Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

The Fifth International Conference on Advances in Databases, Knowledge, and Data Applications

January 27 - February 1, 2013 - Seville, Spain

Keynote: About Bitmap Indexes

Andreas Schmidt

(andreas.schmidt@kit.edu)

Department of Informatics and Business Information Systems
University of Applied Sciences Karlsruhe
Moltkestraße 30
76133 Karlsruhe
Germany

Institute for Applied Sciences
Karlsruhe Institute of Technologie
PO-box 3640
76021 Karlsruhe
Germany

Karlsruhe Institute of Technology

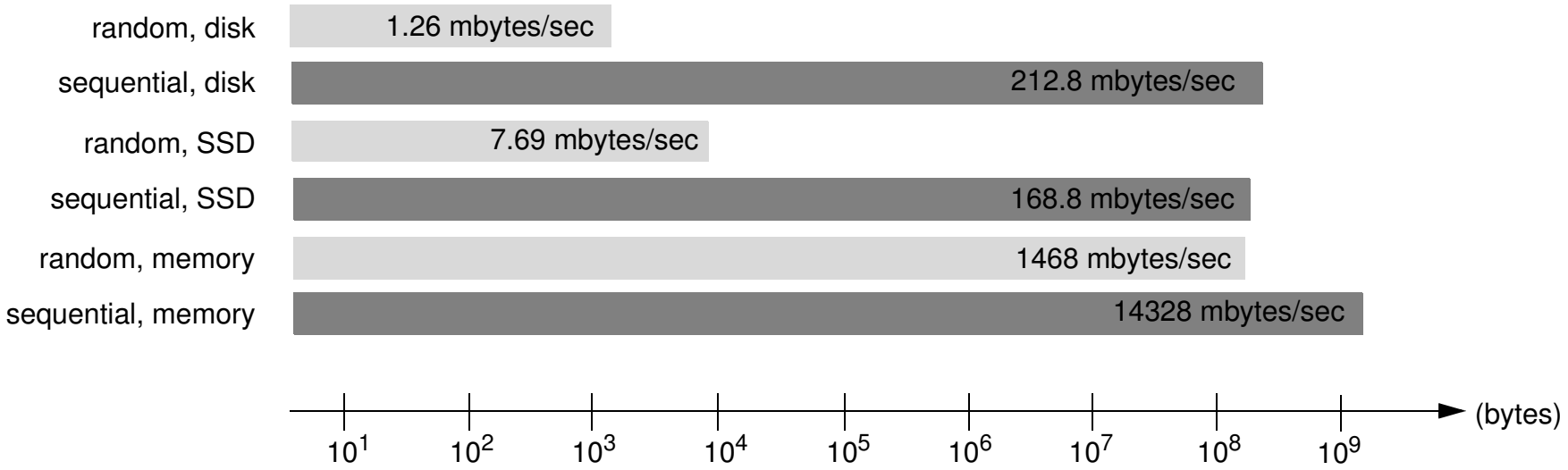
Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Outline

- Motivation
- What is a bitmap index ?
- Possible encodings
  - equality encoding
  - range encoding
  - interval encoding
- High cardinality attributes (algorithms, ...)
  - Encoding
  - Binning
  - Compression
- Bitmaps Indexes in Column-Stores
- Summary

