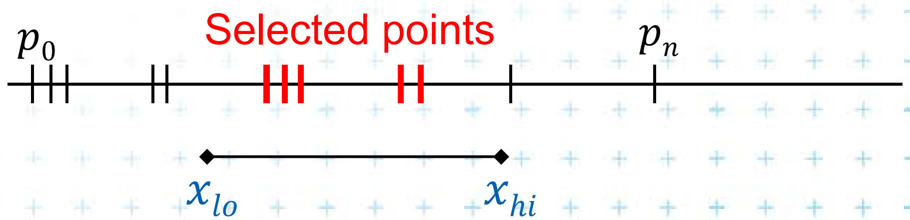
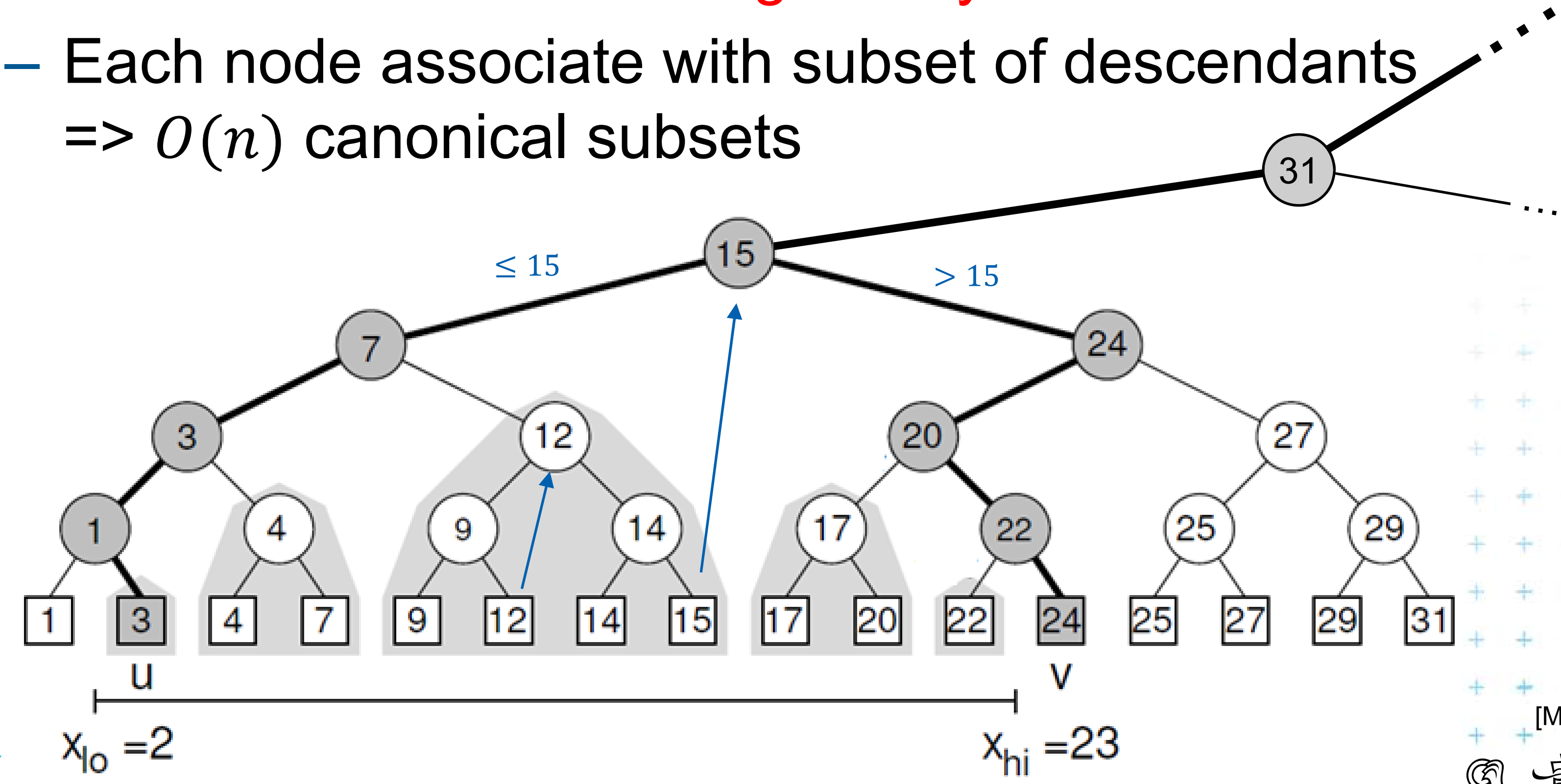
1D range queries (interval queries)

- Query: Search the interval $[x_{lo}, x_{hi}]$
- Search space: Points $P = \{p_1, p_2, \ldots, p_n\}$ on the line
  - a) Binary search in an array
    - Simple, but
    - not generalize to any higher dimensions
  - b) Balanced binary search tree
    - 1D range tree
    - maintains canonical subsets
    - generalize to higher dimensions



