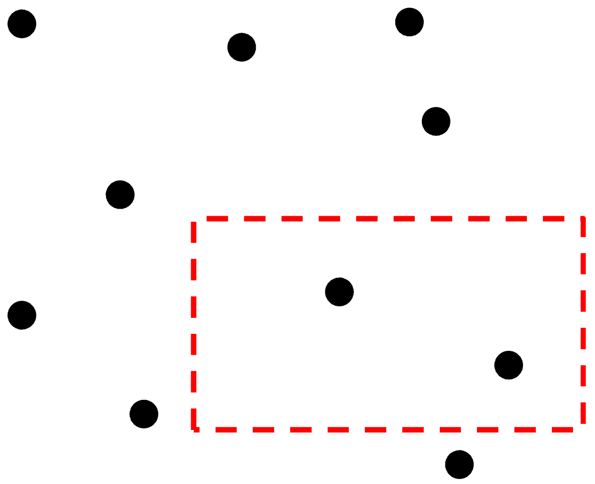
2D Range Query

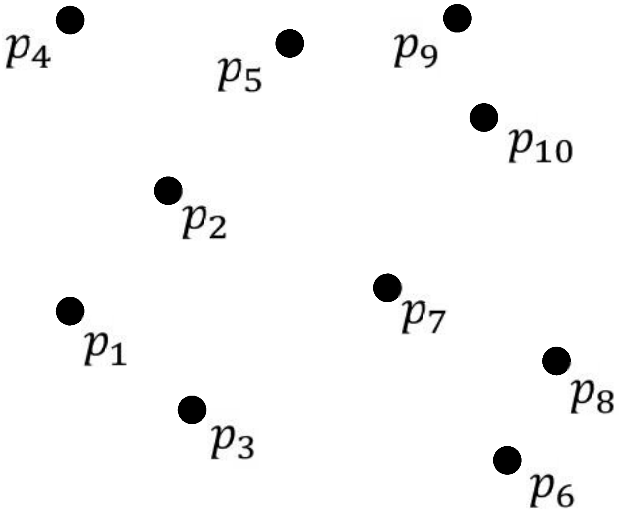
KD-tree and Range tree

2D Orthogonal Range Query

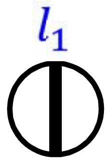
- Data: A set S of n points
- Query: Report/Count subset of S that lie in a rectangle range

