Spatio-Temporal Databases

Outline

- Spatial Databases
- Temporal Databases
- Spatio-temporal Databases
- Multimedia Databases
- .....

Introduction

- Spatio-temporal Databases: manage spatial data whose geometry changes over time
- Geometry: position and/or extent

Examples:

  - Global change data: climate or land cover changes
  - Transportation: cars, airplanes
  - Animated movies/video DBs

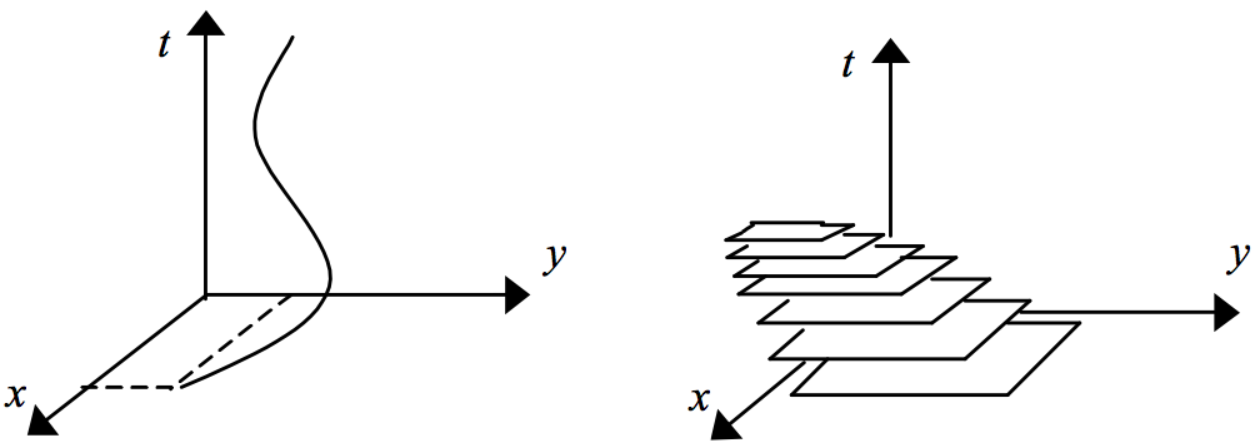
ST DBs

- A special Temporal Database
  - All the features of temporal database
  - Attributes can be spatial also
- Extension of Spatial Databases
  - Objects change instead of being static
  - At any timestamp it is a conventional Spatial Database
- New Database type

Requirements

- Efficient Representation of Space and Time
- Data Models
- Query Languages
- Query processing and Indexing
- GUI for spatio-temporal datasets

Spatio-temporal Objects



(a)
a moving point

(b)
a moving and shrinking region

ST Queries

- *Selection* Queries: “find all objects contained in a given area *Q* at a given time *t*”
- *NN* queries: “find which object became the closest to a given point *s* during time interval *T*,”
- *Aggregate* queries: “find how many objects passed through area *Q* during time interval *T*,” or, “find the fastest object that will pass through area *Q* in the next 5 minutes from now”

ST Queries

- *join* queries: “given two spatiotemporal relations *R*1 and *R*2, find pairs of objects whose extents intersected during the time interval *T*,” or “find pairs of planes that will come closer than 1 mile in the next 5 minutes”
- *similarity* queries: “find objects that moved similarly to the movement of a given object *o* over an interval *T*”

ST Data Types

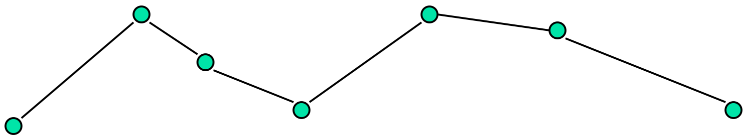
- Moving Points
  - Extent does not matter
  - Each object is modeled as a point (moving vehicles in a GIS based transportation system)
- Moving regions
  - Extent matters!
  - Each object is represented by an MBR, the MBR can change as the object move (airplanes, storm,…)

SP Data Types

- Different Type of changes:
  - Changes are applied discretely
    - Urban planning: appearance or dis-appearance of buildings
  - Changes are applied continuously
    - Moving objects (eg. Vehicles)

