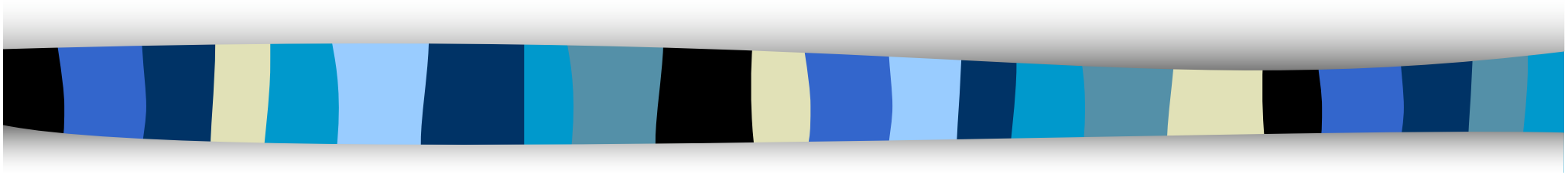
The A-tree: An Index Structure for High-dimensional Spaces Using Relative Approximation



Yasushi Sakurai *(NTT Cyber Space Laboratories)*

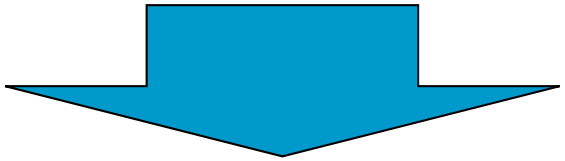
Masatoshi Yoshikawa *(Nara Institute of Science and Technology)*

Shunsuke Uemura *(Nara Institute of Science and Technology)*

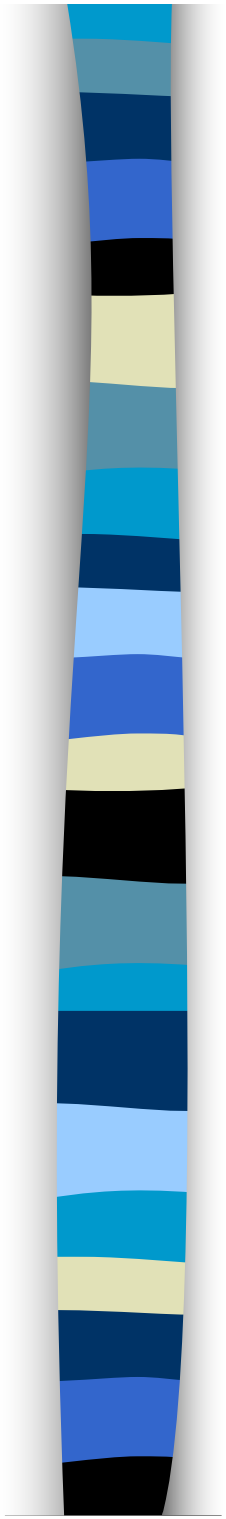
Haruhiko Kojima *(NTT Cyber Solutions Laboratories)*

Introduction

- Demand
  - High-performance multimedia database systems
  - Content-based retrieval with high speed and accuracy
- Multimedia databases
  - Large size
  - Various features, high-dimensional data



- More efficient spatial indices for high-dimensional data

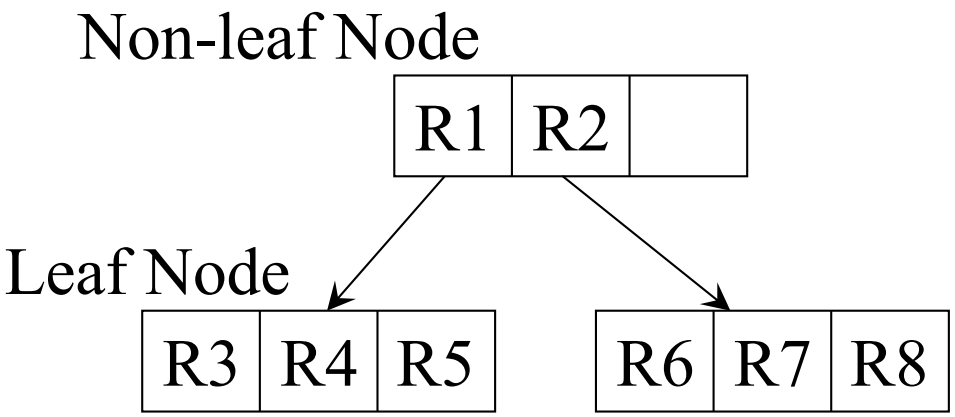


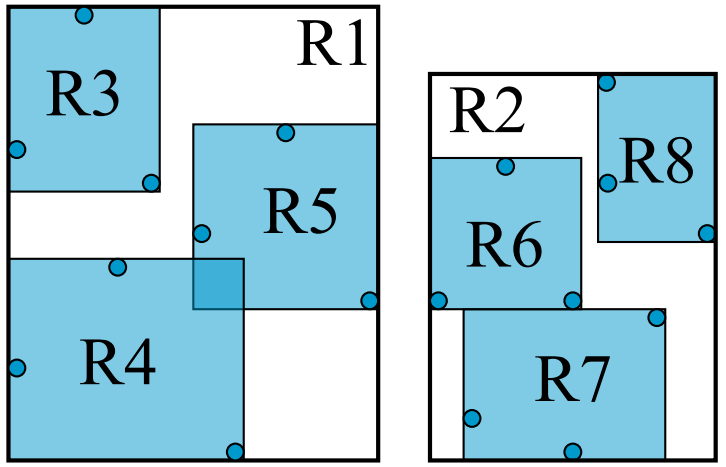
Our Approach

- VA-File and SR-tree are excellent search methods for high-dimensional data.
- Comparisons of them motivated the concept of the A-tree.
  - No comparisons of them have been reported.
  - We performed experiments using various data sets
- Approximation tree (A-tree)
  - Relative approximation: MBRs and data objects are approximated based on their parent MBR.
  - About 77% reduction in the number of page accesses compared with VA-File and SR-tree

Related Work (1)

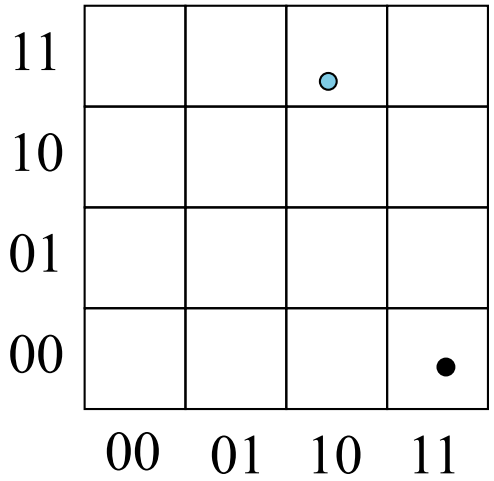
- R-tree family
  - Tree structure using MBRs (Minimum Bounding Rectangles) and/or MBSs (Minimum Bounding Spheres)
  - SR-tree:
    - Structured by both MBRs and MBSs
    - Outperforms SS-tree and R*-tree for 16-dimensional data





Related Work (2)

- VA-File (Vector Approximation File)
  - Use approximation file and vector file
  1. Divide the entire data space into cells
  2. Approximate vector data by using the cells, then create the approximation file
  3. Select candidate vectors by scanning the approximation file
  4. Access to the candidate vectors in the vector file
  - Better than X-tree and R*-tree beyond dimensionality of 6

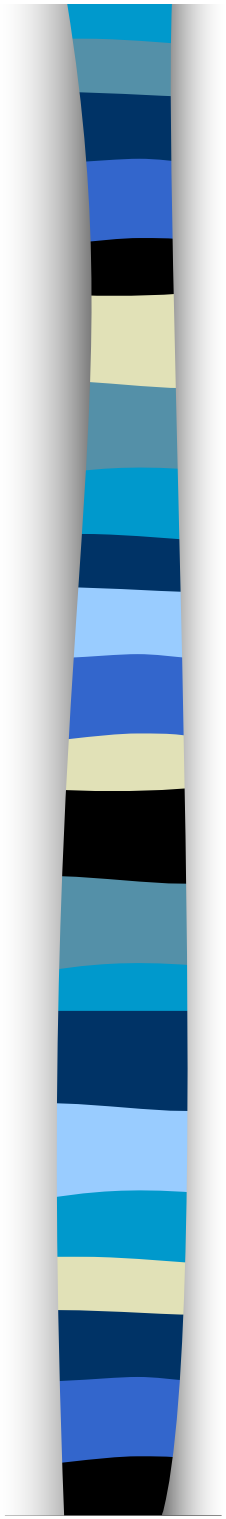


	Approximation	Vector Data
○	10 11	0.6 0.8
●	11 00	0.9 0.1

Experimental Results and Analysis

--- Properties of the SR-tree ---

- Structure suitable for non-uniformly distributed data
  - Structure changes according to data distribution.
- Large entry size for high-dimensional spaces
  - Large entries ⇨ small fanout ⇨ many node accesses
- Changing node size and fanout
  - Larger node size does NOT lead to low IO cost.
  - Larger fanout always contributes to the reduction in node accesses.
- MBS contribution
  - The contribution of MBSs in node pruning is small in high-dimensional spaces.



Experimental Results and Analysis

--- Properties of the VA-File ---

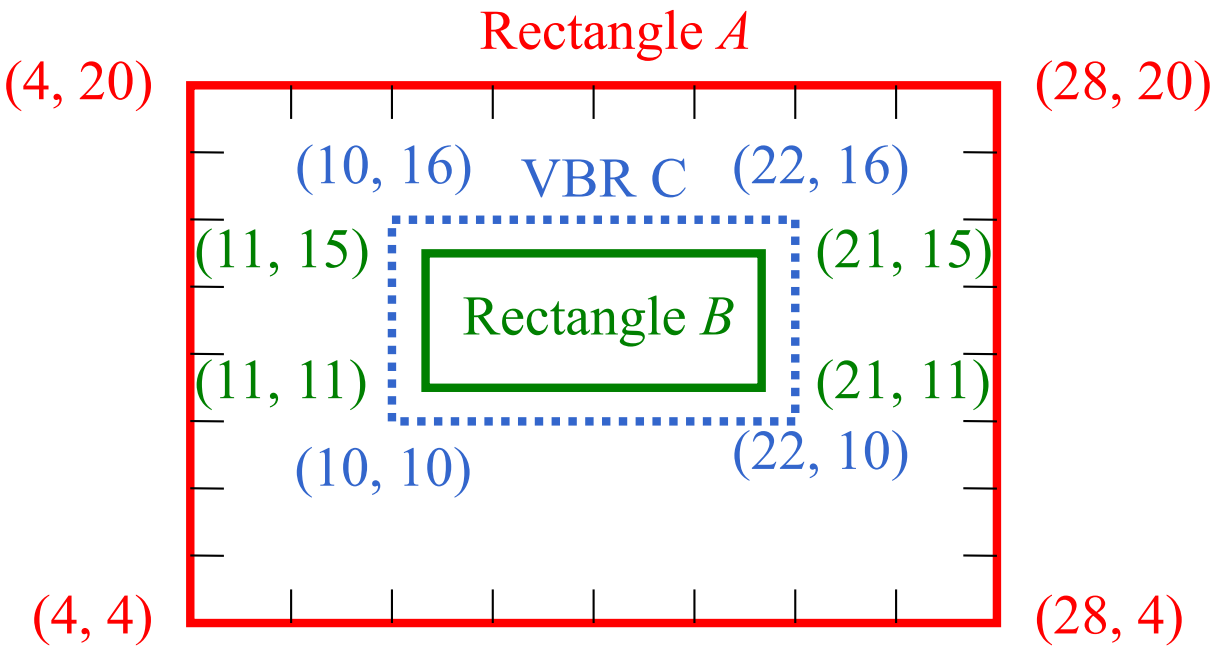
- Data skew degenerates search performance.
  - Absolute approximation: the approximation is independent of data distribution.
  - Effective for uniformly distributed data
  - Unsuitable for non-uniformly distributed data
    - A large amount of dense data tends to be approximated by the same value.
    - Absolute approximation leads to large approximation errors.

The A-tree (Approximation tree)

- Ideas from the SR-tree and VA-File comparison:
  - Tree structure
    - Tree structures are suitable for non-uniformly distributed data.
  - Relative approximation
    - MBRs and data objects are approximated based on their parent bounding rectangle.
    - Small approximation error
    - Small entry size and large fanout ⇨ low IO cost
  - Partial usage of MBSs in high-dimensional searches
    - MBSs are not stored in the A-tree.
    - The centroid of data objects in a subtree is used only for update.

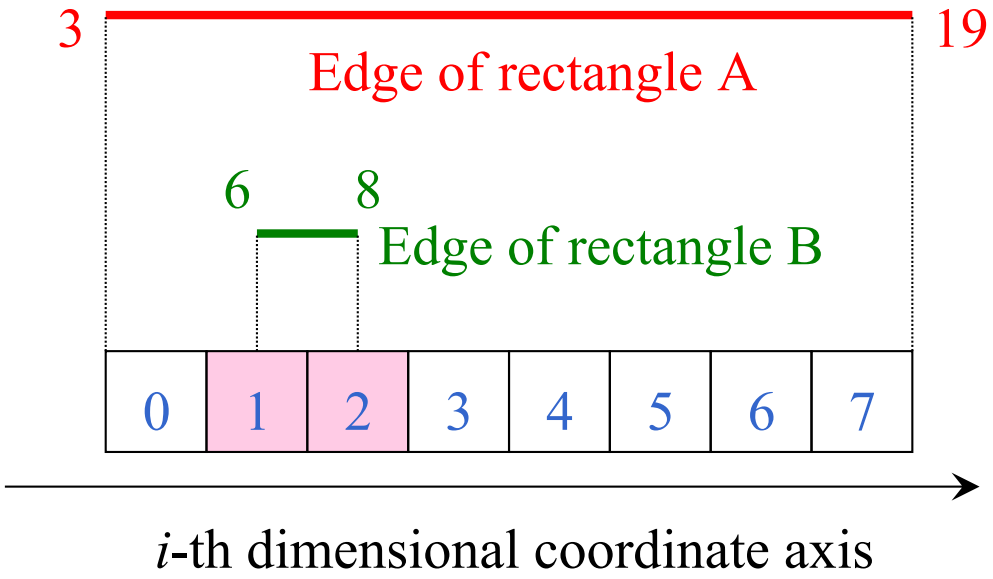
Virtual Bounding Rectangle (VBR)

- C approximates a rectangle *B*.
- C is calculated from rectangles *A* and *B*.
- Search using VBRs guarantees the same result as that of MBRs.



Subspace Code

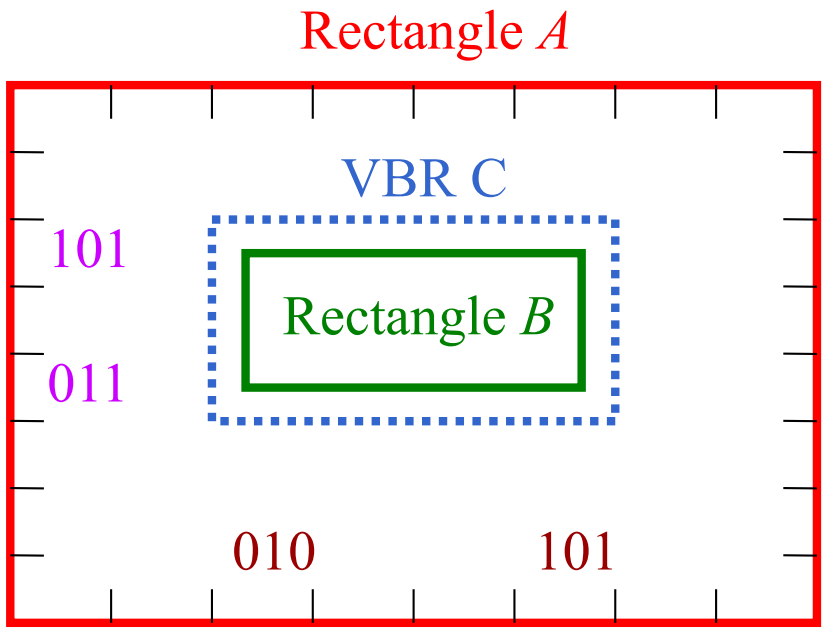
- Subspace code represents a VBR.
- The edge of child MBR B is quantized in relation to the edge of parent MBR A.
- The edge of B is approximated as a pair of 8-ary codes (1, 2) or binary codes (001, 010).



