Indexing Moving Objects In Main Memory

Donatas Saulys, Christian Christiansen, Jan Johansen

Outline

Introduction

- Strongly related to many DB course articles

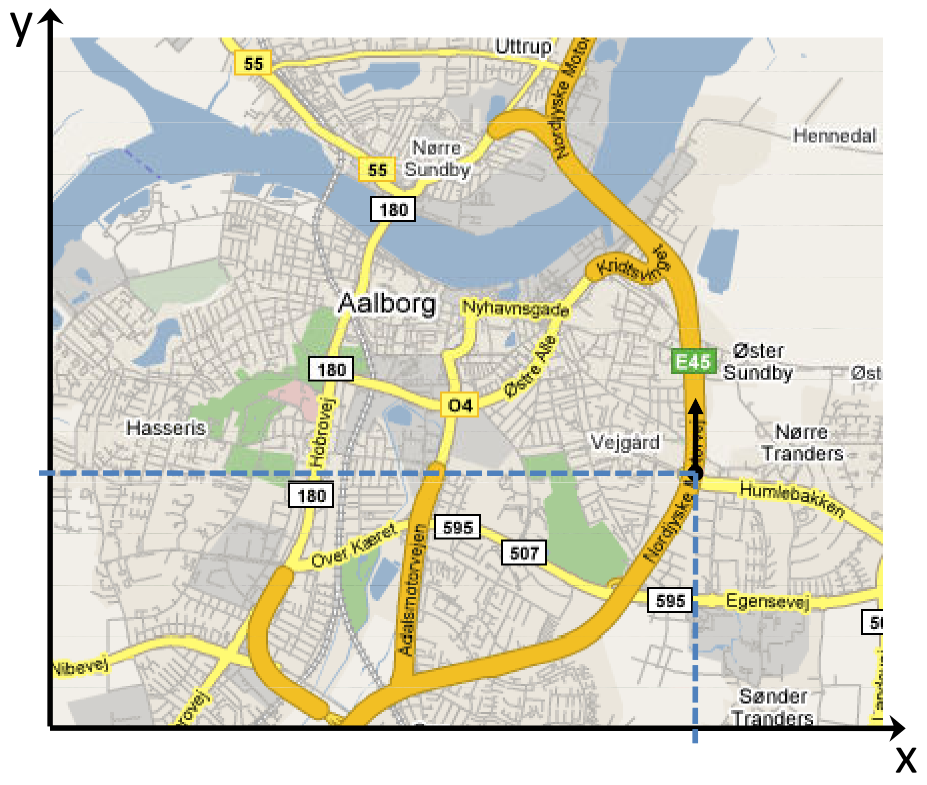
- Indexing of moving objects
- Tree structures
- Main memory

Motivation

- GPS tracking gaining popularity
  - Insurance companies
    - Car theft
    - Speeding
  - Governments
    - Road pricing

