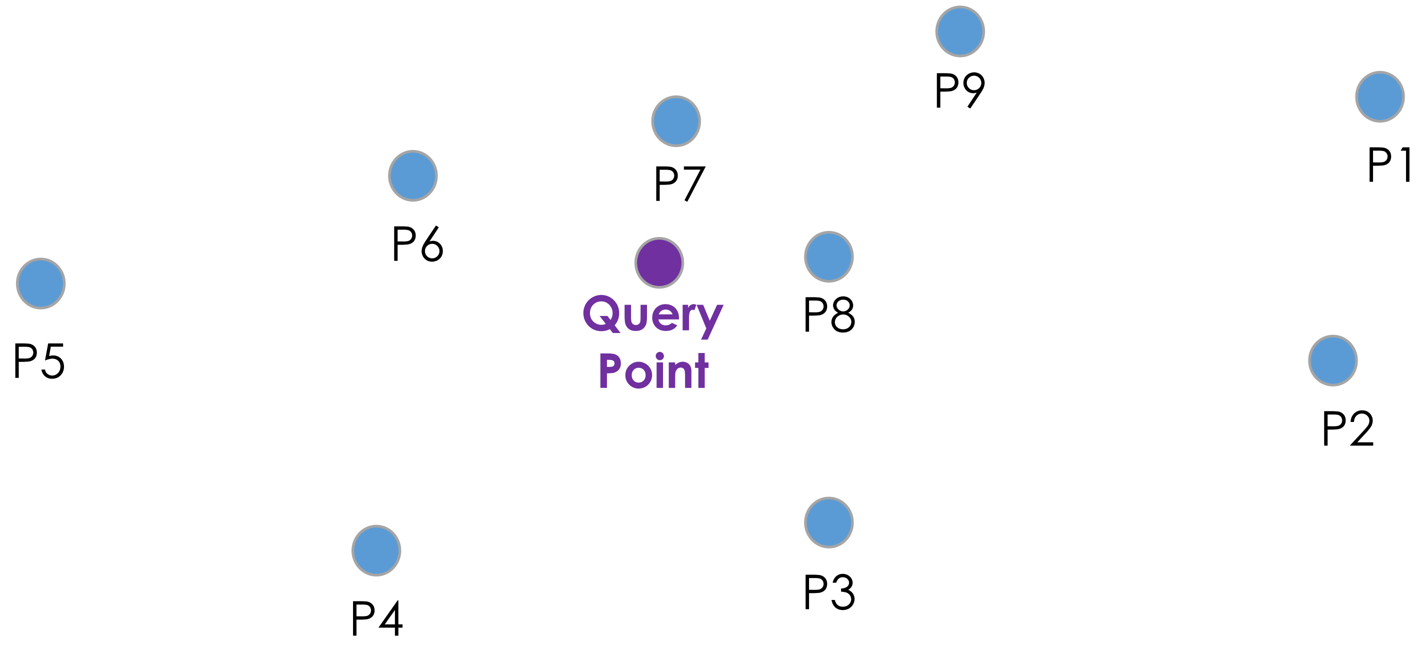
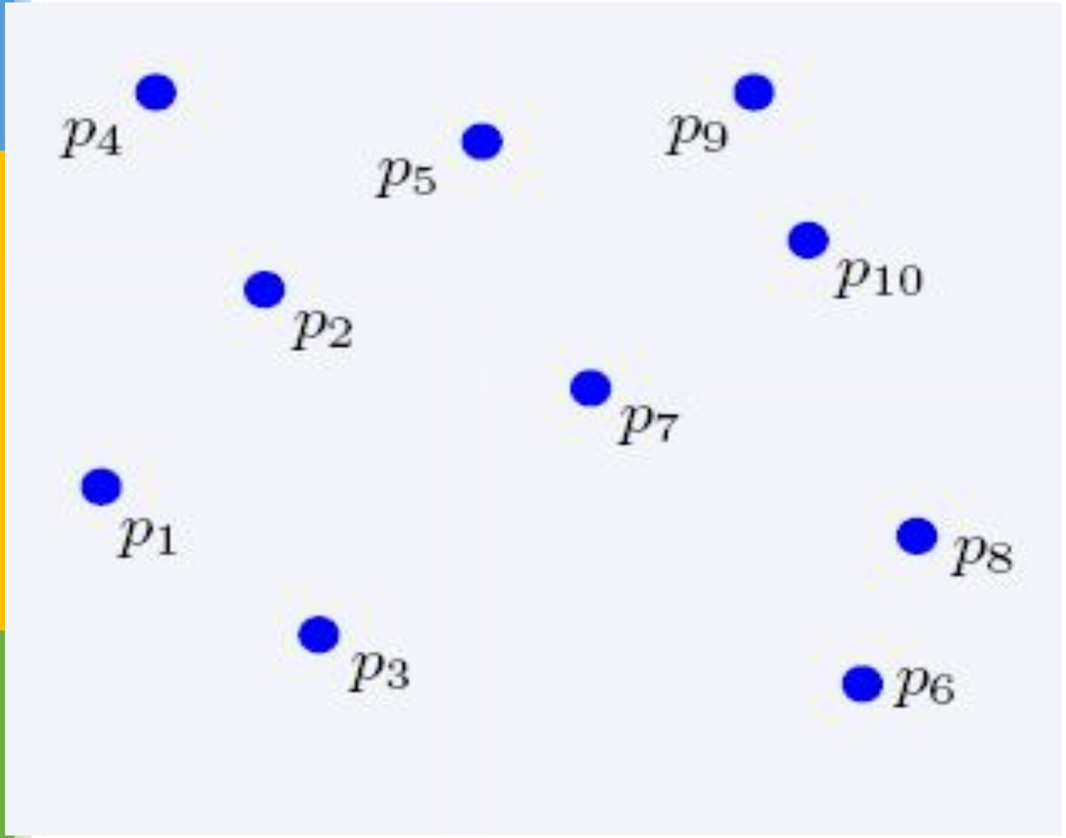
Geometric Data Structures

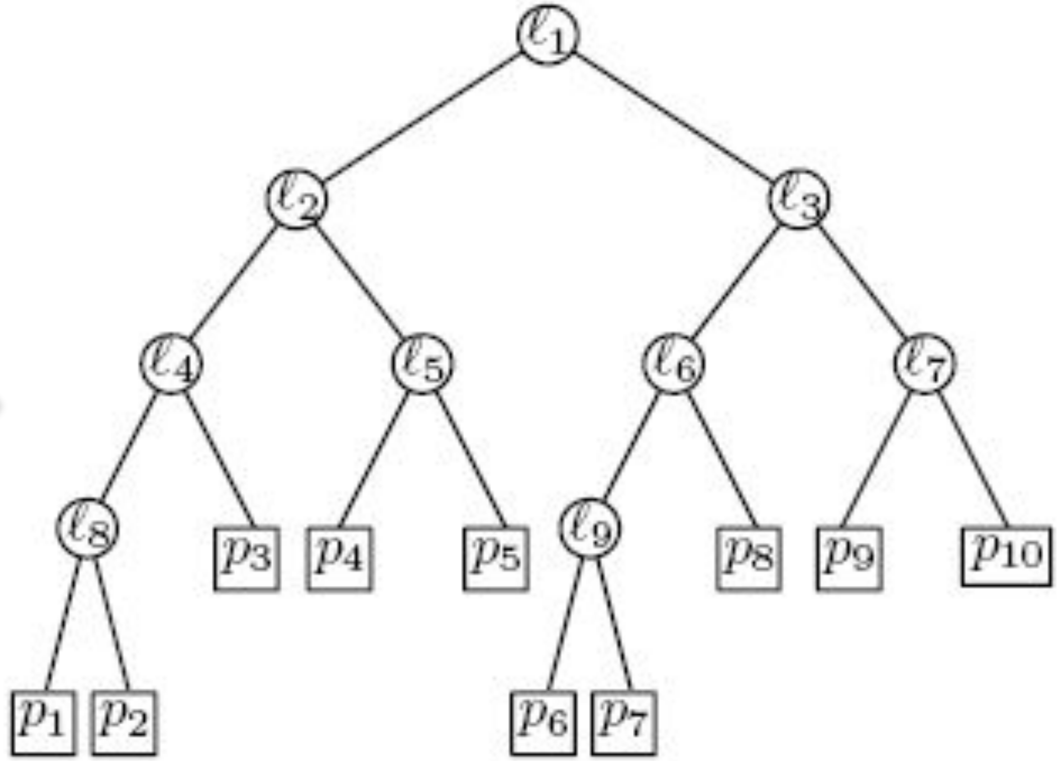
Nearest Neighbor Queries



Retrieve closest two points to the query point

KD Tree

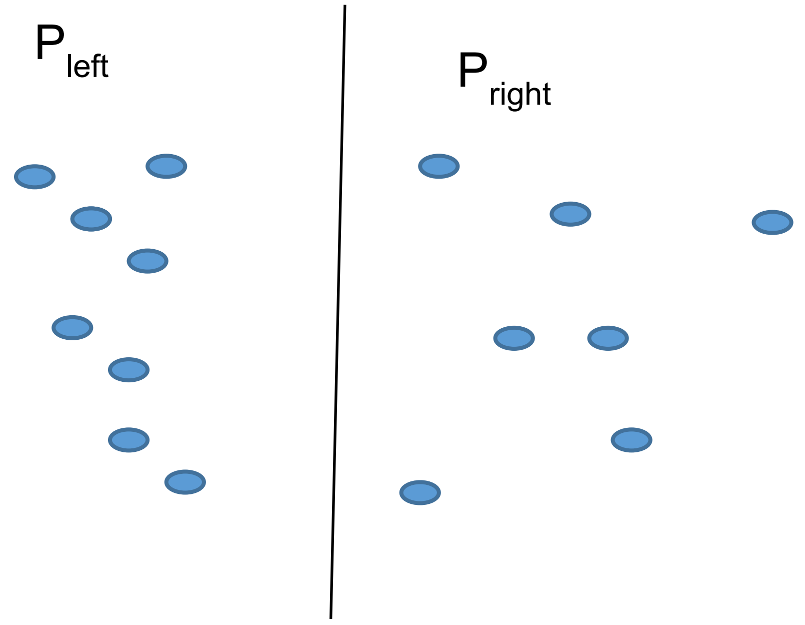


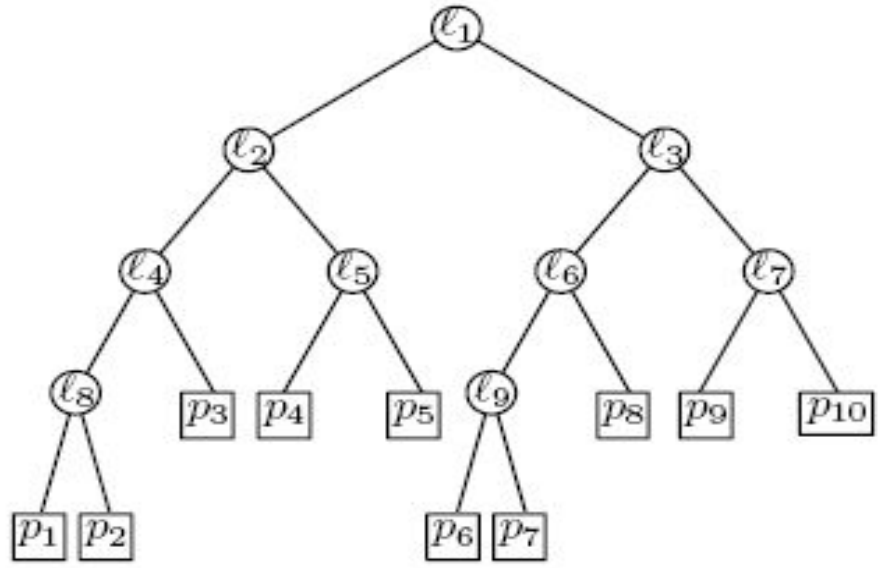


Slide adapted from Dr George Bebis Univ Nevada

KD Tree

- Every node (except leaves) represents a hyperplane that divides the space into two parts.
- Points to the left (right) of this hyperplane represent the left (right) sub-tree of that node.





Slide adapted from Dr George Bebis Univ Nevada

KD Tree

As we move down the tree, we divide the space along alternating (but not always) axis-aligned hyperplanes:

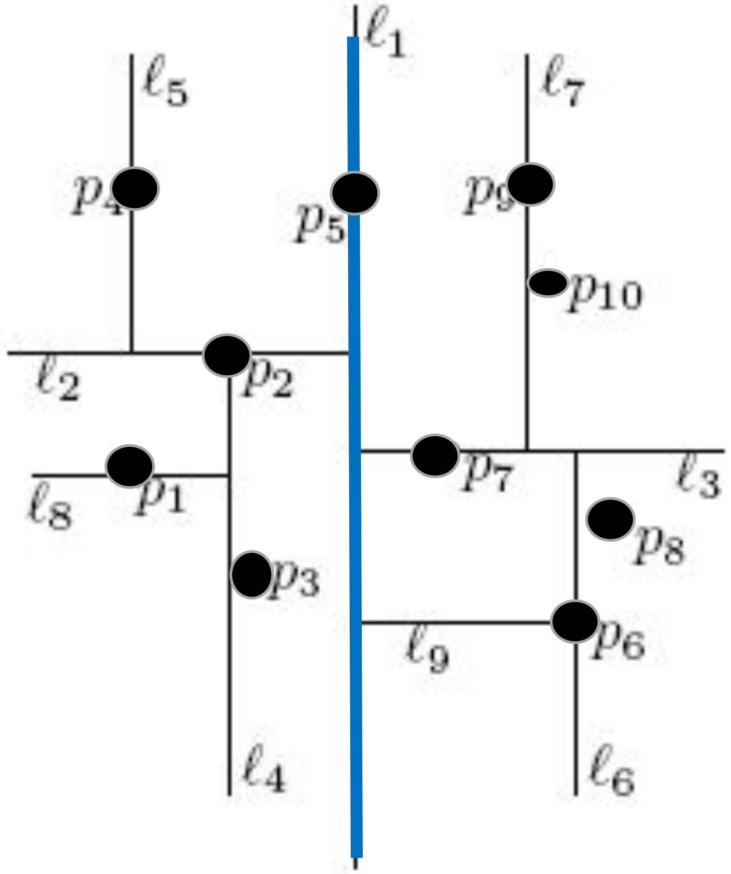
- Split by x-coordinate: split by a vertical line that has (ideally) half the points left or on, and half right.
- Split by y-coordinate: split by a horizontal line that has (ideally) half the points below or on and half above.

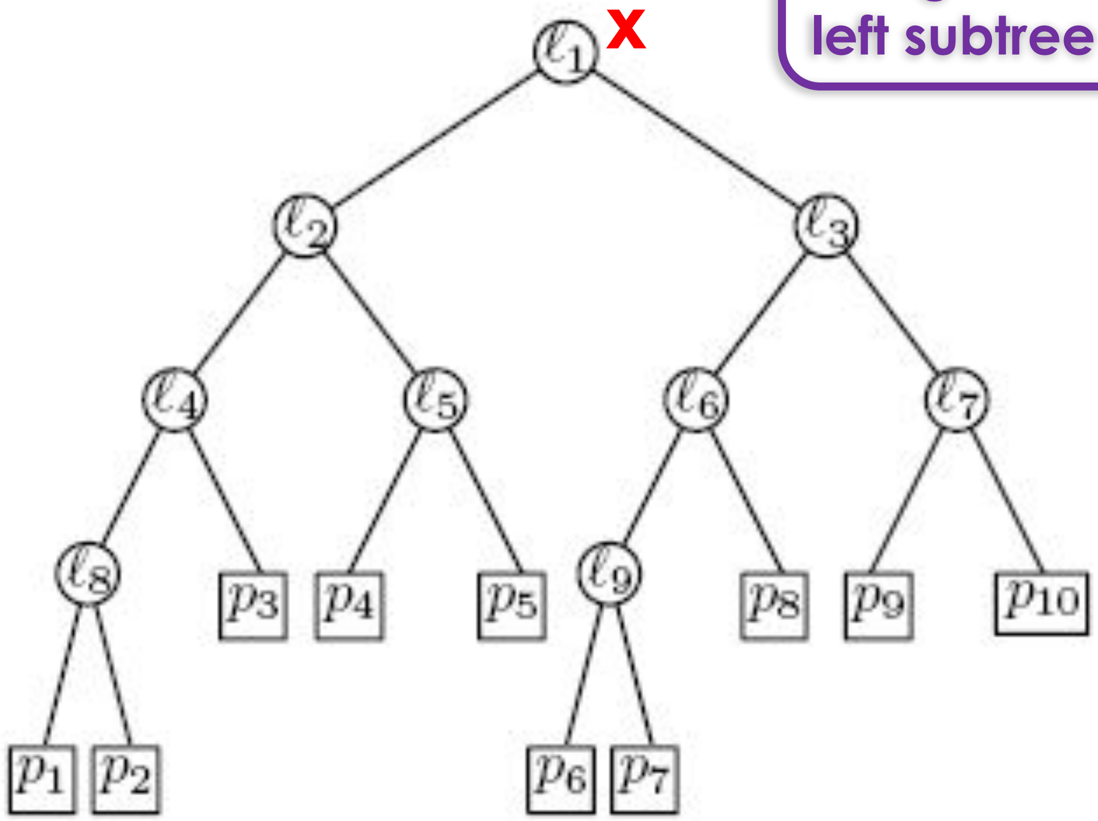
Slide adapted from Dr George Bebis Univ Nevada

KD Tree Construction

Split by x-coordinate: split by a vertical line that has approximately half the points left or on, and half right.

Points on the line go to the left subtree



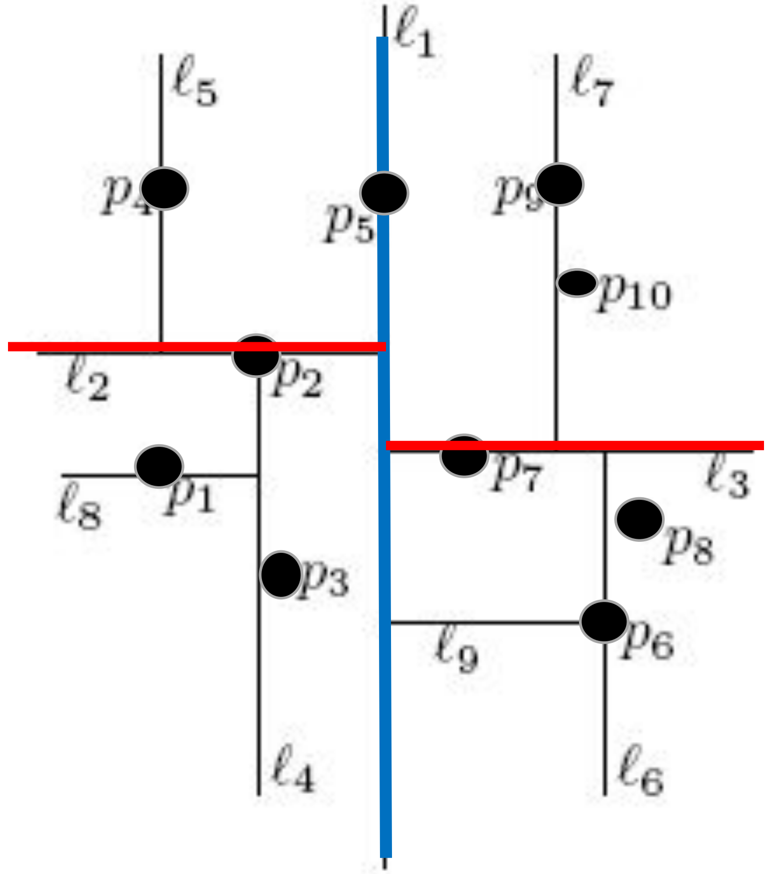


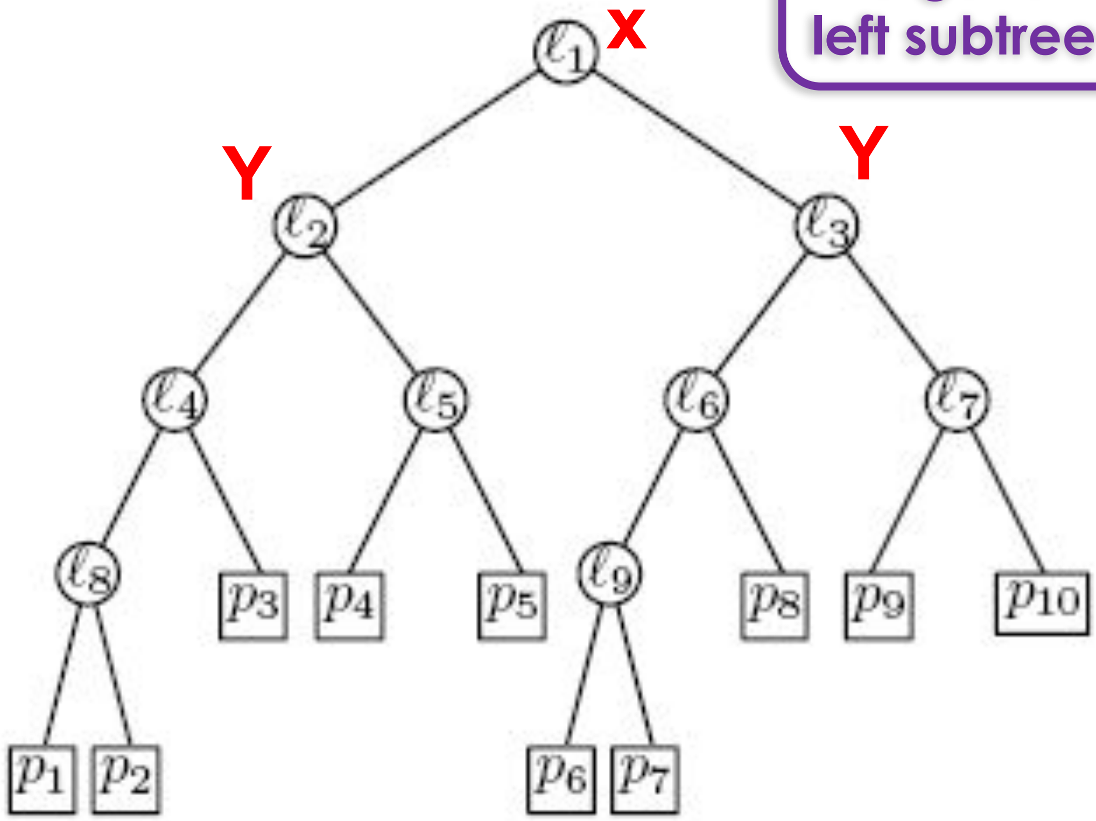
Slide adapted from Dr George Bebis Univ Nevada

KD Tree Construction

Split by y-coordinate: split by a horizontal line that has approximately half the points left or on, and half right.

Points on the line go to the left subtree

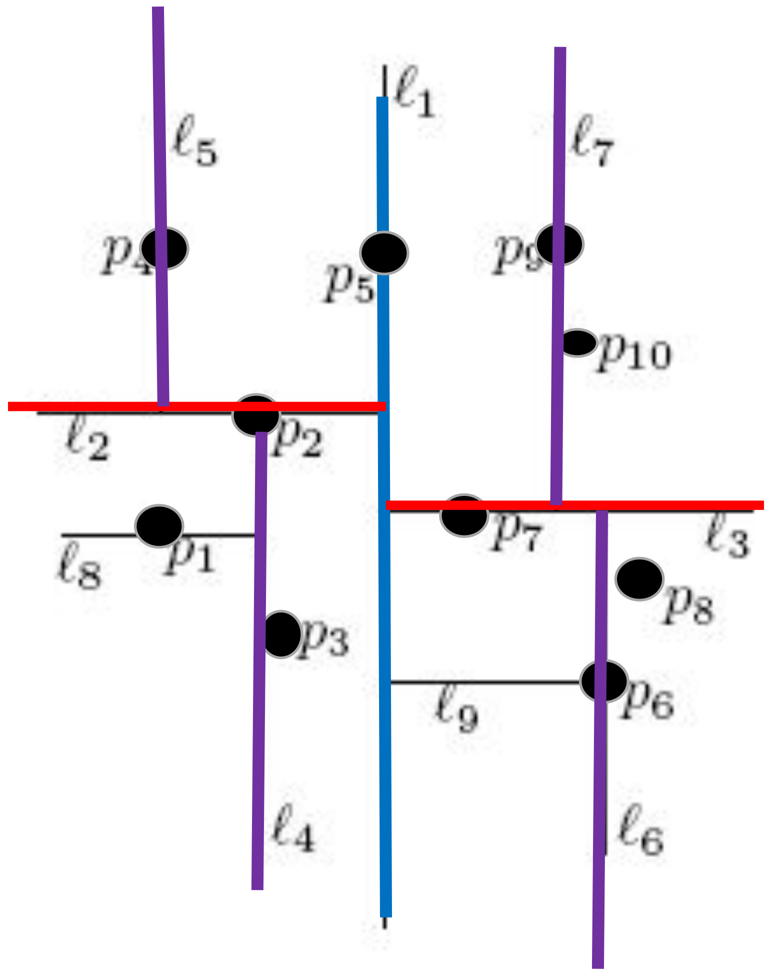


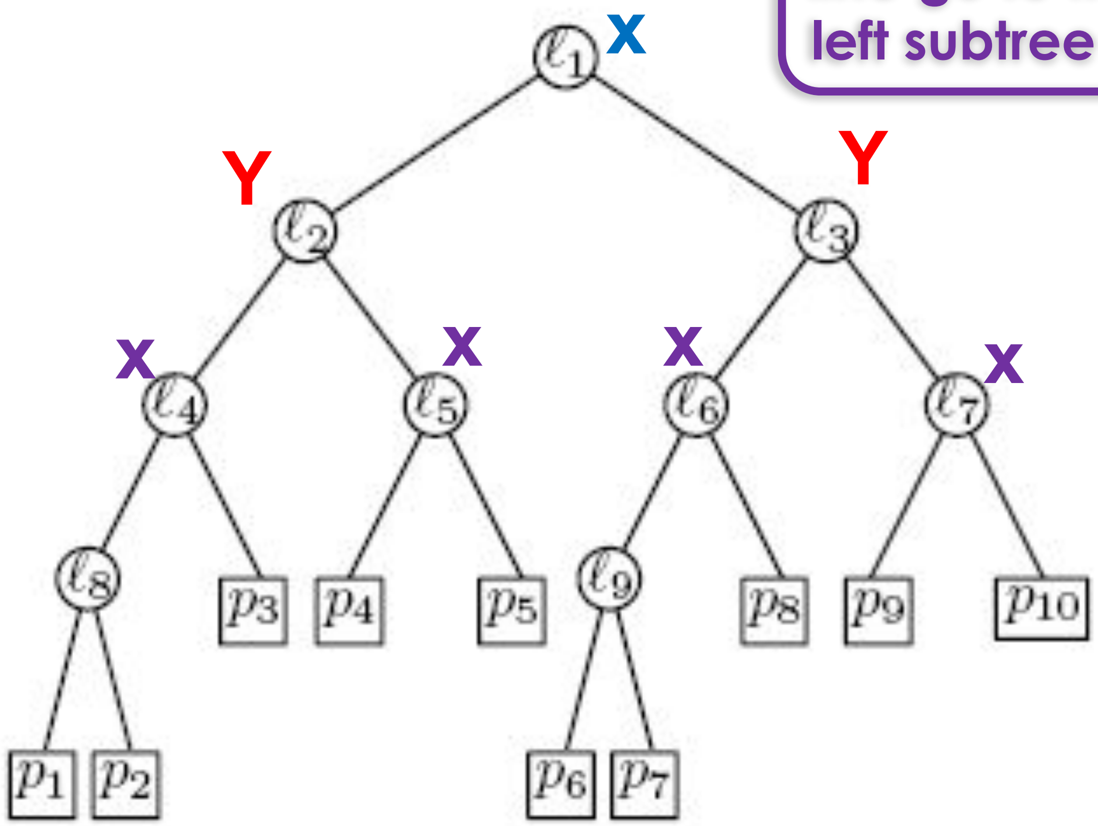


Slide adapted from Dr George Bebis Univ Nevada

KD Tree Construction

<u>Split by x-coordinate</u>*:* split by a vertical line that has approximately half the points left or on, and half right.





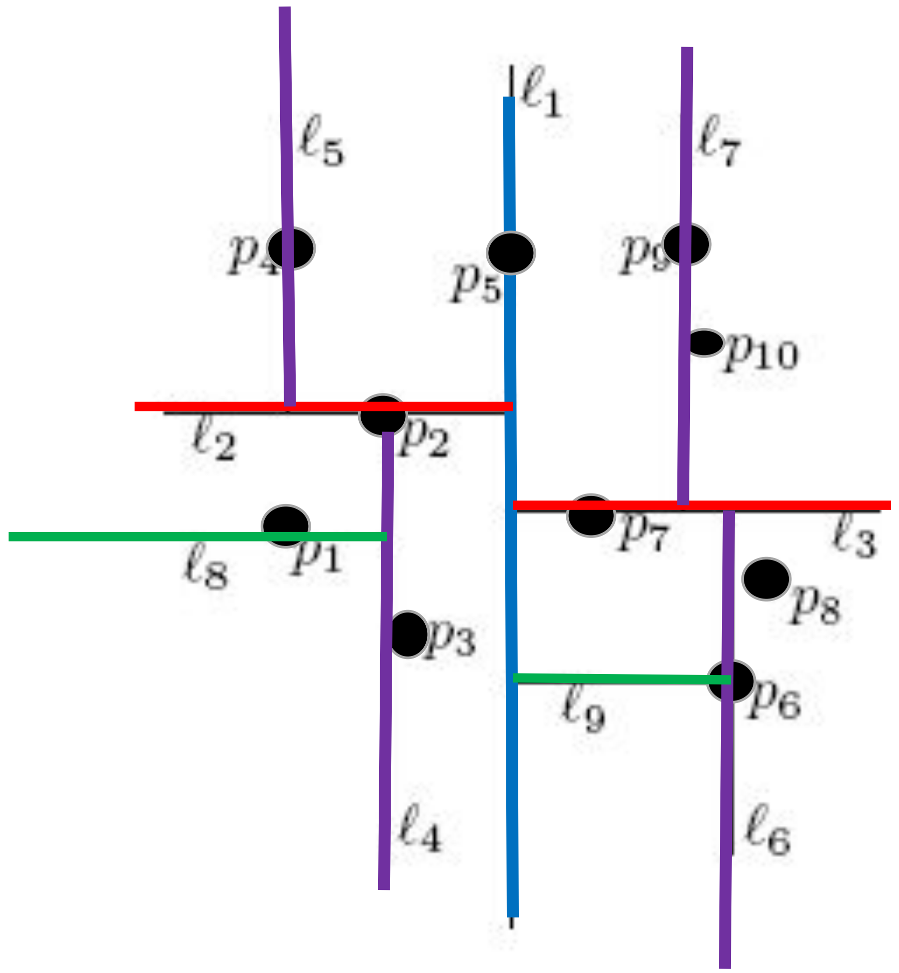
Points on the line go to the left subtree

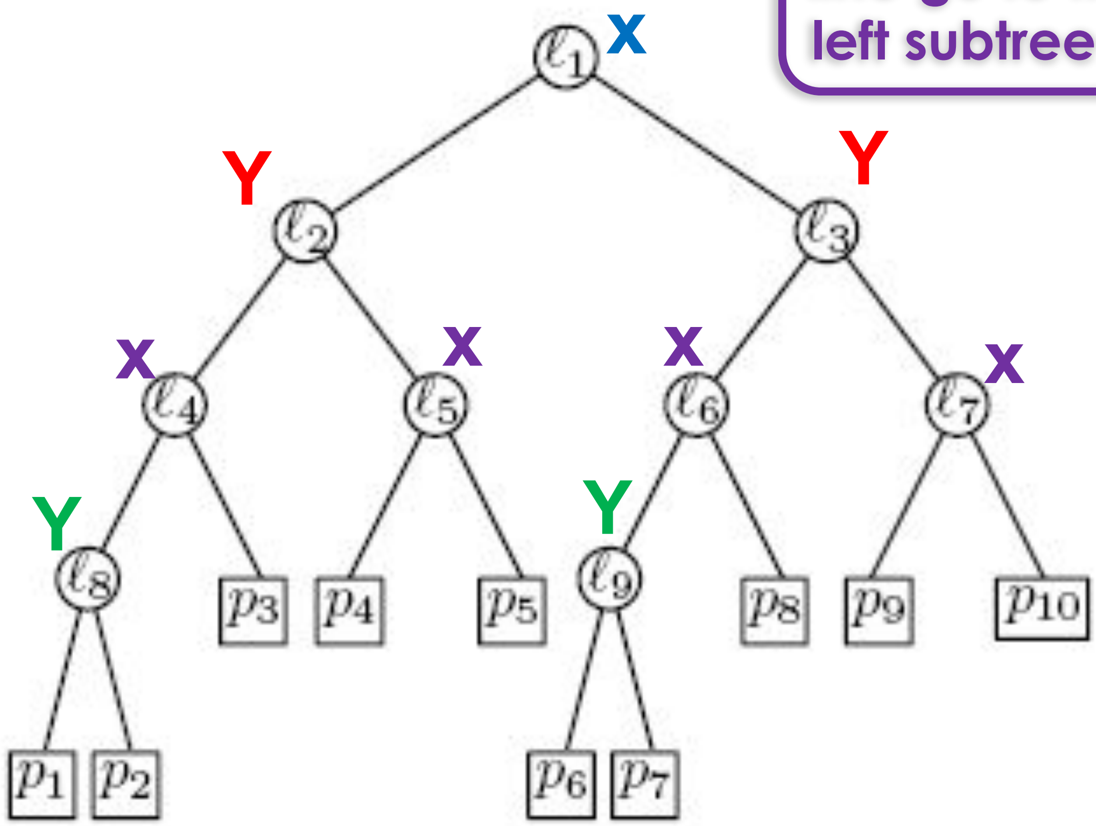
Slide adapted from Dr George Bebis Univ Nevada

KD Tree Construction

<u>Split by y-coordinate</u>: split by a horizontal line that has approximately half the points left or on, and half right.