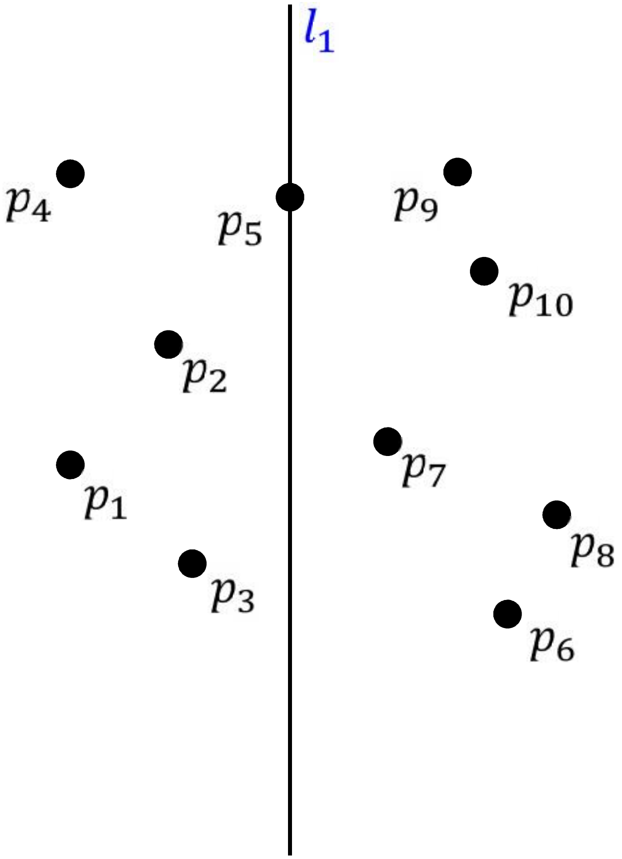


K-d Tree

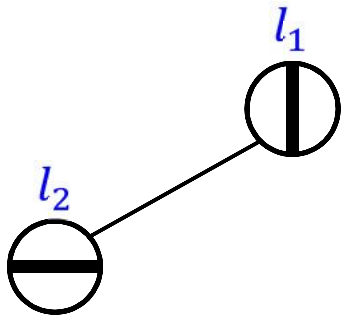


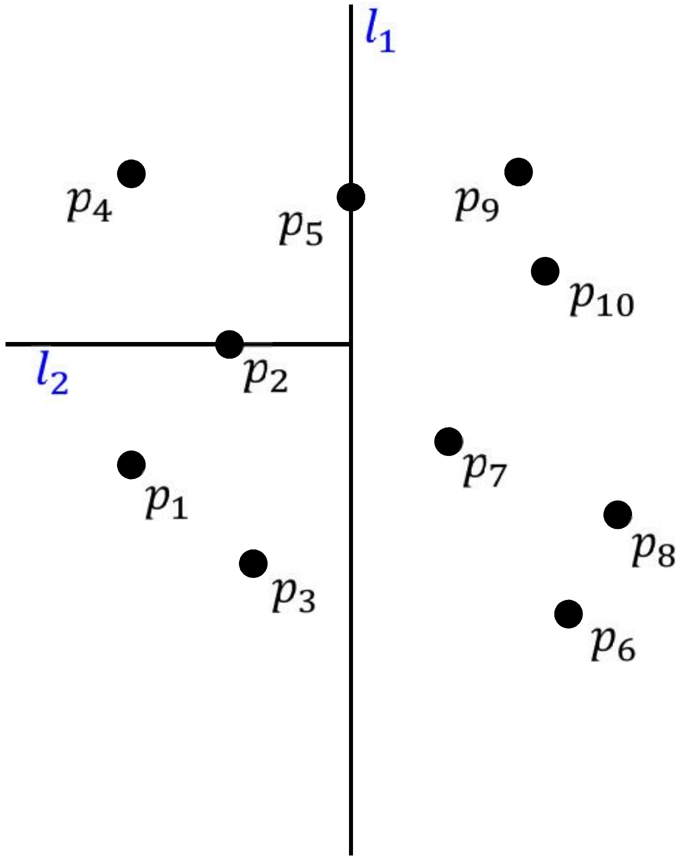
K-d Tree



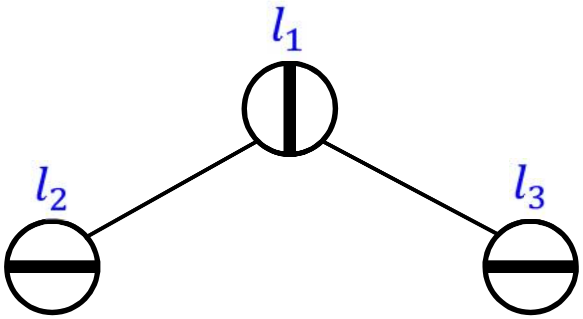


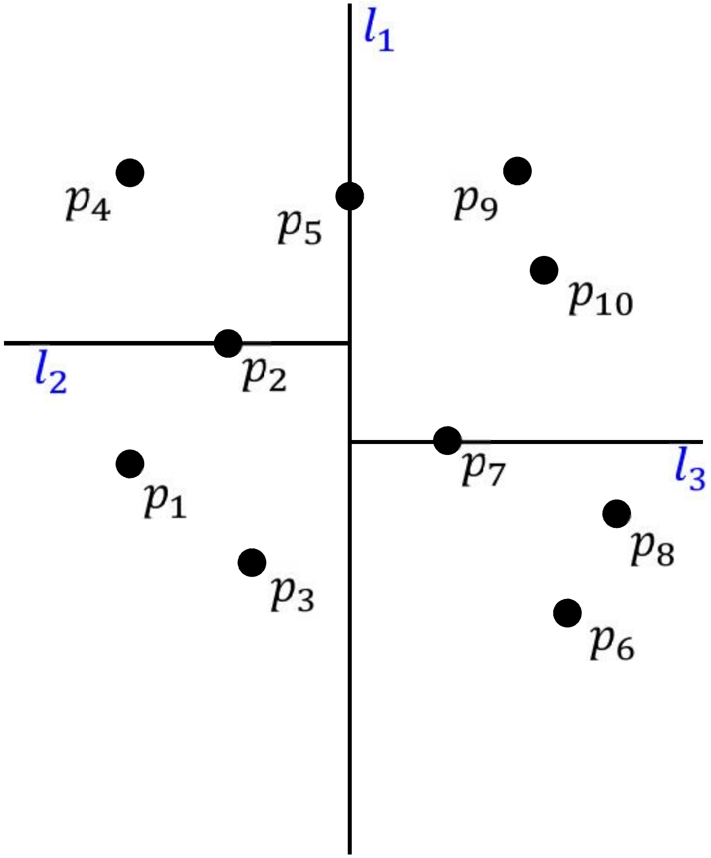
K-d Tree



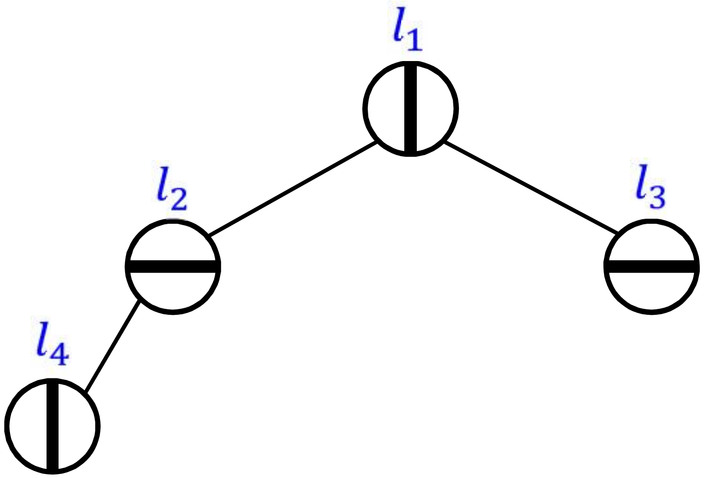


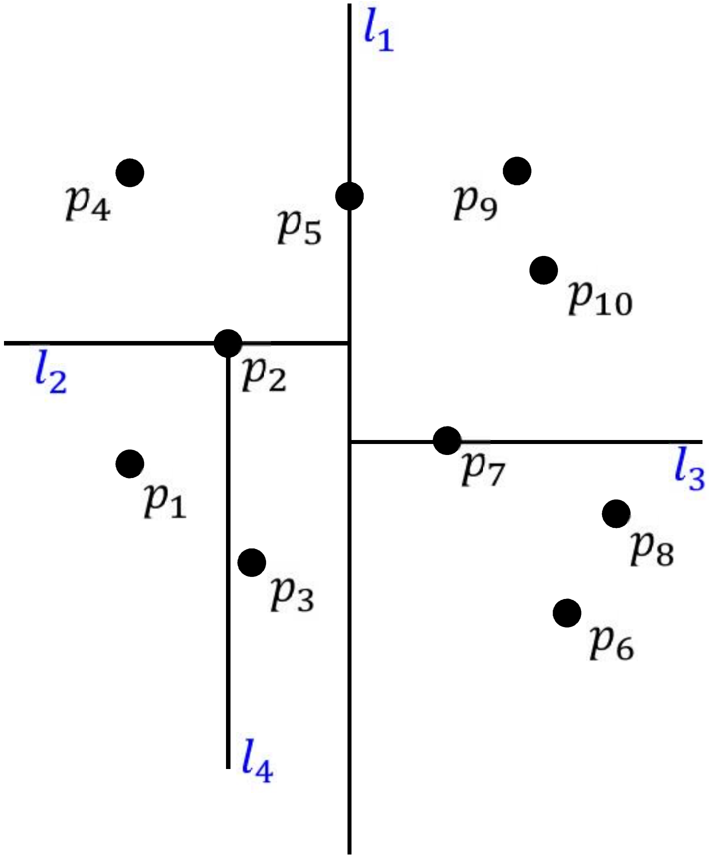
K-d Tree



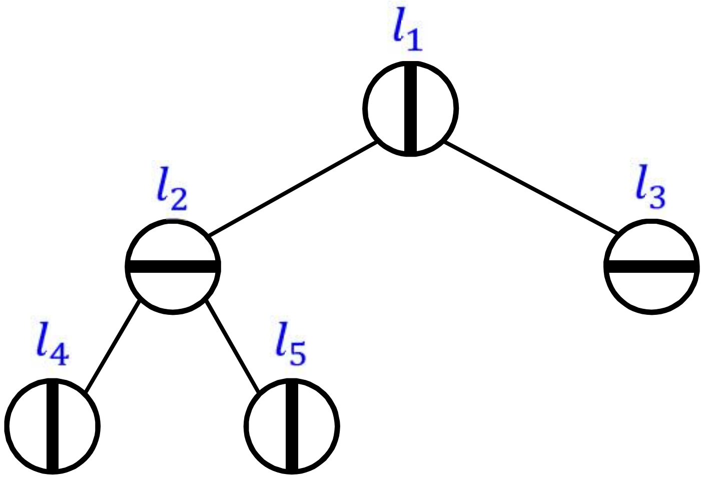


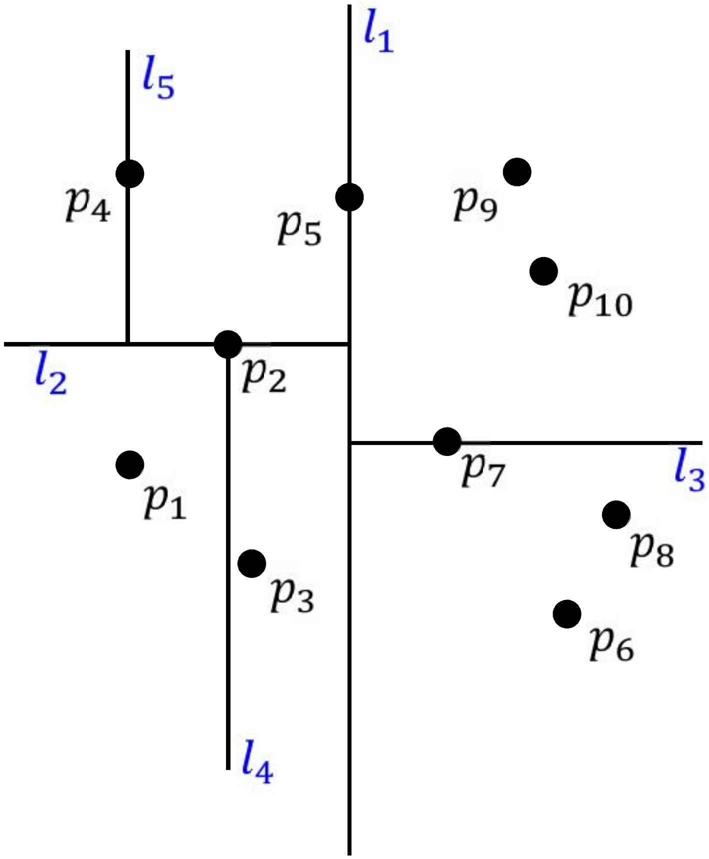
K-d Tree



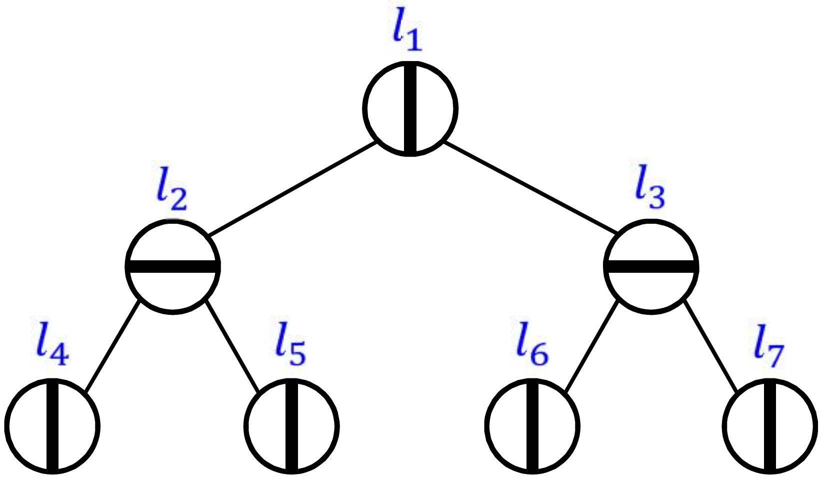


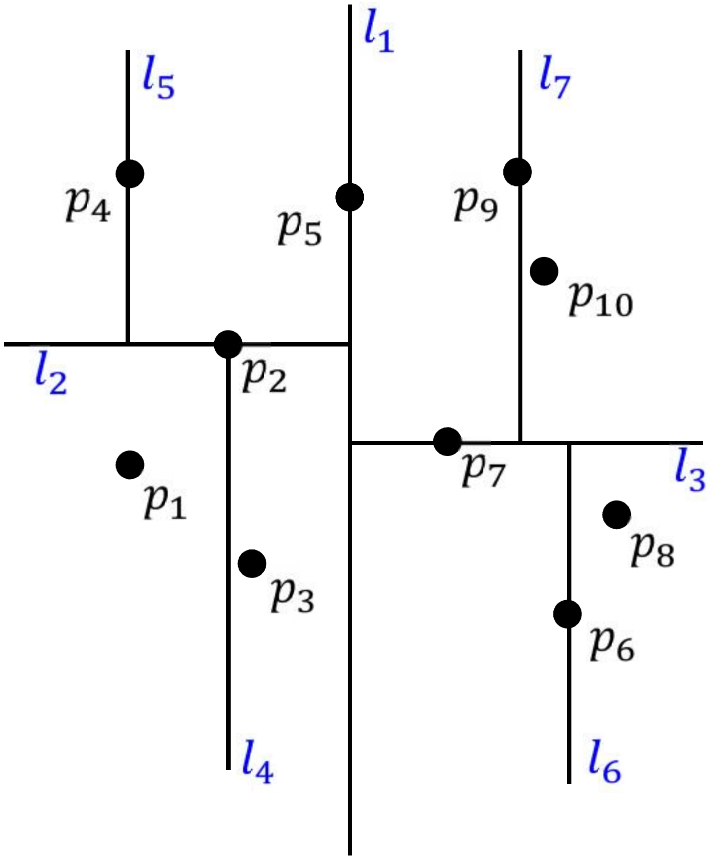
K-d Tree



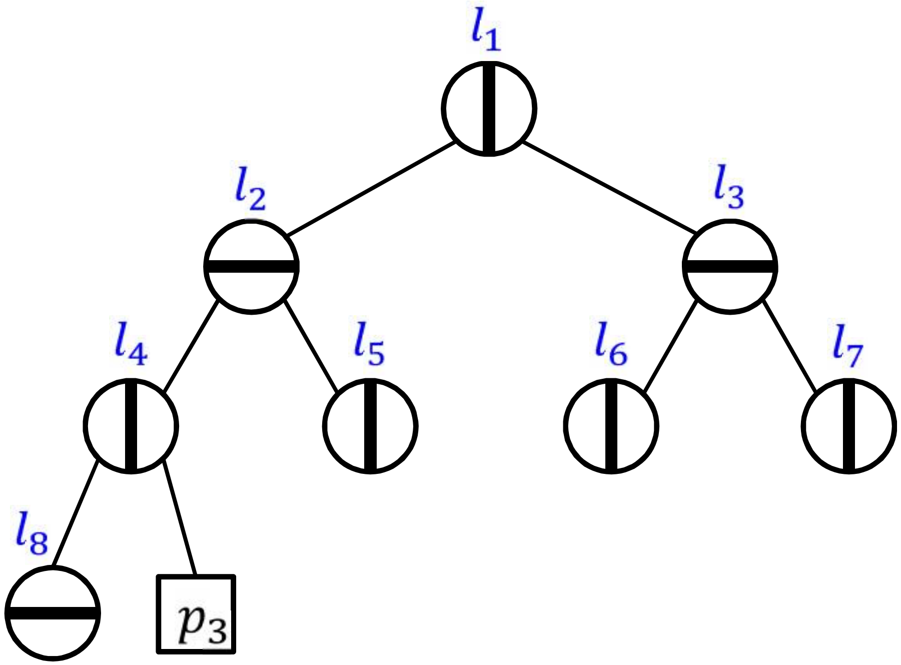


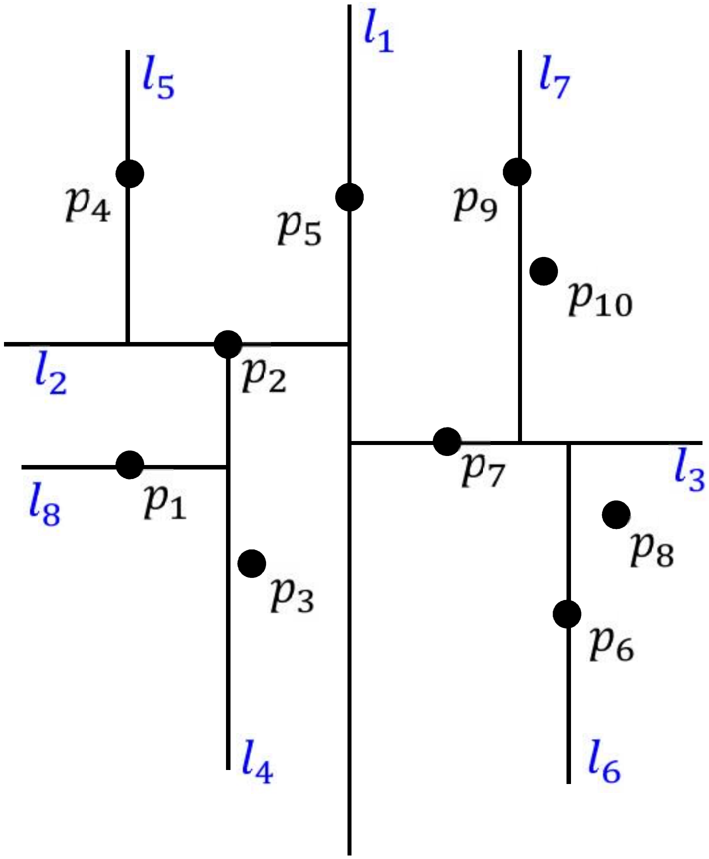
K-d Tree



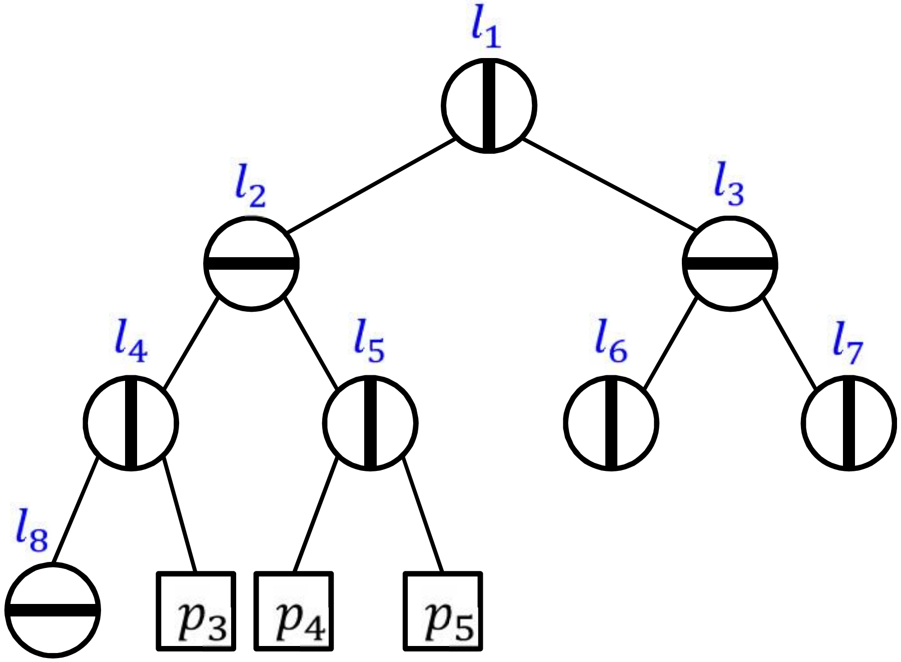


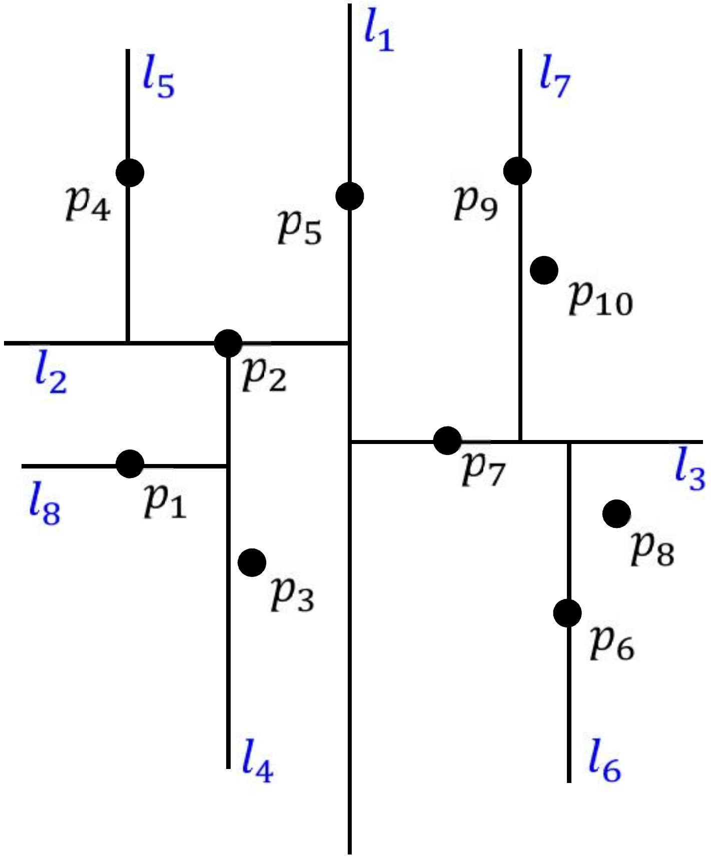
K-d Tree


l_1
l_2
l_3
l_4
l_5
l_6
l_7
l_8
p_3

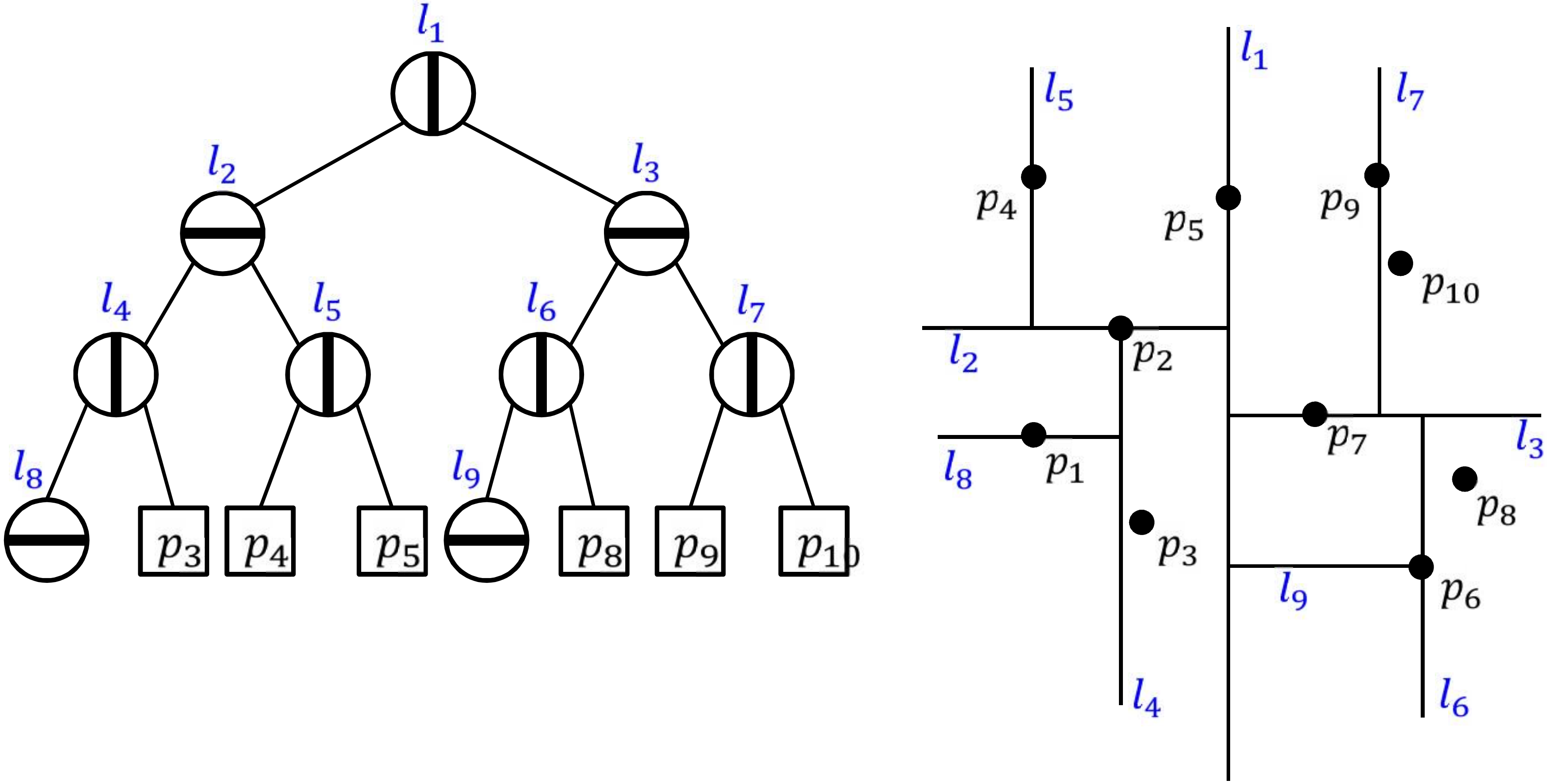

l_1
l_5
l_7
p_4
p_5
p_9
p_{10}
l_2
p_2
p_7
l_3
l_8
p_1
p_8
p_3
p_6
l_4
l_6

K-d Tree

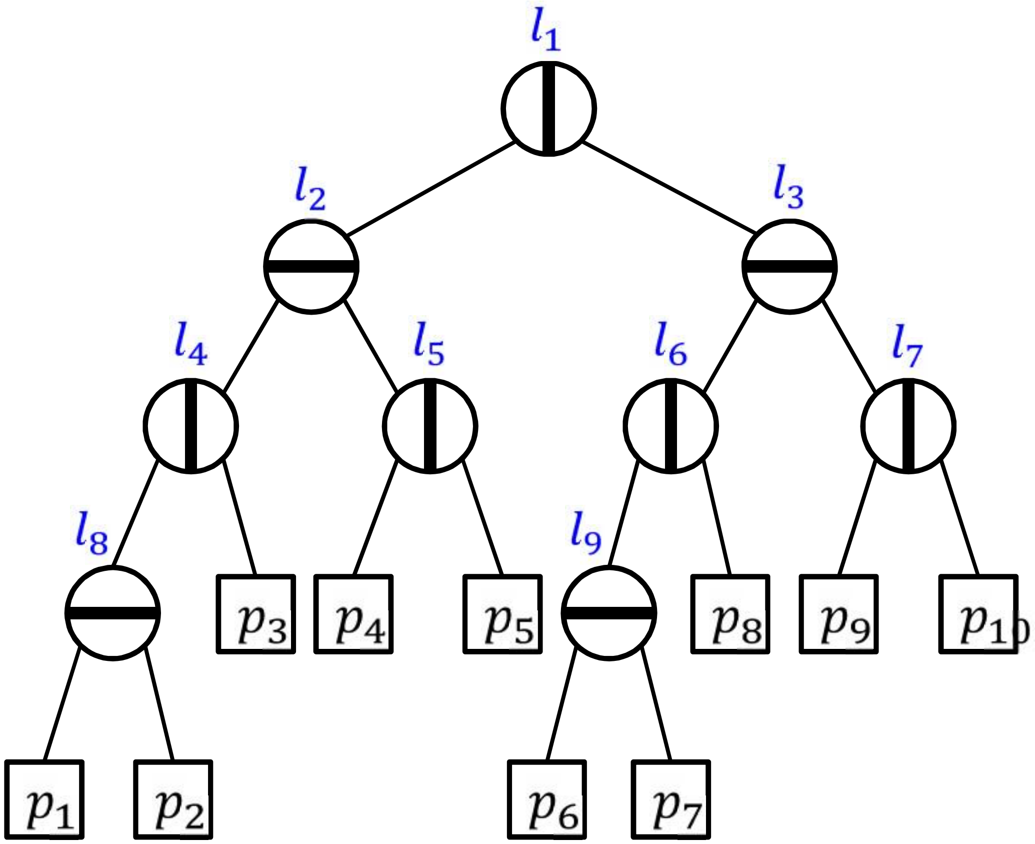

l_1
l_2
l_3
l_4
l_5
l_6
l_7
l_8
p_3
p_4
p_5


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

