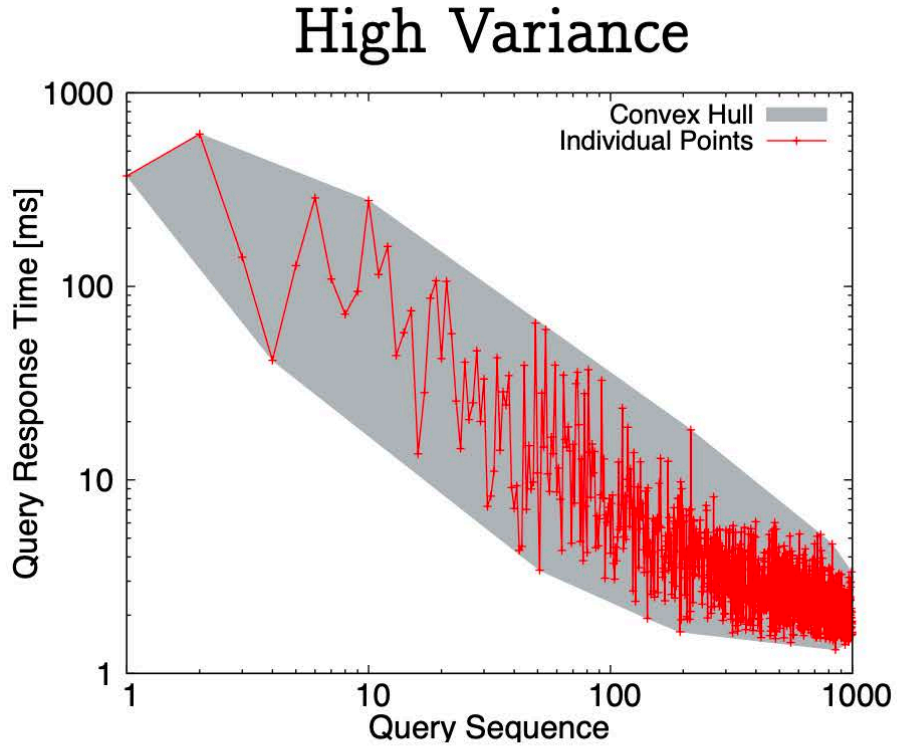
ADAPTIVE ADAPTIVE INDEXING

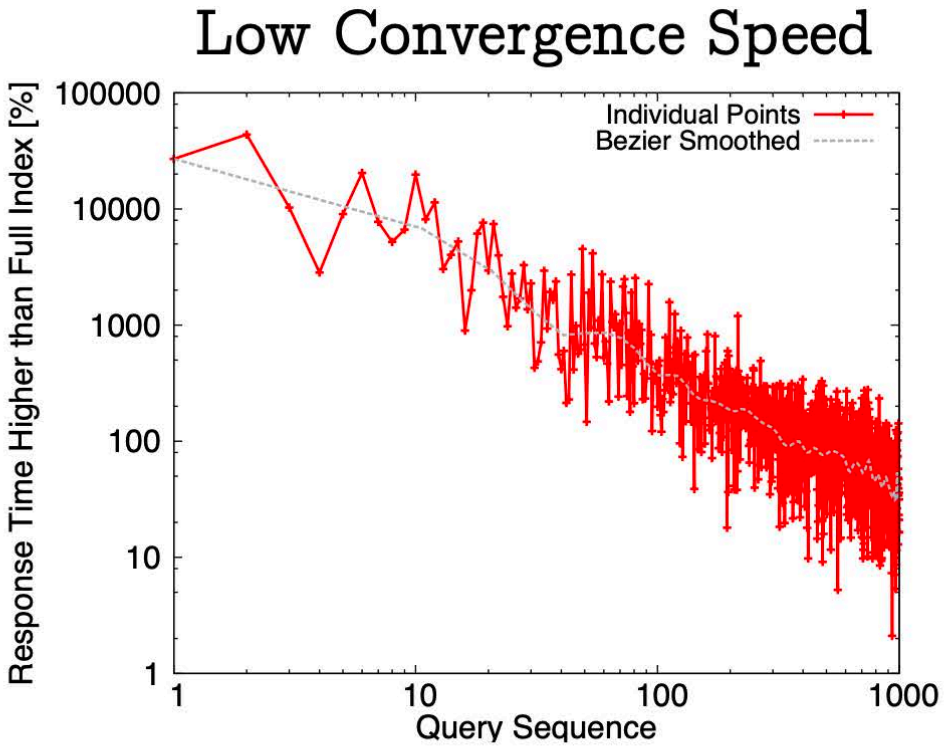
FELIX MARTIN SCHUHKNECHT, JENS DITTRICH, LAURENT LINDEN

ALGORITHMS COME AND GO

The Dodo Bird, a flightless bird with no natural predators that went extinct circa 1681 after discovered by humans as a source of meat in the 15th century

CURRENT STATE OF THE ART (INDEXING)

