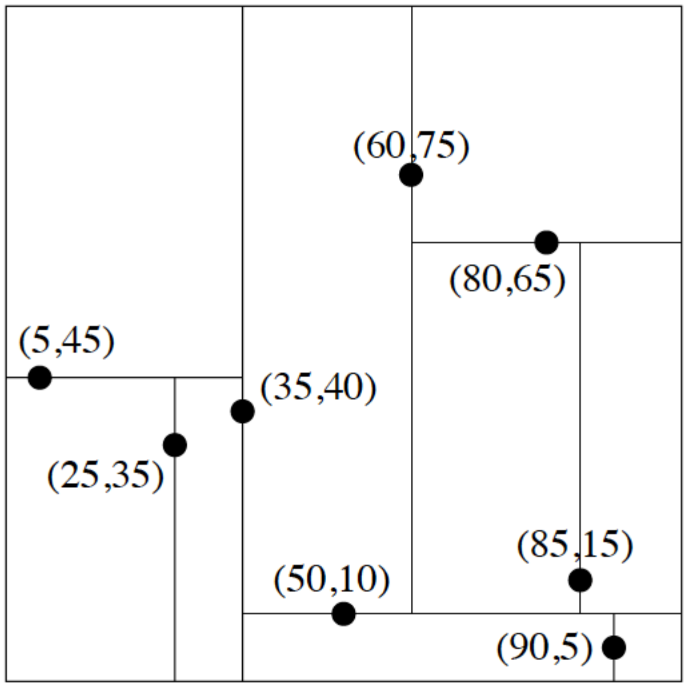
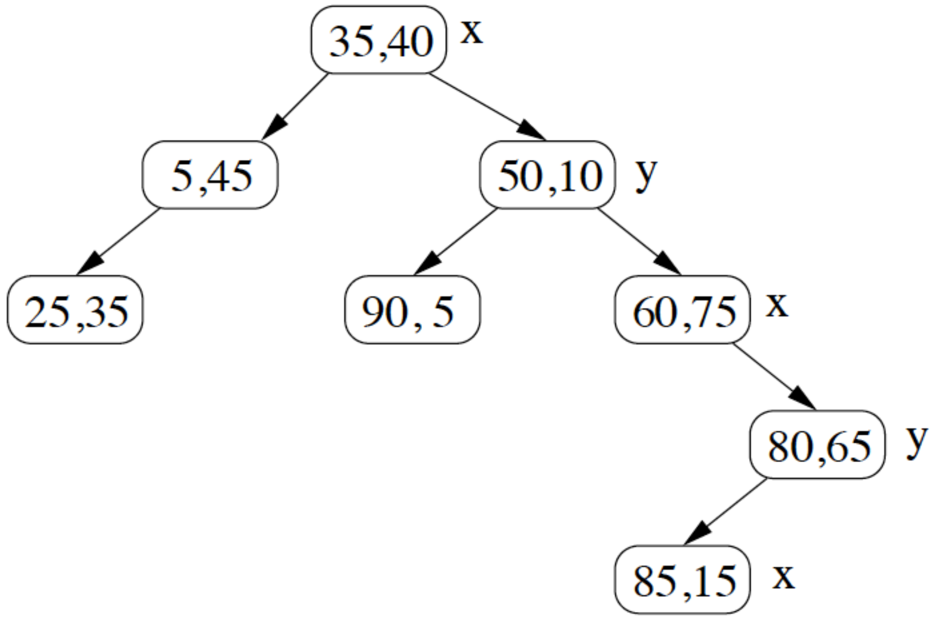
Point / Point-Region Quadtrees

CMSC 420

Reminder: KD-Tree Structure

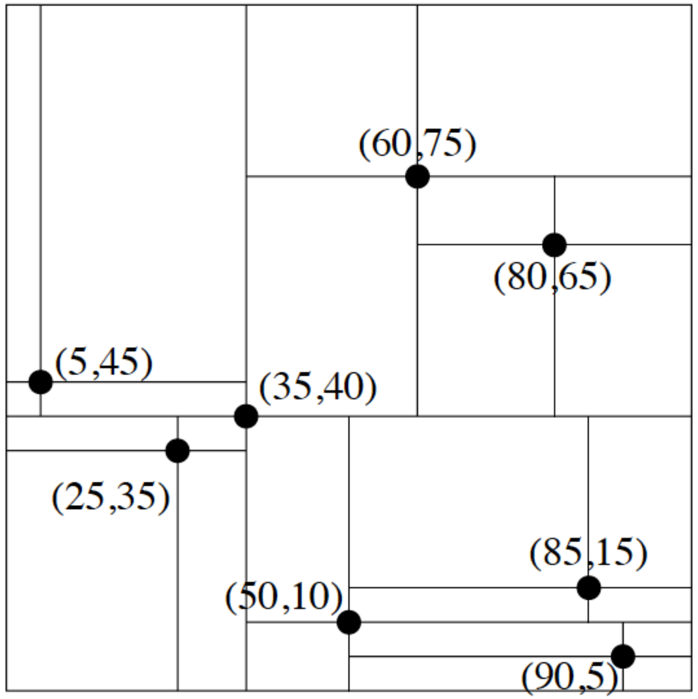
- KD-Trees are binary trees fully defined by the data points.

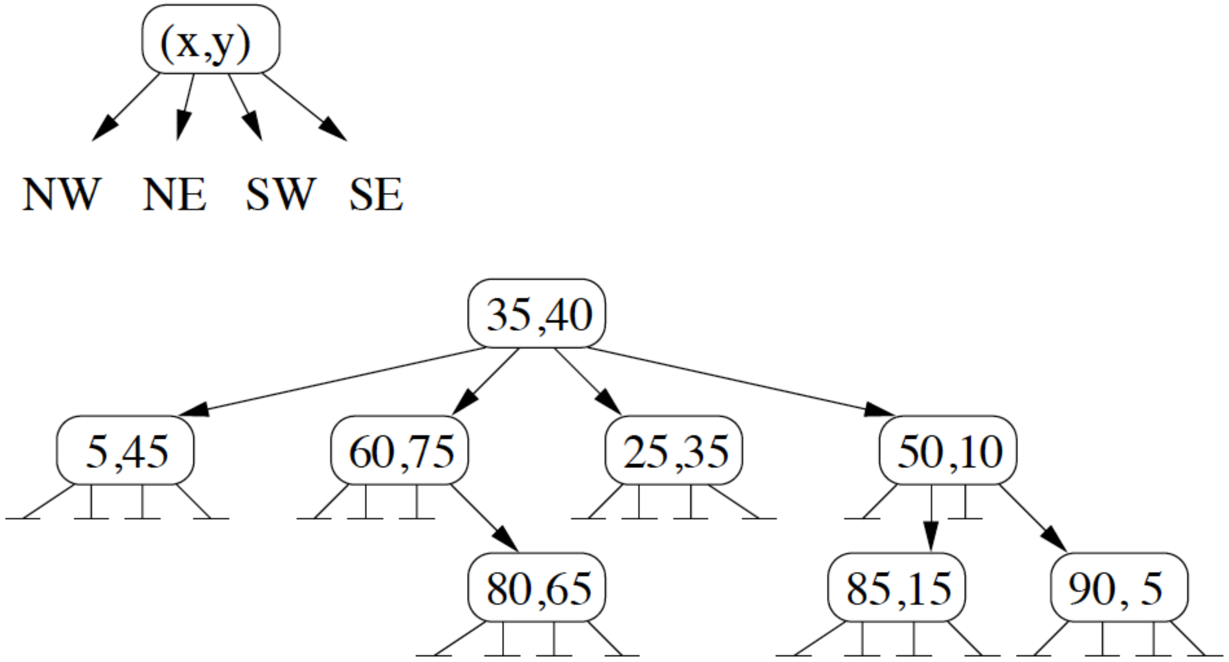




Reminder: KD-Tree Structure

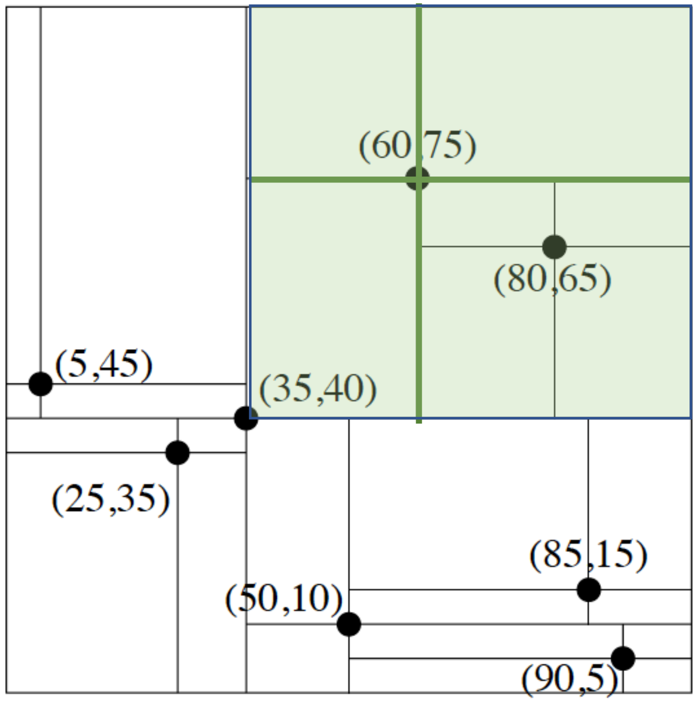
- Point-Quadtrees are a straightforward generalization from binary to 4-ary trees!

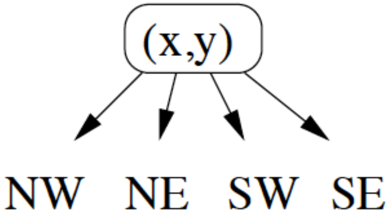




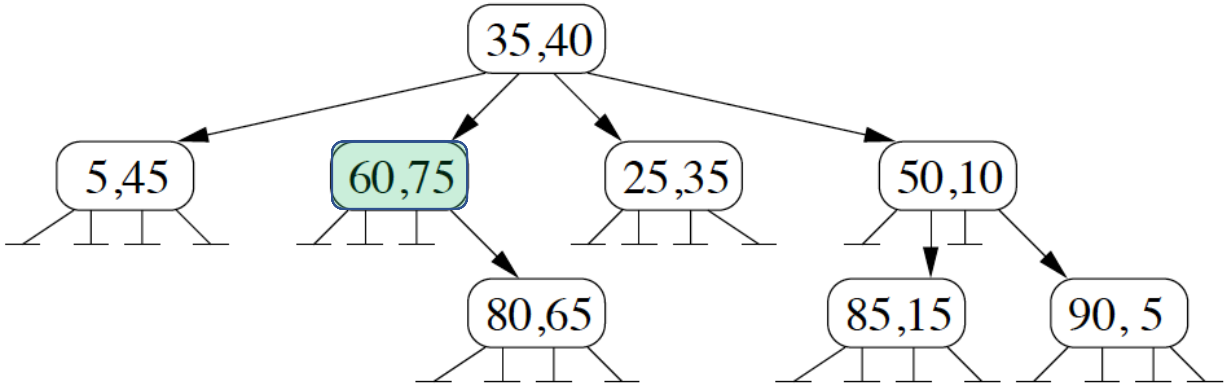
Reminder: KD-Tree Structure

- Point-Quadtrees are a straightforward generalization from binary to 4-ary trees!



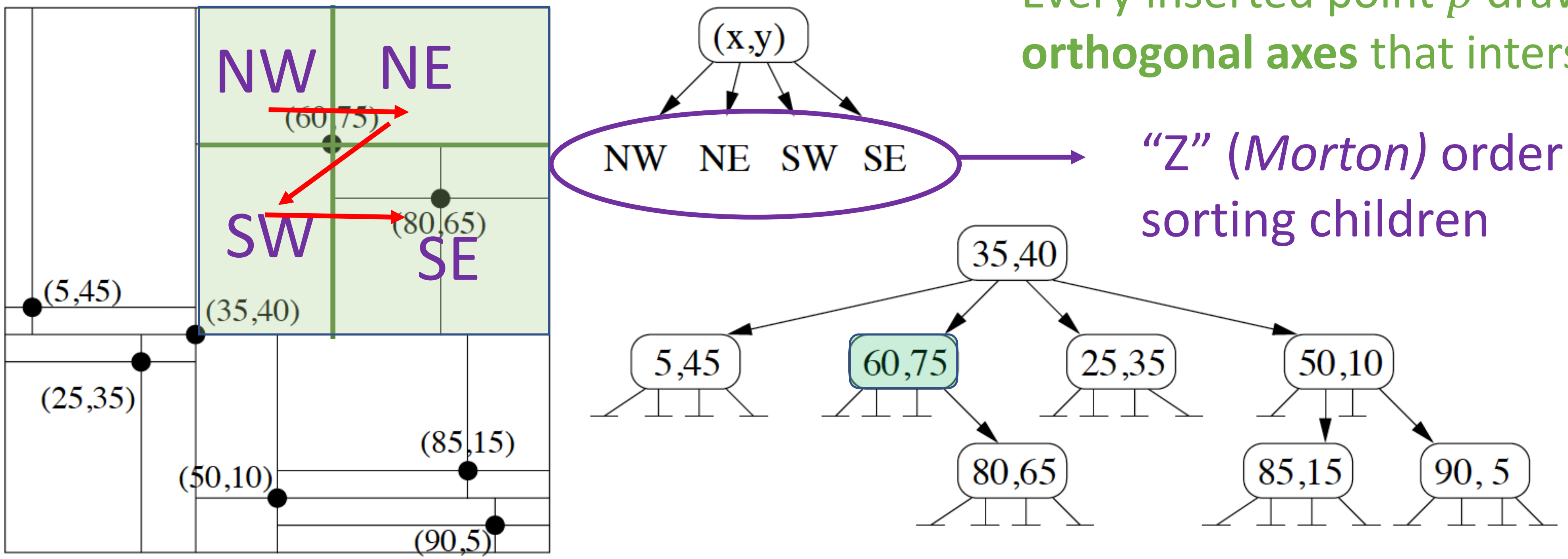


Every inserted point p draws a set of **orthogonal axes** that intersect at p



Reminder: KD-Tree Structure

- Point-Quadtrees are a straightforward generalization from binary to 4-ary trees!



Every inserted point p draws a set of **orthogonal axes** that intersect at p

"Z" (*Morton)* order used for sorting children

Insertion into a Point Quadtree

- Insertion is straightforward.
- If current pointer is `null`, allocate space and return.
- Otherwise, determine quadrant (with 2 comparisons) and recurse there.

Insertion into a Point Quadtree

- Insertion is straightforward.
- If current pointer is `null`, allocate space and return.
- Otherwise, determine quadrant (with 2 comparisons) and recurse there.
- Average height of a point quadtree with n points = …

$\log_2 n$

$\log_4 n$

$n/4$

Something Else

Insertion into a Point Quadtree

- Insertion is straightforward.
- If current pointer is `null`, allocate space and return.
- Otherwise, determine quadrant (with 2 comparisons) and recurse there.
- Average height of a point quadtree with n points = …

$\log_2 n$	$\log_4 n$	$n/4$	Something Else

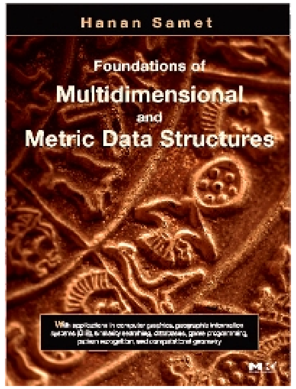
Remember: this means *uniform distribution over inputs…*

Deletion

- Deletion from a Point QuadTree is… *terrible.* ☹
- Find the point to be deleted, delete it
- And re-insert all the points rooted in the relevant subtree ☹

Deletion

- Deletion from a Point QuadTree is… *terrible.* ☹
- Find the point to be deleted, delete it
- And re-insert all the points rooted in the relevant subtree ☹
- Hanan's book has a better, but much more complicated algorithm for Point QuadTree deletion
  - If interested, ask Jason to post pseudocode on Piazza



Range, m-NN, etc…

- Algorithms for range and m-nearest neighbors are a straightforward generalization of those in KD-Trees.
- You can still prune away subtrees and do branch-and-bound and all that cool stuff.

Main issue with point quadtrees

- Fanout increases **exponentially** with dimension d.
- For a d −dimesional space, the Point Quadtree’s node has 2^d children.
- In practice, point quadtrees are used only up to 3 dimensions *(oct-trees).*
- KD-Trees are preferred because of their tractable fan-out and deletion efficiency.

Data-based vs space-based representations

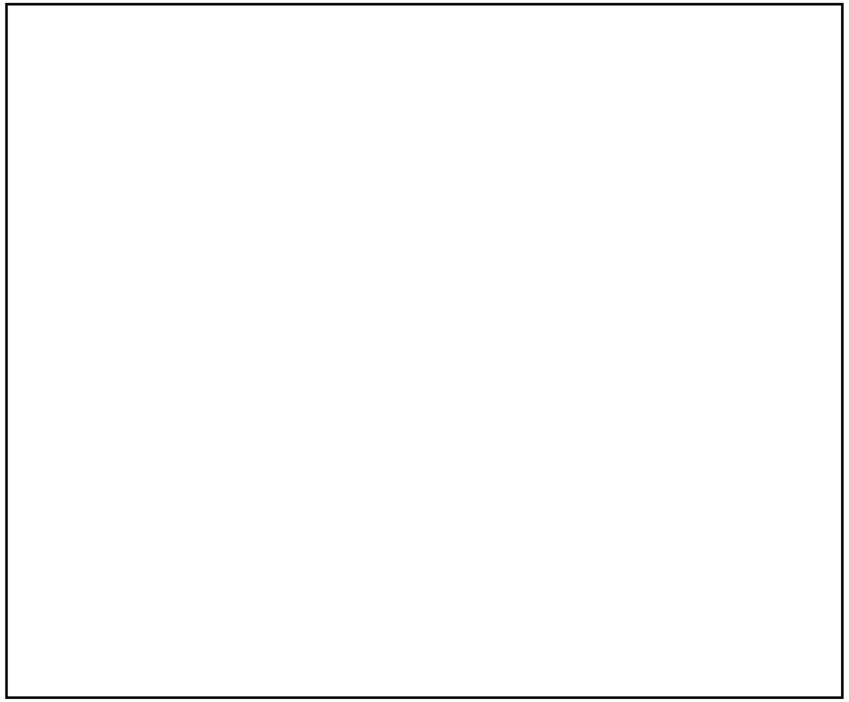
- Point QuadTrees and KD-Trees have the common characteristic that they **split based on data points.**
- This is not the only way: we can also build spatial data structures based on spatial decompositions!
- Examples: Region Quadtrees, Point-Region (P-R) QuadTrees
  - We will talk about PR Quadtrees this semester and post some old slides on Region QuadTrees for your enlightenment

Main idea

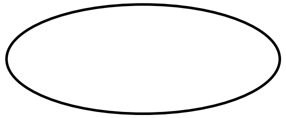
- We will assume a large enough "bounding square" of dimensions $2^{k-1} \times 2^{k-1}$ for some $k \in \mathbb{Z}$.
- Our points are ***agoraphobic***: they will need their own private space.
  - To that end, **every time we insert a point, we will subdivide the half-planes by 4** (or 2, if you're only looking at the edge) **until all the points are separated!**
- This will lead to a tree of fanout 4 with 3 kinds of nodes:
  - White nodes (no point contained)
  - **Black** nodes (point contained)
  - Gray *(mixed)* nodes (with at least one non- White **child node).**

Example

Space



PR-QuadTree

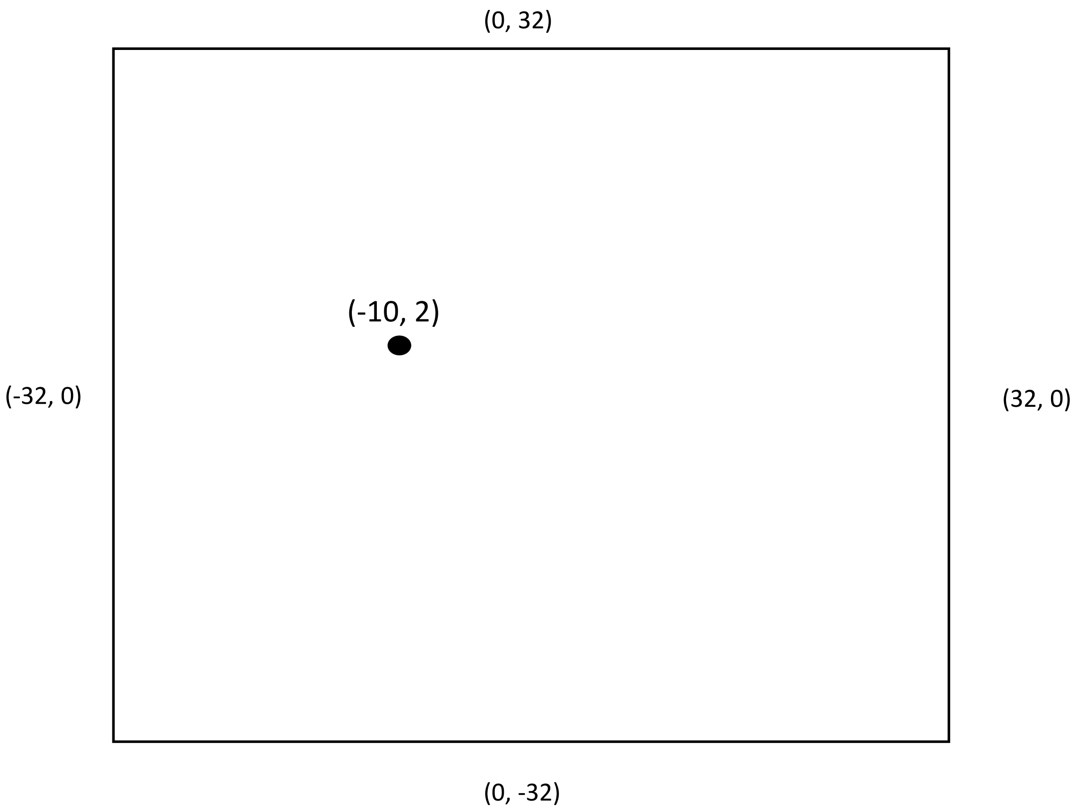


Suppose we select $k = 6 \Rightarrow 2^k = 64$...

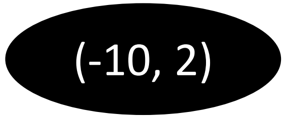
No stored points yet: All we have is a **<u>white</u>** node!

Example

Space



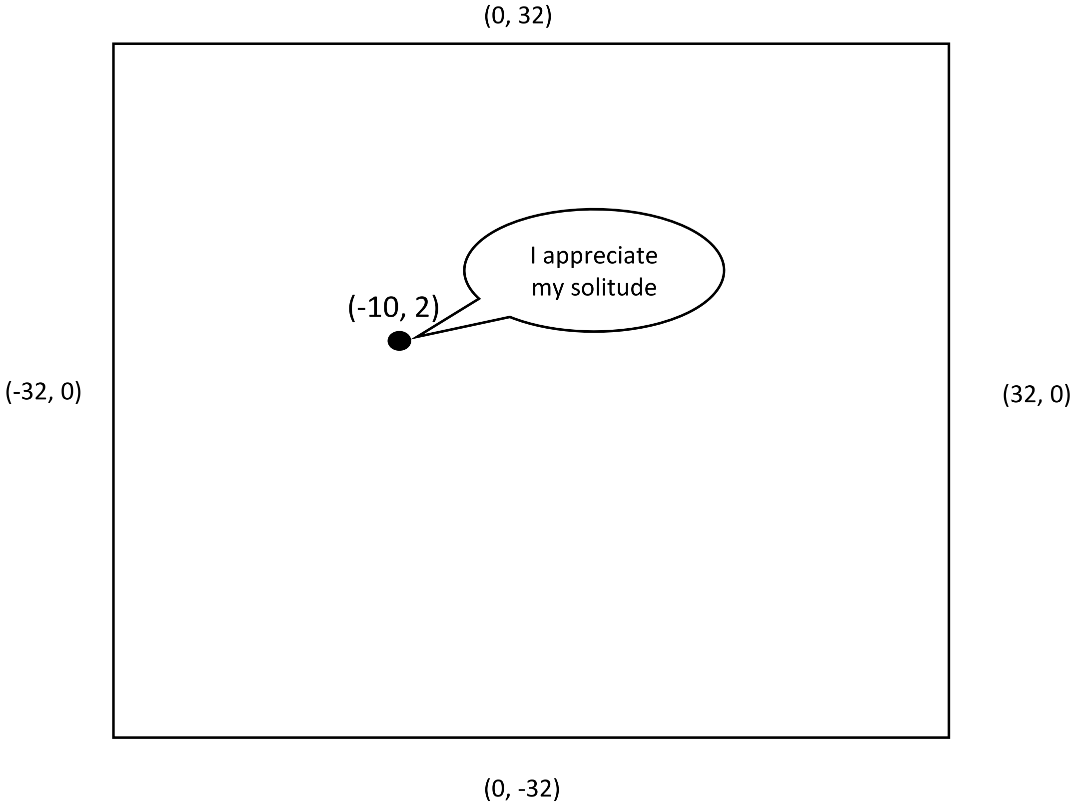
PR-QuadTree



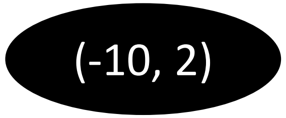
Stored a single point: Entire quadtree is now a **black** node!

Example

Space



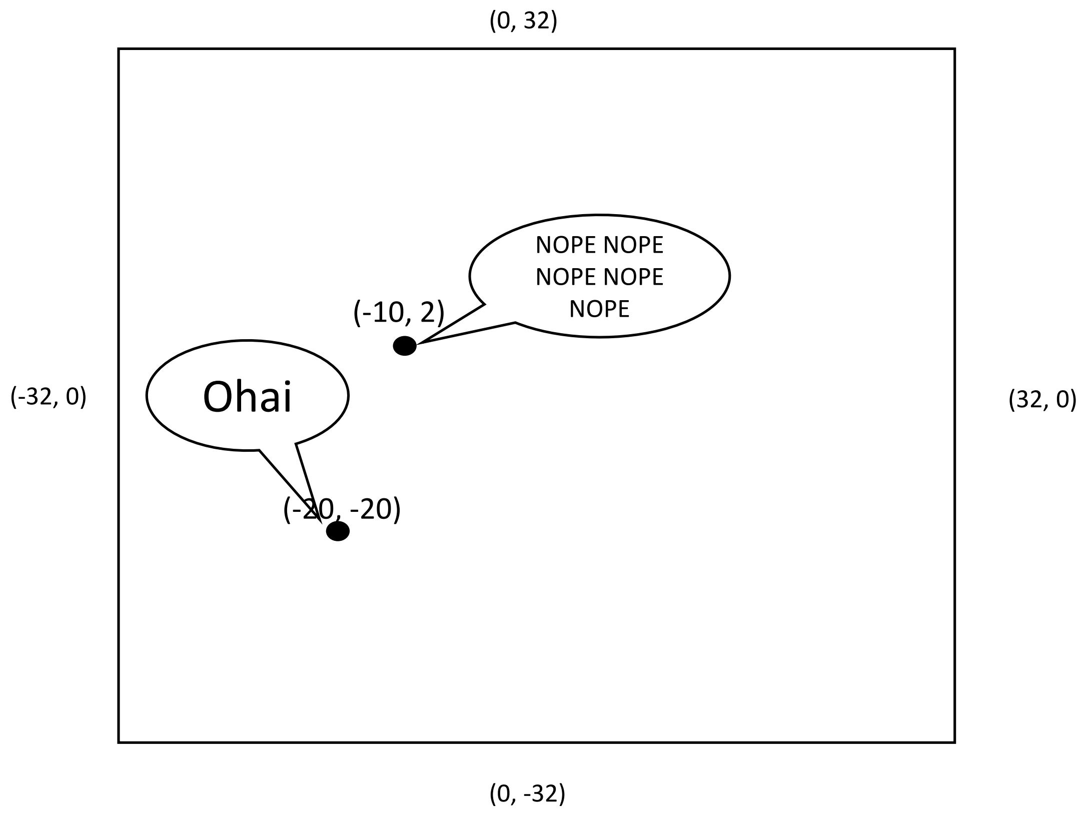
PR-QuadTree



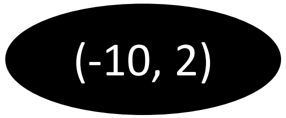
Stored a single point: Entire quadtree is now a **black** node!

Example

Space



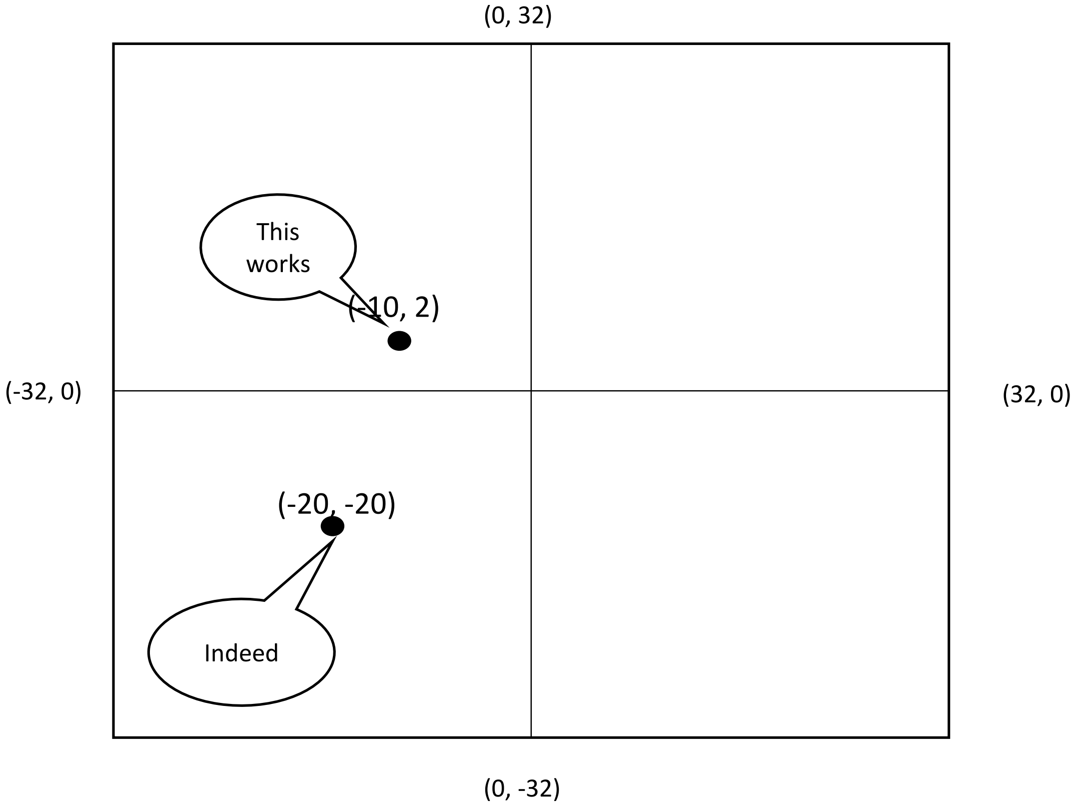
PR-QuadTree



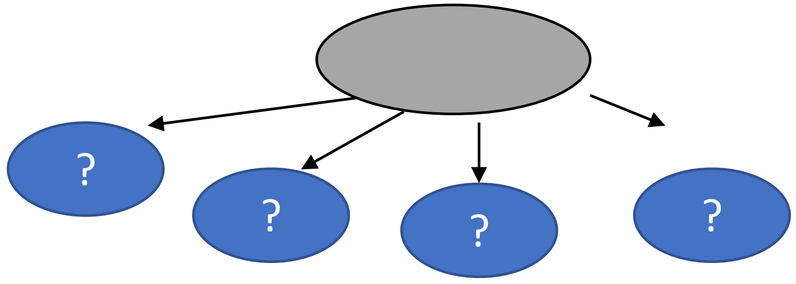
- Inserting (-20, -20) causes **too much commotion** in the black node!
- Split into gray node and 4 children.

Example

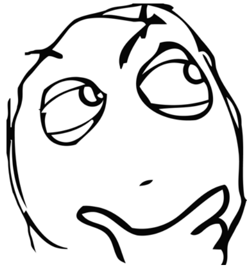
Space



PR-QuadTree



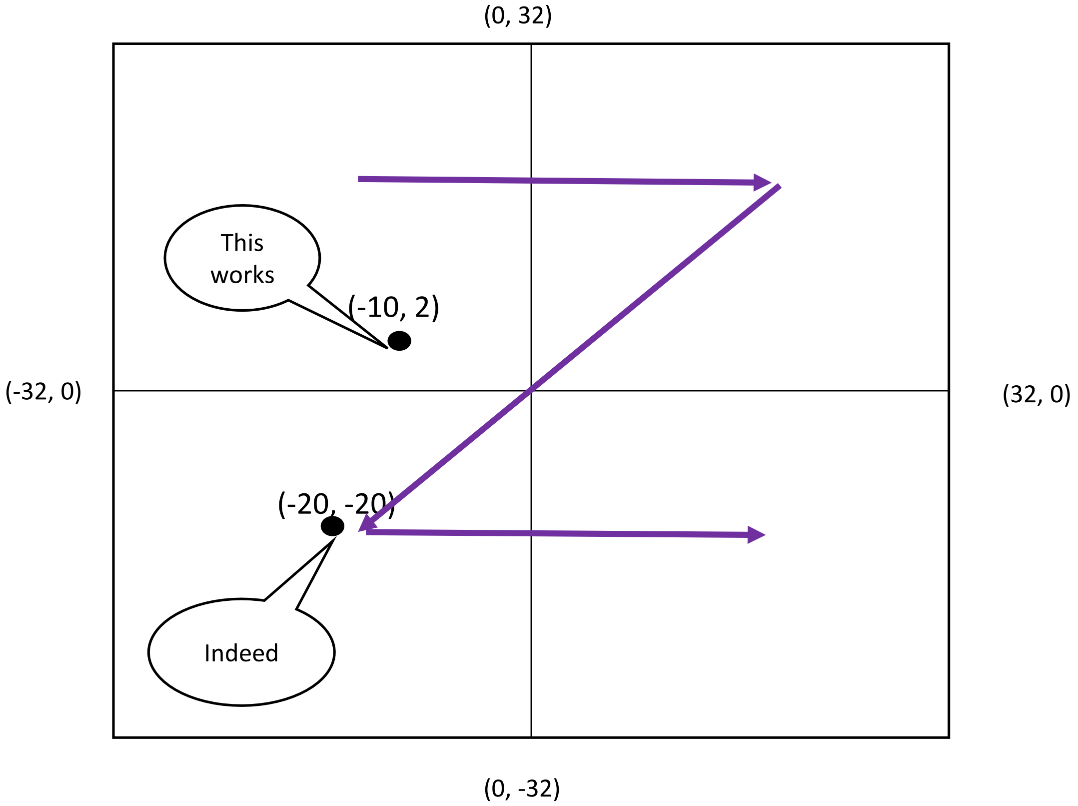
Colors and contents of these nodes?



- Inserting (-20, -20) causes **too much commotion** in the black node!
- Split into gray node and 4 children.

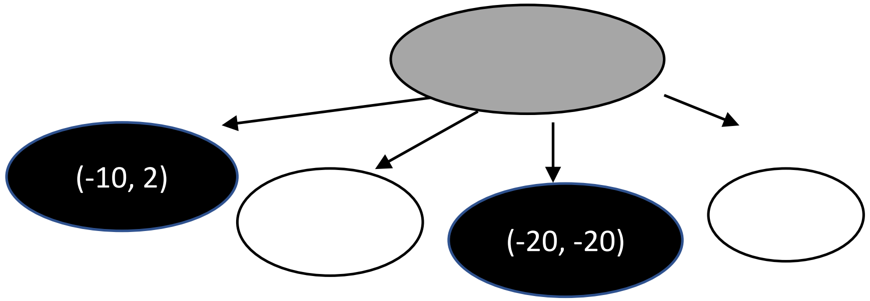
Example

Space



We still do Morton ("Z") order for traversing the children!

PR-QuadTree

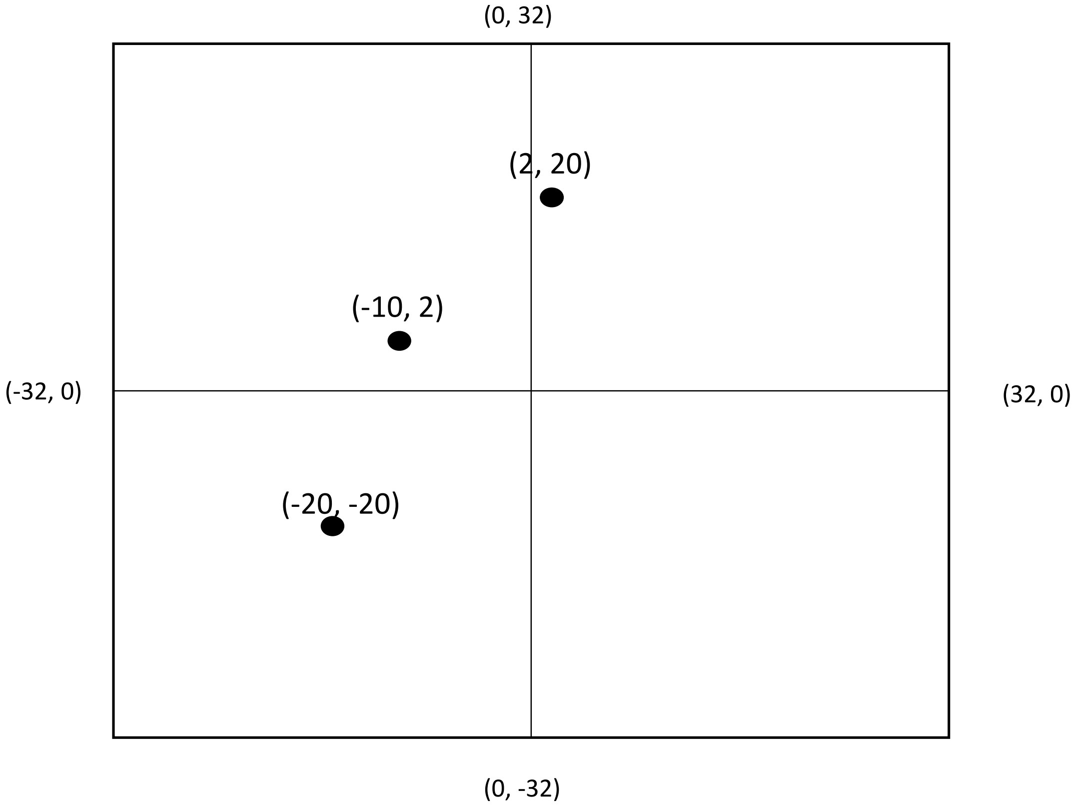


Colors and contents of these nodes?

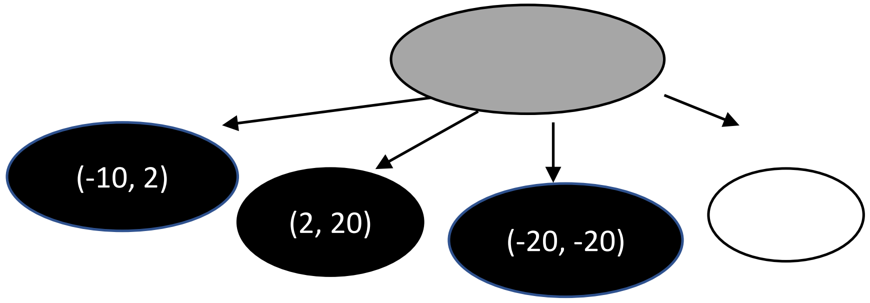
- Inserting (-20, -20) causes **too much commotion** in the black node!
- Split into gray node and 4 children.

Example

Space

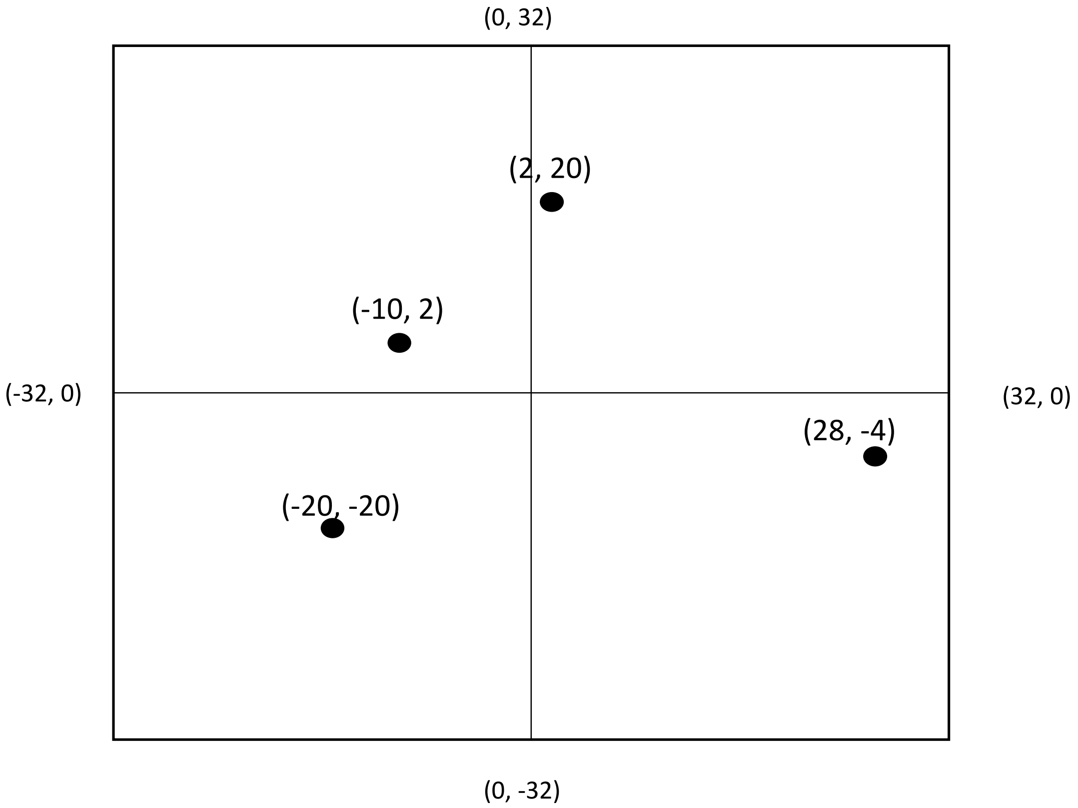


PR-QuadTree

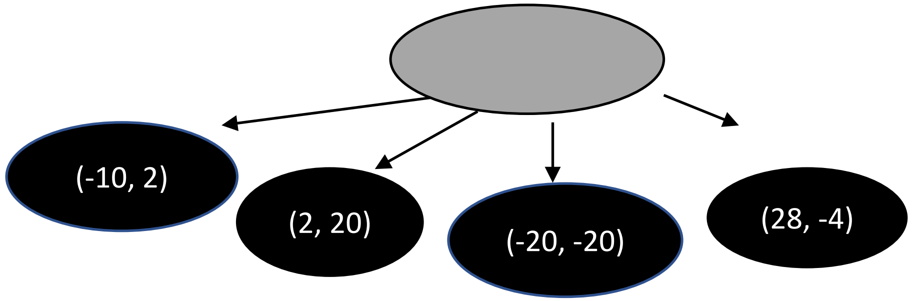


Example

Space

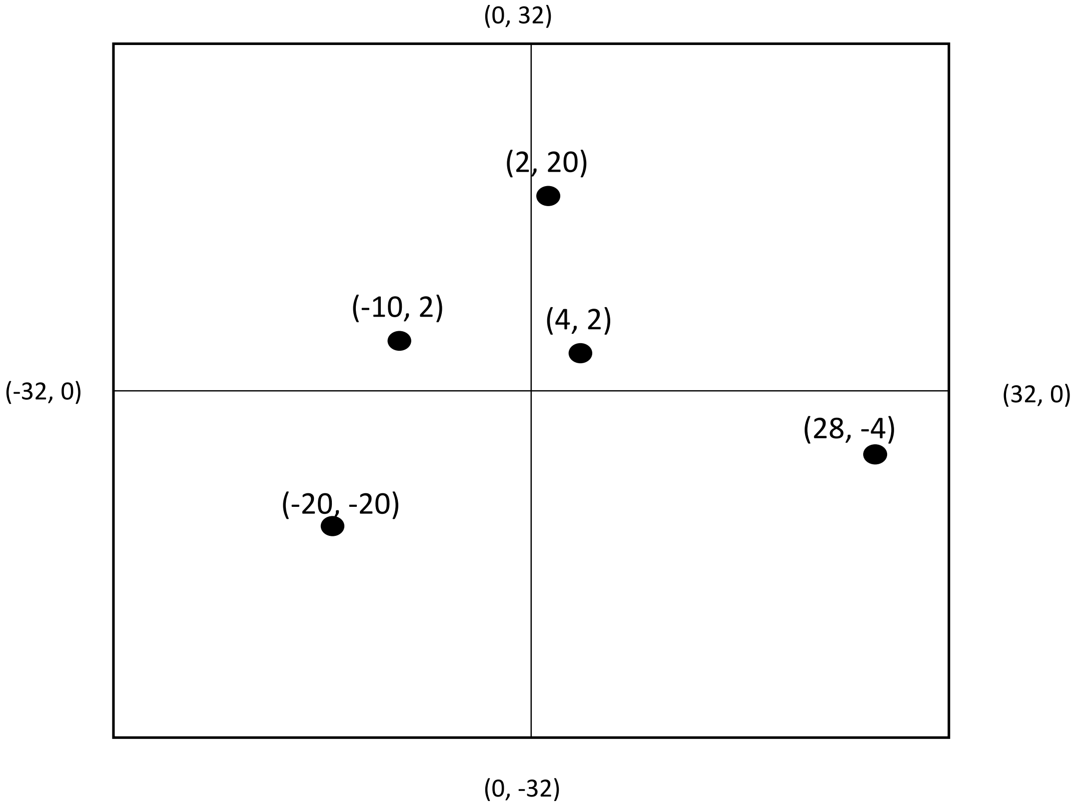


PR-QuadTree

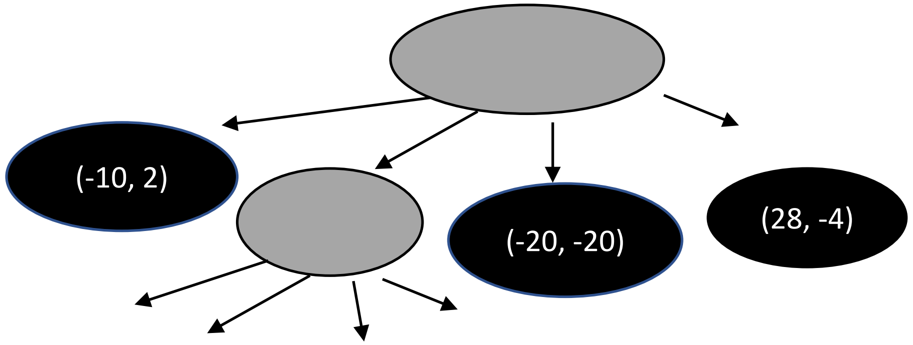


Example

Space

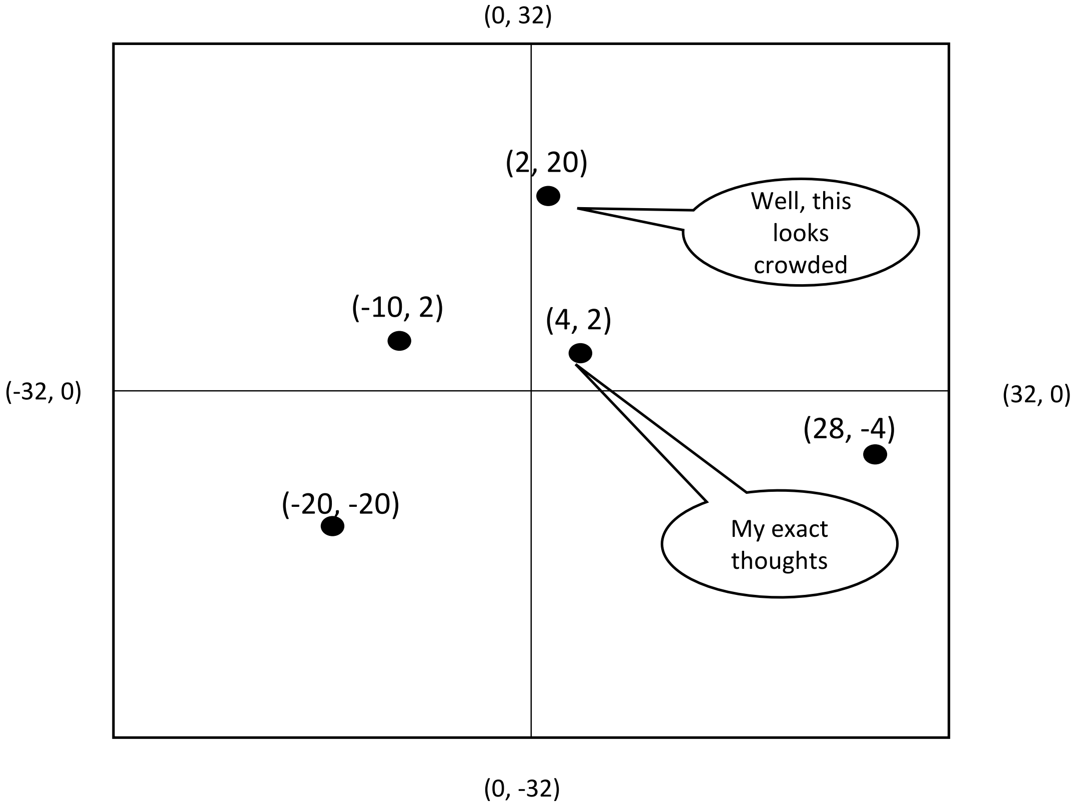


PR-QuadTree

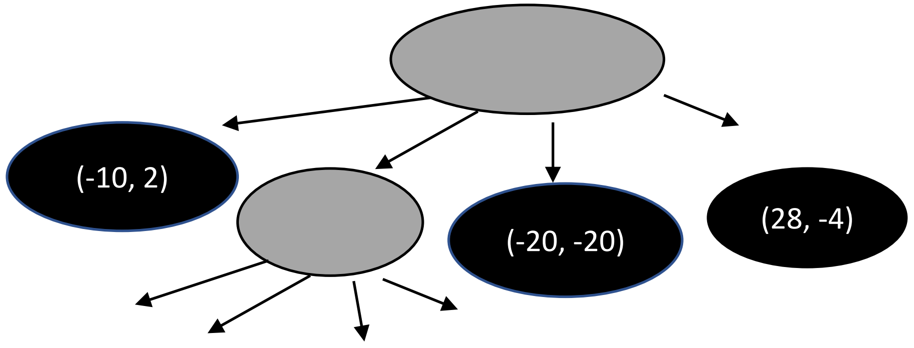


Example

Space

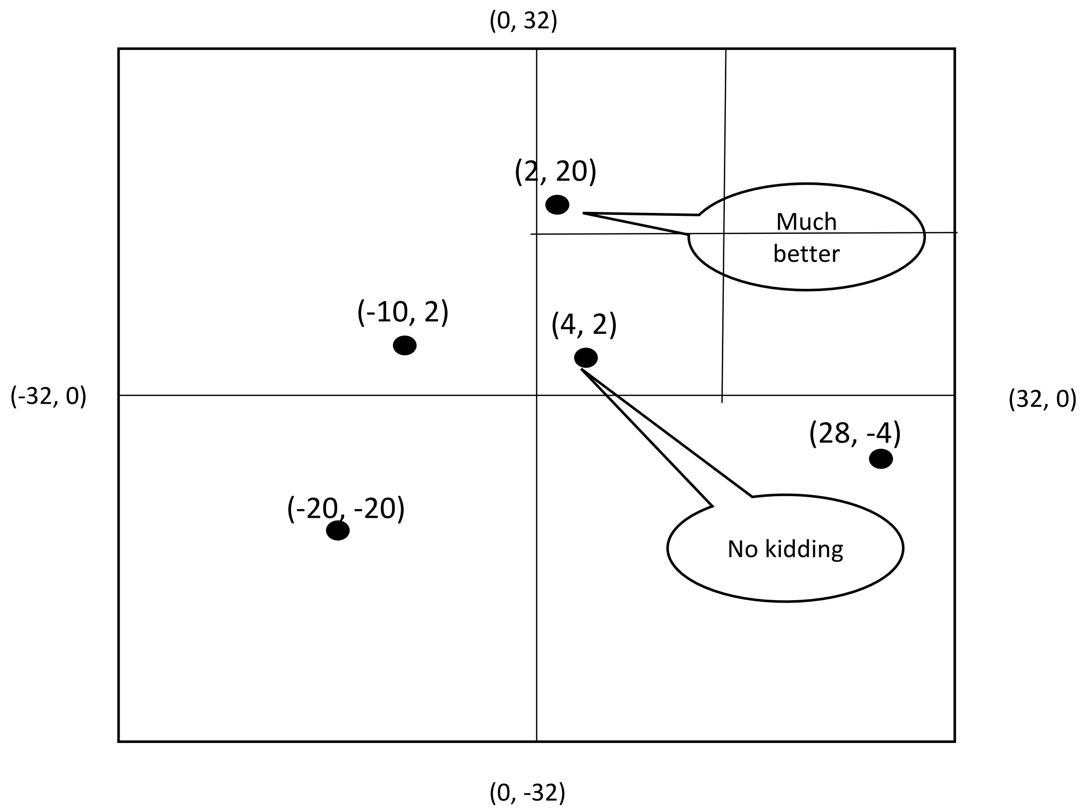


PR-QuadTree

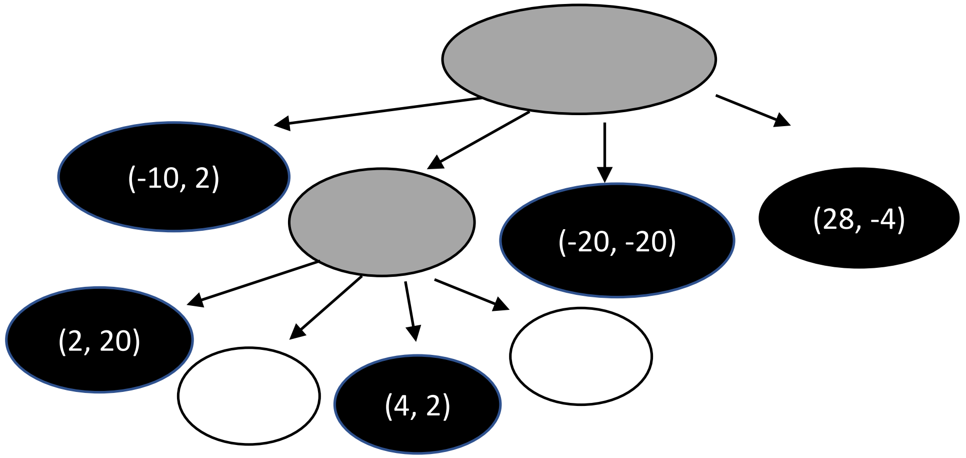


Example

Space

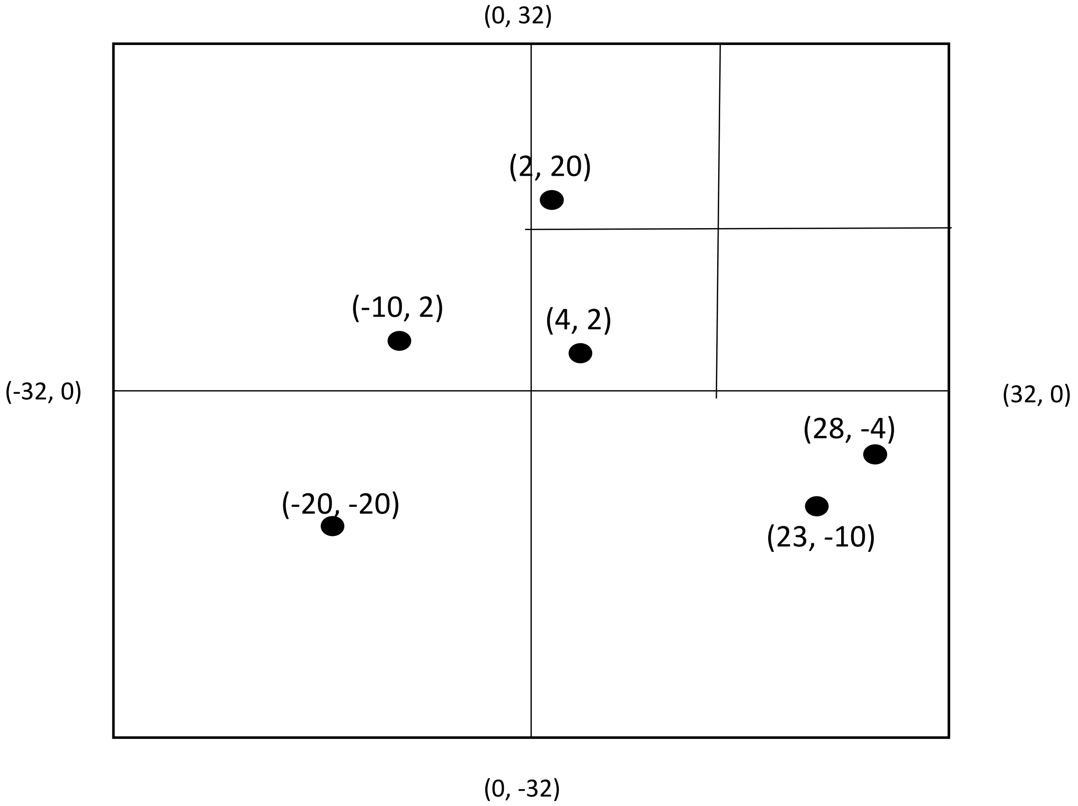


PR-QuadTree

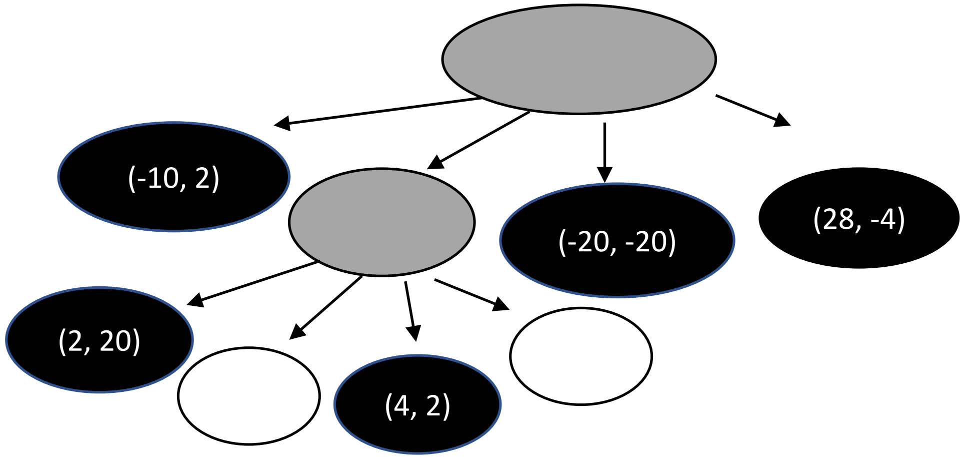


Example

Space

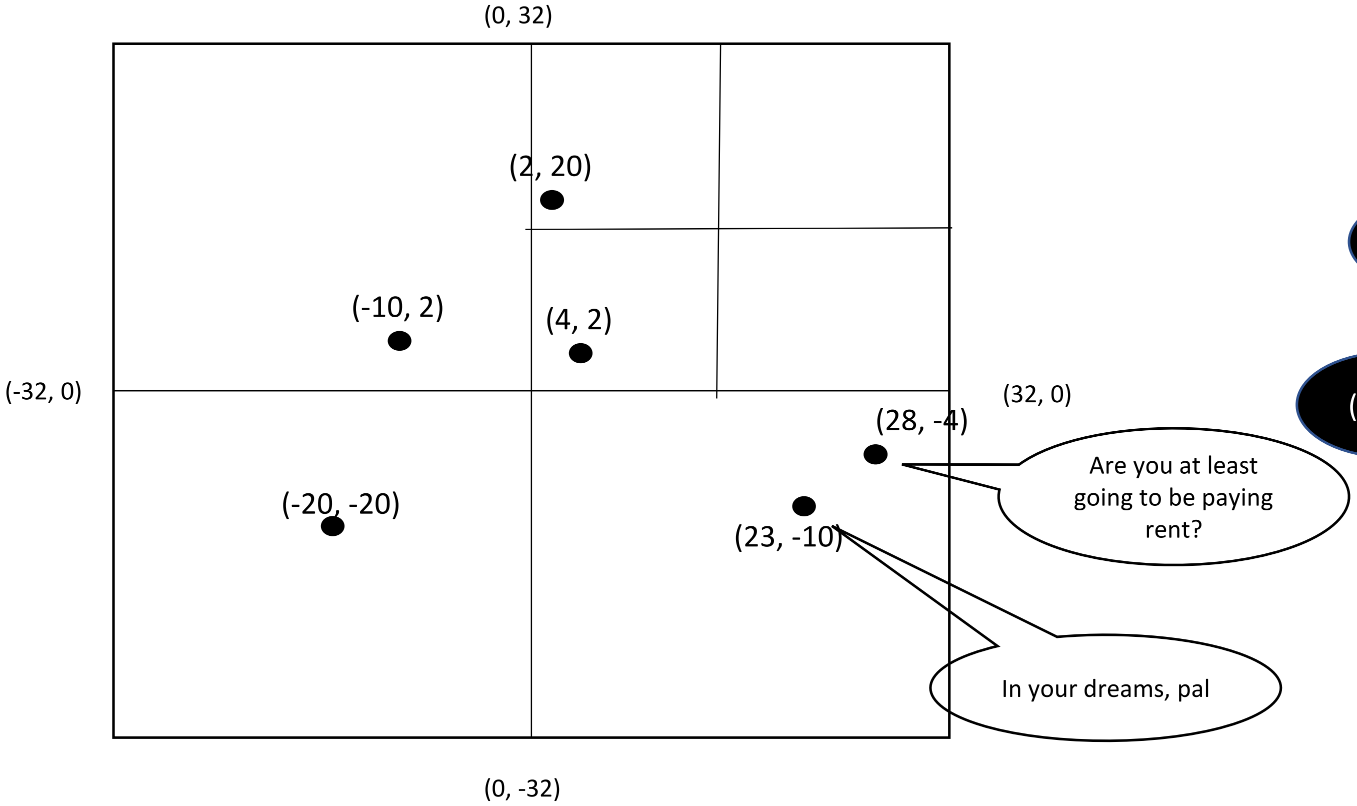


PR-QuadTree

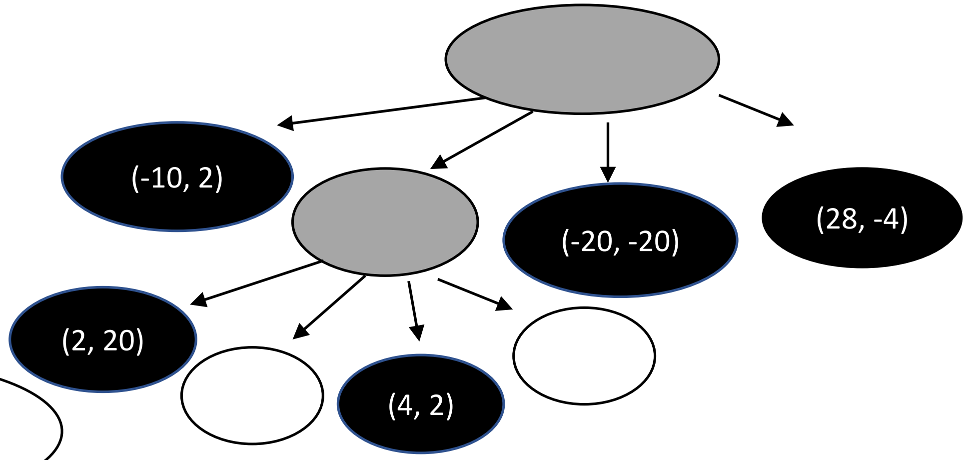


Example

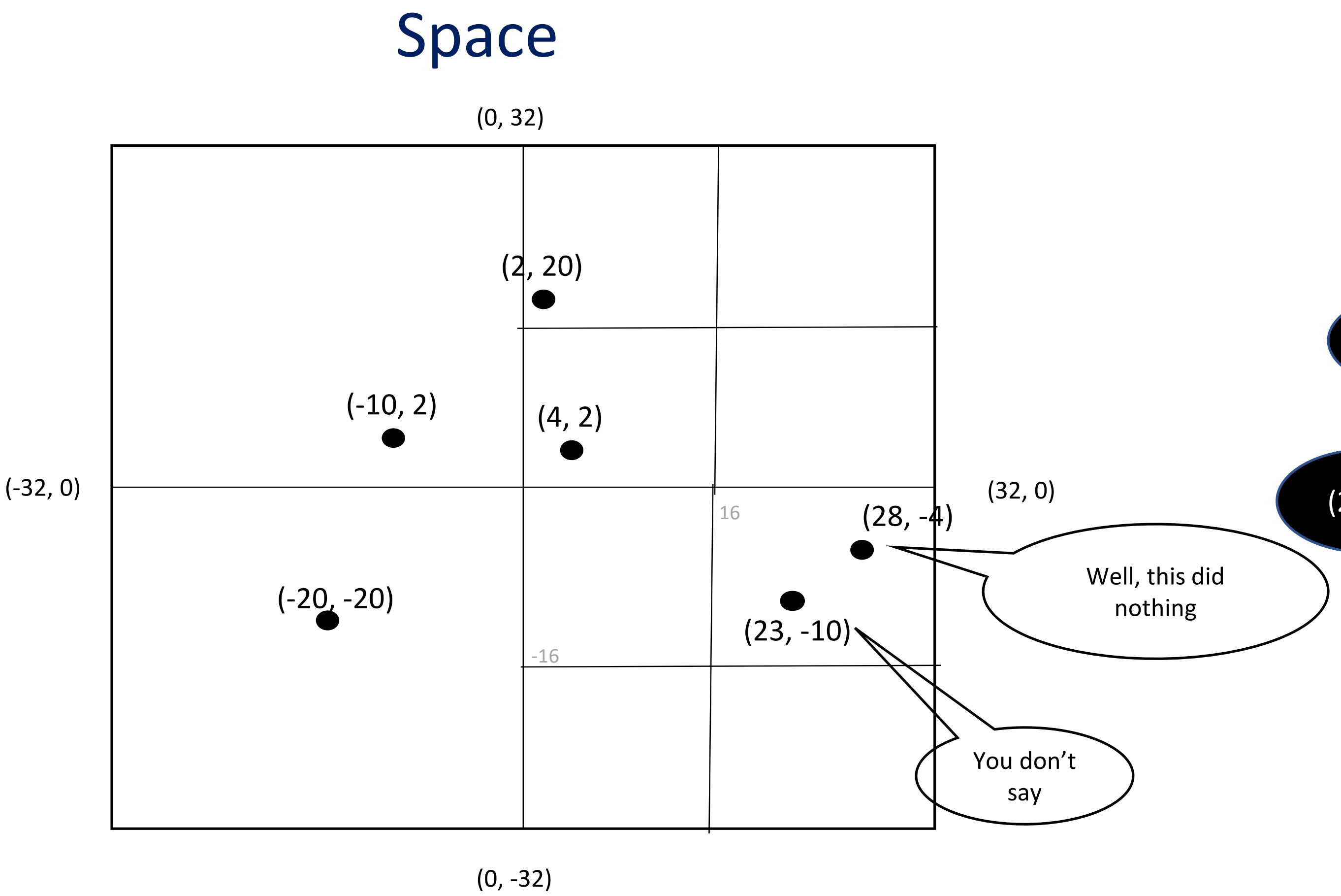
Space

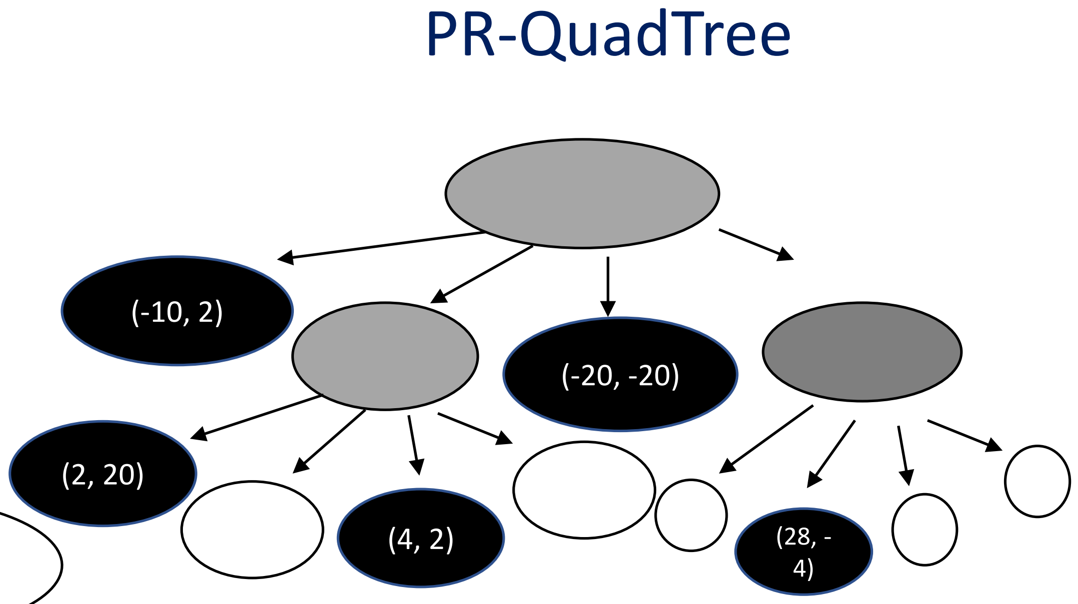


PR-QuadTree



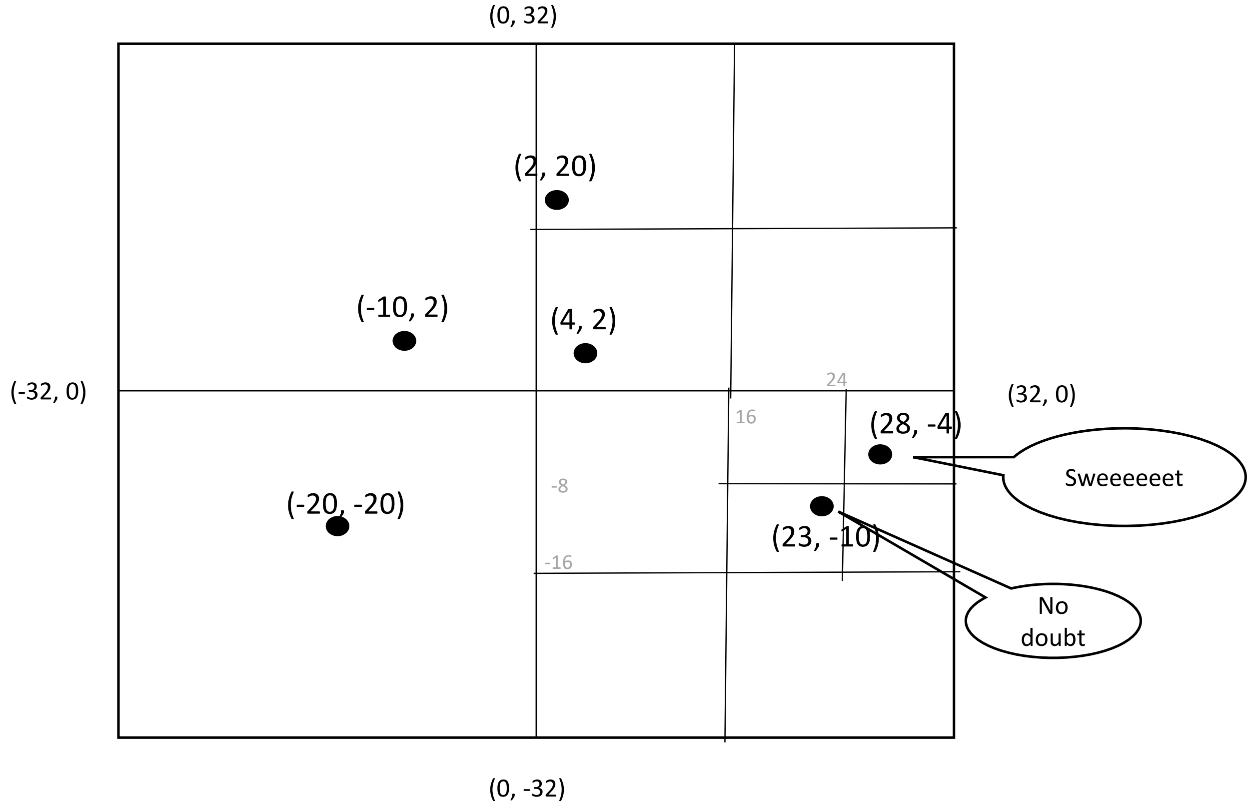
Example


Space
(0, 32)
(2, 20)
(-10, 2)
(4, 2)
(-32, 0)
(32, 0)
16
(28, -4)
Well, this did nothing
(-20, -20)
(23, -10)
-16
You don't say
(0, -32)

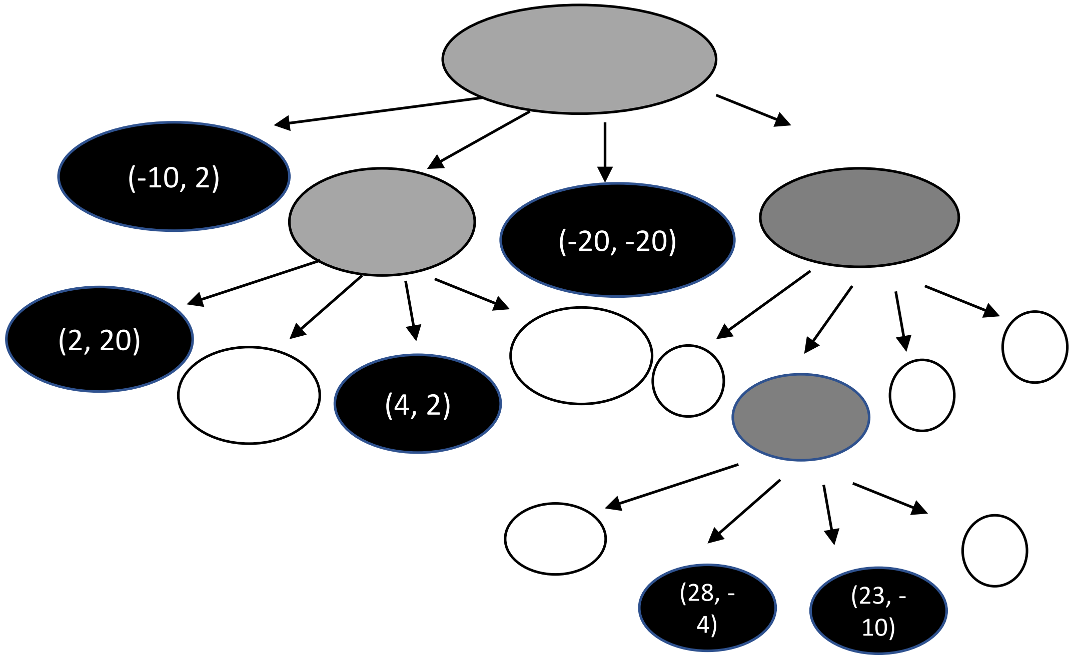

PR-QuadTree
(-10, 2)
(-20, -20)
(2, 20)
(4, 2)
(28, -
4)

Example

Space



PR-QuadTree



Splitting at 2 all the time ?

- This aggressive splitting seems wasteful.
- When inserting 2 points *very close* to each other, we might even have to investigate negative powers of 2!

Splitting at 2 all the time ?

- This aggressive splitting seems wasteful.
- When inserting 2 points *very close* to each other, we might even have to investigate negative powers of 2!
- Two ways to deal with this:

Splitting at 2 all the time ?

- This aggressive splitting seems wasteful.
- When inserting 2 points *very close* to each other, we might even have to investigate negative powers of 2!
- Two ways to deal with this:
  1. **Bucketing (your project):** Treat black nodes as buckets of some provided capacity $b \geq 1$.
     - **Black** nodes can now store b points.

Splitting at 2 all the time ?

- This aggressive splitting seems wasteful.
- When inserting 2 points *very close* to each other, we might even have to investigate negative powers of 2!
- Two ways to deal with this:
  1. **Bucketing (your project)**: Treat black nodes as buckets of some provided capacity $b \geq 1$.
     - **Black** nodes can now store b points.
  2. **Split at positions other than powers of 2** to better fit your data
     - This is the approach of "loose" PR-QuadTrees
     - We will not talk about those, at least not in lecture.

Exercise!

- Insert the following points into an initially empty PR-QuadTree.
  - Parameters:
    - $k = 5$
    - $b = 2$
    - Center of cartesian space is (0, 0)
  - Points: `(20,30), (-20,30), (5,-5), (-8,-2), (-9,-19), (1, 13), (1,16), (5,9), (5,5)`

Conjectures

- Gray nodes with 3 white children and 1 black child are **useless** and can be **pruned**.