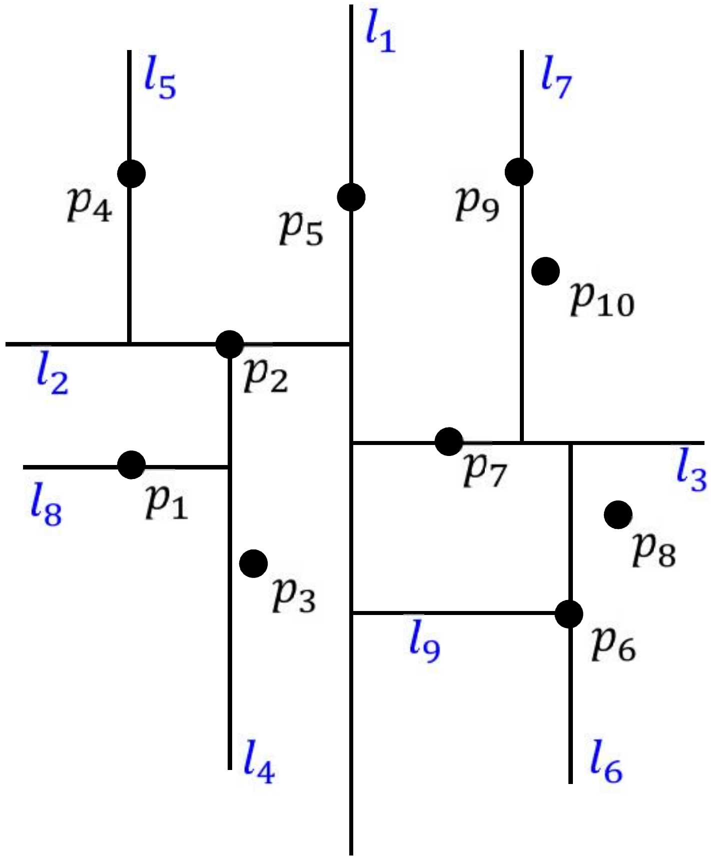
K-d Tree


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

K-d Tree

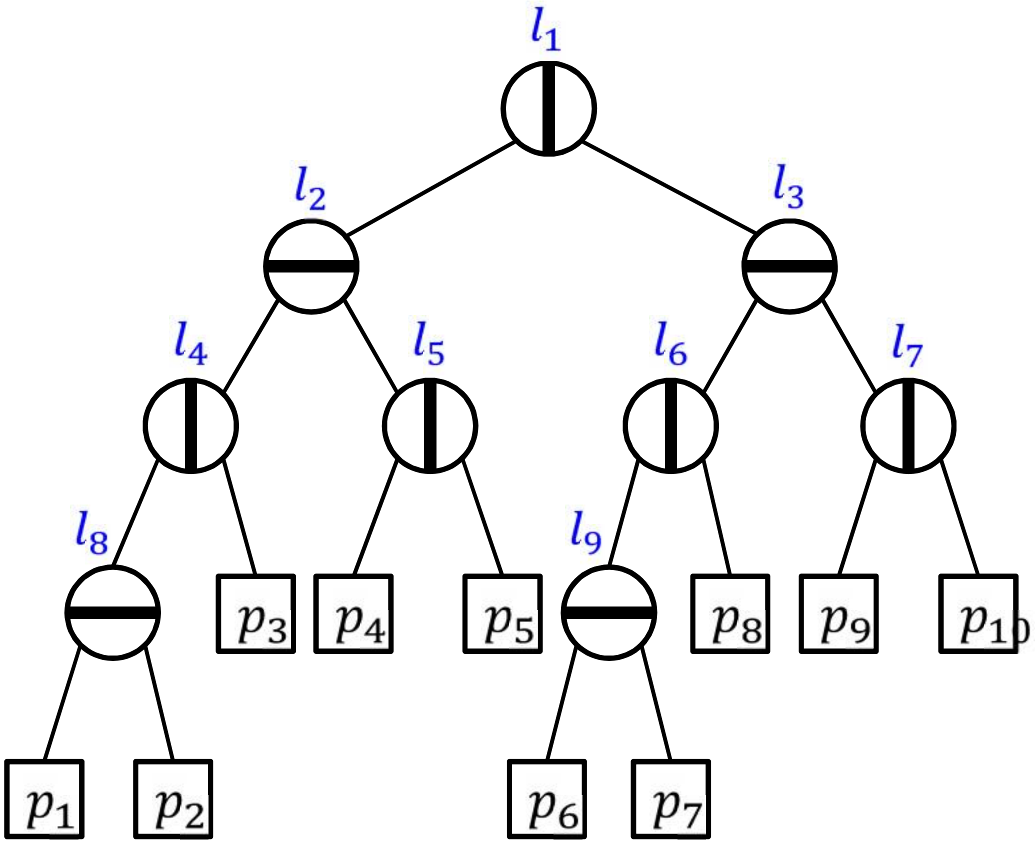


l_1
l_2
l_3
l_4
l_5
l_6
l_7
l_8
l_9
p_3
p_4
p_5
p_8
p_9
p_{10}
p_1
p_2
p_6
p_7



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

K-d Tree



- Space: O(n)

K-d Tree

- If t is a node:
  - t.val: cut value
  - t.dir: cut direction
  - t.left, t.right: child
- If t is a leaf:
  - t.pt: point

K-d Tree

- If t is a node:
  - t.val: cut value
  - t.dir: cut direction
  - t.left, t.right: child
- If t is a leaf:
  - t.pt: point

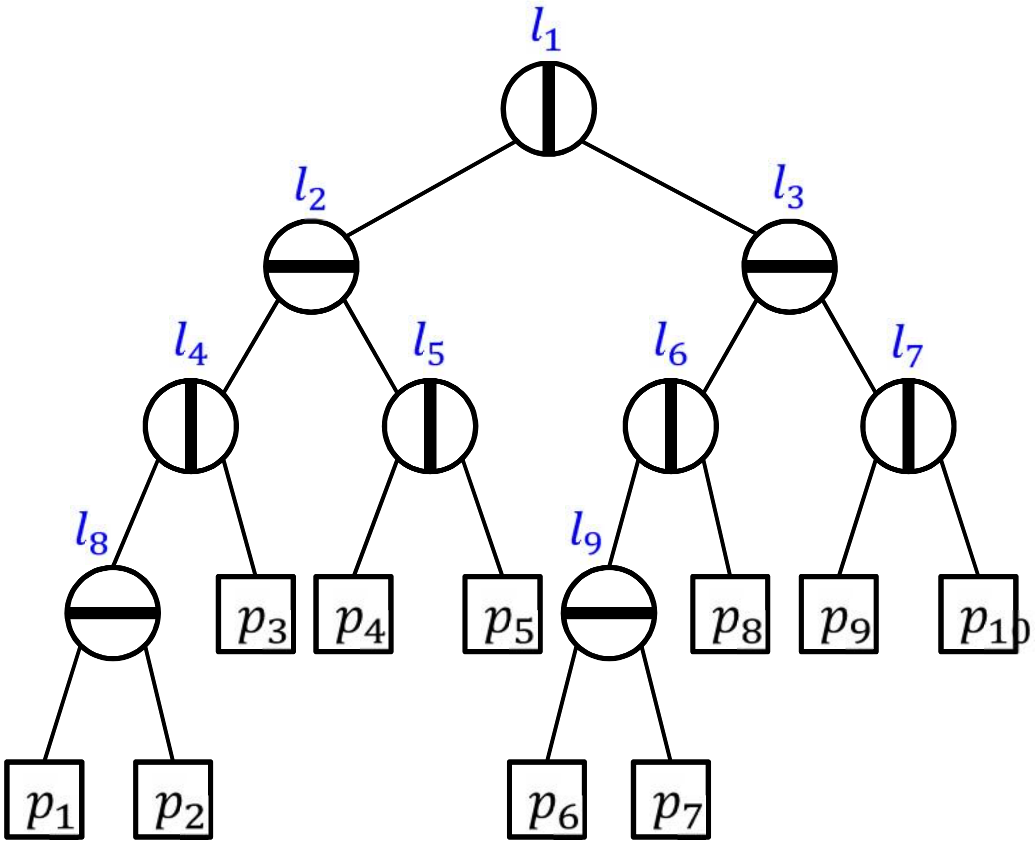
```
BuildTree (S, d)     //d: direction
1.  If $|S|=1$, return leaf t where
    1.  t.pt is the point of S
2.  x be median of d-th coordinates of all
    points in S
3.  L (R) be subset of S whose d-th
    coordinates are no greater than
    (greater than) x
4.  Return node t where
    1.  t.val = x
    2.  t.dir = d
    3.  t.left = BuildTree (L, 3-d)
    4.  t.right = BuildTree (R, 3-d)
```

