Optimizing Multidimensional Index Trees for Main Memory Access

Authors: K. Kim, S. Cha, K. Kwon
Seoul National University, Korea
SIGMOD 2001, California, USA

Presenter: Christian Christiansen

Outline

- Motivation
  - Disk-based vs. Main memory databases
- The CR-tree (Cache conscious)
  - Structure
  - Techniques
    - Pointer elimination
    - MBR key compression
    - Quantization
- Performance studies
- Conclusions
- Evaluation

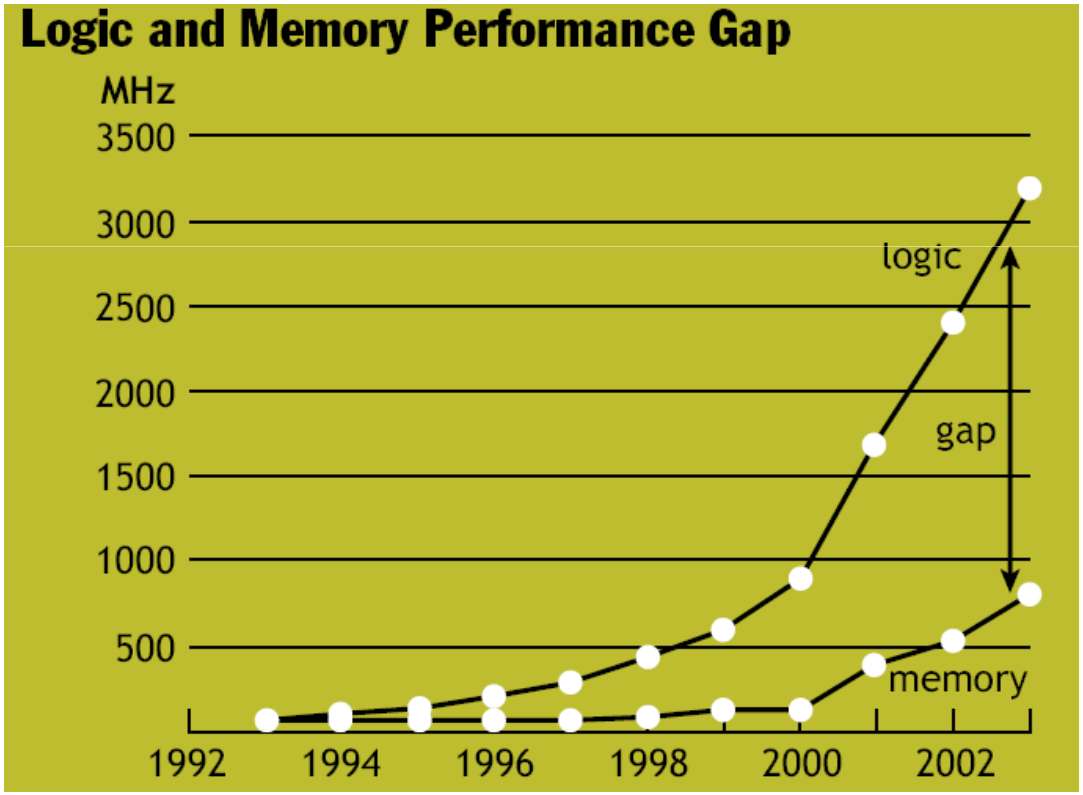
Motivation

- Memory sizes grow, prices drop
- Increasing performance requirements
- Therefore, main memory databases become feasible solution

- However:
  - Existing disk based index structures not suitable
  - Existing main memory structures not optimal
    - E.g. B+-tree, T-tree, etc.

