

Relational Interval Tree

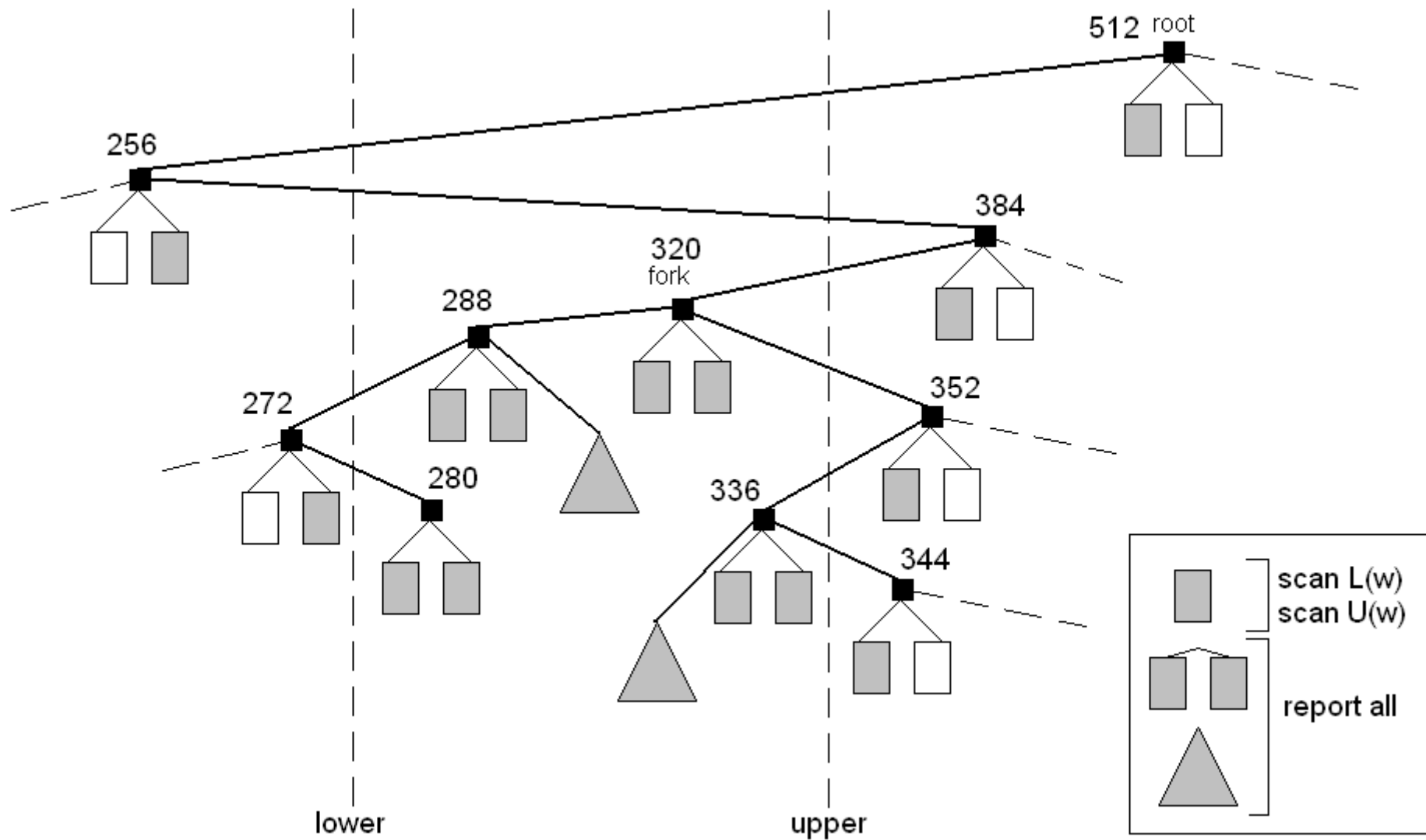
Discussion of the paper:

„Managing Intervals Efficiently in Object-Relational Databases“
from Hans-Peter Kriegel, Marco Pötke and Thomas Seidl

