Database cracking

Stratos Idreos, Martin Kersten and Stefan Manegold
CWI Amsterdam, The Netherlands

The open problem

Dynamic environments

What kind of indices should be used, when and on which data?

Database experts or special tools monitor the system

More difficult in databases with huge datasets

Database cracking

We explore self-organization

Each query triggers physical re-organization of the database

We designed and implemented a DBMS using database cracking

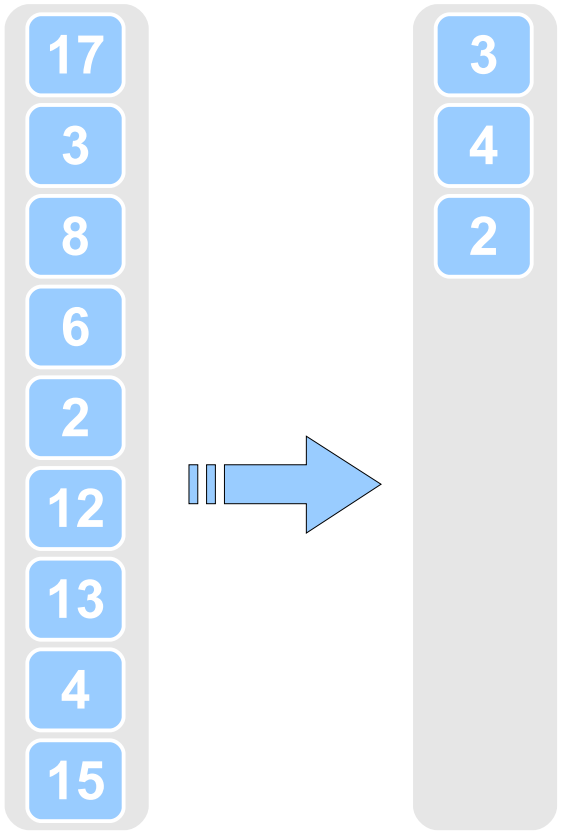
We work on top of MonetDB, a column oriented database system

