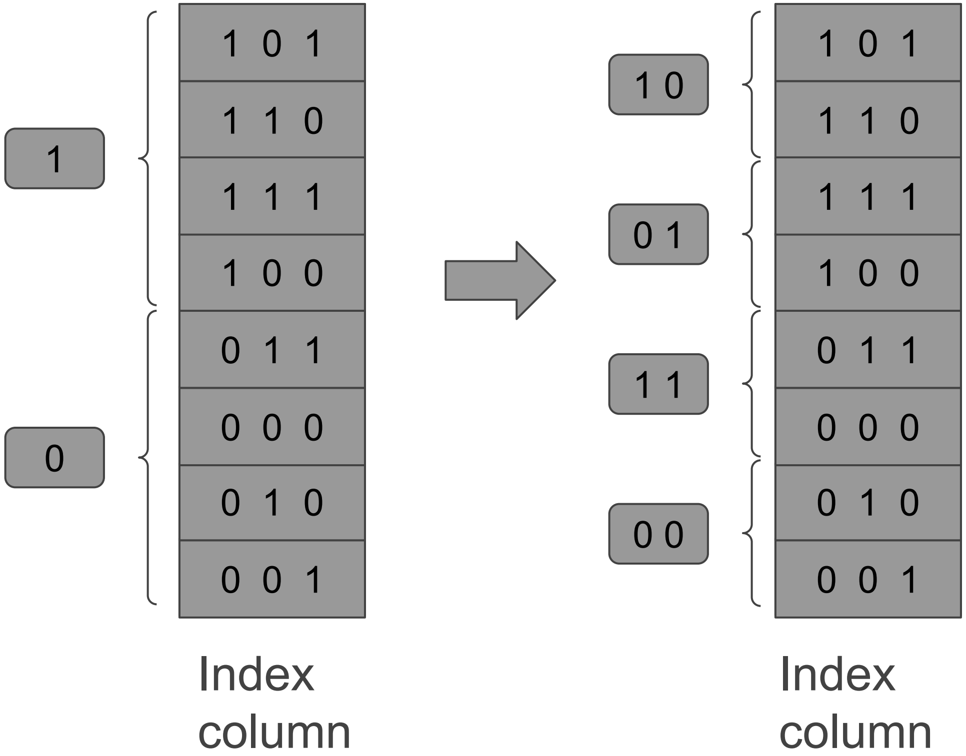
Handling Subsequent Queries

Input for Subsequent Queries

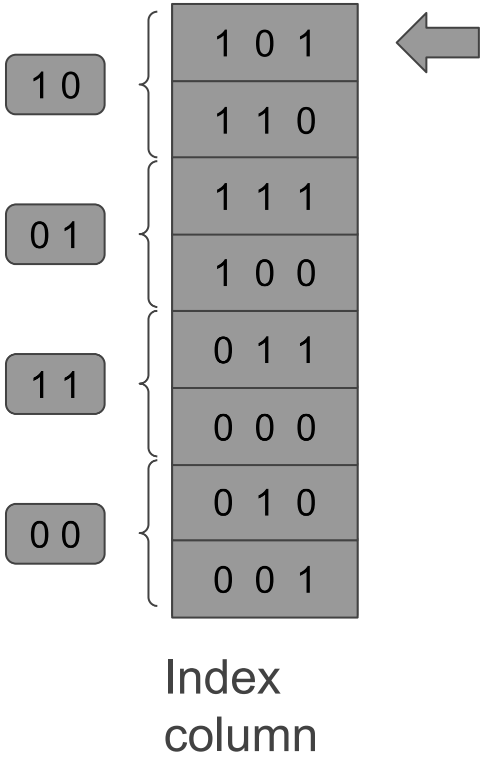


Use In-place Radix Partitioning

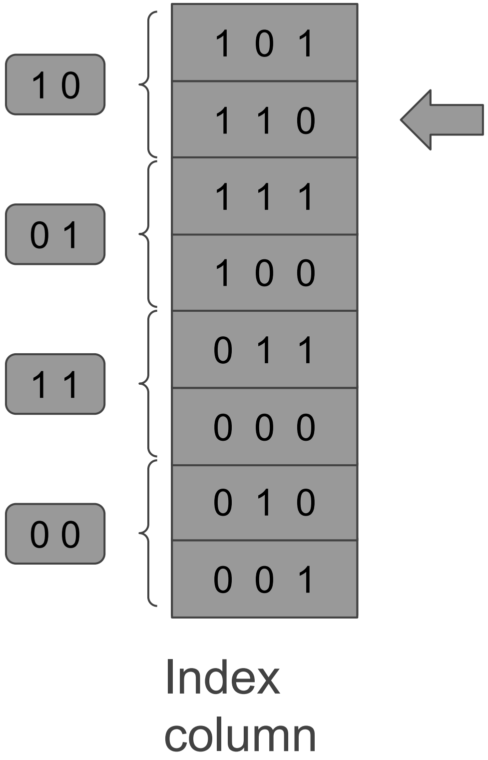
In-place Radix Partitioning



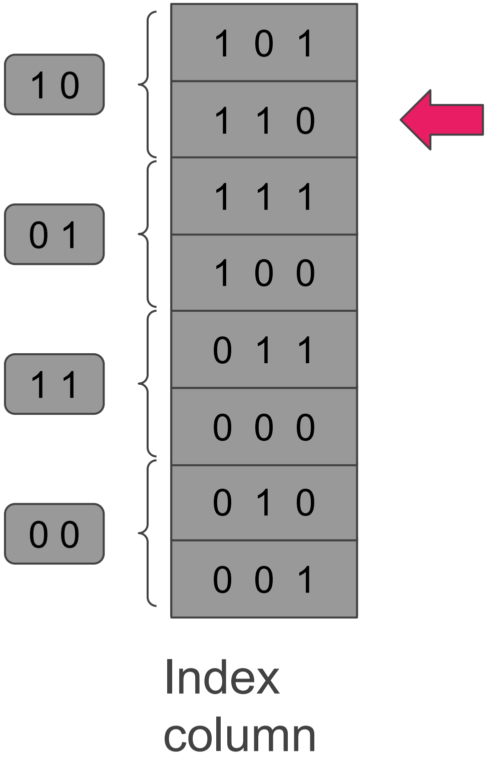
In-place Radix Partitioning



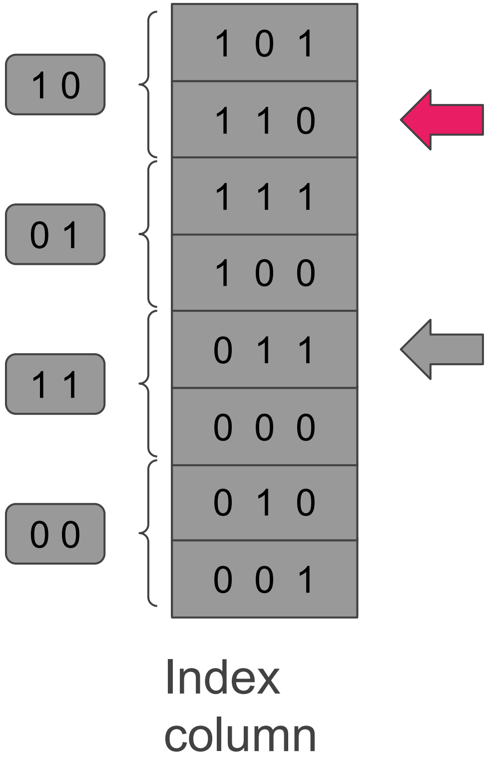
In-place Radix Partitioning



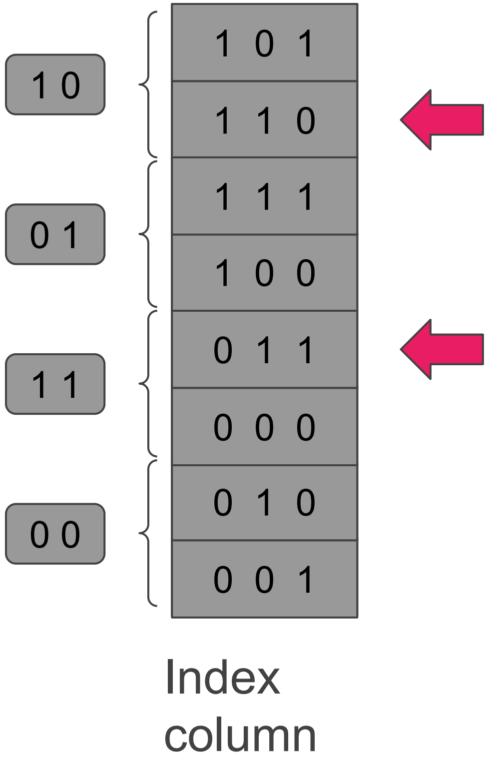
In-place Radix Partitioning



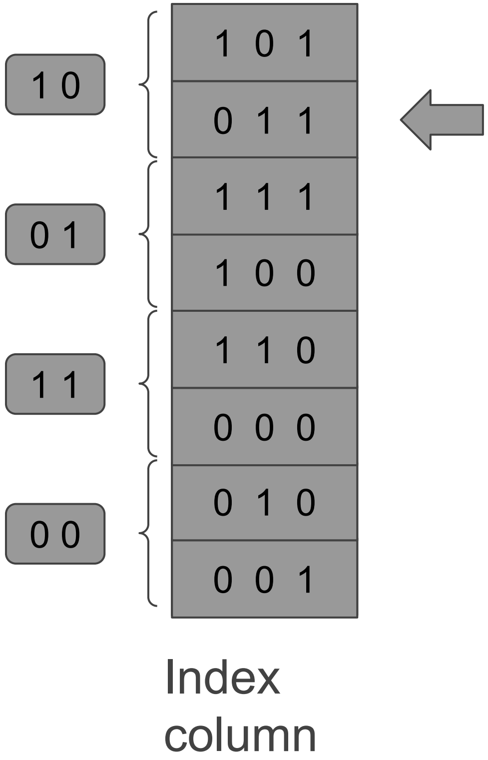
In-place Radix Partitioning



In-place Radix Partitioning

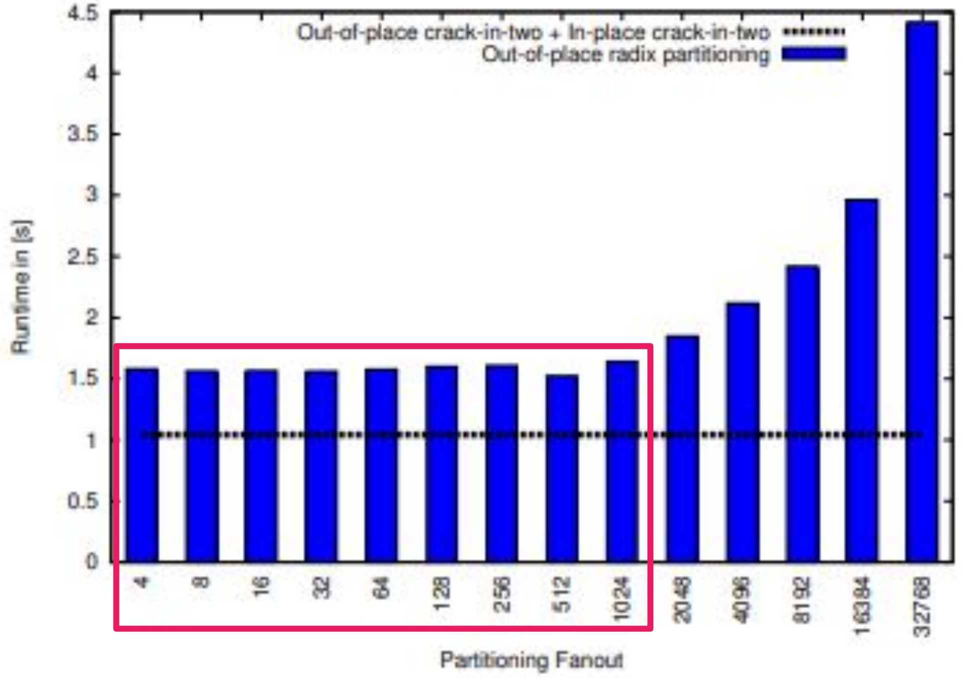


In-place Radix Partitioning



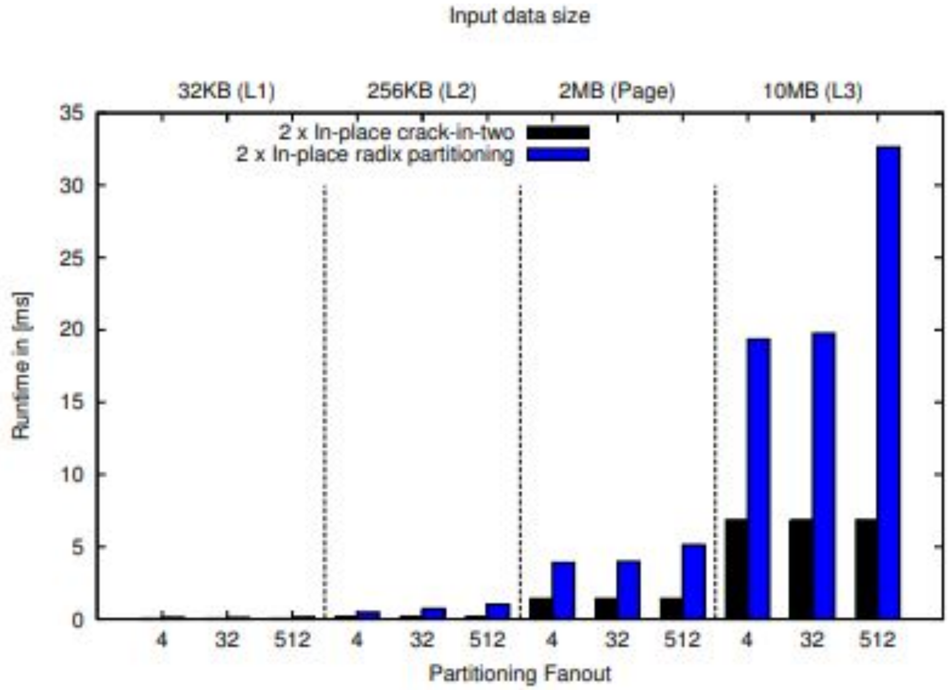
Evaluation of Radix vs Crack-in-2 partitioning

Key Takeaway: We can set k to a very high value (1024) and runtime cost increase will be minimal



Evaluation of Radix vs Crack-in-2 partitioning

