Column-Oriented Databases

Themis Palpanas
University of Paris

Data Intensive and Knowledge Oriented Systems

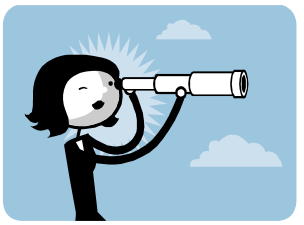
1

- thanks for slides to
  - Christoph Freytag
  - Stratos Idreos
  - Daniel Abadi

2

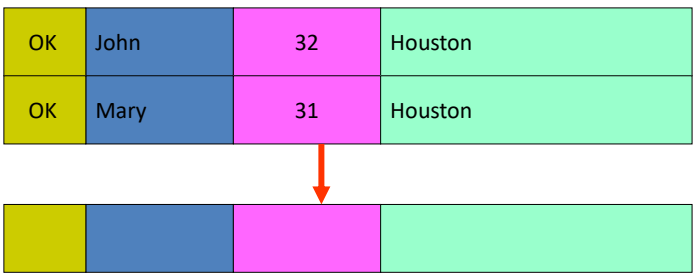
Overview

- Physical layout of data on pages
- Tuple Cracking
- Example Systems

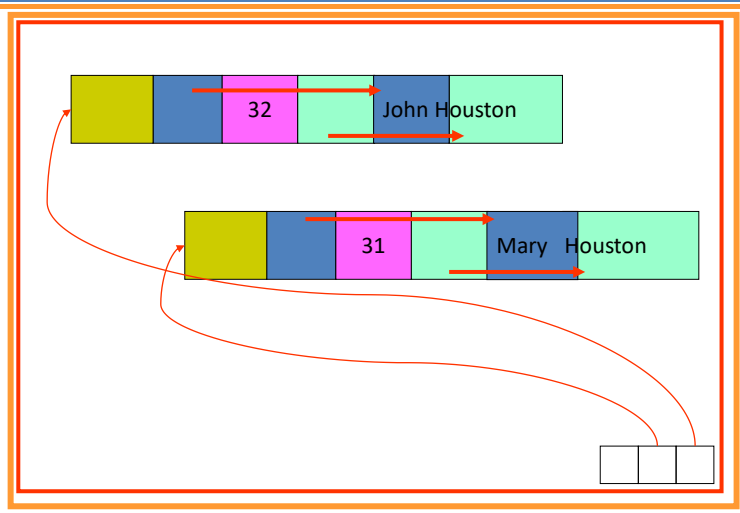


Physical data organization

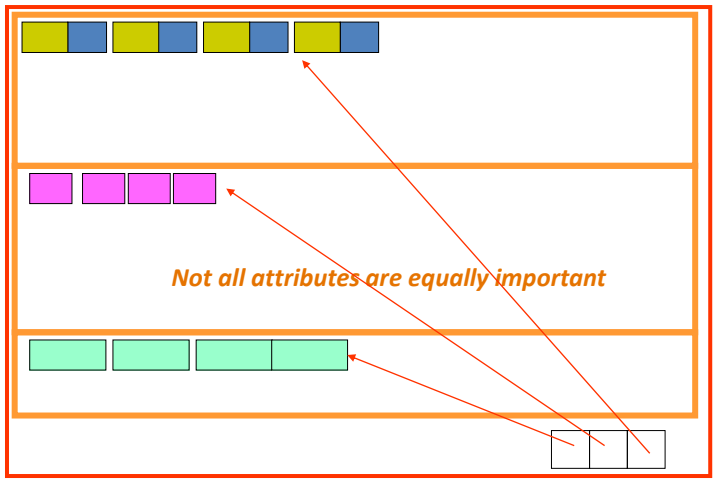
- Early 80s: Tuple oriented storage in most DBMS
  - Easy to access at the cost of wasted space



Physical data organization (2)

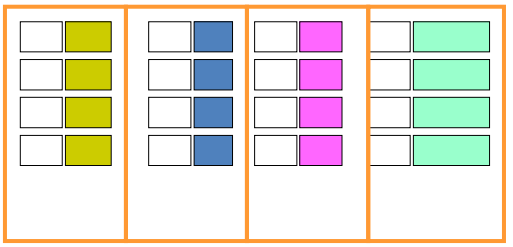


Physical data organization (3)



Alternative - Column Store

- A column orientation is as simple and acts like an array
- Attributes of a tuple are correlated by offset or ID



Row vs. Column-Stores

- Example Customer Relation

Row-Store

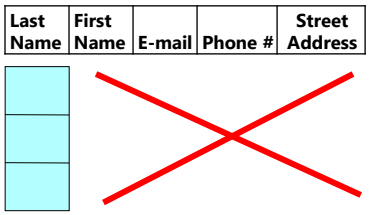
Last Name	First Name	E-mail	Phone #	Street Address

Column-Store

Last Name	First Name	E-mail	Phone #	Street Address

Example 1 – Simple Selection

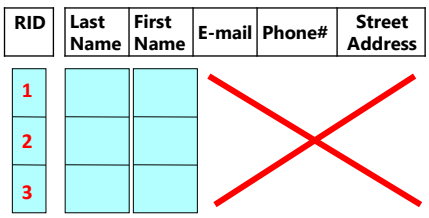
- SELECT LastName FROM Customer



  - Only relevant data are read
    - Found densely stored on page(s)

Example 2 – More Complex Selection

- SELECT LastName, FirstName FROM Customer



- Read from two different file
  - How to bring both columns together?
    - Join needed (not specified in the query!)
    - Internal ID (RID) needed

Column Store – Processing needs

- Different optimization for column oriented database
  - Compression
  - Late Materialization
    - Late Joins with Dictionary and between columns
  - Block Iteration:
    - Process all entries per block
    - Block oriented Join
  - “Invisible” Join

Column oriented storage

- Advantage
  - Each column can be more easily optimized for storage using compression schemes
  - Each column can be replicated for read-only access
  - Brings a much tighter packaging and improves transport through the memory hierarchy (still to prove!)
  - Column orientation benefits data warehousing

Row vs. Column-Stores

Row Store

- Easy to add a new record
- Might read in unnecessary data
- Logical group equals physical grouping

Column Store

- Only need to read in relevant data
- Tuple writes might require multiple seeks
- More Joins needed
- High compression possible

Column Store Systems

- Used in
  - Monet DB (CWI, Amsterdam):
  - X-Store/Vertica (MIT, Stonebraker)
    - Vertica now owned by HP – see www.vertica.com
  - SAP Sysbase IQ: for a long time

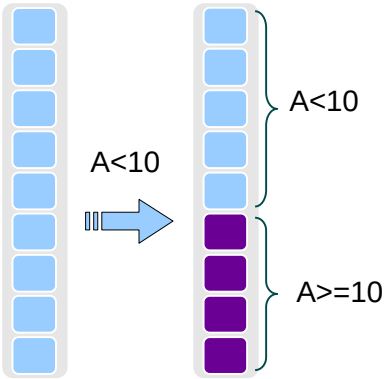
Tuple Cracking (from Monet DB)

Tuple Cracking (introduced by Monet)

- Basic idea
  - Try to use previous access pattern to optimize for the future
  - Constantly reorganize
  - Challenge: How to do it with minimal overhead

Cracking algorithms

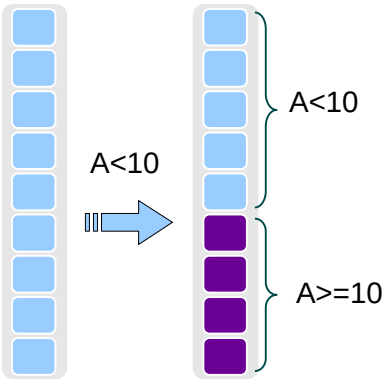
- Physical reorganization happens *per column* based on selection predicates
- Split a piece of a column in two new pieces



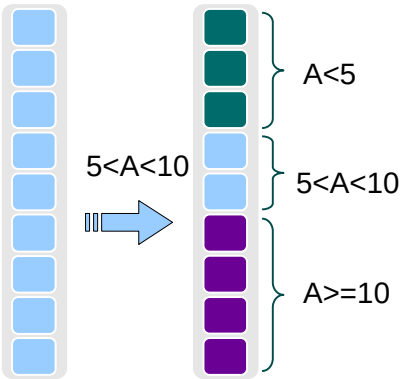
Cracking algorithms

- Physical reorganization happens per column

Split a piece of a column in two new pieces



Split a piece of a column in three new pieces



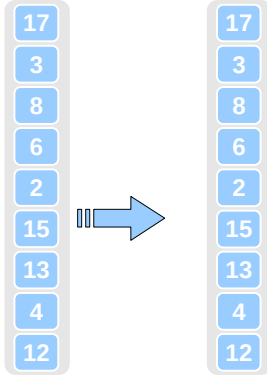
Cracking algorithms

select A>5 and A<10

17
3
8
6
2
12
13
4
15

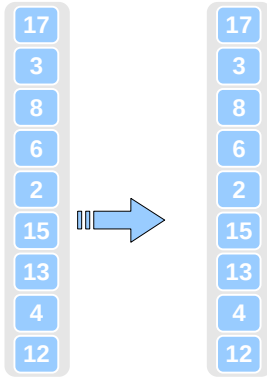
Cracking example

select A>5 and A<10



Cracking example

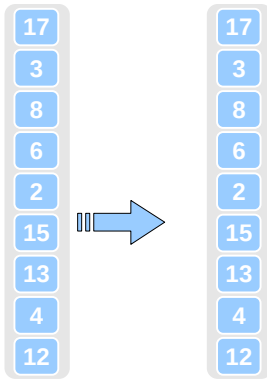
select A>5 and A<10



we need all values <5 at the beginning of the table

Cracking example

select A>5 and A<10

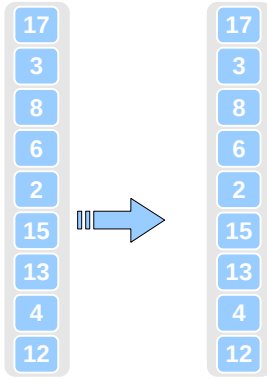


we need all values <5 at the beginning of the table

we need all values >10 at the end of the table

Cracking example

select A>5 and A<10



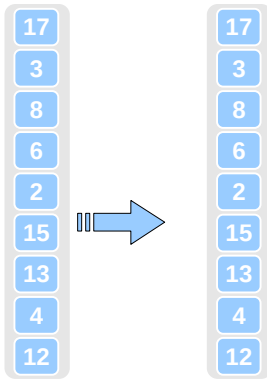
we need all values <5 at the beginning of the table