Trajectories

- Moving objects create trajectories
- Usually we can sample the positions of the objects at periodic time intervals Δt
- Linear Interpolation:easy and usually accurate enough
- Trajectory: a sequence of 2 or 3-dim locations

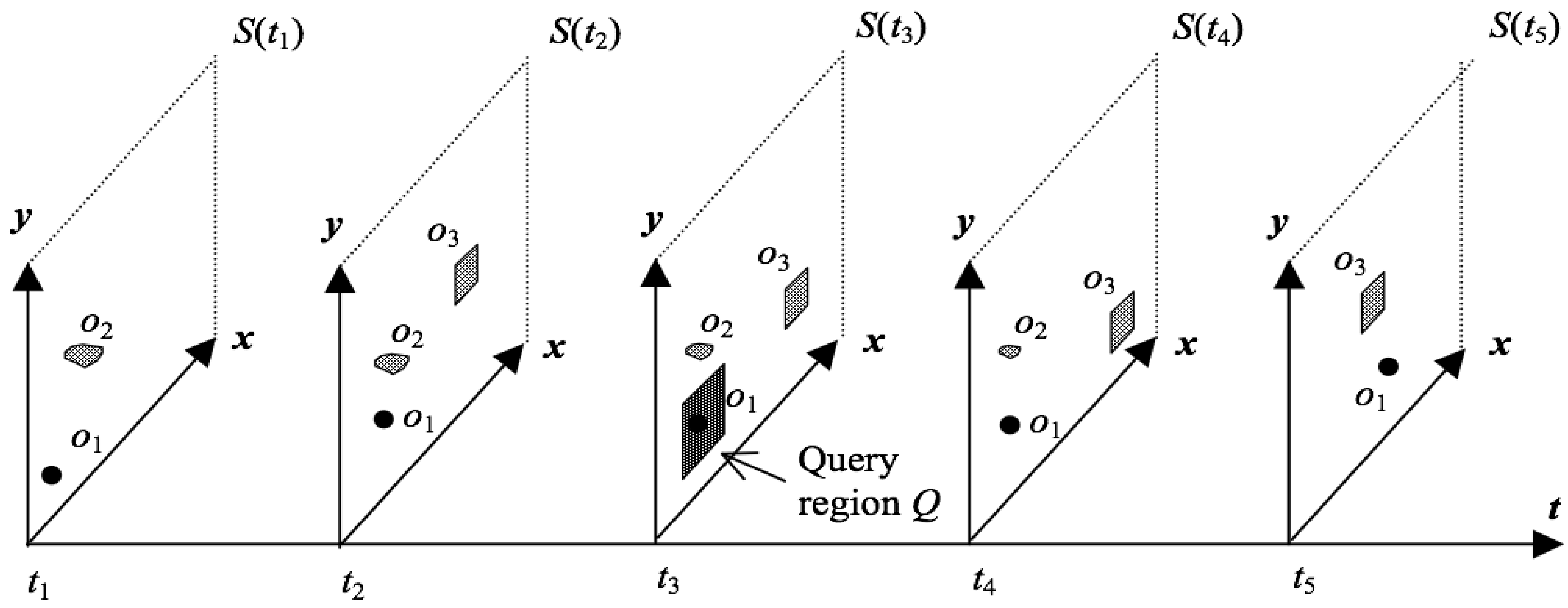


Temporal Environment

- Transaction or Valid time: (usually we assume transaction time)
- Two types of environments:
  - Predicting the future (or current) positions: Each object has a velocity vector. The DB can predict the location at any time $t \geq t_{now}$ assuming linear movement. Queries refer to the future
  - Storing the history. Queries refer to the past states of the spatial database

