Orthogonal Range Searching

slides by Andy Mirzaian

(a subset of the original slides are used here)

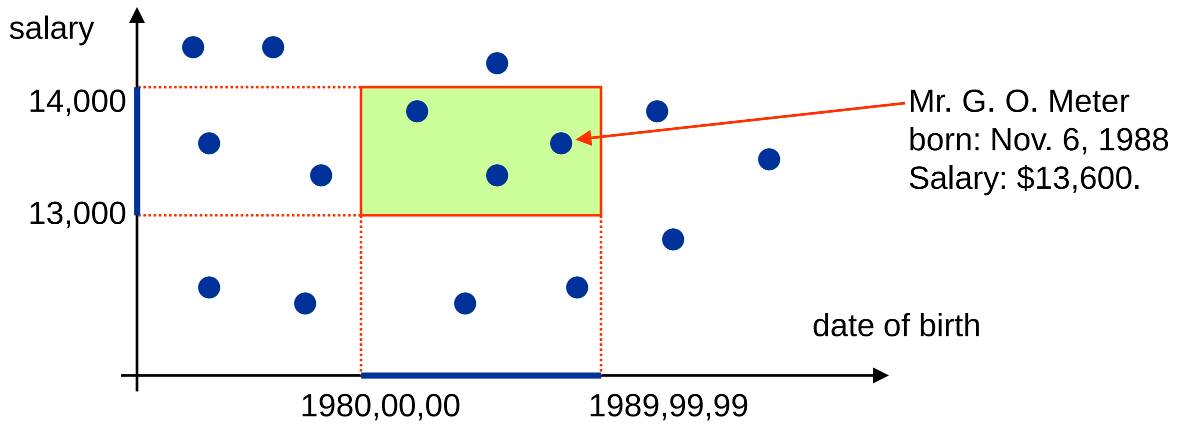
References:

- [M. de Berge et al] chapter 5

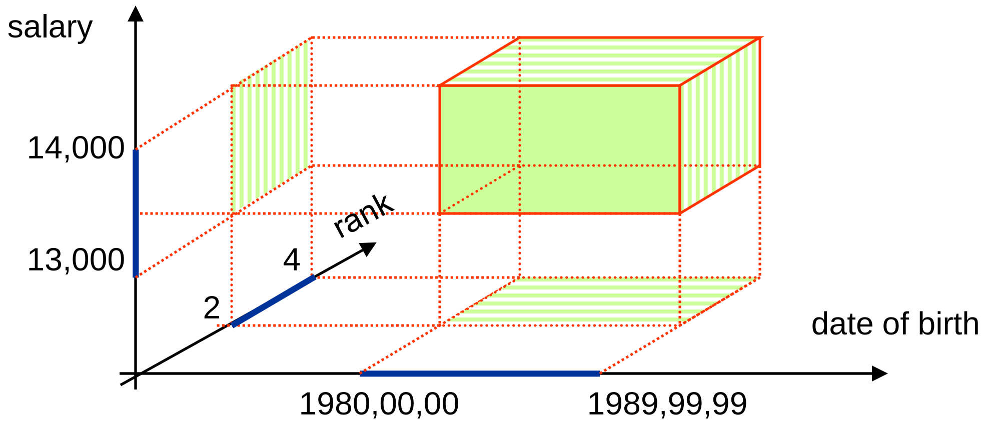
Applications:

- Spatial Databases
- GIS, Graphics: crop-&-zoom, windowing

Orthogonal Range Search: Database Query

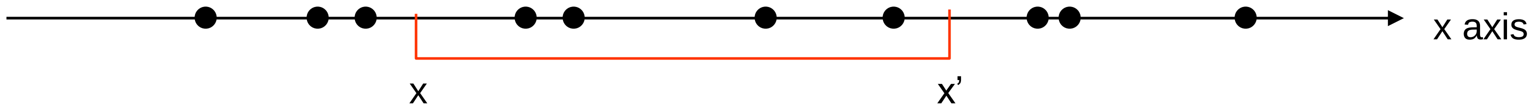


2D Query Rectangle [1980,00,00 : 1989,99,99] × [13,000 : 14,000]



3D Query Orthogonal Range [1980,00,00 : 1989,99,99] × [13,000 : 14,000] × [2 : 4]

1D-Tree: 1-Dimensional Range Searching



Static: Binary Search in a sorted array.

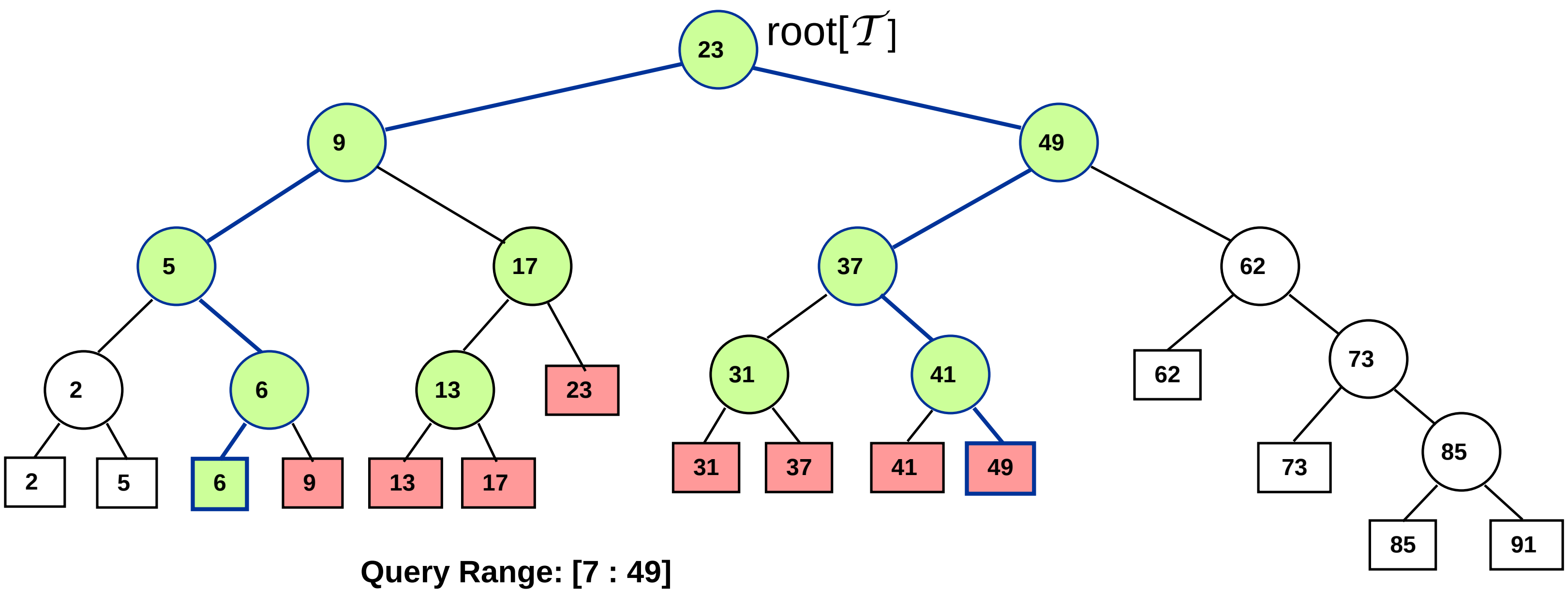
Dynamic: Store data points in some balanced Binary Search Tree $\mathcal{T}$.