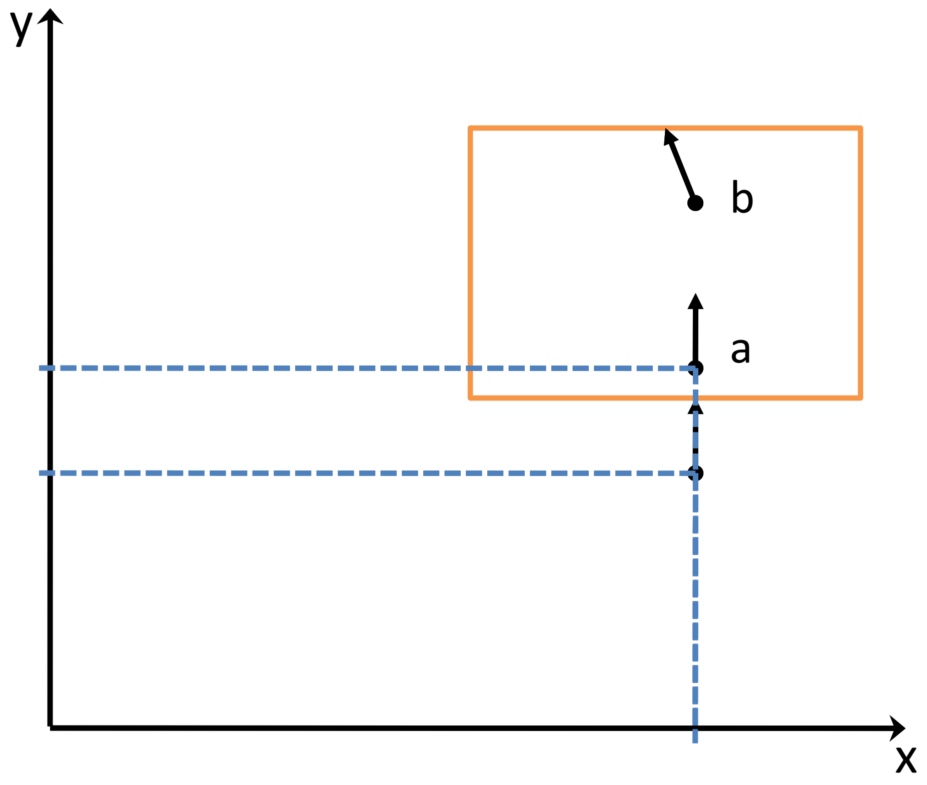
Problem setting

- Monitoring moving objects.
  - Monitored area.
  - Roads on the map.
  - Road network.
  - Cars moving on the road.
  - Monitored object in the system.



Problem setting

- Monitoring moving objects.
  - Object motion.
  - Range query.



Problem Setting

- Large number of moving objects
  - E.g. cars, users of wireless devices etc.
- Support fast queries
  - E.g. which objects are located withing some area?
- High number of updates
  - Maintaining precise location information for many moving objects

Solution overview

Solution Overview

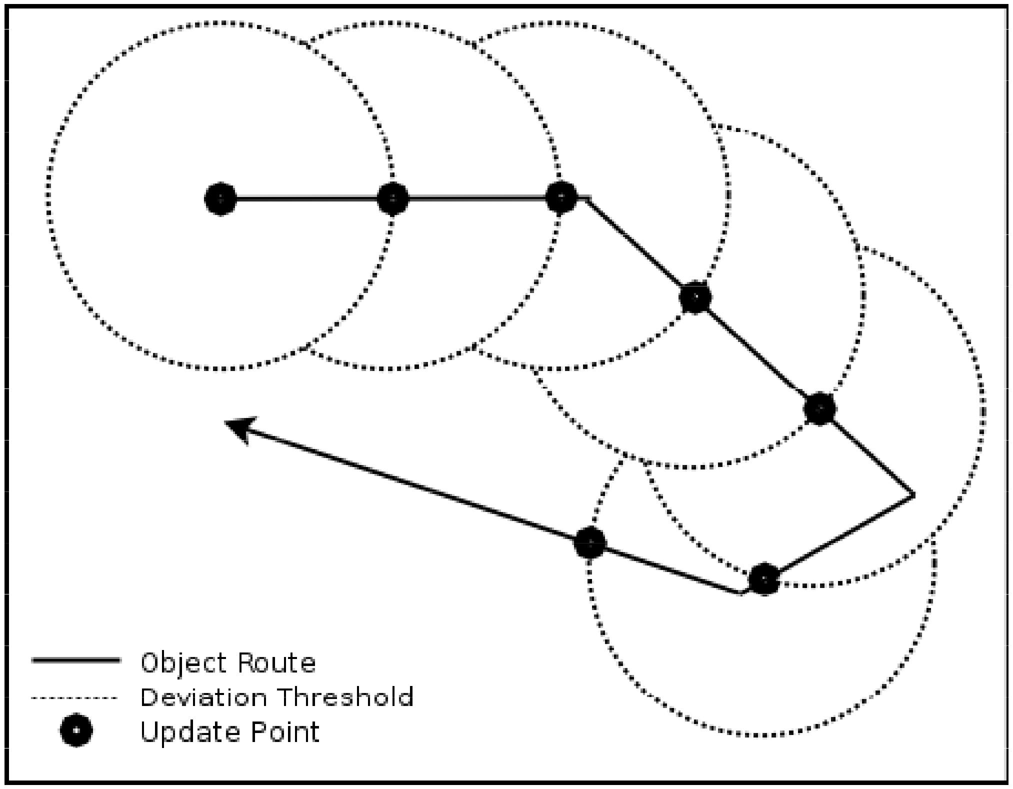
- Efficient moving object tracking
- Data partitioning indexes
  - One Dimensional (e.g. B+-Tree)
  - Multi Dimensional (e.g. R-Tree)
- Space partitioning indexes
  - Grid
  - Quad-trees
- Optimizations
  - A Bottom-up approach
  - Cache-conciousness