Key Takeaway: As input partition size increases, the additional runtime cost of setting a higher k also increases.



Defining the Adaptive Fanout Function

The adaptive fanout function f(s,q) will take the input partition size (s) and query sequence number (q) as inputs, and output the number of fanout bits.

Defining the Adaptive Fanout Function

The adaptive fanout function f(s,q) will take the input partition size (s) and query sequence number (q) as inputs, and output the number of fanout bits.

What predefined values or thresholds do we need before we mathematically define the function?

Adapting Fanout Function

$$b_{first} = \text{number of fanout bits for first query}$$

$$f(s,q) = \begin{cases} b_{\text{first}} & \text{if } q = 0 \\ & \\ & \end{cases}$$

Adapting Fanout Function

$$t_{adapt} = \text{threshold below which fanout adaption starts}$$

$$b_{min} = \text{minimal number of fanout bits during adaption}$$

$$f(s, q) = \begin{cases} b_{\text{first}} & \text{if } q = 0 \\ b_{\text{min}} & \text{else if } s > t_{\text{adapt}} \\ & \\ & \end{cases}$$

Adapting Fanout Function

$$t_{sort} = \text{threshold below which sorting is triggered}$$

$$b_{max} = \text{maximal number of fanout bits during adaption}$$

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

Adapting Fanout Function

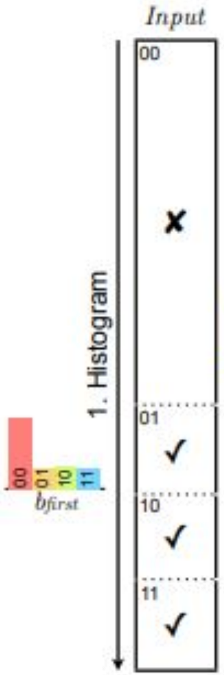
$$b_{sort} = \text{number of fanout bits required for sorting}$$

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

Input Skew

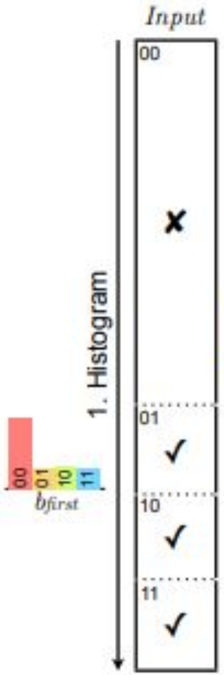
What is the problem with a scenario like this?

And how would you solve it?



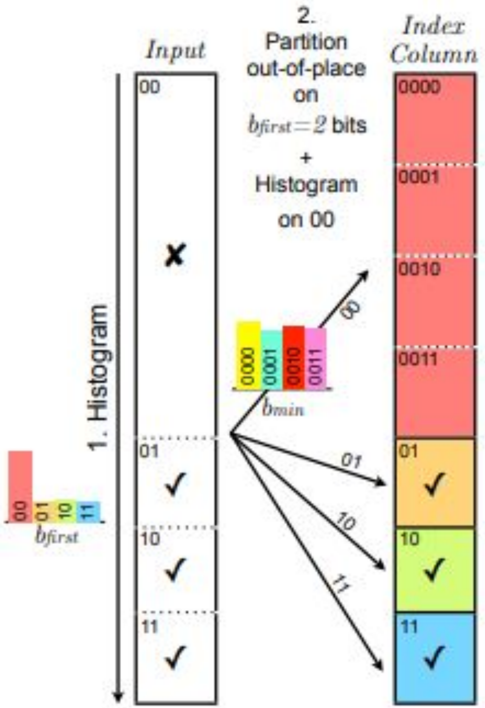
Diffusing Input Skew

If an output partition is greater than a threshold, it is marked for further partitioning



