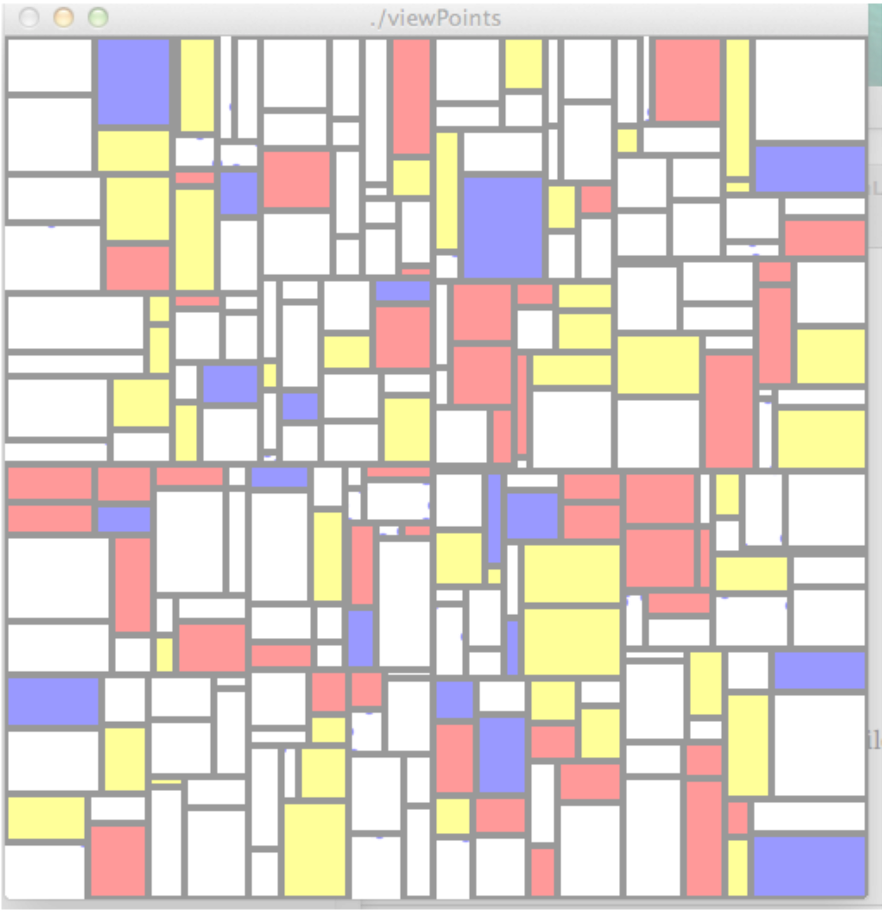
kd-trees

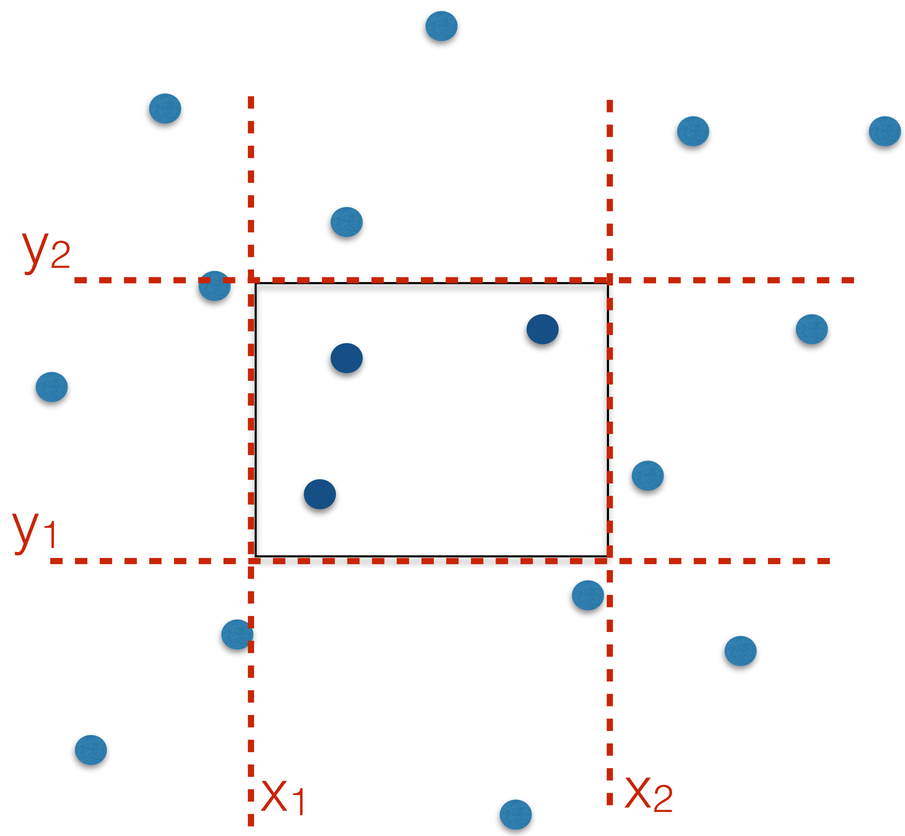
k-dimensional search trees



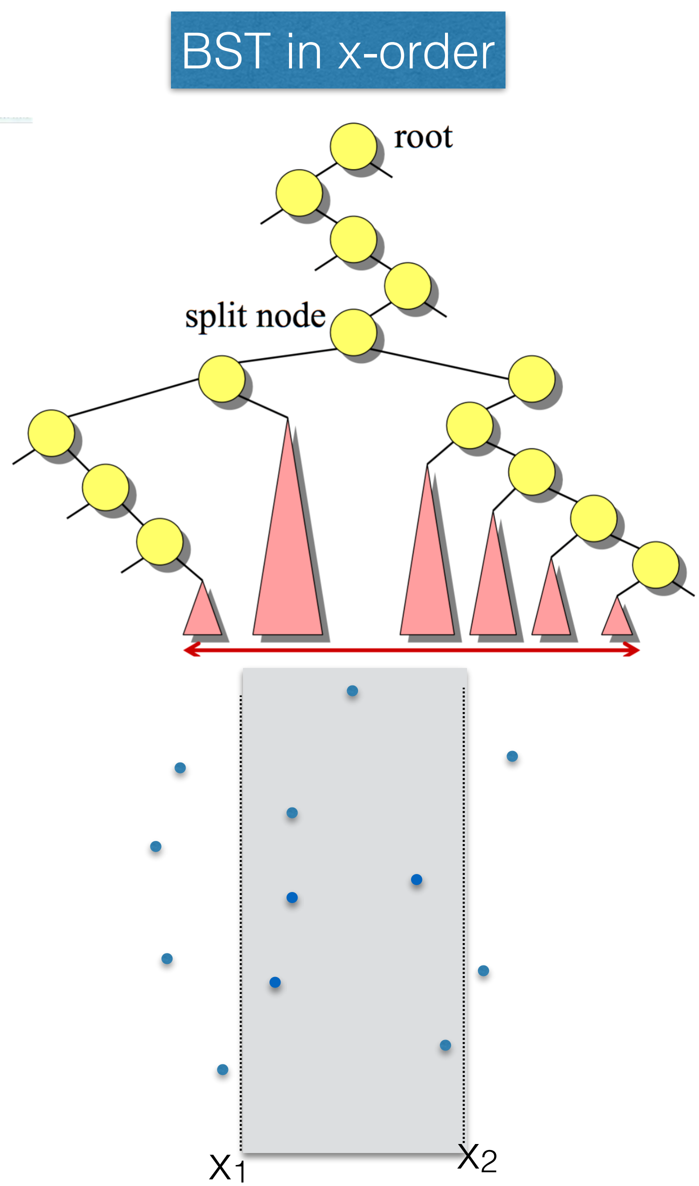
2D Range searching

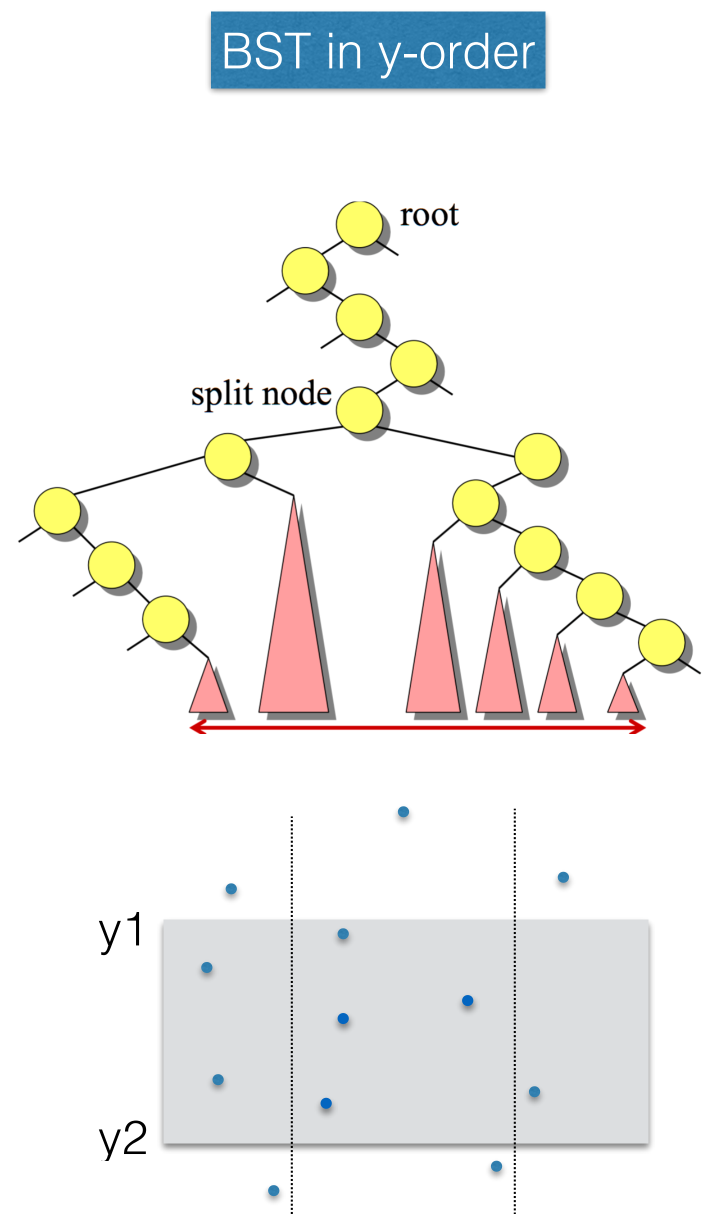
Given a set of n points in 2D and an arbitrary range $[x_1, x_2] \times [y_1, y_2]$, find all points in this range

Build a structure to answer this efficiently

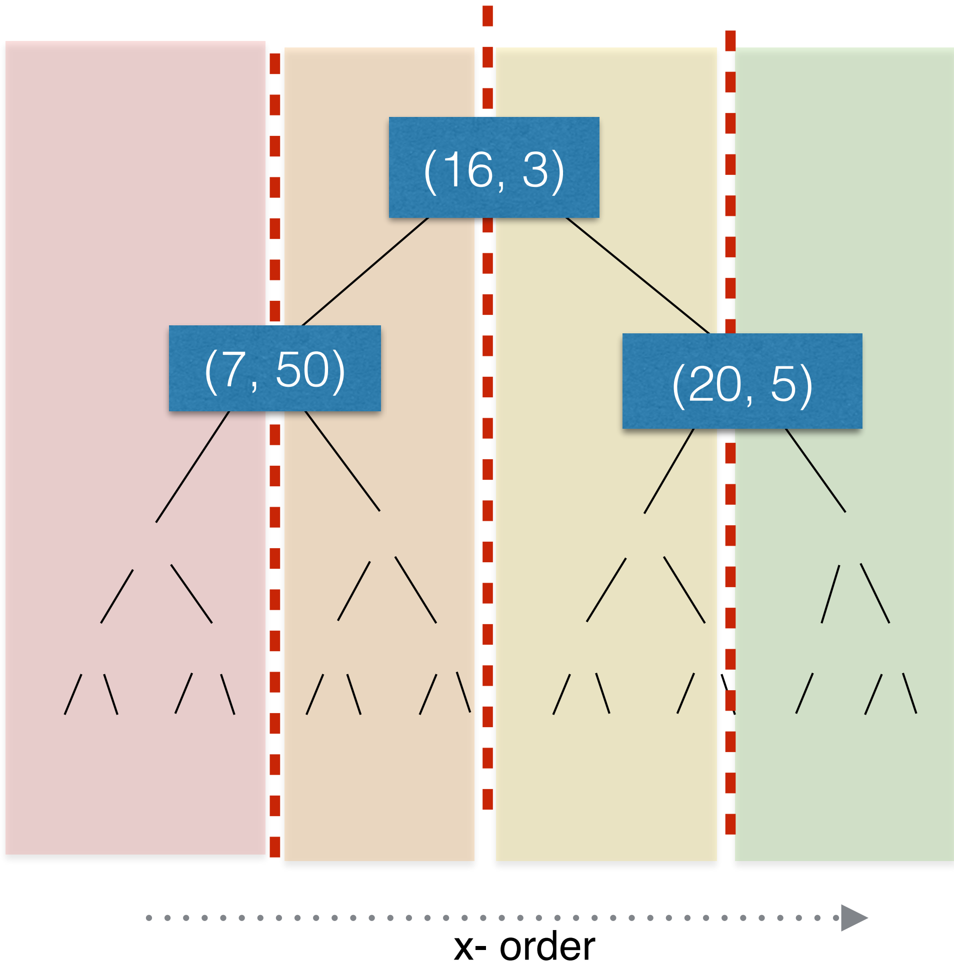


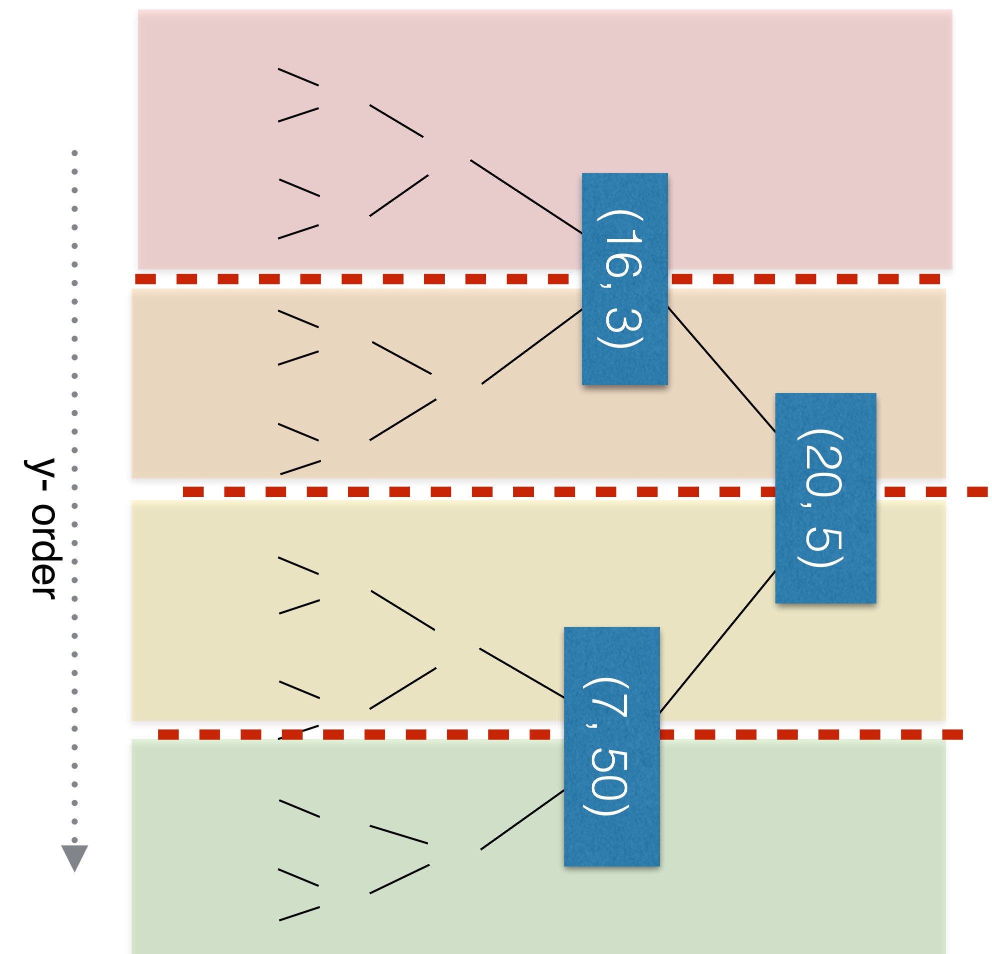
A BST is a 1D structure. If it is ordered by x, it can answer x-range queries. If it is ordered by y, it can answer y-range queries.





A BST is a 1D structure. If it is ordered by x, it can answer x-range queries. If it is ordered by y, it can answer y-range queries.

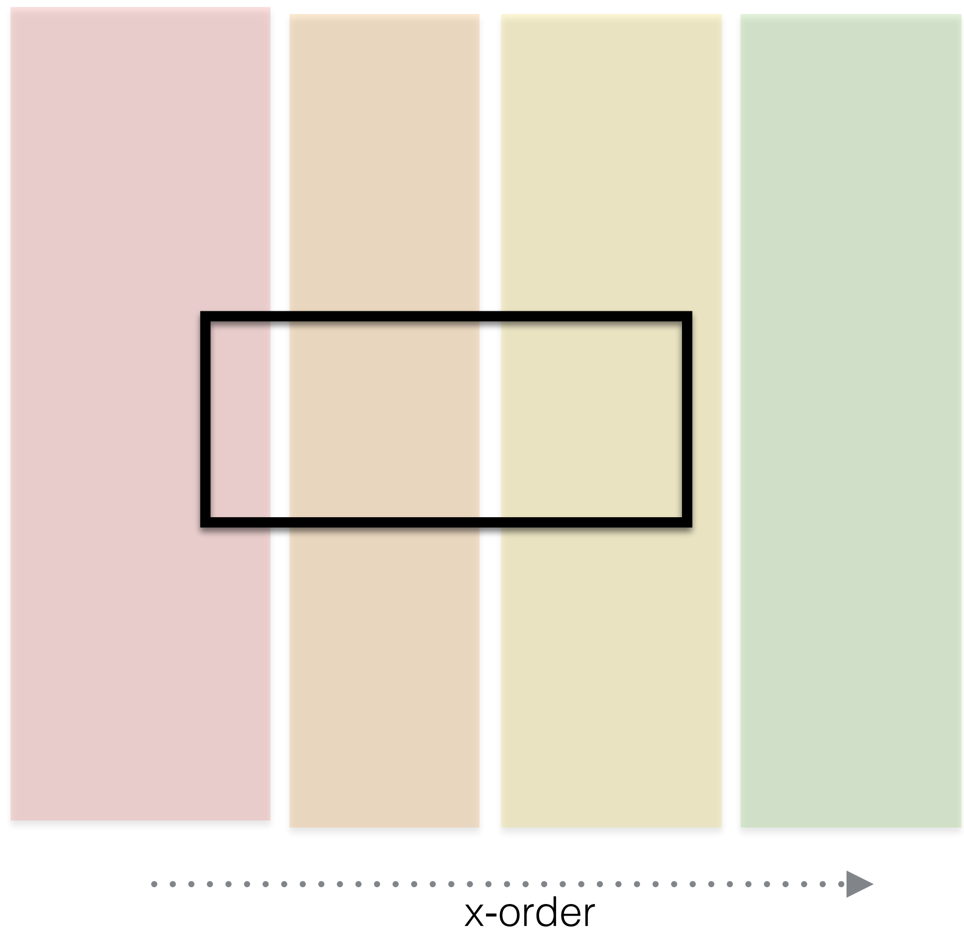




Partitions the space in vertical strips

Partitions the space in horizontal strips

Not a good partition for range-searching!

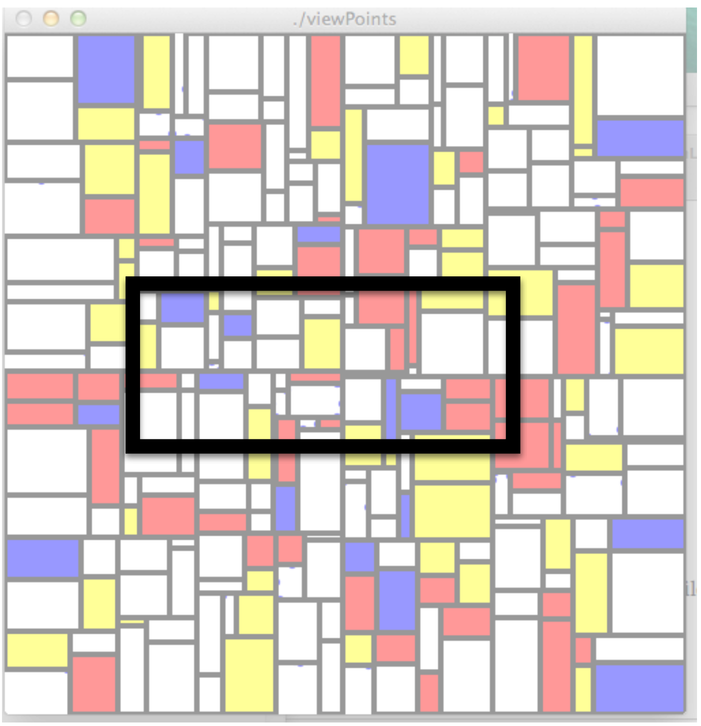


We have to search all vertical strips that intersect the range, which could have a lot of points outside the range.

kd-trees

k-dimensional search trees

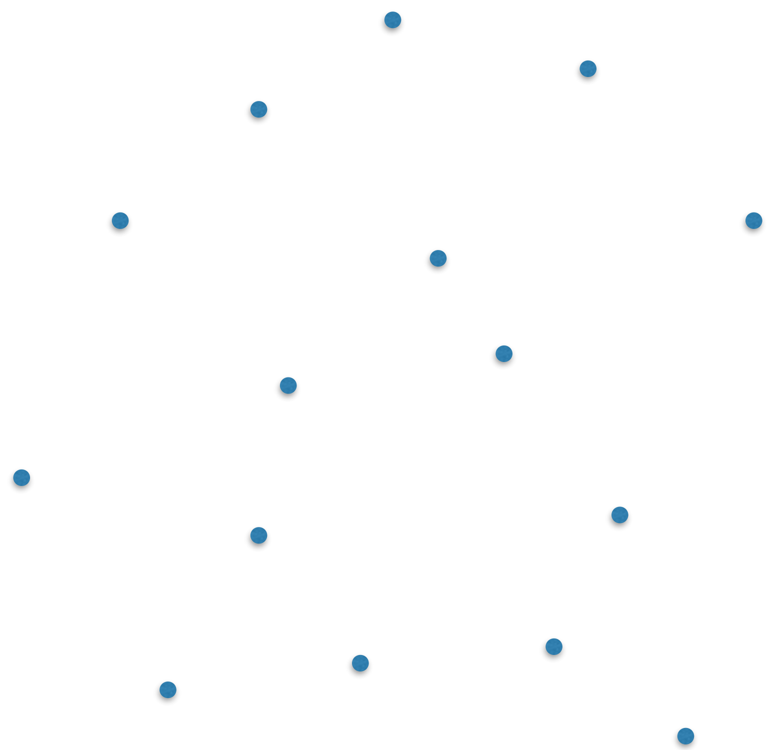
Jon Bentley, in mid 1970s, while an undergrad, came up with this beautifully simple idea to extend the BST to make it useful in both x- and y- dimensions



Space partition of a 2d-search tree

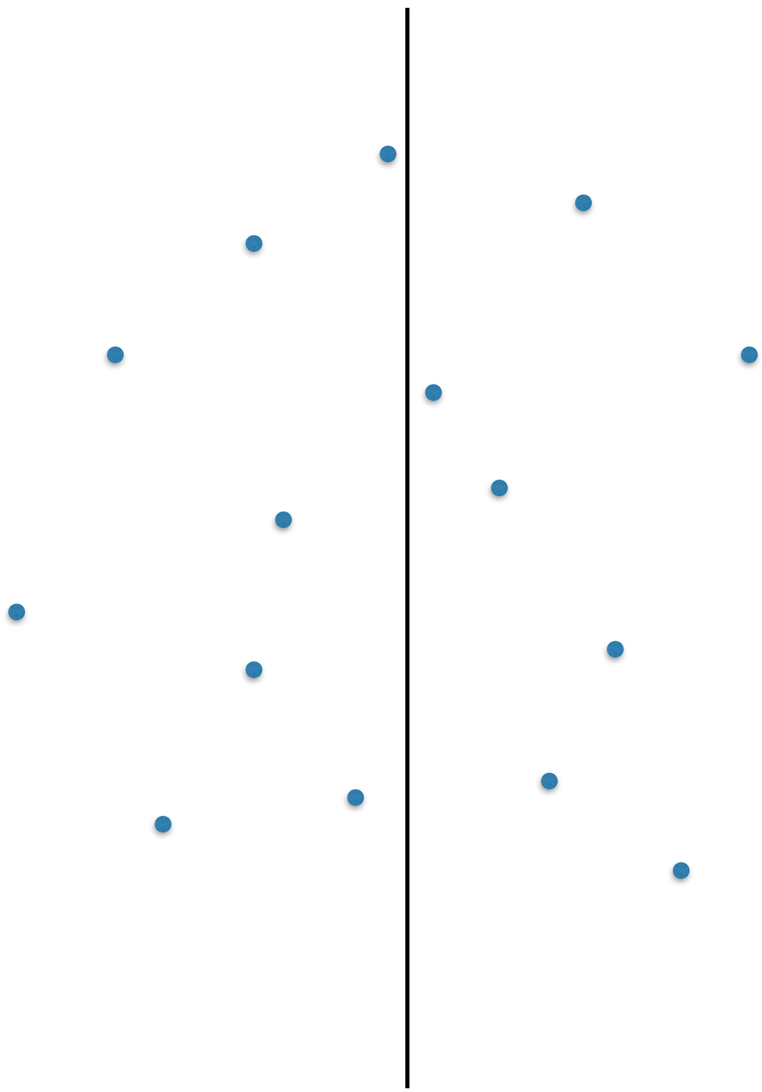
- **The idea:** recursively subdivide the plane by vertical and horizontal cut lines which alternate
- Cut lines are chosen to split the points in half ==> logarithmic height

