On Supporting Containment Queries in Relataional Database Management Systems

Chun Zhang, Jeffrey Naughton
David DeWitt, Qiong Luo
Guy Lohman

ACM SIGMOD 2001

Presented by Dong-Hyuk Im

Table of Contents

- Introduction
- Supporting containment queries
  - Using the inverted lists
  - Using the RDBMS
  - Comparison of two approaches
- Analysis of the comparison results
  - Join algorithms
  - Hardware cache utilization
- Conclusion

Containment Query

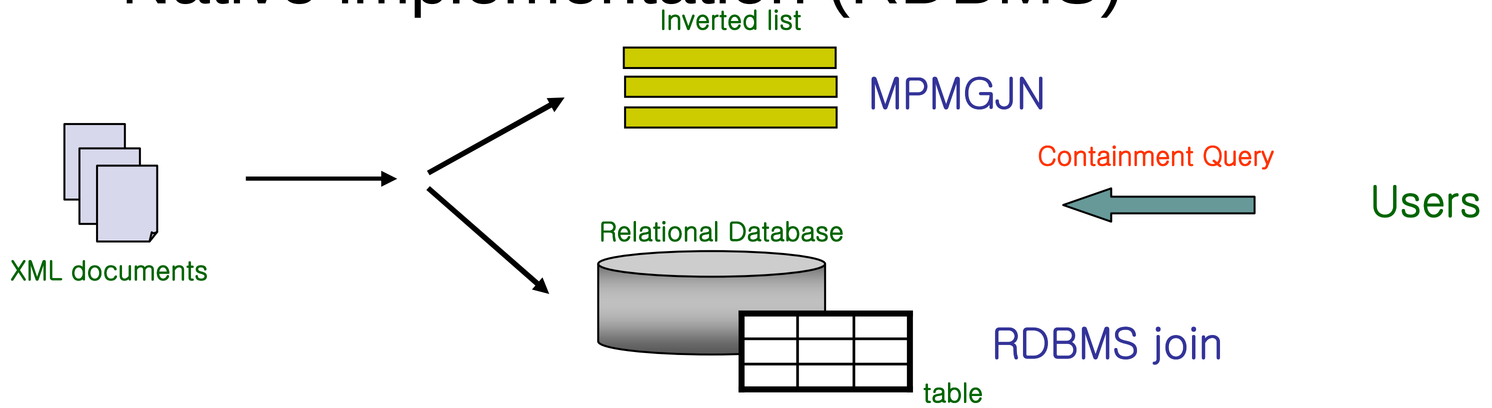
/doc[author='John Smith']//section/title

Indirect containment
(Predecessor-descendant relationship)

Direct containment
(Parent-child relationship)

Motivation

- How should we support containment queries in RDBMS?
- Two type of inverted lists implementation
  - Inverted list engine
  - Native implementation (RDBMS)