we need all values >5 AND <10 in the middle of the table

we need all values >10 at the end of the table

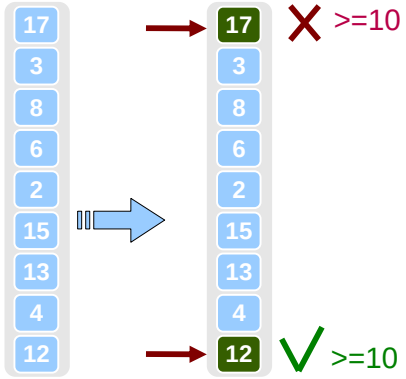
Cracking example

select A>5 and A<10



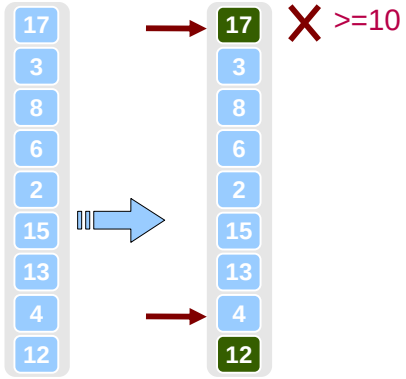
Cracking example

select A>5 and A<10



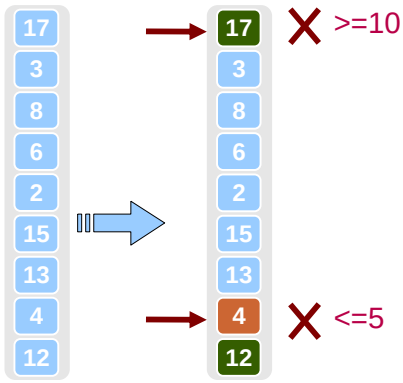
Cracking example

select A>5 and A<10



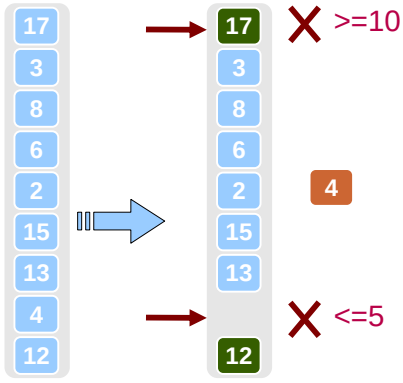
Cracking example

select A>5 and A<10



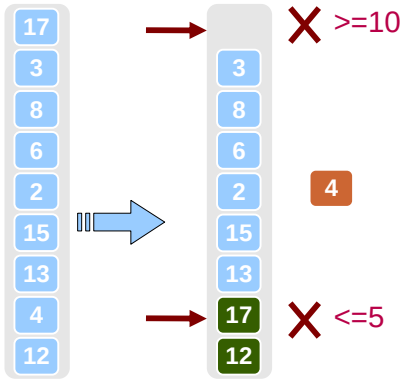
Cracking example

select A>5 and A<10



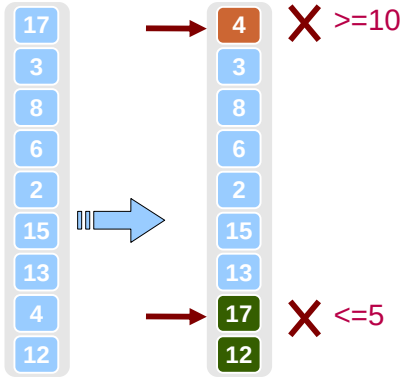
Cracking example

select A>5 and A<10



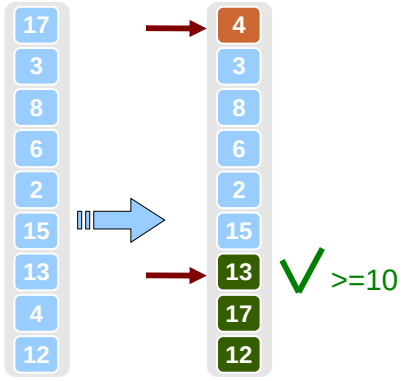
Cracking example

select A>5 and A<10



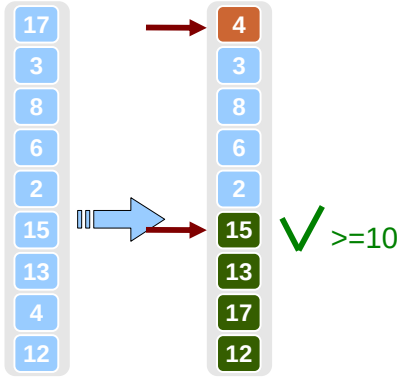
Cracking example

select A>5 and A<10



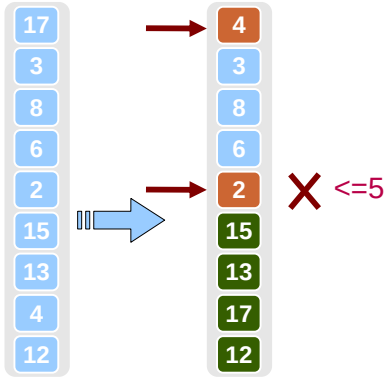
Cracking example

