Yan-Bin Jia, Iowa State University

Kd-Trees (k-Dimensional Trees)

Outline:

I. Construction of a kd-tree

II. Nodes and regions

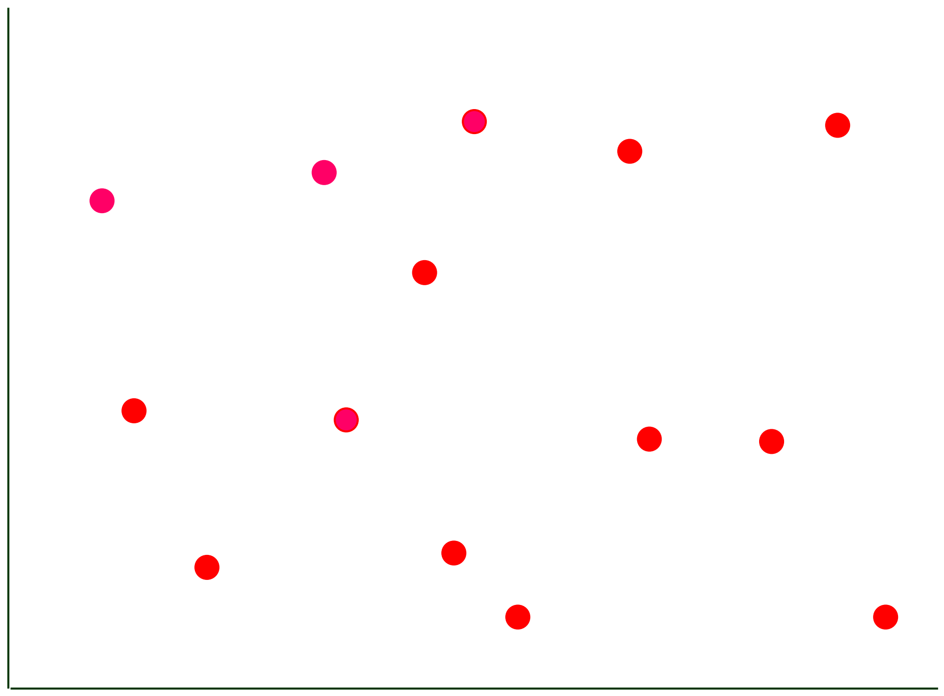
III. Query using a kd-tree

IV. Analysis of query time

Textbook: *Computational Geometry: Algorithms and Applications (*3rd ed.) by M. de Berg et al., Springer-Verlag, 2008.

I. 2D Range Search

Point set: P

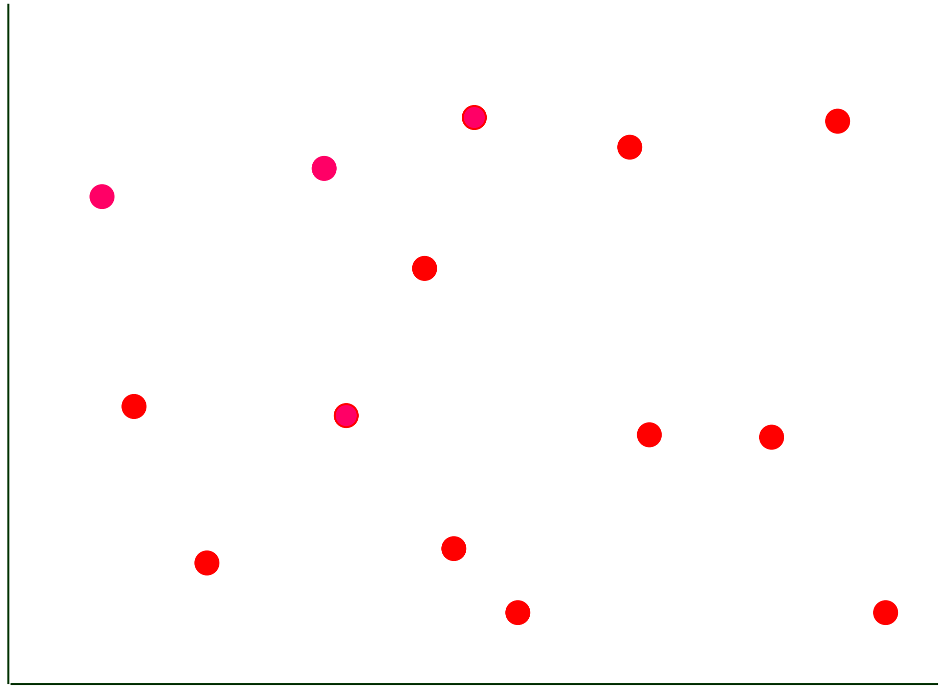


I. 2D Range Search

Point set: P

Assumptions (removable);

- No two points have the same x-coordinate.
- No two points have the same y-coordinate.

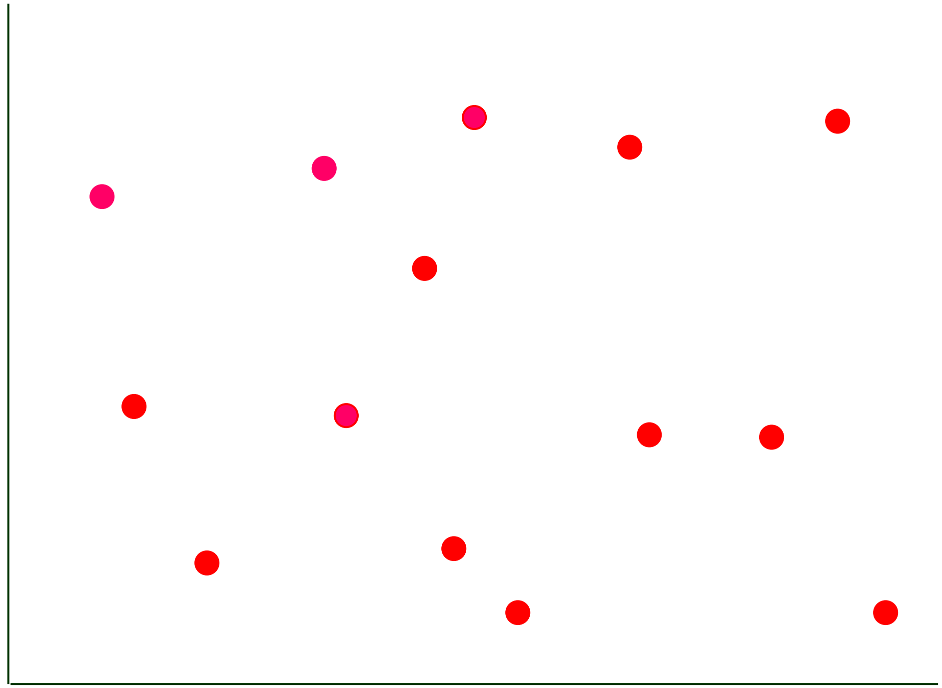


I. 2D Range Search

Point set: P

Assumptions (removable);

- No two points have the same x-coordinate.
- No two points have the same y-coordinate.



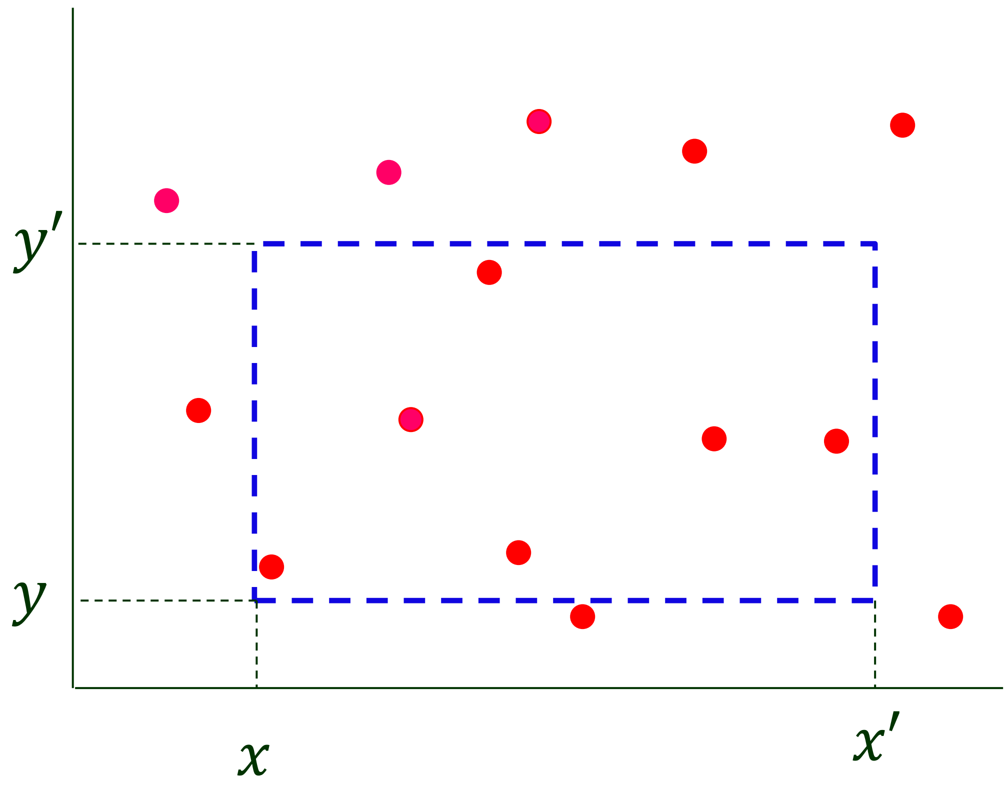
Query rectangle: $[x, x'] \times [y, y']$

I. 2D Range Search

Point set: P

Assumptions (removable);

- No two points have the same x-coordinate.
- No two points have the same y-coordinate.



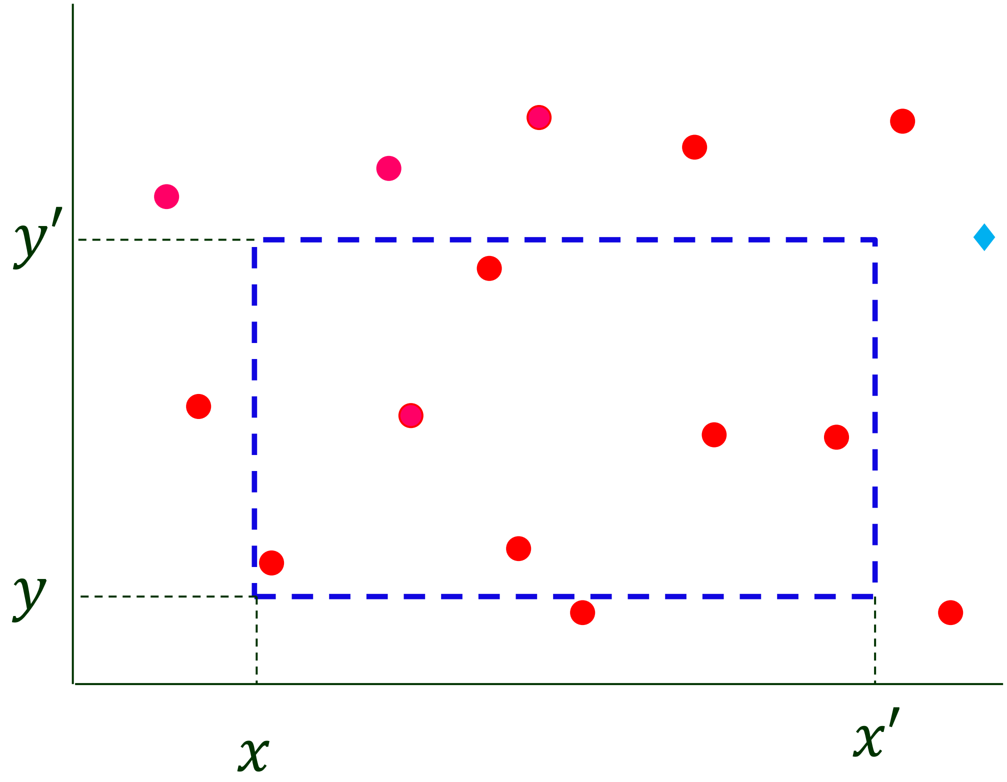
Query rectangle: $[x, x'] \times [y, y']$

I. 2D Range Search

Point set: P

Assumptions (removable);

- No two points have the same x-coordinate.
- No two points have the same y-coordinate.



♦ # answer points may be $\ll |P|$.

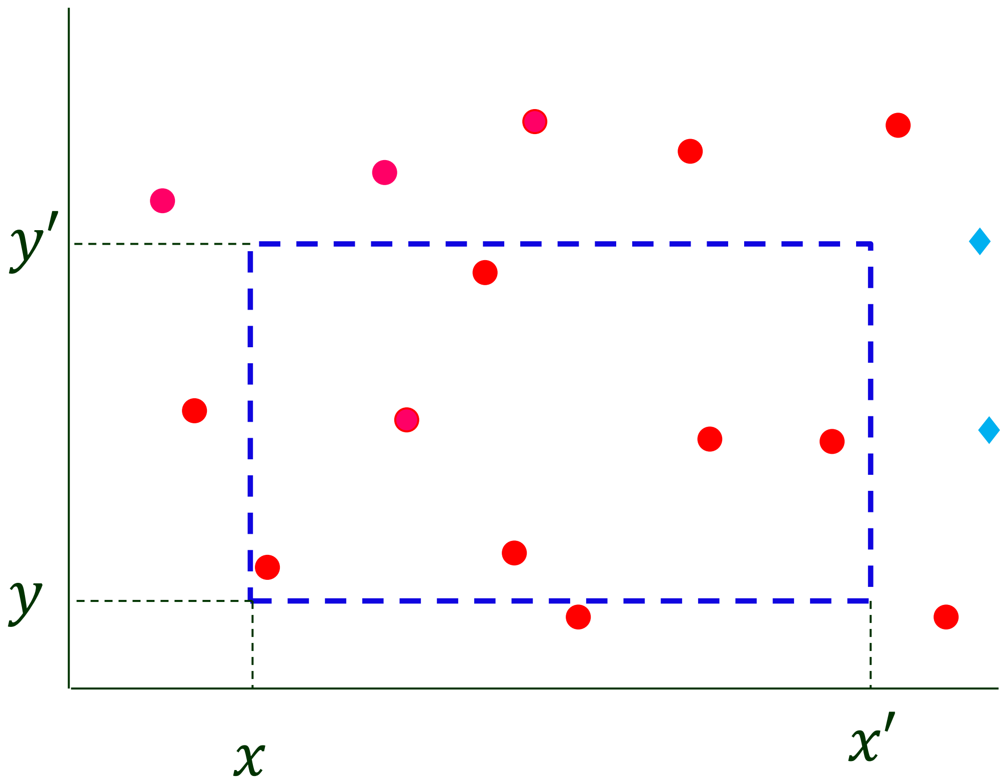
Query rectangle: $[x, x'] \times [y, y']$

I. 2D Range Search

Point set: P

Assumptions (removable);

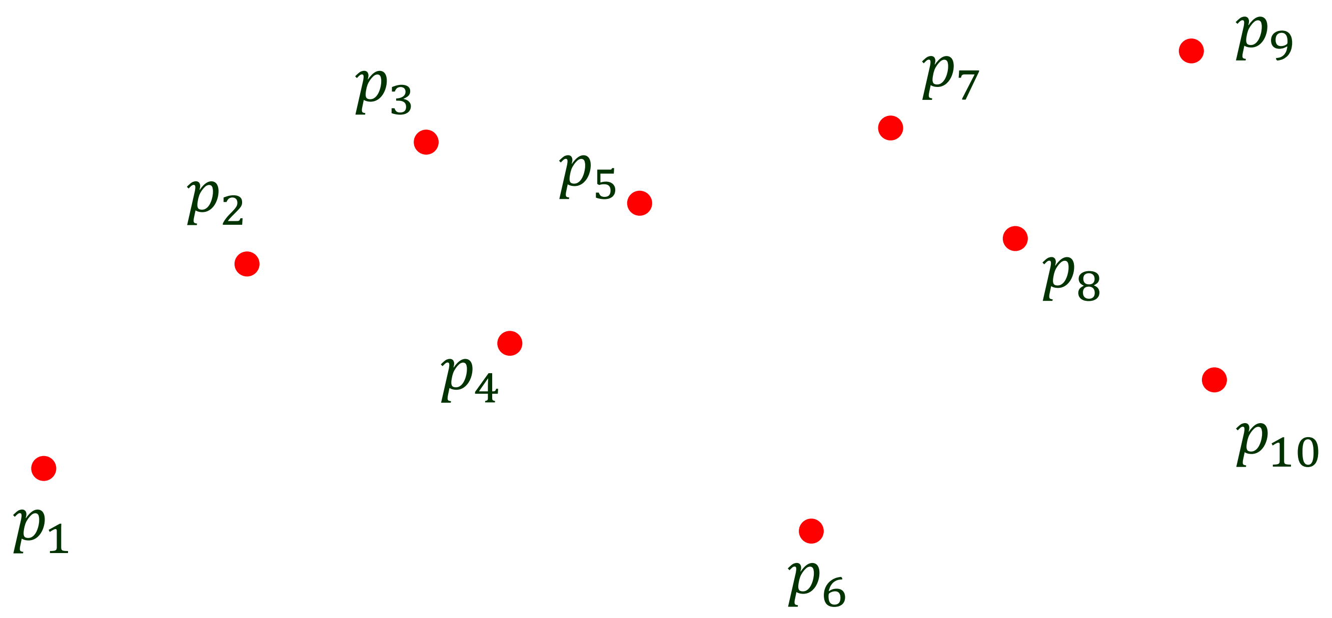
- No two points have the same x-coordinate.
- No two points have the same y-coordinate.



