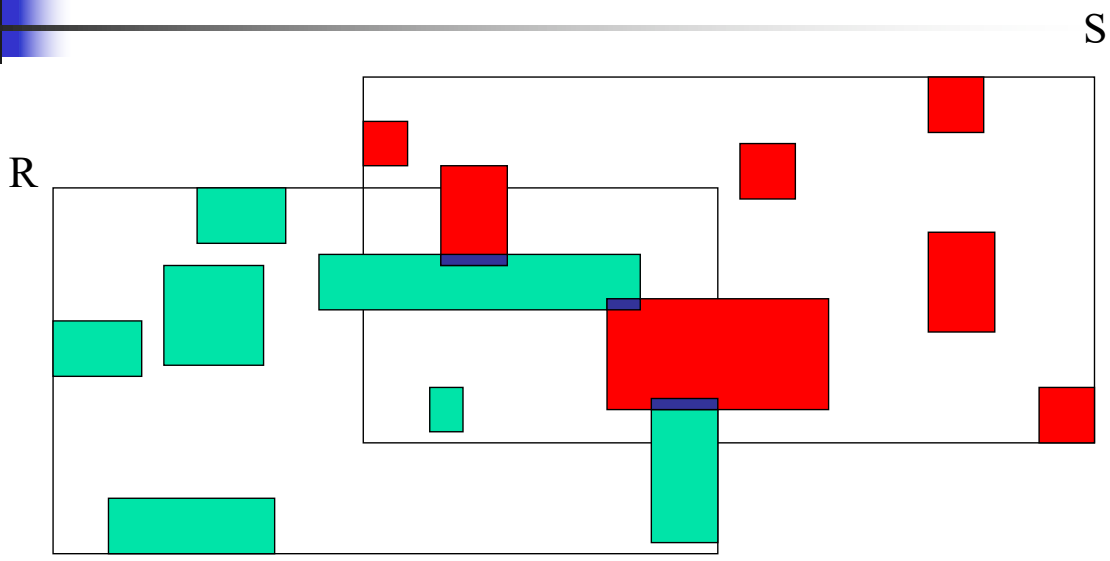
CSEG601 & CSE5601
Spatial Data Management & Application :

Spatial Joins

Sungwon Jung

Big Data Processing and Database Lab.
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

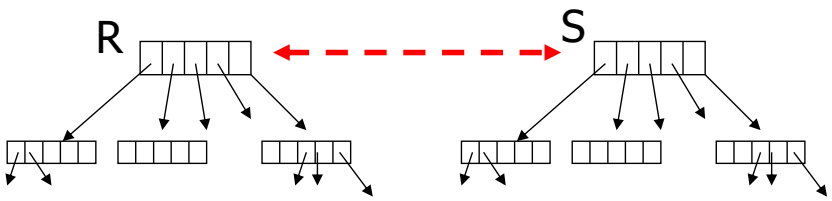
R-tree based Join



Join1(R,S)

- Tree synchronized traversal algorithm

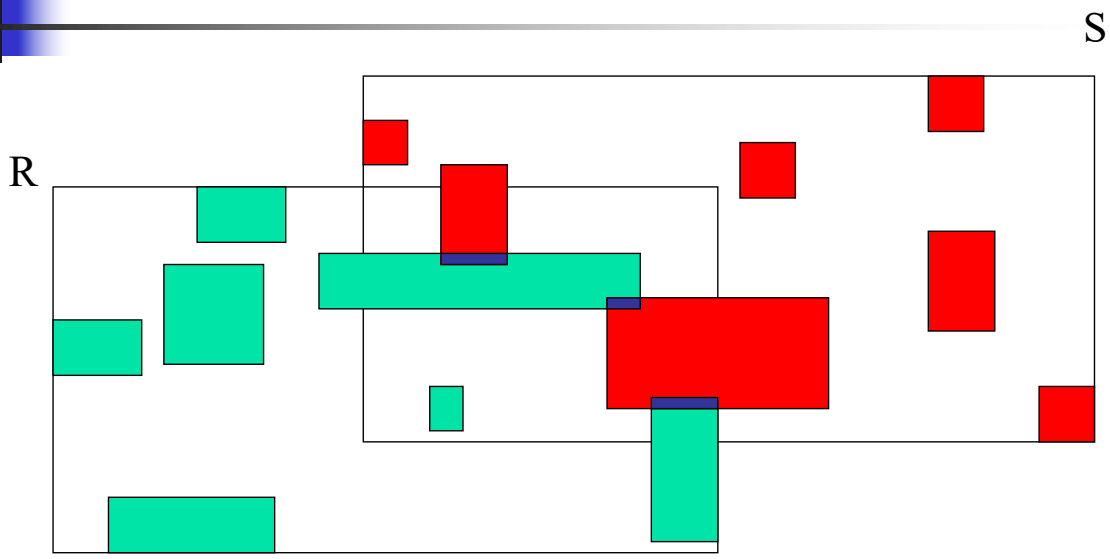
Join1(R,S)
Repeat
Find a pair of intersecting entries E in R and F in S
If R and S are leaf pages then
add (E,F) to result-set
Else Join1(E,F)
Until all pairs are examined



