MonetDB,
Cracking and recycling

Martin Kersten

CWI

Amsterdam

Try to maximize performance

Present

Paste

Materialized Views

Potency

B-tree, Hash Indices

Cracking

MonetDB

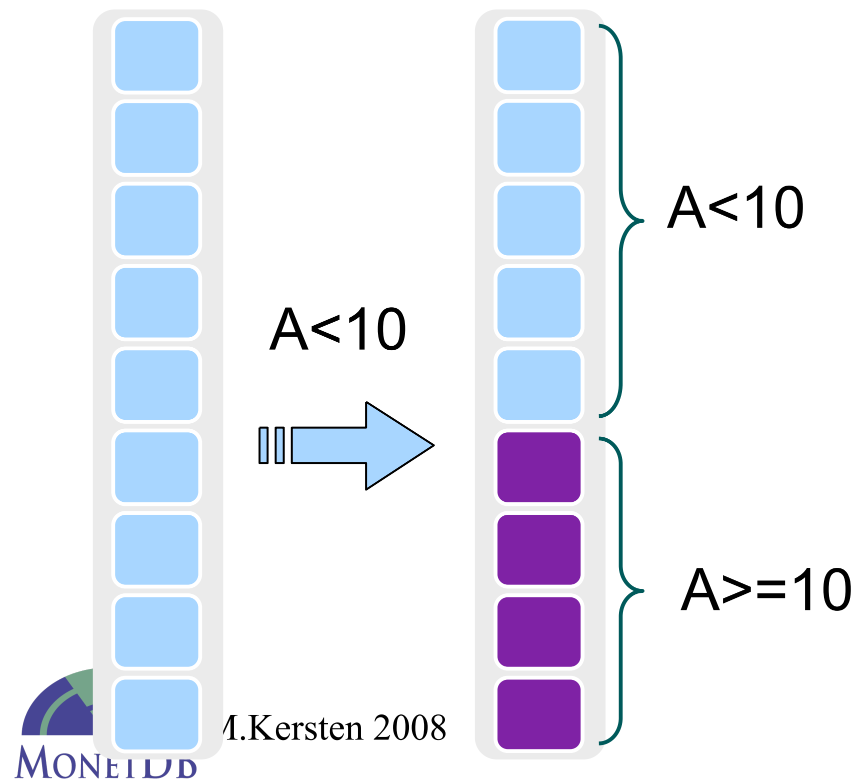
Find a trusted fortune teller

- Indices in database systems focus on:
  - All tuples are equally important for fast retrieval
  - There are ample resources to maintain indices
- MonetDB cracks the database into pieces based on actual query load

Cracking algorithms

Physical reorganization happens per column based on selection predicates.

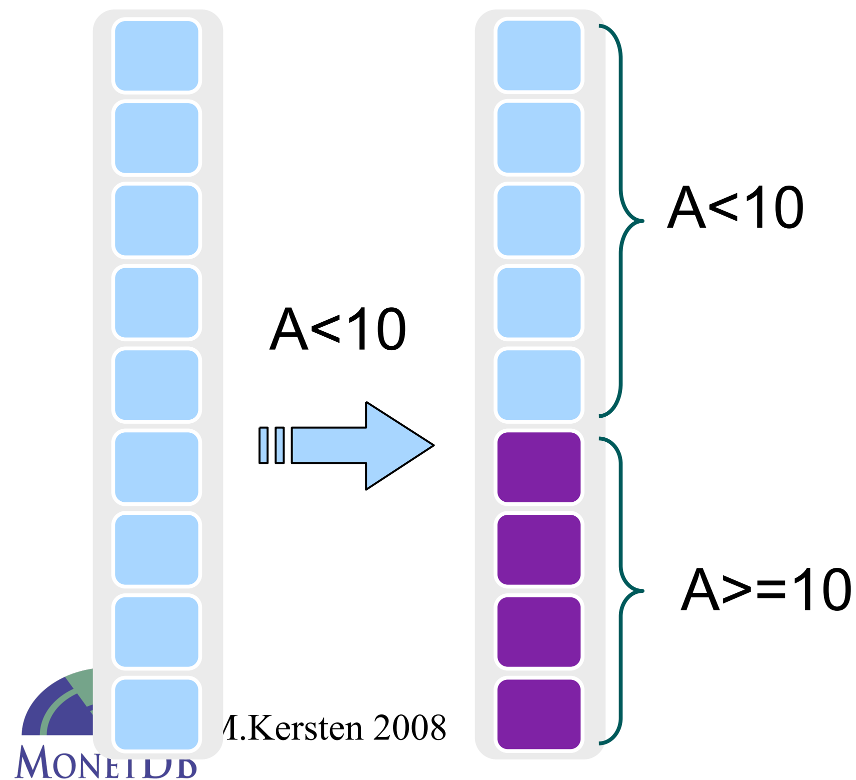
Split a piece of a column in two new pieces



