CSE 250

Data Structures

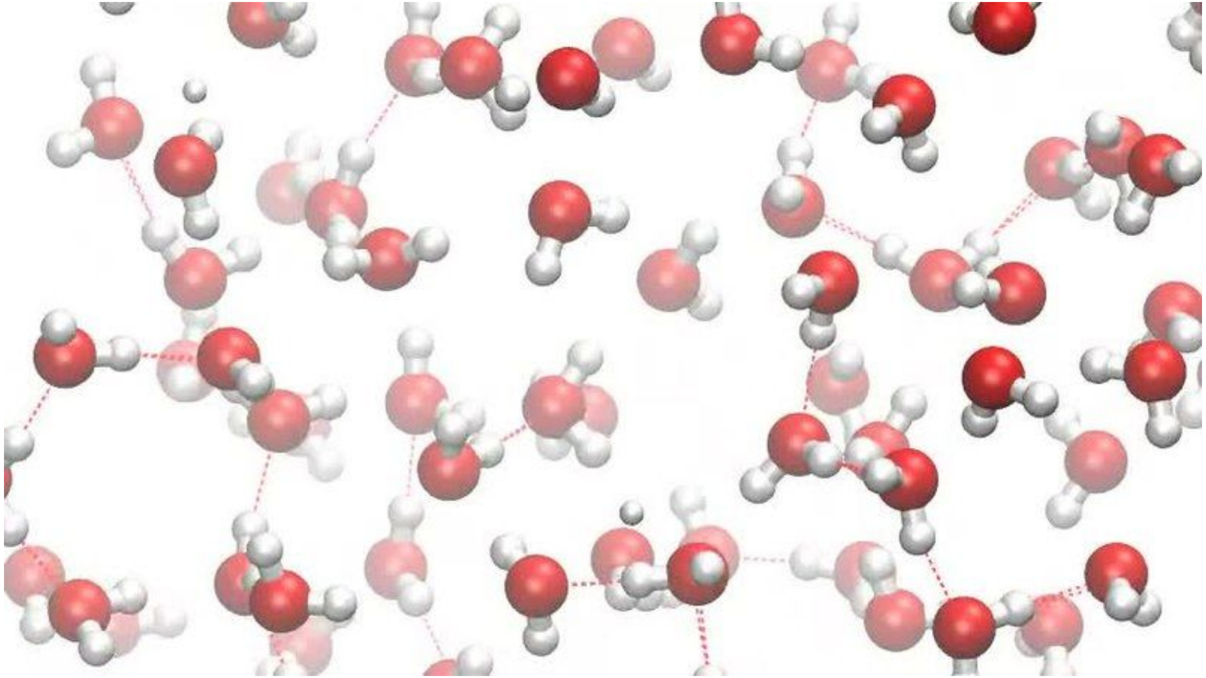
Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Spatial Data Structures

Some Problems are REALLY Big

ESA/Hubble and NASA: http://www.spacetelescope.org/images/potw1006a/

Some Problems are REALLY Small



Molecular Dynamics Simulation of Liquid Water
https://commons.wikimedia.org/wiki/File:A_Molecular_Dynamics_Simulation_of_Liquid_Water_at_298_K.webm

Some Problems are REALLY Detailed

This is **NOT** a photo. It is a computer generated image.

https://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29#/media/File:Glasses_800_edit.png

What do these things have in common?

What do these things have in common?

The have MANY elements (celestial bodies, molecules, mesh cells, etc) which are organized spatially

What do these things have in common?

The have MANY elements (celestial bodies, molecules, mesh cells, etc) which are organized spatially

What "bodies" (other planets, molecules, etc) are close to each other?

Which object(s) will a ray of light bounce/projectile hit?

What objects are closest to a given point?

Which objects fall within a given range?

What do these things have in common?

The have MANY elements (celestial bodies, molecules, mesh cells, etc) which are organized spatially

What "bodies" (other planets, molecules, etc) are close to each other?

Which object(s) will a ray of light bounce/projectile hit?

What objects are closest to a given point?

Which objects fall within a given range?

How can we organize these elements in a way that allows us to efficiently answer these questions?

Related Problems

Mapping

- What's within ½ mile of me?
- What's within 2 minutes of my route?

Games

- What objects are close enough that they might need to be rendered?

Science

- "Big Brain Project": Neuron A fired, so what other neurons are close enough to be stimulated?
- "Astronomy"/"MD": What forces are affecting a particular body, and what forces can we ignore/estimate?

Organizing/Storing Our Data

What data structure have we seen already that lets us efficiently organize/store "sorted" data?

Organizing/Storing Our Data

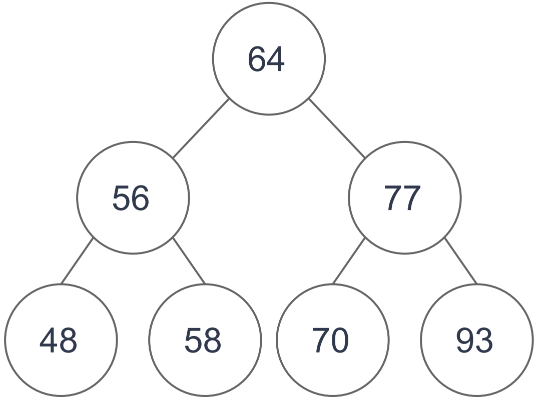
What data structure have we seen already that lets us efficiently organize/store "sorted" data?

Idea: What if we organize our data in a BST

Binary Search Trees (for one dimension)

```
class Node[T <: Comparable](value: T)
{
  /** Guarantee:
      left.value < this.value **/
  val left: Node[T] = Empty

  /** Guarantee:
      right.value >= this.value **/
  val right: Node[T] = Empty
}
```



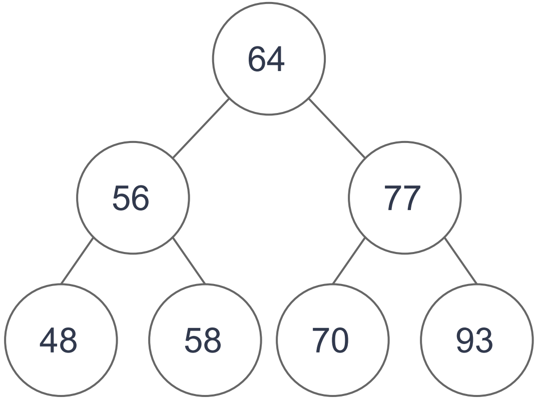
Binary Search Trees (for one dimension)

`Insert`

- Find the right spot: ***O*(depth)**
- Create and insert the node: ***O*(1)**

`Find`

- Find the right node: ***O*(depth)**
- Return the value if it is present: ***O*(1)**



If the tree is balanced, O(depth) = O(log(n))

Multiple Dimensions

This worked for 1-dimensional data…How could we change it to work with 2-dimensional data, ie (Birthday, Zip Code)?

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find me everyone with a specific birthday
2. Find me everyone in a specific zip code
3. Find me everyone that has a specific birthday AND zip code

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find me everyone with a specific birthday
2. Find me everyone in a specific zip code
3. Find me everyone that has a specific birthday AND zip code

Idea 1: BST over birthday

- Operation 2 is ***O*(*n*)**
- Operation 3 is ***O*(log(*n*) + |people sharing a bday|)**

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find me everyone with a specific birthday
2. Find me everyone in a specific zip code
3. Find me everyone that has a specific birthday AND zip code

Idea 1: BST over birthday
- Operation 2 is $\boldsymbol{O(n)}$
- Operation 3 is **$O($ log(n) + |people sharing a bday|)**

Idea 2: BST over zip code
- Operation 1 is $\boldsymbol{O(n)}$
- Operation 3 is **$O($ log(n) + |people sharing a zip|)**

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find me everyone with a specific birthday
2. Find me everyone in a specific zip code
3. Find me everyone that has a specific birthday AND zip code

Idea 1: BST over birthday
- Operation 2 is ***O*(*n*)**
- Operation 3 is ***O*(log(*n*) + |people sharing a bday|)**

Idea 2: BST over zip code
- Operation 1 is ***O*(*n*)**
- Operation 3 is ***O*(log(*n*) + |people sharing a zip|)**

Idea 3: BST over birthday, then zip (lexical order)
- Operation 2 is <u>still</u> ***O*(*n*)**

Why did it fail?

Ideas 1 & 2

BST works by grouping “nearby” values together in the same subtree….

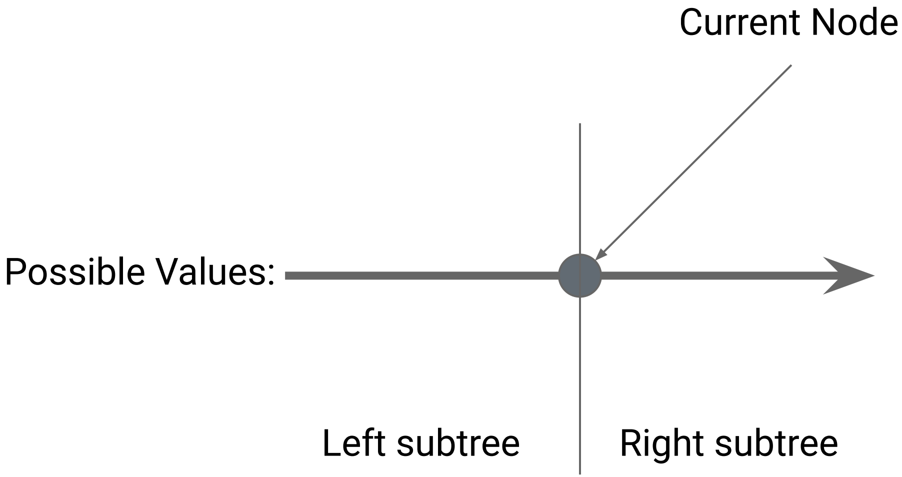
… but “near” in one dimension says nothing about the other!

Idea 3

BST works by partitioning the data…

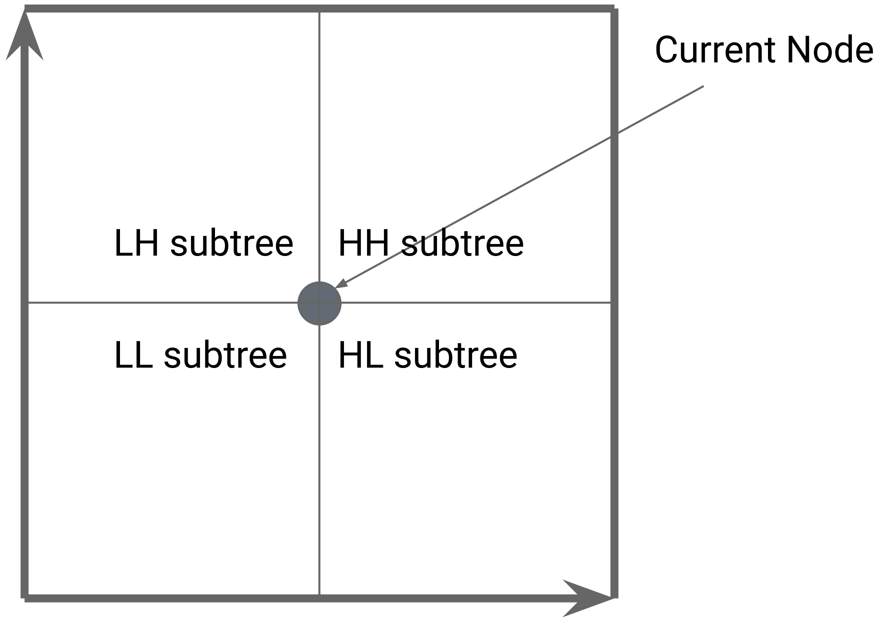
… but lexical order partitions fully on one dimension before partitioning on the other.

Attempt 1 - Partition on BOTH dimensions

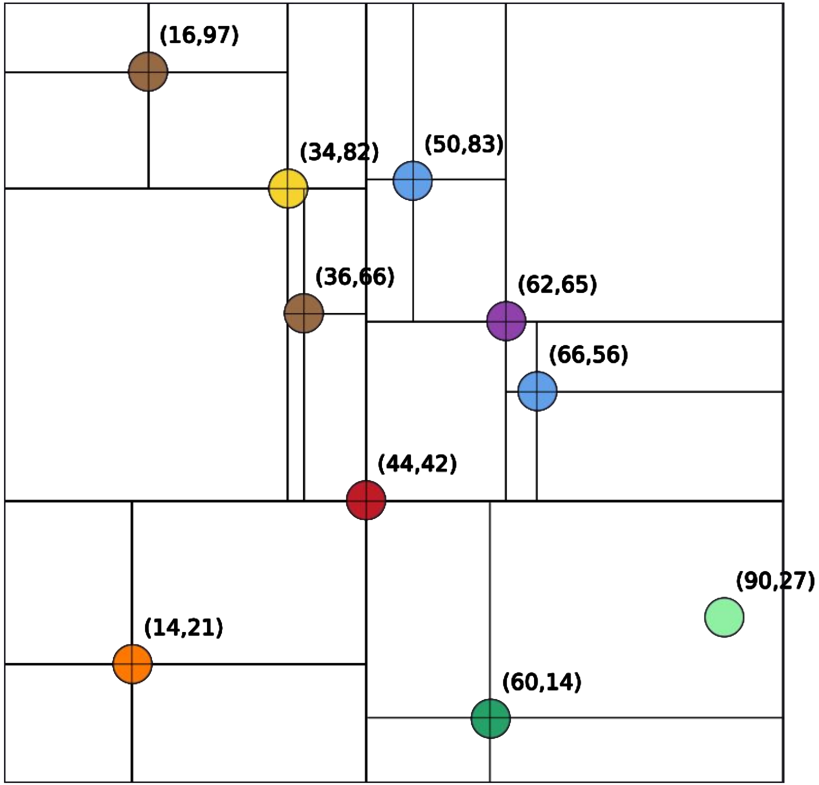


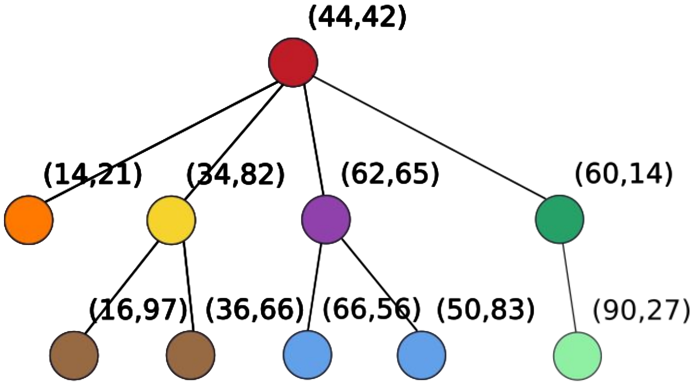
Attempt 1 - Partition on BOTH dimensions

Possible Values:



Each Node has 4 Children





Each Node has 4 Children

“Binary” Search Tree

- Bin - Prefix meaning “2”
- Each node has (at most) 2 children

“Quadary” Search Tree

- Quad - Prefix meaning 4
- Each node has (at most) 4 children
- Usually say: “Quad-Tree” instead

Quad Trees - Find Node

```
def findNode(x: Int, y: Int): Node[T] = {
  var current = root

  while(current.isDefined && (current.x != x || current.y != y) ){
    if(current.x < x){
      if(current.y < y){ current = current.llChild }
      else             { current = current.lhChild }
    } else {
      if(current.y < y){ current = current.hlChild }
      else             { current = current.hhChild }
    }
  }

  return current
}
```

Quad Trees - Find Node

```
def findNode(x: Int, y: Int): Node[T] = {
  var current = root

  while(current.isDefined && (current.x != x || current.y != y) ){
    if(current.x < x){
      if(current.y < y){ current = current.llChild }
      else             { current = current.lhChild }
    } else {
      if(current.y < y){ current = current.hlChild }
      else             { current = current.hhChild }
    }
  }

  return current
}
```

What's the complexity?

Quad Trees - Find Node

```
def findNode(x: Int, y: Int): Node[T] = {
  var current = root

  while(current.isDefined && (current.x != x || current.y != y) ){
    if(current.x < x){
      if(current.y < y){ current = current.llChild }
      else             { current = current.lhChild }
    } else {
      if(current.y < y){ current = current.hlChild }
      else             { current = current.hhChild }
    }
  }

  return current
}
```

What's the complexity? ***O*(*d*)**

Quad Trees - Other Operations

`insert(x, y, value)`

- Find placeholder spot corresponding to (x, y): ***O*(d)**
- Create and inject new node: ***O*(1)**

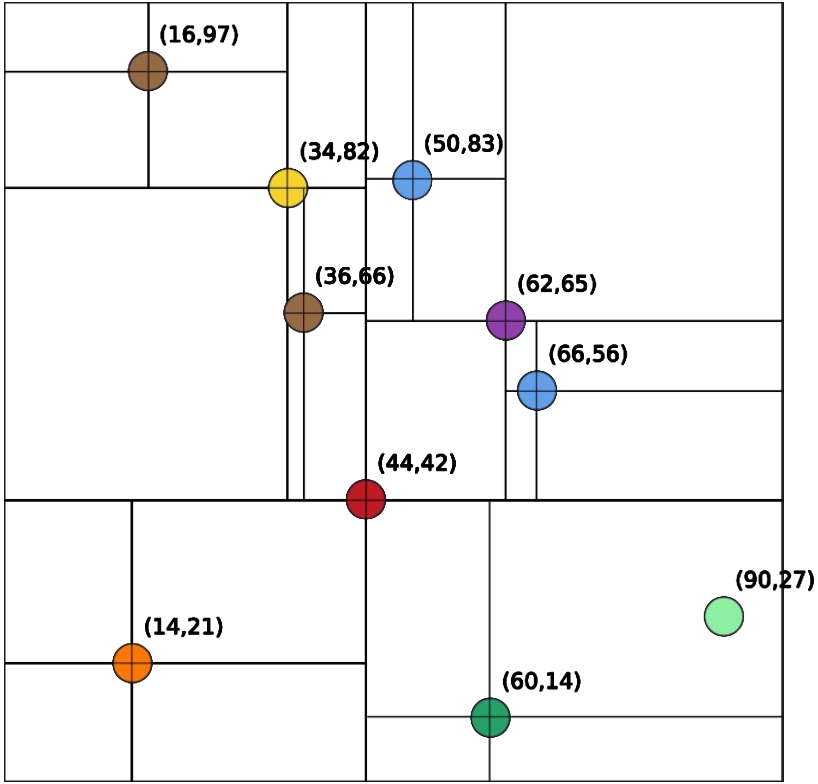
`apply(x, y)`

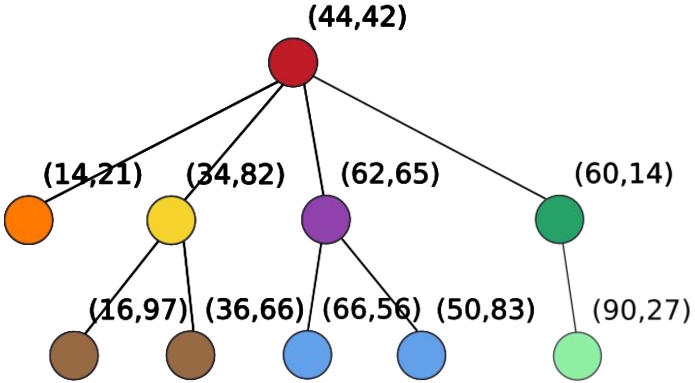
- Find position corresponding to (x, y): ***O*(*d*)**
- Return the node if it exists: ***O*(1)**

`range(xlow, xhigh, ylow, yhigh)`

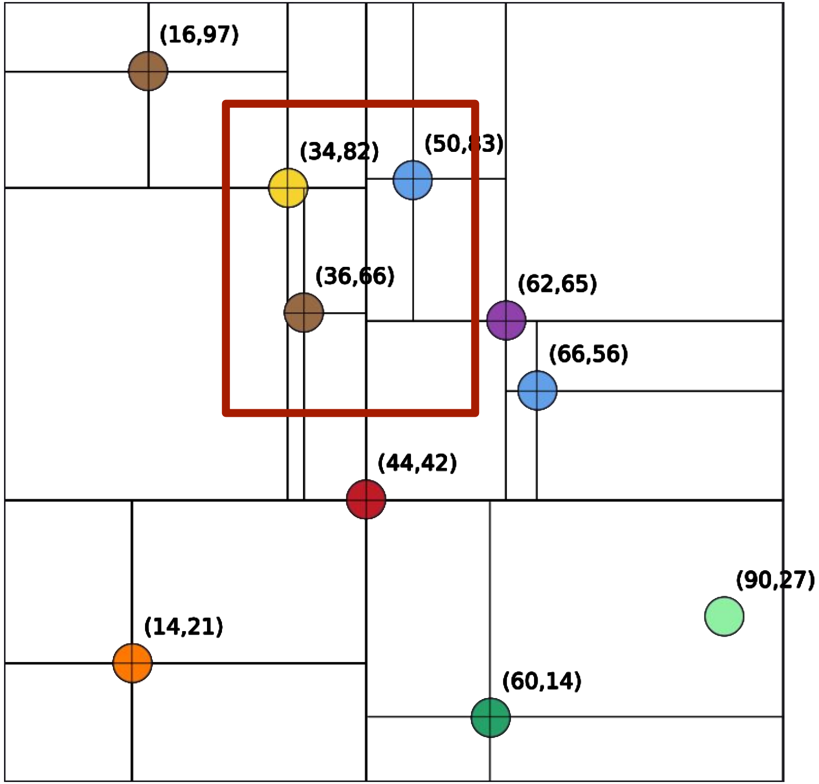
- …?

