DCGI

DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

GEOMETRIC SEARCHING
PART 2: RANGE SEARCH

PETR FELKEL

FEL CTU PRAGUE

felkel@fel.cvut.cz

https://cw.felk.cvut.cz/doku.php/courses/a4m39vg/start

Based on [Berg] and [Mount]

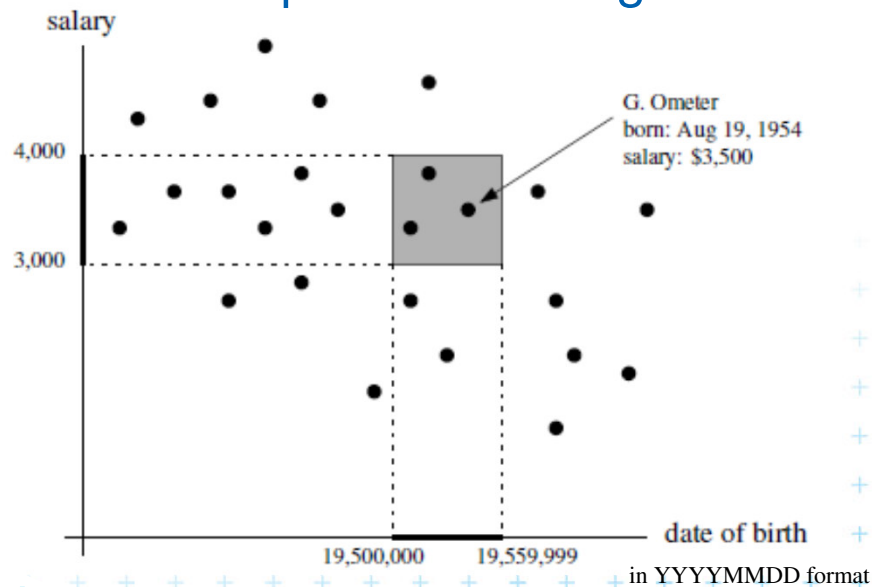**Version from 9.10.2014**

Range search

- Orthogonal range searching

- Canonical subsets

- 1D range tree

- Kd-tree

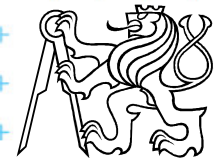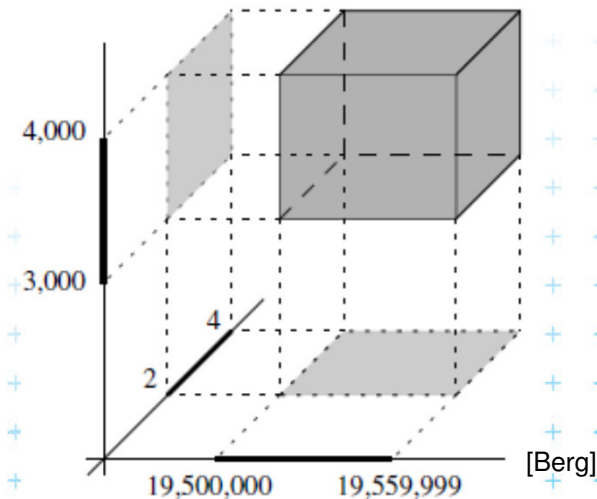- 2D-nD Range tree
  - With fractional cascading (Layered tree)

Orthogonal range searching

– Given a set of points P, find the points in the region Q
  - Search space: a set of points P (somehow represented)
  - Query: intervals Q (axis parallel rectangle)
  - Answer: points contained in Q

– Example: Databases (records->points)
  - Find the people with given range of salary, date of birth, kids, …

2D: axis parallel rectangle

3D: axis parallel box

salary

G. Ometer
born: Aug 19, 1954
salary: $3,500

4,000

3,000

date of birth

19,500,000   19,559,999

in YYYYMMDD format

4,000

3,000

4

2

19,500,000   19,559,999

[Berg]

Felkel: Computational geometry
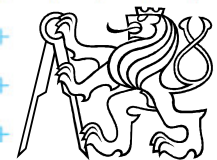
(3)

Orthogonal range searching

- Query region = axis parallel rectangle
  - nDimensional search can be decomposed into set of 1D searches (separable)

Other range searching variants

- Search space S: set of
  - line segments,
  - rectangles, …

- Query region Q: any other region
  - disc,
  - polygon,
  - halfspace, …

- Answer: subset of S laying in Q

- We concentrate on points in orthogonal ranges

How to represent the search space?