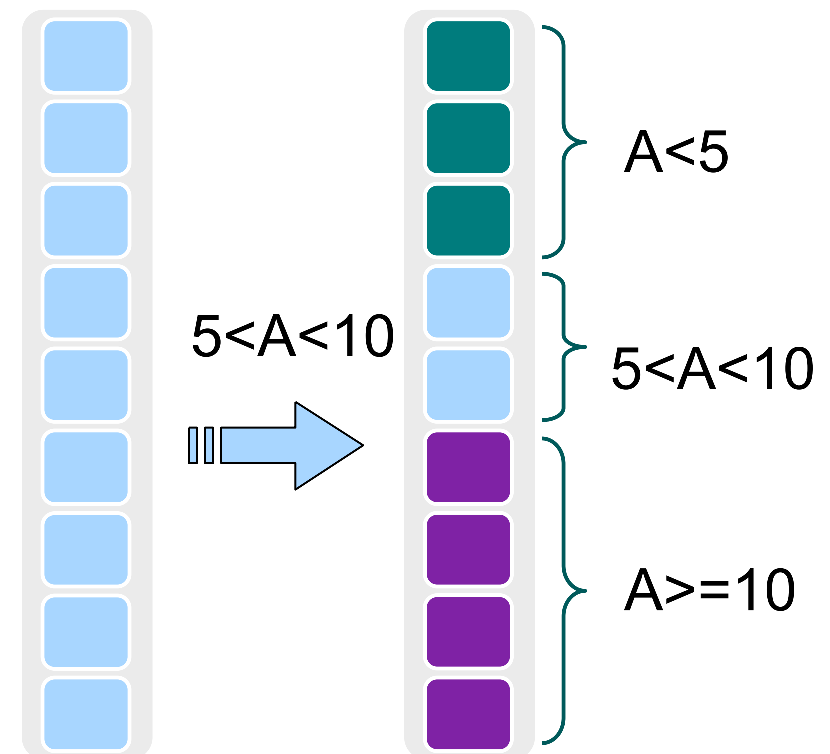
Cracking algorithms

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



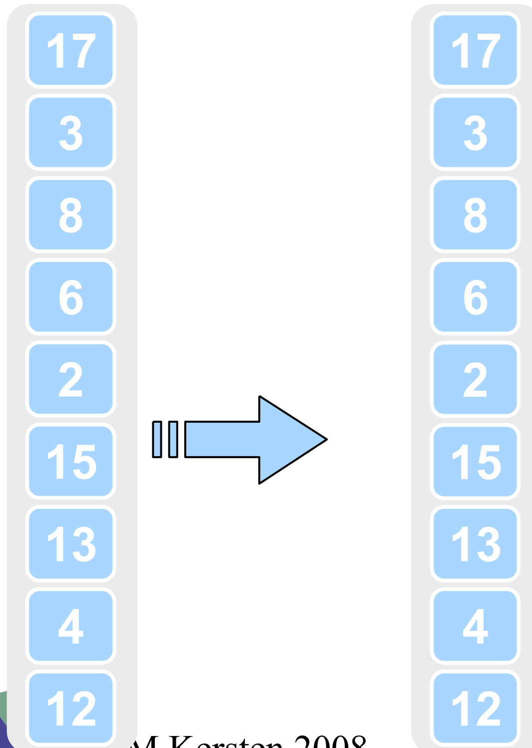
Cracking example

select A>5 and A<10

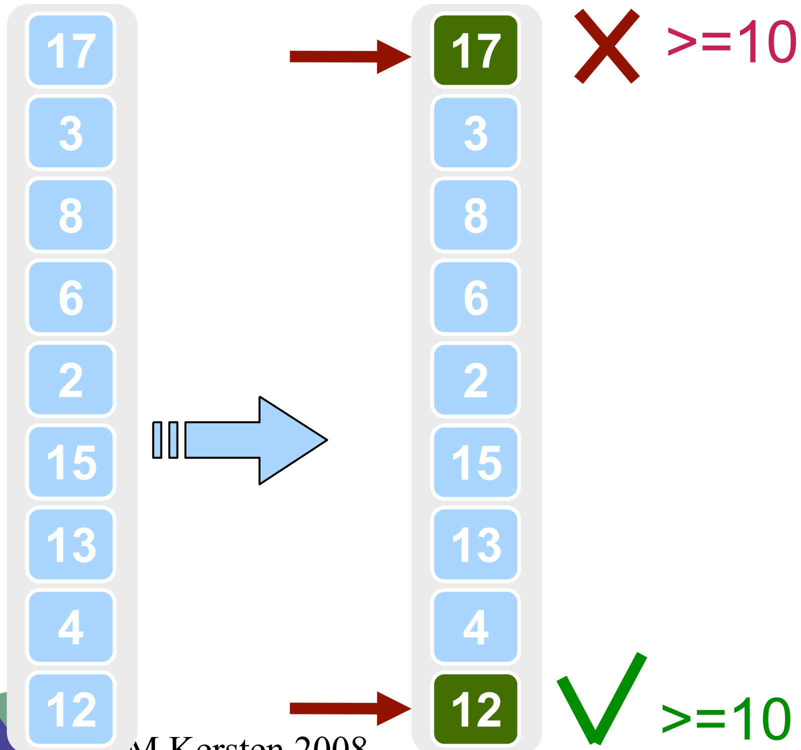
17
3
8
6
2
12
13
4
15

Cracking example

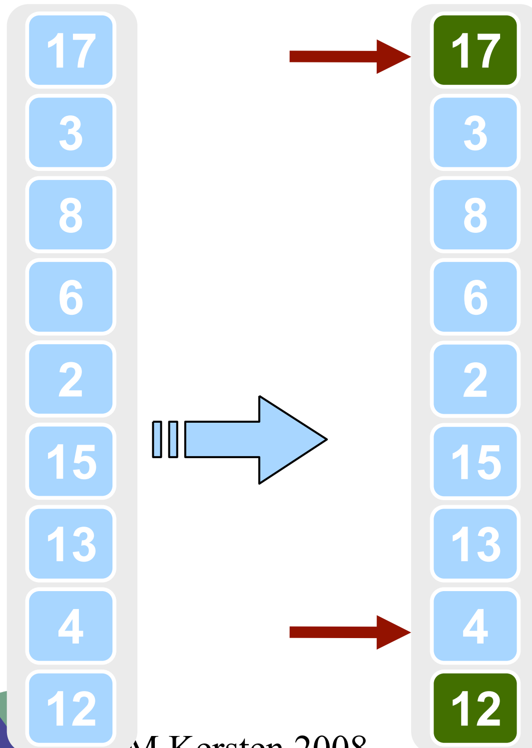
select A>5 and A<10



Cracking example



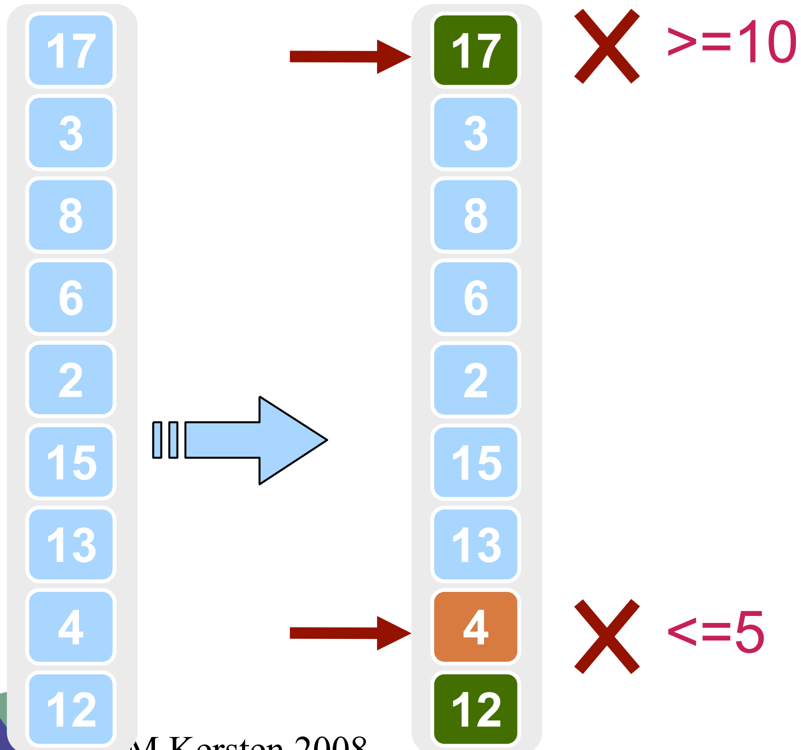
Cracking example



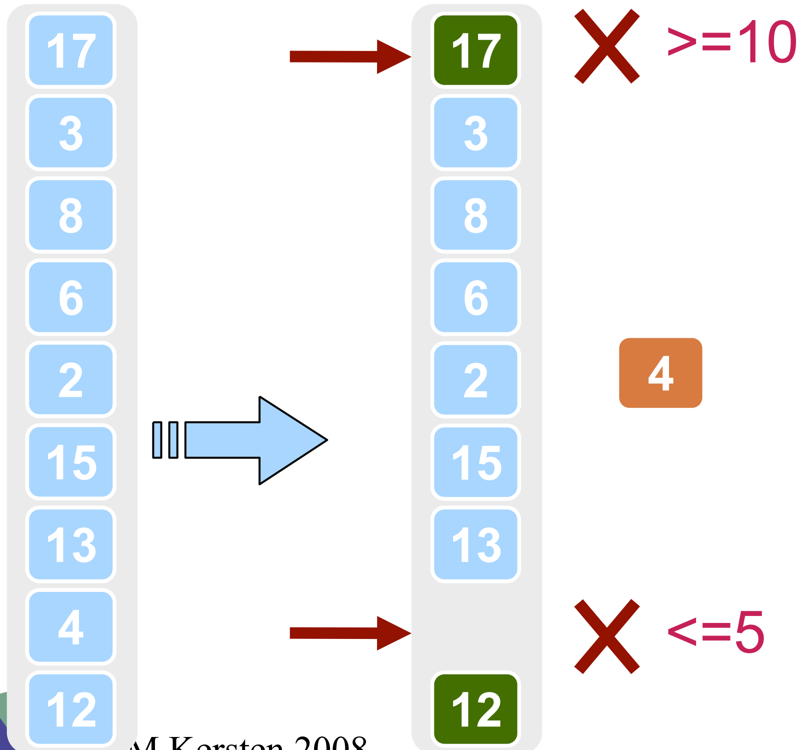
17 3 8 6 2 15 13 4 12

>=10

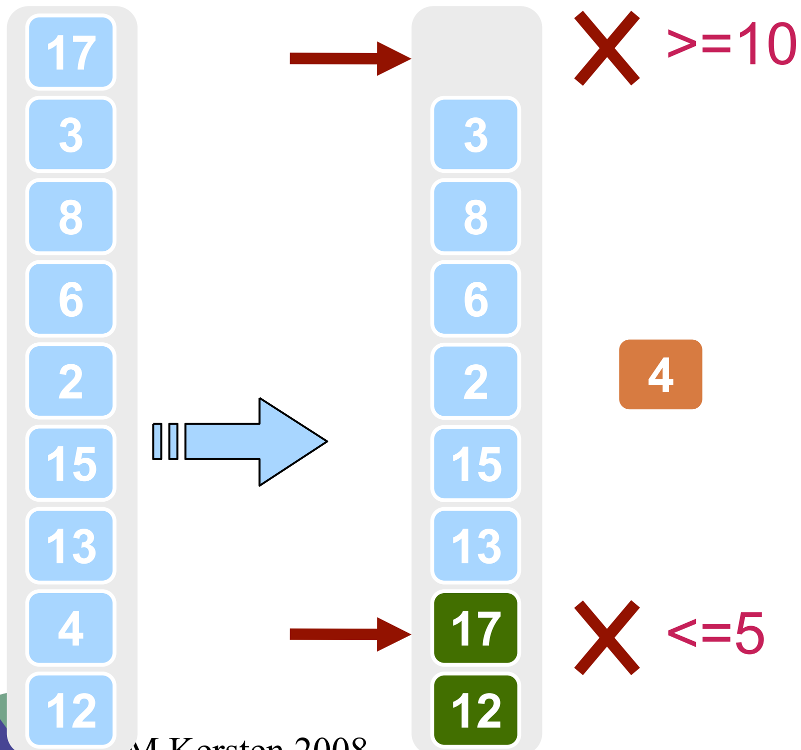
Cracking example



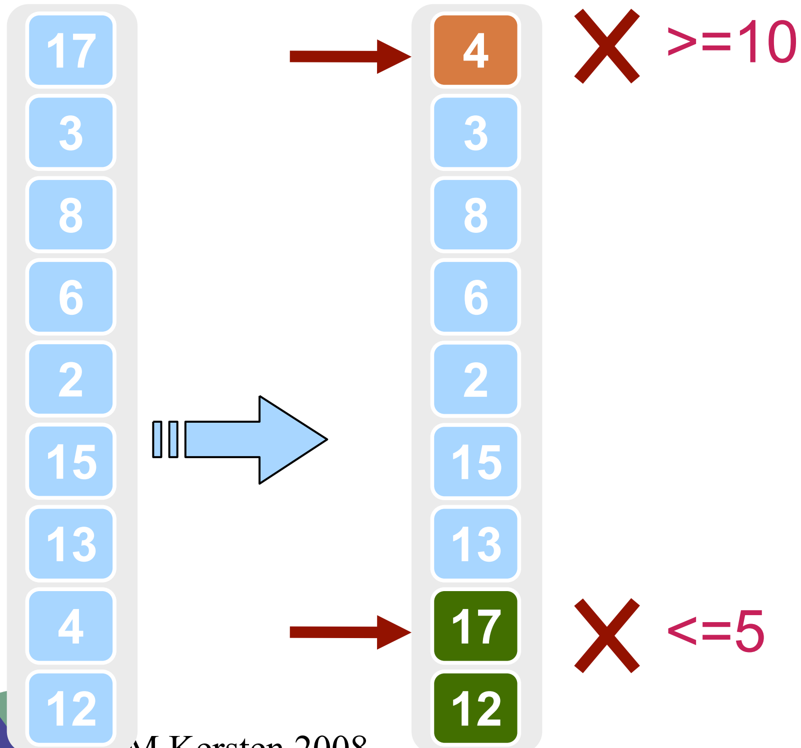
Cracking example



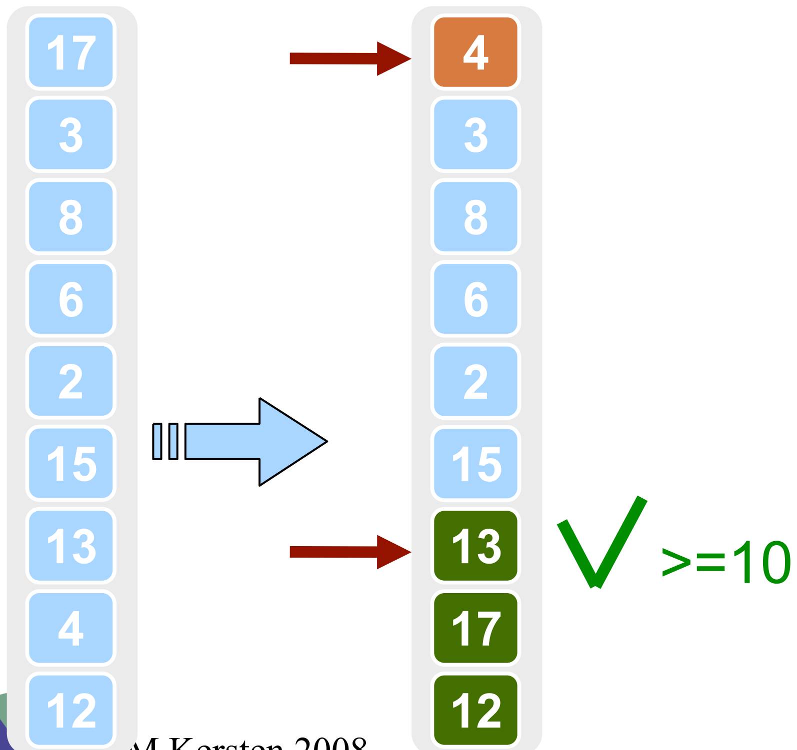
Cracking example



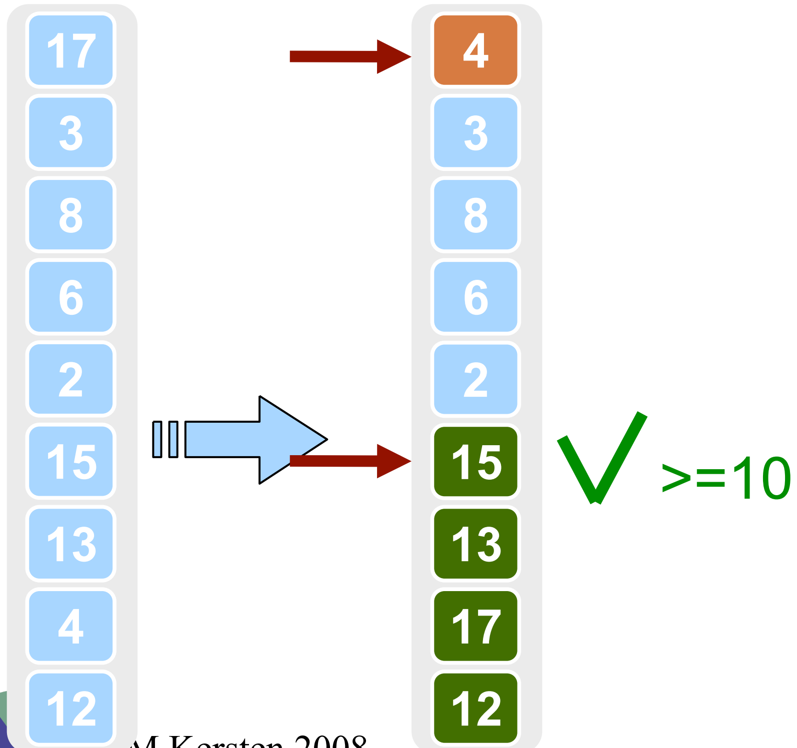
Cracking example



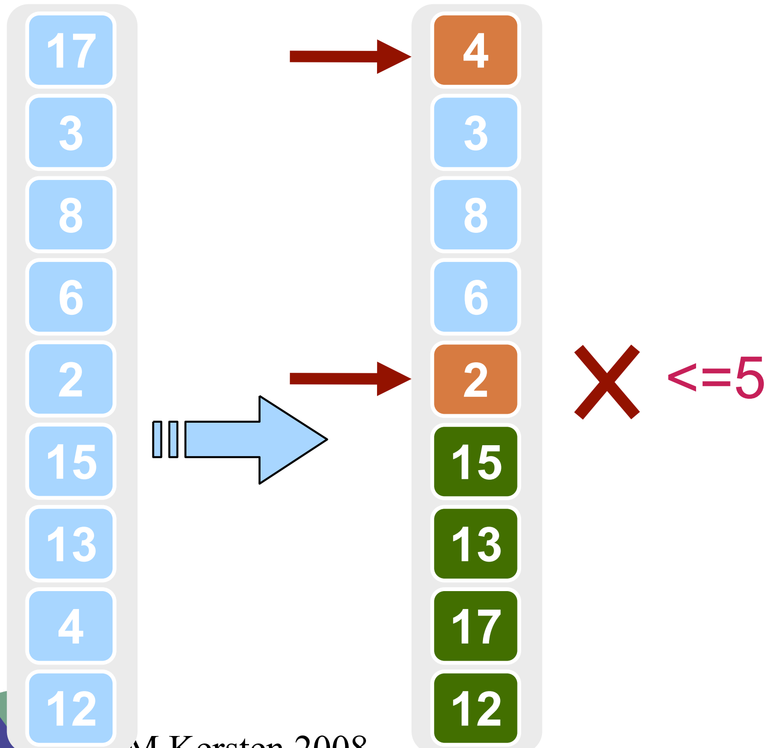
Cracking example



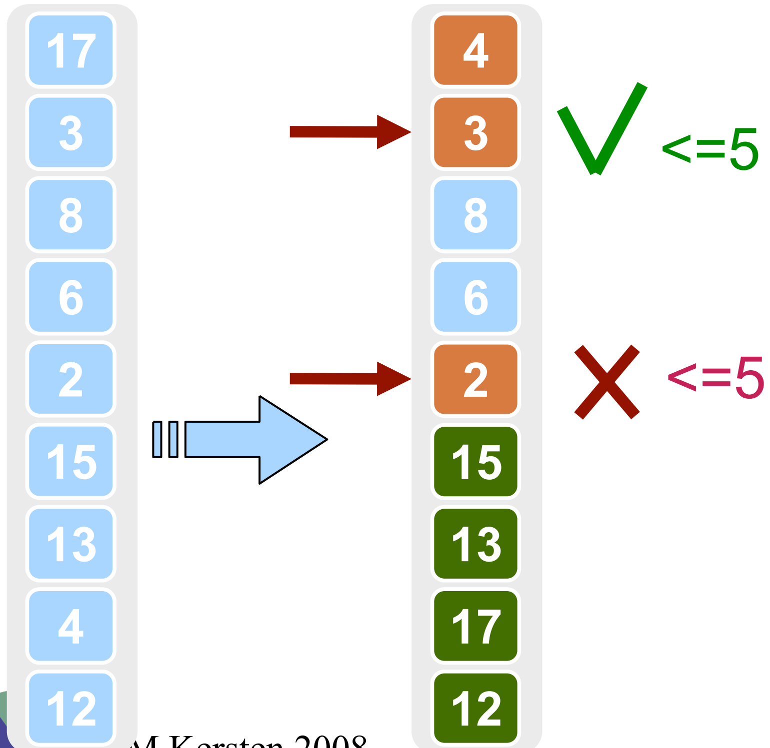
Cracking example



Cracking example

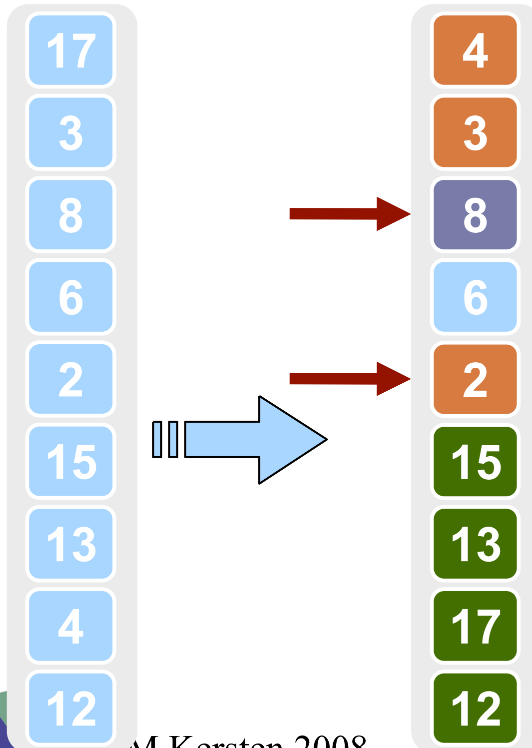


Cracking example



Cracking example

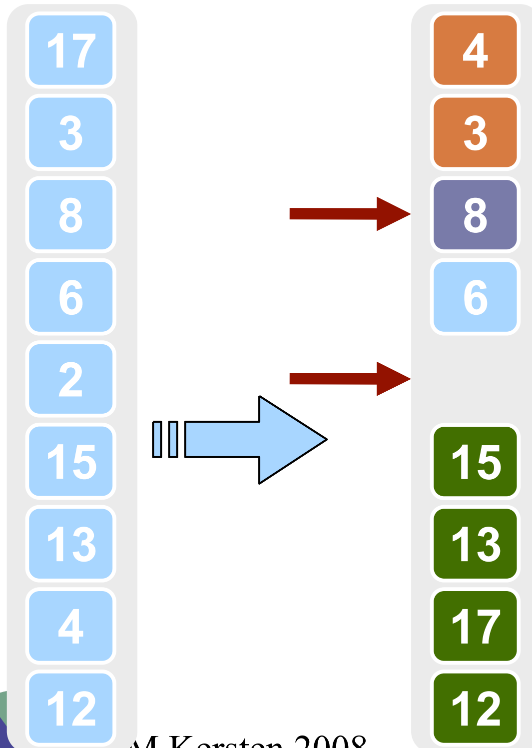
select A>5 and A<10



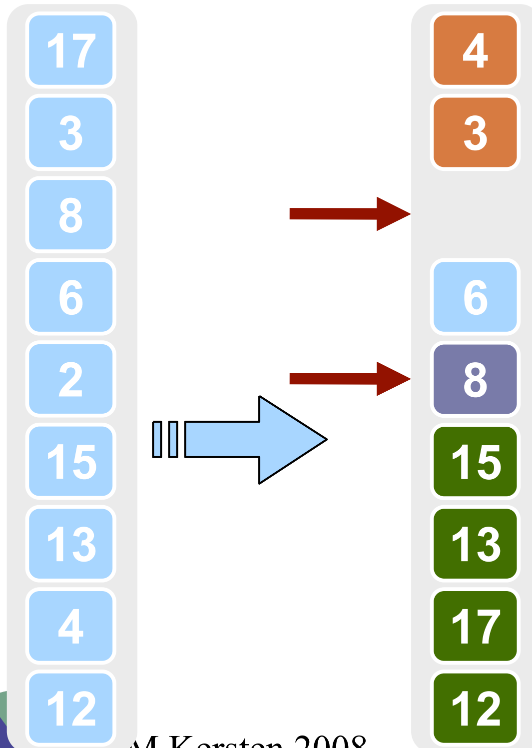
>5 and <10

<=5

Cracking example

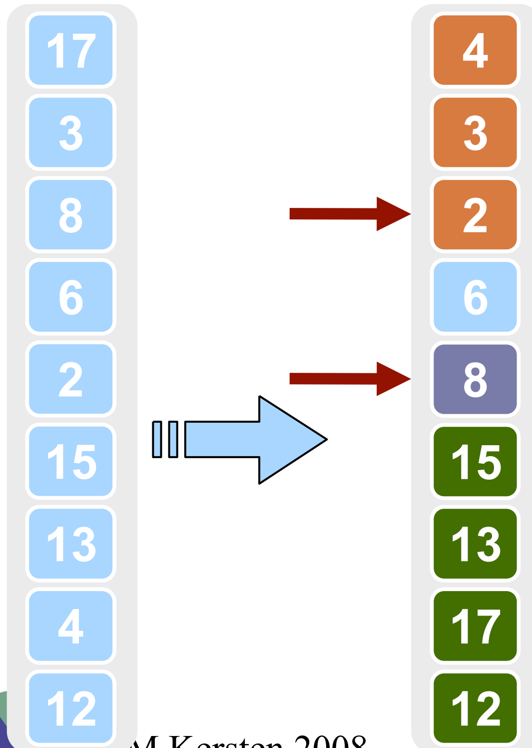


Cracking example



Cracking example

select A>5 and A<10

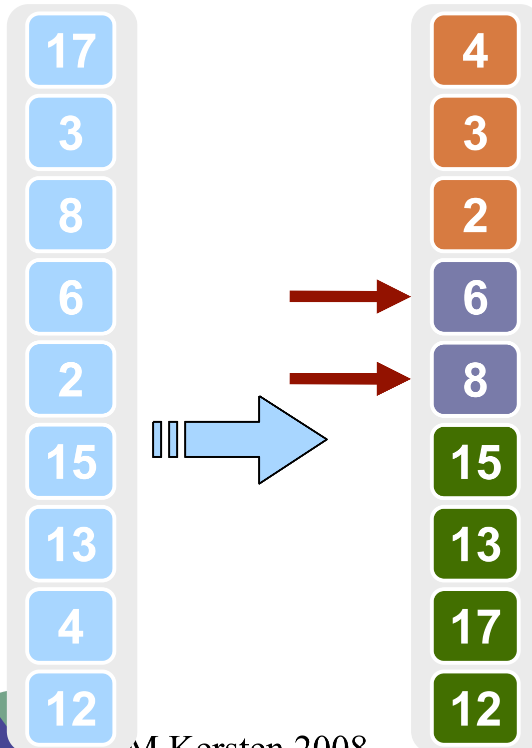


>5 and <10

<=5

Cracking example

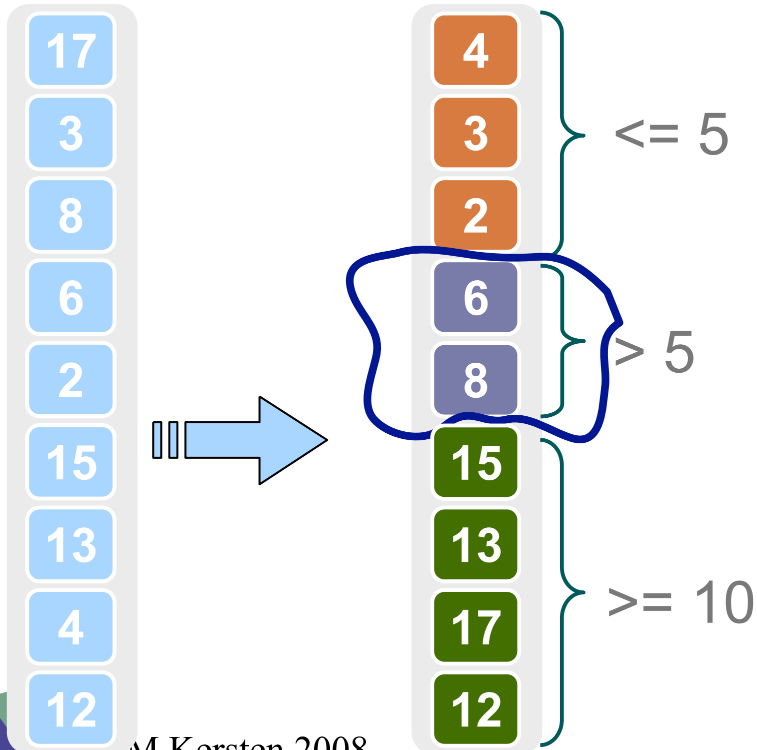
select A>5 and A<10