Let the data points be $P = \{ p_1, p_2, \ldots, p_n \} \subseteq \Re$.

$\mathcal{T}$ is a balanced BST where the data appear at its leaves sorted left to right.

The internal nodes are used to split left & right subtrees.

Assume x(v) = max x(L), where L is any leaf in the left subtree of internal node v.



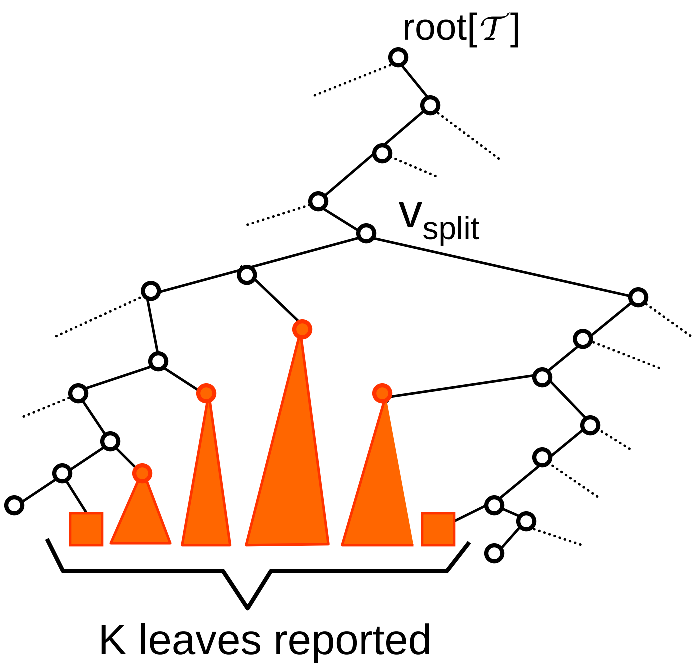
Query Range [x : x']: Call 1DRangeQuery(root[$\mathcal{T}$],x,x')

ALGORITHM 1DRangeQuery (v, x, x')
if v is a leaf **then** **if** $x \le x(v) \le x'$ **then** report data stored at v
else do
  if $x \le x(v)$ **then** 1DRangeQuery (leftchild(v) , x, x')
  if $x(v) < x'$ **then** 1DRangeQuery (rightchild(v), x, x')
od
end

<u>Complexities</u>:

Query Time	O(K + log n)	$\mathcal{T}$,[x,x'] $\rightarrow$ output
Construction Time	O(n log n)	P $\rightarrow$ $\mathcal{T}$
Space	O(n)	store $\mathcal{T}$

[These are optimal]

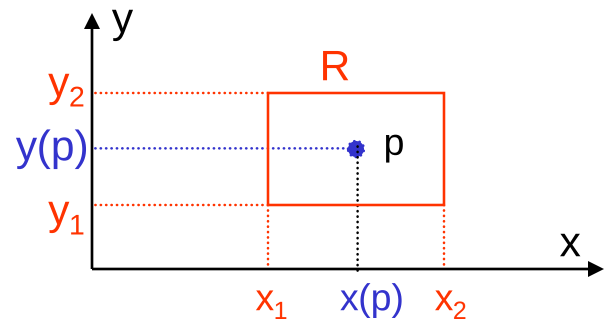


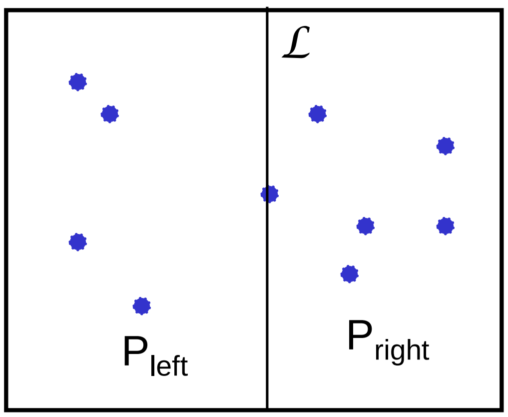
2D-Tree

Consider dimension d=2:

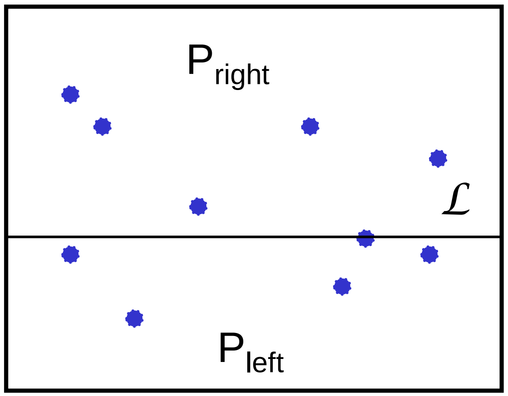
point $p = (x(p) , y(p))$, range $R = [x_1 : x_2] \times [y_1 : y_2]$

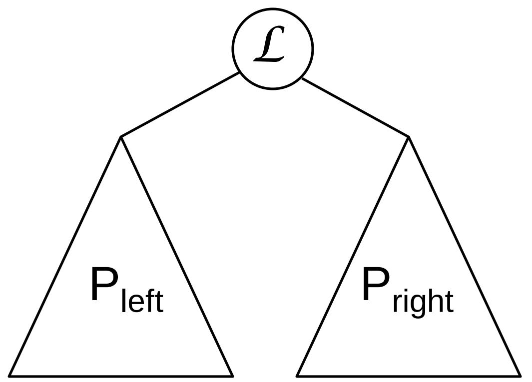
$p \in R \iff x(p) \in [x_1 : x_2]$ and $y(p) \in [y_1 : y_2]$.





OR





2D-tree

$\mathcal{L}$ = vertical/horizontal median split.

Alternate between vertical & horizontal splitting at even and odd depths.

(Assume: no 2 points have equal x or y coordinates.)