Motivation

- Performance gap: Main memory vs. CPU



Motivation

- Main Memory hierarchy

	L1 Cache	L2 Cache	Memory
Block size	16~32B	32~64B	4~16KB
Size	16~64KB	256KB~8MB	~32GB
Hit time	1 clock cycle	1~4 clock cycles	10~40 clock cycles
Backing store	L2 cache	Memory	Disks
Miss penalty	4~20 clock cycles	40~200 clock cycles	~6M clock cycles

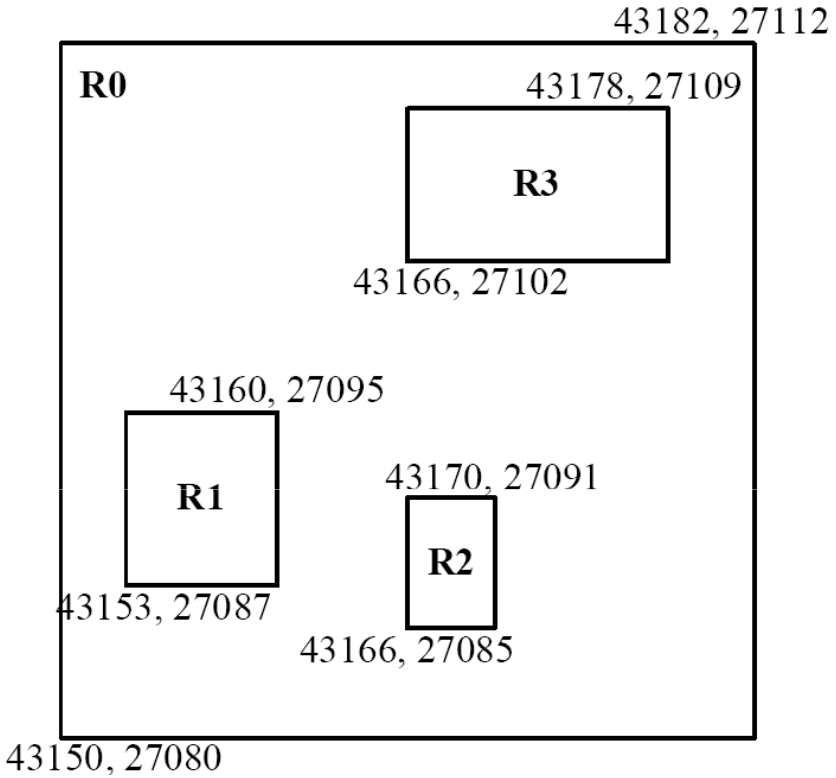
- Loading consecutive chunks of data from main memory is faster

Motivation

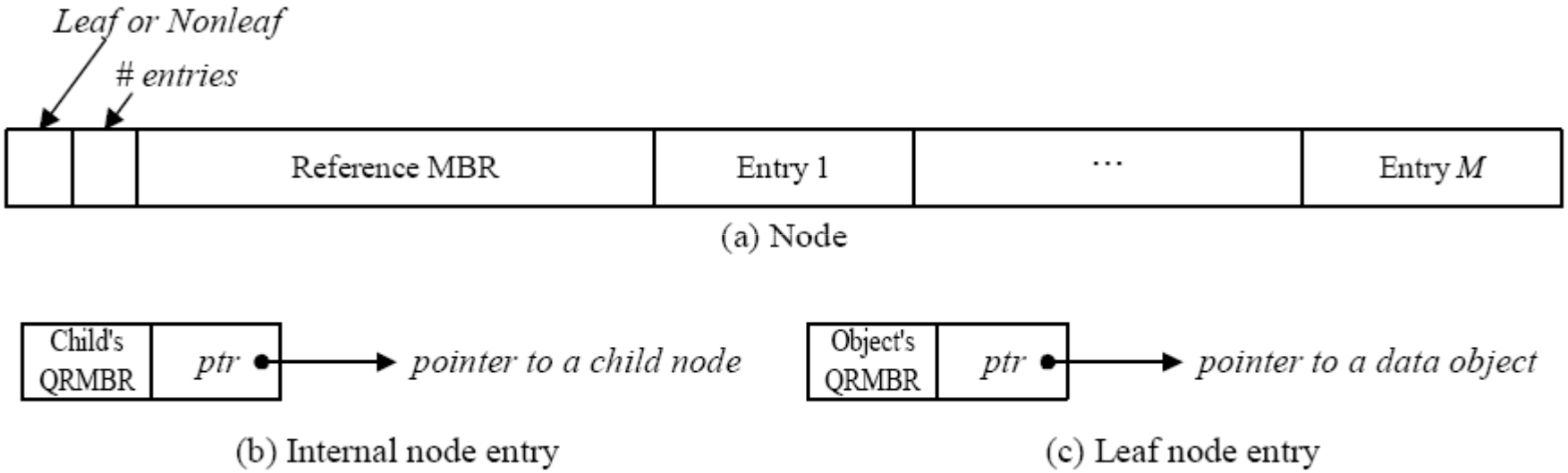
- CSS and CSB+ tree techniques
  - Not applicable for multidimensional index structures

