SSDBM, San Diego, USA, 2015

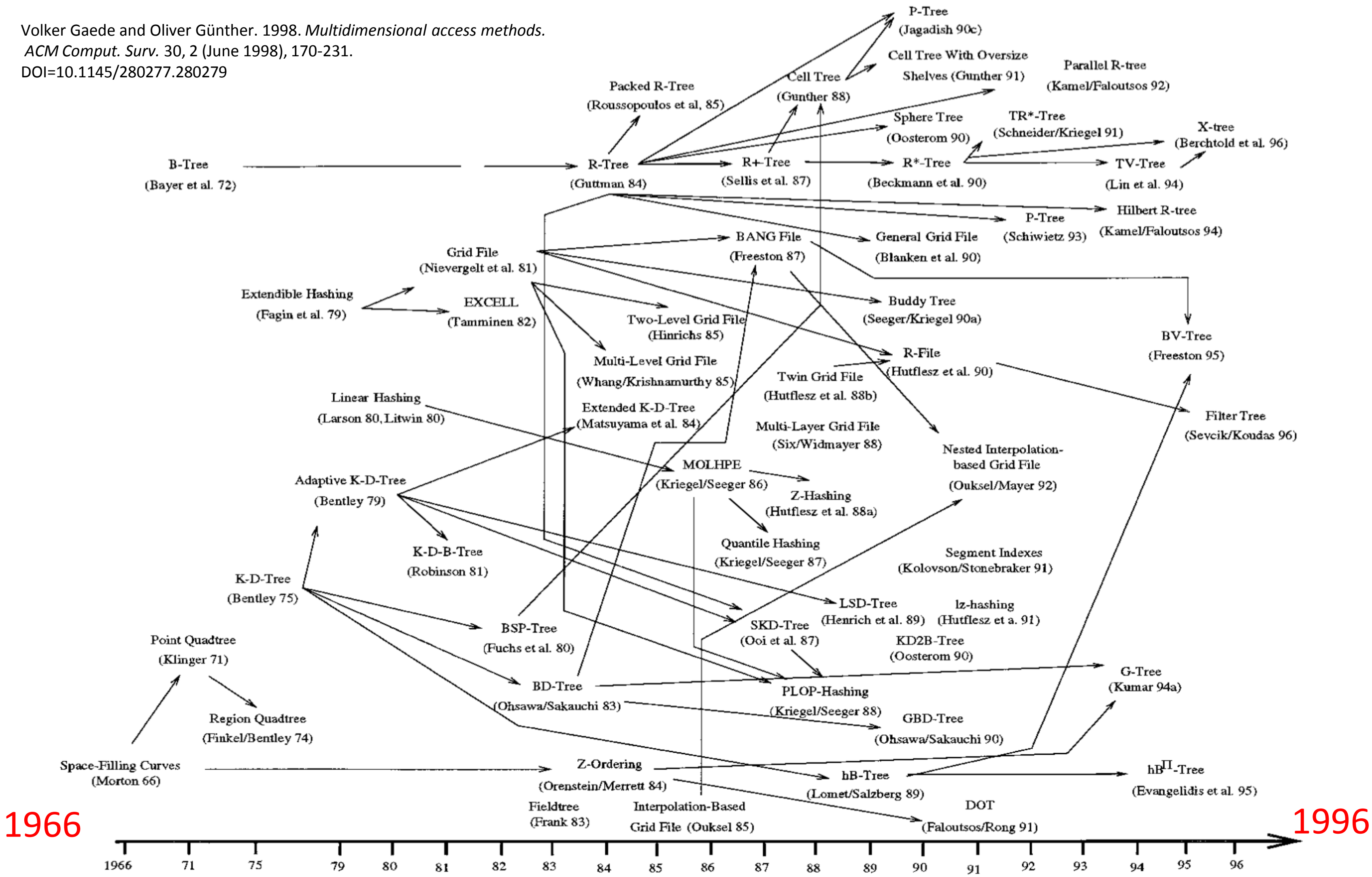
Transparent inclusion, validation, and utilization of main memory domain indexes

Thanh Truong C., Tore Risch
Uppsala University
Sweden

“Many scientific applications involving, e.g., data mining, temporal queries, and spatial analyses, ***require customized indexing to improve performance***”

How many index structures are there ?

Volker Gaede and Oliver Günther. 1998. *Multidimensional access methods.* *ACM Comput. Surv.* 30, 2 (June 1998), 170-231.
DOI=10.1145/280277.280279



How many index structures are used in DBMSs ?

Index structure	ORACLE	MySQL	Microsoft SQL Server
Btree	Y	Y	Y
Hash	Y	Y	Y
R-tree	Y	Y	-
Trie	-	Y	-
Bit-map	Y	-	-

Notes

- Versions
  - Oracle 12c Release 1
  - SQL Server 2014
  - MySQL 5.6
- In MySQL, some storage engines permit some index types, but not all.
- The table does not count “Function based index”.
- In SQL Server, hash index is only available for in-memory tables.

In-memory databases

Hao Zhang, Gang Chen, Kian-Lee Tan, Meihui Zhang
In-Memory Big Data Management and Processing: A Survey,
Knowledge and Data Engineering, IEEE Transactions on
volumn 27, pages 1920 - 1948
DOI: 10.1109/TKDE.2015.2427795

	Systems	Data Model	Workloads	Indexes
Relational Databases	**H-Store**	relational (row)	OLTP	hashing, B^+-tree, binary tree
	Hekaton	relational (row)	OLTP	latch-free hashing, Bw-tree
	HyPer/ ScyPer	relational	OLTP, OLAP	hashing, balanced search tree, ART
	SAP HANA	relational, graph, text	OLTP, OLAP	timeline index, CSB^+-tree, inverted index
NoSQL Databases	**MemepiC**	key-value	object operations, analytics	hashing, skip-list
	MongoDB	document (bson)	object operations, analytics	B-tree
	RAMCloud	key-value	object operations	hashing
	Redis	key-value	object operations	hashing
Graph Databases	**Bitsy**	graph	OLTP	N/A
	Trinity	graph	graph operations	N/A
Cache Systems	**Memcached**	key-value	object operations	hashing
	MemC3	key-value	object operations	hashing
	TxCache	key-value	OLTP	hashing
Big Data Analytics Systems	**M3R**	key-value	analytics	N/A
	Piccolo	key-value	analytics	hashing
	Spark/ RDD	RDD	analytics	N/A
Real-time Processing Systems	**Spark Streaming**	RDD	streaming	N/A
	Yahoo! S4	Event	streaming	hashing

How many index structures are used in In-memory databases ?