Subspace Code

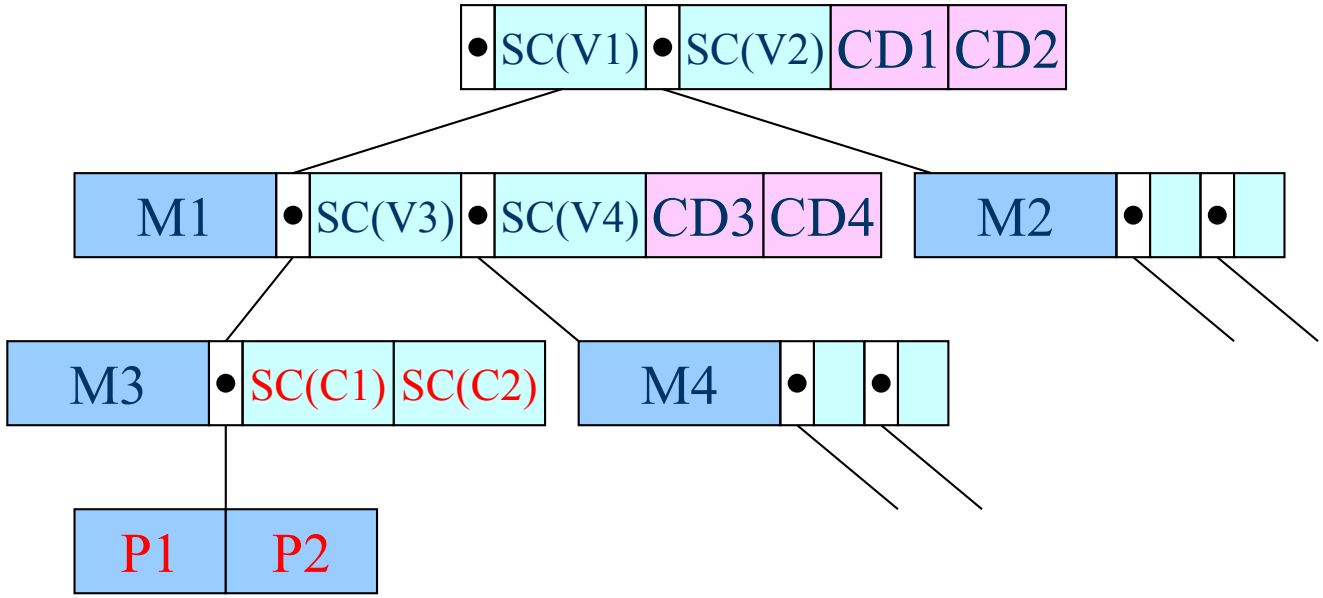
- C is the VBR of *B* in *A*
- C is represented by the subspace codes:

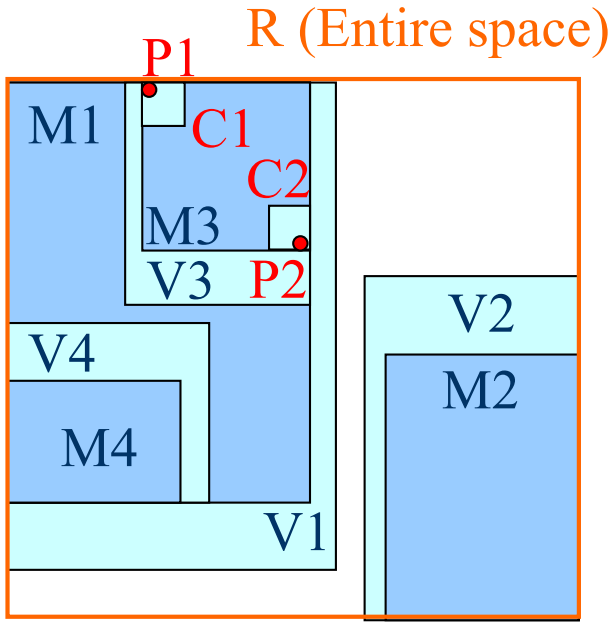
S = (010, 011, 101, 101)



The A-tree Structure

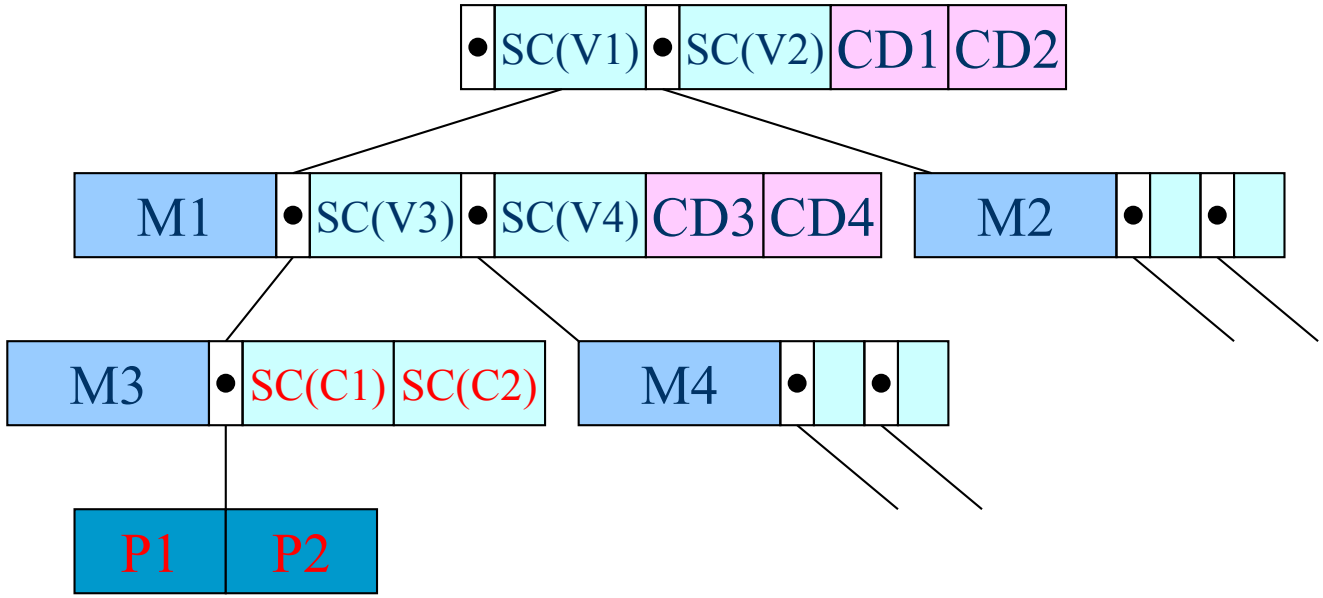
- Relative approximation:
  - MBRs and data objects in child nodes are approximated based on parent MBR.
- Configuration
  - One node contains partial information of rectangles in two consecutive generations.