CPU – Time Tuning

- Two ways to improve CPU – time
  - Restricting the search space
  - Spatial sorting and plane sweep

Reducing CPU bottleneck



Join2(R,S,IntersectedVol)

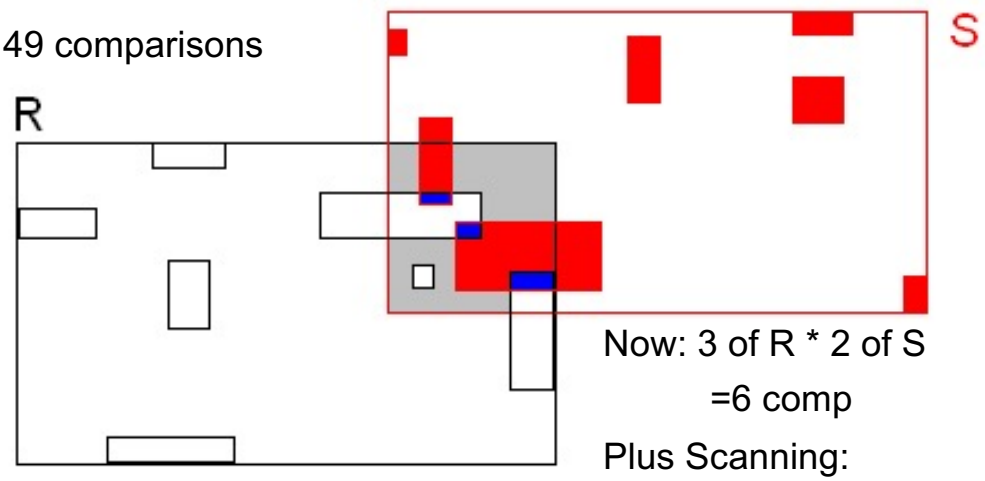
Join2(R,S,IV)
Repeat
Find a pair of intersecting entries E in R and F in S that overlap with IV
If R and S are leaf pages then
add (E,F) to result-set
Else Join2(E,F,CommonEF)
Until all pairs are examined

- In general, number of comparisons equals
  - size(R) + size(S) + relevant(R)*relevant(S)
- Reduce the product term

Restricting the search space

Join1: 7 of R * 7 of S

= 49 comparisons



Now: 3 of R * 2 of S

=6 comp

Plus Scanning:

7 of R + 7 of S

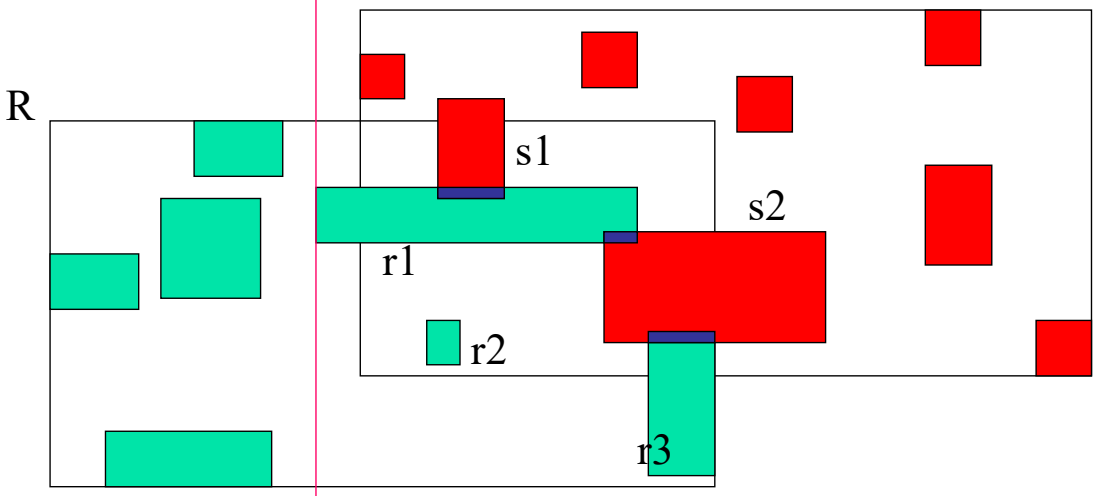
= 14 comp

Using Plane Sweep

Sort the set of rectangles w.r.t. the *x* values of their lower-left corners in the intersected area of R and S