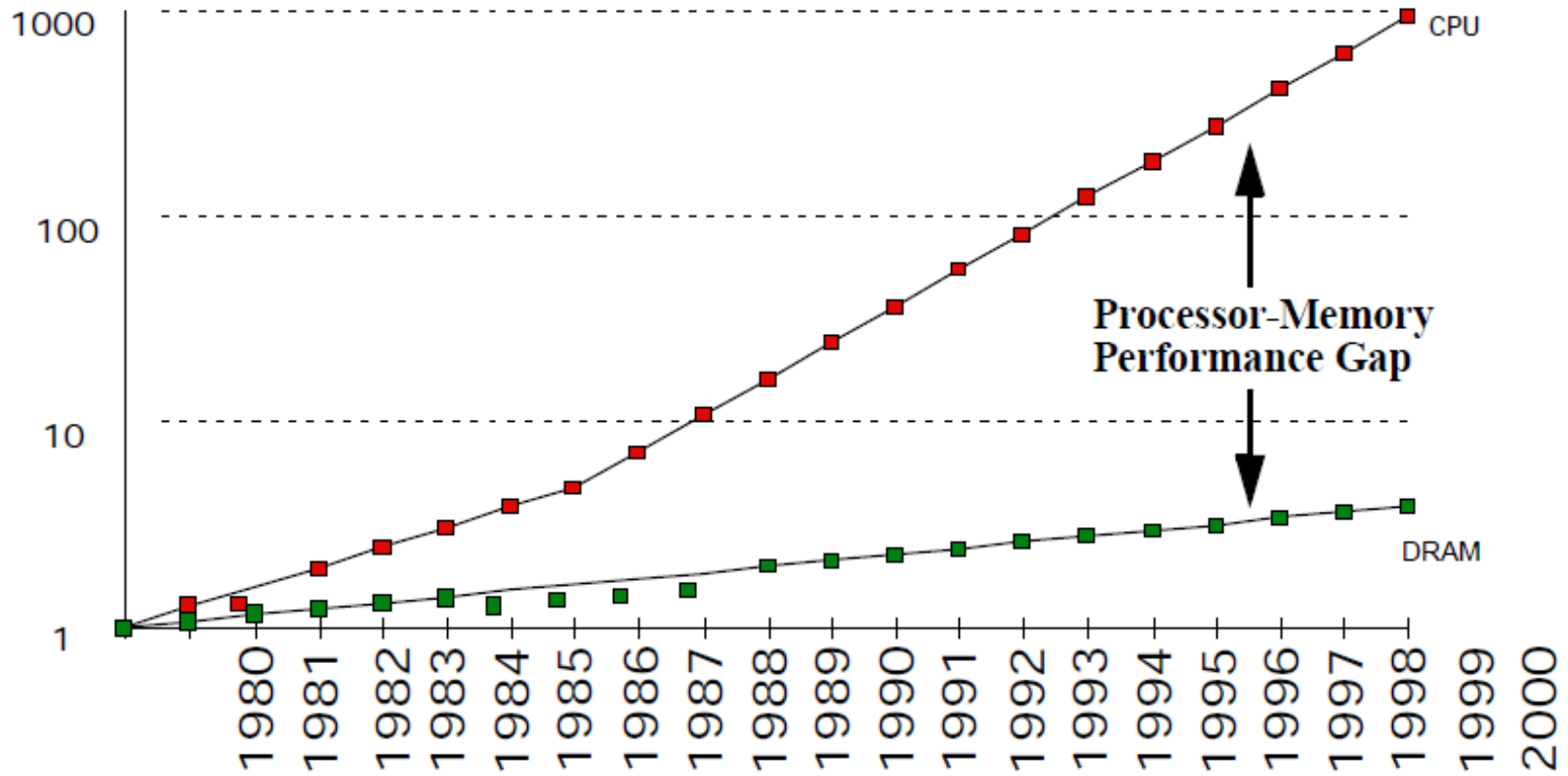
Motivation: Access speed

Comparision of random and sequential memory access



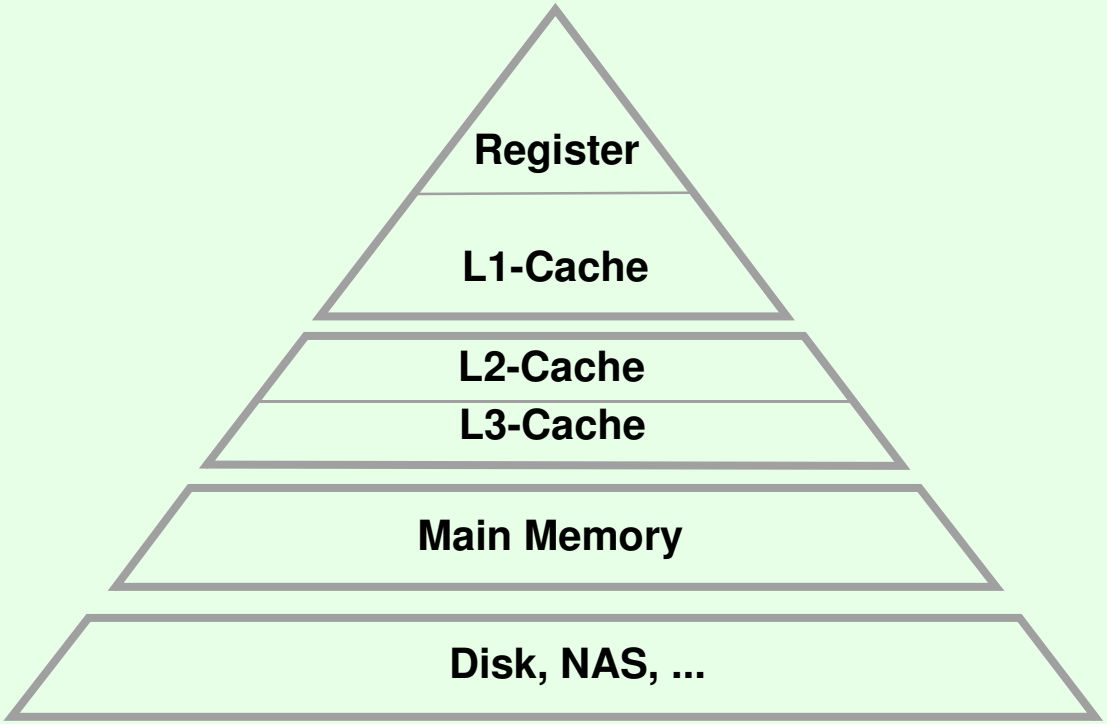
Source: Adam Jacobs: The Pathologies of Big Data, acmqueue, July 2009

CPU Memory Gap



Source: Hennessy, J.L.; Patterson, D.A. *Computer Organization and Design*, 2nd ed. San Francisco: Morgan Kaufmann Publishers, 1997.

Memory Hierarchy



typical access times:

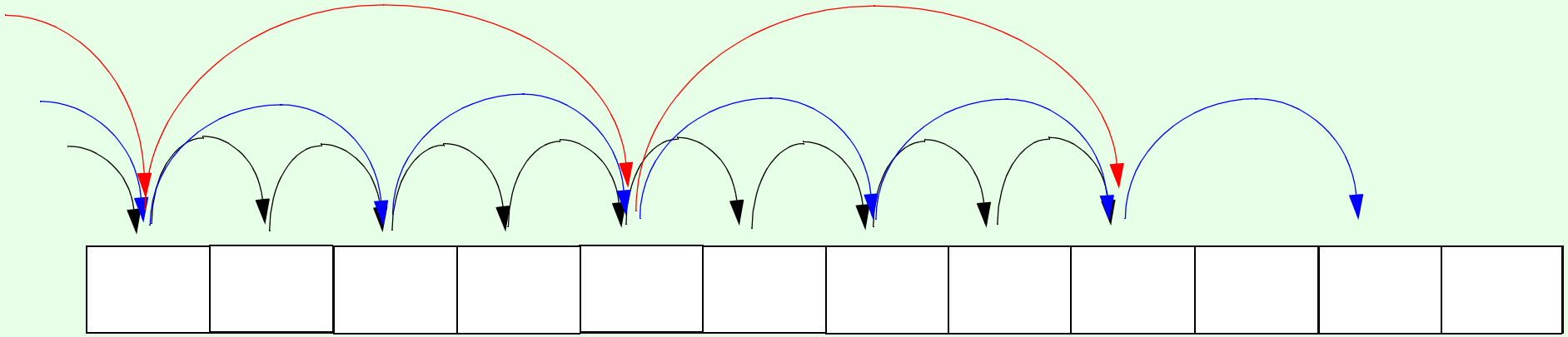
register:	0.5 ns
L1 cache:	2 ns
L2 cache:	6 ns
Main Memory:	60 ns

[Car02] Carlos Carvalho: The Gap between Processor and Memory Speeds, ICCA, 2002

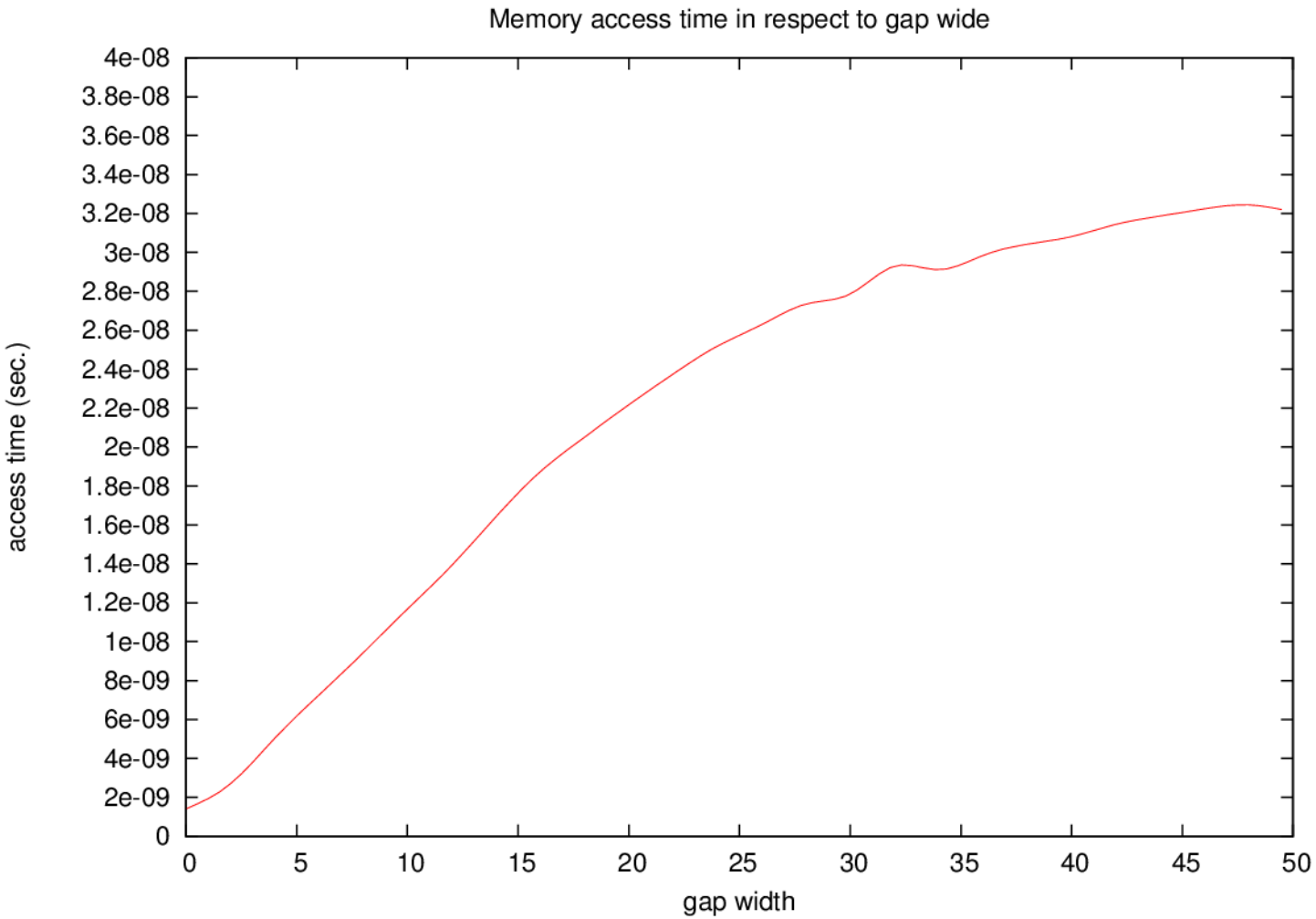
Motivation: Cache Consciousness

Motivation experiment:

