R-Trees

V.S. Subrahmanian

Fall 2011

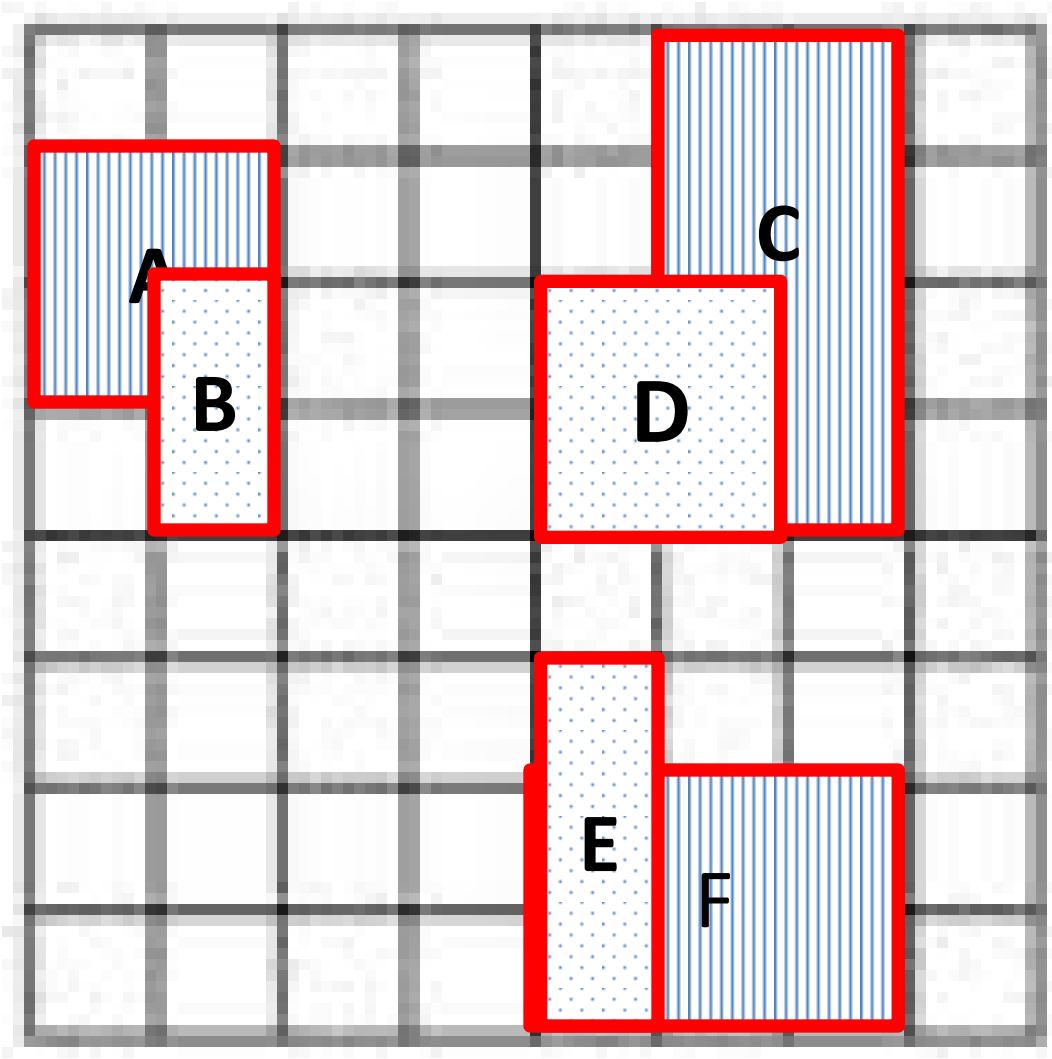
R-trees

- R-trees are a generalization of B-trees to store rectangular data.
- As before, we assume rectangles are characterized by 4 numbers (LEFT,RIGHT,BOTTOM,TOP).
- An R-tree has an *order m* which is the number of rectangles that can be stored in a node.
- Each R-tree node also has up to *m* children.

R-trees

- Every node in an R-tree represents a region which is the minimal bounding rectangle (MBR) containing all the rectangles labeling that node.
- All “data rectangles” are stored at leaf nodes.
- Non-leaf nodes are supposed to be at least ½ full except for the root.
- Rectangles labeling non-leaf nodes are “synthetic” rectangles.

Example



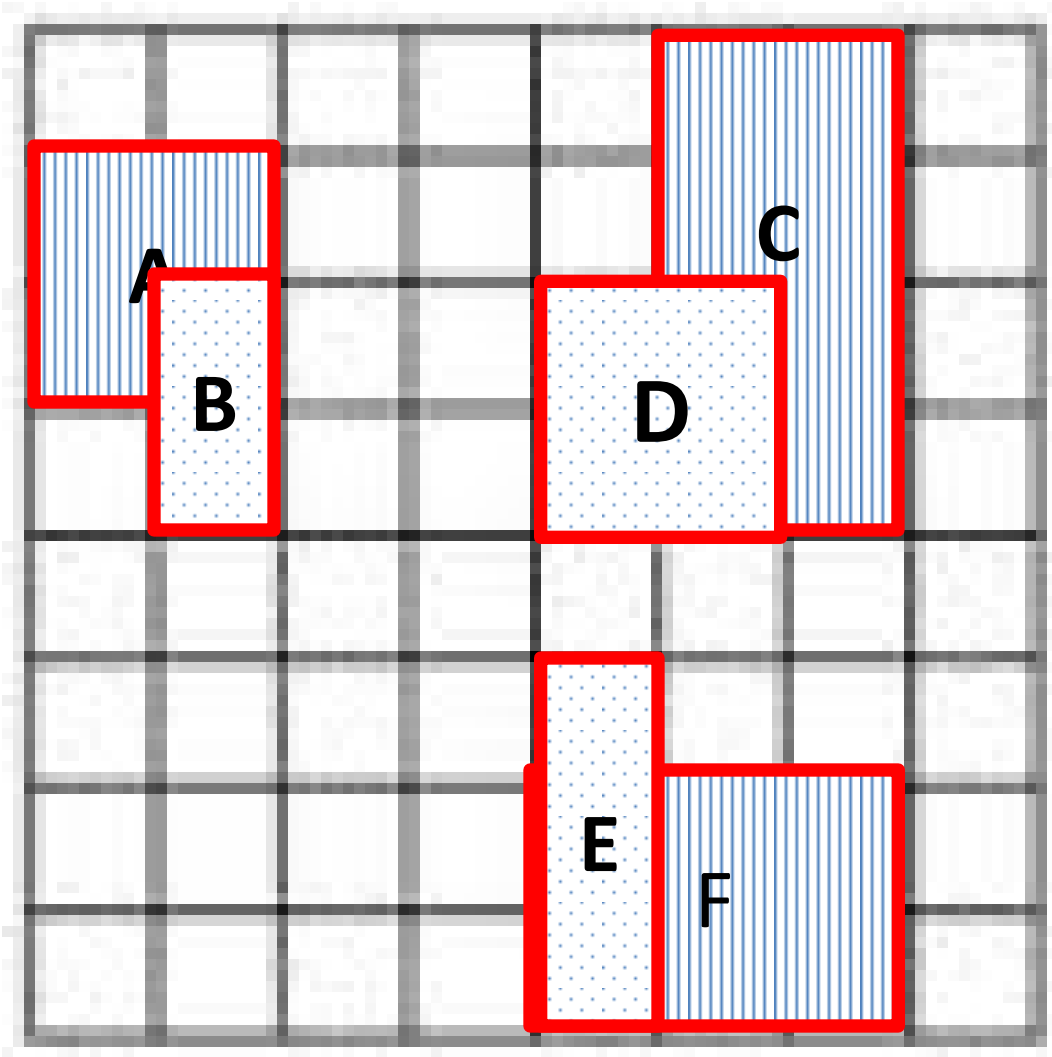
Insert A

Consider an R-tree of order 2. Each node has up to 2 rectangles and each node has up to 2 children. Create a root with A in it.

Insert B

Insert B. There is space in the root, so just put B there.