Points on the line go to the left subtree

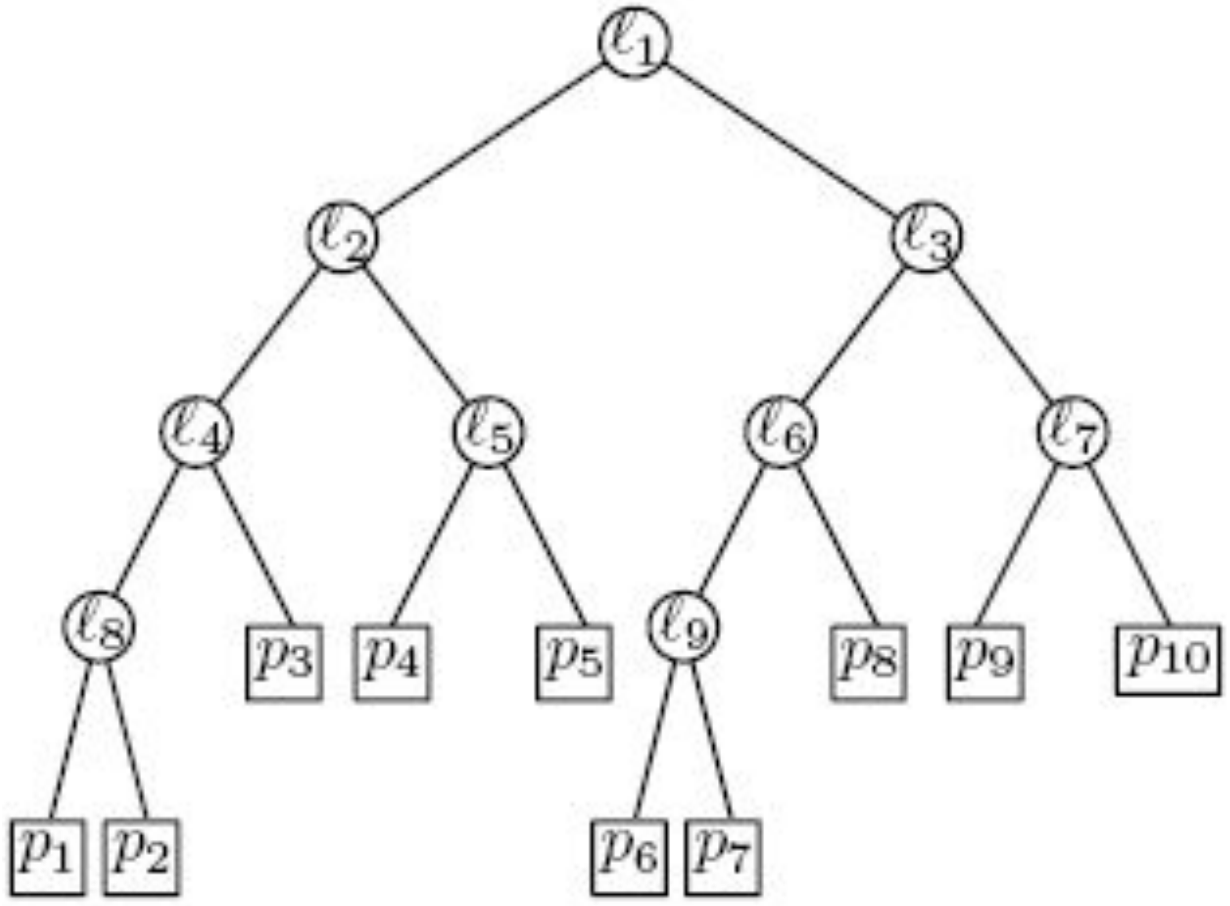




Slide adapted from Dr George Bebis Univ Nevada

KD Tree Node Structure

- A KD-tree node has 5 fields
  - Splitting axis
  - Splitting value
  - Data
  - Left pointer
  - Right pointer



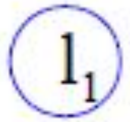
Slide adapted from Dr George Bebis Univ Nevada

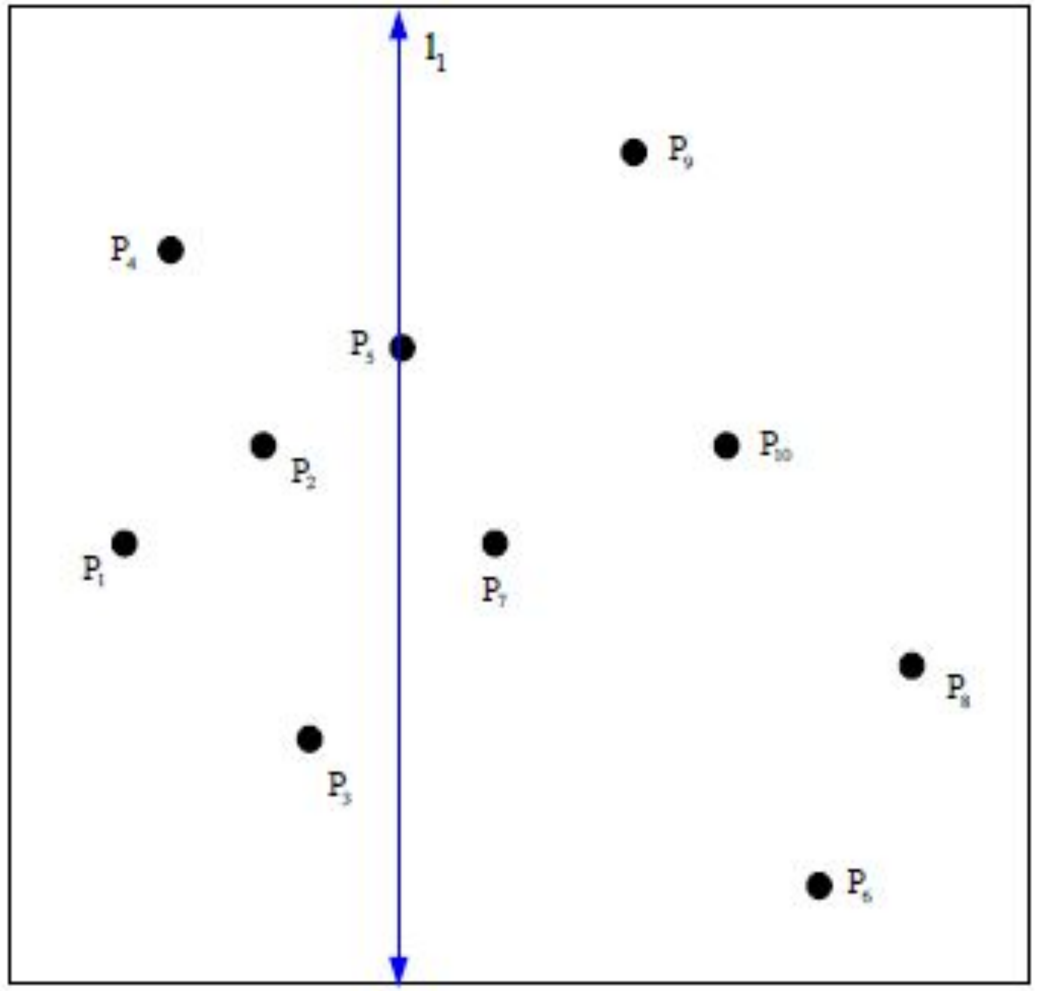
KD Tree Splitting Strategies

- **Divide by finding median** Assumes all the points are available ahead of time.
- **Divide perpendicular to the axis with widest spread**
  - Split axes might not alternate
- And many more….

Slide adapted from Dr George Bebis Univ Nevada

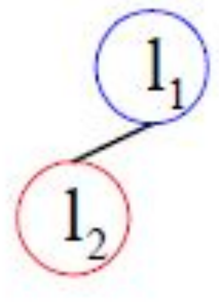
Example – using median (alternating between axis) data stored at the leaves

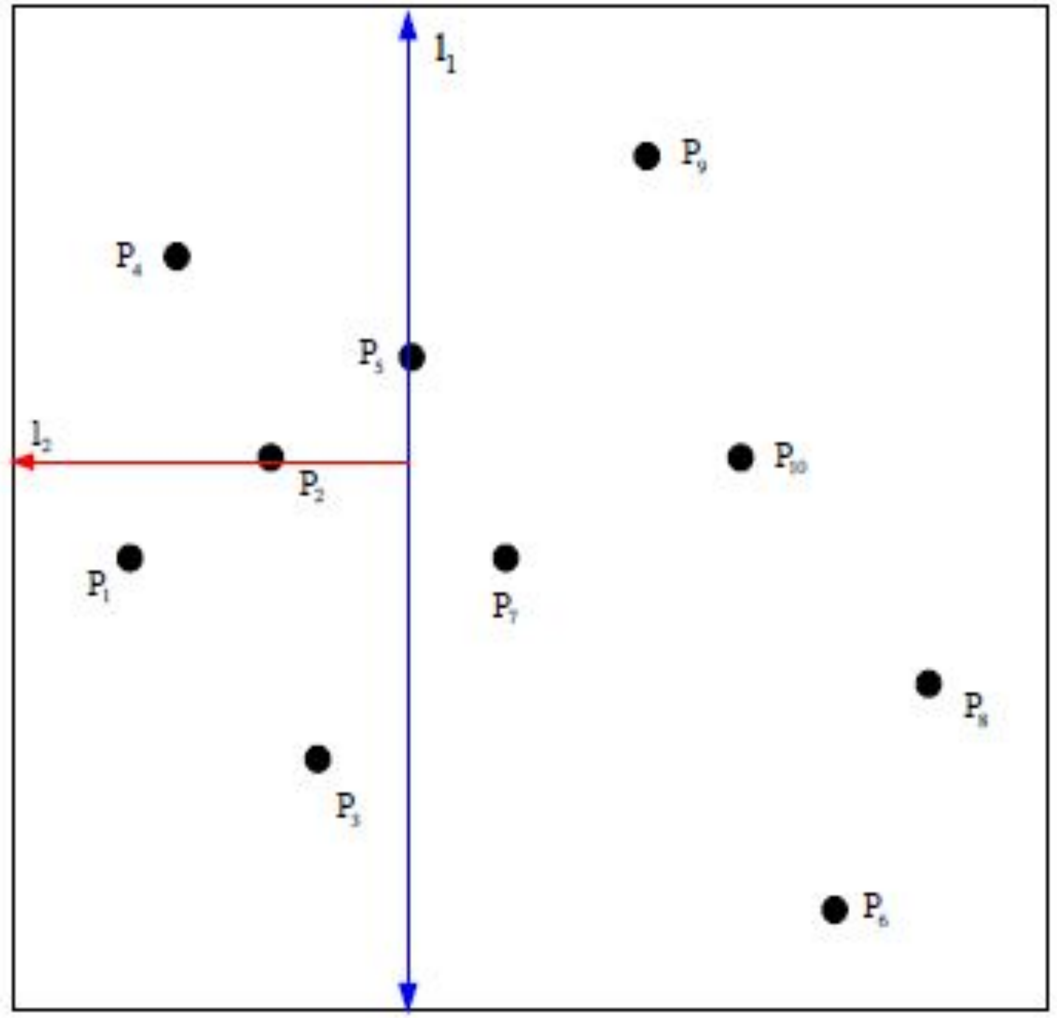




Slide adapted from Dr George Bebis Univ Nevada

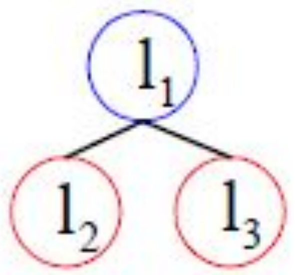
Example – using median (alternating between axis) data stored at the leaves

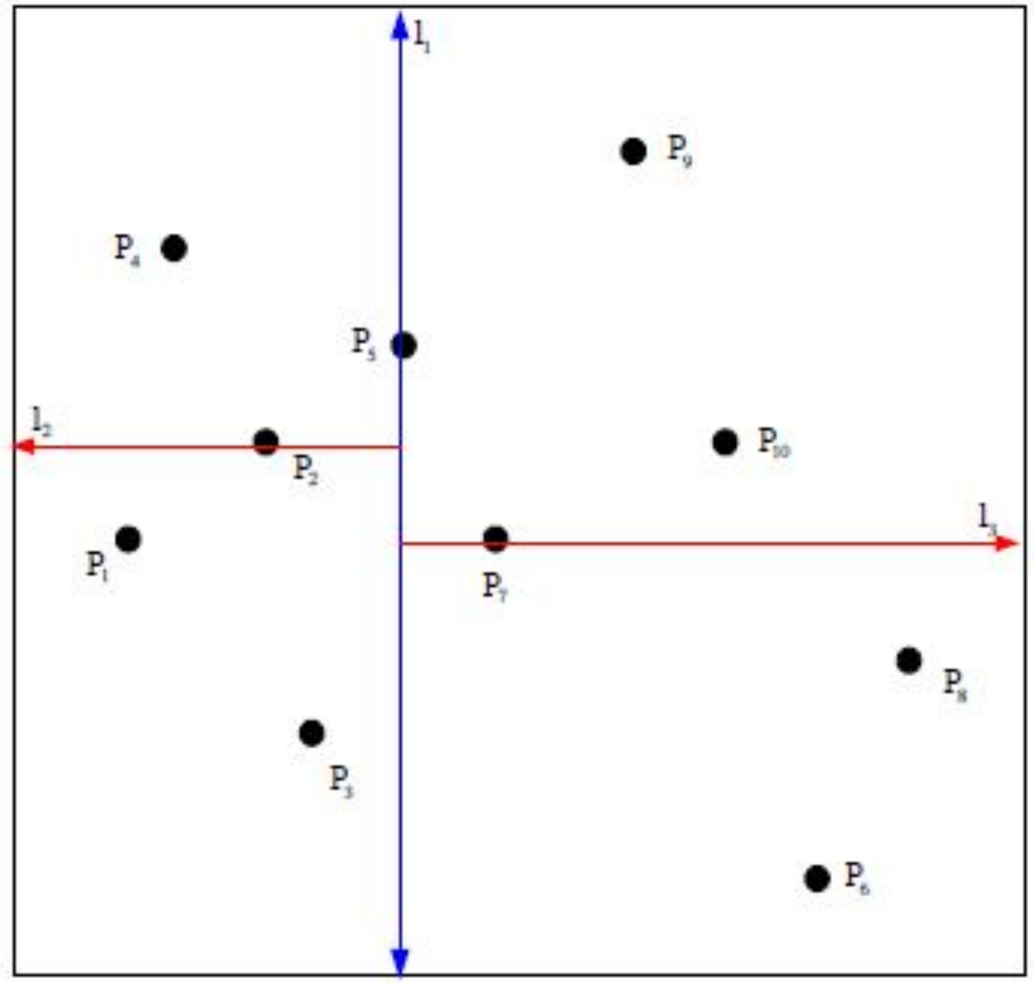




Slide adapted from Dr George Bebis Univ Nevada

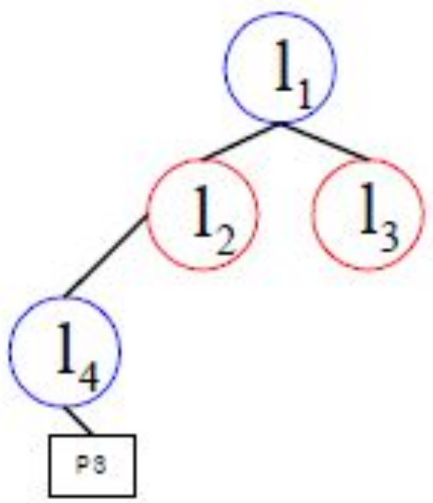
Example – using median (alternating between axis) data stored at the leaves

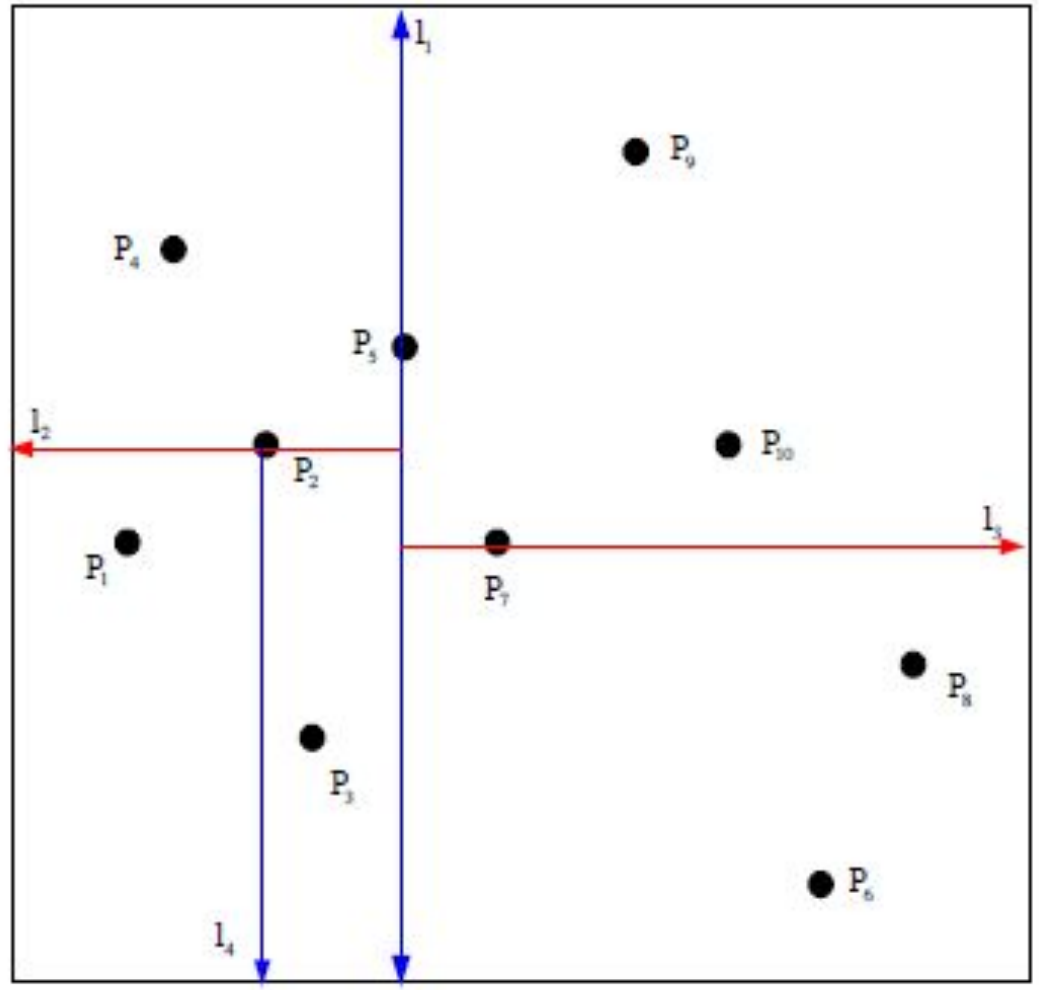




Slide adapted from Dr George Bebis Univ Nevada

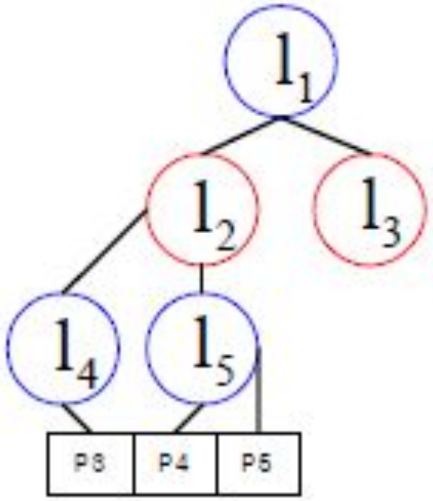
Example – using median (alternating between axis) data stored at the leaves

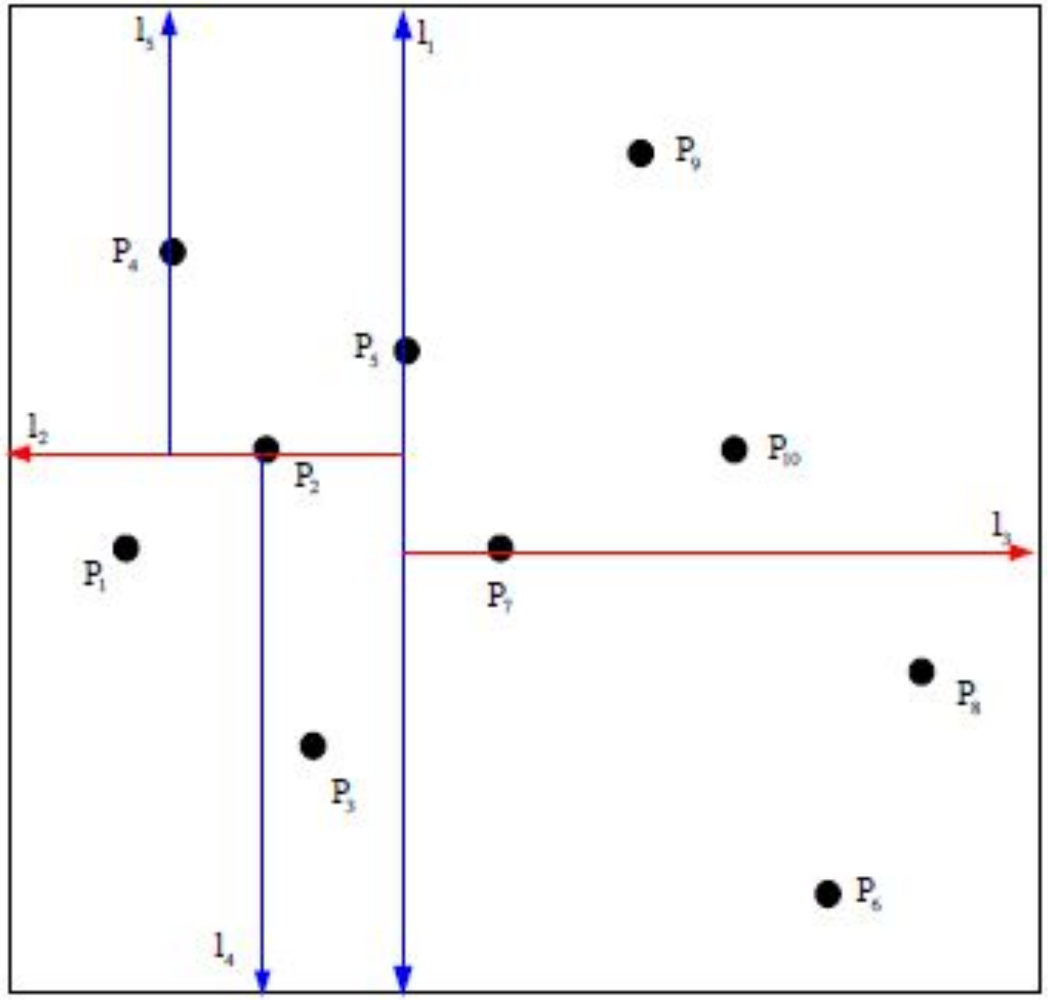




Slide adapted from Dr George Bebis Univ Nevada

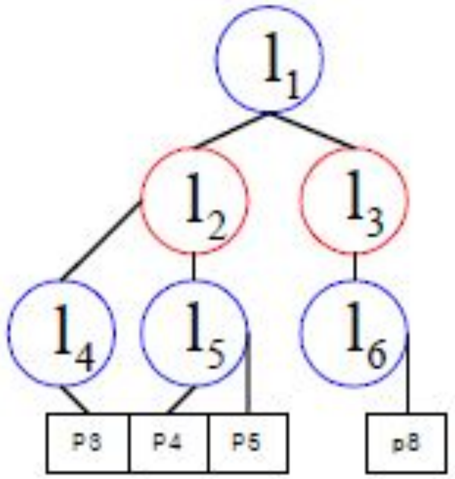
Example – using median (alternating between axis) data stored at the leaves

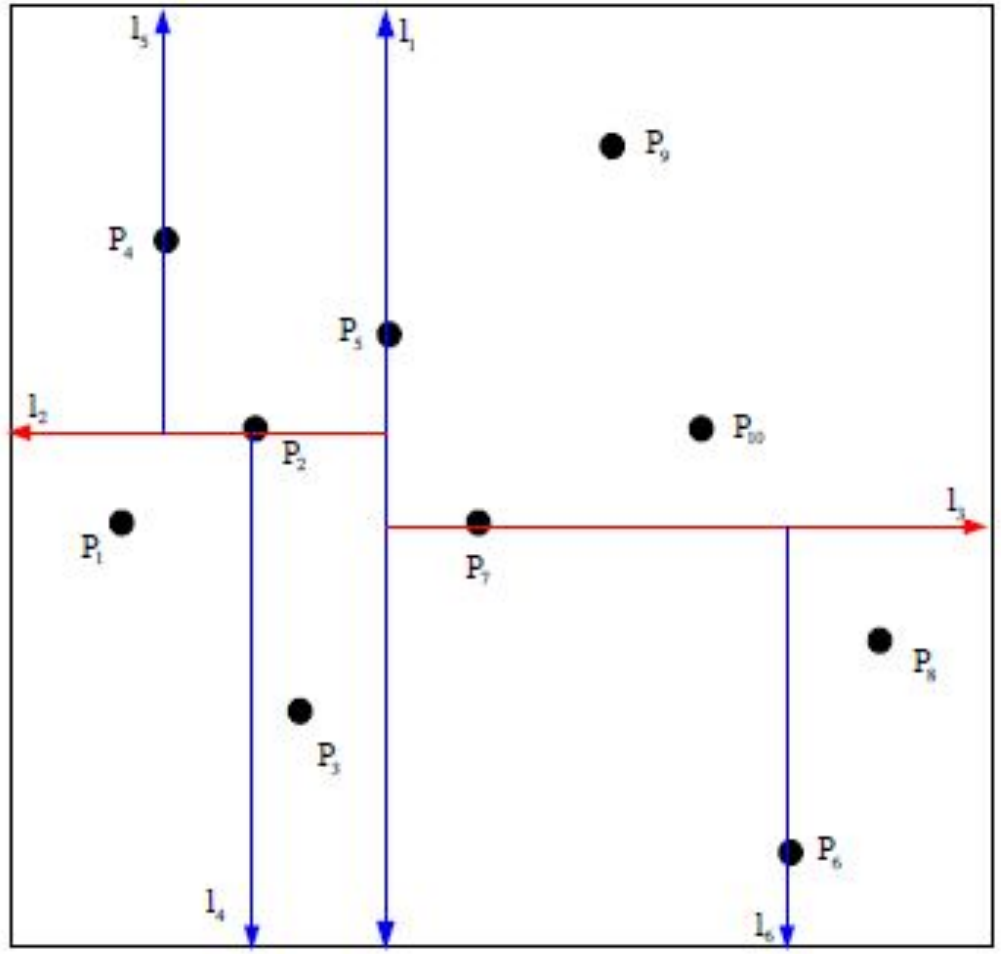




Slide adapted from Dr George Bebis Univ Nevada

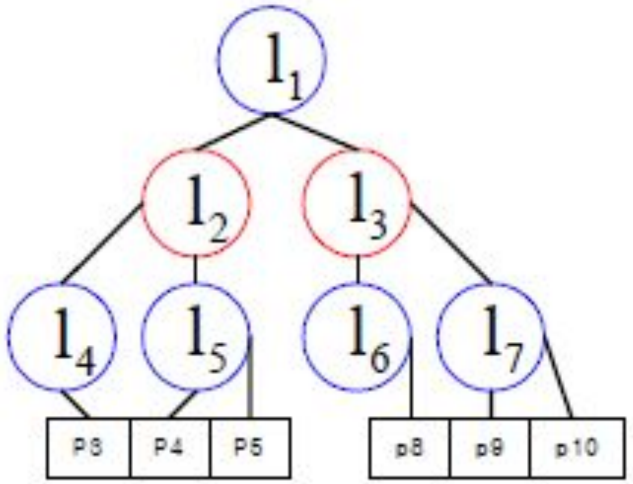
Example – using median (alternating between axis) data stored at the leaves

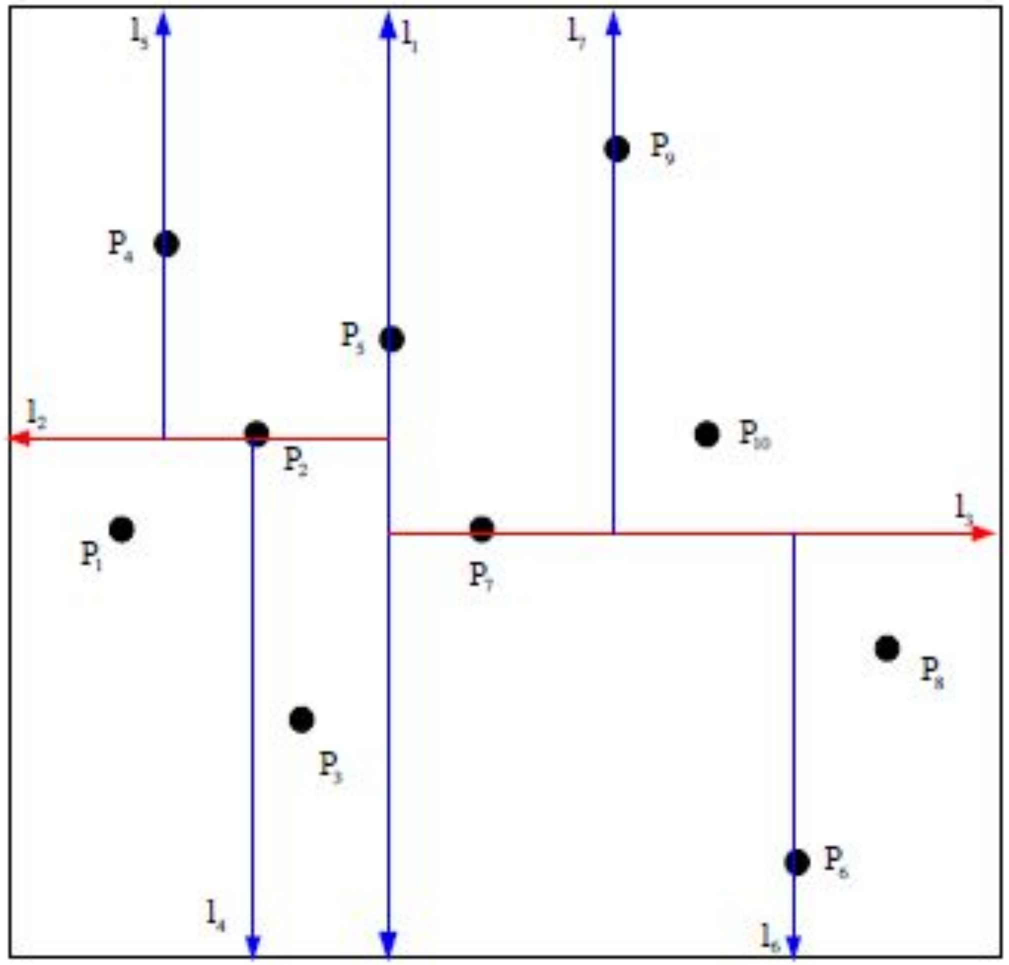




Slide adapted from Dr George Bebis Univ Nevada

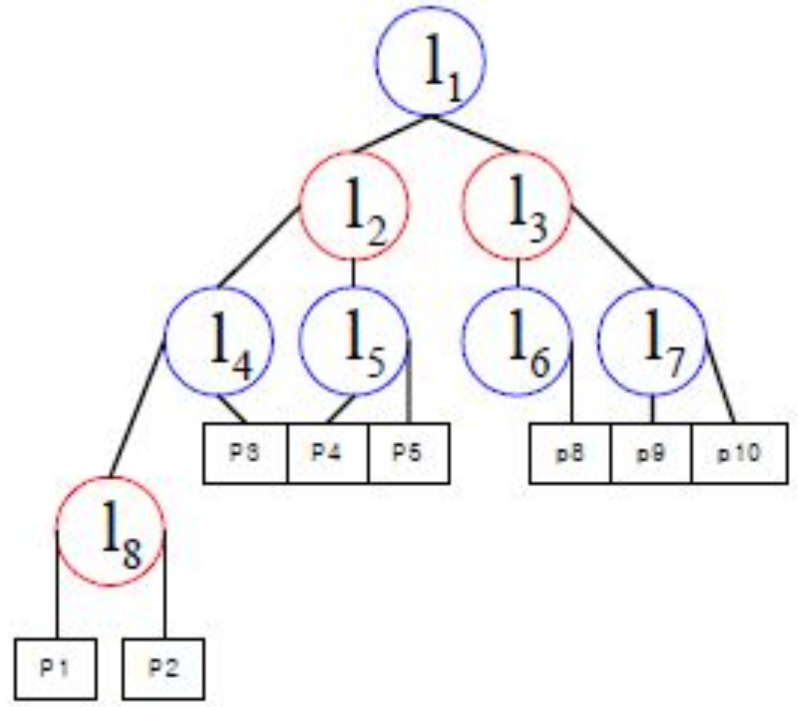
Example – using median (alternating between axis) data stored at the leaves

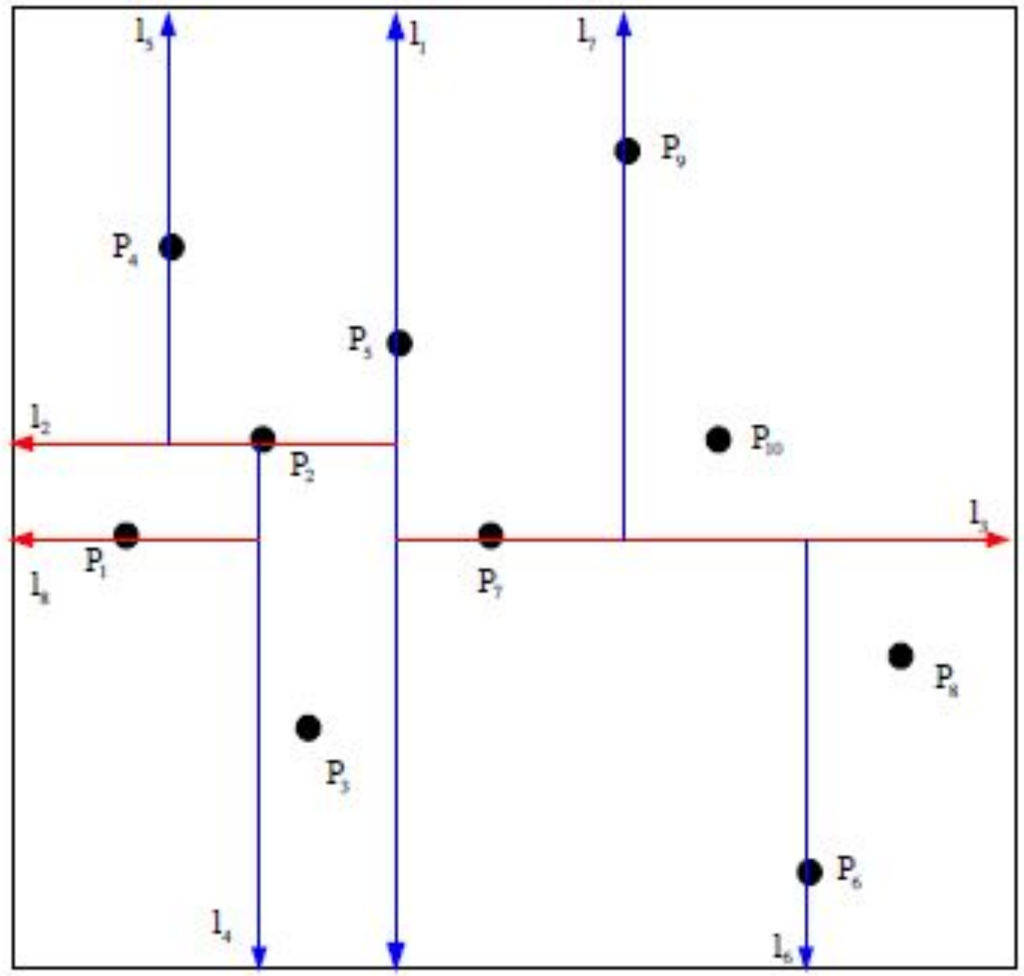




Slide adapted from Dr George Bebis Univ Nevada

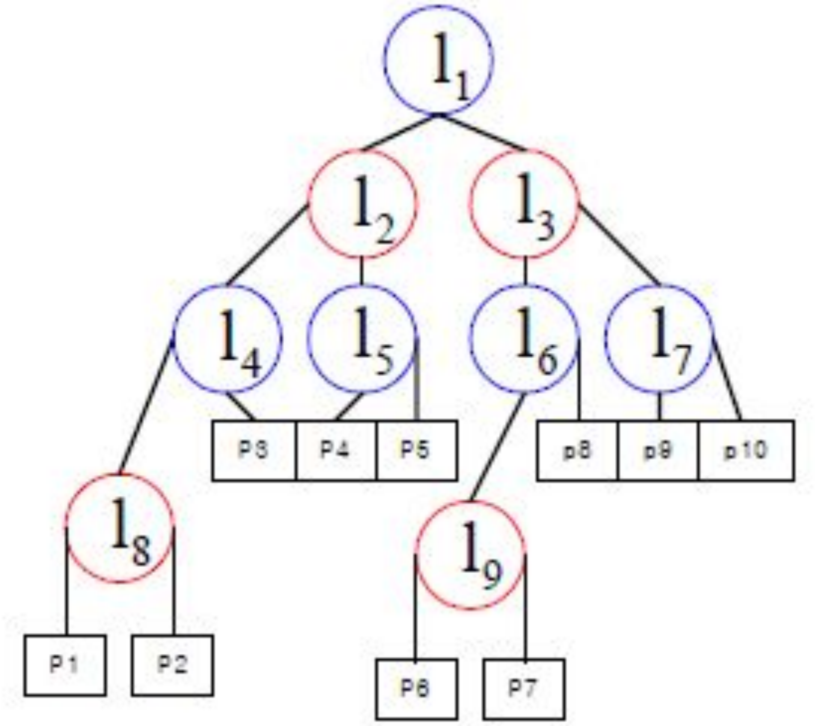
Example – using median (alternating between axis) data stored at the leaves