presented by

Daniel Bucher

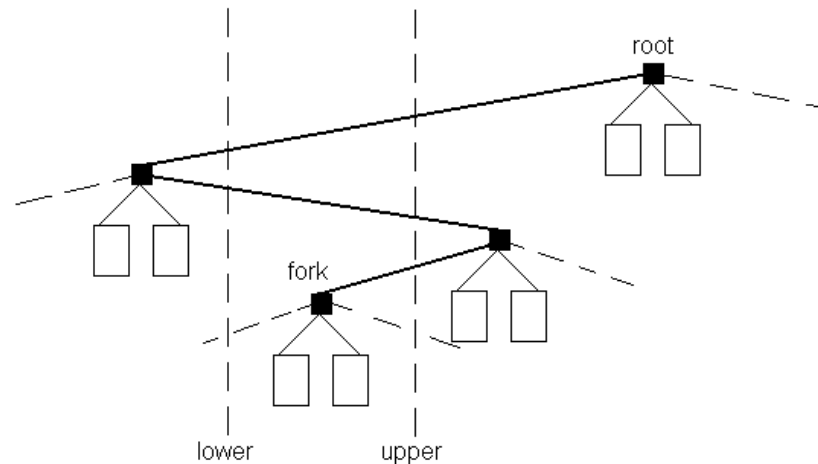
Relational Interval Tree



Relational Interval Tree

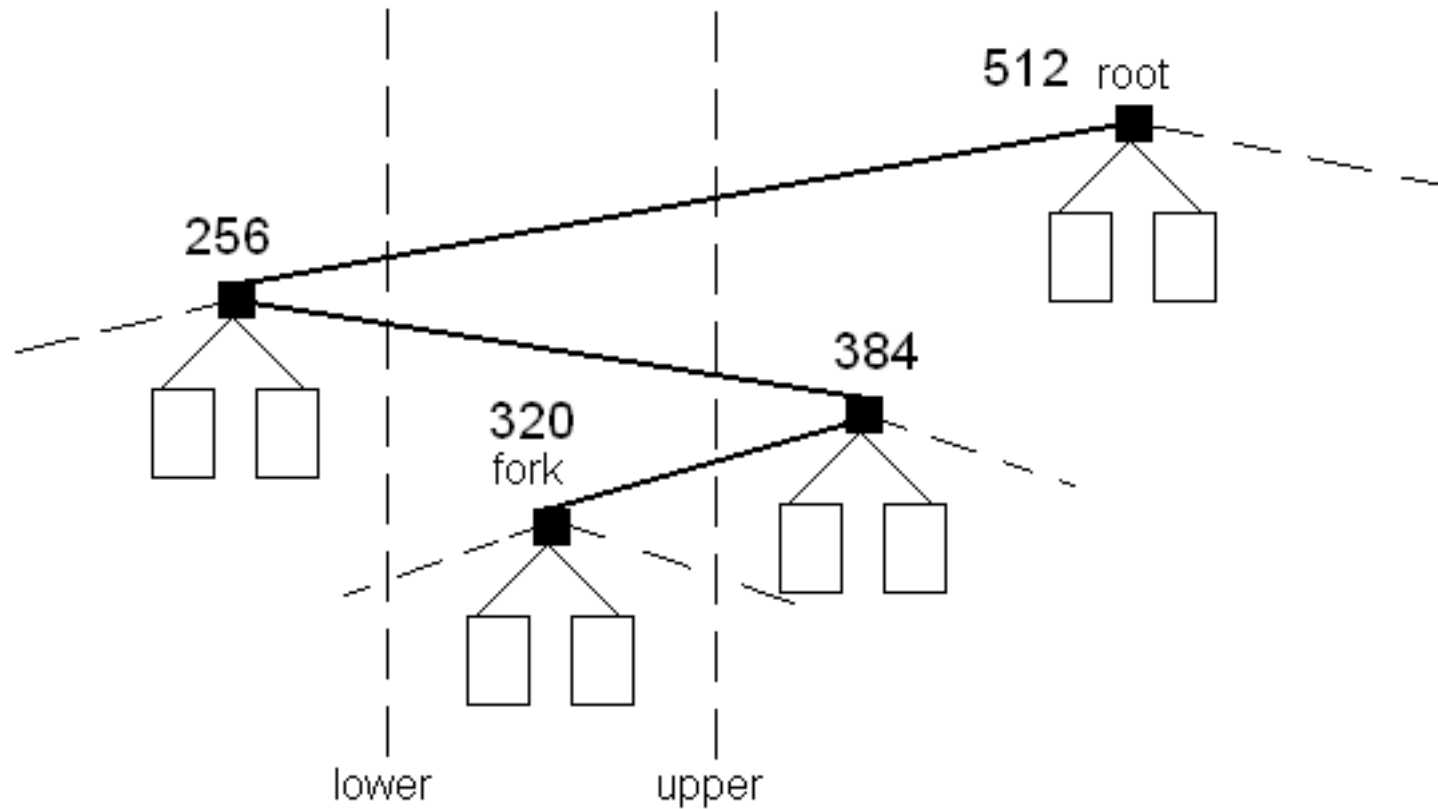
Every interval (l,u) is registered at the first node w from root which it intersects.

This node is called fork node



Also points are possible and represented as (p,p)

Example



Inserting interval (300, 380)

Relational Interval Tree

Mapping the secondary structure to a relational schema:

- For every list $L(w) = \{l_1, \dots, l_{n_w}\}$
we can represent the same information by the
tuple set $\{(w, l_1), \dots, (w, l_{n_w})\}$
- We do exactly the same for $U(w)$

These two relations (node, lower) and (node, upper) can be organized efficiently by the already built-in composite indexes of the RDBMS.