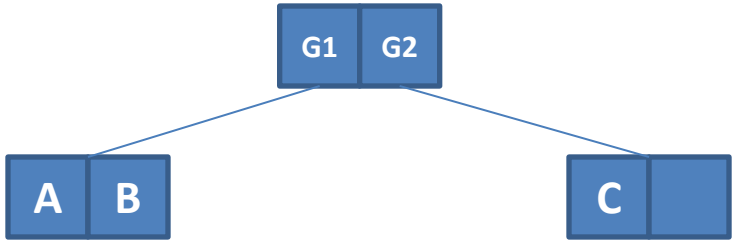
Example



Insert C

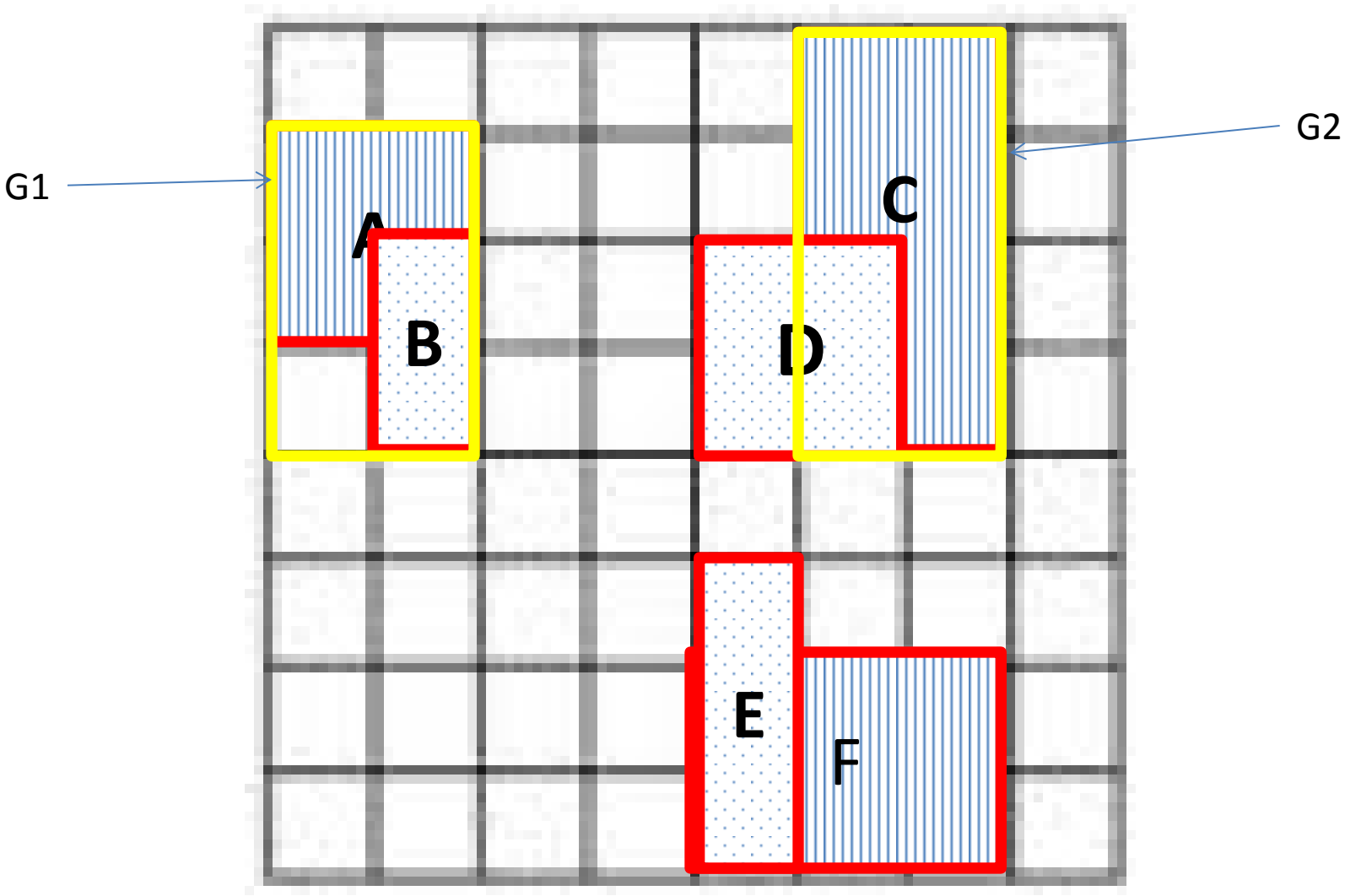
The root is now overflowing. So need to group A,B,C into two buckets. Each bucket must have at most 2 rectangles. Sum of the areas of the MBRs of the two buckets should be minimized. How should we split A,B,C into two buckets?

Insert C

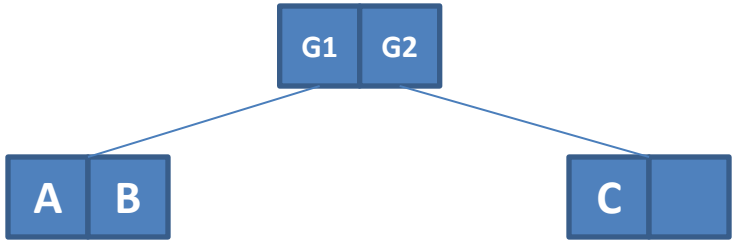


Clearly, the MBR of A,B is pretty small. So it is best to have one bucket with A,B in it, and the other with just C in it. G1 and G2 are synthetic rectangles the MBRs of the rectangles in their children. So G2 is identical to C.

Example

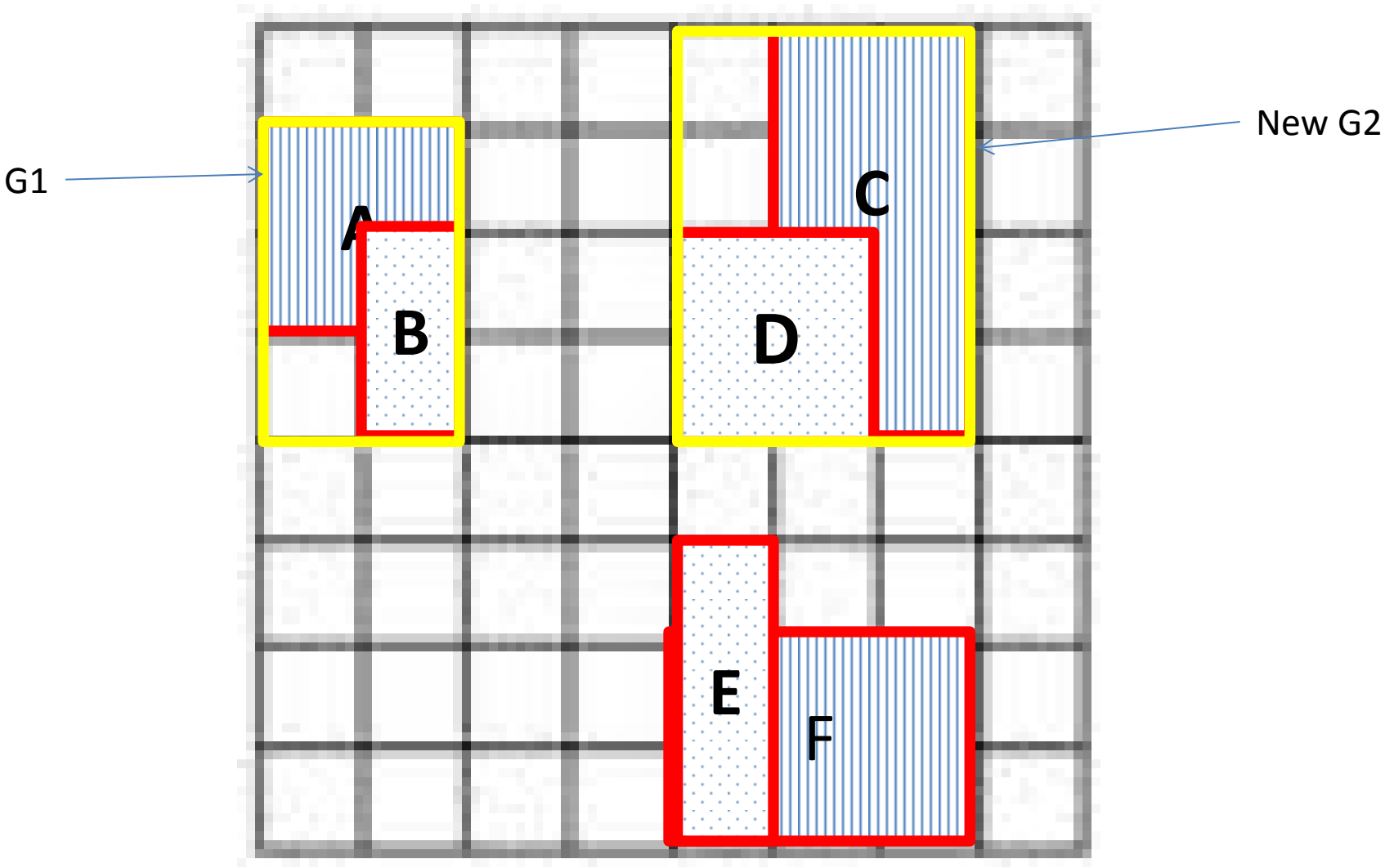


Insert D

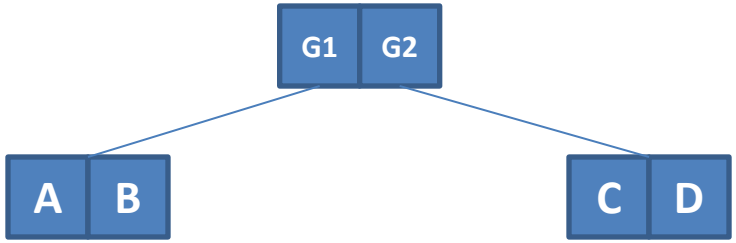


We want to insert D now. D causes us to have to make a choice – go left or right? We go in whichever direction causes the smallest increase in the area of the synthetic rectangle. In this case, G2 would have to be expanded “less” for D to fit into it.