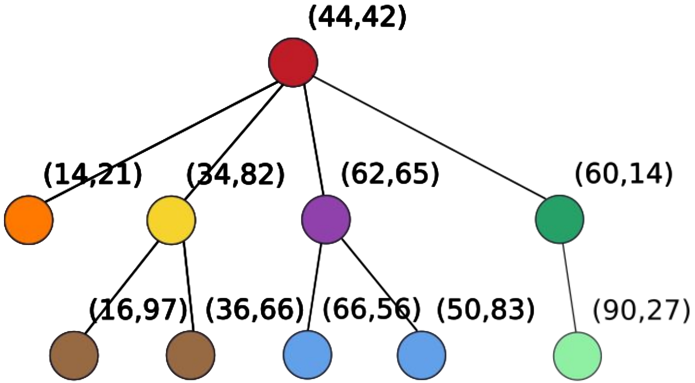
Quad Trees - Range



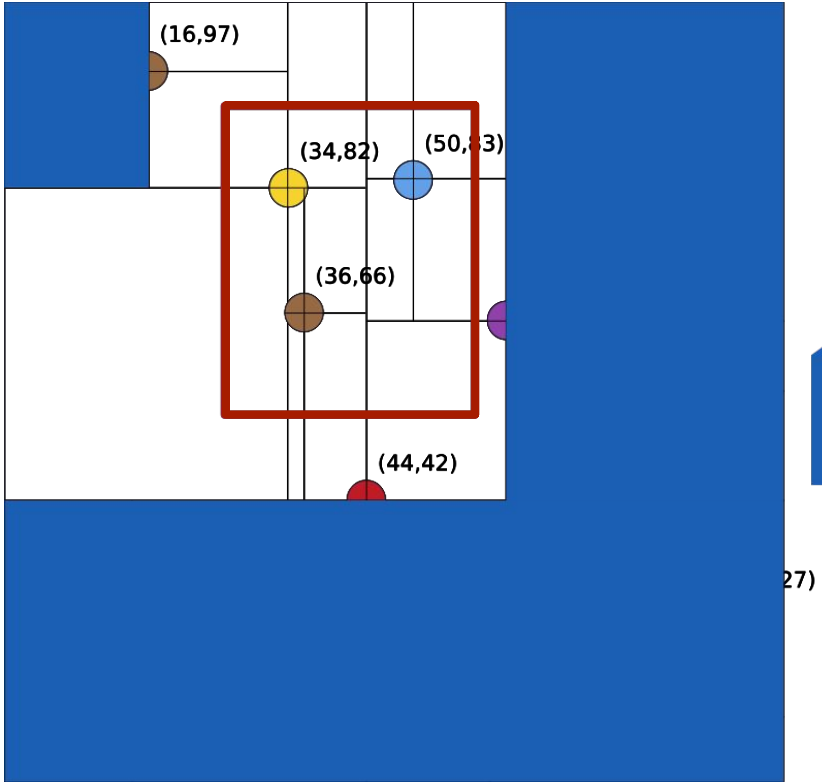


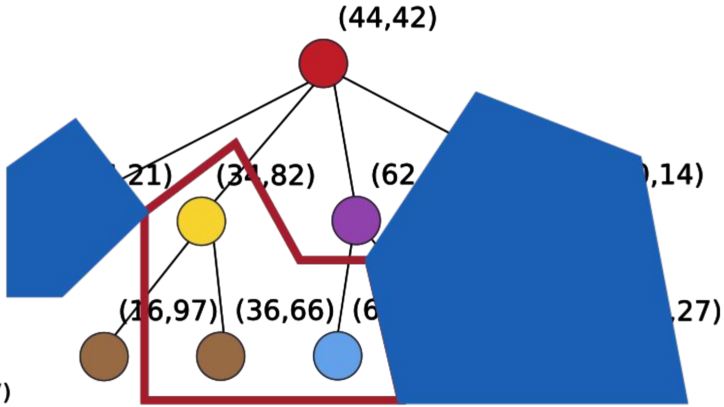
Quad Trees - Range





Quad Trees - Range





Quad Trees - Find Node (With Range)

```
def findNode(x: Int, y: Int): Node[T] = {
  var current = root
  var range = Rectangle(-∞, -∞, ∞, ∞)

  while(current.isDefined && (current.x != x || current.y != y) ){
    if(current.x < x) {
      if(current.y < y){ current = current.llChild;
                         current.range = range.crop(Rectangle(-∞, -∞, x, y)) }
      else             { current = current.lhChild;
                         current.range = range.crop(Rectangle(-∞, y, x, ∞)) }
    } else {
      if(current.y < y){ current = current.hlChild;
                         current.range = range.crop(Rectangle(x, -∞, ∞, y)) }
      else             { current = current.hhChild;
                         current.range = range.crop(Rectangle(x, y, ∞, ∞)) }
    }
  }
  return current
}
```

Quad Trees - Range

```
def range( target: Rectangle ): Seq[Node[T]] = {
  val ret = Buffer[Node[T]]()

  def visit(current: Node[T]) = {
    if( target.intersect(current.range).isEmpty ) { return }
    if( target.contains(current.x, current.y) ){ ret.append(current) }
    if( ll.isDefined ) { visit(llChild) }
    if( lh.isDefined ) { visit(lhChild) }
    if( hl.isDefined ) { visit(hlChild) }
    if( hh.isDefined ) { visit(hhChild) }
  }
  visit(root)
}
```

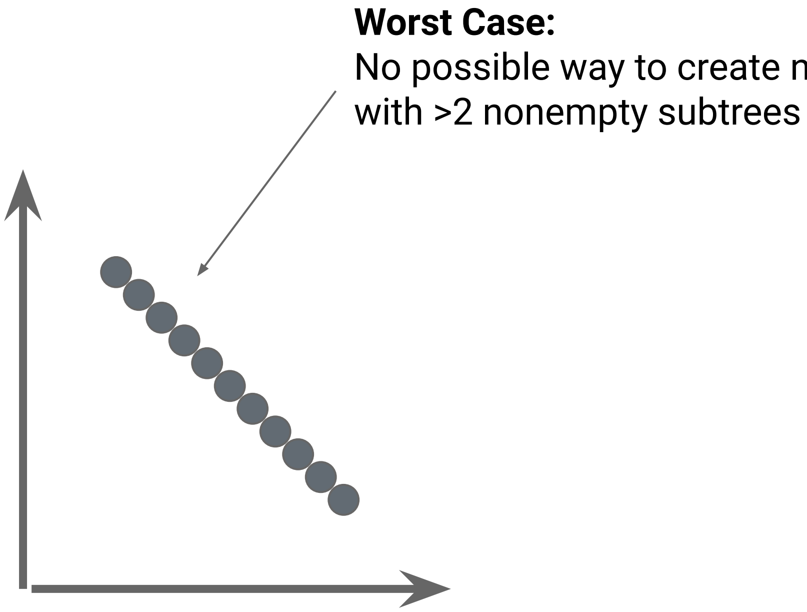
Quad Trees - Challenges

Creating a balanced Quad Tree is hard

- Impossible to always split collection elements evenly across all four subtrees
  (though depth = O(log(n)) still possible)

Keeping the quad tree balanced after updates is significantly harder

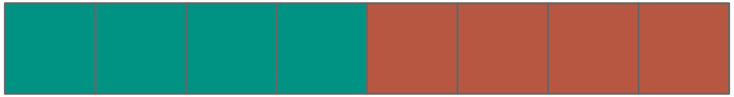
- No “simple” analog for rotate left/right.



Quad Trees - Challenges

Problem: Every node has 4 children!

Revisiting Lexical Order

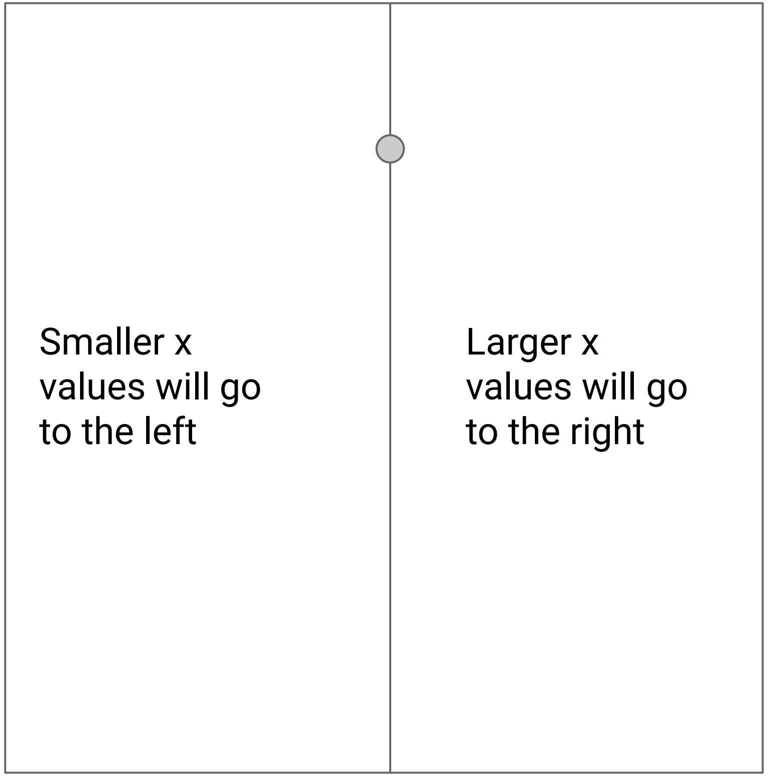


Problem: Searches on lexical order partition all of one dimension first

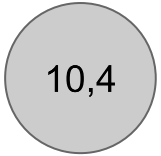
Revisiting Lexical Order

Idea: Alternate dimensions

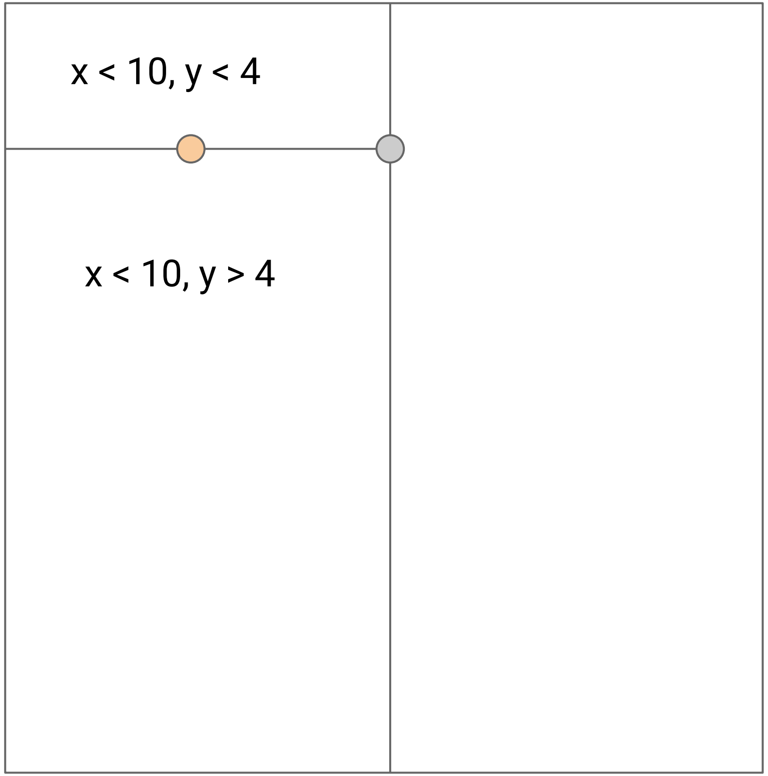
k-D Tree Example



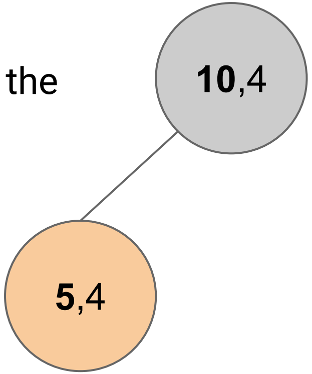
Nodes at level 1 will partition on the first dimension, x



k-D Tree Example - insert(5,4)

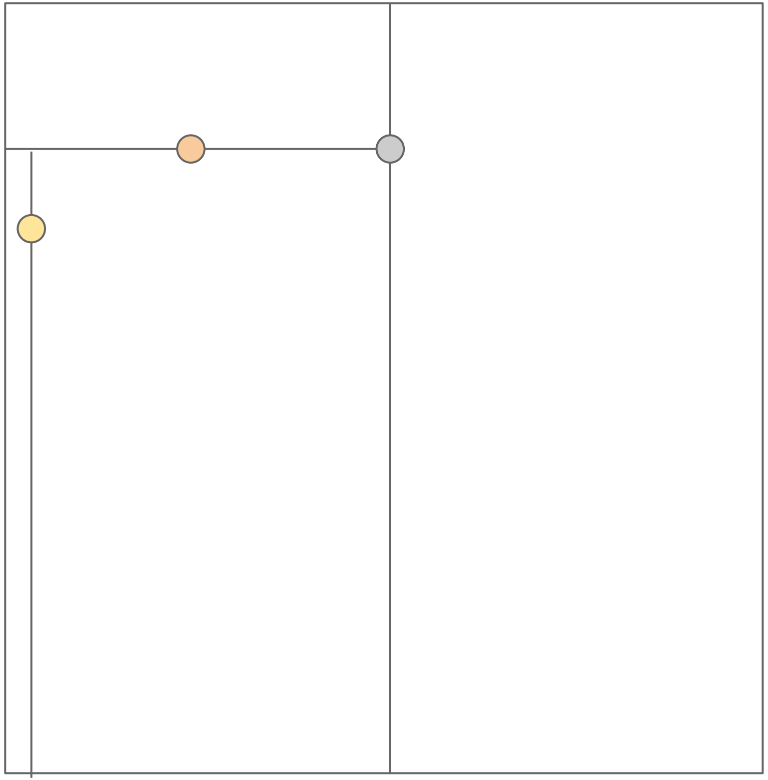


5 < 10, so insert into the left subtree



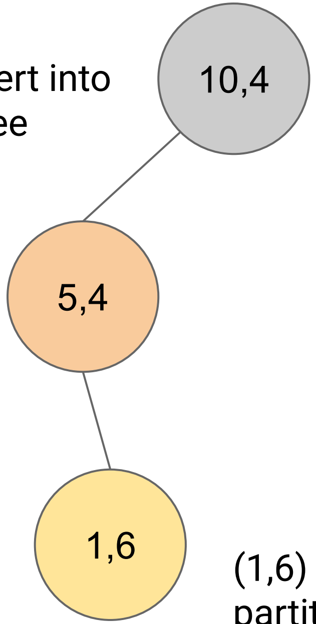
(5,4) is at level 2 so it will partition it's children on the second dimension, y

k-D Tree Example - insert(1,6)