True

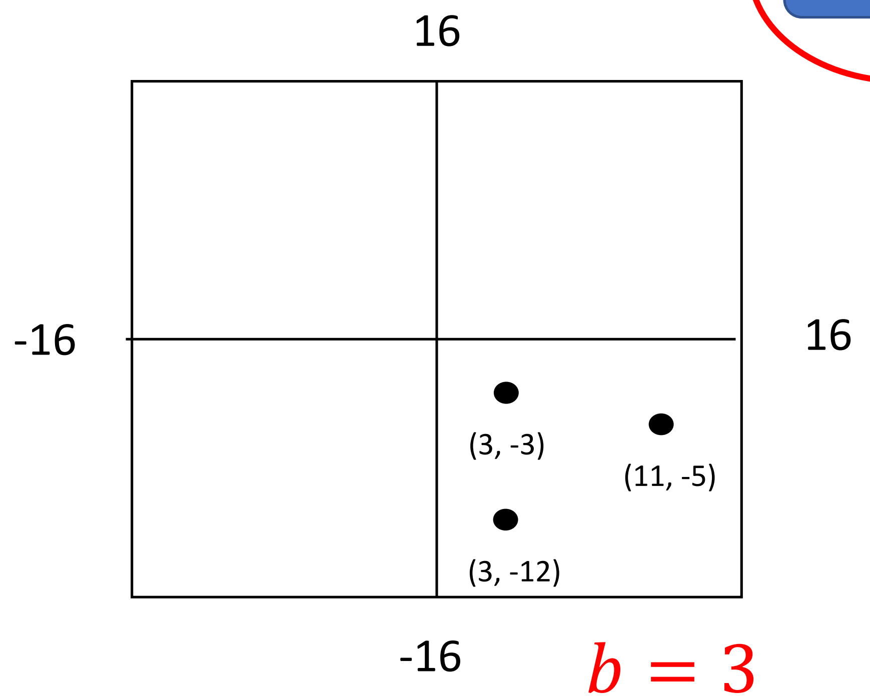
False

Conjectures

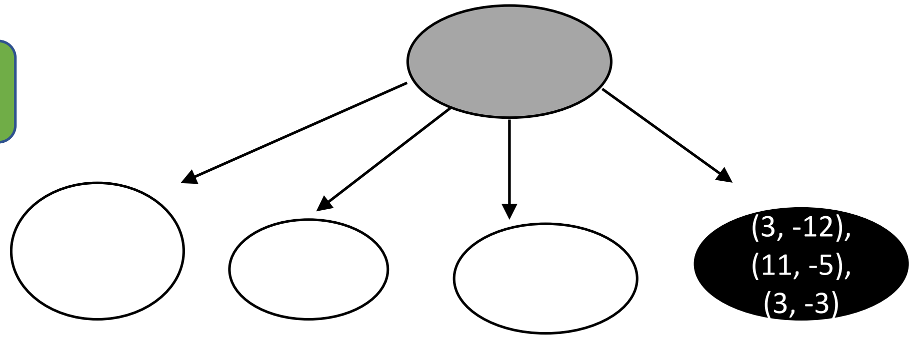
- Gray nodes with 3 white children and 1 black child are **useless** and can be **pruned**.

True

False



$b = 3$

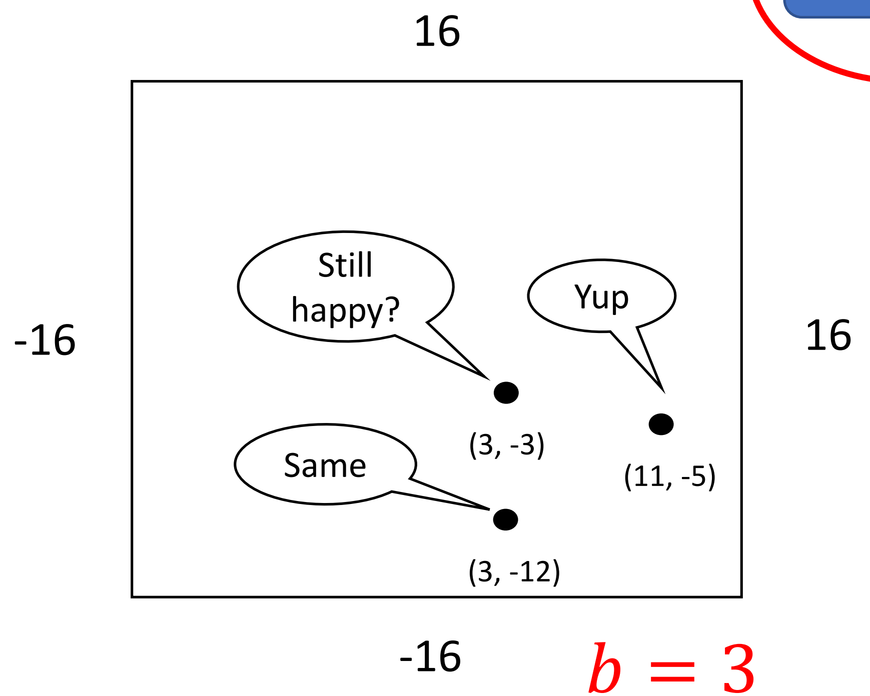


Conjectures

- Gray nodes with 3 white children and 1 black child are **useless** and can be **pruned**.

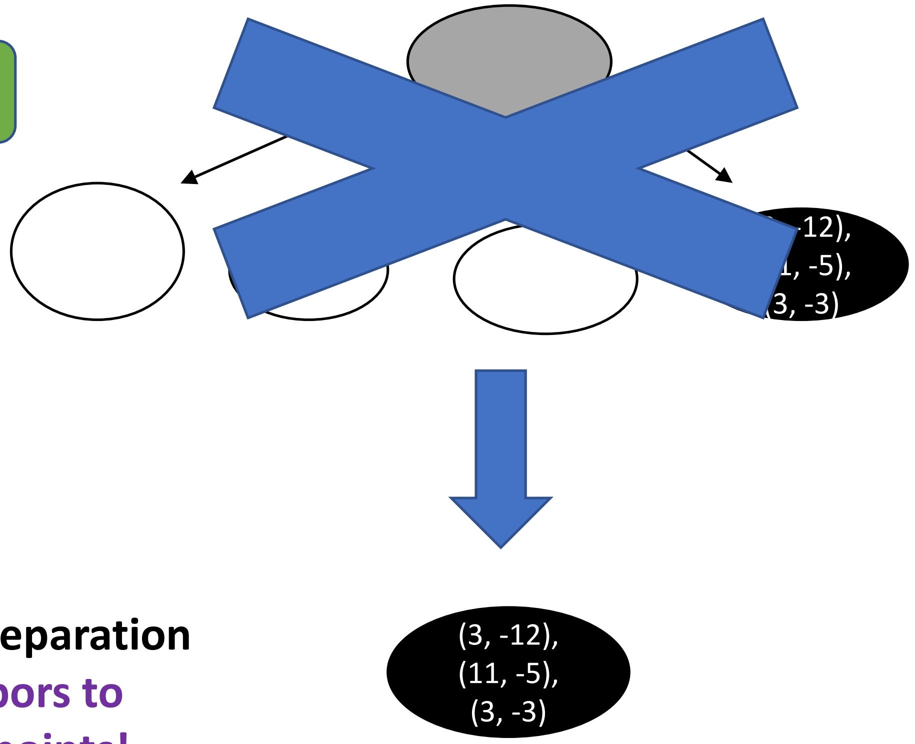
True

False



$b = 3$

No need for further separation
if there are no neighbors to “annoy” the current points!



Conjectures

- Gray nodes with only black and white children are **useless** and can be **pruned**.

True

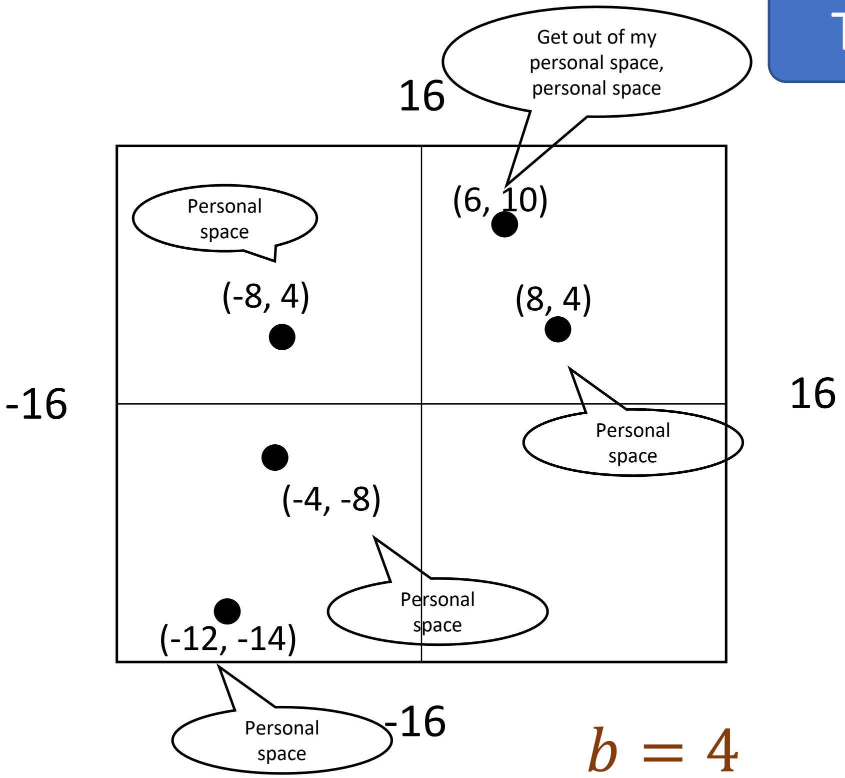
False

Conjectures

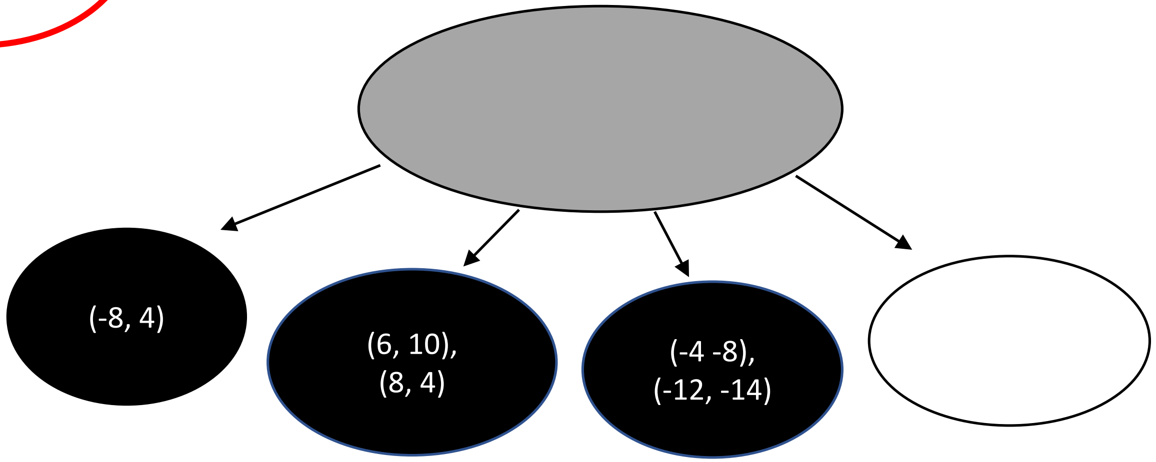
- Gray nodes with only black and white children are **useless** and can be **pruned**.

True

False



$b = 4$



What if merging them all into a new black node surpasses parameter b?

Conjectures

- Gray nodes with only black and white children, where the black nodes ***collectively*** contain $\leq b$ points are useless and can be pruned.

True

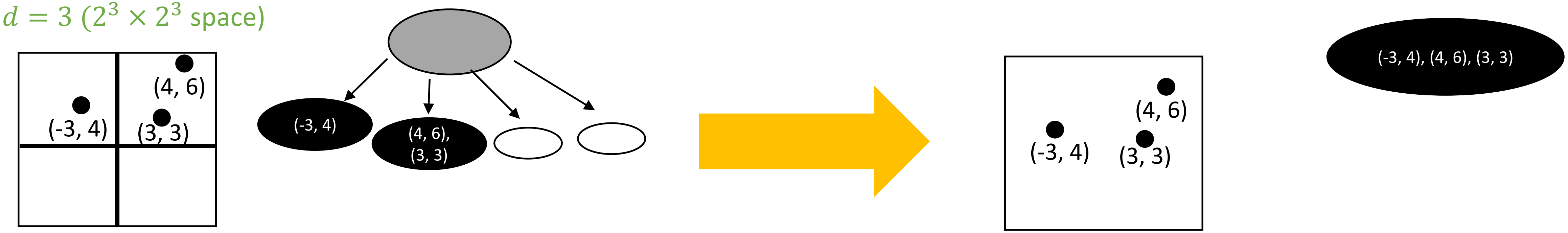
False

Conjectures

- Gray nodes with only black and white children, where the black nodes ***collectively*** contain $\leq b$ points are **useless** and can be **pruned**.

True False

$d = 3$ ($2^3 \times 2^3$ space)



$b = 3$

Conjectures

- In a PR-QuadTree, black and white nodes can only be *leaf nodes.*

True

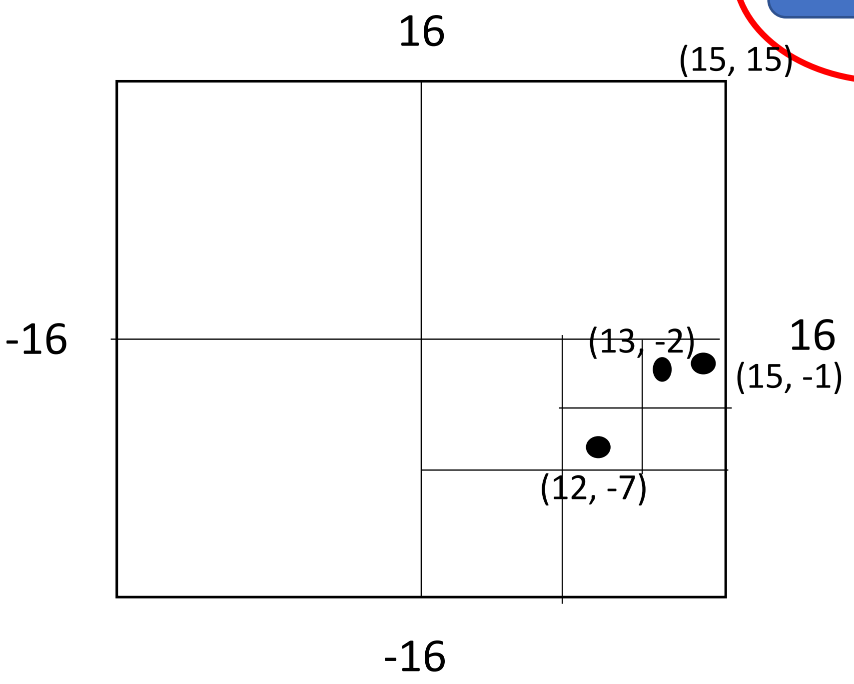
False

Conjectures

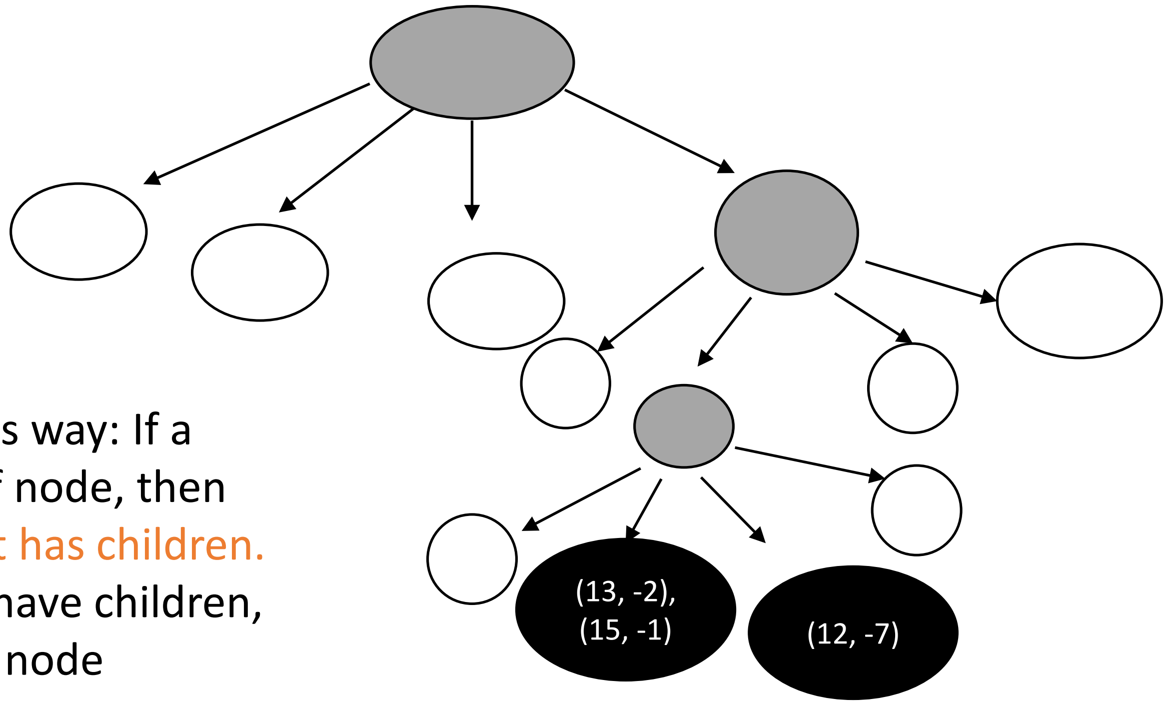
- In a PR-QuadTree, black and white nodes can only be *leaf nodes.*

True

False



Think about it this way: If a node is not a leaf node, then this means that it has children. In order for it to have children, it ***must*** be a grey node



Deletion…

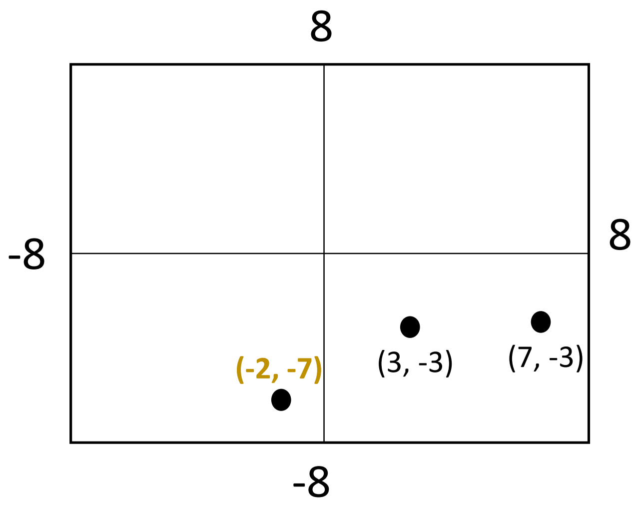
- Insertion of a $(b + 1)$th point into a **black node** *splitted* the node into a gray node with 4 children…
- Symmetrically, deletion of the last point from a **black** node will turn it into a white node

Deletion…

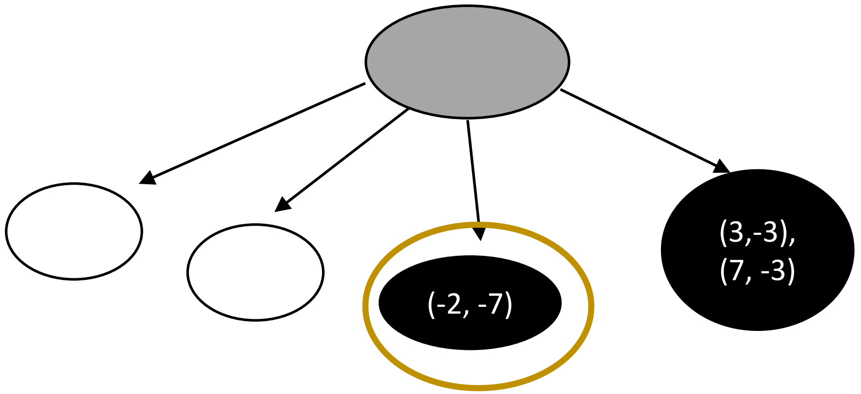
- Insertion of a $(b+1)$th point into a **black node** *splitted* the node into a gray node with 4 children…
- Symmetrically, deletion of the last point from a **black** node will turn it into a white node
- This might trigger a collapse of the parent!

Deletion…

- Insertion of a $(b+1)$th point into a **black node** *splitted* the node into a gray node with 4 children…
- Symmetrically, deletion of the last point from a **black** node will turn it into a white node
- This might trigger a collapse of the parent!
- E.g deletion of (-2, -7) below

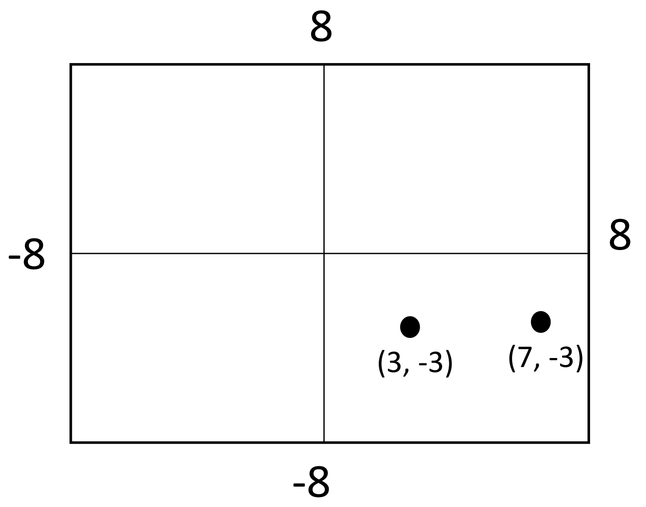


$b = 2$

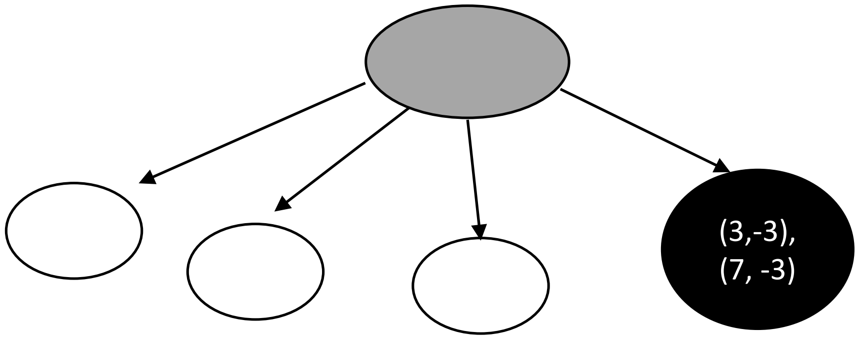


Deletion…

- Insertion of a $(b+1)$th point into a **black node** *splitted* the node into a gray node with 4 children…
- Symmetrically, deletion of the last point from a **black** node will turn it into a white node
- This might trigger a collapse of the parent!
- E.g deletion of (-2, -7) below

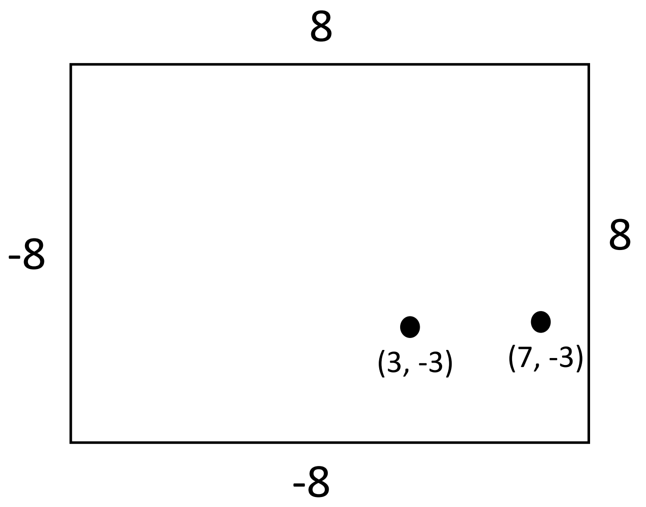


$b = 2$

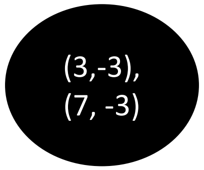


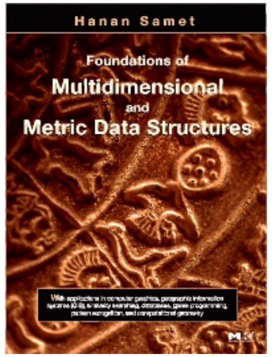
Deletion…

- Insertion of a $(b+1)$th point into a **black node** *splitted* the node into a gray node with 4 children…
- Symmetrically, deletion of the last point from a **black** node will turn it into a white node
- This might trigger a collapse of the parent!
- E.g deletion of (-2, -7) below



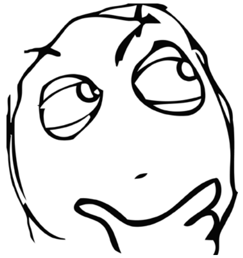
$b = 2$

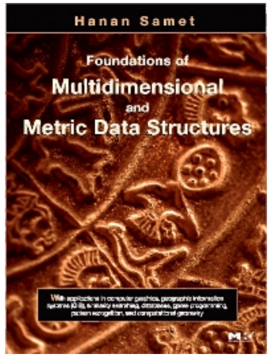




“Trie – Based” quadtrees?

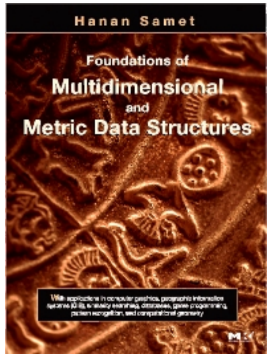
- Many resources, including *the bible*, call quadtrees like the PR “trie-based” quadtrees.





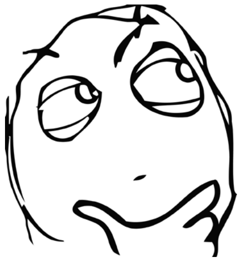
"Trie – Based" quadtrees?

- Many resources, including *the bible*, call quadtrees like the PR "trie-based" quadtrees.
- For a long time, Jason could not understand why.



“Trie – Based” quadtrees?

- Many resources, including *the bible*, call quadtrees like the PR “trie-based” quadtrees.
- For a long time, Jason could not understand why.
- **Let’s see if you can!**
- True or False: The worst-case time complexity of `search` depends on the number n of points stored in the PR-Quadtree.



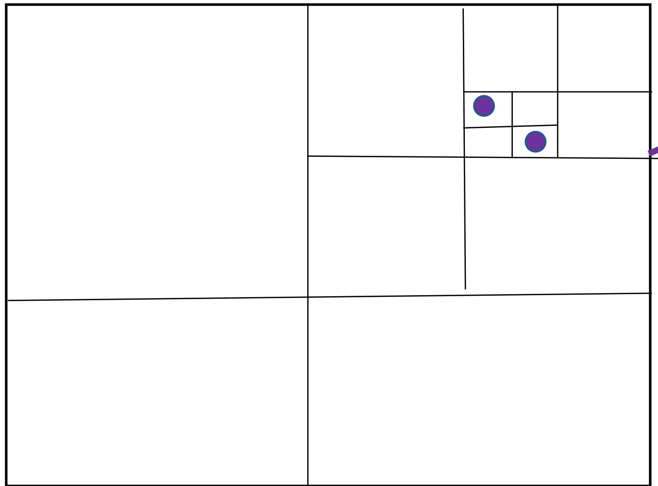
True
(Give me a $\mathcal{O}(f(n))$)

False
(What does it depend on, then?)

“Trie – Based” quadtrees?

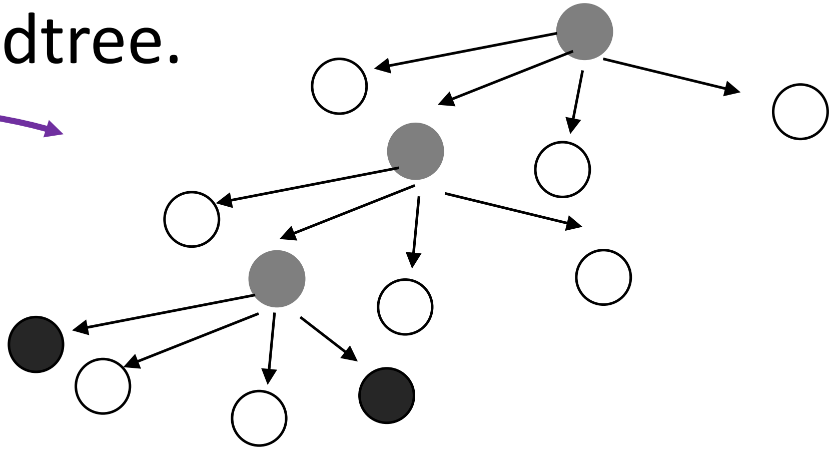
- Many resources, including *the bible*, call quadtrees like the PR “trie-based” quadtrees.
- For a long time, Jason could not understand why.
- **Let’s see if you can!**

- True or False: The worst-case time complexity of `search` depends on the number n of points stored in the PR-Quadtree.



True
(Give me a
$\mathcal{O}(f(n))$)

False
(What does it depend on, then?)



Worst case produced by the ***minimum distance*** between two points!

Spatial Queries in PR-Quadtrees

1. ***Descending phase:*** Still **aggressively descending** towards anchor point, to cover as many points as we can in a short amount of time.

Spatial Queries in PR-Quadtrees

1. ***Descending phase:*** Still **aggressively descending** towards anchor point, to cover as many points as we can in a short amount of time.
2. ***Backtracking phase:*** One difference with KD-Trees
   - KD-Trees: Ask the question: *"Does it make sense for me to visit the **opposite subtree**"?*
   - It **doesn't** make sense if:
     - We have a **range query** and the separating hyperplane's (line in 2D) "current dimension" value is `> anchor.coords[currDim]+ range (<, - respectively)`
     - We have a ***k*-NN** query and the "worst candidate circle" (i.e the distance - priority - of the *current* ***worst*** *neighbor within the* k best neighbors) has the same properties as above.

Spatial Queries in PR-Quadtrees

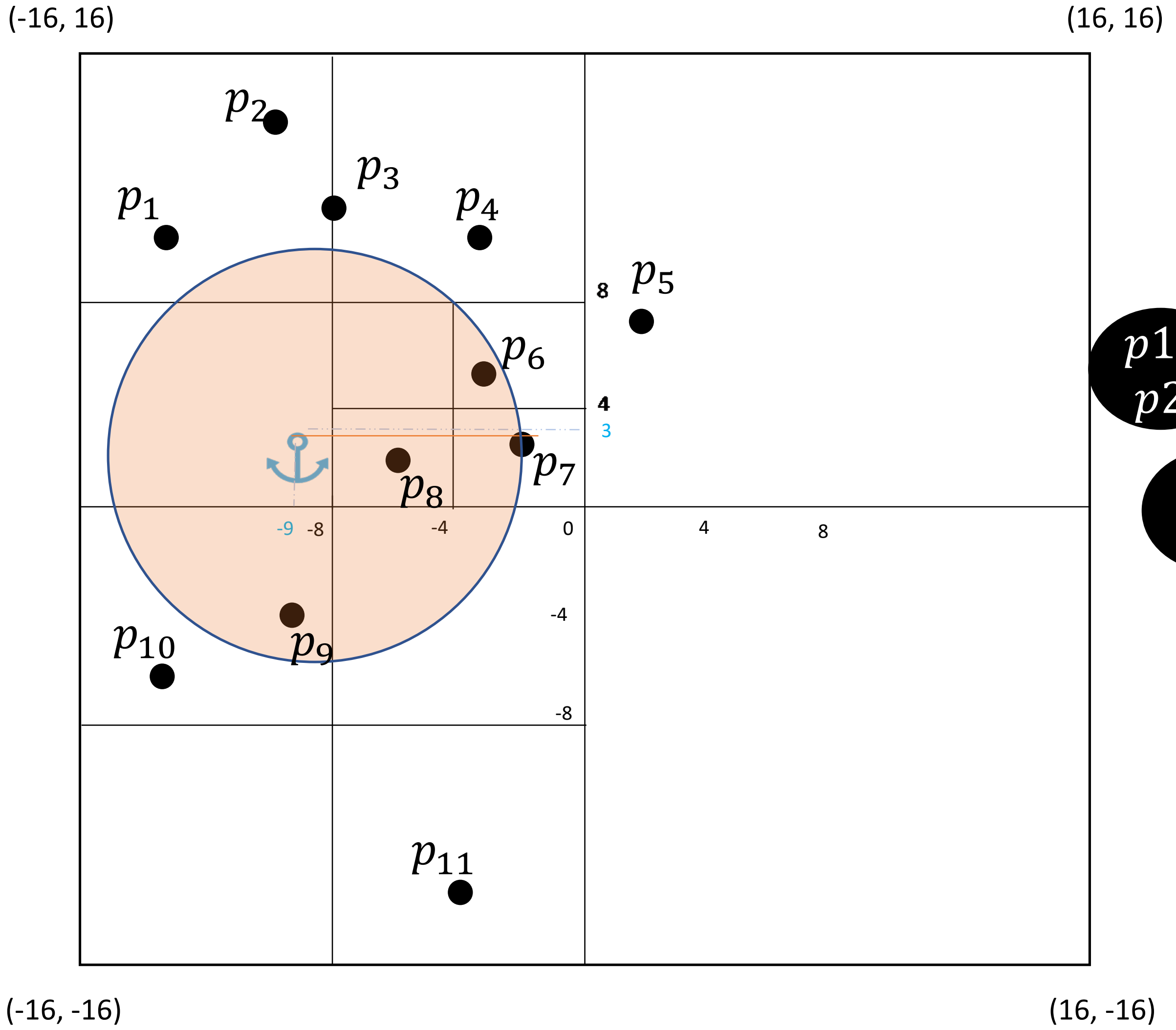
1. ***Descending phase:*** Still **aggressively descending** towards anchor point, to cover as many points as we can in a short amount of time.
2. ***Backtracking phase:*** One difference with KD-Trees
   - KD-Trees: Ask the question: *"Does it make sense for me to visit the* ***opposite subtree****"?*
   - It **doesn't** make sense if:
     - We have a **range query** and the separating hyperplane's (line in 2D) "current dimension" value is `> anchor.coords[currDim]+ range (<, -` respectively)
     - We have a ***k*-NN** query and the "worst candidate circle" (i.e the distance - priority - of the *current* ***worst*** *neighbor within the* k best neighbors) has the same properties as above.
   - In PR-QuadTrees, we have three different other subtrees to consider!

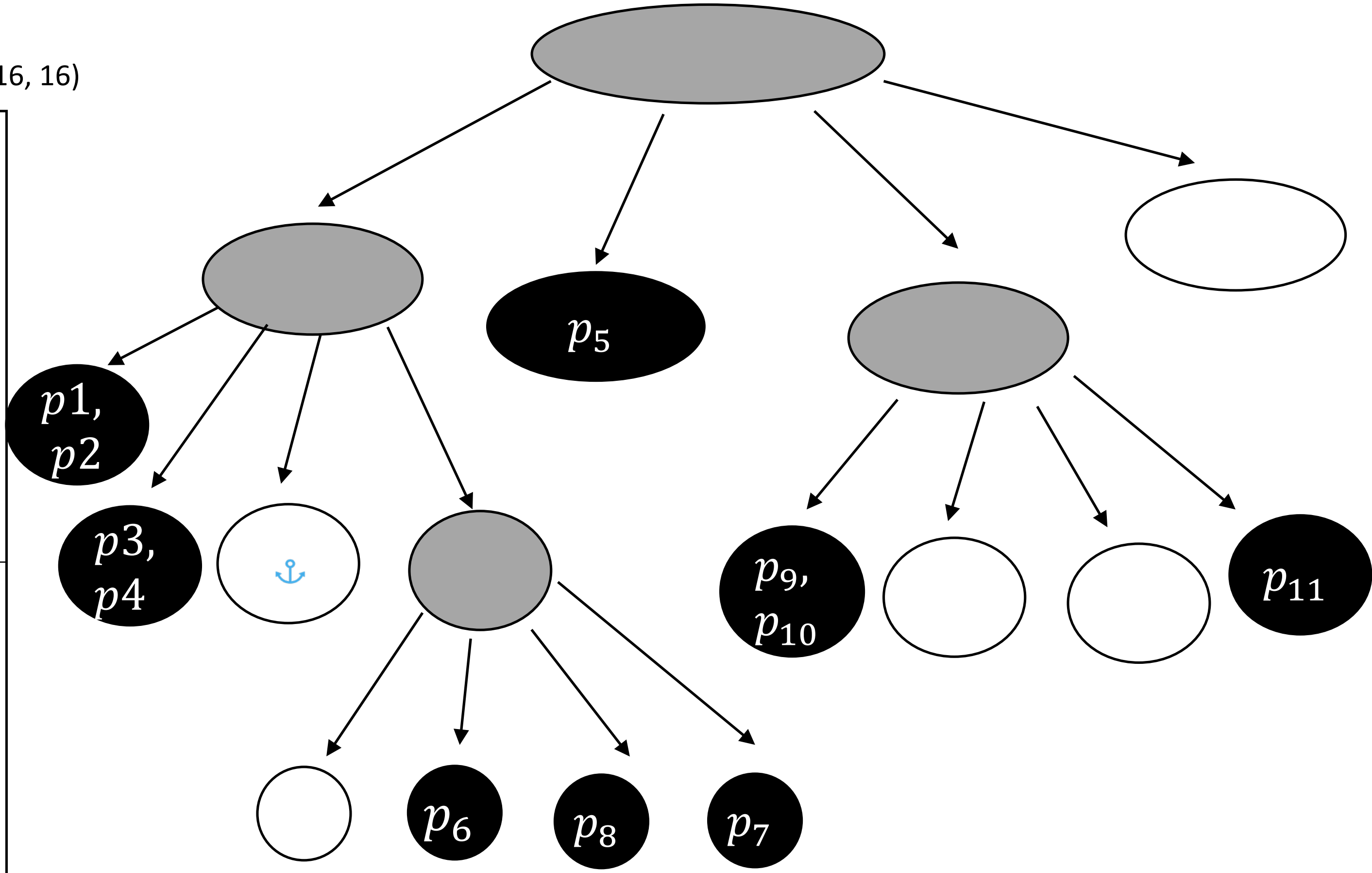
Spatial Queries in PR-Quadtrees

1. ***Descending phase:*** Still **aggressively descending** towards anchor point, to cover as many points as we can in a short amount of time.
2. ***Backtracking phase:*** One difference with KD-Trees
   - KD-Trees: Ask the question: *"Does it make sense for me to visit the* ***opposite subtree****"?*
   - It **doesn't** make sense if:
     - We have a **range query** and the separating hyperplane's (line in 2D) "current dimension" value is `> anchor.coords[currDim]+ range (<, - respectively)`
     - We have a ***k*-NN** query and the "worst candidate circle" (i.e the distance - priority - of the *current* ***worst*** *neighbor within the* k best neighbors) has the same properties as above.
   - In PR-QuadTrees, we have three different other subtrees to consider!
     - ***How should we proceed?***

Example of range query in PR-QuadTree

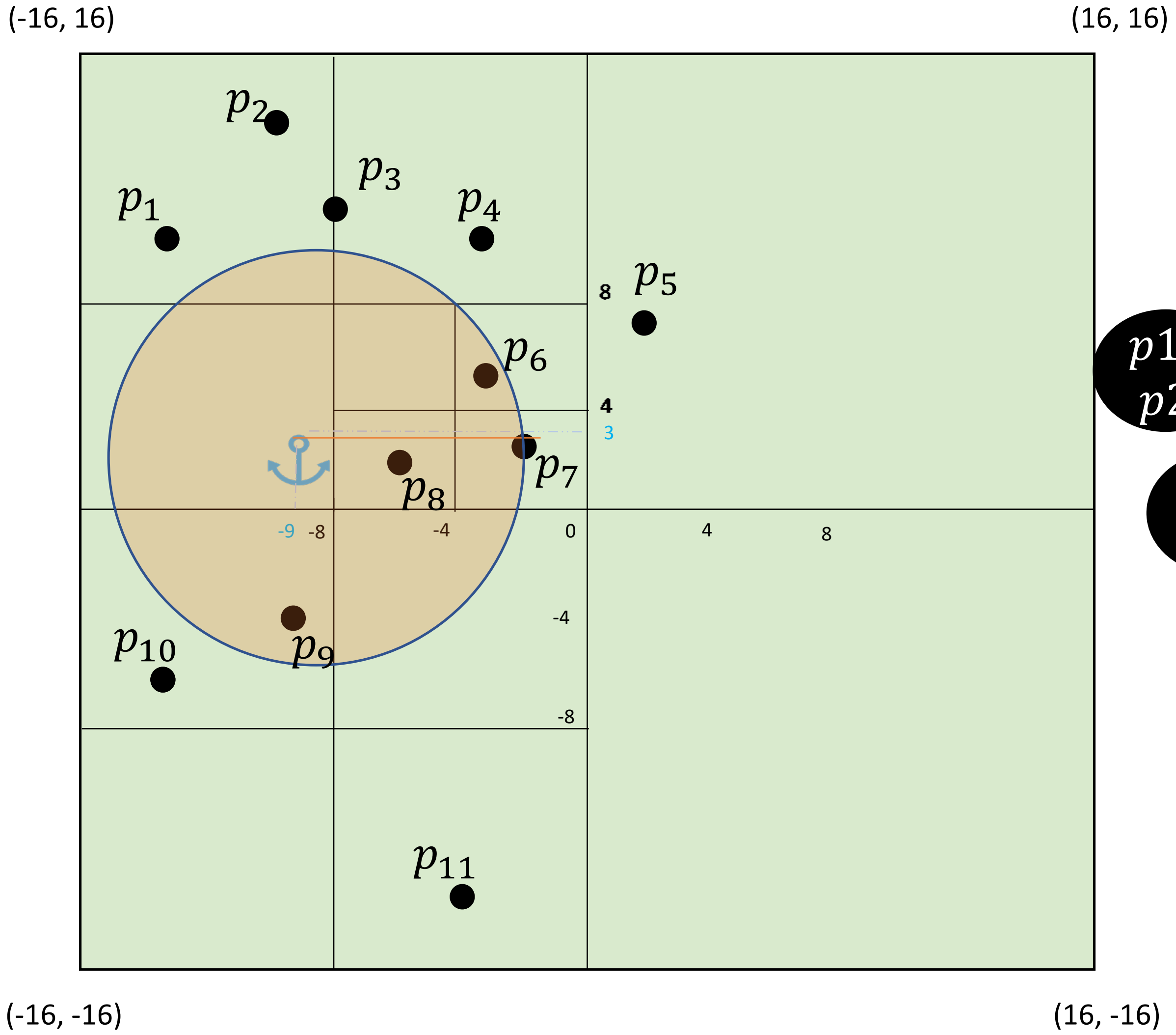
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

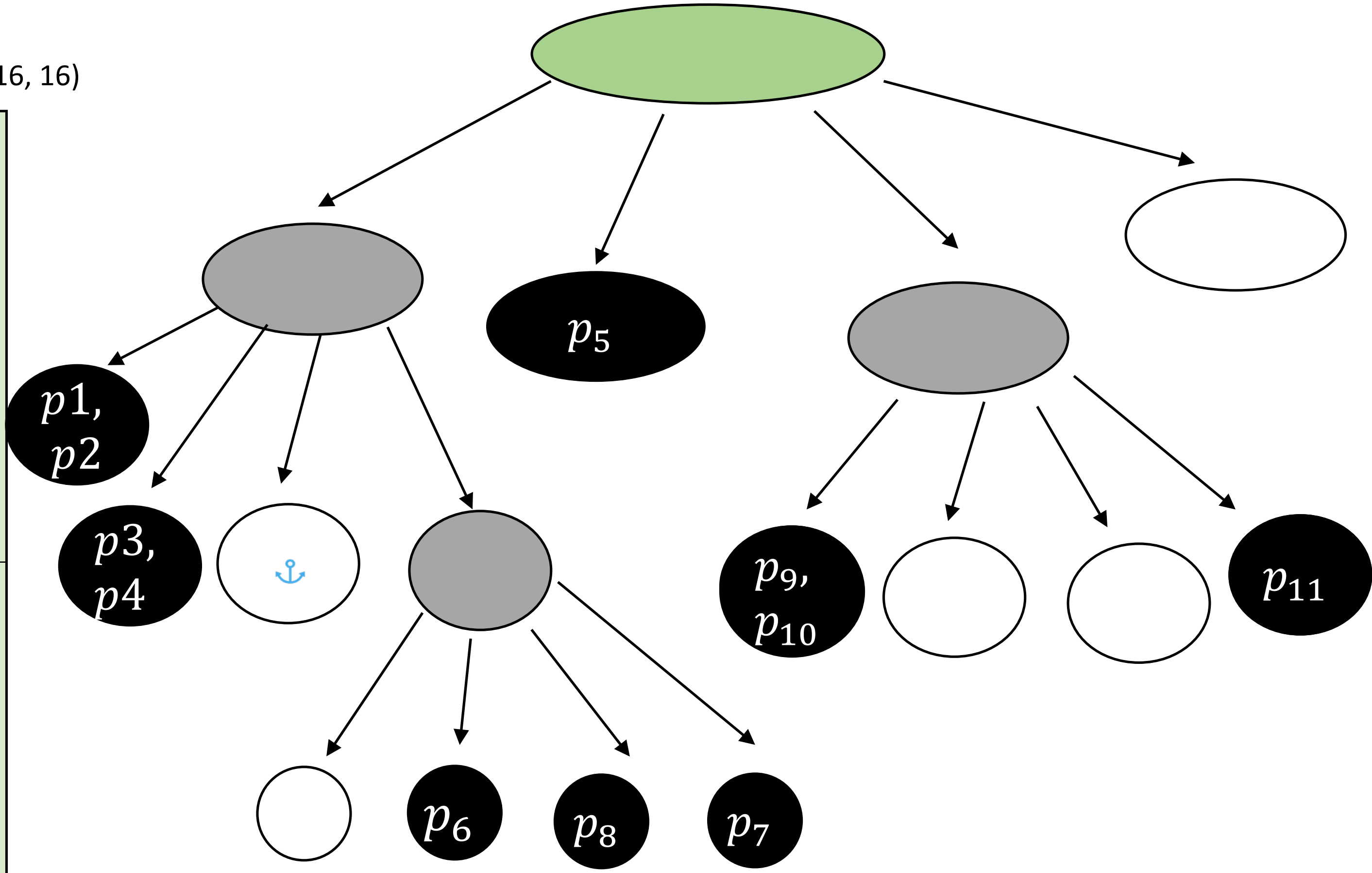




Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

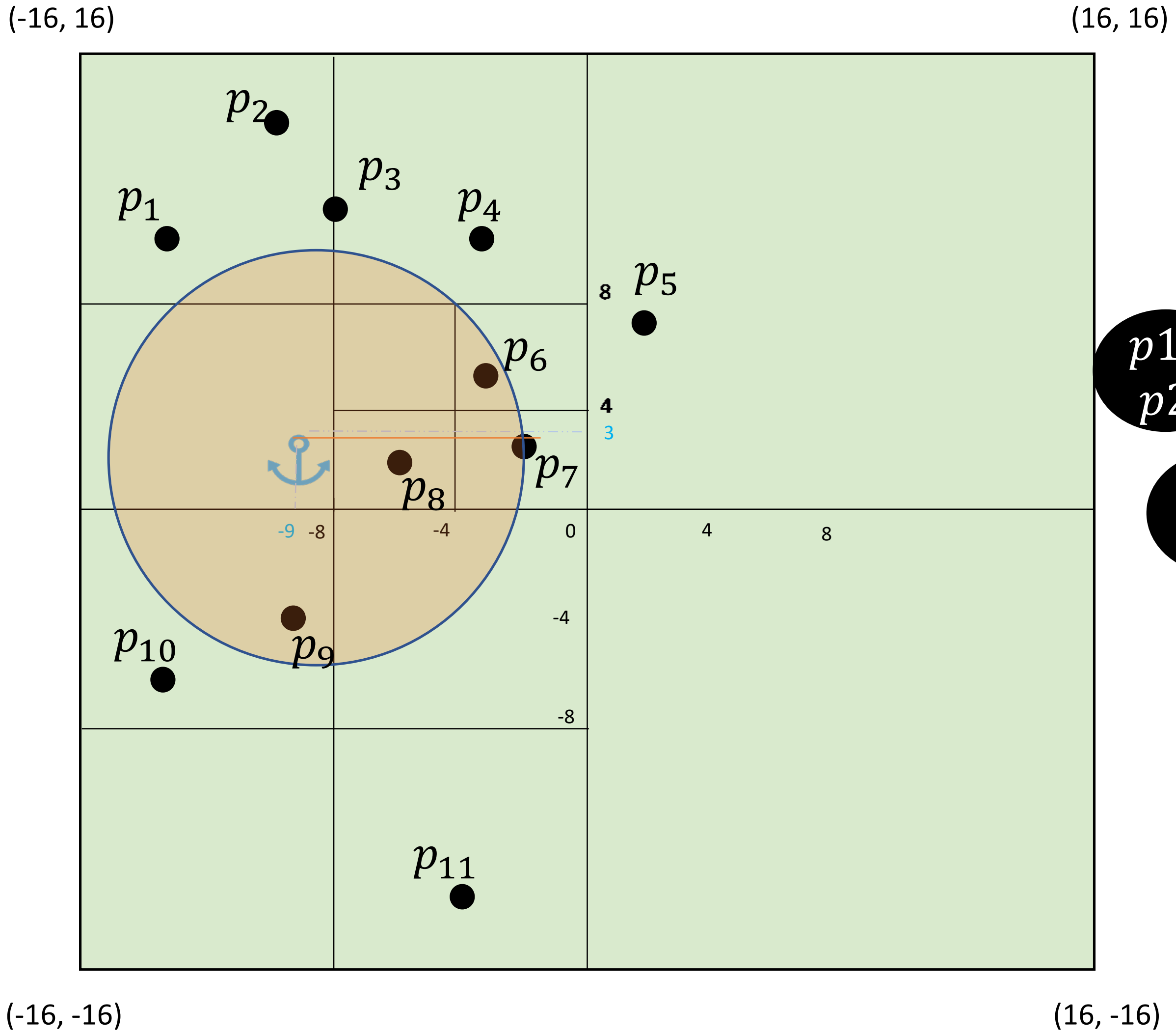


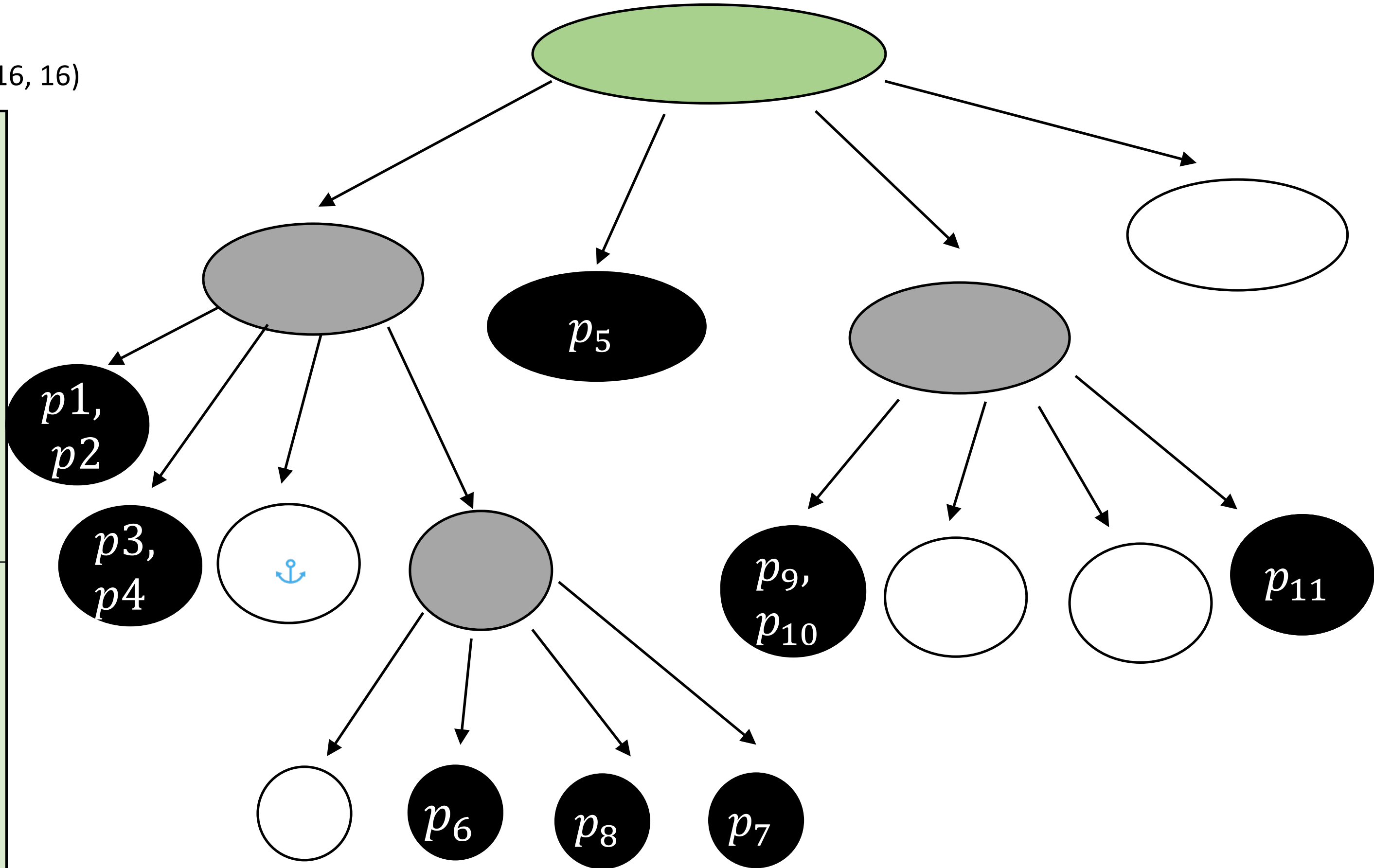


Visit root

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

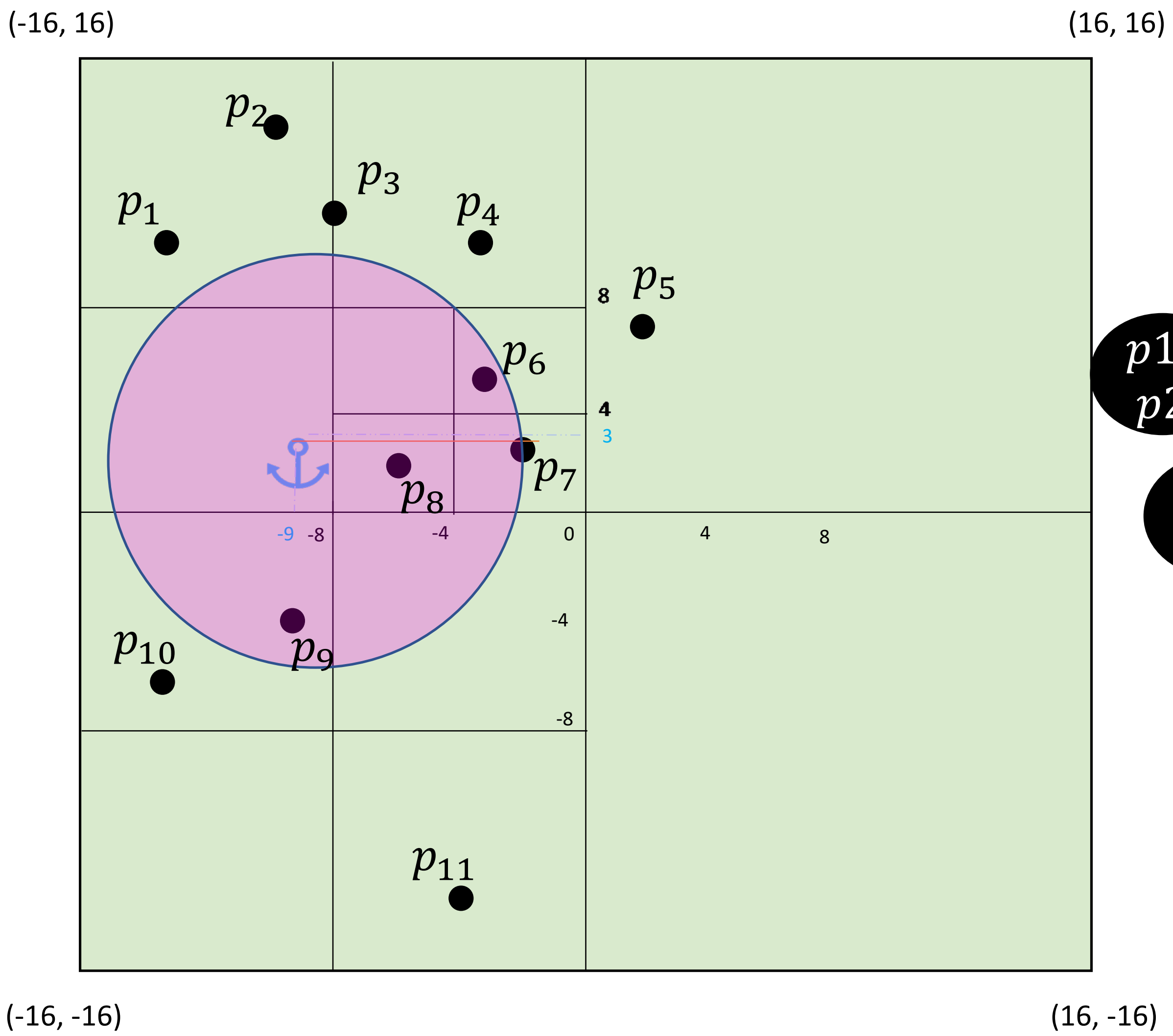


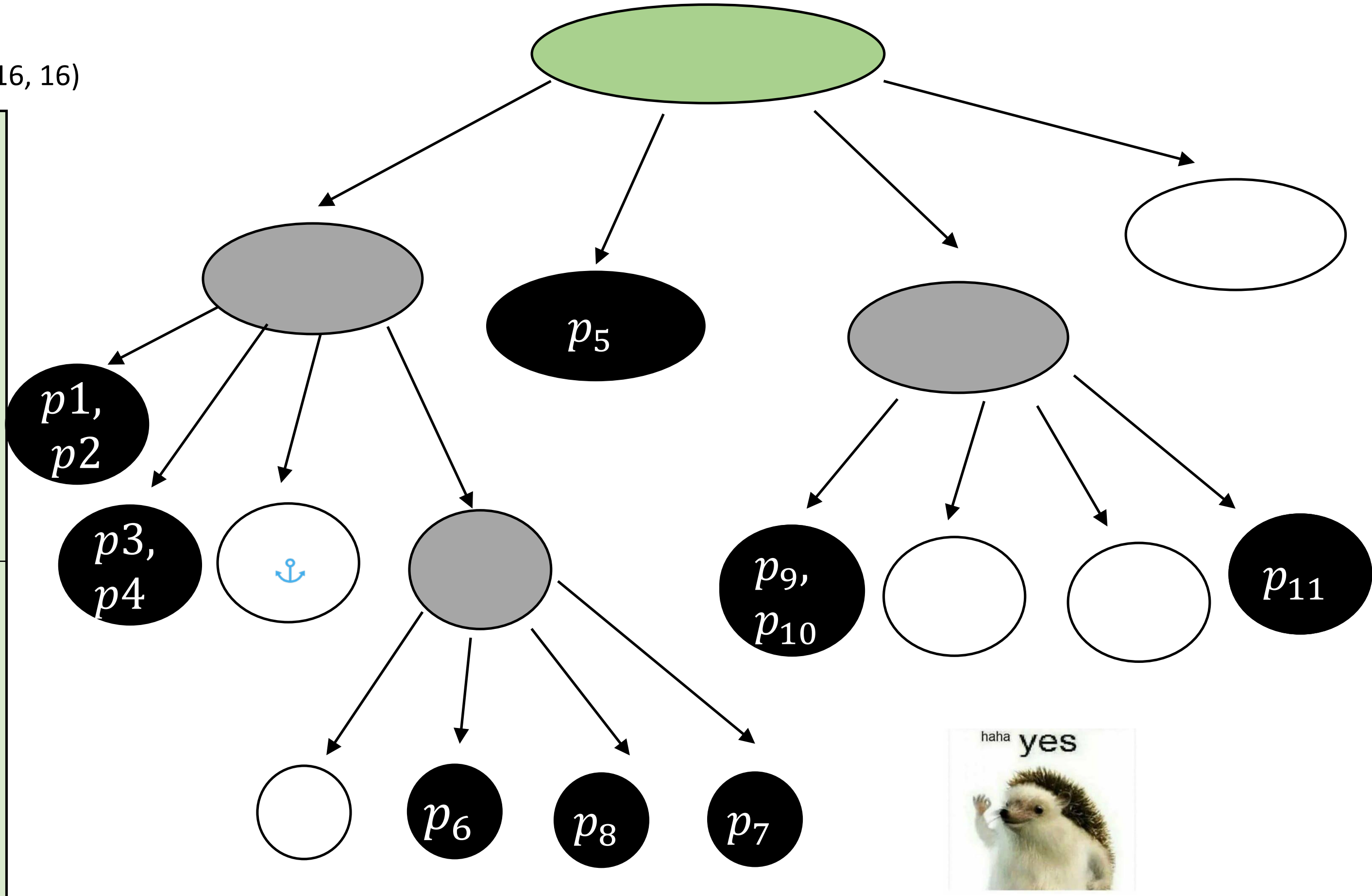


Is there a **non-zero area intersection** between the quadrant spanned by the root and the range?

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

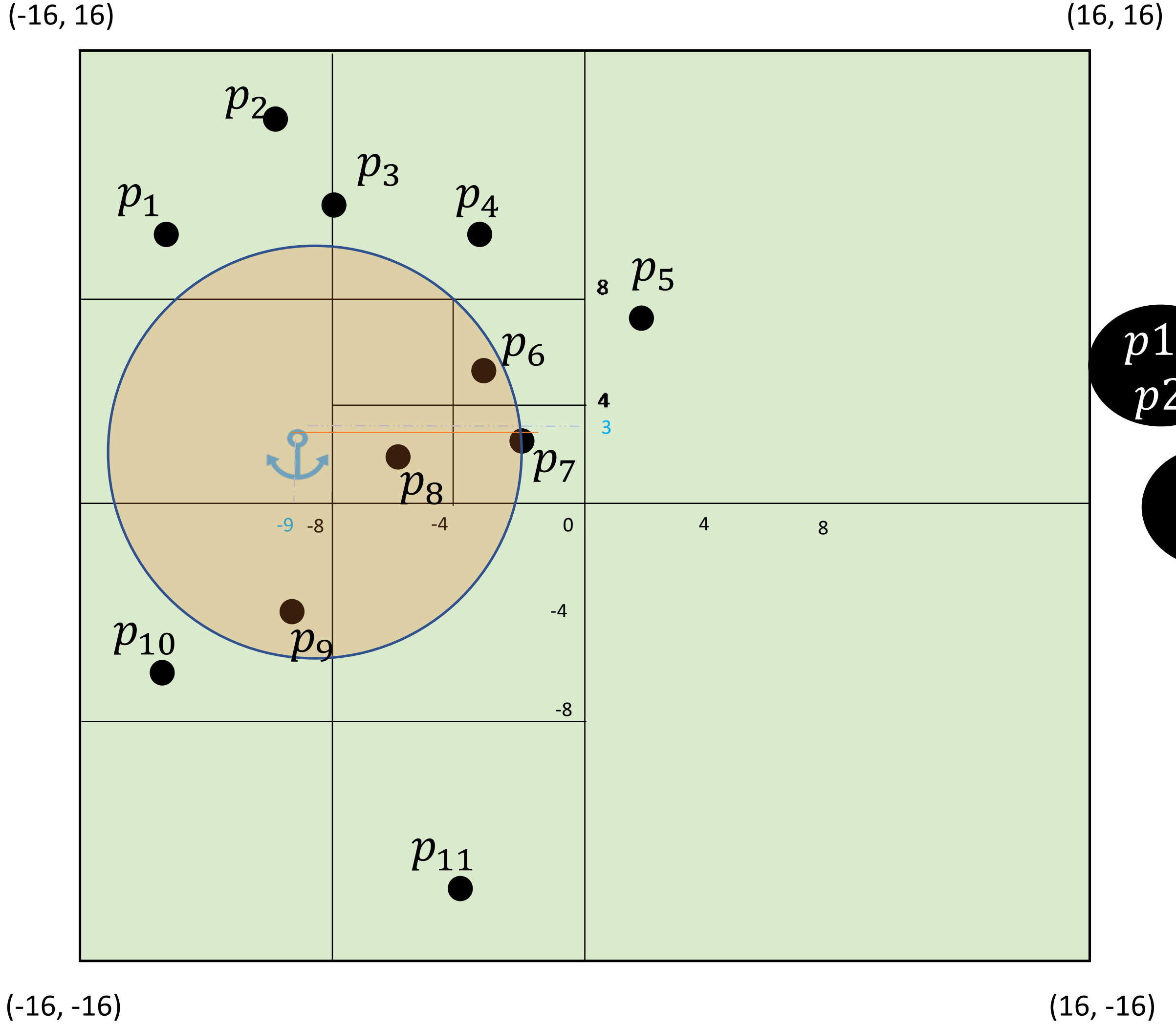


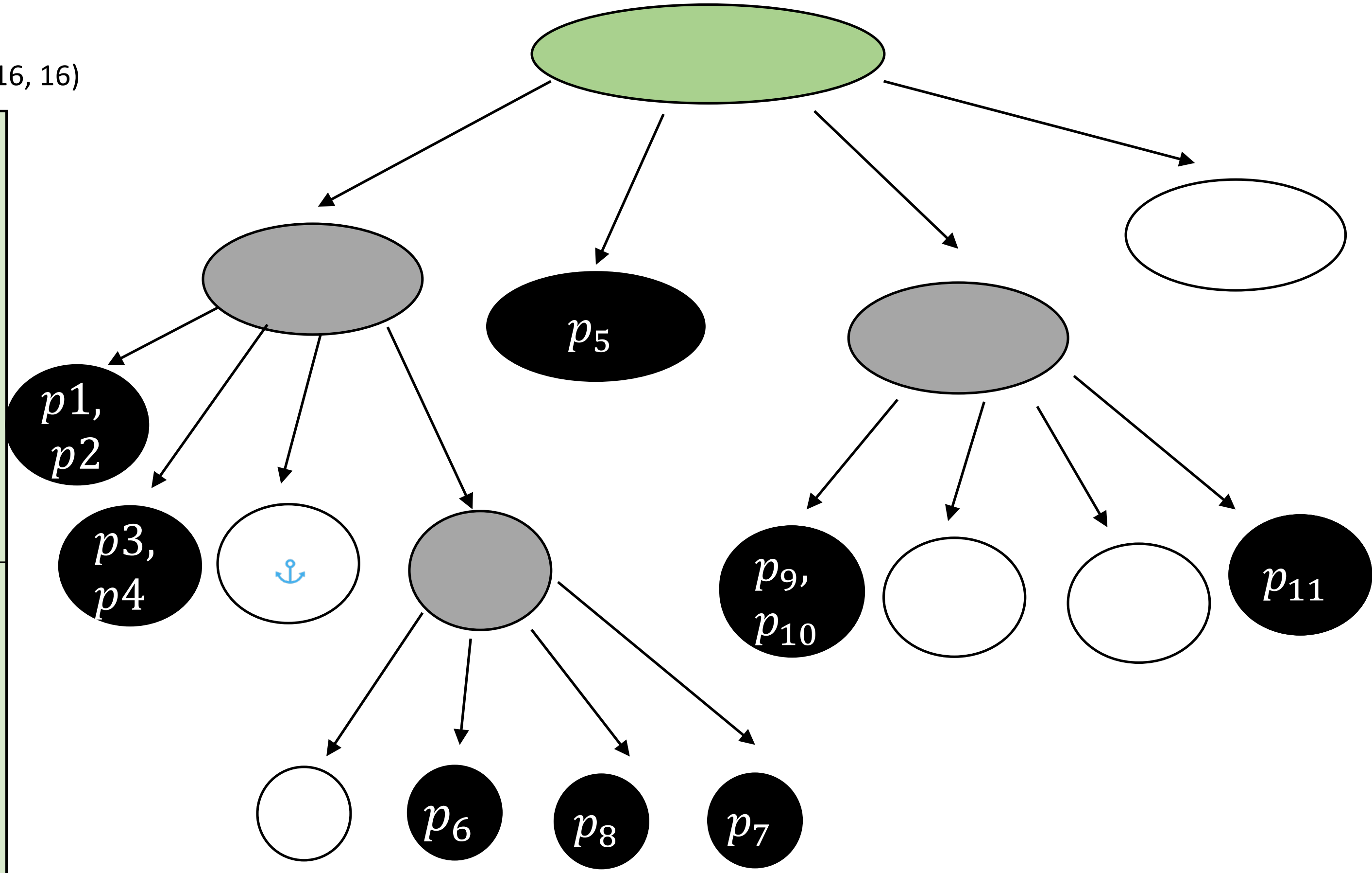


Is there a **non-zero area intersection** between the quadrant spanned by the root and the range? **YUP!**

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

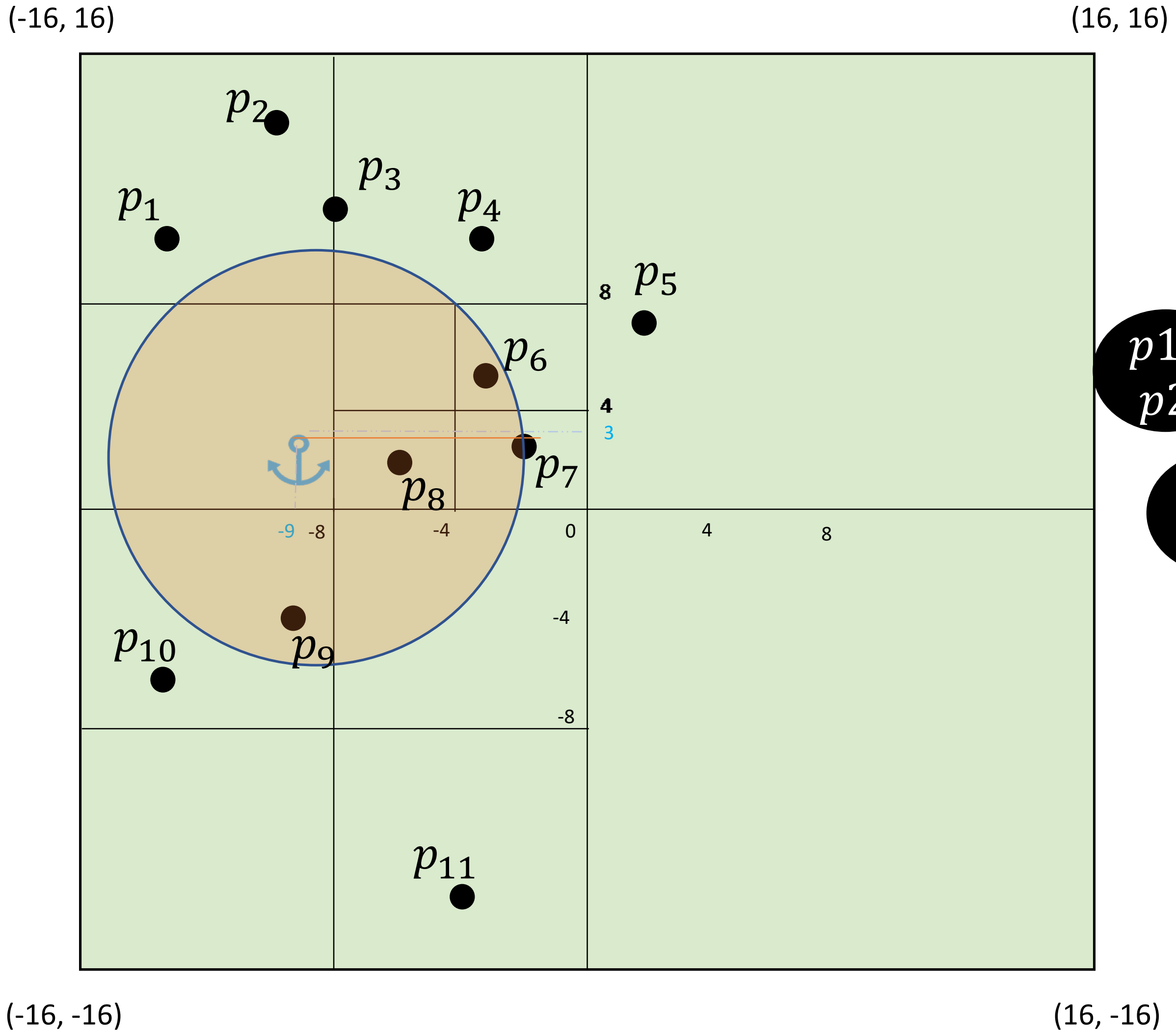


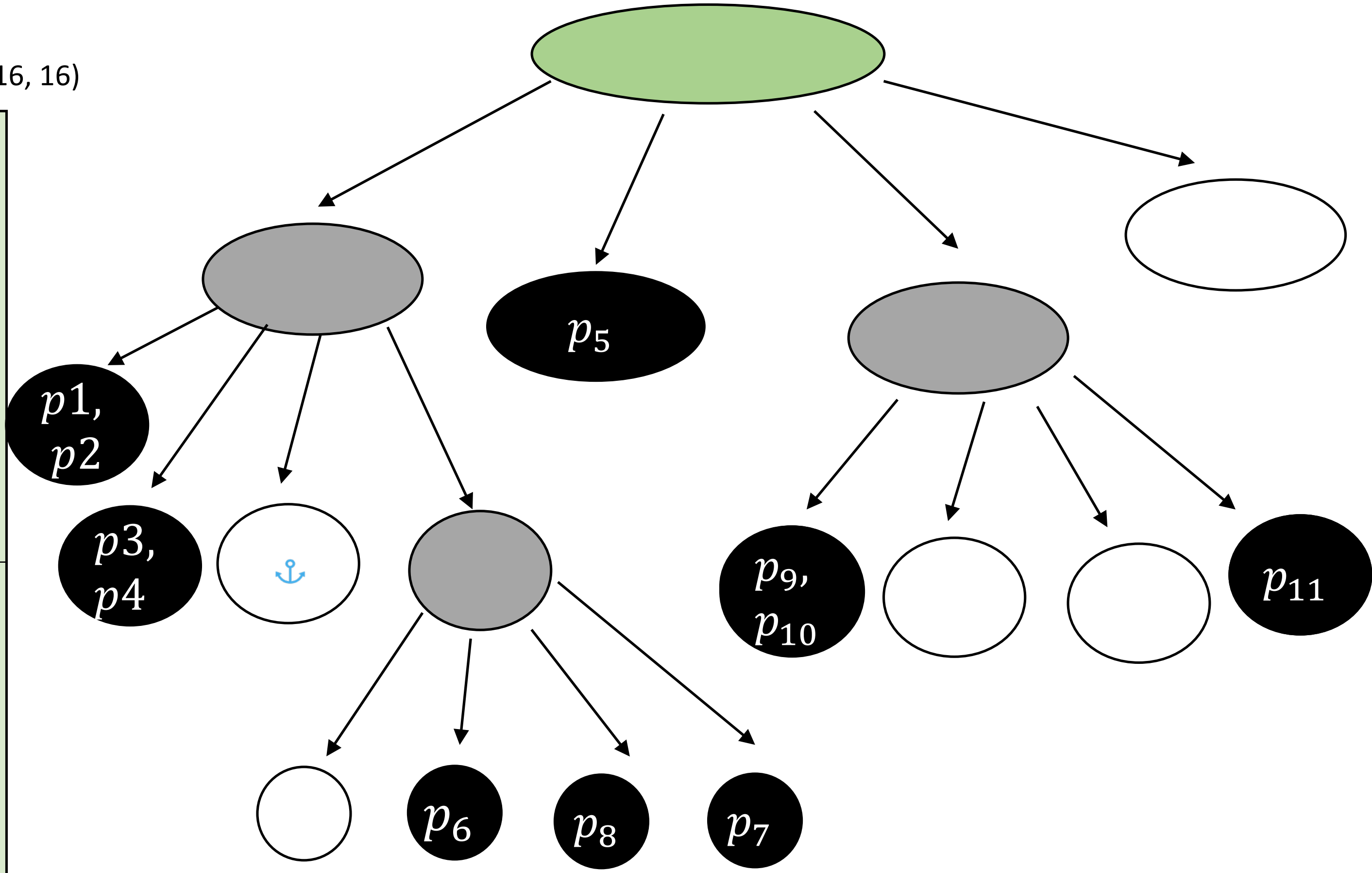


So it **makes sense to stay in the tree**, and we don't have to return.

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

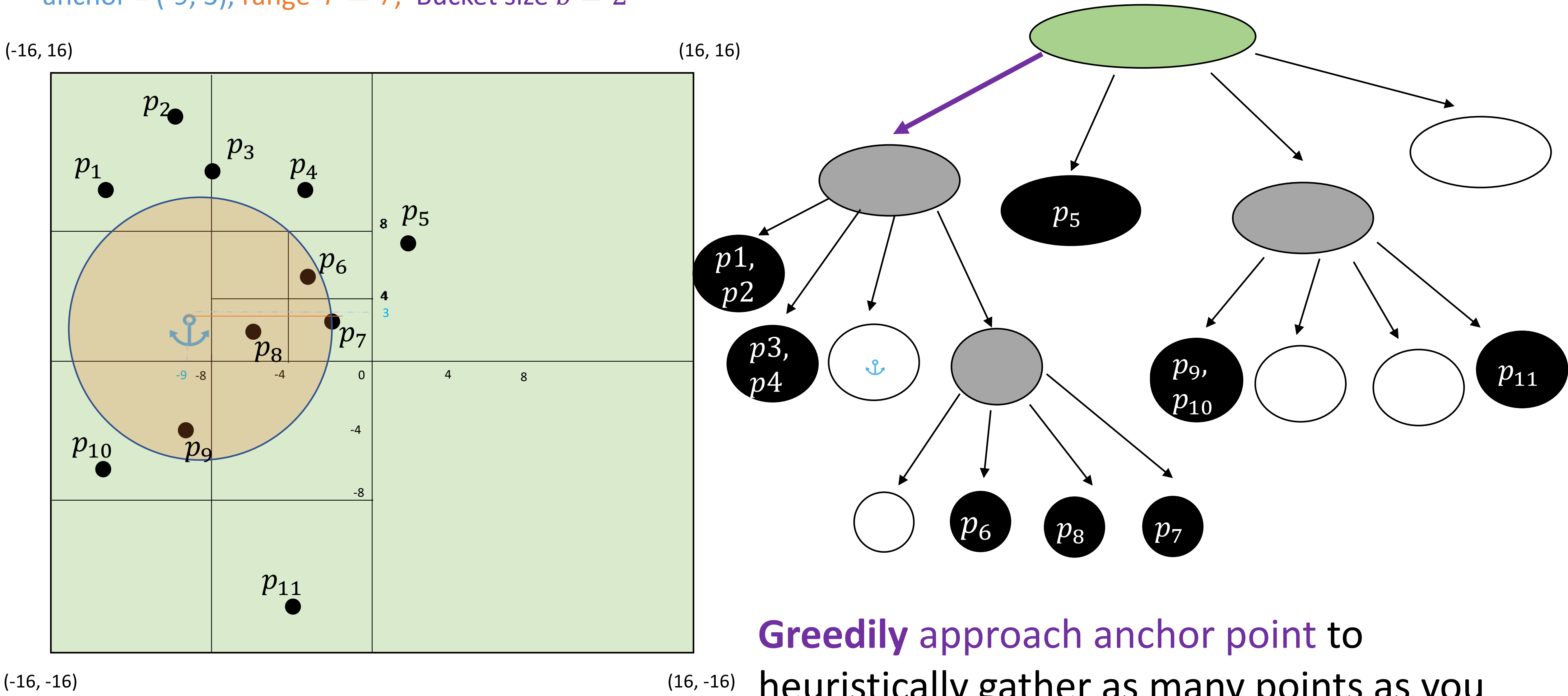




This analysis might seem **stupid** to do at the root of the tree, but is exactly the process that needs to happen in every subtree!