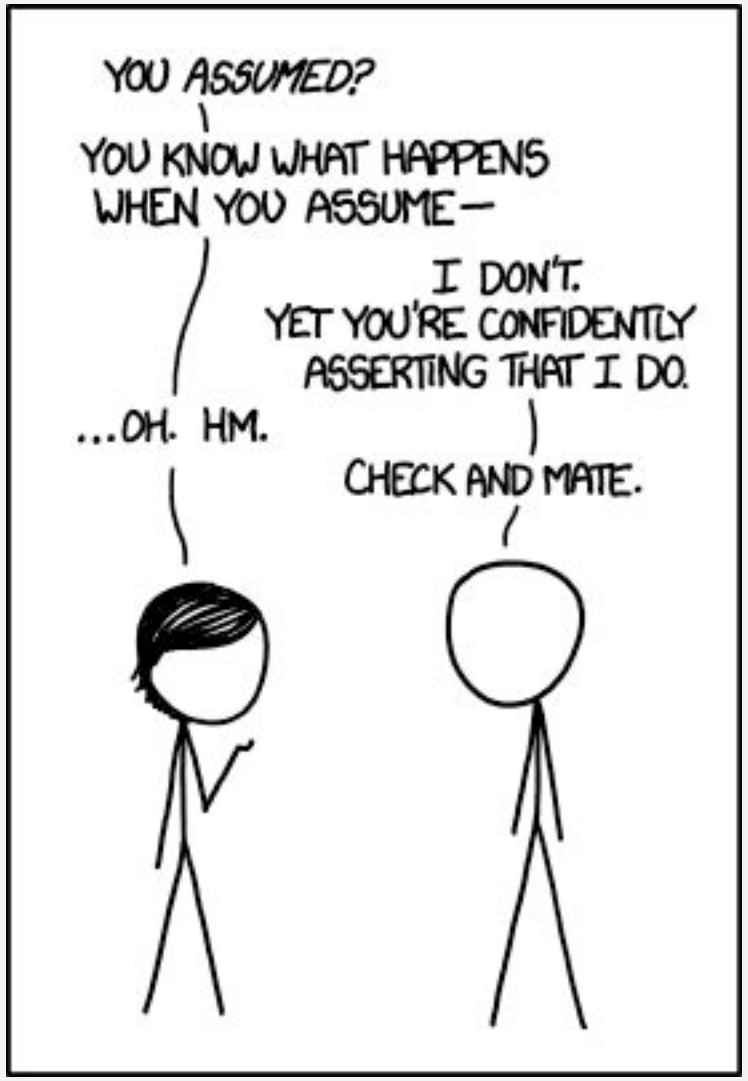




INTRODUCING ADAPTIVE ADAPTIVITY

FIRST GENERAL PRINCIPLE

- Make no assumptions



RATIONALE

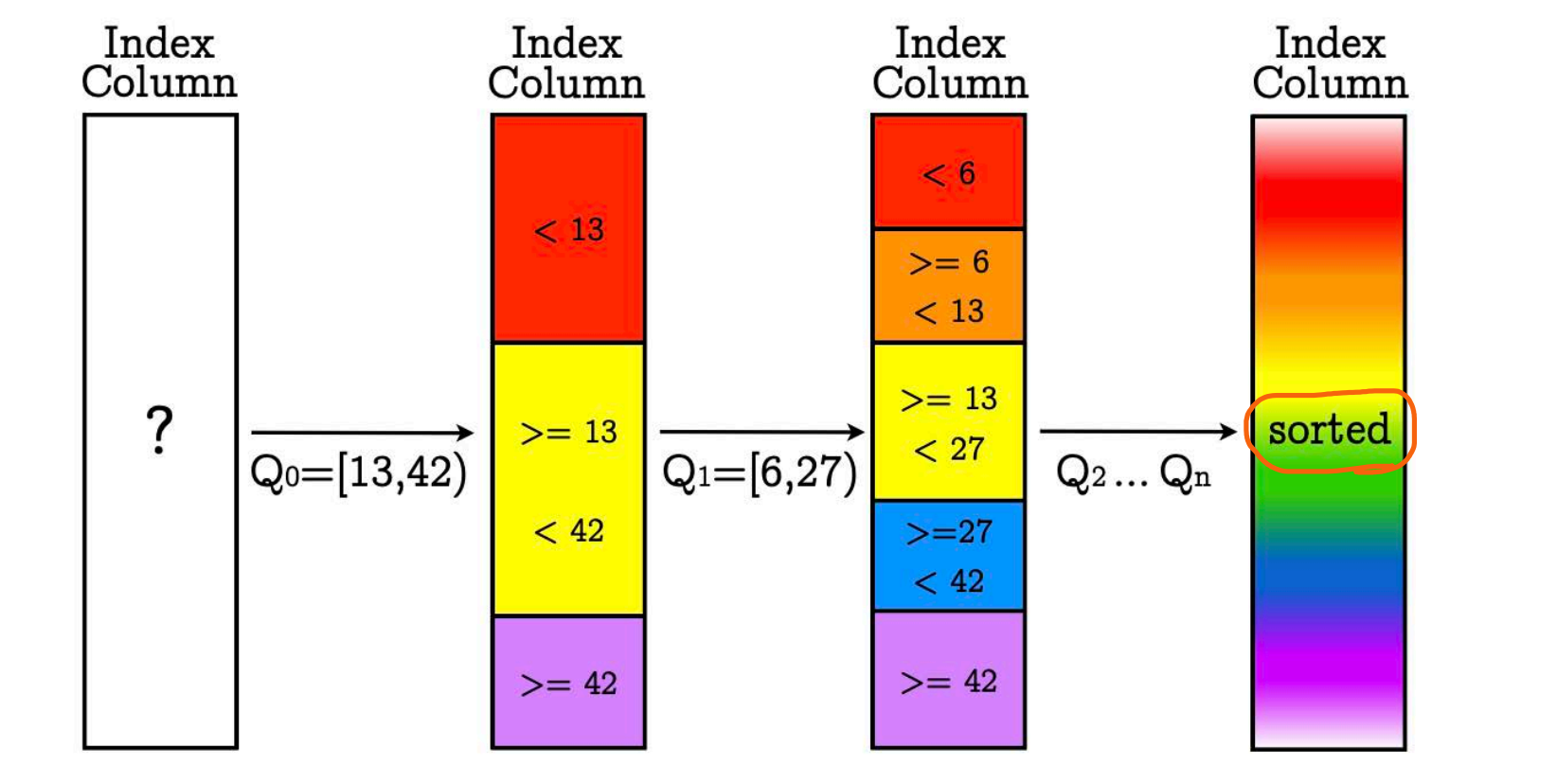
- By making no assumptions, we reduce overhead such as machine pre-processing, and labor costs.
- Also, we have no knowledge of the incoming workload.

OVERVIEW

- The algorithm consists of 3 main components:
  1. Index refinement generalization (partition-in-K)
  2. Adaptive reorganization => (picking a good K value)
  3. Defusing skewed key distributions (Skew Correction)

GENERAL STRATEGY

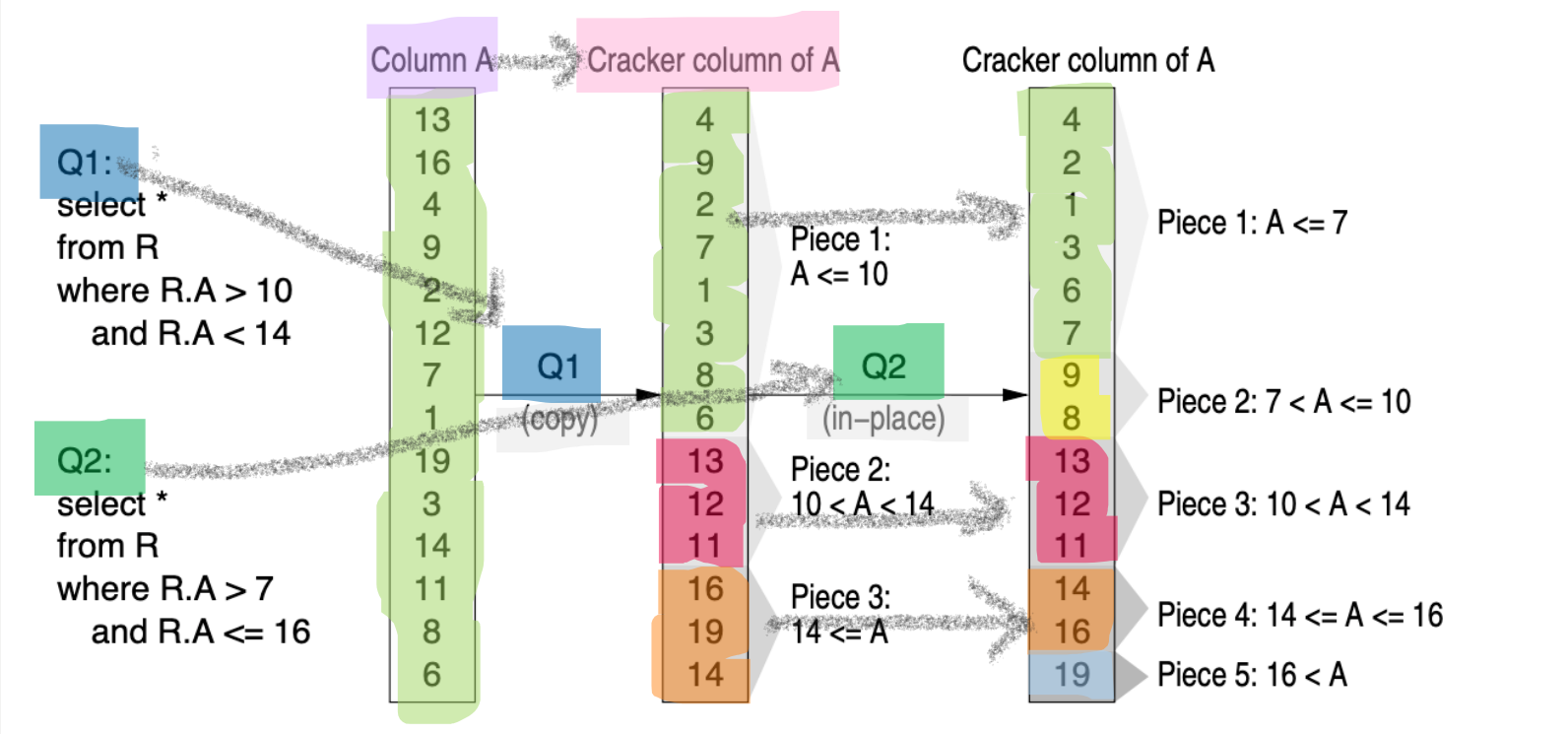
- Defer index maintenance until query processing
- Reorganize data dynamically as queries come in to improve queries