Cracking a database

select A>5 and A<10

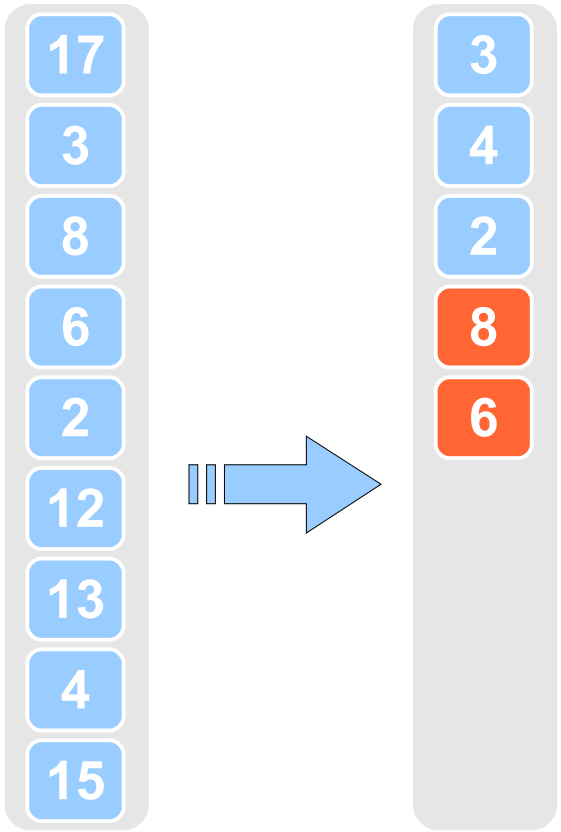
Cracking a database

select A>5 and A<10

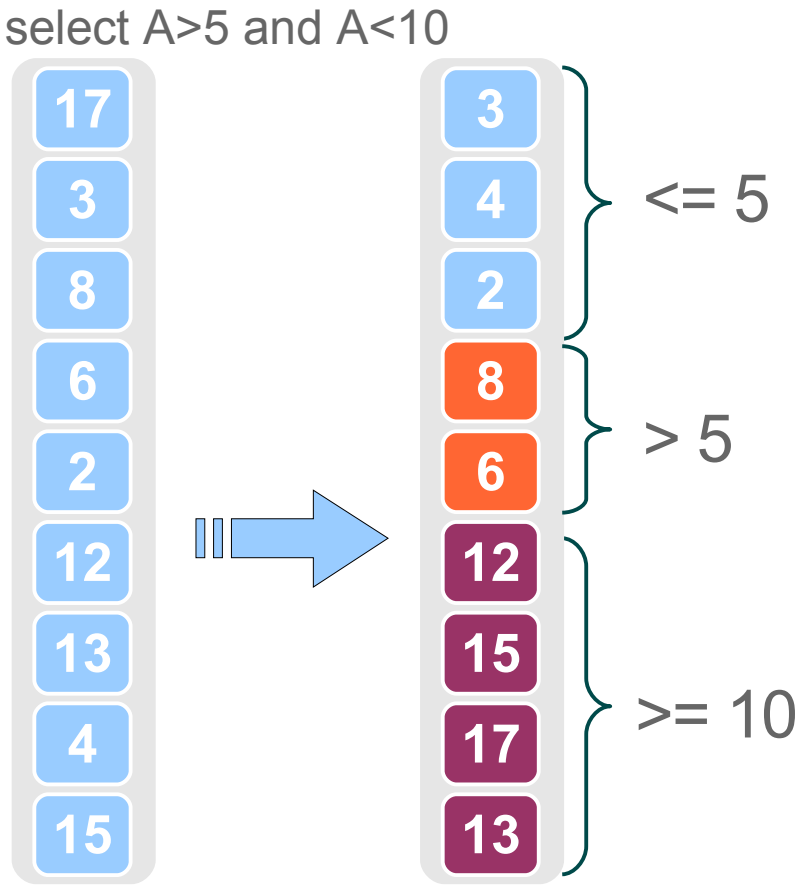


Cracking a database

select A>5 and A<10

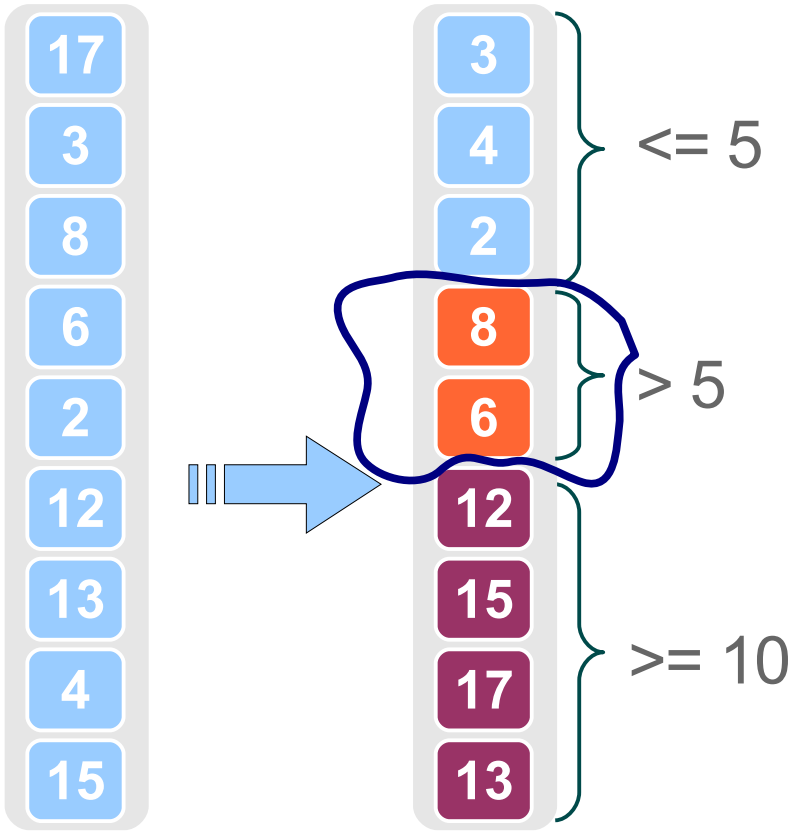


Cracking a database

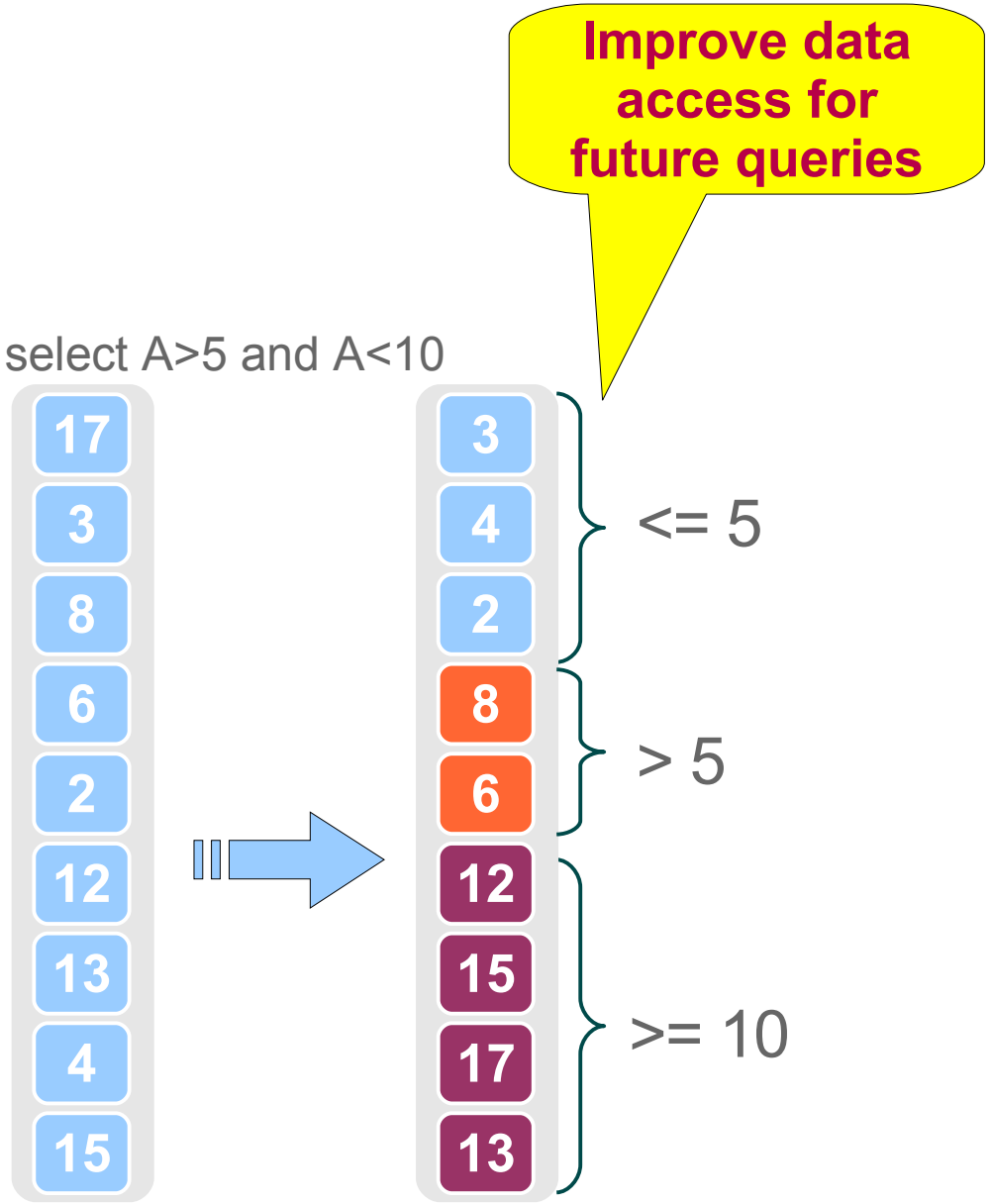


Cracking a database

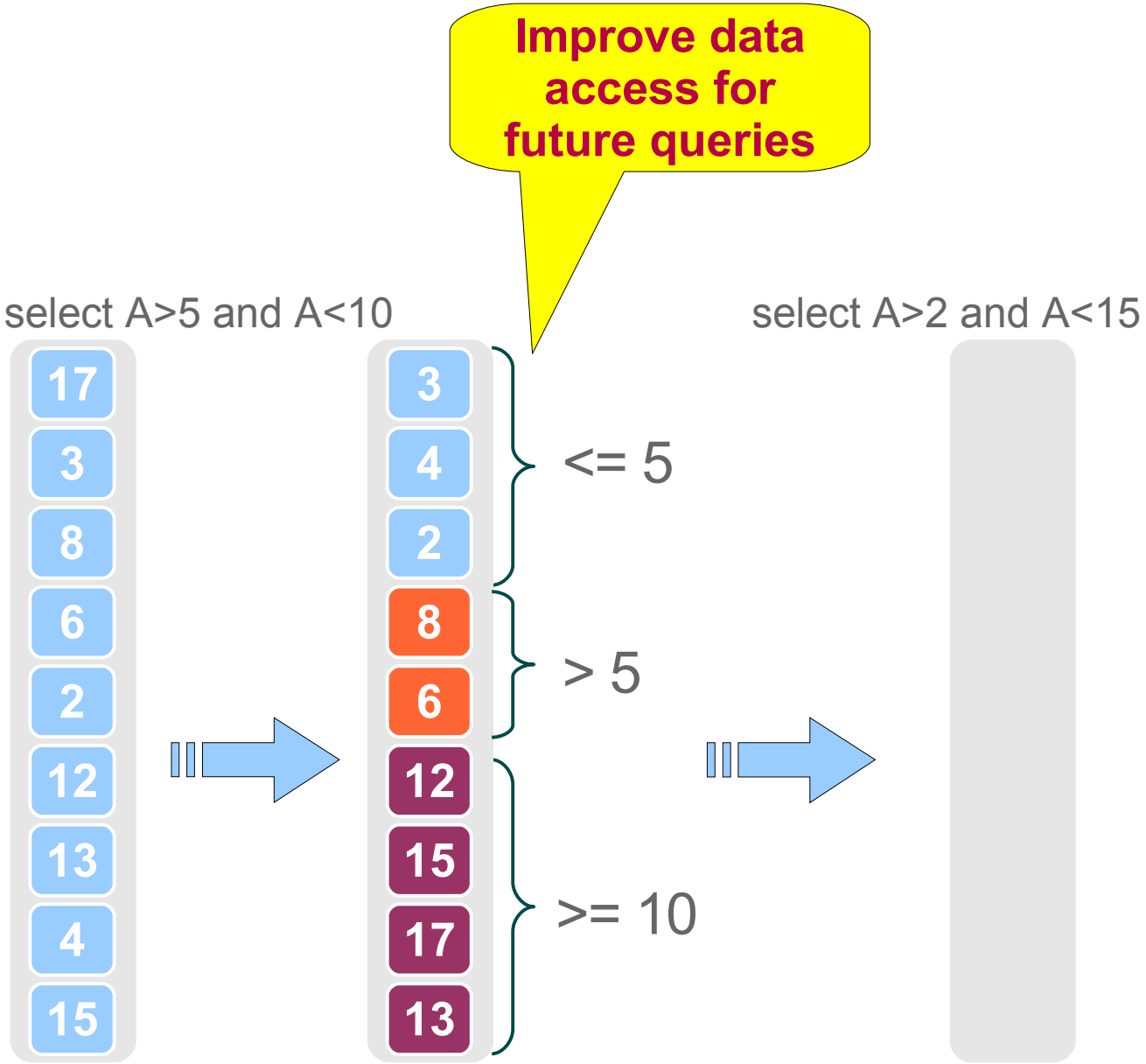