  E.g. 16 Byte MBRs

- Pointer elimination will only reduce node size by 25%
- This does not change tree height significantly

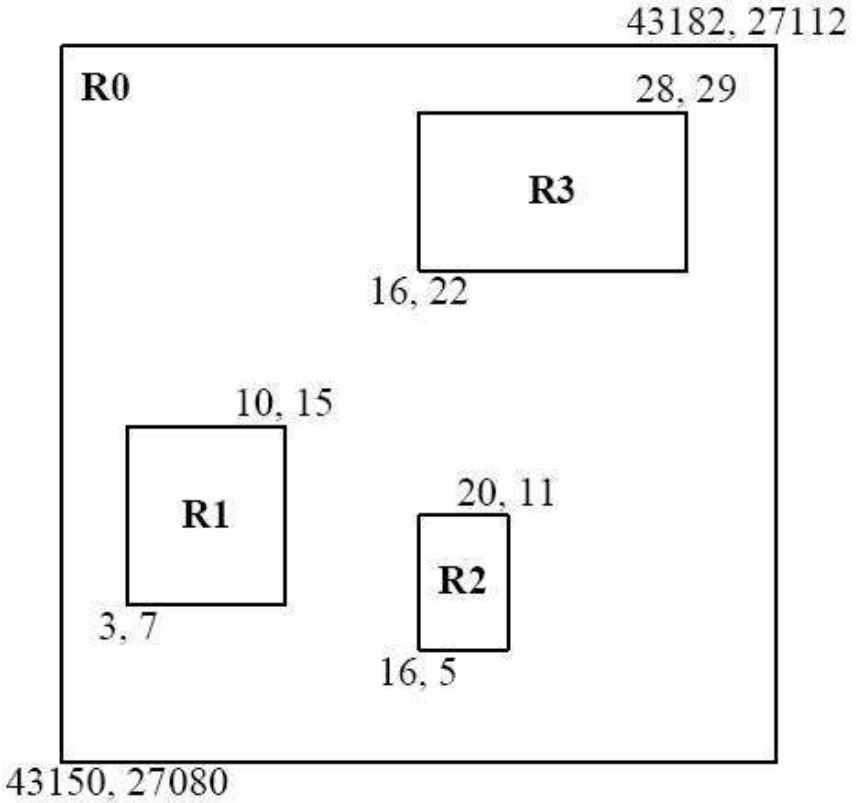


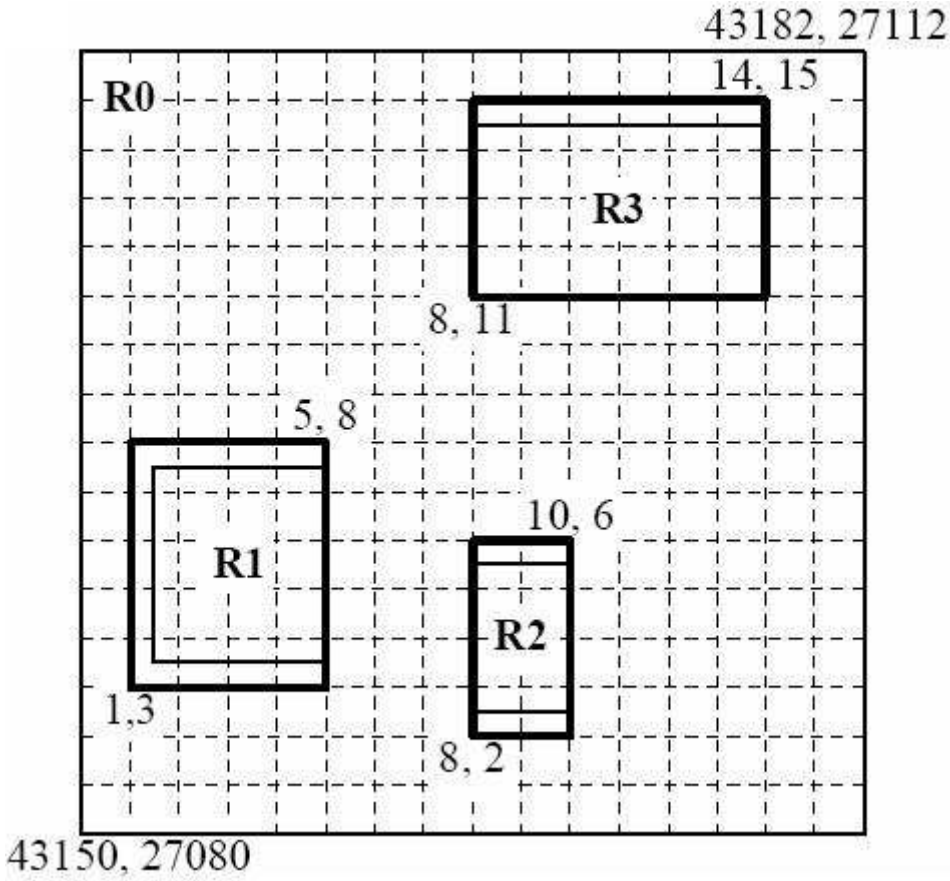
CR-tree data structure



CR-tree

- MBR key compression
- Quantization





Reducing index search time

- Key compare
- Cache miss
- TLB miss

$$T_{index\ search} = c \cdot N_{node\ access} \cdot (C_{key\ compare} + C_{cache\ miss} + C_{TLB\ miss} / c)$$

- Thus, index search time depends most on:

$$c \cdot N_{node\ access}$$

Reducing index search time

- Reducing $c \cdot N_{node\ access}$
  - Changing node size
  - Compressing index entries
  - Clustering index entries into nodes

MBR compression

- Desirable properties
  - Overlap check without decompression
  - Simplicity
    - Computationally simple
    - Use already cached data
- Guarantees provided after quantization
  - Relative representation: loss-less
  - Quantization: lossy

