Computational Geometry
[csci 3250]

Laura Toma

Bowdoin College

Geometric data

- points (location, ..)
- segments (roads, …)
- polygons, polyhedrons
- maps/meshes

“global” problems

- closest pair
- convex hull
- all intersections

“search” problems

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

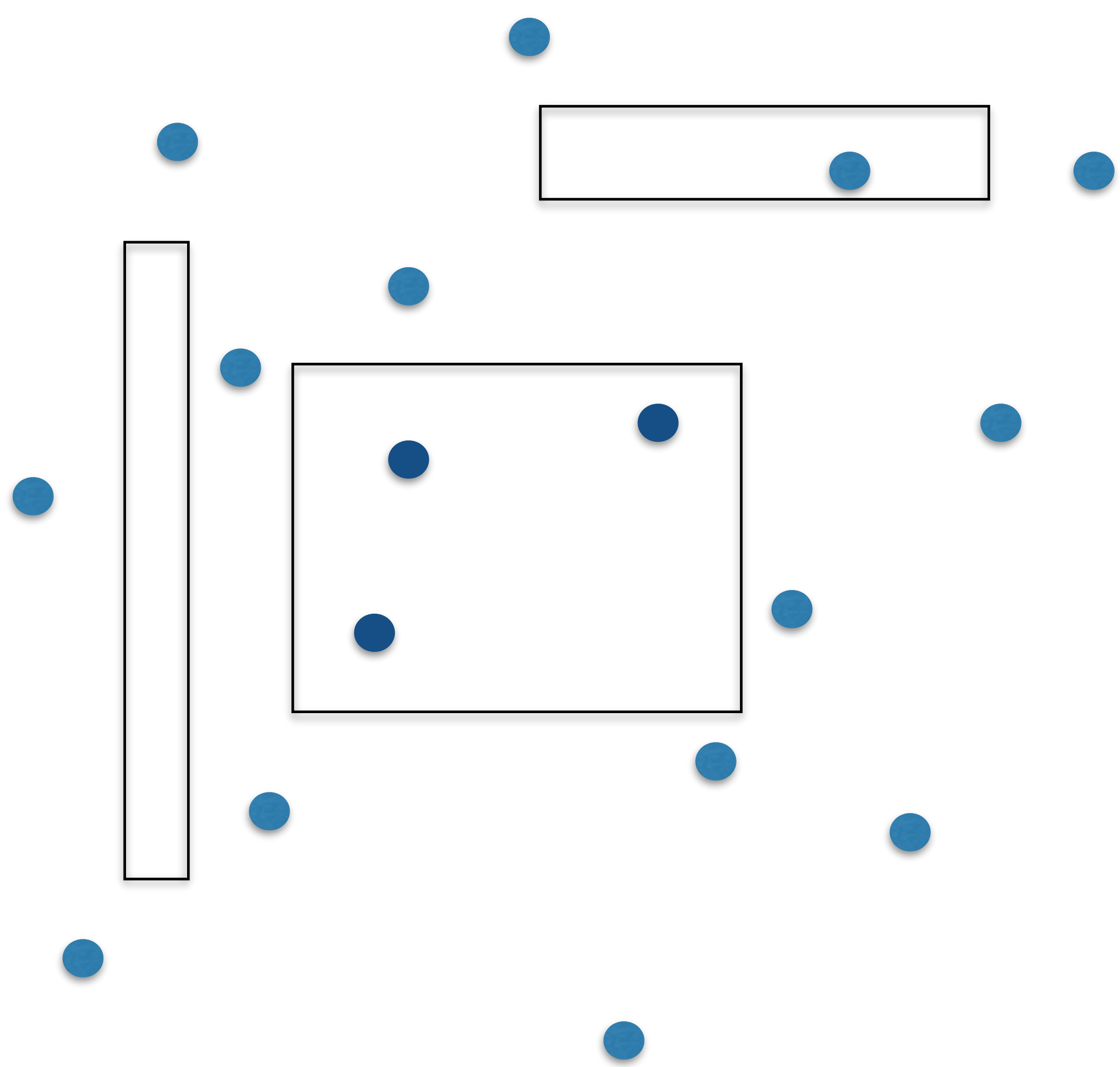
Techniques

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

Today

Range searching (2D)

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

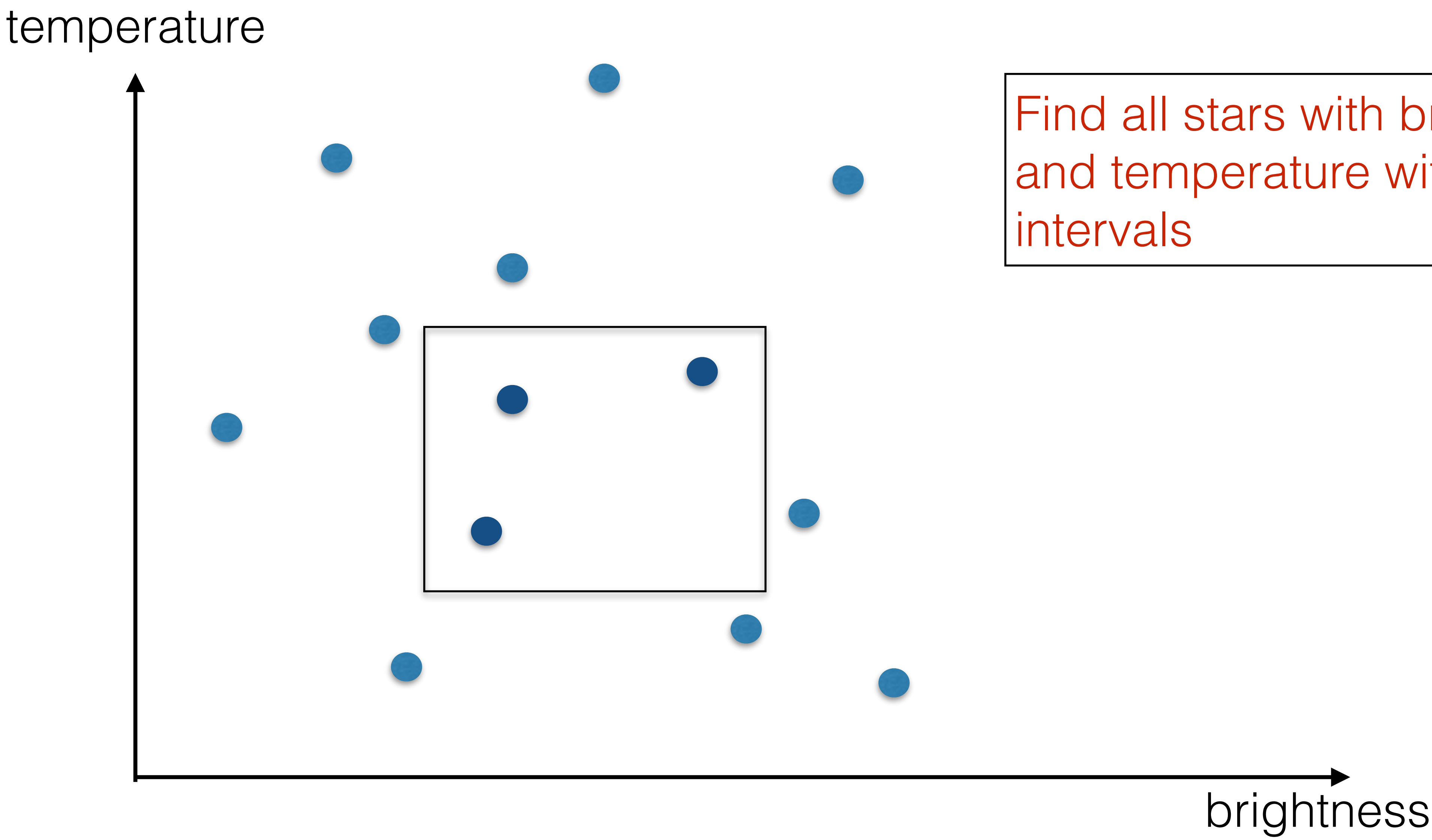


2D

Examples

Arise in settings that are not geometrical.

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

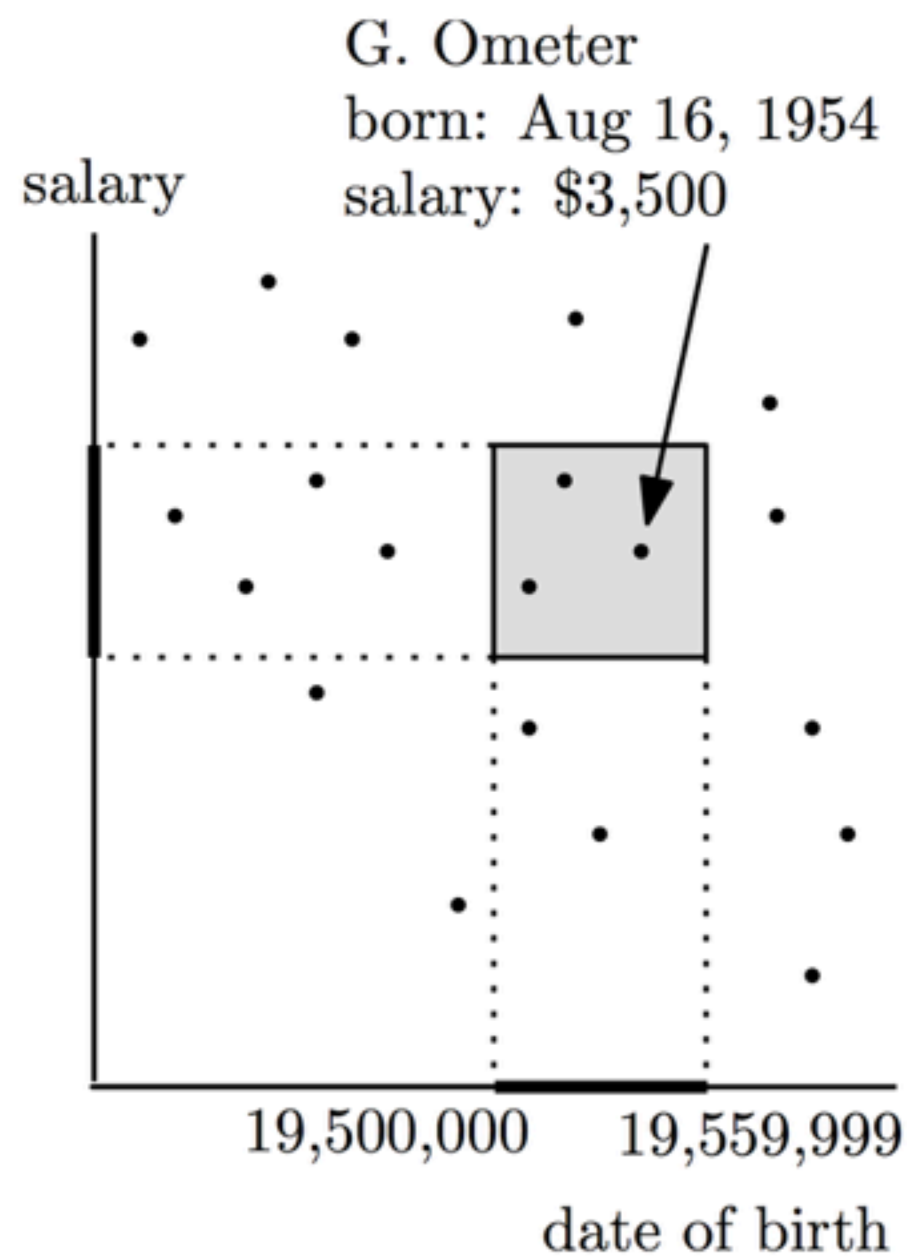


Find all stars with brightness and temperature within given intervals

Examples

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

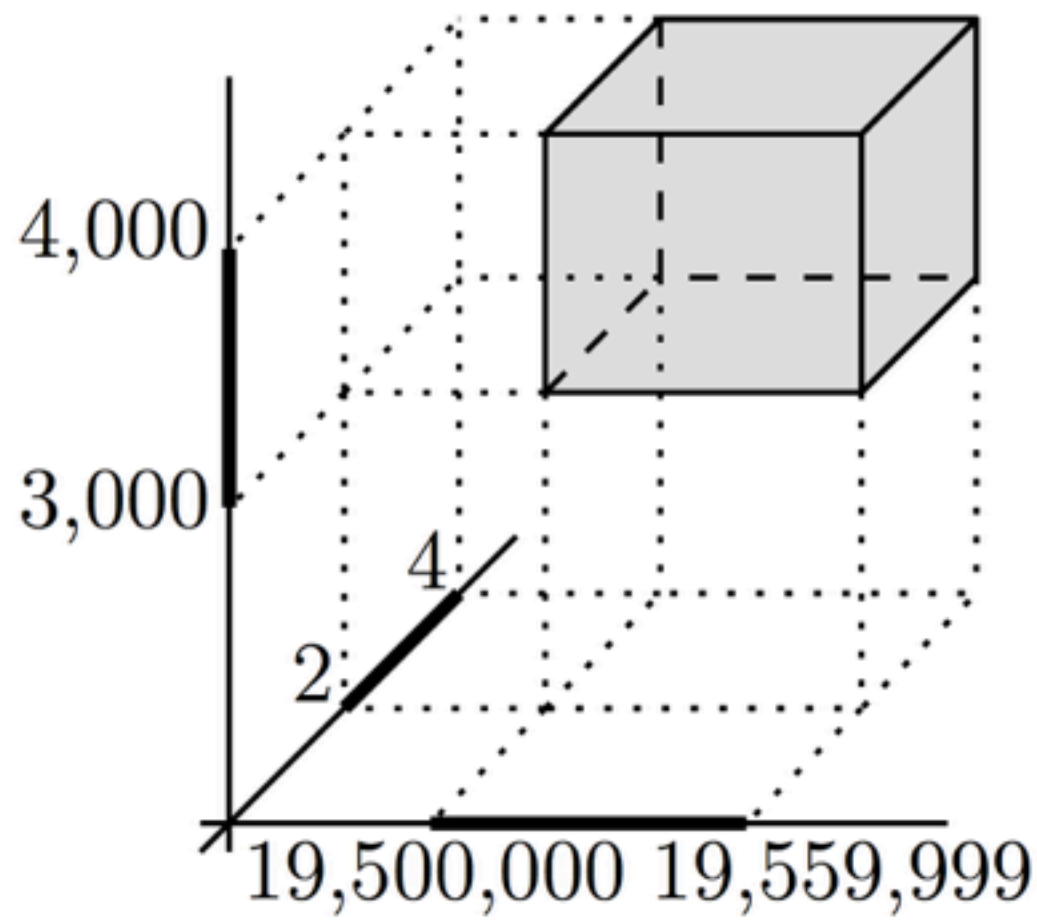
A database query may ask for all employees with age between a_1 and a_2, and salary between s_1 and s_2



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

Examples

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



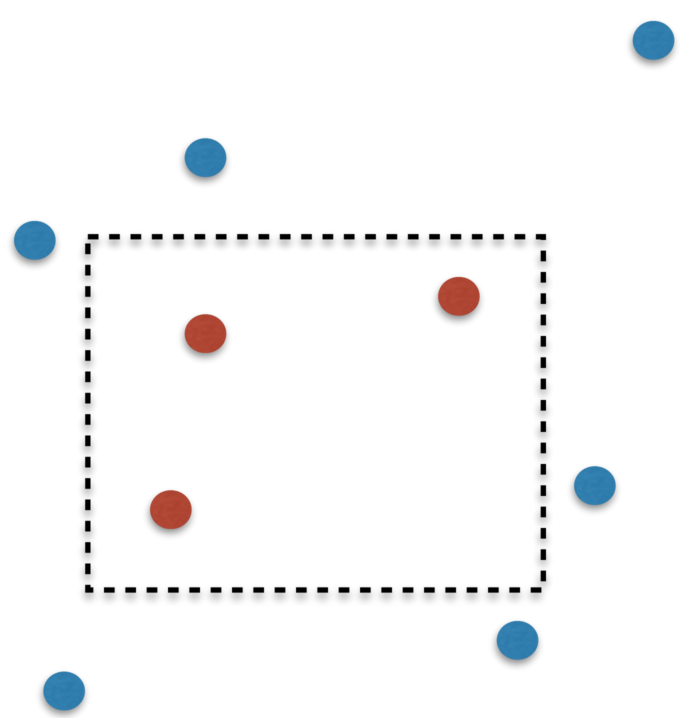
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

Range searching (2D)

The naive approach:

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

Do better. What sort of bounds can we expect?

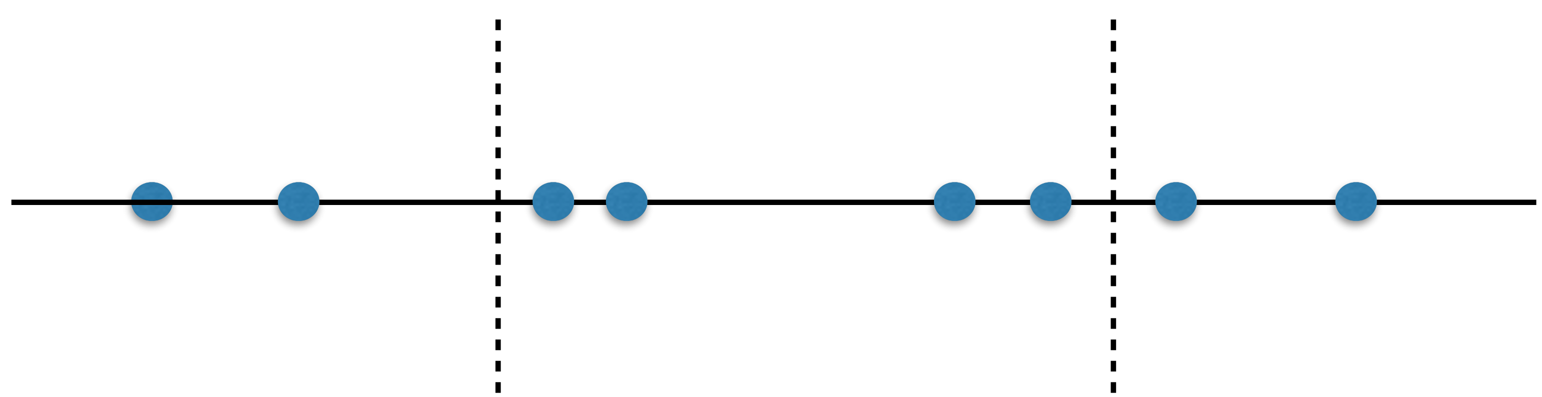


Points are static or dynamic?

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

1D Range searching

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

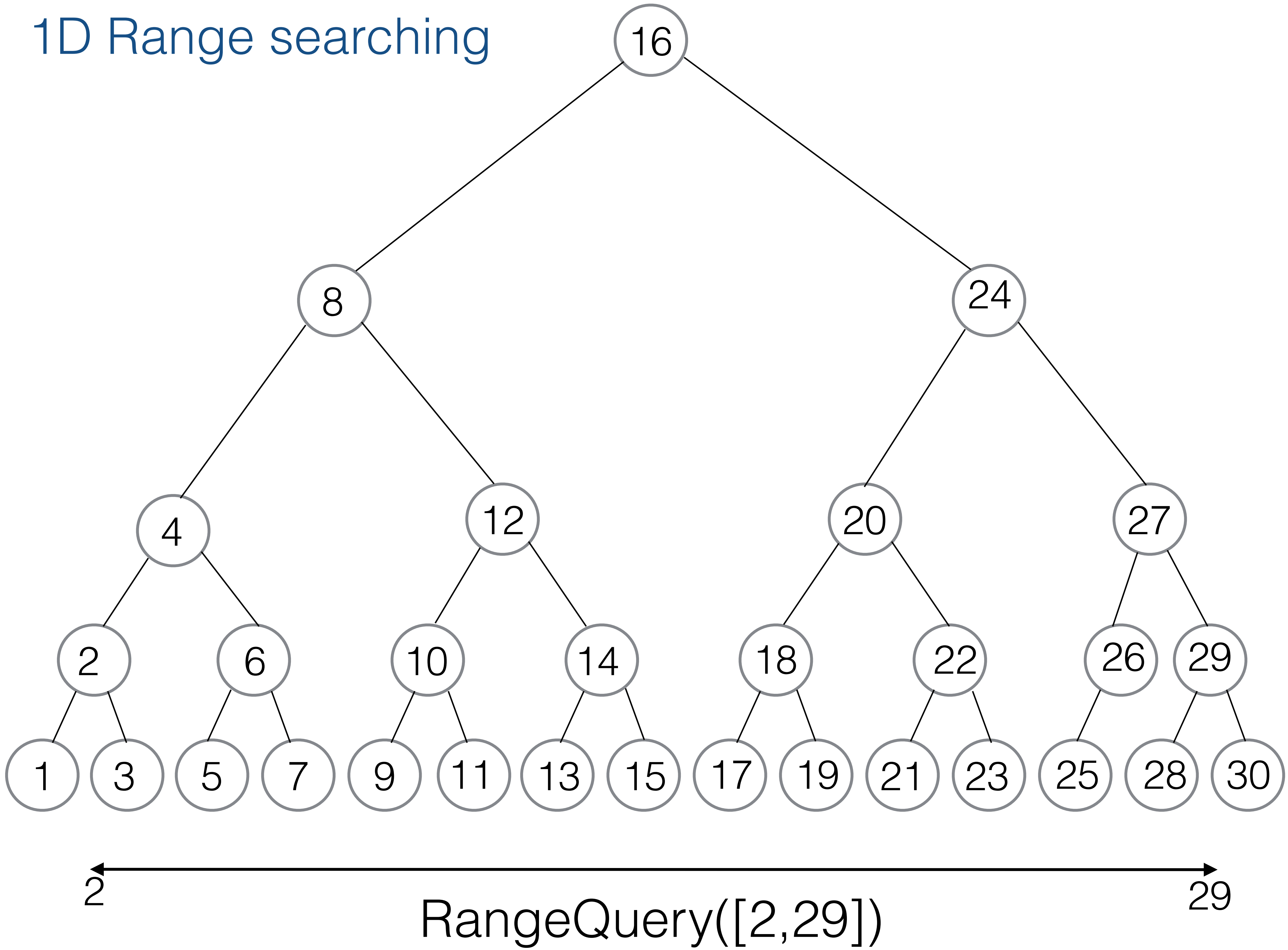


1D

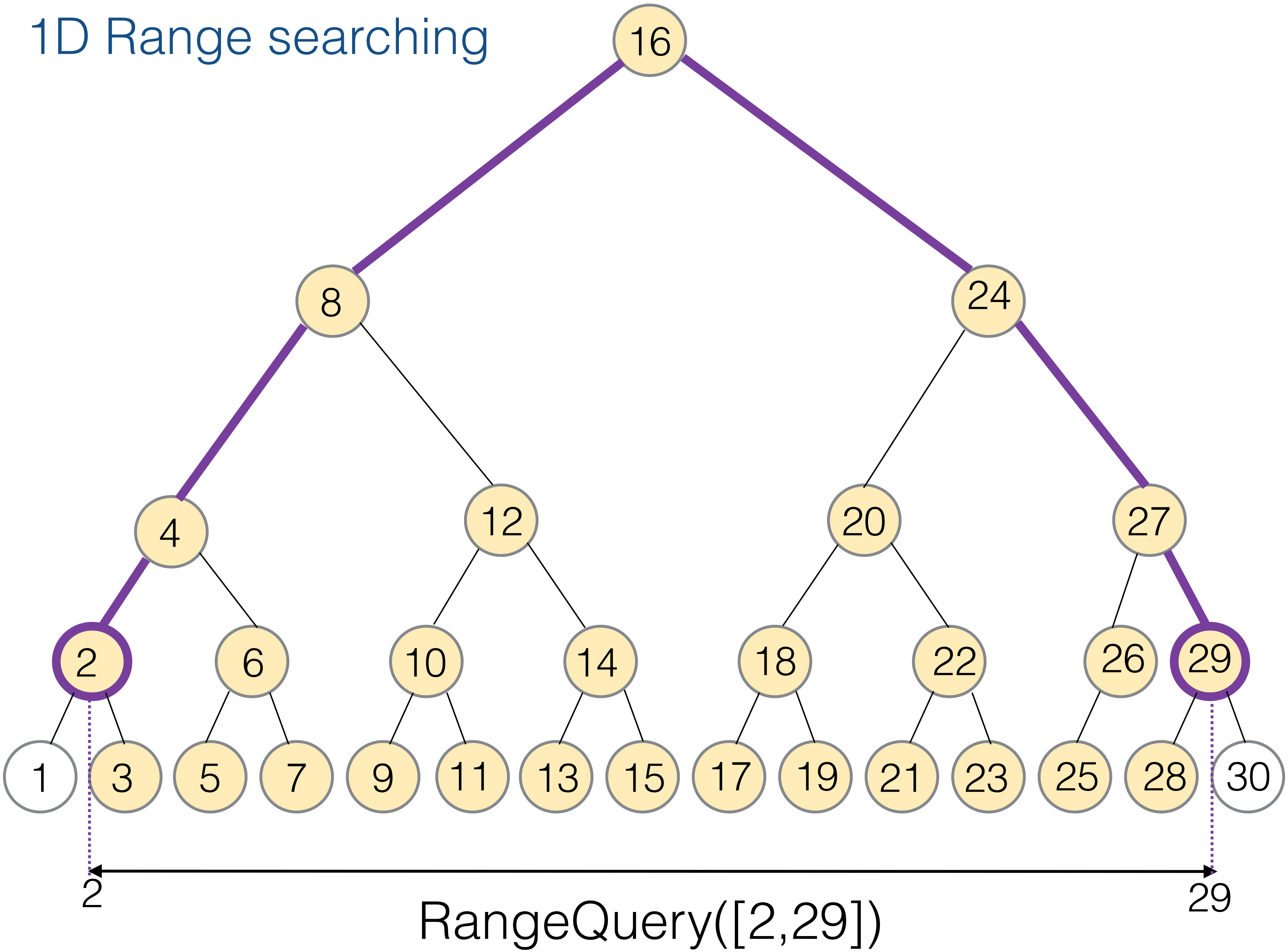
Example

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

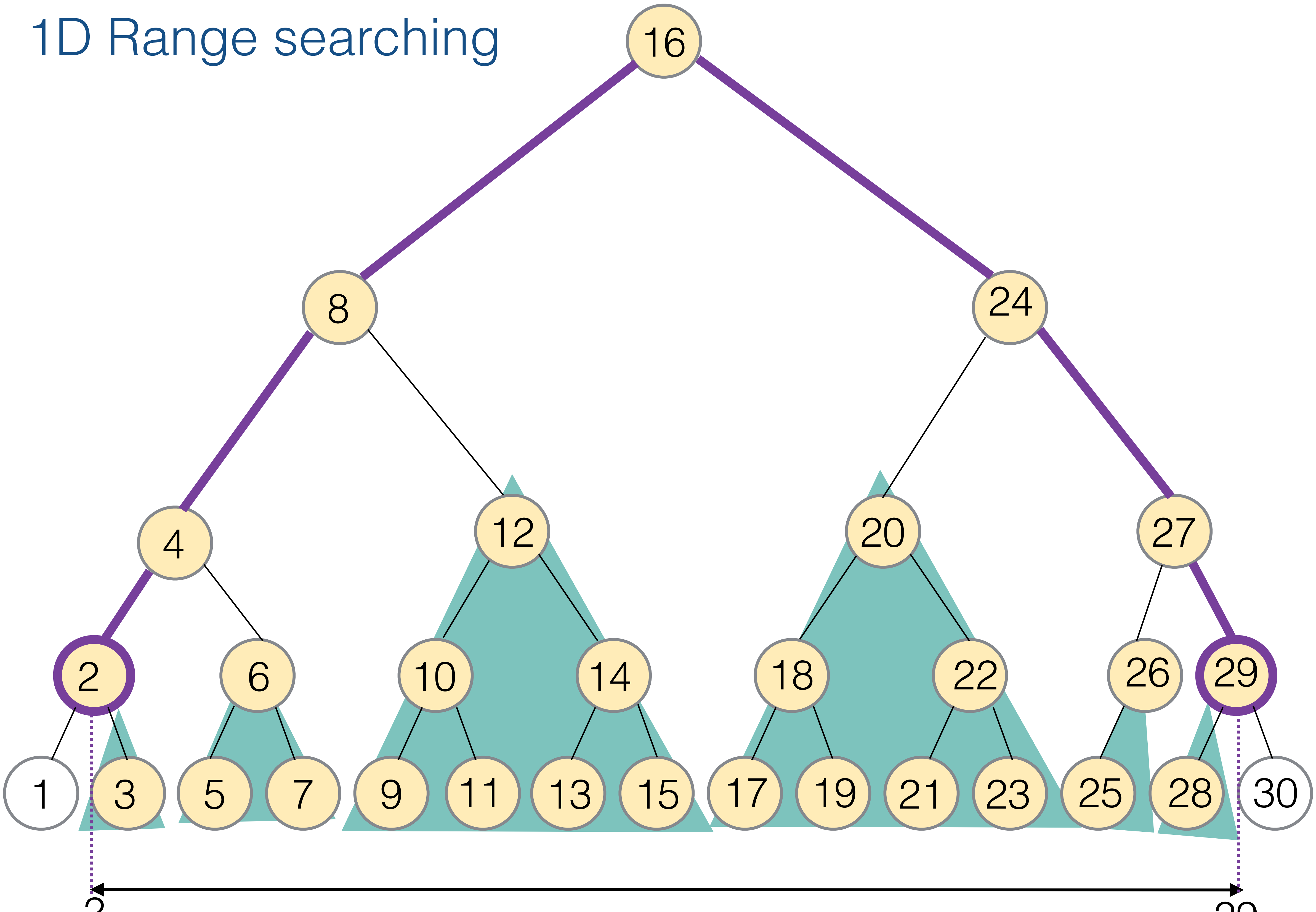
1D Range searching



1D Range searching



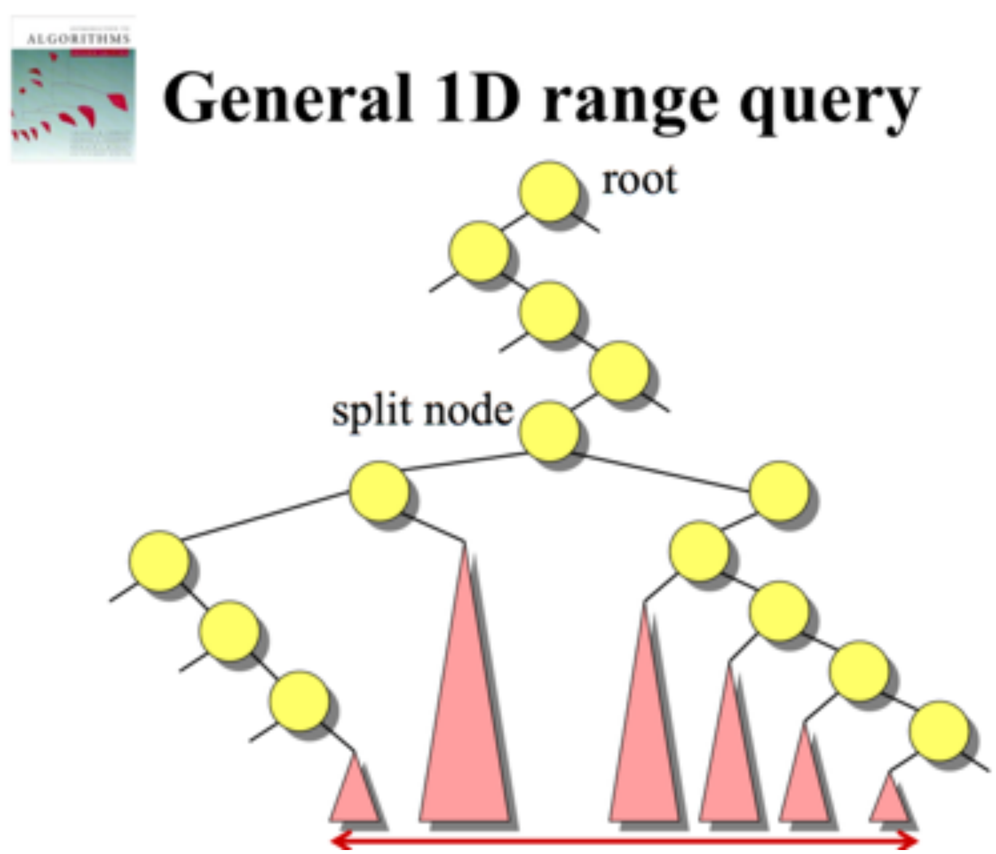
1D Range searching



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

1D

- What's known: A set of n points can be pre-processed into a BBST such that:
  - Build: O(n lg n)
  - Space: O(n)
  - Range queries: O(lg n +k)
  - dynamic structure: points can be inserted/deleted in O(lg n)



1D

- What's known: A set of n points can be pre-processed into a BBST such that:
  - Build: O(n lg n)
  - Space: O(n)
  - Range queries: O(lg n +k)
  - dynamic structure: 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 n^2 +k)

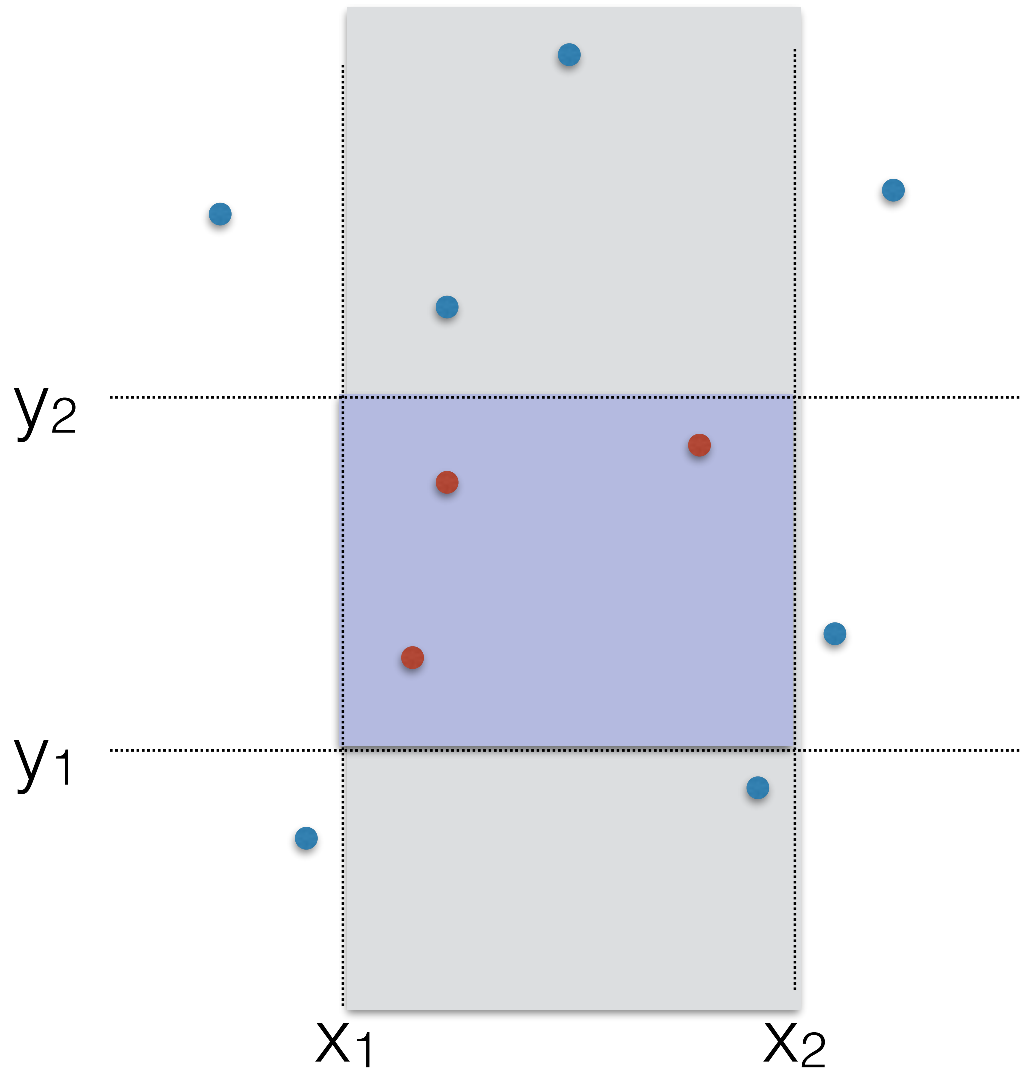
?perhaps?

