Chương 3: Các cấu trúc dữ liệu đa chiều

Nguyễn Thị Oanh
Bộ môn HTTT – Viện CNTT & TT
oanhnt@soict.hut.edu.vn

Plan

- Phương pháp lưu/truy nhập DL dạng điểm (Point Access Methods)
  - k-D trees
  - Point Quadtrees
  - MX-Quadtrees
- Phương pháp lưu/truy nhập DL dạng vùng (chữ nhật) (Spatial Access Methods):
  - R-trees

1. k-D trees

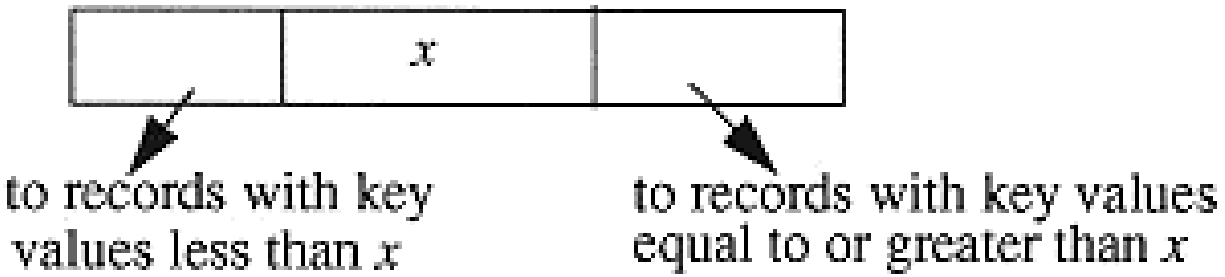
k-D trees

- Dành lưu trữ dữ liệu **điểm đa chiều** (k-dimension)
  - 2-tree: lưu DL điểm 2 chiều
  - 3-tree: lưu DL điểm 3 chiều
  - …
  - Mỗi điểm là vector có k phần tử

- **Không** lưu **DL vùng**

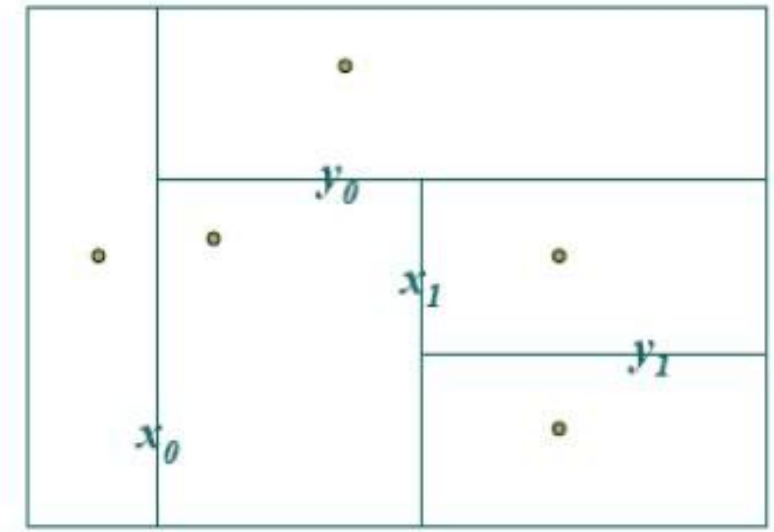
k-D trees

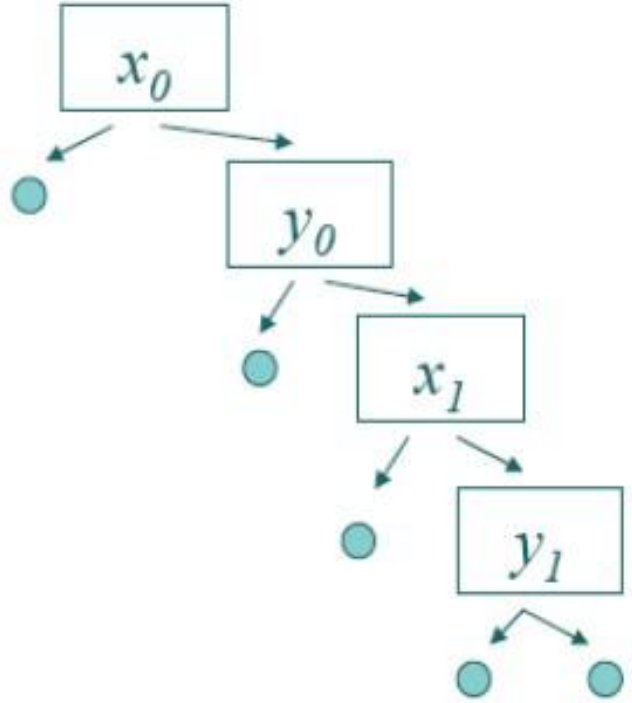
- Là mở rộng của cây nhị phân



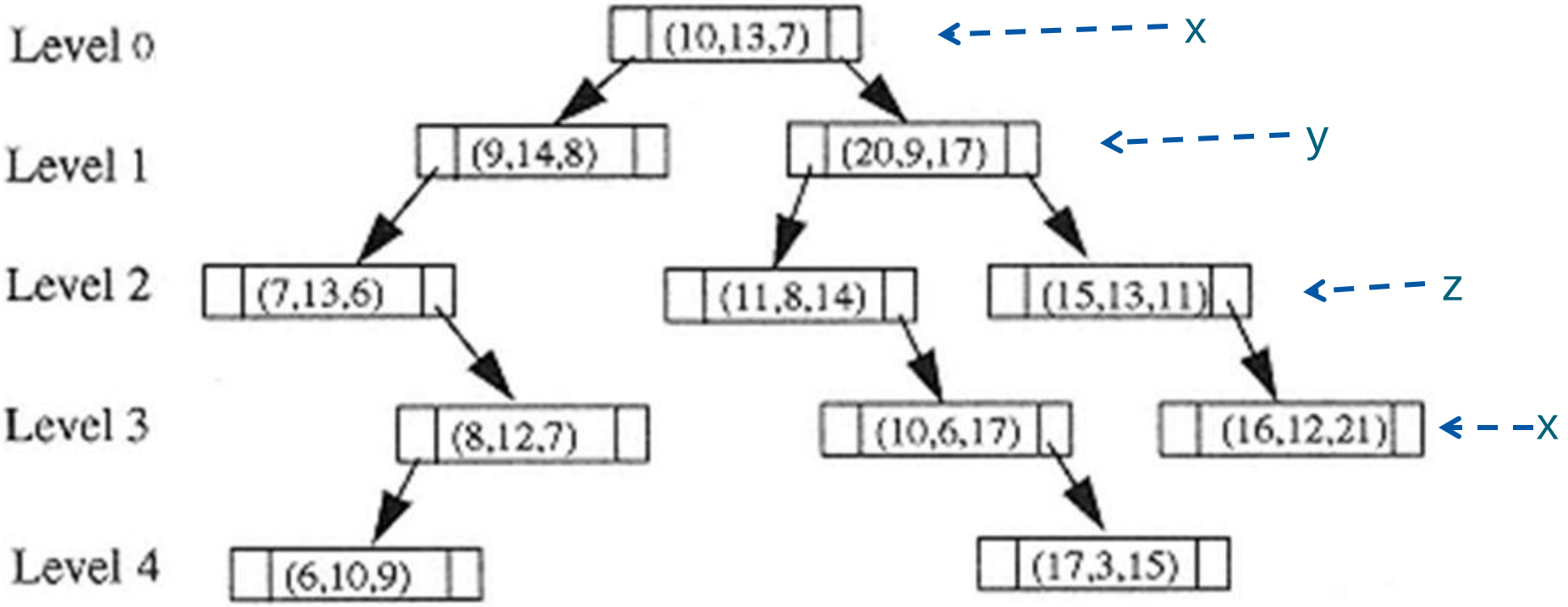
- Ở mỗi mức, các bản ghi sẽ được chia theo giá trị của 1 chiều nhất định.
  - Mức 0: giá trị chiều 0
  - Mức 1: giá chị chiều 1, …
  - Mức *k-1*: giá trị chiều *k-1*
  - Mức *k:* giá trị chiều *0, …*

VD: 2-D trees





VD: 3-D trees



Cây được xây dựng phụ thuộc vào **thứ tự các điểm được đưa vào**

2-D trees

- Cấu trúc 1 nút:

INFO	XVAL	YVAL
LLINK	RLINK	

- Định nghĩa: 2-d tree là cây nhị phân thỏa mãn:
  - Nếu nút N ở mức chẵn :

$$\forall M \in N.LLINK : M.XVAL < N.XVAL \,\&$$

$$\forall P \in N.RLINK : P.XVAL \geq N.XVAL$$

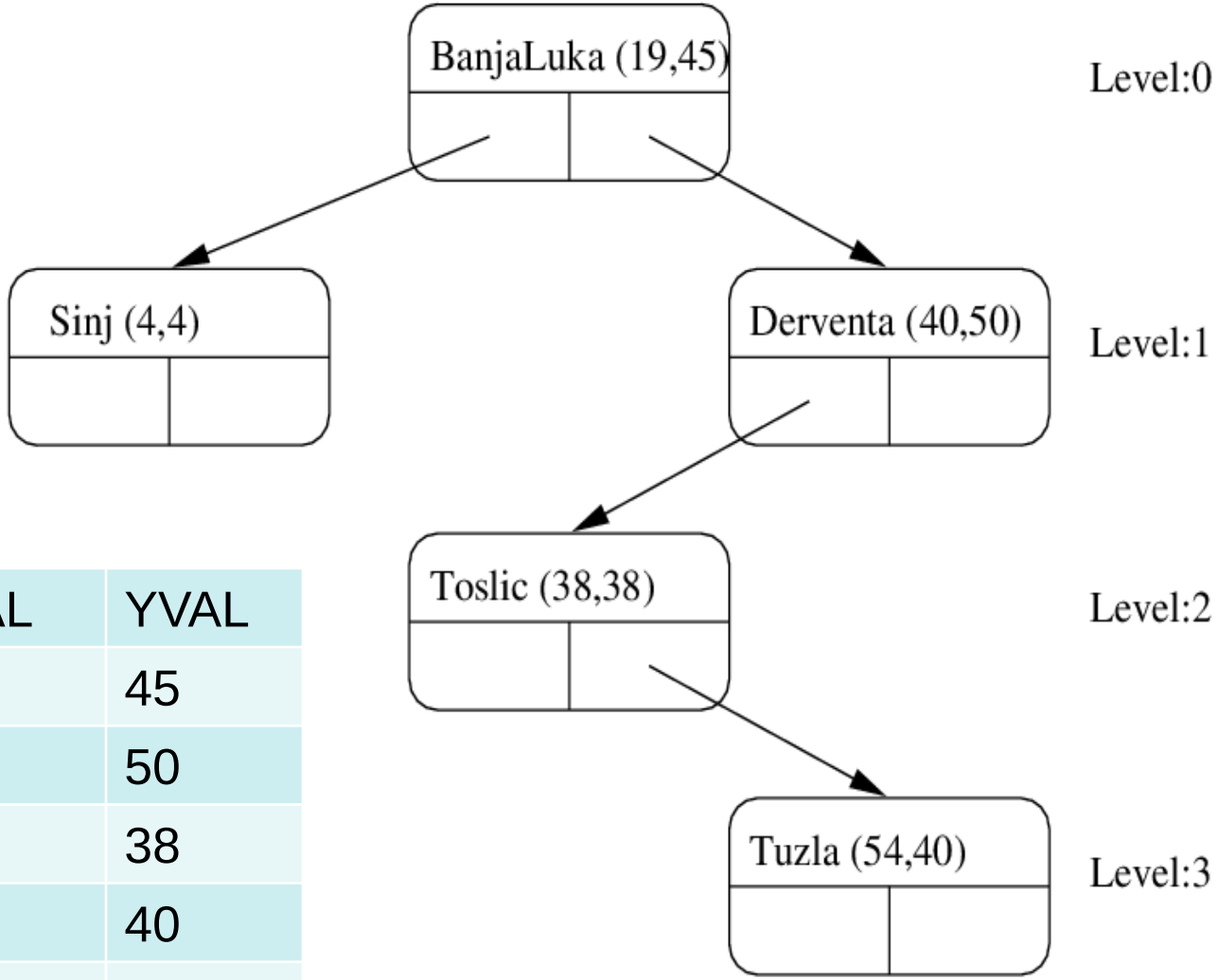
  - Nếu nút N ở mức lẻ:

$$\forall M \in N.LLINK : M.YVAL < N.YVAL \,\&$$

$$\forall P \in N.RLINK : P.YVAL \geq N.YVAL$$

2-D trees

- Ví dụ:



- Thứ tự insert:

INFO	XVAL	YVAL
Banja Luka	19	45
Derventa	40	50
Toslic	38	38
Tuzla	54	40
Sinji	4	4

Thêm/ Tìm kiếm trong 2-D trees

- Nút cần thêm: P(info, x, y)
- Lặp:
  - Nút đang duyệt: N
  - Nếu N.XVAL = x và N.YVAL = y thì ghi đè N và kết thúc
  - Nếu N ở **mức chẵn** (0, 2, 4, …):
    - Nếu **x < N.XVAL** thì duyệt cây bên trái,
    - nếu không duyệt cây con bên phải
  - Nếu N ở **mức lẻ** (1, 3, 5, …):
    - Nếu **y < N.YVAL** thì duyệt cây bên trái,
    - nếu không duyệt cây con bên phải

Xóa nút trên 2-D trees

- T: 2-D tree
- Nút cần xóa (XVAL, YVAL) = (x, y)
- Thuật toán:
  - Tìm N: **N.XVAL = x & N.YVAL = y**
  - Nếu N là **nút lá**: đặt LLINK or RLINK của cha N về NULL và giải phóng N. Kết thúc
  - Nếu N là **nút trong**:
    - ***Tìm nút thay thế (R)*** ở trong 2 cây con (T_f *và* T_r)
    - ***Thay các giá trị*** không phải con trỏ bằng giá trị của R
    - Lặp để ***xóa R***

Tìm nút thay thế cho nút bị xóa ?