- Not all possible combination can be in the output (not the whole power set)

- => Represent only the "selectable" things
  (a well selected subset –> one of the canonical subsets)

DCGI

Subsets selectable by given range class

■ The number of subsets that can be selected by simple ranges Q is limited

■ It is usually much smaller than the power set of P

  – Power set of P where P = {1,2,3,4} (potenční množina)
    is {{ }, {1},{2},{3},{4}, {1,2},{1,3},{1,4}, {2,3},…,{2,3,4},
    {1,2,3,4} }   … $O(2^n)$
    i.e. set of all possible subsets

  – Simple rectangular queries are limited

    • Defined by max 4 points along 4 sides
      => $O(n^4)$ of $O(2^n)$ power set

    • Moreover – not all sets can be formed
              by □ query

   e.g. sets {1,4} and {1,2,4} cannot be formed

[Mount]

DCGI
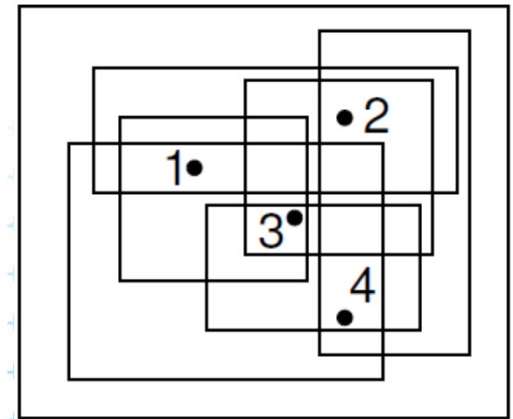
Canonical subsets S_i

- Search space $S=(P,Q)$ represented as a collection of canonical subsets $\{S_1, S_2, \ldots, S_k\}$, each $S_i \subseteq S$,
  - S_i may overlap each other (elements can be multiple times there)
  - Any set can be represented as disjoint union disjunktní sjednocení of canonical subsets S_i each element knows from which subset it came
  - Elements of disjoint union are ordered pairs (x, i) (every element x with index i of the subset S_i)

- S_i may be selected in many ways
  - from n singletons $\{p_i\}$ ... $O(n)$
  - to power set of P   ... $O(2^n)$
  - Good DS balances between total number of canonical subsets and number of CS needed to answer the query

DCGI
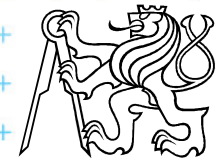
1D range queries (interval queries)

- 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
    (values in inner nodes are not reachable in particular level below,
    to get them, we must traverse back to root)

  b) Balanced binary search tree

  - 1D range tree
  - maintains canonical subsets
 - generalize to higher dimensions
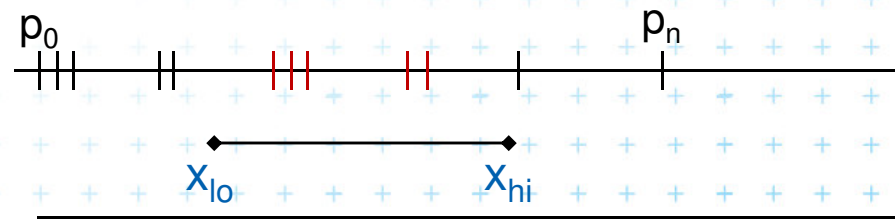
1D range tree definition

- ## Balanced binary search tree

  - 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]

