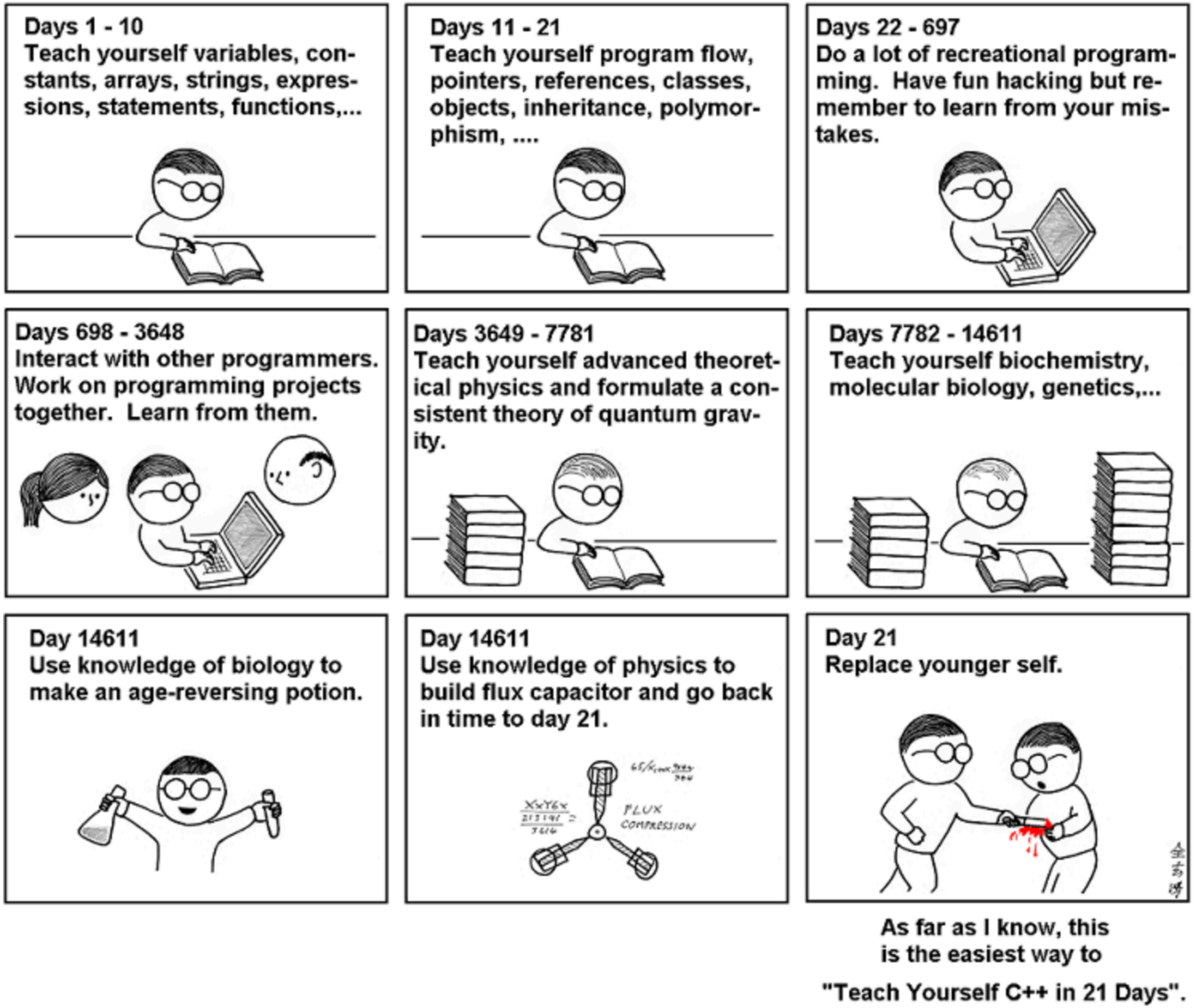
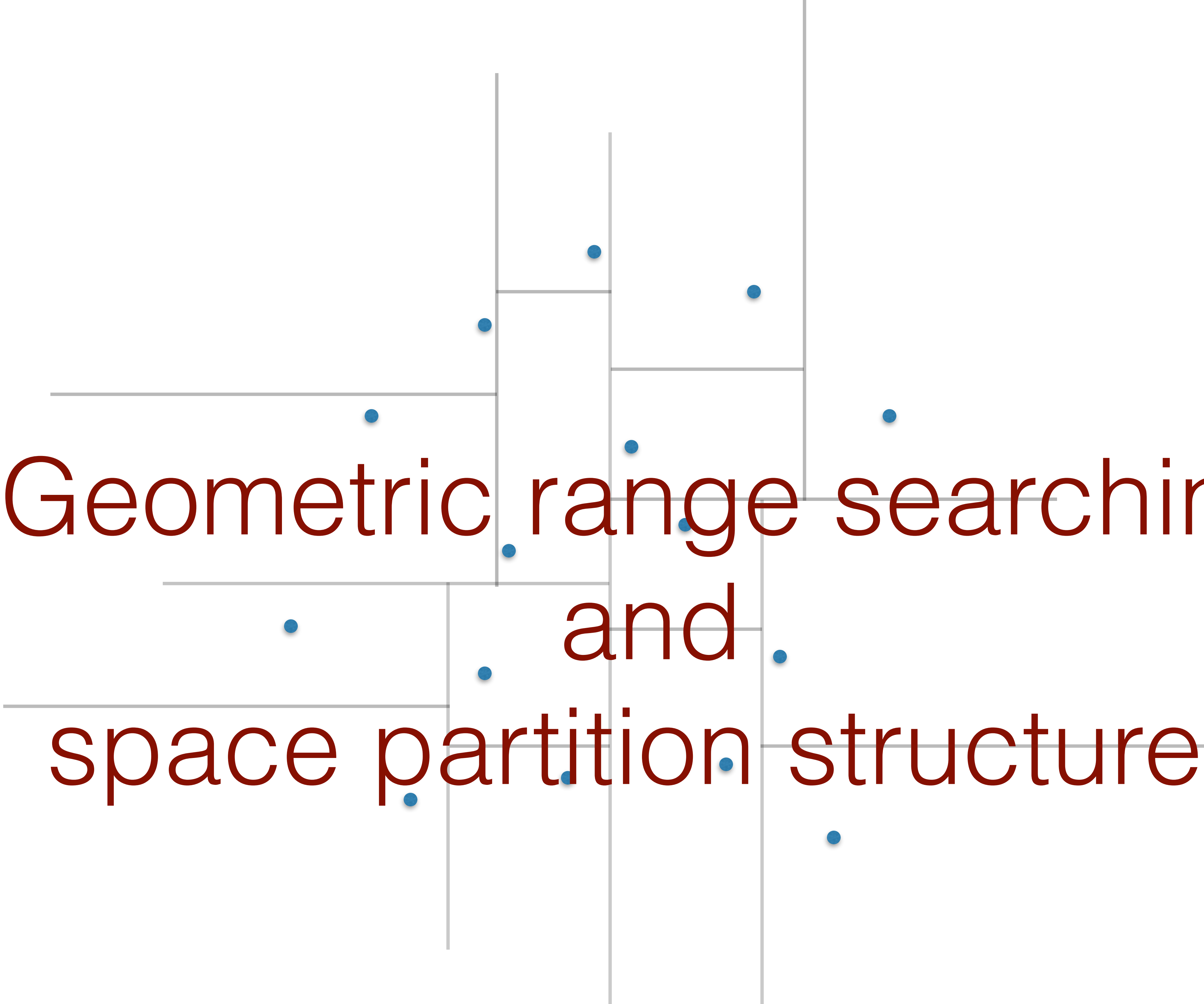
How to Teach Yourself Programming

https://abstrusegoose.com/249



Seriously, why is everyone in **such a rush**?



Geometric range searching and space partition structures

Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

Where we are

"Global" problems

- closest pair
- convex hull
- intersections
- …

Geometric search problems

- range searching
- nearest neighbor
- k-nearest neighbor
- find all roads within 1km of current location
- ..

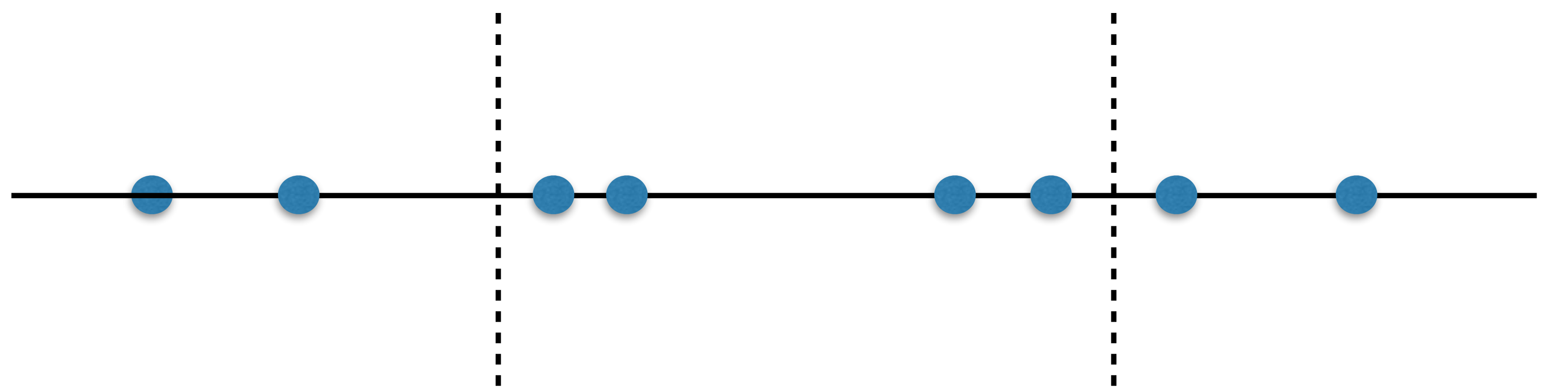
Techniques

- divide-and-conquer
- incremental
- plane sweep
- space decomposition
- ..

Today

1D Range searching

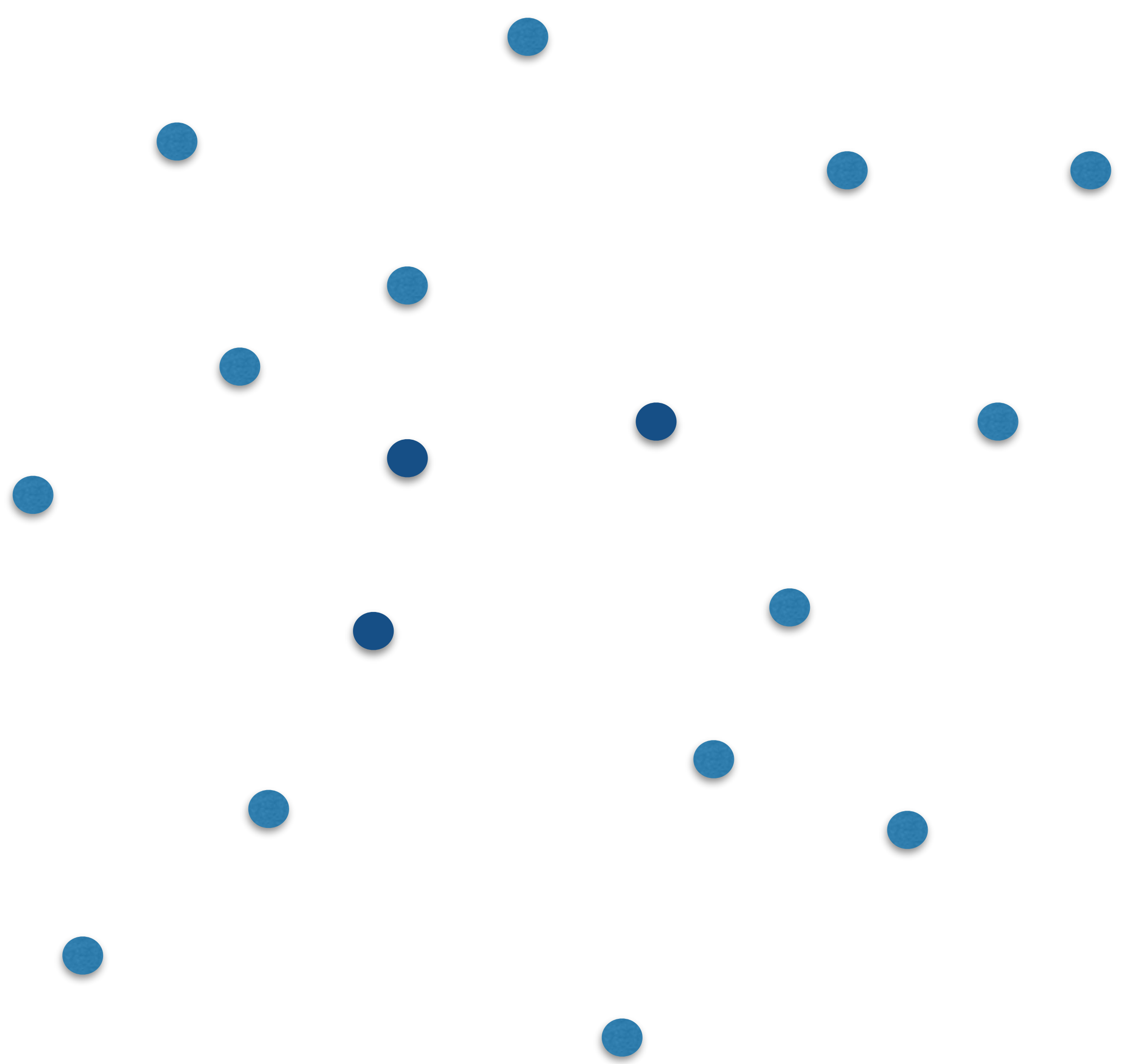
Given a set of n points on the real line, preprocess them into a data structure to support fast range queries.



1D

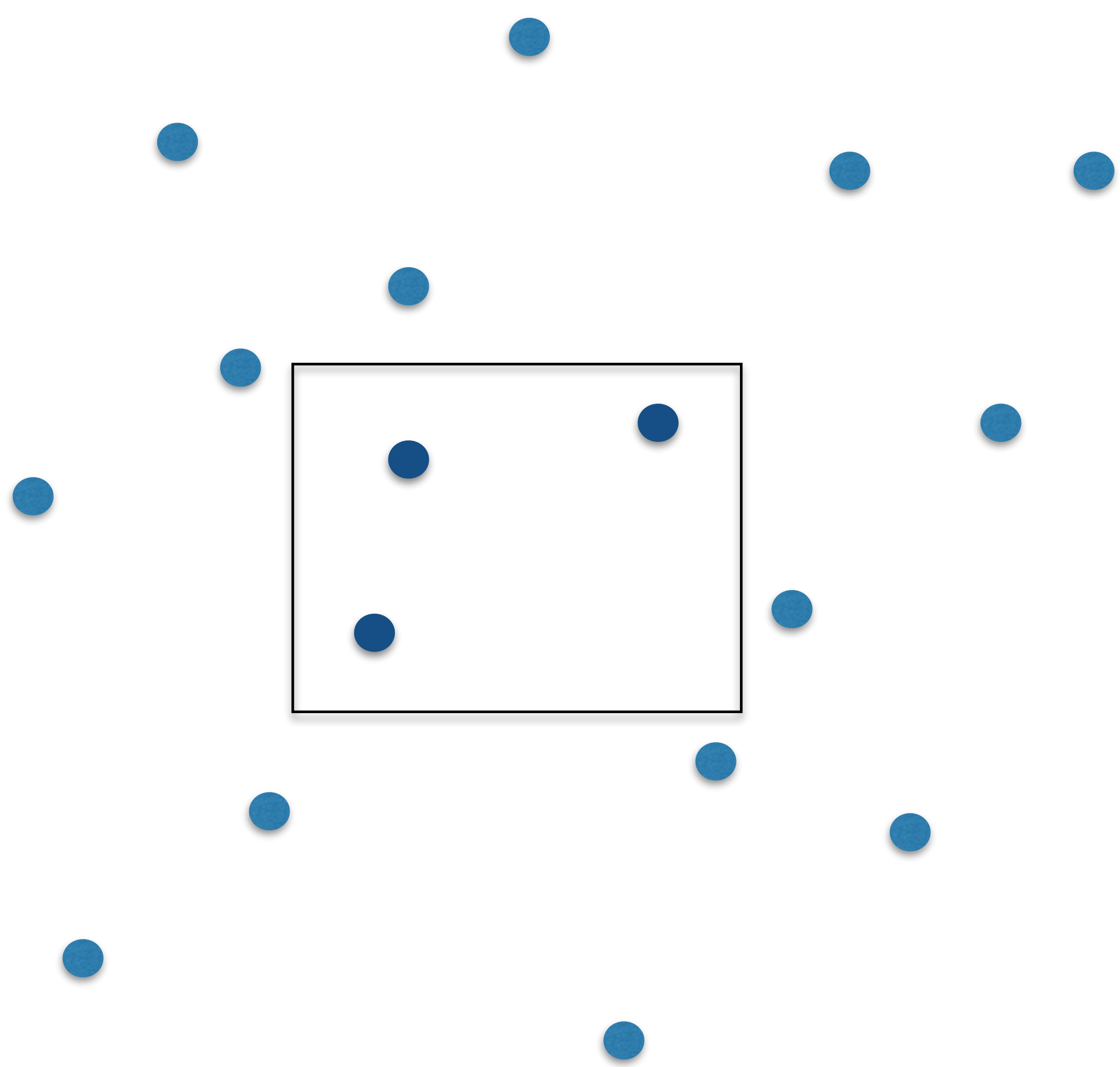
2D Range searching

Given a set of points, preprocess them into a data structure to support fast range queries.



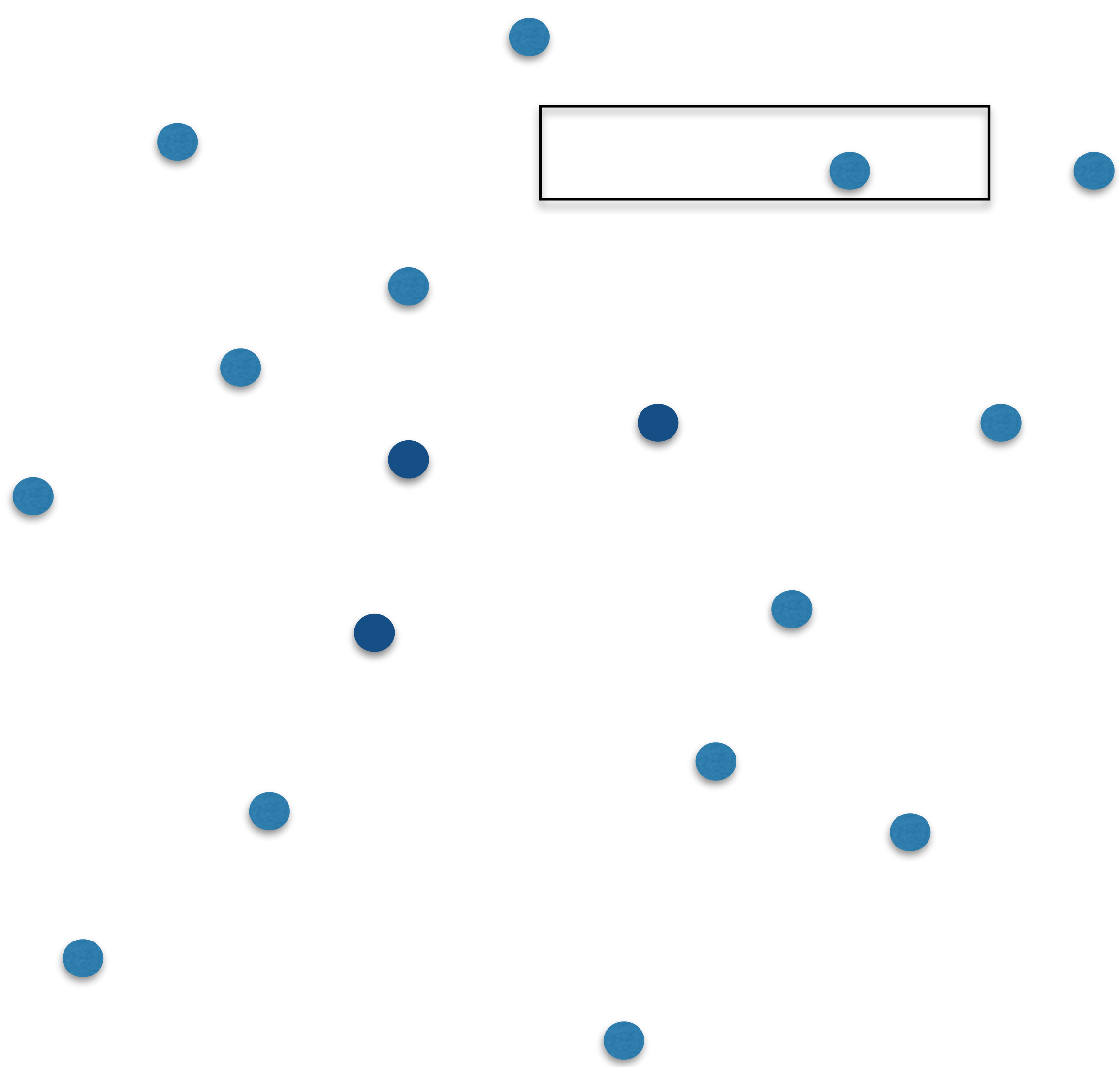
2D Range searching

Given a set of points, preprocess them into a data structure to support fast range queries.



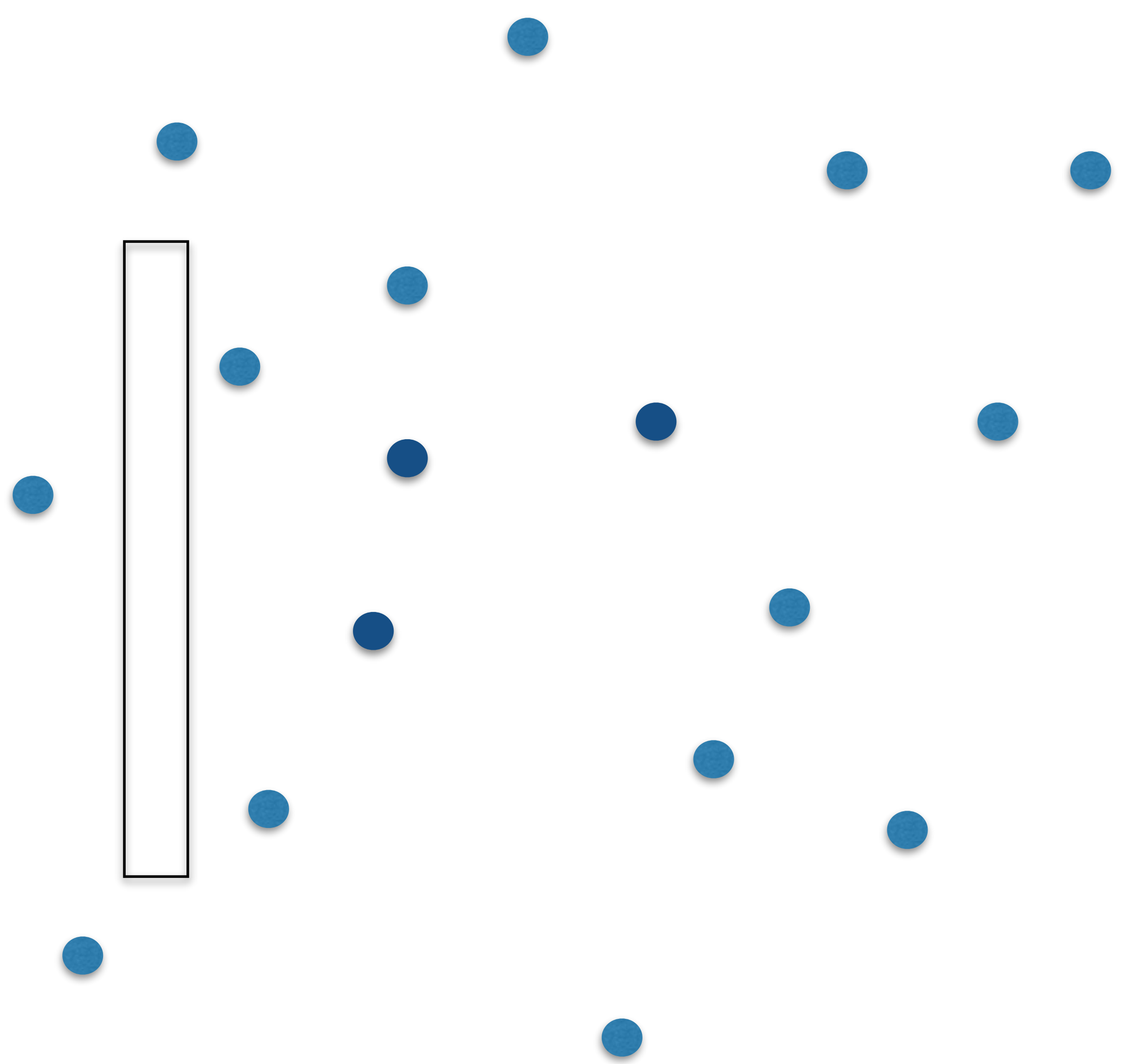
2D Range searching

Given a set of points, preprocess them into a data structure to support fast range queries.



2D Range searching

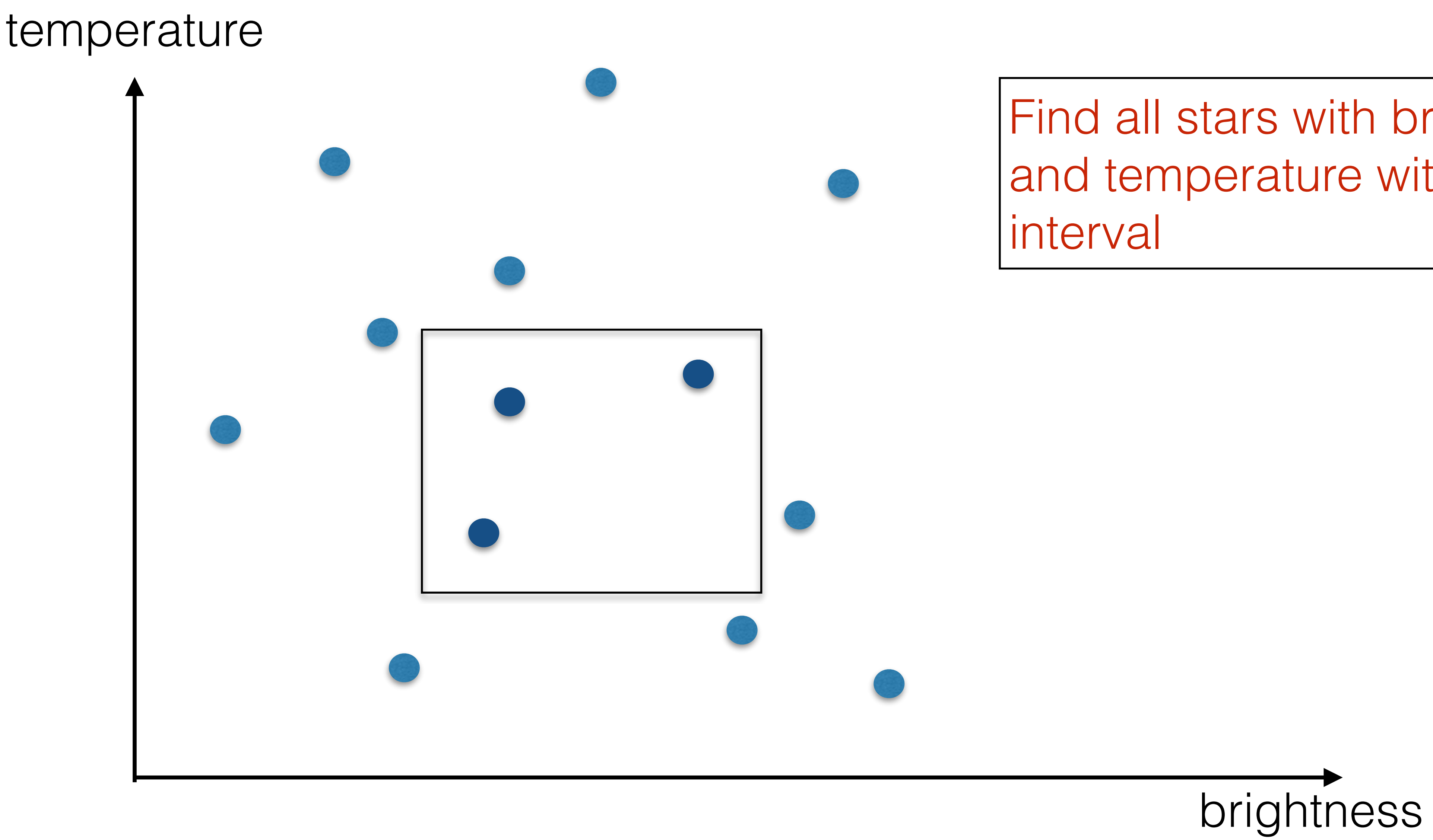
Given a set of points, preprocess them into a data structure to support fast range queries.



Why?

Arise in settings that are not geometrical.

Database of stars. A star = (brightness, temperature,……)

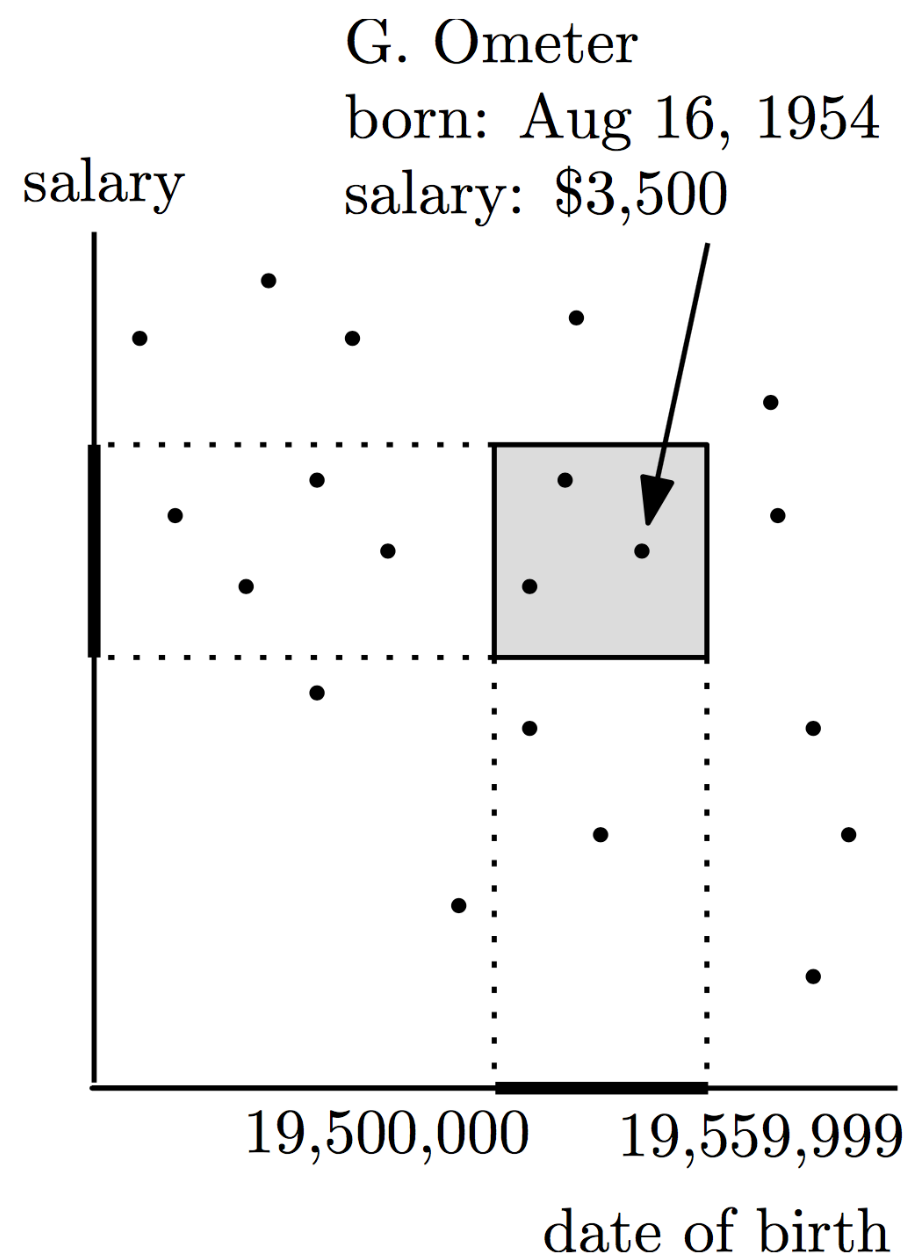


Find all stars with brightness and temperature within a given interval

Why?

Database of employees. An employee = (age, salary,……)

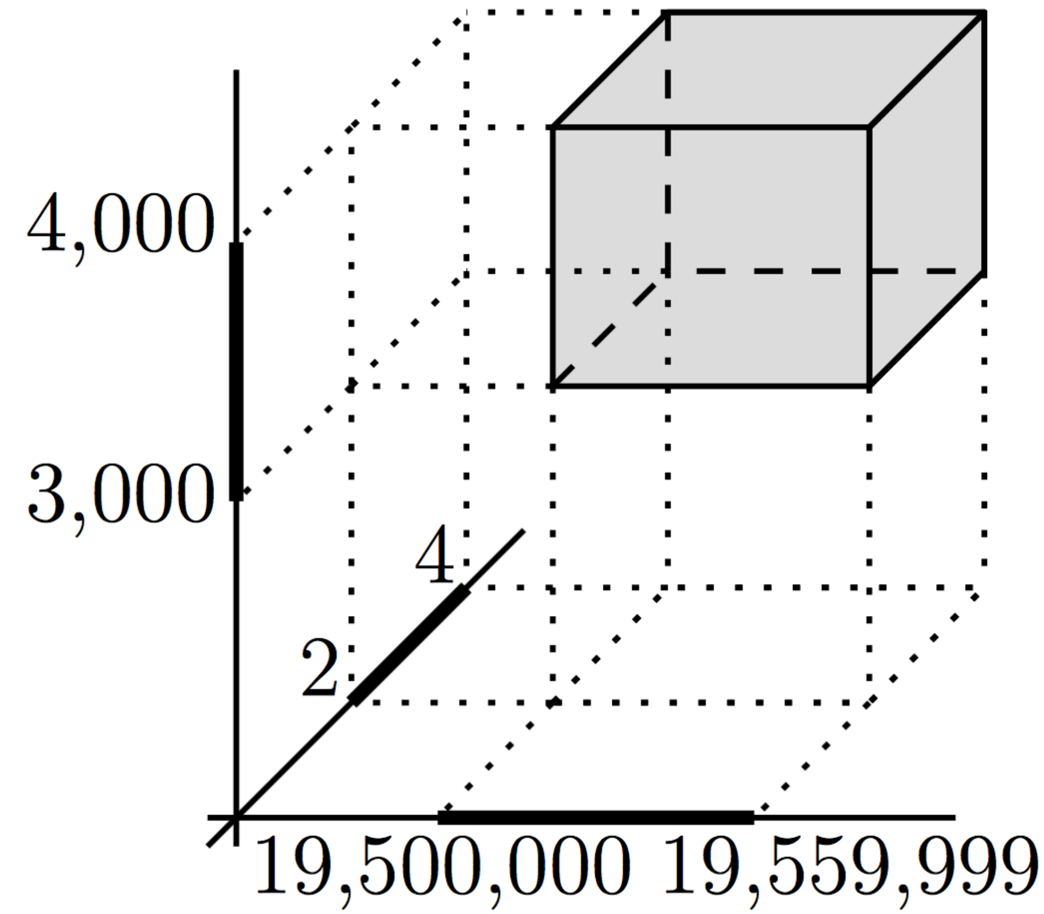
A database query may ask for all employees with age between a_1 and a_2, and salary between s_1 and s_2



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

Why?

Example of a 3-dimensional (orthogonal) range query: children in $[2, 4]$, salary in $[3000, 4000]$, date of birth in $[19,500,000\ ,\ 19,559,999]$



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

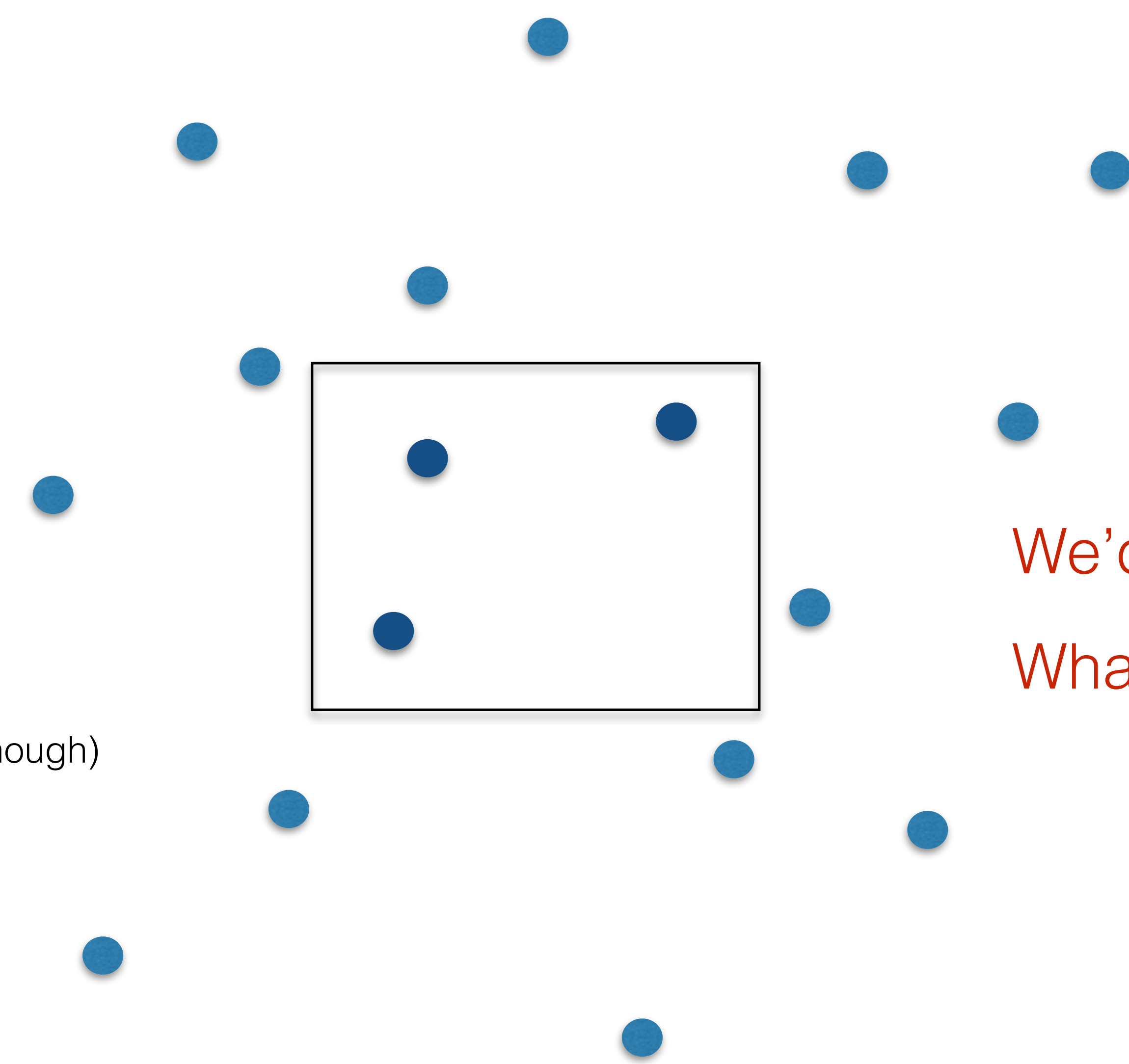
- n: size of the input (number of segments)
- k: size of output (number of points inside range)

Range searching in 2D

How?

The naive approach:

- No data structure: traverse and check in O(n)
- Note: good when k is large



Points are static or dynamic?

We'll assume static (it's hard enough)

We'd like to do better.

What sort of bounds can we expect?

- n: size of the input (number of points)
- k: size of output (number of points inside range)

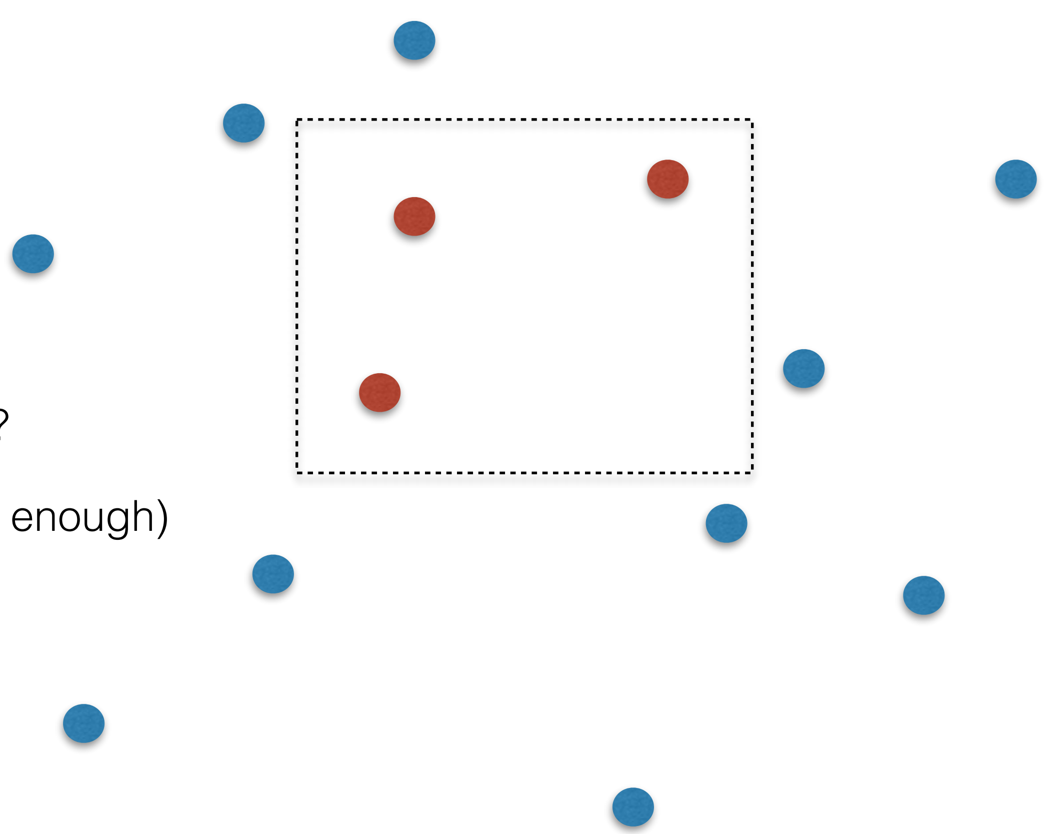
Range searching in 2D: How?

The naive approach:

- No data structure: traverse and check in O(n)
- Analysis: O(n)
  - Note: good when k is large

We'd like to do better.

What sort of bounds can we expect?



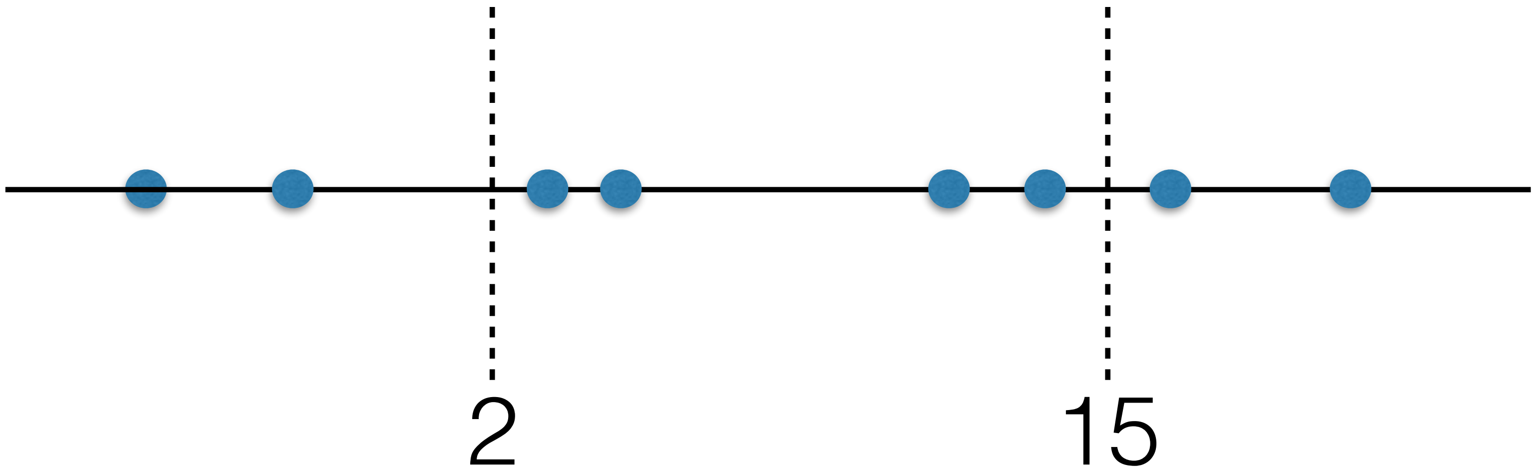
Points are static or dynamic?

We'll assume static (it's hard enough)

1D range searching

1D Range searching

Given a set of n points on the real line, preprocess them into a data structure to support fast range queries.



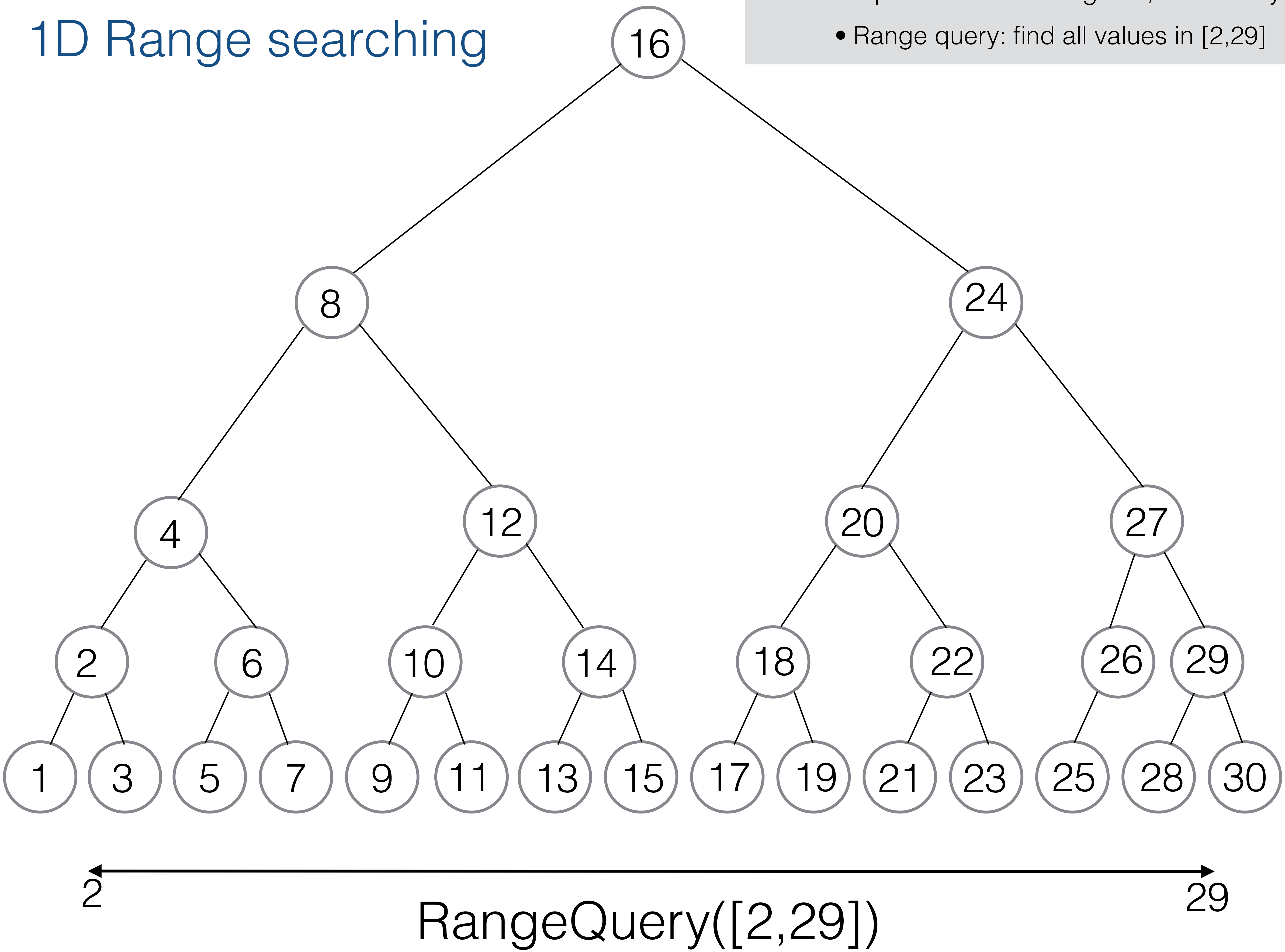
find all values in [2,15]

How do we solve this and how fast?