Creating the index

```
CREATE TABLE Intervals
```

```
  (node int, lower int, upper int, id int);
```

```
CREATE INDEX lowerIndex
```

```
  ON Intervals (node, lower);
```

```
CREATE INDEX upperIndex
```

```
  ON Intervals (node, upper);
```

Updates

To insert an interval (l,u) :

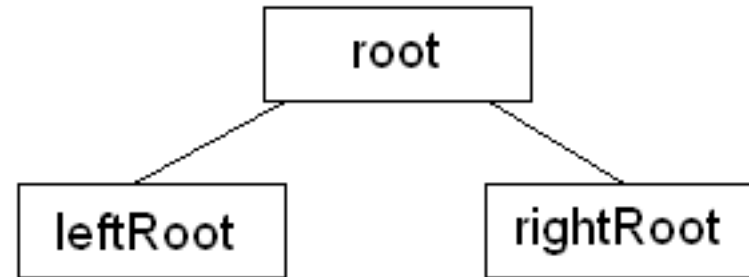
- Search for its fork node
- Add its l to $L(w)$ and its u to $U(w)$

We do this by inserting the relational indexes into the schema from before.

Dynamic Expansion of the Data

For this we need new parameters:

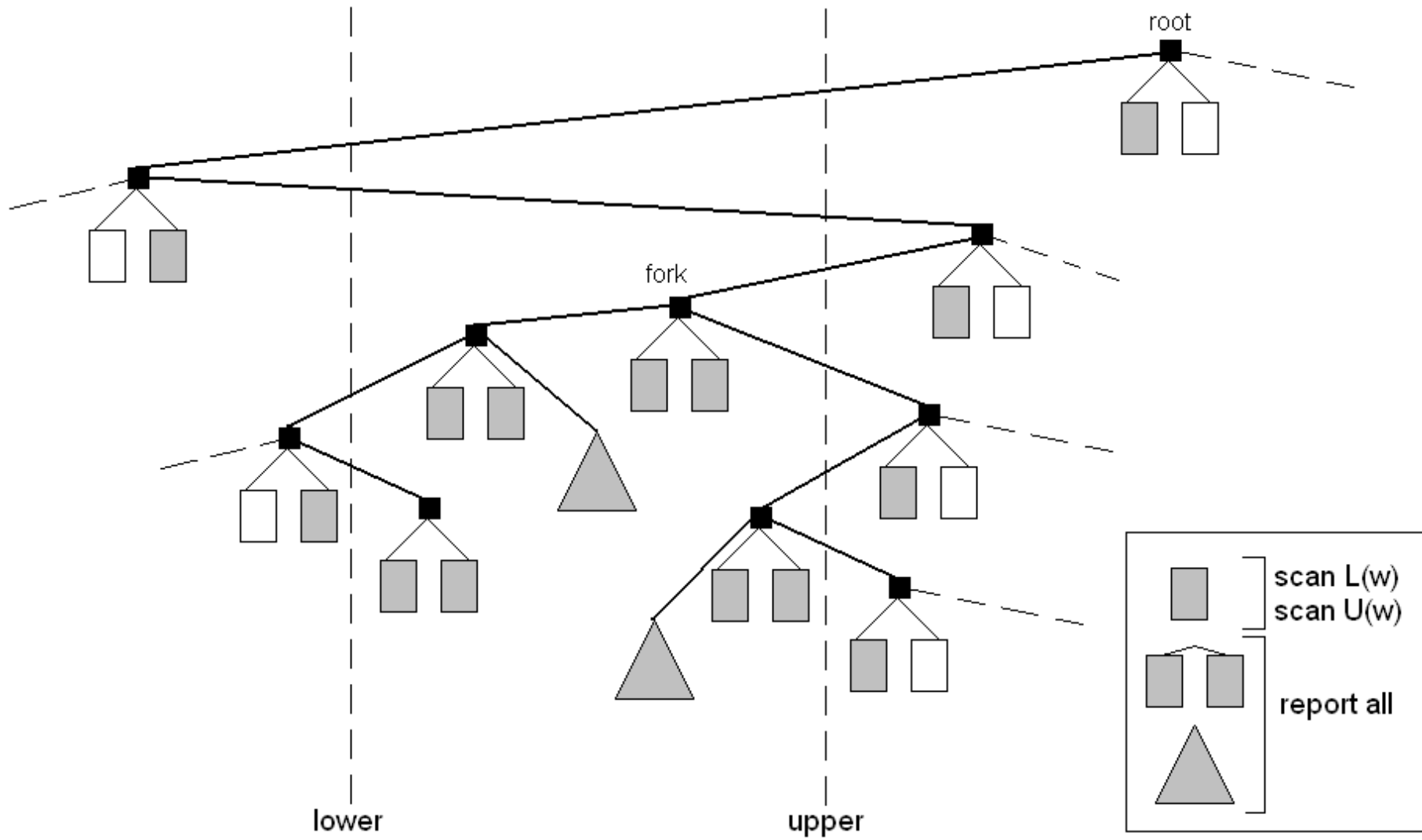
- root
- offset
- leftRoot and rightRoot
- minstep