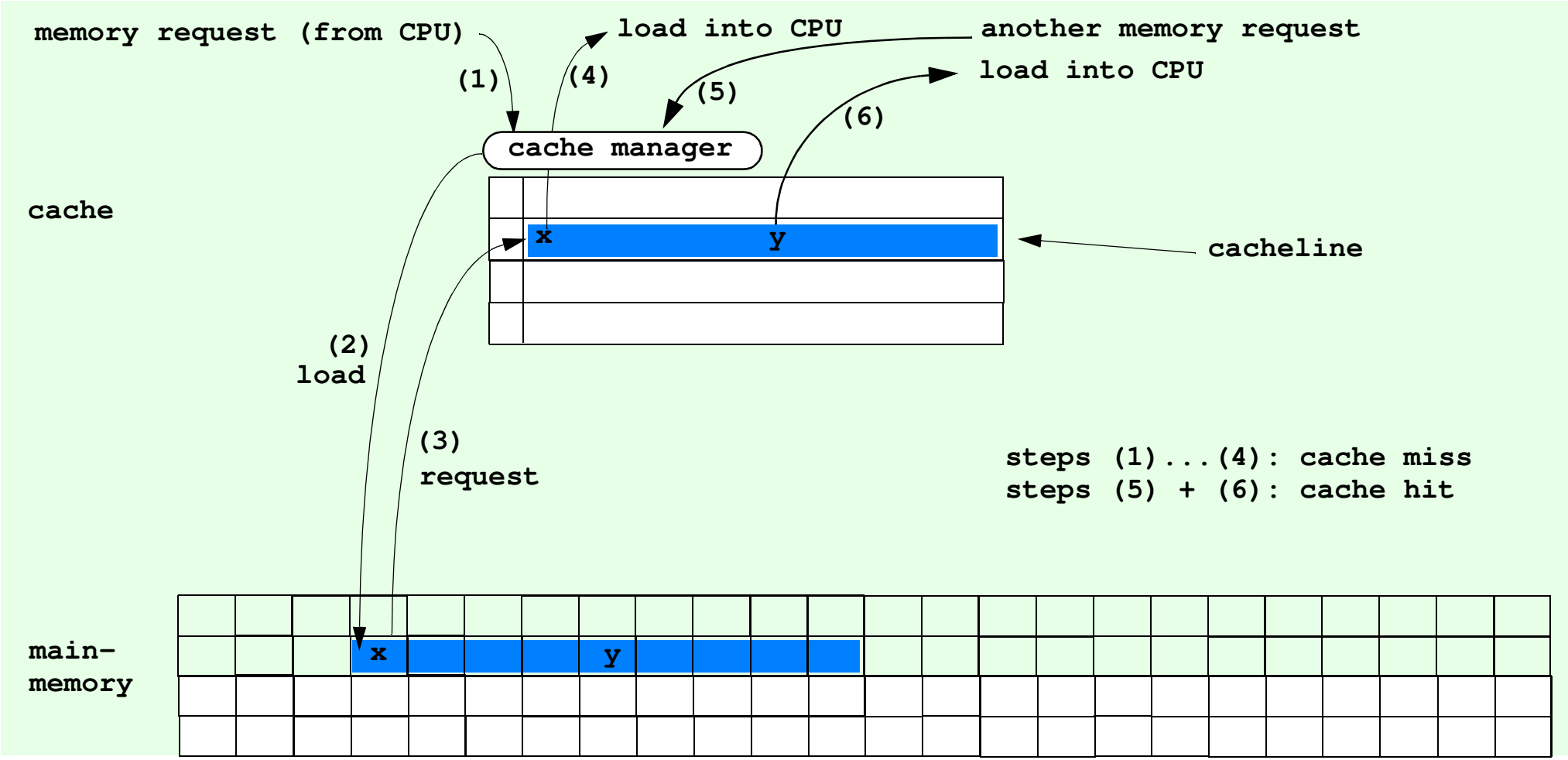
```
clock->start();
uint i = 0;
for (uint k = 1; k < steps; k++) {
   x += mem[i];
   i += stepwide;
}
clock->stop();
cout << stepwide << "\t" << clock->CPUTime() << endl;
```



Access time in respect to gap width



Cache & Cacheline



Bitmap Index: Basic Idea

- A bitmap consists of a number of 0/1 values (1 bit) in sequence.
- The length of a bitmap is defined by the number of datasets which must be indexed (n)
- For an attribute with *m* possible values (cardinality) the bitmap-index consists of *m* bitmaps
- Each possible value of an attribute is associated with one bitmap.
- The occurence of a 1-bit at position *k* means, that dataset *k* has the correponding value of that bitmap

Bitmap Index: Memory Consumption

- Table with $1 * 10^6$ datasets, Attribute A, cardinality: 2 (gender)

Bitmap:

A = {male, female}

memory consumption:

mem = $(2 * 10^6)/8 = 2.5 * 10^5$ bytes

Test for specific value (i.e. 'male'):

mem_{test} = 125000 bytes, sequential

B+-Tree:

A = {male, female}

memory consumption (only leafs):

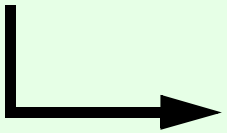
mem = $4 * 10^6$

Test for specific value (i.e. 'male'):

mem_{test} = at least $2 * 10^6$, sequential/ random

Simple Bitmap Example (equality encoding)

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed



single	married	widowed
1	0	0
0	0	1
0	1	0
1	0	0
0	0	0
1	0	0
0	1	0
0	0	1

Simple Bitmap Example (equality encoding)

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
	41	M	married
	35	M	single
	77	F	
	22	F	single
	47	M	married
	81	F	widowed

81	77	53	47	41	35	34	22
0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0

single	married	widowed
1	0	0
0	0	1
0	1	0
1	0	0
0	0	0
1	0	0
0	1	0
0	0	1

Equality encoded bitmaps

- One bitmap per distinct attribute value
- Optimized for exact match queries like: $Q_e : a_i = v_j$
- Instead of reading the plain data and evaluating the query, the matching bitmap is used to find the matching tuples
- Advantage:
  - Much less data to read
  - Bitmap is cache conscious
  - Hardware support for finding next ’1’ bit
- Queries like Q : a_1 = v1 OR $a_1 = v_2$ OR $a_1 = v_3$ are mapped on hardware supported OR-operation over multiple bitmaps
- Multidimensional queries like Q : $a_1 = v_1$ AND $a_2 = v_2$ AND $a_3 = v_3$ are mapped on hardware supported AND-operation over multiple bitmaps

Range Encoding

- What is about queries of type: Q_e : age >= 40:
  - OR-operations over all the bitmaps for values 41, 47, 53, 77, 81
  - use of another encoding schema

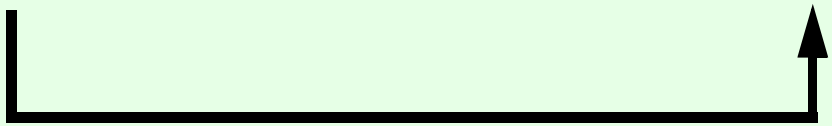
- Range Encoding
  - Encoding: like equality encoding, but additionally set all bits at position k for bit-maps with lower values also to ’1’
  - At most two bitmaps need to be accessed per query

[CY98] Chee-Yong Chan und Yannis Ioannidis: Bitmap Index Design and Evaluation. Proceedings of the 1998 ACM SIGMOD Conference.