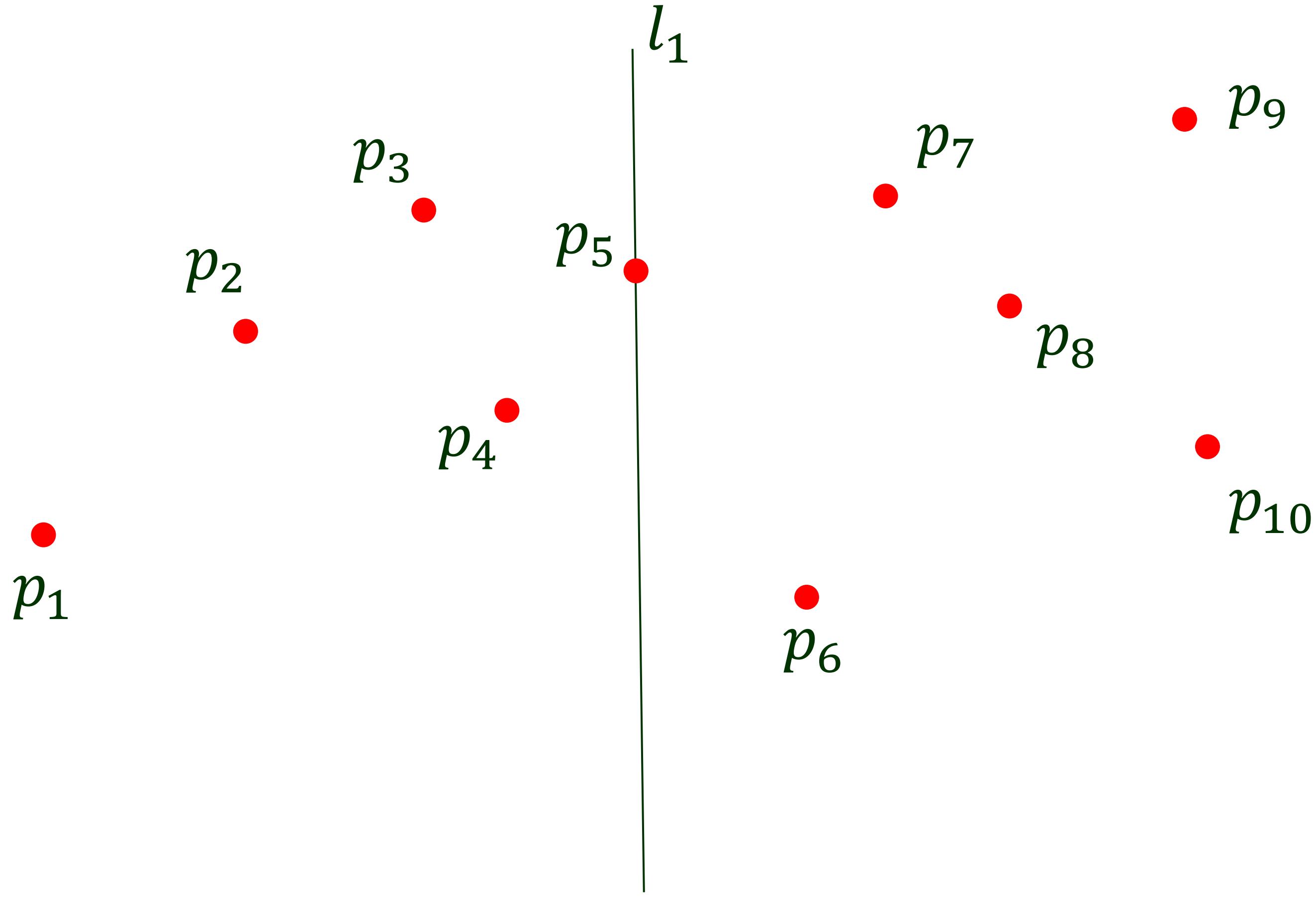
- ♦ # answer points may be $\ll |P|$.
- ♦ Many queries.