Example

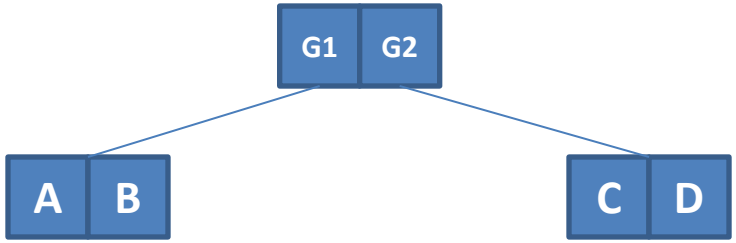


Insert D



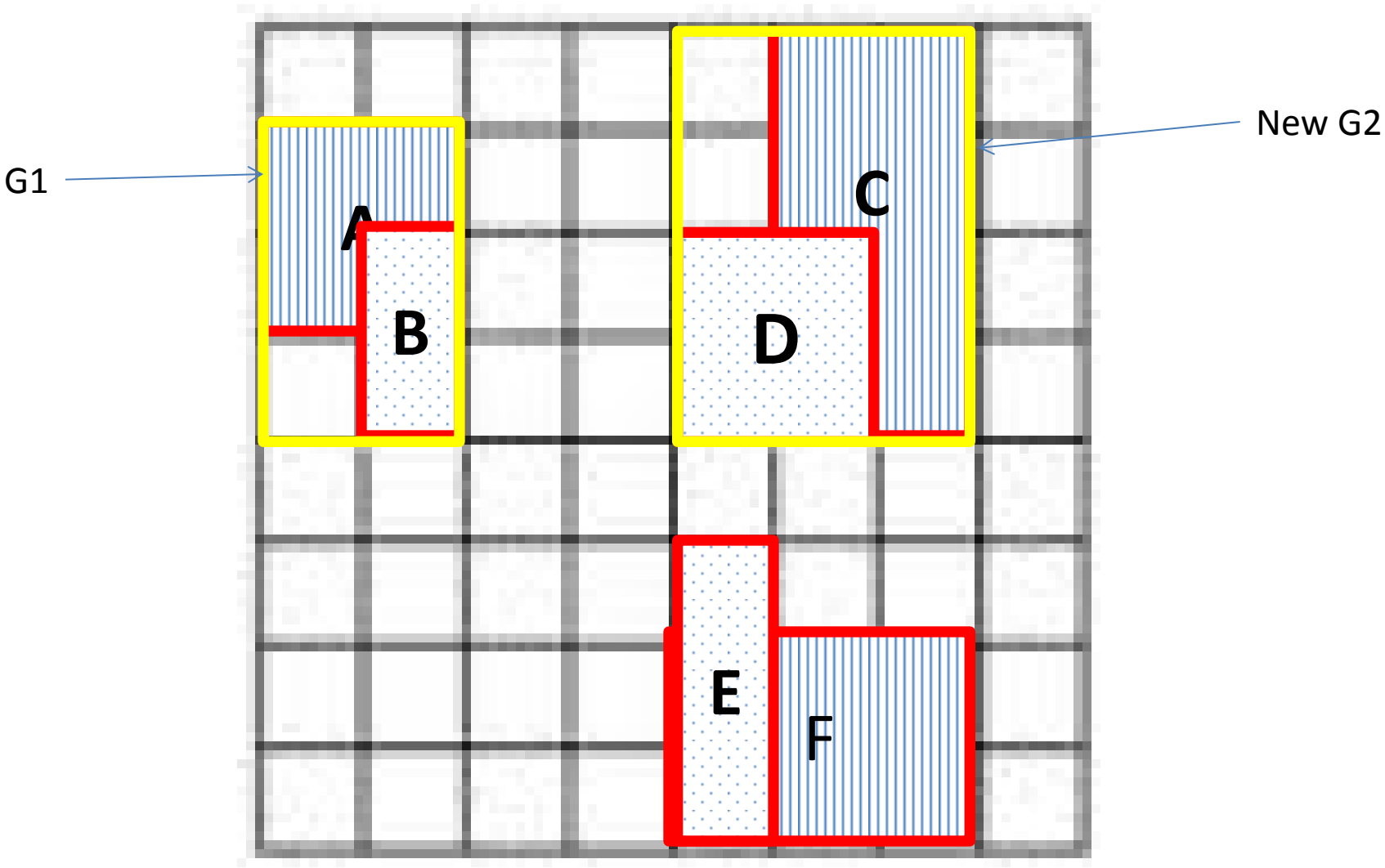
So D ends up in the same node as C.

Insert E

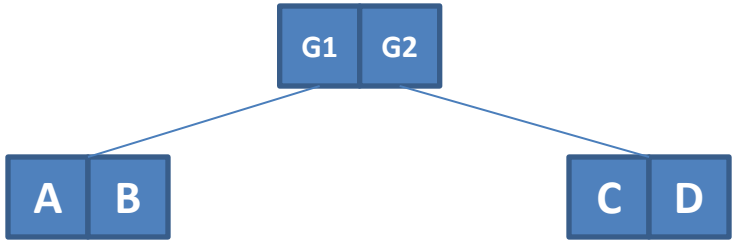


Now insert E. Should we branch left or right from the root? We go in whichever direction causes the smallest increase in area of the synthetic rectangle involved.

Example

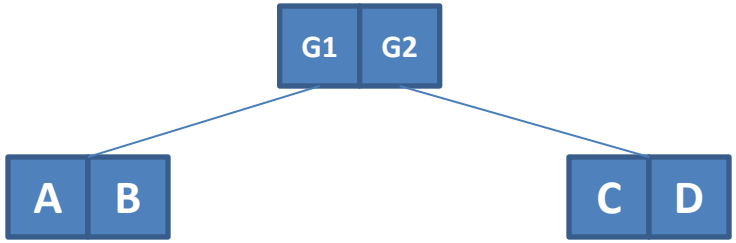


Insert E



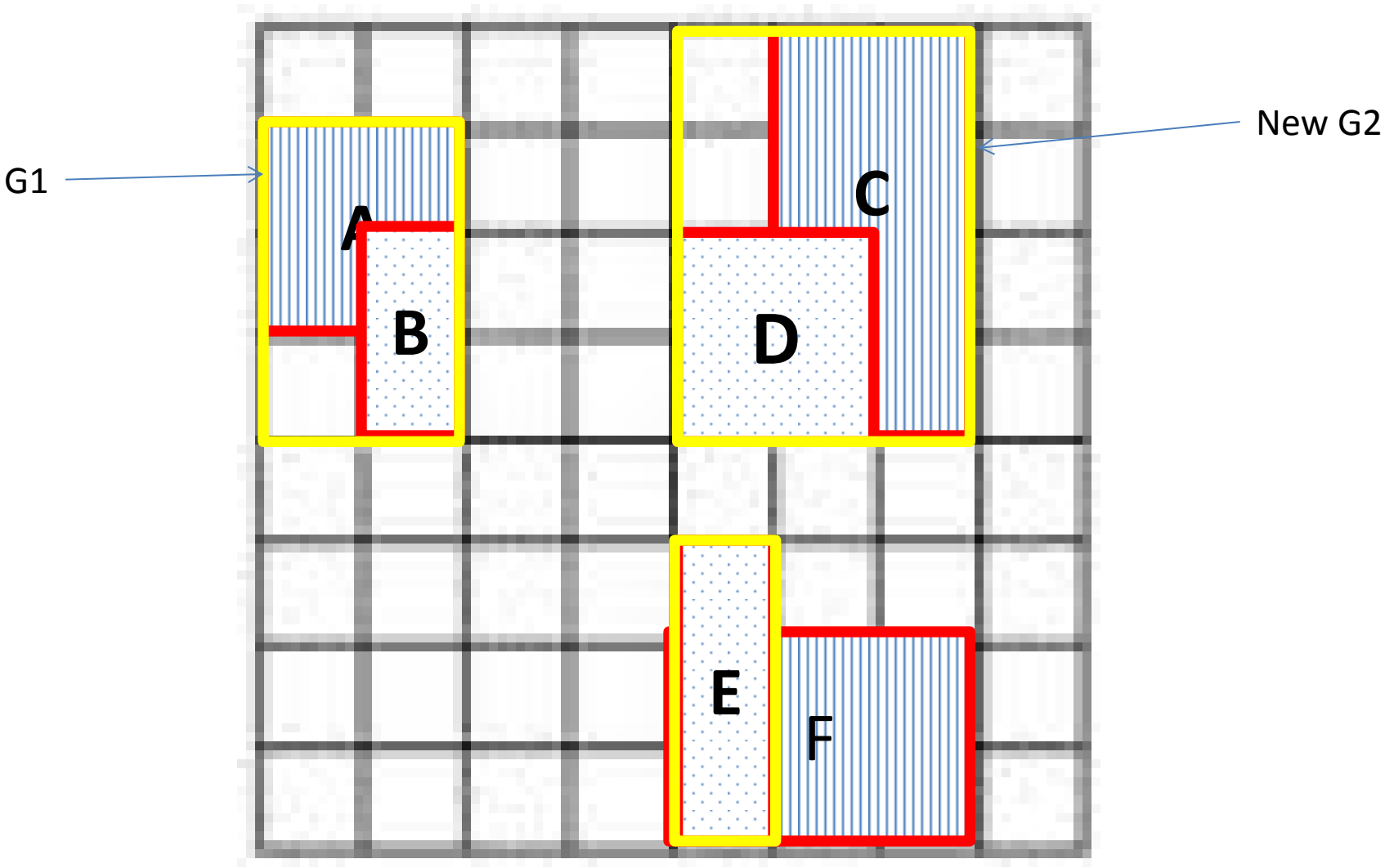
So we now branch right (G2) because the increase in the area of an expanded G2 would be smaller than the increase in area of an expanded G1 that contains E. But when we come to the right child, we find that it is already full – so must split it.