The 2d-search tree



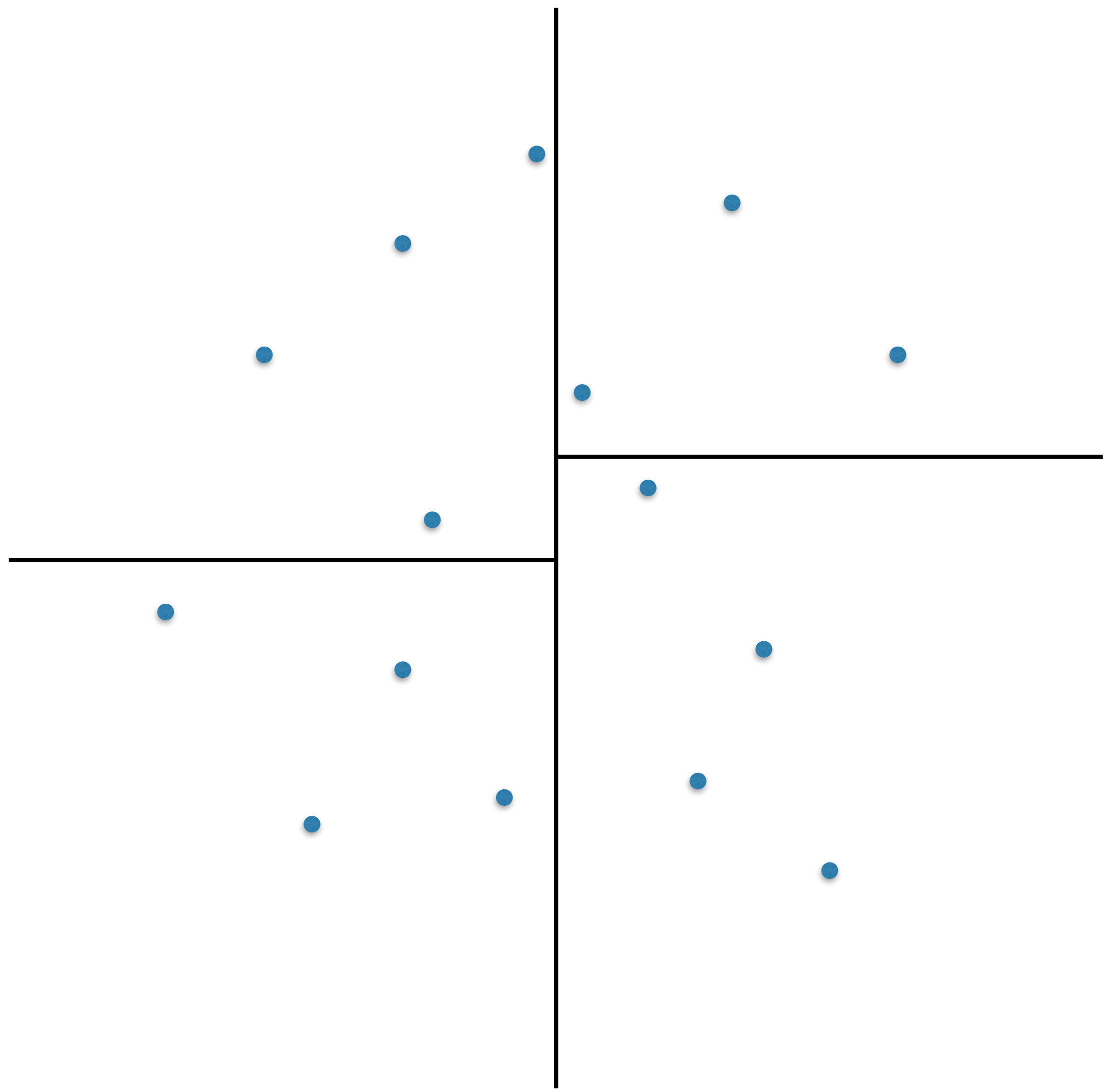
The 2d-search tree

split points in two halves with a **vertical** line



The 2d-search tree

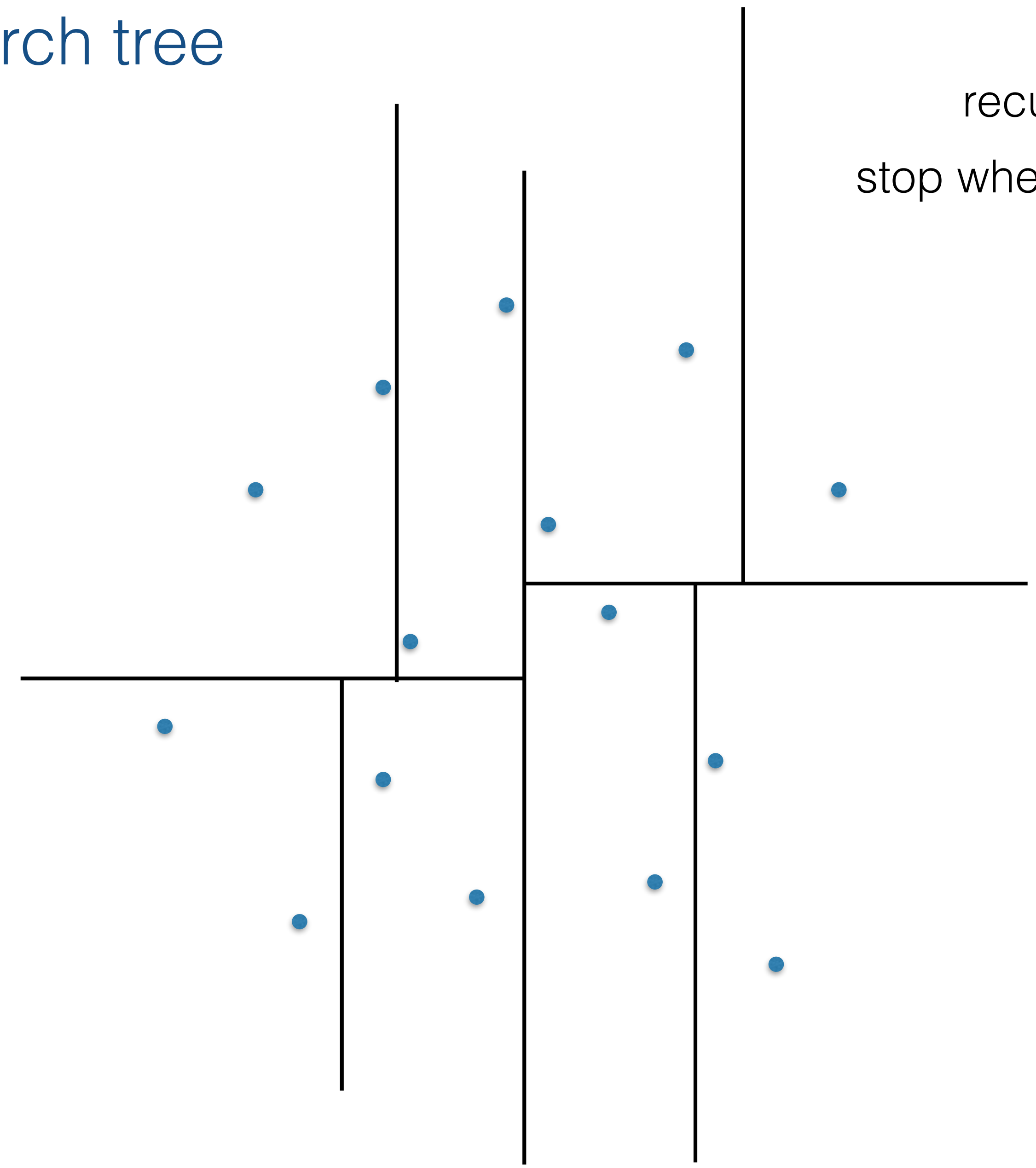
split each side into half with a **horizontal** line



The 2d-search tree

recurse

stop when a region has 1 point



The 2d-search tree

stop when a region has 1 point

The 2d-search tree

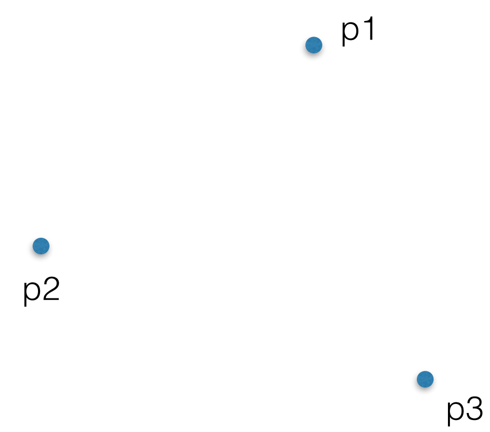
Couple of variants based on how exactly to choose the splitting line

- Choose the cut line so that it falls in between the points. Internal nodes store lines, and points are only in leaves.
- Chose the cut line so that it goes through the median point, and store the median in the internal node.
- Choose the cut line so it goes through the median point. Internal nodes store lines, and points are only in leaves.

This is the standard choice and simplifies the details

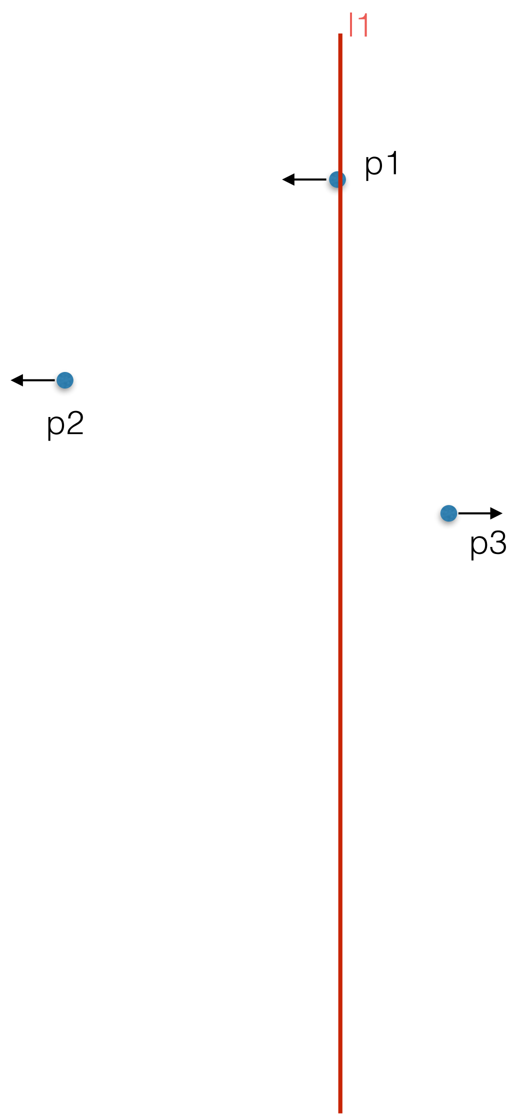
The 2d-search tree

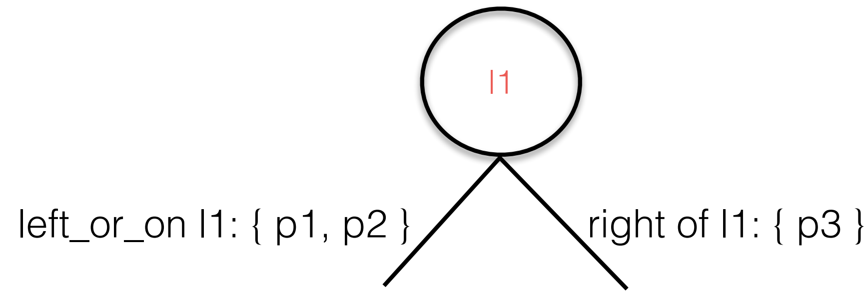
Include the median point to the **first** side, consistently.



The 2d-search tree

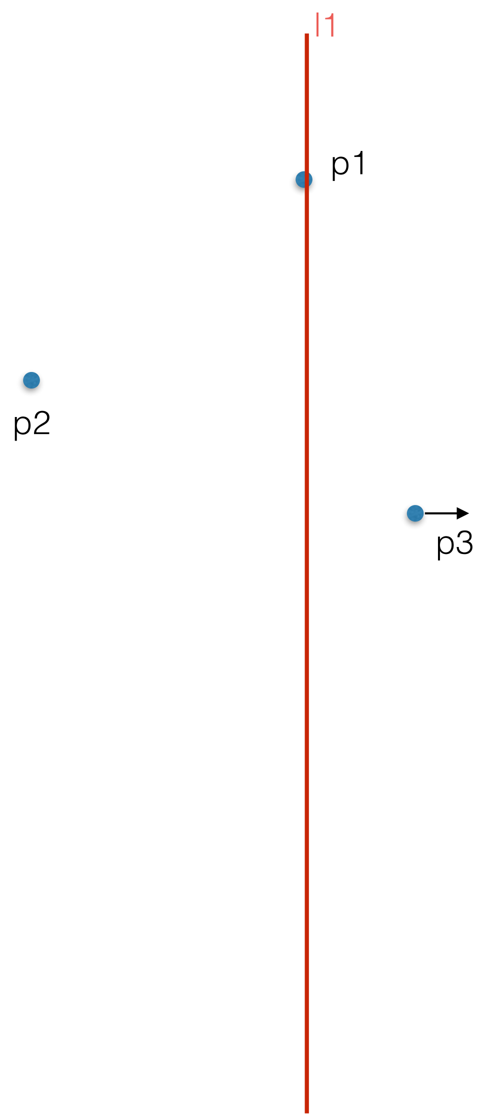
split with vertical line through x-median
include median in left child

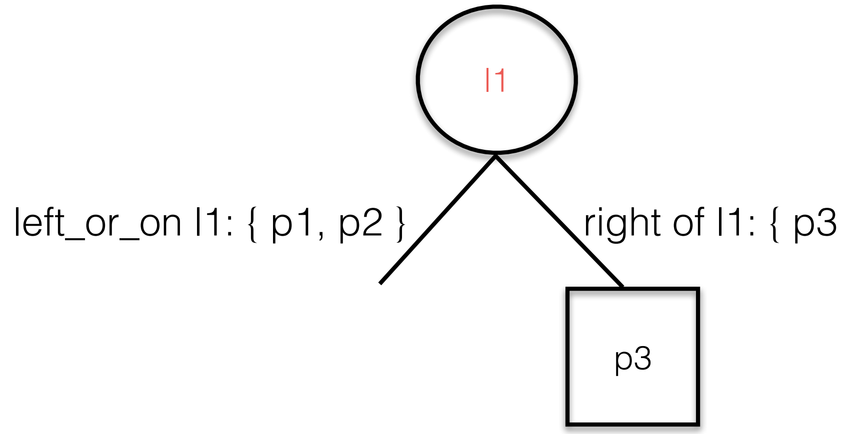




The 2d-search tree

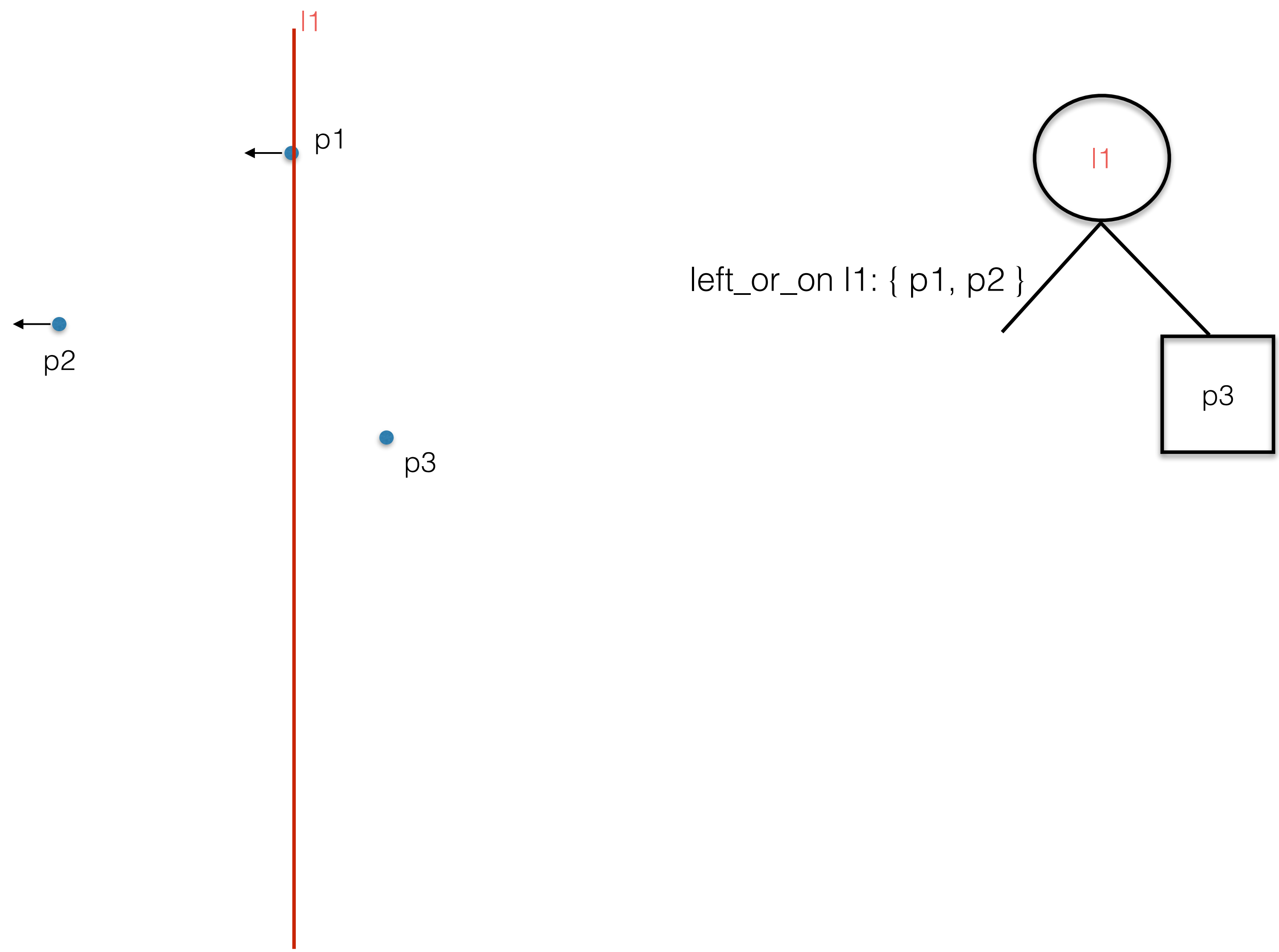
right of l1: p3 => leaf





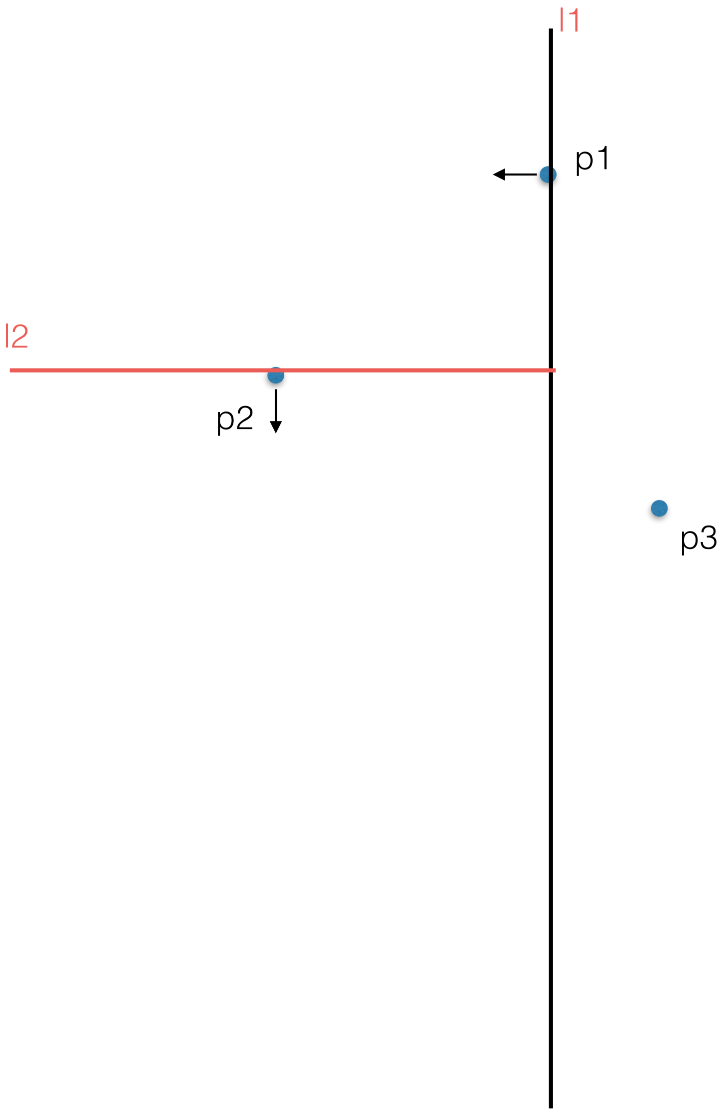
The 2d-search tree

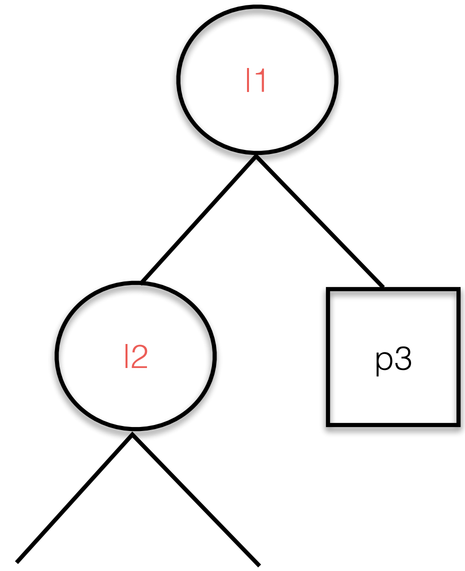
left_on l1: p1,p2 => recurse



The 2d-search tree

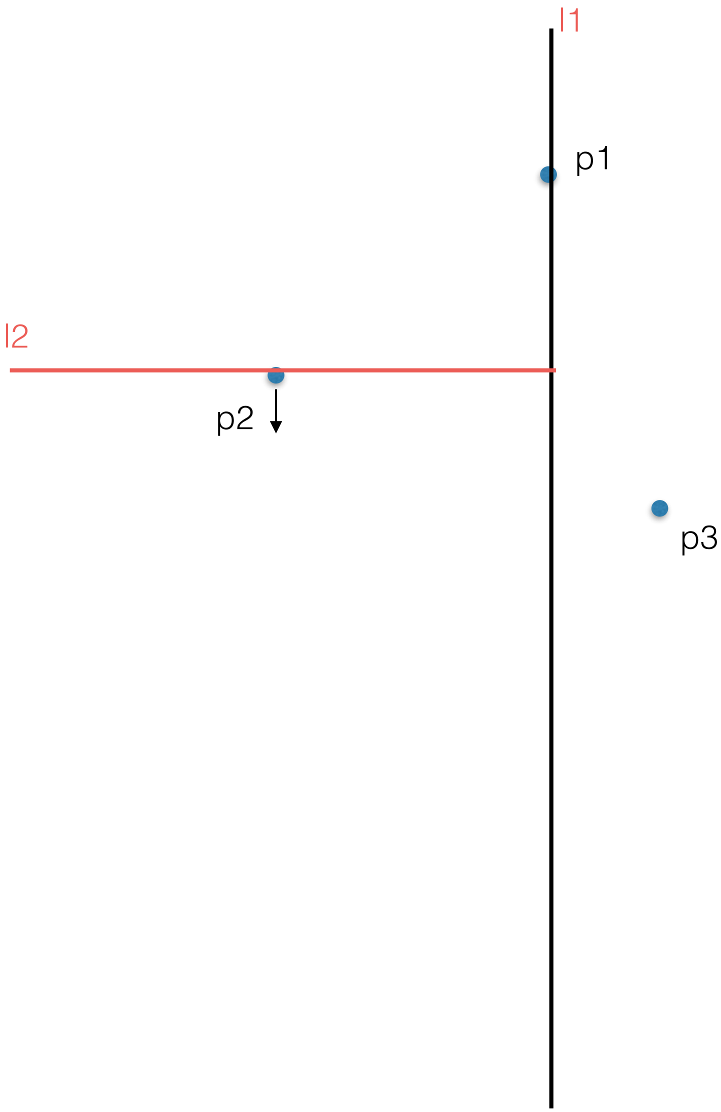
split with horizontal line through y-median
include median in left child

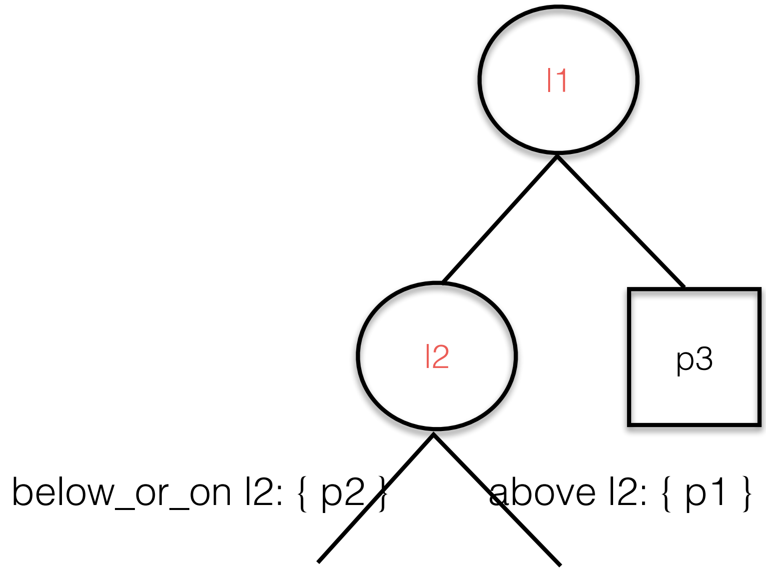




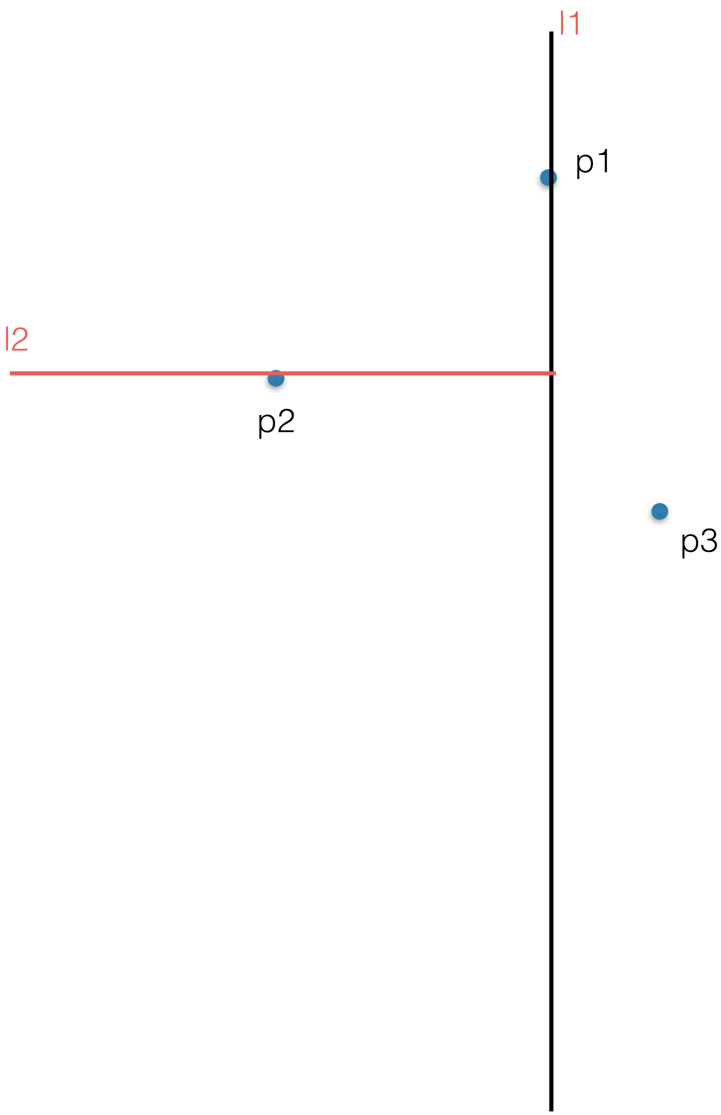
The 2d-search tree

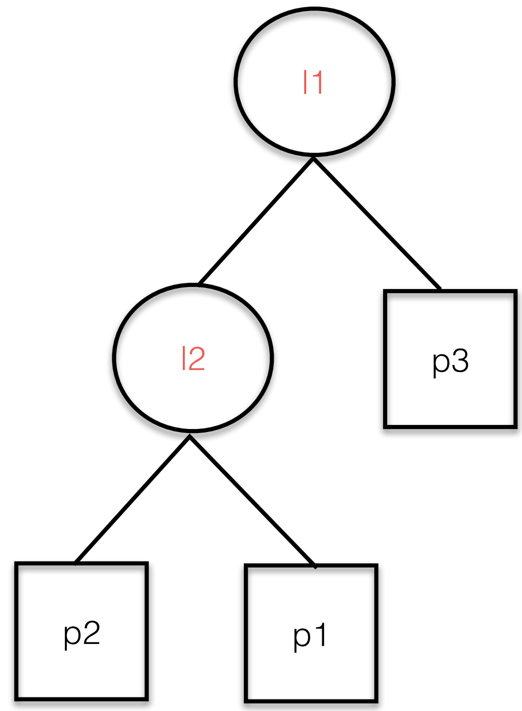
split with horizontal line through y-median
include median in left child