select A>5 and A<10



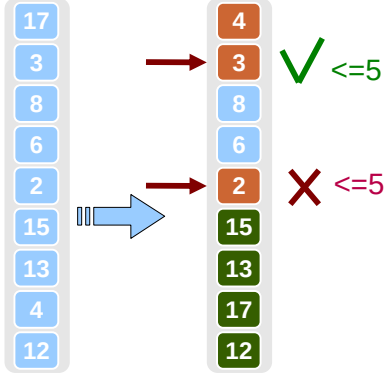
Cracking example

select A>5 and A<10



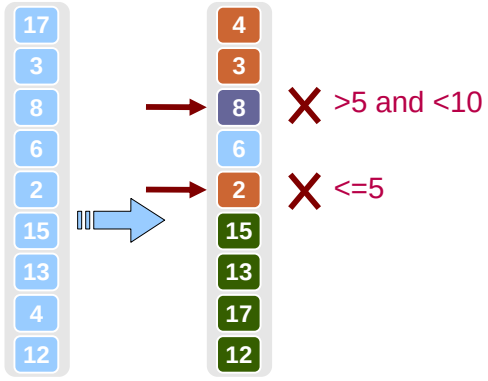
Cracking example

select A>5 and A<10



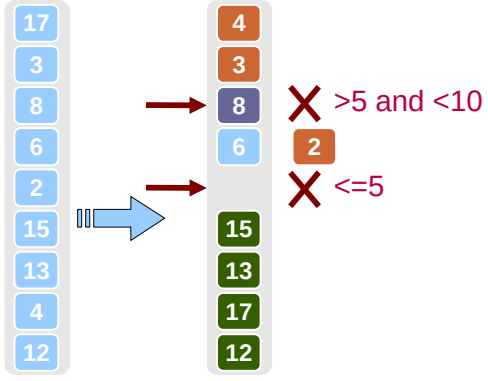
Cracking example

select A>5 and A<10



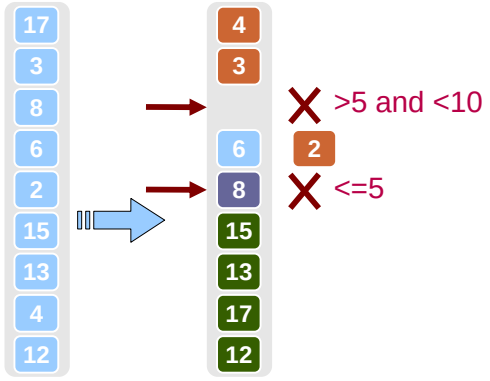
Cracking example

select A>5 and A<10



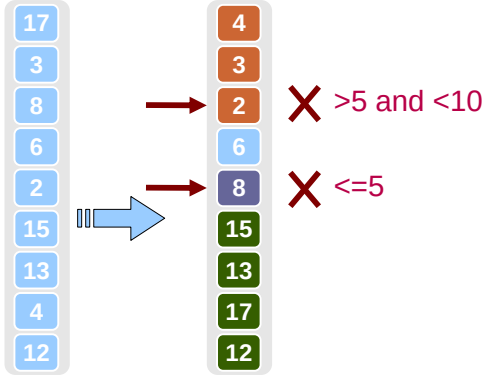
Cracking example

select A>5 and A<10



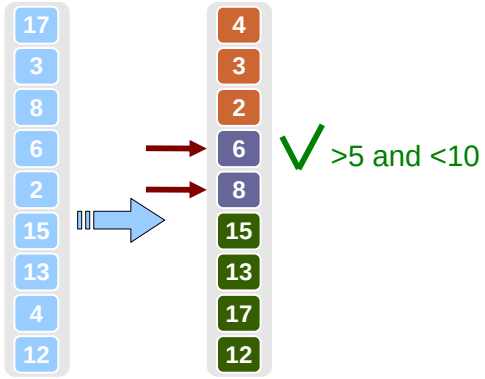
Cracking example

select A>5 and A<10



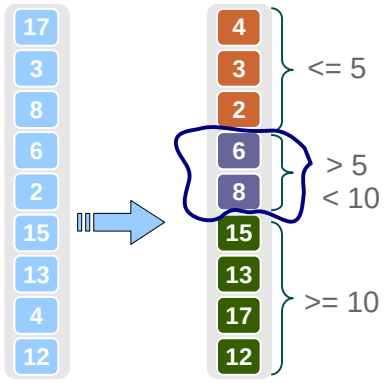
Cracking example

select A>5 and A<10

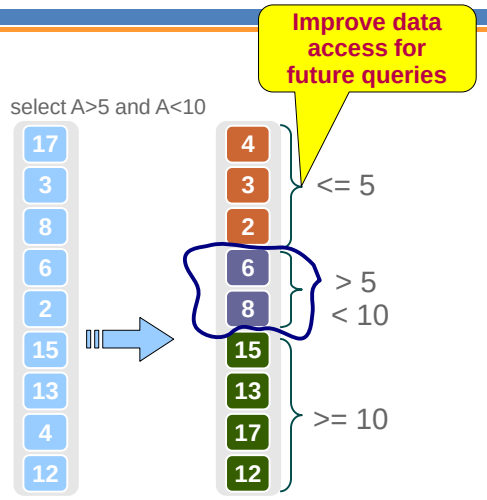


Cracking example

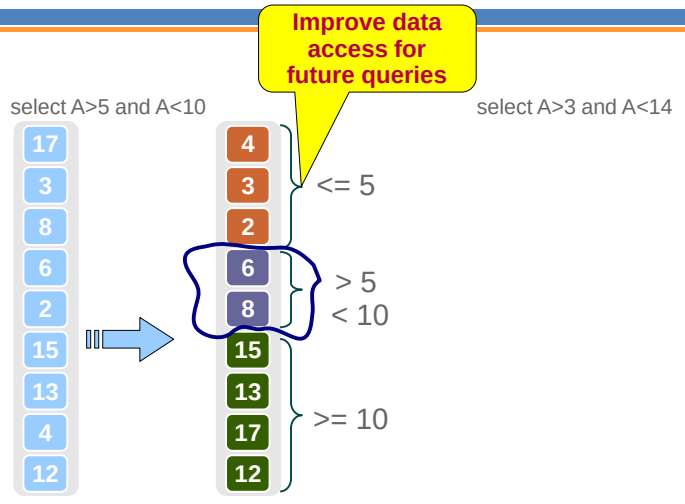
select A>5 and A<10



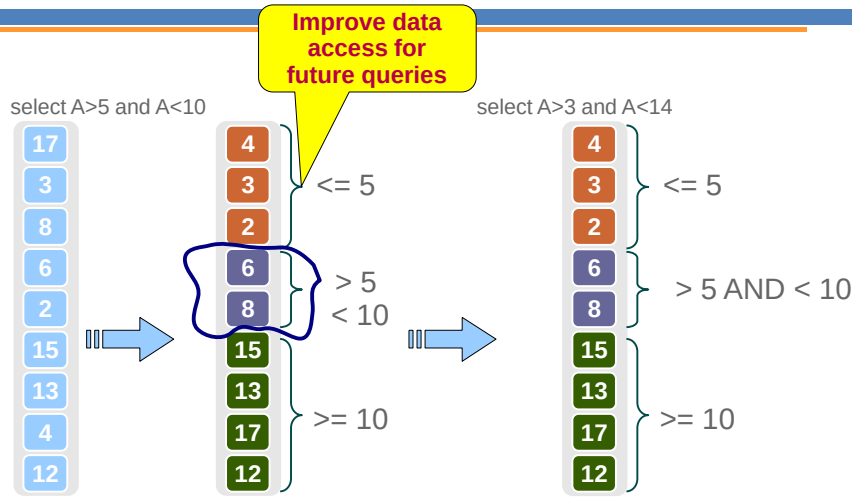
Cracking example



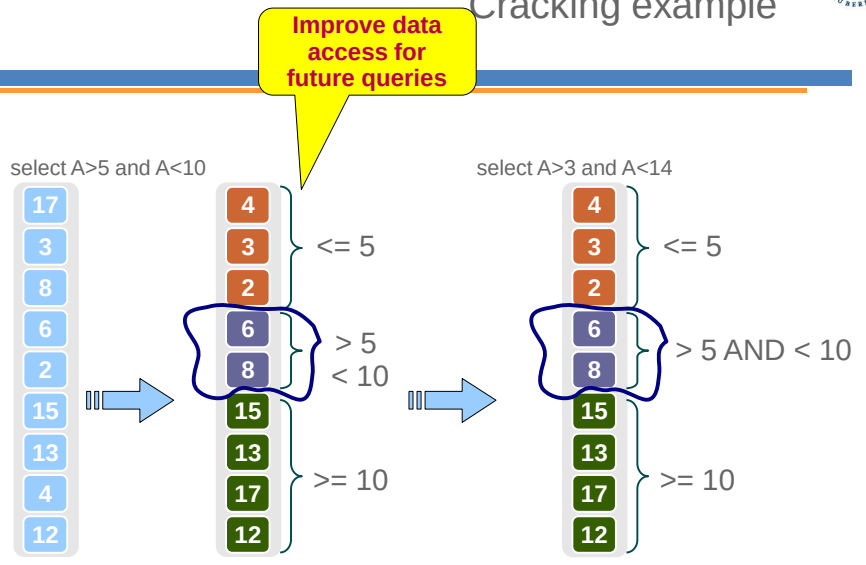
Cracking example



