Geometric range searching

Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

Where we are

"Global" problems

- closest pair
- convex hull
- intersections
- ..

Geometric search problems

- range searching
- nearest neighbor
- k-nearest neighbor
- find all roads within 1km of current location
- ..

Techniques

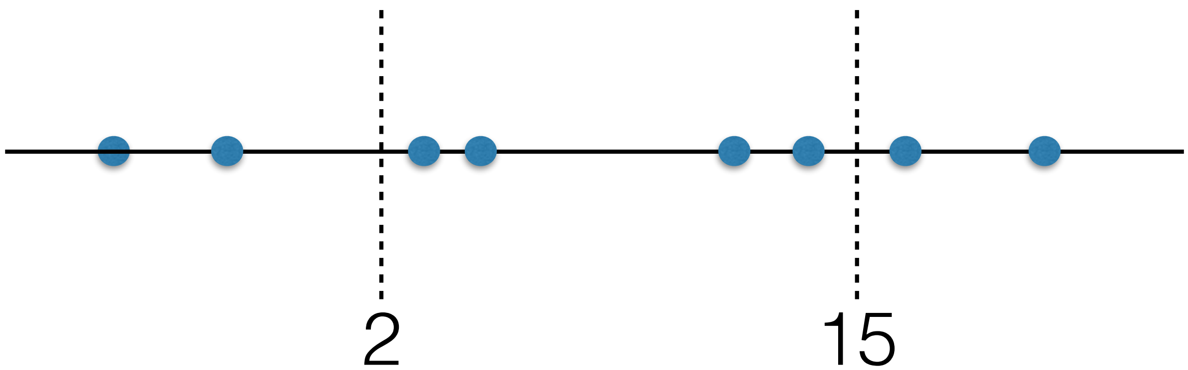
- divide-and-conquer
- incremental
- space decomposition
- plane sweep
- …

next

We start with1D range searching

1D Range searching

Given a set of n points on the real line and an interval [a,b], find all points in [a,b]



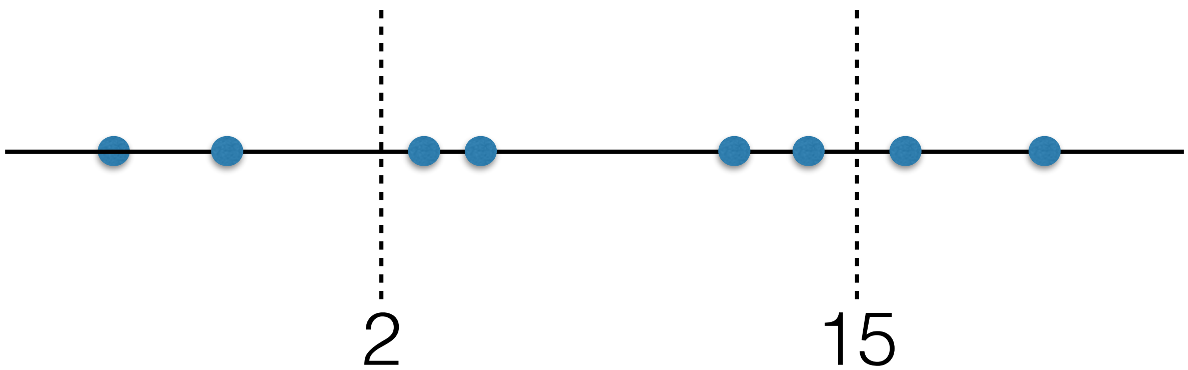
find all values in [2,15]

Assume first that the points are **fixed**, i.e. don't change.

What can we do?

1D Range searching

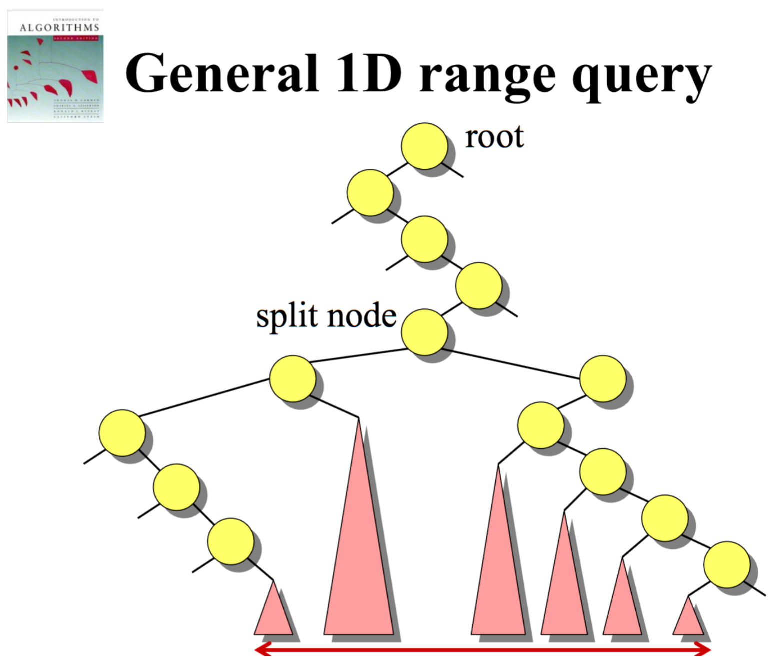
Given a set of n points on the real line and an interval [a,b], find all points in [a,b]



Assume now that the points are **dynamic**, i.e. in addition to range queries, we want to be able to **insert** and **delete** points.

1D Range searching

- A set of n points in 1D can be pre-processed into a BBST such that:
  - Build: $O(n \lg n)$
  - Space: $\Theta(n)$
  - Range queries: $O(\lg n + k)$
  - Dynamic: points can be inserted/deleted in $O(\lg n)$



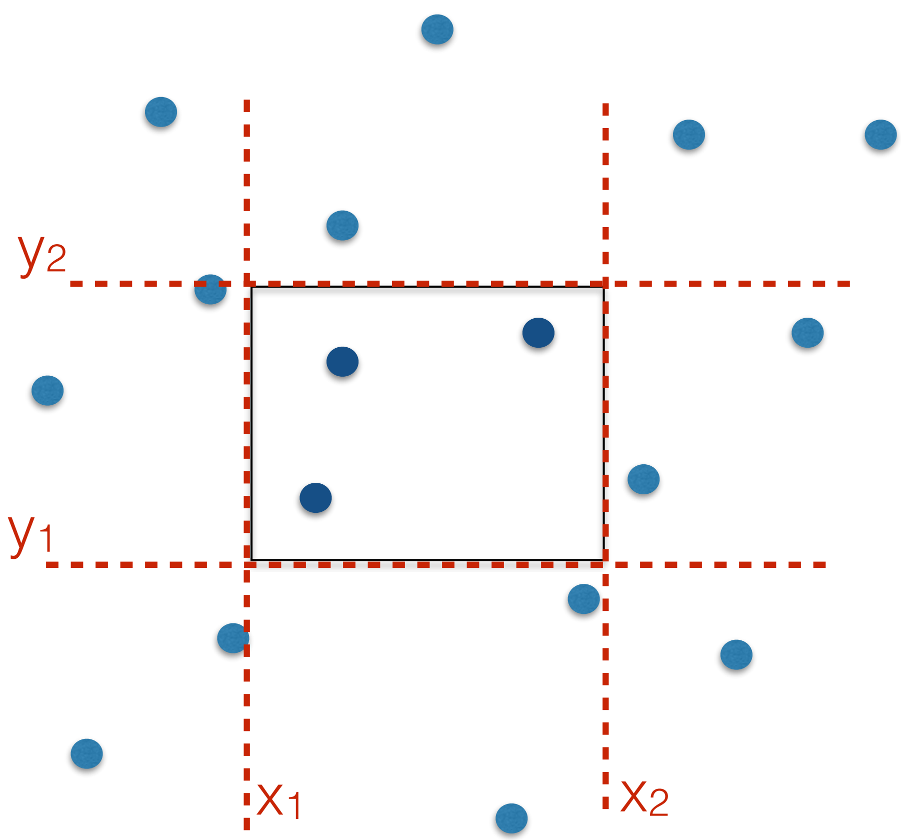
The k points in the range sit in O(lg n) subtrees