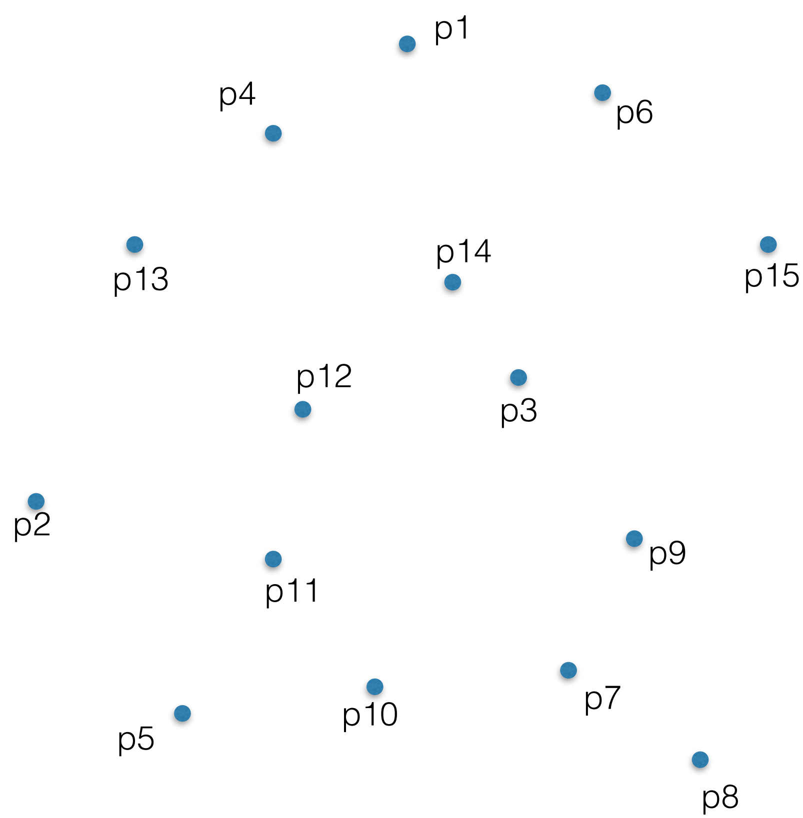


The 2d-search tree



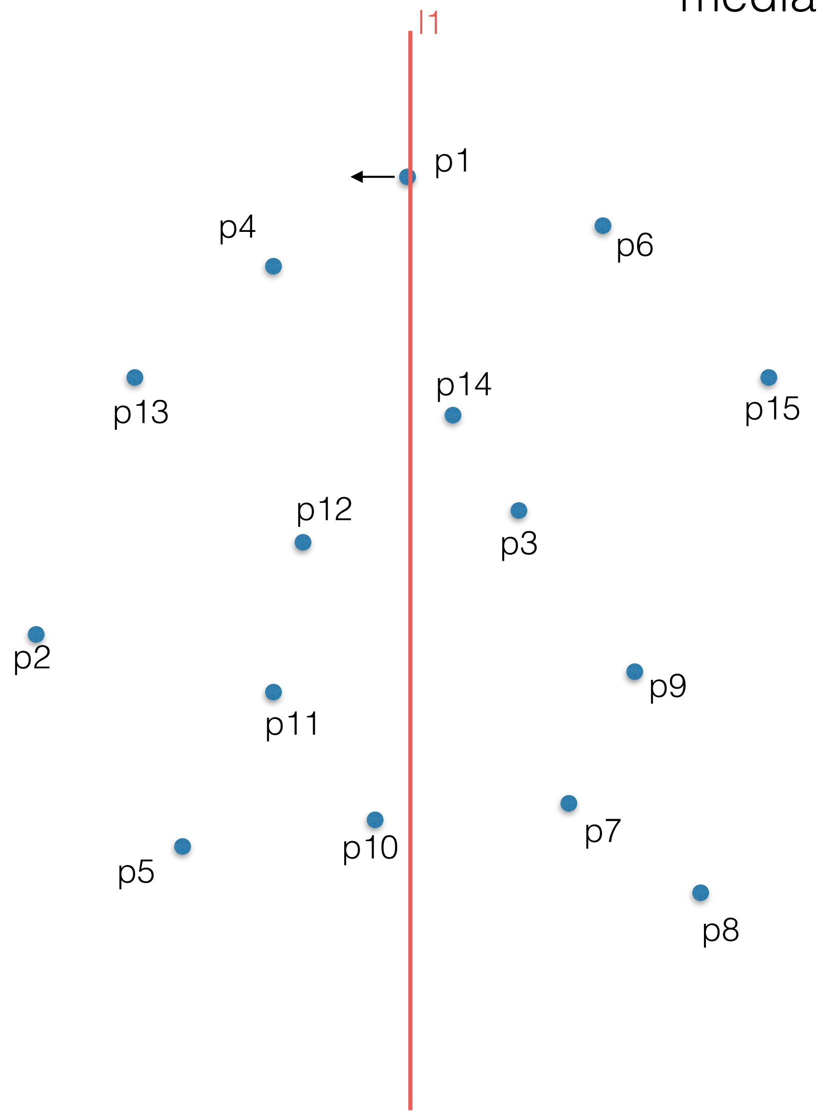


A bigger example



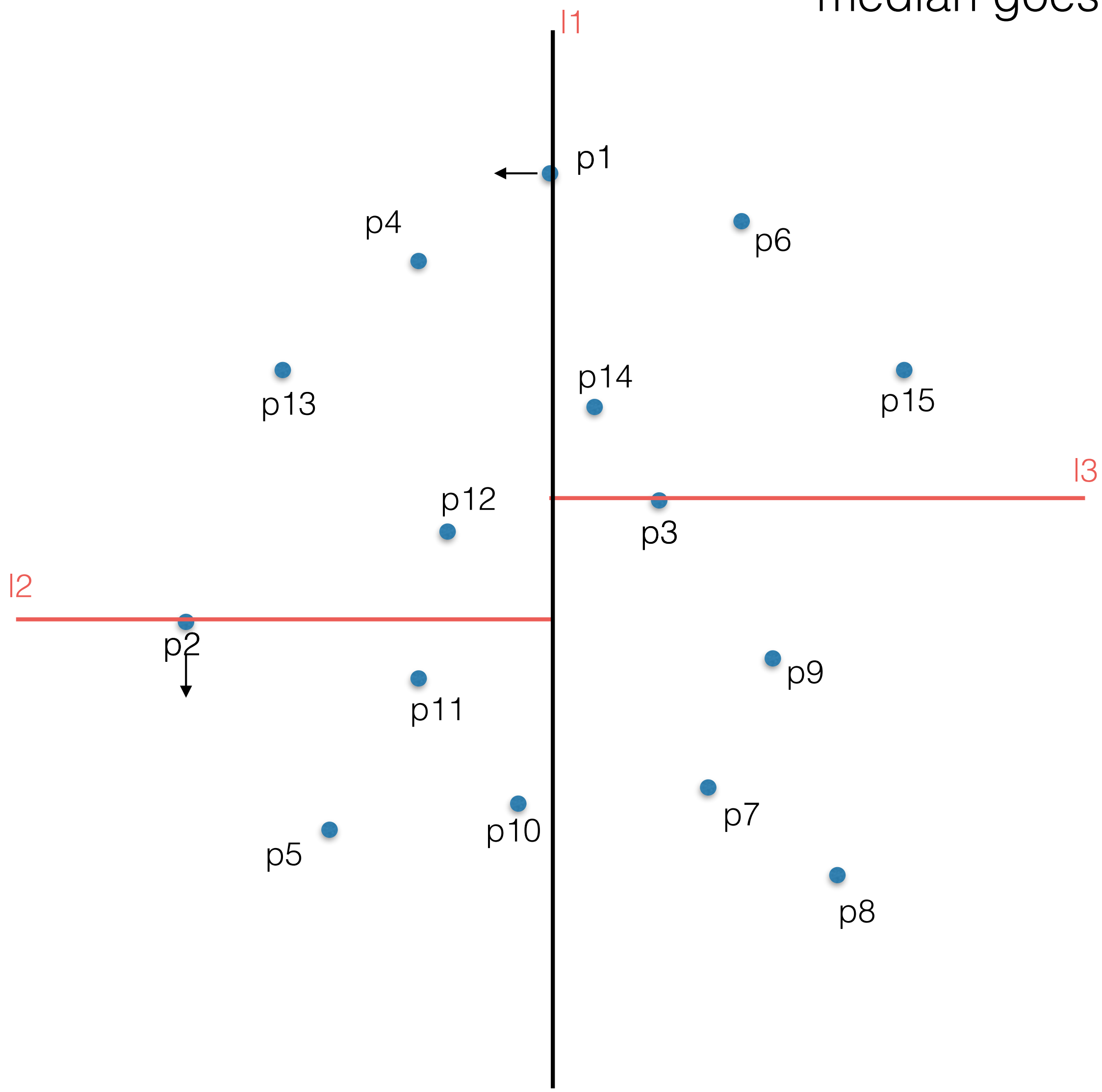
A bigger example

split with vertical line through x-median
median goes to the left side



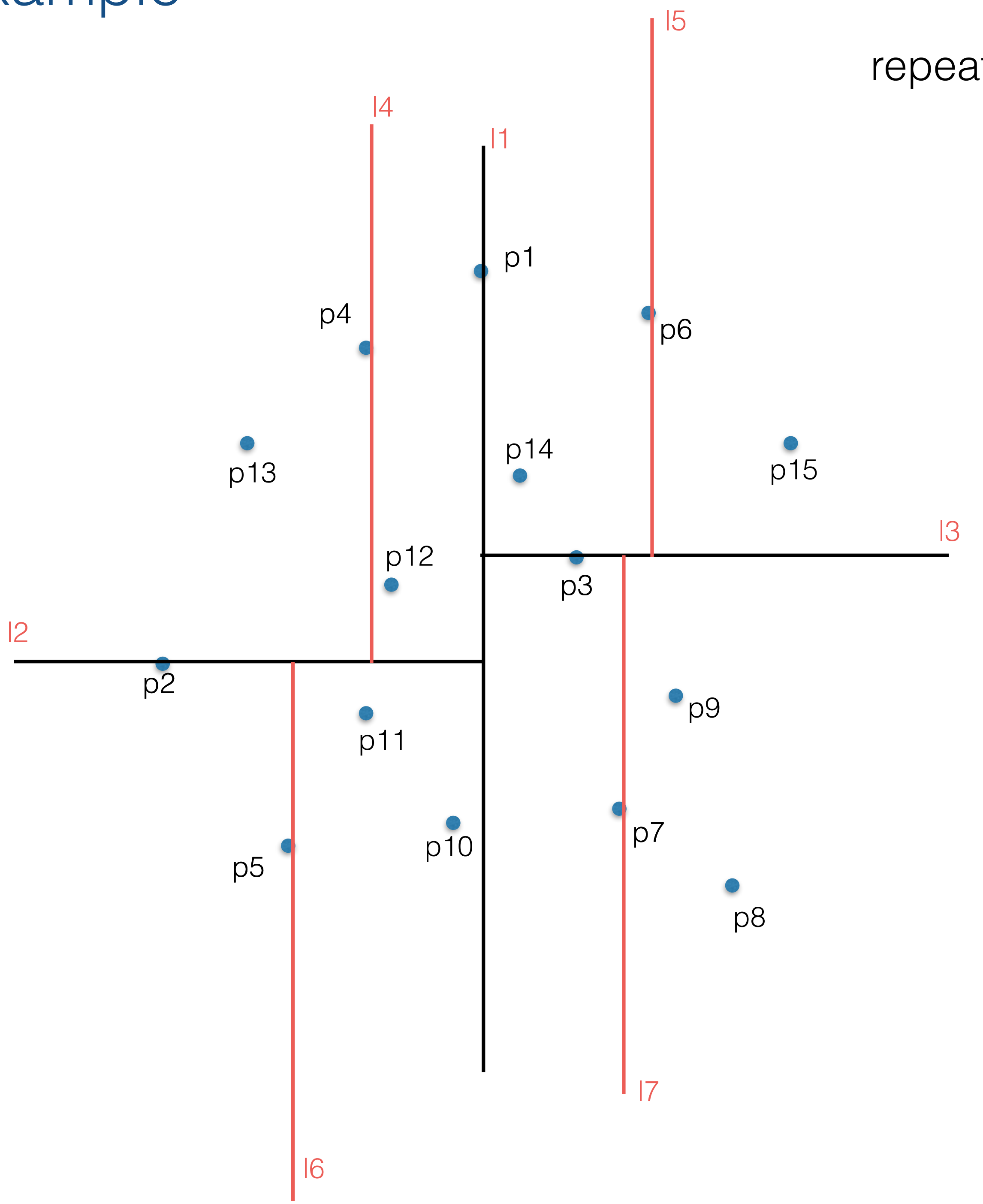
A bigger example

split each side with horizontal line through y-median
median goes to the left side

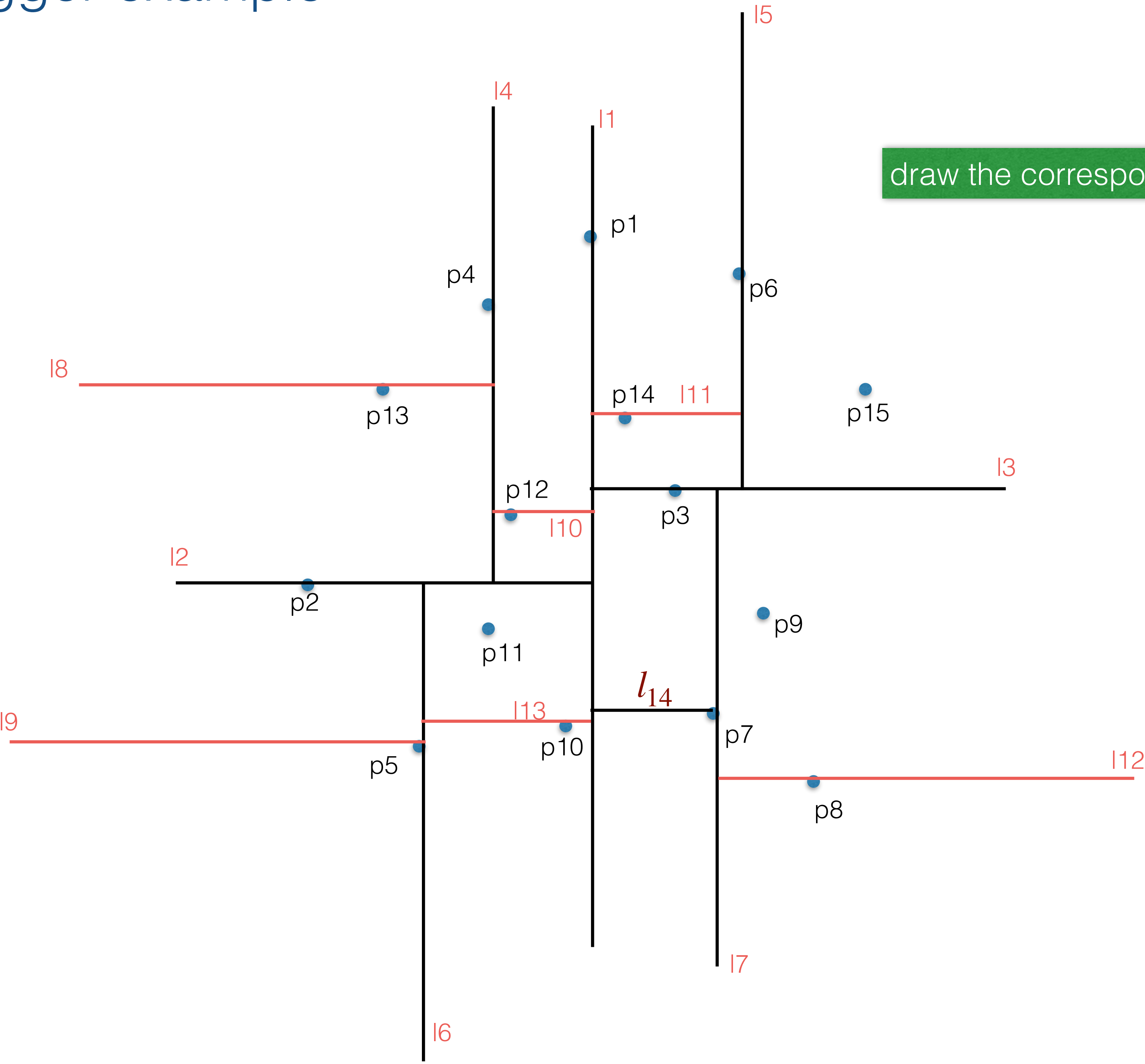


A bigger example

repeat



A bigger example



draw the corresponding 2d-tree

Given a set of points P:

How do we build their kd-tree?

screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

The 2d-search tree

Algorithm BUILDKDTREE(P, *depth*)

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** *depth* is even
4. **then** Split P with a vertical line ℓ through the median x-coordinate into P_1 (left of or on ℓ) and P_2 (right of ℓ)
5. **else** Split P with a horizontal line ℓ through the median y-coordinate into P_1 (below or on ℓ) and P_2 (above ℓ)
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE(P_1, $depth + 1$)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE(P_2, $depth + 1$)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

Analysis: Let T(n) be the time to build a kdtree of n points.

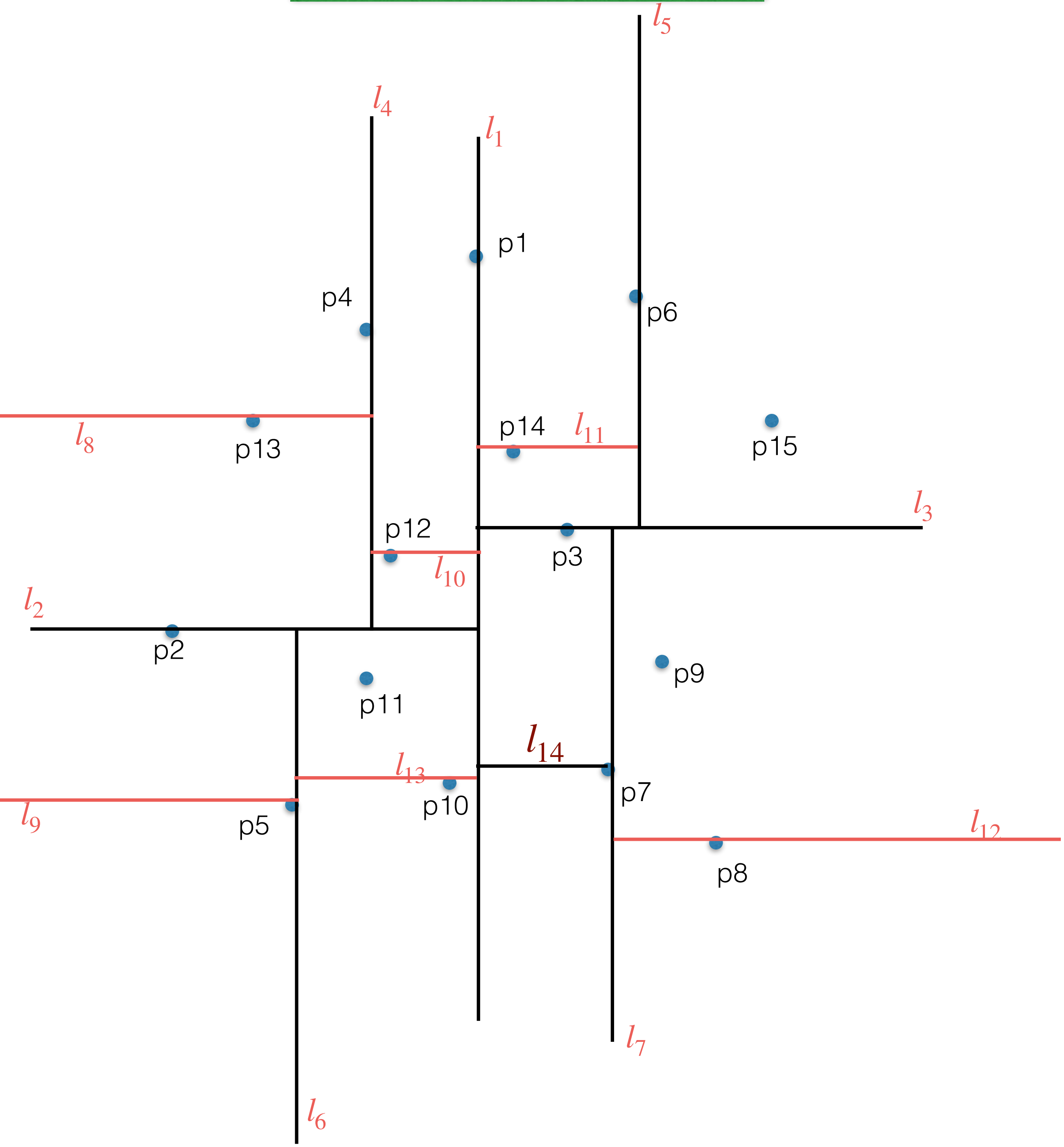
Then $T(n) = 2T(n/2) + O(n)$, which solves to $O(n \lg n)$.

The 2d-search tree

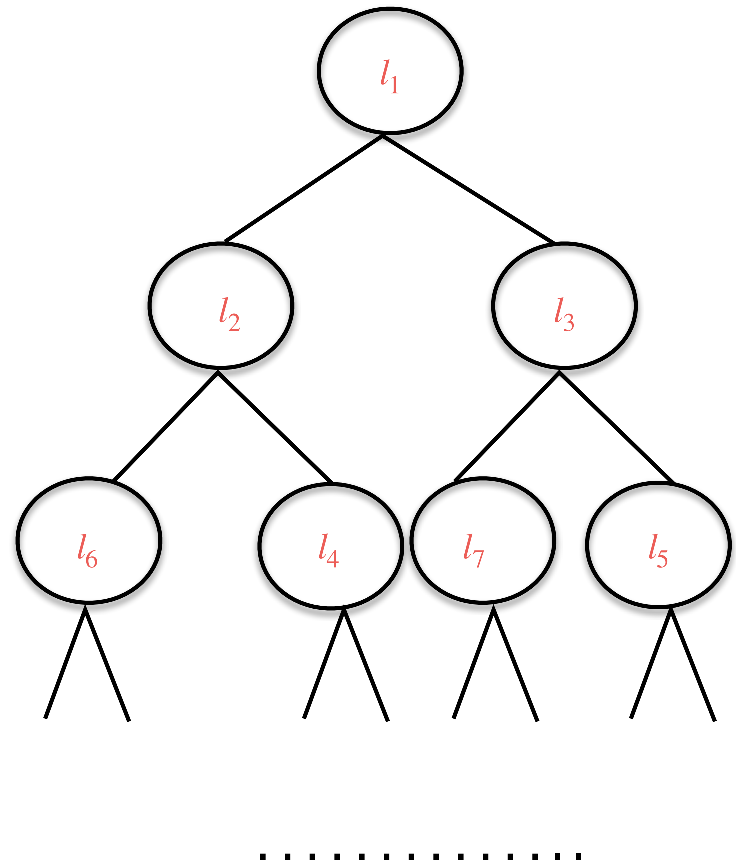
Theorem: A 2d-tree for a set of n points in the plane can be built in $\Theta(n \lg n)$ time and uses $\Theta(n)$ space.