Example of range query in PR-QuadTree

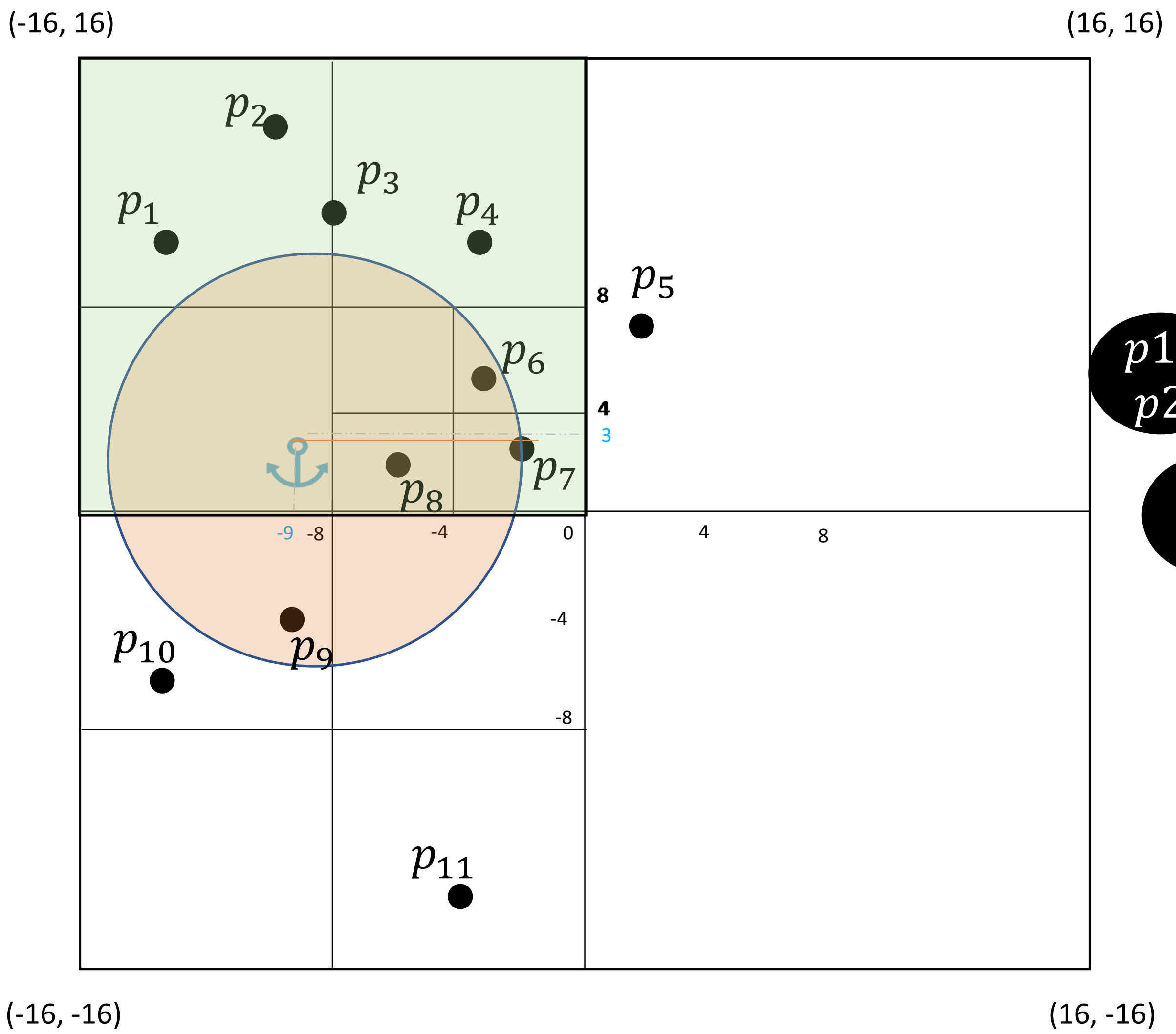
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

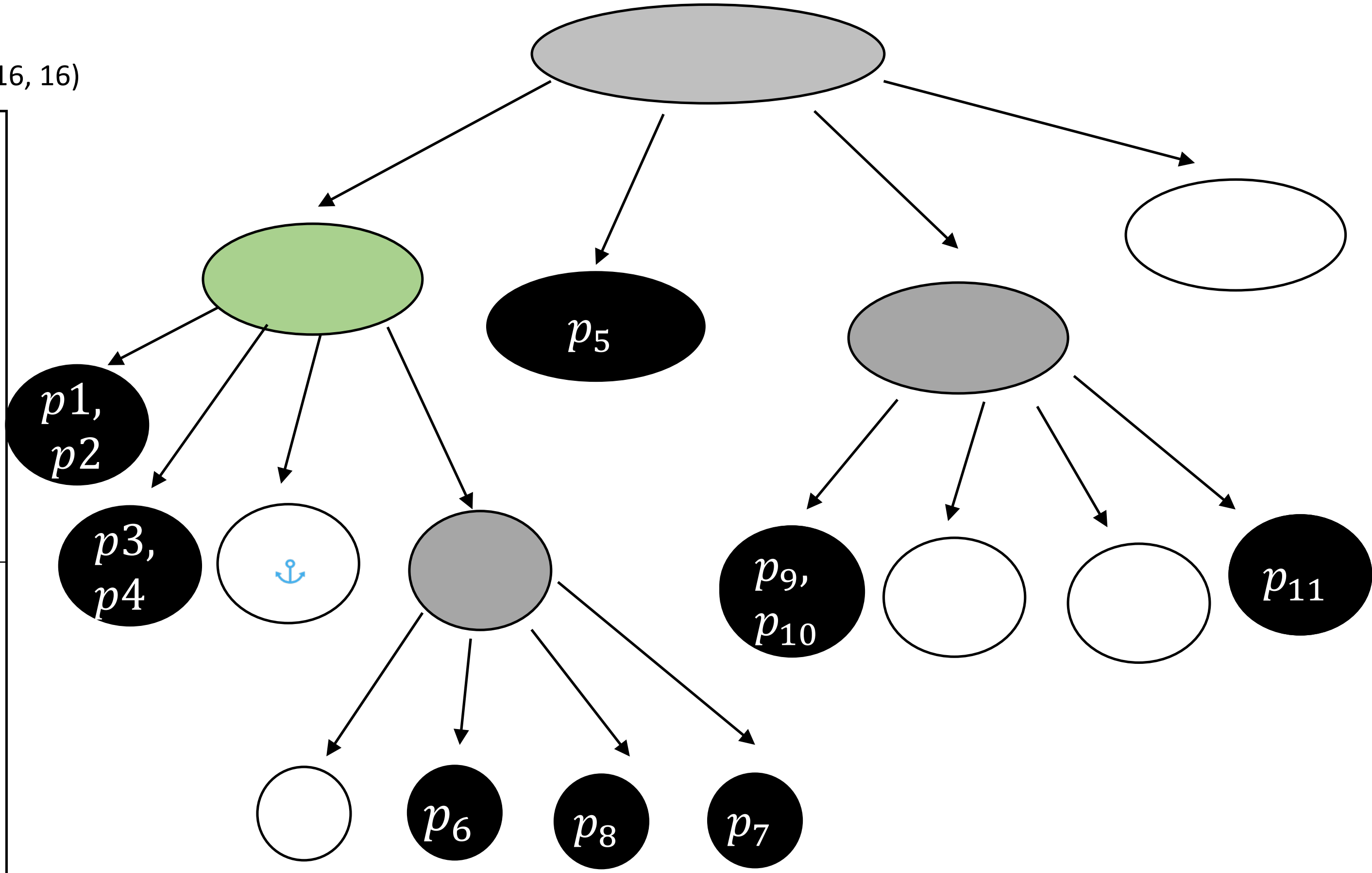


Greedily approach anchor point to heuristically gather as many points as you can fast. Visit first (NW) subtree.

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

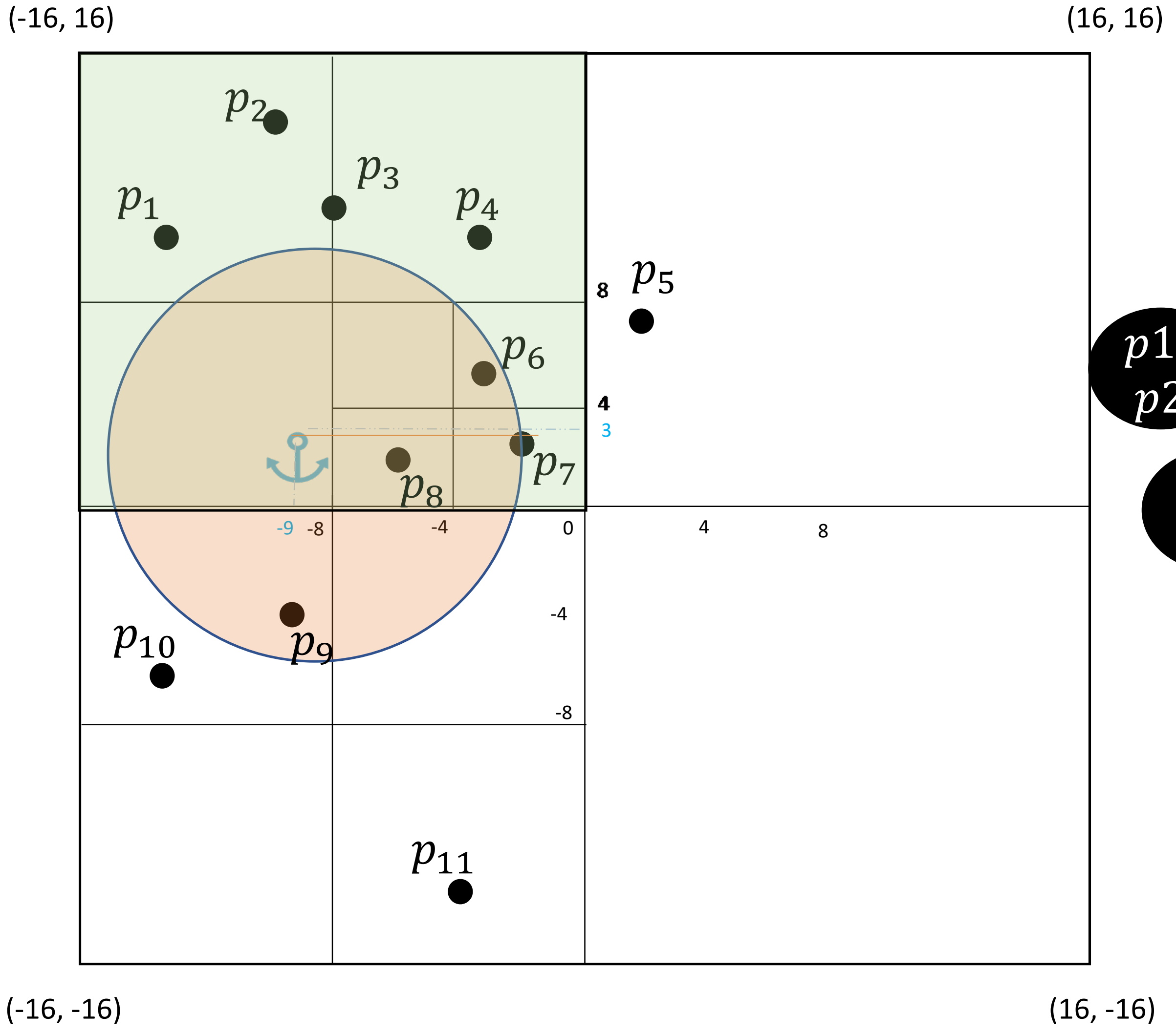


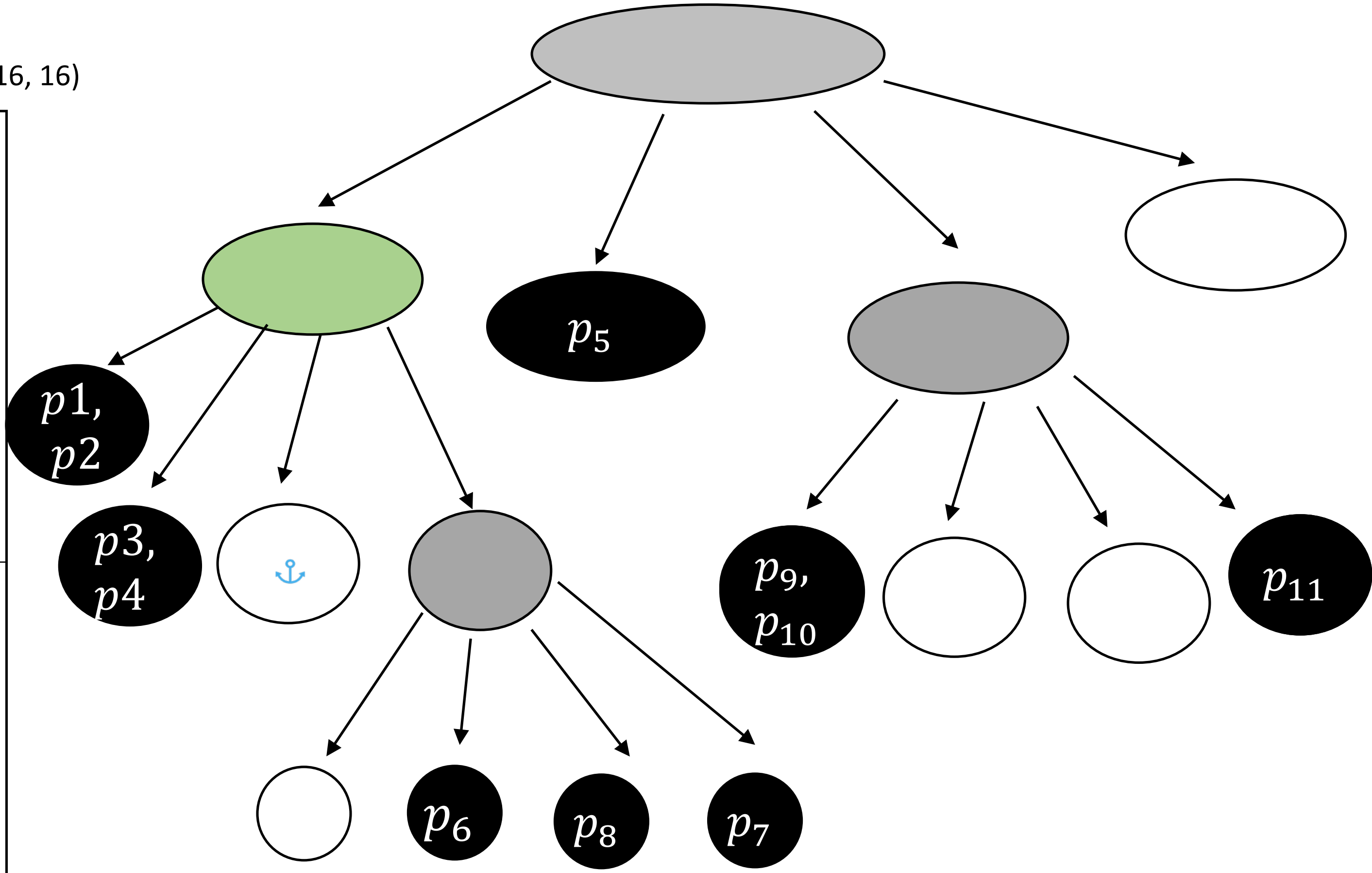


Greedily approach anchor point to heuristically gather as many points as you can fast. Visit first (NW) subtree.

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

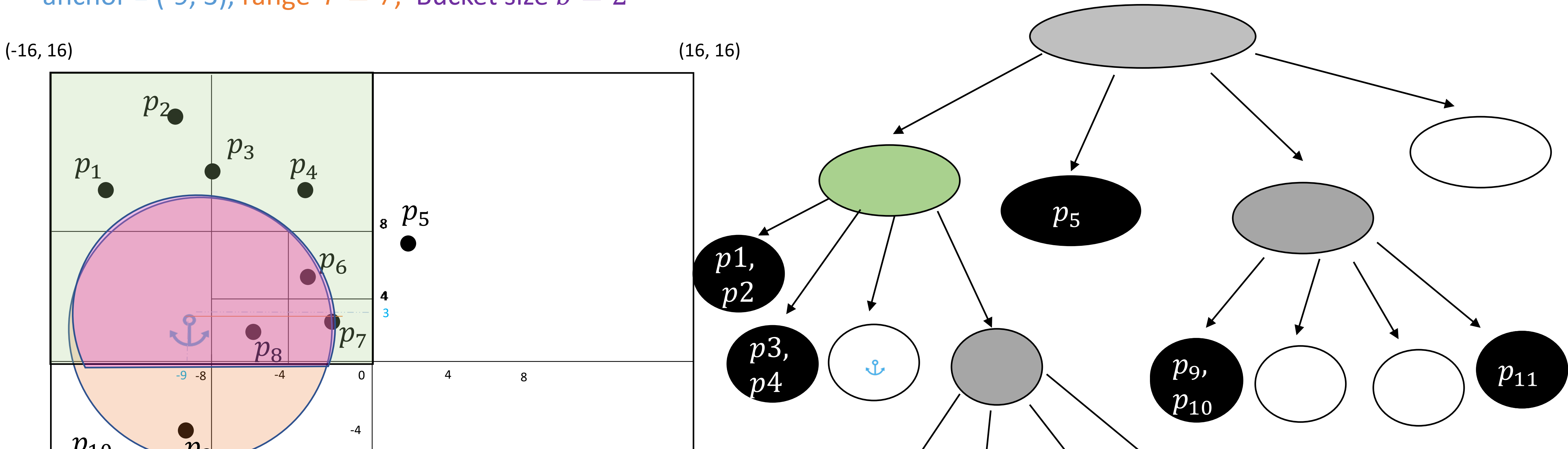




Is there a **non-zero intersection** between range and subtree?

Example of range query in PR-QuadTree

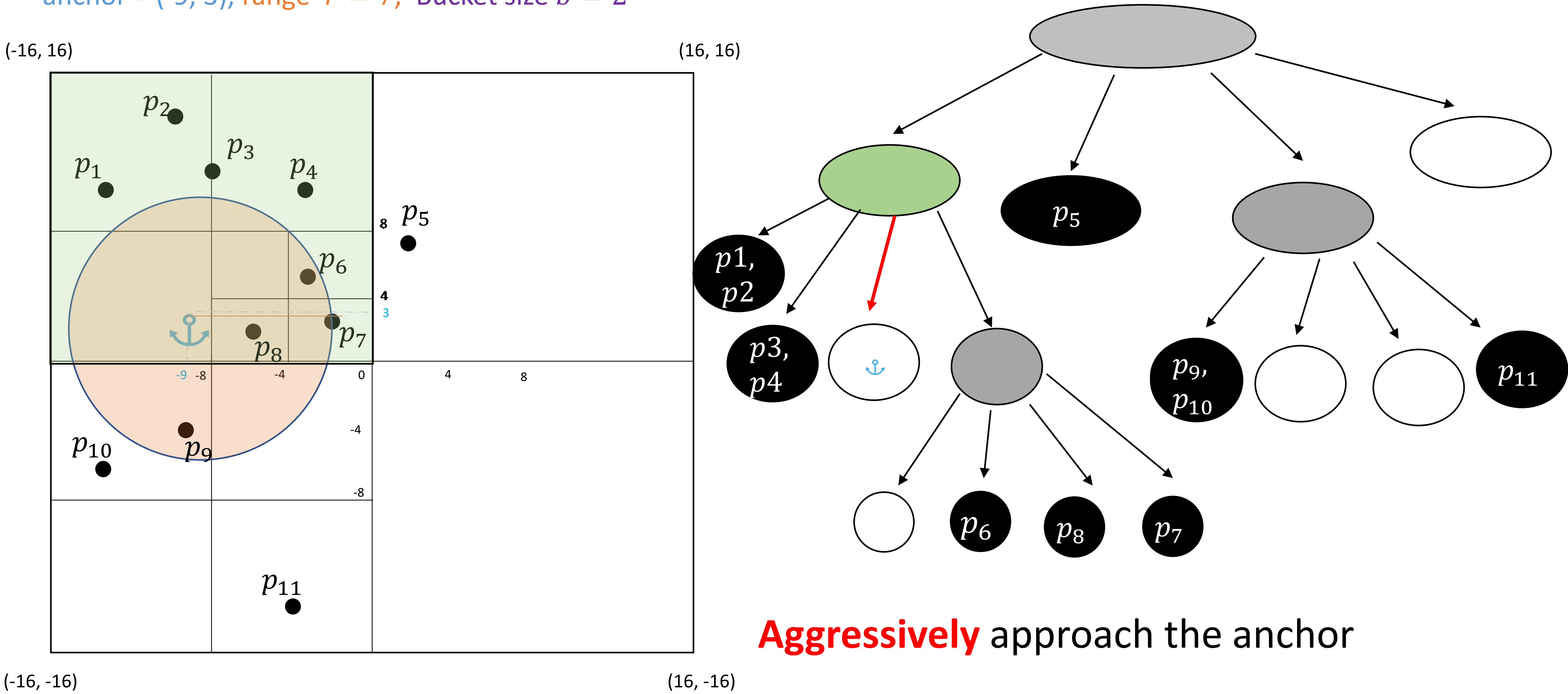
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$




Is there a **non-zero intersection** between range and subtree?

Example of range query in PR-QuadTree

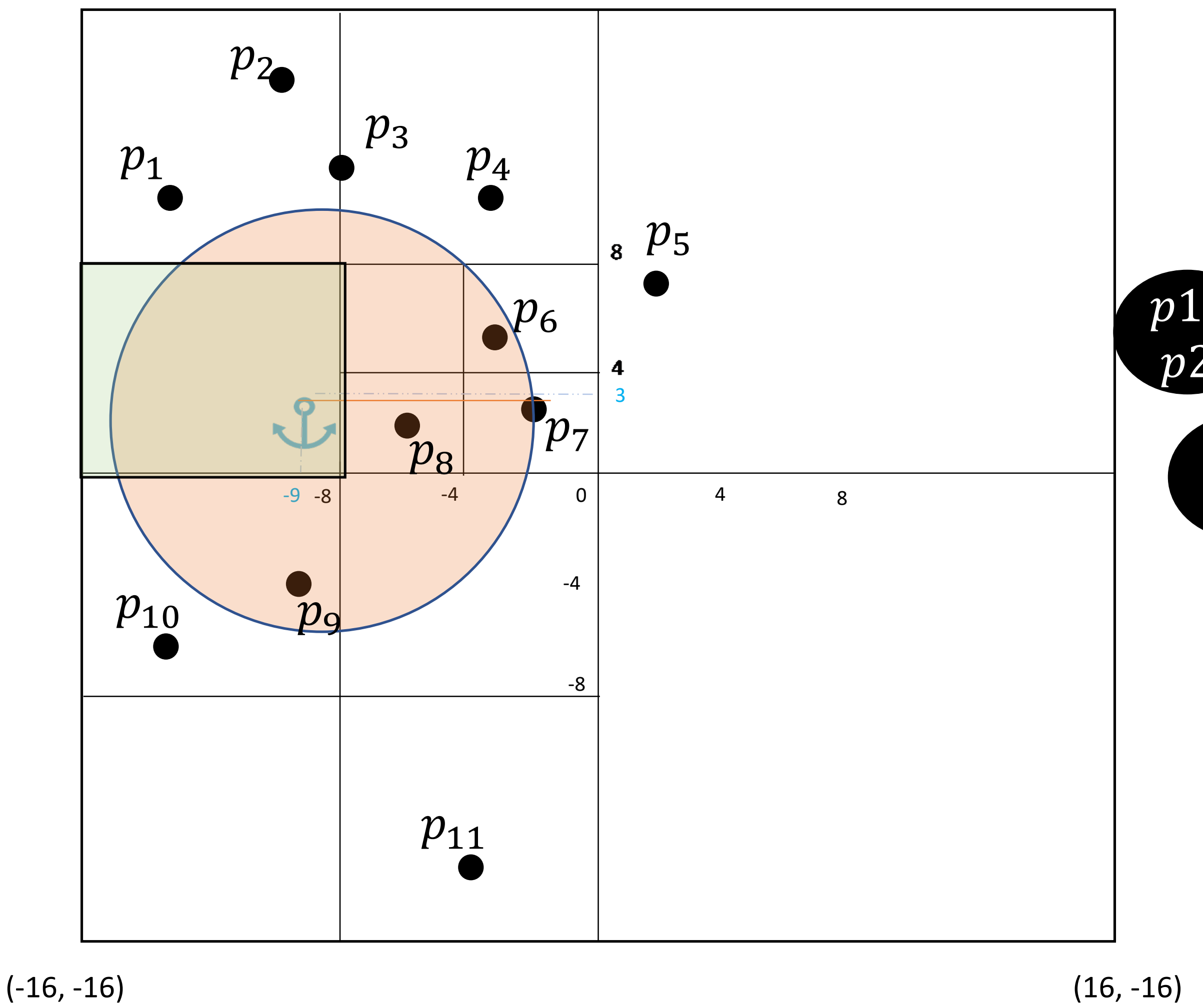
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

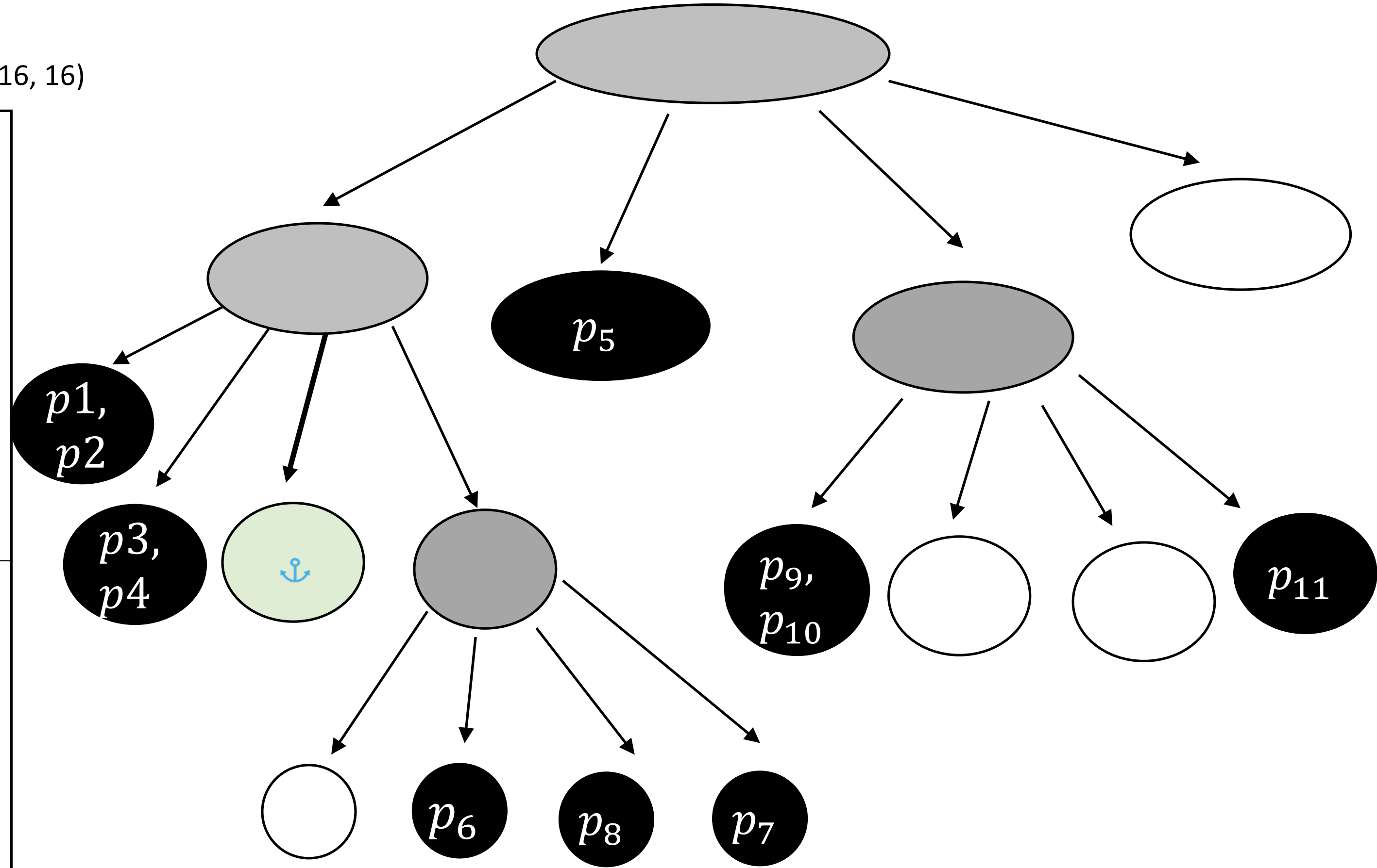


Aggressively approach the anchor

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

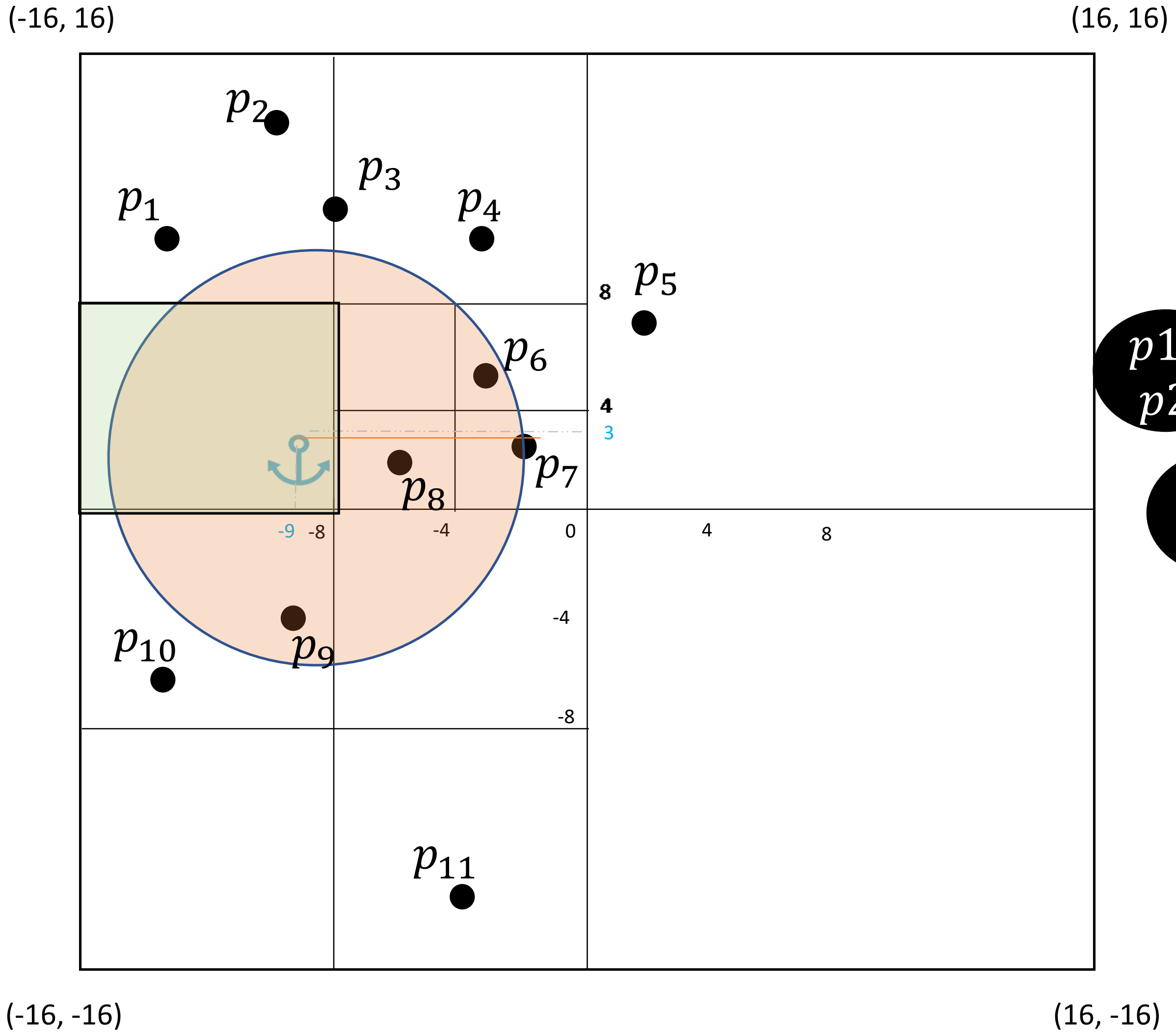


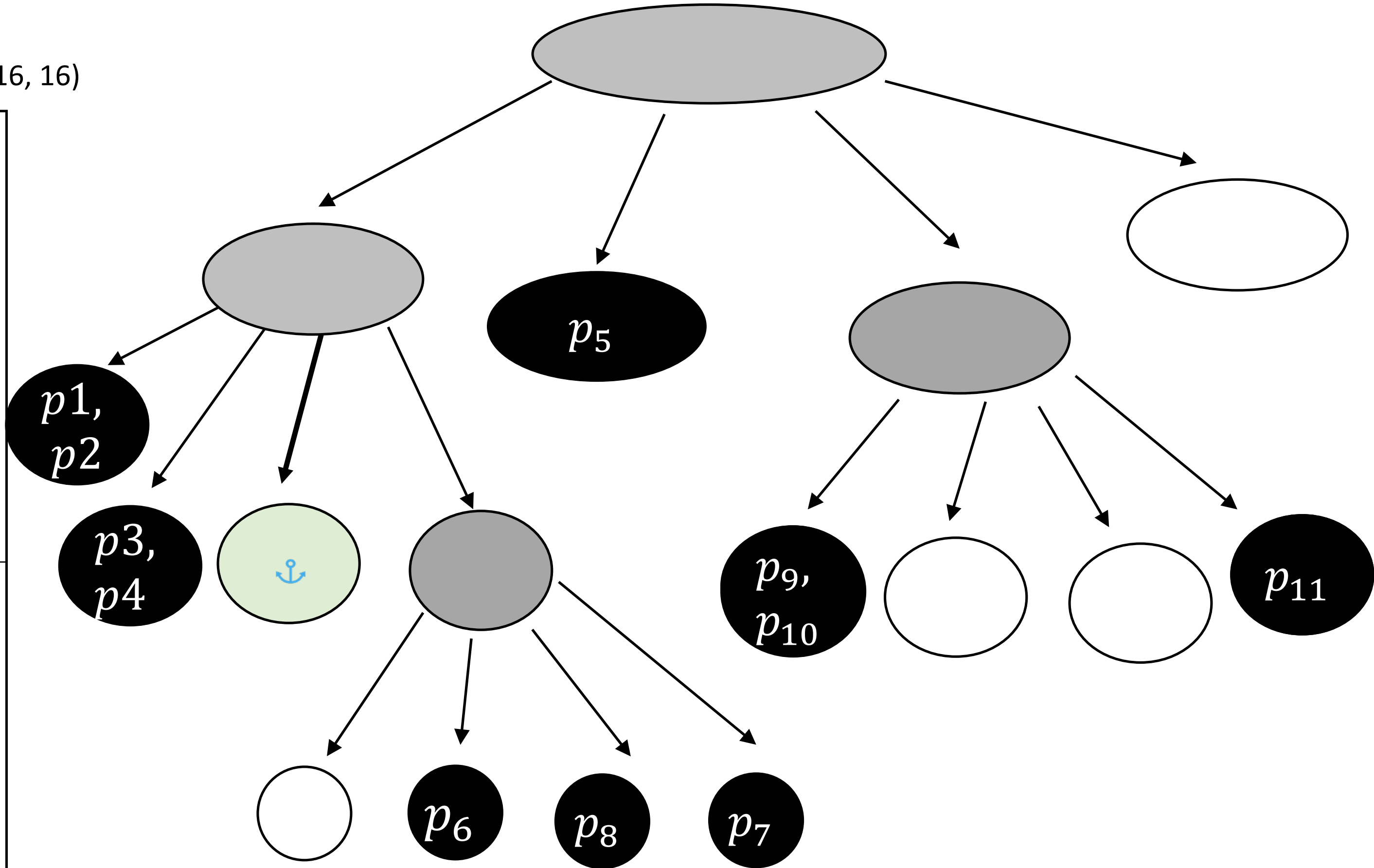


Aggressively approach the anchor.

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

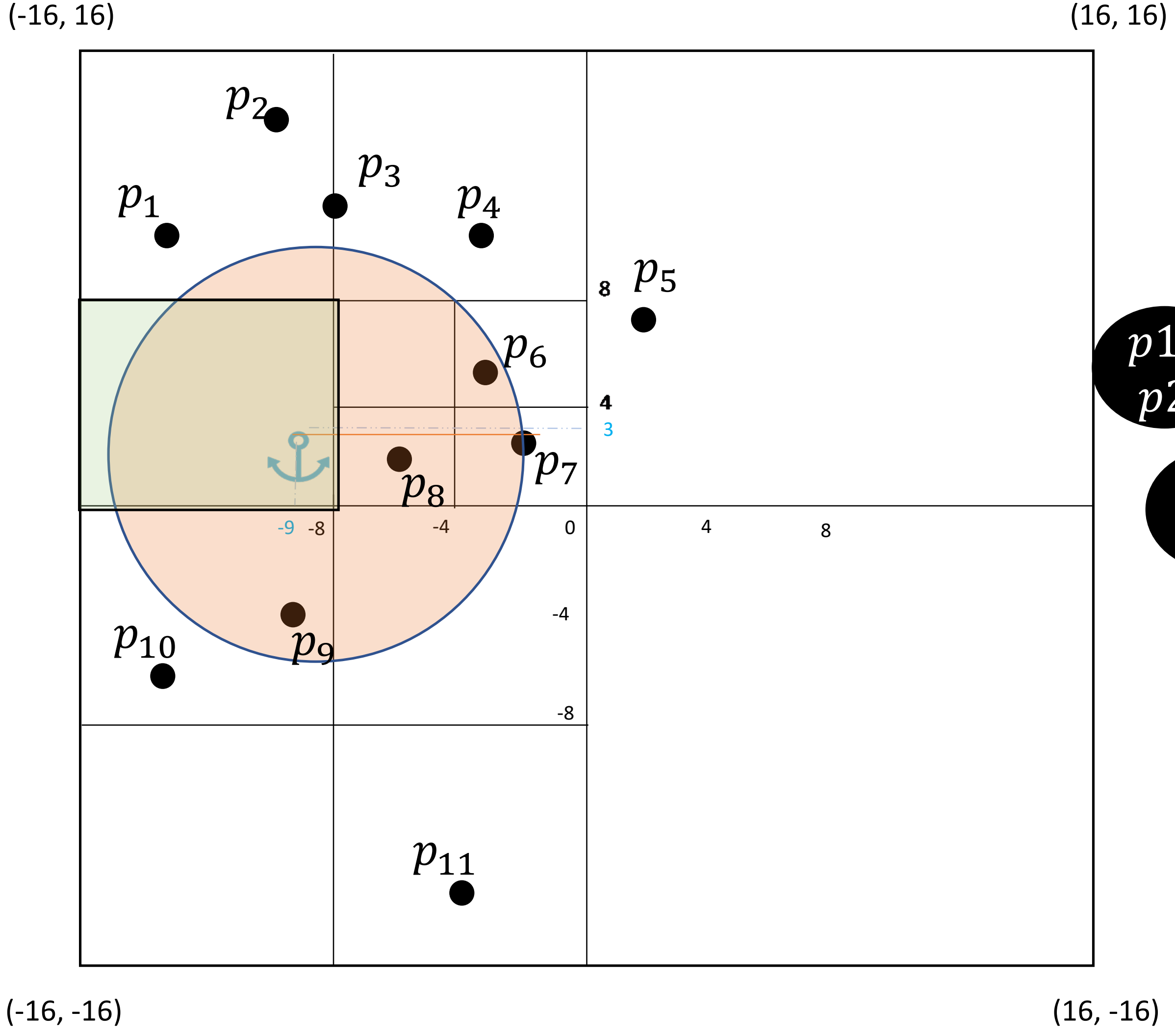


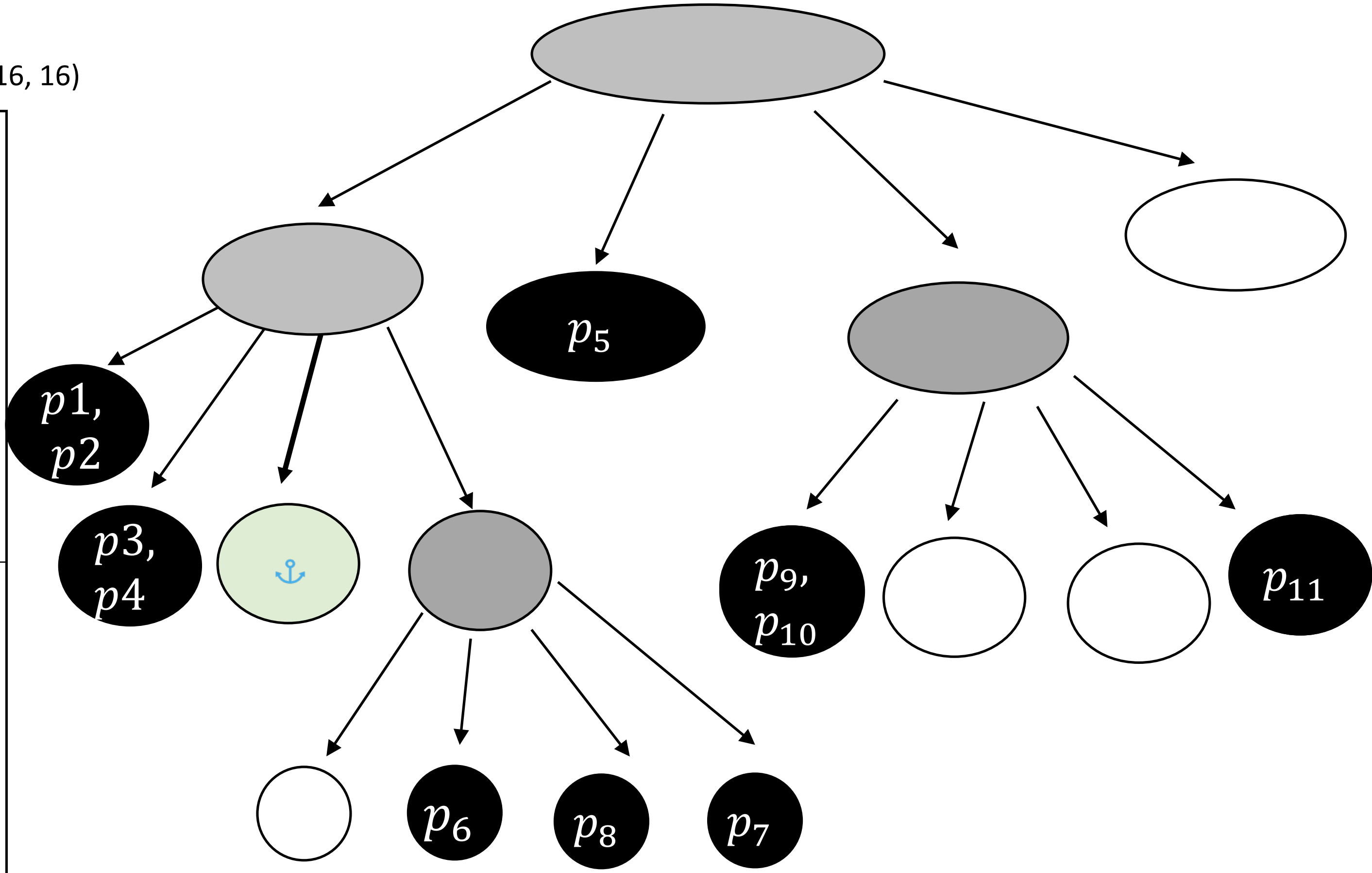


While there **exists** a non-zero intersection, the current node is also a white node, so he can't quite contribute to the solution!

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

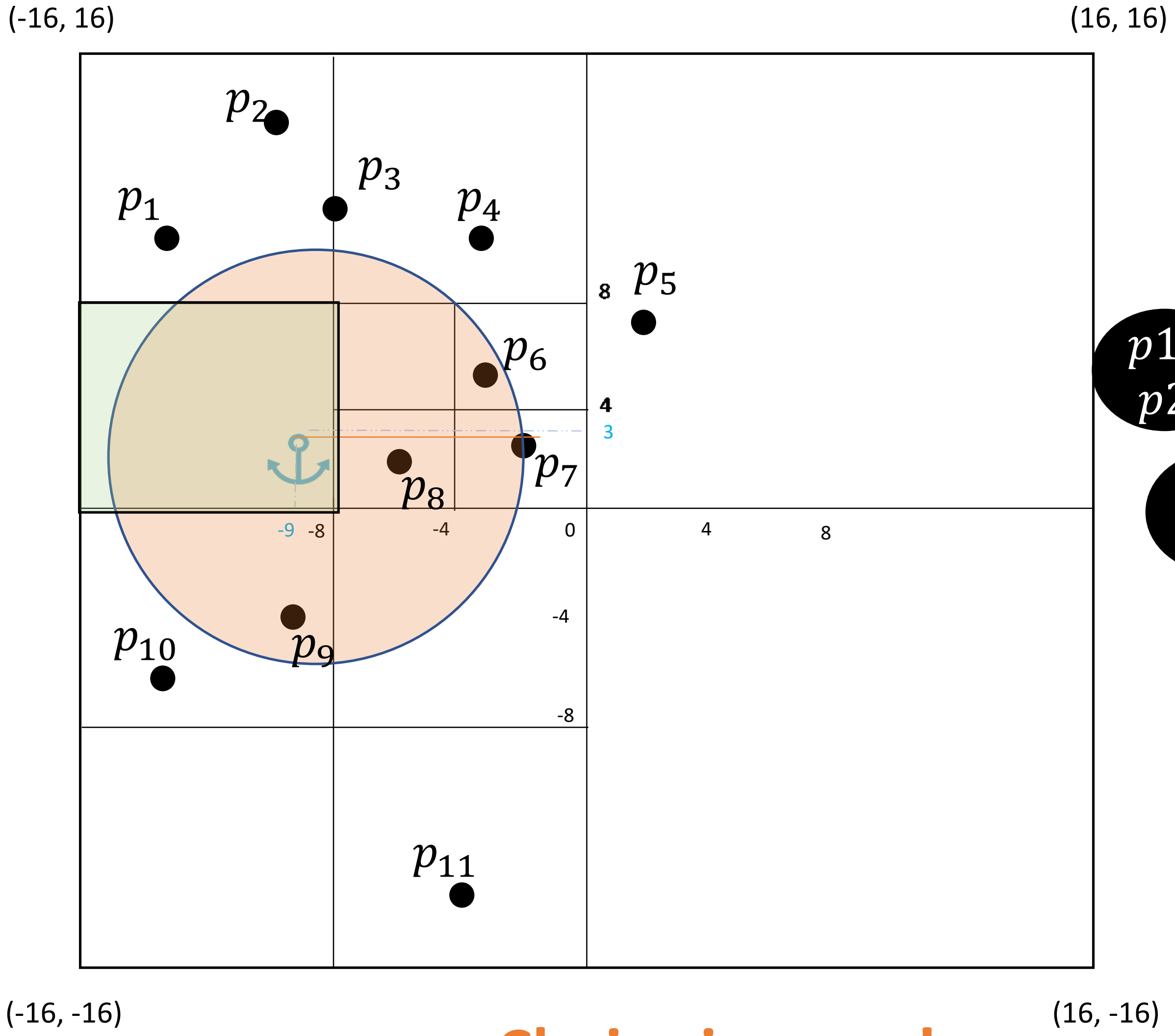


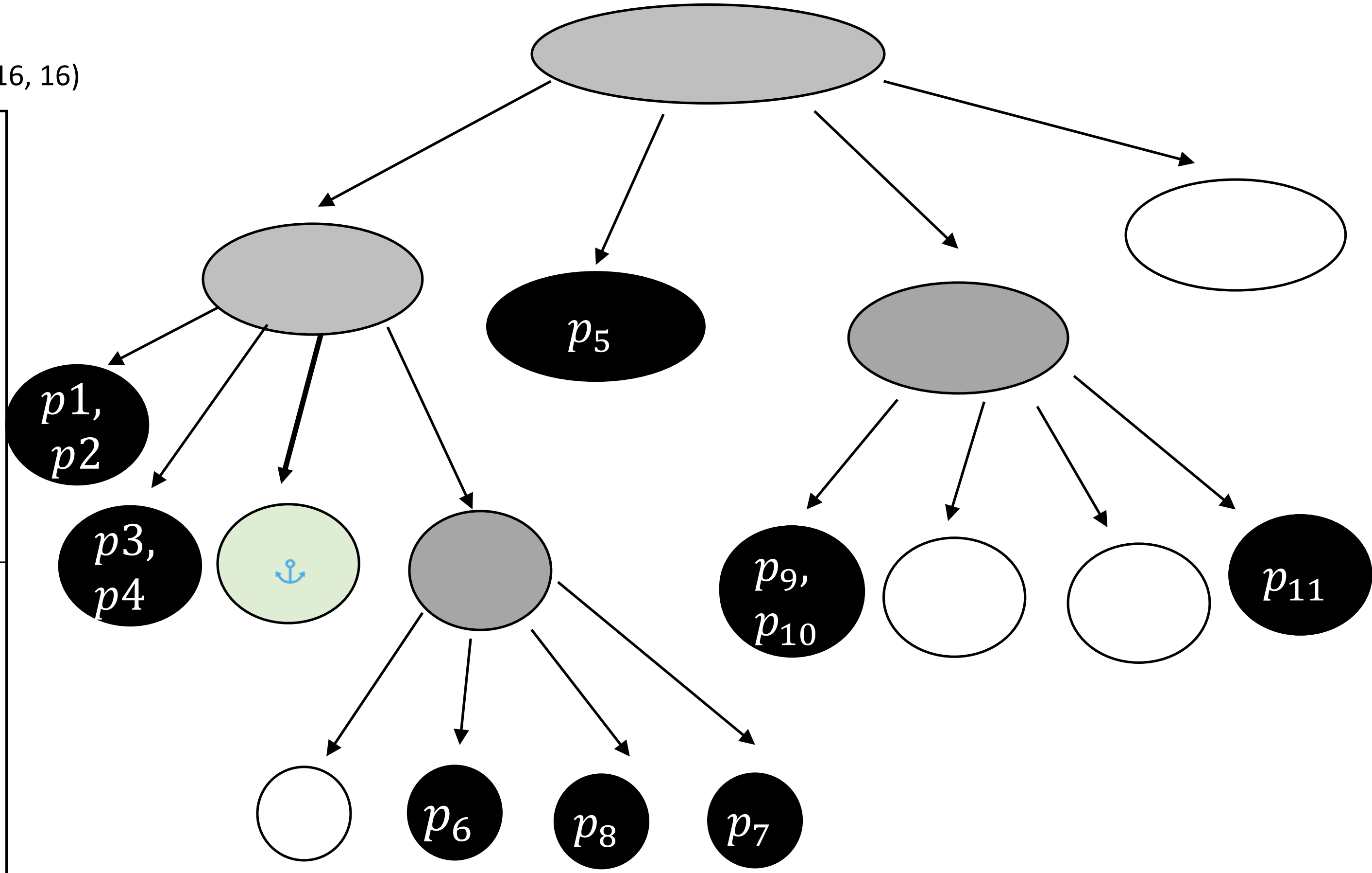


Implementation detail: do I actually **visit** a white node first? Do I check for intersection first to have a common implementation?

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



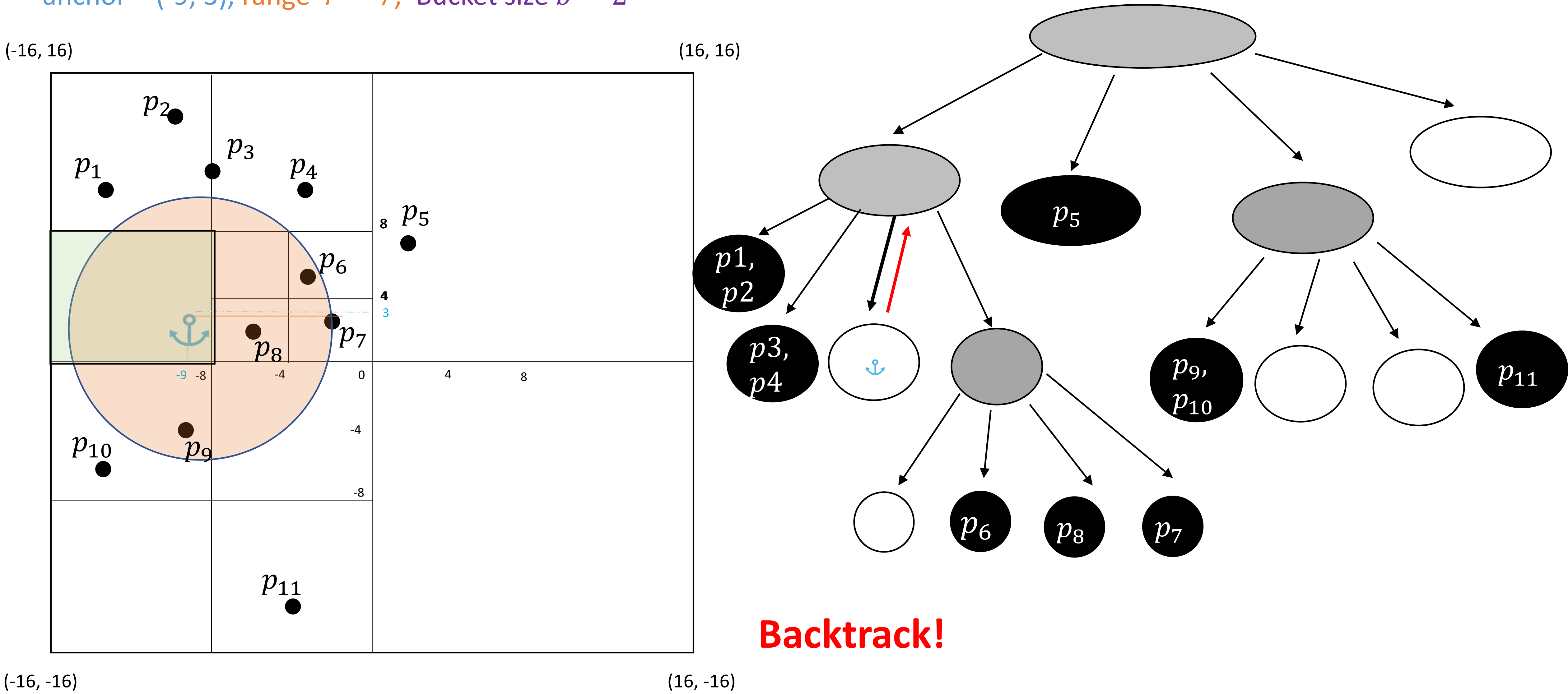


Choice is yours!

Implementation detail: do I actually **visit** a white node first? Do I check for intersection first to have a common implementation?

Example of range query in PR-QuadTree

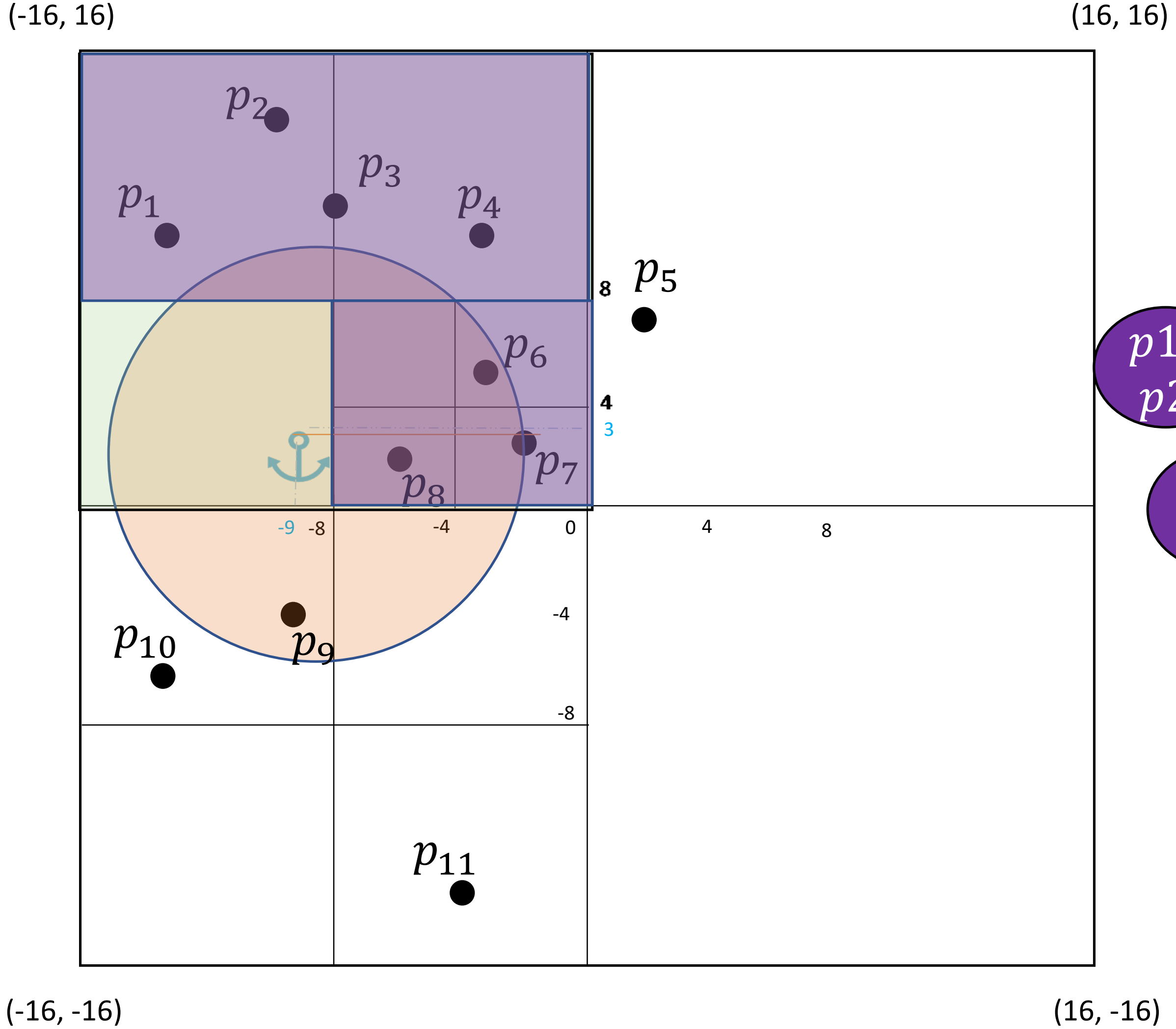
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

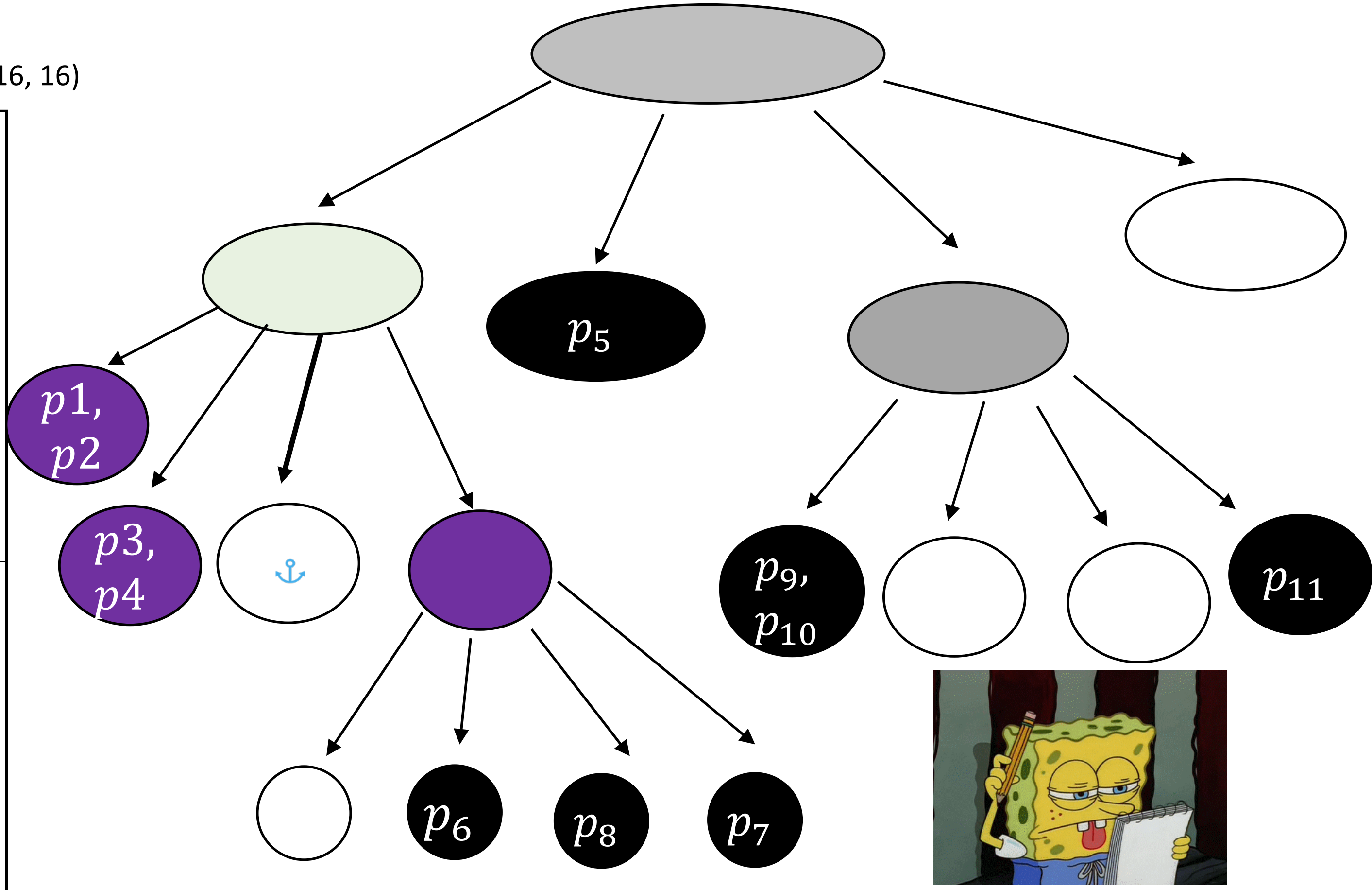


Backtrack!

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

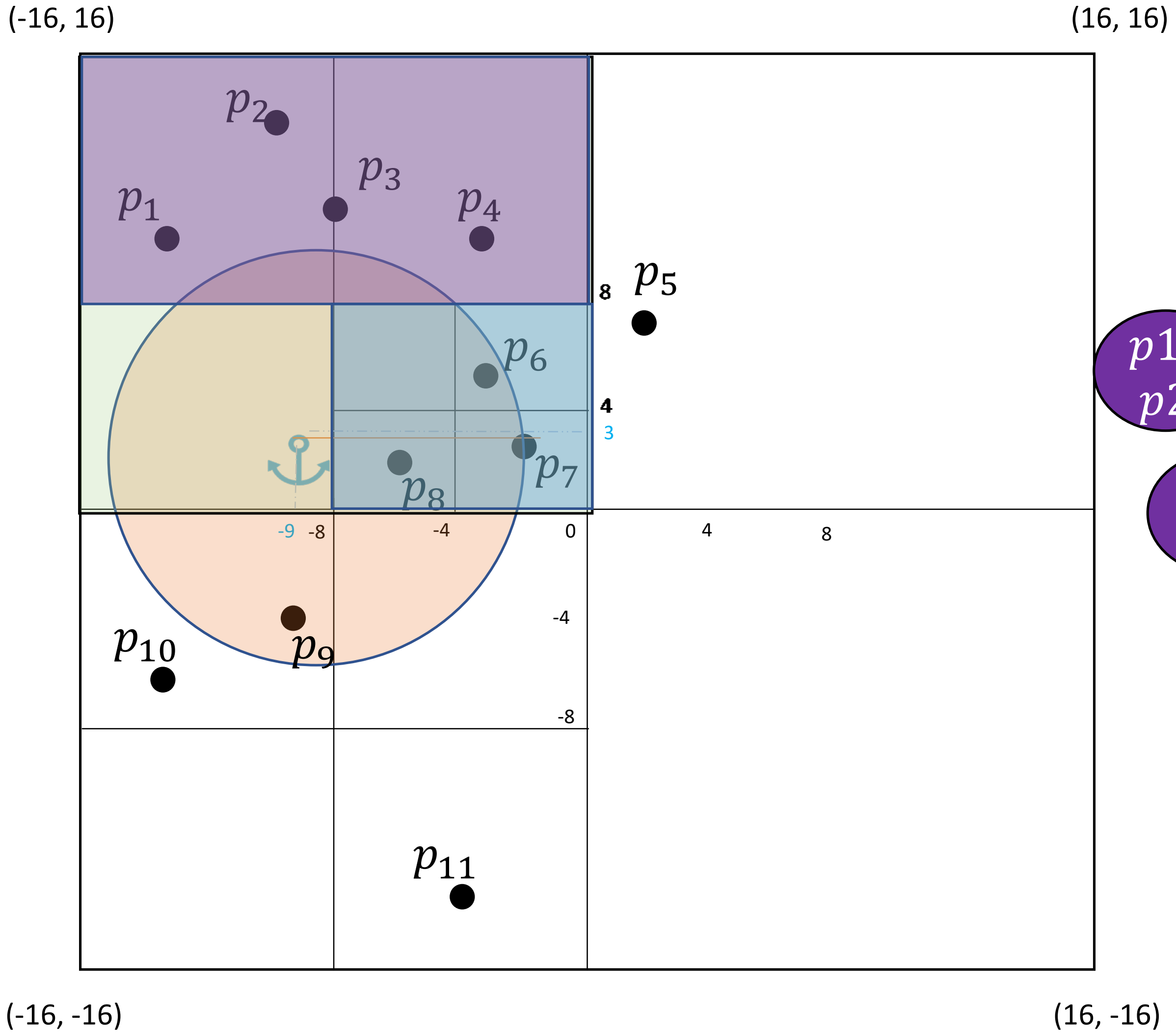


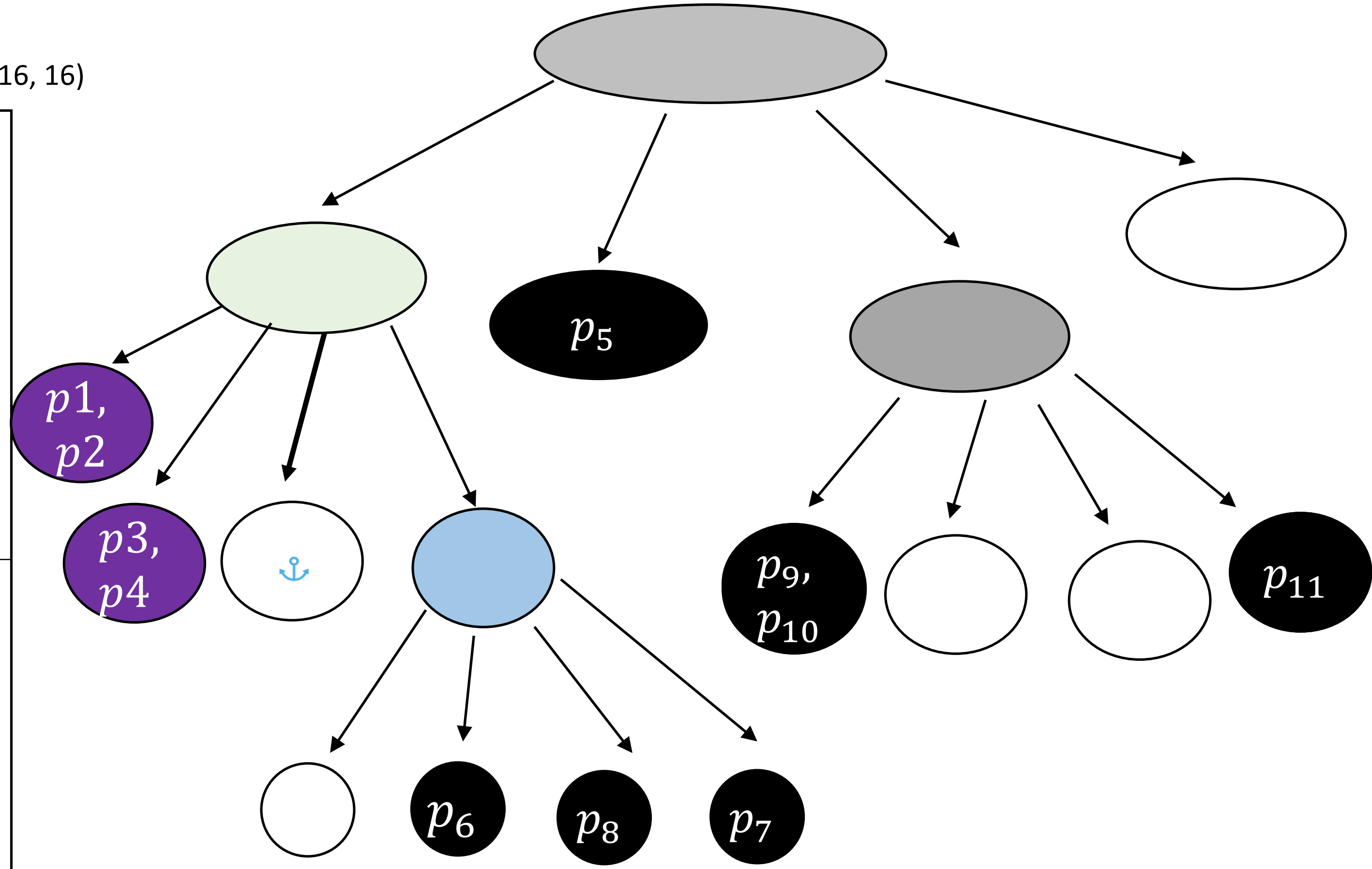


Which other subtree do I visit first?

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

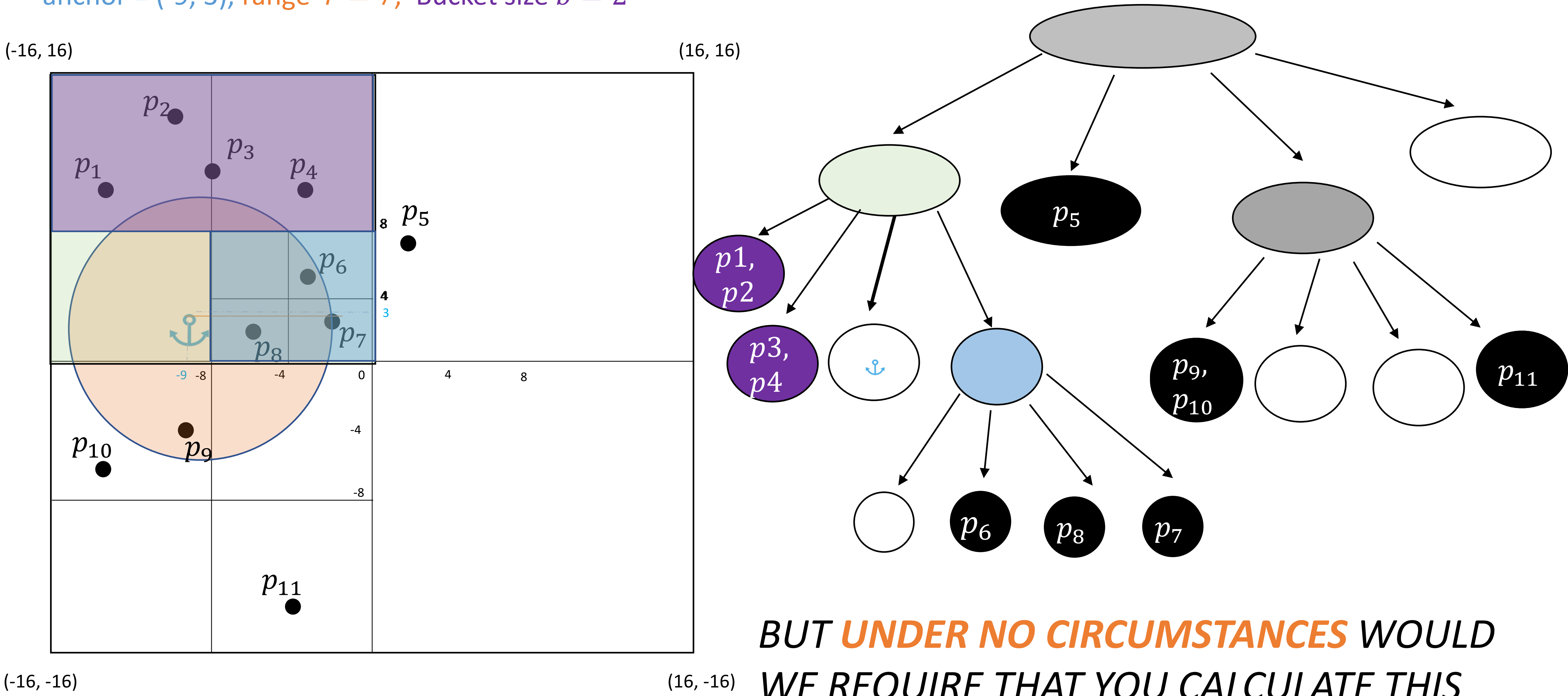




*Seems as if the **SE child** would bear the most fruit fastest, since it has **the largest area of intersection** with the range…*

Example of range query in PR-QuadTree

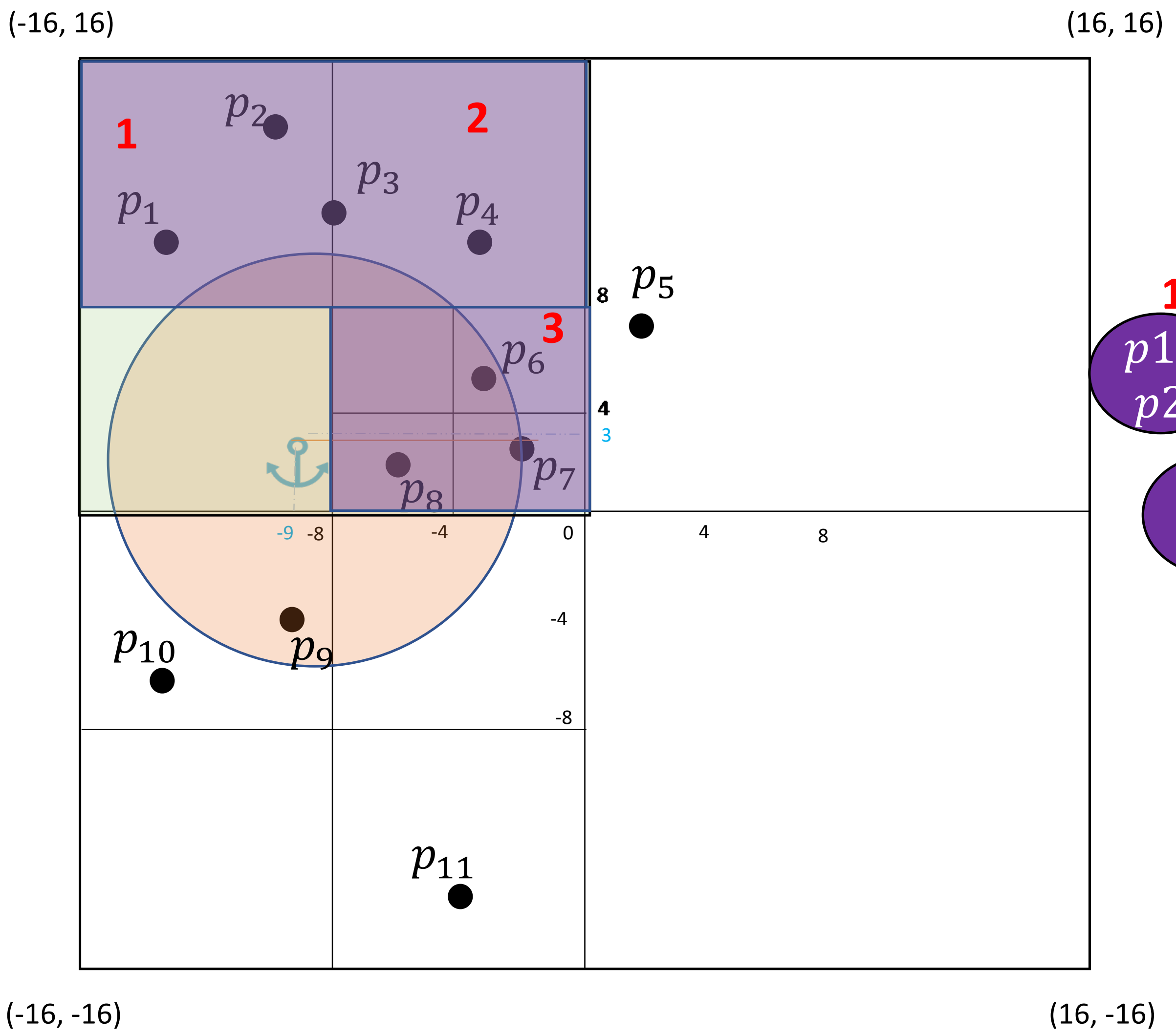
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

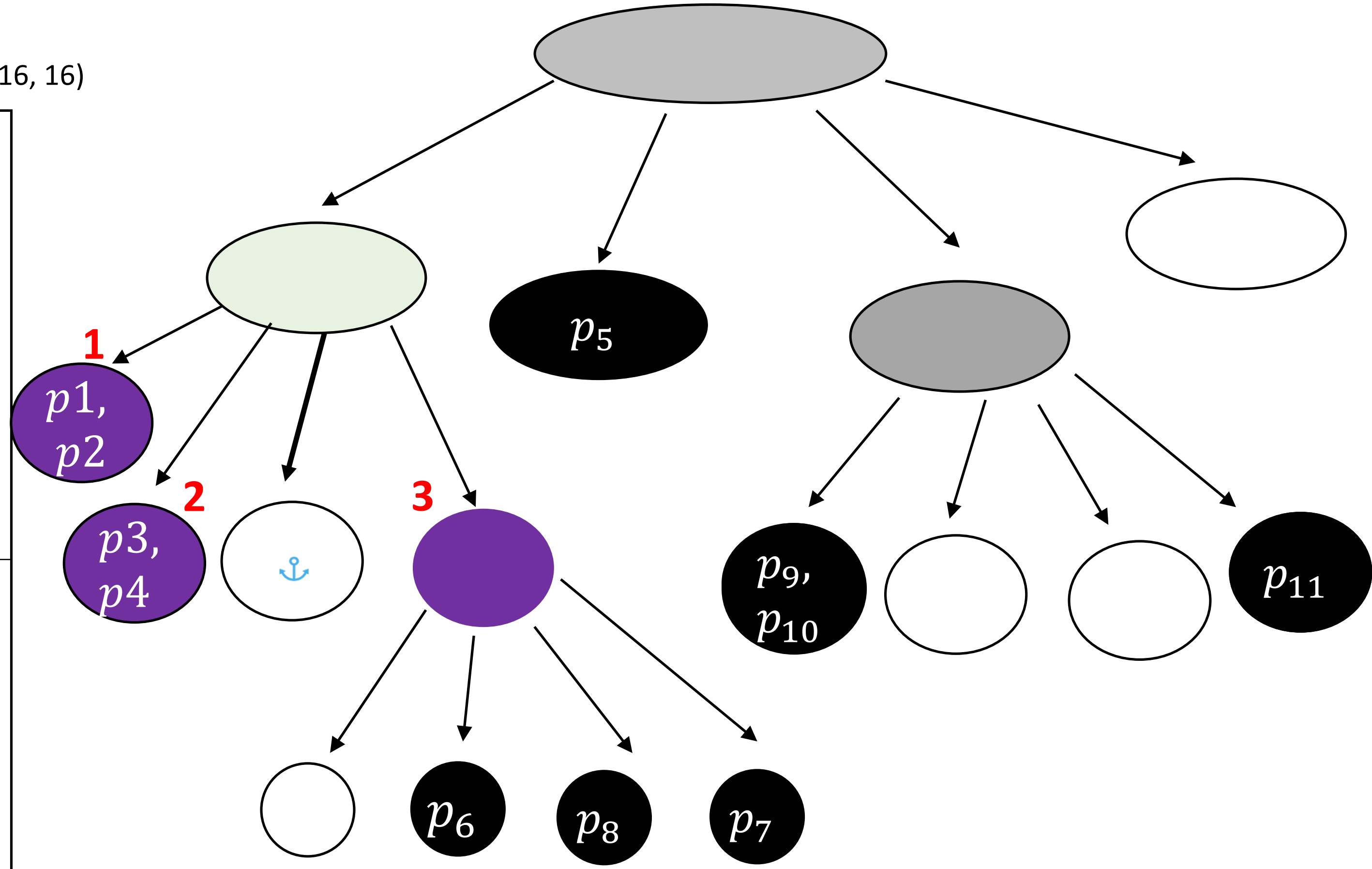


BUT UNDER NO CIRCUMSTANCES WOULD WE REQUIRE THAT YOU CALCULATE THIS AREA….