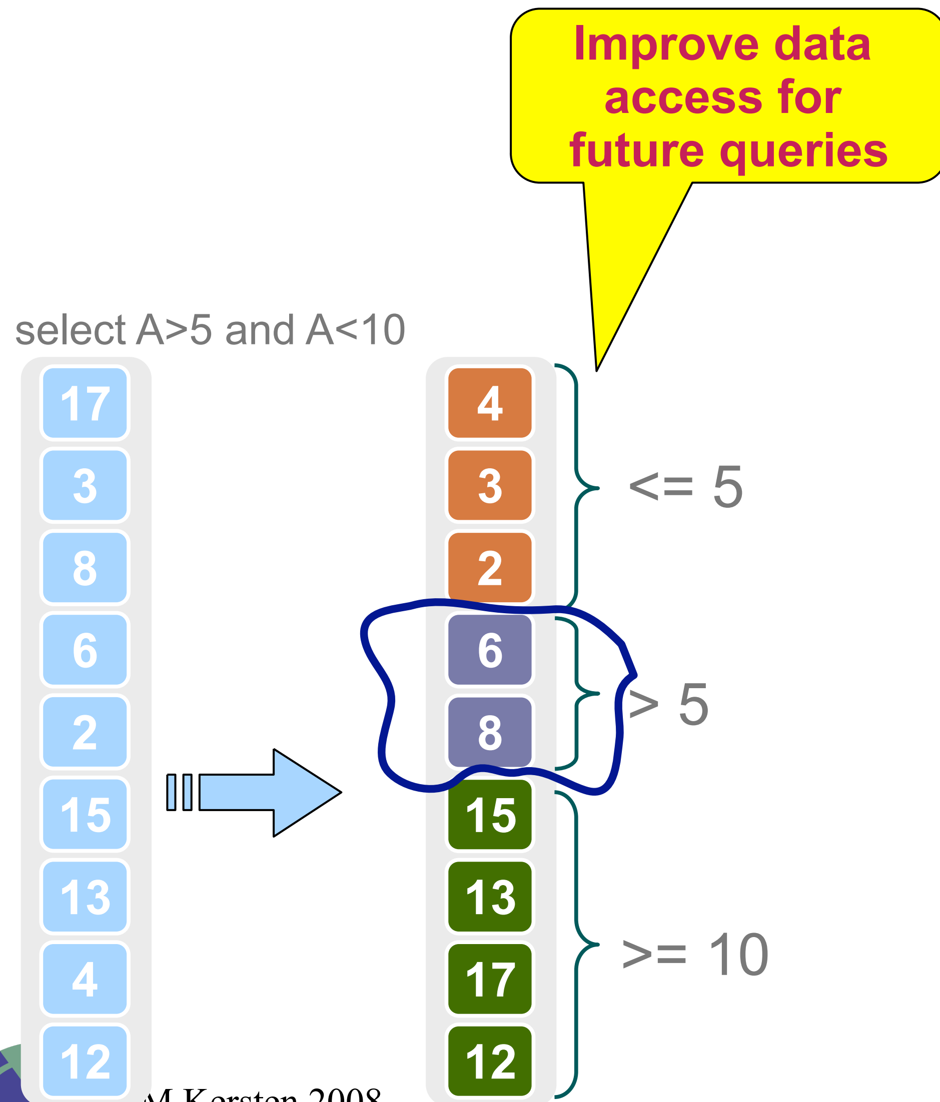
>5 and <10

Cracking example

select A>5 and A<10



Cracking example

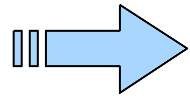


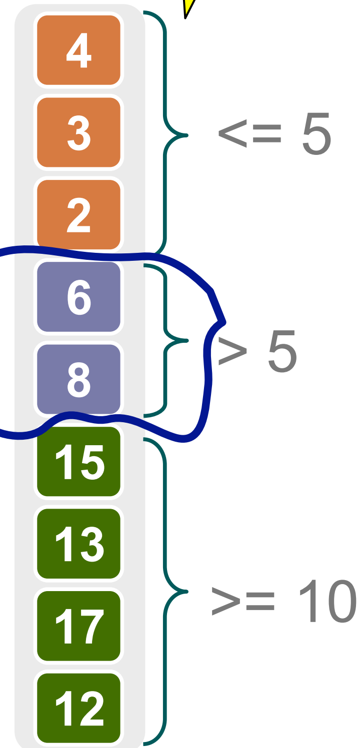
Cracking example

Improve data access for future queries

select A>5 and A<10

select A>3 and A<14



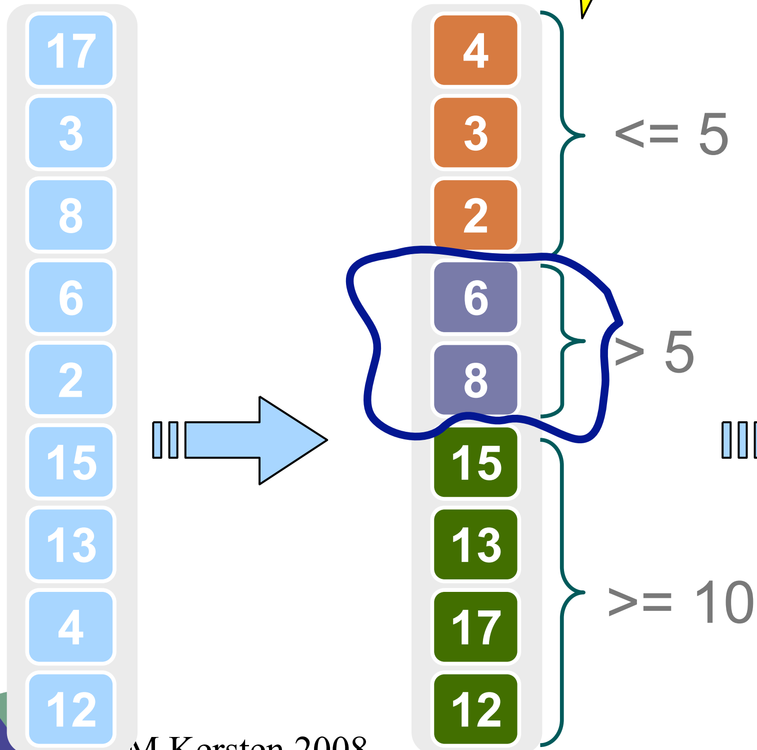


Cracking example

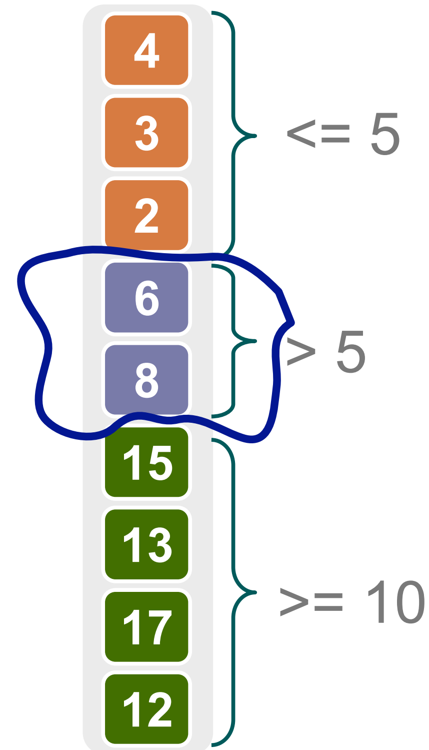
Cracking example

Improve data access for future queries

select A>5 and A<10

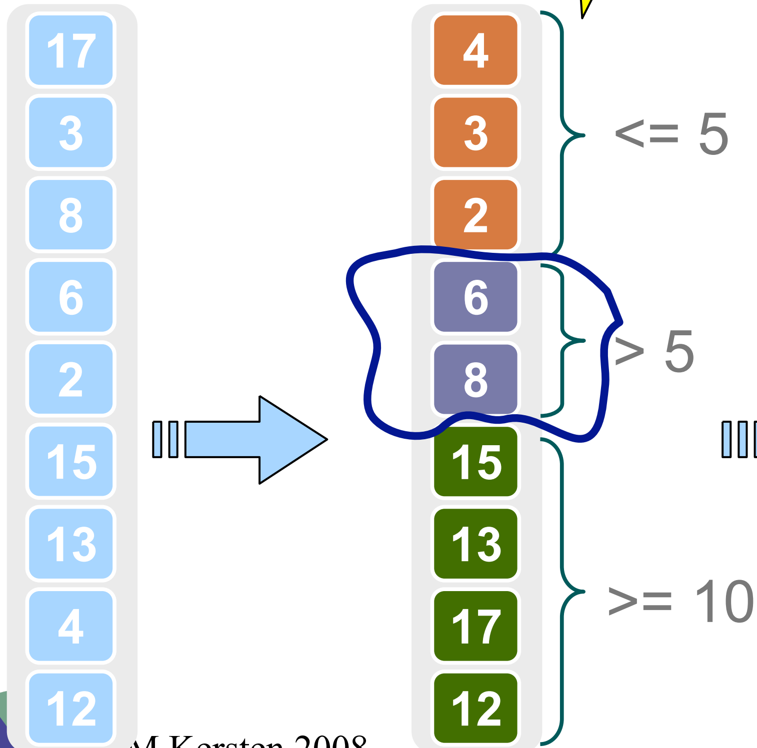


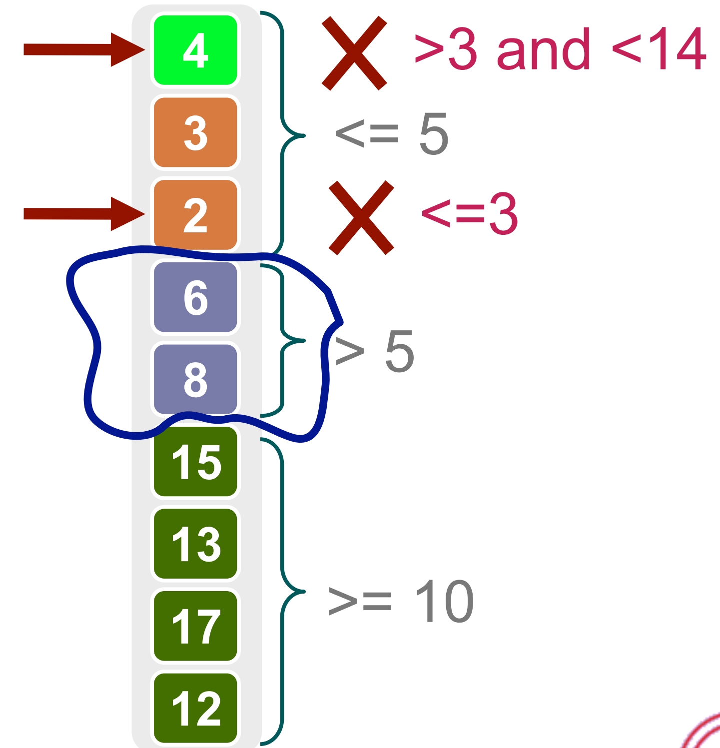
select A>3 and A<14



Cracking example

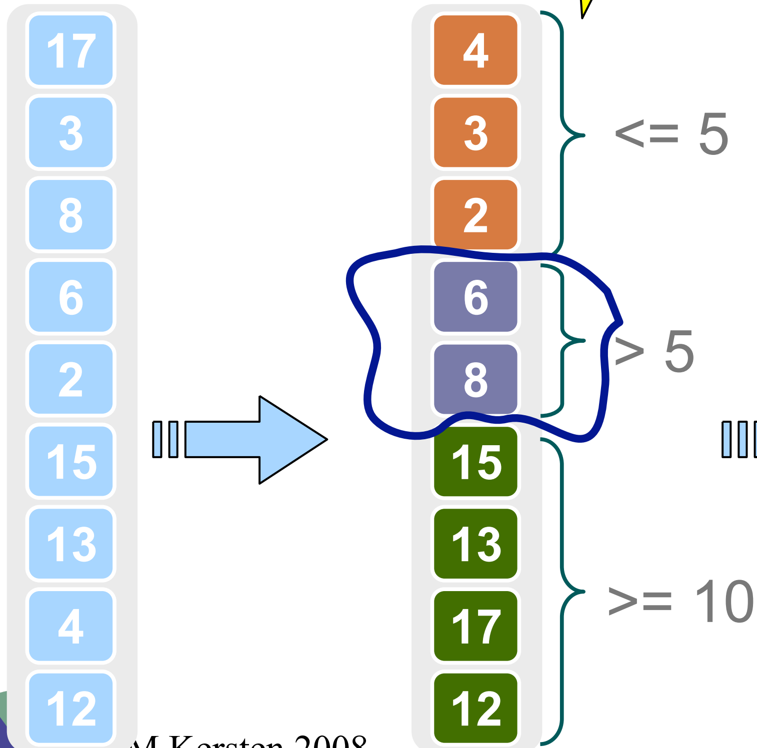
racking example

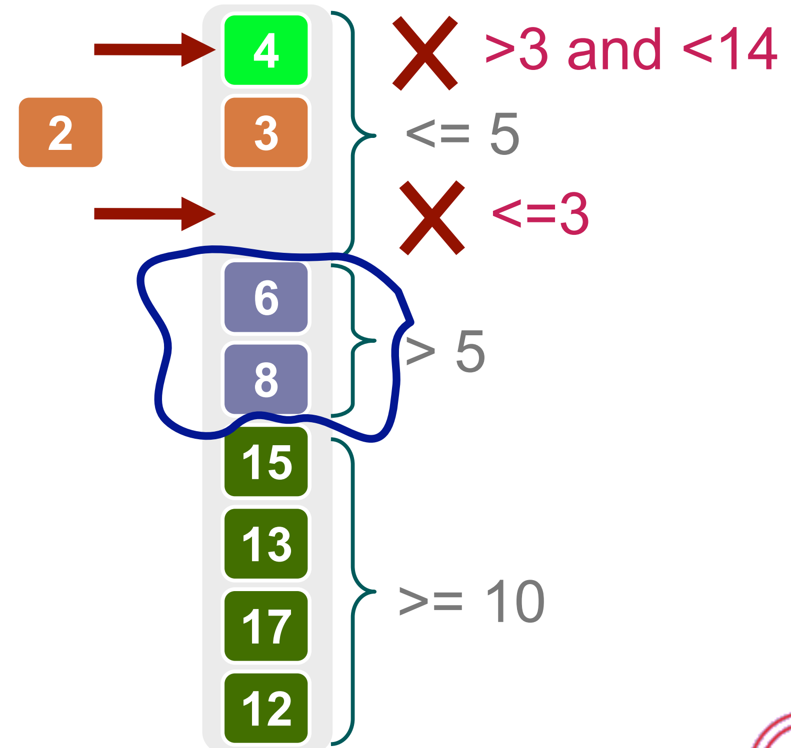




Cracking example

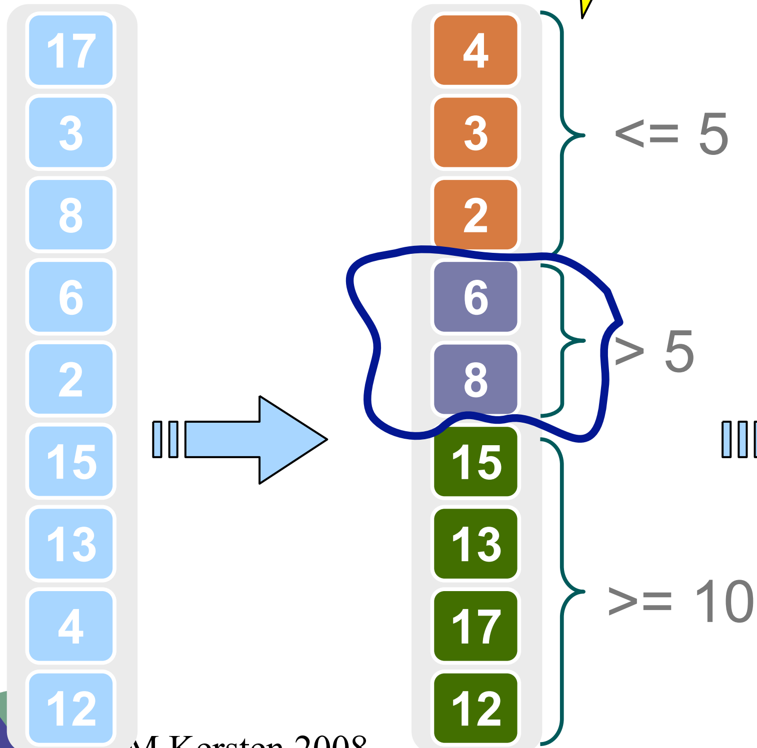
Improve data access for future queries

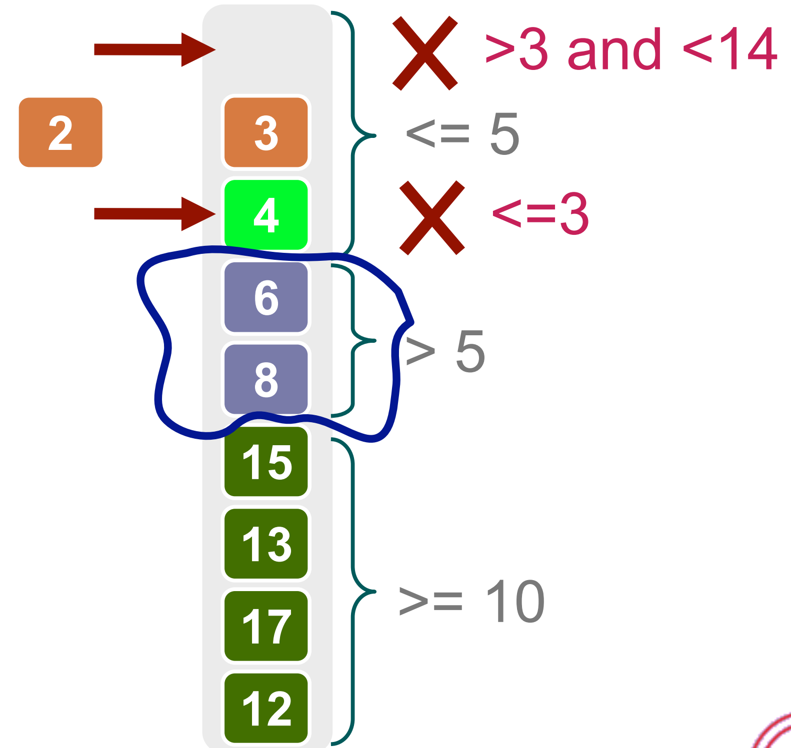




Cracking example

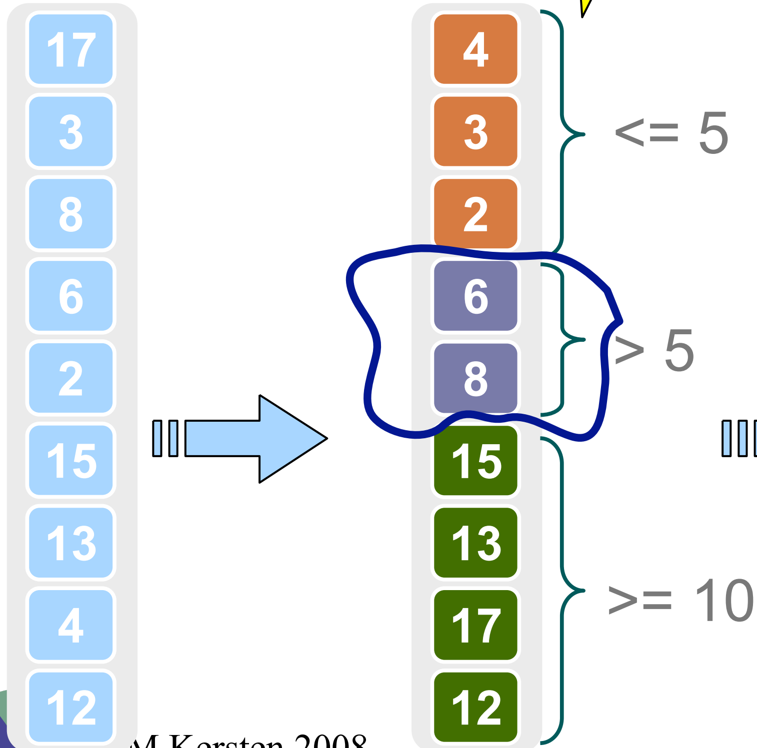
Improve data access for future queries

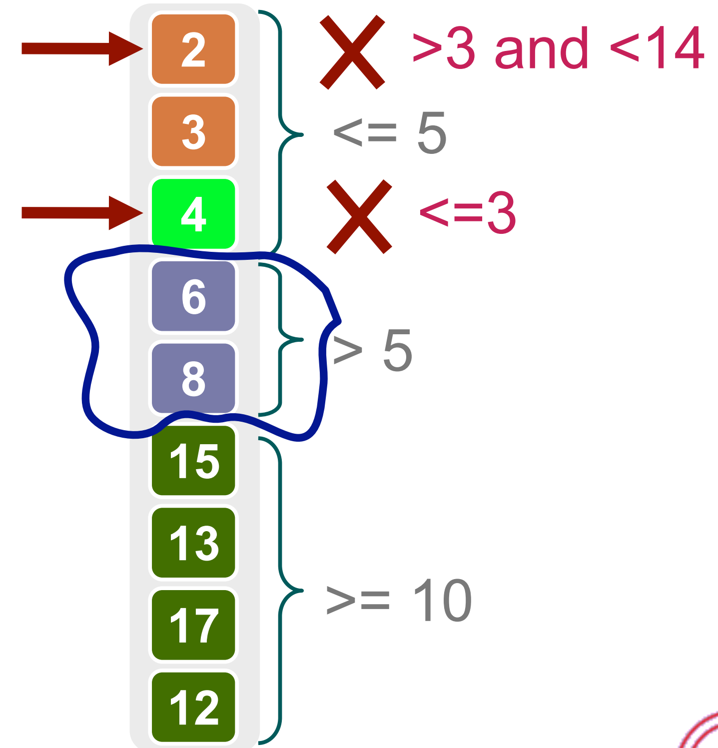




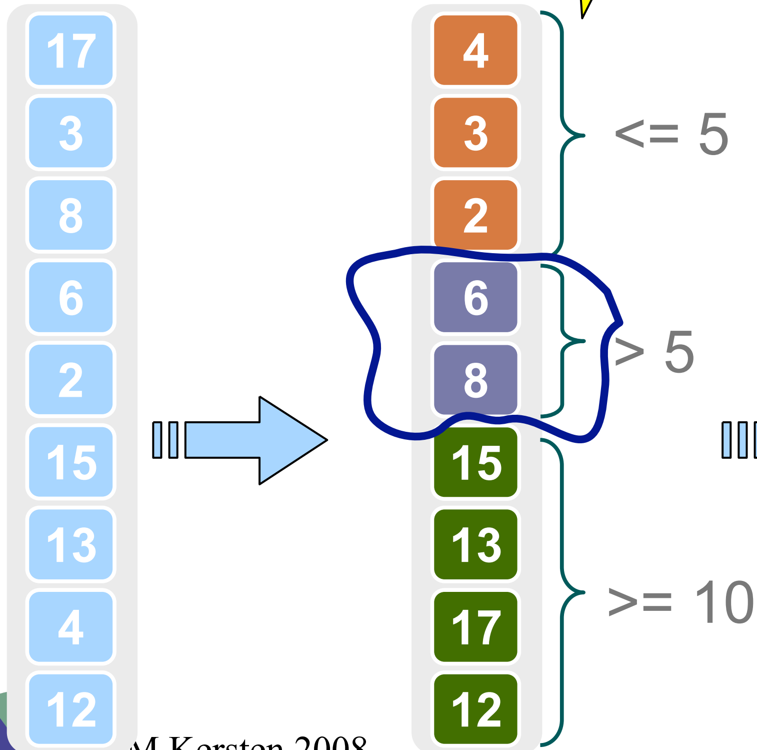
Cracking example

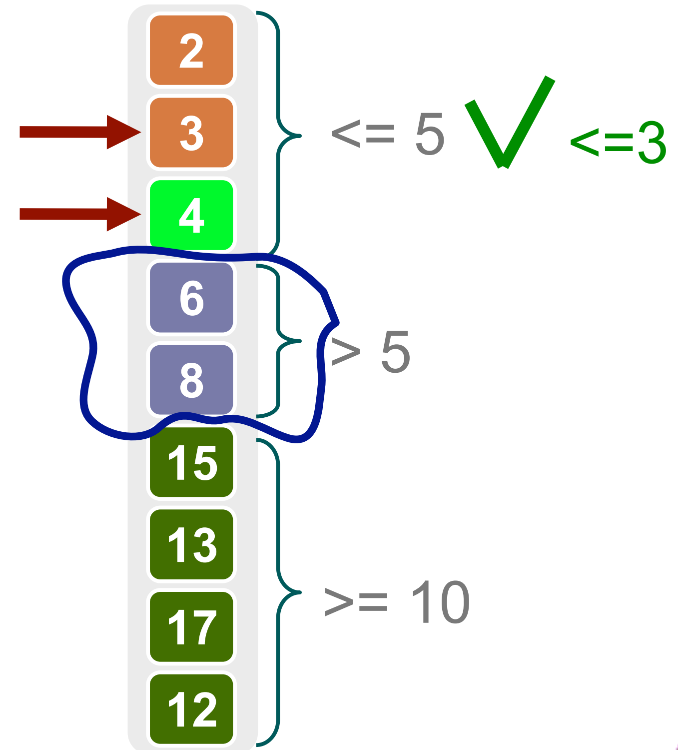
Improve data access for future queries



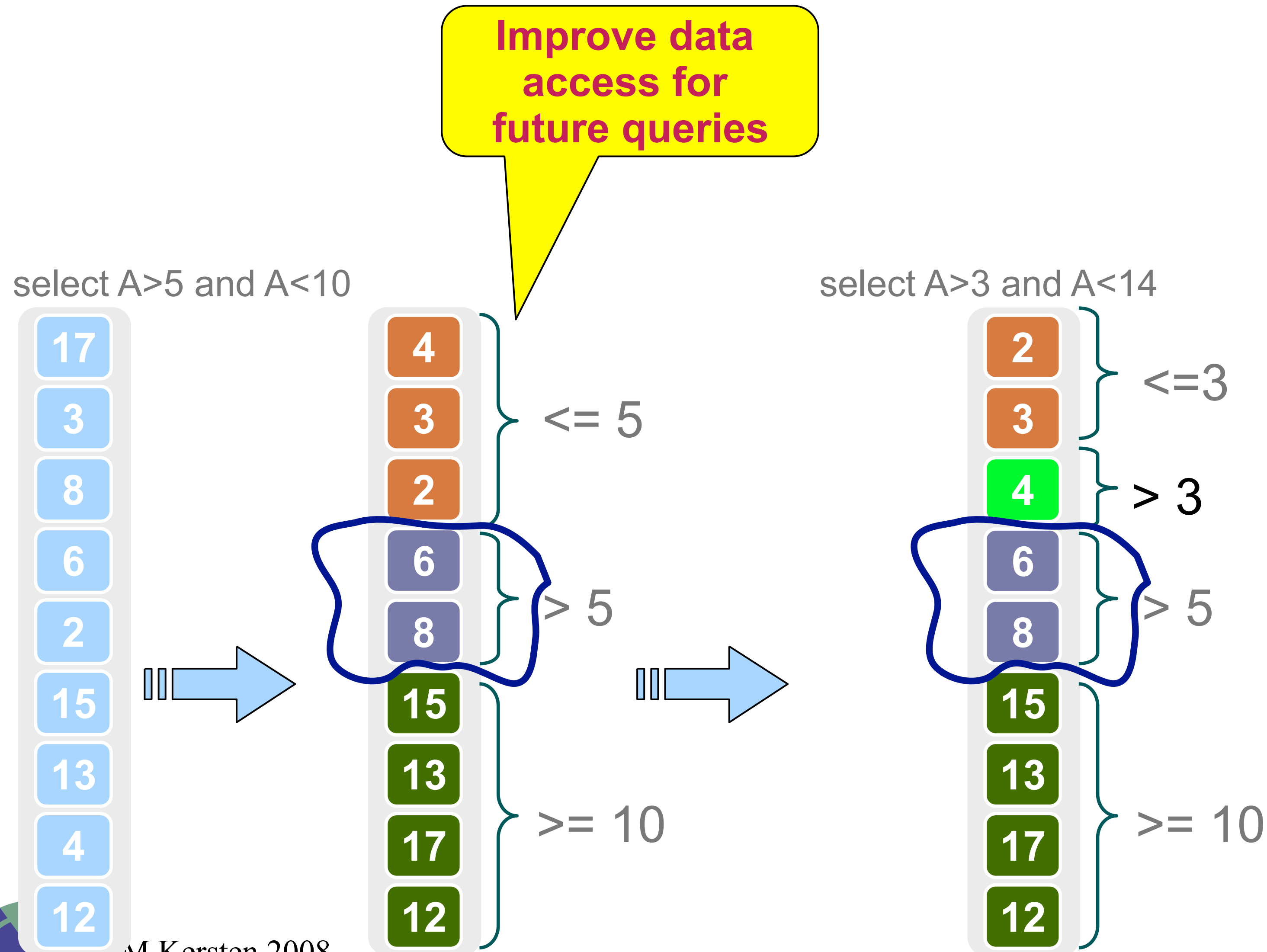


Cracking example





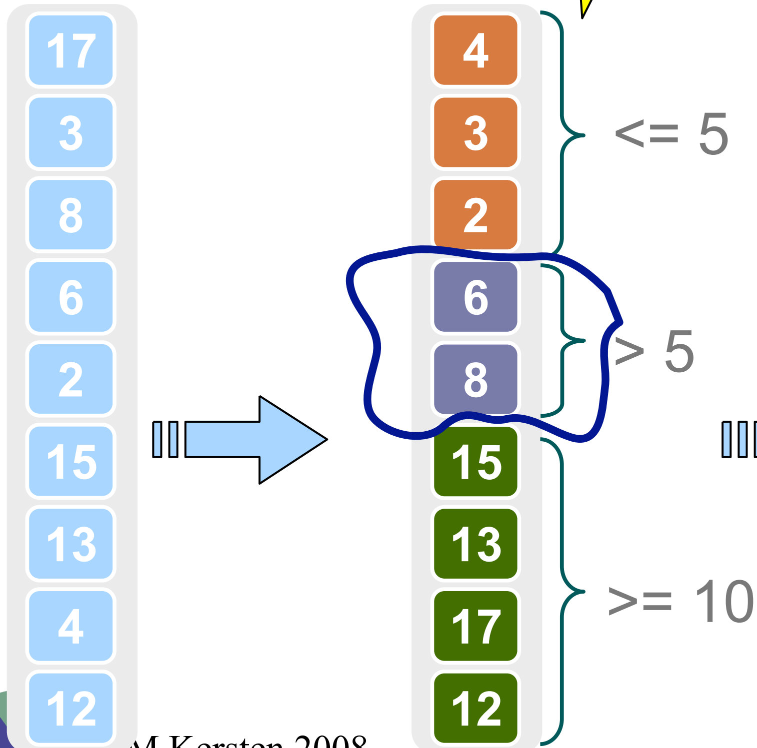
Cracking example

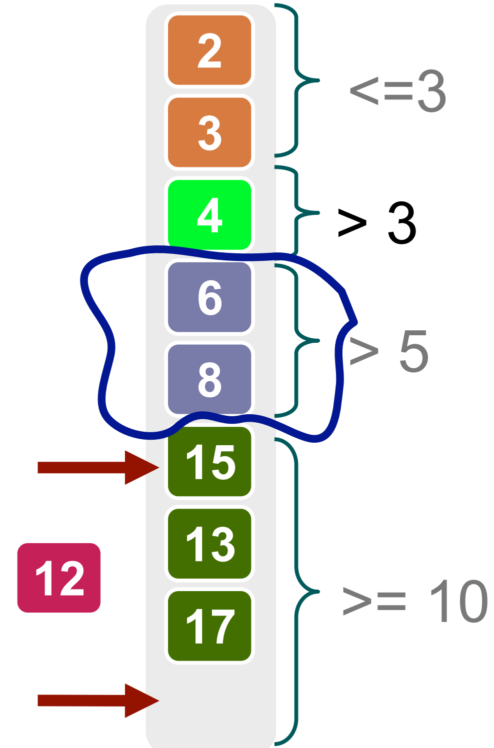


Cracking example

Cracking example

Cracking example