r1	r2	s1	s2	r3

S

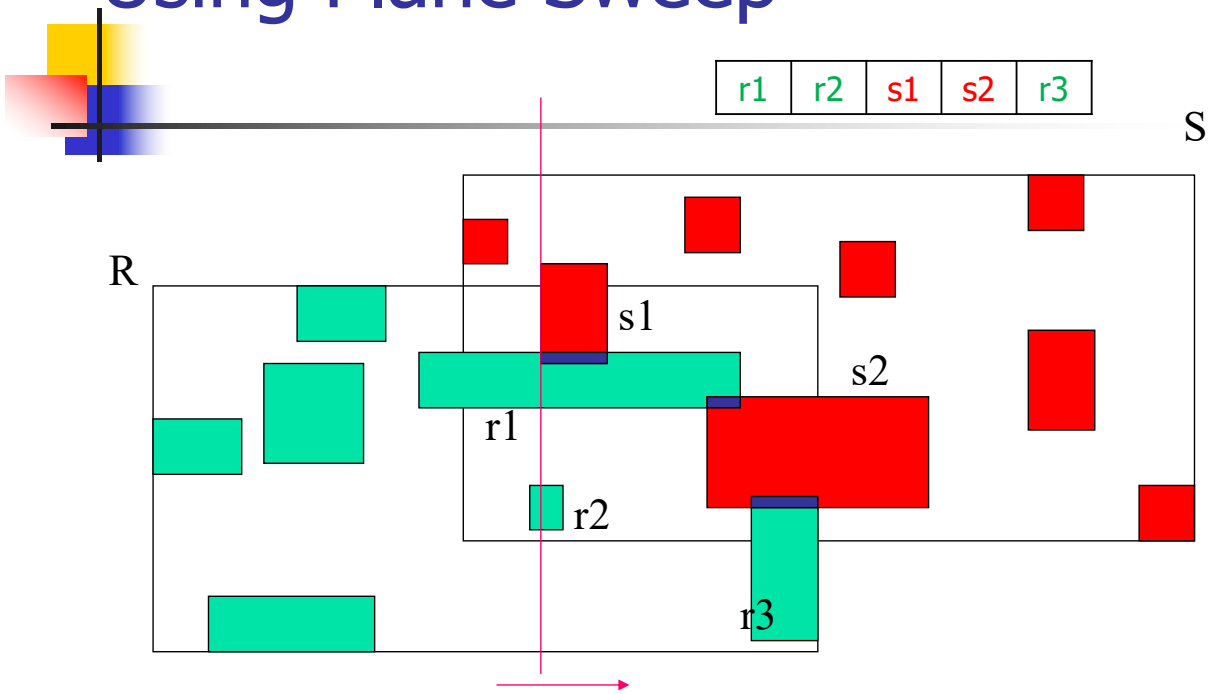


Consider the extents along x-axis

Start with the first entry r1

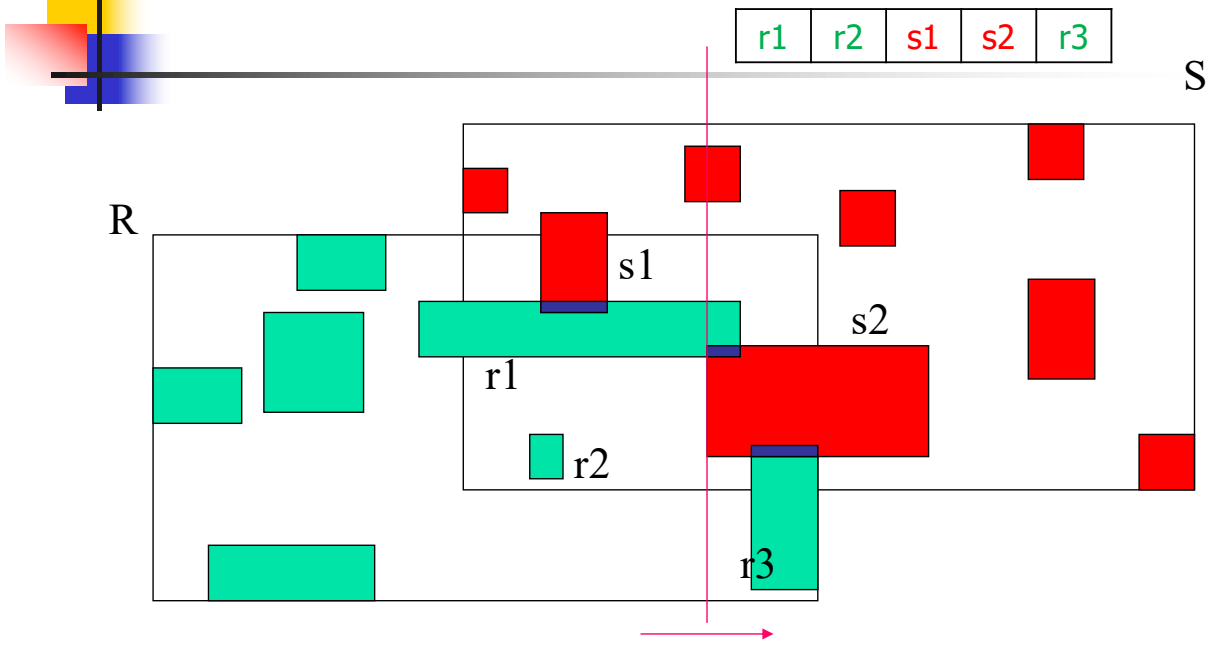
sweep a vertical line

Using Plane Sweep