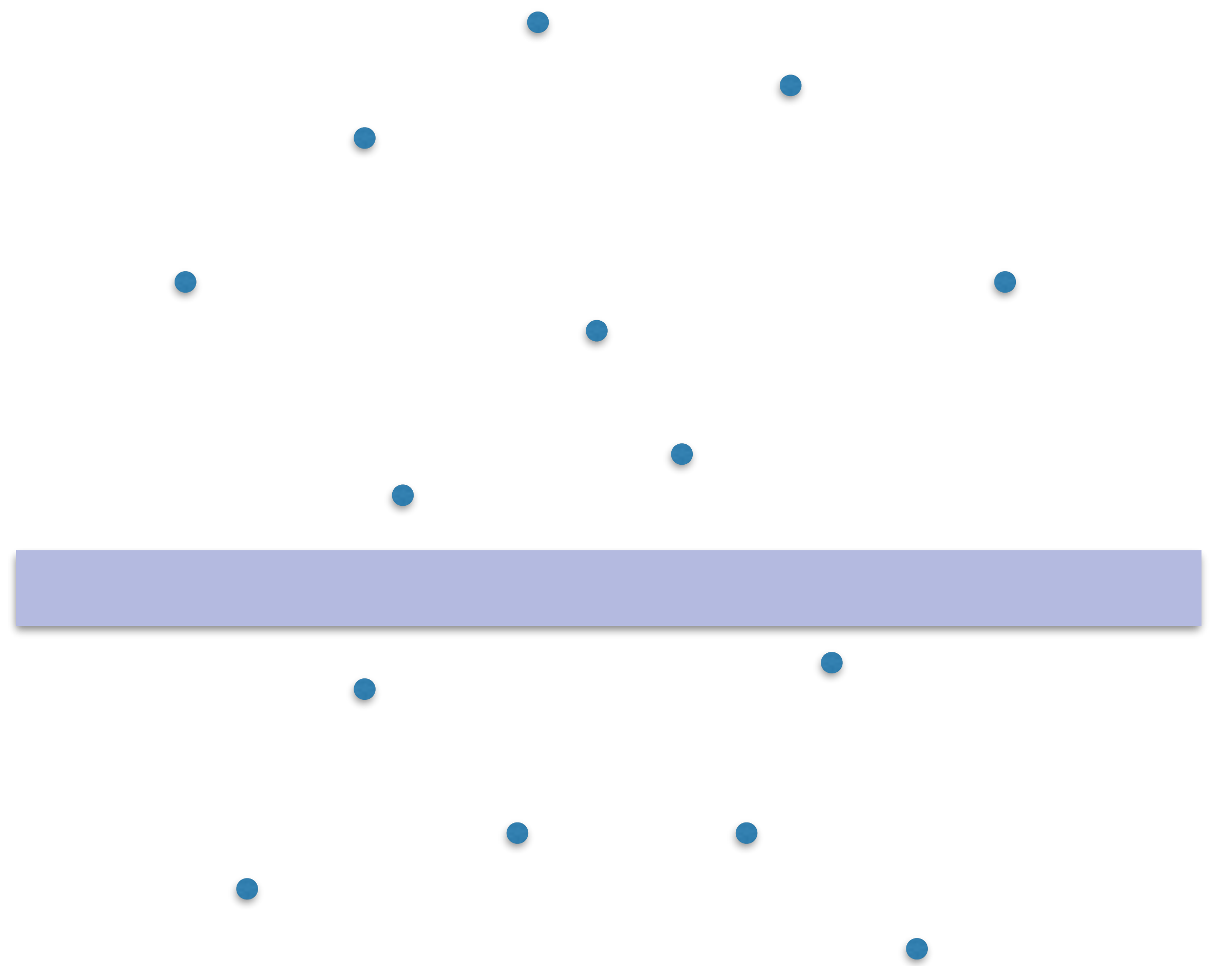
Denote query $[x_1, x_2] \times [y_1, y_2]$

Could it be as simple as ..?

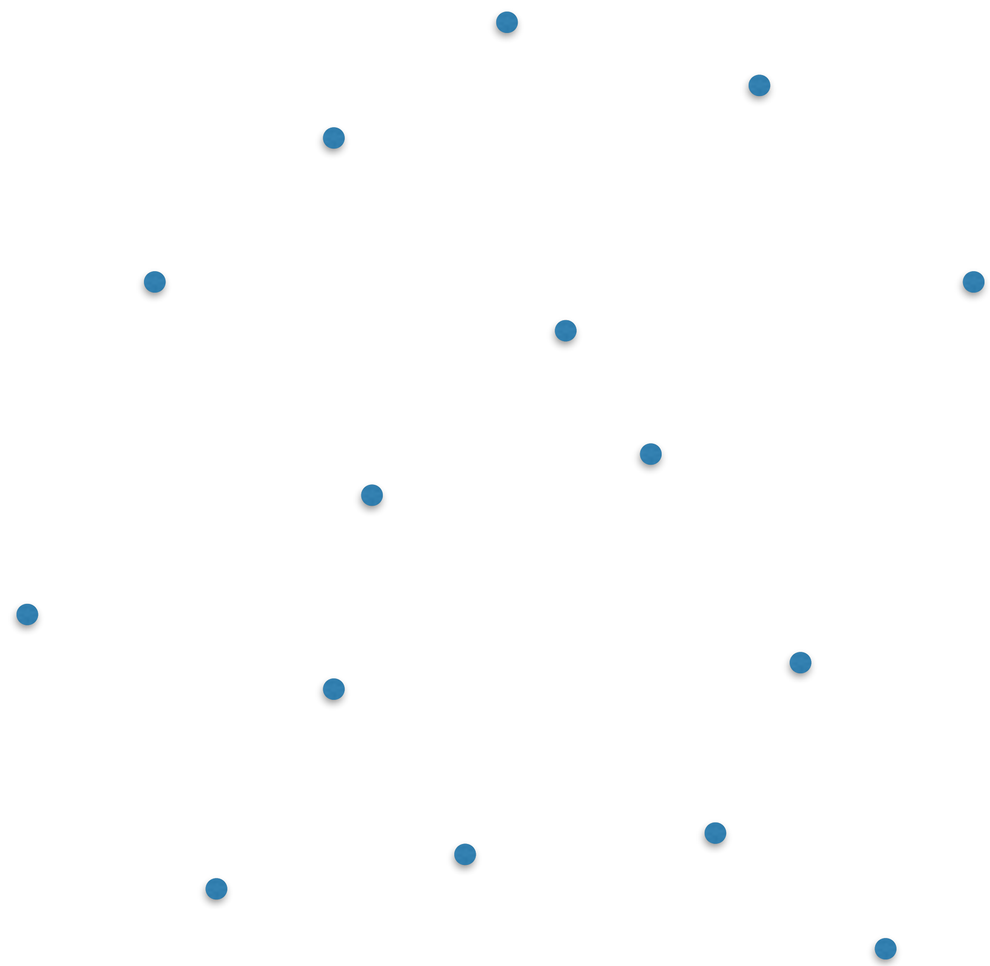
- Idea
  - Find all points with the x-coordinates in the correct range $[x_1, x_2]$

    Store points in a BBST on x-coord
  - Out of these points, find all points with the y-coord in the correct range $[y_1, y_2]$
- Time?