Constructing 2D-Tree

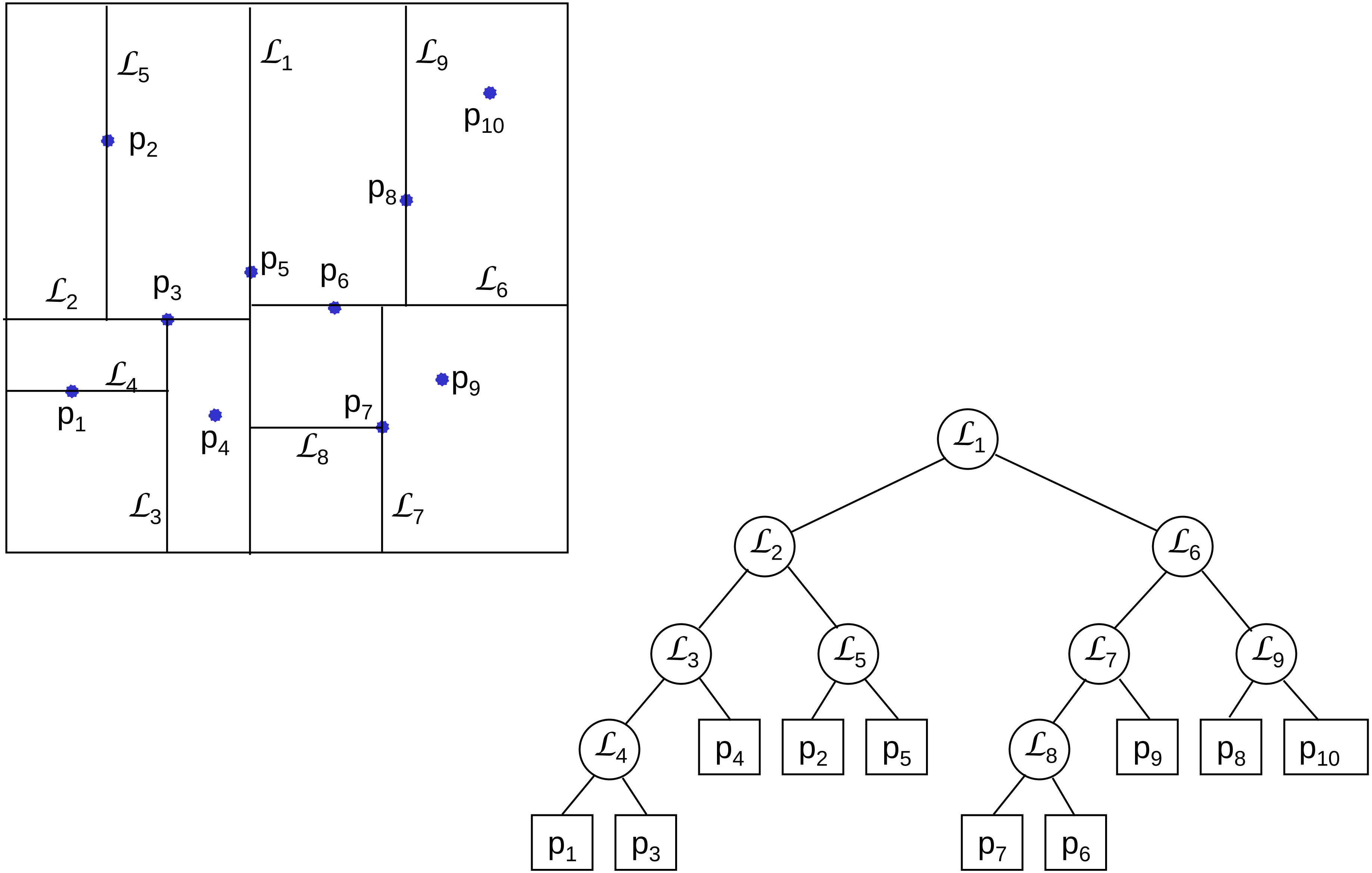
Input: $P = \{ p_1, p_2, \ldots, p_n \} \subseteq \Re^2$ off-line.
Output: 2D-tree storing P.

Step 1: Pre-sort P on x & on y, i.e., 2 sorted lists Û = (Xsorted(P), Ysorted(P)).
Step 2: root[$\mathcal{T}$] $\leftarrow$ Build2DTree (Û , 0)
end

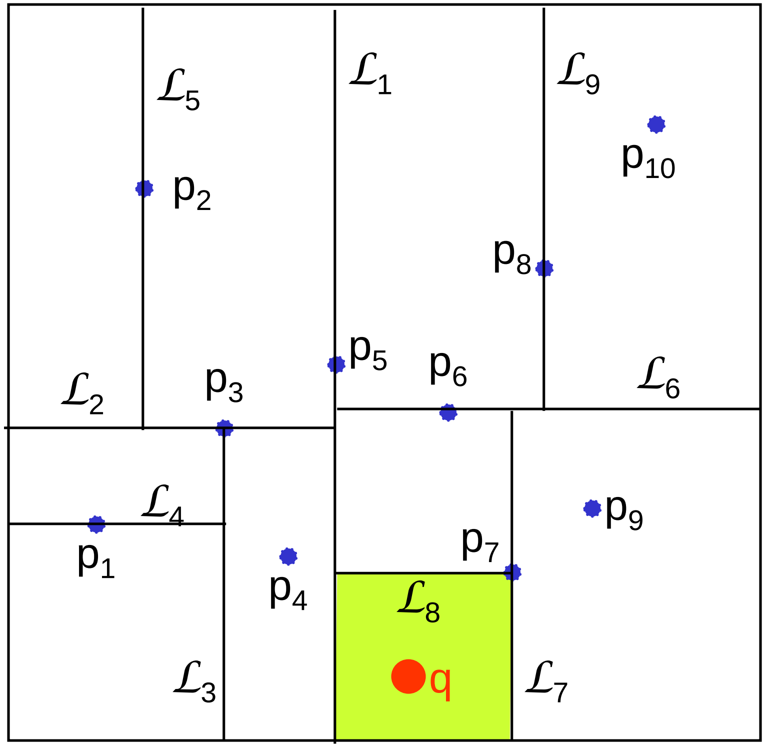
Procedure Build2DTree (Û , depth)
if Û contains one point **then return** a leaf storing this point
else do
if depth is even
then x-median split Û, i.e., split data points in half by a vertical line $\mathcal{L}$ through x-median of Û and reconfigure $\hat{U}_{left}$ and $\hat{U}_{right}$.
else y-median split Û, … by a horizontal line $\mathcal{L}$, and reconfigure $\hat{U}_{left}$ and $\hat{U}_{right}$.
v $\leftarrow$ a newly created node storing line $\mathcal{L}$
leftchild(v) $\leftarrow$ Build2DTree ($\hat{U}_{left}$, 1+depth)
rightchild(v) $\leftarrow$ Build2DTree ($\hat{U}_{right}$, 1+depth)
return v
end

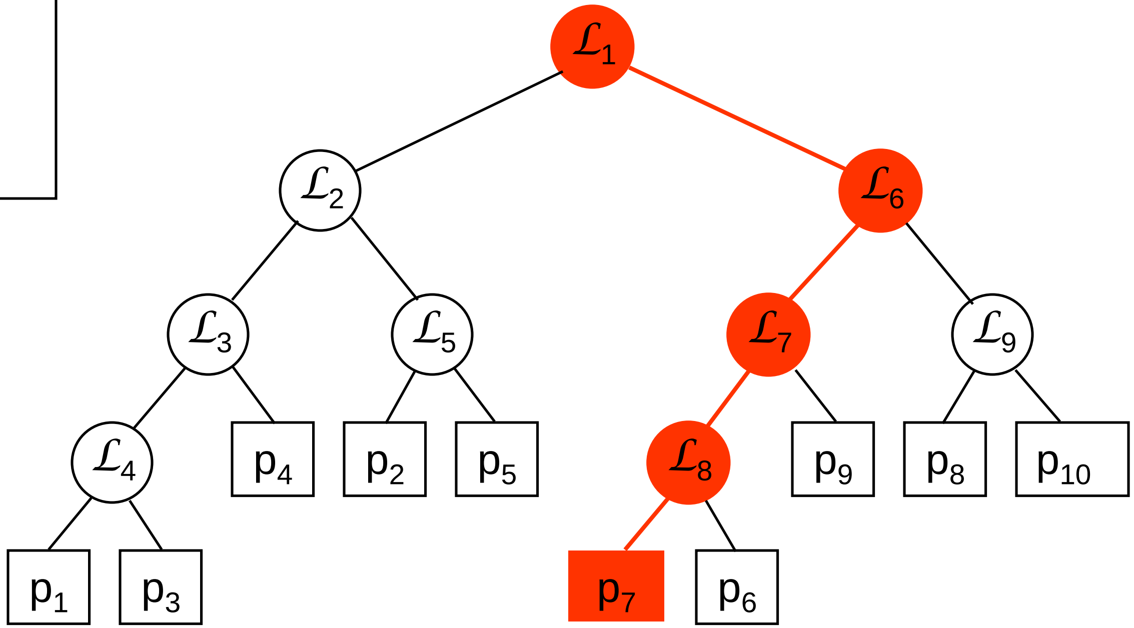
T(n) = 2 T(n/2) + O(n) = O(n log n) time.