Solution Overview

- Efficient moving object tracking
- General case
  - Objects send updates every X time unit (e.g. 1sec)
  - High bandwidth usage
  - Large amounts of updates on the Index structure
- Shared-prediction updates
  - Object's position is predicted both on the client and on the server
  - An update is issued only when the object's predicted position deviates by some threshold value Δd from its real position
  - Lower bandwidth usage
  - Fewer updates on the Index structure

Solution Overview

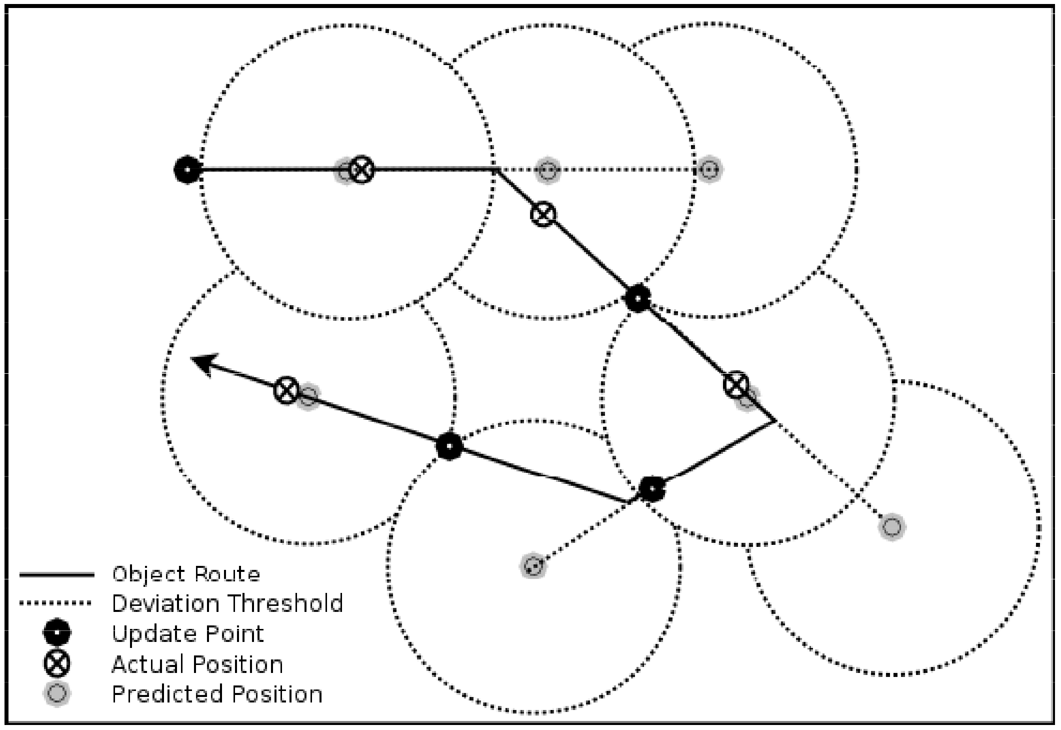
- Shared-prediction updates
  - Point-based tracking



- Object’s predicted position is its last reported position
- When the object is more than some Δd from its last reported position, it triggers an update
- Guarantee of maximum Δd deviation