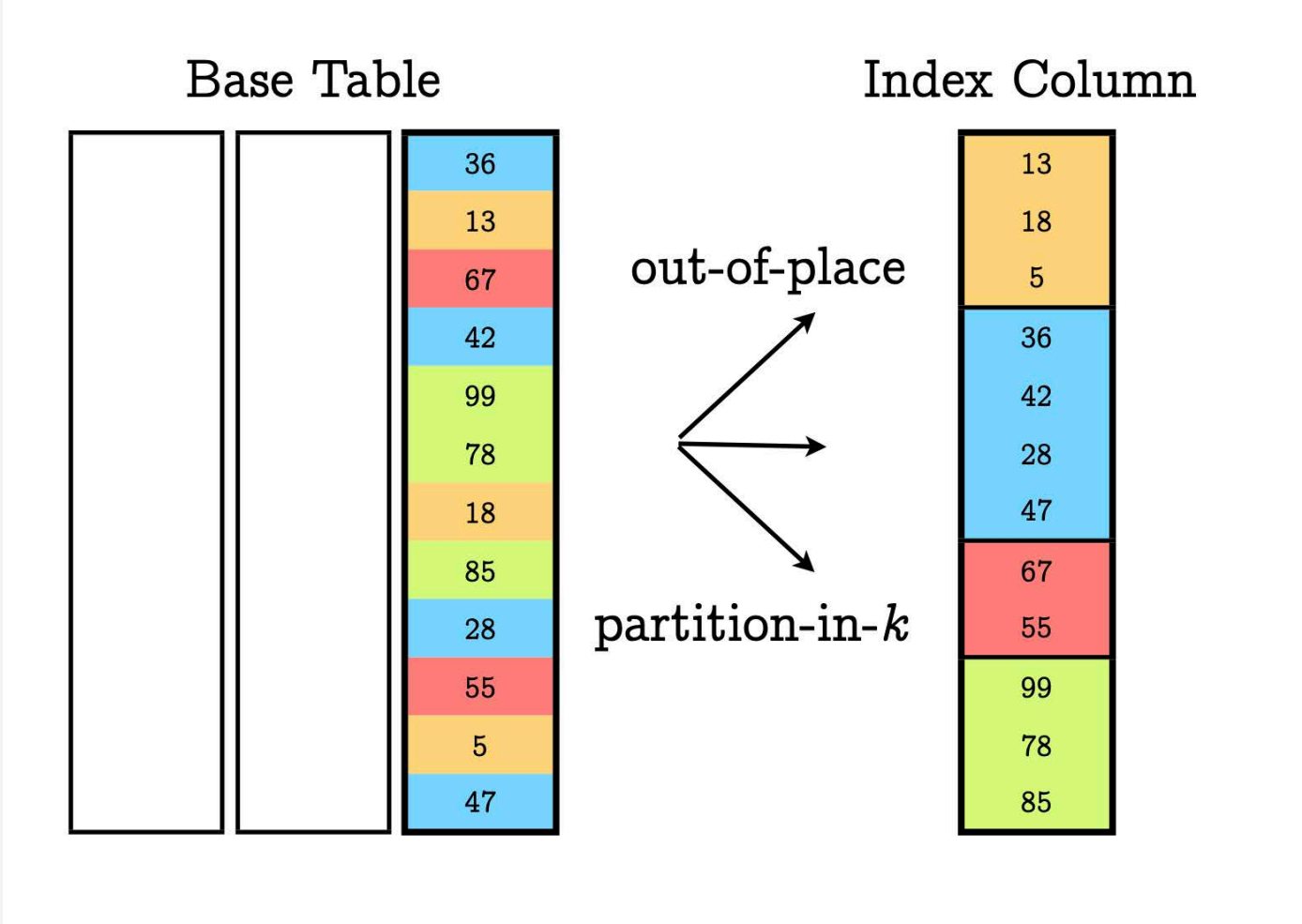
DEEP DIVE: CRACKING

- **Database Cracking:** create indexes adaptively and incrementally as a side-product of query processing.
- **Common Methods:** Standard Cracking, Stochastic Cracking