Cracking example

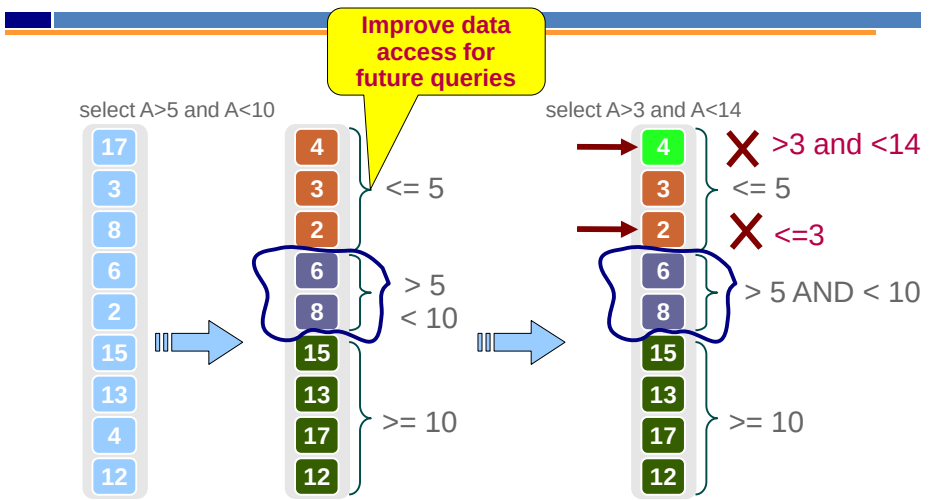


Cracking example

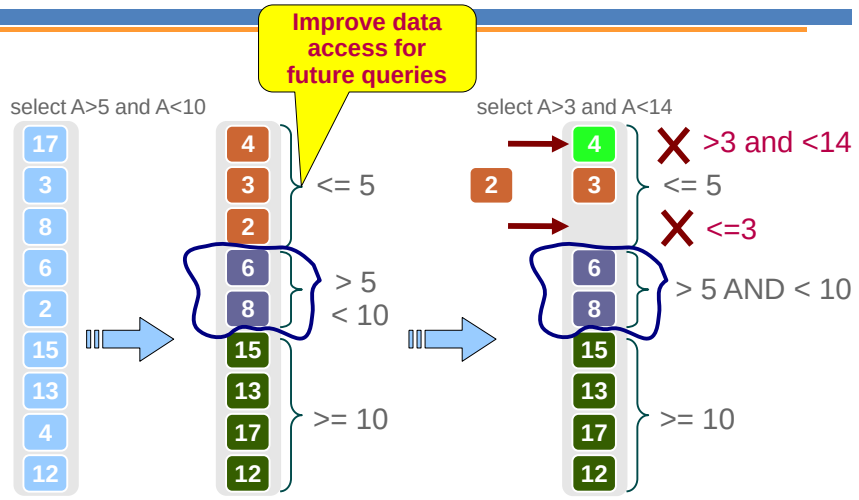


Cracking example

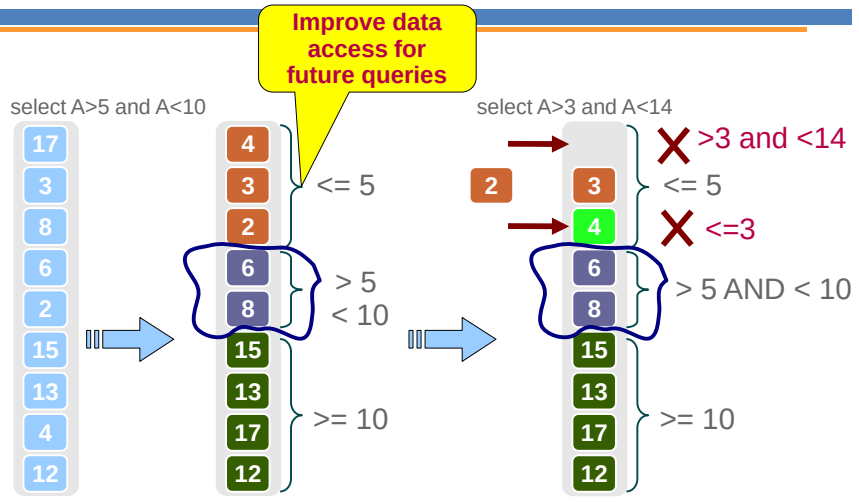
racking example



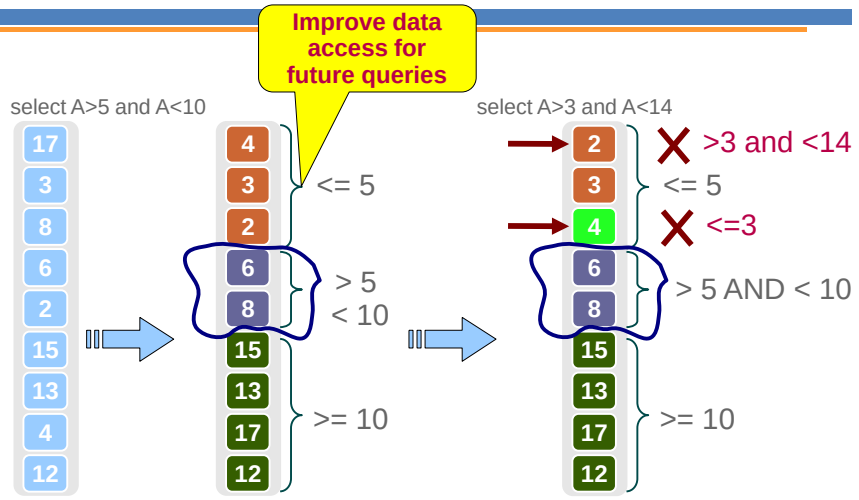
Cracking example



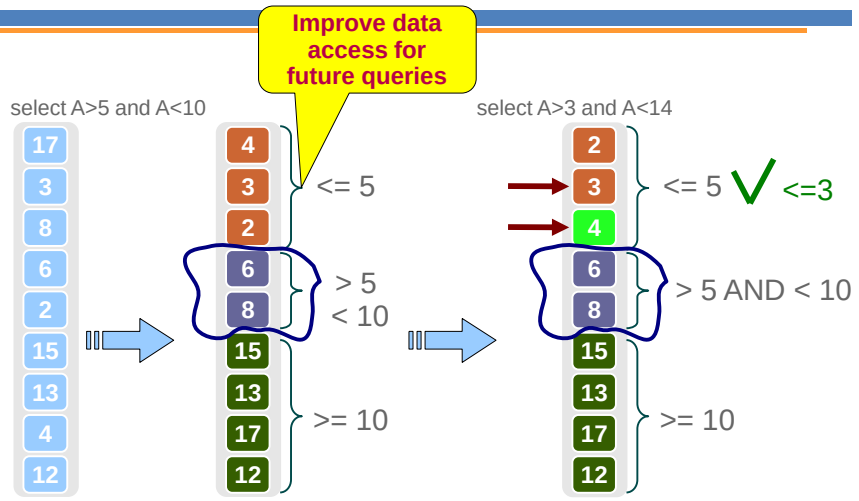
Cracking example



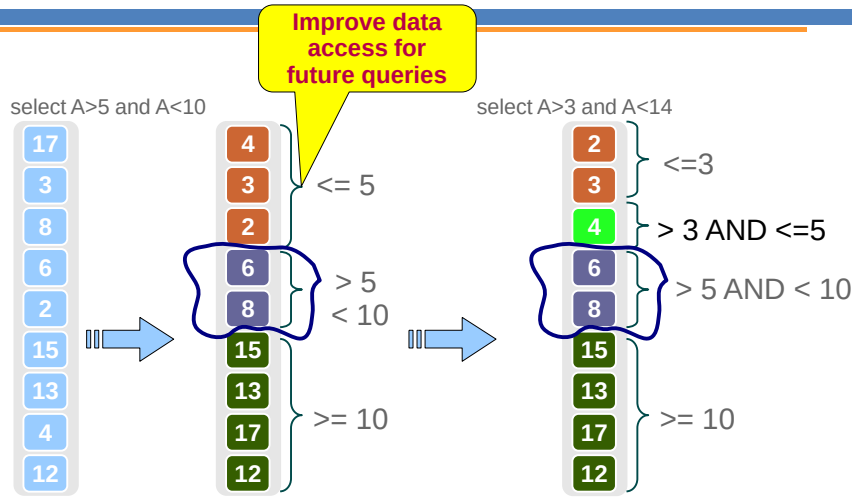
Cracking example



Cracking example



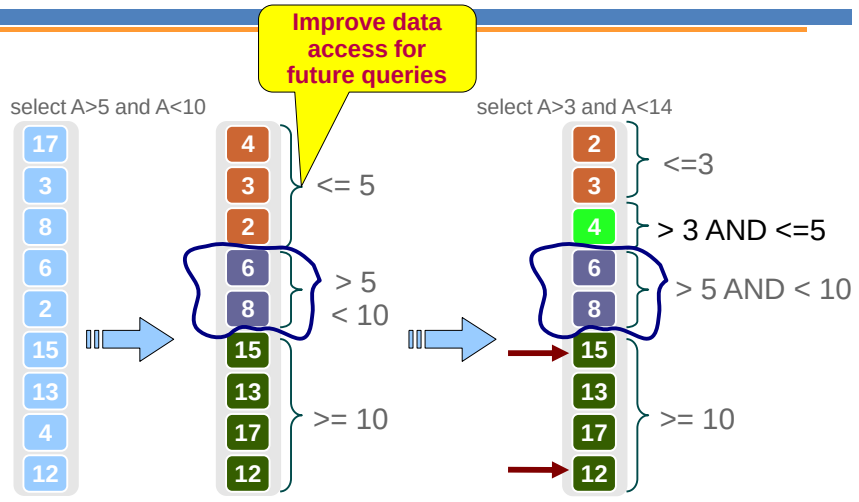
Cracking example



8. 53

53

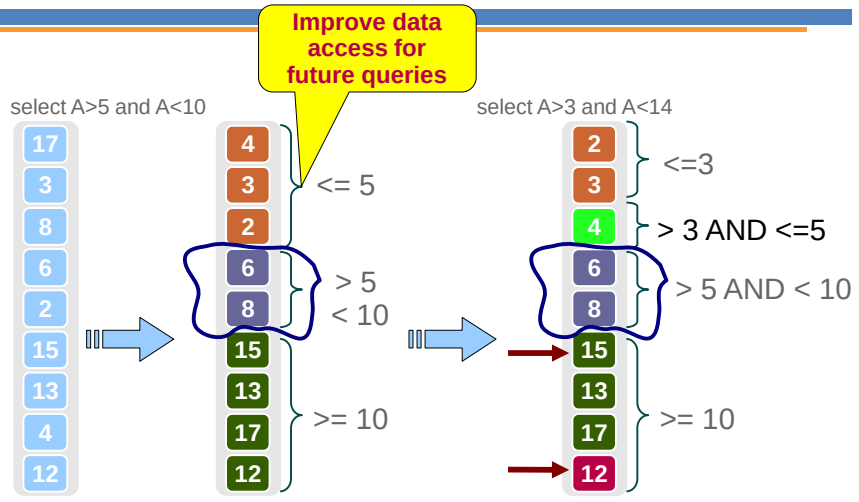
Cracking example



8. 54

54

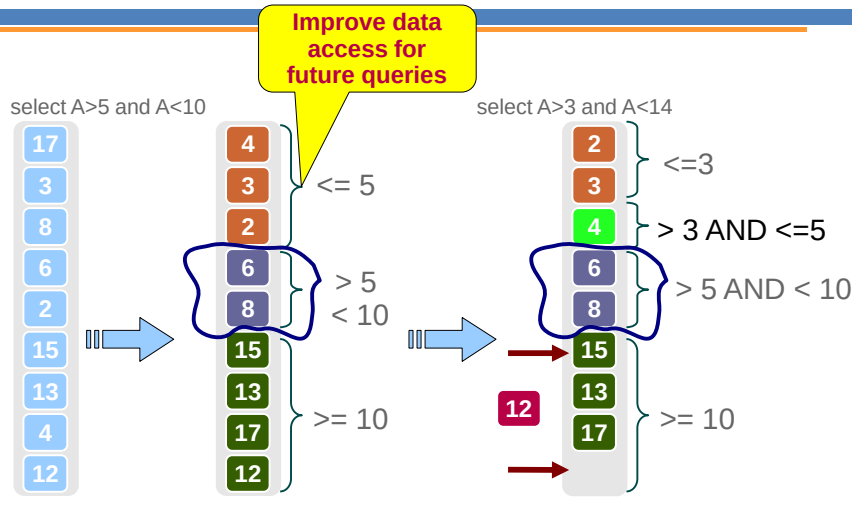
Cracking example



8. 55

55

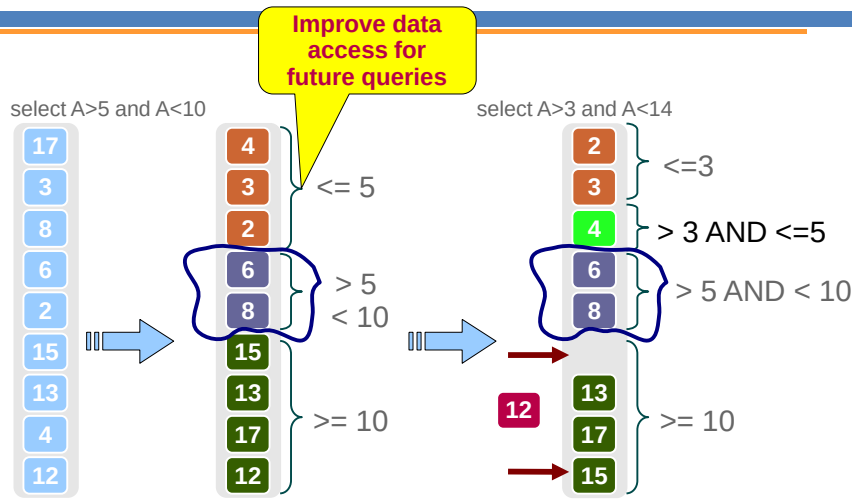
Cracking example



8. 56

56

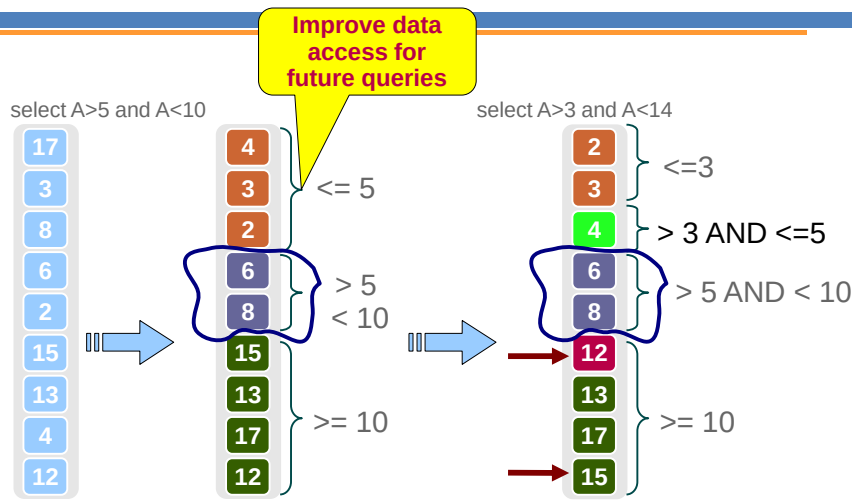
Cracking example



8. 57

57

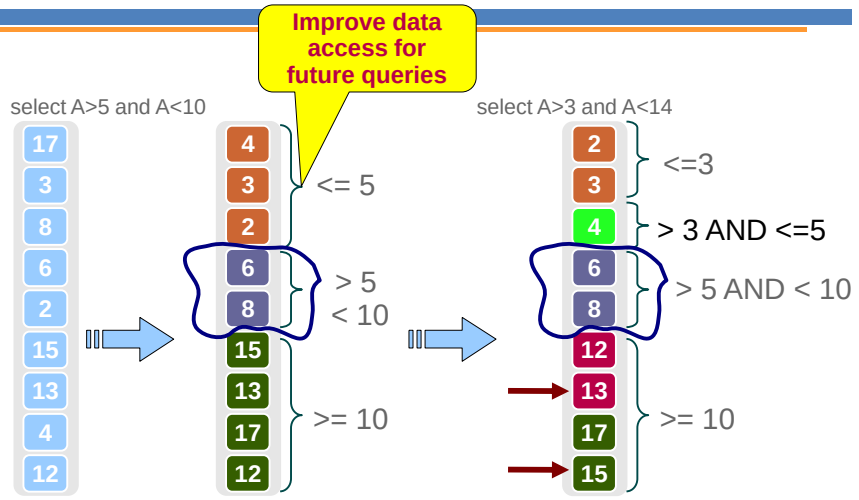
Cracking example



8. 58

