# PRINCIPLES OF DATA ORGANISATION

Spatial join for External Memory

# MOTIVATION

- Key, pointer pairs ~ index
- Non-spatial join
- Spatial join in secondary memory
  - We focus only on intersection joins

# HIERARCHICAL TRAVERSAL

- Both datasets must be indexed using a hierarchical index
  - E.g., R-tree
- Synchronized traversal can be used to test the join condition
- Similar to iterative filter and refine approach

# SYNCHRONIZED TRAVERSAL

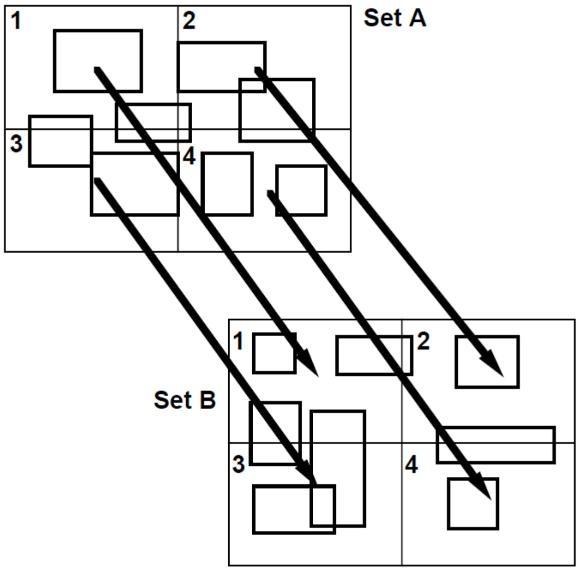
- The algorithm traverses the two trees in a synchronized fashion and compares bounding objects at given levels
- If a node corresponding to a part of the space does not match the condition it can be excluded from the traversal

```
INDEXED_TRAVERSAL_JOIN(rootA, rootB)
INPUT: Roots of the structures representing the sets to be joined
OUTPUT: Pairs of intersecting rectangles

queue ← CreateQueue();
queue.Add(pair(rootA, rootB));
WHILE NOT(queue.Empty()) DO
   nodePair ← queue.Pop();
   pairs ← IdentifyIntersectingPairs(nodePair);
   FOREACH p ∈ pairs DO
      IF p is leaf THEN ReportIntersection(p);
      ELSE queue.Add(p);
```

# PARTITIONING

- Often applied when neither of the sets to be joined is indexed
- The set is partitioned
  - Resulting partitions should be small enough to fit in internal memory
- Once the data are partitioned, each pair of overlapping partitions is read into internal memory and internal memory techniques are used



# PARTITION JOIN OF UNIFORM DATA

```
GRID_JOIN(setA, setB)
INPUT: Sets of objects to be joined
OUTPUT: Pairs of intersecting objects

{ determine the partitions: }
m ← AvailableInternalMemory();
mbrSize ← BytesToStoreMBR();
minNrOfPartitions ← (setA.Size() + setB.size())*mbrSize() / m;
partList ← DeterminePartitions(minNrOfPartitions);

{ object appears in every partition it intersects }
partitionPointersA ← PartitionData(partList, SetA);
partitionPointersB ← PartitionData(partList, SetB);

FOREACH part ∈ partList DO
   partitionA ← ReadPartition(partitionPointersA, part);
   partitionB ← ReadPartition(partitionPointersB, part);
   PLANE_SWEEP(partitionA, partitionB); { or any other algorithm for internal memory }
```

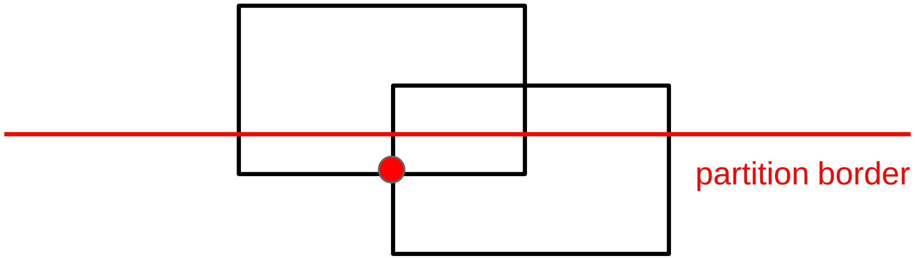
# AVOIDING DUPLICATE RESULTS

**Sort and remove duplicates**

- Requires sorting, which implies increased computational demands
- The duplicities get together

**Reference point method**

- A consistently chosen reference point is selected from the intersecting region.
- Intersection is reported only if the reference point lies within given partition.



# PARTITIONING – SKEWED DISTRIBUTIONS

- Basic grid algorithm is rarely used since the objects distribution is often not uniform
- Patel & DeWitt 1996 proposed to group partitions using a mapping function to minimize skew by creating partitions having similar number of items