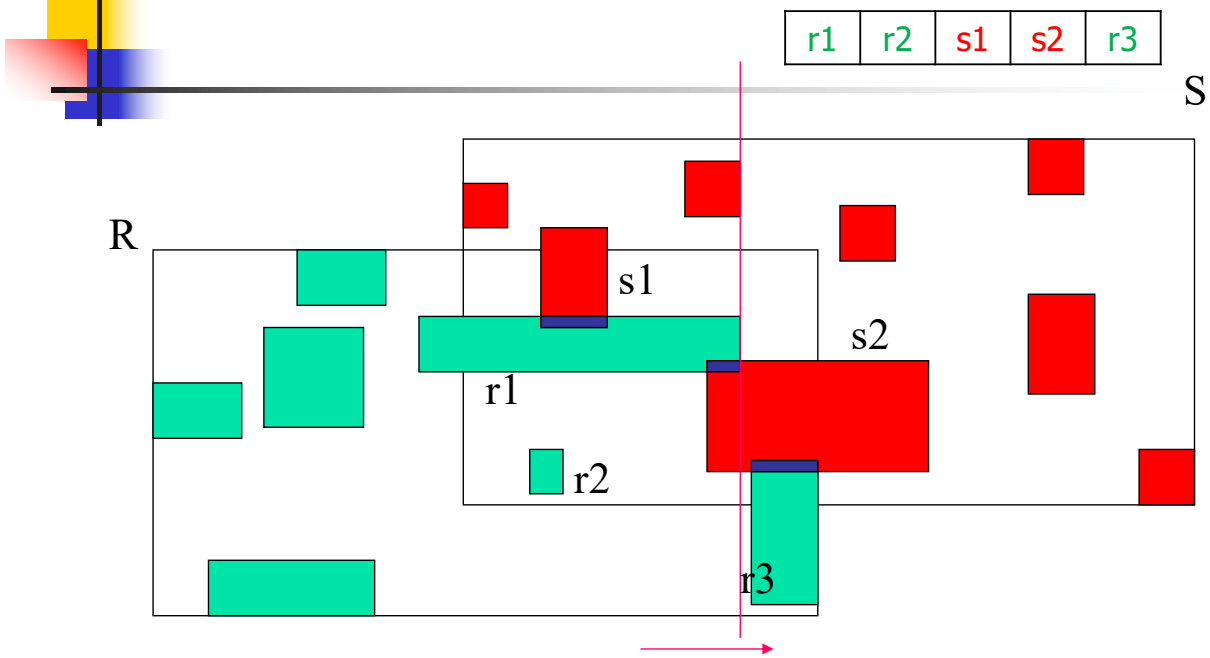
Check if (r1,s1) intersect along y-dimension
Add (r1,s1) to result set

Using Plane Sweep



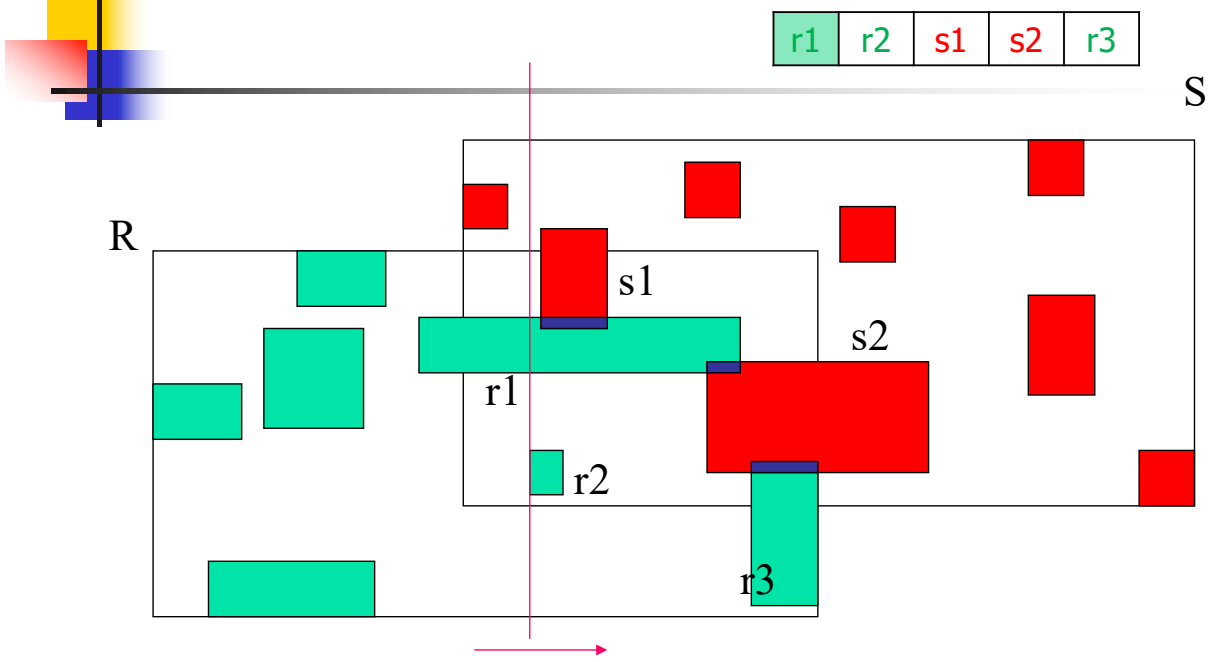
Check if (r1,s2) intersect along y-dimension
Add (r1,s2) to result set