58

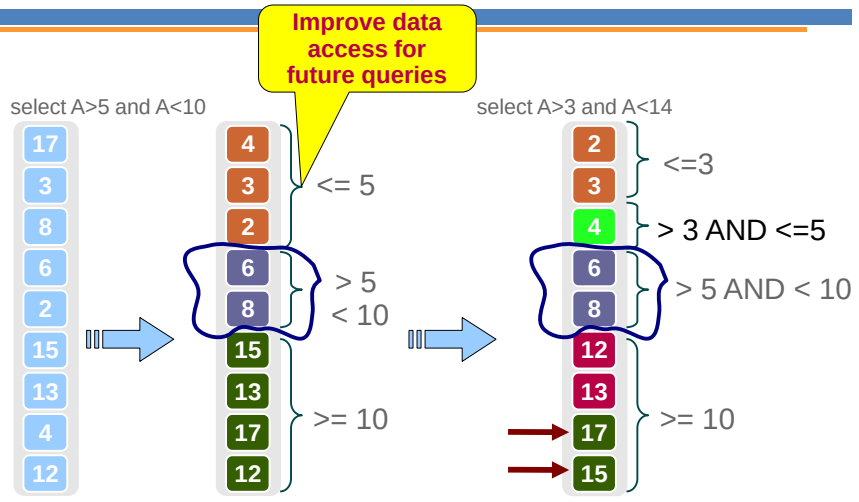
Cracking example



8. 59

59

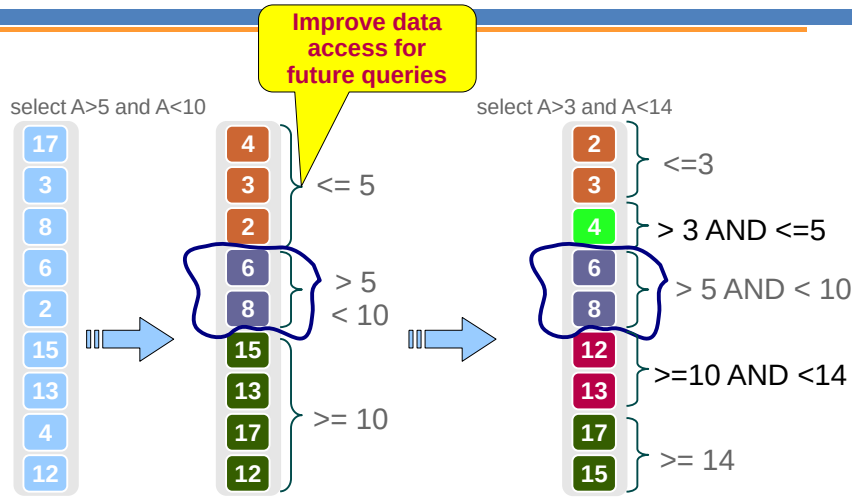
Cracking example



8. 60

60

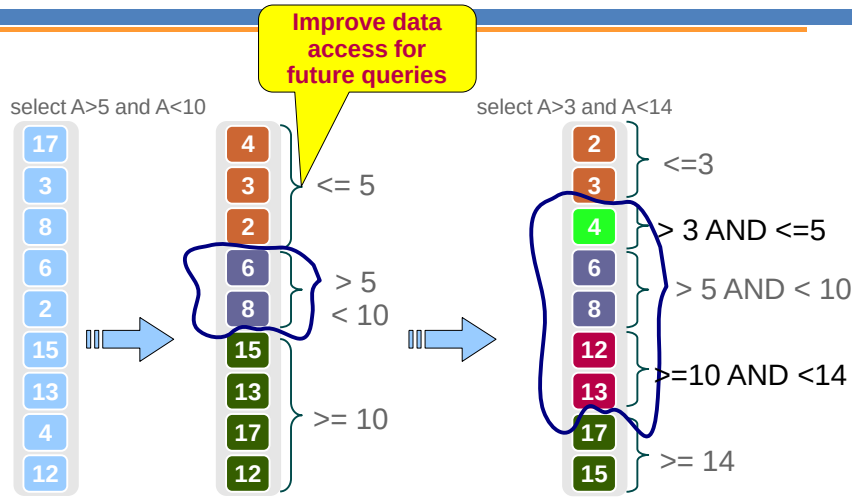
Cracking example



8. 61

61

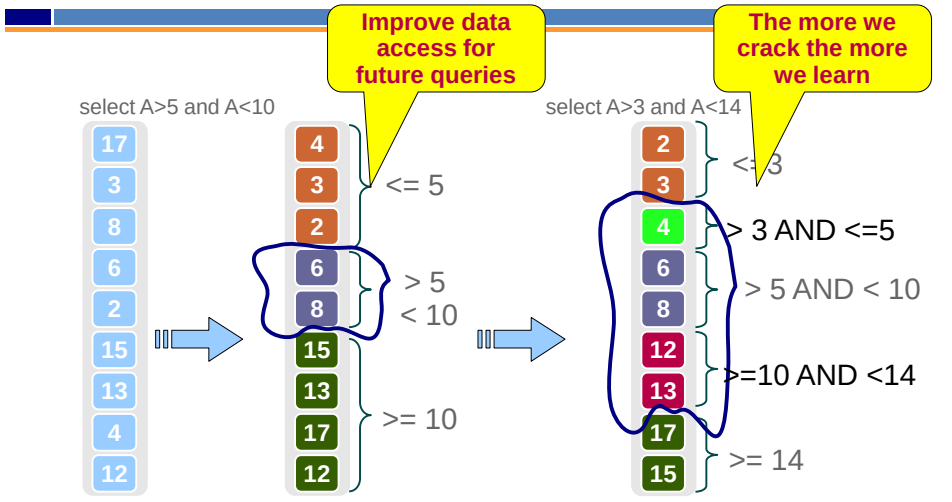
Cracking example



8. 62

62

Cracking example

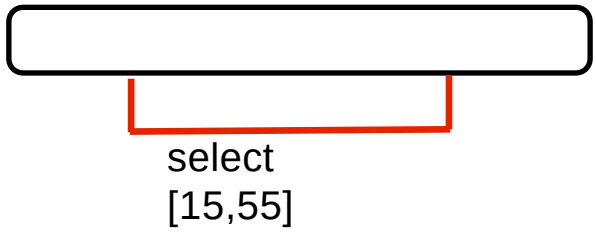


8. 63

63

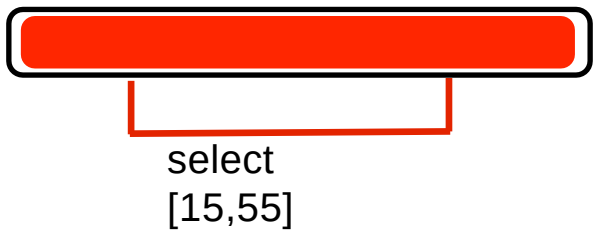
Database Cracking CIDR 2007

64



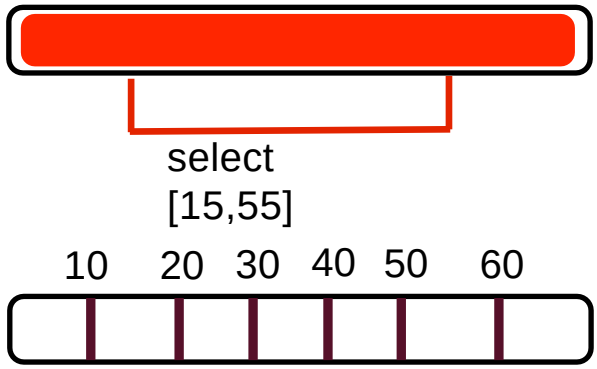
Database Cracking CIDR 2007

65



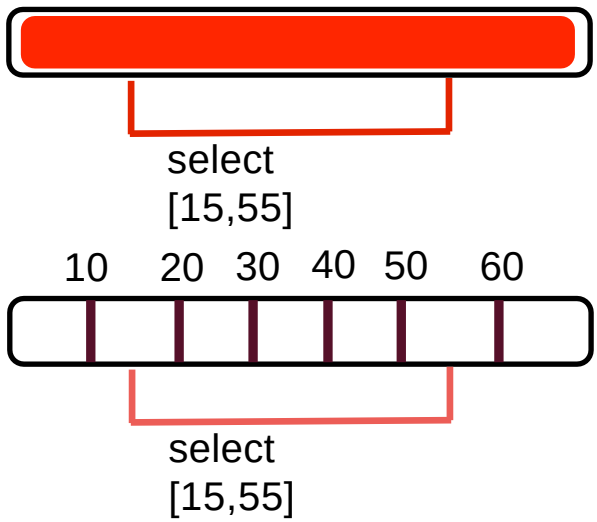
Database Cracking CIDR 2007

66



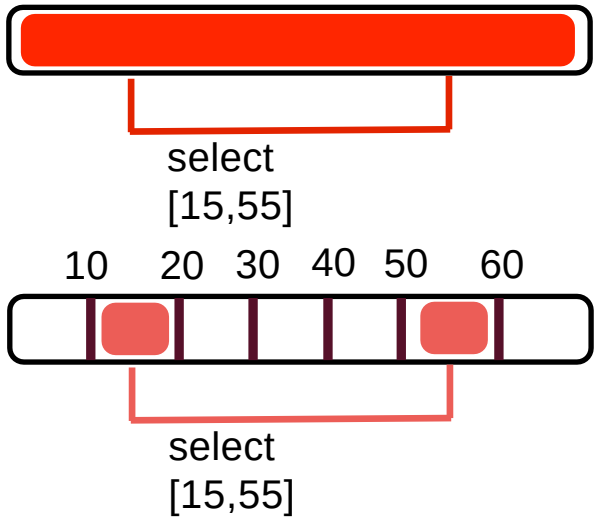
Database Cracking CIDR 2007

67



Database Cracking CIDR 2007

68

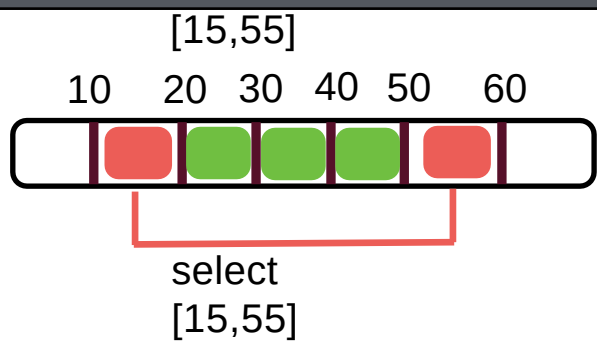


Database Cracking CIDR 2007

69

touch at most two pieces at a time