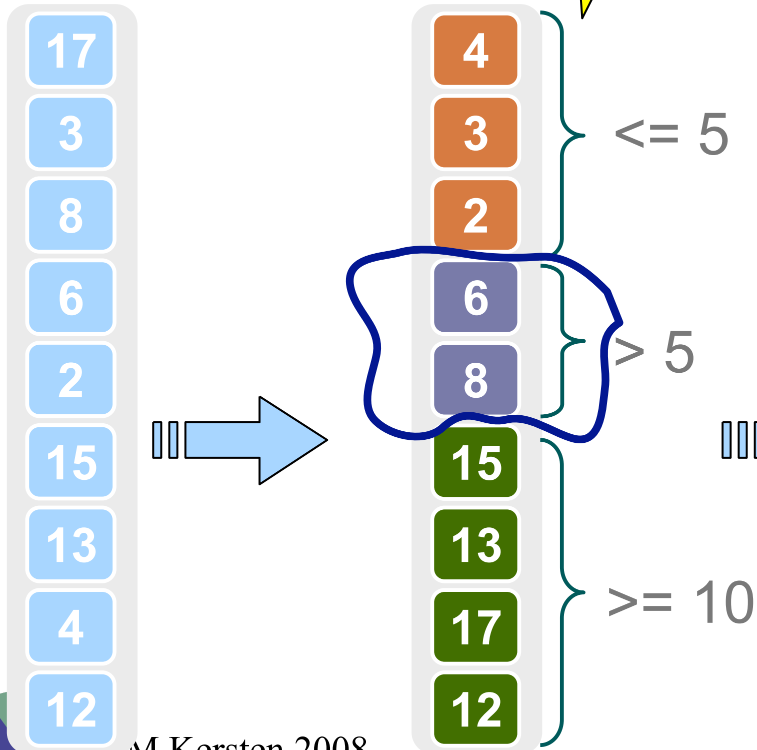


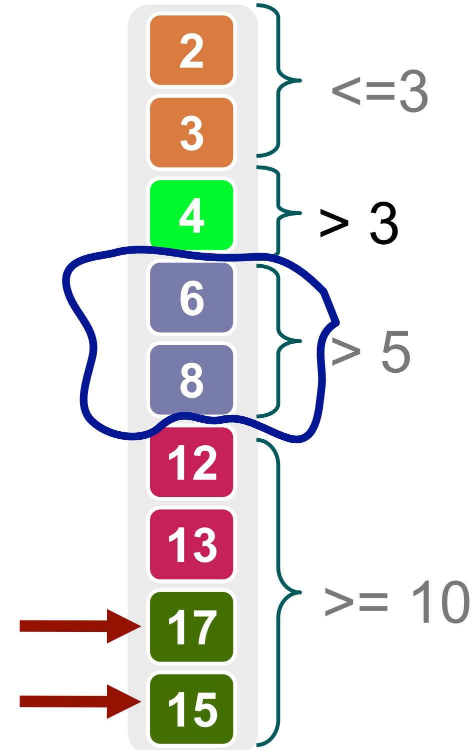
Cracking example

Cracking example

Cracking example

Cracking example

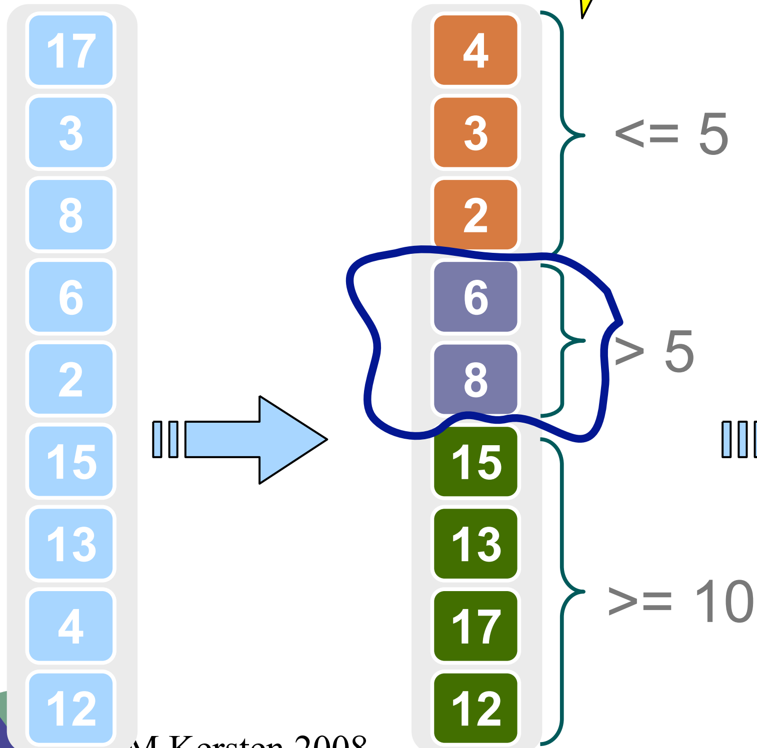




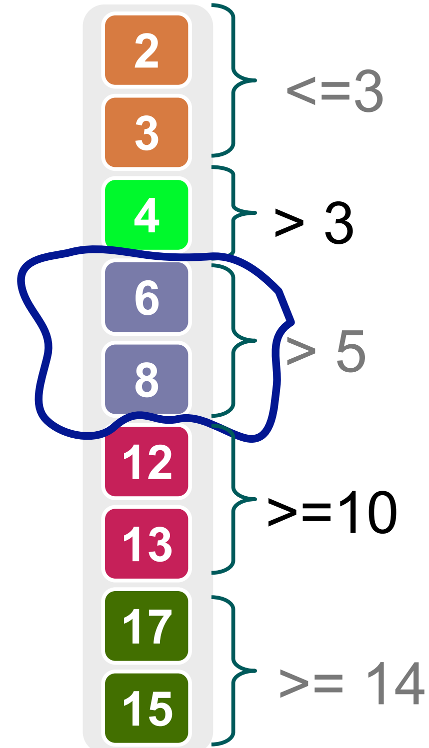
Cracking example

Improve data access for future queries

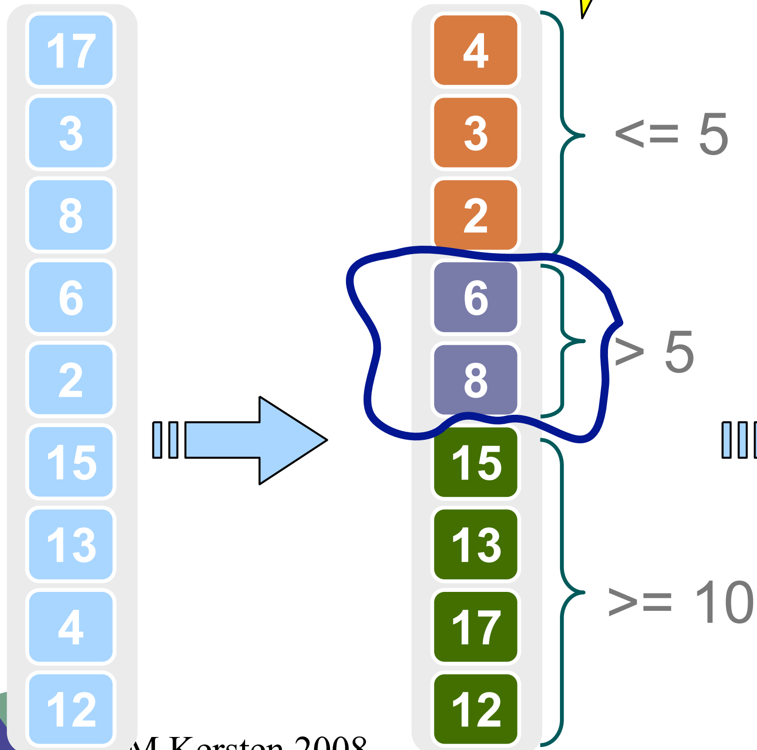
select A>5 and A<10

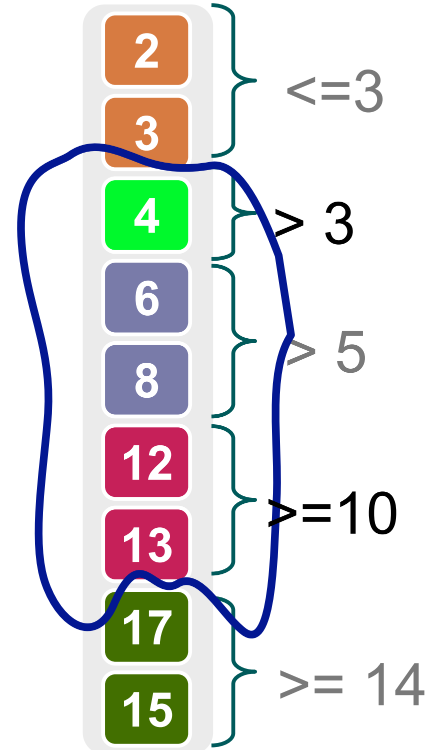


select A>3 and A<14

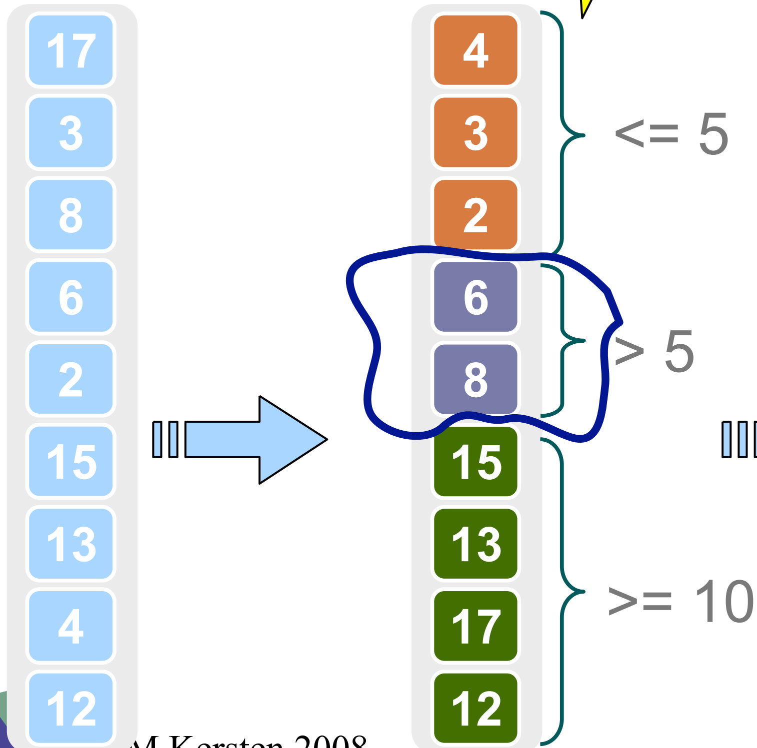


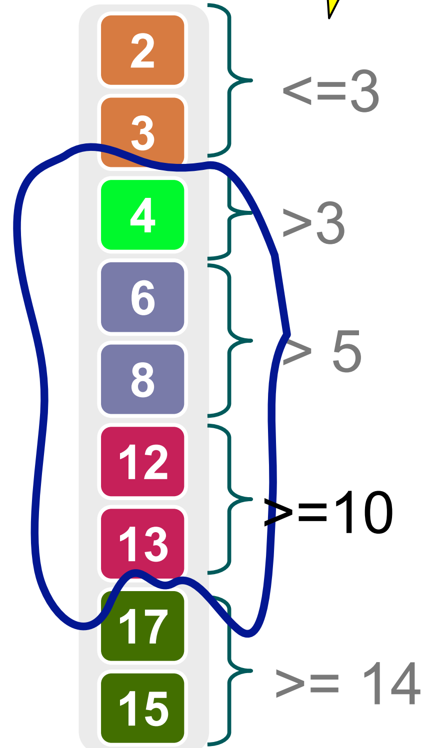
Cracking example





Cracking example

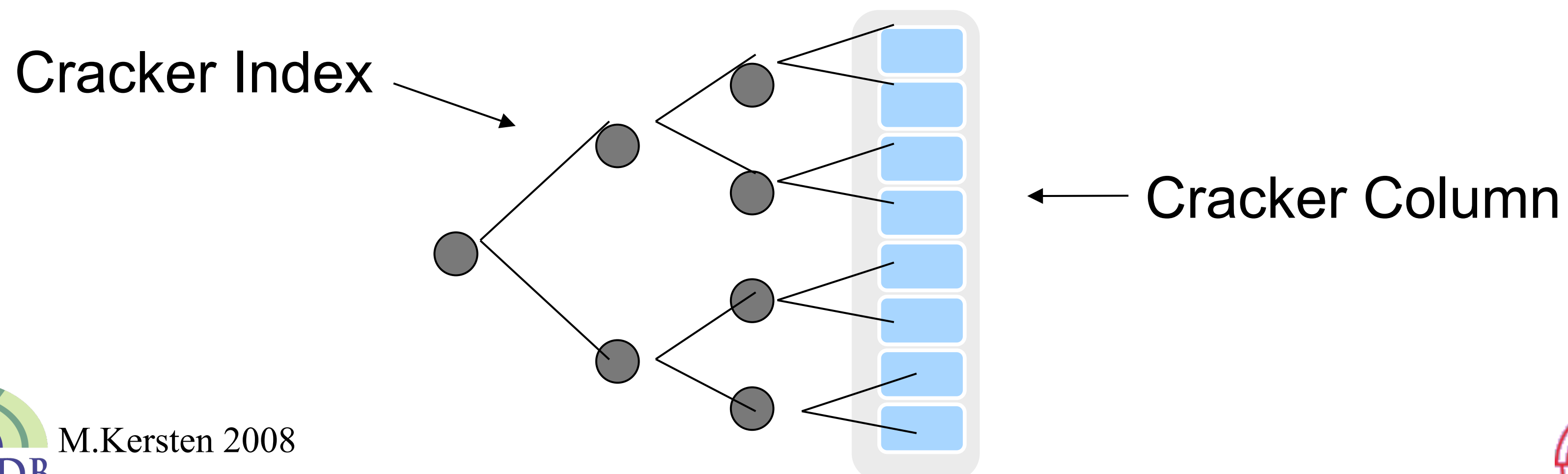




The first time a range query is posed on an attribute *A*, a cracking DBMS makes a copy of column *A*, called the *cracker column* of *A*

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*



Try to avoid useless investments

A simple range query

Response time (milli secs)

7000
1000
100
10
1
0.2

0 200 400 600 800 1000

Query sequence

PostgreSQL
PostgreSQL with B-Tree
MySQL
MySQL with B-Tree
MonetDB/SQL
MonetDB/SQL with cracking

Try to avoid useless investments

TPC-H query 6

Response time (milli secs)

10000
1000
100
10

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

0 5 10 15 20 25

Query sequence

Try to avoid useless investments

- Cracking is easy in a column store and is part of the critical execution path
- Cracking works under high volume updates