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

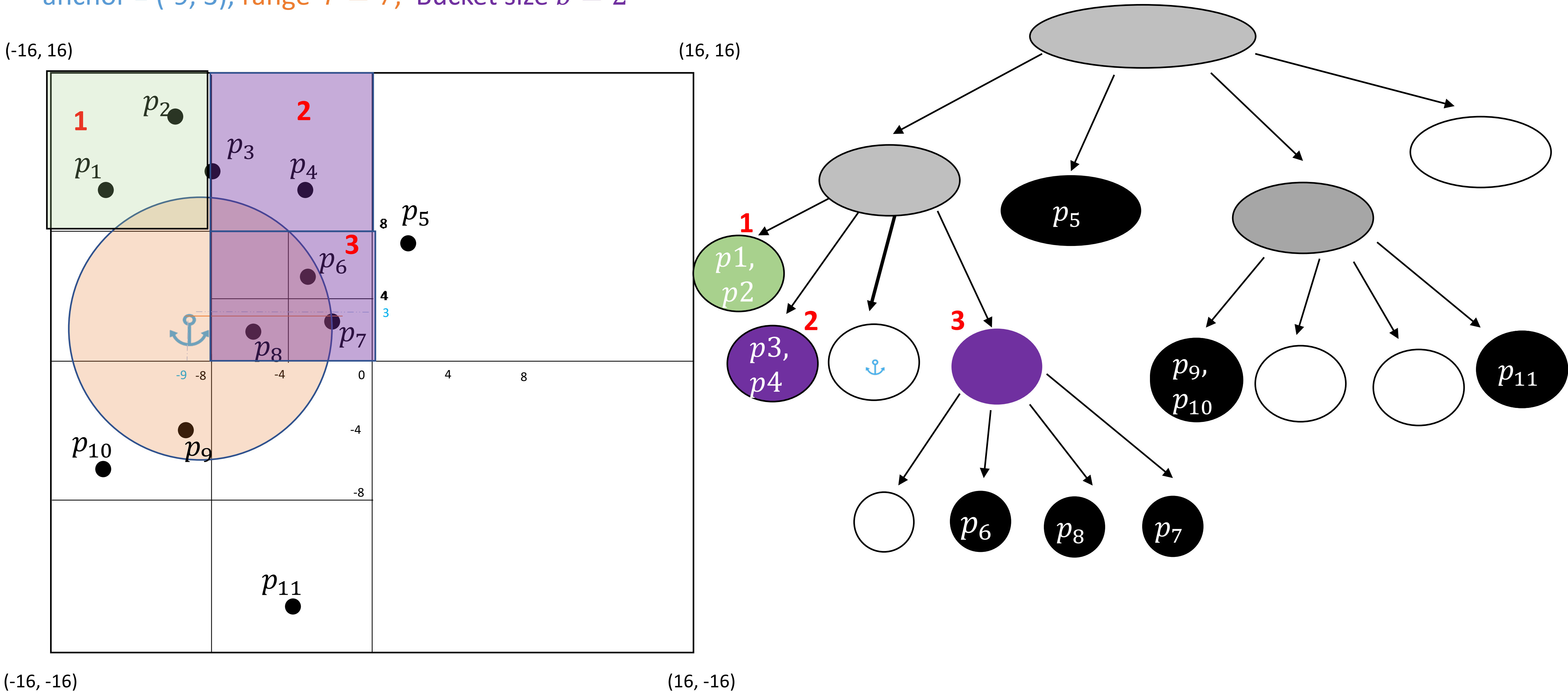




Just do ***Z-order*** *over the remaining children!*

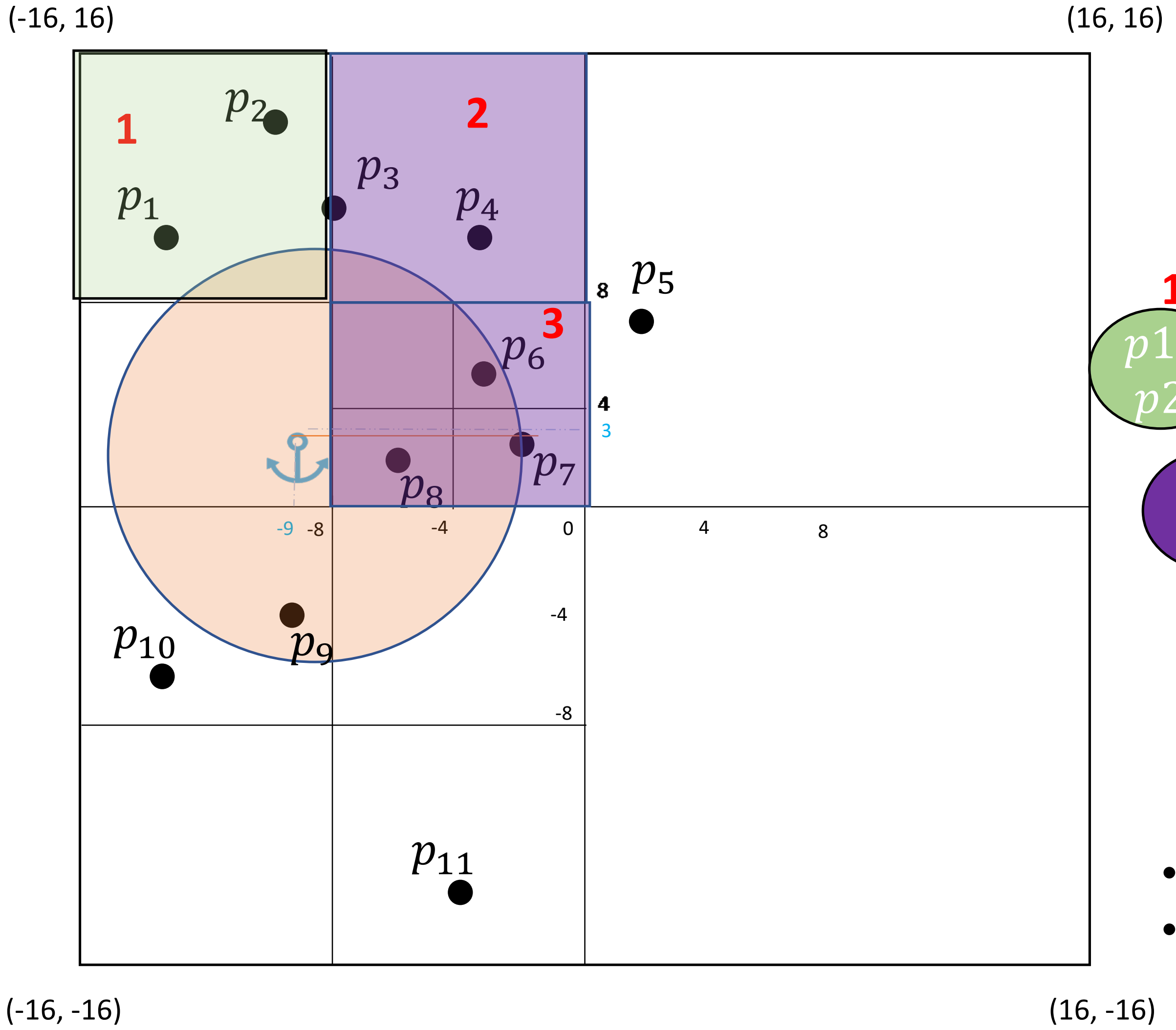
Example of range query in PR-QuadTree

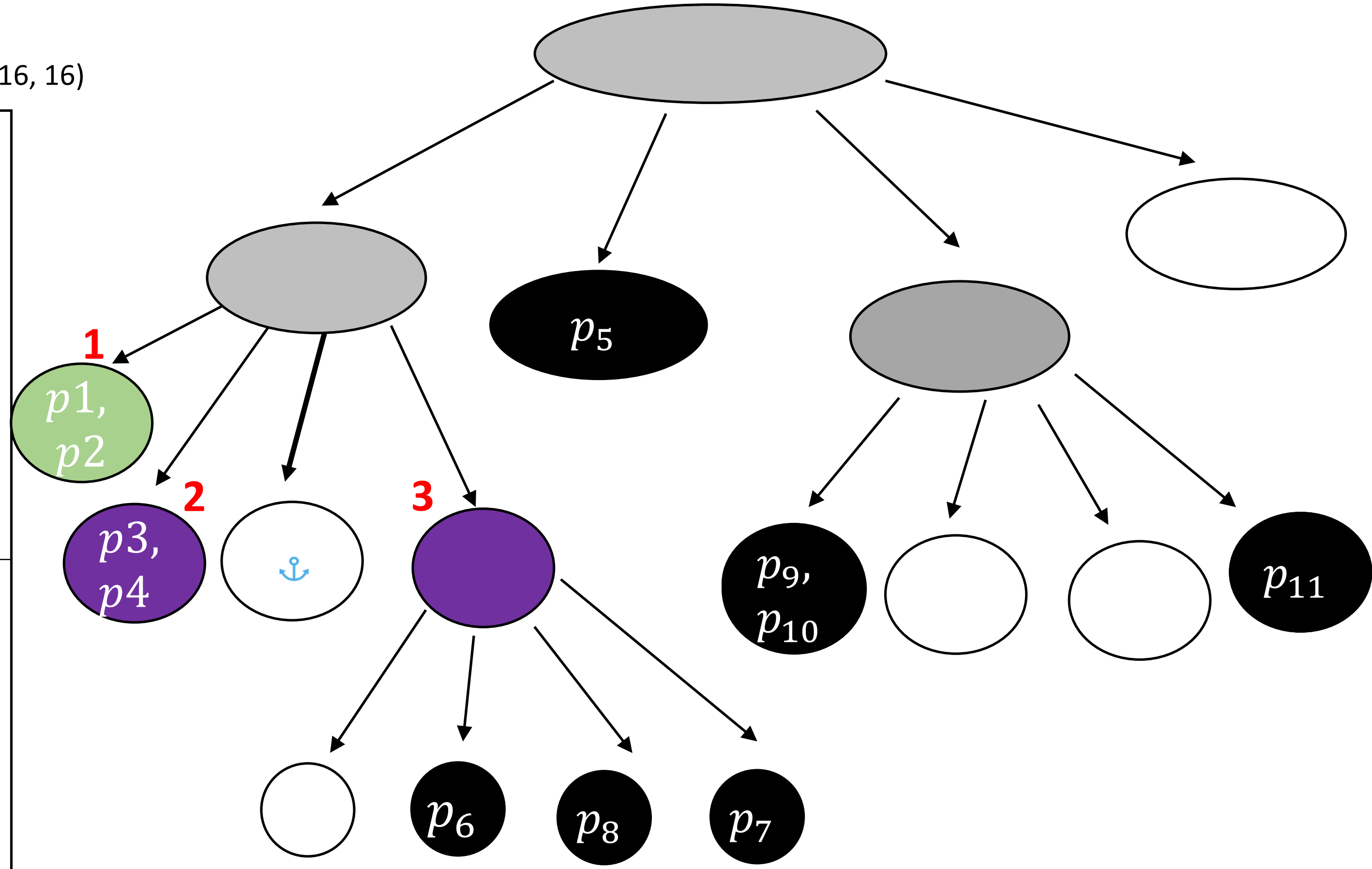
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

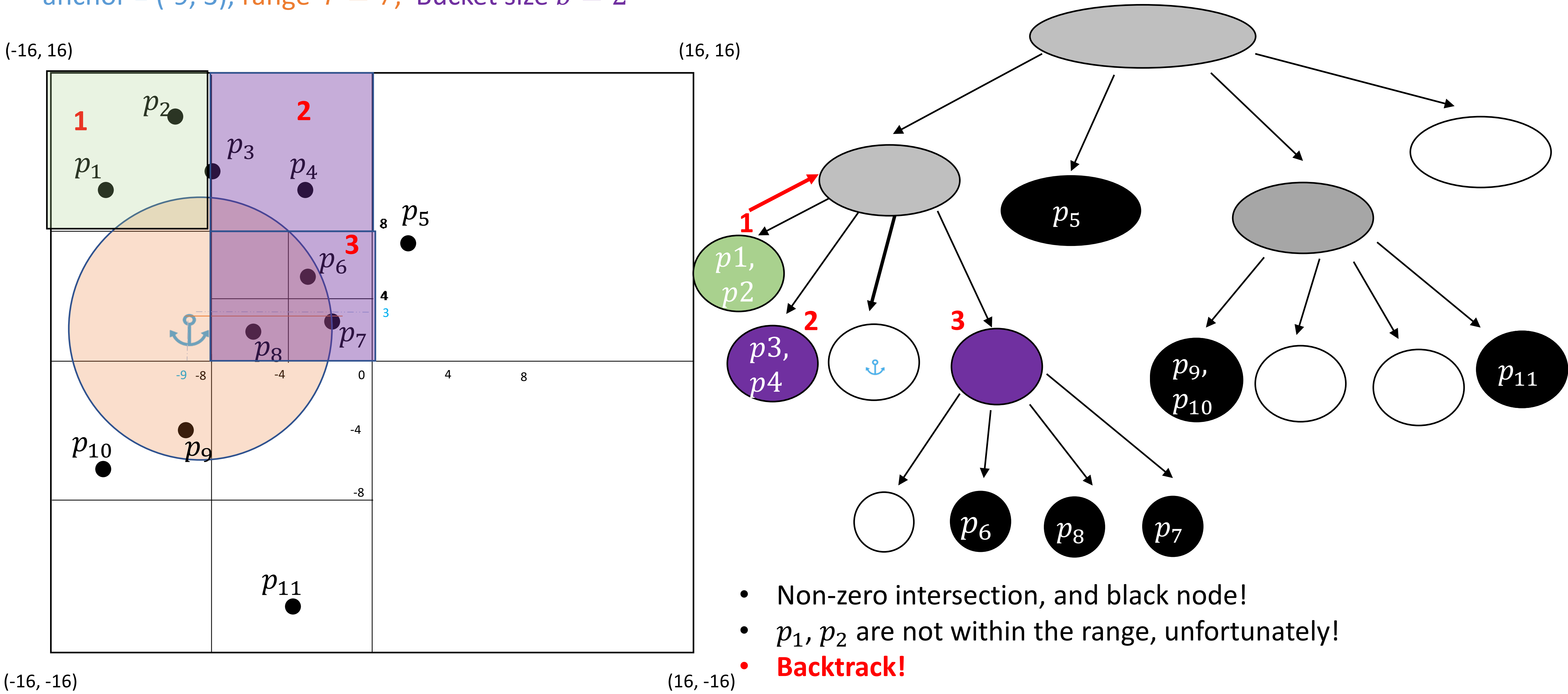




- Non-zero intersection, and black node!
- p_1, p_2 are not within the range, unfortunately!

Example of range query in PR-QuadTree

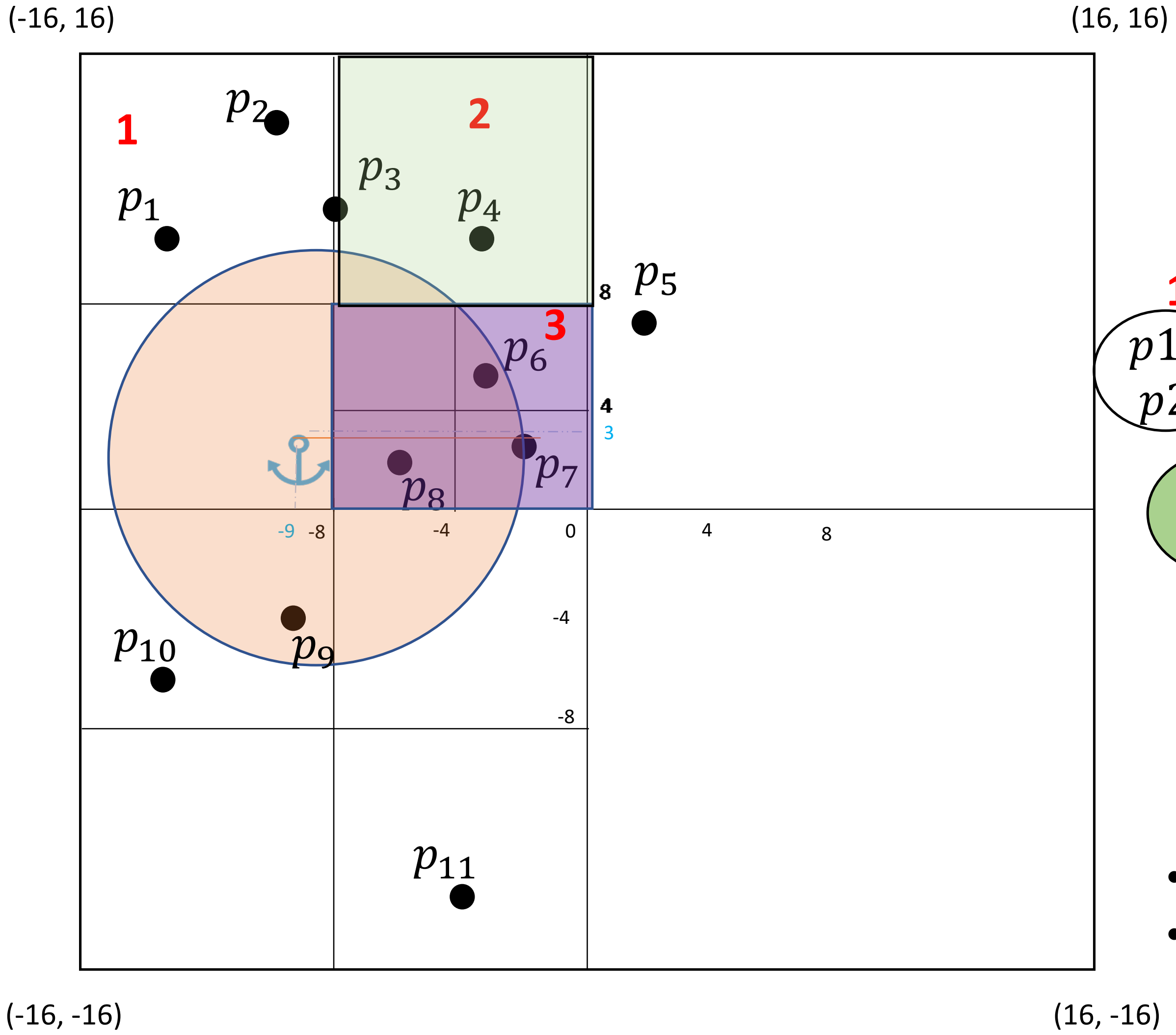
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

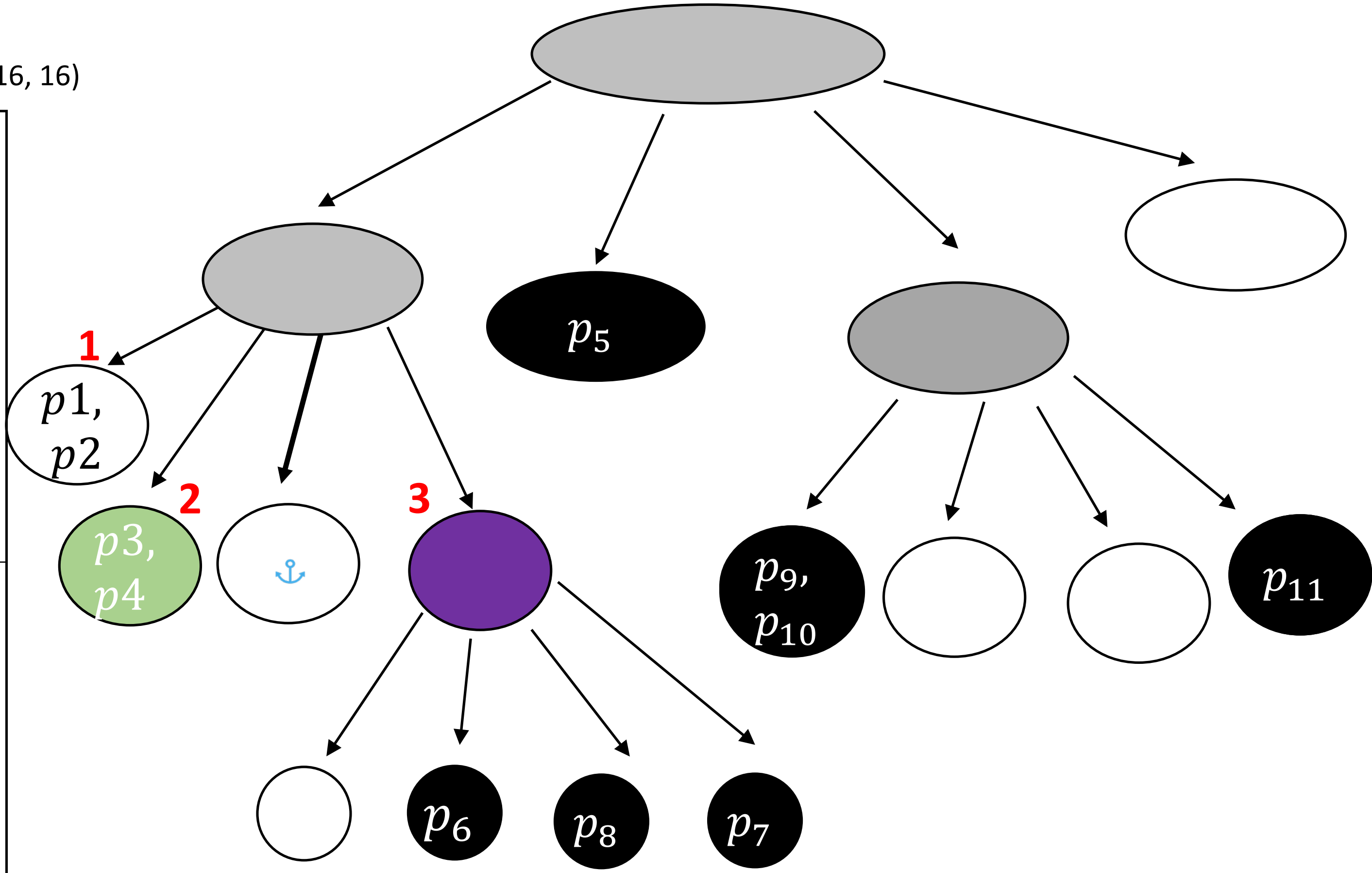


- Non-zero intersection, and black node!
- p_1, p_2 are not within the range, unfortunately!
- **Backtrack!**

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

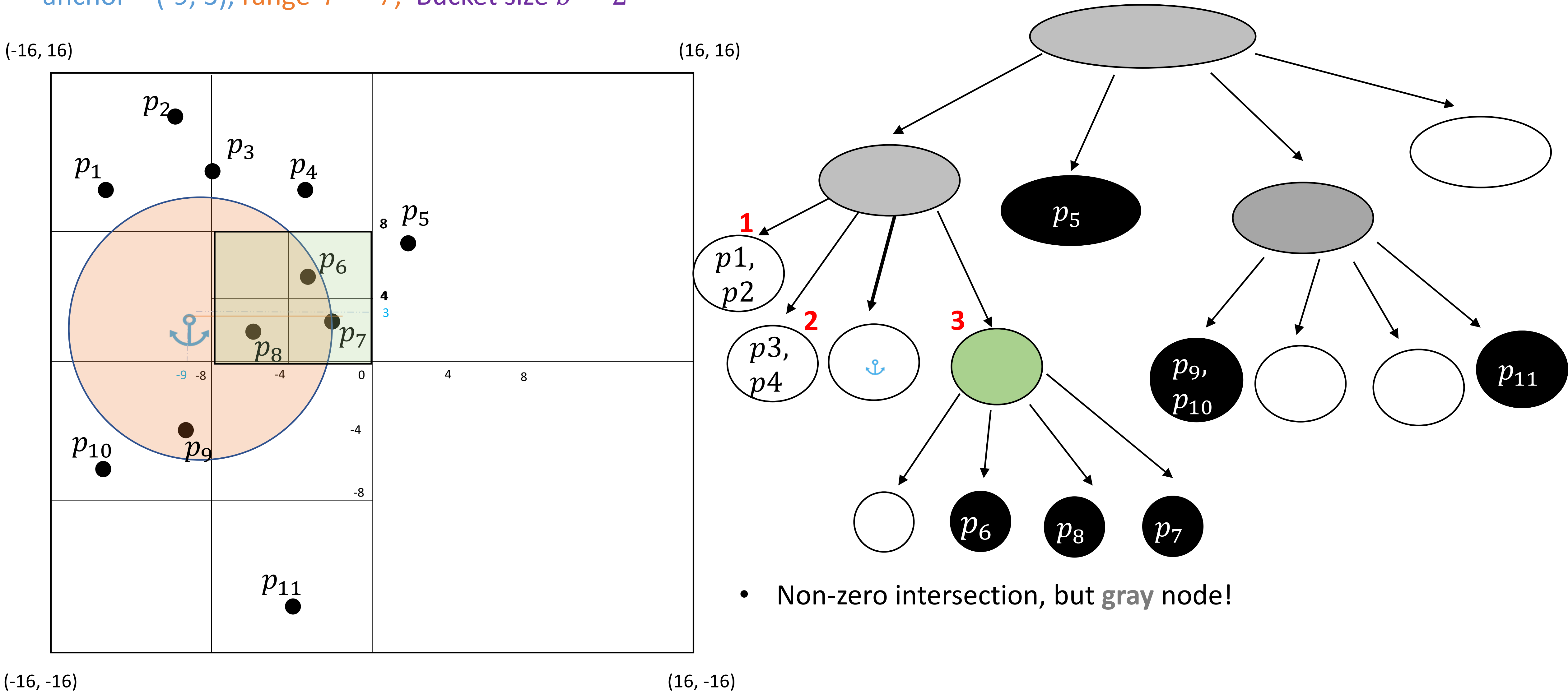




- Black node, non-zero intersection
- p_3 and p_4 are too far away ☹

Example of range query in PR-QuadTree

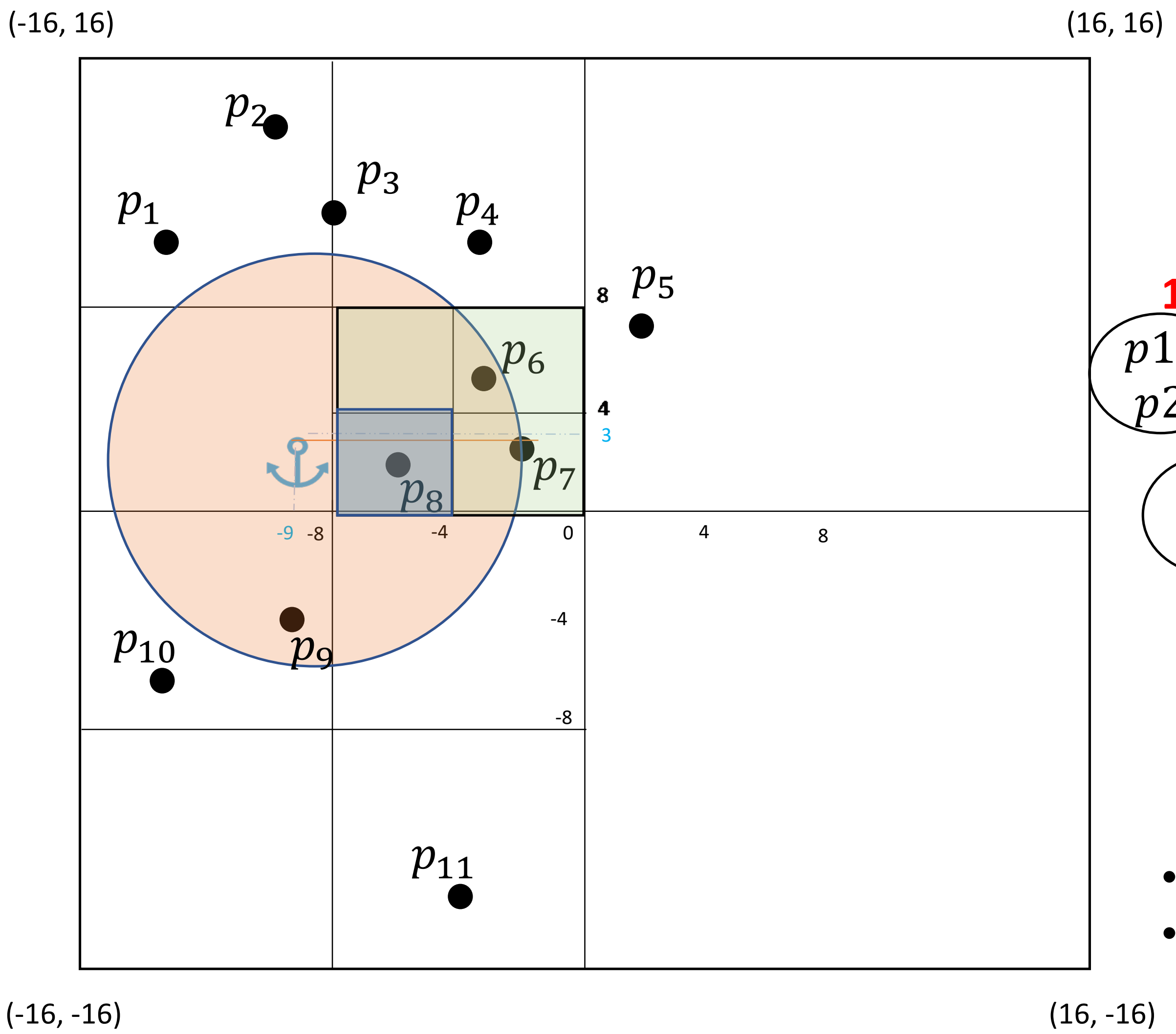
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

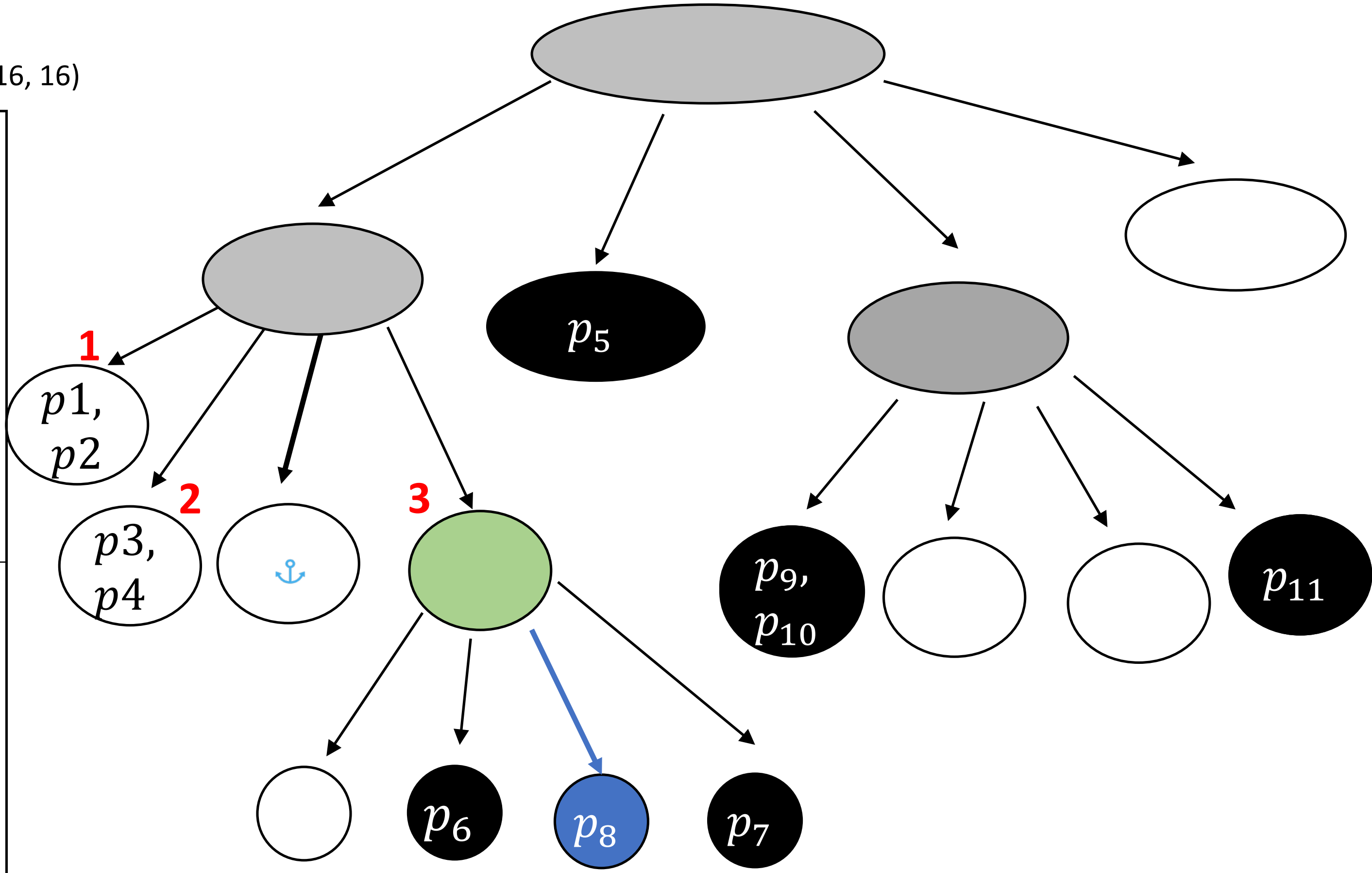


- Non-zero intersection, but **gray** node!

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

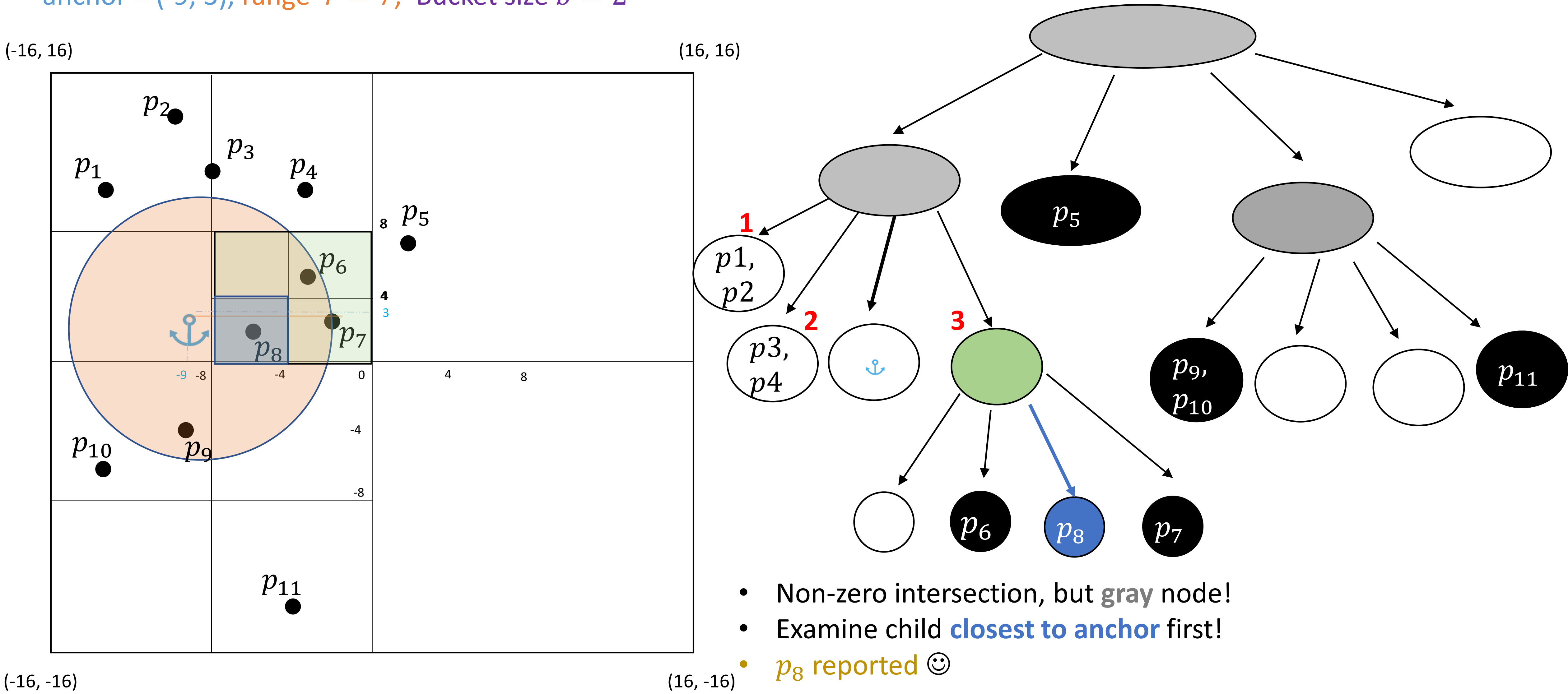




- Non-zero intersection, but gray node!
- Examine child **closest to anchor** first!

Example of range query in PR-QuadTree

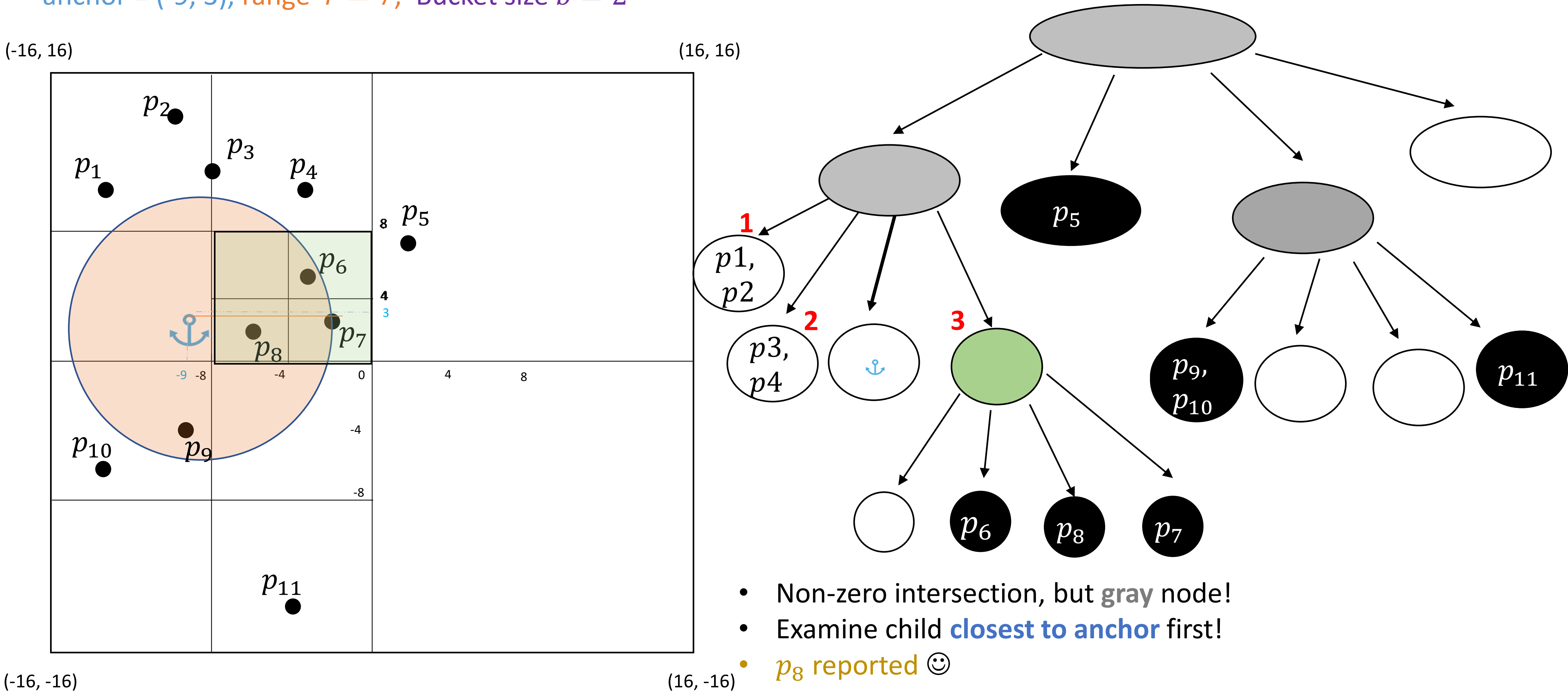
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



- Non-zero intersection, but **gray** node!
- Examine child **closest to anchor** first!
- p_8 reported ☺

Example of range query in PR-QuadTree

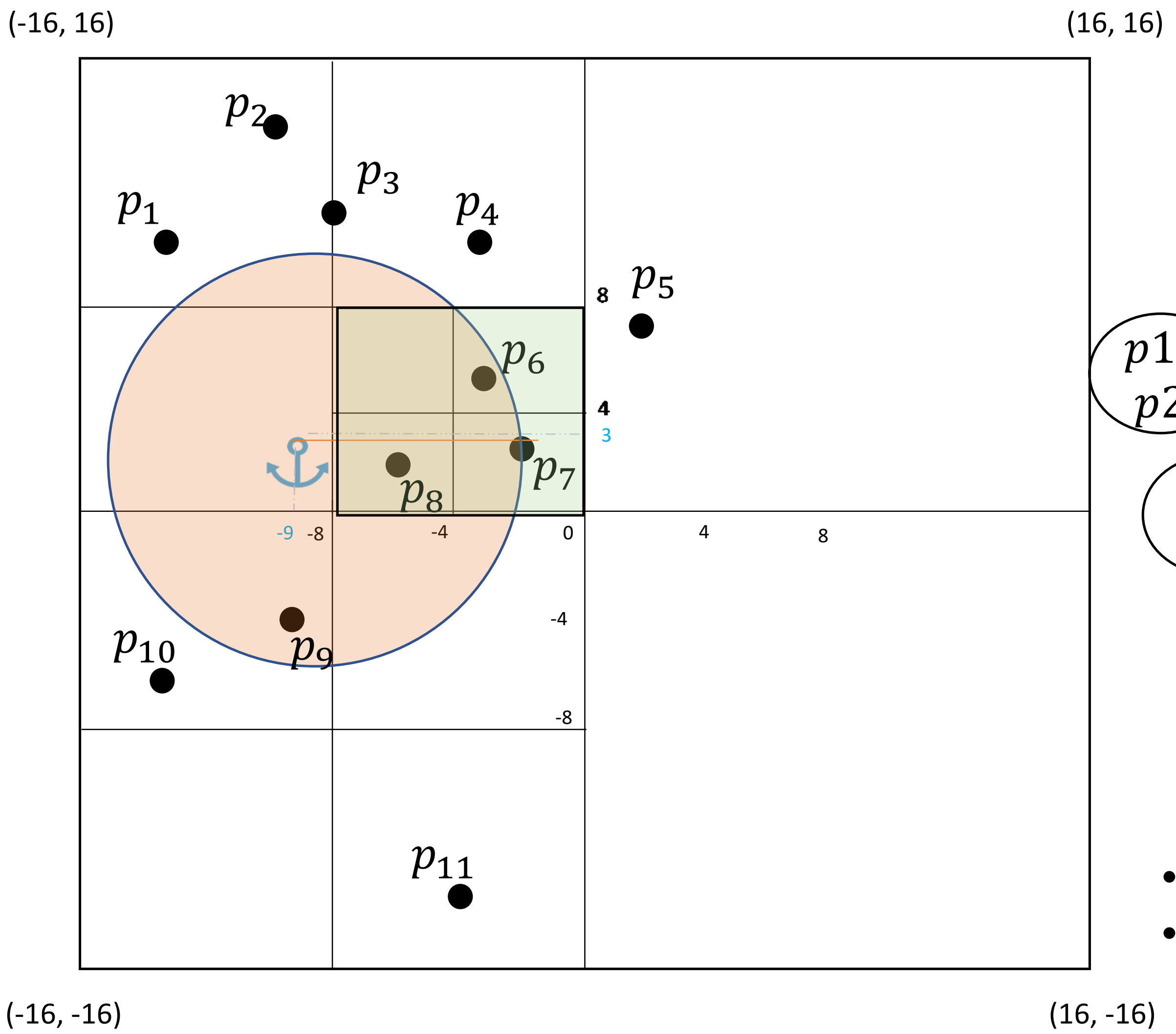
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

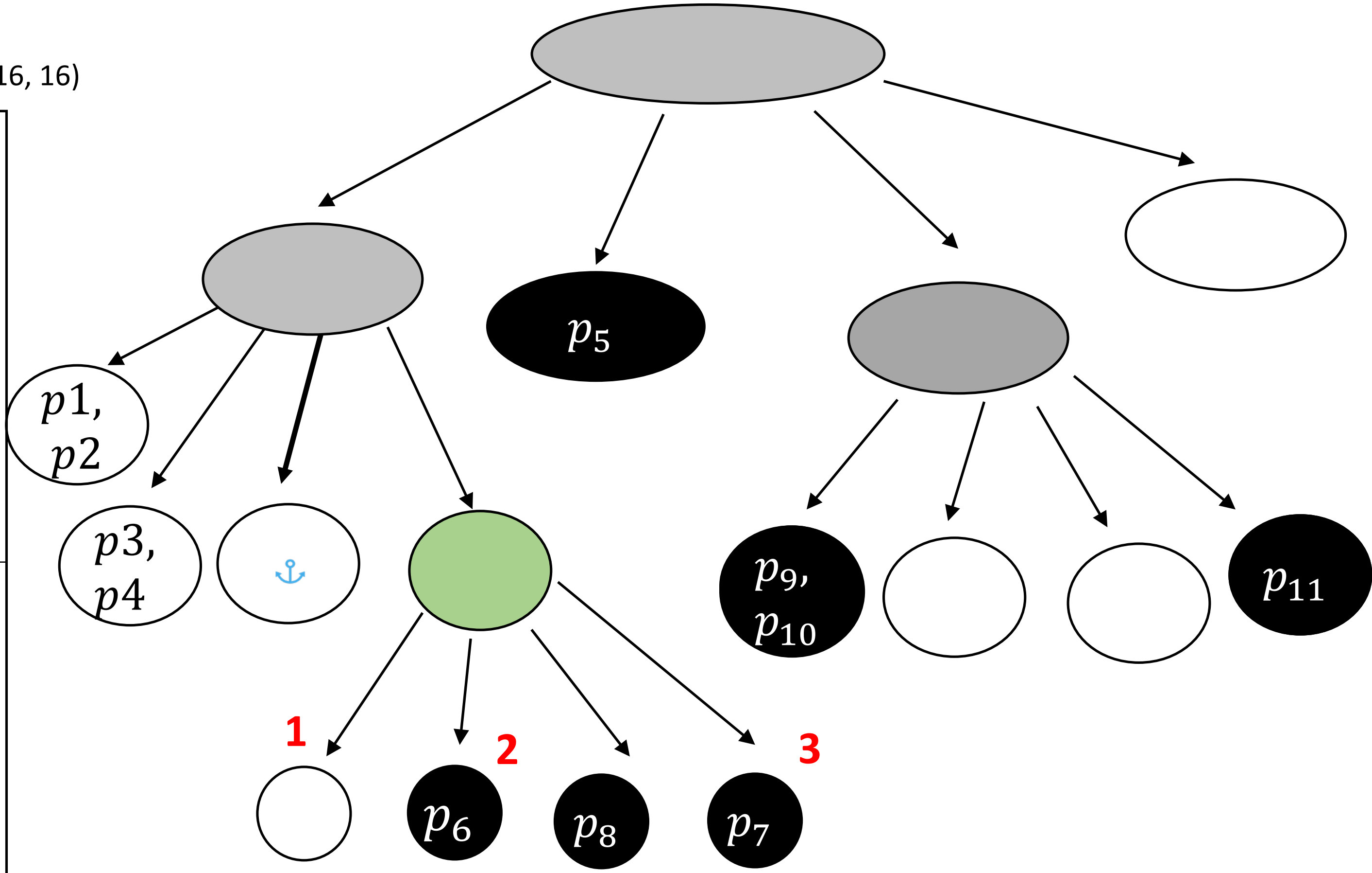


- Non-zero intersection, but **gray** node!
- Examine child **closest to anchor** first!
- p_8 reported ☺

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

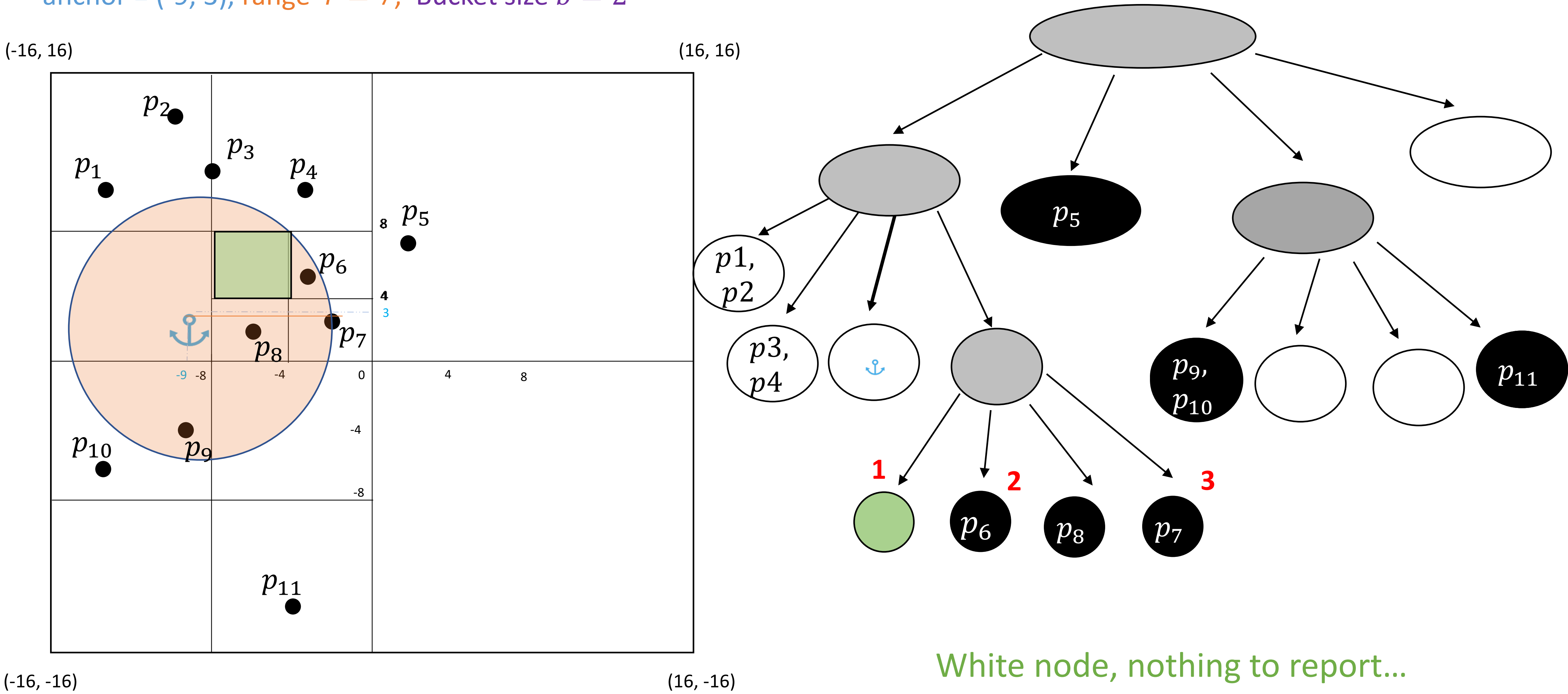




- Non-zero intersection, but **gray** node!
- Again, do not prioritize children based on area of intersection, but **on Z-order**!

Example of range query in PR-QuadTree

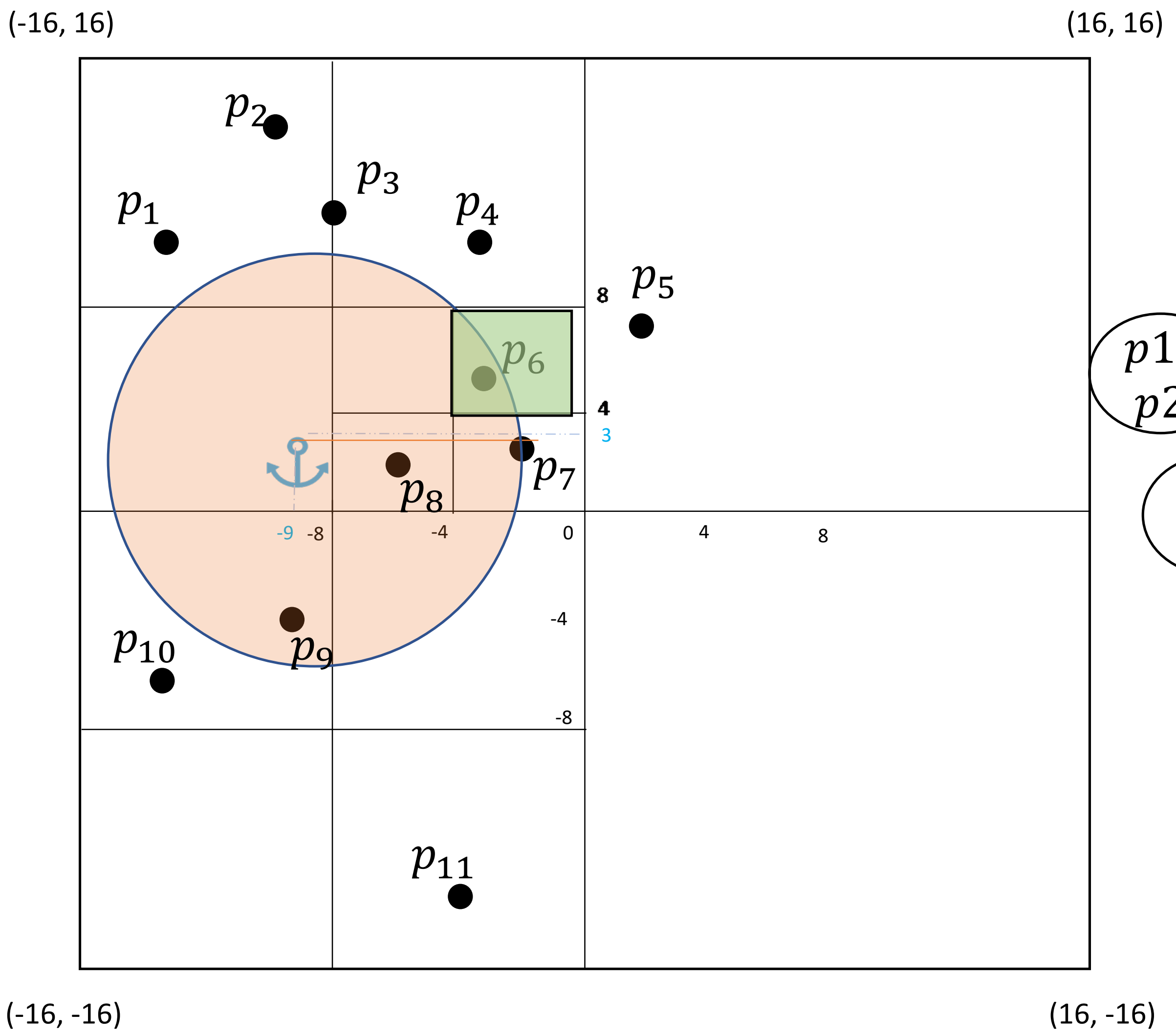
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

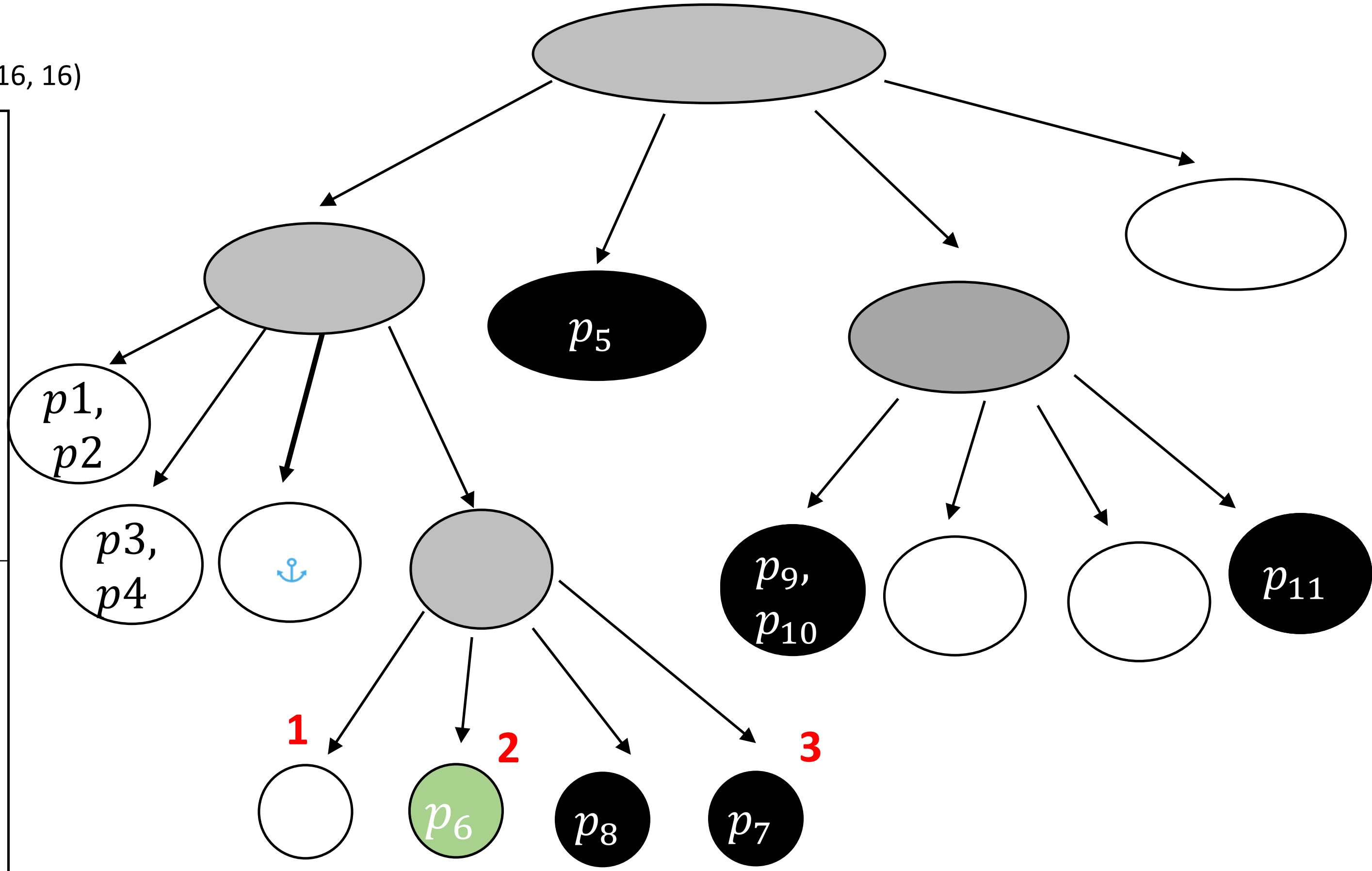


White node, nothing to report…

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

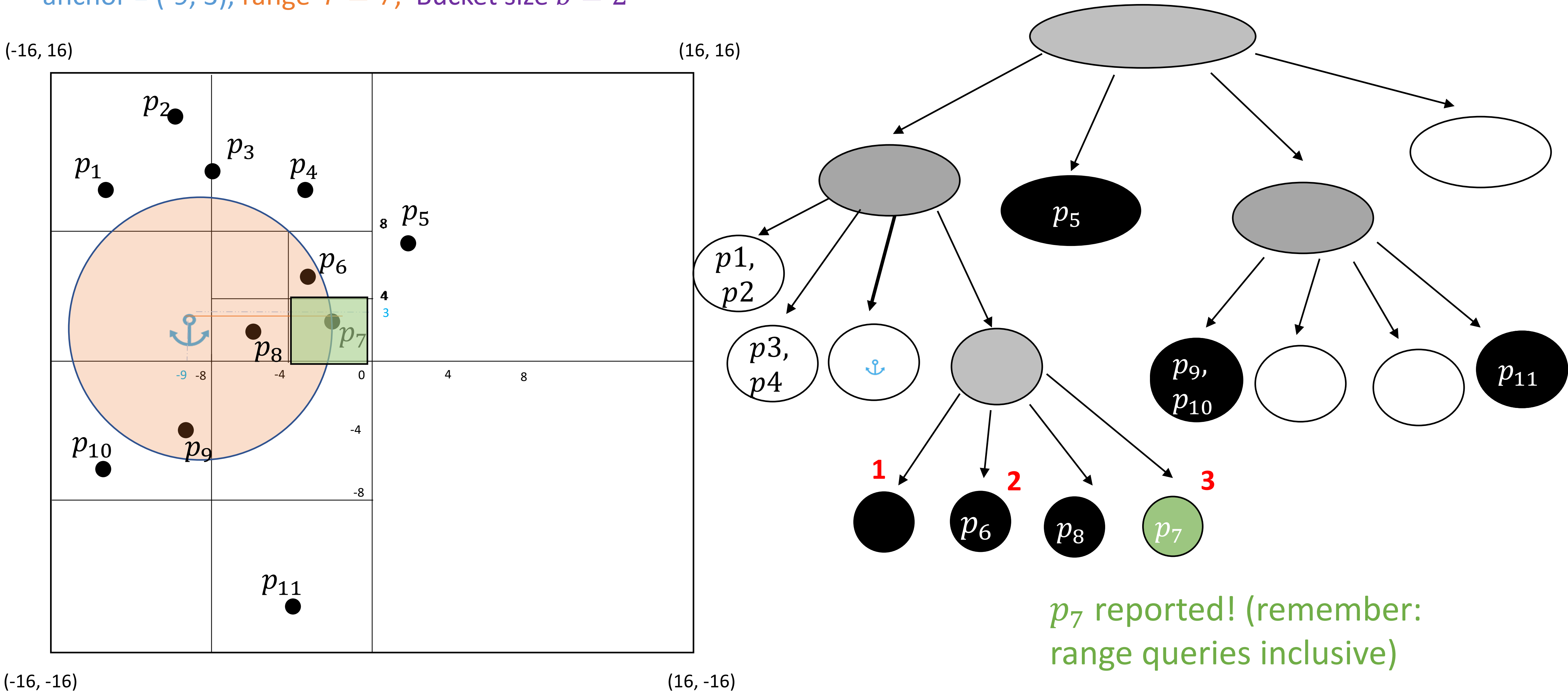




p_6 reported!

Example of range query in PR-QuadTree

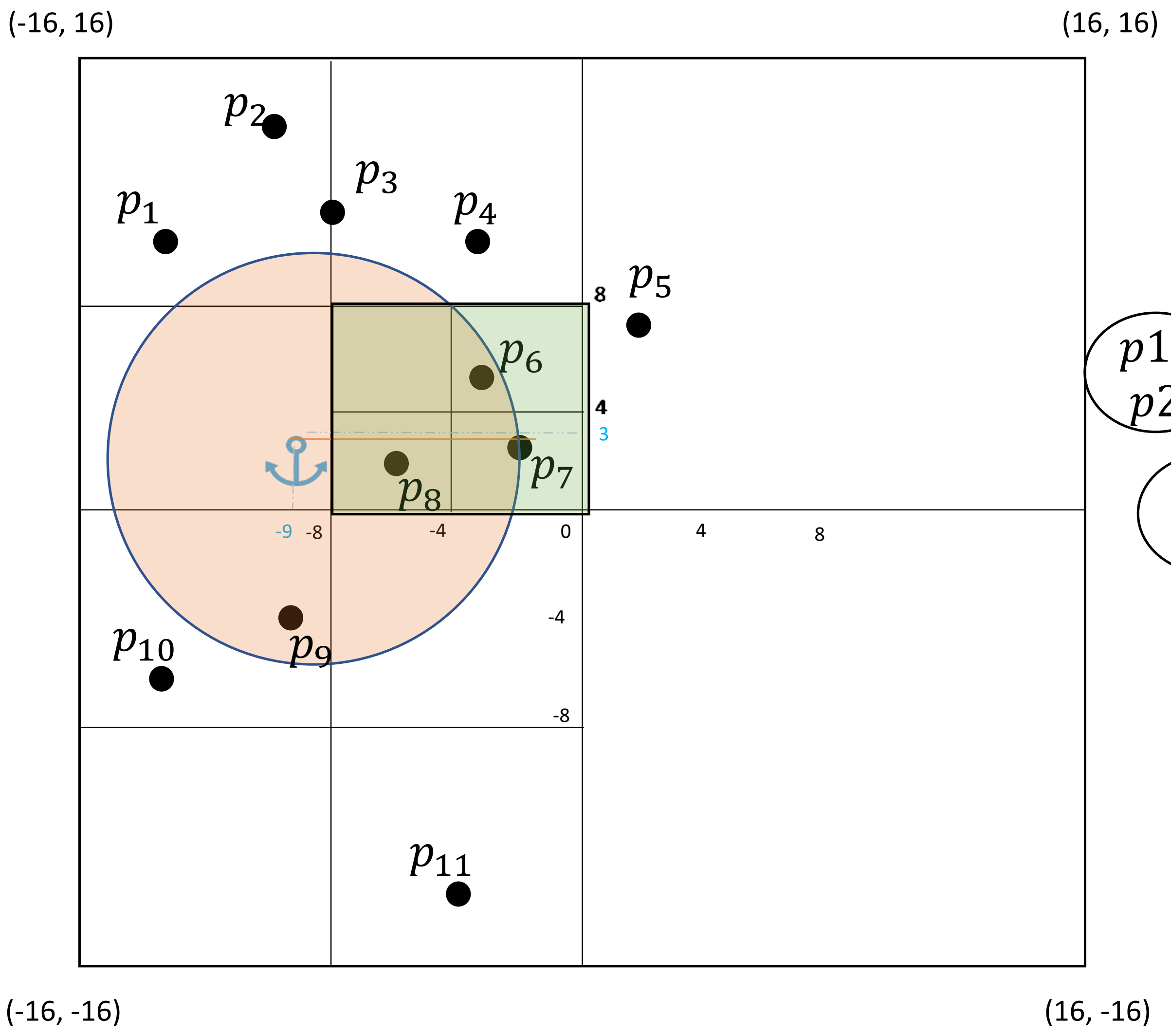
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

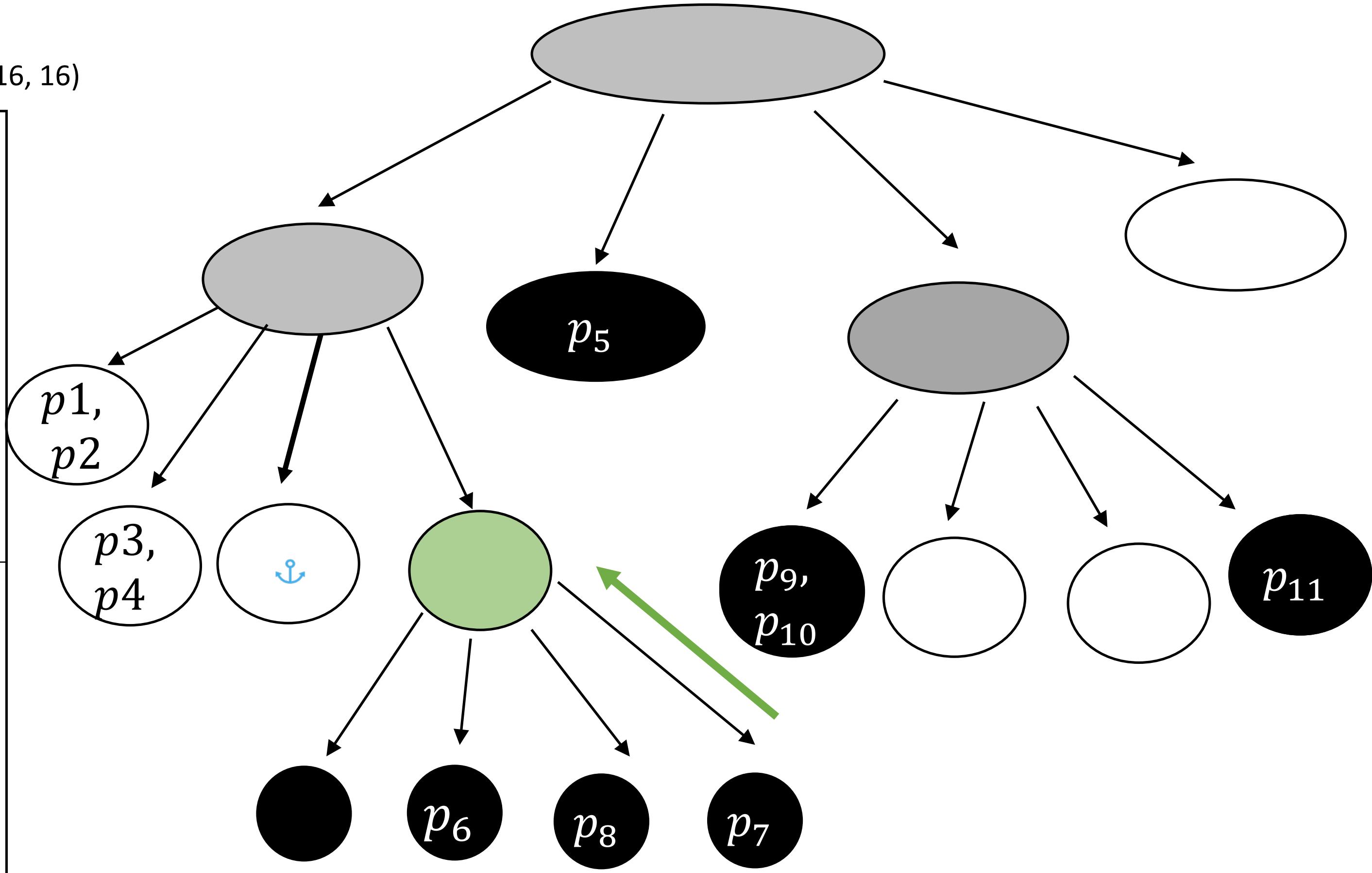


p_7 reported! (remember: range queries inclusive)