1D Range searching

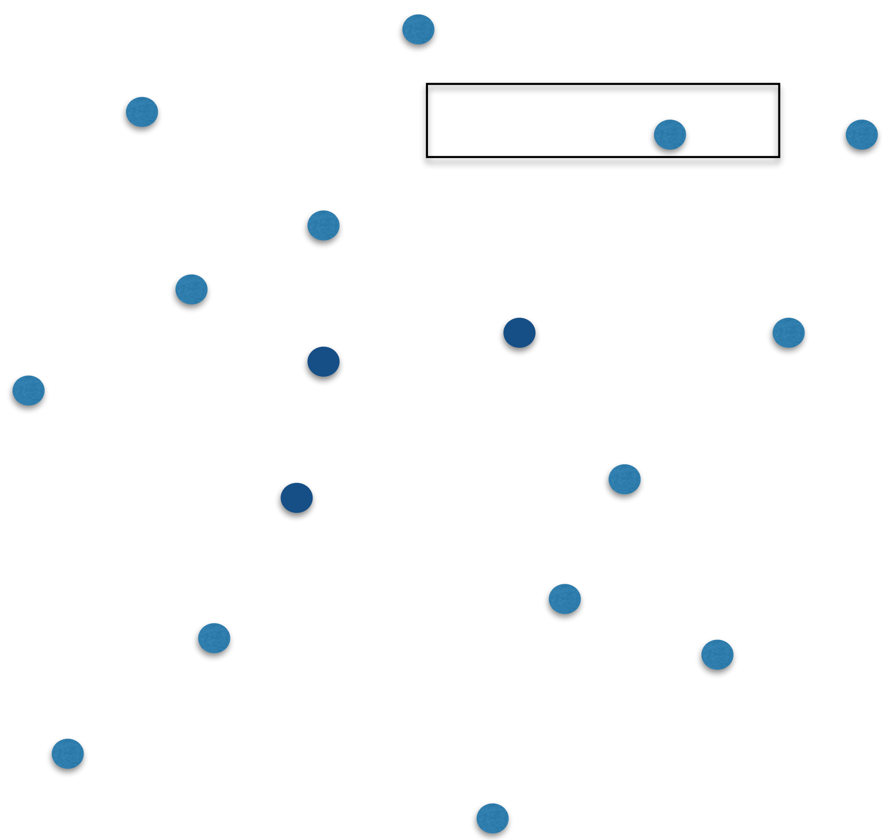
- A set of n points in 1D can be pre-processed into a BBST such that:
  - Build: $O(n \lg n)$
  - Space: $\Theta(n)$
  - Range queries: $O(\lg n + k)$
  - Dynamic: points can be inserted/deleted in $O(\lg n)$

2D Range searching

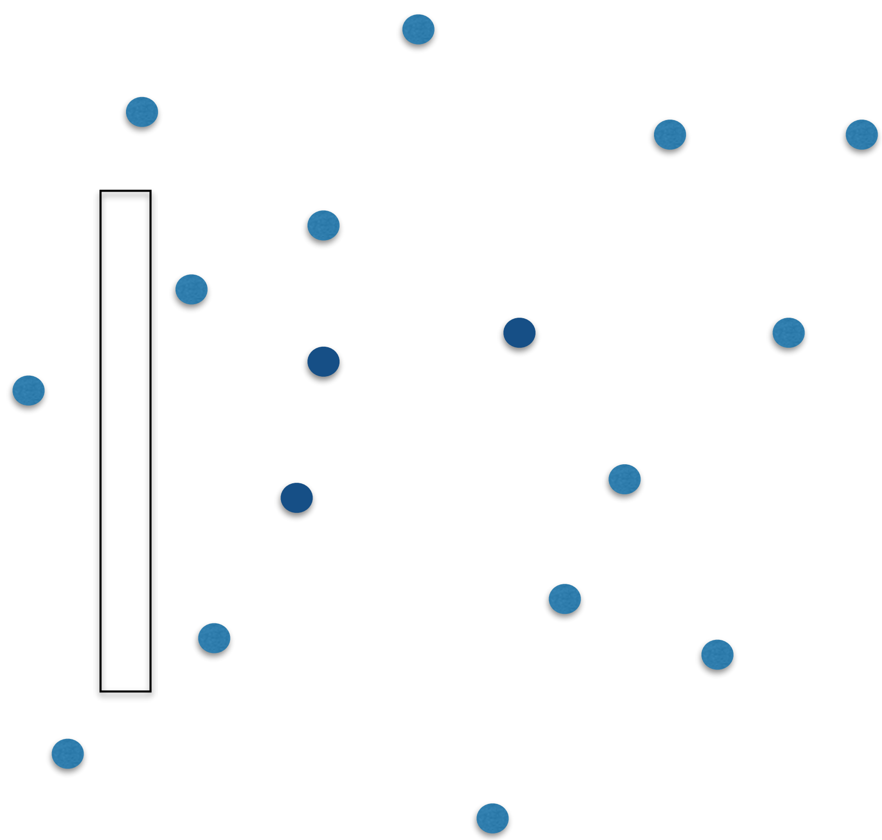
Given a set of n points in 2D and an arbitrary range $[x_1, x_2] \times [y_1, y_2]$, find all points in this range