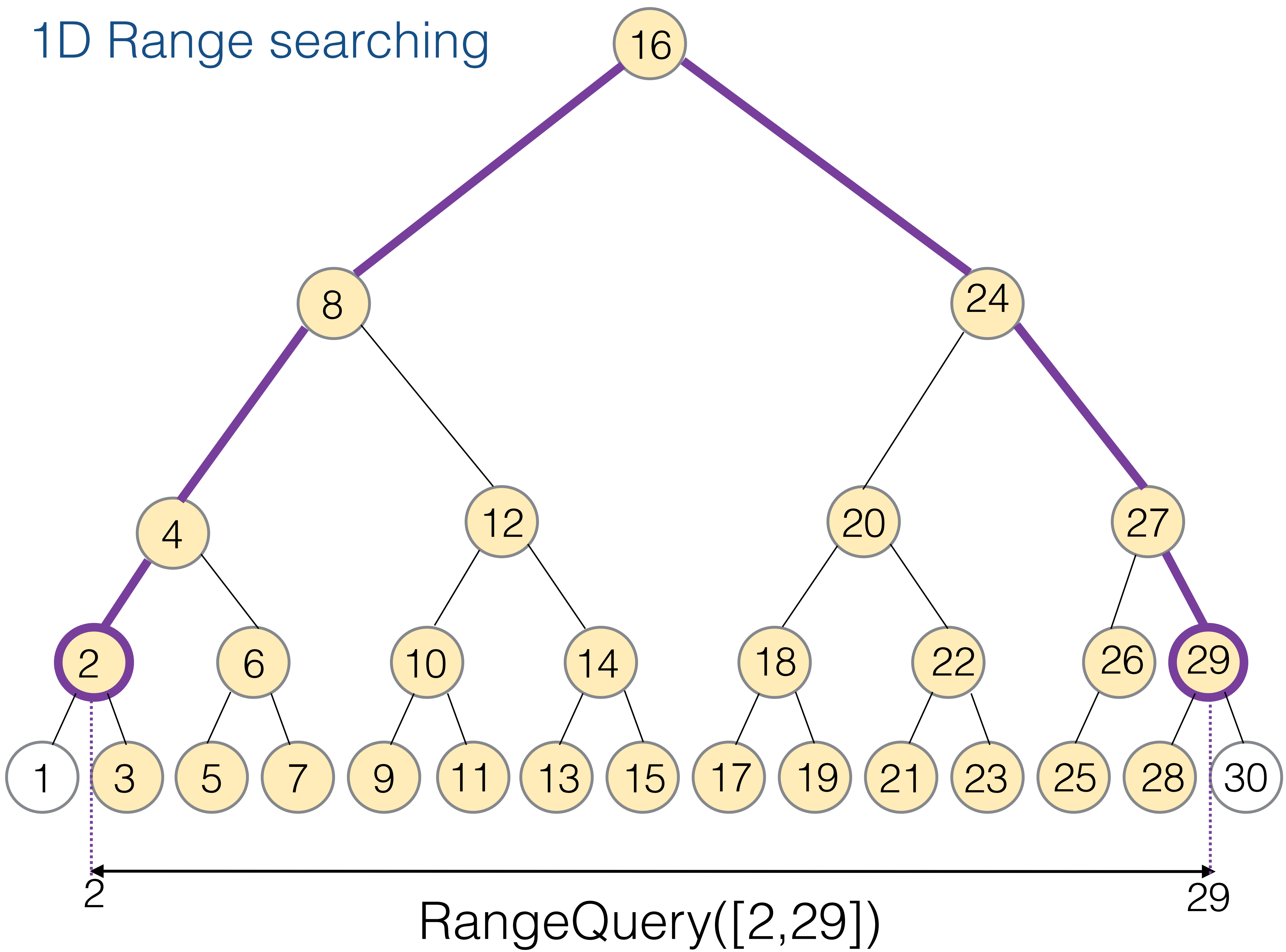
1D Range searching

Example

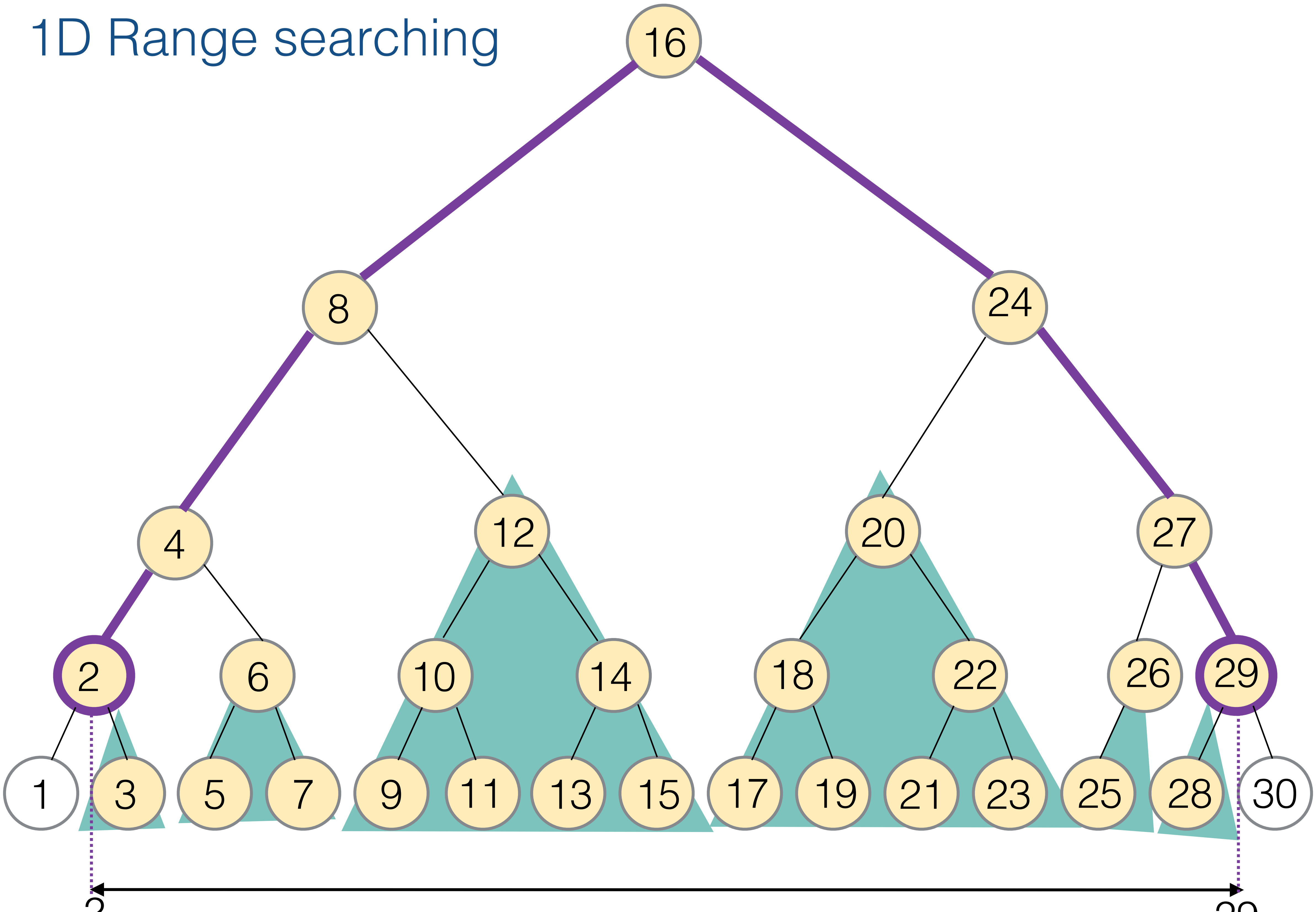
- Input: values 1 through 30, in arbitrary order
- Range query: find all values in [2,29]



1D Range searching



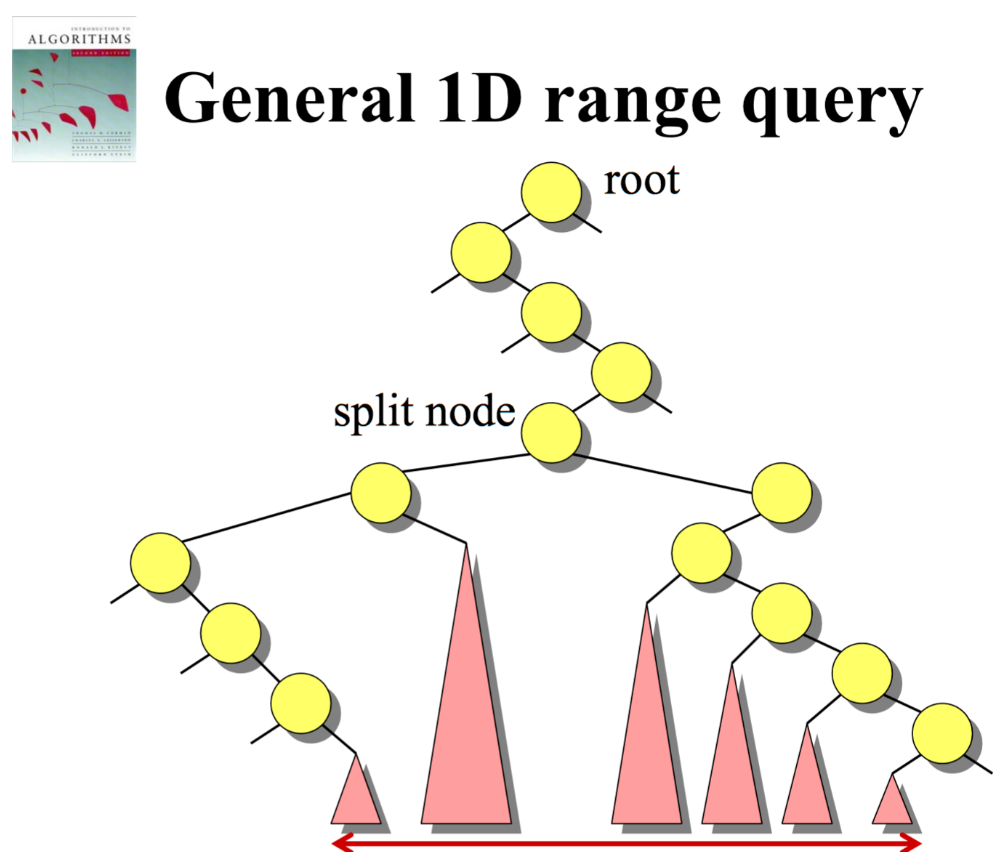
1D Range searching



The k points in the range are in O(lg n) subtrees

1D Results

- A set of n points (1D) can be pre-processed into a BBST such that:
  - Build: O(n lg n)
  - Space: O(n)
  - Range queries: O(lg n +k)
  - Dynamic: points can be inserted/deleted in O(lg n)



1D Results

- A set of n points (1D) can be pre-processed into a BBST such that:
  - Build: O(n lg n)
  - Space: O(n)
  - Range queries: O(lg n +k)
  - Dynamic: points can be inserted/deleted in O(lg n)

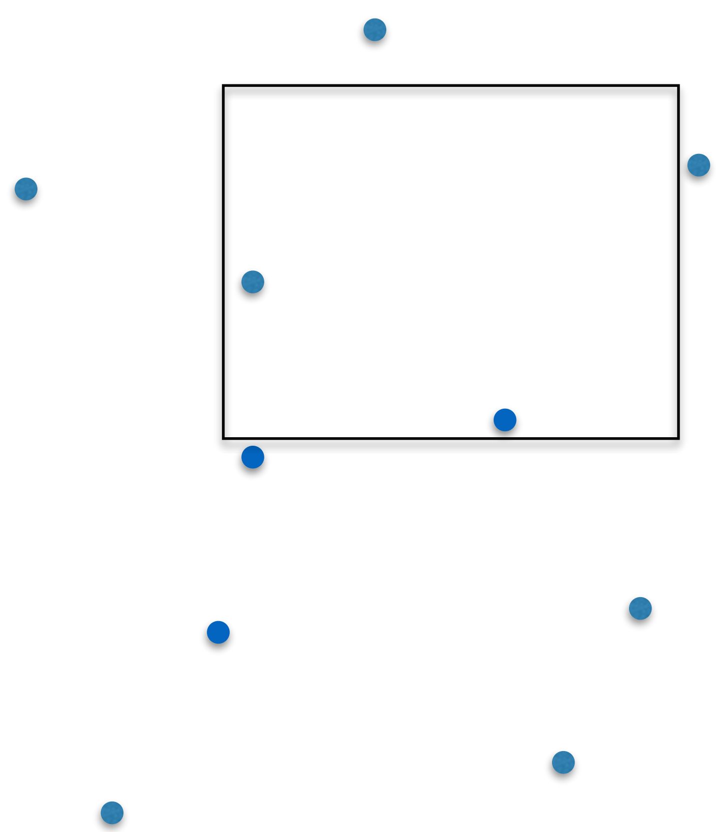
2D

- A set of n 2d-points can be pre-processed into a structure such that:
  - Build: O(n lg n)
  - Space: O(n)
  - Range queries: O($\lg^2$ n + k)

These bounds would be nice

But how?

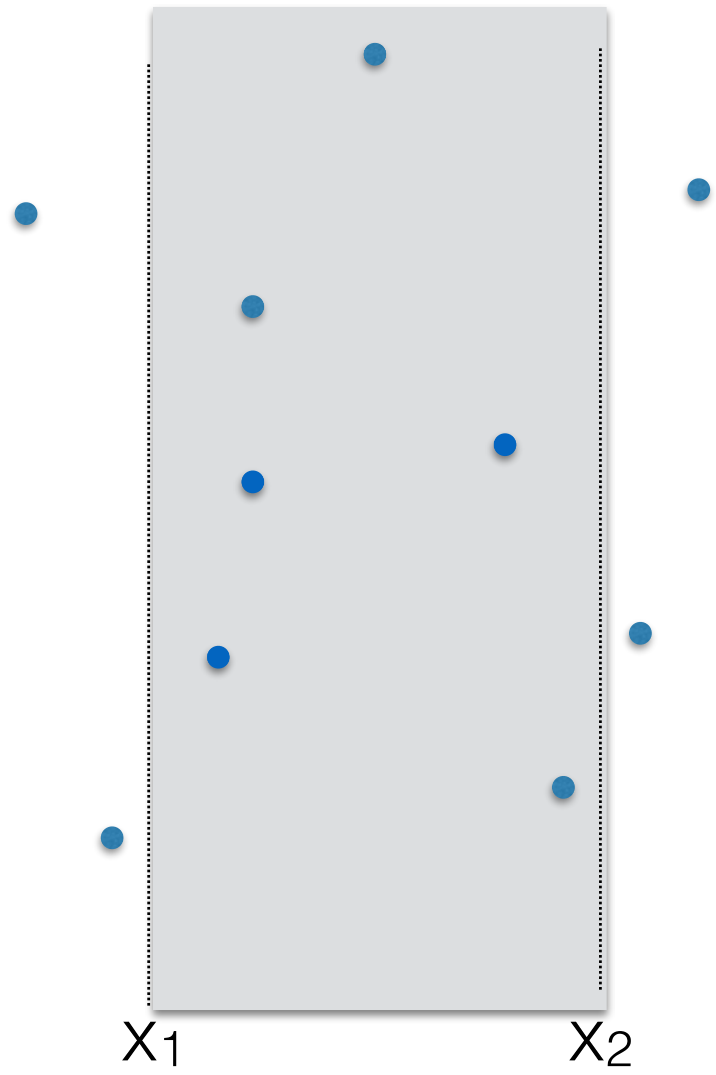
How could we use 1D structure for 2D?



Denote query $[x_1, x_2] \times [y_1, y_2]$

Could it be as simple as ..

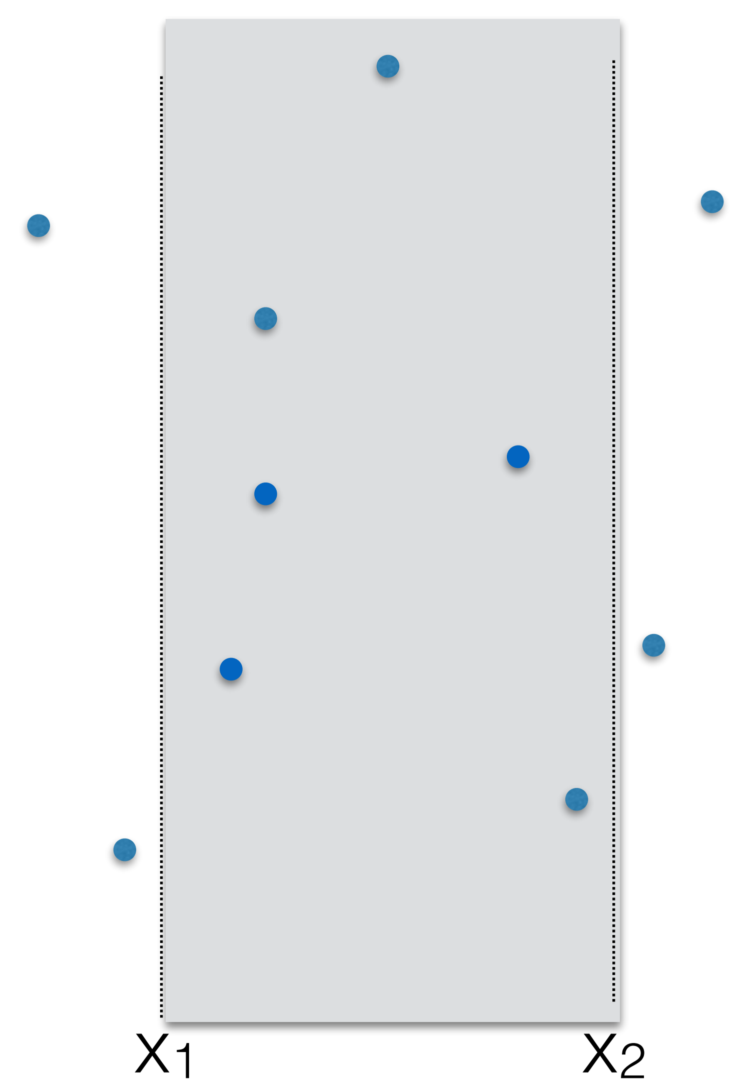
- Find all points with the x-coordinates in the correct range $[x_1, x_2]$



Denote query $[x_1, x_2] \times [y_1, y_2]$

Could it be as simple as ..

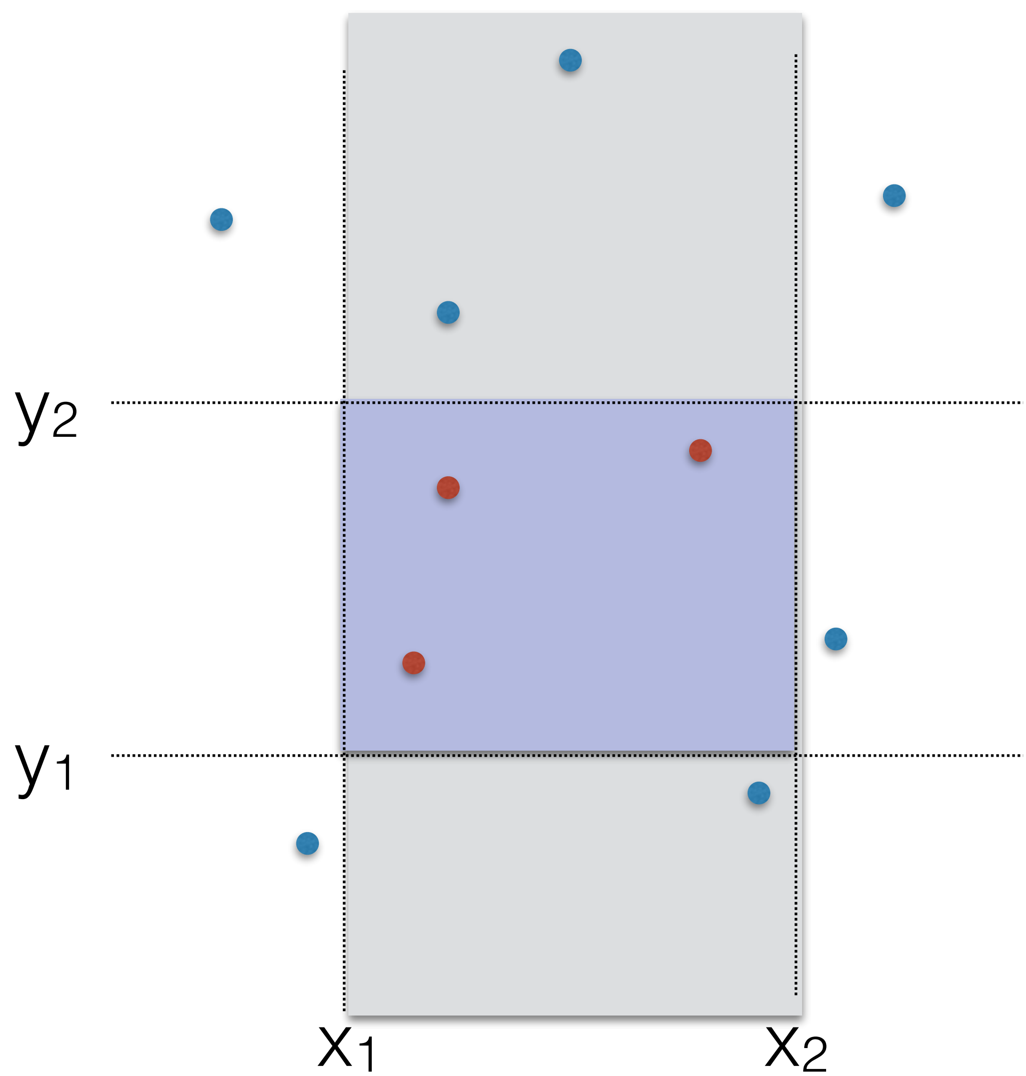
- Find all points with the x-coordinates in the correct range $[x_1, x_2]$
- Out of these points, find all points with the y-coord in the correct range $[y_1, y_2]$



Denote query $[x_1, x_2] \times [y_1, y_2]$

Could it be as simple as ..

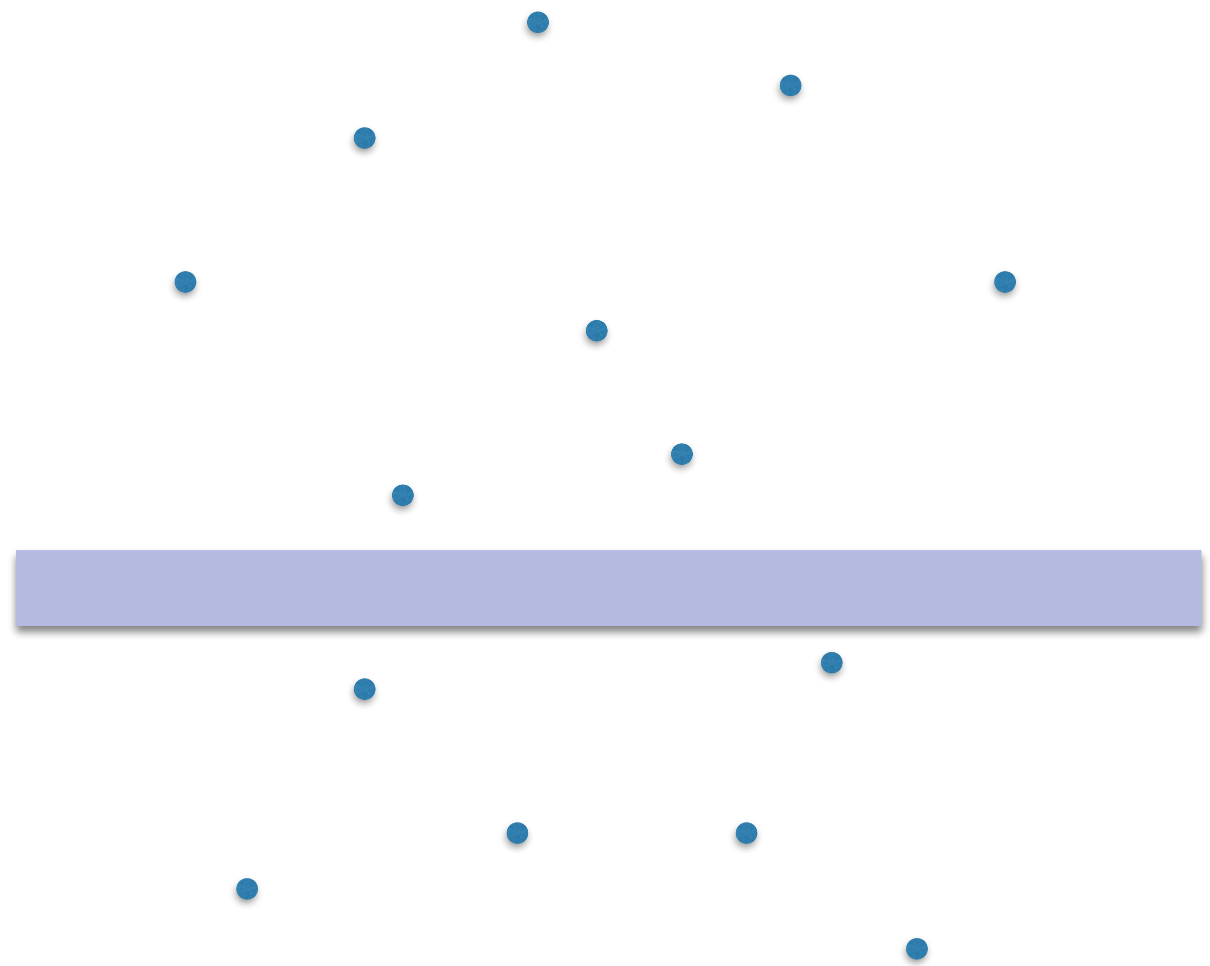
- Find all points with the x-coordinates in the correct range $[x_1, x_2]$
- Out of these points, find all points with the y-coord in the correct range $[y_1, y_2]$



Does this work?

How fast is it?

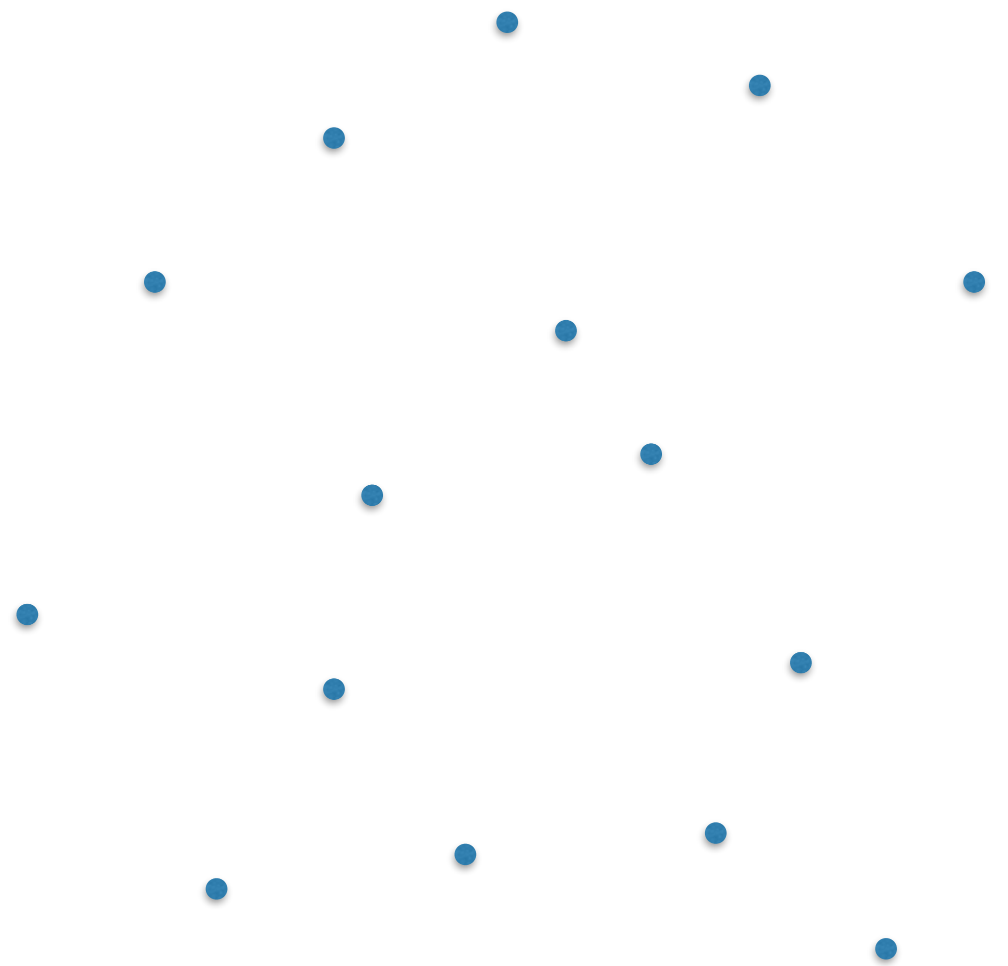
Come up with a worst case scenario



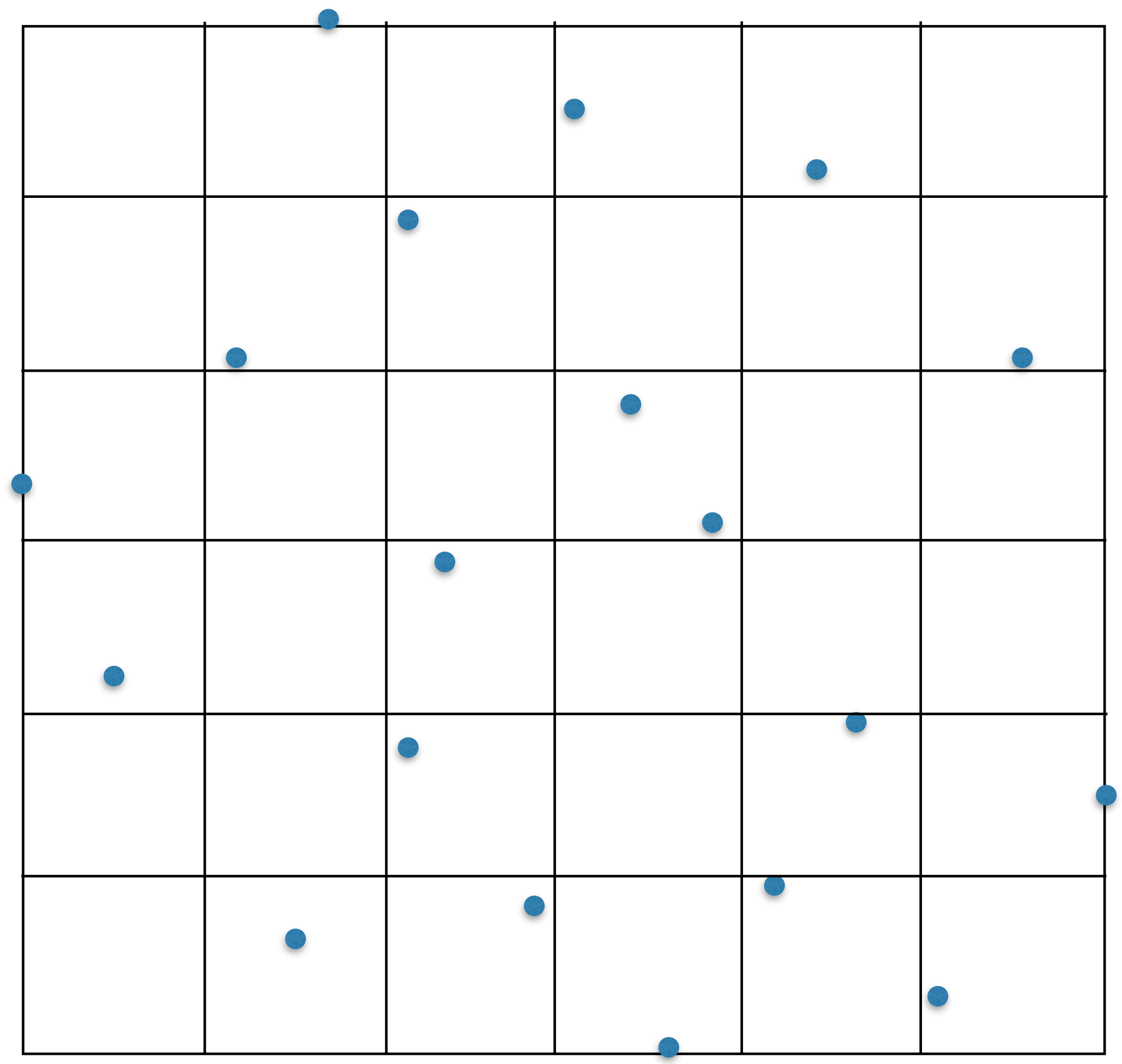
We’ll partition the space, store it in a data structure and use it to speed up searching

Space decomposition methods

The simplest space decomposition is a grid



The grid method



For example…

```
class Grid  {

    double x1, x2, y1, y2;              // the bounding box of the grid

    int m;                              // number of cells in the grid (m-by-m)

    double cellsize_x, cellsize_y;  // size of a grid cell

    List<point2D*>   ***g;              //2D array of list*;  g[i][j] contains the pointer
                                        //to the list of points that lie in cell [i][j]




    Grid (Point p[], int n, int m);

    List<Point2D*>* rangeQuery(double x1, x2, y1, y2);

    ....
};
```

The grid method

- Creating a grid of m-by-m cells from a set of points P
    1. Figure out a rectangle that contains P (e.g. x_{min}, x_{max}, y_{min},y_{max})
    2. Allocate a 2d array of lists, all initially empty
    3. For each point p in P: figure out which cell i, j contains p, and insert p in the list corresponding to g[i][j]

The grid method

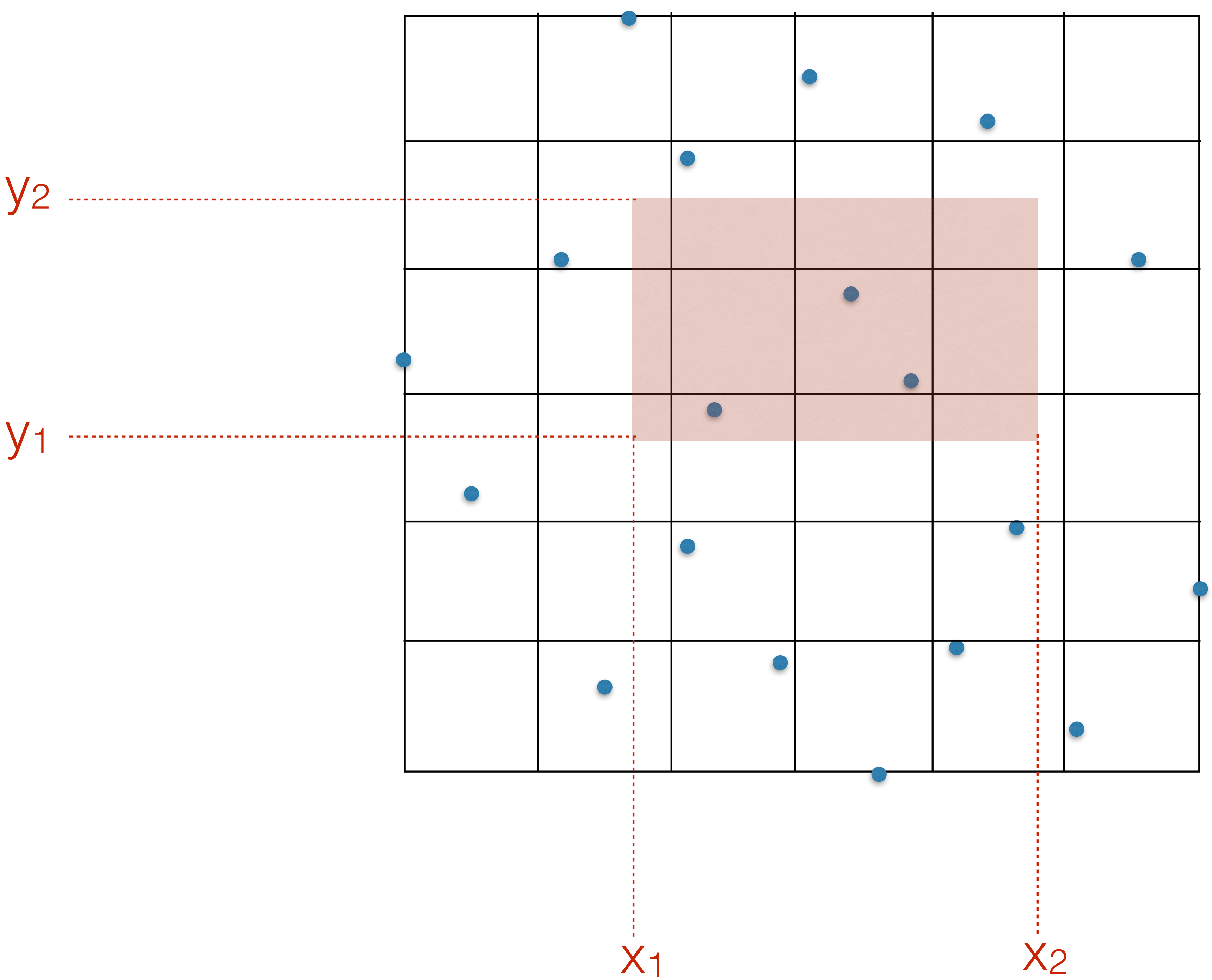
- Creating a grid of m-by-m cells from a set of points P
  1. Figure out a rectangle that contains P (e.g. x_{min}, x_{max}, y_{min}, y_{max})
  2. Allocate a 2d array of lists, all initially empty
  3. For each point p in P: figure out which cell i, j contains p, and insert p in the list corresponding to g[i][j]

```
g = new (List<point2D*>**)[m];
for (int i=0; i<m; i++) {
        g[i] = new (List<point2D*>*) [m];
        for (int j=0; j<m; j++) {
                g[i][j] = new List<point2D*>;
        }
}
```

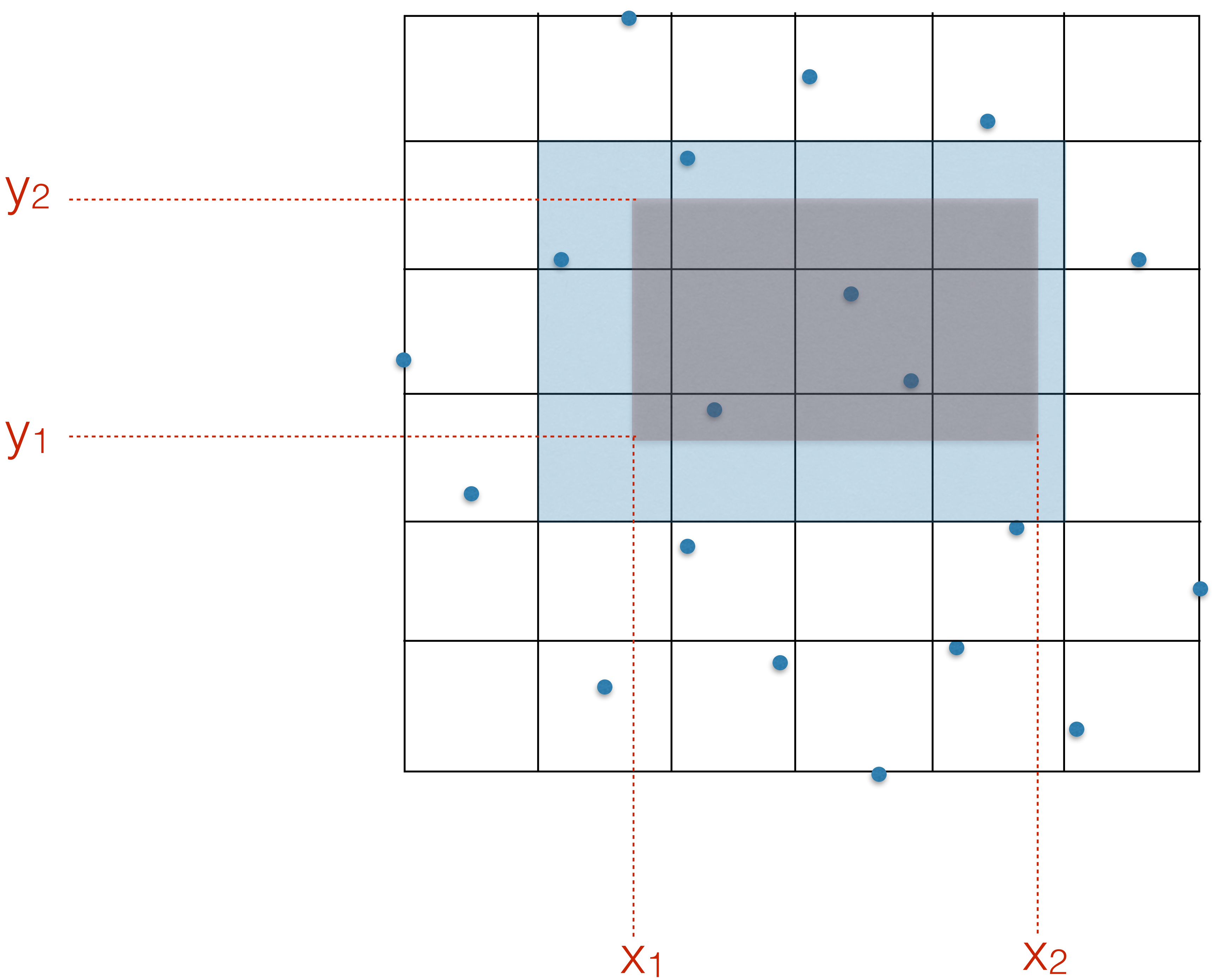
for each point p

```
j = (p.x - xmin)/cellsize_x;
i = (ymax - p.y/cellsize_y;
g[i][j]->insert(&p);
```

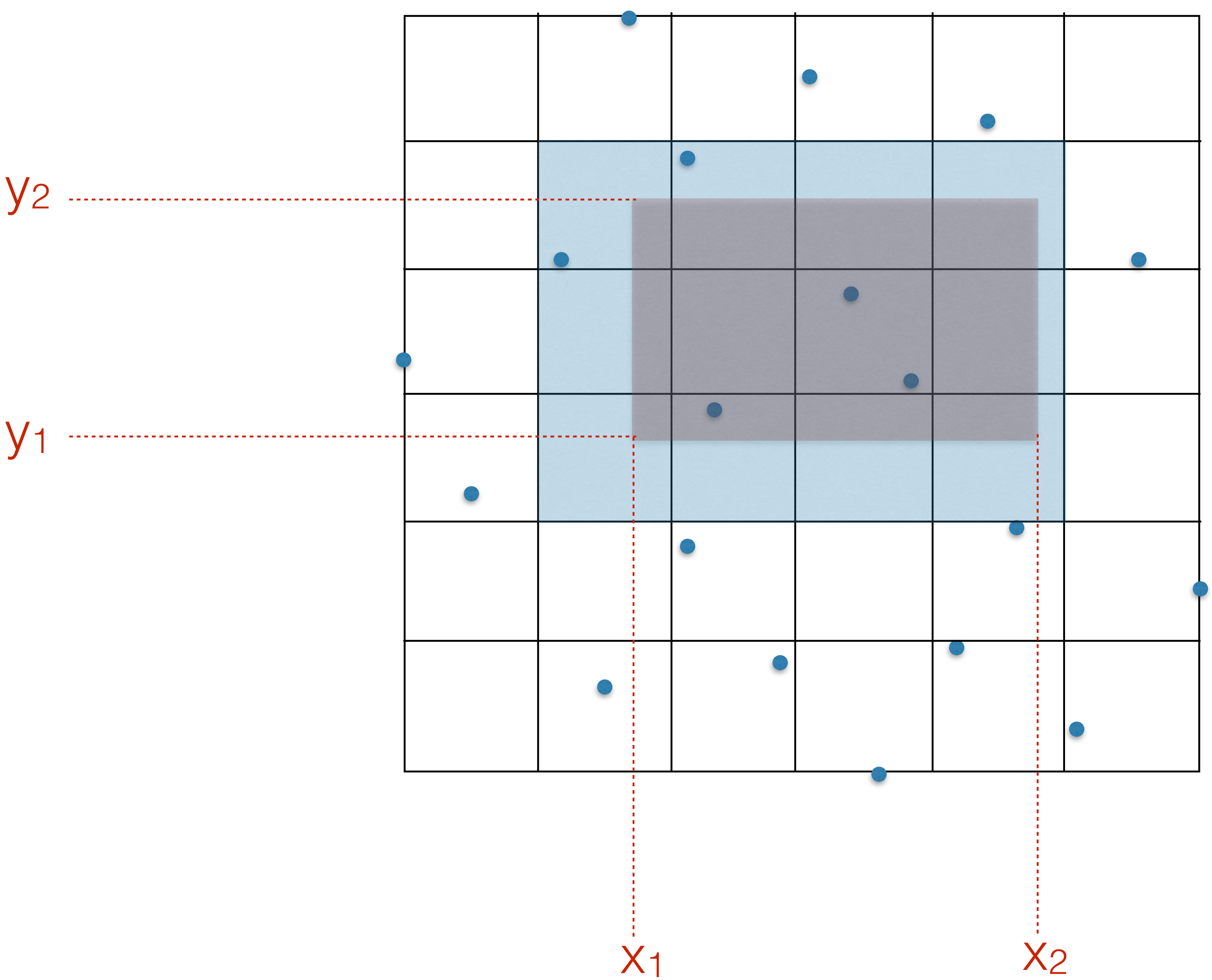
How do we answer range searches with a grid?



How do we answer range searches with a grid?



How do we answer range searches with a grid?



Analysis

- How long does a range query take ?
- How many points in a cell?
- How do the points look like for the worst-case to be good?
- How to chose m?

The grid method

- \+ Grids perform well if points are uniformly distributed
- \+ Grids can be used as heuristic for many other problems besides range searching (e.g. closest pair, neighbor queries)
- \- No worst case guarantees
- \+ simple to implement

We've seen this!

2d search trees

3d search trees

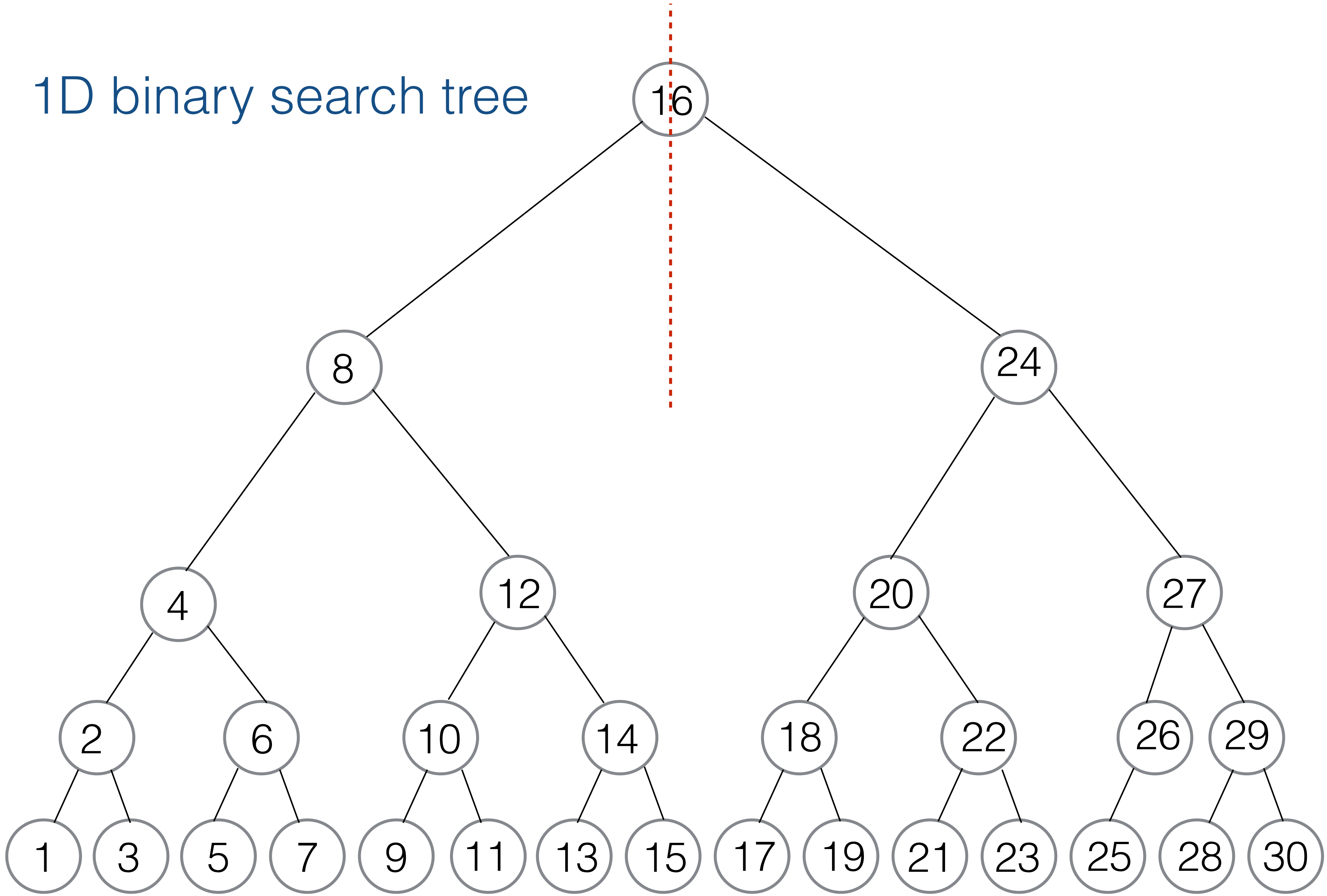
4d search trees

…

k-dimensional search trees

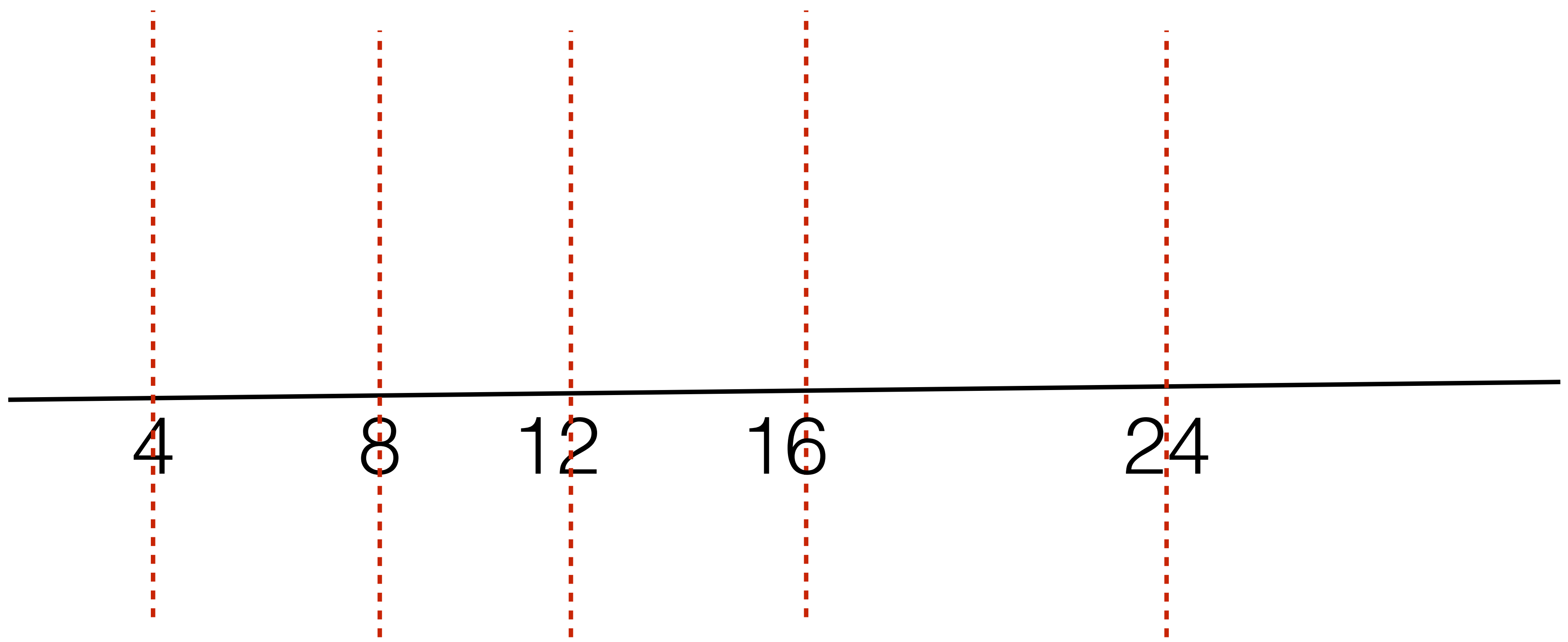
kd trees

1D binary search tree



a space decomposition: left tree represents all values <=16; right tree represents all values >16; and so on

1D binary search tree

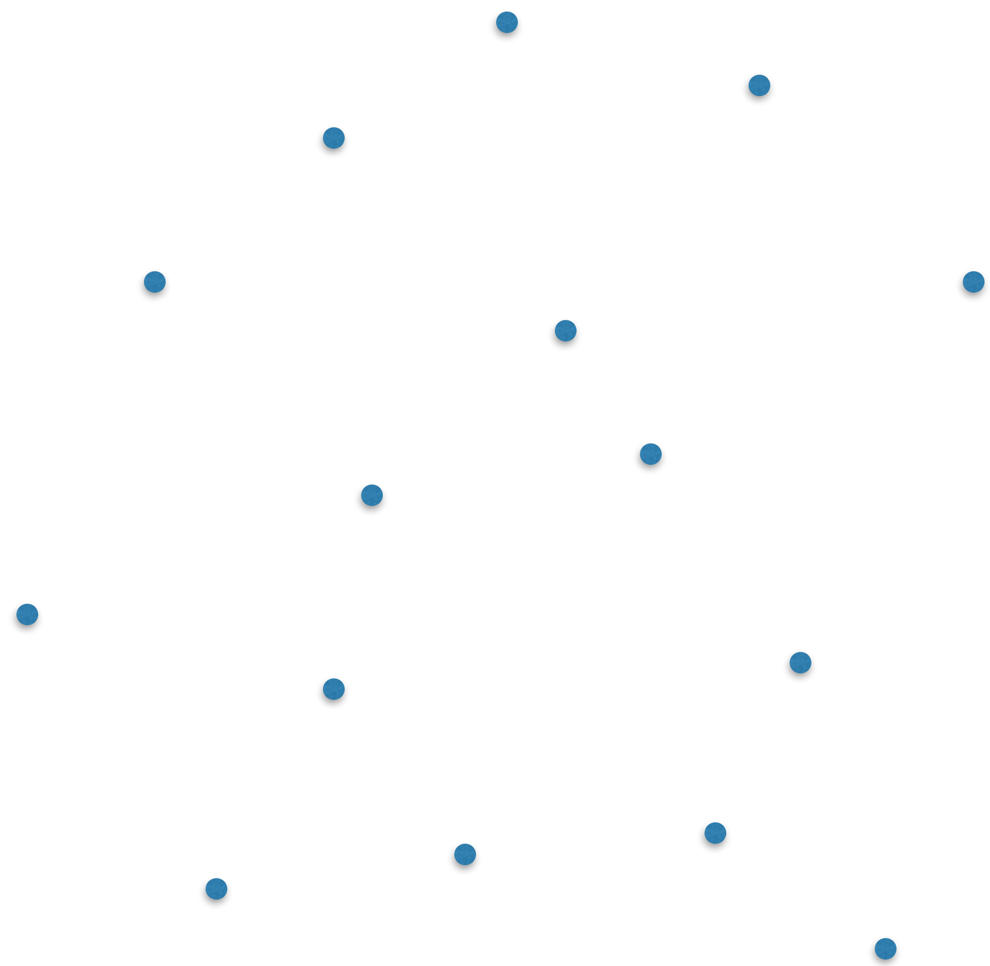


- to search for a value, find the region of space where it would be if it were in the input

2d binary search trees

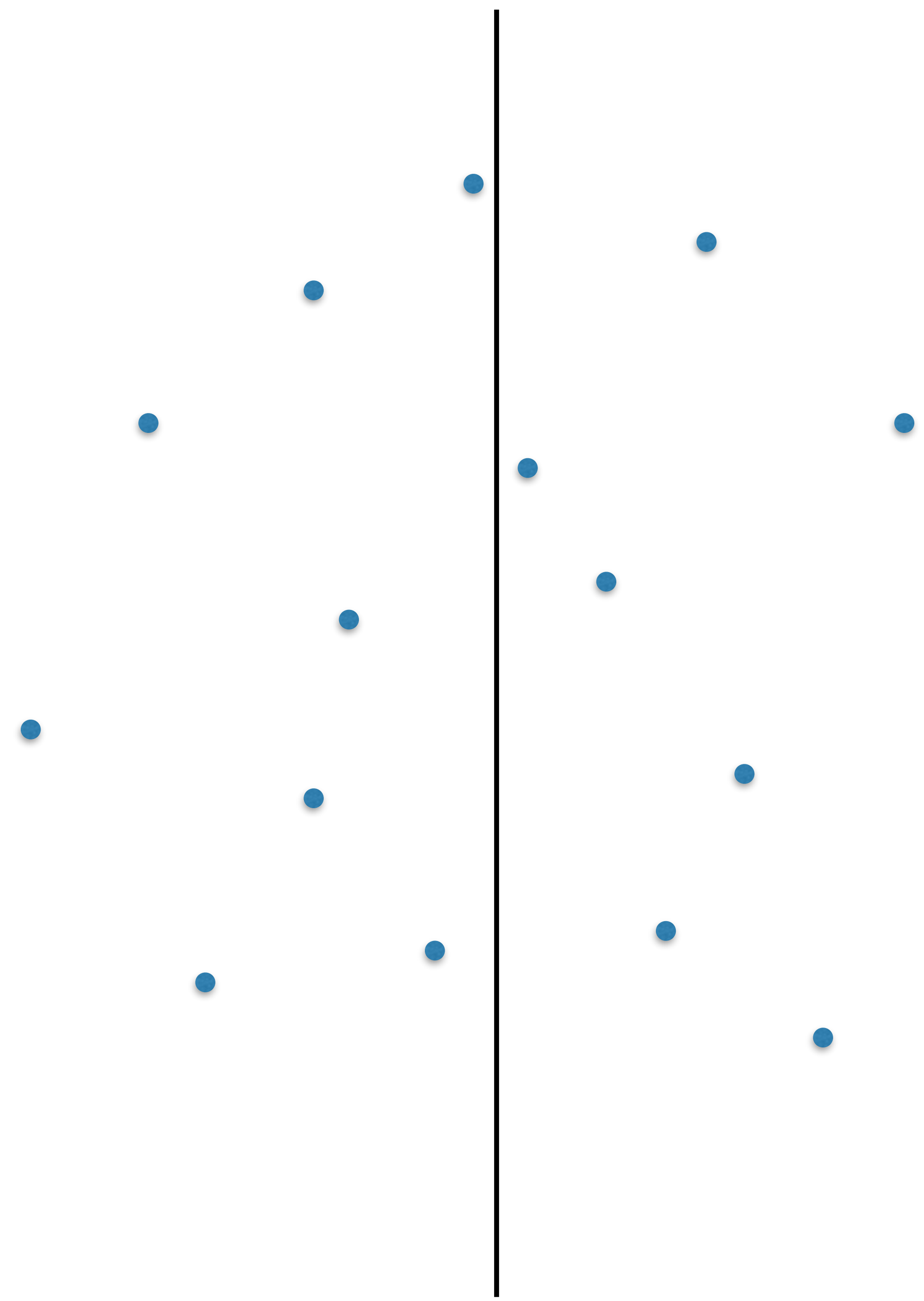
- **The idea:** A binary tree which recursively subdivides the plane by vertical and horizontal cut lines
- Vertical and horizontal lines alternate
- Cut lines are chosen to split the points in two (==> logarithmic height)