- Nếu xóa N ➔ tìm nút thay thế R: mọi nút thuộc cây con trái (***N.LLINK)*** / phải của N cũng thuộc cây con trái (***R.LLINK)*** / phải tương ứng của R:
  - Nếu nút N ở mức chẵn :

$$\forall M \in N.LLINK : M.XVAL < R.XVAL \;\&$$

$$\forall P \in N.RLINK : P.XVAL \geq R.XVAL$$

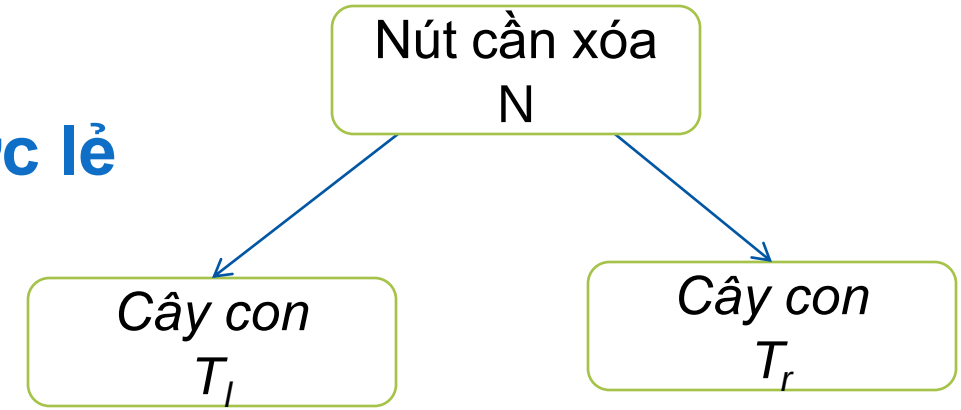
  - Nếu nút N ở mức lẻ:

$$\forall M \in N.LLINK : M.YVAL < R.YVAL \;\&$$

$$\forall P \in N.RLINK : P.YVAL \geq R.YVAL$$

Tìm nút thay thế cho nút bị xóa:

- Nếu N: **mức chẵn**
  - T_r : không rỗng ➔ nút R trong cây con T_r có giá trị ***XVAL nhỏ nhất*** là nút thay thế
  - T_r : rỗng ➔ tìm nút thay thế bên cây T_l **(How ?)**
    - **Tìm nút R' bên cây trái T_l có XVAL nhỏ nhất**
    - **N.RLINK = N.LLINK, N.LLINK = NULL**
- Tương tự nếu N ở **mức lẻ**



Truy vấn phạm vi trên 2-D trees

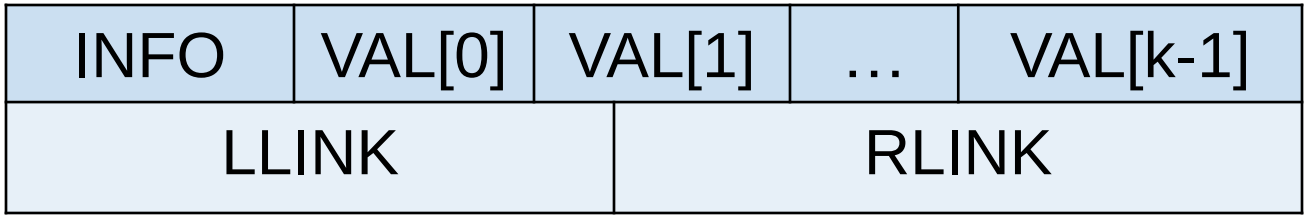
- Truy vấn phạm vi (*range query*):

  1 điểm **(x_c, y_c)** + 1 khoảng cách **r**

- Tìm các điểm (x,y) trên cây 2-D sao cho khoảng cách từ đó đến (x_c, y_c) <= ***r***
  - L1: $x_c - r \le x \le x_c + r \qquad y_c - r \le y \le y_c + r$
  - L2 (Euclide)
  - …

2-D trees ➔ k-D tree

- $p(x_0, x_1, .., x_{k-1})$



- N: 1 nút thuộc k-D tree nếu
  - Mọi nút M thuộc cây bên trái của N: **M.VAL[i] <N.VAL[i]**
  - Mọi nút P thuộc cây bên phải của N: **P.VAL[i] >=N.VAL[i]**

  i = level(N) mod k

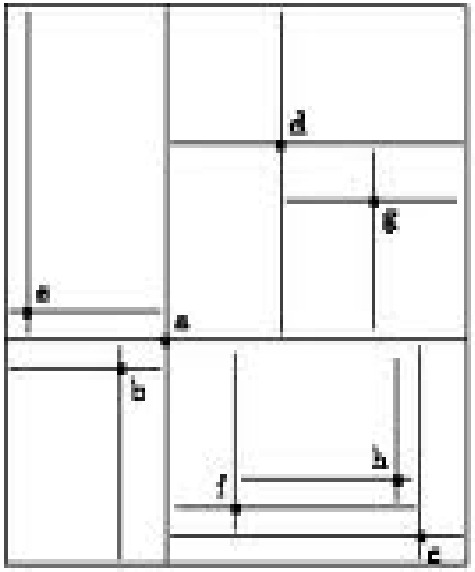
k-D trees: Lưu ý

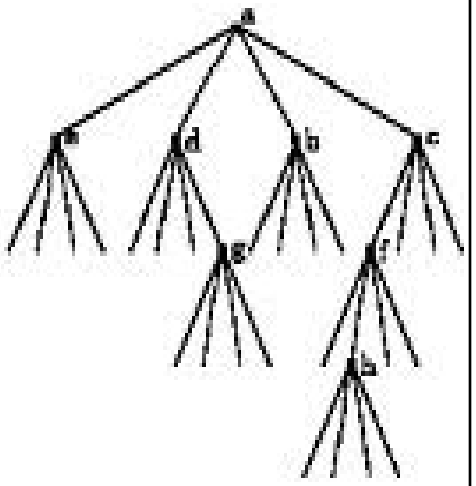
- Cây không cân bằng
- Tùy thuộc vào thứ tự dữ liệu được insert vào
- Một số biến thể:
  - k-D-B-tree: k-D tree + cây cân bằng (B-tree)
  - LSD-tree (Local Split Decision tree): đánh chỉ mục 2 mức: main memory + disk
  - VA-file (Vector Approximation file)

2. Cây tứ phân dạng điểm (Point Quadtrees)

Cây tứ phân dạng điểm

- Mỗi điểm trong cây sẽ chia **1 vùng thành 4 vùng con** theo cả 2 chiều ngang và dọc (N.XVAL & N.YVAL):
  - NW (Northwest)
  - SW (Southwest)
  - NE (Northeast)
  - SE (Southeast)