Query rectangle: $[x, x'] \times [y, y']$

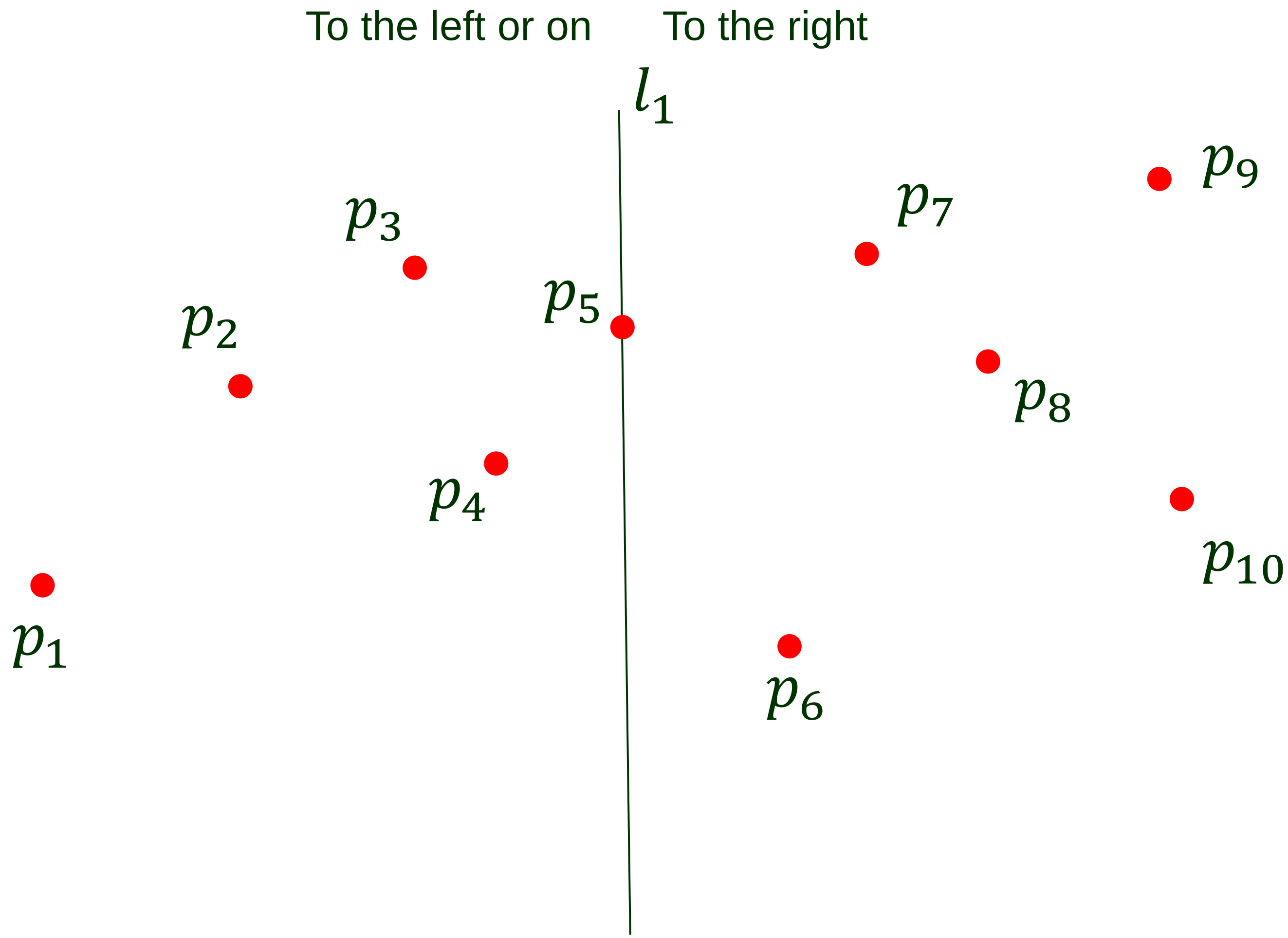
Generation of a Kd-Tree



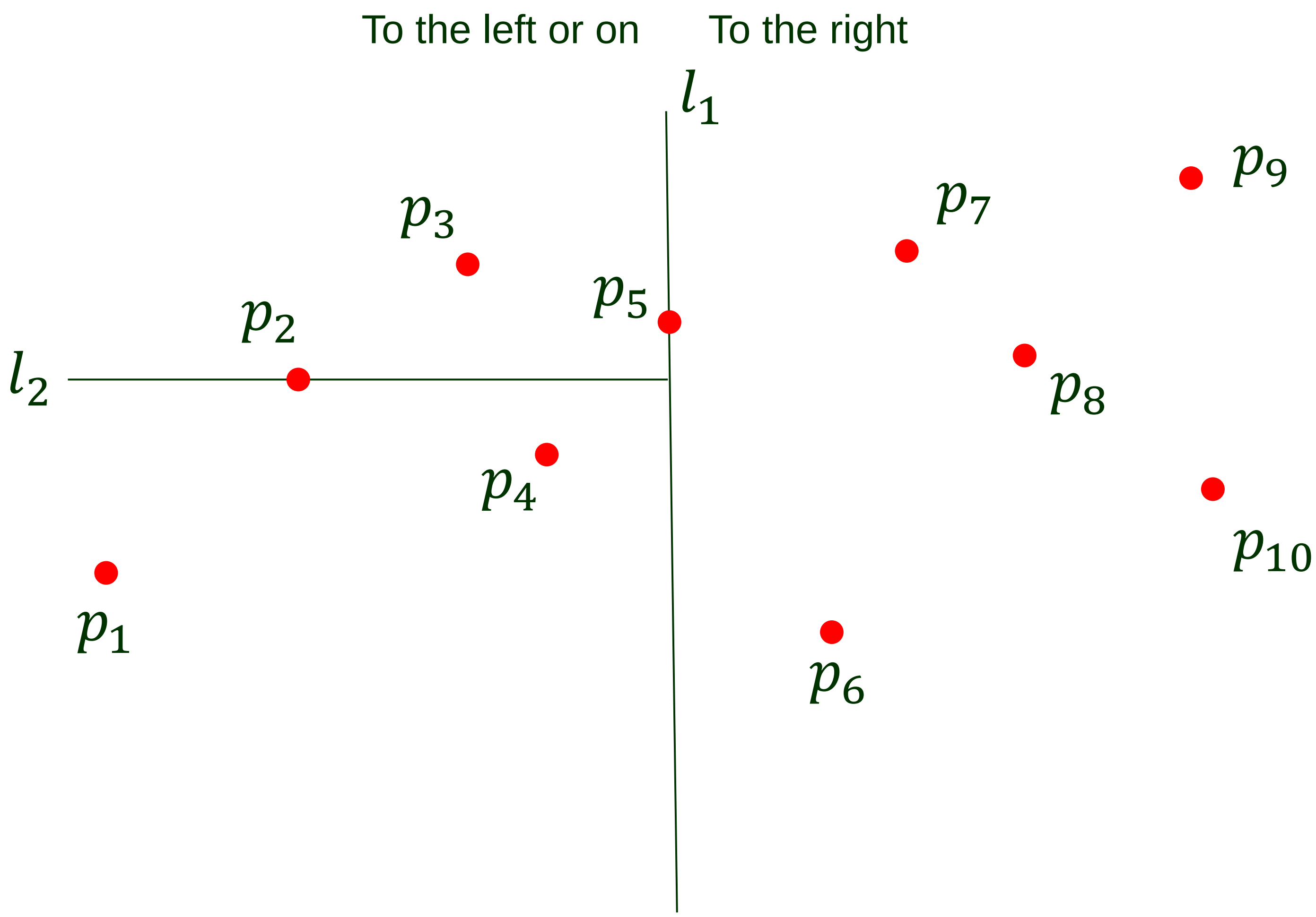
Generation of a Kd-Tree



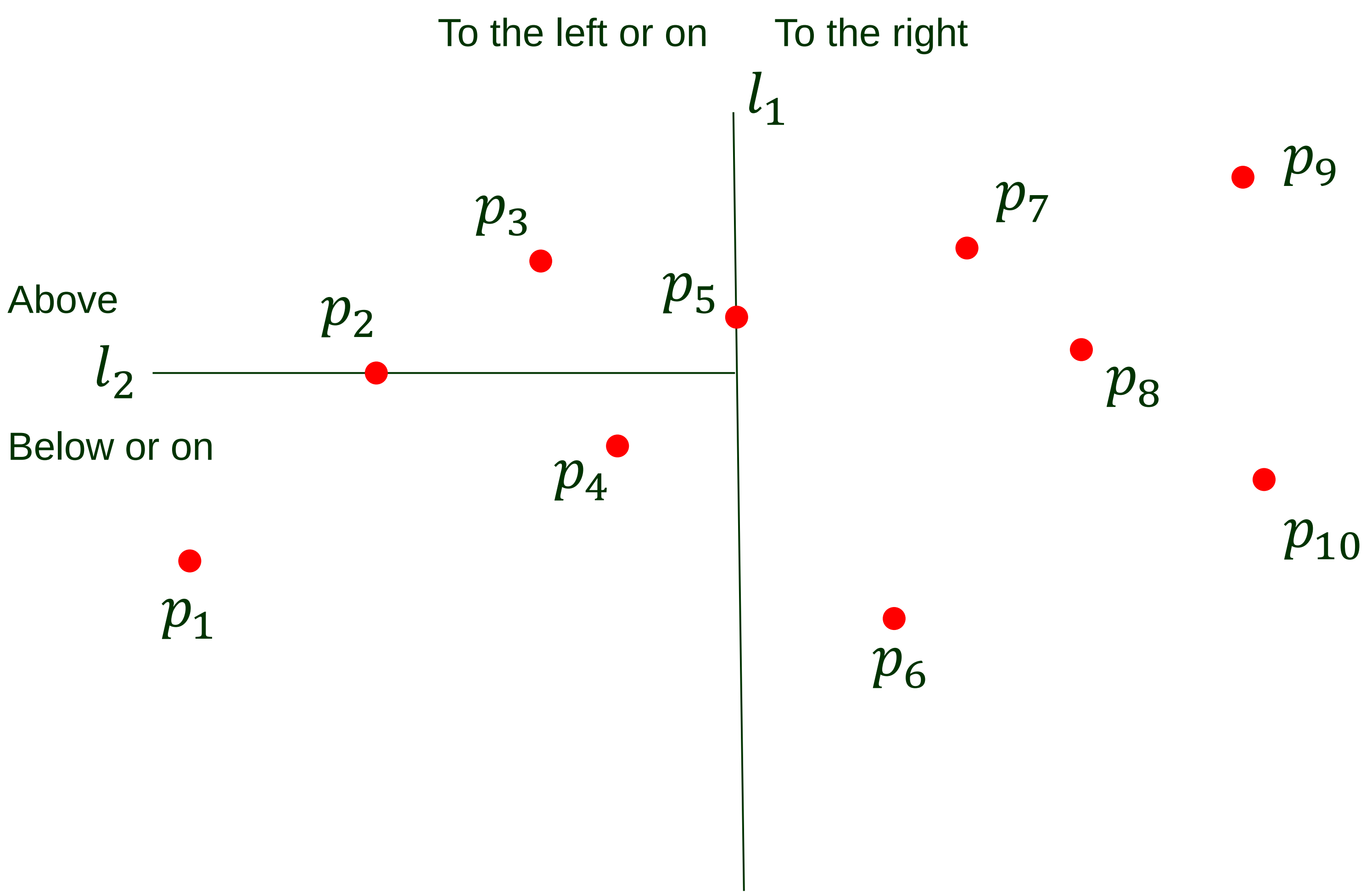
Generation of a Kd-Tree



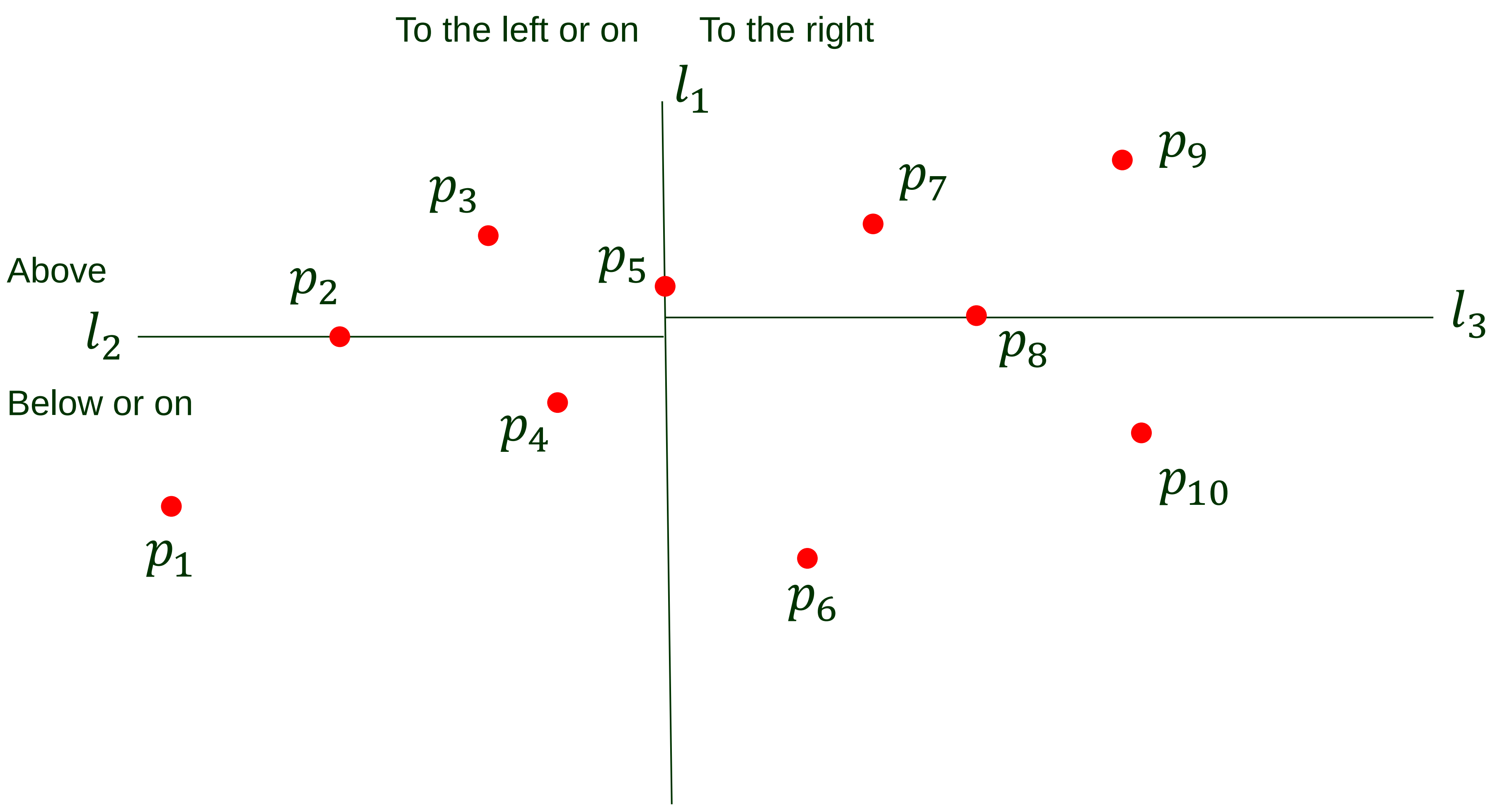
Generation of a Kd-Tree



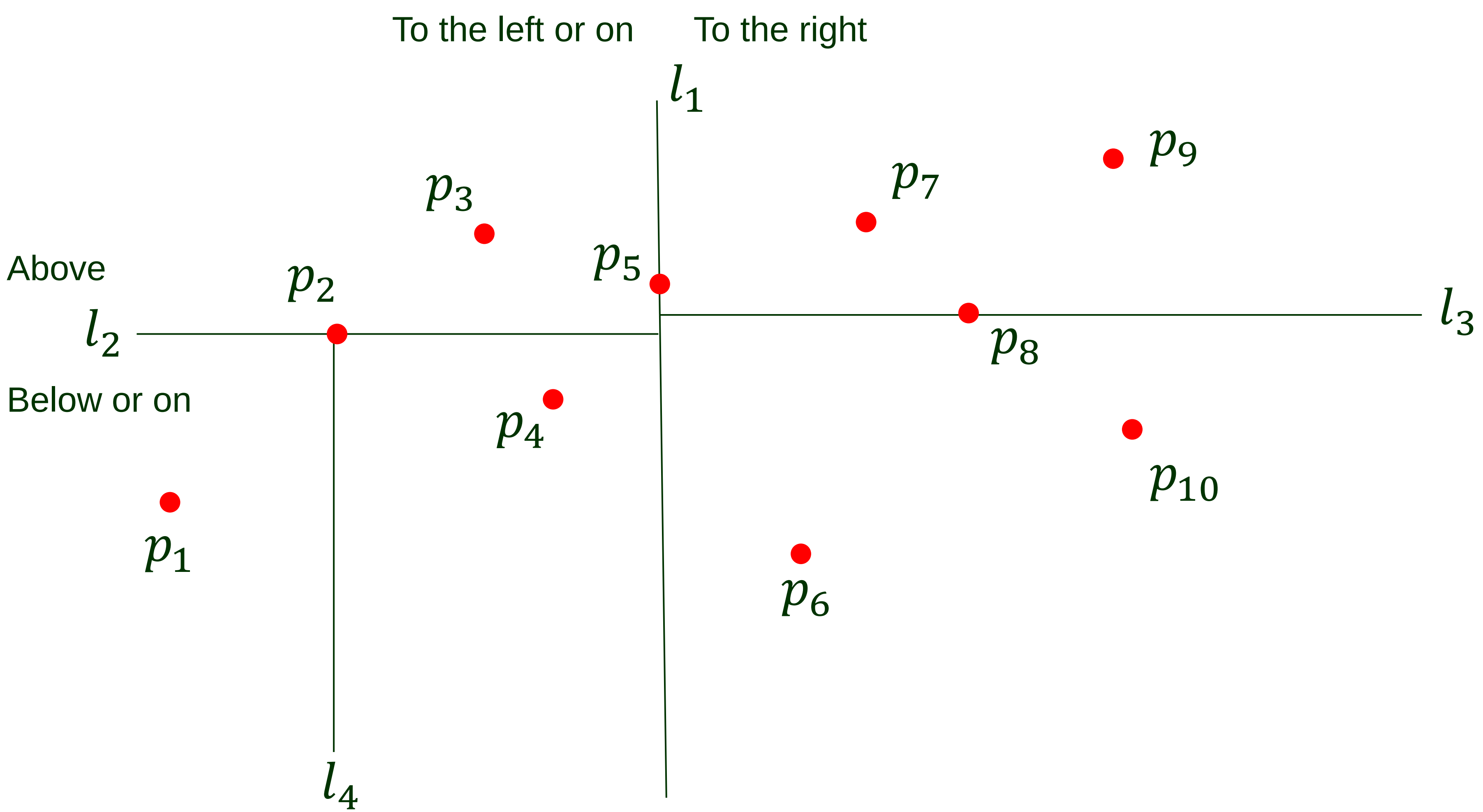
Generation of a Kd-Tree



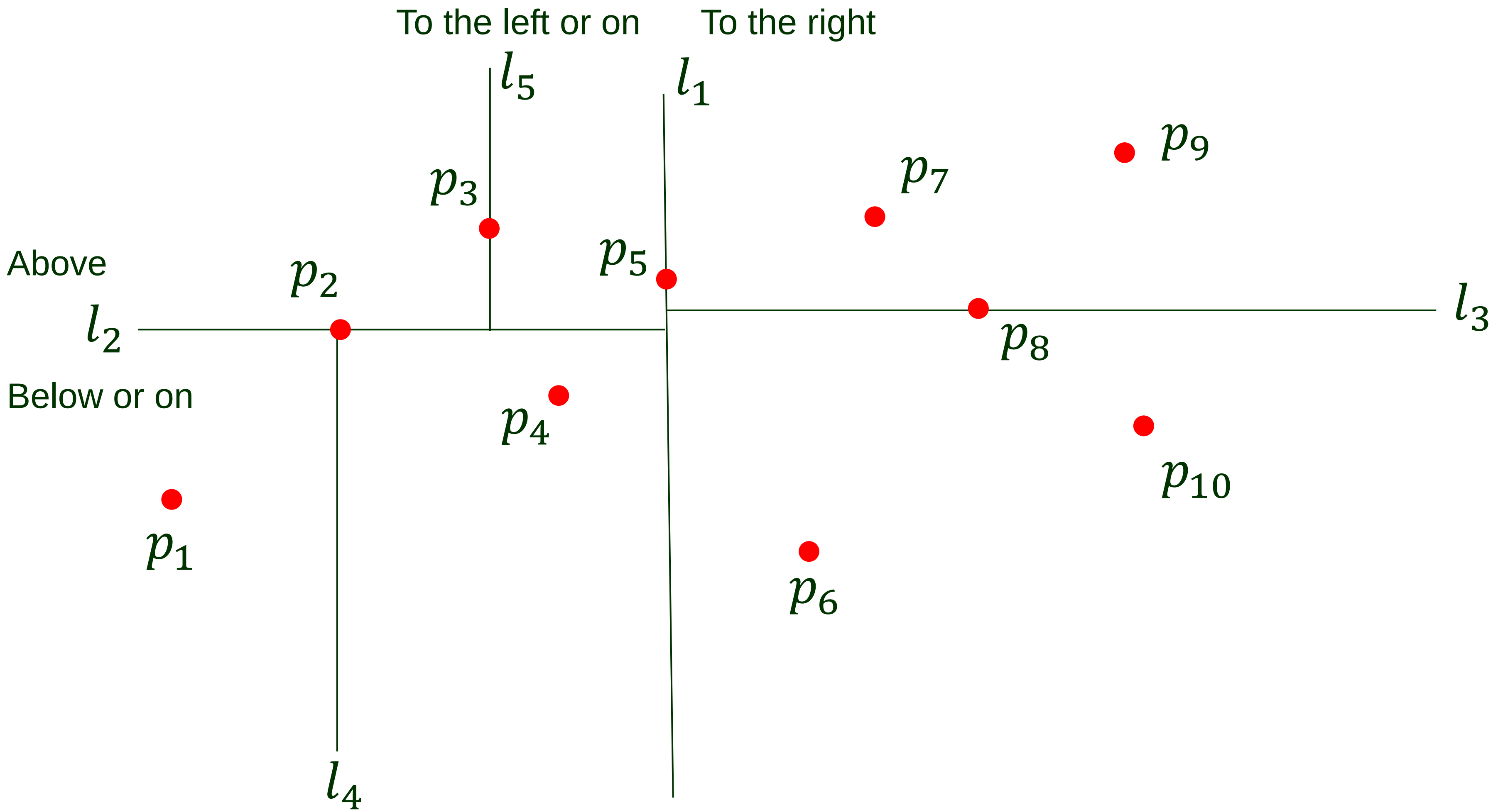
Generation of a Kd-Tree



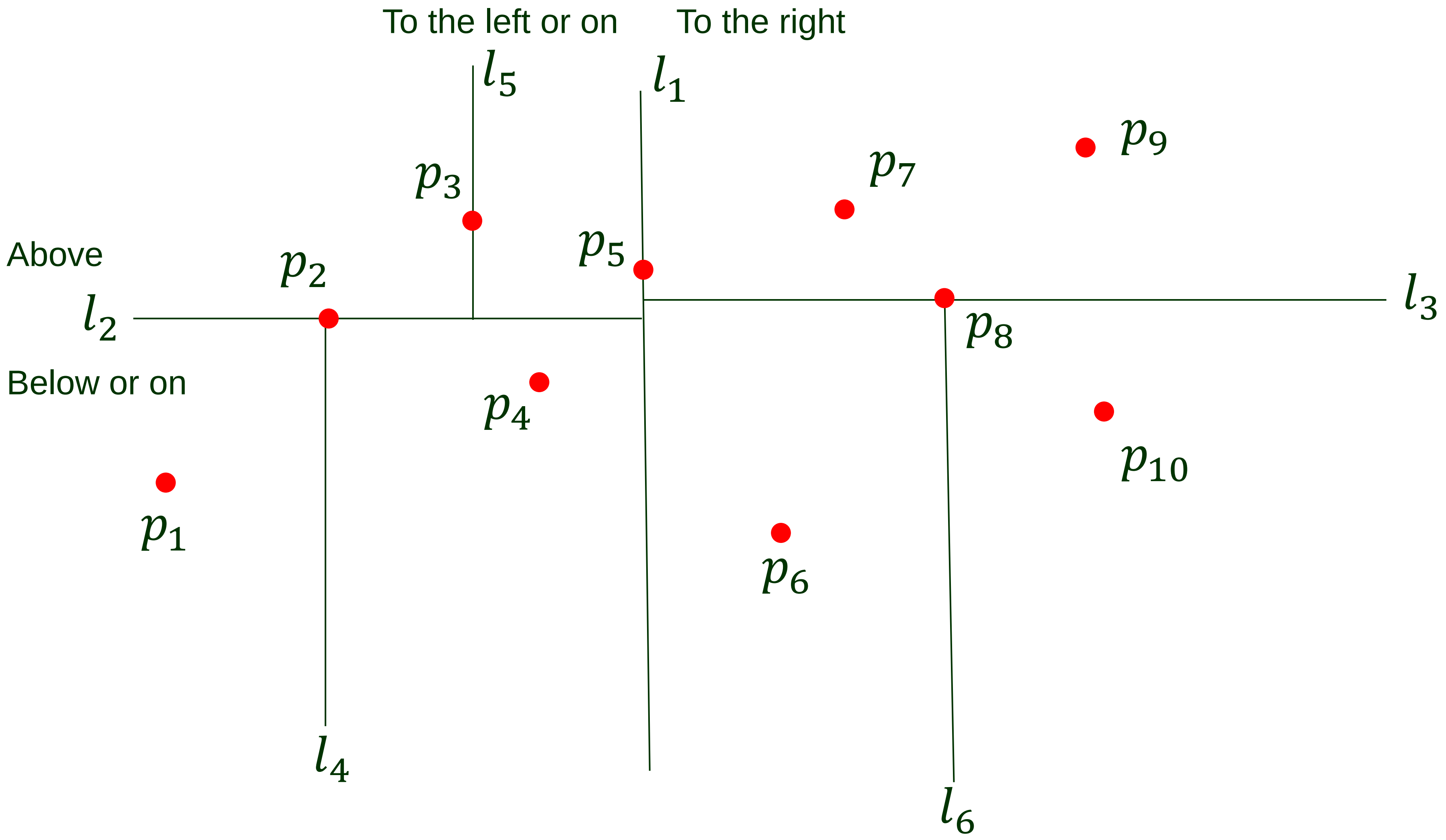
Generation of a Kd-Tree



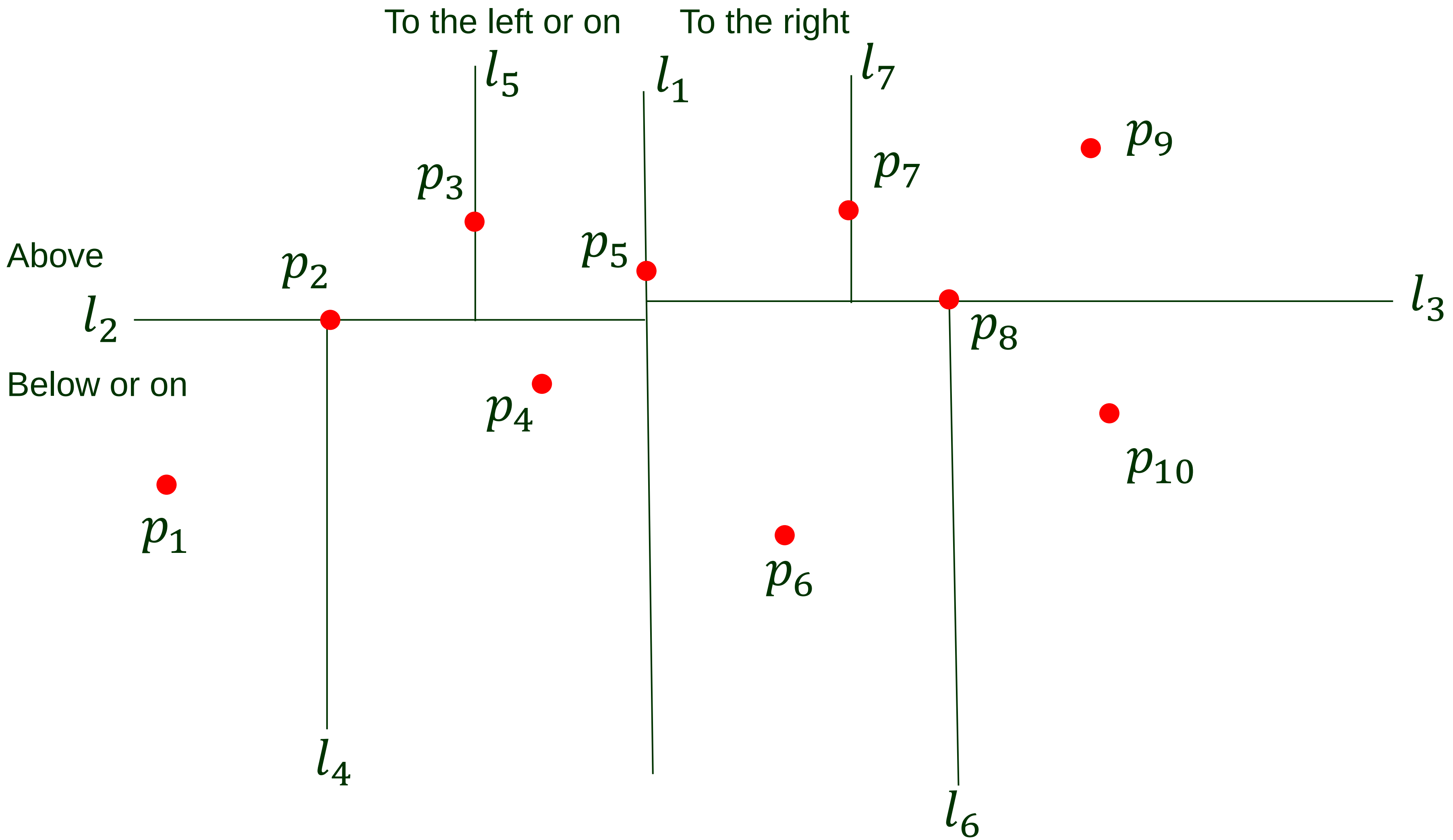
Generation of a Kd-Tree



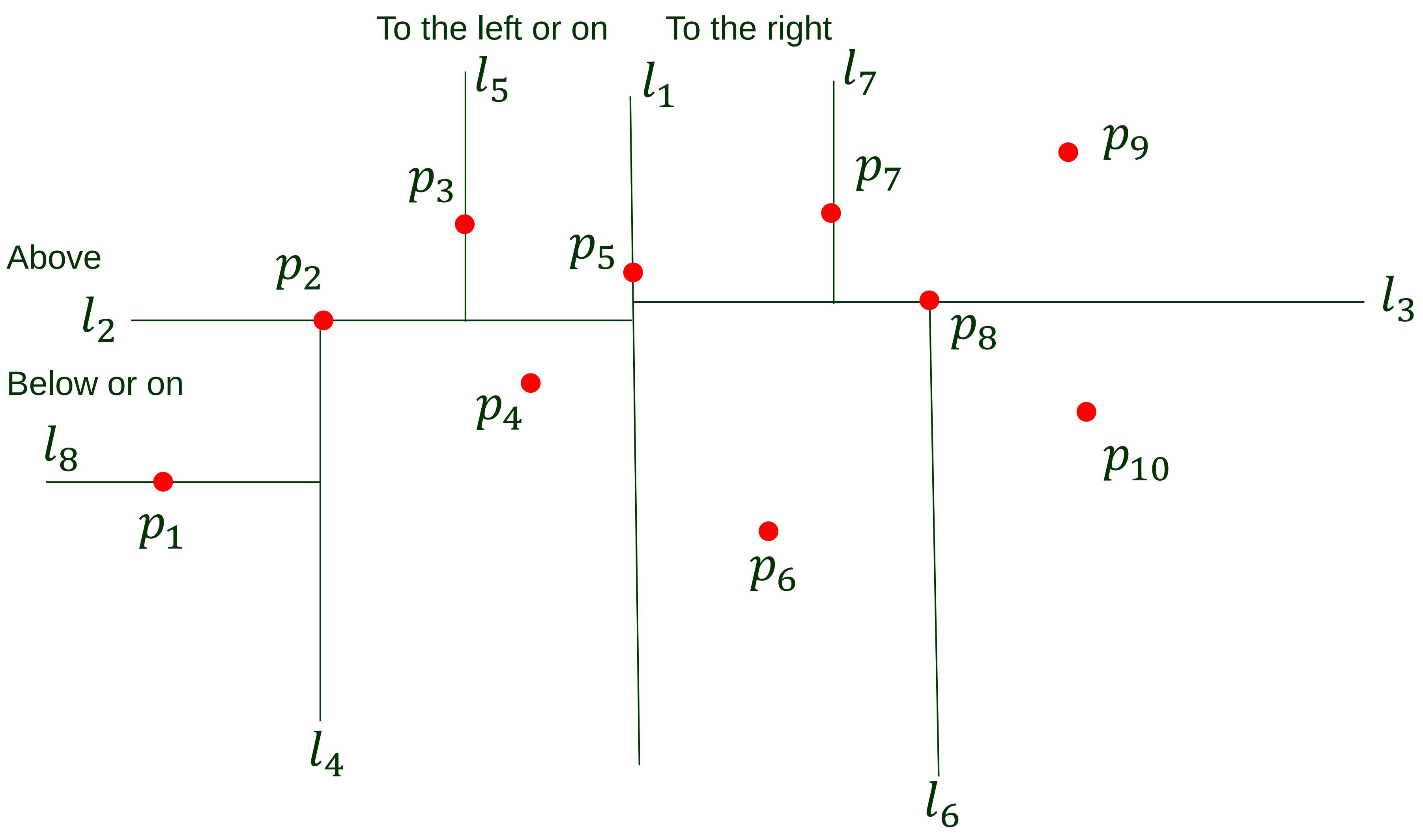
Generation of a Kd-Tree



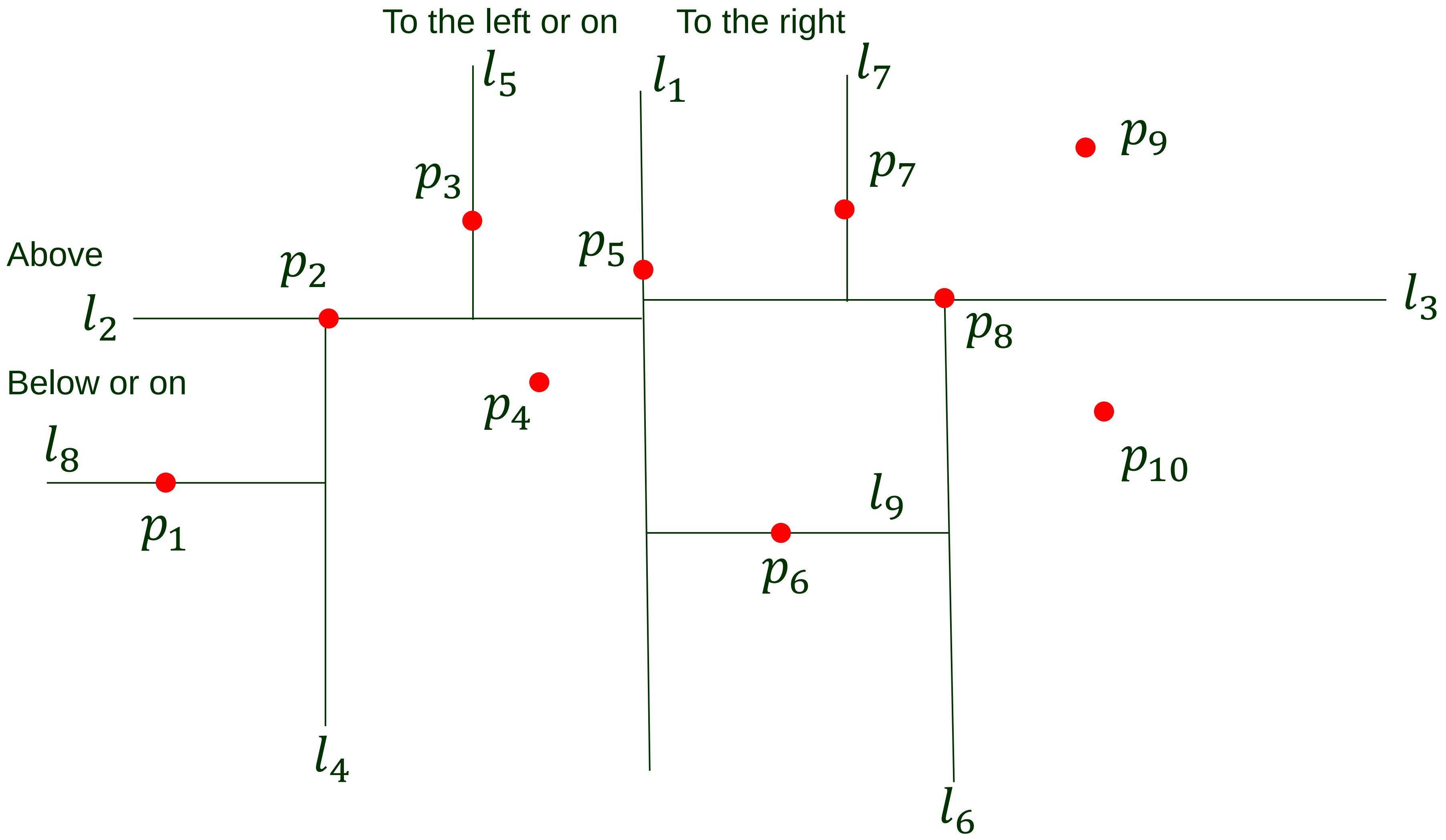
Generation of a Kd-Tree



Generation of a Kd-Tree



Generation of a Kd-Tree



Strategy

- Split the point set P equally with a vertical line l_1.

  Store the splitting line l_1 at the root.

Strategy

- Split the point set P equally with a vertical line l_1.