2d binary search trees



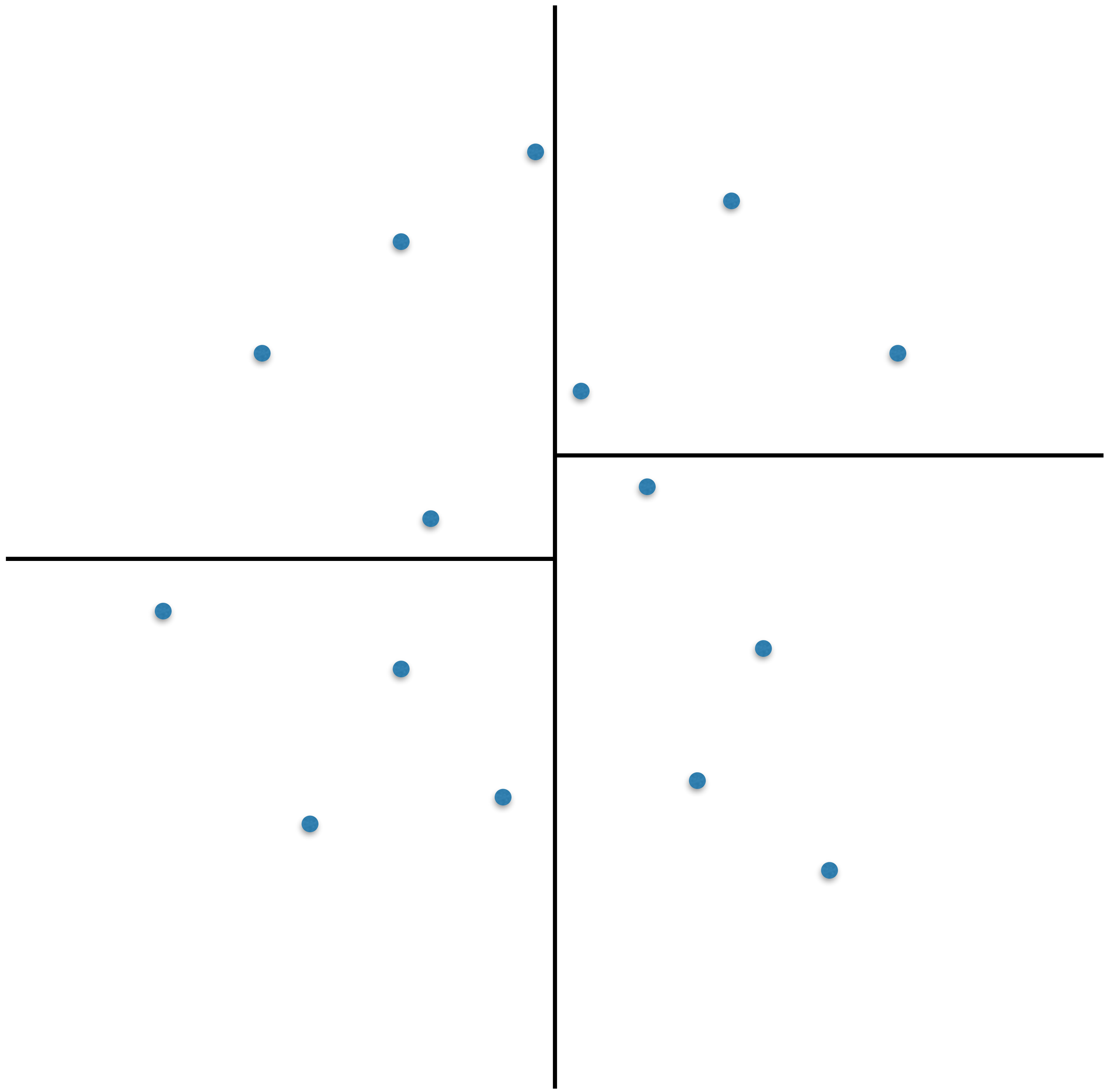
2d binary search trees

split points in two halves with a vertical line

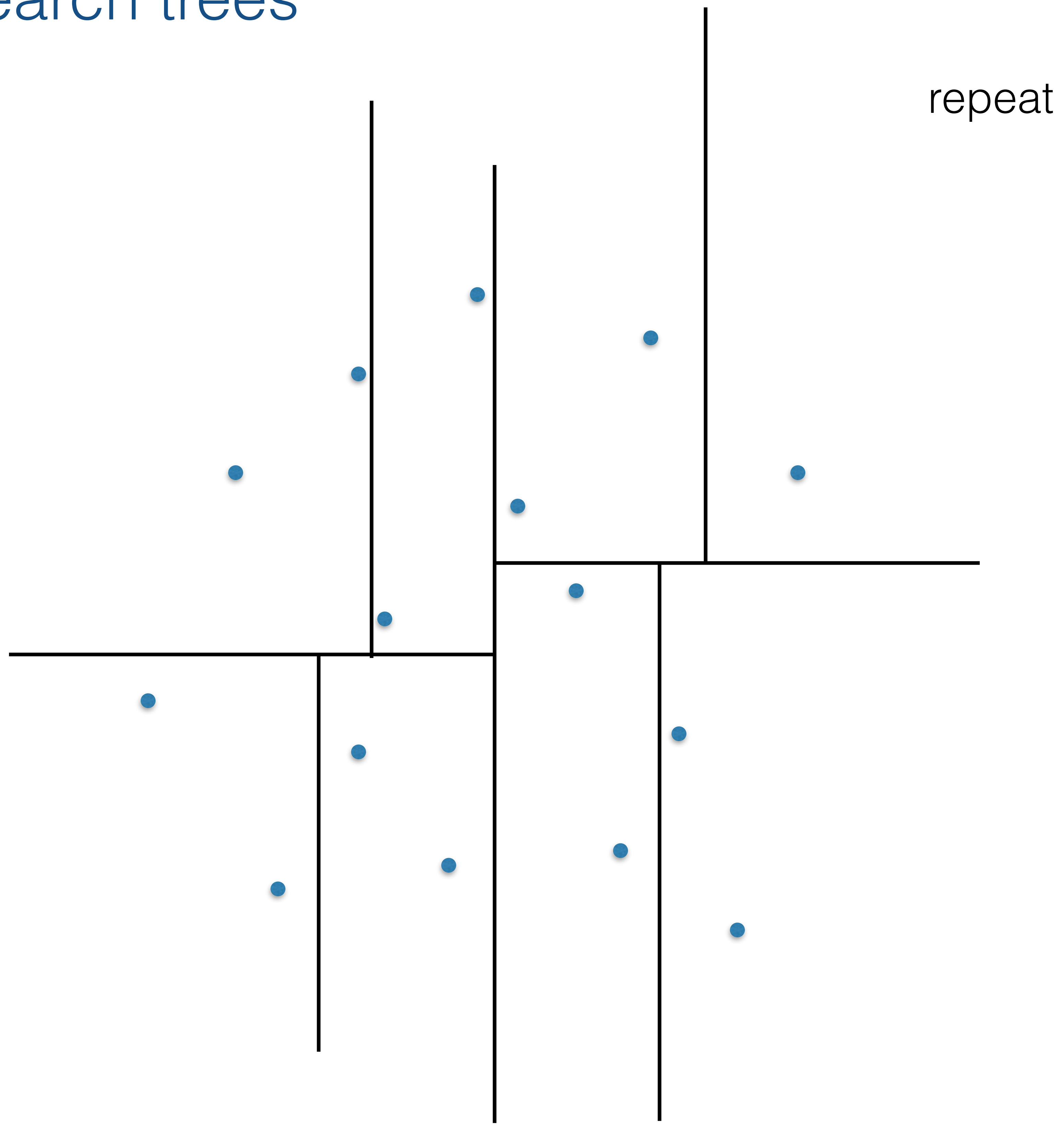


2d binary search trees

split each side into half with a horizontal line

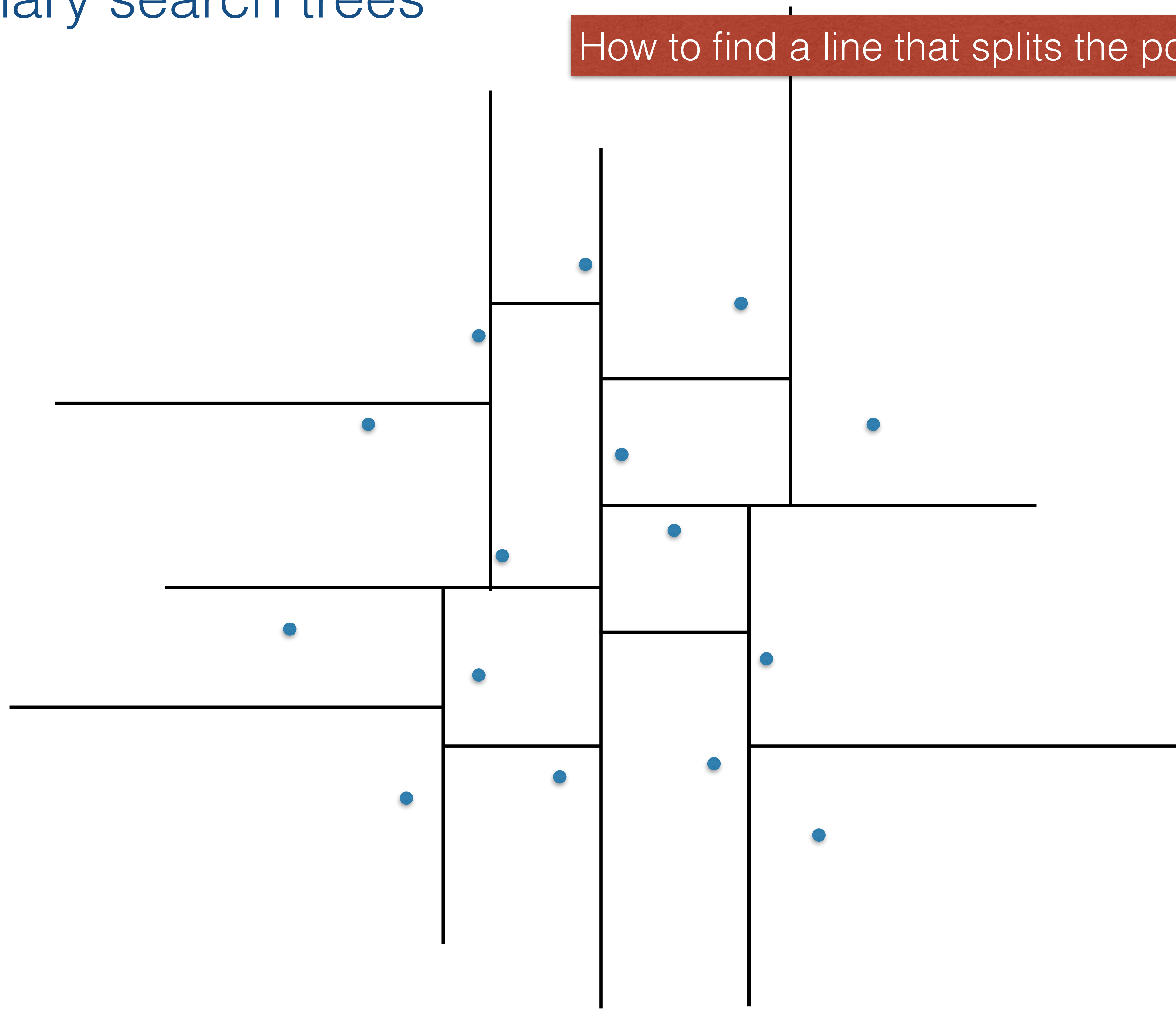


2d binary search trees



2d binary search trees

How to find a line that splits the points in half?



2d binary search trees

How to find a line that splits the points in half?

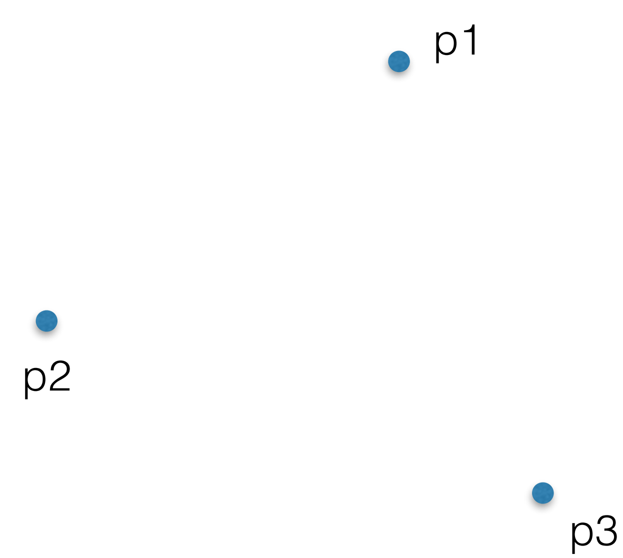
Variants:

- 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 that it falls in between the points. Internal nodes store lines, and points are only in leaves.
- Choose the cut line so it goes through the median point. Internal nodes store lines, and points are only in leaves.
    - if n is even, assign the median to the e.g. smaller (left/below) one, consistently

This is standard and simplifies the details

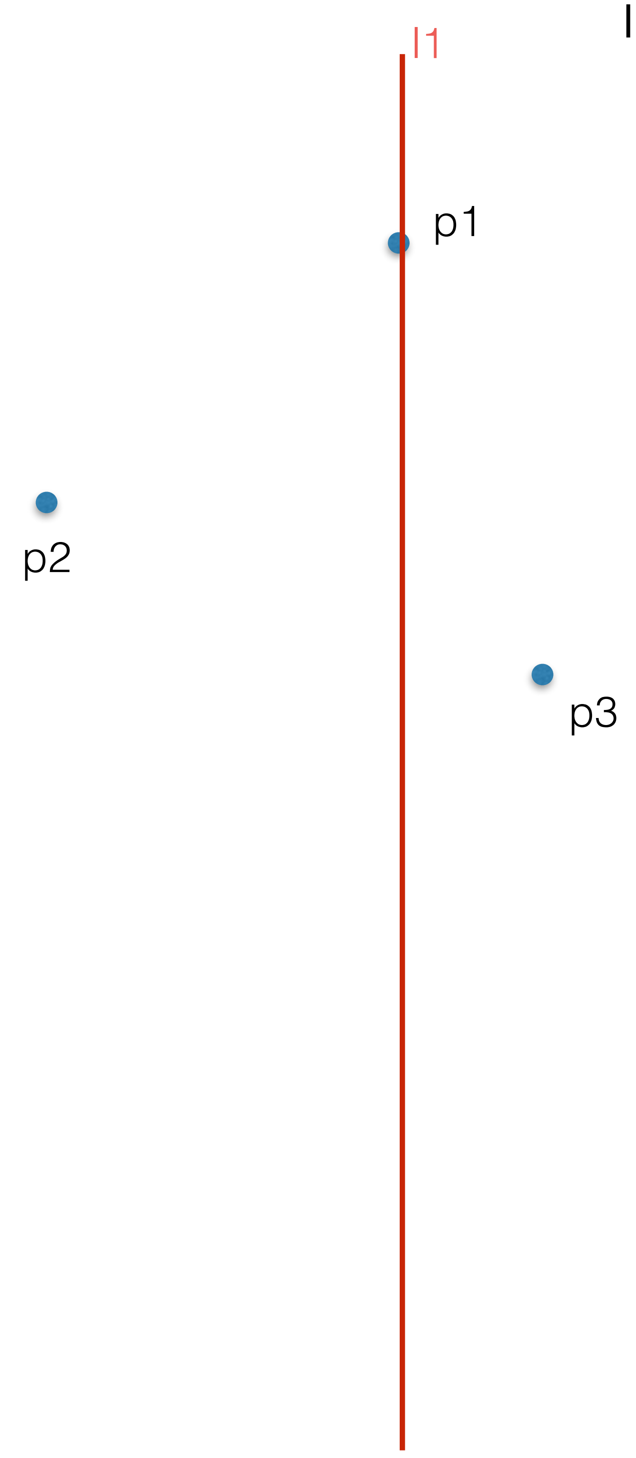
Let’s see what this means

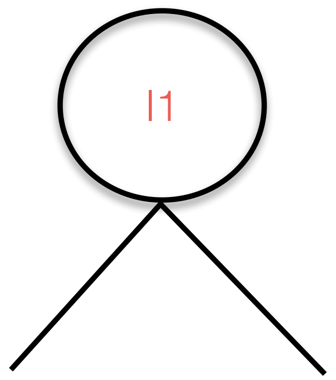
2d binary search trees



2d binary search trees

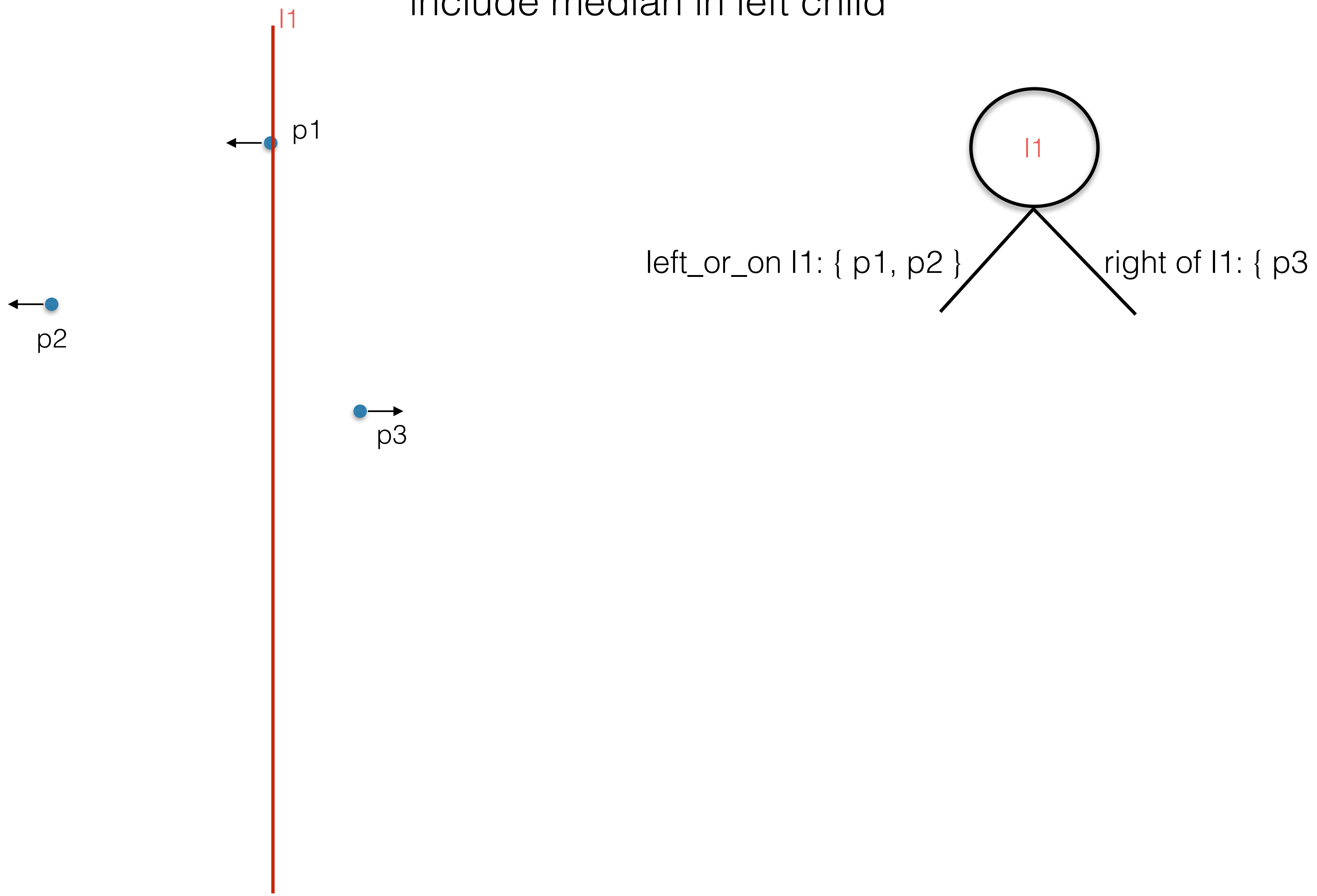
split with vertical line through x-median
include median in left child





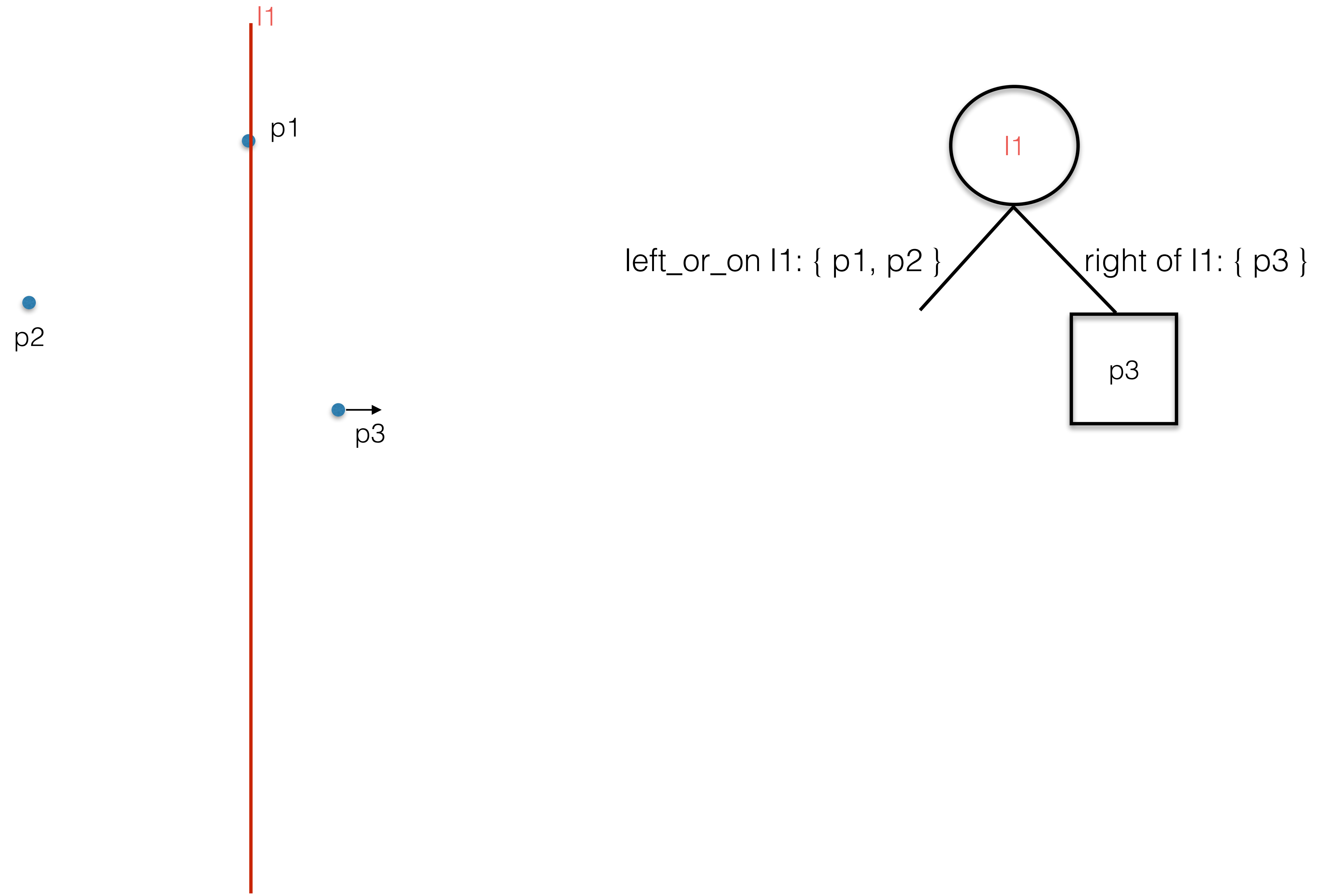
2d binary search trees

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



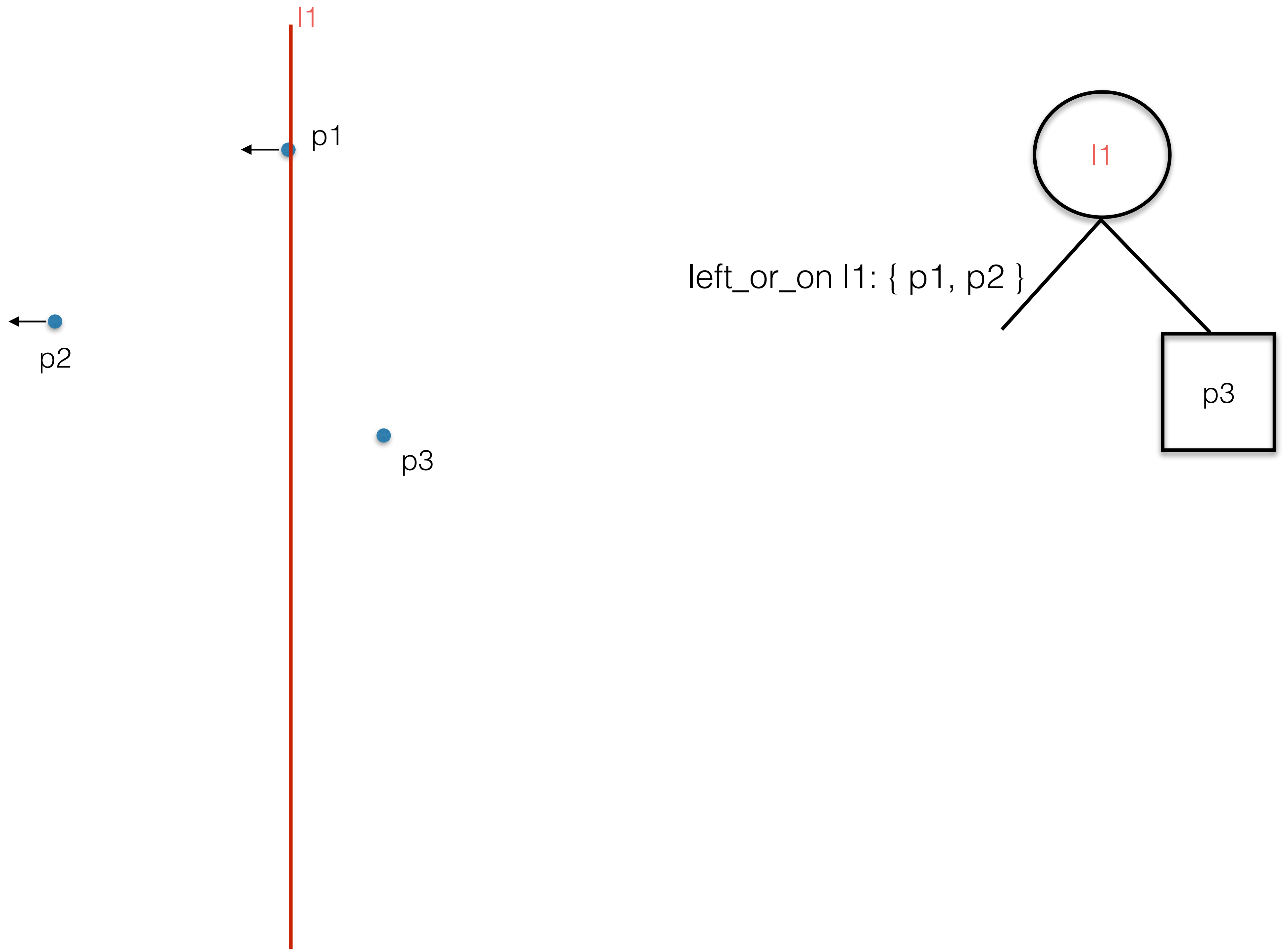
2d binary search trees

right of l1: p3 => leaf



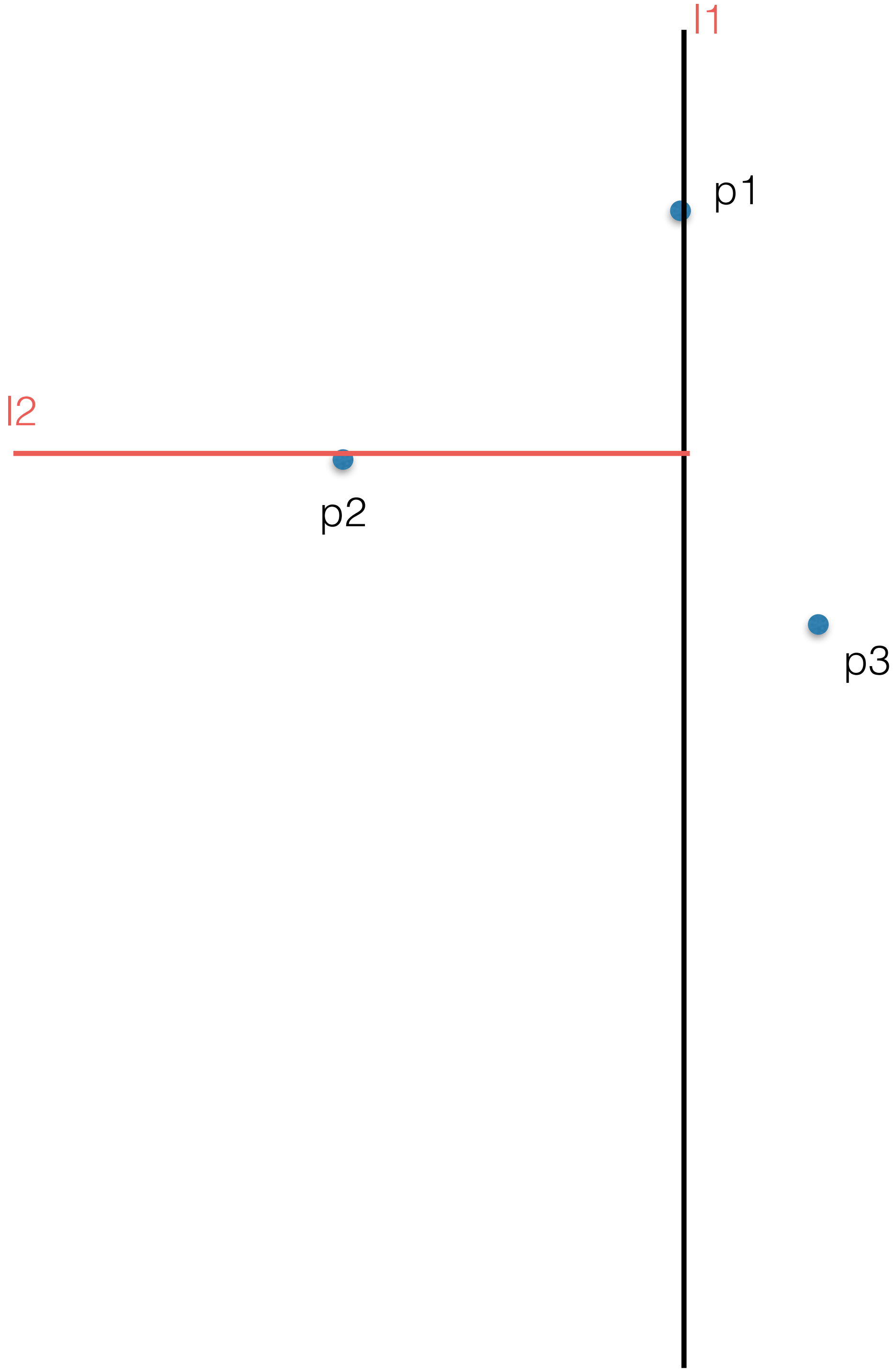
2d binary search trees

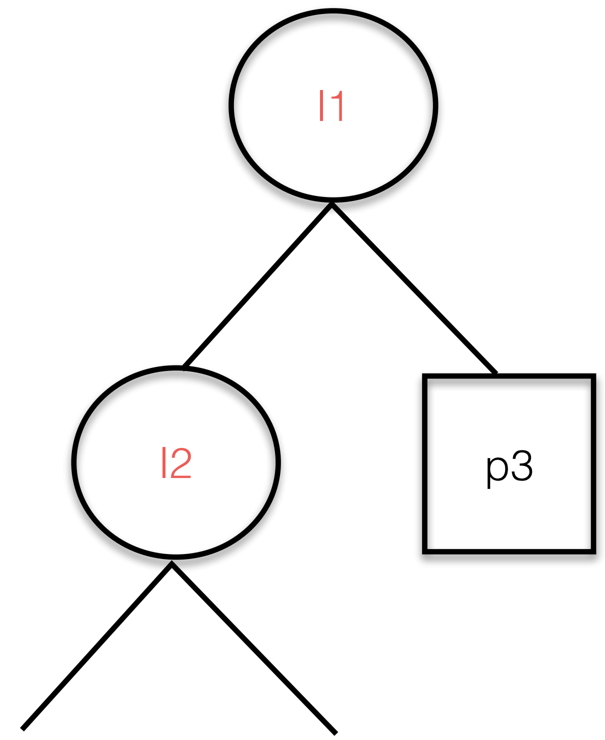
left_on l1: p1,p2 => recurse



2d binary search trees

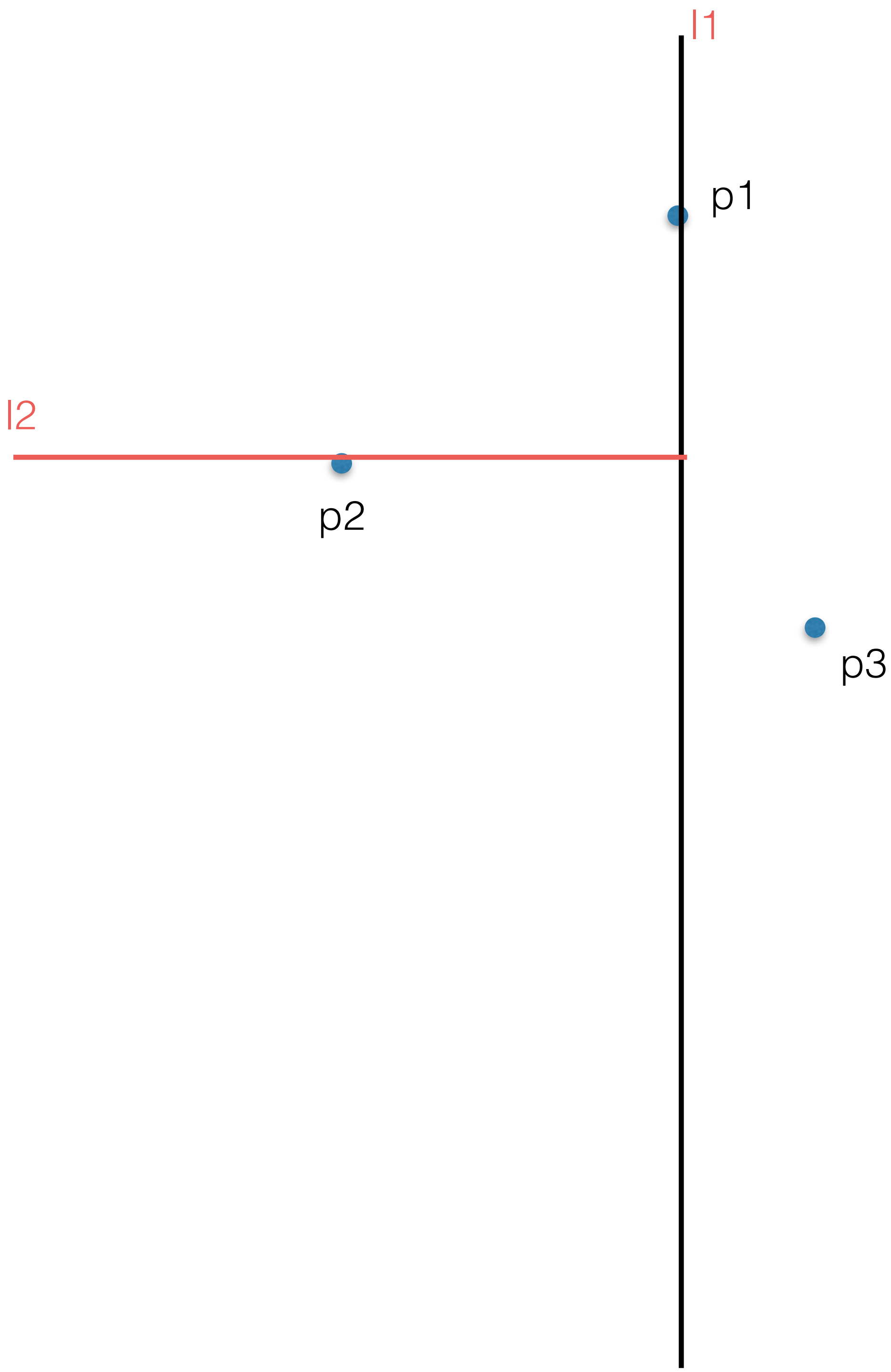
split with horizontal line through y-median
include median in left child