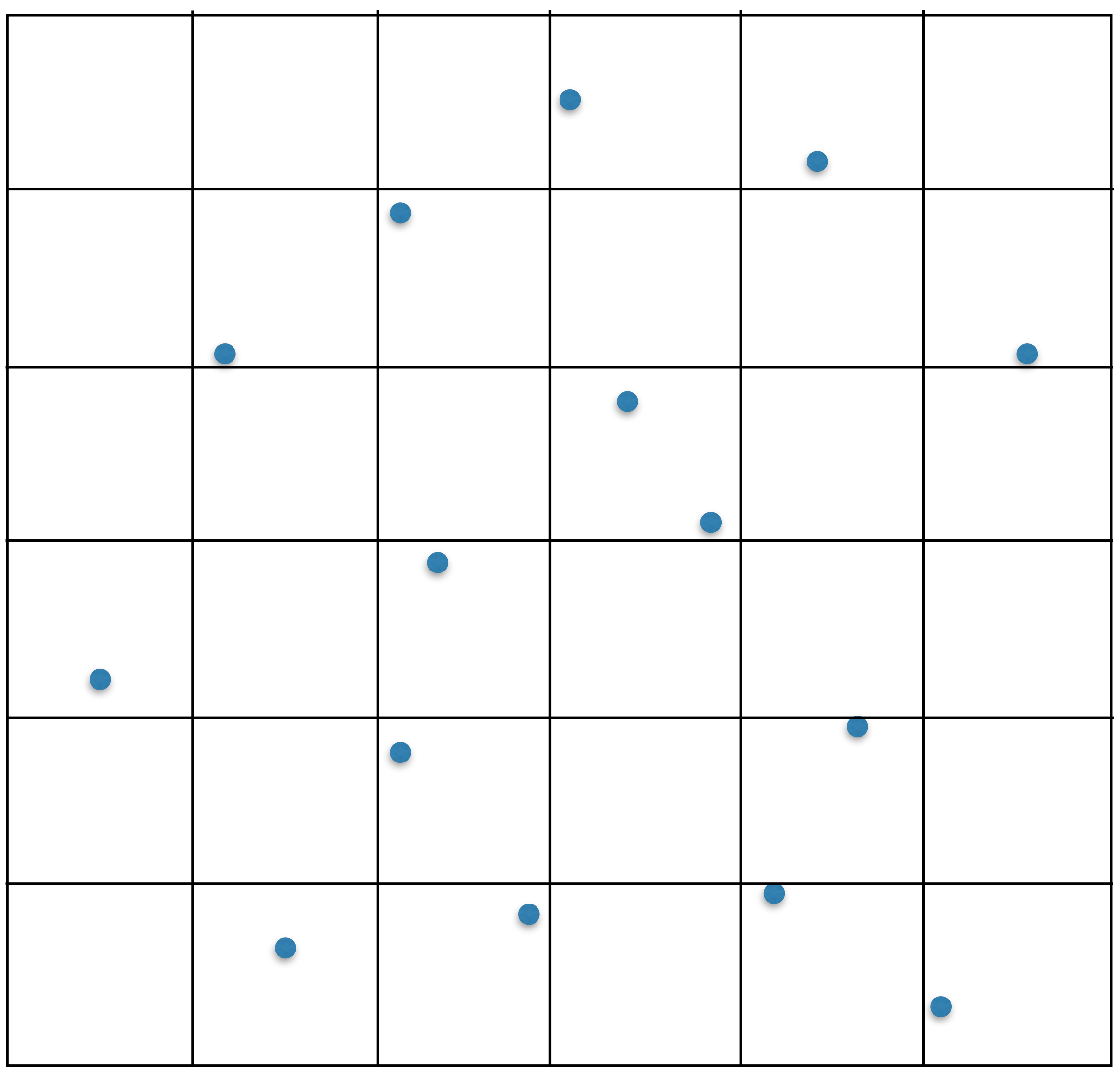




Space decomposition



The grid method



The grid method

```
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

- To build a grid of m-by-m cells from a set of points P
  - figure out a rectangle that contains P: for e.g. x_{min}, x_{max}, y_{min}, y_{max}
  - allocate g as a 2d array of lists, all initially empty

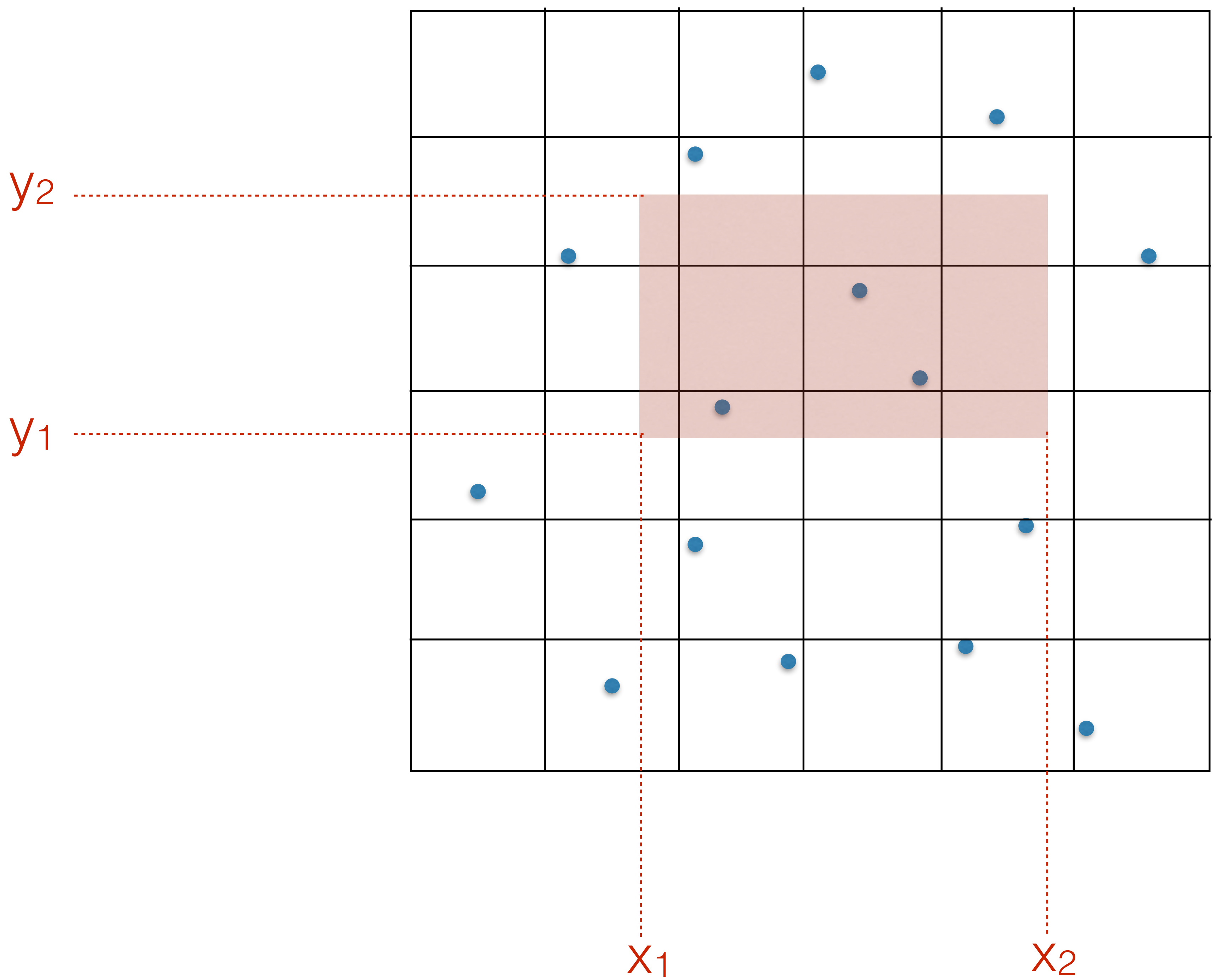
```
g = new (List<point2D*>**)[m];
for (int i=0; i<m; i++) {
        g[i] = new (List<point2D*>*) [m];
        for (int j=0; j<m; j++) {
                g[i][j] = new List<point2D*>;
        }
}
```

  - for each point p in P: figure out which cell i, j contains p, and insert p in g[i][j]

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

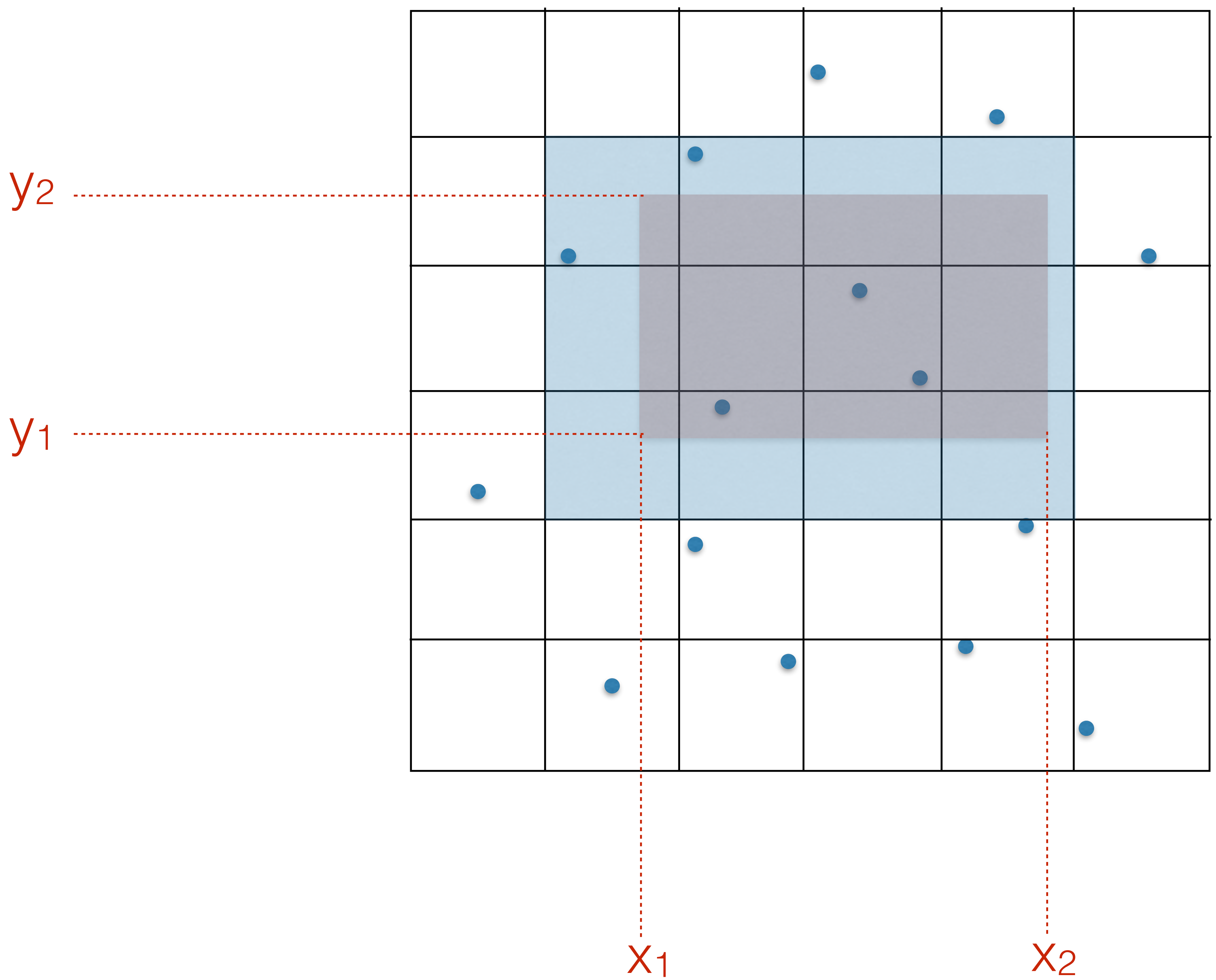
The grid method

- Range queries



The grid method

- Range queries



The grid method

Analysis

- How many points in a cell?
    - worst case
    - best case
- How long does a range query take ?
    - worst-case
    - points are uniformly distributed
- How to chose m?

The grid method

Analysis

- How many points in a cell?
    - worst case
    - best case
- How long does a range query take ?
    - worst-case?
    - points are uniformly distributed?
- How to chose m?

Grids perform well if points are uniformly distributed, and less well if they are not.

Grids can be used as heuristic for many other problems besides range searching (e.g. closest pair, neighbor queries)

2d search trees

3d search trees

4d search trees

…

kd trees

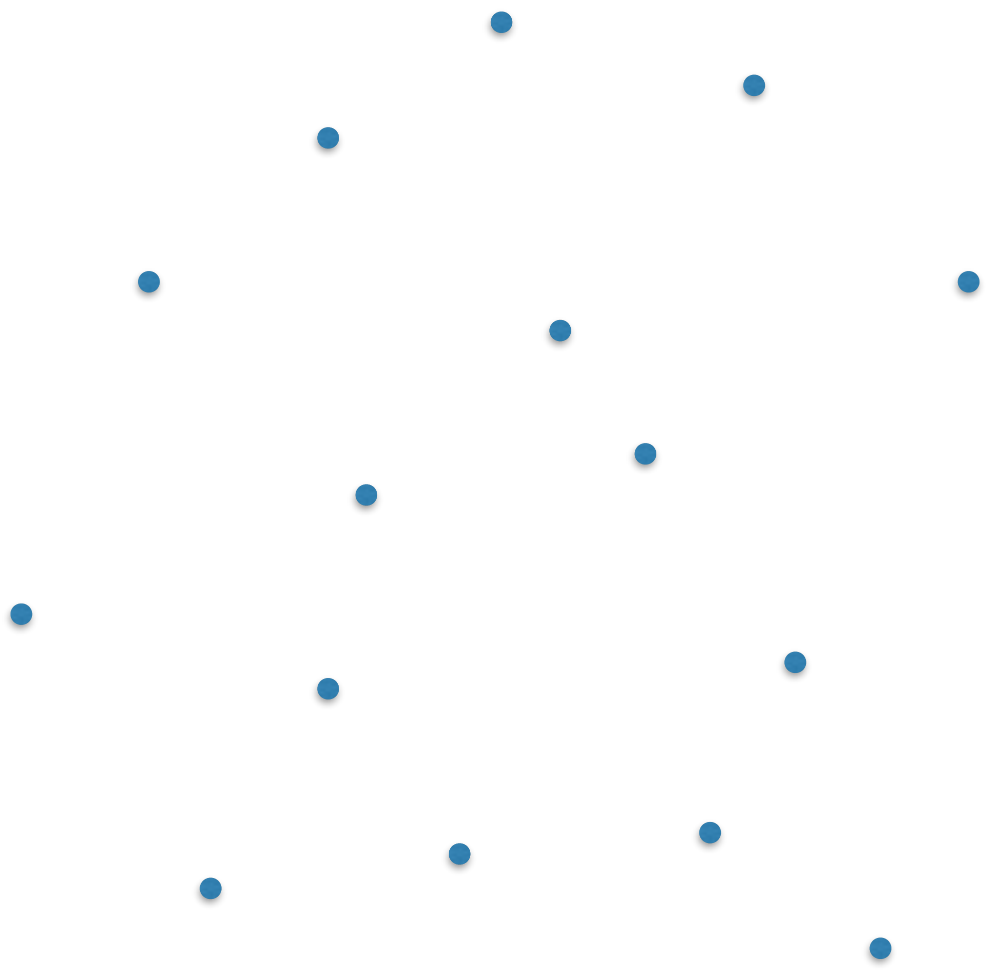
2d binary search trees

The idea: A binary tree which recursively subdivides the plane by vertical and horizontal cut lines

Vertical and horizontal lines alternate

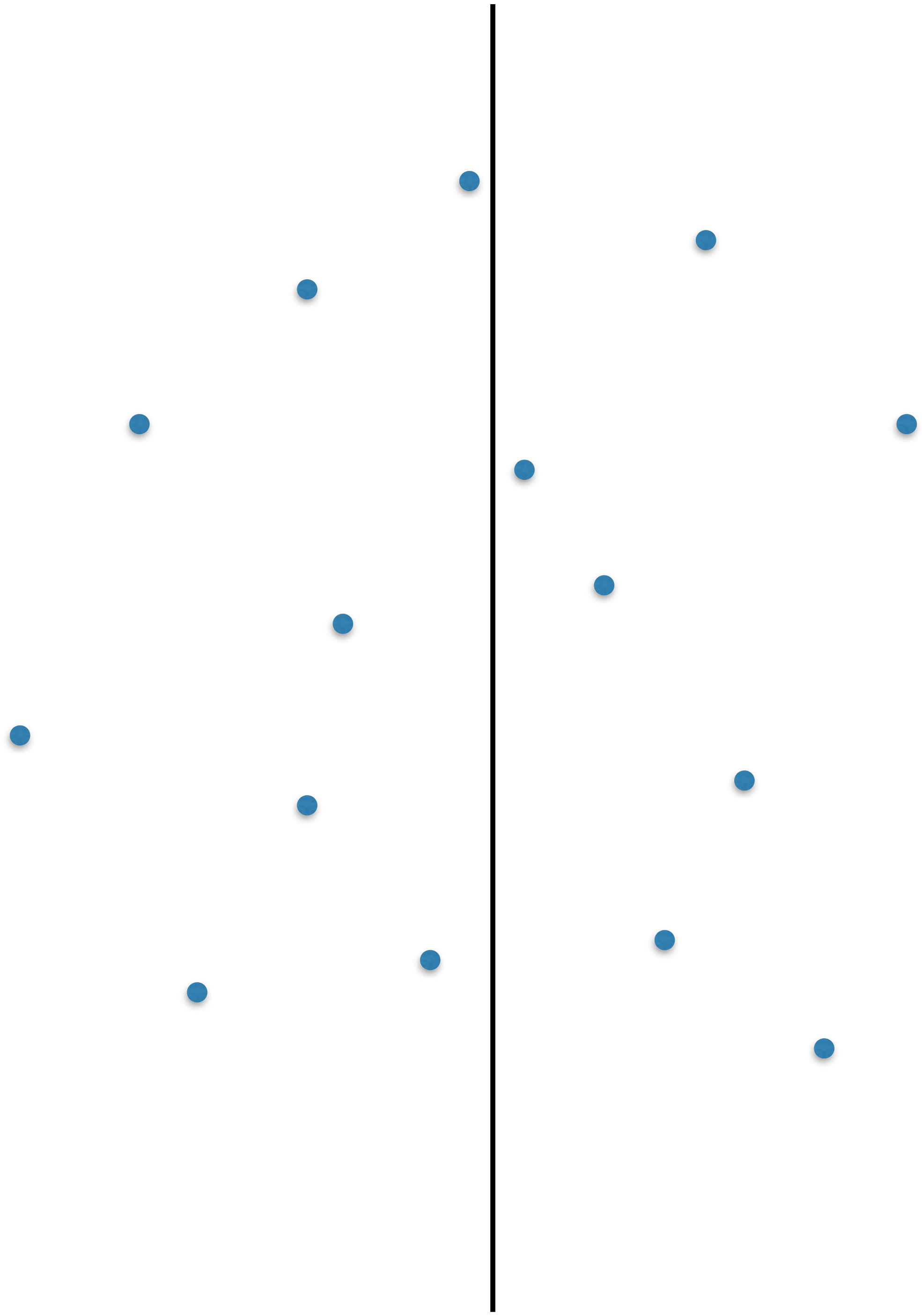
Cut lines are chosen to split the points in two (==> logarithmic height)

2d binary search trees



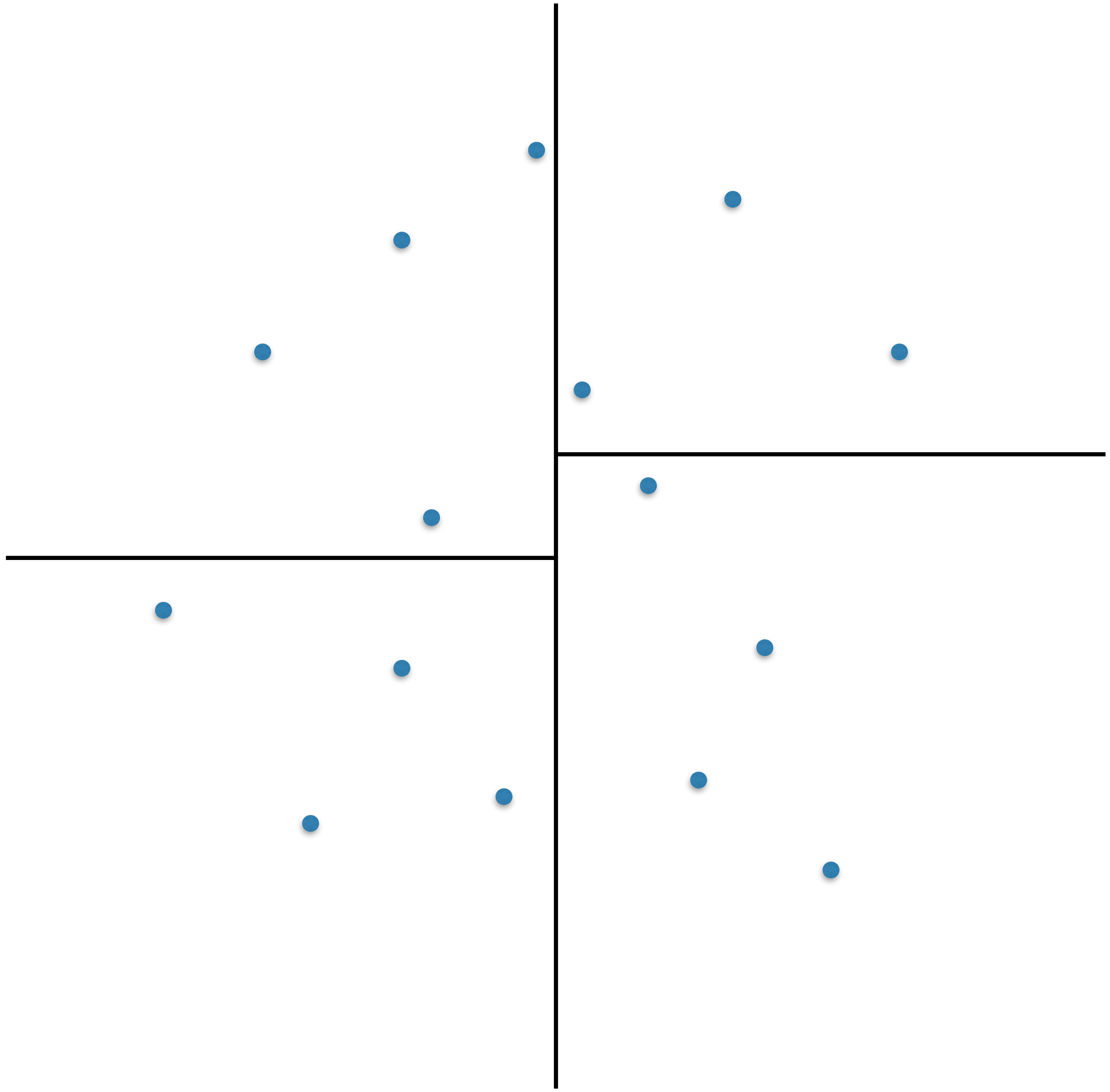
2d binary search trees

split points in two halves with a vertical line