select A>5 and A<10



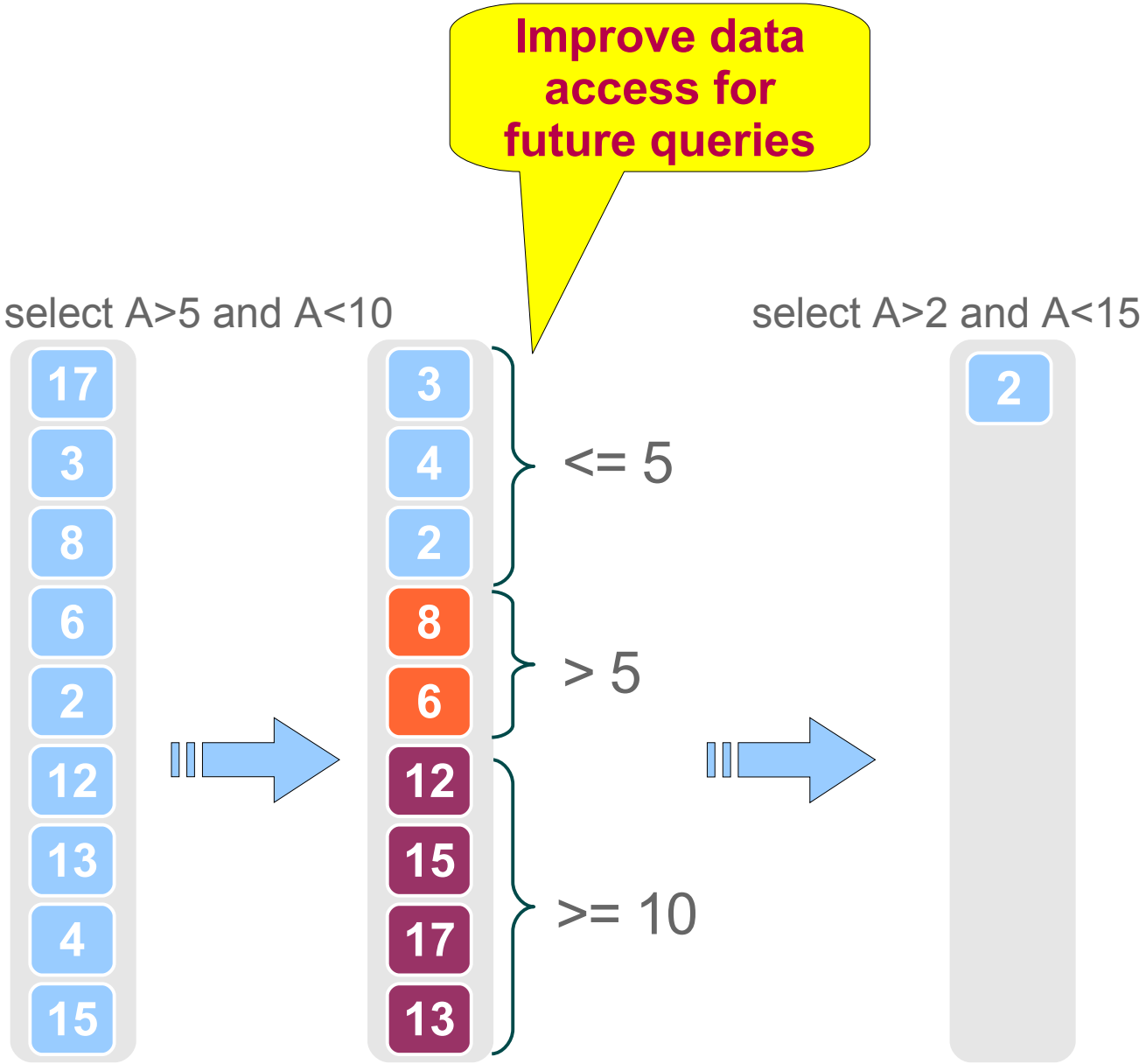
Cracking a database



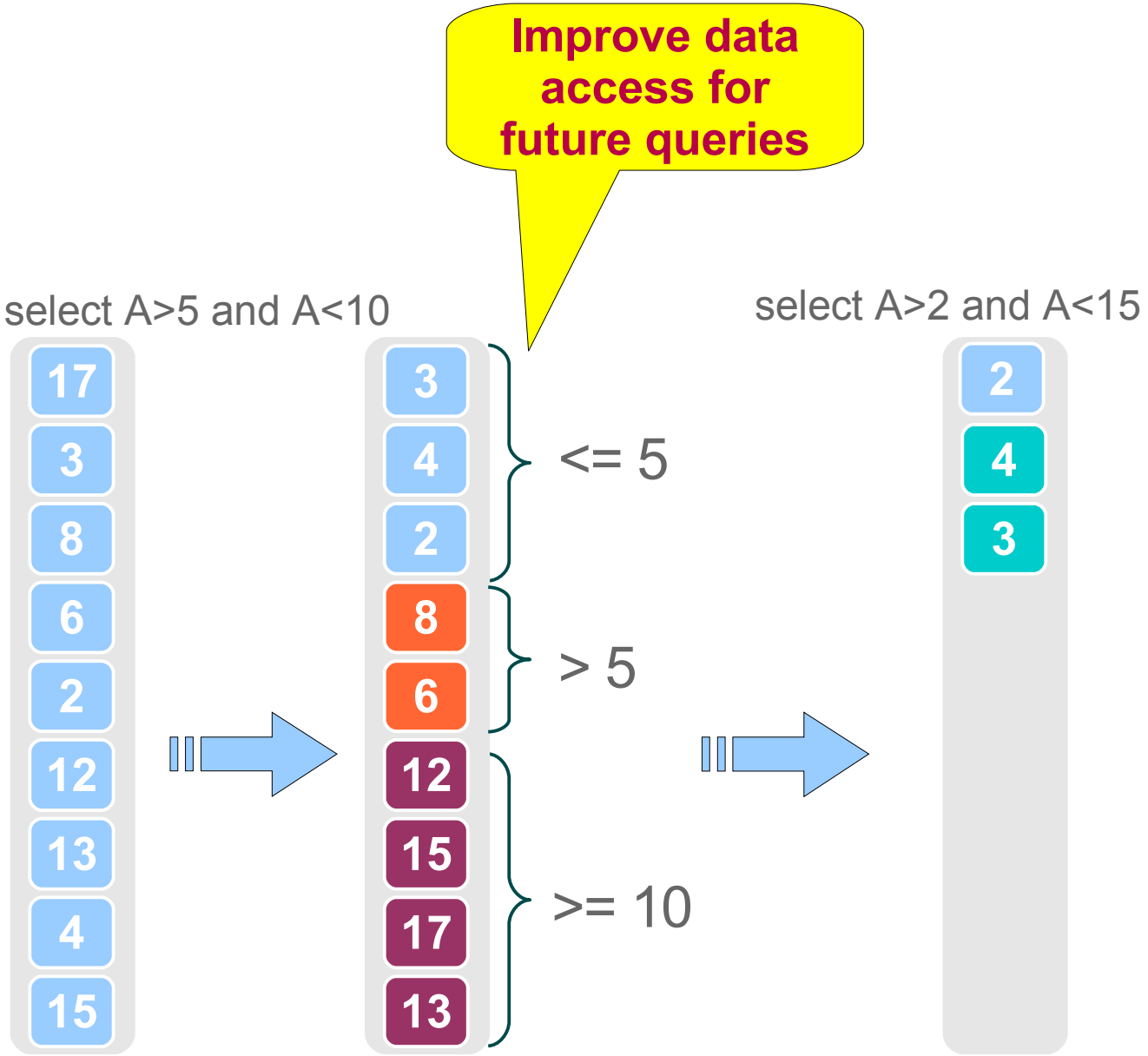
Cracking a database



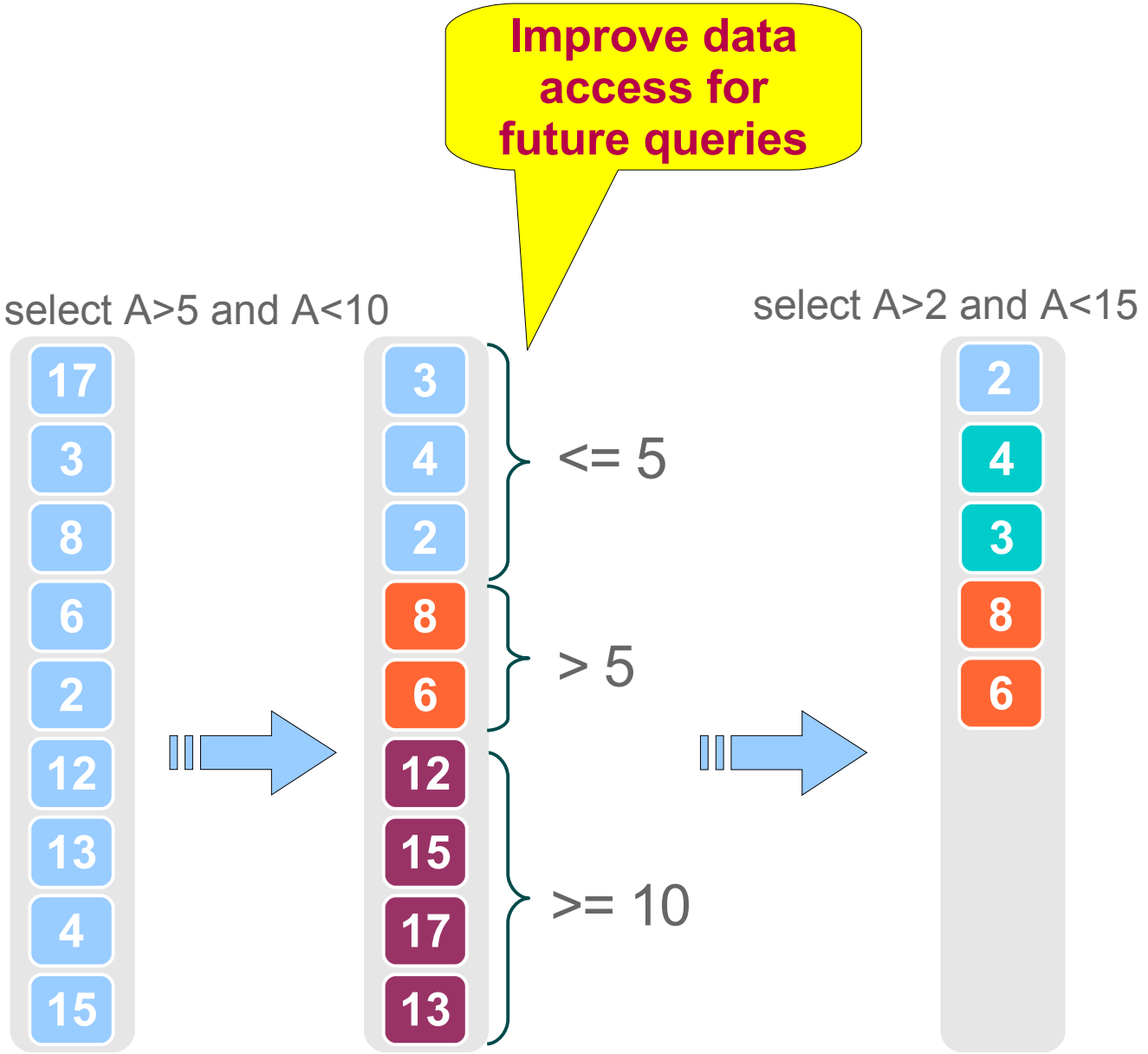
Cracking a database



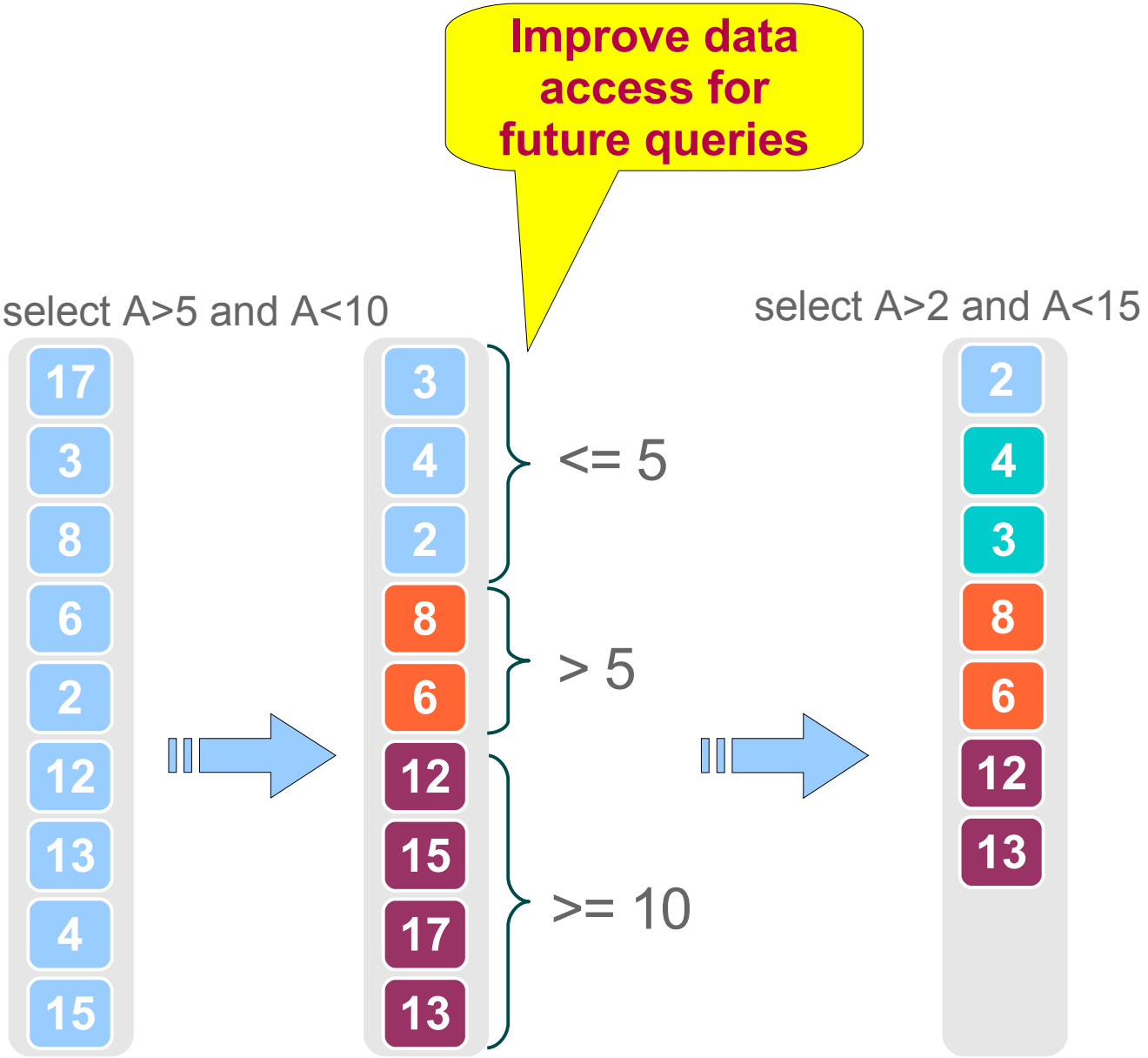
Cracking a database



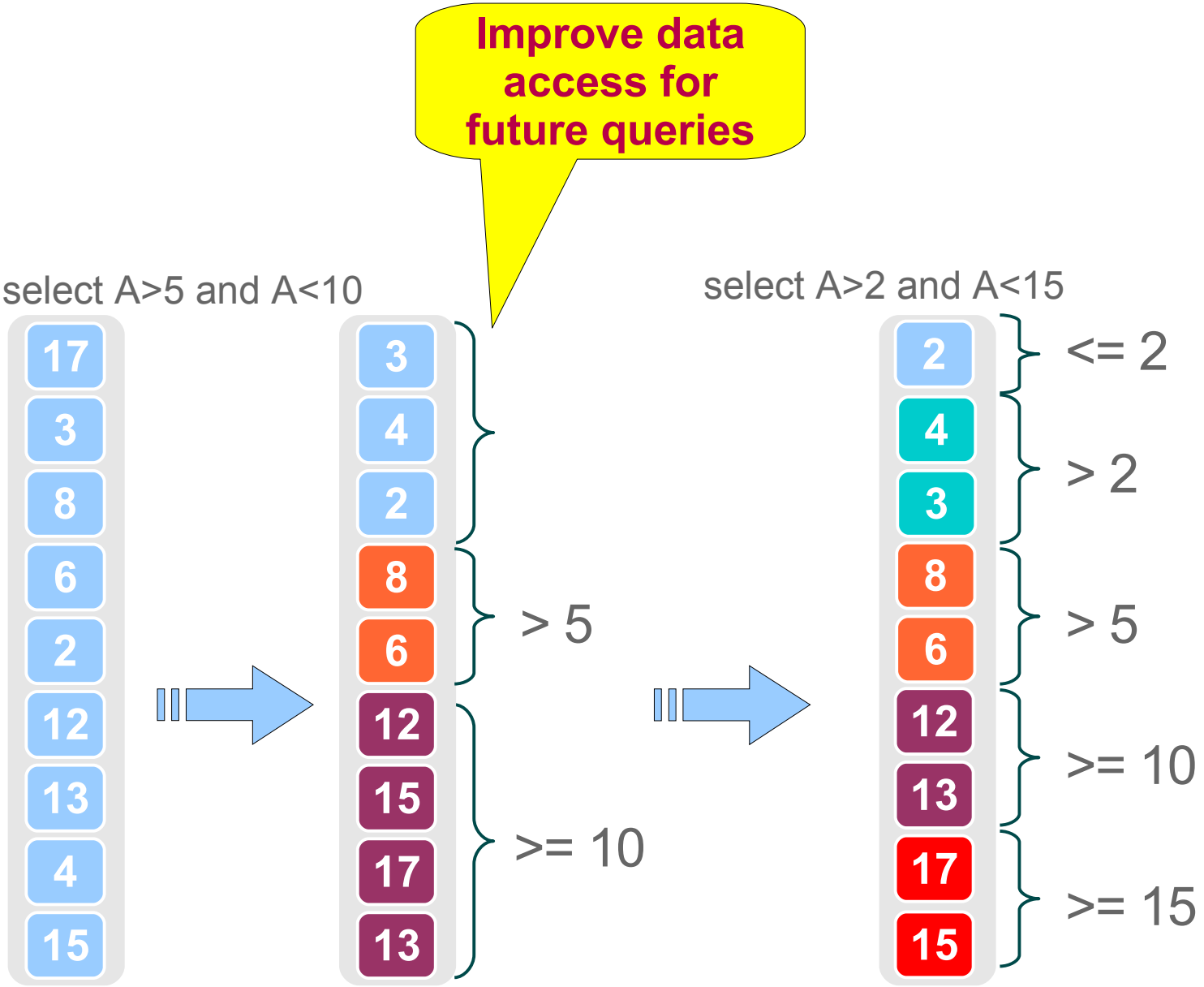
Cracking a database