Equality Encoding

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

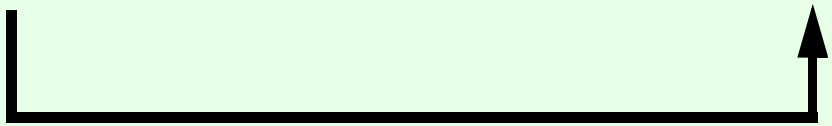
[22]	[34]	[35]	[41]	[47]	[53]	[77]	[81]
0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1



Range Encoding [CY98]

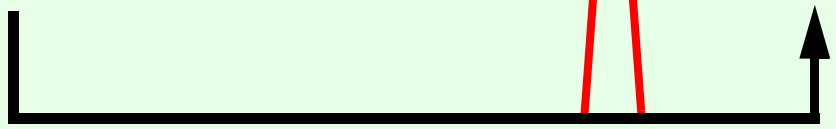
name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[22]	[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1



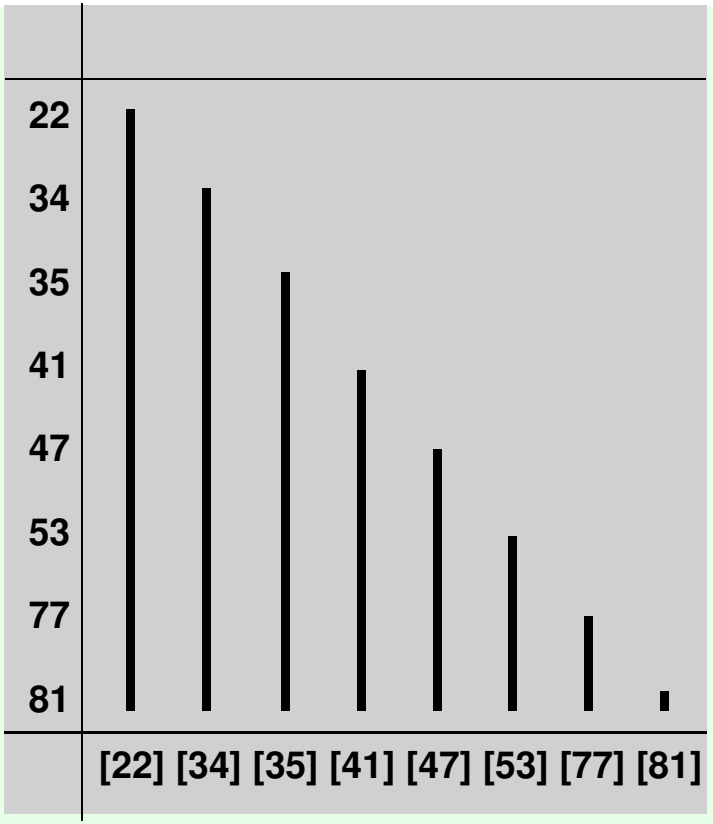
name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[22]	[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1



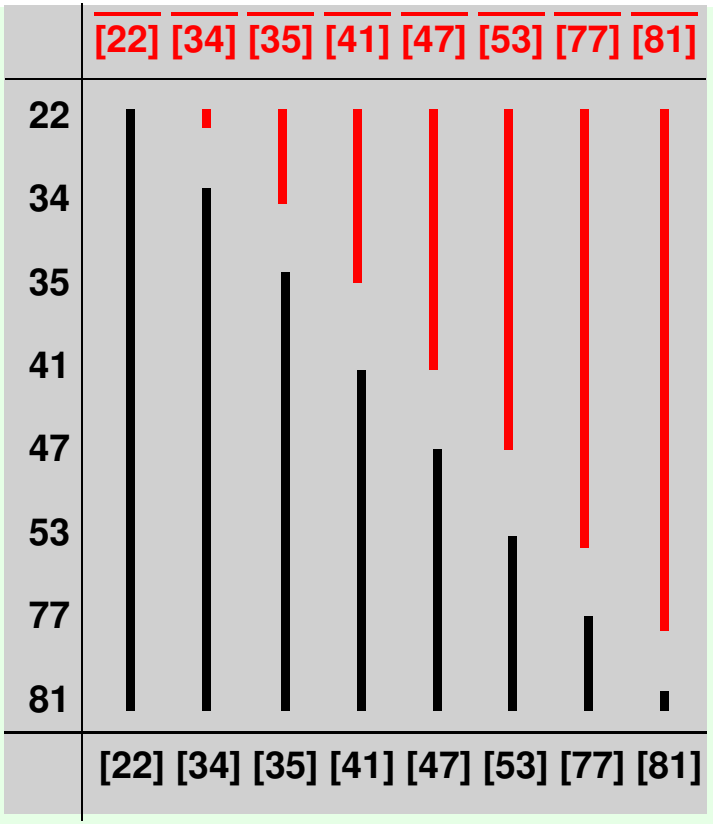
Range Encoding [CY98]

	[22]	[34]	[35]	[41]	[47]	[53]	[77]	[81]
34	1	1	0	0	0	0	0	0
53	1	1	1	1	1	1	0	0
41	1	1	1	1	0	0	0	0
35	1	1	1	0	0	0	0	0
77	1	1	1	1	1	1	1	0
22	1	0	0	0	0	0	0	0
47	1	1	1	1	1	0	0	0
81	1	1	1	1	1	1	1	1



Range Encoding [CY98]

	[22]	[34]	[35]	[41]	[47]	[53]	[77]	[81]
34	1	1	0	0	0	0	0	0
53	1	1	1	1	1	1	0	0
41	1	1	1	1	0	0	0	0
35	1	1	1	0	0	0	0	0
77	1	1	1	1	1	1	1	0
22	1	0	0	0	0	0	0	0
47	1	1	1	1	1	0	0	0
81	1	1	1	1	1	1	1	1



Query: age >= 41

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	0	0	0	0	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	1	1	1	0
0	0	0	0	0	0	0
1	1	1	1	0	0	0
1	1	1	1	1	1	1

Query: age >= 41

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	0	0	0	0	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	1	1	1	0
0	0	0	0	0	0	0
1	1	1	1	0	0	0
1	1	1	1	1	1	1

Query: age >= 41

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	0	0	0	0	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	1	1	1	0
0	0	0	0	0	0	0
1	1	1	1	0	0	0
1	1	1	1	1	1	1

Query: age < 41

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	0	0	0	0	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	1	1	1	0
0	0	0	0	0	0	0
1	1	1	1	0	0	0
1	1	1	1	1	1	1

Query: age < 41

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	1	0	0	0	0
1	1	0	1	1	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0
1	1	0	1	1	1	0
0	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	0	1	1	1	1

Query: age < 41

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	1	0	0	0	0
1	1	0	1	1	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0
1	1	0	1	1	1	0
0	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	0	1	1	1	1

Query: 35 =< age < 53

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	0	0	0	0	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	1	1	1	0
0	0	0	0	0	0	0
1	1	1	1	0	0	0
1	1	1	1	1	1	1

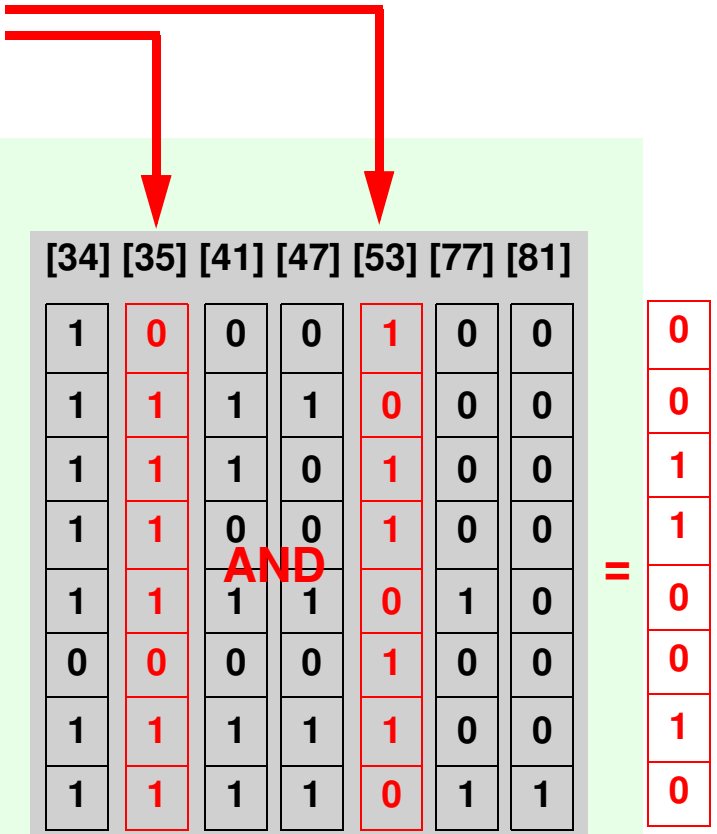
Query: 35 =< age < 53

name	age	sex	family_stat
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]
1	0	0	0	1	0	0
1	1	1	1	0	0	0
1	1	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	1	0	1	0
0	0	0	0	1	0	0
1	1	1	1	1	0	0
1	1	1	1	0	1	1