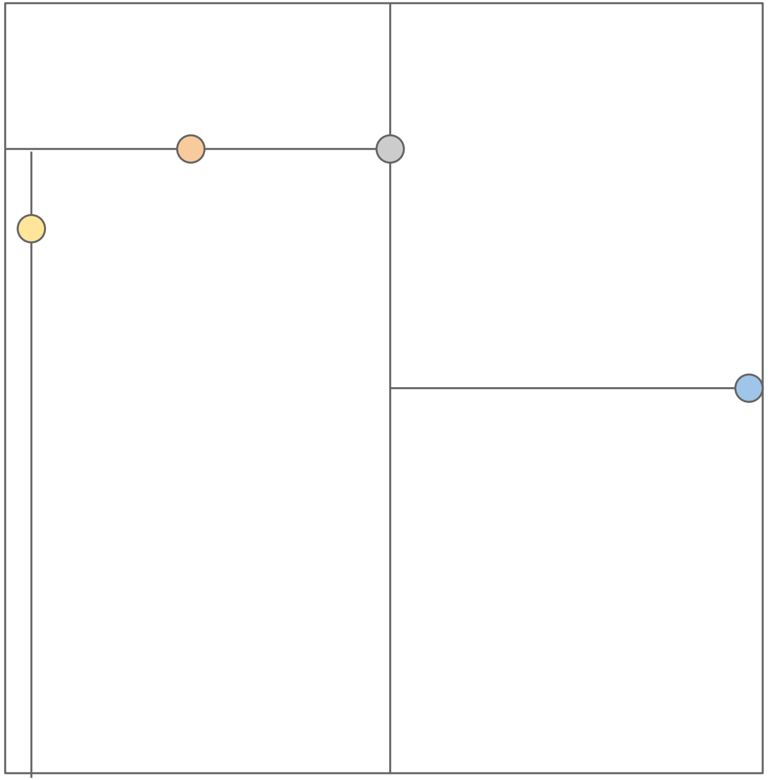
1 < 10, so insert into the left subtree

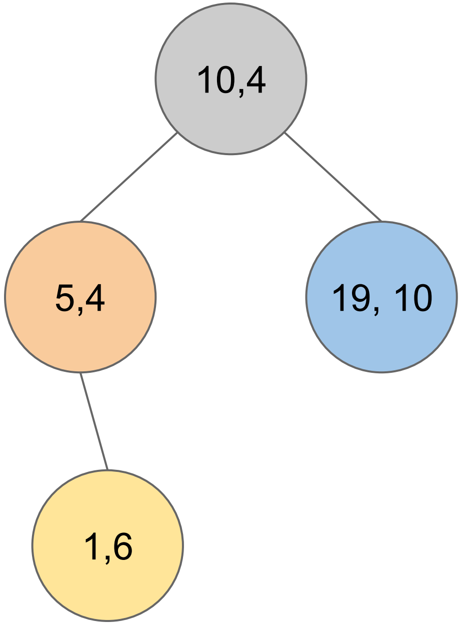
6 > 4 so insert into the right subtree



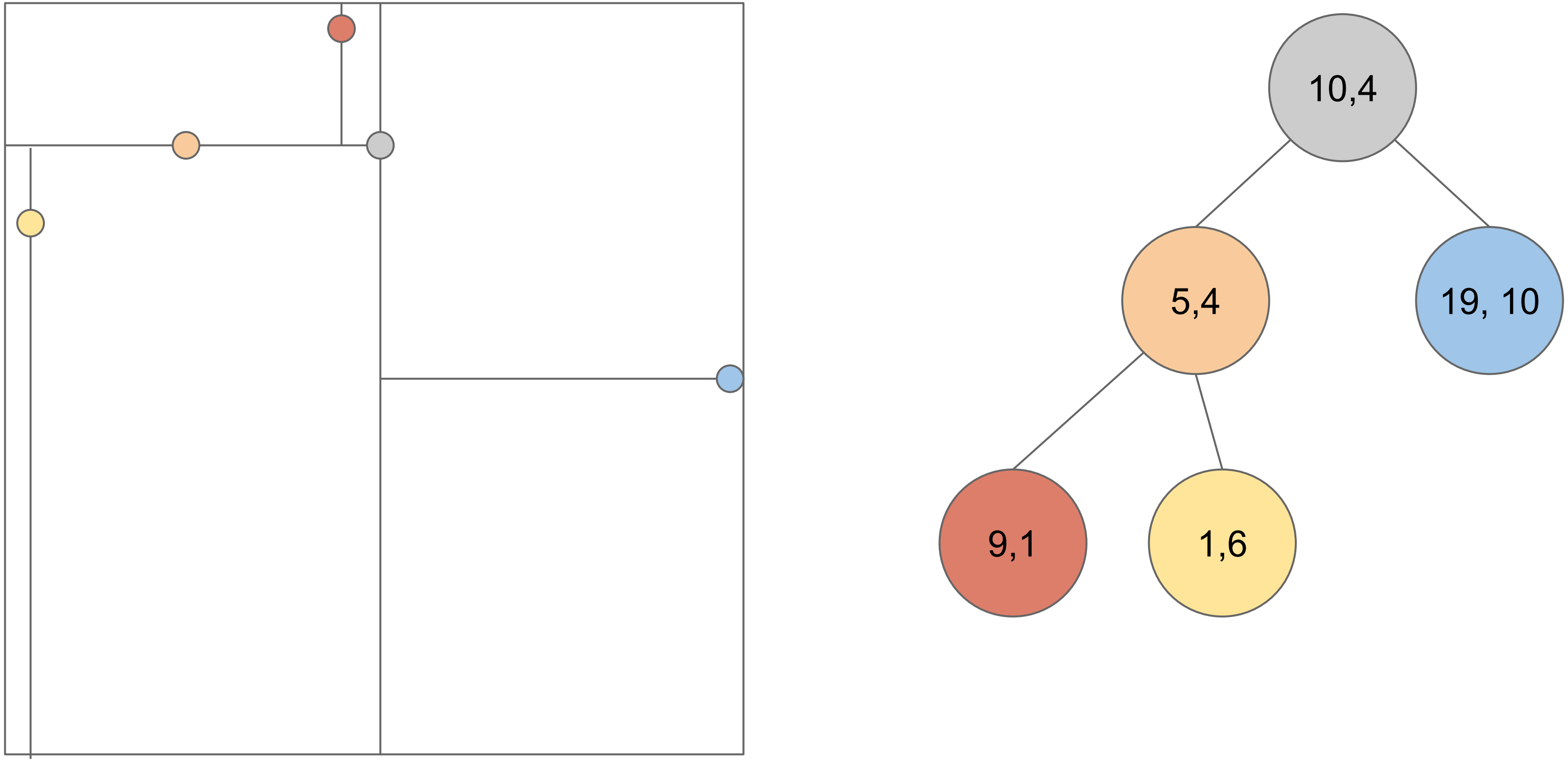
(1,6) is at level 3 so it will partition it's children on the first dimension, x

k-D Tree Example - insert(19,10)

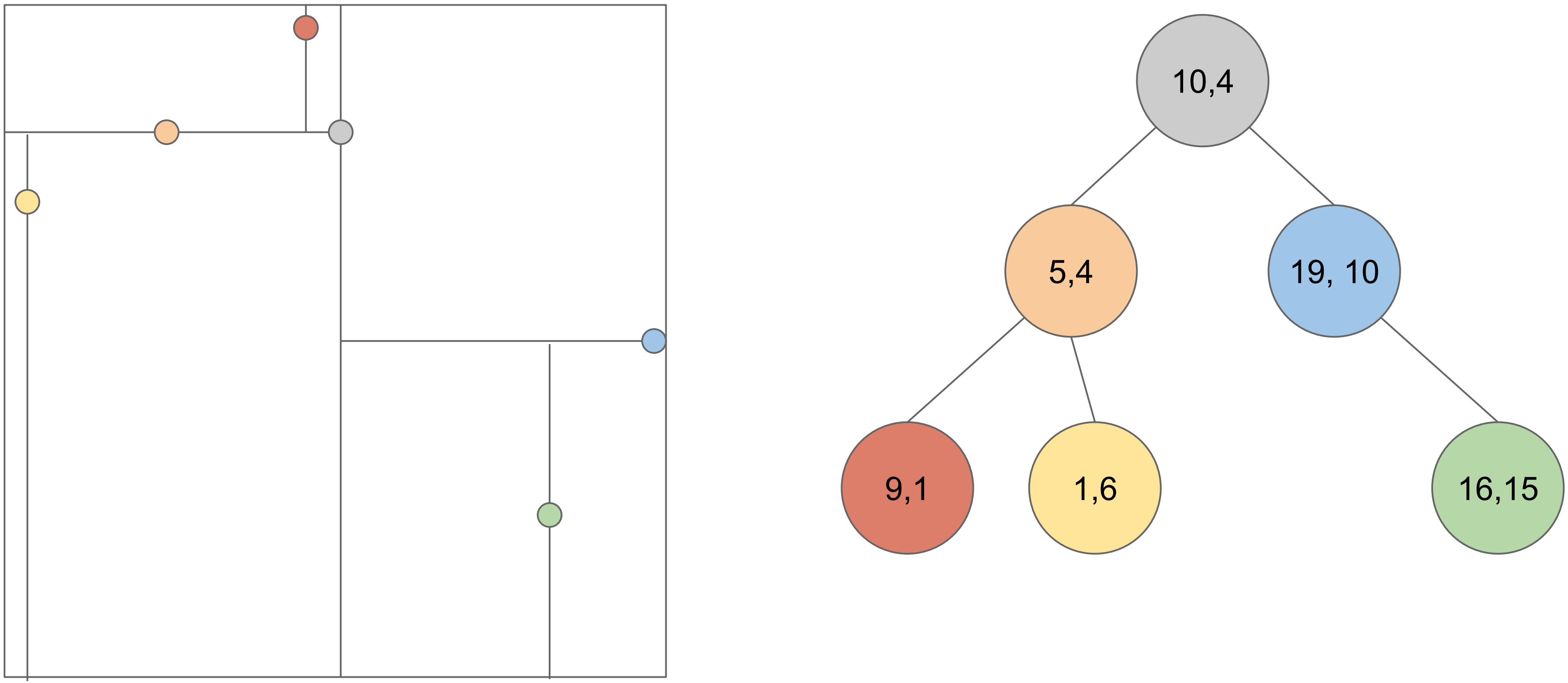




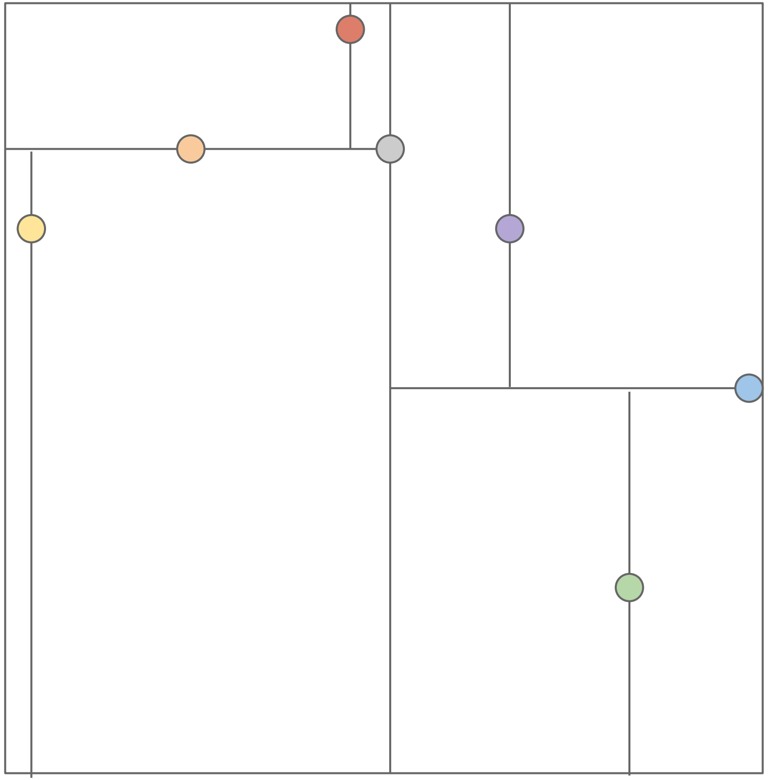
k-D Tree Example - insert(9,1)

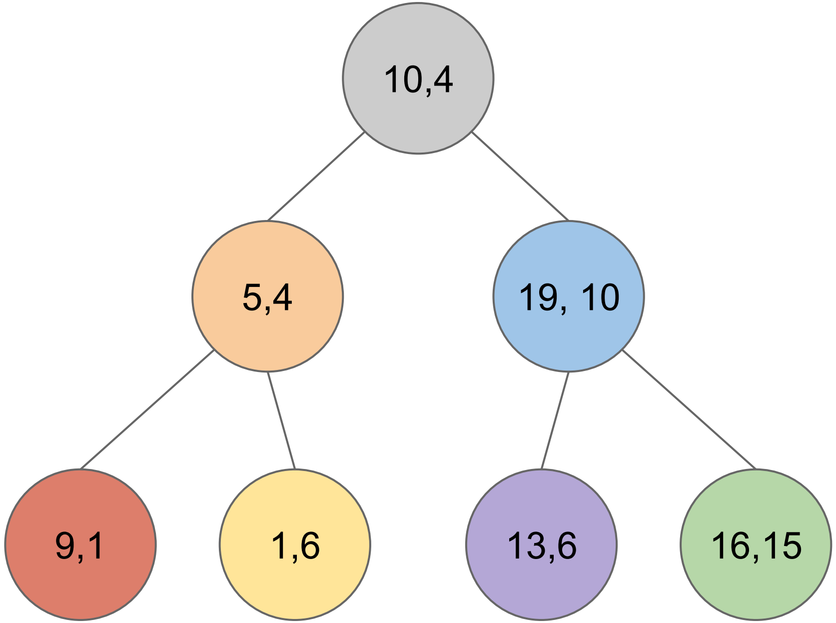


k-D Tree Example - insert(16,15)

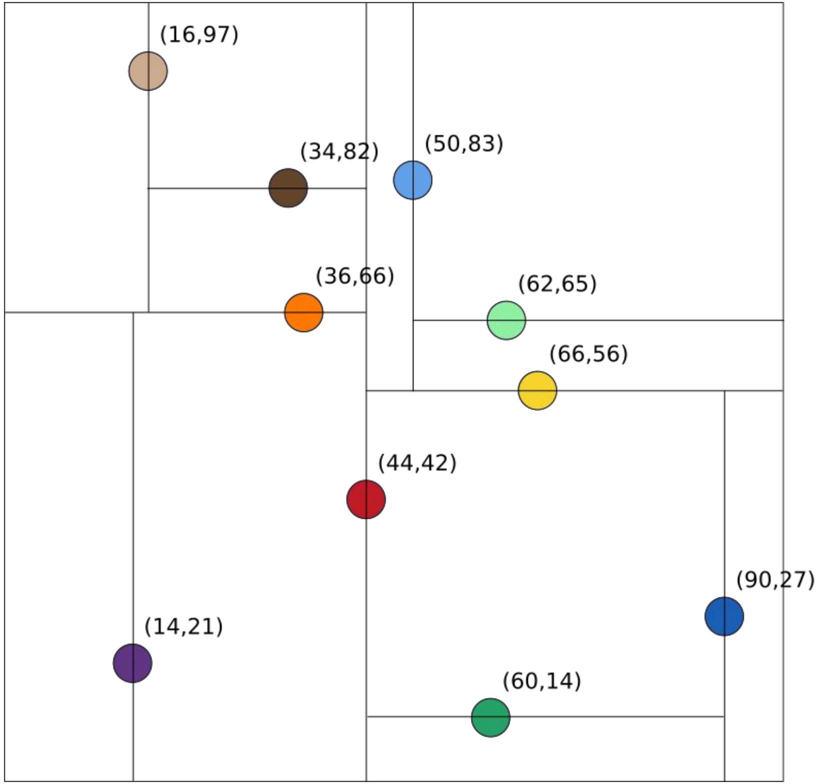


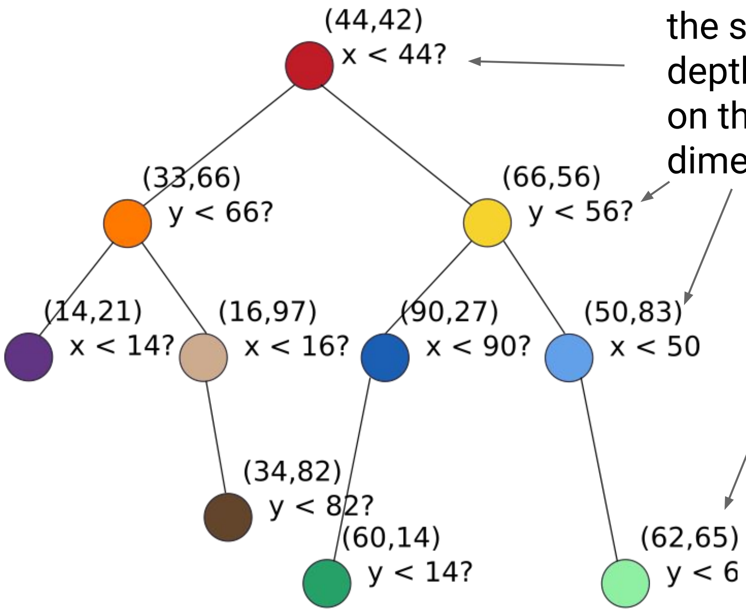
k-D Tree Example - insert(13,6)





k-D Trees - Bigger Example





All nodes at the same depth partition on the same dimension

k-D Trees - Find Node

```
def findNode(x: Int, y: Int): Node[T] = {
  var current = root
  var depth = 0

  while(current.isDefined && (current.x != x || current.y != y) ){
    if(depth % 2 == 1) { if(current.x < x) { current = current.left }
                         else              { current = current.right }
    else                 { if(current.y < y) { current = current.left }
                         else              { current = current.right }
    depth += 1
  }
  return current
}
```

k-D Trees - Find Node

```
def findNode(x: Int, y: Int): Node[T] = {
  var current = root
  var depth = 0

  while(current.isDefined && (current.x != x || current.y != y) ){
    if(depth % 2 == 1) { if(current.x < x) { current = current.left }
                         else             { current = current.right }
    else               { if(current.y < y) { current = current.left }
                         else             { current = current.right }
    depth += 1
  }
  return current
}
```

What's the complexity?

k-D Trees - Find Node

```
def findNode(x: Int, y: Int): Node[T] = {
  var current = root
  var depth = 0

  while(current.isDefined && (current.x != x || current.y != y) ){
    if(depth % 2 == 1) { if(current.x < x) { current = current.left }
                         else              { current = current.right }
    else               { if(current.y < y) { current = current.left }
                         else              { current = current.right }
    depth += 1
  }
  return current
}
```

What's the complexity? ***O*(*d*)**

k-D Trees - Other Operations

`insert(x, y, value)`

- Find placeholder spot corresponding to (x, y): ***O*(*d*)**
- Create and inject new node: **O(1)**

`apply(x, y)`

- Find position corresponding to (x, y): ***O*(*d*)**
- Return node if it exists: ***O*(1)**

Nearest Neighbor

What if we want to find the closest point to our target?

Nearest Neighbor

What if we want to find the closest point to our target?

