

Computational Geometry

Winter term 2016/17

Orthogonal Range Queries

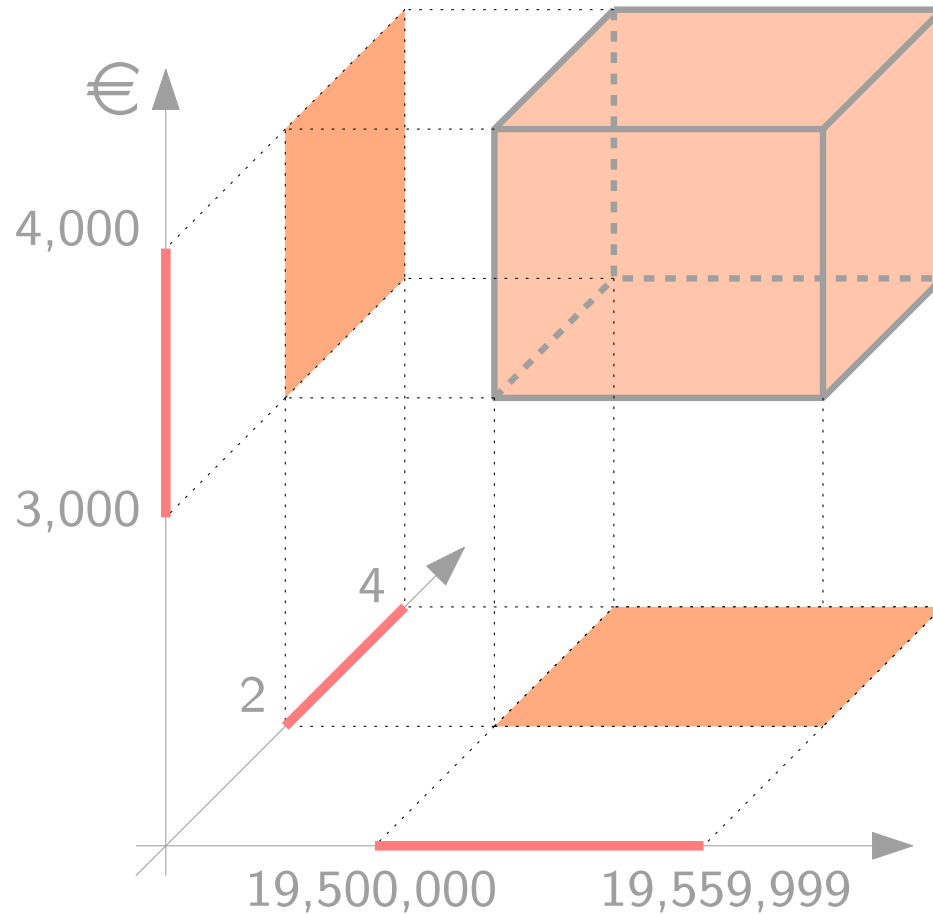
or

Fast Access to Data Bases

Lecture #5

Orthogonal Range Queries

Example: Personnel management in a company



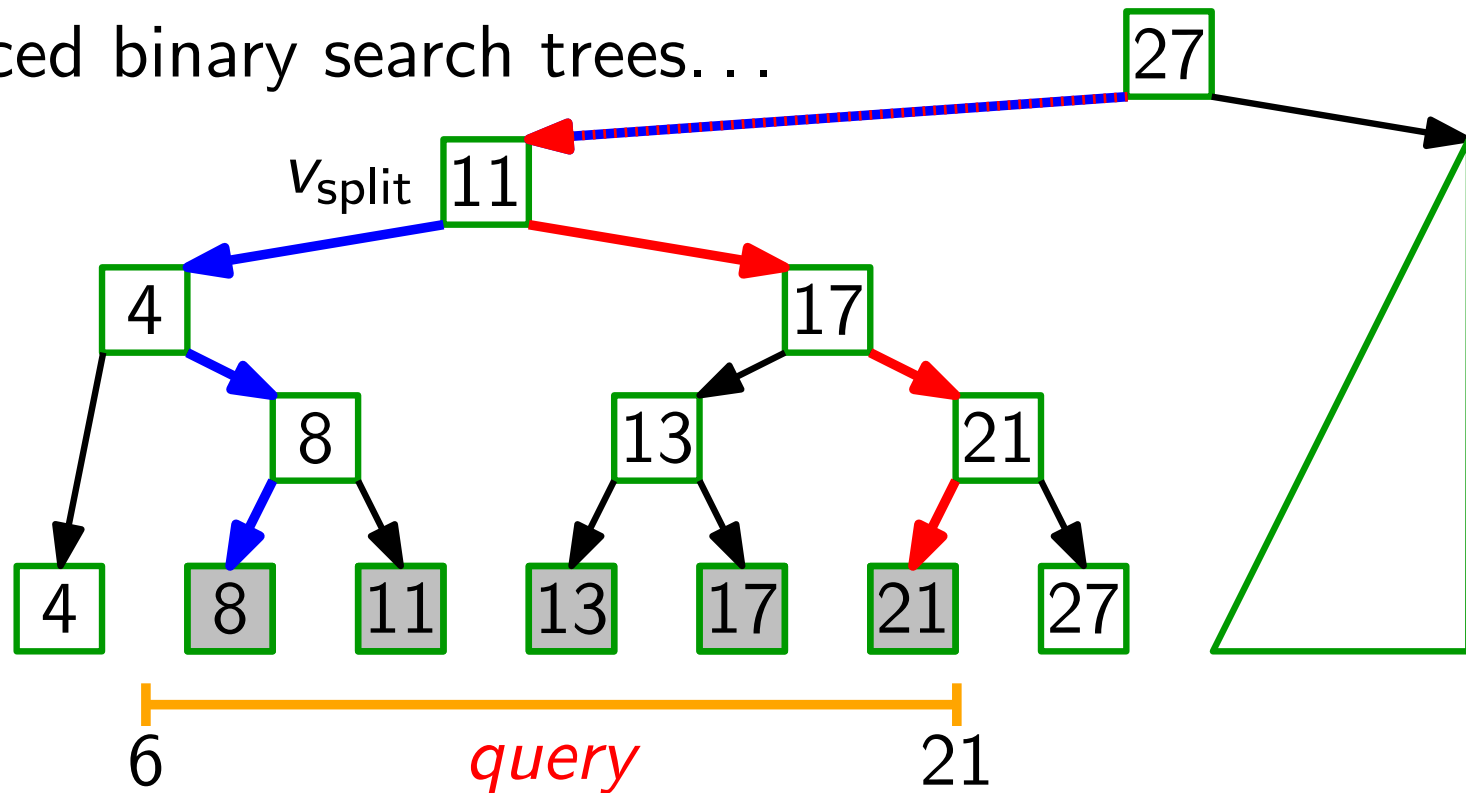
Typical queries for data bases!