Solution Overview

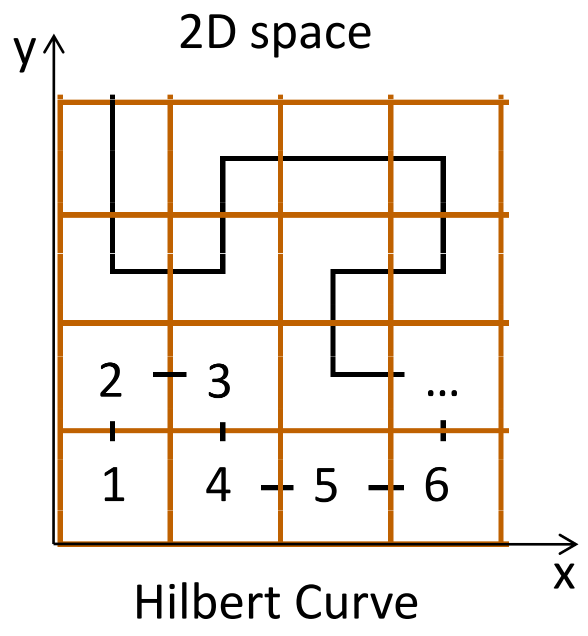
- Shared-prediction updates
  - Vector-based tracking



- Object's predicted positions are given by a linear function of time, i.e., by a start position and a velocity vector
- When the object is more than some Δd from its predicted position it triggers an update
- Guarantee of maximum Δd deviation

Solution Overview

- Data partitioning indexes
  - One Dimensional
- Idea - mapping 2D to 1D
- Proximity is preserved
- Use an existing 1D index (e.g. B+-Tree)