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

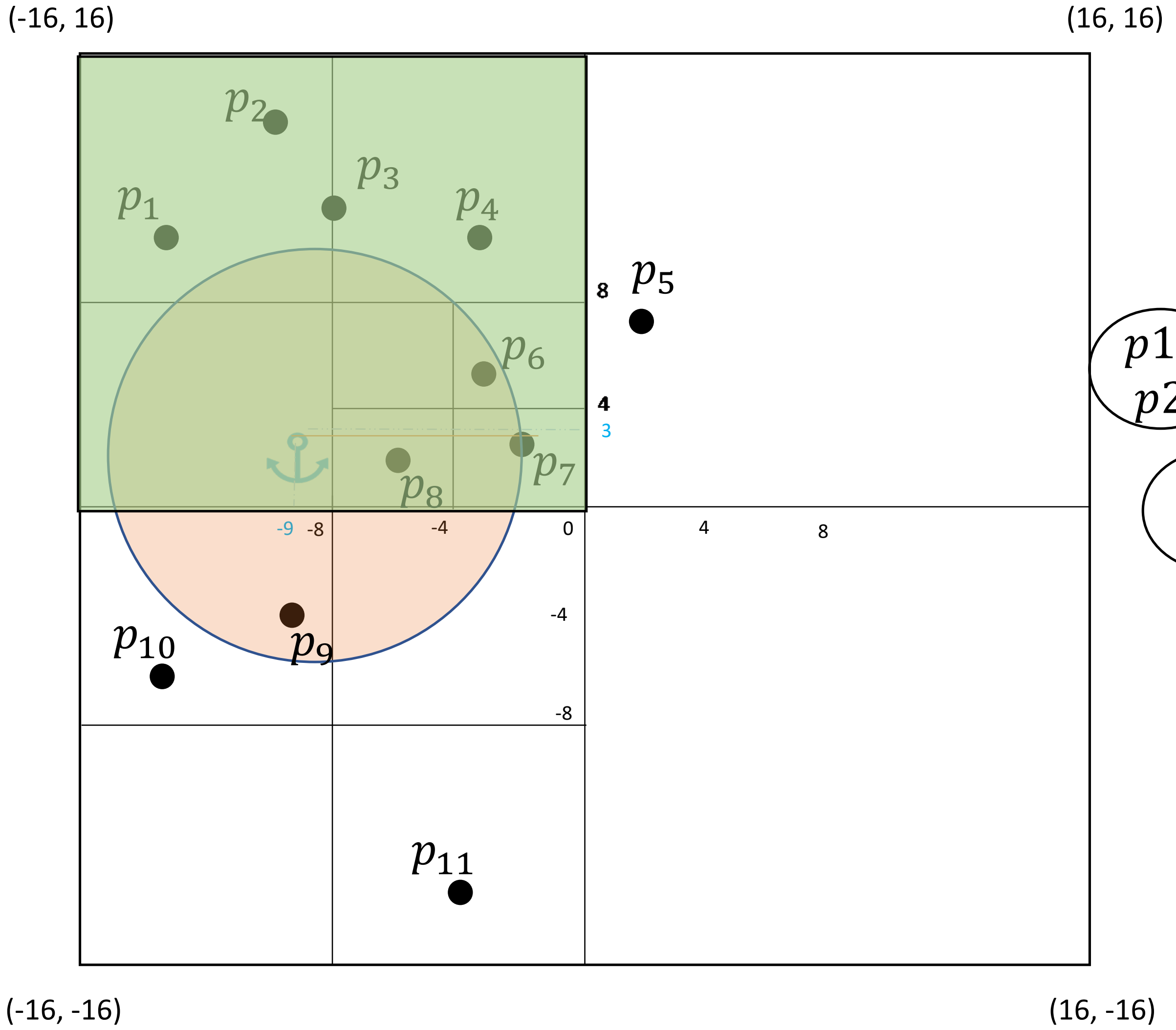


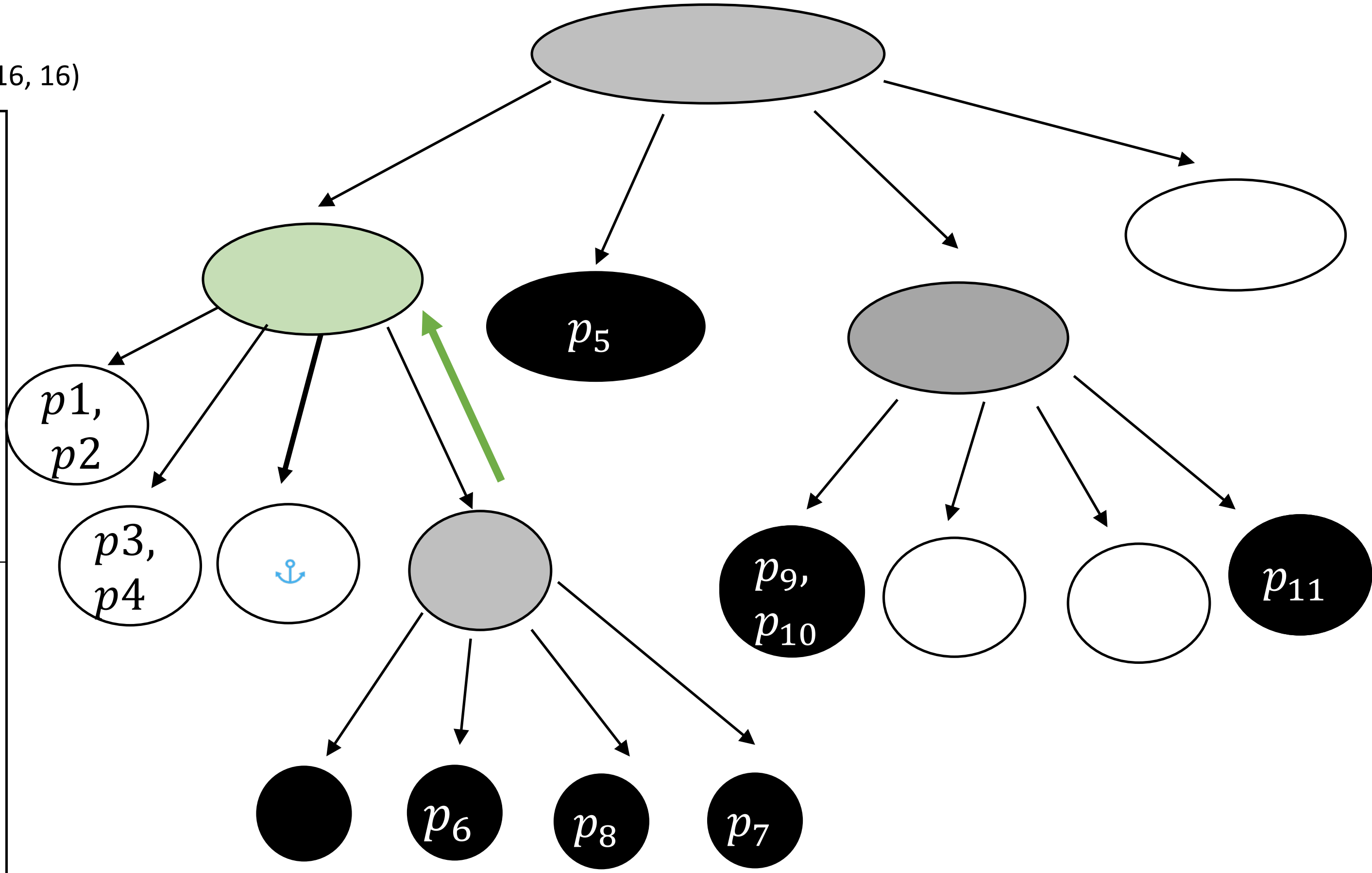


Backtracking all the way to the root…

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

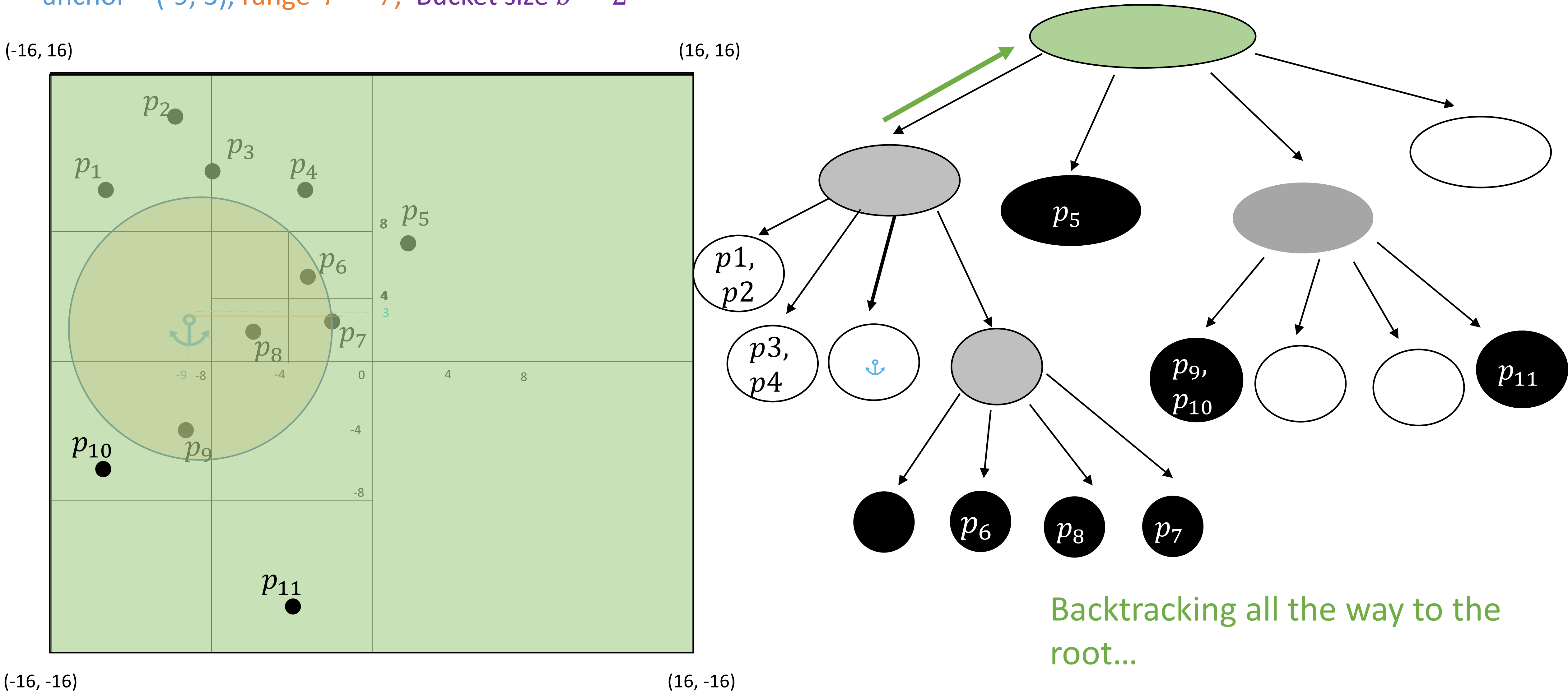




Backtracking all the way to the root…

Example of range query in PR-QuadTree

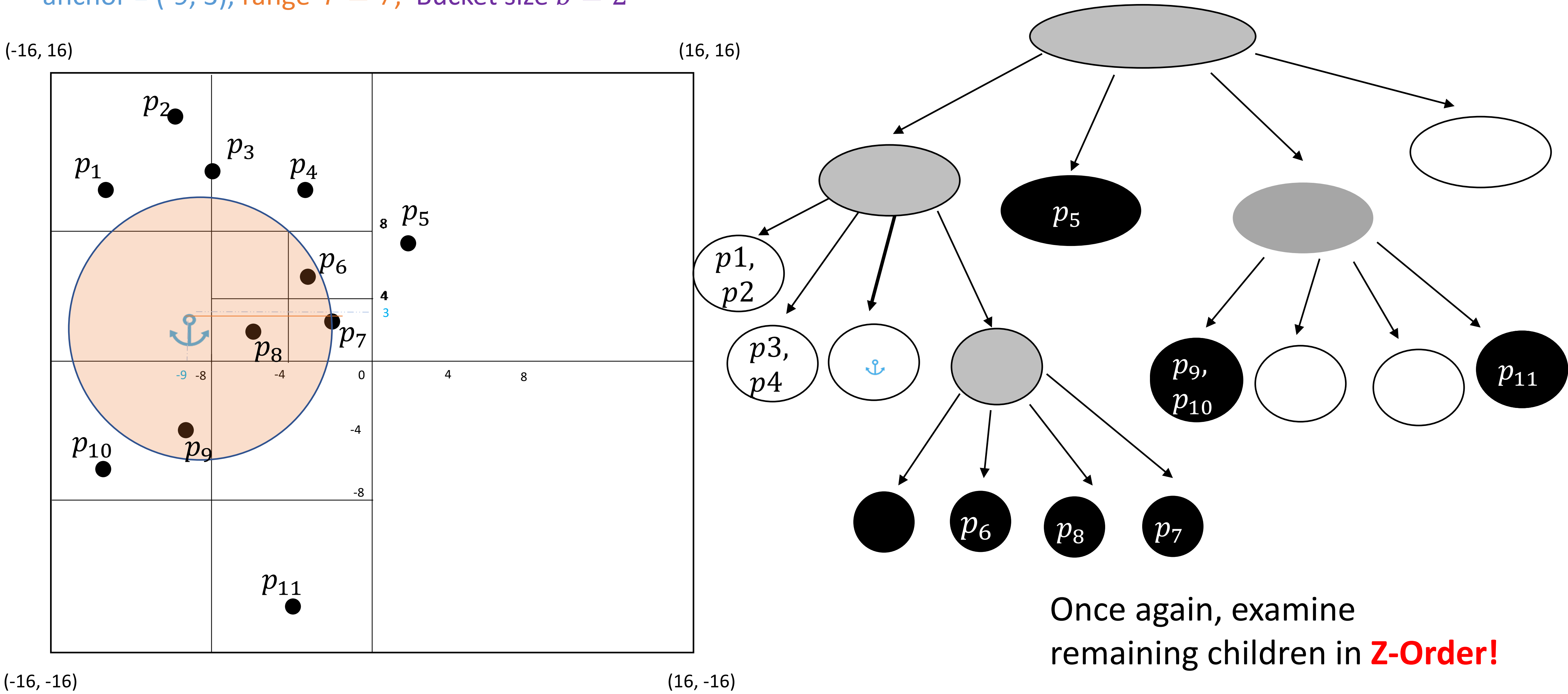
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



Backtracking all the way to the root…

Example of range query in PR-QuadTree

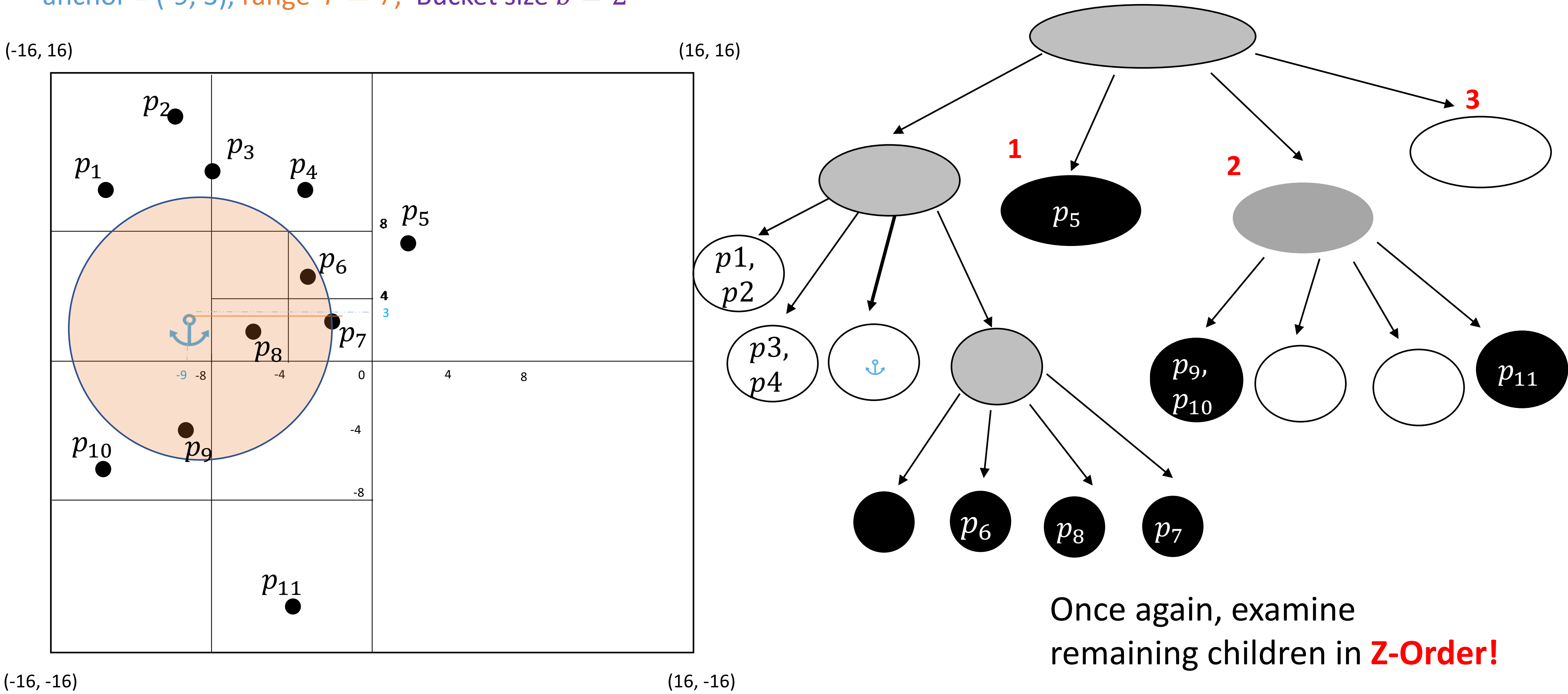
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



Once again, examine remaining children in **Z-Order!**

Example of range query in PR-QuadTree

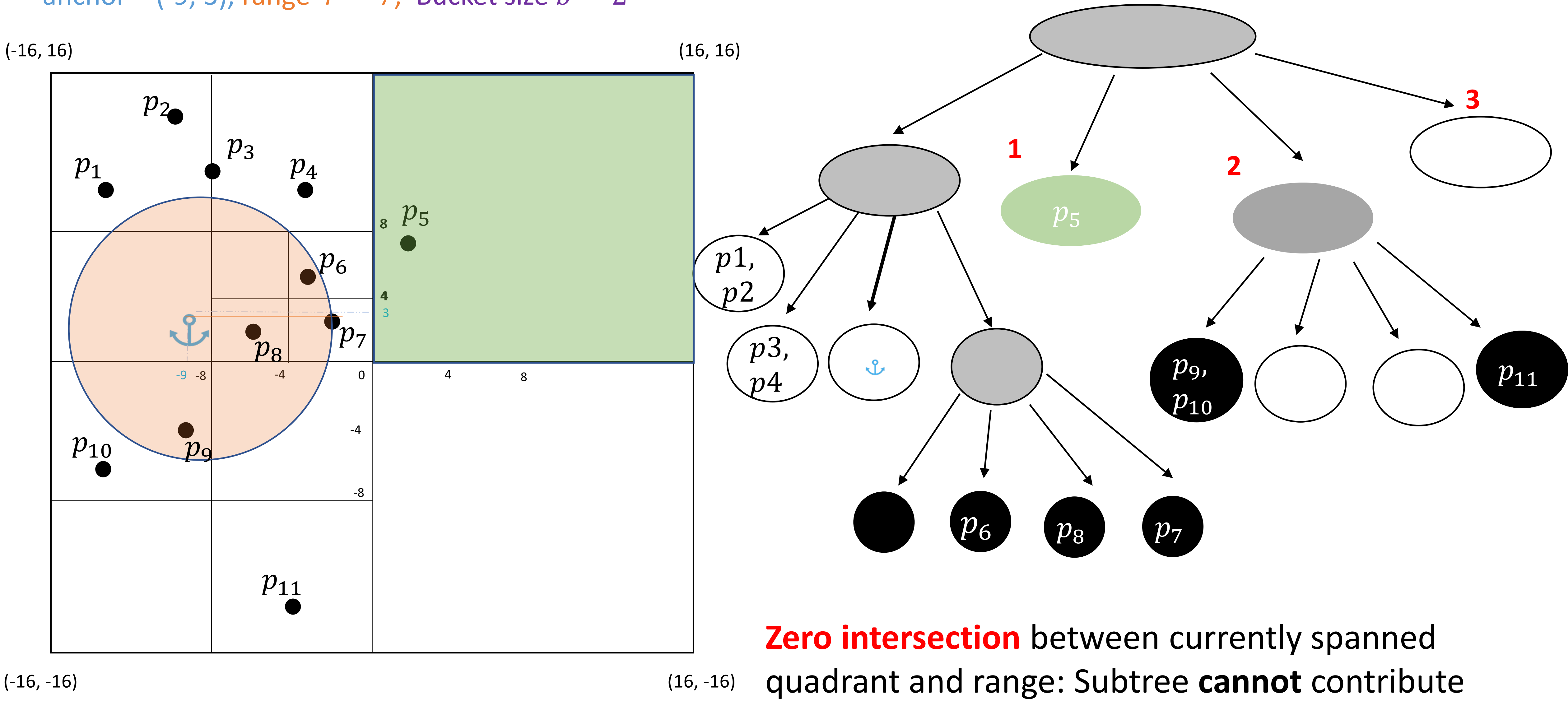
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



Once again, examine remaining children in **Z-Order!**

Example of range query in PR-QuadTree

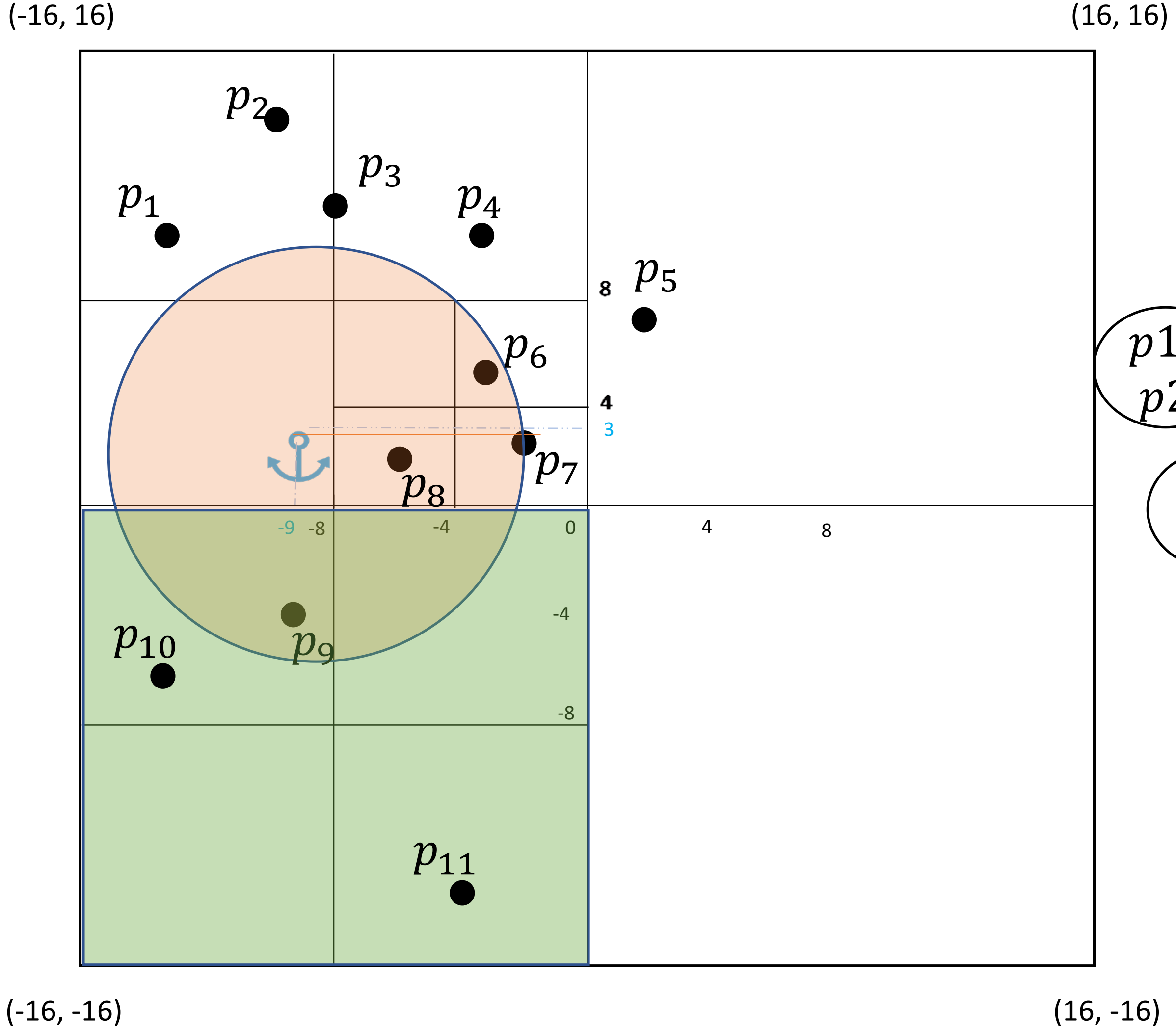
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

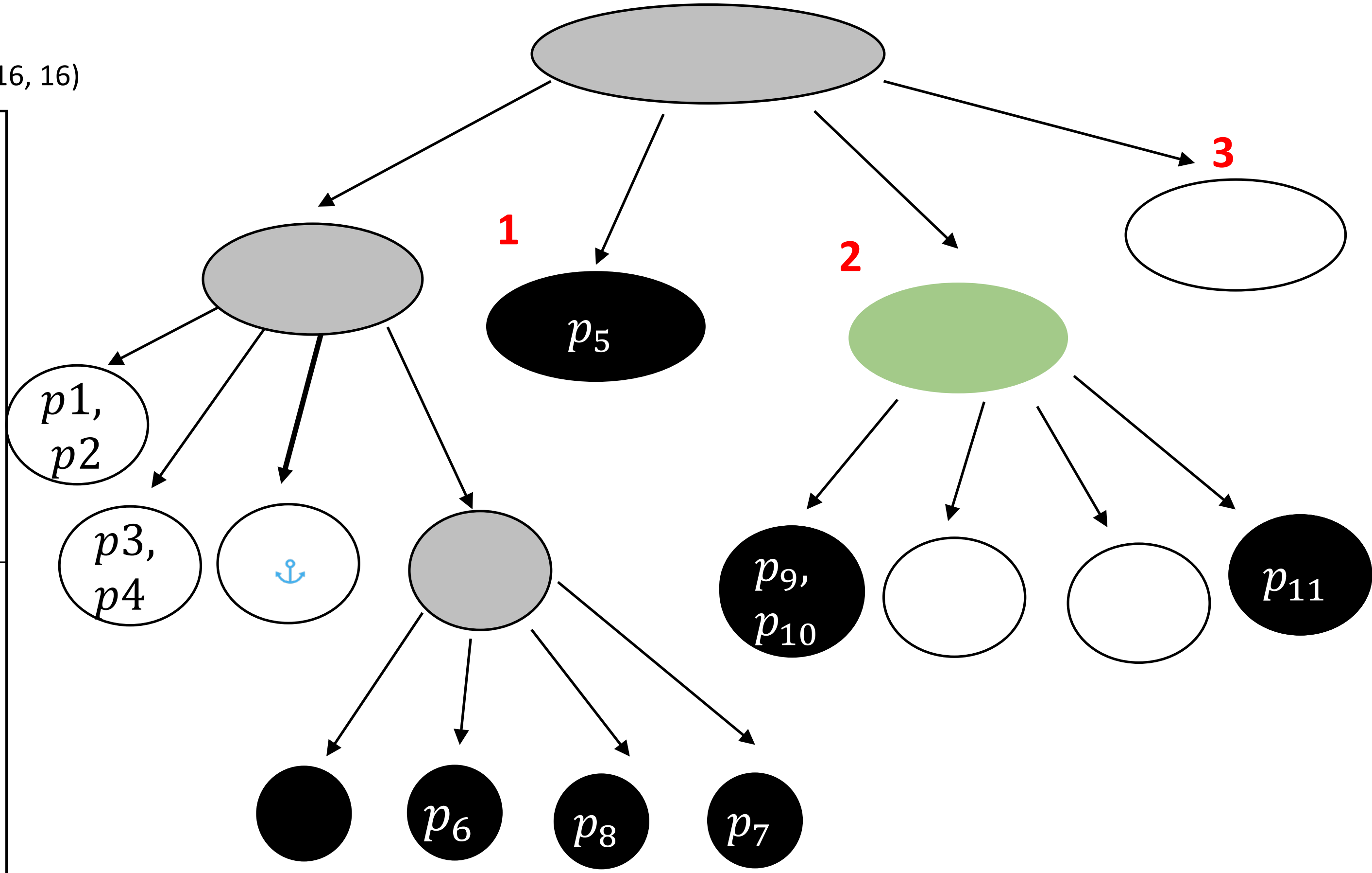


Zero intersection between currently spanned quadrant and range: Subtree **cannot** contribute to solution set. Immediately backtrack.

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

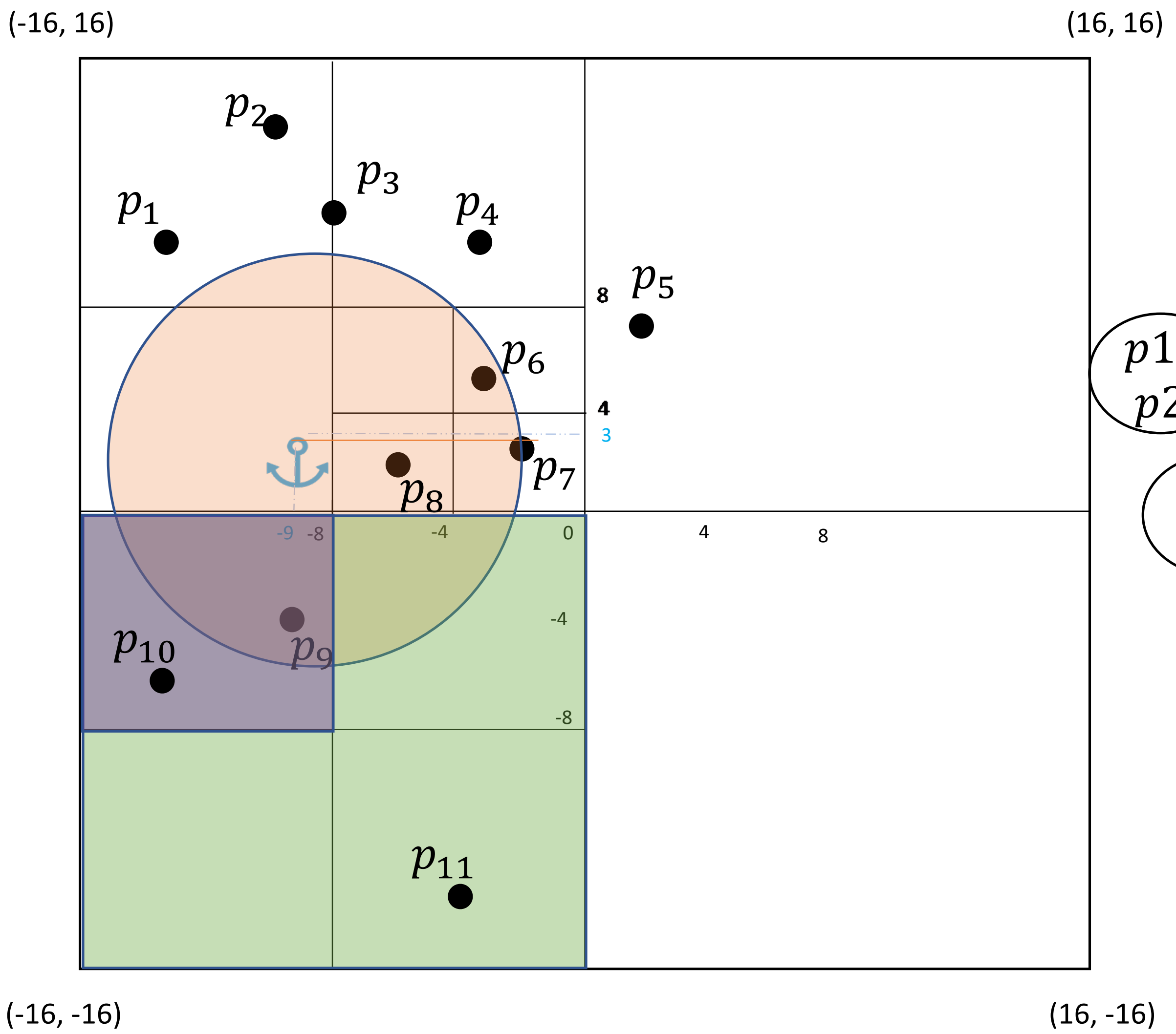


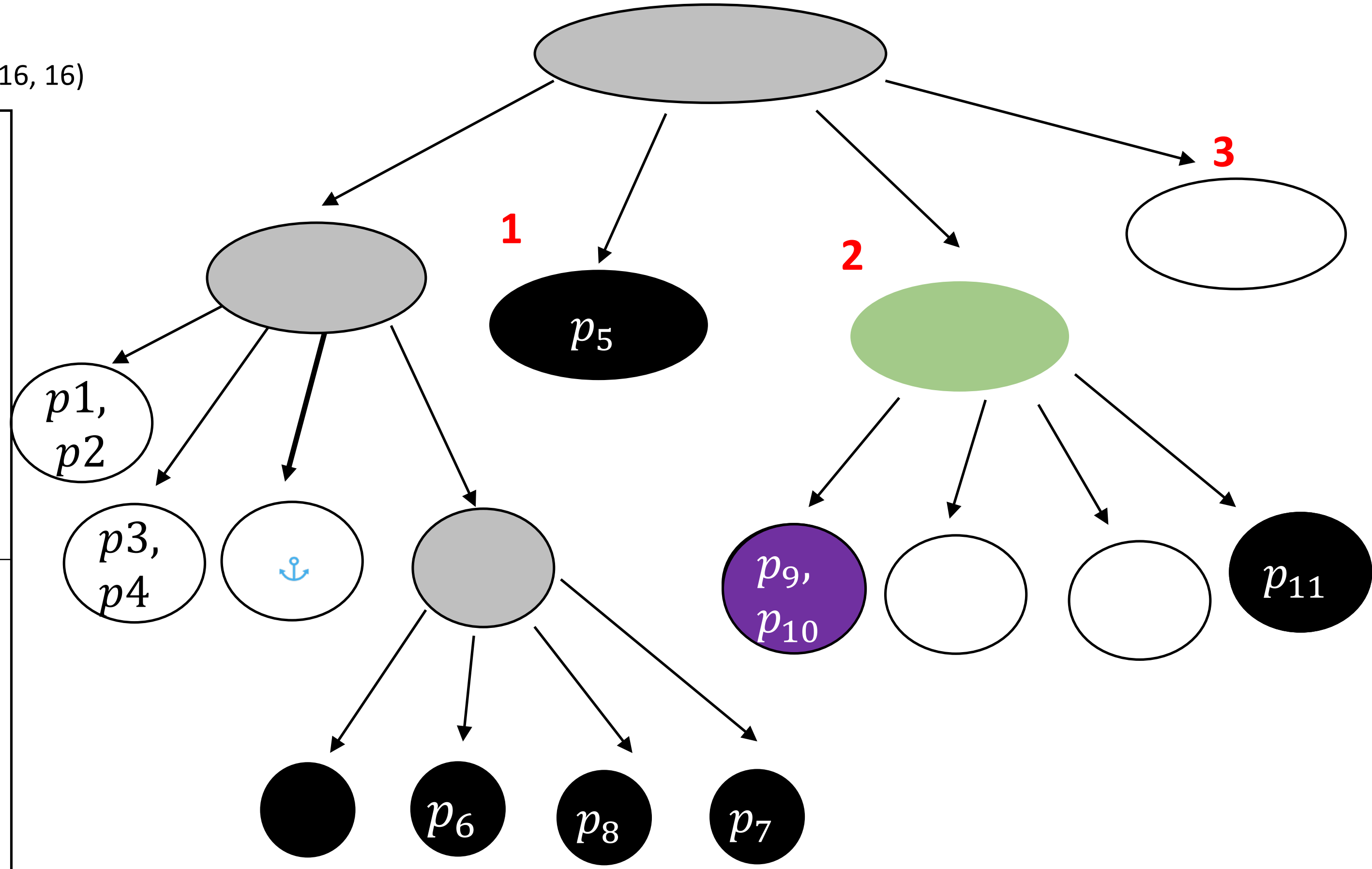


- **Non-zero intersection** between currently spanned quadrant and range!
- Also, gray node

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

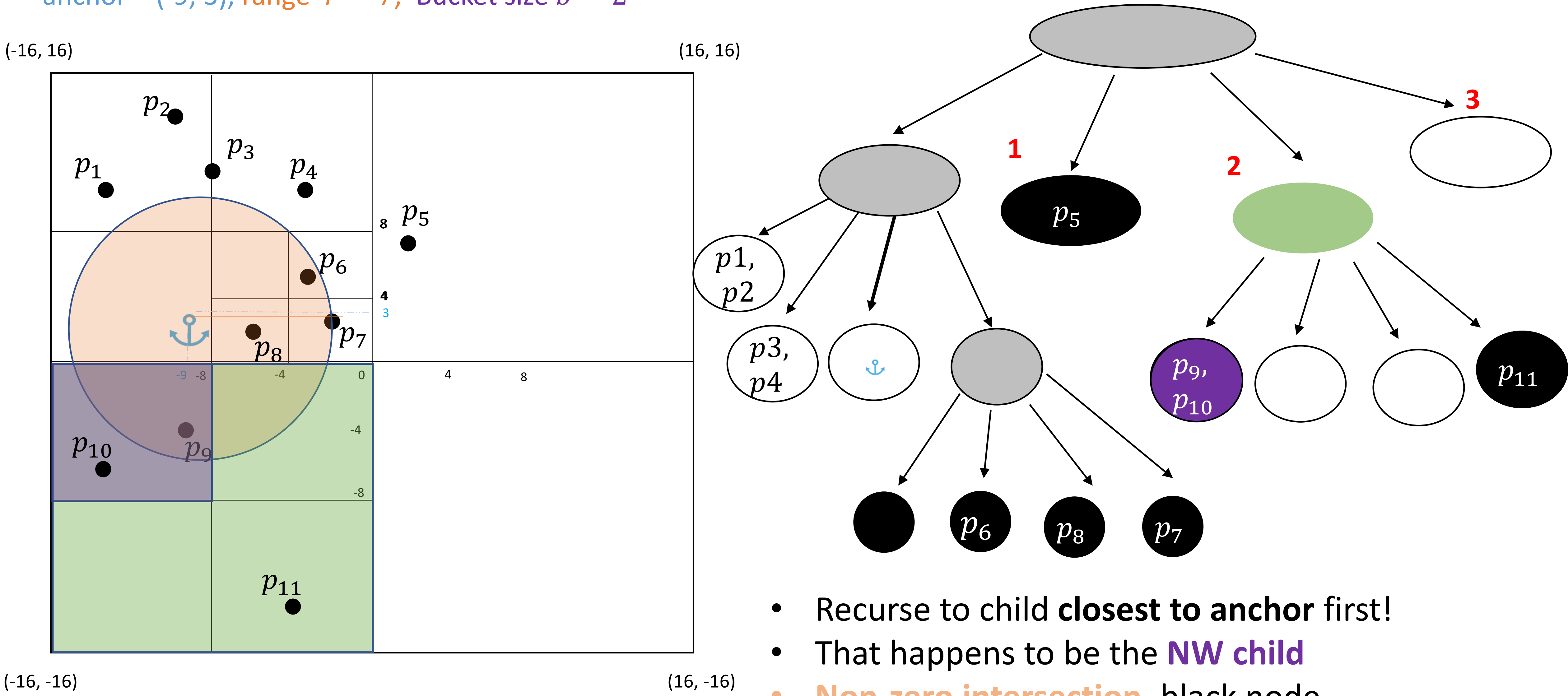




- Recurse to child **closest to anchor** first!
- That happens to be the **NW child**

Example of range query in PR-QuadTree

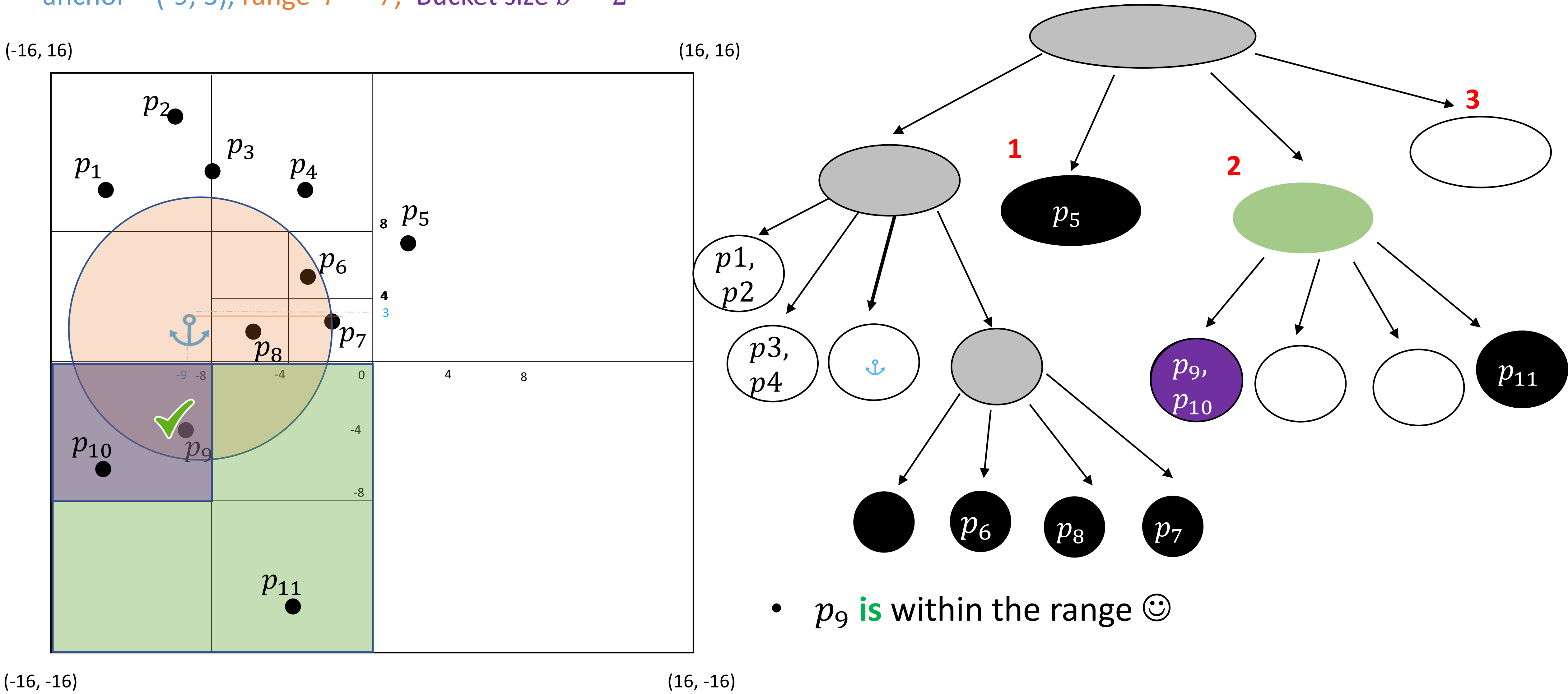
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



- Recurse to child **closest to anchor** first!
- That happens to be the **NW child**
- **Non-zero intersection**, black node

Example of range query in PR-QuadTree

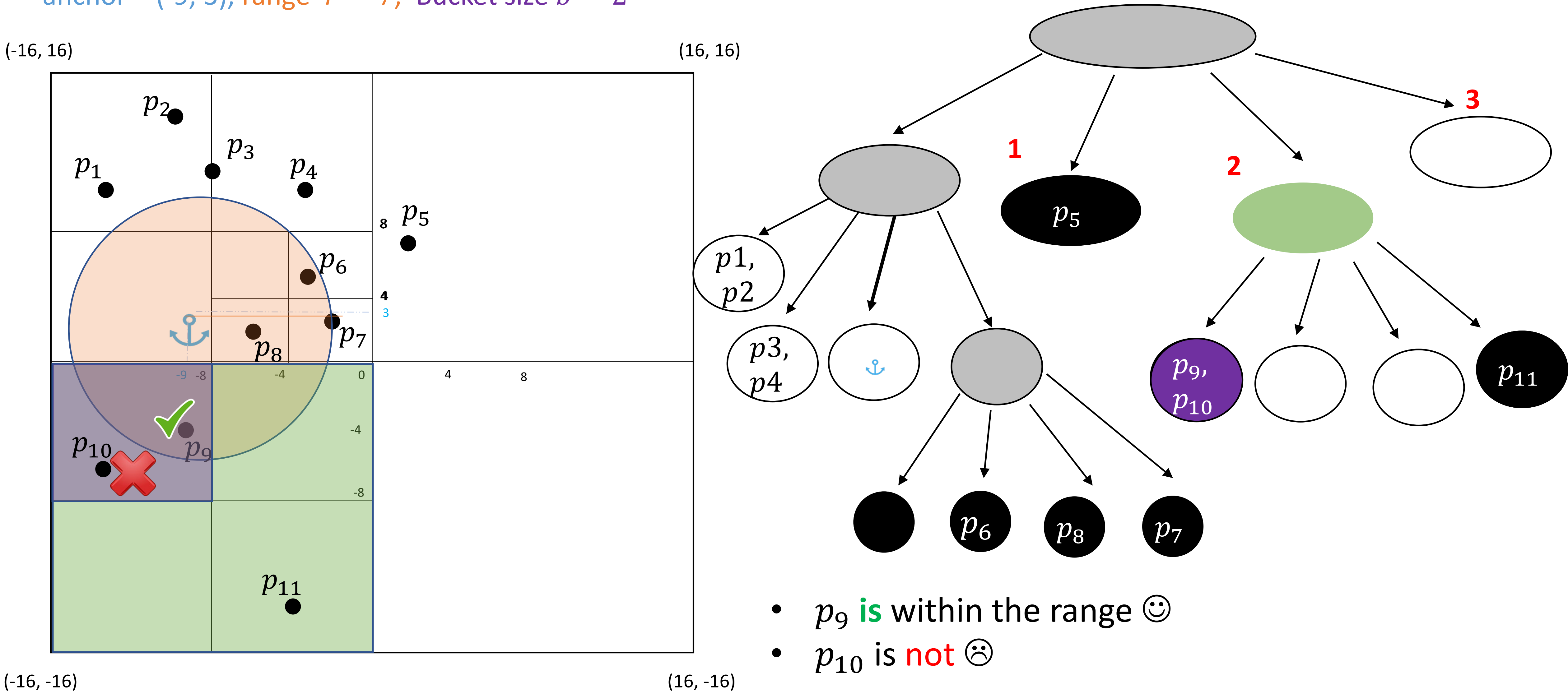
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$



- p_9 **is** within the range ☺

Example of range query in PR-QuadTree

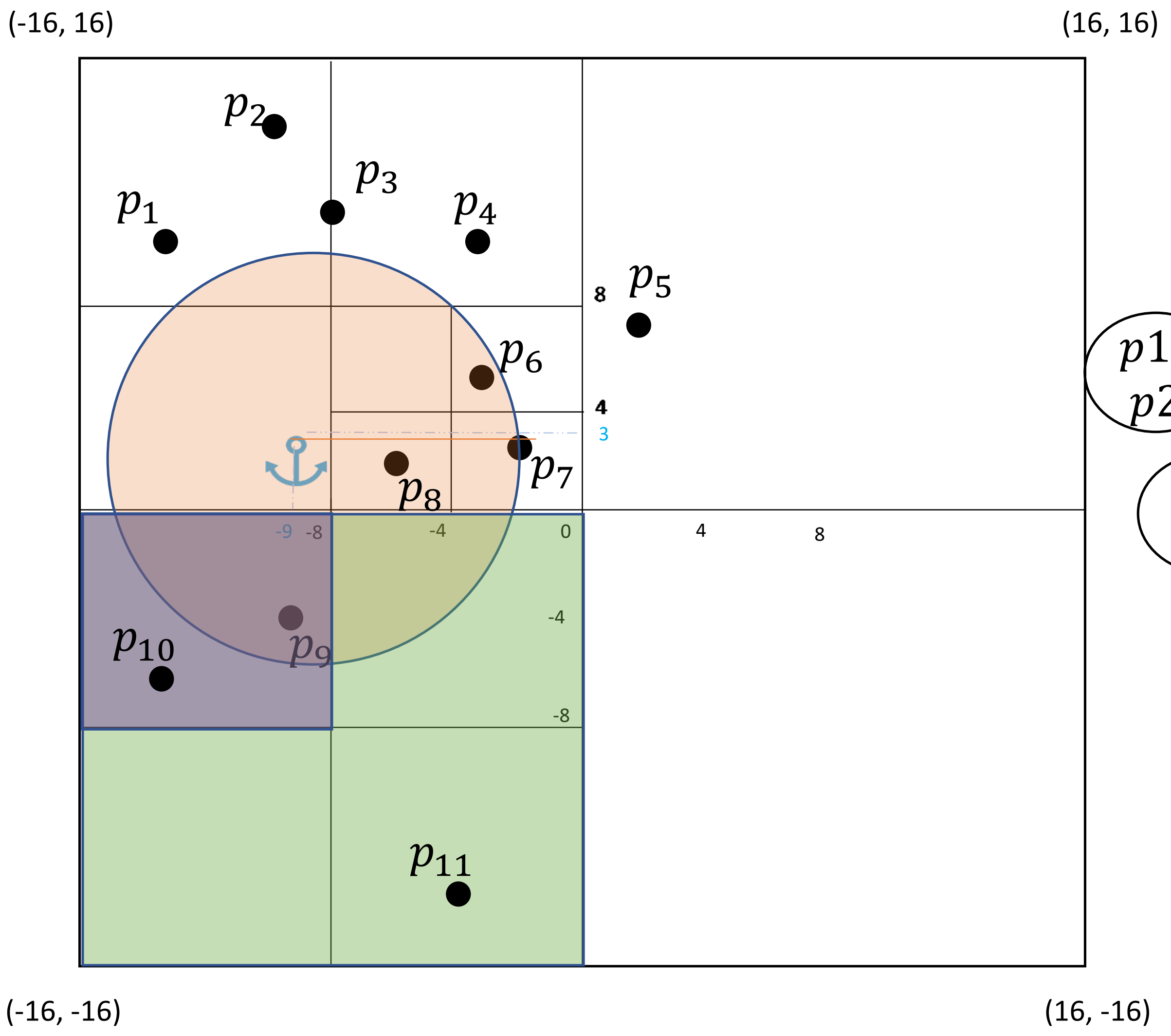
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

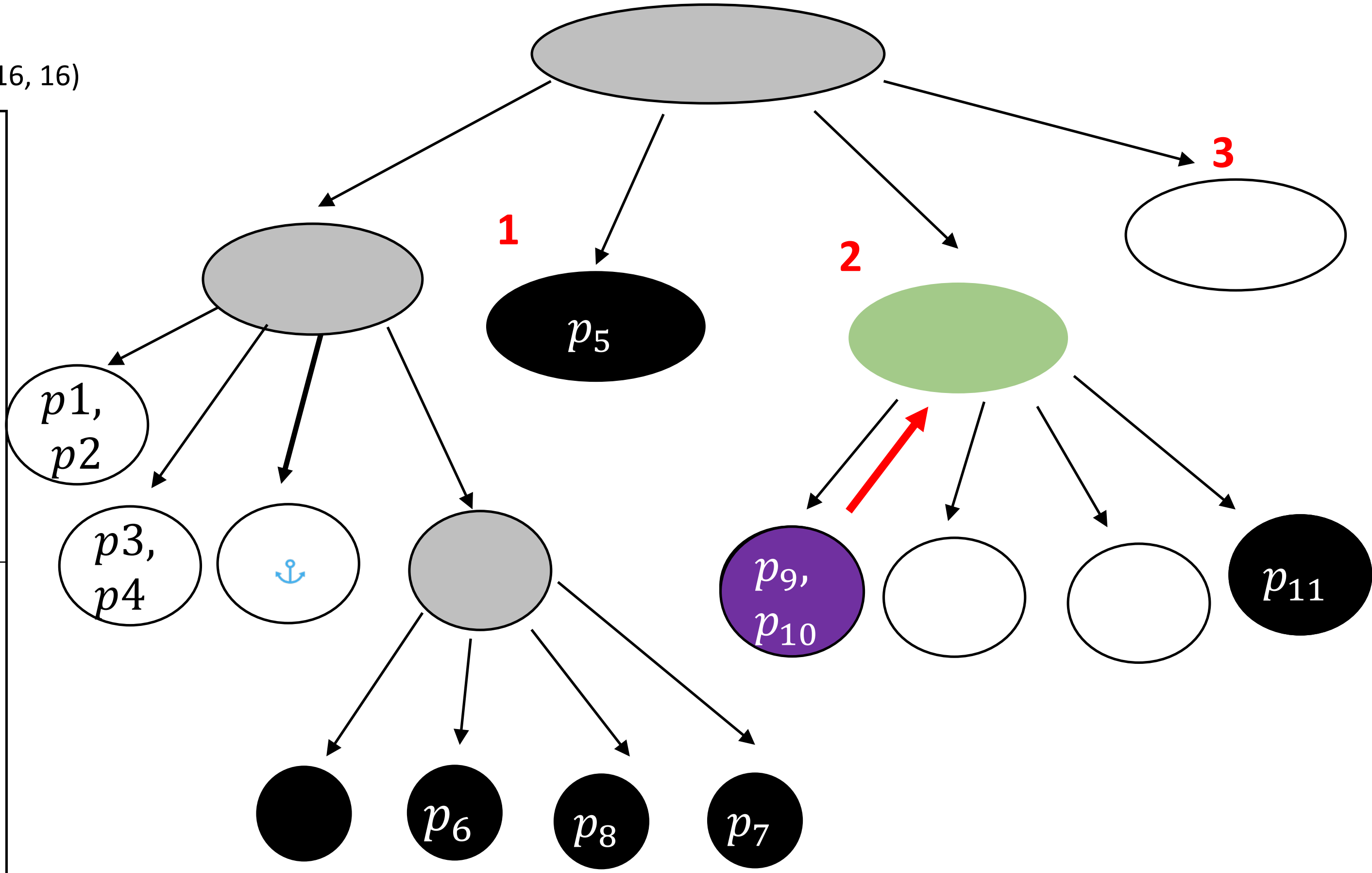


- p_9 **is** within the range ☺
- p_{10} is not ☹

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

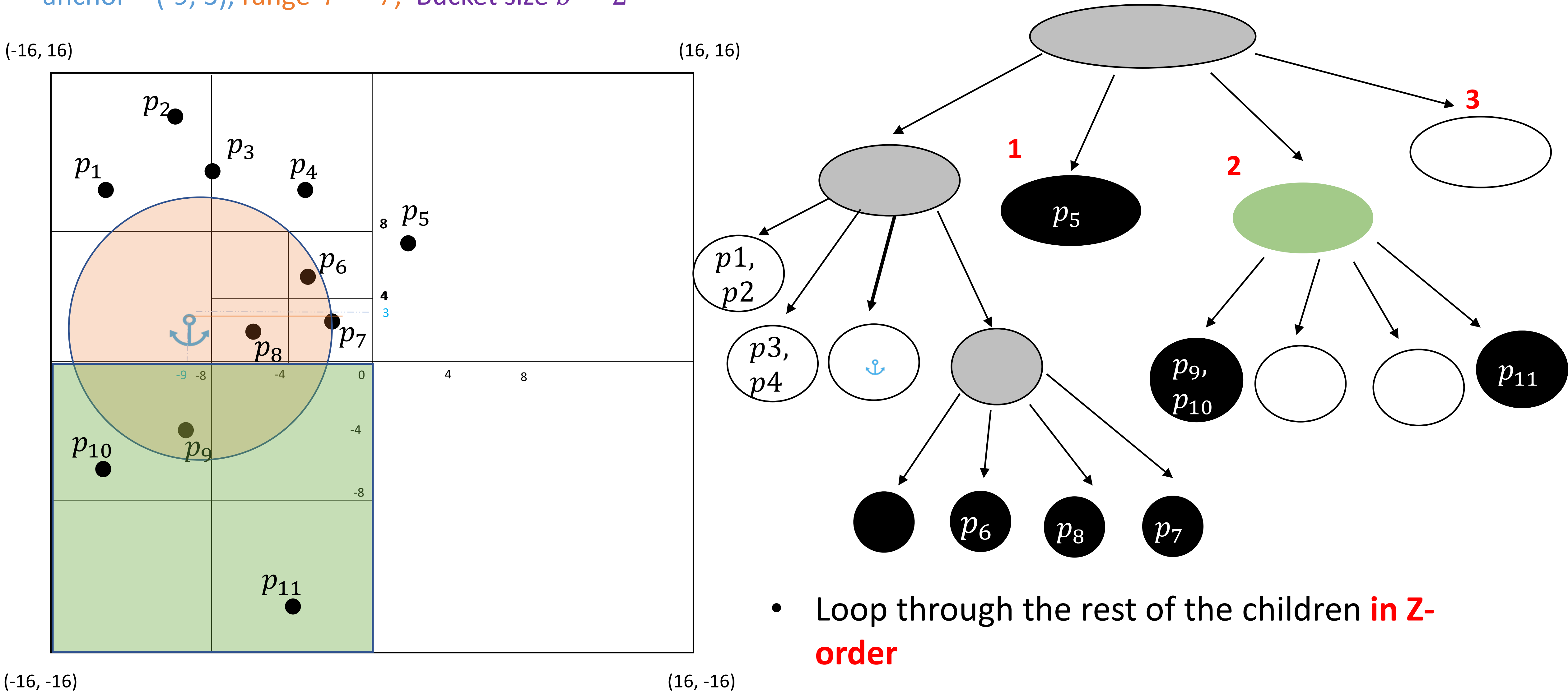




- **Backtrack**

Example of range query in PR-QuadTree

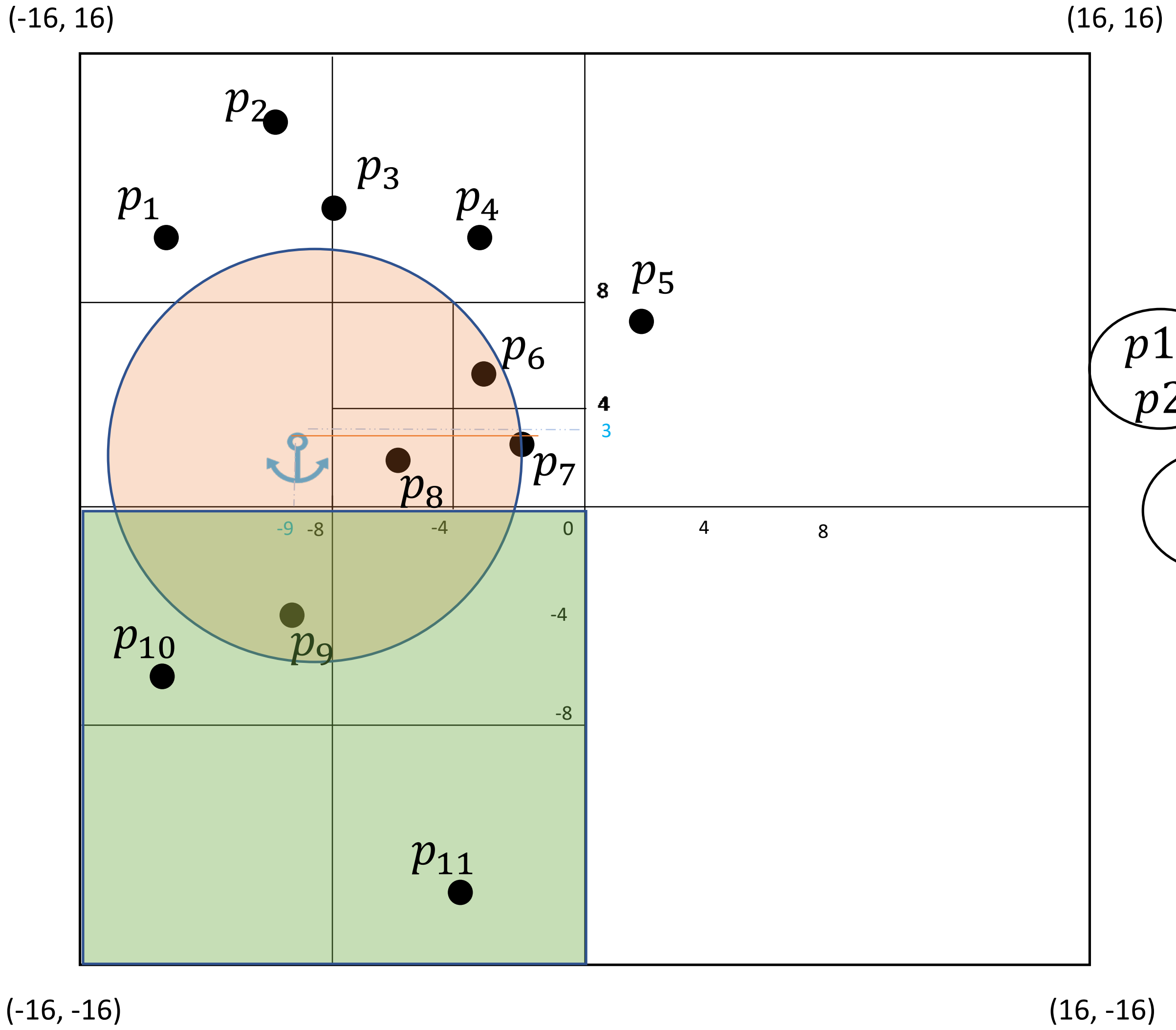
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

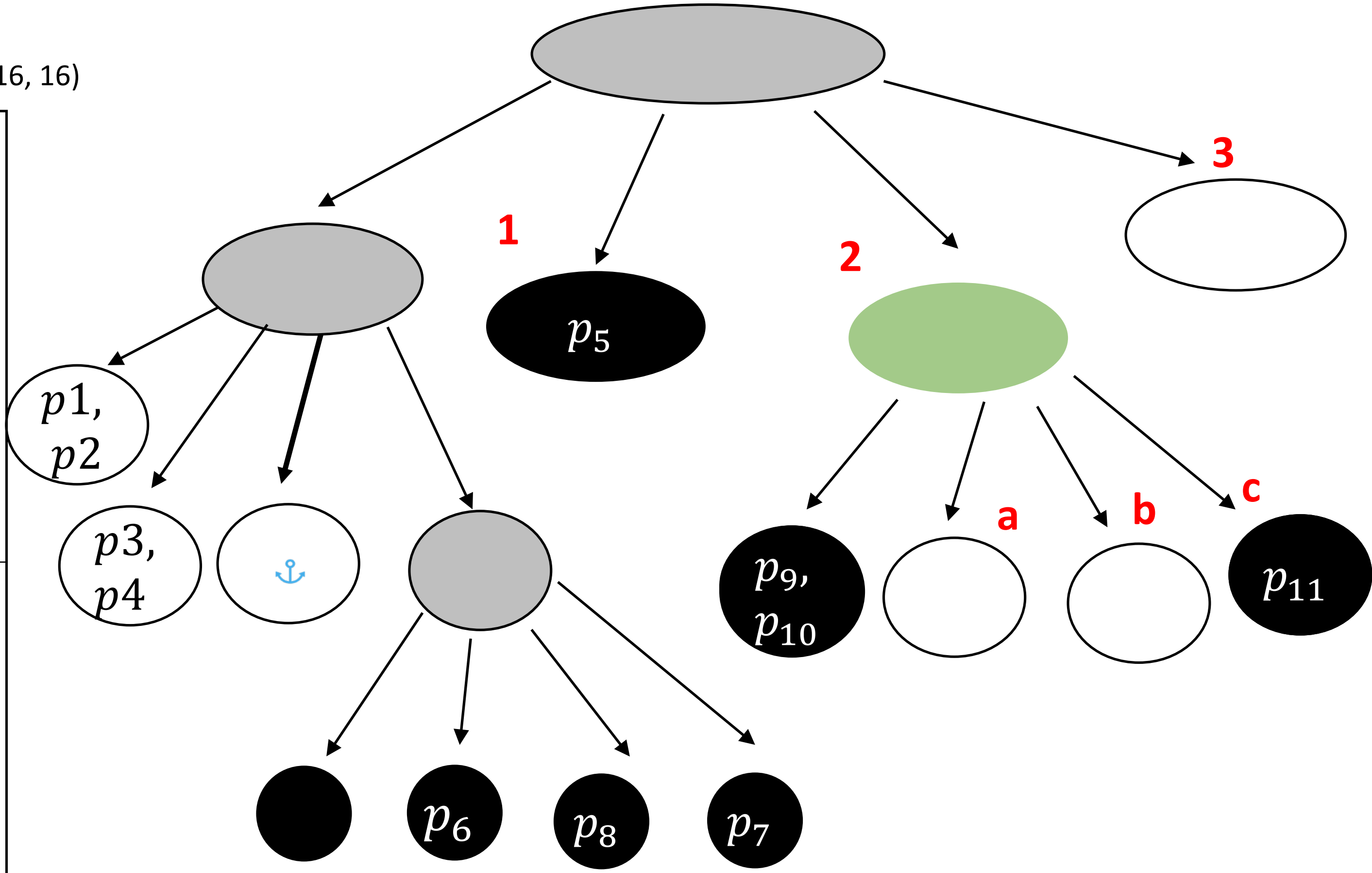


- Loop through the rest of the children **in Z-order**

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

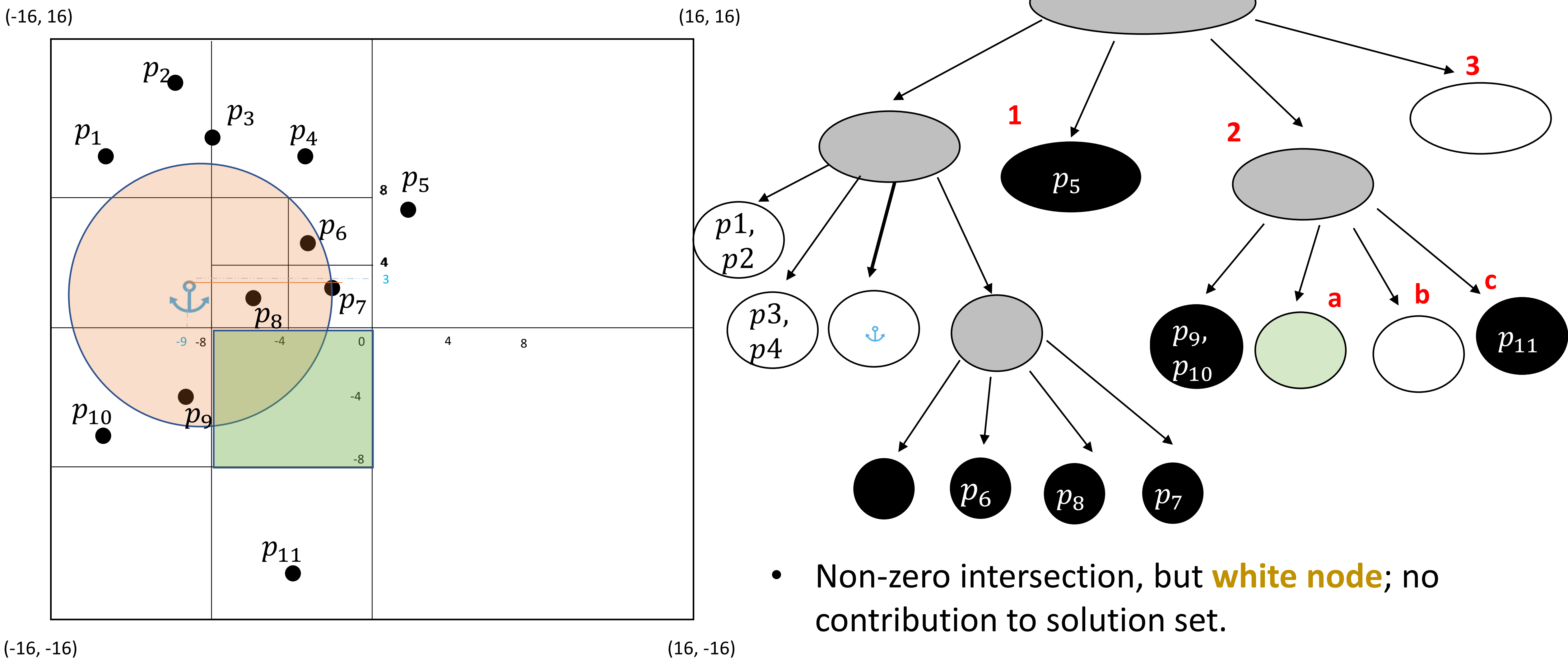




- Loop through the rest of the children **in Z-order**

Example of range query in PR-QuadTree

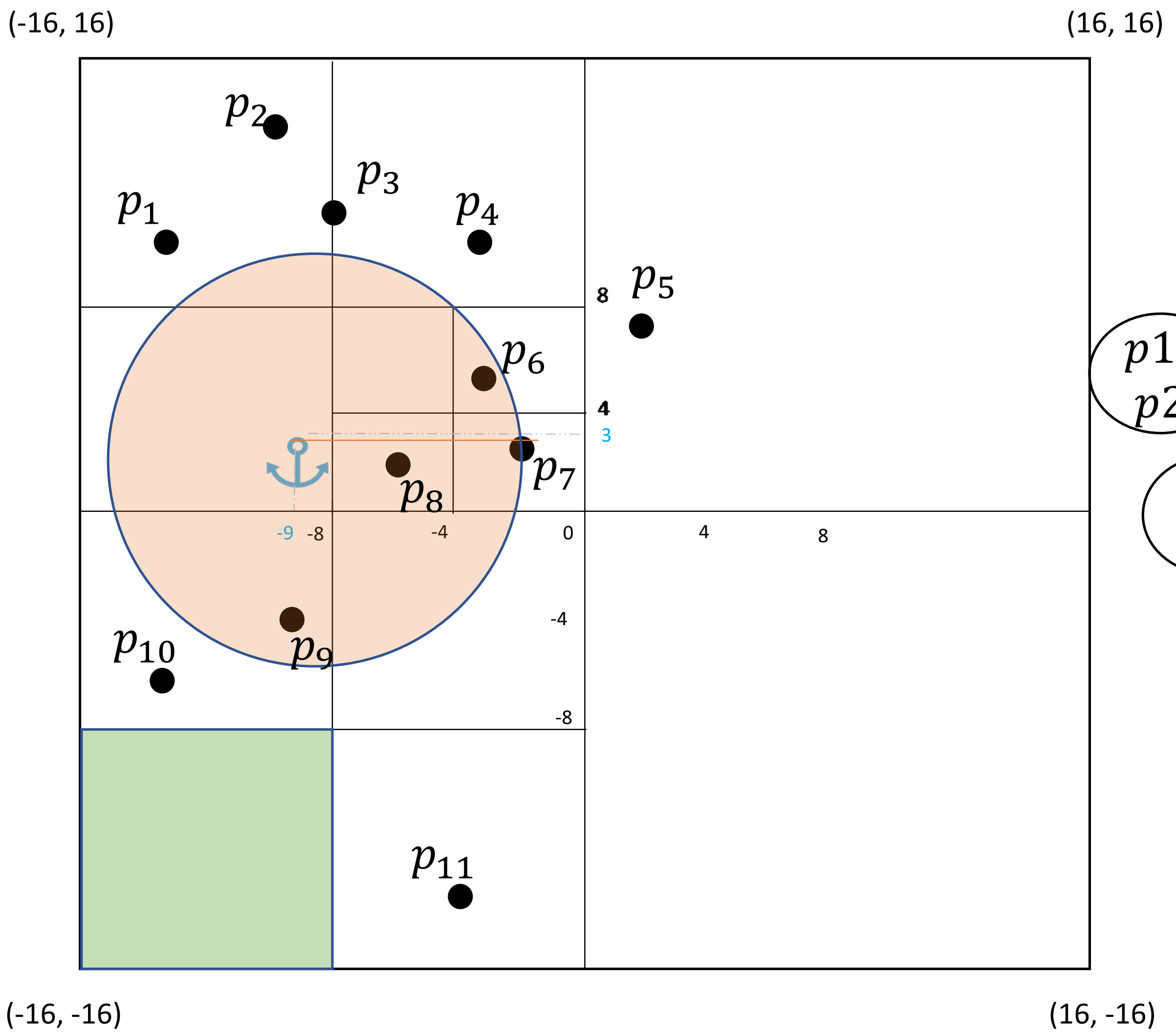
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

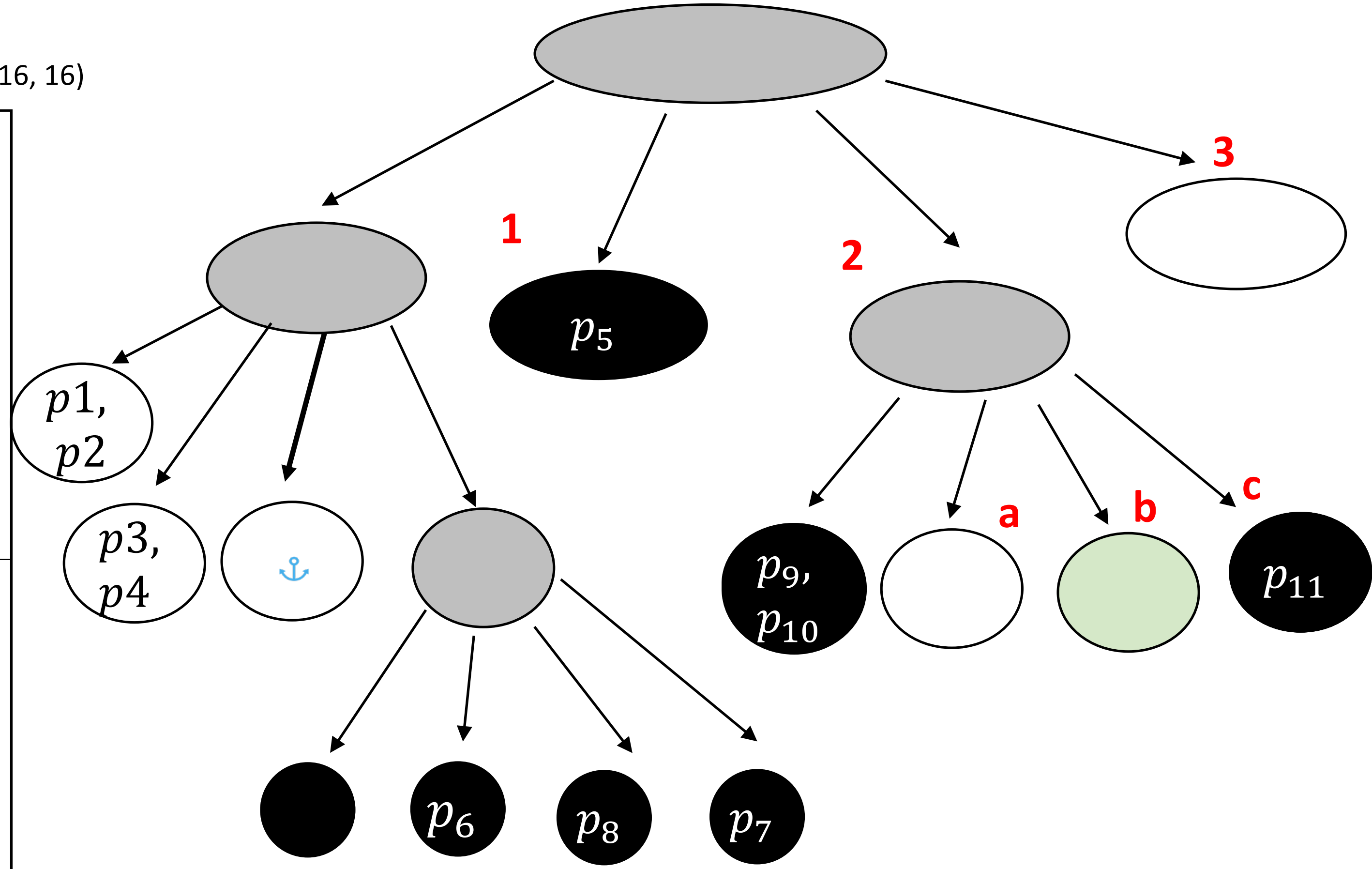


- Non-zero intersection, but **white node**; no contribution to solution set.

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

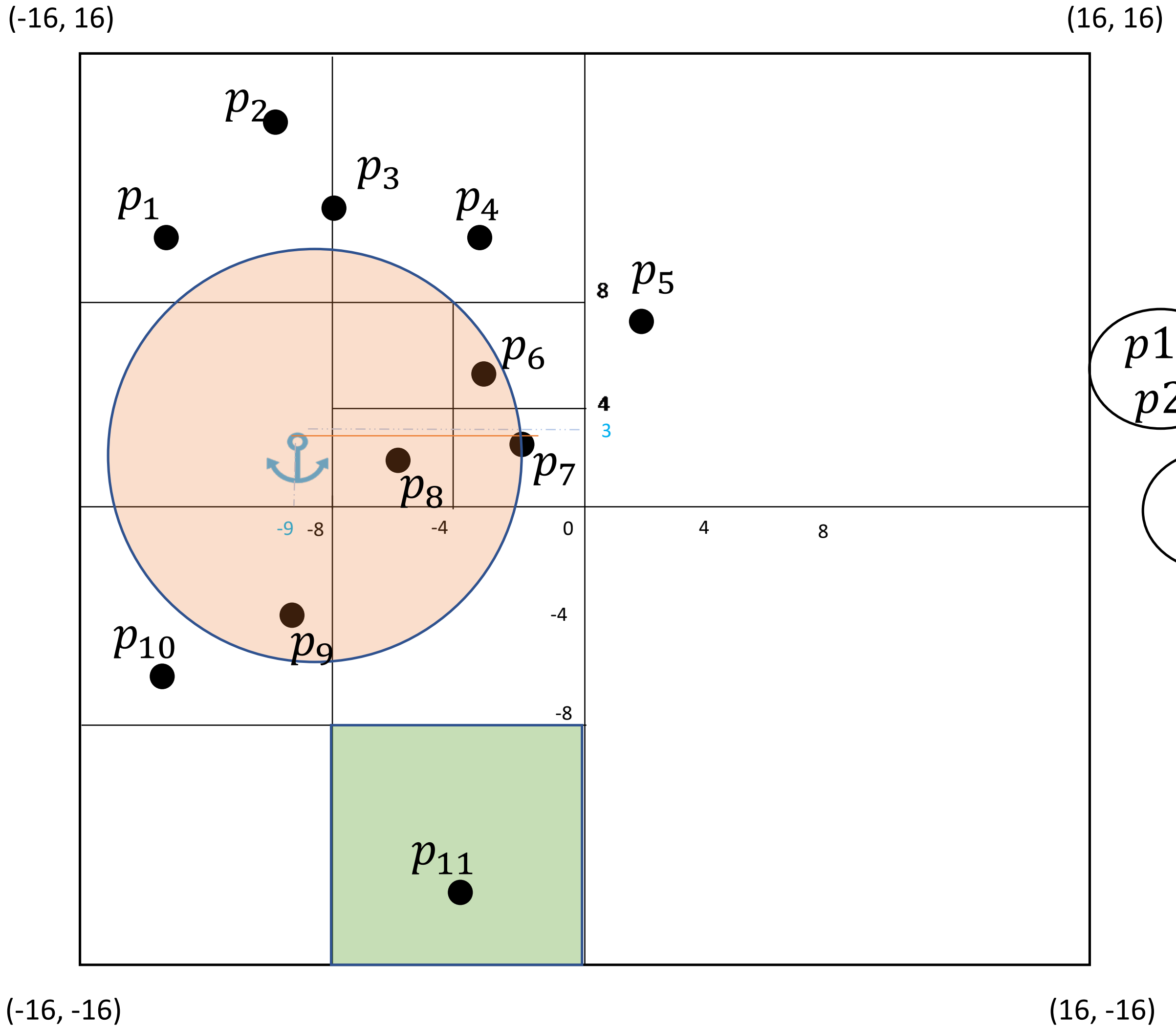


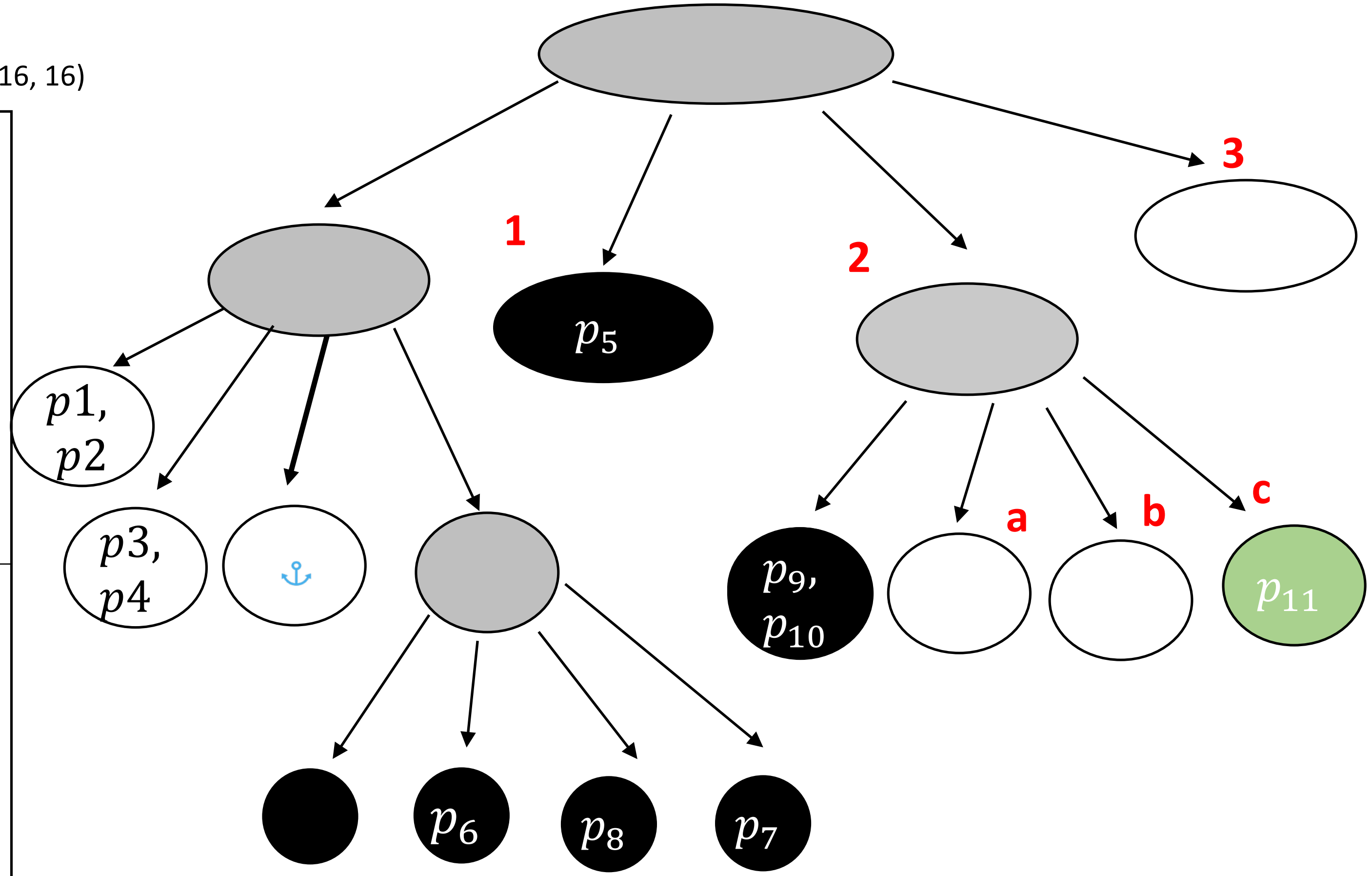


- **Zero** intersection **and white node**; no contribution to solution set.