pieces become smaller and smaller



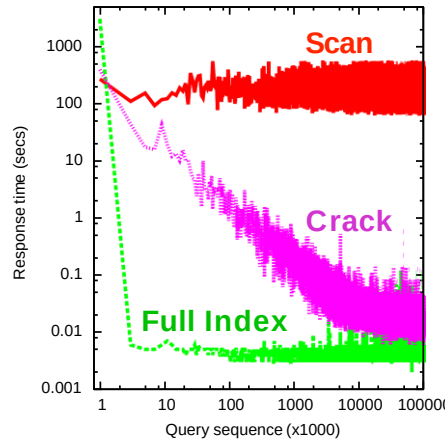
Database Cracking CIDR 2007

70

continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column



Database Cracking CIDR 2007

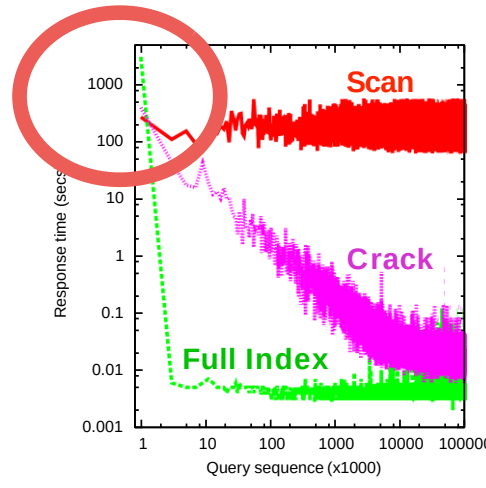
71

continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead



Database Cracking CIDR 2007

72

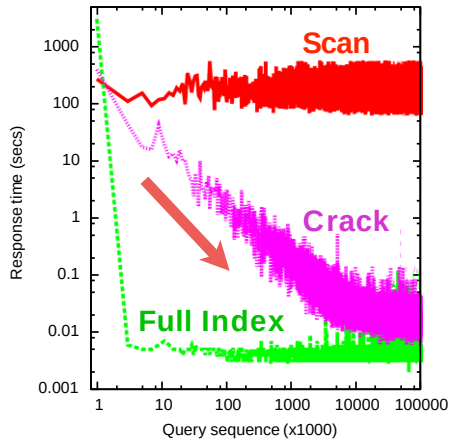
continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead

continuous improvement



Database Cracking CIDR 2007

73

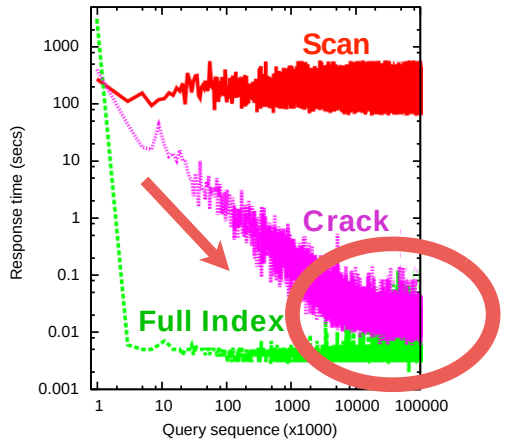
continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead

continuous improvement



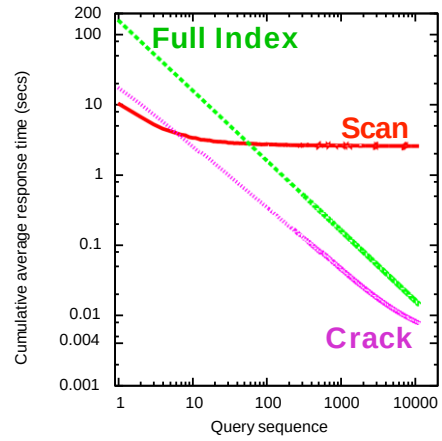
Database Cracking CIDR 2007

74

continuous adaptation

set-up

10K random selections
selectivity 10%
random value ranges
in a 30 million integer column