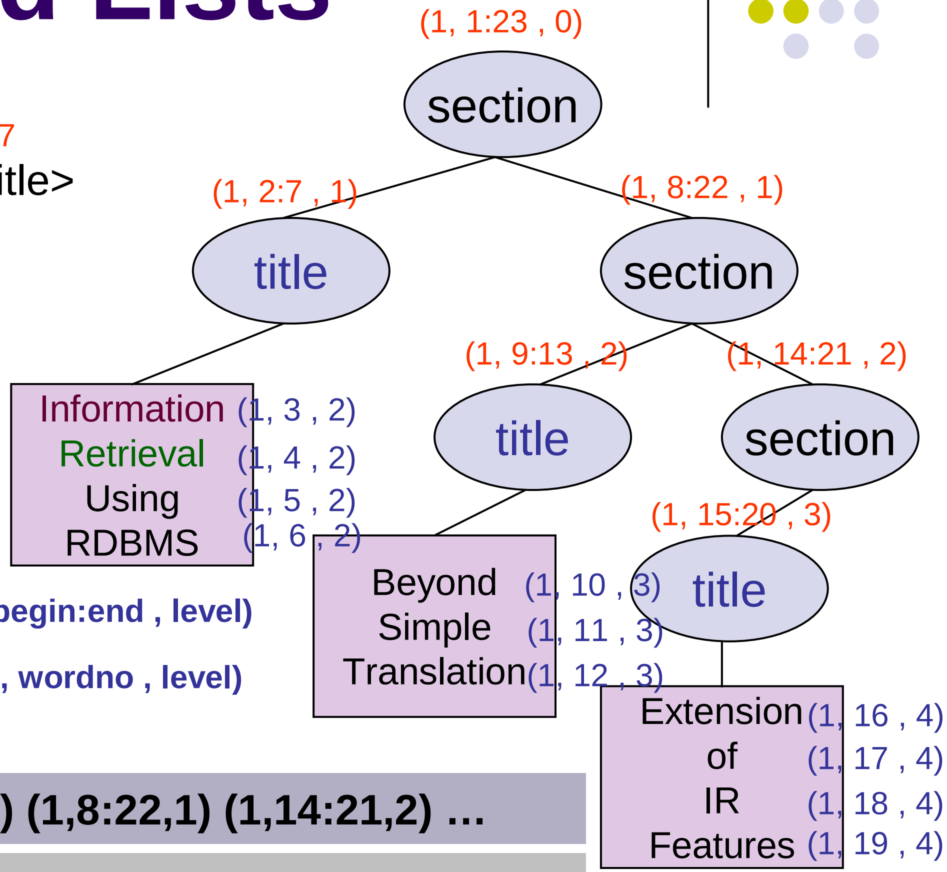


- Compare, find differences and seek reasons

Using the Inverted Lists

```
1
<section>
  2         3           4          5       6       7
  <title> Information Retrieval Using RDBMS </title>
  8
  <section>
    9       10     11      12          13
    <title> Beyond Simple Translation </title>
    14
    <section>
      15      16        17 18 19        20
      <title> Extension of IR Features </title>
    21
    </section>
  22
  </section>
23
</section>
```

A sample XML document



Element : (docno, begin:end , level)

Text word : (docno, wordno , level)

Element index	<section> →	**(1,1:23,0) (1,8:22,1) (1,14:21,2) …**
	<title> →	**(1, 2:7, 1) (1, 9:13, 2) (1, 15:20, 3) …**
Text index	**"information"** →	**(1, 3, 2) … …**
	"retrieval" →	**(1, 4, 2) … …**

Inverted lists

MPMGJN : The Join Method of the Inverted List Engine

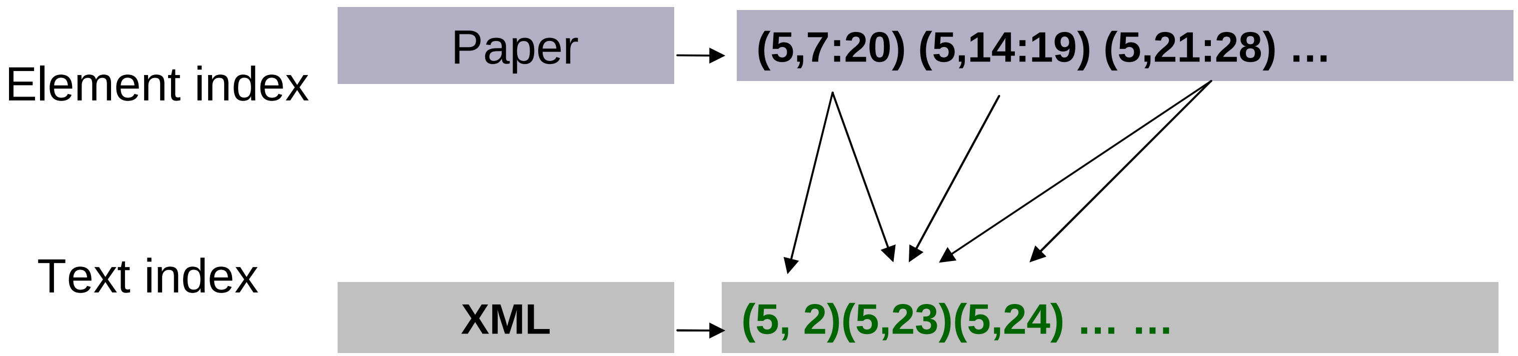
Containment property

ex) (5,7:20) nests (5,10)

Element : (docno, begin:end)

Text word : (docno, wordno)

Paper// ‘XML’



Using an RDBMS

- Inverted lists to relational tables

<section>

Inverted List

(1, 1:23, 0)
(1, 8:22, 1)
(1, 14:21, 2)

ELEMENTS table

term	docno	begin	end	level
section	1	1	23	0
section	1	8	22	1
section	1	14	21	2

```
-- E // 'T'
select  *
from    ELEMENTS e, TEXTS t
where   e.term = 'E'
and     t.term = 'T'
and     e.docno = t.docno
and     e.begin < t.wordno
and     t.wordno < e.end
```

"information"

Inverted List

(1, 3, 2)
….