Using Plane Sweep



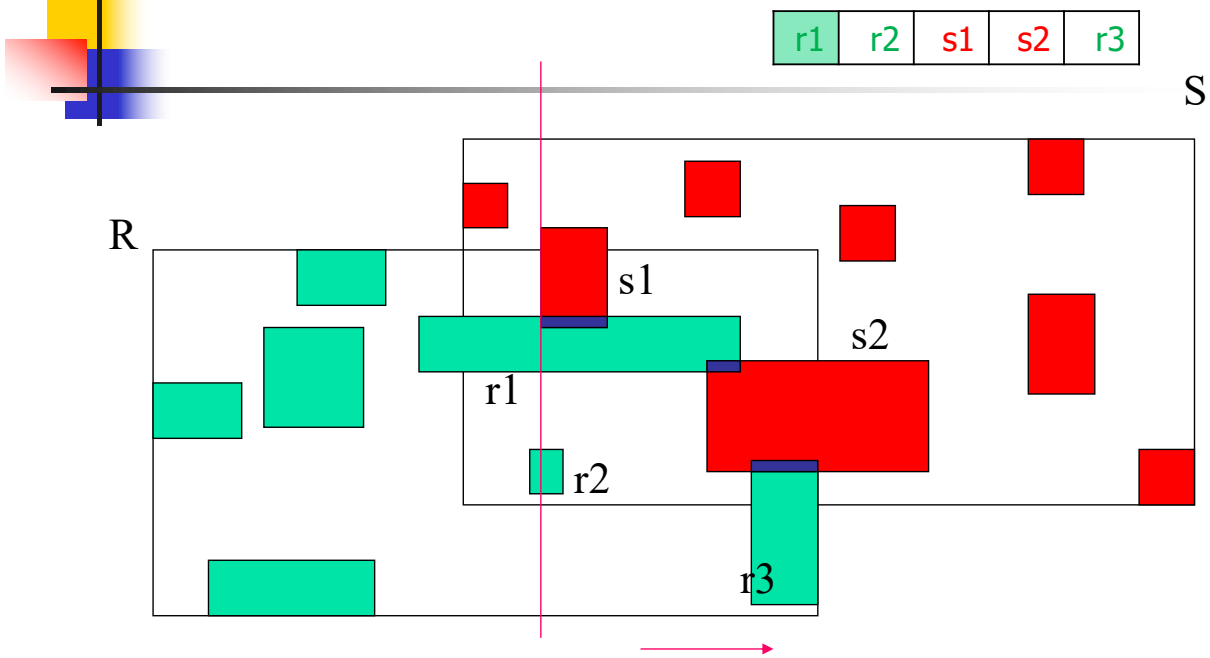
Reached the end of r1
Start with next entry r2