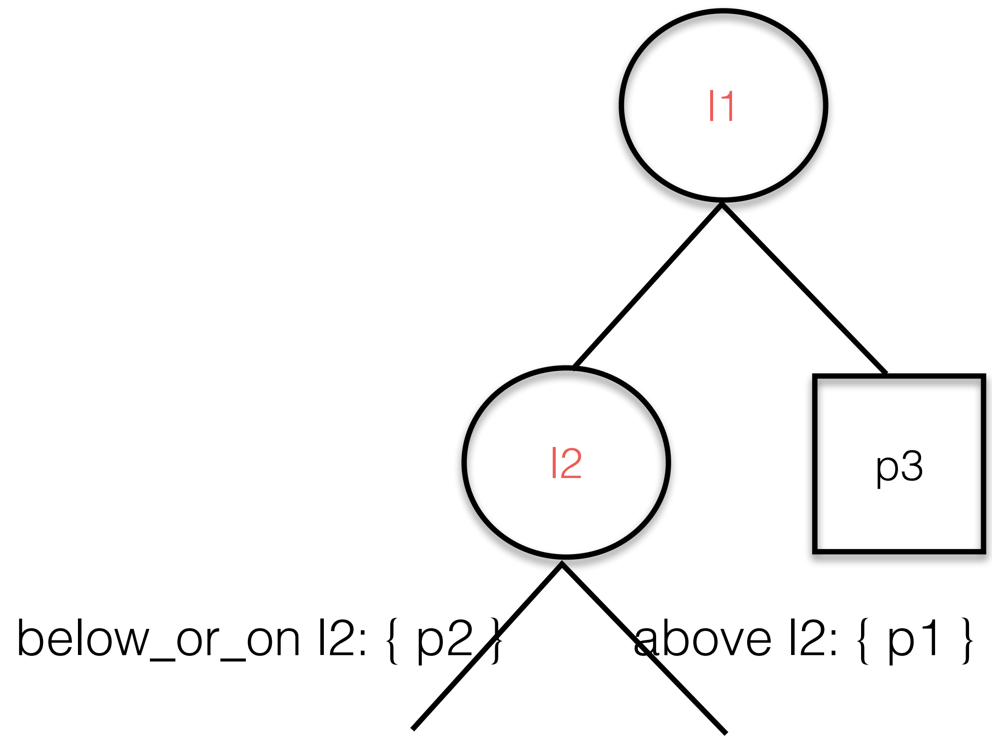




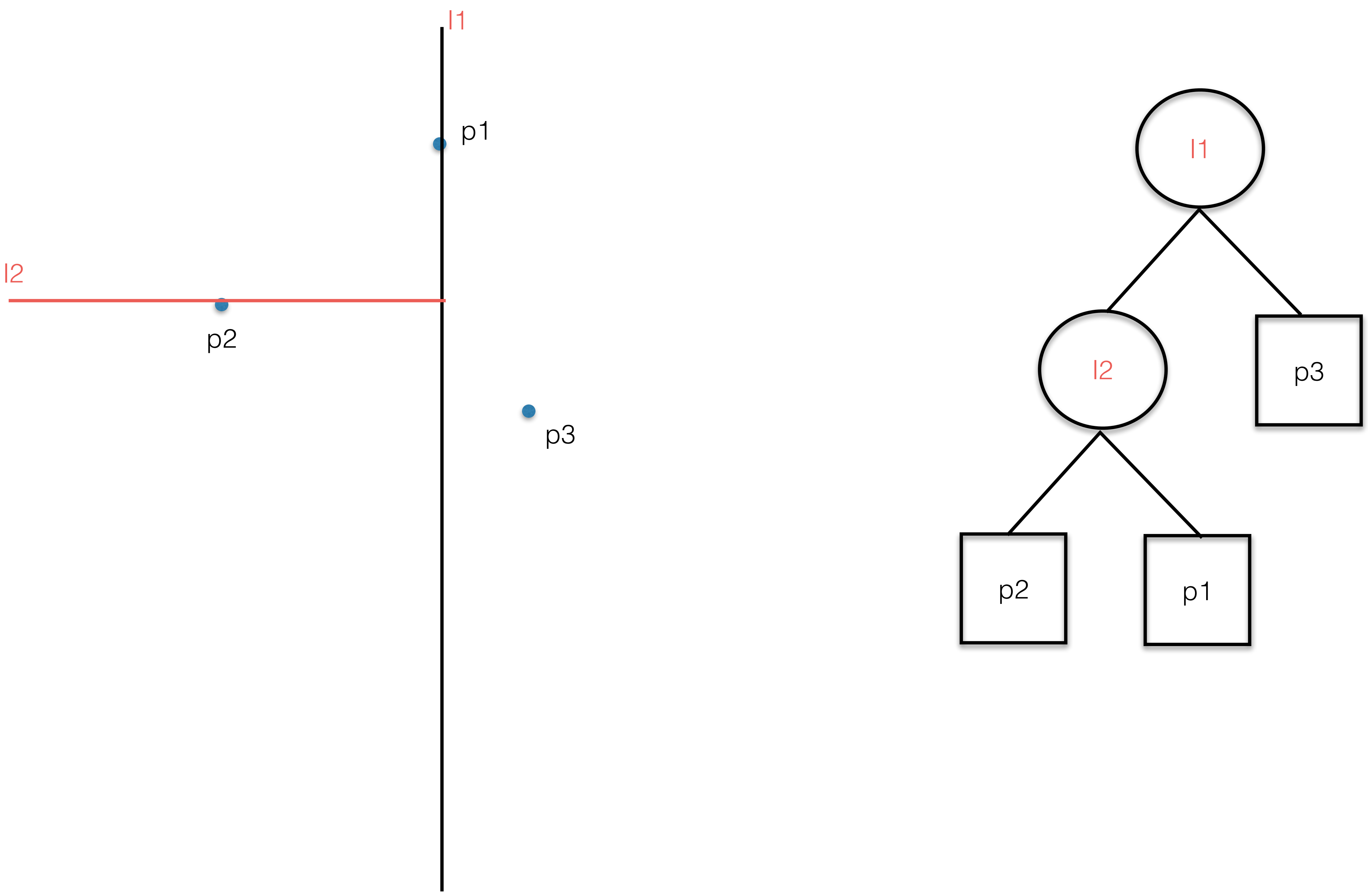
2d binary search trees

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

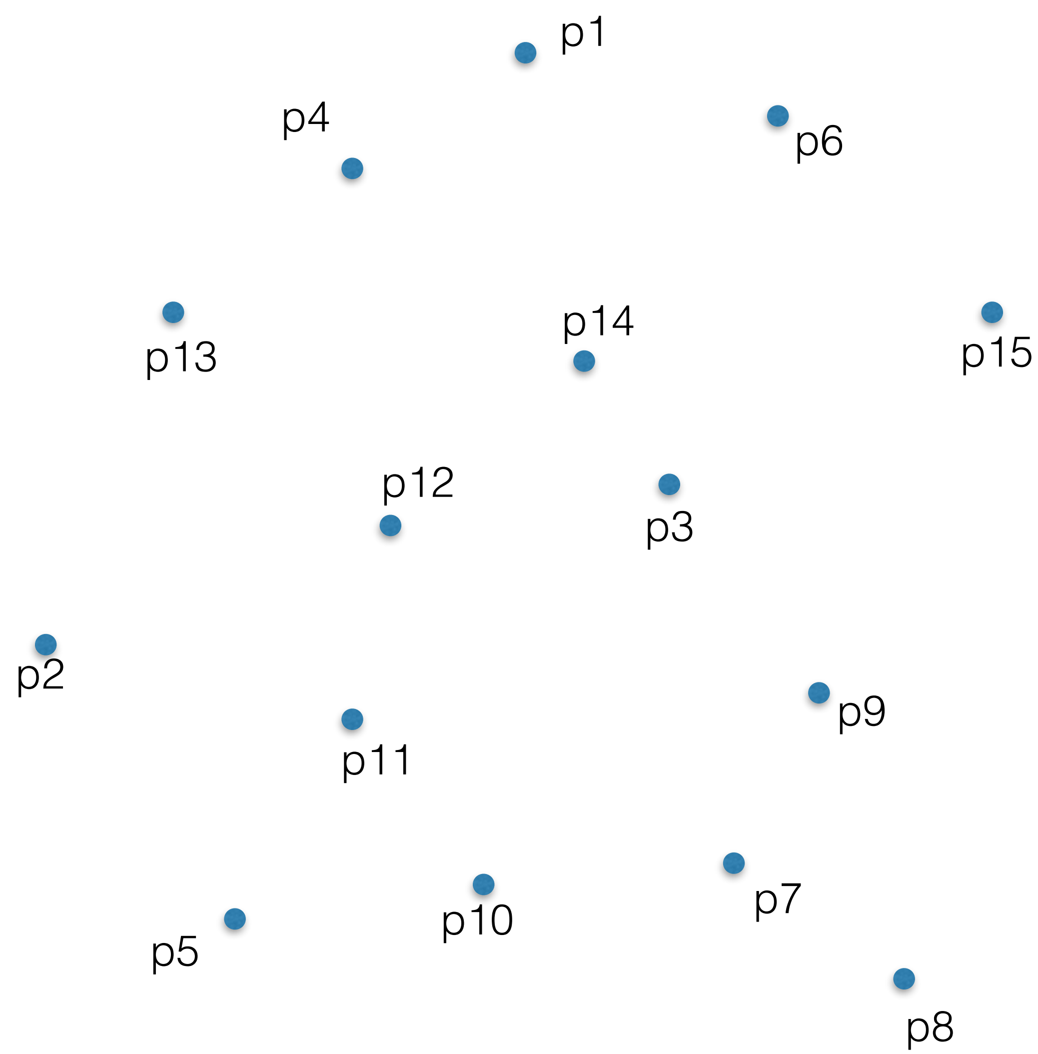




2d binary search trees

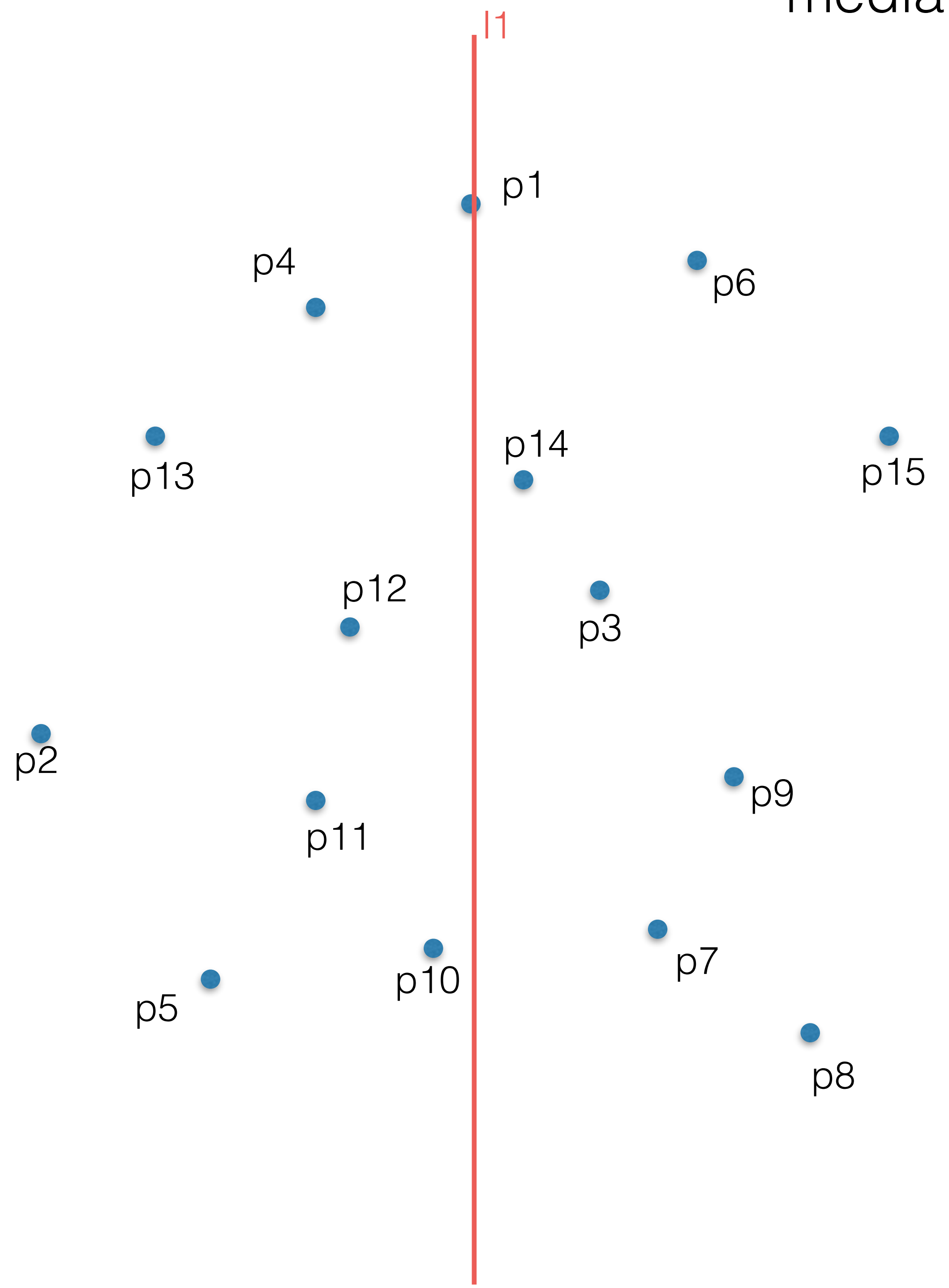


A bigger example



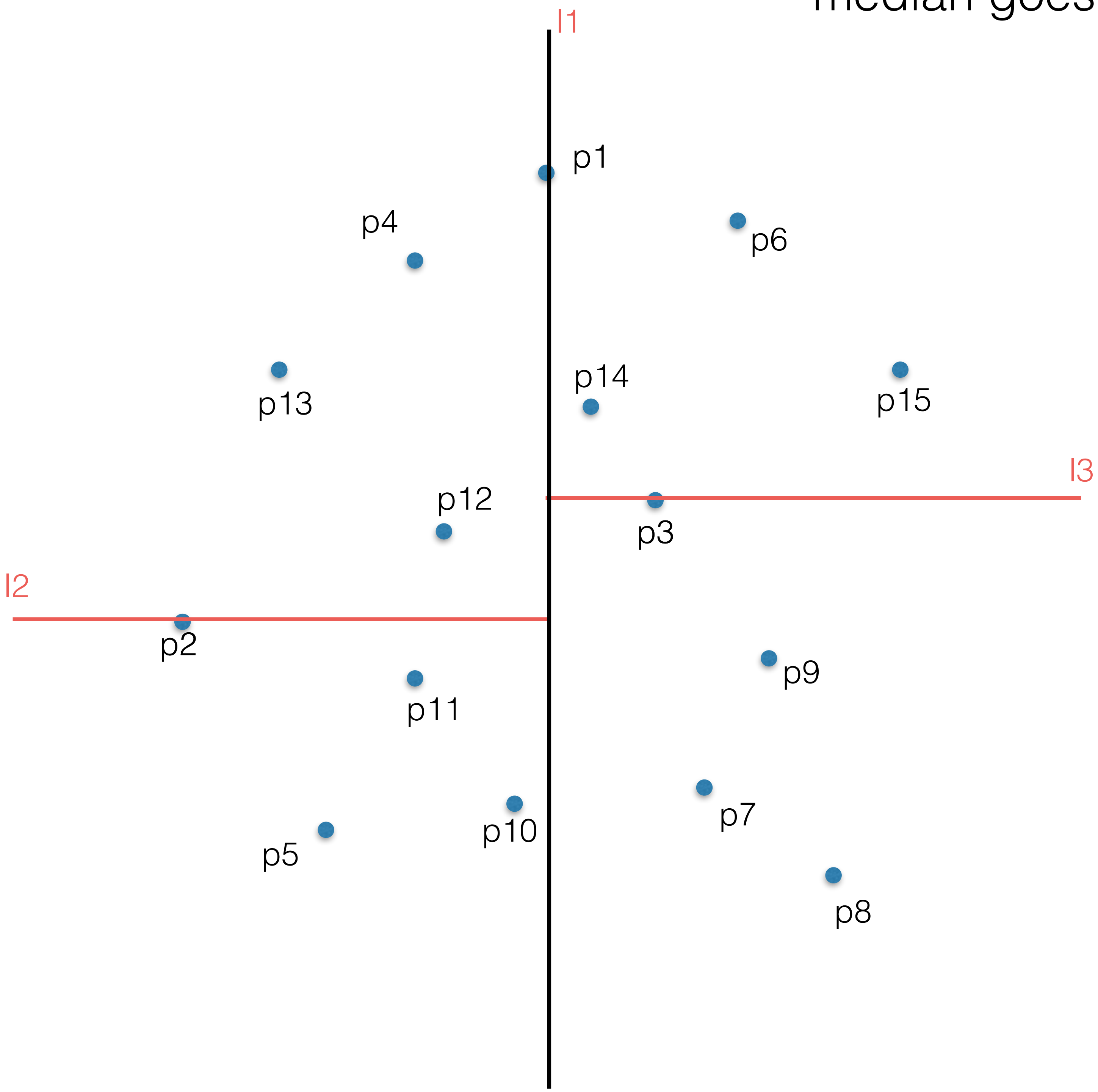
A bigger example

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



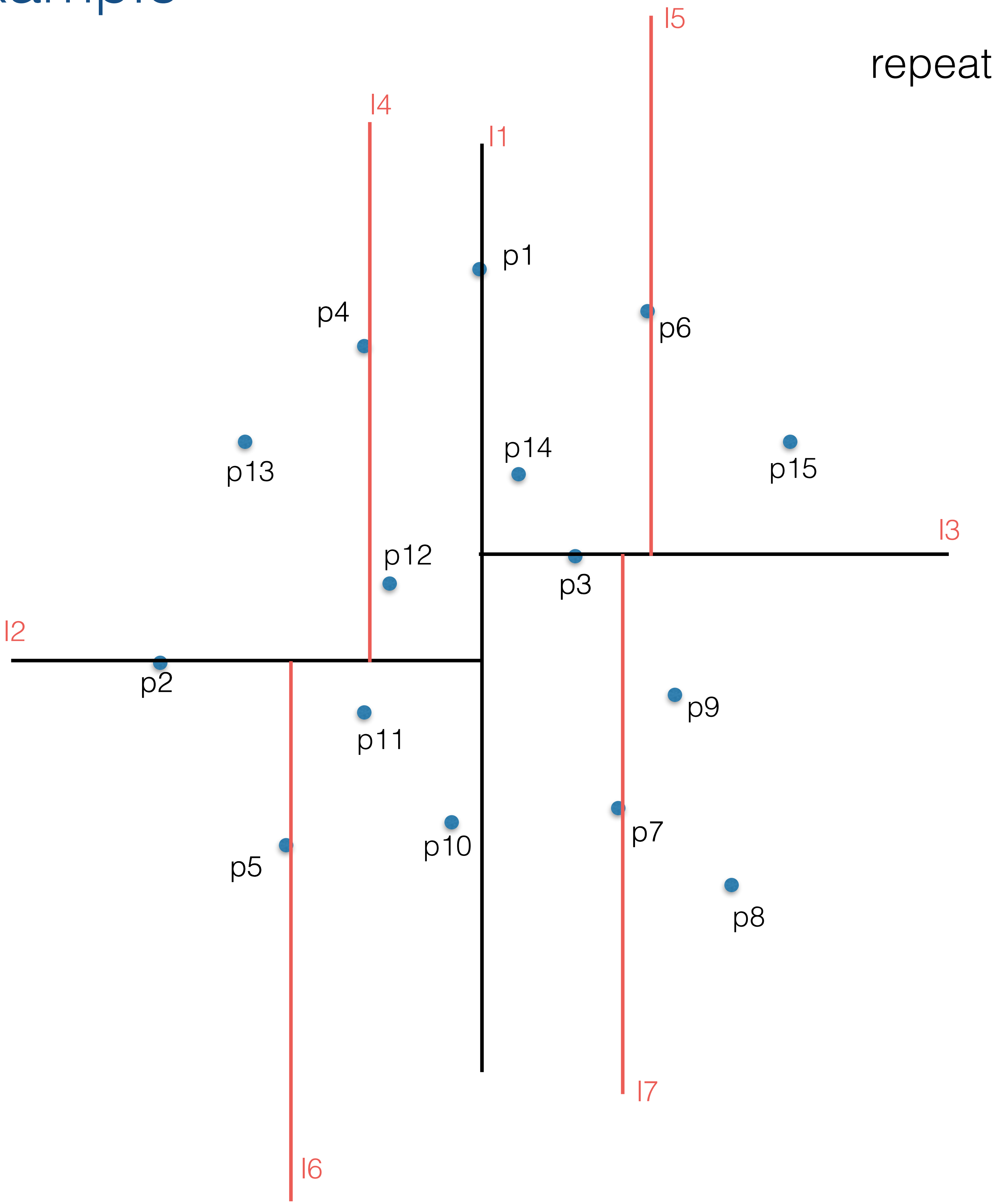
A bigger example

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

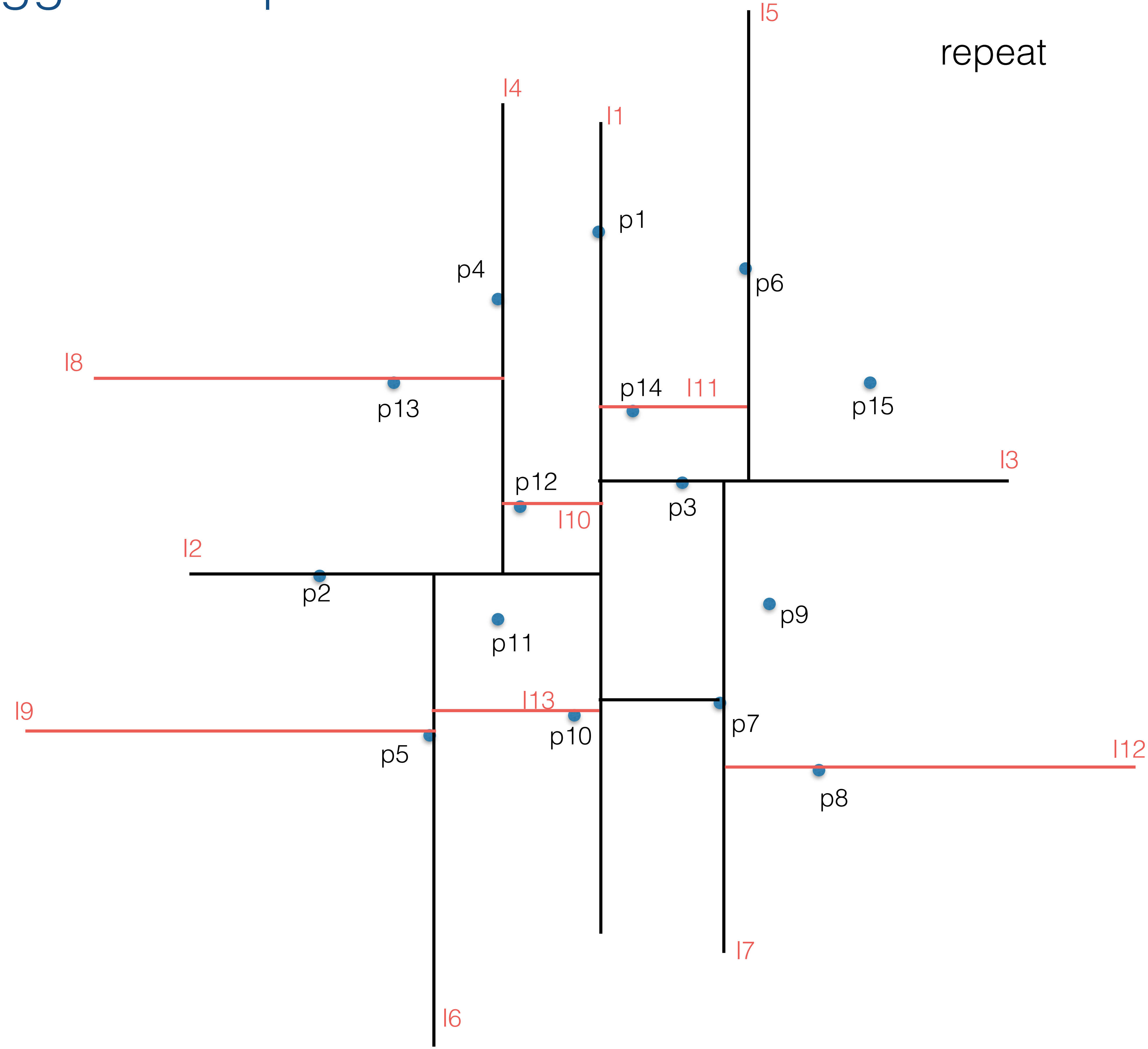


A bigger example

repeat

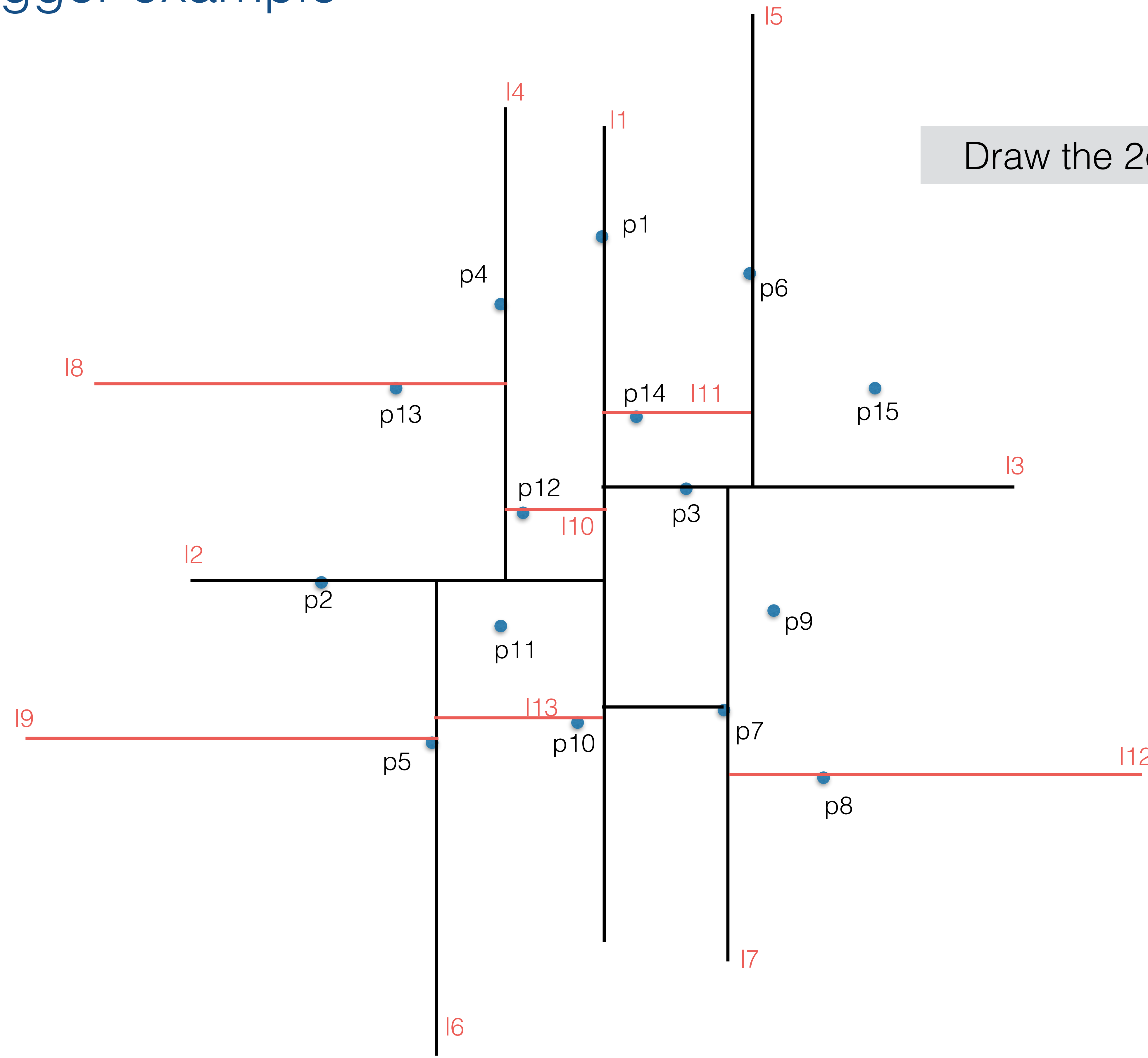


A bigger example



A bigger example

Draw the 2d-tree!



2d binary search trees

Analysis

1. How to build it and how fast?
2. How much space does it take?
3. How to answer range queries and how fast?

2d binary search trees construction

Algorithm $\textsc{BuildKdTree}(P, depth)$

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. $\nu_{\text{left}} \leftarrow \textsc{BuildKdTree}(P_1, depth + 1)$
7. $\nu_{\text{right}} \leftarrow \textsc{BuildKdTree}(P_2, depth + 1)$
8. Create a node ν storing ℓ, make ν_{left} the left child of ν, and make ν_{right} the right child of ν.
9. **return** ν

2d binary search trees construction

Analysis?

Algorithm $\textsc{BuildKdTree}(P, \mathit{depth})$

if P contains only one point
  then return a leaf storing this point
  else if depth is even
    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)
    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)
  $v_{\text{left}} \leftarrow \textsc{BuildKdTree}(P_1, \mathit{depth}+1)$
  $v_{\text{right}} \leftarrow \textsc{BuildKdTree}(P_2, \mathit{depth}+1)$
  Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
  return v

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

2d binary search trees construction

1. How to build it and how fast?

   - Let $T(n)$ be the time needed to build a 2d tree of n points
   - Then

     $$T(n) = 2T(n/2) + O(n)$$

   - This solves to **$O(n \lg n)$**

- Practical notes
  - The $O(n)$ median finding algorithm is not practical. Either do a randomized median finding (QuickSelect); or
  - Better: pre-sort P on x- and y-coord and pass them along as argument, and maintain the sorted sets through recursion

    P_1-sorted-by-x, P_1-sorted-by-y

    P_2-sorted-by-x, P_2-sorted-by-y

2d binary search trees

2. How much space does it take?

2d binary search trees

2. How much space does it take?

O(n)

2d binary search trees

3. How to answer range queries?

Let’s work through an example to get the intuition.

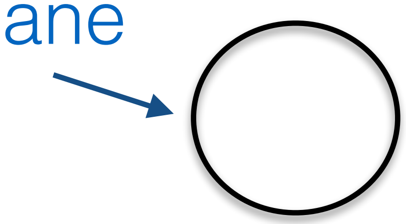
Range queries

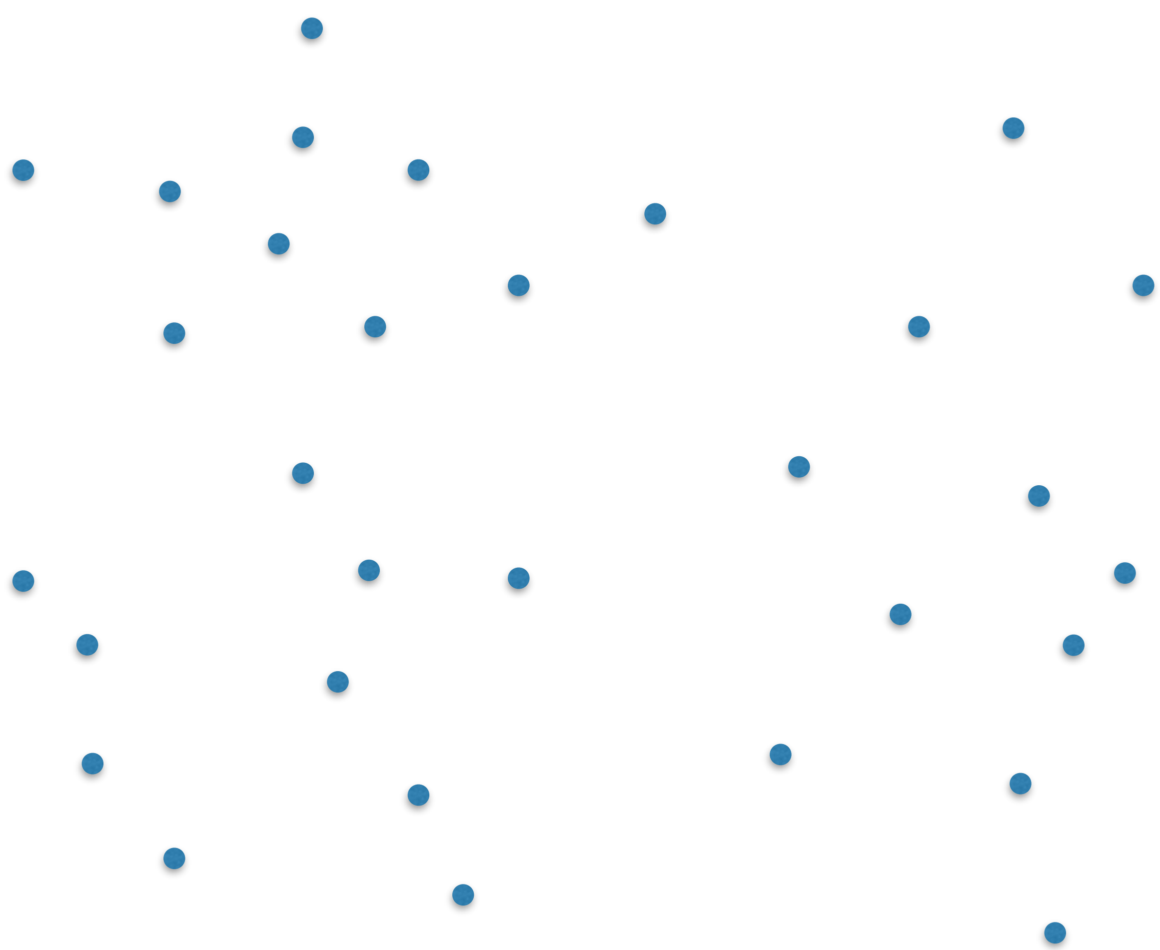
We'll use that:

A 2d-tree defines a hierarchical partition of the space, where each node in the tree represents a region of space.

Let's see what this means..

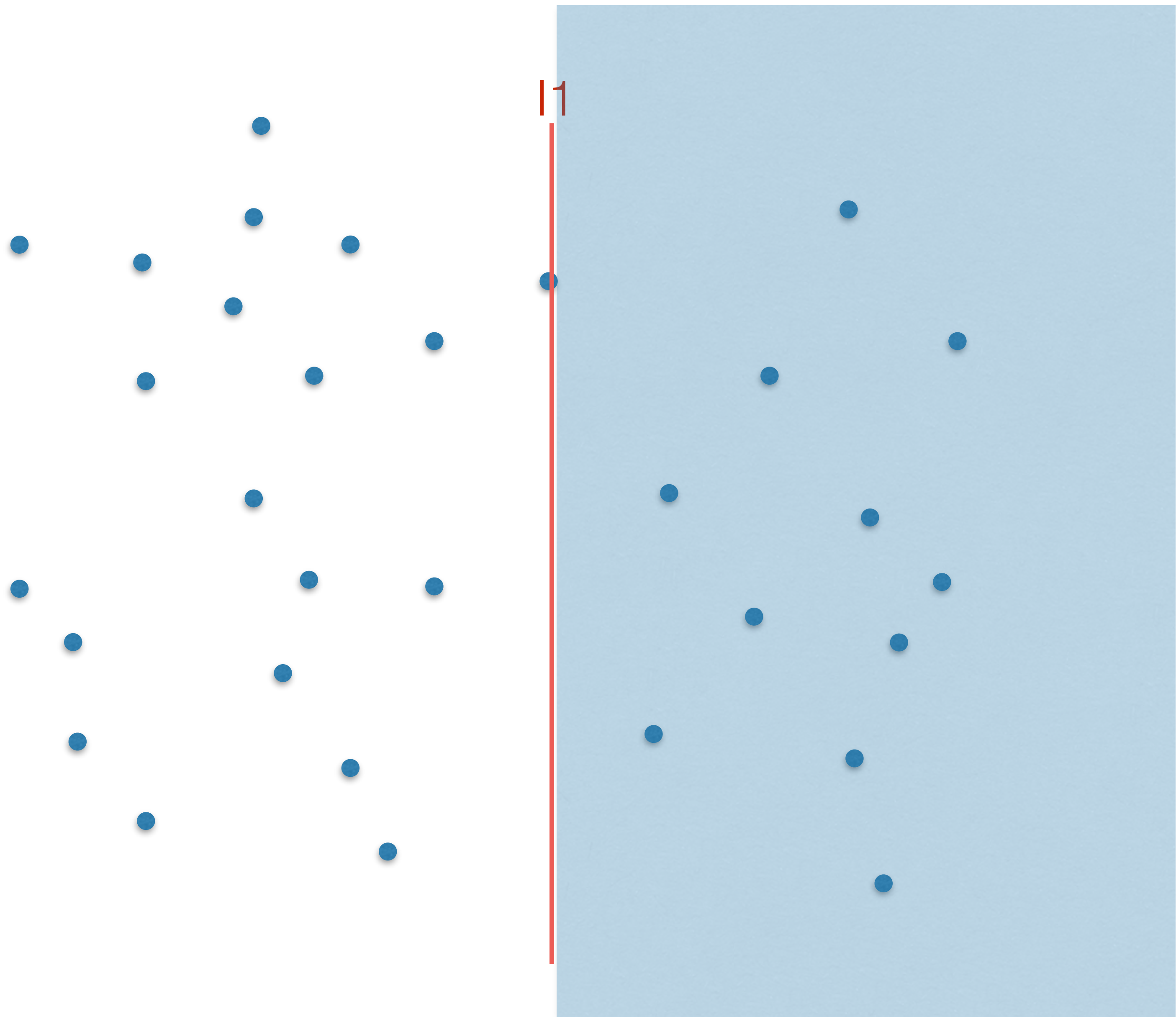
The **region** of a node

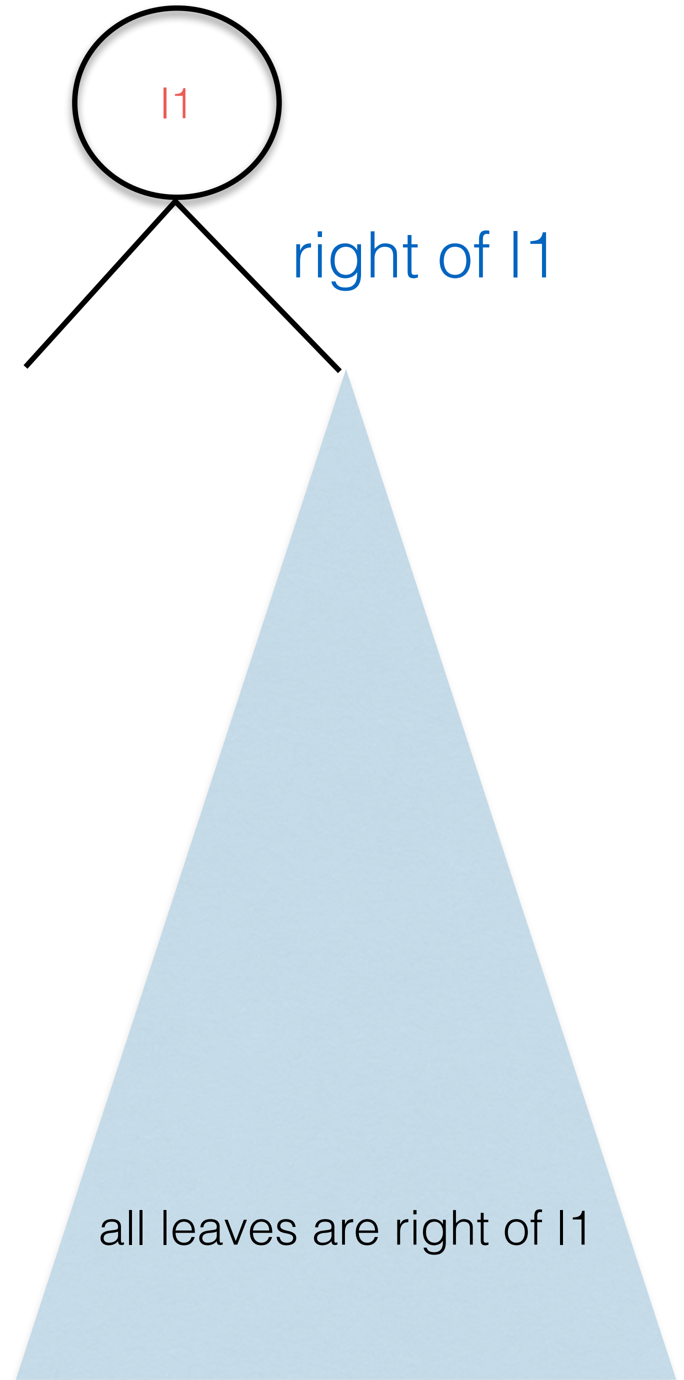




Each node in the tree corresponds to a region in the plane.

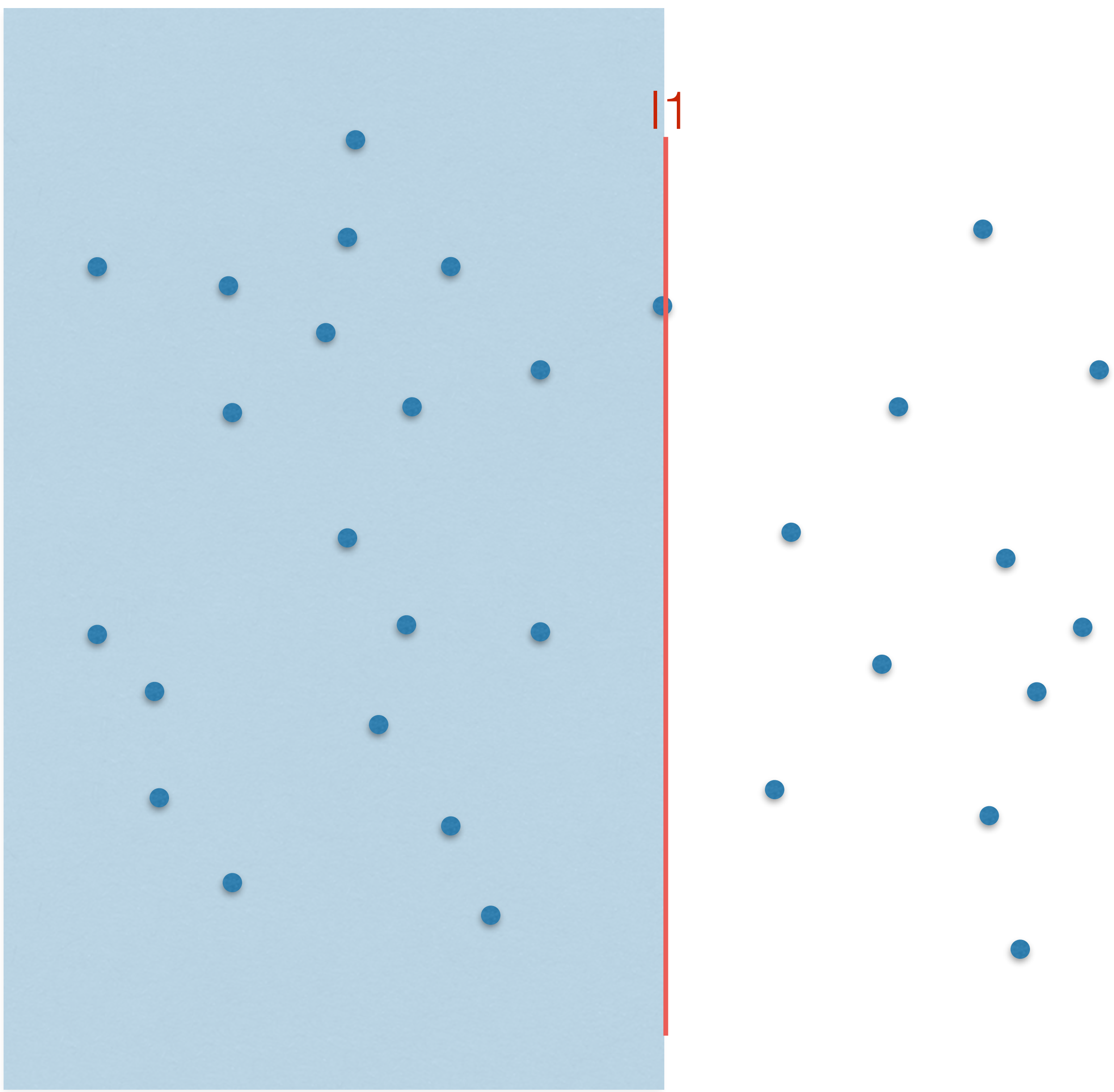
The **region** of a node

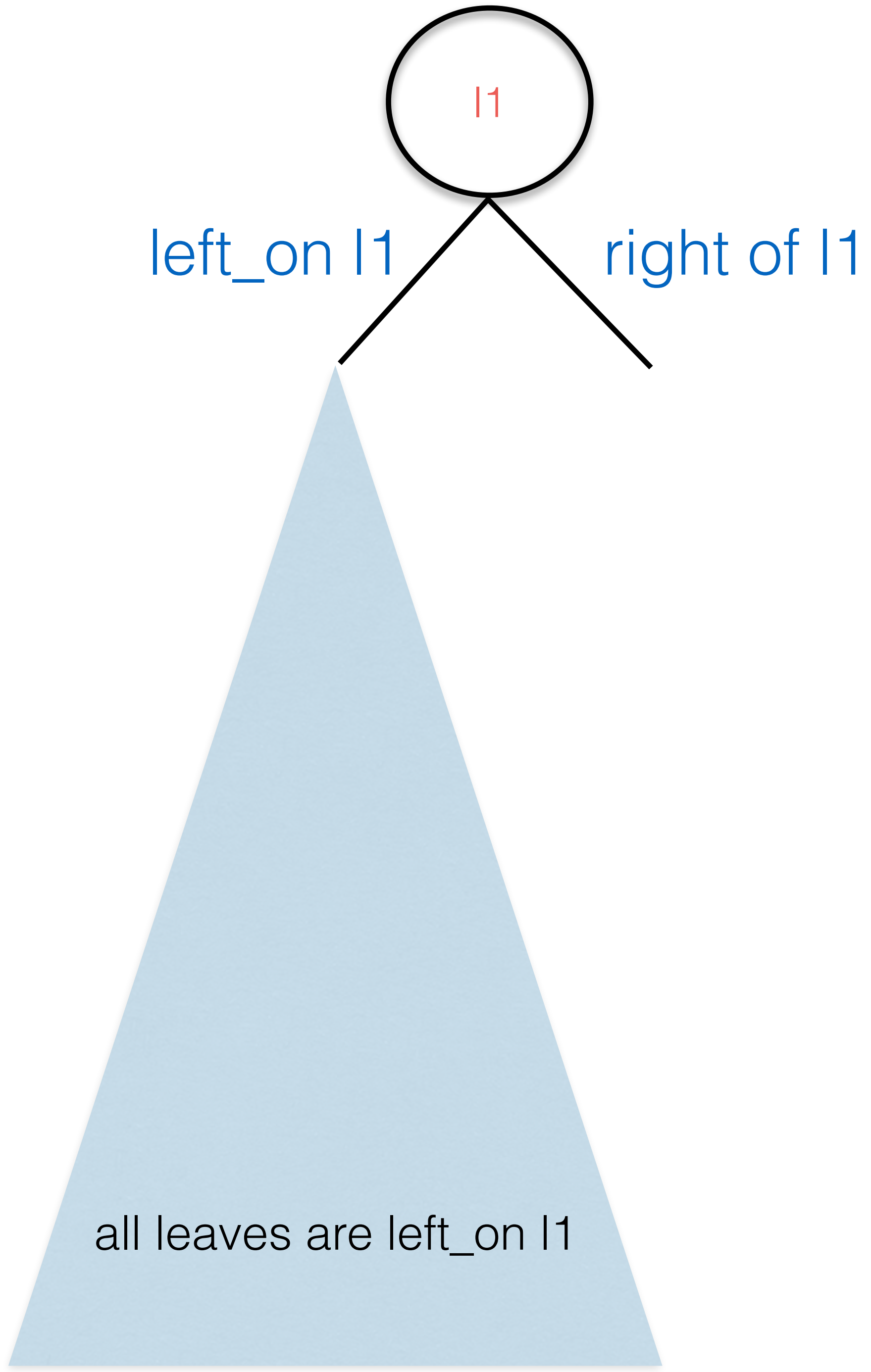




Each node in the tree corresponds to a region in the plane.

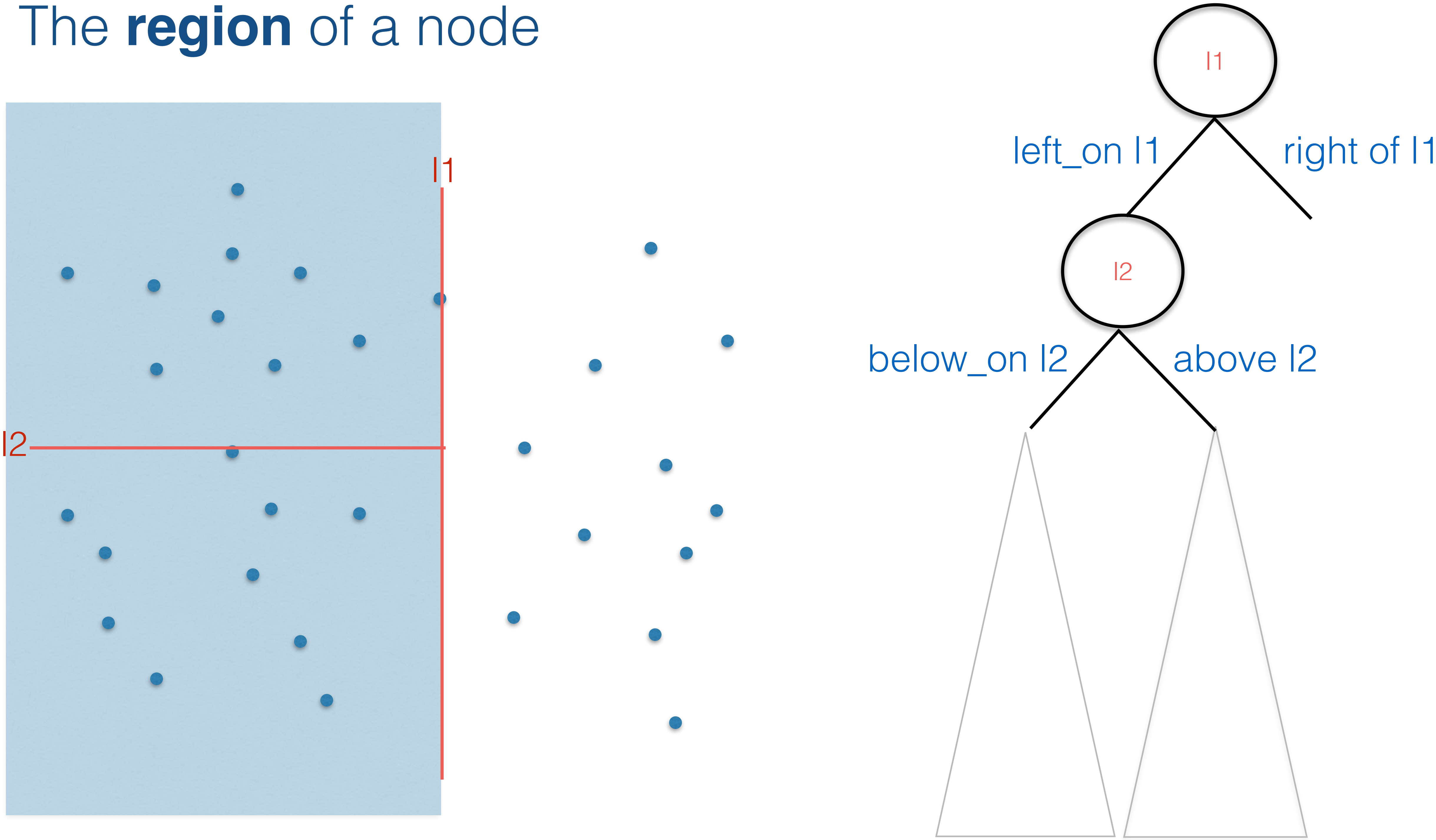
The **region** of a node





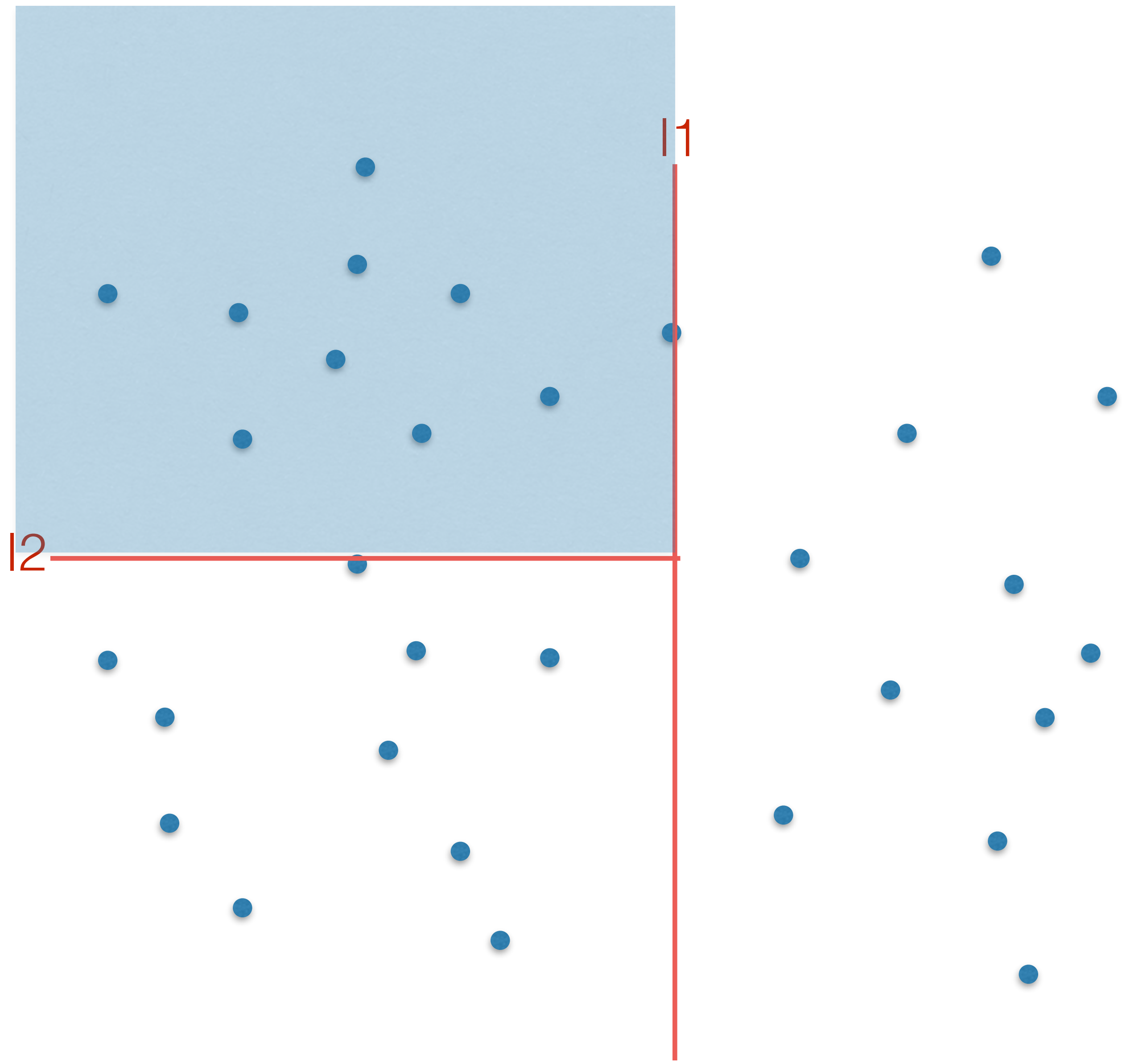
Each node in the tree corresponds to a region in the plane.

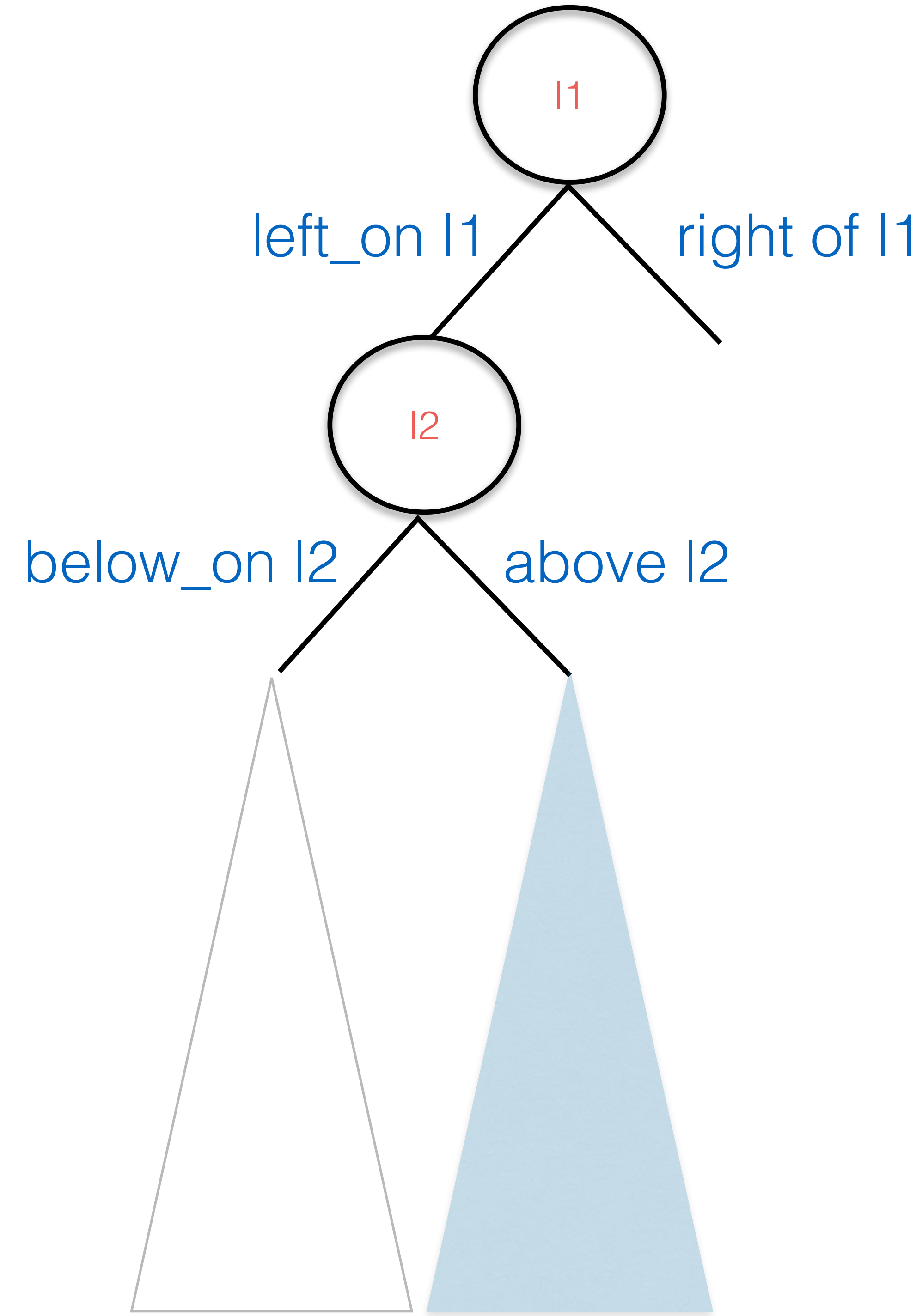
The **region** of a node



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

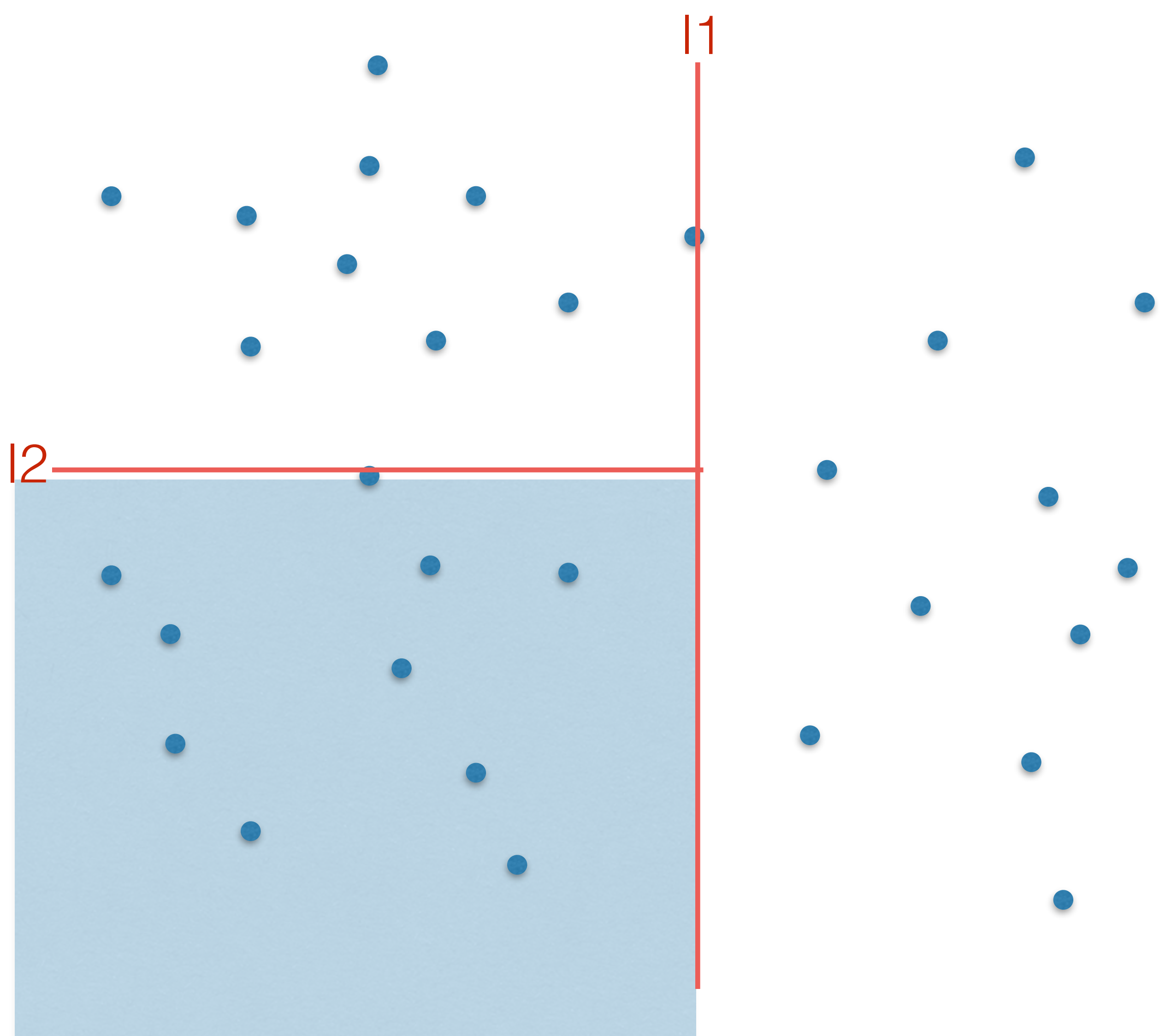
The **region** of a node

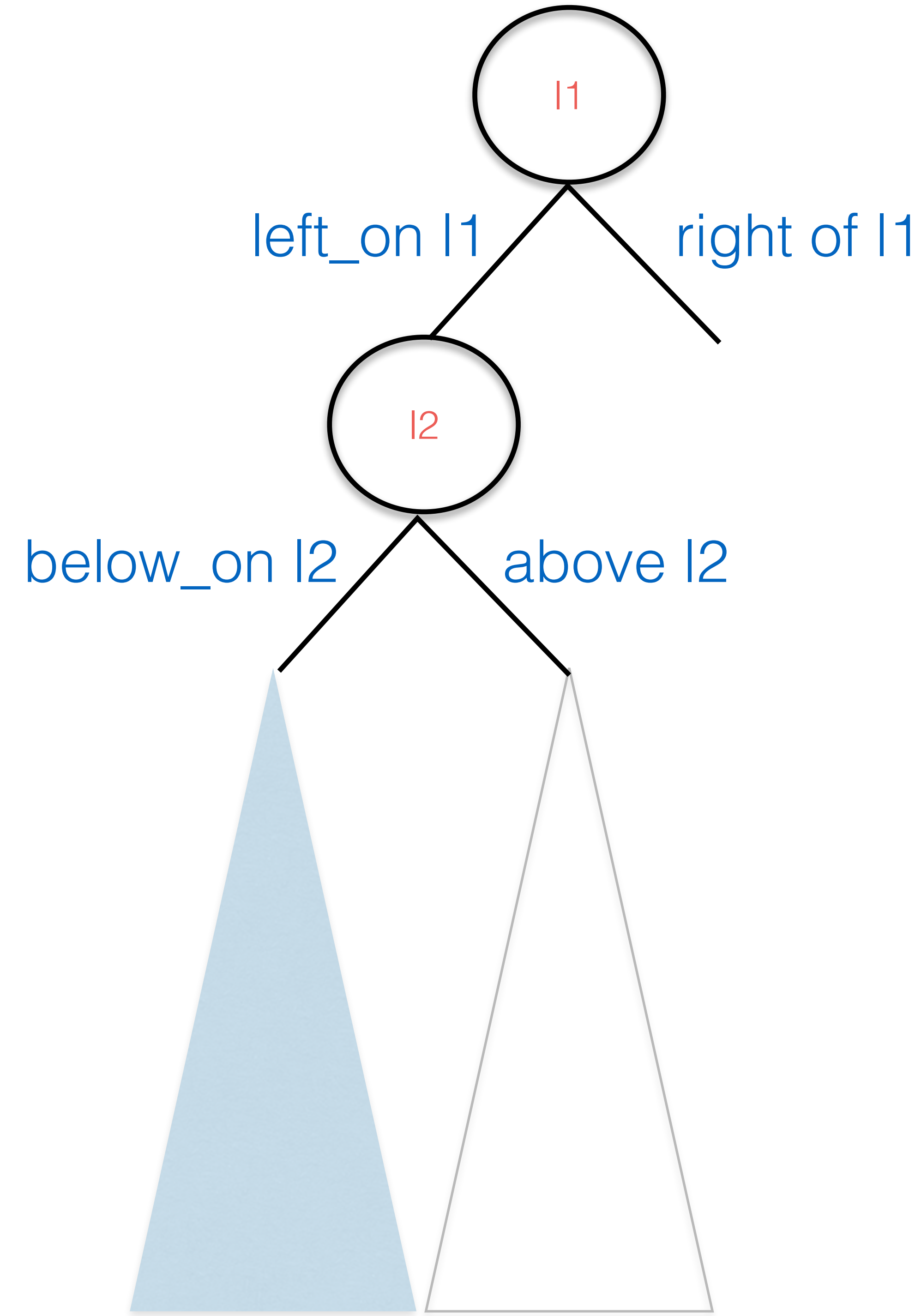




Each node in the tree corresponds to a region in the plane.