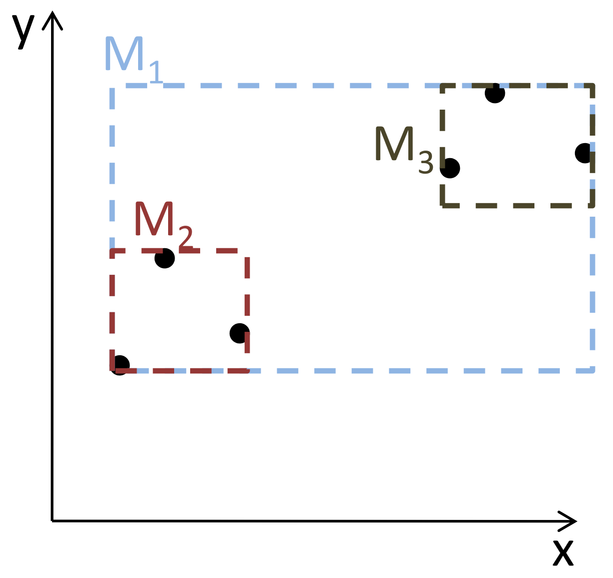
Hilbert Curve

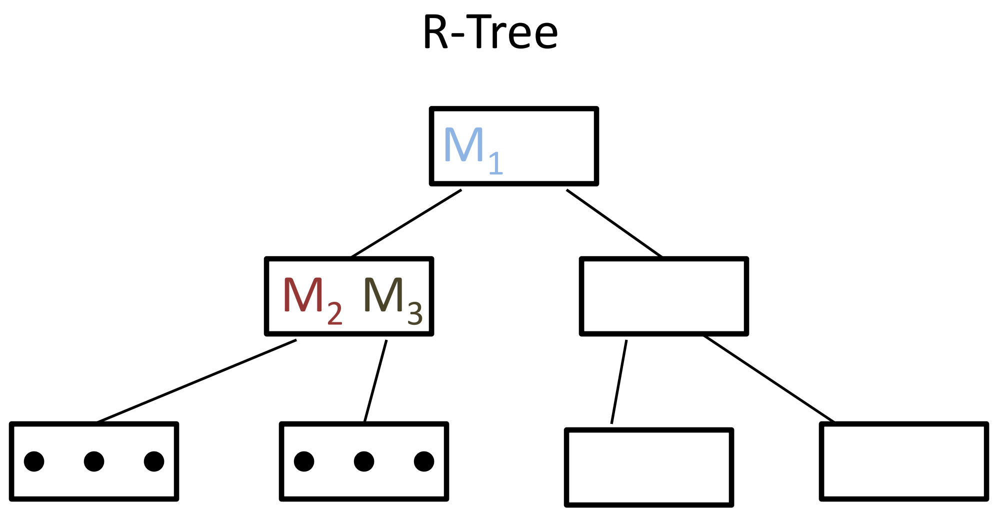
Solution Overview

- Examples
- Bx-tree
  - Supports PB and VB tracking
  - Efficient and robust
  - Fast update performance
  - Good query performance
  - Poor predictive query support
- Bdual-tree
  - Supports PB and VB tracking
  - Uses duality transformation to store linear functions
  - Good predictive query support
  - Outperforms the Bx-tree in updates and queries

Solution Overview

- Data partitioning indexes
  - Multi Dimensional (e.g. R-Tree)
- Index keys - MBRs



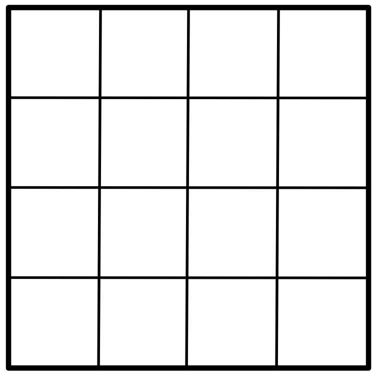


Solution Overview

- Examples
- R-tree
  - Supports PB tracking
  - Used for indexing static spatial data
- R+-tree
  - Avoids overlapping rectangles in intermediate nodes
  - Up to 50% savings in disk accesses compared to an R-tree when querying
- TPR-tree
  - Well suited for indexing moving objects
  - Supports PB and VB tracking
  - Supports predictive queries
- TPR*-tree
  - An Optimized TPR-tree