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

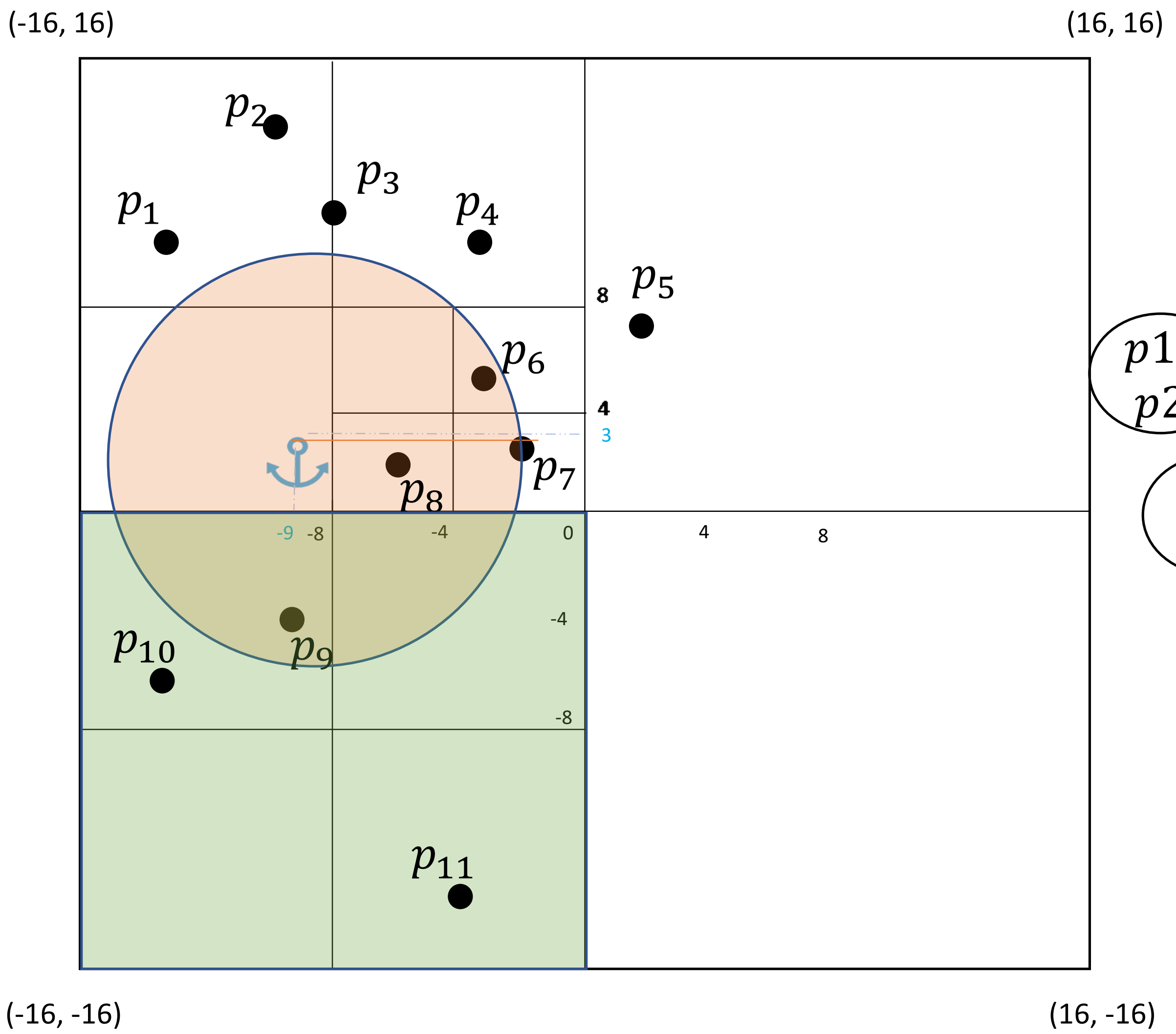


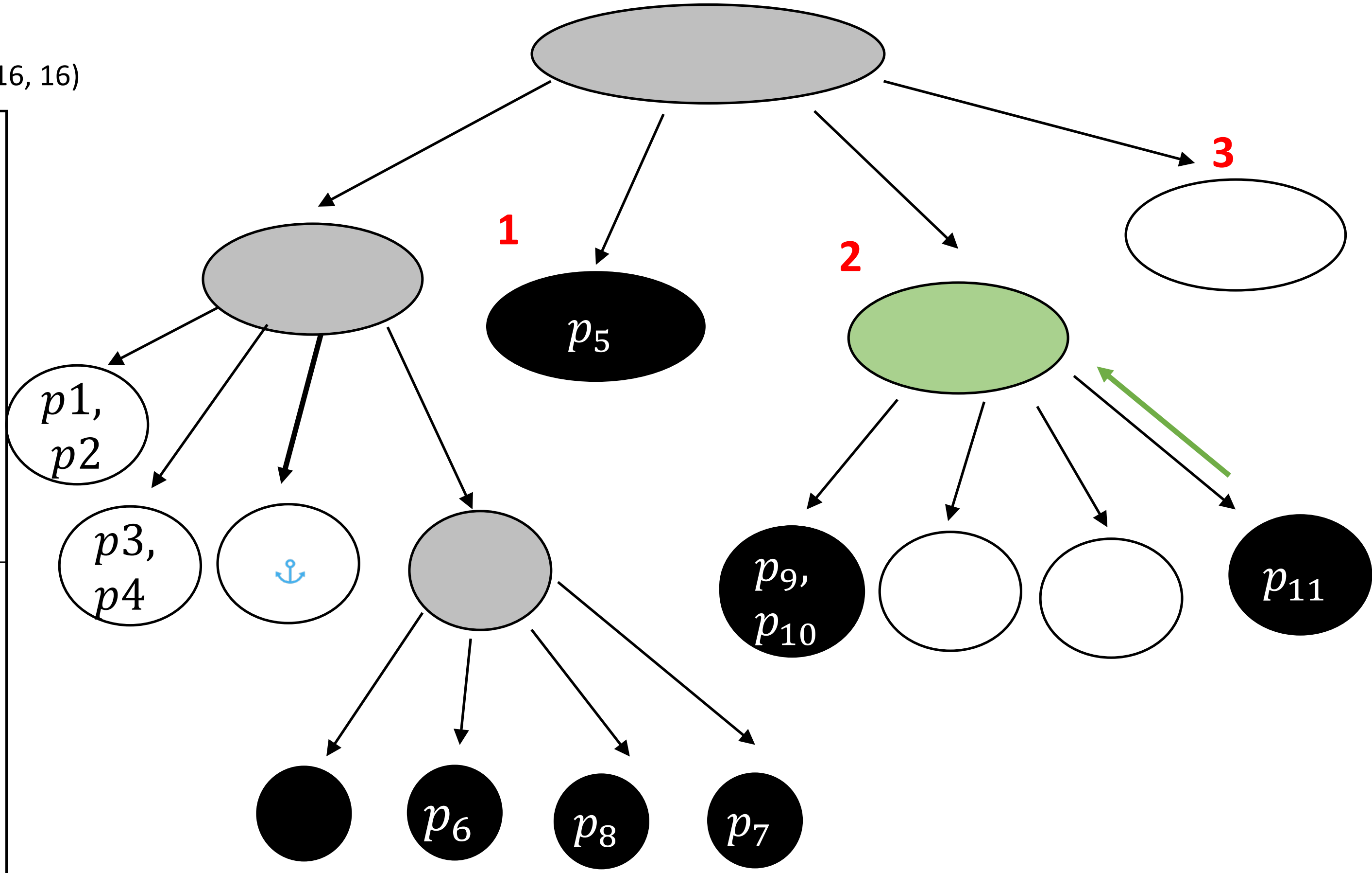


- Black node, but zero intersection ☹
- **Can't contribute to solution set ☹**

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

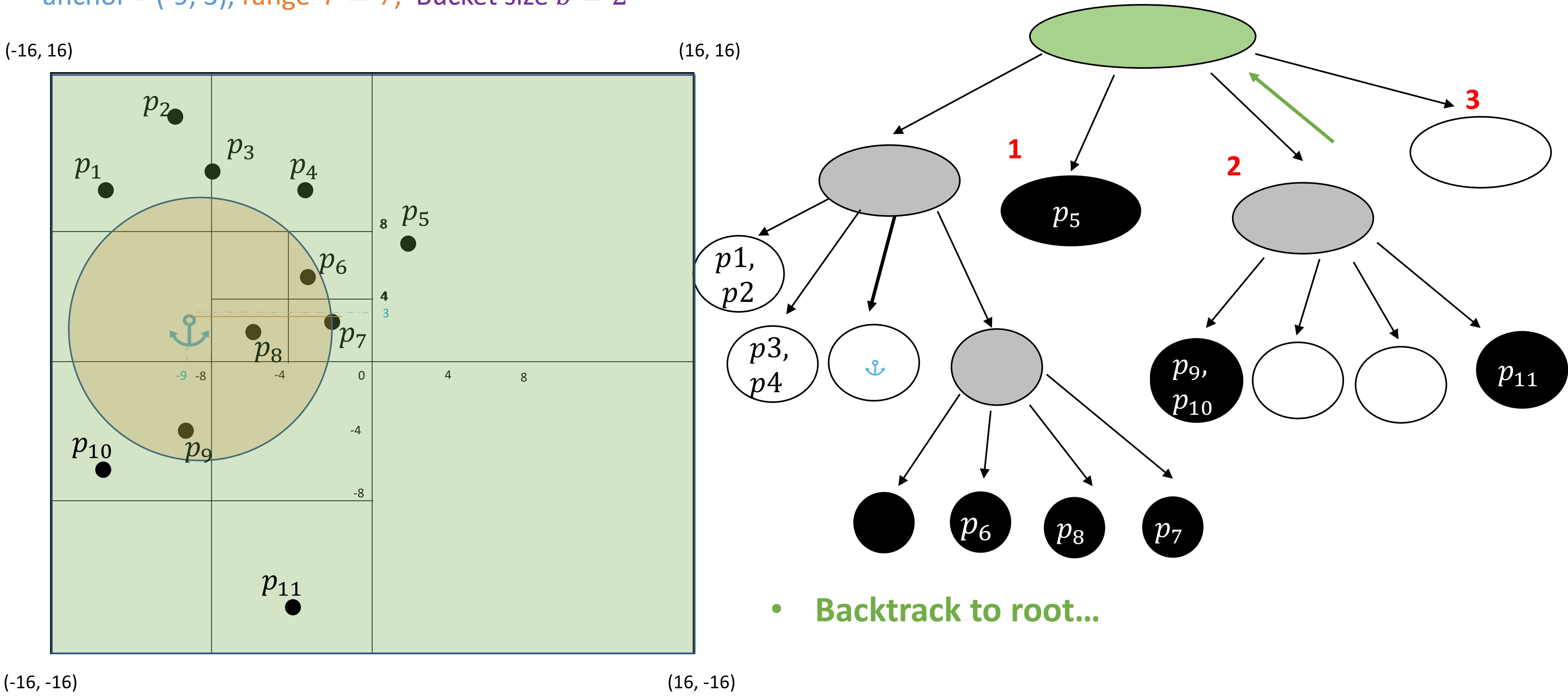




- **Backtrack to root…**

Example of range query in PR-QuadTree

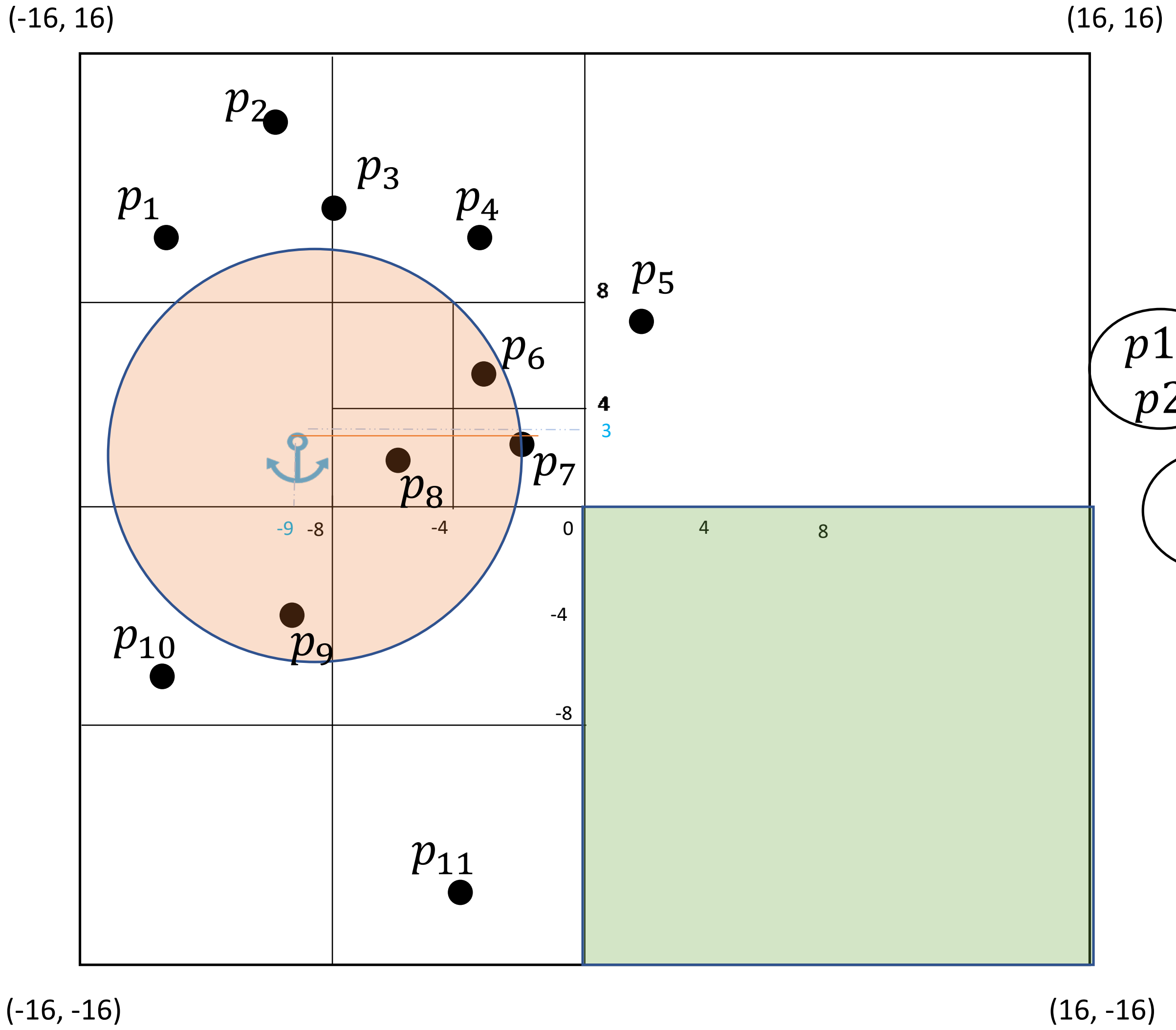
anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

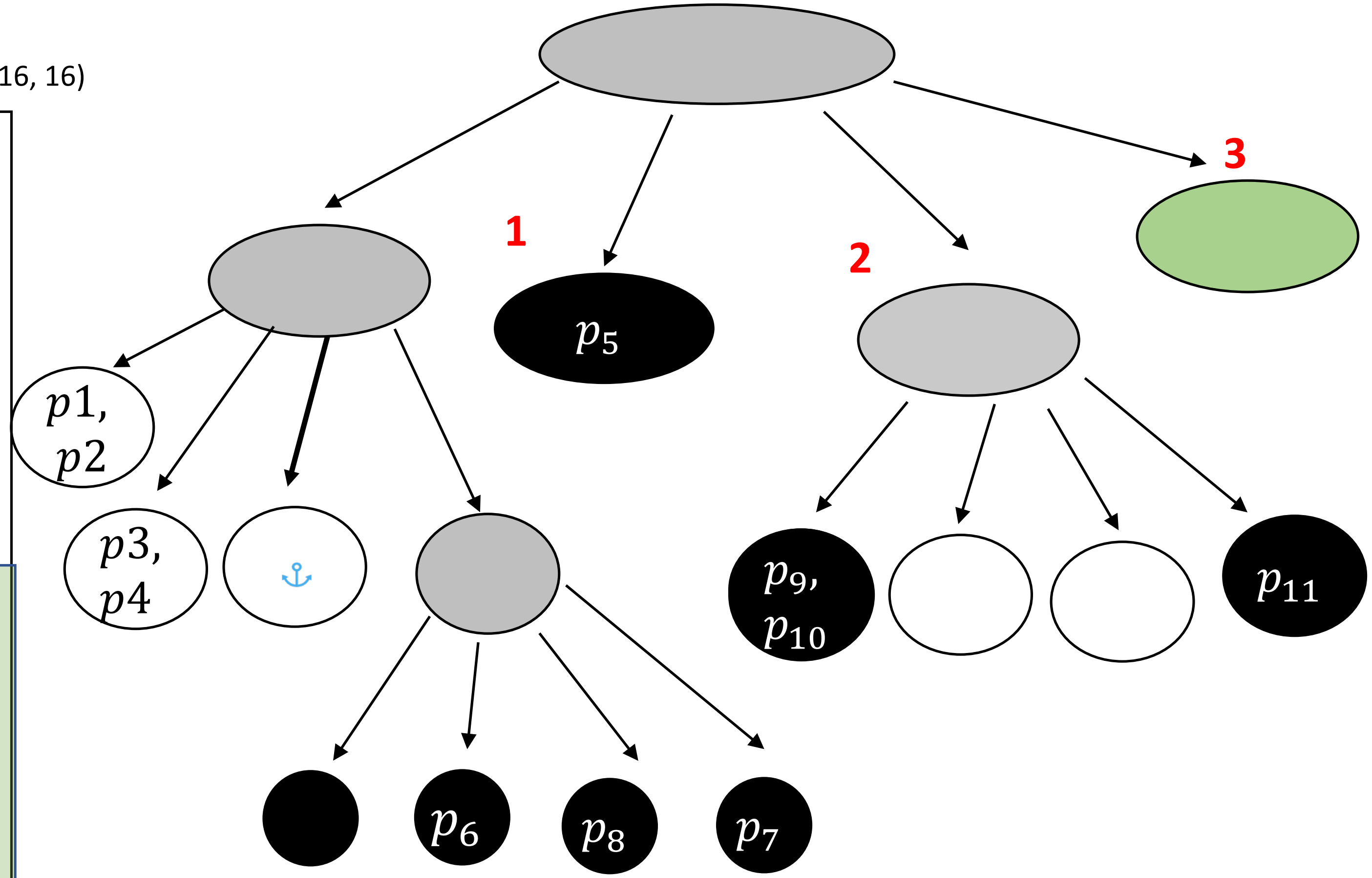


- **Backtrack to root…**

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

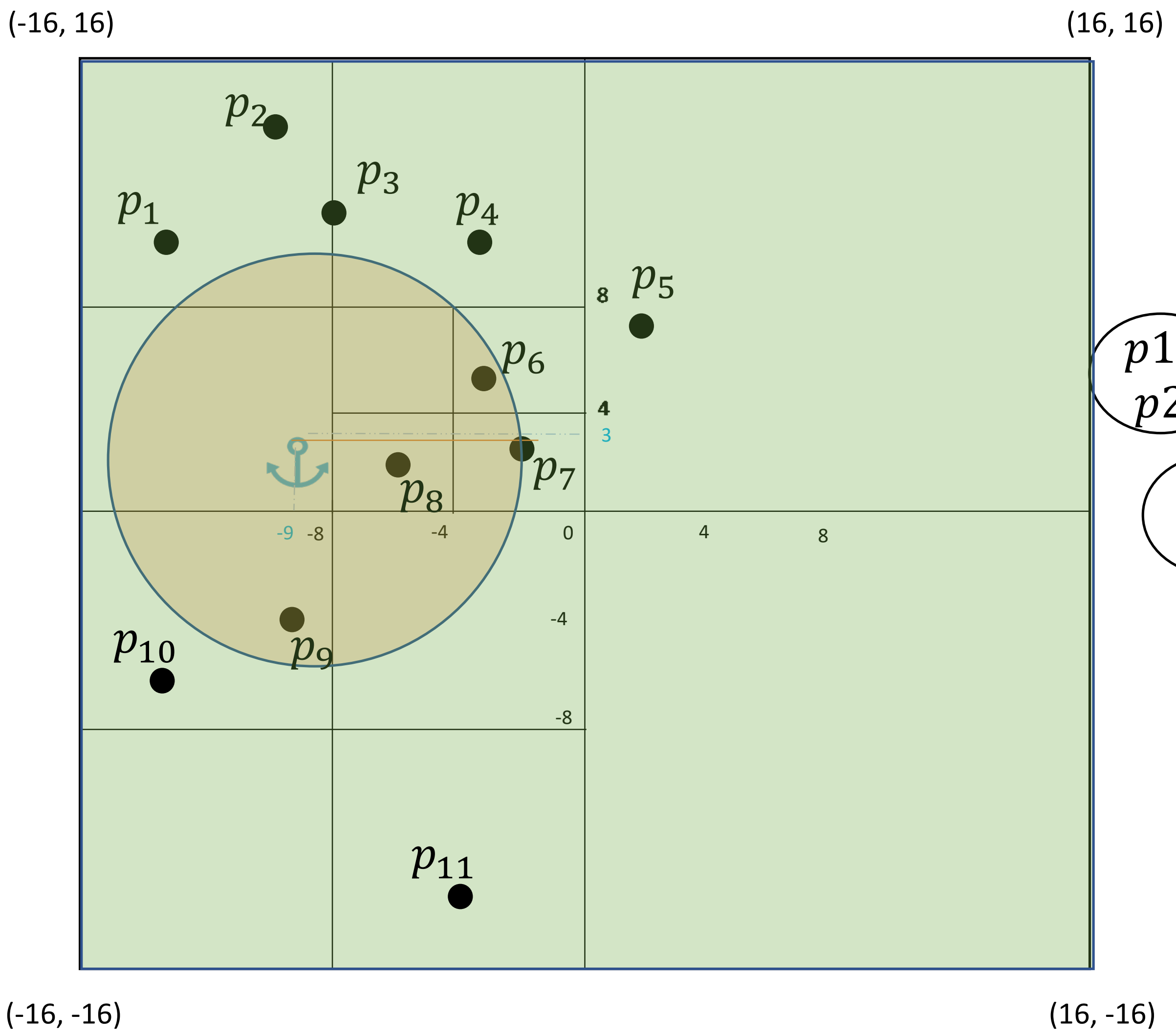


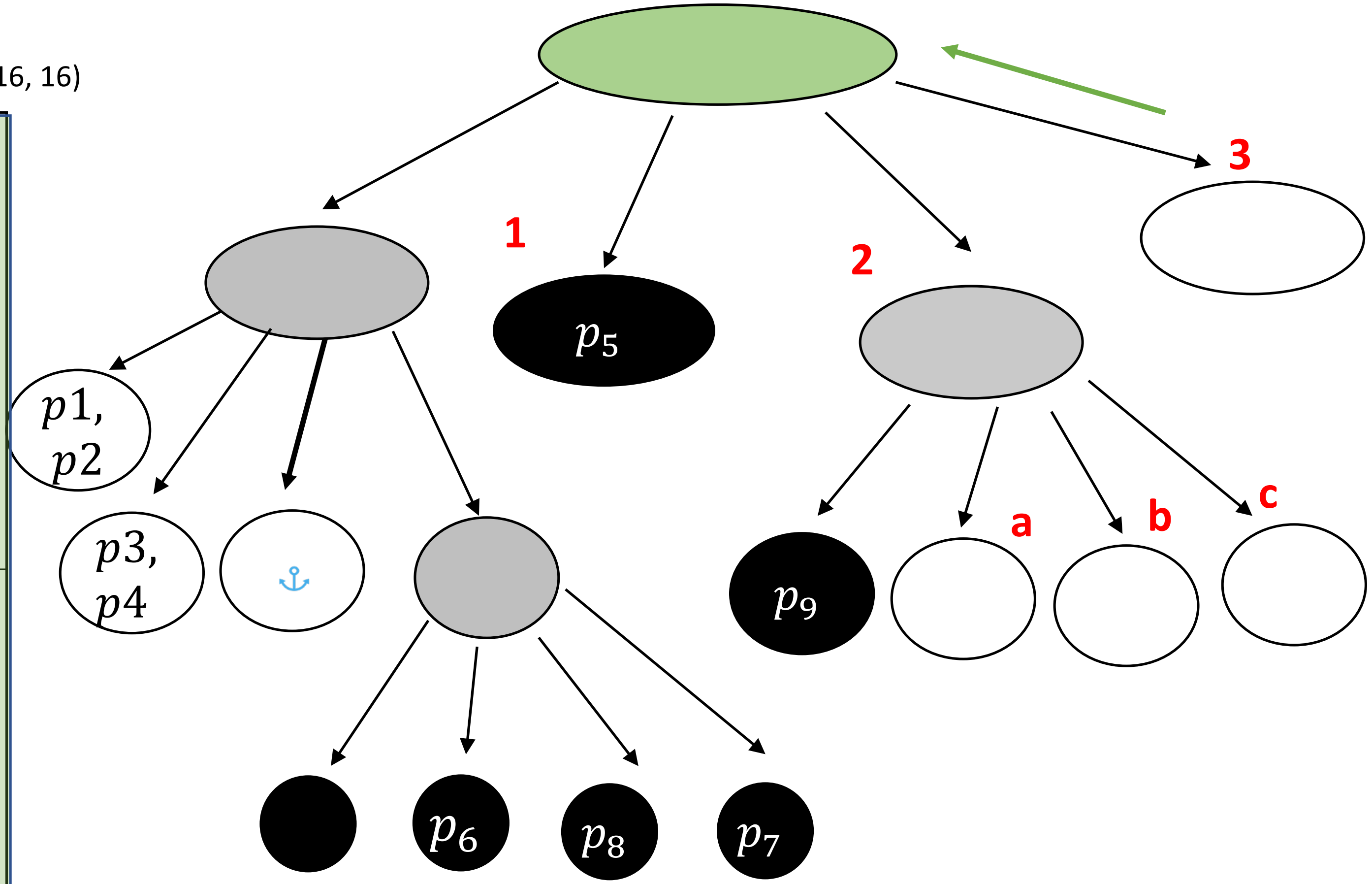


- White node **and** beyond range: impossible to contribute to solution set!

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

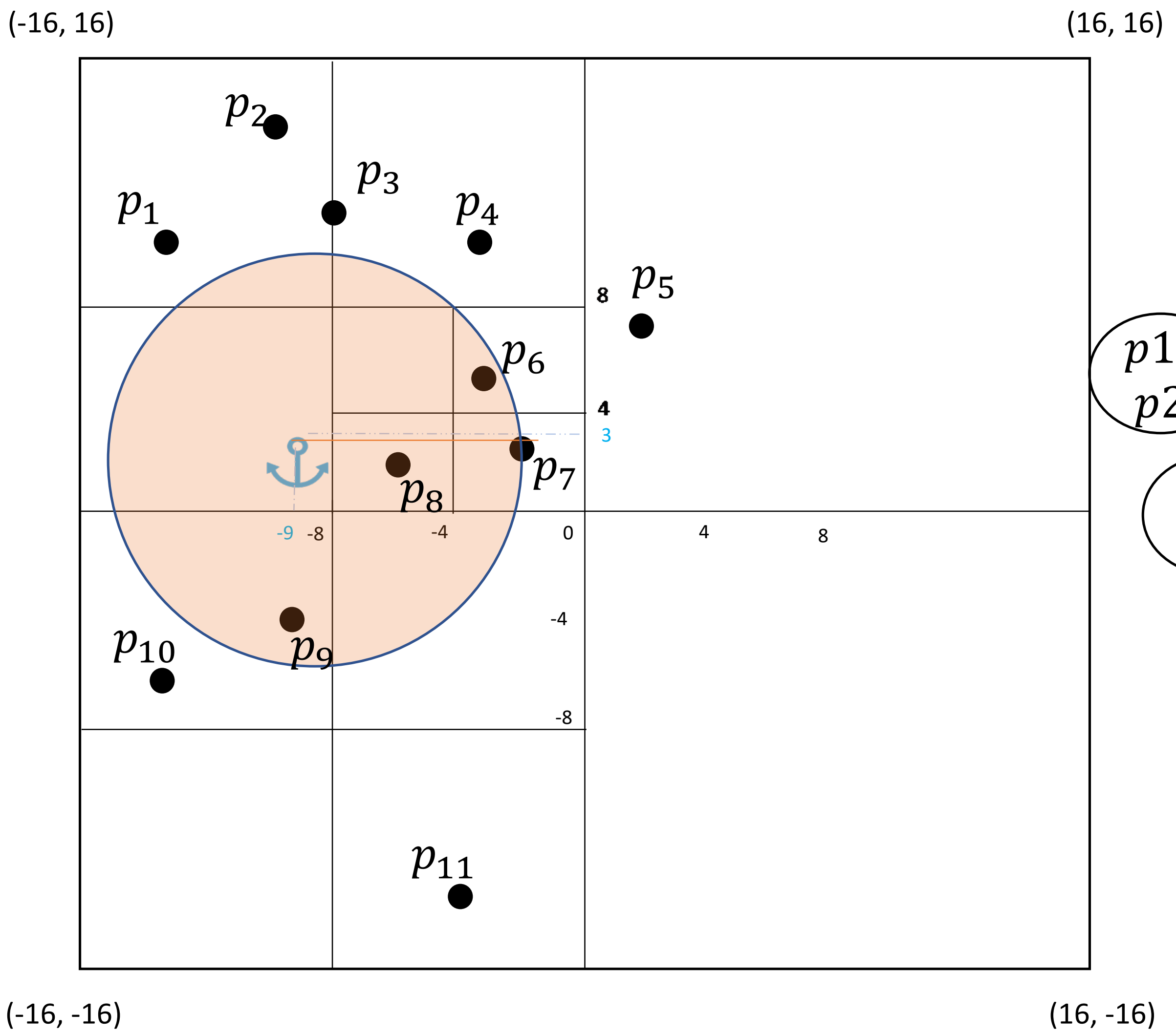


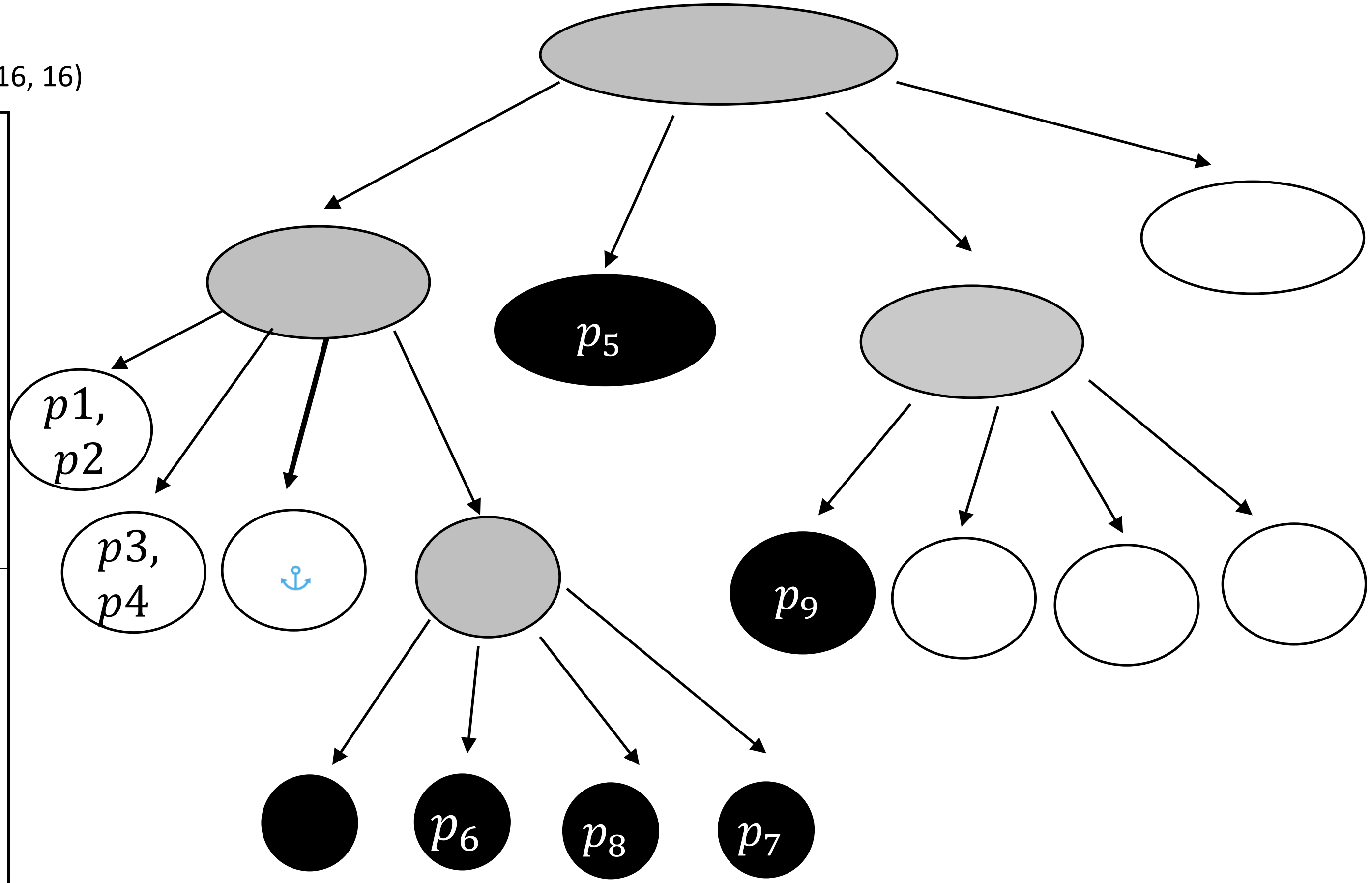


- White node **and** beyond range: impossible to contribute to solution set!
- Backtrack….

Example of range query in PR-QuadTree

anchor = (-9, 3), range $r = 7$, Bucket size $b = 2$

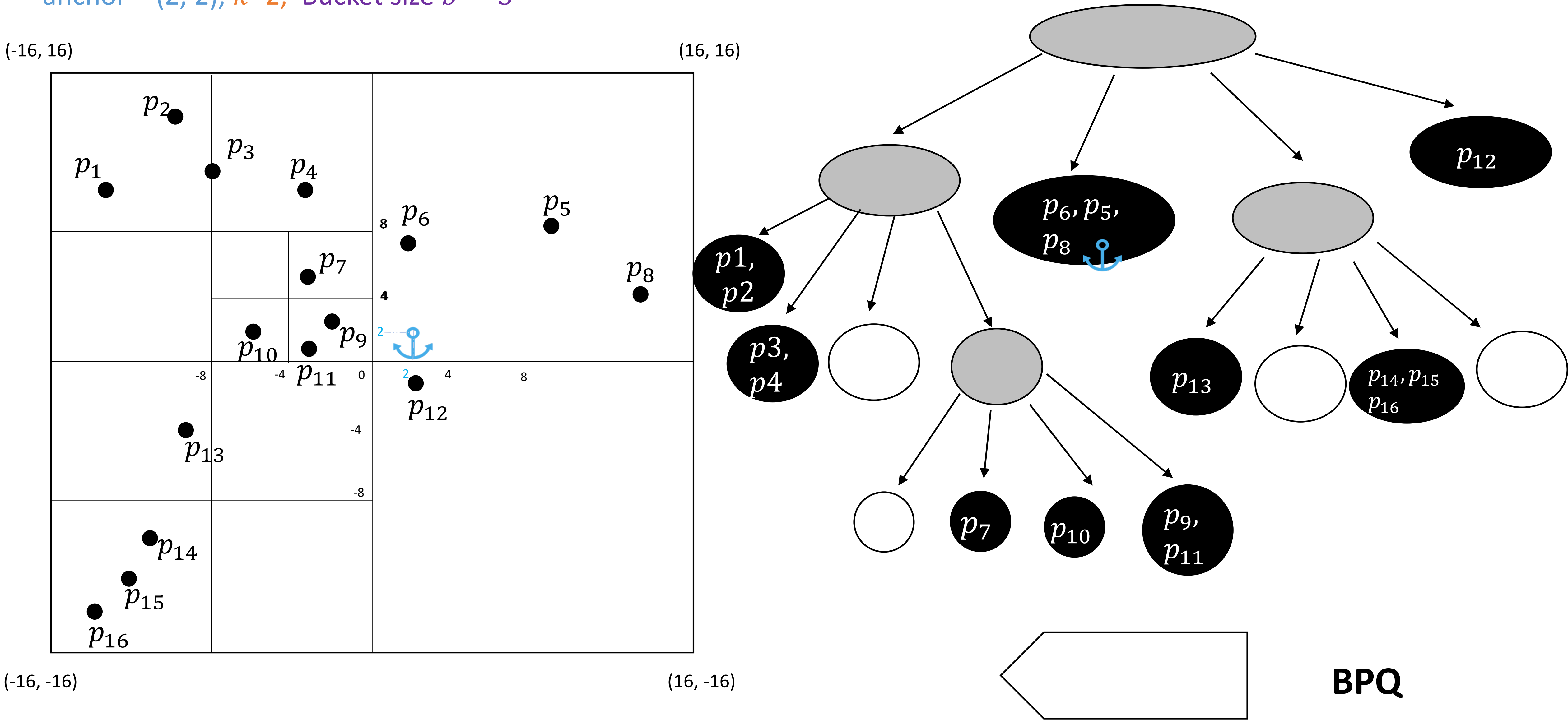




- White node **and** beyond range: impossible to contribute to solution set!
- Backtrack….
- Done! ☺ Solution set: $\{p_6, p_8, p_7, p_9\}$

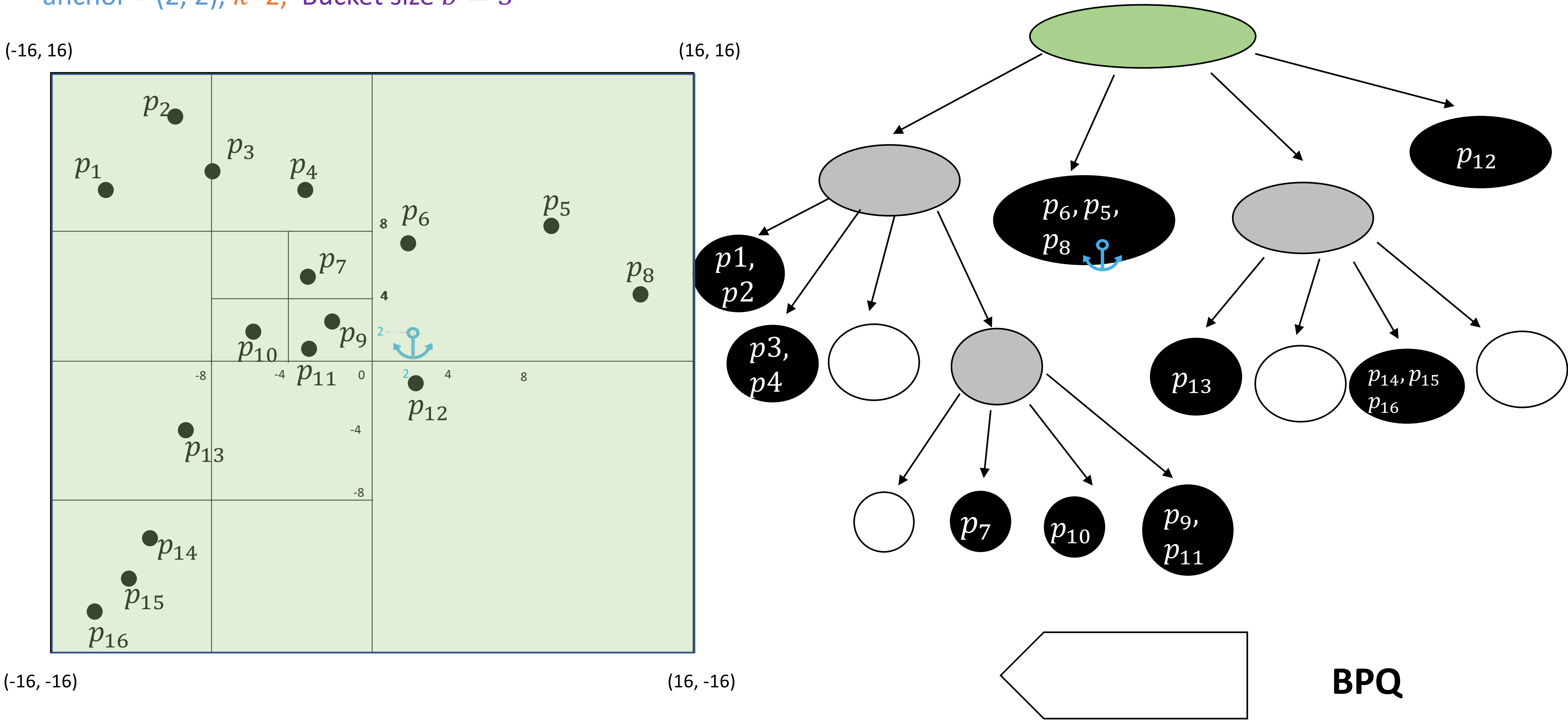
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



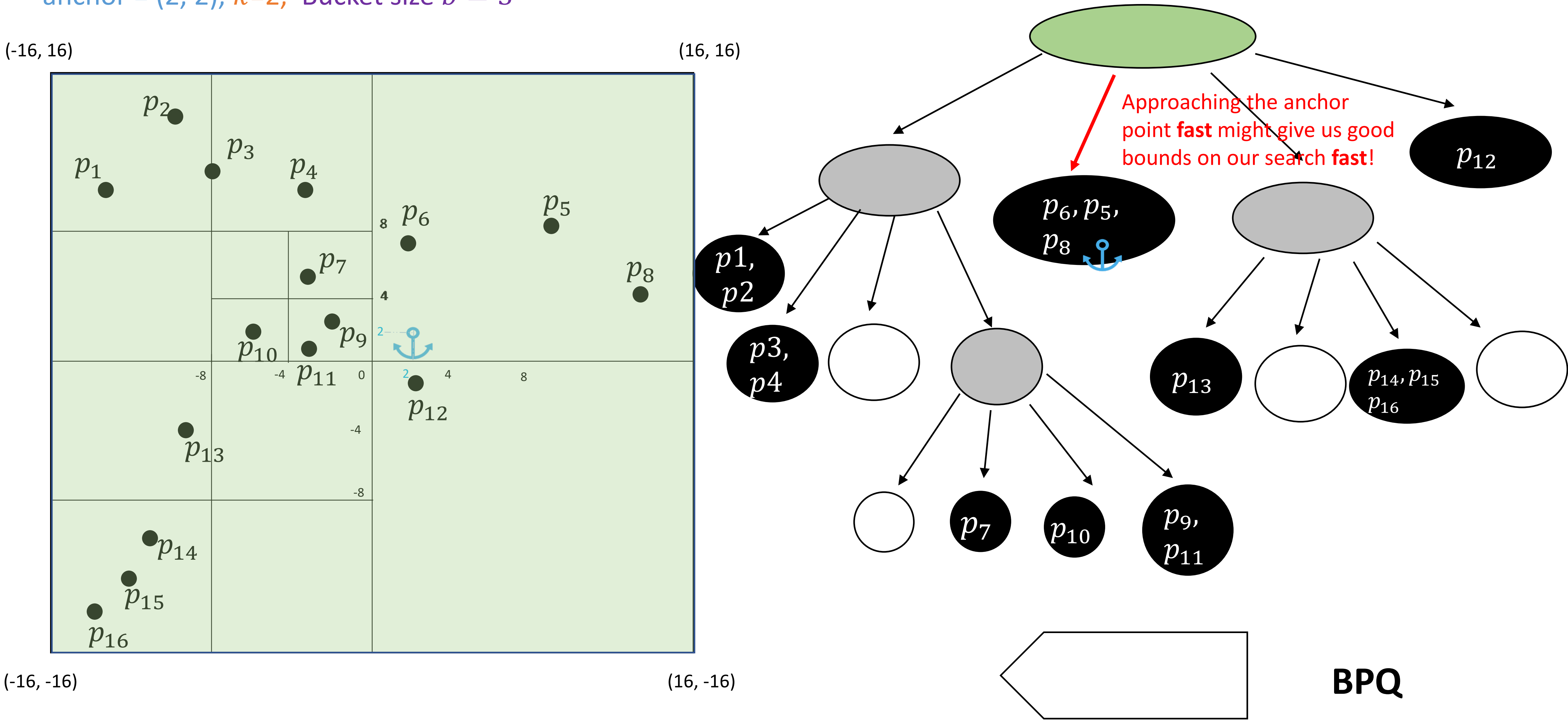
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



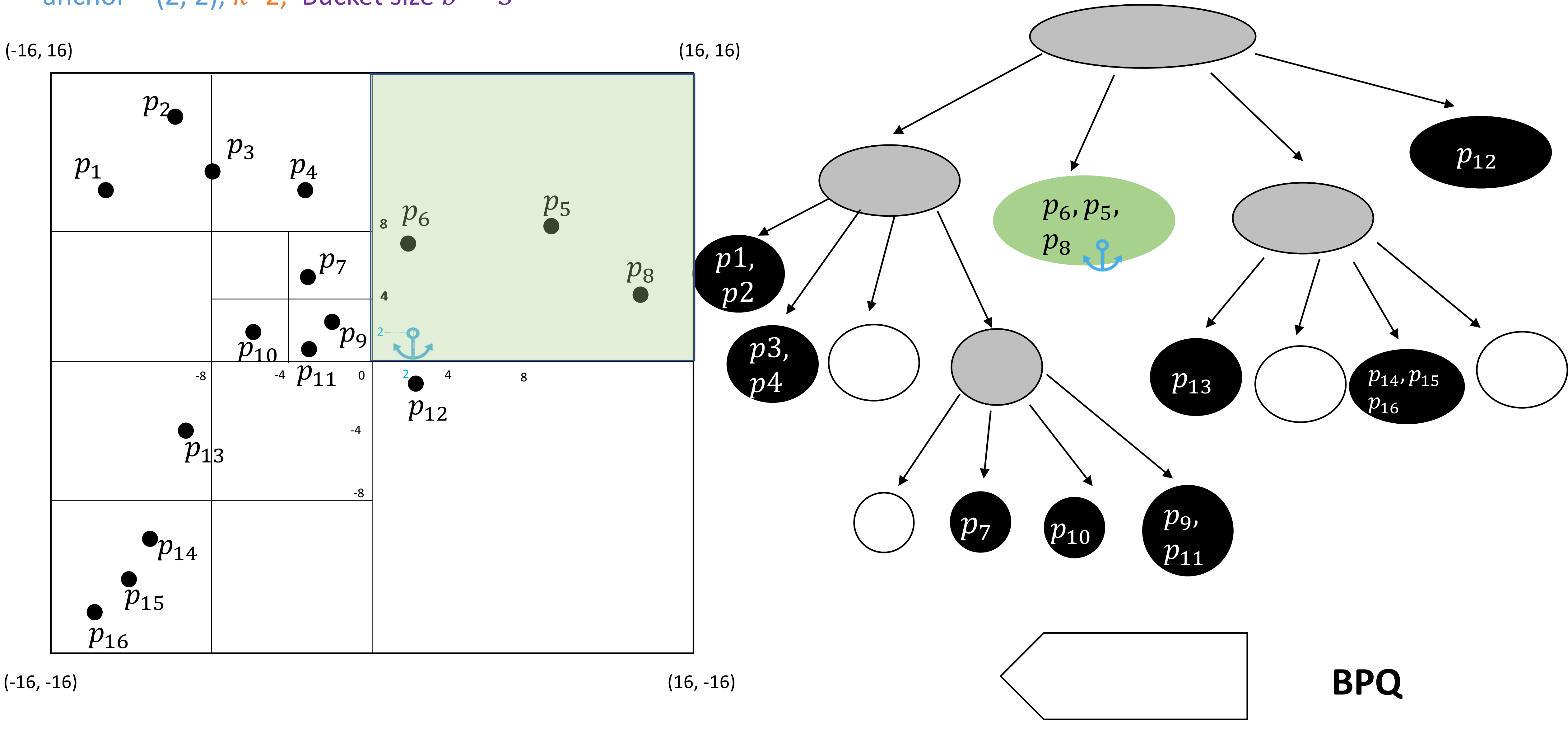
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size $b = 3$



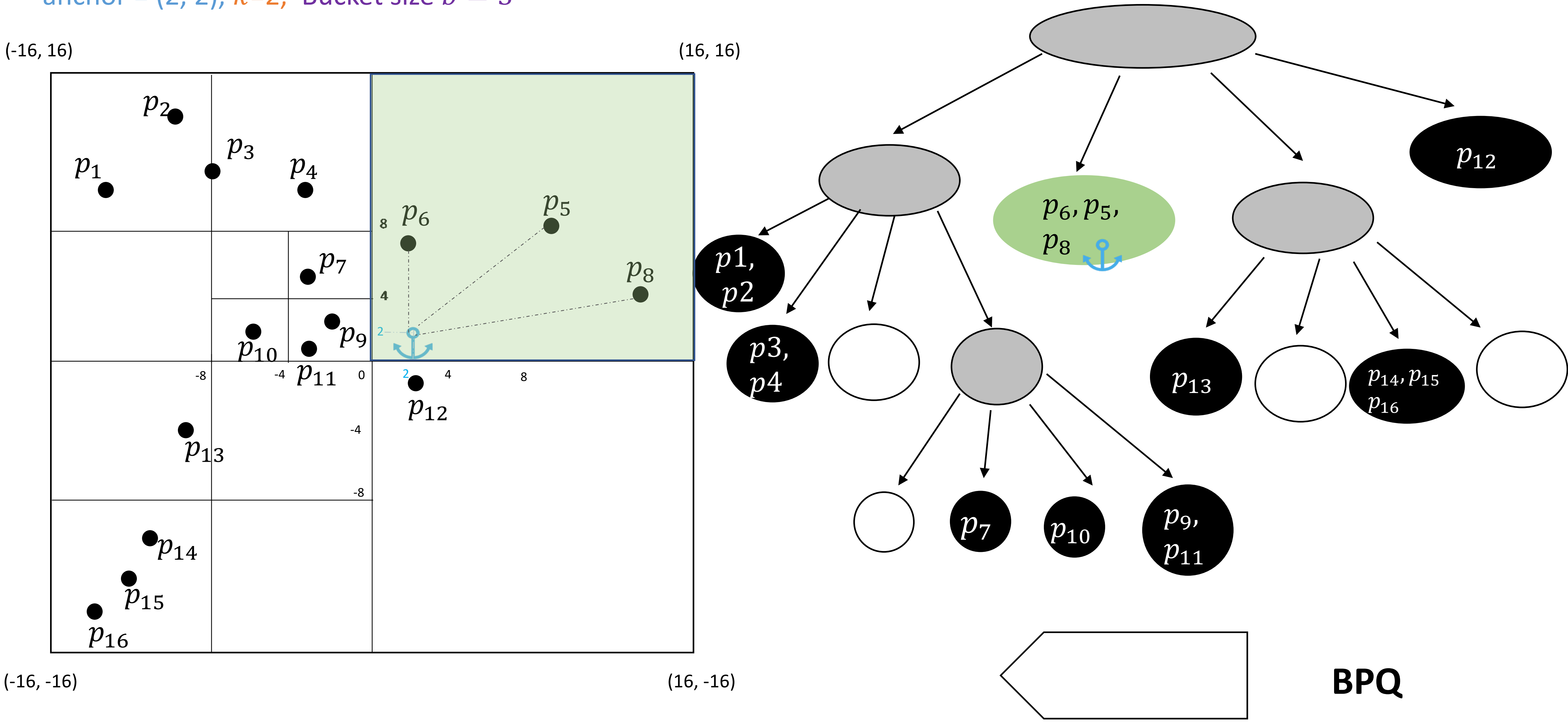
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



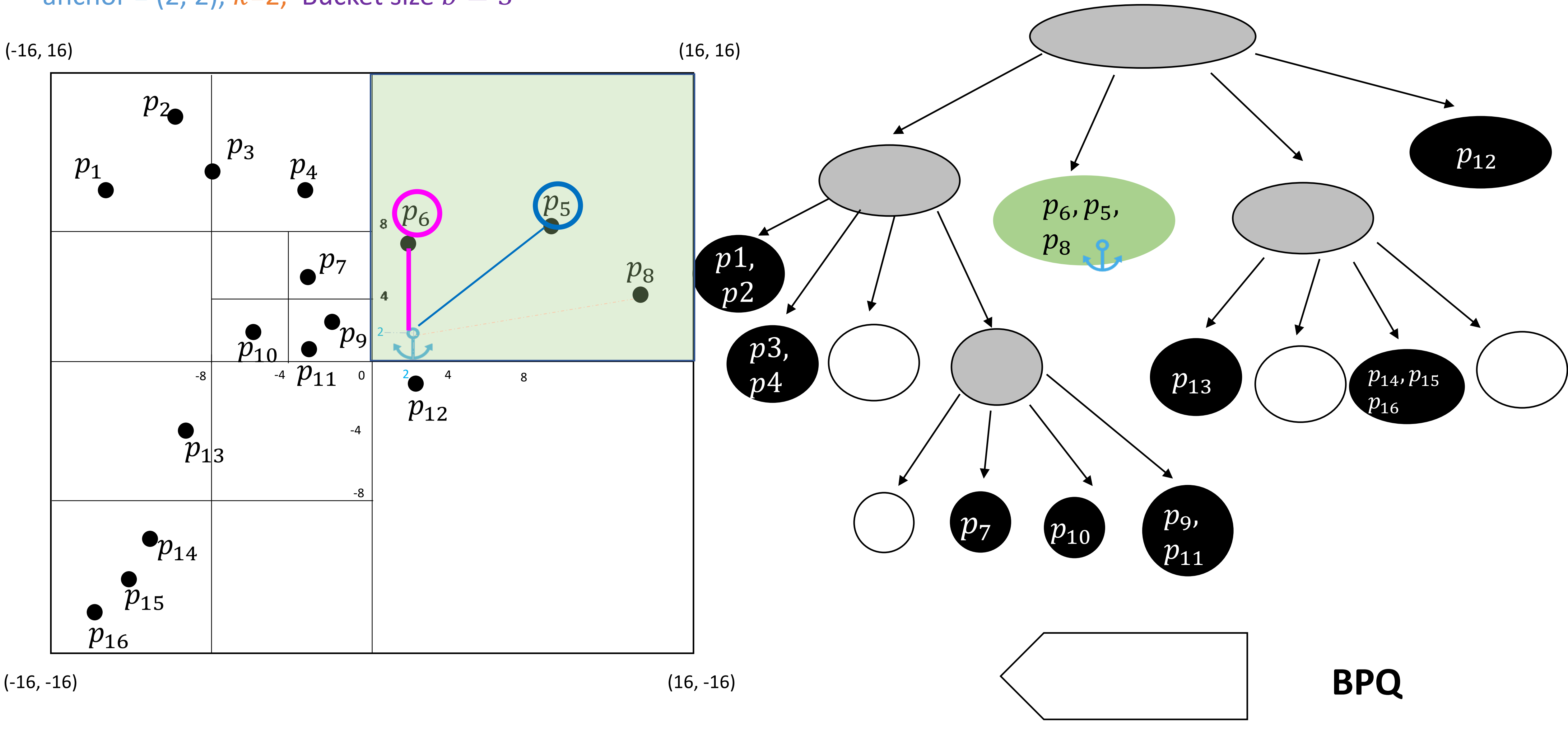
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



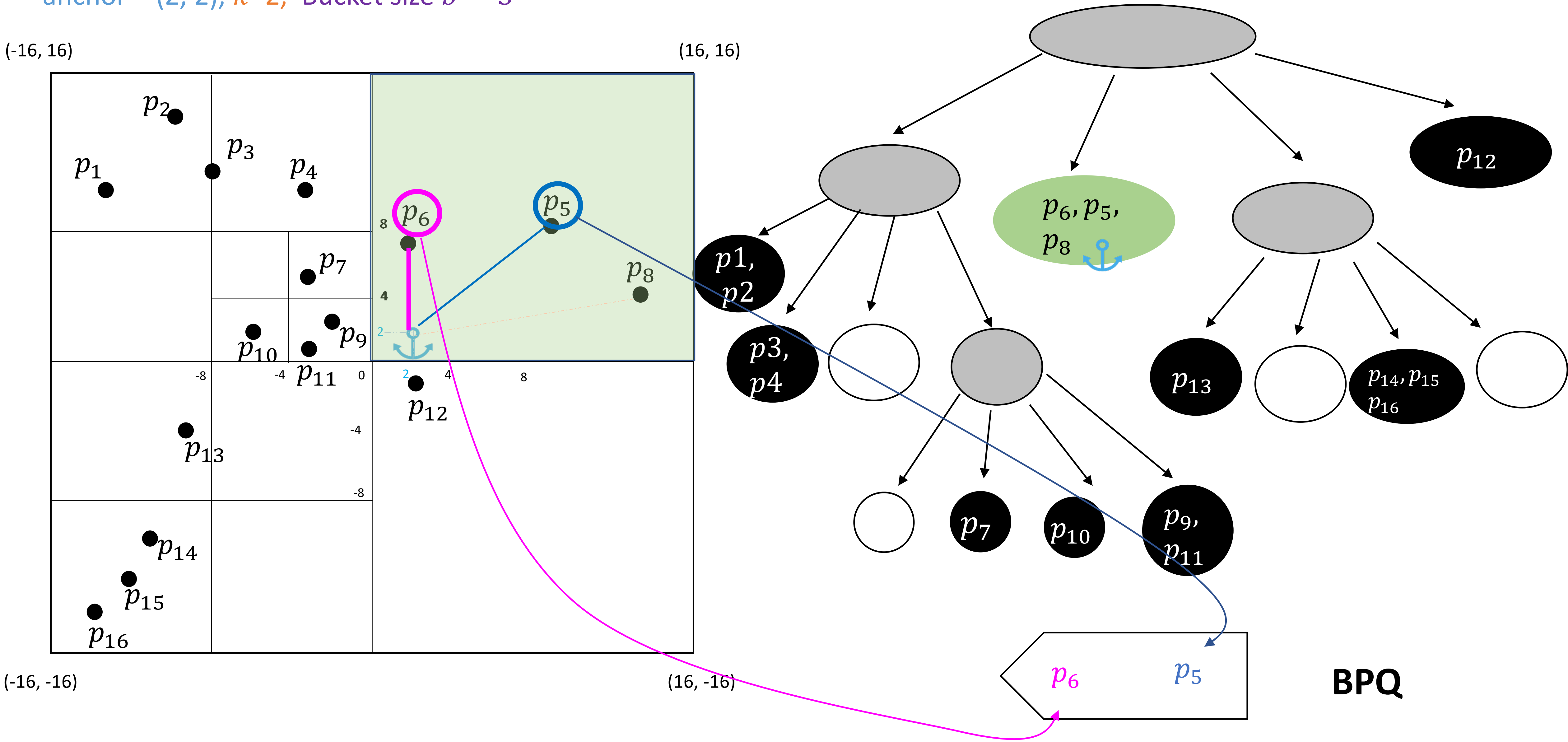
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



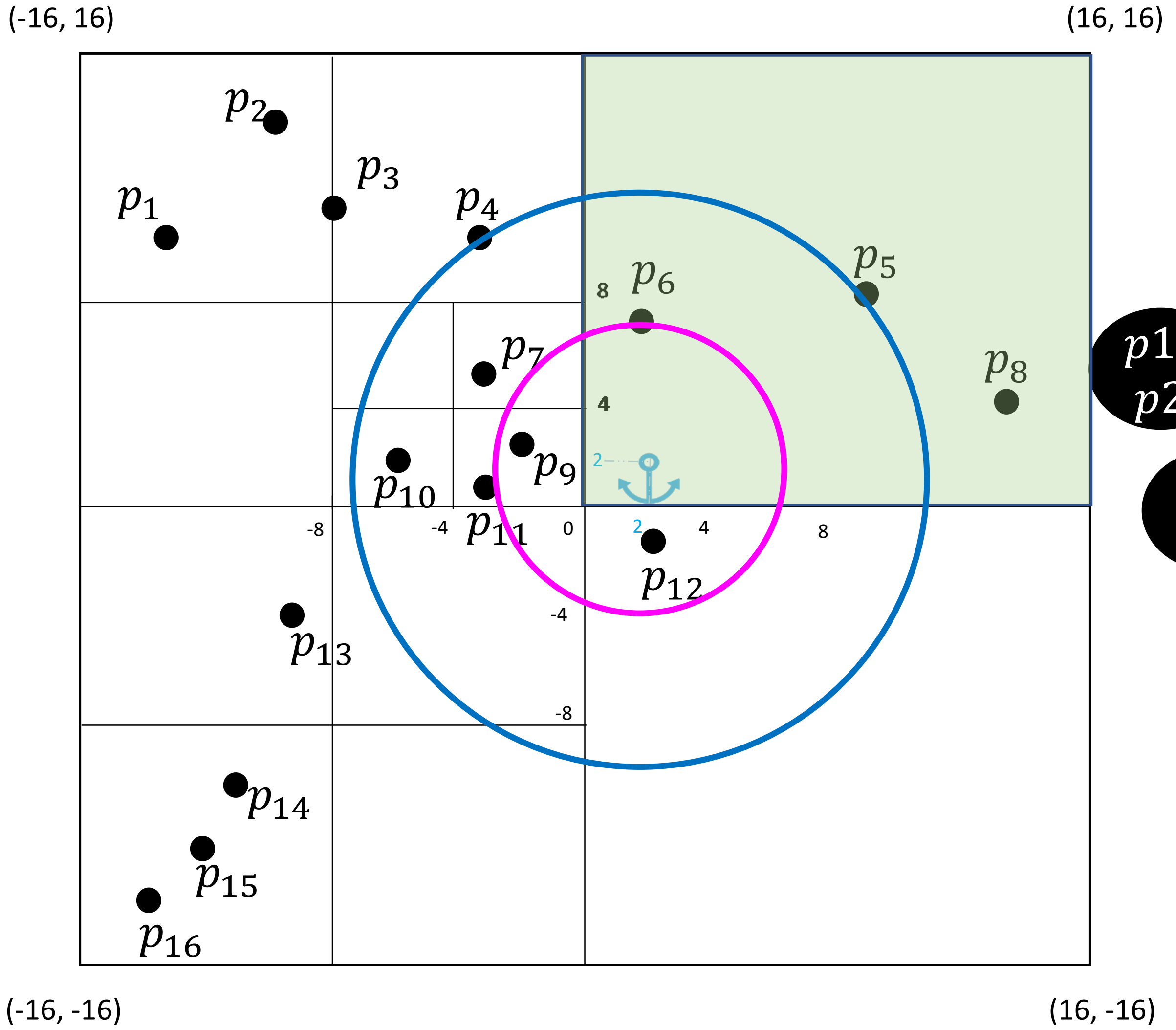
Example of k-NN query in PR-QuadTree

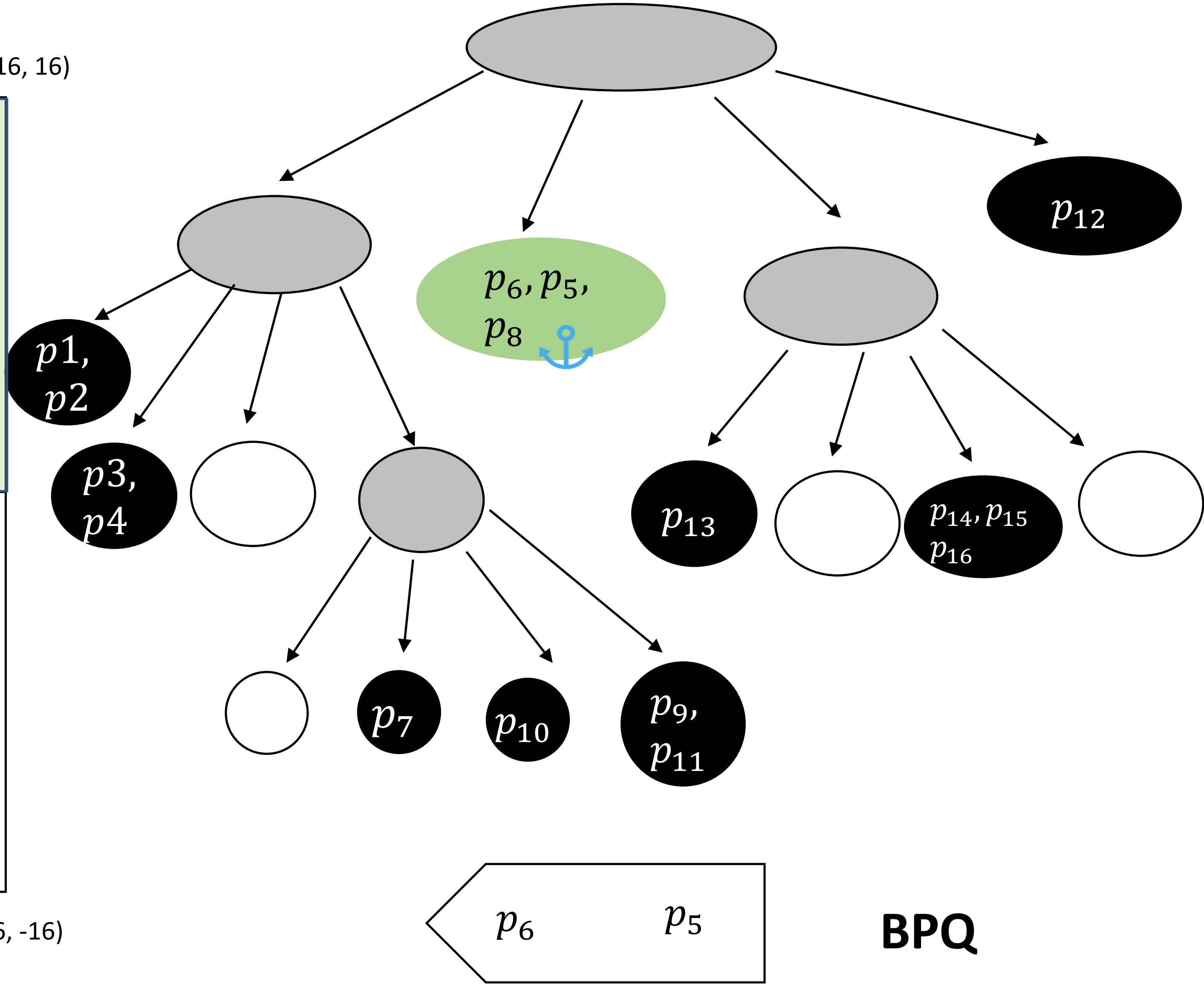
anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

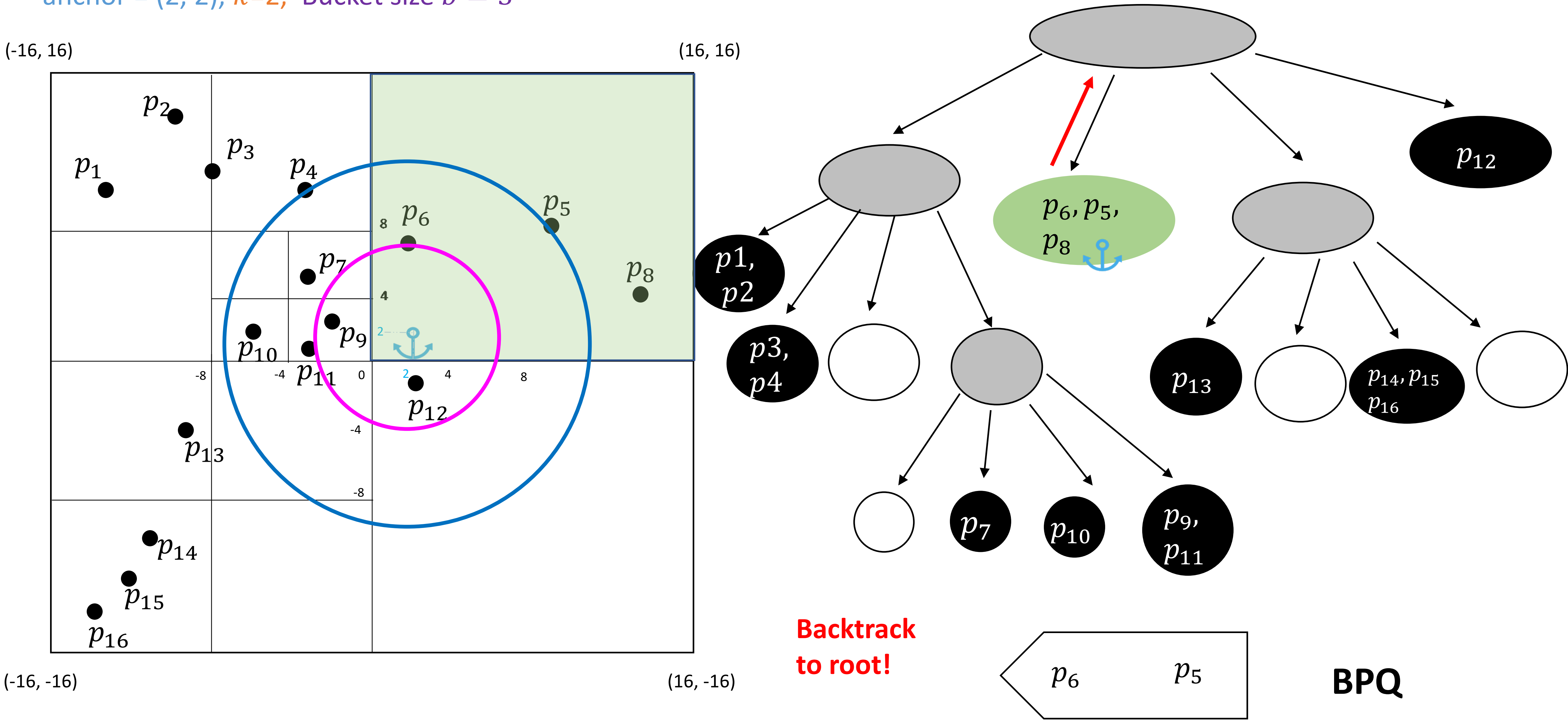
anchor = (2, 2), k=2, Bucket size b = 3





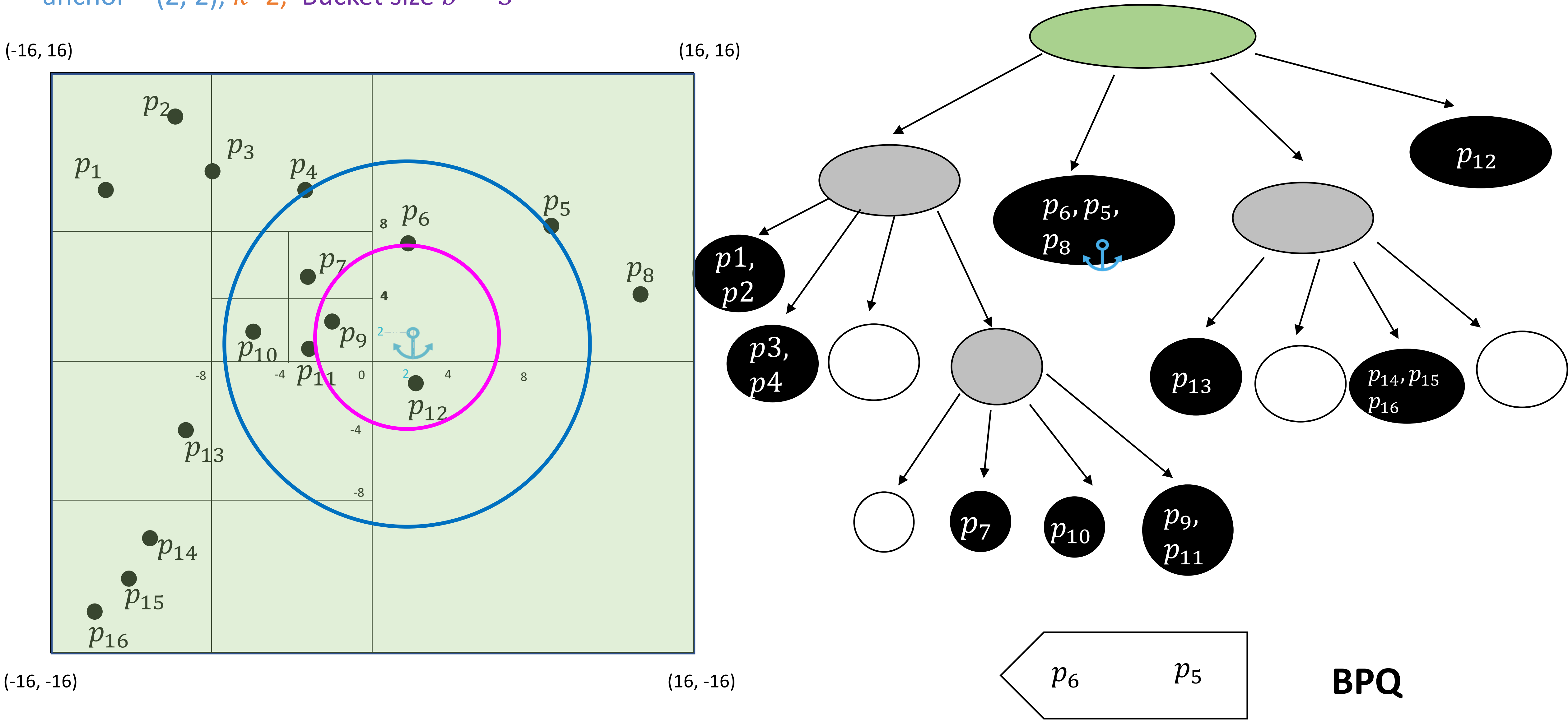
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



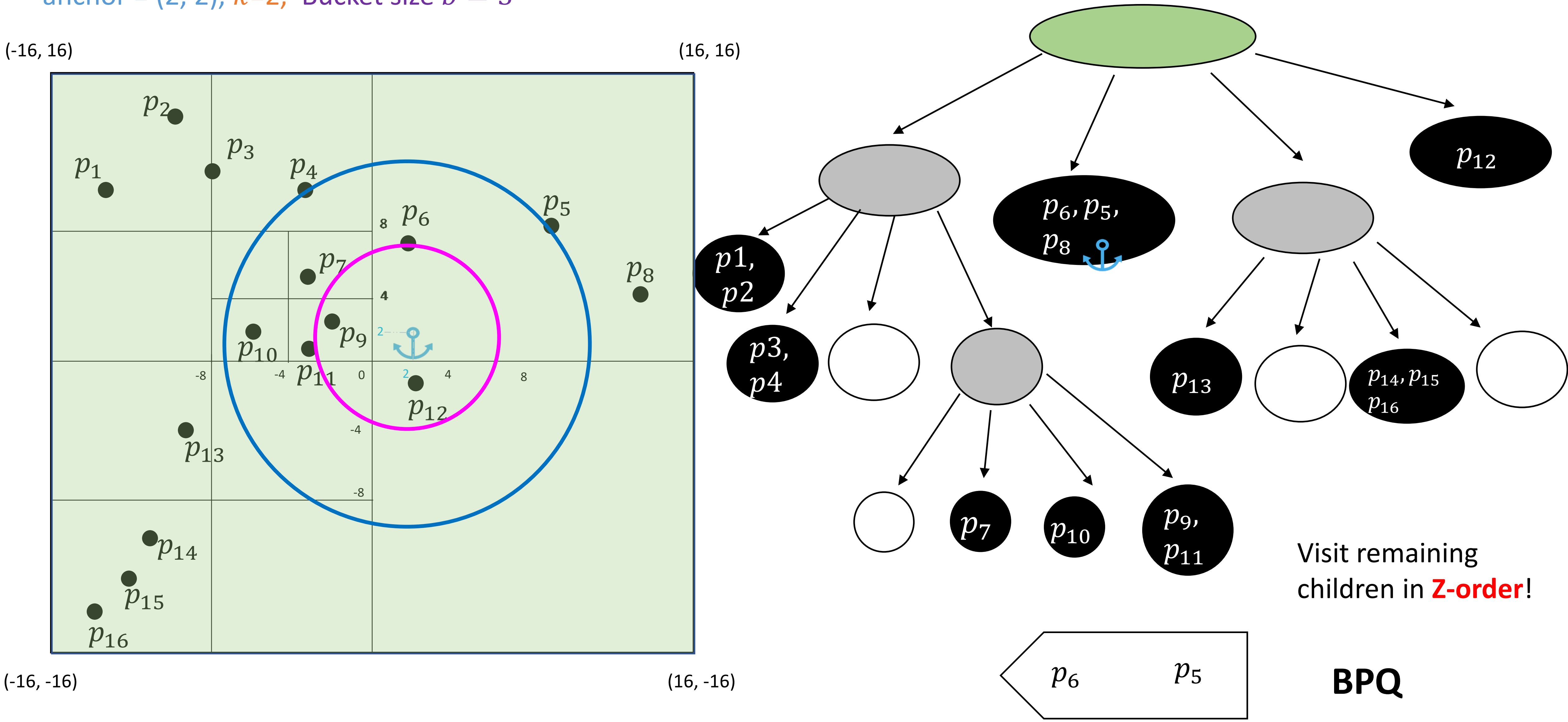
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3

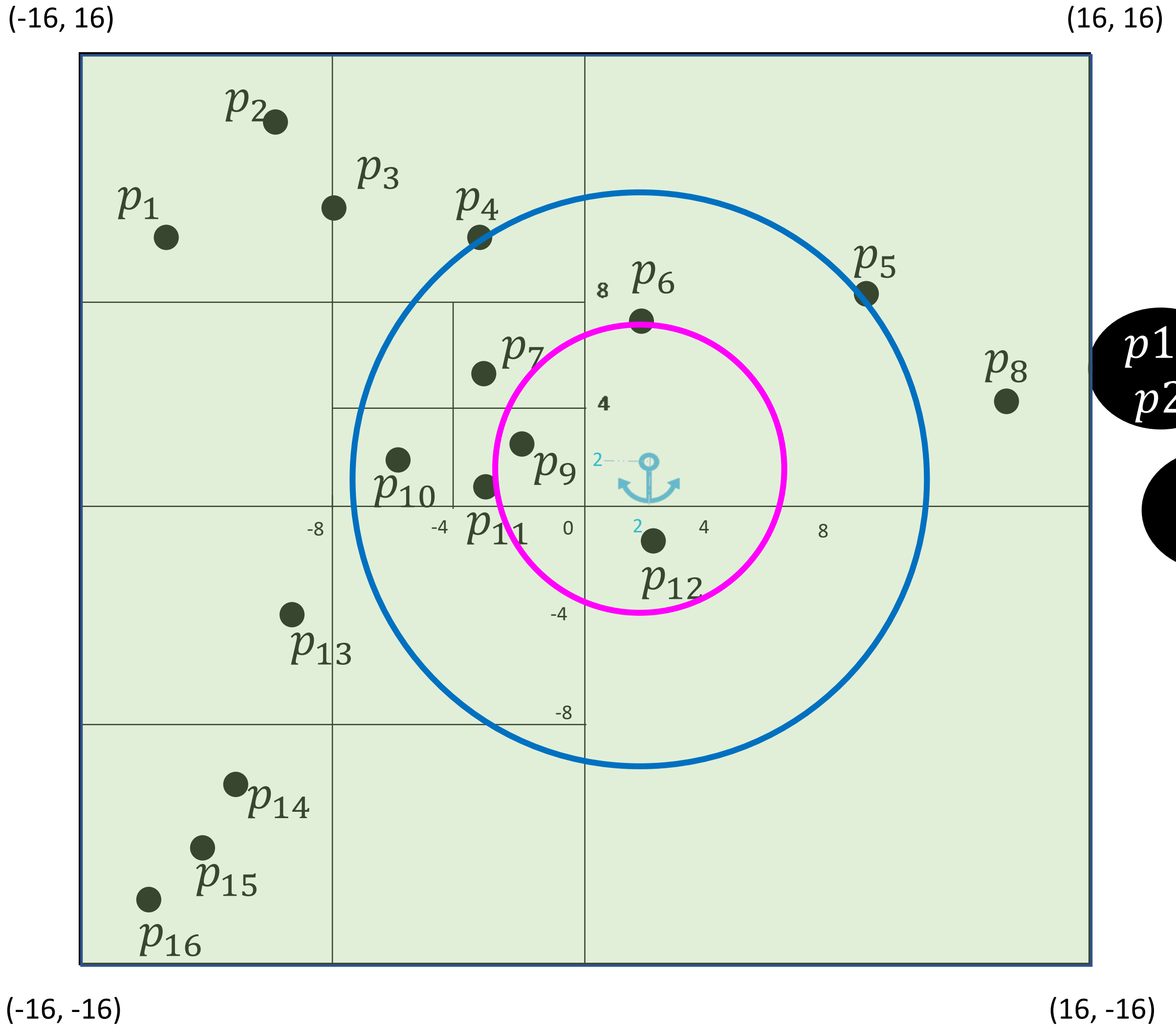


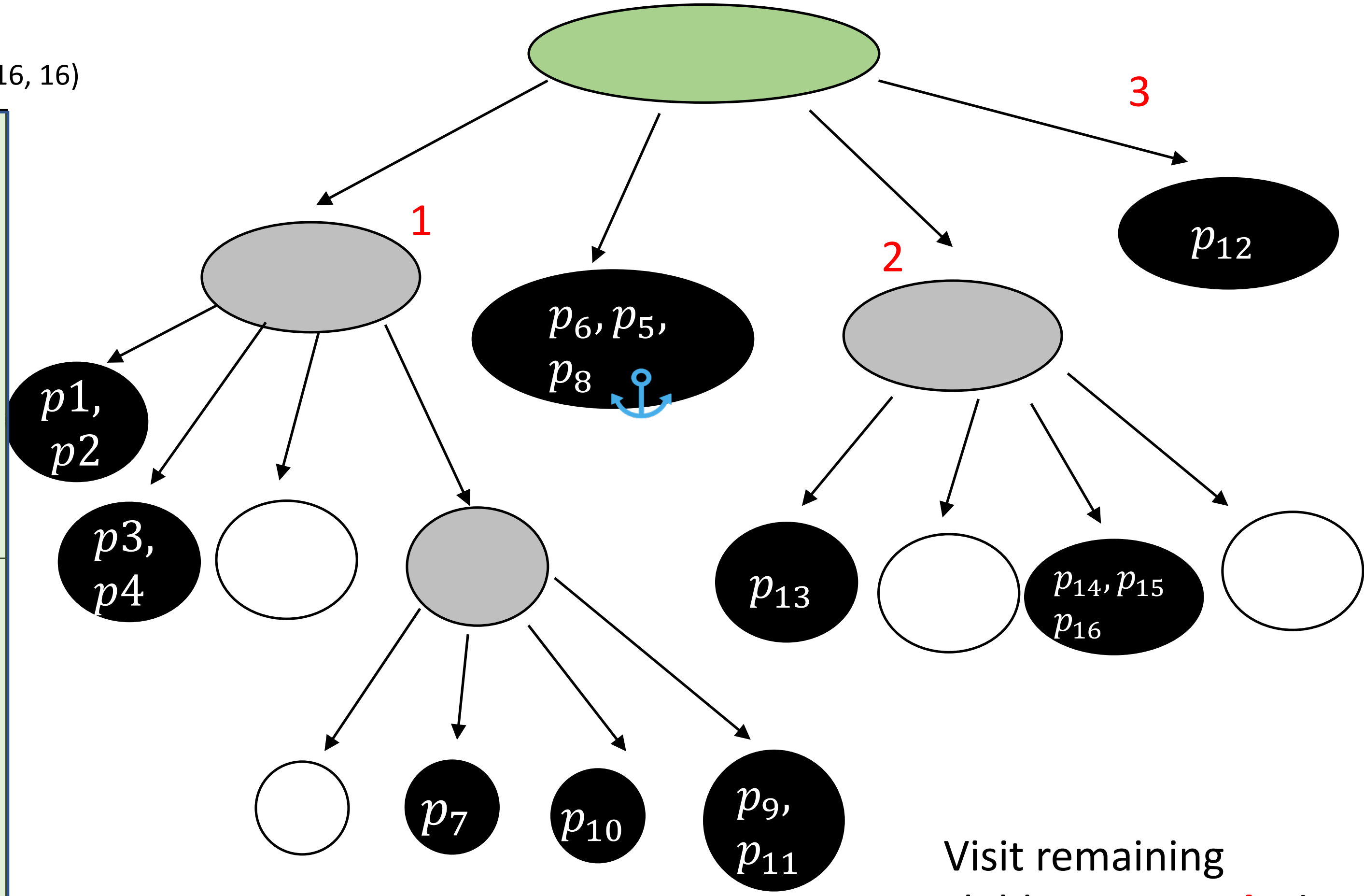
Visit remaining children in **Z-order**!