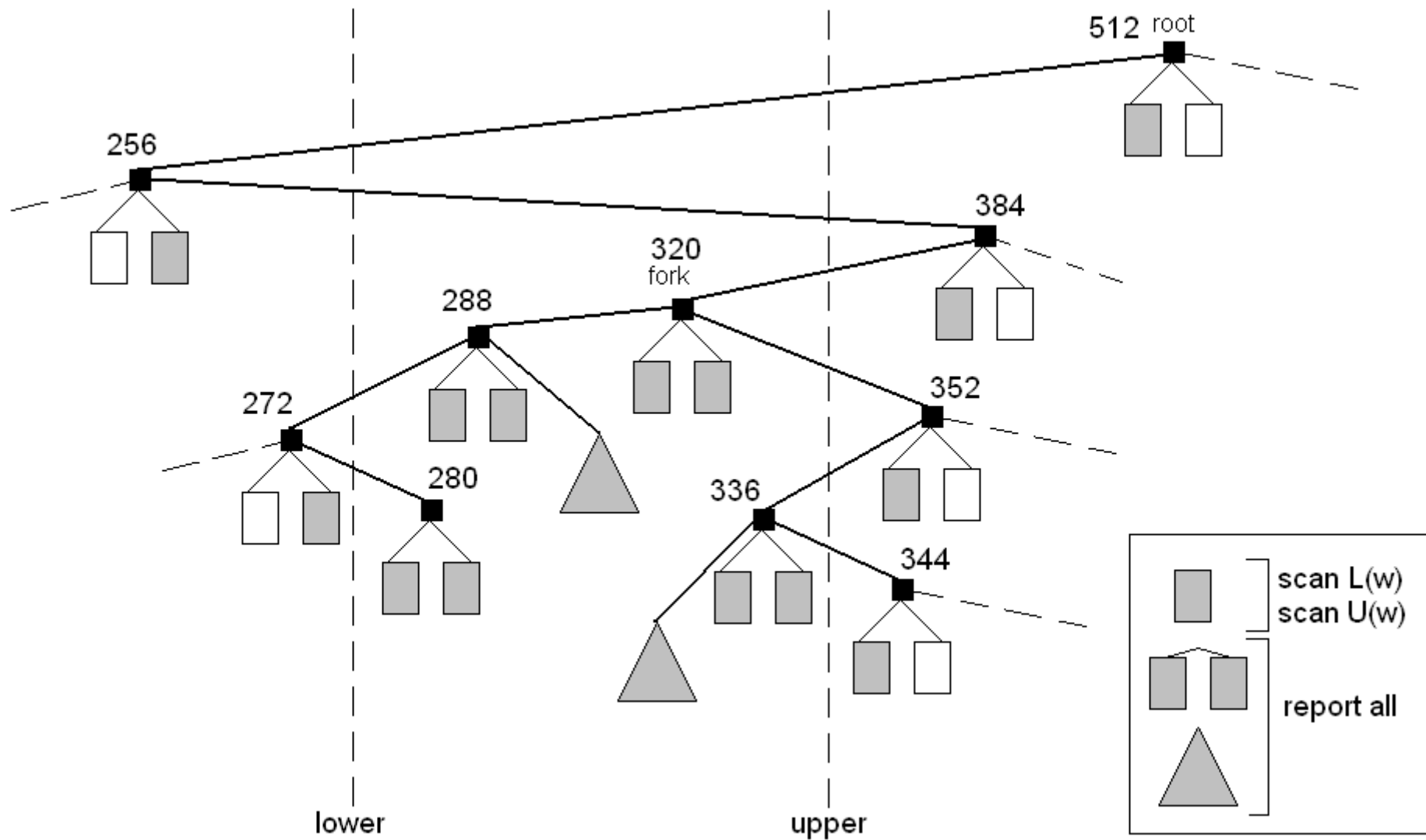
Query

- Search the fork node
- All nodes on the path to the fork node lie on the left or on the right of the query interval so check the corresponding list
- Descend from the fork node as deep as necessary and check required lists.