1d Range Searching

Task: Preprocess a finite set $P \subset \mathbb{R}$ such that for any interval $[x, x']$ the set $P \cap [x, x']$ can be reported quickly. [2 min]

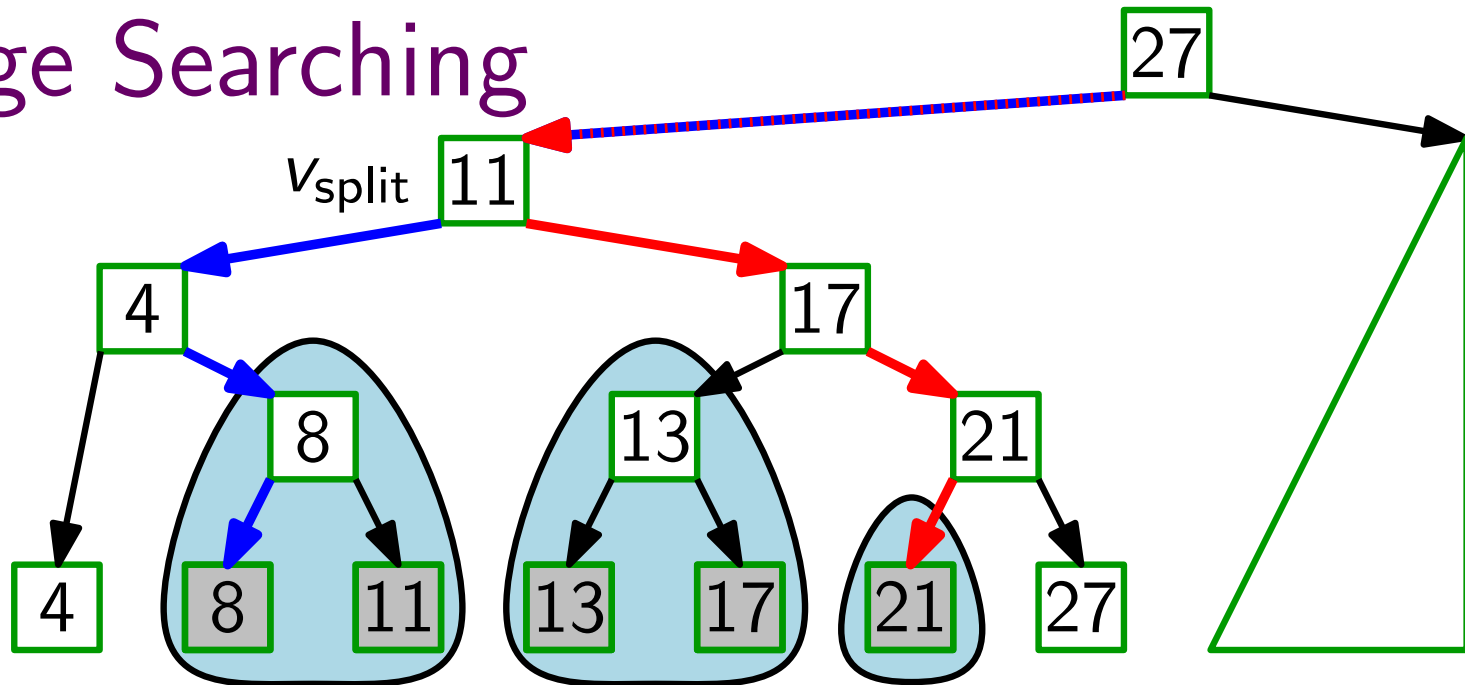
Solution: balanced binary search trees...

1. Search $x = 6$.
2. Search $x' = 21$.
3. Return all leaves 'inbetween'.



Small changes: – keys only in leaves
 – inner nodes store maxima of their left subtrees

1d Range Searching



Observe:

- The result of a query is the disjoint union of at most $2h$ *canonical subsets*, where
- $h \in O(\log n)$ is the tree height,
 - a canonical subset is an interval that contains all points stored in a subtree.

Theorem.

A set of n real numbers can be preprocessed in $O(n \log n)$ time and $O(n)$ space such that 1d range queries take $O(k + \log n)$ time, where $k = |\text{output}|$.

output sensitive!

Extensions to 2d

Think...

[3 min]

Task: Preprocess a finite set $P \subset \mathbb{R}^2$ such that for any range query $R = [x, x'] \times [y, y']$ the set $P \cap R$ can be reported quickly.