Using Plane Sweep



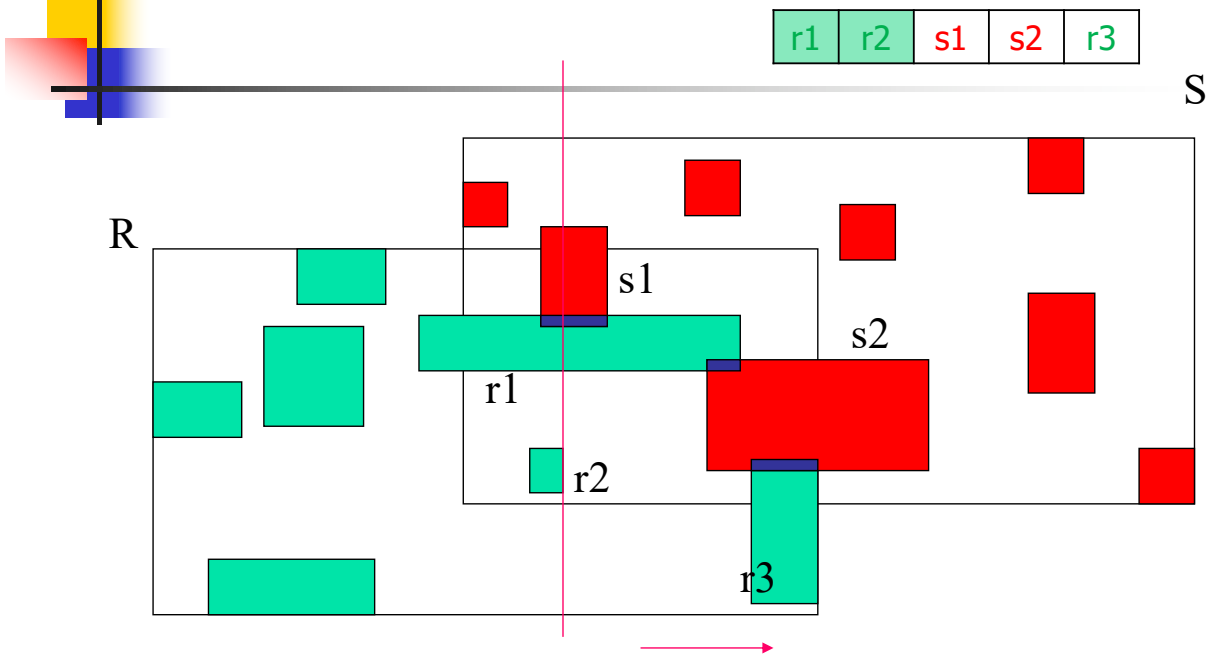
Reposition sweep line

Using Plane Sweep



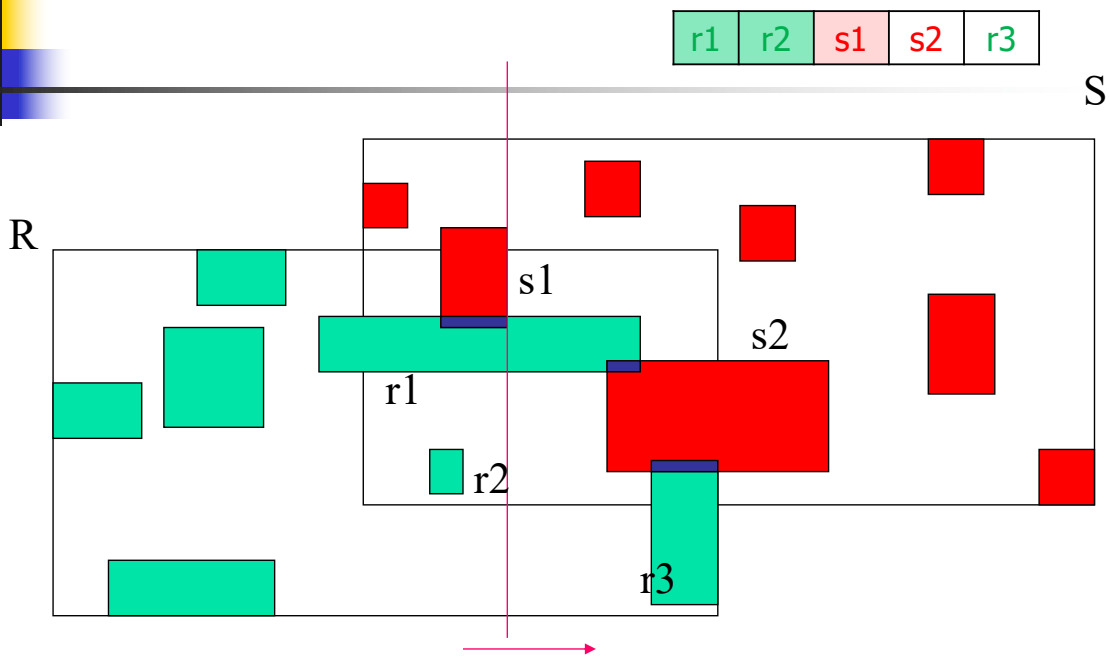
Check if r2 and s1 intersect along y
Do not add (r2,s1) to result

Using Plane Sweep



Reached the end of r2
Start with next entry s1

Using Plane Sweep



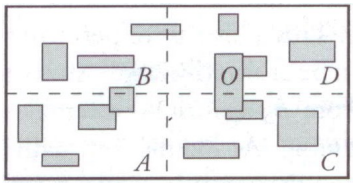
Total of 2(r1) + 1(r2) + 0 (s1)+ 1(s2)+ 0(r3) = 4 comparisons

Spatial Join Algorithm using two R trees with different depths

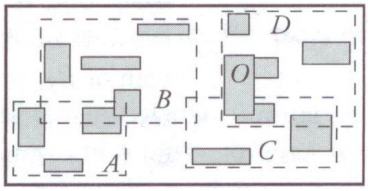
```
SJ(R1, R2:R_NODE);
01     BEGIN
02          FOR (all Er2 in R2) DO
03              FOR (all Er1 in R1) DO
04                  IF (overlap(Er1.rect, Er2.rect)) THEN
05                     IF (R1 and R2 are leaf pages) THEN
06                        output(Er1.oid, Er2.oid)
07                     ELSE IF (R1 is a leaf page) THEN
08                        ReadPage(Er2.ptr);
09                        SJ(Er1.ptr, Er2.ptr)
10                     ELSE IF (R2 is a leaf page) THEN
11                        ReadPage(Er1.ptr);
12                        SJ(Er1.ptr, Er2.ptr)
13                     ELSE
14                        ReadPage(Er1.ptr), ReadPage(Er2.ptr);
15                        SJ(Er1.ptr,Er2.ptr)
16                     END-IF
17                  END-IF
18              END-FOR
19          END-FOR;
20     END.
```