Query: 35 =< age < 53

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed



Query: 35 =< age < 53

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[34]	[35]	[41]	[47]	[53]	[77]	[81]	
1	0	0	0	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	1	0	0	1
1	1	0	0	1	0	0	1
AND							=
1	1	1	1	0	1	0	0
0	0	0	0	1	0	0	0
1	1	1	1	1	0	0	1
1	1	1	1	0	1	1	0

Interval Encoding

Basic Idea:

- Find $\lceil |A| / 2 \rceil$ intervals, so that every equality, range or interval query could be answered using at most 2 bitmaps

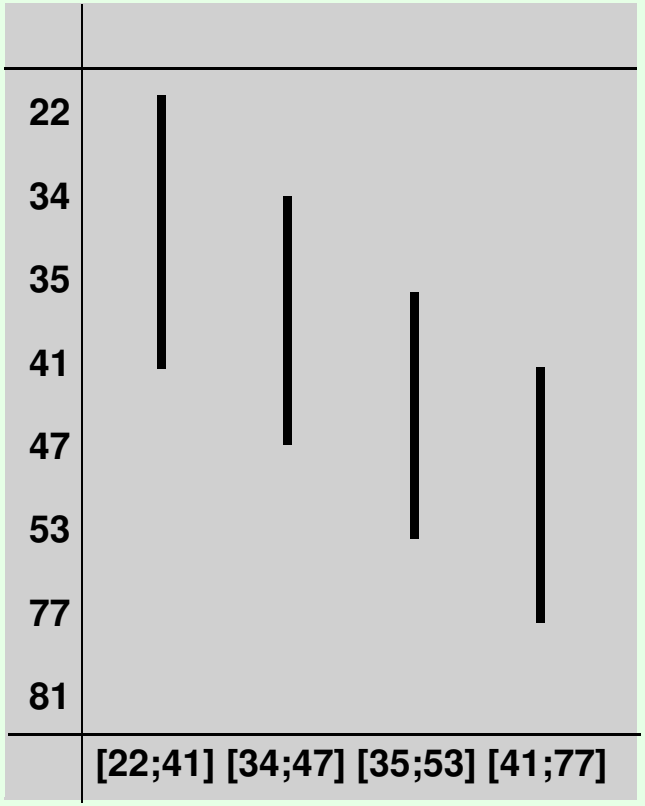
name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[22;41]	[34;47]	[35;53]	[41;77]
1	1	0	0
0	0	1	1
1	1	1	1
1	1	1	0
0	0	0	1
1	0	0	0
0	0	1	1
0	0	0	0

Chee-Yong Chan and Yannis E. Ioannidis. 1999. An efficient bitmap encoding scheme for selection queries. In Proceedings of the 1999 ACM SIGMOD international conference on Management of data (SIGMOD '99). ACM, New York, NY, USA

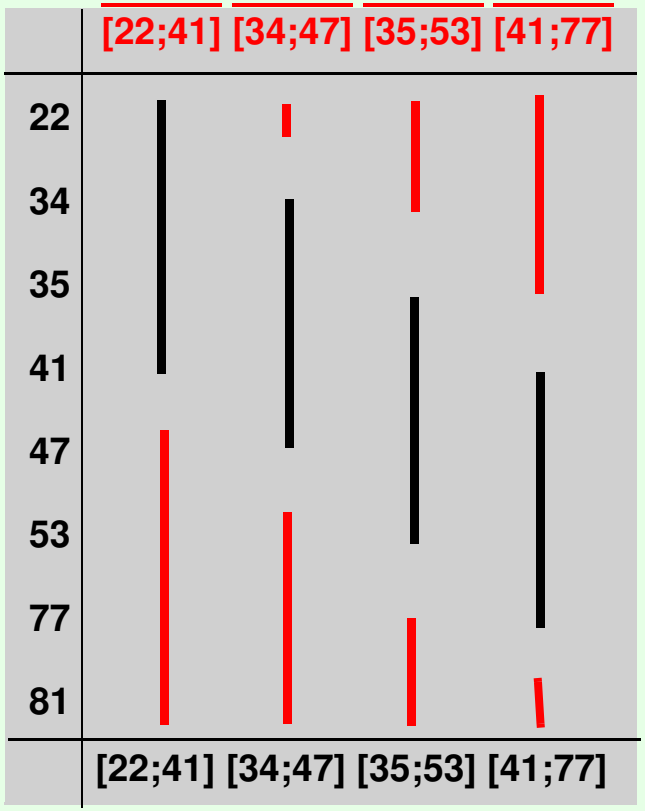
Interval Encoding

age	[22;41]	[34;47]	[35;53]	[41;77]
34	1	1	0	0
53	0	0	1	1
41	1	1	1	1
35	1	1	1	0
77	0	0	0	1
22	1	0	0	0
47	0	0	1	1
81	0	0	0	0



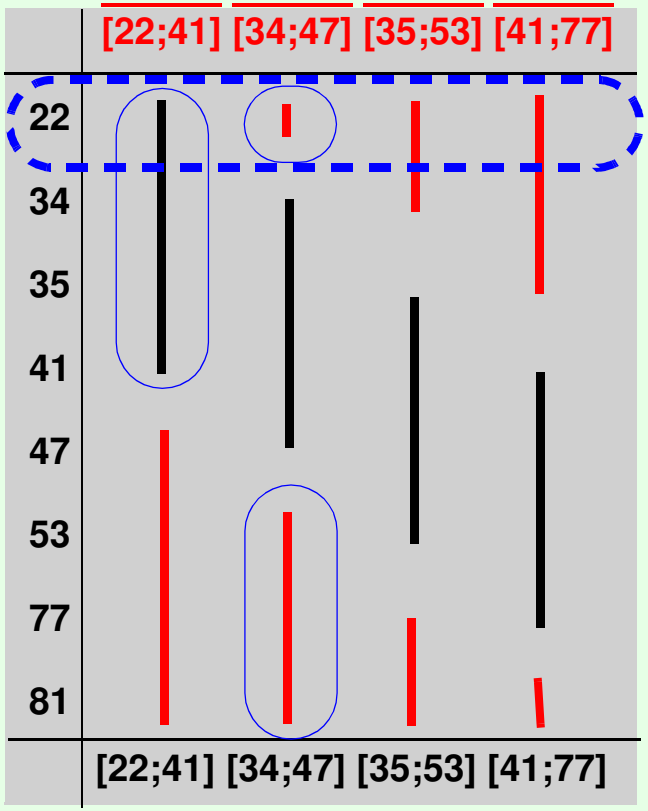
Interval Encoding

age	[22;41]	[34;47]	[35;53]	[41;77]
34	1	1	0	0
53	0	0	1	1
41	1	1	1	1
35	1	1	1	0
77	0	0	0	1
22	1	0	0	0
47	0	0	1	1
81	0	0	0	0



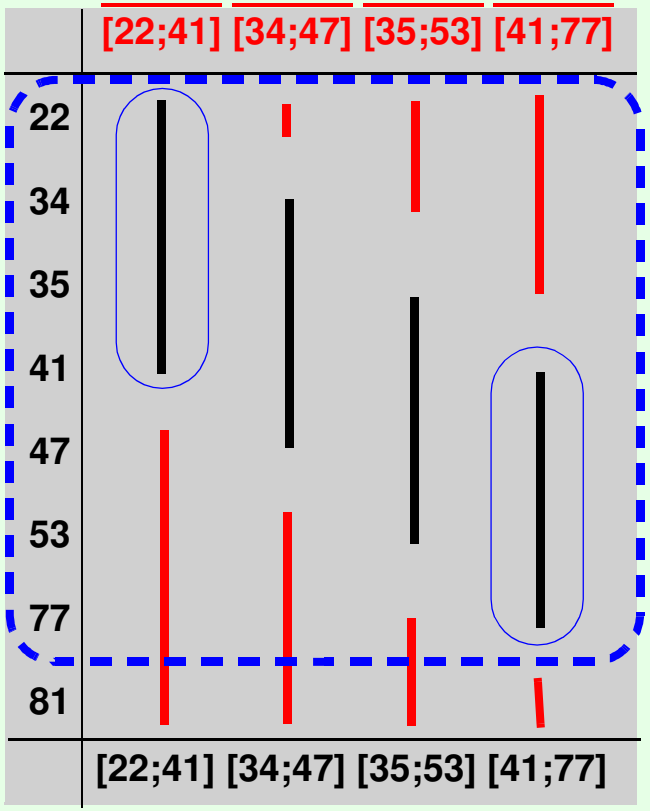
Interval Encoding

- Queries:
  - age = 22: [22; 41] AND [34;47]
  - age =< 77:
  - 35 < age =< 77:



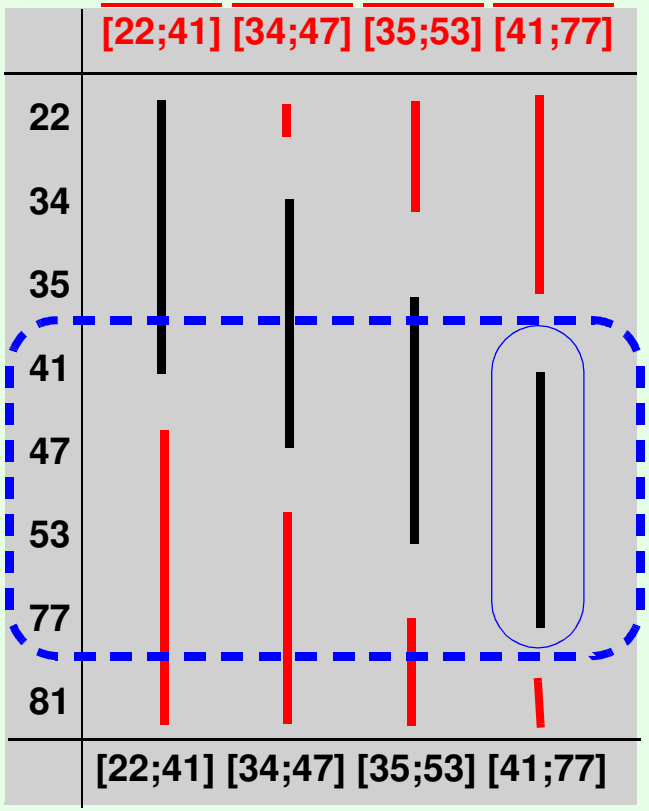
Interval Encoding

- Queries:
  - age = 22: [22; 41] AND $\overline{[34;47]}$
  - age =< 77: [22;41] OR [41;77]
  - 35 < age =< 77:



Interval Encoding

- Queries:
  - age = 22: [22; 41] AND $\overline{[34;47]}$
  - age =< 77: [22;41] OR [41;77]
  - 35 < age =< 77: [41;77]



Multidimensional Ad-Hoc Queries

- Bitmap-Index and B-trees are build for one attribute each
- Multi-dimensional queries require to access multiple indexes.
- Combining indexes (multidimensional query):
  - B-Tree: The final result consists of the intersection/union/... of the preliminary lists. Construction of final result is probably expensive (union semantics, duplicate elimination)
  - Bitmap-Index: Construction of final result is cheap and can be done in constant time (small, fixed size, hardware supported AND/OR operations, no sorting necessary)

sounds intuitive ... but what is about high dimensional cardinality attributes? i.e.floating point numbers ?

High dimensional data

- Number of bitmaps depends on the cardinality of an attribute.
  => High cardinality leads to a high number of bitmaps for an attribute
- Example: 1 Million datasets with floating point numbers in ascending order
  - Number of bitmaps: 10^6
  - Size of a bitmap: $10^6/8$ = 125000 bytes
  - Size of bitmap index: $1.25 * 10^{11}$ bytes - upps !!
- Solutions:
  - More compact encoding strategies (i.e. Wu and Buchmann, $\log_2(|A|)$, [WB98])
  - Binning
  - Compression

Binning

- One bitmap (bin) per value range
- Problem: not all results from the bitmap check belong to the final result
- Solution: Subsequent candidate check for the ambiguous values
- Goal: keep the number of values in the candidate check small
- Example: Algorithm GenericRangeEval [Stock01]

[Stock01] Stockinger, K.: Design and implementation of bitmap indices for scientific data, International Symposium on Database Engineering and Applications, 2001.

Binning Example

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

[0;20)	[0;40)	[0;60)	[0;80)	[0;100)
0	1	1	1	1
0	0	1	1	1
0	0	1	1	1
0	1	1	1	1
0	0	0	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	0	1

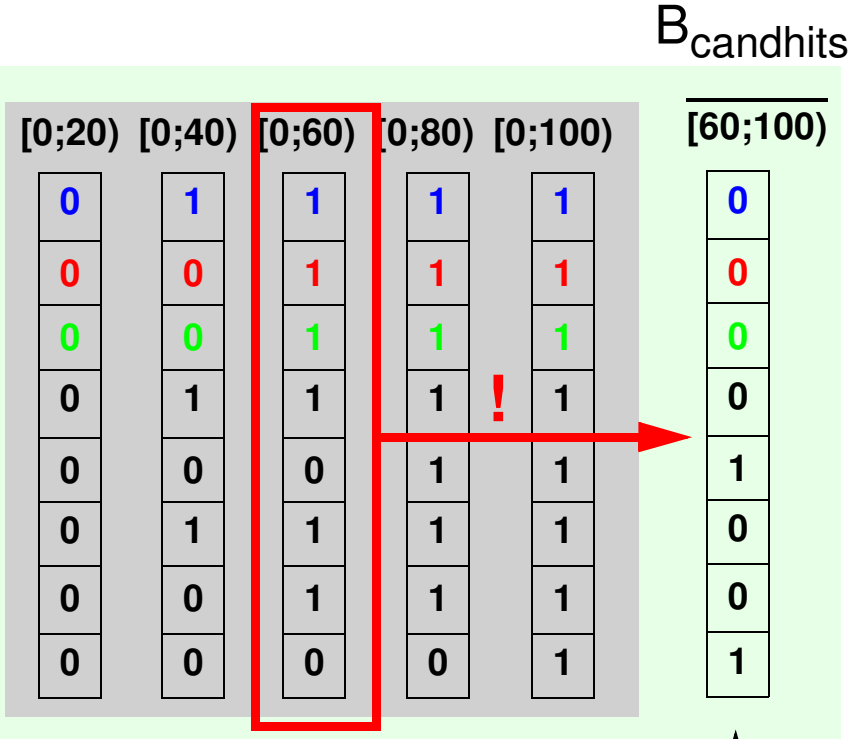
Query: age > 71

$B_{candhits} = \overline{[0;60)} = [60;100)$

$B_{cands} = [0;60)\ XOR\ [0;80)$

Binning Example

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed



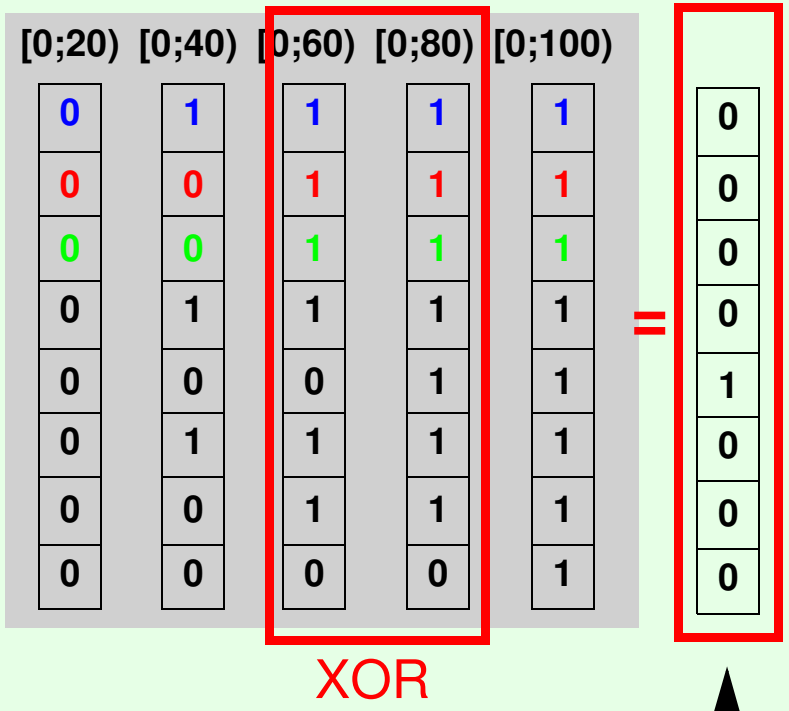
Query: age > 71

$B_{candhits} = \overline{[0;60)} = [60;100)$

$B_{cands} = [0;60)$ XOR $[0;80)$

Binning Example

name	age	sex	family_state
...	34	M	single
...	53	F	widowed
...	41	M	married
...	35	M	single
...	77	F	
...	22	F	single
...	47	M	married
...	81	F	widowed