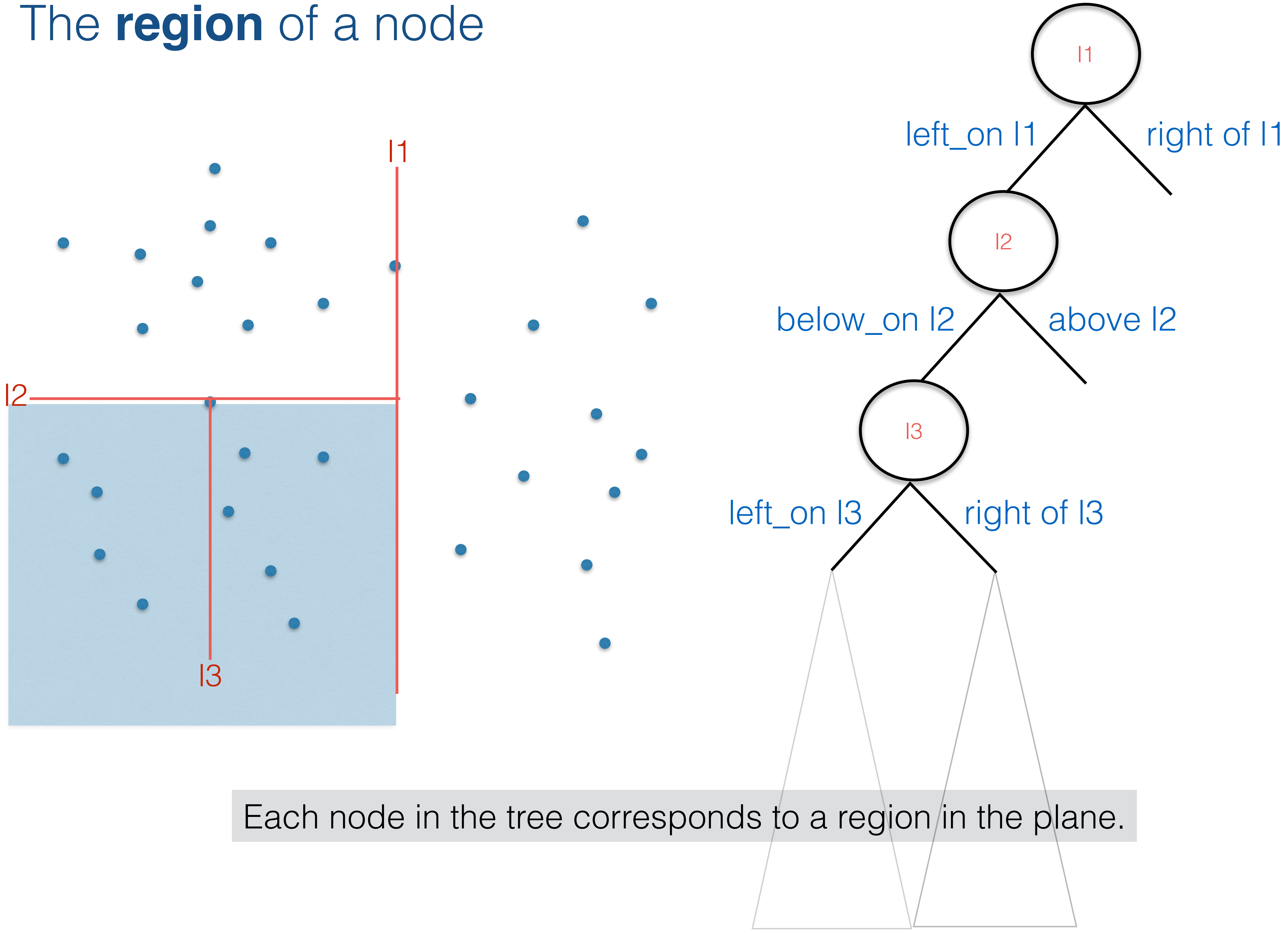
The **region** of a node





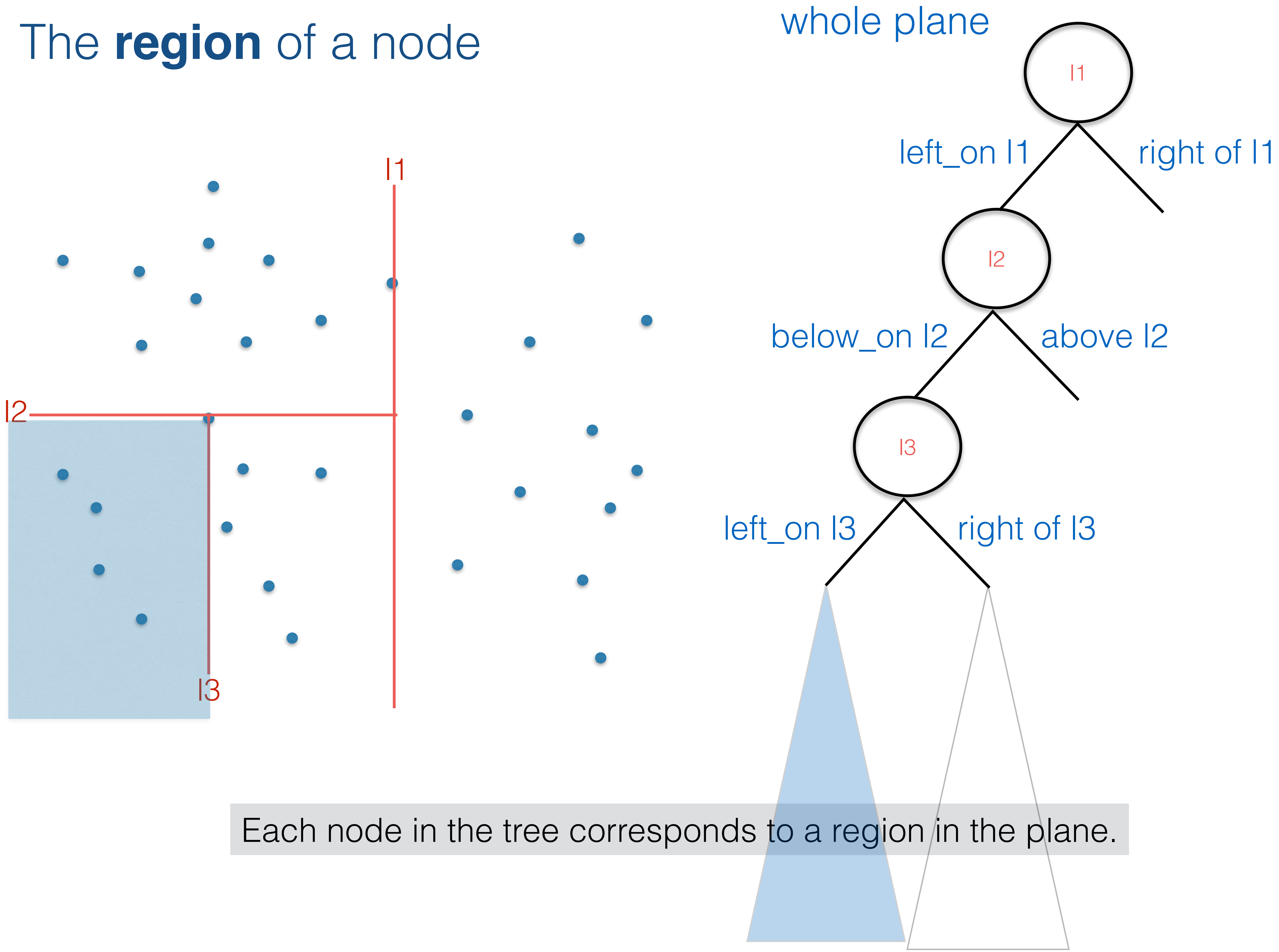
Each node in the tree corresponds to a region in the plane.

The **region** of a node



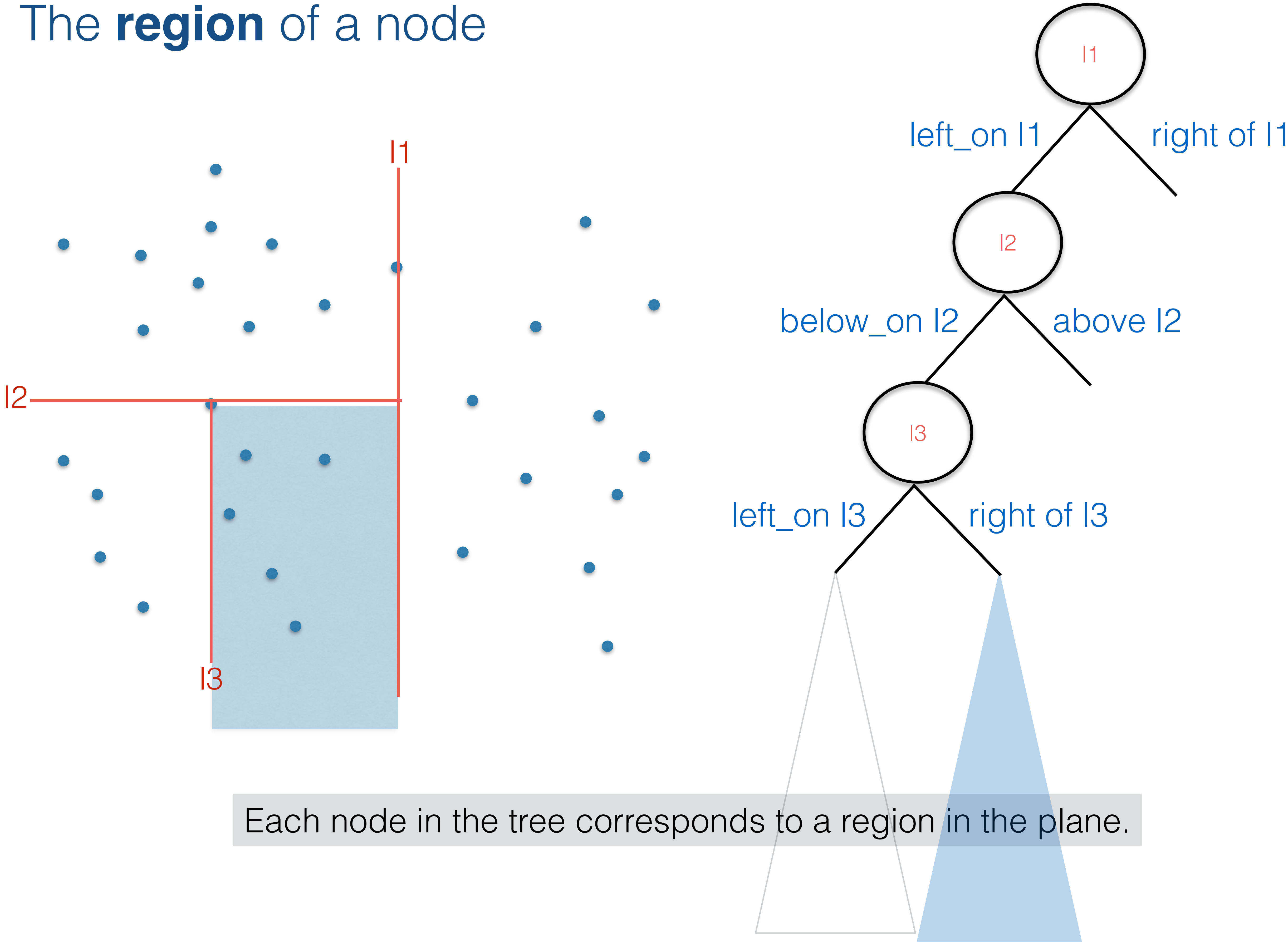
Each node in the tree corresponds to a region in the plane.

The **region** of a node



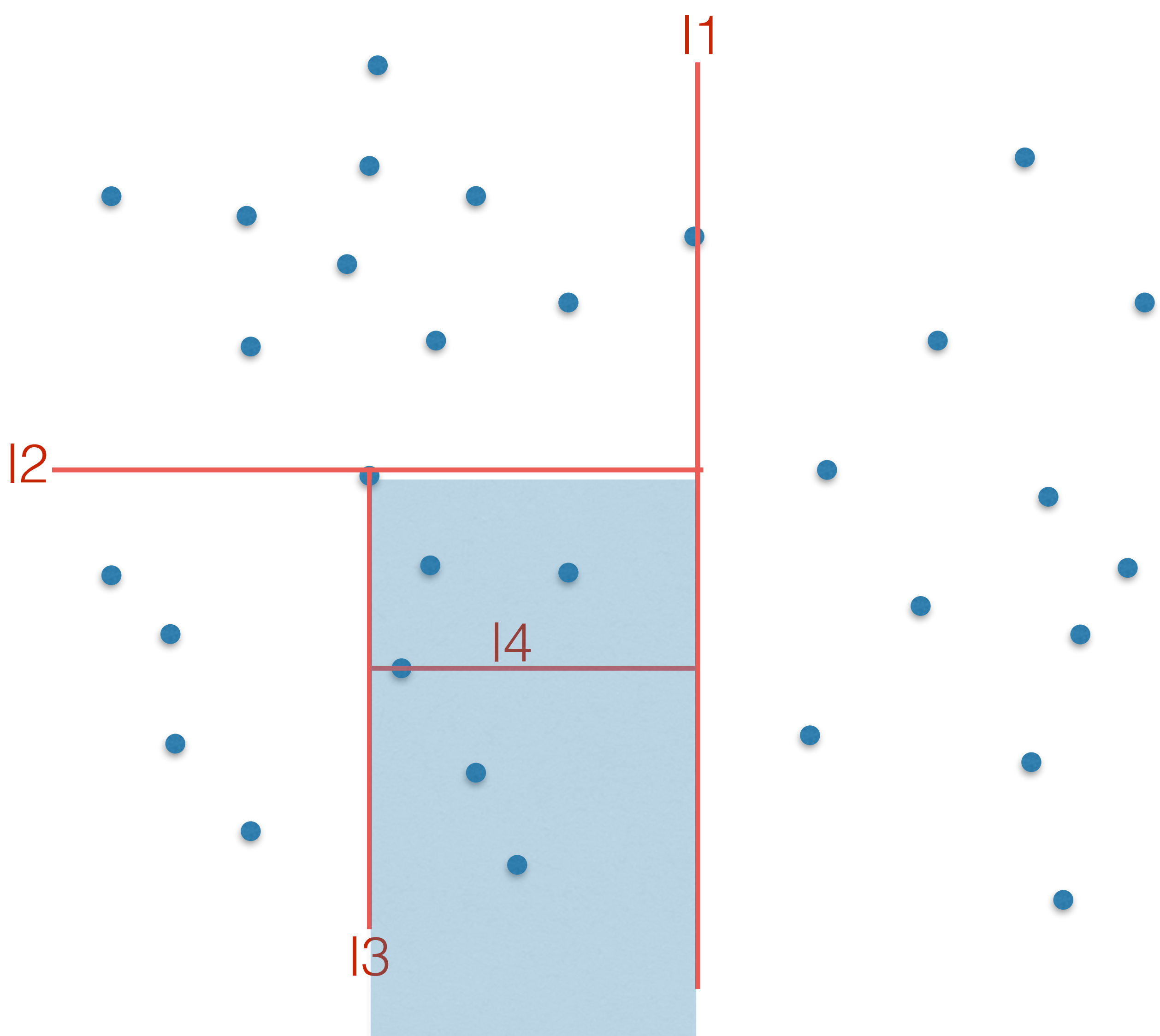
Each node in the tree corresponds to a region in the plane.

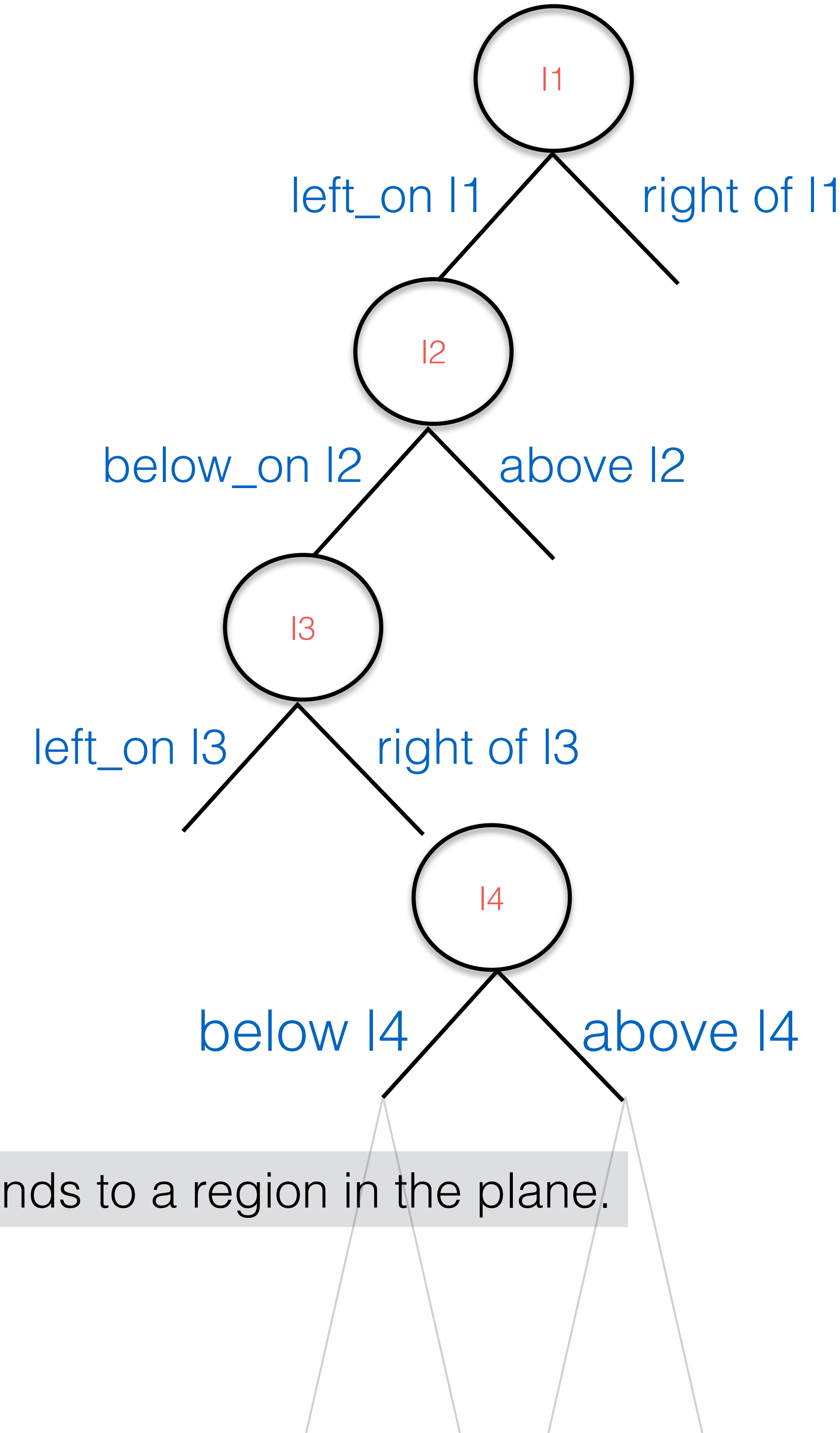
The **region** of a node



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

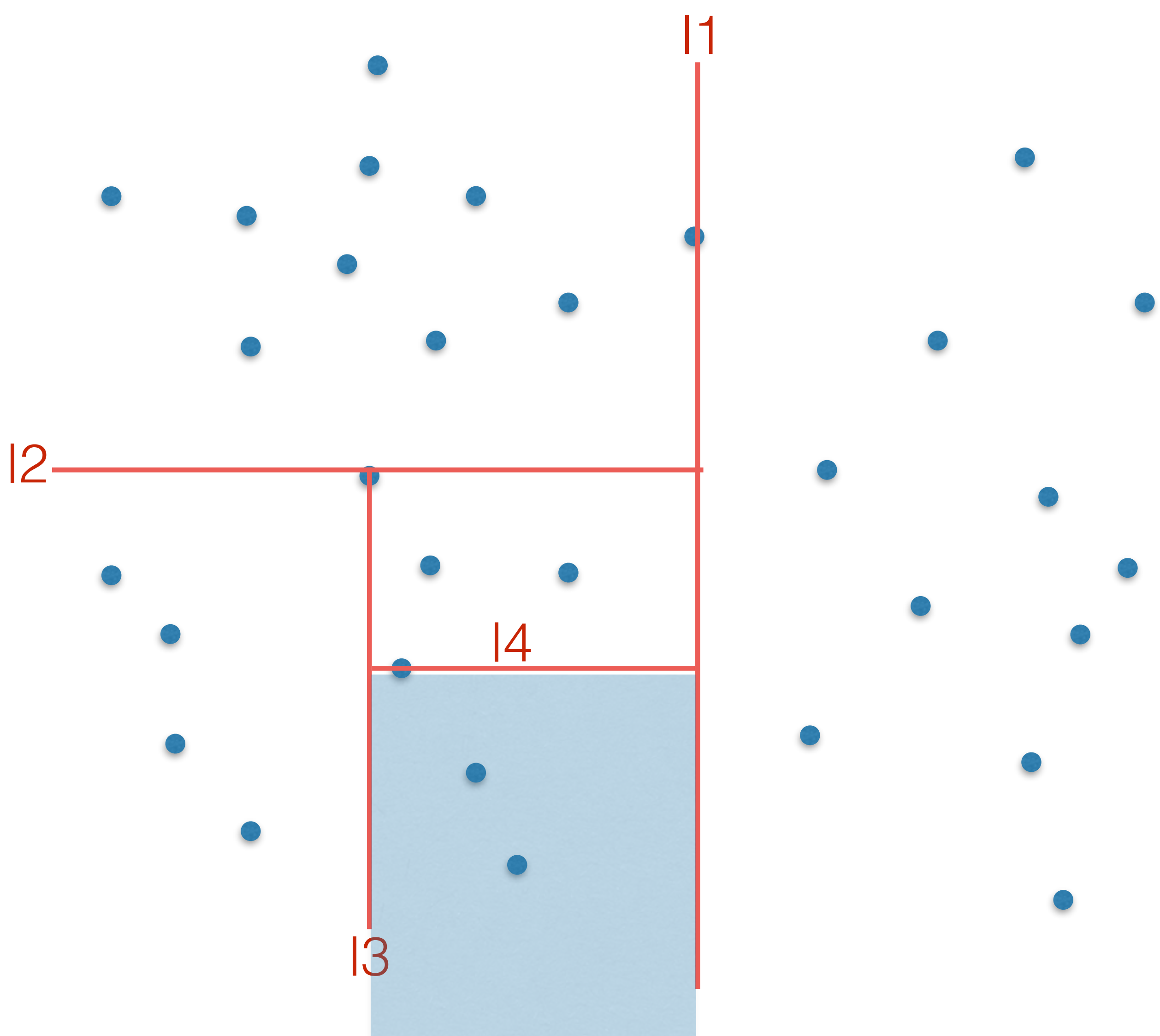
The **region** of a node

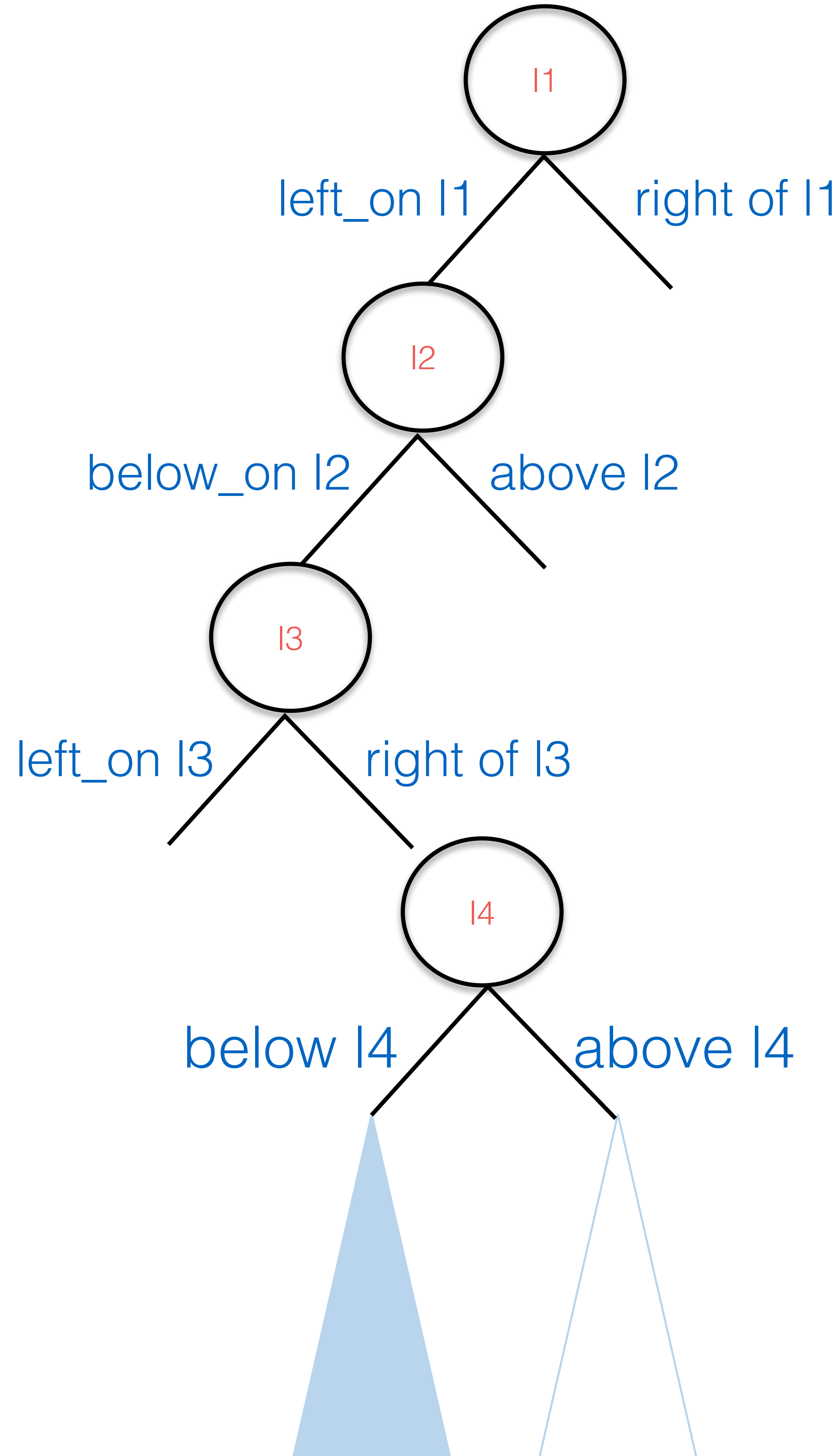




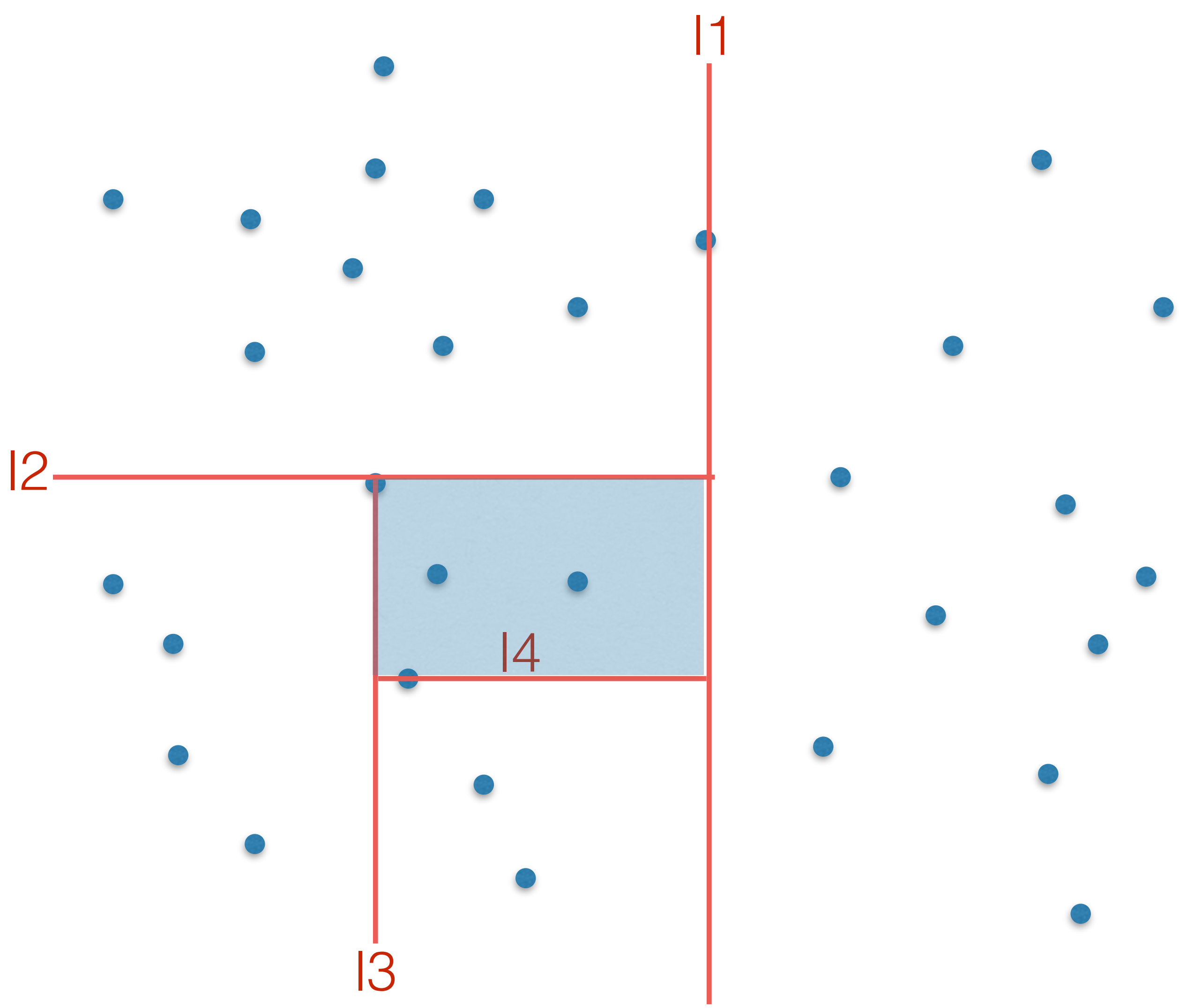
Each node in the tree corresponds to a region in the plane.

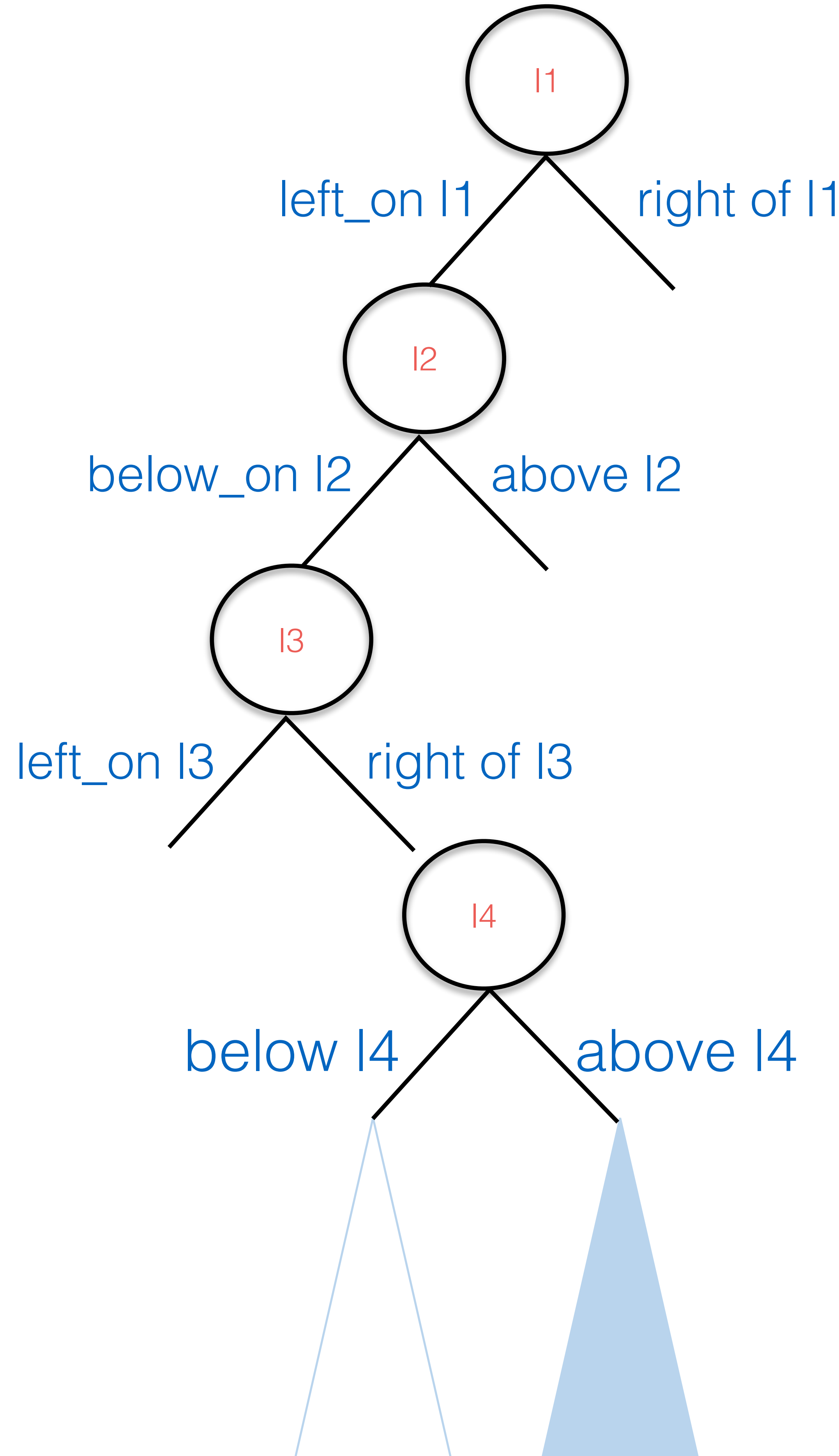
The **region** of a node



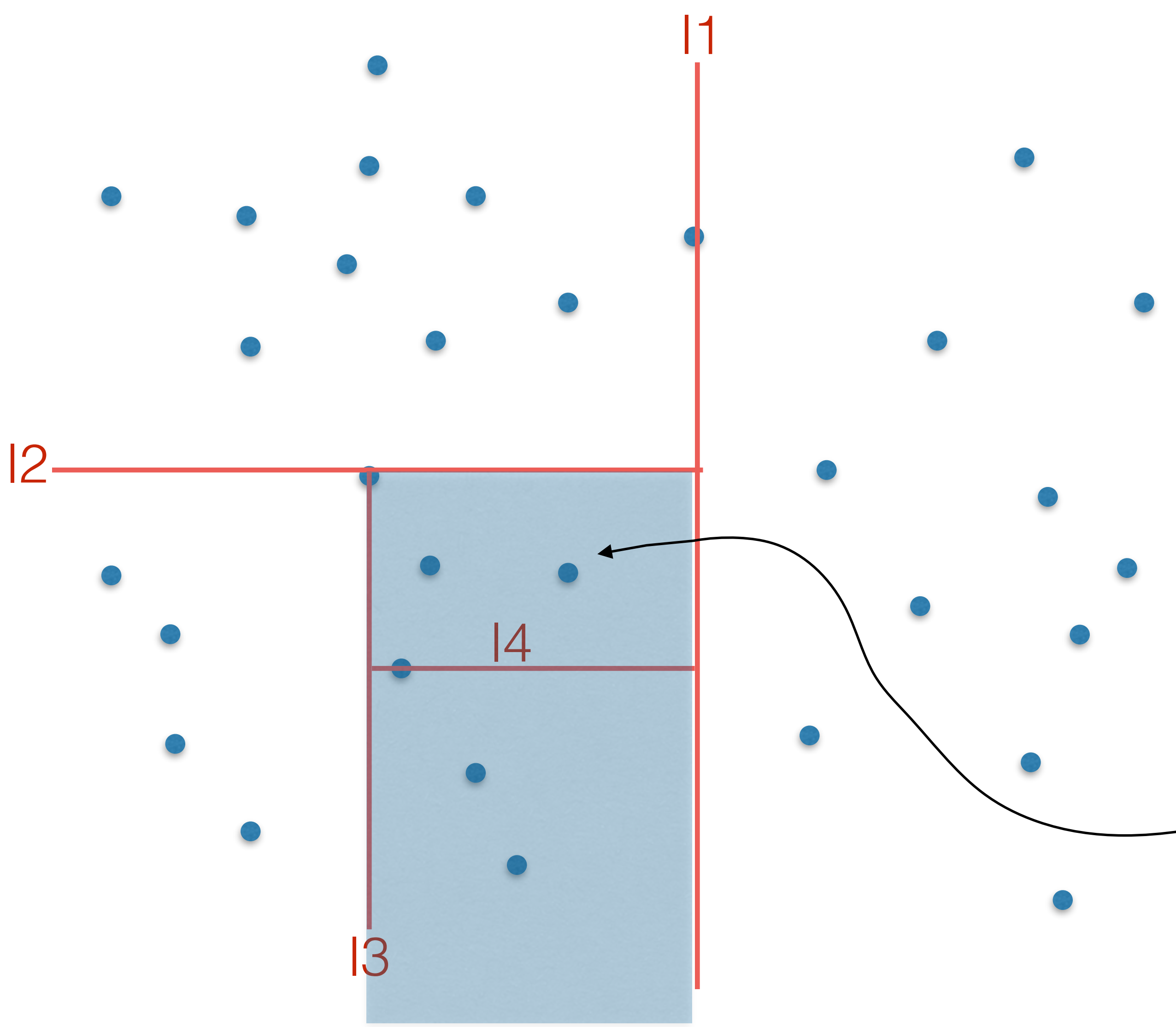


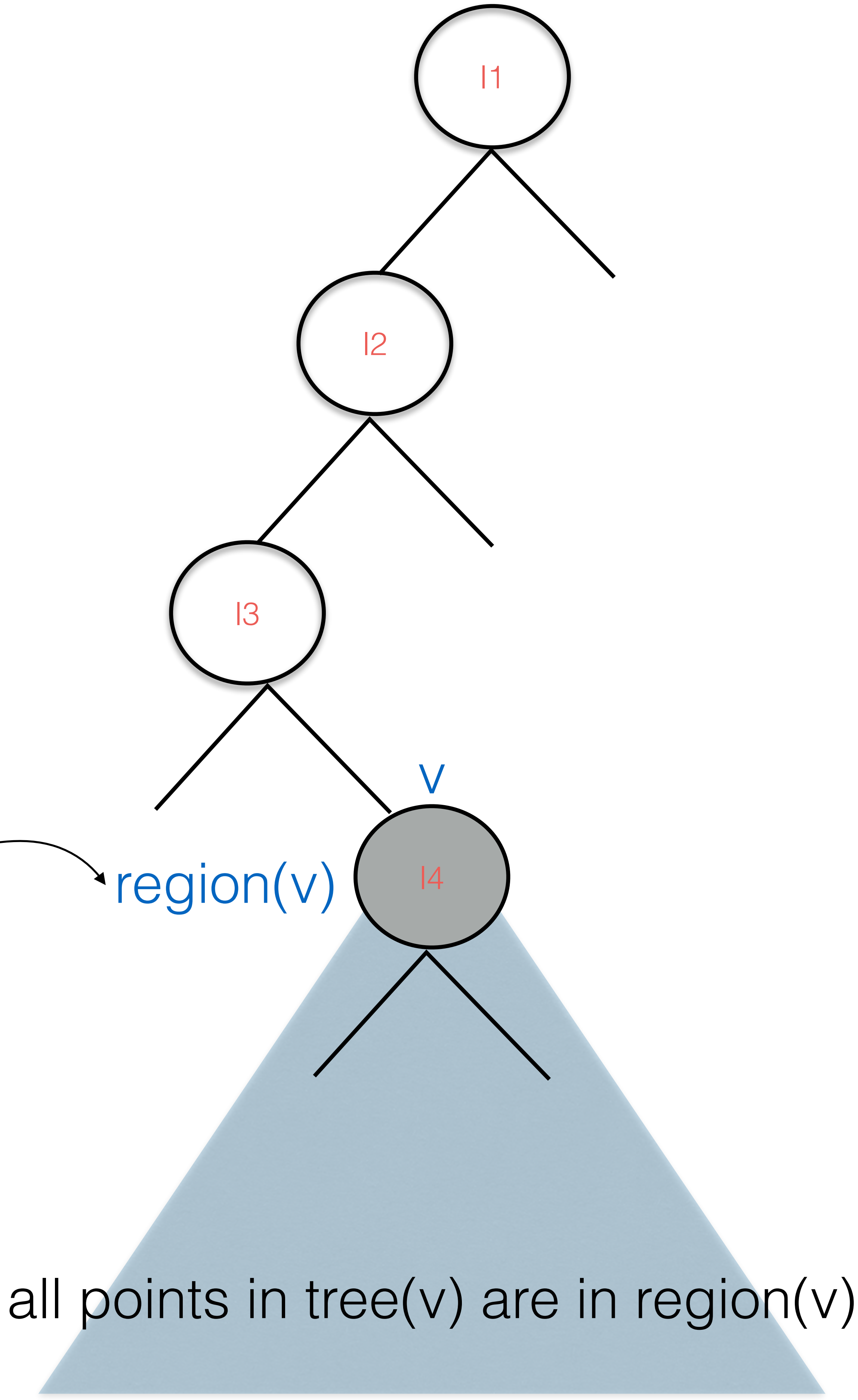
The **region** of a node





The **region** of a node

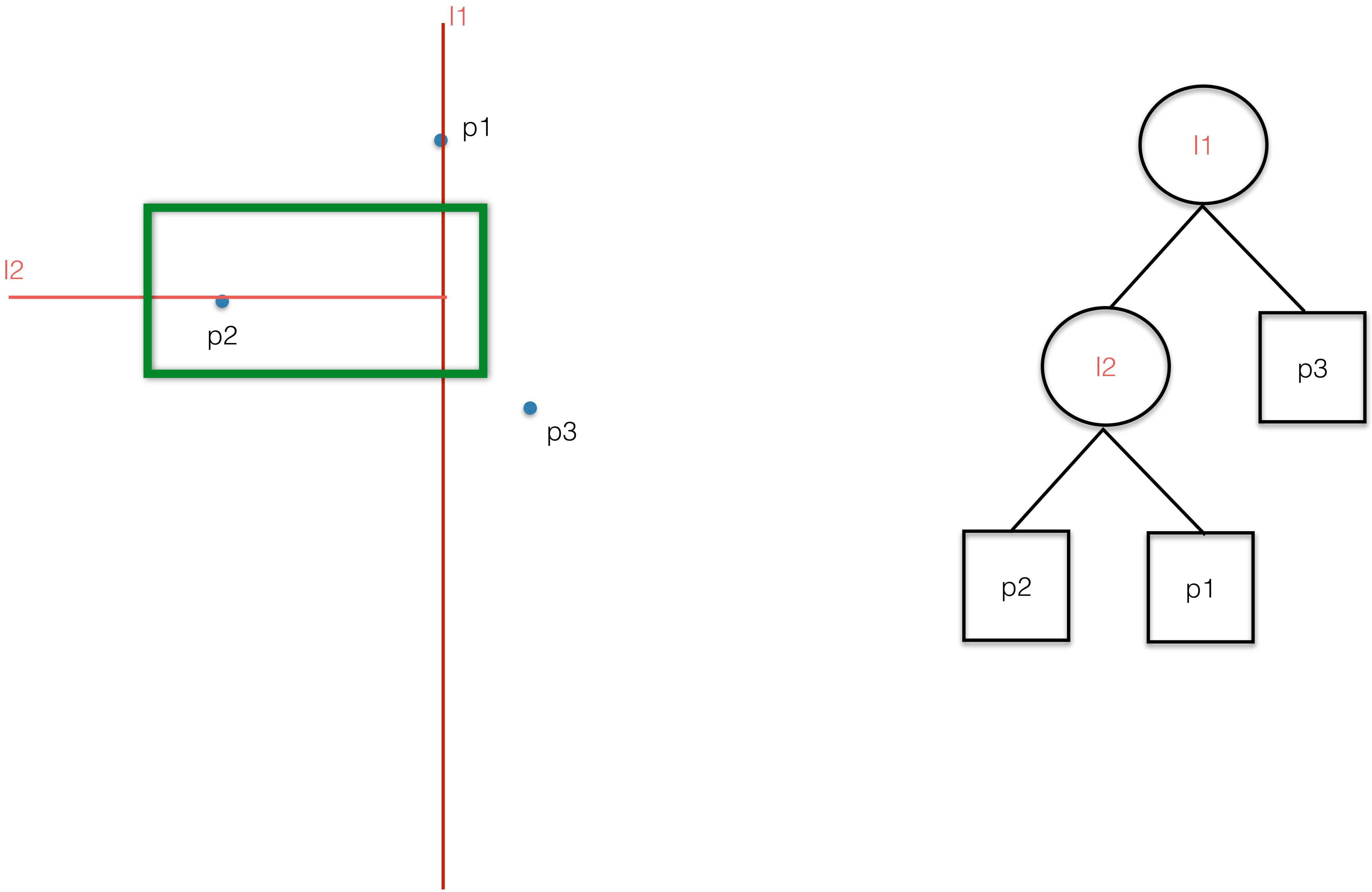




Each node in the tree corresponds to a region in the plane.

We’ll use this insight to answer range queries

Range queries on 2d-binary-search-trees

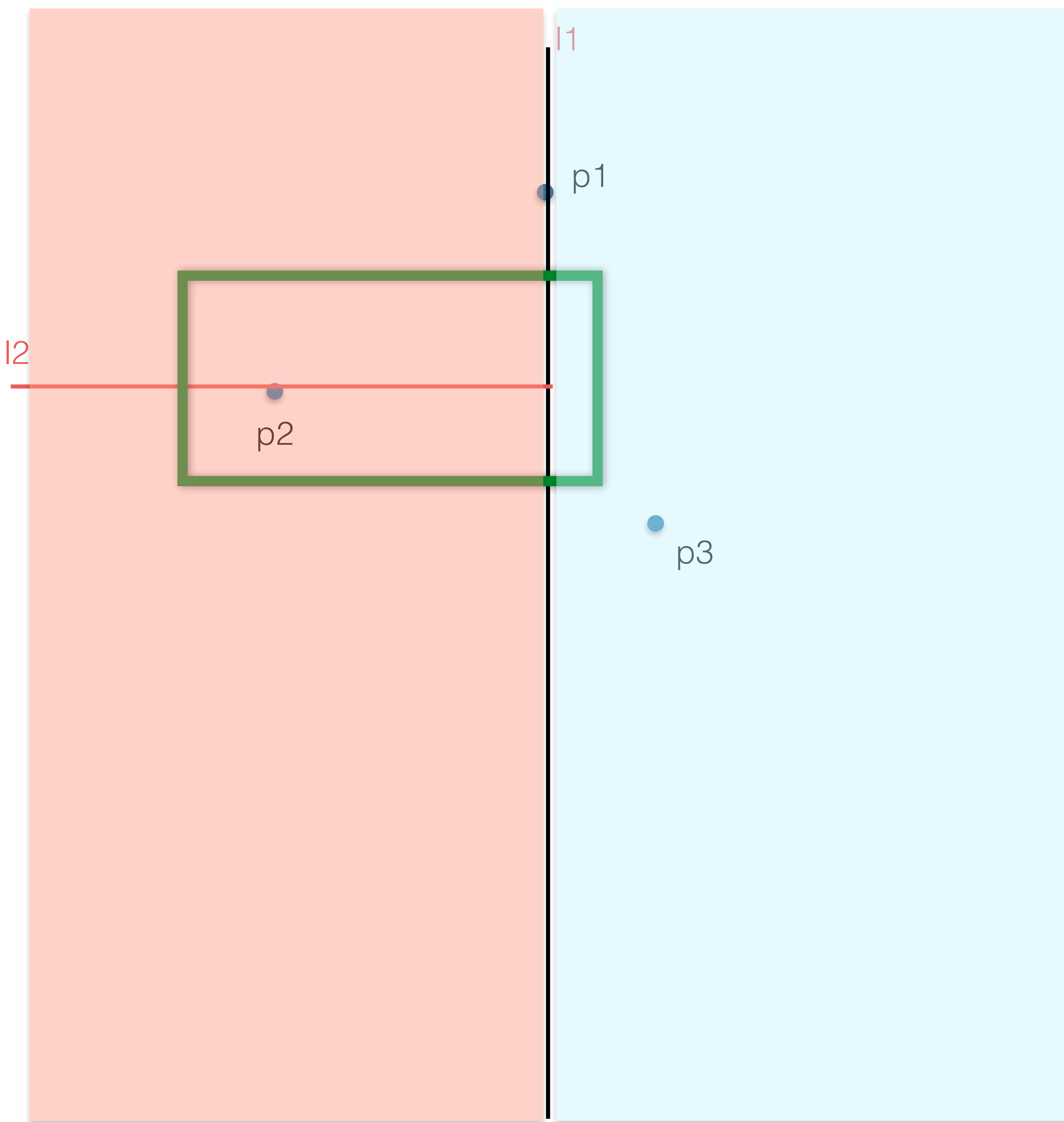


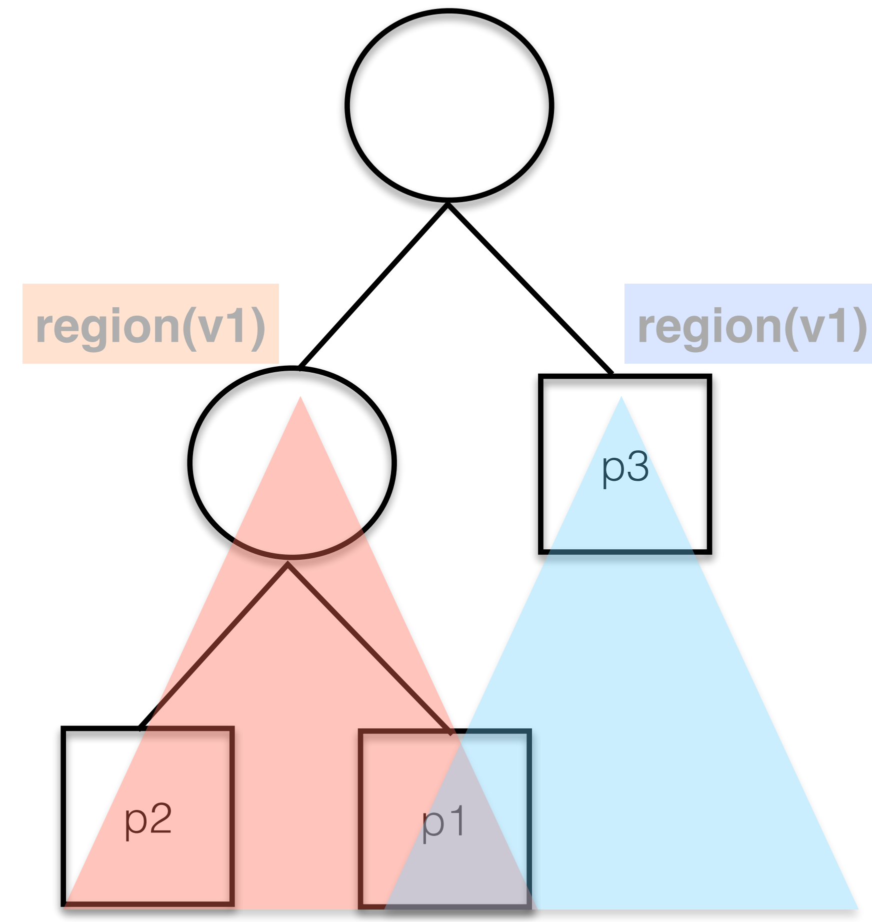
Let’s bring in the space partition defined by the tree

We are at the root node, looking at the two children. To which child should send the query to?

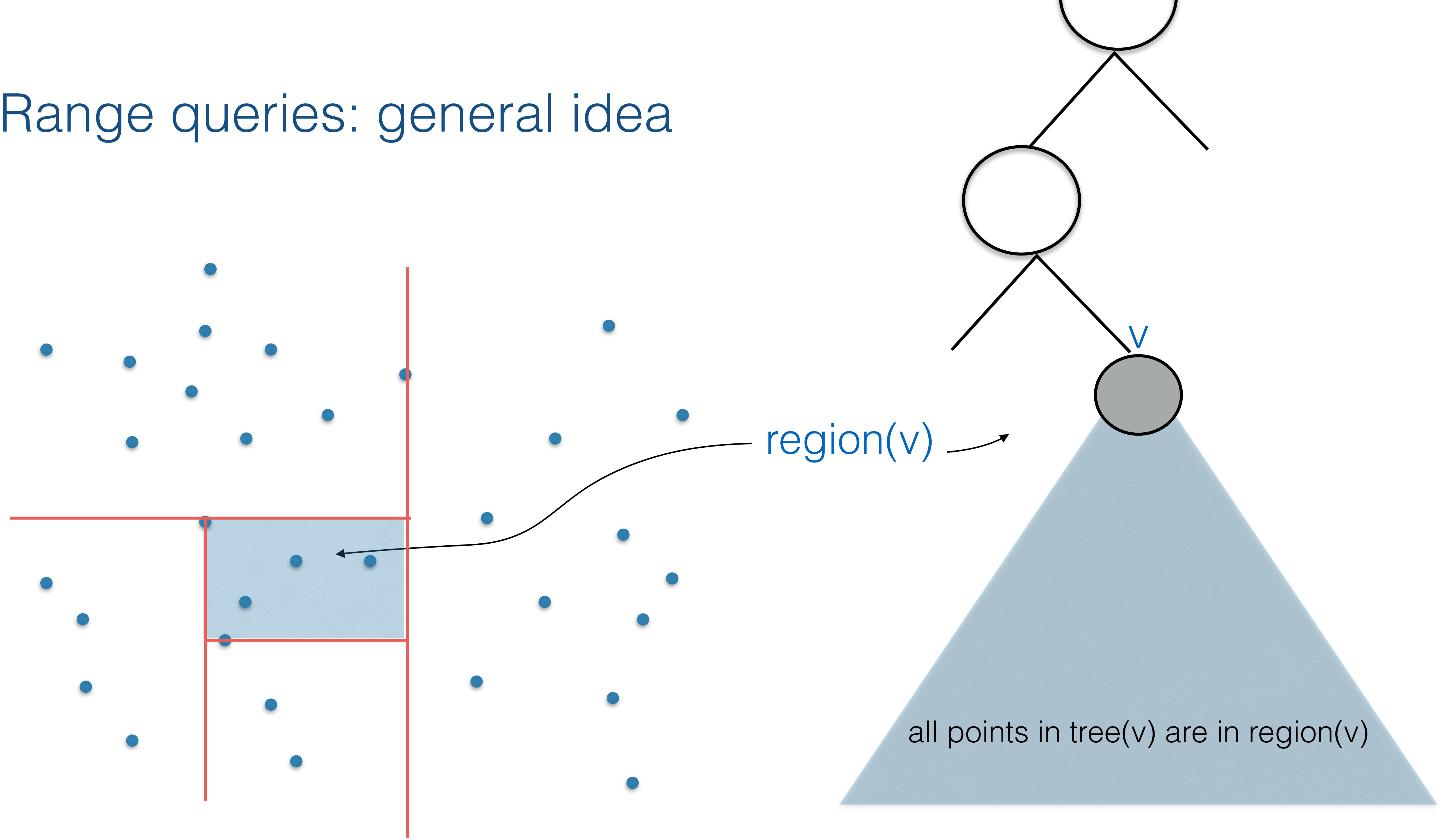
Can left child contain points in the range?