Problem: Can't just do normal find; the target may not be in the tree at all

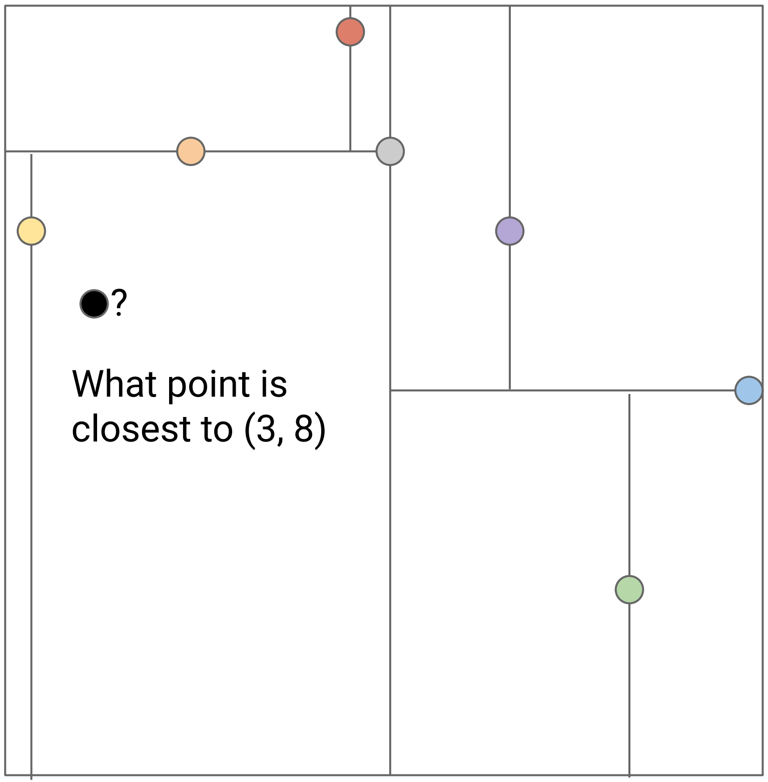
Nearest Neighbor

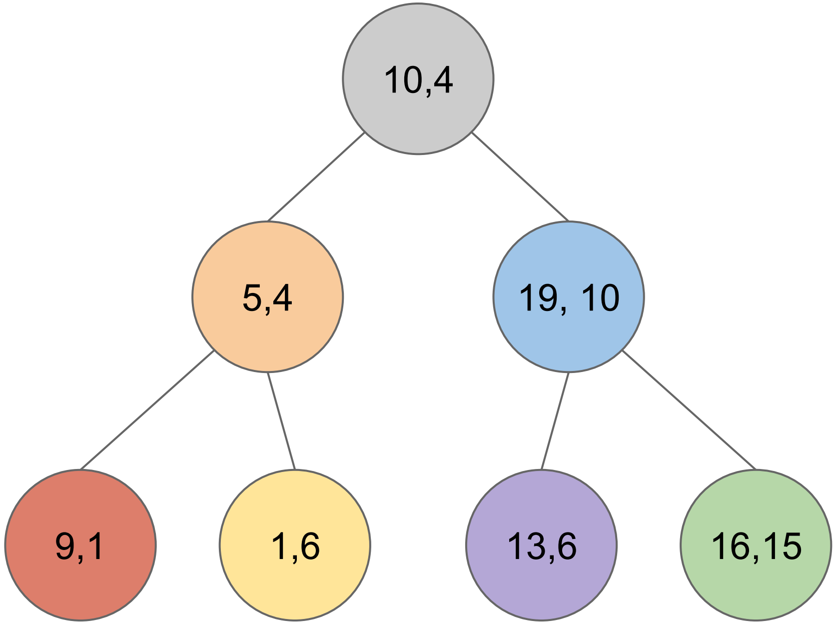
What if we want to find the closest point to our target?

Problem: Can't just do normal find; the target may not be in the tree at all

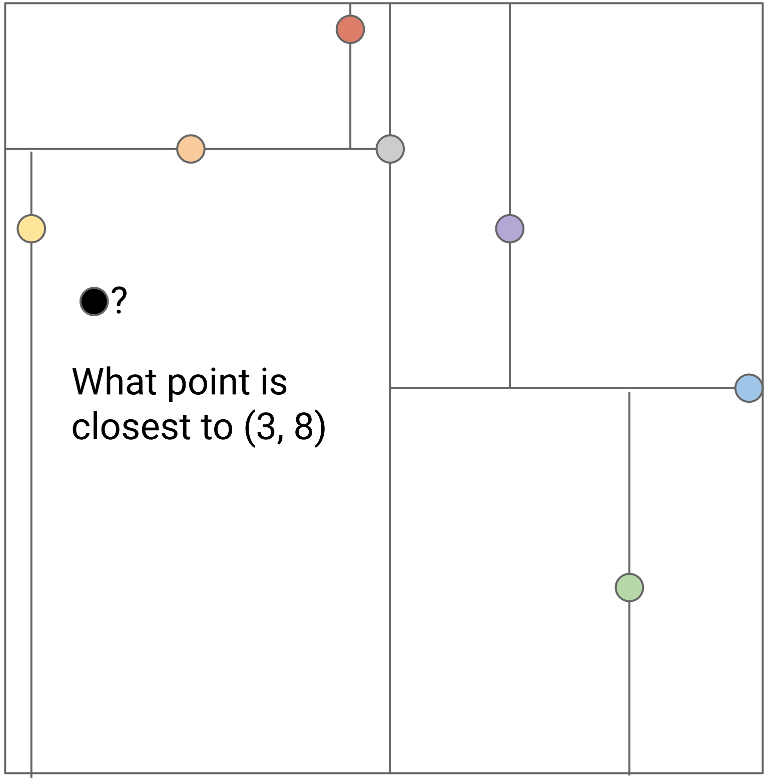
Idea: Search like normal until we hit a leaf, then go back up the tree and see if there's a possibility we missed something.

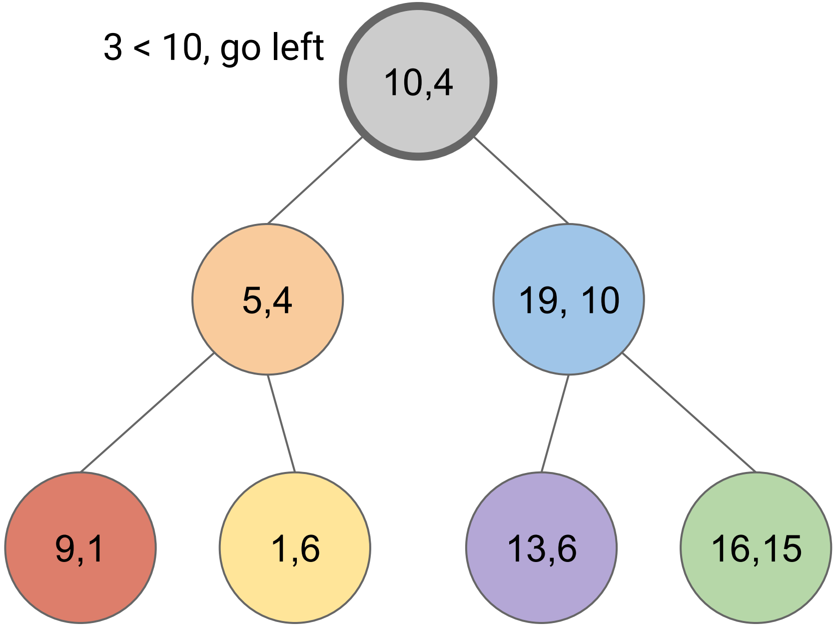
Nearest Neighbor - Example 1



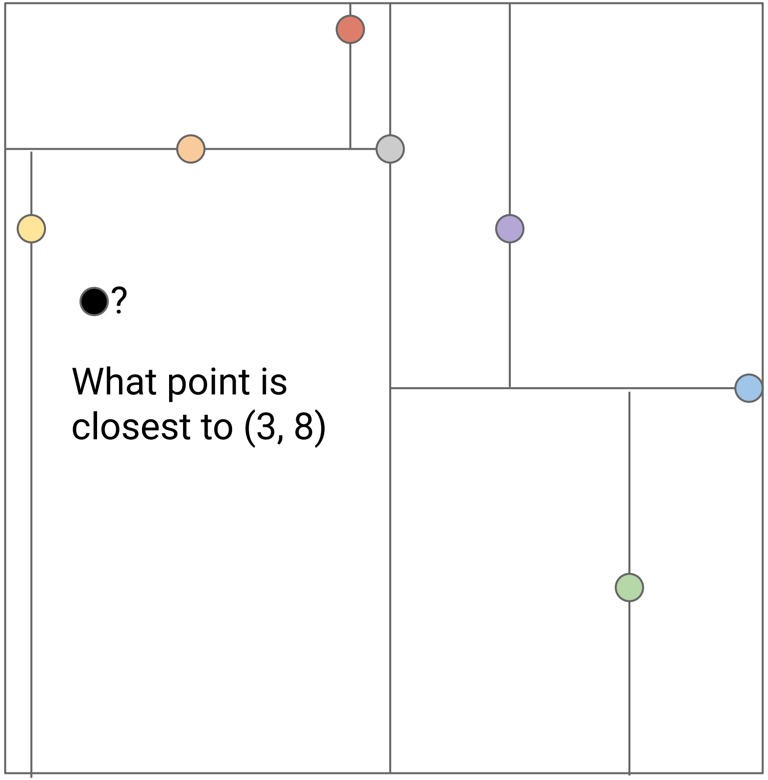


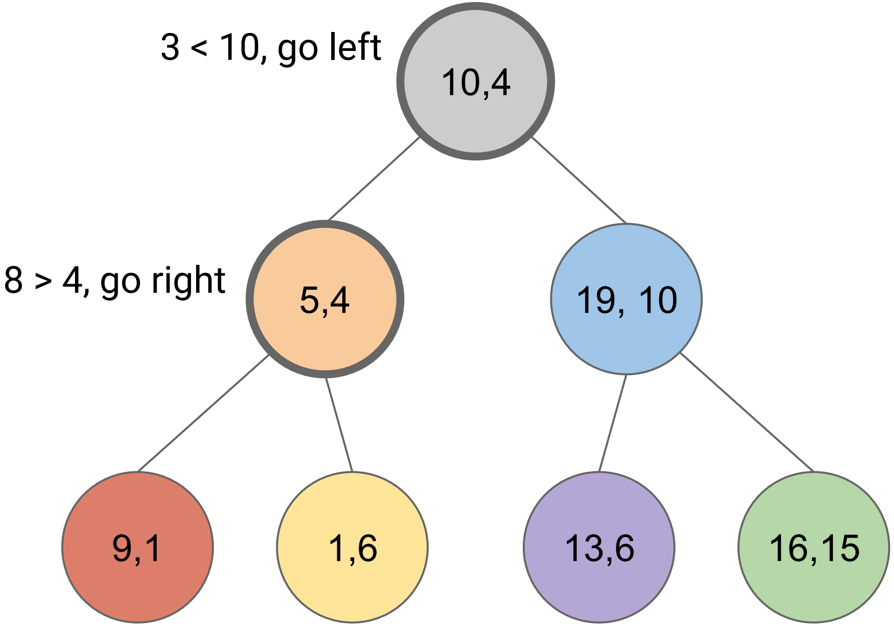
Nearest Neighbor - Example 1



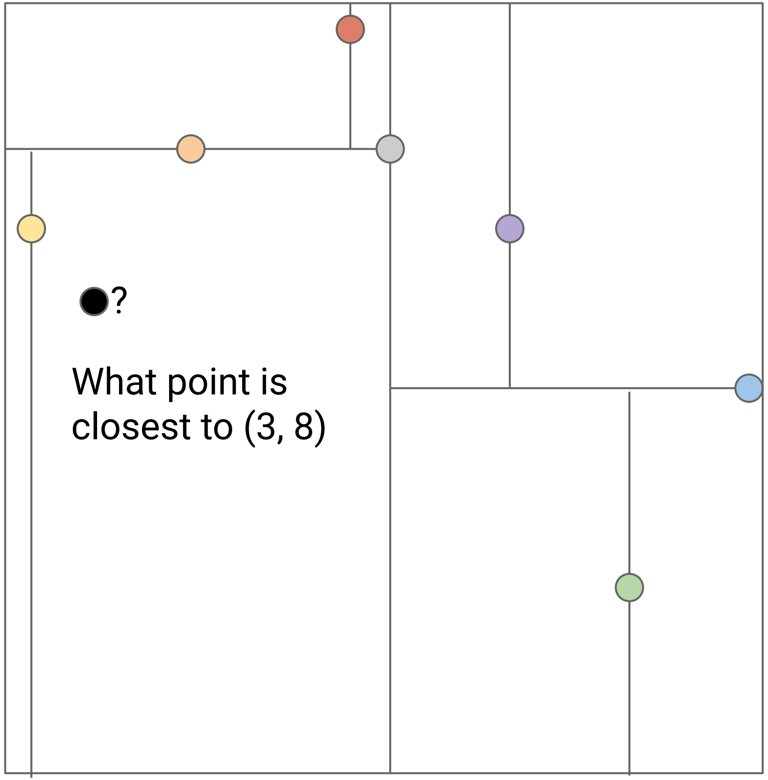


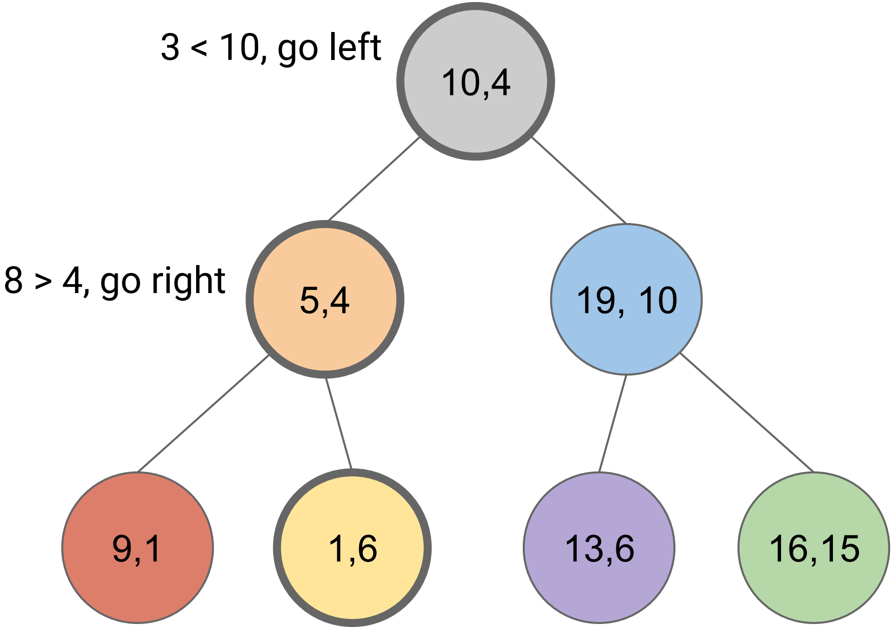
Nearest Neighbor - Example 1





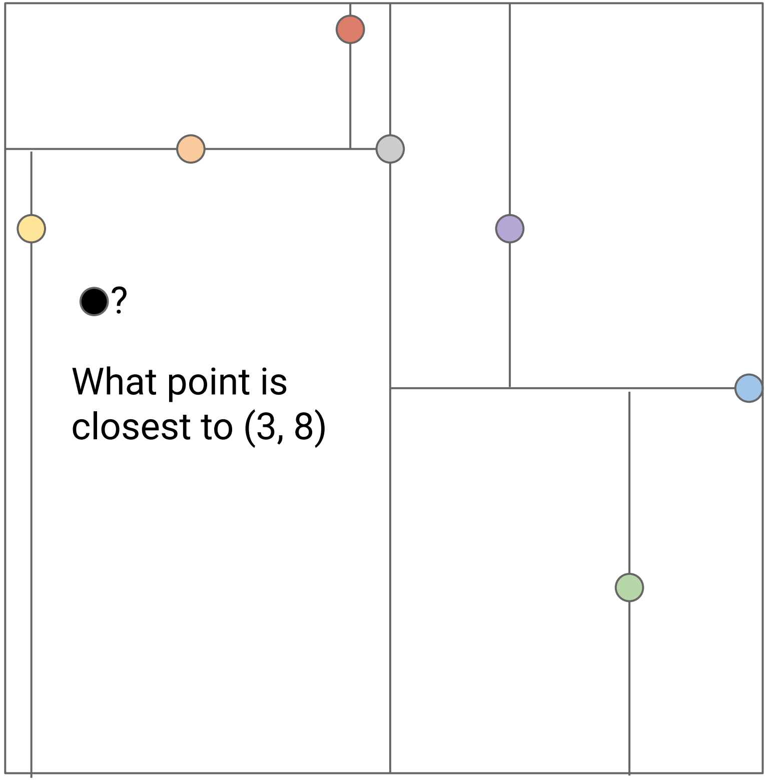
Nearest Neighbor - Example 1



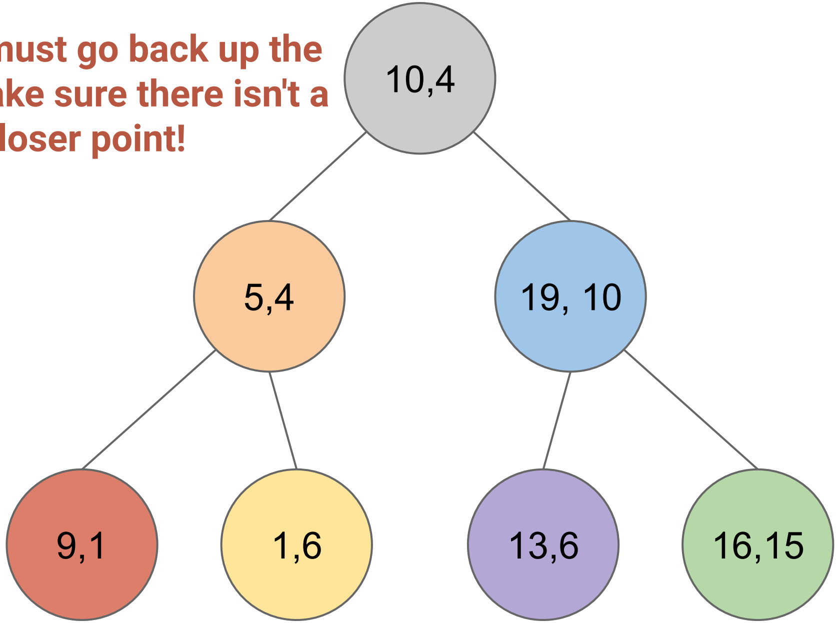


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

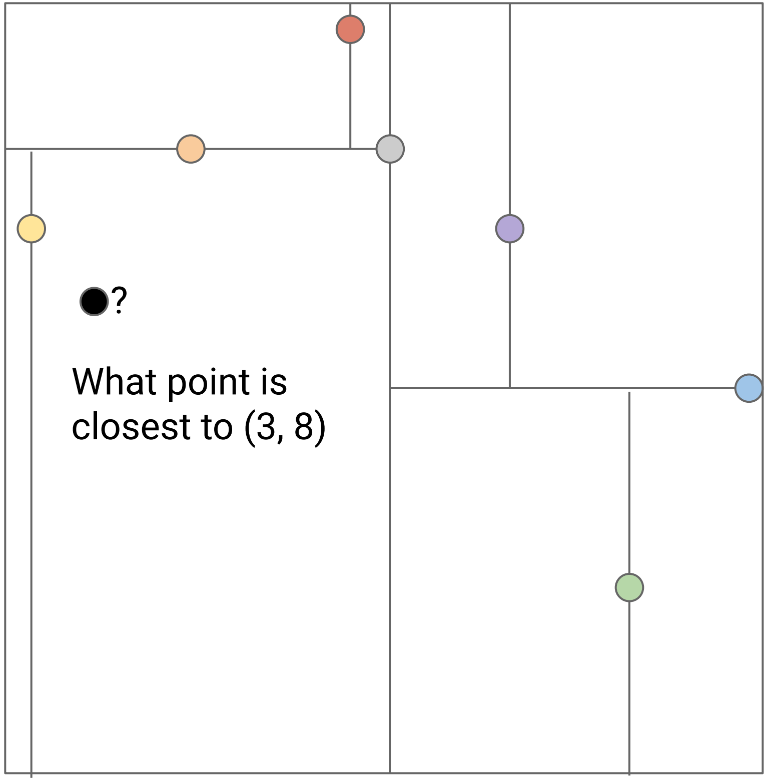


Now we must go back up the tree to make sure there isn't a closer point!