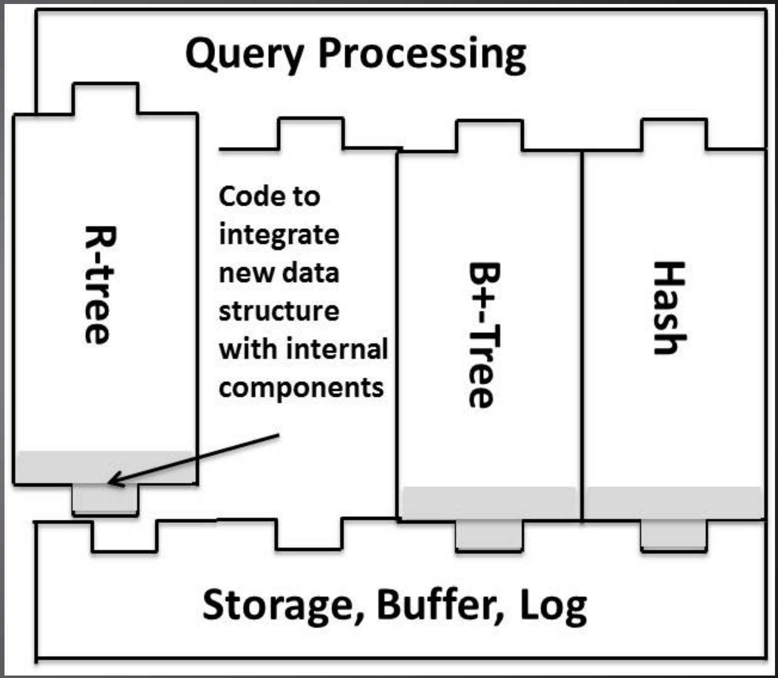
Hao Zhang, Gang Chen, Kian-Lee Tan, Meihui Zhang
In-Memory Big Data Management and Processing: A Survey,
Knowledge and Data Engineering, IEEE Transactions on volumn 27, pages 1920 - 1948
DOI: 10.1109/TKDE.2015.2427795

Why ?

Because it is very challenging

Here are some challenges C1,..,C5

- C1. Understanding the DB kernel
- C2. Re-implementing the datastructure
- C3. Integrating with other DB internal components
- C4. Extending query processor
- C5. Validating the index's functionalities



Only database (kernel) expert can do it!

Solution?

Some extensible indexing frameworks

- GiST

J Hellerstein. M.,J.F.Naughton, and A. Pfeffer: *Generalized search trees for database systems*, Proc. VLDB Conf., pp 562–573, 1995.

- Extensible Indexing – Orcale 8i

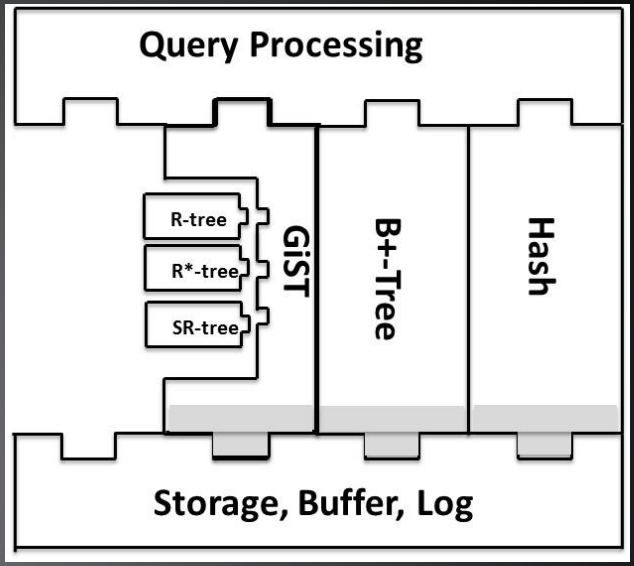
J. Srinivasan, R. Murthy, S. Sundara, N. Agarwal, and S. DeFazio: *Extensible indexing: a framework for integrating domain-specific indexing schemes into oracle8i*. Proc. ICDE Conf., pp 91–100, 2000.

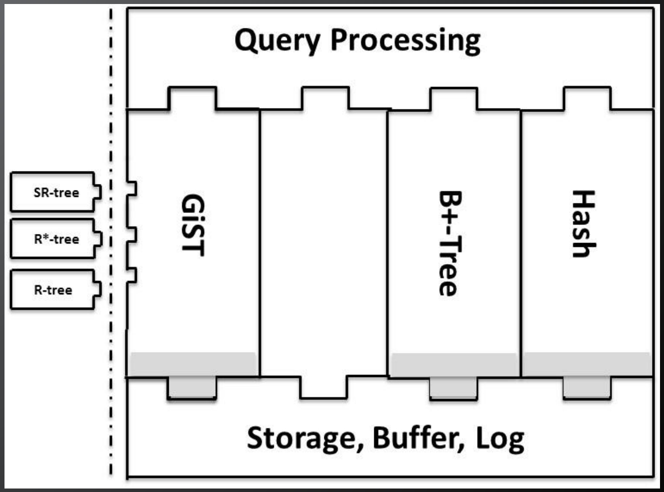
- SP-GisT

W. G. Aref and I. F. Ilyas: *An extensible index for spatial databases*, Proc. SSDBM, pp 49–58, 2005.

Reviews

- These frameworks specifies coding conventions and primitives.
- Solved C1 - Understanding DB kernel -
- Solved C3 – Integrating with other kernel components





The remaining unsolved challenges

- C2 - Re-implementing the index implentation

  It is not OK if the index implementation

  - has ownership.
  - Is available in binary.
  - or being very complex to re-implement, i.e; Judy-tries

  ????