Spatial Hash Join

- Hash join based on Space-driven structures (with redundancy)

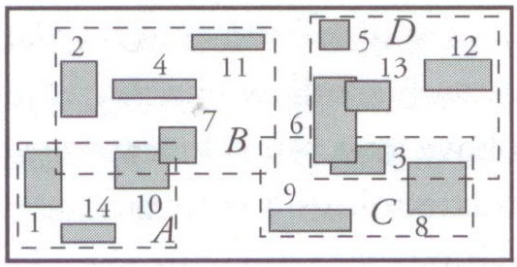


- Hash join based on Data-driven structures (with overlapping)



Spatial Hash Join based on overlapping

- Step 1: ***Initial partitioning***
  - Partition R
  - Each bucket has roughly same number of rectangles
  - Each bucket should fit in memory
  - The overlapping of bucket extents is minimized

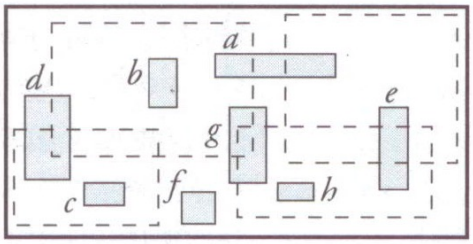


  - Content of the buckets

$$A = \{1, 14, 10\}, \quad B = \{2, 4, 7, 11\},$$
$$C = \{3, 8, 9\}, \quad D = \{5, 6, 12, 13\}$$

Spatial Hash Join based on overlapping

- Step 2: ***Second partitioning***
  - Partition S
  - Keep same buckets from step 1
  - Assign rectangle of S to any bucket whose extent overlaps it

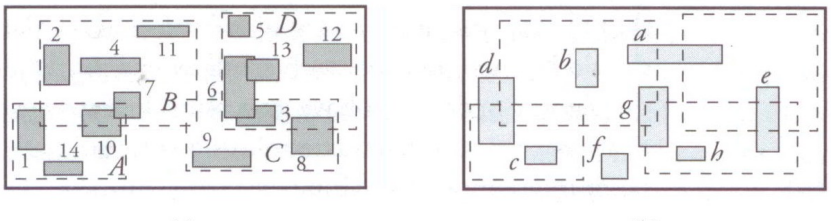


  - Content of the buckets

$$A' = \{c, d\}, \qquad B' = \{a, b, d, g\},$$
$$C' = \{e, g, h\}, \qquad D' = \{a, e\}$$

Spatial Hash Join based on overlapping

- Step 3: ***Join phase***
  - From step 1 and 2, we have two sets of buckets
  - Each bucket from R and S has same extent and location
  - Test each rectangle from bucket where *plane-sweeping* algorithm can be used

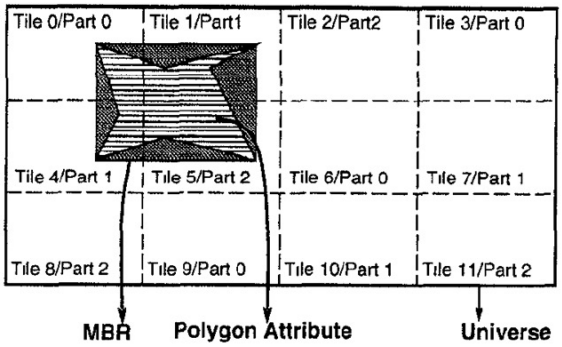


(a) (b)