Insert E

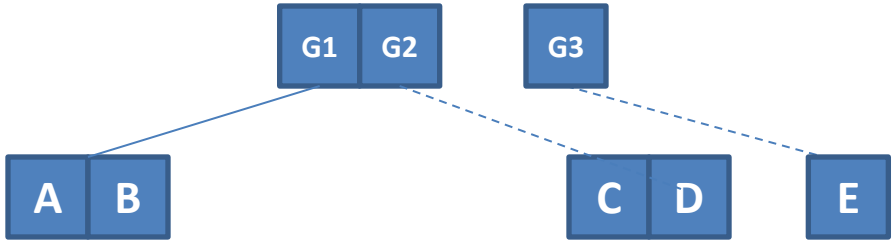


The right child contains C,D,E – so if we split it in two, we want two synthetic rectangles with minimal area – one is the MBR of C,D and the other is the MBR of just E.

Example

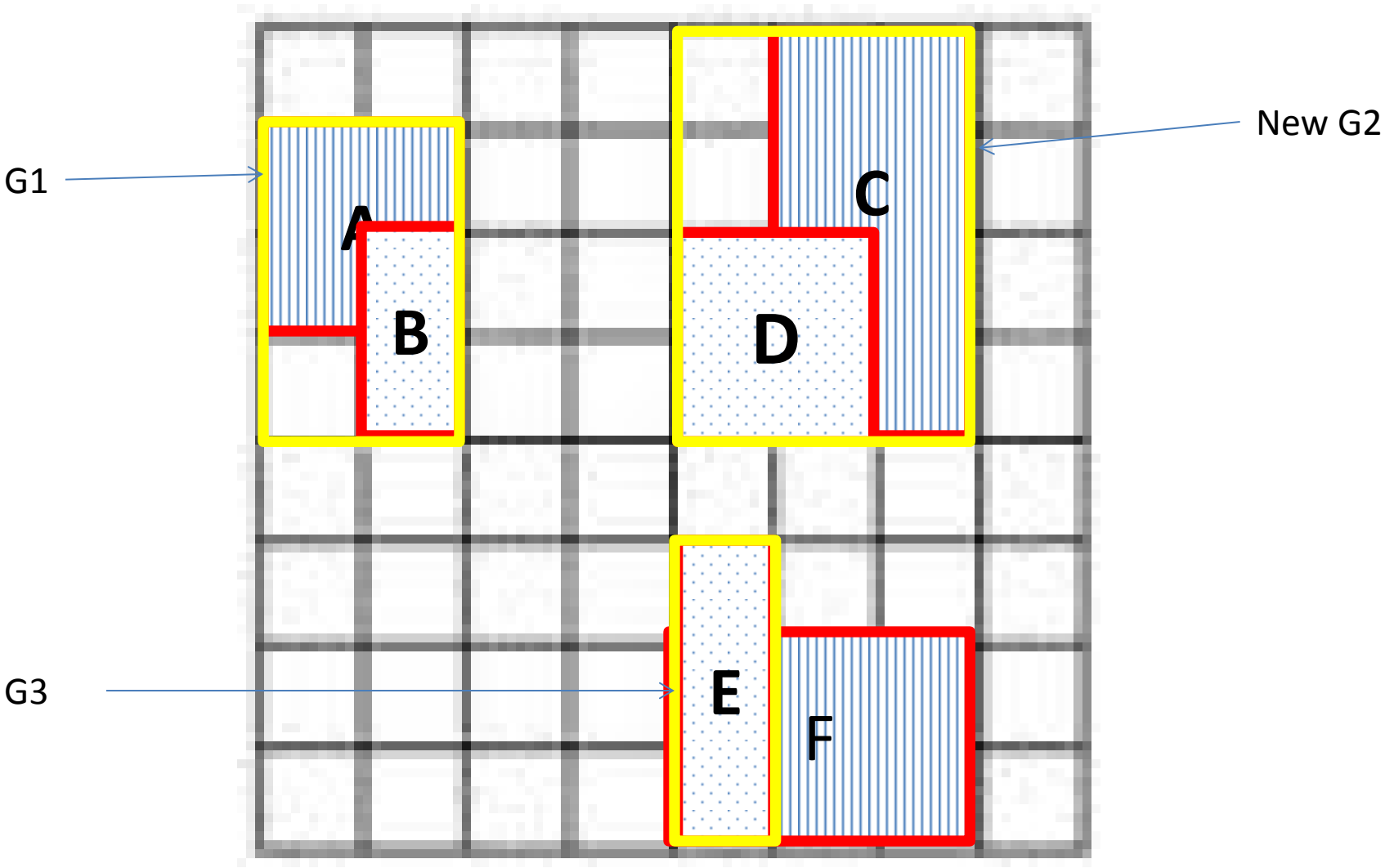


Insert E

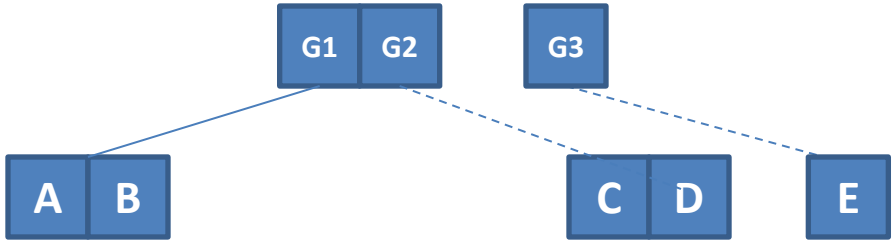


The situation is as shown above. But this causes a synthetic rectangle G3 (identical to the MBR of E which equals E) to be promoted up. This causes the root node to split.