2D-Tree Example



Query Point Search in 2D-Tree





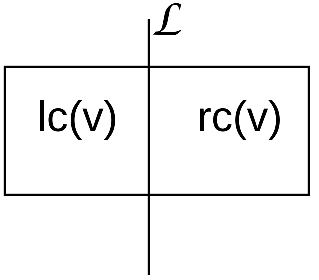
2D-Tree node regions

region(v) = rectangular region (possibly unbounded) covered by the subtree rooted at v.

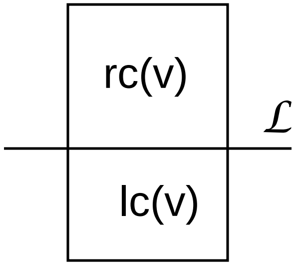
region (root[$\mathcal{T}$]) = $(-\infty : +\infty) \times (-\infty : +\infty)$

Suppose region(v) = $\langle x_1 : x_2 \rangle \times \langle y_1 : y_2 \rangle$
what are region(leftchild(v)) and region(rightchild(v))?

With x-split:
region(lc(v)) = $\langle x_1 : x(\mathcal{L})] \times \langle y_1 : y_2 \rangle$
region(rc(v)) = $(x(\mathcal{L}) : x_2 \rangle \times \langle y_1 : y_2 \rangle$



With y-split:
region(lc(v)) = $\langle x_1 : x_2 \rangle \times \langle y_1 : y(\mathcal{L})]$
region(rc(v)) = $\langle x_1 : x_2 \rangle \times (y(\mathcal{L}) : y_2 \rangle$



2D-Tree Range Search

For range $R = [x_1 : x_2] \times [y_1 : y_2]$ call **Search2DTree (root[$\mathcal{T}$] , R)**

ALGORITHM **Search2DTree (v , R)**

1. **if** v is a leaf **then if** $p(v) \in R$ **then** report p(v)
2. **else if** $region(lc(v)) \subseteq R$
3. **then** **ReportSubtree (lc(v))**
4. **else if** $region(lc(v)) \cap R \neq \varnothing$
5. **then** **Search2DTree (lc(v) , R)**

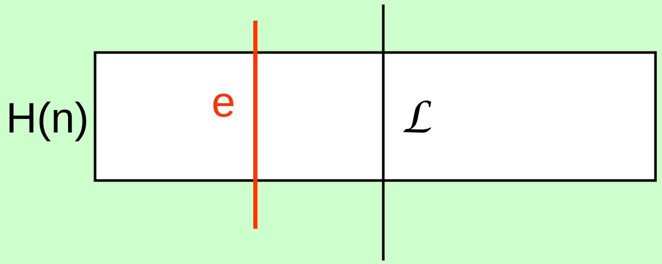
6. **if** $region(rc(v)) \subseteq R$
7. **then** **ReportSubtree (rc(v))**
8. **else if** $region(rc(v)) \cap R \neq \varnothing$
9. **then** **Search2DTree (rc(v) , R)**

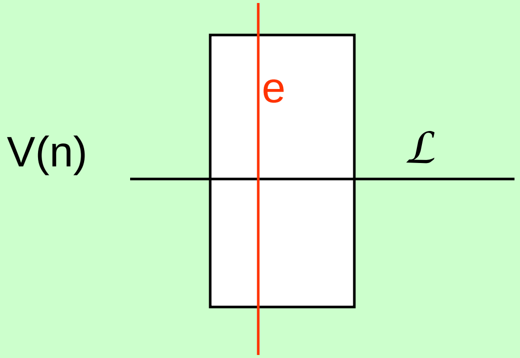
end

- **region(v)** can either be passed as input parameter, or explicitly stored at node v, $\forall v \in \mathcal{T}$.
- **ReportSubtree(v)** is a simple linear-time in-order traversal that reports every leaf descendent of node v.

Running Time of Search2DTree

- K = # of points reported.
- Lines 3 & 7 take O(K) time over all recursive calls.
- Total # nodes visited (reported or not) is proportional to # times conditions of lines 4 & 8 are true.
- region(v)∩R≠∅ & region(v) ⊄ R ⇔ a bounding edge e of R intersects region(v).
- R has ≤ 4 bounding edges. Let e (assume vertical) be one of them.
- Define H(n) (resp. V(n)) = worst-case number of nodes v that intersect e for a 2D-tree of n leaves, assuming root corresponds to an x-split (resp. y-split).





$$\begin{cases} H(n) = V(n/2) + 1 \\ V(n) = 2H(n/2) + 1 \\ (H(1) = V(1) = 1) \end{cases} \Rightarrow \begin{cases} H(n) = 2H(n/4) + 2 \\ V(n) = 2V(n/4) + 3 \end{cases} \Rightarrow \begin{cases} H(n) = 3\sqrt{n} - 2 \\ V(n) = 4\sqrt{n} - 3 \end{cases}$$

$$\Rightarrow \text{ Running Time } = O(K + \sqrt{n}).$$

dD-Tree Complexities

2D-Tree

- ❑ Query Time : $O(K + \sqrt{n})$ worst-case, $O(K + \log n)$ average
- ❑ Construction Time : $O(n \log n)$
- ❑ Storage Space: $O(n)$

dD-Tree d-dimensions

Use round-robin splitting at successive levels on the d dimensions $x_1, x_2, \ldots, x_d$.

- ❑ Query Time: $O(dK + d\, n^{1-1/d})$
- ❑ Construction Time: $O(d\, n \log n)$
- ❑ Space: $O(dn)$

How can we improve the query time?