2D Range searching



2D Range searching

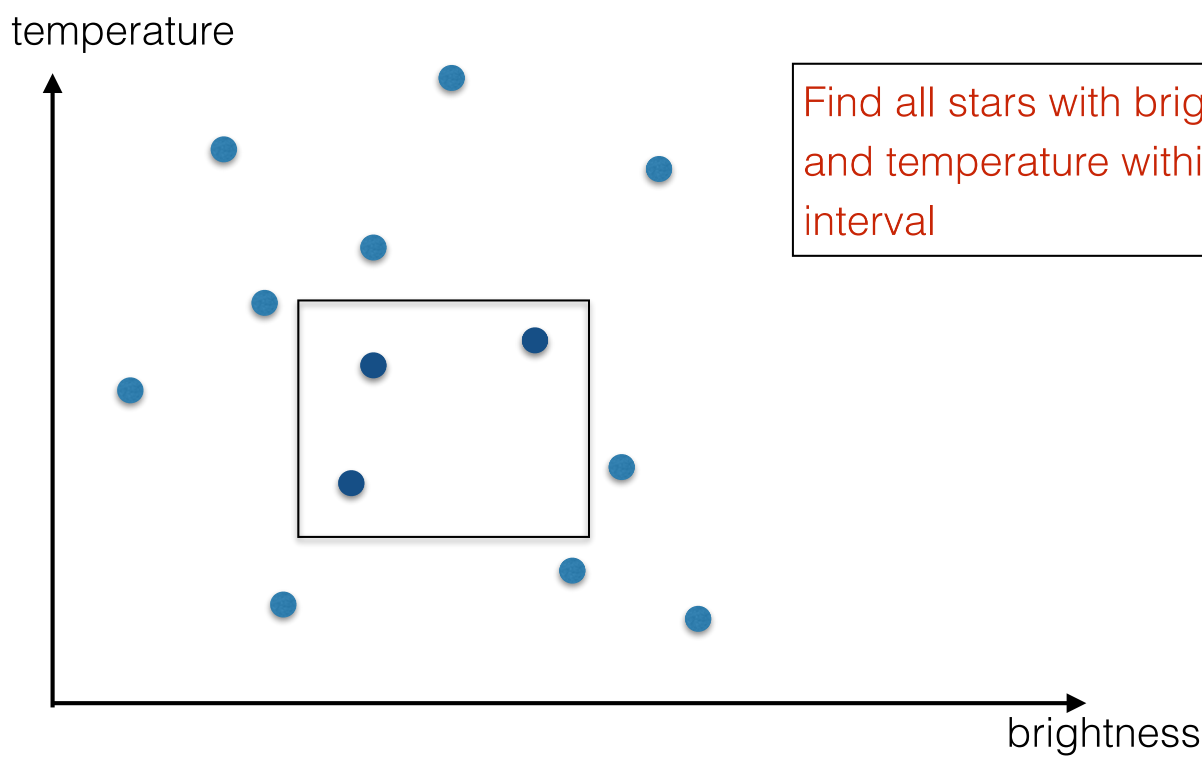


Why range searching?

Searching is a fundamental operation. This is the multi-dimensional version of the “report all points in this interval”

Interestingly, it comes up in settings that are not geometrical

e.g. Database of stars. A star = (brightness, temperature,……)

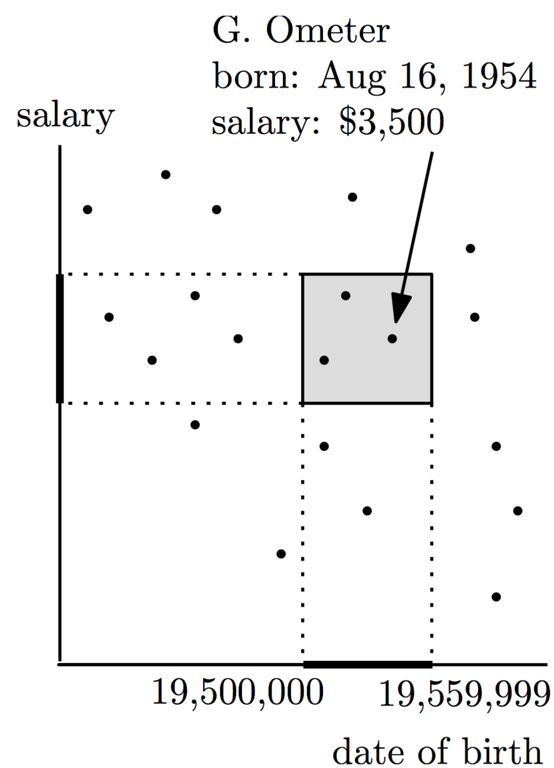


Find all stars with brightness and temperature within a given interval

Why range searching?

e.g. Database of employees. An employee = (age, salary,……)

A database query may ask for all employees with age between a_1 and a_2, and salary between s_1 and s_2

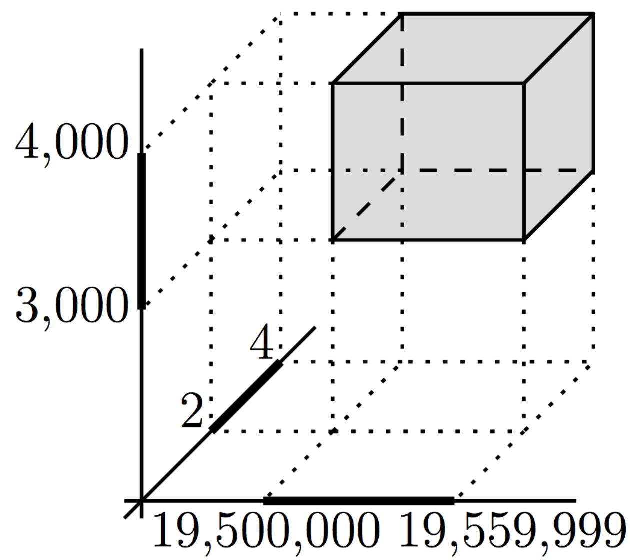


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

Why range searching?

3d-range searching, etc

Example of a 3-dimensional (orthogonal) range query: children in $[2, 4]$, salary in $[3000, 4000]$, date of birth in $[19,500,000\ ,\ 19,559,999]$

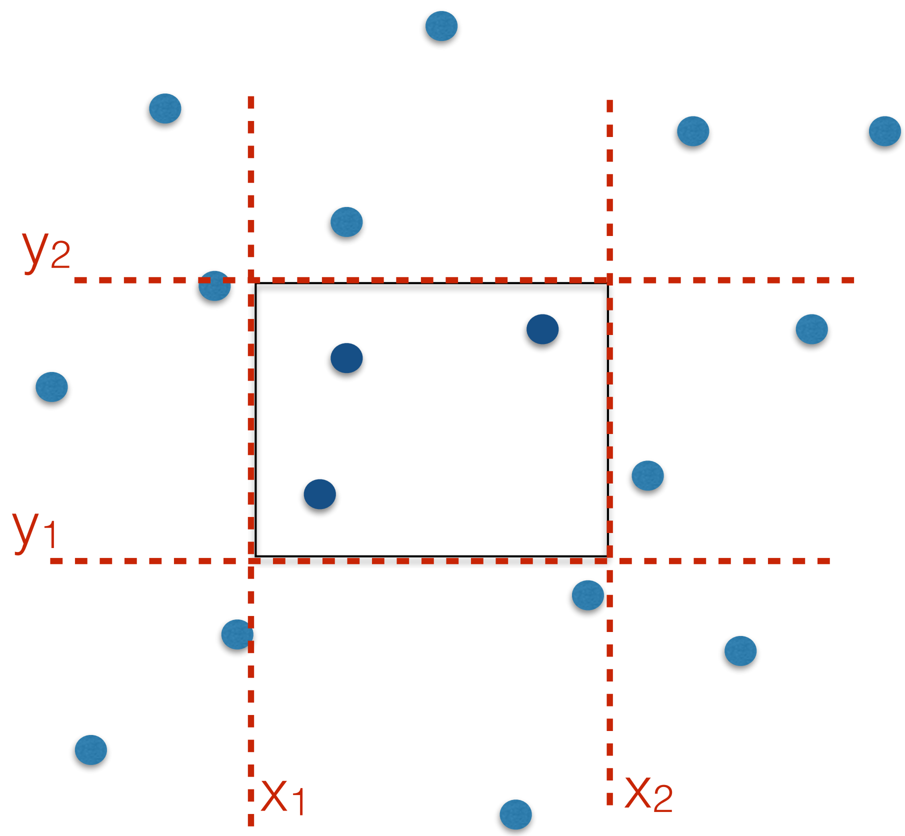


screenshot from Mark van Kreveld slides at http://www.cs.uu.nl/docs/vakken/ga/slides5a.pdf