CR-tree variants

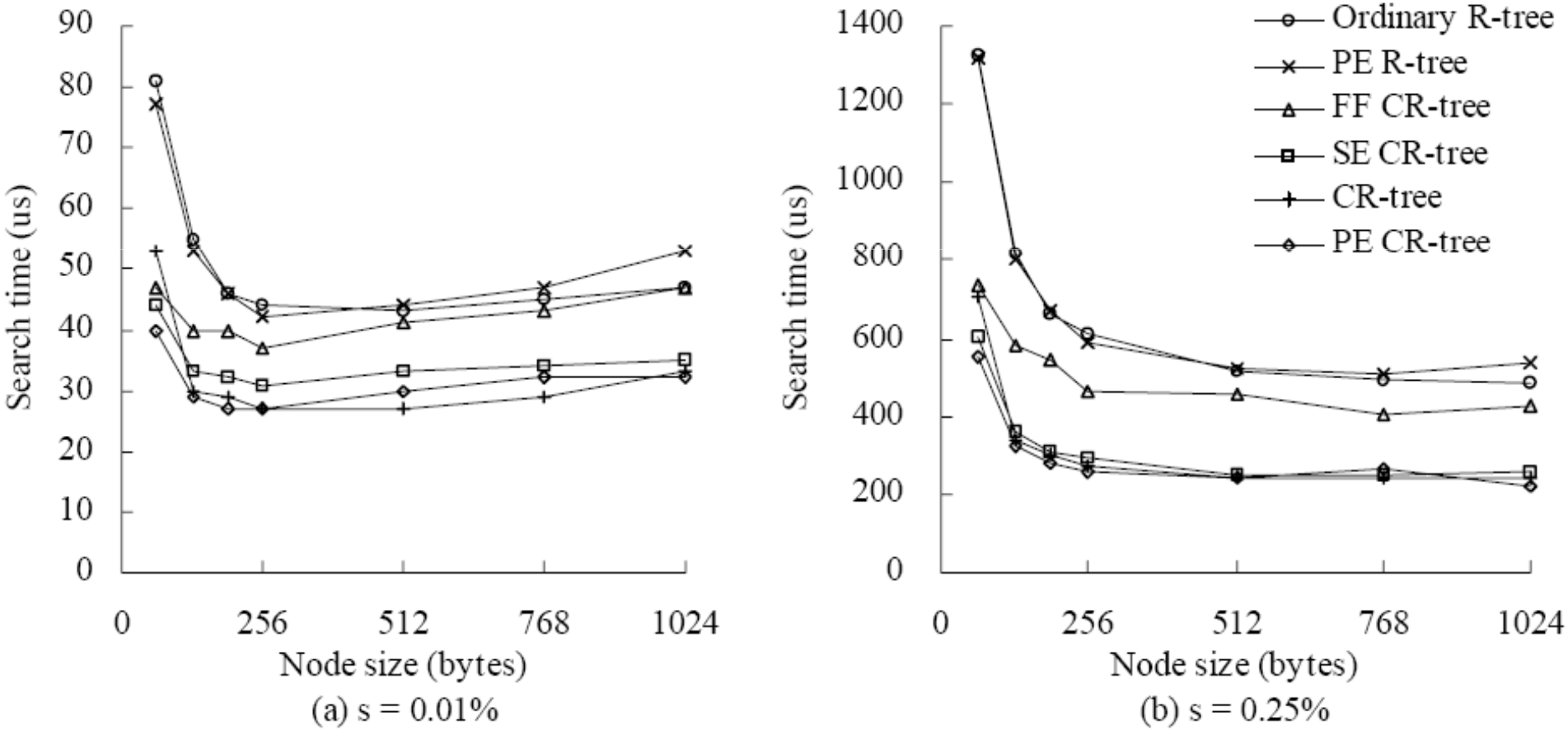
- PE CR-tree
  - Pointer Eliminated
- SE CR-tree
  - Space-Efficient
- FF CR-tree
  - False-hit Free

Performance studies

- SUN UltraSPARC
  - 400Mhz, 8MB L2 cache
- 6 implementations of 2D index structures
  - Ordinary R-tree
  - PE R-tree
  - CR-tree
  - PE CR-tree
  - SE CR-tree
  - FF CR-tree
- 2 data sets of one million small rectangles each