1D range tree definition

- Balanced binary search tree (with repeated keys)
  - leaves – sorted points
  - inner node label – the largest key in its left child
  - Each node associate with subset of descendants => $O(n)$ canonical subsets



[Mount]

Canonical subsets and <2,23> search

- Canonical subsets for this subtree are — #

{ {1}, {3}, …, {31}, — 16

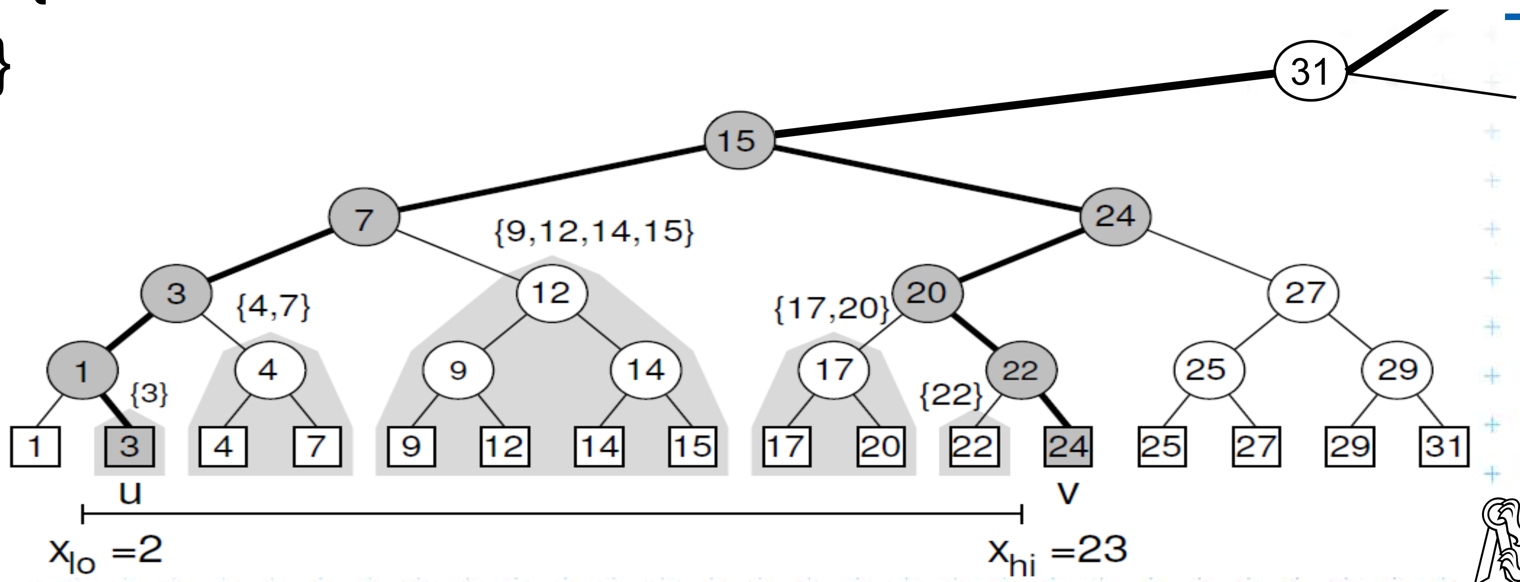
{1, 3}, {4, 7}, …, {29, 31} — 8

{1, 3, 4, 7}, {9, 12, 14, 15}, …, {25, 27, 29, 31} — 4

{1, 3, 4, 7, 9, 12, 14, 15}, {17, 20, 22, 24, 25, 27, 29, 31} — 2