Can right child contain points in the range?

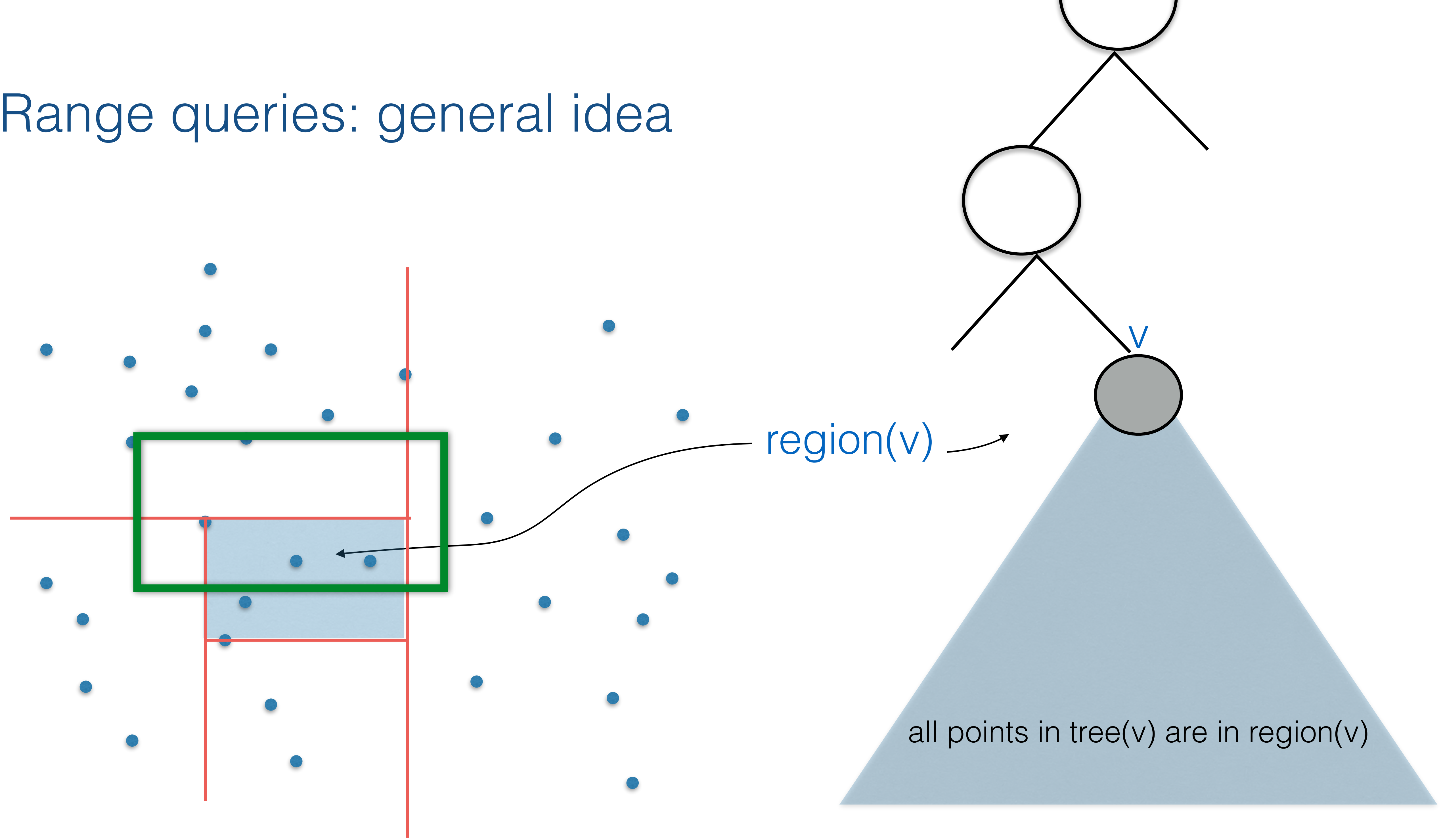


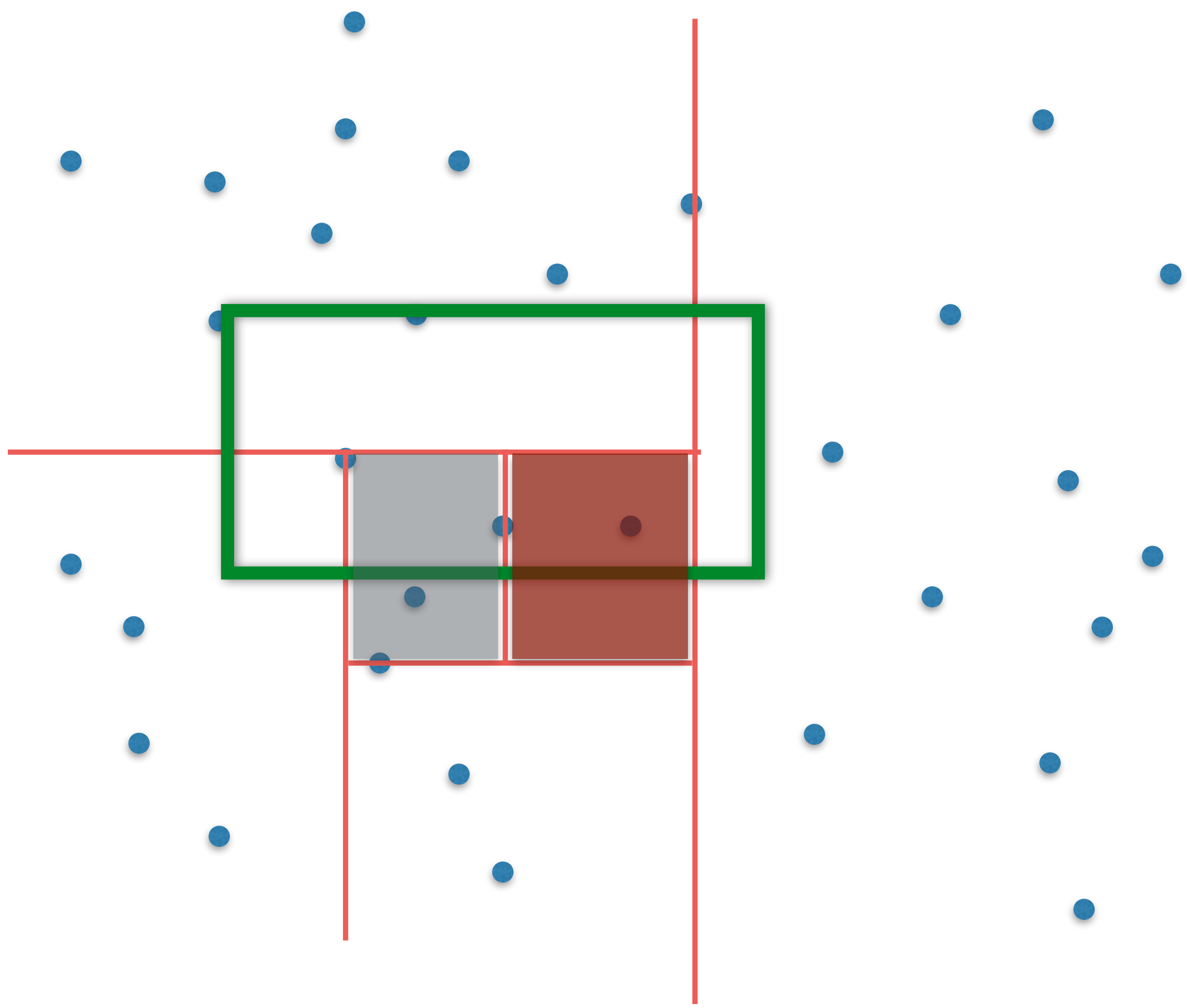


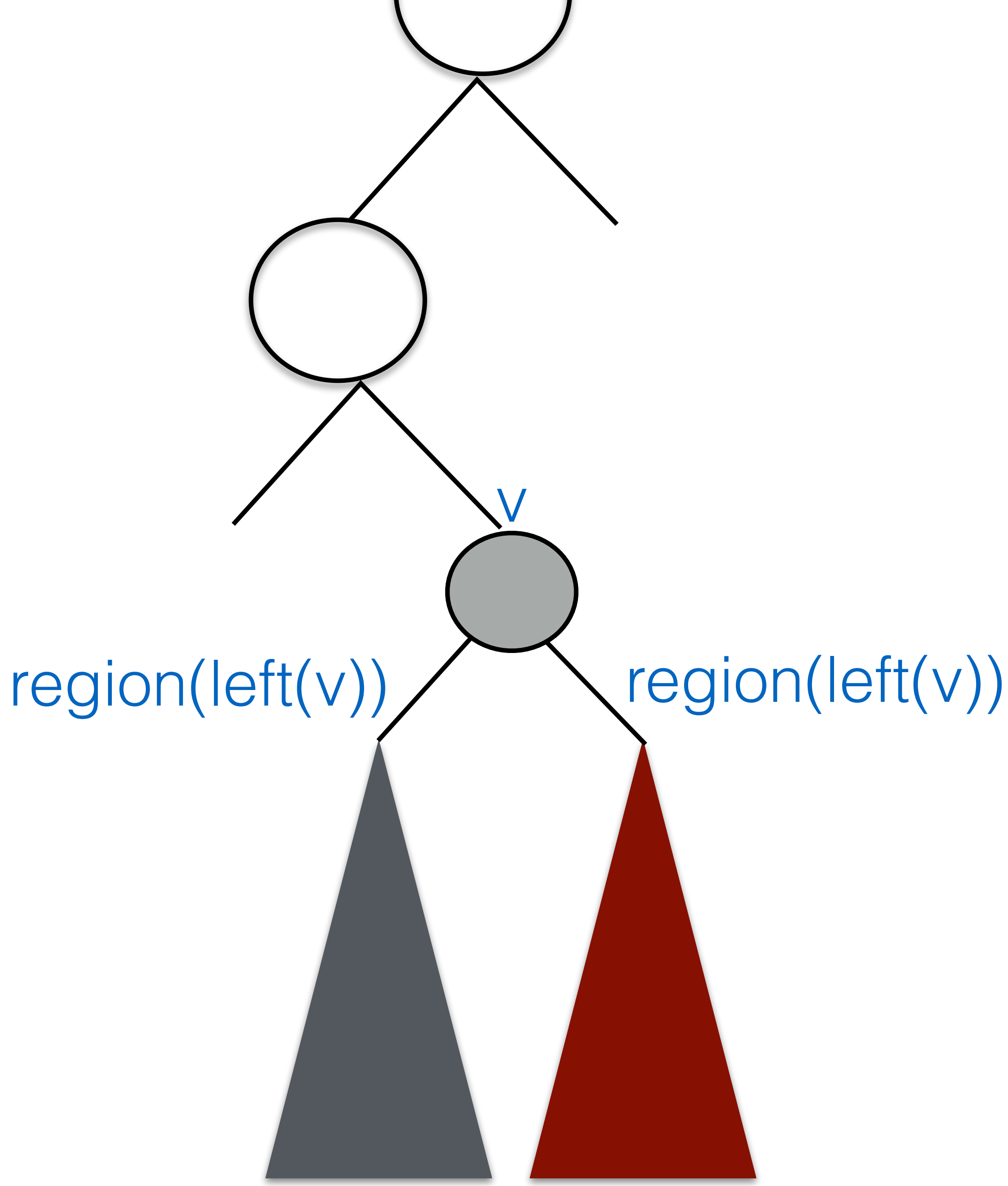
Range queries: general idea



Range queries: general idea

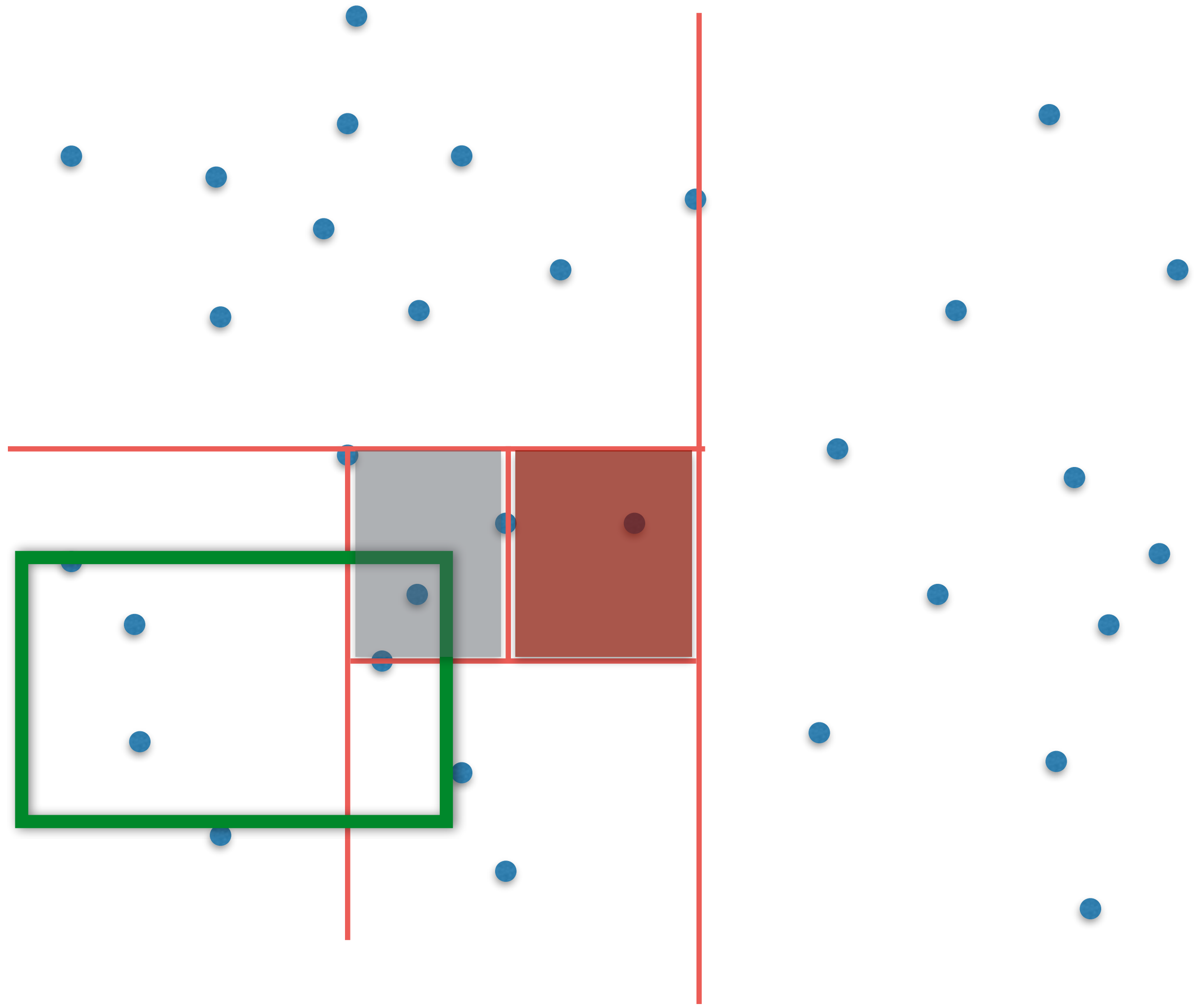


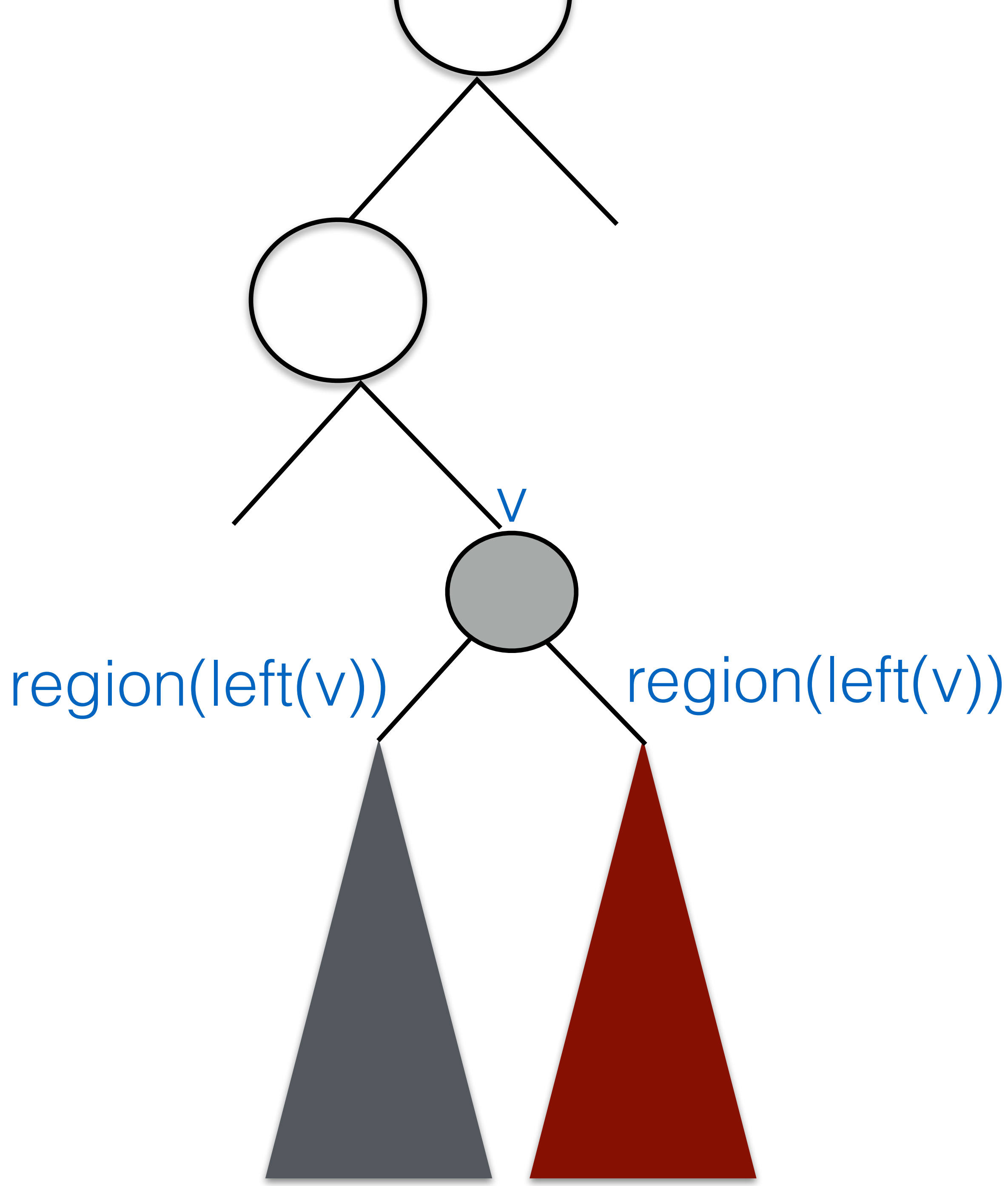




Case 1:range intersects both children

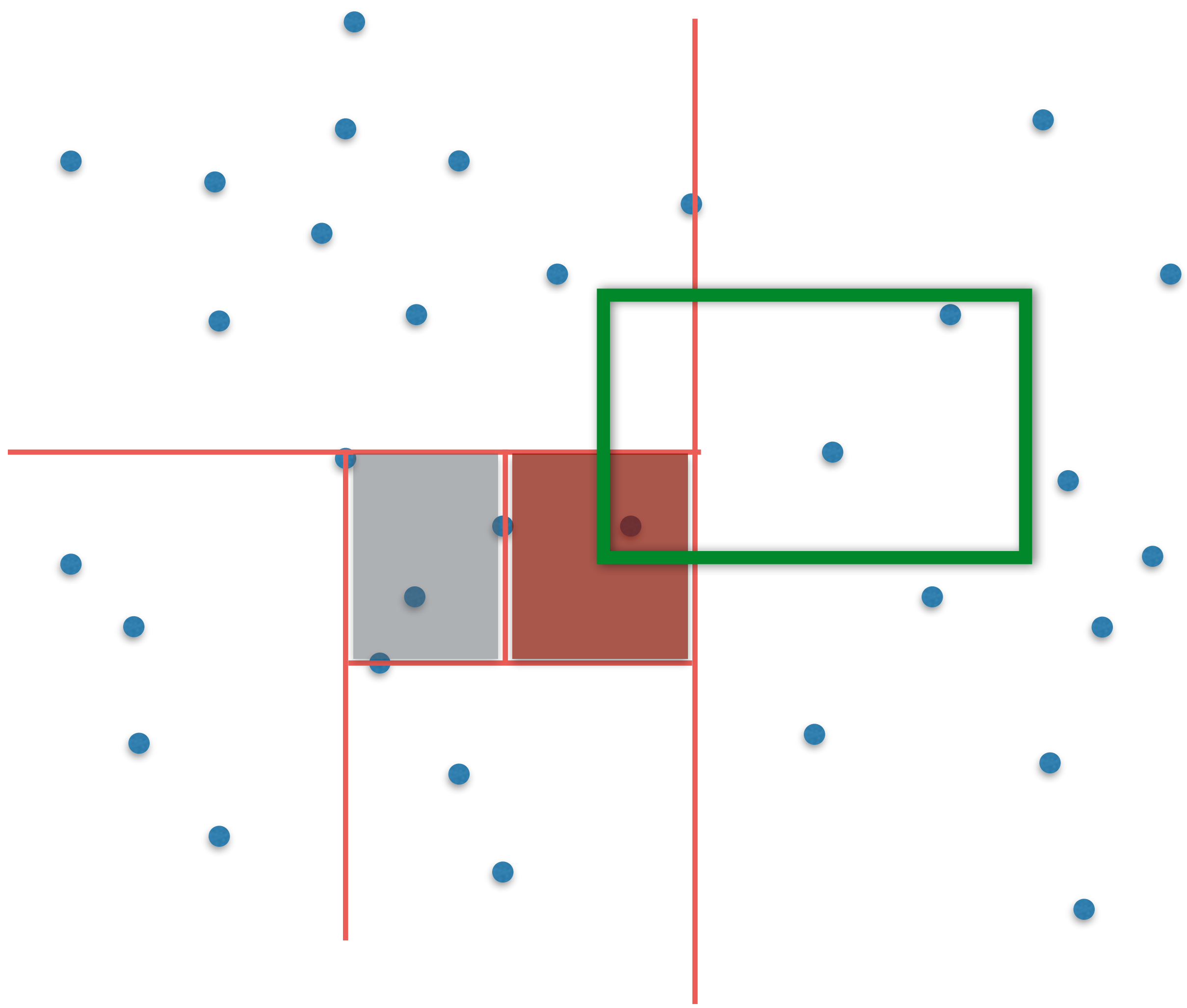
Case 2

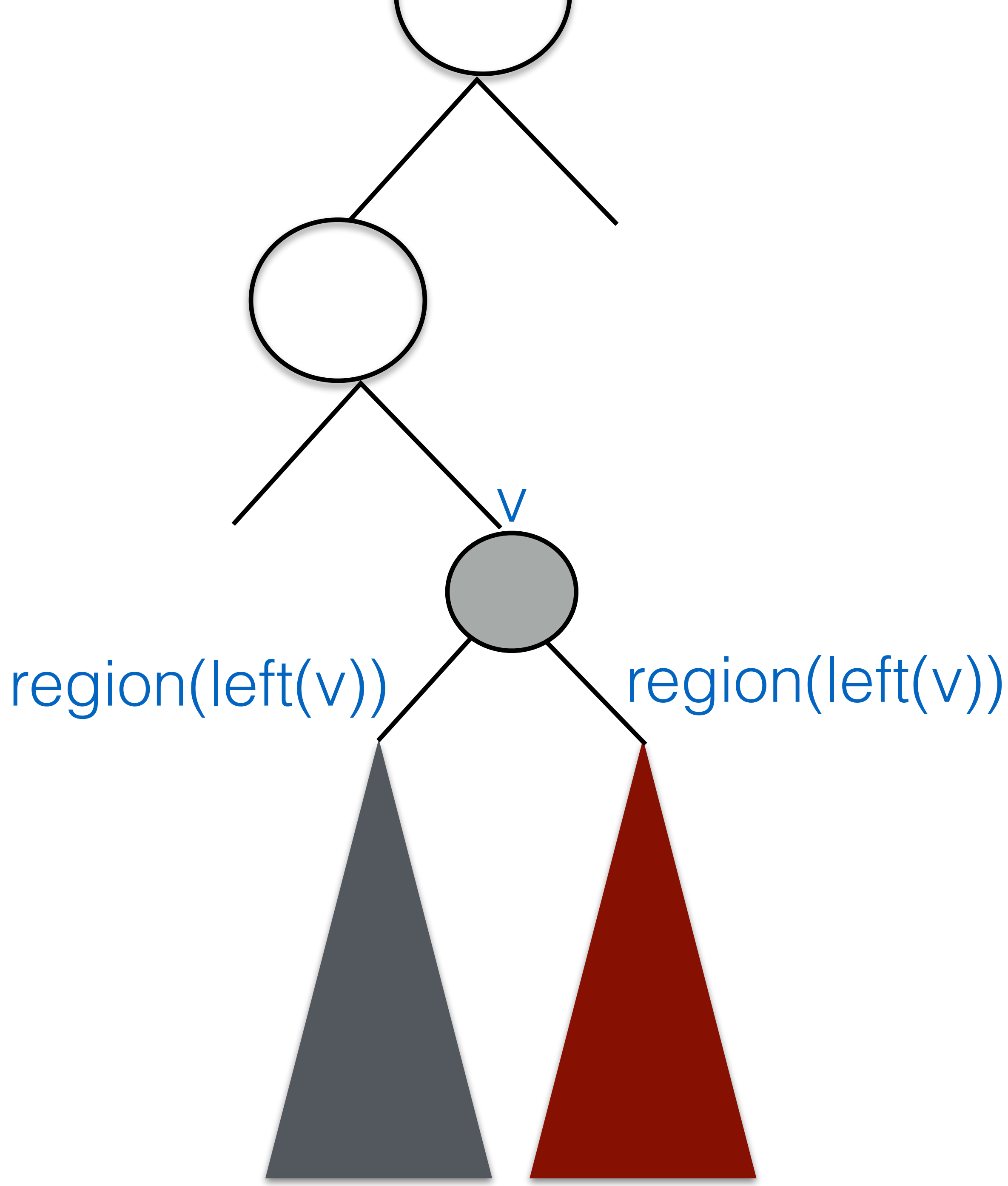




Case 2: range intersects only one child

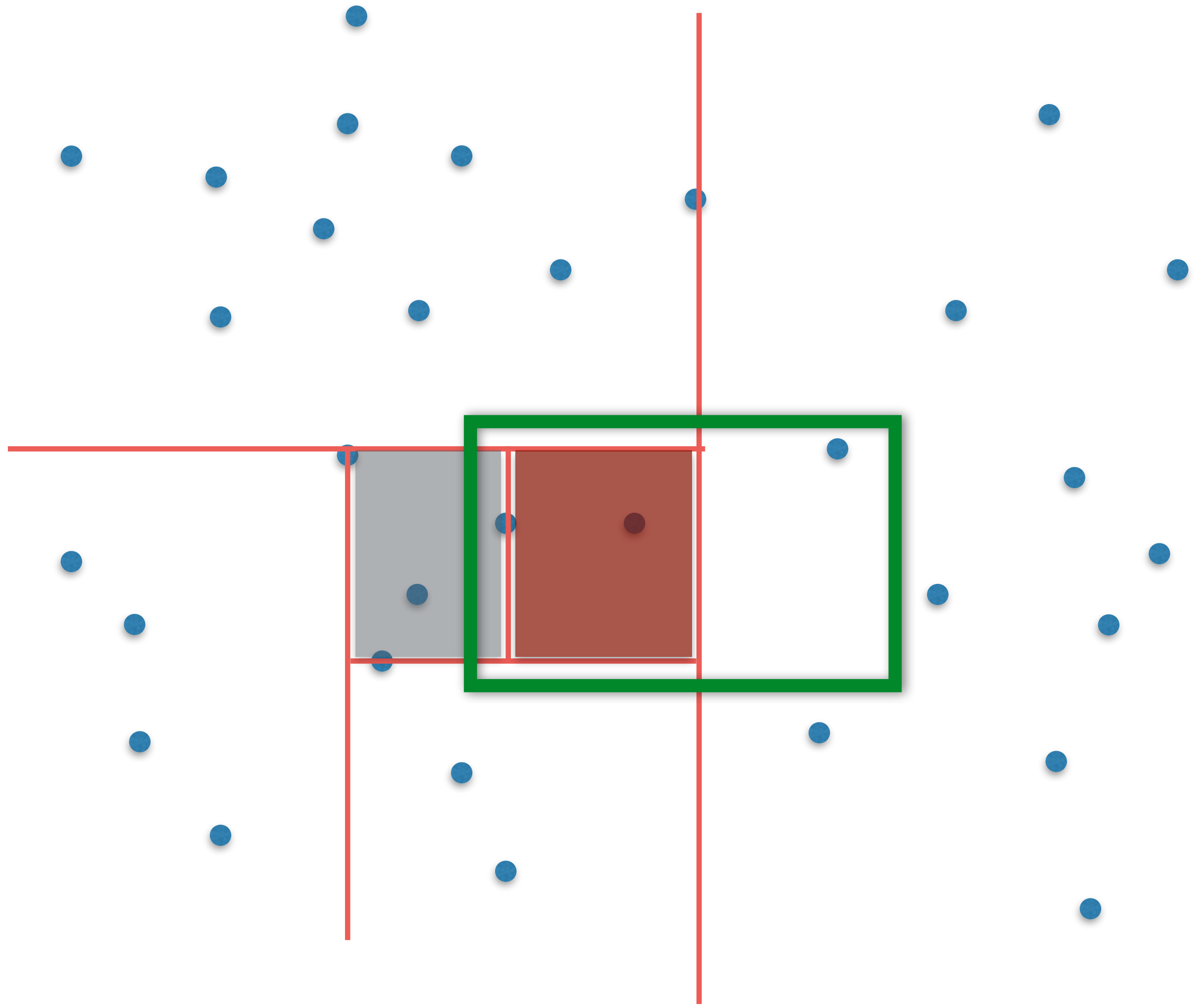
Range queries: general idea

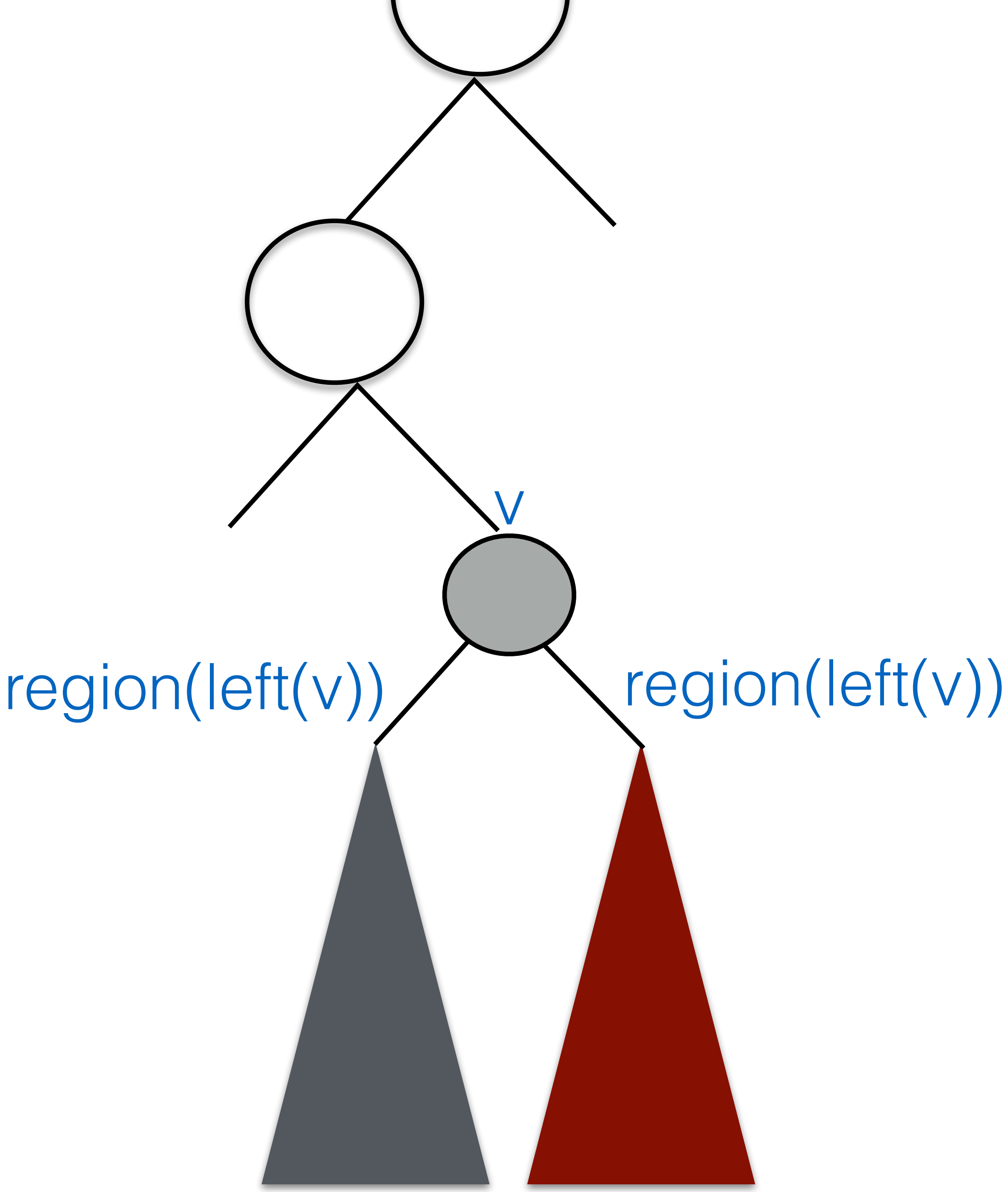




Case 2: range intersects only one child

Range queries: general idea





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

How long does a range query take?

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

Here a standard analysis does not work..

If at any node we would visit **one** child => O(lg n)

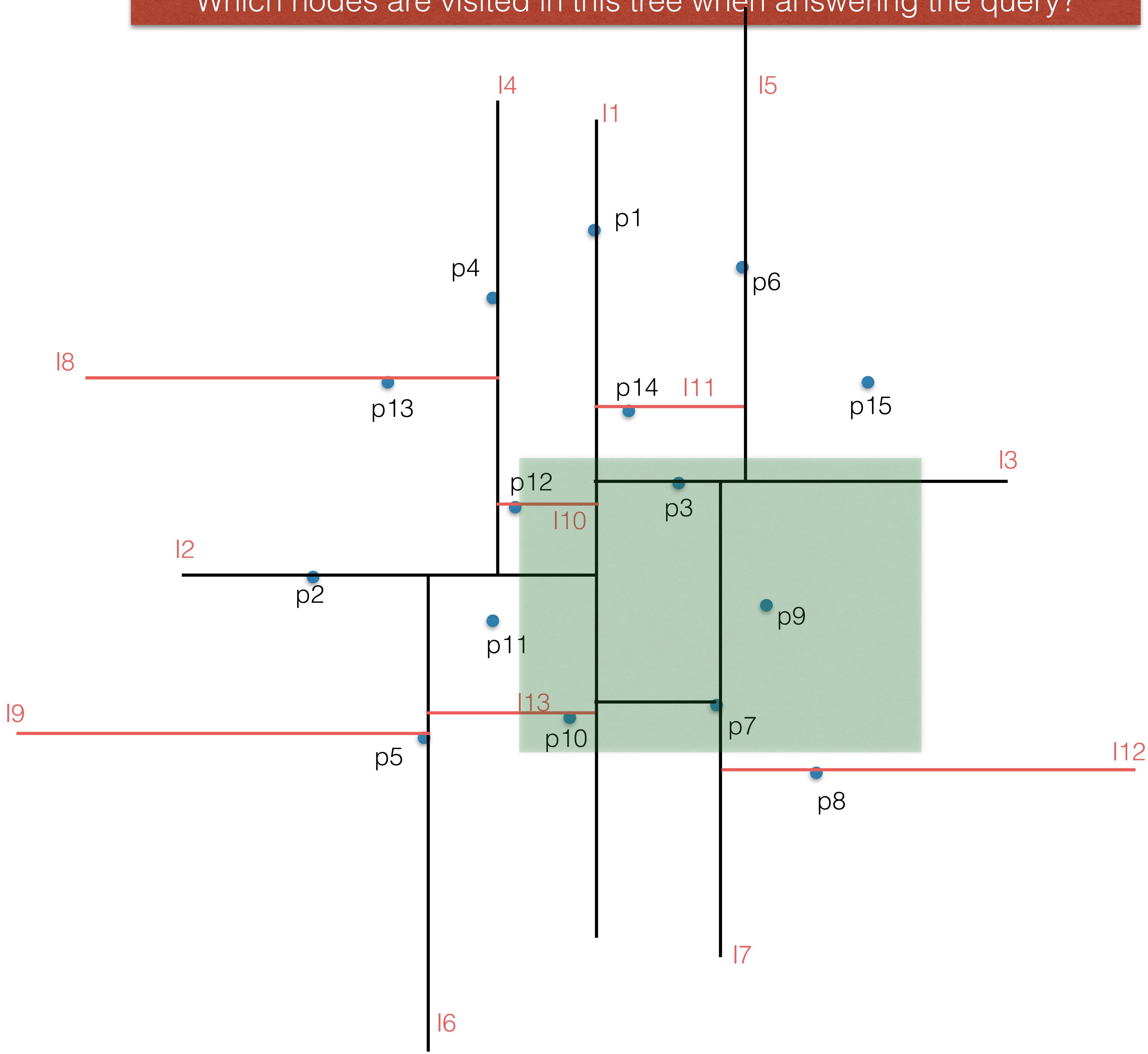
If at any node we would visit **both** children => O(n)

Here we are in between

We visit the children intersected by the query range, which can be one or both

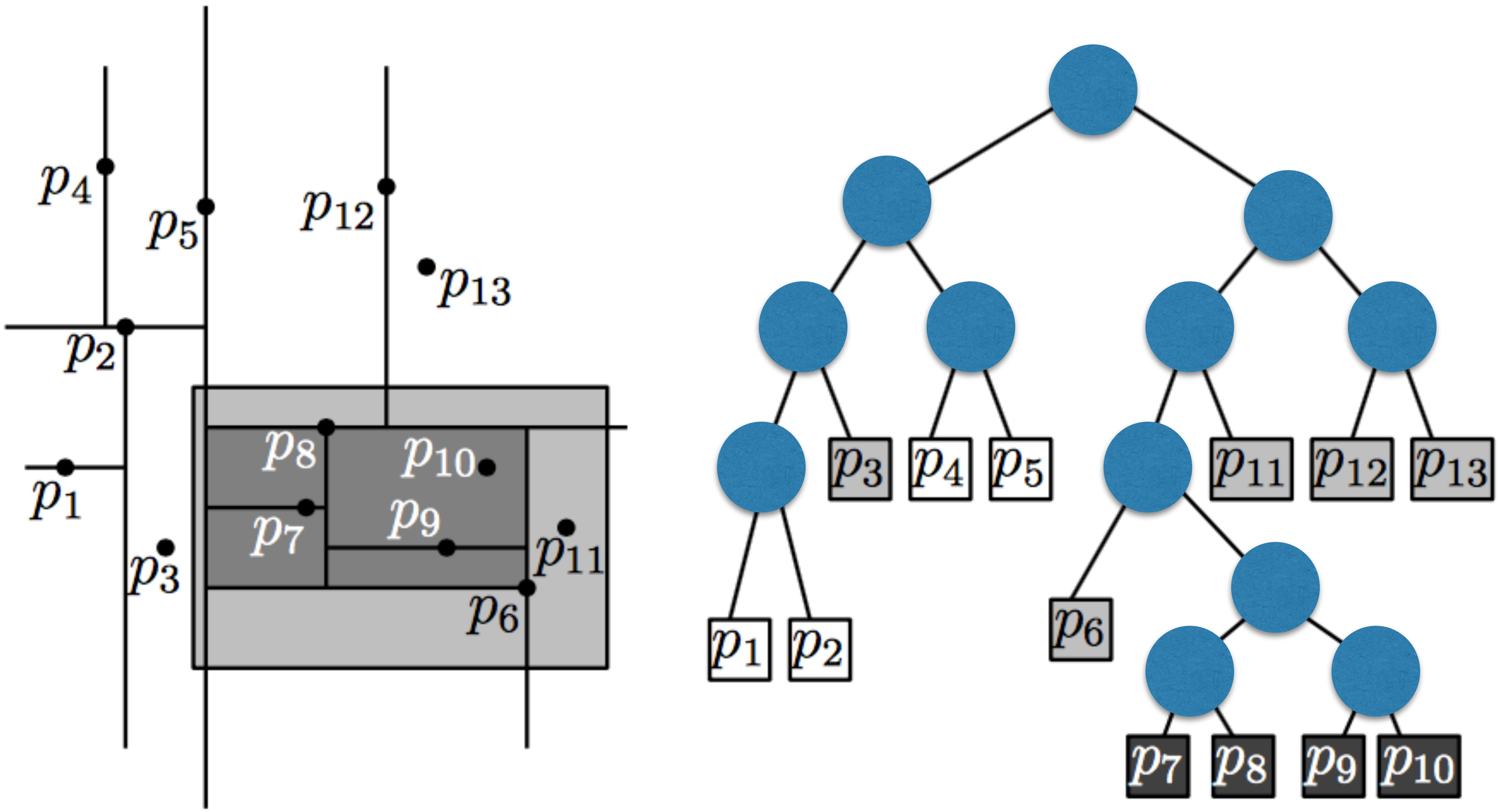
To analyze the time to answer a range query we'll look at the nodes visited in the tree when answering a query

Which nodes are visited in this tree when answering the query?



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

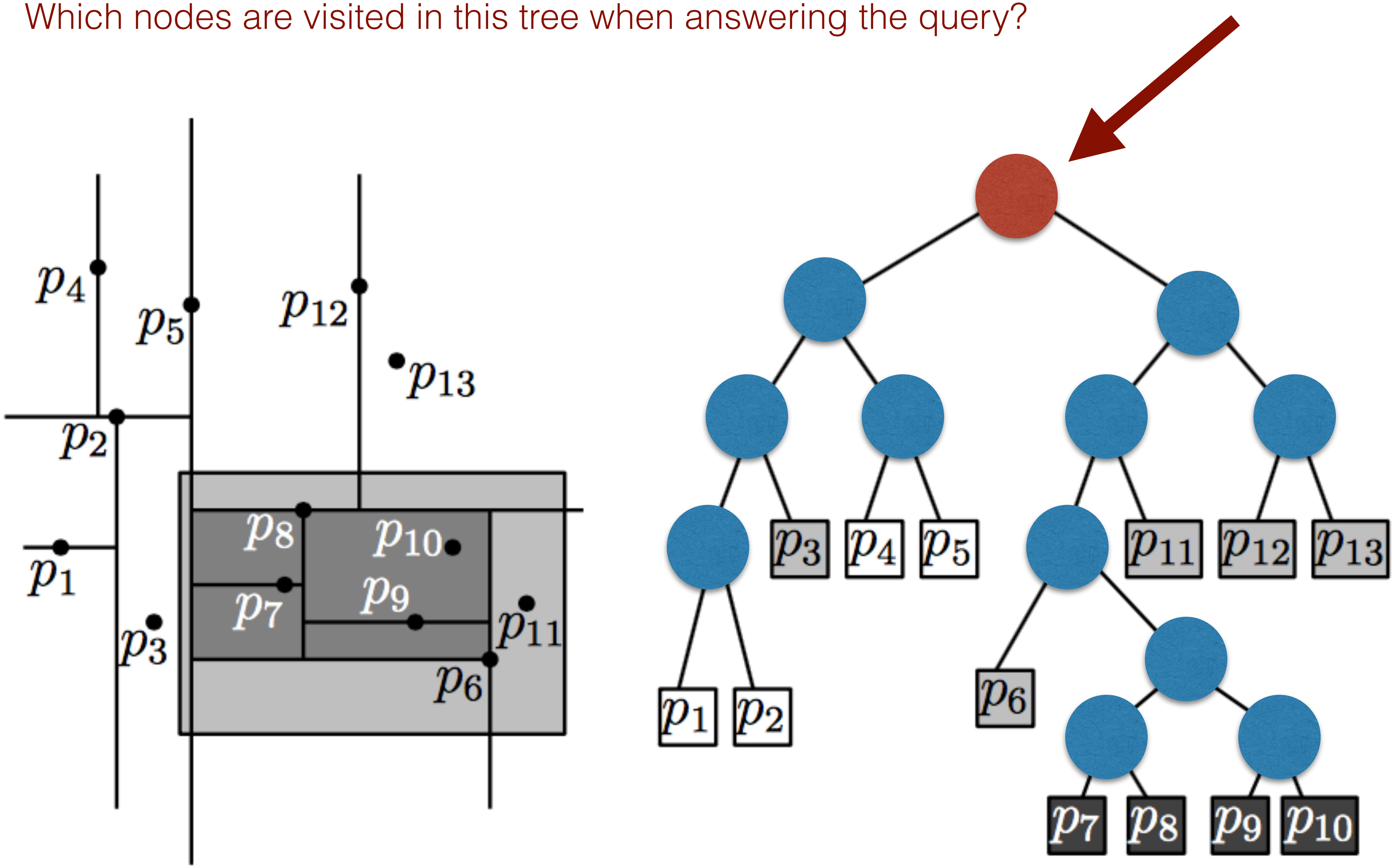
Which nodes are visited in this tree when answering the query?



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

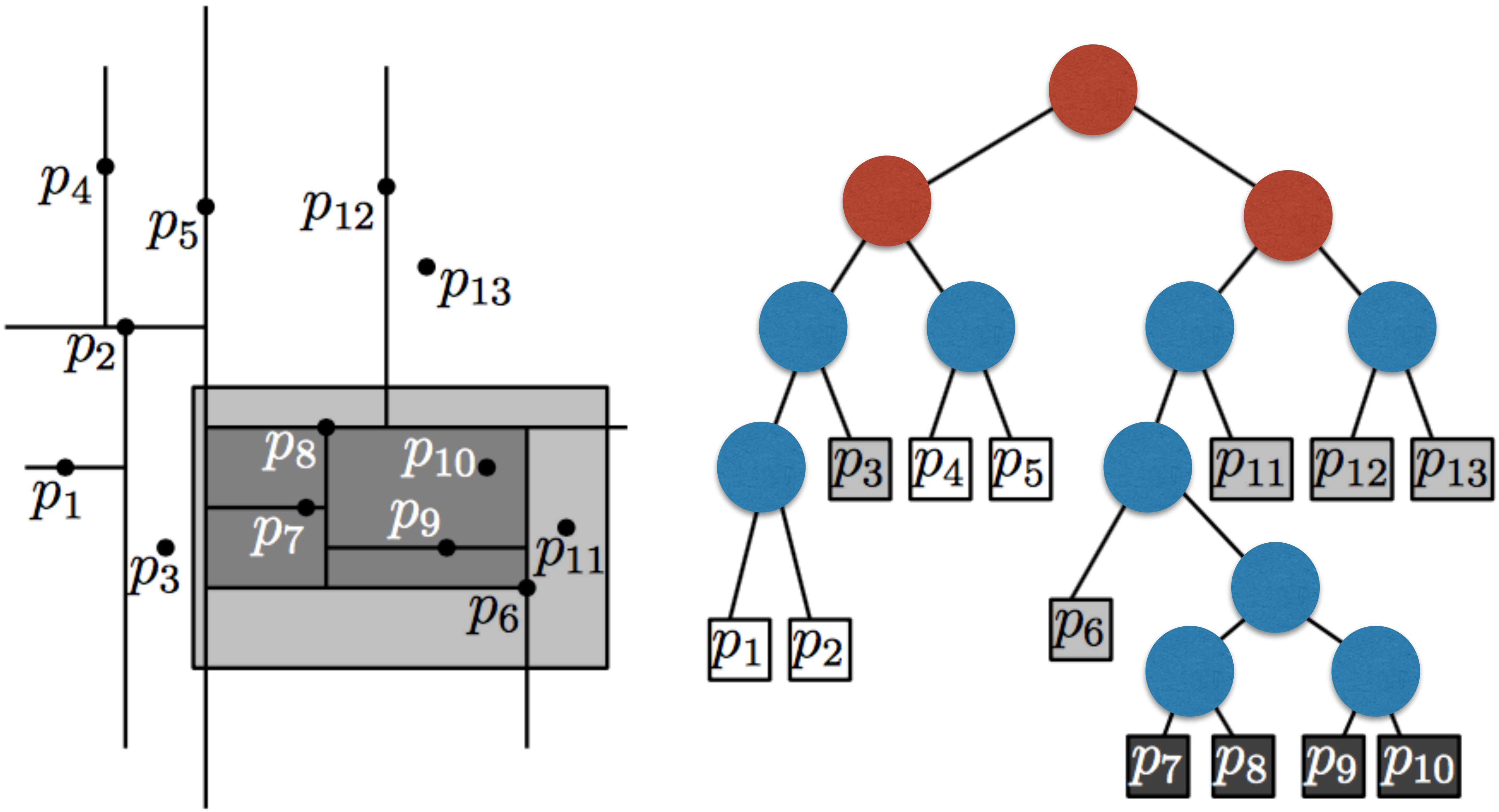
Which nodes are visited in this tree when answering the query?



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

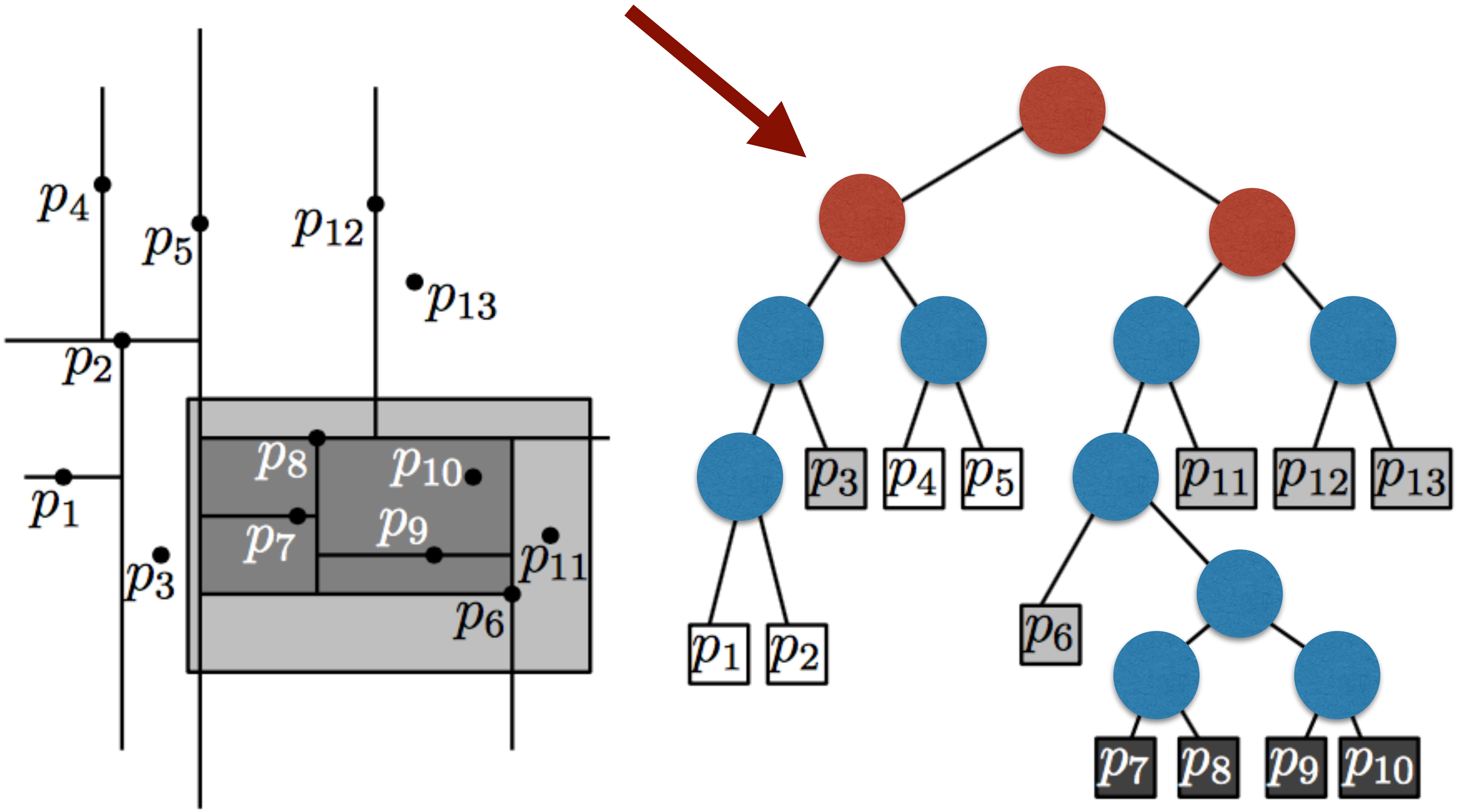
Which nodes are visited in this tree when answering the query?