Range Trees

2D-Tree

- Query Time: $O(K + \sqrt{n})$
- Construction Time: O(n log n)
- Space: O(n)

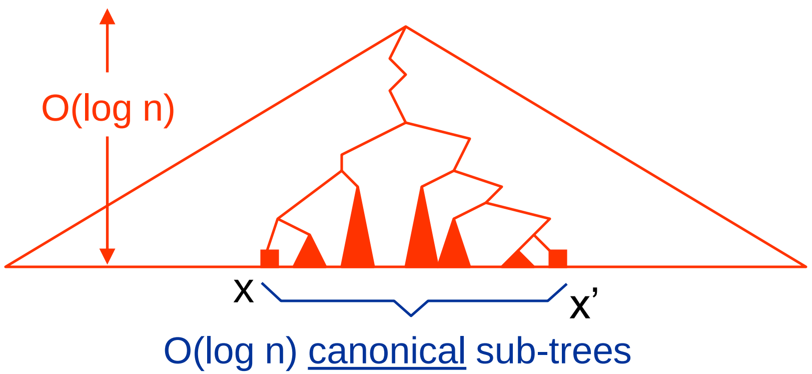
→

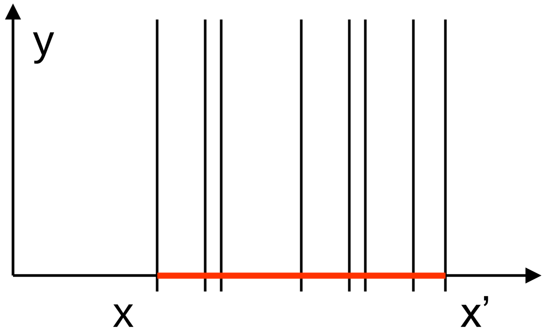
2D Range Tree

- Query Time: $O(K + \log^2 n)$

  O(K + log n) by Fractional Cascading
- Construction Time: O(n log n)
- Space: O(n log n)

Range R = [x : x’] × [y : y’]
1D Range Tree on x-coordinates:

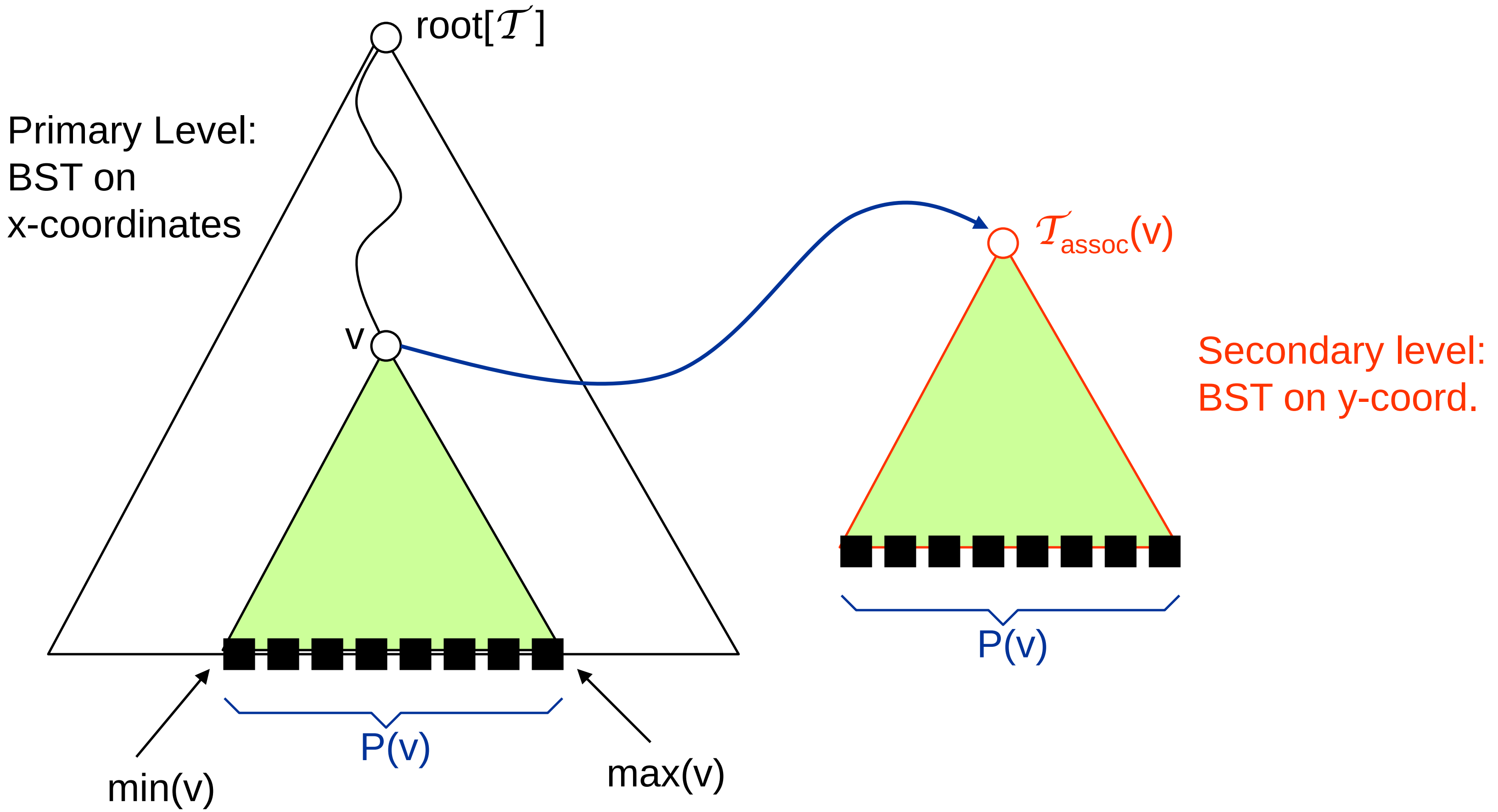




Each x-range [x : x’] can be expressed as the disjoint union of O(log n) canonical x-ranges.

Range Trees

2-level data structure:



Range Tree Construction

ALGORITHM Build 2D Range Tree (P)

Input: $P = \{ p_1, p_2, \dots, p_n \} \subseteq \Re^2$, $P = (P_x, P_y)$
represented by pre-sorted list on x (named P_x) and on y (named P_y).

Output: pointer to the root of 2D range tree for P.

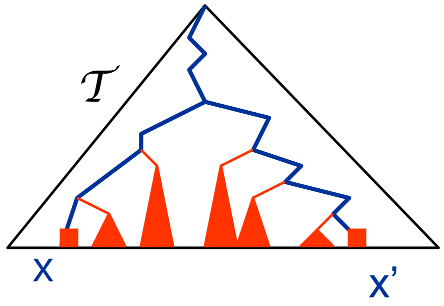
Construct $\mathcal{T}_{assoc}$, bottom up, based on P_y,
but store in each leaf the points, not just their y-coordinates.

if $|P| > 1$
then do
$P_{left} \leftarrow \{ p \in P \mid p_x \leq x_{med}$ of $P \}$ (* both lists P_x and P_y should split *)
$P_{right} \leftarrow \{ p \in P \mid p_x > x_{med}$ of $P \}$
$lc(v) \leftarrow$ Build 2D Range Tree (P_{left})
$rc(v) \leftarrow$ Build 2D Range Tree (P_{right})
od
$min(v) \leftarrow min(P_x)$; $max(v) \leftarrow max(P_x)$
$\mathcal{T}_{assoc}(v) \leftarrow \mathcal{T}_{assoc}$
return v
end

$T(n) = 2T(n/2) + O(n) = O(n \log n)$ time.
This includes time for pre-sorting.

2D Range Query

```
ALGORITHM   2DRangeQuery ( v, [x : x’] × [y : y’] )
1.    if   x ≤  min(v) & max(v) ≤ x’
2.         then   1DRangeQuery (𝒯assoc(v) , [y : y’] )
3.         else if  v is not a leaf  do
4.                if   x ≤  max(lc(v))
5.                     then  2DRangeQuery ( lc(v), [x : x’] × [y : y’] )
6.                if   min(rc(v)) ≤ x’
7.                     then  2DRangeQuery ( rc(v), [x : x’] × [y : y’] )
8.          od
end
```



- Line 2 called at roots of red canonical sub-trees, a total of O(log n) times.
  Each call takes $O(K_v + \log |\mathcal{T}_{assoc}(v)|) = O(K_v + \log n)$ time.
- Lines 5 & 7 called at blue shoulder paths. Total cost O(log n).
- Total Query Time = $O(\log n + \sum_v (K_v + \log n)) = O(\sum_v K_v + \log^2 n) = O(K + \log^2 n)$.