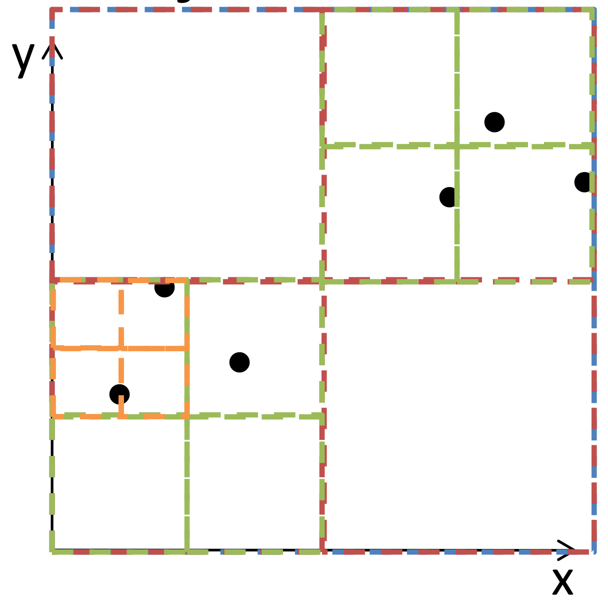
Solution Overview

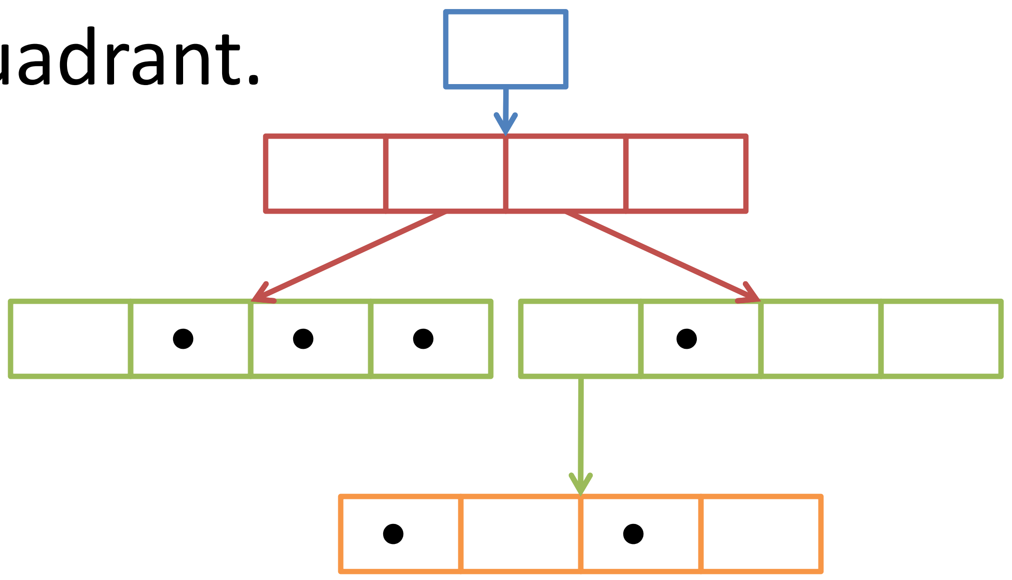
- Space partitioning indexes
  - Grid
- Predefined area partitioned into cells
  - Memory allocated for the whole area
- Direct access to the cell
  - Cell Memory address from coordinates
- Fast updates
- Query performance depends a lot on the data distribution and the cell size
- Poor nearest neighbor query support
- Space inefficient


Grid

Solution Overview

- Space partitioning indexes
  - Quad-trees
- Dynamic partitioning based to the number of objects in each quadrant.



