Main Memory Operation Buffering for Efficient R-Tree Update

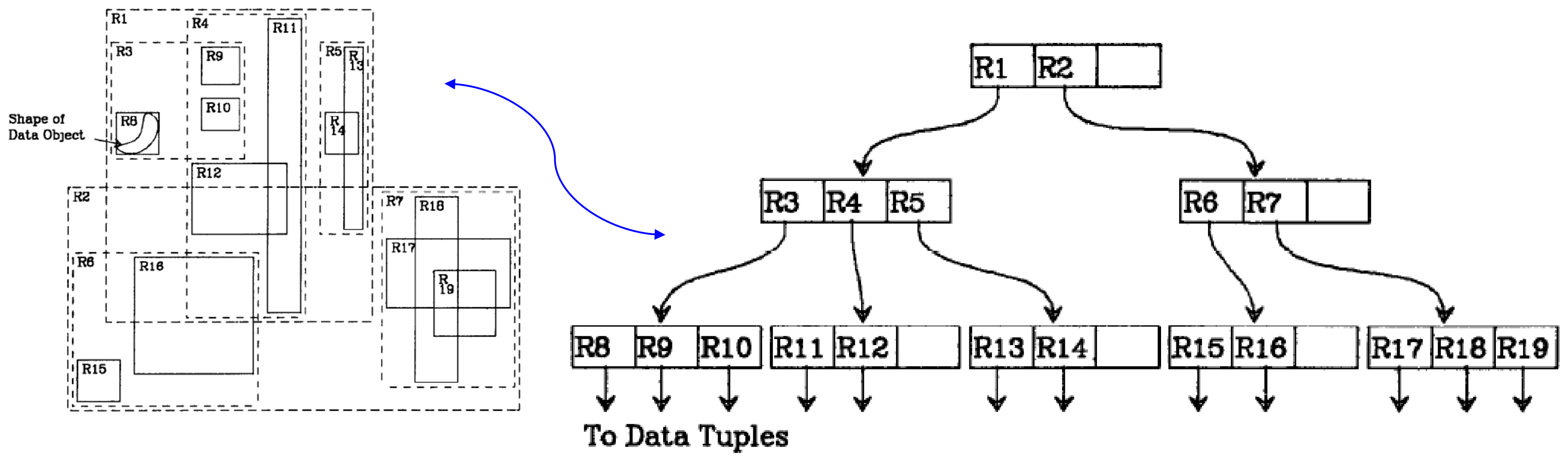
Based on Biveinis, Šaltenis and Jensen (Aalborg University) paper in VLDB 2007

daisy
Center for Data-intensive Systems

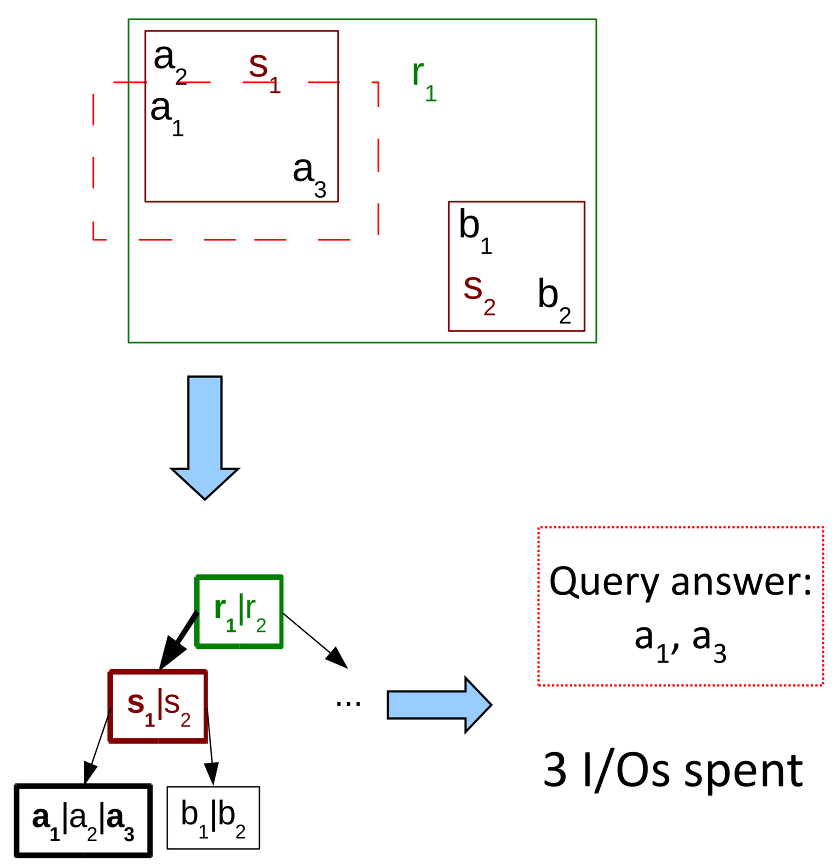
Setting

- A typical pervasive computing scenario
  - Sampling of continuous processes via large numbers of sensors
  - Maintenance of an up-to-date current state of the processes
  - Query processing against the current state
- Example: moving objects
  - The current positions of moving objects
  - Large populations of objects are anticipated (mobile phone users)
  - Updates are very frequent.
    - E.g., 600,000 objects, each issuing an update every minute yields 10,000 updates per second.
- Indexing is essential to efficient query processing.
- The index must be stored on disk, at least in part.
- Existing indices do not support massive update loads.