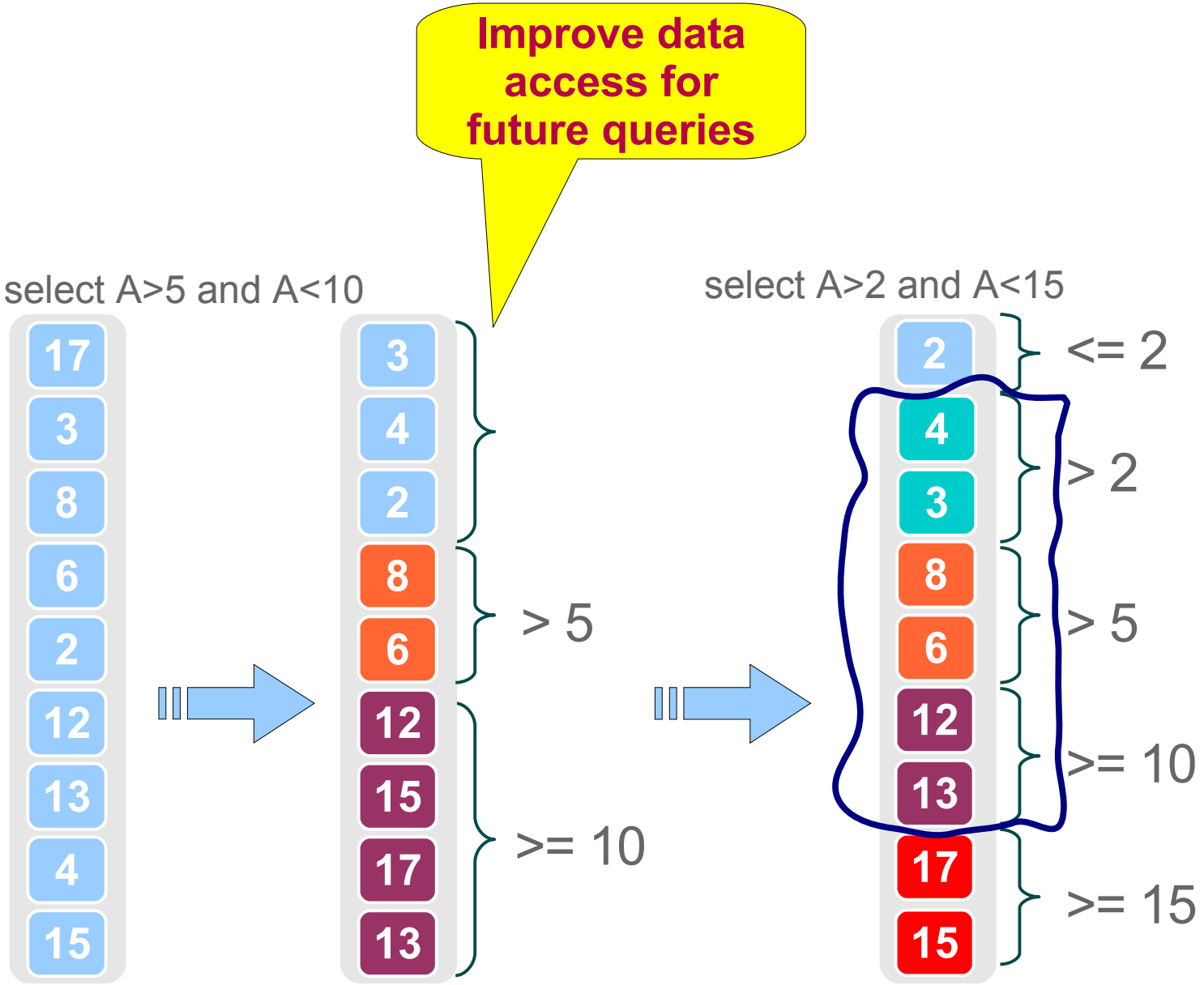
Cracking a database



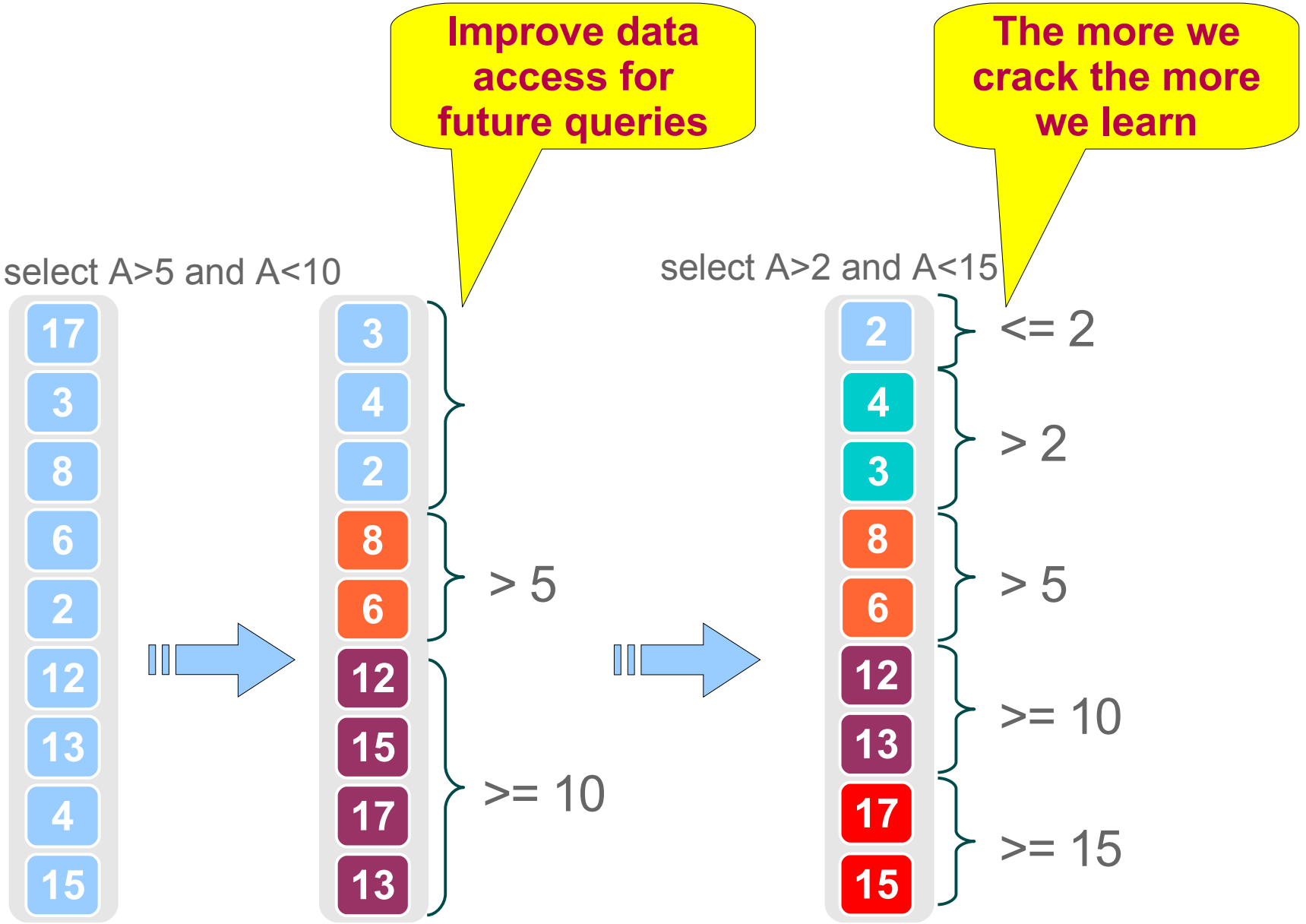
Cracking a database



Cracking a database



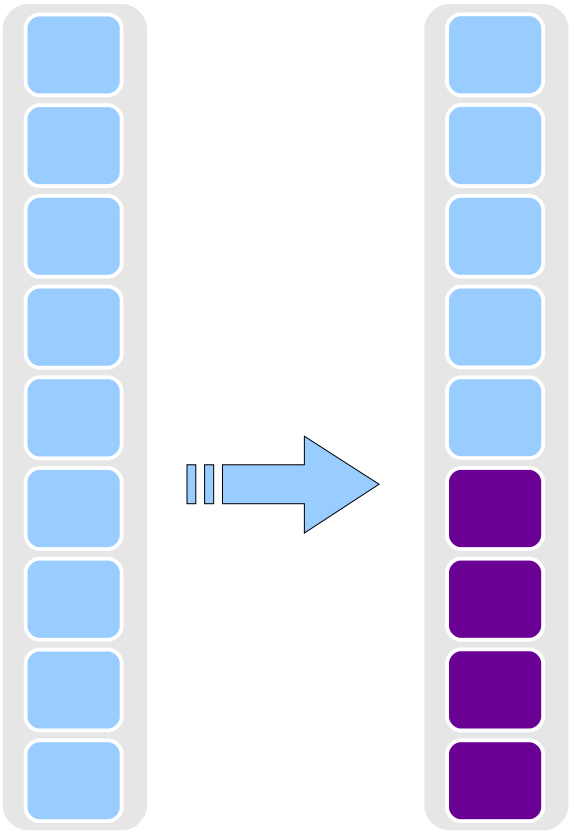
Cracking a database



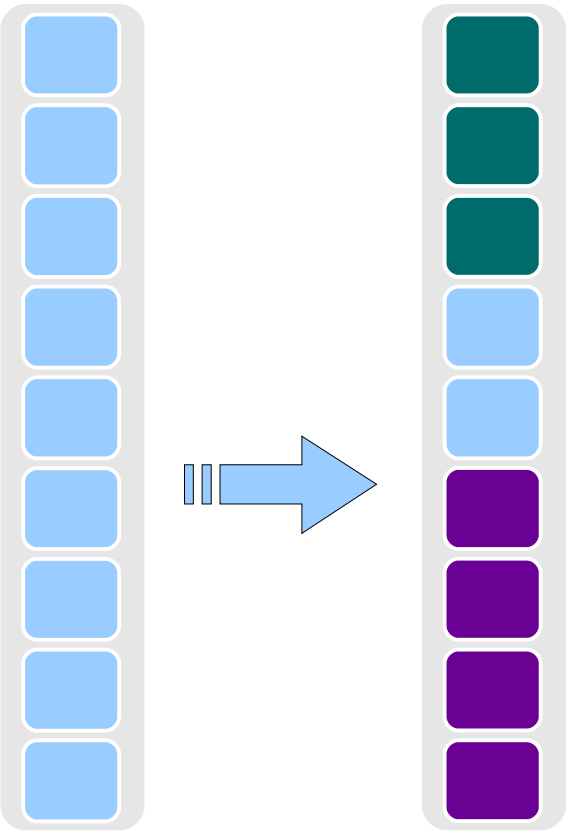
Cracking algorithms

There are two types of cracking algorithms

Split a piece in
two new pieces



Split a piece in