TEXTS table

term	docno	wordno	level
information	1	3	2

```
-- section // 'information'
select  *
from    ELEMENTS e, TEXTS t
where   e.term = 'section'
and     t.term = 'information'
and     e.docno = t.docno
and     e.begin < t.wordno
and     t.wordno < e.end
```

Experimental Setting

- Data sets
  - Shakespeare, DBLP, Synthetic
- Inverted list engine
  - Written in C++
  - Use BerkeleyDB library
- RDBMS
  - DB2 UDB v7.1 and SQLServer 7.0
  - Index using different combinations of columns
    - The two-column index (term,docno)
    - the cover index (all columns)

The Queries

Queries	term1 frequency	term2 frequency	result rows
QS1	90	277	2
QS2	107,833	277	36
QS3	107,833	3,231	1,543
QS4	107,833	1	1
QD1	654	55	13
QD2	4,188	712	672
QD3	287,513	6,363	6,315
QD4	287,513	3	3
QG1	50	1,000	809
QG2	134,900	55,142	1,470
QG3	701,000	165,424	21,936
QG4	50	82,712	12
QG5	701,000	17	4

S : Shakespeare

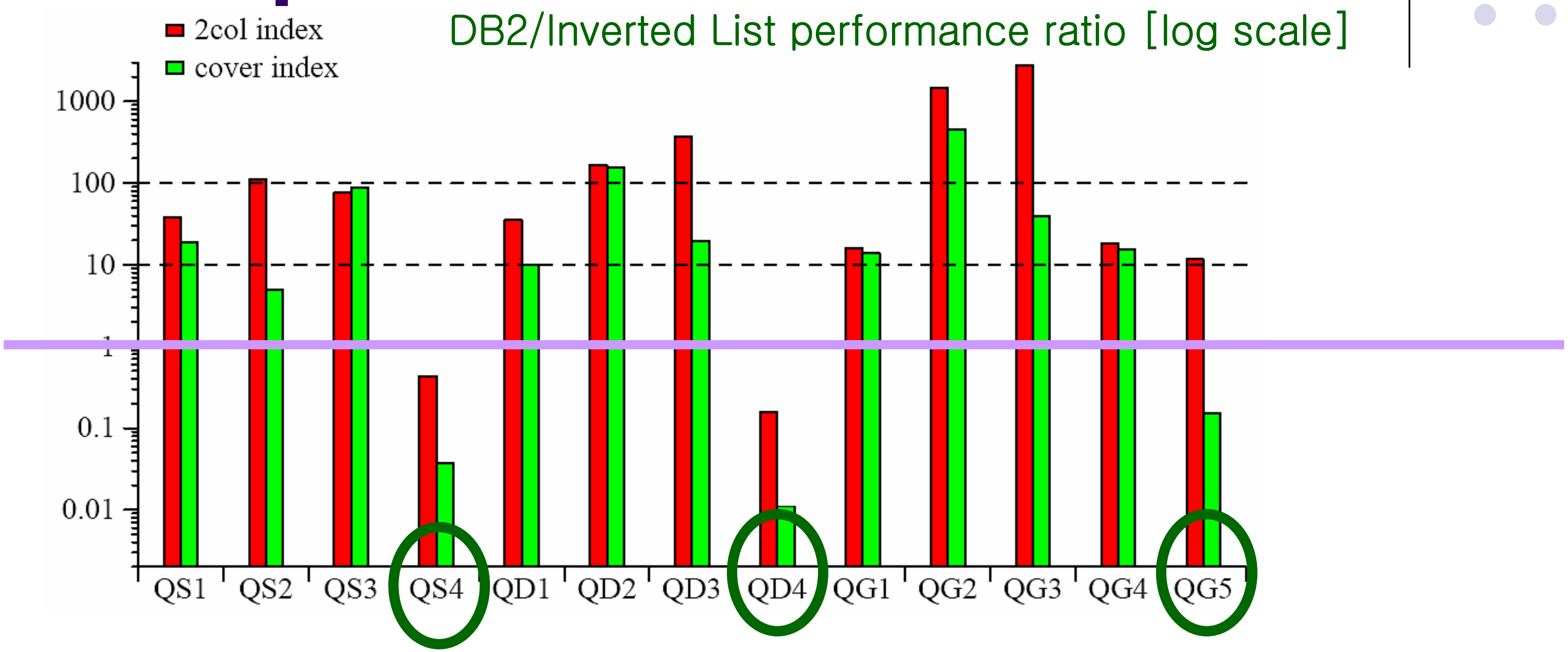
D : DBLP

G : Generated data

- “term1 contains term2” (term1 : element , term2 : text word)

  Ex) Paper// ‘XML’