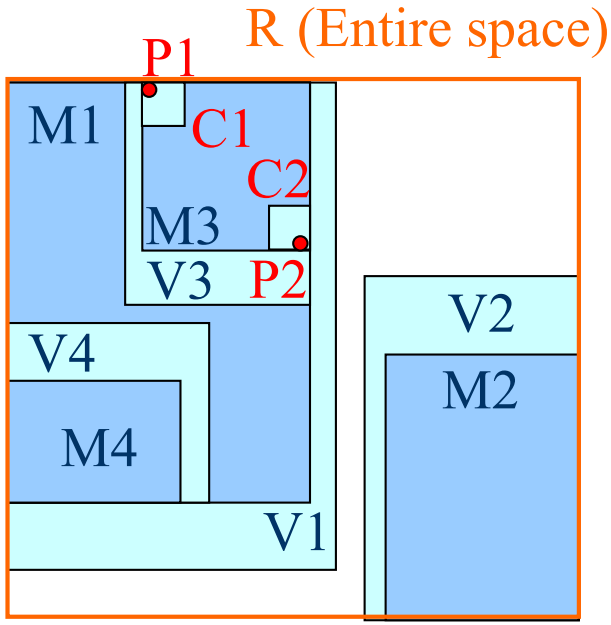




The A-tree Structure

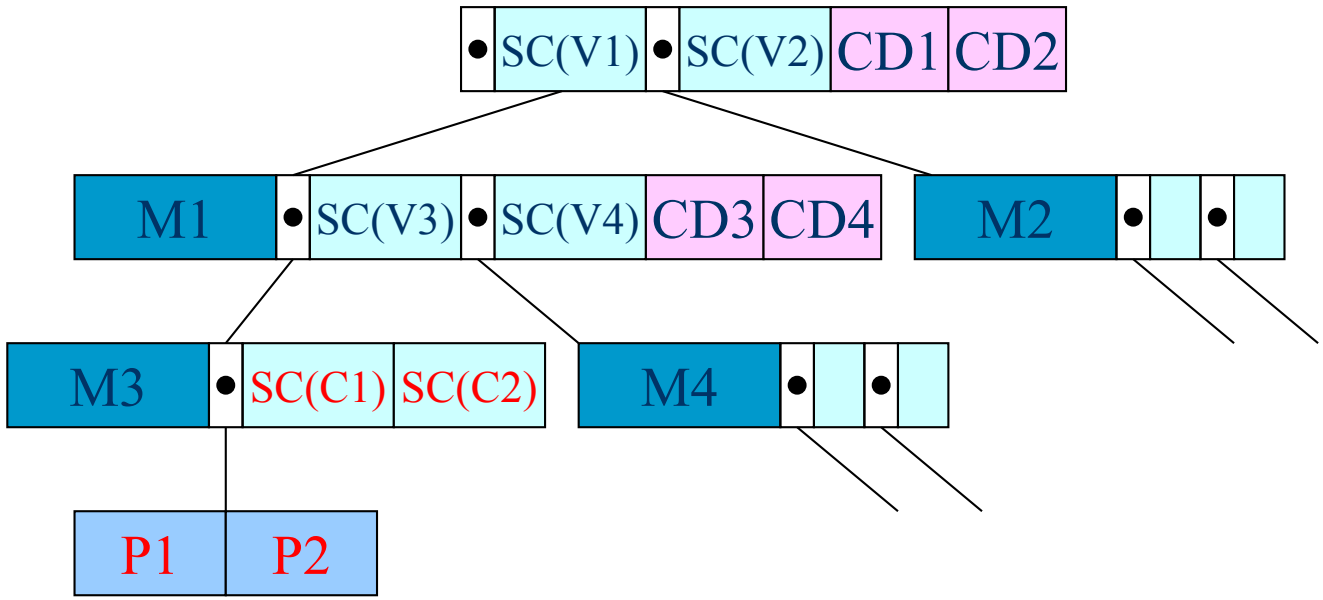
P1 and P2: data objects,

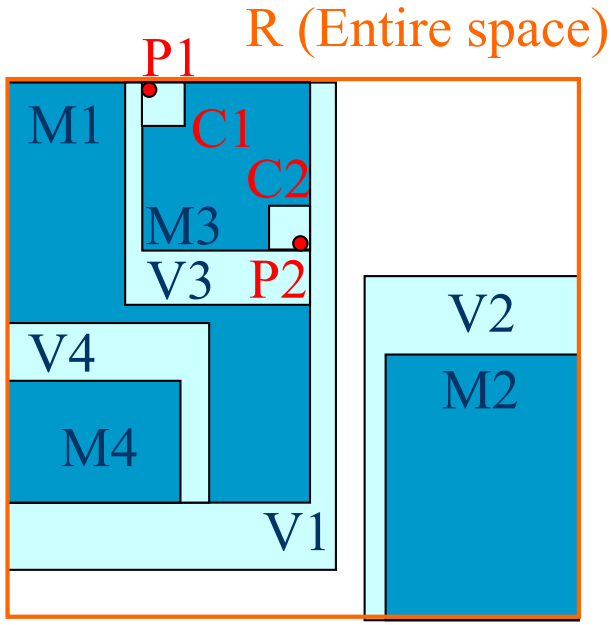




The A-tree Structure

P1 and P2: data objects, M1 -- M4: MBRs

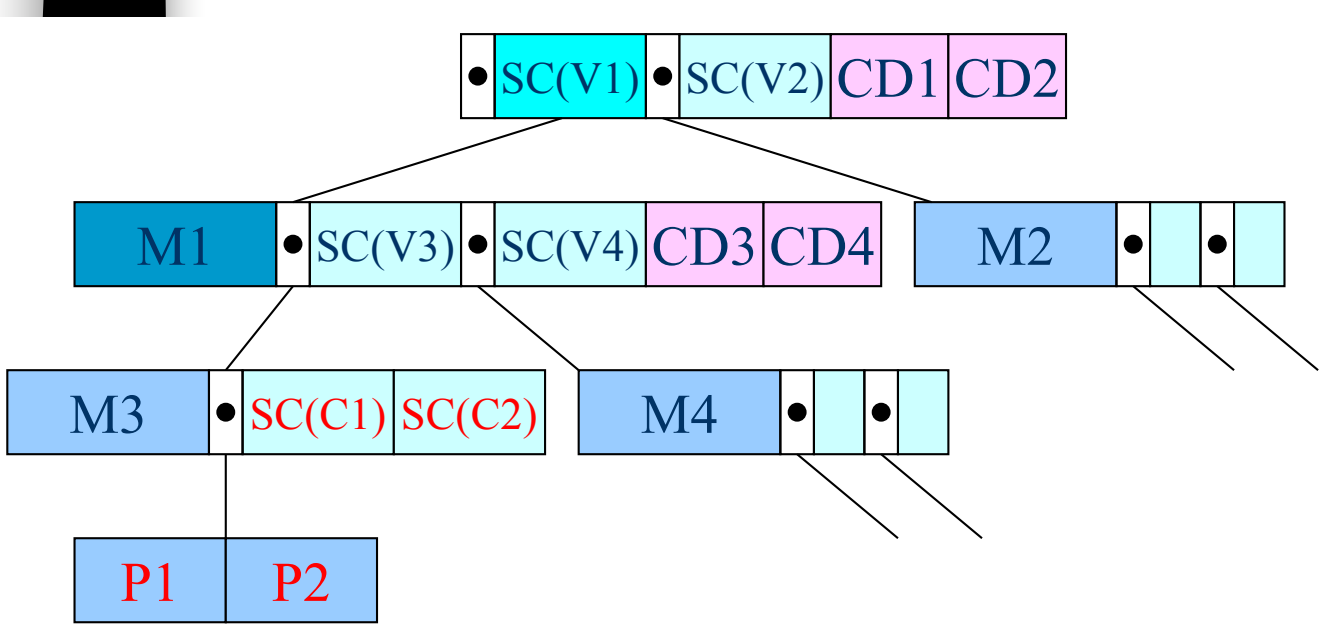


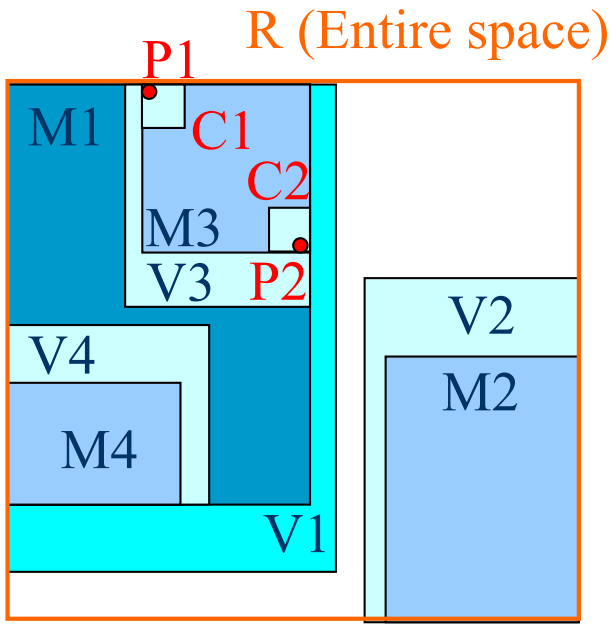


The A-tree Structure

P1 and P2: data objects, M1 -- M4: MBRs

SC(V1) -- SC(V4): subspace codes of VBRs for the MBRs

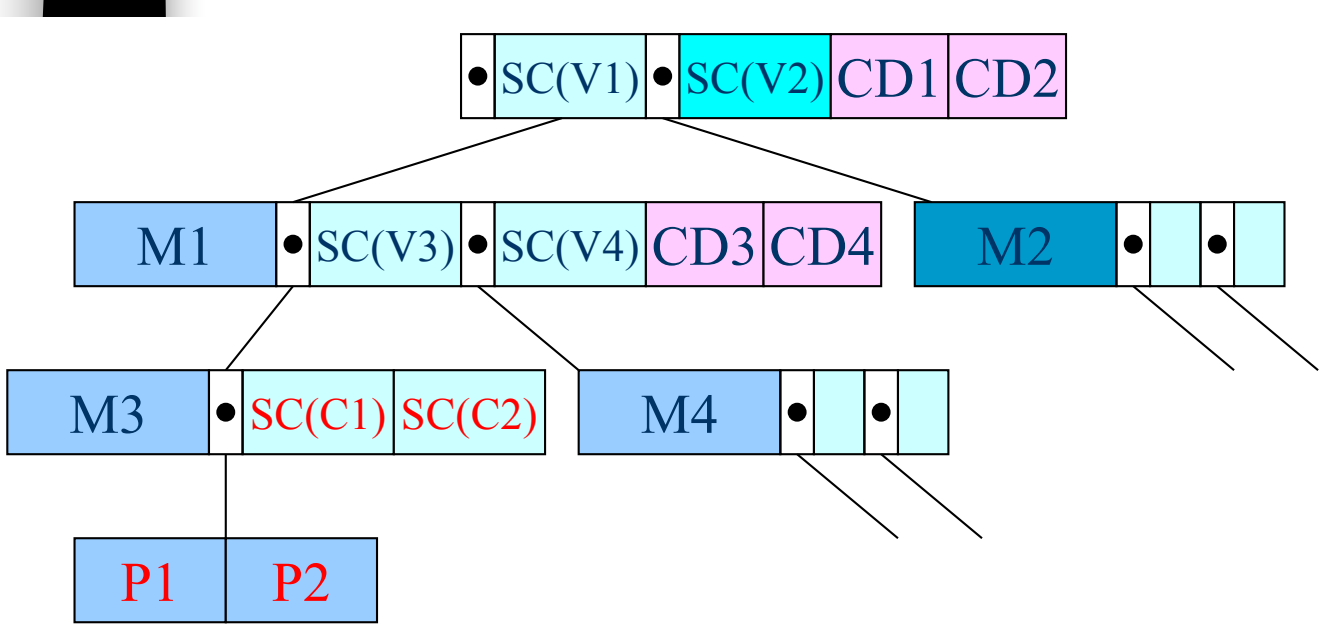


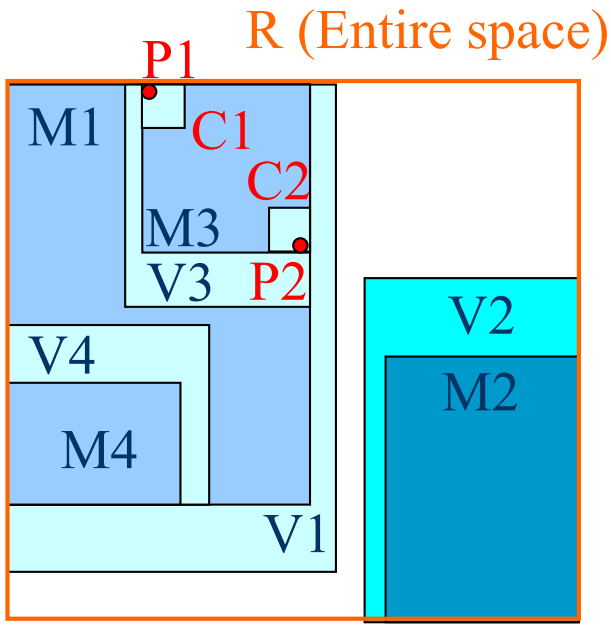


The A-tree Structure

P1 and P2: data objects, M1 -- M4: MBRs

SC(V1) -- SC(V4): subspace codes of VBRs for the MBRs

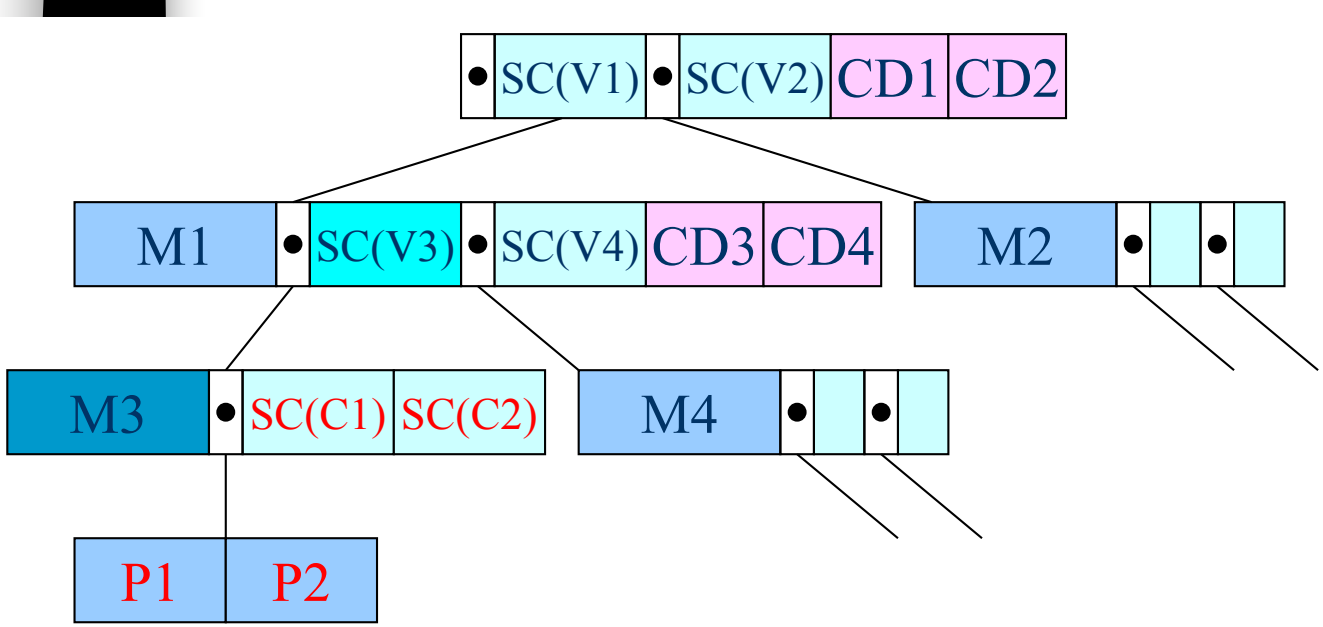


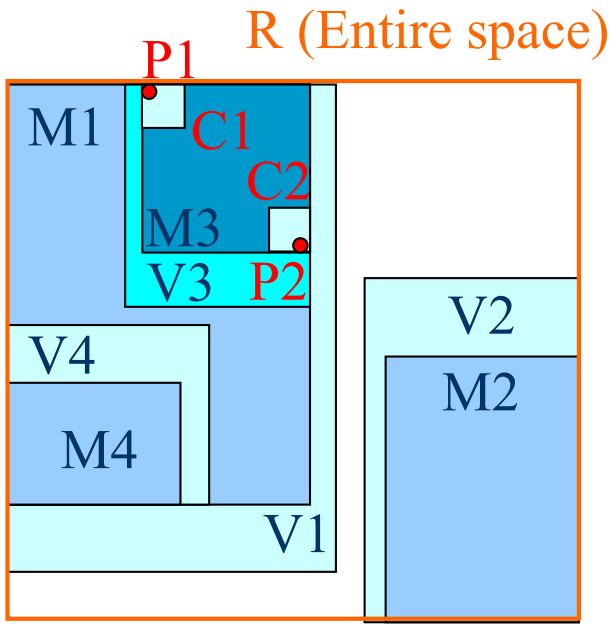


The A-tree Structure

P1 and P2: data objects, M1 -- M4: MBRs

SC(V1) -- SC(V4): subspace codes of VBRs for the MBRs

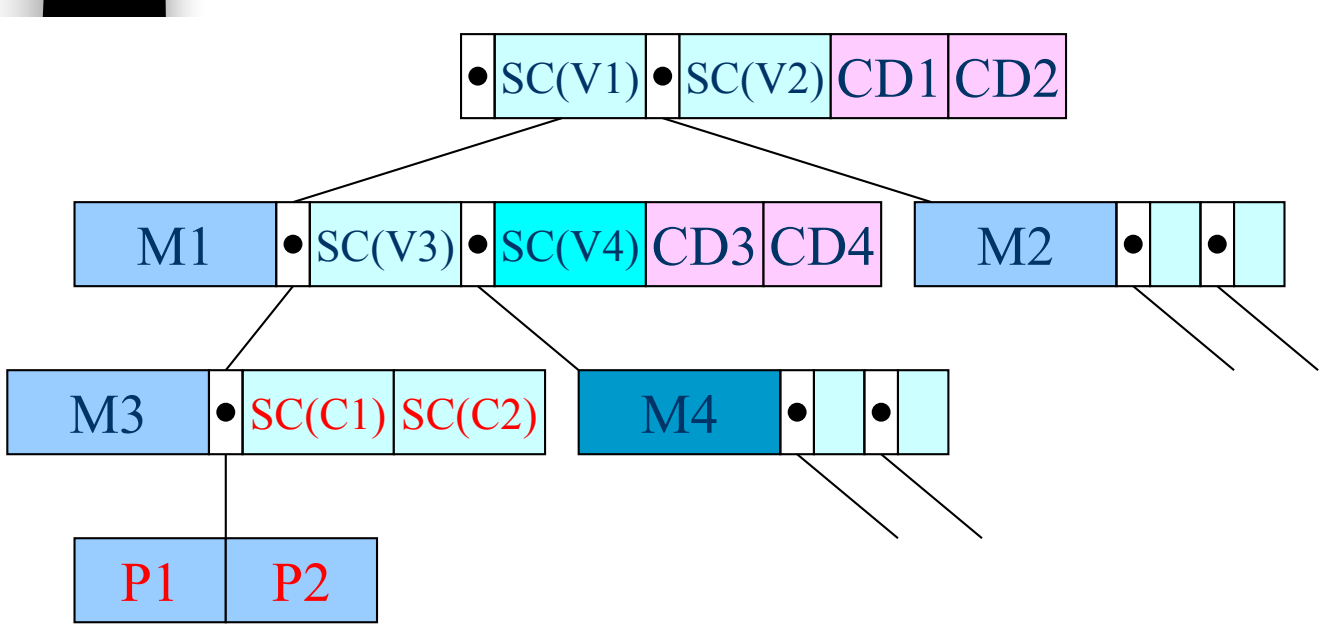


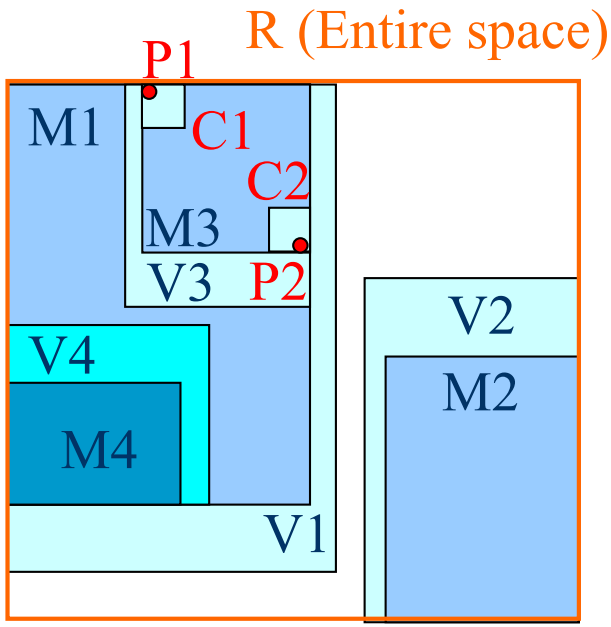


The A-tree Structure