2D Range searching

Given a set of n points in 2D and an arbitrary range $[x_1, x_2] \times [y_1, y_2]$, find all points in this range

Build a structure to answer this efficiently



1D

- A set of n points in 1D can be pre-processed into a BBST such that:
  - Build: $O(n \lg n)$
  - Space: $\Theta(n)$
  - Range queries: $O(\lg n + k)$
  - Dynamic: points can be inserted/deleted in $O(\lg n)$

2D

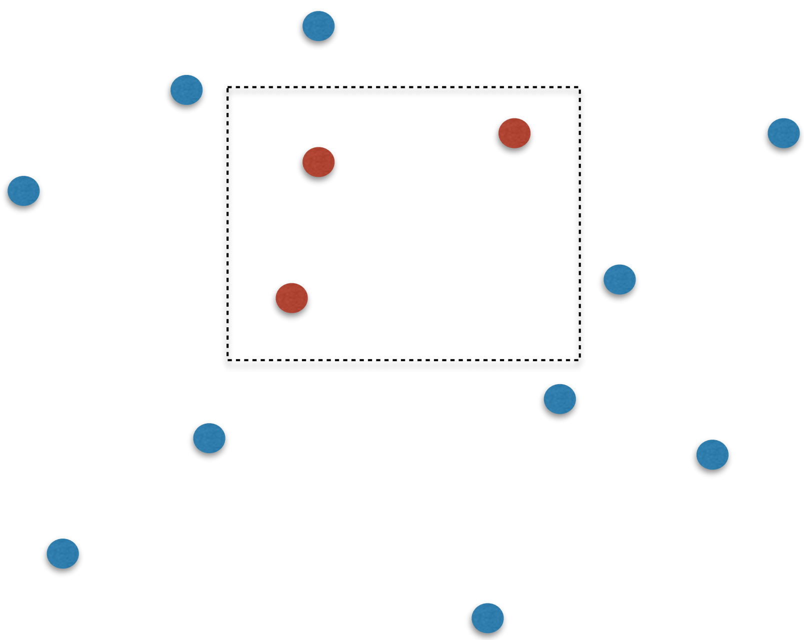
- A set of n points in 2D can be pre-precessed in a ??2d-BBST??? such that
  - Build: $O(n \lg n)$
  - Space: $\Theta(n)$
  - Range queries: $O(\lg^2 n + k)$

These bounds would be nice

But how?

- n: size of the input (number of points)
- k: size of output (number of points inside range)

2D Naive Approach



Points are static or dynamic?

We'll assume static (it's hard enough)

The naive approach: just traverse and check in O(n)

Analysis:

Build: none

Space: none

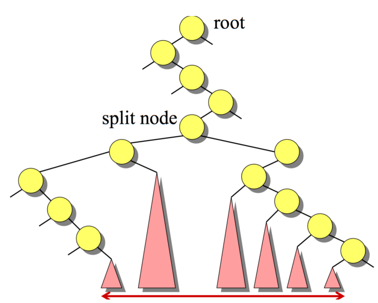
2d range query: $O(n)$ ← We want $O(\lg^2 n + k)$

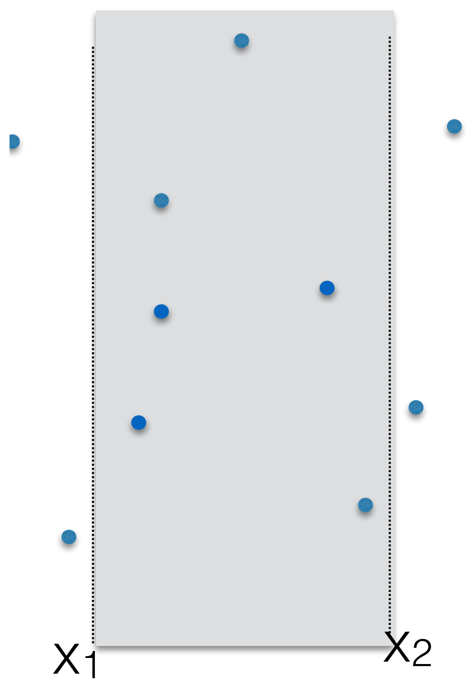
query $[x_1, x_2] \times [y_1, y_2]$

How about this:

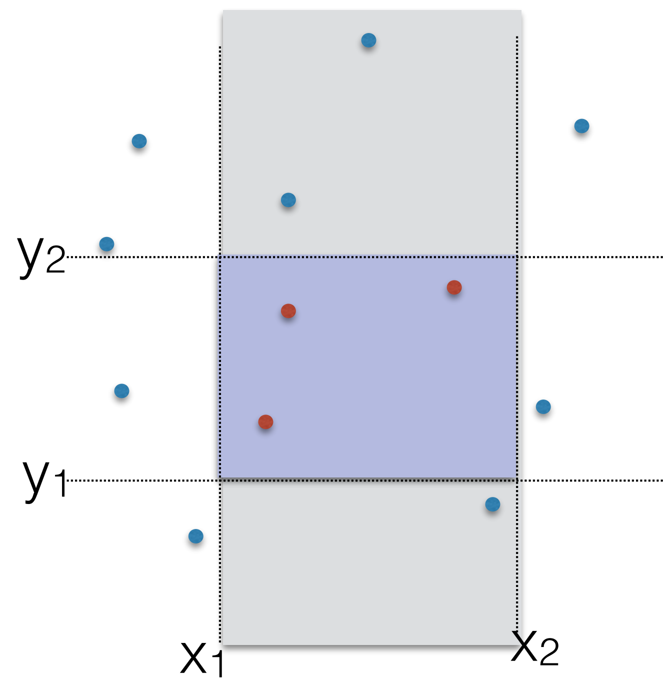
1. Find all points with the x-coords in $[x_1, x_2]$