BPQ

Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



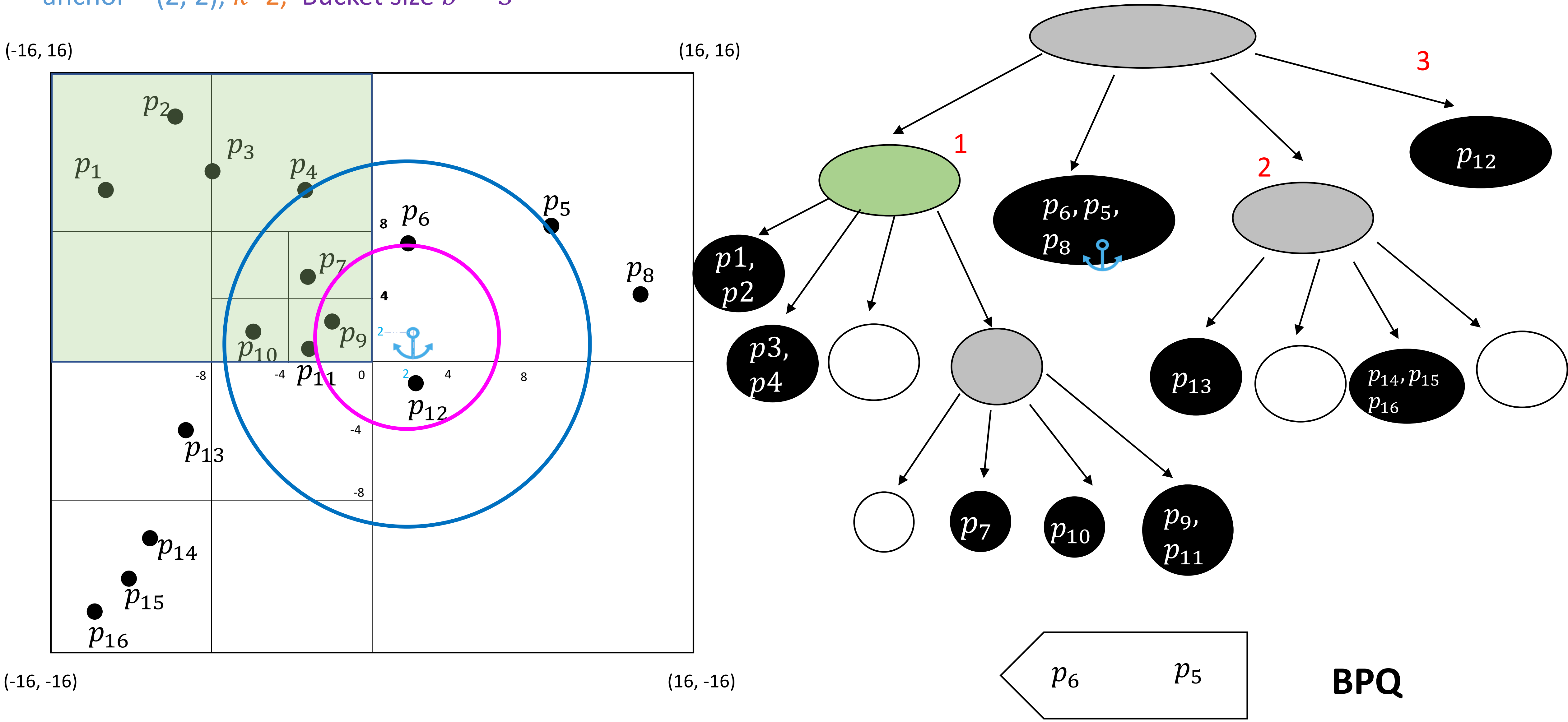


Visit remaining children in **Z-order**!

BPQ

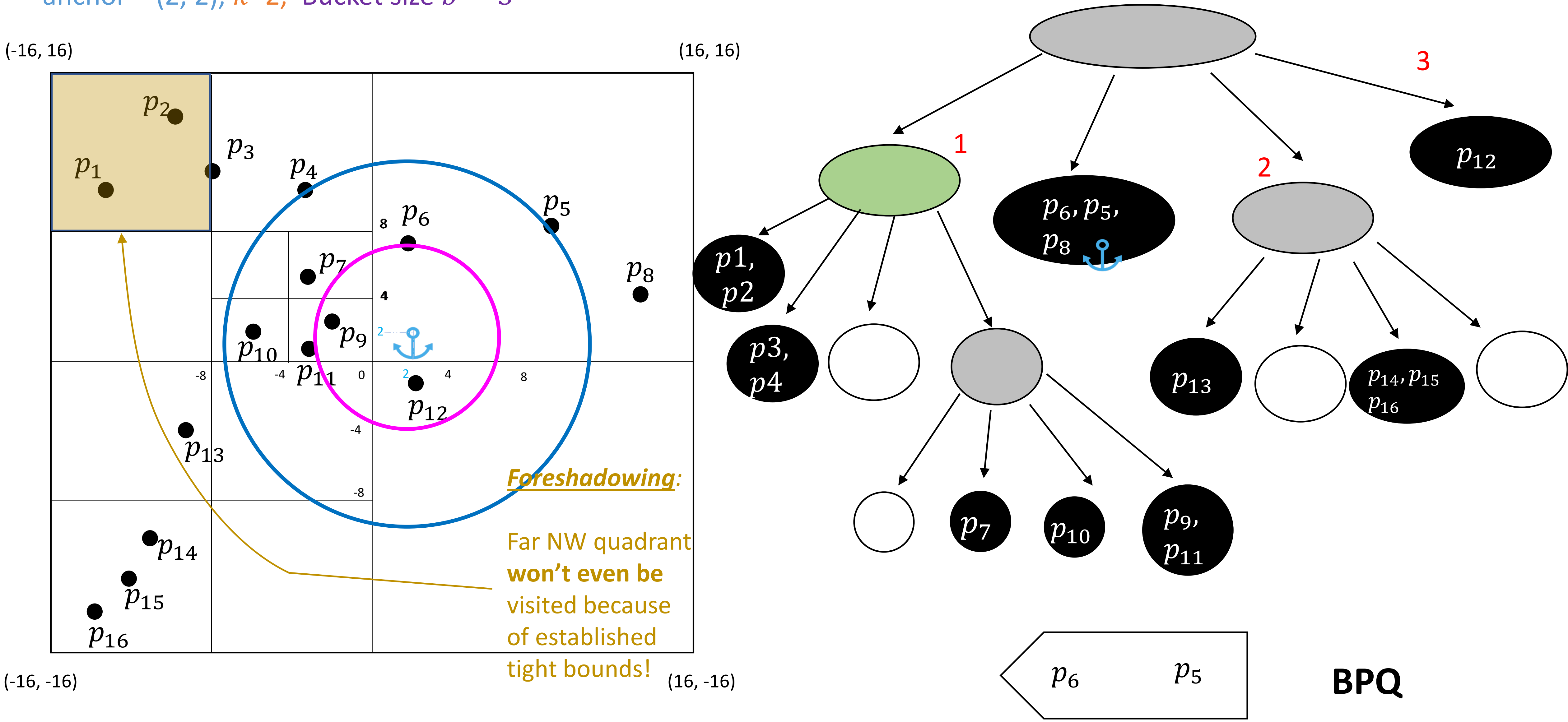
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



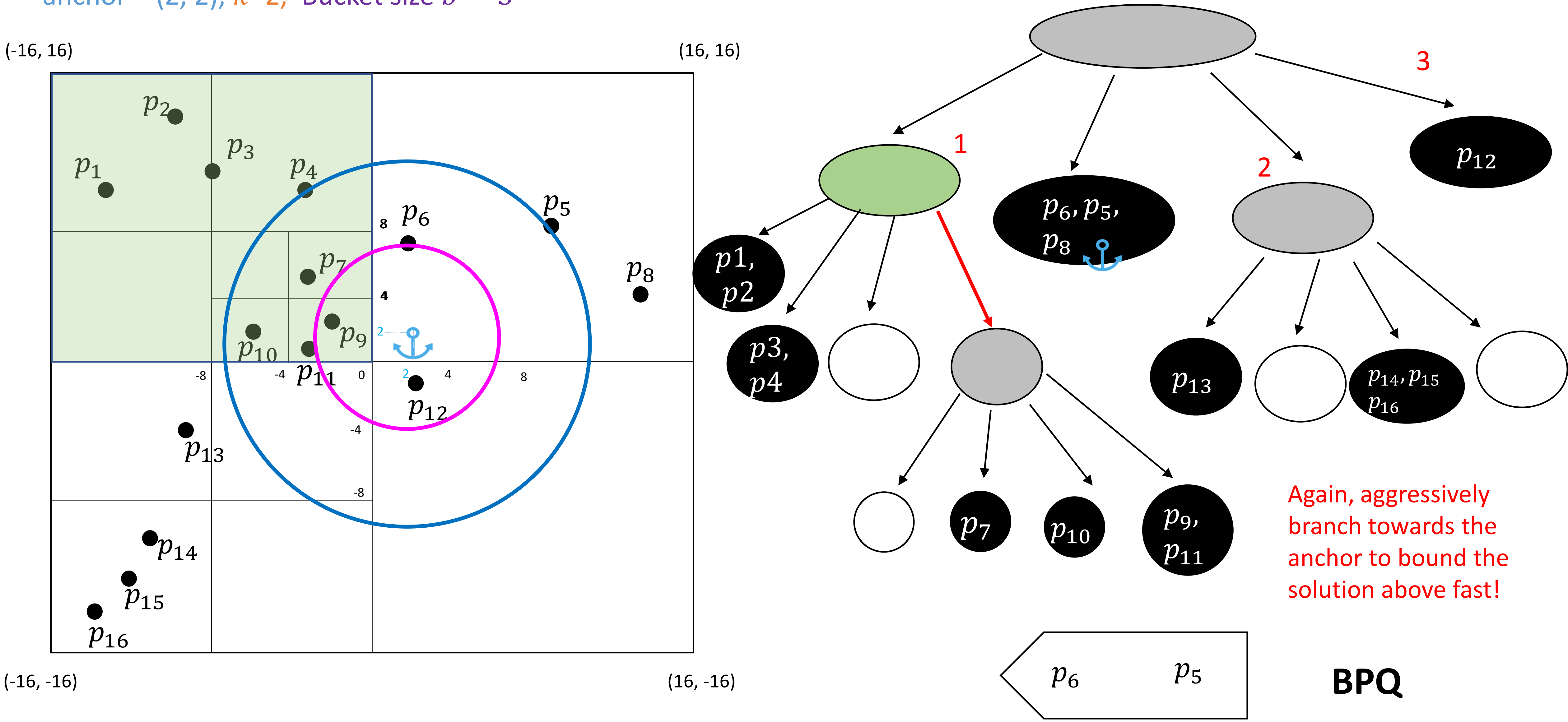
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3

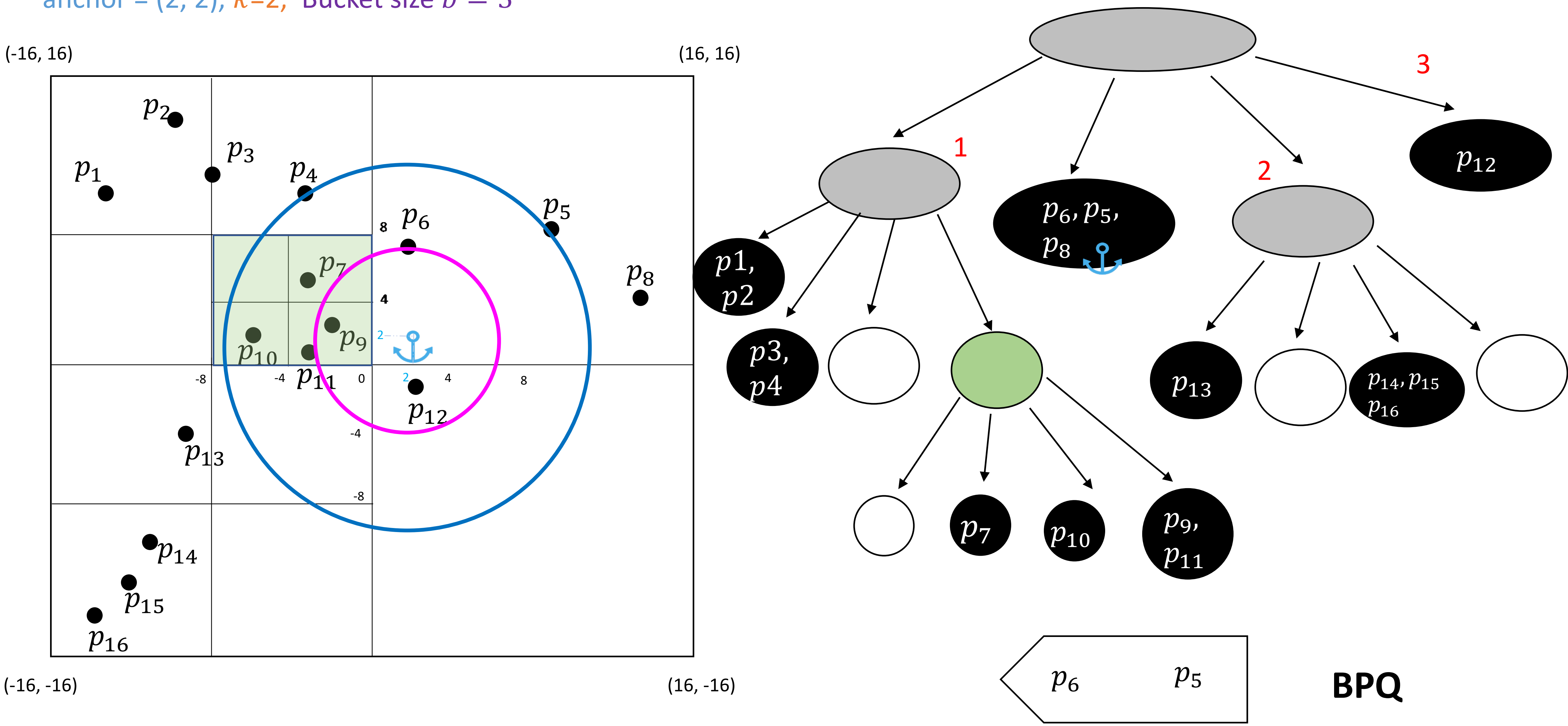


Again, aggressively branch towards the anchor to bound the solution above fast!

BPQ

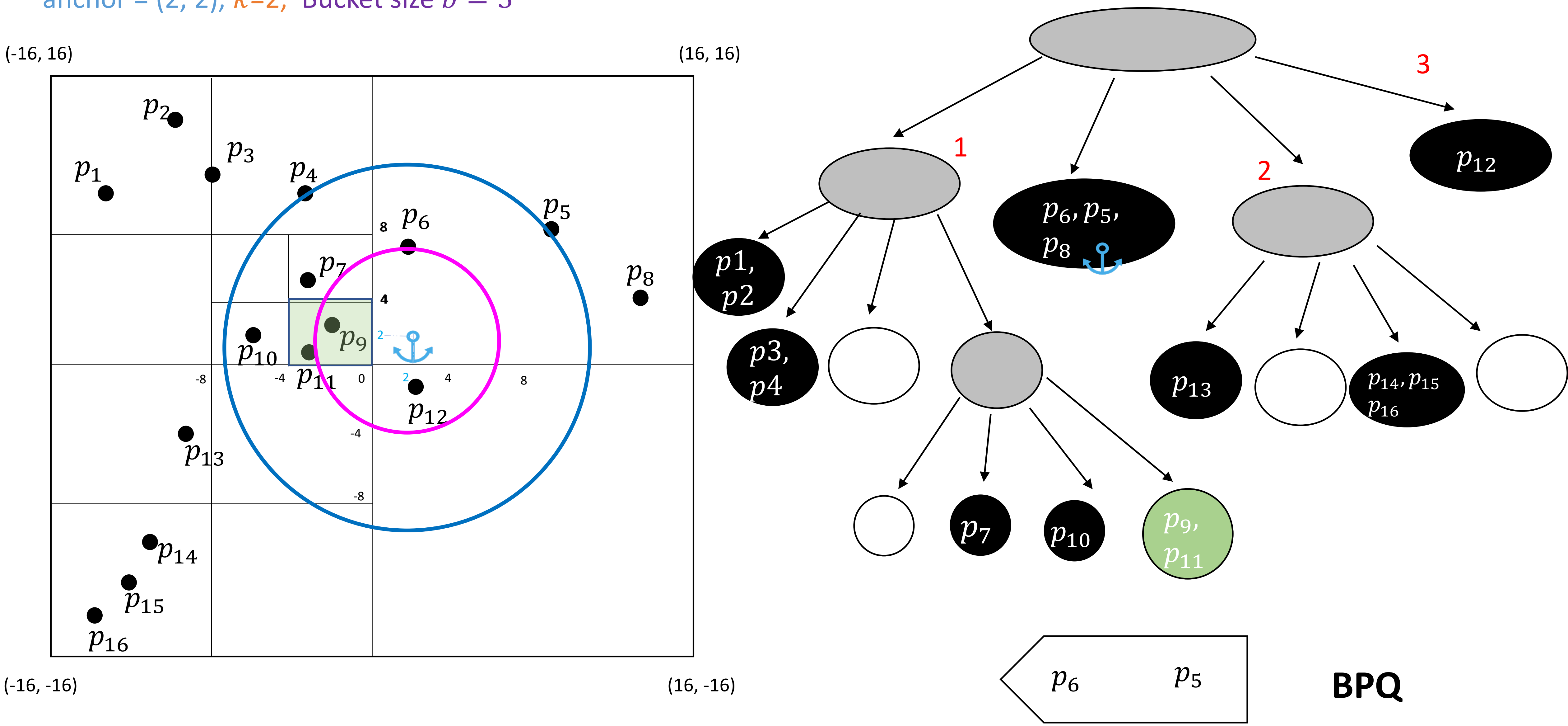
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



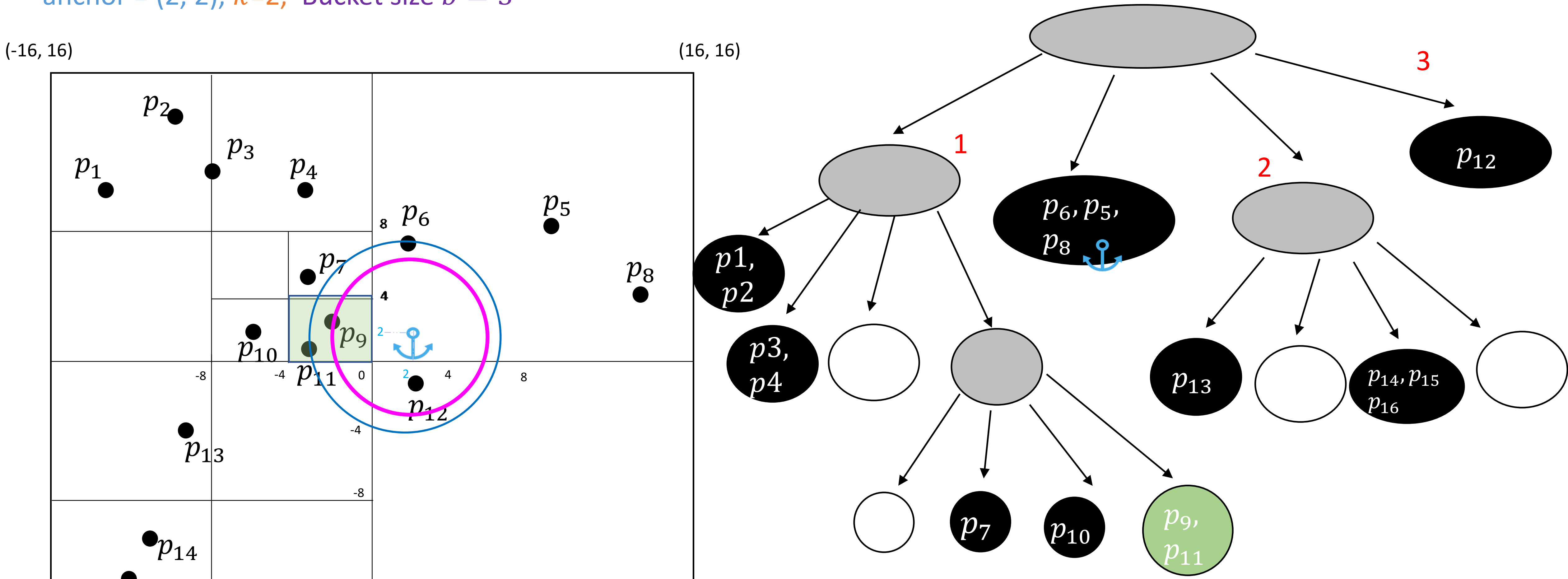
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3

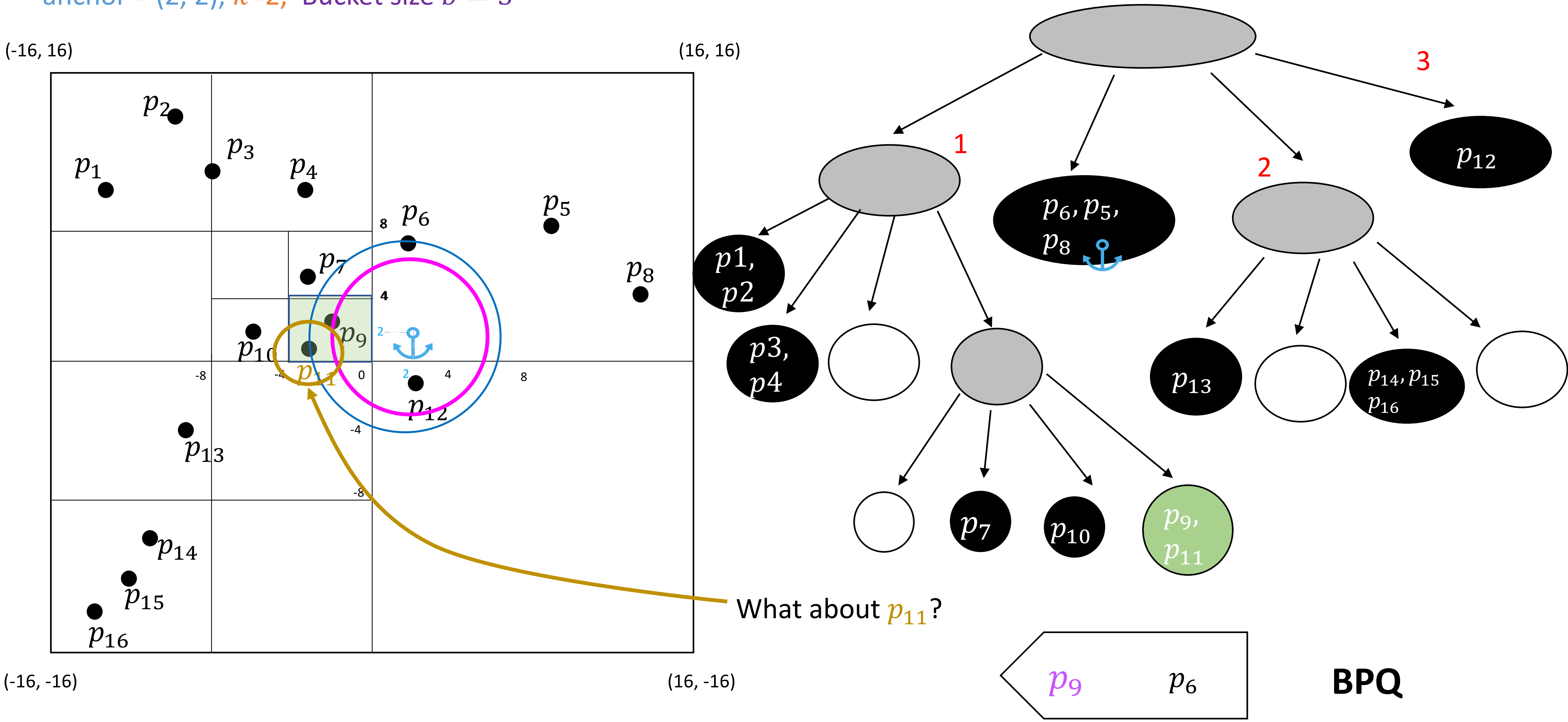


p_9 improves upon p_6!
New best neighbor!

BPQ

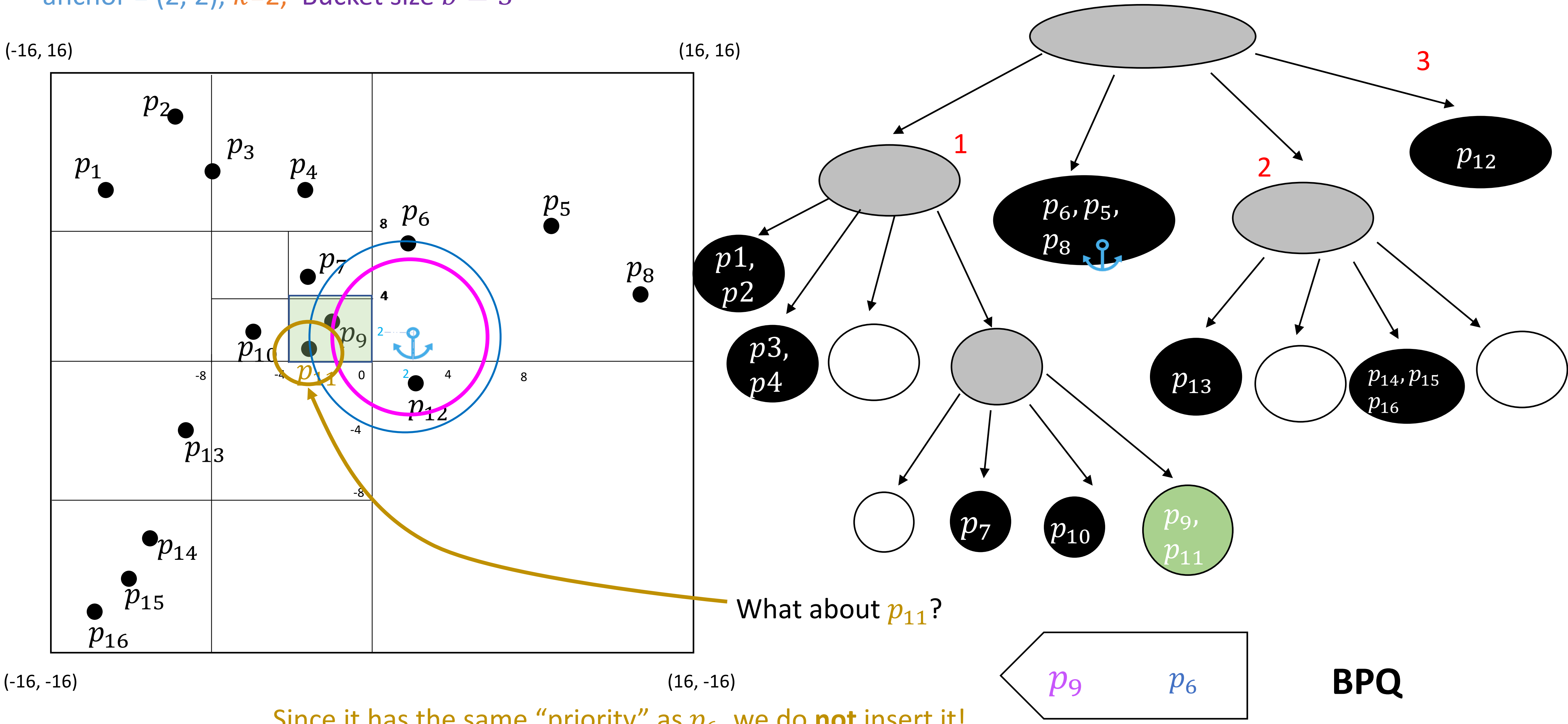
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



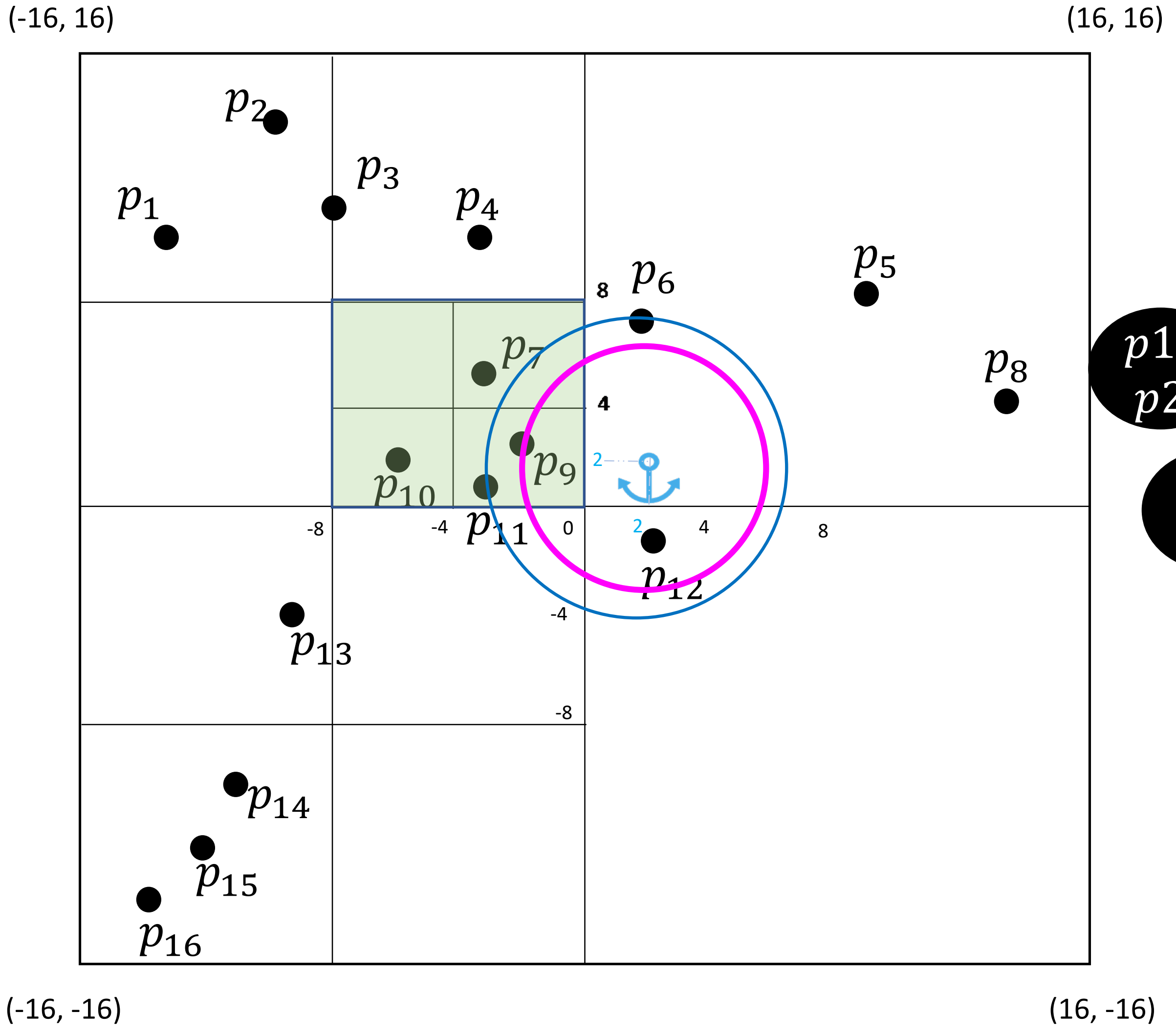
What about p_{11}?

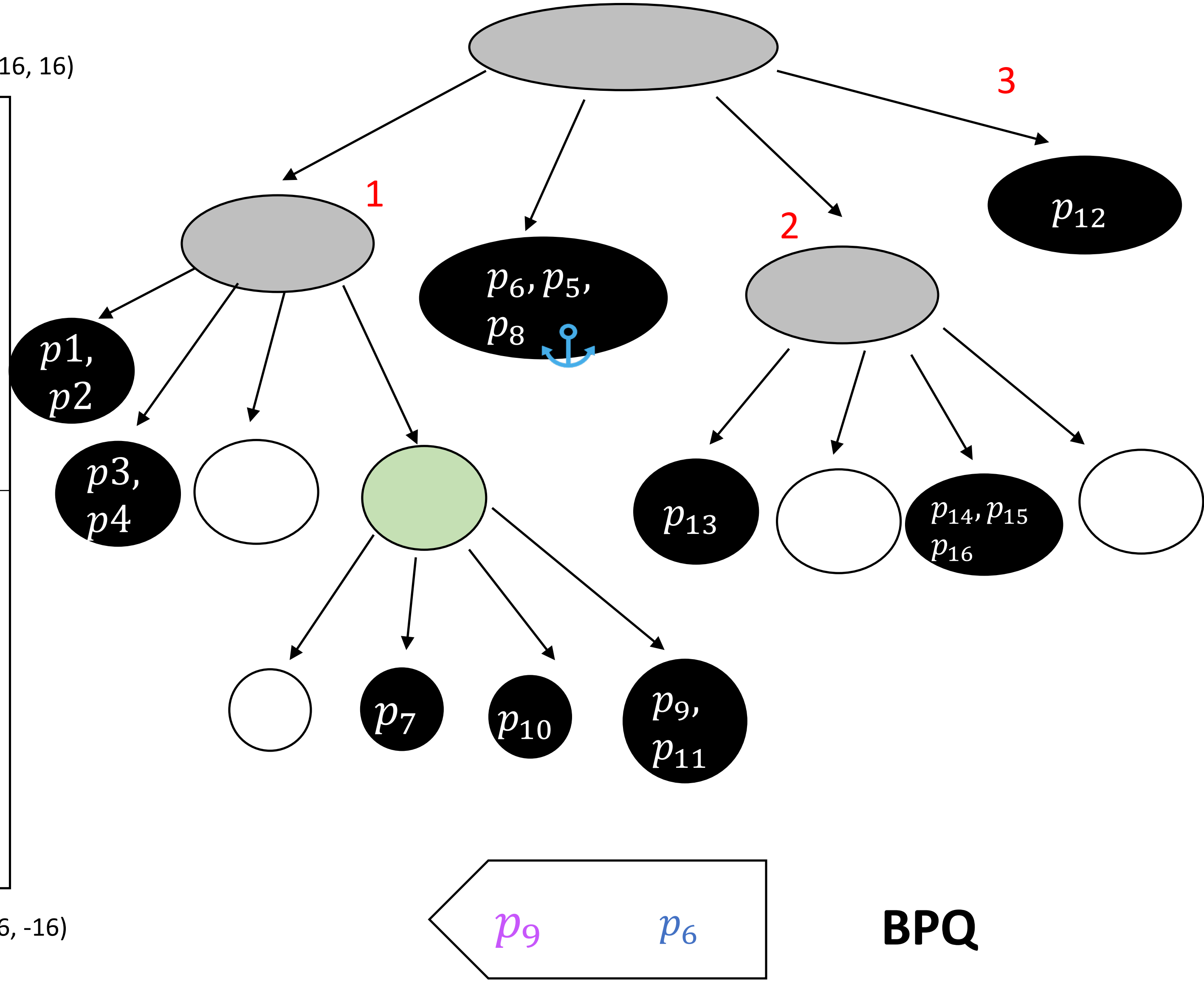
Since it has the same "priority" as p_6, we do **not** insert it!

p_6 is now our second-best neighbor!

Example of k-NN query in PR-QuadTree

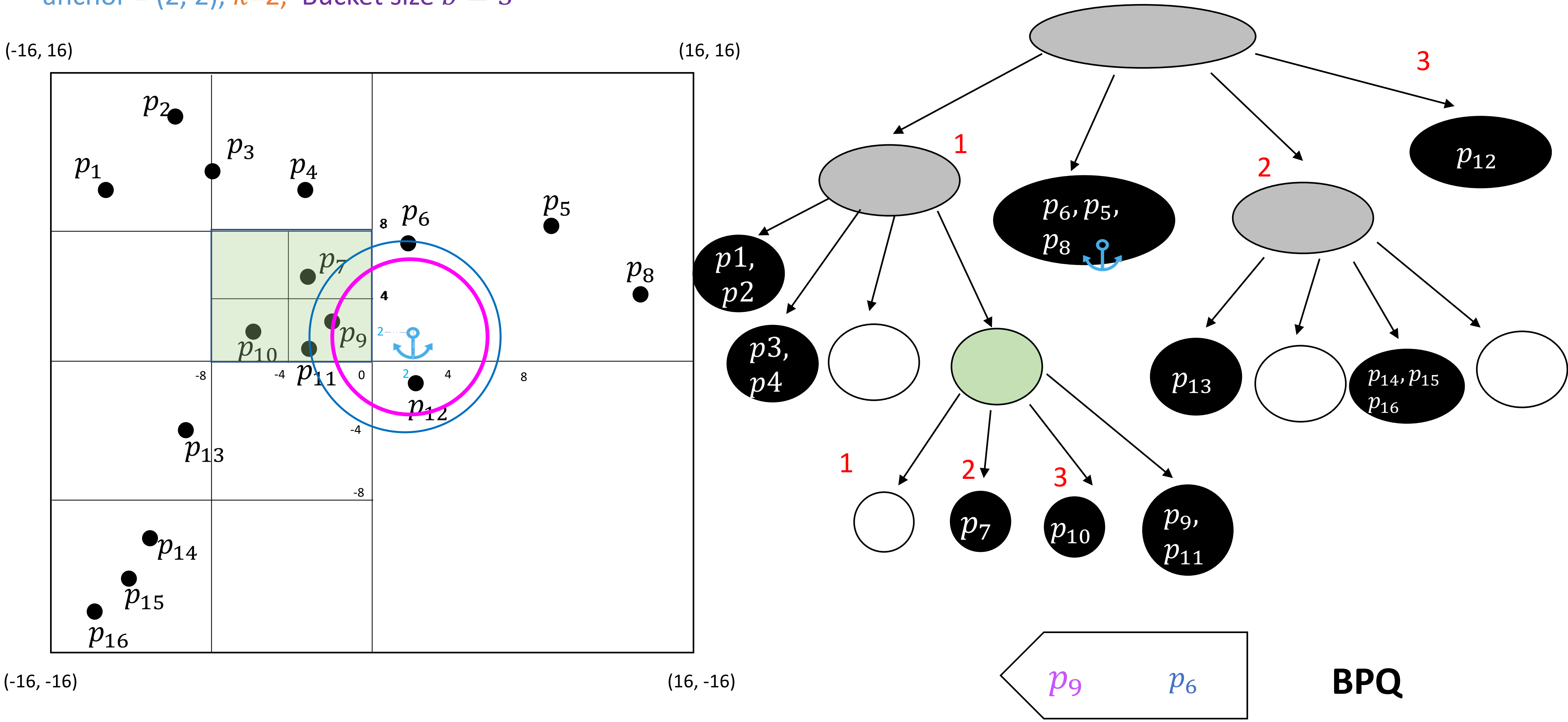
anchor = (2, 2), k=2, Bucket size $b = 3$





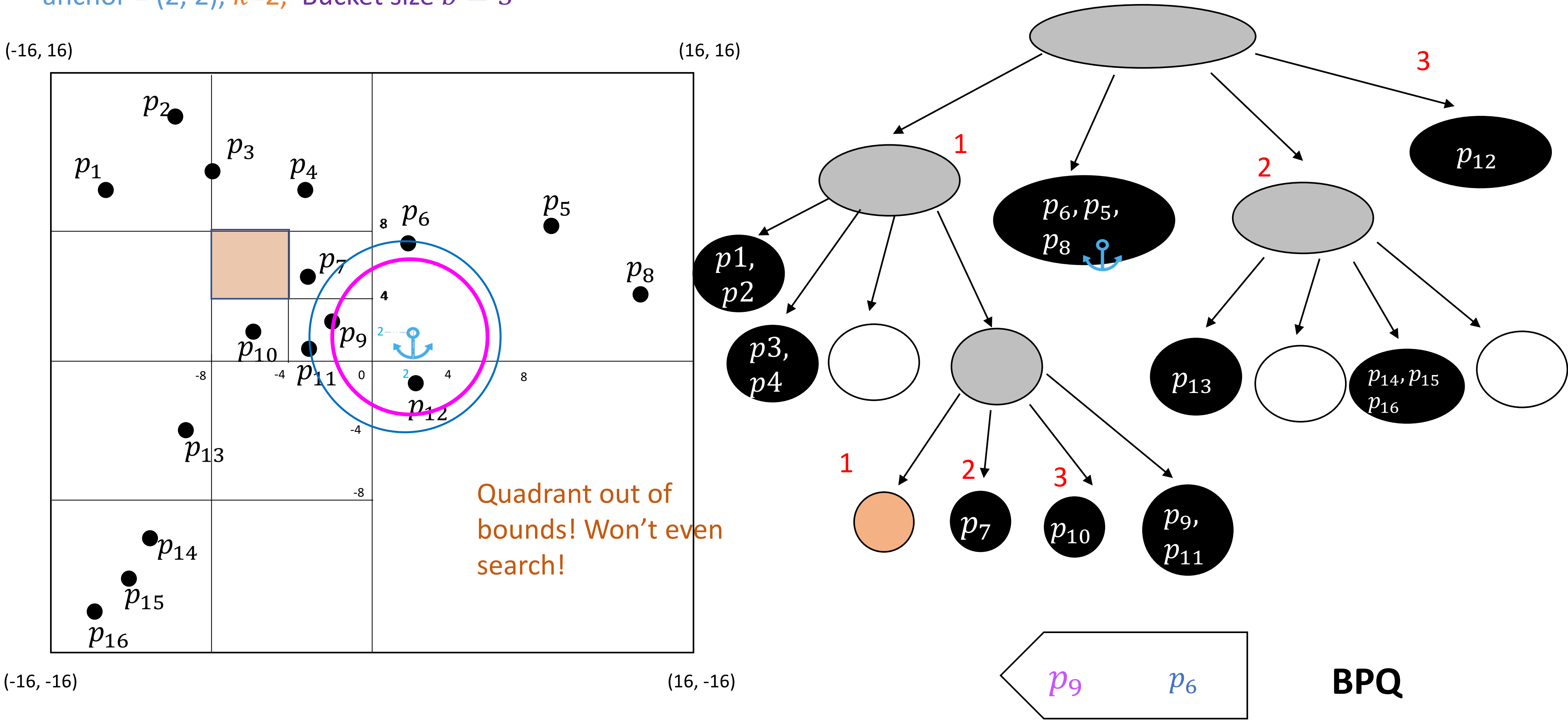
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



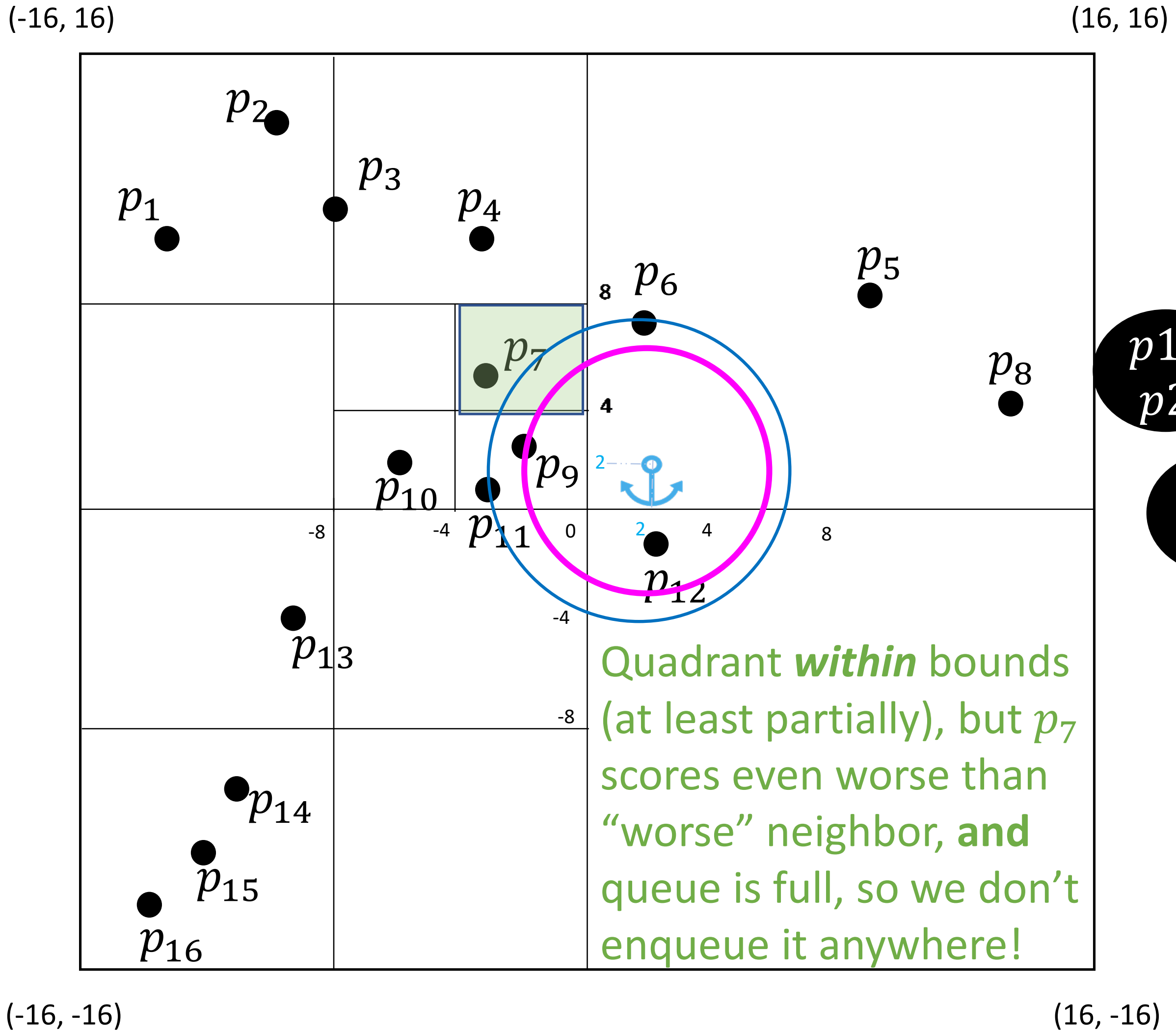
Example of k-NN query in PR-QuadTree

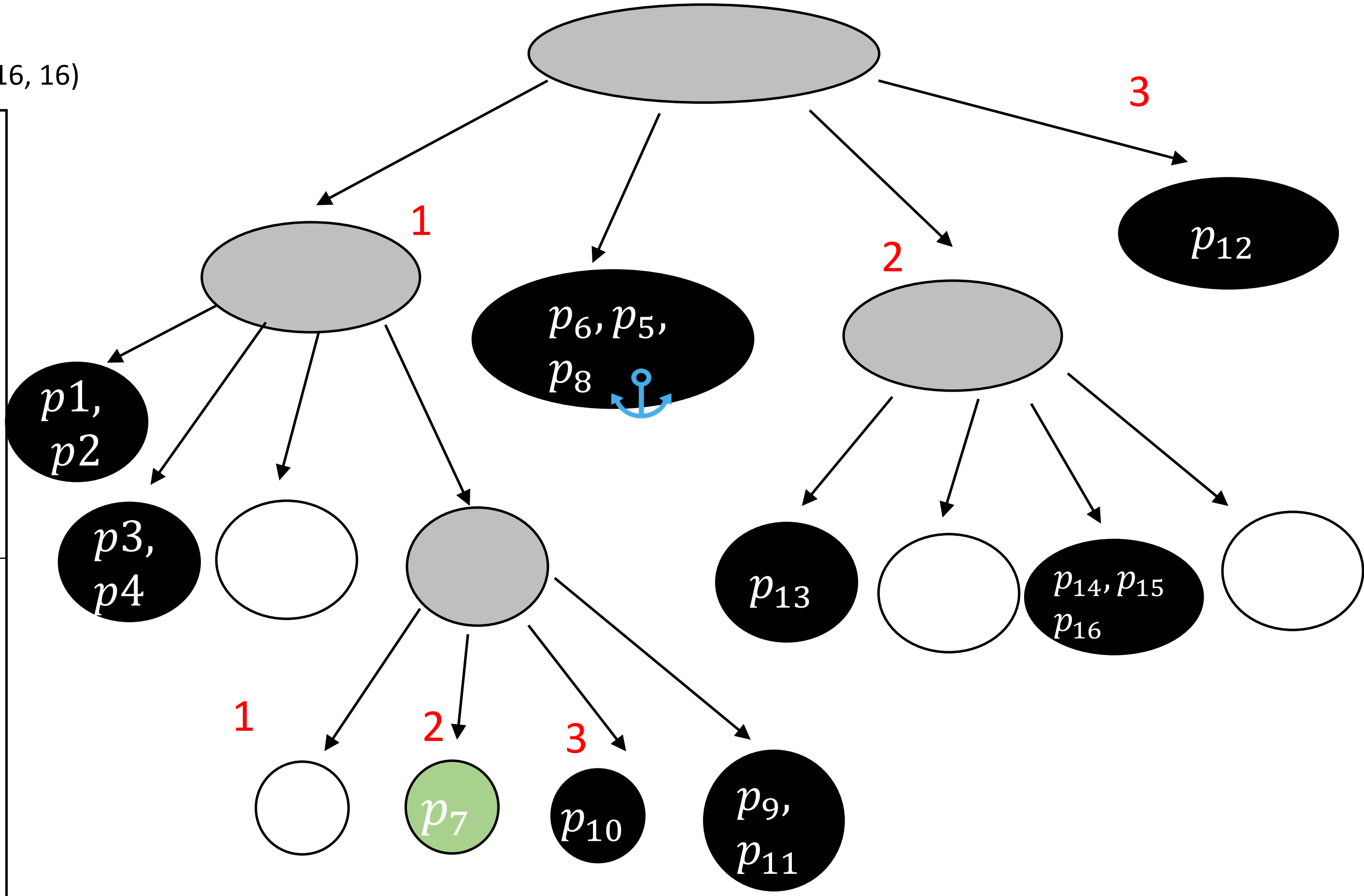
anchor = (2, 2), k=2, Bucket size b = 3

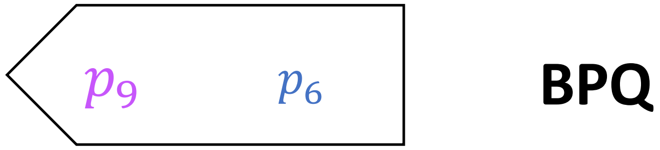


Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3

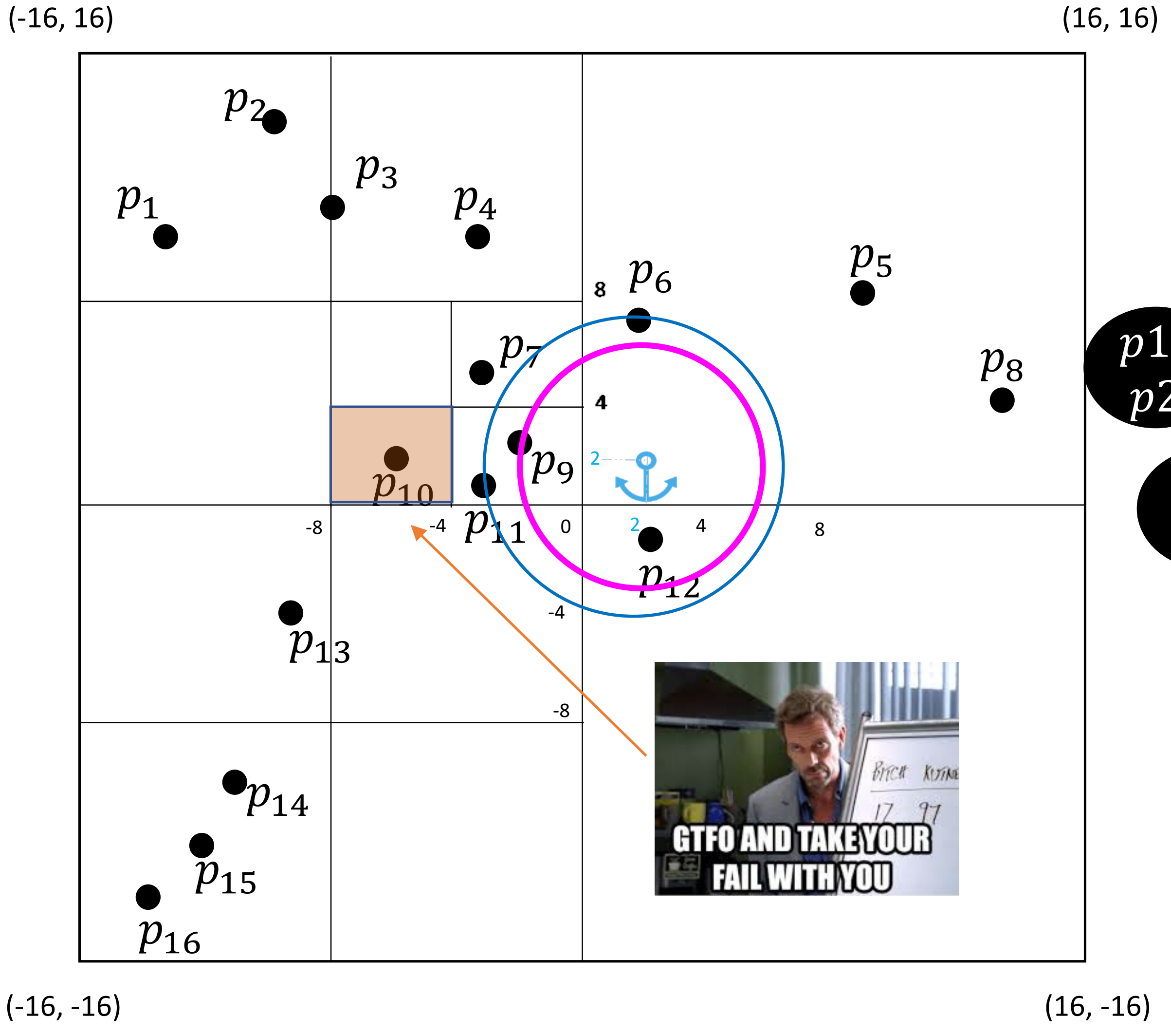


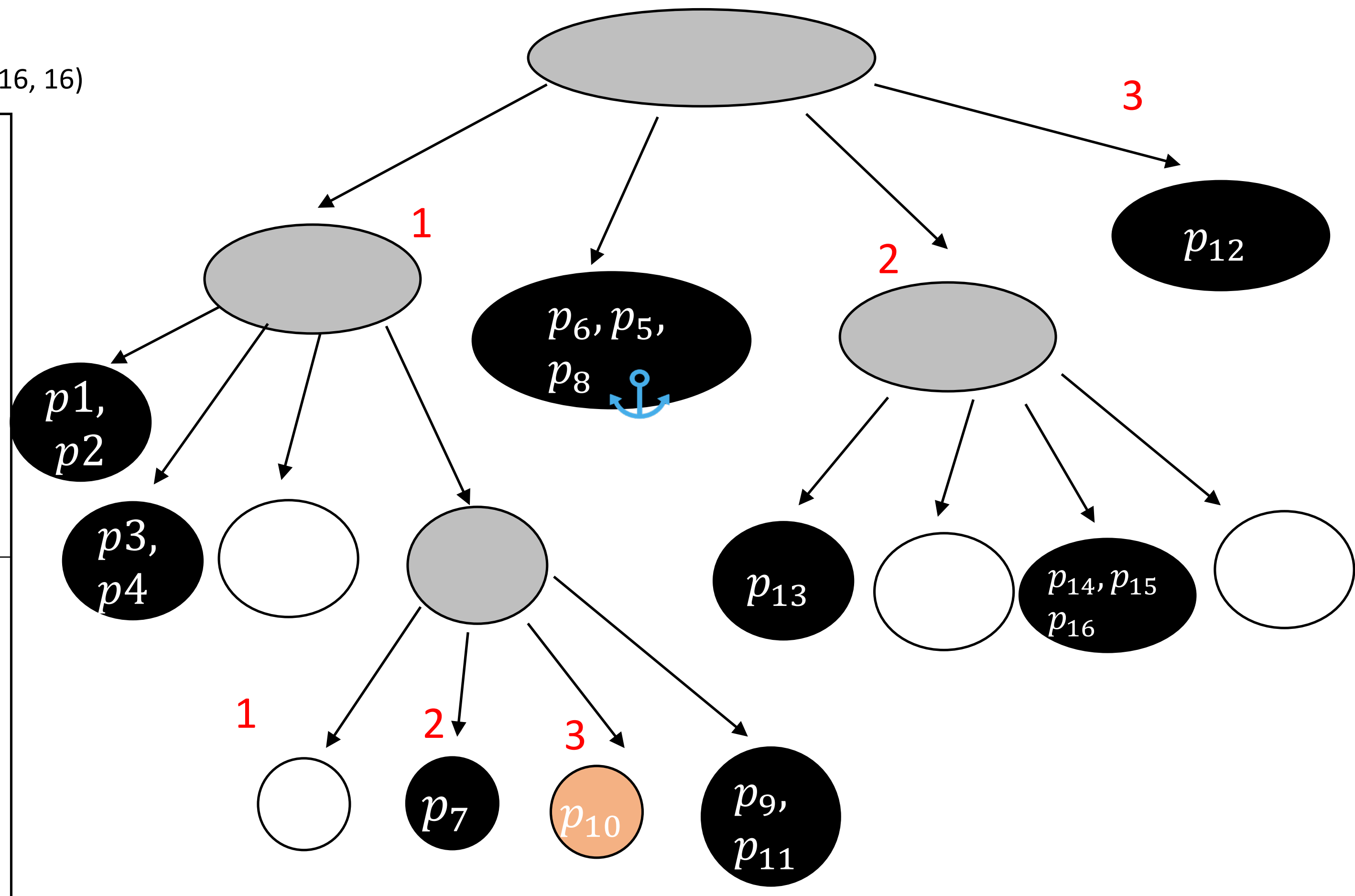




Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3

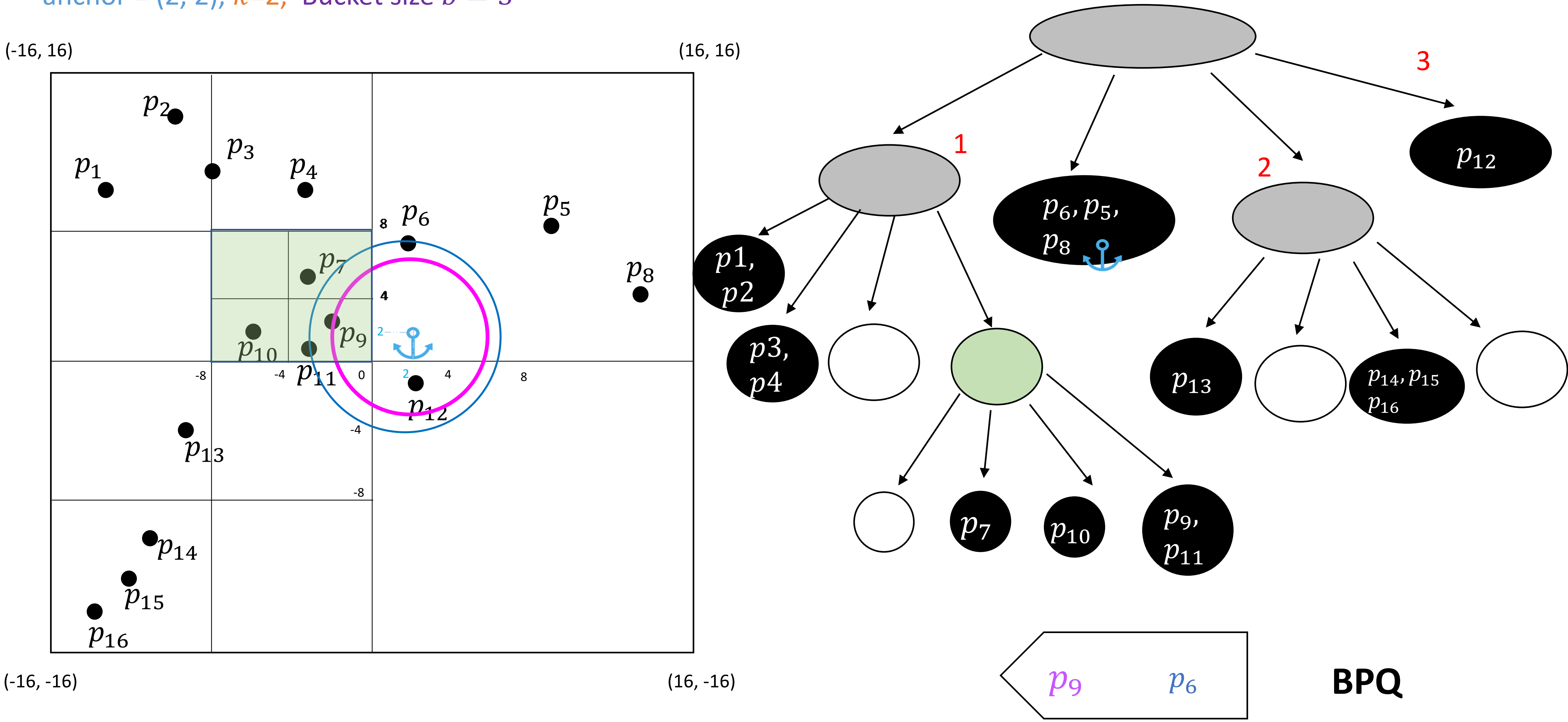




BPQ

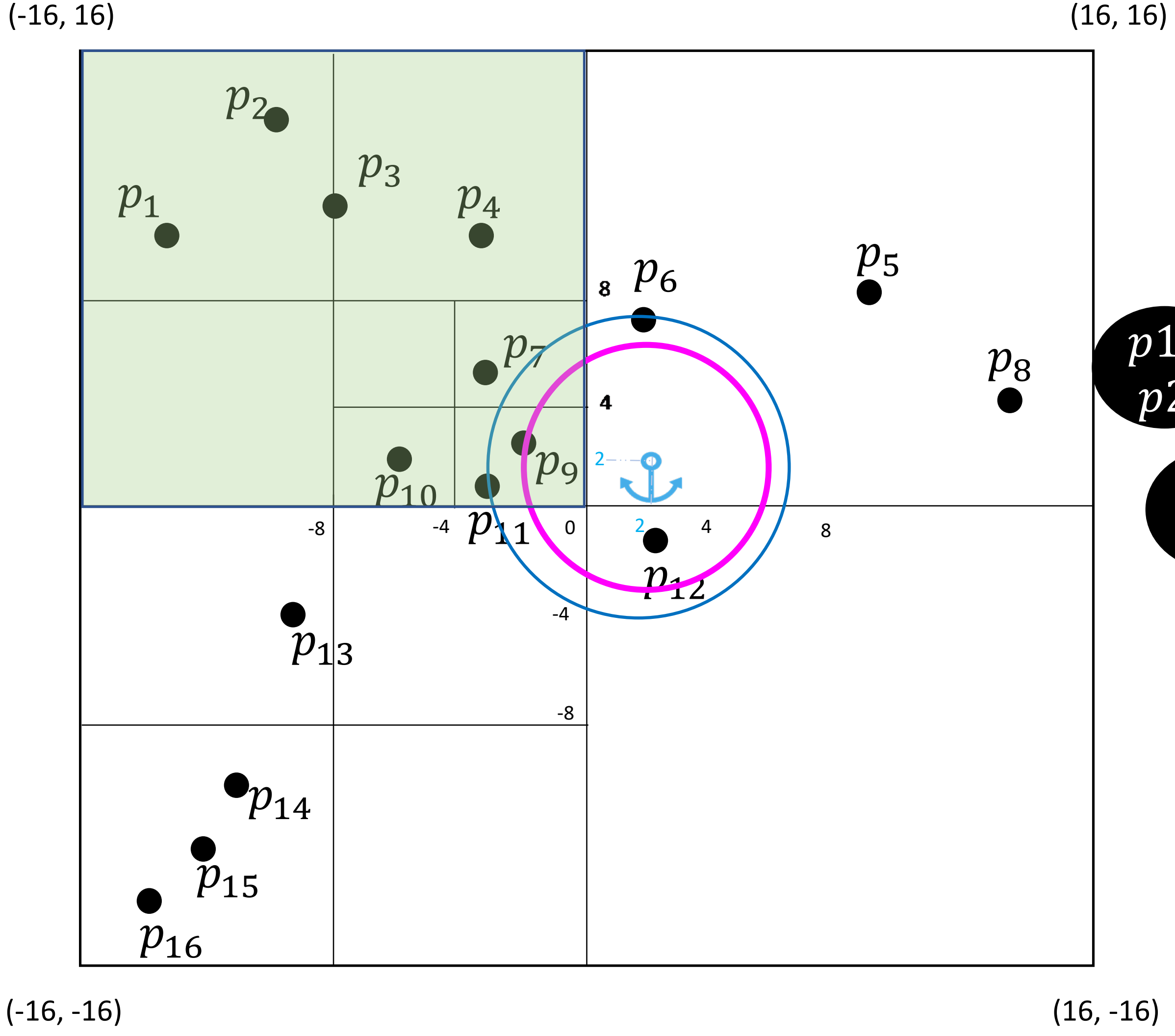
Example of k-NN query in PR-QuadTree

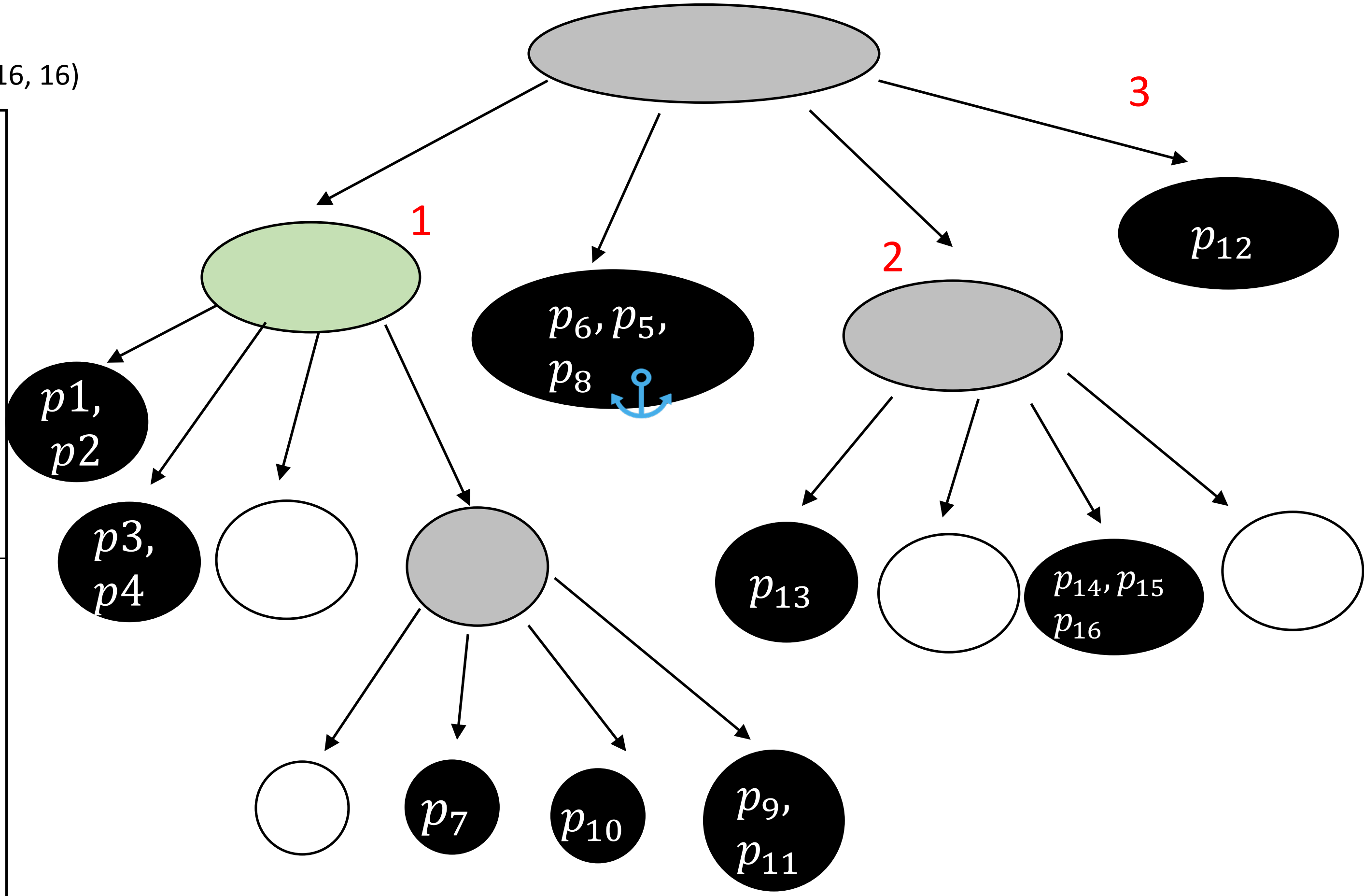
anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3