Example

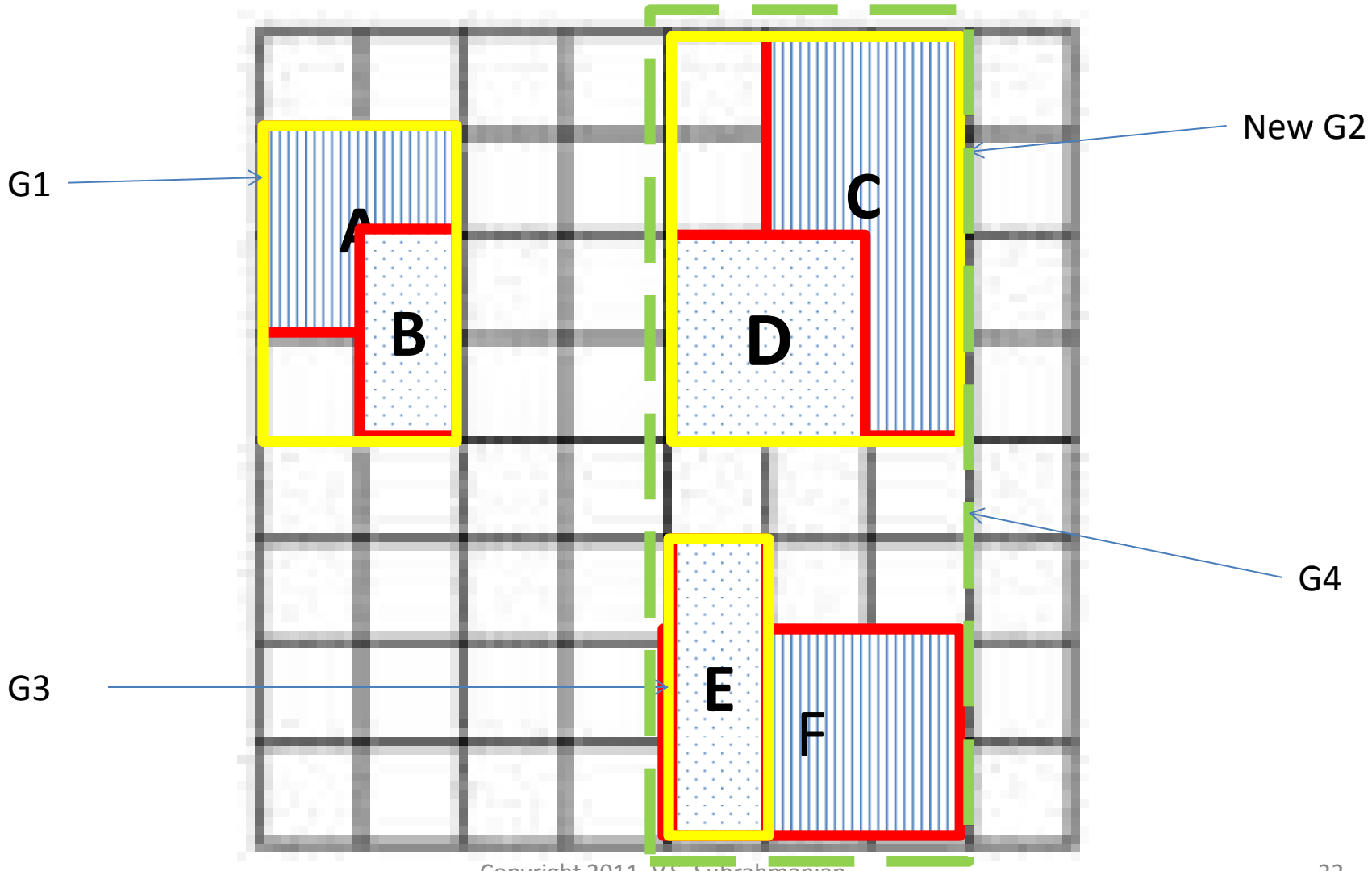


Insert E

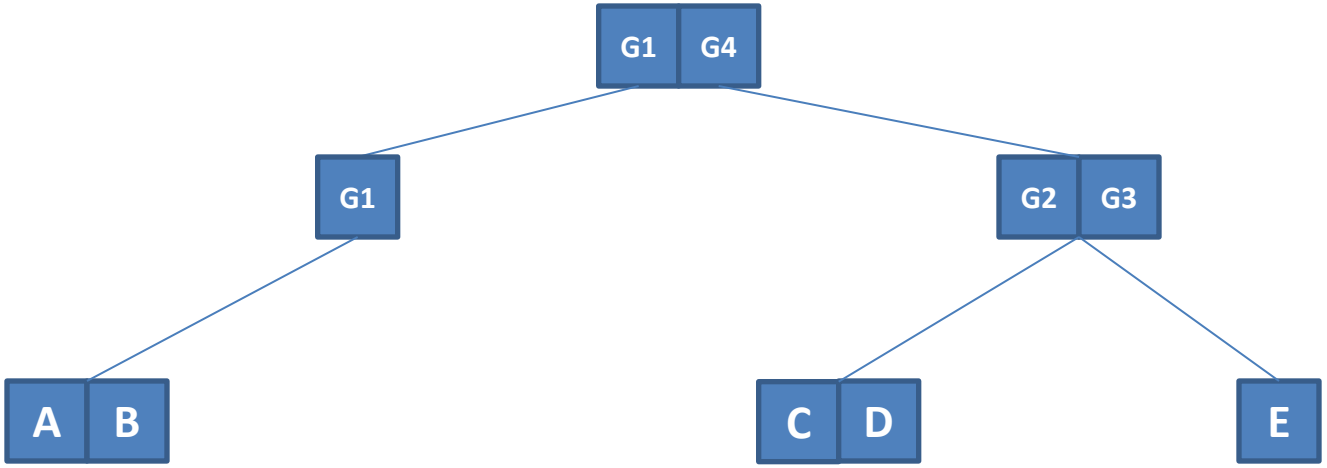


We must decide how to merge G1,G2,G3 into two. Let us see how this works. Area(MBR(G1,G2))+Area(G3)= 28+3=31. Area(G1)+MBR(G2,G3)=6+24=30; Area(MBR(G1,G3))+Area(G2)= 35+12 = 47. So minimal area is obtained by merging G2,G3.

Example

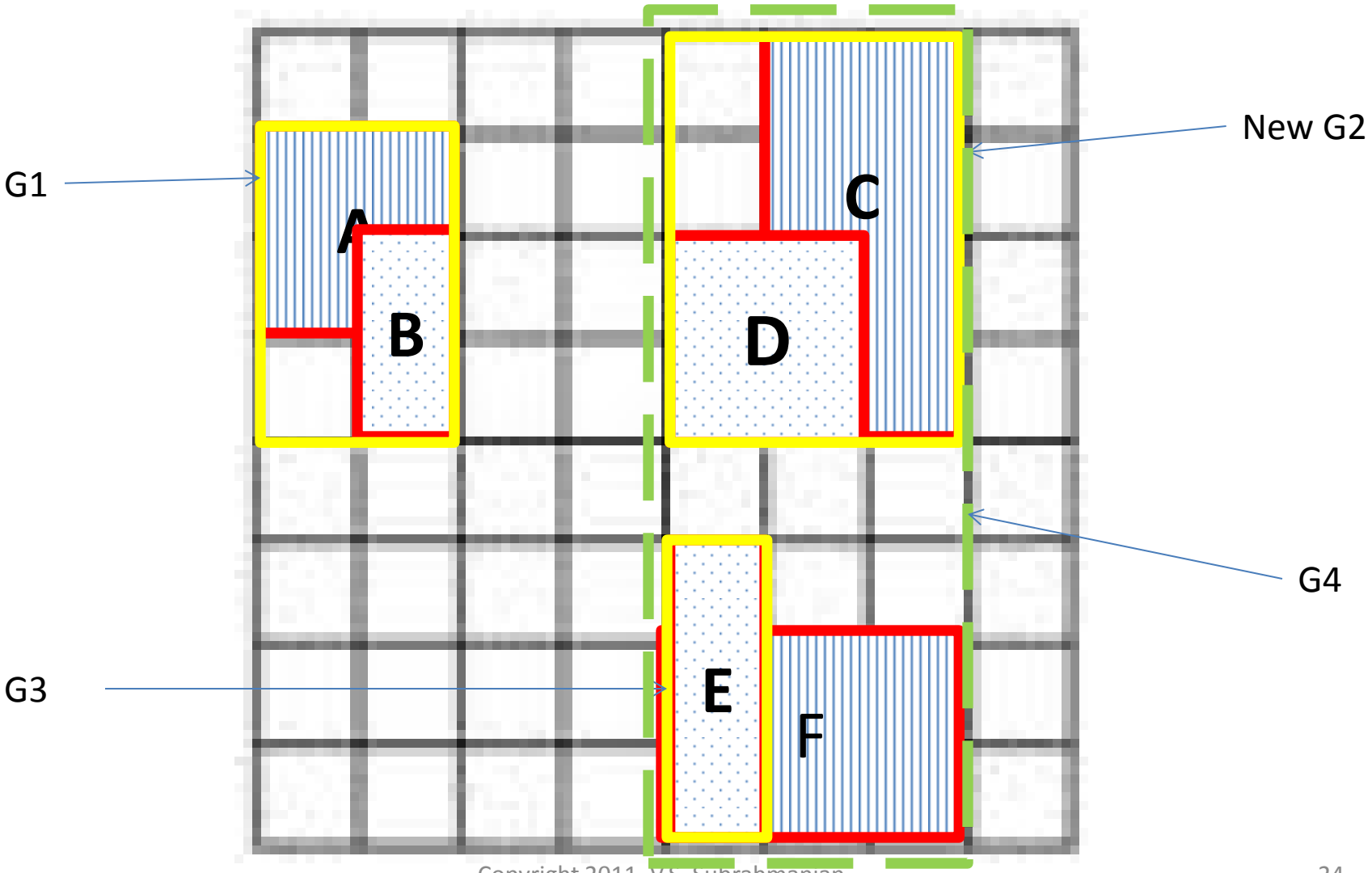


Insert E

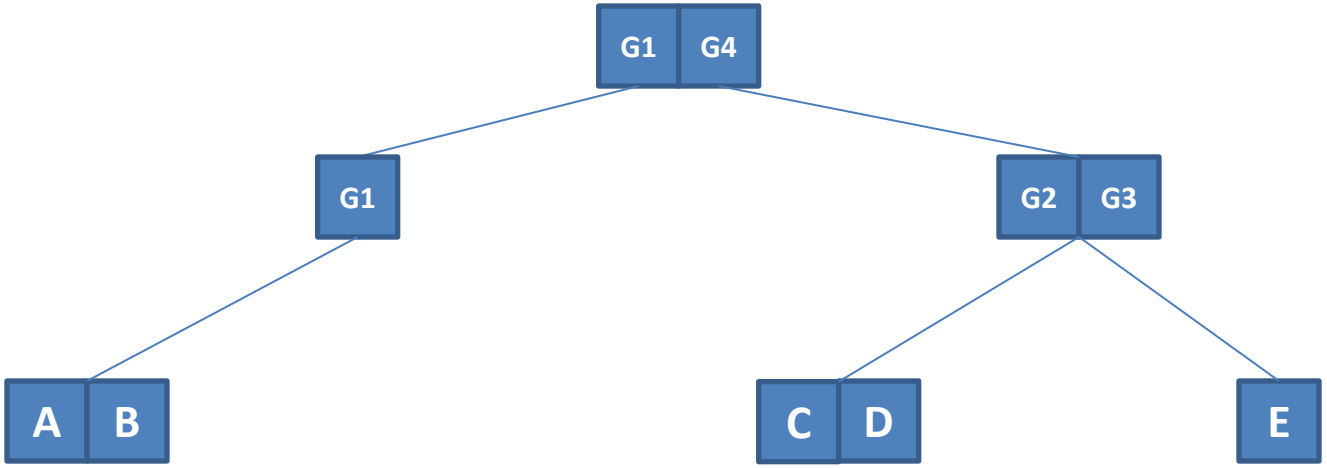


We must decide how to merge G1,G2,G3 into two. Let us see how this works. Area(MBR(G1,G2))+Area(G3)= 28+3=31. Area(G1)+MBR(G2,G3)=6+24=30; Area(MBR(G1,G3))+Area(G2)= 35+12 = 47. So minimal area is obtained by merging G2,G3.