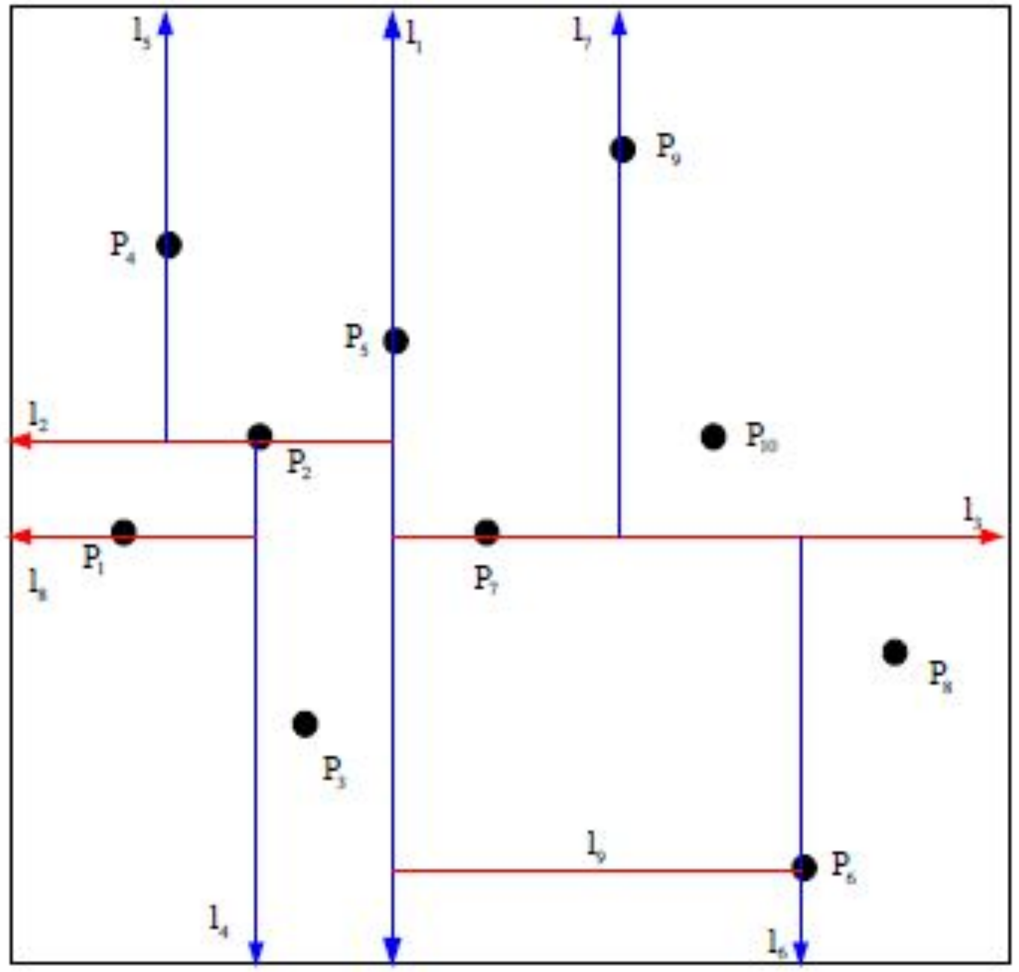




Slide adapted from Dr George Bebis Univ Nevada

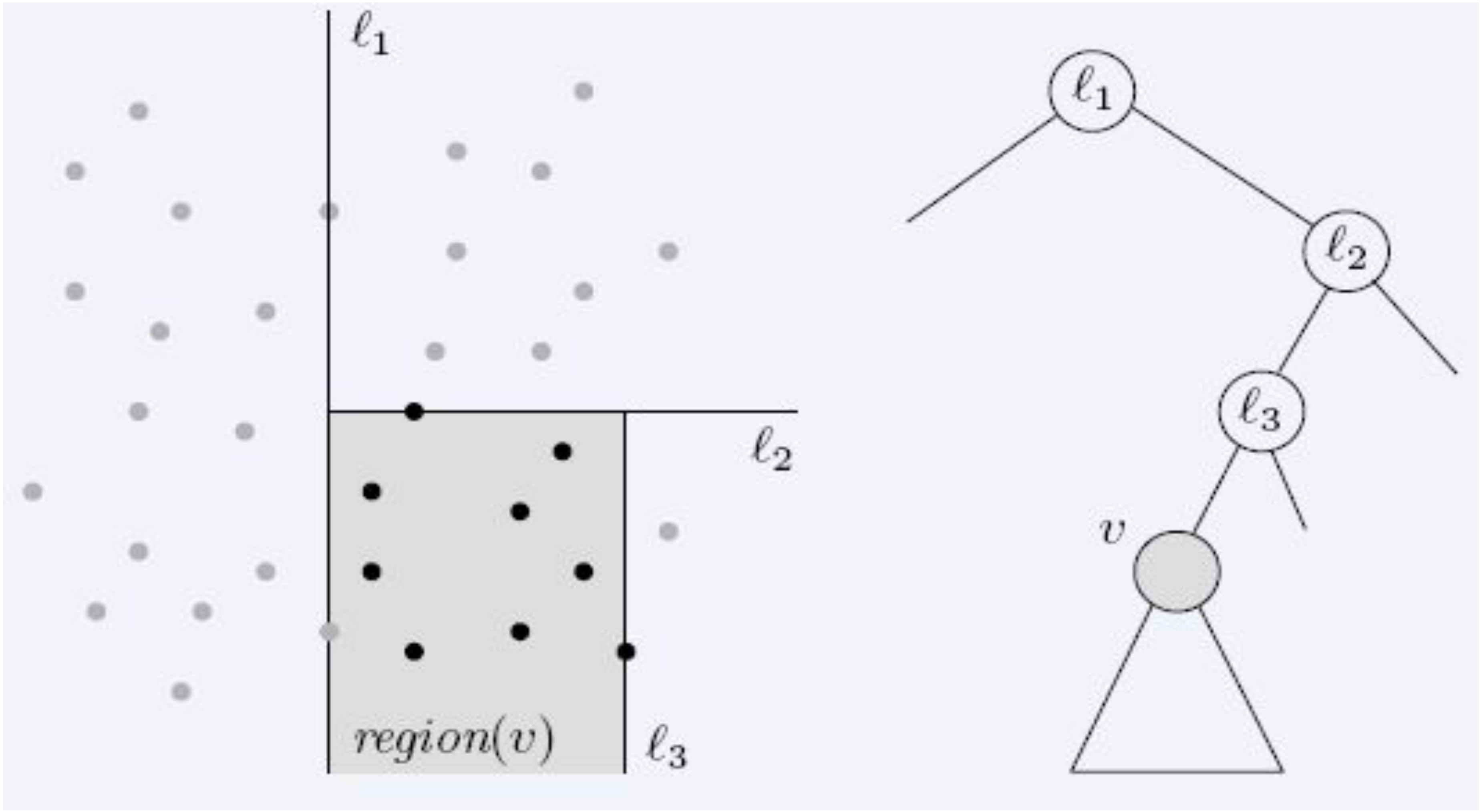
Example – using median (alternating between axis) data stored at the leaves





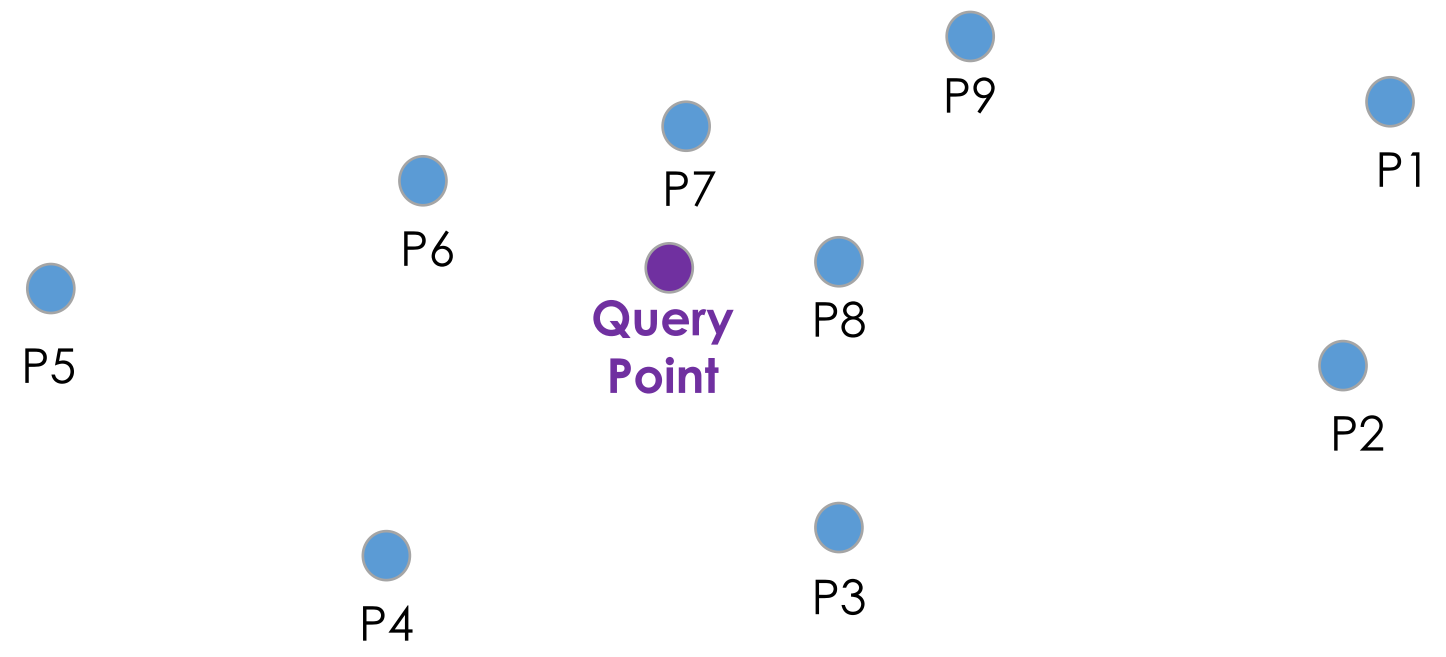
Slide adapted from Dr George Bebis Univ Nevada

Region of node v



Region(v) : the subtree rooted at **v** stores the points in black dots

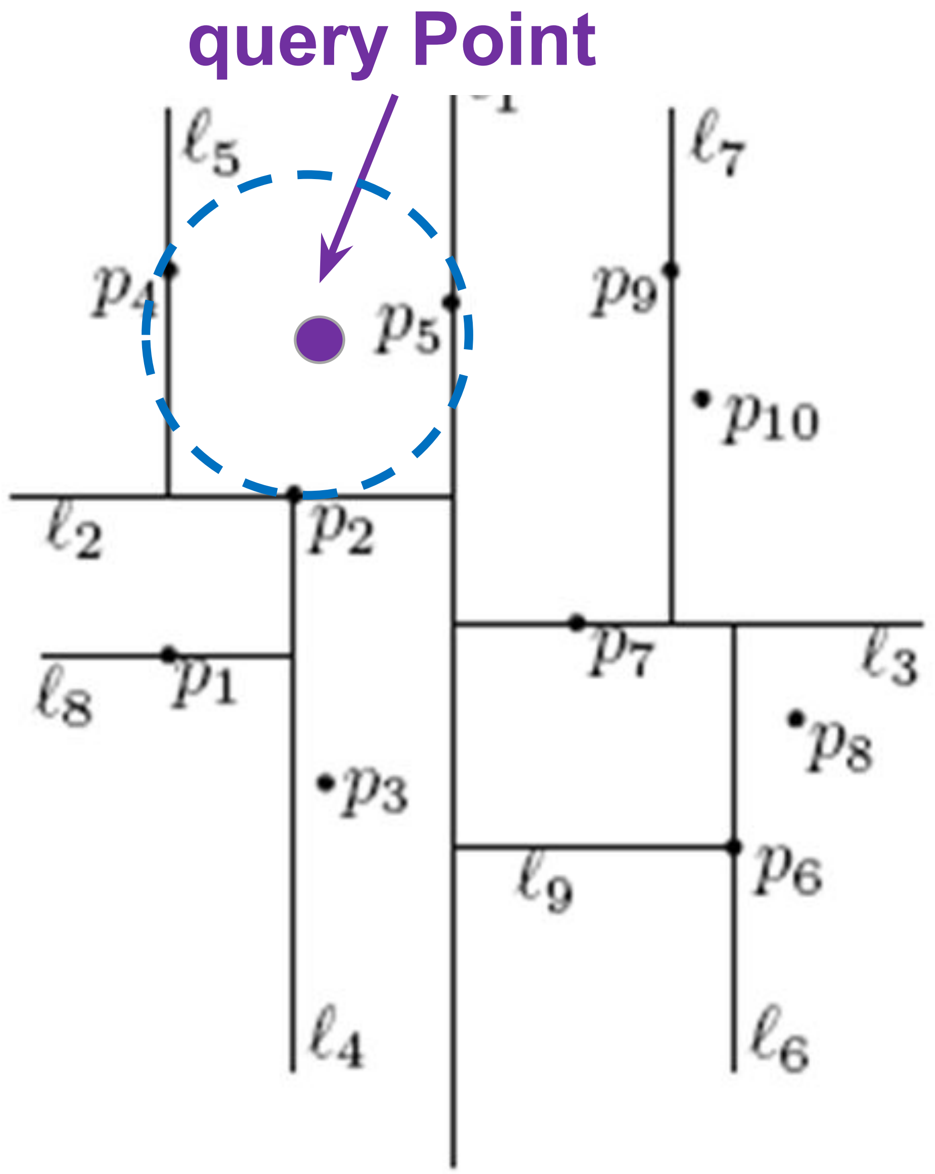
Recall Nearest Neighbor Queries



Retrieve closest k points to the query point

KD Trees – Nearest Neighbor Search

- KNN □ Find k nearest neighbor of the given query point.
- First we will discuss 1-NN and then generalize to K nearest neighbors.
- **Key Idea:** start off with an estimate on nearest neighbor and then keep updating whenever we find a better one.

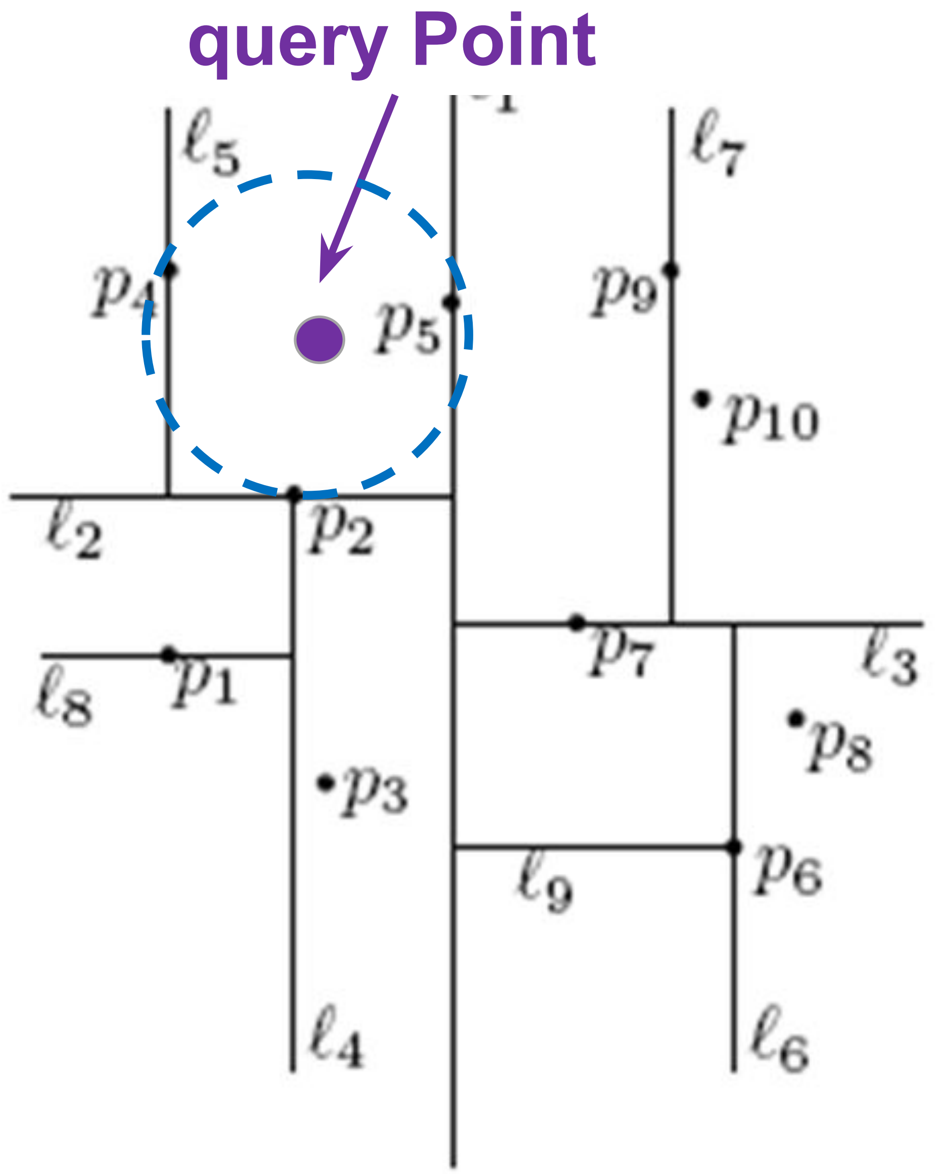


KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space without actually computing distance to the actual data points

2) Also if my initial estimate is bad, I would end up doing a lot of replacements.



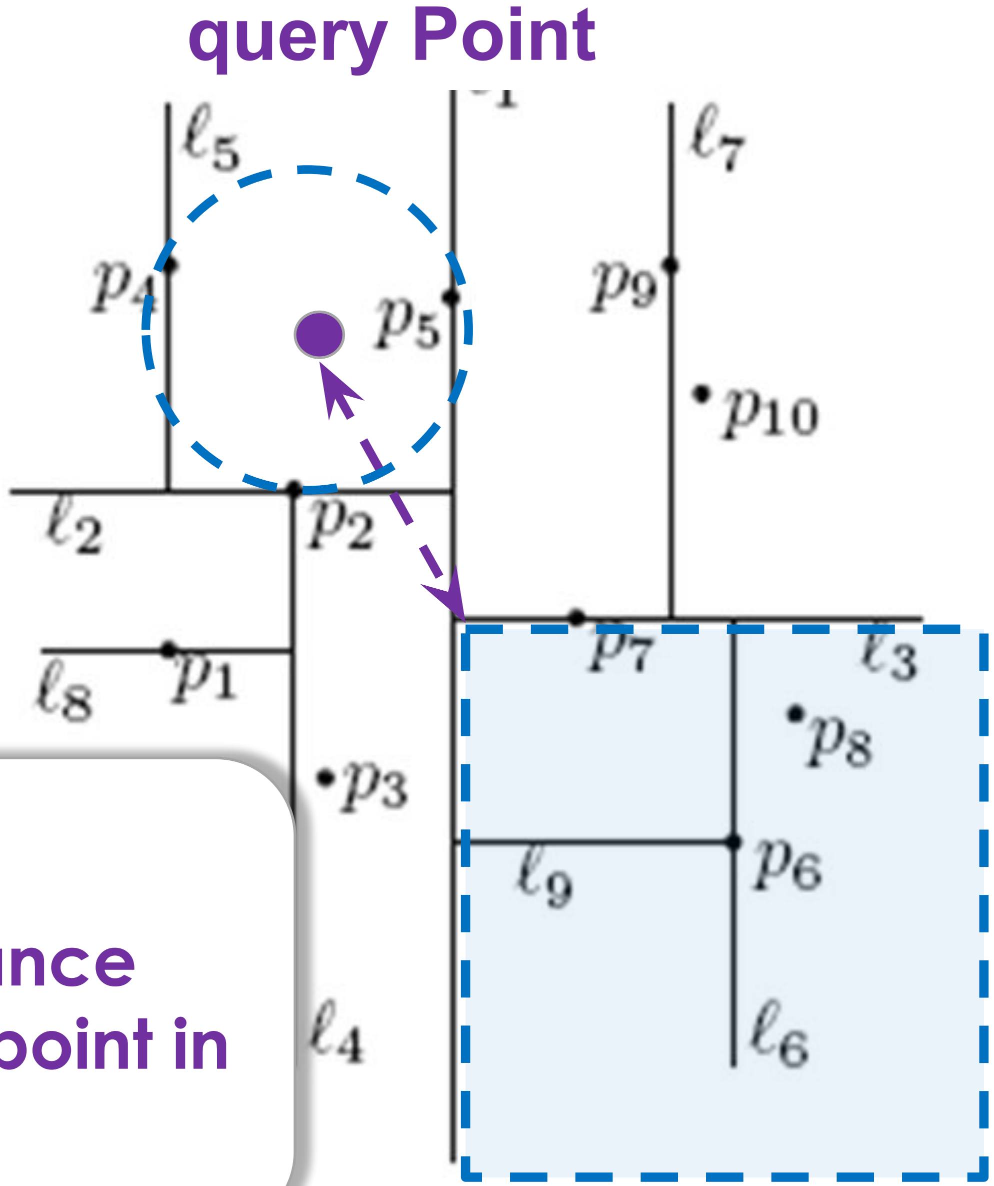
KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) **Need a smart way to prune through the data space**

2) Also if my initial estimate is bad, I would end up doing a lot of replacements.

- **Consider the left child of L3**
- **What is the minimum possible distance between the query point and any point in the region of lc(L3)?**



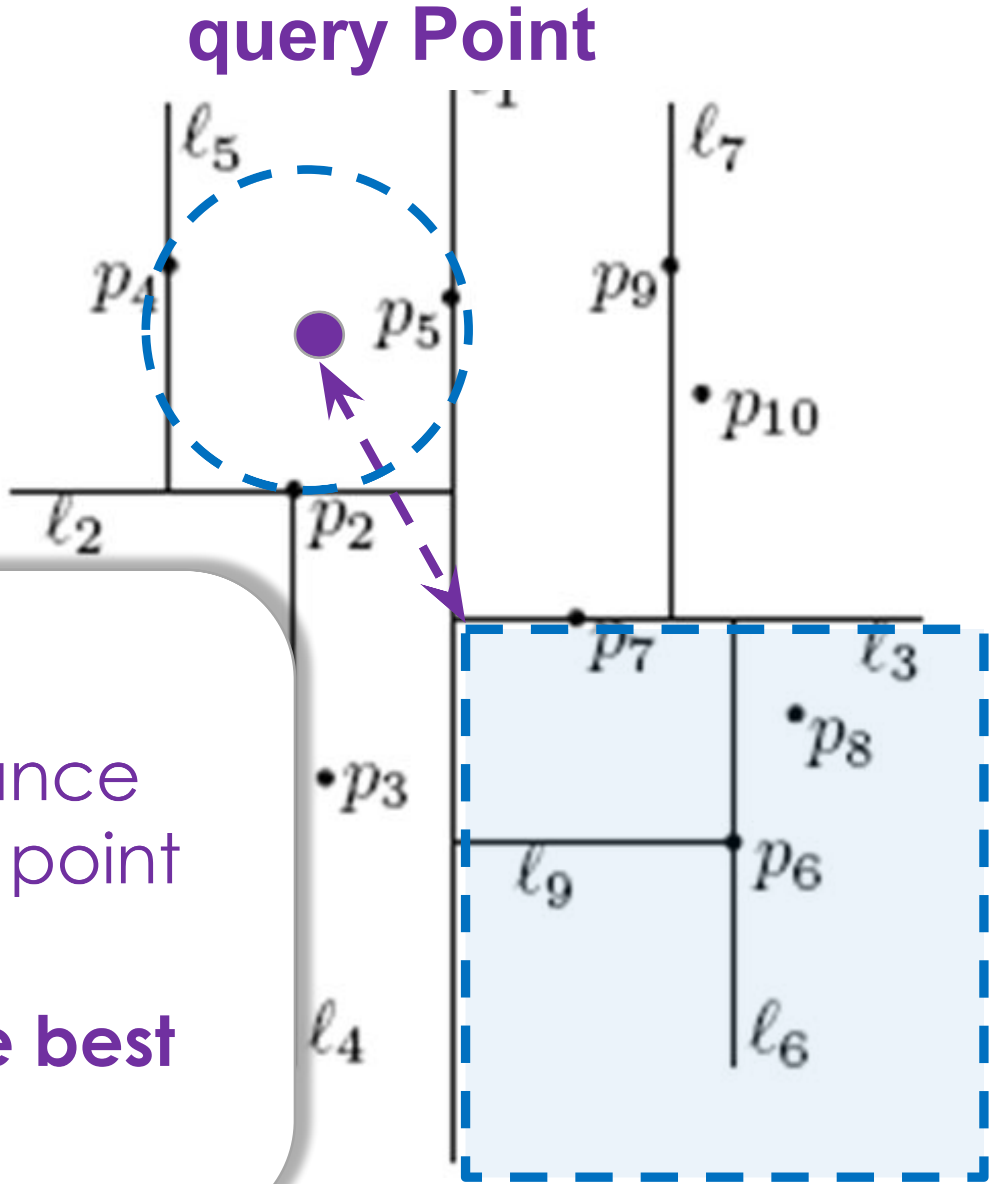
KD Trees – Nearest Neighbor Search

- Two things to consider for such a approach:

1) **Need a smart way to prune through the data space**

2) Also if my initial estimate is bad, I

- Consider the left child of L3
- What is the minimum possible distance between the query point and any point in the region of lc(L3)?
- **If it is less than our current distance best estimate (P2), then what???**

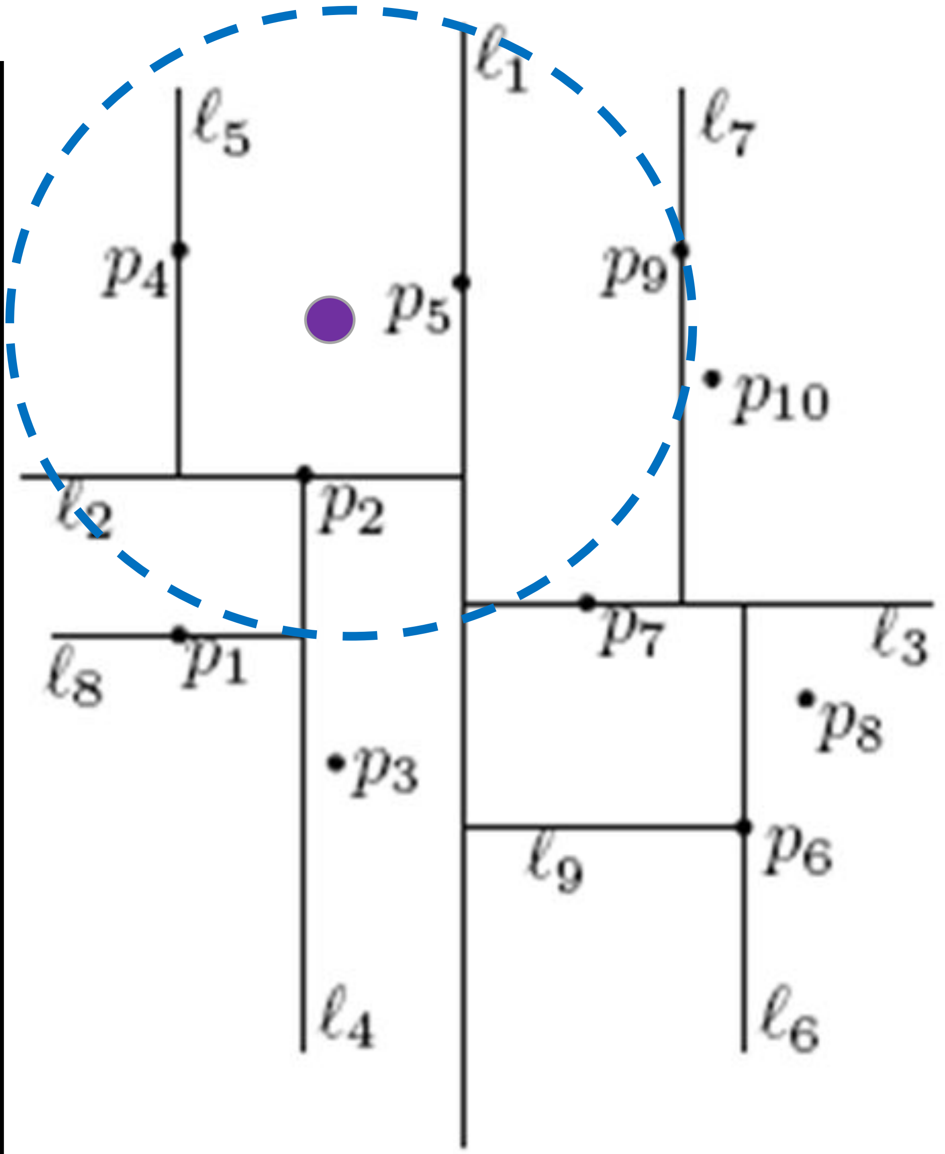


KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space

2) **if initial estimate is bad □ a lot of replacements.**
   - **E.g., P9 would be a very bad initial estimate**

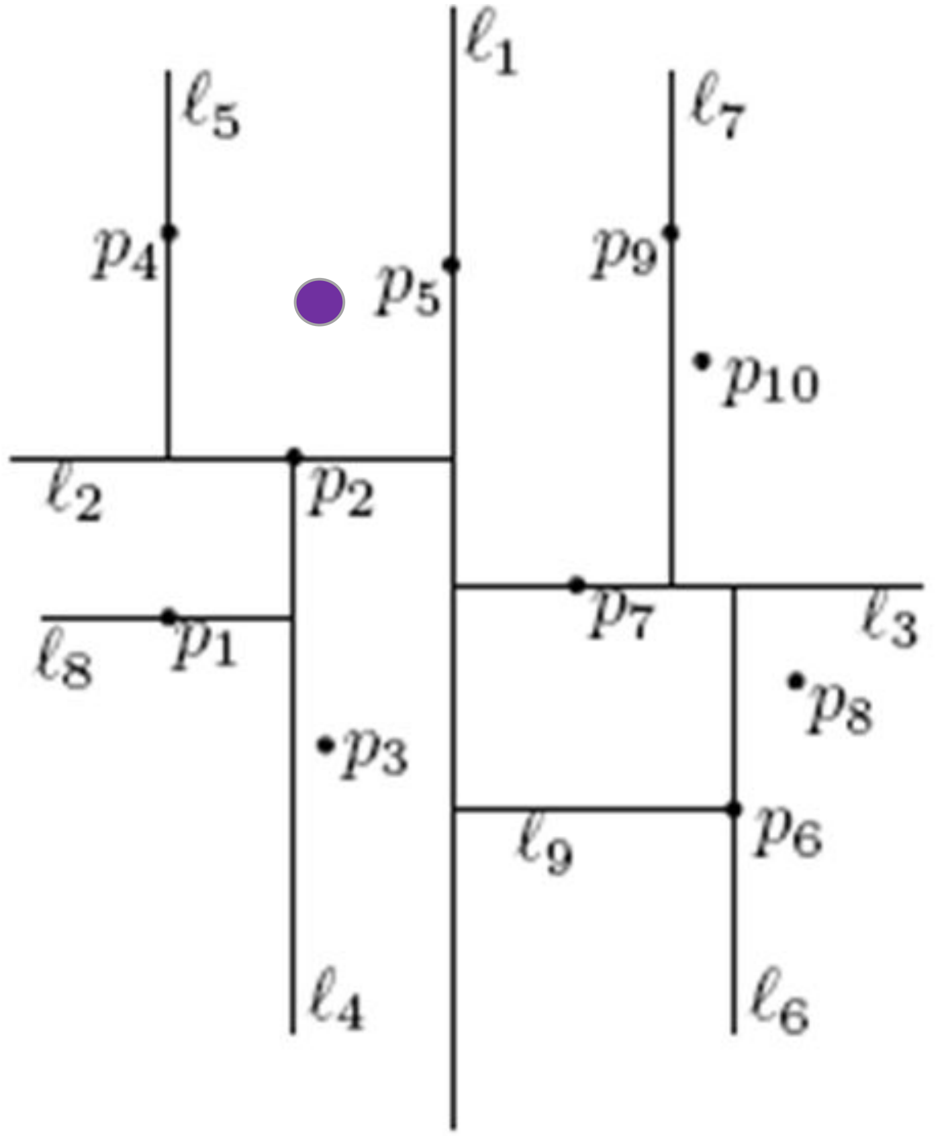


KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space

2) **if initial estimate is bad □ a lot of replacements.**
   - **E.g., P9 would be a very bad initial estimate**
   - **Structure of KD-Tree helps us again.**

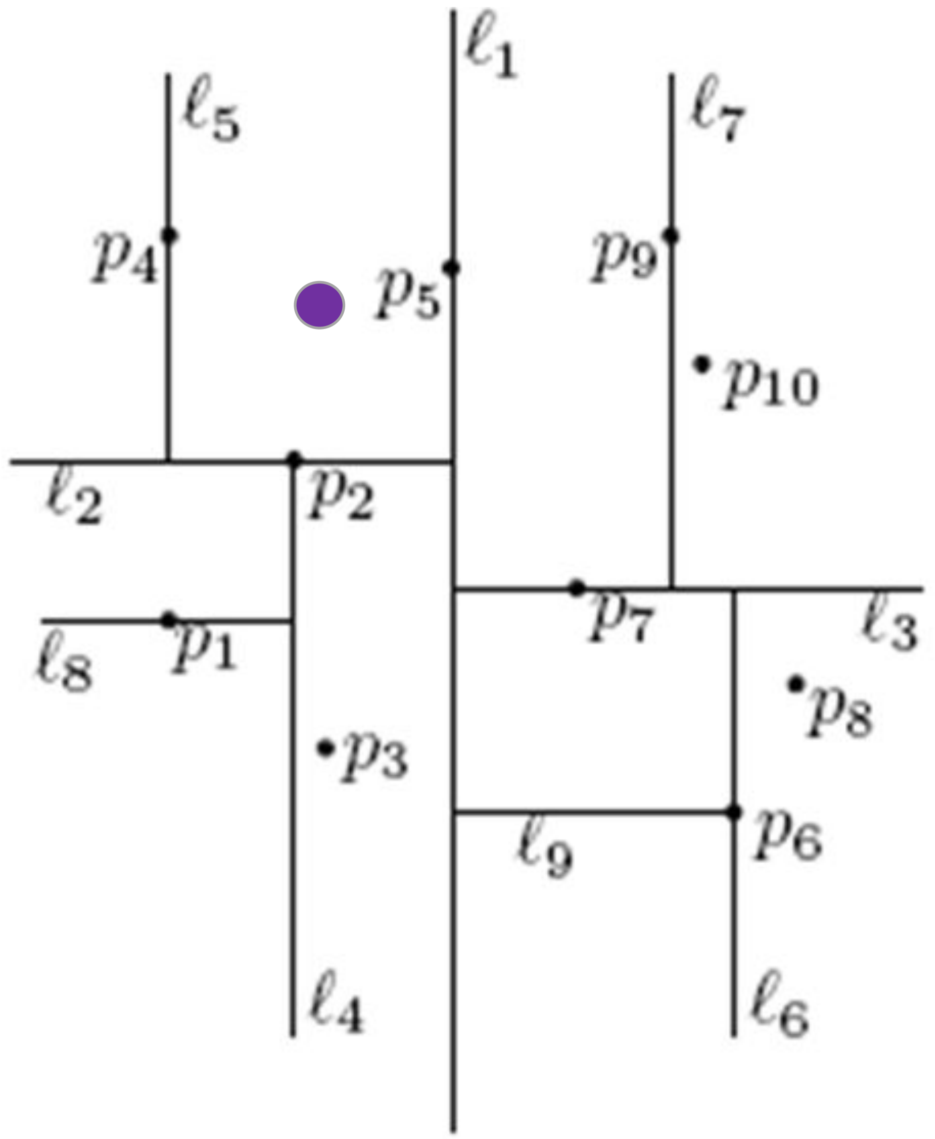


KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space

2) Need good initial estimates
   - E.g., P9 would be a very bad
   - Structure of KD-Tree helps
   - **“Search” of the query point in the query tree and take the leaf where the search terminates.**
   - **Works ok in practice**

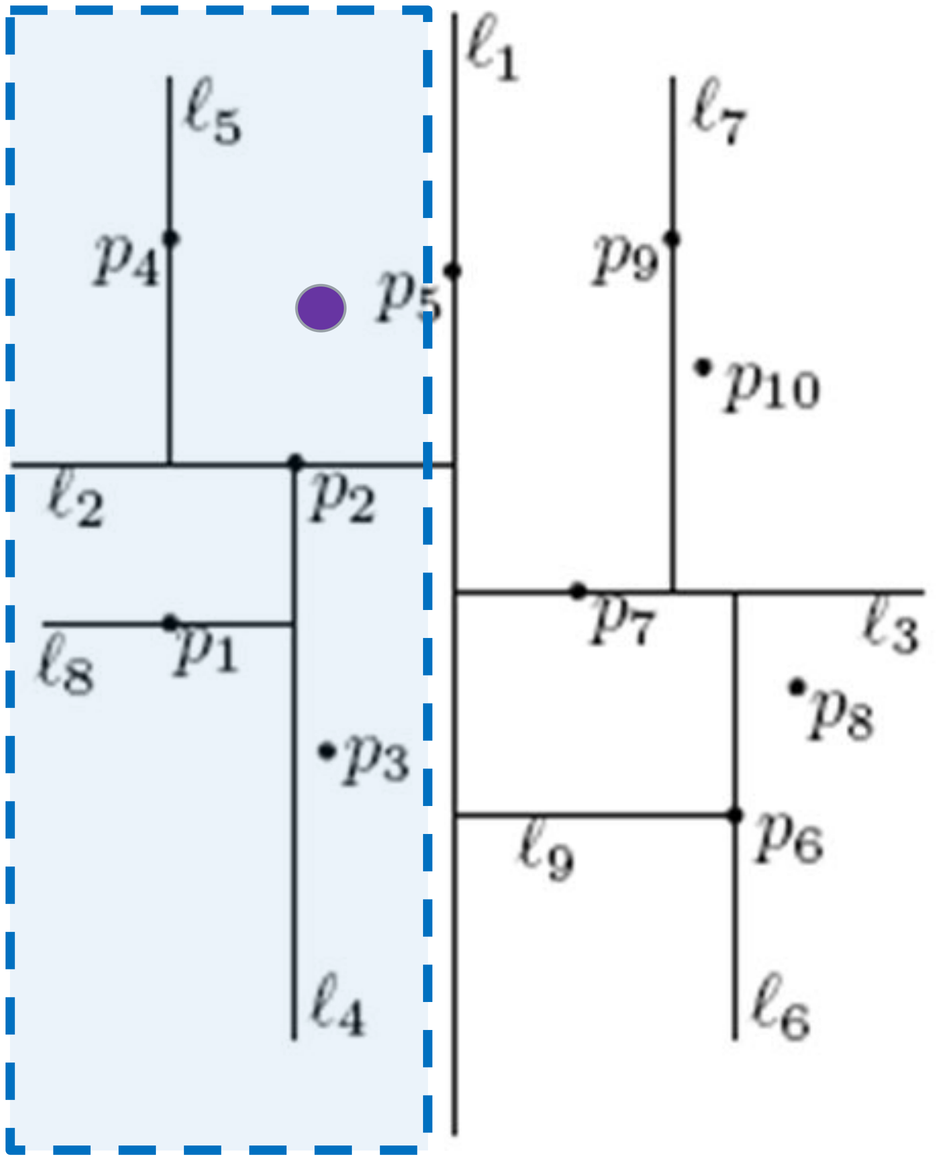


KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space

2) Need good initial estimates
   - E.g., P9 would be a very bad
   - Structure of KD-Tree helps
   - **"Search" of the query point in the query tree and take the leaf where the search terminates.**
   - **Works ok in practice**