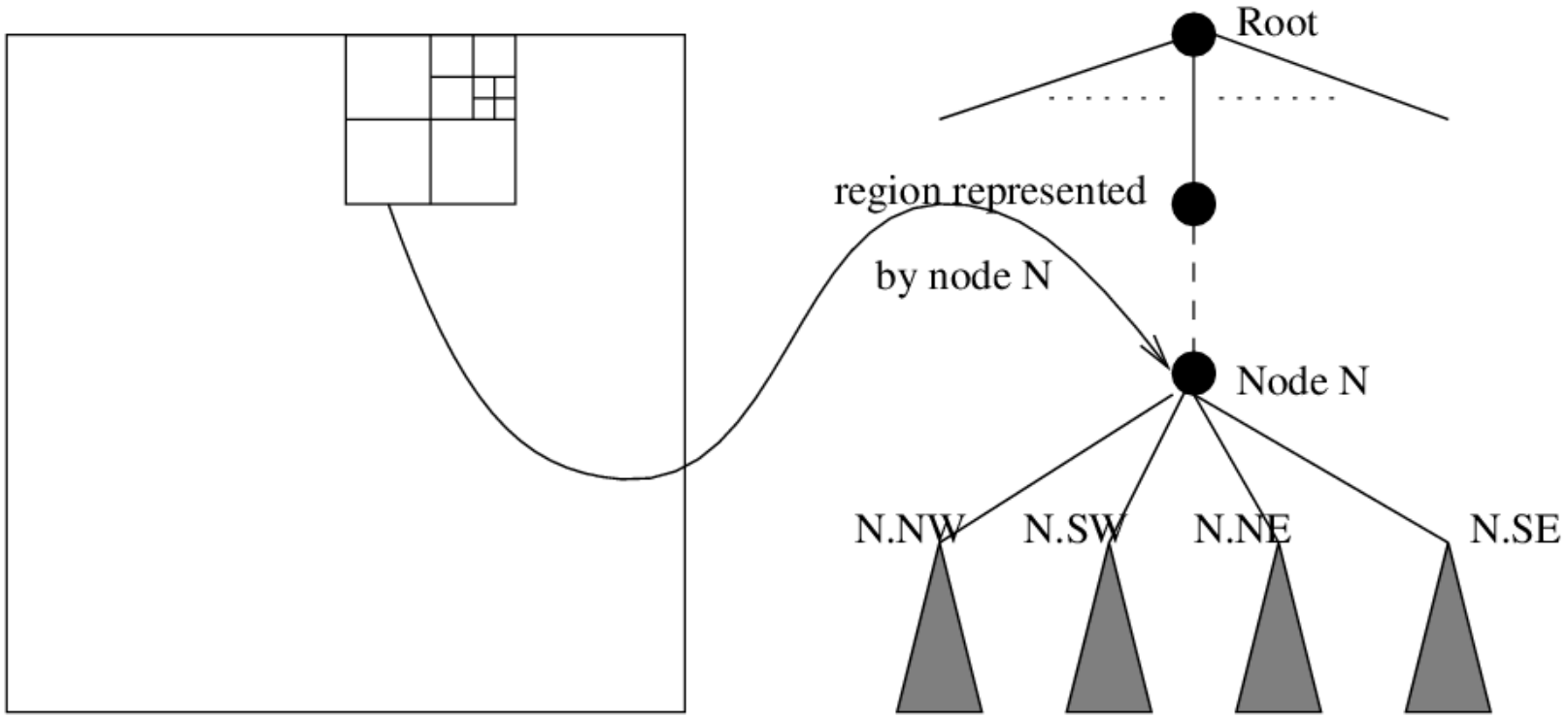


point quadtree

YVAL		
	NW	NE
	SW	SE
	XVAL	

Cây tứ phân dạng điểm

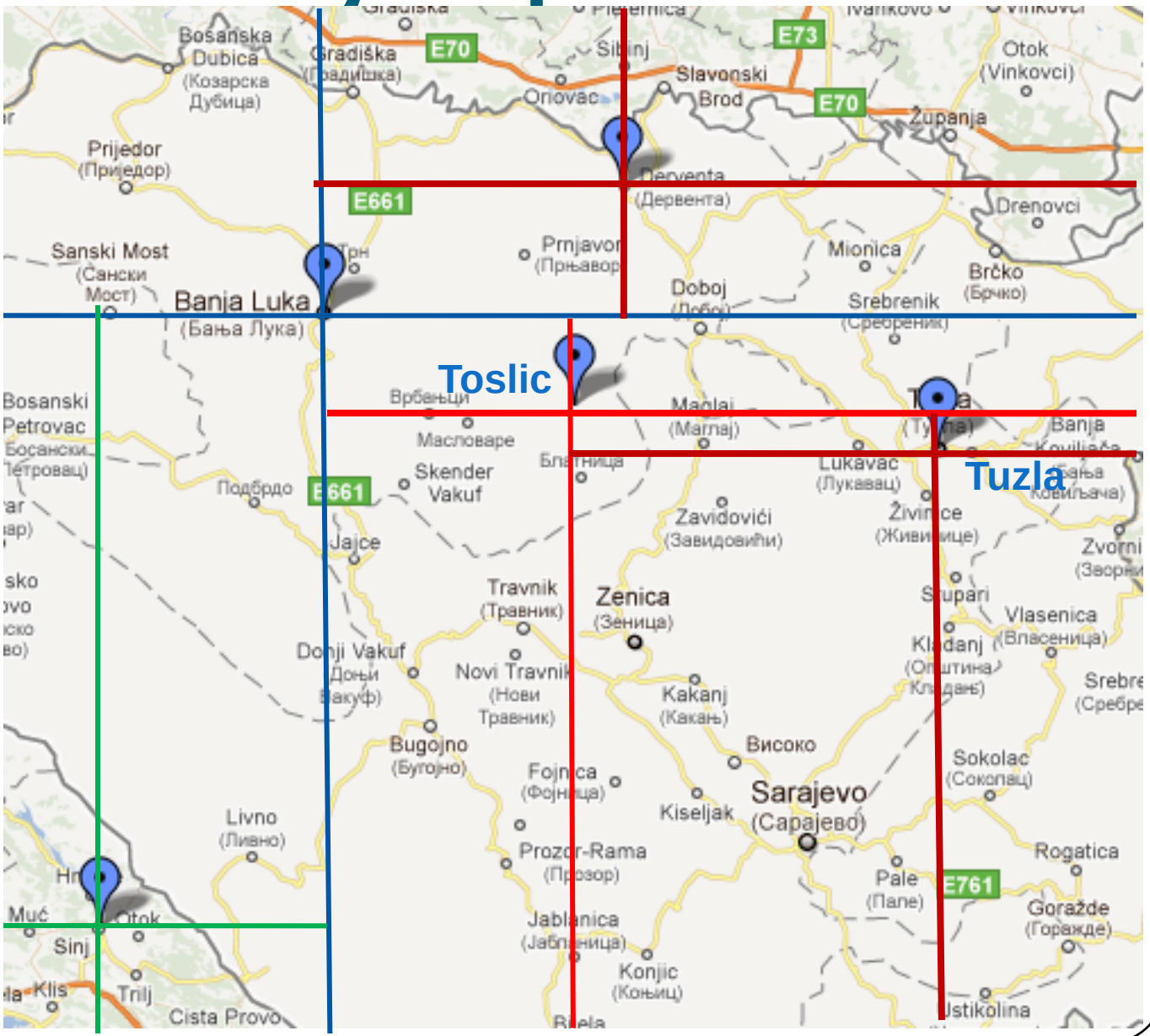
- Mỗi nút trong cây tứ phân ngầm biểu diễn 1 vùng:



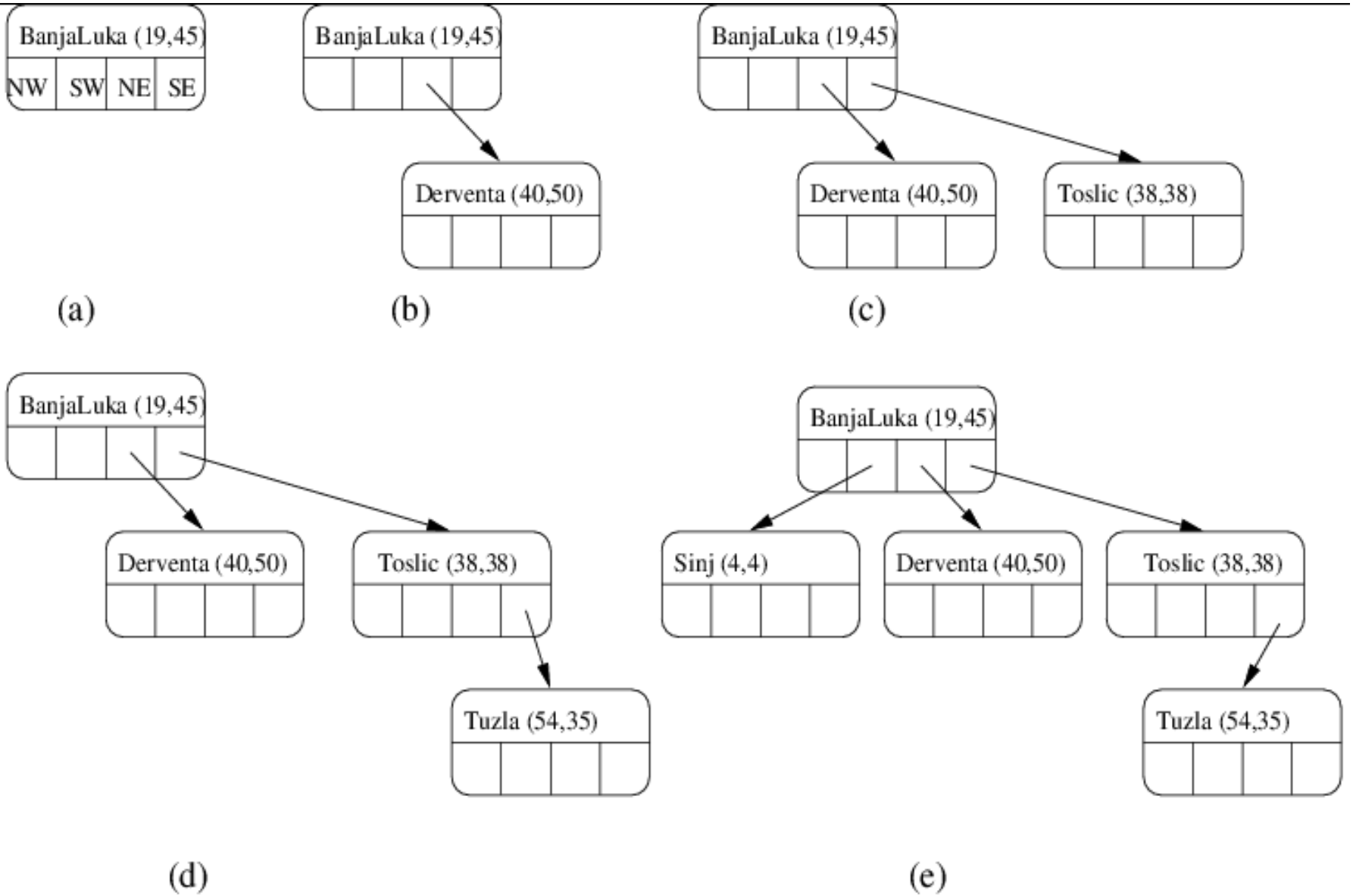
Thêm DL vào cây tứ phân

- Banja Luka

  (19, 45)
- Derventa (40, 50)
- Toslic (38, 38)
- Tuzla (54, 40)
- Sinji (4,4)



Thêm DL vào cây tứ phân



Xóa DL trong cây tứ phân

- Nút lá: đơn giản
  - Thiết lập lại con trỏ ở nút cha = NULL và giải phóng nút cần xóa
- Nút trong: phức tạp
  - Cần tìm nút thay thế ➔ ?????
  - VD: Xóa nút gốc trong ví dụ trên ?

Tìm nút thay thế khi xóa nút trong

- Nút xóa : N
- Nút thay thế R đảm bảo:

$$\forall R1 \in N.NW \Rightarrow R1 \in R.NW$$
$$\forall R2 \in N.SW \Rightarrow R2 \in R.SW$$
$$\forall R3 \in N.NE \Rightarrow R3 \in R.NE$$
$$\forall R4 \in N.SE \Rightarrow R4 \in R.SE$$

- Ex. N: « Banja Luka » ➔ R: « Toslic »
- *Không phải lúc nào cũng tìm được nút thay thế*

Xóa DL trong cây tứ phân (..)

- Xóa nút trong ➔ tìm nút thay thế

➔chèn lại các nút trỏ bởi NW, SW, NE, SE

➔Trường hợp tồi nhất: tất cả các nút bị thay đổi!!!!!

Truy vấn phạm vi trên cây tứ phân

RangeQueryPointQuadtrees(T:newqtnodetype, C: cycle)

{ if *region(T)* $\cap$ *C* = $\varnothing$ then Halt

else

(a) if *(T.XVAL, T.YVAL)* $\in$ *C* then **print**(*T.XVAL, T.YVAL*);

(b) **RangeQueryPointQuadtrees**(T.NW, C);

(c) **RangeQueryPointQuadtrees**(T.SW, C);

(d) **RangeQueryPointQuadtrees**(T.NE, C);

(e) **RangeQueryPointQuadtrees**(T.SE, C);

}

3. MX-Quadtrees

MX-Quadtrees

- Hình dáng cây không phụ thuộc vào:
  - Số nút thêm vào
  - Thứ tự thêm vào
- Cho phép xóa và truy vấn hiệu quả

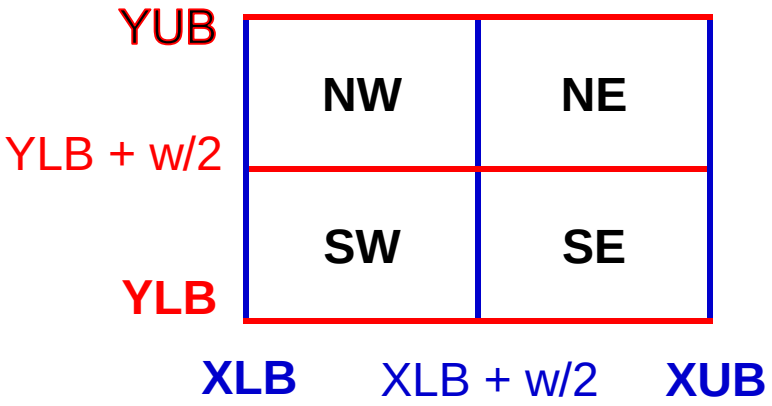
MX-Quadtrees

- Dữ liệu được chia theo lưới $2^k \times 2^k$
- ***k*** tự chọn, nhưng sau khi chọn thì ***k*** phải không được thay đổi

- Cấu trúc nút:
  - Tương tự cây tứ phân dạng điểm
  - Thông tin về vùng biểu diễn (XLB, XUB, YLB, YUB)
  - Nút gốc (root) : XLB = 0, XUB = 2^k, YLB = 0, YUB = 2^k