Updates

- Base columns are updated as normally
- We need to update the cracker column and the cracker index
- Efficiently
- Maintain the self-organization properties
- Two issues:
  - When
  - How

When to propagate updates in cracking

- Follow the workload to maintain self-organization
- 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

Updates aware select

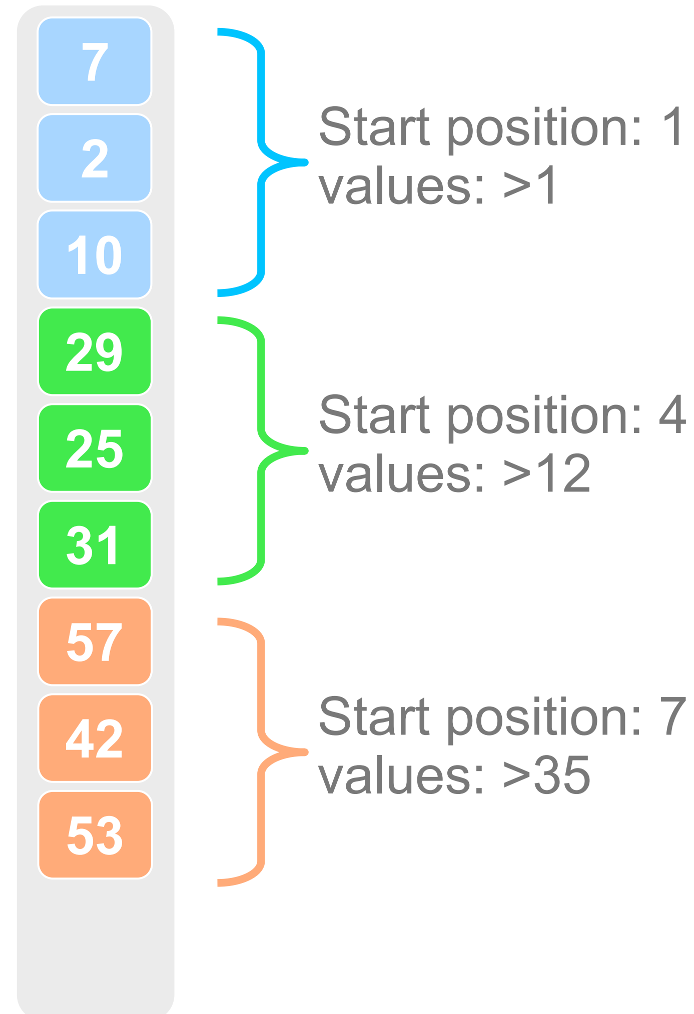
- We extended the cracker select operator to apply the needed updates before cracking

- The select operator:
  1. Search the pending insertions column
  2. Search the pending deletions column
  3. If Steps 1 or 2 find tuples run an update algorithm
  4. Search the cracker index
  5. Physically reorganize the cracker column
  6. Update the cracker index
  7. Return a slice of the cracker column

Merging

Insert a new tuple with value 9

The new tuple belongs to the blue piece



Merging

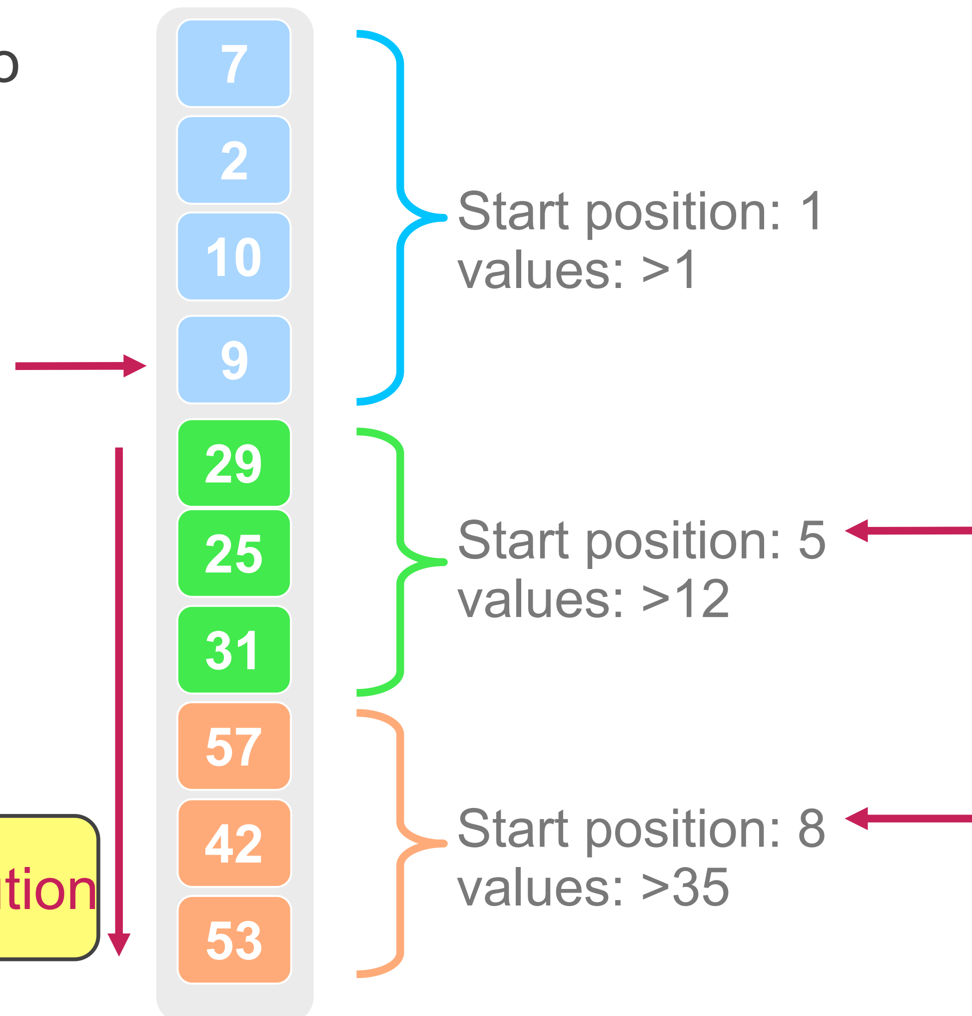
Insert a new tuple with value 9

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



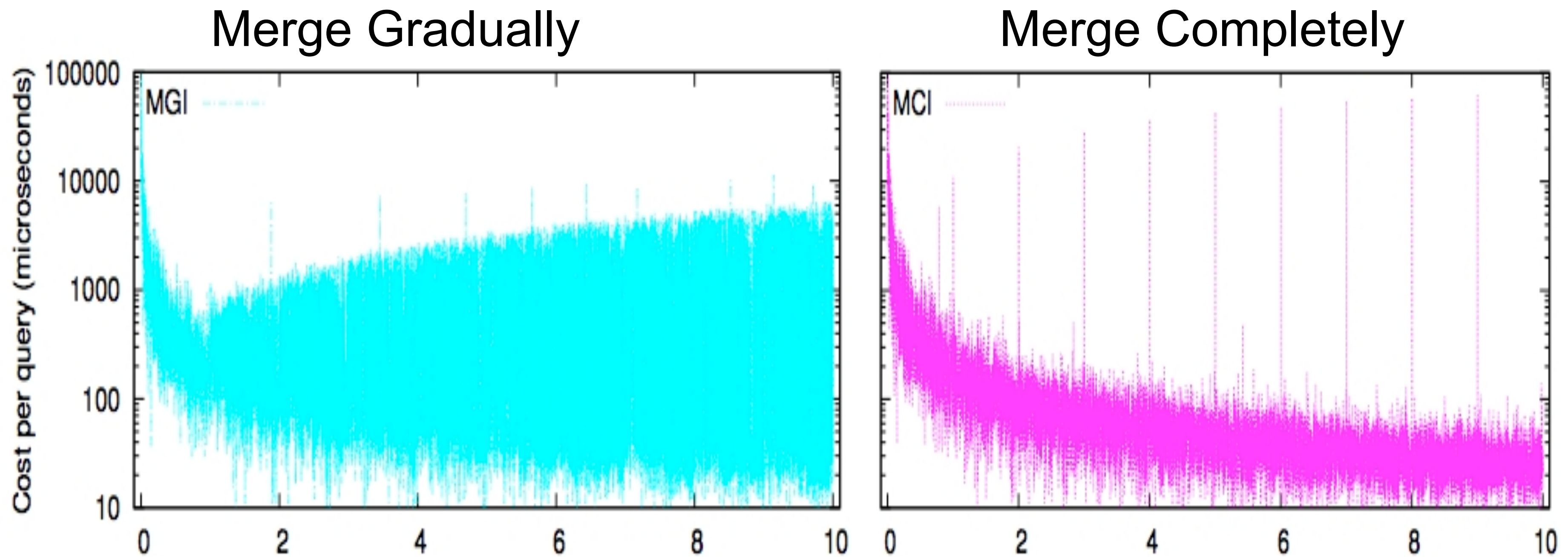
Merging by Hopping

Insert a new tuple with value 9

We need to make enough room to fit the new tuples

Merge Gradually

- A query merges only the qualifying values, i.e., only the values that it needs for a correct and complete result



We avoid the large peaks but...