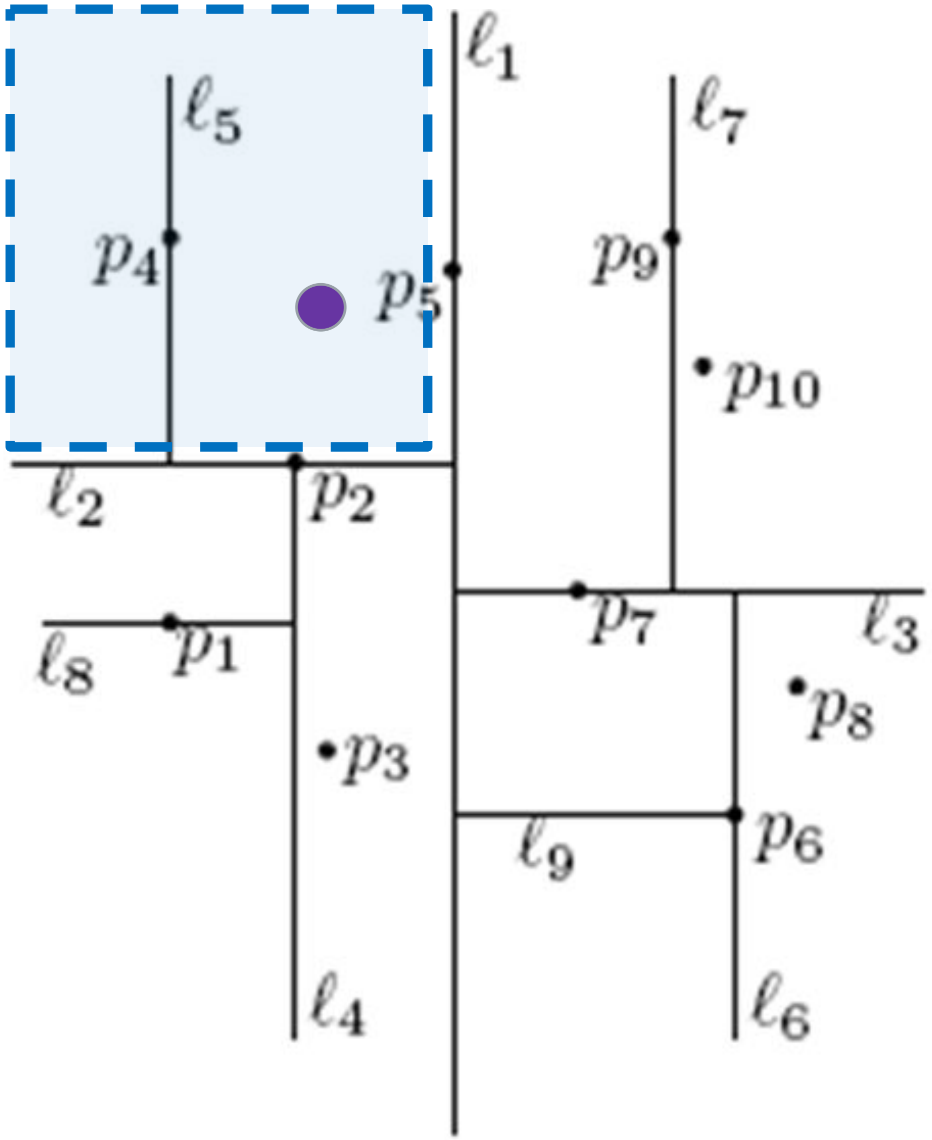


KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space

2) Need good initial estimates
   - E.g., P9 would be a very bad
   - Structure of KD-Tree helps
   - **"Search" of the query point in the query tree and take the leaf where the search terminates.**
   - **Works ok in practice**

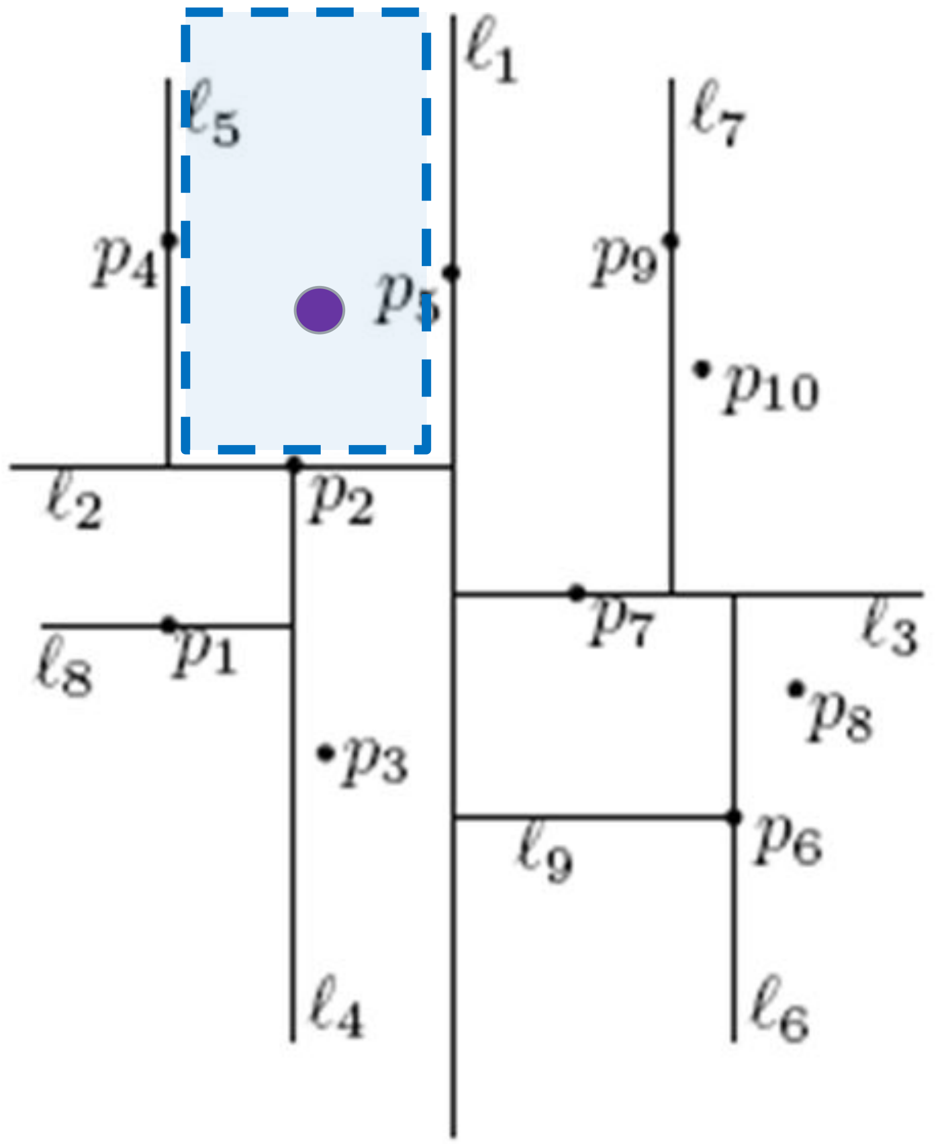


KD Trees – Nearest Neighbor Search

- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space

2) Need good initial estimates
   - E.g., P9 would be a very bad
   - Structure of KD-Tree helps
   - **"Search" of the query point in the query tree and take the leaf where the search terminates.**
   - **Works ok in practice**

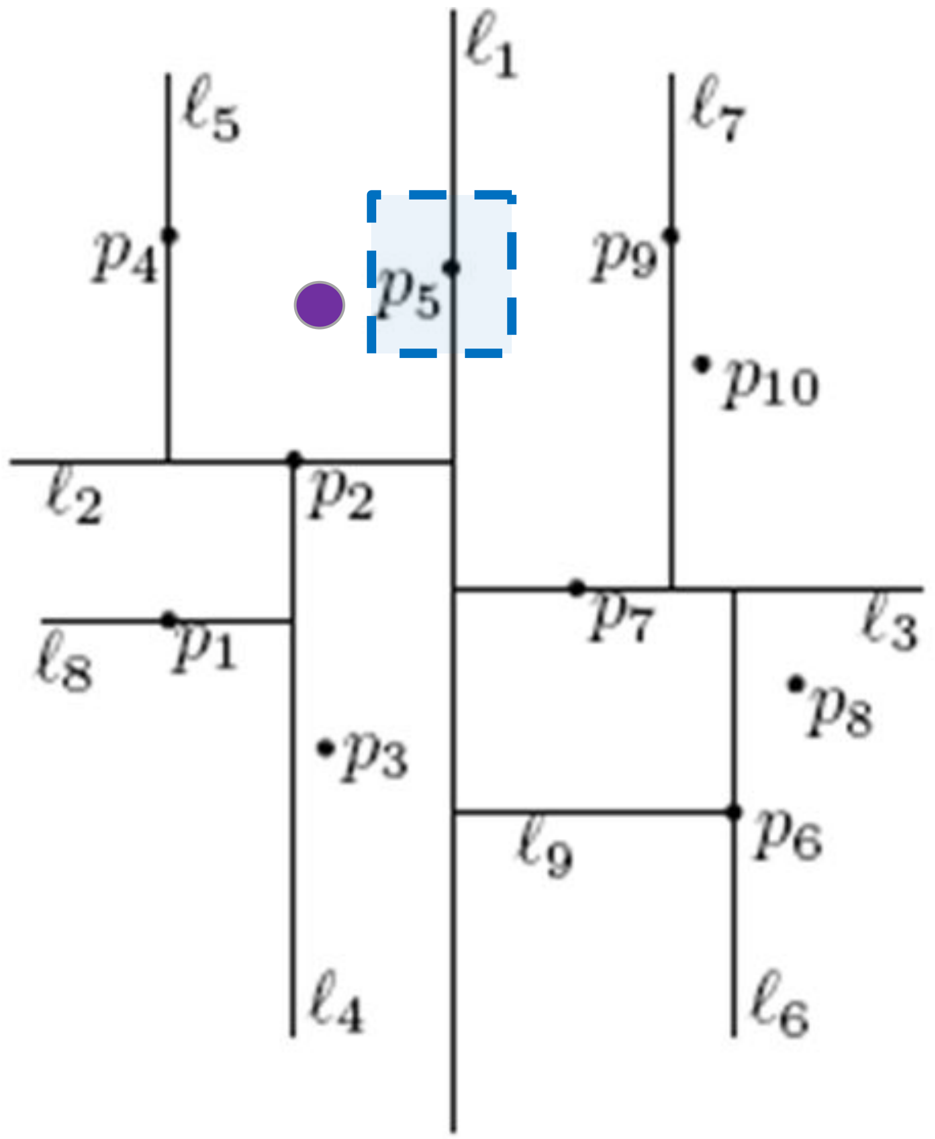


KD Trees – Nearest Neighbor Search

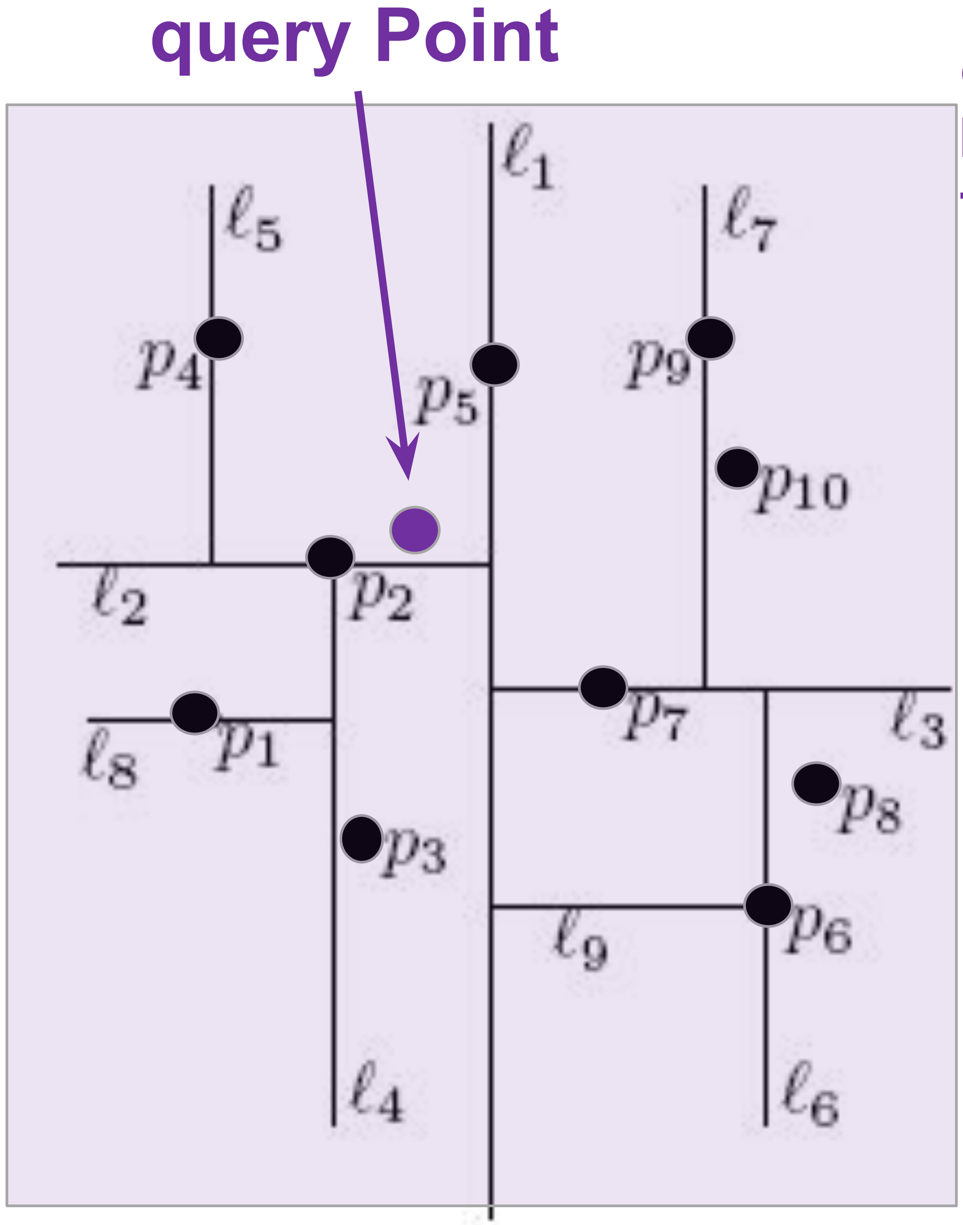
- **Two things to consider for such a approach:**

1) Need a smart way to prune through the data space

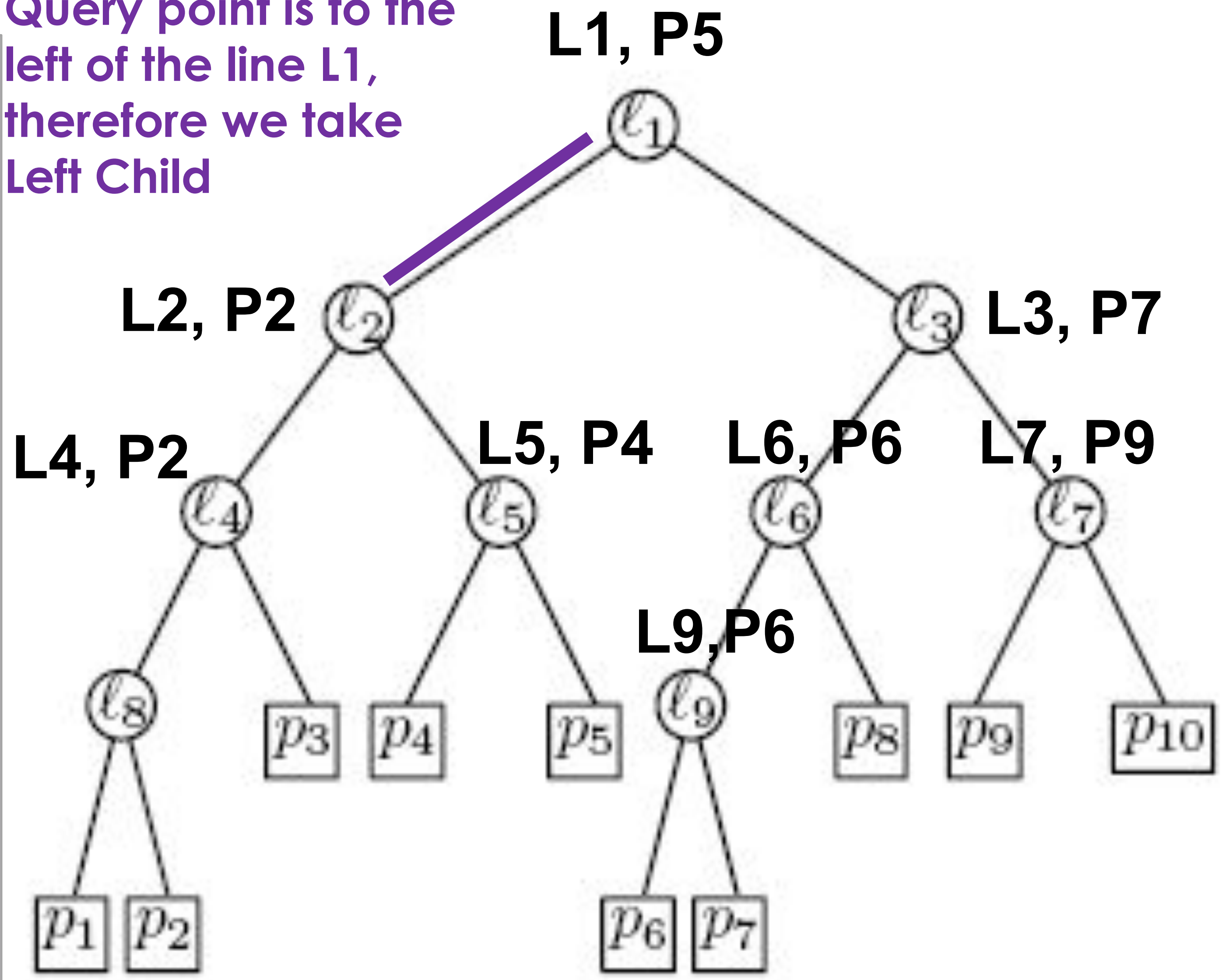
2) Need good initial estimates
   - E.g., P9 would be a very bad
   - Structure of KD-Tree helps
   - **"Search" of the query point in the query tree and take the leaf where the search terminates.**
   - **Works ok in practice**



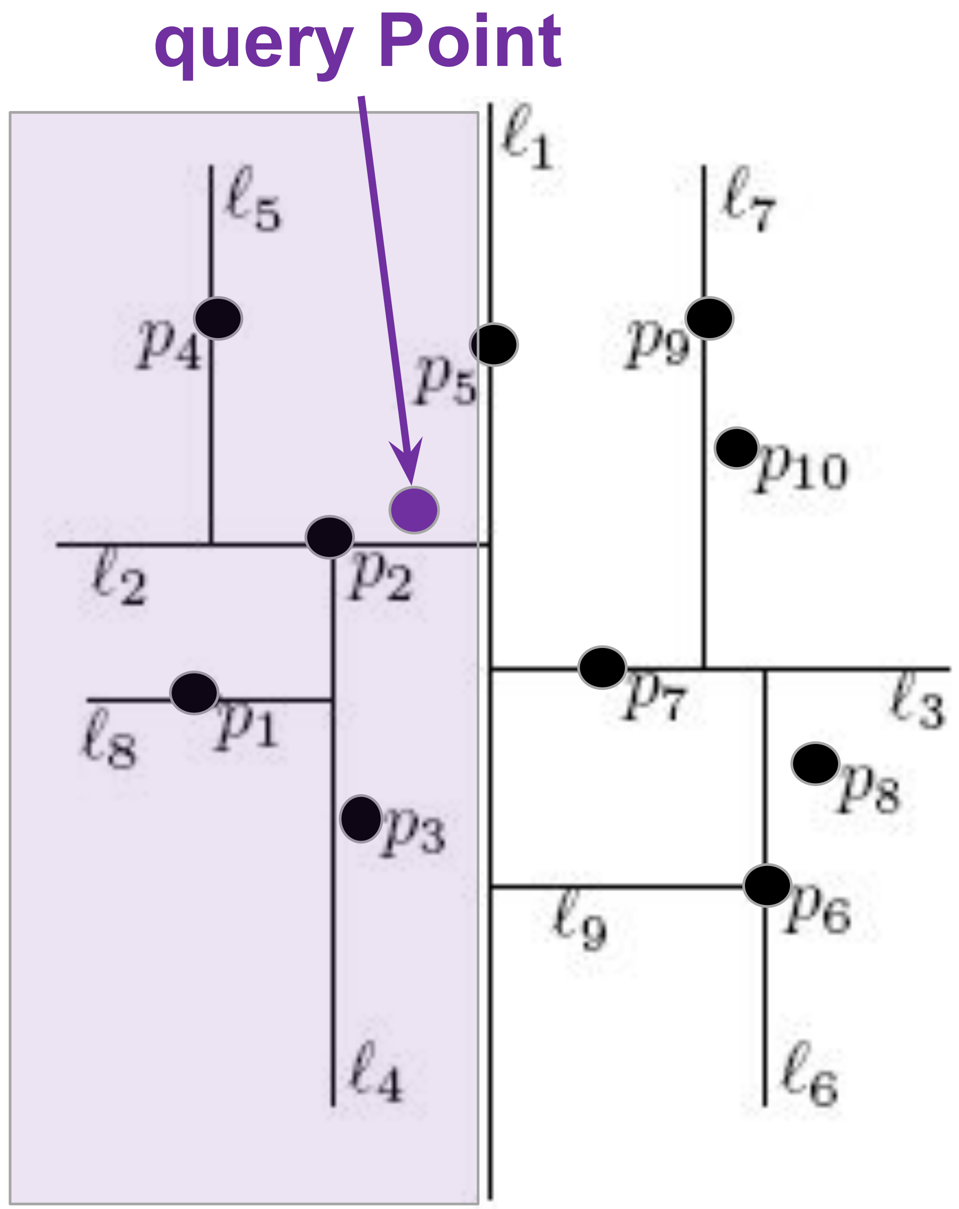
KD-tree: 1-NN Query Running



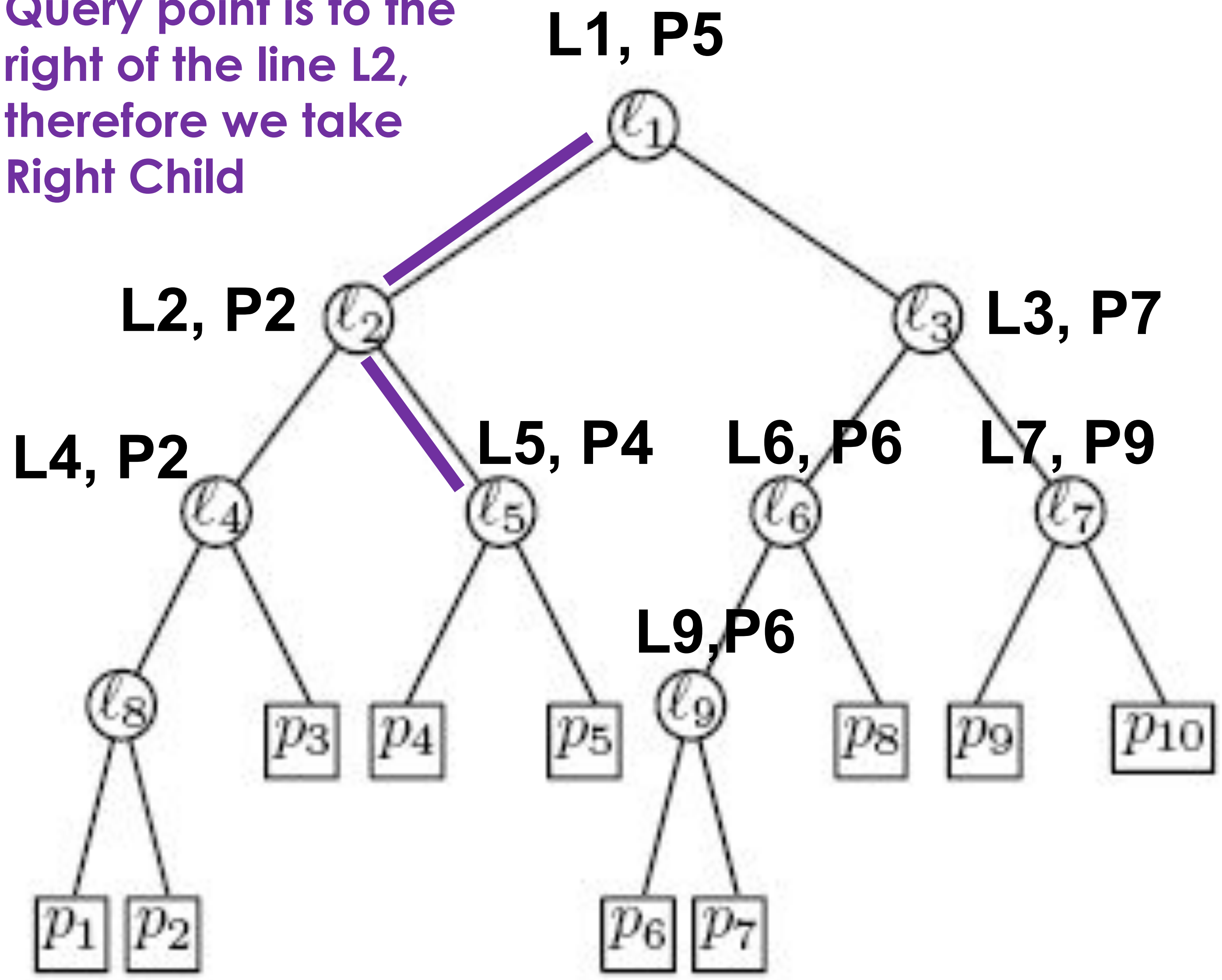
Query point is to the left of the line L1, therefore we take Left Child



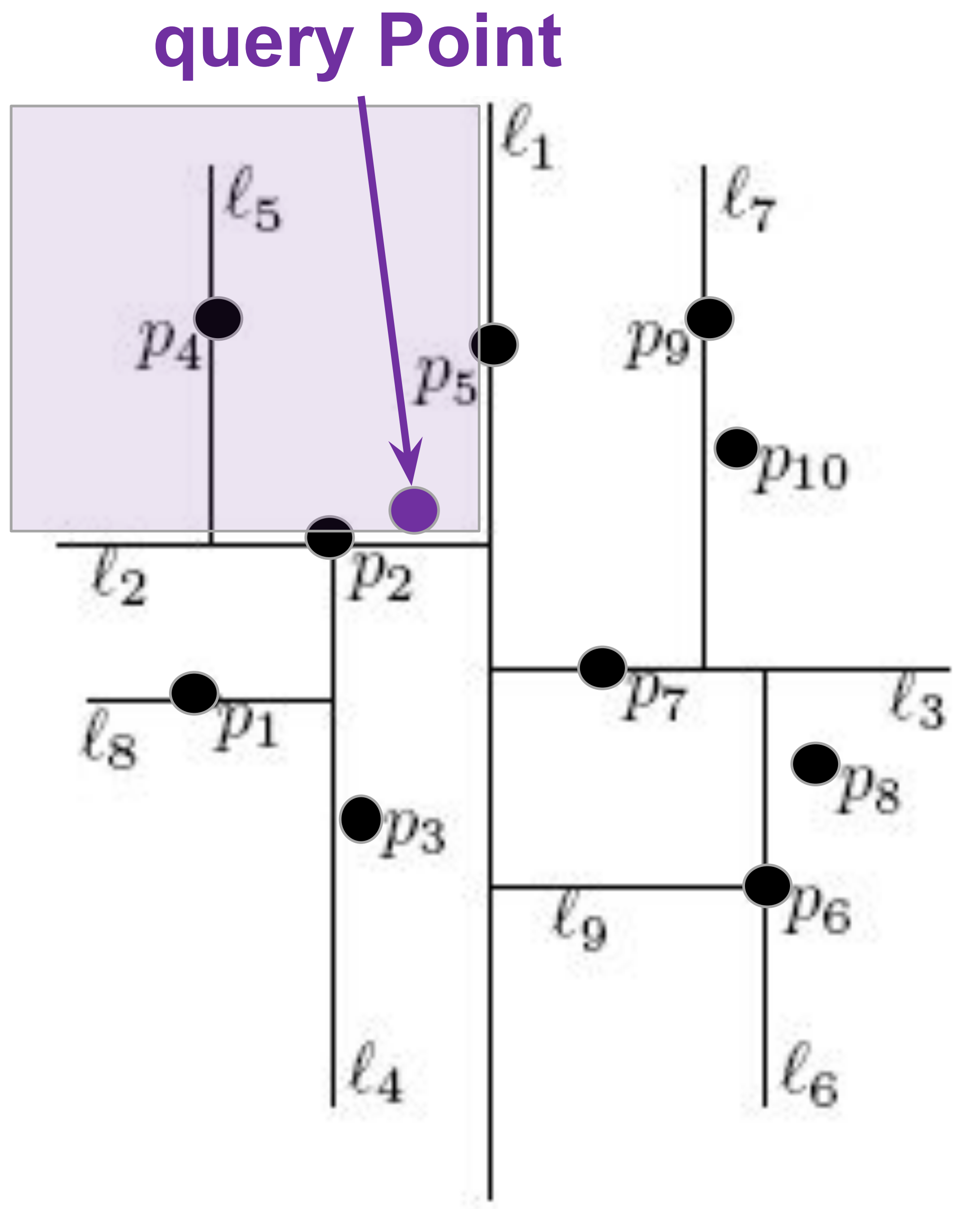
KD-tree: 1-NN Query Running



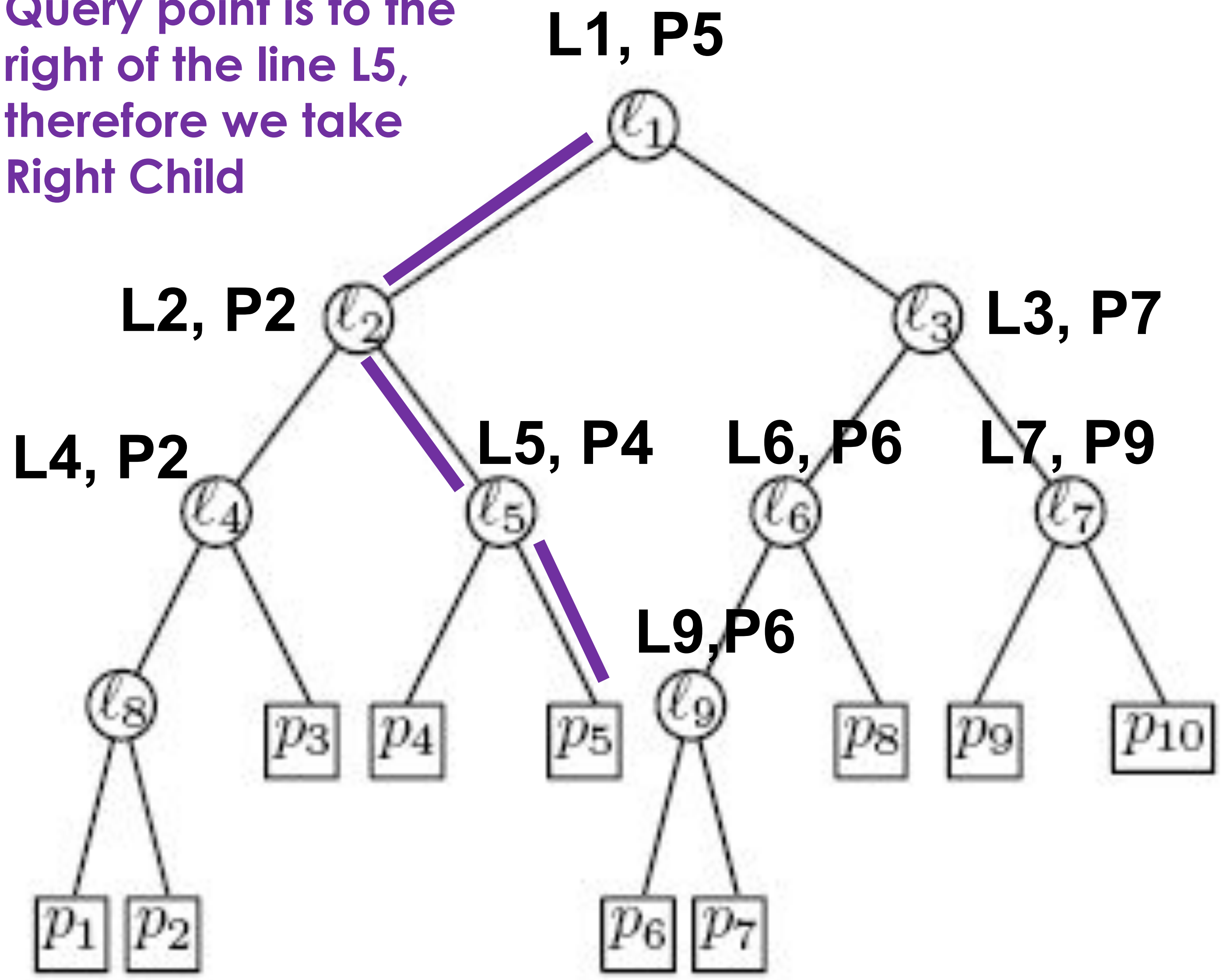
Query point is to the right of the line L2, therefore we take Right Child