$O(n)$

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

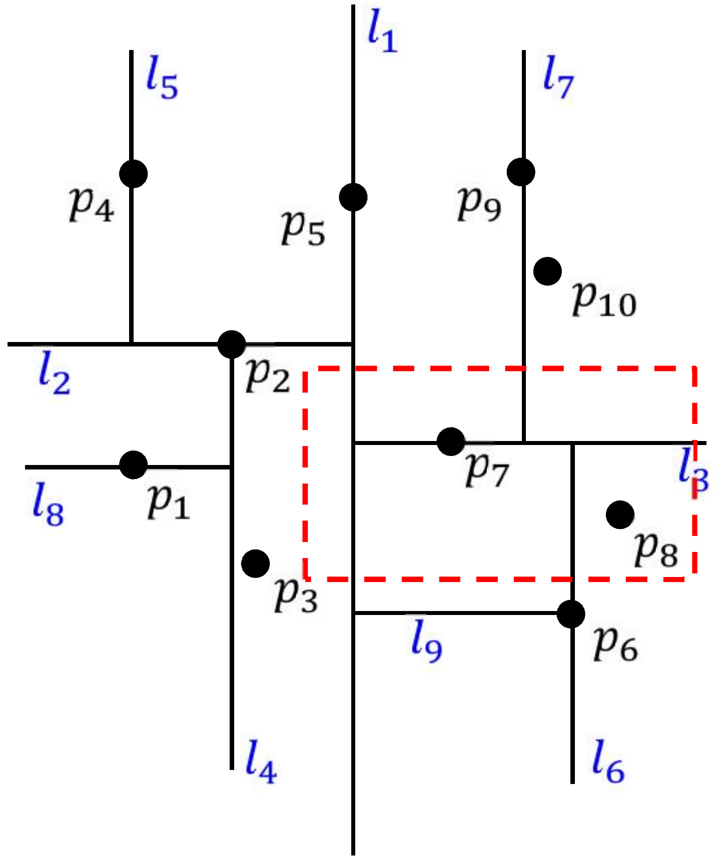
$2T(n/2)$

K-d Tree

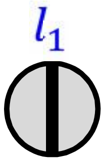


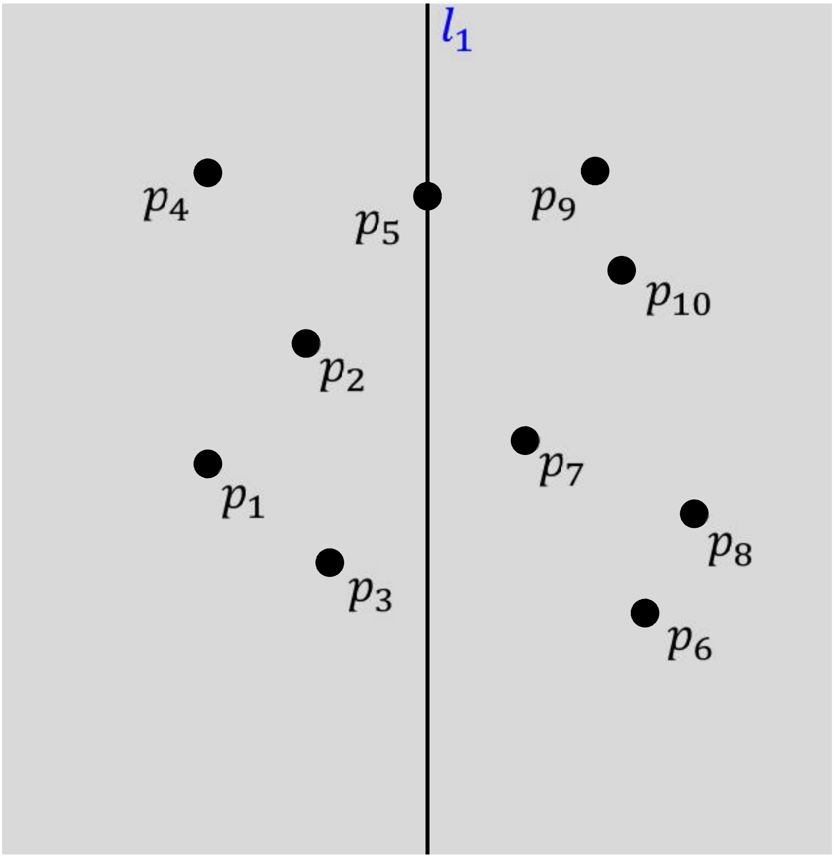
- Space: O(n)
- Build time: O(n Log n)