DCGI
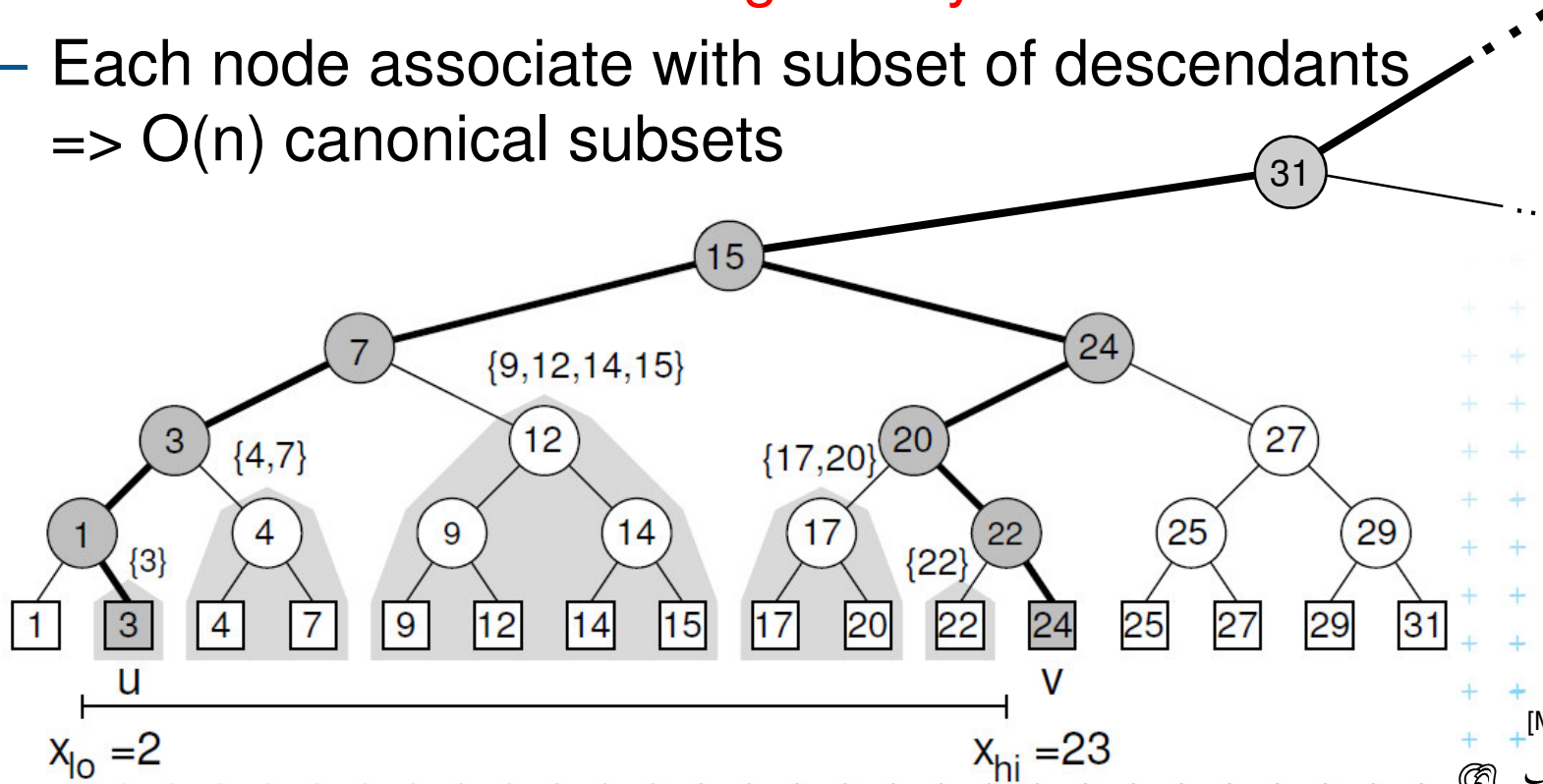
Canonical subsets and <2,23> search

- 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]

DCGI
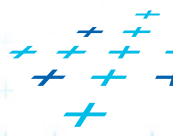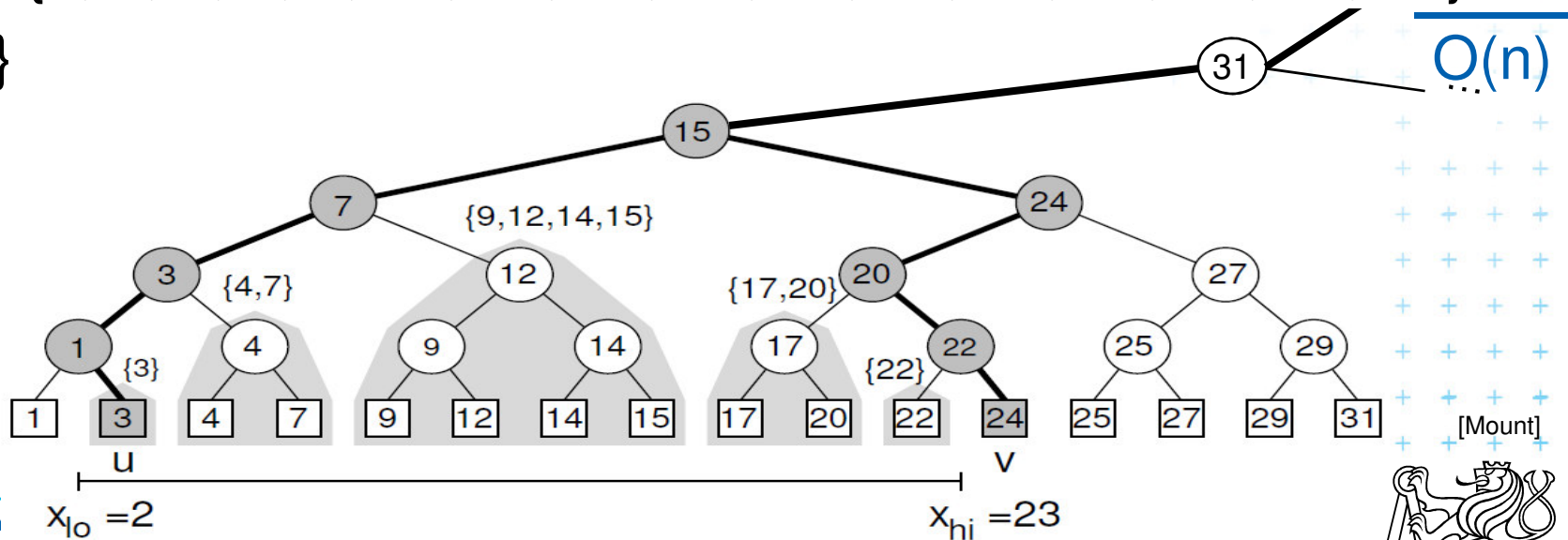
1D range tree search interval <2,23>