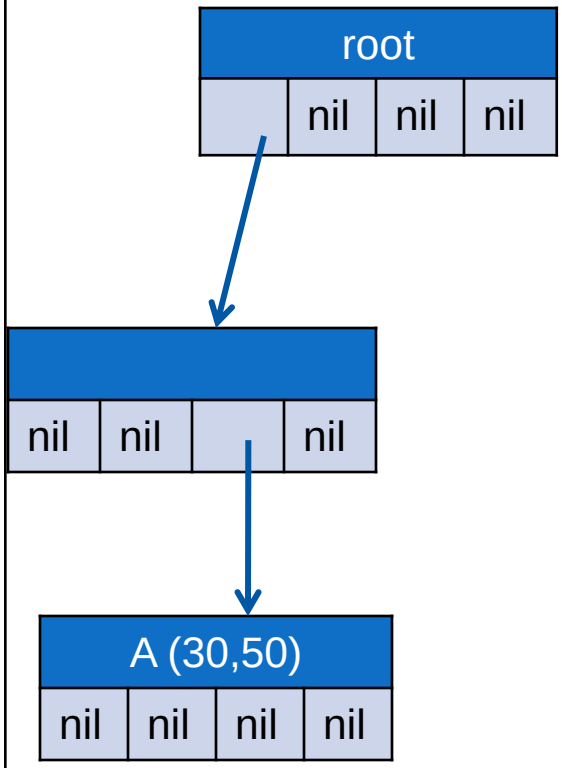
MX-Quadtrees

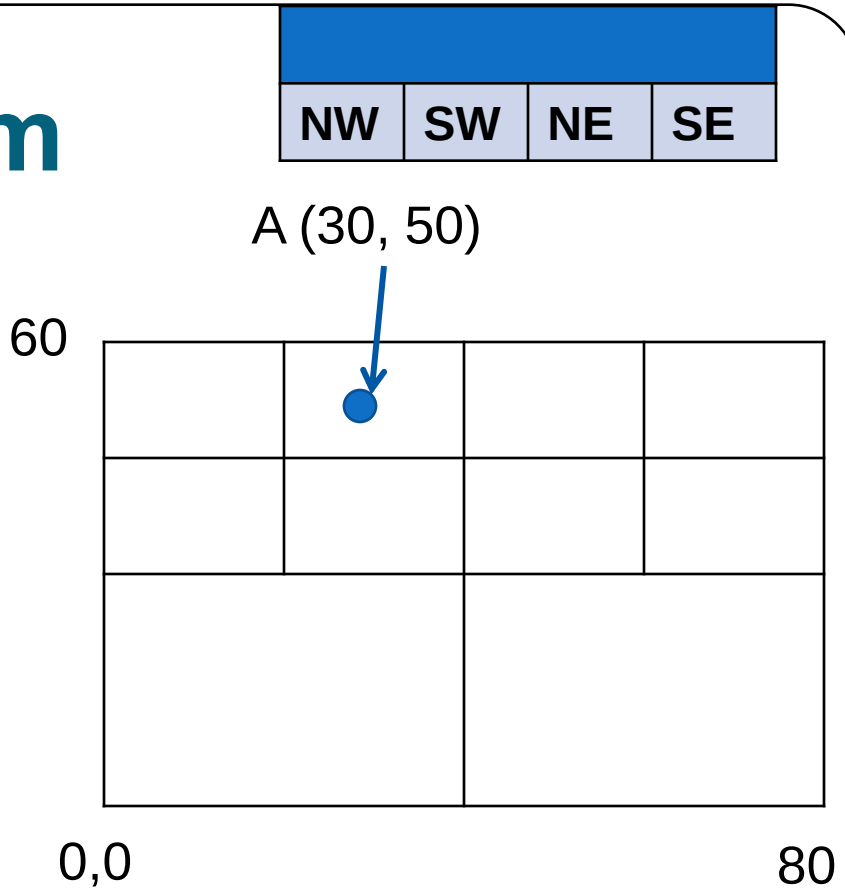
- Cấu trúc nút (…):
  - Các nút con của N (với ***w = N.XUB – N.XLB***):



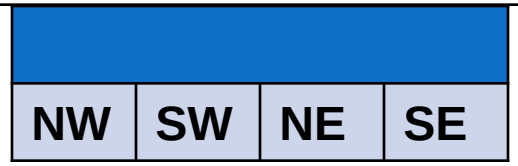
Child	XLB	XUB	YLB	YUB
NW	N.XLB	N.XLB+$\frac{w}{2}$	N.YLB+$\frac{w}{2}$	N.YLB+w
SW	N.XLB	N.XLB+$\frac{w}{2}$	N.YLB	N.YLB+$\frac{w}{2}$
NE	N.XLB+$\frac{w}{2}$	N.XLB+w	N.YLB+$\frac{w}{2}$	N.YLB+w
SE	N.XLB+$\frac{w}{2}$	N.XLB+w	N.YLB	N.YLB+$\frac{w}{2}$