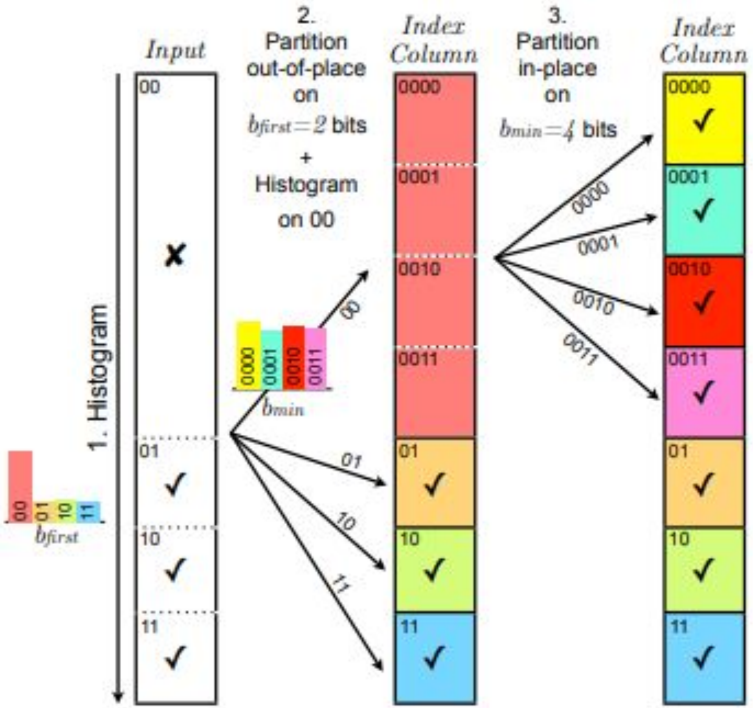
Diffusing Input Skew

A histogram is built for each skewed partition as the keys are being transferred



Diffusing Input Skew

Recursively partition each skewed partition until all of the partitions are below the threshold



Summary of Meta-Adaptivity

Meta-Adaptivity adjusts partitioning fanout based on input partition size

It uses Radix Partitioning which gives us higher throughput and faster convergence for minimal cost

Input skew is diffused using recursive partitioning

Experiments

Baselines

Standard cracking	• Great under uniform random workloads

Review: What limitations does standard cracking have?

Baselines

Standard cracking	• Great under uniform random workloads • Suffers from sequential workloads
Stochastic cracking	• Introduces randomness and decouples partitioning from queries

Baselines

Standard cracking	• Great under uniform random workloads • Suffers from sequential workloads
Stochastic cracking	• Introduces randomness and decouples partitioning from queries
Hybrid cracking	• A class of techniques aiming to improve convergence

Baselines

Standard cracking	• Great under uniform random workloads • Suffers from sequential workloads
Stochastic cracking	• Introduces randomness and decouples partitioning from queries
Hybrid cracking	• A class of techniques aiming to improve convergence
Sort + Search **Scan**	• Extreme cases • Full sorting and no sorting

Yes!

Can the meta-adaptive index emulate our baselines?

Emulation (1/3)

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

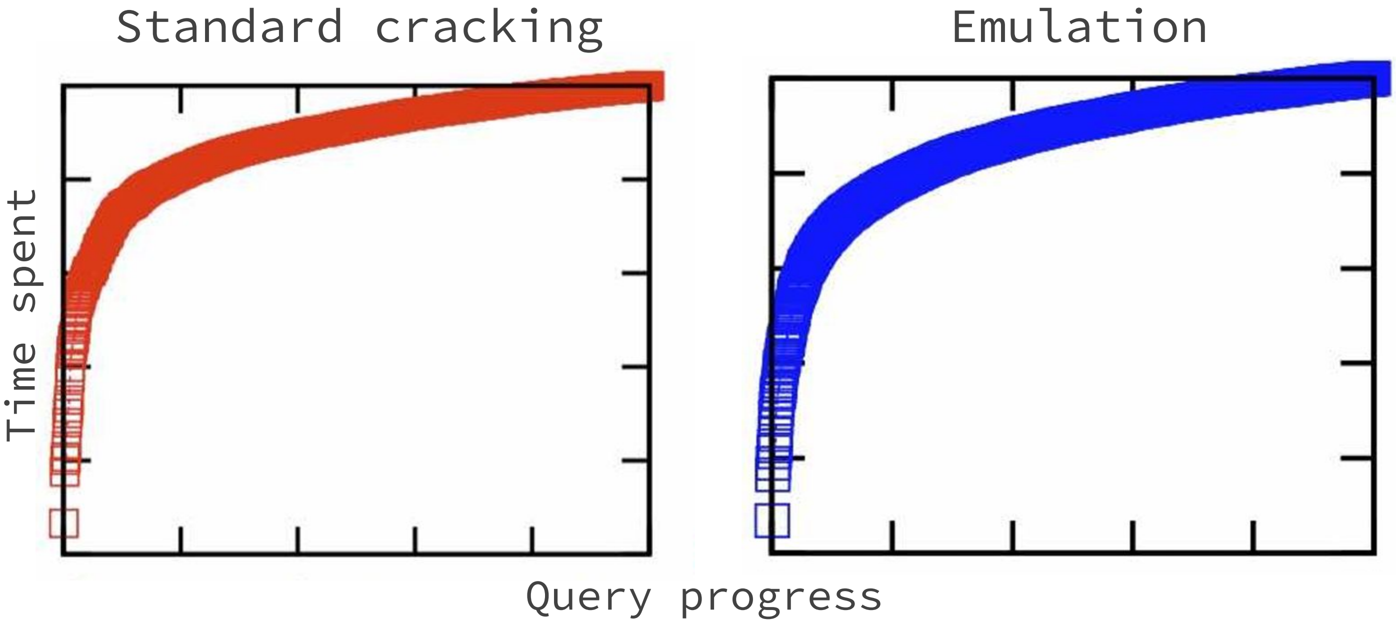
$b_{first} = 1$

$b_{min} = 1$

$b_{max} = 1$

$t_{adapt} = 0$

$t_{sort} = 0$



How do we emulate standard cracking?