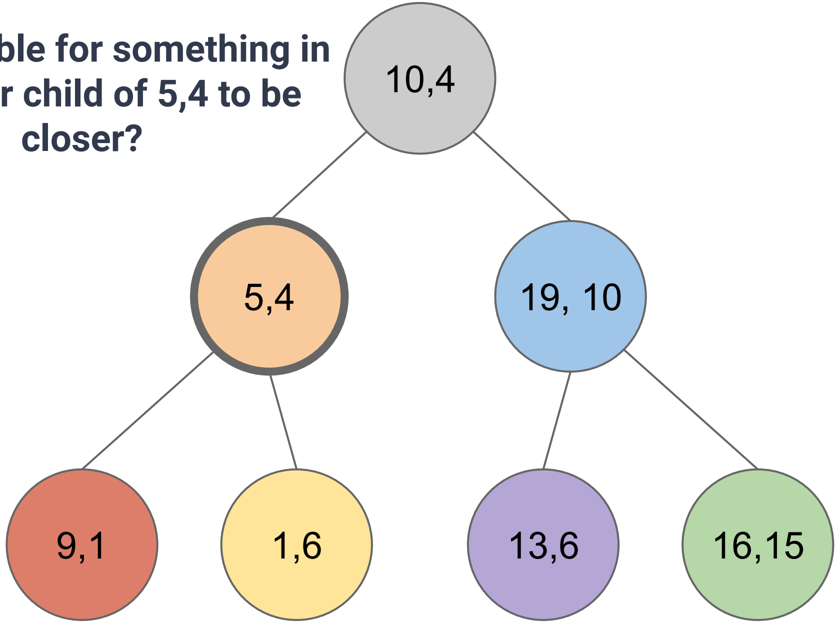


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

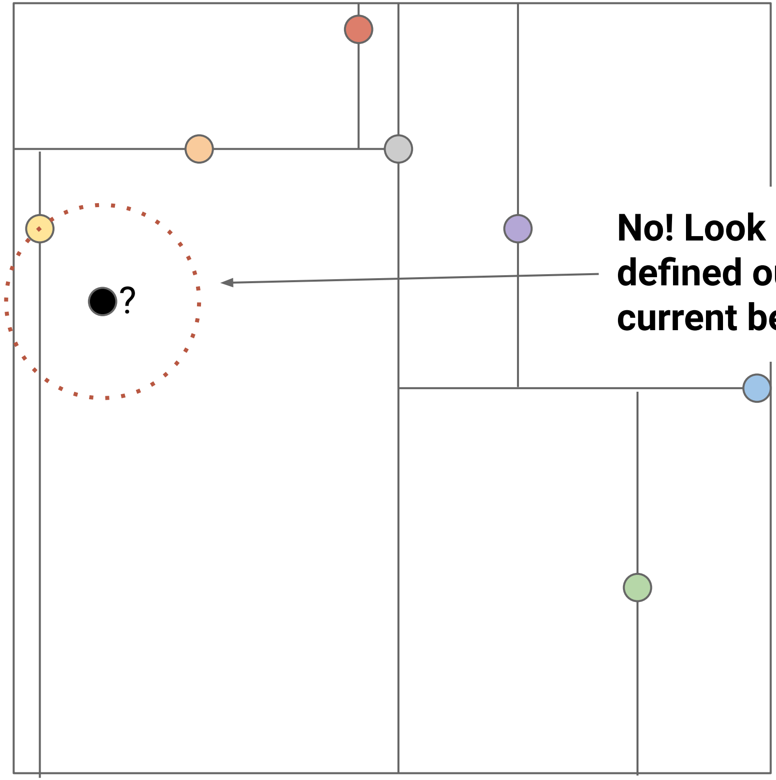


Is it possible for something in the other child of 5,4 to be closer?



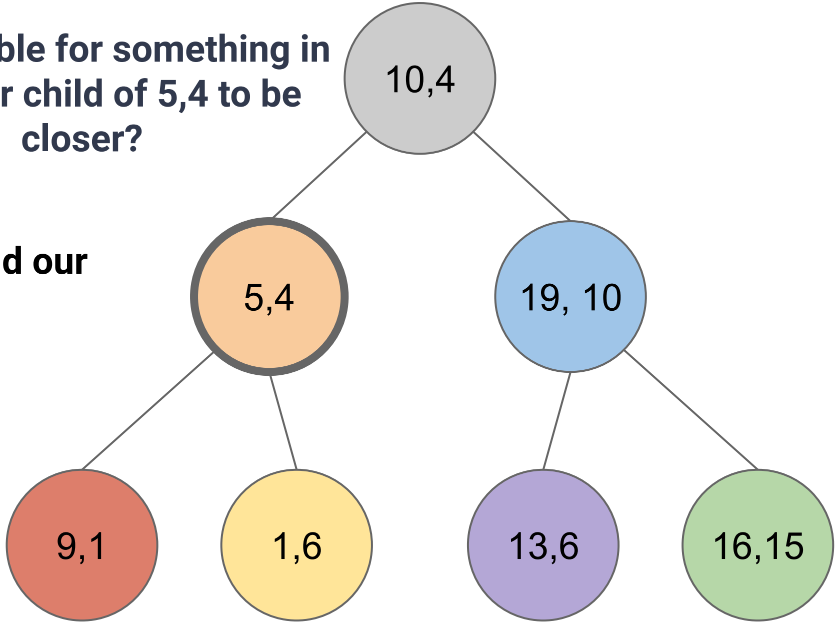
Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1



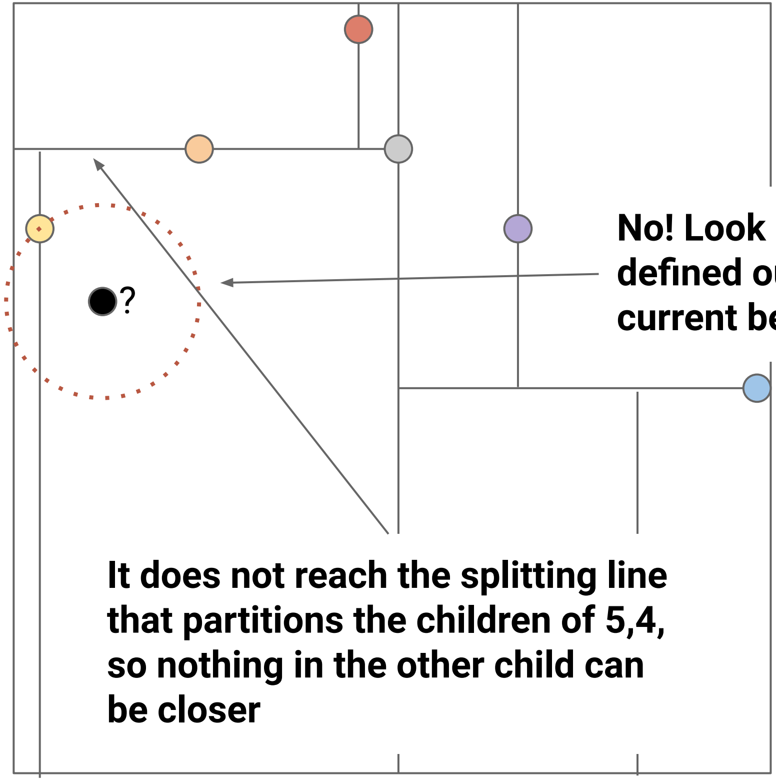
No! Look at the area defined our target and our current best.

Is it possible for something in the other child of 5,4 to be closer?



Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

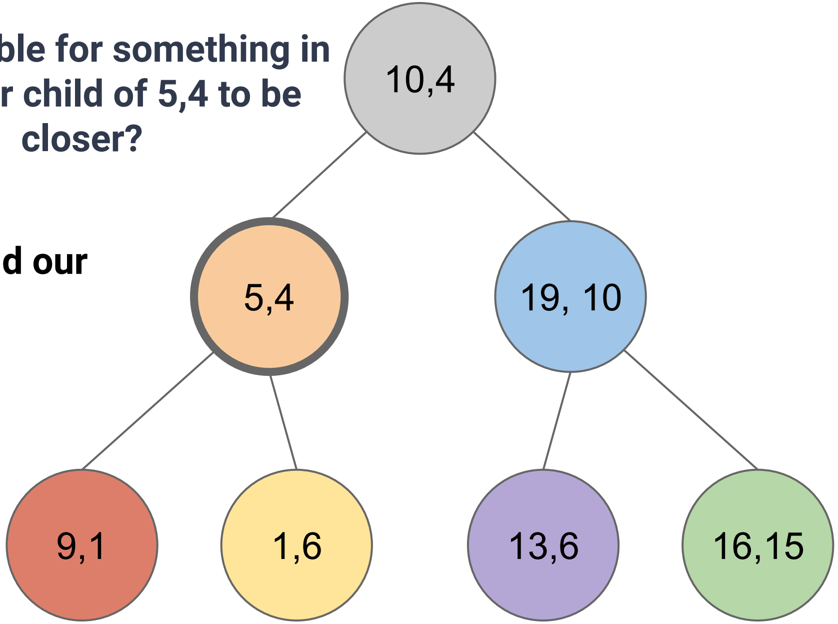
Nearest Neighbor - Example 1



Is it possible for something in the other child of 5,4 to be closer?

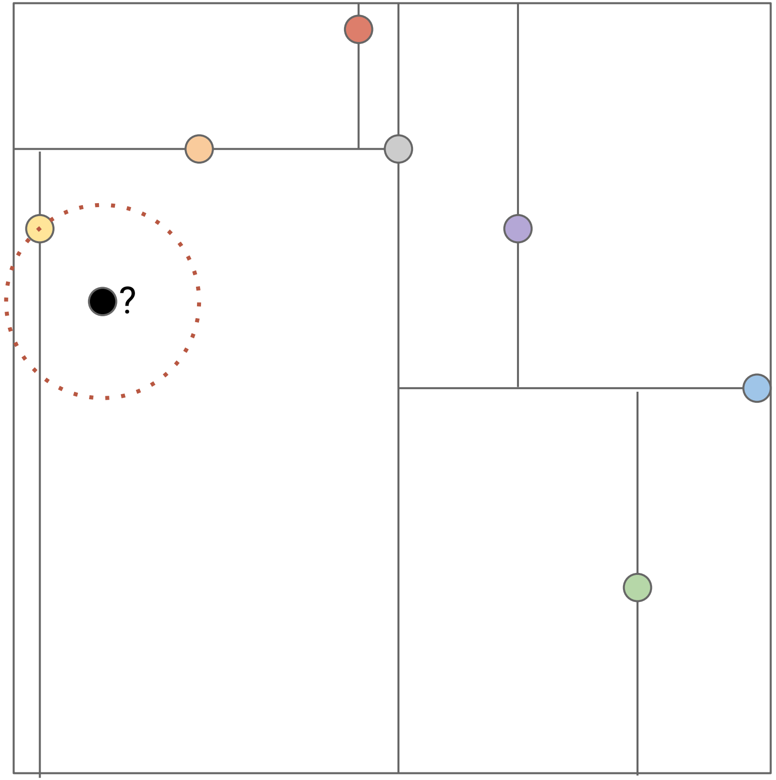
No! Look at the area defined our target and our current best.

It does not reach the splitting line that partitions the children of 5,4, so nothing in the other child can be closer

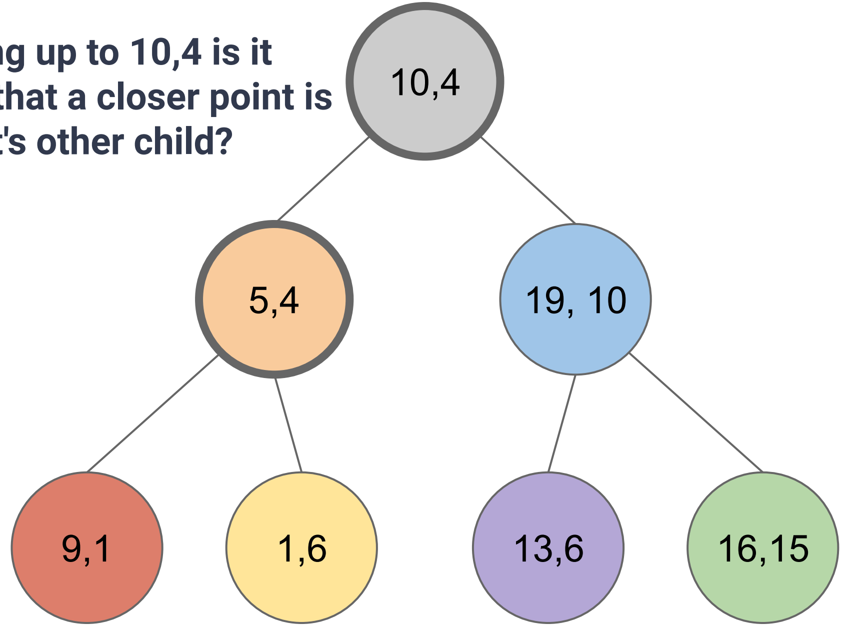


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

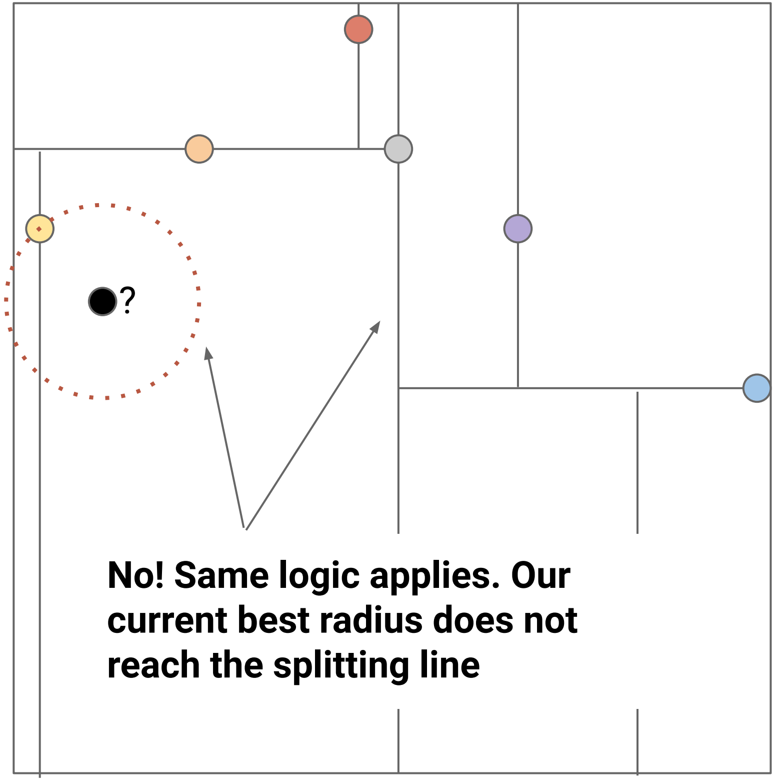


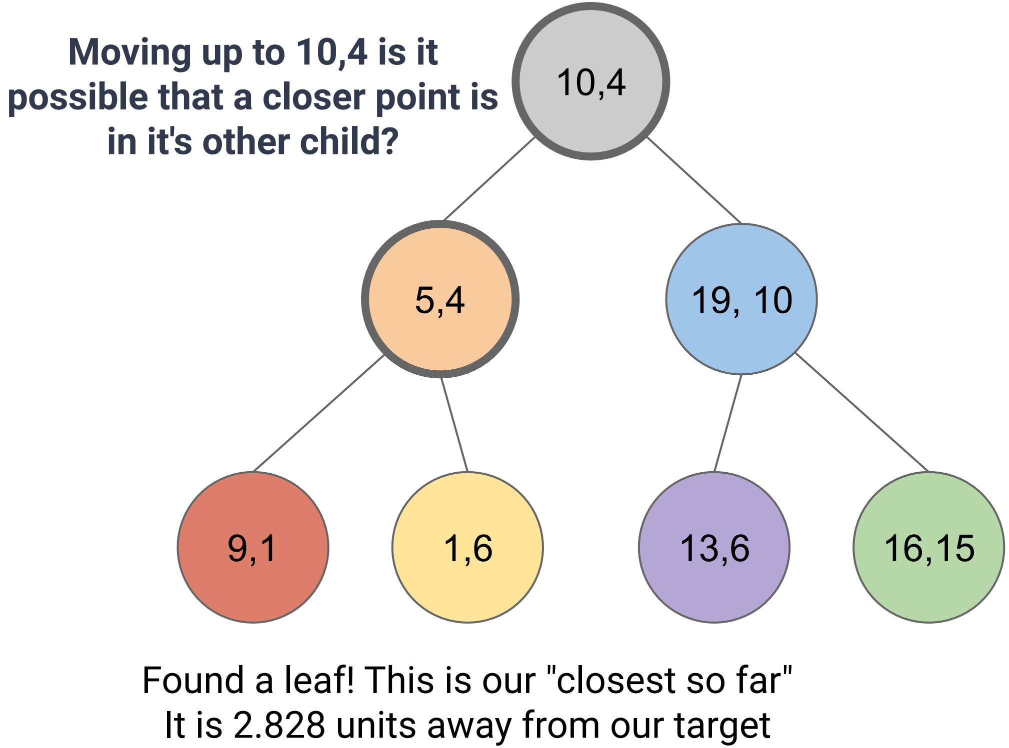
Moving up to 10,4 is it possible that a closer point is in it's other child?