  Store the splitting line l_1 at the root.

- Split the left subset of points equally with a horizontal line l_2.

Strategy

- Split the point set P equally with a vertical line l_1.

  Store the splitting line l_1 at the root.

- Split the left subset of points equally with a horizontal line l_2.

  Split the right subset of points equally with a horizontal line l_3.

Strategy

- Split the point set P equally with a vertical line l_1.

  Store the splitting line l_1 at the root.

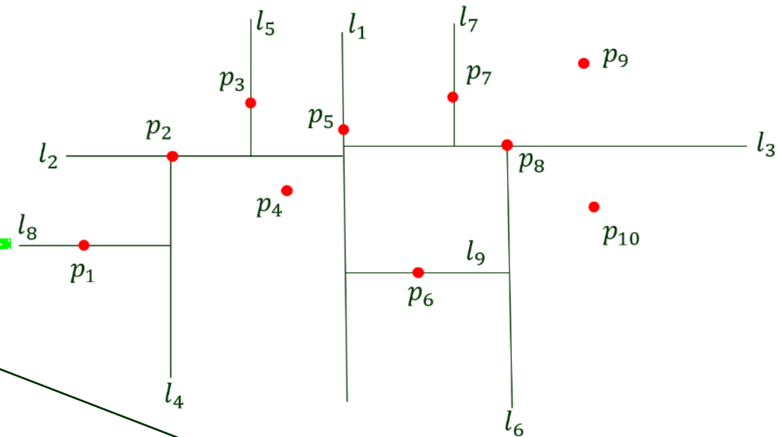
- Split the left subset of points equally with a horizontal line l_2.

  Split the right subset of points equally with a horizontal line l_3.

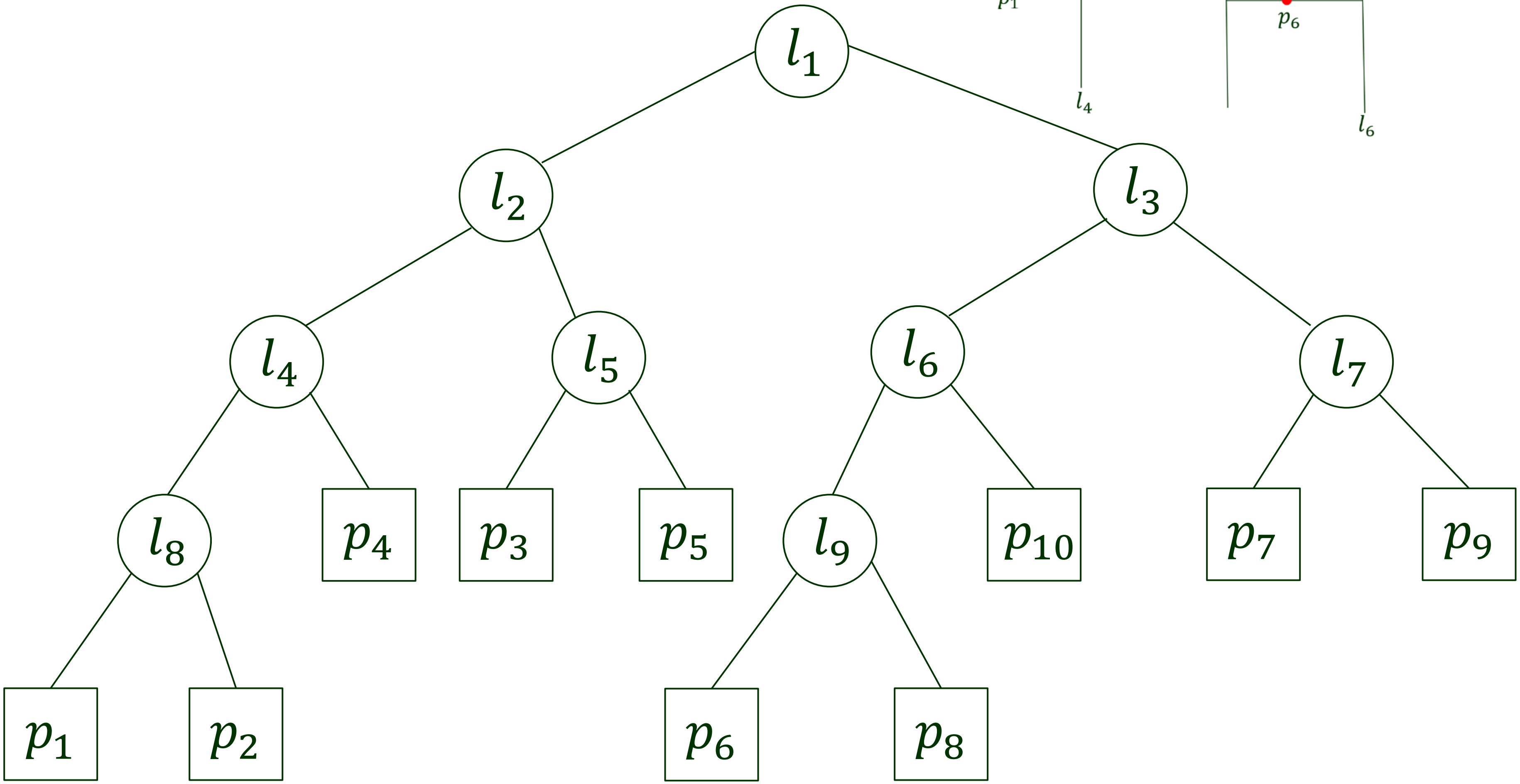
- Split the four subsets vertically again.

…

Kd-Tree

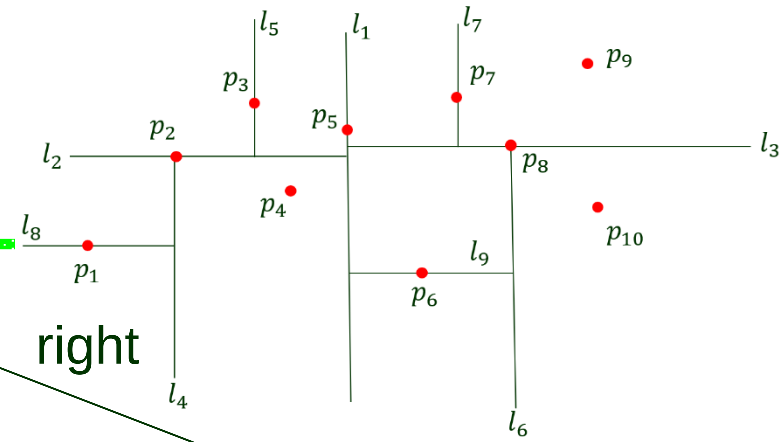


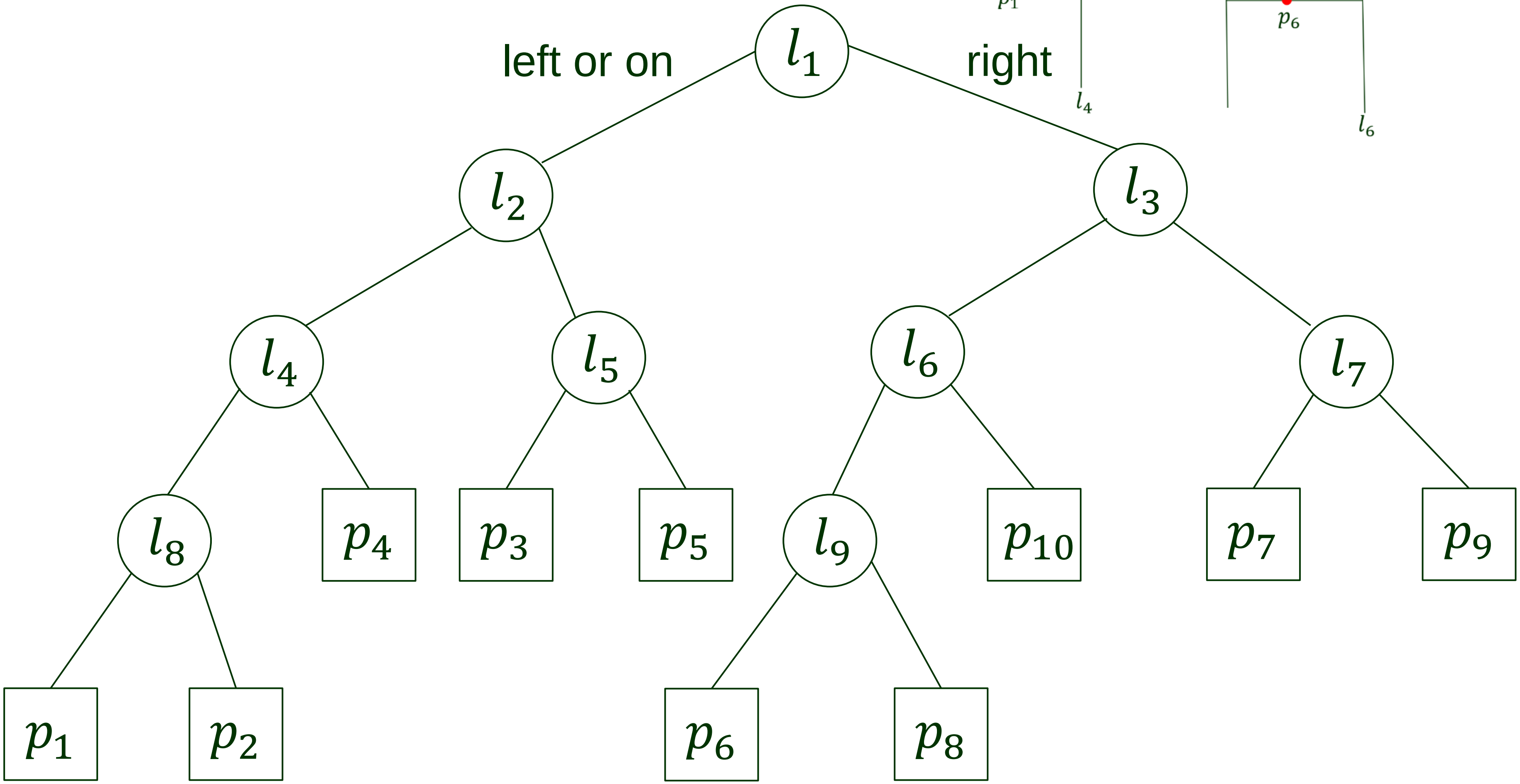
l_5 l_1 l_7
p_9
p_3 p_7
p_5
p_2
l_2 l_3
p_8
p_4
l_8 p_{10}
l_9
p_1
p_6
l_4
l_6