three new pieces

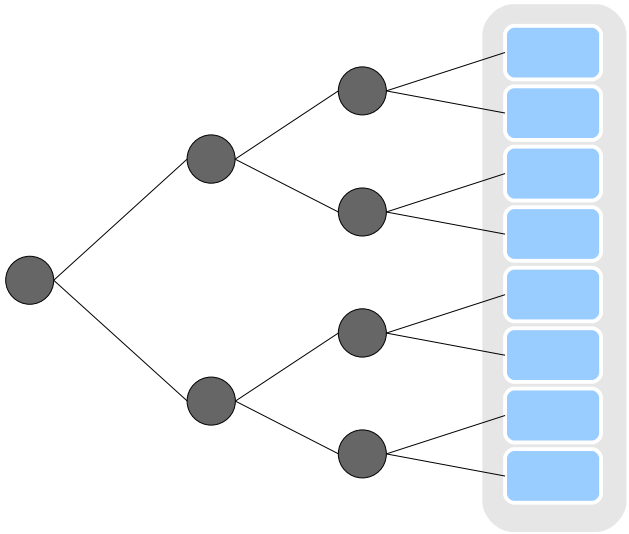


Design

The first time a range query is posed on an attribute *A*, a cracking DBMS makes a copy of column *A*, called the *cracker column* of *A*

A cracker column is continuously physically re-organized based on queries that need to touch attribute such as the result is in a contiguous space