Emulation (2/3)

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

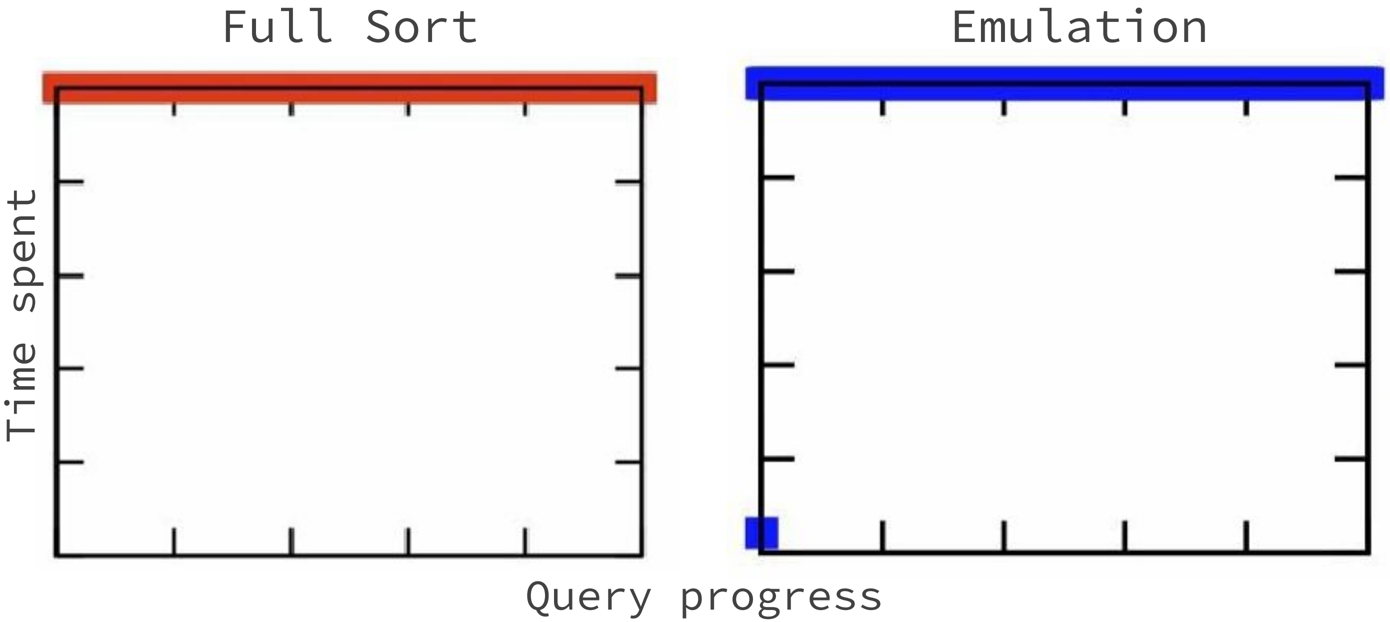
$b_{first} = 0$

$b_{min} = 0$

$b_{max} = 0$

$t_{adapt} = 0$

$t_{sort} = 100\%$



How do we emulate a fully sorted index?

Emulation (3/3)