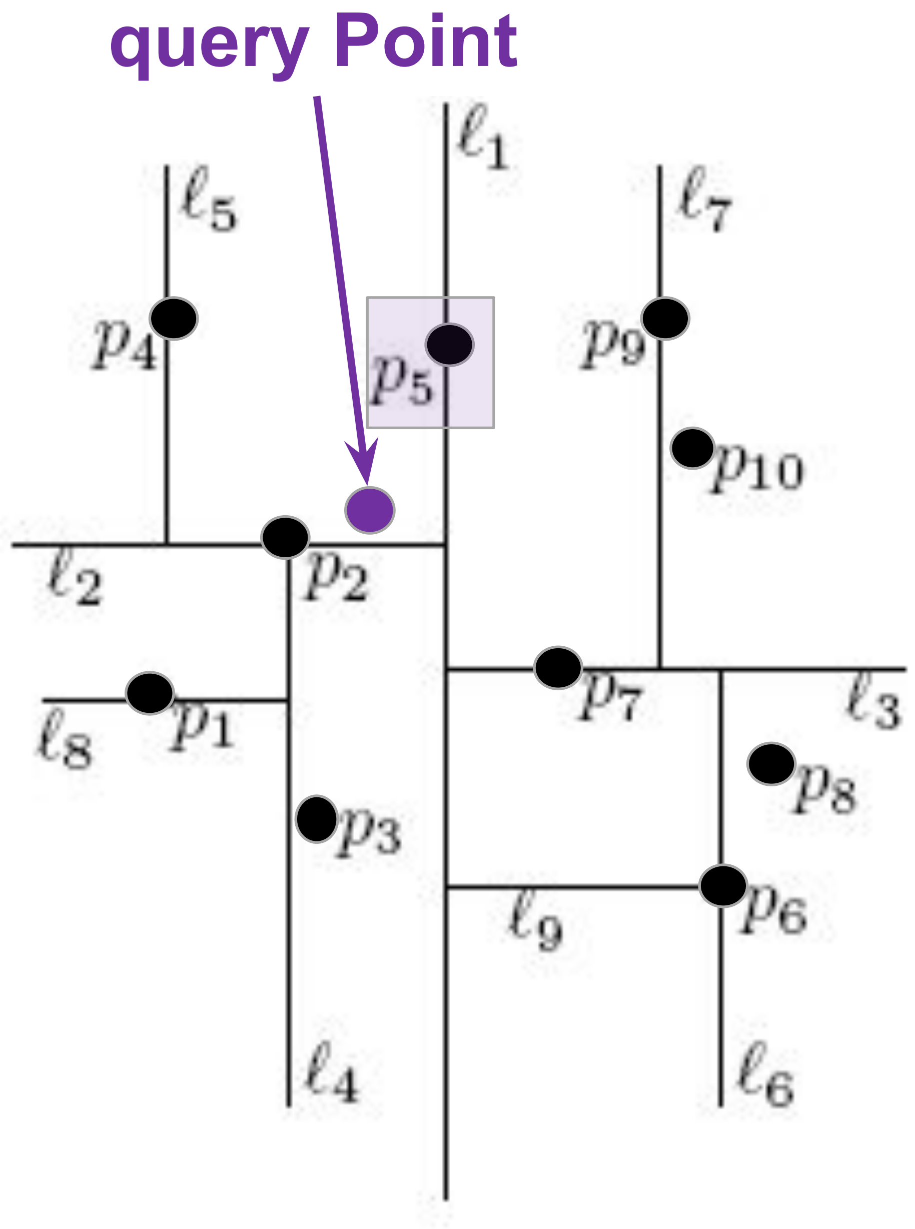
KD-tree: 1-NN Query Running



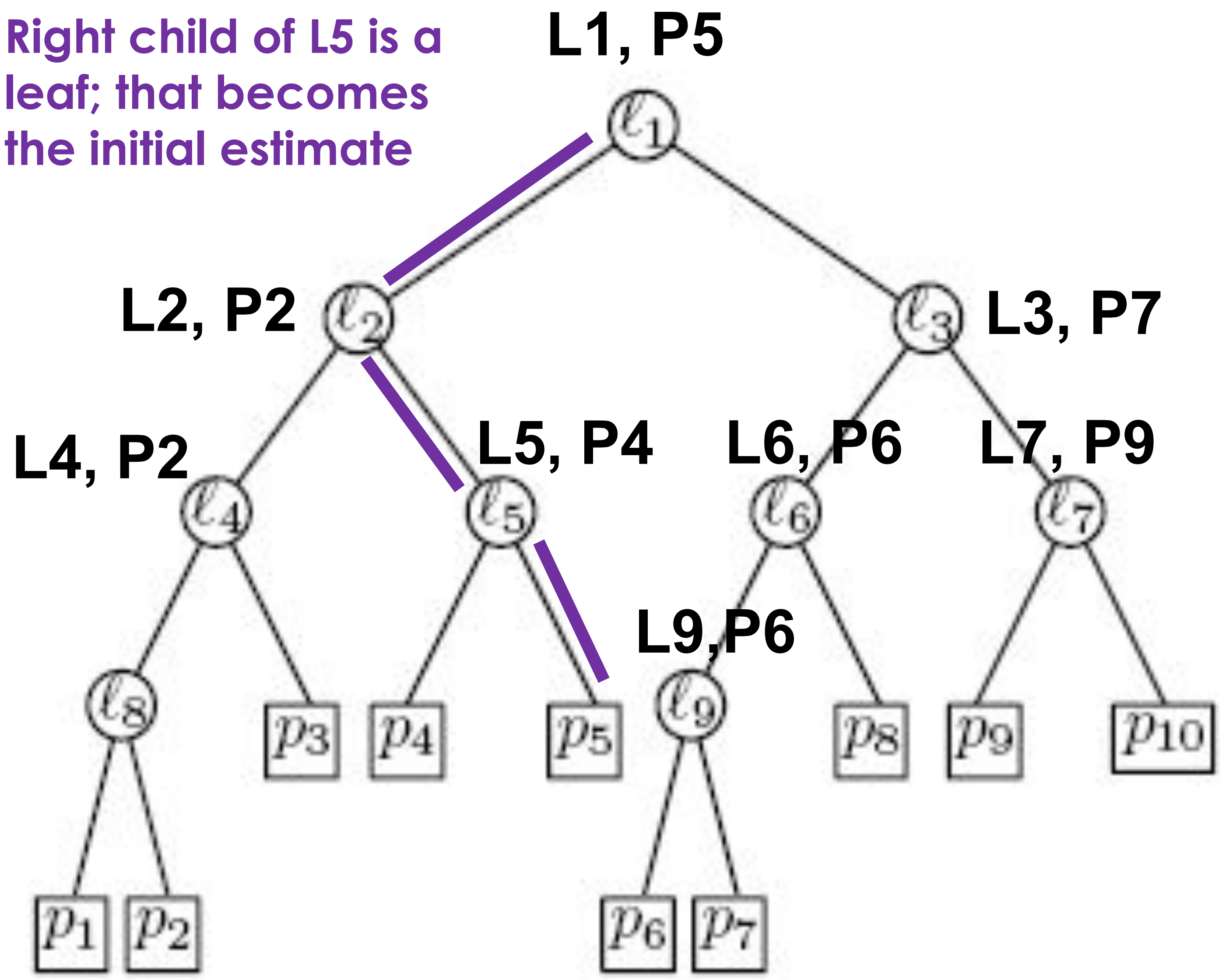
Query point is to the right of the line L5, therefore we take Right Child



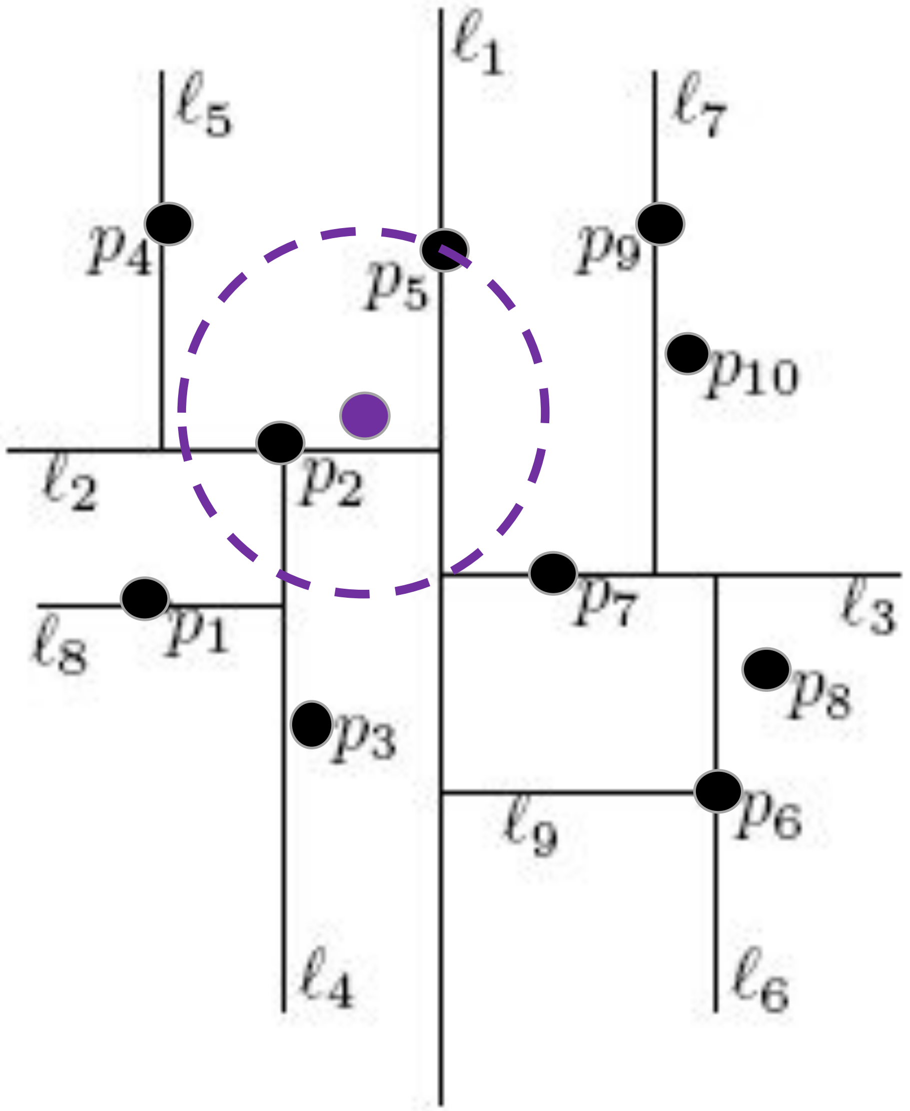
KD-tree: 1-NN Query Running



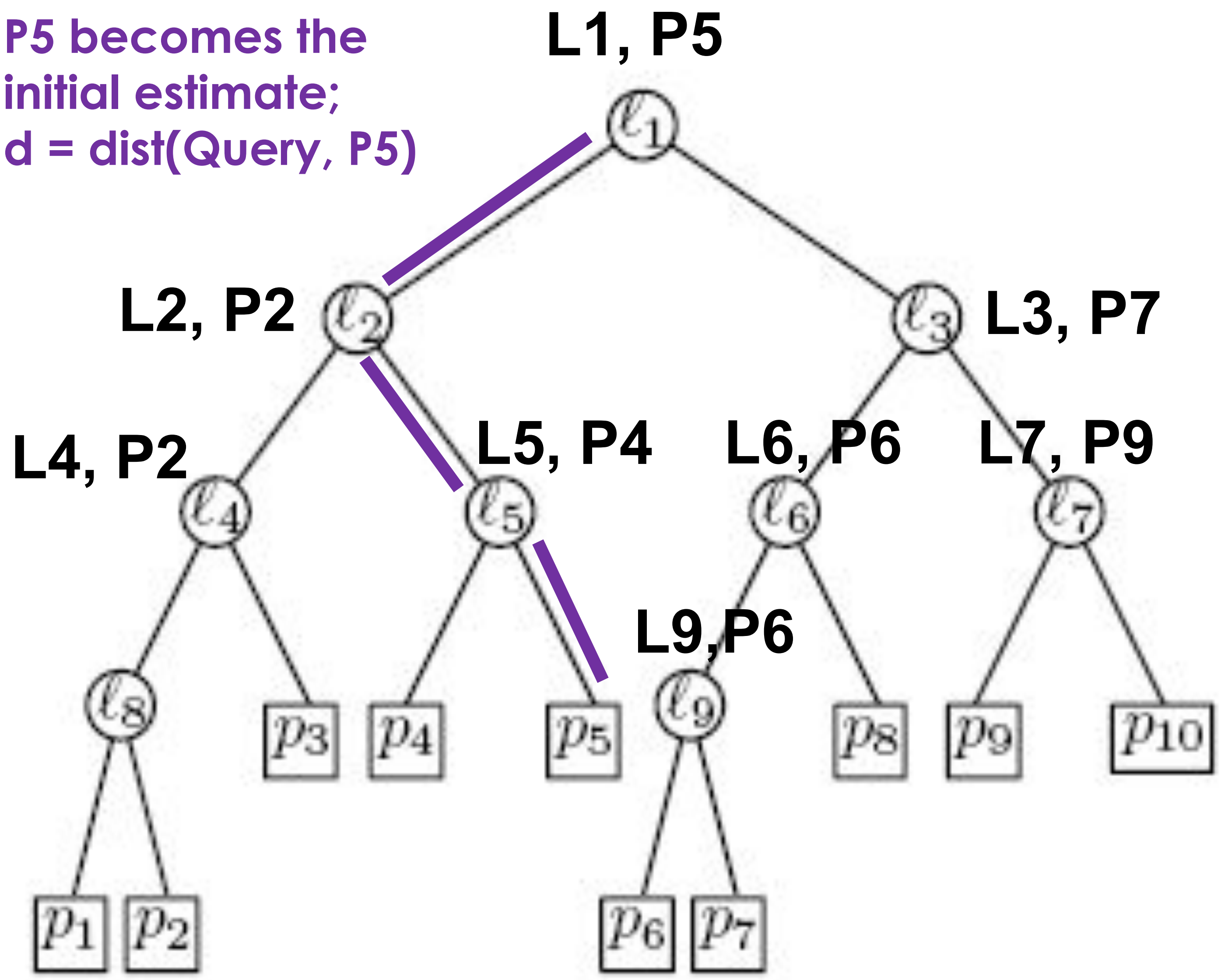
Right child of L5 is a leaf; that becomes the initial estimate



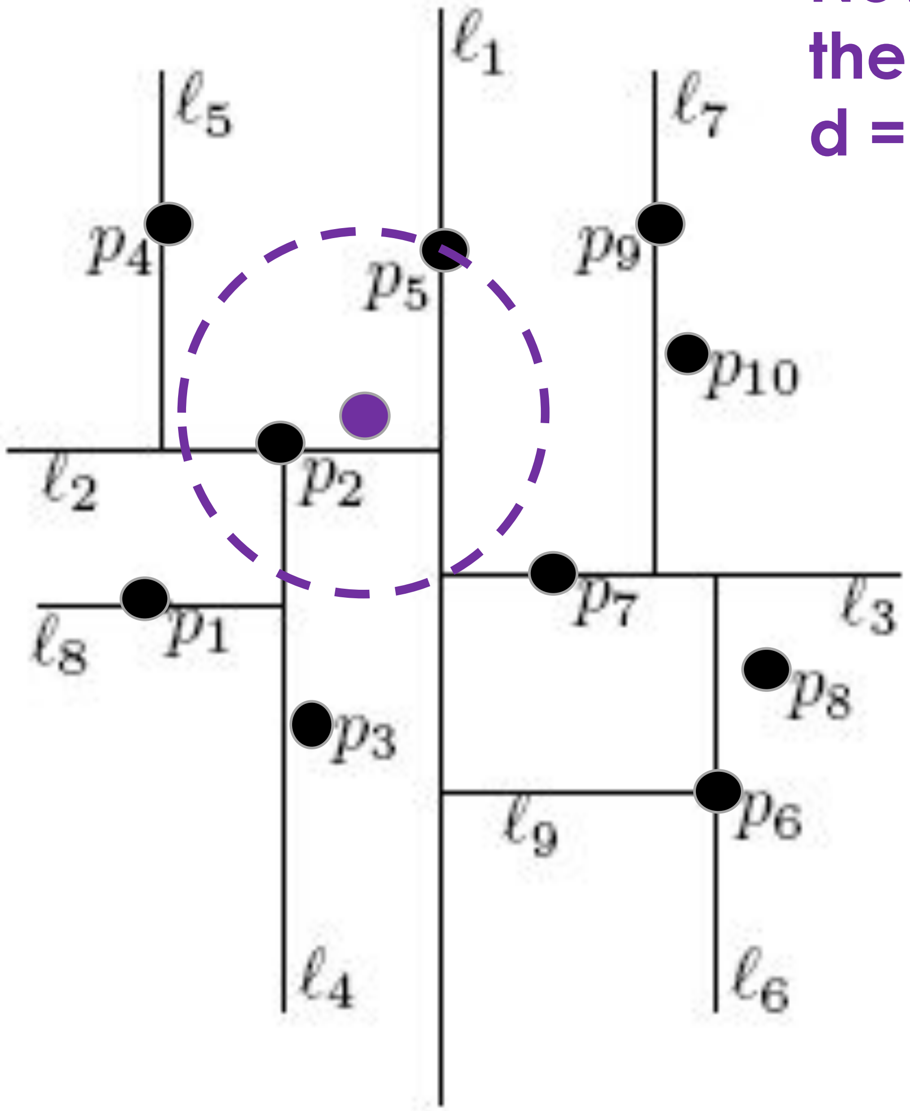
KD-tree: 1-NN Query Running



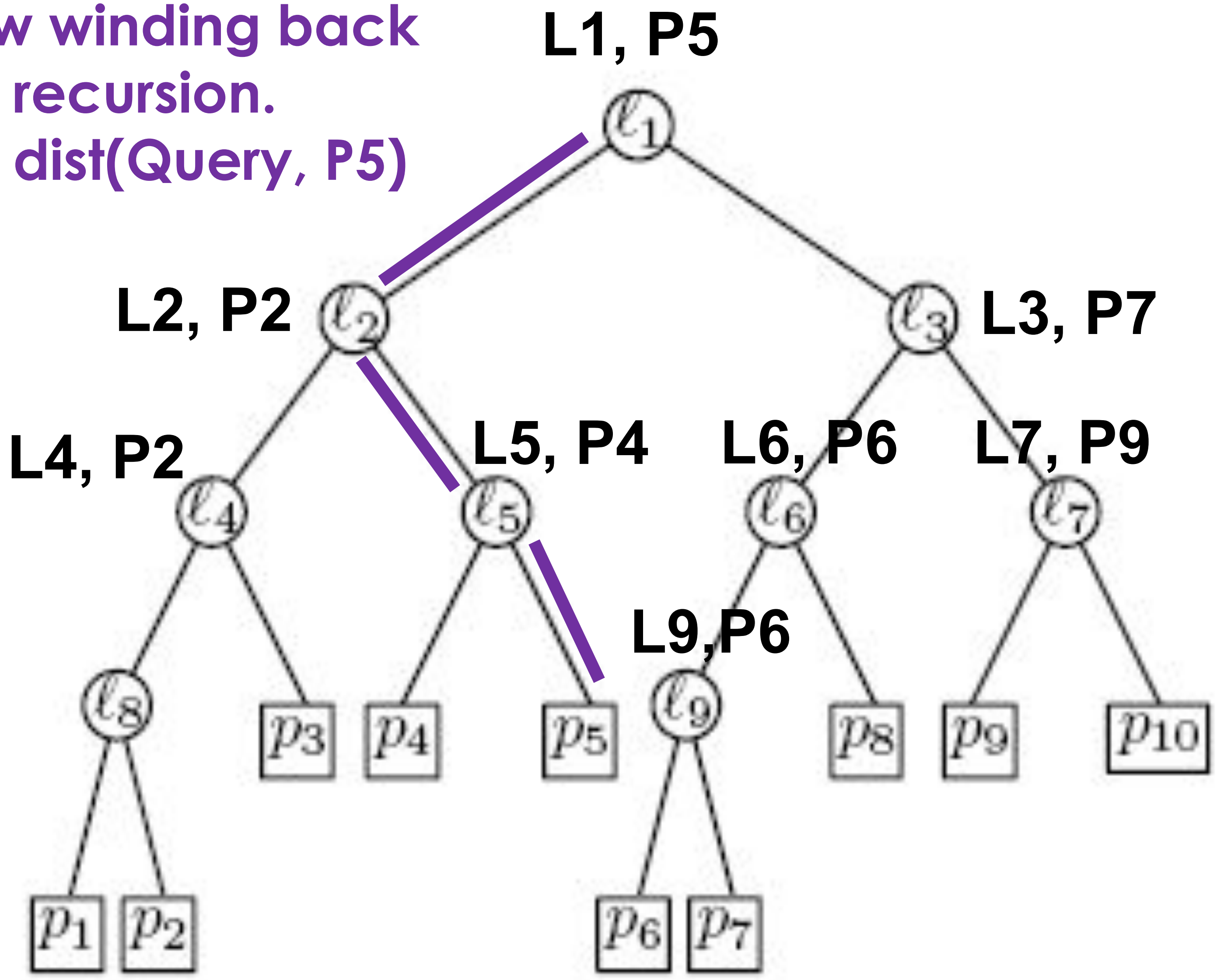
P5 becomes the initial estimate;
d = dist(Query, P5)



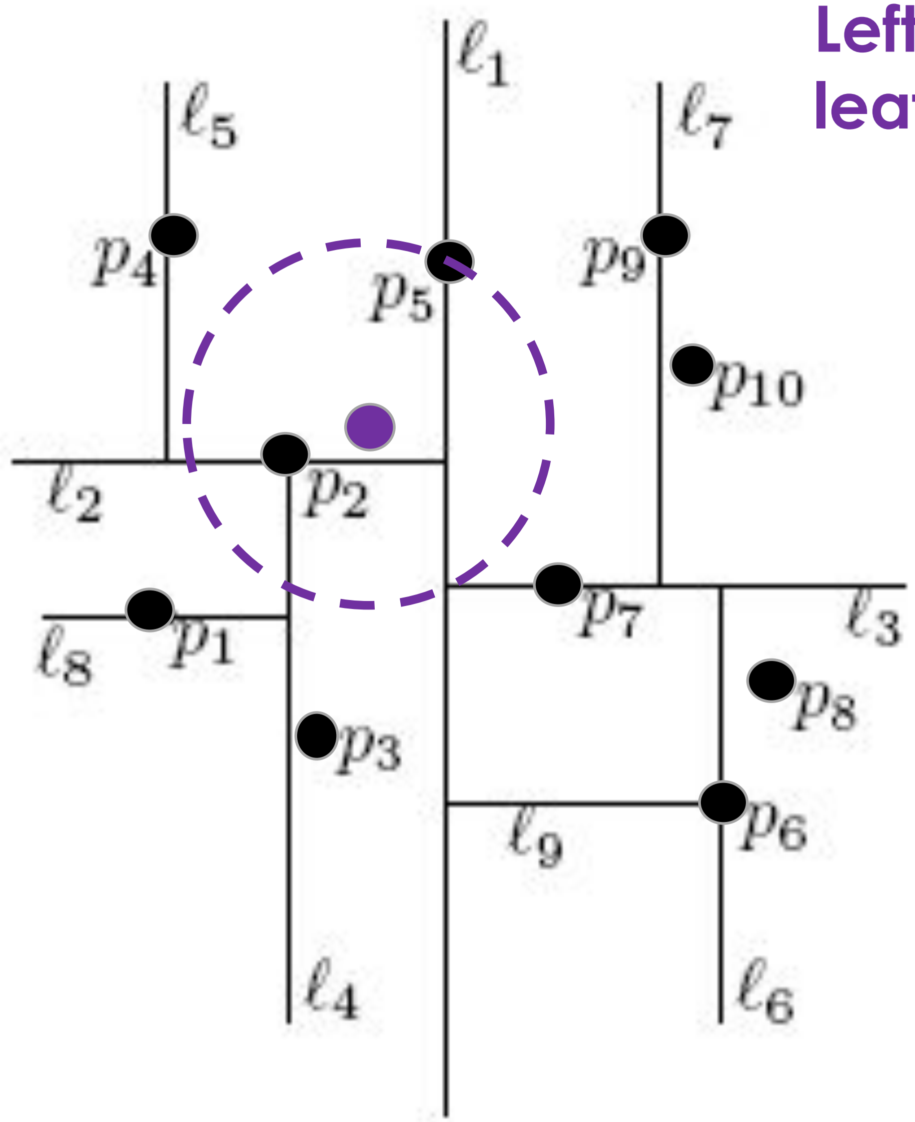
KD-tree: 1-NN Query Running



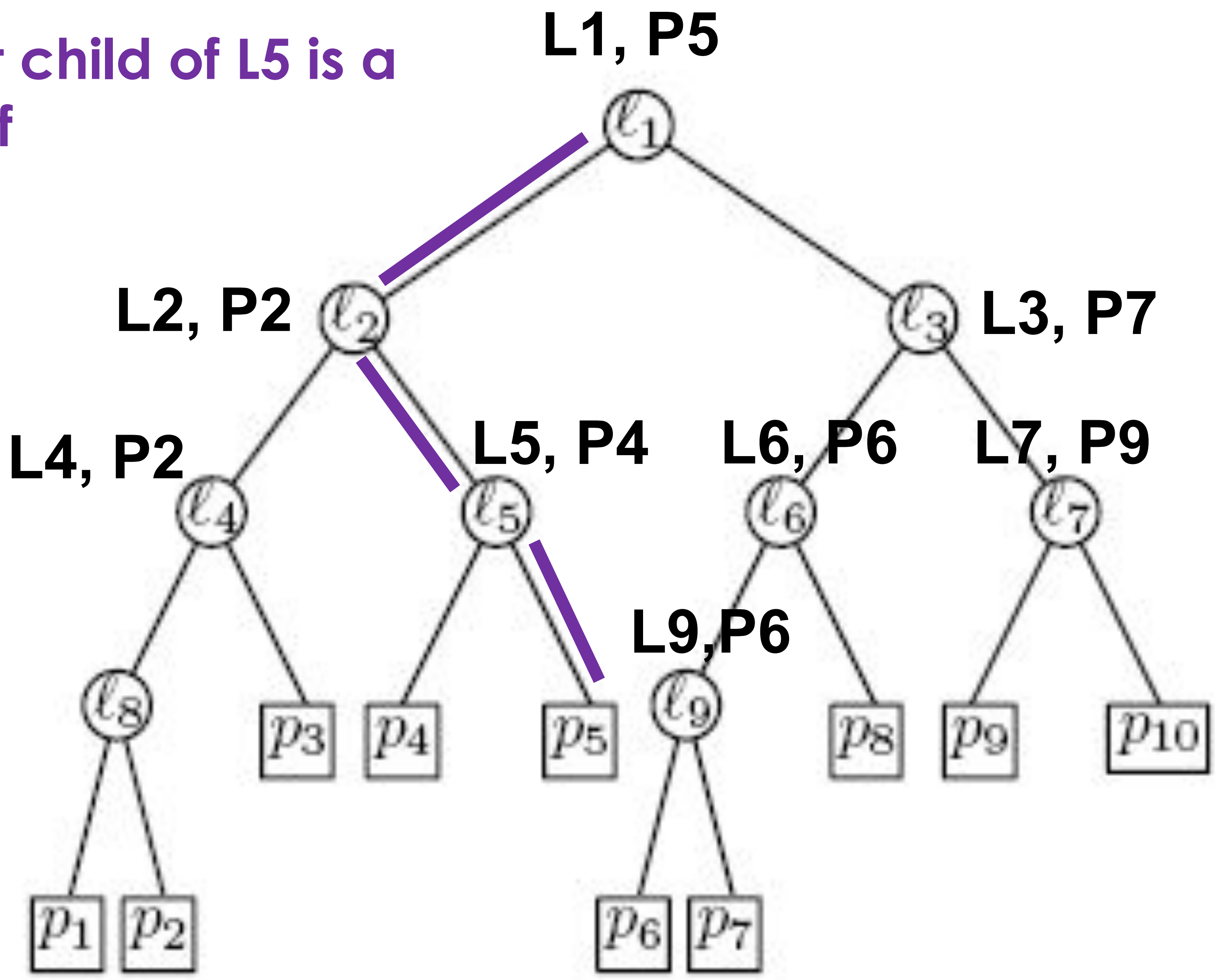
Now winding back the recursion.
d = dist(Query, P5)



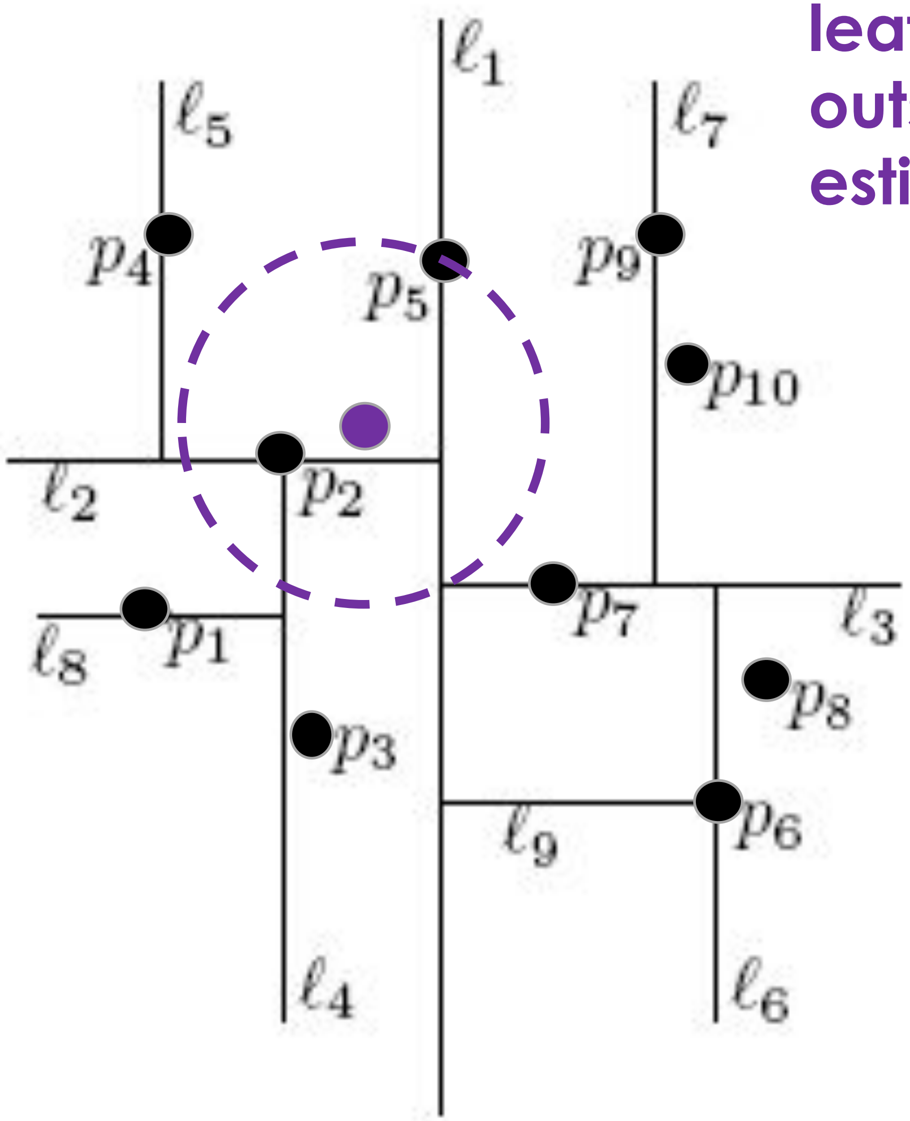
KD-tree: 1-NN Query Running



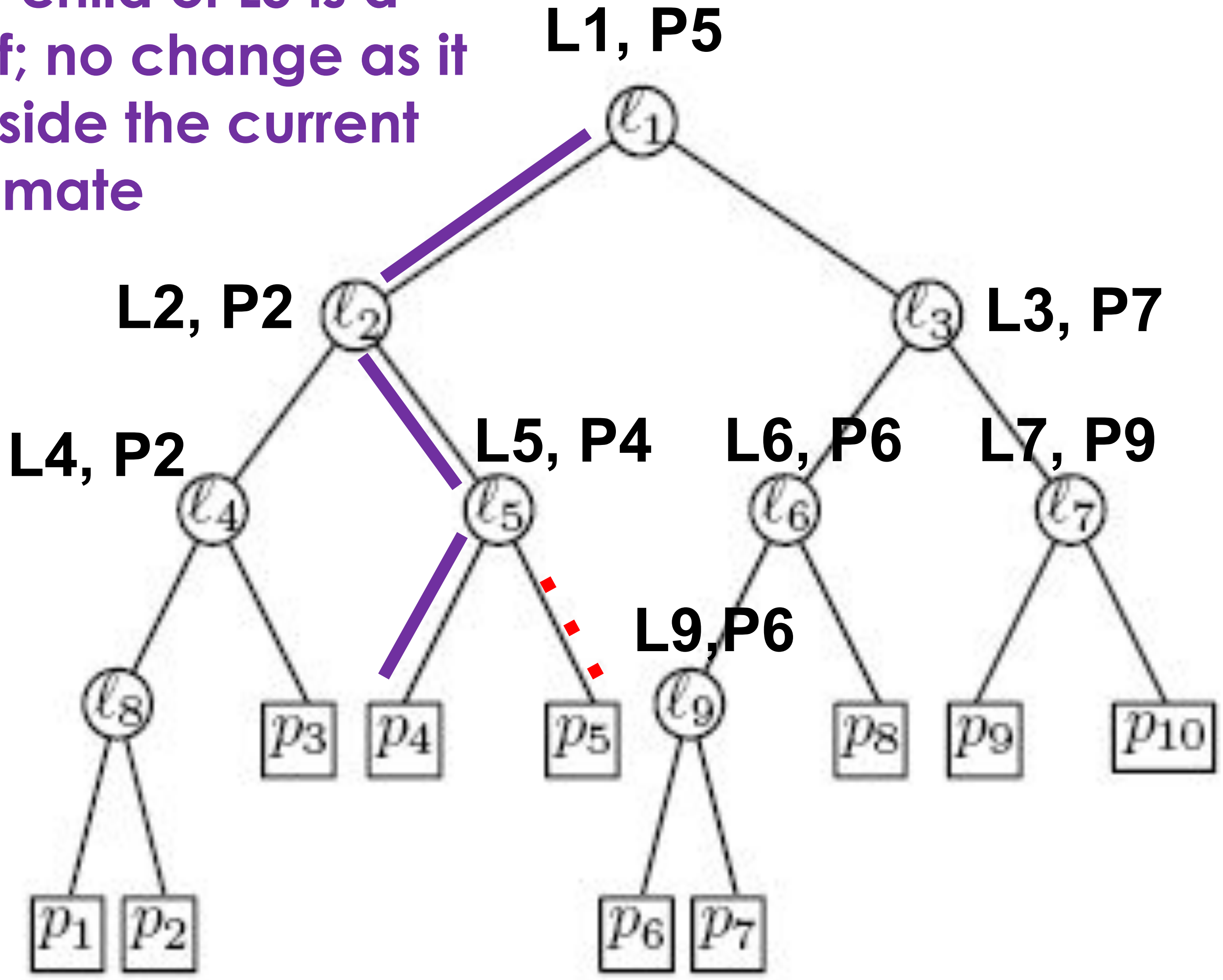
Left child of L5 is a leaf



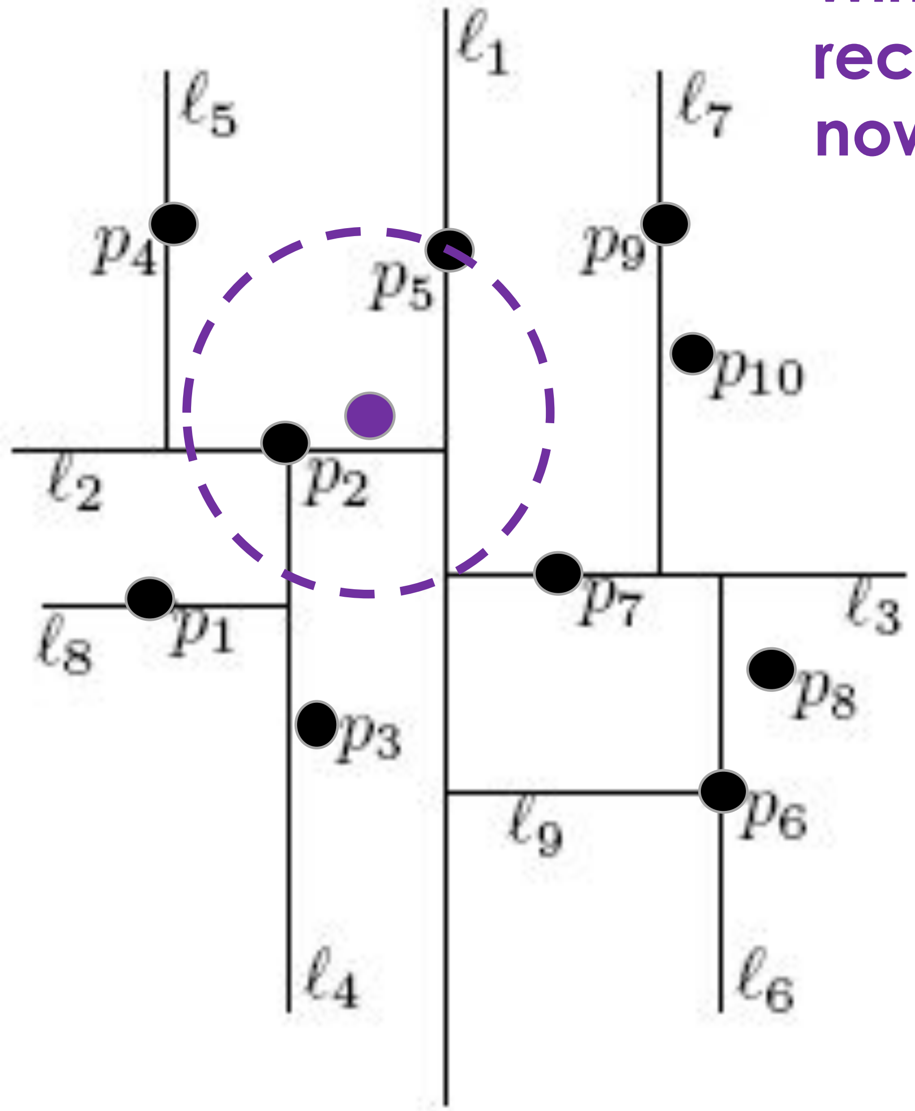
KD-tree: 1-NN Query Running



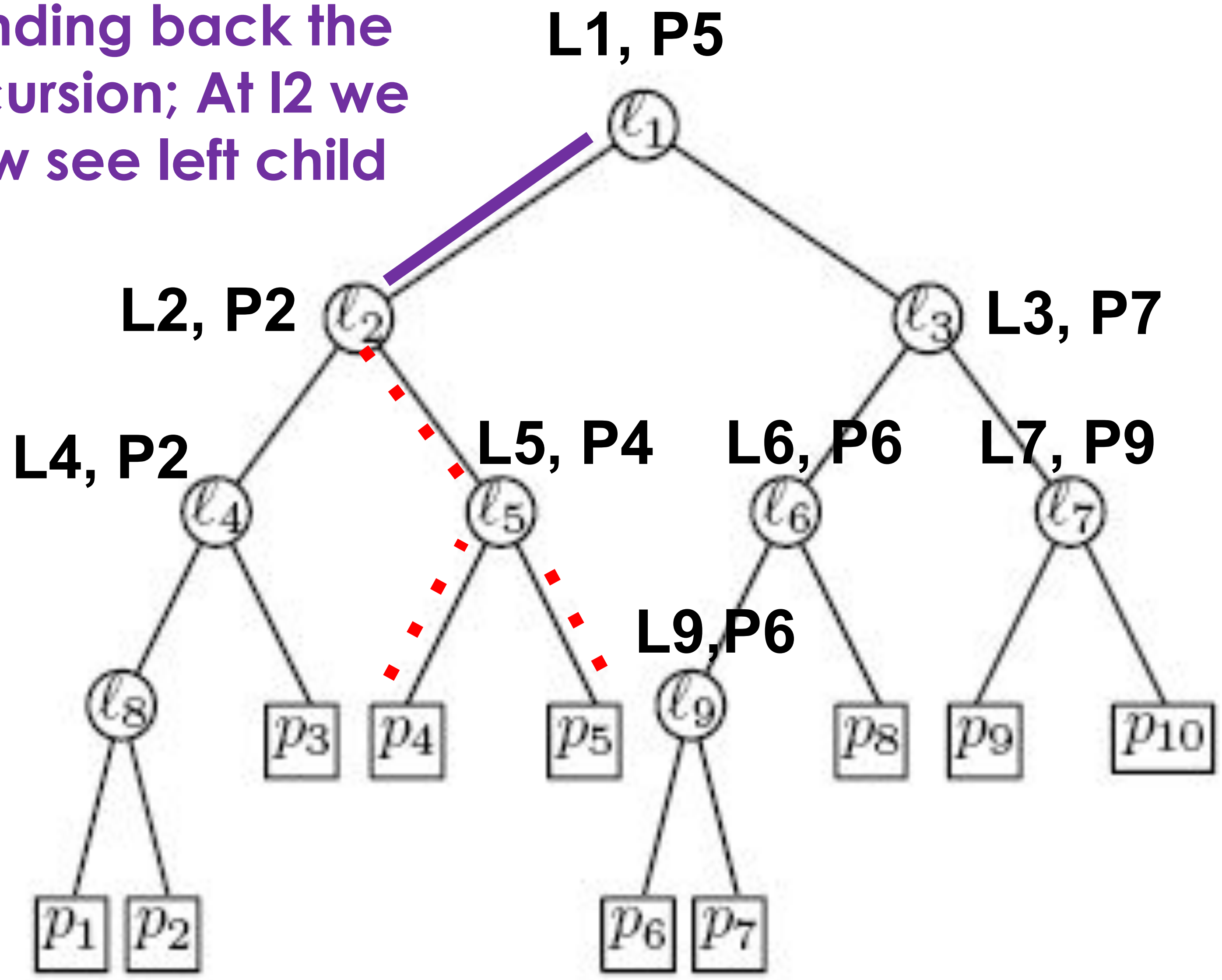
Left child of L5 is a leaf; no change as it outside the current estimate