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

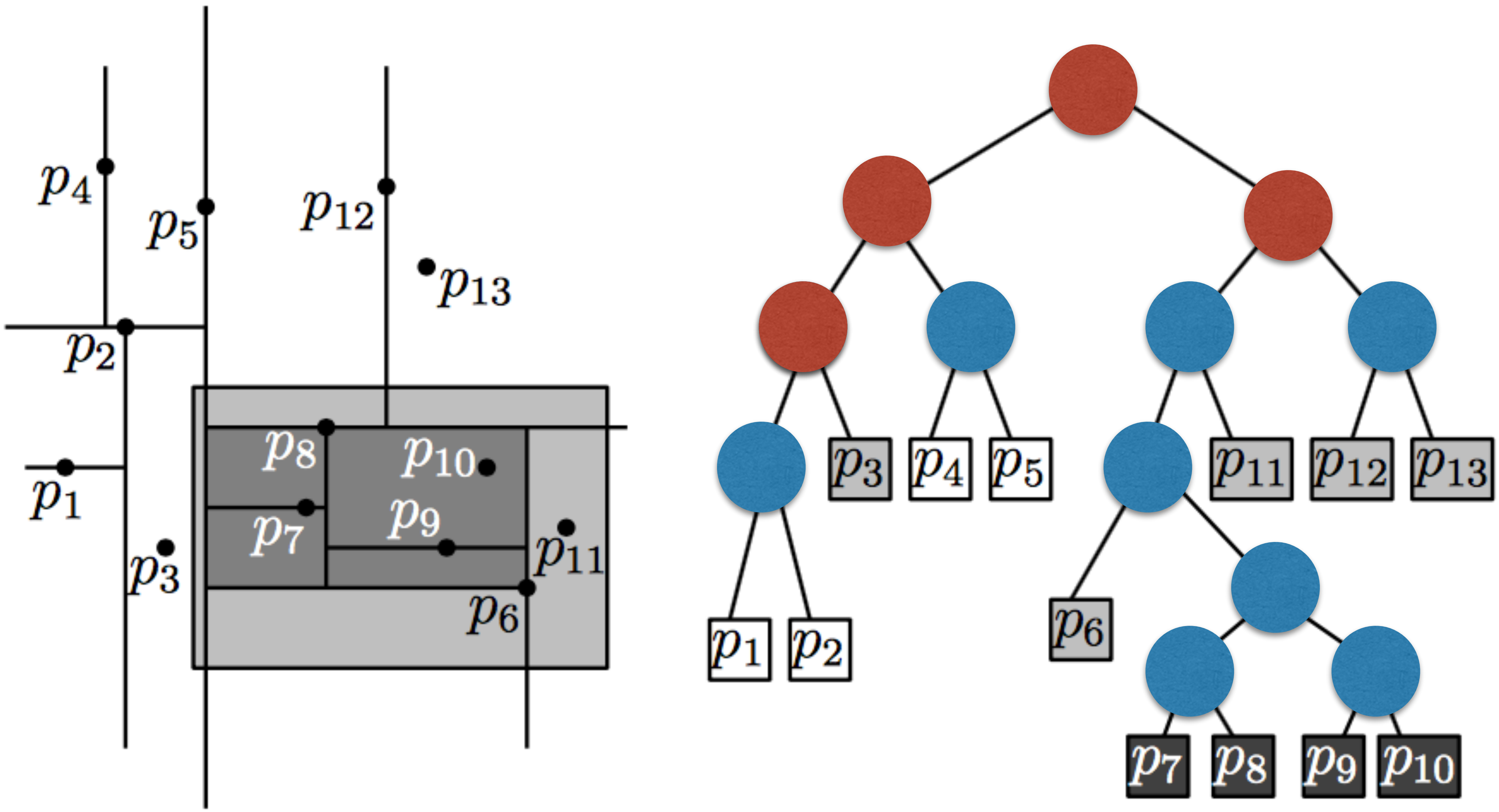
Which nodes are visited in this tree when answering the query?



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

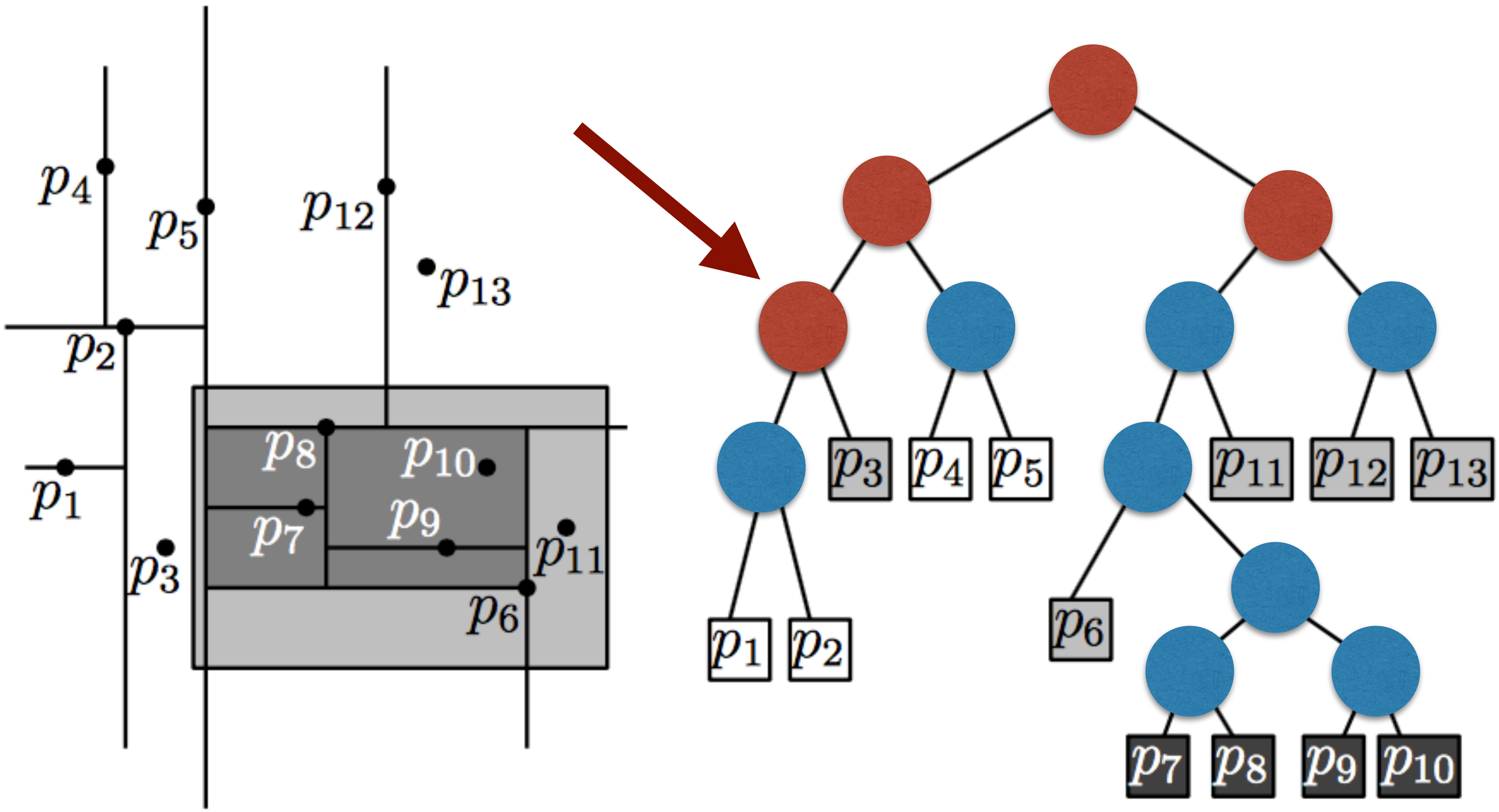
Which nodes are visited in this tree when answering the query?



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

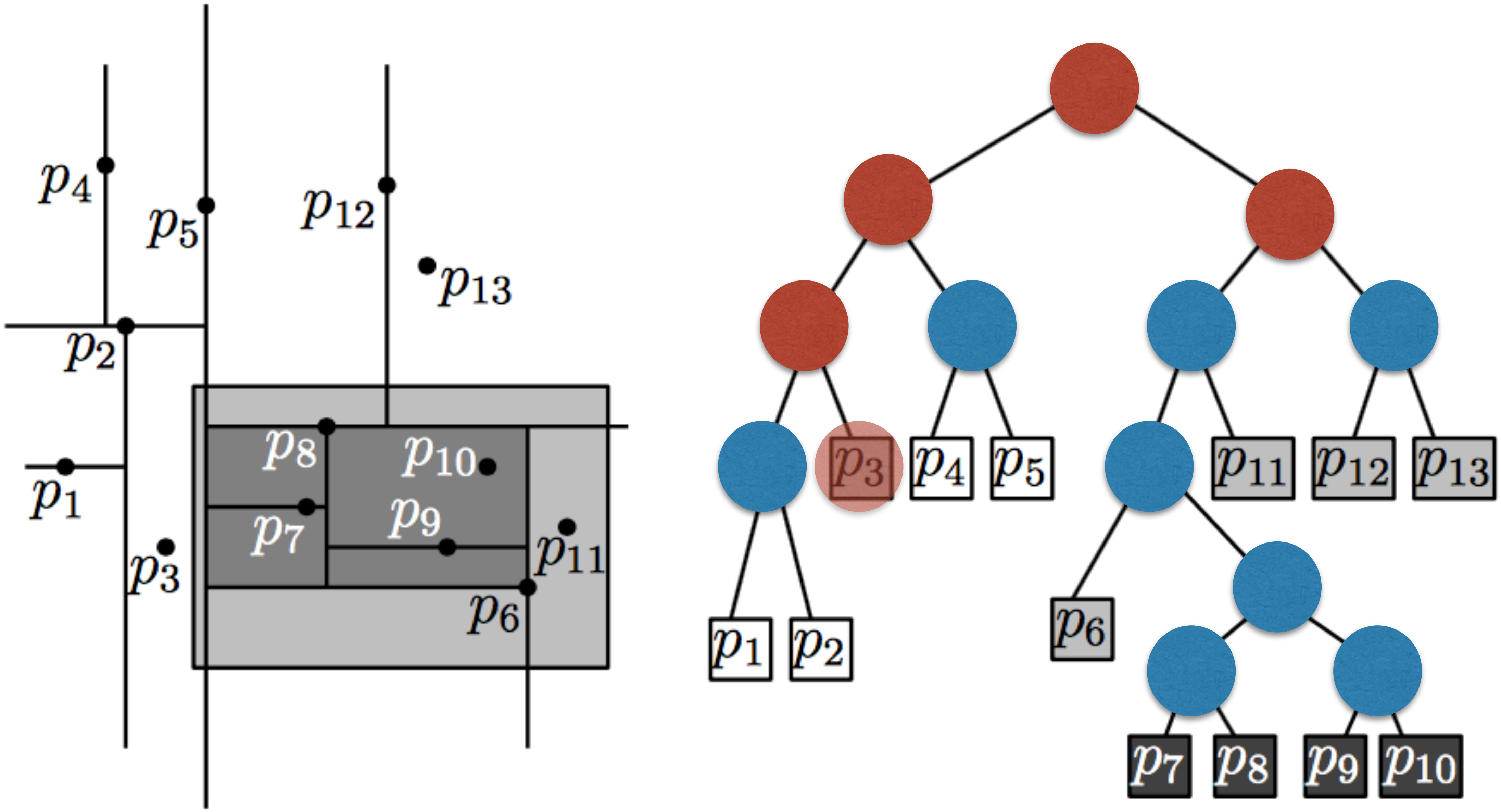
Which nodes are visited in this tree when answering the query?



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

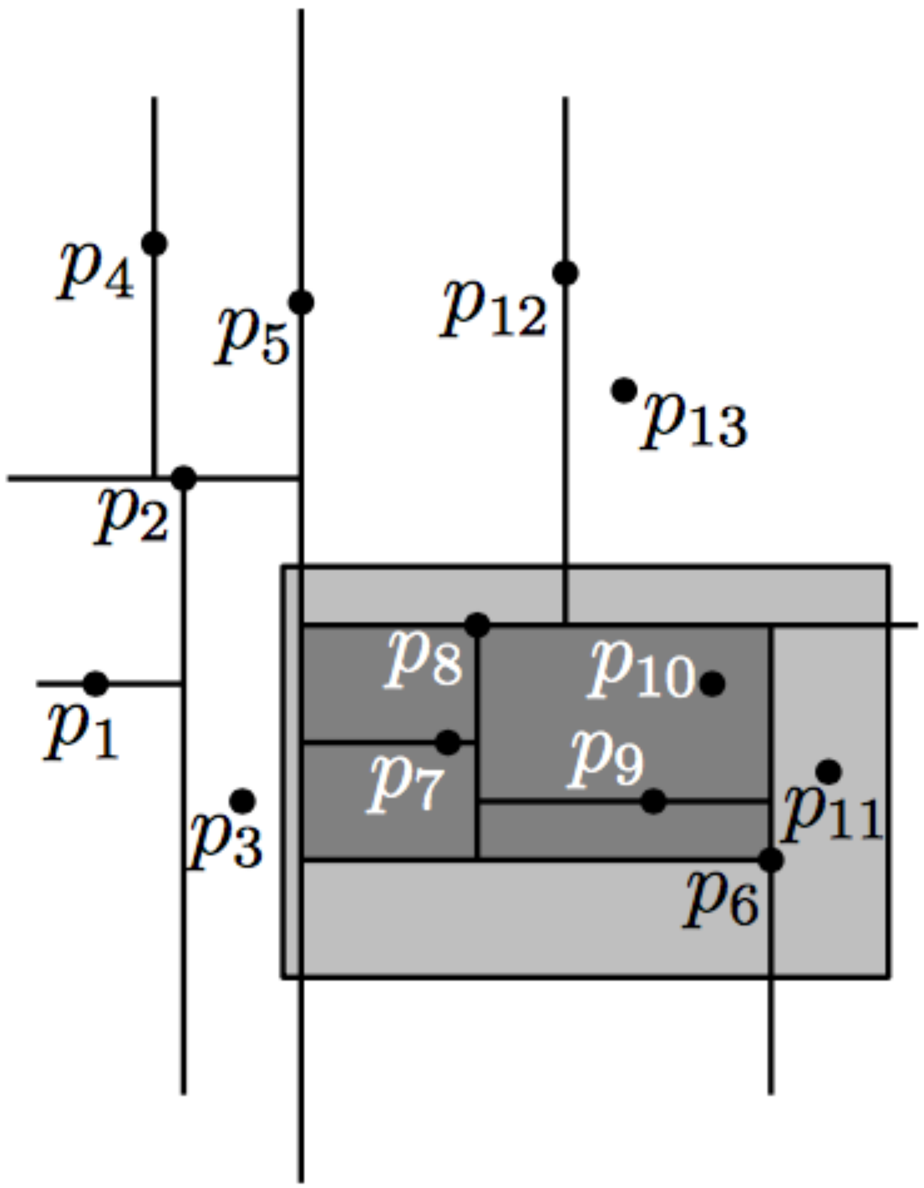
Which nodes are visited in this tree when answering the query?

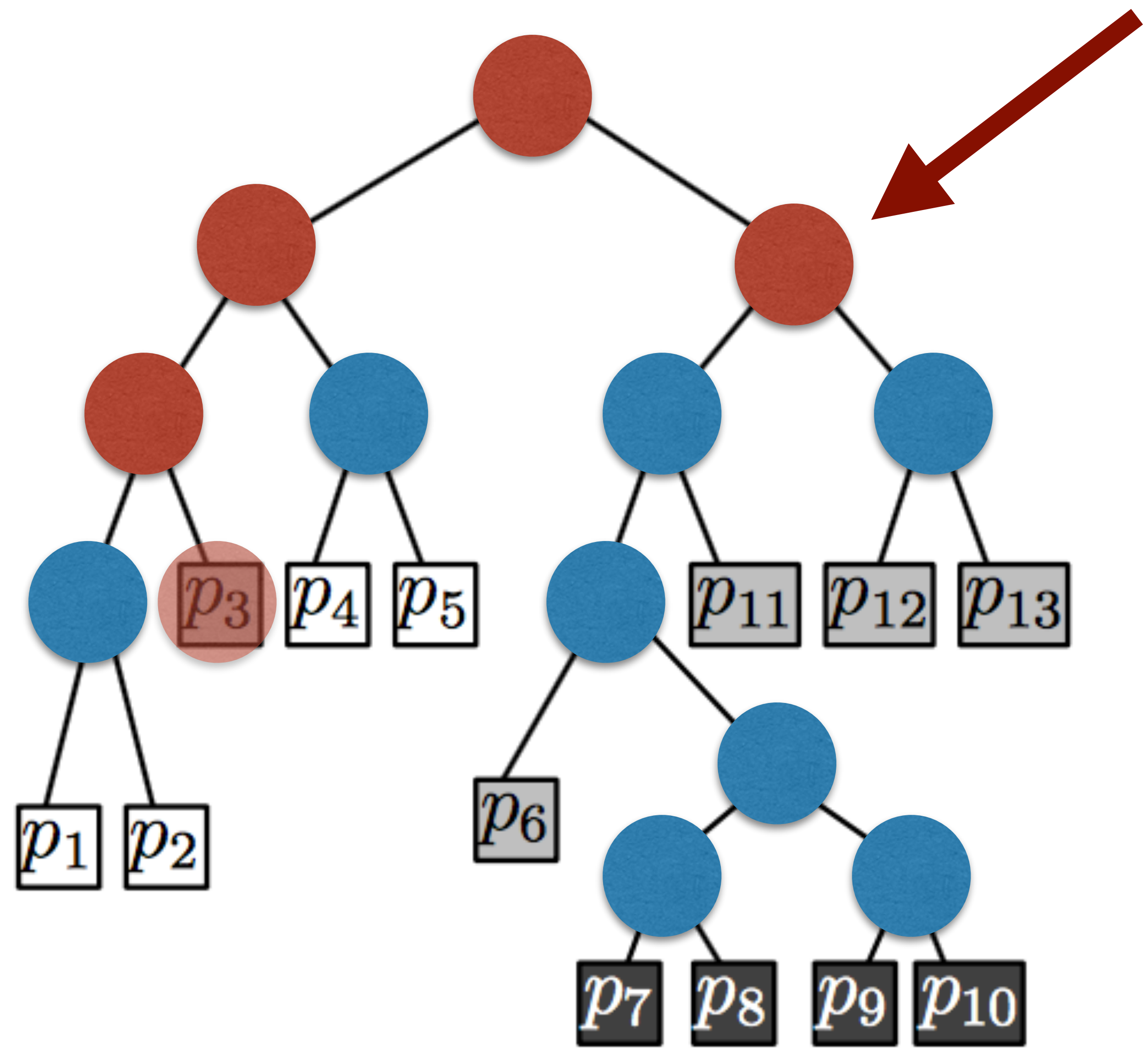


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

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

Which nodes are visited in this tree when answering the query?

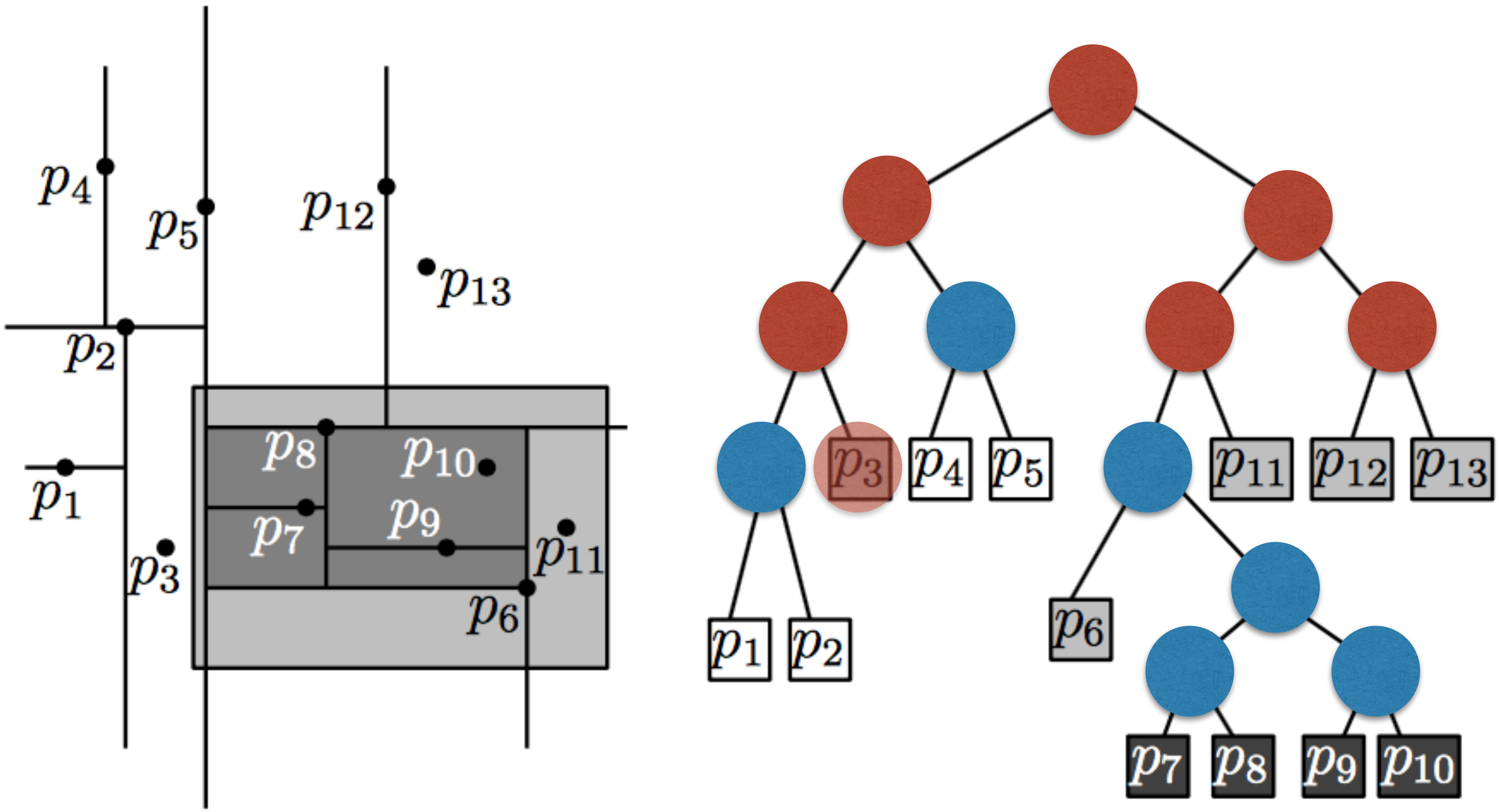




screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

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

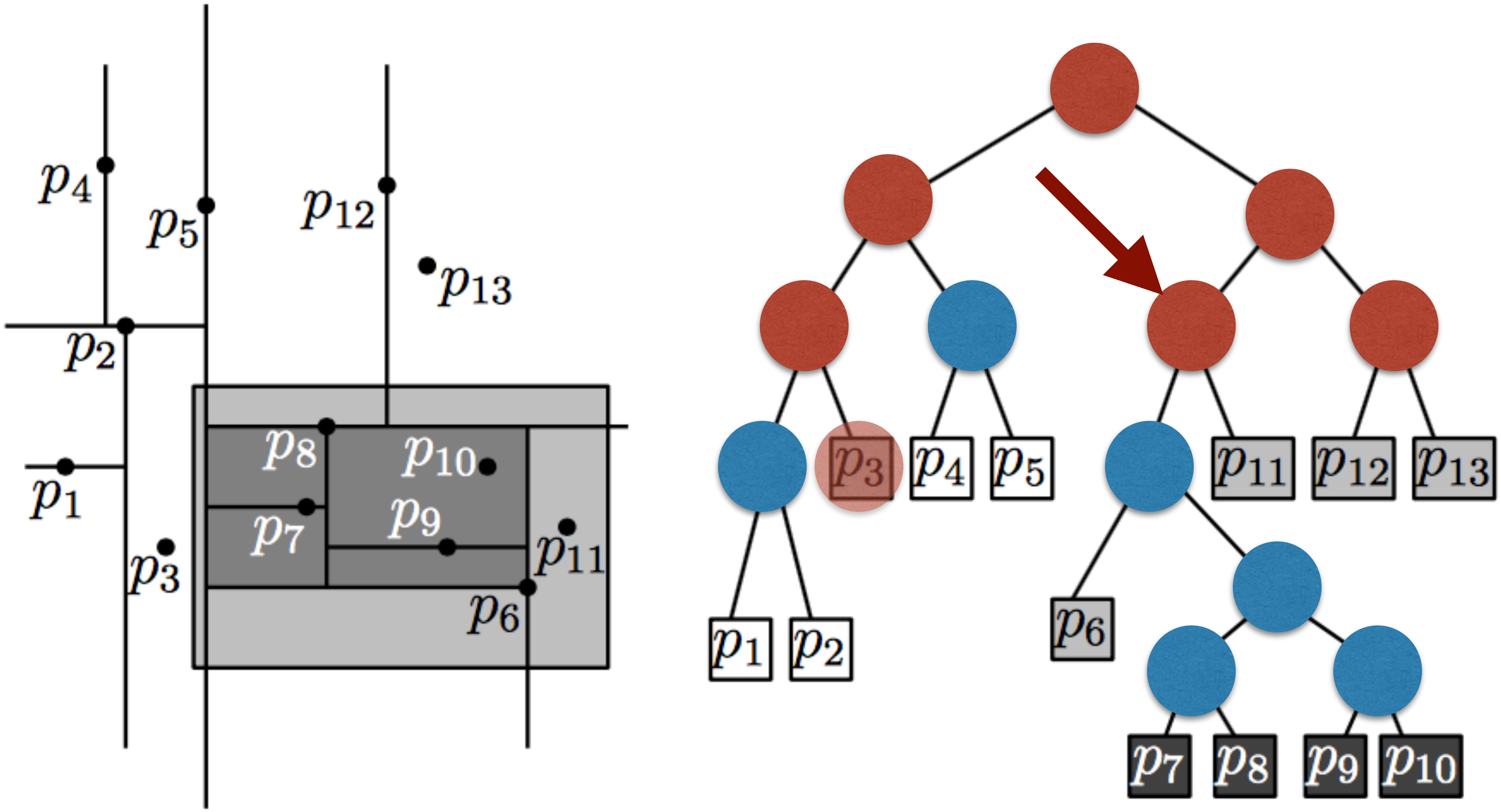
Which nodes are visited in this tree when answering the query?



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

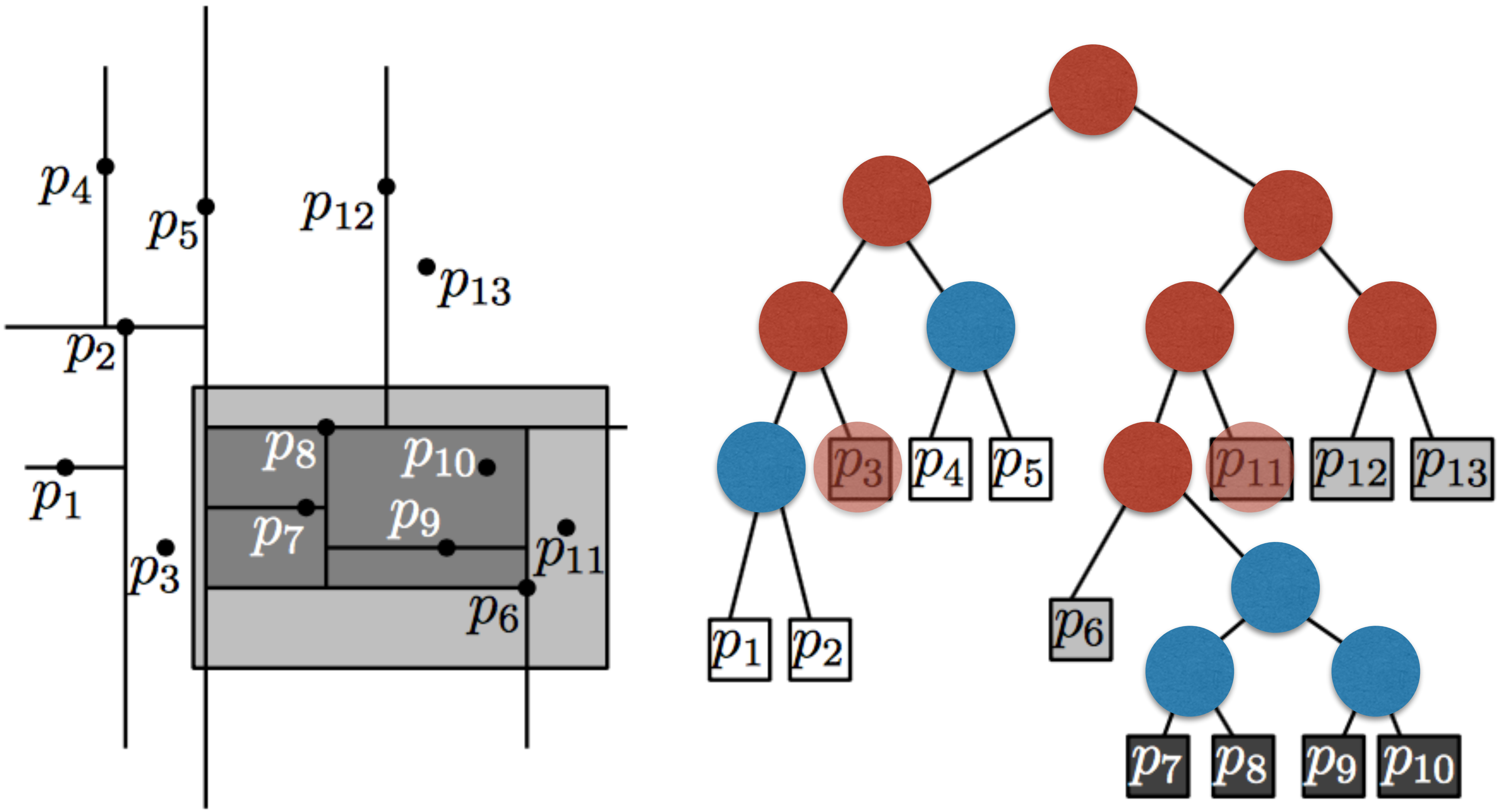
Which nodes are visited in this tree when answering the query?



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

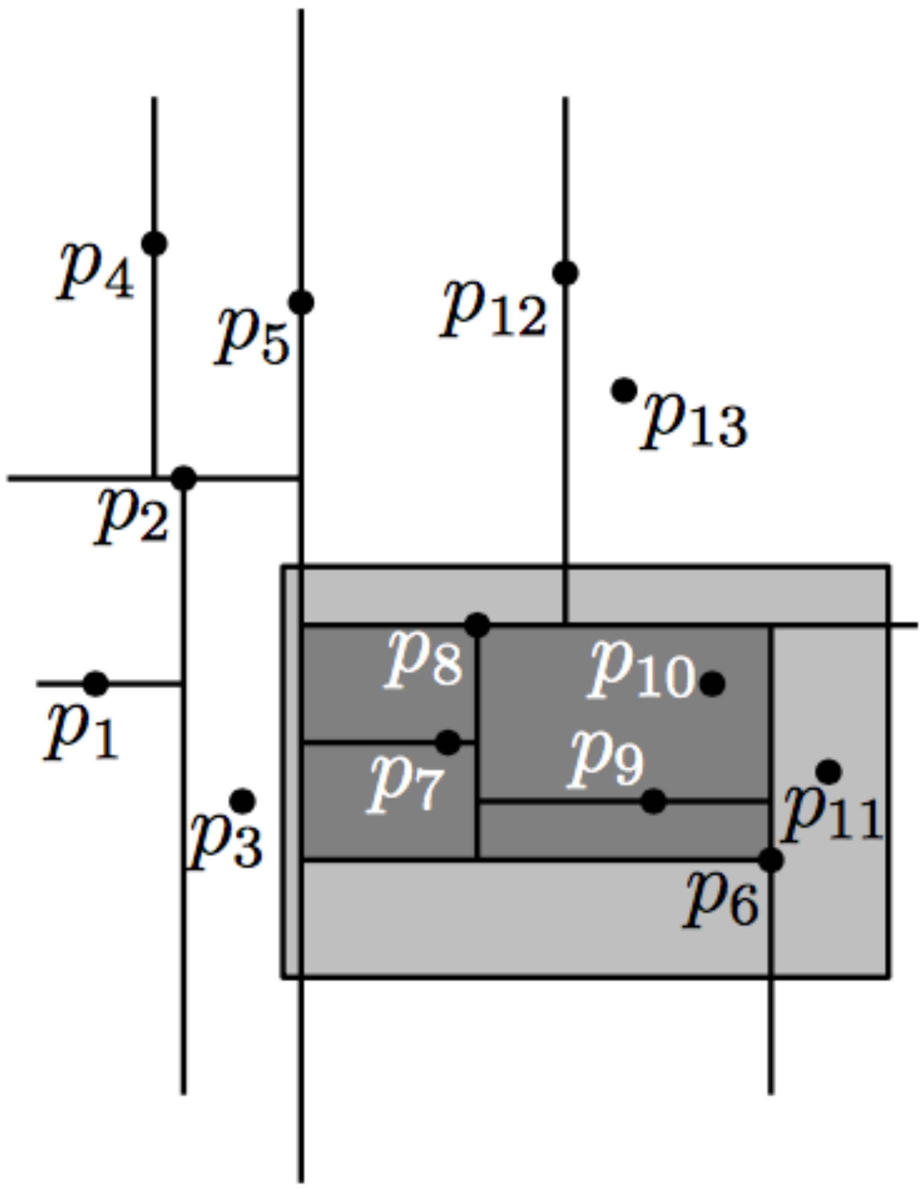
Which nodes are visited in this tree when answering the query?

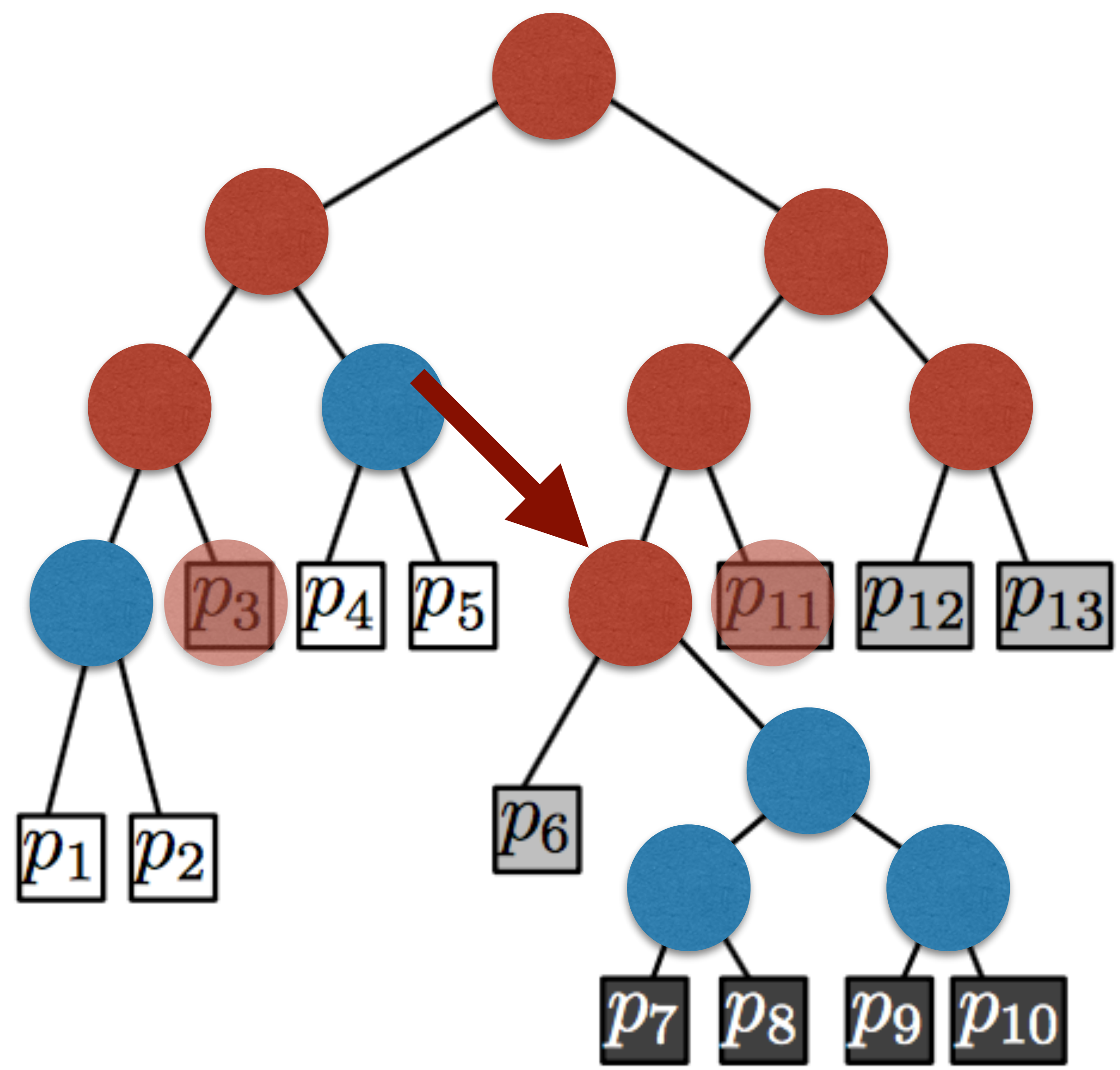


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

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

Which nodes are visited in this tree when answering the query?

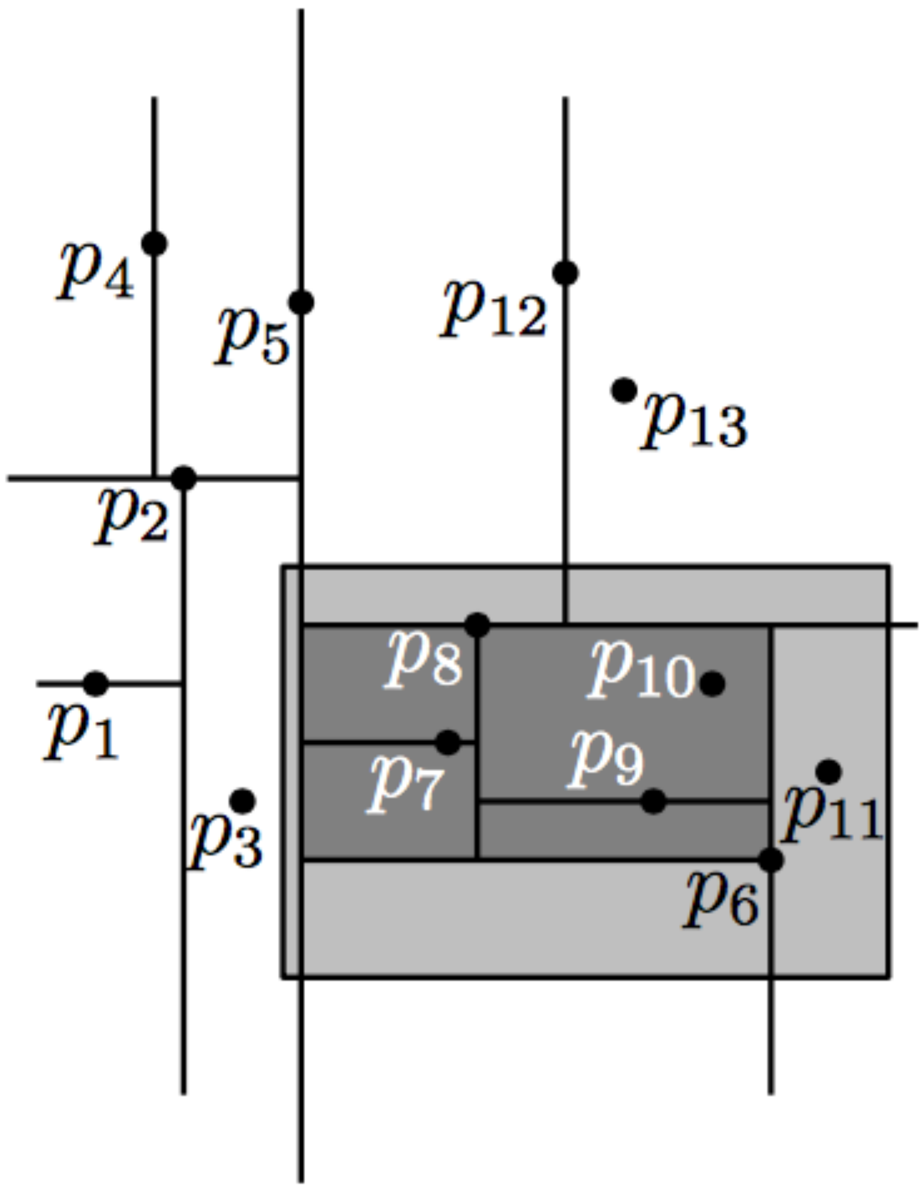


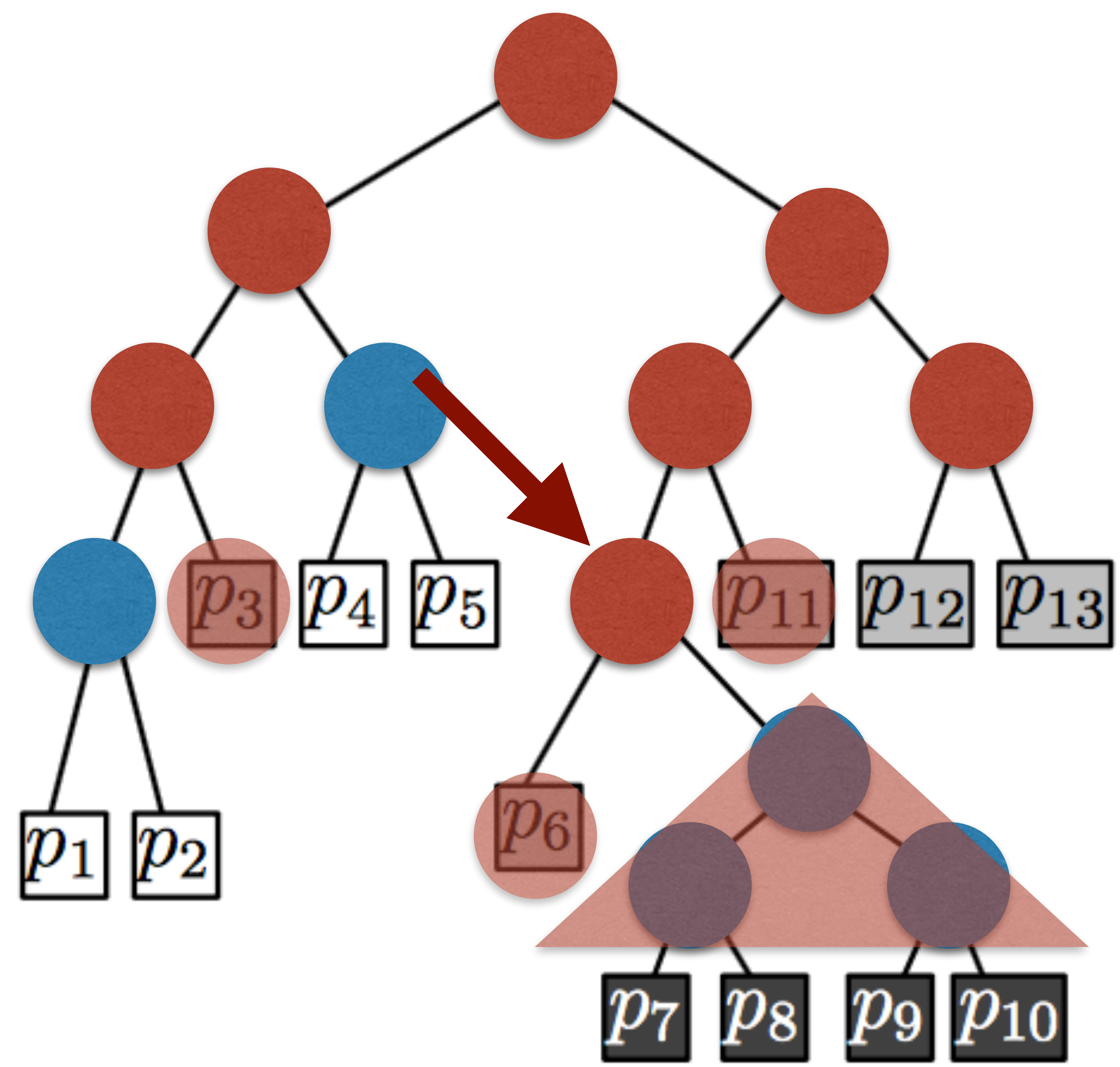


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

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

Which nodes are visited in this tree when answering the query?

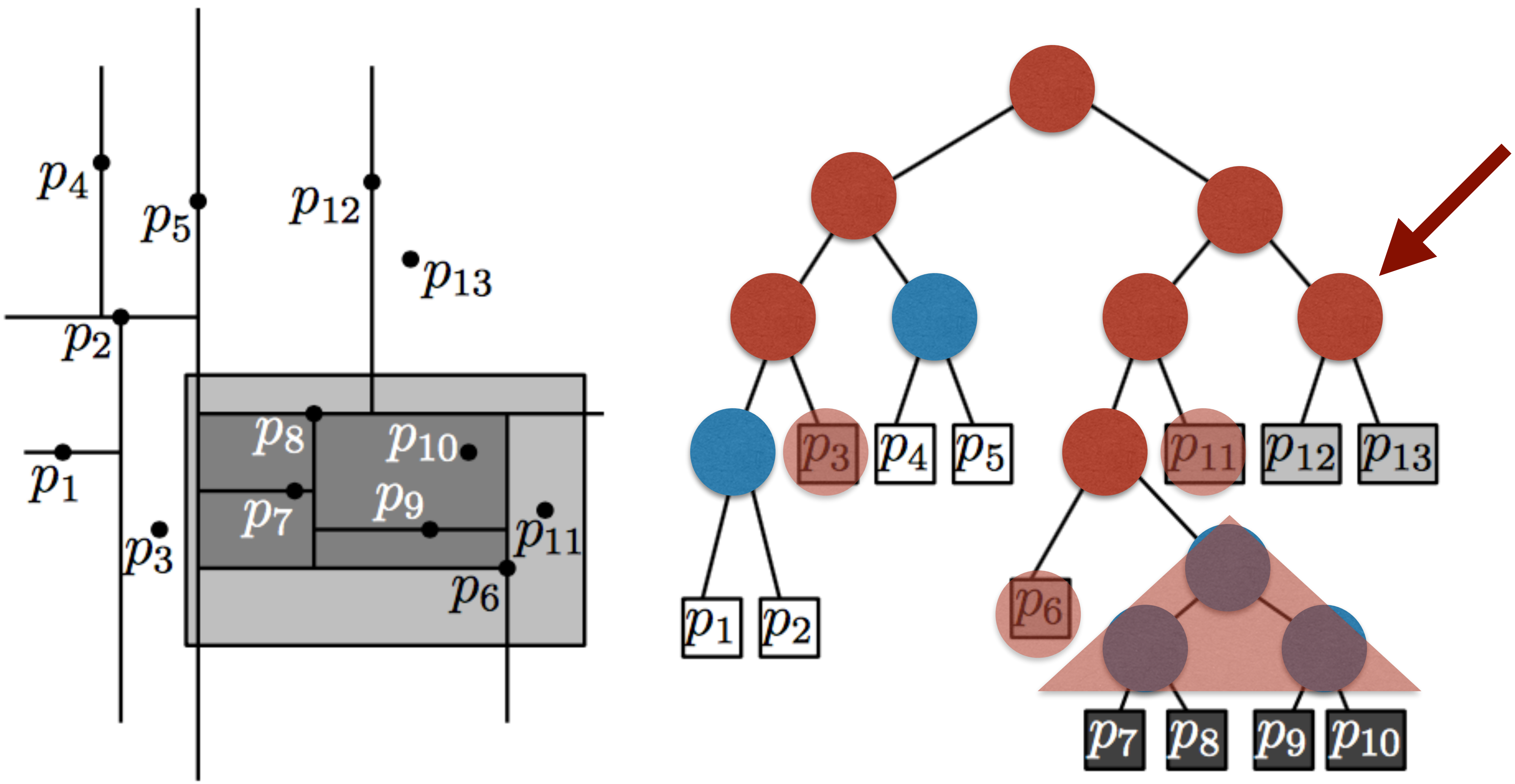




screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

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

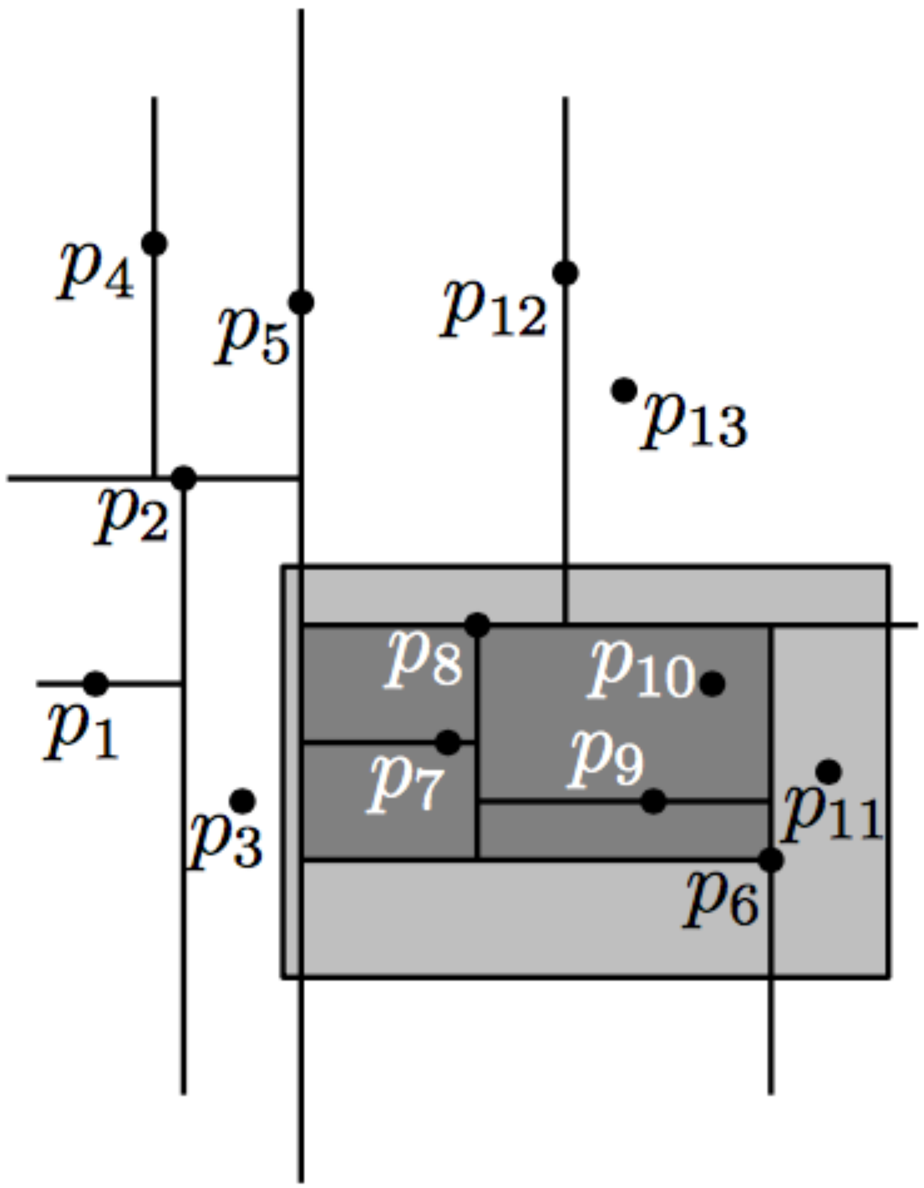
Which nodes are visited in this tree when answering the query?