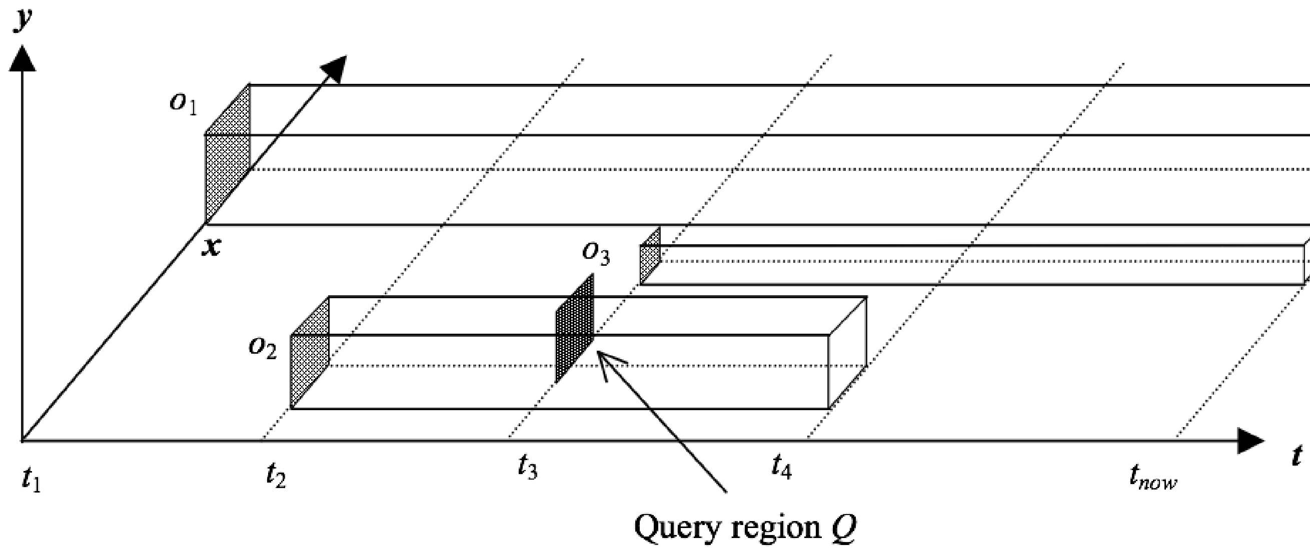
The Historical Environment

- Spatio-temporal Evolution



Indexing using R-trees

- Assume that time is another dimension, use a 3D R-tree
- Store the objects as their 3D MBR. How to compute that?



Problems of 3D R-tree

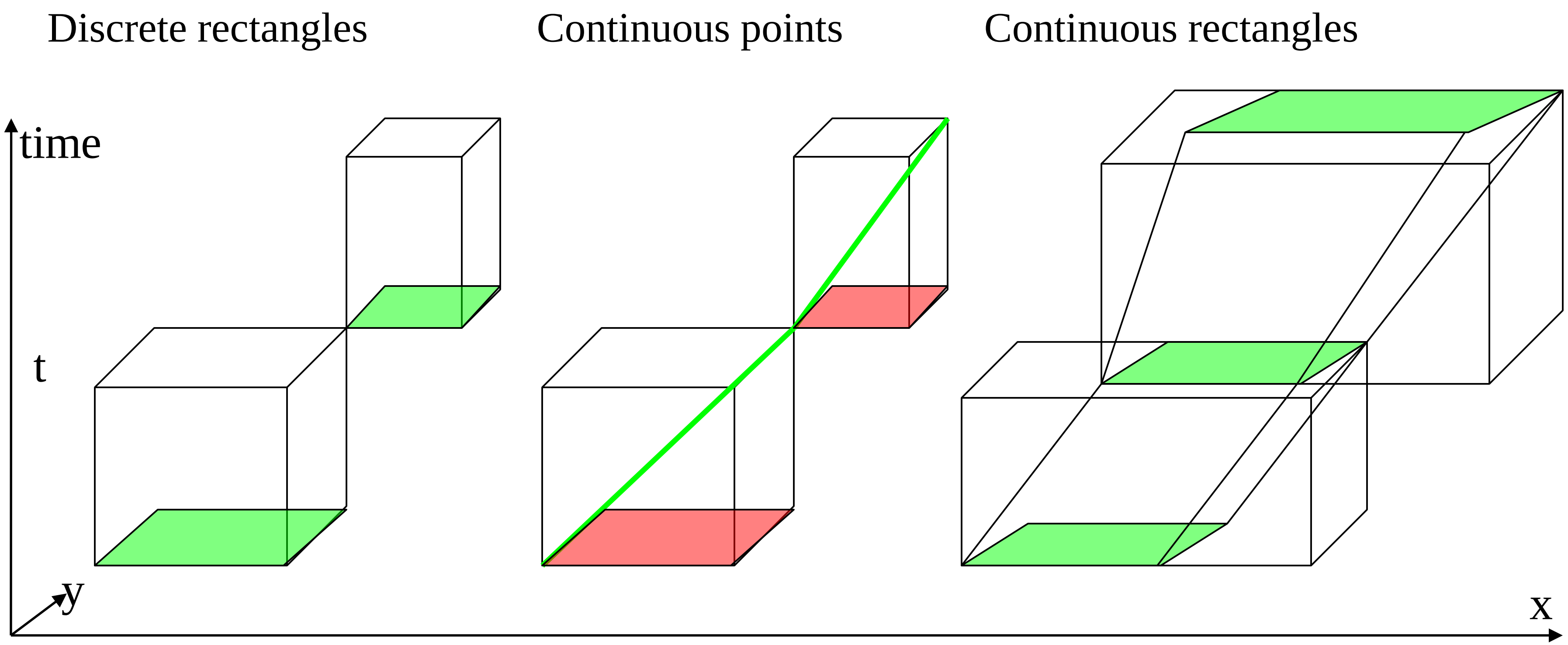
- How to store "now"? Use a large value…
- Common ending problem
- Long lived objects will have very long MBRs, difficult to cluster
- Extensive overlap and empty space → bad query performance for specific queries
- Also, works only for discrete changes