  - [A, A'], [B, B'], [C, C'], [D, D'] must be joined

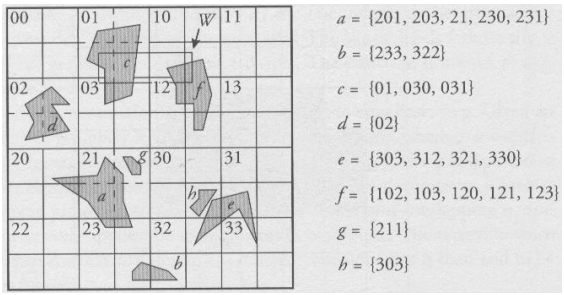
Spatial Hash Join based on redundancy

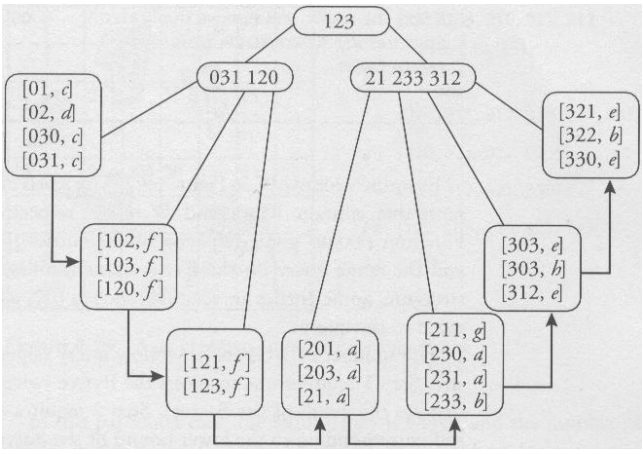
- Partition based Spatial Merge Join
  - Both sets R and S are partitioned with replication
  - Space is regularly tiled
  - Partitions either correspond to tiles or are determined from them using hashing



z-Ordering Spatial Join

- z-ordering tree



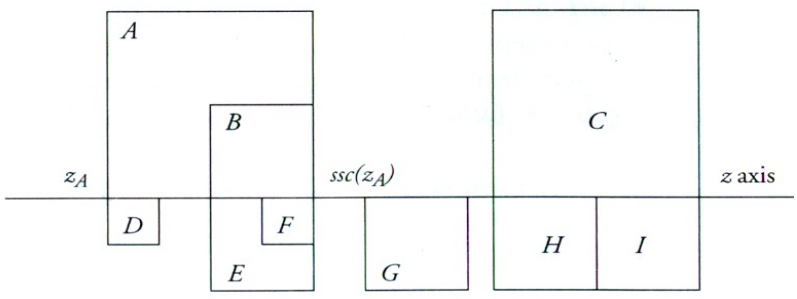


z-Ordering Spatial Join

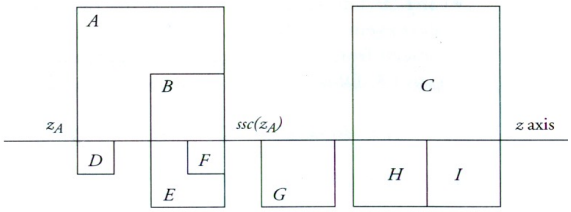
- Assumptions
  - The leaves of each z-ordering tree represent a list L of entries of the form [*z, oid*], ordered on *z*
    - *oid* : *id* of an object whose approximation contains the cell with *z*-order value (key) *z*
    - One cell with key *z* is contained in a cell with key *z'* if z' is a prefix of *z*
- Basic Concept of Join algorithm
  - The lists of entries corresponding to the two relations, *L1* and *L2*, are merged.
  - Keep one pair of entries from the two lists as a candidate for the refinement step if one key is a prefix of the other.
  - Candidate pairs have to be sorted in order to remove duplicates.

z-Ordering Spatial Join

- Notations
  - Rectangles ***above*** **(** ***below*** **)** the *z axis* represent cells of list *L1*(*L2*).
  - ***Lowest*** (***Largest***) value of a rectangle on the ***z*** axis is the key ***z*** (***z'***) associated with the cell ***c***.
  - c : *z'* = *scc*(*z*) (smallest cell in the lower-right corner of z)
  - If z = 30, (a maximal depth of 7 for the decomposition) z' = 3033333.



z-Ordering Spatial Join



ZORDERINGJOIN (L_1,L_2: list of *ids*): set of pairs of entries

```
begin
  result: Set of pairs of ids, initially empty
  while not (eof(L1) and empty(S1) and eof (L2) and empty (S2))
  begin
    event = MIN (CURRENT(L1), SCC(top(S1)),
                 CURRENT(L2), SCC(top(S2)))
    if (event = CURRENT(L1)) then // left bound of a rectangle
      ENTRY (L1, S1)
    else if (event = SCC(top(S1))) then // right bound of a rectangle
      result += EXIT (S1, S2)
    else if (event = CURRENT(L2)) then // left bound of a rectangle
      ENTRY (L2, S2)
    else if (event = SCC (top(S2))) then // right bound of a rectangle
      result += EXIT (S2, S1);
  end while
  sort result; remove duplicates;
  return result
end
```

	C_1	S_1	C_2	S_2	Event Actions
Step 0	A	()	D	()	
Step 1	B	(A)	D	()	event = current (L_1) = A
Step 2	B	(A)	E	(D)	event = current (L_2) = D
Step 3	B	(A)	E	()	event = scc(top(S_2)) = D result={[A,D]}
Step 4	C	(B,A)	E	()	event = current (L_1) = B
step 5	C	(B,A)	F	(E)	event = current (L_2) = E
Step 6	C	(B,A)	G	(F,E)	event = current (L_2) = F
Step 7	C	(A)	G	(F,E)	event = scc(top(S_1)) = B result= {[A,D]} + {[B,F], [B,E]}
Step 8	C	()	G	(F,E)	event = scc(top(S_1)) = A result= {[A,D],[B,F], [B,E]} + {[A,F],[A,E]}