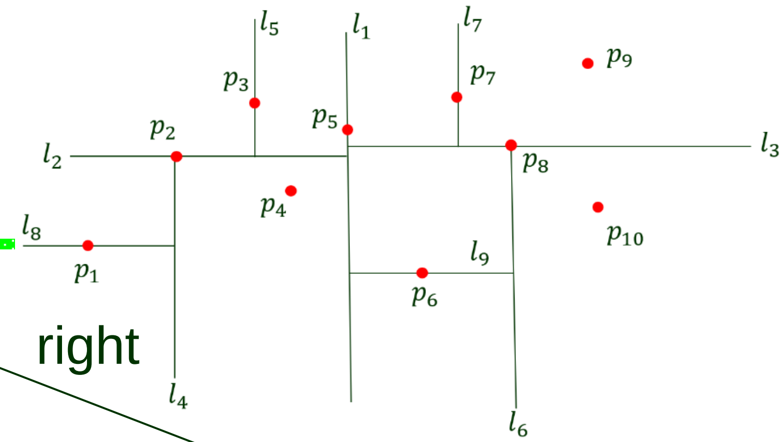
l_1
l_2 l_3
l_4 l_5 l_6 l_7
l_8 p_4 p_3 p_5 l_9 p_{10} p_7 p_9
p_1 p_2 p_6 p_8

Kd-Tree

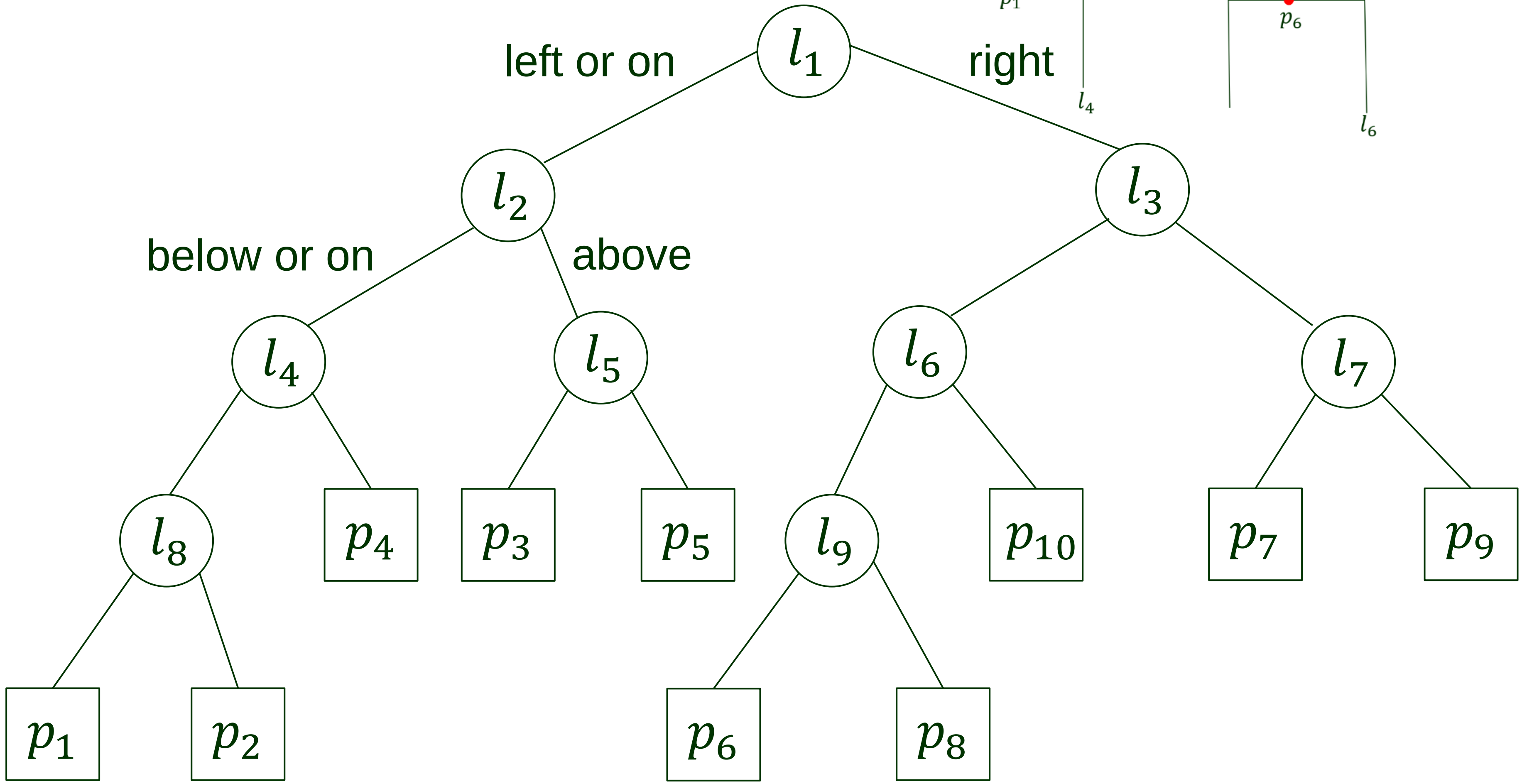

l_5
l_1
l_7
p_9
p_3
p_7
p_5
p_2
l_2
p_8
l_3
p_4
l_8
p_{10}
p_1
l_9
p_6
l_4
l_6


left or on
l_1
right
l_2
l_3
l_4
l_5
l_6
l_7
l_8
p_4
p_3
p_5
l_9
p_{10}
p_7
p_9
p_1
p_2
p_6
p_8

Kd-Tree



l_5
l_1
l_7
p_9
p_3
p_7
p_5
p_2
l_2
l_3
p_8
p_4
l_8
p_{10}
p_1
l_9
p_6
l_4
l_6



l_1
left or on
right
l_2
l_3
below or on
above
l_4
l_5
l_6
l_7
l_8
p_4
p_3
p_5
l_9
p_{10}
p_7
p_9
p_1
p_2
p_6
p_8

Splitting

At the *median*: $\lceil n/2 \rceil$ th smallest number.

Splitting

At the *median*: $\lceil n/2 \rceil$ th smallest number.

- Vertical split by l

$$\{\, p \mid p \text{ on or to the left of } l \,\}$$
$$\cup \ \{\, p \mid p \text{ to the right of } l \,\}$$

Splitting

At the *median*: $\lceil n/2 \rceil$ th smallest number.

- Vertical split by l

$$\{\, p \mid p \text{ on or to the left of } l \,\}$$
$$\cup \; \{\, p \mid p \text{ to the right of } l \,\}$$

- Horizontal split by l

$$\{\, p \mid p \text{ on or below } l \,\}$$
$$\cup \; \{\, p \mid p \text{ above } l \,\}$$

Tree Construction

BuildKdTree(P, d)

depth

```
if |P| = 1
    then return a leaf
    else if d even
            then split P with a vertical line l through
                 the median x-coordinate into P1 and P2
            else split P with a horizontal line l through
                 the median y-coordinate into P1 and P2
            v_left ← BuildKdTree(P1, d + 1)
            v_right ← BuildKdTree(P2, d + 1)
```

if $|P| = 1$

then return a leaf

else if d even

then split P with a vertical line l through the median x-coordinate into P_1 and P_2

else split P with a horizontal line l through the median y-coordinate into P_1 and P_2

$v_{left} \longleftarrow$ BuildKdTree$(P_1, d+1)$

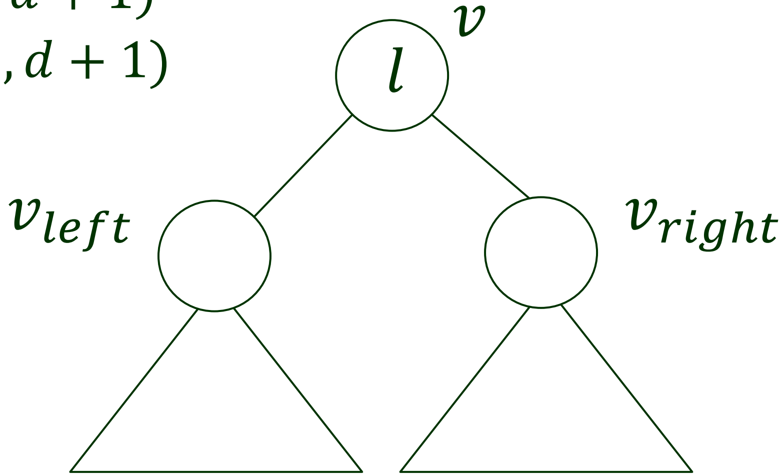
$v_{right} \longleftarrow$ BuildKdTree$(P_2, d+1)$

Tree Construction

```
BuildKdTree(P, d)
```

depth

```
if |P| = 1
    then return a leaf
    else if d even
            then split P with a vertical line l through
                 the median x-coordinate into P_1 and P_2
            else split P with a horizontal line l through
                 the median y-coordinate into P_1 and P_2
            v_left ←BuildKdTree(P_1, d + 1)
            v_right ←BuildKdTree(P_2, d + 1)
```

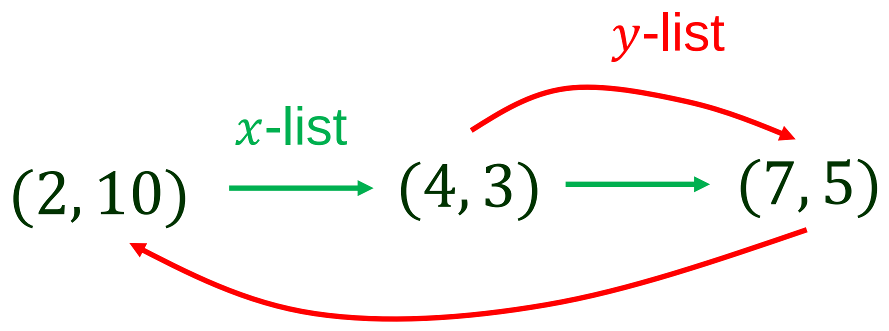


Presorting

For efficiency of splitting, maintain n points in two lists:

- one sorted on the x-coordinate
- the other sorted on the y-coordinate

Every point is stored only once.



Building Time

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ O(n) + 2T(\lceil \frac{n}{2} \rceil) & \text{if } n > 1 \end{cases}$$

Building Time

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ O(n) + 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) & \text{if } n > 1 \end{cases}$$

$$T(n) = O(n \log n)$$

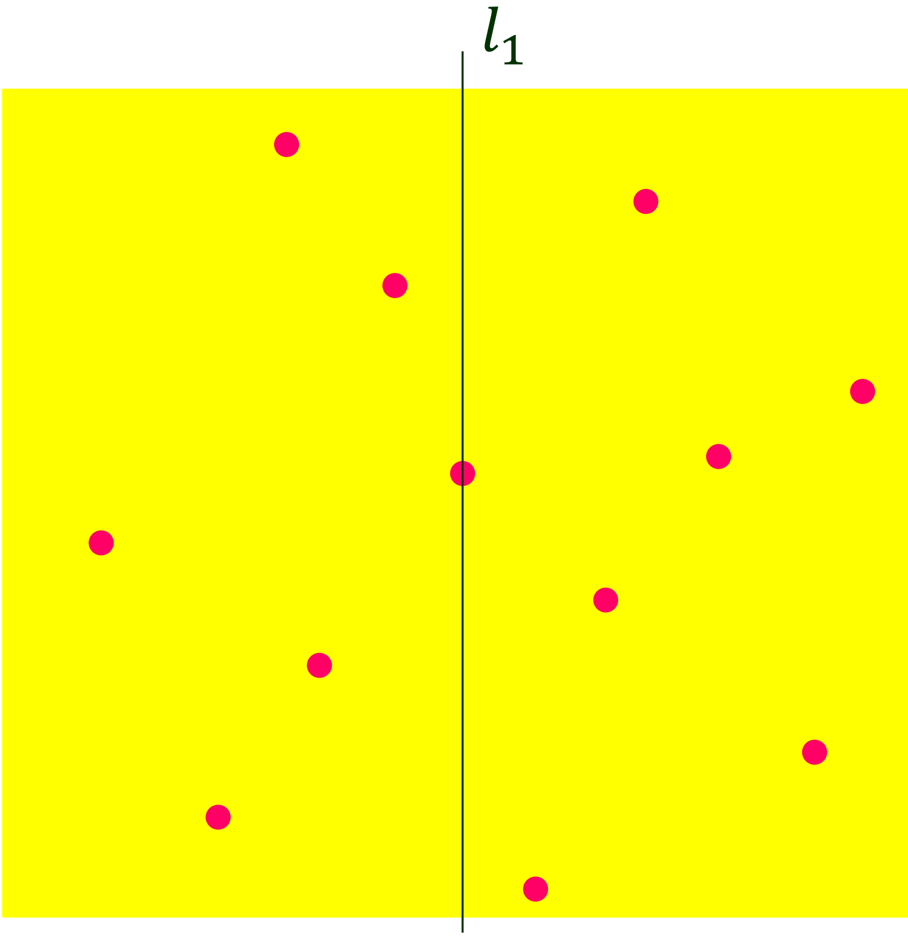
Building Time

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ O(n) + 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) & \text{if } n > 1 \end{cases}$$

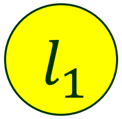
$$T(n) = O(n \log n)$$

Subsumes sorting time!