Comparison Results

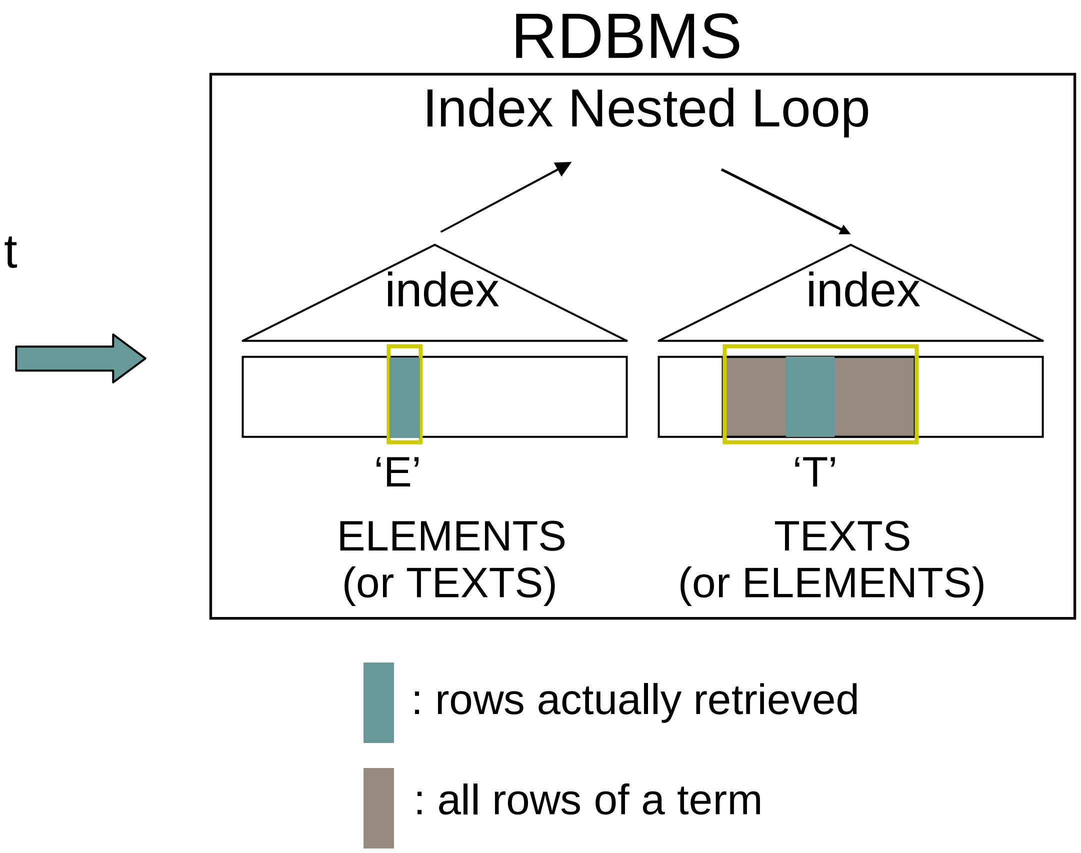


- DB2 outperforms the inverted list engine for some of the queries
- DB2 performs worse than the inverted list engine for most the queries
- The cover index is better than using the 2col index

Why Does the RDBMS Sometimes Perform Better?

- Index nested-loop join is used

```
-- E // 'T'
select   *
from     ELEMENTS e, TEXTS t
where    e.term = 'E'
and      t.term = 'T'
and      e.docno = t.docno
and      e.begin < t.wordno
and      t.wordno < e.end
```



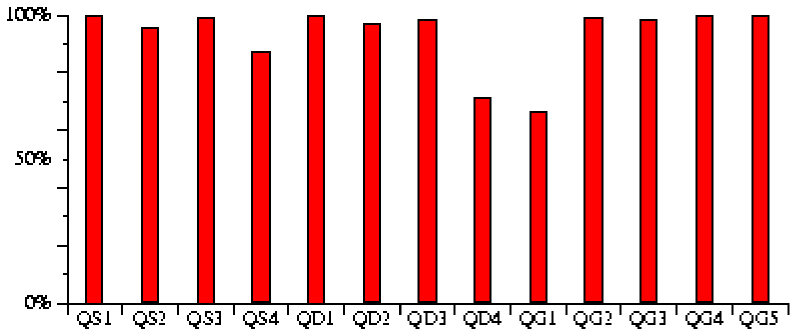
Why Does the RDBMS Usually Perform Worse?