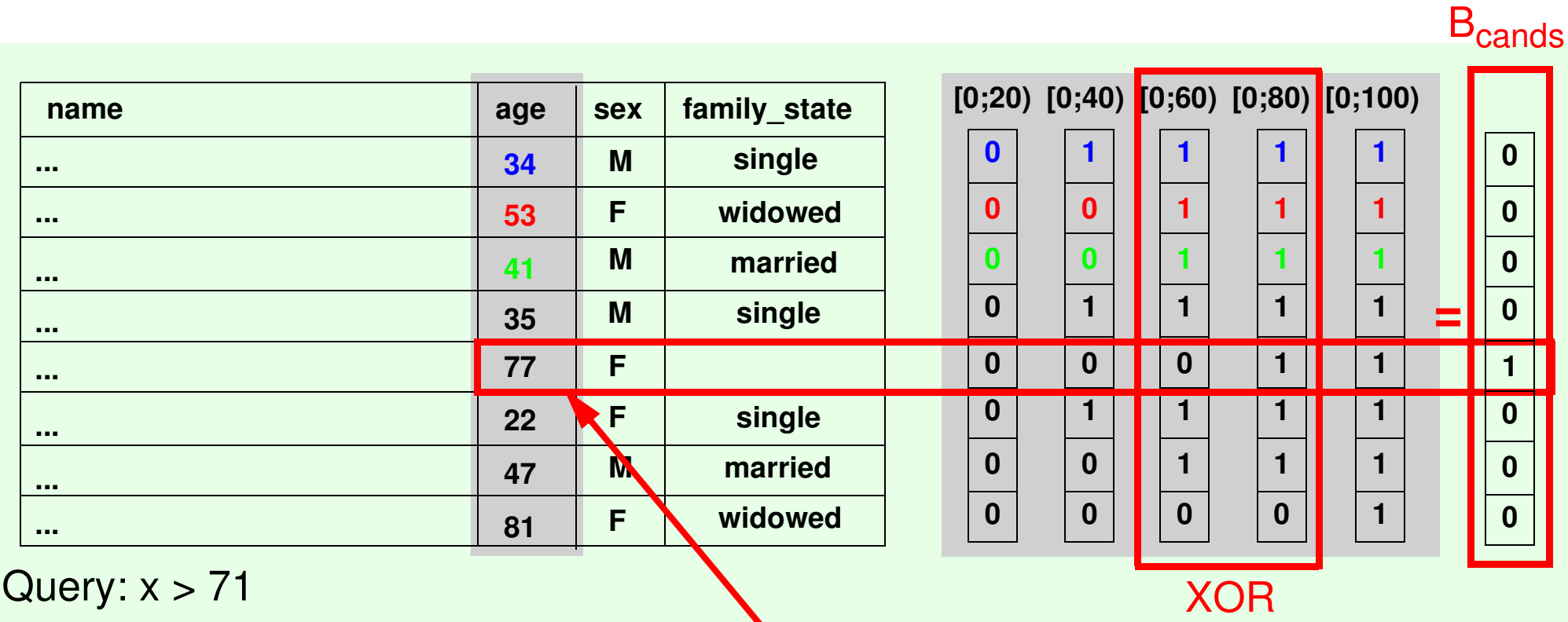


Query: x > 71

$B_{candhits} = \overline{[0;60)} = [60;100)$

$B_{cands} = [0;60)$ XOR $[0;80)$

Binning Example

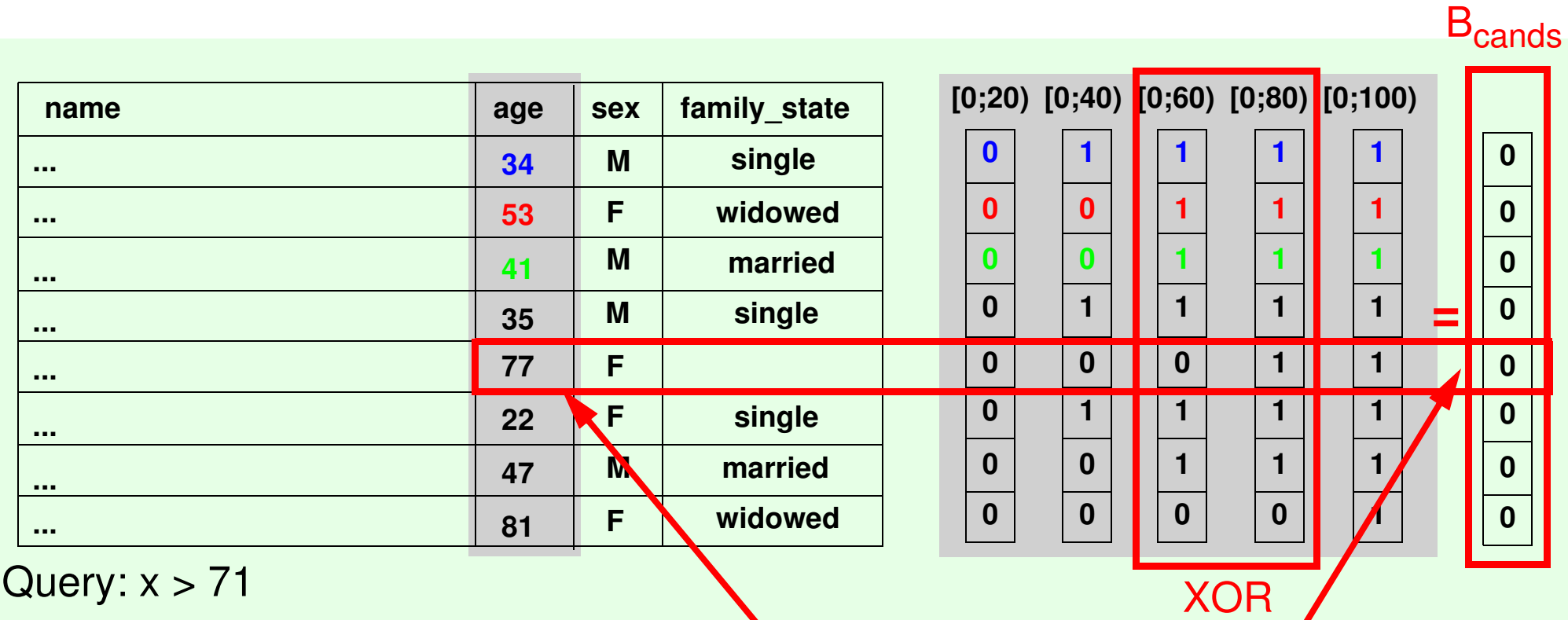


Query: x > 71

$B_{candhits} = \overline{[0;60)} = [60;100)$

$B_{cands} = [0;60)$ XOR $[0;80)$

Binning Example



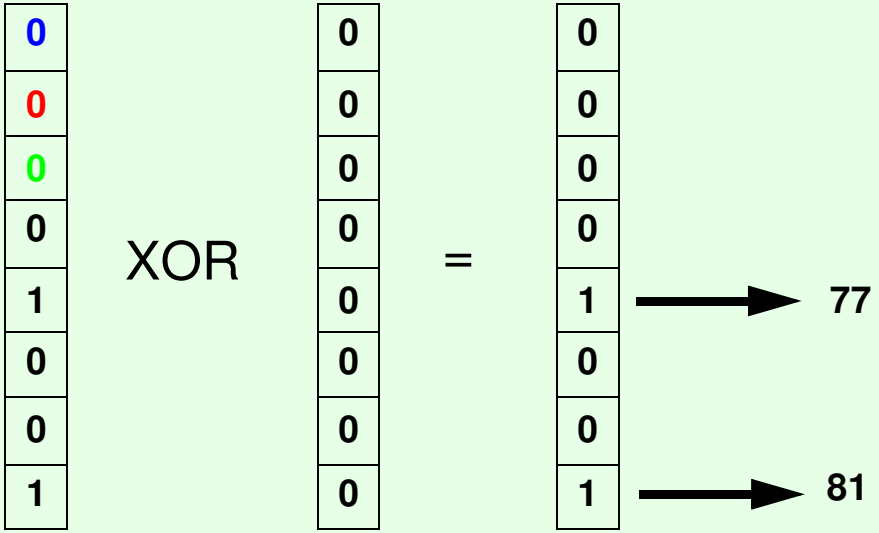
Query: x > 71

$B_{candhits} = \overline{[0;60)} = [60;100)$

$B_{cands} = [0;60)$ XOR $[0;80)$

Binning Example

$B_{hits} = B_{candhits}$ XOR $B_{cands(manually\text{-}checked)}$ =



Compression

- Run Length Encoding allows operations (AND, OR, NOT, XOR) on compressed bitmaps
- Algorithms: Byte Aligned Bitmap Code (BBC), Word Aligned Hybrid Code (WAH), ...
- Studies [WU04] showed, that even with high cardinality attributes the size of a compressed bitmap index has only about half the size of a B-tree.

[WU04] Kesheng Wu and Ekow Otoo and Arie Shoshani: On the Performance of Bitmap Indices for High Cardinality Attributes, 2004

OLAP vs. OLTP

- Uncompressed bitmaps also suitable for OLTP
  - Update/Insert/Delete: O(1)
  - Binning with high cardinality attributes
- Compression disqualifies bitmaps use for OLTP
  - Update to expensive
  - But: Append mode with immutable records is still possible

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Bitmaps in Column-Stores

Bitmap Indexes and Column Stores

ID	Name	Firstname	date-of-birth	sex
31	Waits	Tom	1949-12-07	M
45	Benigni	Roberto	1952-10-27	M
65	Jarmusch	Jim	1953-01-22	M
77	Ryder	Winona	1971-10-29	F
81	Rowlands	Gena	1930-06-19	F
82	Perez	Rosa	1964-09-06	F

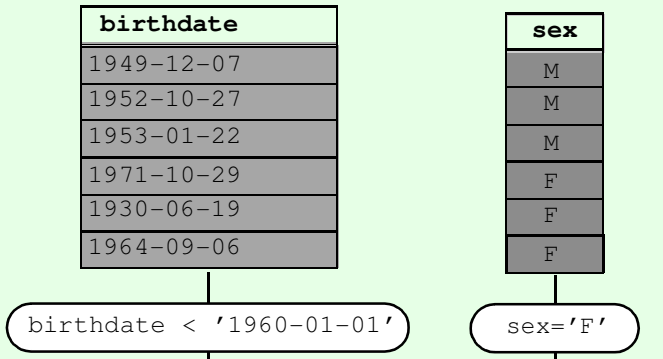
Row-Store

31	Waits	Tom	1949-12-07	M
45	Benigni	Roberto	1952-10-27	M
65	Jarmusch	Jim	1953-01-22	M
77	Ryder	Winona	1971-10-29	F
81	Rowlands	Gena	1930-06-19	F
82	Perez	Rosa	1964-09-06	F

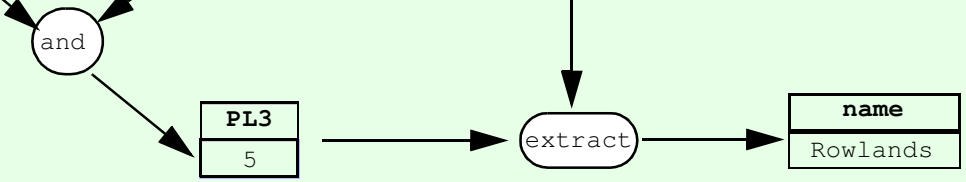
Column-Store

31	Waits	Tom	1949-12-07	M
45	Benigni	Roberto	1952-10-27	M
65	Jarmusch	Jim	1953-01-22	M
77	Ryder	Winona	1971-10-29	F
81	Rowlands	Gena	1930-06-19	F
82	Perez	Rosa	1964-09-06	F

PositionLists




```
select name
  from person
 where birthdate < '1960-01-01'
   and sex='F'
```



PositionLists

- PositionLists store Tuple IDs in ascending order (no duplicates)
- Implementation variants:
  - Dynamic array
  - uncompressed bitmap
  - compressed bitmap
- Best implementation (performance/storage consumption) heavily depends on selectivity

for more information about this, see Daniel Kimmig's talk on wednesday after lunch

[SK03] Schmidt, A, Kimmig, D.: Considerations about implementation variants for position-lists. Accepted paper for the Fifth International Conference on Advances in Databases, Knowledge, and Data Applications, Sevilla, Spain, 2013.

Summary

- Sequential read characteristic and small footprint make bitmap-indexes an interesting alternative to traditional B-Trees.
- For low cardinality attributes bitmap-indexes are superior in space and time requirements