- C4 - Extending query processor

  ????

- C5 - Validating the index's functionalities

  ????

Our motto

~~Only database (kernel) expert can do it!~~

"It should not be necessary to be a database kernel expert to introduce a new domain index"

Our solution

- The paper title:

“Transparent inclusion, utilization, and validation of main memory domain indexes”

- The paper itself
  - Transparent inclusion – to solve C1, C2, C3
    - no index implementation code changed .
  - Transparent utilization – to solve C4
    - automatically transforms queries to utilize the new added index.
  - Transparent validation – to solve C5
    - Automatically generates and executes queries to test the new added index

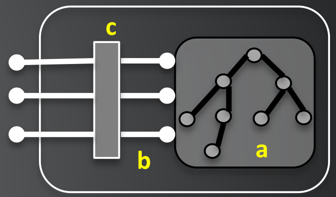
- The result :

The generalized extensible indexing framework: Main-memory eXternal Index Manager (Mexima).

- Website: http://www.it.uu.se/research/group/udbl/mexima/

How to introduce a new index ?

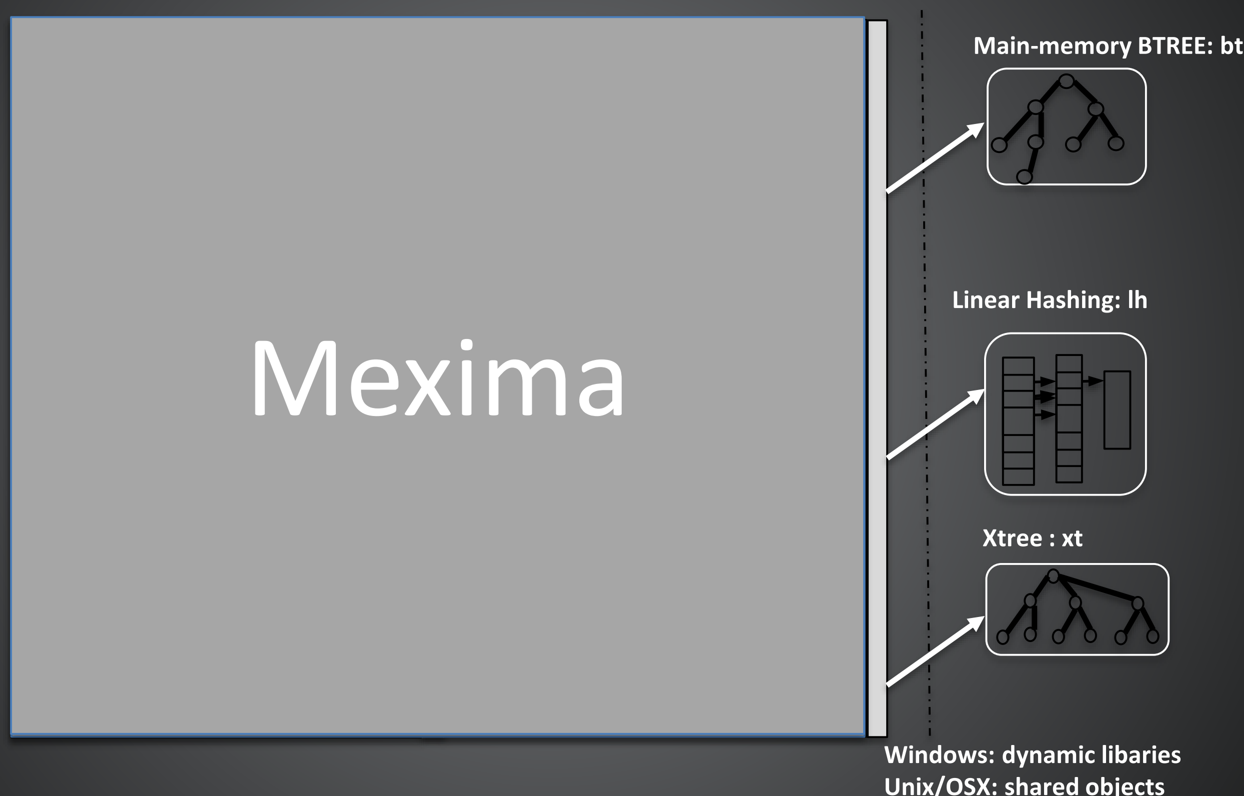
- Grab the *index implementation (a)*
- Study the public *index API (b)*
- Write the *index driver (c)*



(glue code) that interfaces Mexima and the index API

➔ Compiled as dynamic library called as *index extension*

At the end of the day




```
/* Load main-memory BTREE index*/
load_extension(”bt”);
```