Query



Example

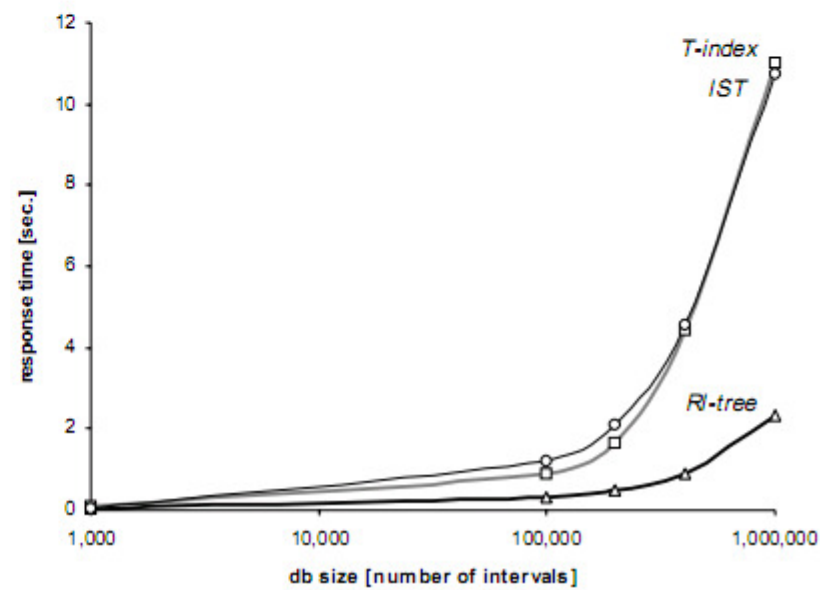
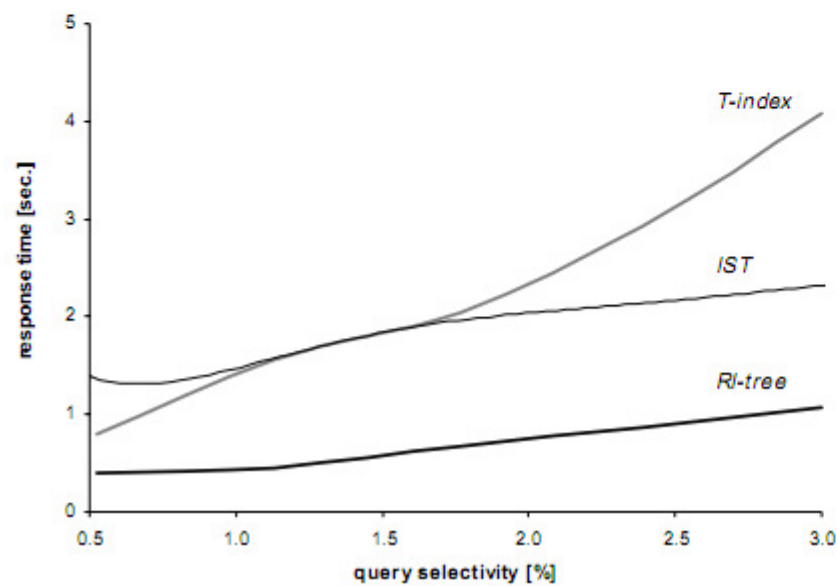
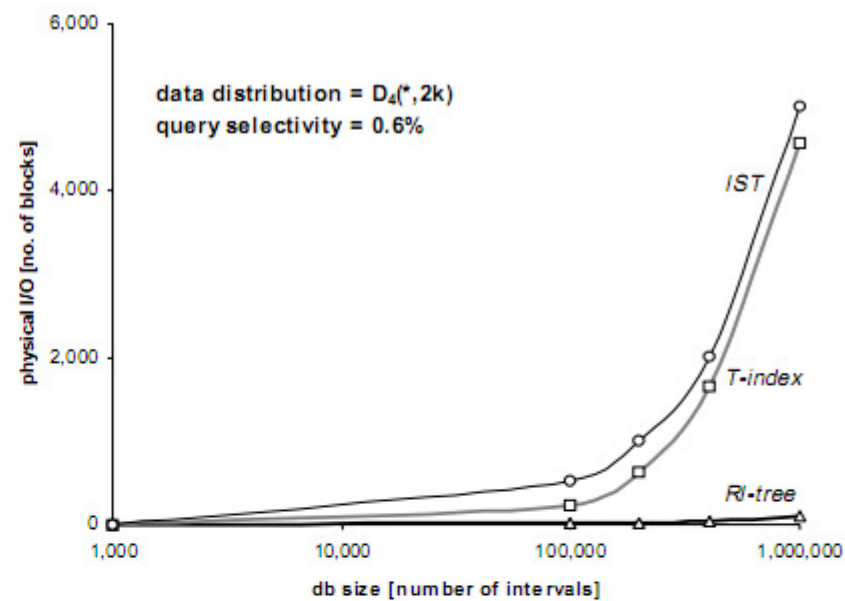
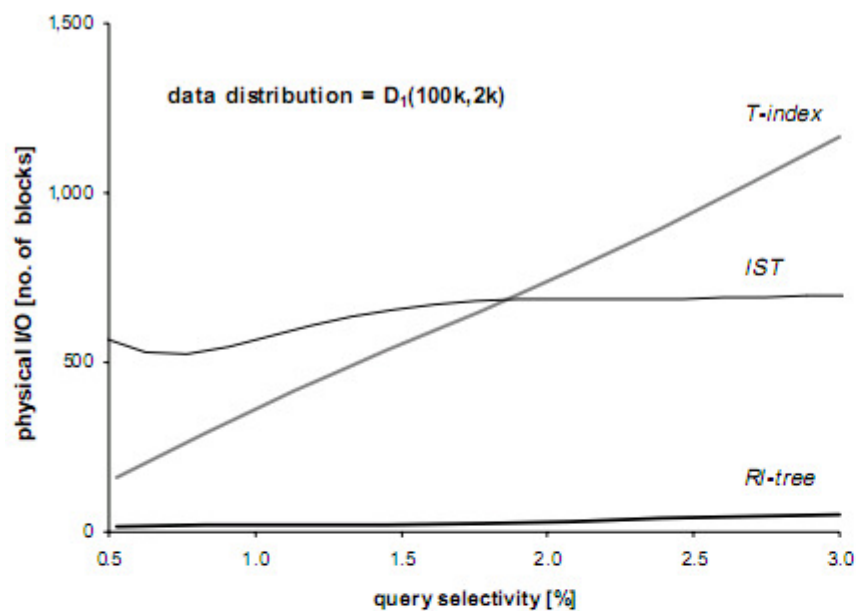