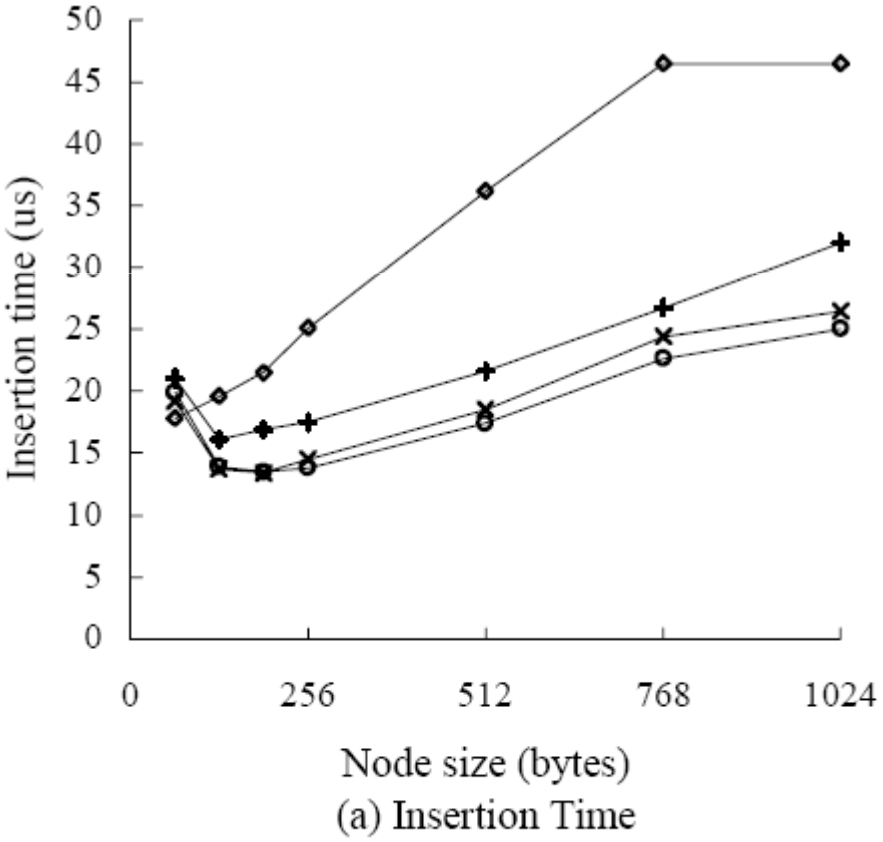
Search performance

- For 10.000 query rectangles

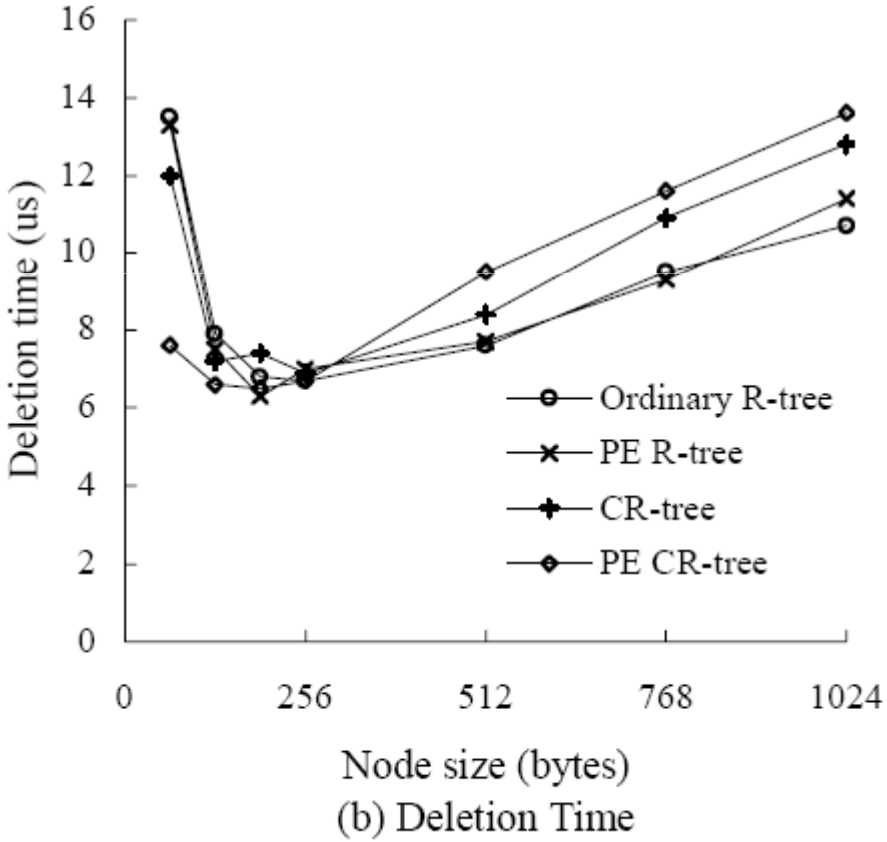


Update performance

- Inserting and deleting 100,000 objects



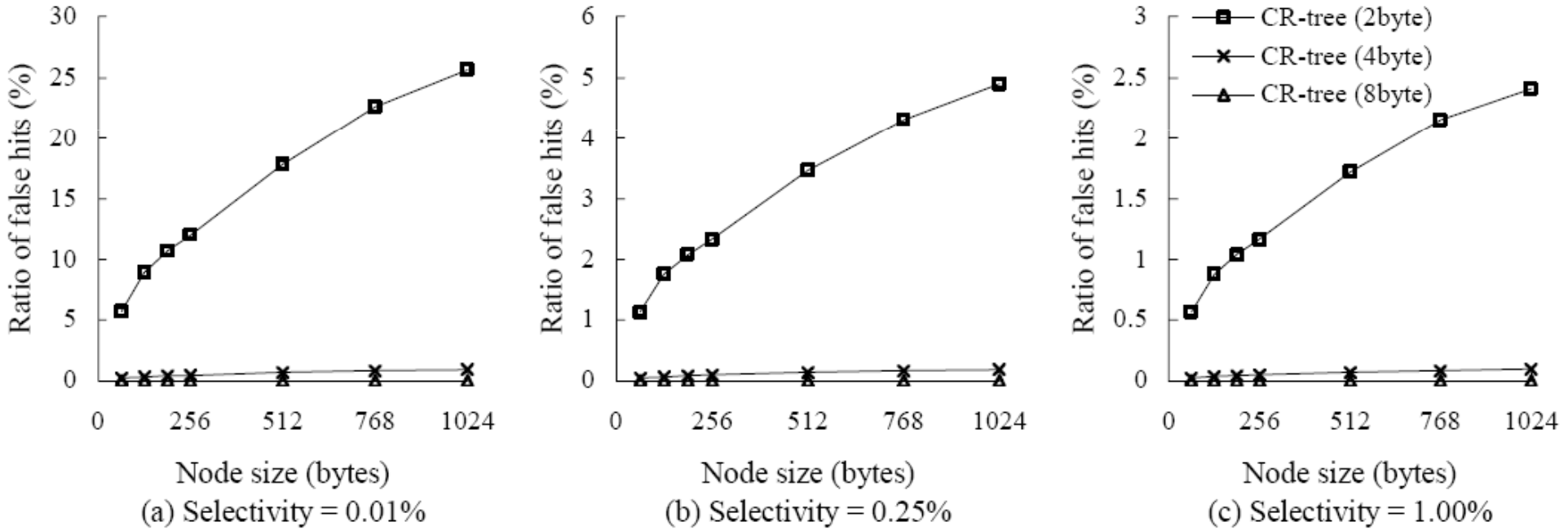
(a) Insertion Time



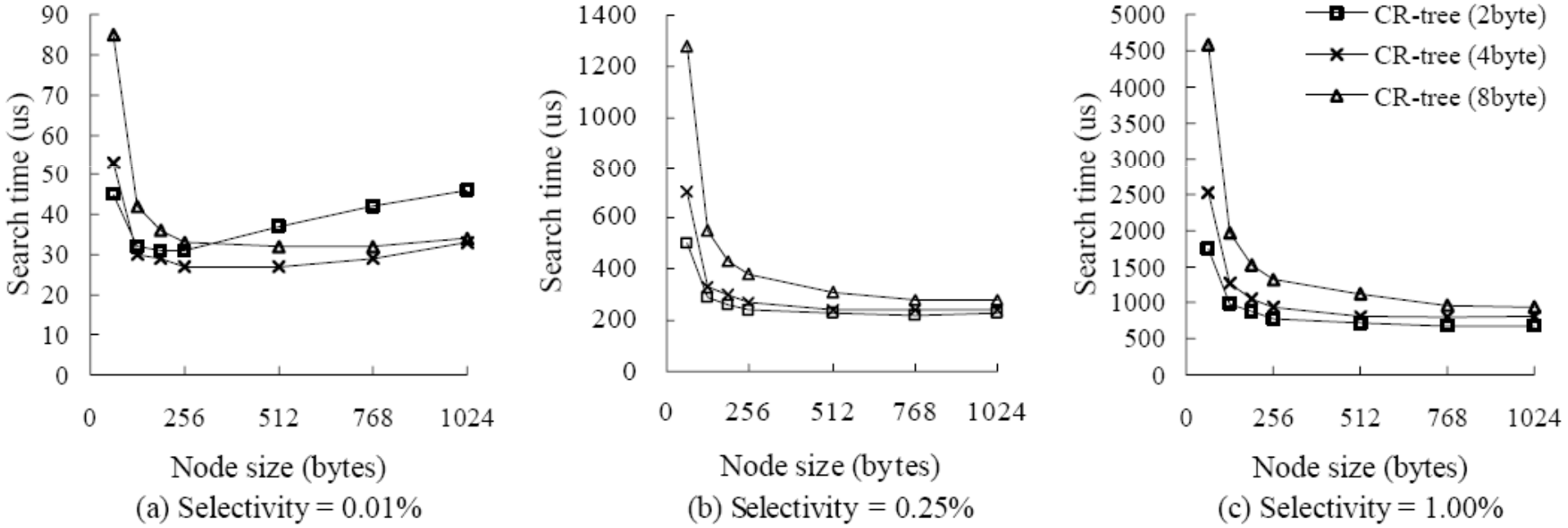
(b) Deletion Time

Impact of quantization levels