- Possible answers
  - Bind-out cost — NO
  - Bad query plan — NO
  - I/O overhead — NO

Bindout is a small fraction of total execution time

Optimizer's pick matched a good plan

Each query is CPU-bound and operates out of buffer pool with little I/O overhead



Percentage of CPU cost

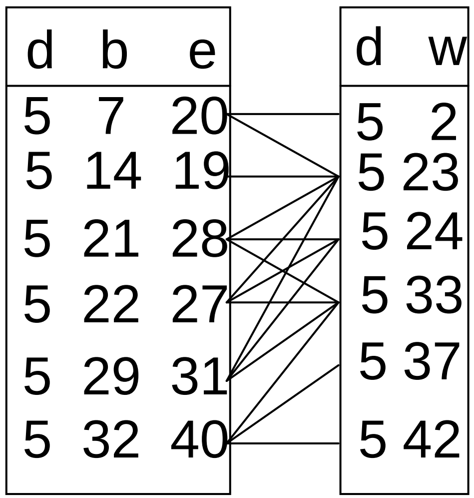
Then… what on earth makes the RDBMS perform worse???

MPMGJN vs. Standard Merge Join

Three merging predicates
d1 = d2, b <= w, w <= e

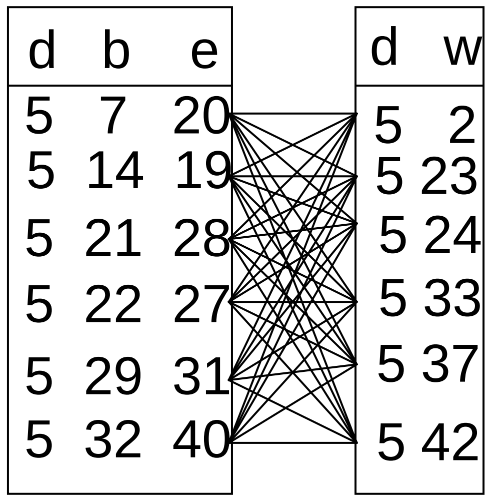
One merging predicate: d1 = d2,
Additional filtering predicates:
b <= w, w <= e