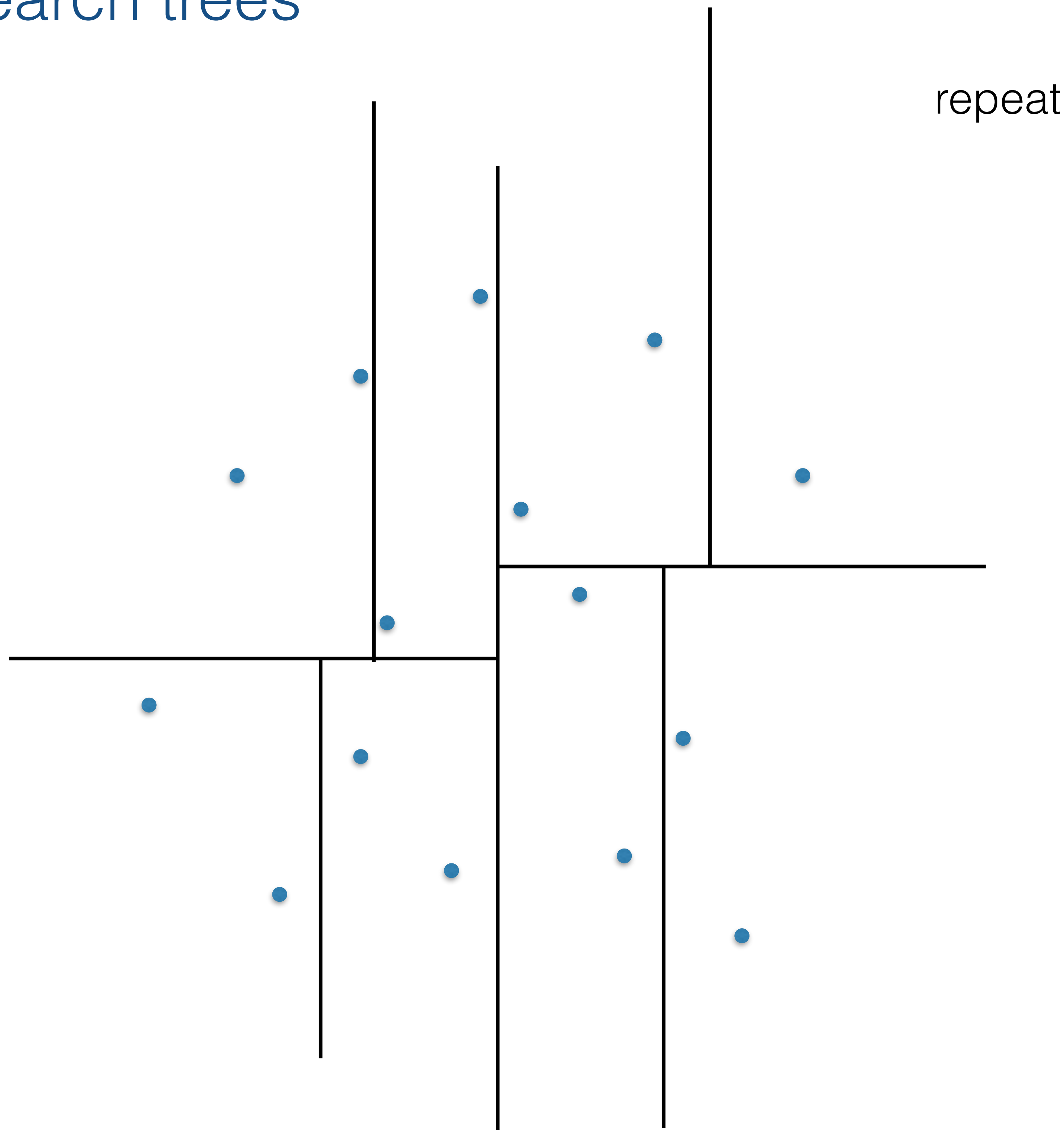


2d binary search trees

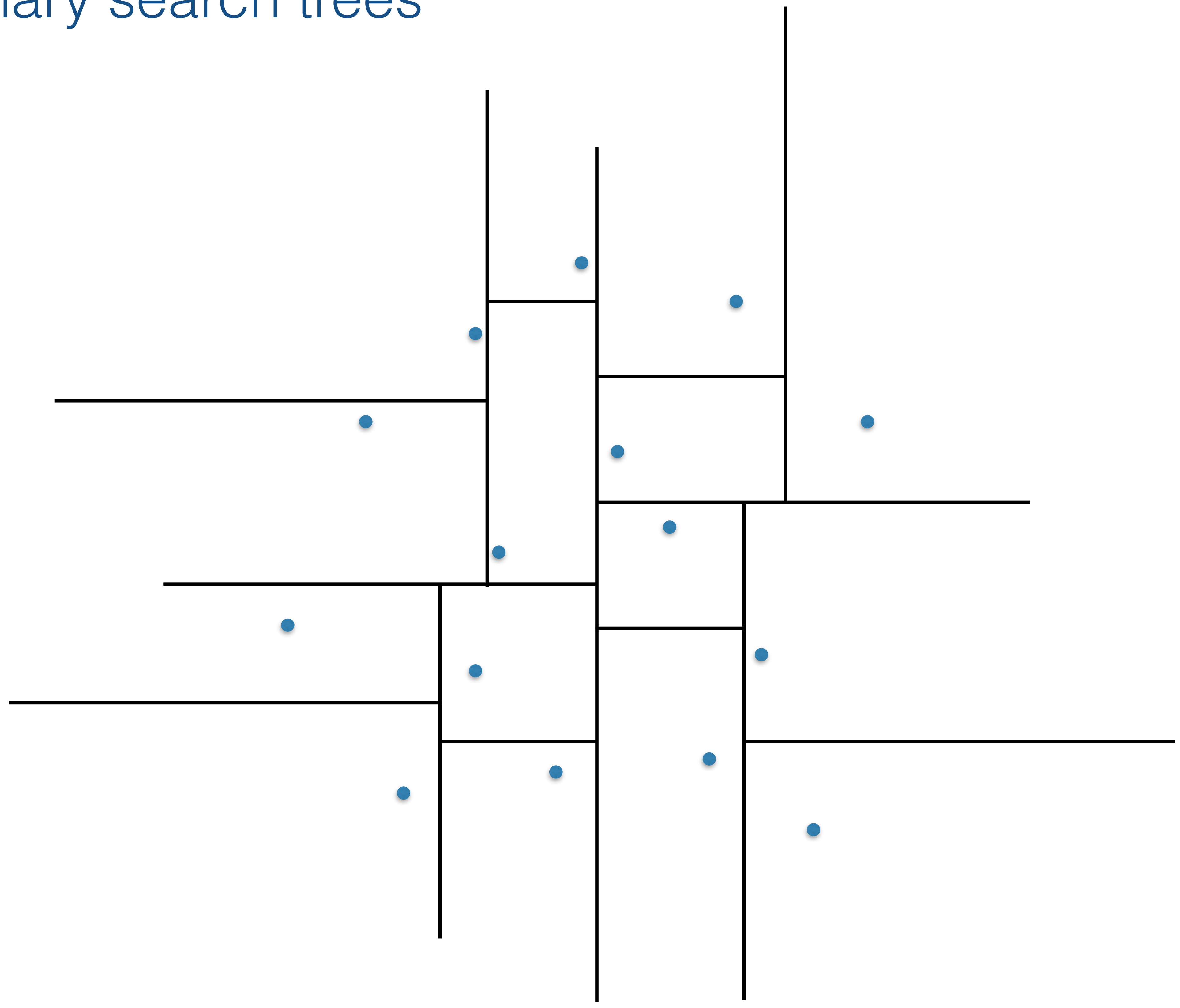
split each side into half with a horizontal line



2d binary search trees



2d binary search trees



2d binary search trees

Variants:

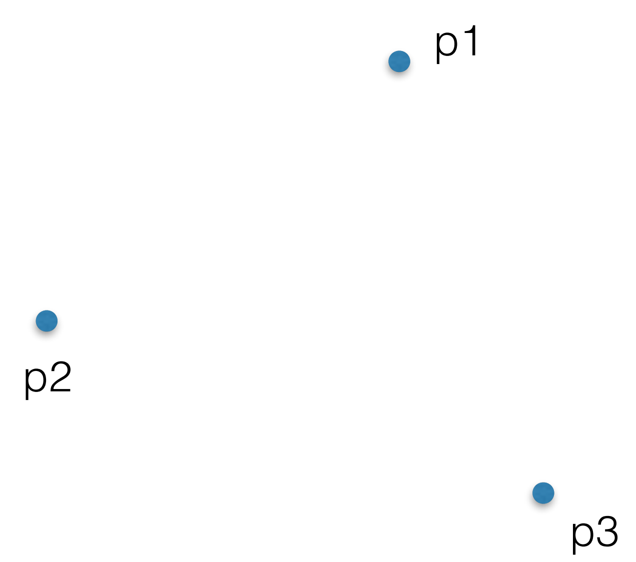
- Choose the cut line so that it falls in between the points. Internal nodes store lines, and points are only in leaves.

- Choose the cut line so it goes through the median point. Assign the median to the e.g. smaller (left/below) side, consistently. 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.

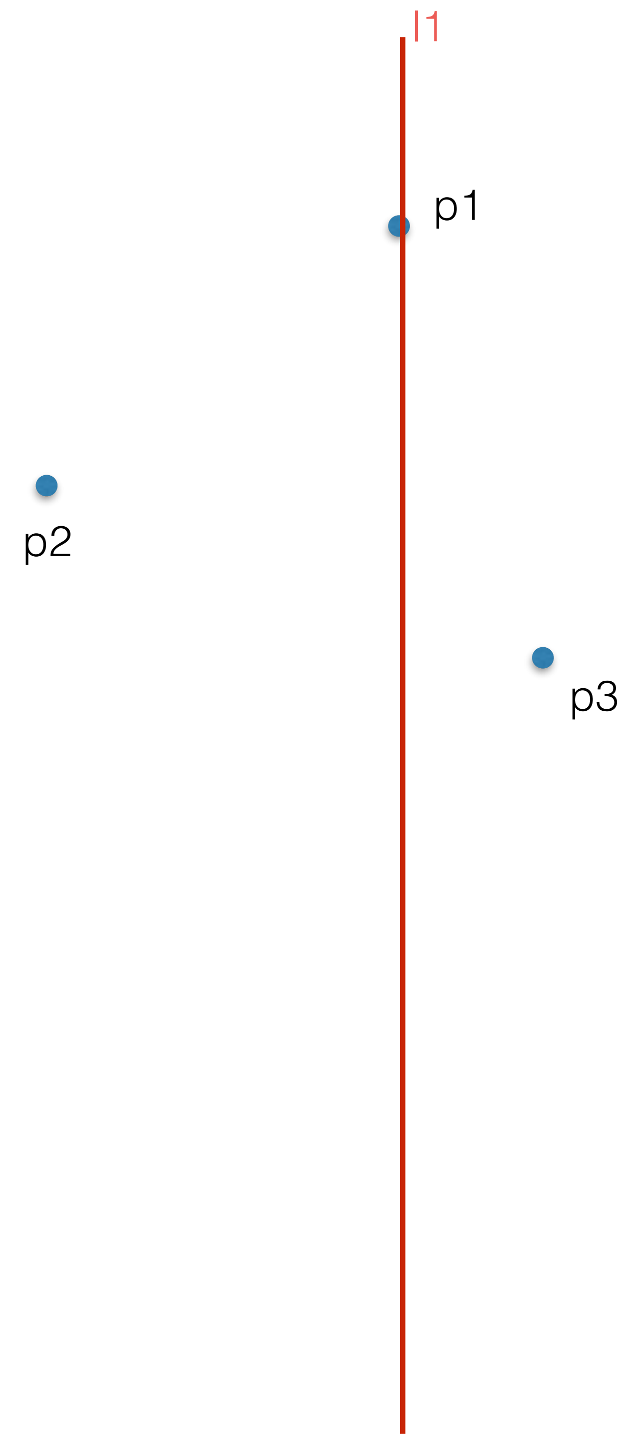
- …

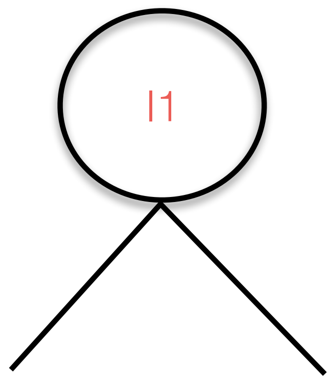
2d binary search trees



2d binary search trees

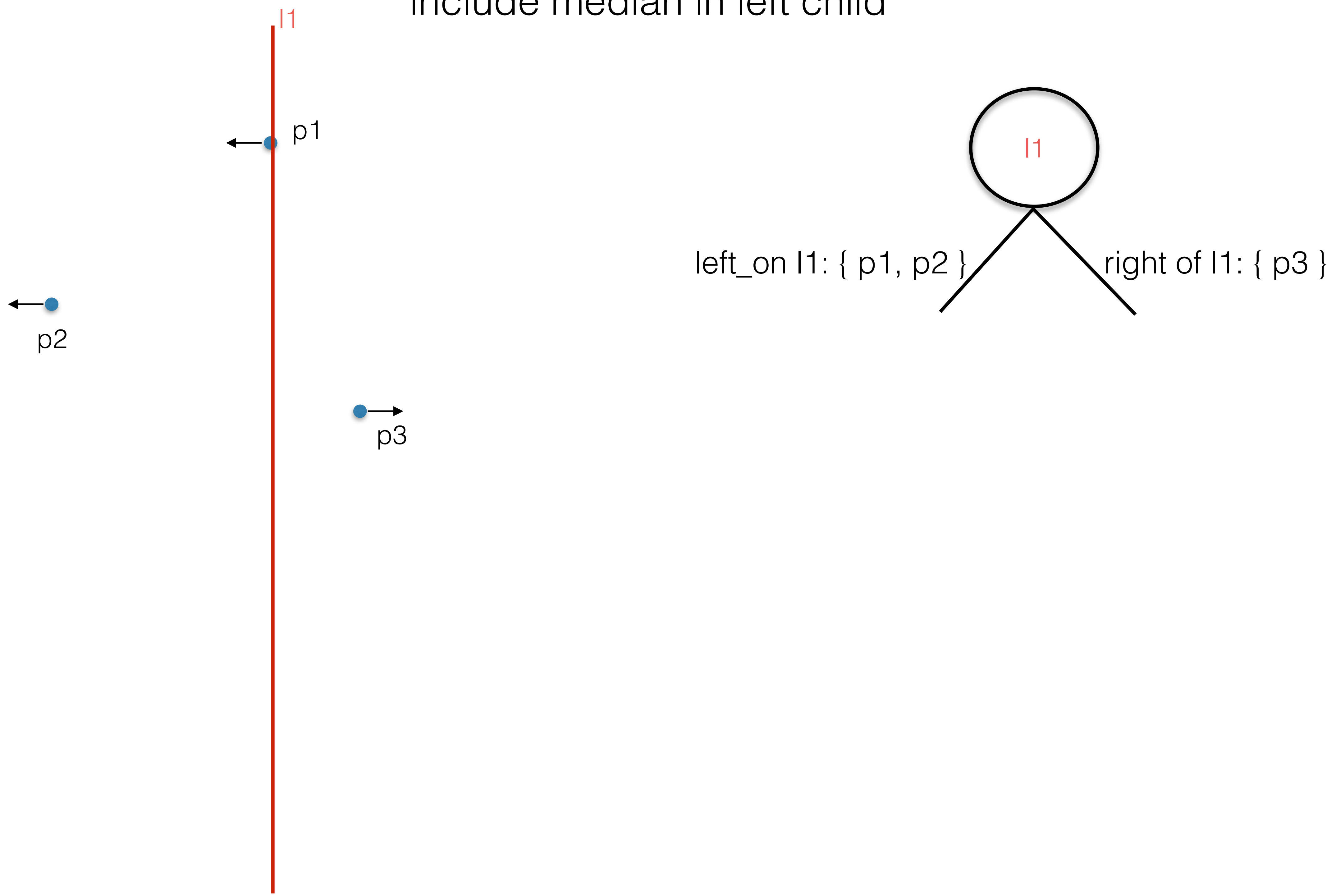
split with vertical line through x-median
include median in left child





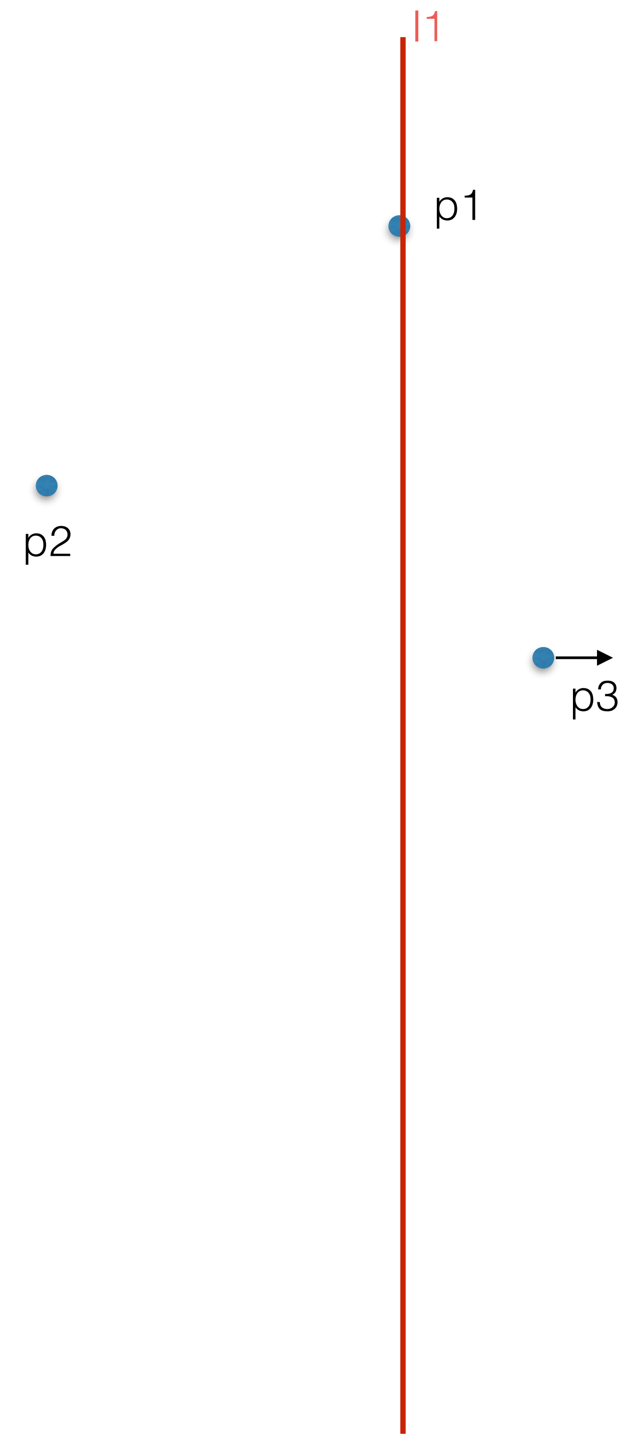
2d binary search trees

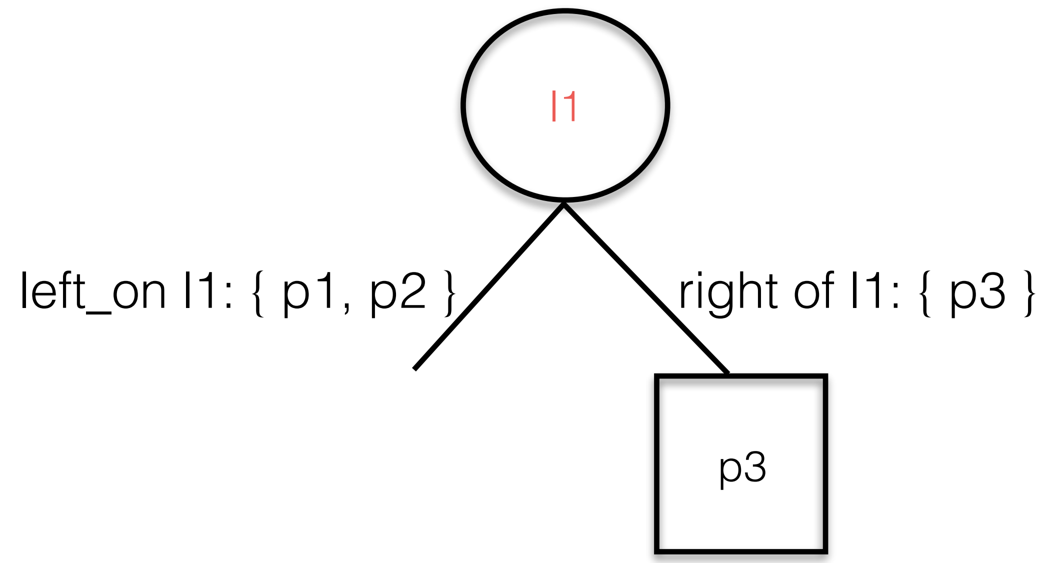
split with vertical line through x-median
include median in left child



2d binary search trees

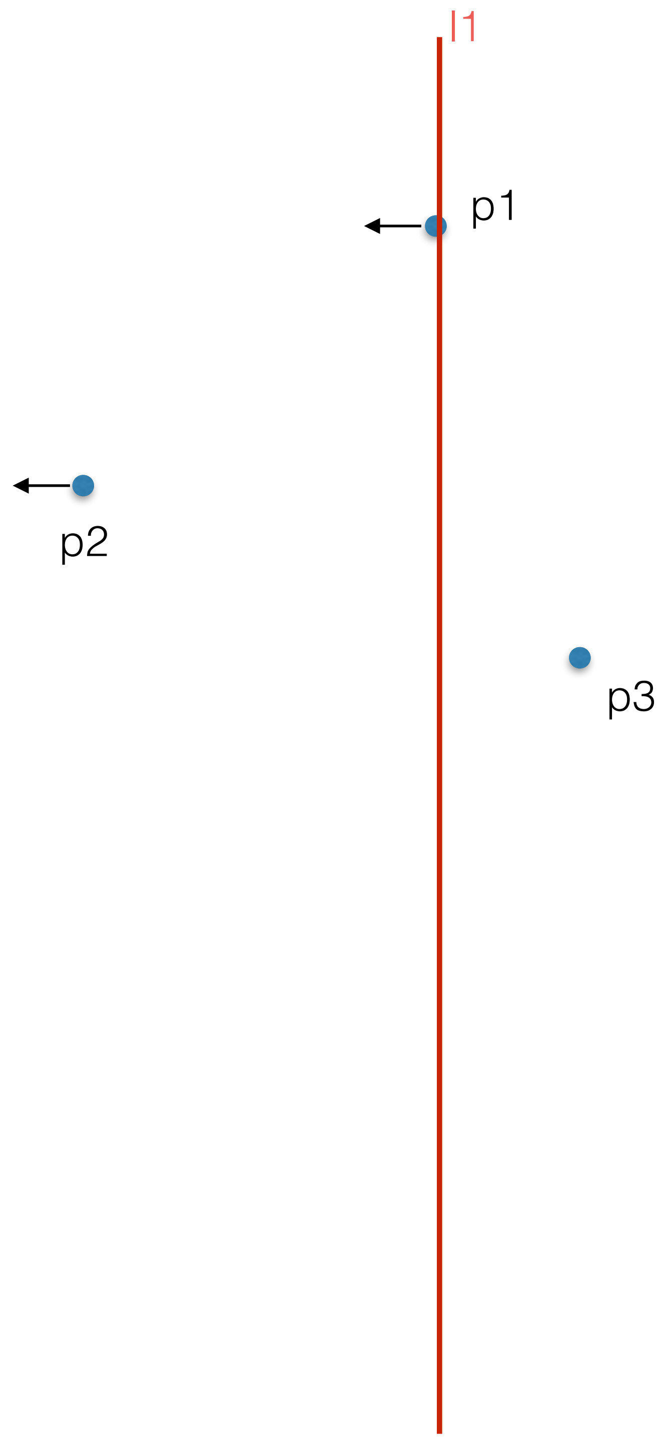
right of l1: p3 => leaf

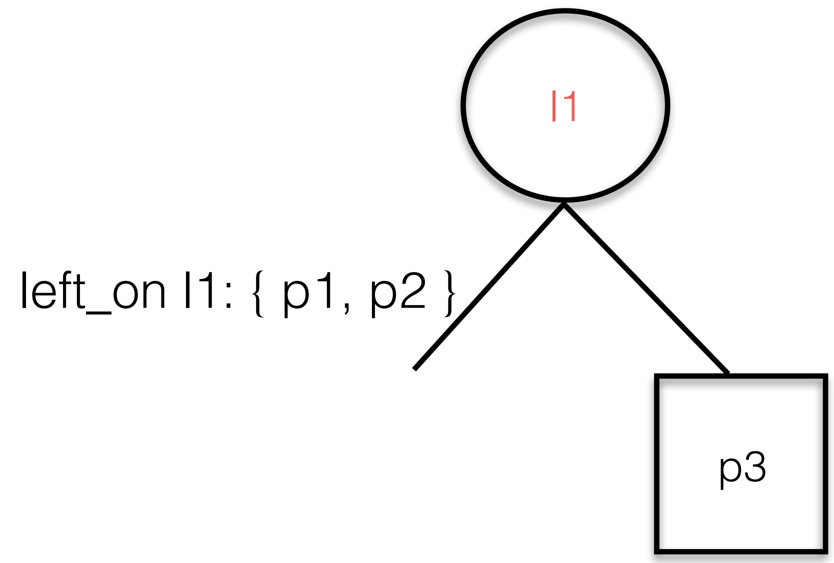




2d binary search trees

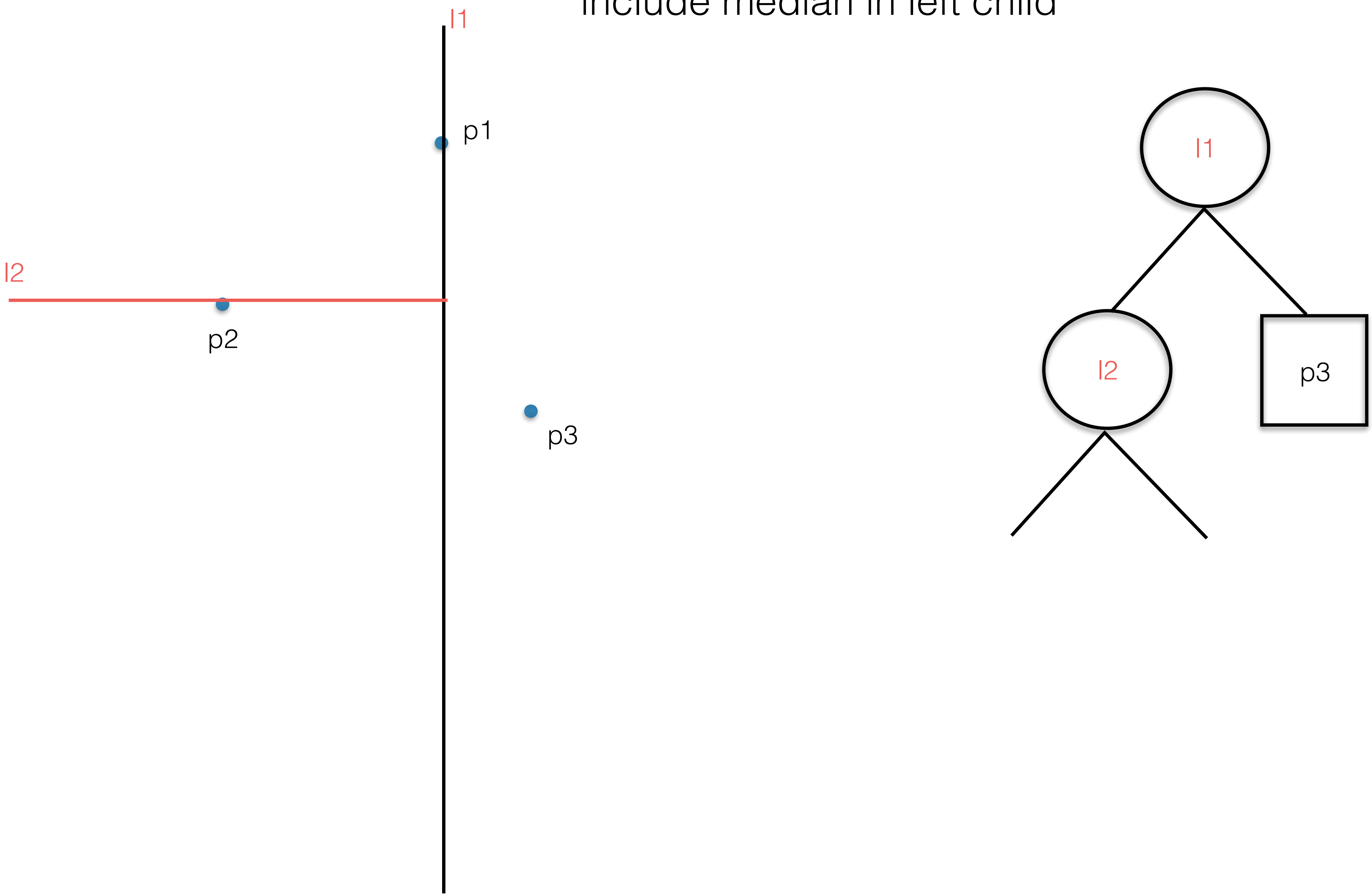
left_on l1: p1,p2 => recurse





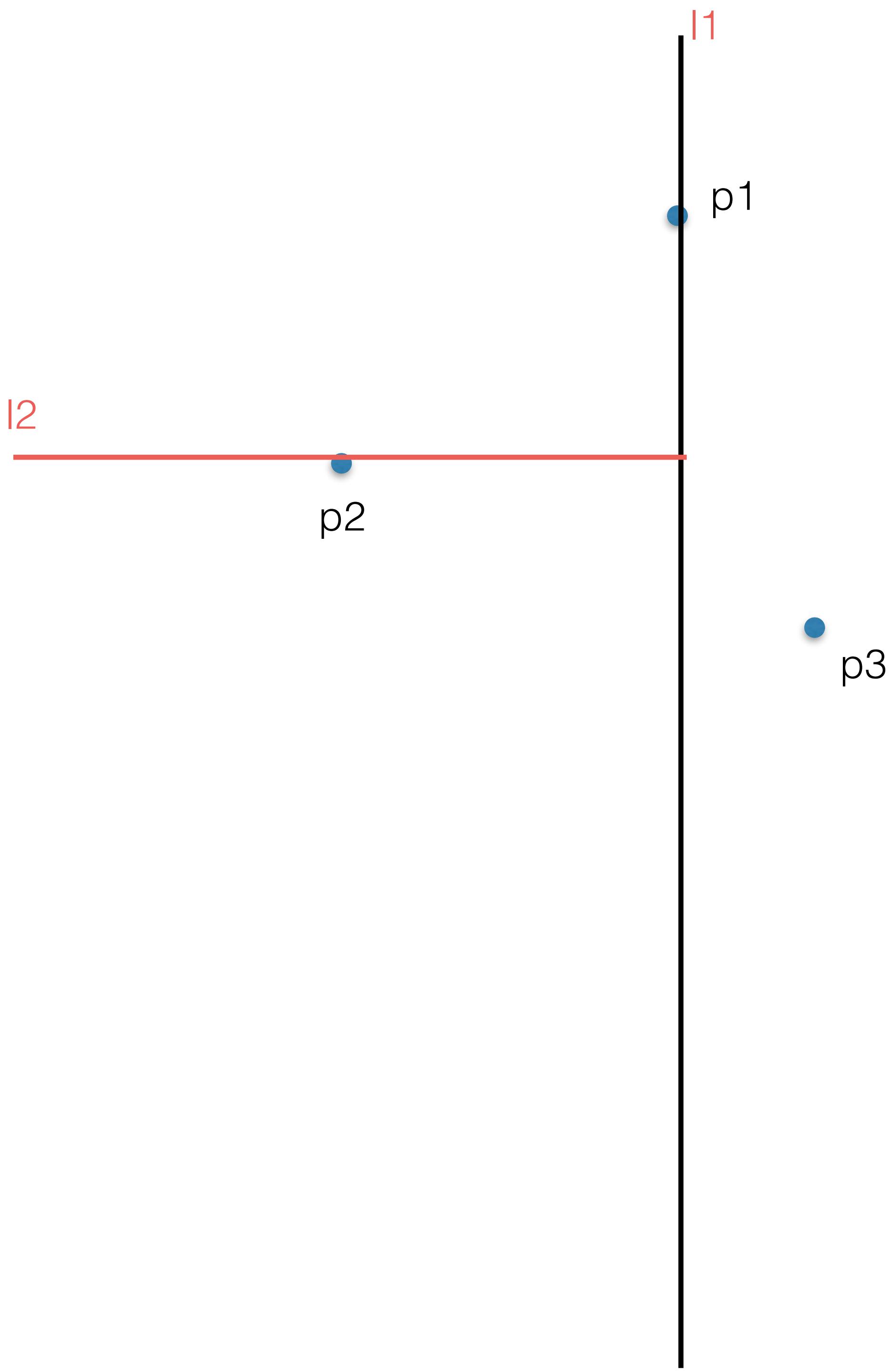
2d binary search trees

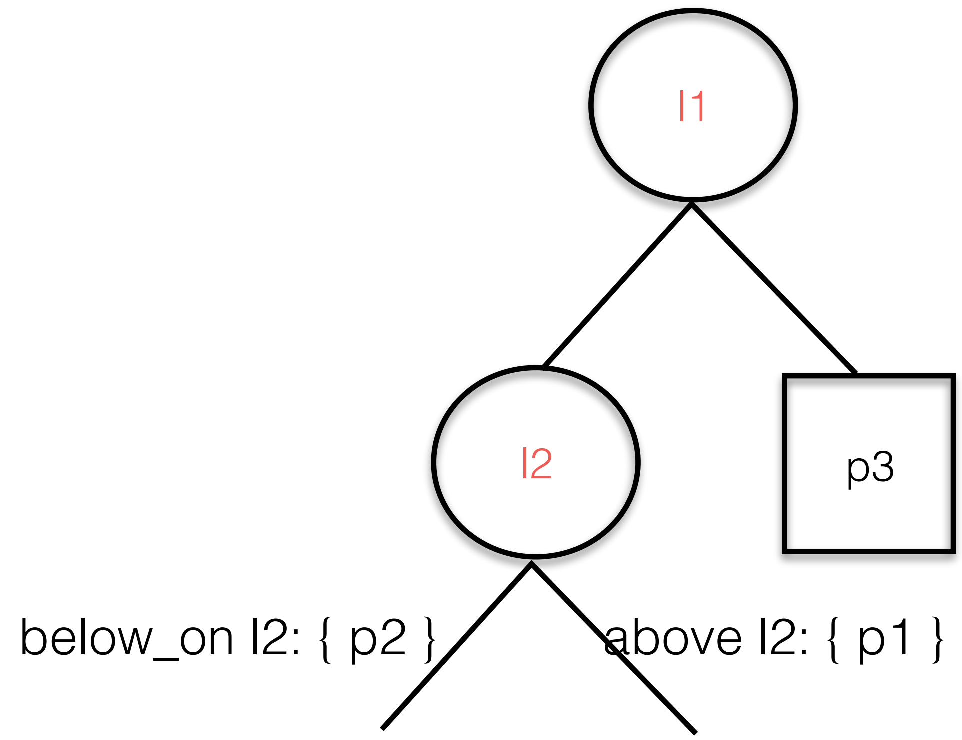
split with horizontal line through y-median
include median in left child



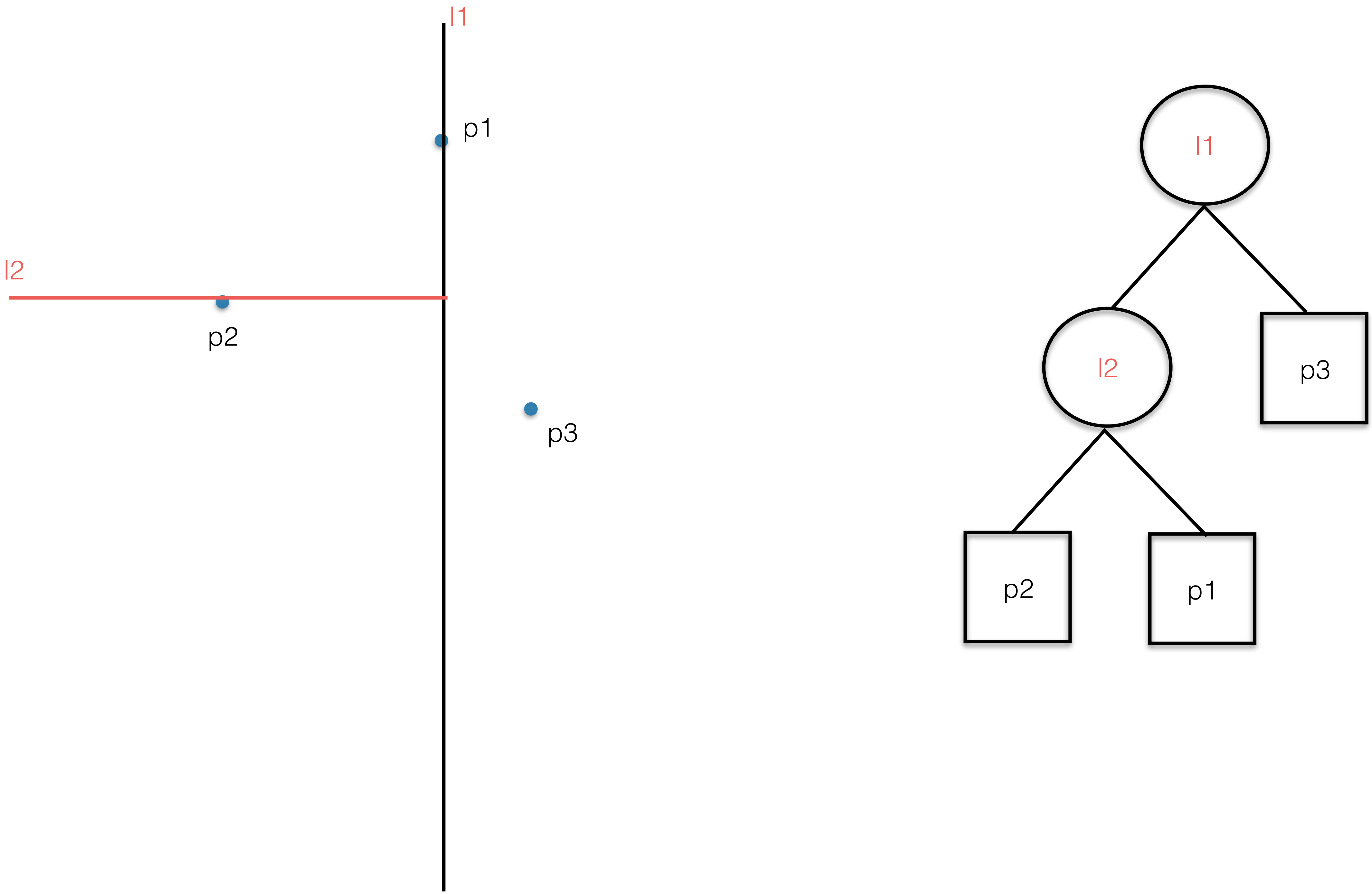
2d binary search trees

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

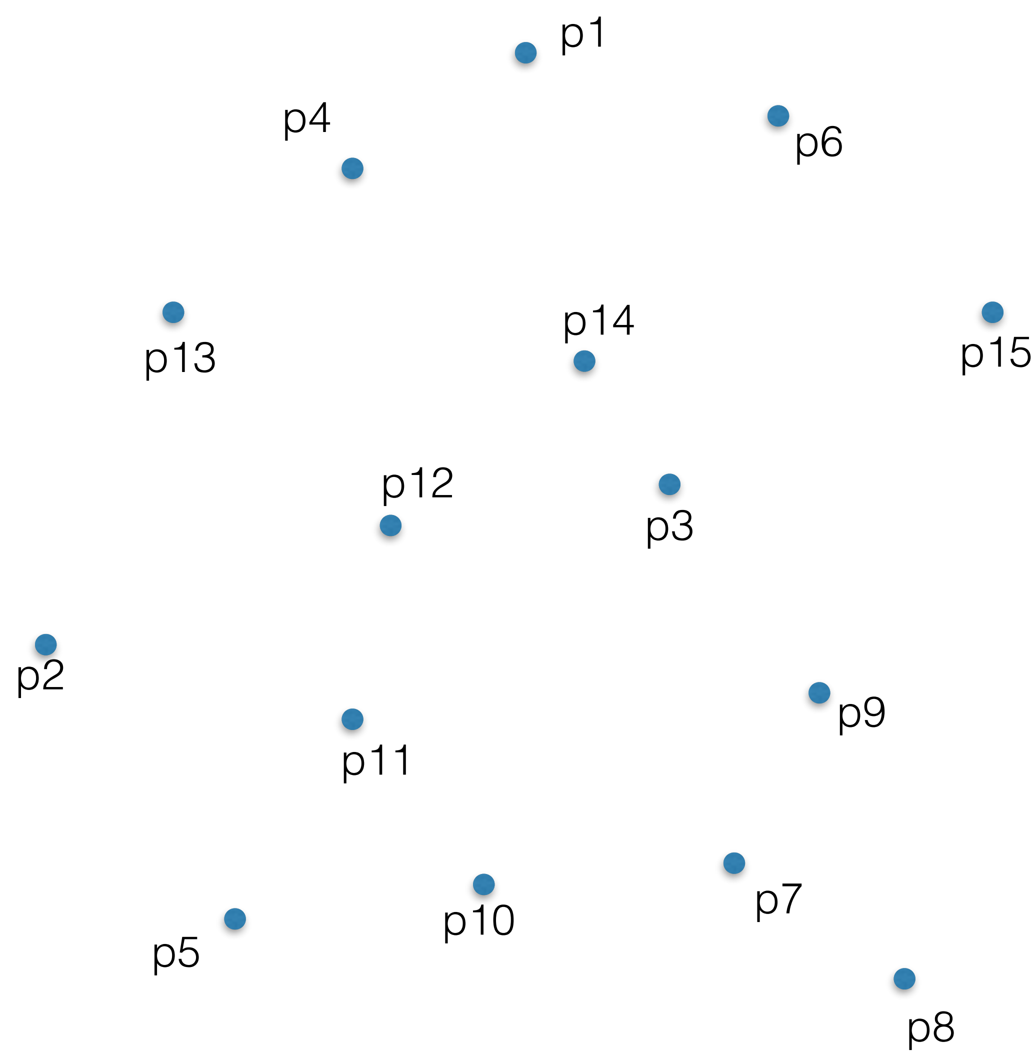




2d binary search trees

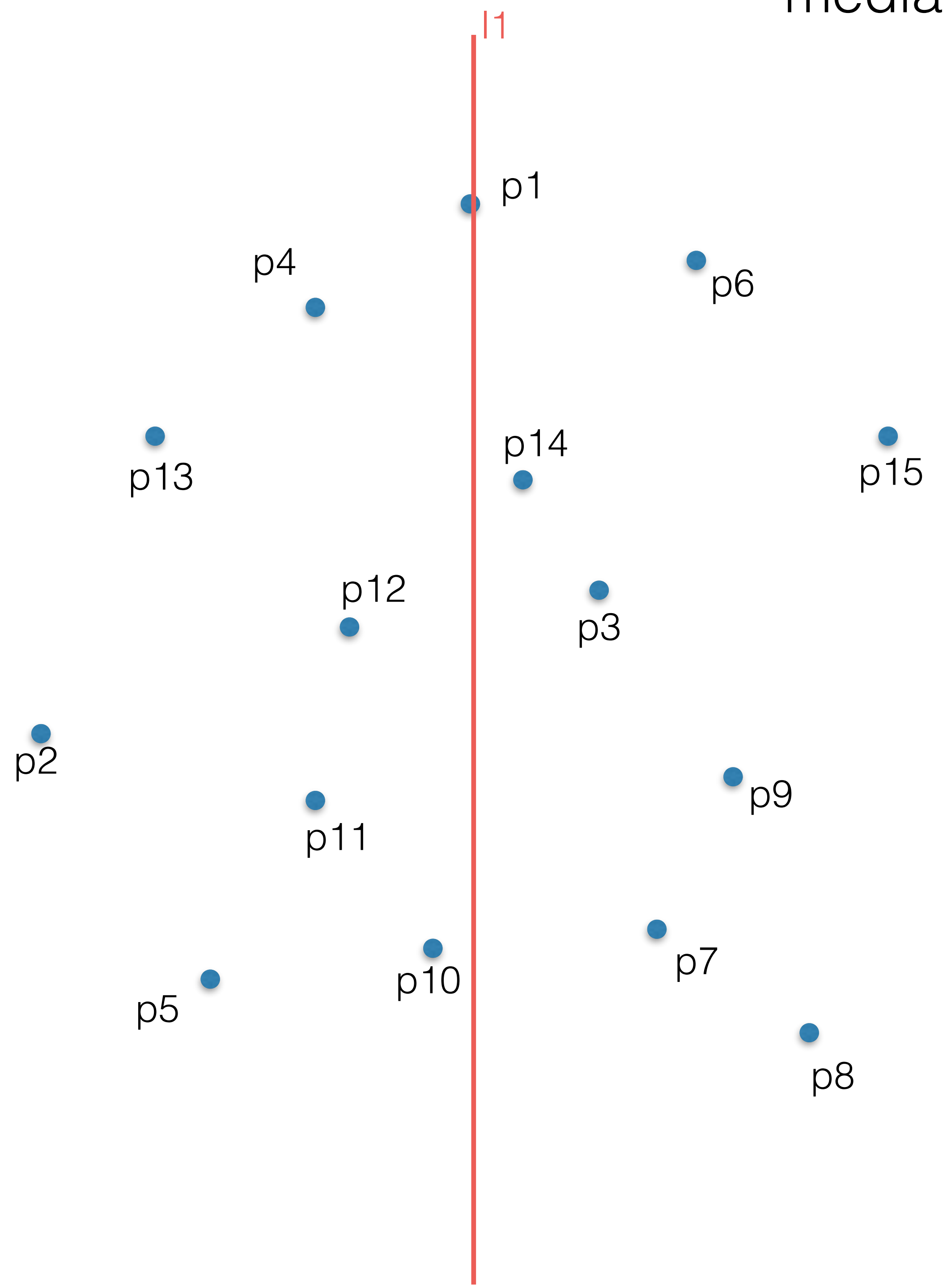


A bigger example



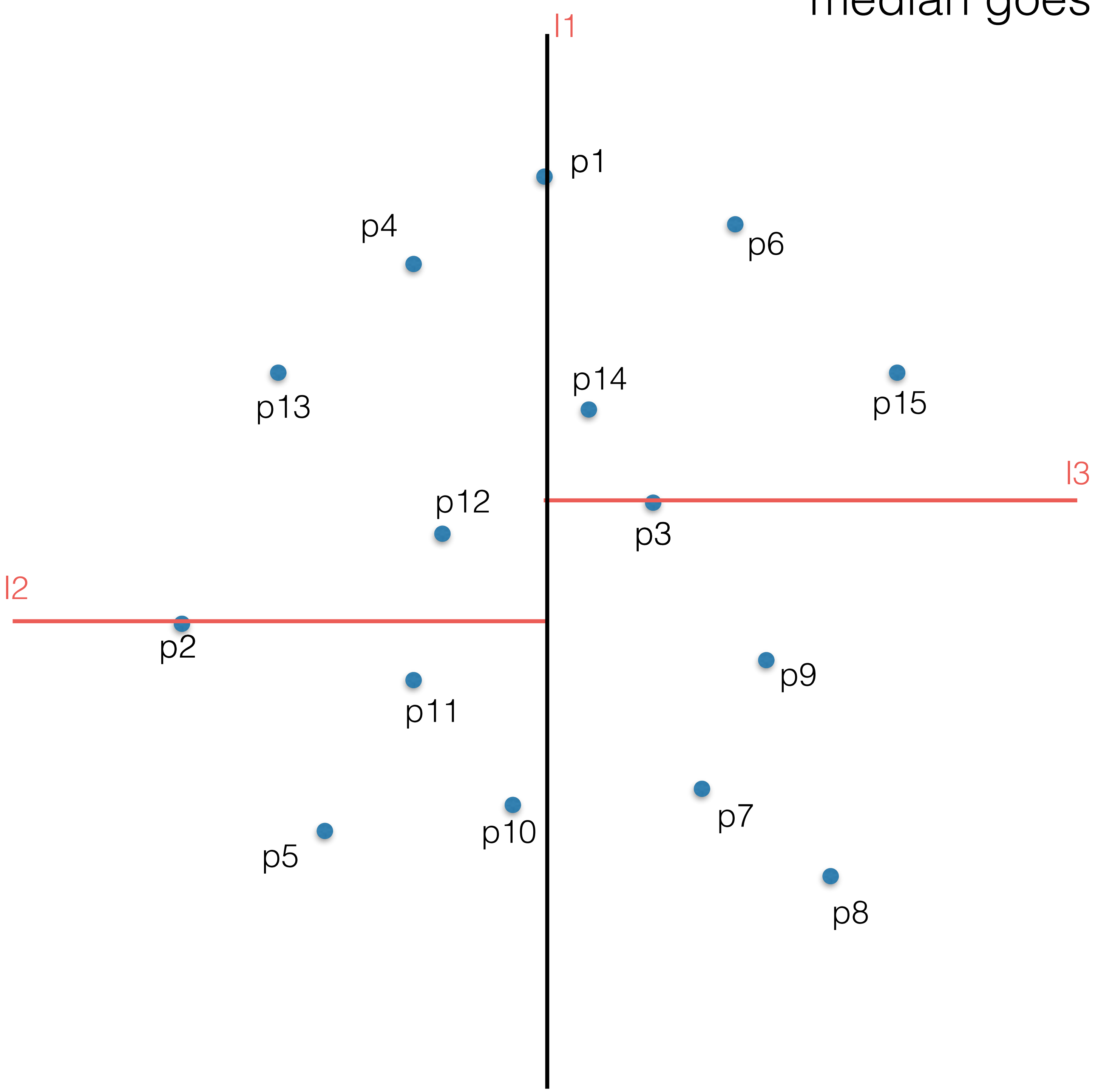
A bigger example

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



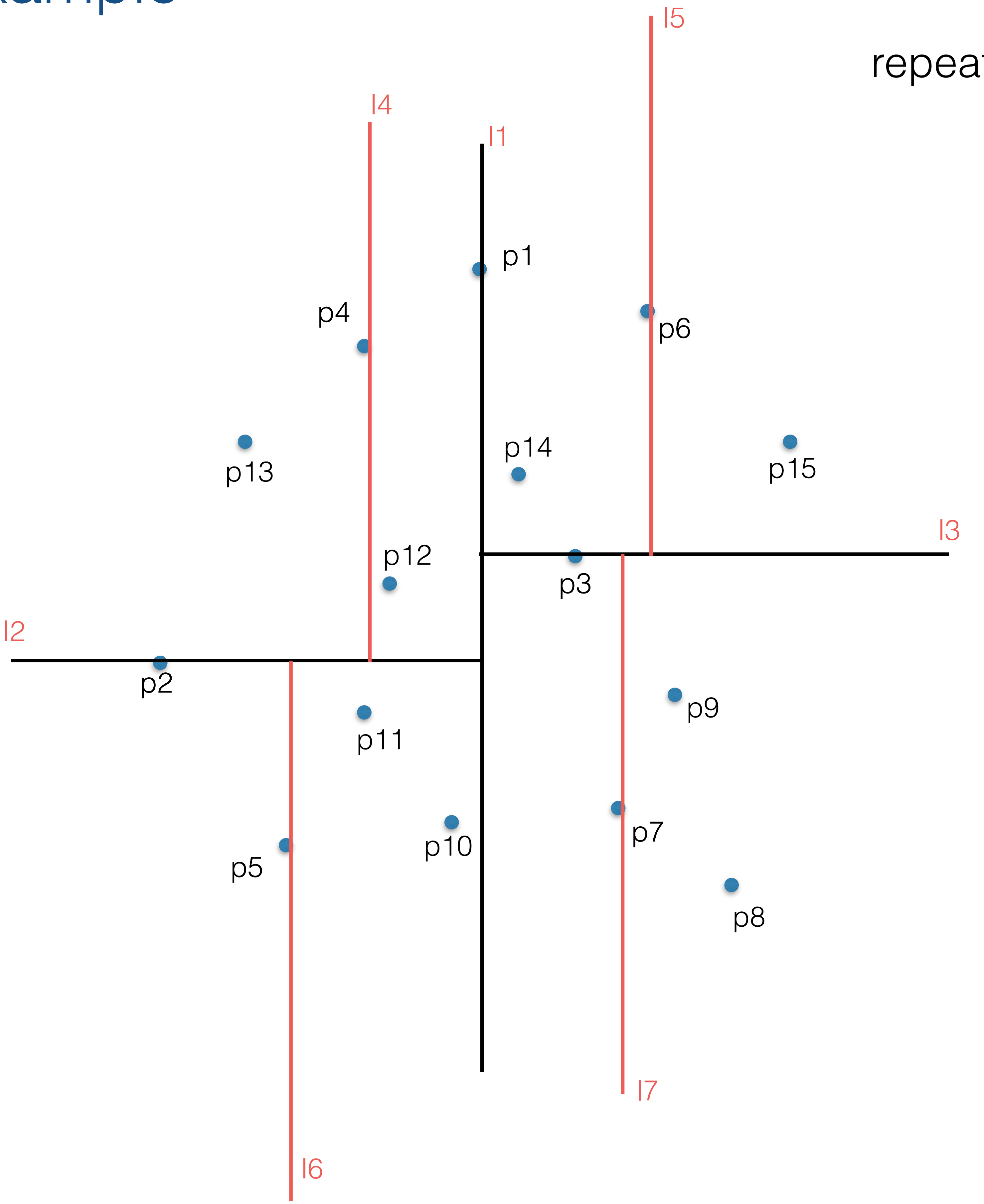
A bigger example

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

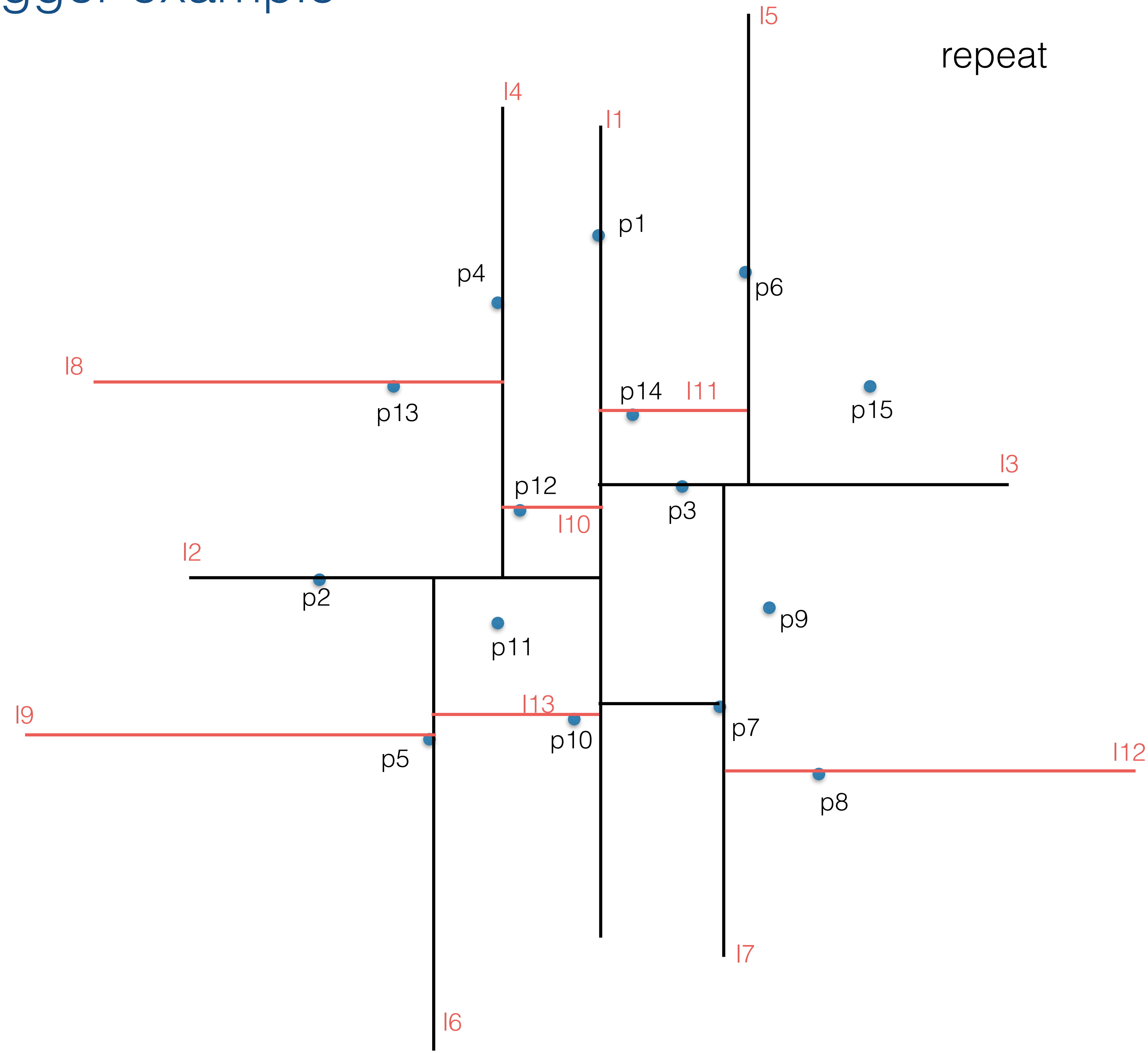


A bigger example