PPR-tree

- Better idea, partially persistent R-tree
- Two approaches: Multiversion and overlapping
- Multi-version: use the idea of the MVBT applied to R-tree.

Indexing Moving Objects

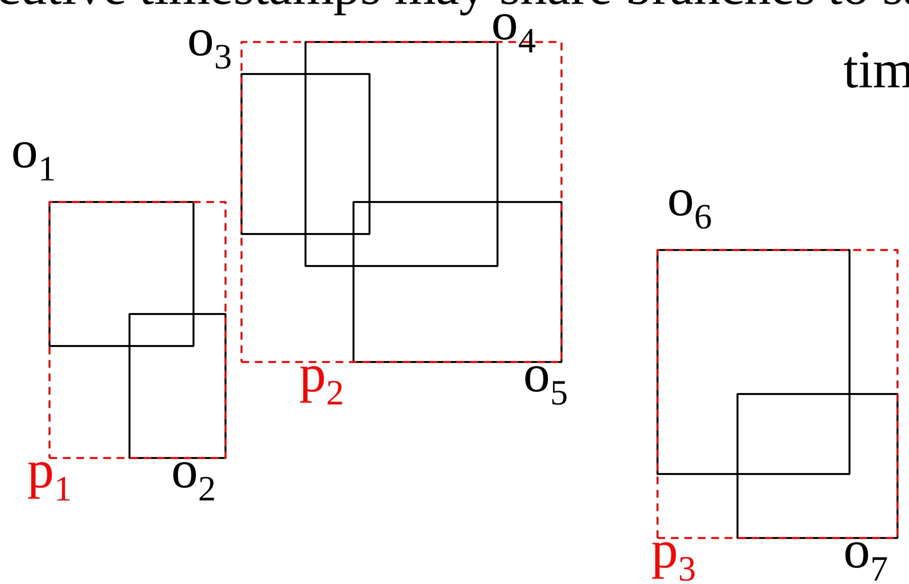
- The problem of indexing any type of moving objects can be reduced to indexing discrete rectangles.