Inverted List Engine



MPMGJN

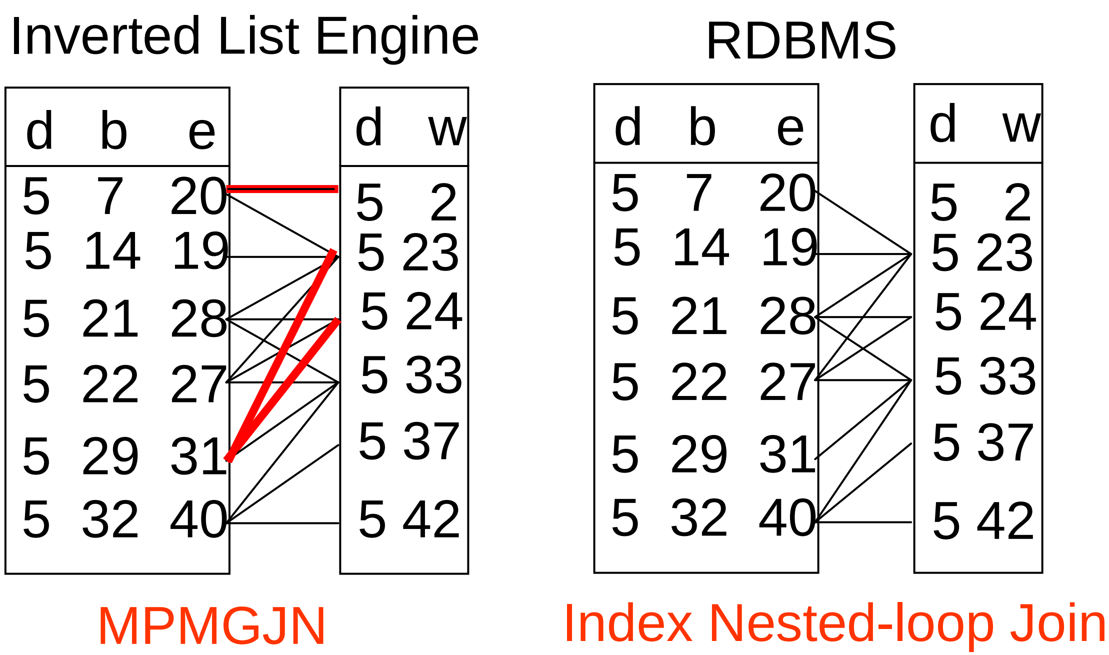
RDBMS



Standard Merge Join

- MPMGJN : avoid some row comparisons

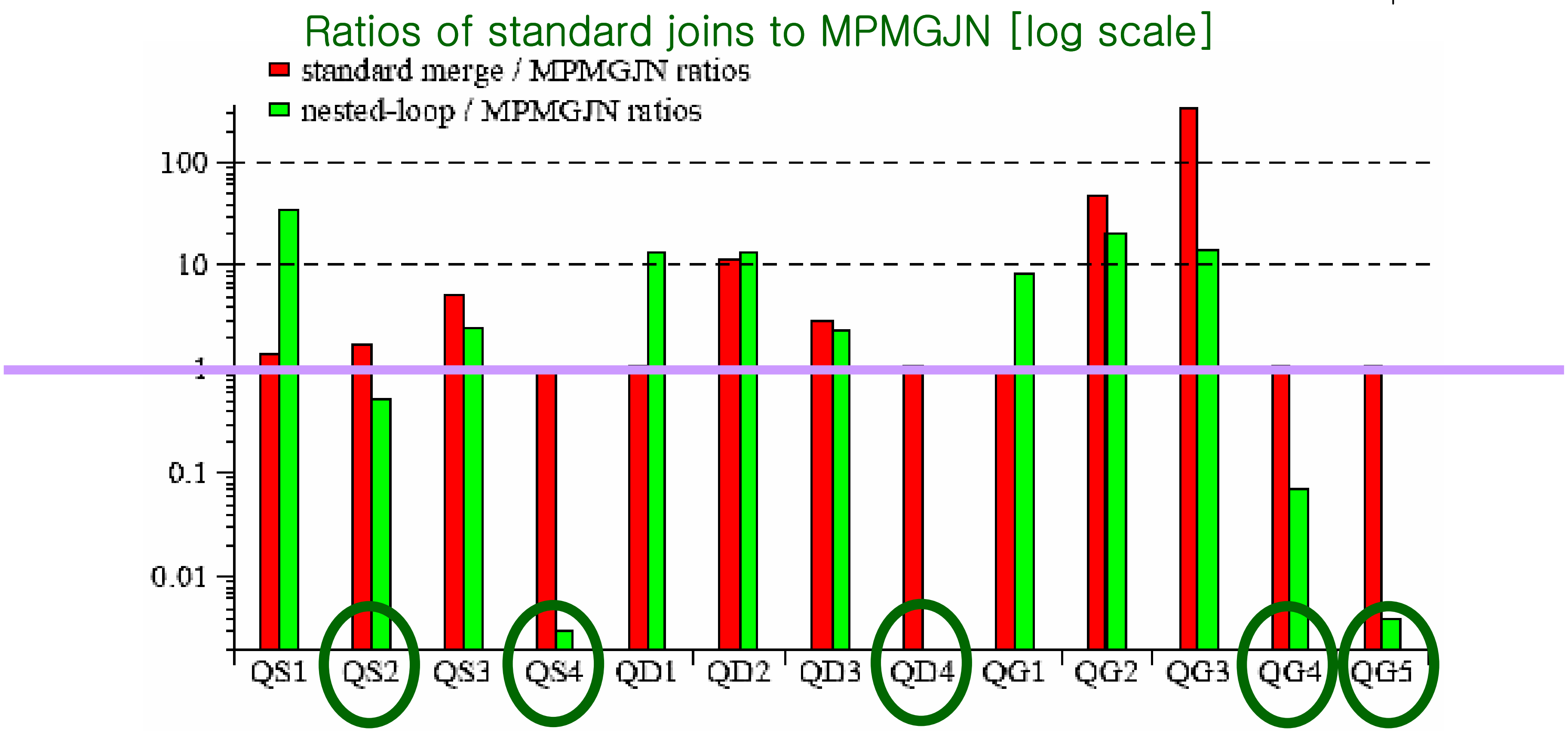
MPMGJN vs. INL Join



- **INL** join does fewer comparisons than **MPMGJN**

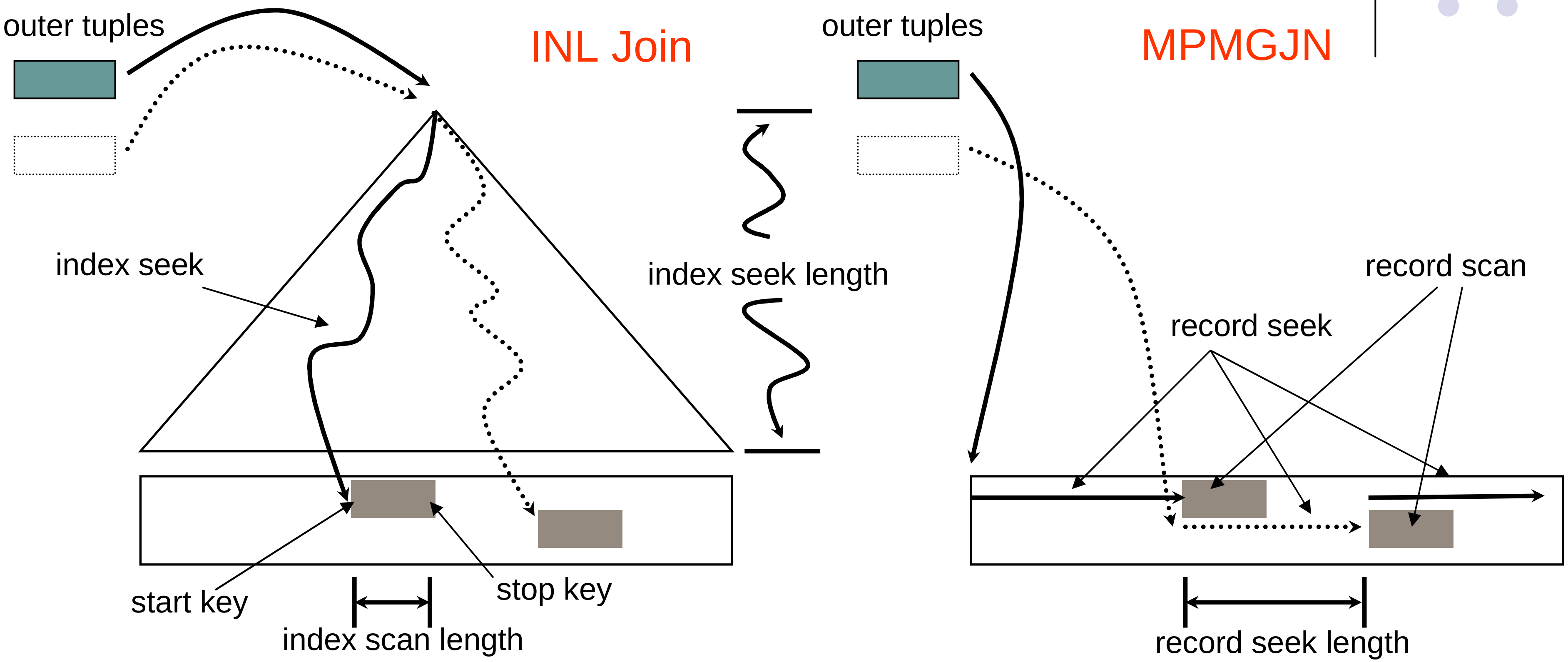
INL join always performs best?