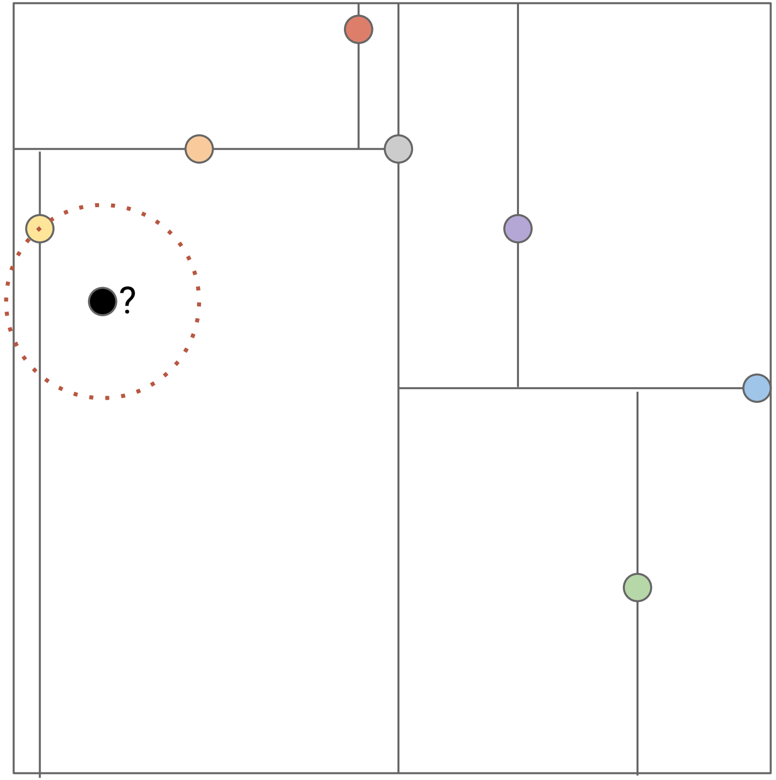
Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

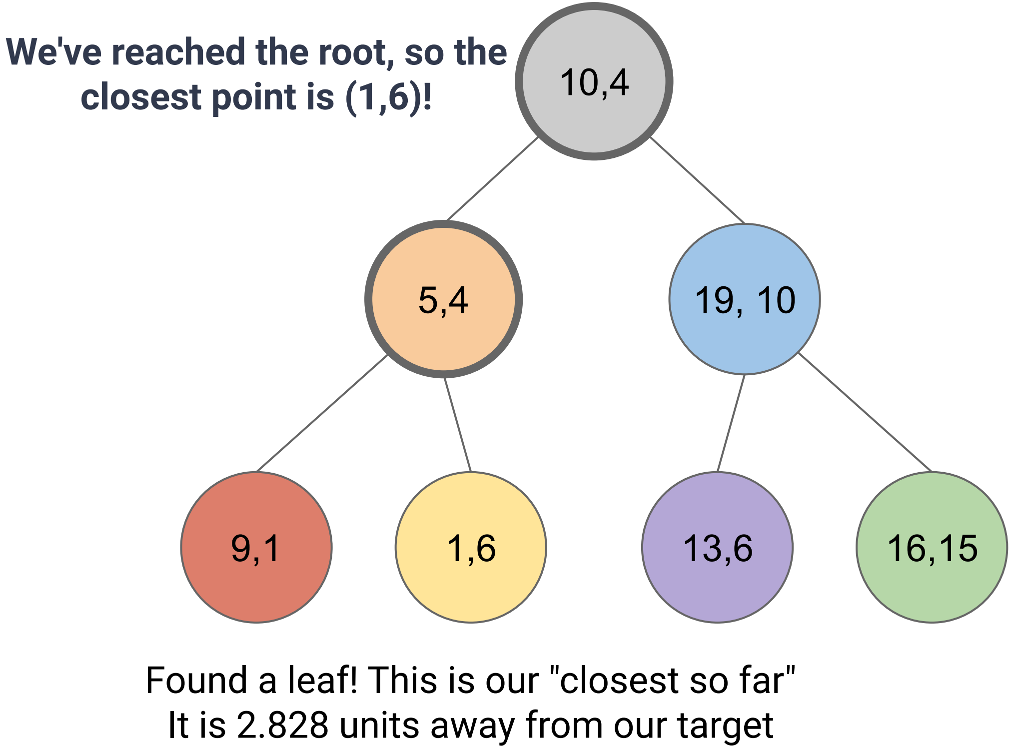
Nearest Neighbor - Example 1



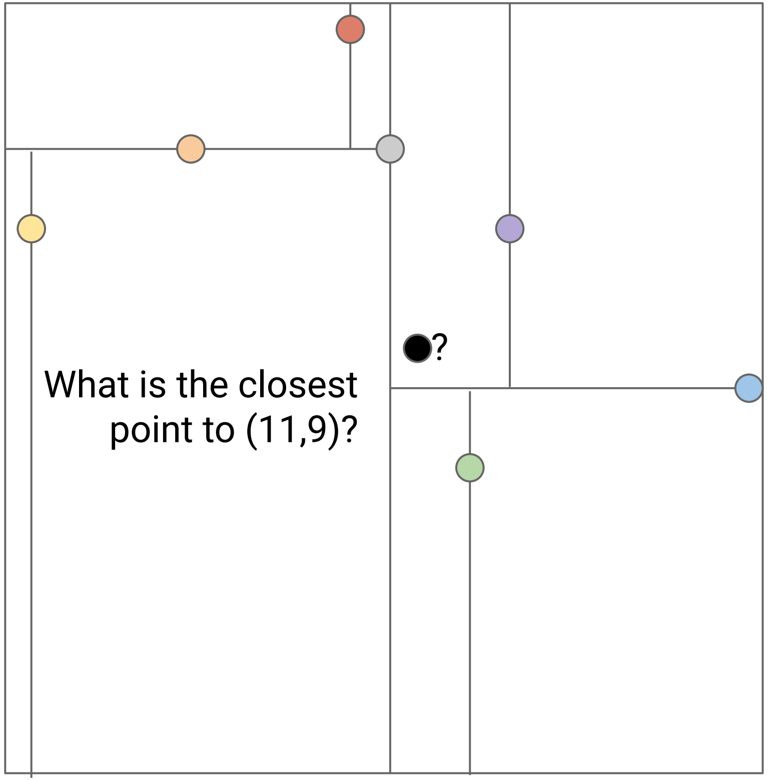


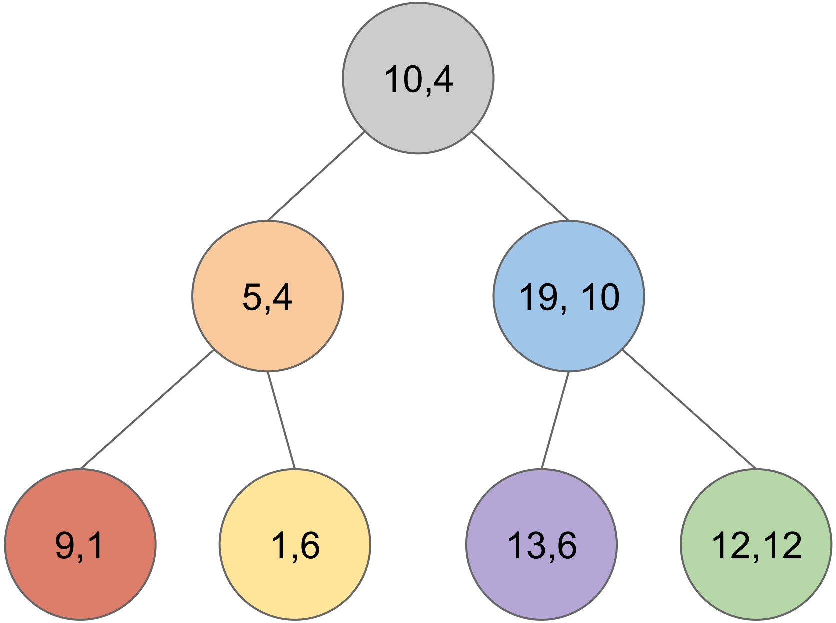
Nearest Neighbor - Example 1



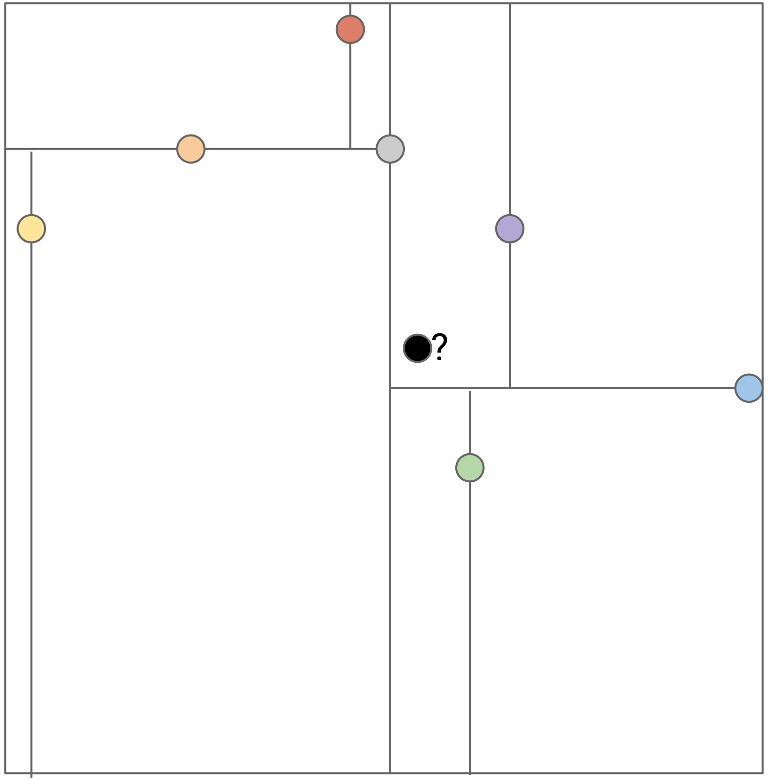


Nearest Neighbor - Example 2

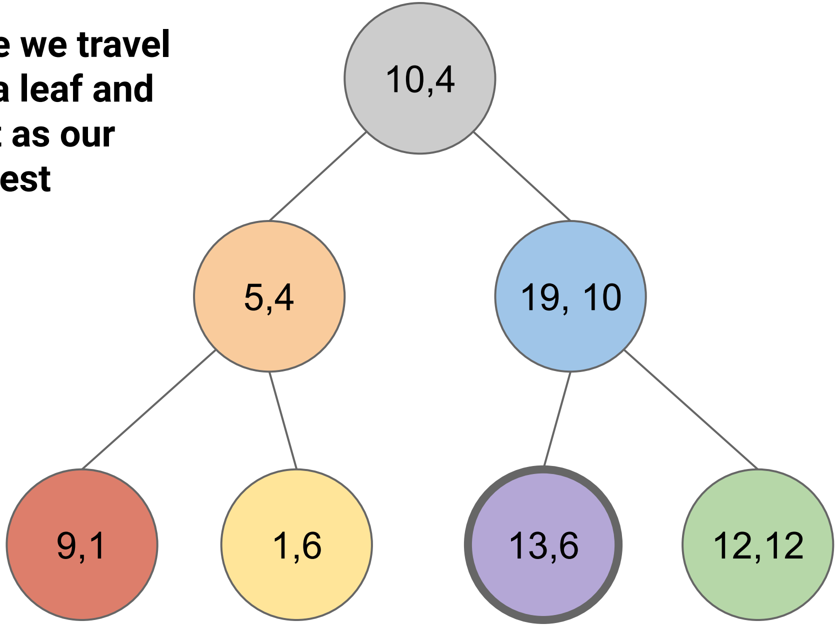




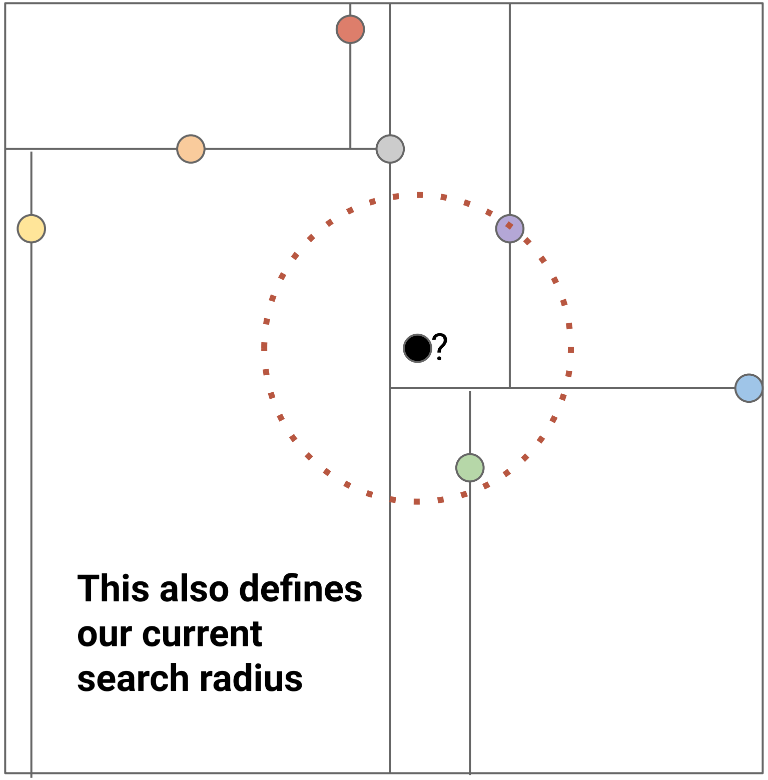
Nearest Neighbor - Example 2



As before we travel down to a leaf and treat that as our current best

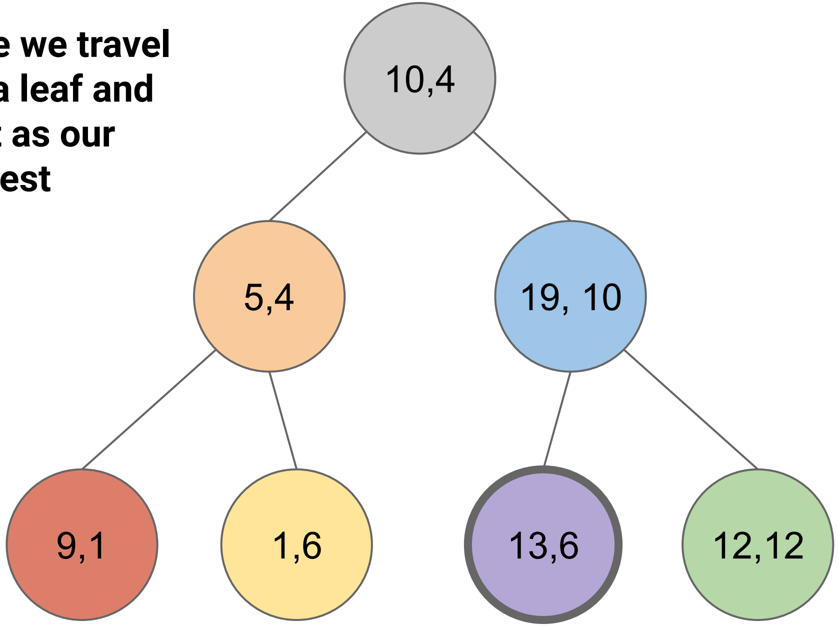


Nearest Neighbor - Example 2

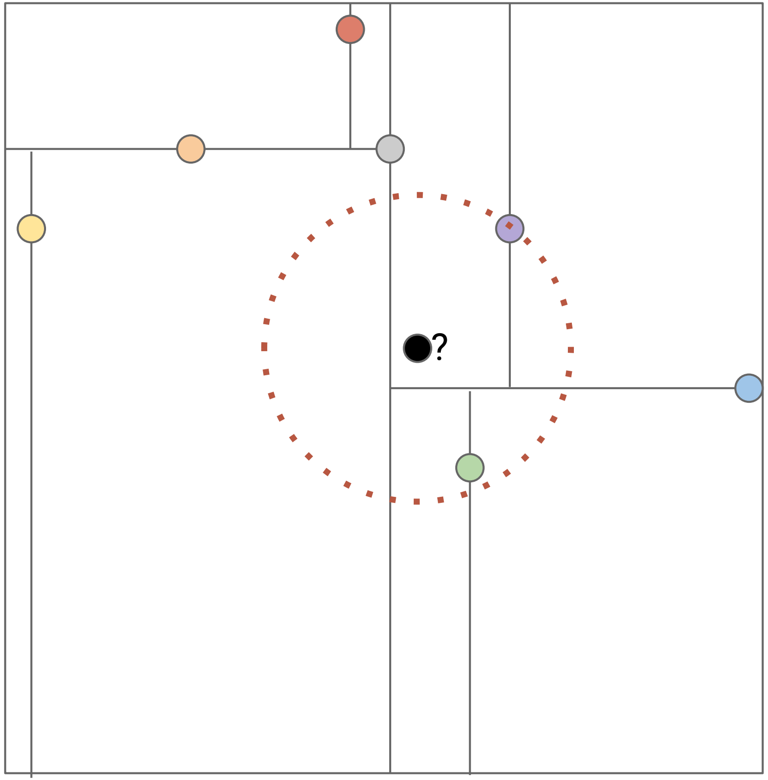


This also defines our current search radius

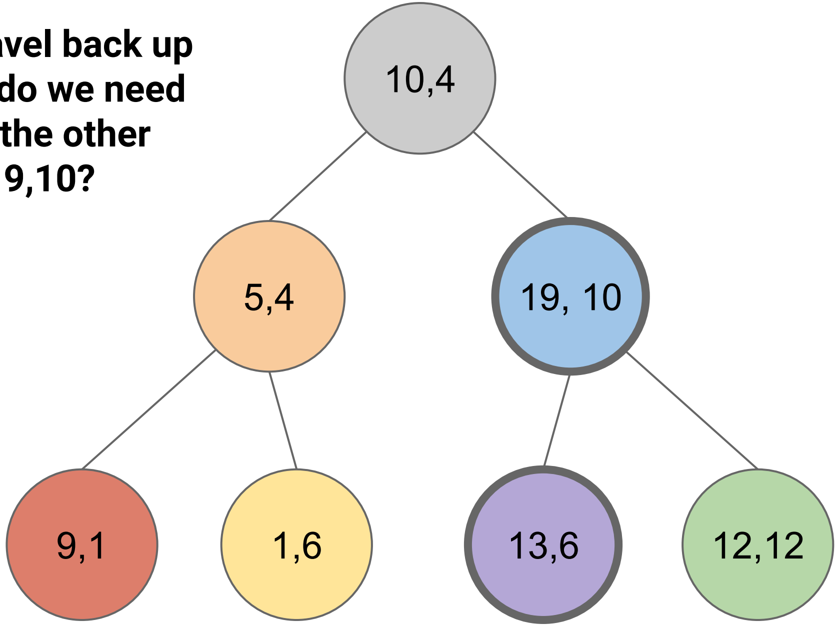
As before we travel down to a leaf and treat that as our current best



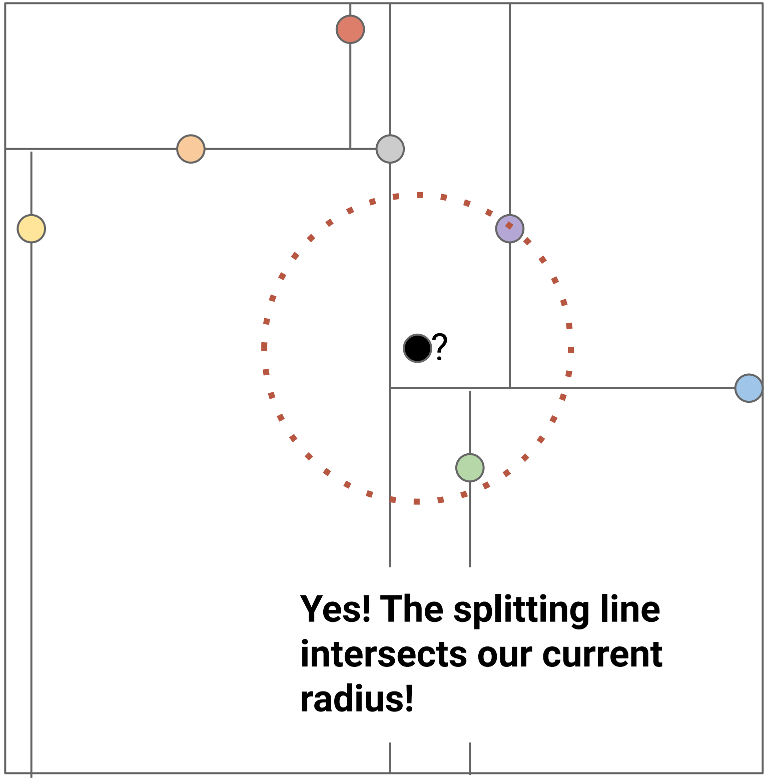
Nearest Neighbor - Example 2



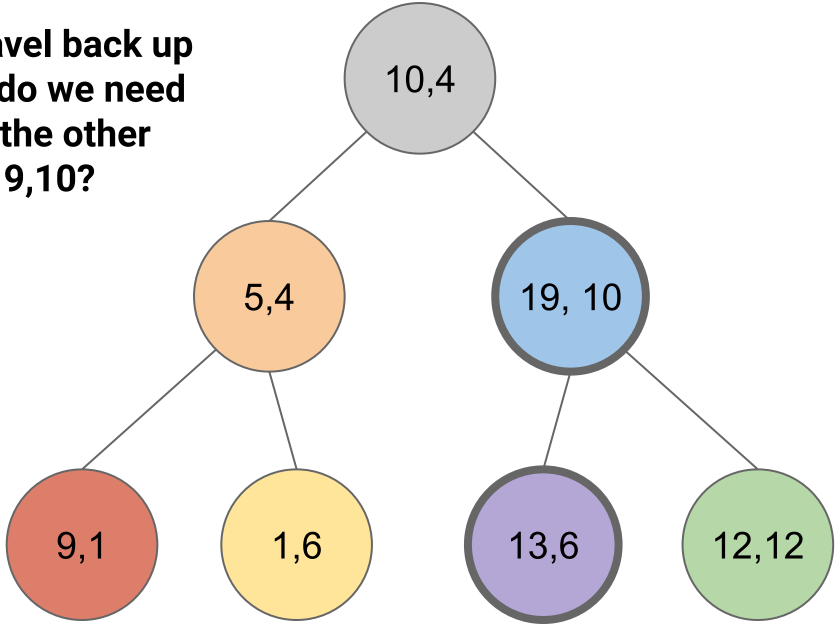
As we travel back up the tree, do we need to check the other child of 19,10?