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

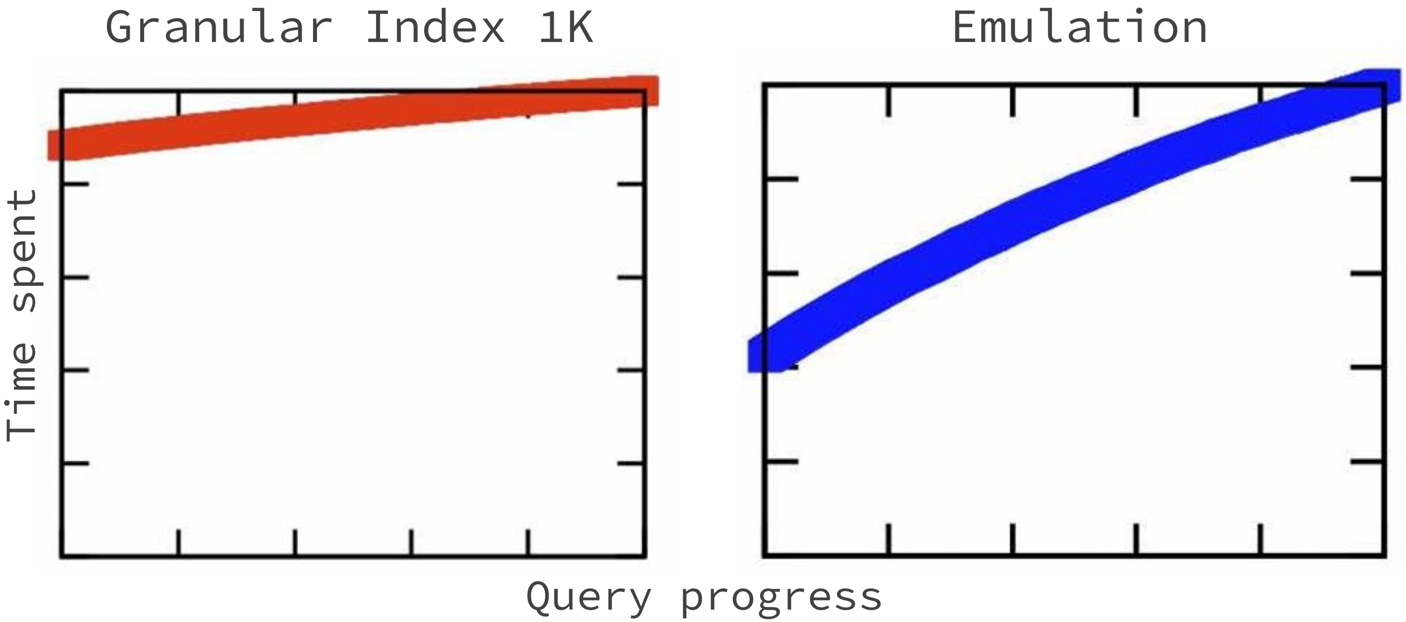
$b_{first} = 10$

$b_{min} = 1$

$b_{max} = 1$

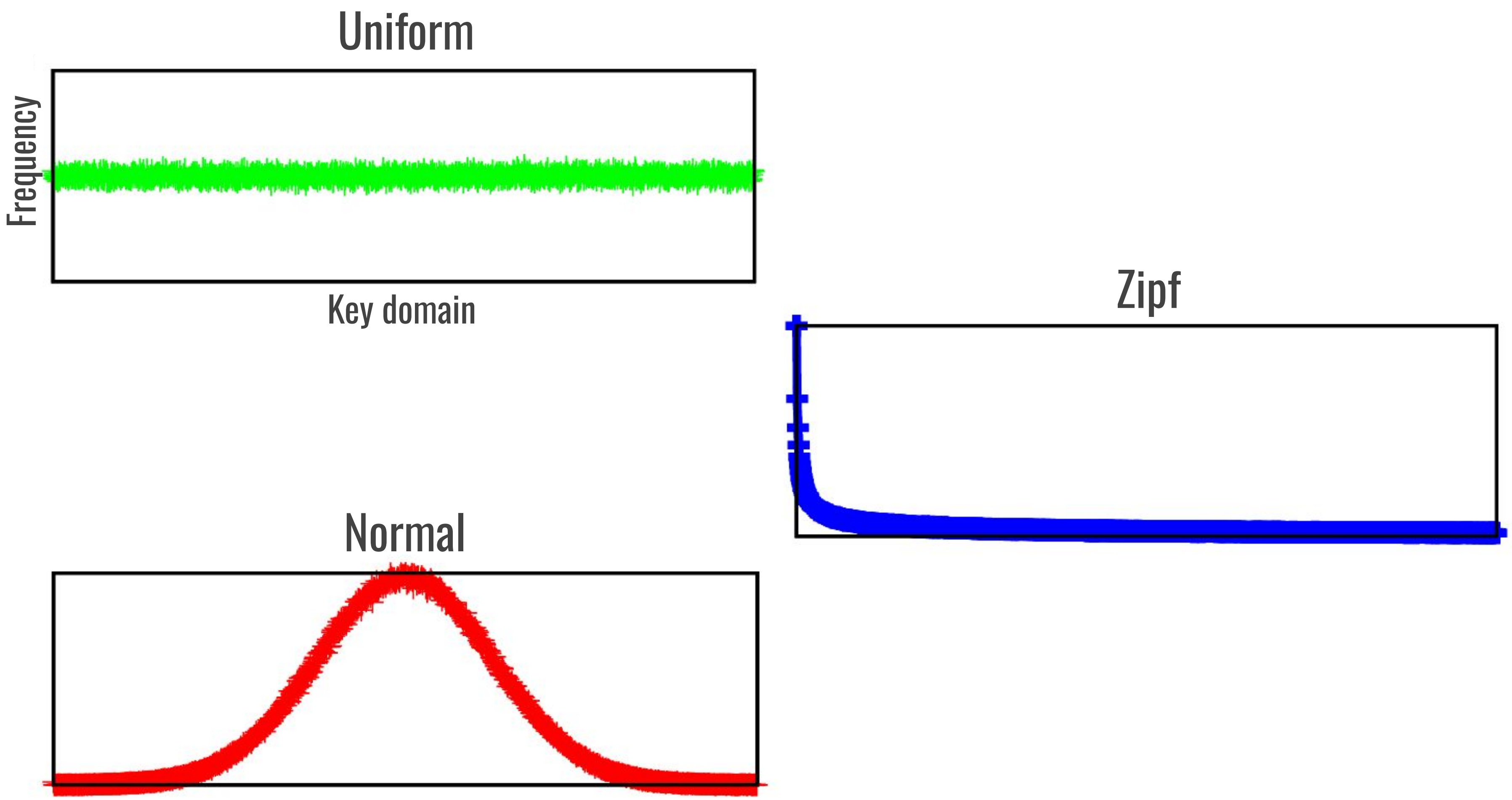
$t_{adapt} = 0$

$t_{sort} = 0$



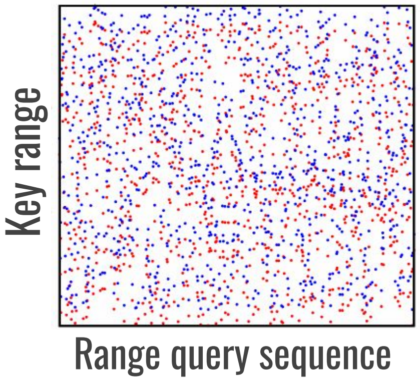
What if we want to include 1024 partitions initially?