For each cracker column, there is a *cracker index*



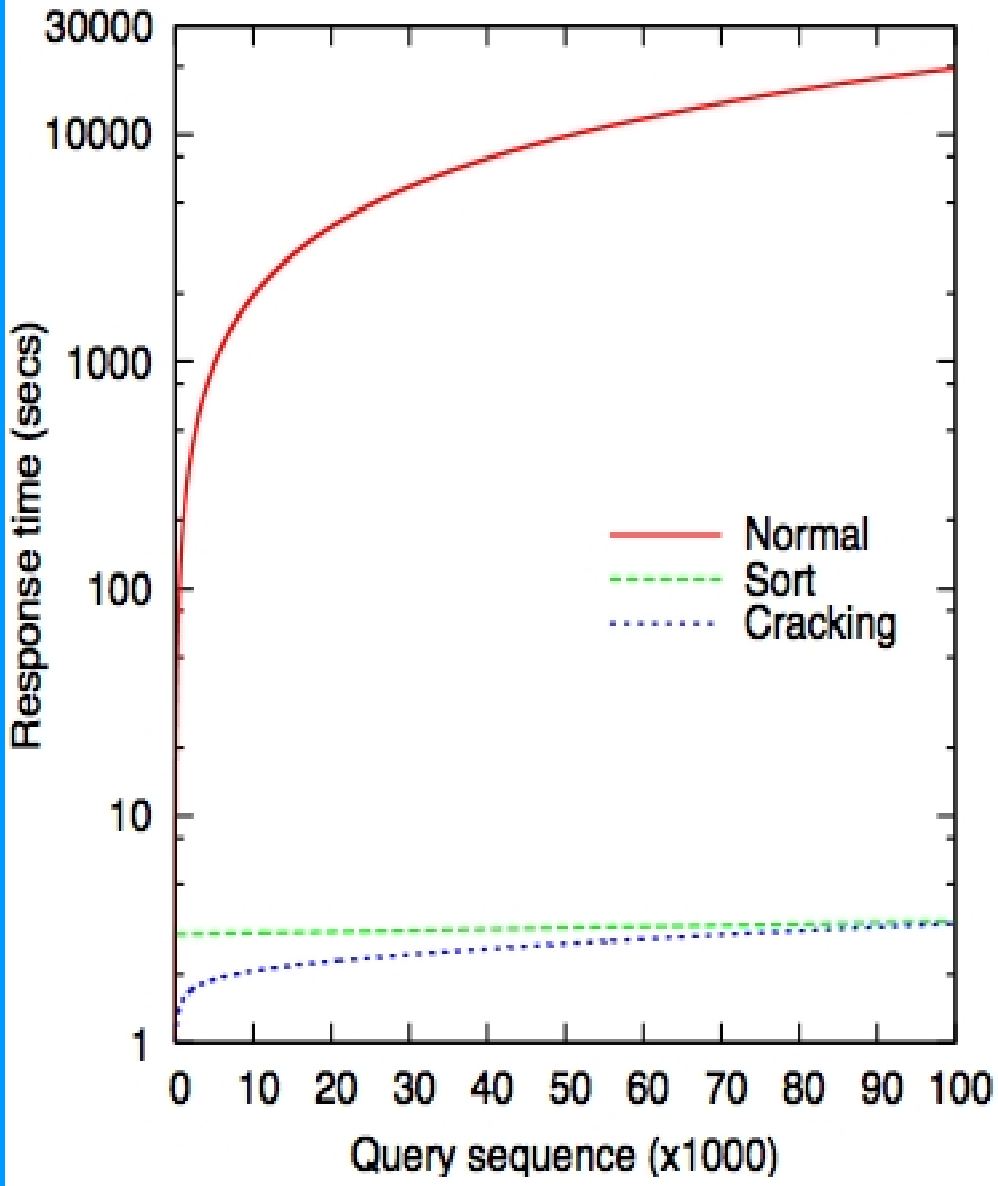
The cracker select operator

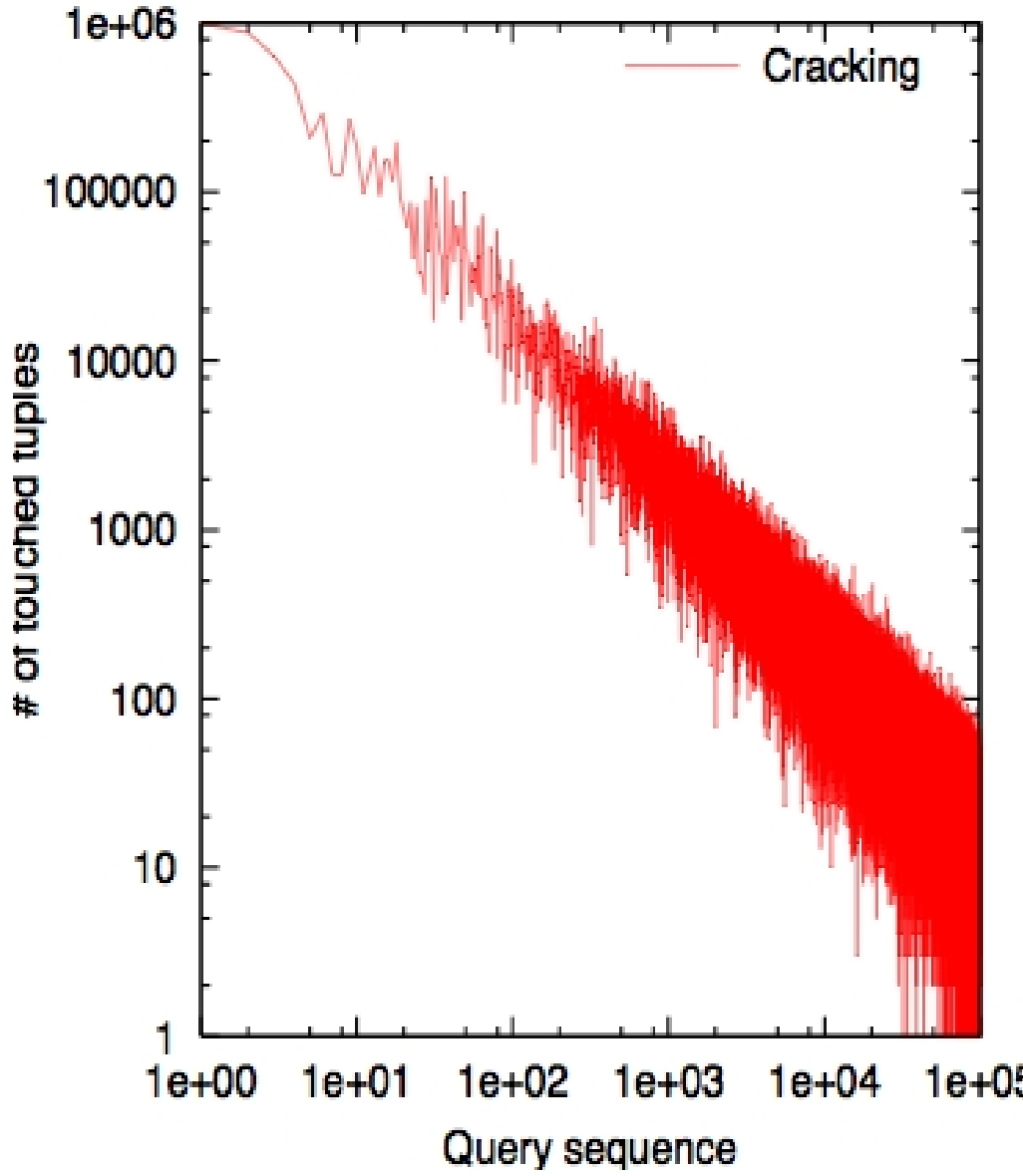
- The simple select operator:
  1. Scans the column
  2. Return a new column that contains qualifying values

- The crackers select operator:
  1. Searches the cracker index
  2. Physically re-organizes pieces found
  3. Update the cracker index
  4. Return a slice of the cracker column as result

  - More steps but faster because we analyze less data

Testing the select operator





Research and opportunities for cracking

- Optimization
  - optimal piece size / granularity / index (avl) depth
- Exploit cracking for join queries, aggregate queries etc.
- Concurrency issues
- Cracking histograms
- Distributed cracking
- A priori cracking
- ...

Research and opportunities for cracking

- Optimization
  - optimal piece size / granularity / index (avl) depth
- Exploit cracking for join queries, aggregate queries etc.
- Concurrency issues
- Cracking histograms
- Distributed cracking
- A priori cracking
- ...

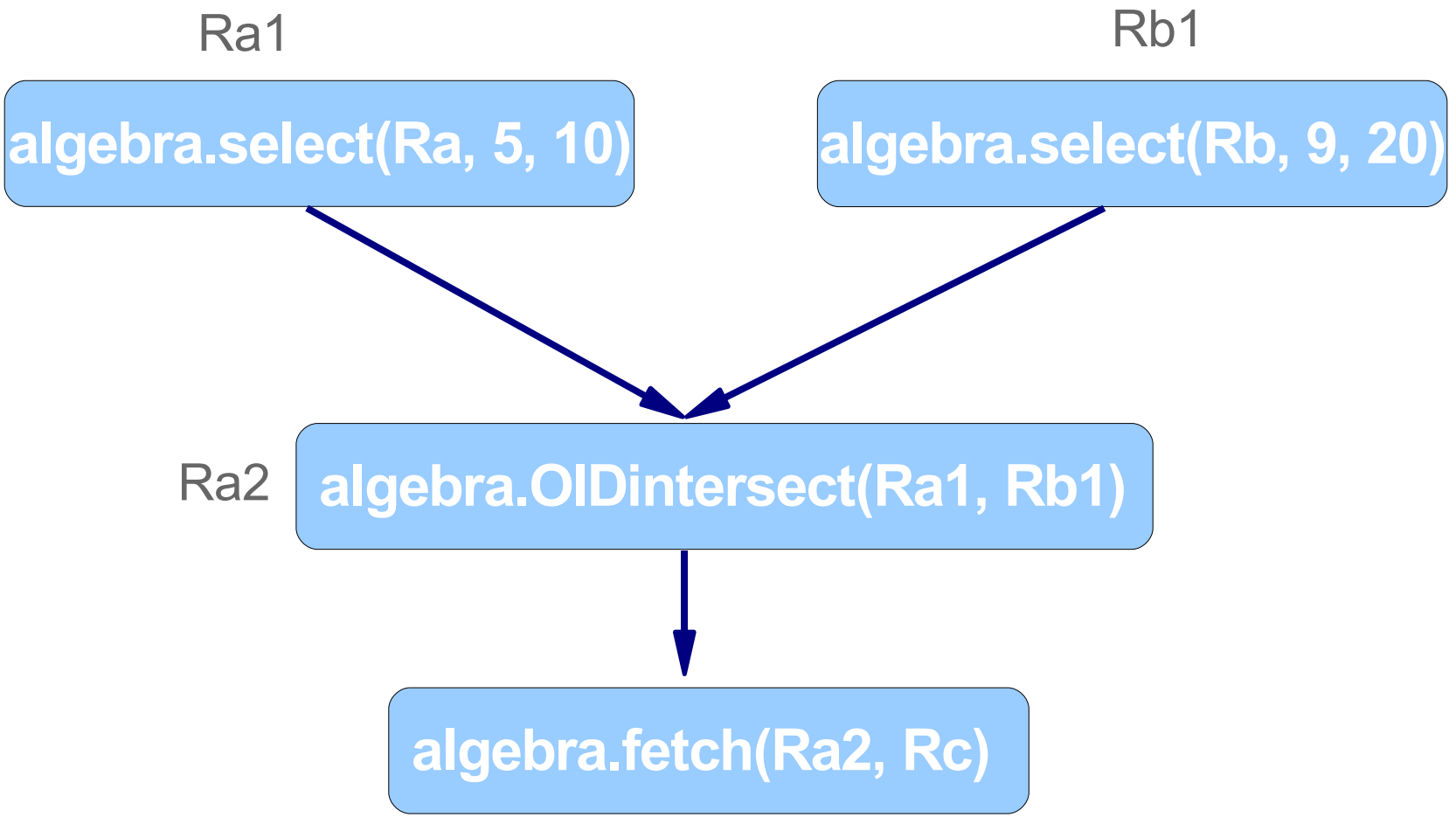
THANK YOU!