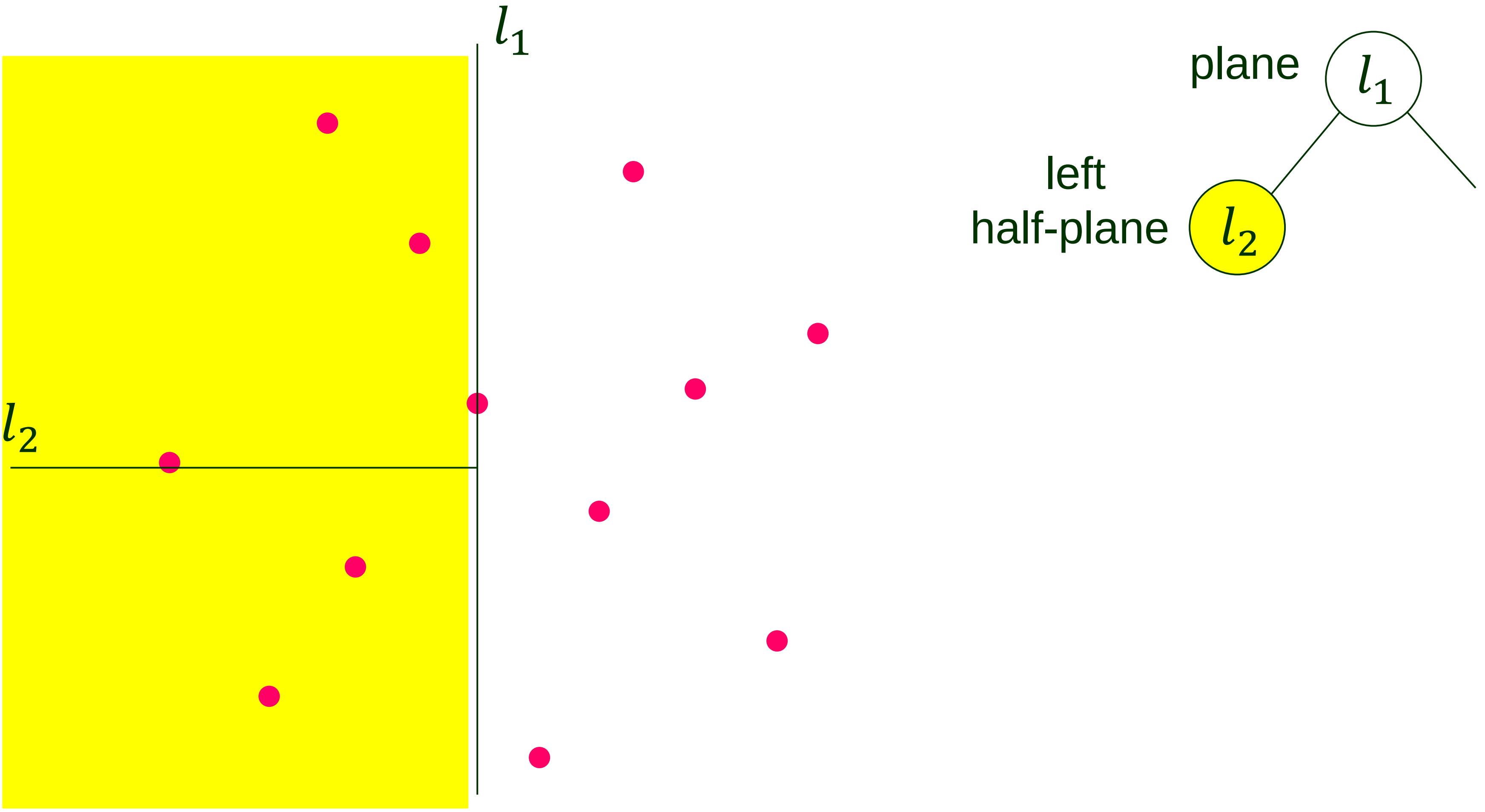
II. Root of the Kd-Tree



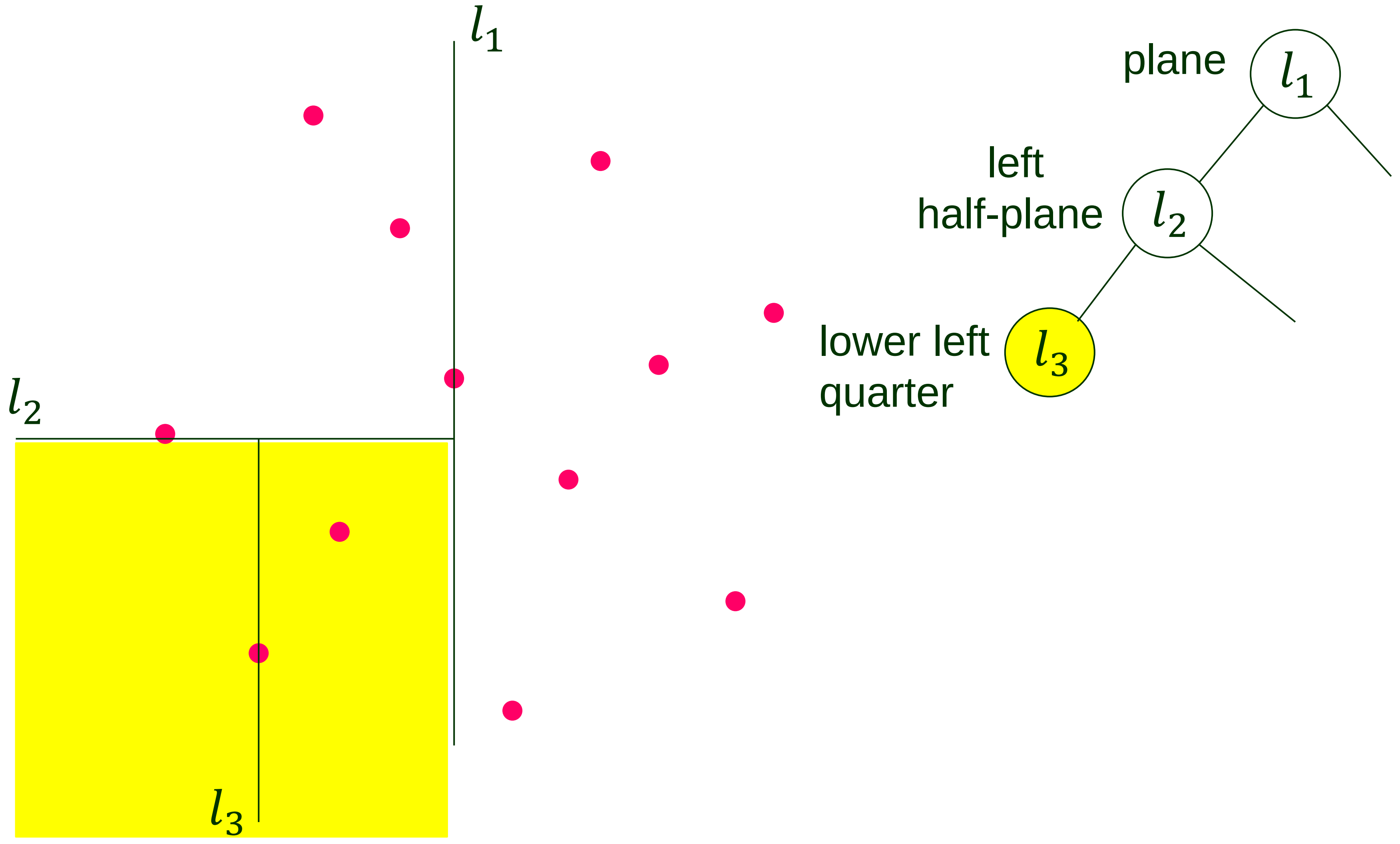
plane



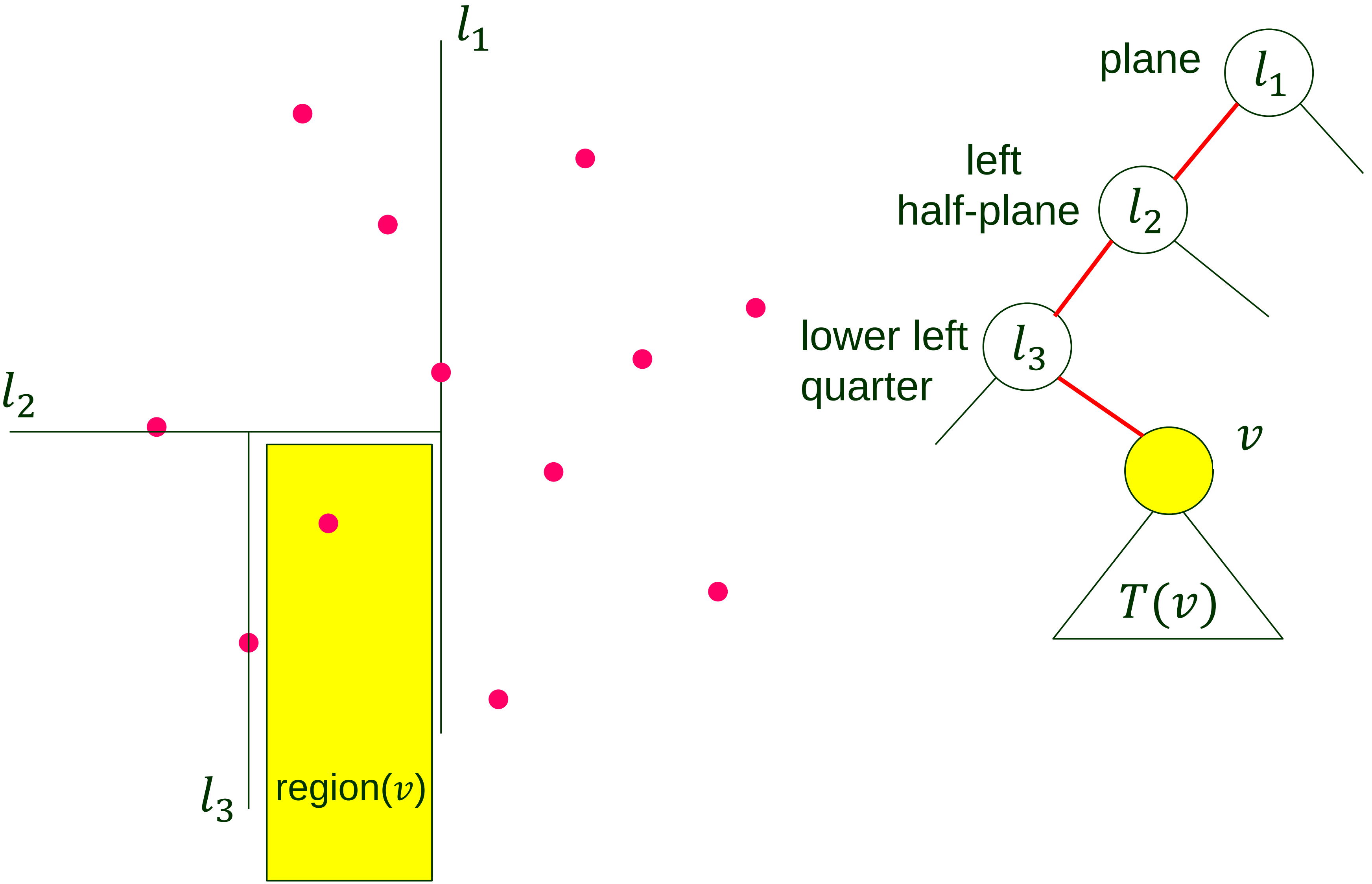
Internal Node



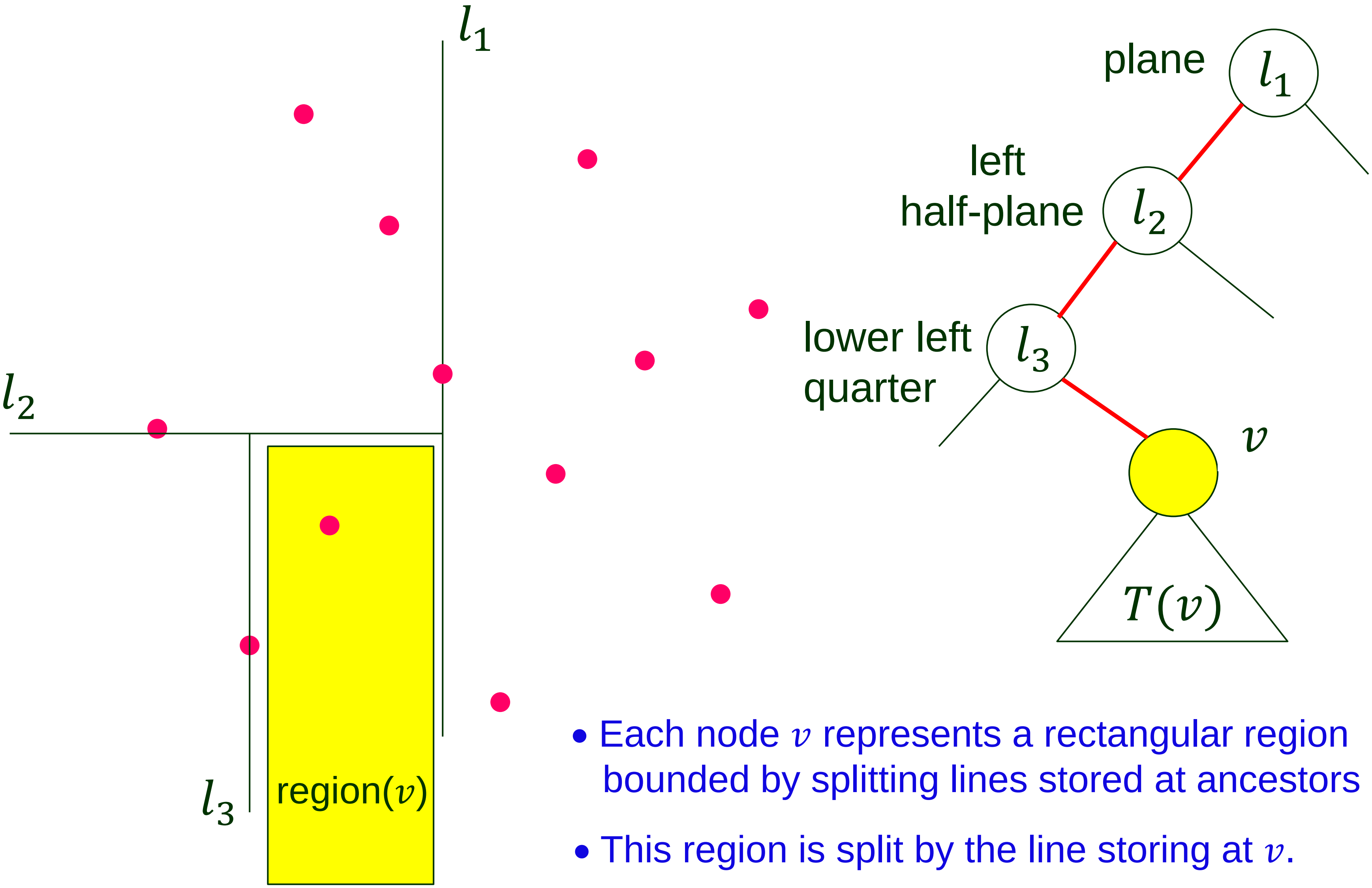
(cont'd)



Node as a Rectangular Region

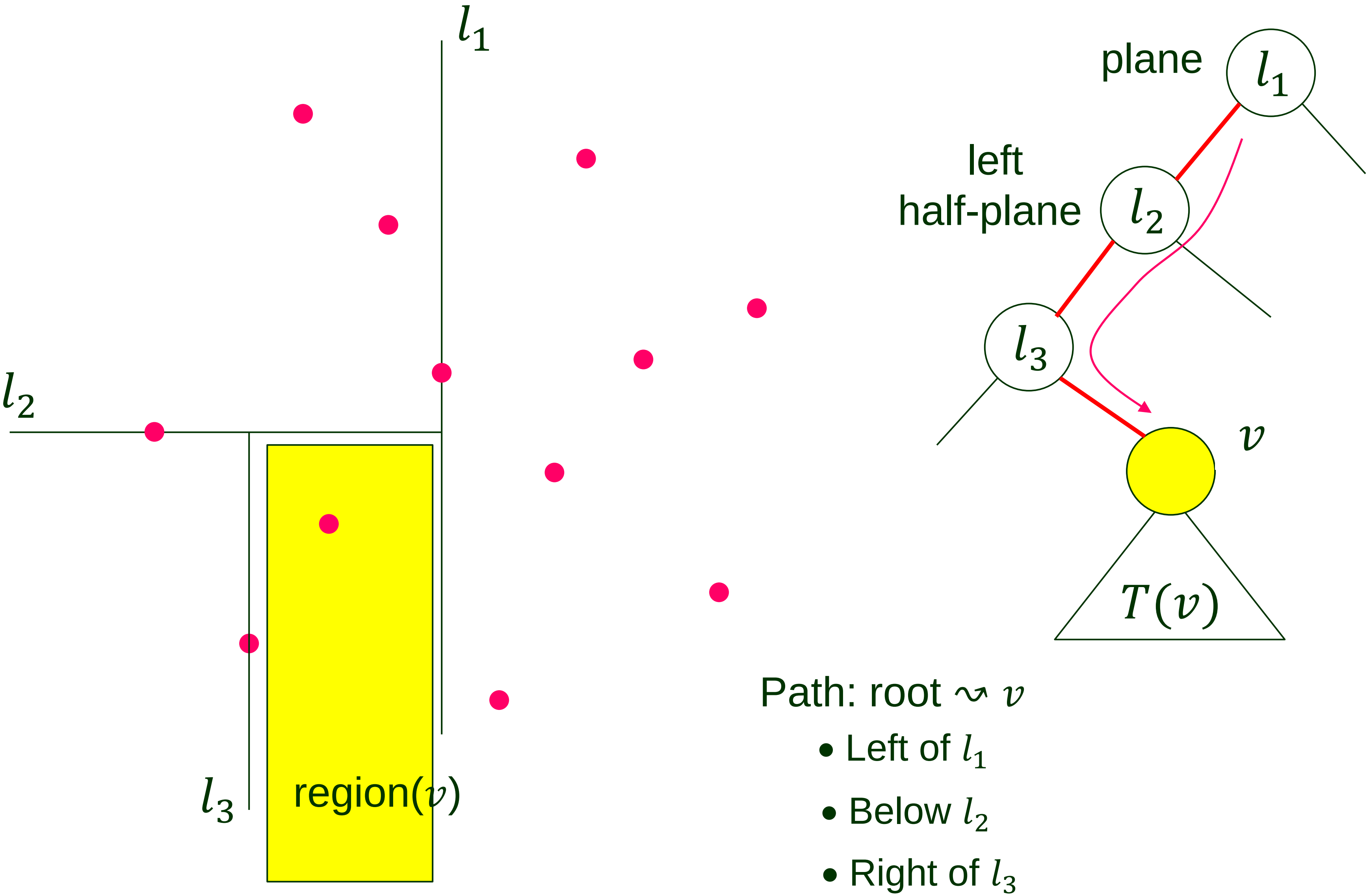


Node as a Rectangular Region



- Each node v represents a rectangular region bounded by splitting lines stored at ancestors of v.
- This region is split by the line storing at v.

How to Locate the Region?



Path: root $\rightsquigarrow v$

- Left of l_1
- Below l_2
- Right of l_3

III. Range Searching

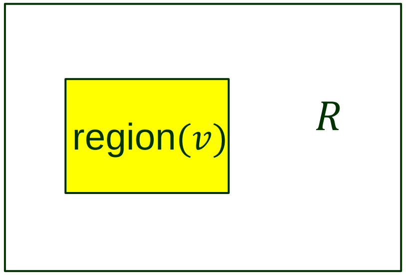
Strategy: Search the subtree rooted at v only if the query rectangle R intersects region(v).

III. Range Searching

Strategy: Search the subtree rooted at v only if the query rectangle R intersects region(v).

- region(v) $\subseteq R$

  Report all leaves (points) of $T(v)$.

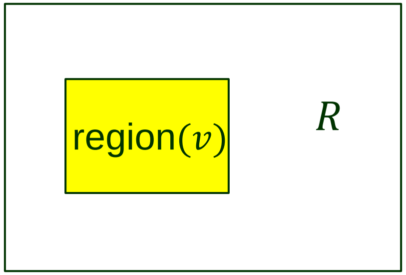


III. Range Searching

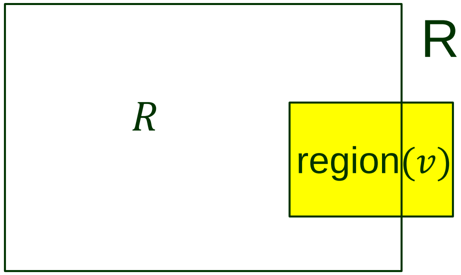
Strategy: Search the subtree rooted at v only if the query rectangle R intersects region(v).

- region(v) $\subseteq R$

  Report all leaves (points) of $T(v)$.