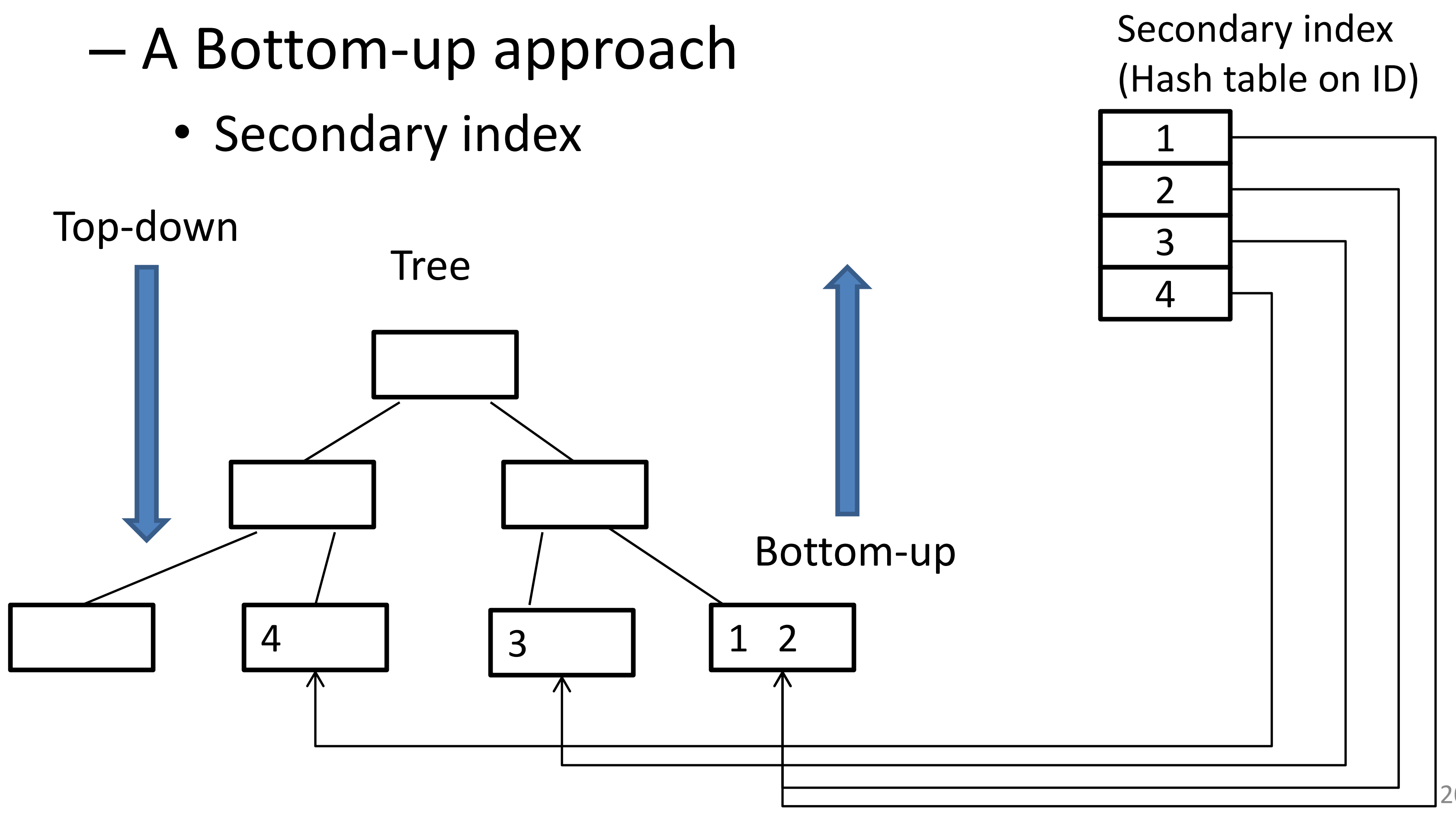


Solution Overview

- Examples
- Bucket PR-quadtree
  - Supports fanout on the lowest level, which reduces tree height.
- XBR-tree
  - Supports indexing moving objects and performing range queries about the history of object trajectories.
- Oct-tree
  - An index structure for indexing 3D objects

Solution Overview

- Optimizations
  - A Bottom-up approach
    - Secondary index



Solution Overview

- Optimizations
  - Cache-conciousness
- Data from MM to the CPU cache is loaded in blocks
  - Block size = cache line size
- Idea:
  - Put as much index data as possible into the size of a block
  - E.G. Hold more index entries in a node of a Tree without increasing the size of the node (fixed node size = cache line size)
- Examples
  - CR-tree
  - CSB+-tree

Project goals

Project Goal

- A study of ways to index moving objects in main memory.
  - Our contribution: An overview of the existing solutions.
  - Benchmark testing of index structures implemented in main memory.
  - In tests we focus on performance in the areas of updating, querying and space consumption.

Semester Goal

- Get an overview of the problem of indexing moving objects in main memory and the existing solutions.
- Implementing an R-tree in main memory.
- Determine the performance improvement of using bottom-up updating.
- Compare the performance with a simple grid structure.

Next Semesters Goal

- Choosing and implementing the optimal B-trees, R-trees, Quad-trees and Grids in main memory.
- Conducting performance study comparing all the implemented index structures.

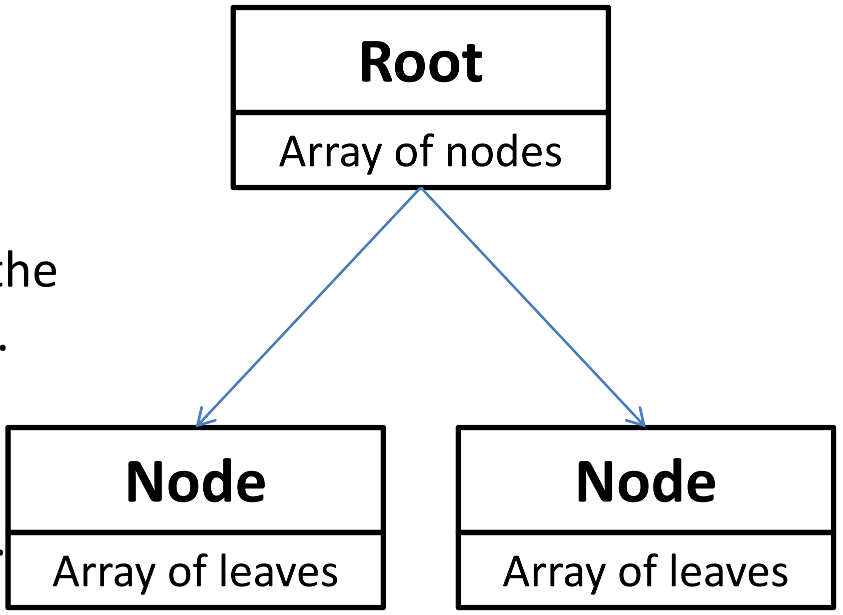
Project status

Status : Implementation

- Two R-tree variations implemented:
  - Basic r-tree
    - Data contained in tree.
  - Pointer-based r-tree
    - Method input is generally passed as pointers to the actual input.