Example: Insert F

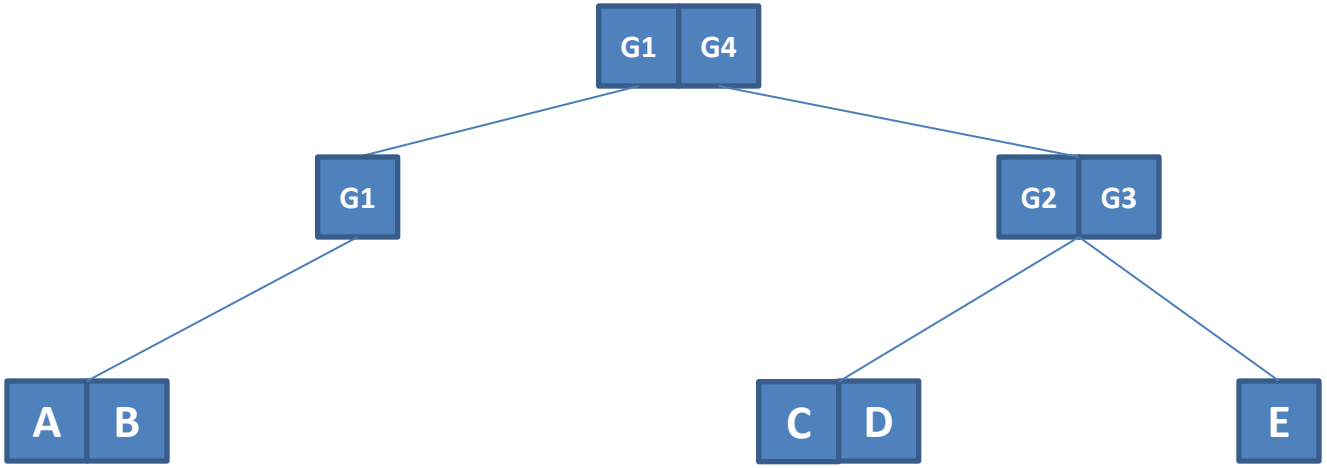


Insert F



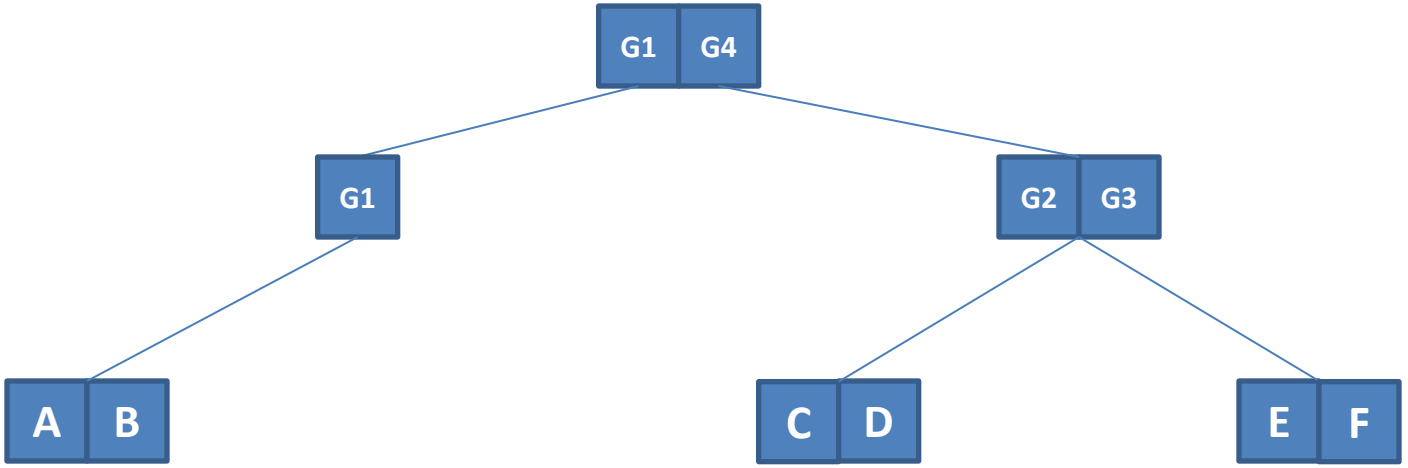
Look at root. Which area would have to be expanded less in order to accommodate F? G1 or G4? Clearly G4. So branch right.

Insert F



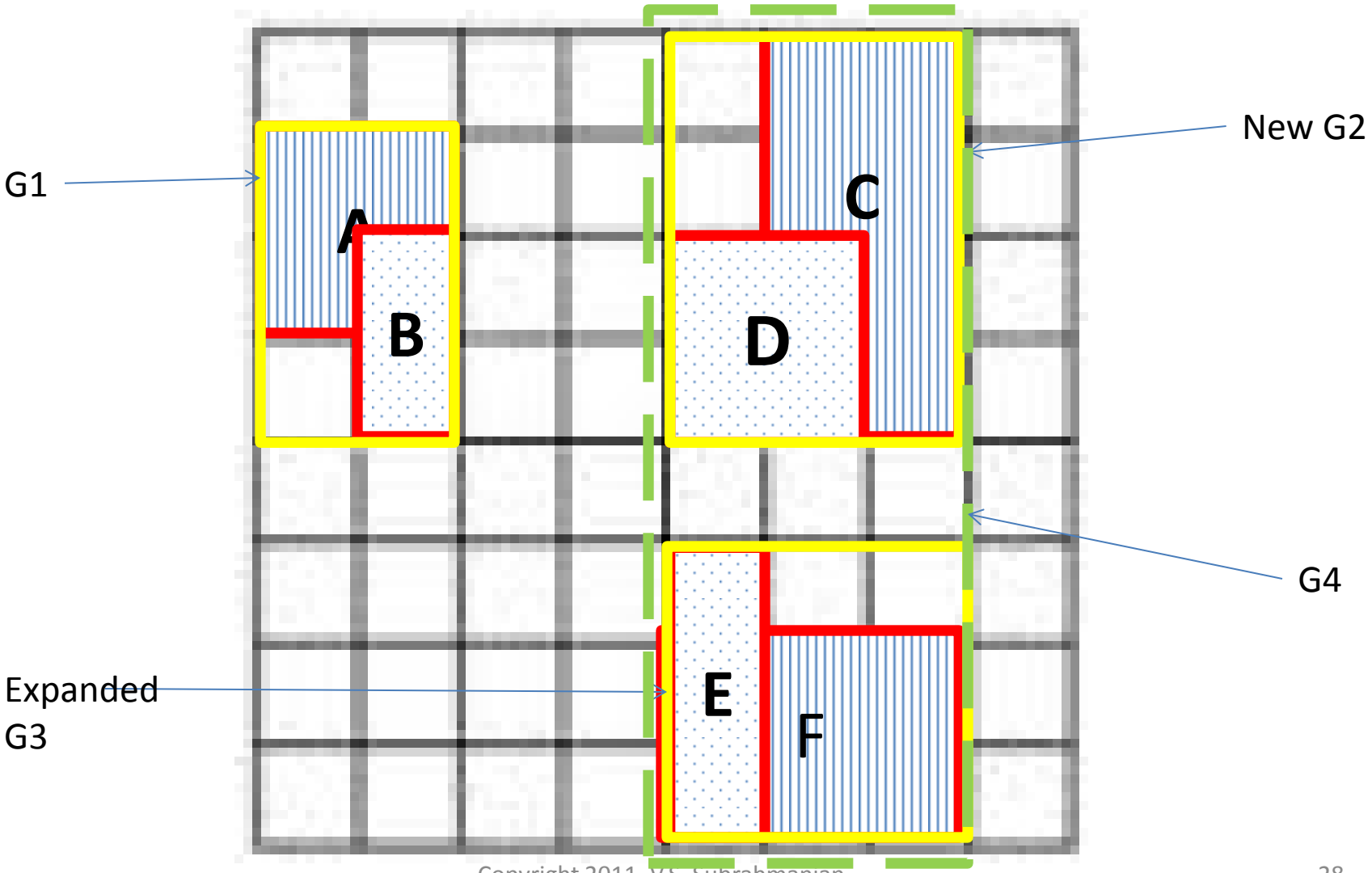
Now look at the node {G2,G3}. Which area would have to be expanded least in order to accommodate F? G2 or G3? Clearly G3. So branch right. There is space here, so insert.

Insert F

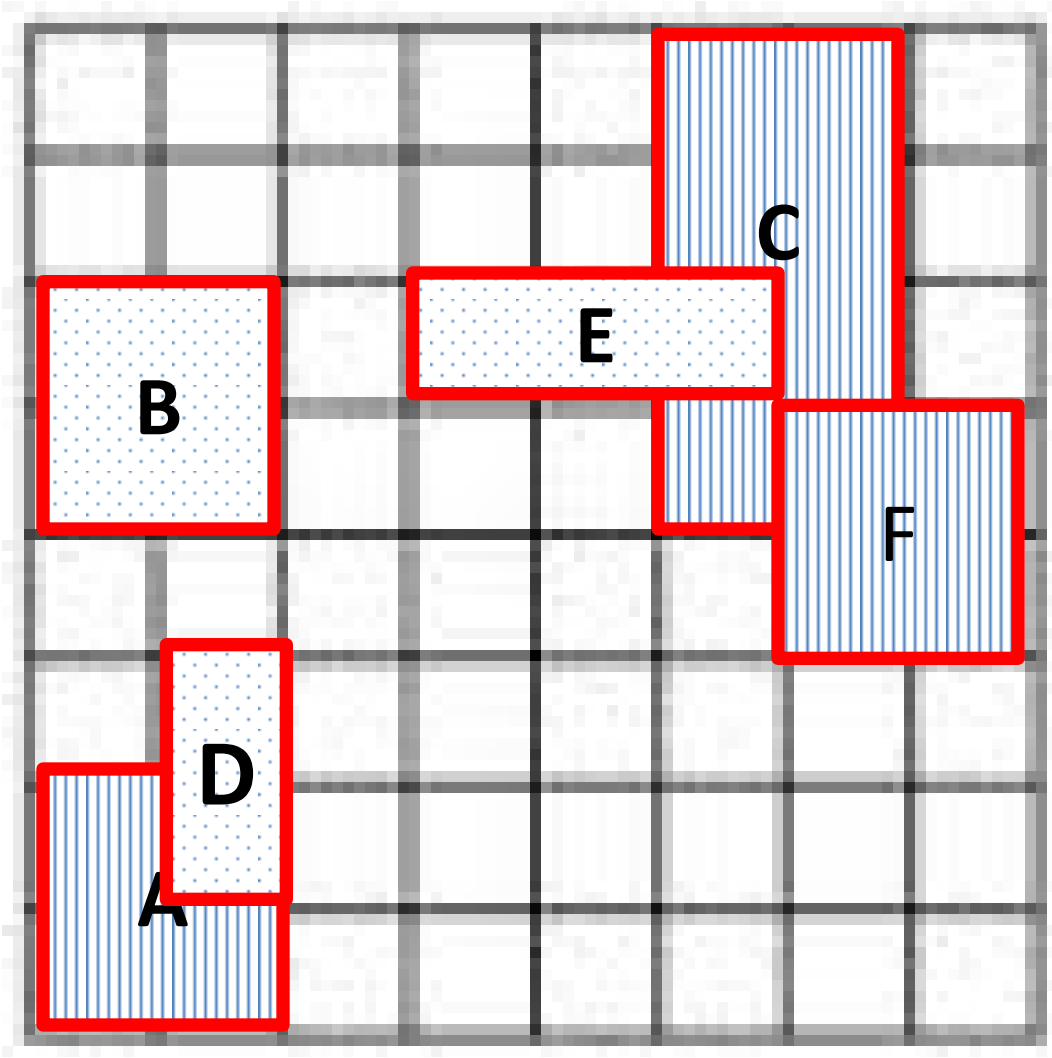


Now look at the node {G1,G3}. Which area would have to be expanded least in order to accommodate F? G2 or G3? Clearly G3. So branch right. There is space here, so insert.