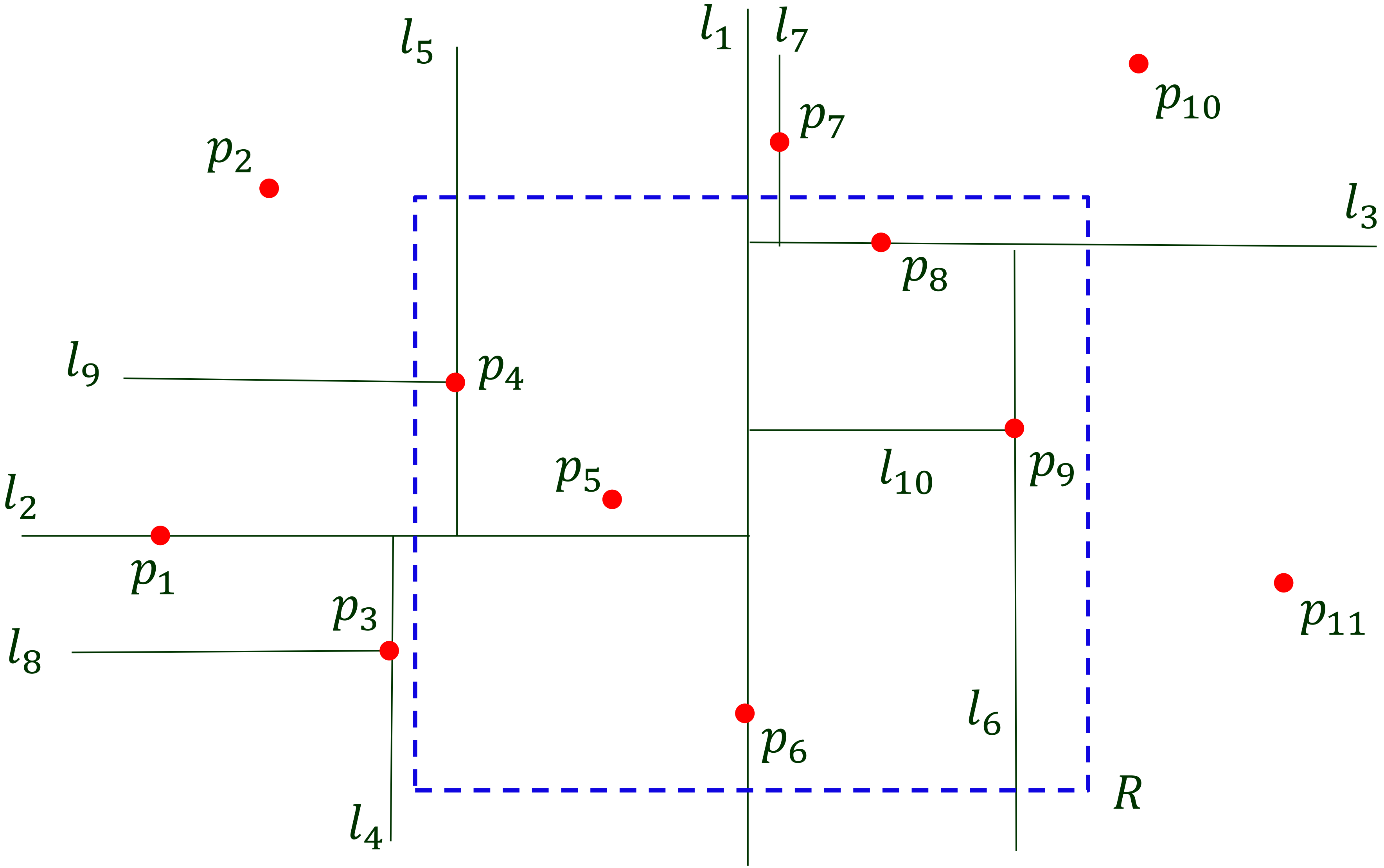


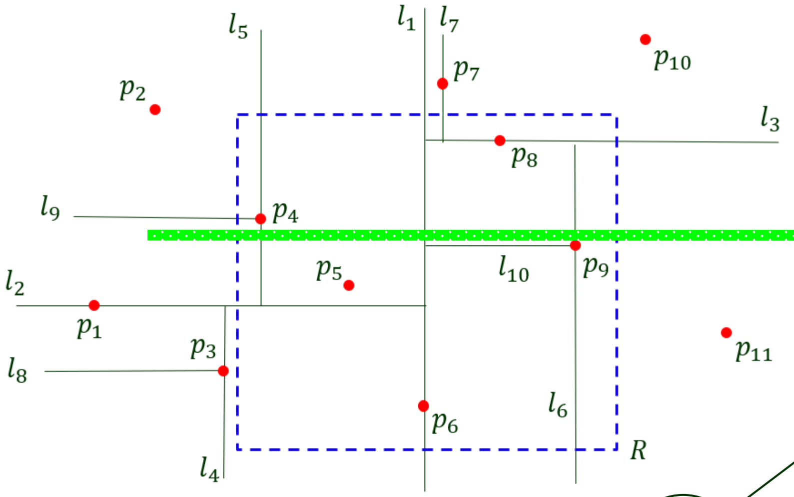
- region(v) $\not\subseteq R$ and region(v) $\cap R \neq \emptyset$



Recursively check region(lchild(v)) and region(rchild(v)).

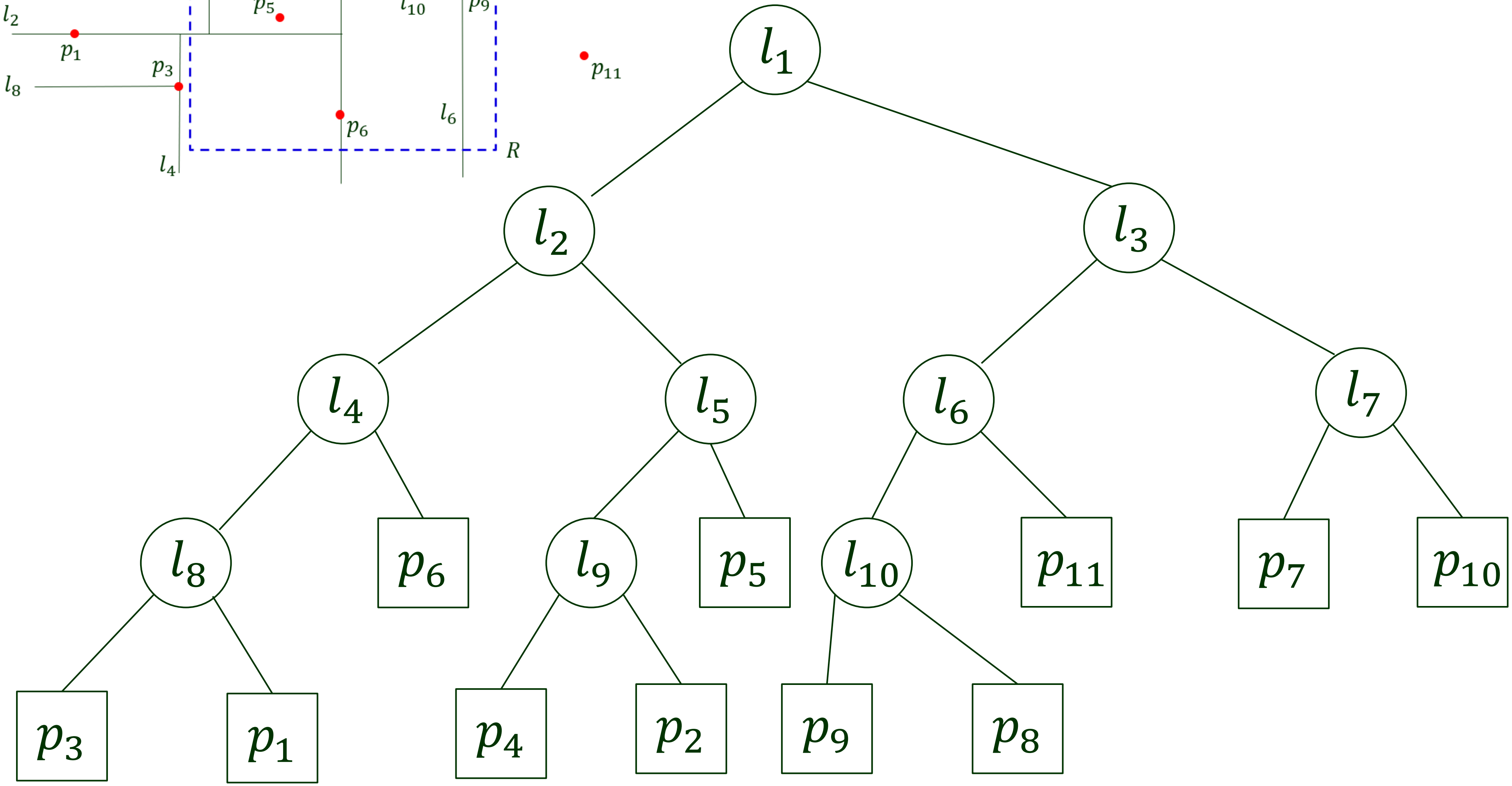
Example





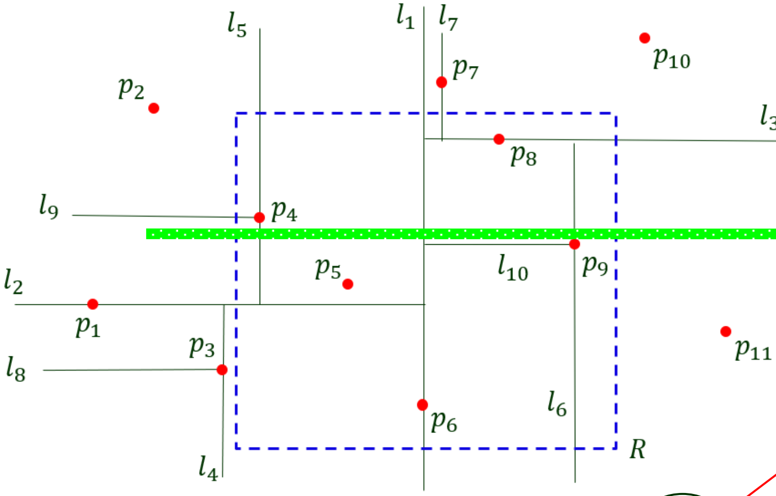
l_5
l_1
l_7
p_{10}
p_7
p_2
l_3
p_8
l_9
p_4
l_{10}
p_9
p_5
l_2
p_1
p_{11}
p_3
l_8
p_6
l_6
R
l_4

(cont'd)

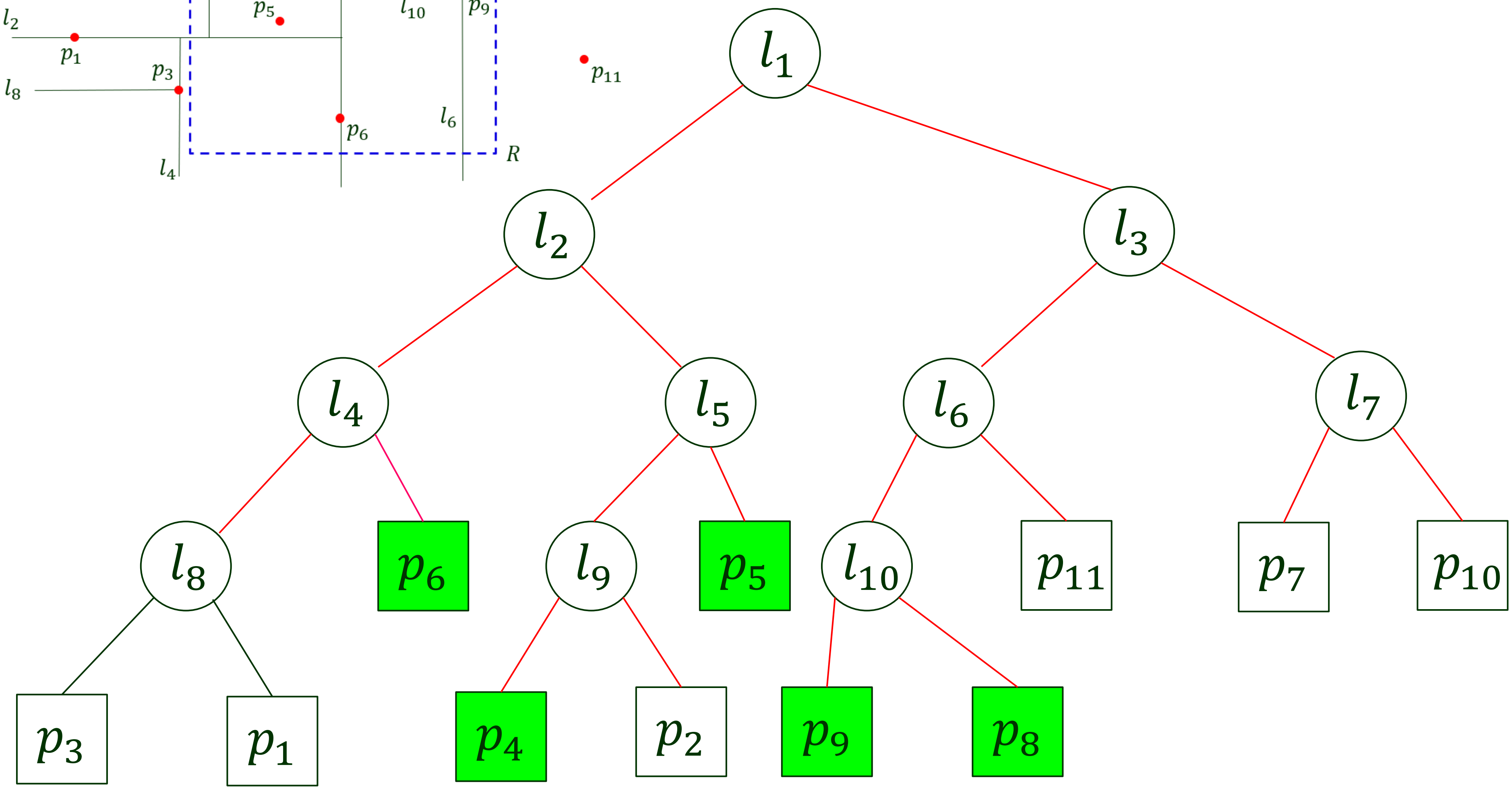


l_1
l_2
l_3
l_4
l_5
l_6
l_7
l_8
p_6
l_9
p_5
l_{10}
p_{11}
p_7
p_{10}
p_3
p_1
p_4
p_2
p_9
p_8

(cont'd)



l_5
l_1
l_7
p_7
p_{10}
p_2
l_3
p_8
l_9
p_4
p_5
l_{10}
p_9
l_2
p_1
p_3
p_{11}
l_8
p_6
l_6
R
l_4



l_1
l_2
l_3
l_4
l_5
l_6
l_7
l_8
p_6
l_9
p_5
l_{10}
p_{11}
p_7
p_{10}
p_3
p_1
p_4
p_2
p_9
p_8

Search Algorithm

```
SearchKdTree(v, R)

if v is a leaf
   then if v ∈ R
           then report v
   else if region(lchild(v)) ⊆ R
           then ReportSubtree(lchild(v)) // O(# leaves)
           else if region(lchild(v)) ∩ R ≠ ∅
                   then SearchKdTree(lchild(v), R)
        if region(rchild(v)) ⊆ R
           then ReportSubtree(rchild(v)) // O(# leaves)
           else if region(rchild(v)) ∩ R ≠ ∅
                   then SearchKdTree(rchild(v), R)
```

Search Algorithm

```
SearchKdTree(v, R)

if v is a leaf
   then if v ∈ R
           then report v
   else if region(lchild(v)) ⊆ R
           then ReportSubtree(lchild(v)) // O(# leaves)
           else if region(lchild(v)) ∩ R ≠ ∅
                    then SearchKdTree(lchild(v), R)
        if region(rchild(v)) ⊆ R
           then ReportSubtree(rchild(v)) // O(# leaves)
           else if region(rchild(v)) ∩ R ≠ ∅
                    then SearchKdTree(rchild(v), R)
```

Current region is maintained through recursive calls:

Search Algorithm

```
SearchKdTree(v, R)

if v is a leaf
    then if v ∈ R
             then report v
    else if region(lchild(v)) ⊆ R
             then ReportSubtree(lchild(v)) // O(# leaves)
             else if region(lchild(v)) ∩ R ≠ ∅
                      then SearchKdTree(lchild(v), R)
         if region(rchild(v)) ⊆ R
             then ReportSubtree(rchild(v)) // O(# leaves)
             else if region(rchild(v)) ∩ R ≠ ∅
                      then SearchKdTree(rchild(v), R)
```

Current region is maintained through recursive calls:

- l: line stored at node v

Search Algorithm

```
SearchKdTree(v, R)

if v is a leaf
   then if v ∈ R
           then report v
   else if region(lchild(v)) ⊆ R
           then ReportSubtree(lchild(v)) // O(# leaves)
           else if region(lchild(v)) ∩ R ≠ ∅
                    then SearchKdTree(lchild(v), R)
        if region(rchild(v)) ⊆ R
           then ReportSubtree(rchild(v)) // O(# leaves)
           else if region(rchild(v)) ∩ R ≠ ∅
                    then SearchKdTree(rchild(v), R)
```

Current region is maintained through recursive calls:

- l: line stored at node v
- region(lchild(v)) = region(v) ∩ half-plane left of or below l

IV. Time Analysis – Count Node Visits

Determine the *number* of visited nodes.

Type 1

- Nodes reported as a traversal of some subtree (lines 3, 5 & 9)

```
1. if v is a leaf
2.    then if v ∈ R
3.              then report v
4.    else if region(lchild(v)) ⊆ R
5.            then ReportSubtree(lchild(v))
6.            else if region(lchild(v)) ∩ R ≠ ∅
7.                     then SearchKdTree(lchild(v),R)
8.         if region(rchild(v)) ⊆ R
9.            then ReportSubtree(rchild(v))
10.           else if region(rchild(v)) ∩ R ≠ ∅
11.                    then SearchKdTree(rchild(v),R)
```

IV. Time Analysis – Count Node Visits

Determine the *number* of visited nodes.