Query a K-d tree

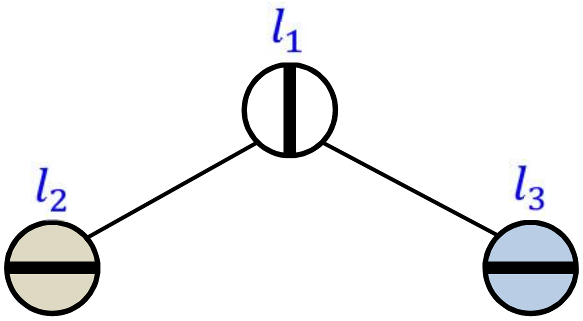


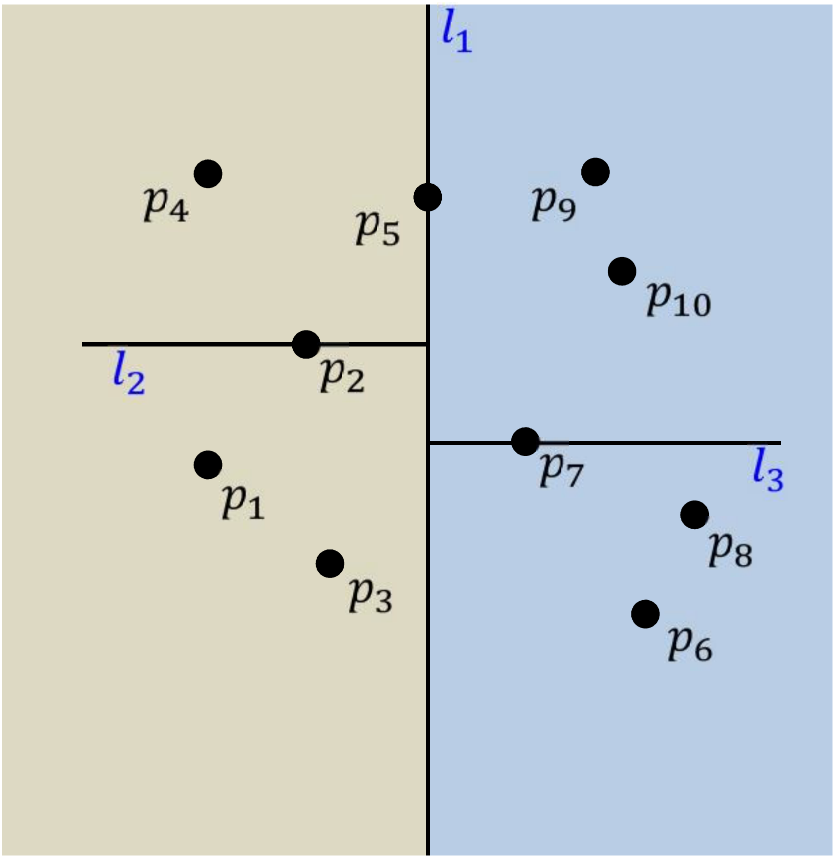
Range of a node



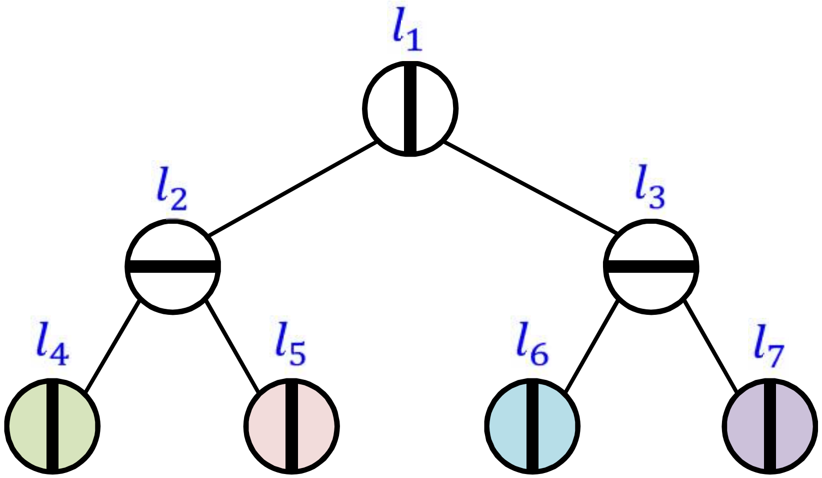


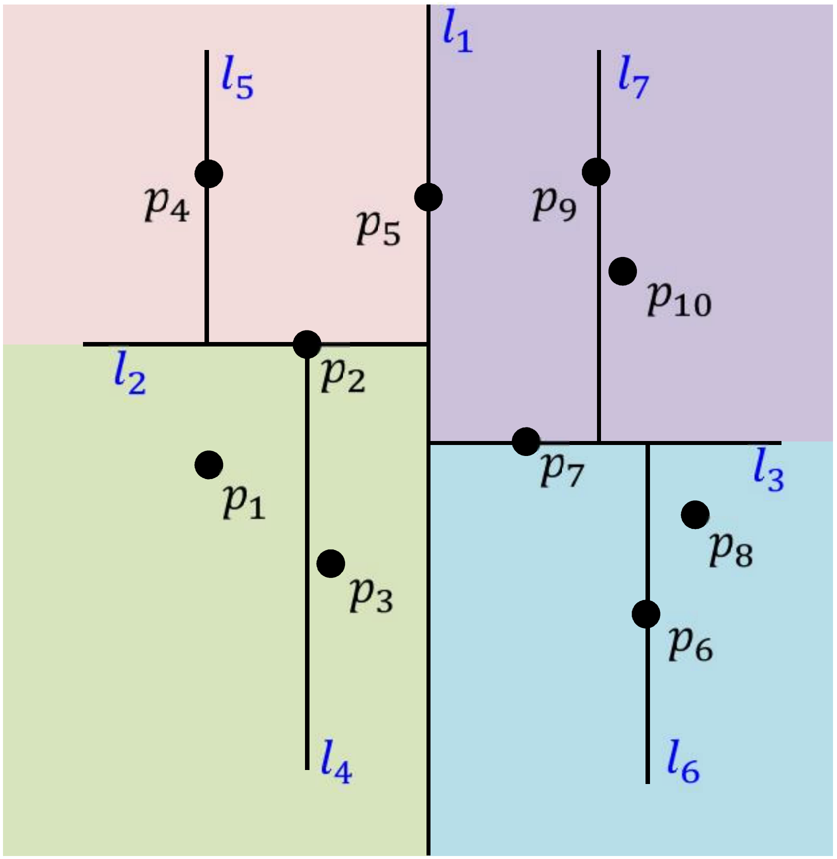
Range of a node



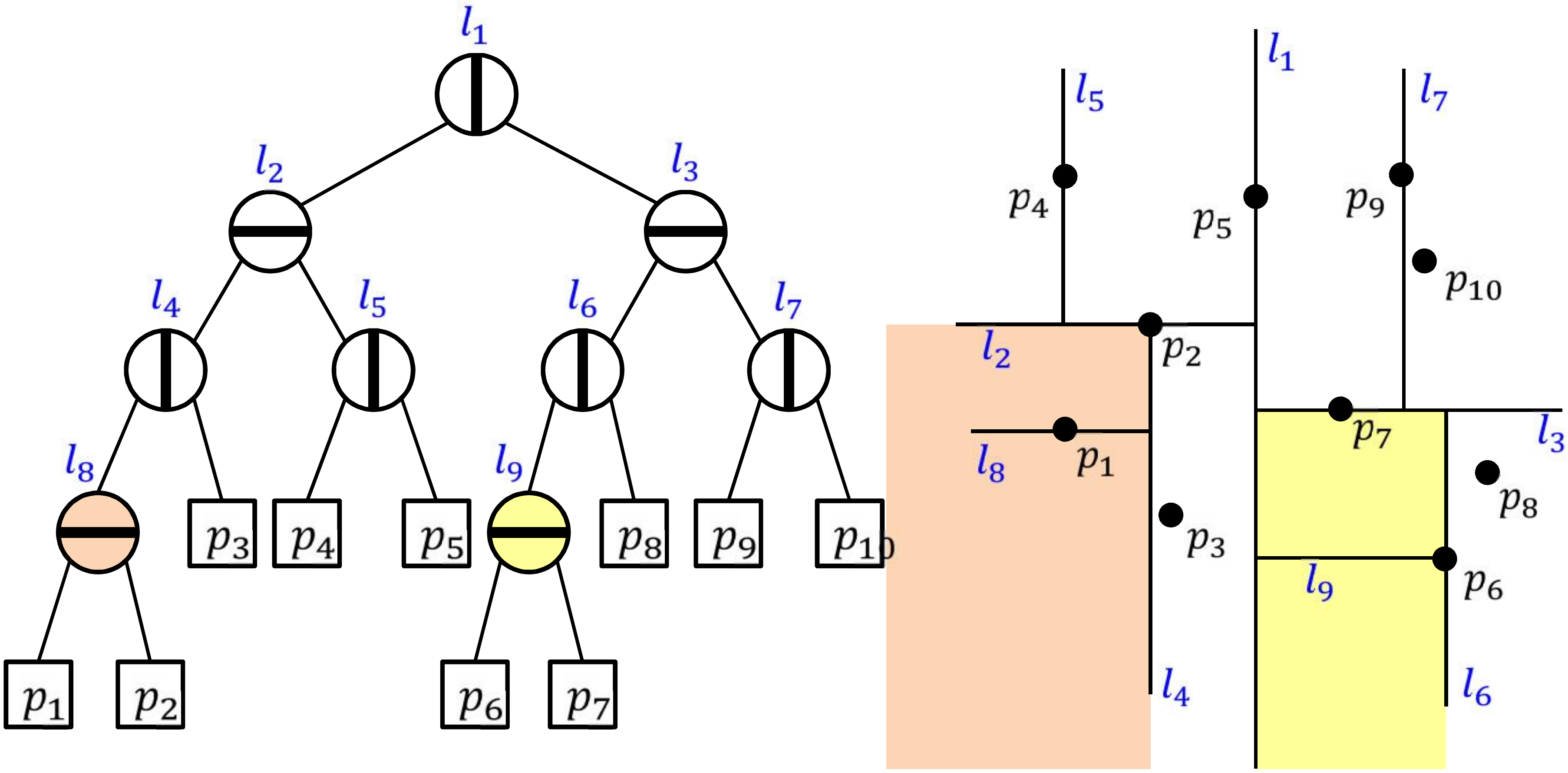


Range of a node

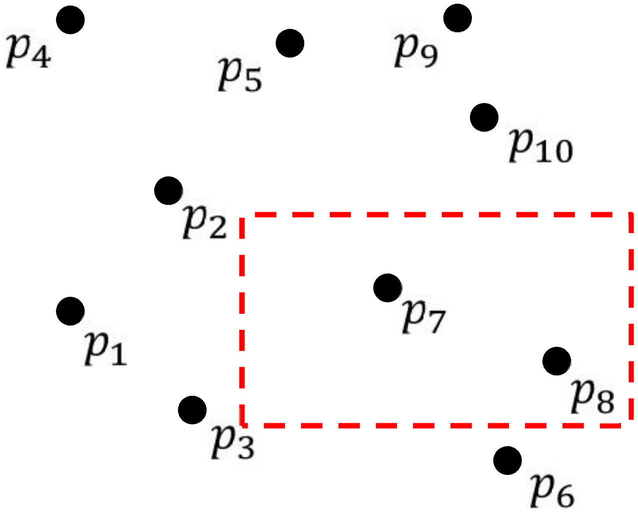




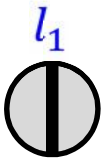
Range of a node

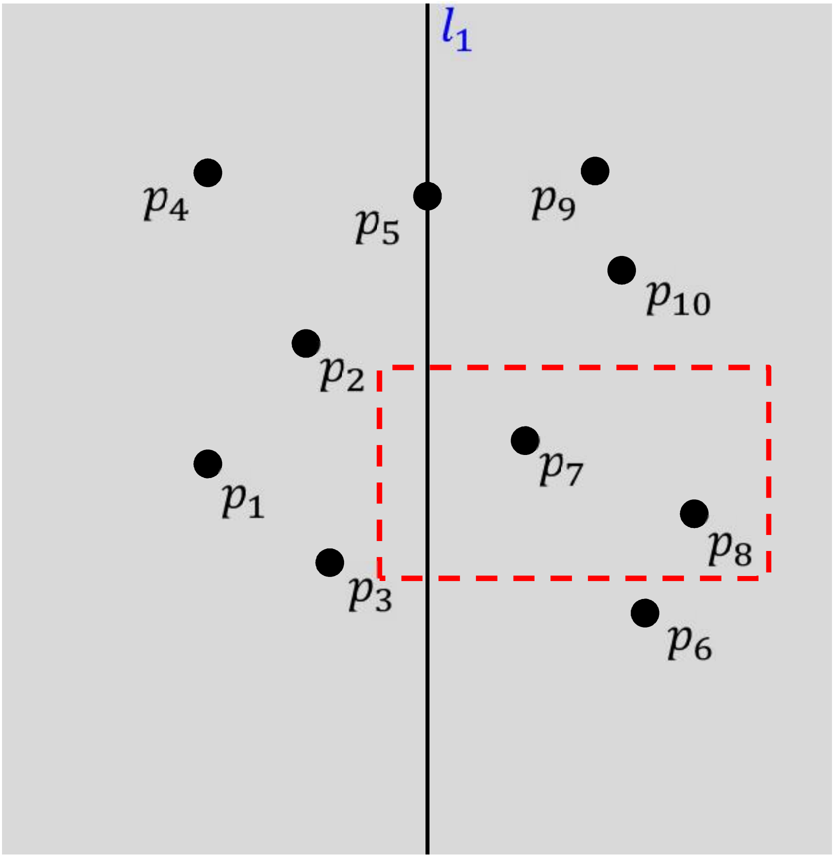


Query a K-d tree

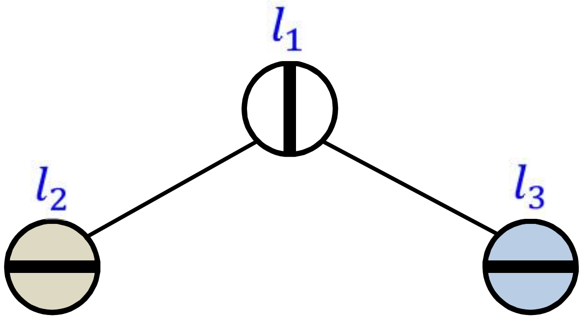


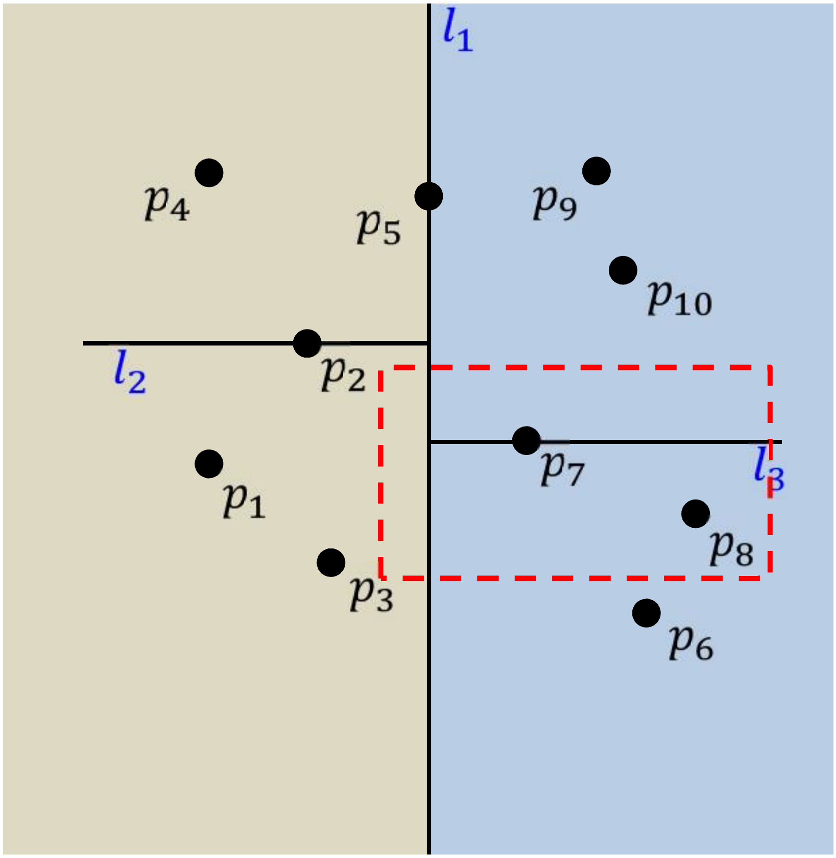
Query a K-d tree



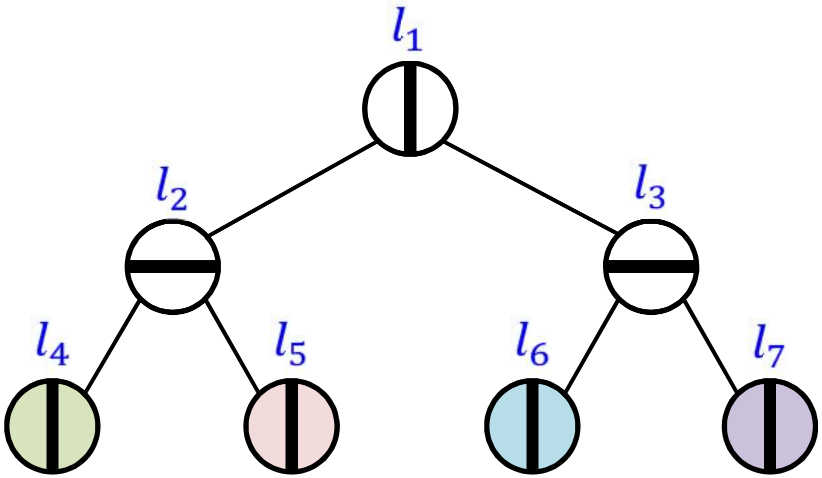


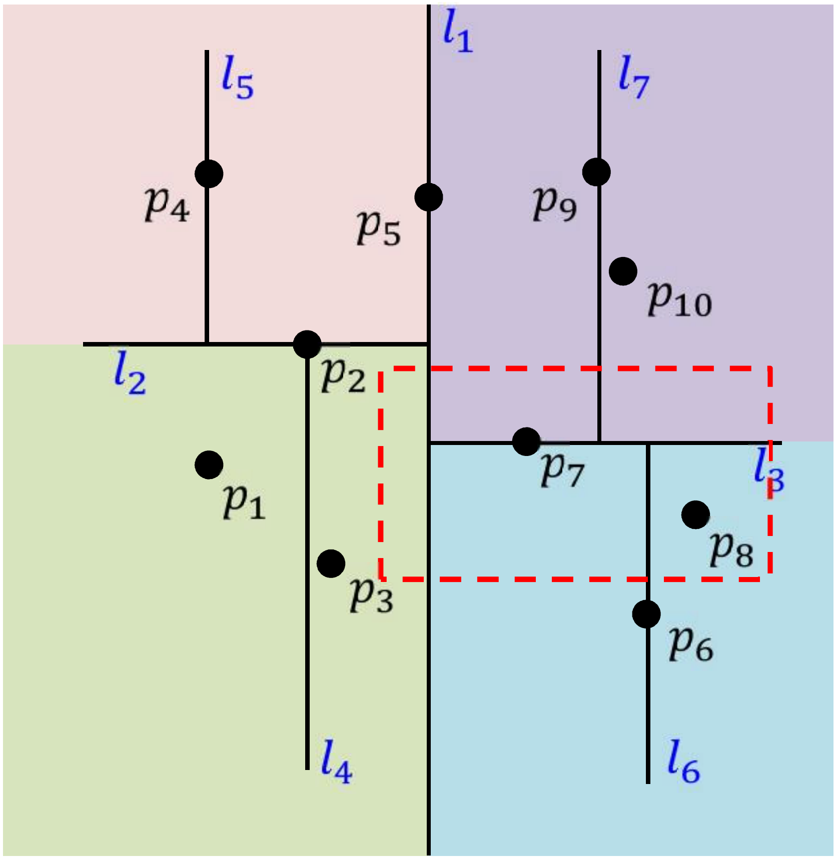
Query a K-d tree



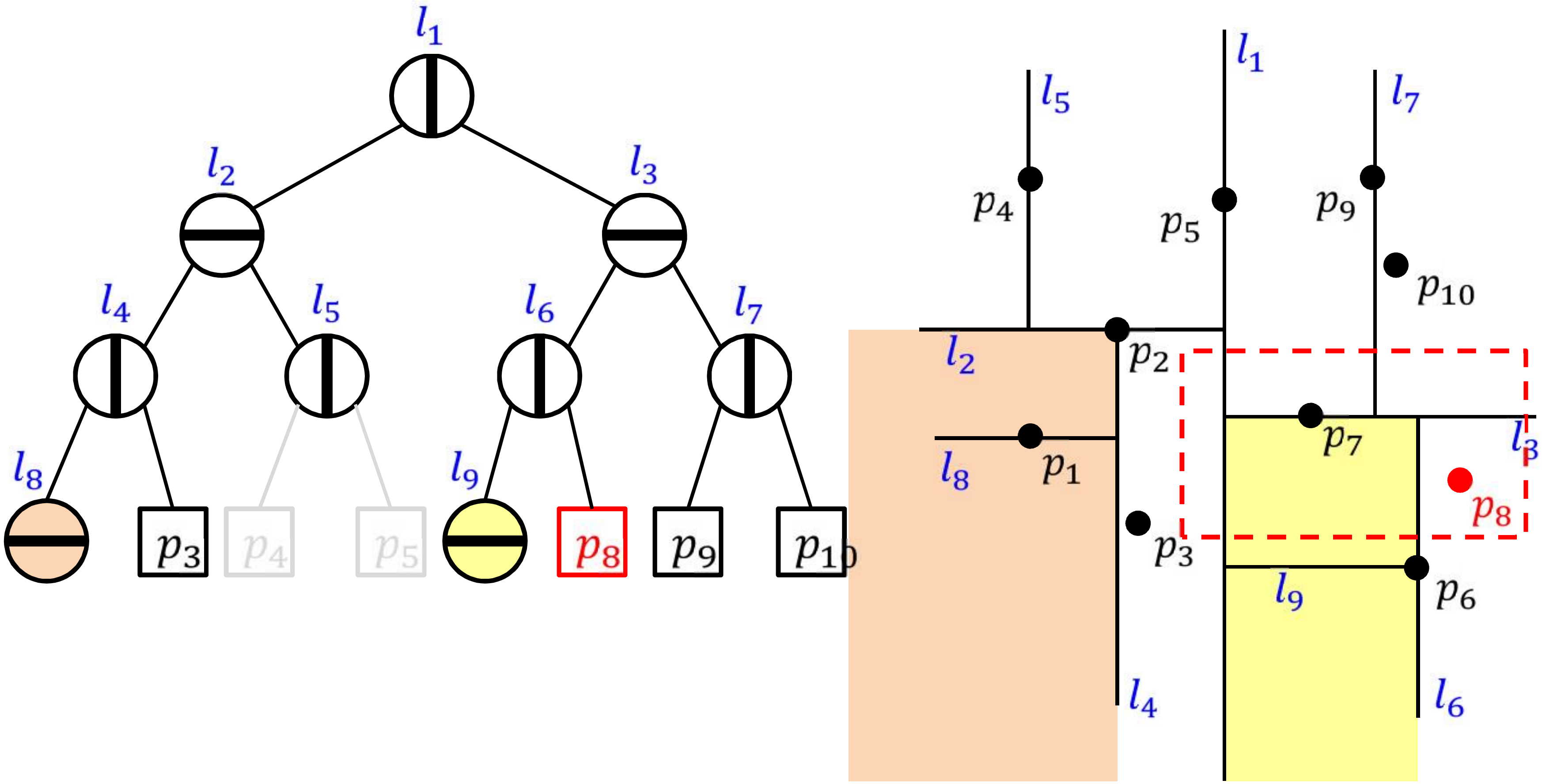


Query a K-d tree

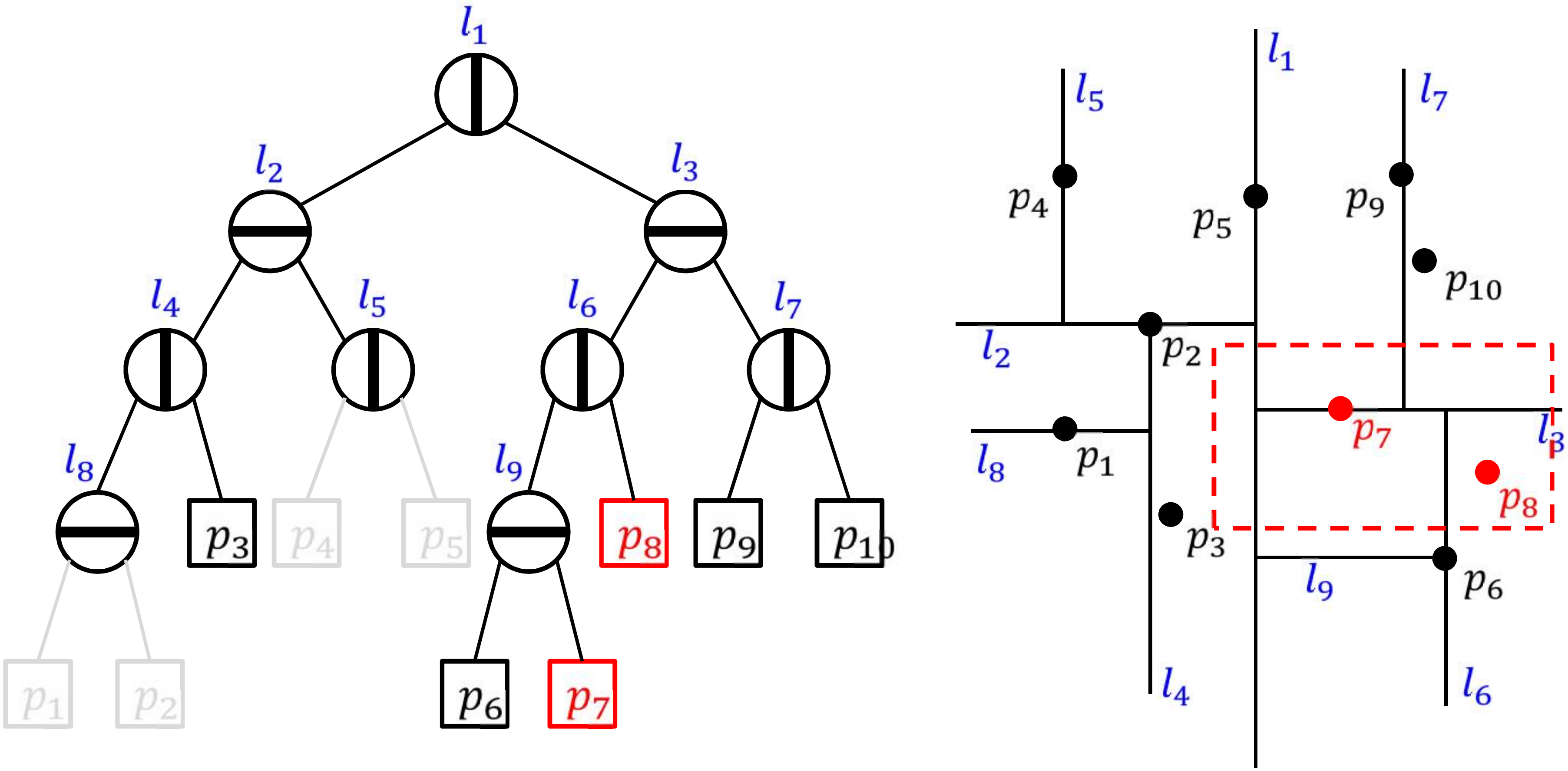




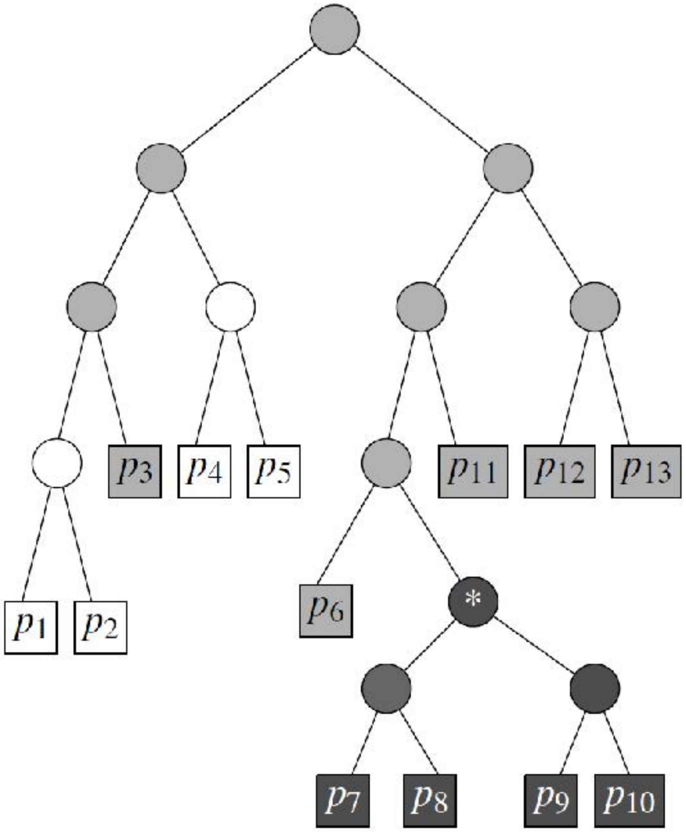
Query a K-d tree

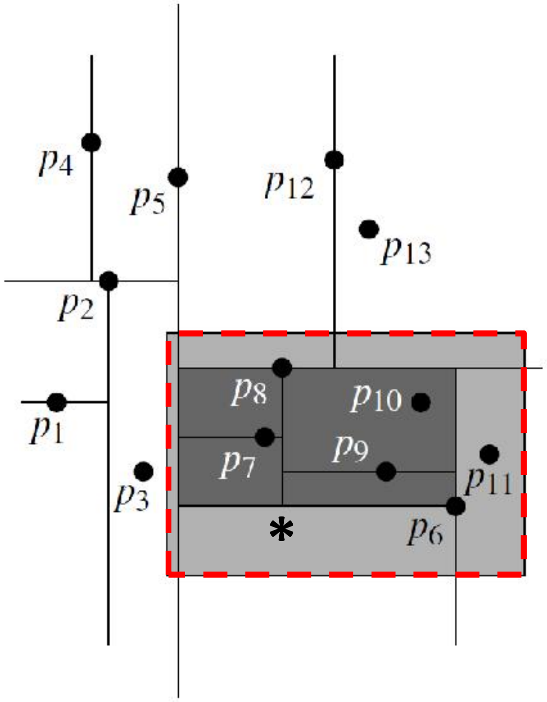


Query a K-d tree



Query a K-d tree





Range of * lies within the query range.
The whole subtree rooted at * is reported.