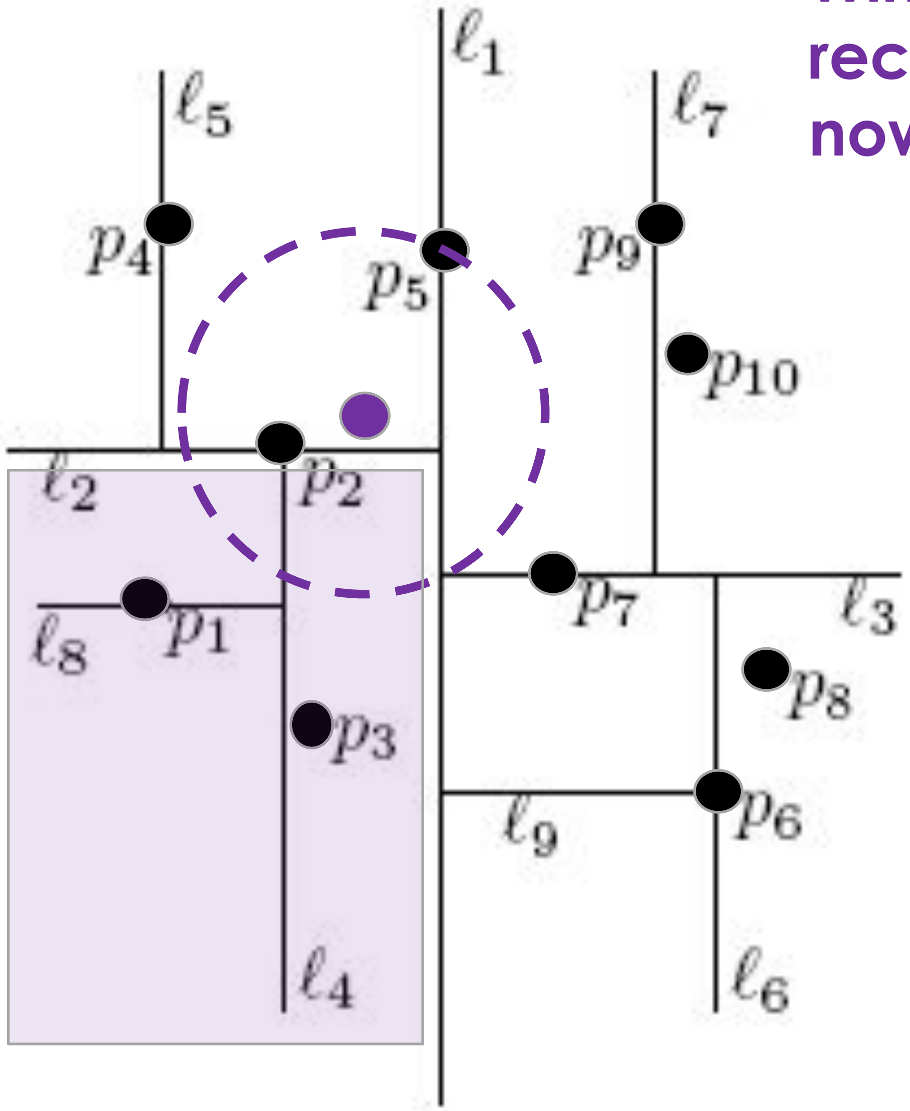
KD-tree: 1-NN Query Running



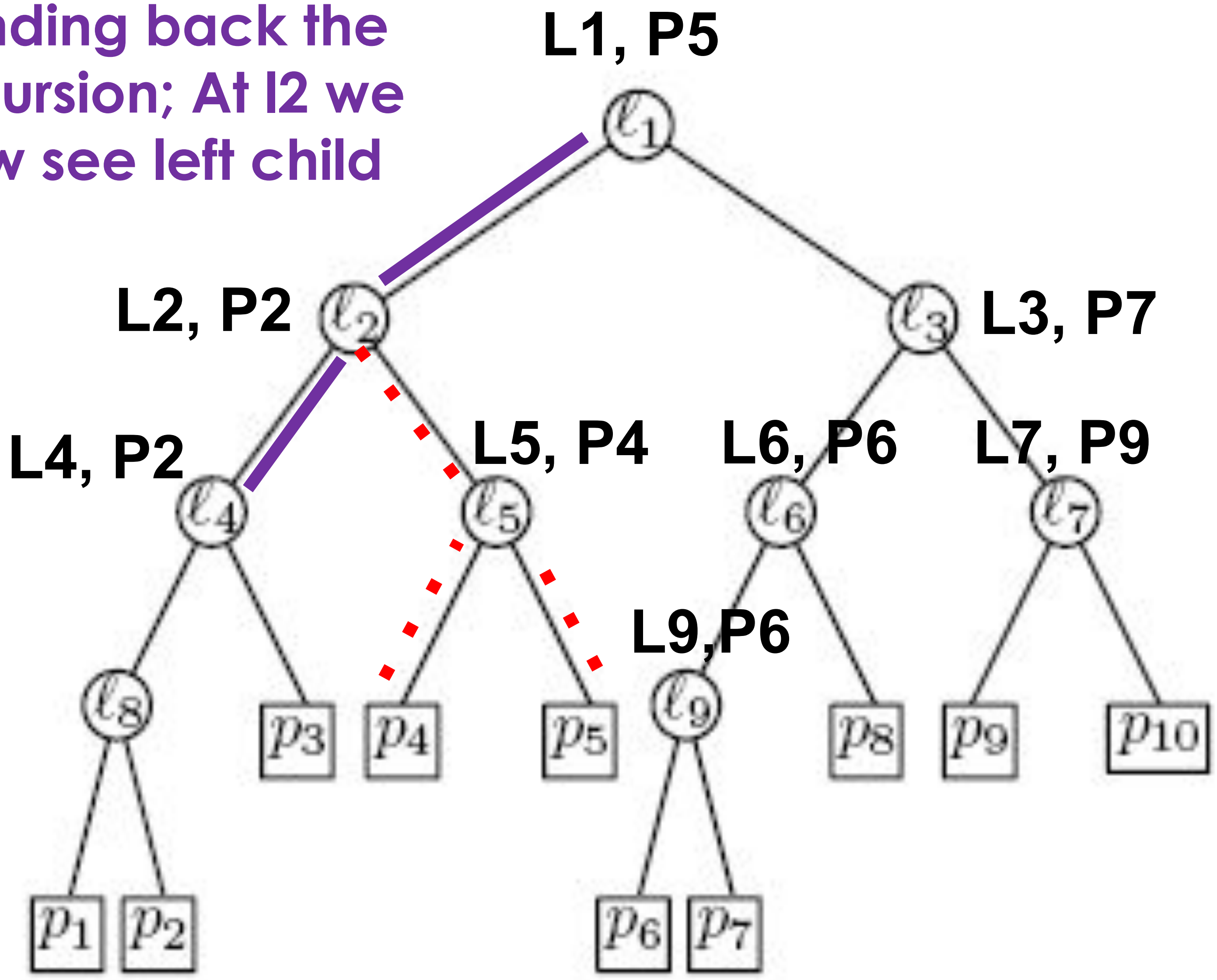
Winding back the recursion; At l2 we now see left child



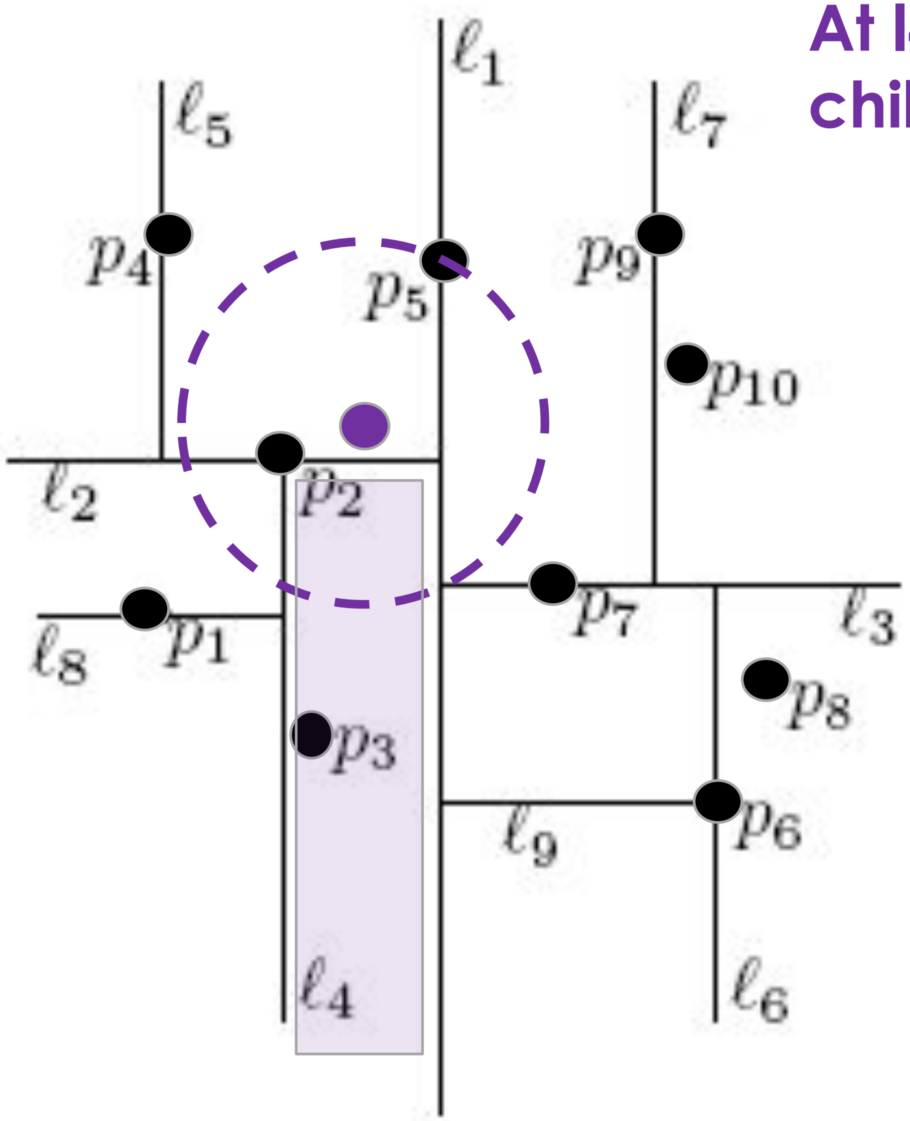
KD-tree: 1-NN Query Running



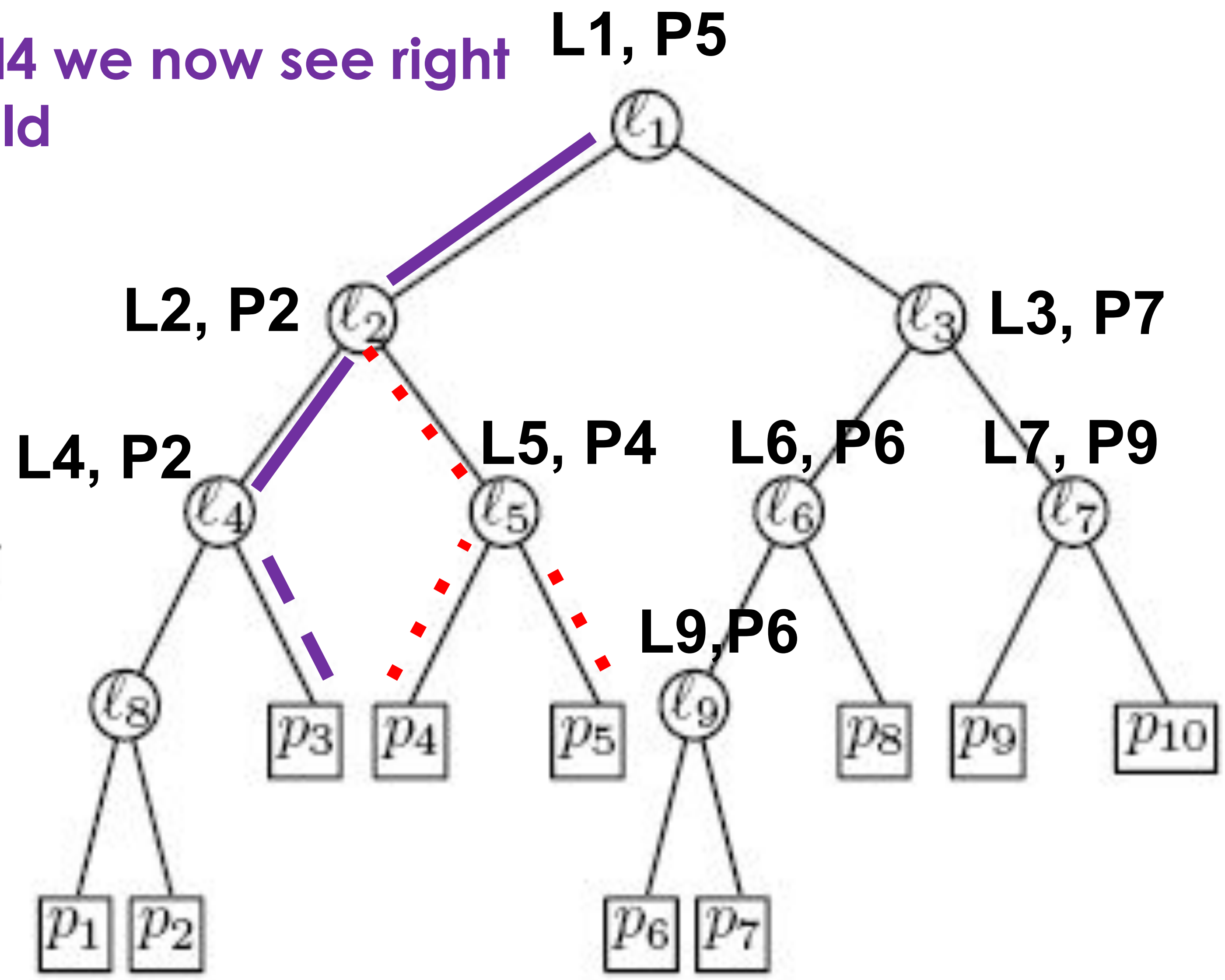
Winding back the recursion; At l2 we now see left child



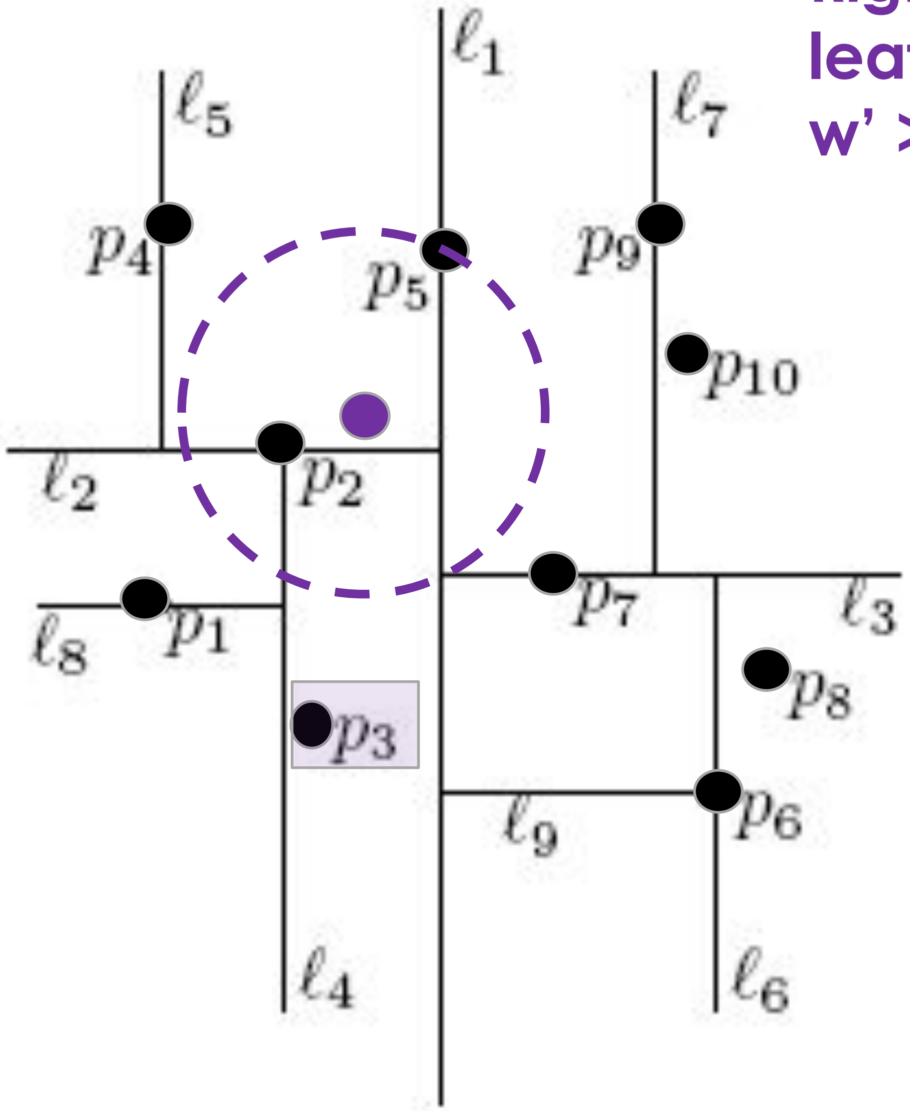
KD-tree: 1-NN Query Running



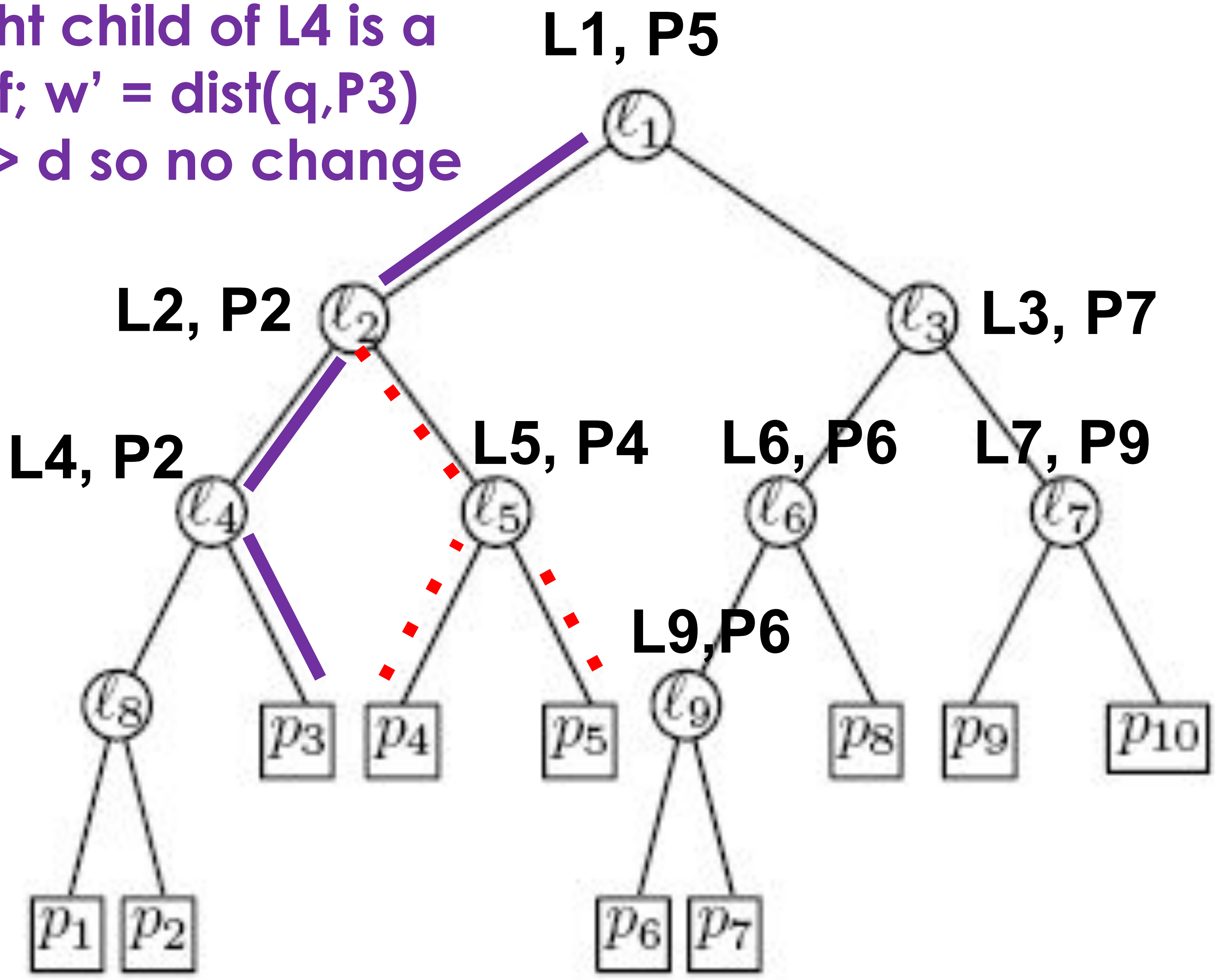
At l4 we now see right child



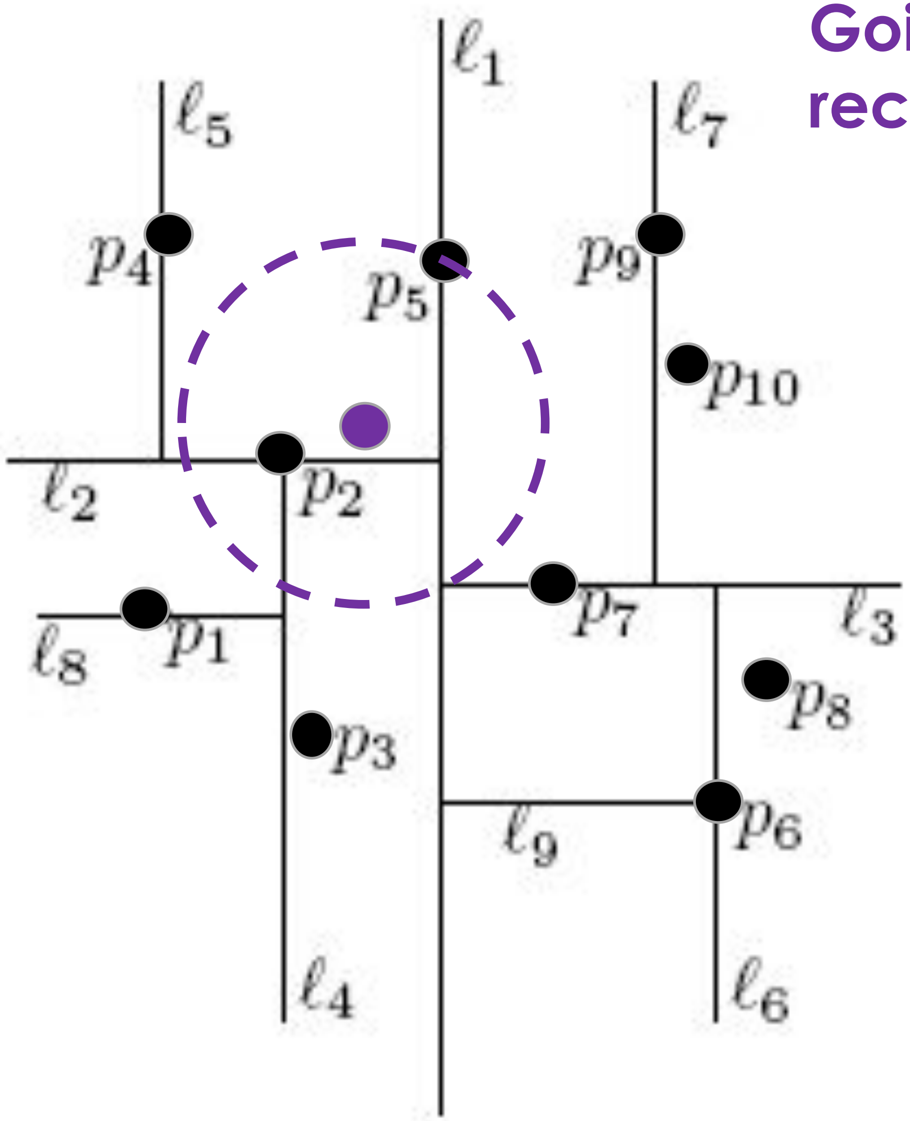
KD-tree: 1-NN Query Running



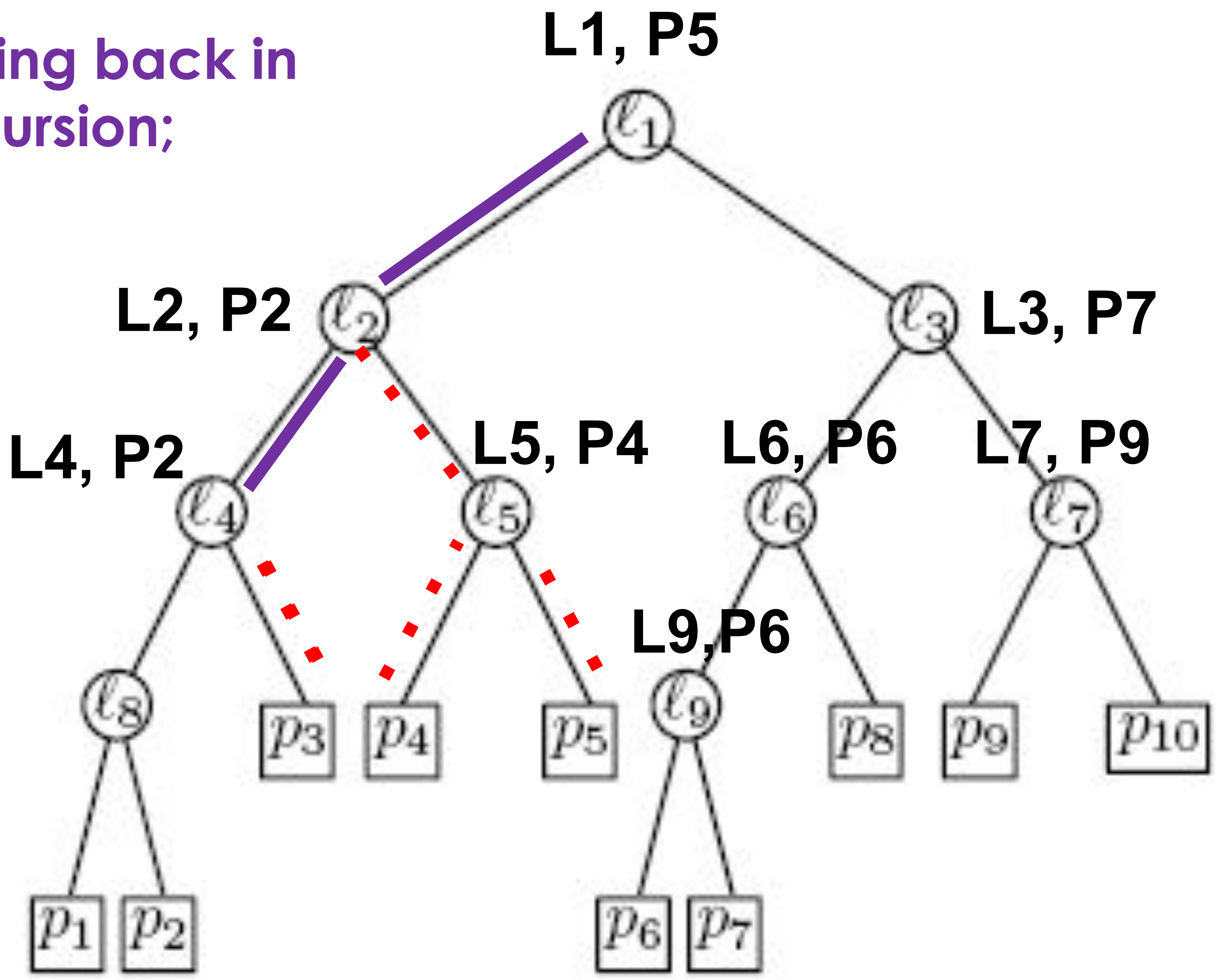
Right child of L4 is a leaf; w’ = dist(q,P3)
w’ > d so no change



KD-tree: 1-NN Query Running

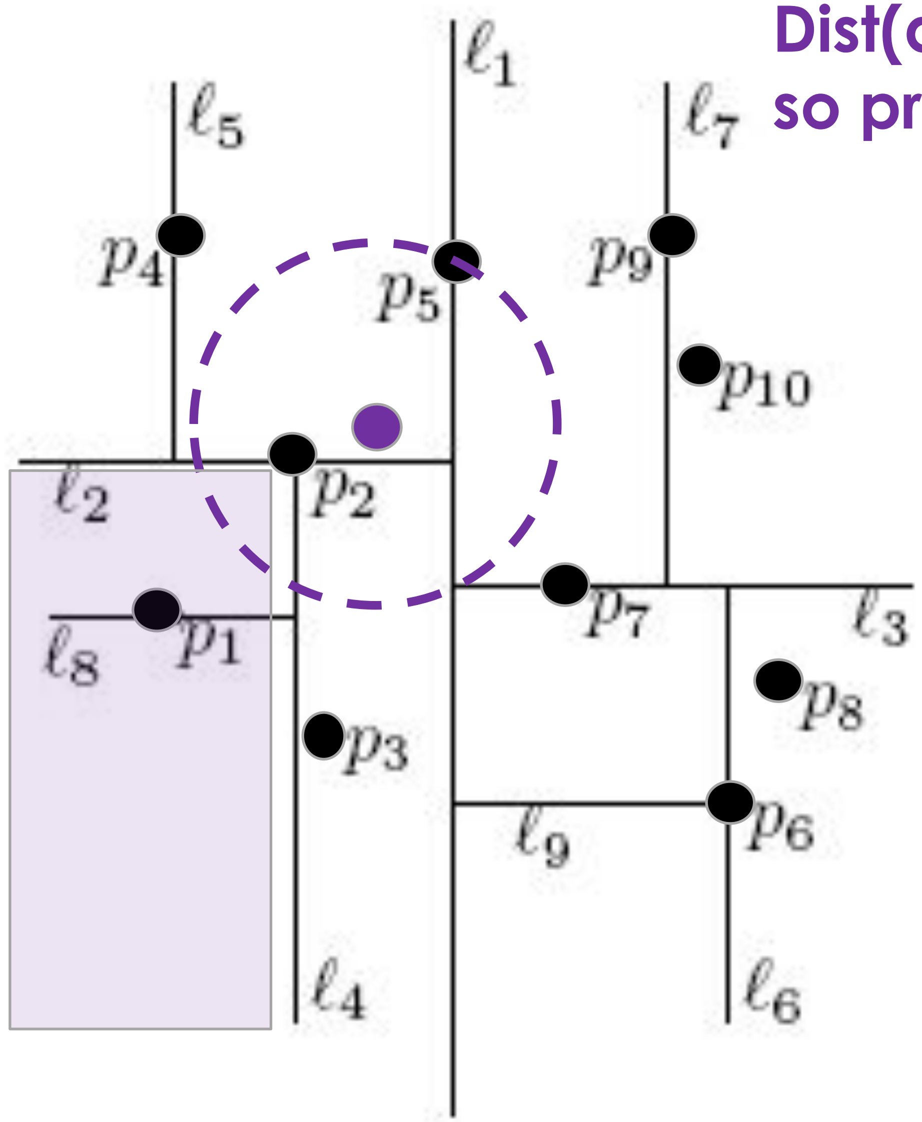


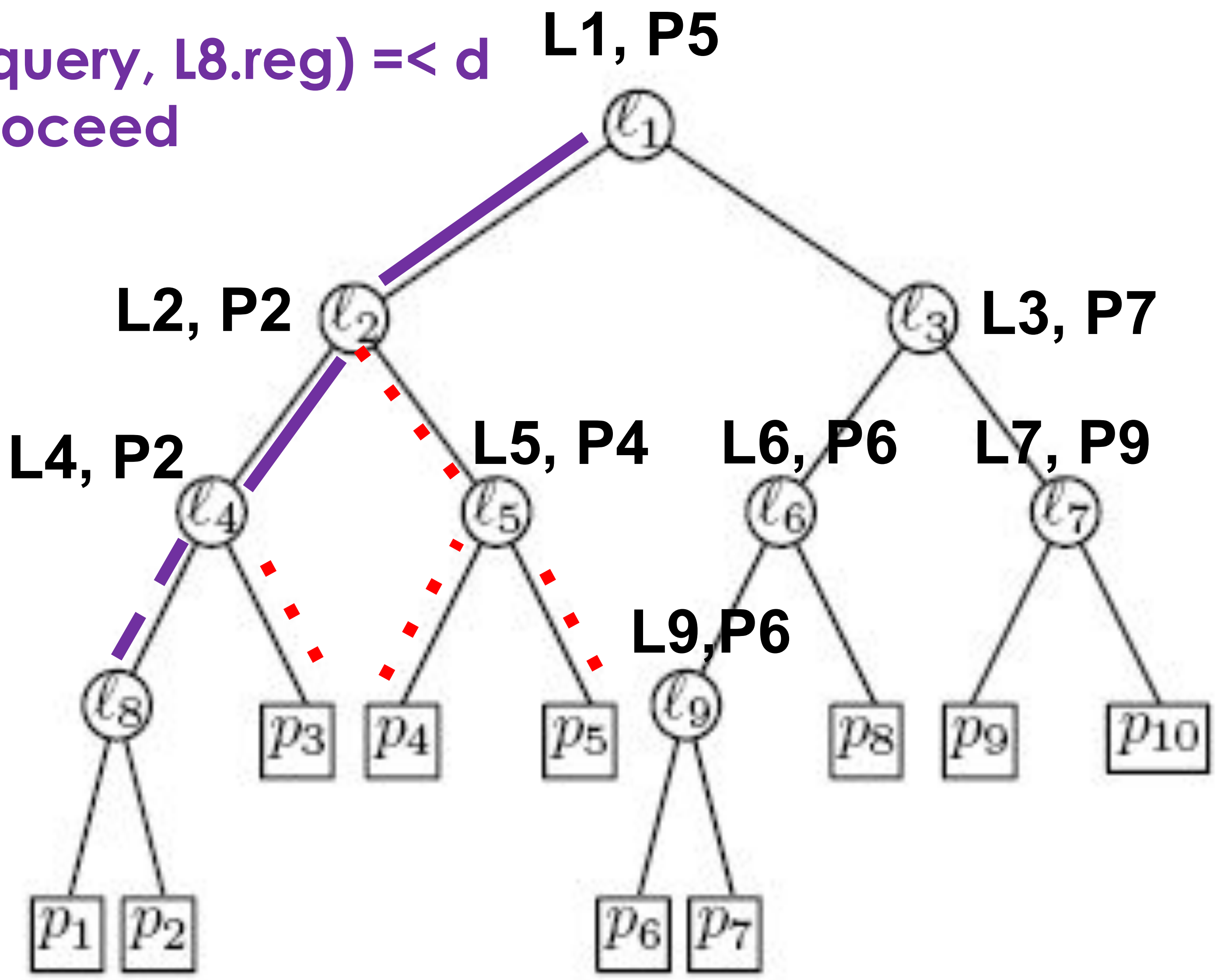
Going back in recursion;