BBST in x-order



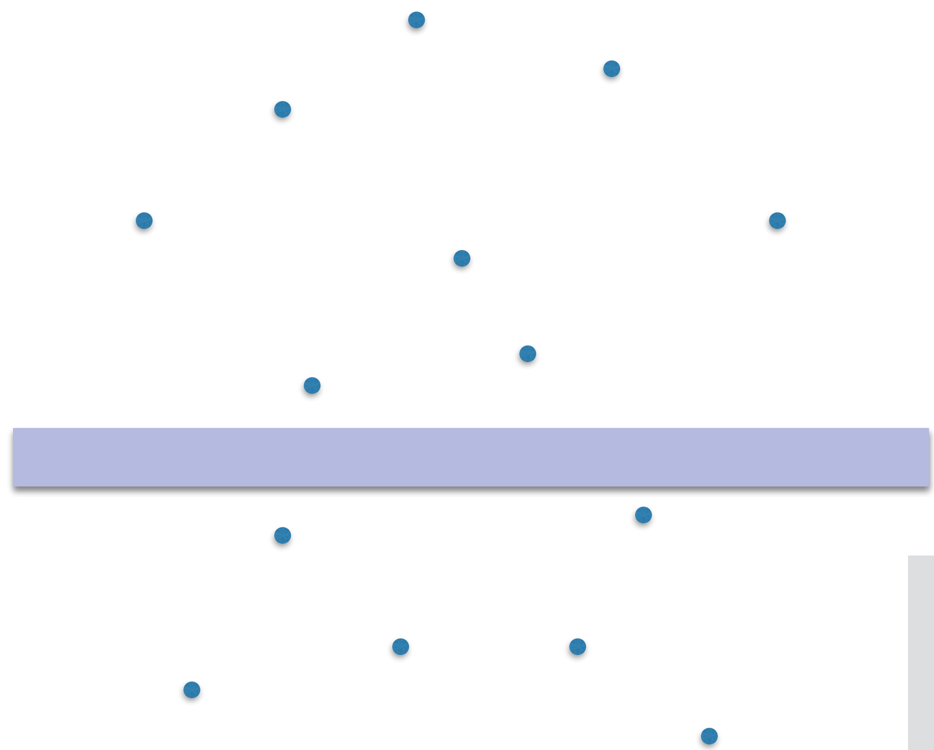


2. Traverse these points and find those with y-coord in $[y_1, y_2]$



- This works, but it's worst case is slow
- $O(\lg n + k')$ to find the points in the vertical strip, and then $O(k')$ to traverse and find the points in $[y_1, y_2]$

- The problem is that the nb. of points in $[x_1, x_2]$ can be large, and the nb. of points in $[x_1, x_2] \times [y_1, y_2]$ may be small
- Worst case: $k' = n, k = 0$



to find the points in this range takes $O(\lg n + n) = O(n)$ time

Searching via space partition structures

We'll partition the space, store it in a data structure and use it to speed up searching

the grid heuristic

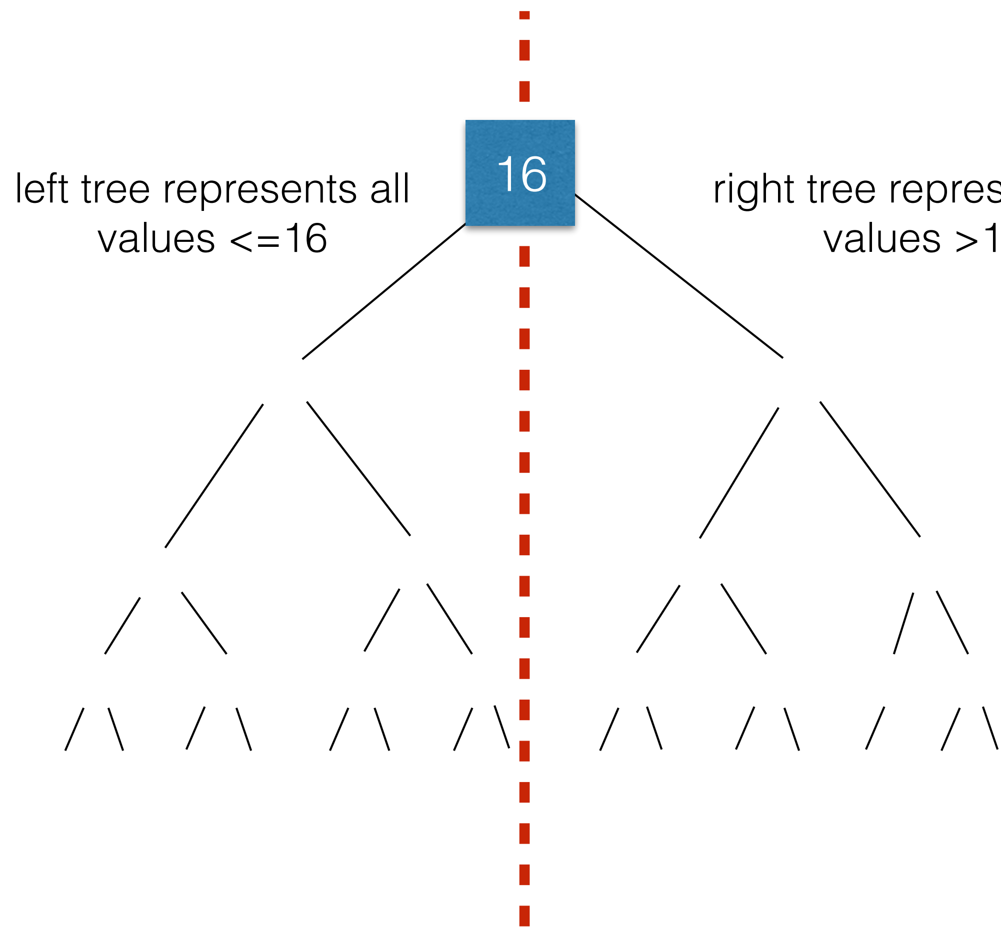
kd-trees

range-trees

First, let’s look in 1D:

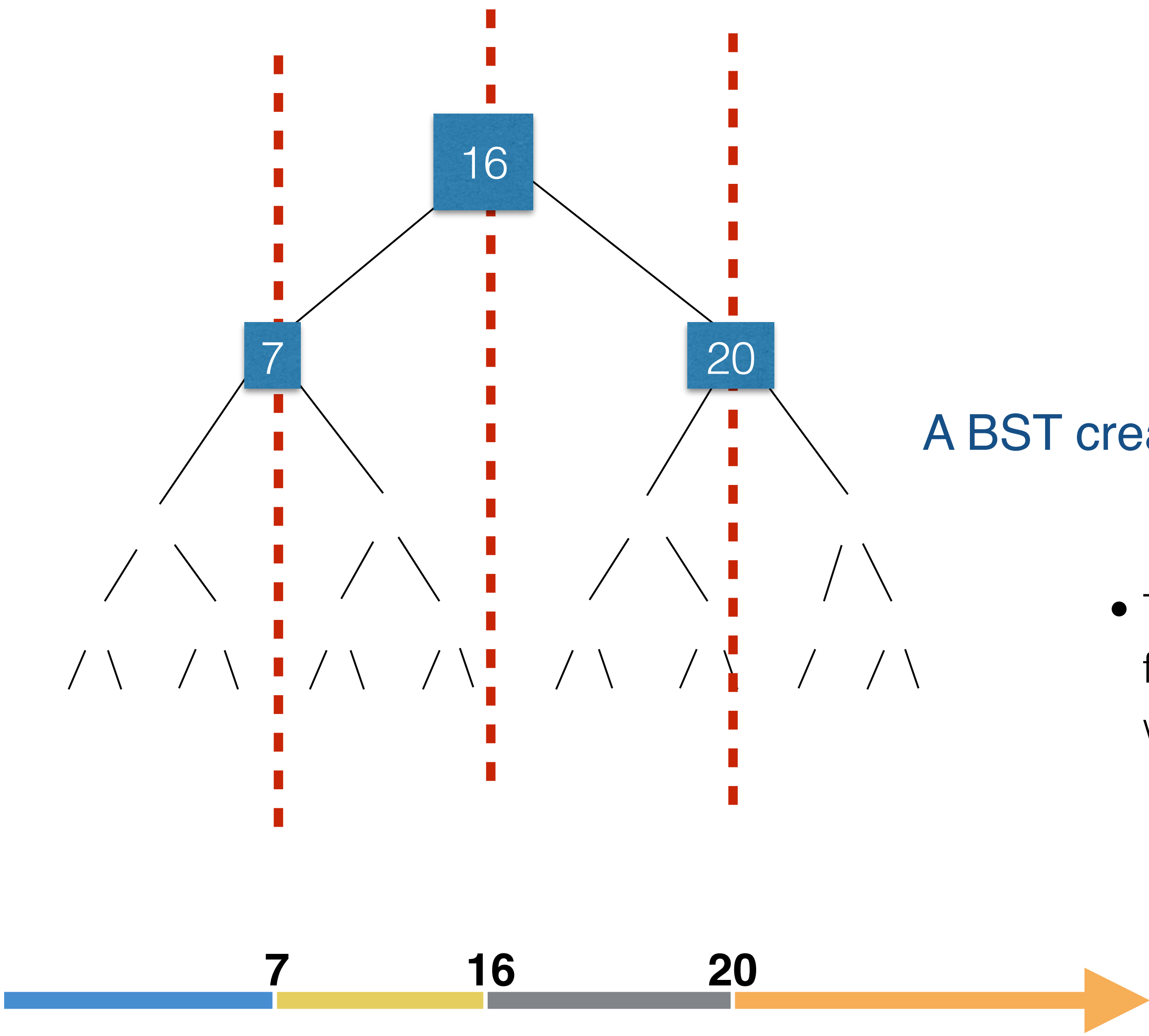
The BBST as a space partition structure

First, let’s look in 1D: The BST as a space partition structure



A BST creates an implicit space partition

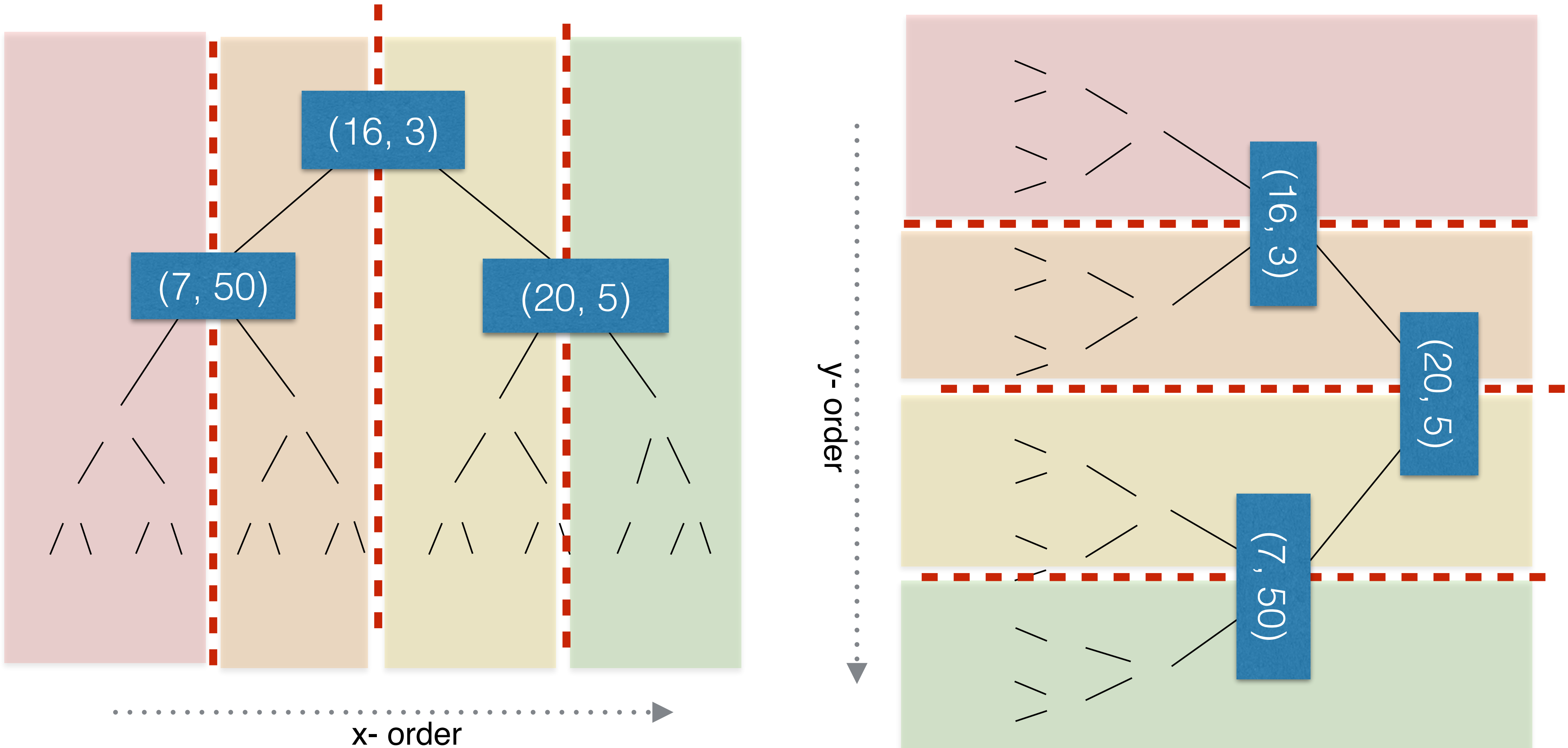
First, let's look in 1D: The BST as a space partition structure