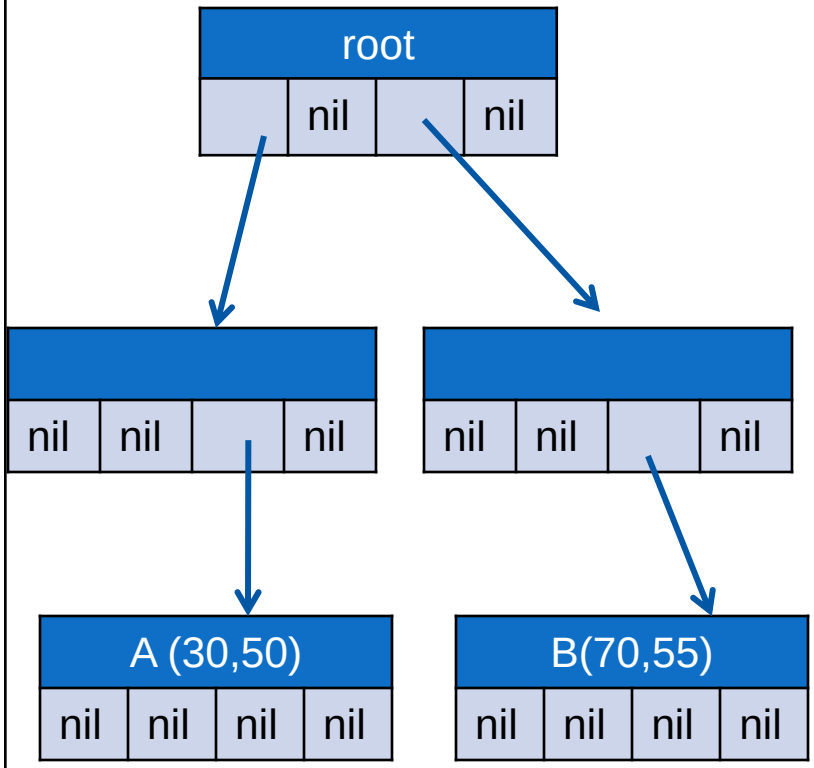
MX-Quadtree: Thêm

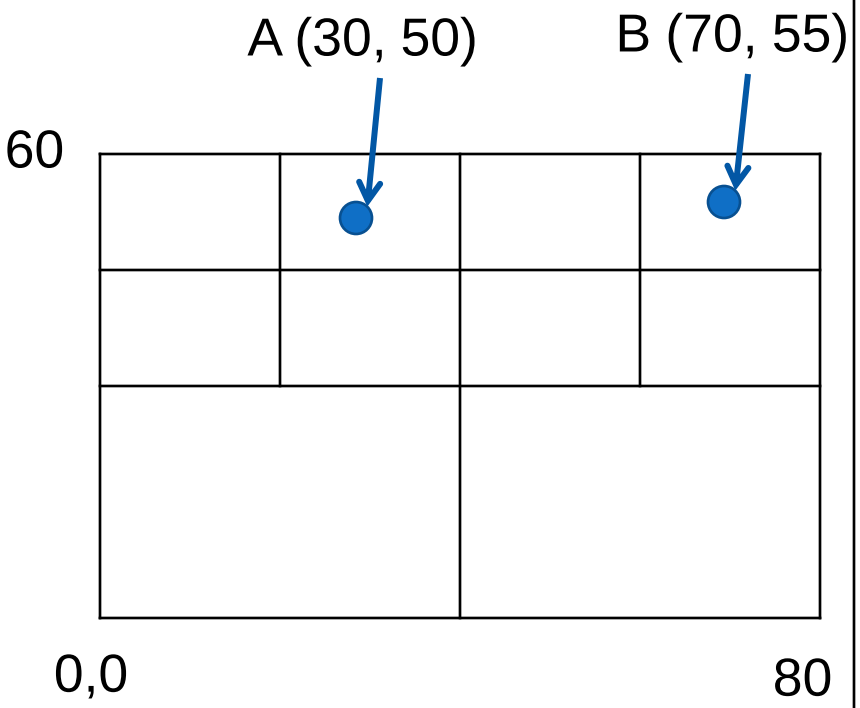




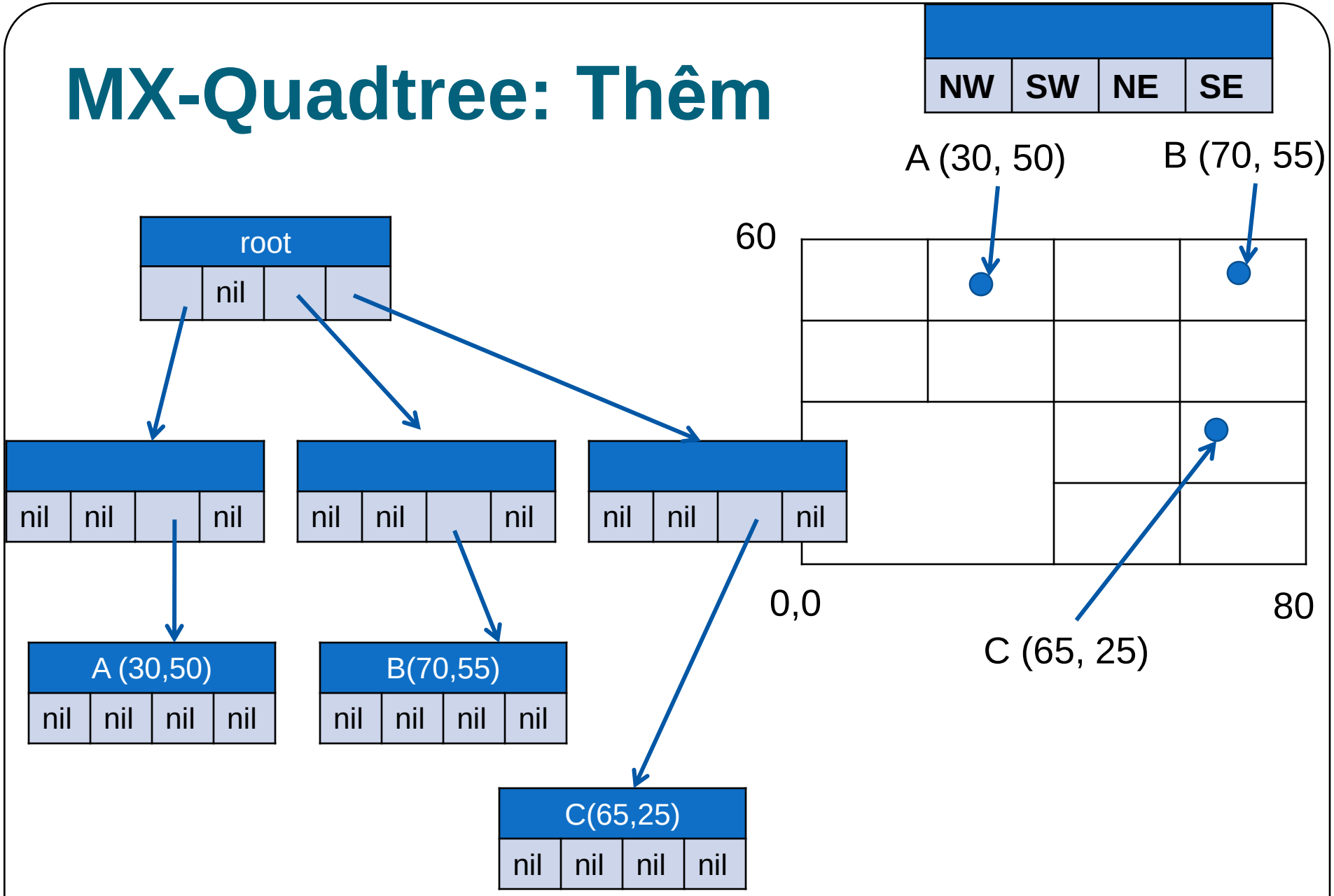
MX-Quadtree: Thêm



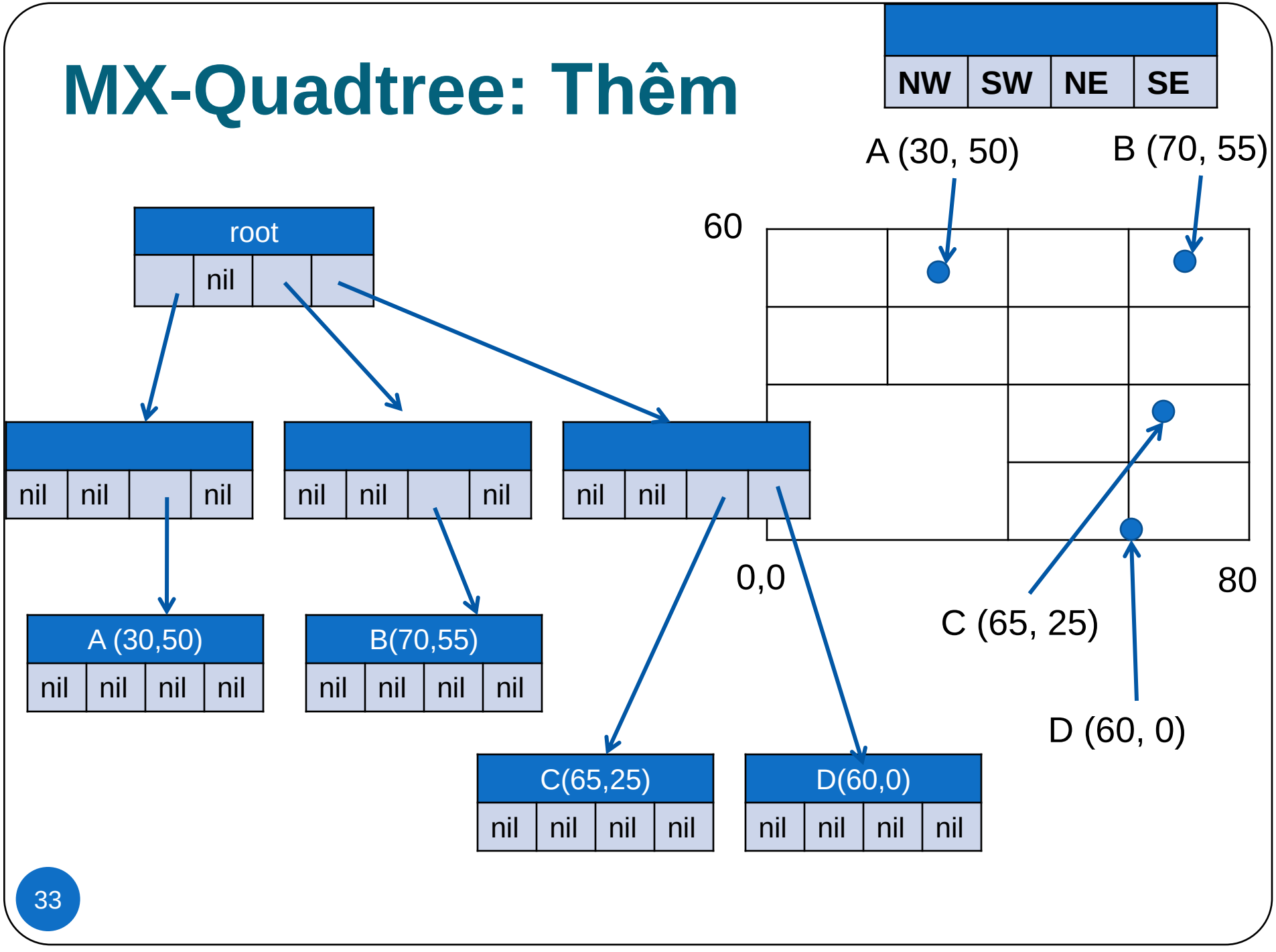




MX-Quadtree: Thêm



MX-Quadtree: Thêm



MX-Quadtree: Nhận xét

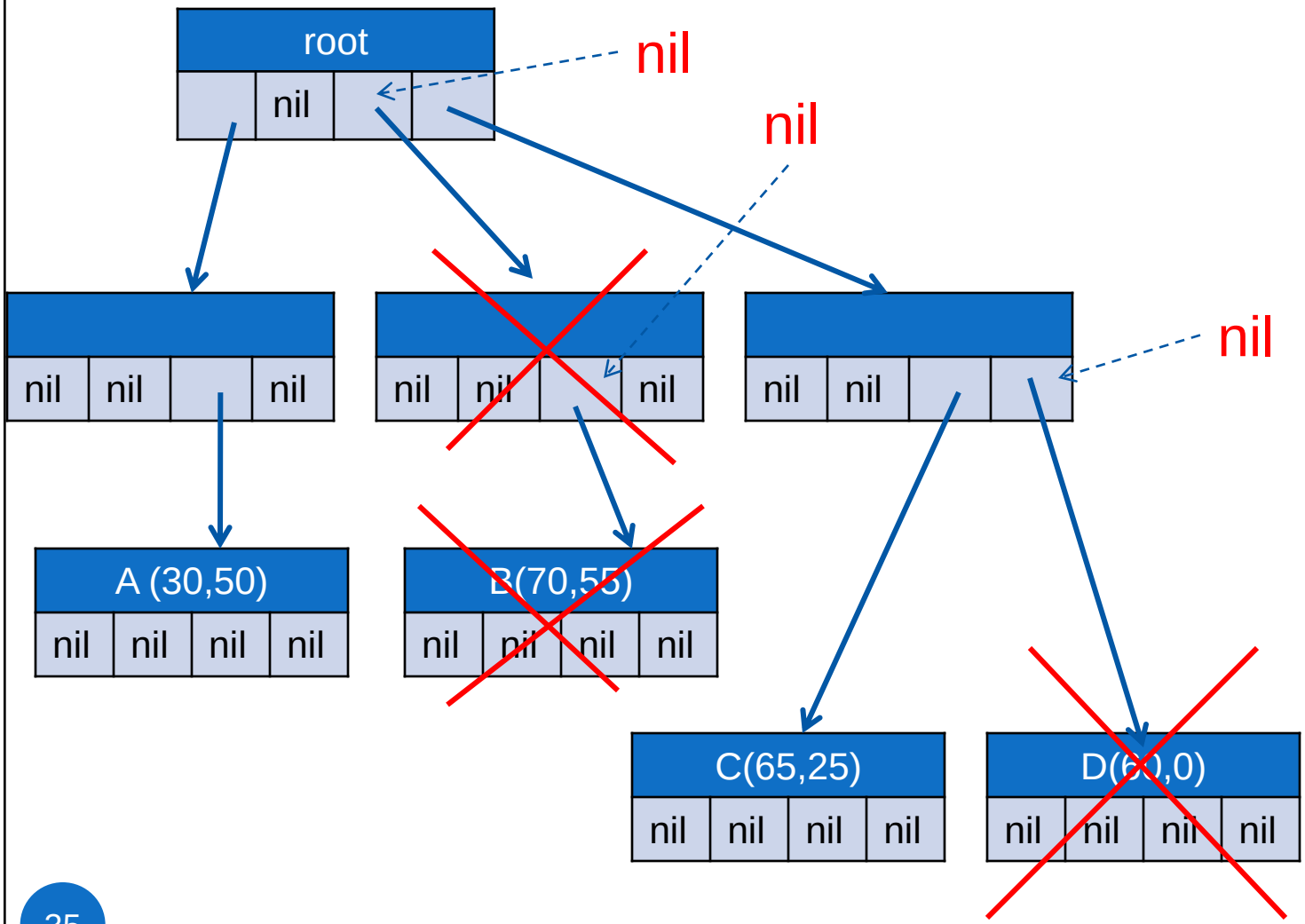
- Tất cả các điểm được biểu diễn bởi nút lá

- Nếu N là nút trong của cây MX-quadtree, thì vùng biểu diễn bởi N chứa ít nhất 1 điểm dữ liệu

➔Xóa dễ dàng

(Độ phức tạp: O(k))

MX-Quadtree: Xóa



MX-Quadtree: Truy vấn phạm vi

- Tương tự cây tứ phân dạng điểm
- Điểm khác biệt:
  - Kiểm tra 1 điểm có nằm trong phạm vi truy vấn hay không chỉ thực hiện ở **mức lá**

MX-Quadtree ➔ PR-Quadtree

- MX-Quadtree:
  - Dư nhiều nút nếu phân bố điểm thưa
  - Nếu nhiều hơn 1 điểm mà có trong vùng nhỏ nhất ➔?

➔PR-quadtree: biến thể của MX-quadtree

  - Nếu 1 nút chỉ có 1 điểm ➔ lưu vào nút đó mà không cần lưu vào nút lá
  - Chỉ phân chia vùng nếu vùng chứa nhiều hơn 1 điểm

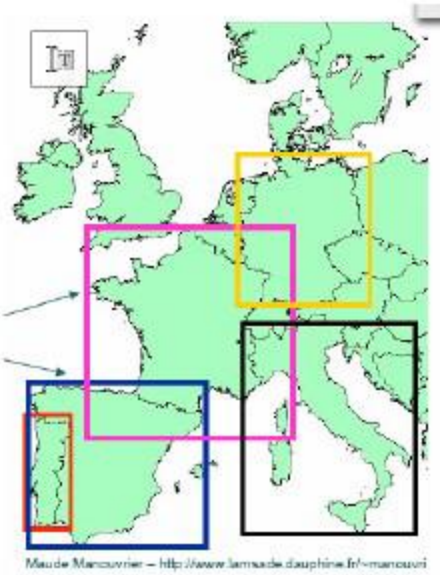
4. R-trees

R-trees

- K-D tree, Point-Quadtree, MX-Quadtree:

(Point Access Methods)

  – Truy cập đĩa nhiều ➔ chậm



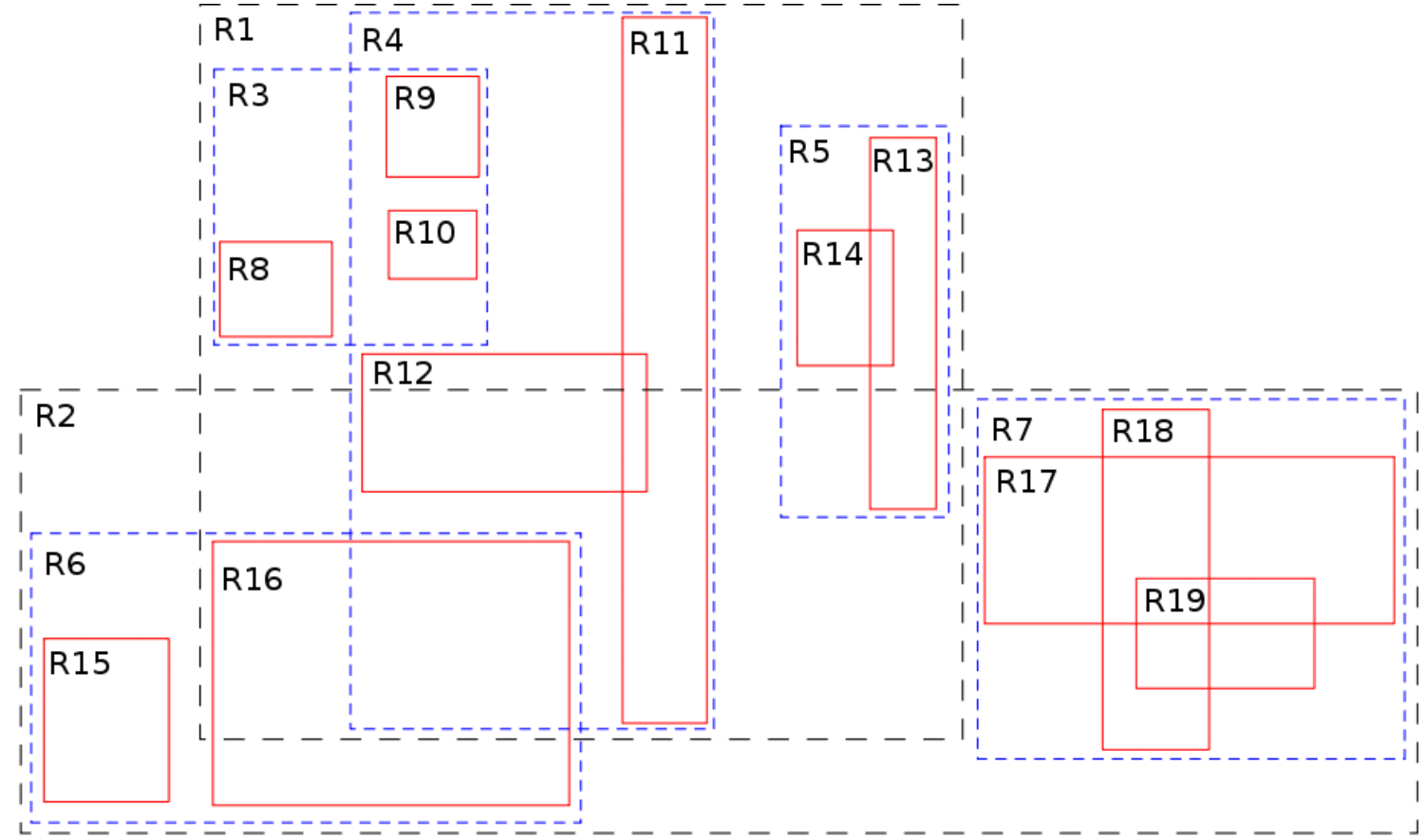
- R-trees (Rectangle-trees):