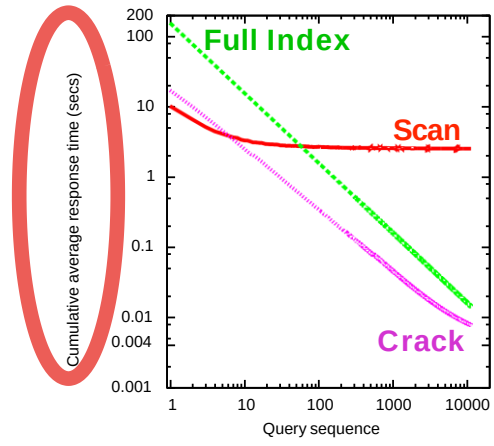
Database Cracking CIDR 2007

75

continuous adaptation

set-up

10K random selections
selectivity 10%
random value ranges
in a 30 million integer column



Database Cracking CIDR 2007

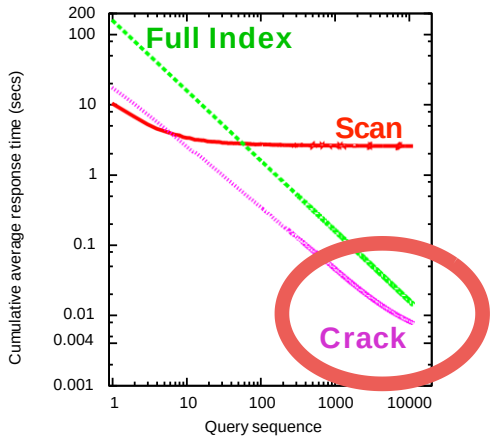
76

continuous adaptation

set-up

10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

10K queries later,
Full Index still has not
amortized the initialization costs

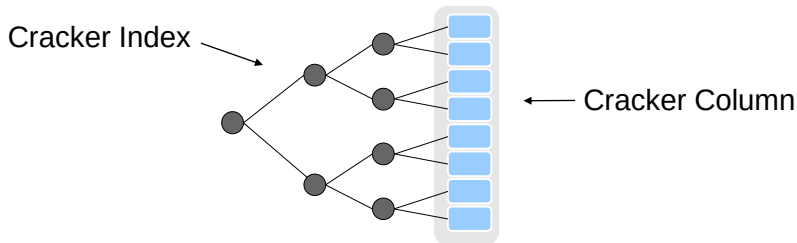


Database Cracking CIDR 2007

77

Cracking – further 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 reorganized 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



8. 78

78

Cracking - Updates

- Update to
  - cracker column
  - cracker index
- Properties:
  - Maintain the self-organization properties
  - Two issues to solve:
    - When
    - How

When to propagate updates

Updates become part of query processing

- When an update arrives, it is not applied
- For each cracker column there is
  - a pending insertions column
  - and a pending deletions column
- Pending updates are applied only when a query needs the specific values

Update aware Select Operator

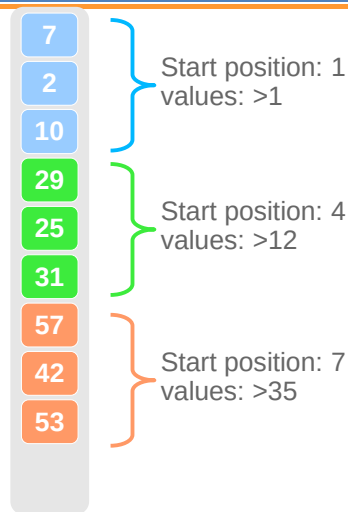
- Approach:
  - Select operator applies needed updates before cracking
- Steps of the Select operator:
  - Search the pending insertions column
  - Search the pending deletions column
  - If Steps 1 or 2 find tuples run update algorithm
  - Search the cracker index
  - Physically reorganize the cracker column
  - Update the cracker index
  - Return a slice of the cracker column

Problem: Merging in Update

Insert a new tuple with value 9

9

The new tuple belongs to the blue piece



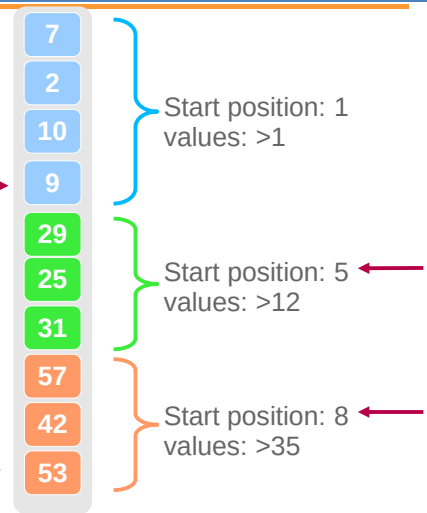
First Approach: Merging

The new tuple belongs to the blue piece

Pieces in the cracker column are ordered

Tuples inside a piece are not ordered

Shifting is not a viable solution

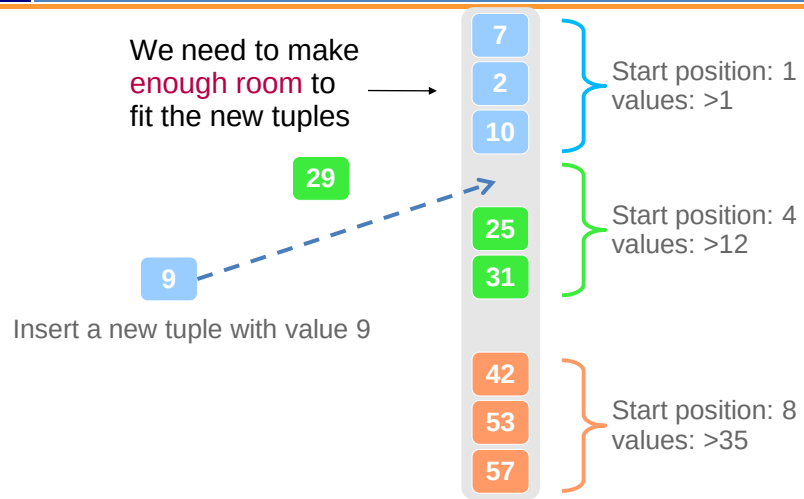


8. 83

83

Second Approach: Merging by Hopping

We need to make enough room to fit the new tuples



Insert a new tuple with value 9

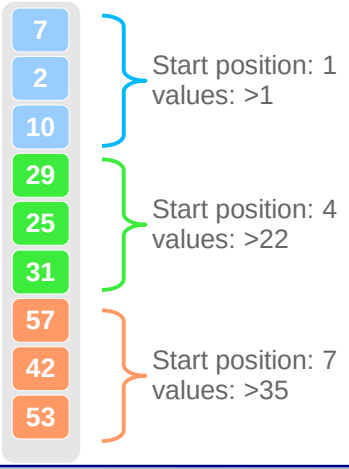
8. 84

84

Third Approach: Minimalism Principle

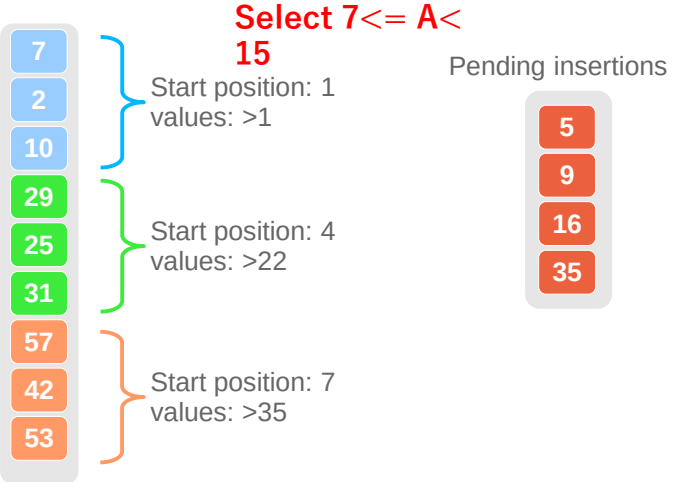
- Touch only the pieces that are relevant for the current query