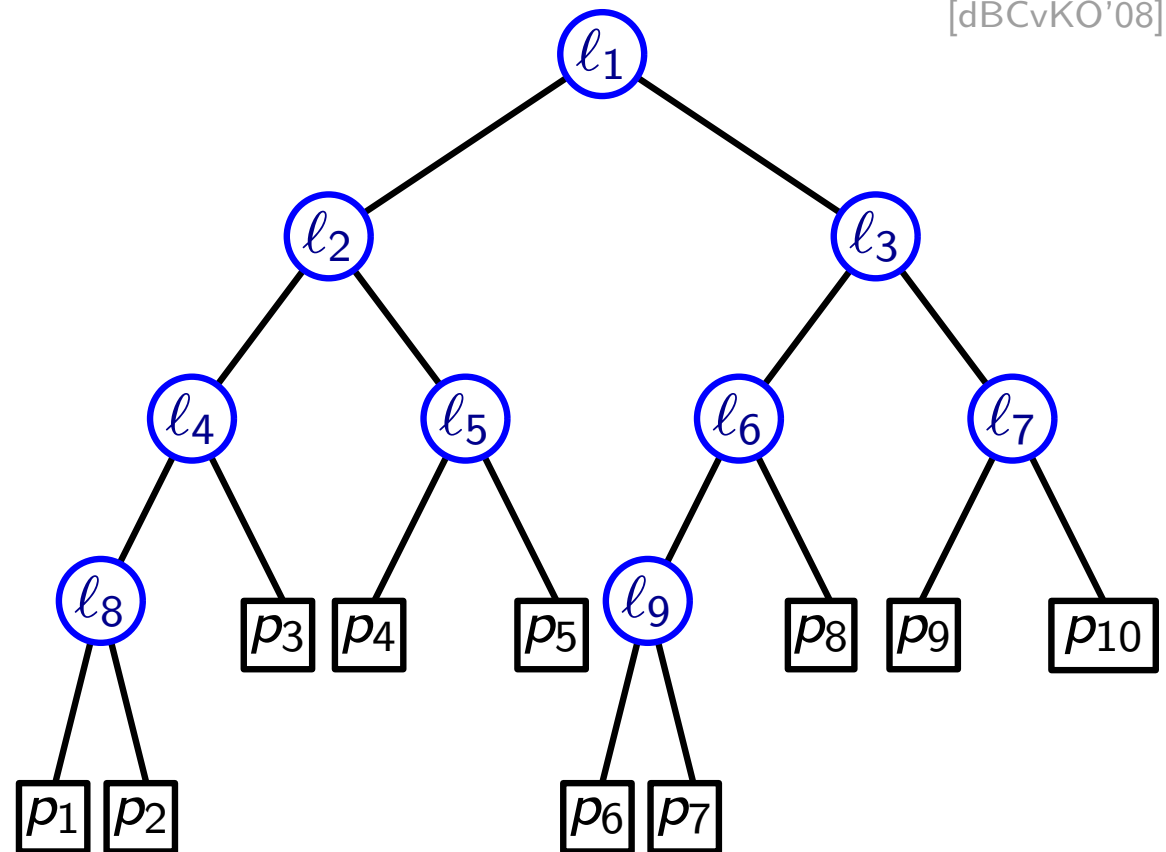
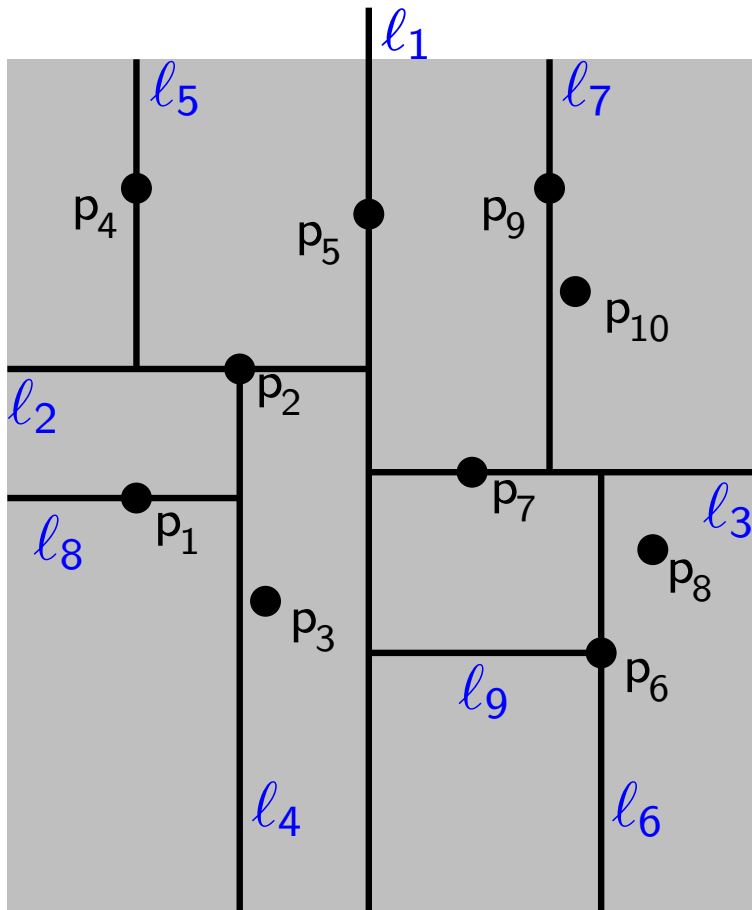
Solutions:

- *one* tree;
query path alternates between x - and y -coordinate } *kd-tree*
- first-level tree for x -coordinates;
many second-level trees for y -coordinate } *range tree*