P1 and P2: data objects, M1 -- M4: MBRs

SC(V1) -- SC(V4): subspace codes of VBRs for the MBRs



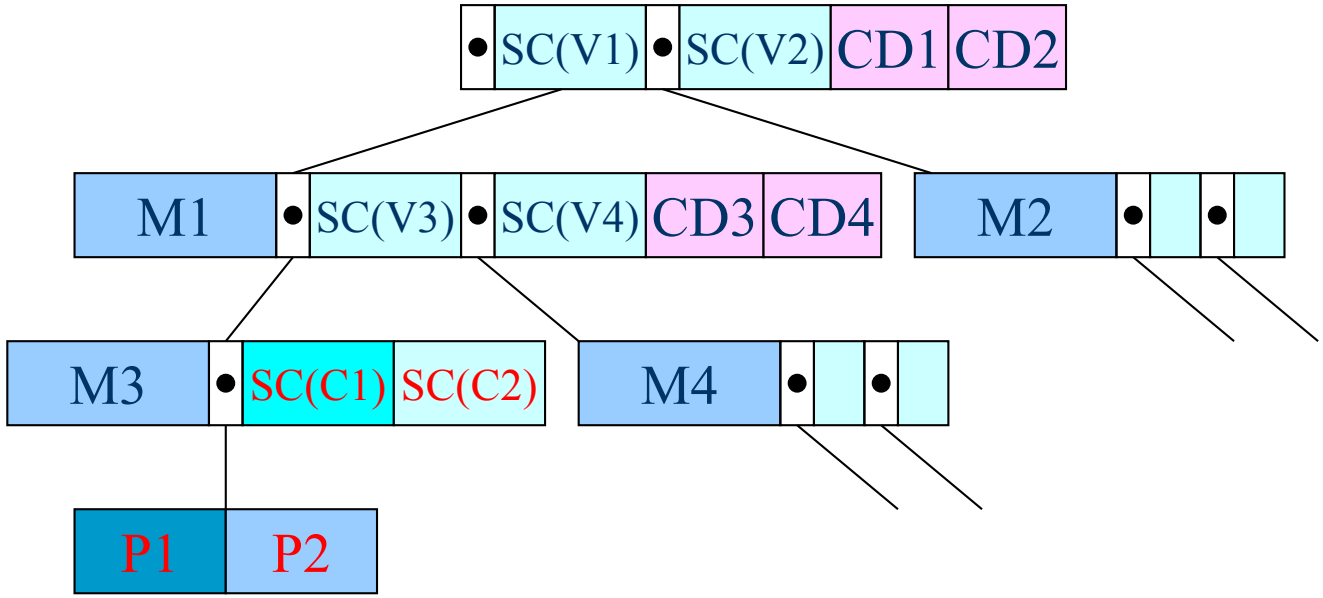


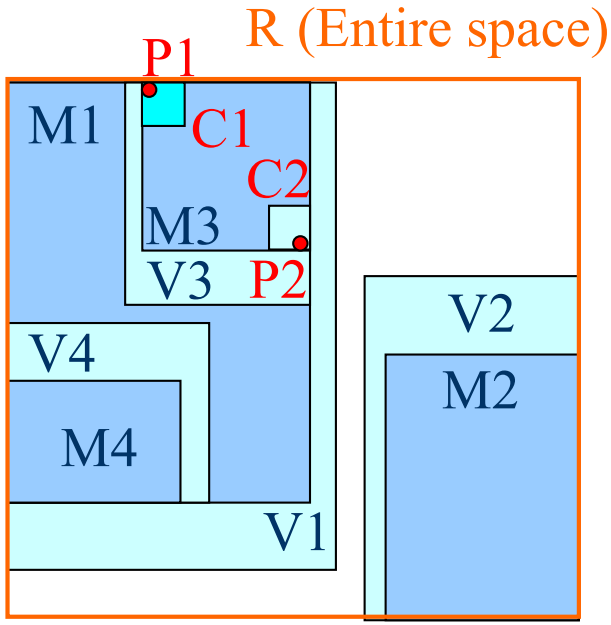
The A-tree Structure

P1 and P2: data objects, M1 -- M4: MBRs

SC(V1) -- SC(V4): subspace codes of VBRs for the MBRs

SC(C1) and SC(C2): subspace codes of VBRs for the data objects



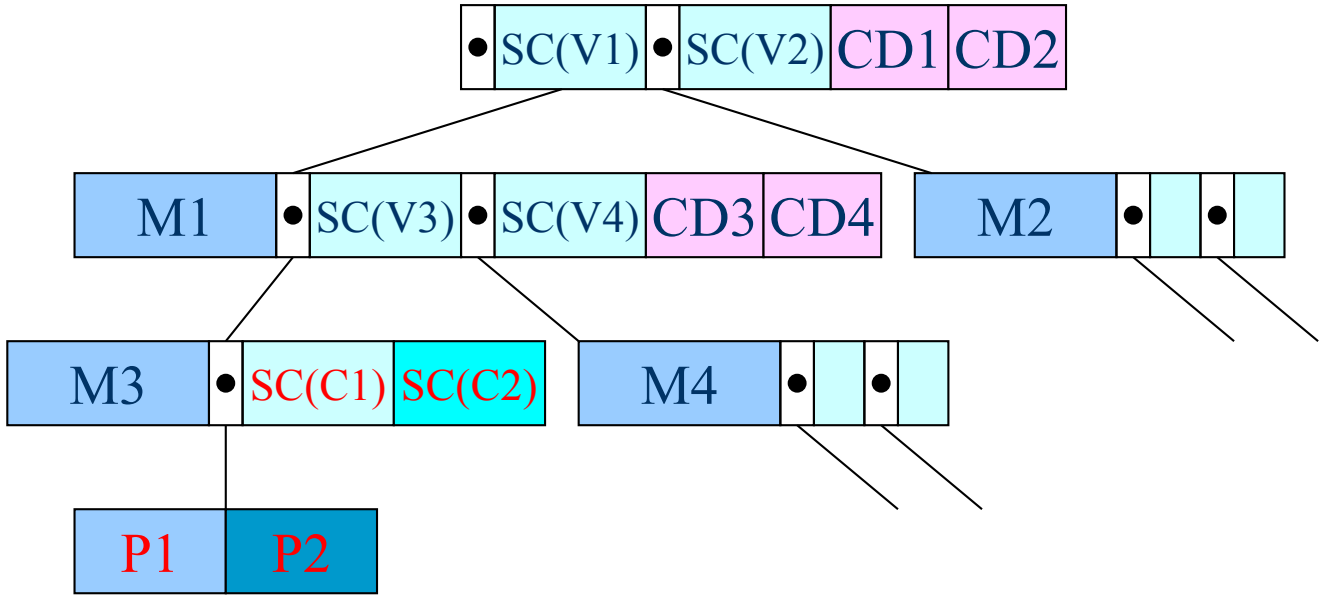


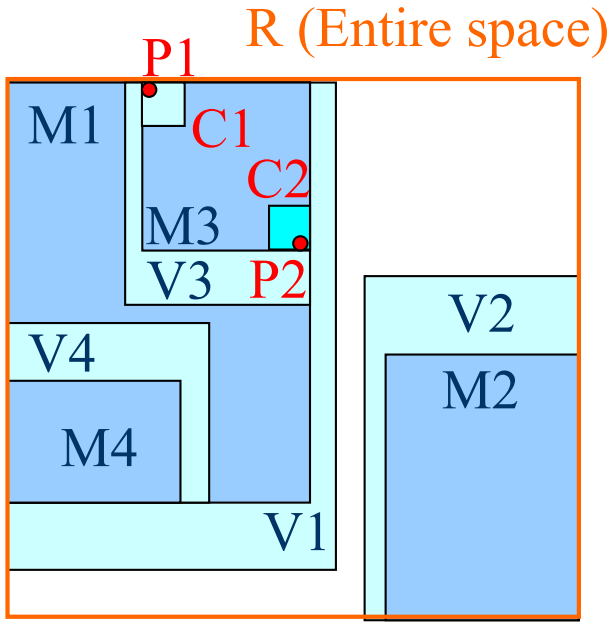
The A-tree Structure

P1 and P2: data objects, M1 -- M4: MBRs

SC(V1) -- SC(V4): subspace codes of VBRs for the MBRs

SC(C1) and SC(C2): subspace codes of VBRs for the data objects





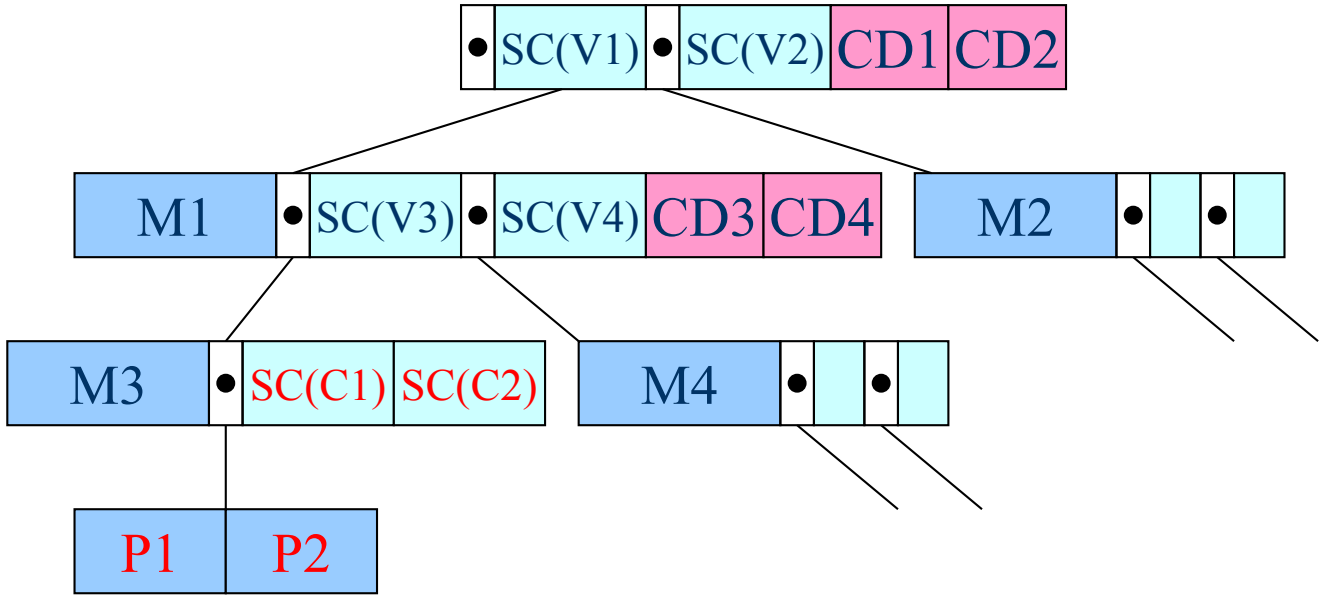
The A-tree Structure

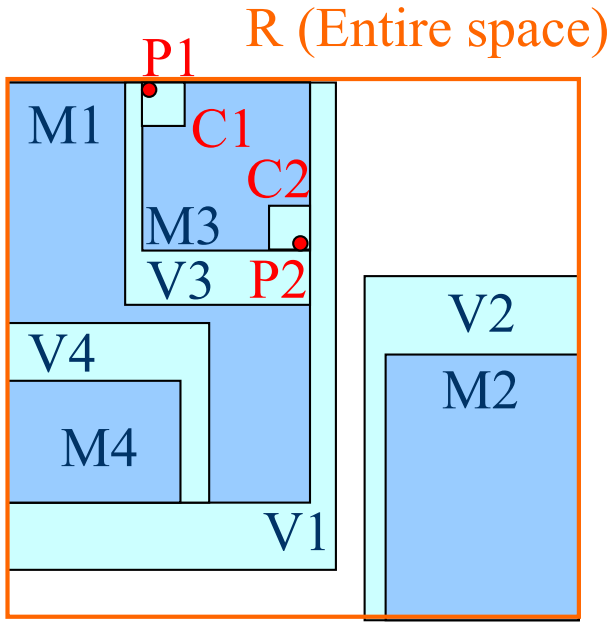
P1 and P2: data objects, M1 -- M4: MBRs

SC(V1) -- SC(V4): subspace codes of VBRs for the MBRs

SC(C1) and SC(C2): subspace codes of VBRs for the data objects

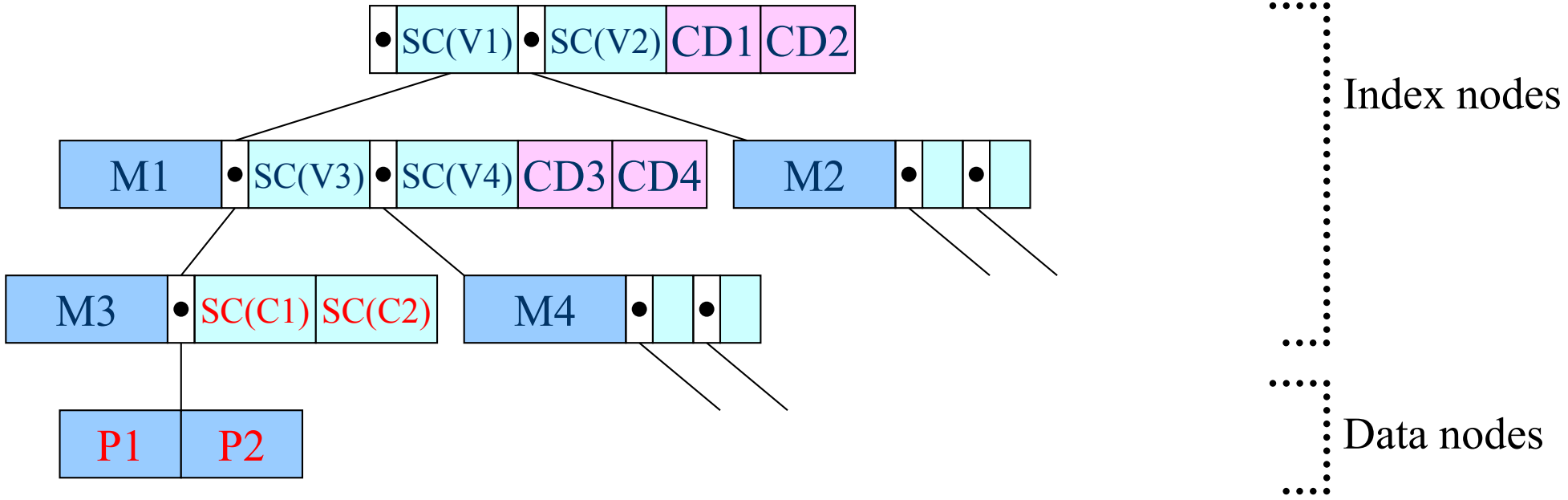
CD1 -- CD4: centroid of the data objects in the subtree