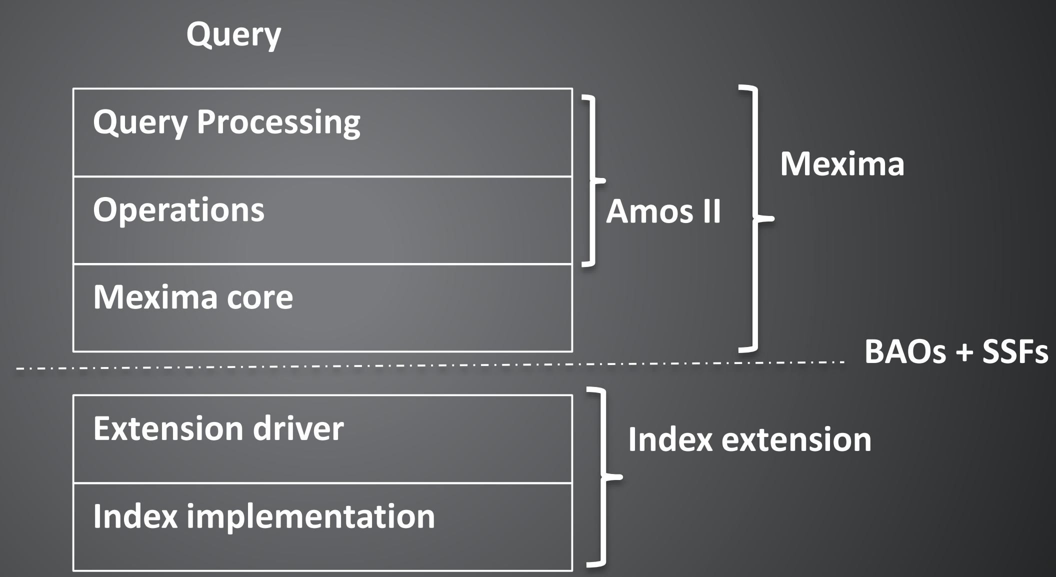
At the end of the day (cont.)

```
/* Create a table to store salaries of people given social security numbers*/
create function salary(Number ssn)->Number sl as stored;

/*create BTREE on sl*/
create_index(”salary”, ”sl”, ”BTREE”, ”);

/*Add data*/
set salary(8301318971) = 2000;
set salary(8501332978) = 3000;
. . .
set salary(8001335978) = 4000;
/*Query*/
SELECT ssn, sl
FROM  Number ssn, Number sl
WHERE salary(ssn) = sl AND sl >= 3000;
```

Mexima



Mexima interface = BAOs + SSFs

The index driver code contains

- Basic access operators (BAOs)
  - *create()*, *drop()*, *put()*, *delete()*, *get()*,
  - and *map()* that scans the index by applying a specified mapper function on each index entry.
  - implemented as C functions

(***details in the paper)

The index driver code contains

- Special search functions (SSFs)
  - Examples:
    - interval search on B-trees: *bt_select_range()*
    - and proximity search on X-trees/R-trees: *xt_proximity_search()*
    - and KNN search on X-trees/R-trees: *xt_knn_search()*
  - Implemented as foreign functions (UDFs)

(*** details in the paper)

But it is not enough ...

- How new index is utilized in query?
  - Option 1

    End-user can manually call a SSF in query by reformulating the query

  - Option 2 

    End-user can express query naturally, but the query optimizer should be able to utilize the index.

The query processor should transparently transform the query to SSF if possible to utilize the index → **SSF translation rules**.

SSF translation rules

- An SSF translation rule describes how query fragments are translated to a new format to expose SSFs.
- Examples

#	Index type	priority	Index sensitive function	Relation operators	SSF
1	*B-tree*	*1*	*Nil*	>=, <=	*btree_select_range*
2	*X-tree*	*1*	*distance*	<=	*xt_proximity_search*
3	*X-tree*	*2*	*Knn*	*nil*	*xt_knn_search*

Example - Table

- Table *images(id, hist)*
  - *Id,* image's identifier
  - *hist,* histogram as image's feature vector

Btree index

Xtree index

Example - Query 1

”Query 1 finds images q whose identifiers are between 30 and 100”

Input query

Q1(q):-

images(q, _) AND
q >= 30 AND
q <= 100

MAP

→

Intermediate query
TQ1(q):-
(q,_) in
btree_select_range(#’images’, 0, 30,100)

SSF Btree range search

With Mexima, it is done by the following SSF translation rule

#	Index type	priority	Index sensitive function	Relation operators	SSF	pf
1	*B-tree*	*1*	*Nil*	*>=, <=*	*btree_select_range*	*F*
2	*X-tree*	*1*	*distance*	*<=*	*xt_proximity_search*	*T*
3	*X-tree*	*2*	*Knn*	*nil*	*xt_knn_search*	*F*

Example - Query 1

"Query 1 f... between 30 and 100"

Index type	Index sensitive function	Relation operators	SSF
B-tree	*Nil*	>=, <=	*btree_select_range*

Input query

Q1(q):-

images(q, _) AND
q >= 30 AND
q <= 100

Intermediate query

TQ1(q):-
(q,_) in
btree_select_range(#'images', 0, 30,100)

SSF Btree range search

Form (i):

P(…iv,..) AND (iv r1 expression) AND
(iv r2 expression) AND
. . .
(iv rn expression)

Here,

- ***iv*** is a variable bound to an indexed column of table *P(…)*. We say ***iv*** is an *indexed variable*.
- r_i are comparison operators in the set ***relop***, $r_i \in$ *relop*, where ***relop*** $=\{=, <, >, >=, <=\}$.

...e following SSF translation rule

...	Relation operators	SSF	...
	>=, <=	*btree_select_range*	*F*
	<=	*xt_proximity_search*	
	nil	*xt_knn_search*	

Example - Query 2

” For a given image *x* find the images *q* whose feature vectors are closer than epsilon (*eps = 0.11*).”

Input query

Q2(x, q):-

images(x, hist_x) AND
images(q, hist_q) AND
distance (hist_x, hist_q) <= 0.11

MAP

Intermediate query

TQ2(x, q):-
image(x, hist_x) AND
(q,hist_q) in
xtree_proximity_search(#’images’,1, hist_x, 0.11) AND
distance (hist_x, hist_q) <= 0.11

SSF X-tree proximity search

With Mexima, it is done by the following SSF translation rule

#	Index type	priority	Index sensitive function	Relation operators	SSF	pf
1	*B-tree*	*1*	*Nil*	*>=, <=*	*btree_select_range*	*F*
2	*X-tree*	*1*	*distance*	*<=*	*xt_proximity_search*	*T*
3	*X-tree*	*2*	*Knn*	*nil*	*xt_knn_search*	*F*

Example - Query 2

Index type	Index sensitive function	Relation operators	SSF
X-tree	*distance*	*<=*	*xt_proximity_search*

" For a given image *x* find the images *q* whose feature vectors are closer than epsilon (*eps = 0.11*)."