Assume: *General position!*

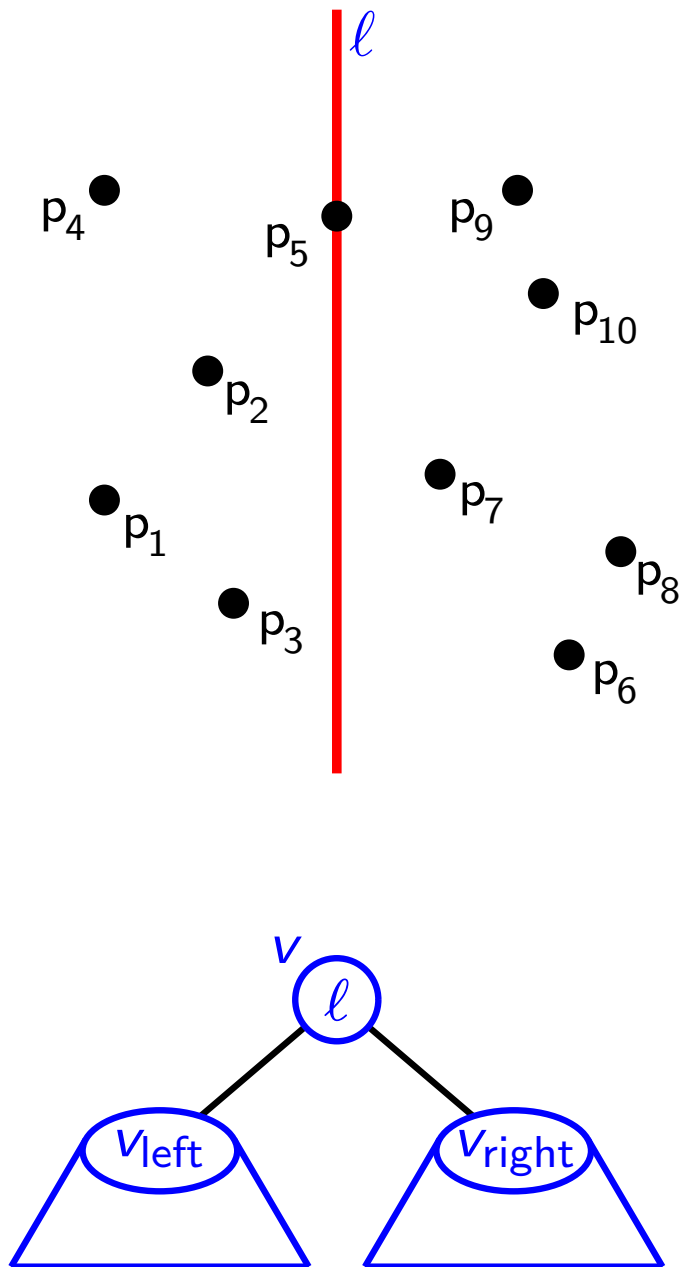
Here: no two points have the same x - or y -coordinate.

Kd-Trees: Example



- Split any region that contains more than one point.
- Horizontal split lines/segments belong to the region below.
Vertical left.

Kd-Trees: Construction



Pseudo-code:

```

BuildKdTree(points  $P$ , int  $depth$ )
  if  $|P| = 1$  then
    return (leaf storing the pt in  $P$ )
  else
    if  $depth$  is even then
      split  $P$  with the vertical line
         $\ell: x = x_{\text{median}(P)}$  into
         $P_1$  (pts left of or on  $\ell$ ) and
         $P_2 = P \setminus P_1$ 
    else
      split  $P$  horizontally...
     $v_{\text{left}} \leftarrow \text{BuildKdTree}(P_1, depth + 1)$ 
     $v_{\text{right}} \leftarrow \text{BuildKdTree}(P_2, depth + 1)$ 
    create a node  $v$  storing  $\ell$ 
    make  $v_{\text{left}}$  and  $v_{\text{right}}$  the children of  $v$ 
    return ( $v$ )
  