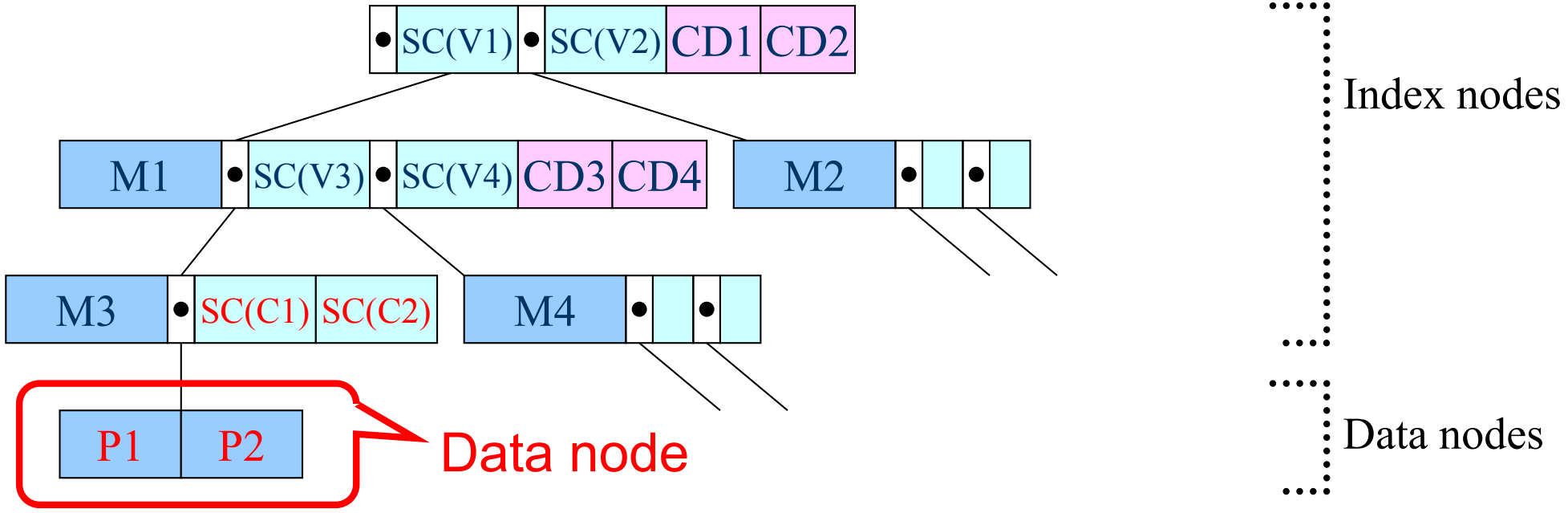
The A-tree Structure

- Data nodes
- Index nodes
  - leaf nodes
  - intermediate nodes
  - root node



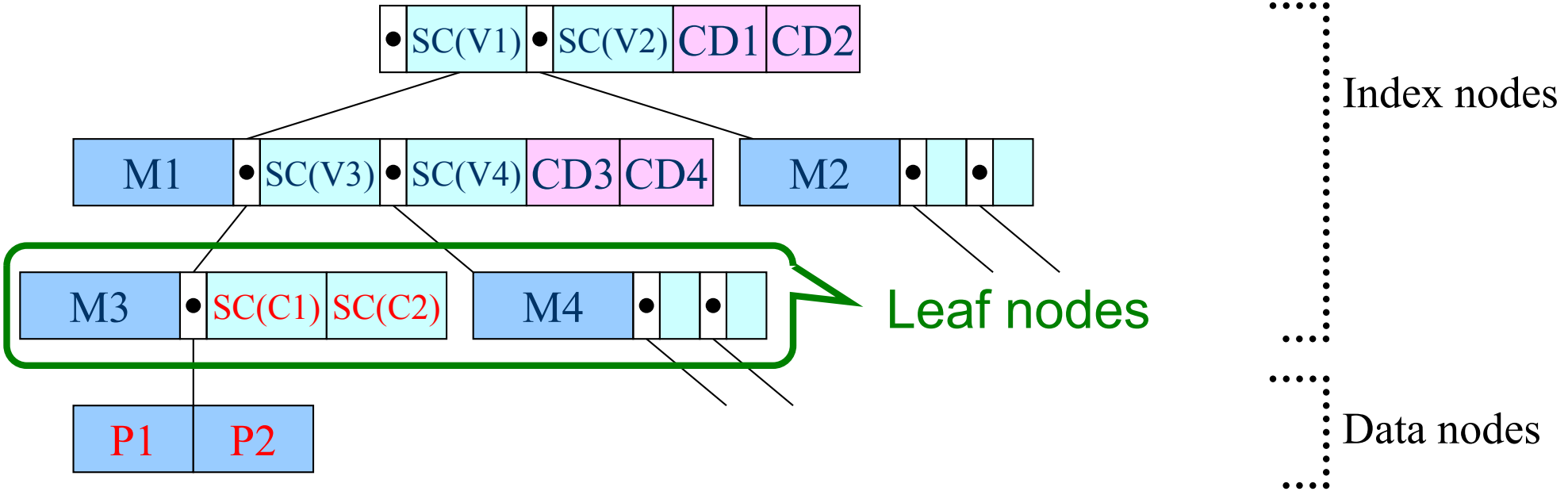
The A-tree Structure

- Data node
  - data objects
  - pointers to the data description records



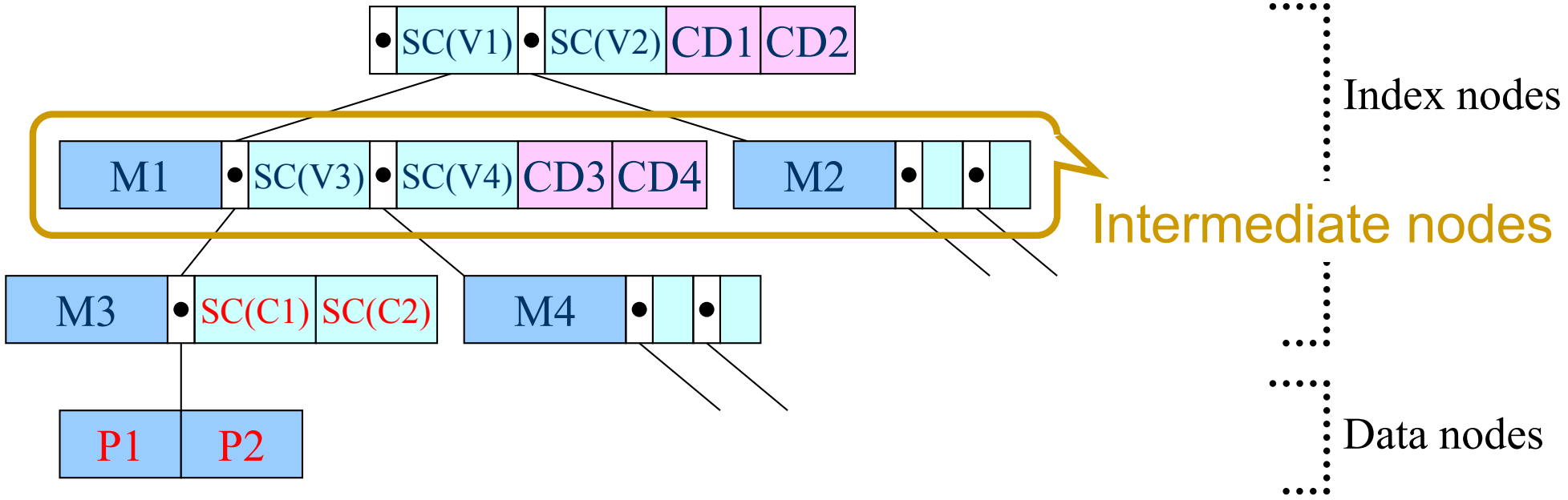
The A-tree Structure

- Leaf node
  - an MBR
  - a pointer to the data node
  - subspace codes of VBRs



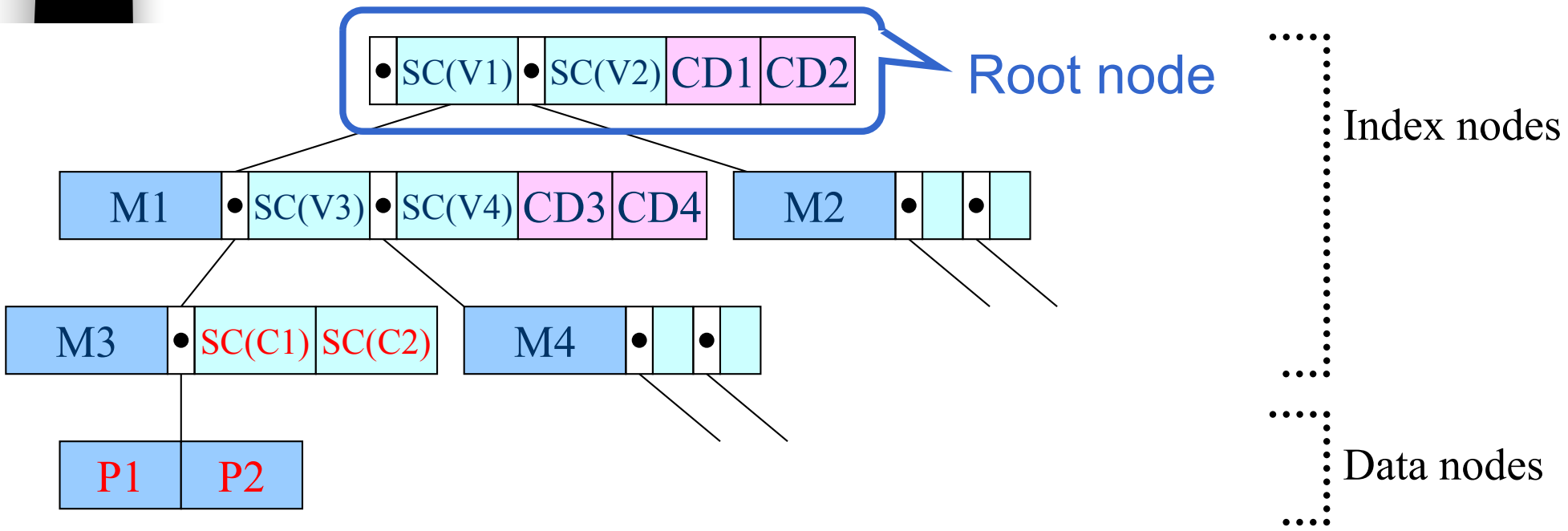
The A-tree Structure

- Intermediate node
  - an MBR
  - a list of entries
    - a pointer to the child node
    - the subspace code of a VBR
    - the centroid of data objects in the subtree
    - the number of the data objects



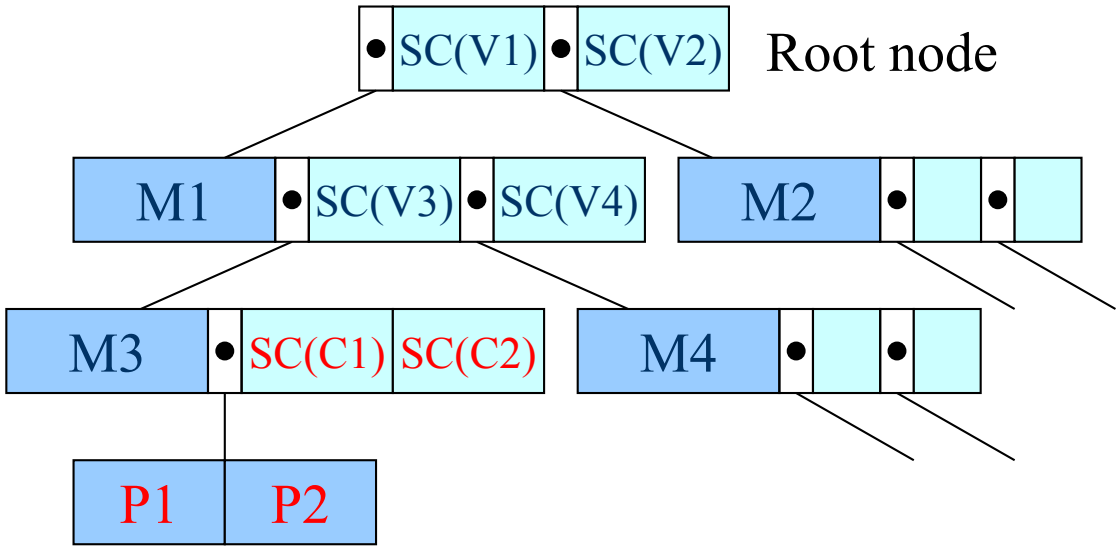
The A-tree Structure

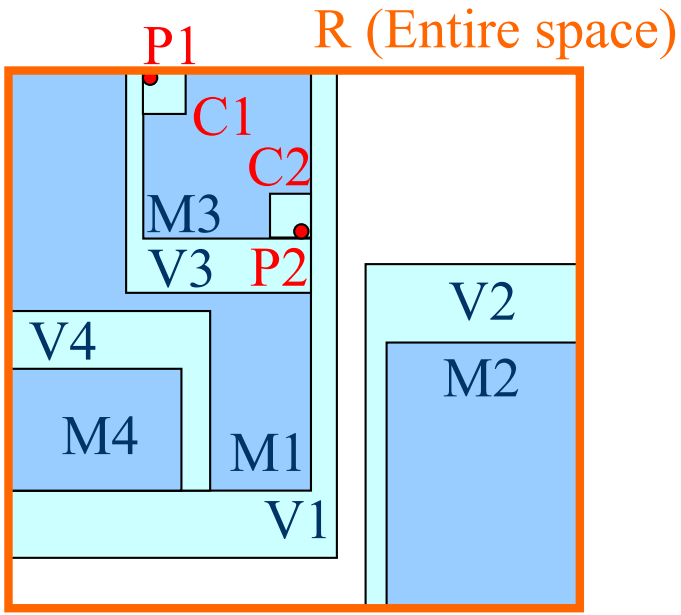
- Root node:
  - a list of entries
    - a pointer to the child node
    - the subspace code of a VBR
    - the centroid of data objects in the subtree
    - the number of the data objects



Search Algorithm

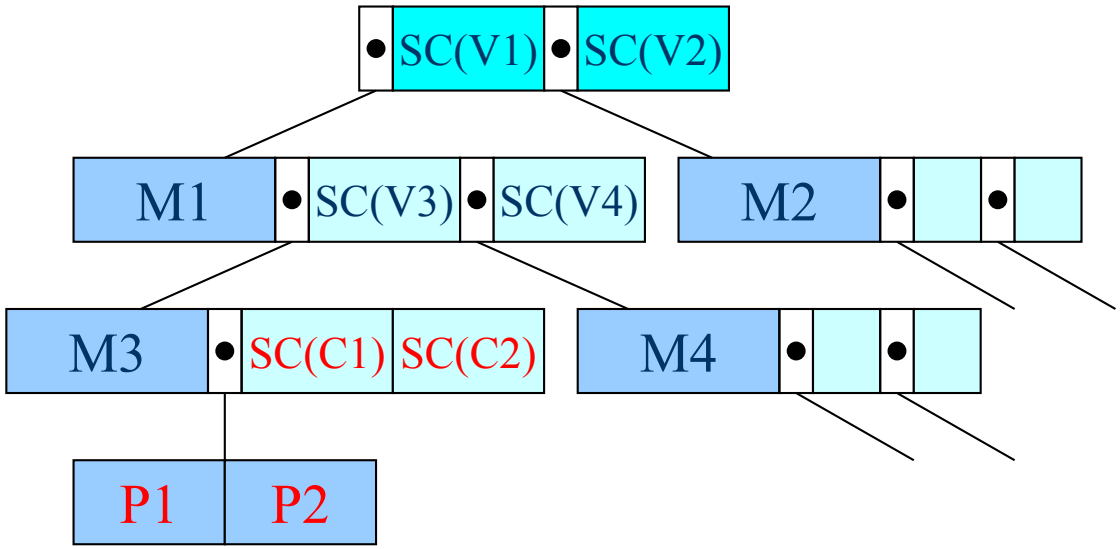
- Basic ideas:
  - VBRs are calculated from parent MBR and the subspace codes.
  - Exception: the entire space is used in the root node.
  - The algorithm uses calculated VBRs for pruning.

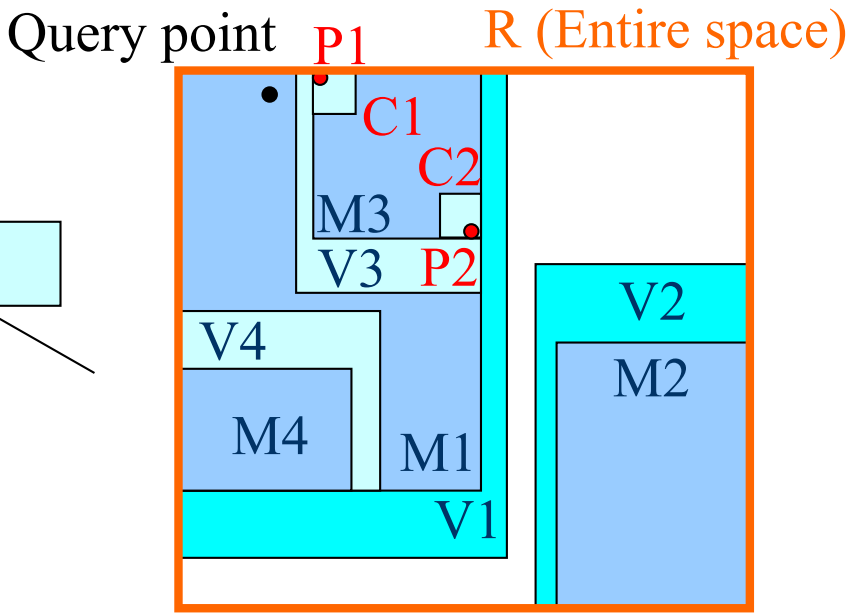




Search Algorithm

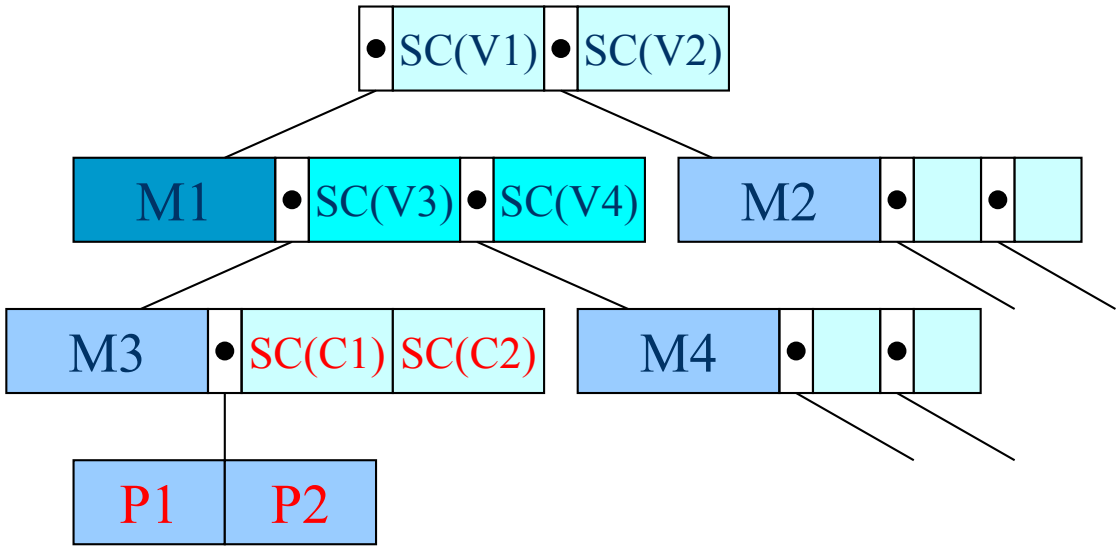
- Calculate V1 and V2 from R, SC(V1) and SC(V2)