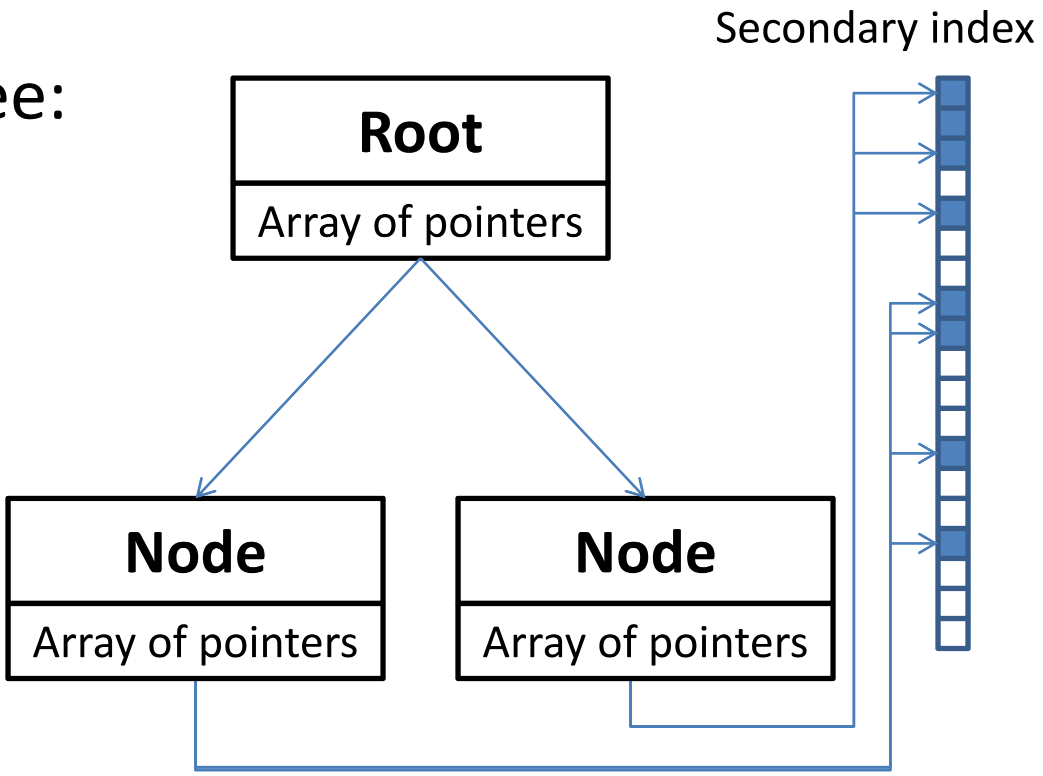
Status : Implementation

- The basic r-tree:
  - MBRs and points
    - Internal nodes store MBRs with pointers to the child nodes in the array.
    - Leaf nodes store point coordinates and other object data in the array.
  - Splitting and merging nodes



Status : Implementation

- The pointerbased r-tree:
  - MBRs and points.
    - MBRs are stored in the objects to which they belong.
    - Pointers to children are stored in the arrays.
    - Objects are stored in a secondary index.
  - Splitting and merging nodes



Status : Report

- What goes into the report?
  - Introducing the subject
  - Defining our problem
  - Exploring solutions
  - Choosing implementations
  - Conducting experiments
  - Concluding on results

Topic Questions

1. Scope of the project
2. Other directions for the project
3. Does any of the index introductions need more elaboration or do we go too much into detail in some parts?