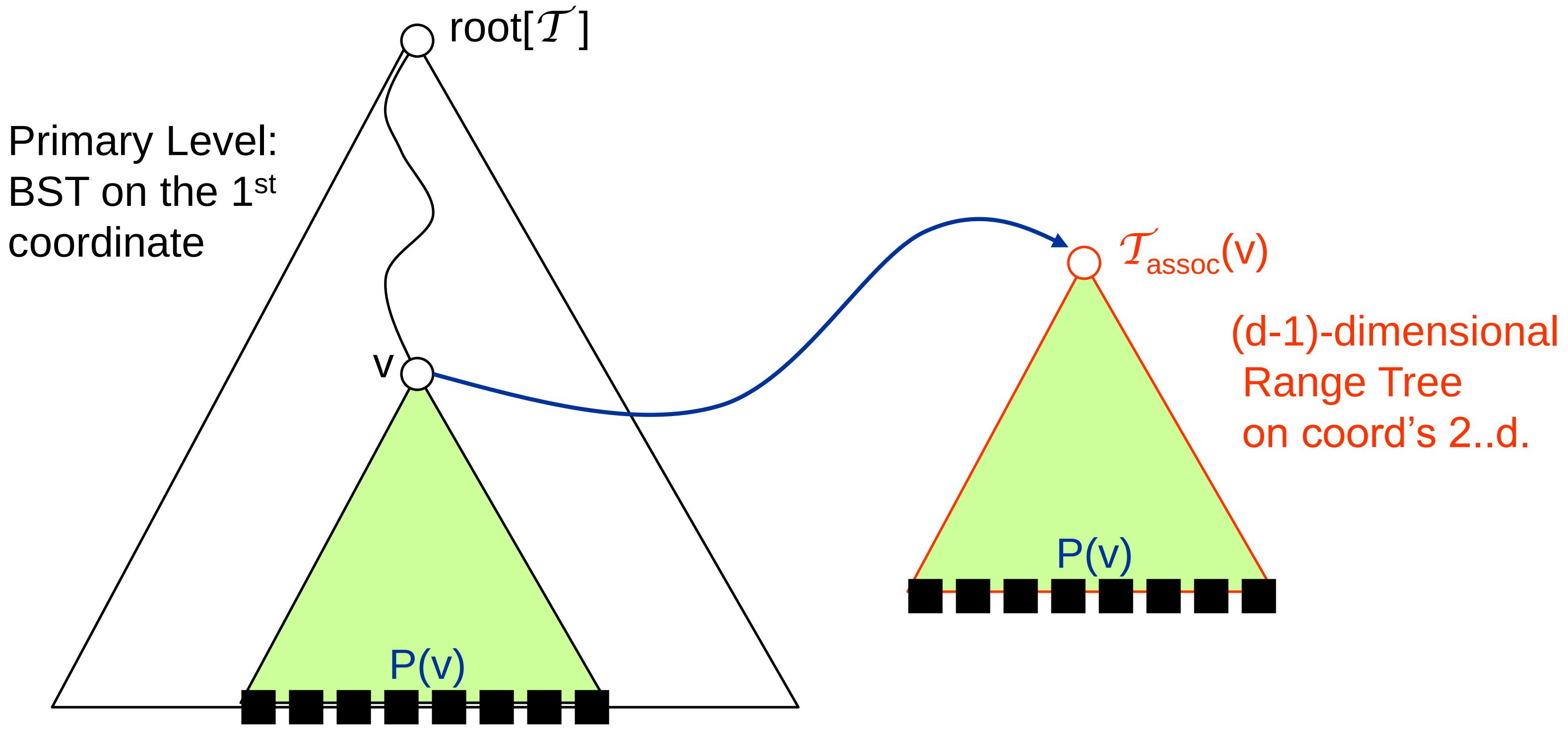
Query Time: $O(K + \log^2 n)$ will be improved to O(K + log n) by Fractional Cascading

Construction Time: O(n log n)

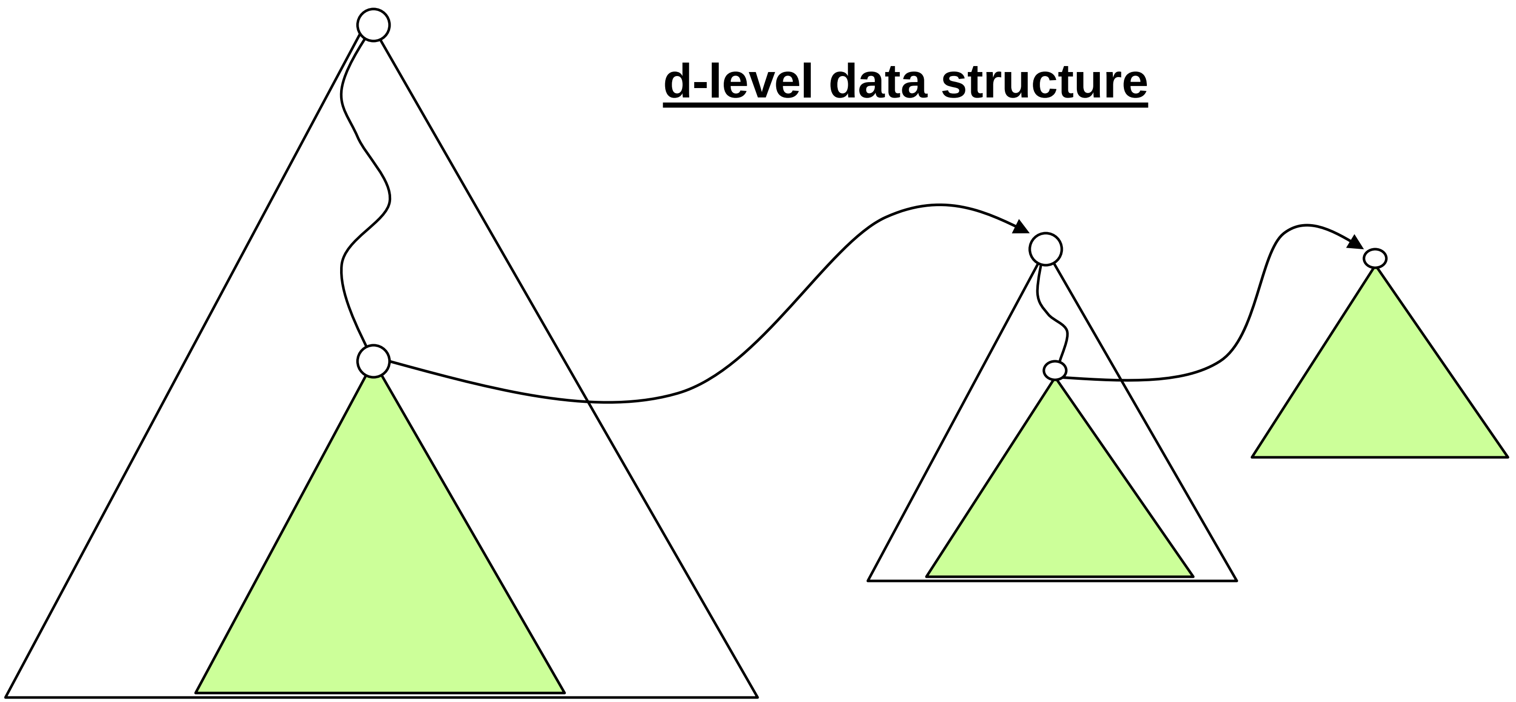
Space: O(n log n)

Higher Dimensional Range Trees

$P = \{ p_1, p_2, \ldots, p_n \} \subseteq \Re^d, \quad p_i = (x_{i1}, x_{i2}, \ldots, x_{id}), i=1..n.$



Higher Dimensional Range Trees



Higher Dimensional Range Trees

Query Time: $Q_d(n) = O(K + \log^d n)$ improved to $O(K + \log^{d-1} n)$ by Frac. Casc.