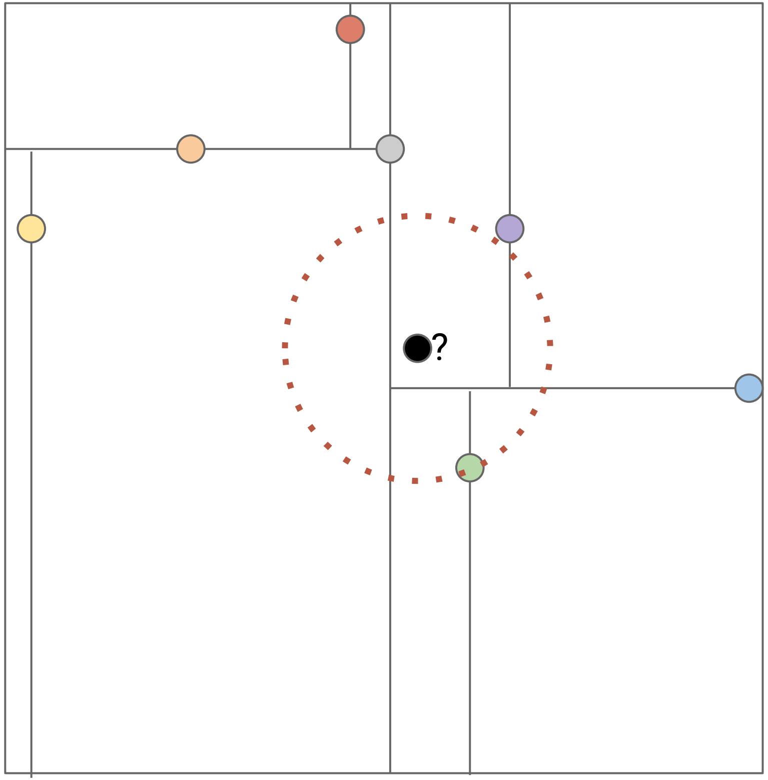
Nearest Neighbor - Example 2



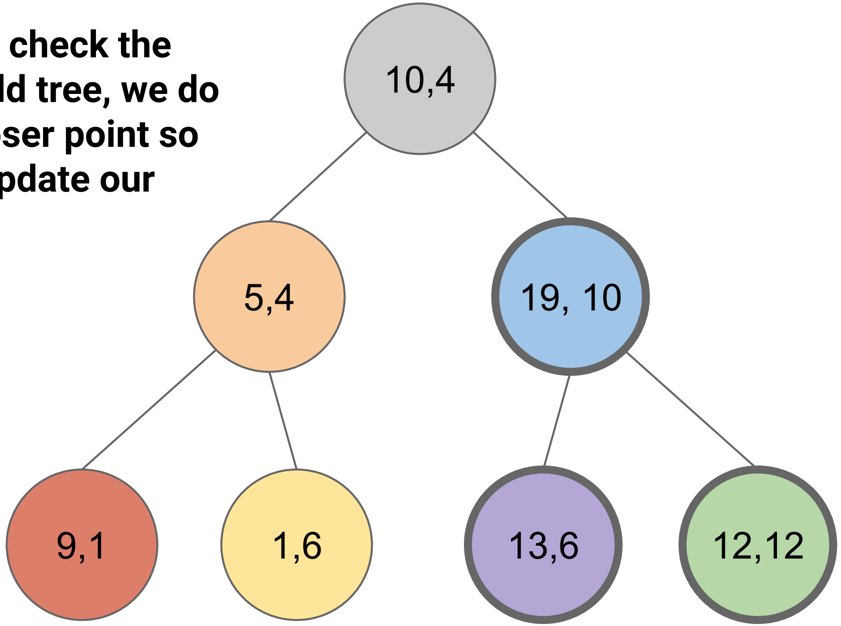
As we travel back up the tree, do we need to check the other child of 19,10?



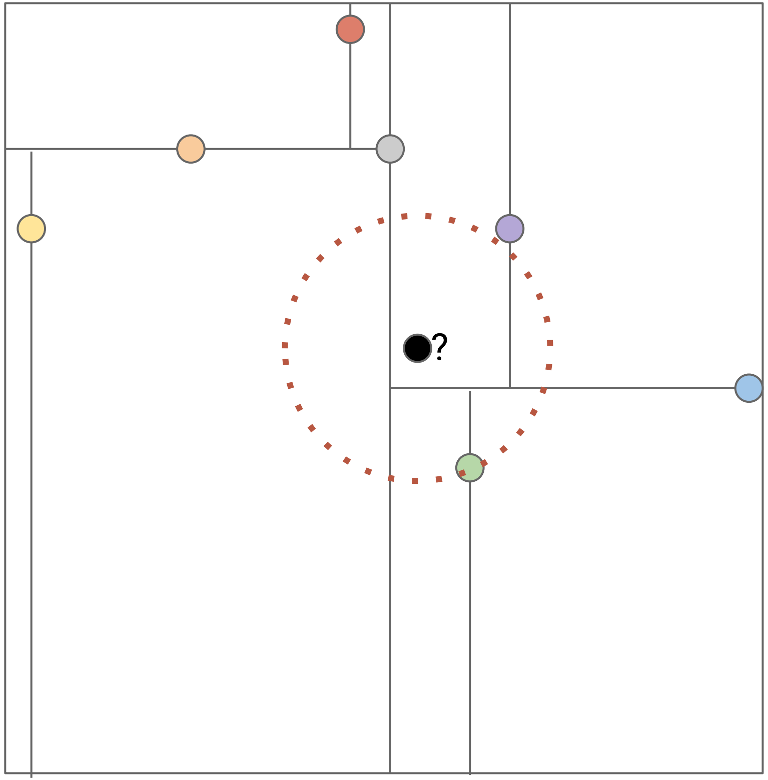
Nearest Neighbor - Example 2



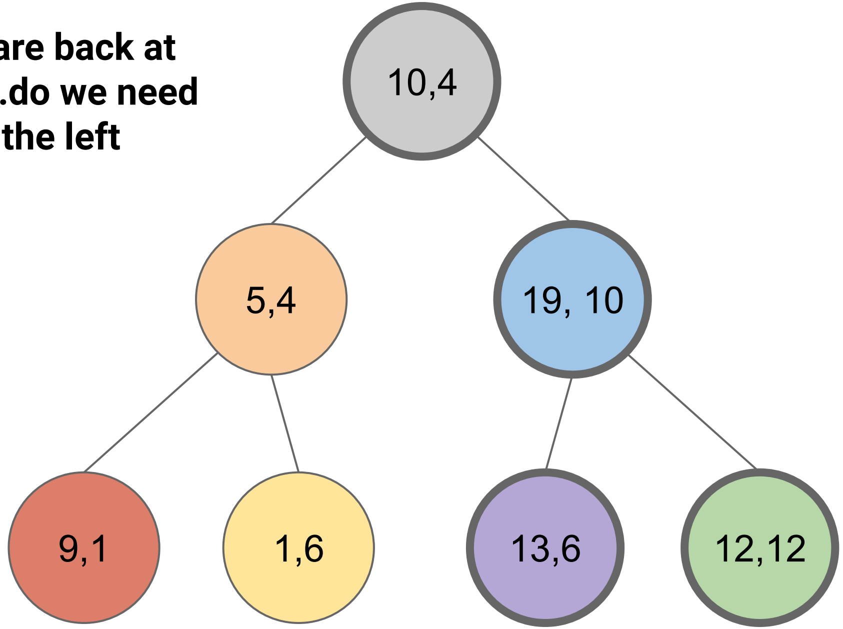
When we check the other child tree, we do find a closer point so we can update our radius.



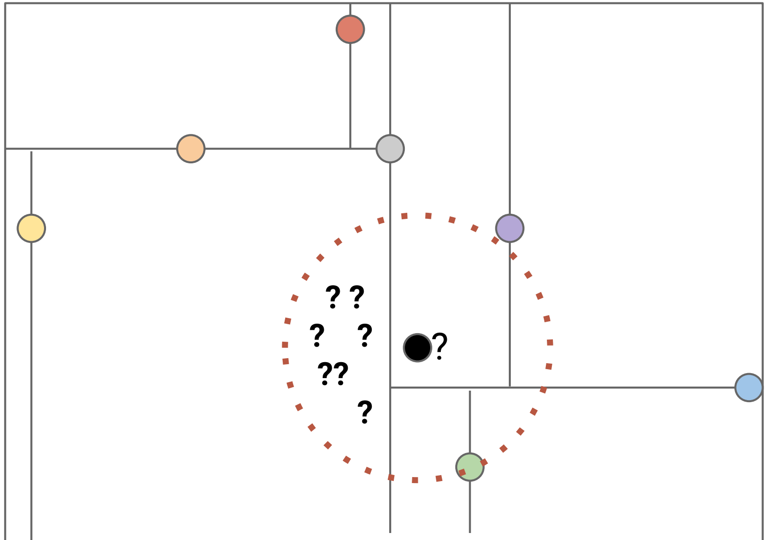
Nearest Neighbor - Example 2



Now we are back at the root...do we need to check the left subtree?

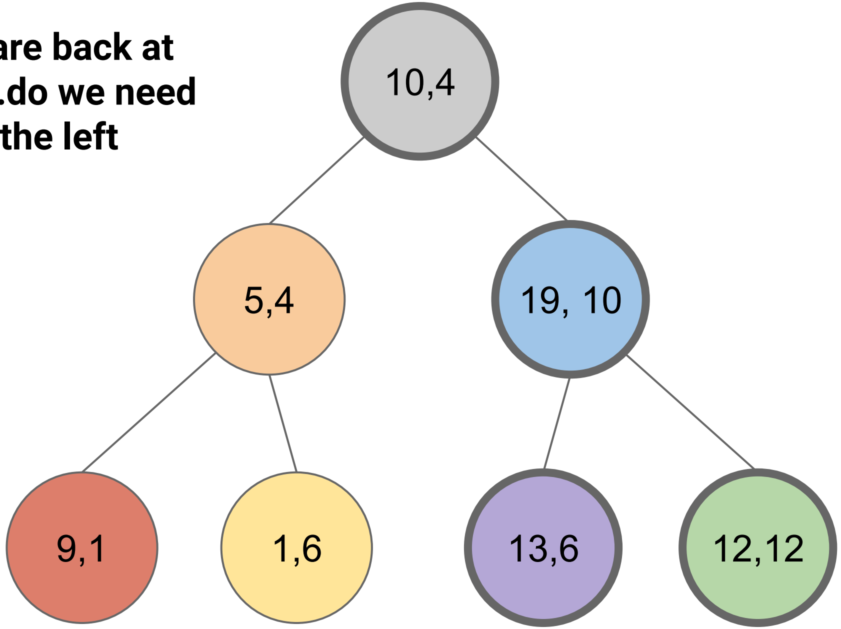


Nearest Neighbor - Example 2



Yes :(
Our search radius intersects with the splitting line, so it's possible there are points in the other subtree closer to us…

Now we are back at the root…do we need to check the left subtree?



Generalization: k-Nearest Neighbors

Finding one point can be as fast as $O(\log(n))$, but as slow as $O(n)$...

What if we want to find the k-Nearest Neighbors instead?

Idea: Keep a list of the k-nearest points, and the furthest point defines our "search radius"

k-D Trees

- Can generalize to k>2 dimensions
  - Depth 0: Partition on Dimension 0
  - Depth 1: Partition on Dimension 1
  - …
  - Depth k-1: Partition on Dimension k-1
  - Depth k: Partition on Dimension 0
  - Depth k+1: Partition on Dimension 1
  - Depth i: Partition on Dimension (i mod k)

The name k-D tree comes from this generalization (k-Dimensional Tree)

- In practice, `range()` and `knn()` become ~ $O(n)$ for k > 3
  - If a subtree's range overlaps with the target in even one dimension, we need to search it. (Curse of Dimensionality)

Other Problems: N-Body Problem

What if we want to compute interactions between one body and every other body?

Naively, this would be $O(n^2)$...but likely we don't care as much about interactions with bodies that are very very far away.

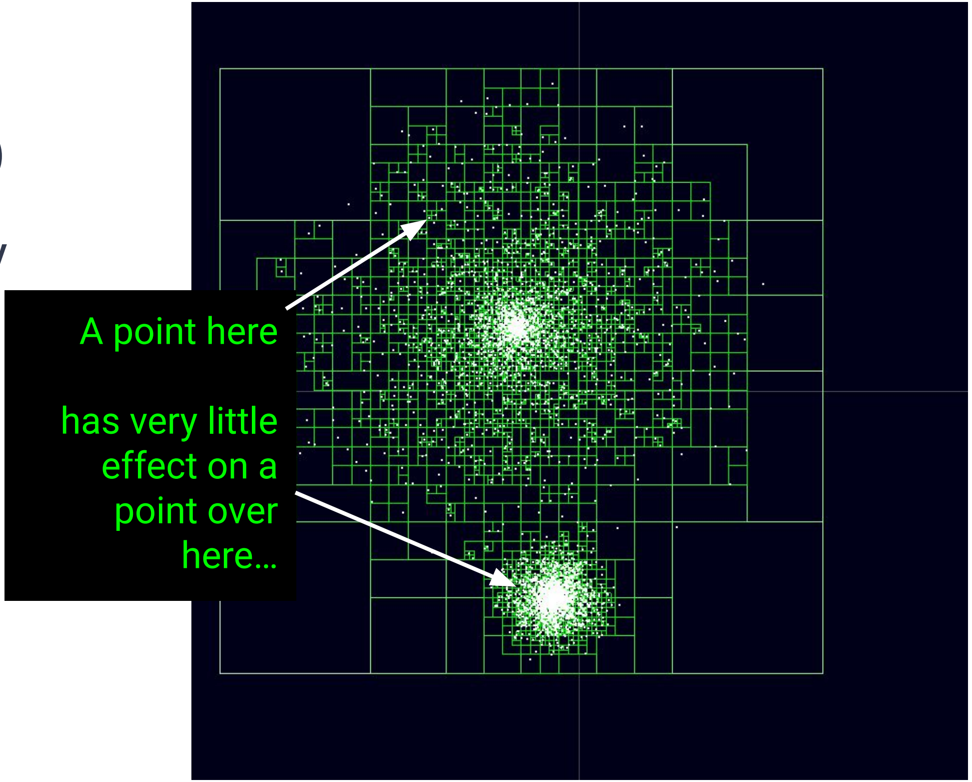
Other Problems: N-Body Problem

Idea: Divide our points into a quadtree (or octree in 3 dimensions)

Do full calculation for points closeby (in the same box)

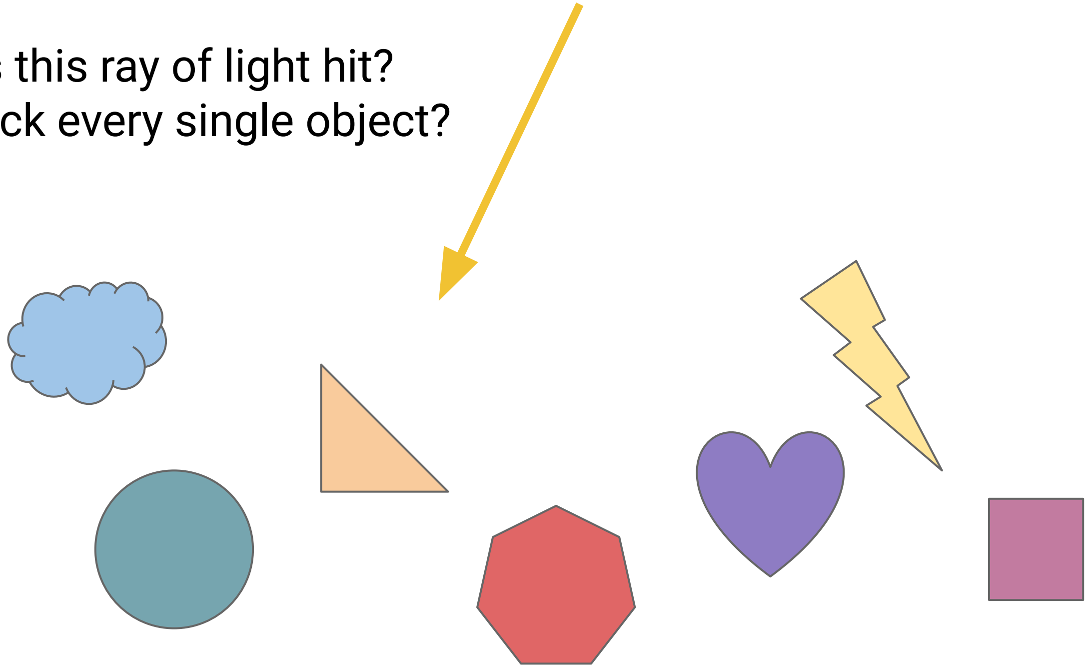
Compute a summary (ie total force and center of mass) for each box that can be applied to far away boxes