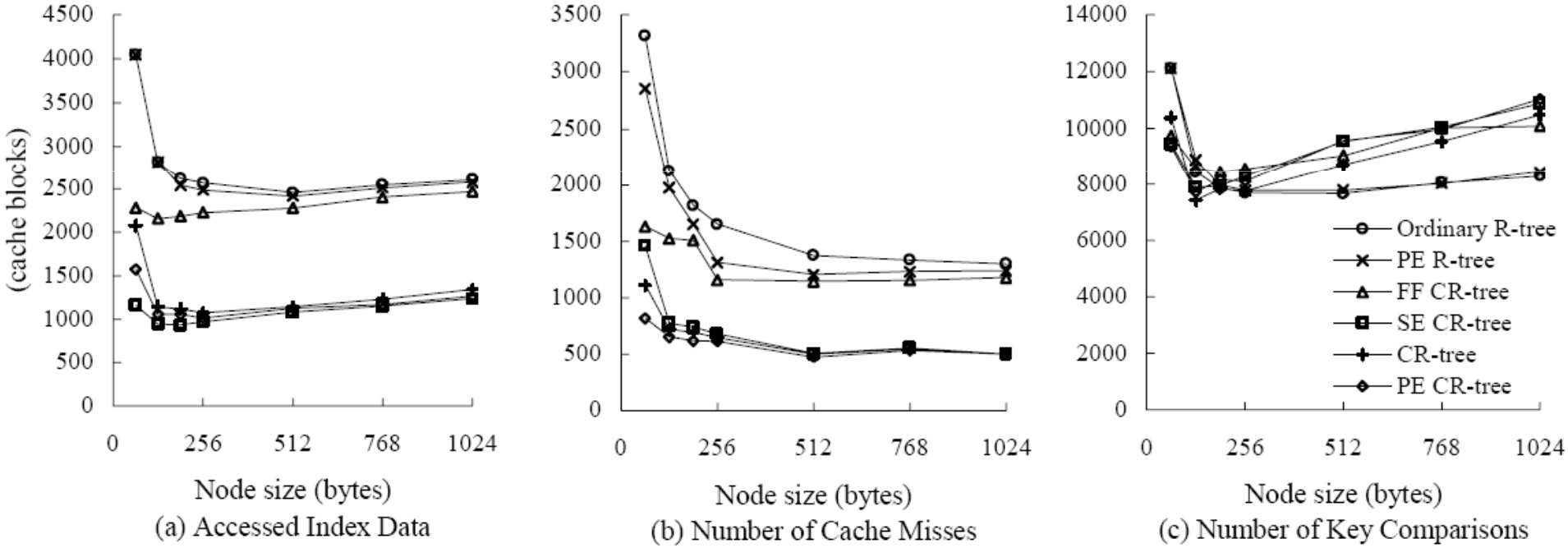
- Quantization levels 2^4, 2^8 and 2^{16} correspond to
- QRMBRs of 2B, 4B and 8B, respectively



Impact of quantization levels



More on search performance



Conclusions

- 2D CR-tree and its variants (PE, SE, FF) outperform ordinary R-tree
  - Up to 2.5 times faster search time
  - Use about 60% less memory space
  - Maintains similar update performance

Evaluation

- The good:
  - Original idea
  - Well written paper. Provides good overview
  - Actual implementation of structures to verify performance claims
- The ’could-be-improved’
  - Explanation of memory hierarchy issues
  - Certain graphs are vaguely commented
    - Fig. 5, 6+9 almost identical, 7
  - English: ”selectivity” = ”query area size”?