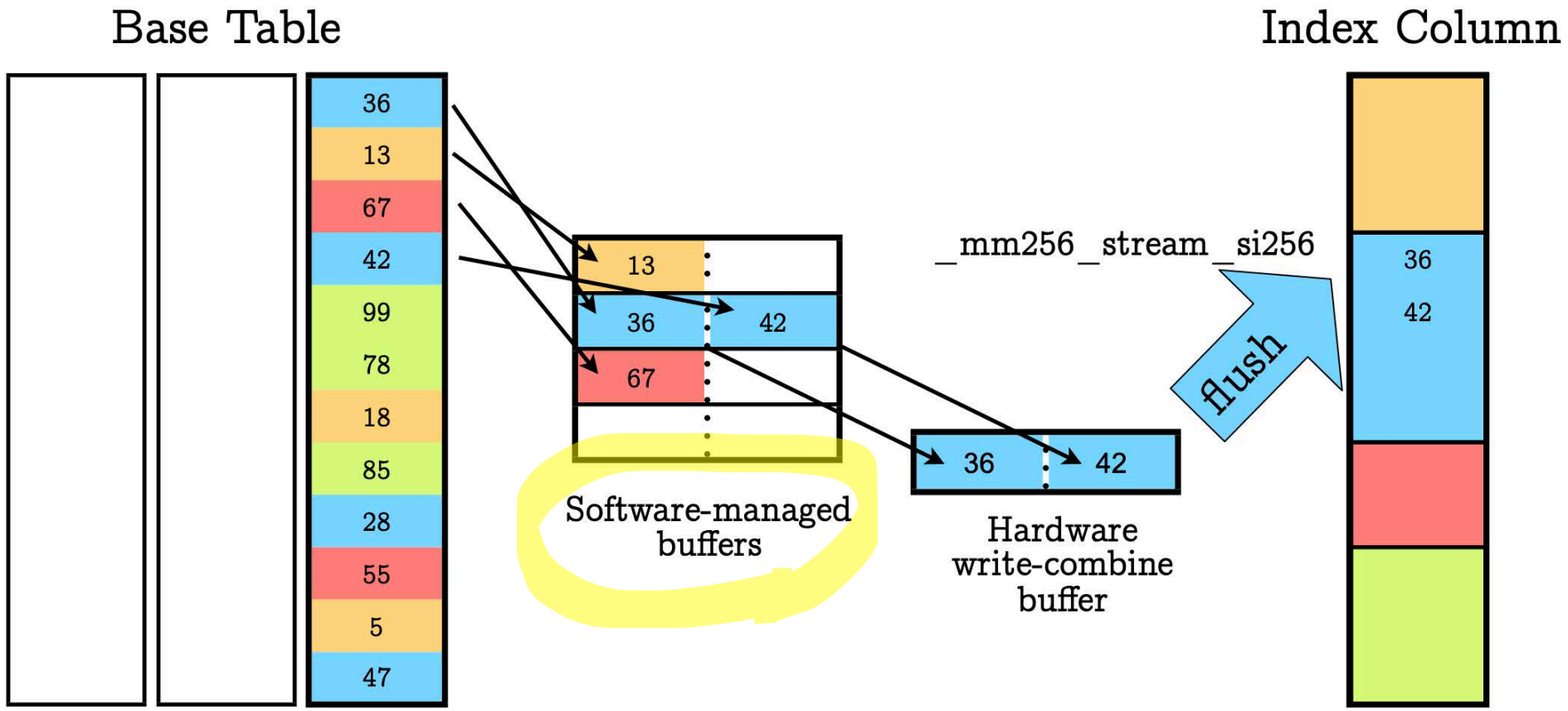
Standard Cracking



HANDLING THE FIRST QUERY



HANDLING THE FIRST QUERY



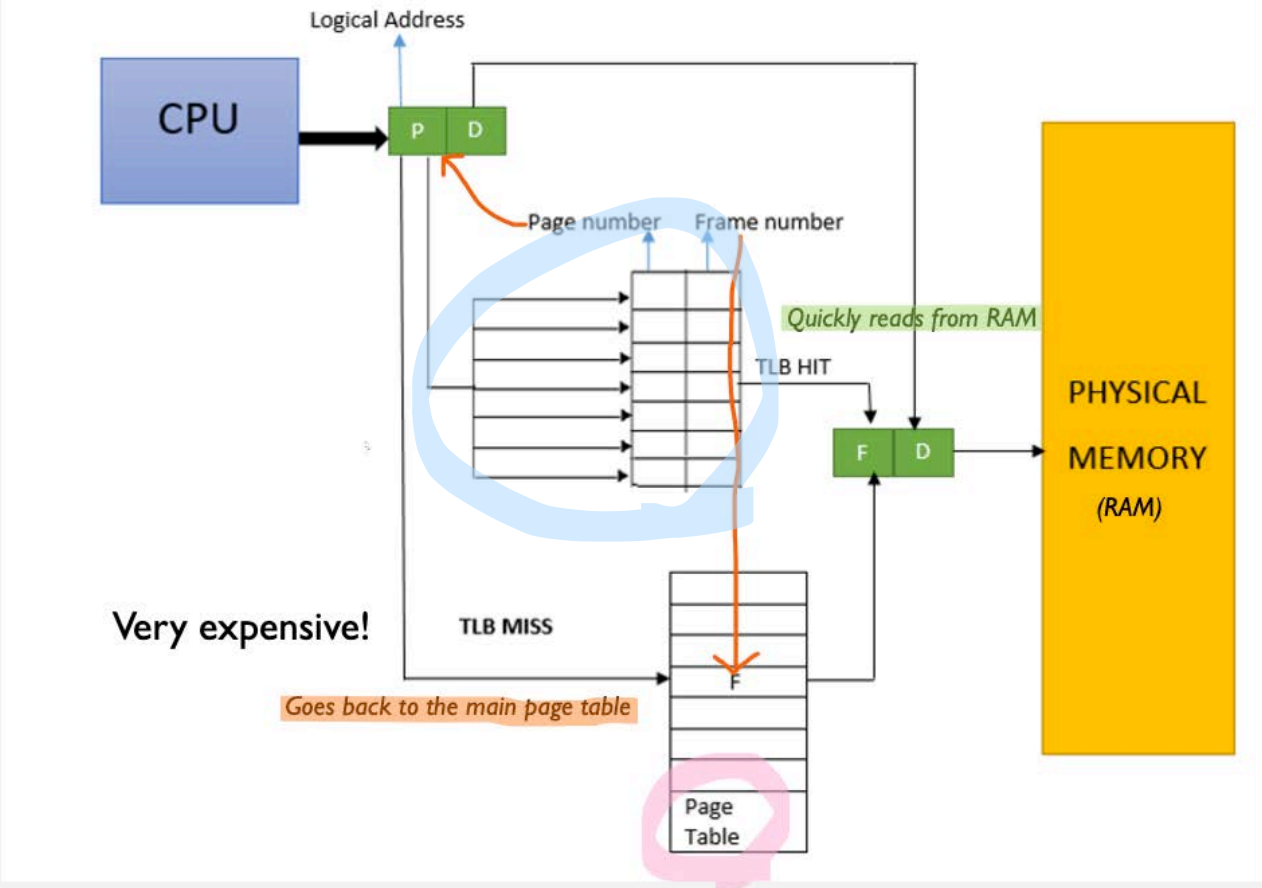
DEEP DIVE: TLB THEORY

Effective Memory Access Time (EMAT):

= WEIGHTED COST OF HIT
+ WEIGHTED COST OF MISS

= H * (C+M) + (1-H) * (C+2M)

H = Hit ratio of TLB
M = Memory access time
C = TLB access time

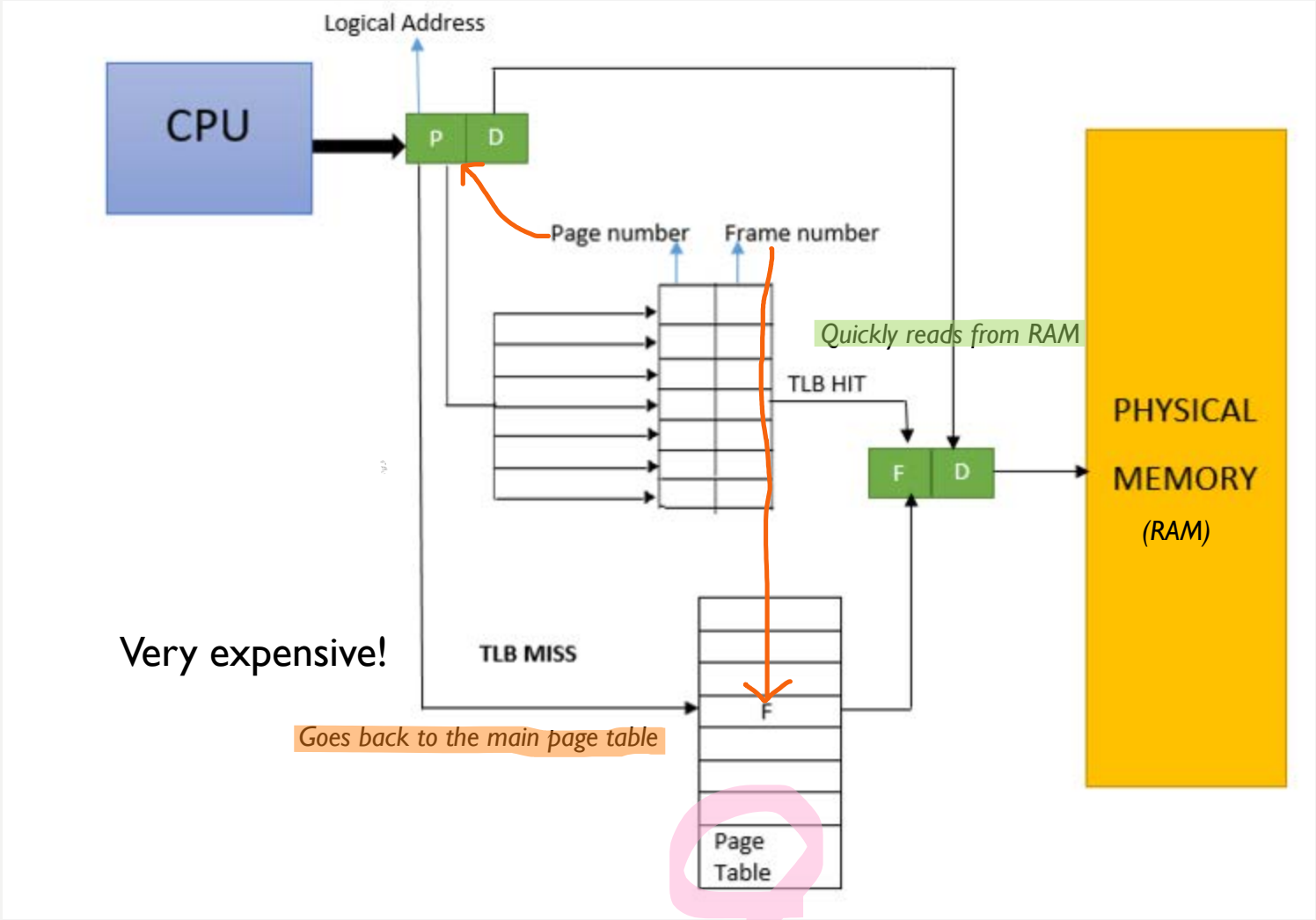


DEEP DIVE: TLB EXAMPLE

Example:
size: 12 bits – 4,096 entries
hit time:1 clock cycle
miss penalty: 30 clock cycles
miss rate: 1%

Effective Memory Access Time (EMAT):
(1+30) × 99% +
(1 + (2 x 30)) × 1% =
31.30

(31.30 clock cycles per memory access)