Cracking vs Indices and Sorting

- How cracking compares to a sorting strategy?
  - Sort data upfront and then use binary search
- For a sorting strategy we have to make an investment upfront
- Sorting needs prior knowledge of query workload
- Similar arguments stand for indices

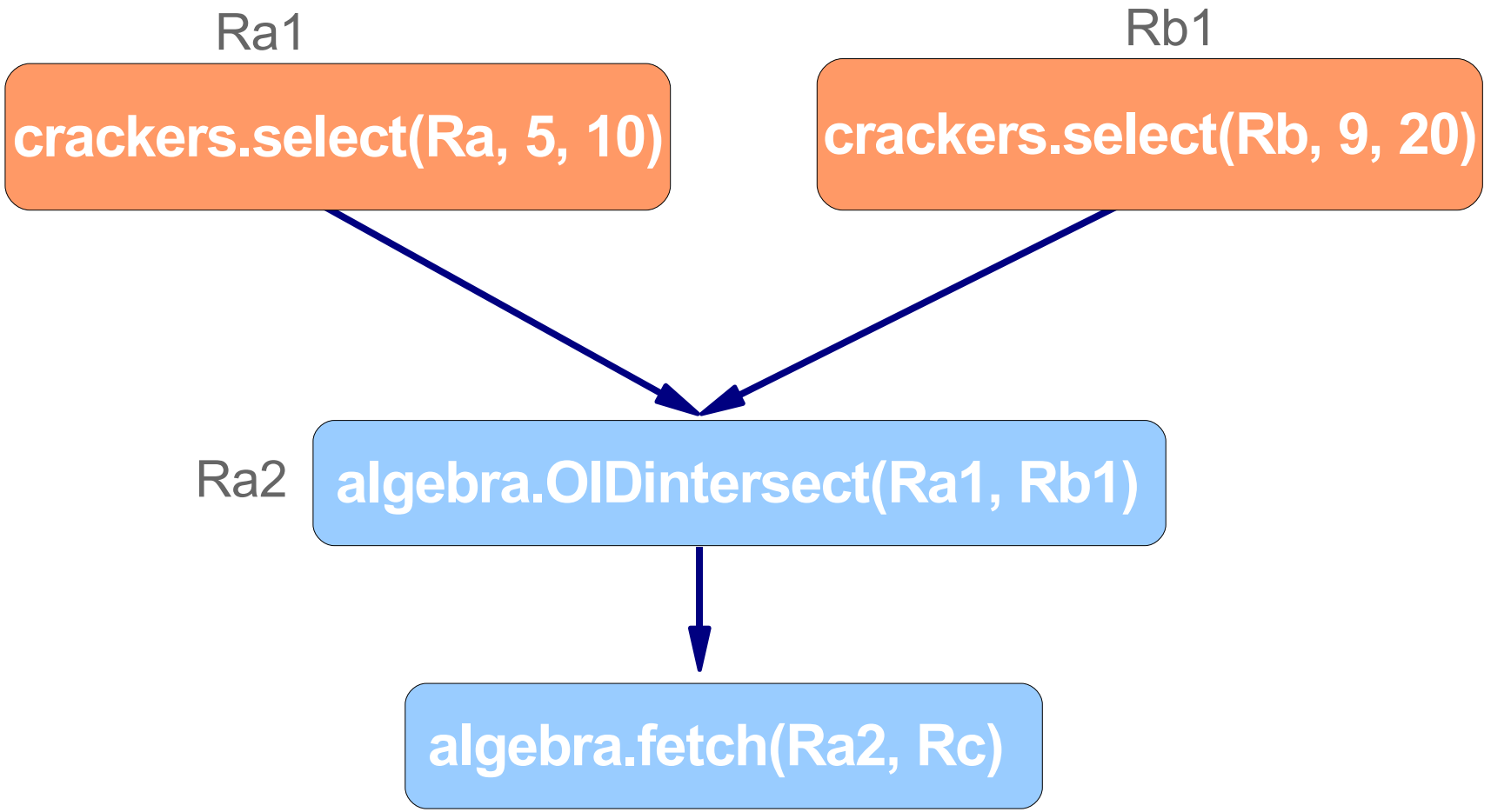
Impact on query plan

select R.c from R where 5 < R.a < 10 and 9 < R.b < 20



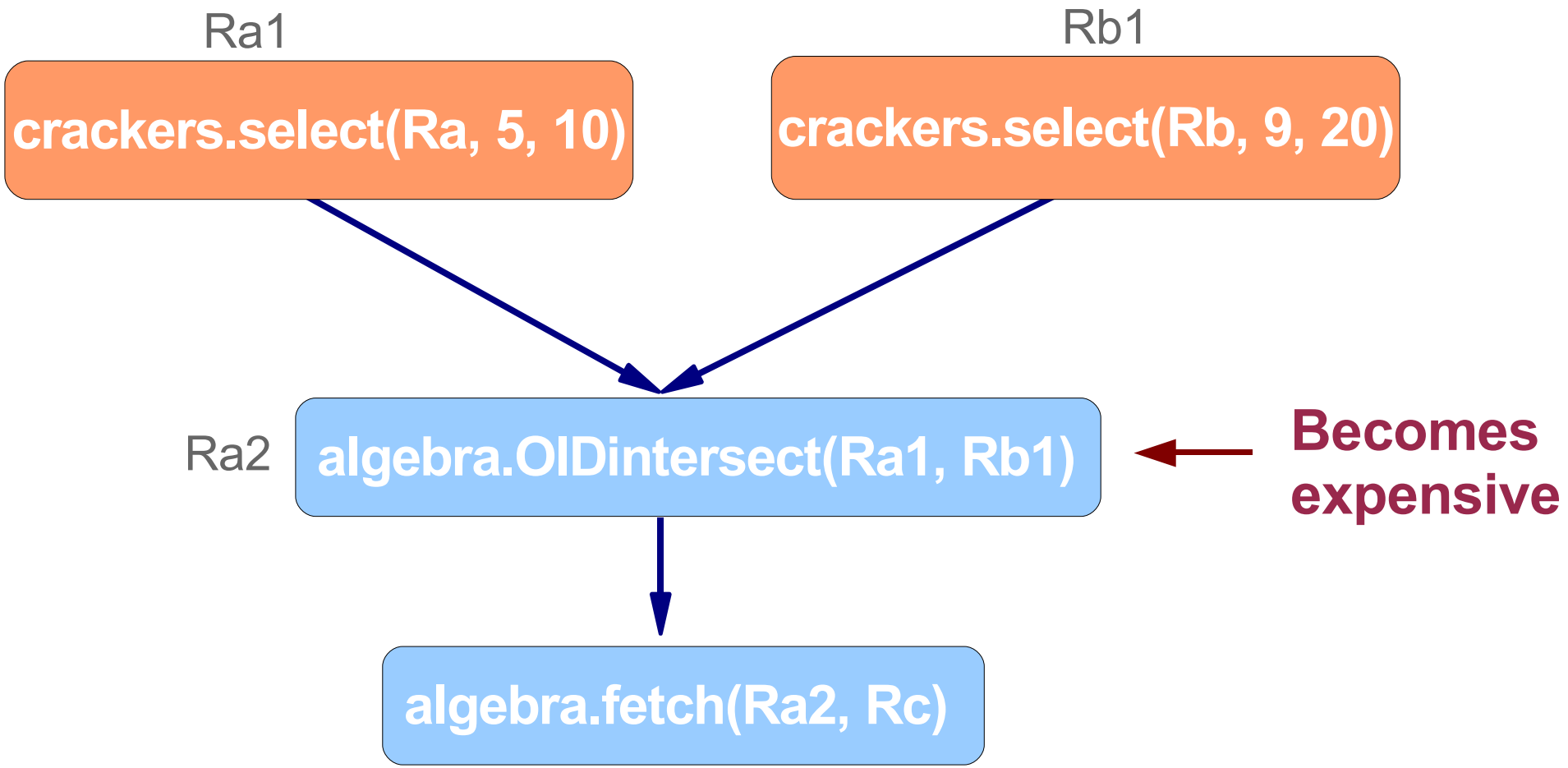
Impact on query plan

select R.c from R where 5 < R.a < 10 and 9 < R.b < 20