{1, 3, 4, 7, 9, 12, 14, 15, 17, 20, 22, 24, 25, 27, 29, 31} — 1

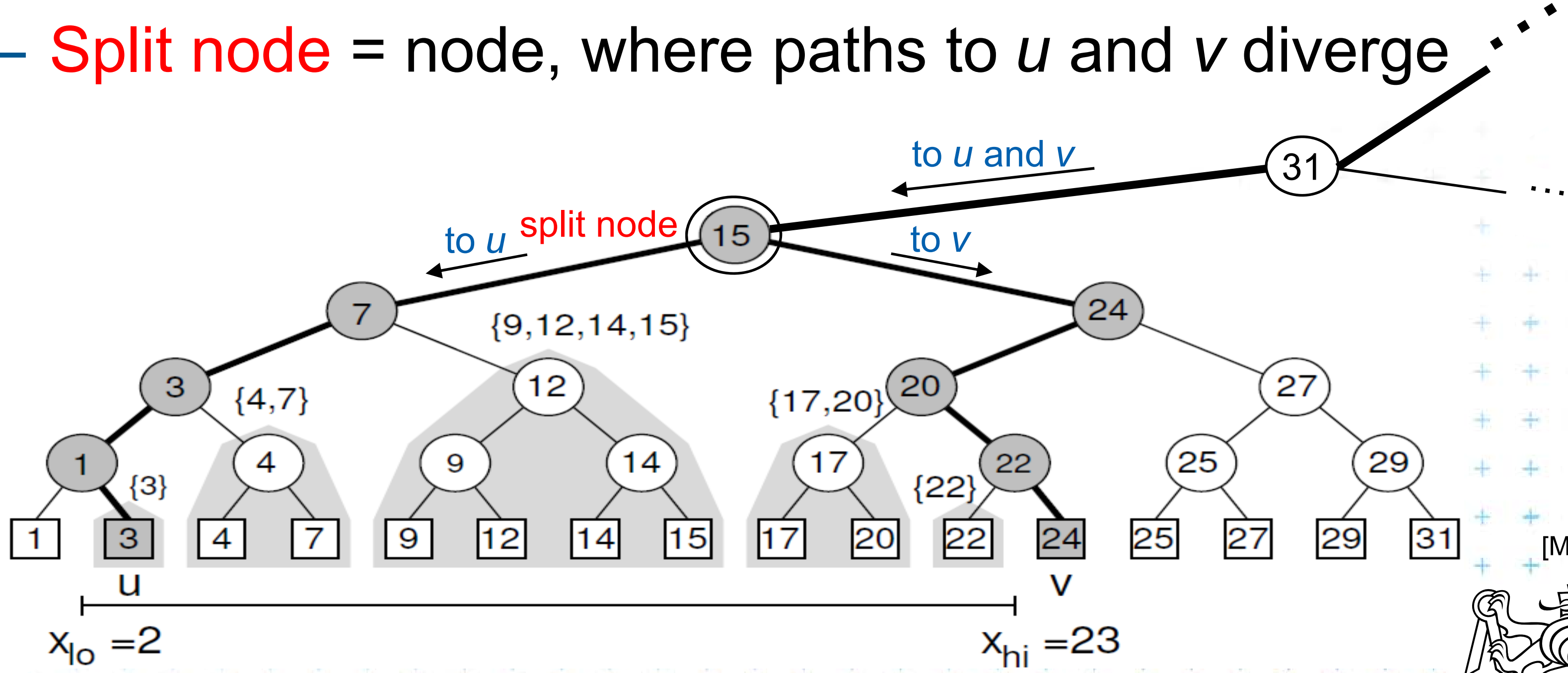
} — O(n)



[Mount]

1D range tree search interval <2,23>

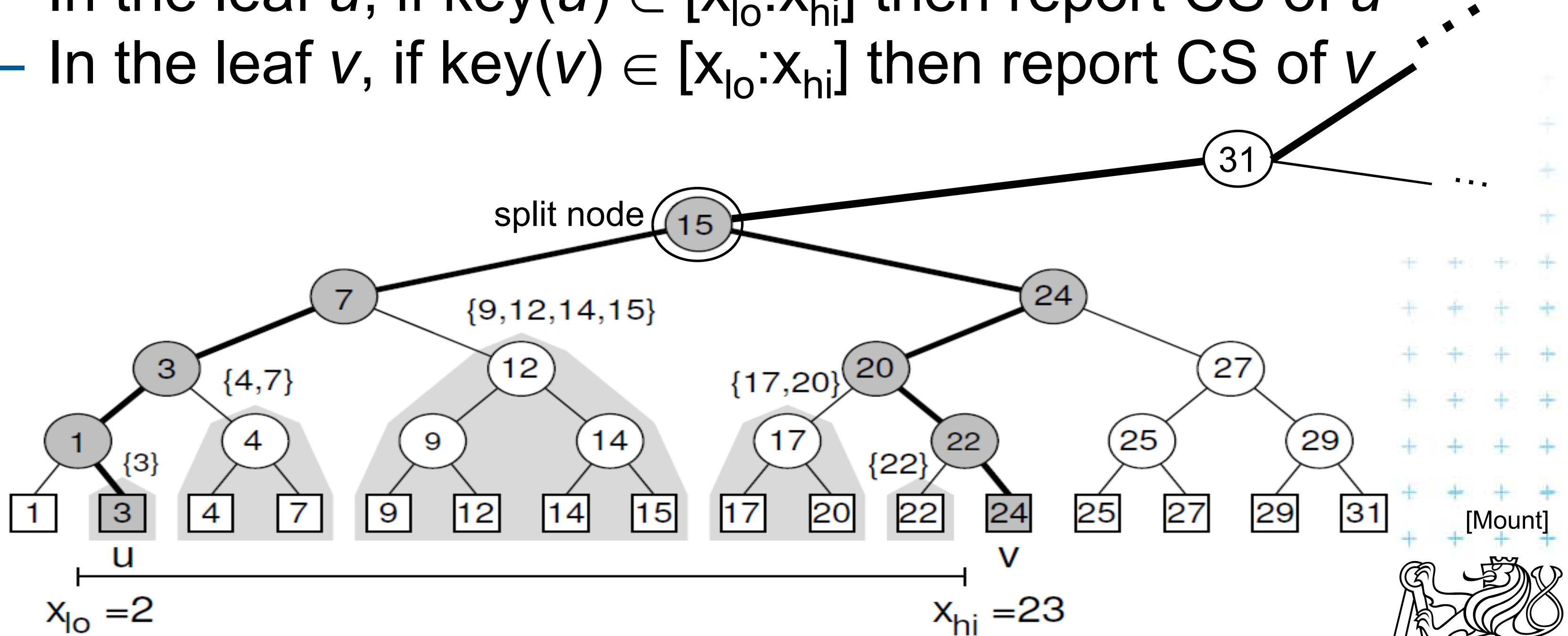
- Canonical subsets for any range found in O(log *n*)
  - Search x_{lo}: Find leftmost leaf *u* with key(*u*) $\geq x_{lo}$ 2 -> 3
  - Search x_{hi}: Find leftmost leaf *v* with key(*v*) $\geq x_{hi}$ 23 -> 24
  - Points between *u* and *v* lie within the range => report canon. subsets of maximal subtrees between *u* and *v*
  - Split node = node, where paths to *u* and *v* diverge