Comparison of Join Algorithm



- MPMGJN performs at least as well as the standard merge join and better than the INL join for most queries
- INL join performs better than MPMGJN for QS2,QS4,QD4,Q4 and QG5

MPMGJN vs. INL Join



Seeking index is repeated

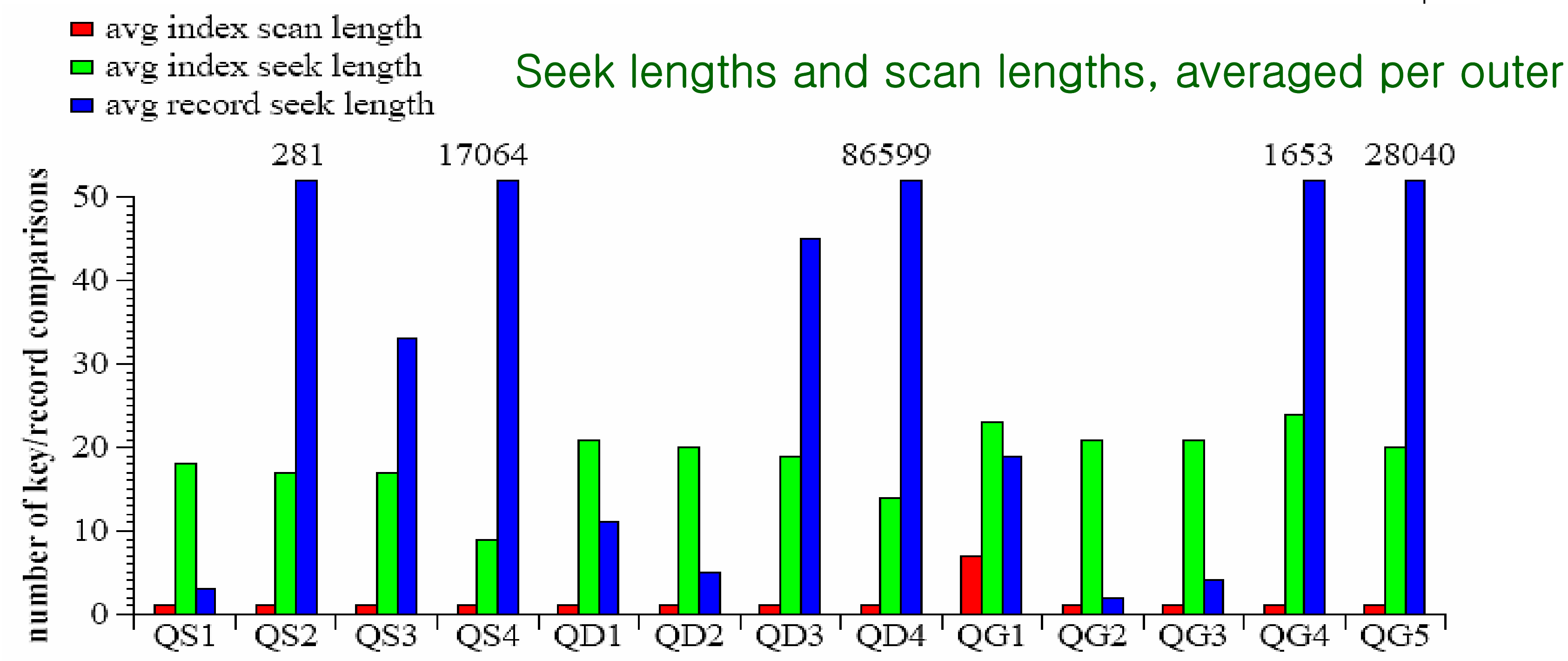
Start key predicate:

term = value and docno=outer.docno and wordno > outer.begin

Stop key predicate:

term = value and docno= outer.docno and wordno < outer.end

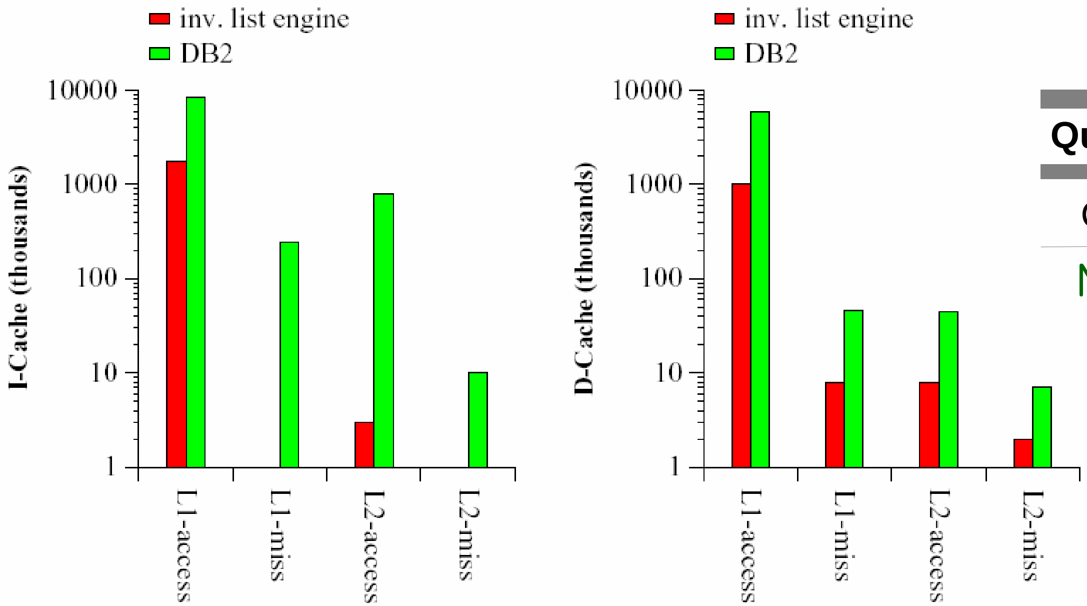
MPMGJN vs. INL Join



- **QS1,QD1,GD2,QG1,QG2,QG3 : record seek lengths are shorter** ⇨ **MPMGJN > INL Join**
- **QS2,QS4,QD4,QG4,QG5 : record seek lengths are much longer** ⇨ **MPMGJN < INL Join**
- **QS3,QD3 : record seek lengths are longer** ⇨ **MPMGJN > INL Join**

Hardware Cache Utilization

Query QG1 [log scale]



Queries	MPMGJN	Standard merge join
QG1	1,000	1,000

Number of row pairs of comparison

DB2 : use standard merge join in QG1

- The two algorithms should perform the same, however the inverted list engine performs faster than DB2
- DB2 has better data cache miss ratio and much worse instruction cache miss ratio
- Cache is a distinct factor that affects performance

Conclusion

- Two important factor to the performance difference
  - The join algorithm
  - The hardware cache utilization
- An RDBMS will be able to natively support containment queries efficiently!.....
  by combining MPMGJN and better cache utilization