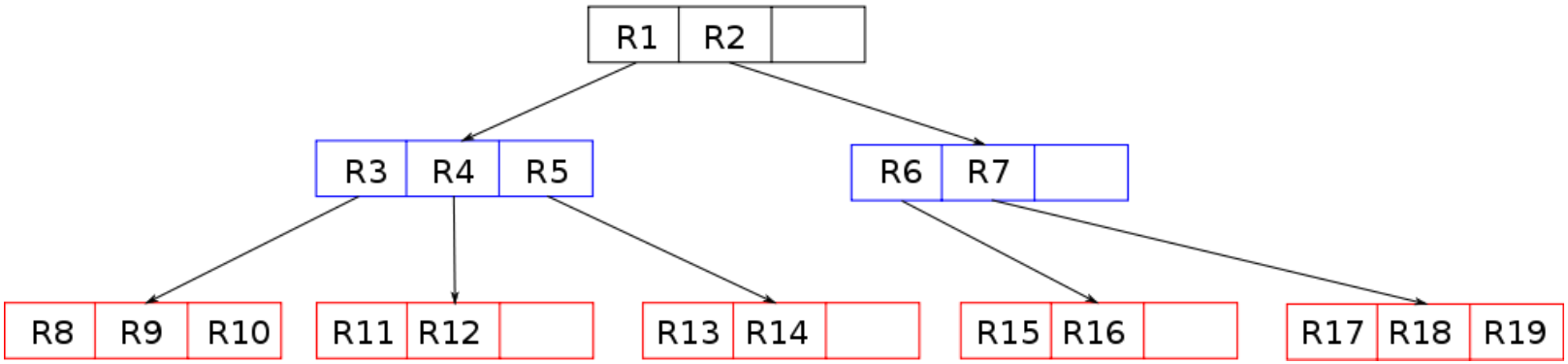
(Spatial Access Methods)

  – Hiệu quả cho lưu trữ DL lớn trên đĩa

  – Cách hiệu quả để tối thiểu số lần truy nhập đĩa

  (quản lý dữ liệu theo vùng)


R1
R4
R11
R3
R9
R5
R13
R10
R14
R8
R12
R2
R7
R18
R17
R6
R16
R19
R15

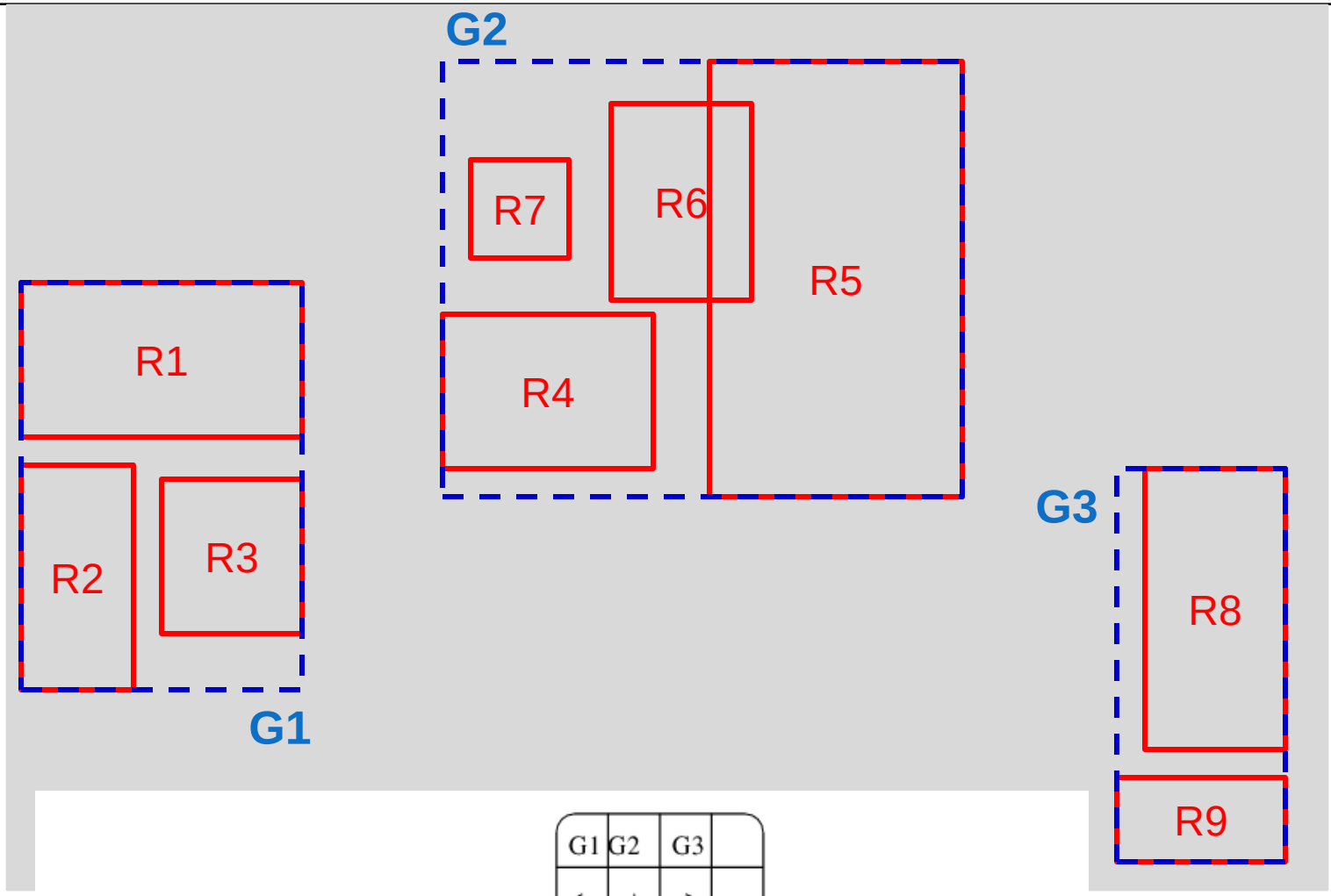

R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19

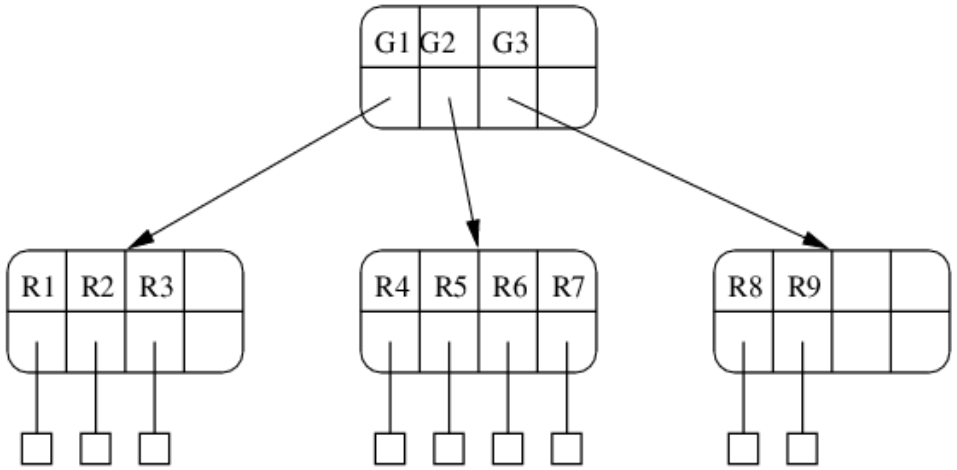
R-trees

- Phân chia không gian DL bằng các hình chữ nhật tối thiểu (MBR – Minimum bounding Rectangles)
- Các vùng có thể chồng nhau (overlapped)
- Dữ liệu lưu ở các nút lá, mỗi nút lá chứa nhiều dữ liệu

(tổ chức DL trong mỗi nút lá là tùy chọn)

- Mỗi R-tree có bậc ***K*** :
  - Mỗi nút trong của cây (có thể loại trừ nút gốc) chứa nhiều nhất K vùng và ít nhất $\lceil K/2 \rceil$ vùng


G2
R7
R6
R5
R4
R1
R2
R3
G1
G3
R8
R9


G1 G2 G3
R1 R2 R3
R4 R5 R6 R7
R8 R9
Leaf Nodes

R-tree bậc 4

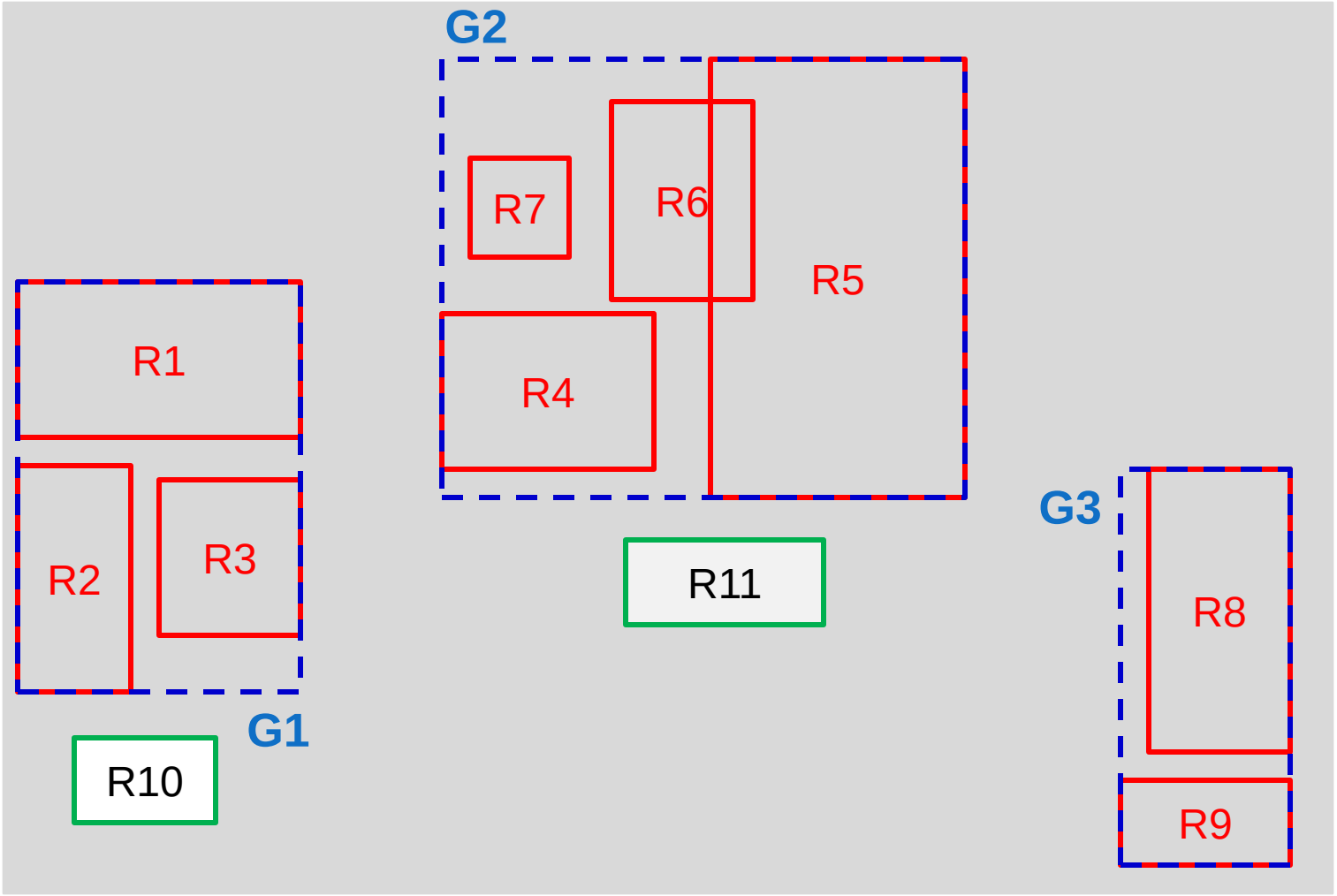
R-trees

- 2 loại vùng:
  - « Real » rectangles (vùng ở nút lá): R8, …, R19
  - « Group» rectangles (vùng ở nút trong): R1, .., R7