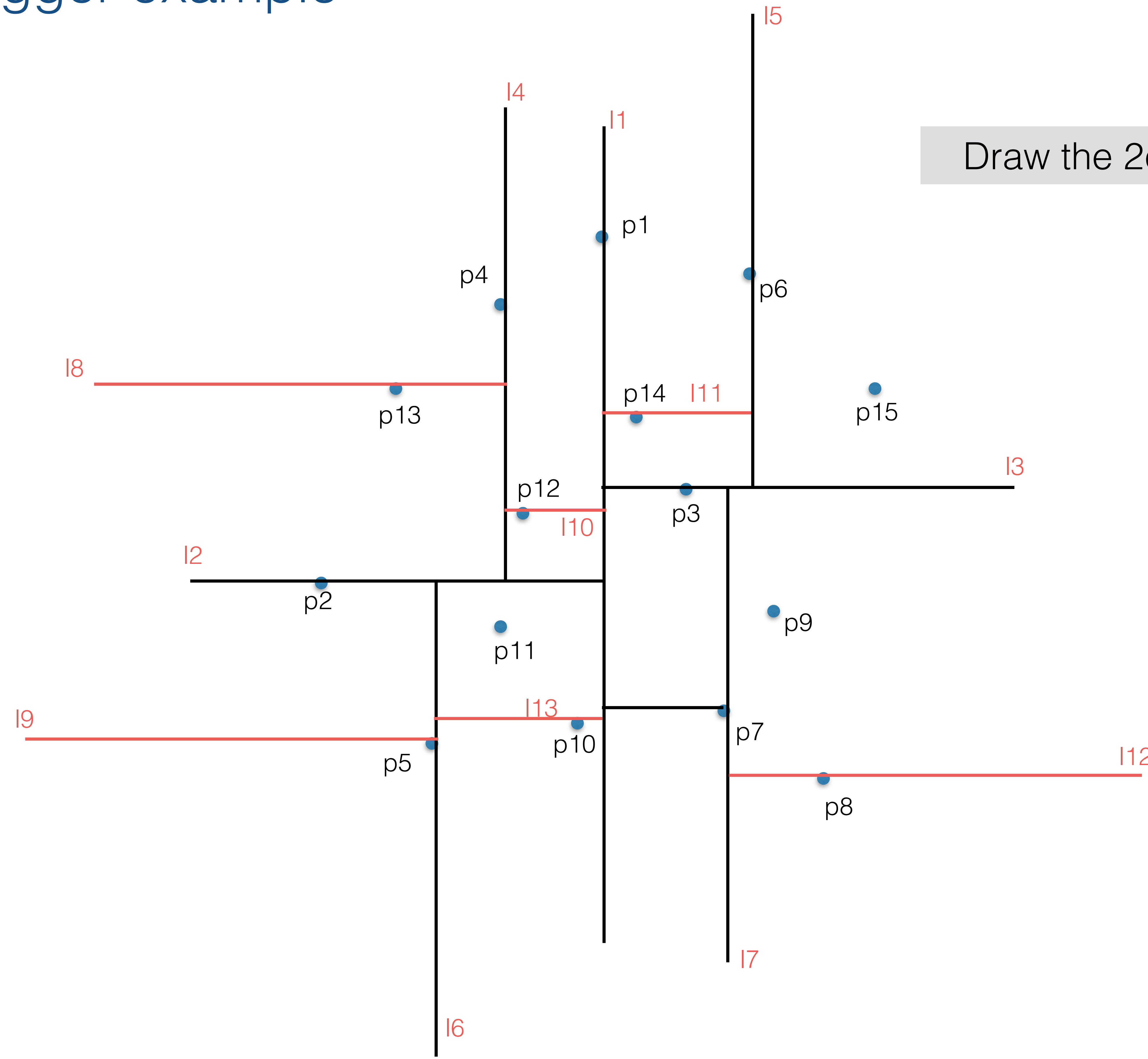
repeat



A bigger example



A bigger example



Draw the 2d-tree!

2d binary search trees

Questions

- How much space does it take?
- How to build it and how fast?
- How to answer range queries and how fast?

2d binary search trees construction

How fast?

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. $\nu_{\text{left}} \leftarrow$ BUILDKDTREE(P_1, *depth* $+ 1$)
7. $\nu_{\text{right}} \leftarrow$ 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

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

2d binary search trees construction

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 **$O(n \lg n)$**

- 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

How much space does it take?

2d binary search trees

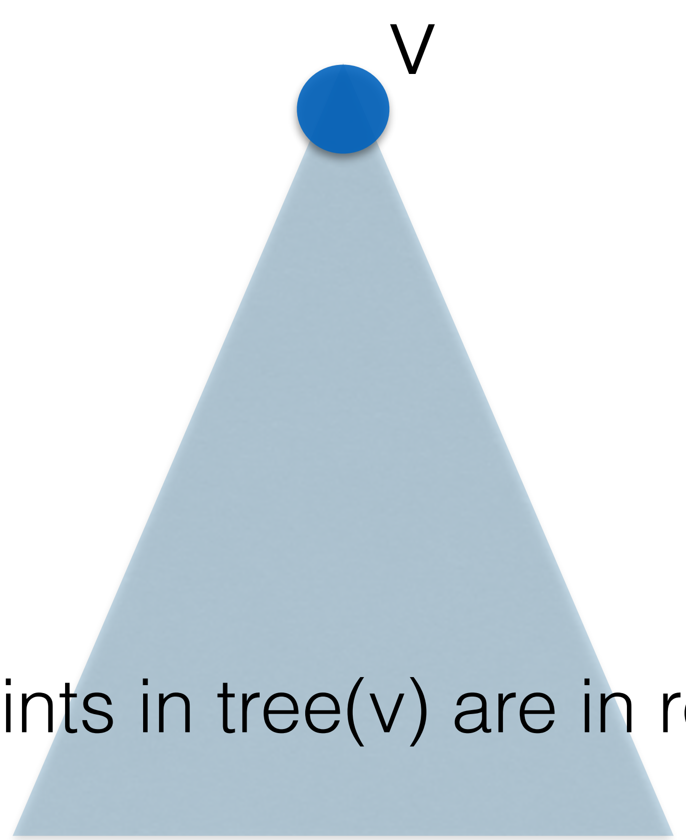
How much space does it take?

O(n)

2d binary search trees

How to answer range queries?

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

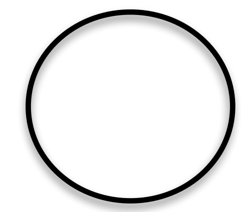


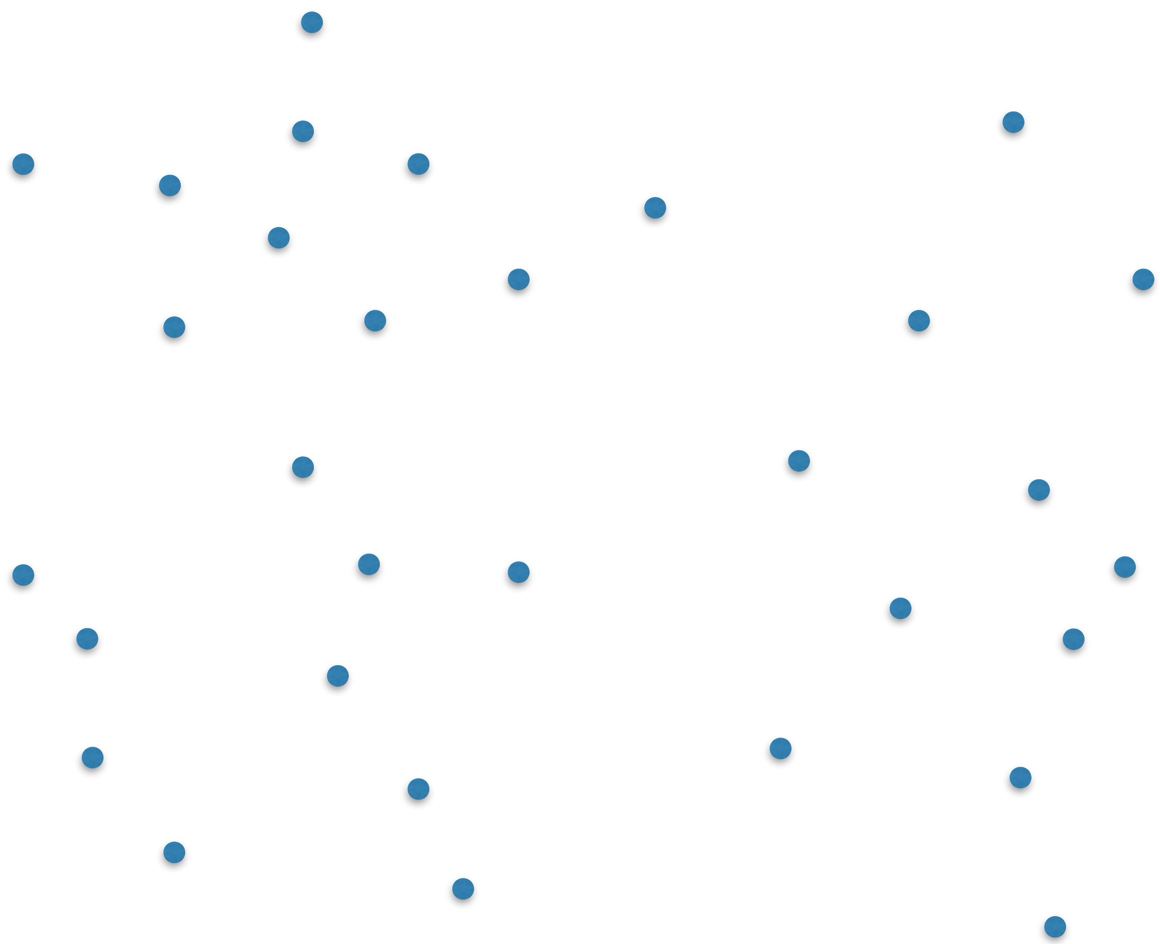
v: node in the tree

region(v):

all points in tree(v) are in region(v)

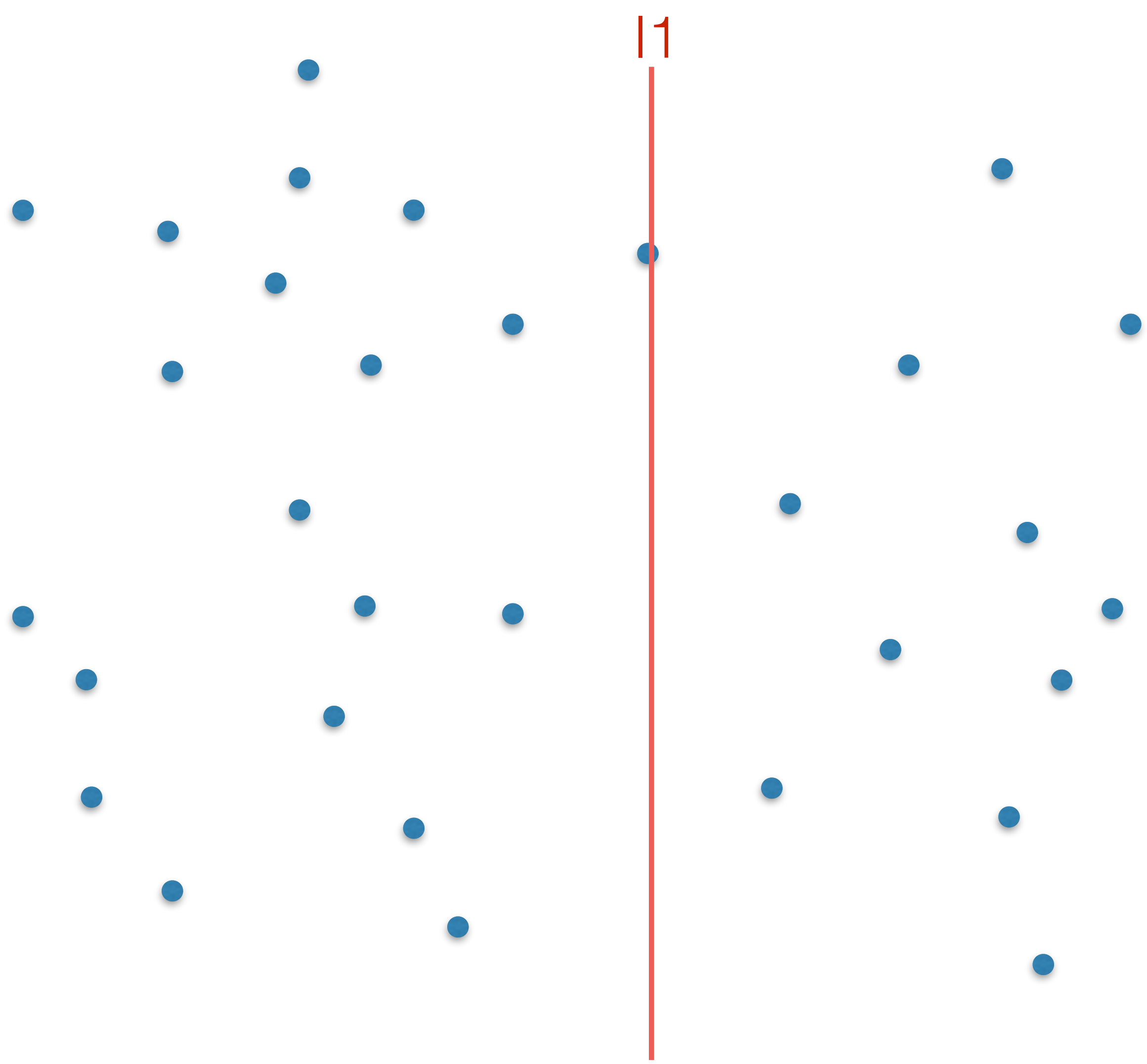
Regions of nodes

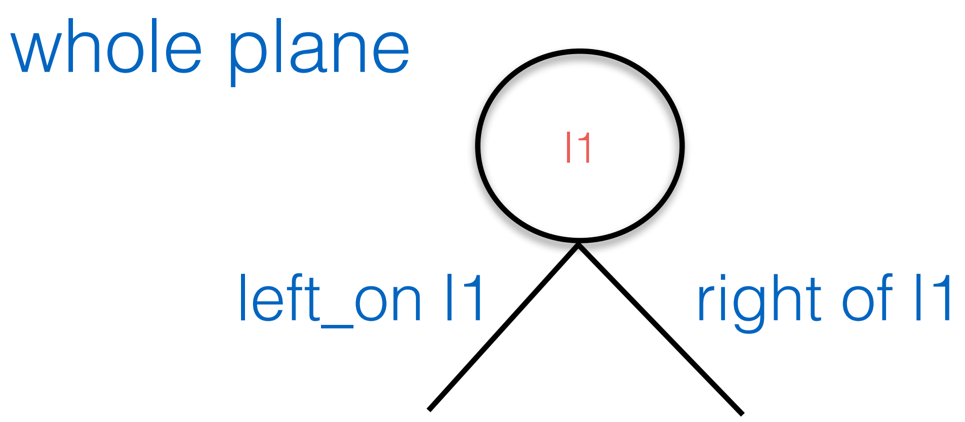




Each node in the tree corresponds to a region in the plane.

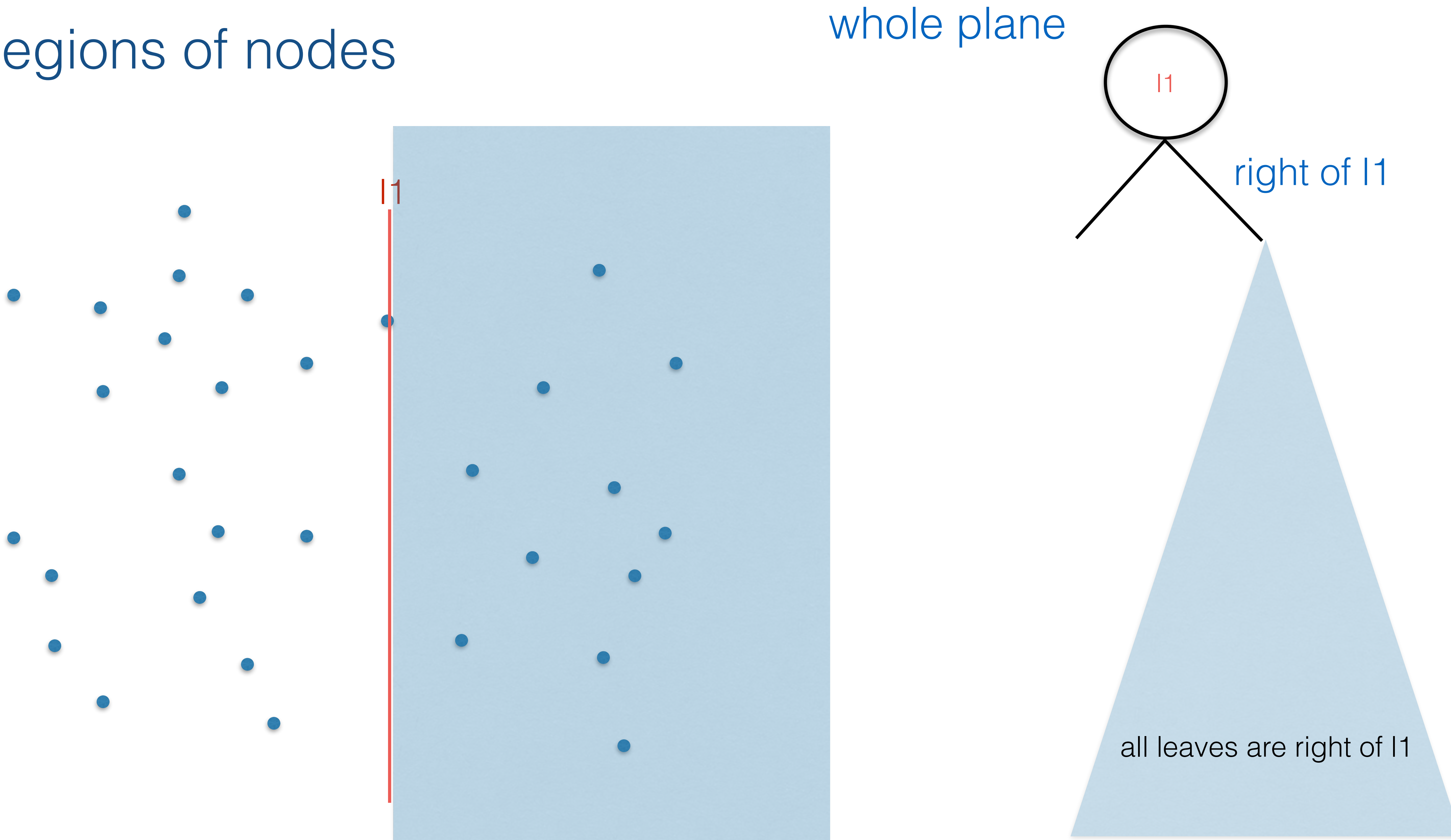
Regions of nodes





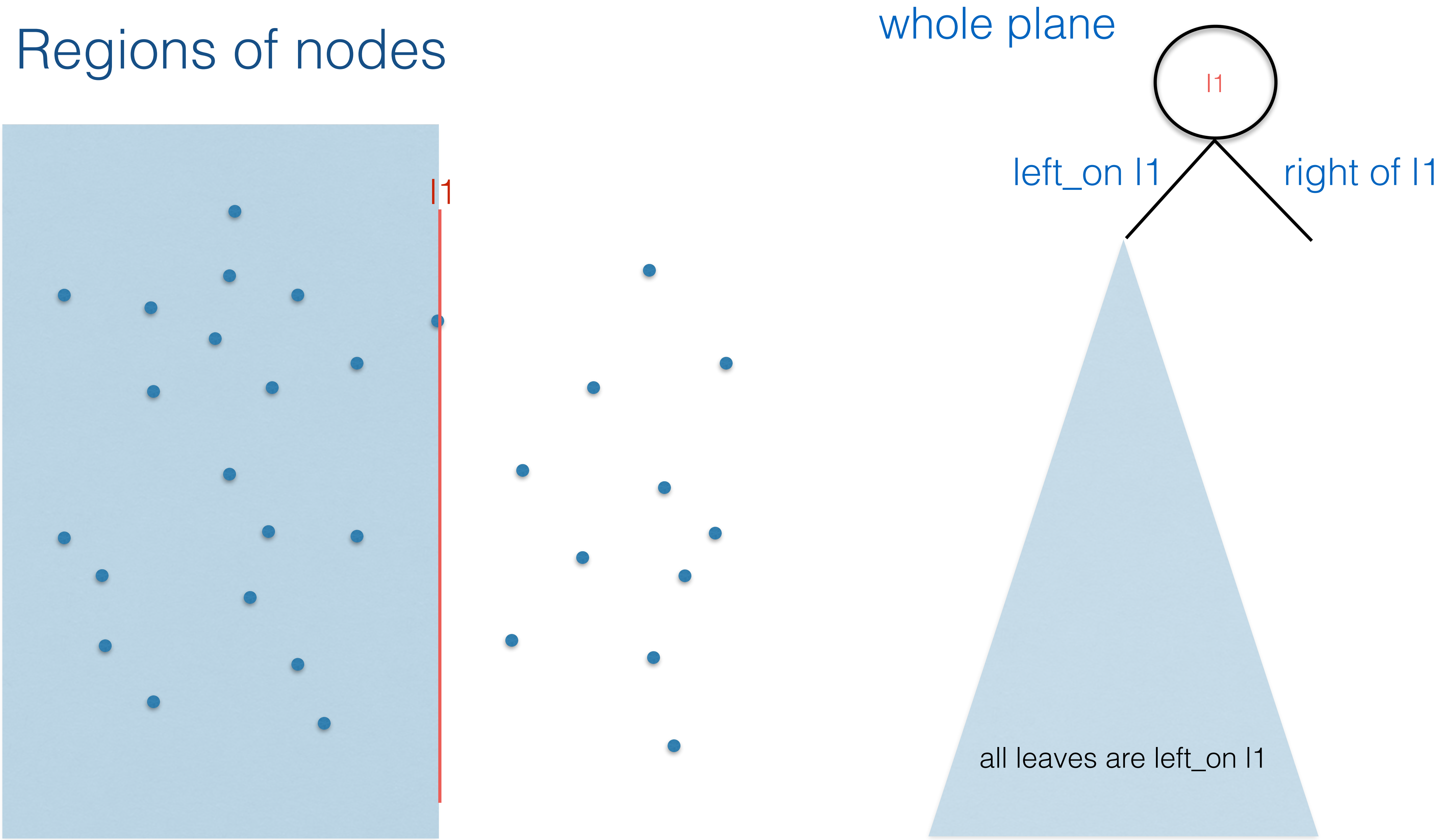
Each node in the tree corresponds to a region in the plane.

Regions of nodes



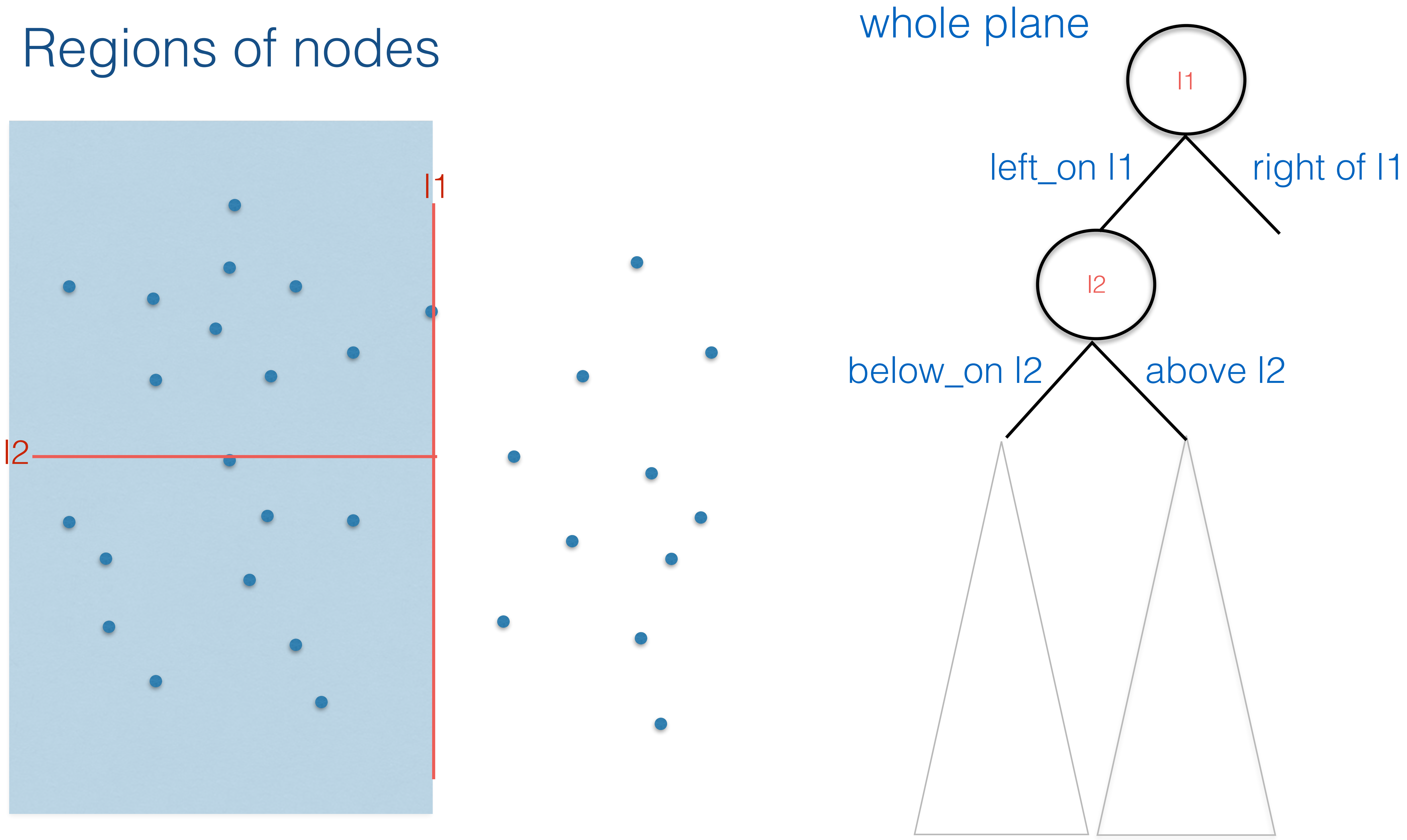
Each node in the tree corresponds to a region in the plane.

Regions of nodes



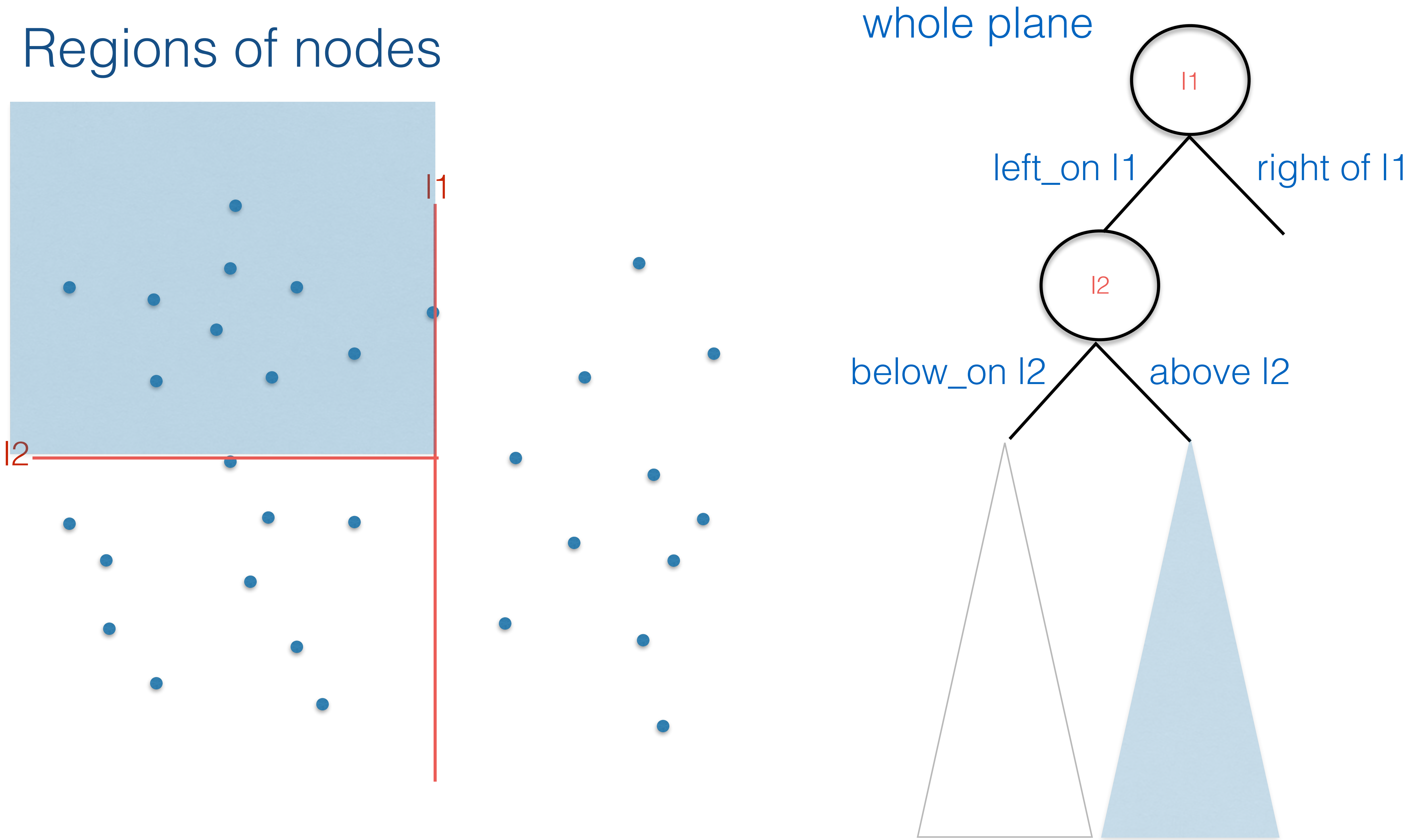
Each node in the tree corresponds to a region in the plane.

Regions of nodes



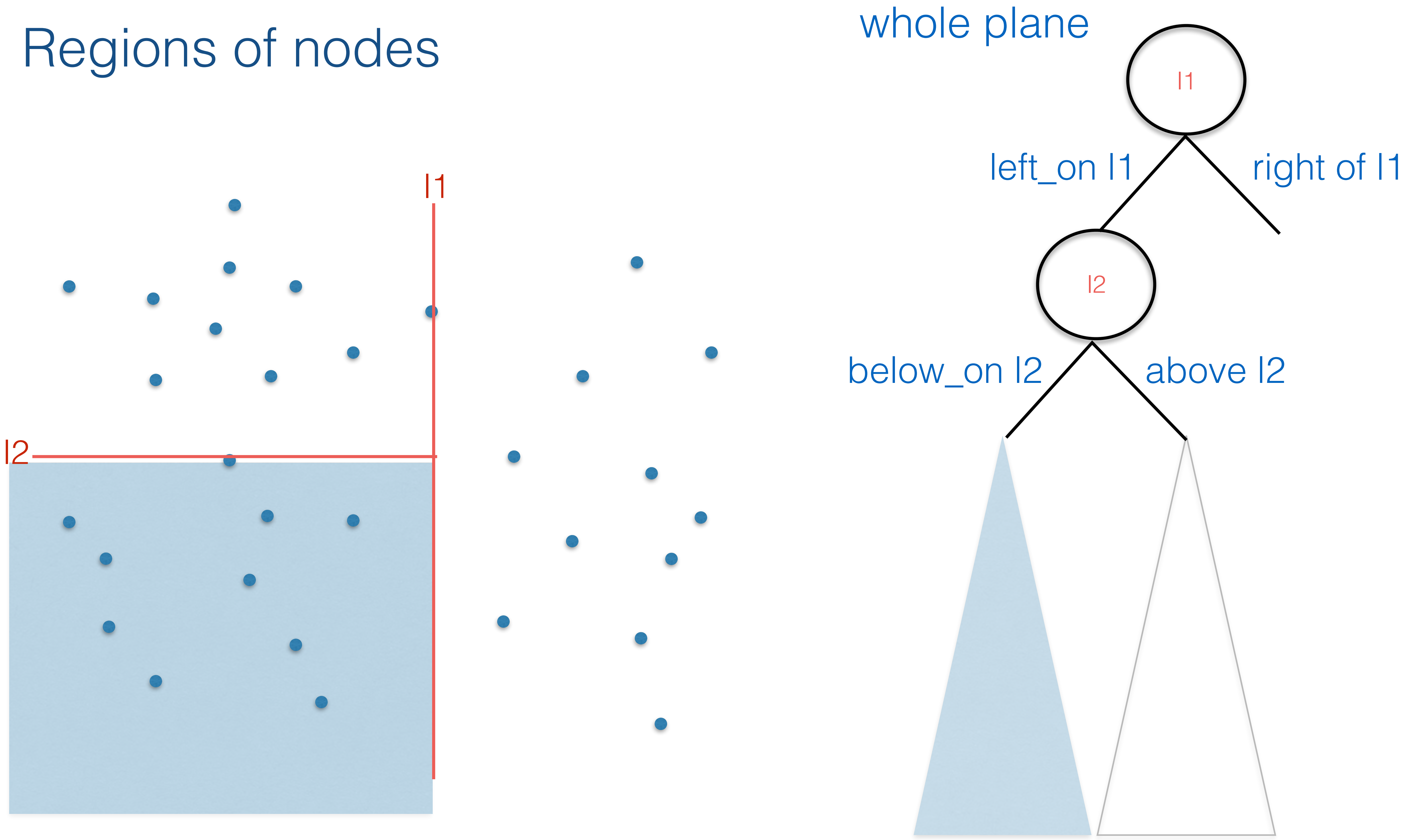
Each node in the tree corresponds to a region in the plane.

Regions of nodes



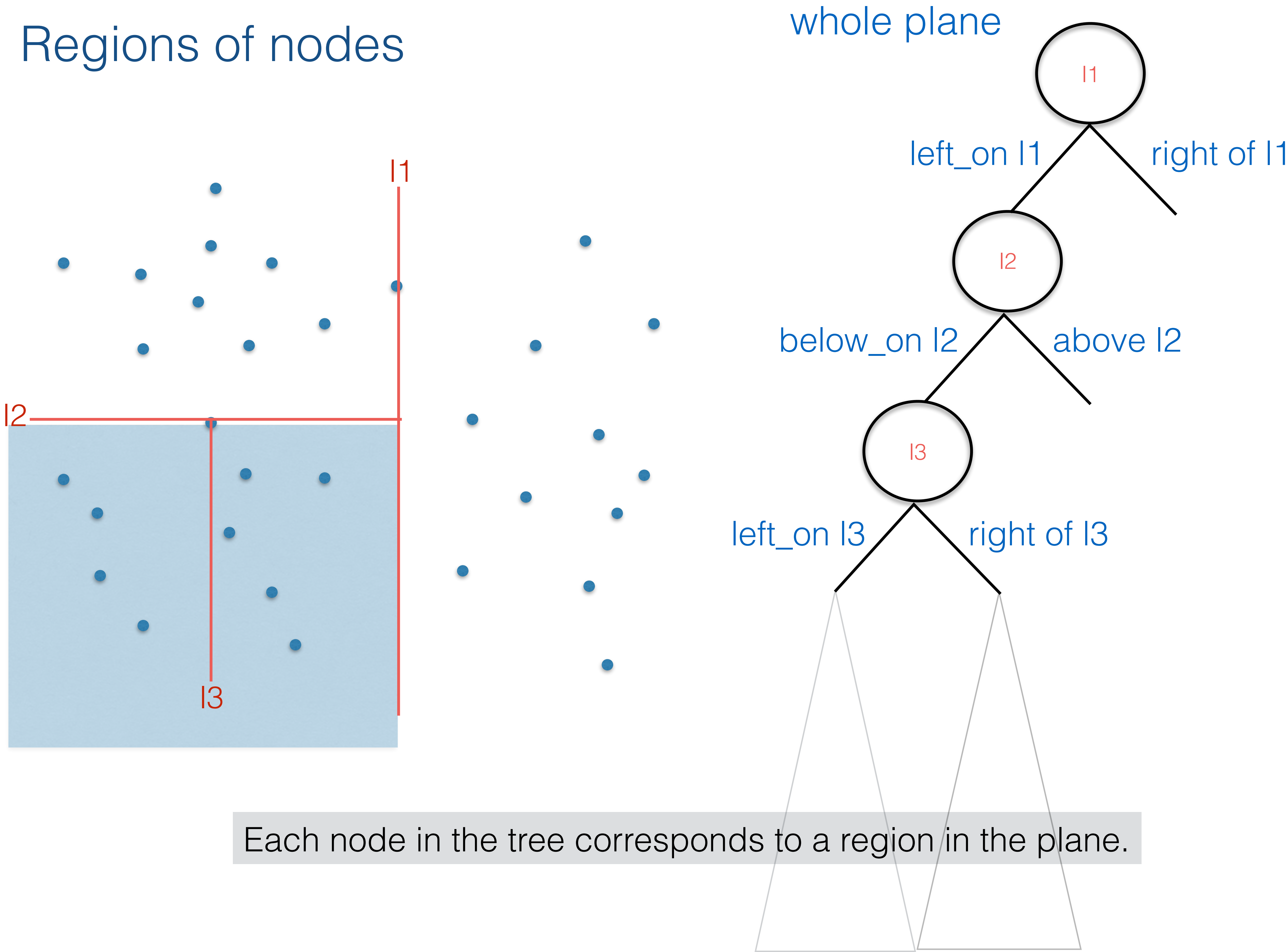
Each node in the tree corresponds to a region in the plane.

Regions of nodes



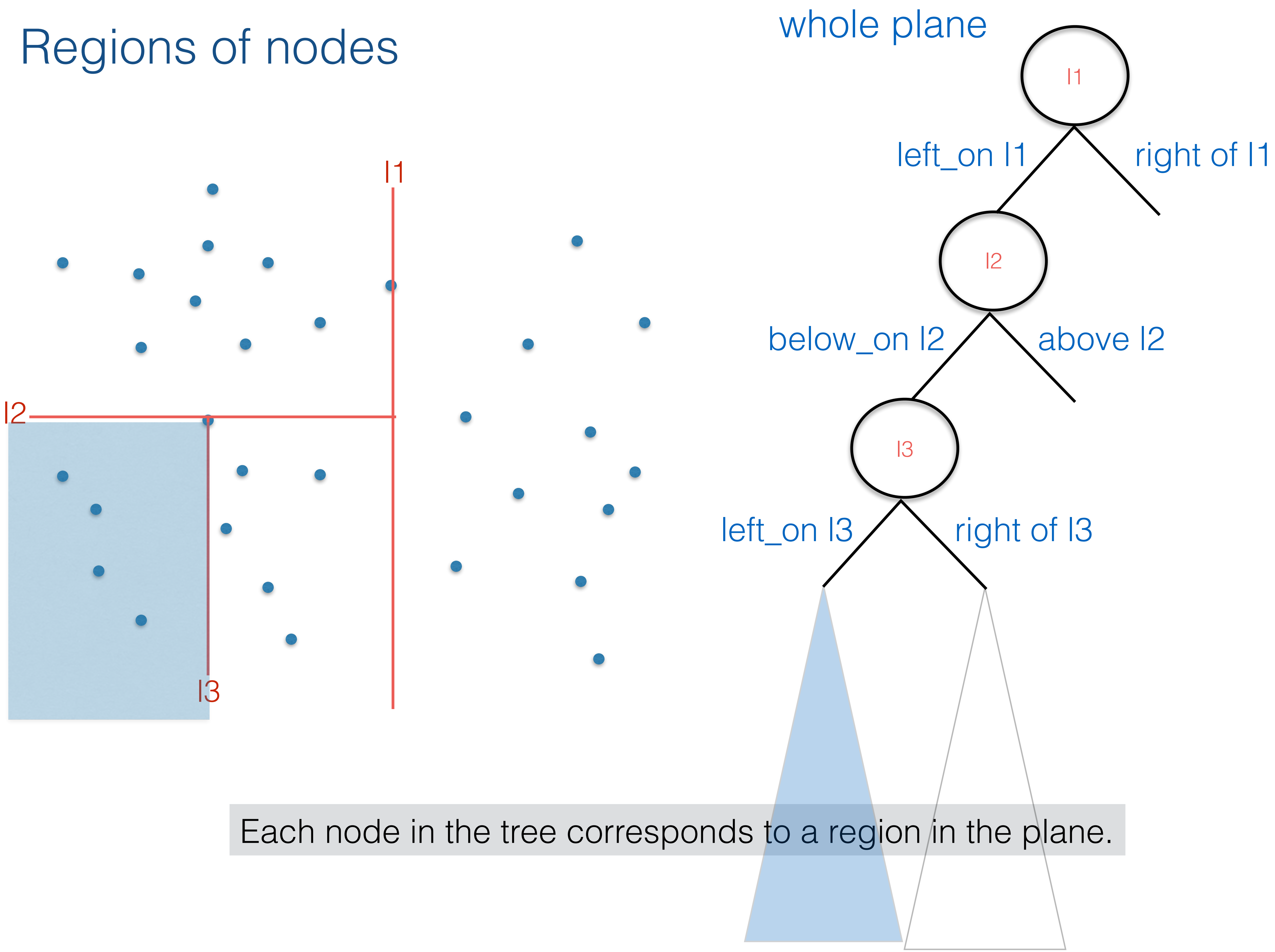