A BST creates an implicit space partition

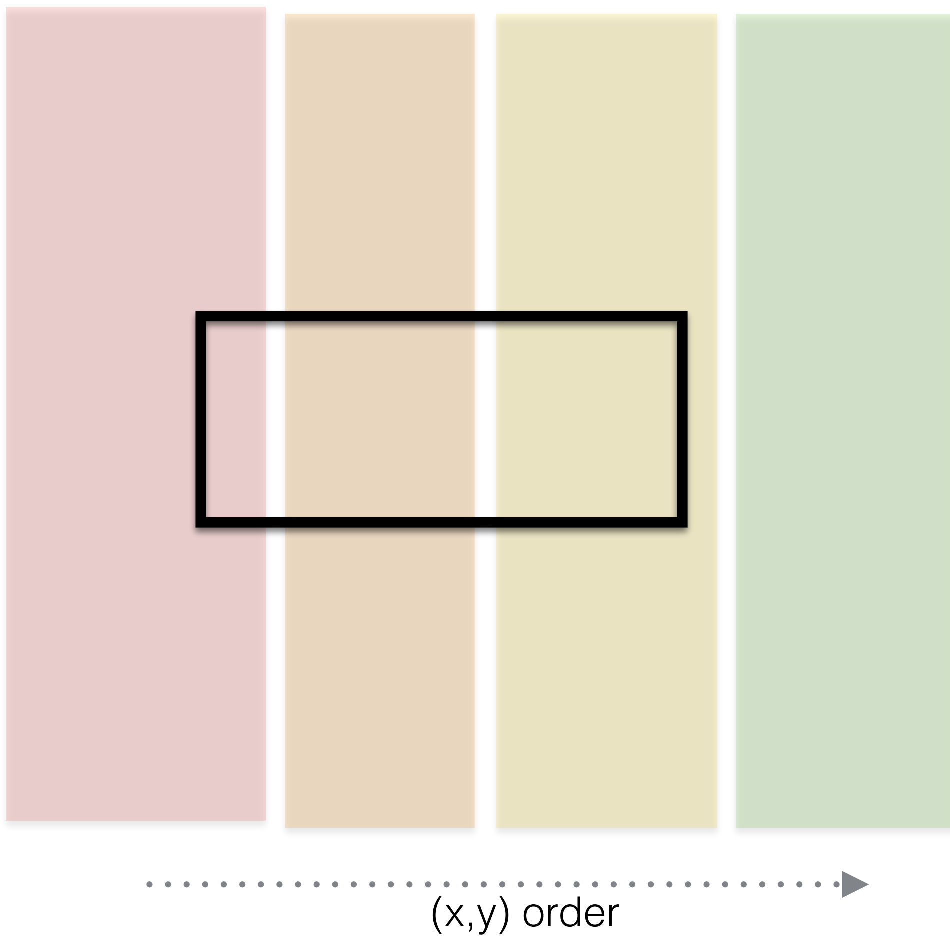
- To search for a value we need to find the region of space that would contain this value

Using a BST on 2d points



Partitions the space in vertical/horizontal stripes

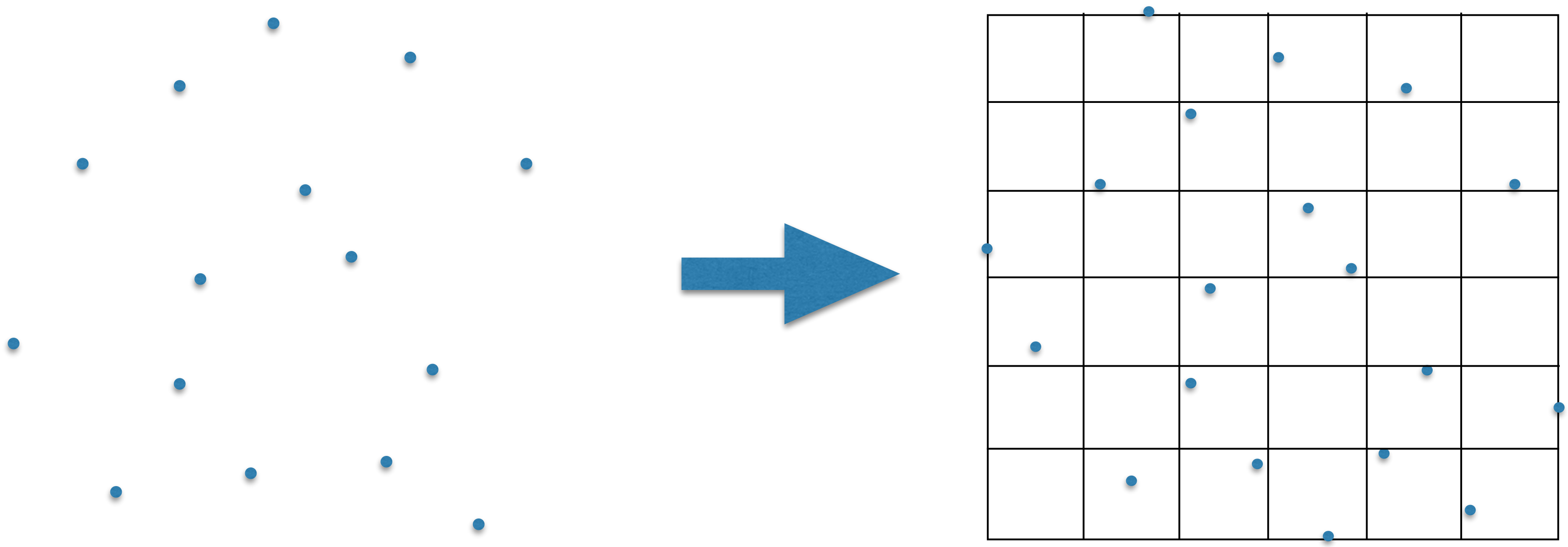
Not a good partition for range-searching!



We have to search all vertical strips that intersect the range, which could have a lot of points outside the range.

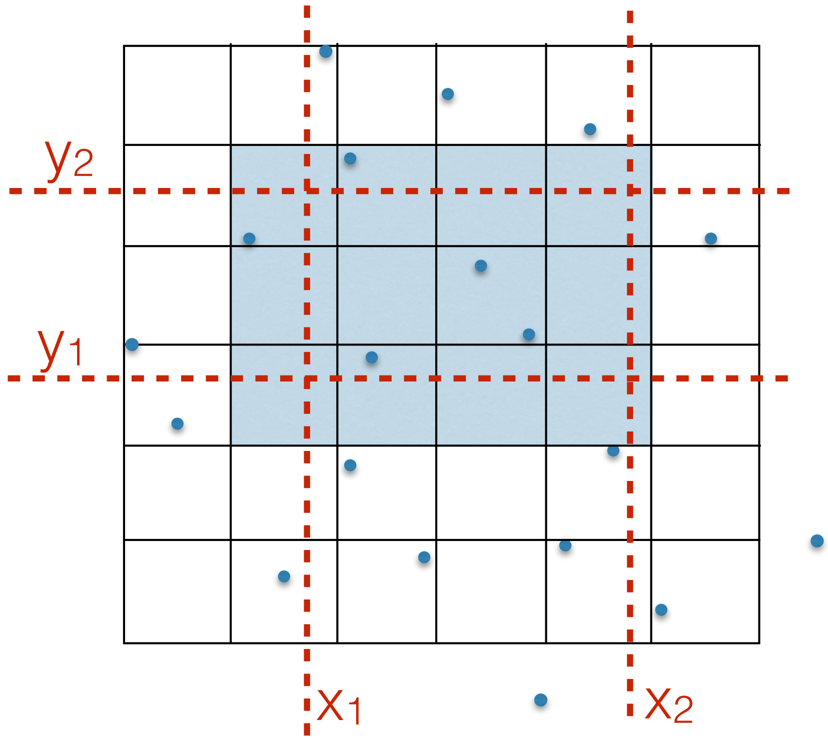
The grid

The simplest space decomposition is a grid



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

2D range searches with a grid



- 2d range queries: traverse all cells that intersect the range
- Exact bound depends on how many points are in the cells
- Choose grid size $m = O(\sqrt{n})$ and hope for $O(1)$ points per cell. In this case, a range query takes $O(k)$
- Worst case is bad: points are not uniformly distributed, a range query could take $O(n)$ even if no points are reported