Query a K-d tree

- If t is a node:
  - t.val: cut value
  - t.dir: cut direction
  - t.left, t.right: child
  - t.range: range
- If t is a leaf:
  - t.pt: point

```
Query (t, r)          //r: query range
1.  If t is a leaf
    1.  If t.pt is inside r, return t.pt
    2.  Else return NULL
2.  If t.range is inside r
    1.  ReportTree (t)
3.  Else if t.range intersects r
    1.  Return Query (t.left, r)
        Query (t.right, r)
```

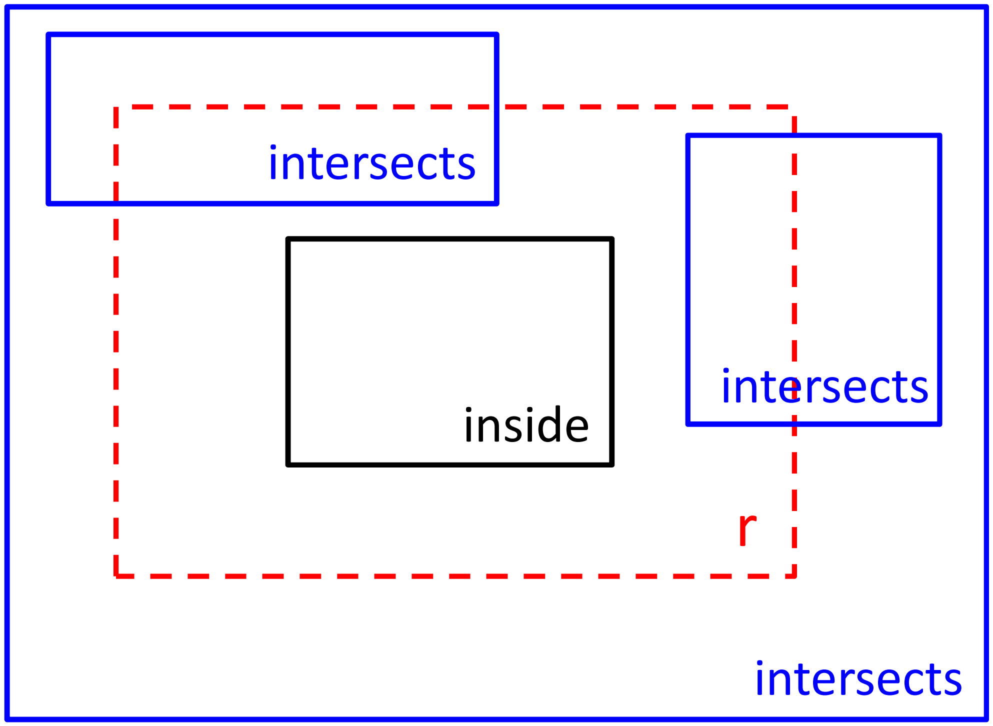
Query a K-d tree

- Complexity:
  - Total time for (1-2): O(k)
  - # calls to Query: ???

```
Query (t, r)          //r: query range
1.  If t is a leaf
    1.  If t.pt is inside r, return t.pt
    2.  Else return NULL
2.  If t.range is inside r
    1.  ReportTree (t)
3.  Else if t.range intersects r
    1.  Return Query (t.left, r)
        Query (t.right, r)
```

Query a K-d tree

- Query(t, r) is called if t's parent's range intersects (but does not lie inside) r