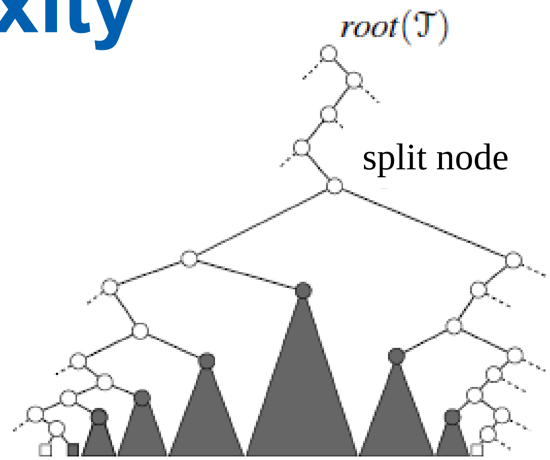
[Mount]

1D range tree search

- Reporting the subtrees (below the split node)
  - On the path to *u* whenever the *path goes left*, report the canonical subset (CS) associated to right child
  - On the path to *v* whenever the *path goes right*, report the canonical subset associated to left child
  - In the leaf *u*, if key(*u*) $\in [x_{lo}:x_{hi}]$ then report CS of *u*
  - In the leaf *v*, if key(*v*) $\in [x_{lo}:x_{hi}]$ then report CS of *v*



1D range tree search complexity



[Berg]

- Path lengths O(log n)

  => O(log n) canonical subsets (subtrees)

- Range counting queries
  - Return just the number of points in given range
  - Sum the total numbers of leaves stored in maximum subtree roots … O(log n) time

- Range reporting queries
  - Return all k points in given range
  - Traverse the canonical subtrees … O(log n + k) time

- O(n) storage, O(n log n) preprocessing (sort P)

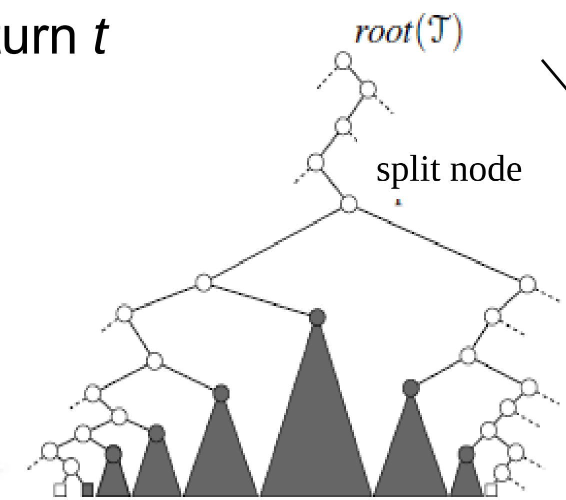
Find split node

FindSplitNode(*T*, [x:x’])

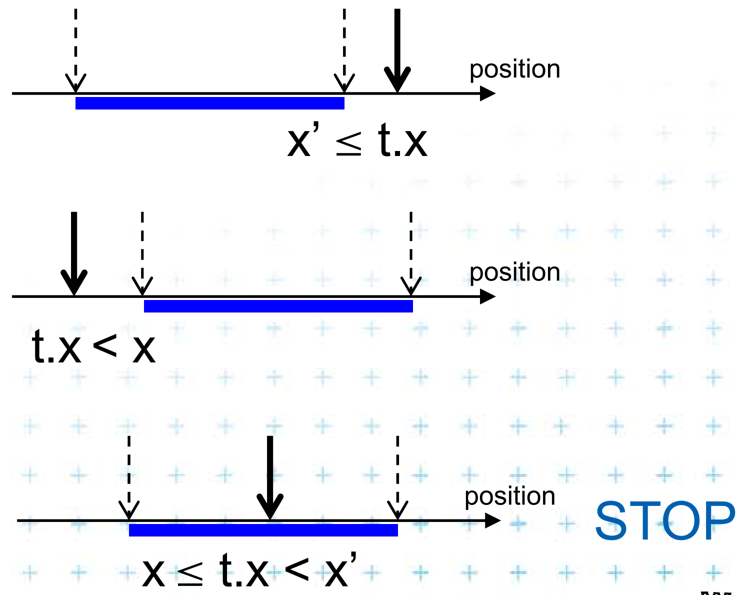
Input: Tree *T* and Query range [x:x’], x ≤ x’