Background: the R-tree

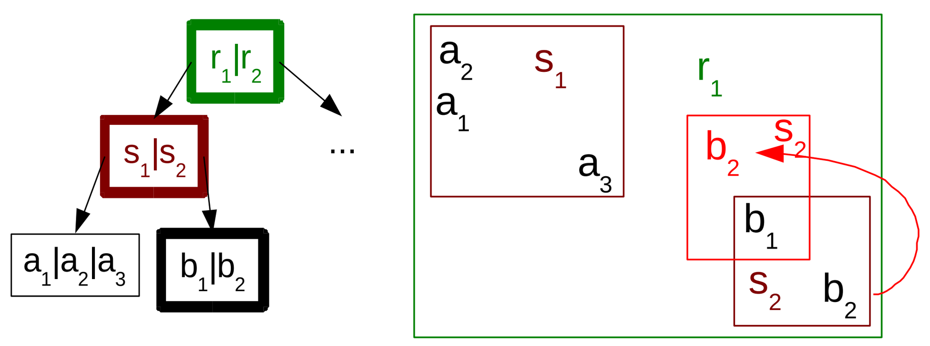


R-Tree Strength: Range Query



R-Tree Weakness: Update

- Let's update position of b2
  - **Delete** the old b_2: 2 traversals!
  - **Insert** the new b_2: 2 traversals!



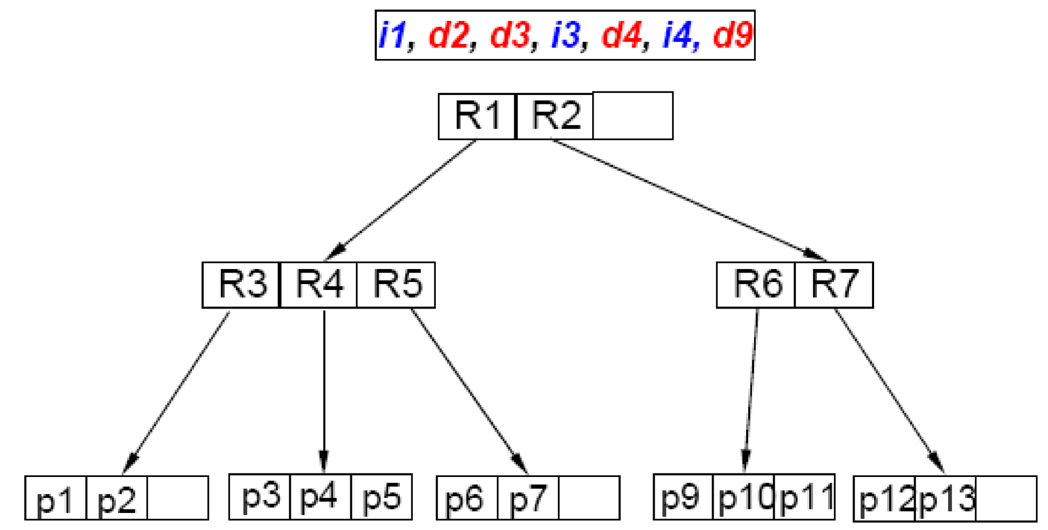
- 1 traversal = 3 I/Os
- 1 update = 12 I/Os!
- **Conclusion:** R-tree updates are expensive, can we do better?

Observations

- Several updates to the same leaf cause separate traversals
- Update locality is not exploited
- Main memory is not used
- High rate of updates is required to sustain accuracy