Construction Time: $T_d(n) = O(n \log^{d-1} n)$

Space: $S_d(n) = O(n \log^{d-1} n)$

$$\left\{\begin{array}{c} T_d(n) = 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + O(n) \\ T_2(n) = O(n \log n) \end{array}\right\} \Rightarrow T_d(n) = O(n \log^{d-1} n)$$

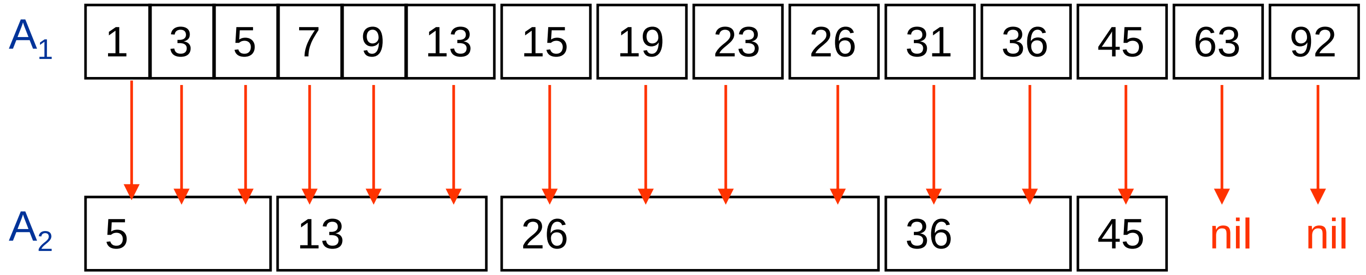
$$\left\{\begin{array}{c} S_d(n) = 2S_d\left(\frac{n}{2}\right) + S_{d-1}(n) + O(1) \\ S_2(n) = O(n \log n) \end{array}\right\} \Rightarrow S_d(n) = O(n \log^{d-1} n)$$

$$\left\{\begin{array}{c} Q_d(n) = O(K) + \hat{Q}_d(n) \\ \hat{Q}_d(n) = O(\log n) + O(\log n) \cdot \hat{Q}_{d-1}(n) \\ \hat{Q}_2(n) = O(\log^2 n) \end{array}\right\} \Rightarrow \left\{\begin{array}{c} \hat{Q}_d(n) = O(\log^d n) \\ Q_d(n) = O(K + \log^d n) \end{array}\right.$$

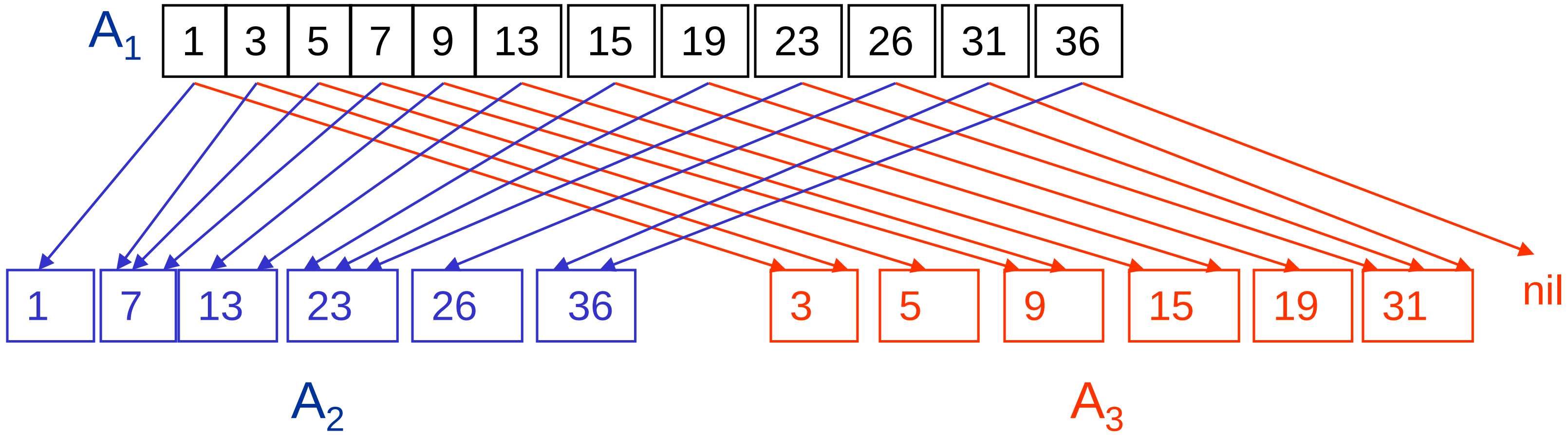
Fractional Cascading

IDEA: Save repeated cost of binary search in many sorted lists for the same range [y : y'] if the list contents for one are a subset of the other.

- $A_2 \subseteq A_1$
- Binary search for y in A_1 to get to $A_1[i]$.
- Follow pointer to A_2 to get to $A_2[j]$.
- Now walk to the right in each list.