Output: The node, where the paths to x and x’ split or the leaf, where both paths end

1. *t = root(T)*
2. while(*t* is not a leaf **and (**x’ ≤ t.x **or** t.x < x**)**) // *t* out of the range [x:x’]
3. if(*x’* ≤ t.x) *t = t.left*
4. else *t = t.right*
5. return *t*



[Berg]



1D range search

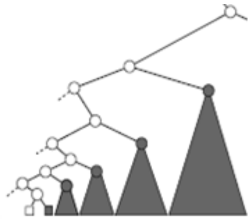
(2D on slide 30)

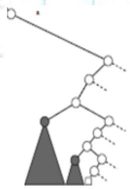
1dRangeQuery(*t*, [x:x’])

Input: 1d range tree *t* and Query range $[x : x']$

Output: All points in *t* lying in the range

```
t_split = FindSplitNode( t, x, x’ )          // find interval point t ∈ [x:x’]
if( t_split is leaf )    // e.g. Searching [16:17] or [16:16.5] both stops in the leaf 17 in the previous example
    check if the point in t_split must be reported    // t_x ∈ [x : x']
else // follow the path to x, reporting points in subtrees right of the path
    t = t_split.left
    while( t is not a leaf )
        if( x ≤ t.x)
            ReportSubtree( t.right )    // any kind of tree traversal
            t = t.left
        else t = t.right
    check if the point in leaf t must be reported
    // Symmetrically follow the path to x’ reporting points left of the path
    t = t_split.right …
```