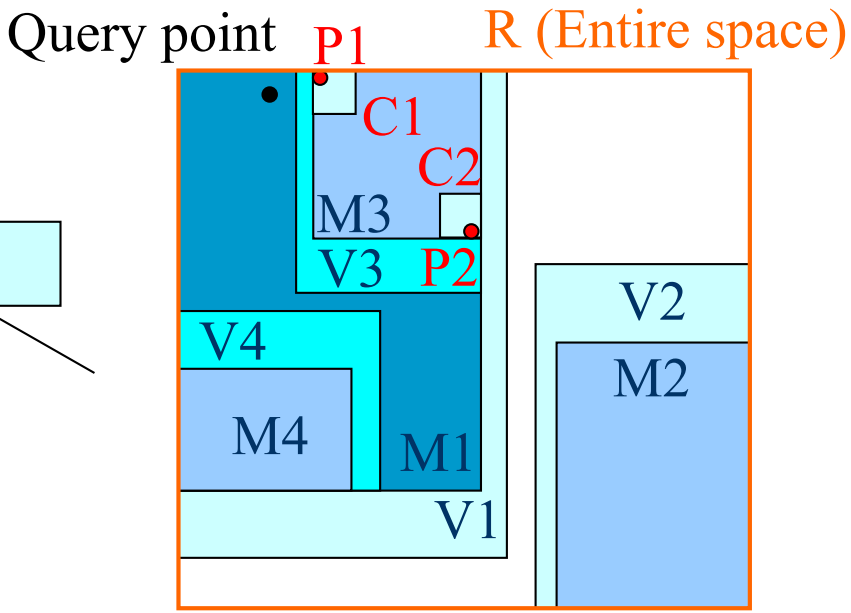




Search Algorithm

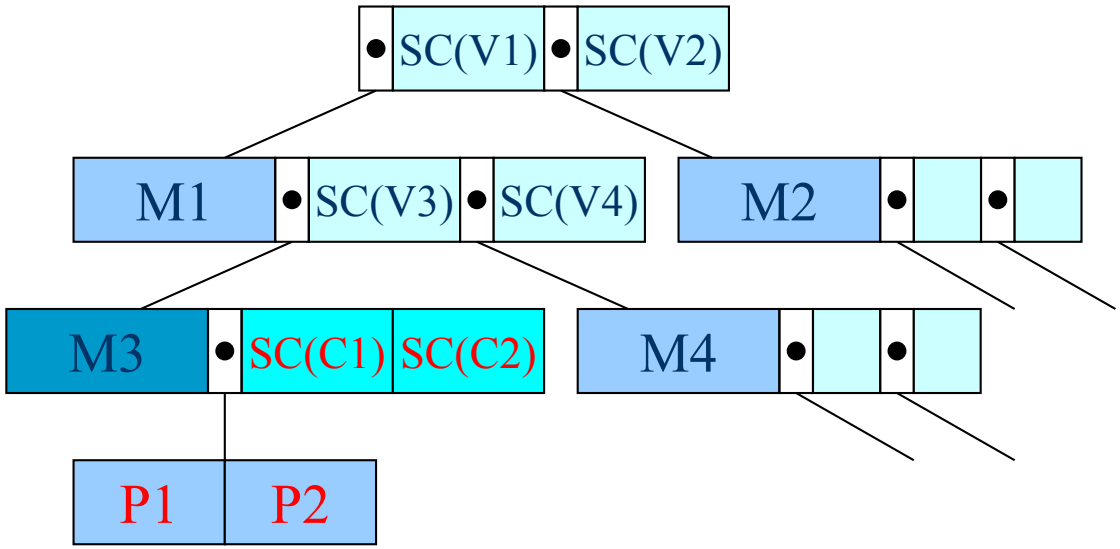
- Calculate V1 and V2 from R, SC(V1) and SC(V2)
- Calculate V3 and V4 from M1, SC(V3) and SC(V4)

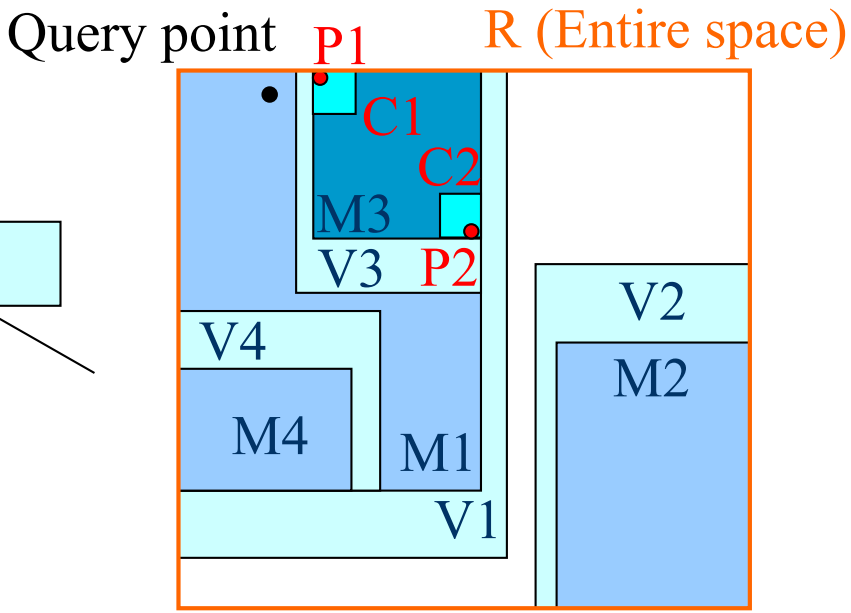




Search Algorithm

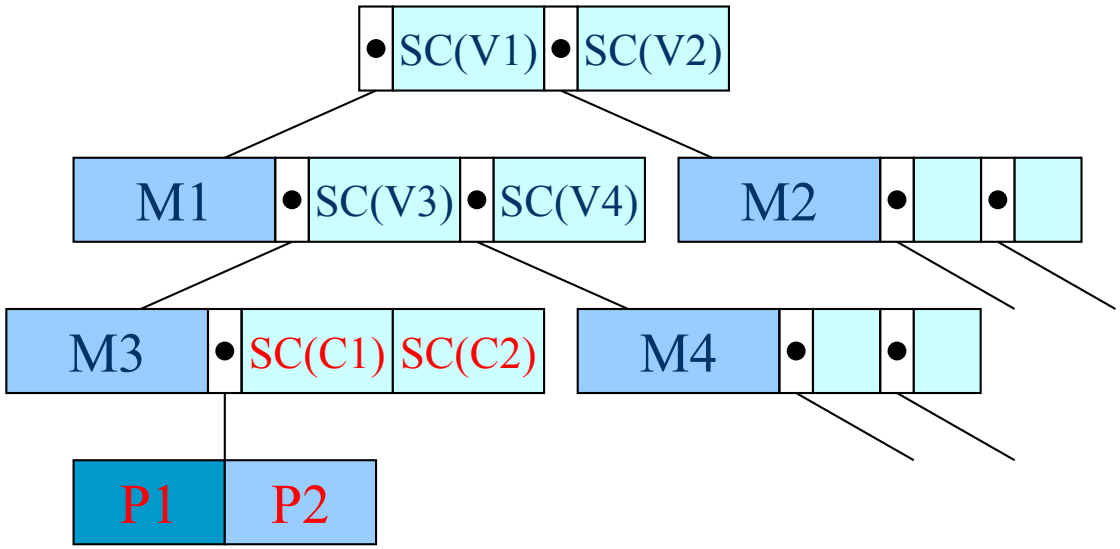
- Calculate V1 and V2 from R, SC(V1) and SC(V2)
- Calculate V3 and V4 from M1, SC(V3) and SC(V4)
- Calculate C1 and C2 from M3, SC(C1) and SC(C2)

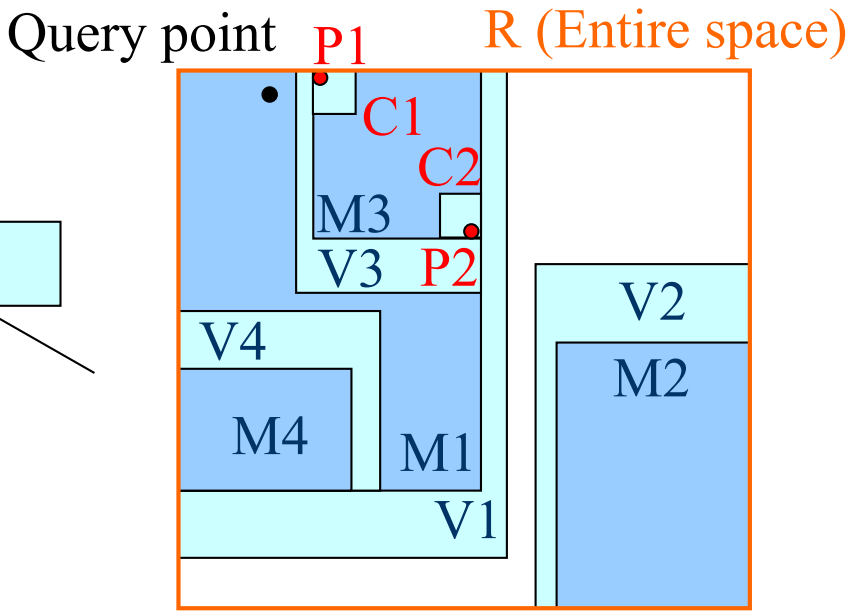




Search Algorithm

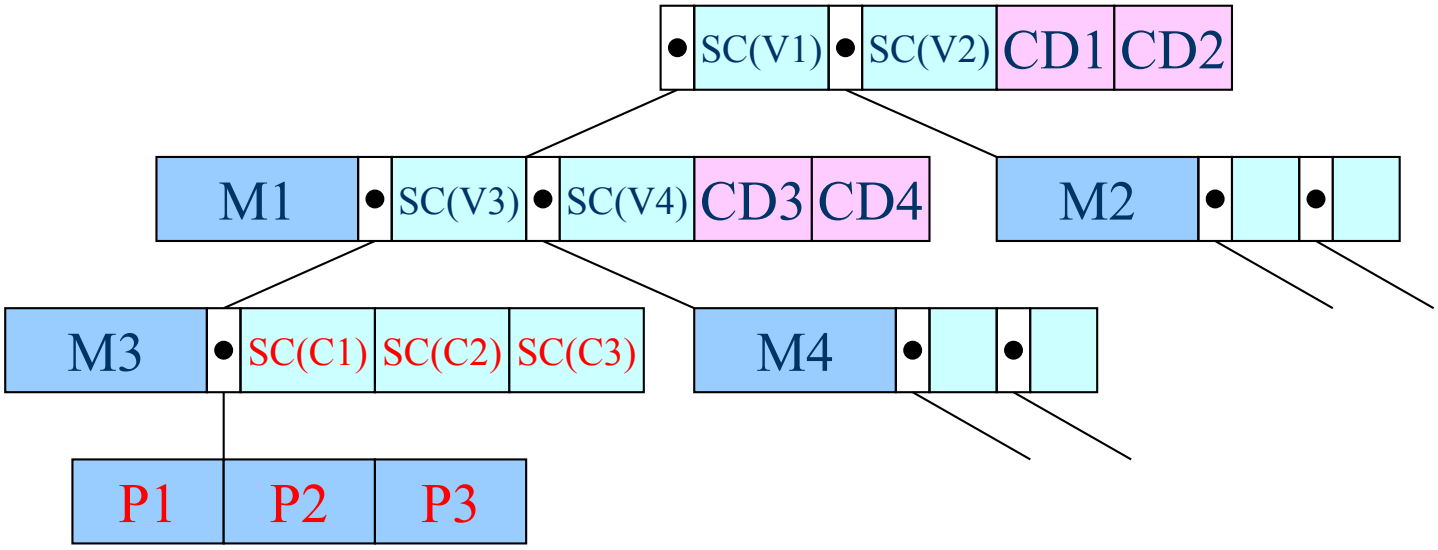
- Calculate V1 and V2 from R, SC(V1) and SC(V2)
- Calculate V3 and V4 from M1, SC(V3) and SC(V4)
- Calculate C1 and C2 from M3, SC(C1) and SC(C2)
- Access to P1



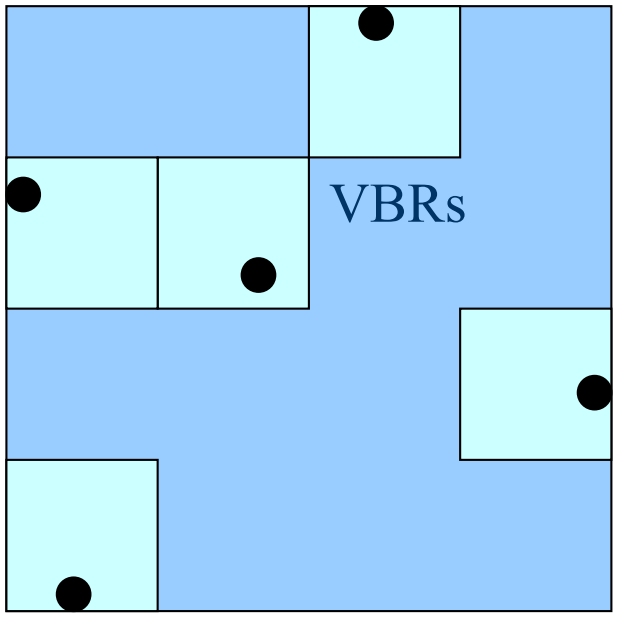


Update Algorithm

- Basic idea:
  - Based on the update algorithm of the SR-tree, but:
  - Needs to update subspace codes

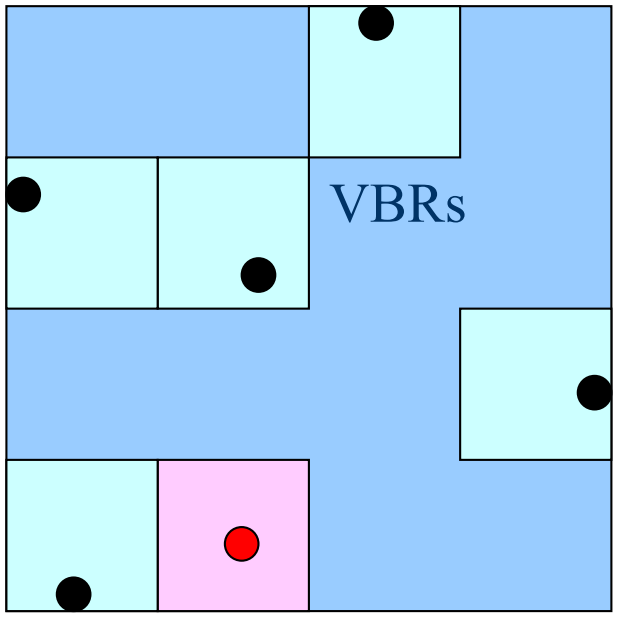


Code Calculation



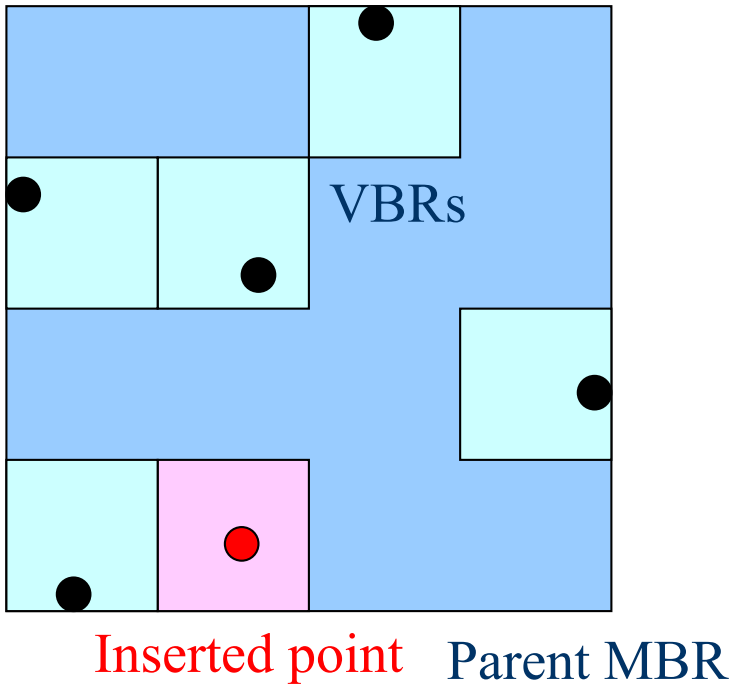
Code Calculation

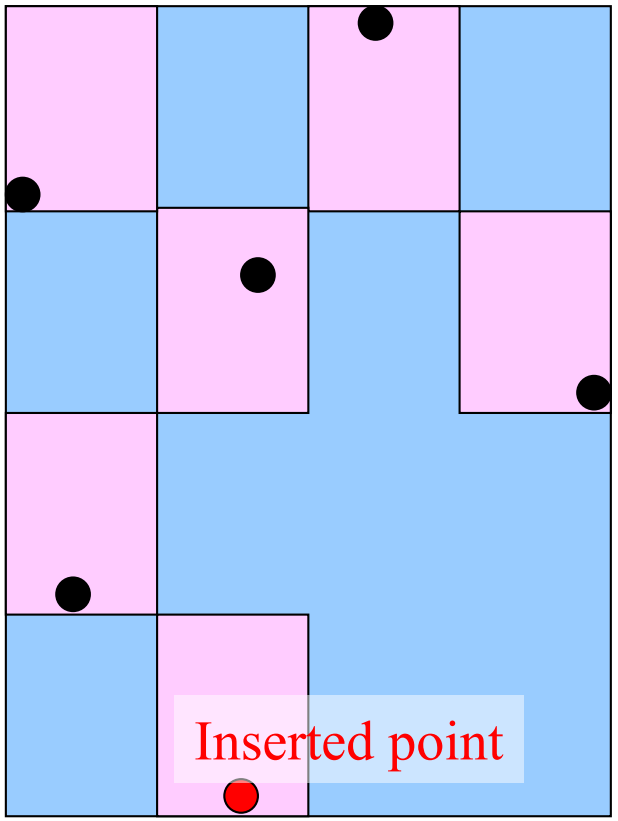
- If parent MBR does not change, calculate the subspace code for the inserted data object.