Such range must contain a border edge of r

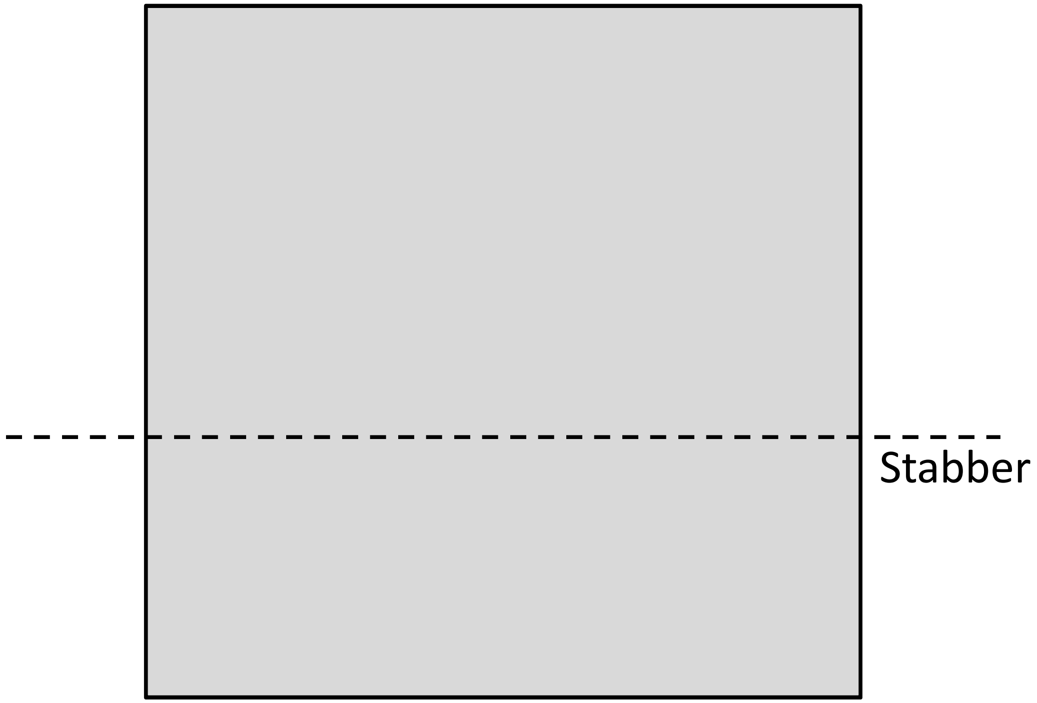
Query a K-d tree

- How many nodes of a k-d tree can be “stabbed” by a line (i.e., the line passes through the node’s range)?

Query a K-d tree

- How many nodes of a k-d tree can be "stabbed" by a line (i.e., the line passes through the node's range)?

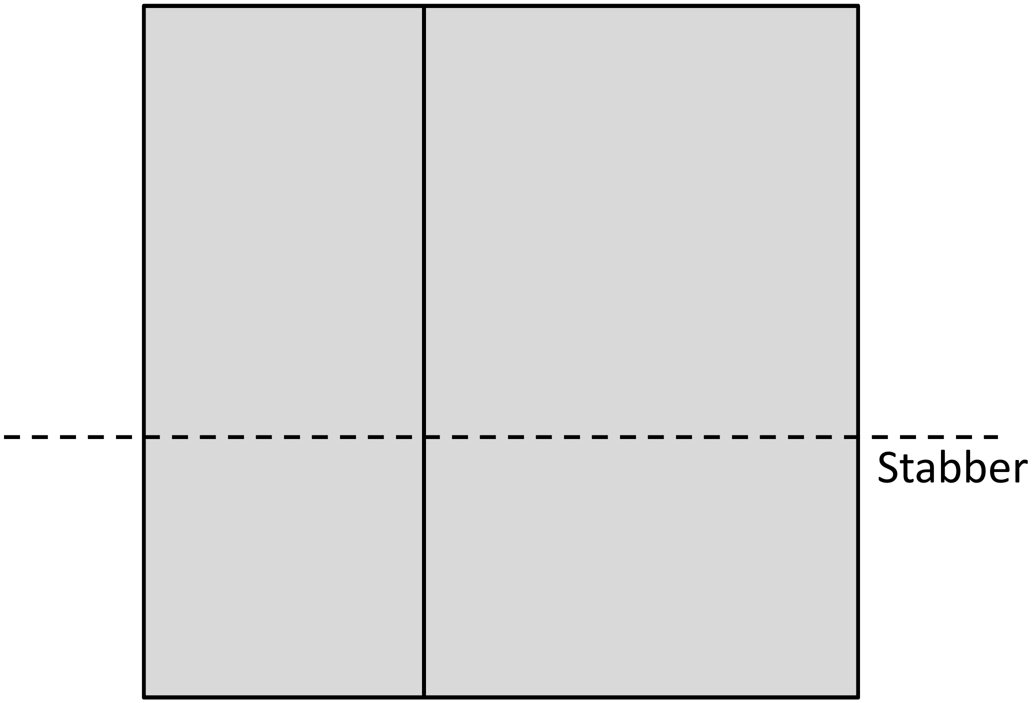
Tree level	# stabbed nodes
1	1



Query a K-d tree

- How many nodes of a k-d tree can be “stabbed” by a line (i.e., the line passes through the node’s range)?

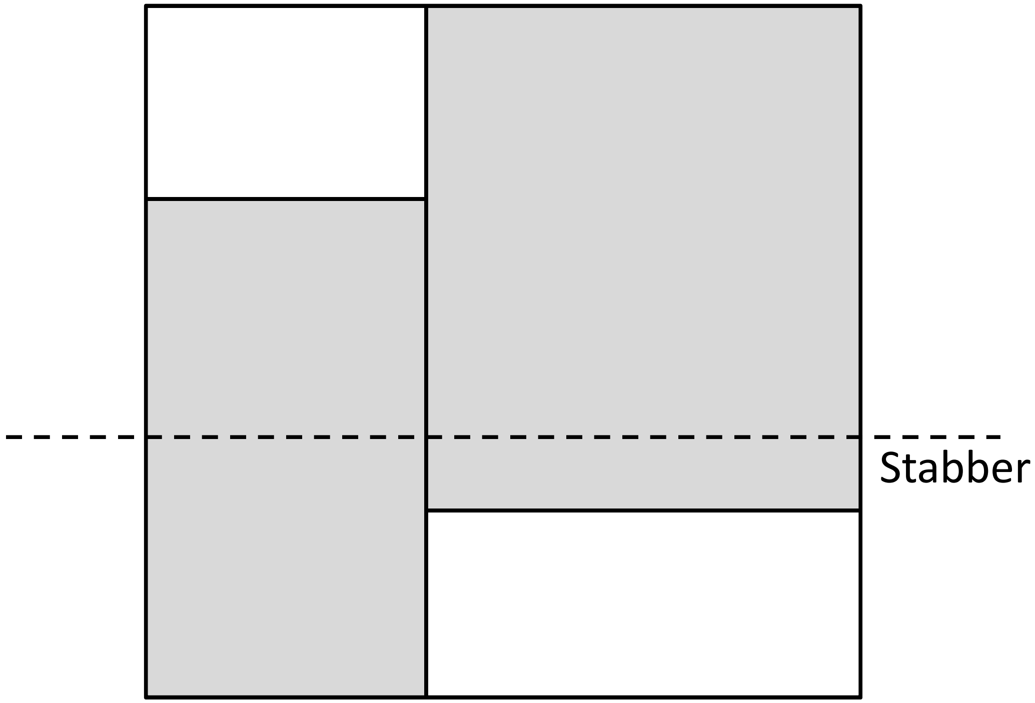
Tree level	# stabbed nodes
1	1
2	2



Query a K-d tree

- How many nodes of a k-d tree can be “stabbed” by a line (i.e., the line passes through the node’s range)?

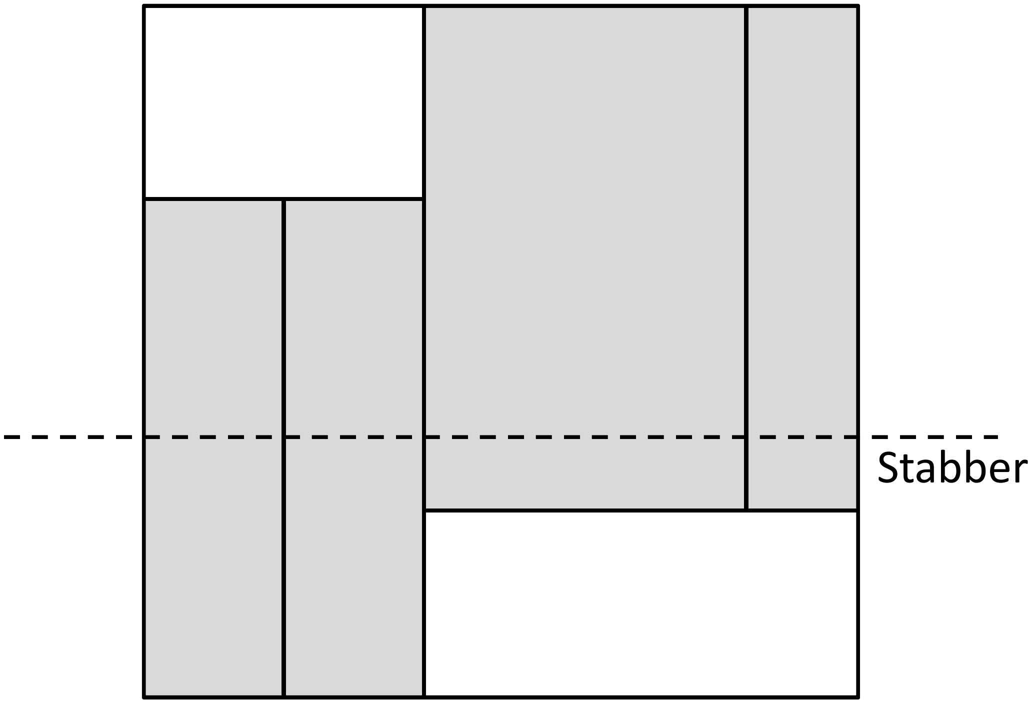
Tree level	# stabbed nodes
1	1
2	2
3	2



Query a K-d tree

- How many nodes of a k-d tree can be "stabbed" by a line (i.e., the line passes through the node's range)?

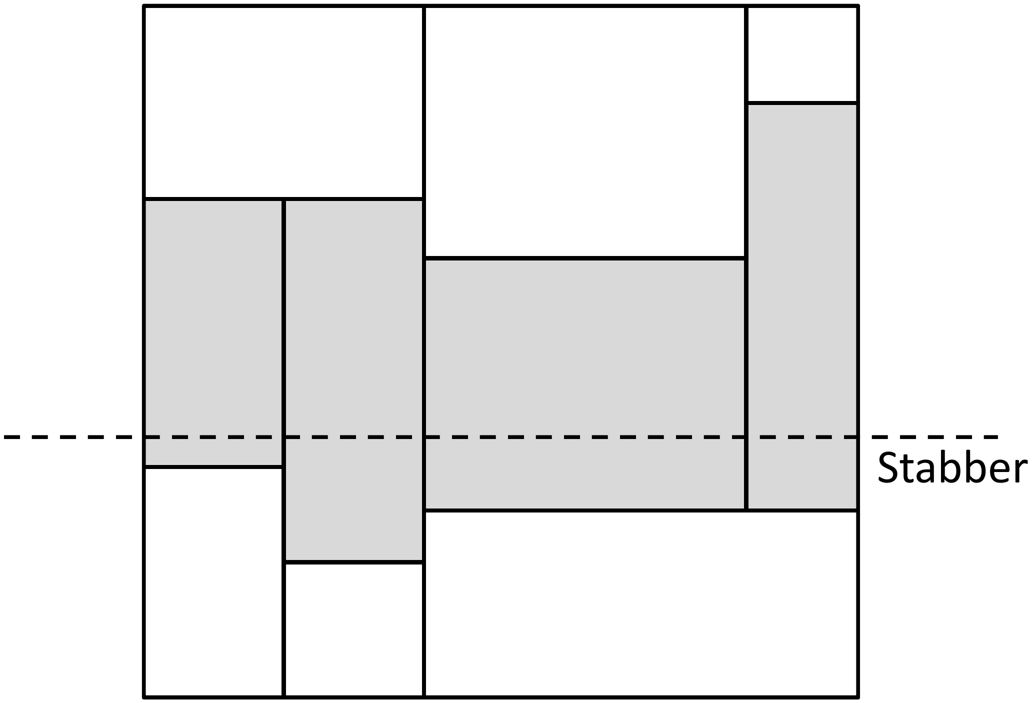
Tree level	# stabbed nodes
1	1
2	2
3	2
4	4



Query a K-d tree

- How many nodes of a k-d tree can be “stabbed” by a line (i.e., the line passes through the node’s range)?

Tree level	# stabbed nodes
1	1
2	2
3	2
4	4
5	4
k	$2^{\lfloor k/2 \rfloor}$



Query a K-d tree

- How many nodes of a k-d tree can be "stabbed" by a line (i.e., the line passes through the node's range)?
  - The root is stabbed.
  - All stabbed nodes form a binary tree of depth $\frac{1}{2}\log n$.
  - Hence total number is O($2^{\frac{1}{2}\log n} = \sqrt{n}$).