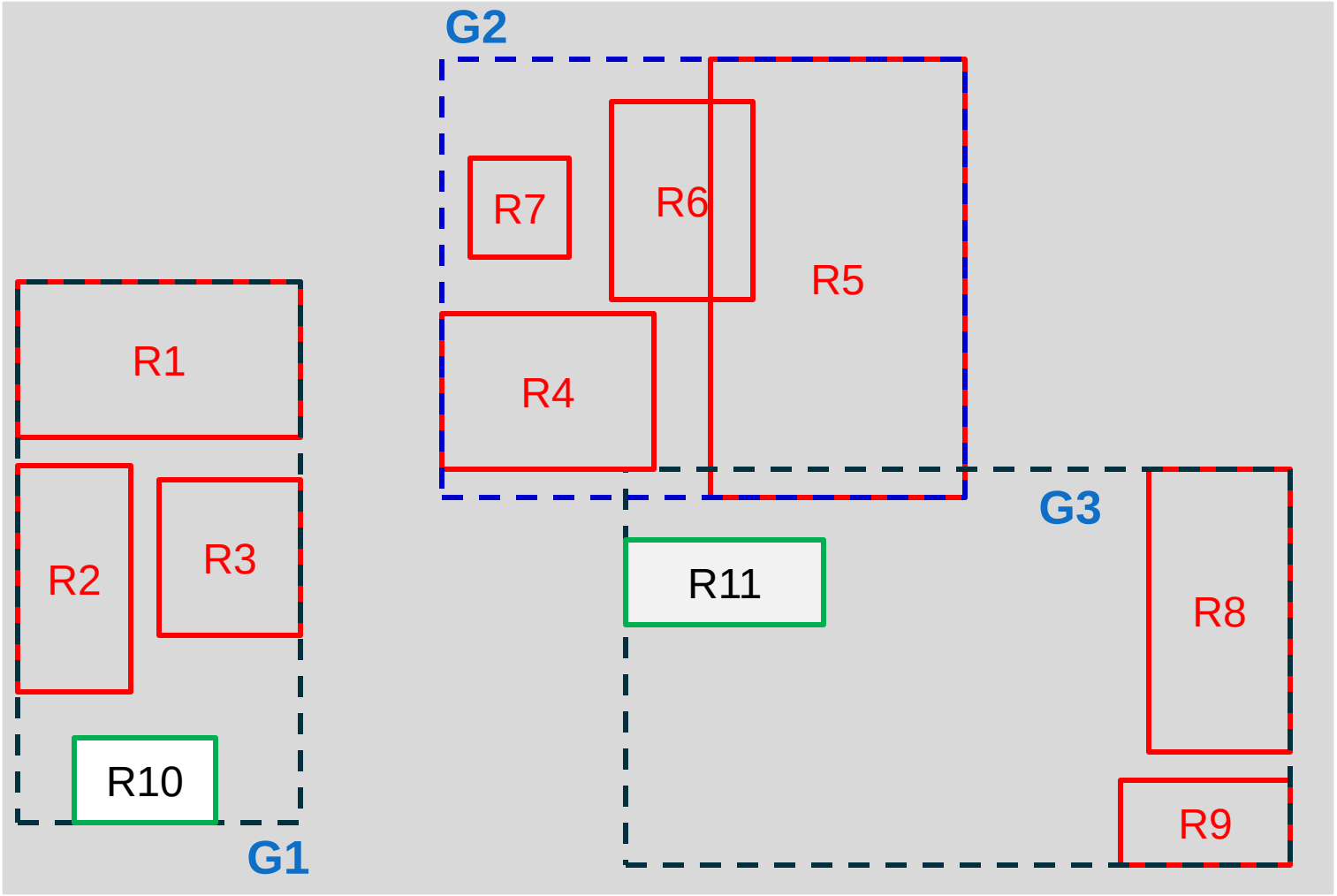
- Cấu trúc của nút trong R-tree bậc K:

Rec1	Rec2	…	RecK
Link1	Link2	…	LinkK

R-trees: Thêm

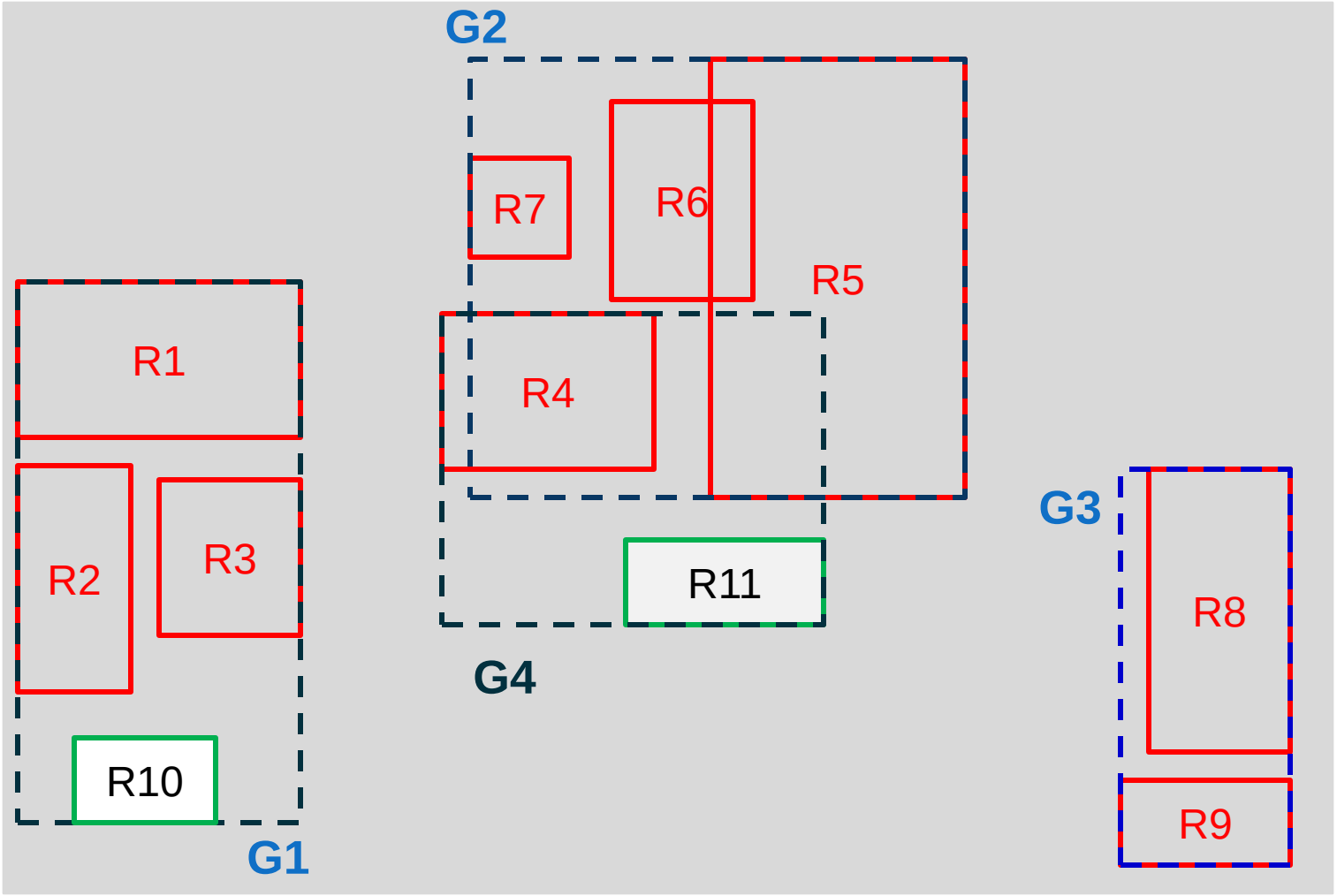


R-trees: Thêm (...)



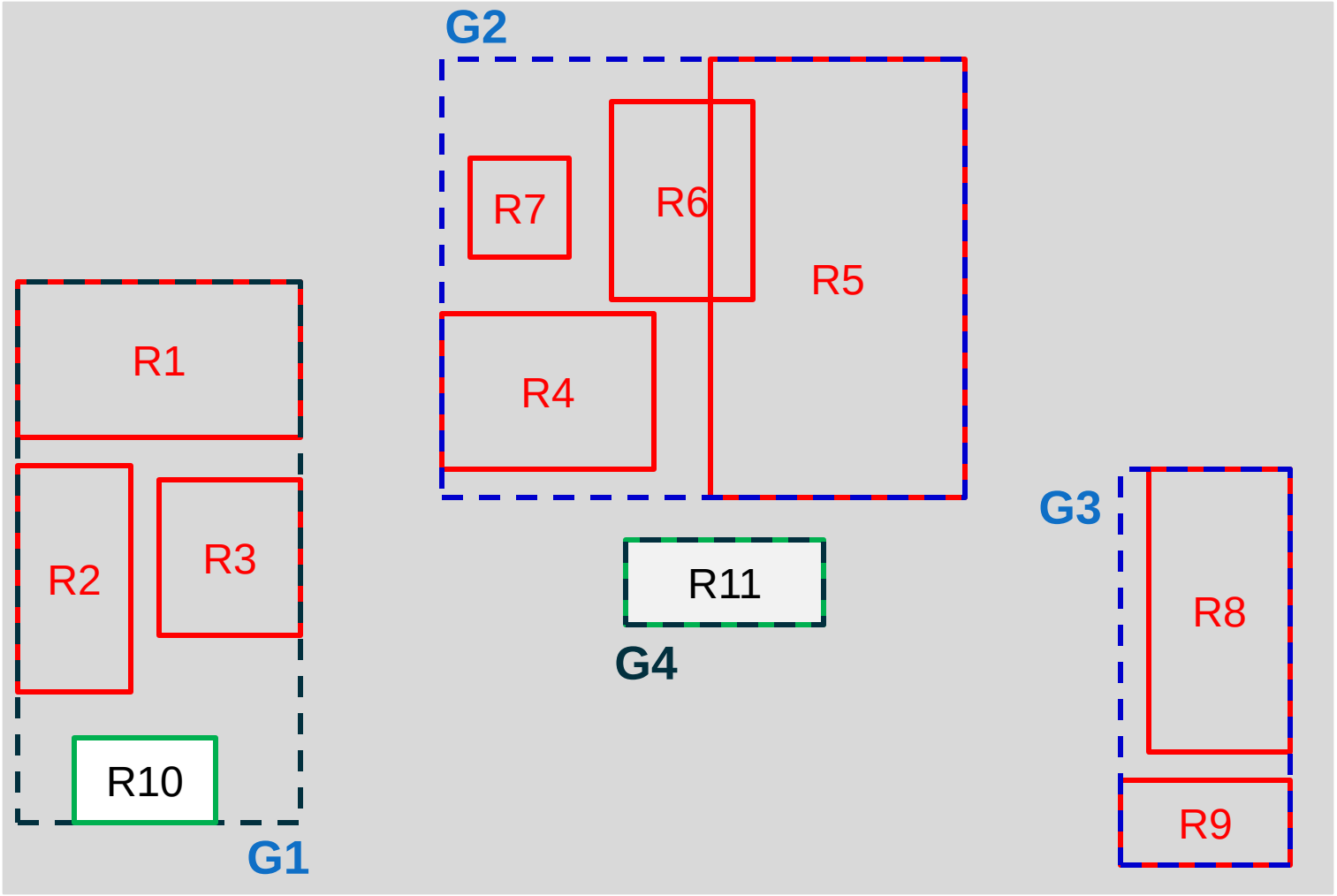
Solution 1

R-trees: Thêm (…)



Solution 2

R-trees: Thêm (…)



Incorrect insertion

R-trees: Thêm (…)