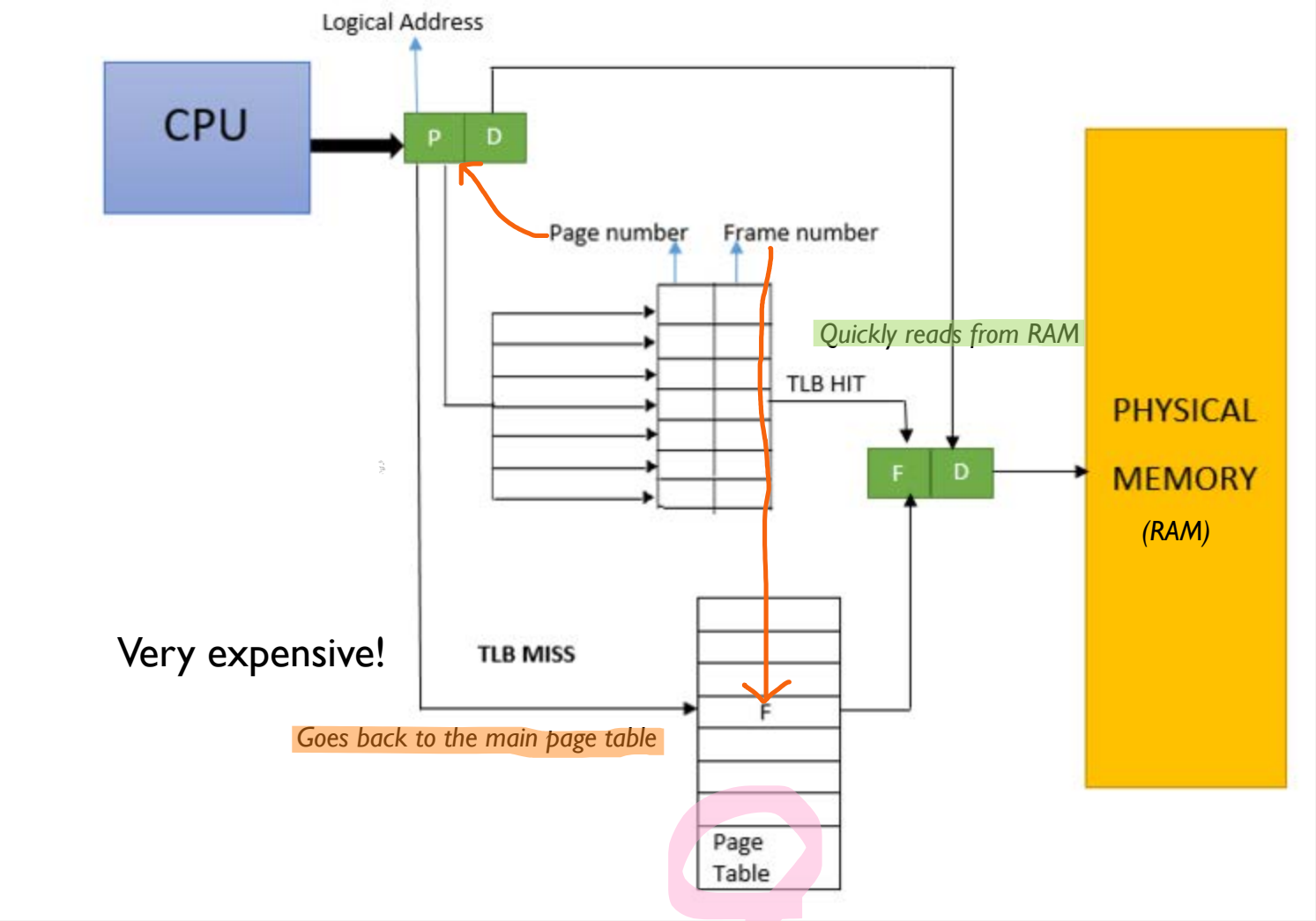
DEEP DIVE: TLB W/ SW BUFFER

Example:
size: 12 bits – 4,096 entries
hit time:1 clock cycle
miss penalty: 30 clock cycles
miss rate: **0.5%**

Effective Memory Access Time (EMAT):
(1+30) × 99.5% +
(1 + (2 x 30)) × **0.5%** =
31.15

(31.15 clock cycles per memory access)

… 31 is best we can do!



INDEX REFINEMENT GENERALIZATION

HANDLING SUBSEQUENT QUERIES

Open Question:

How do we adjust fanout for subsequent queries?

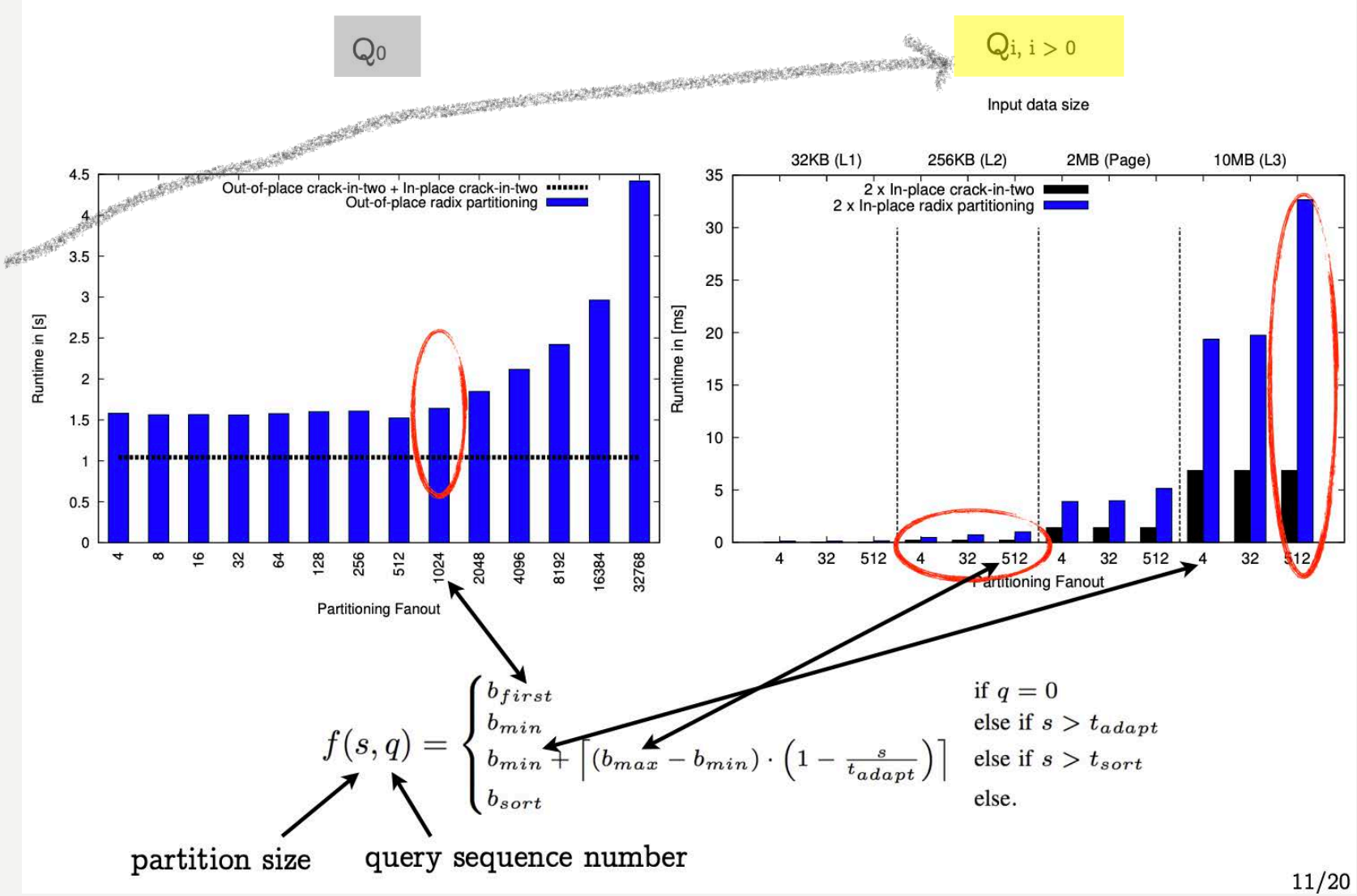
Solution:

We learn by experimentation,

Smaller fanout for the larger inputs,

Larger fanout for the smaller inputs

Penalty is acceptable.



CALCULATING PARTITIONS

Parameter	Meaning
b_{first}	Number of fan-out bits in the very first query.
t_{adapt}	Threshold below which fan-out adaption starts.
b_{min}	Minimal number of fan-out bits during adaption.
b_{max}	Maximal number of fan-out bits during adaption.
t_{sort}	Threshold below which sorting is triggered.
b_{sort}	Number of fan-out bits required for sorting.
$skewtol$	Threshold for tolerance of skew.

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

CALCULATING PARTITIONS

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

CALCULATING PARTITIONS

Parameter	Meaning
b_{first}	Number of fan-out bits in the very first query.
t_{adapt}	Threshold below which fan-out adaption starts.
b_{min}	Minimal number of fan-out bits during adaption.
b_{max}	Maximal number of fan-out bits during adaption.
t_{sort}	Threshold below which sorting is triggered.
b_{sort}	Number of fan-out bits required for sorting.
$skewtol$	Threshold for tolerance of skew.