```

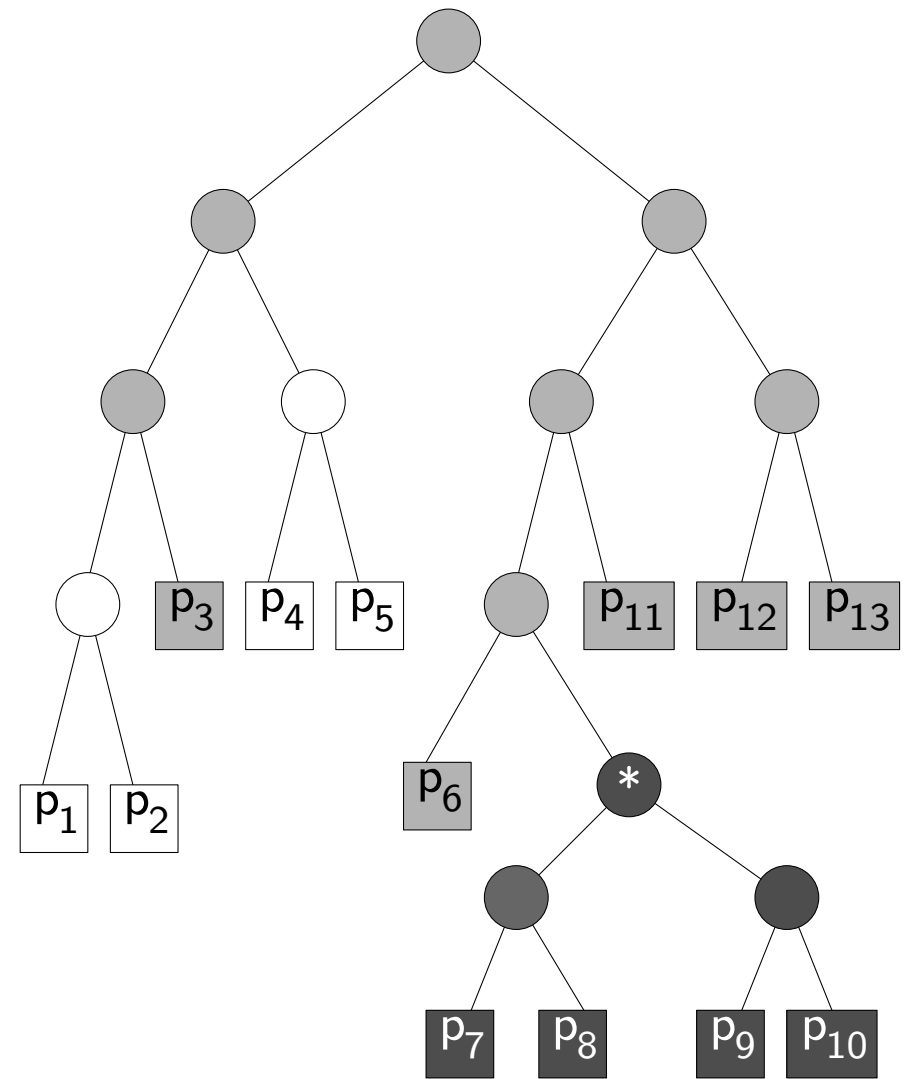
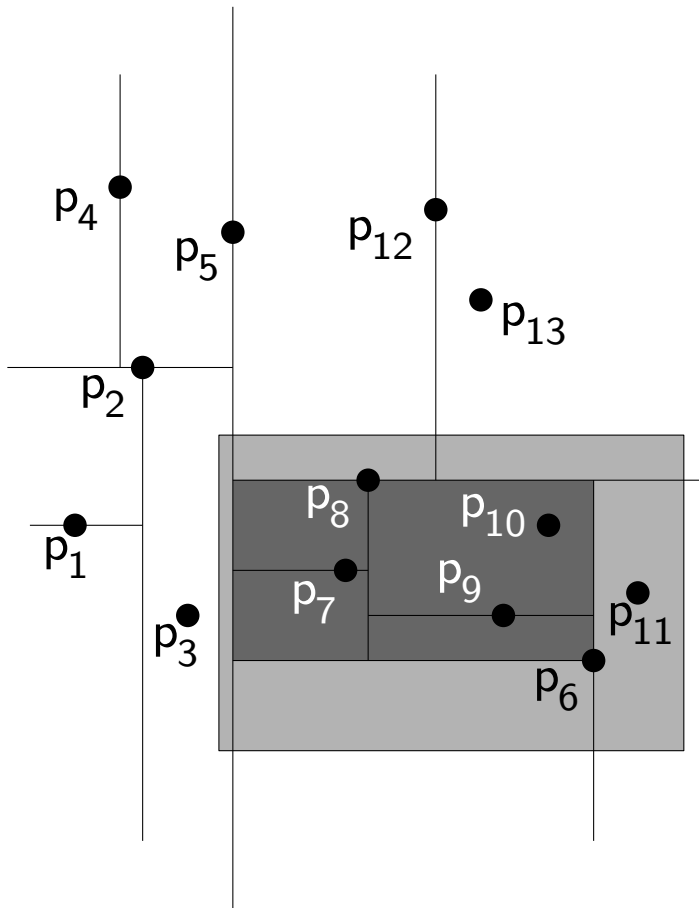
Kd-Trees: Analysis

Construction time?

$$T(n) = \left\{ \begin{array}{ll} O(1) & \text{if } n = 1 \\ O(n) + 2T(\lceil n/2 \rceil) & \text{else.} \end{array} \right\} \stackrel{\text{see Mergesort!}}{=} O(n \log n)$$

Lemma: A kd-tree for a set of n pts in the plane takes $O(n \log n)$ time to construct and uses $O(n)$ storage.

Kd-Trees: Querying



Lemma. Querying a kd-tree for n pts in the plane with an axis-parallel rectangle R takes $O(k + \sqrt{n})$ time, where $k = |\text{output}|$.

idea: $O(\sqrt{n})$ regions of the kd-tree intersect a vertical/horizontal line.