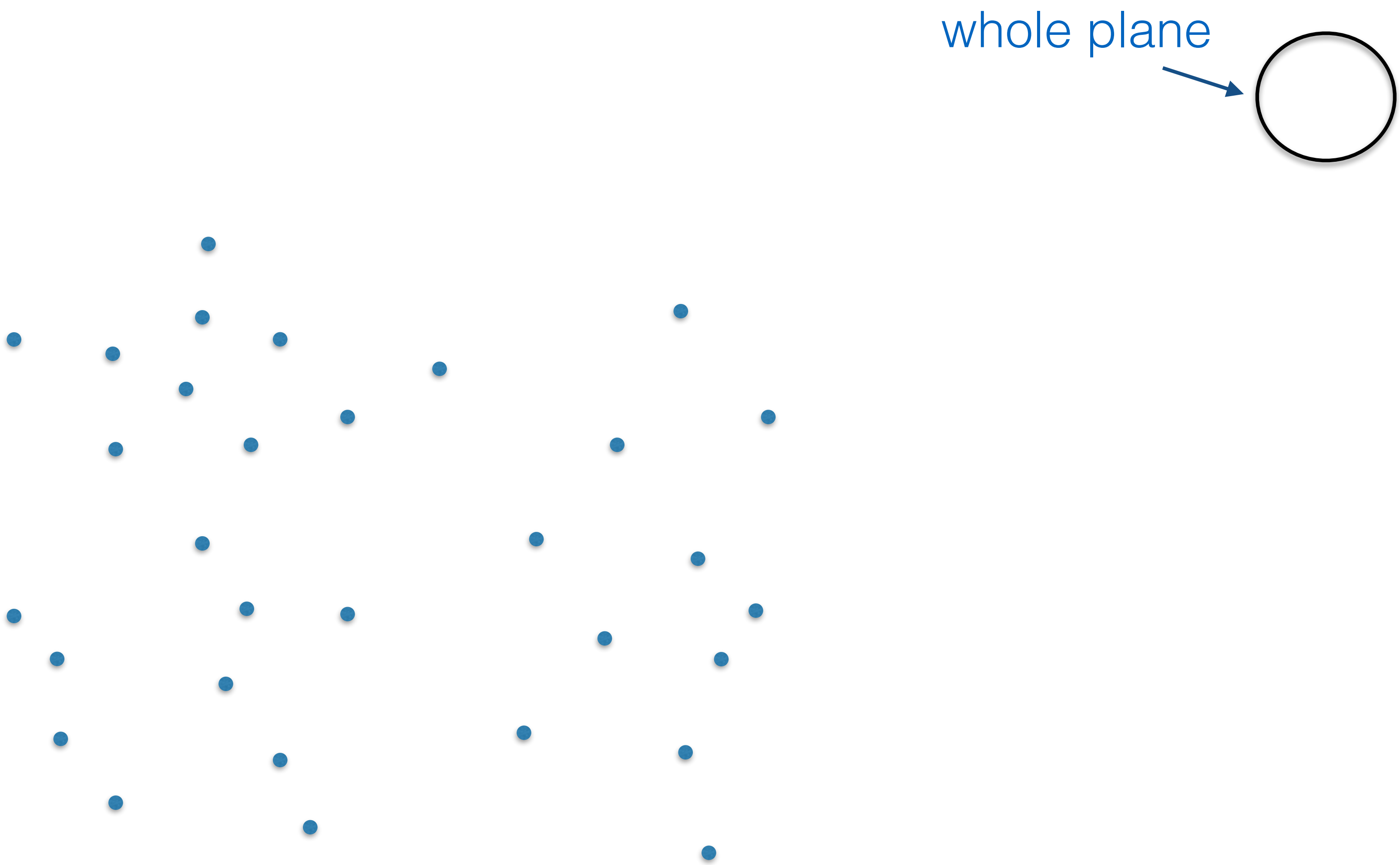
- Practical notes
  - The O(n) median finding algorithm is not practical. Either use a randomized median finding (QuickSelect); or, better,
  - Avoid needing to find a median by pre-sorting the points
    - sort P by x- coord and, separately by y-coord **only once** at the beginning, before building the tree, and pass them as parameters
    - *BuildKDtree (P-sorted-by-x, P-sorted-by-y, depth)*

↑ you'll work out the details in project 3 space partition corresponding to the 2d-tree

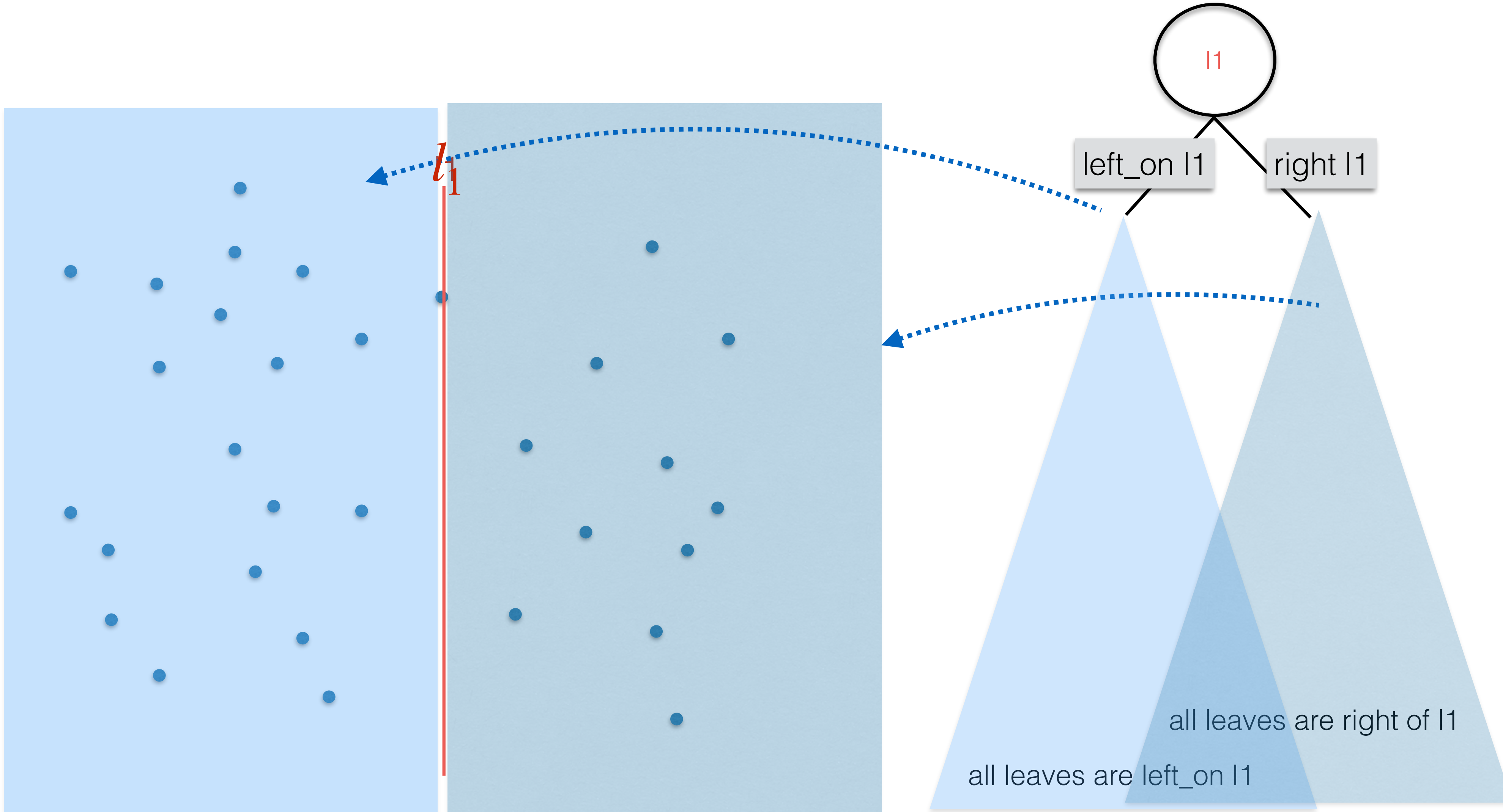


the corresponding 2d-tree



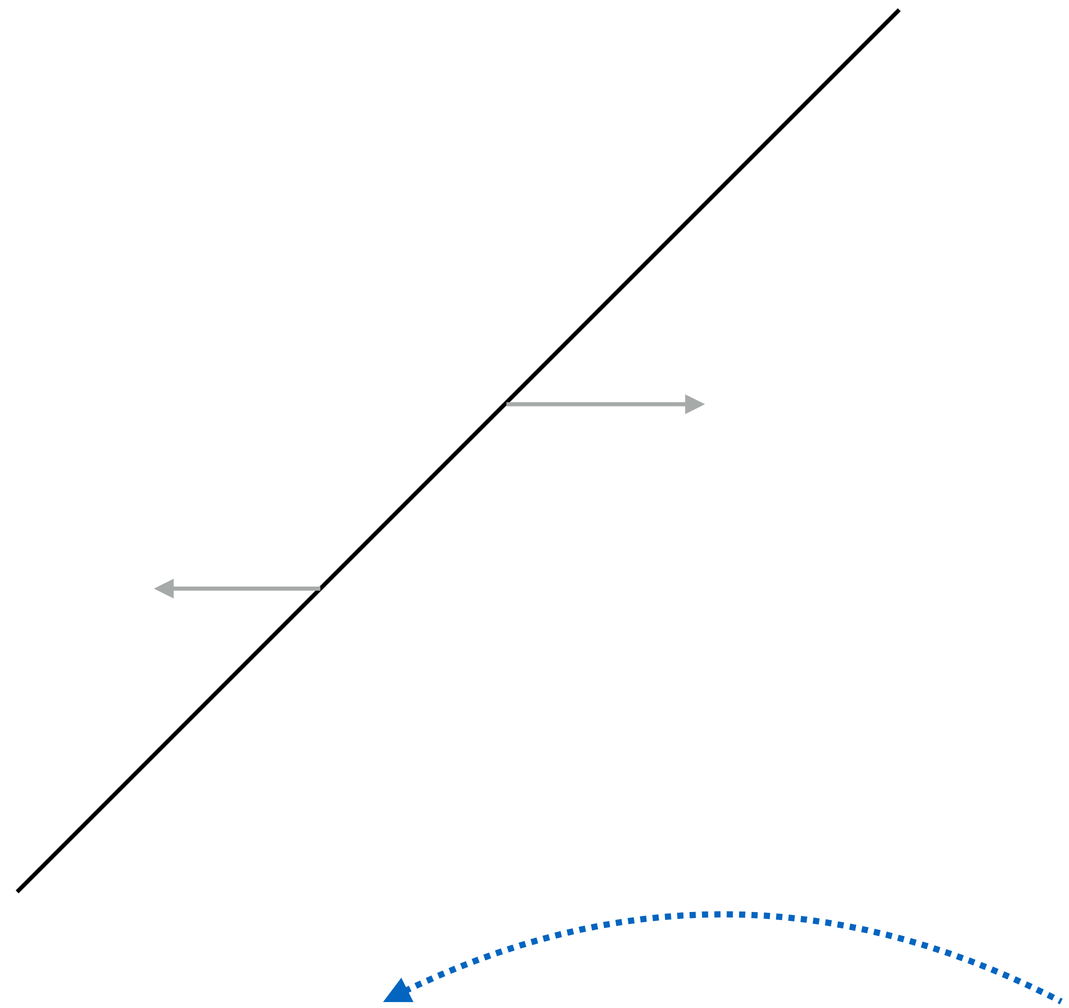


Each node in the tree corresponds to a region in the plane.

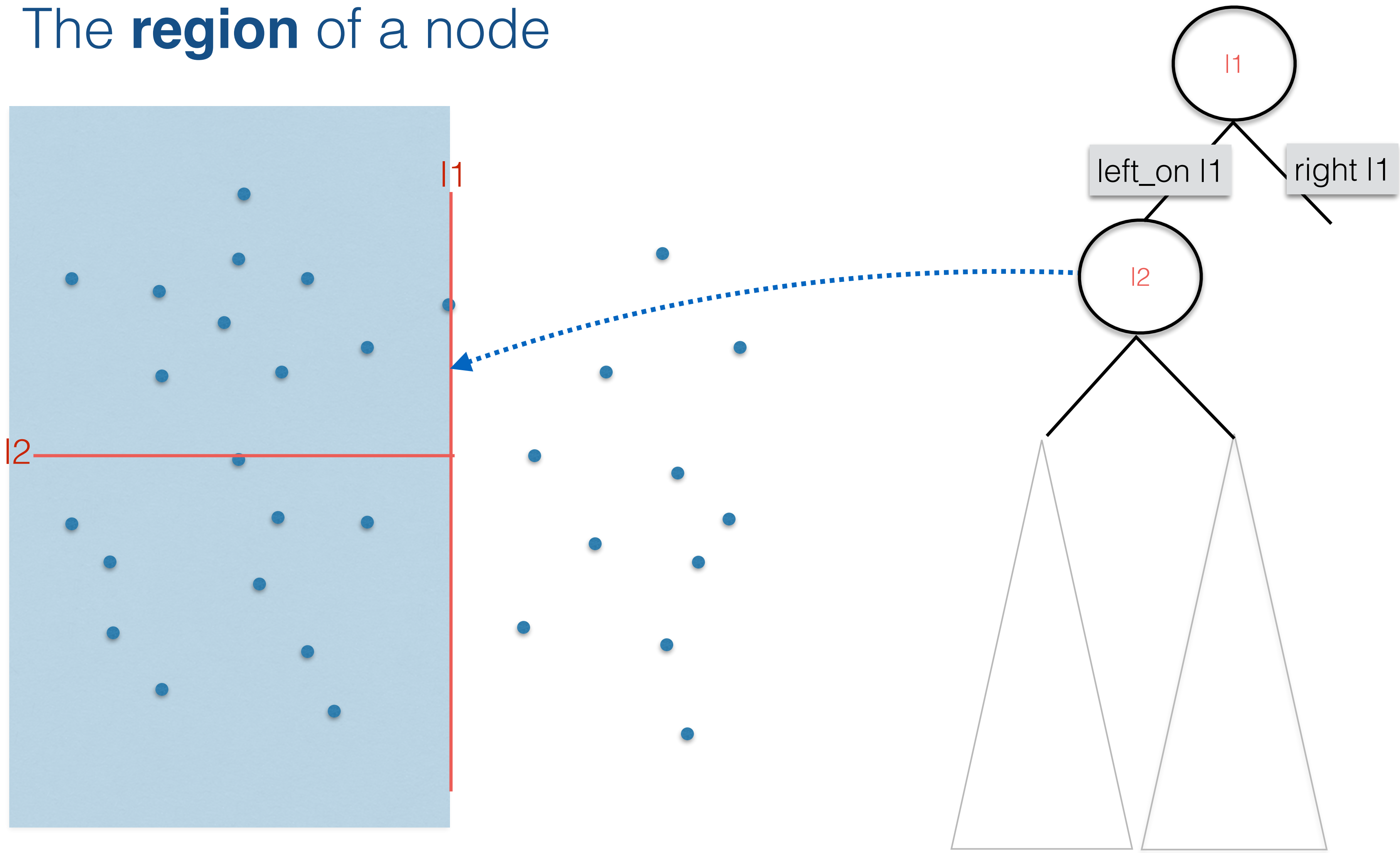


l_1
l1
left_on l1
right l1
all leaves are right of l1
all leaves are left_on l1

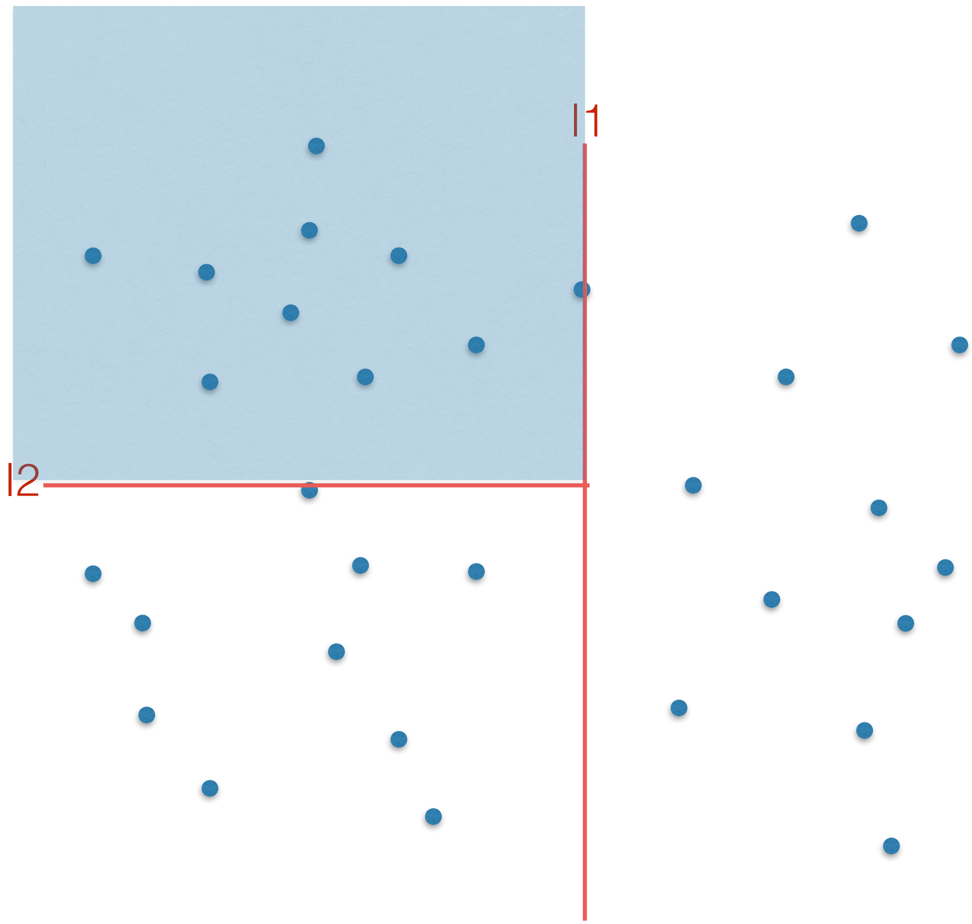
A line in the plane defines two **half-planes**