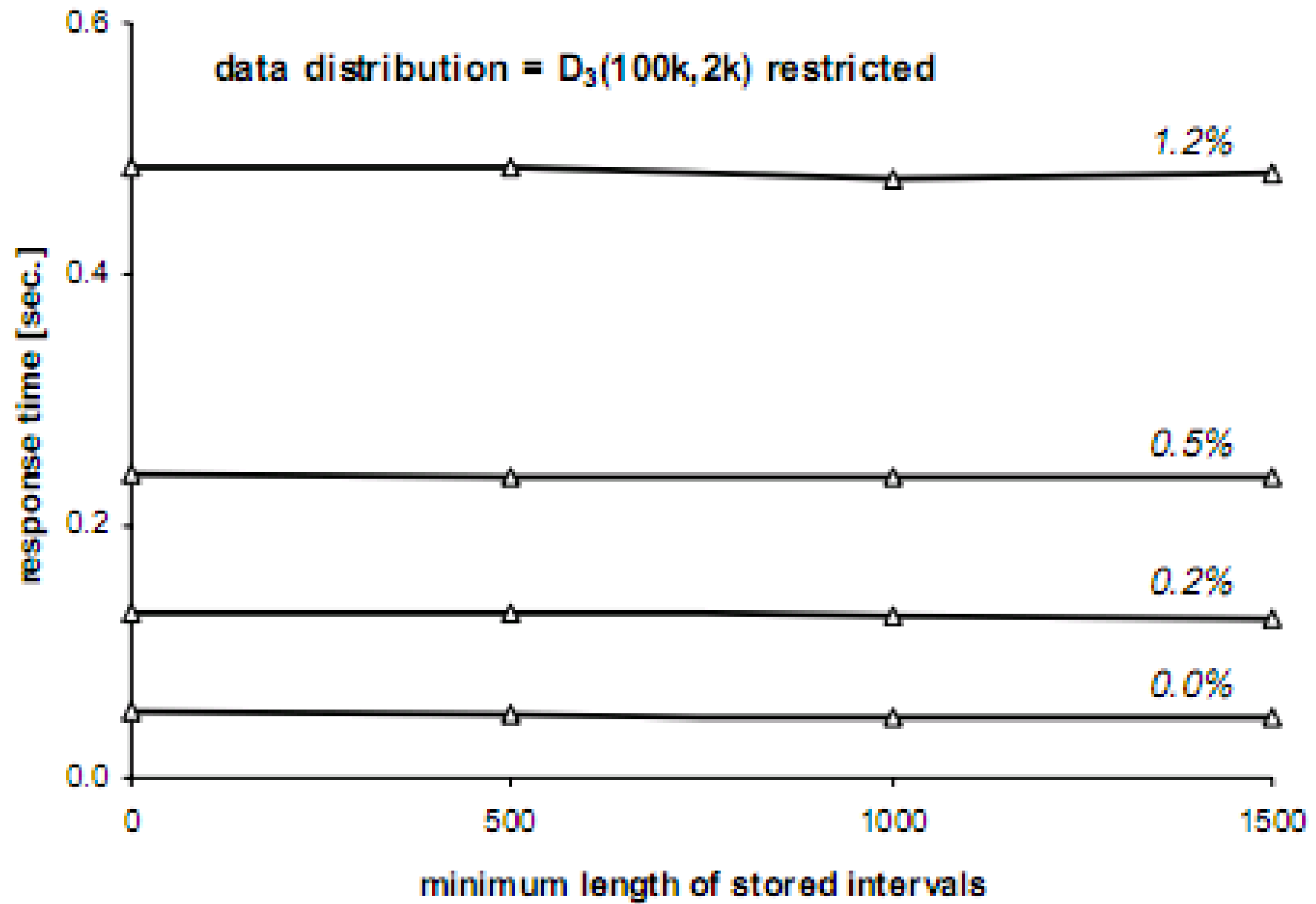
Query interval (274, 340)