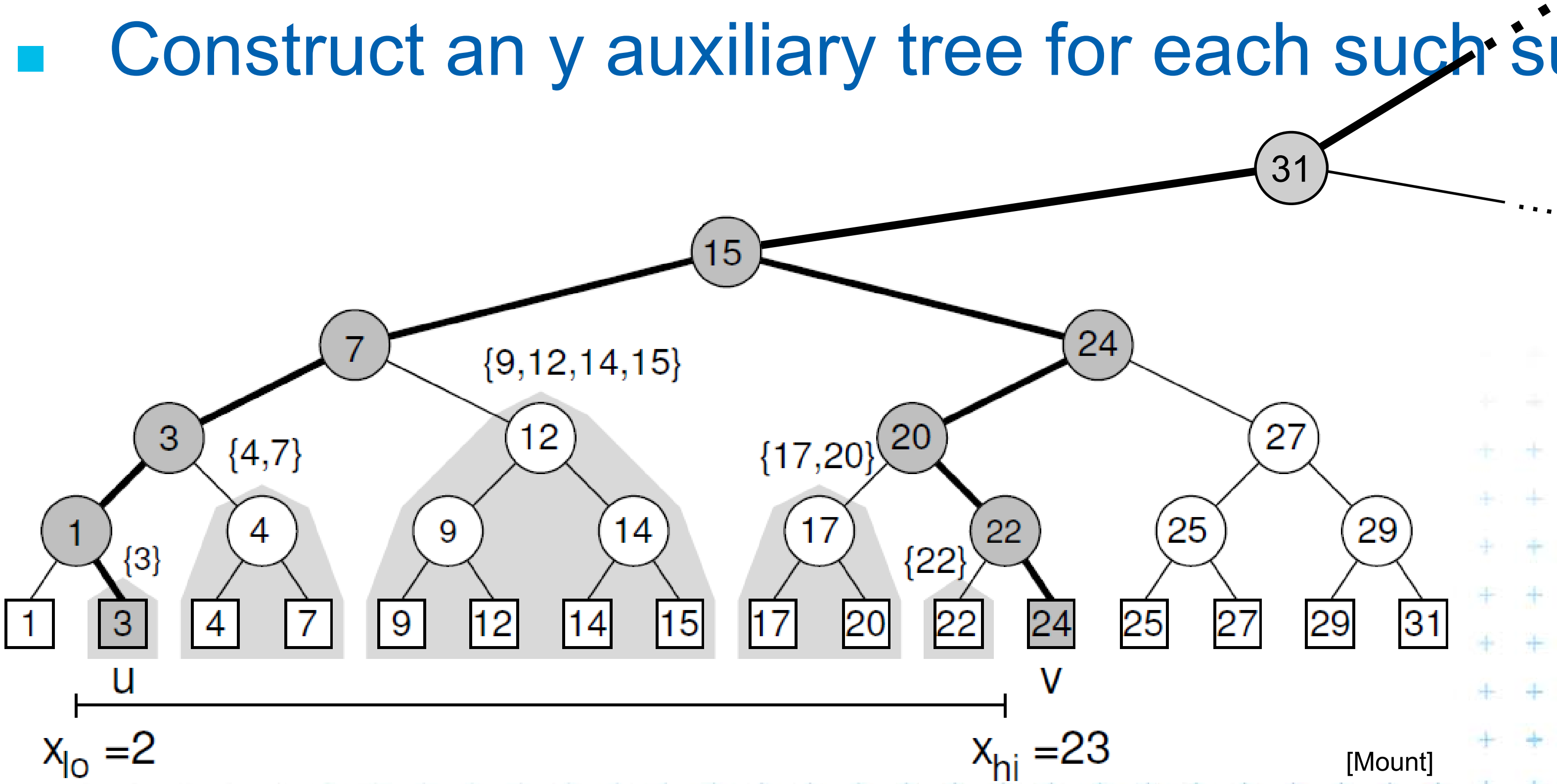


Multidimensional range searching

- Equal principle – find the largest subtrees contained within the range
- Separate one *n*-dimensional search into *n* 1-dimensional searches
- Different tree organization
  - Orthogonal (Multilevel) range search tree
    e.g. nd range tree
  - Kd tree

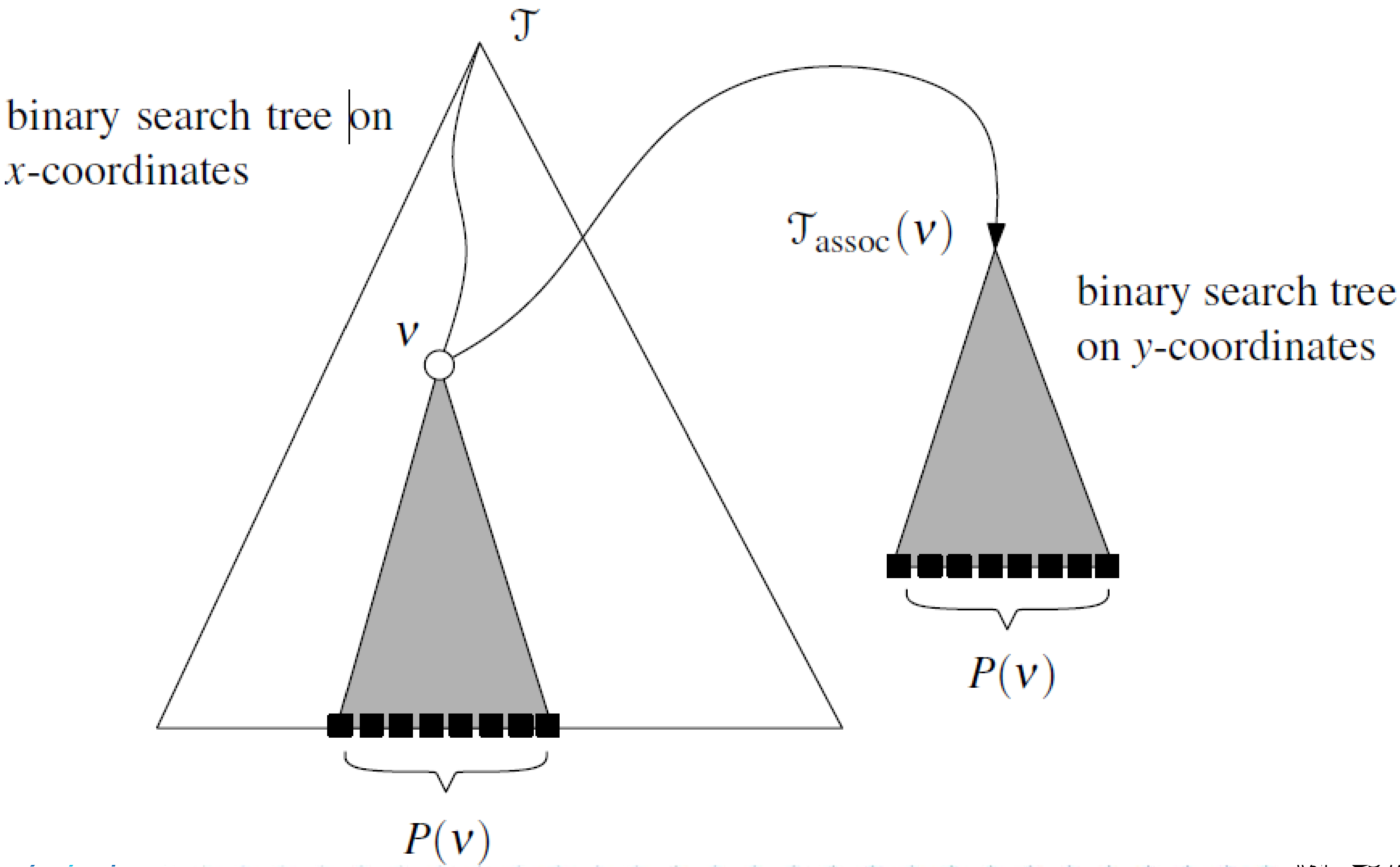
From 1D to 2D range tree

- Search points from $[Q.x_{lo}, Q.x_{hi}]$ $[Q.y_{lo}, Q.y_{hi}]$
- 1d range tree: log n canonical subsets based on x
- Construct an y auxiliary tree for each such subset