Code Calculation

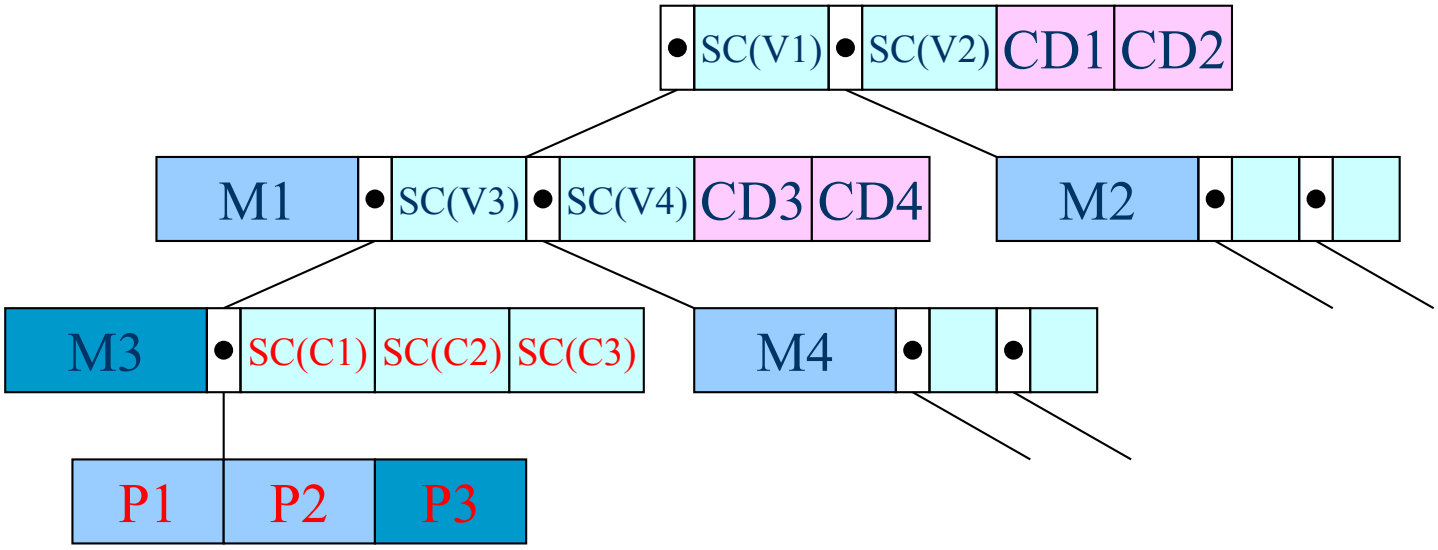
- If parent MBR does not change, calculate the subspace code for the inserted data object.
- If parent MBR changes, calculate all subspace codes





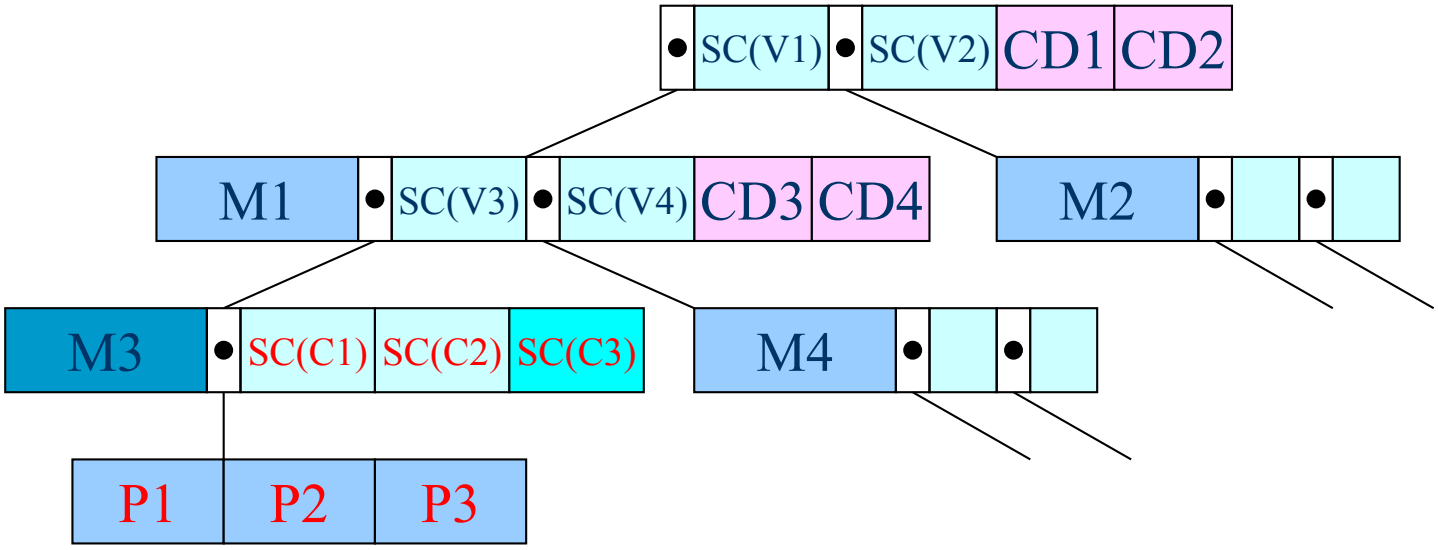
Update Algorithm

- Update data node and leaf node
  - Insert a new data object P3
  - Update M3



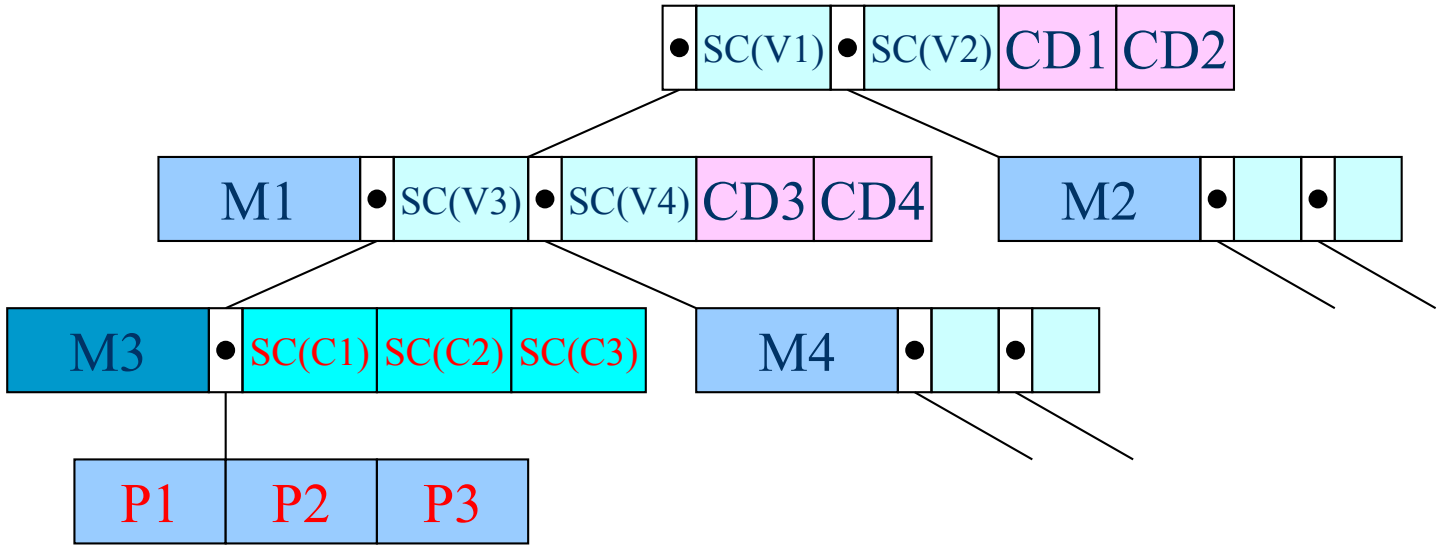
Update Algorithm

- Update data node and leaf node
  - Insert a new data object P3
  - Update M3
  - If M3 does not change, calculate SC(C3).



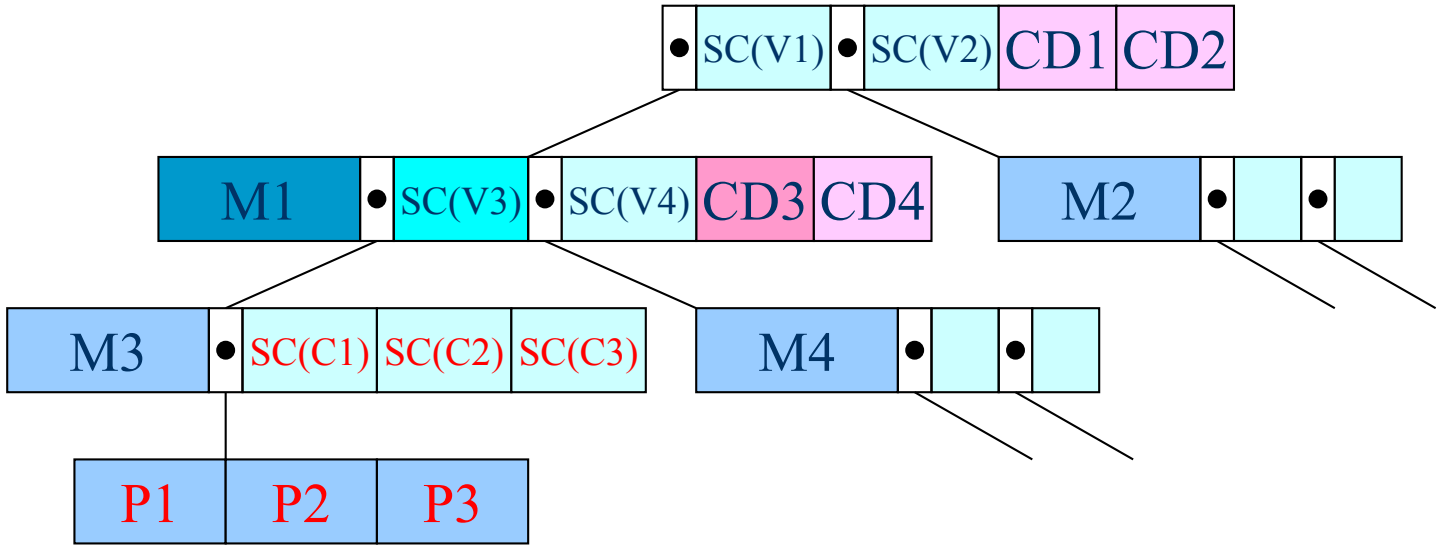
Update Algorithm

- Update data node and leaf node
  - Insert a new data object P3
  - Update M3
  - If M3 does not change, calculate SC(C3).
  - If M3 changes, calculate SC(C1), SC(C2) and SC(C3).



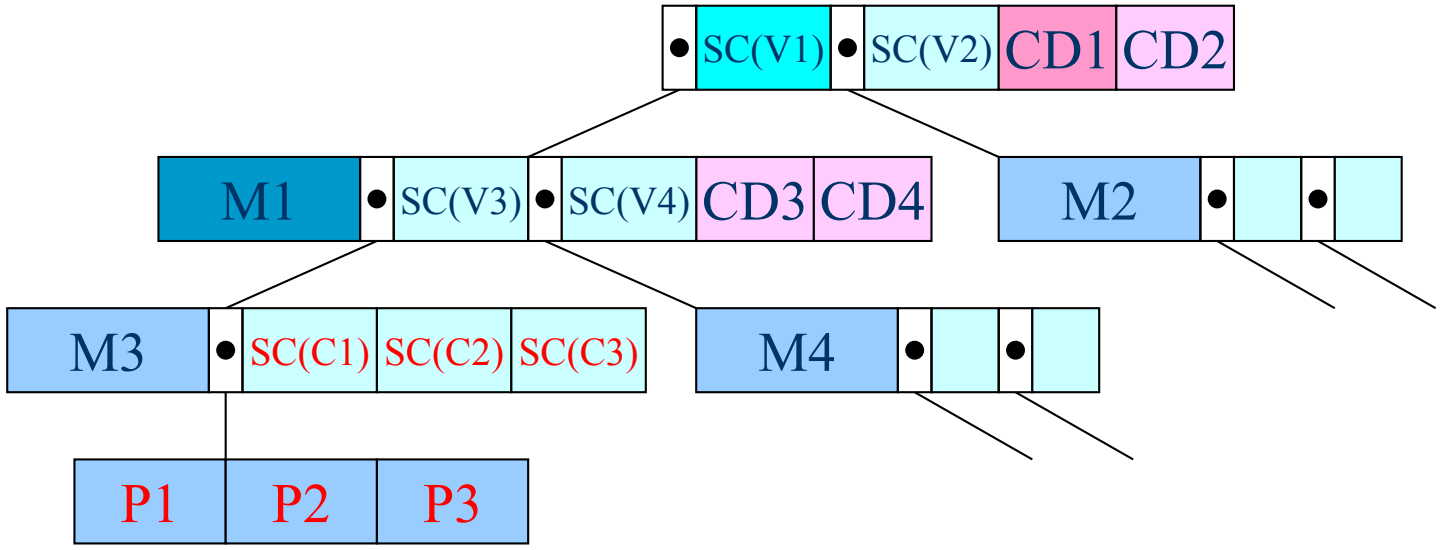
Update Algorithm

- Update intermediate node
  - If M3 changes, update M1.
  - If M3 changes but M1 does not change, calculate SC(V3).
  - If M1 changes, calculate SC(V3), SC(V4).
  - Calculate CD3



Update Algorithm

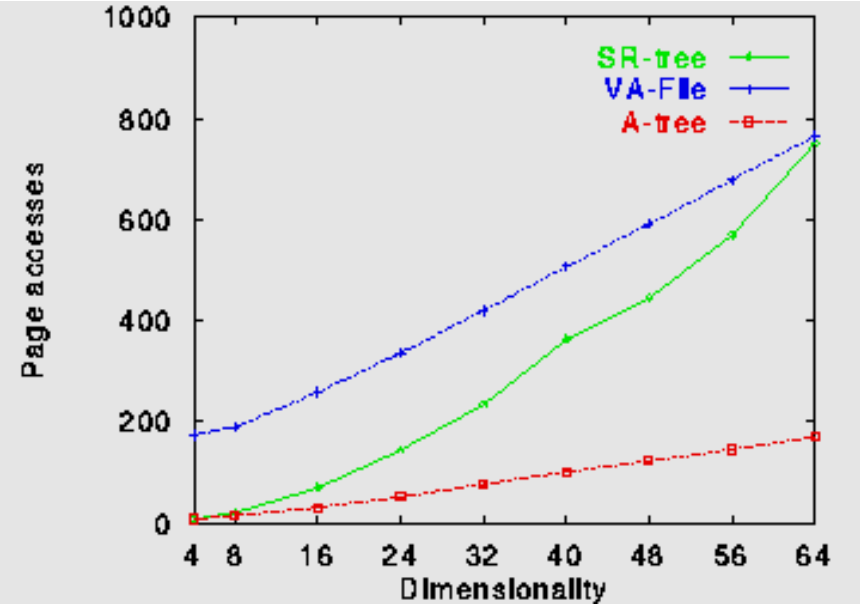
- Update root node
  - If M1 changes, calculate SC(V1)
  - Calculate CD1



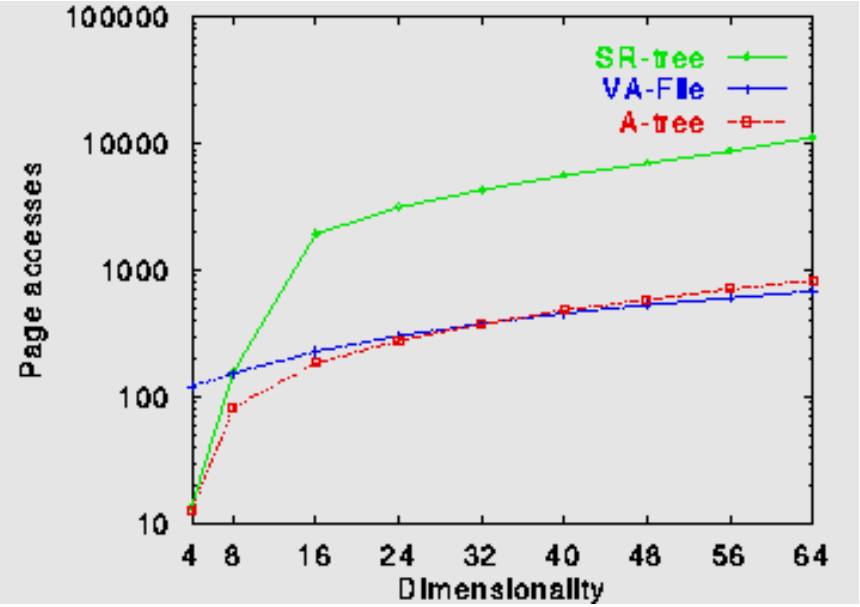
Performance Test

- Data sets: real data set (hue histogram image data), uniformly distributed data set, cluster data set.
- Data size: 100,000
- Dimension: varies from 4 to 64
- Page size: 8 KB
- 20-nearest neighbor queries
- Evaluation is based on the average for 1,000 insertion or query points.
- CPU: 296 MHz
- Code length:
  - The code length that gave the best performance was chosen.
  - A-tree: code length varies from 4 to 12.
  - VA-File: code length varies from 4 to 8 according to [18].