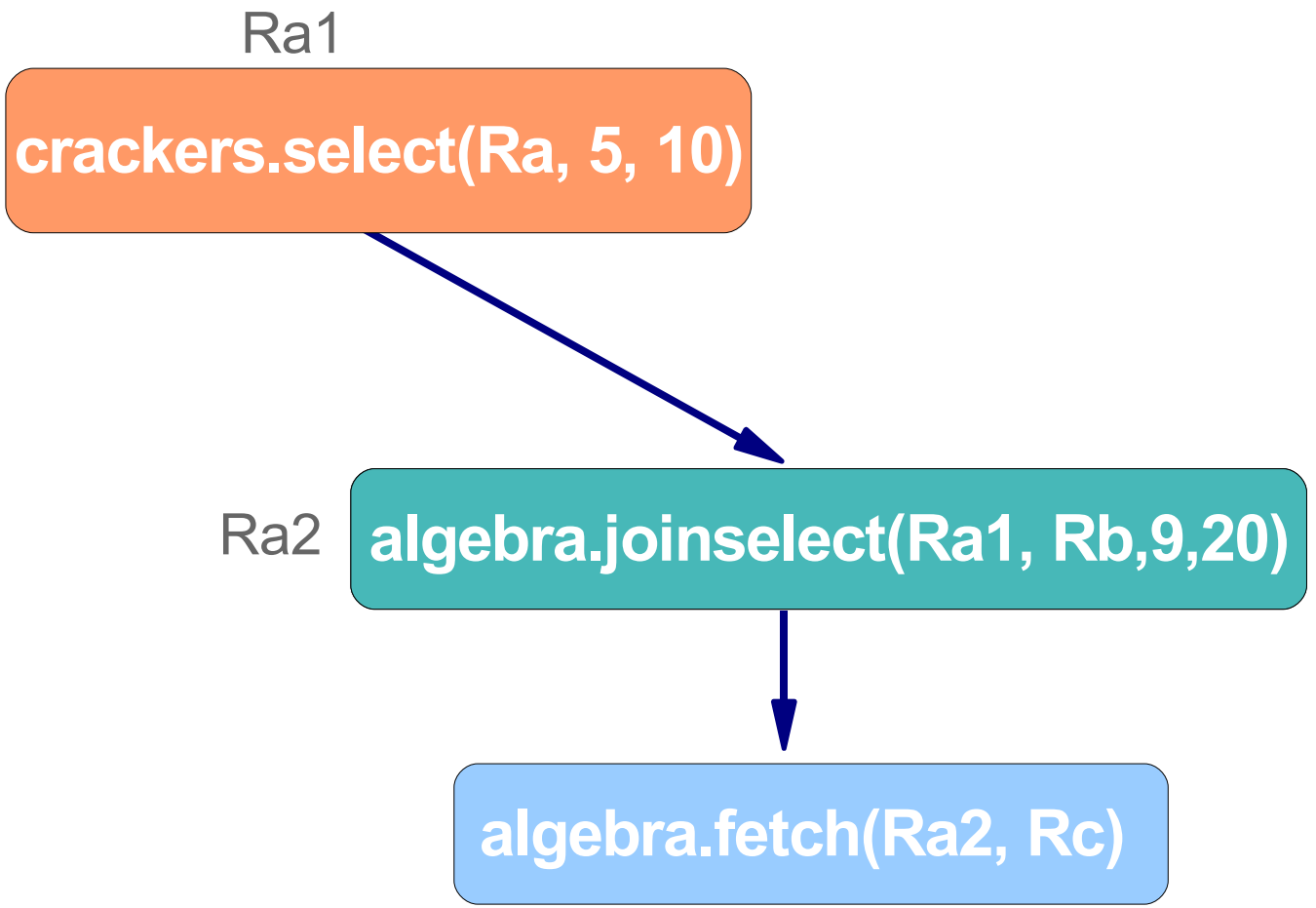
Impact on query plan

select R.c from R where 5 < R.a < 10 and 9 < R.b < 20

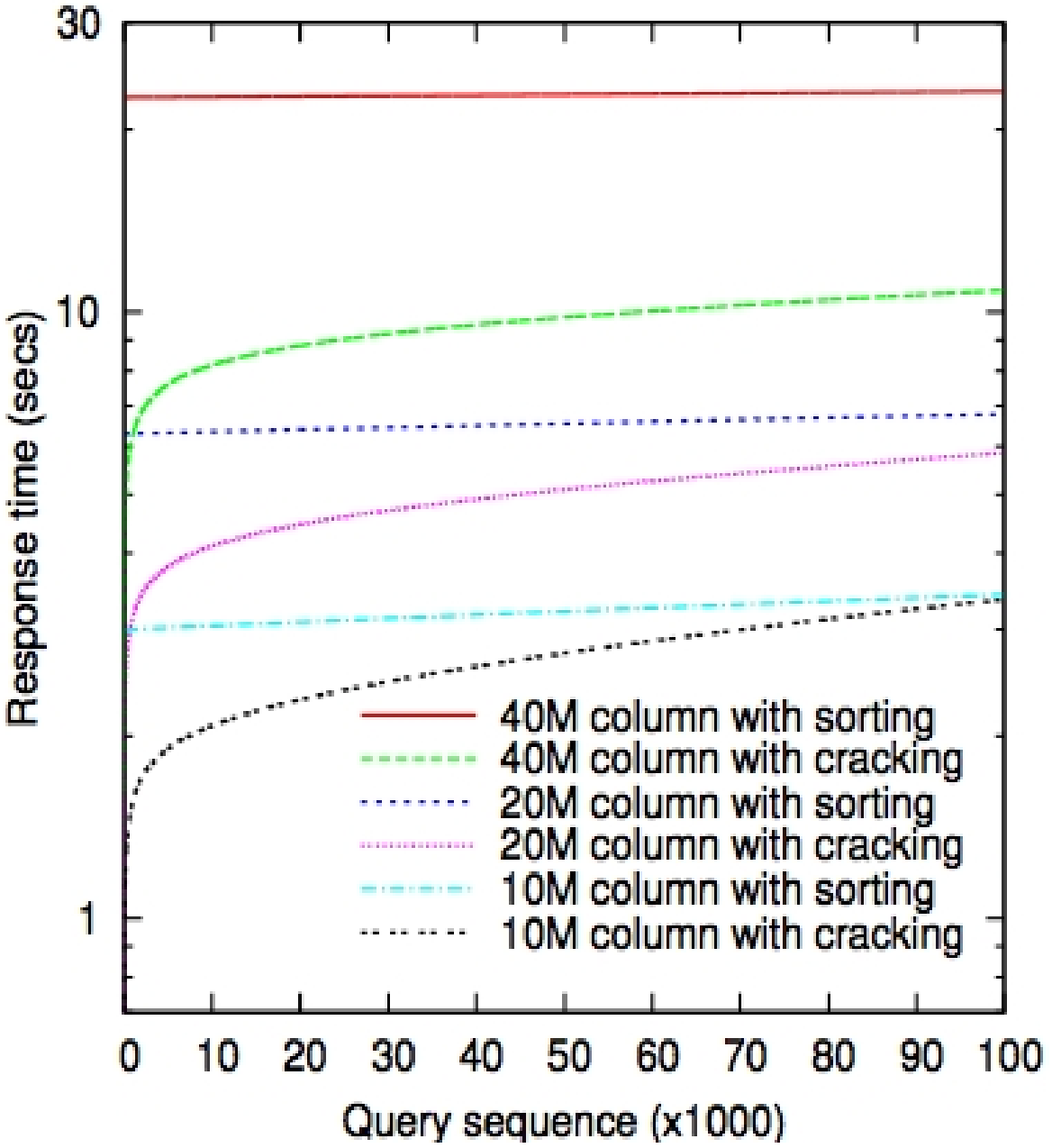


Impact on query plan

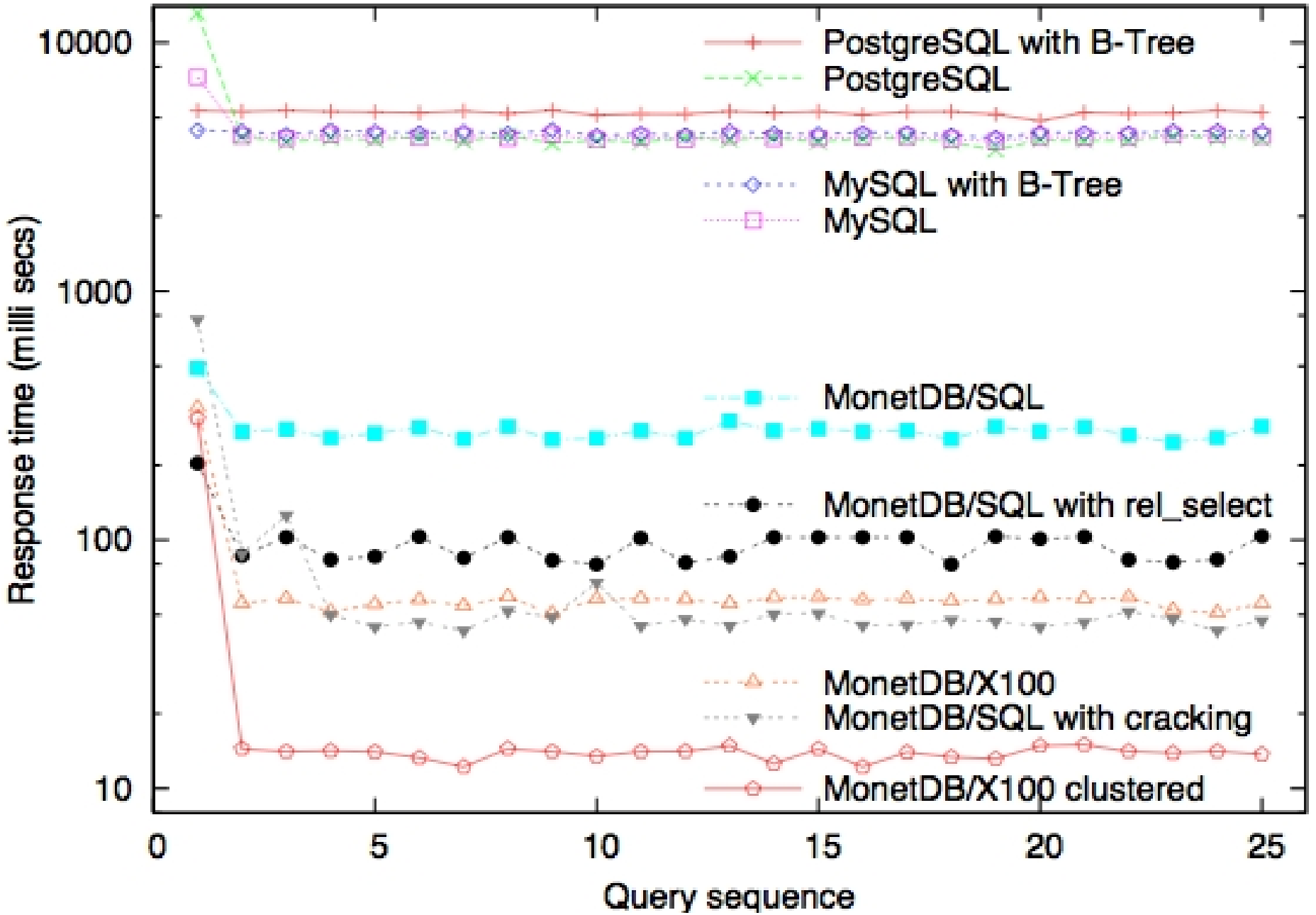
select R.c from R where 5 < R.a < 10 and 9 < R.b < 20



Scalability



TPC-H query 6


10000
1000
100
10
Response time (milli secs)
0
5
10
15
20
25
Query sequence
PostgreSQL with B-Tree
PostgreSQL
MySQL with B-Tree
MySQL
MonetDB/SQL
MonetDB/SQL with rel_select
MonetDB/X100
MonetDB/SQL with cracking
MonetDB/X100 clustered