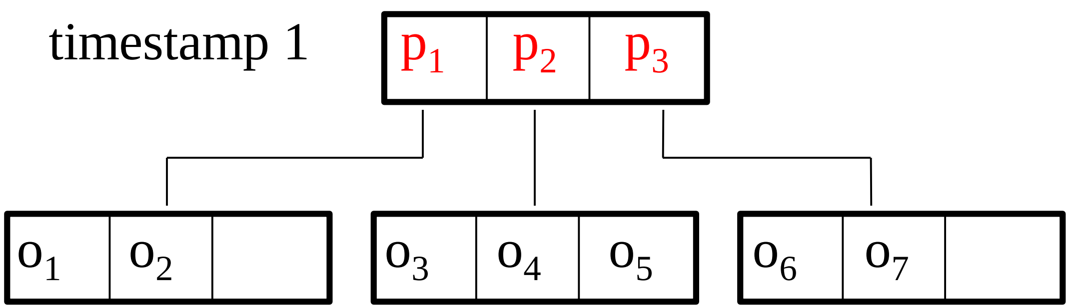
Historical R-trees (HR-trees)

- An R-tree is maintained for each timestamp in history.
- Trees at consecutive timestamps may share branches to save space.

timestamp 1

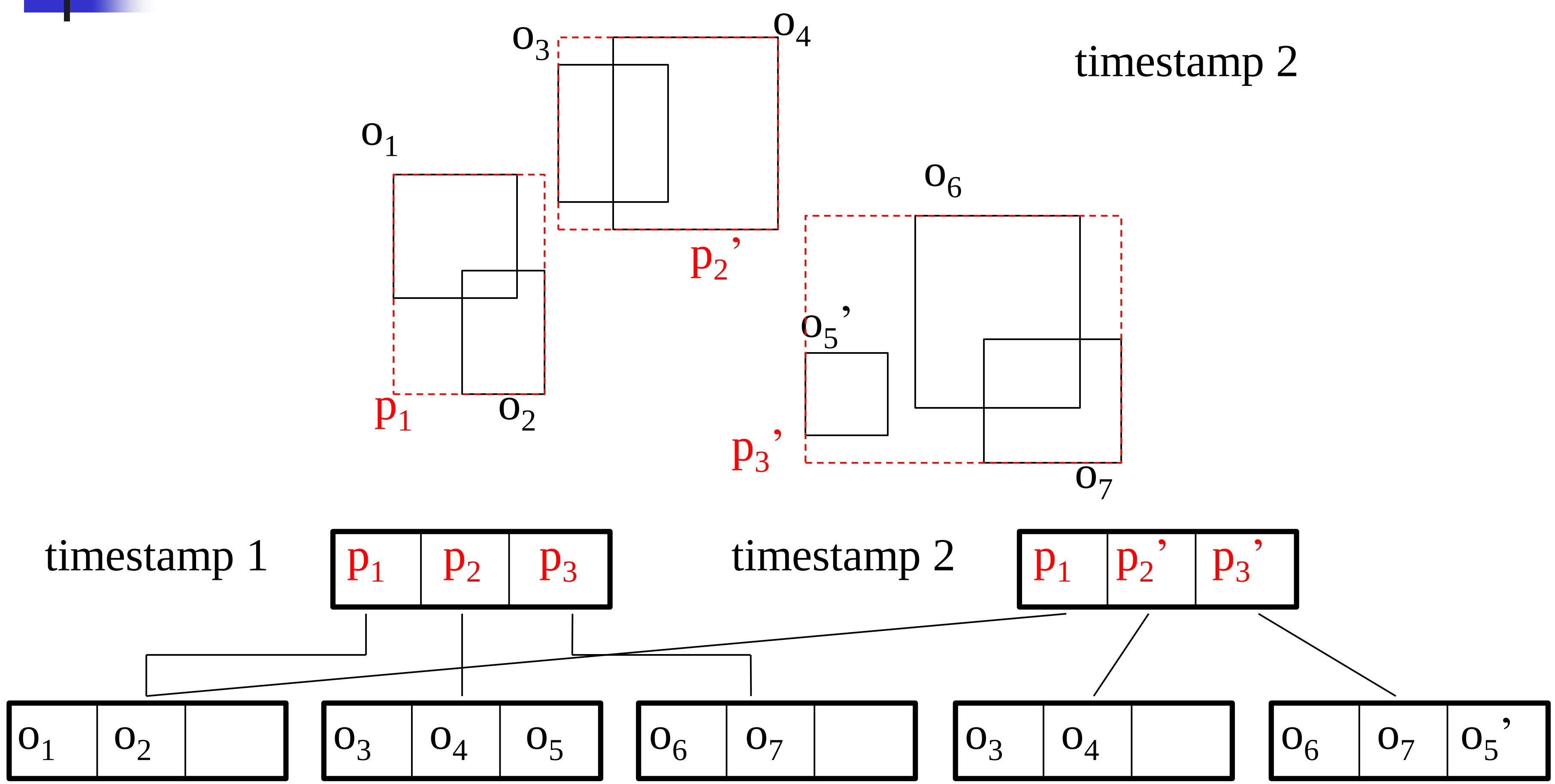


timestamp 1



Historical R-trees

- An R-tree is maintained for each timestamp in history.
- Trees at consecutive timestamps may share branches to save space.



HR-trees: Pros and Cons

- HR-trees answer timestamp queries very efficiently.
  - A timestamp query degenerates into a spatial window query handled by the corresponding R-tree at the query timestamp.

- Not quite efficient:
  - Expensive space consumption.

  ⇨ A node needs to be duplicated even when only one object moves.

  - Interval query processing is inefficient.

  ⇨ Although redundancy (from duplication) is necessary to maintain good timestamp query performance, it is excessive in HR-trees.

MVR-tree

- Consider a 2D R-tree that evolves over time
- Use the Multiversion B-tree approach to store the evolution of the tree…
- Need to consider some “spatial” issues:
  - No unique siblings, split methods, copies due to time split
- To insert a new object, compute a bounding box that encloses the object at all time instants, insert this bb as MBR

MVR-Tree

- An update or a query at some time instant t searches only among the spatial objects that are alive at t
- Space is linear to the number of updates, the problem of “now” is avoided
- Very efficient for snapshot or small interval queries

An improvement: The MV3R-tree [Tao & Papadias 01]

- The MV3R-tree consists of a multi-version R-tree (MVR-tree) and an auxiliary 3D R-tree built on the leaves of the MVR-tree.

- The MVR-tree is optimized from the original multi-version framework, taking into account spatial properties.

Redundancy in MVR-trees

- As with HR-trees, multi-version structures also aim at optimizing timestamp queries.
  - To achieve this goal, the original multi-version framework still leads to considerable redundancy, though much less than HR-trees.

- Excessive redundancy is harmful.
  - Space consumption is increased.
  - Compromise interval query performance, as multiple copies of the same object need to be retrieved.

Optimizing MVR-trees: Reducing Redundancy

- There are some heuristics to reduce data redundancy in MVR-trees in order to lower the space consumption and improve interval query performance.
  - Insertion
    1. General Key Split
    2. Insert in node after reinserting one of the entries
    3. Insert in another node
    4. Version split

- The resulting trees contain much less redundancy
  - Lower tree sizes and accelerate interval queries.
  - Timestamp queries are only slightly compromised.