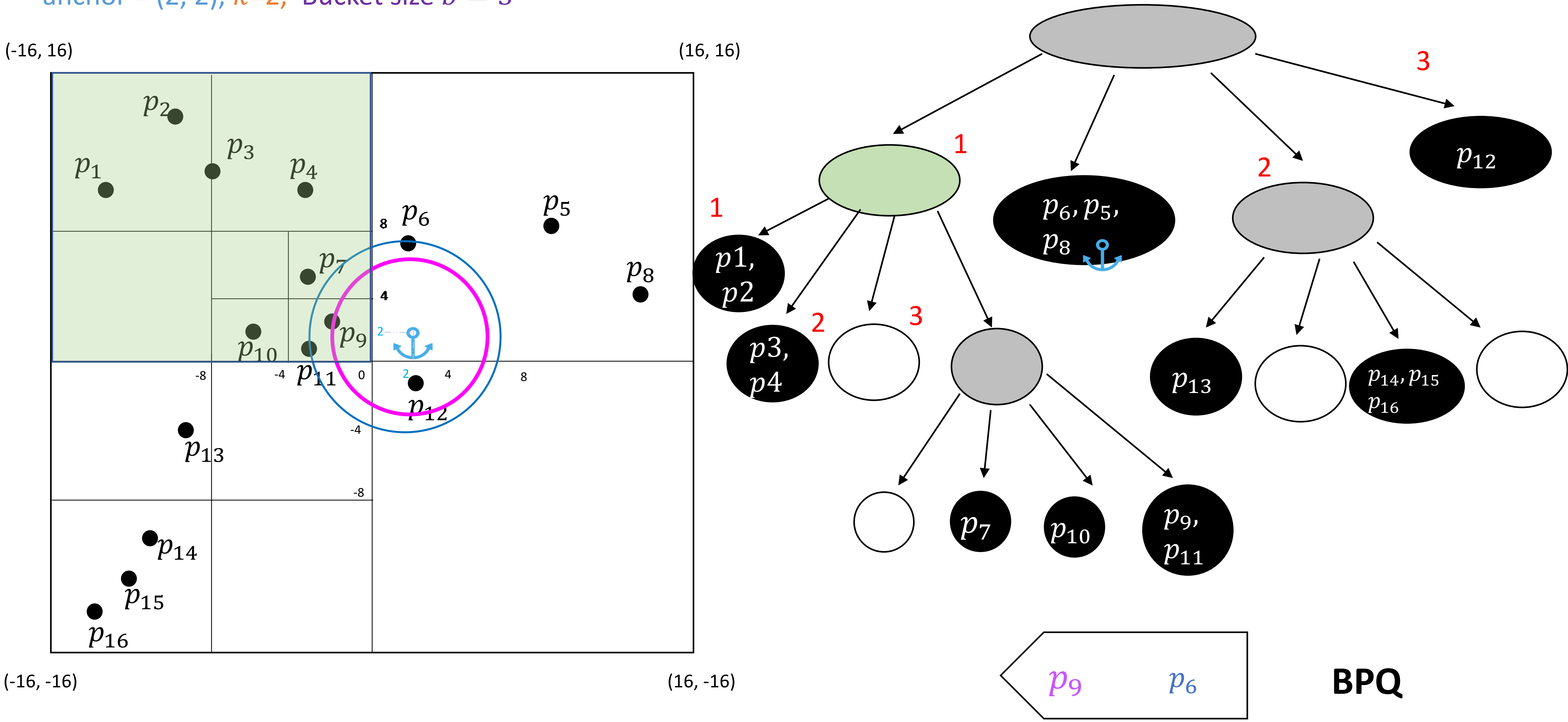




BPQ

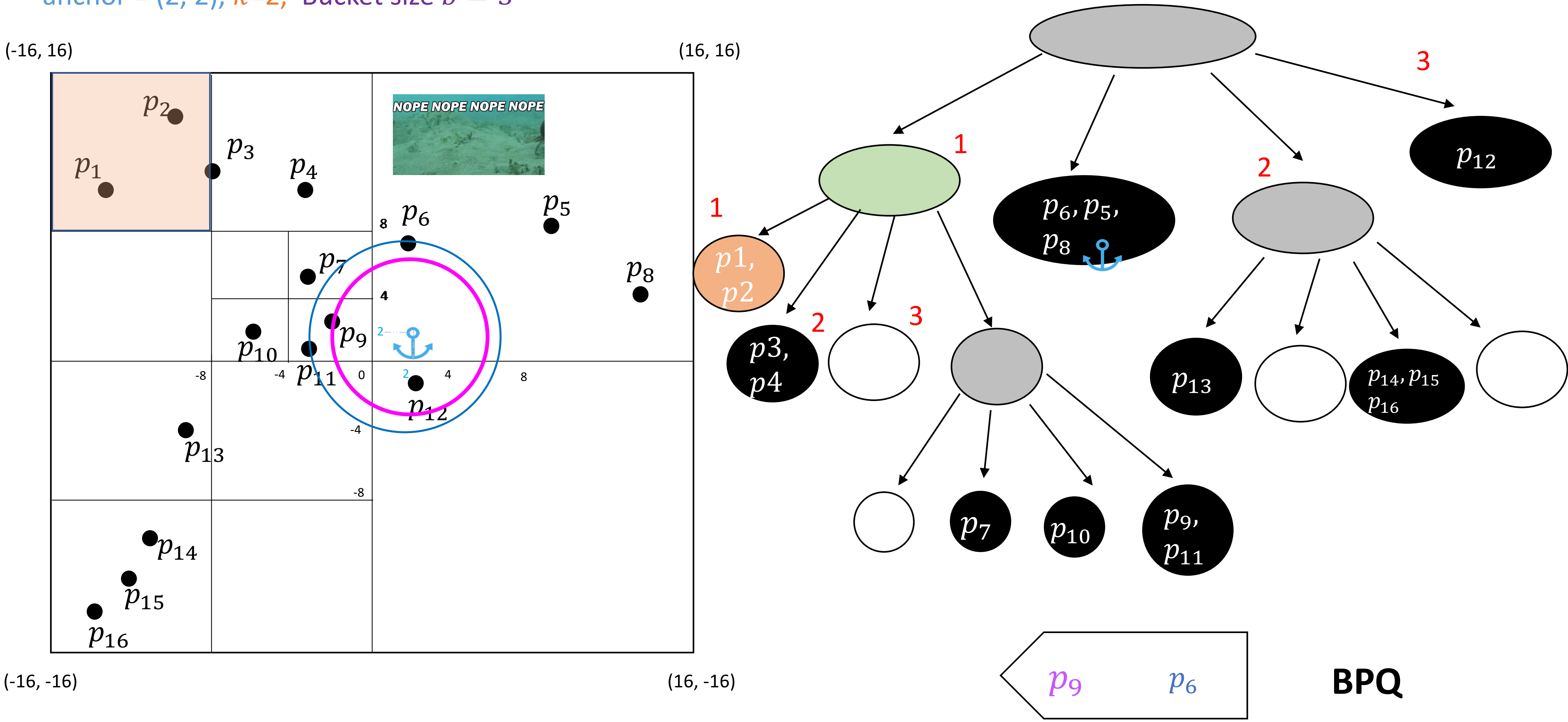
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



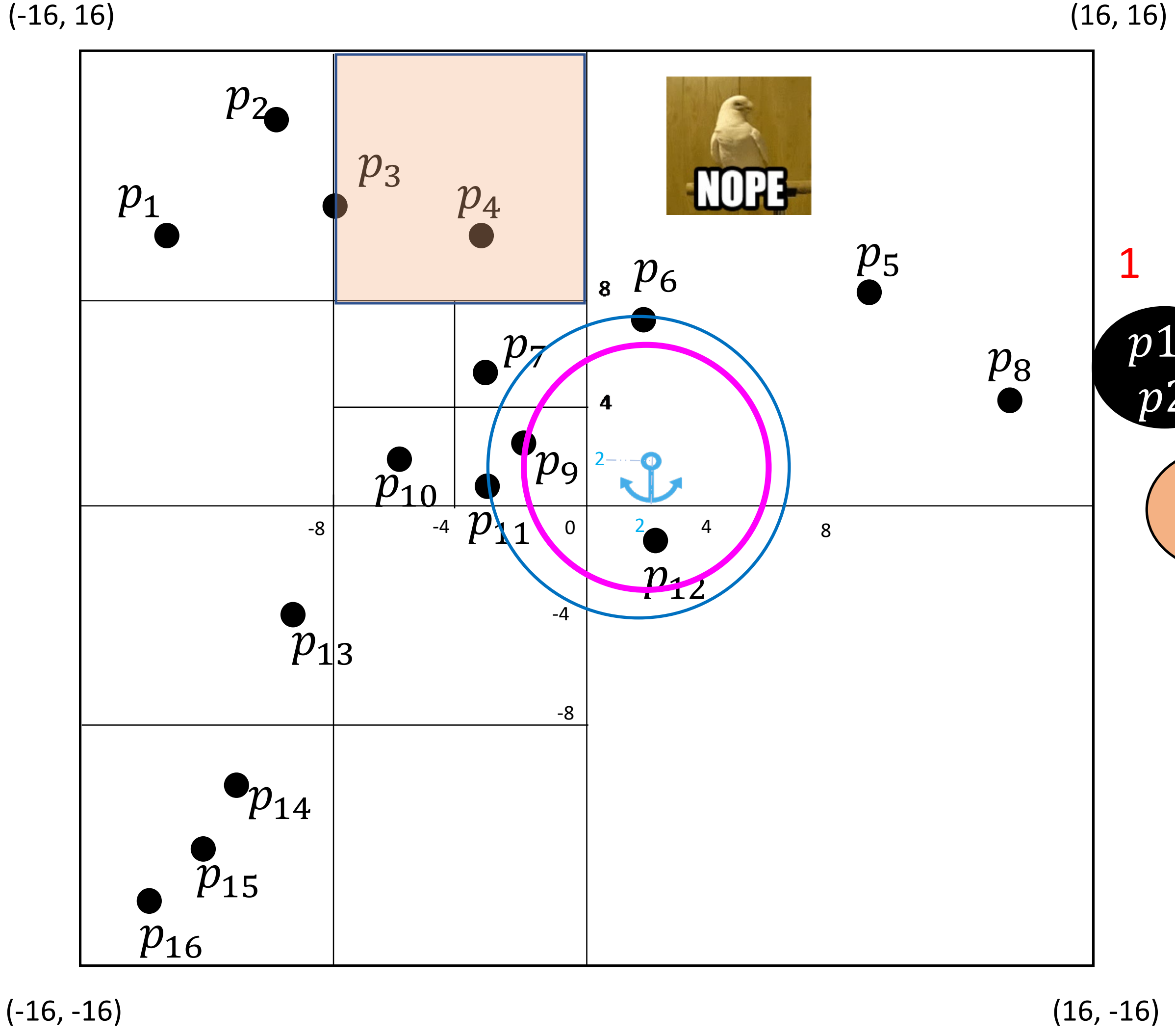
Example of k-NN query in PR-QuadTree

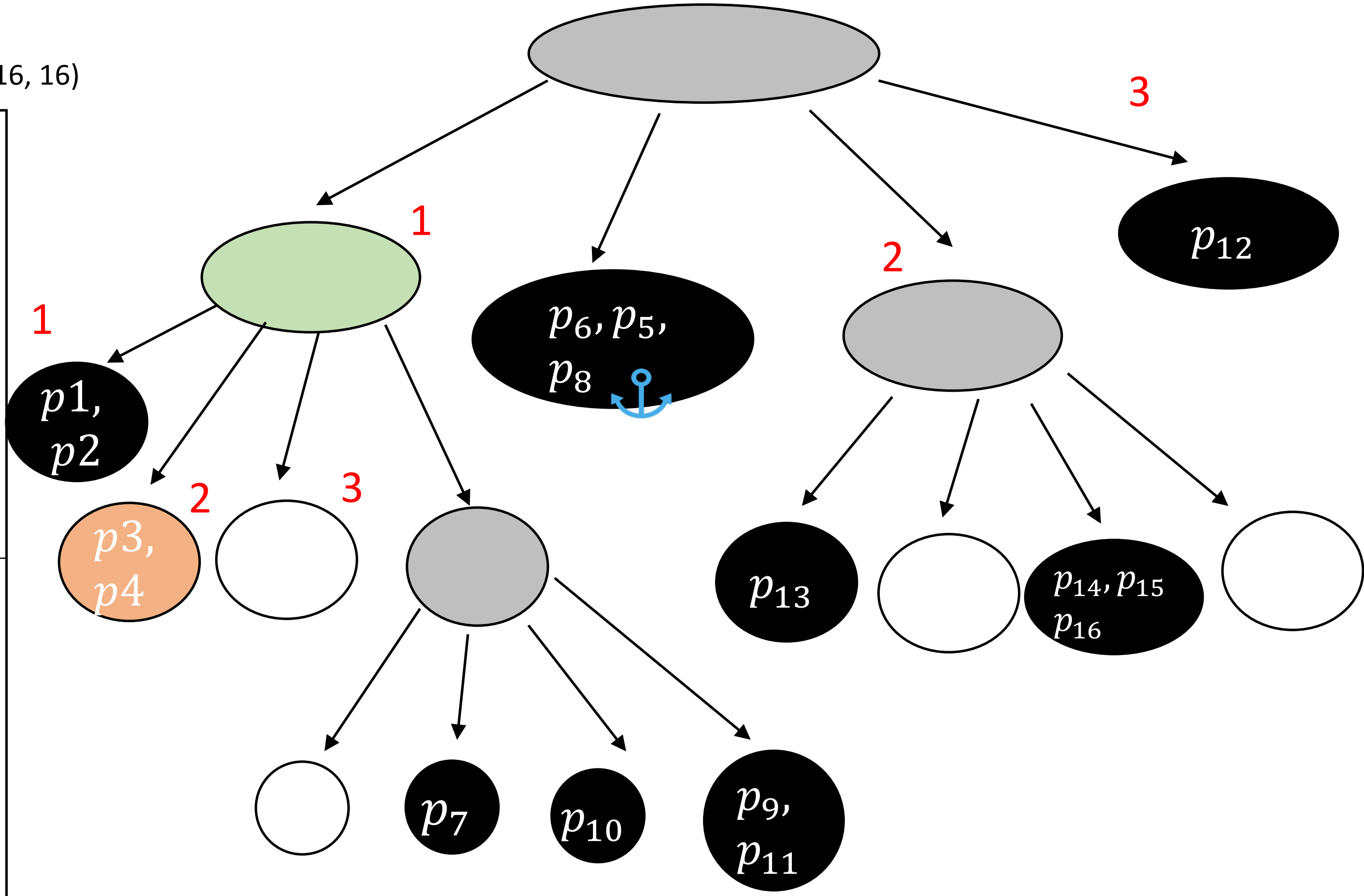
anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



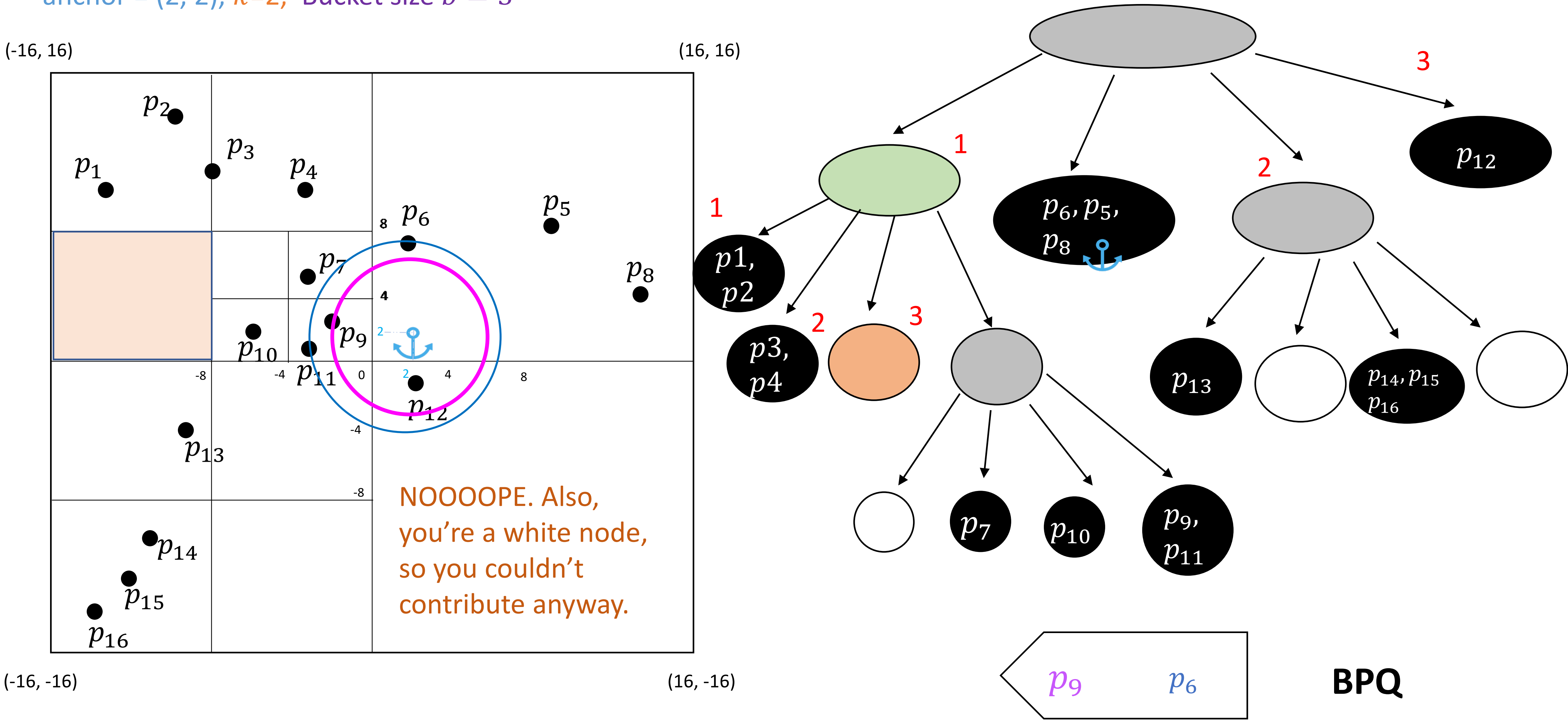


p_9 p_6

BPQ

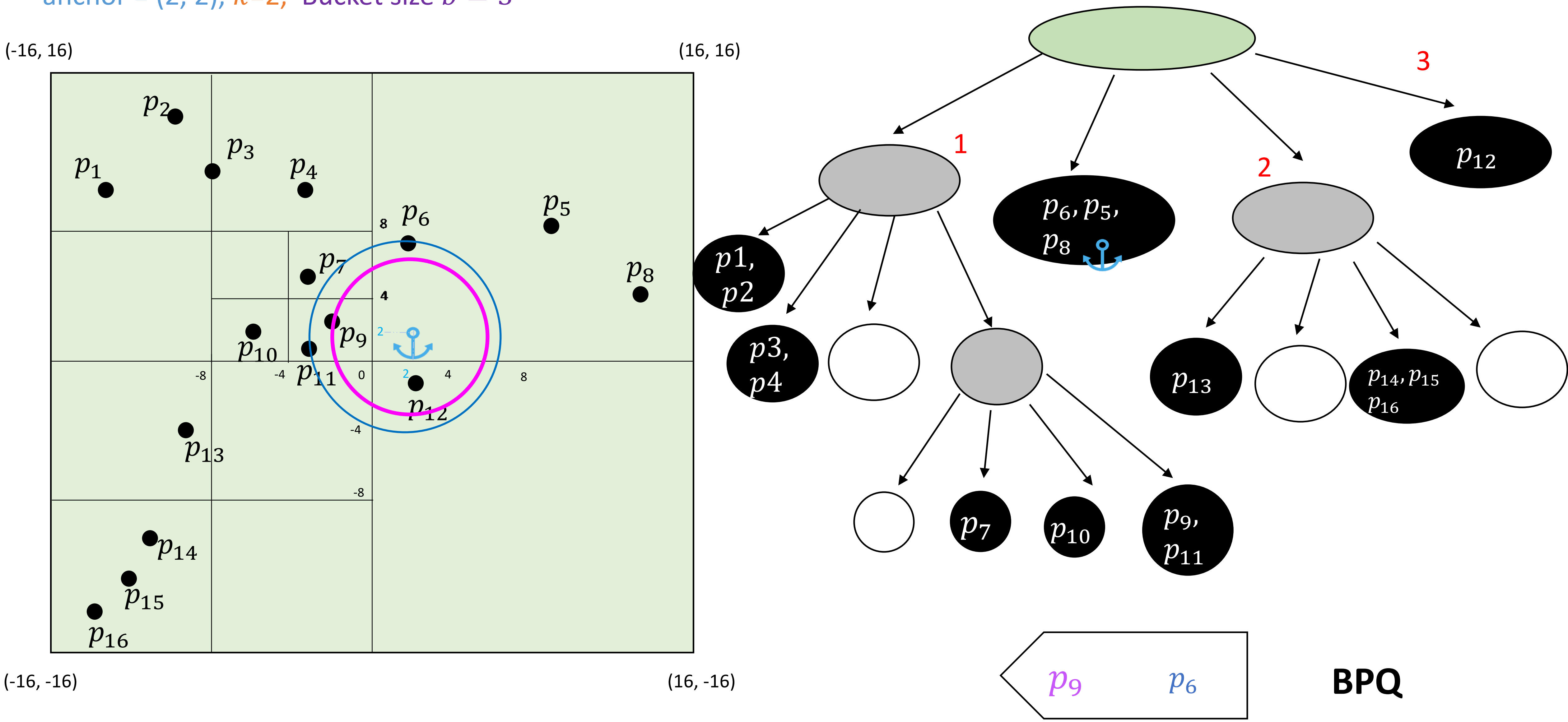
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



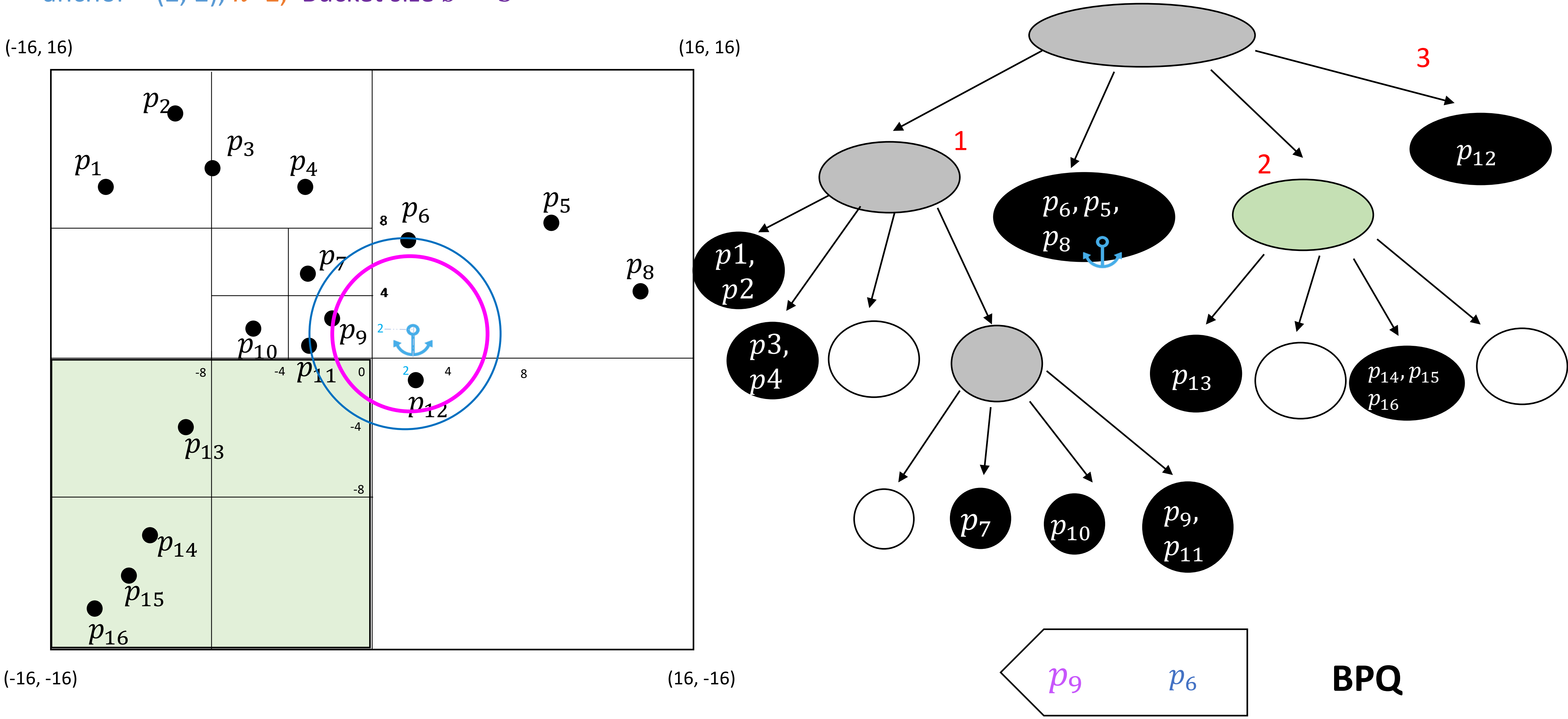
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



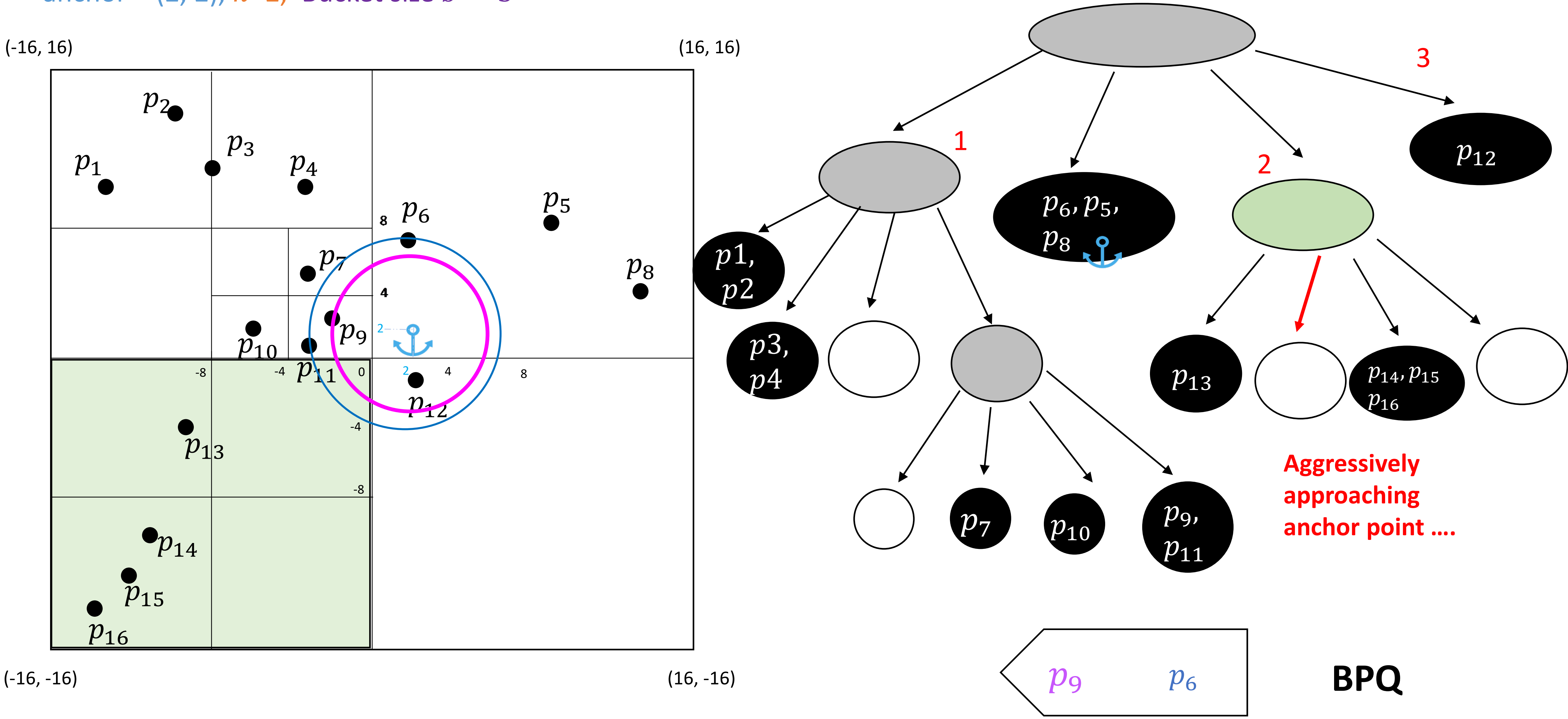
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

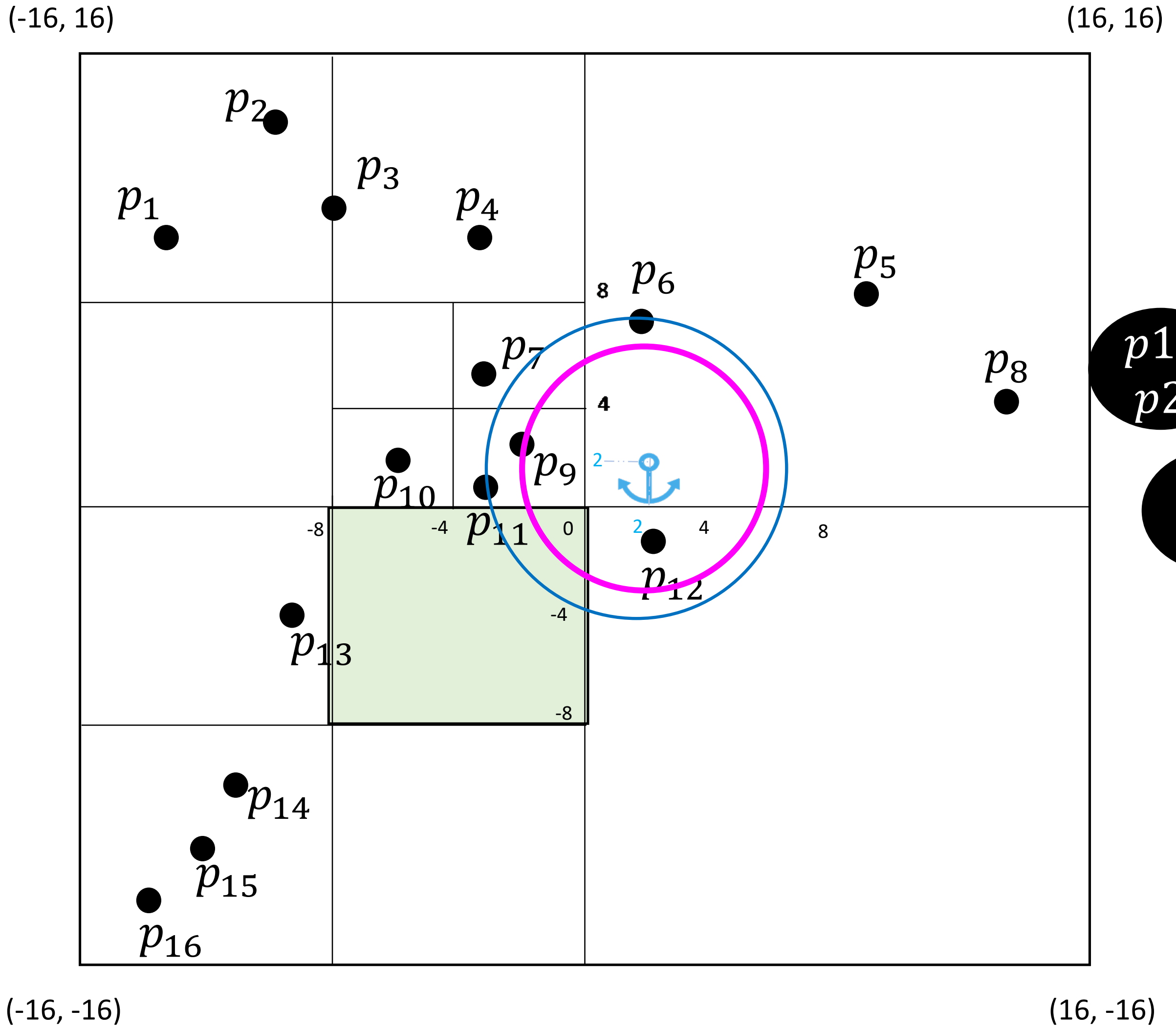
anchor = (2, 2), k=2, Bucket size b = 3

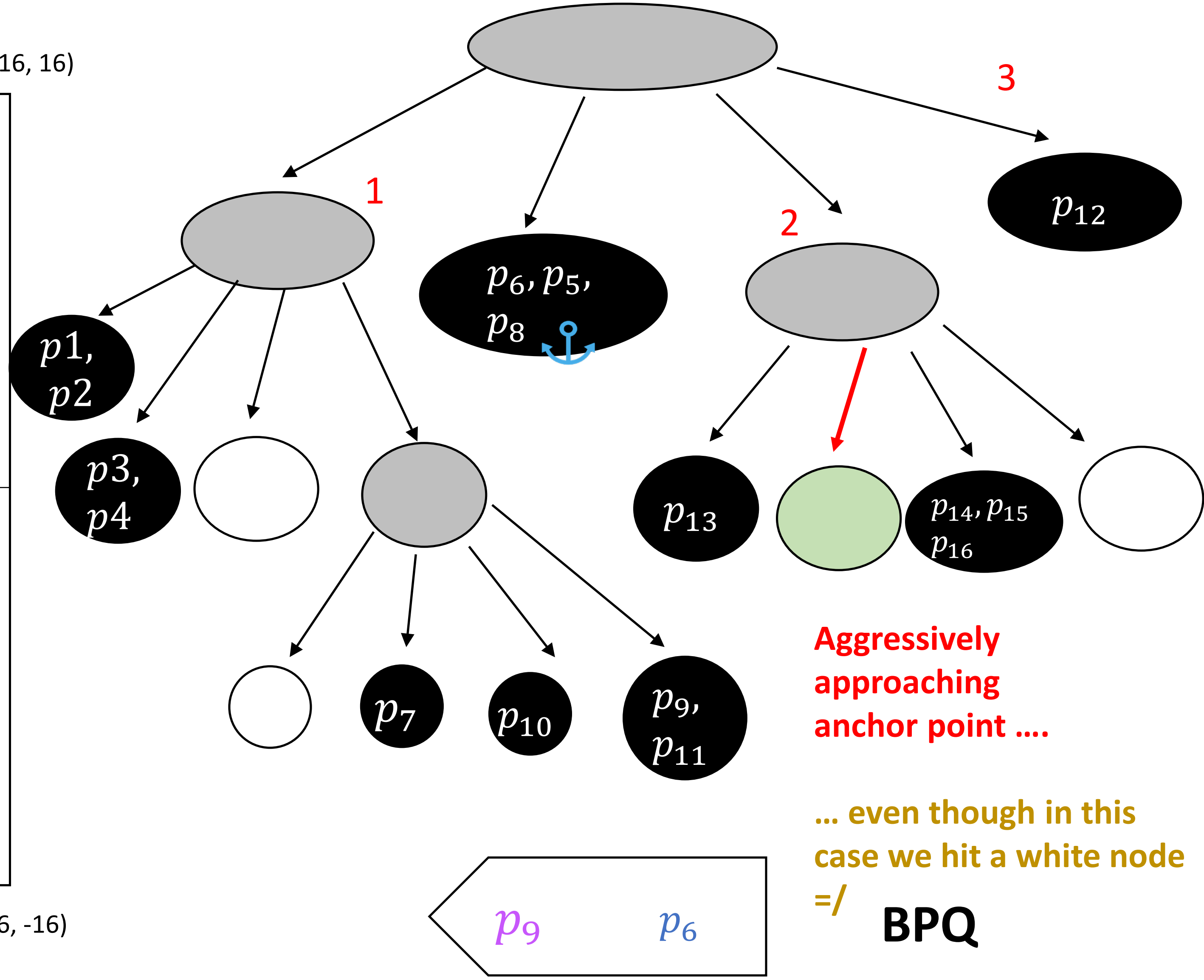


Aggressively approaching anchor point ….

Example of k-NN query in PR-QuadTree

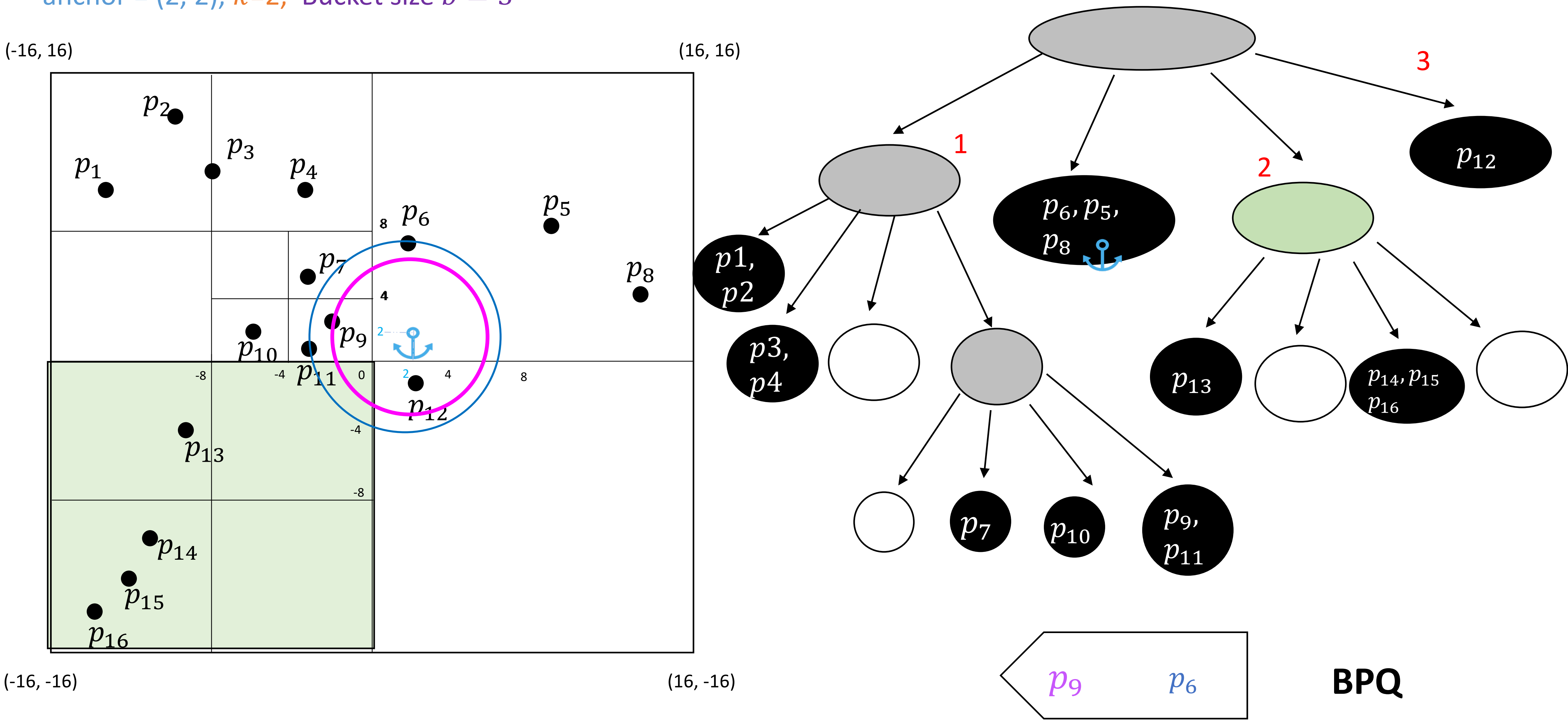
anchor = (2, 2), k=2, Bucket size $b = 3$





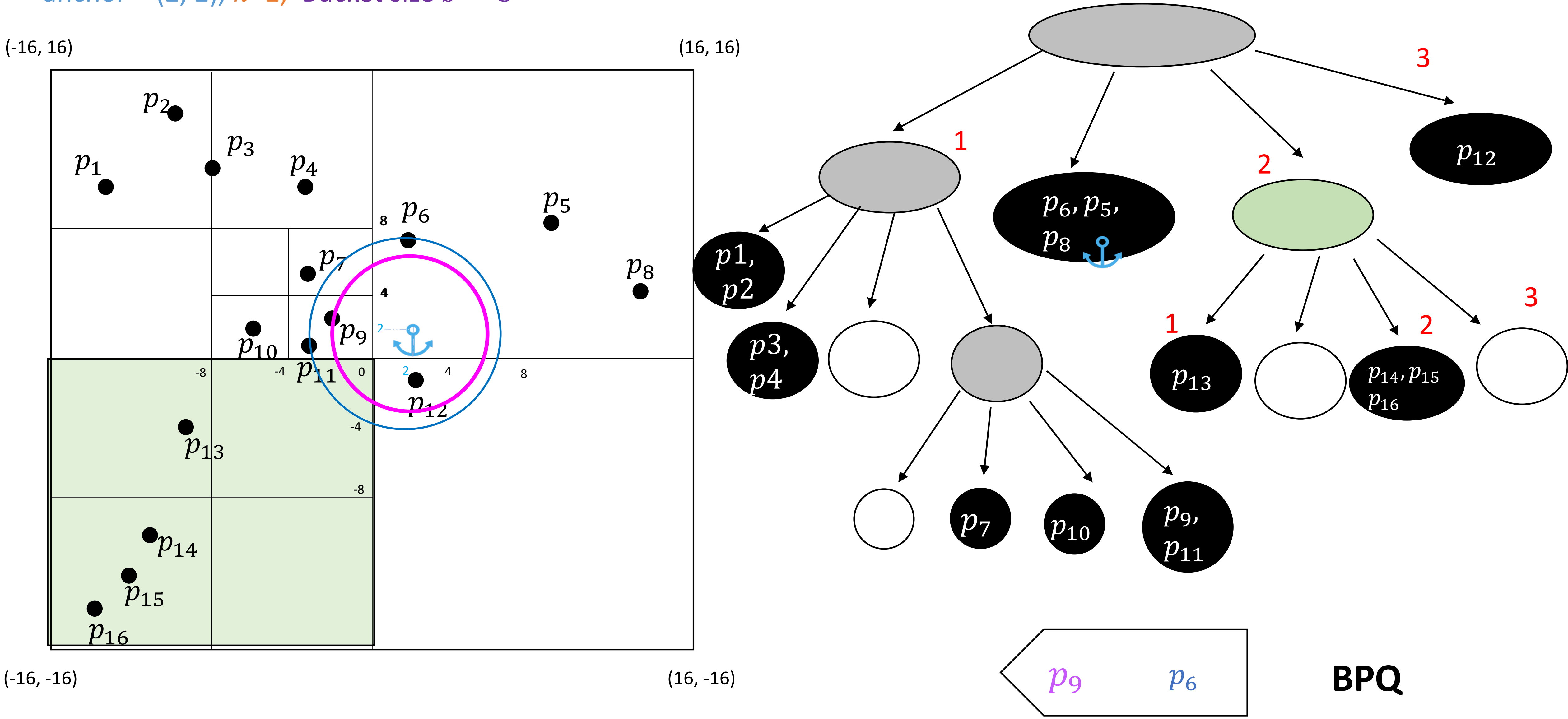
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



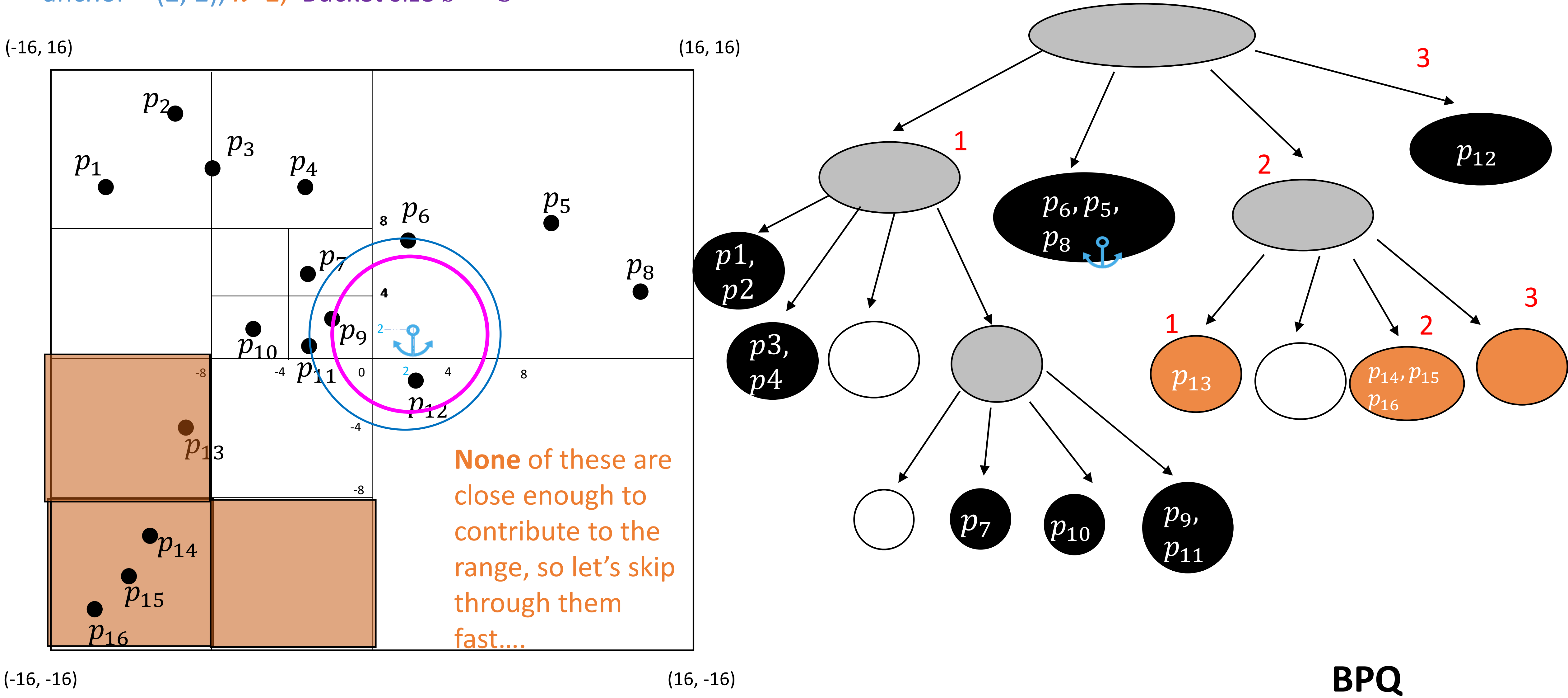
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3

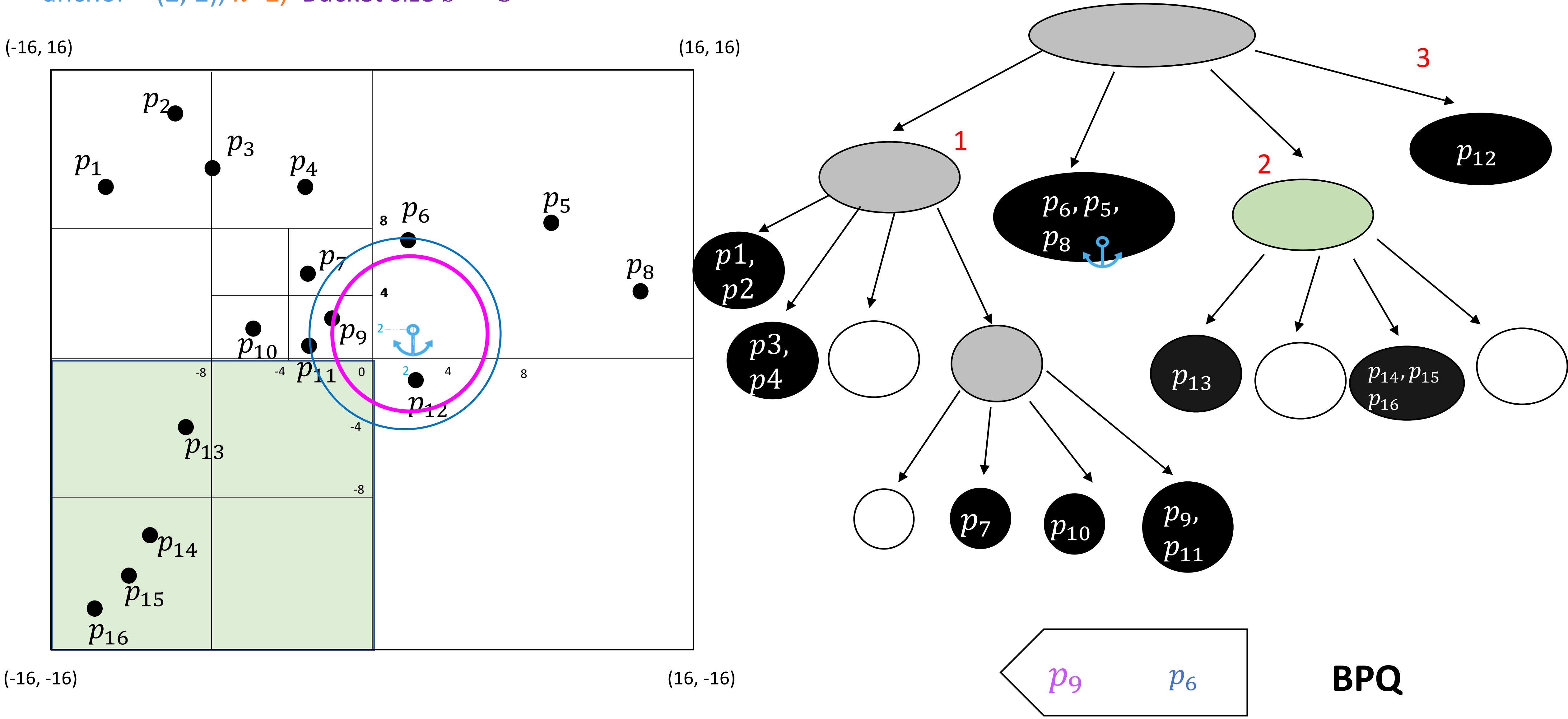


None of these are close enough to contribute to the range, so let's skip through them fast....

BPQ

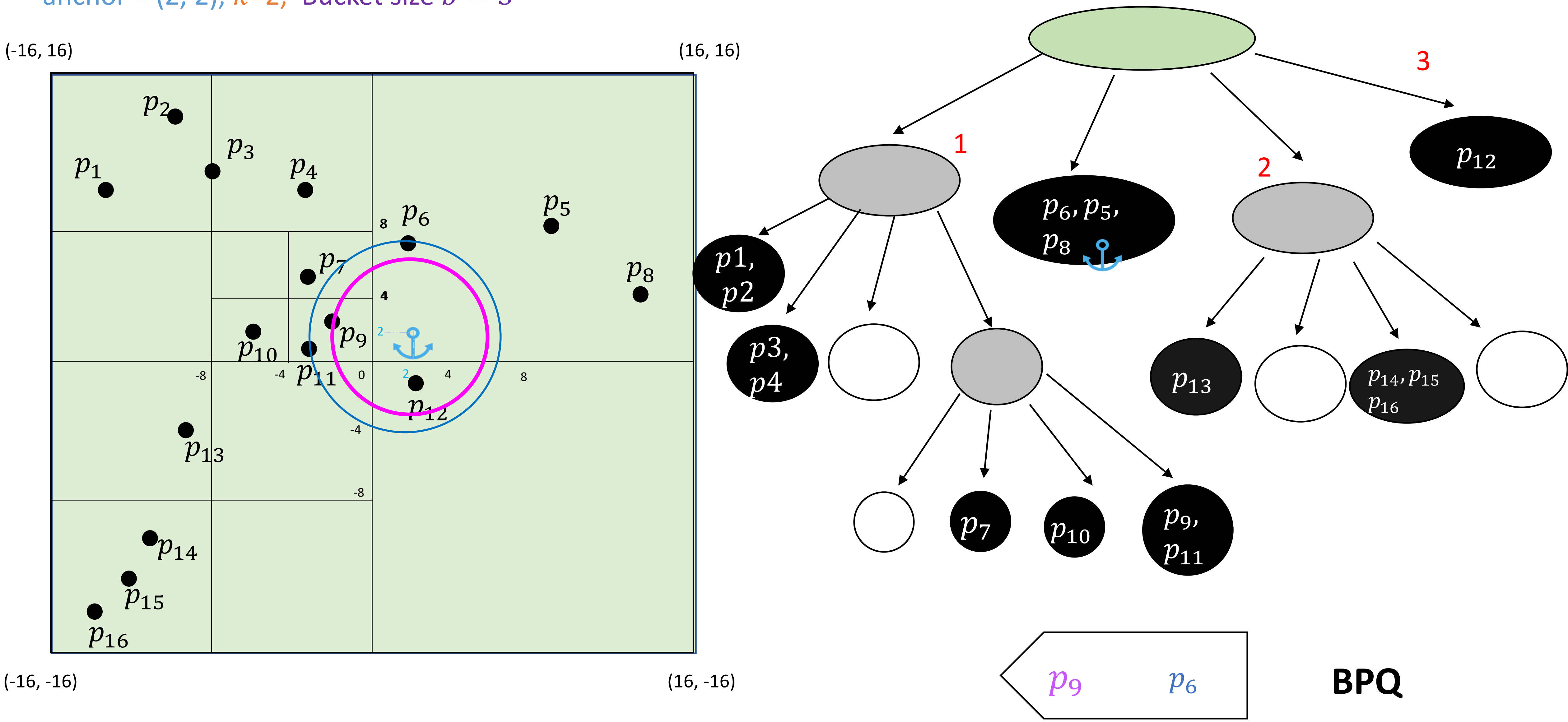
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



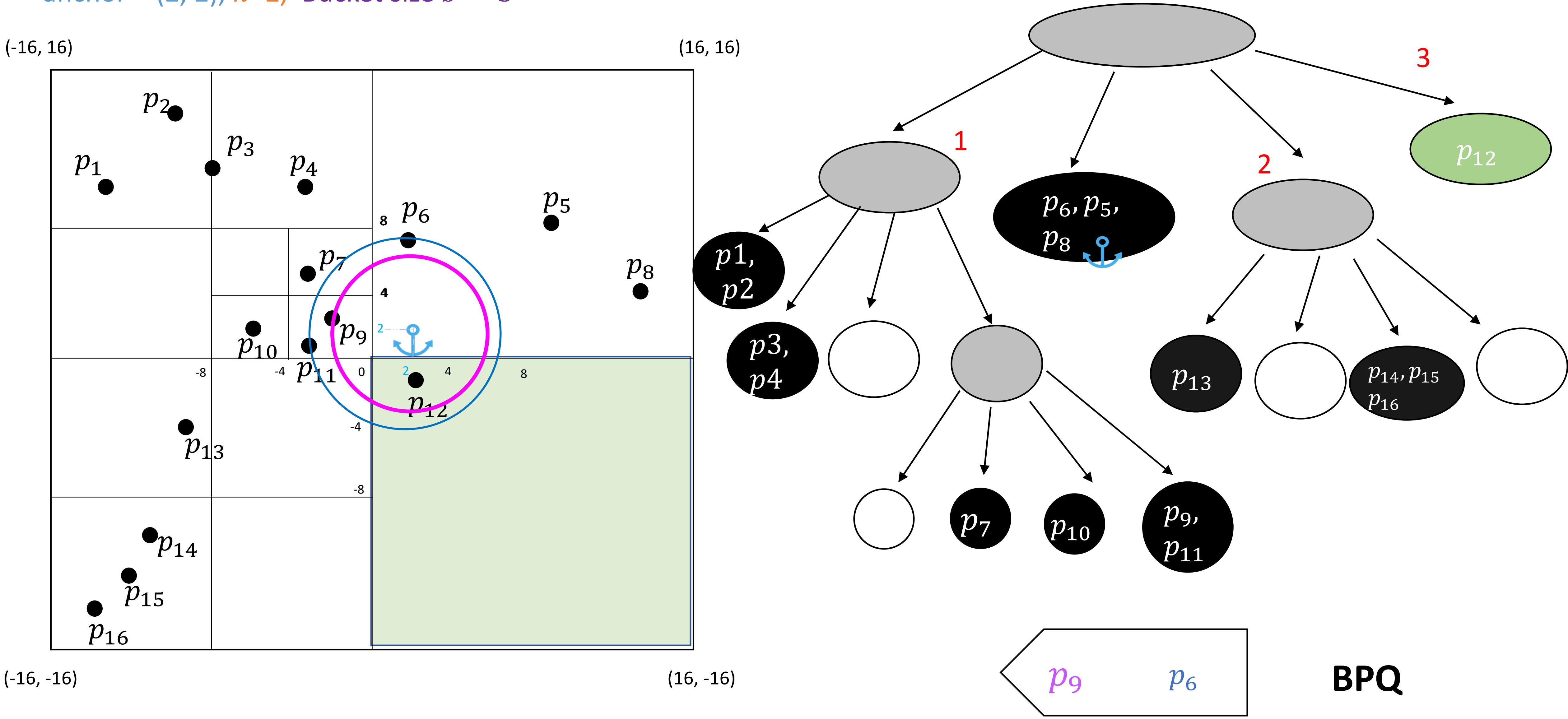
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



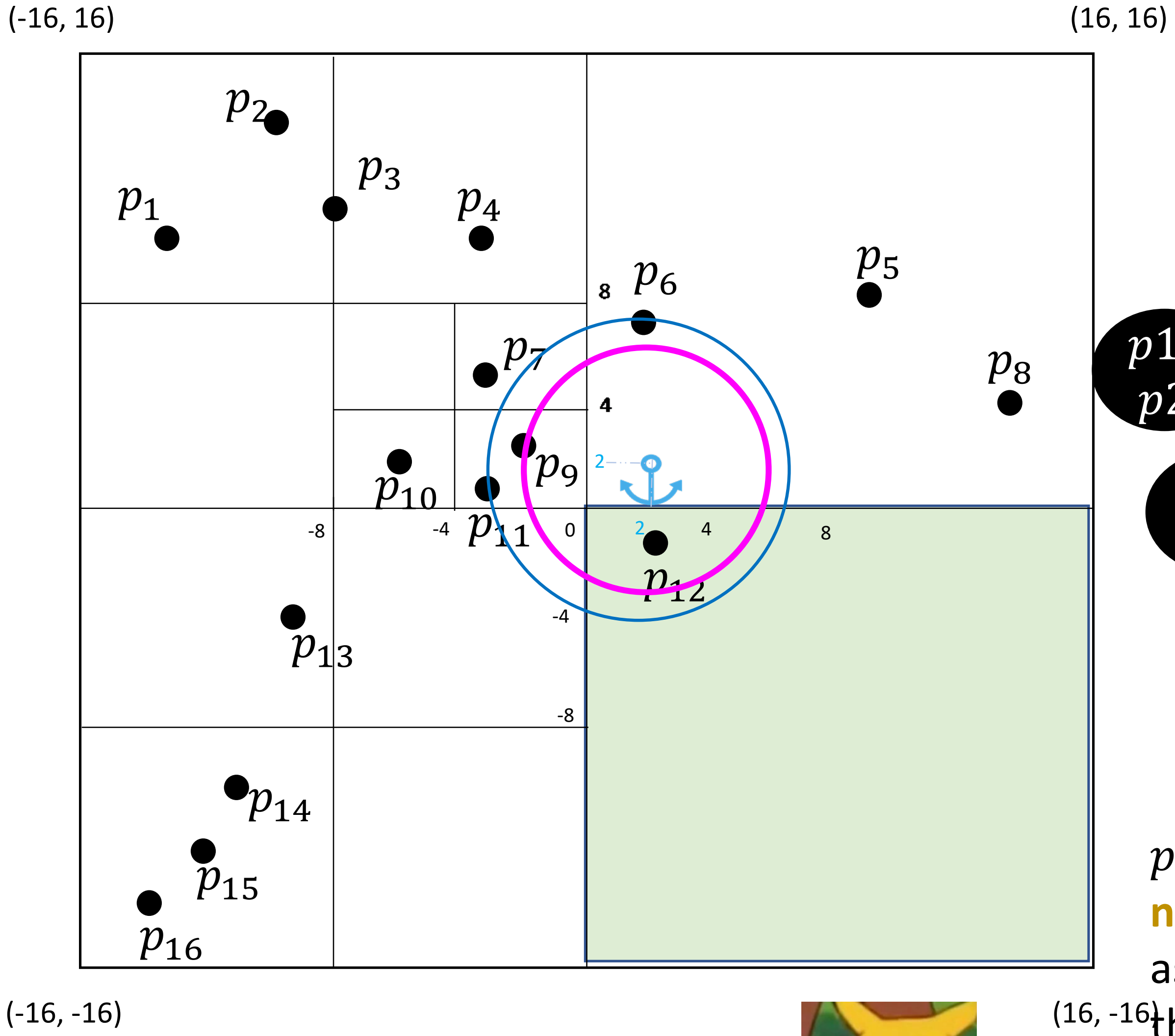
Example of k-NN query in PR-QuadTree

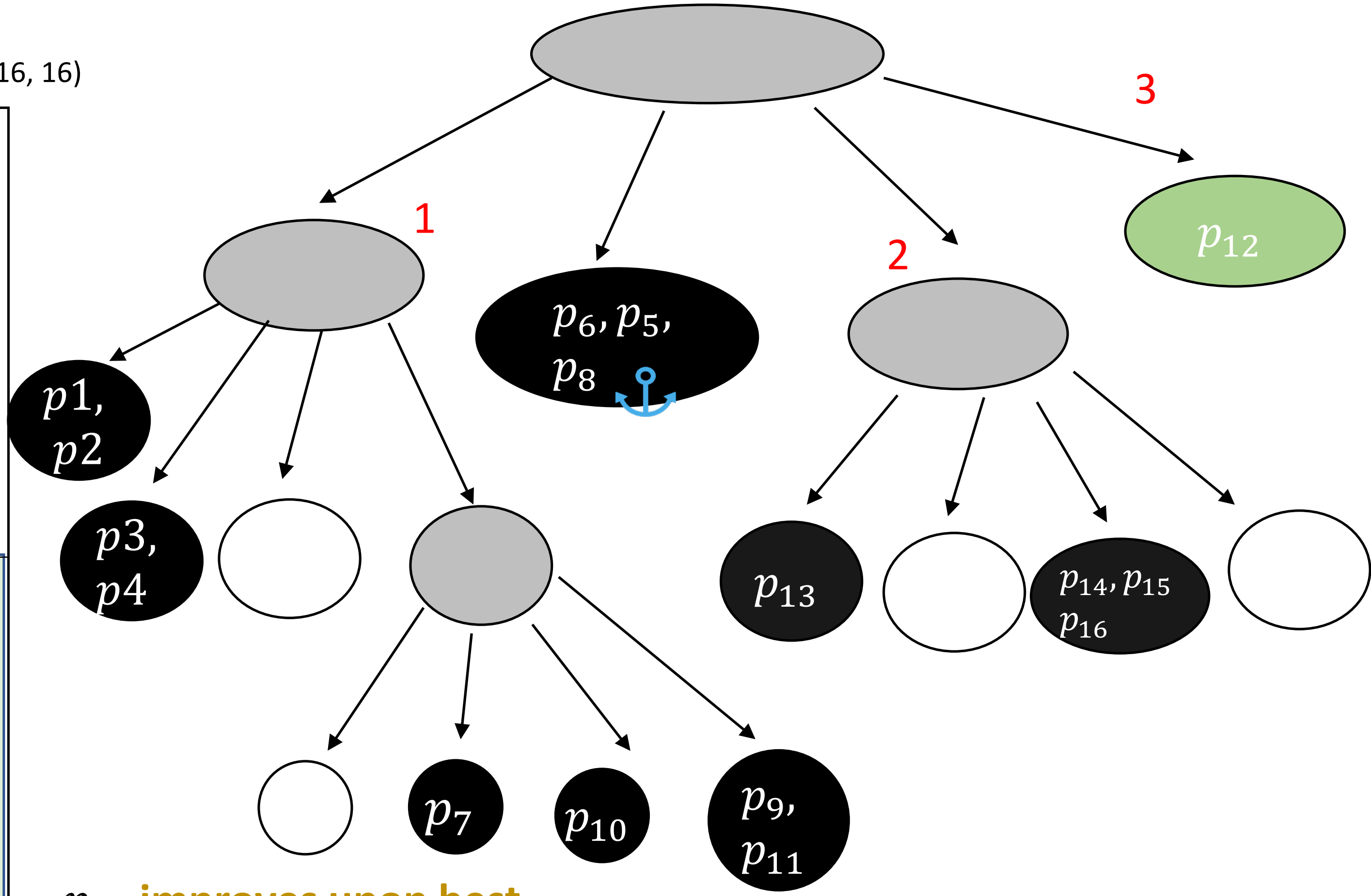
anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



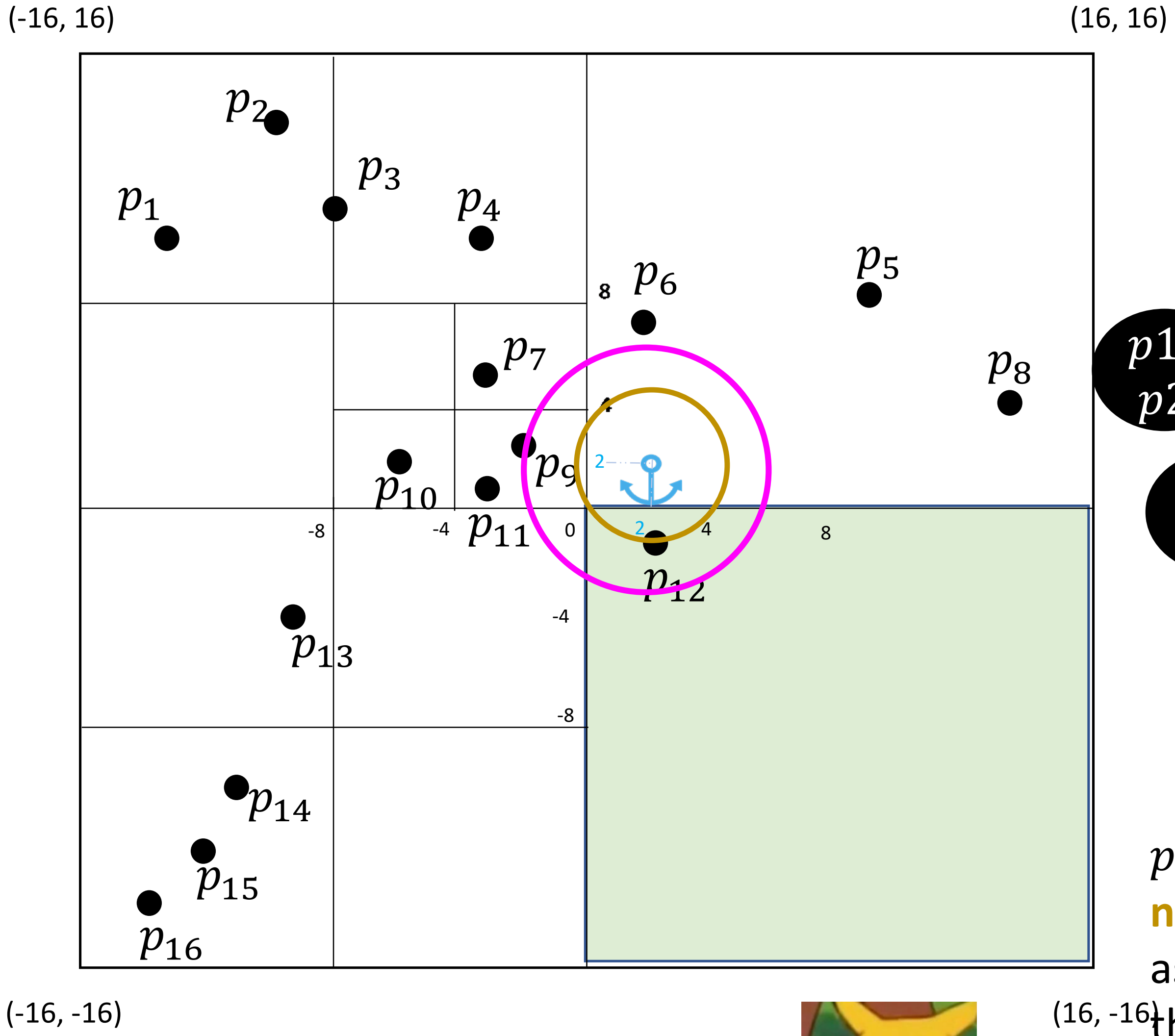


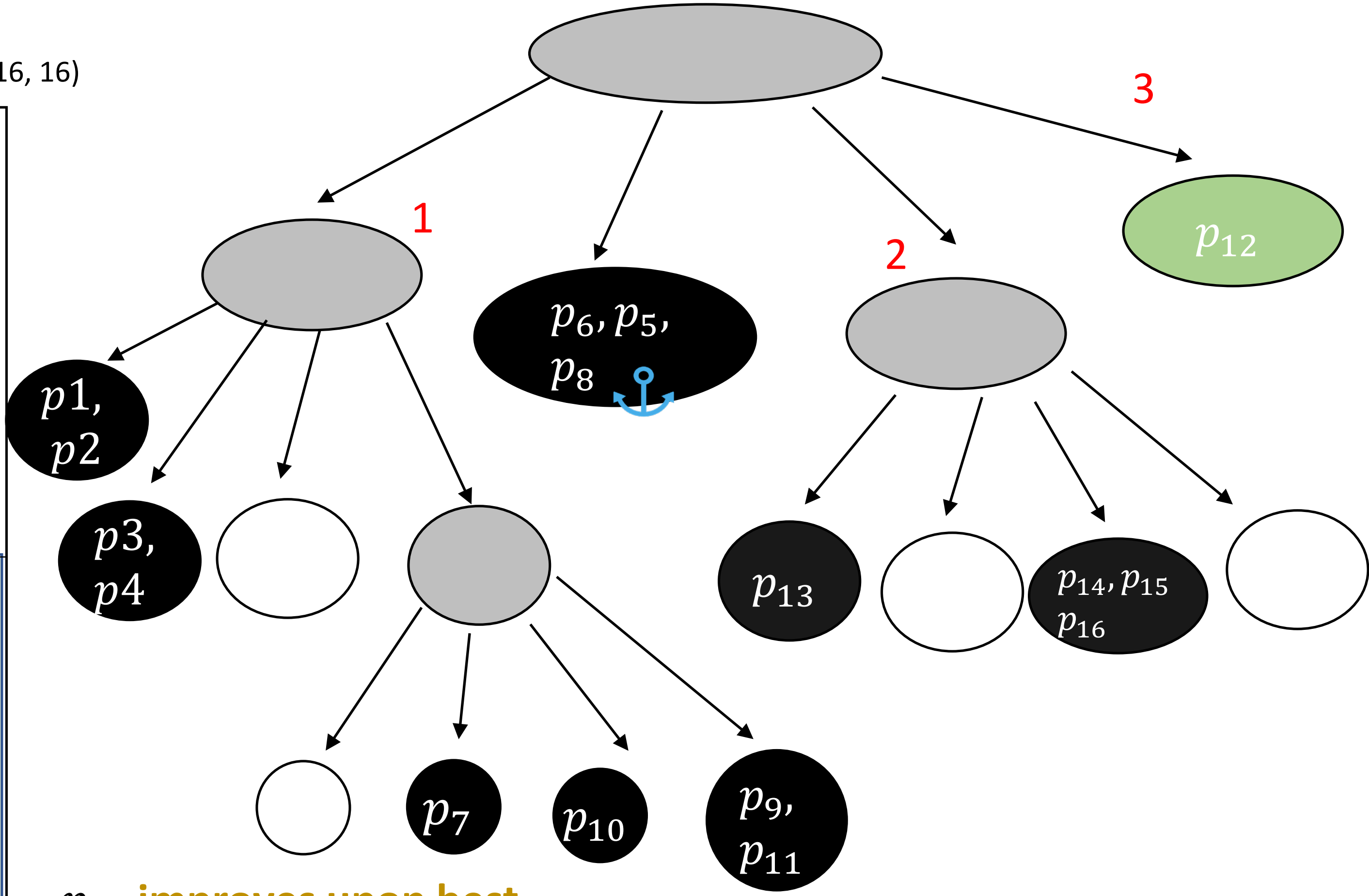
p_{12} **improves upon best neighbor**! Insert into BPQ as new best neighbor, throw away worst neighbor (p_6)!

BPQ

Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



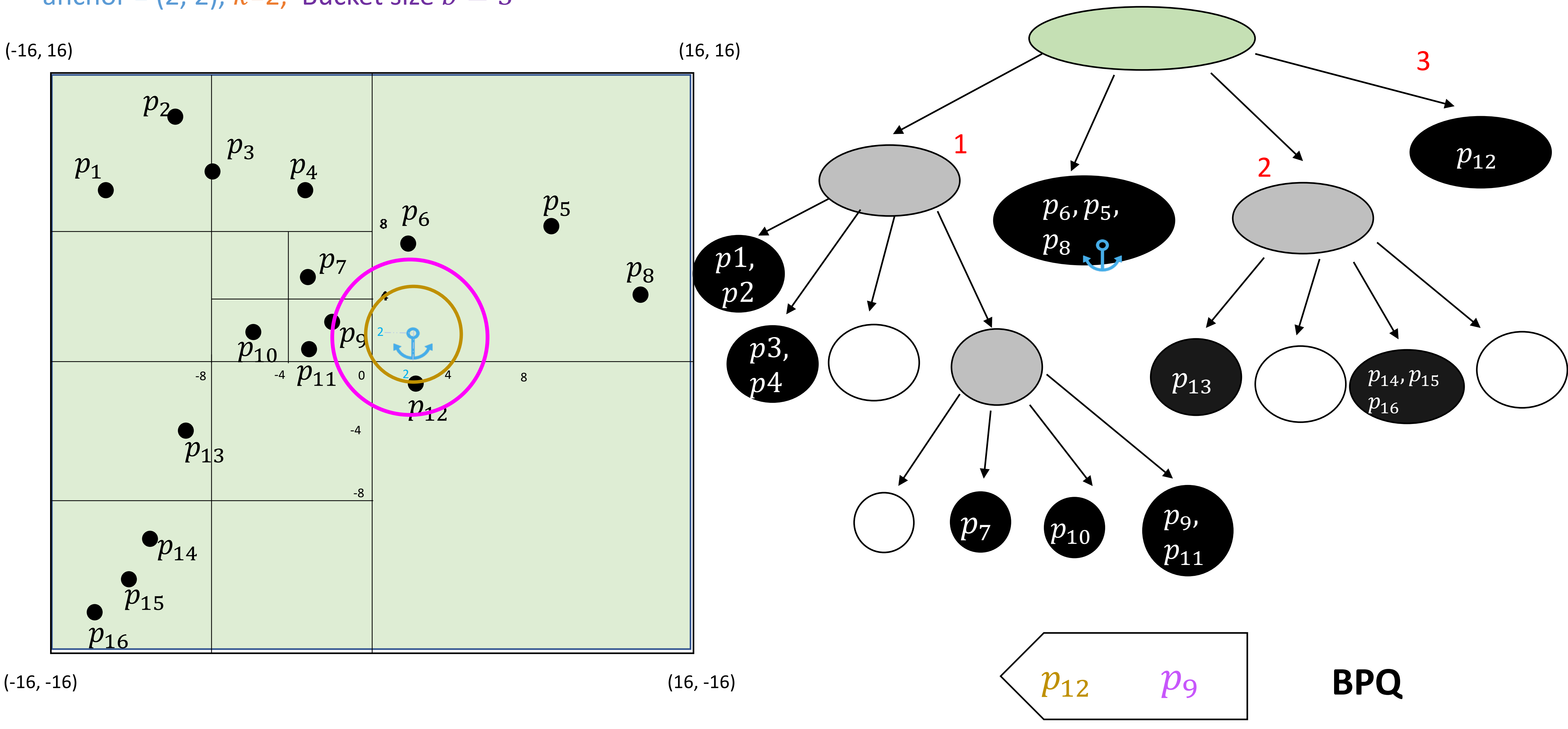


p_{12} **improves upon best neighbor**! Insert into BPQ as new best neighbor, throw away worst neighbor (p_6)!

BPQ

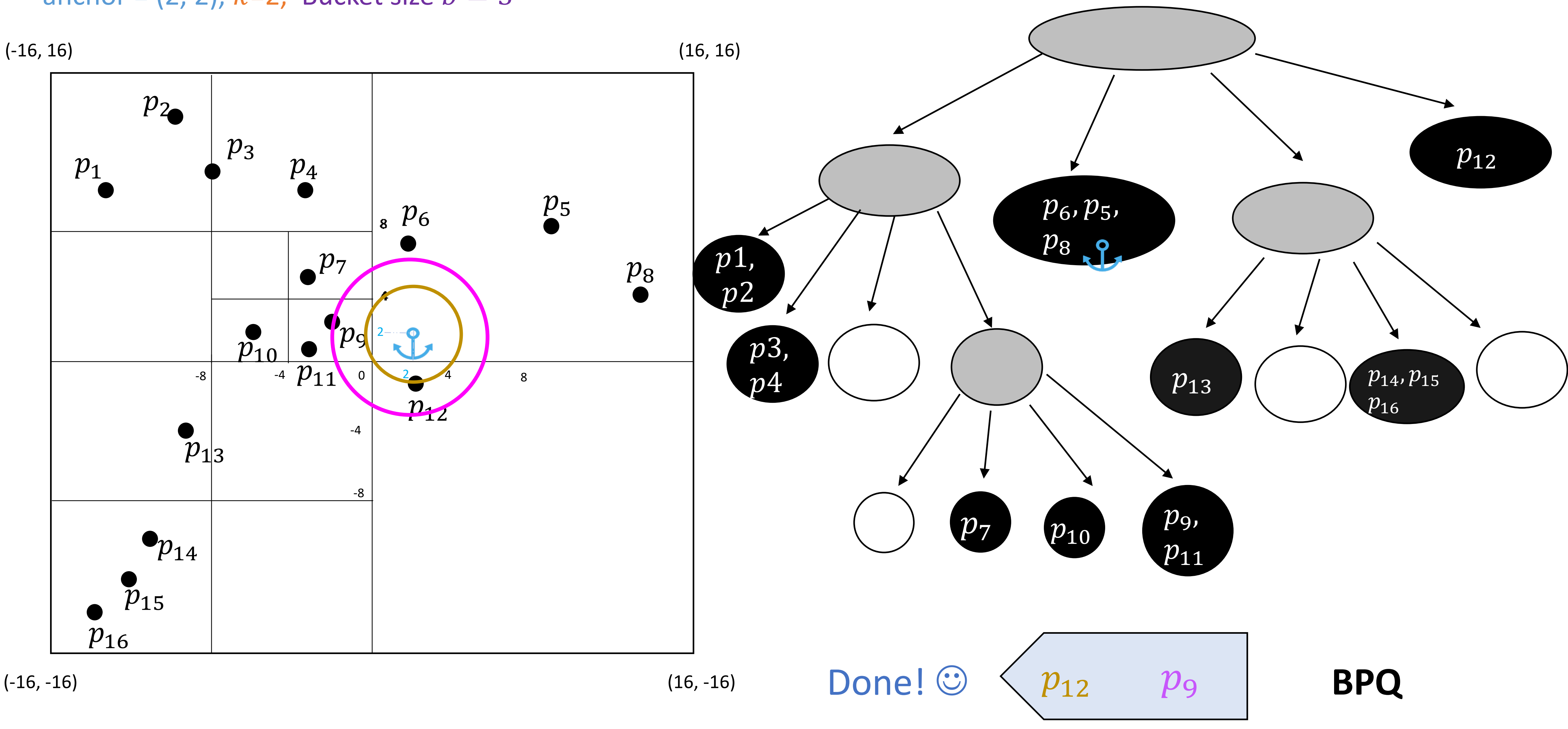
Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



Example of k-NN query in PR-QuadTree

anchor = (2, 2), k=2, Bucket size b = 3



One clarification

- You yourselves might want to implement an improved version of your project where you prioritize the visiting of the residual sub-children based on either area of intersection or proximity to anchor.

One clarification

- You yourselves might want to implement an improved version of your project where you prioritize the visiting of the residual sub-children based on either area of intersection or proximity to anchor.
- You are **definitely allowed** to do this, but we
  a) **Do not require it**, we instead recommend Z-order to keep things simple, and
  b) Suggest you implement a **working version of your project first** to secure 100% and then think about submitting an improved version.

Summary

- In the **descending** phase, you act in exactly the same way as with the KD-Tree in 2 dimensions: aggressively descend towards the point.
  - You get quicker to the anchor point too, since you can move in directions non-orthogonal to axes.

Summary

- In the **descending** phase, you act in exactly the same way as with the KD-Tree in 2 dimensions: aggressively descend towards the point.
    - You get quicker to the anchor point too, since you can move in directions non-orthogonal to axes.
- In the **backtracking** phase, you should loop over your children **in Z-order**, and check whether each one of them can contribute to the solution set.
    - Check the **`protected`** method `doesQuadIntersectAnchorRange()` in `PRQuadNode.java`.
    - Do **not** prioritize based on area of intersection or some other complicated geometrical criteria.

PR-QuadTrees VS KD-Trees

PR-QuadTrees		KD-Trees	
+	-	+	-
$(\forall n \geq 2)[\log_4 n < \log_2 n]$ offers better search	Intractable fanout for large d because of exponential dependency, even for basic operations like insert, search, delete	For dictionary queries and range, can work quite well even in large d.	Nearest neighbor search can be quite inefficient for large d
Faster operations for 2D spaces	Sensitive to relative proximity of points	Easy to understand and code (Binary trees are KD-Trees with $k = 1$!)	Special cases of deletion need care when coding
Insensitive to insertion order	Tricky coding	…	Deletion rather inefficient (calls to `findMin()` for both subtrees in levels with `currDim != desiredDim`)
…	…	….	Sensitive to insertion order