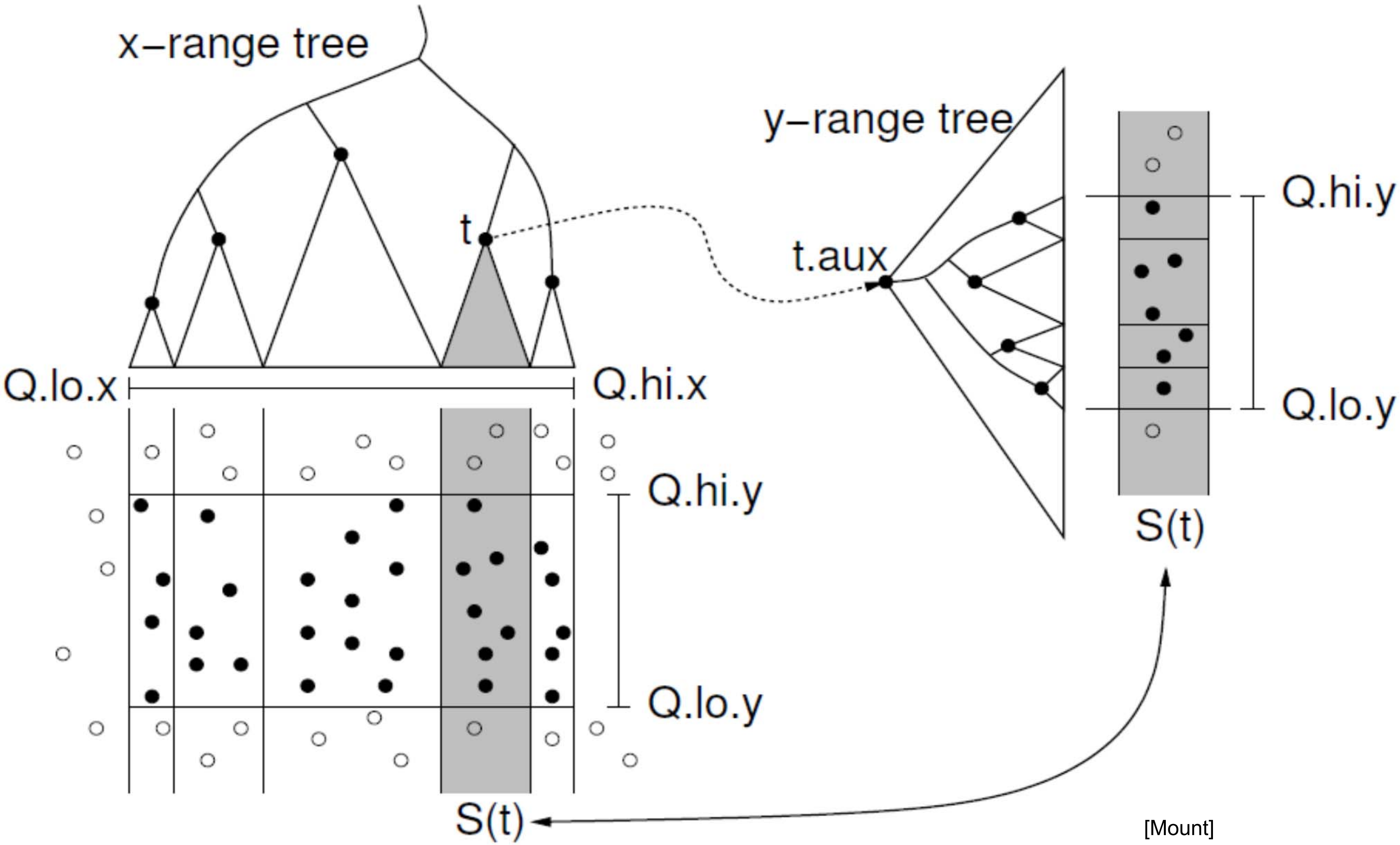


[Mount]

y-auxiliary tree for each canonical subset



2D range tree



[Mount]

2D range search

2dRangeQuery(*t*, [x:x’] × [y:y’])

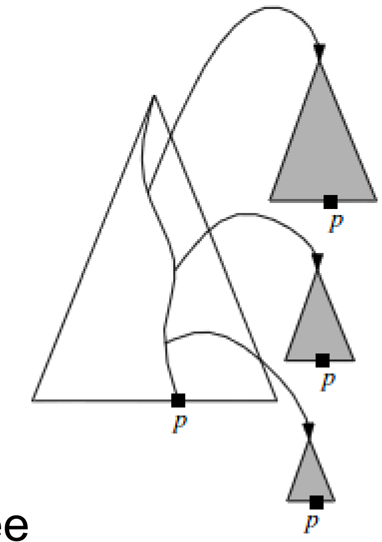
Input: 2d range tree *t* and Query range