Input query

Q2(x, q):-

images(x, hist_x) AND
images(q, hist_q) AND
distance (hist_x, hist_q) <= 0.11

Intermediate query

TQ2(x, q):-

image(x, hist_x) AND
(q,hist_q) in
xtree_proximity_search(#'images',1, hist_x, 0.11) AND
distance (hist_x, hist_q) <= 0.11

Form (ii):

P(…iv,..) AND isf(…,iv, …) r_1 *expression AND*
isf(…,iv, …) r_2 *expression AND*
. . .
isf(…,iv, …) r_n *expression*

Here, ***iv*** is an indexed variable occurring in parameter position of an index sensitive function ***isf()***.

Example - Query 3

” Find the *k = 10* closest images compared to a given image x”

Input query

Q3(x, hist_x):-

images(x, hist_x) AND
images(q, hist_q) AND
(q, hist_q) in knn(hist_x, 10, #’images’)

MAP

Intermediate query

TQ3(x, hist_x):-
image(x, hist_x) AND
(q,hist_q) in (q,_) in xt_knn_search (#‘images’, 1, hist_x, 10)

SSF X-tree KNN search

With Mexima, it is done by the following SSF translation rule

#	Index type	priority	Index sensitive function	Relation operators	SSF	pf
1	*B-tree*	*1*	*Nil*	*>=, <=*	*btree_select_range*	*F*
2	*X-tree*	*1*	*distance*	*<=*	*xt_proximity_search*	*T*
3	*X-tree*	*2*	*Knn*	*nil*	*xt_knn_search*	*F*

Index type	Index sensitive function	Relation operators	SSF
X-tree	*Knn*	*nil*	*xt_knn_search*

Input query

Q3(x, hist_x):-

images(x, hist_x) AND
images(q, hist_q) AND
(q, hist_q) in knn(hist_x, 10, #'images')

Intermediate query

TQ3(x, hist_x):-
image(x, hist_x) AND
(q,hist_q) in (q,_) in xt_knn_search (#'images', 1, hist_x, 10)

Form (iii):
P(…,iv,…) AND (..,iv,..) in isf(…..,P,..)

Reviews of query fragment forms

Form (i):
P(…iv,..) AND (iv r_1 expression) AND
(iv r_2 expression) AND
. . .
(iv r_n expression)

Form (ii):
P(…iv,..) AND isf(…,iv, …) r_1 expression AND
isf(…,iv, …) r_2 expression AND
. . .
isf(…,iv, …) r_n expression

Form (iii): P(…,iv,…) AND (..,iv,..) in isf(…..,P,..)

Form (iv): P(…iv,..) AND F(iv) relop expression

Form (v): P(…,iv,…) AND F(isf(…,iv, …)) relop expression

Mexima

Transparently transformation to utilize indexing

isf(…) relop expression

isf(…) LIKE expression

Oracle

Advisor tools to suggest on reformulating the query to utilize indexing

- D. Benoit, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin: Automatic SQL tuning in Oracle 10g, Proc. VLDB Conf, pp 1098-1109, 2004.
- Oracle Inc: *Query Optimization in Oracle Database 10g Release 2*. http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-general-query-optimization-10gr-130948.pdf , 2005

** T. Truong, T. Risch: *Scalable Numerical Queries by Algebraic Inequality Transformations*, Proc. Database Systems for Advanced Applications (DASFAA), pp 95-109, 2014

Our solution

- The paper title:

“Transparent inclusion, utilization, and validation of main memory domain indexes”

- The paper itself
  - ❖ Transparent inclusion
    - o no index implementation code changed .
  - ❖ Transparent utilization
    - o automatically transforms queries to utilize the new added index.
  - ❖ Transparent validation
    - o Automatically generates and executes queries to test the new added index

What to test ?

- BAOs: correctness of BAOs
- SSFs
  - Correctness of SSFs
  - Correctness of SSF translation rules

BAO tester

- Automatically tests correctness of *put()*, *get()*, *delete()*, *map()*, and *drop()*.

- Index key generator as queries

#	Idxtype	Index key type	Index KeyGenerators
1	*B-tree*	*Number*	*select uniform_int(1000,0,10000)*
2	*X-tree*	*Vector-Number*	*select uniform_vec_real(1000,5,0,1)*
3	*X-tree*	*Vector-Number*	*select CSV_file_rows("colorhistogram.csv")*

BAO tester (cont.)

- Populate generated data into
  - Table *I_Table(k, v)*, having index to test at column k
  - Table *R_Table(k,v)*, having Hash index at k
- Execute BAO tester algorithms (***)
- Validate *I_Table* against *R_Table*

*** details in the paper

SSF tester - Ideas

- Create sample tables with and without the index
- Auto-generate validation queries
- Recall, SSF translation rules transform these queries

→ Same value returned if there is no index, or no matching SSF translation rules

SSF tester – Ideas (cont.)

- SSF parameter generators

#	Index type	SSF name	SSF parameter generator	SSF Parameter types
1	*B-tree*	*btree_select_range*	*select l, u from Number l, Number u where l in uniform_int(100, 0,10000) and u in uniform_int(100,0,10000)*	*(Number, Number)*
2	*B-tree*	*btree_select_open*	*select u from Number u where u in uniform_int(100, 0,10000)*	*(Number)*
3	*X-tree*	*xtree-proximity-search.*	*select x, d from Vector of Number x, Number d where x in uniform_vec_real(100,5,0,1) and d in uniform_real(100,0, 1.4)*	*(Vector of Number, Number)*
4	*X-tree*	*xtree_knn-search*	*select x, k from Vector of Number x, Number k where x in uniform_vec_real(100,5,0,1) and k in uniform_int(0,5)*	*(Vector of Number, Number)*

It is getting more complicated!

SSF tester – Ideas (cont.)