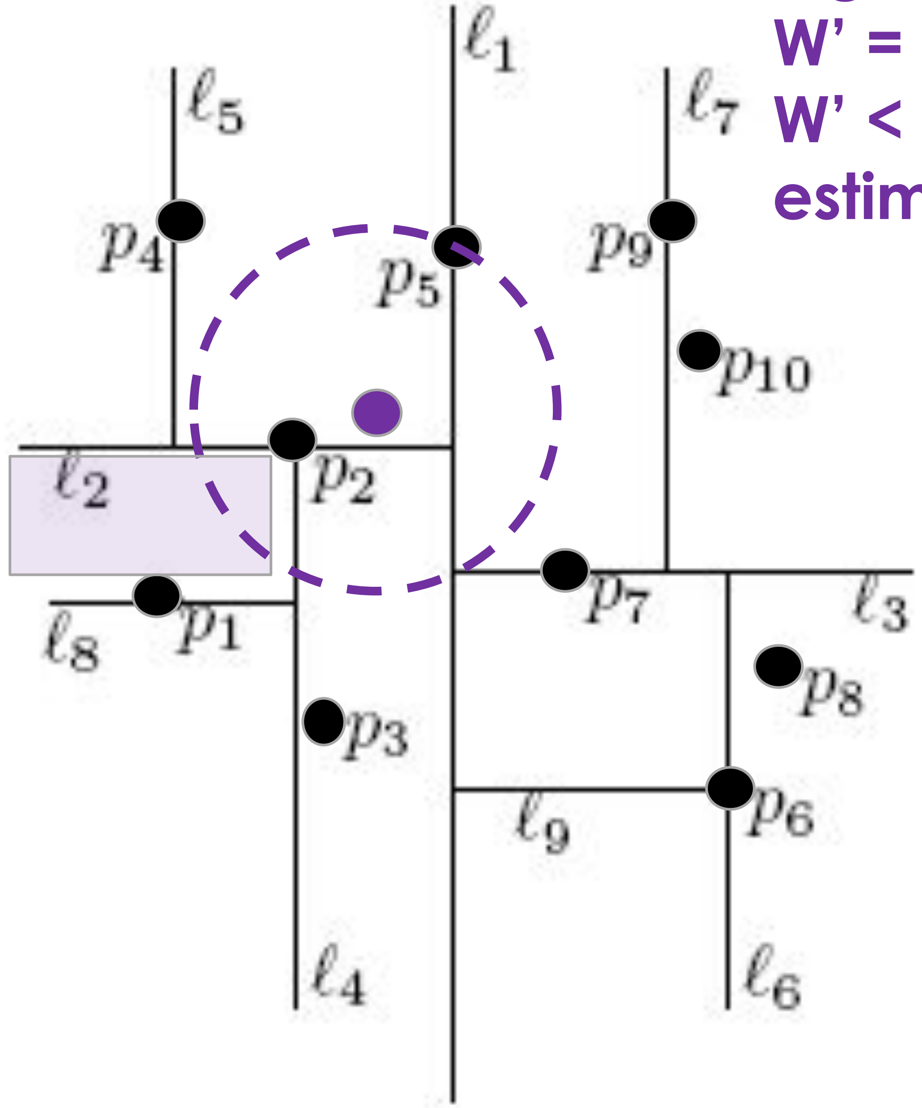
KD-tree: 1-NN Query Running

Dist(query, L8.reg) =< d
so proceed

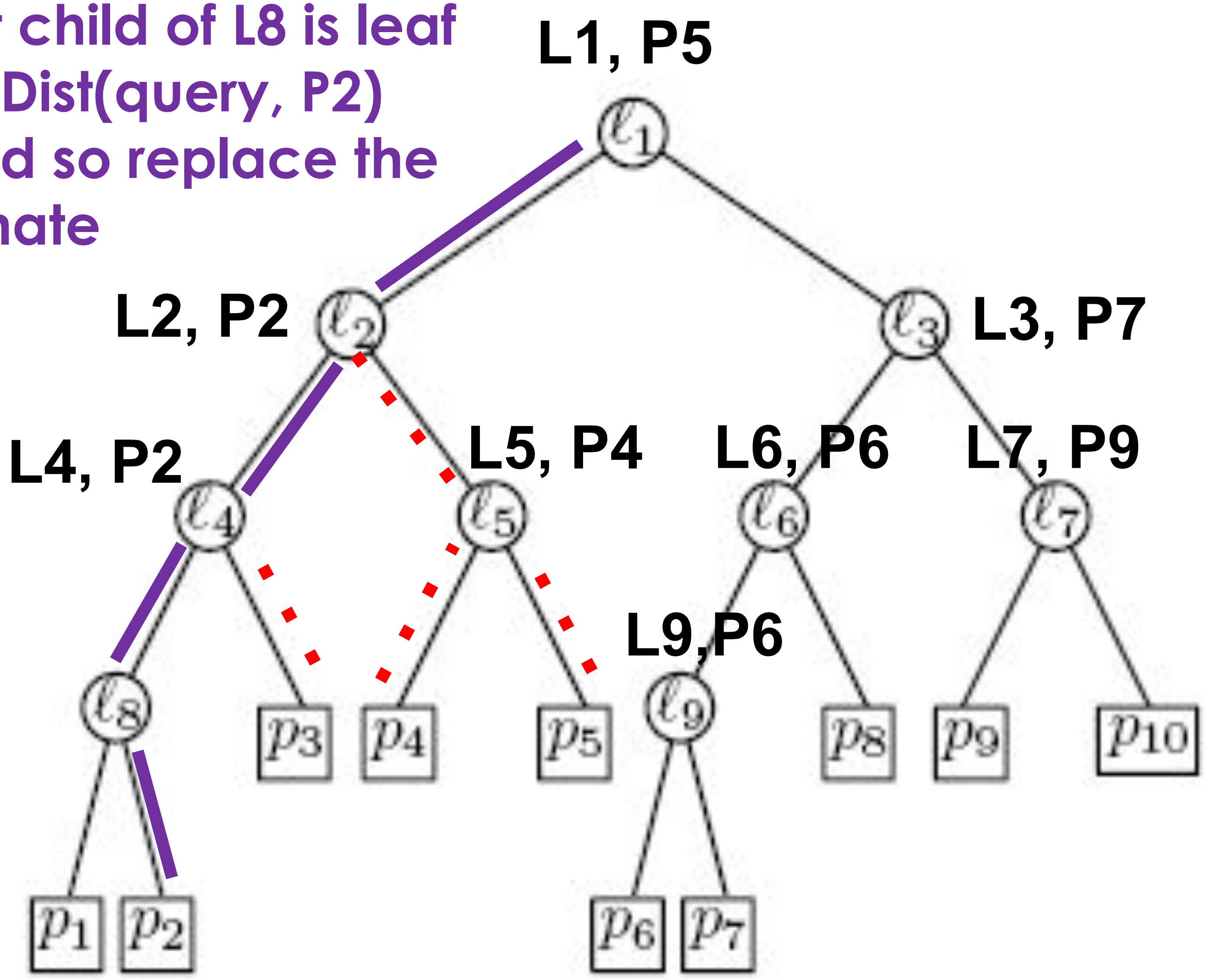




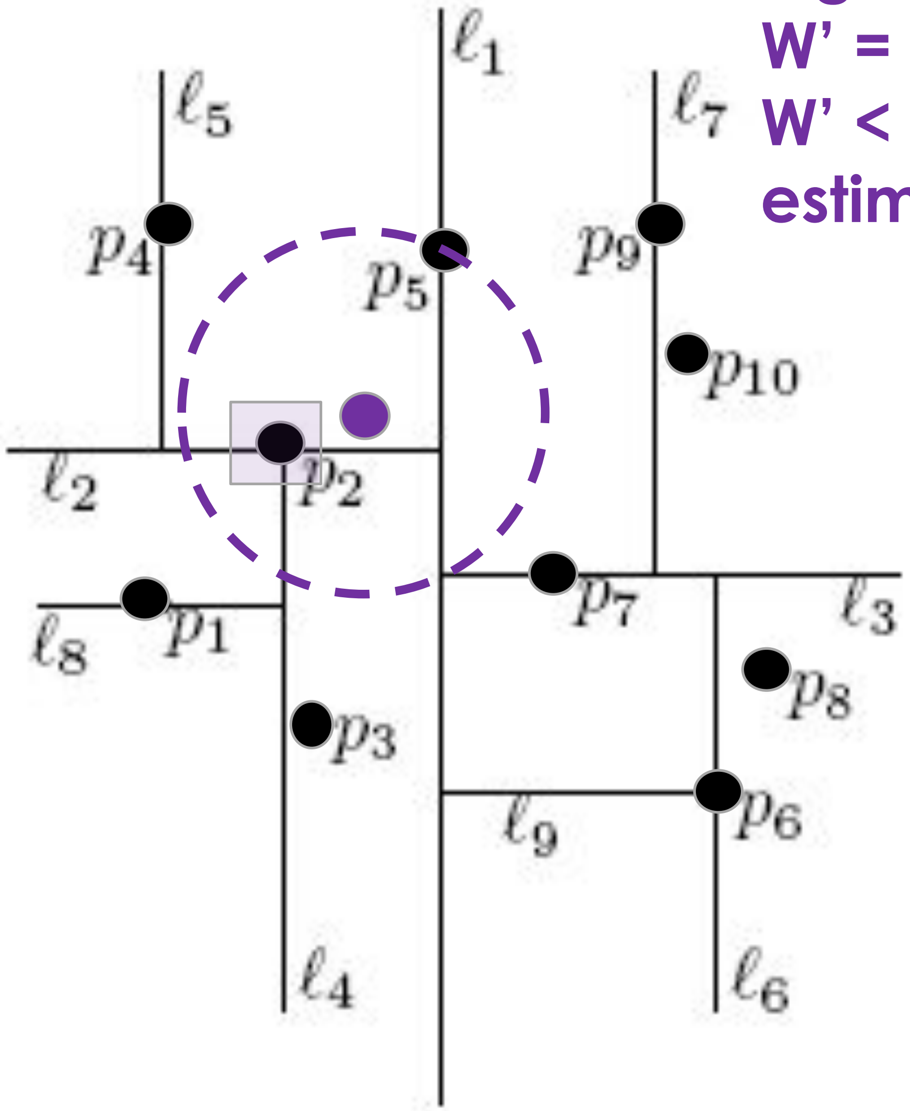
KD-tree: 1-NN Query Running



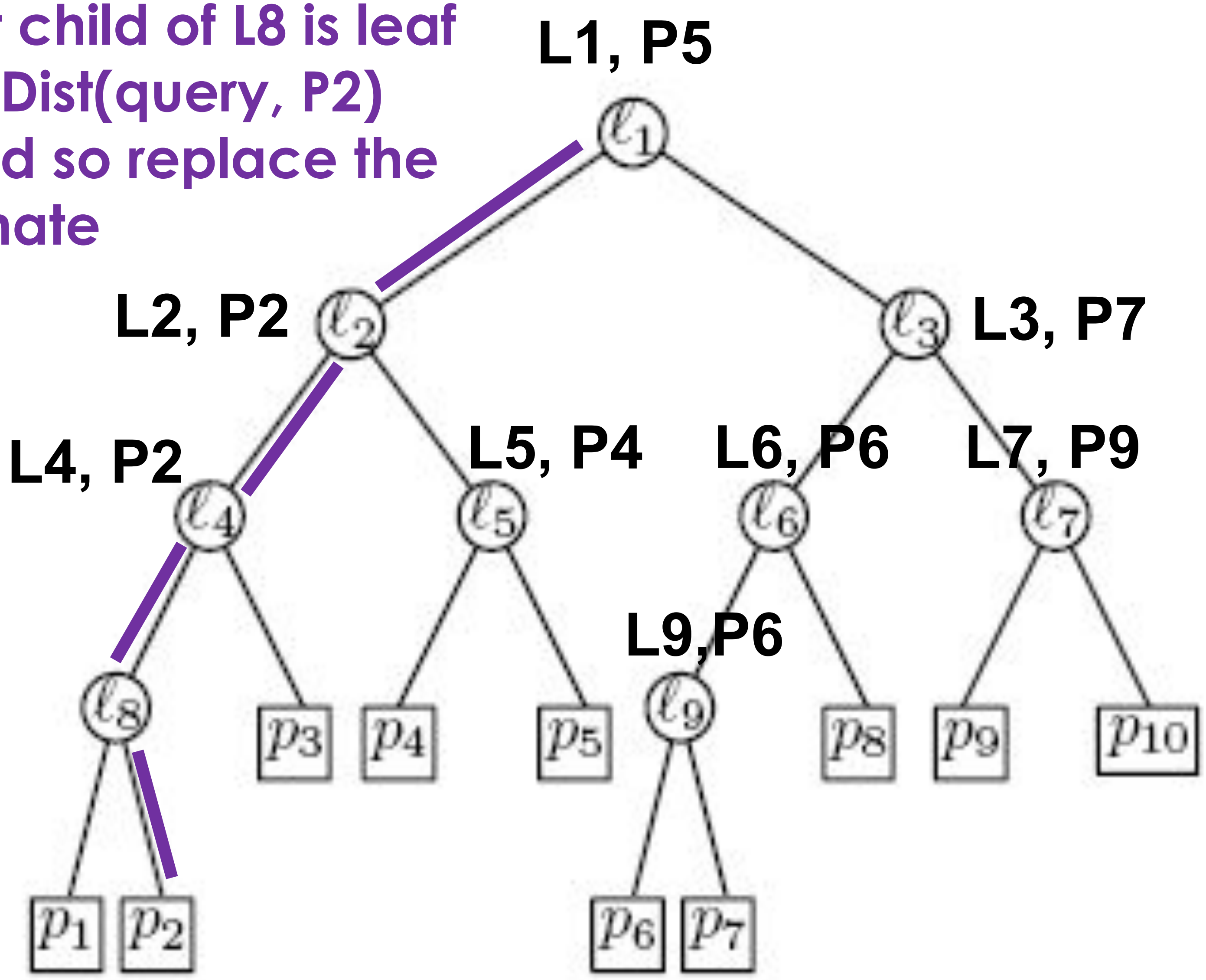
Right child of L8 is leaf
W' = Dist(query, P2)
W' < d so replace the estimate



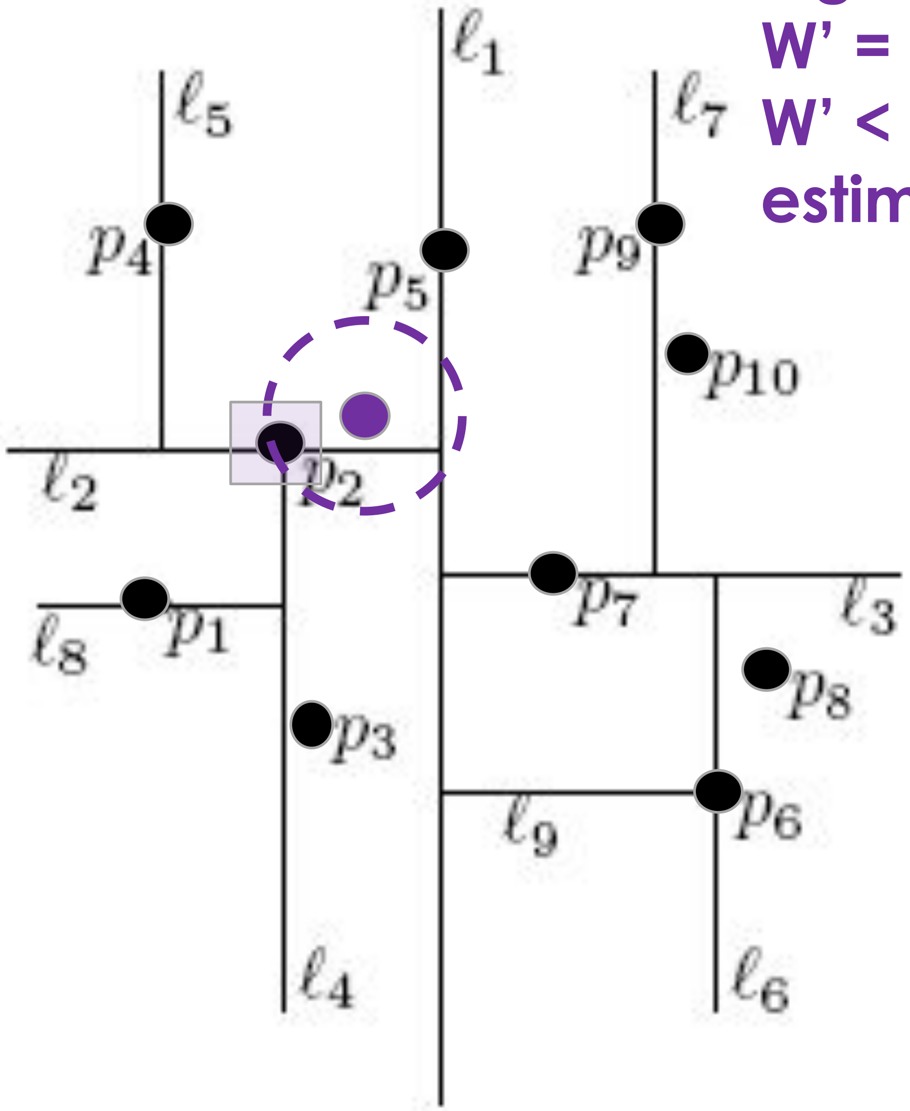
KD-tree: 1-NN Query Running



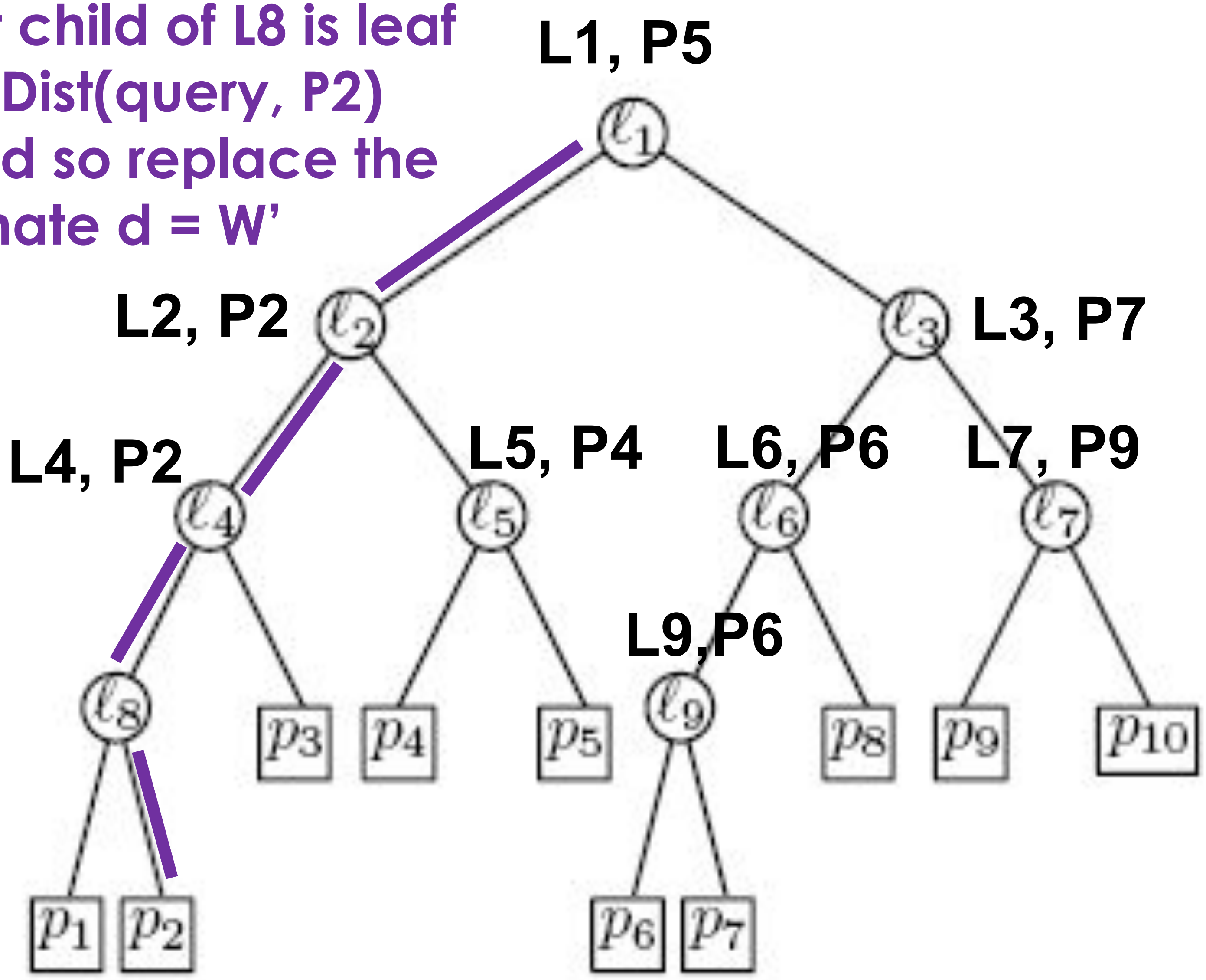
Right child of L8 is leaf
W' = Dist(query, P2)
W' < d so replace the estimate



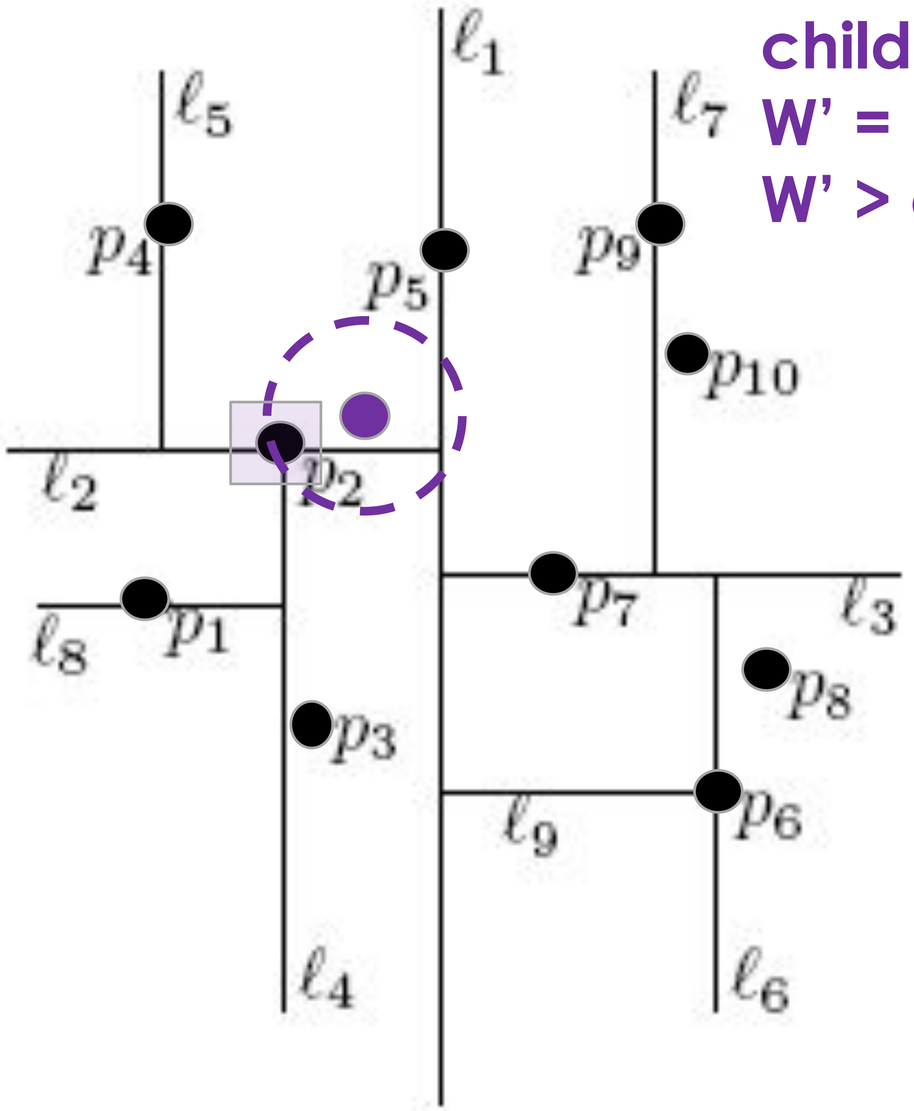
KD-tree: 1-NN Query Running



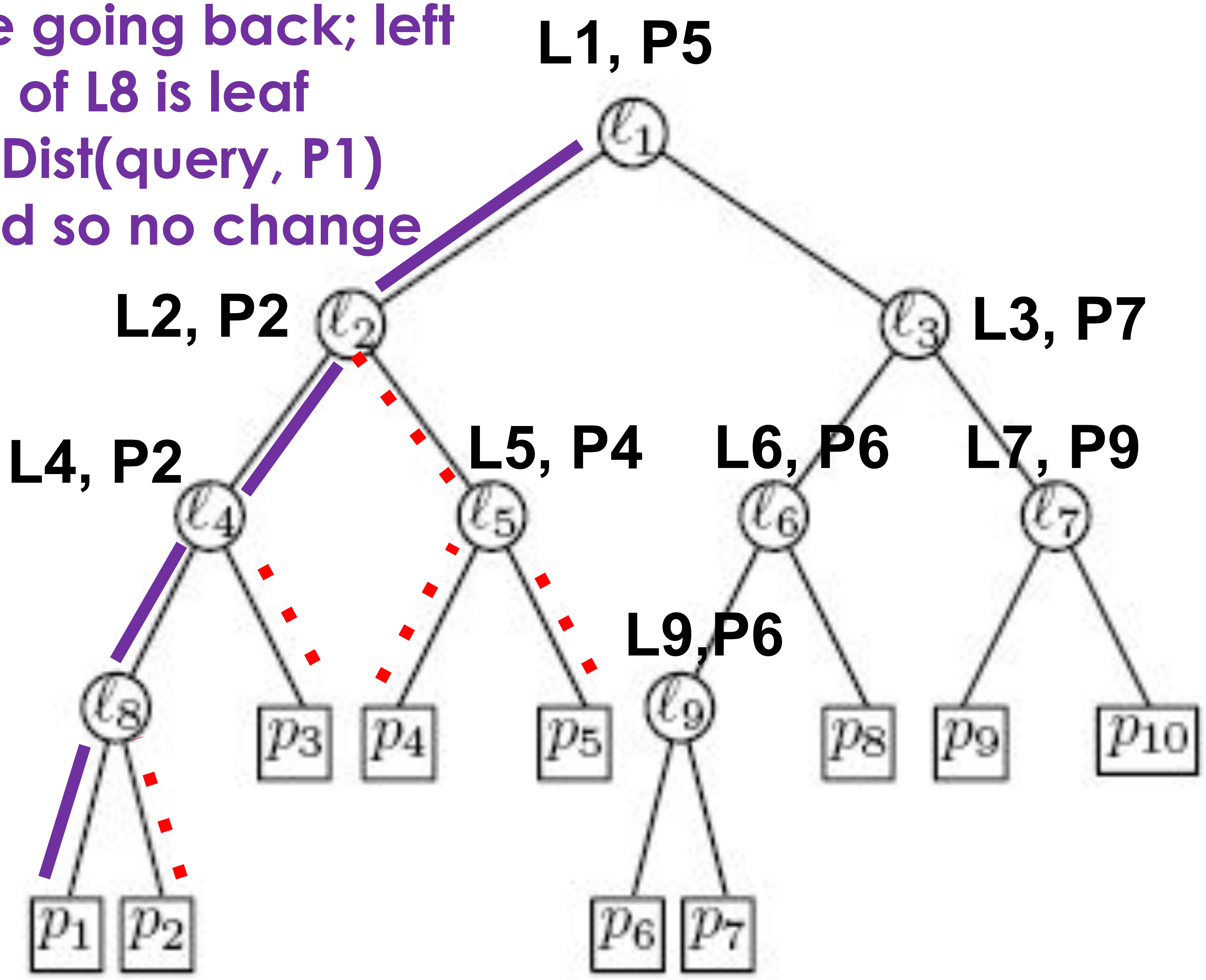
Right child of L8 is leaf
W' = Dist(query, P2)
W' < d so replace the estimate d = W'



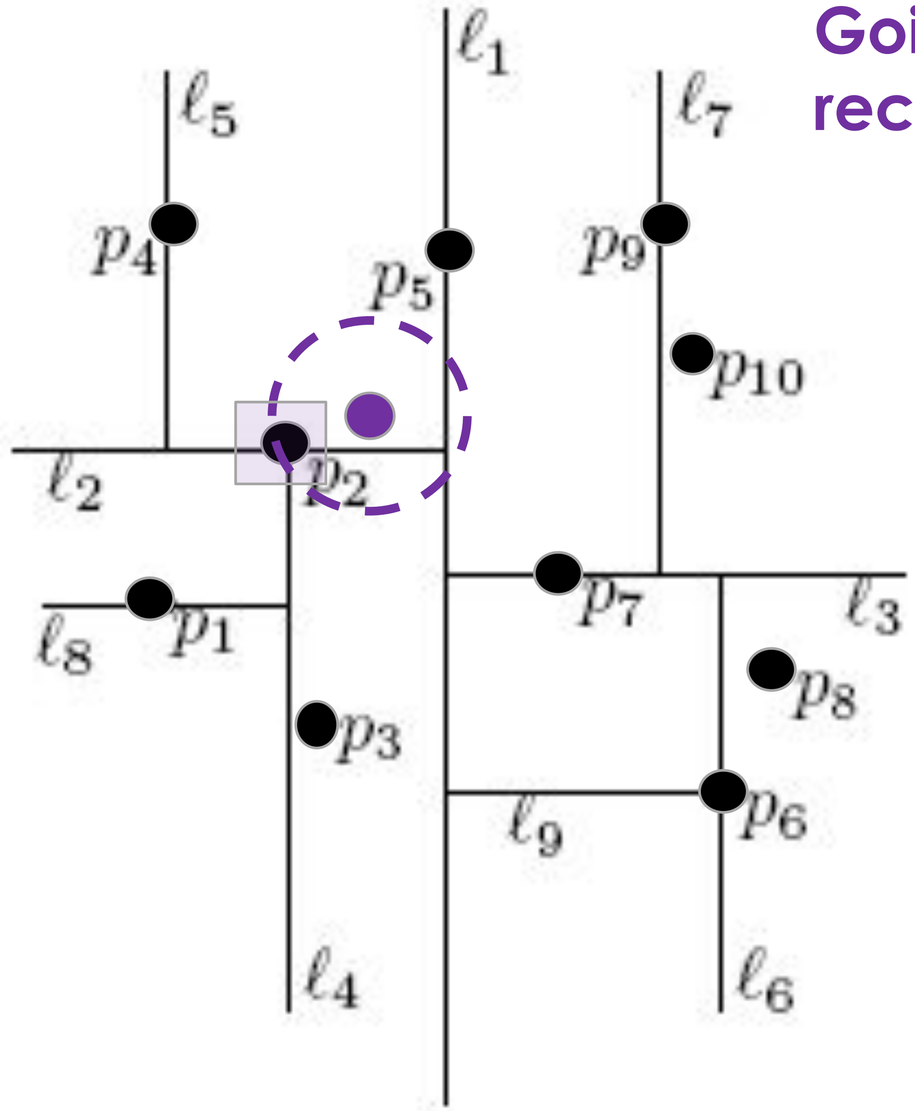
KD-tree: 1-NN Query Running



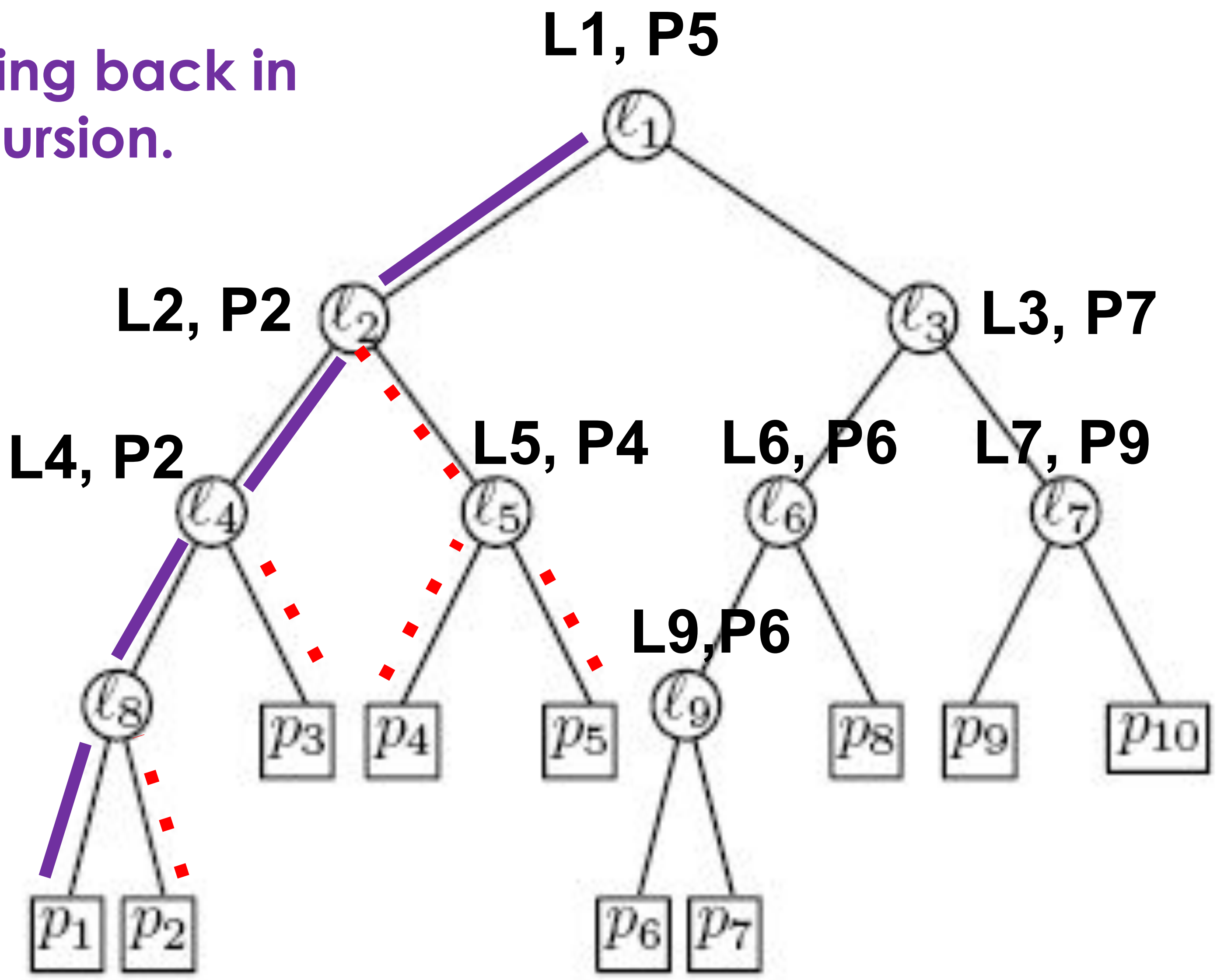
While going back; left child of L8 is leaf
W' = Dist(query, P1)
W' > d so no change



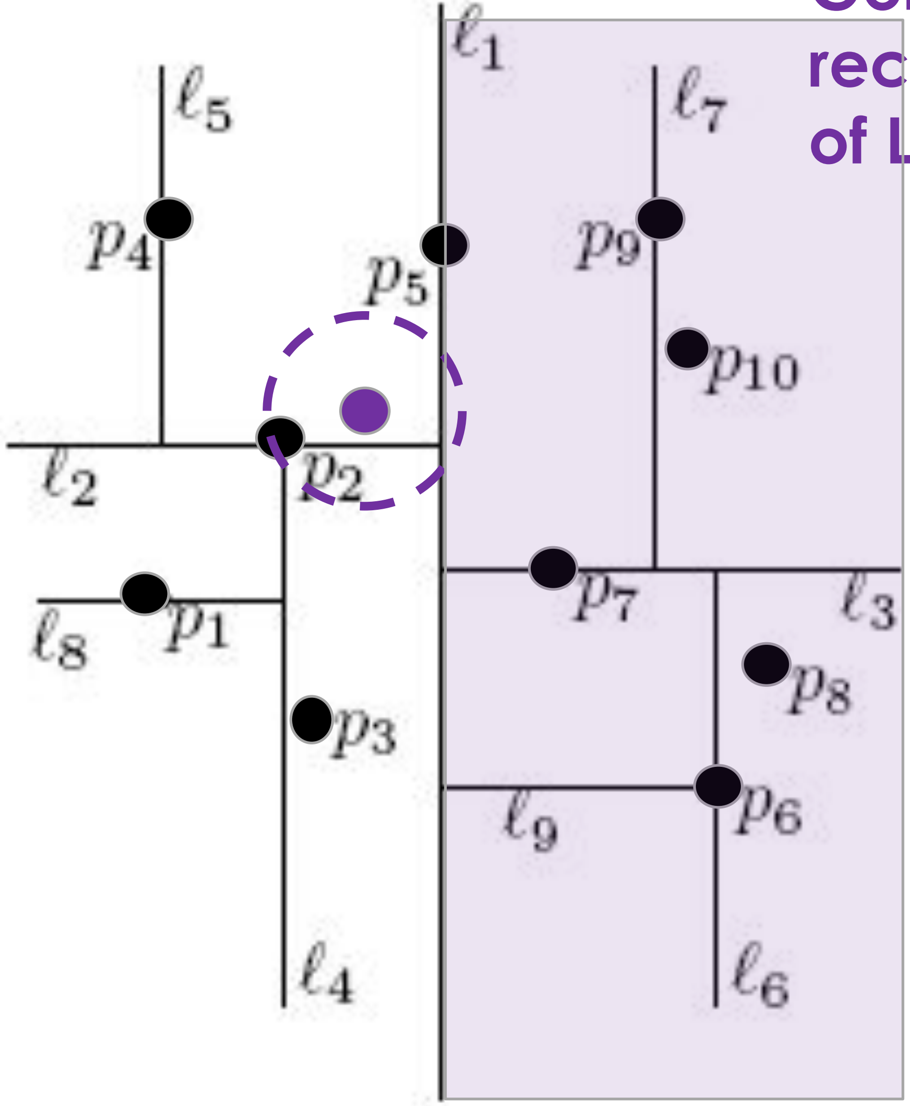
KD-tree: 1-NN Query Running



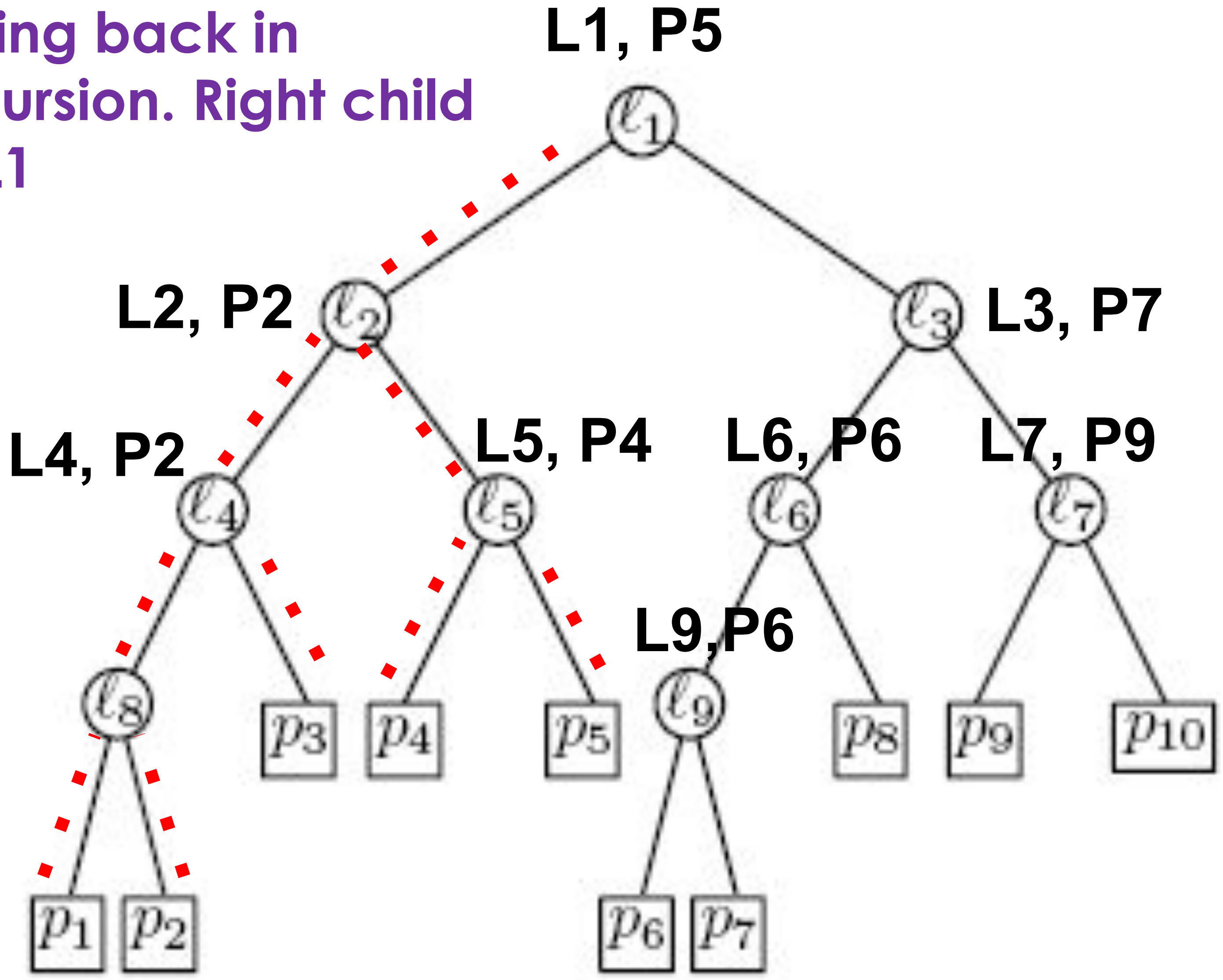
Going back in recursion.