Output: All points in *t* laying in the range

t_{split} = FindSplitNode(*t, x, x’*)
if(t_{split} is leaf)
    check if the point in t_{split} must be reported … t.x ∈ [x:x’], t.y ∈ [y:y’]
else // follow the path to *x*, calling 1dRangeQuery on y
    t = t_{split}.left // path to the left
    while(t is not a leaf)
        if($x \leq$ t.x)
            1dRangeQuerry(t_{assoc}(*t.right*), [*y:y’*]) // check associated subtree
            t = t.left
        else *t = t.right*
    check if the point in leaf *t* must be reported … t.x ≤ x’, t.y ∈ [y:y’]
    Similarly for the path to x’ … // path to the right

2D range tree

- Search $O(\log^2 n + k)$ … $\log n$ in x, $\log n$ in y
- Space $O(n \log n)$
  - $O(n)$ the tree for x-coords
  - $O(n \log n)$ trees for y-coords
    - Point p is stored in all canonical subsets along the path from root to leaf with p,
    - once for x-tree level (only in one x-range)
    - each canonical subsets is stored in one auxiliary tree
    - $\log n$ levels of x-tree => $O(n \log n)$ space for y-trees



[Berg]

- Construction - $O(n \log n)$
  - Sort points (by x and by y). Bottom up construction

Canonical subsets

- Canonical subsets for this subtree are #

  { {1}, {3}, …, {31}, 16

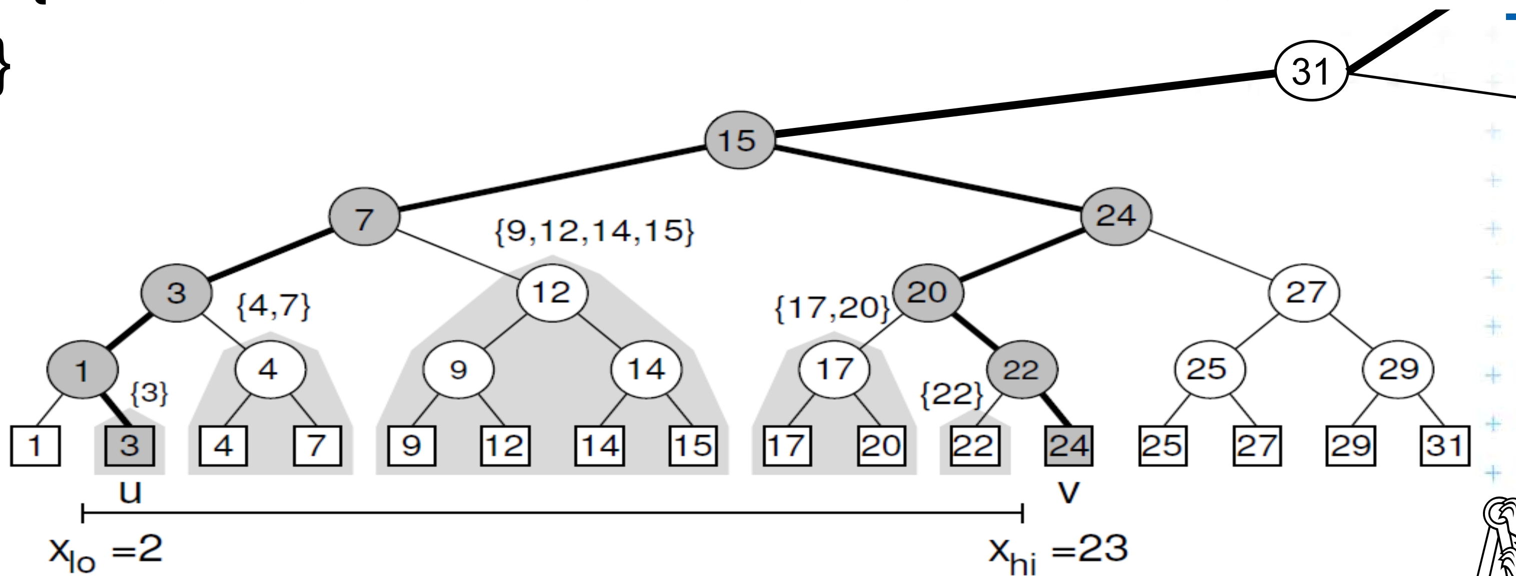
  {1, 3}, {4, 7}, …, {29, 31} 8

  {1, 3, 4, 7}, {9, 12, 14, 15}, …, {25, 27, 29, 31} 4

  {1, 3, 4, 7, 9, 12, 14, 15}, {17, 20, 22, 24, 25, 27, 29, 31} 2

  {1, 3, 4, 7, 9, 12, 14, 15, 17, 20, 22, 24, 25, 27, 29, 31} 1

  } O(n)



[Mount]