The grid method

+ Simple to implement

+ Perform well if points are uniformly distributed

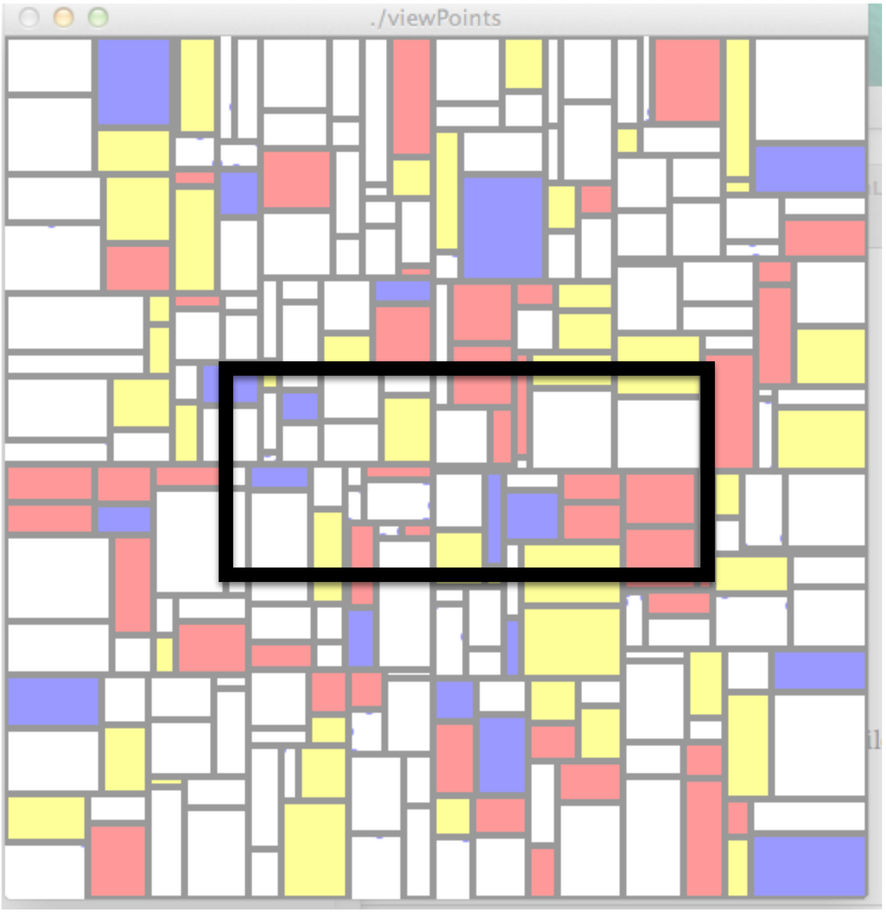
+ Can be used for many other problems besides range searching (e.g. closest pair, neighbor queries)

- Gridding gives no guarantee on bounds. It's an heuristic.

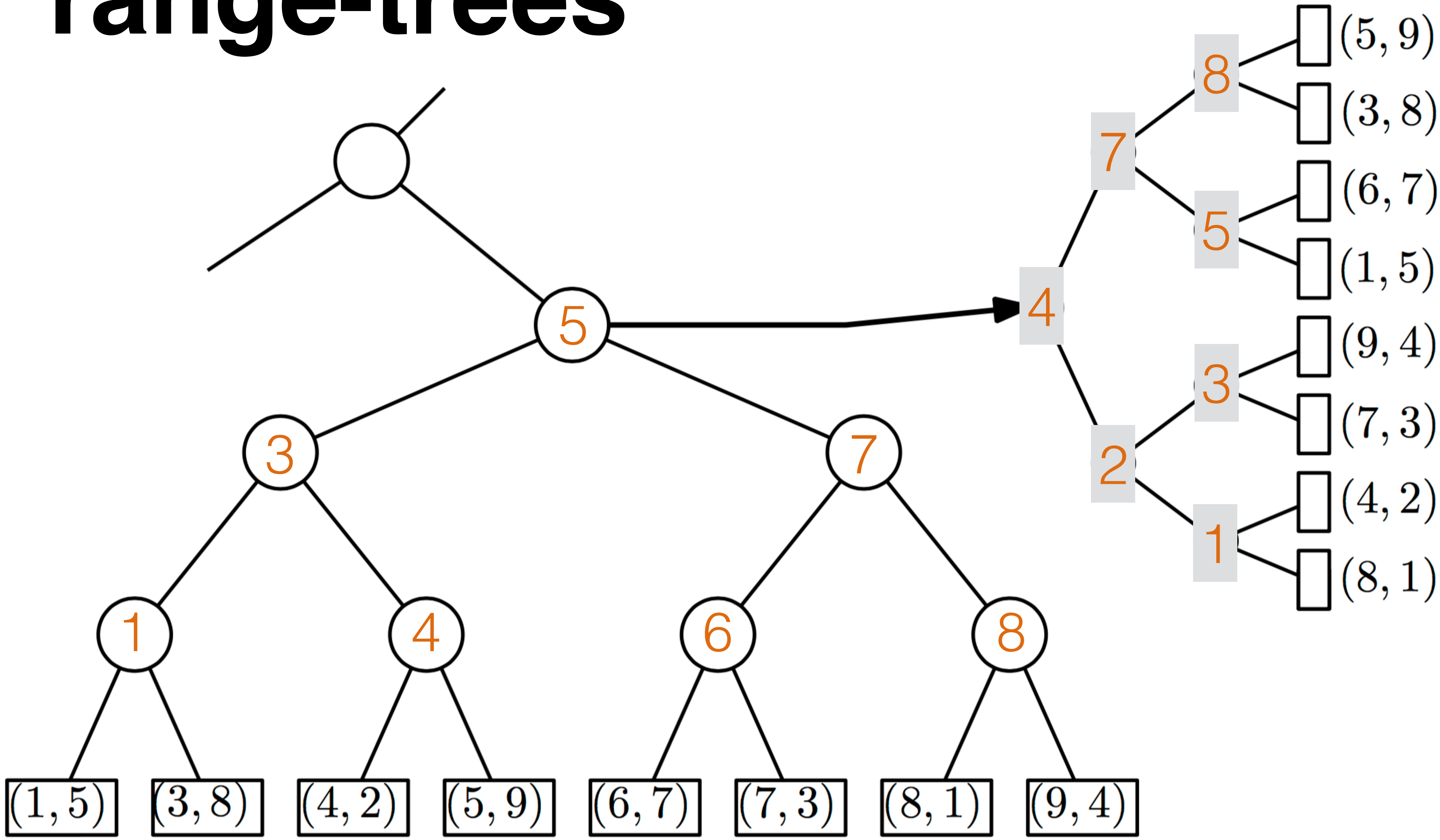
Next we'll see two structures that extend the BST

kd-trees

k-dimensional search trees



range-trees



screen shot from Mark van Kreveld, http://www.cs.uu.nl/docs/vakken/ga/slides5b.pdf)