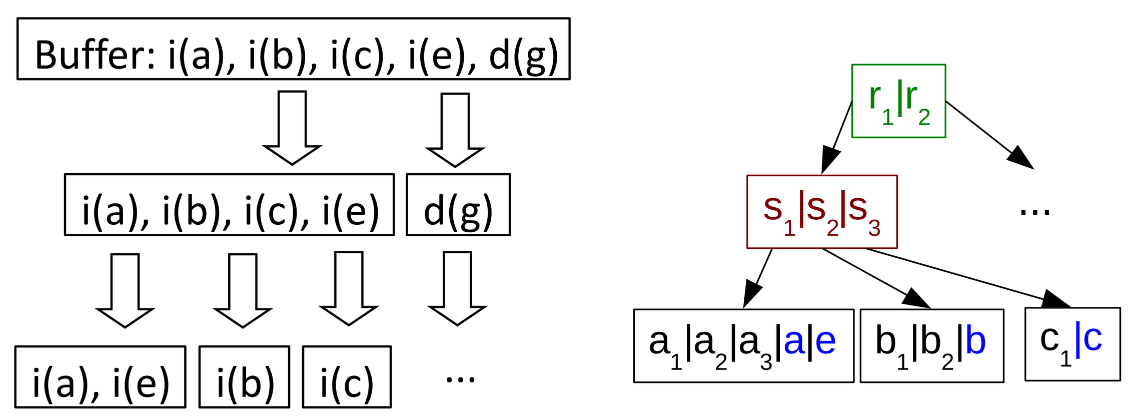
The R^R-tree: the Data Structure

- A disk R-tree without any data structure modifications
- A single buffer for all incoming updates
  - Any amount of memory, the more, the better
  - Organized as a main-memory R-tree (can be accessed as a list too)
  - Data contains the flag to tell deletions from insertions