Average cost increases significantly

The Ripple

Touch only the pieces that are relevant for the current query

The Ripple

Touch only the pieces that are relevant for the current query

Start position: 1
values: >1

Start position: 4
values: >22

Start position: 7
values: >35

The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

Start position: 1
values: >1

Start position: 4
values: >22

Start position: 7
values: >35

The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

7
2
10
29
25
31
57
42
53

Start position: 1
values: >1

Start position: 4
values: >22

Start position: 7
values: >35

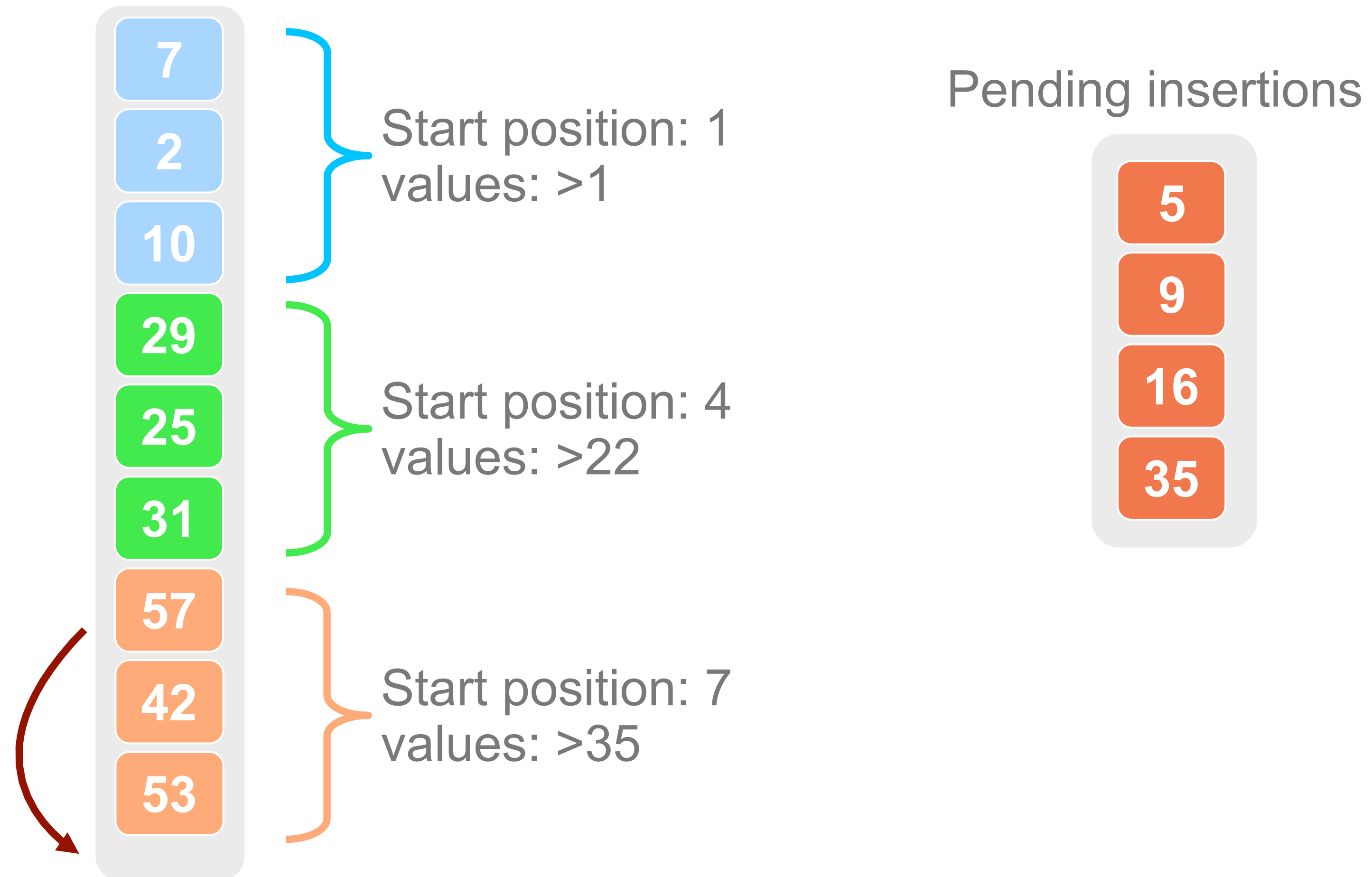
Pending insertions

5
9
16
35

The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15



The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

7, 2, 10	Start position: 1 values: >1
29, 25, 31	Start position: 4 values: >22
57, 42, 53	Start position: 7 values: >35

Pending insertions

5, 9, 16, 35

The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

| 7 |
| 2 |
| 10 |
| 29 |
| 25 |
| 31 |
| 57 |
| 42 |
| 53 |

Start position: 1
values: >1

Start position: 4
values: >22

Start position: 7
values: >35

Pending insertions

| 5 |
| 9 |
| 16 |
| 35 |

Avoid shifting down non interesting pieces

The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

Column	Piece
7	Start position: 1 values: >1
2	
10	
29	Start position: 4 values: >22
25	
31	
57	Start position: 7 values: >35
42	
53	

Pending insertions

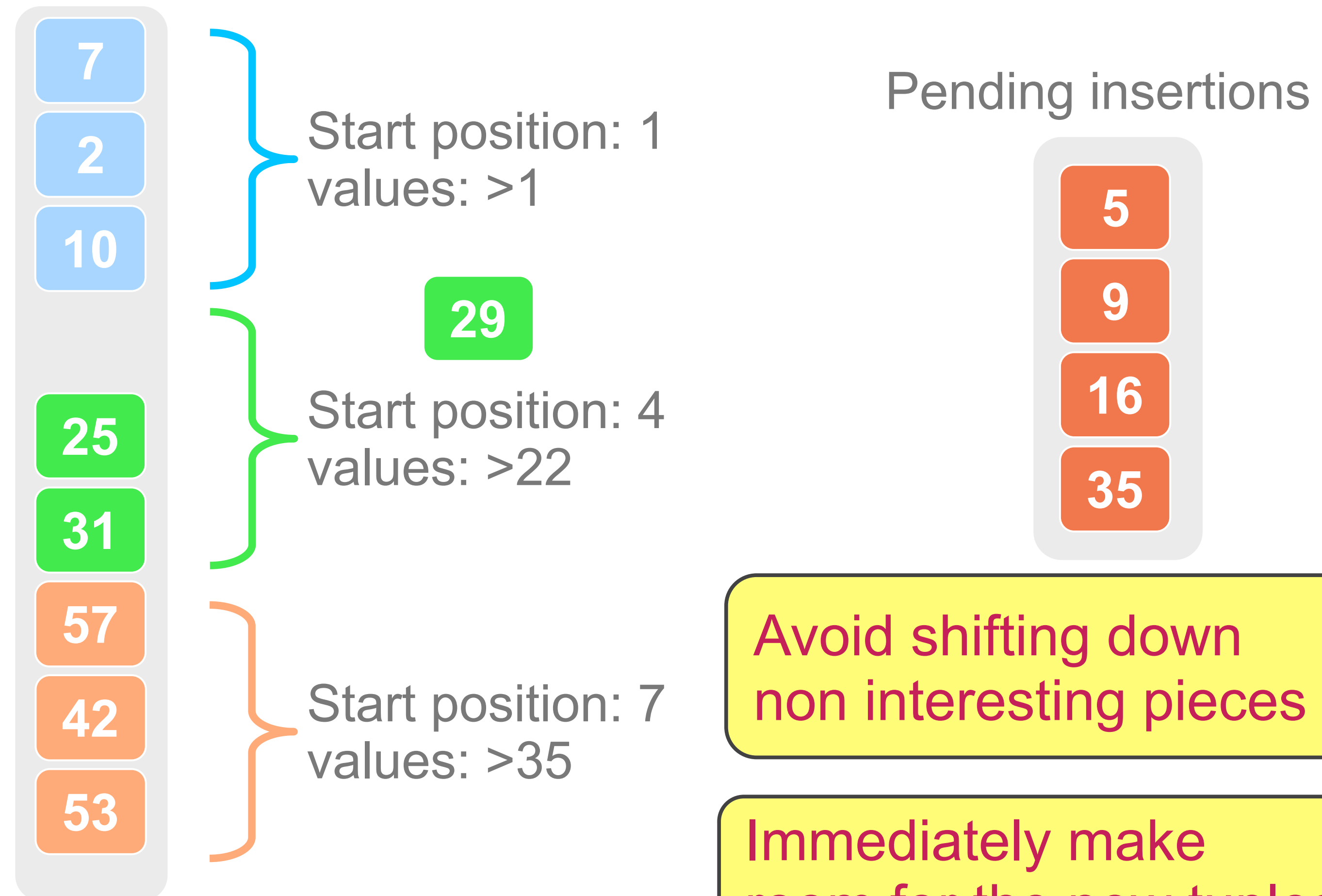
5
9
16
35

Avoid shifting down non interesting pieces

The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15



The Ripple

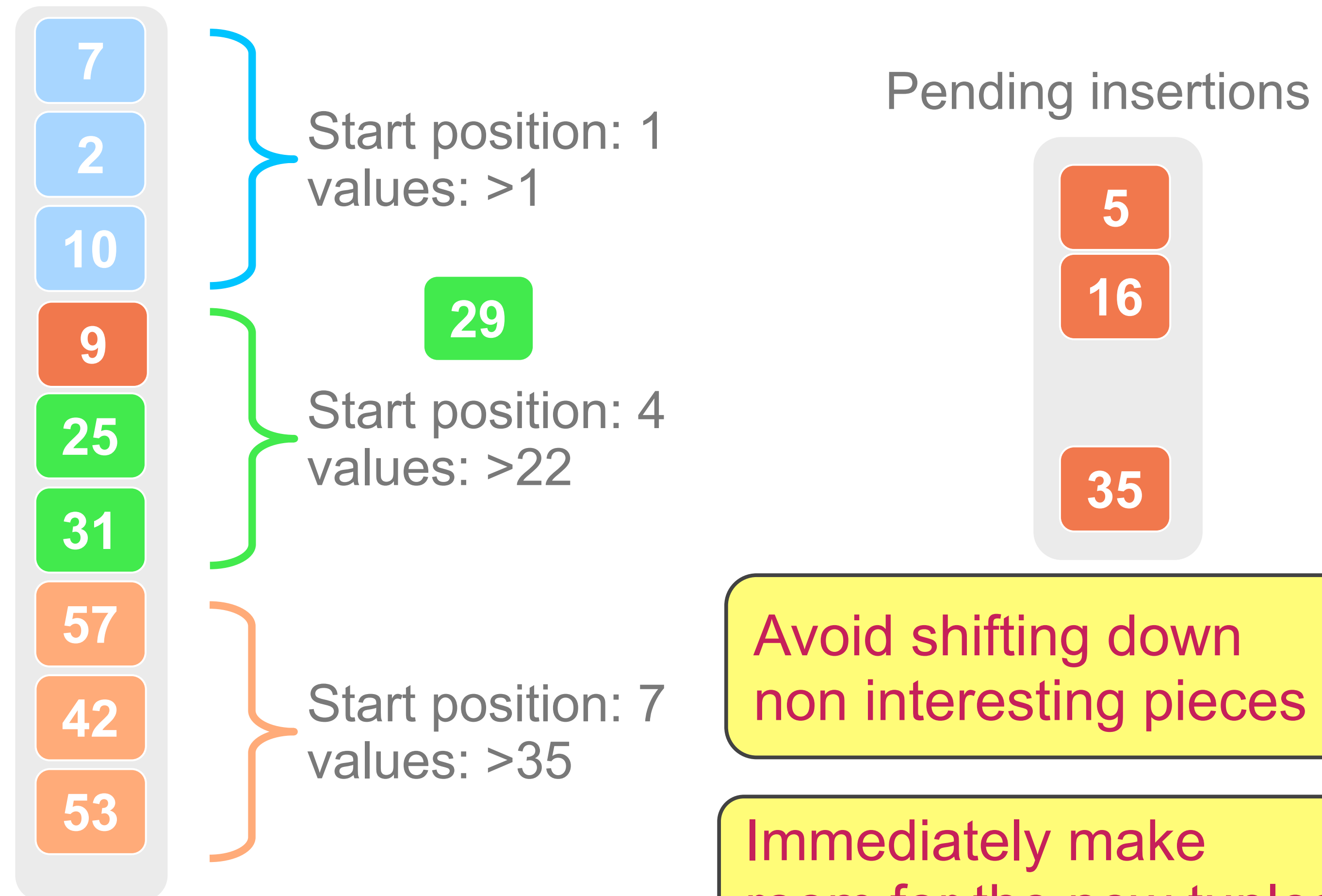
Touch only the pieces that are relevant for the current query