Type 1

- ♦ Nodes reported as a traversal of some subtree (lines 3, 5 & 9)

```
1.  if v is a leaf
2.     then if v ∈ R
3.              then report v
4.     else if region(lchild(v)) ⊆ R
5.             then ReportSubtree(lchild(v))
6.             else if region(lchild(v)) ∩ R ≠ ∅
7.                      then SearchKdTree(lchild(v), R)
8.          if region(rchild(v)) ⊆ R
9.             then ReportSubtree(rchild(v))
10.            else if region(rchild(v)) ∩ R ≠ ∅
11.                     then SearchKdTree(rchild(v), R)
```

Linear in the number of reported points.

Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

```
1.  if v is a leaf
2.     then if v ∈ R
3.             then report v
4.     else if region(lchild(v)) ⊆ R
5.             then ReportSubtree(lchild(v))
6.             else if region(lchild(v)) ∩ R ≠ ∅
7.                      then SearchKdTree(lchild(v), R)
8.          if region(rchild(v)) ⊆ R
9.             then ReportSubtree(rchild(v))
10.            else if region(rchild(v)) ∩ R ≠ ∅
11.                     then SearchKdTree(rchild(v), R)
```

Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node v satisfies

Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node v satisfies

$$\text{region}(v) \cap R \neq \emptyset$$
$$\text{region}(v) \not\subseteq R$$

Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node v satisfies

$$\text{region}(v) \cap R \neq \emptyset$$
$$\text{region}(v) \not\subseteq R$$

$$\Downarrow$$

\# type 2 nodes $\leq$ # regions intersected by the four edges of R

Type 2 Nodes

Type 2

- ♦ Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node v satisfies

$$\text{region}(v) \cap R \neq \emptyset$$
$$\text{region}(v) \not\subseteq R$$

$$\Downarrow$$

\# type 2 nodes $\leq$ # regions intersected by the four edges of R

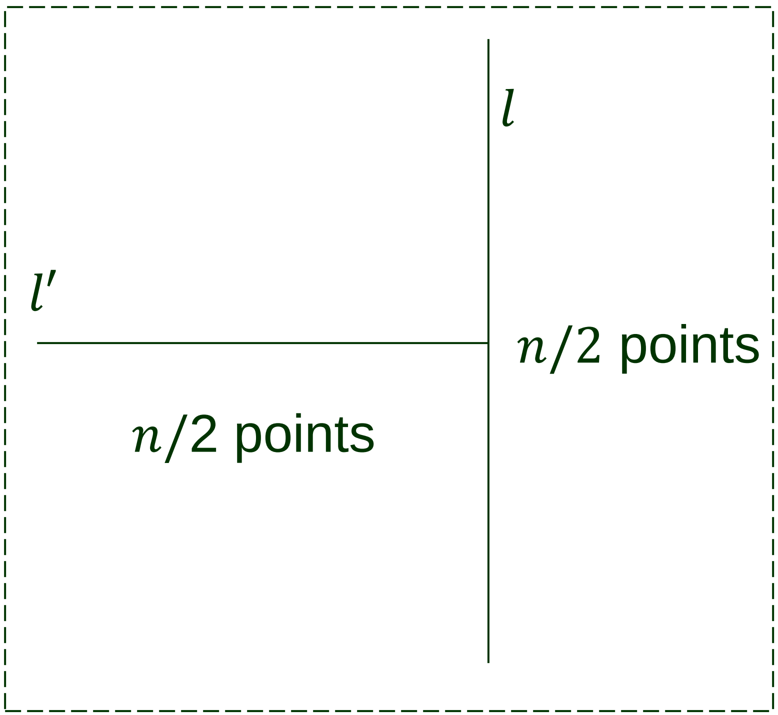
- How to bound #regions intersected by a vertical line?

Setting up a Recurrence

$Q(n)$: number of intersected regions in a kd-tree which

- stores n points, and
- has a vertical line l as the root.

Consider a vertical edge (as a line) of the query box.

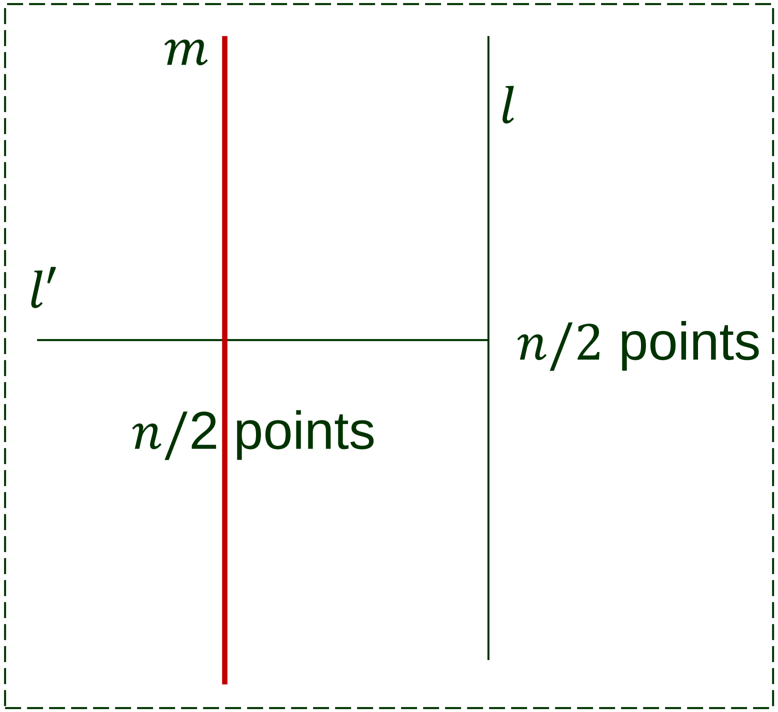


Setting up a Recurrence

$Q(n)$: number of intersected regions in a kd-tree which

- stores n points, and
- has a vertical line l as the root.

Consider a vertical edge (as a line) of the query box.



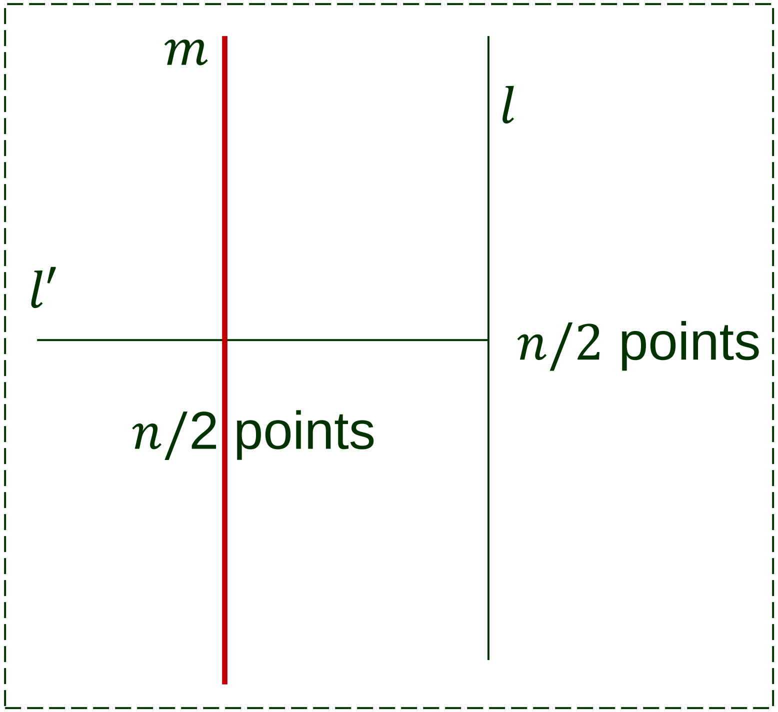
A vertical line m intersects one side of l, say, its left side.

Setting up a Recurrence

$Q(n)$: number of intersected regions in a kd-tree which

- stores n points, and
- has a vertical line l as the root.

Consider a vertical edge (as a line) of the query box.



A vertical line m intersects one side of l, say, its left side.

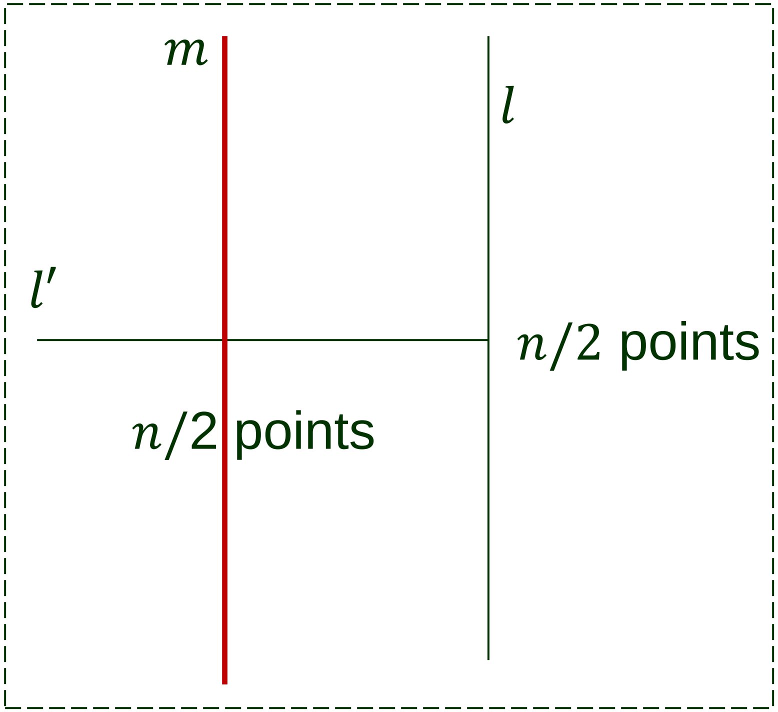
The region corresponds to (i.e., is partitioned by) a horizontal splitting line l'.

Setting up a Recurrence

$Q(n)$: number of intersected regions in a kd-tree which

- stores n points, and
- has a vertical line l as the root.

Consider a vertical edge (as a line) of the query box.



A vertical line m intersects one side of l, say, its left side.

The region corresponds to (i.e., is partitioned by) a horizontal splitting line l'.

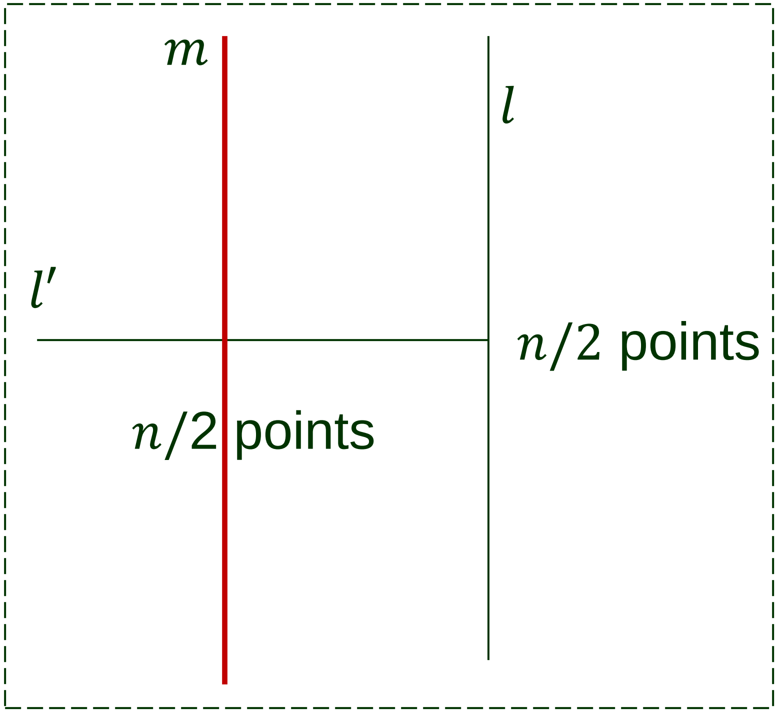
m intersects both children of l', whereas it intersects only one of l.

Setting up a Recurrence

$Q(n)$: number of intersected regions in a kd-tree which

- stores n points, and
- has a vertical line l as the root.

Consider a vertical edge (as a line) of the query box.



A vertical line m intersects one side of l, say, its left side.

The region corresponds to (i.e., is partitioned by) a horizontal splitting line l'.

m intersects both children of l', whereas it intersects only one of l.

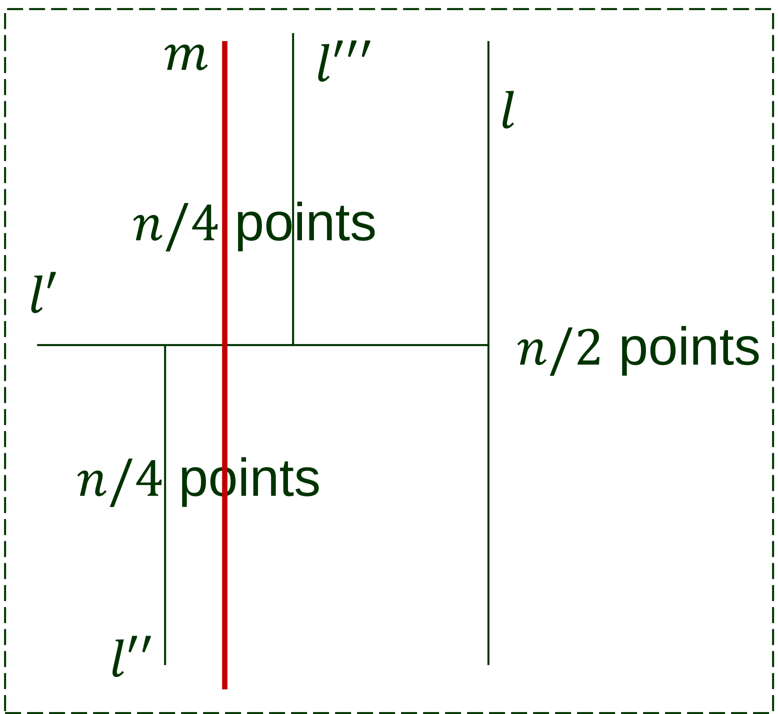
$\Downarrow$

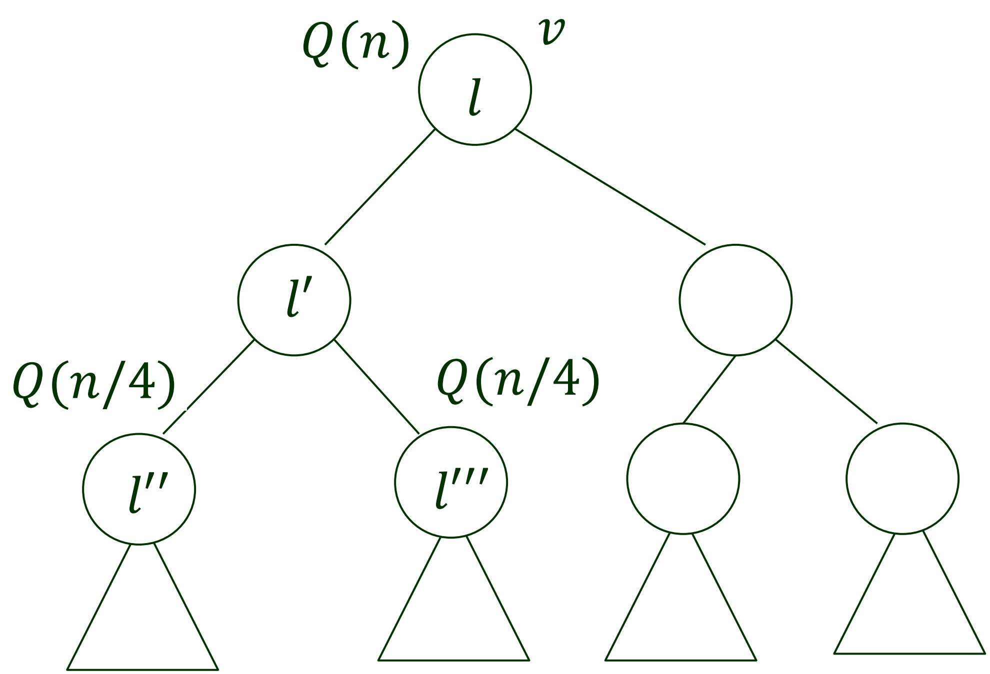
Not a recurrence situation!

Recurrence (cont'd)

Go down two levels!

- Four nodes at depth 2
- Each corresponds to $n/4$ points.
- Only two represent intersected regions.





Running Time

$$Q(n) = \begin{cases} Q(1) & \text{if } n = 1 \\ 2 + 2\,Q(n/4) & \text{if } n > 1 \end{cases}$$

Running Time

$$Q(n) = \begin{cases} Q(1) & \text{if } n = 1 \\ 2 + 2\,Q(n/4) & \text{if } n > 1 \end{cases}$$

2 out of 4 regions
represented by
grandchild nodes

Running Time

$$Q(n) = \begin{cases} Q(1) & \text{if } n = 1 \\ 2 + 2\,Q(n/4) & \text{if } n > 1 \end{cases}$$

2 out of 4 regions
represented by
grandchild nodes

$$\Downarrow$$

$$Q(n) = O(\sqrt{n})$$

Generalization to Higher Dimensions

- At the root, split into two subsets based on x_1 coordinate.
- At depth 1, partition based on x_2 coordinate.

$$\vdots$$

- At depth $d - 1$, partition based on x_d coordinate.
- At depth d, partition based on x_1 coordinate.

$$\vdots$$

Recursion stops when the subset has one point.

d-Dimensional Kd-Tree

Binary tree with n leaves (points)

- $O(n)$ storage
- $O(n \log n)$ construction time
- $O(n^{1-\frac{1}{d}} + k)$ query time

d-Dimensional Kd-Tree

Binary tree with n leaves (points)

- $O(n)$ storage
- $O(n \log n)$ construction time
- $O(n^{1-\frac{1}{d}} + k)$ query time

reported points