Select 7<= A< 15

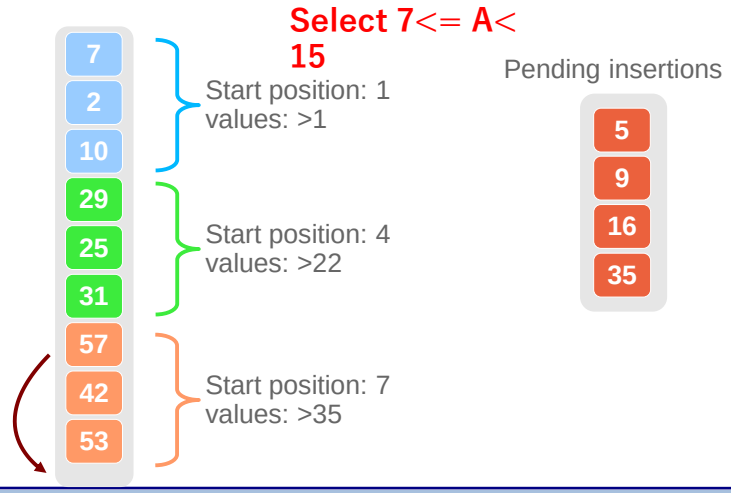


Minimalism Principle: Ripple Approach

Select 7<= A< 15



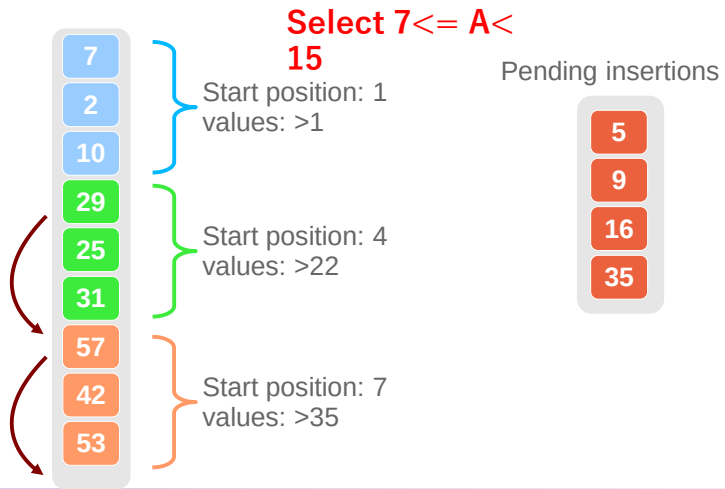
Minimalism Principle: Ripple Approach



8. 87

87

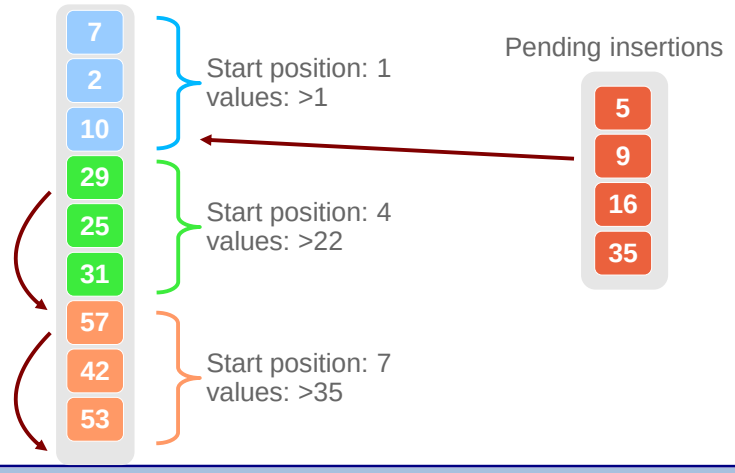
Minimalism Principle: Ripple Approach



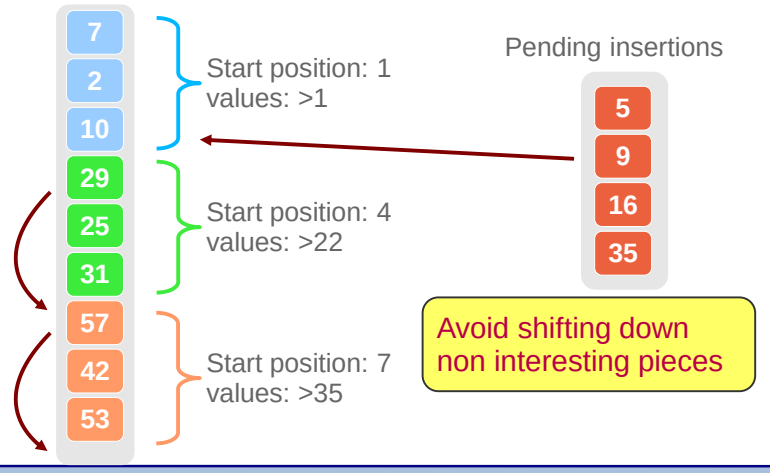
8. 88

88

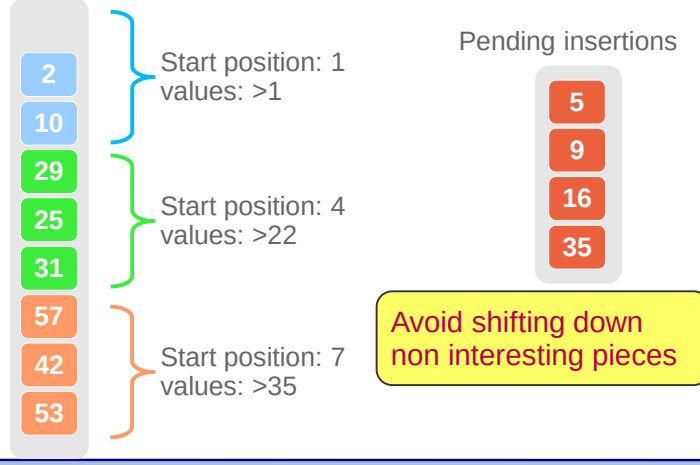
Minimalism Principle: Ripple Approach



Minimalism Principle: Ripple Approach



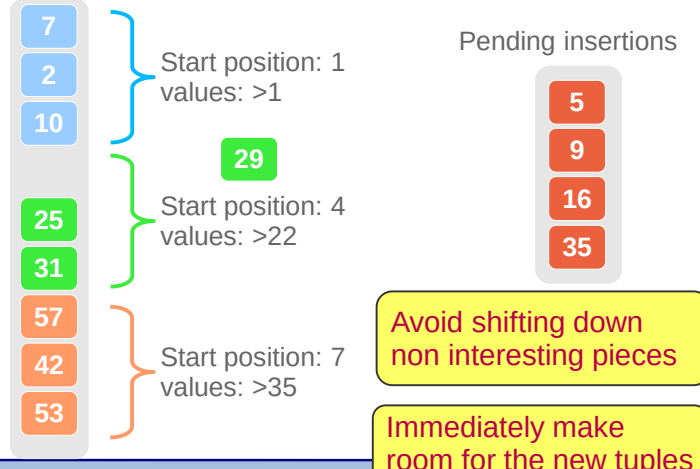
Minimalism Principle: Ripple Approach



8. 91

91

Minimalism Principle: Ripple Approach



8. 92

92

The Ripple

7
2
10
9
25
31
57
42
53

Start position: 1
values: >1

29

Start position: 4
values: >22

Start position: 7
values: >35

Pending insertions

5
16
35

Avoid shifting down non interesting pieces

Immediately make room for the new tuples

8. 93

93

The Ripple

7
2
10
9
25
31
57
42
53

Start position: 1
values: >1

29

Start position: 4
values: >22

Start position: 7
values: >35

Pending insertions

5
16
35

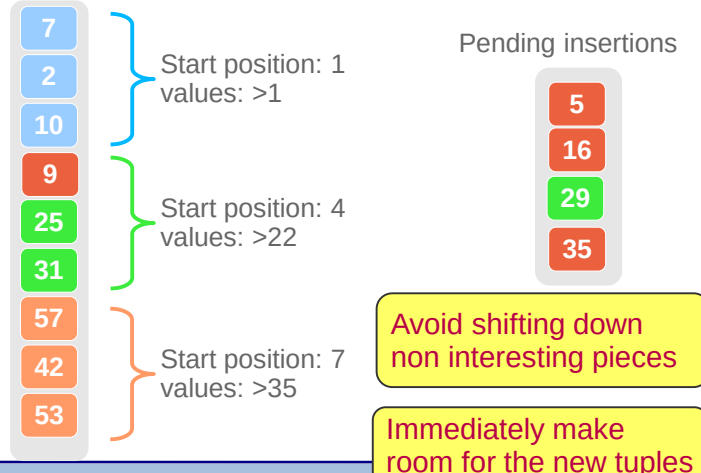
Avoid shifting down non interesting pieces

Immediately make room for the new tuples

8. 94

94

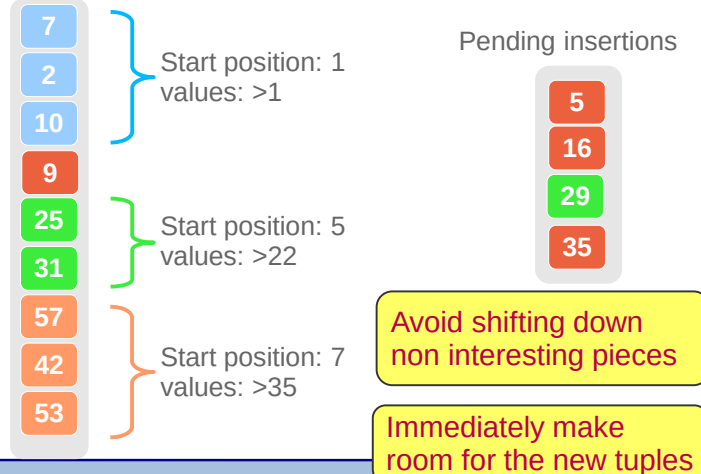
Minimalism Principle: Ripple Approach



8. 95

95

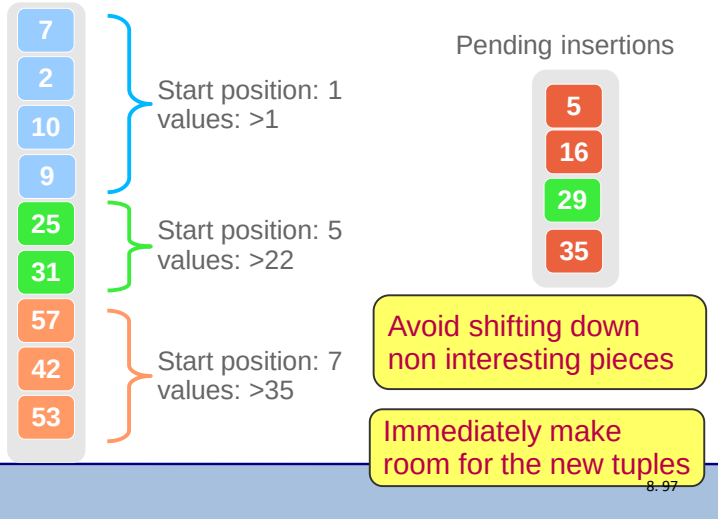
Minimalism Principle: Ripple Approach



8. 96

96

Minimalism Principle: Ripple Approach



Questions???