Example: Insert F



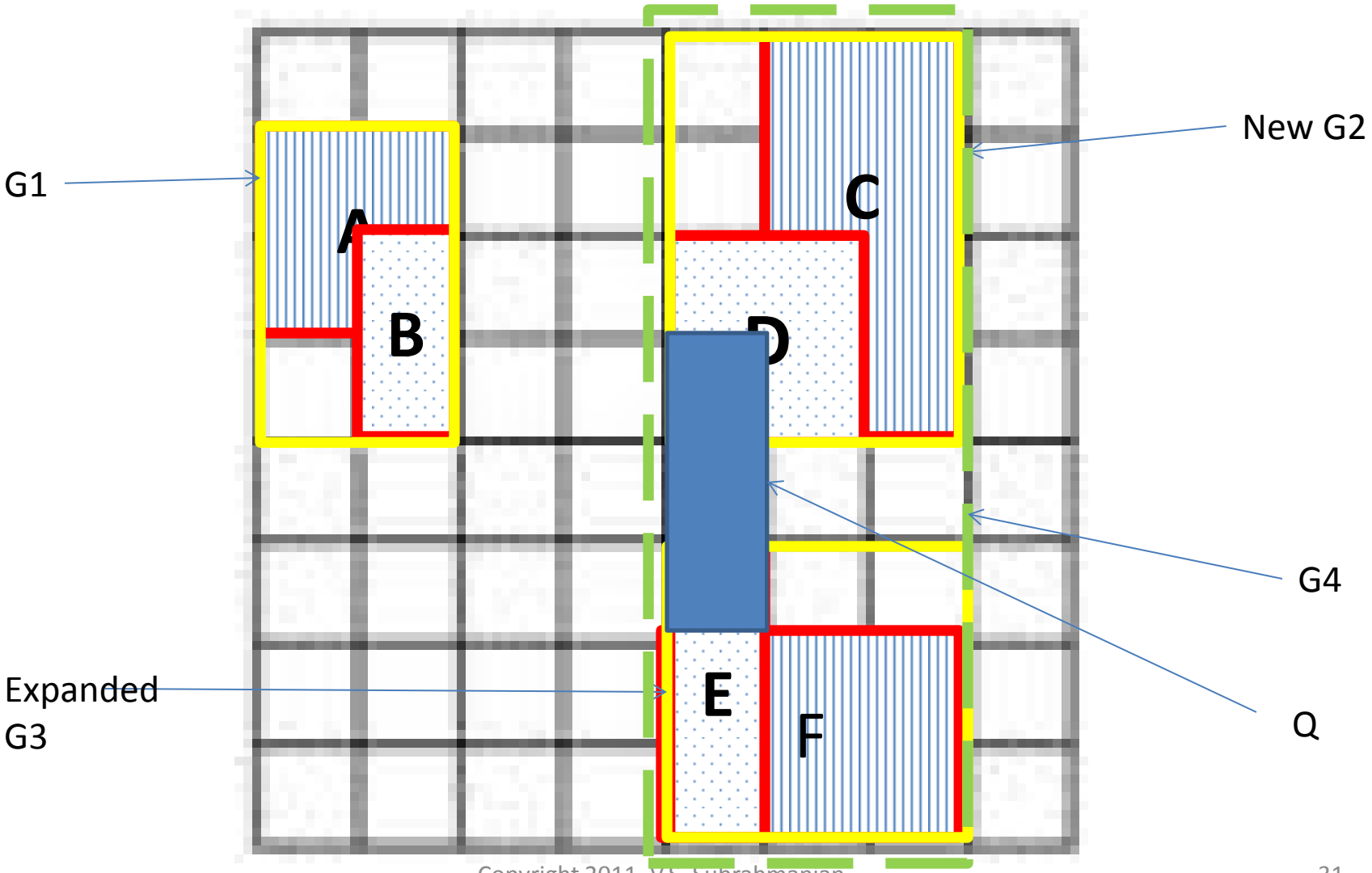
In-class Exercise: Build an R-Tree



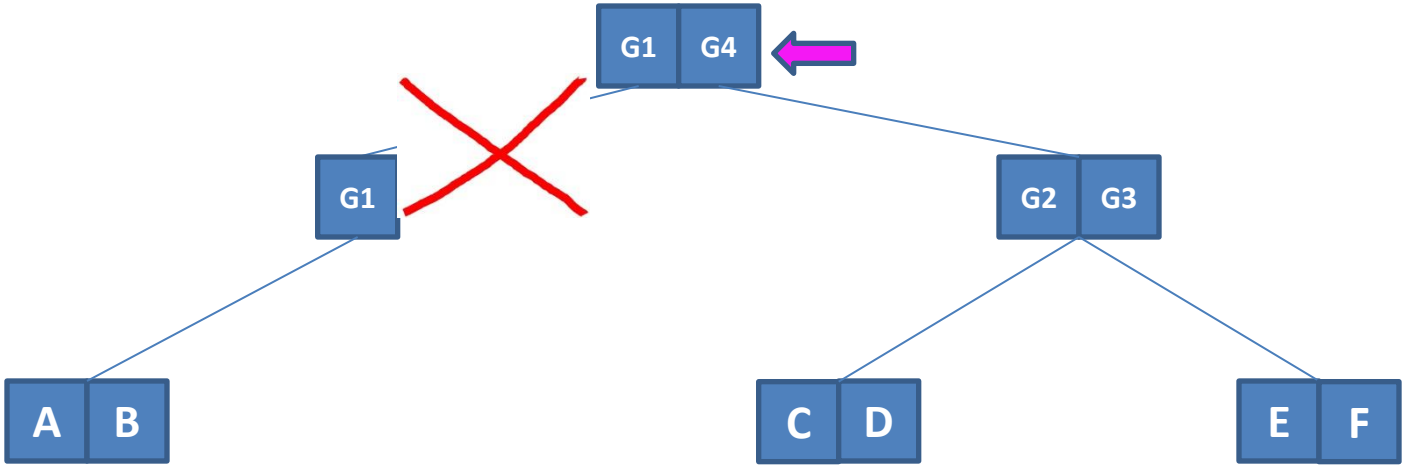
Range Queries

- INPUT: An R-Tree T and a query rectangle Q.
- OUTPUT: All rectangles R in T that intersect Q.
- **Algorithm sketch:**
- **VISIT-NODE**
  - If N is not a leaf, then check each synthetic rectangle S labeling N. If S intersects Q, then visit S' child, otherwise prune that child.
  - If N is a leaf, check each rectangle R labeling N. If R intersects Q, insert R into the solution.