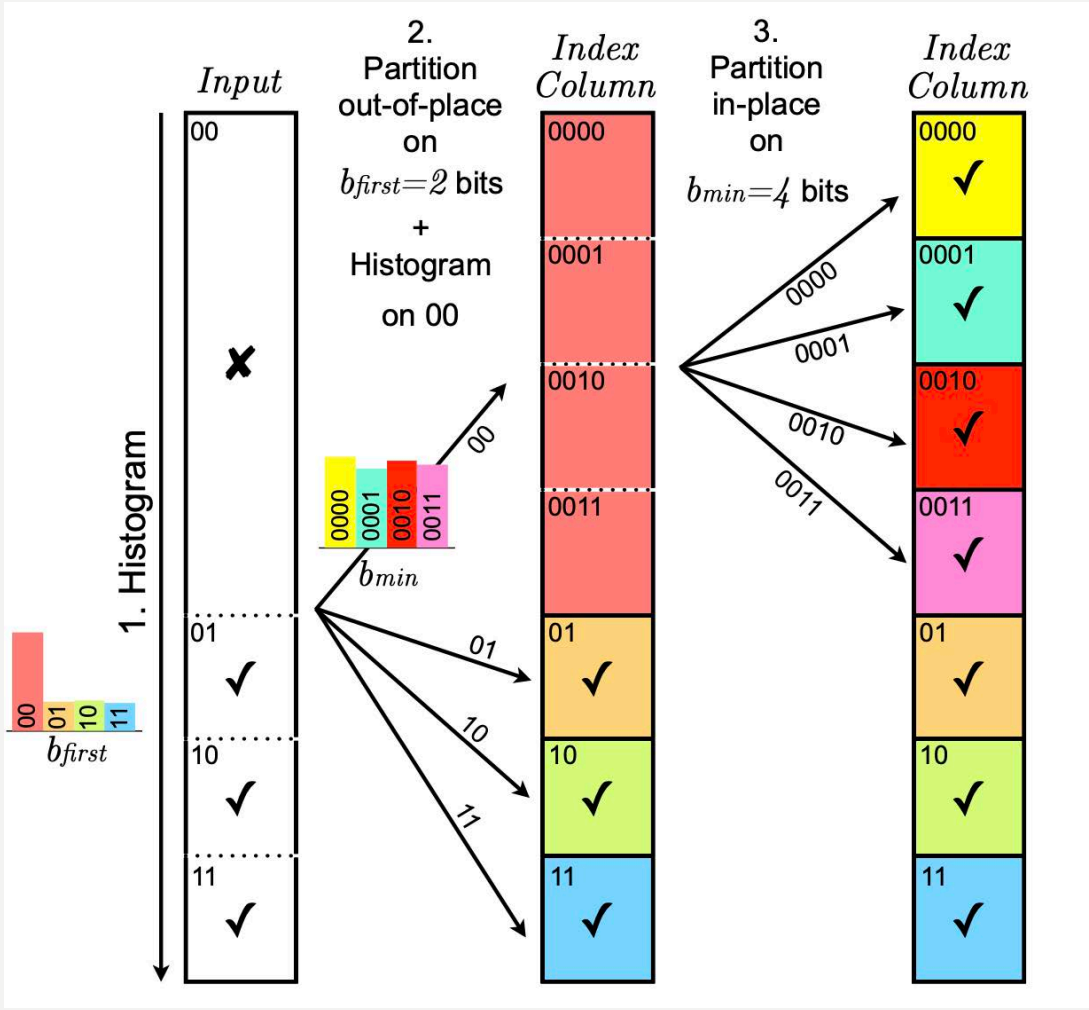
$$f(s,q) = \begin{cases} b_{first} & \text{if } q = 0 \\ b_{min} & \text{else if } s > t_{adapt} \\ b_{min} + \left\lceil (b_{max} - b_{min}) \cdot \left(1 - \frac{s}{t_{adapt}}\right) \right\rceil & \text{else if } s > t_{sort} \\ b_{sort} & \text{else.} \end{cases}$$

CALCULATING PARTITIONS

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

Parameter	Meaning
b_{first}	Number of fan-out bits in the very first query.
t_{adapt}	Threshold below which fan-out adaption starts.
b_{min}	Minimal number of fan-out bits during adaption.
b_{max}	Maximal number of fan-out bits during adaption.
t_{sort}	Threshold below which sorting is triggered.
b_{sort}	Number of fan-out bits required for sorting.
$skewtol$	Threshold for tolerance of skew.

ADJUSTING SKEW



DEFUSING SKEWED KEY DISTRIBUTIONS

RESULTS

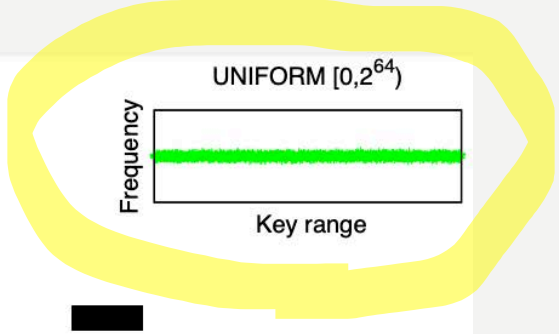
- 2x Speedup over the best baseline
- Faster convergence compared to the state-of the art (almost immediate)
- Suitable for multiple workloads

PERFORMANCE

DC = standard cracking

DD1R = stochastic cracking

HCS = hybrid cracking



$b_{first}=10$

$b_{min}=3$

$b_{max}=6$

$t_{adapt}=64MB$

$t_{sort}=256KB$

TOP PERFORMANCE:

ADAPTIVE INDEXING + SIMULATED ANNEALING

… 2X FASTER THAN THE BEST!

BASELINES

- To better help our understanding of the final results, we see this chart for some help in understanding other methods.

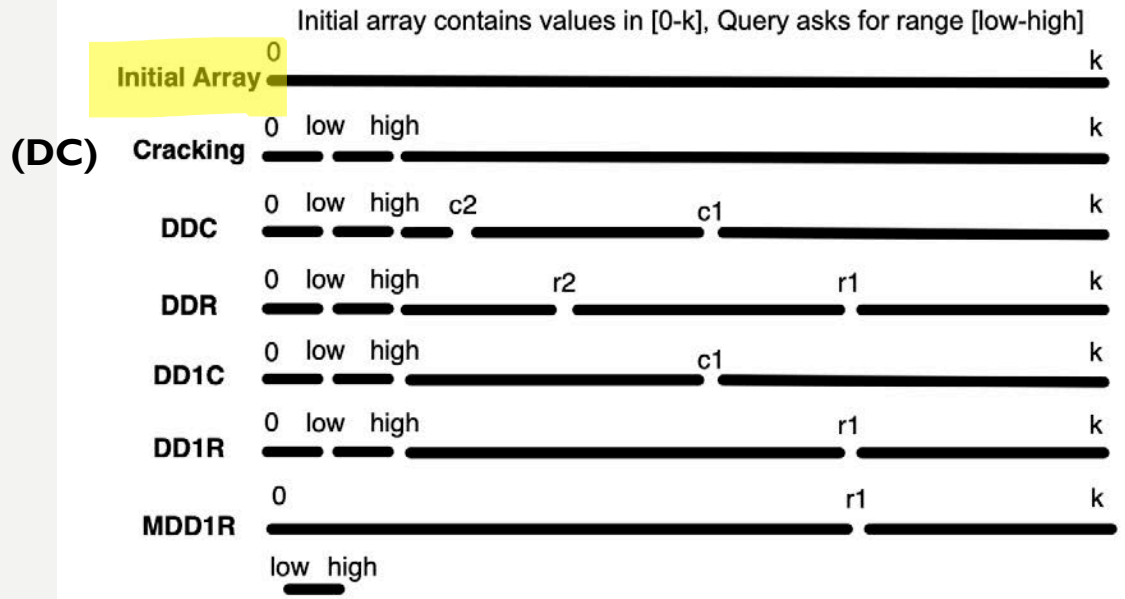


Figure 3: Cracking algorithms in action.

BASELINES

- To better help our understanding of the final results, we see this chart for some help in understanding other methods.