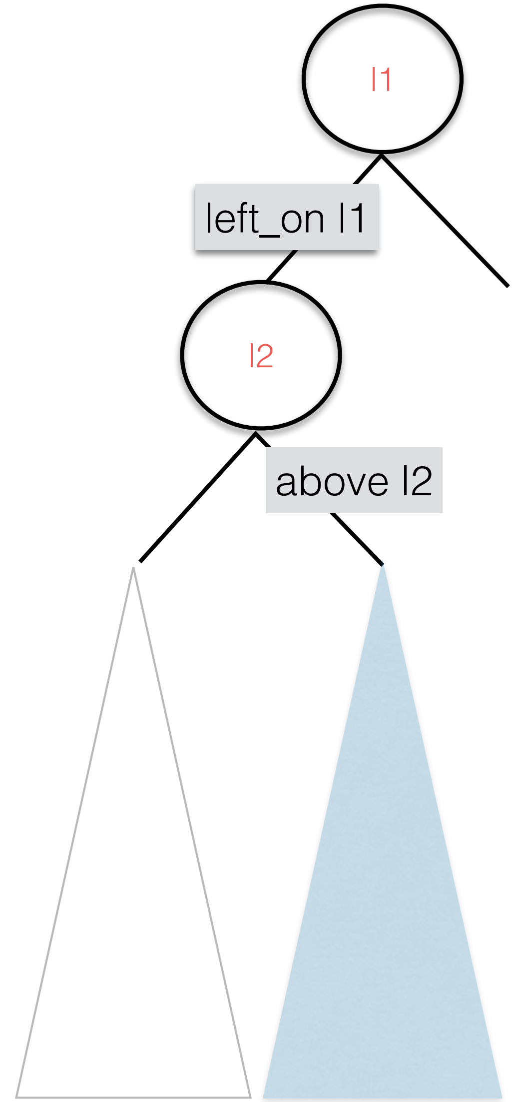


The **region** of a node

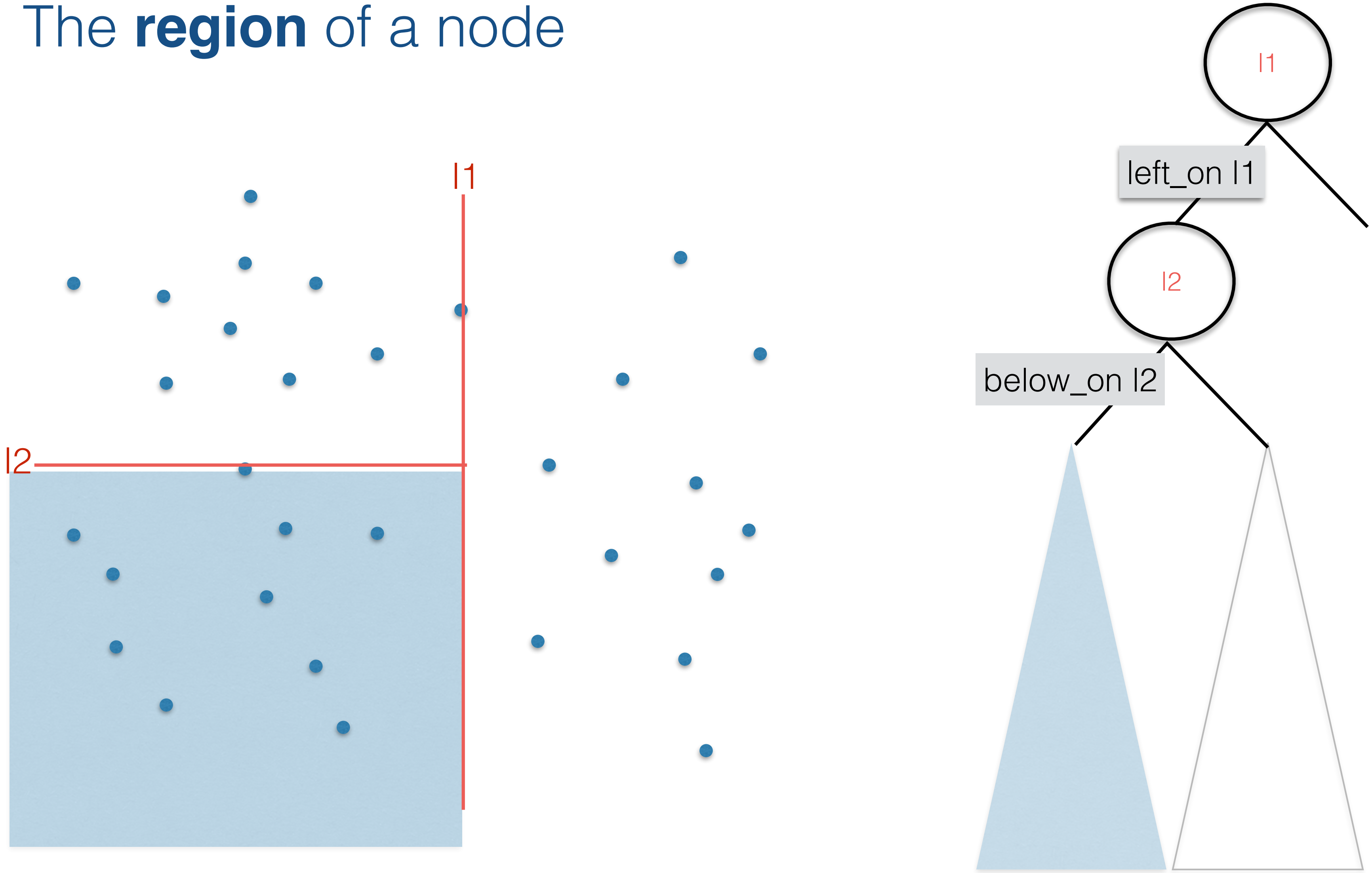


The **region** of a node

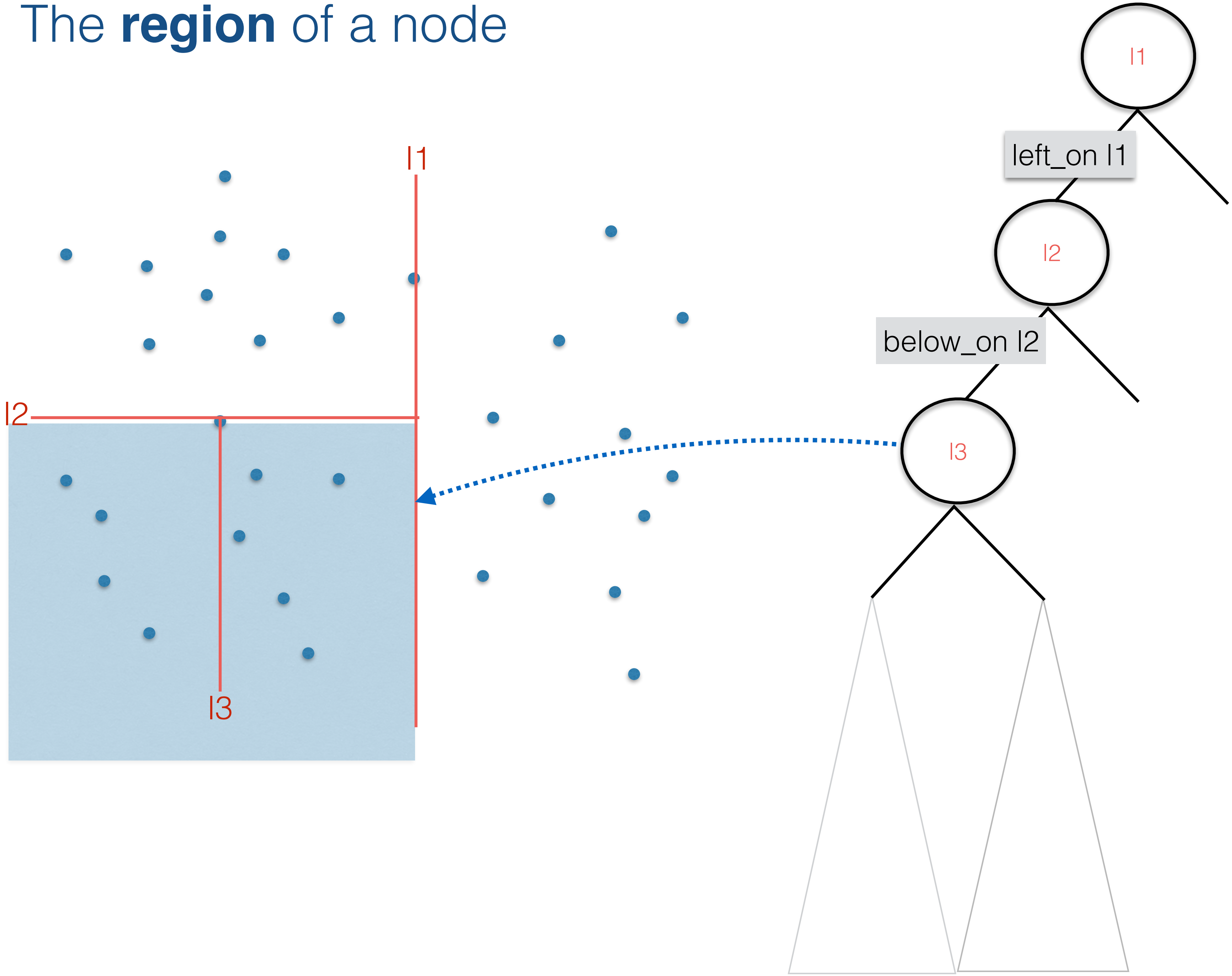




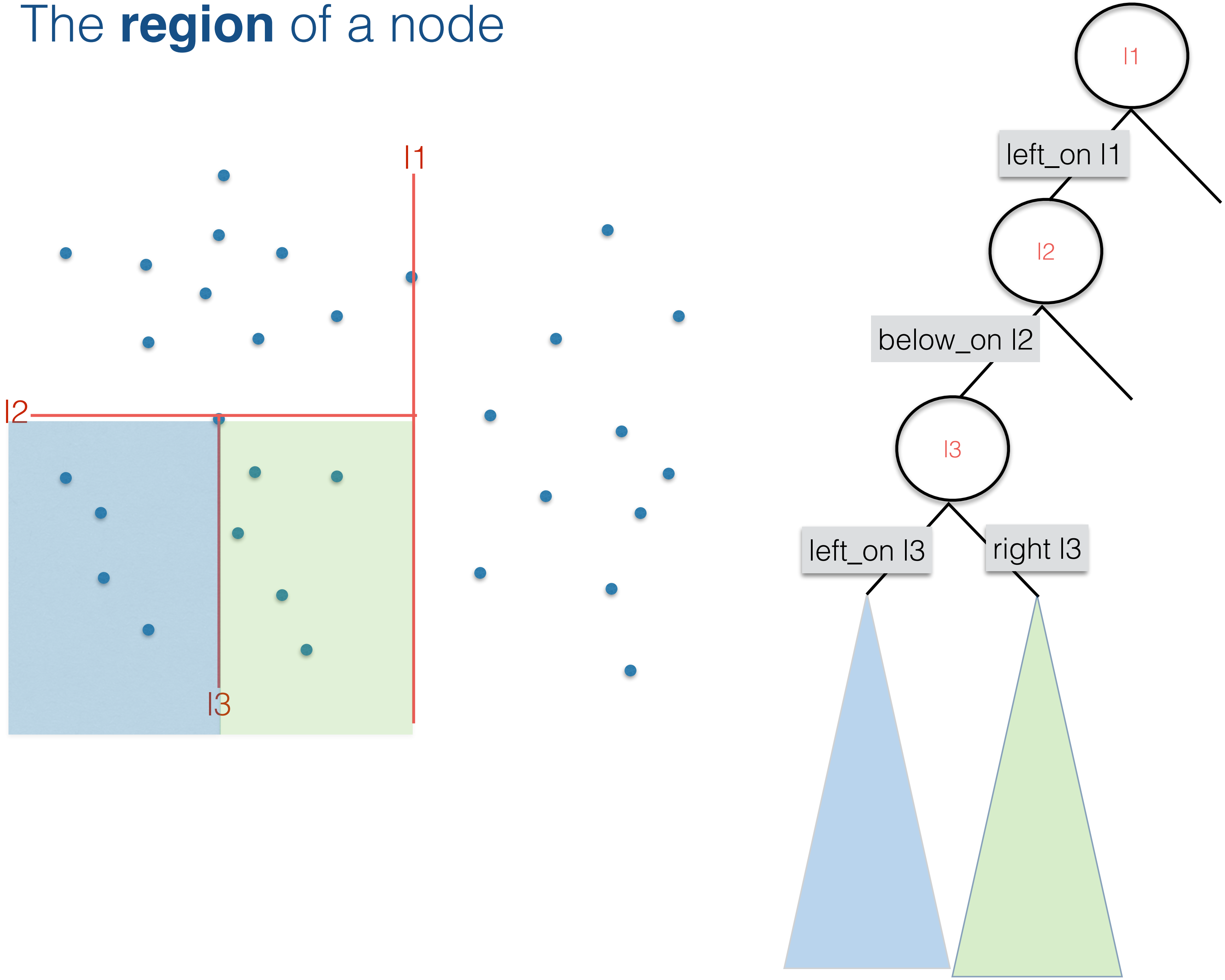
The **region** of a node



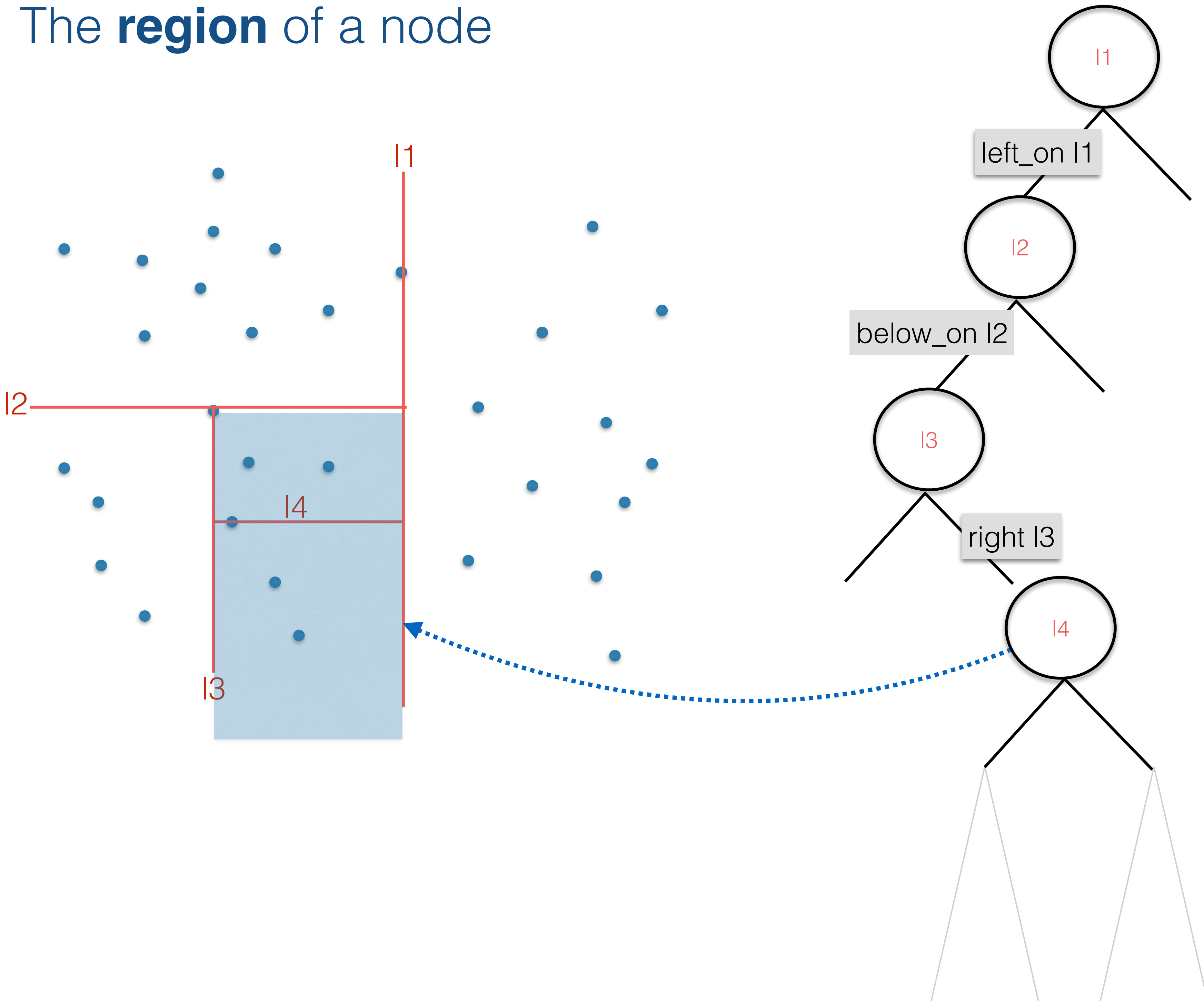
The **region** of a node



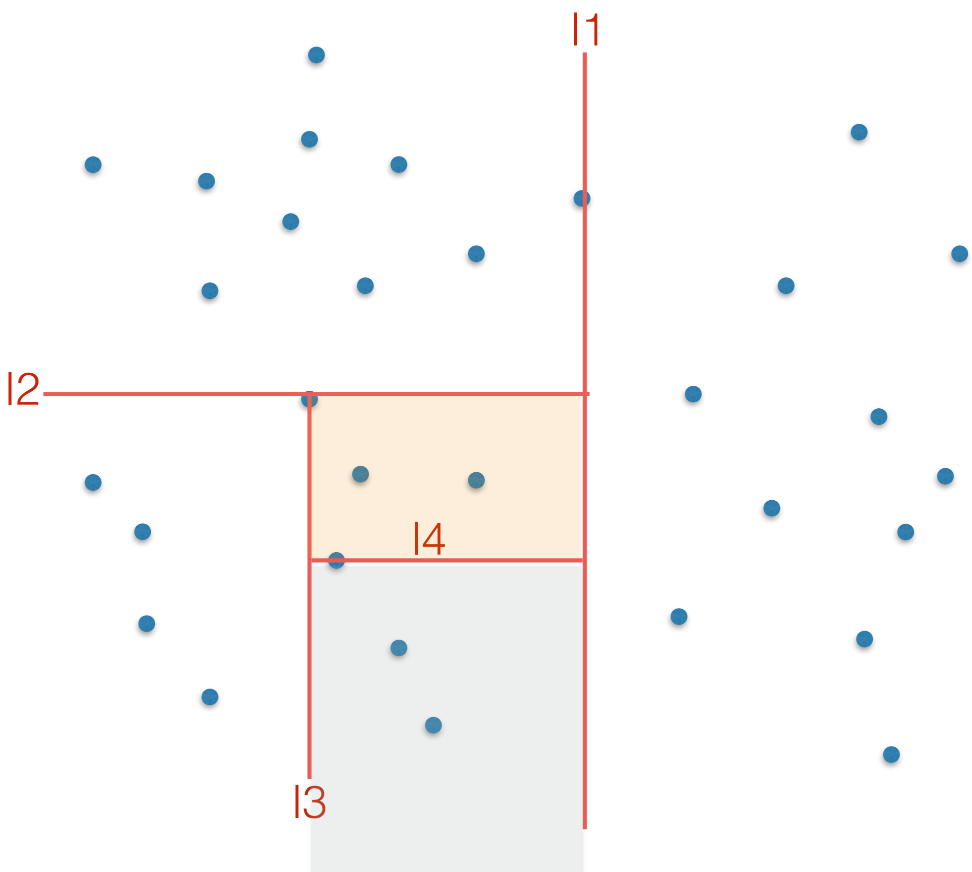
The **region** of a node

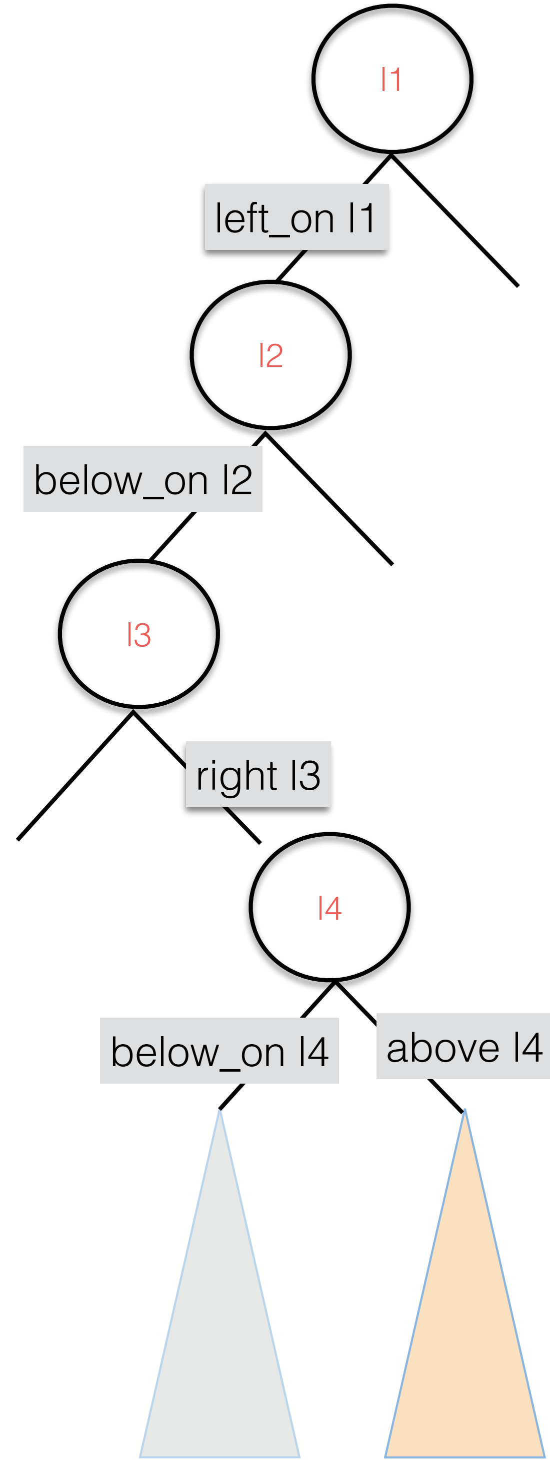


The **region** of a node

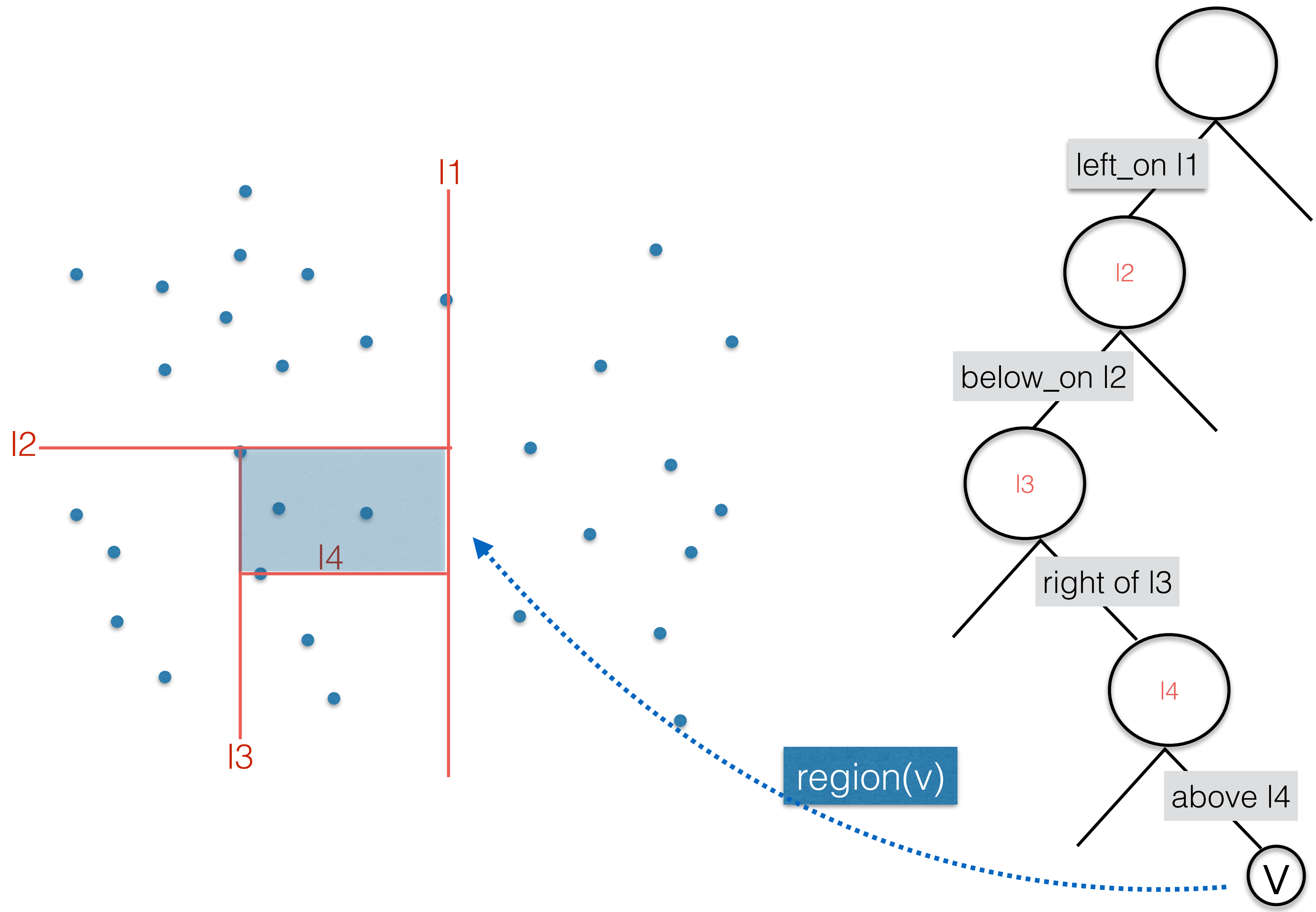


The **region** of a node

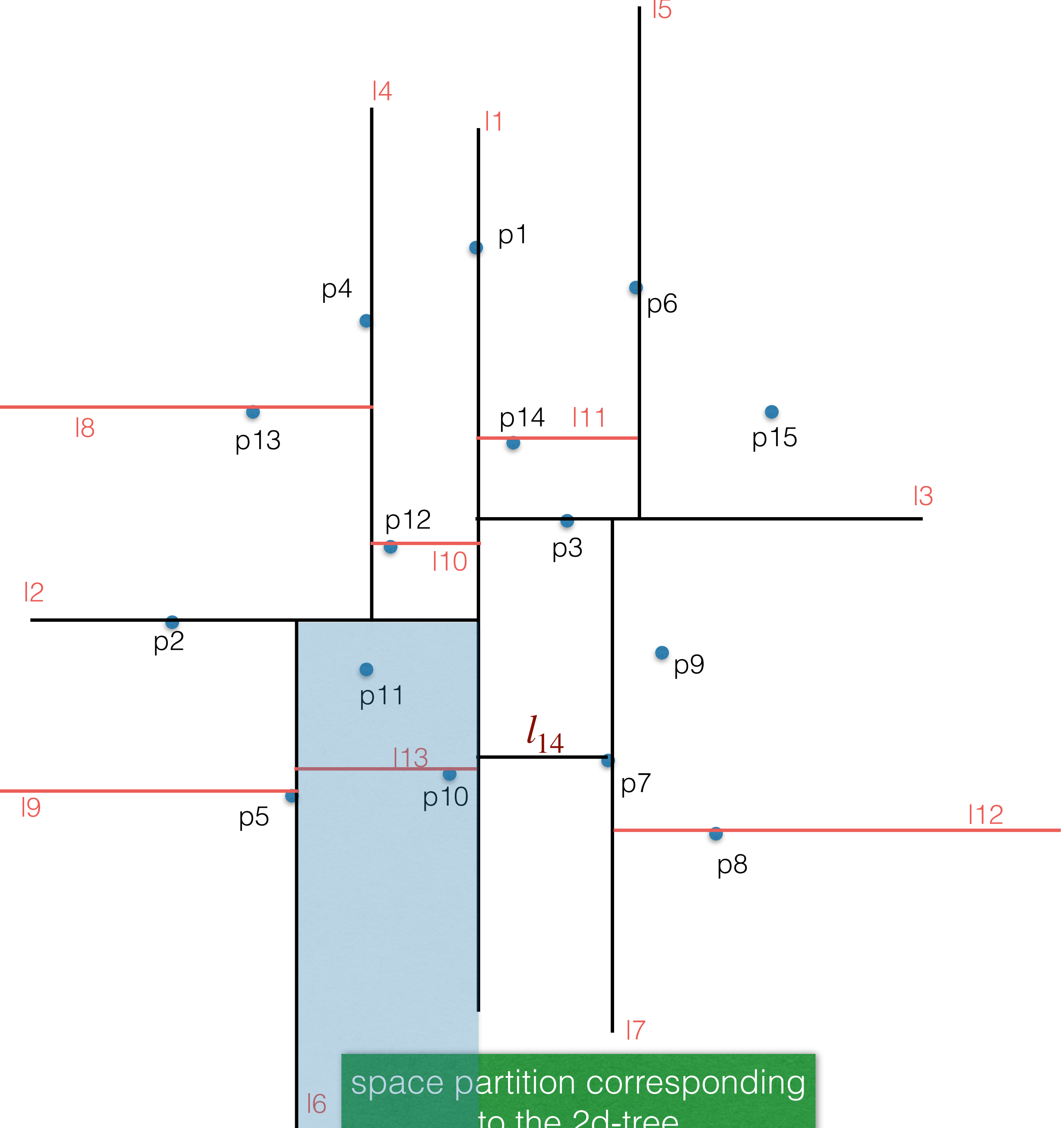




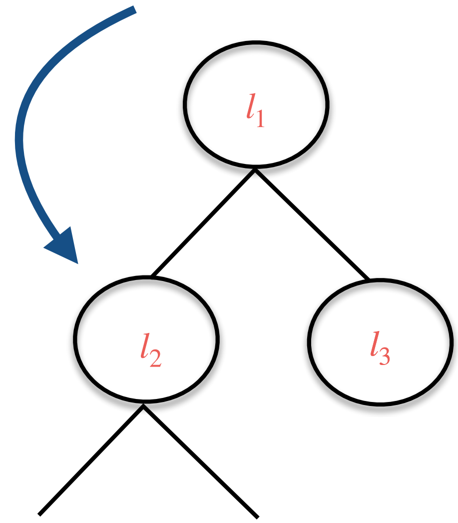
Each node in the tree corresponds to a region in the plane, which is the intersection of the half-planes of all it's ancestors.



Exercise: express the region corresponding to, e.g., second grandchild of this node

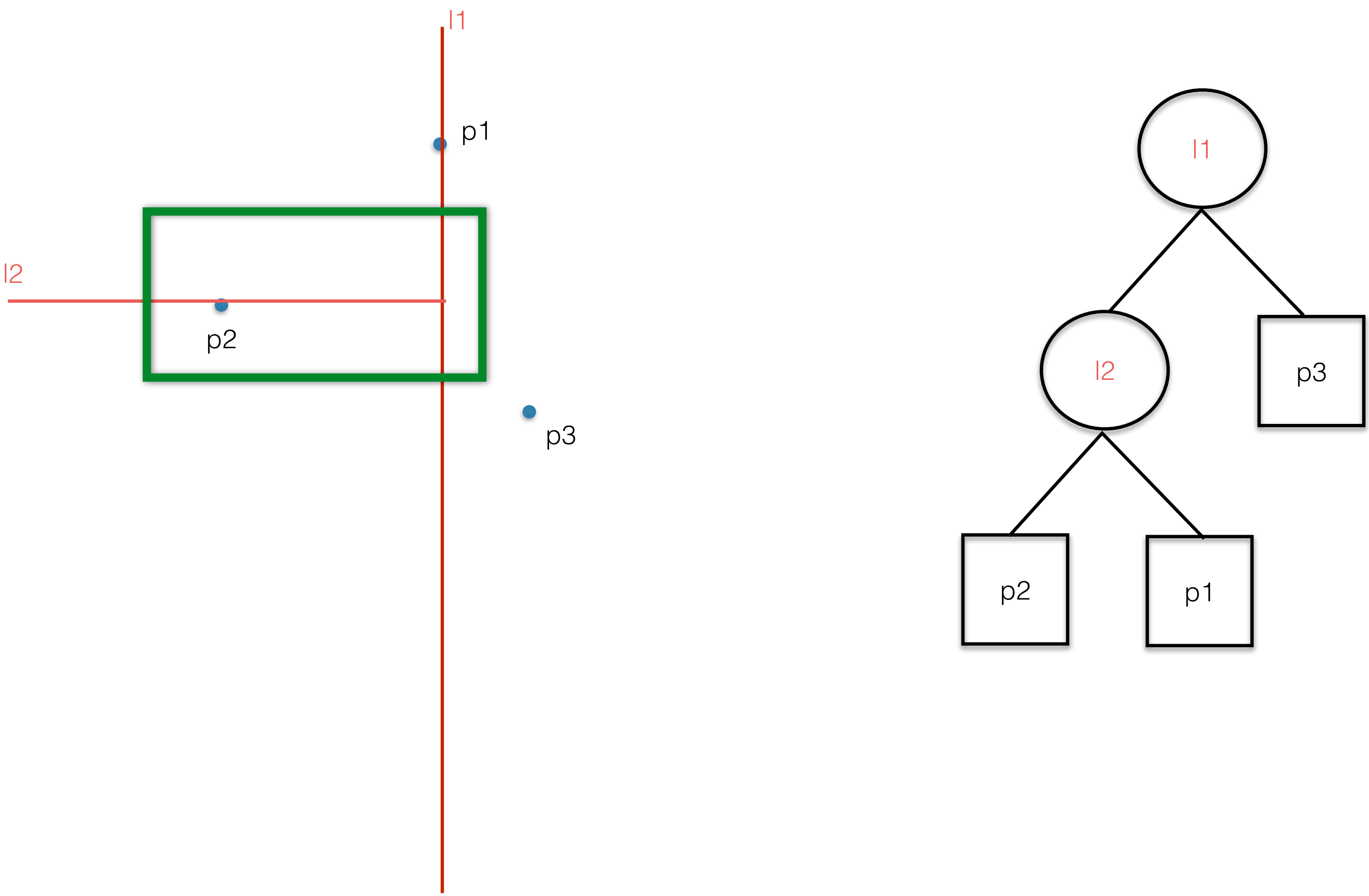


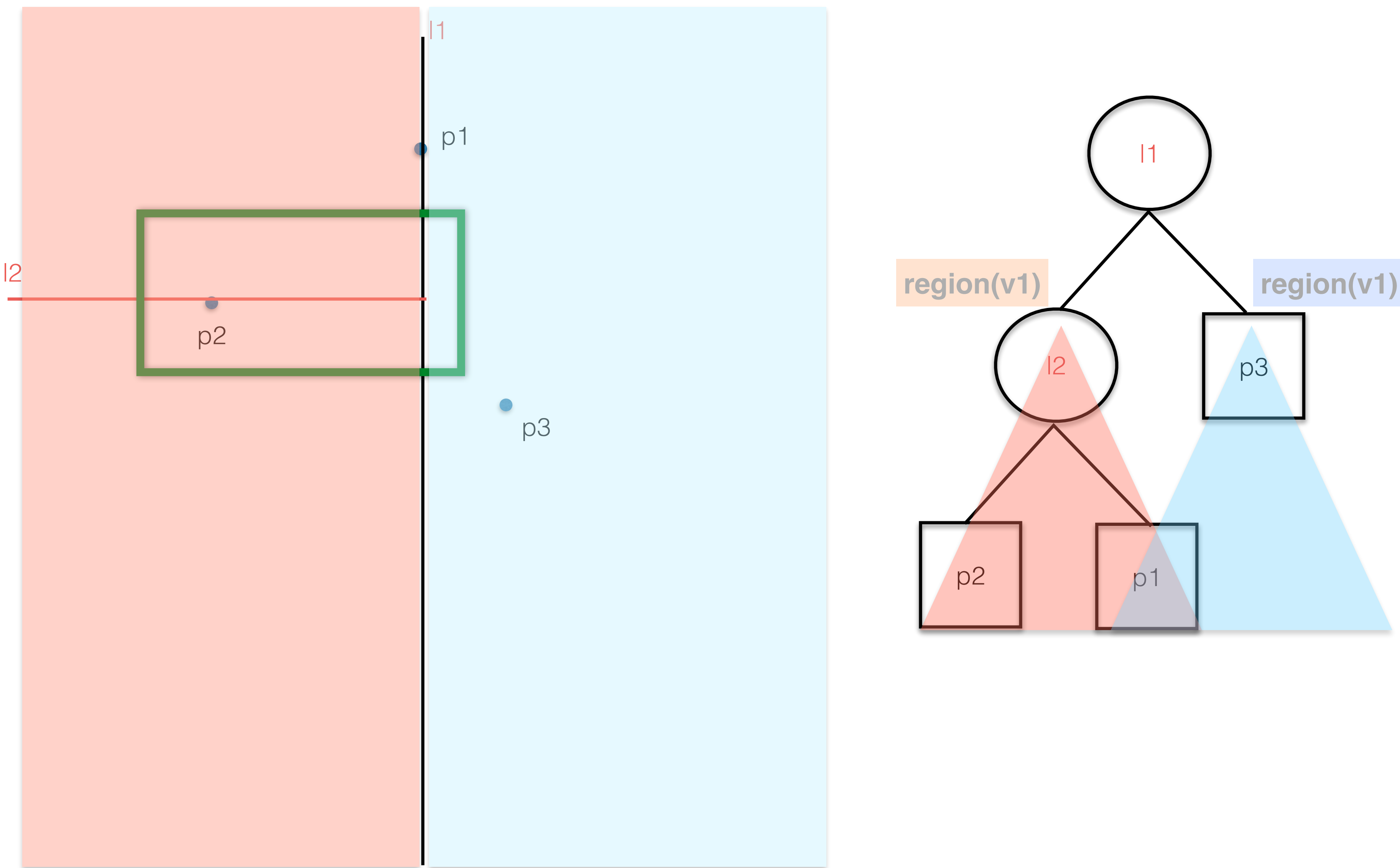
space partition corresponding to the 2d-tree