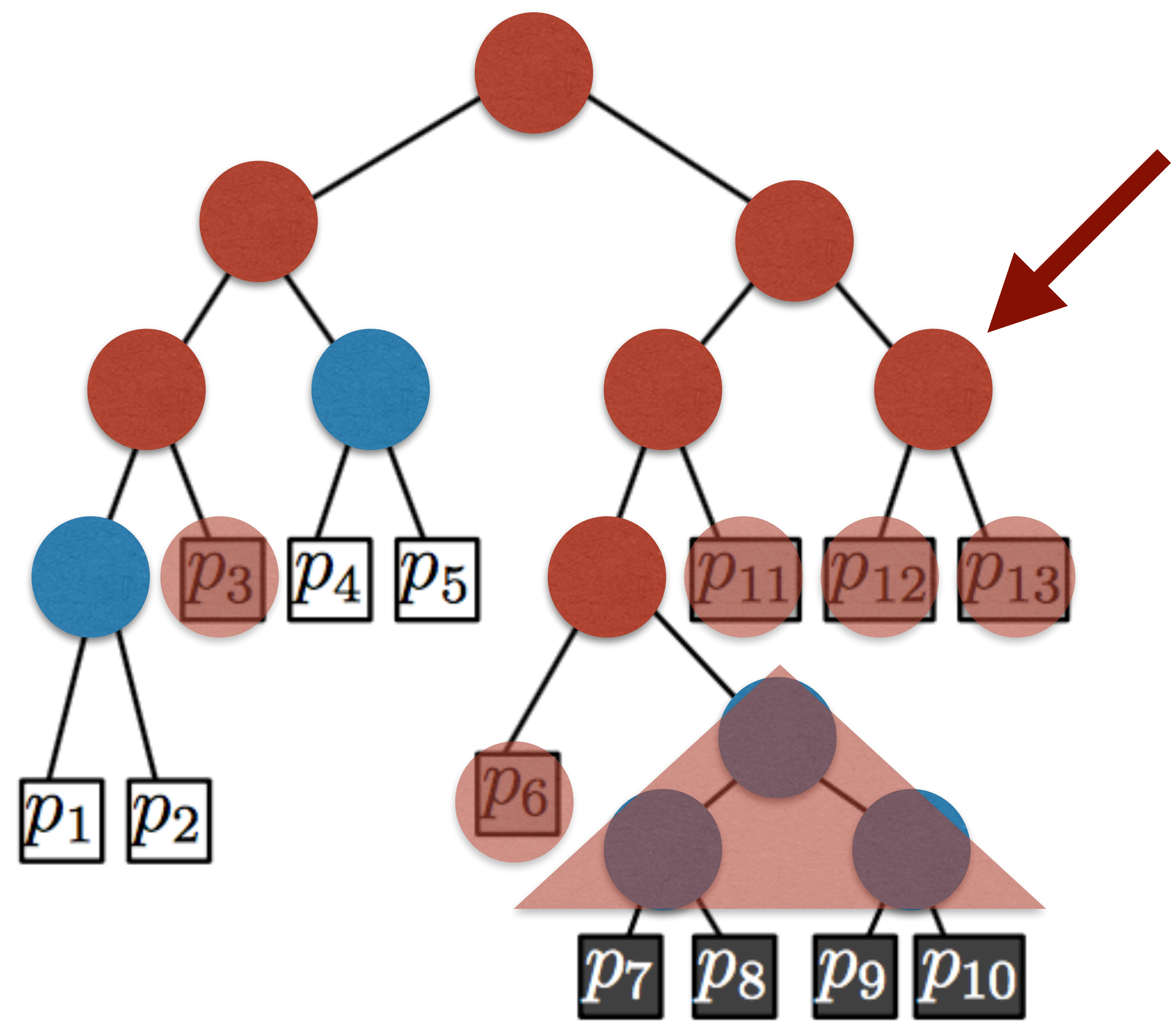


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

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

Which nodes are visited in this tree when answering the query?



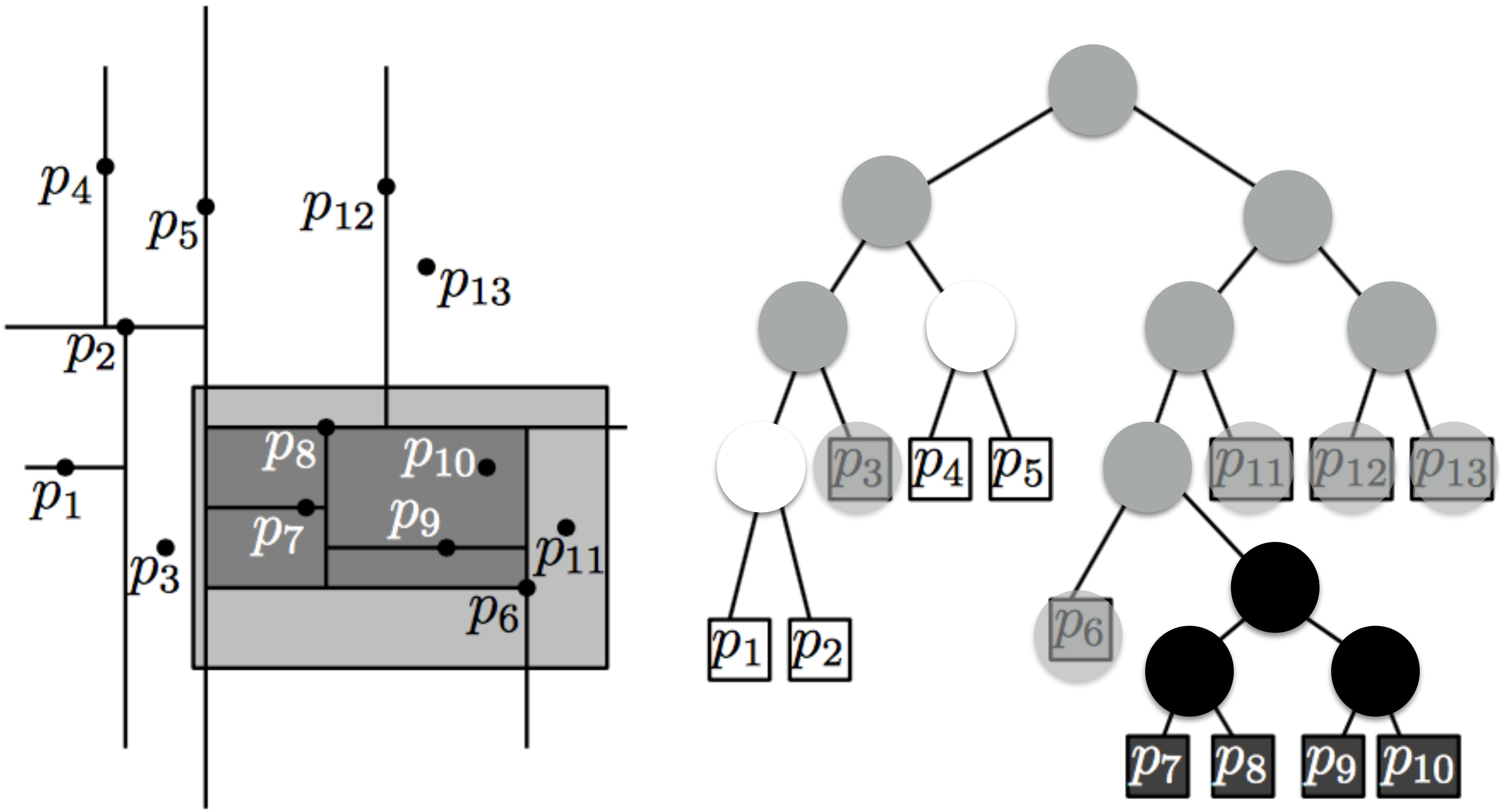


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

nodes never visited by the query

visited by the query, but unclear if they lead to output

visited by the query, whole subtree is output



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

nodes never visited by the query

visited by the query, but unclear if they lead to output

visited by the query, whole subtree is output

Consider region(node) and how it intersects range R

nodes never visited by the query

visited by the query, but unclear if they lead to output

visited by the query, whole subtree is output

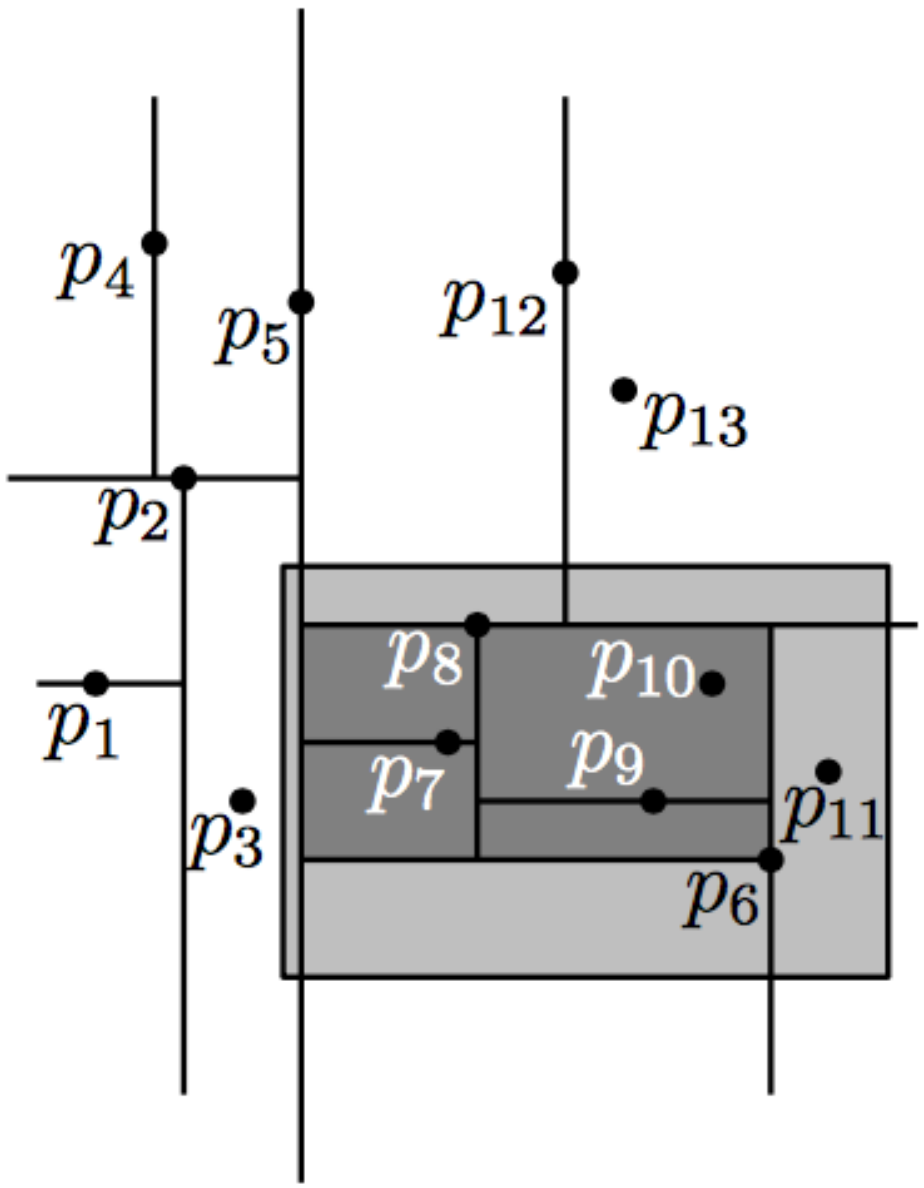
R does not intersect region(v)

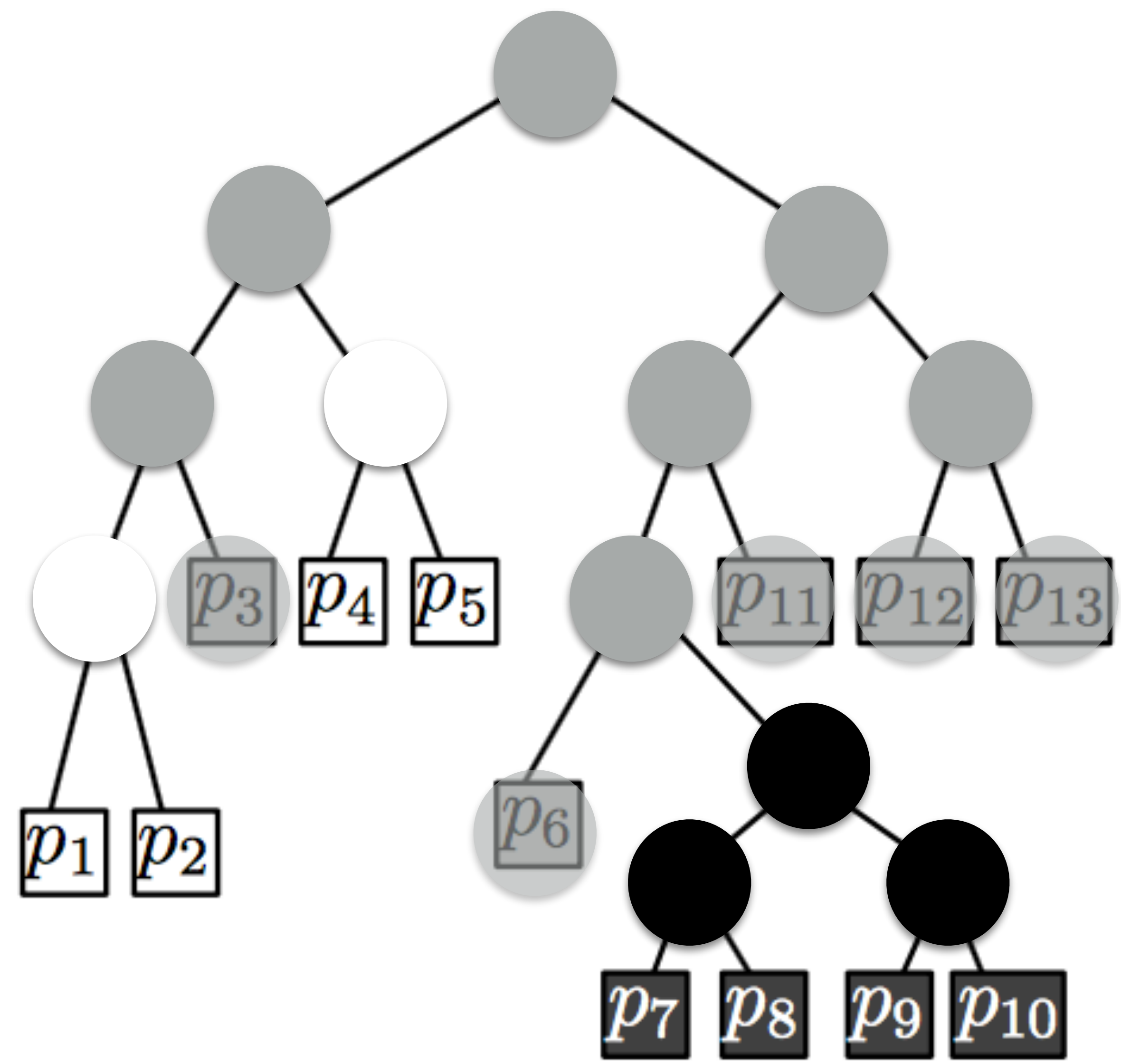
Consider region(node) and how it intersects range R

region(v) is contained in R

R intersects region(v), but region(v) not contained in R

Time to answer range query = O(nb.black + nb.grey nodes)



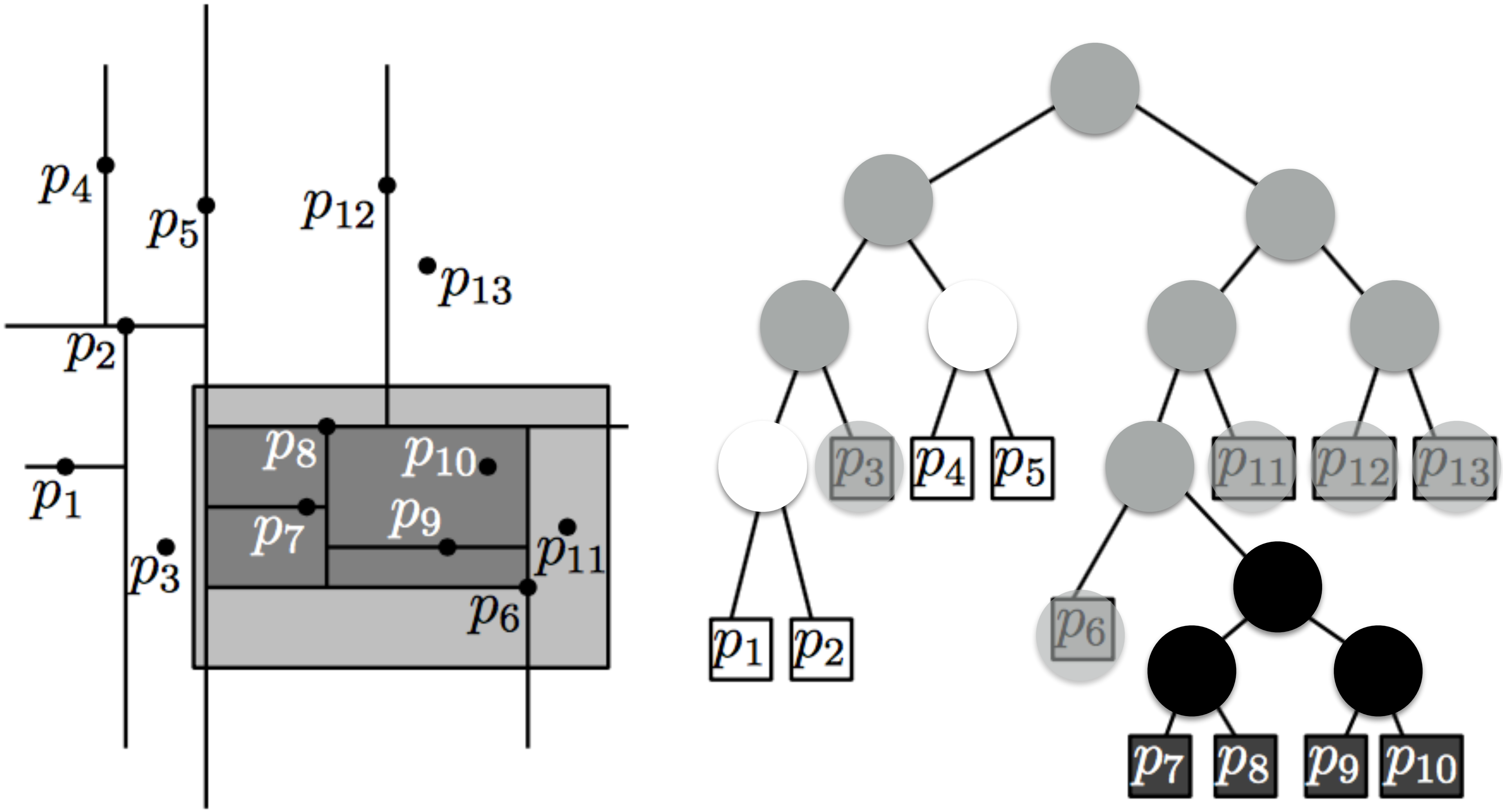


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

How many black nodes?

Observation: Each black leaf contain a point that's reported => k leaves

Can be shown that the nb. of internal black nodes is k-1



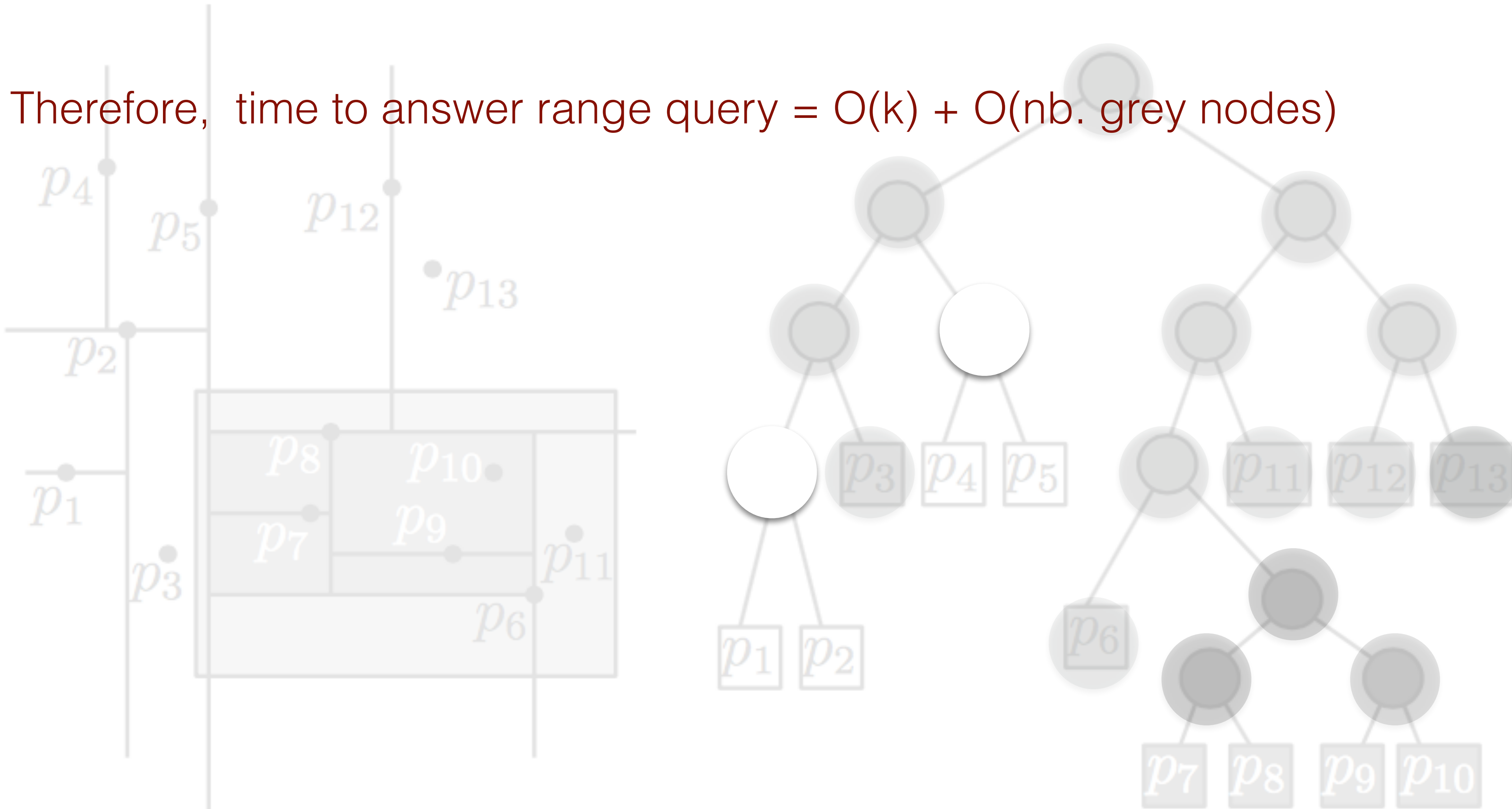
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

How many black nodes?

Observation: Each black leaf contain a point that's reported => k leaves

Can be shown that the nb. of internal black nodes is k-1

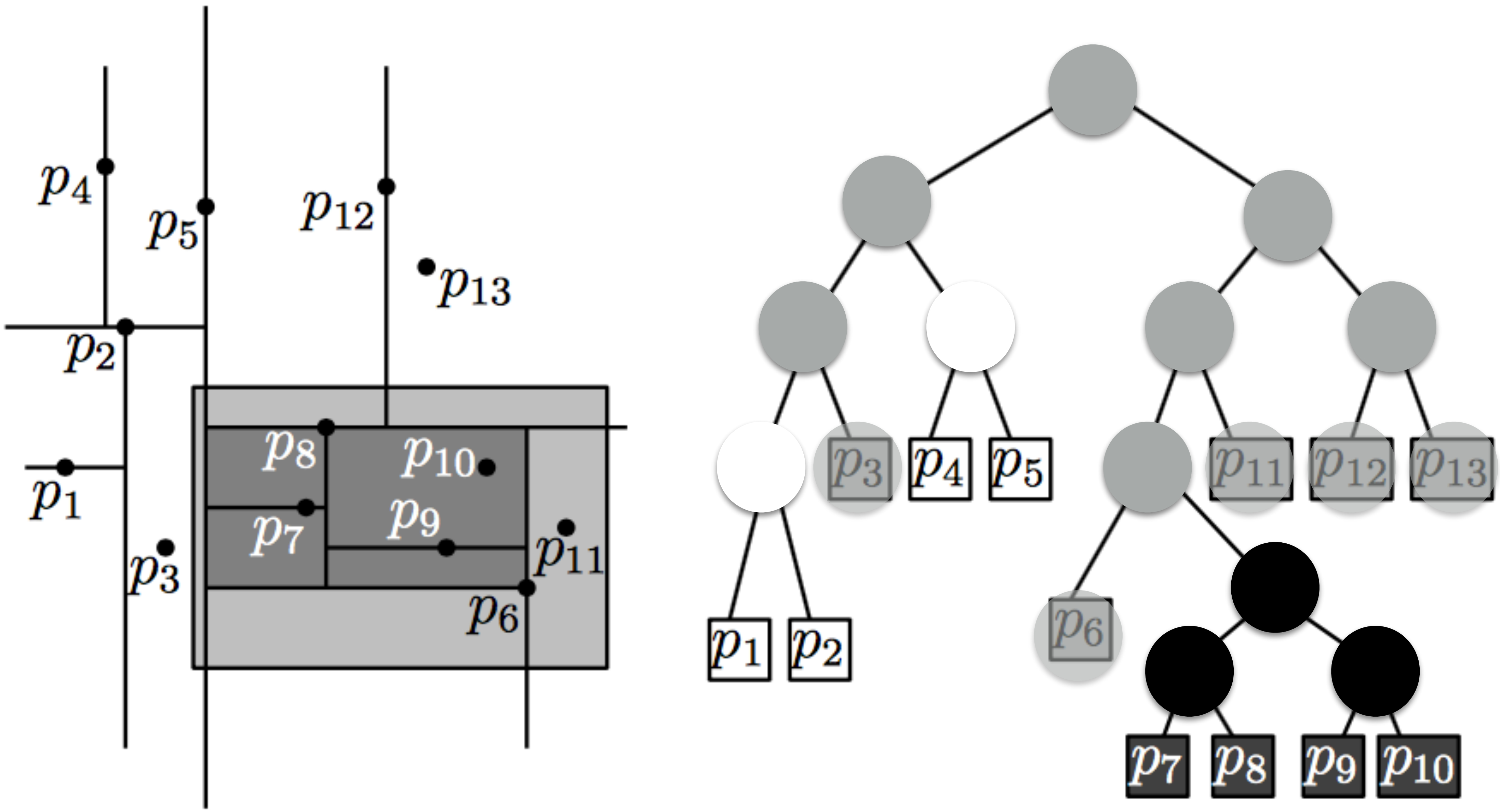
Therefore, time to answer range query = O(k) + O(nb. grey nodes)



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

How many grey nodes?

region(v) intersects R, but region(v) not contained in R



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

How many grey nodes?

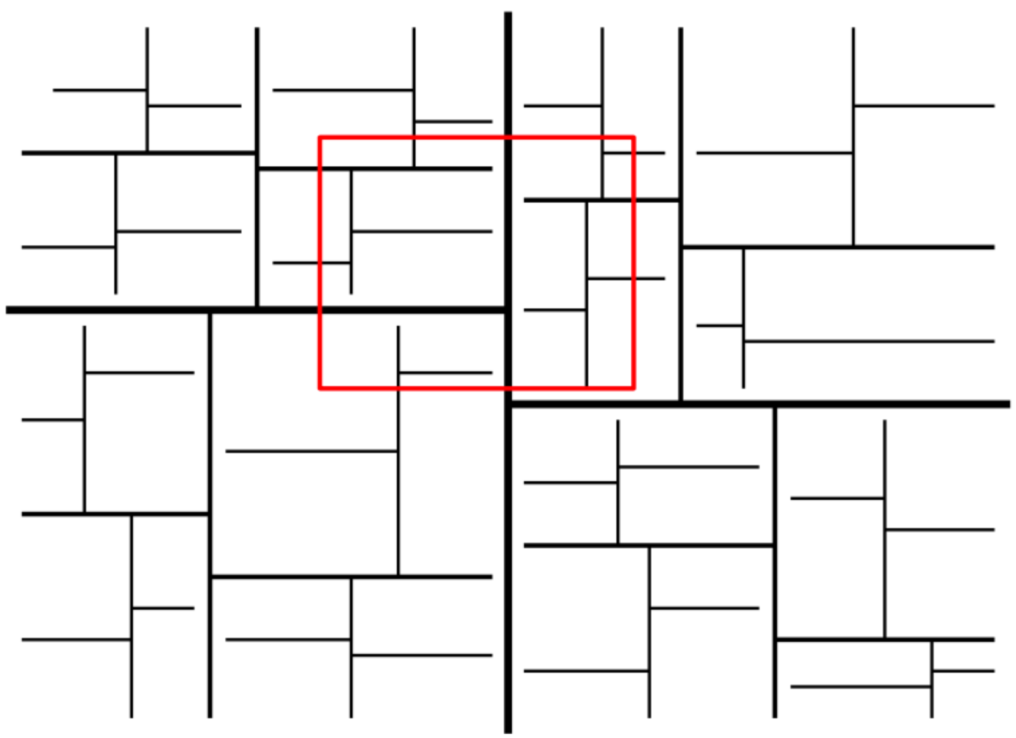
region(v) intersects R, but region(v) not contained in R

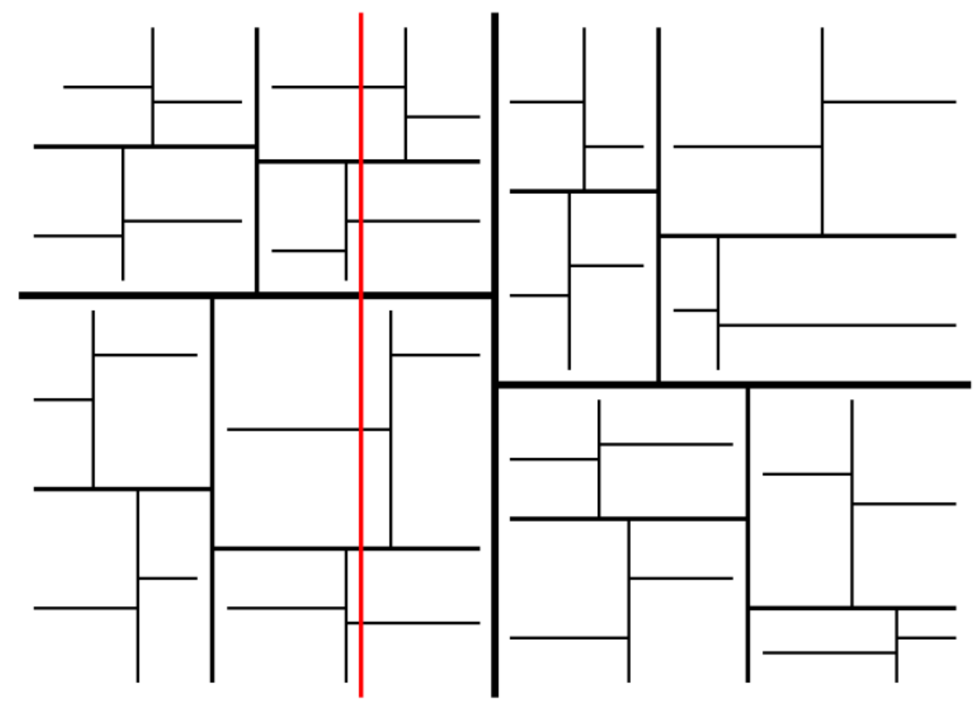
How many nodes are such that the boundary of their region intersects the boundary of the range?

How many grey nodes?

region(v) intersects R, but region(v) not contained in R

How many nodes are such that the boundary of their region intersects the boundary of the range?



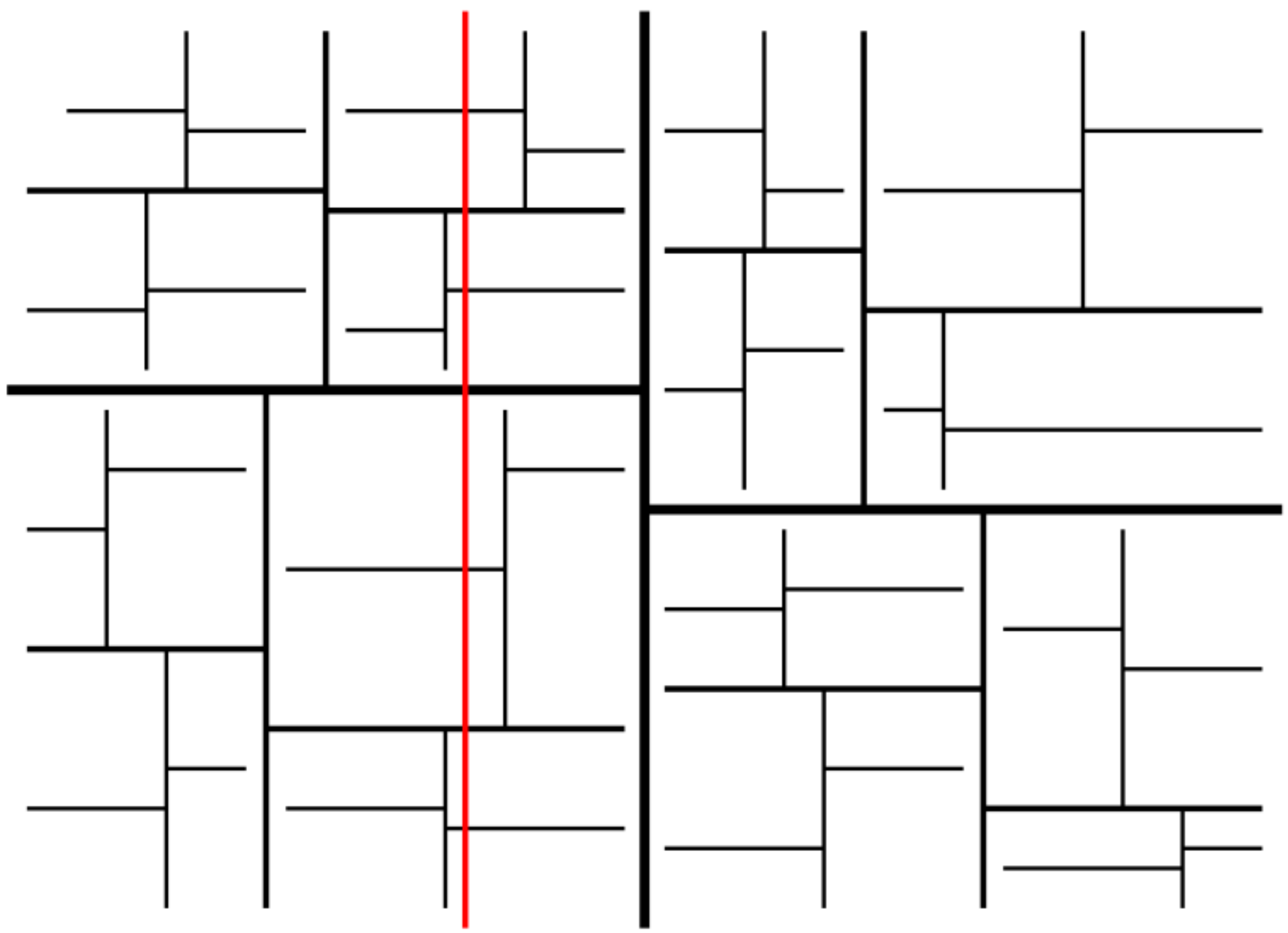


Simplified problem:

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?

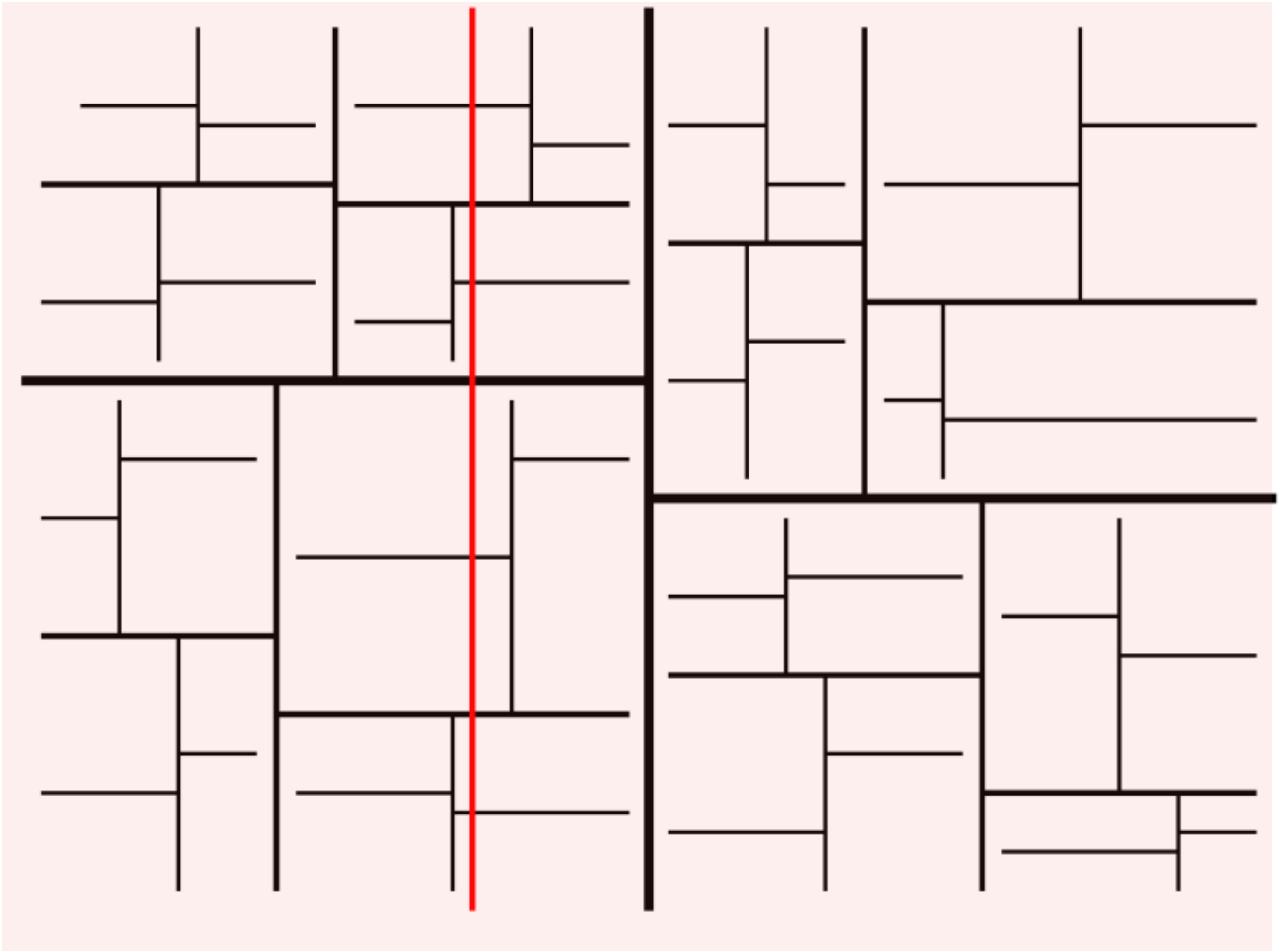
We'll think recursively, starting at the root:



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

We'll think recursively, starting at the root:

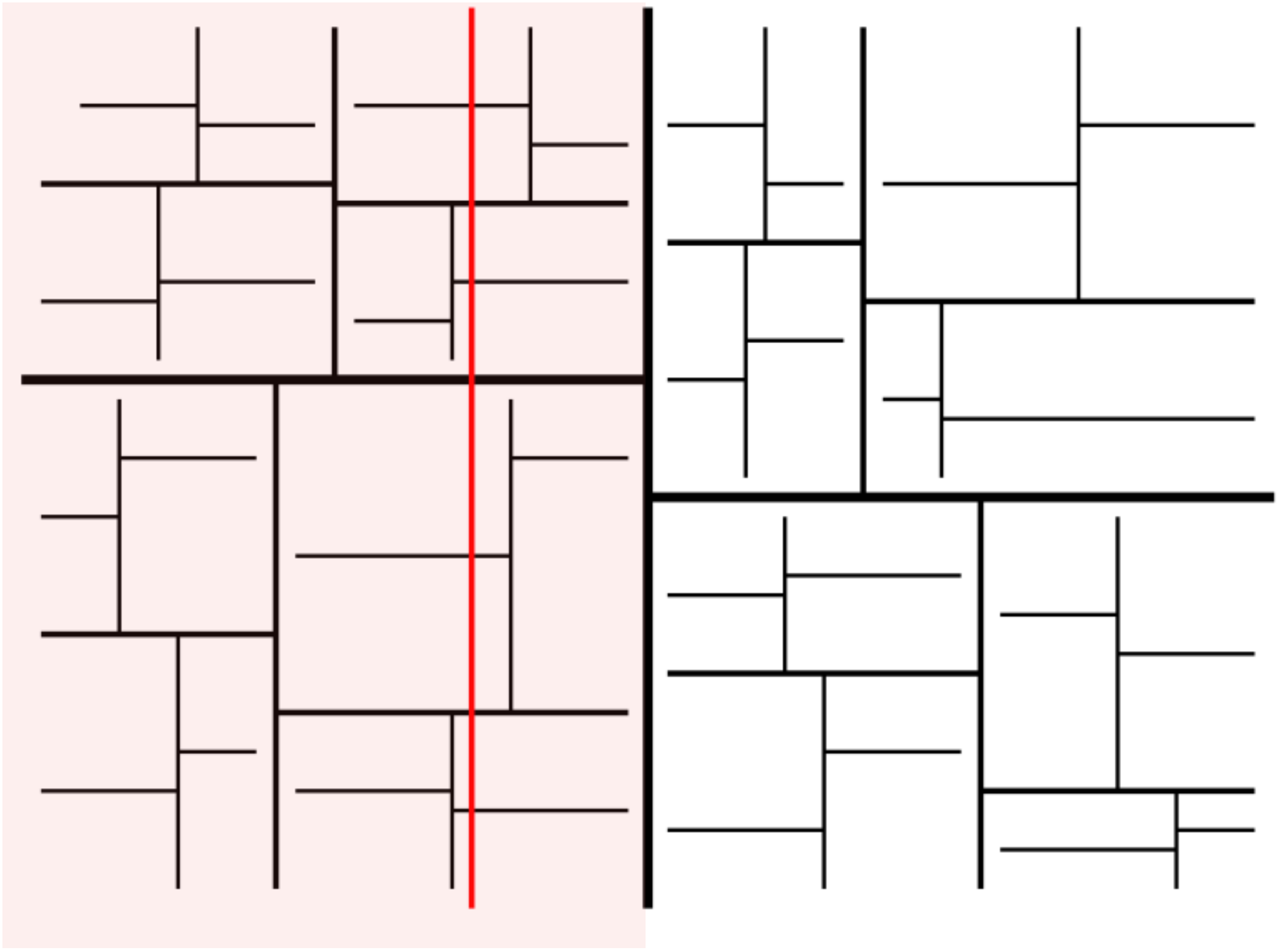
- depth=0: region(root) intersects l $+1$



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

We'll think recursively, starting at the root:

- depth=1: only one of {left, right} child intersects l $+1$

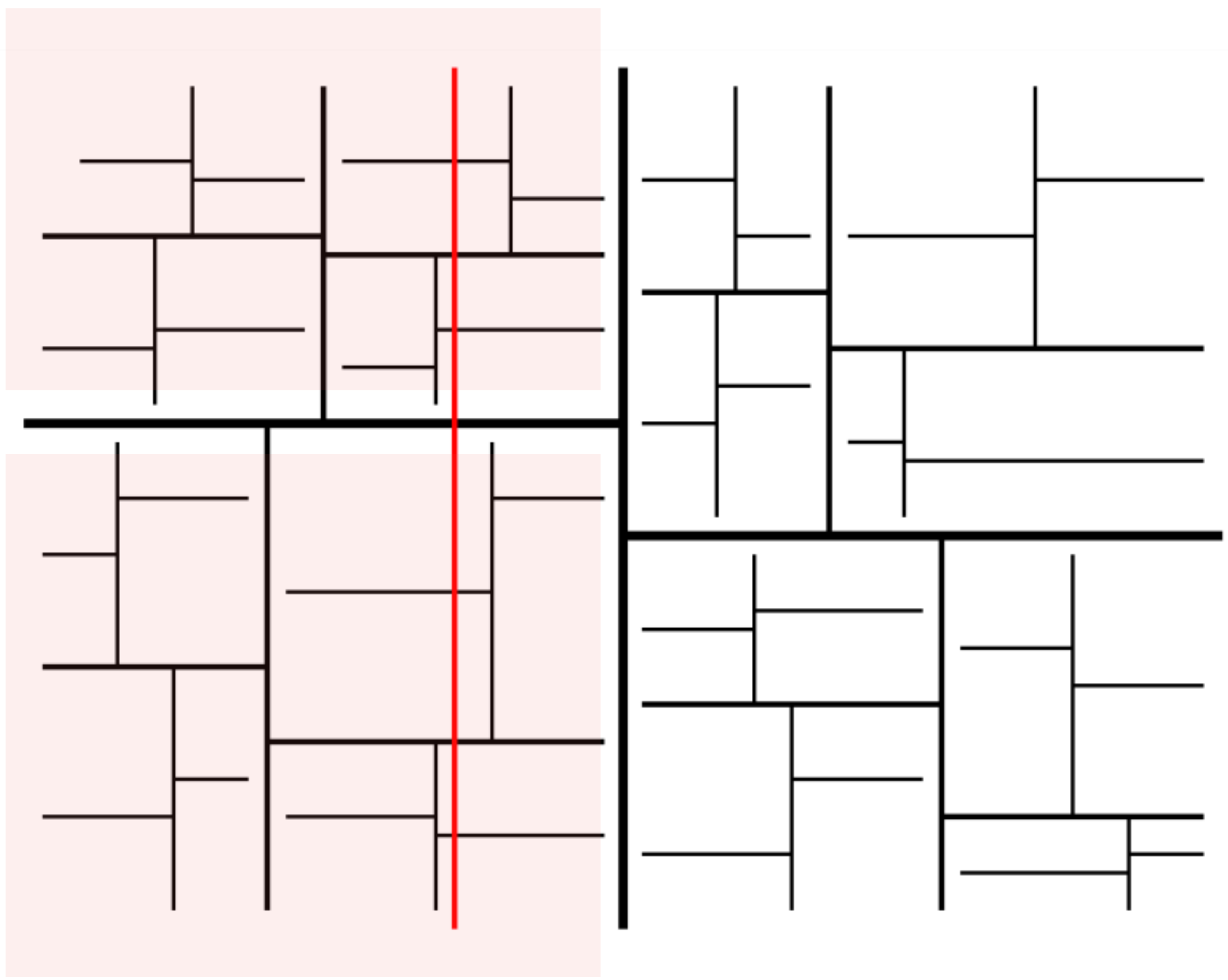


Number of nodes v such that region(v) intersects a vertical line l?

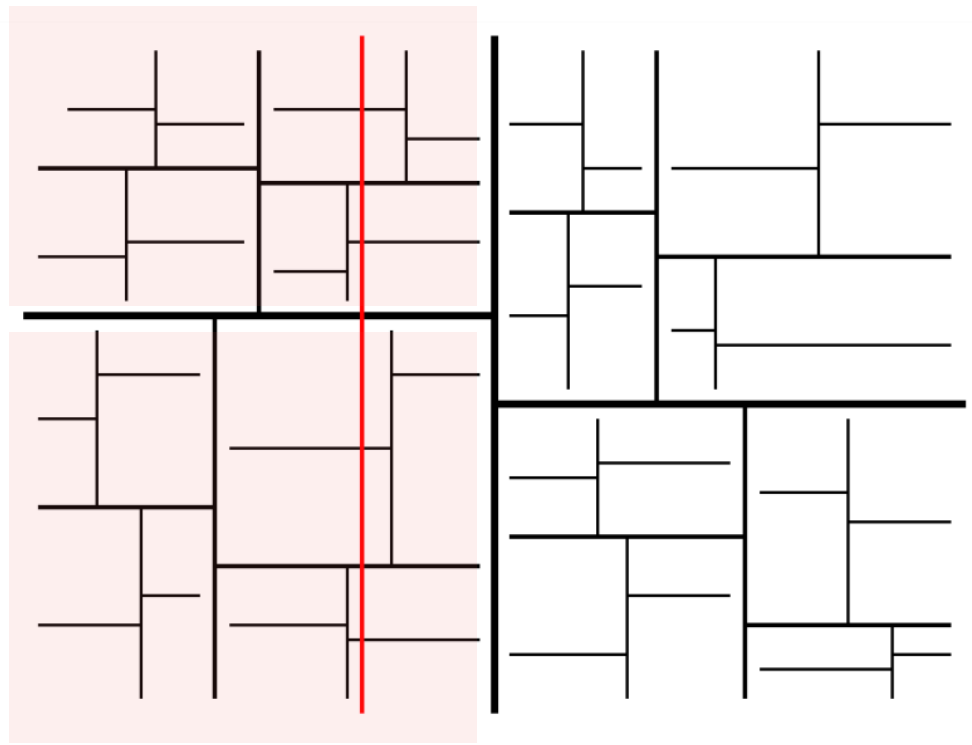
We'll think recursively, starting at the root:

- depth=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 $G(1) = 1$



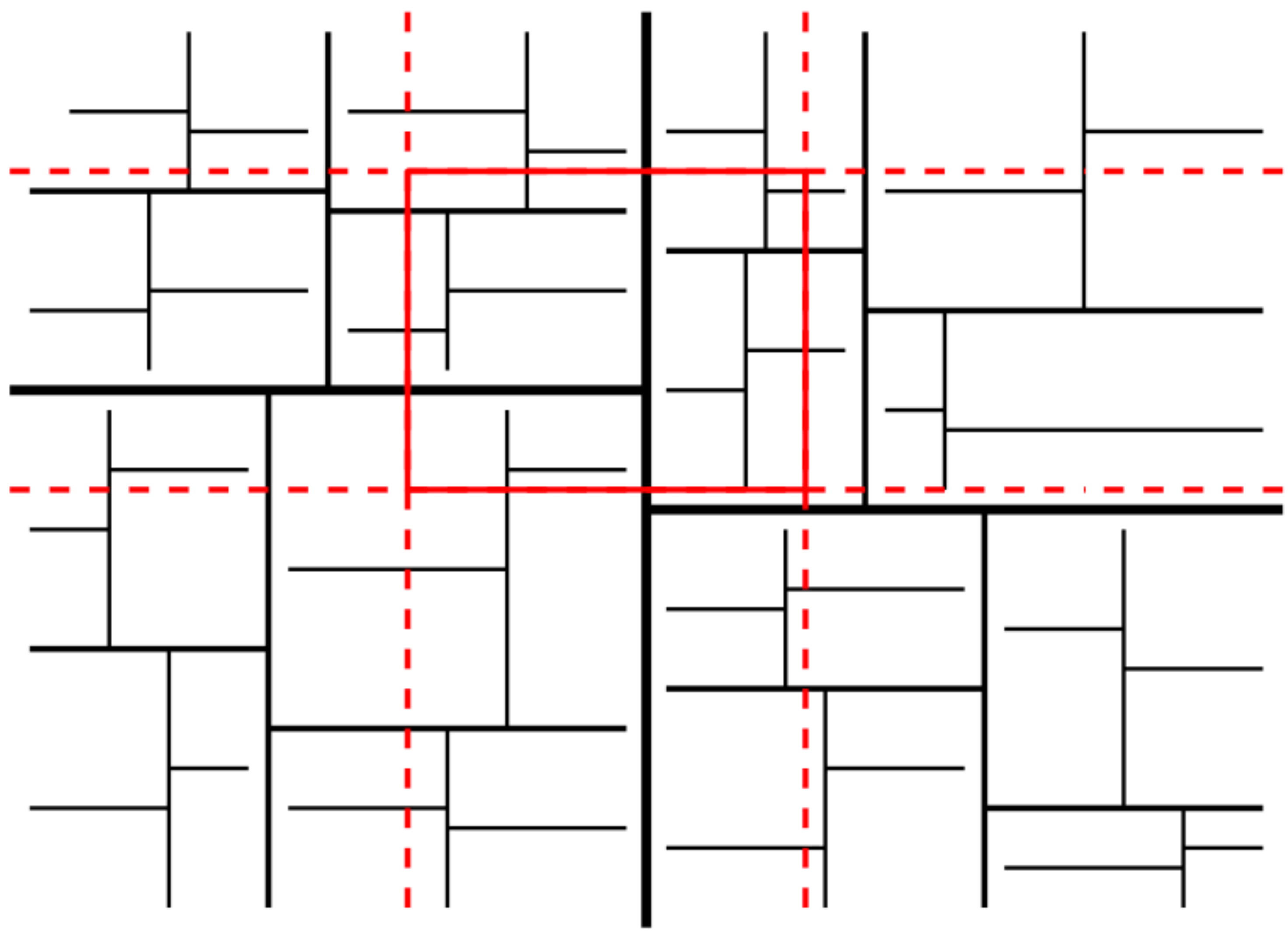
- This solves to $G(n) = O(\sqrt{n})$

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

What we got so far:

- The number of grey nodes 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?

- The nb. grey nodes for a query rectangle is at most the nb. grey nodes for two vertical and two horizontal lines, so it is at most $4 \times O(\sqrt{n}) = O(\sqrt{n})$



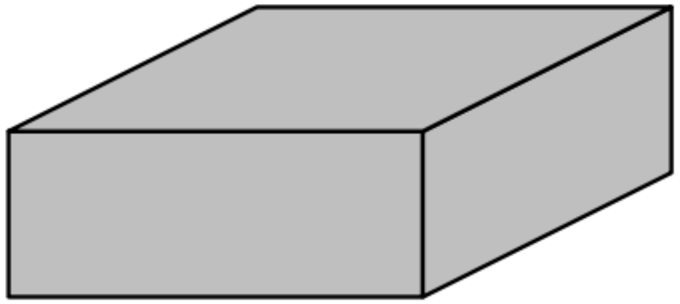
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

- **Theorem**: A set of n points in the plane can be preprocessed in $O(n \lg n)$ time into a data structure of $O(n)$ size so that any 2D range query can be answered in $O(\sqrt{n} + k)$ time, where k is the nb. points reported.

kd-tree (2d-tree)

kd-tree in higher dimensions

kd-tree in 3D: 3d-tree



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

- Construction: The construction of a 2D kd-tree extends to 3D
- Answering range queries: Exactly the same as in 2D
- Analysis:

Let $G_3(n)$ be the number of grey nodes for a query with an axes-parallel plane in a 3D kd-tree

$$G_3(1) = 1$$

$$G_3(n) = 4 \cdot G_3(n/8) + O(1)$$

Higher dimensions

Theorem: A set of n points in d-space can be preprocessed in $O(n \log n)$ time into a data structure of $O(n)$ size so that any d-dimensional range query can be answered in $O(n^{1-1/d} + k)$ time, where k is the number of answers reported