↓ in \mathbb{R}^d

$$O(k + n^{1-1/d})$$

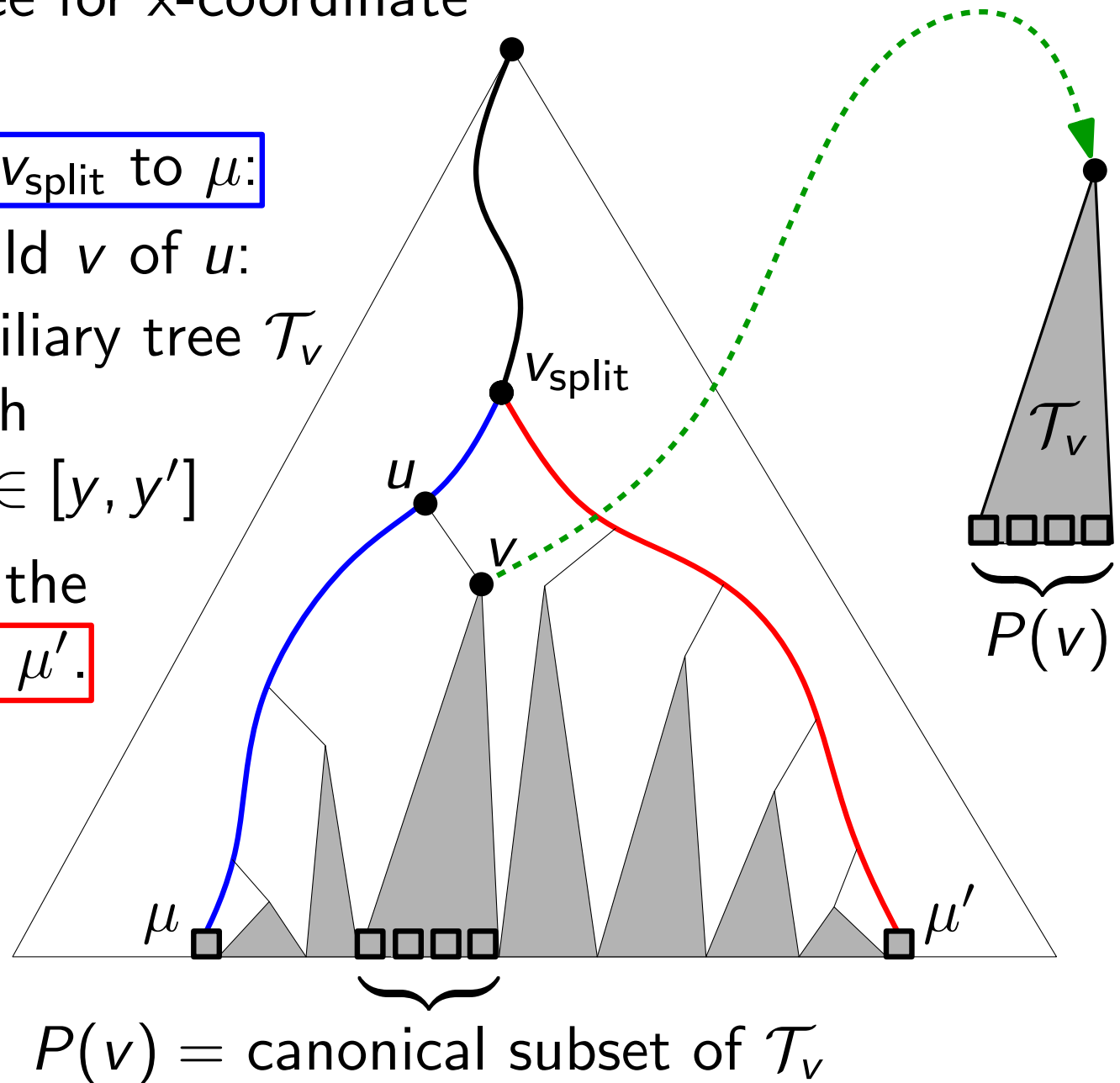
Range Trees: Query Algorithm

1. Search in main tree for x-coordinate
2. For each node u
on the path from v_{split} to μ :

For the right child v of u :

Search in auxiliary tree \mathcal{T}_v
for points with
y-coordinate $\in [y, y']$

3. Symmetrically for the
path from v_{split} to μ' .



Range Trees: Construction

Build2DRangeTree(point[] P)

construct 2nd-level tree \mathcal{T}_P on P (y -order)

if $P = \{p\}$ then

 | create leaf v :

else

$x_{\text{mid}} =$ median x -coordinate of P

$P_{\text{left}} =$ pts in P with x -coordinate $\leq x_{\text{mid}}$

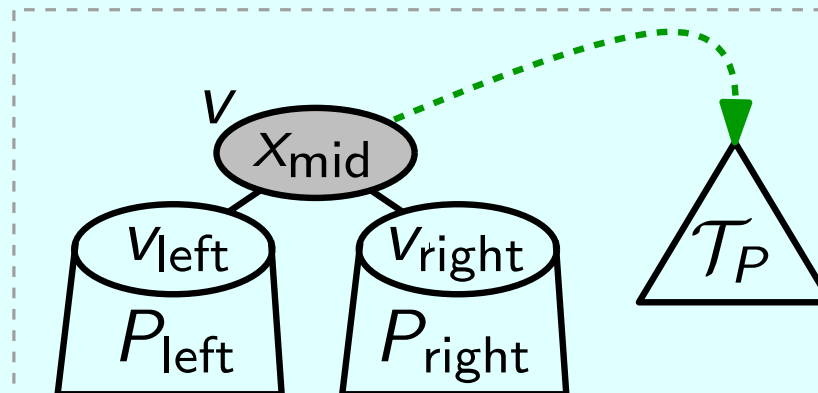
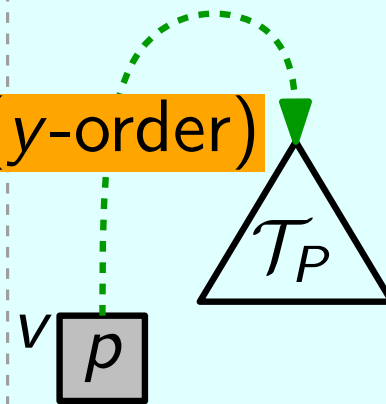
$P_{\text{right}} =$ pts in P with x -coordinate $> x_{\text{mid}}$

$v_{\text{left}} =$ Build2DRangeTree(P_{left})

$v_{\text{right}} =$ Build2DRangeTree(P_{right})

create node v :

return v



Running time?

$O(n \log n)$:- (

Better:

Pre-sort once,
then build tree
bottom-up
in linear time.