Runtime is now ***O*(*n*log(*n*))**



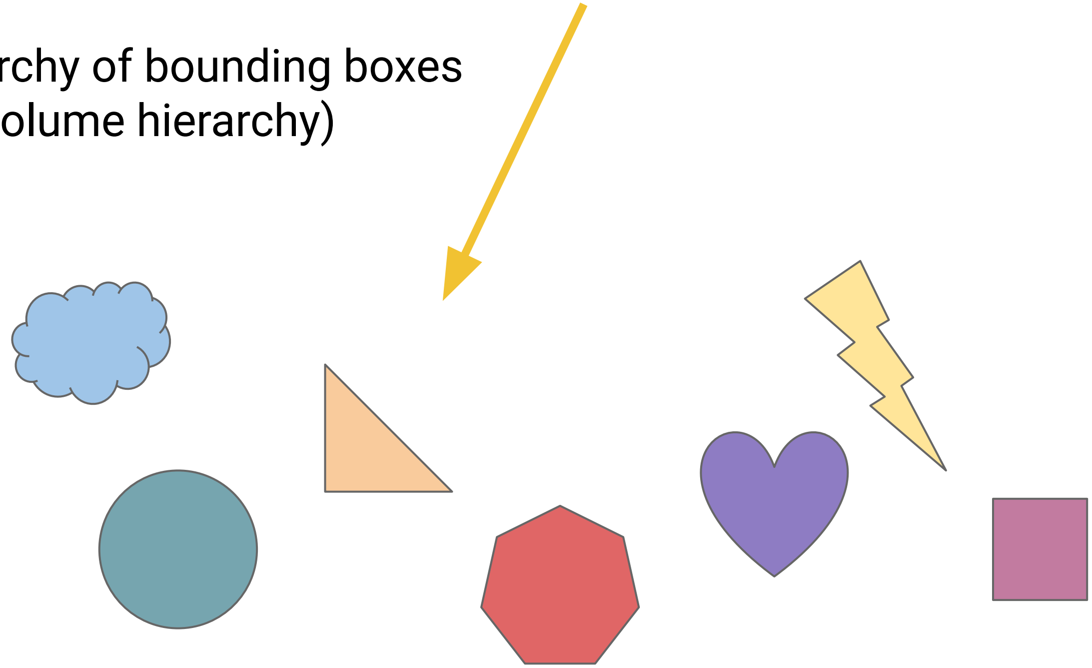
Other Problems: Ray/Path Tracing

Which object does this ray of light hit?
Do we have to check every single object?



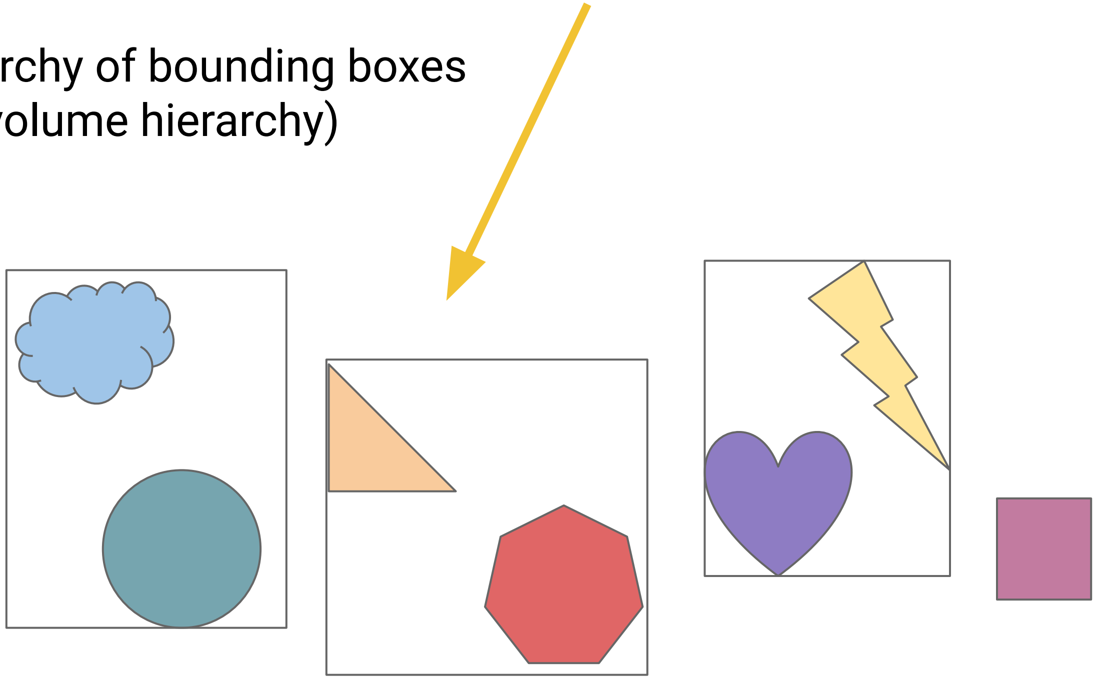
Other Problems: Ray/Path Tracing

Idea: Build a hierarchy of bounding boxes
(BVH - Bounding volume hierarchy)



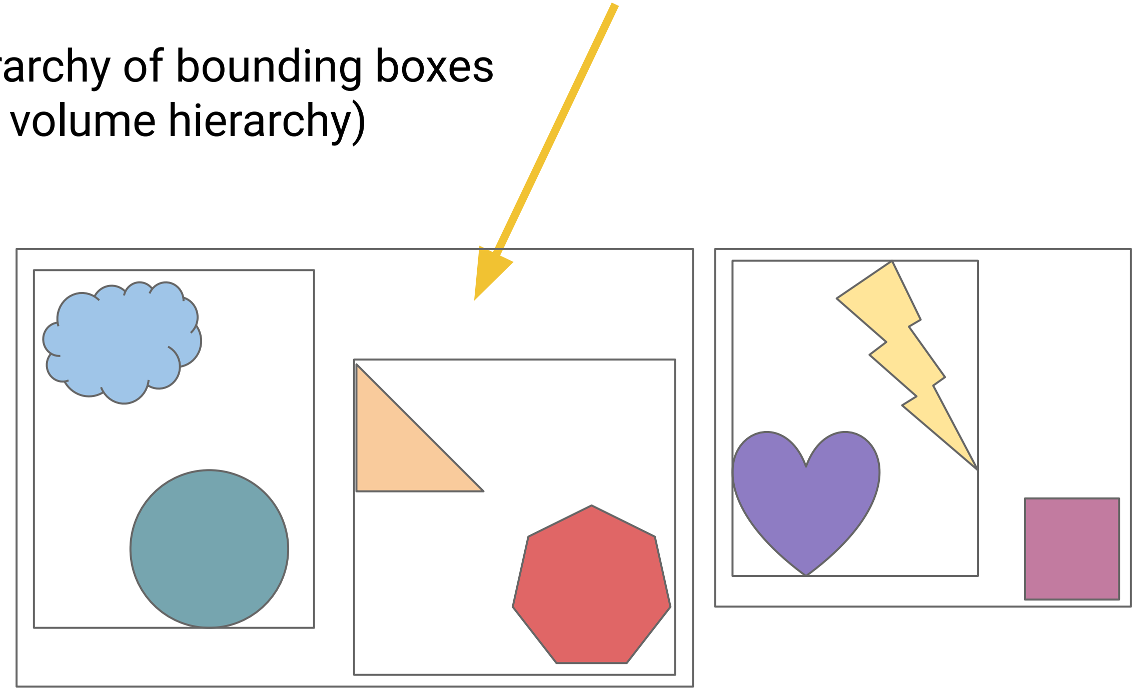
Other Problems: Ray/Path Tracing

Idea: Build a hierarchy of bounding boxes
(BVH - Bounding volume hierarchy)



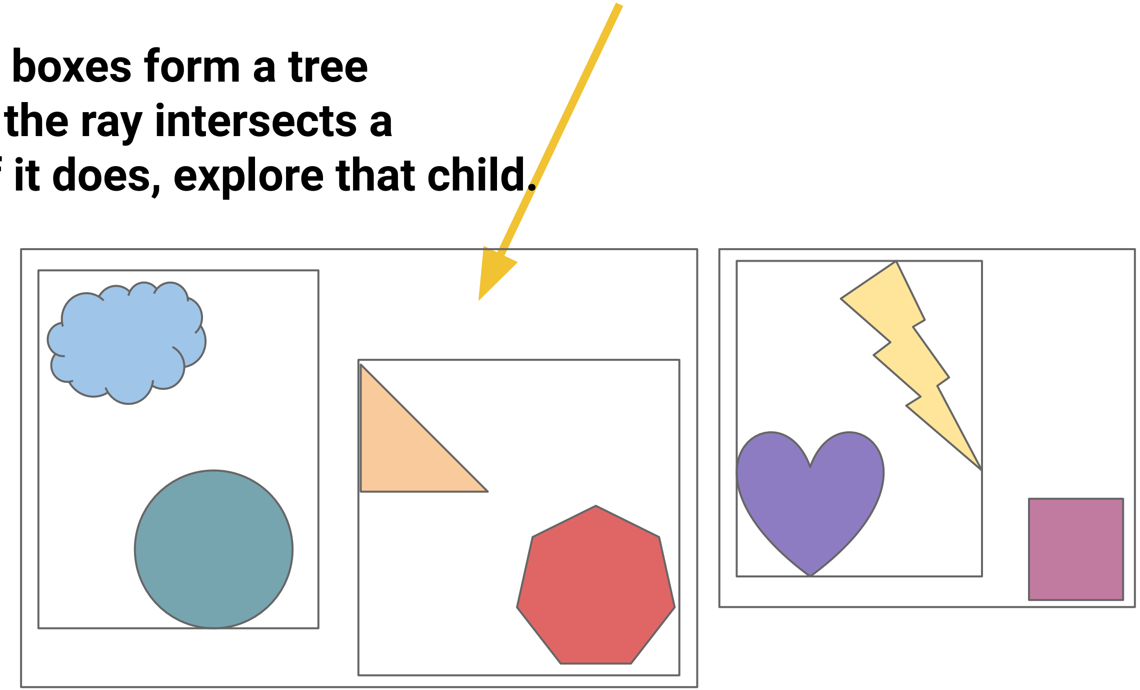
Other Problems: Ray/Path Tracing

Idea: Build a hierarchy of bounding boxes
(BVH - Bounding volume hierarchy)



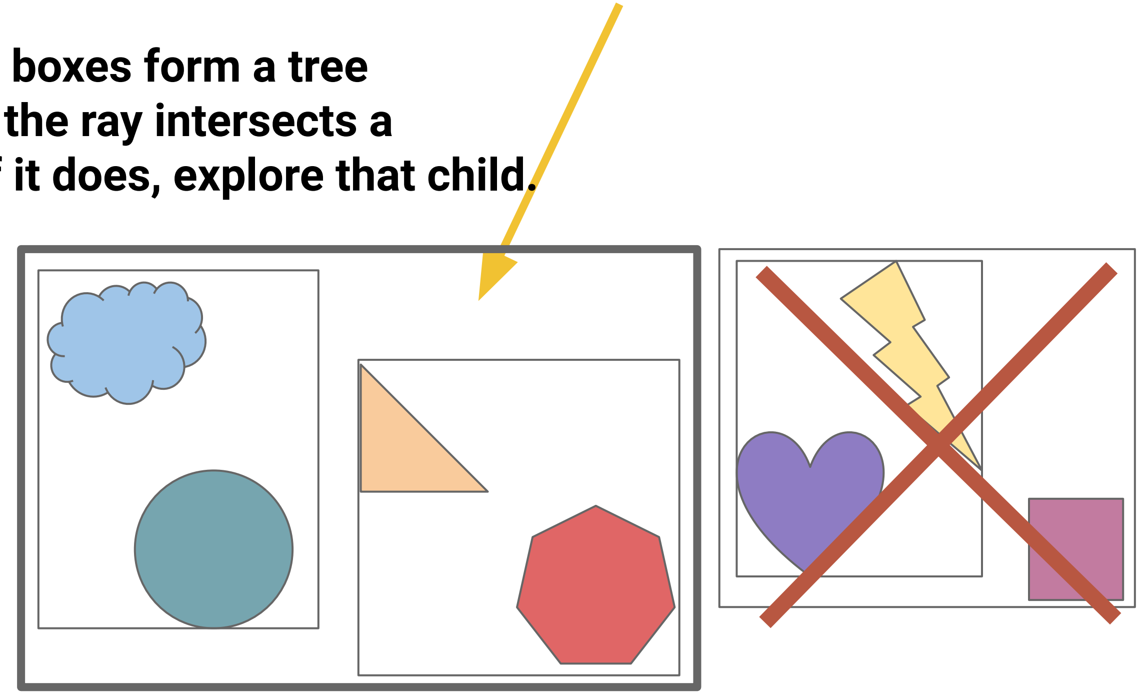
Other Problems: Ray/Path Tracing

These bounding boxes form a tree
We can check if the ray intersects a bounding box. If it does, explore that child. If not, ignore it.



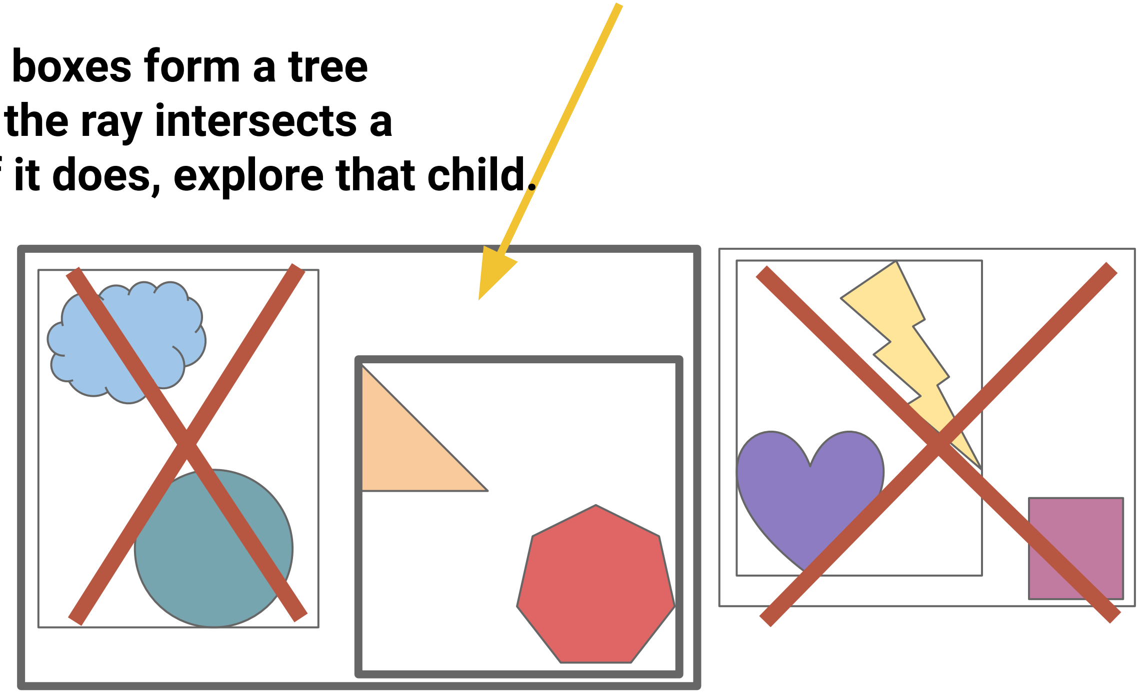
Other Problems: Ray/Path Tracing

These bounding boxes form a tree
We can check if the ray intersects a bounding box. If it does, explore that child. If not, ignore it.



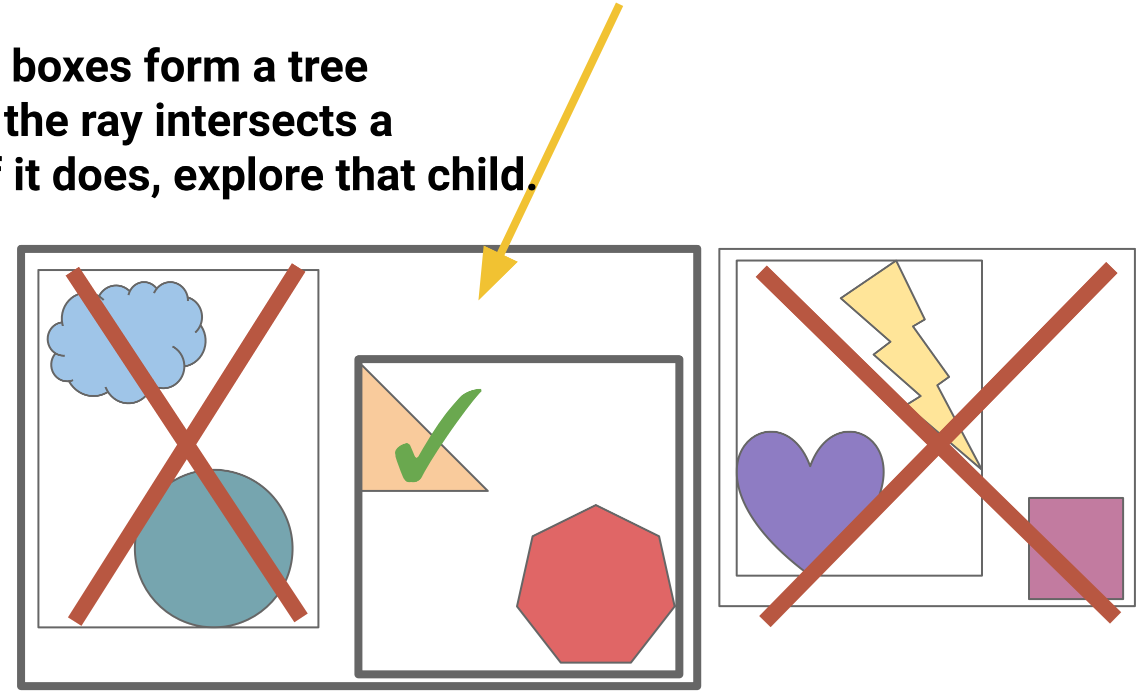
Other Problems: Ray/Path Tracing

These bounding boxes form a tree
We can check if the ray intersects a bounding box. If it does, explore that child. If not, ignore it.



Other Problems: Ray/Path Tracing

These bounding boxes form a tree
We can check if the ray intersects a bounding box. If it does, explore that child. If not, ignore it.



Other Problems: Ray/Path Tracing

If we build our BVH effectively, the runtime becomes logarithmic.