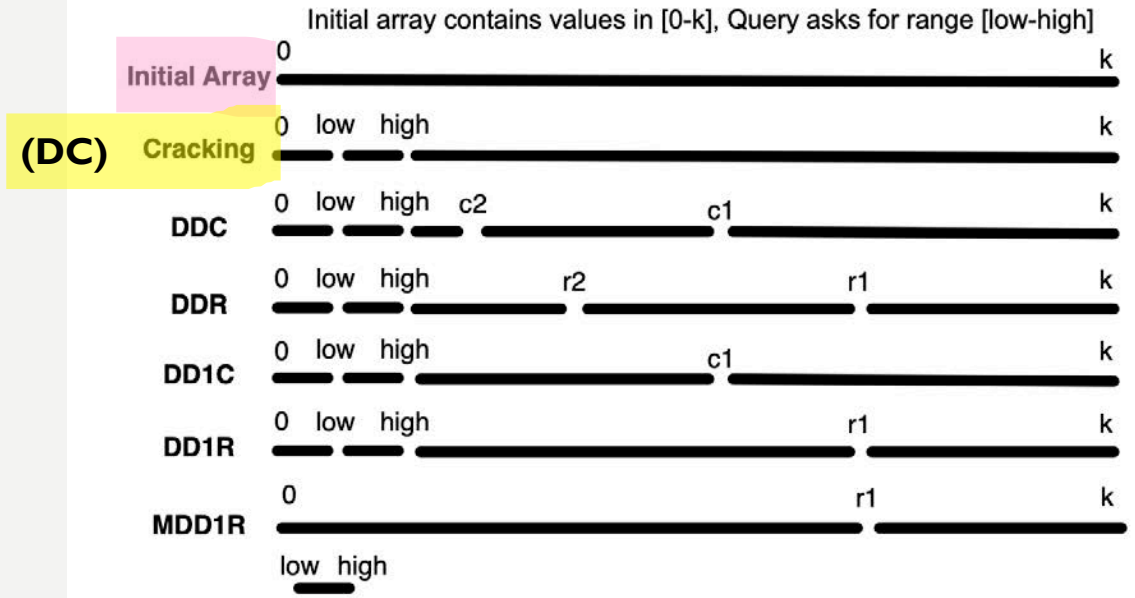


Figure 3: Cracking algorithms in action.

BASELINES

- To better help our understanding of the final results, we see this chart for some help in understanding other methods.

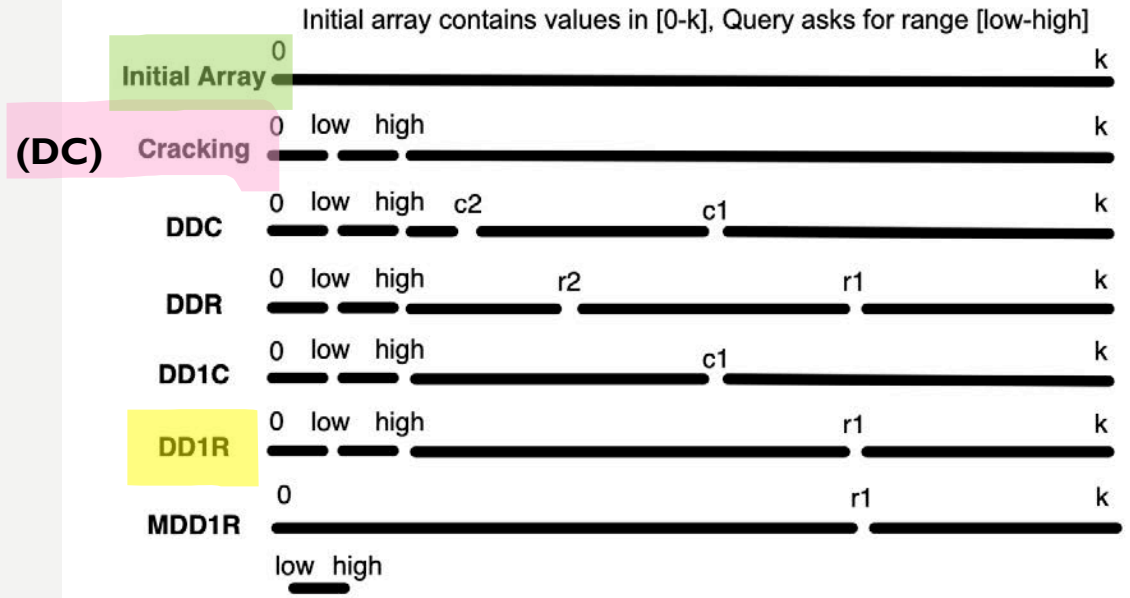
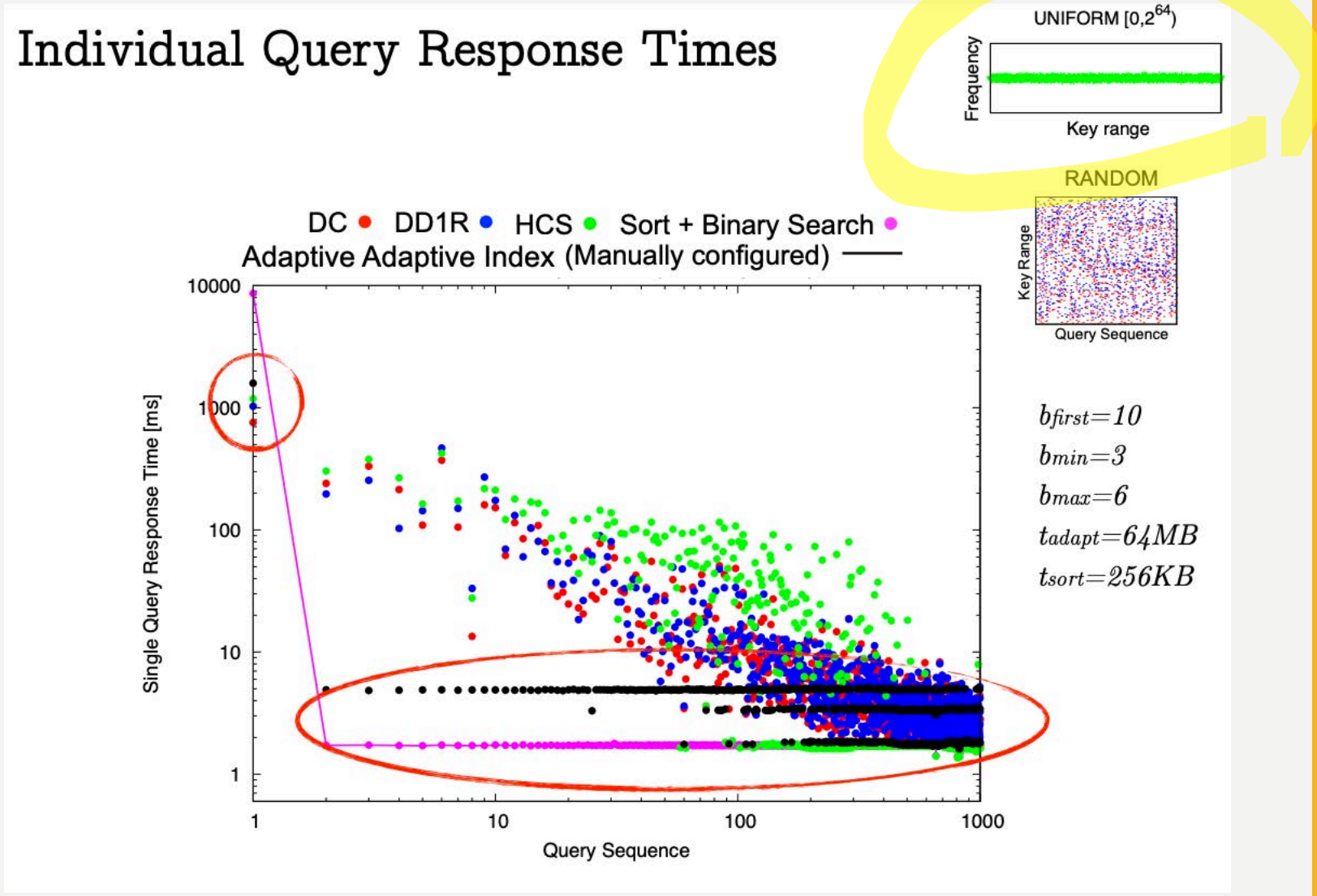


Figure 3: Cracking algorithms in action.