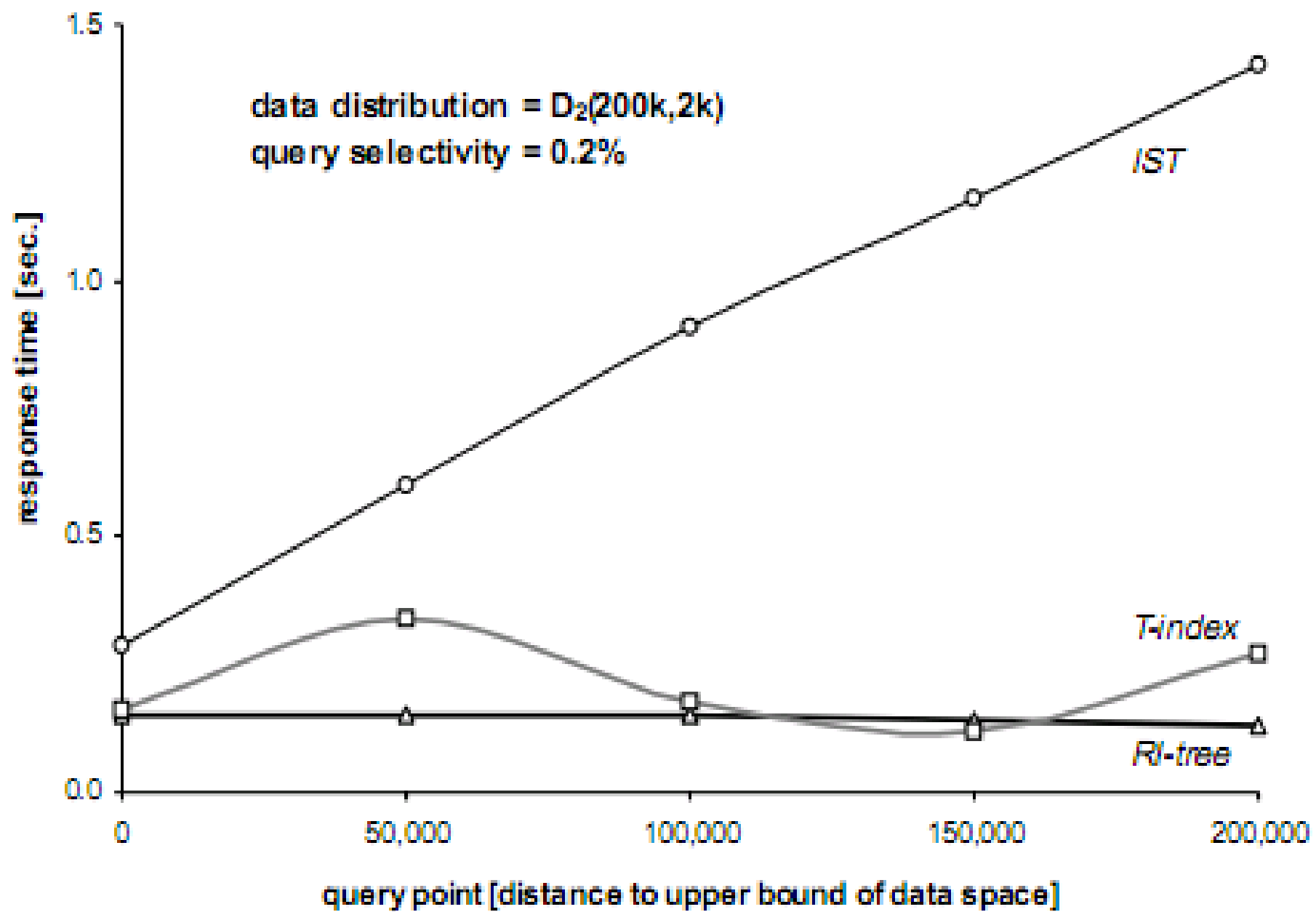
Handling Temporal Intervals

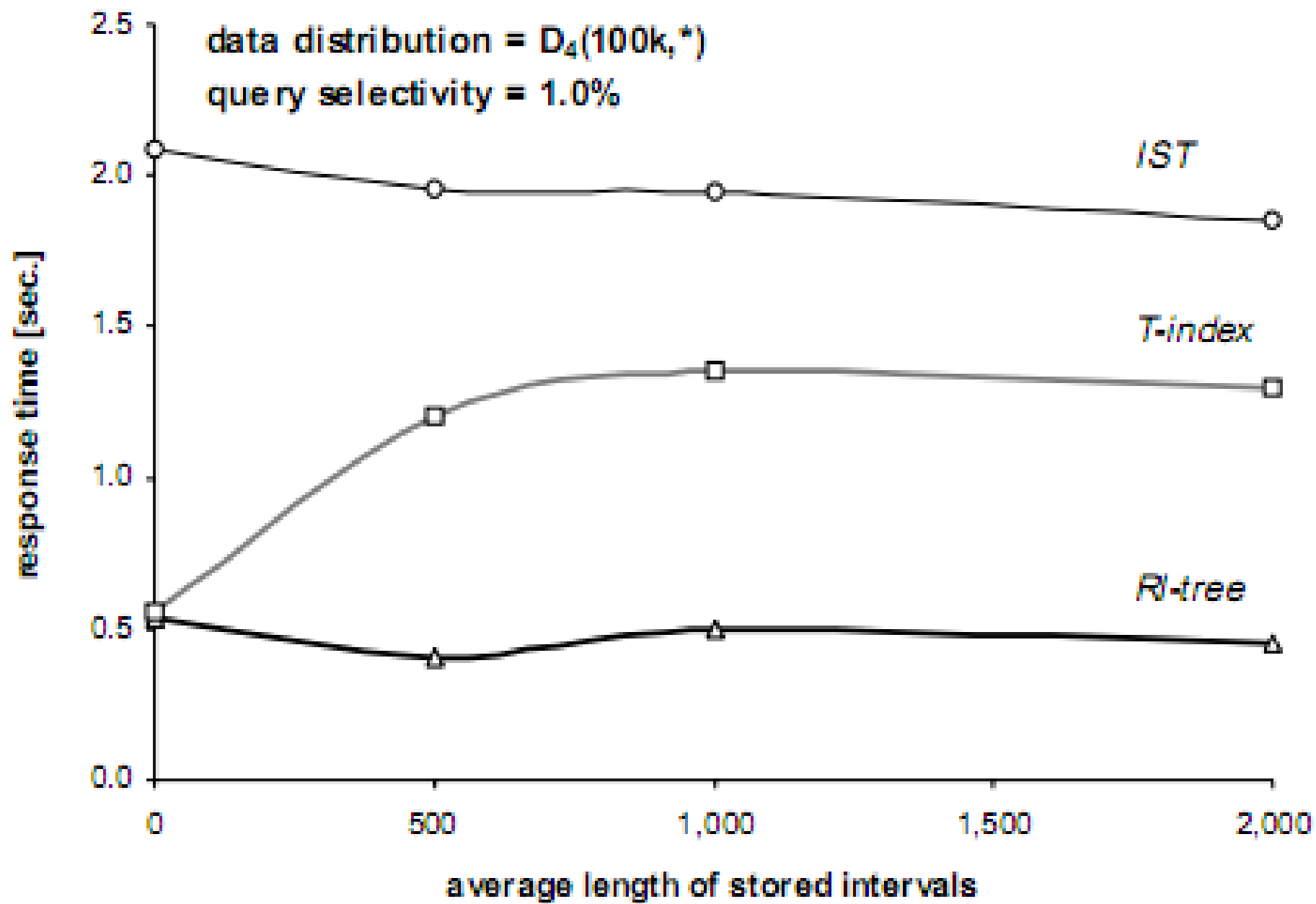
- $\text{infinity} = \text{fork}_{\infty}$
- $\text{now} = \text{fork}_{\text{now}}$

Replaced at query time by MAXINT and MAXINT-1 so it's still possible to use the normal SQL query algorithm and the tree doesn't degenerate.









Conclusion

- Integration possible in any relational DBMS because it uses the built-in indexes and needs no changes or additions. Only needs SQL support.
- Outperforms competitors which also could be integrated
- Thanks to parameters now and infinity temporal interval are no problem.

The RI-Tree is great!

Insertion

PROCEDURE insertInterval

```
(int lower, int upper, int id){
//initialize offset and shift interval
if (offset = NULL) offset = lower;
int l = lower-offset, u = upper-offset;

// update leftRoot and rightRoot
if (u<0 and l <= 2*leftRoot)
    leftRoot = -2^floor((log2(-l)));
if (0<l and u >= 2*rightRoot)
    rightRoot = 2^floor((log2(u)));

// descend the tree down to the fork node
int node, step;
if (u<0) node = leftRoot;
elseif (0<l) node = rightRoot;
else /* 0 is fork node */ node =0;
```

```
for (step = abs(node/2);
    step >= 1; step /=2){
    if (u<node) node -= step;
    elseif (node<l) node += step;
    else /* fork reached */ break;
}

if (node != 0 and step < minstep)
    minstep = step;

INSERT INTO Intervals
VALUES (:node, :lower, :upper, :id);
}
```

SQL Query

```
SELECT id FROM Intervals i,  
       leftNodes left, rightNodes right  
WHERE (i.node = left.node  
       AND i.upper >= :lower)  
OR (i.node = right.node  
    AND i.lower <= upper)  
OR (i.node BETWEEN :lower – offset  
    AND :upper – offset);
```

Optimized SQL Query

```
SELECT id FROM Intervals i, leftNodes left
  WHERE (i.node BETWEEN left.min AND left.max
        AND i.upper >= :lower
UNION ALL
SELECT id FROM Intervals i, rightNodes right
  WHERE i.node = right.node
        AND i.lower <= :upper
```