Search Performance



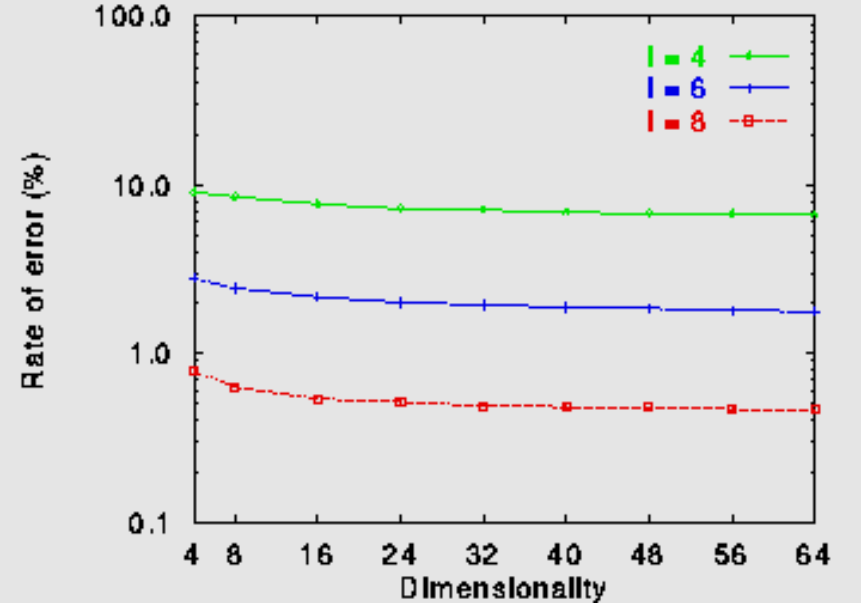
Real data



Uniformly distributed data

- A-tree gives significantly superior performance!
- 77% reduction in number of page accesses for 64-dimensional real data
- Relative approximation
  - Small entry size and large fanout ⇒ low IO cost

Influence of Code Length



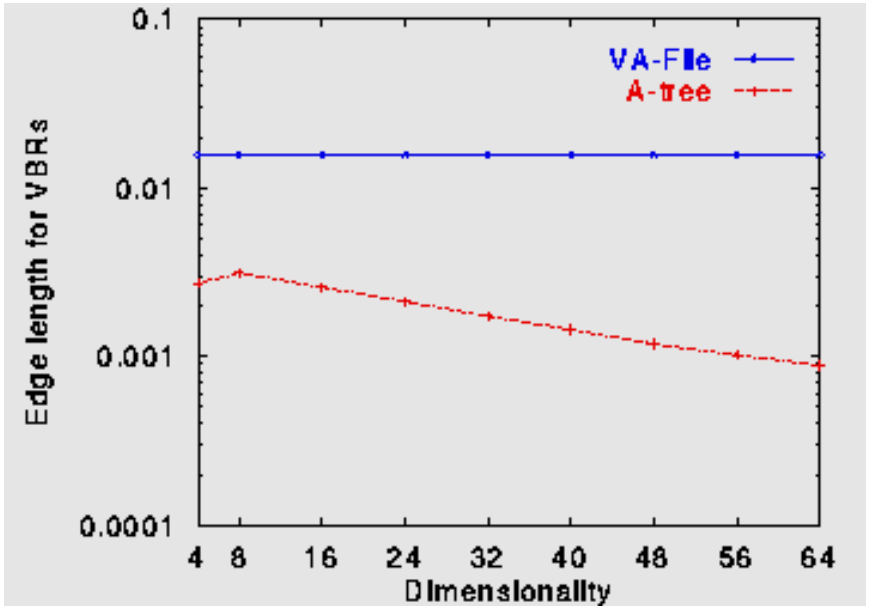
- Approximation error ε: error of the distance between p and V_i during a search

$$\varepsilon = (1-r)\cdot 100, \quad r = \frac{1}{S}\sum_{i=1}^{S}\frac{\|p, V_i\|}{\|p, M_i\|}$$

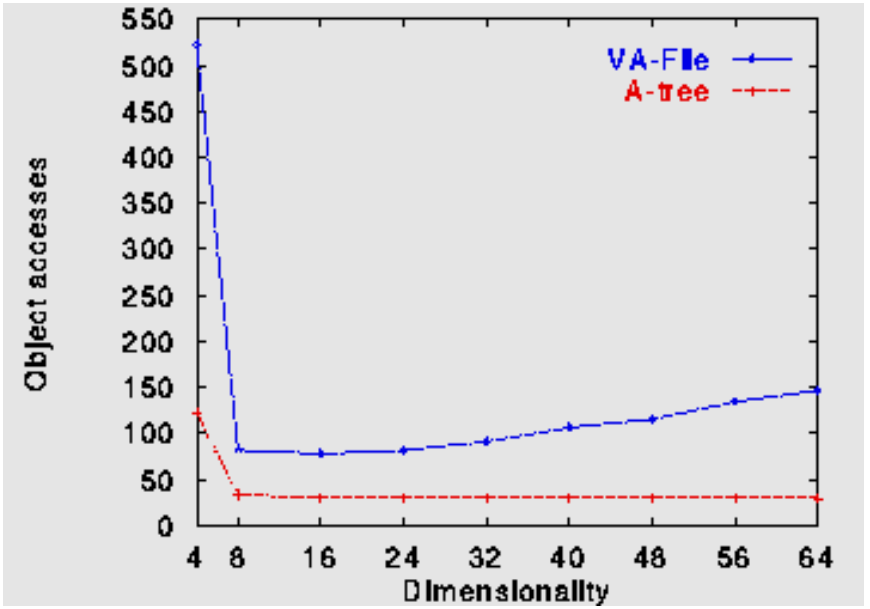
p: query point, S: the number of visited VBRs,
V_i: visited VBRs, M_i : the MBRs corresponding to V_i

- Optimum code length depends on dimensionality and data distribution

VA-File/A-tree Comparison



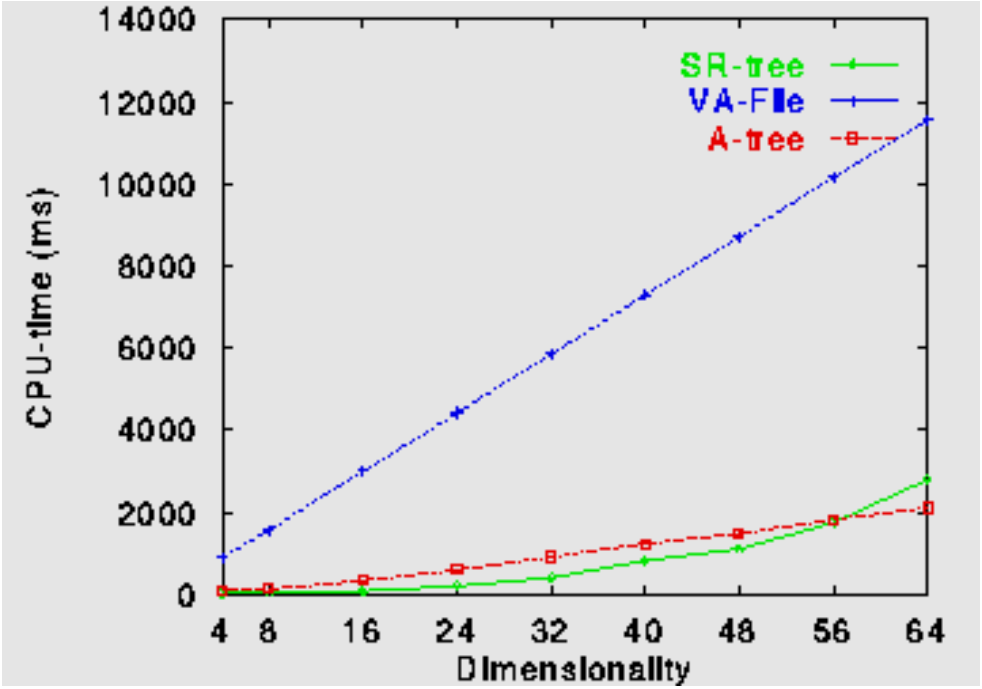
Edge length of VBRs/cells



Number of data object accesses

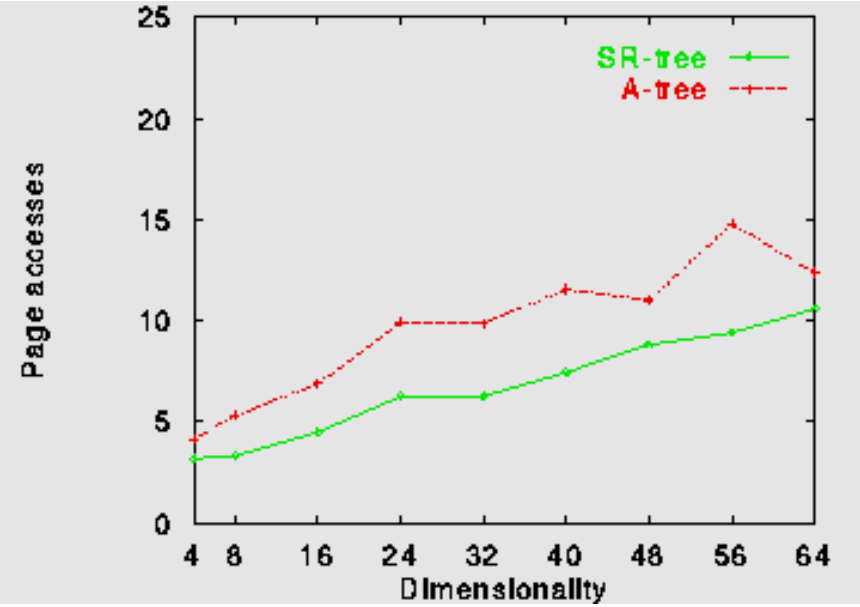
- VA-File (absolute approximation)
  - approximated using the entire space ⇨ edge length 2^{-l}
- A-tree (relative approximation)
  - approximated using parent MBR ⇨ smaller VBR size, fewer object accesses

CPU-time

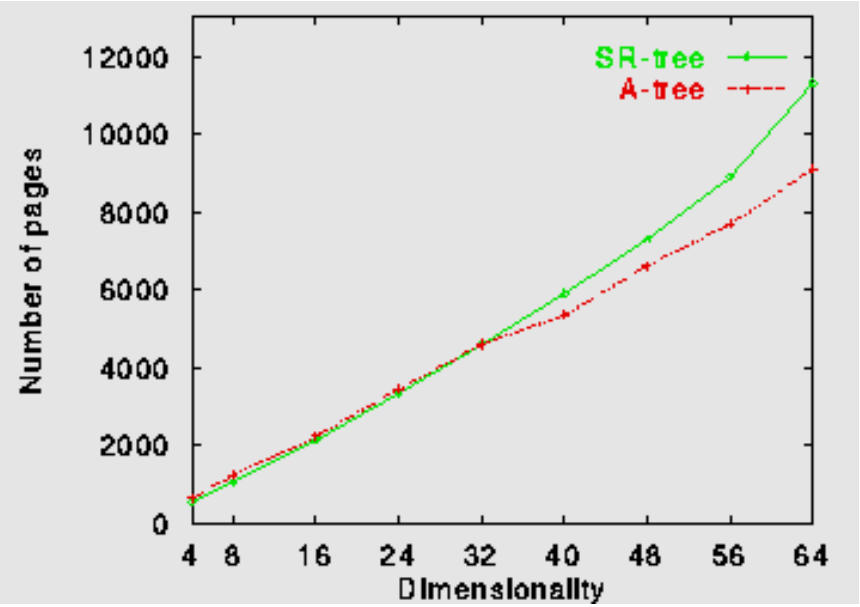


- CPU-time for real data
  - Similar to the SR-tree and outperforms the VA-File
- VA-File
  - Calculates the approximated position coordinate for all objects
- A-tree
  - Reducing node accesses leads to low CPU cost.

Insertion and Storage Cost



Insertion cost



Storage cost

- Increase in the insertion cost is modest.
- About 20% less storage cost for 64-dimensional data

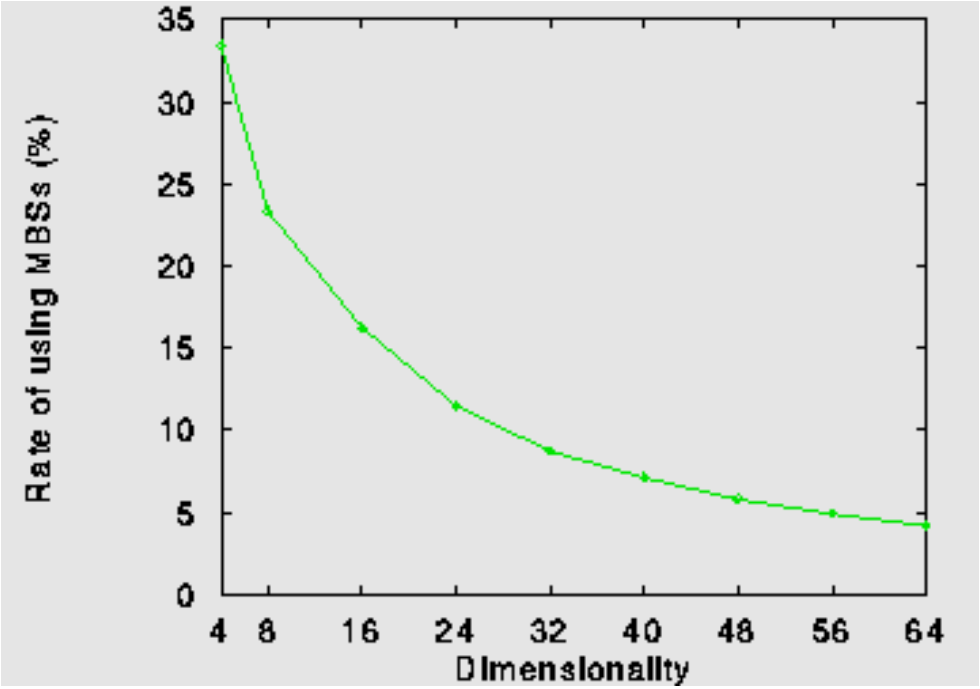
(1) VBRs need only small storage volumes.

(2) The number of index nodes is extremely small.

Conclusions

- The A-tree offers excellent search performance for high-dimensional data
  - Relative approximation
    - MBRs and data objects in child nodes are approximated based on parent MBR.
  - About 77% reduction in the number of page accesses compared with VA-File and SR-tree
- Future work
  - Cost model for finding optimum code length

Contribution of MBSs for Pruning



- SR-tree contains both MBRs and MBSs but:

  the frequency of the usage of MBSs decreases as dimensionality increases.