Total
construction
time $O(n \log n)$

Range Trees: Space Consumption

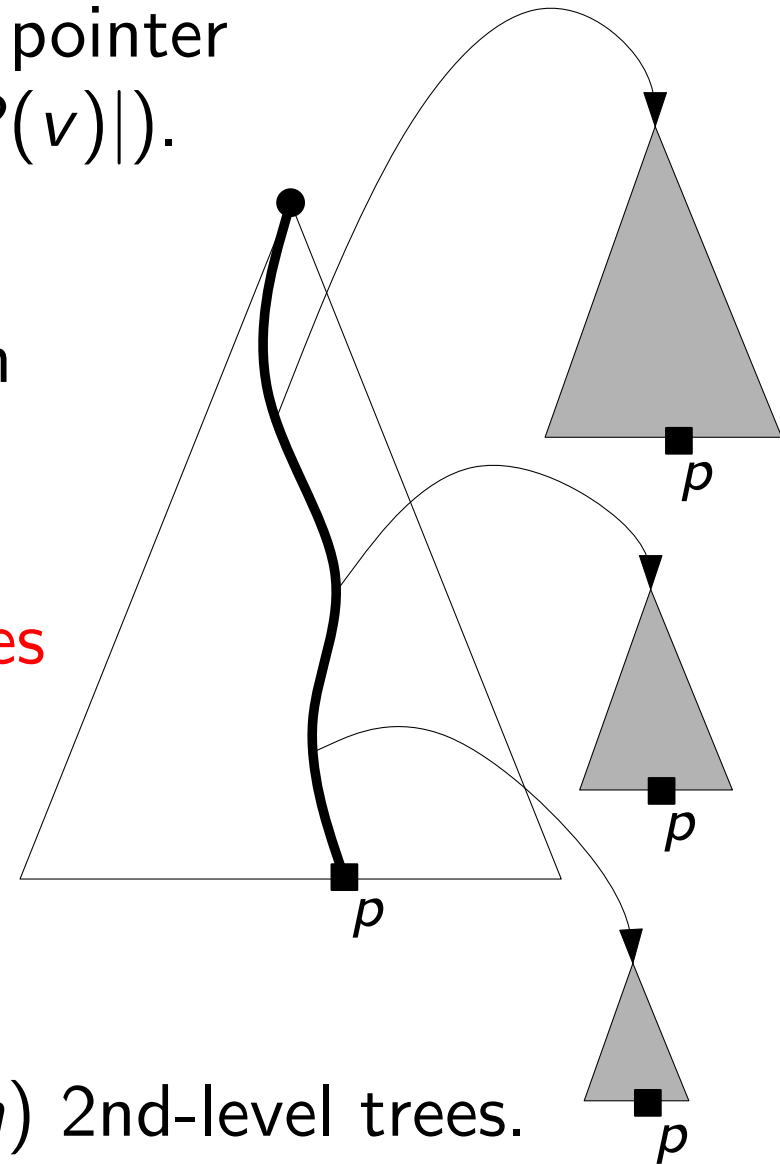
Each node v of the 1st-level tree has a pointer to a 2nd-level tree \mathcal{T}_v with $|\mathcal{T}_v| = \Theta(|P(v)|)$.

Q: What's the *total* space consumption of all 2nd-level trees?

What's your guess:

- $\Theta(n^2)$,
- $\Theta(n \log n)$,
- $\Theta(n \log^2 n)$, or
- $\Theta(n)$?

How many trees will contain a given point?



A: Each $p \in P$ is stored in $h = \Theta(\log n)$ 2nd-level trees.

$\Rightarrow \Theta(n \log n)$ space

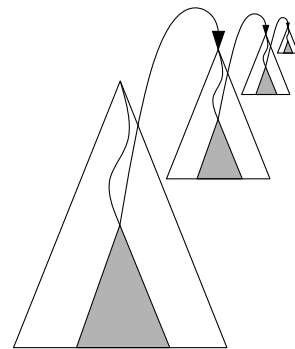
Range Trees: Query time

$$\begin{aligned}
 T(n, k) &= \sum_{u \in \text{paths to } x \text{ and } x'} O(k_u + \log n) \\
 &= O(\sum_u k_u) + O(\sum_u \log n) \\
 &= O(k) + 2h \cdot O(\log n) \\
 &= O(k + \log^2 n)
 \end{aligned}$$

$\mathbb{R}^d?$

$O(n \log^{d-1} n)$ storage and construction time

$O(k + \log^d n)$ query time



See Chapter 5.4 in the book

Computational Geometry: Algorithms and Applications

by de Berg et al., Springer-Verlag, 3rd edition 2008.

Comparison

	kd-tree	range tree
construction time	$O(n \log n)$	$O(n \log n)$
storage	$O(n)$	$O(n \log n)$
query time	$O(k + \sqrt{n})$	$O(k + \log^2 n)$

Note: *trade-off* between space and query time

General Sets of Points

Idea: use *composite numbers* $(a|b)$ with lex order

$$p = (x, y) \longrightarrow \hat{p} = ((x|y), (y|x)) \longrightarrow \text{unique coordin.}$$

$$\text{range } R = [x, x'] \times [y, y']$$



$$\hat{R} = [(x| - \infty), (x'| + \infty)] \times [(y| - \infty), (y'| + \infty)]$$

Show: $p \in R \Leftrightarrow \hat{p} \in \hat{R}$

This removes our assumption about the input points being in general position.

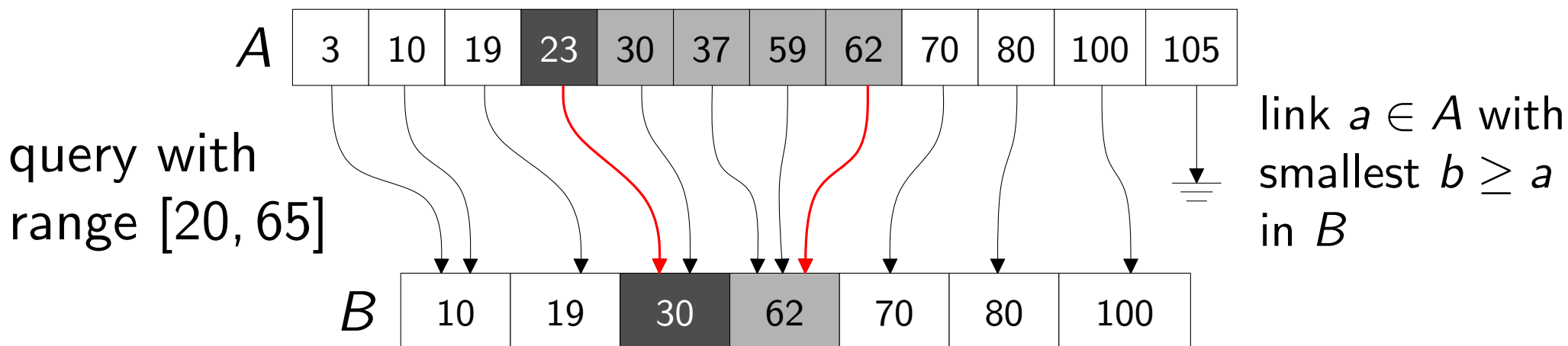
We can use kd-trees and range trees for *any* set of points; no matter how many points have the same x- or y-coordinates.