Testing Response Times


Uniform
Frequency
Key domain
Zipf
Normal

Upper bound
Lower bound

Random



Uniform

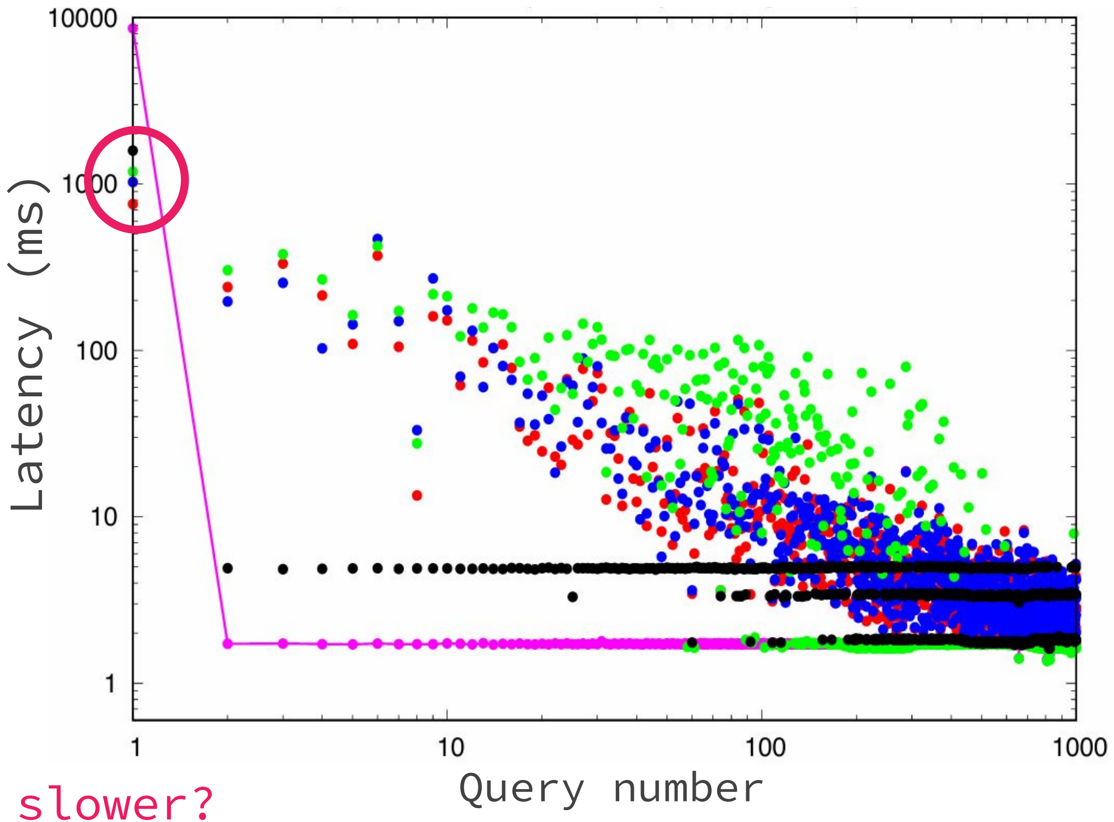
Standard cracking

Stochastic cracking

Hybrid crack-sort

Full sort

Meta-adaptive (manual)



Is meta-adaptive slower?

Zipf

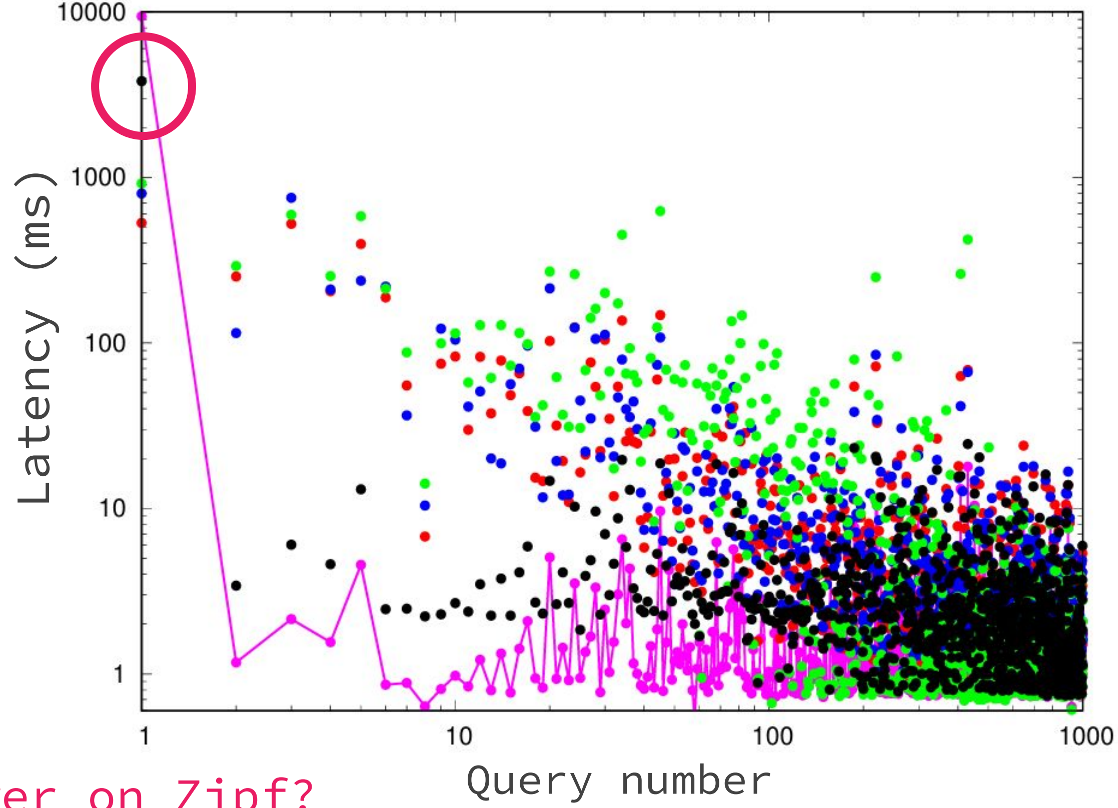
Standard cracking

Stochastic cracking

Hybrid crack-sort

Full sort

Meta-adaptive (manual)



Why is it slower on Zipf?