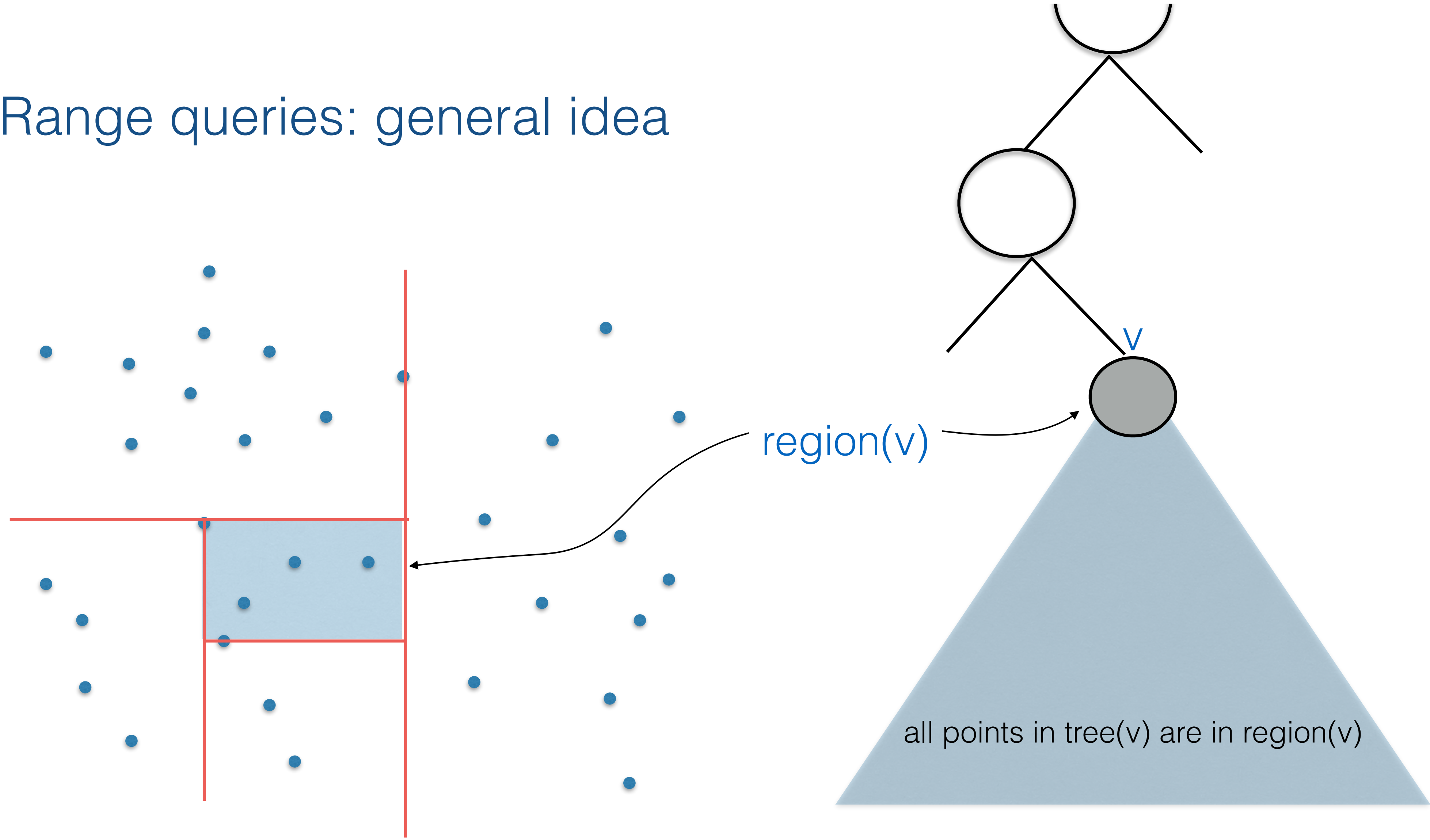
the corresponding 2d-tree

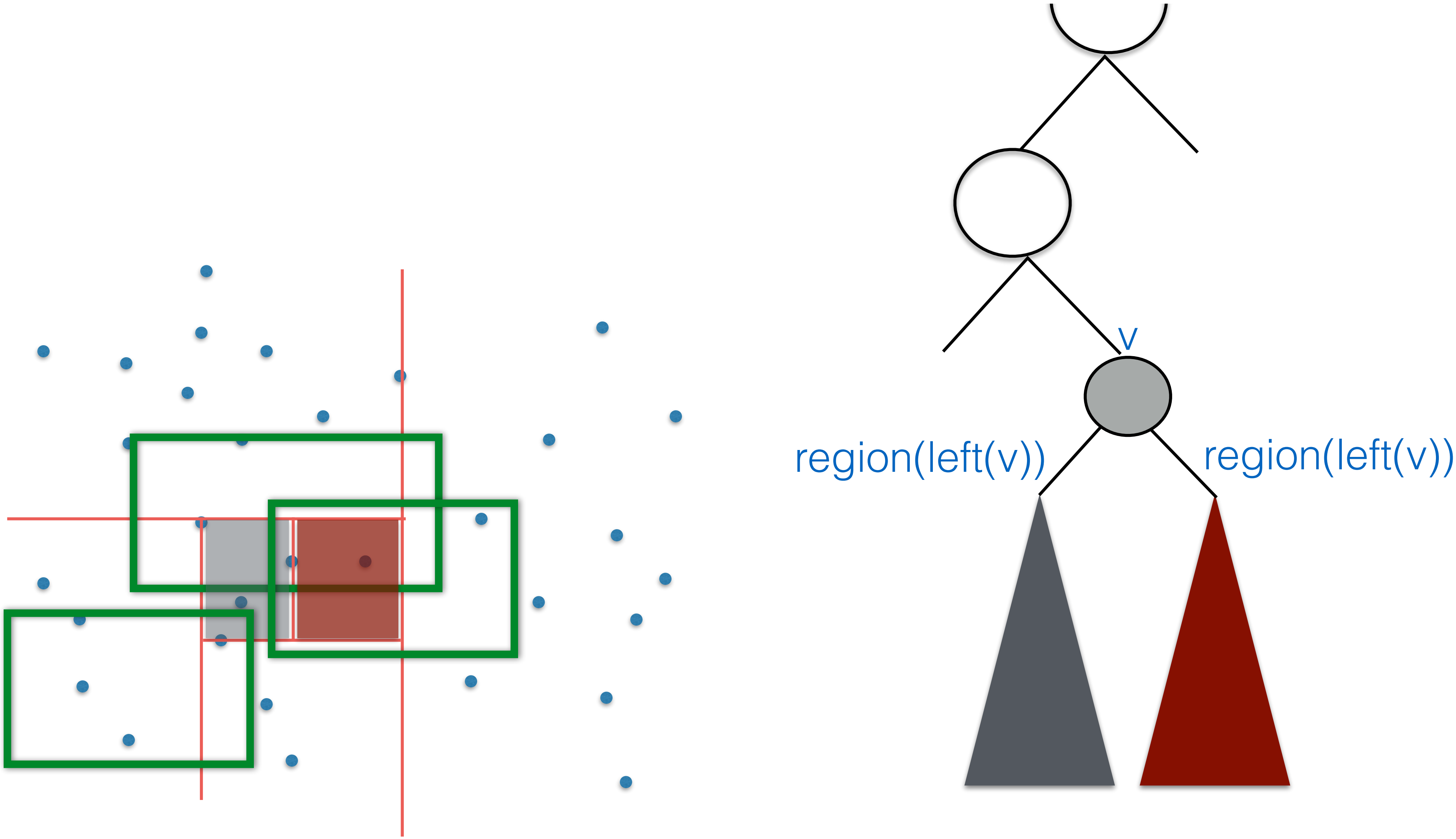
How do we answer range queries on kd-trees ?




l1
p1
l2
p2
p3
l1
region(v1)
region(v1)
l2
p3
p2
p1

Range queries: general idea





Case 1:range intersects both children

Case 2: range intersects only one child

Case 3: child completely contained in range

Algorithm SEARCHKDTREE(ν, R)

Input. The root of (a subtree of) a kd-tree, and a range R

Output. All points at leaves below ν that lie in the range.

```
if ν is a leaf
    then Report the point stored at ν if it lies in R
    else if region(lc(ν)) is fully contained in R
            then REPORTSUBTREE(lc(ν))
            else if region(lc(ν)) intersects R
                    then SEARCHKDTREE(lc(ν), R)
         if region(rc(ν)) is fully contained in R
            then REPORTSUBTREE(rc(ν))
            else if region(rc(ν)) intersects R
                    then SEARCHKDTREE(rc(ν), R)
```

screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

Analysis:

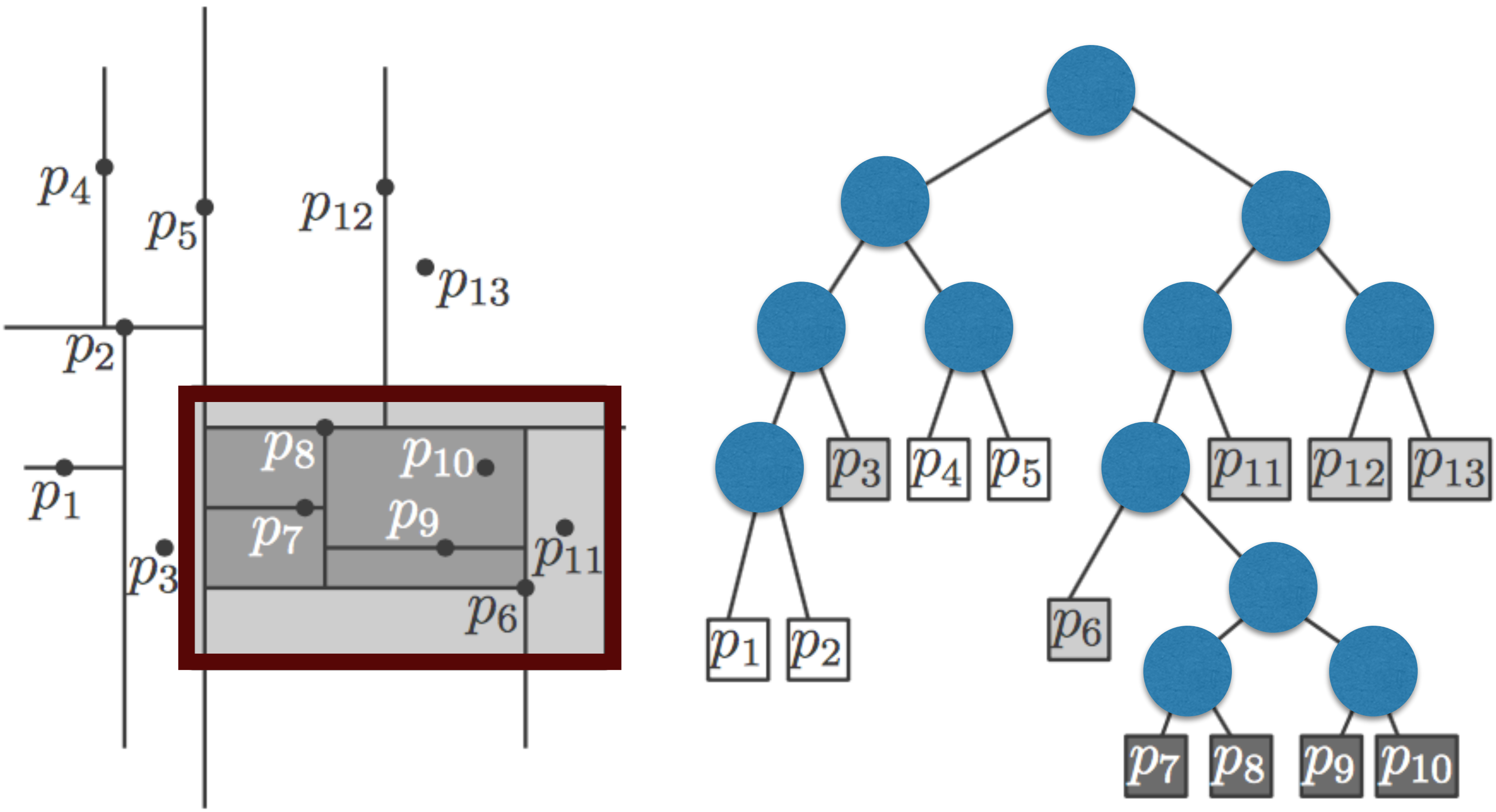
What’s the running time of a range search with a kd-tree?

The running time of a range-search with a kd-tree

- For each node we can visit and recurse on **one** or **both** of its children
- The nb. of points in a child is half the nb. of points in the parent
- If at every node v we only recurse on one child =>
  $T(n) = T(n/2) + O(1)$ which solves to $O(\lg n)$
- If at every node v we recurse on both children:
  $T(n) = 2T(n/2) + O(1)$ which solves to $O(n)$
- Here we visit the children intersected by the range, which can be one or both

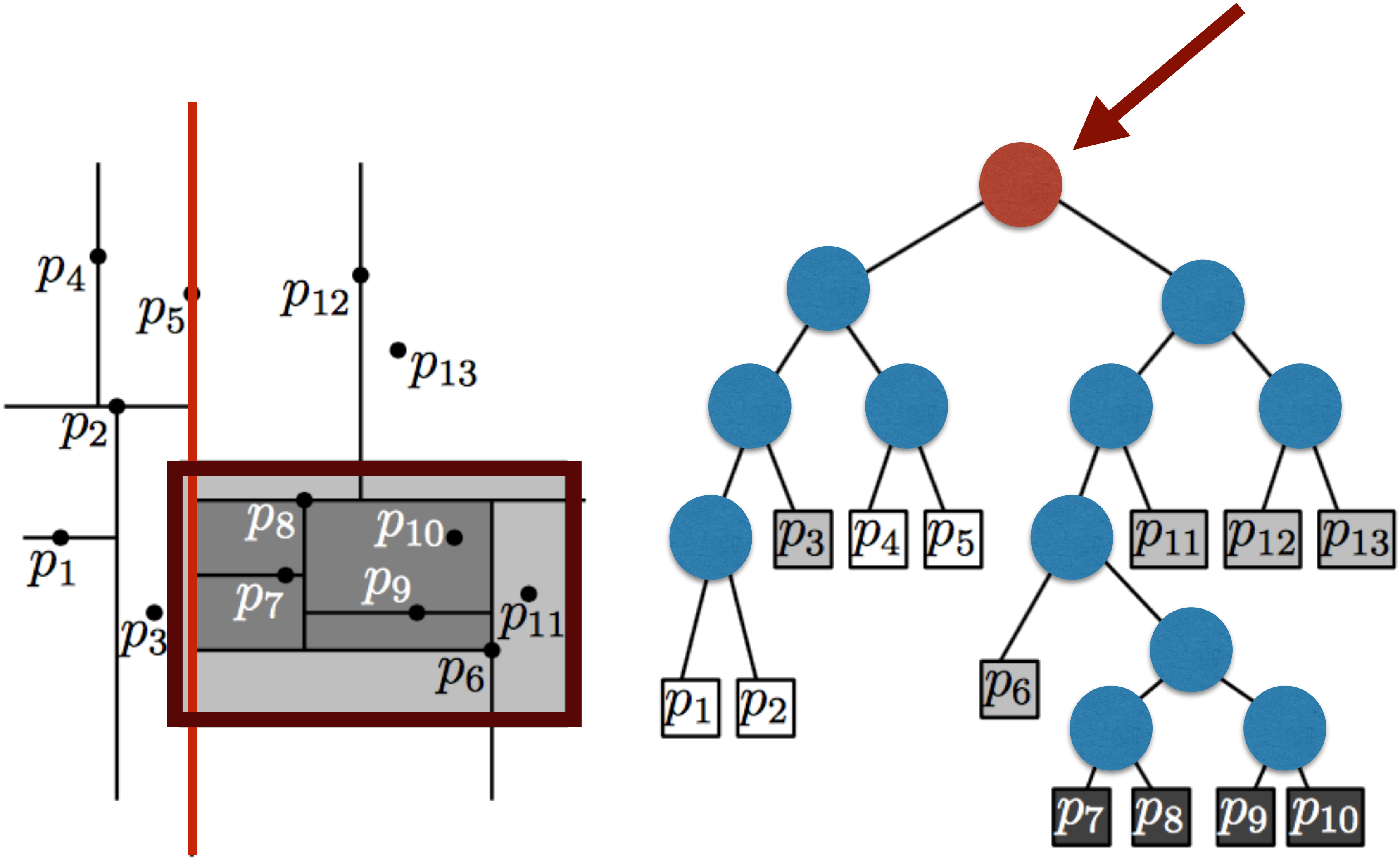
- So what's the running time of a search? $O(\lg n)$? or $O(n)$?

To analyze the time to answer a range query we’ll look at the nodes visited in the tree



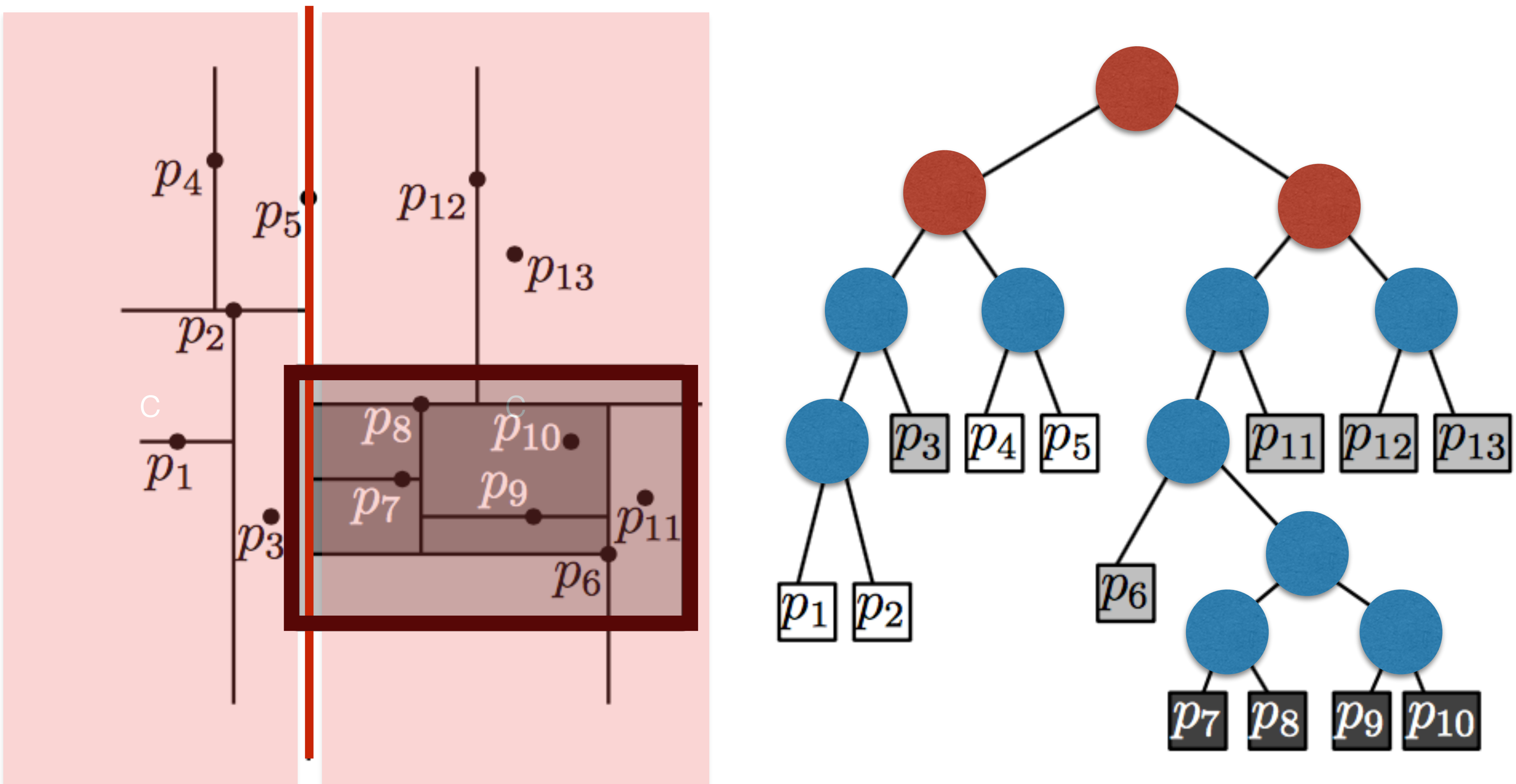
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



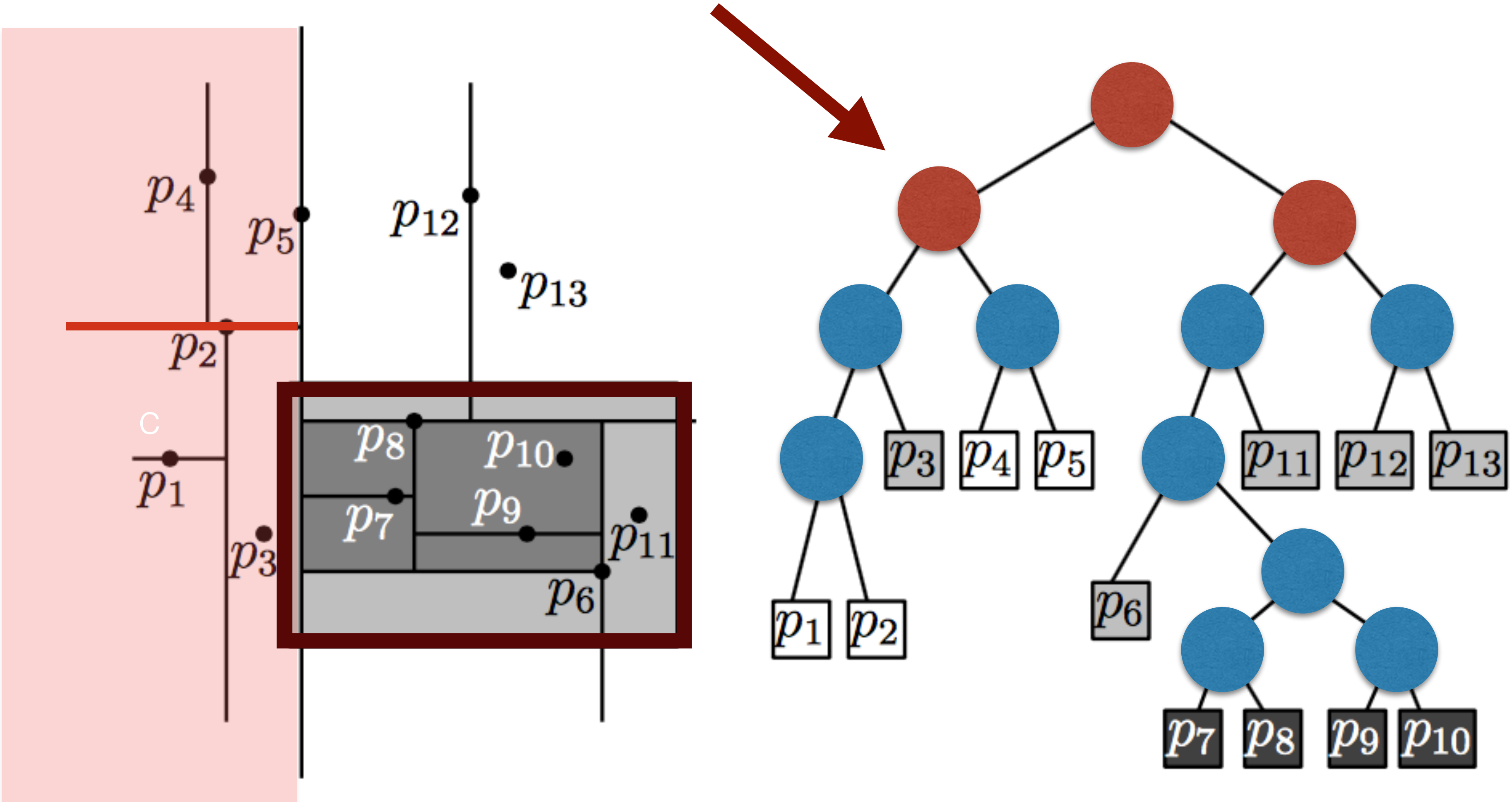
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



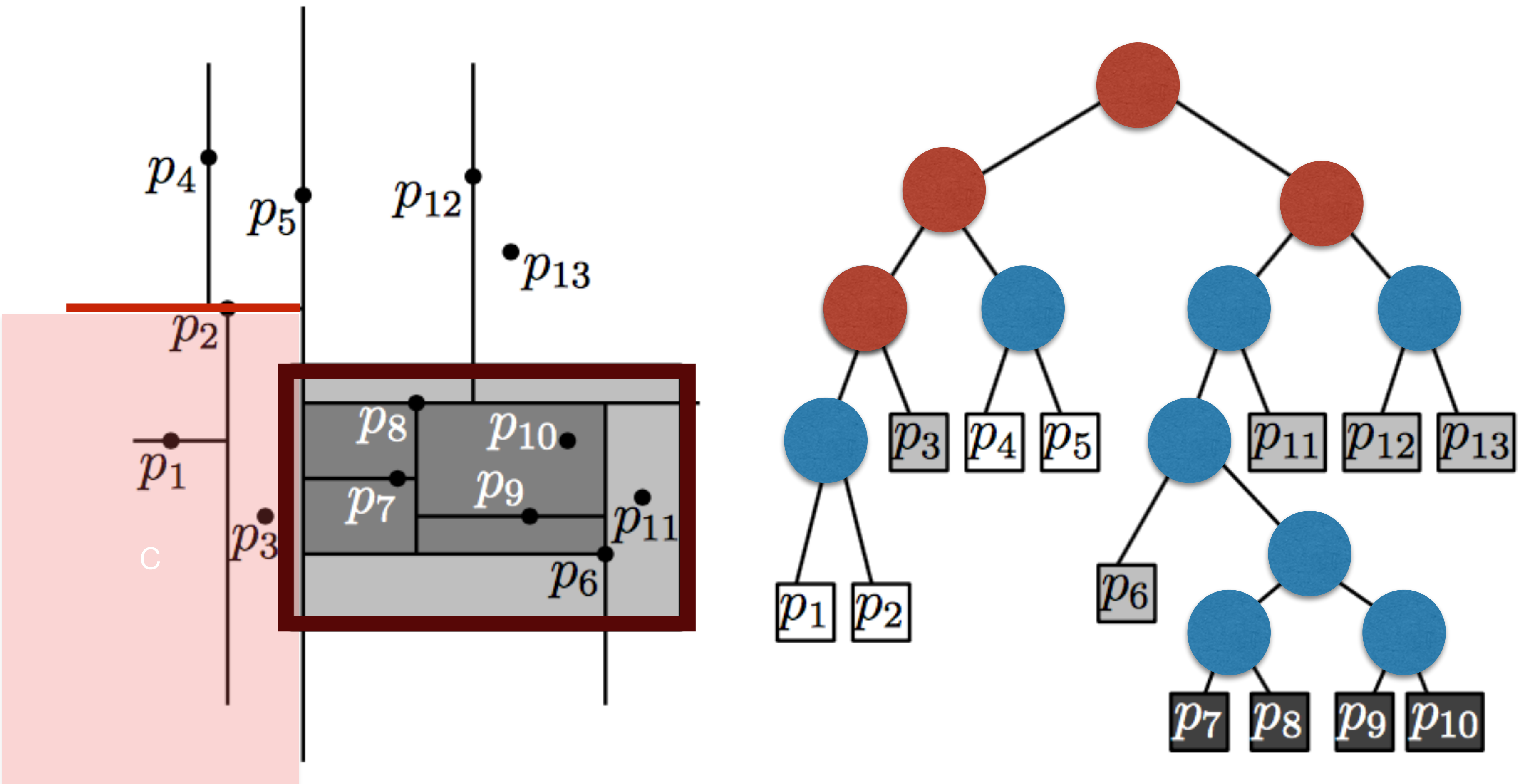
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



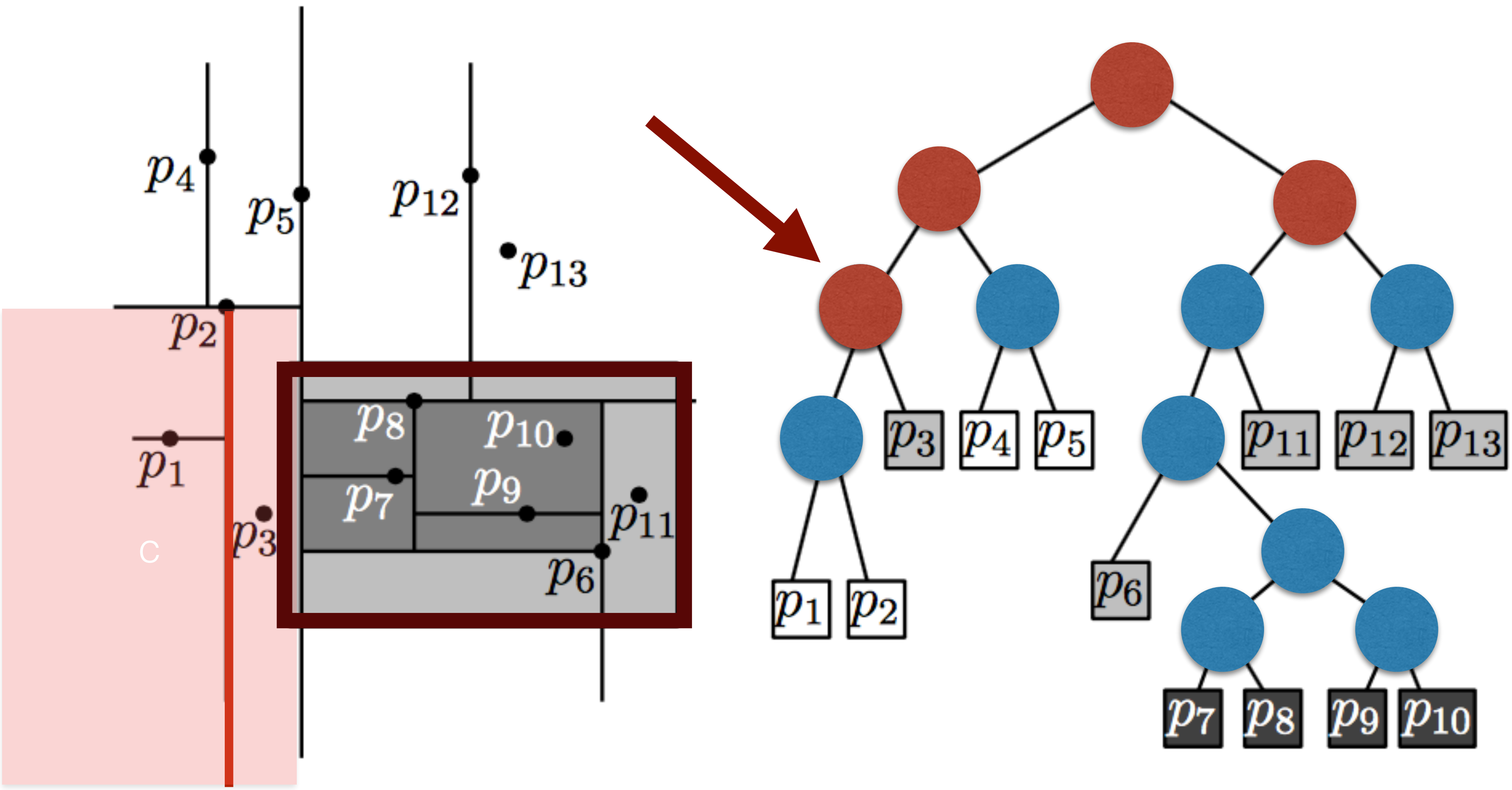
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