Select 7<= A< 15

The Ripple

Touch only the pieces that are relevant for the current query

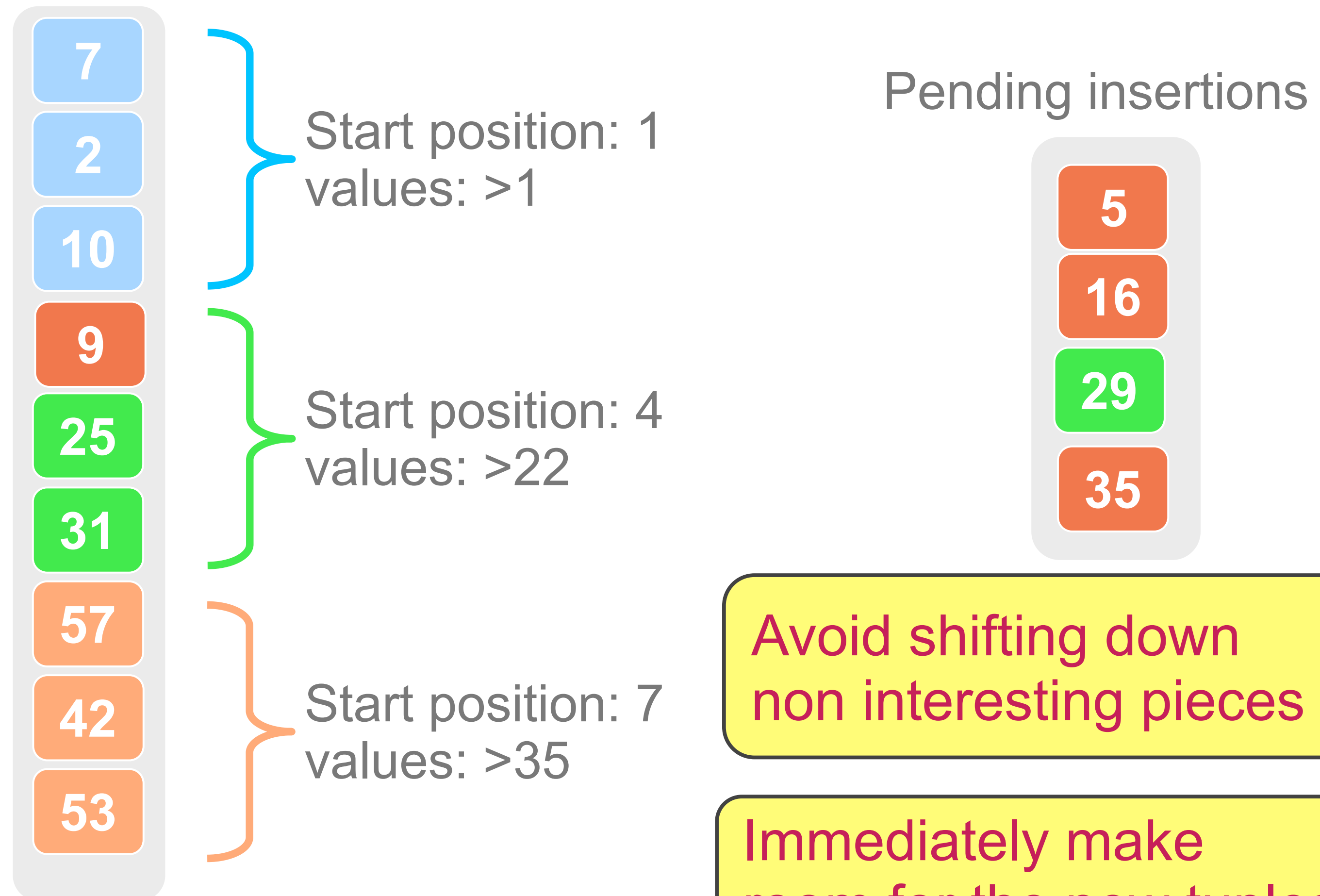
Select 7<= A< 15



The Ripple

Touch only the pieces that are relevant for the current query

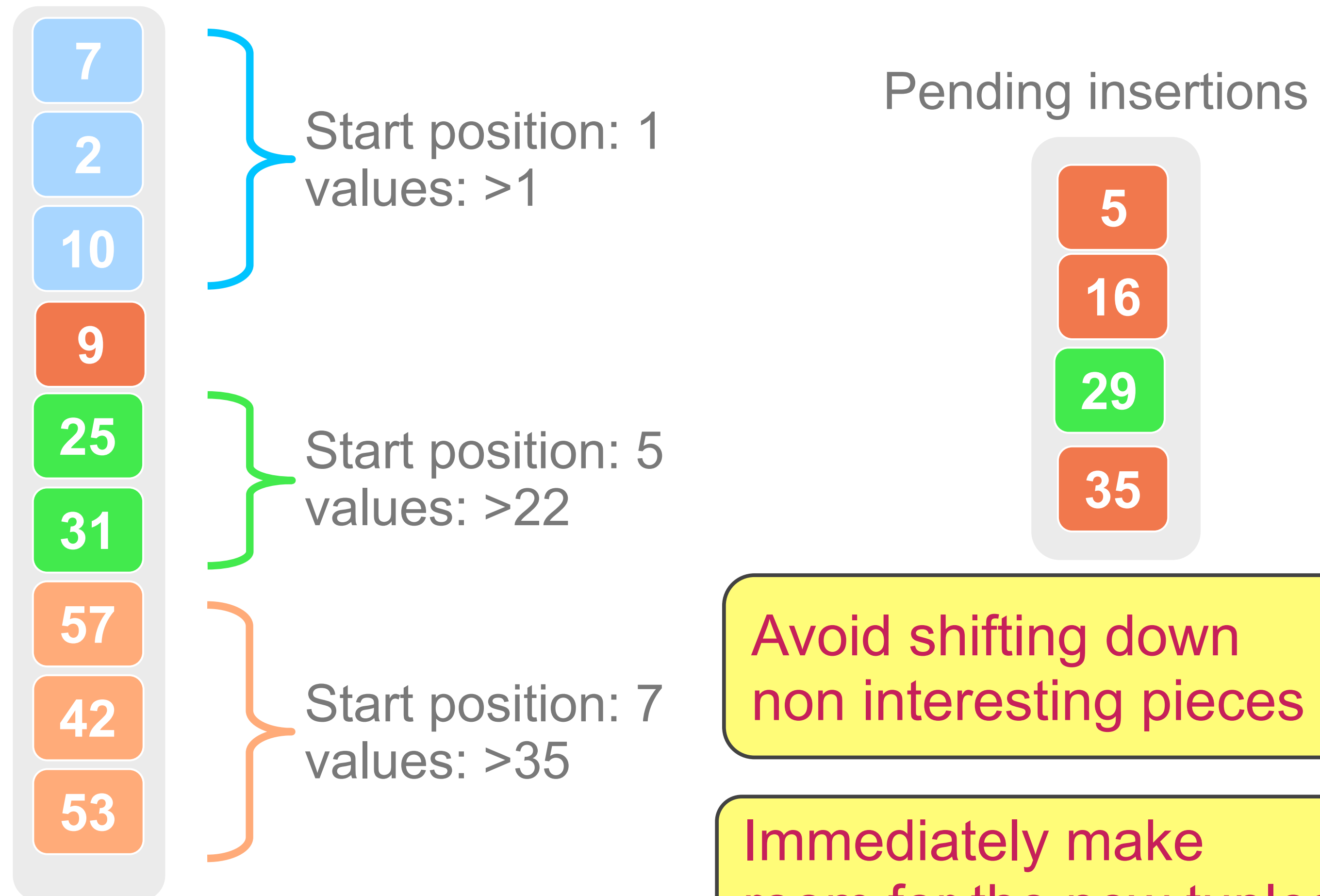
Select 7<= A< 15



The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

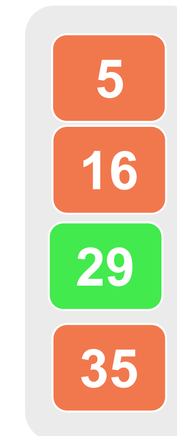


The Ripple

Touch only the pieces that are relevant for the current query

Select 7<= A< 15

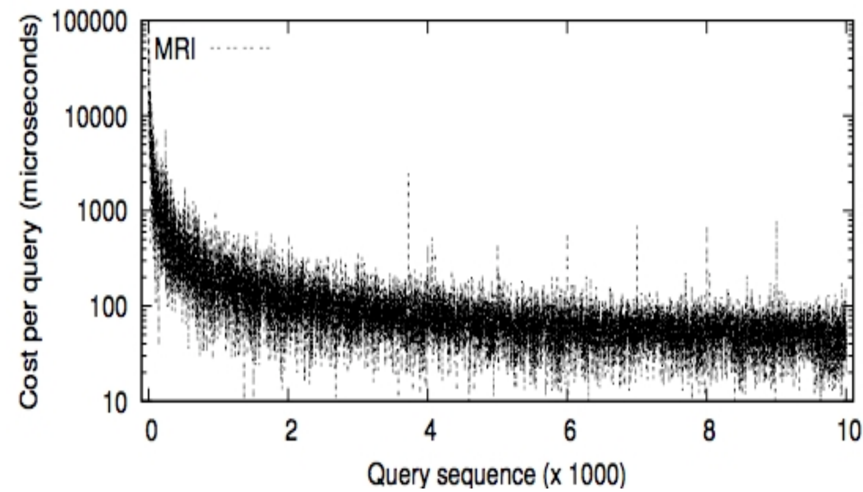
Pending insertions



Avoid shifting down non interesting pieces

Immediately make room for the new tuples

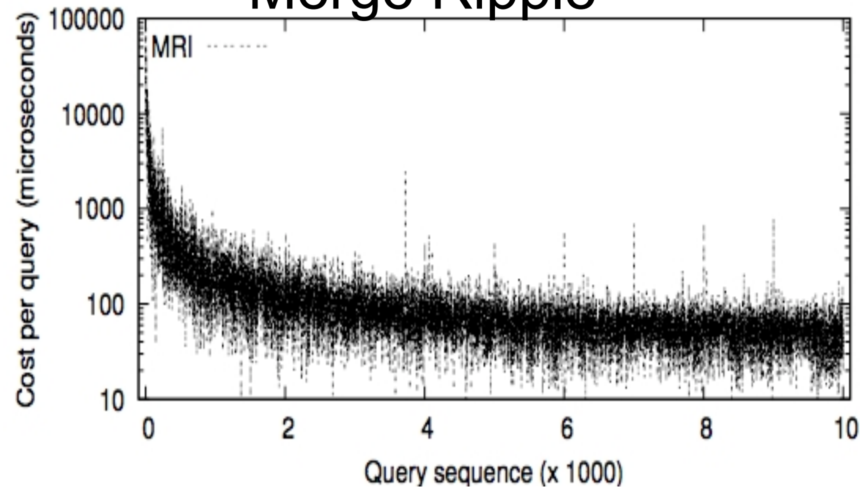
The Ripple



Maintain high performance through the whole query sequence in a self-organizing way

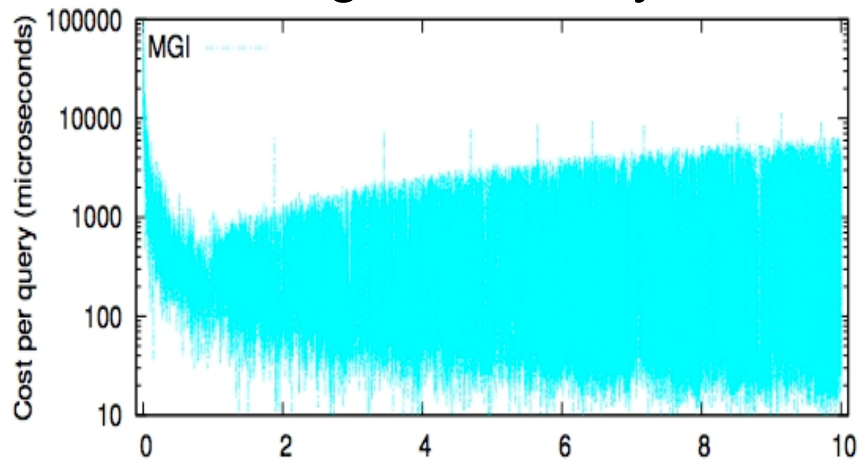
The Ripple

Merge Ripple

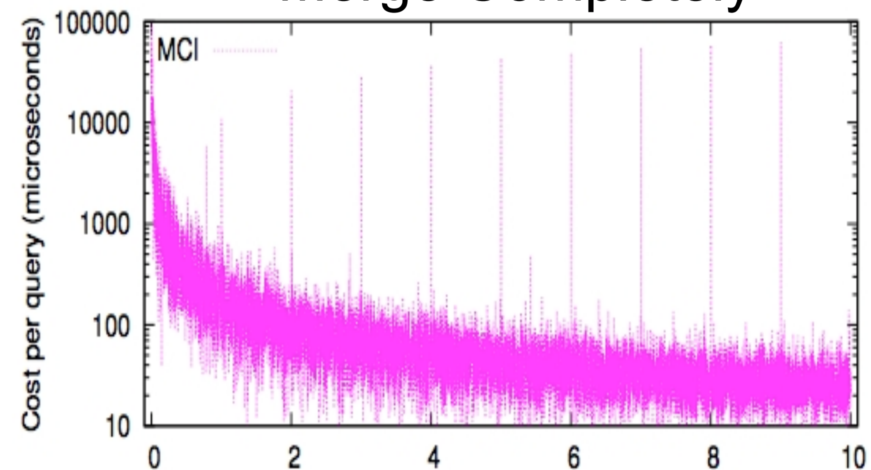


Maintain high performance through the whole query sequence in a self-organizing way

Merge Gradually



Merge Completely



Recycling intermediates

MonetDB Background

- Operator-at-a-time execution paradigm
- Canonical implementation of a column-store
  - Reduced dimensionality
  - Finer granularity
  - Simplified overlap analysis
- Recycler extension of MonetDB engine

MonetDB Architecture

```
function user.s1_2(A0:date, ...):void;
  X5 := sql.bind("sys","lineitem",...);
  X10 := algebra.select(X5,A0);
  X12 := sql.bindIdx("sys","lineitem",...);
  X15 := algebra.join(X10,X12);
  X25 := mtime.addmonths(A1,A2);
  ...
```

```
function user.s1_2(A0:date, ...):void;
  X5 := sql.bind("sys","lineitem",...);
  X10 := algebra.select(X5,A0);
  X12 := sql.bindIdx("sys","lineitem",...);
  X15 := algebra.join(X10,X12);
  X25 := mtime.addmonths(A1,A2);
  ...
```

SQL

XQuery

MAL

Tactical Optimizer

Recycler Optimizer

MAL

MonetDB Kernel

Run-time Support
Admission & Eviction

Recycle Pool

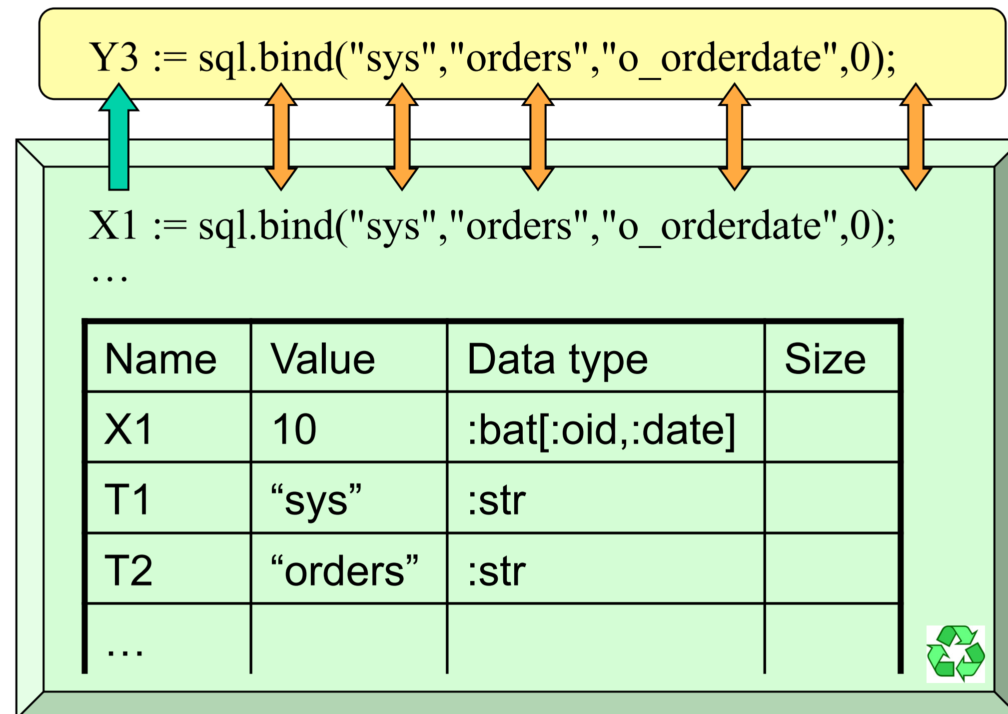
MonetDB Server

Instruction Matching

Run time comparison of

- instruction types
- argument values

Exact matching



Name	Value	Data type	Size
X1	10	:bat[:oid,:date]	
T1	"sys"	:str	
T2	"orders"	:str	
…			

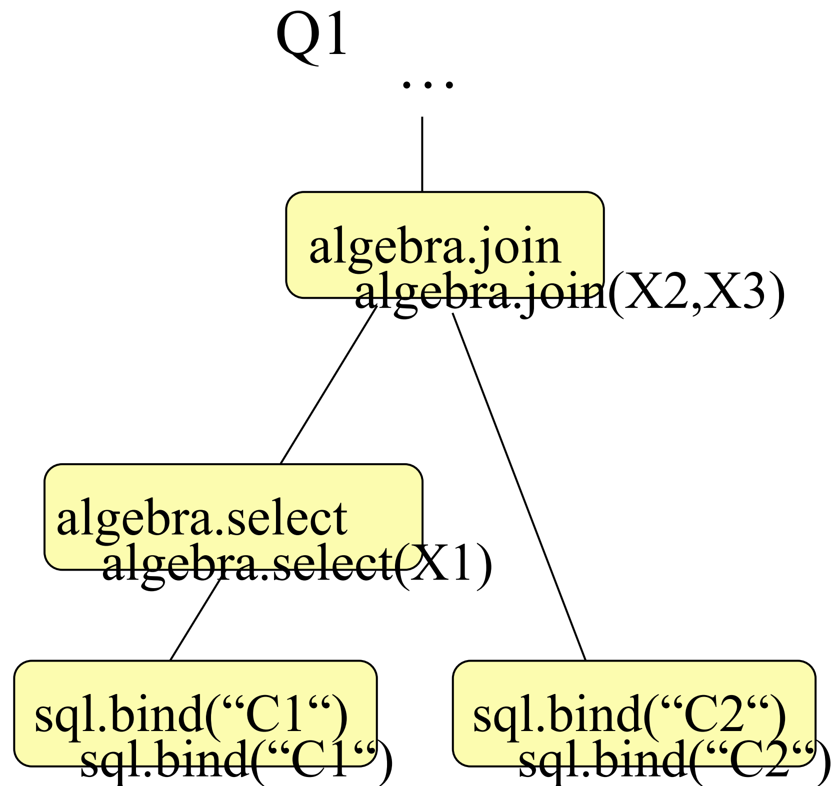
Instruction Subsumption

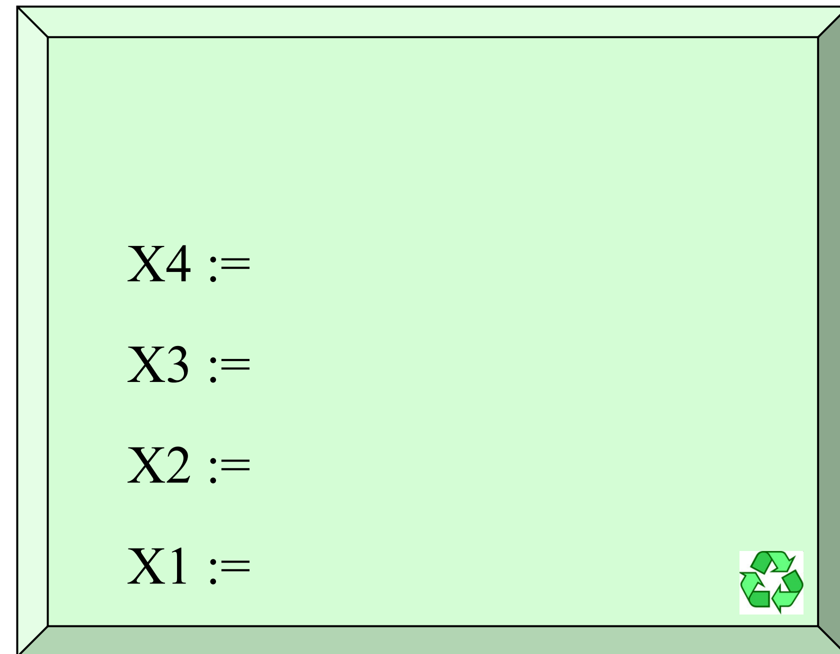
```
Y3 := algebra.select(X1,20,45);
```

```
X3 := algebra.select(X1,10,80);
...
X5 := algebra.select(X1,20,60);
```

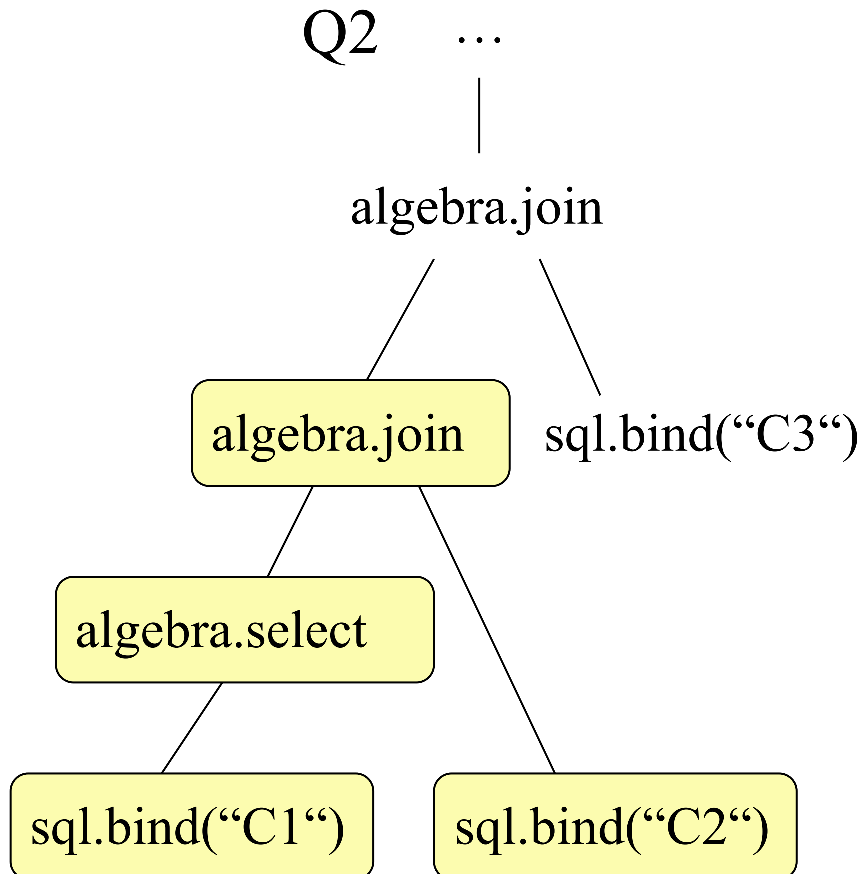
Name	Value	Data type	Size
X1	10	:bat[:oid,:int]	2000
X3	130	:bat[:oid,:int]	700
X5	150	:bat[:oid,:int]	350
…			