Fractional Cascading

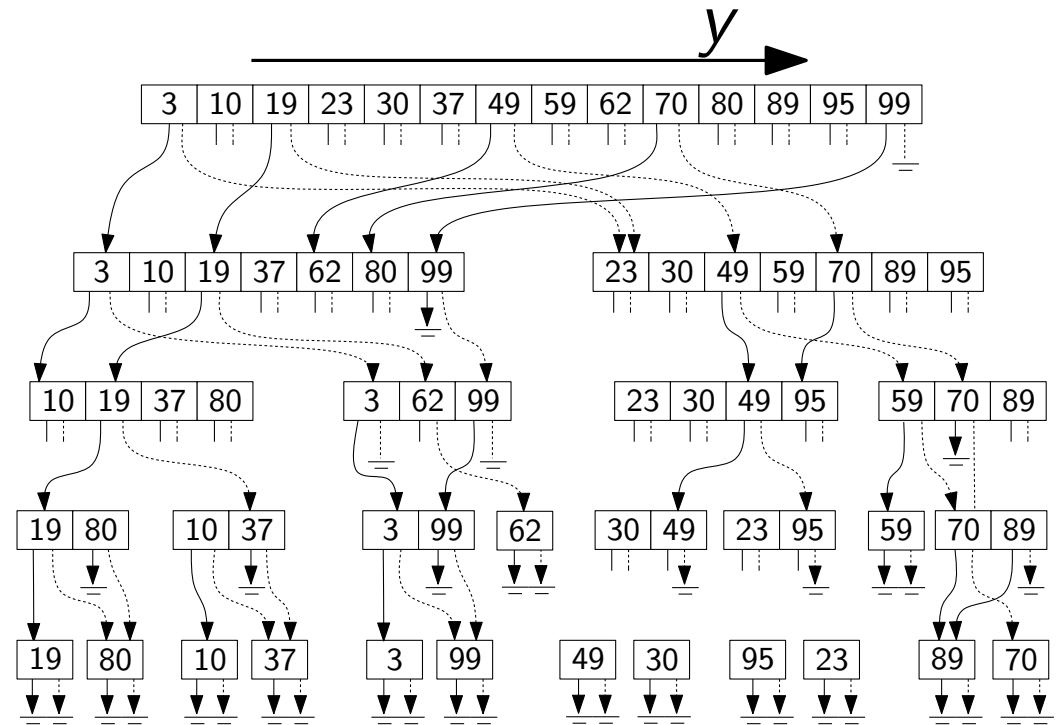
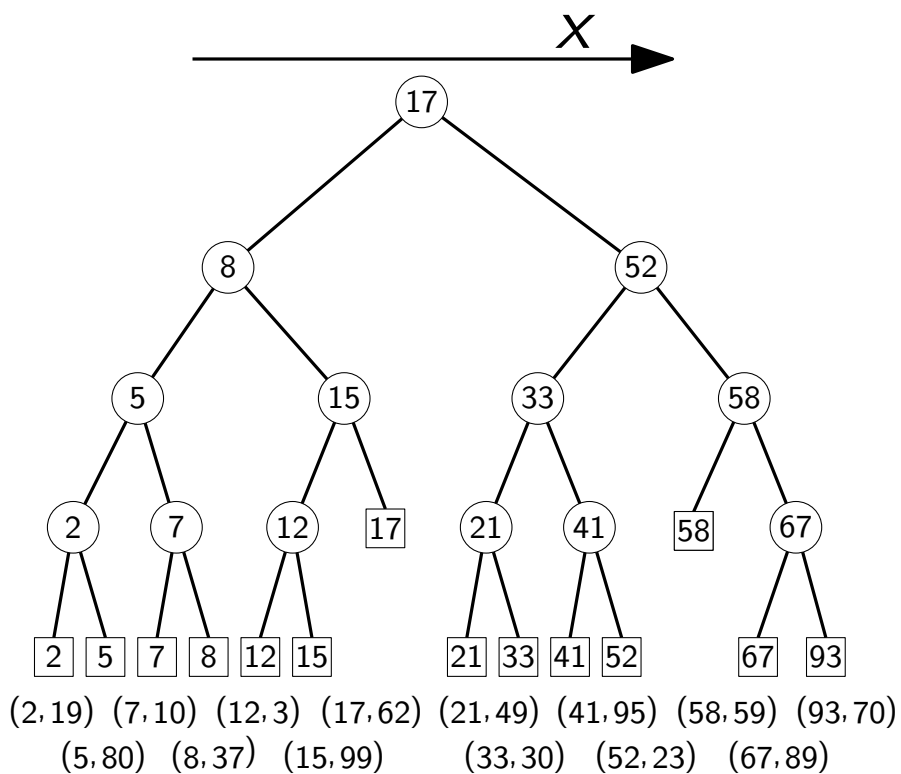
Task 1: Given sets $B \subset A \subset \mathbb{N}$ stored in sorted order in arrays $A[1..n]$ and $B[1..m]$, support 1d range queries in the multiset $A \cup B$ in $k + 1 \cdot \log n$ time!

We allow $n \log m$ bits extra space.

Task 2: Assuming that task 1 can be solved, speed up 2d range queries: $O(k + \log^2 n) \rightarrow O(k + \log n)$ time!



Layered Range Trees



[dBCvKO'08]

Theorem: Let $d \geq 2$ and let P be a set of n pts in \mathbb{R}^d . Given $O(n \log^{d-1} n)$ preprocessing time & storage, d -dim range queries on P can be answered in $O(k + \log^{d-1} n)$ time.