Each node in the tree corresponds to a region in the plane.

Regions of nodes



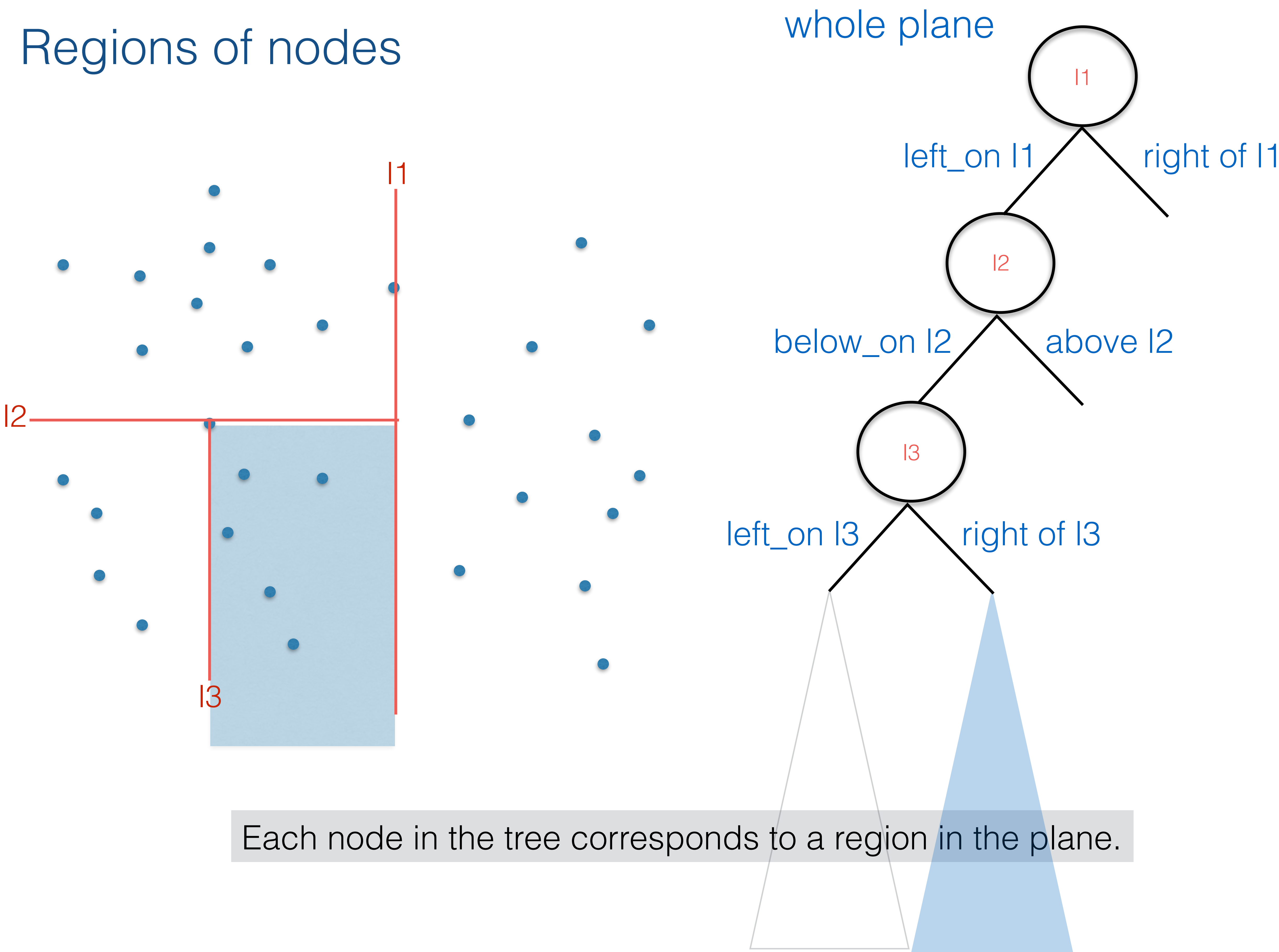
Each node in the tree corresponds to a region in the plane.

Regions of nodes



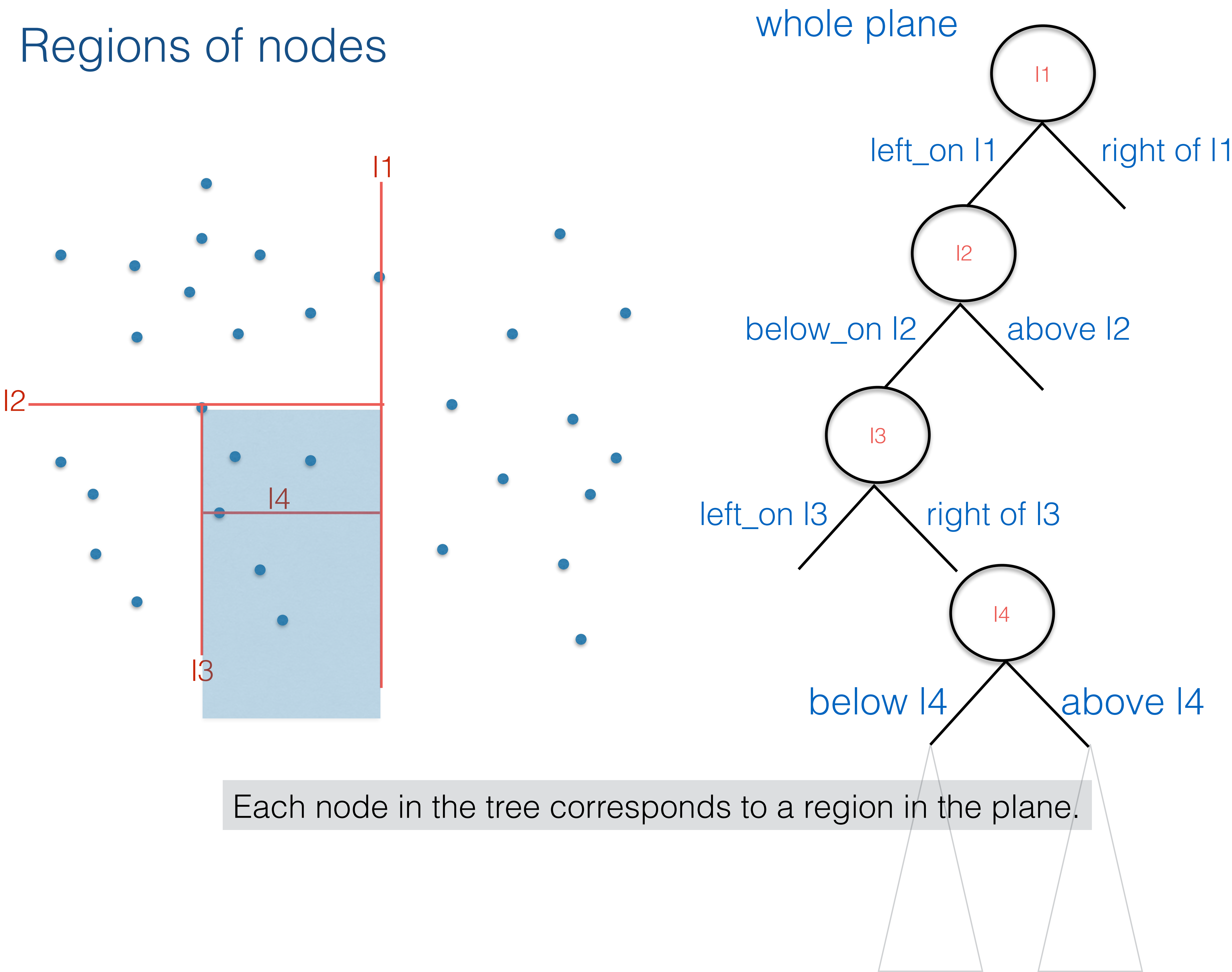
Each node in the tree corresponds to a region in the plane.

Regions of nodes



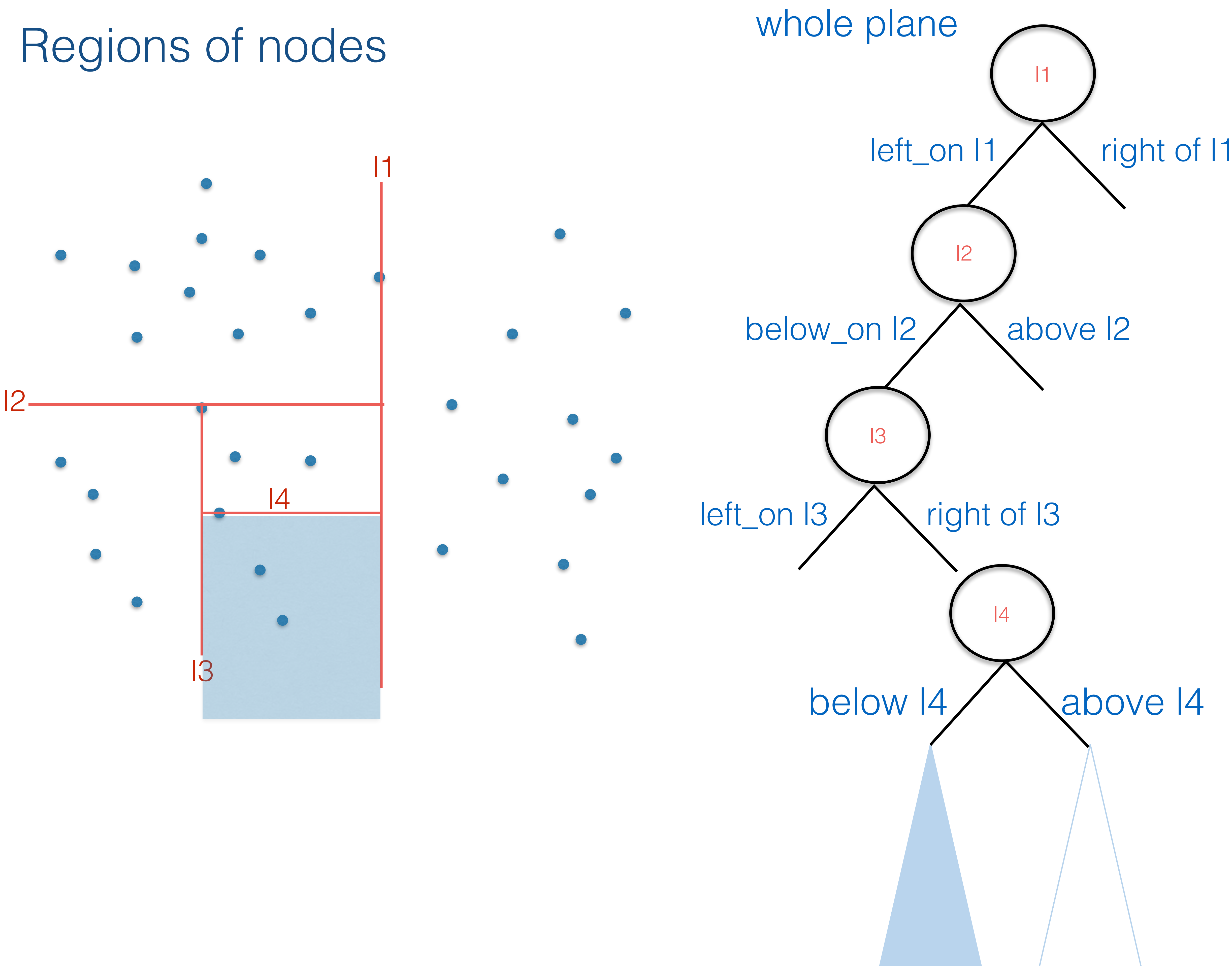
Each node in the tree corresponds to a region in the plane.

Regions of nodes

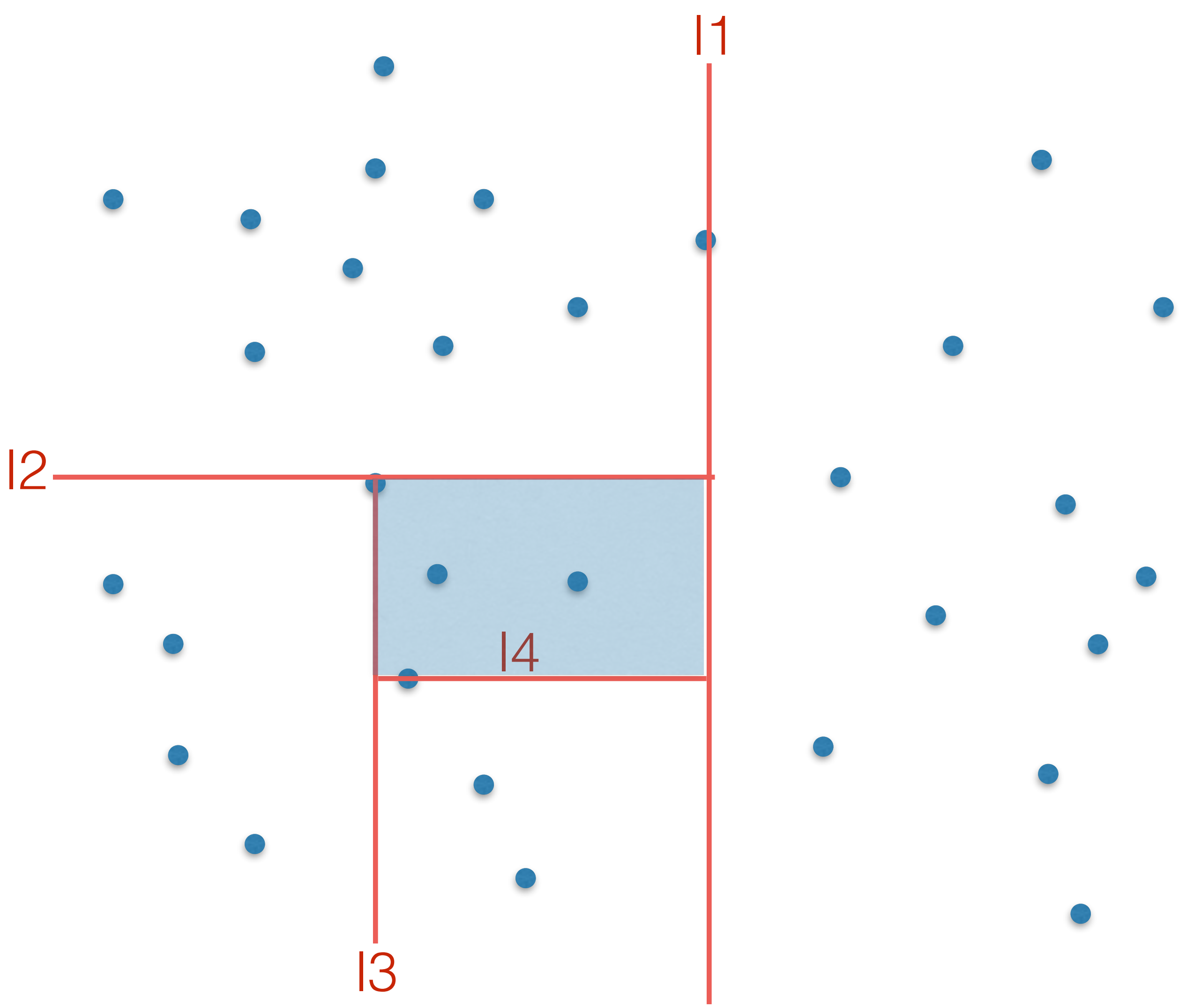


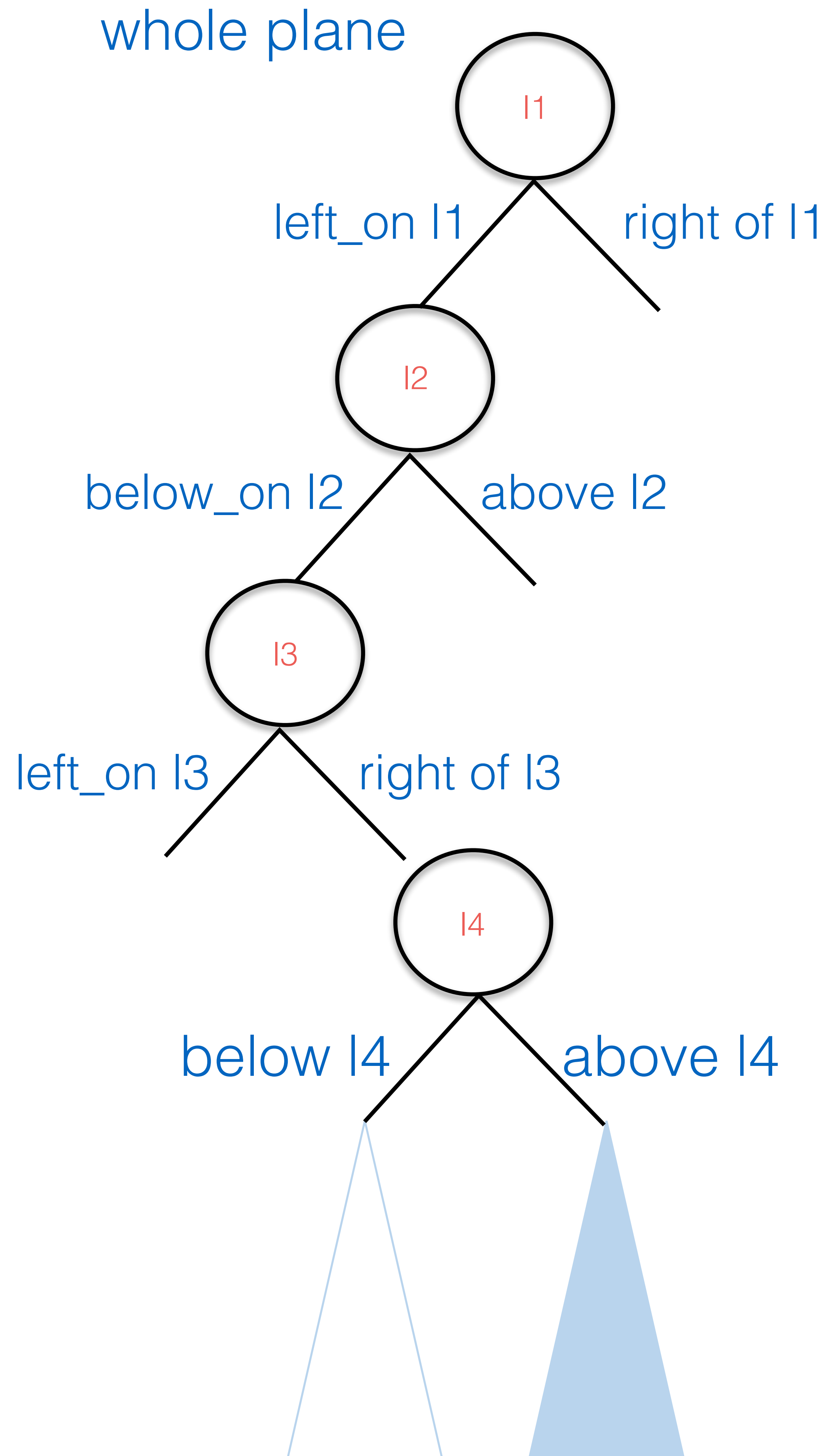
Each node in the tree corresponds to a region in the plane.

Regions of nodes

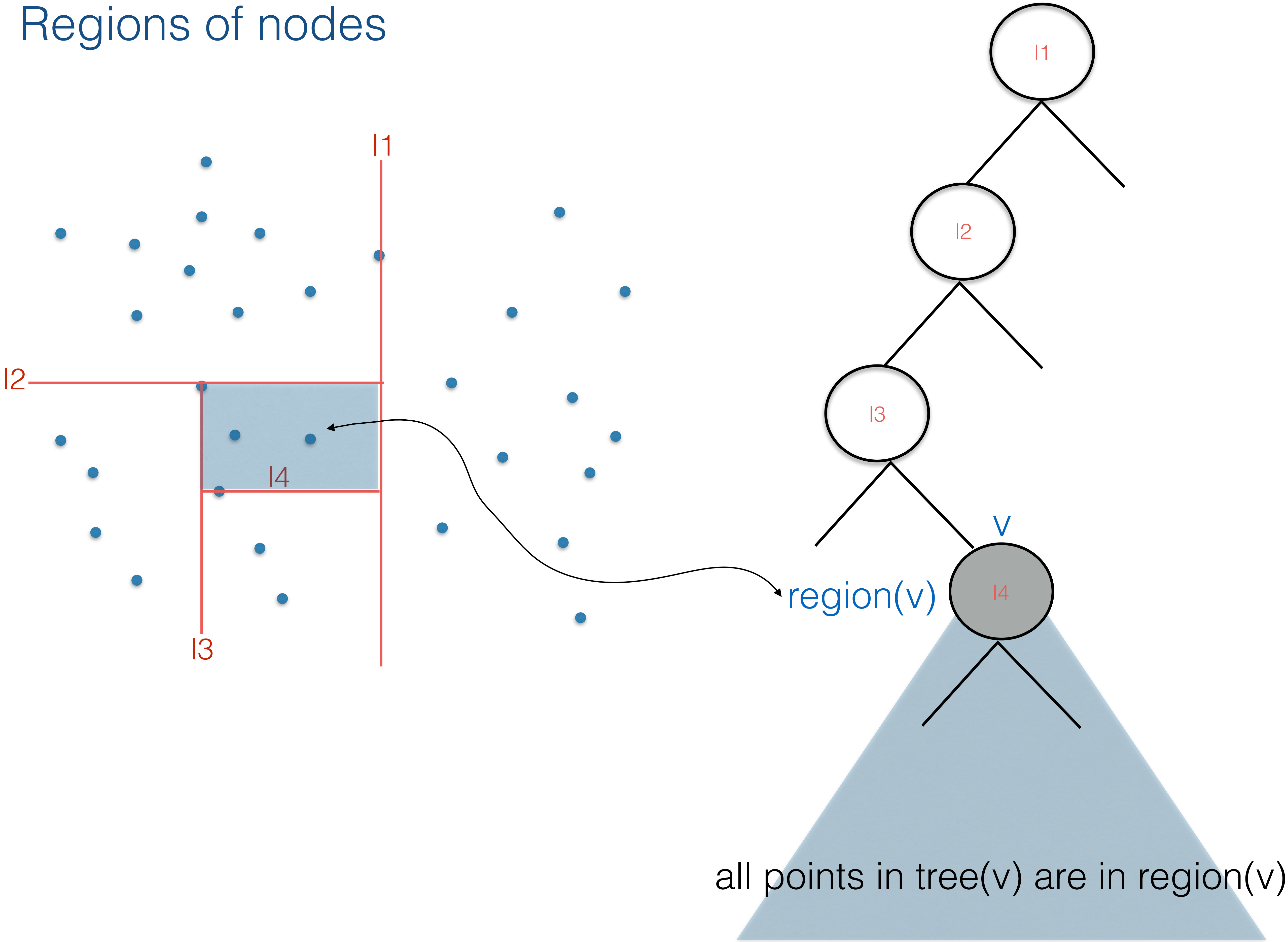


Regions of nodes

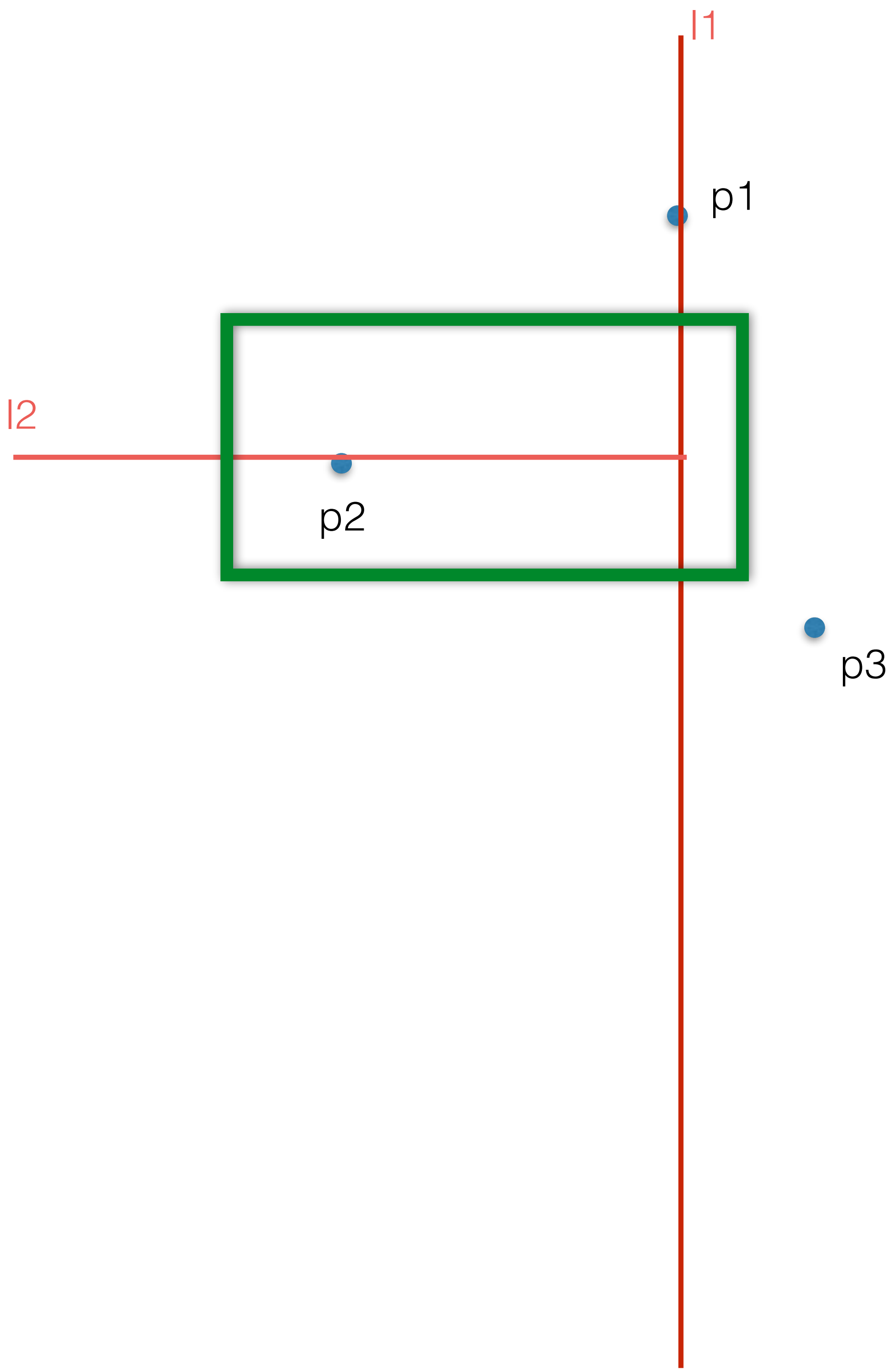


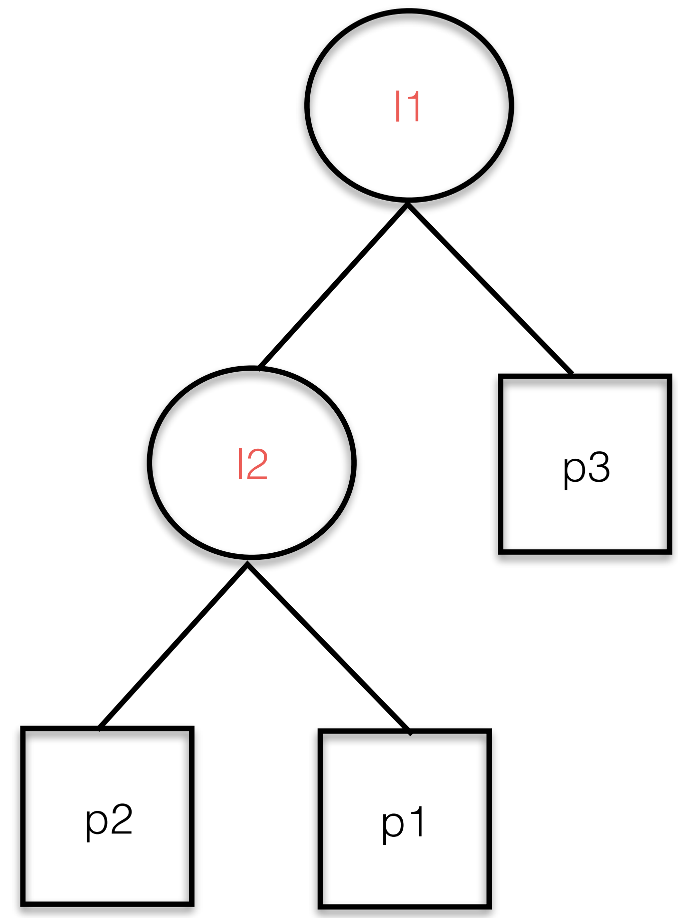


Regions of nodes

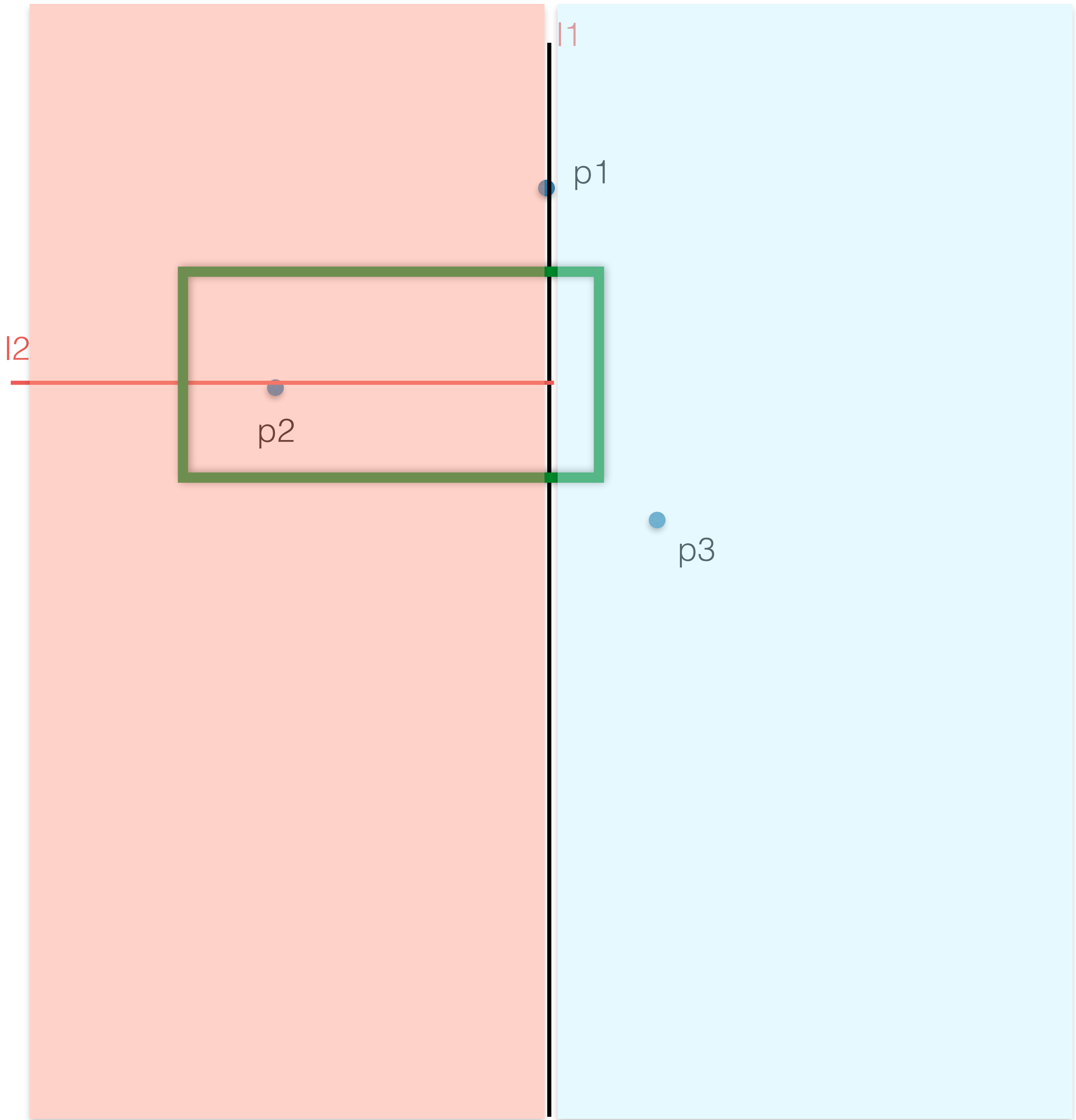


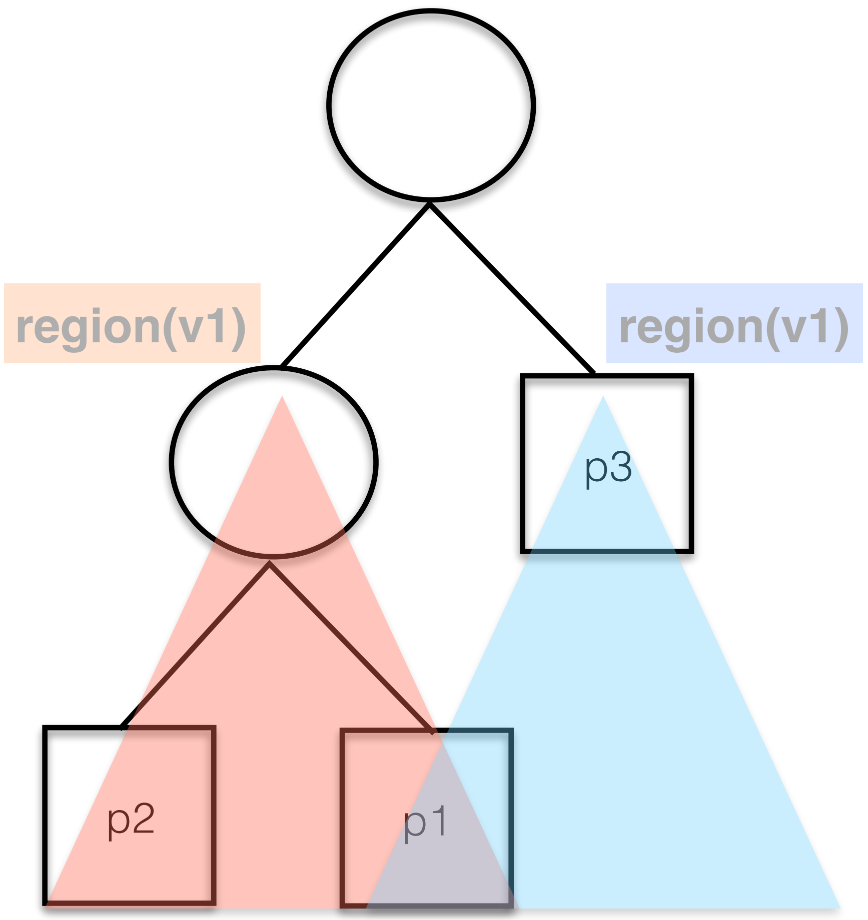
Range queries on 2d binary search trees



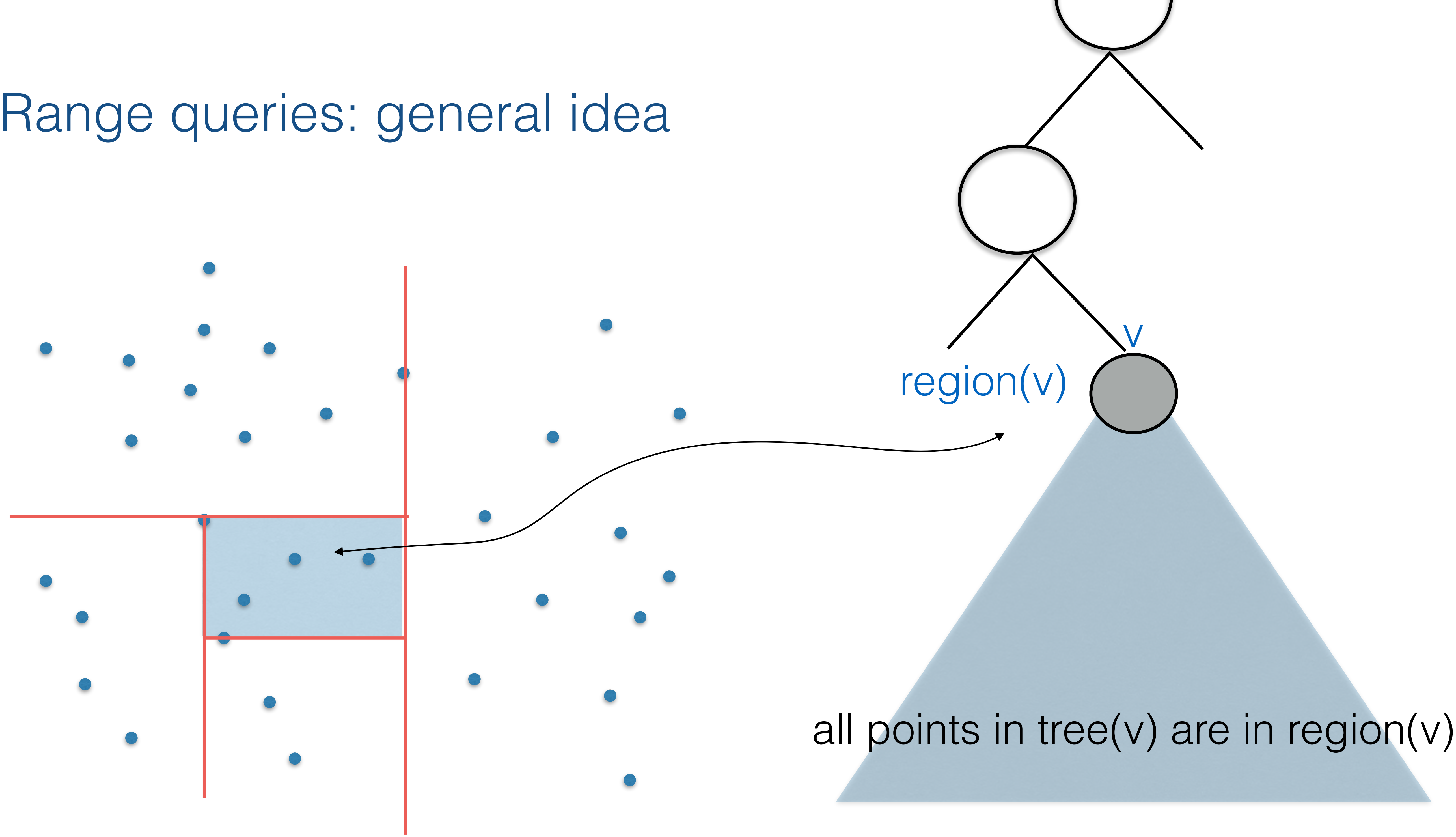


Range queries on 2d binary search trees

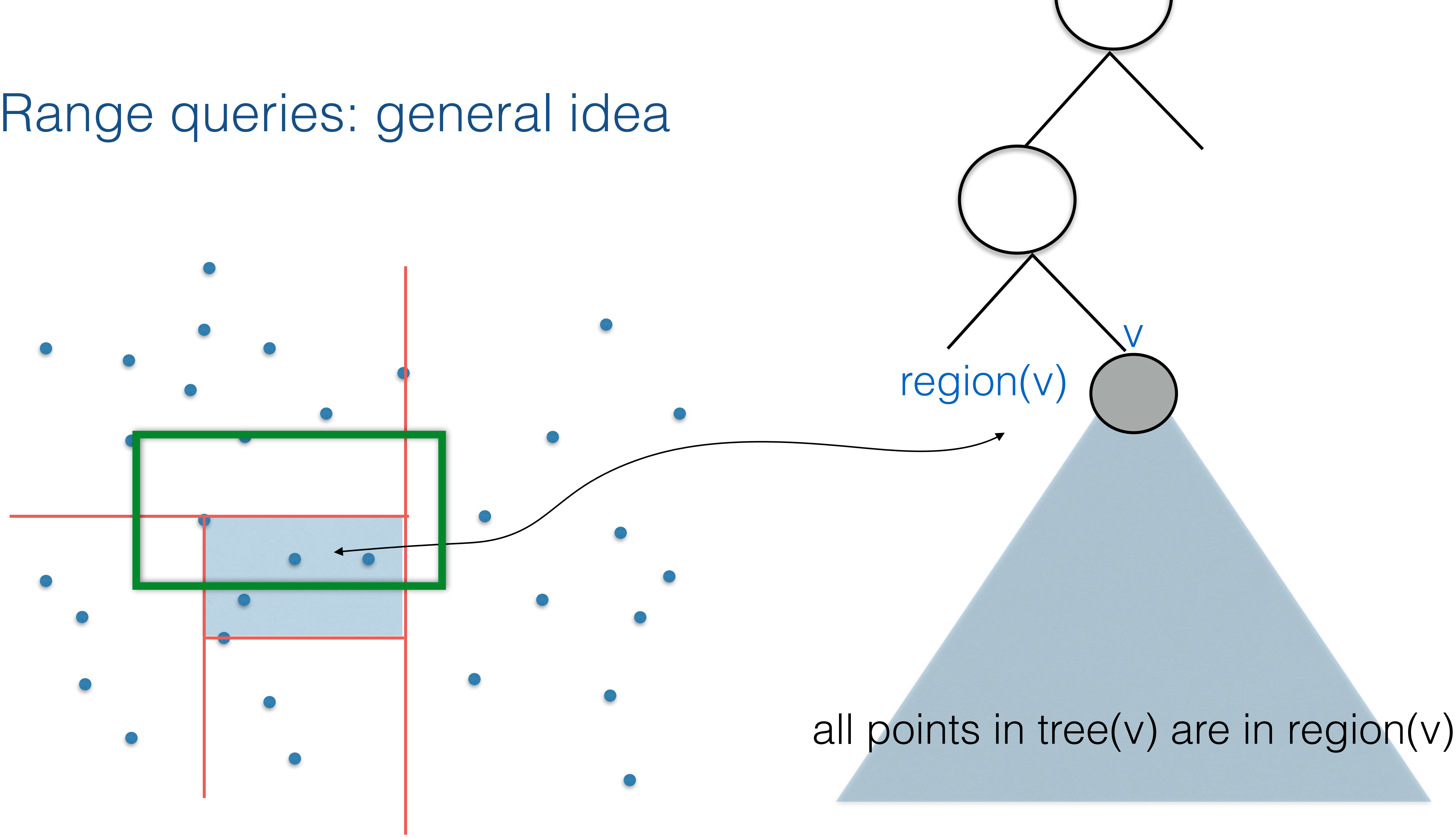




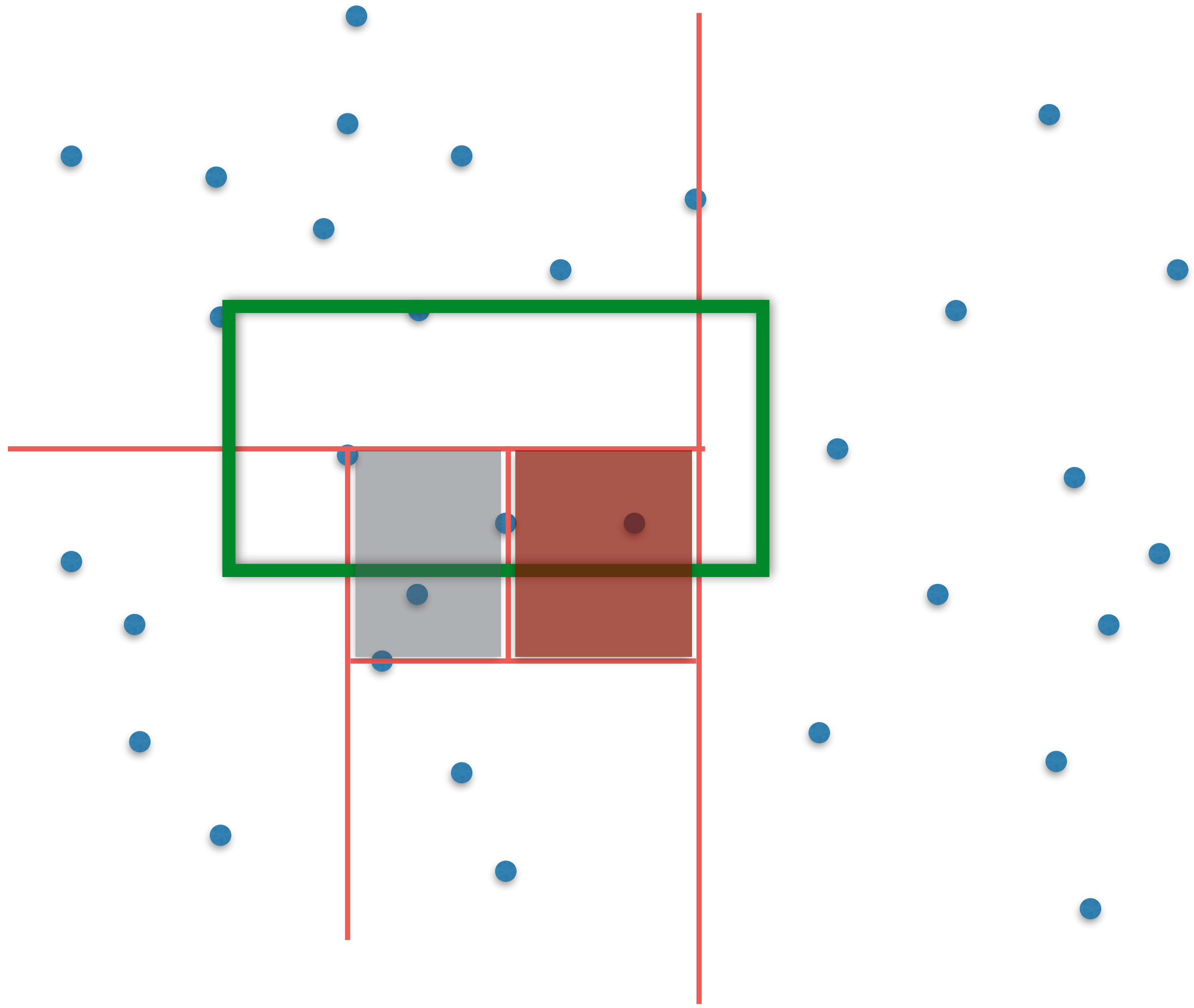
Range queries: general idea

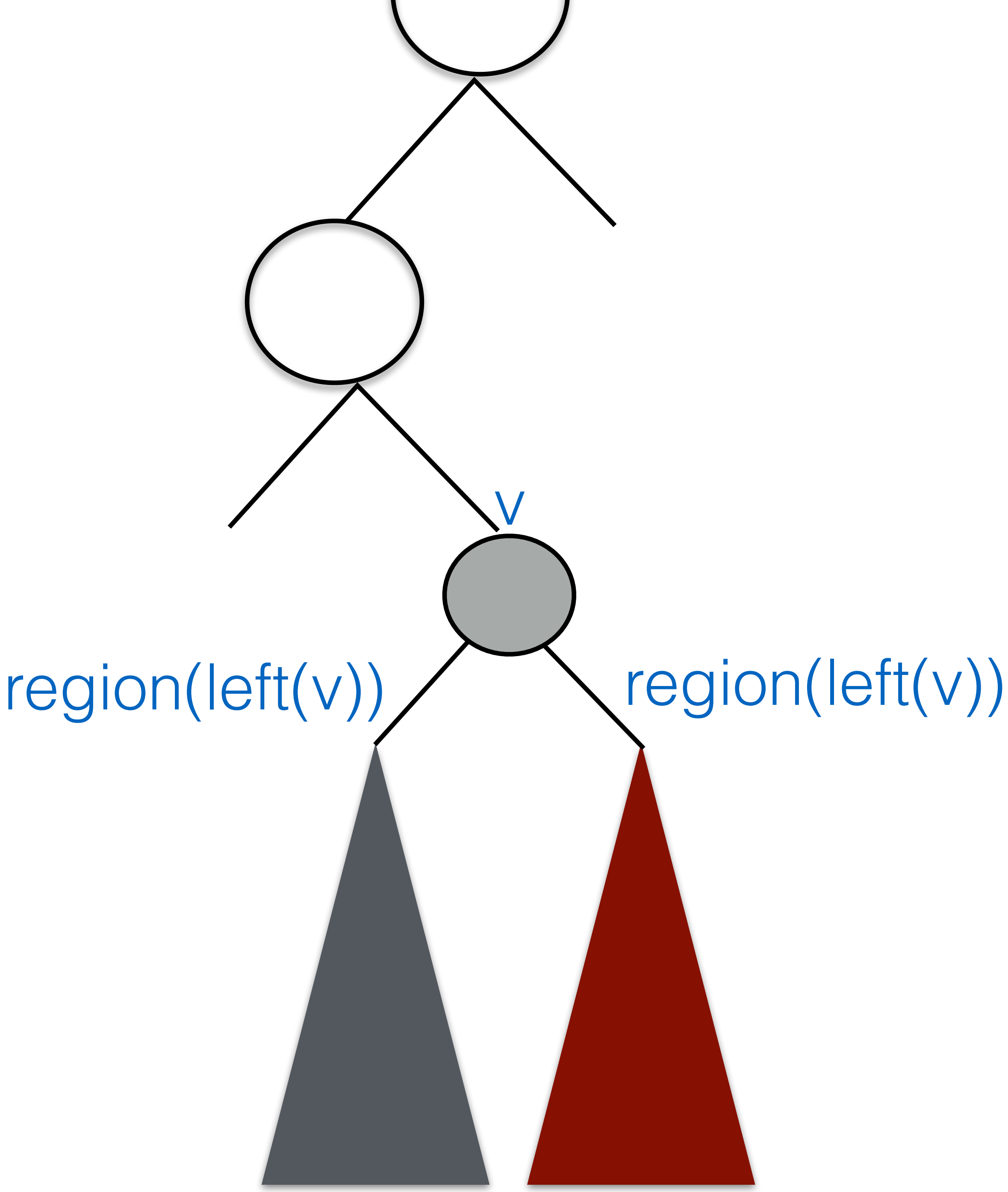


Range queries: general idea

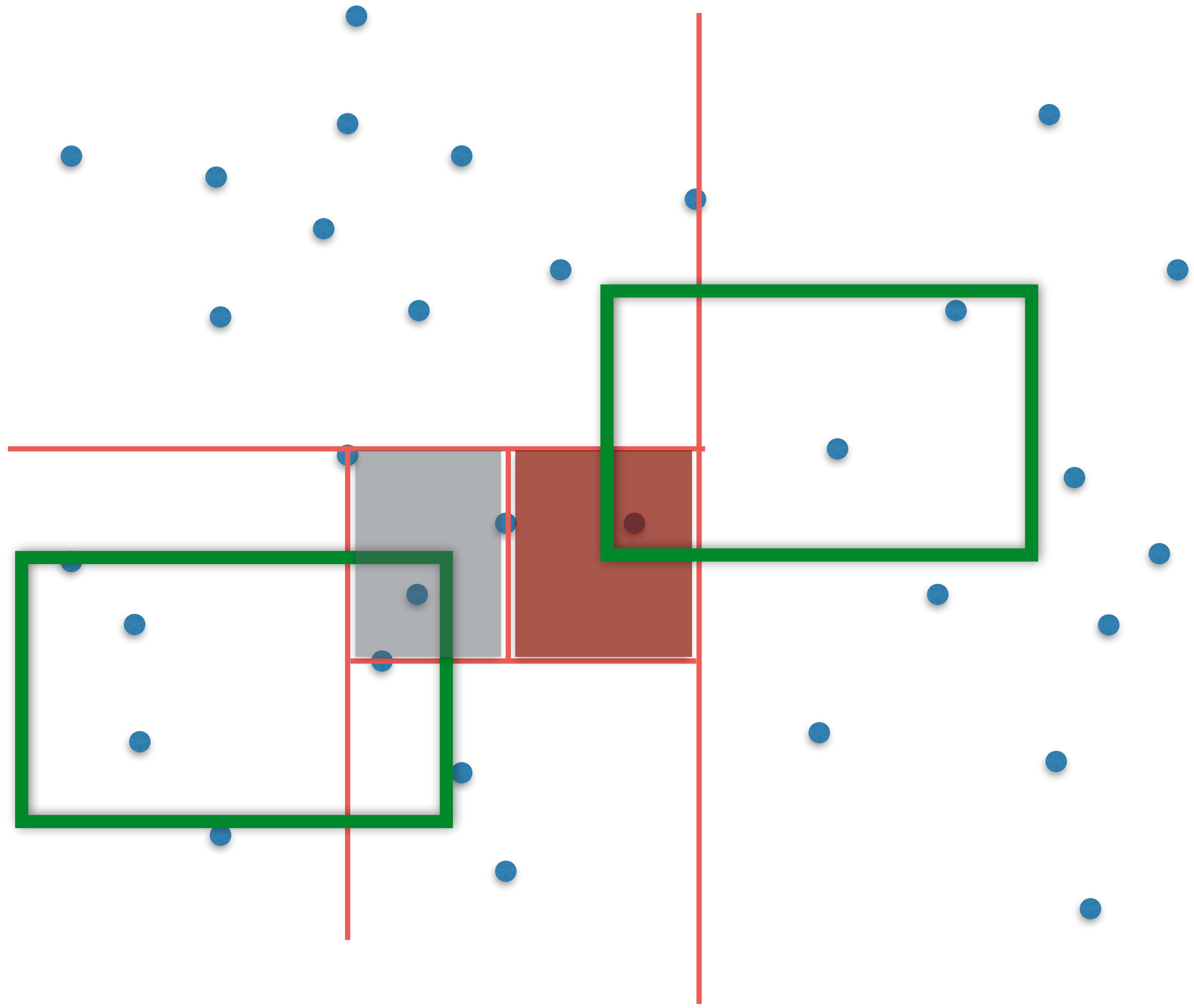


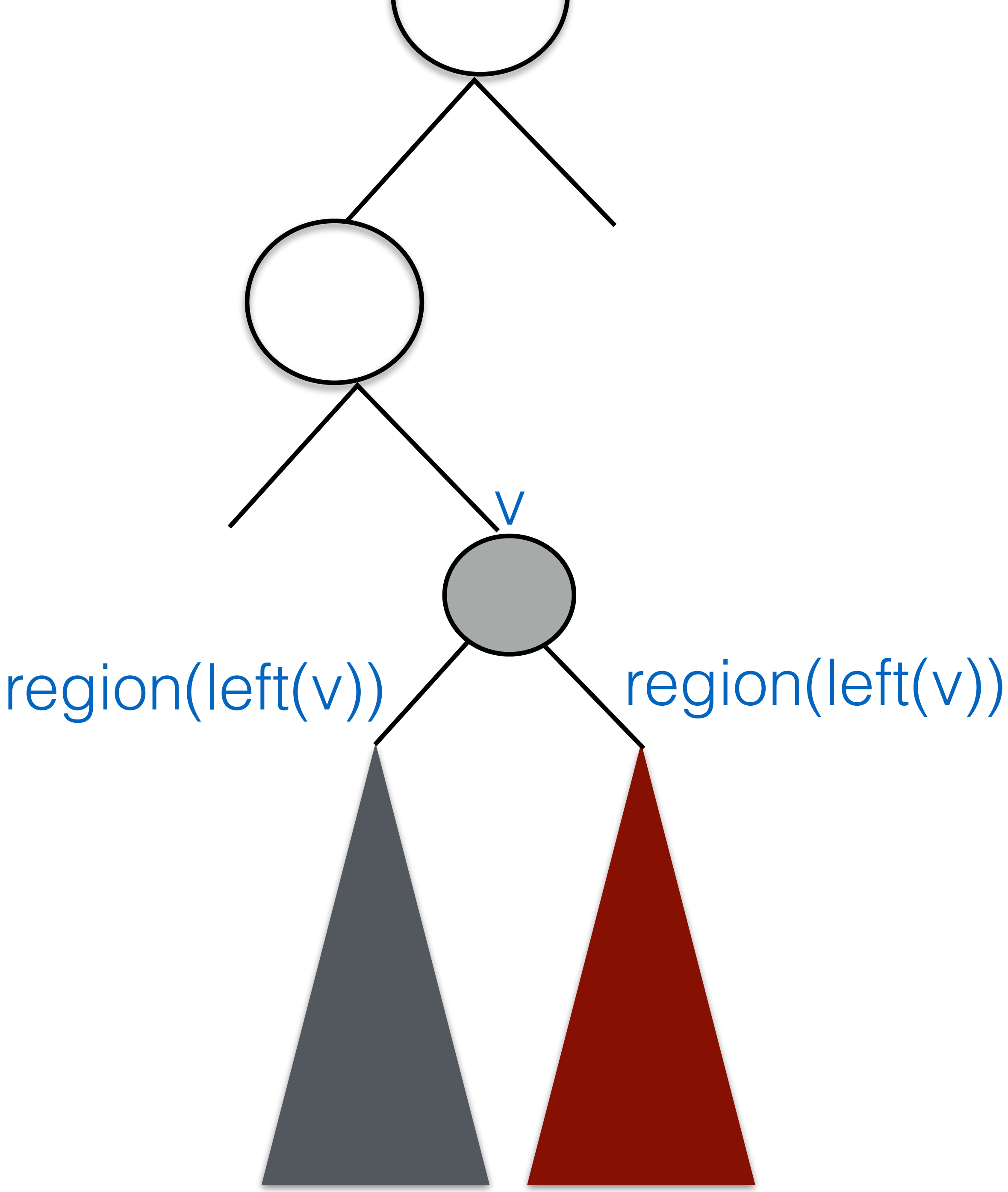
Range queries: general idea



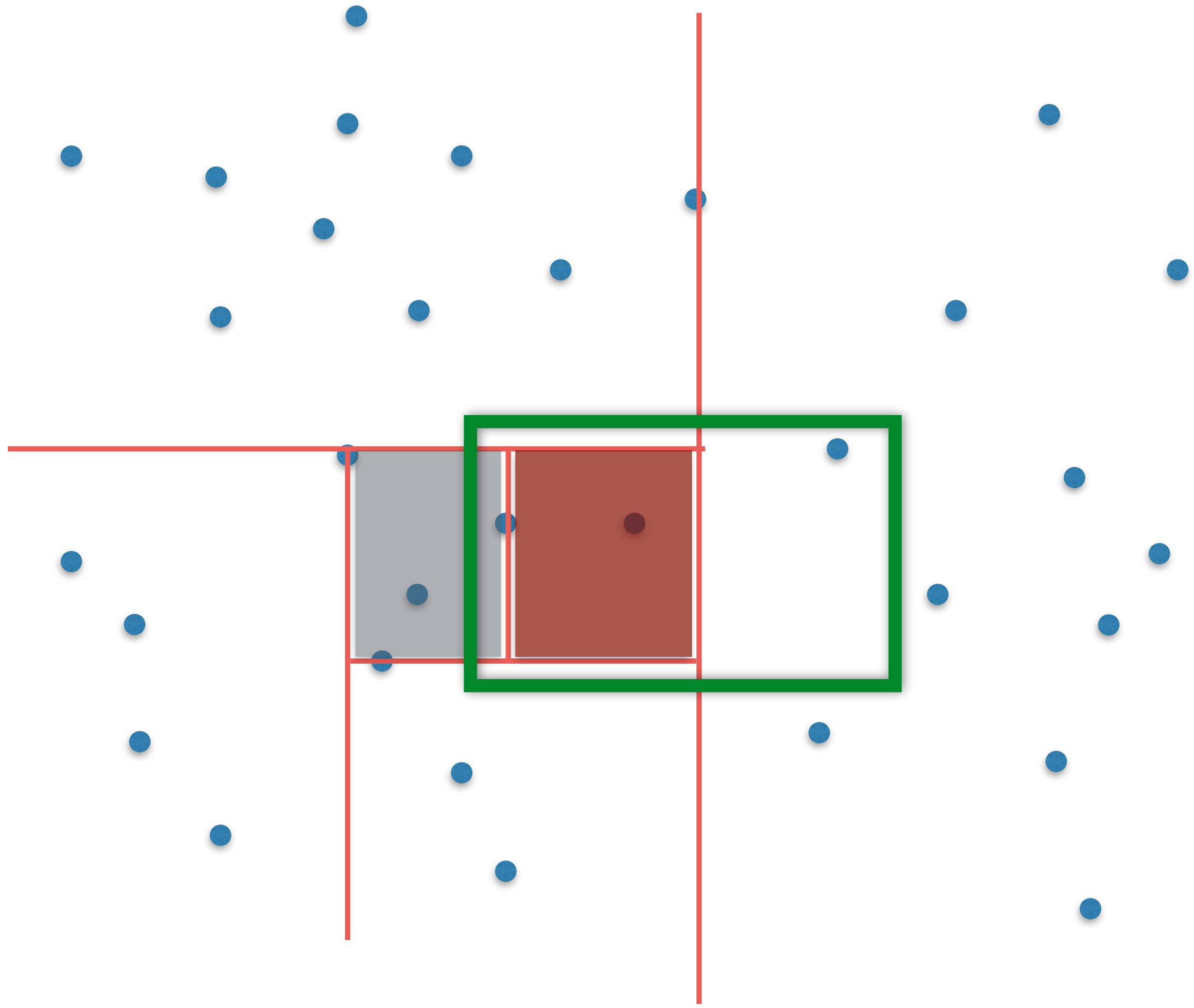


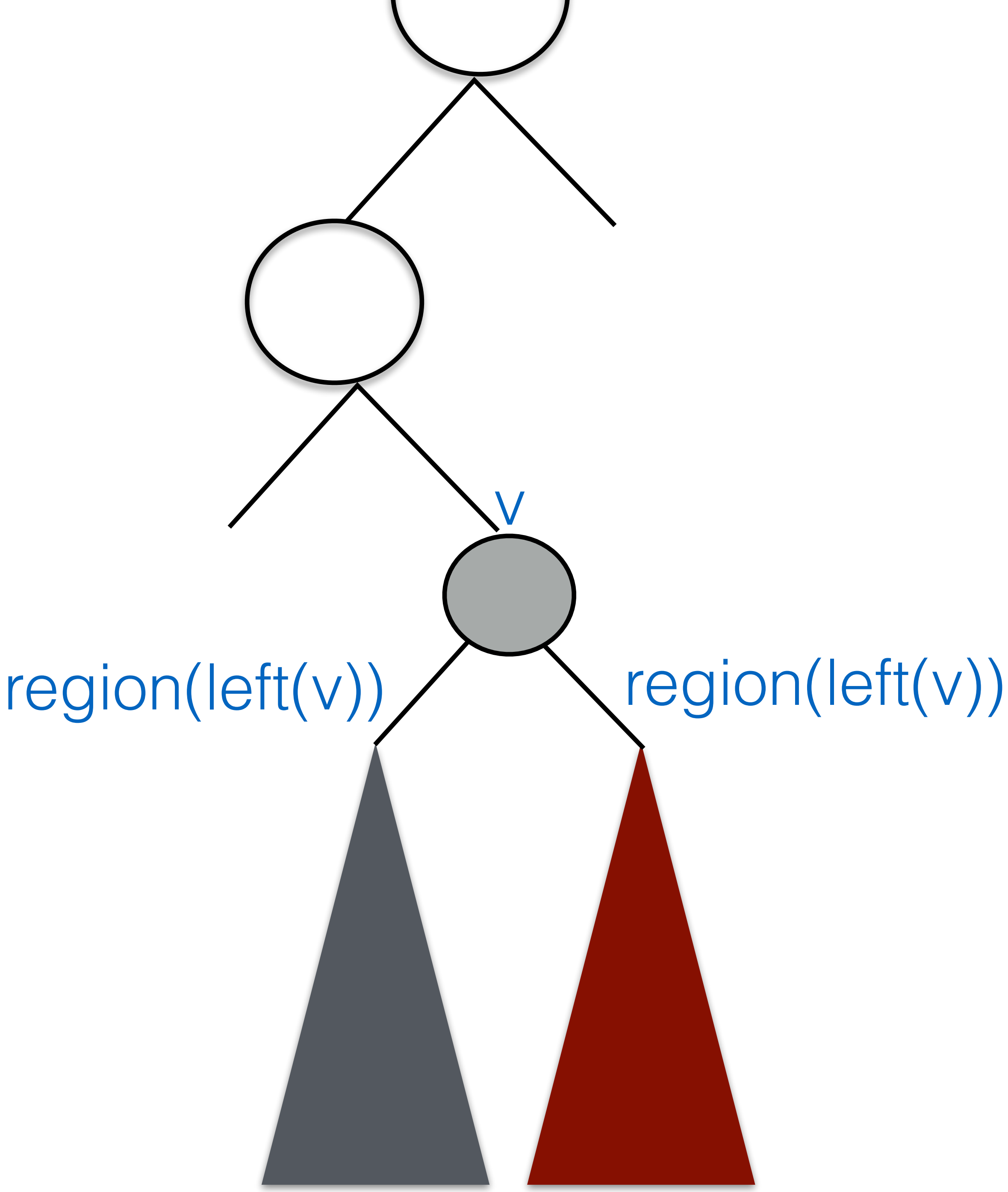
Range queries: general idea





Range queries: general idea





Algorithm SEARCHKDTREE(ν, R)

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

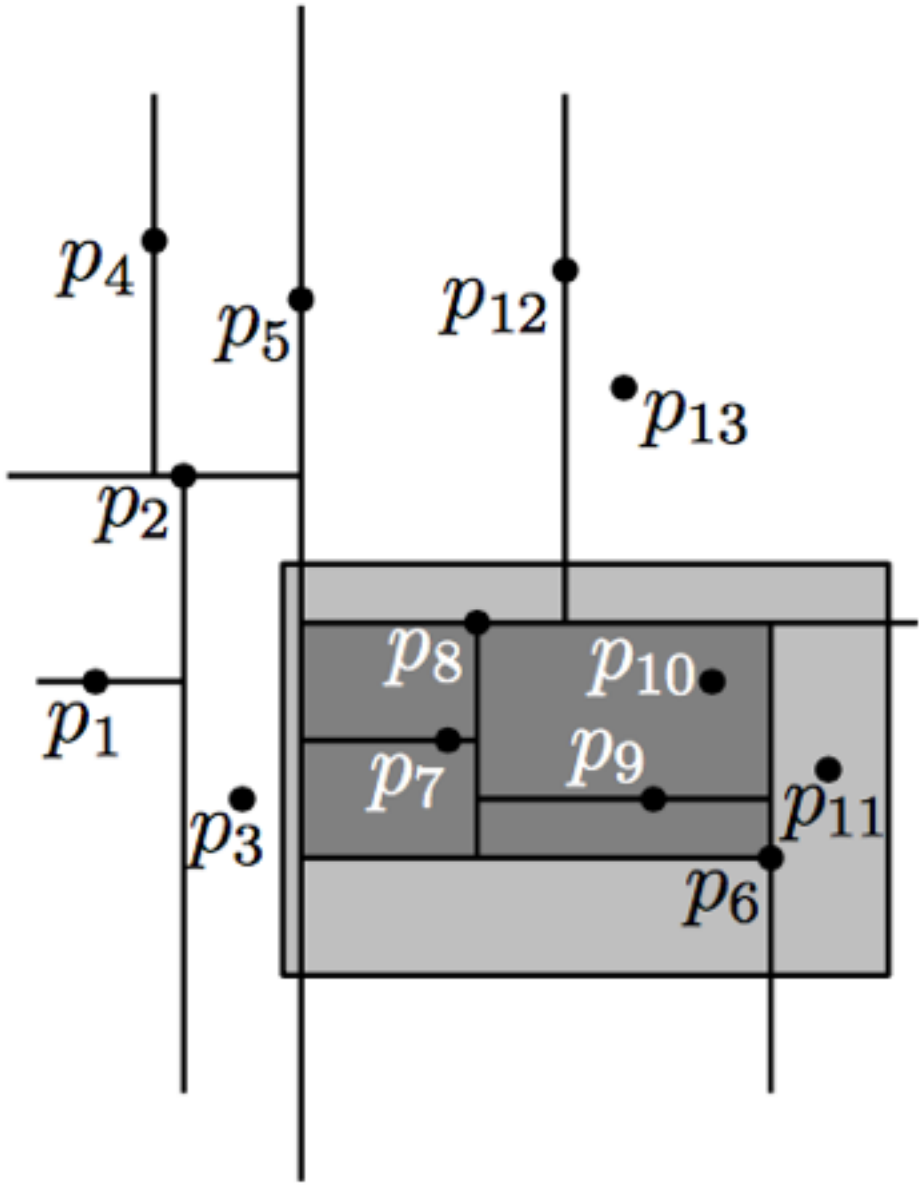
Output. All points at leaves below ν that lie in the range.

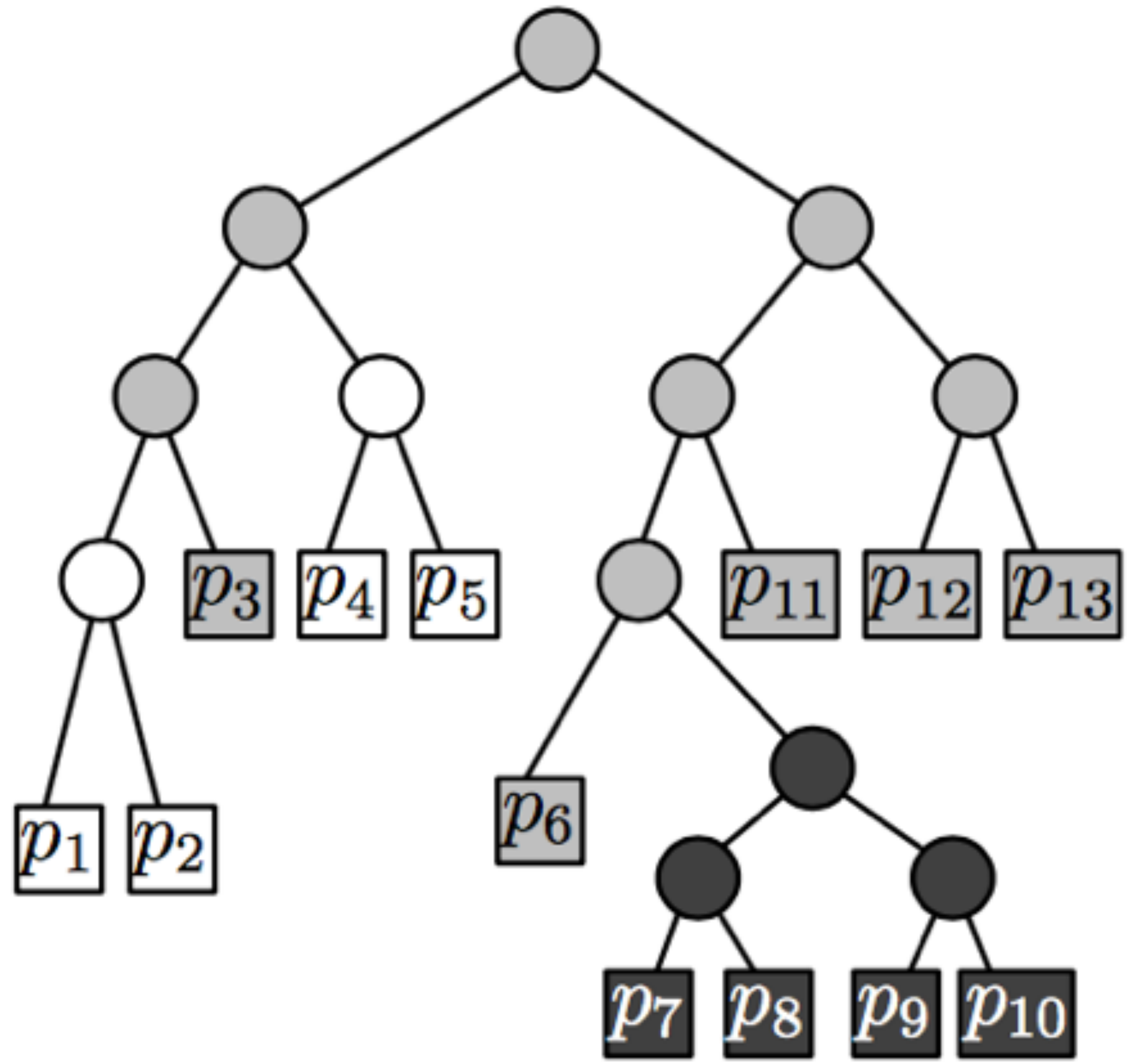
```
if ν is a leaf
    then Report the point stored at ν if it lies in R
    else if region(lc(ν)) is fully contained in R
            then REPORTSUBTREE(lc(ν))
            else if region(lc(ν)) intersects R
                    then SEARCHKDTREE(lc(ν), R)
         if region(rc(ν)) is fully contained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