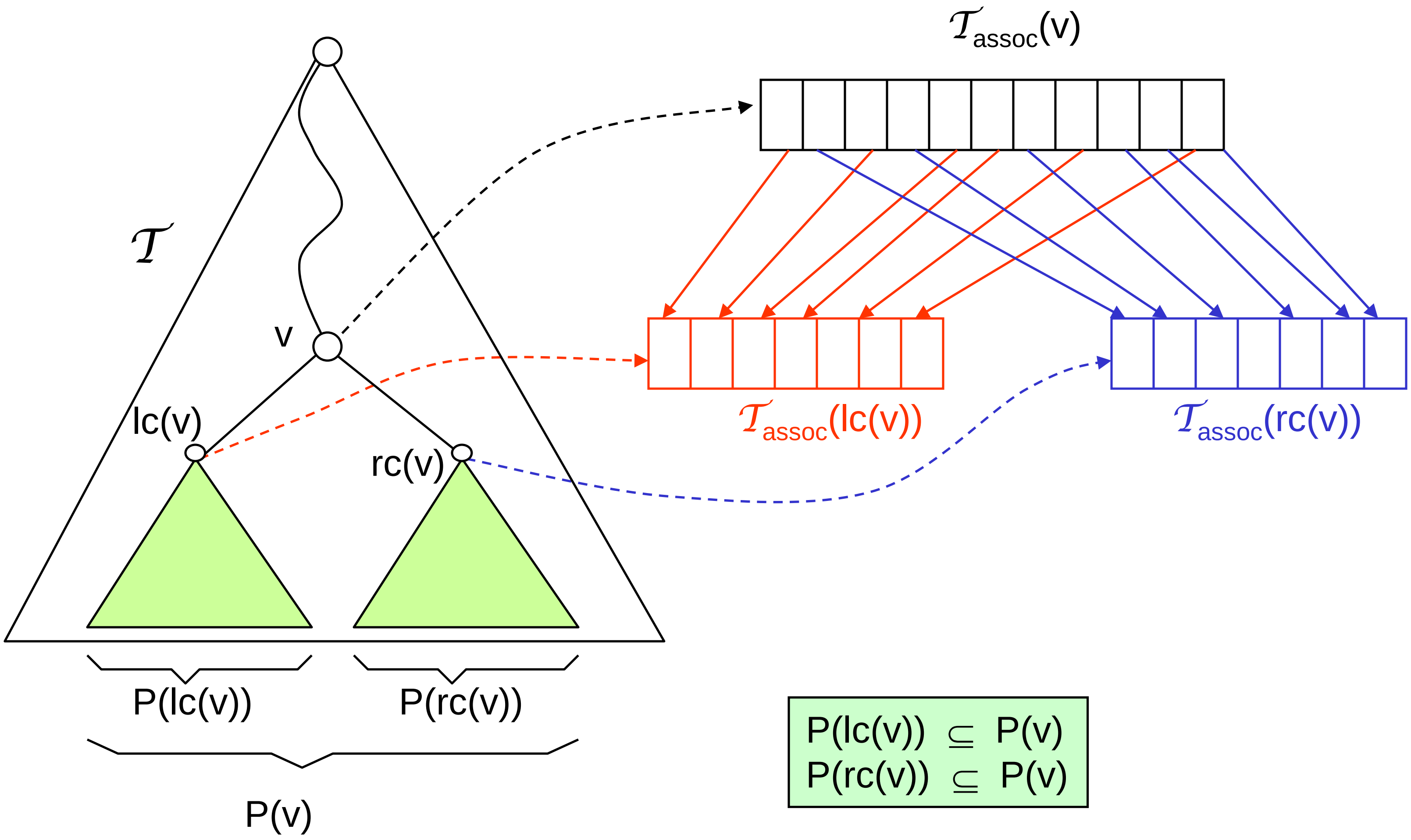


Fractional Cascading

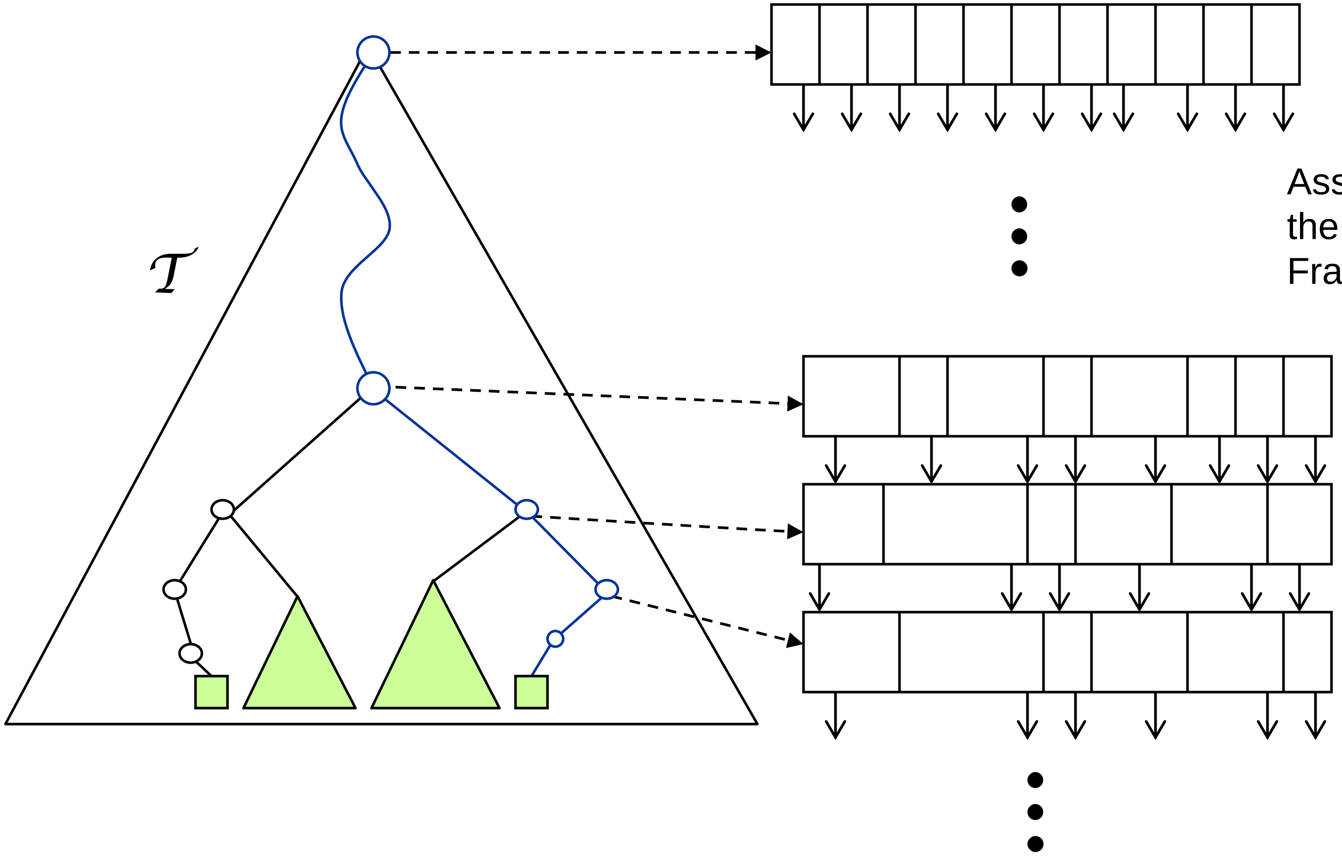


- $A_2 \subseteq A_1$, $A_3 \subseteq A_1$.
- No binary search in A_2 and A_3 is needed.
- Do binary search in A_1.
- Follow blue and red pointers from there to A_2 and A_3.
- Now we have the starting point in each sorted list. Walk to the right & report.

Layered 2D Range Tree



Layered 2D Range Tree



Associated Structures at the secondary level by Fractional Cascading

Layered 2D Range Tree (by Fractional Cascading)

Query Time:

$Q_2(n) = O(\log n + \sum_v (K_v + \log n)) = O(\sum_v K_v + \log^2 n) = O(K + \log^2 n)$

improves to:

$Q_2(n) = O(\log n + \sum_v (K_v + 1)) = O(\sum_v K_v + \log n) = O(K + \log n)$.

For d-dimensional range tree query time improves to:

$$\left\{\begin{array}{c} Q_d(n) = O(K) + \hat{Q}_d(n) \\ \hat{Q}_d(n) = O(\log n) + O(\log n)\cdot \hat{Q}_{d-1}(n) \\ \hat{Q}_2(n) = O(\log n) \end{array}\right\} \Rightarrow \; Q_d(n) = O(K + \log^{d-1} n)$$