- Join three index property tables
  - SSF translation rule table

#	Index type	priority	Index sensitive function	Relation operators	SSF	pf
1	*B-tree*	*1*	*Nil*	>=, <=	*btree_select_range*	*F*
2	*X-tree*	*1*	*distance*	<=	*xt_proximity_search*	*T*
3	*X-tree*	*2*	*Knn*	*nil*	*xt_knn_search*	*F*

  - Index key generator table

#	Idxtype	Index key type	Index KeyGenerators
1	*B-tree*	*Number*	*select uniform_int(1000,0,10000)*
2	*X-tree*	*Vector-Number*	*select uniform_vec_real(1000,5,0,1)*
3	*X-tree*	*Vector-Number*	*select CSV_file_rows("colorhistogram.csv")*

  - SSF parameter generator table

#	Index type	SSF name	SSF parameter generator	SSF Parameter types
1	*B-tree*	*btree_select_range*	*select l, u* *from Number l,* *Number u* *where l in uniform_int(100, 0,10000) and u in uniform_int(100,0,10000)*	*(Number, Number)*
		...		

SSF tester – Validation queries

- Validation query matching Form (I)

Formula	Descriptions
select iv, v *from IT iv, Number v,* *T1 p1, T2 p2,.., Tm pm* *where I_Table(iv, v) and* *(p1, p2, …,pm) in (SPG) and* *(iv r1 p1) and* *(iv r2 p2) and* *. . .* *(iv rm pm);*	**Here,** • **SSF parameters types in the SSF parameter generator table are** $T_1,.., T_m$ **(Table 3)** • ***IT*** **is the index key type in the index key generator table (Table 2)** • ***SPG*** **is the SSF parameter generator (Table 3) for parameters** $p_1,…,p_m$**, and that** r_i **are the** ***relops*** **in Form (i).**

Example
select iv, v ***from Number iv, Number v, Number p1, Number p2*** ***where I_Table(iv, v) and*** ***(p1, p2) in (select l, u from Number l, Number u*** ***where l in uniform_int(100, 0,10000) and*** ***u in uniform_int(100,0,10000)) and*** ***iv >= p1 and iv <= p2;***

SSF tester – Validation queries

- Validation queries matching Form (II)

<table>
<tr><th>Formula</th><th>Descriptions</th></tr>
<tr><td>select iv, v
from IT iv, Number v,
T_1 p_1, T_2 p_2,.., T_m $p_{m,}$,
T_j res
where I_table(iv, v) and
(p_1, p_2, …,p_m) in (SPG) and
res = ISF (iv, p_1,..,p_{j-1}) and
(res r_1 p_j) and
. . .
(res r_m p_m);</td><td>Here,
• j is the arity of ISF()</td></tr>
<tr><th colspan="2">Example</th></tr>
<tr><td colspan="2">select iv, v
from Vector of Number iv, Number v, Vector of Number p1,
Number p2, Number res
where I_Table(iv, v) and
(p1, p2) in (select x, d from Number x, Number d
where x in uniform_vec_real(100,5,0,1) and
d in uniform_real(100,0, 1.4)) and
res = distance(iv, p1) and res<= p2;</td></tr>
</table>

SSF tester – Validation queries

- Validation queries matching Form (III)

Formula
select iv, v *from IT iv, Number v,* $T_1\ p_1, T_2\ p_2,.., T_m\ p_m,$ *where I_table(iv,v) and* $(p_1, p_2, \ldots, p_m)$ *in (SPG) and* *(iv,v) in ISF* $(p_1,..,p_m,$ *I_Table)*
Example
select iv, v ***from Vector of Number iv, Number v, Vector of Number p1, Number p2*** ***where I_Table(iv, v) and*** ***(p1, p2) in (select x, k from Number x, Number k where x in uniform_vec_real(100,5,0,1) and k in uniform_int(0,5)) and*** ***(iv,v) in knn(p1, p2, #'images');***

Experiments

Experiment - Purposes

- Code size
  - Compare coding size to introduce some indexes between Mexima vs other extensible indexing frameworks
  - To show Mexima requires no code change, driver code (glue code) is small
- BAO overhead
  - Time to run a stand-alone index implementation
  - Time to run it when plugging into Mexima
  - To investigate the overhead = Penalty of using Mexima
- Impact of SSF translation rules
  - Time to run queries with/without SSF translation rules
  - To show the importance of query rewrite to utilize indexes

Experiment - Settings

- All performance experiments were repeated 10 times, from which the average figures were calculated after removing outlier results if any.
- The experiments were run under Windows 7 on an Intel (R) Core(TM) i5 760 @2.80GHz 2.93 GHz CPU with 8GB RAM, using the Visual Studio 10 32 bits C compiler.

Experiment – Code size

- Count number of code C/C++ lines of glue code vs other extensible indexing systems

	GiST	SP-GiST	Mexima	Factor
B-tree	5031	--	116	43
KD-tree	--	572	118	5
R-tree	1133	--	120	9.5
Trie	--	580	120	5

- PostgreSQL version 9.3.5, http://www.postgresql.org/ftp/source/v9.3.5/
- SP-GiST version 0.0.1, https://www.cs.purdue.edu/spgist/
- Mexima http://www.it.uu.se/research/group/udbl/mexima

- Mexima requires
  - no code change to the index implentation
  - litle coding effort for the driver (interface)

Experiment – BAO overhead

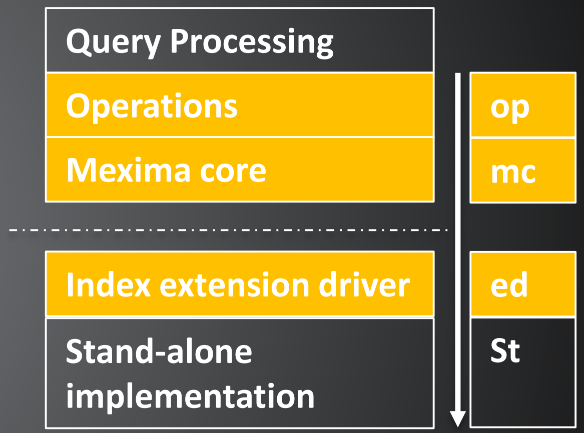
- *The total time*

tot = op + mc + ed + st

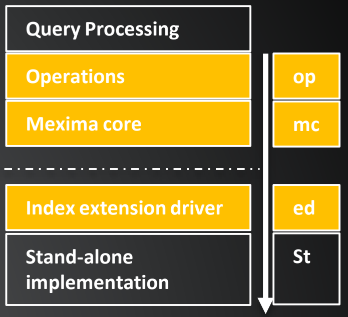
- The *Mexima overhead,*

o= op + mc + ed

- Breaking down the overhead
  - %op ?
  - %mc?
  - %ed?