To analyze RangeSearch(), we look at the nodes visited in the kd-tree

- **White nodes:** nodes never visited by the query
- **Grey nodes:** visited by the query, but unclear if they lead to output
- **Black nodes:** visited by the query, whole subtree is output

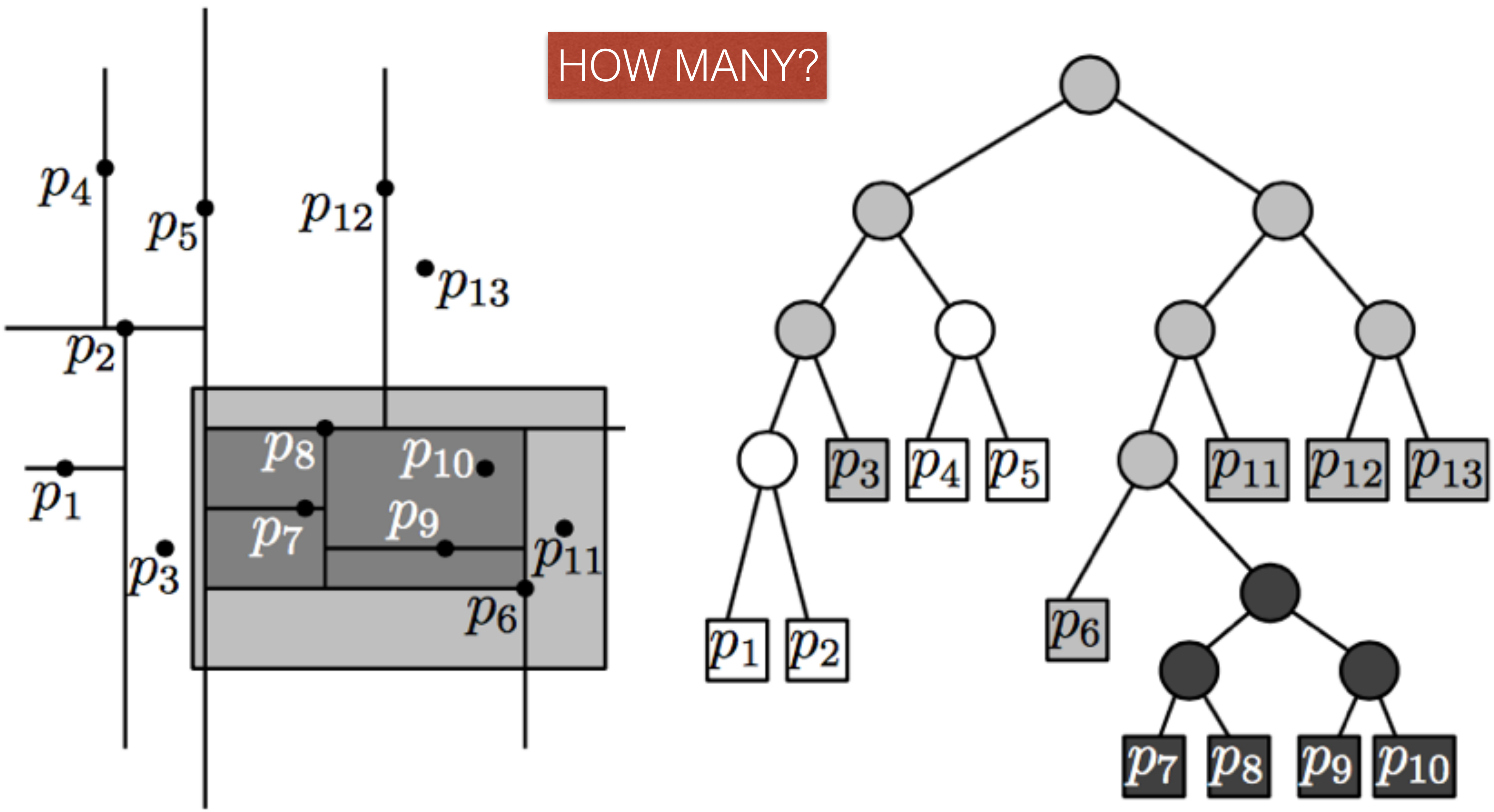




screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

Claim:

- Run time = O(number of black and grey nodes)

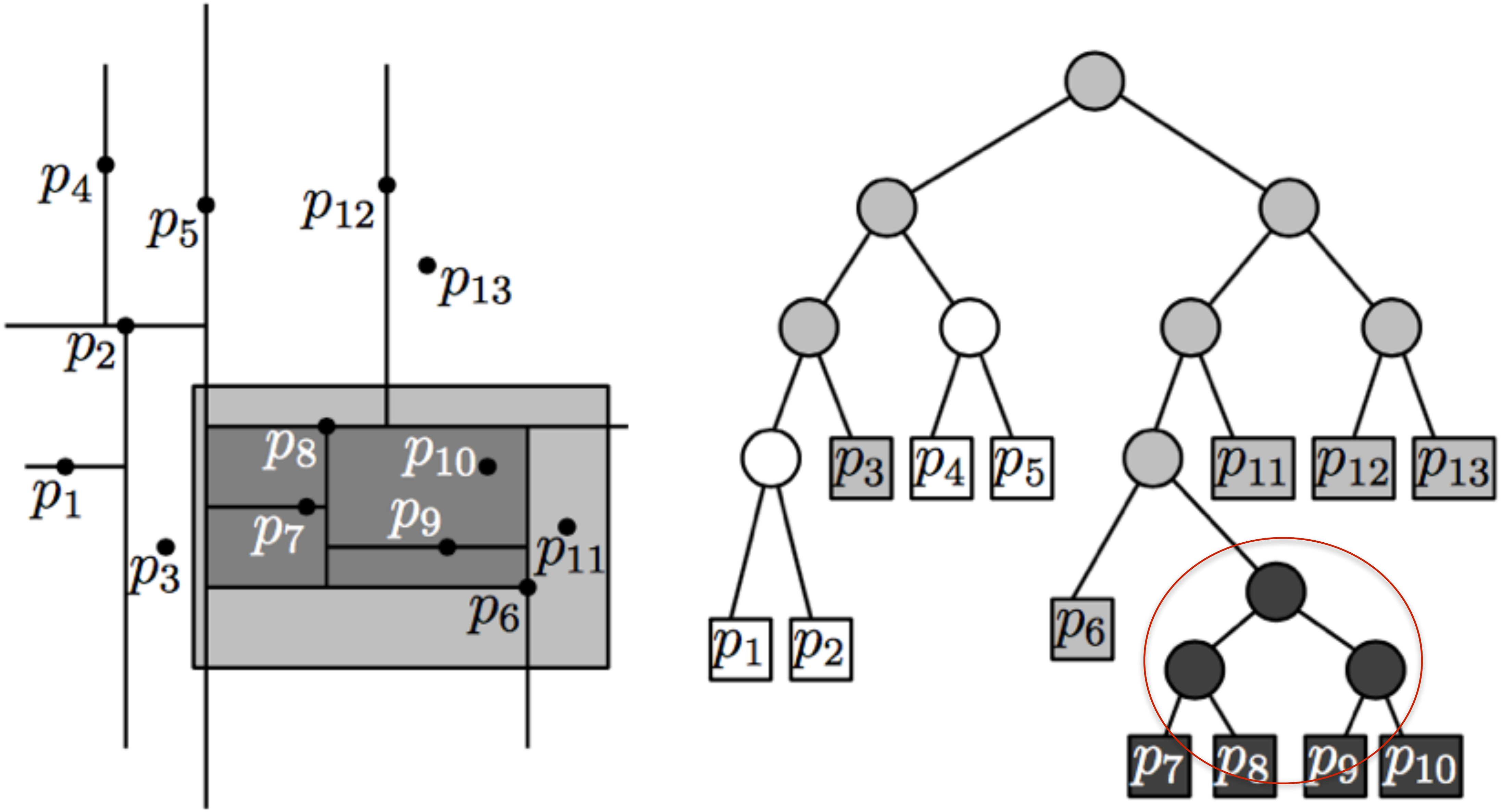


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

How many black nodes?

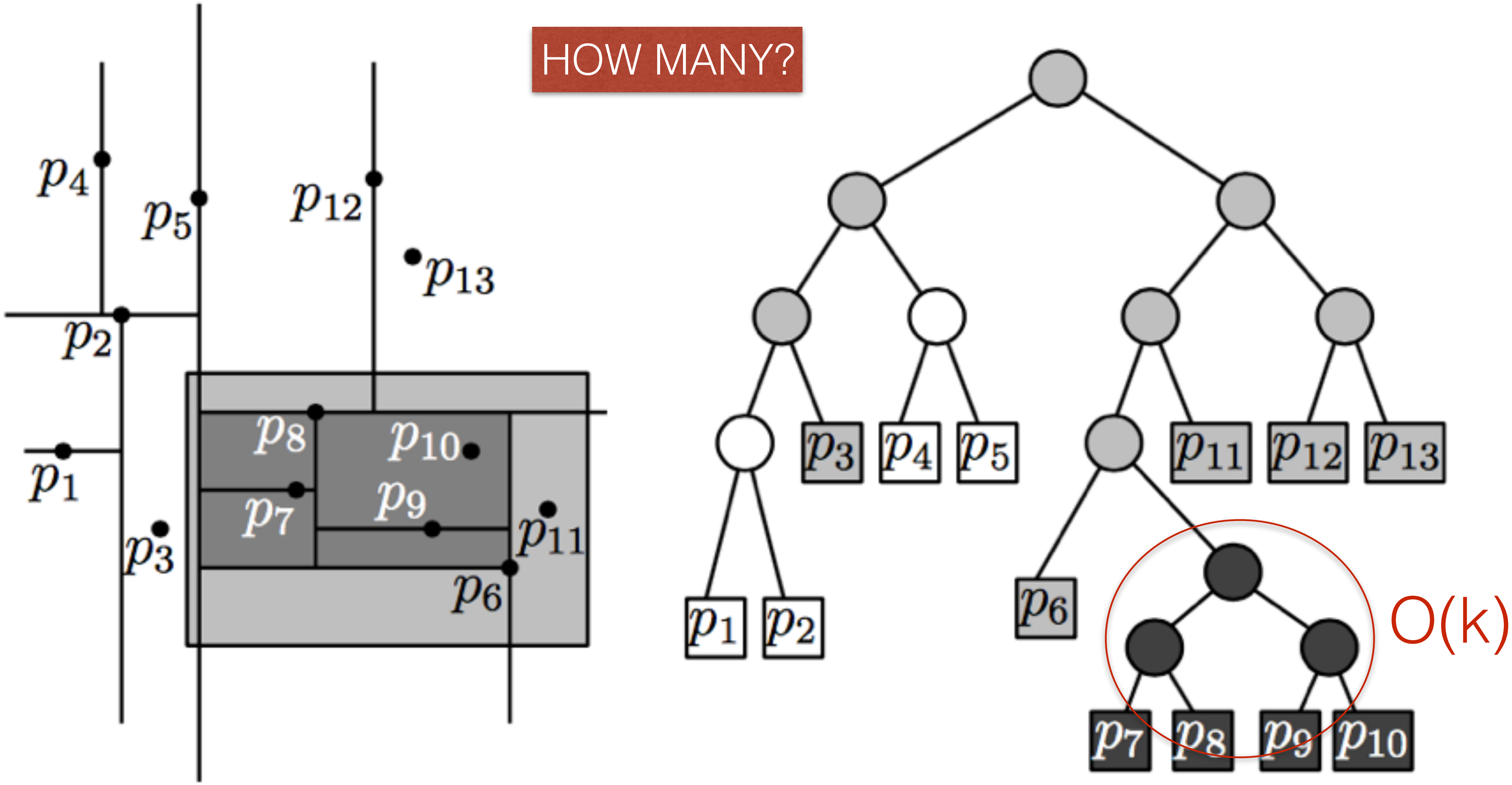
Claim:

- Each black leaf contain a point that's reported
- Number of black nodes is O(k)



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

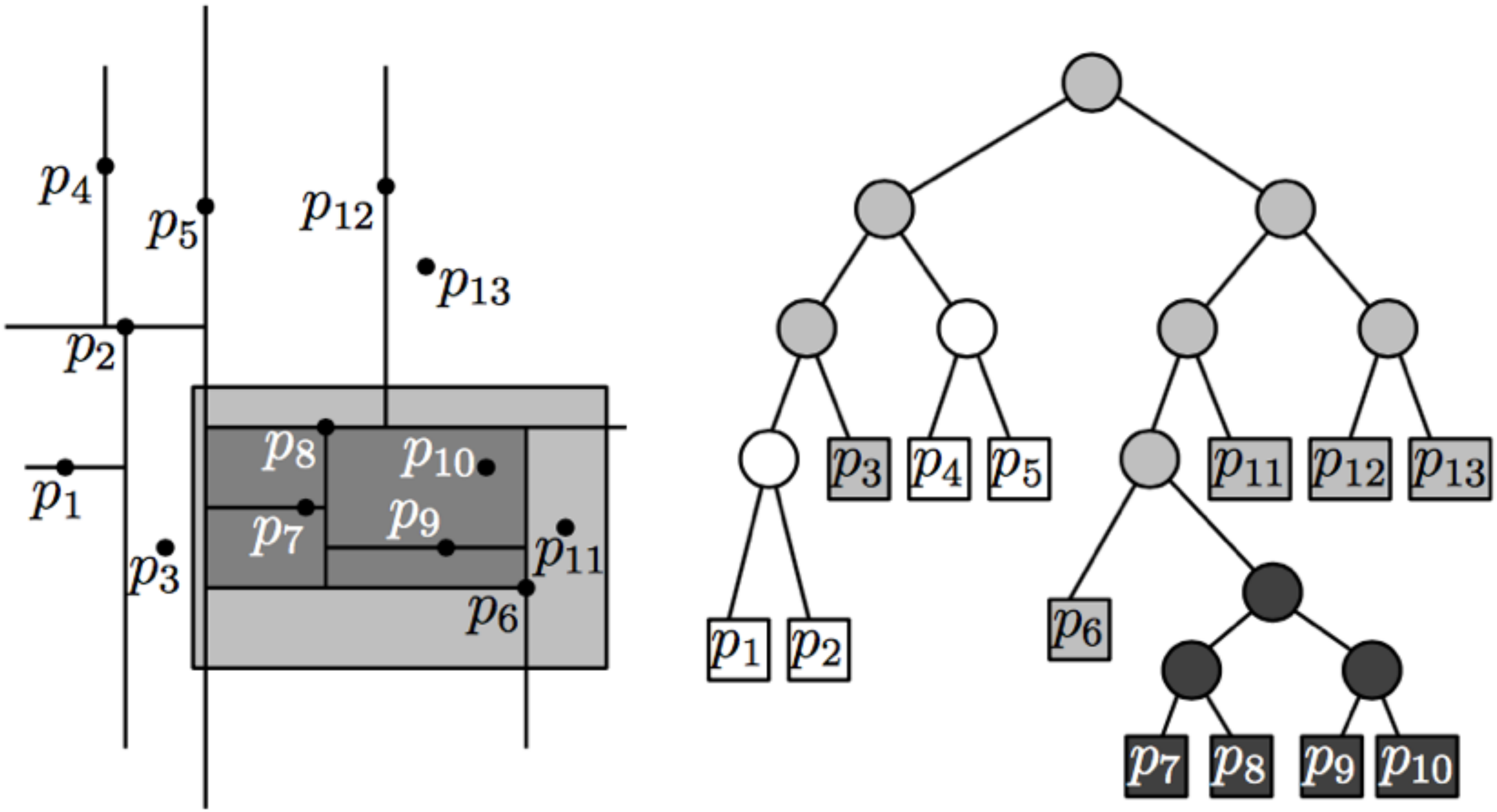
Claim: run time = O(k) + O(grey nodes)



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

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

What does it mean in terms of region(v) intersecting the range R?