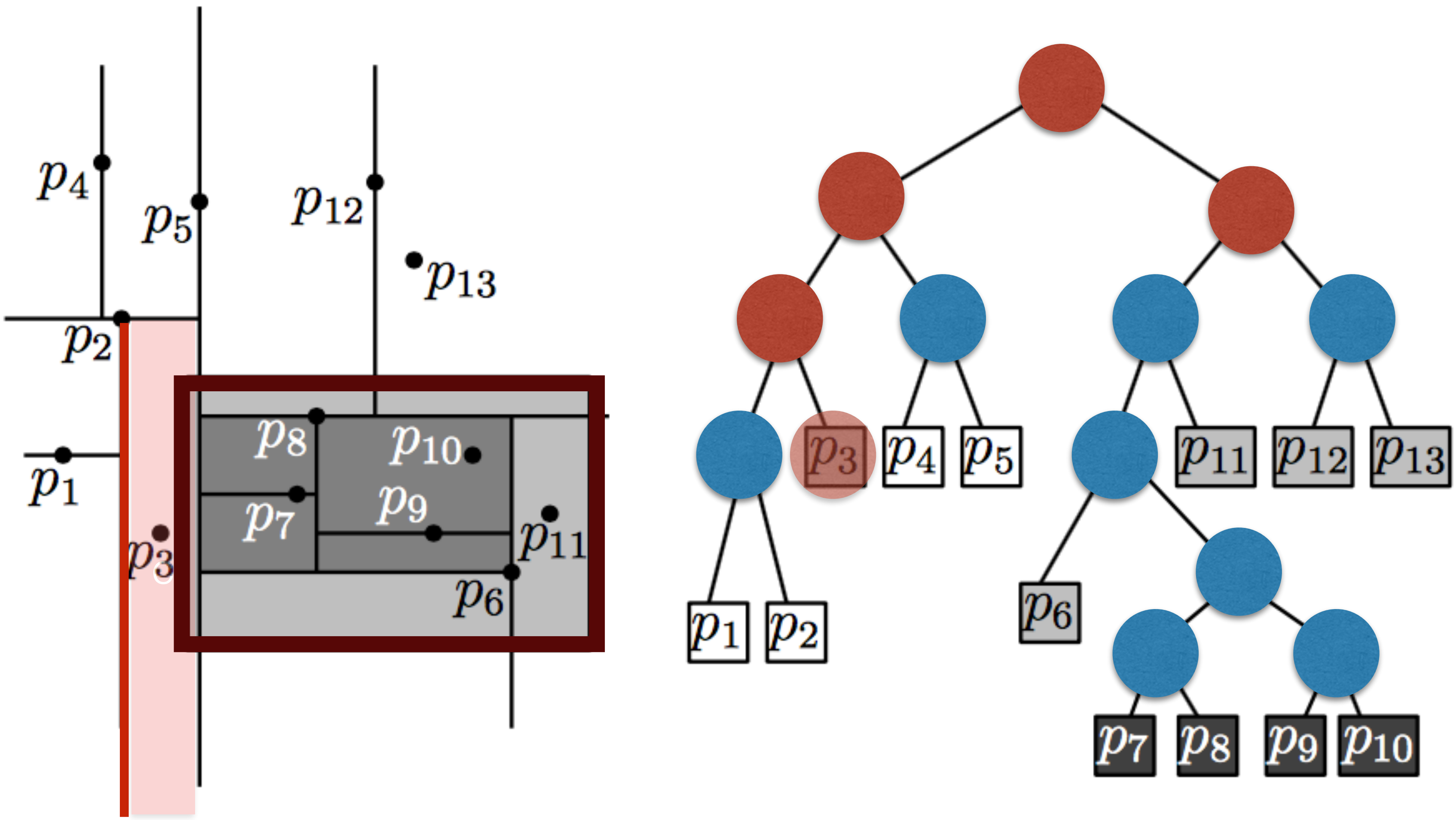
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



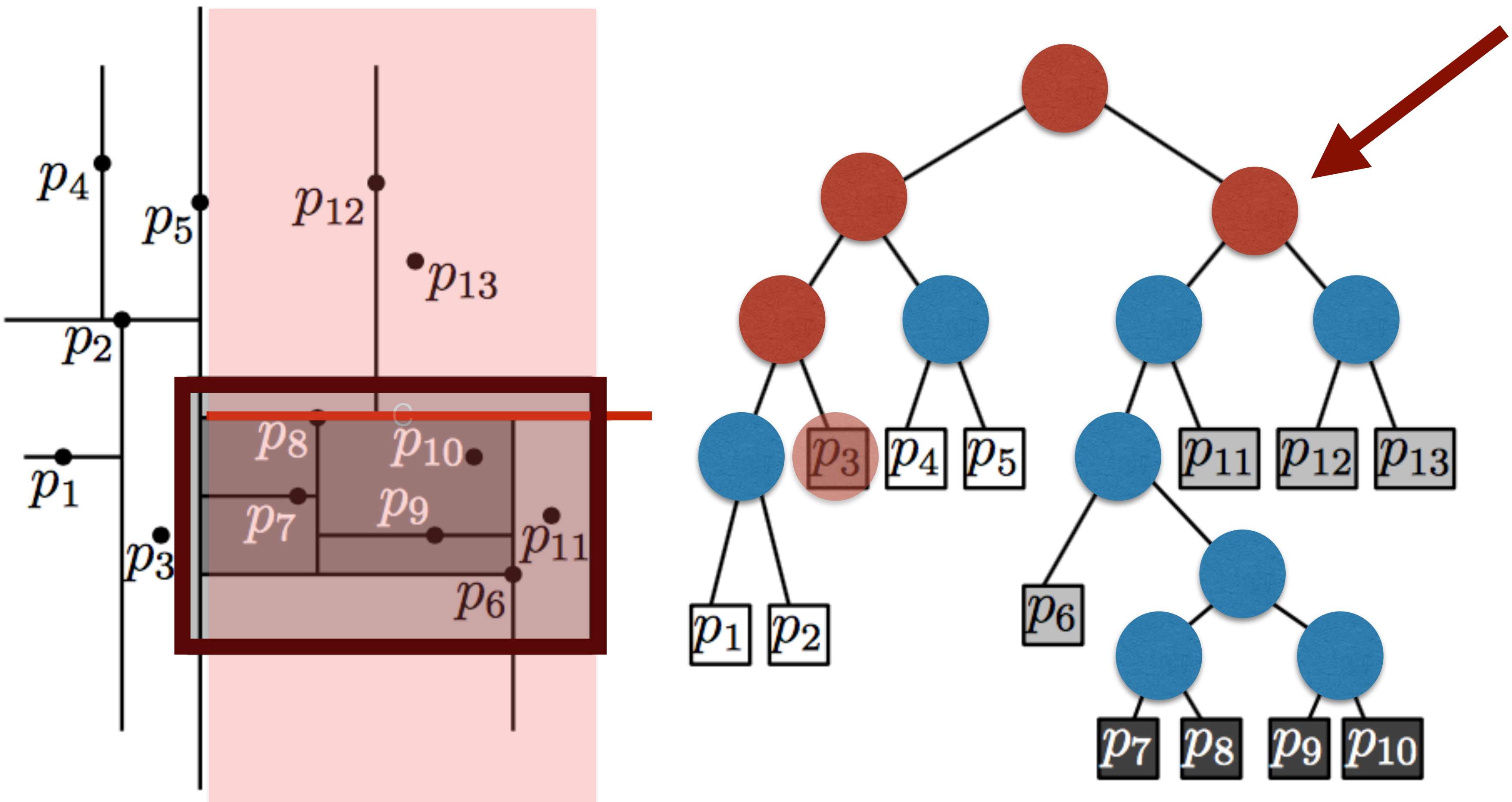
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



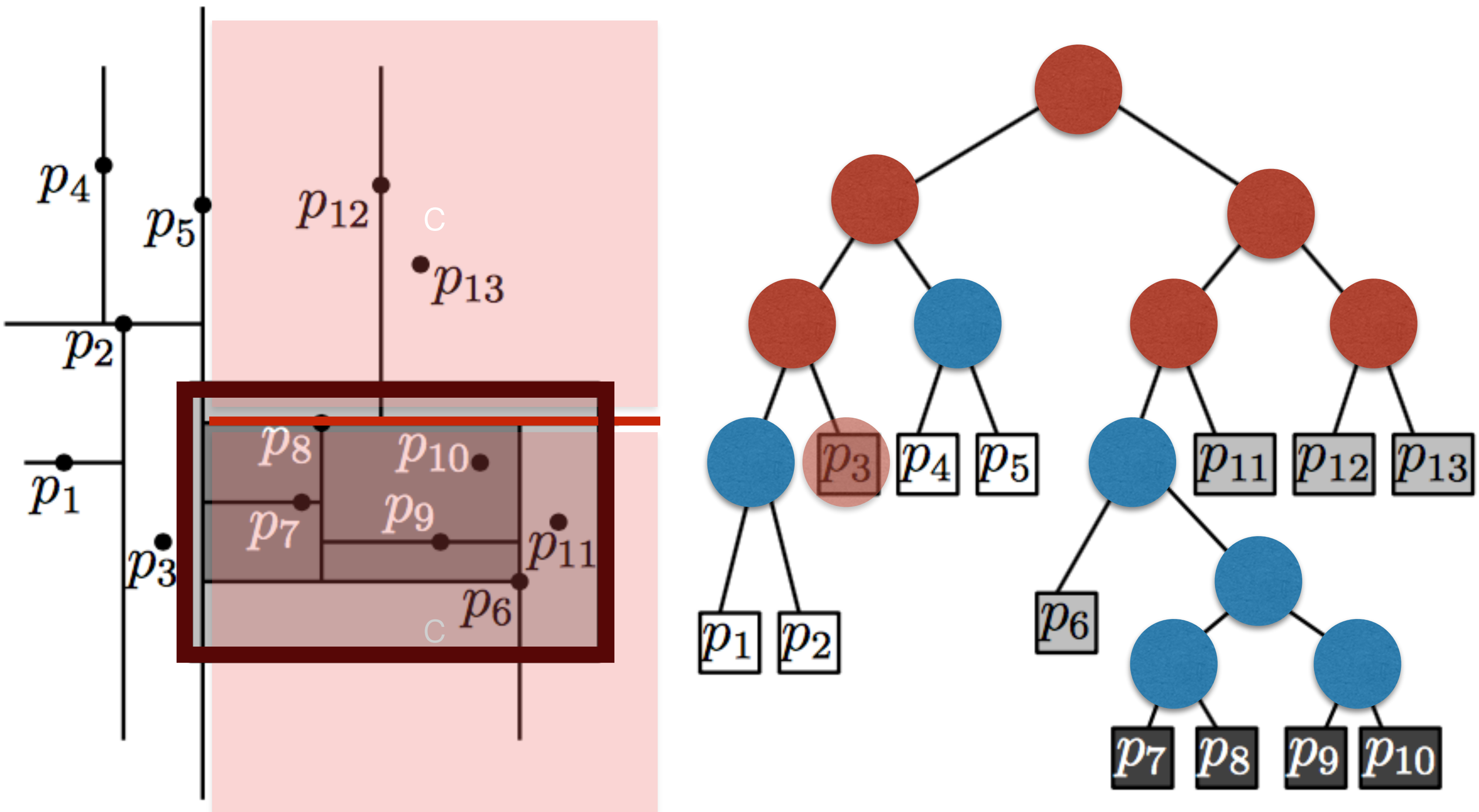
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



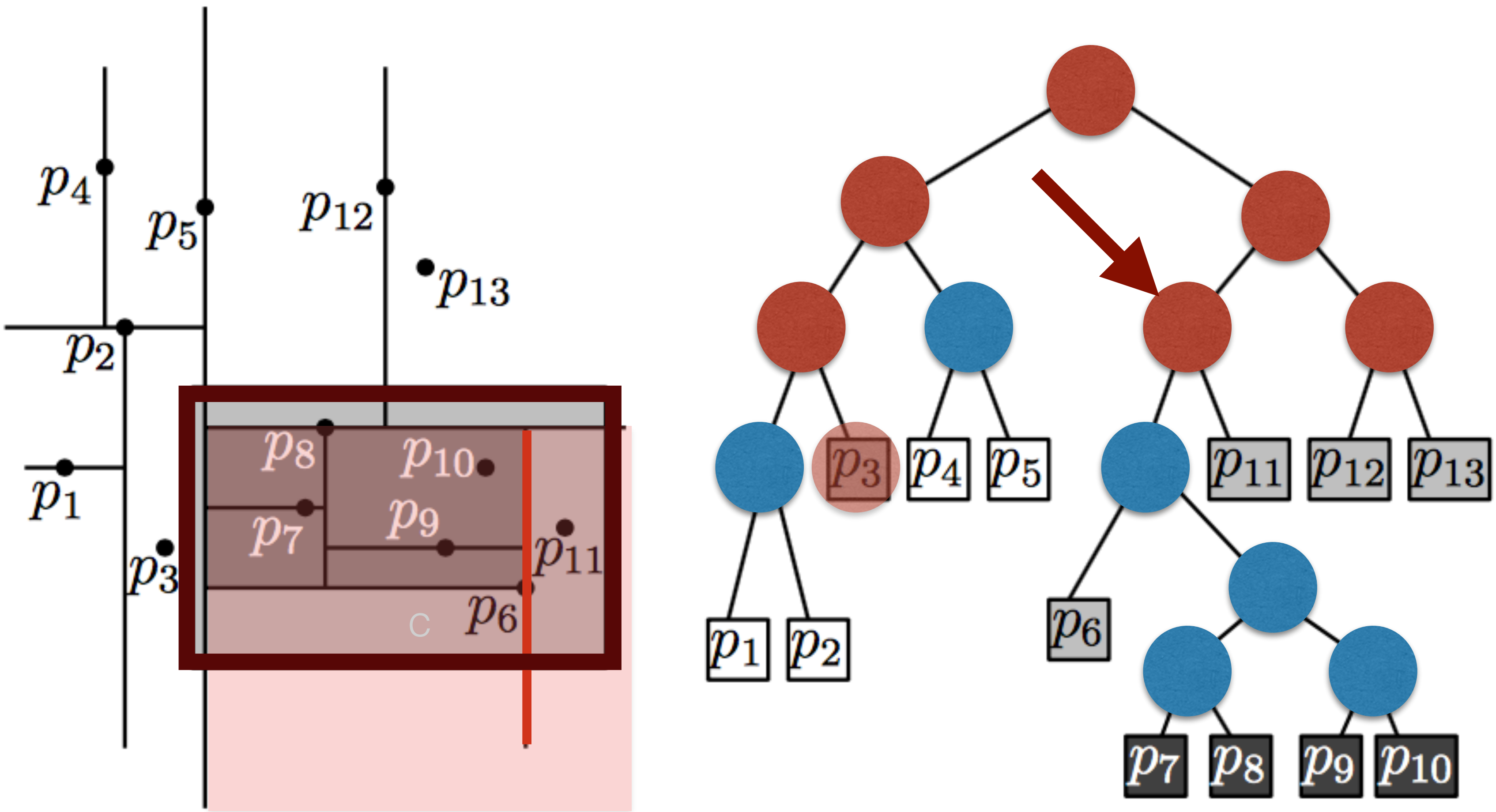
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



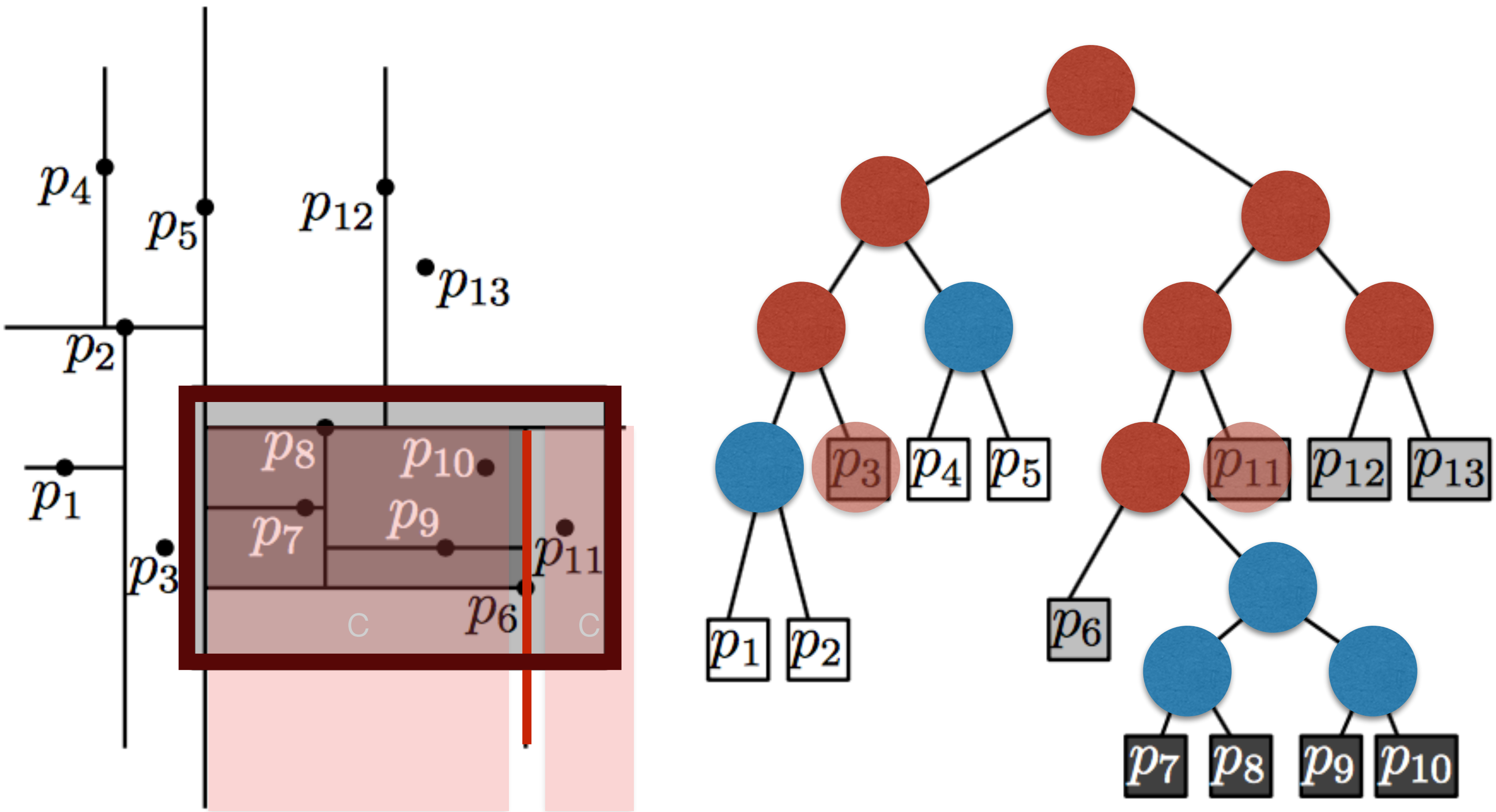
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



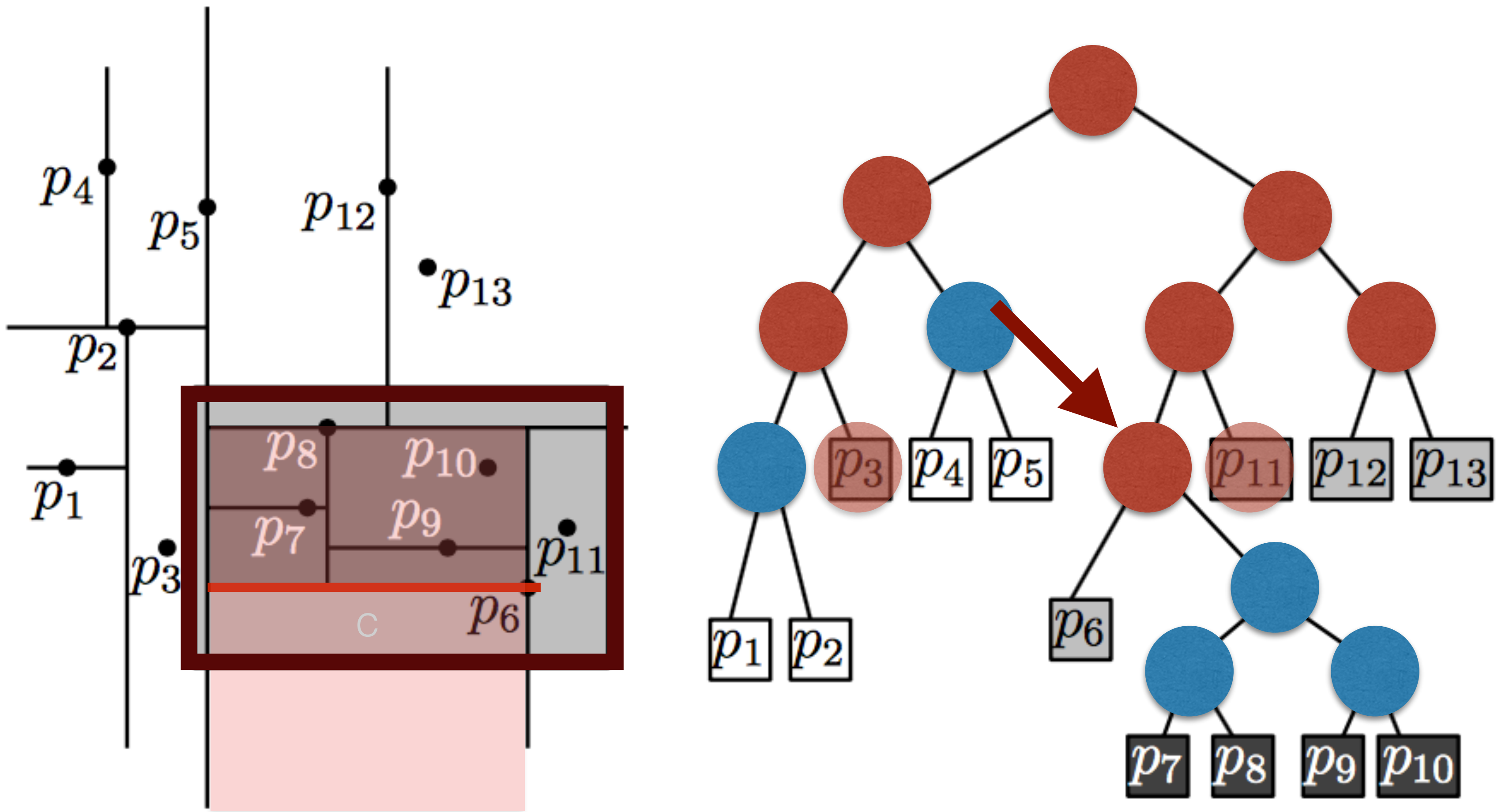
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



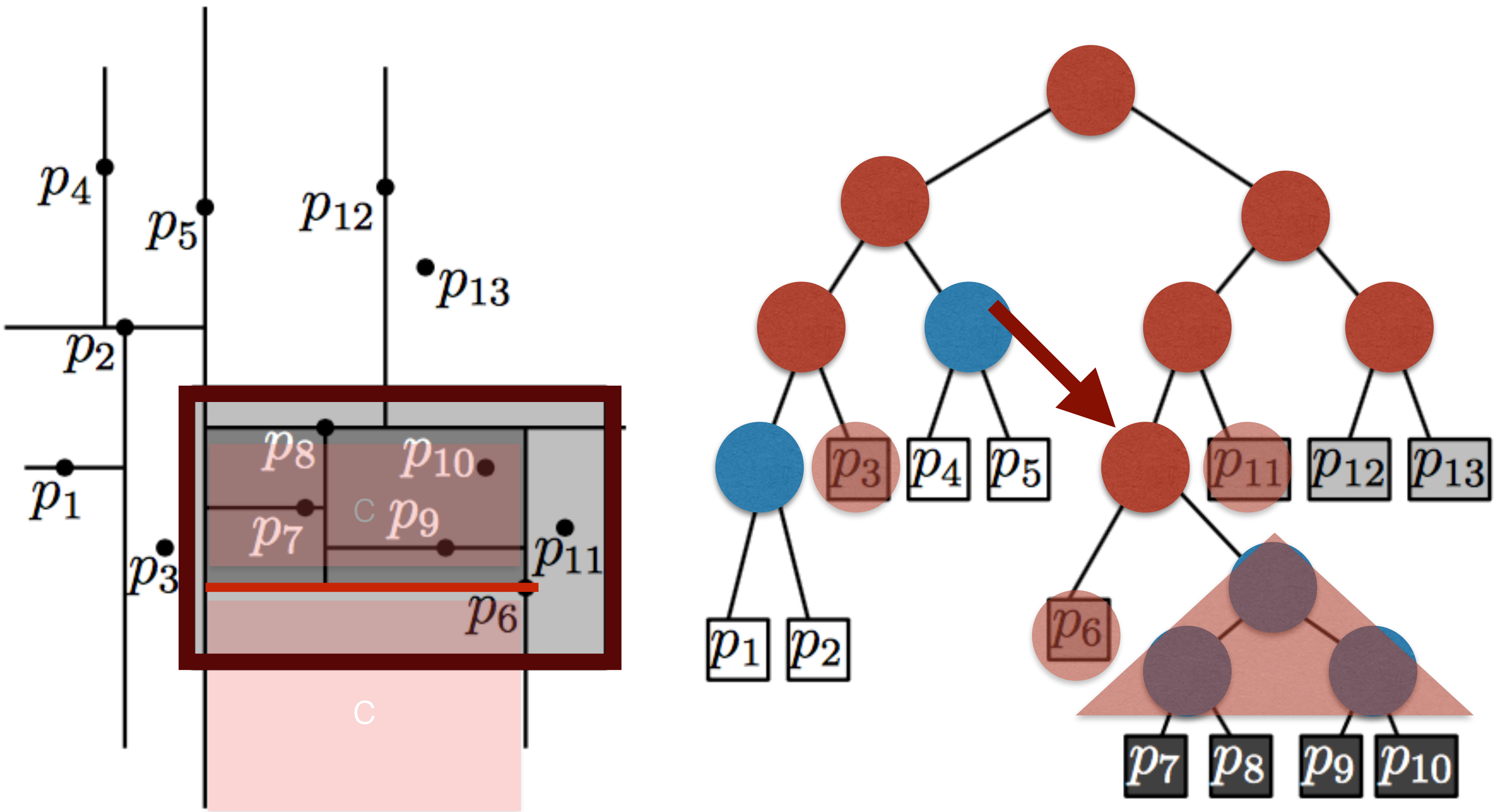
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