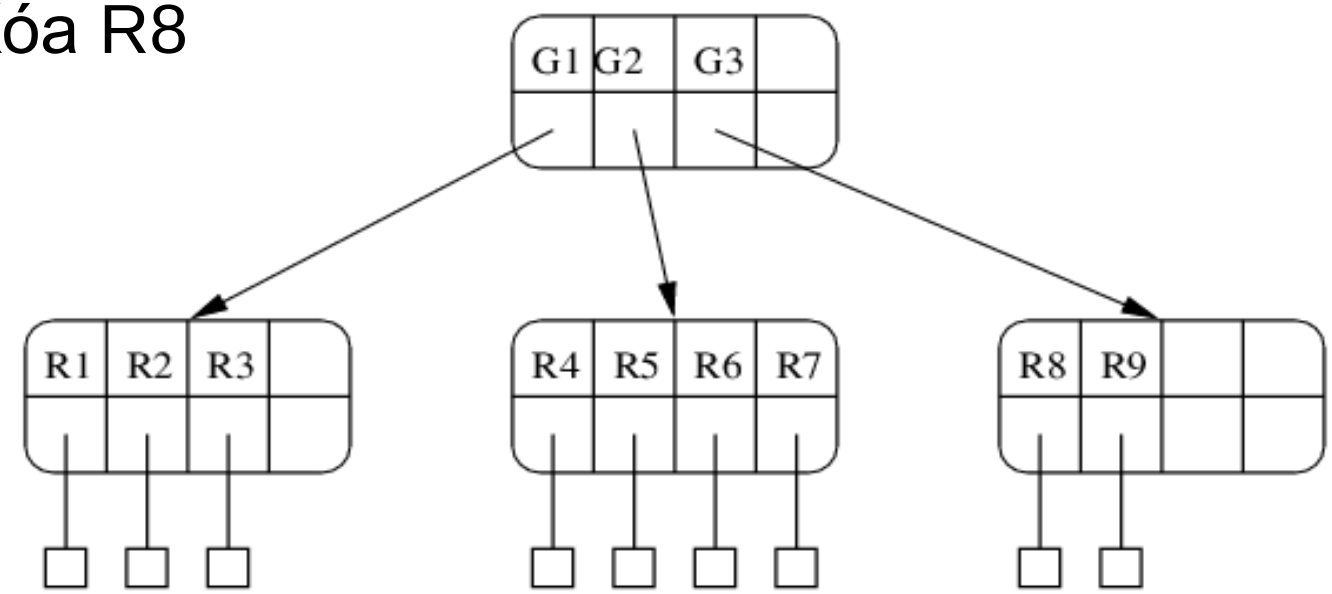
- Có thể phải tách để tạo thêm các nút mới
- Thêm 1 nút có thể phải thực hiện ở nhiều mức (level)
- Tiêu chí tách: Tối thiểu tổng không gian chiếm bởi các MBR
  - Tránh phải quản lý các vùng không có DL
  - Giảm sự chồng chéo của các MBR ➔ tăng hiệu quả khi tìm kiếm
- Có các thuật toán để xác định cách tách với độ phức tạp khác nhau

R-trees: Xóa

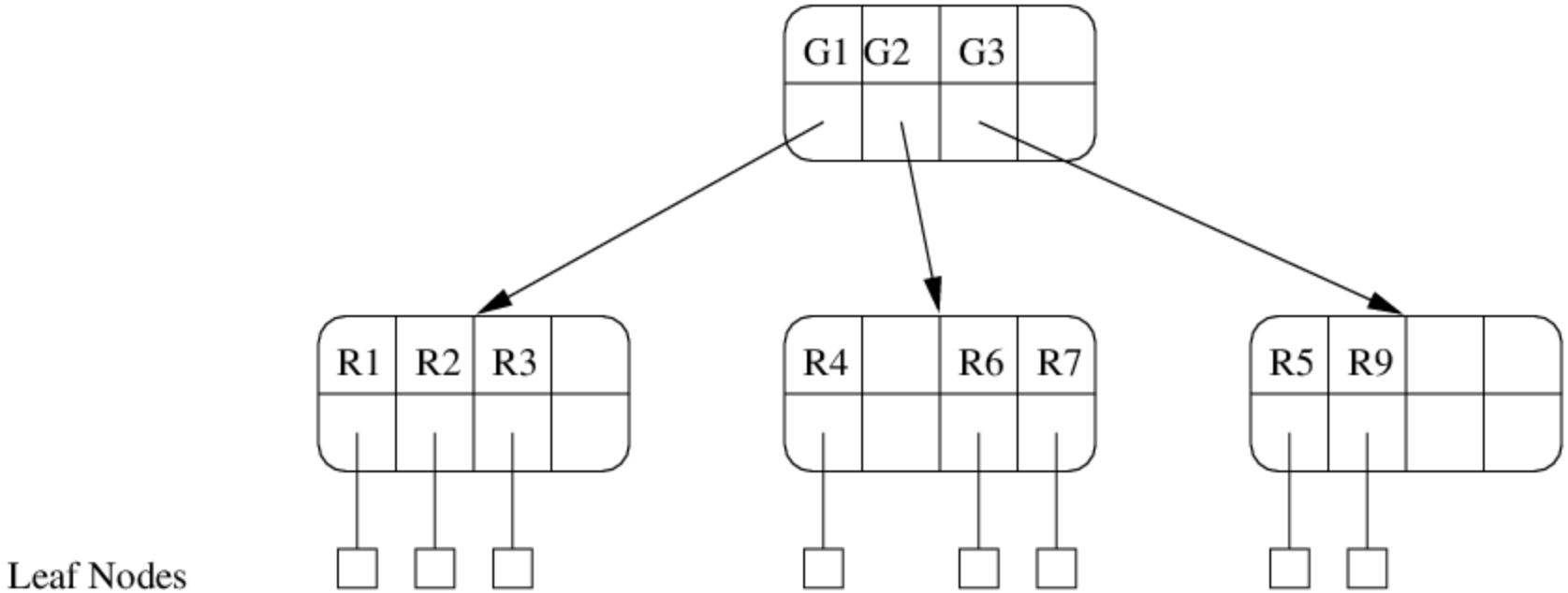
- Có thể gây ra 1 nút có số vùng (MBR) < K/2

➔**Sắp lại DL** nếu số vùng trong 1 nút < K/2

- **Ví dụ:** Xóa R8



R-trees: Xóa (…)

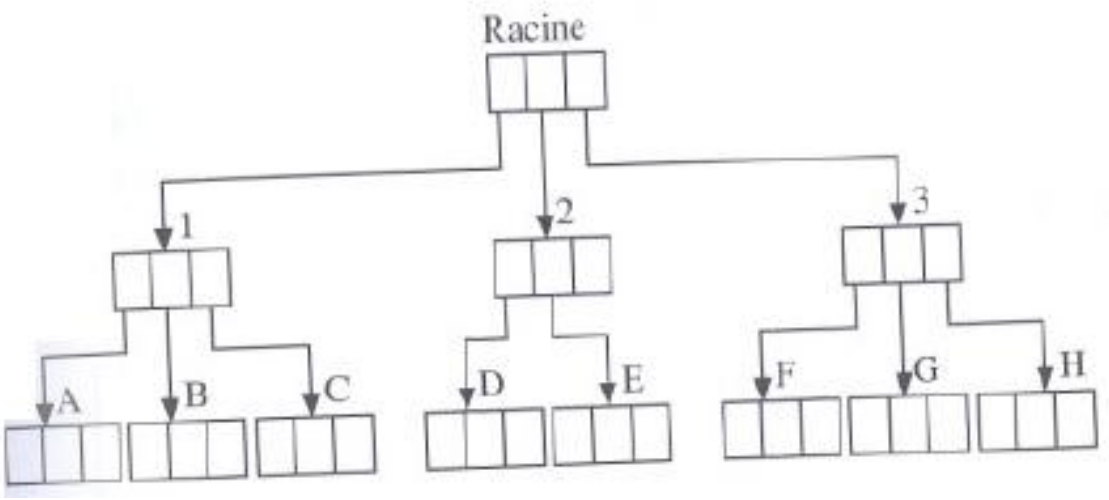


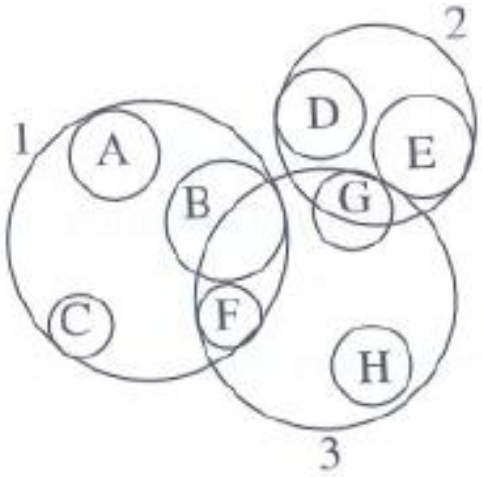
R-trees: Lưu ý

- Tìm kiếm: có thể phải thực hiện theo nhiều nhánh do có sự chồng chéo của các MBR
- Hình dáng cây phụ thuộc vào thứ tự insert DL
- Các biến thể: tối thiểu overlap giữa các MBR
  - R+-tree (1987): không cho phép MBR giao nhau
  - R*-tree (1990): tối thiểu các vùng chồng nhau và không gian chiếm bởi các MBR
    - ➔ SS-tree (similar-search tree), SR-tree (sphere-rectangle tree),TV-tree (telescopic vector tree)
  - X-tree (1996): tạo nút để quản lý tốt hơn sự giao nhau giữa các vùng

R-trees: Lưu ý

- SS-tree





- SR-tree: kết hợp của R*-tree và SS-tree

Tổng kết

- Point Quadtrees:
  - Dễ cài đặt
  - **Thêm/Tìm kiếm:** Nếu cây có n nút ➔ có thể cây cũng có độ cao là n ➔ xem hoặc tìm kiếm DL có độ phức tạp **O (n)**
  - **Xóa:** phức tạp do phải tìm nút thay thế cho nút xóa
  - Truy vấn phạm vi : $O(2\sqrt{n})$

Tổng kết

- k-D trees:
  - Dễ cài đặt
  - **Thêm/Tìm kiếm:** Nếu cây có n nút ➔ có thể cây cũng có độ cao là n ➔ xem hoặc tìm kiếm DL có độ phức tạp **O (n)**
  - Thực tế: độ sâu tìm kiếm thường dài hơn Point Quadtree
  - **Xóa:** cần tìm nút thay thế (không phức tạp)
  - Truy vấn phạm vi : $O(kn^{1-1/k})$

Tổng kết

- MX-quadtrees:
  - Đảm bảo độ cao cây <= n, lưới chia vùng: 2^n x 2^n
  - Insert/Delete/Search: O(n)
  - Truy vấn phạm vi hiệu quả: O(N +2^h)

    (h: độ cao của cây, N: số điểm thỏa mãn truy vấn)

Tổng kết

- R-trees:
  - Hiệu quả trong việc truy nhập đĩa (do nhiều vùng được lưu trên 1 nút ➔ cây ko quá cao), thích hợp với DL lớn

  ➔ Được sử dụng rộng rãi
  - Có thể không hiệu quả khi tìm kiếm nhất là truy vấn phạm vi do các MBR có thể giao nhau

- R-tree: thích hợp cho DL lớn
- DL có kích thước indexes nhỏ: MX-quadtree hợp lý

Không gian với số chiều lớn

Curse of dimensionality

- Hiệu quả các kỹ thuật đánh chỉ mục giảm nhanh khi số chiều dữ liệu tăng: *curse of dimensionality*
  - Số các vùng được phân hoạch tăng theo hàm mũ: k^d
    - (d: số chiều, k: số phân hoạch trên 1 chiều)

  ➔ Nhiều vùng rỗng hoặc chỉ có 1 phần tử ➔ ko hiệu quả
  - Hiệu ứng biên: nếu câu truy vấn nằm gần biên của 1 vùng ➔ nhiều điểm « hàng xóm » ➔ ko hiệu quả trong tìm kiếm
  - Không gian của hyper-rectangle, hyper-sphere tăng theo hàm mũ theo d ➔ tìm kiếm toàn bộ KGian
  - Kết quả tìm kiếm KNN không ổn định

Giảm chiều dữ liệu

- Karhunen–Loeve transform (KLT) và các biến thể:
  - Xét đến mối tương quan giữa các thành phần trong không gian đặc trưng
  - VD: Eigenimage, PCA (principal component analysis), SVD (singular value decomposition)

vision.ece.ucsb.edu/publications/97GMIPSVD.pdf

*cs-people.bu.edu/evimaria/cs565/**fast**map.pdf*

- Phân cụm các chiều trong không gian đặc trưng (column-wise clustering)
  - Các chiều có giá trị xấp xỉ nhau được nhóm thành 1 cụm