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

2D Range searching: Analysis

- **White nodes:** nodes never visited by the query

  R does not intersect region(v)

- **Grey nodes:** visited by the query, but unclear if they lead to output

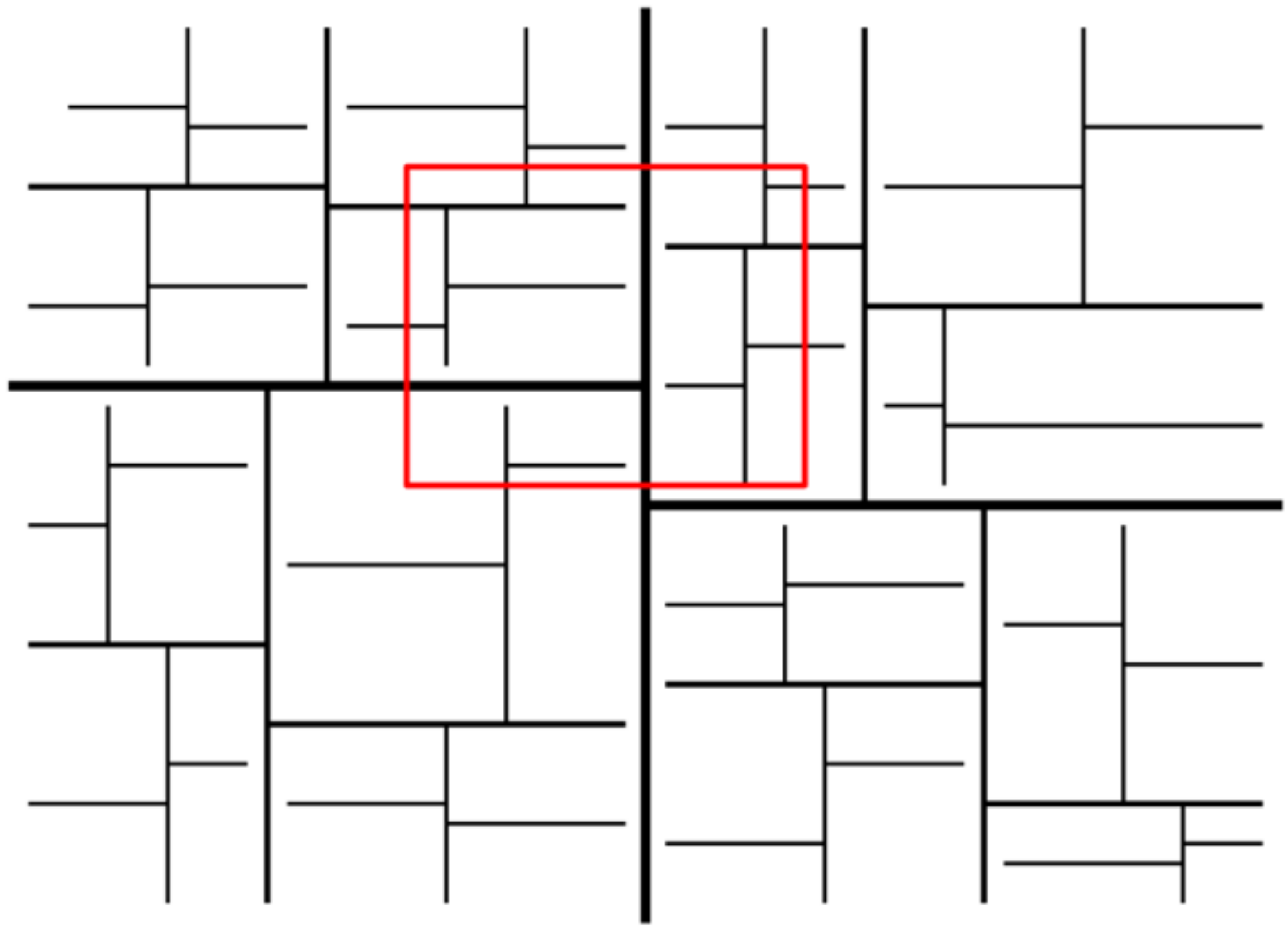
  R intersects region(v), but region(v) not contained in R

- **Black nodes:** visited by the query, whole subtree is output

  region(v) is contained in R

Claim: The region of a gray node intersects the boundary of R

How many grey nodes?

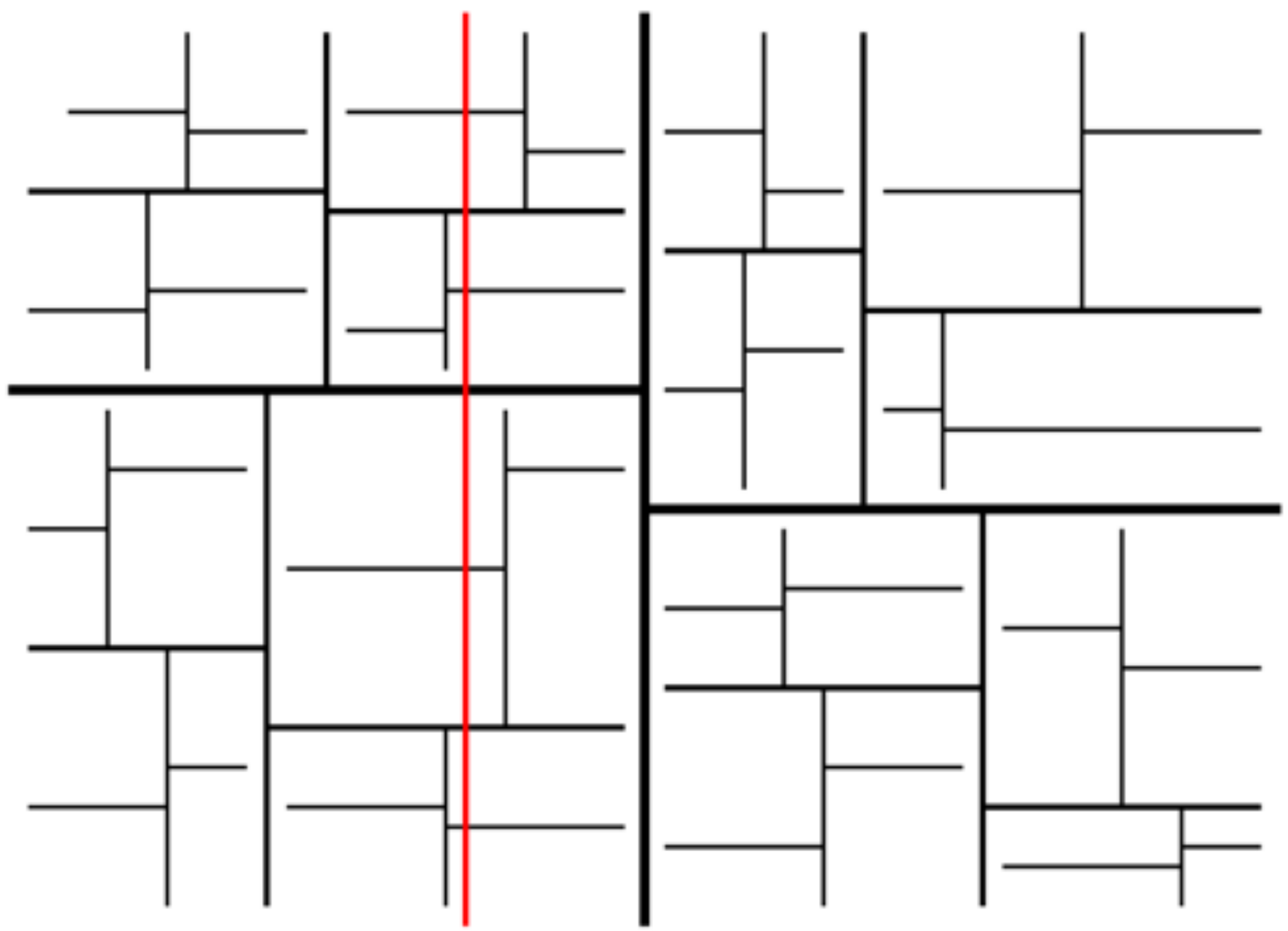


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

How many nodes in a kd-tree are such that the boundary of their region intersects the boundary of the range?

Simplified problem:

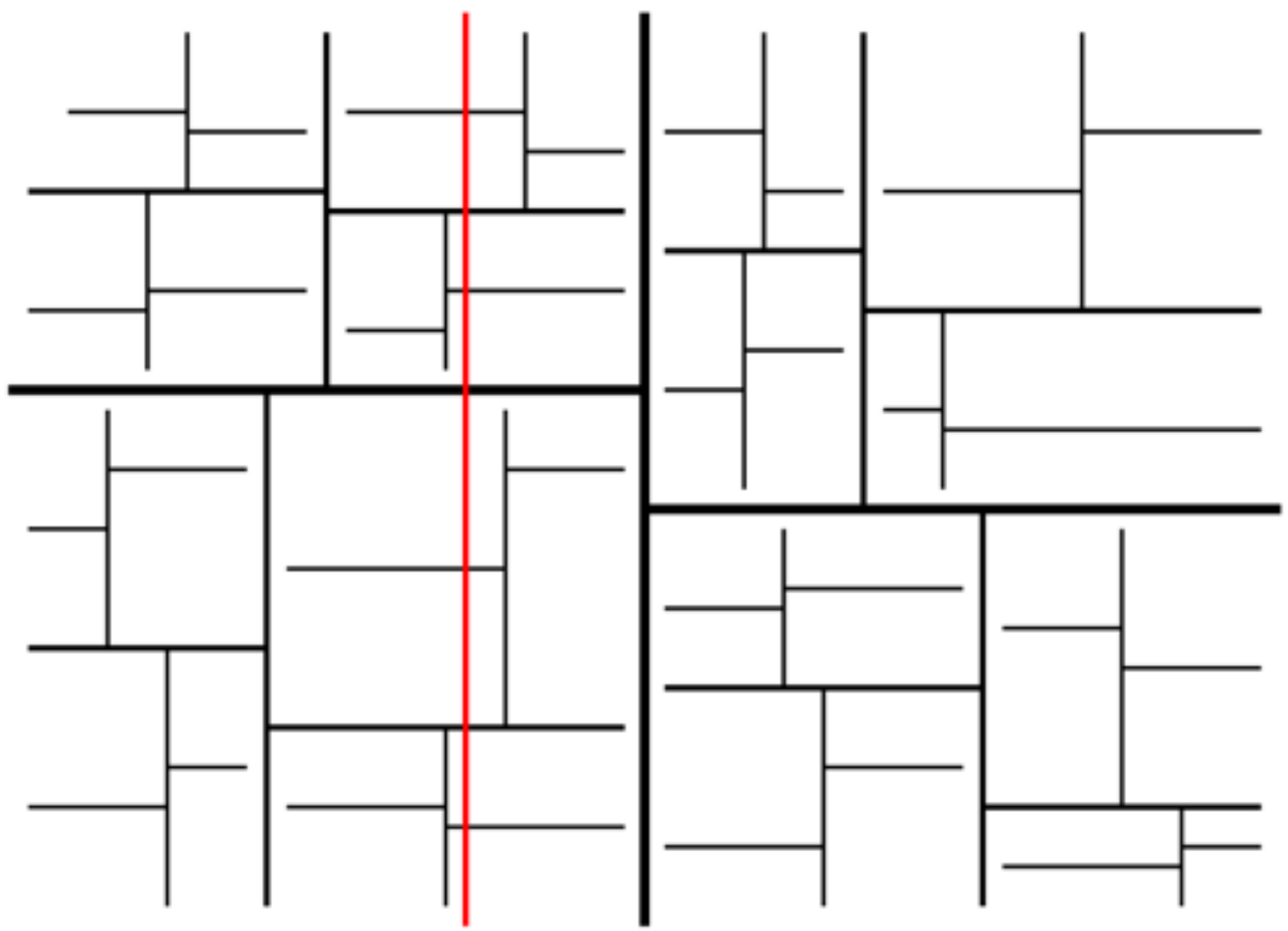
We'll try to count the number of grey nodes whose region intersects a vertical line l.



Simplified problem:

We’ll try to count the number of grey nodes whose region intersects a vertical line l.

We’ll think recursively, starting at the root:

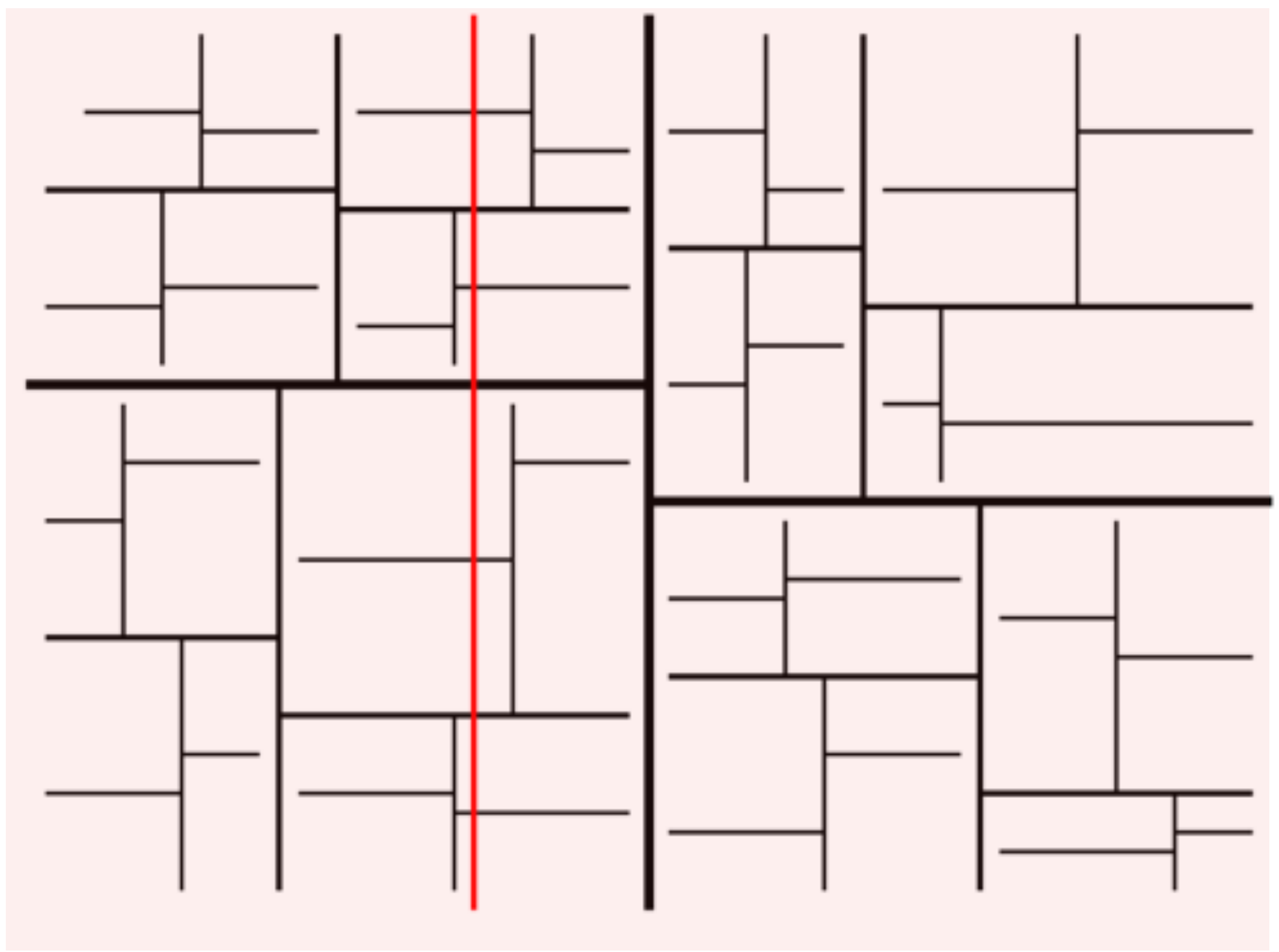


Simplified problem:

We'll try to count the number of grey nodes whose region intersects a vertical line l.

We'll think recursively, starting at the root:

- depth=0: region(root) intersects l

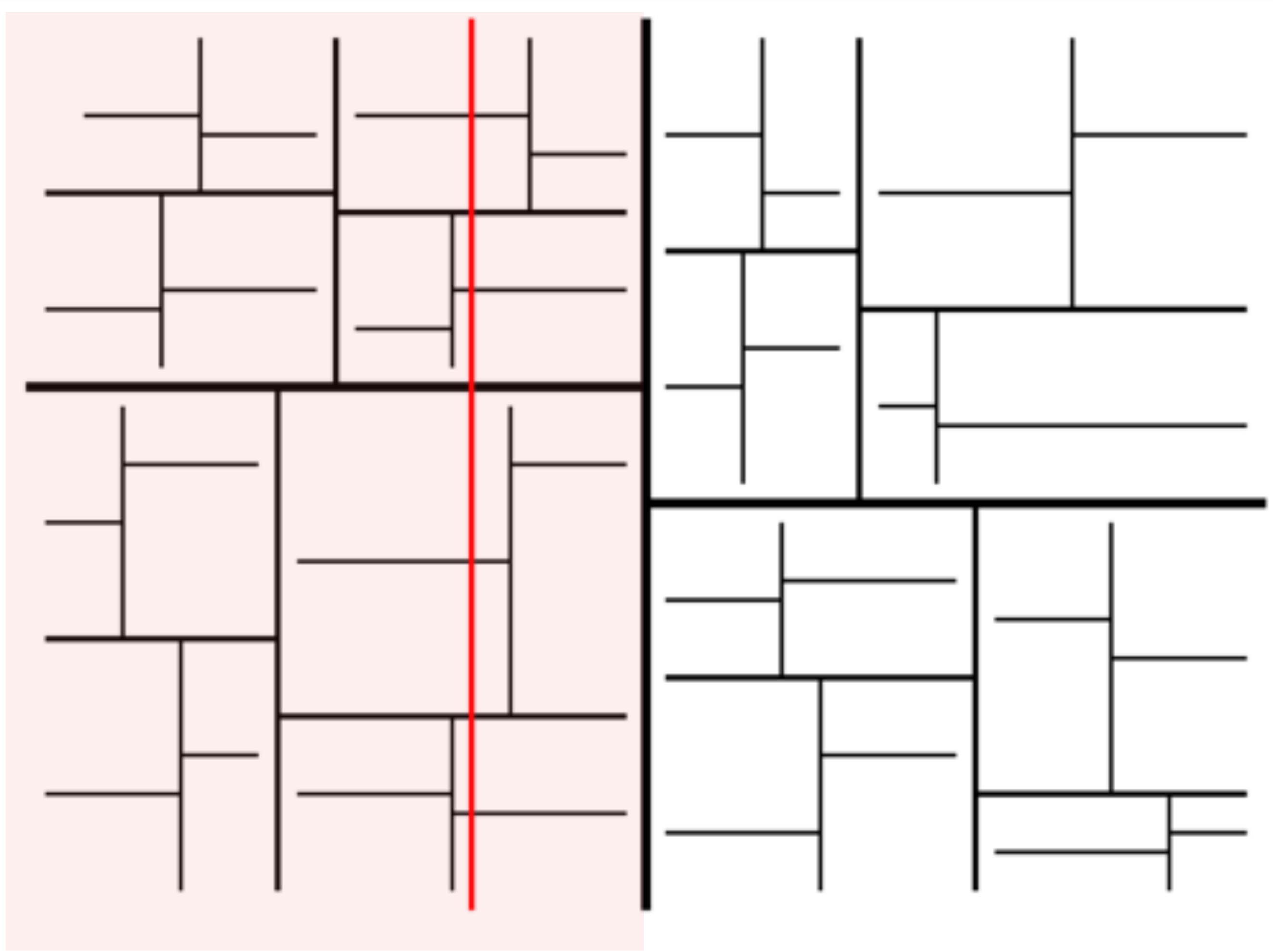


Simplified problem:

We'll try to count the number of grey nodes whose region intersects a vertical line l.

We'll think recursively, starting at the root:

- depth=1: only one of left and right child of root intersects l

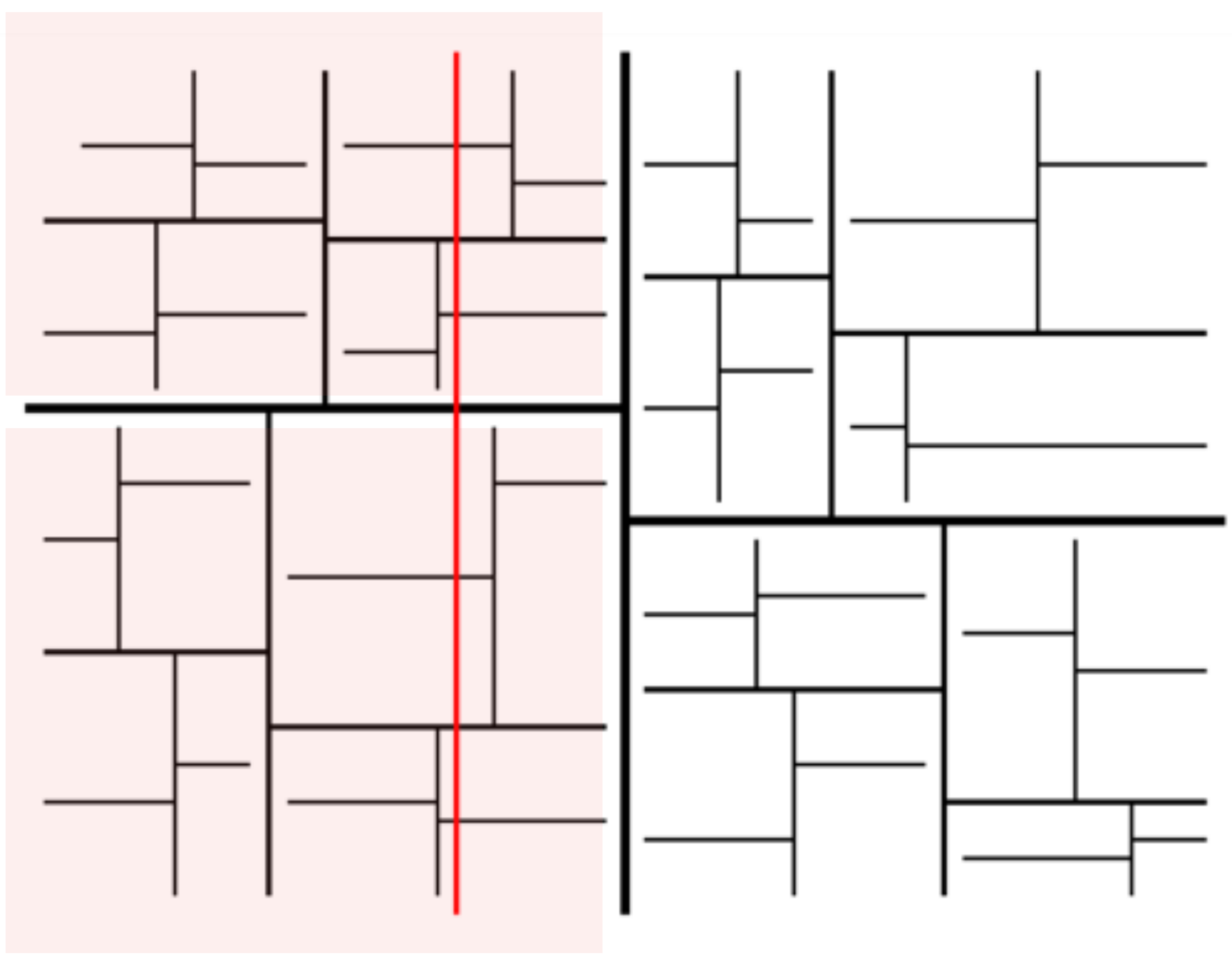


Simplified problem:

We'll try to count the number of grey nodes whose region intersects a vertical line l.

We'll think recursively, starting at the root:

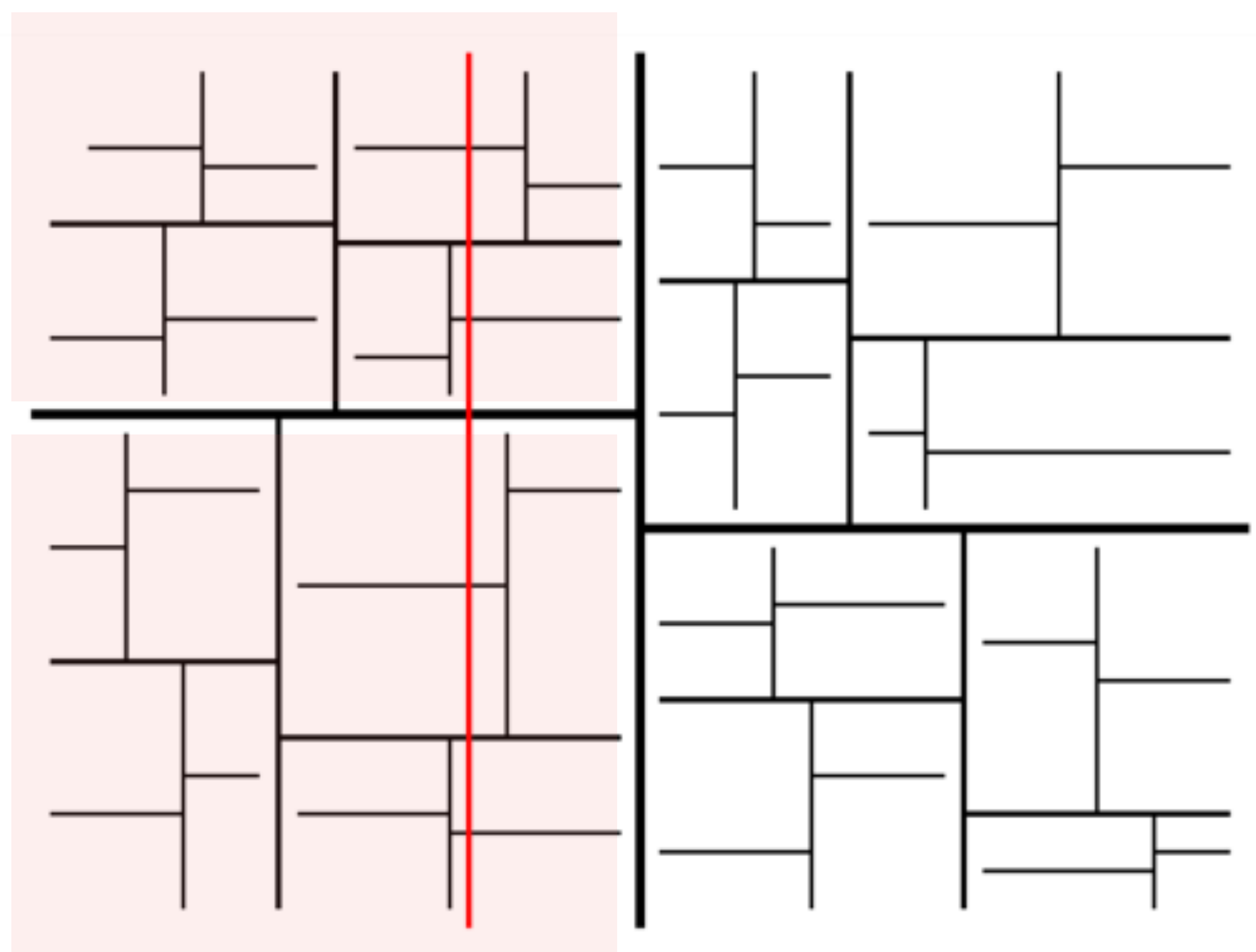
- depth=2: both left and right child intersect l, and we can recurse



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

Proof:

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

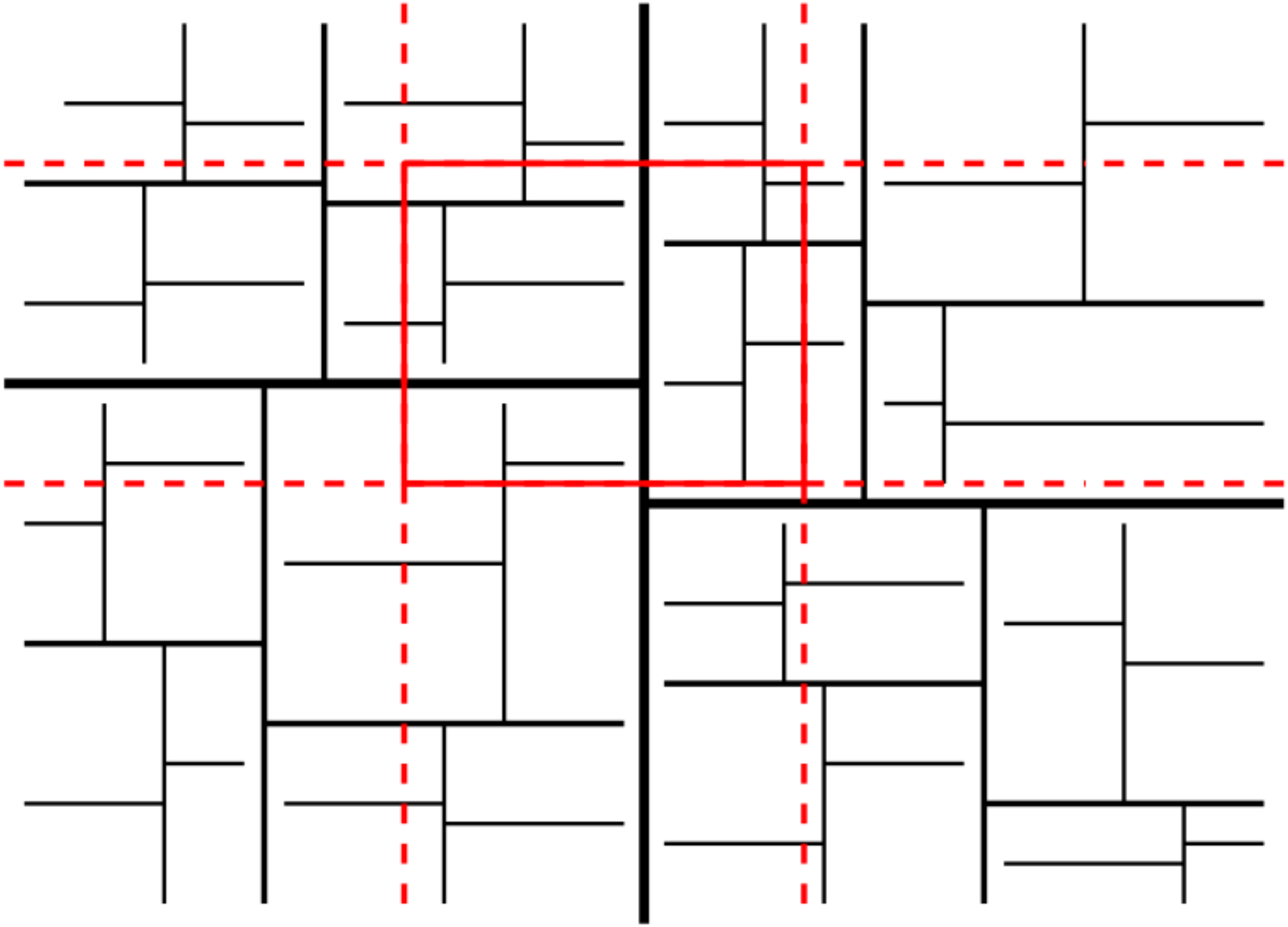


The number of grey nodes if the query were a vertical line is $O(\sqrt{n})$

The same is true if the query were a horizontal line

How about a query rectangle?

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



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

The number of grey nodes for a query rectangle is at most the number of grey nodes for two vertical and two horizontal lines, so it is at most $4 \cdot O(\sqrt{n}) = O(\sqrt{n})$!

For black nodes, reporting a whole subtree with k leaves, takes $O(k)$ time (there are $k-1$ internal black nodes)

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 $O(n \log n)$ 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 number of answers reported

For range counting queries, we need $O(\sqrt{n})$ time

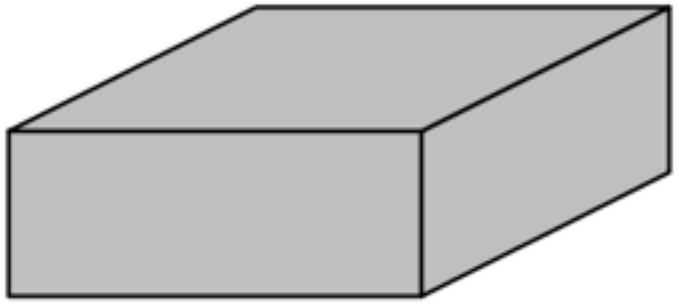
screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

n	$\log n$	$\sqrt{n}$
4	2	2
16	4	4
64	6	8
256	8	16
1024	10	32
4096	12	64
1.000.000	20	1000

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

3D: 3d-tree

3D: 3d-tree



- A 3D kd-tree alternates splits on x-, y- and z-dimensions
- A 3D range query is a cube
- The construction if a 2D kd-tree extends to 3D
- The 3D range query is 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

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