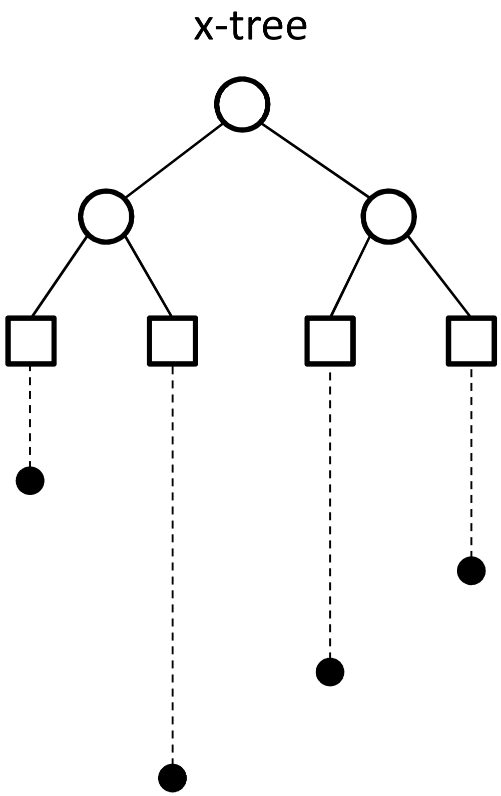
Query a K-d tree

- Complexity:
  - Total time for (1-2): O(k)
  - # calls to Query:$O(\sqrt{n})$
  - Overall:
    - Report: $O(\sqrt{n} + k)$
    - Count: $O(\sqrt{n})$

```
Query (t, r)          //r: query range
1.  If t is a leaf
    1.  If t.pt is inside r, return t.pt
    2.  Else return NULL
2.  If t.range is inside r
    1.  ReportTree (t)
3.  Else if t.range intersects r
    1.  Return Query (t.left, r) □
        Query (t.right, r)
```

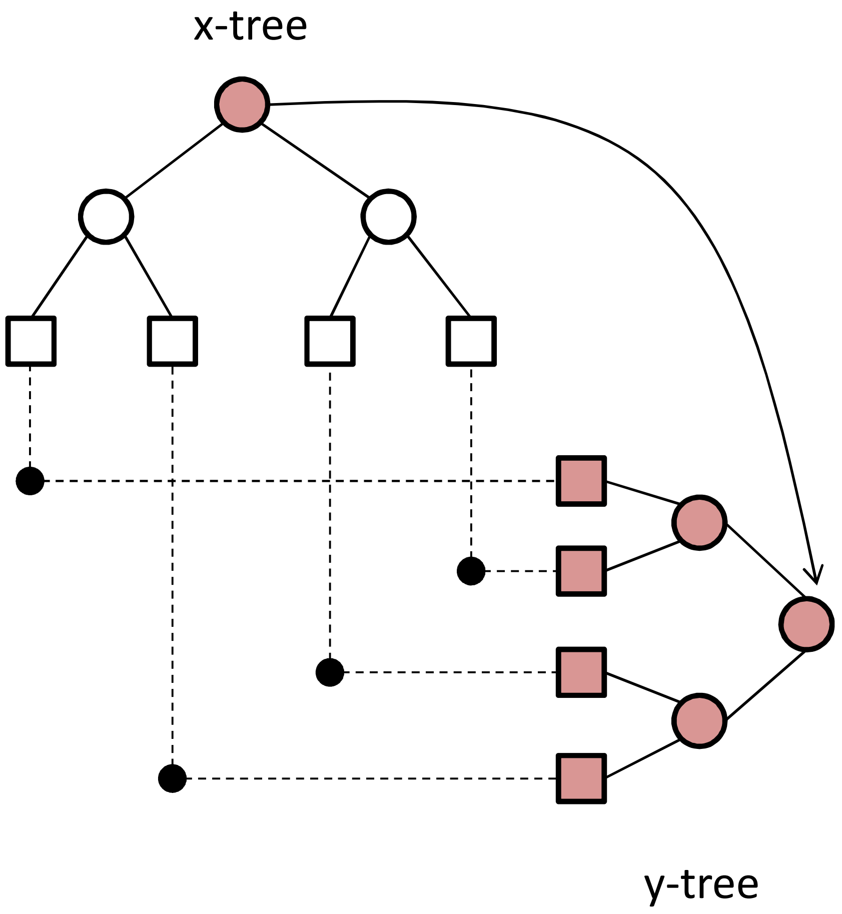
Range Tree

- One binary tree in X (x-tree)



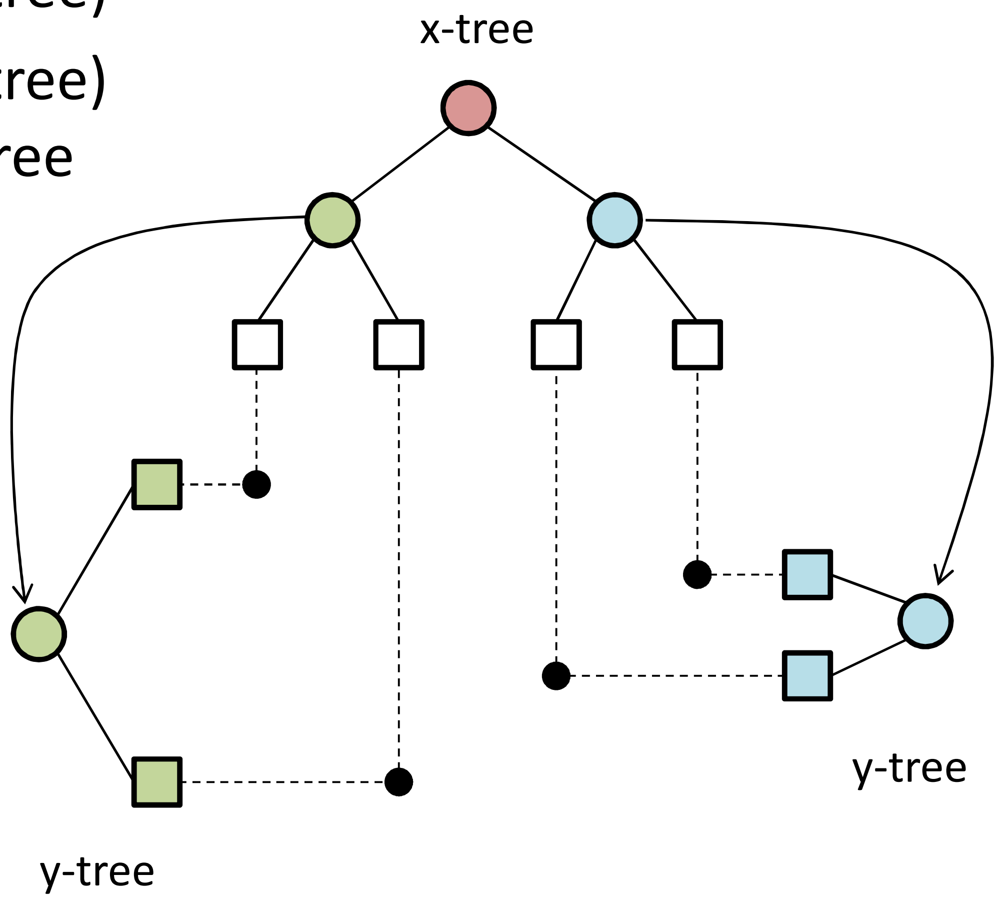
Range Tree

- One binary tree in X (x-tree)
- One binary tree in Y (y-tree) for each node in the x-tree



Range Tree

- One binary tree in X (x-tree)
- One binary tree in Y (y-tree) for each node in the x-tree

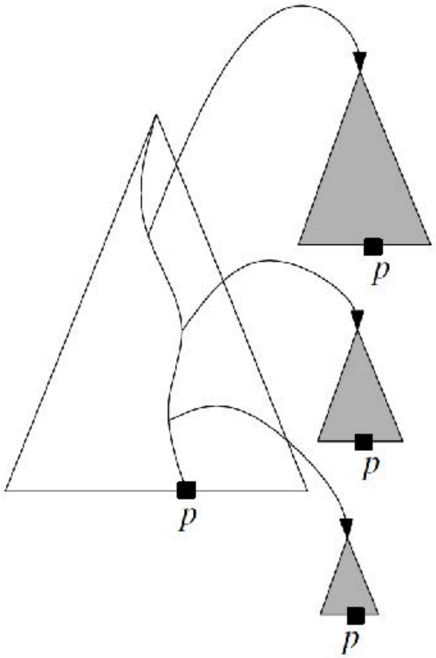


Range Tree

- Space complexity:
  - Size of each tree (x- or y-) is linear to # of leaves
  - Let T_i be # of trees of which p_i is a leaf, total space is

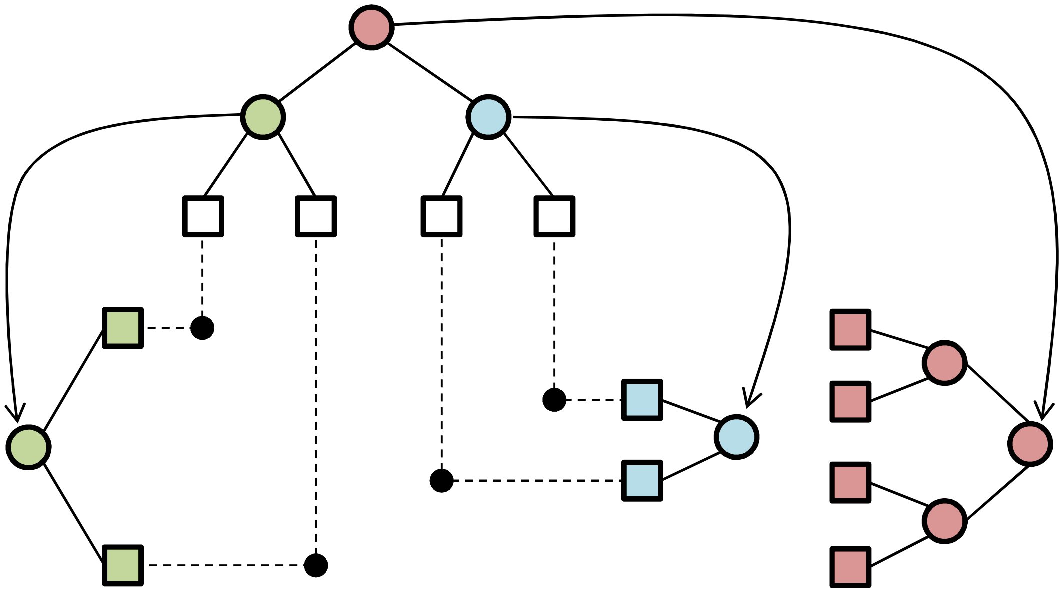
$$O(\sum_{i=1}^{n} T_i)$$

  - T_i = $O(\log\ n)$
  - Total space is $O(n\ \log\ n)$



Range Tree

How to build it?



Range Tree

- If t is a node of x-tree:
  - t.val: cut value
  - t.left, t.right: child
  - t.ytree: y-tree
- If t is a leaf of x-tree:
  - t.pt: point
  - t.ytree: a y-tree with a single point

```
BuildXTree (S)      //S: point set
1.  If |S|=1, return leaf t where
    1.  t.pt and t.ytree are the point of S
2.  x be median of X coordinates of all
    points in S
3.  L (R) be subset of S whose X
    coordinates are no greater than
    (greater than) x
4.  Return node t where
    1.  t.val = x
    2.  t.left = BuildXTree (L)
    3.  t.right = BuildXTree (R)
    4.  t.ytree = MergeYTree (t.left.ytree,
        t.right.ytree)
```

O(n) (steps 2–3)

2T(n/2) (steps 4.2–4.3)

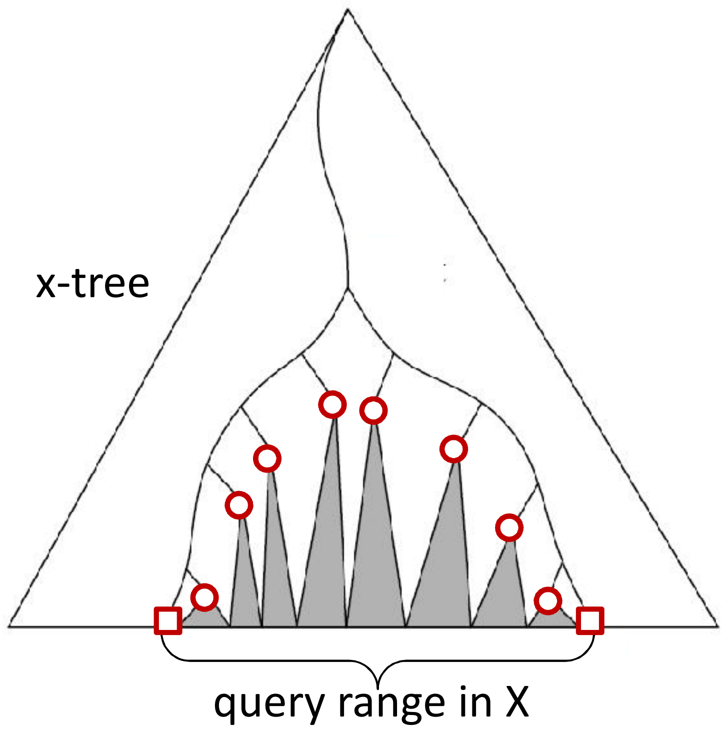
O(n) (step 4.4)

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

Range Tree

- Space complexity: O(n Log n)
- Building time: O(n Log n)

Query a range Tree



```
Query (t, rX, rY)
//rX, rY: query range in X and Y
1.  If t is a leaf
    1.  If t.pt is inside {rX,rY}, return t.pt
    2.  Else return NULL
2.  If t.range is inside rX
  ○ 1.  QueryY (t.ytree, rY)
3.  Else if t.range intersects rX
    1.  Return Query (t.left, rX, rY) □
        Query (t.right, rX, rY)
```

1D range query

Complexity of QueryY(): O(Log n_t + k_t)
# Query() calls: O (Log n)
Total complexity: $O(log^2 n + k)$

Can be improved to O(log n + k)
(using *fractional cascading*, see book/note)