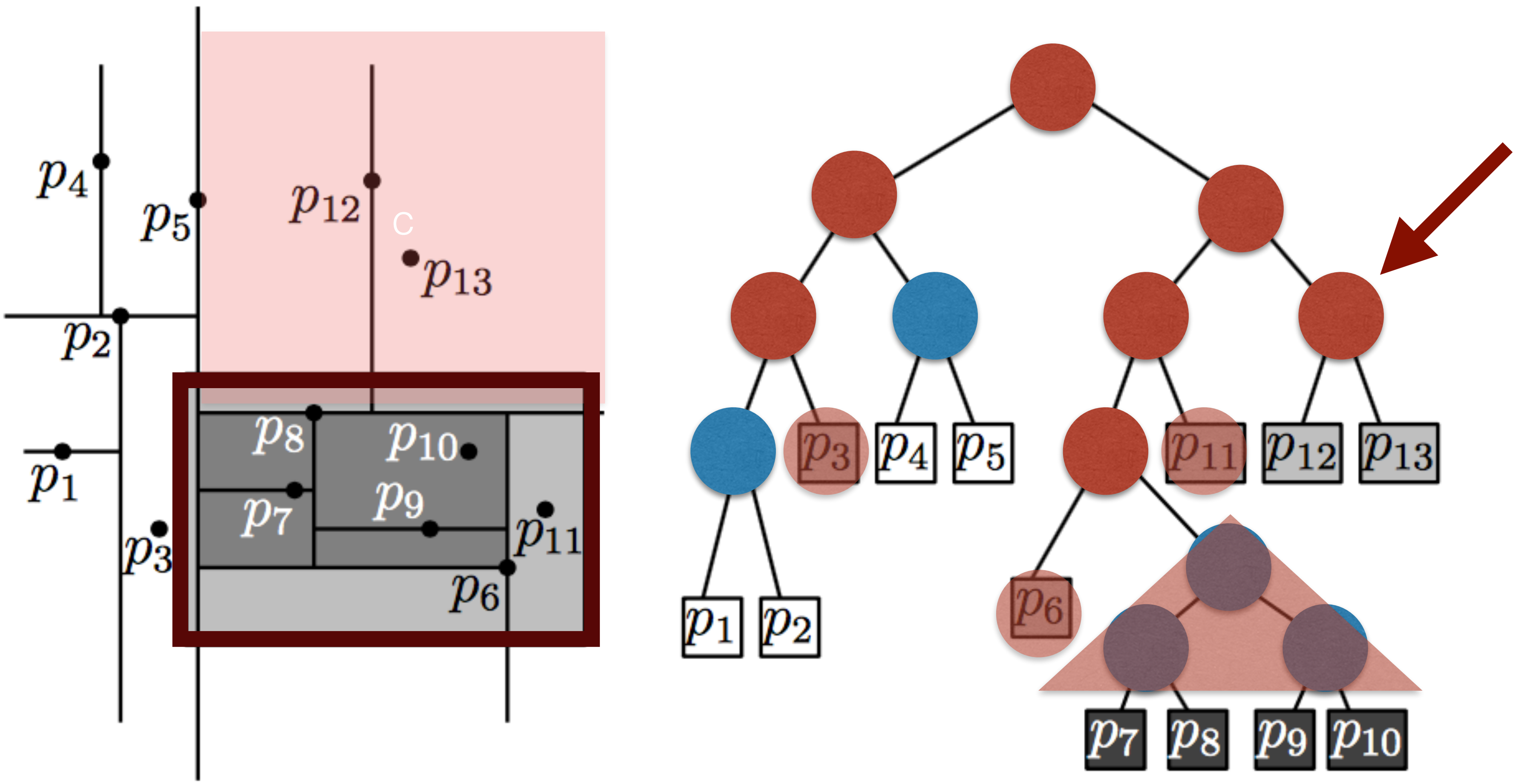
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



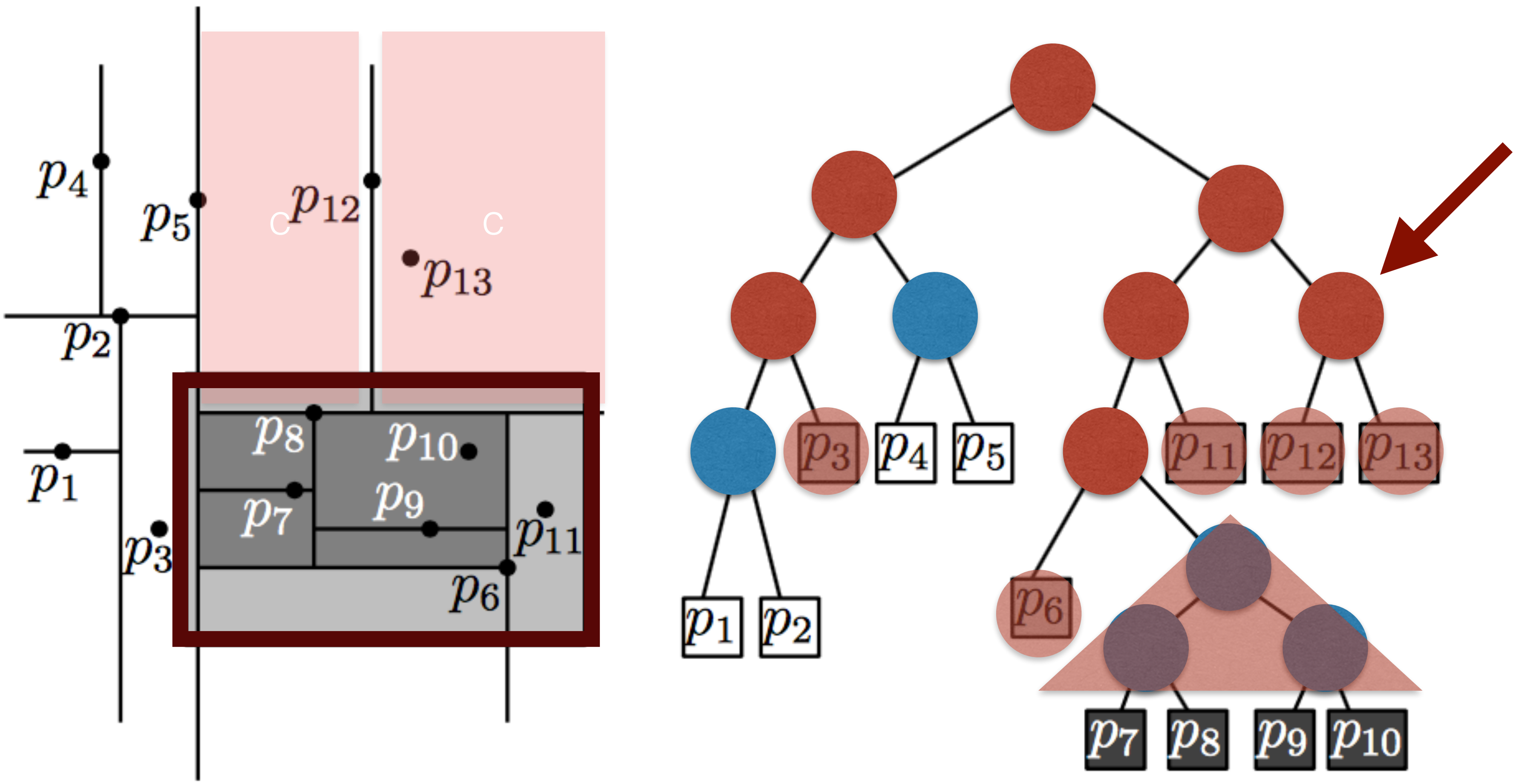
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree



screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

To analyze the time to answer a range query we'll look at the nodes visited in the tree

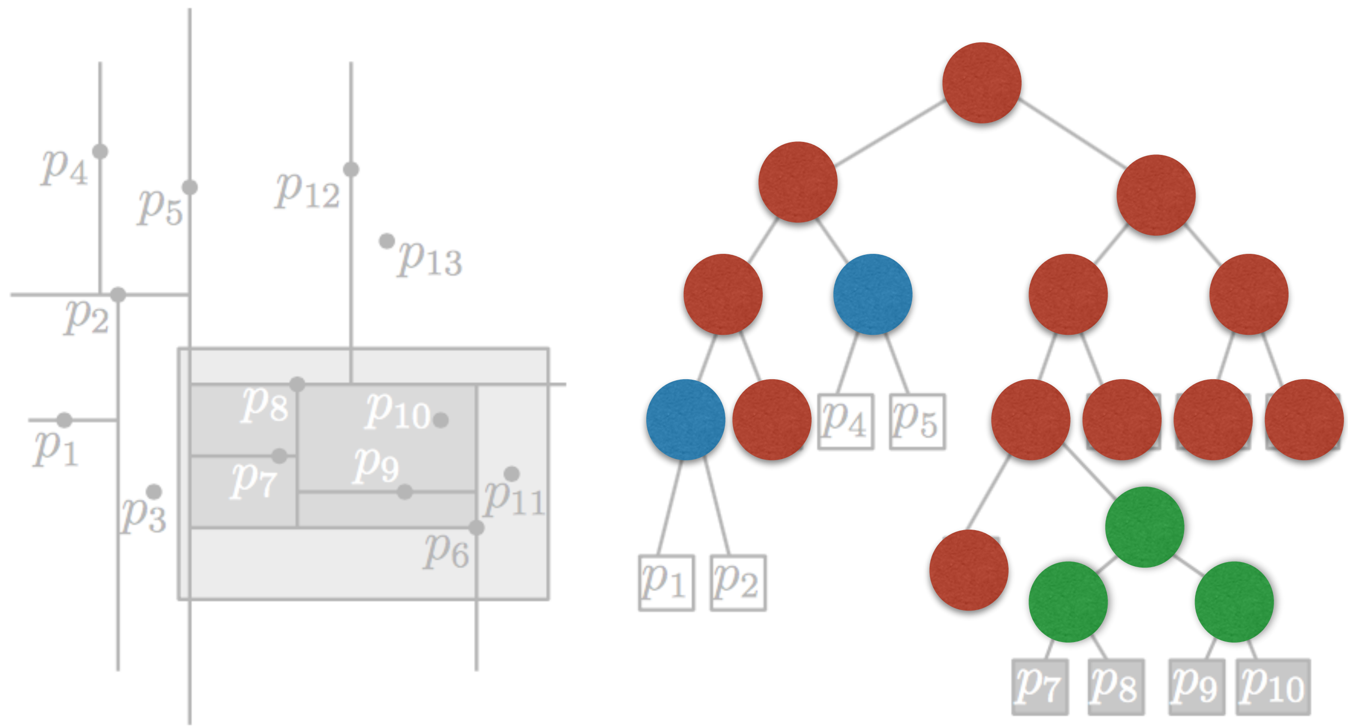


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

never visited by the query

visited may or may not have points to report

visited and whole subtree is output



points guaranteed to be in the range

The time to answer a range search = O(green nodes + red nodes)

Furthermore, nb.green nodes = O(k), where k = size of output

- never visited by the query
- visited may or may not have points to report
- visited and whole subtree is output

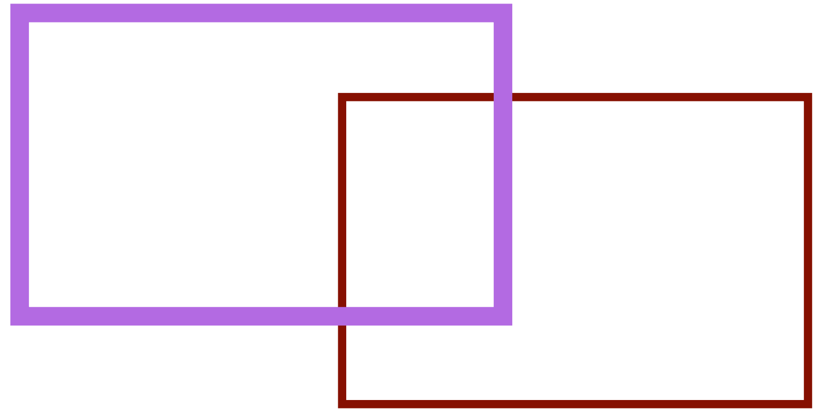
Claim:

- region(v) does not intersect the range
- region(v) intersects the range but is not included in the range
- region(v) is completely included in the range

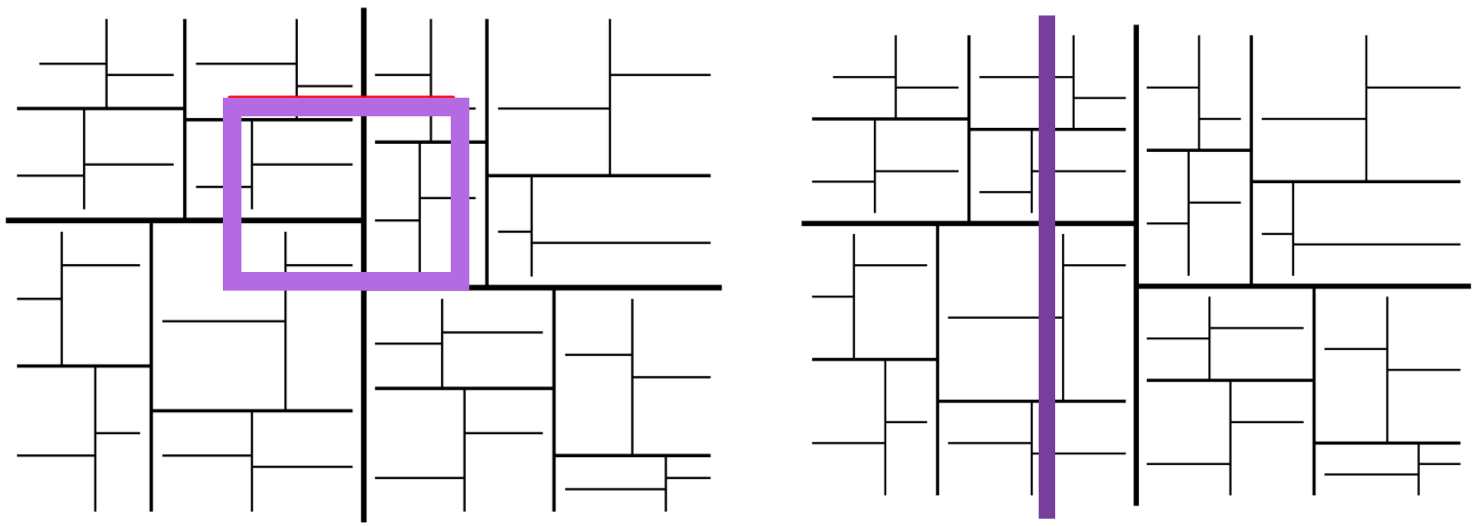
The time to answer a range search = O(red nodes) + O(k)

how many red nodes?

How many red nodes?



nb. red nodes = nb. nodes such that the boundary of their region intersects the boundary of the range

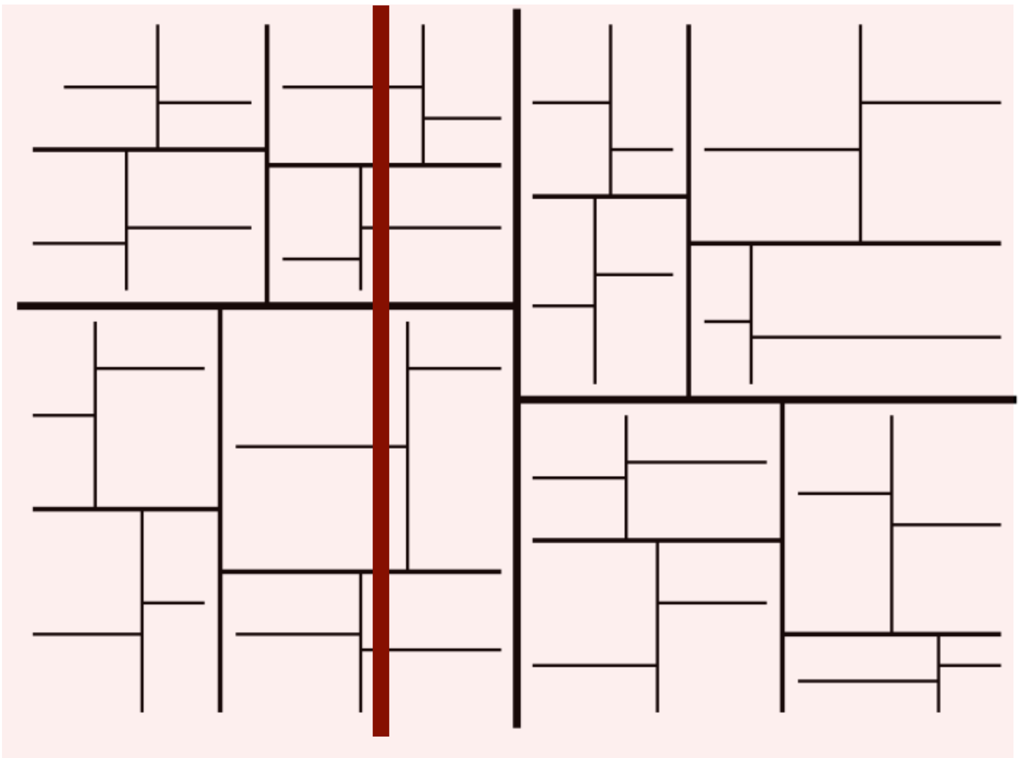


Simplified problem: We'll count the number of nodes whose region intersects a vertical line l.

Number of nodes v such that region(v) intersects a vertical line l?

We'll think recursively, starting at the root:

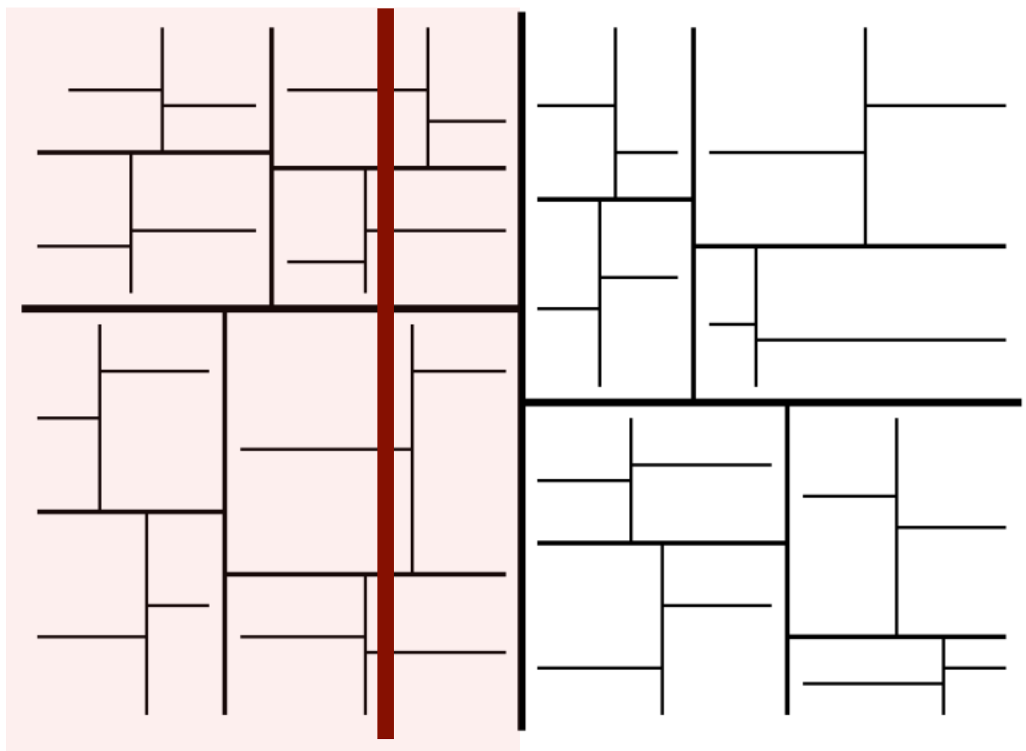
- depth=0: region(root) intersects l $+1$



Number of nodes v such that region(v) intersects a vertical line l?

We'll think recursively, starting at the root:

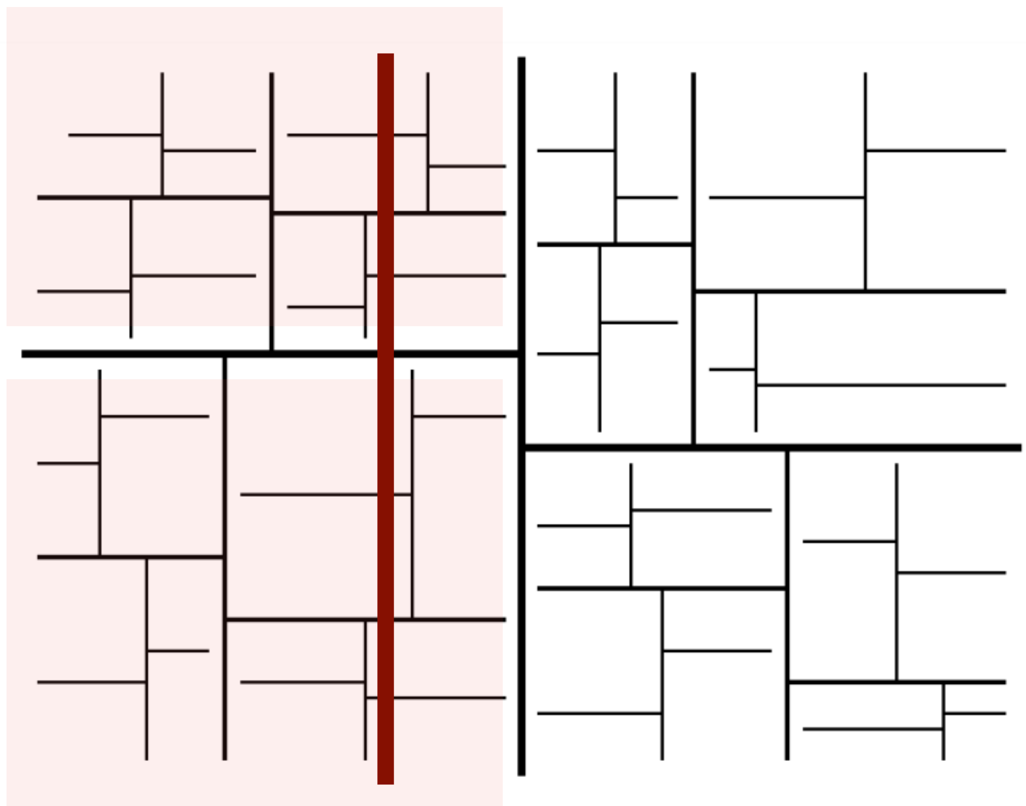
- depth=0: region(root) intersects l $+1$
- depth=1: only one of {left, right} child intersects l $+1$



Number of nodes v such that region(v) intersects a vertical line l?

We'll think recursively, starting at the root:

- depth=0: region(root) intersects l — +1
- depth=1: only one of {left, right} child intersects l — +1
- depth=2: both {left, right} child intersect l — recurse



Number of nodes v such that region(v) intersects a vertical line l?

We'll think recursively, starting at the root:

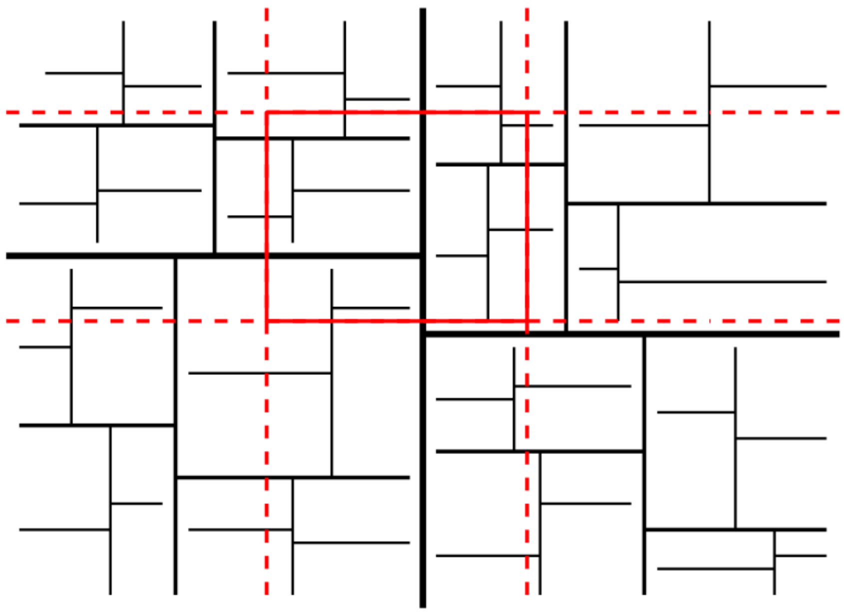
- depth=0: region(root) intersects l — $+1$
- depth=1: only one of {left, right} child intersects l — $+1$
- depth=2: both {left, right} child intersect l — recurse

- Let G(n) = nb. of nodes in a kdtree of n points whose regions interest a vertical line l.
- Then $G(n) = 2 + 2G(n/4)$, and $G(1) = 1$
- This solves to $G(n) = O(\sqrt{n})$

Theorem: Any vertical or horizontal line stabs $O(\sqrt{n})$ regions in the kdtree.

What we know so far:

- The number of red nodes (regions intersected) if the query were a vertical line is $O(\sqrt{n})$
- The same is true if it were a horizontal line
- How about a query rectangle?



If the boundary of a region intersects the range => it must intersect at least one of the two vertical and two horizontal lines

- Theorem: The number of nodes in the kdtree whose region intersects a query range is at most $4 \times O(\sqrt{n}) = O(\sqrt{n})$

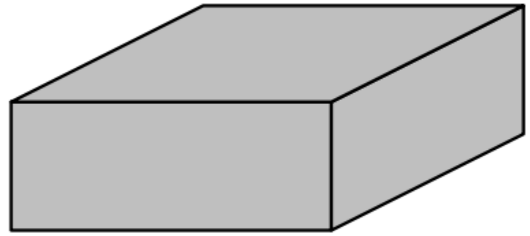
Kd-trees in 2D

Theorem: The kd-tree for a set of n points in 2D

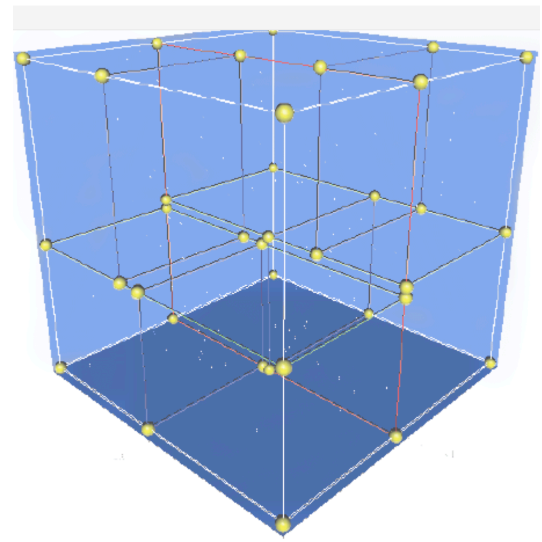
- space: $O(n)$
- built: $O(n \lg n)$ time
- 2-dimensional range queries: $O(\sqrt{n} + k)$ time, where k is the nb. points reported.

Kdtrees generalize easily to d-dimensions

kd-trees in 3D



- A 3D range query is a cube
- A 3D kd-tree alternates splits on x-, y- and z-dimensions



- Construction: Same as in 2D
- Answering range queries: Exactly the same as in 2D
- Analysis:
  - Let $G_3(n)$ = nb. of nodes in a kdtree of n points whose regions interest a vertical line l.
  - Then $G_3(n) = 4G_3(n/8) + O(1)$, and $G(1) = 1$
  - This solves to $G_3(n) = O(n^{2/3})$
  - 3D-range queries in $O(n^{2/3} + k)$

Kd-trees in higher dimensions

Theorem: The kd-tree for a set of n points in d-space:

- uses $O(n)$ space
- can be built in $O(n \lg n)$ time
- d-dimensional range query can be answered in $O(n^{1-1/d} + k)$ time, where k is the nb. points reported.