- Bitmap-Indexes are very well suited for multidimensional ad-hoc queries
- With the concpets of
  - binning
  - compression,

  bitmap indexes can also be used for attributes with high cardinality like floating point numbers
- Bitmaps/Compressed bitmaps can also be used in Column-stores for the implementation of position lists.

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Literature

- [Car02] Carlos Carvalho: The Gap between Processor and Memory Speeds, ICCA, 2002
- [CI98] Chee-Yong Chan und Yannis Ioannidis: Bitmap Index Design and Evaluation. Proceedings of the 1998 ACM SIGMOD Conference, 1999
- [CI99] Chee-Yong Chan and Yannis E. Ioannidis. 1999. An efficient bitmap encoding scheme for selection queries. In Proceedings of the 1999 ACM SIGMOD international conference on Management of data (SIGMOD '99), 1999
- [Jac09] Adam Jacobs: The Pathologies of Big Data, acmqueue, 2009
- [Joh99] Theodore Johnson. Performance Measurements of Compressed Bitmap Indices. In Proceedings of the 25th International Conference on Very Large Data Bases,1999
- [Koud00] Nick Koudas: Space efficient bitmap indexing. In Proceedings of the ninth international conference on Information and knowledge management, 2000

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Literature

- [NQ97] Patrick O'Neil and Dallan Quass: Improved query performance with variant indexes. In Proceedings of the 1997 ACM SIGMOD international conference on Management of data, 1997
- [RSW04] D Rotem, K Stockinger, Wu: Efficient binning for bitmap indices on high-cardinality attributes, 2004
- [SK03] Schmidt, A, Kimmig, D.: Considerations about implementation variants for position-lists. Accepted paper for the Fifth International Conference on Advances in Databases, Knowledge, and Data Applications, Sevilla, Spain, 2013
- [Stoc01] Stockinger, K.; , "Design and implementation of bitmap indices for scientific data," International Symposium on Database Engineering and Applications, 2001
- [Stoc02] K. Stockinger: Bitmap Indices for Speeding Up High-Dimensional Data Analysis, 2002

Literature

- [SWS04] Kurt Stockinger, Kesheng Wu, and Arie Shoshani: Evaluation Strategies for Bitmap Indices with Binning, 2004
- [WB98] Ming-Chuan Wu; Buchmann, A.P.; , "Encoded bitmap indexing for data warehouses," 14th International Conference on Data Engineering, 1998
- [WOS01] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. 2001. A performance comparison of bitmap indexes. In Proceedings of the tenth international conference on Information and knowledge management, 2001
- [WOS04] Wu K, Otoo E, Shoshani A: On the Performance of Bitmap Indices for High Cardinality Attributes, 2004
- [WY98] Kun-Lung Wu; Yu, P.S.; , "Range-based bitmap indexing for high cardinality attributes with skew," Computer Software and Applications Conference, 1998
- [YJ00] Sihem Amer-Yahia and Theodore Johnson. 2000. Optimizing Queries on Compressed Bitmaps. In Proceedings of the 26th International Conference on Very Large Data Bases, 2000