RESPONSE TIME

DC = standard cracking
DD1R = stochastic cracking
HCS = hybrid cracking

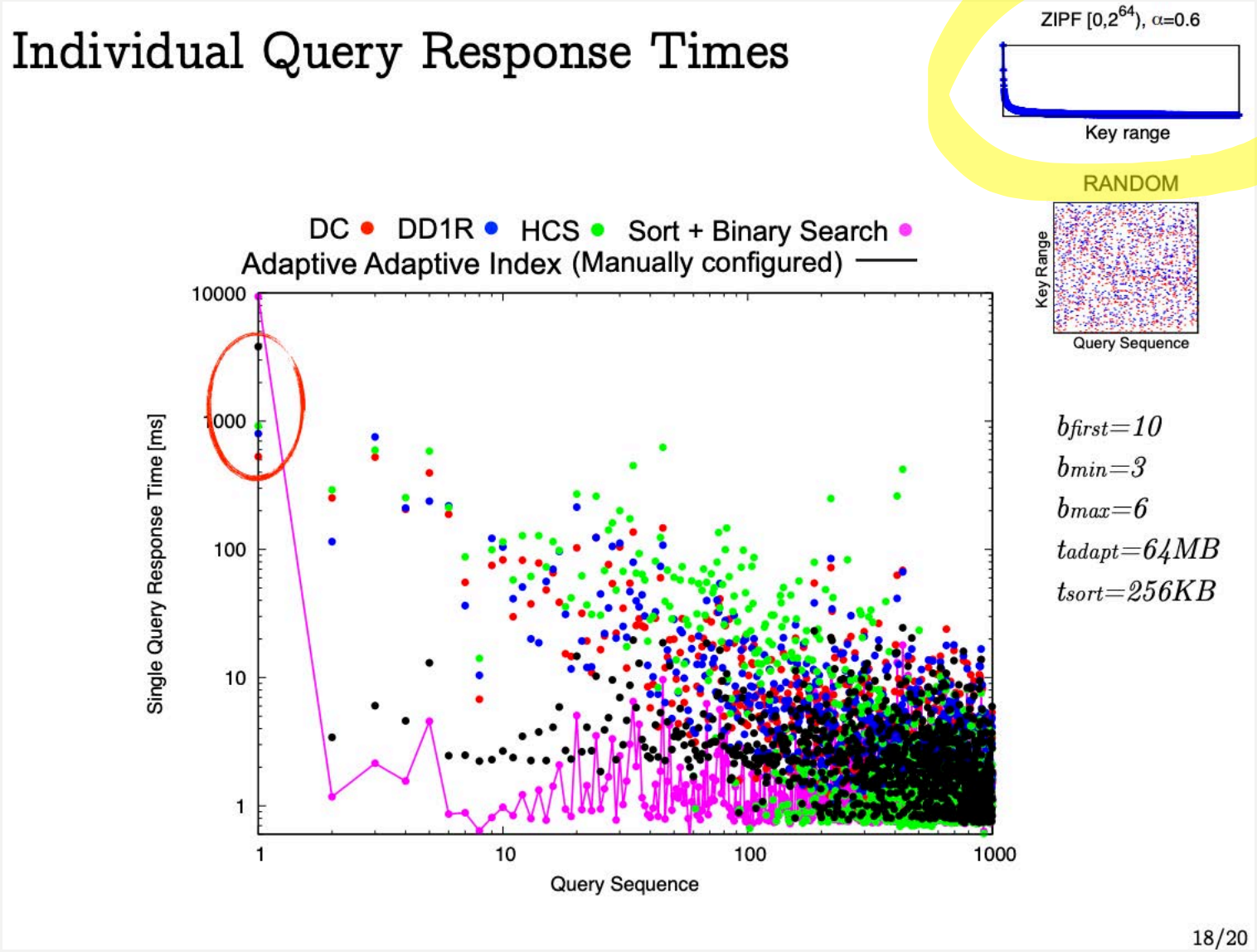
Meta-adaptive indexing shows highly stable convergence!



RESPONSE TIME

DC = standard cracking
DD1R = stochastic cracking
HCS = hybrid cracking

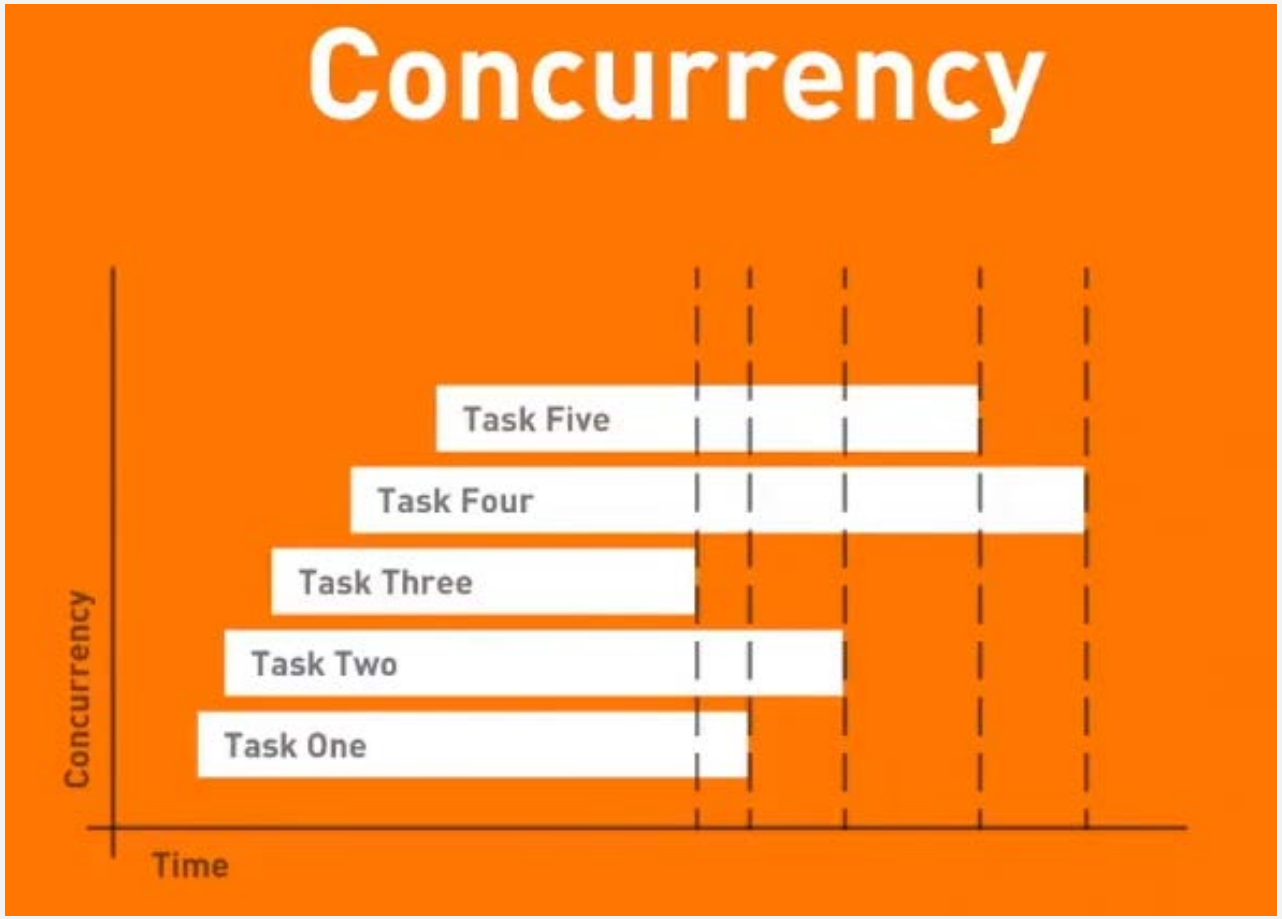
MAINTAINS FAST CONVERGANCE WITH DIFFICULT DATA



CAVEATS

- Meta-adaptivity is still sensitive to certain workloads
- Very large workloads that don't fit
- Also, what if the data being queried is antagonistic to the algorithm?

IDEAS FOR IMPROVEMENT



IDEAS FOR IMPROVEMENT

Adding Statistics for:

- Modes: Calculating top-n modes for skew detection
- Variance: Incoming queries can be indexed for range variance to make indexing more

$$z = \frac{x - \mu}{\sigma}$$

Formula for z-score

$$\sigma^2 = \frac{\sum (x - \mu)^2}{N}$$

Formula for variance

THANK YOU!