Experiment – BAO overhead (cont.)

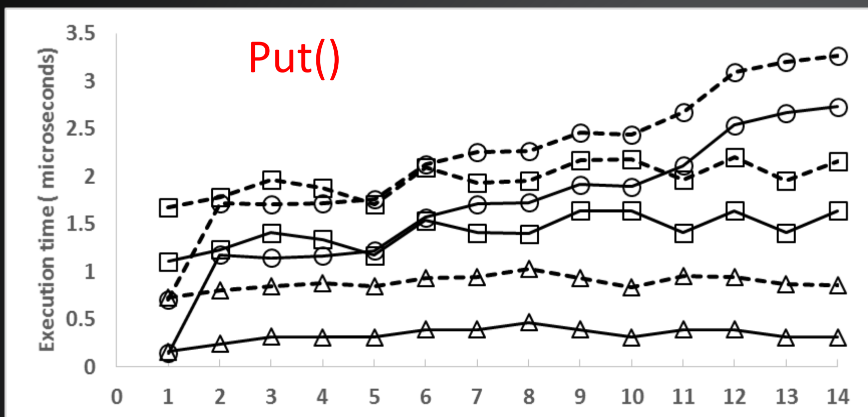


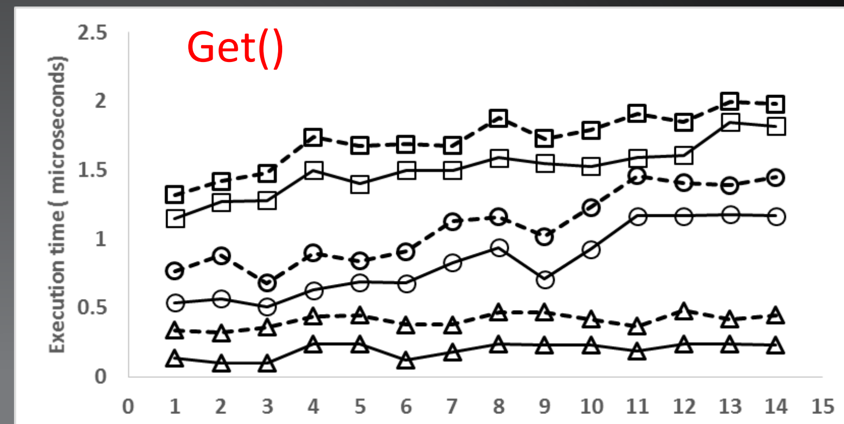
BAO	Index	o	%op	%mc	%ed
Put	LH	0.56	51.7%	36.2%	12.1%
	B-tree	0.53	52.3%	35.8%	11.9%
	Judy-trie	0.54	52%	35.3%	11.7%
Get	LH	0.26	37.2%	47.1%	15.7%
	B-tree	0.23	36.6%	47.6%	15.7%
	Judy-trie	0.22	36%	48%	16%
Map	LH	0.07	32.1%	50.9%	17%
	B-tree	0.07	34.4%	49.2%	16.4
	Judy-trie	0.07	33.7%	49.7%	16.6%
Delete	LH	0.42	45%	41.3%	13.7%
	B-tree	0.42	43.3%	42.5%	14.2%
	Judy-trie	0.41	43.4%	42.5%	14.1%

- Data size = 5 milion key/value pairs
- 1000 random inserts, lookups, deletes.
- The average overhead in microseconds per call