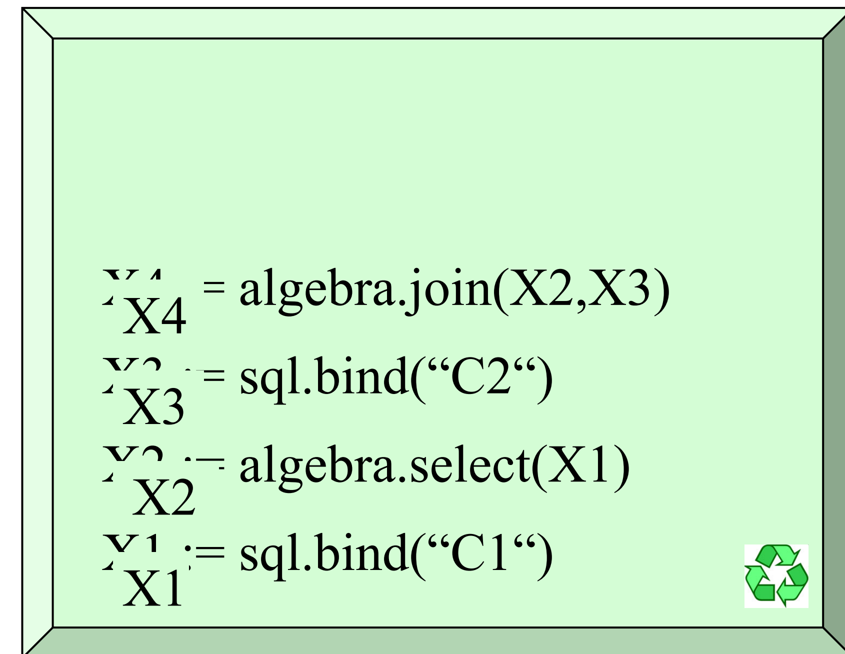
Recycle Pool: a Cache with Lineage



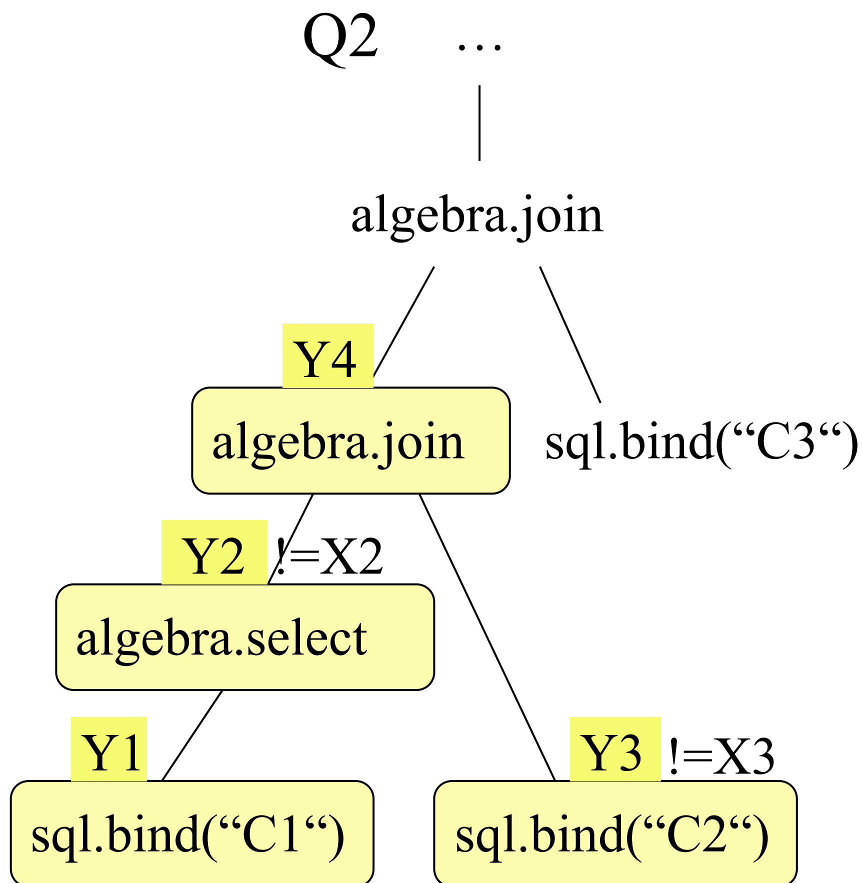


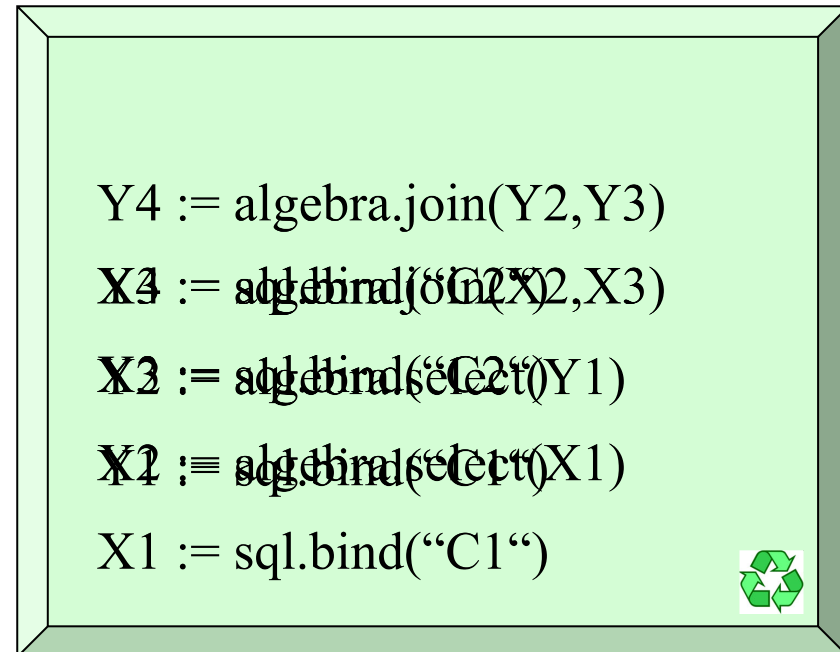
Recycle Pool: a Cache with Lineage





Mismatching





Admission Policies

Decide about storing the results

- KEEPALL
  - all instructions advised by the optimizer
- CREDIT
  - instructions supplied with credits
  - storage ‘paid’ with 1 credit
  - reuse returns credits
  - lack of reuse limits admission and resource claims

Cache Policies

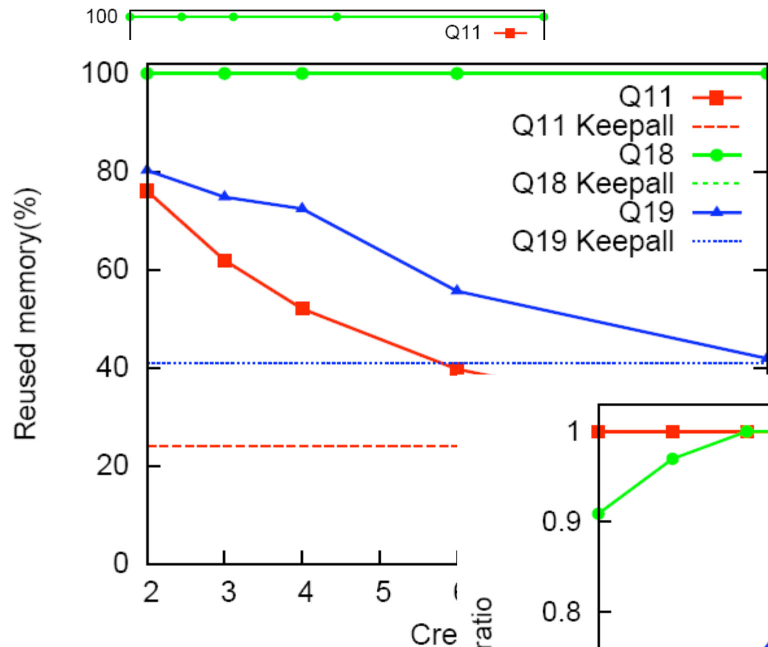
- Decide about eviction of intermediates
- Filter ‘top’ instructions without dependents
- Pick instructions with smallest utility
  - LRU : time of computation or last reuse
  - BENEFIT : estimated contribution to performance: CPU and I/O costs, recycling
- Triggered by resource limitations (memory or entries)

TPC-H Evaluation

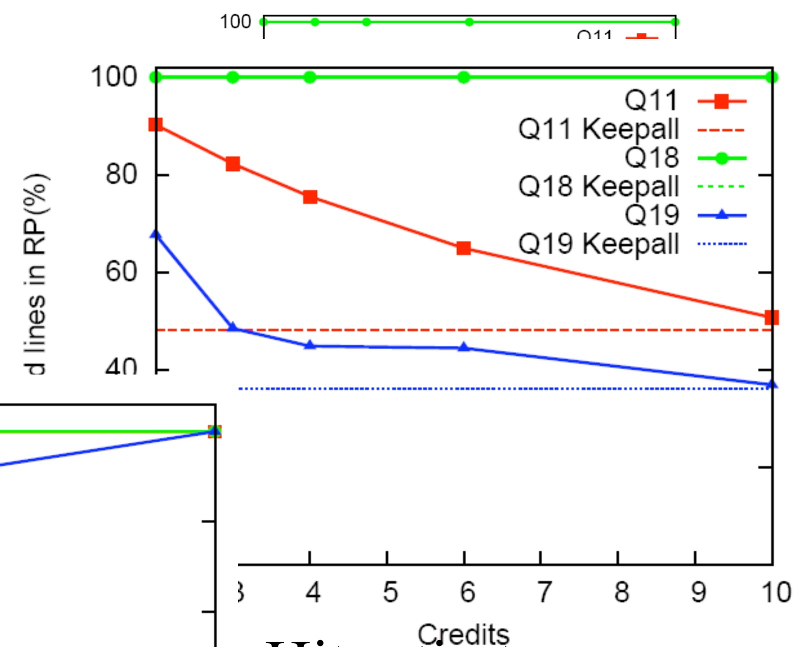
- SF1
- Baseline performance
- Impact of design choices
  - Admission policies
  - Cache policies

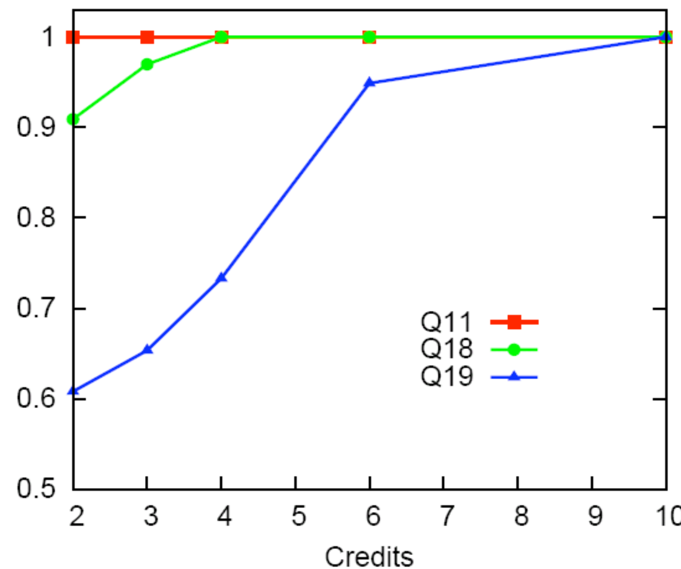
CREDIT Admission Impact

Reused memory



Reused entries



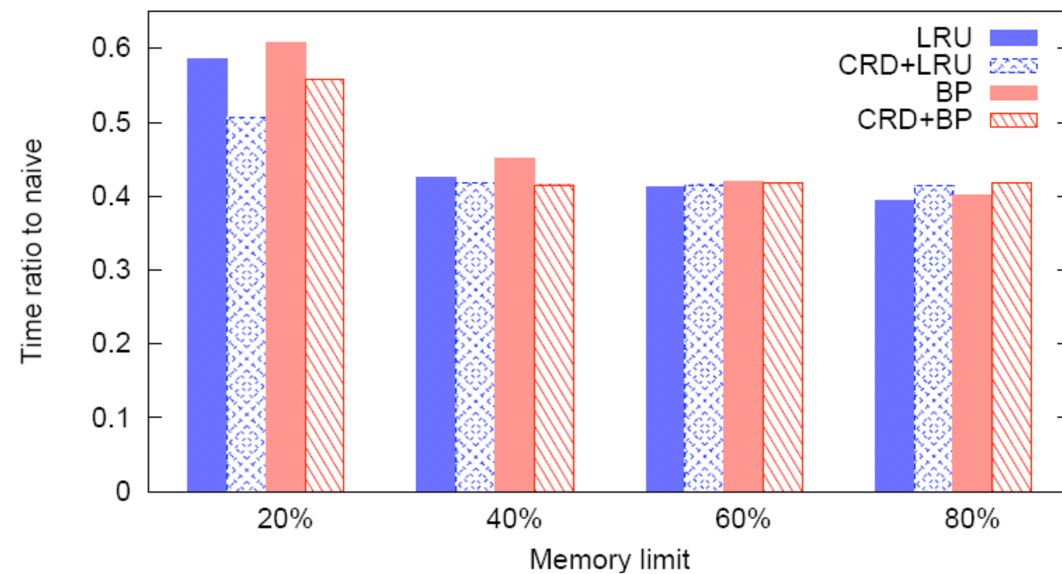


Hit ratio to KeepAll

Cache Policies Evaluation

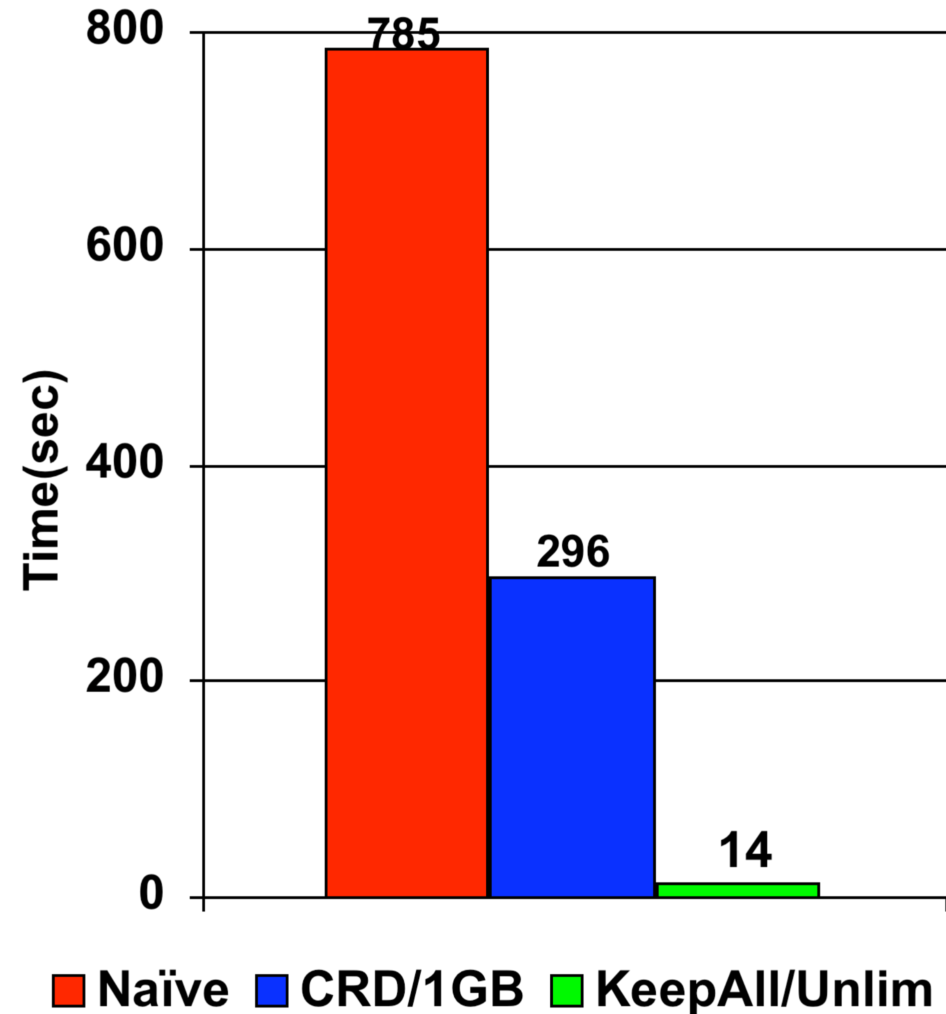
200 TPC-H queries

RP	Total	Reuse
Memory	4GB	42.7%
Entries	5219	28%



SkyServer Evaluation

- 100 GB subset of DR4
- 100-query batch from January 2008 log
- 1.5GB intermediates, 99% reuse
- Join intermediates major contributor to savings



Summary

- Database architecture augmented with recycling intermediates
- Significant performance benefits in SkyServer and TPC-H
- Self-organizing technique
- Extension to MonetDB transforming materialization overhead into benefit

Future Work

- Refining and developing admission and cache policies
- Opportunities by query class recognition
- Automatic switch to suitable policies

- Application to pipelined architectures

Recycling

Is

Green