Interval Query with Multi-Version Structures

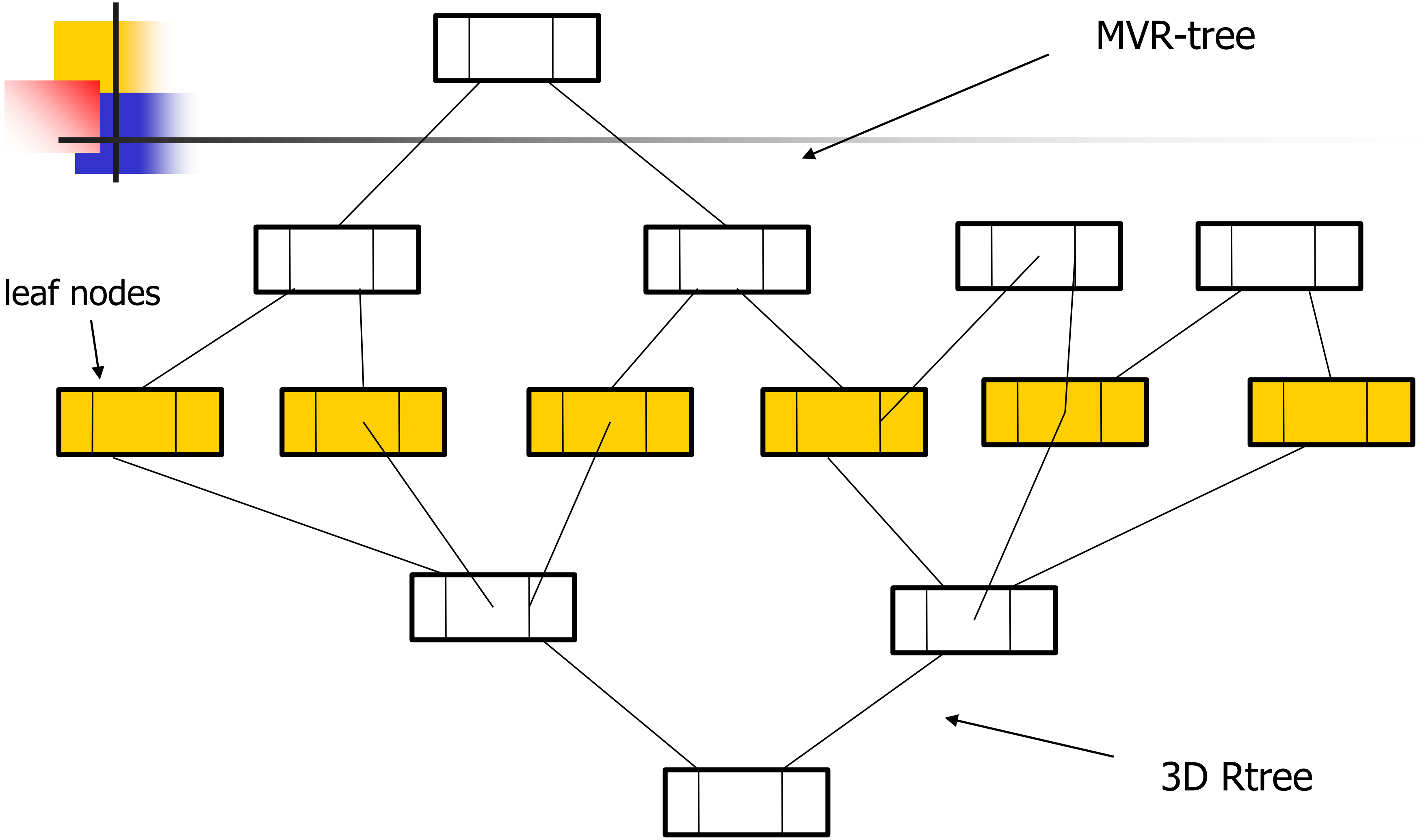
- Need to search several logical trees responsible for the queried timestamps.

- Since multiple parent entries may point to the same child node, it is imperative to avoid duplicate accesses to the child.
    1. If a node is re-visited, the entire subtree rooted at the node needs to be re-visited.

Solution? Keep a hash table, other solutions exist also.

Building a 3D R-tree on the Leaves of the MVR-tree

- Build a 3D R-tree on the leaf nodes of the MVR-tree
- The size of the 3D R-tree is much smaller than a complete 3D R-tree as the number of leaf nodes is significantly lower than the number of actual objects.
- Since the 3D R-tree is not an acylic graph but a tree, we do not have duplicate visit problem.
- Long interval queries are processed with the auxiliary 3D R-trees.


MVR-tree
leaf nodes
3D Rtree

What about moving objects?

- Problem: the MBR representation creates large empty space
- Use artificial deletes, approximate the object using many small MBRs
- But then, the space is increased
- Use an algorithm to distribute a small number of splits to the objects that need them most

What about moving objects?

- If objects move with linear functions of time:

  Minimize total volume by splitting in equidistant points
- Given K splits you can decide the best splits in O(KlogN) time.

Reference:
[Tao & Papadias 01]:MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. VLDB 2001: 431-440