➔ Overhead < 0.6 microsecond

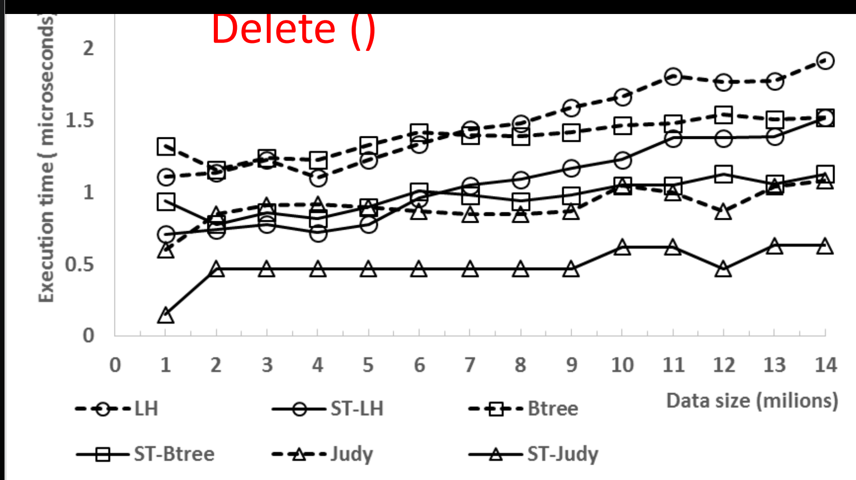
Experiment – Overhead w.r.t data size

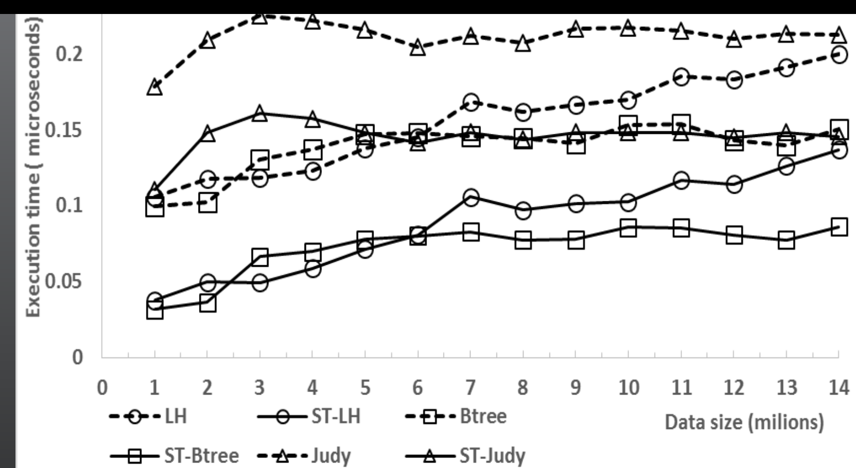




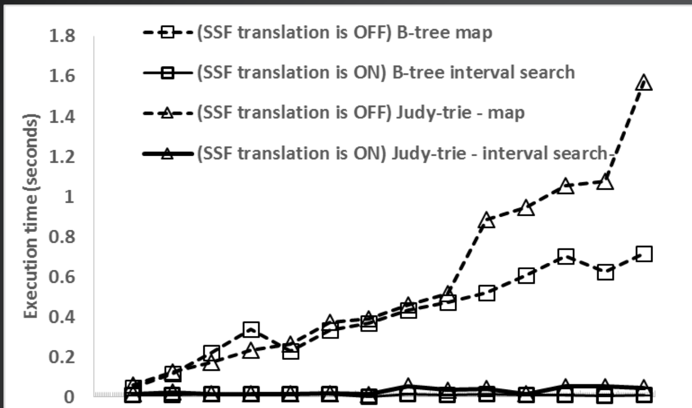
The stand-alone index implementations are always faster

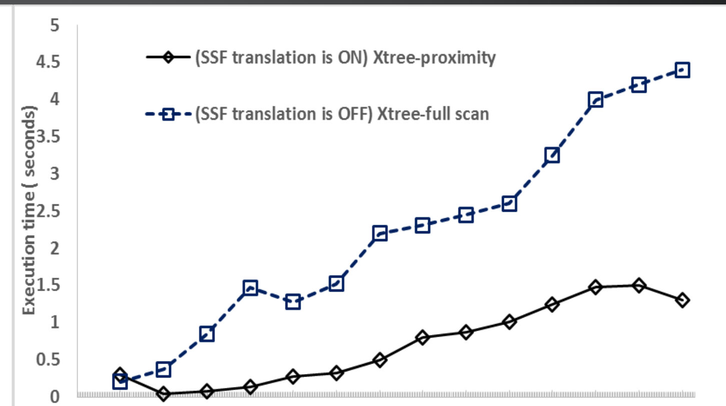
The overhead is independent of the database size





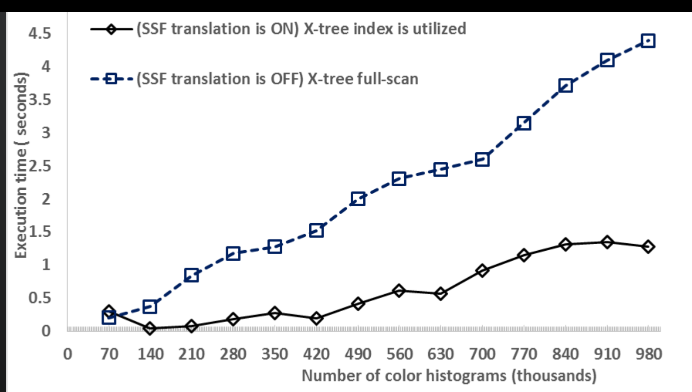
Experiment – Impact of SSF translation rules

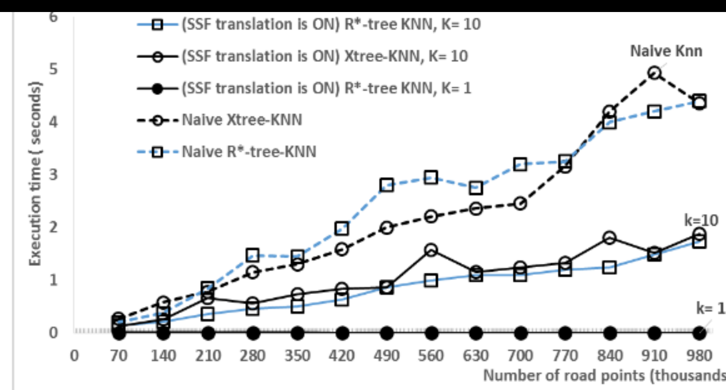




With SSF translation rules, queries run faster.

It made indexes utilized





Xt-trees on Proximity search

X-trees and R*-trees on KNN search

Experiment – Side notes

- Bugs found in the following used index implementations using Mexima tester
  - X-trees [1]
  - R*-trees [2]
  - B-trees [1]
- Comparisons
  - Judy-trie [ref] outperformed B-trees in get(), insert(),delete(), but not map()
  - For 2D – 4D, X-trees is as good as R*-trees
  - For higher dimension (9D), X-trees is applicable and scale

[1] http://www.it.uu.se/research/group/udbl/mexima

[2] http://www.ics.uci.edu/~salsubai/rstartree.html

Conclusions & Future work

- Conclusions
  - The Mexima framework allows plugging-in of main-memory domain index implementations with ease
    - without code changes
    - a simple Mexima driver for BAOs and SSFs
    - declare index properties as queries
    - transparently, Mexima makes new indexes utilized
    - automatically generating and executing validation queries, Mexima validates correctness of BAOs and SSFs
  - Tool for testing and comparing indexes
- Future work
  - More indexes will be plugged-in
  - It might put additional requirement to Mexima

Thank you!