Example

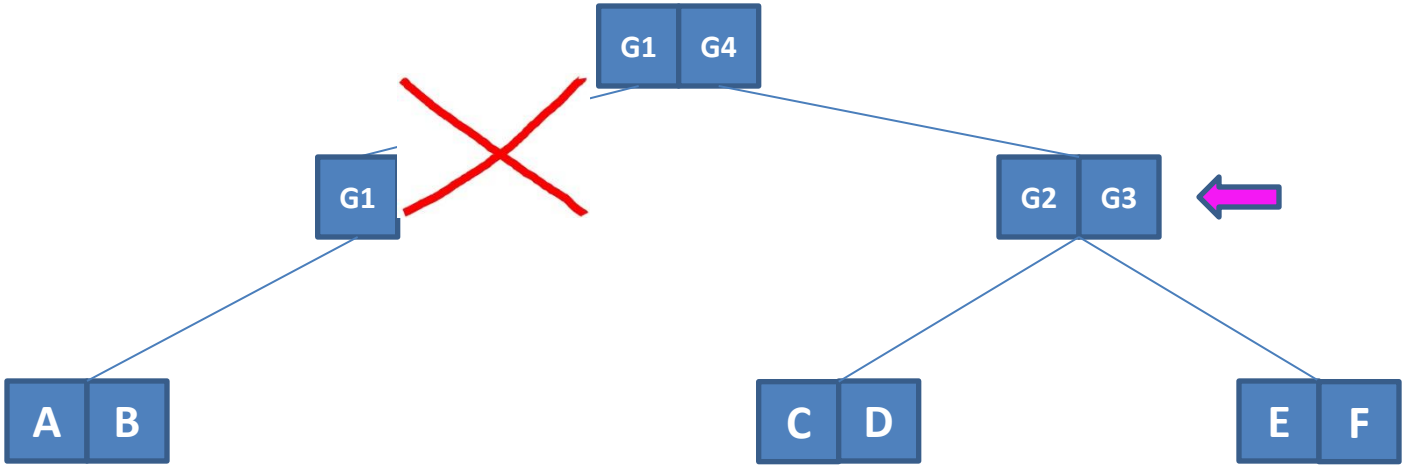


Range Query Q



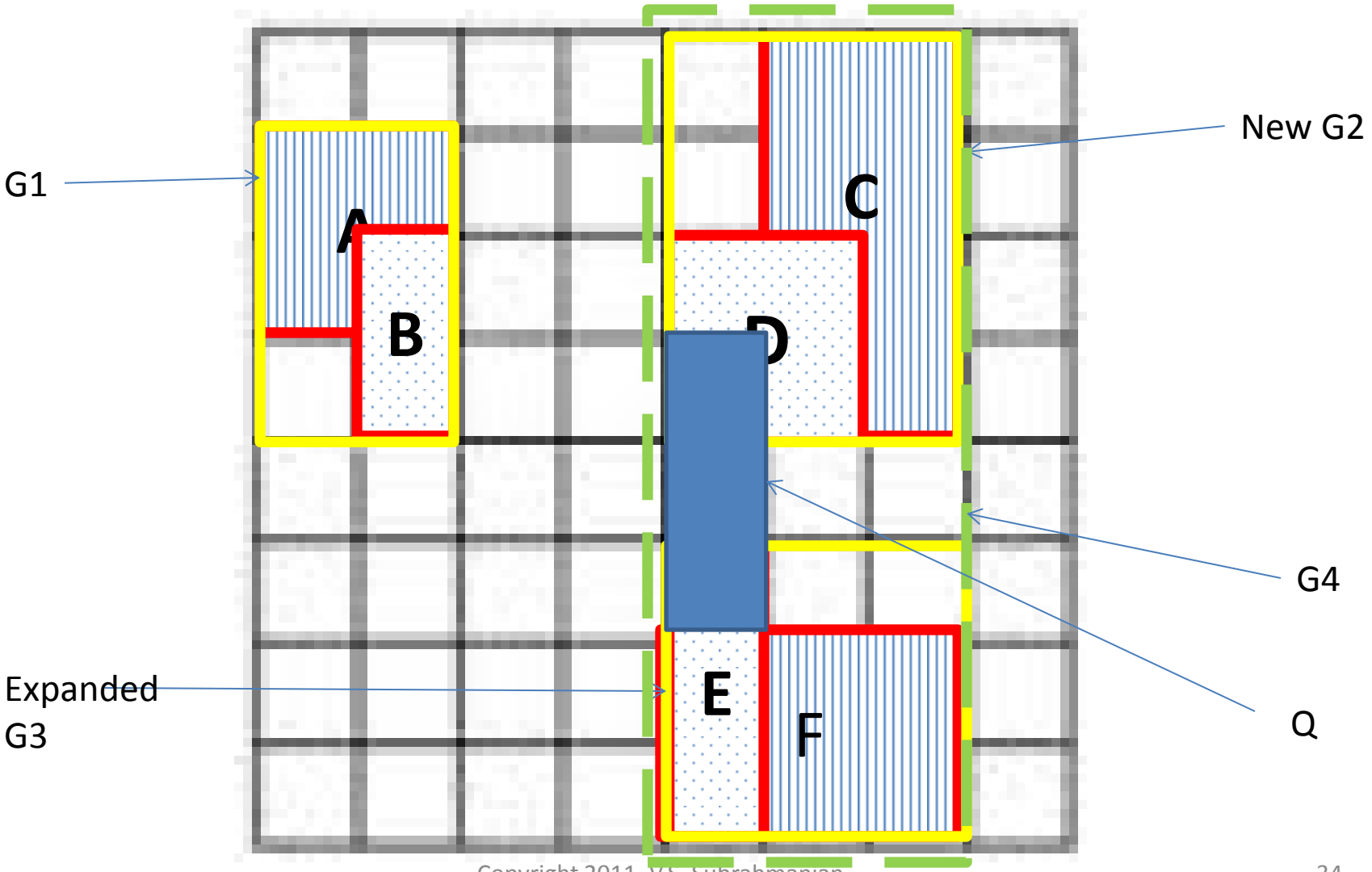
Root is not a leaf node. Check if Q intersects G1 and G4. Q does NOT intersect G1 – so can prune the entire subtree associated with G1. Q does intersect G4 – so must consider that.

Range Query Q

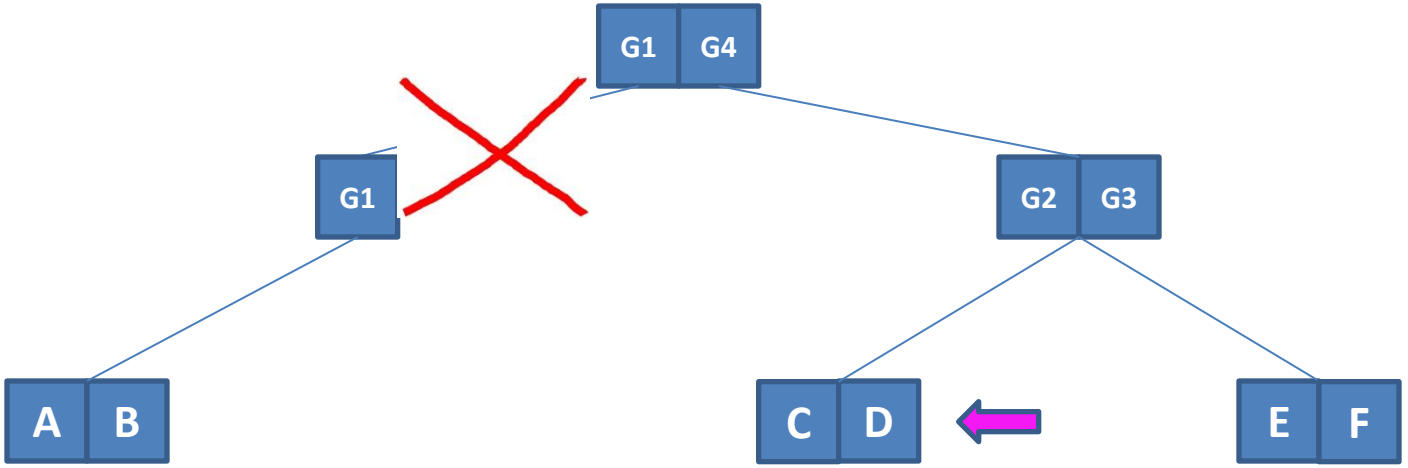


The node being visited is not a leaf. Check to see if Q intersects G2, G3. It intersects both, so must check both.

Example: Range Query Q



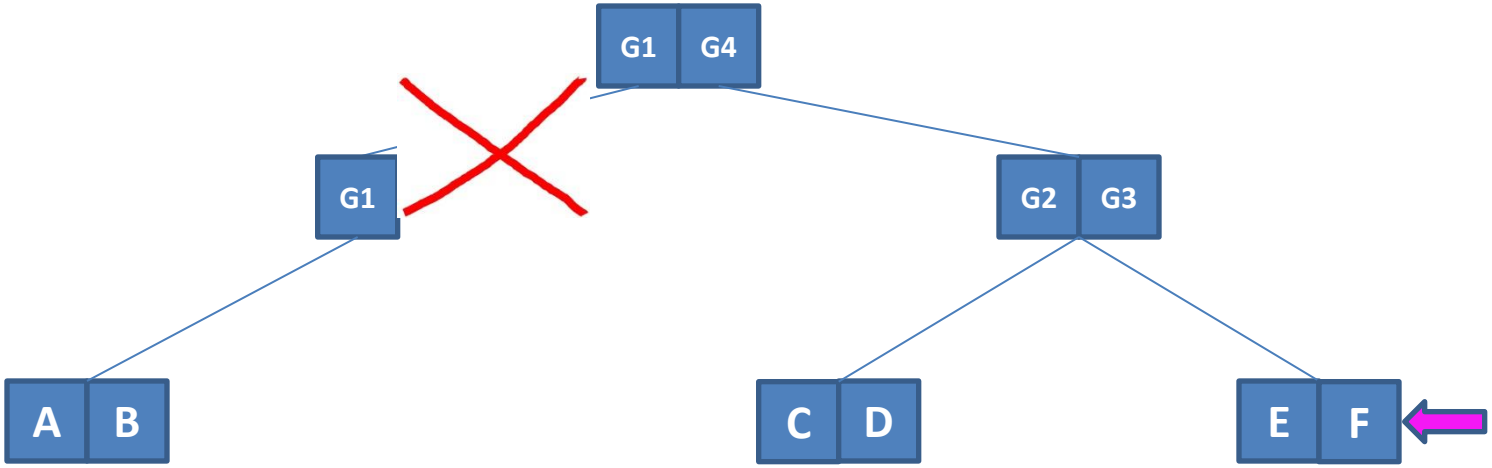
Range Query Q



SOLUTION
{C,D}

The node being visited is a leaf. Check to see if Q intersects C,D. it intersects both (because edges overlap.).

Range Query Q



SOLUTION
{C,D,E,F}

The node being visited is a leaf. Check to see if Q intersects E,F. it intersects both (because edges overlap.). Done.

In-class Exercise: Range Query Q