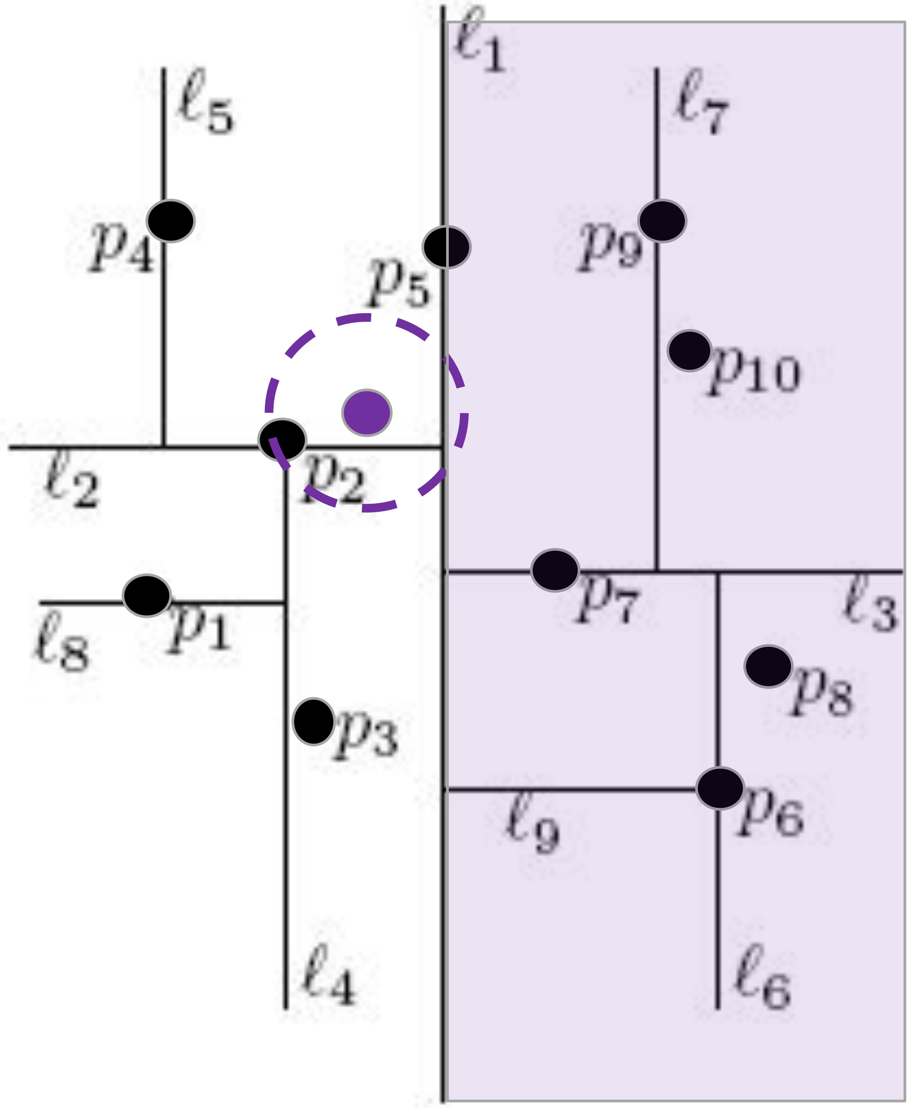
KD-tree: 1-NN Query Running



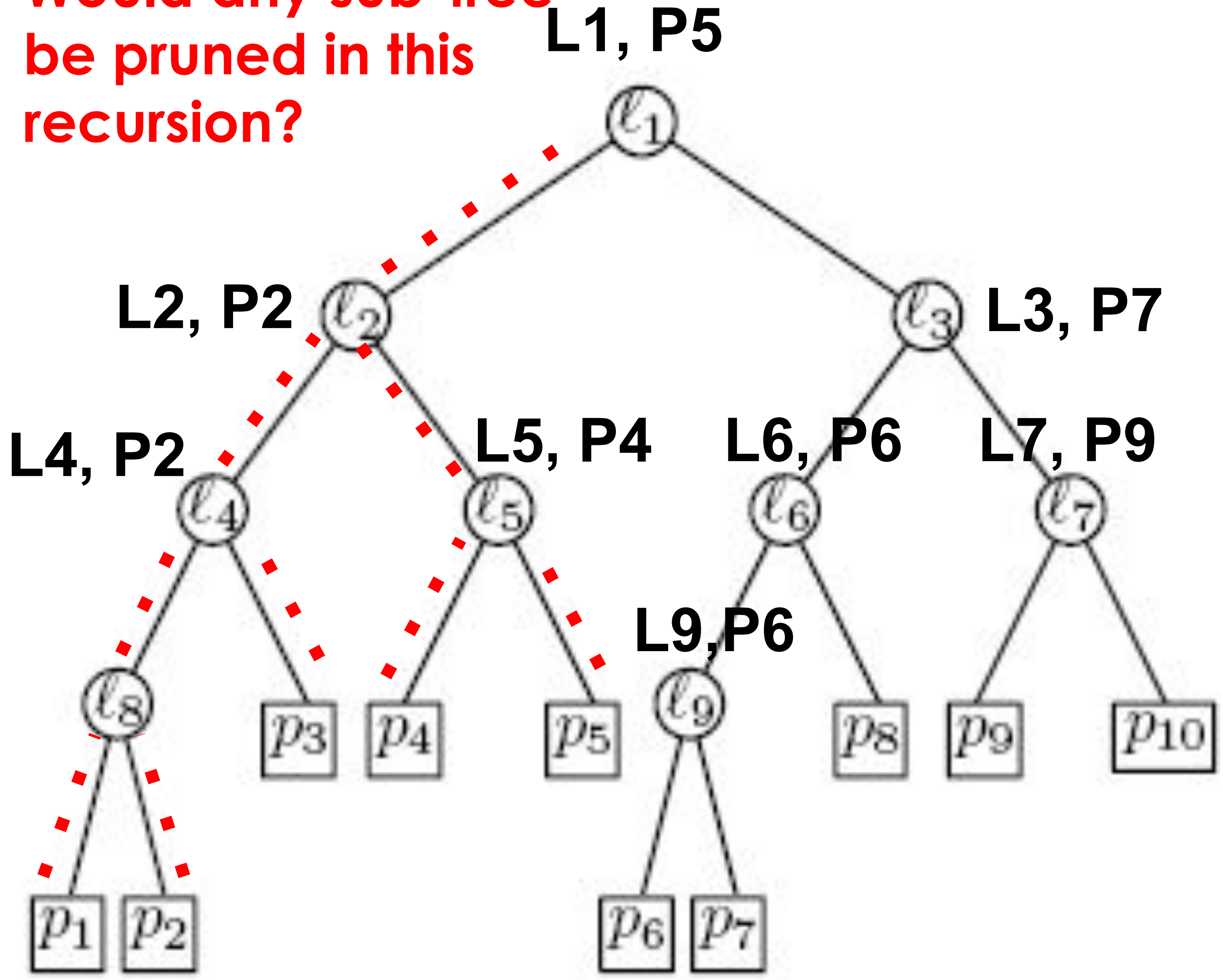
Going back in recursion. Right child of L1



KD-tree: 1-NN Query Running



Would any sub-tree be pruned in this recursion?

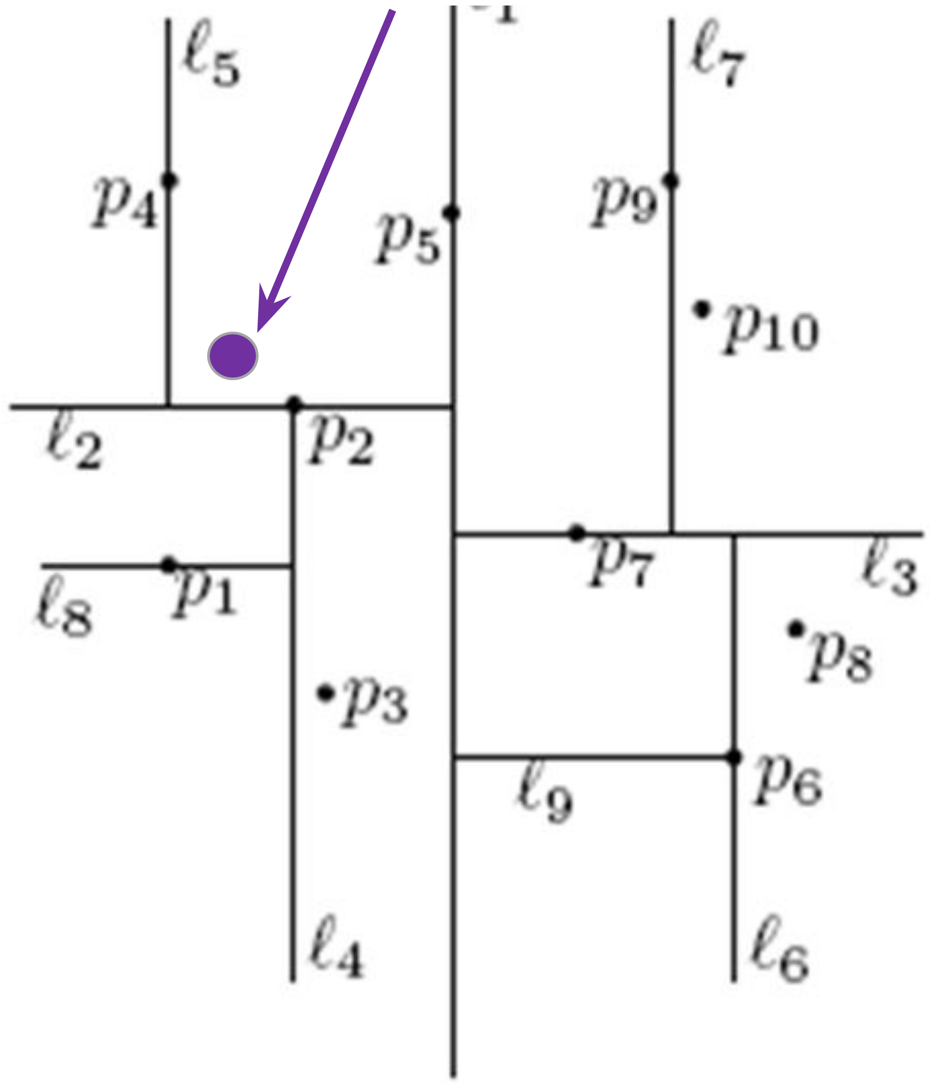


KD Trees – K-Nearest Neighbor Search

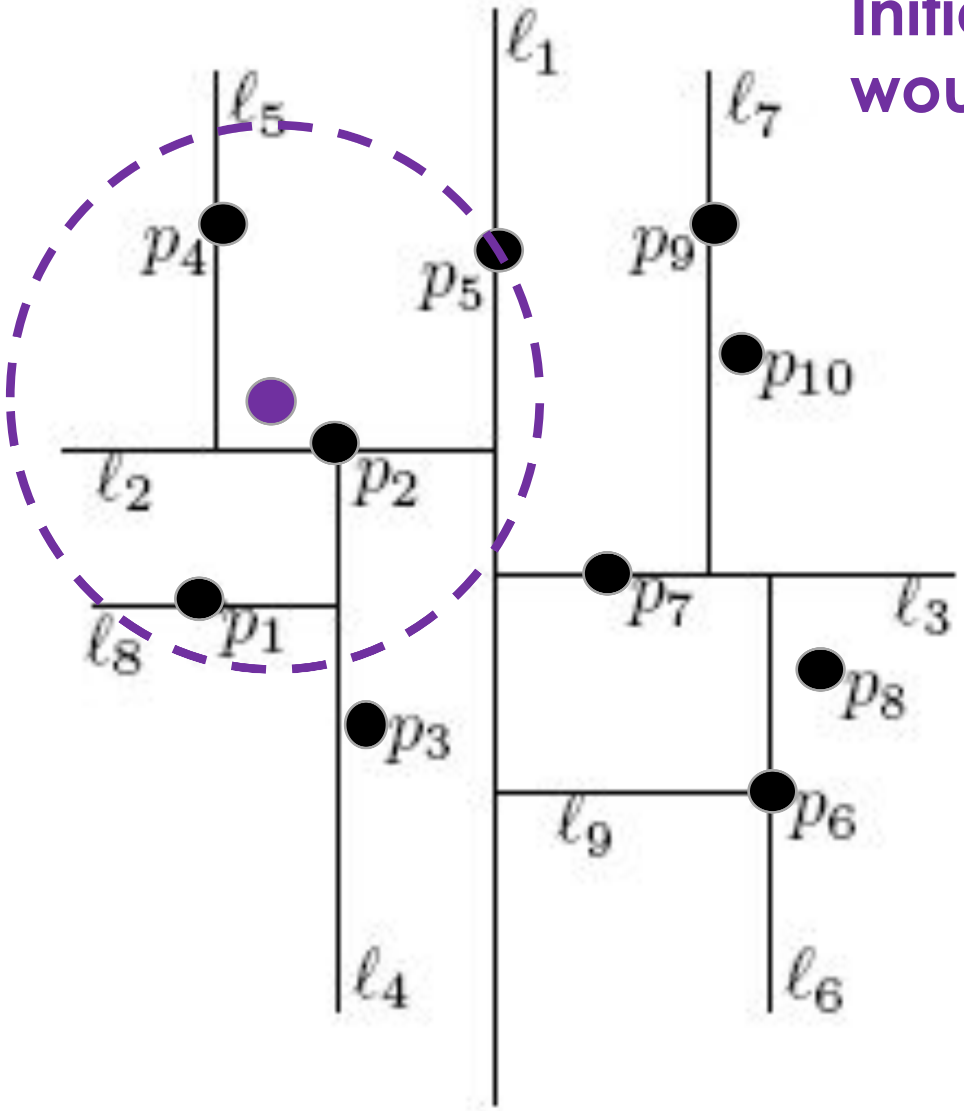
- Similar approach for K-nearest neighbors
- **Three key ideas:**

1) Bounded Priority Queue:
2) "d" thing is defined a distance from q to current farthest point.
3) First we fill up k points in our bounded priority then begin searching the remaining tree while pruning.

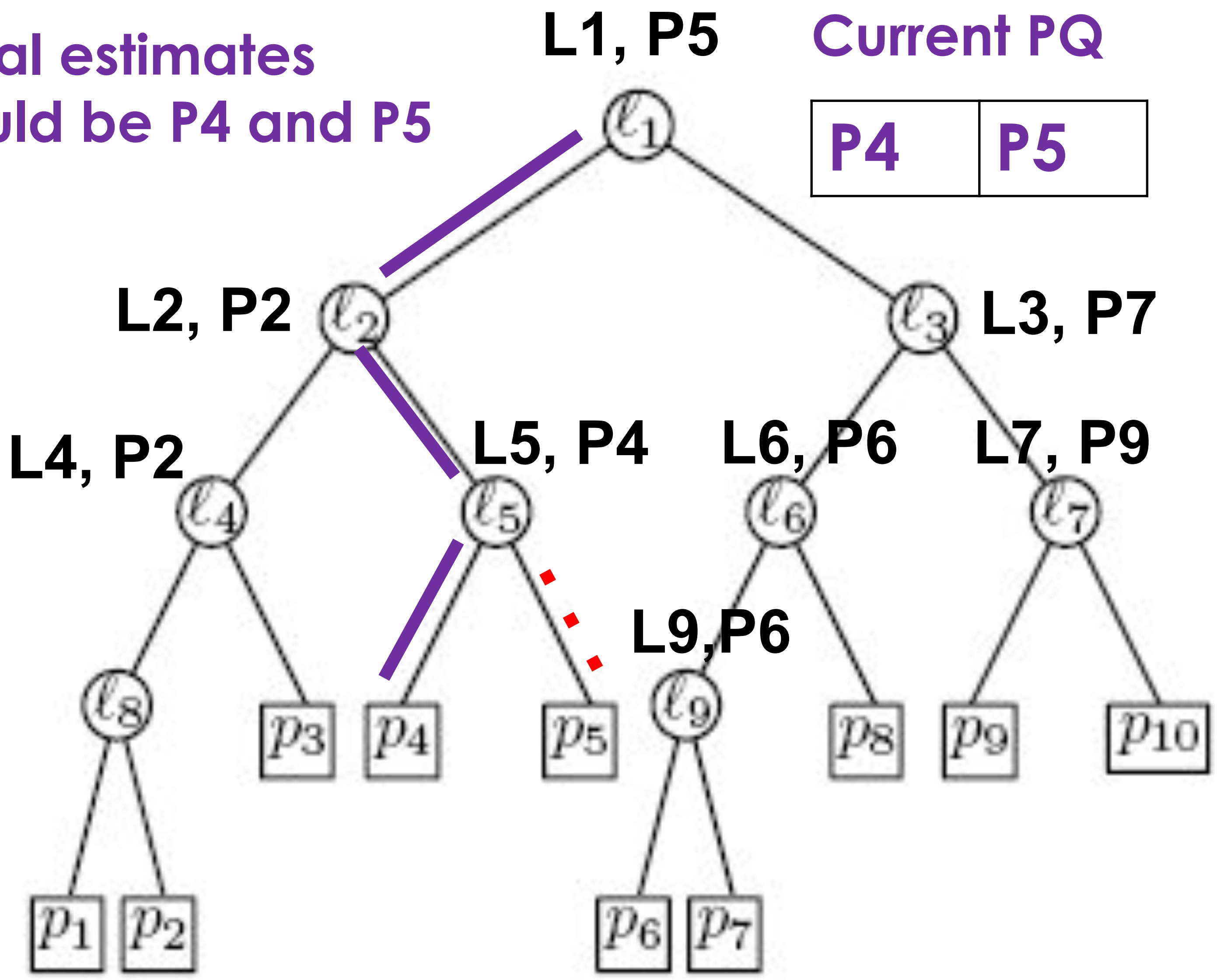
query Point for 2-NN



KD-tree: 2-NN Query Running



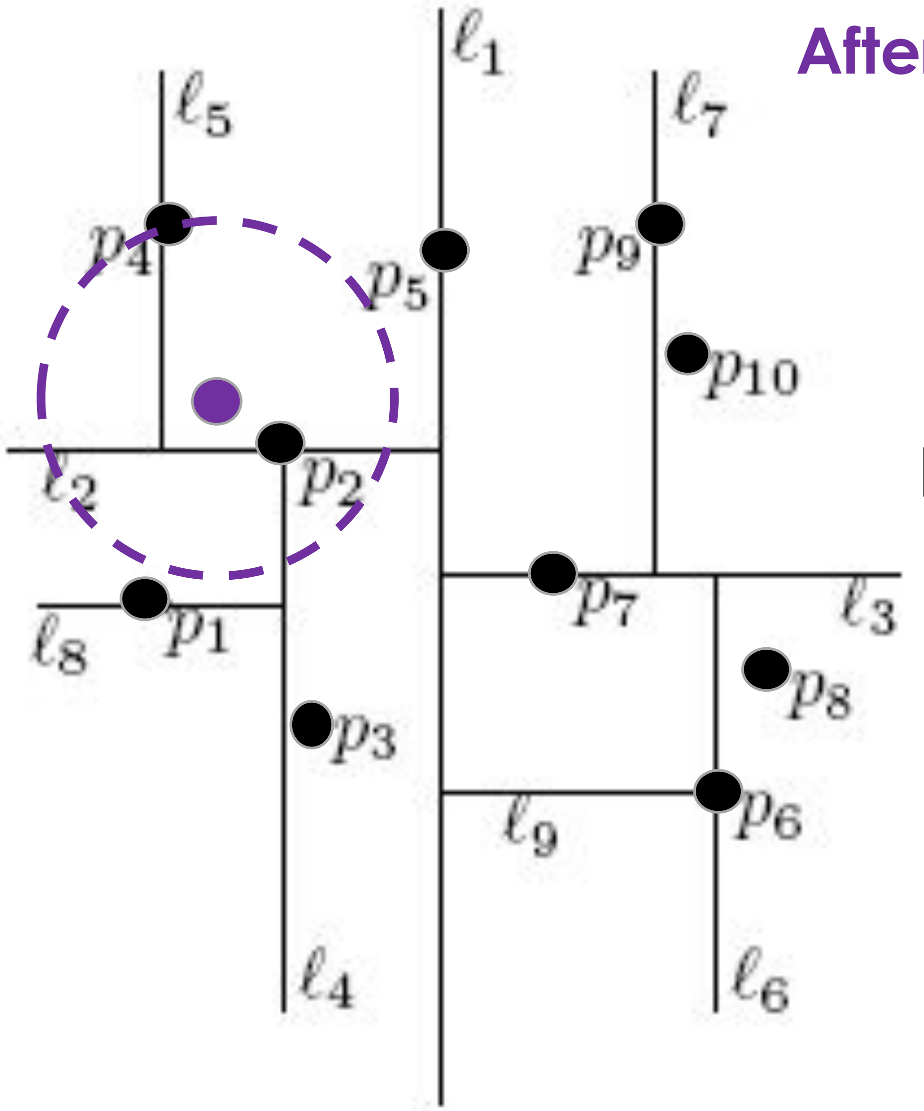
Initial estimates would be P4 and P5



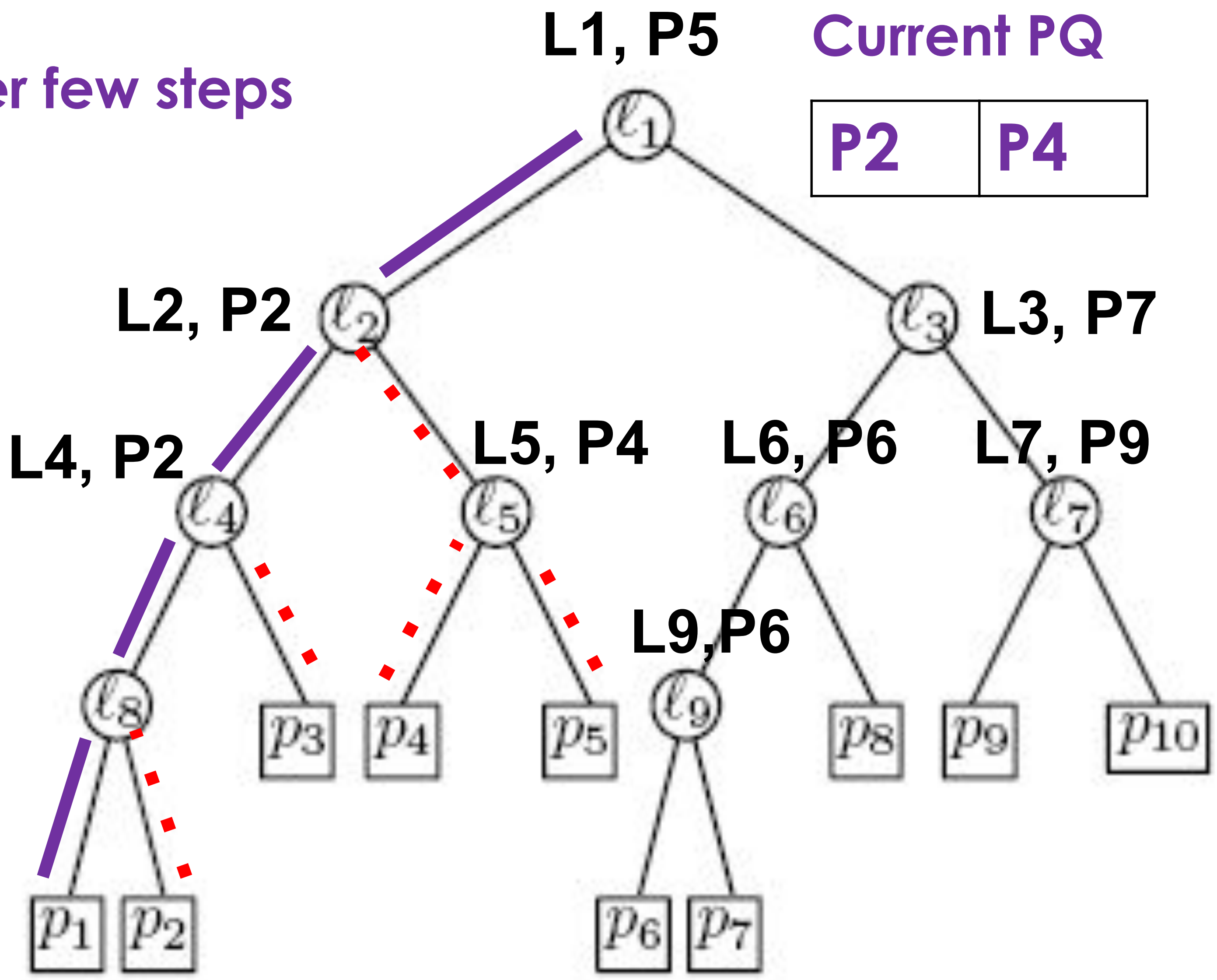
Current PQ

P4	P5

KD-tree: 2-NN Query Running



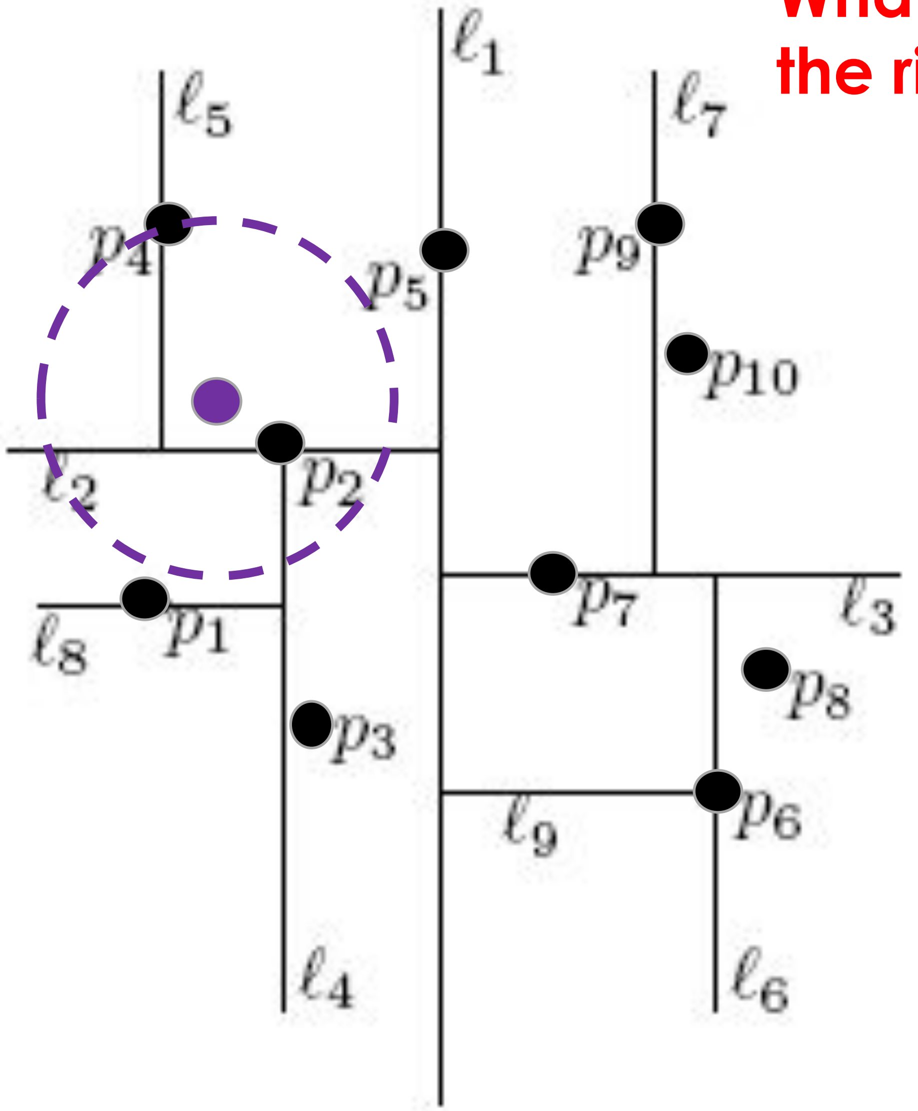
After few steps



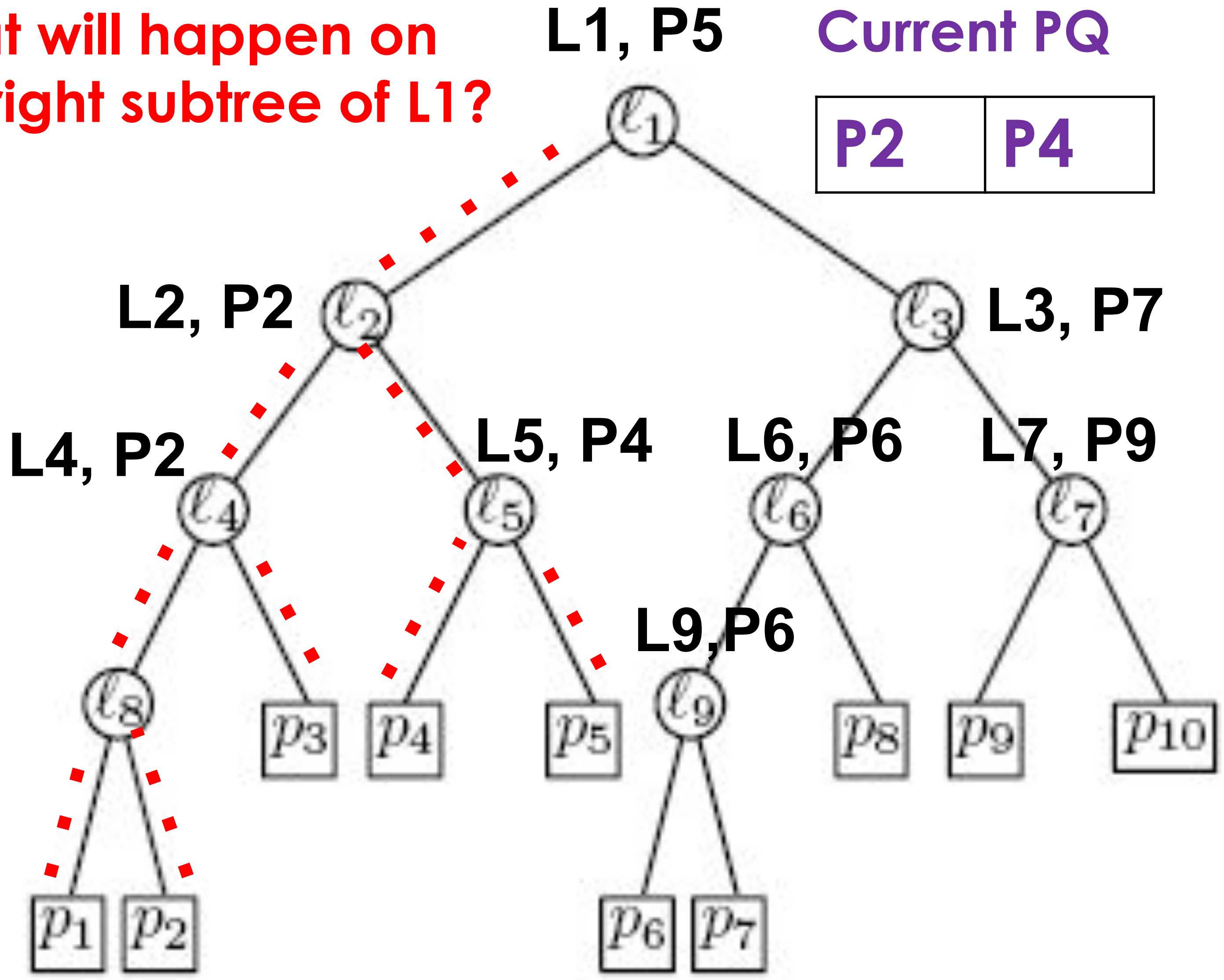
Current PQ

P2	P4

KD-tree: 2-NN Query Running



What will happen on the right subtree of L1?



Current PQ

P2	P4