Normal

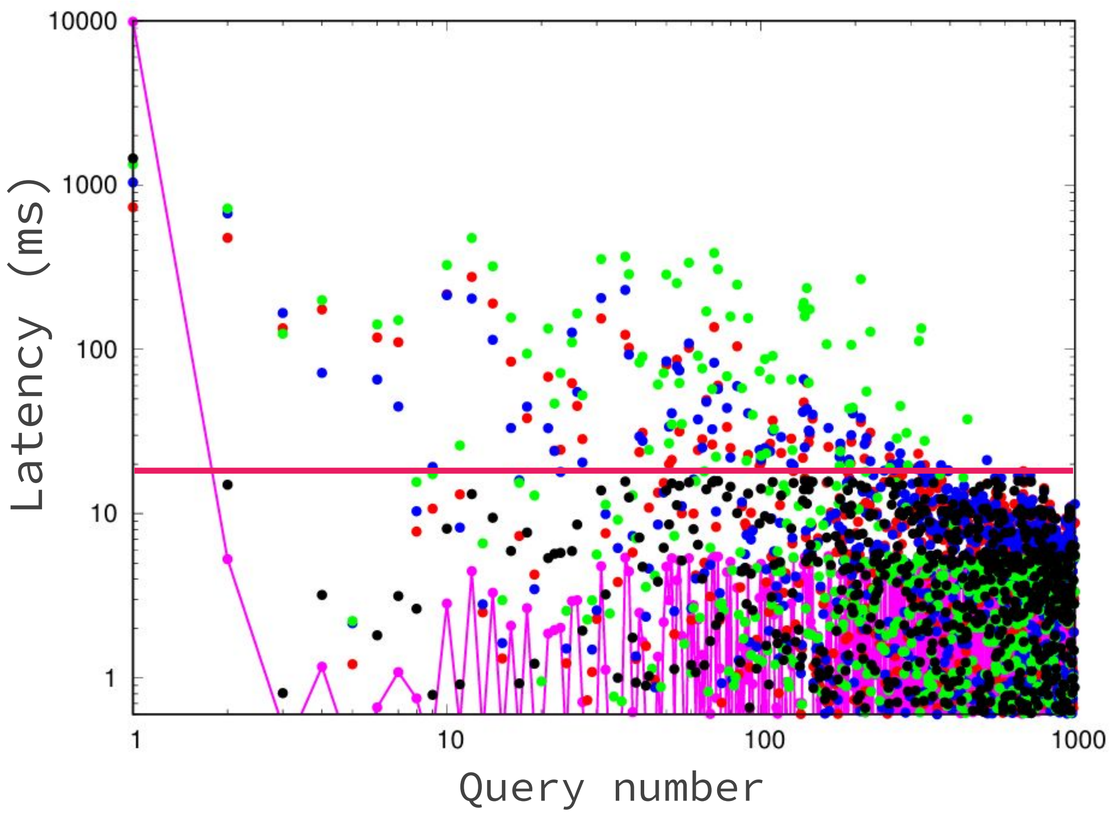
Standard cracking

Stochastic cracking

Hybrid crack-sort

Full sort

Meta-adaptive (manual)



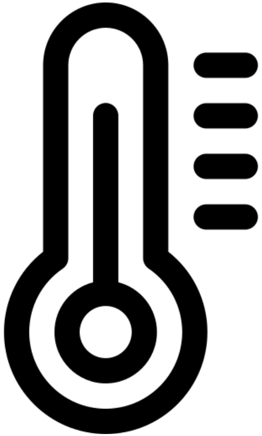
Tuning

`How do we tune our parameters?`

Simulated Annealing!

Simulated Annealing

Simulated annealing approximates global optimum through a stochastic procedure

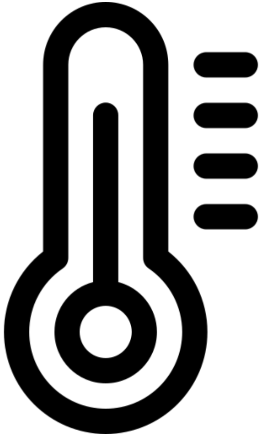


What does annealing refer to in “real” life?

Simulated Annealing

Accept a new configurati
probability $e^{-dQRT/temp}$

Decrease temperature over ti



Can you think of a limitation of this method?

Simulated Annealing

Parameter	Uniform	Normal	Zipf
b_{first}	12 bits	10	5
b_{min}	2 bits	1	3
b_{max}	5 bits	5	5
t_{adapt}	218MB	102	211
t_{sort}	354KB	32	32
skewtol	4x	5	5

Cumulative Latency

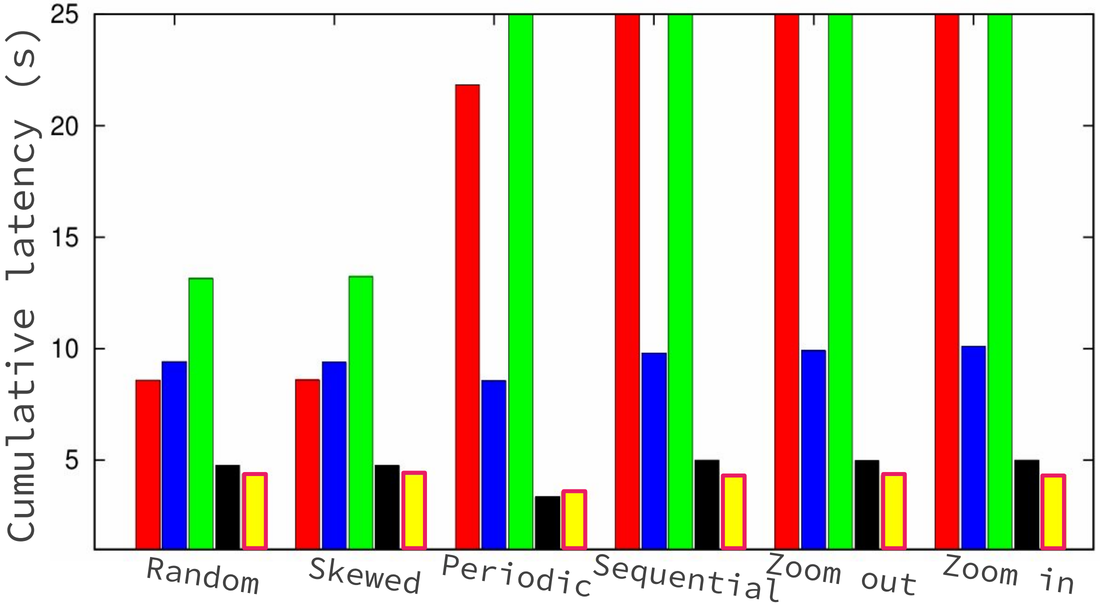
Standard cracking

Stochastic cracking

Hybrid crack-sort

Meta-adaptive (manual)

Meta-adaptive (auto)



Conclusion

Fanout in k is a versatile enough mechanism to emulate other cracking algorithms

The “meta-adaptive” index performs better than alternative cracking algorithms by better distributing its efforts

Commentary

What we think

Binyamin: I do not think “meta-adaptive” is a good characterization of their technique. In addition, there could have been more index comparisons.

Arun: The paper does a great job generalizing various cracking methods, but the title is misleading as it does not encompass all adaptive indexing techniques.

Parthiv: The paper mentions input variance in the beginning and that Adaptive Adaptive Indexing will be better on it, but this is not explicitly backed up during the mathematical analysis and experiment section.

Thank you! Questions?