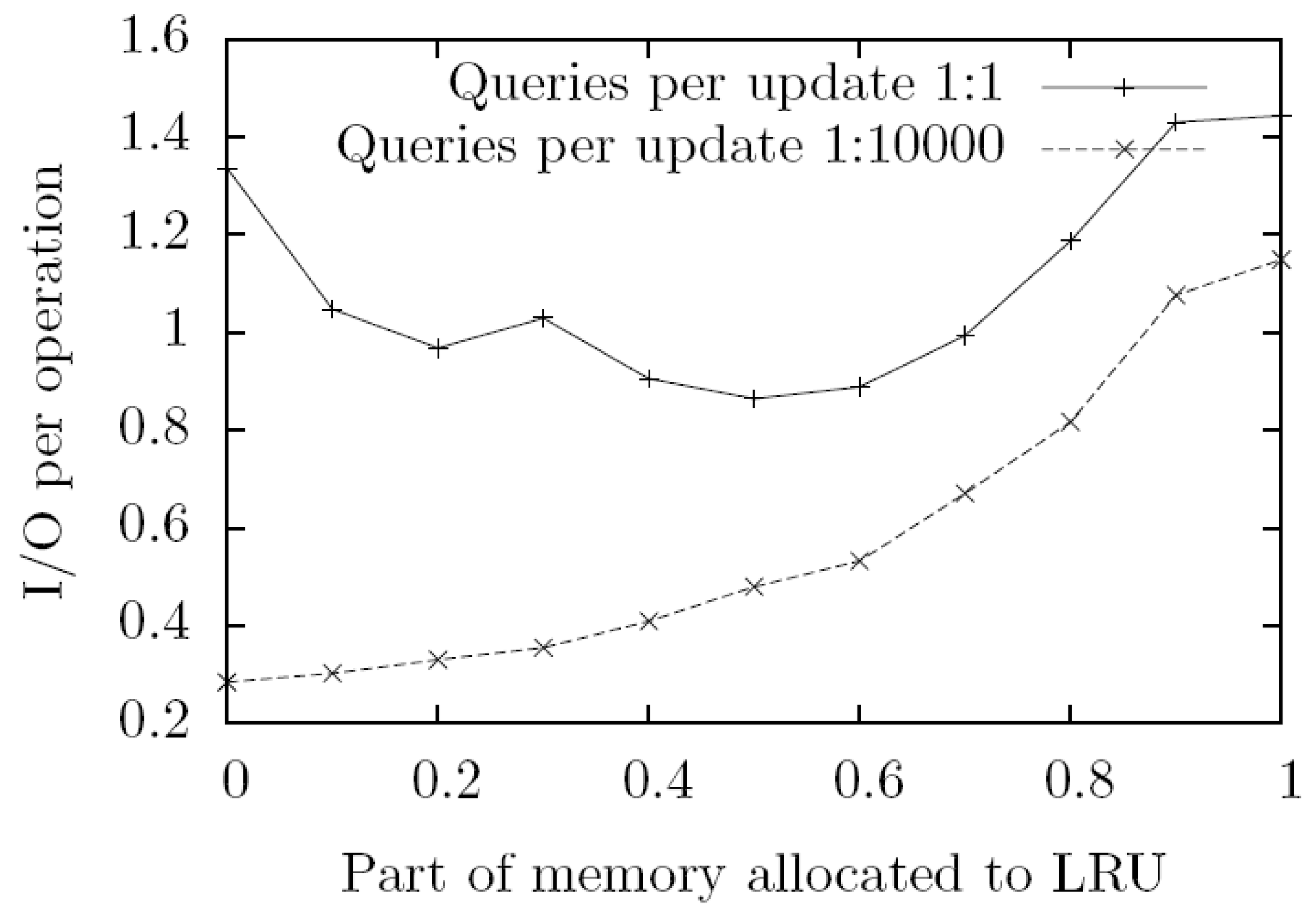


The R^R-tree: Buffer Emptying

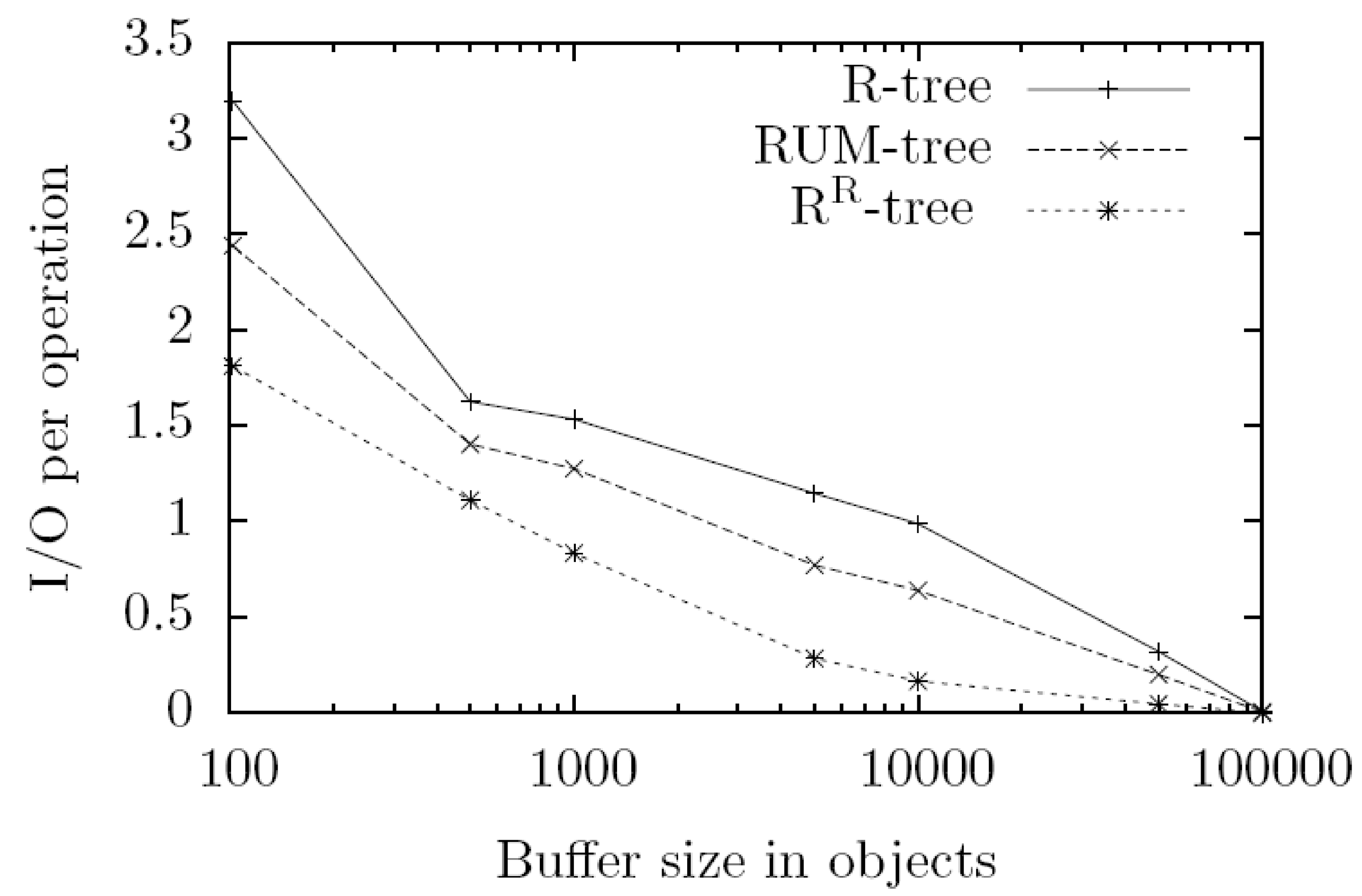
- When the buffer gets full, it is processed on the main tree
- Updates travel down the tree, sharing I/Os
- Small groups should be filtered



Experiments: Main-Memory Utilisation



Experiments: Index Comparison



Summary

- Presented a new main-memory buffering technique for R-tree type indices
- The general idea is to speed up updates by allowing these to share I/O.
- Uses partial buffer emptying
- Empirical studies show that the proposal improves on existing proposals.
- See the paper for the analytical study
- Future work
  - Application to other types of indices
  - Better main-memory indexing
  - Exploring the query performance/update performance trade-off