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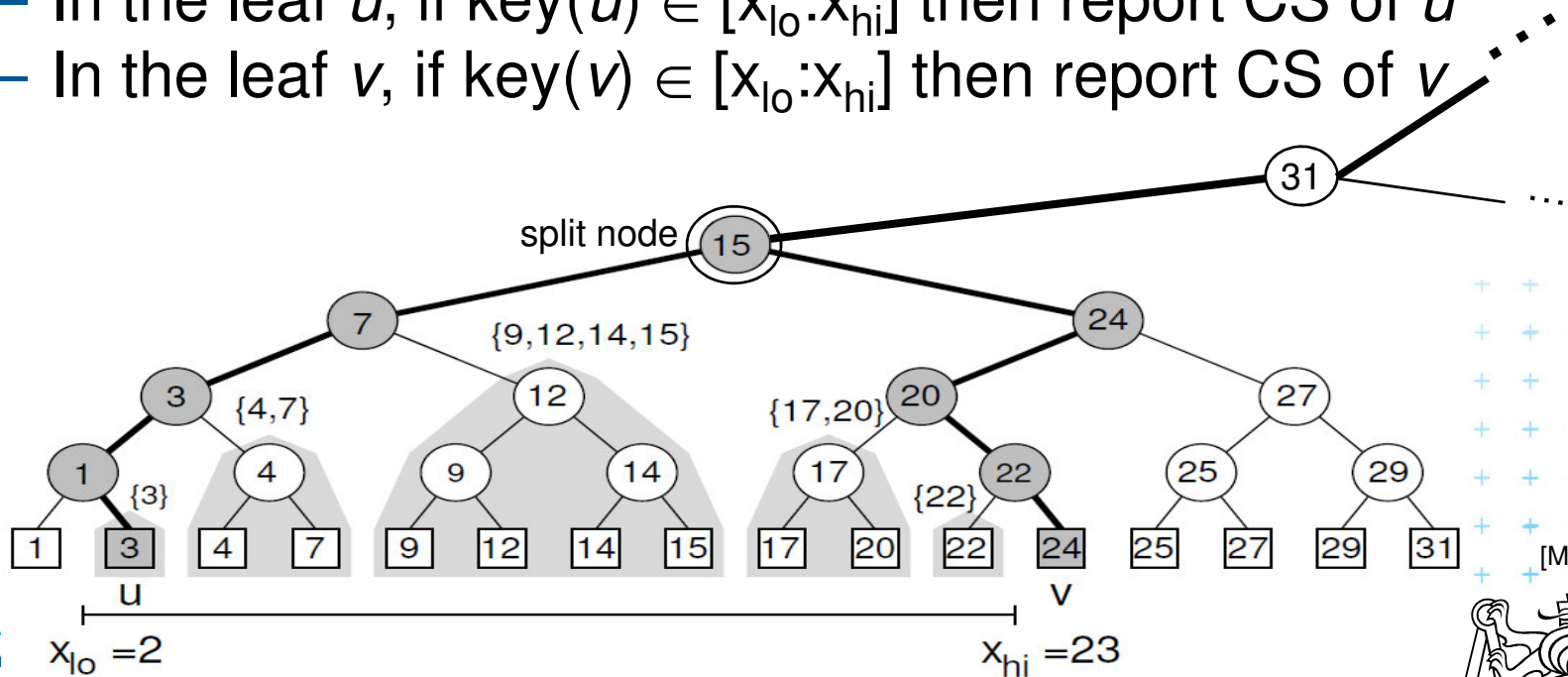 - Search x_{hi}: Find leftmost leaf v with $key(v) \geq x_{hi}$ 23 -> $\boxed{24}$
  - Points between u and v lie within the range => report canon. subsets of maximal subtrees between u and v
  - Split node = node, where paths to u and v diverge

to u and v

to u split node (15) to v

(31)

...

$\{9,12,14,15\}$

(7)   (24)

(3) $\{4,7\}$ (12) $\{17,20\}$ (20)   (27)

(1)   (4)   (9)   (14)   (17)   (22)   (25)   (29)

$\{3\}$ $\{22\}$

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

u   v

$x_{lo} = 2$ $x_{hi} = 23$

[Mount]

Felkel: Computational geometry

(12)
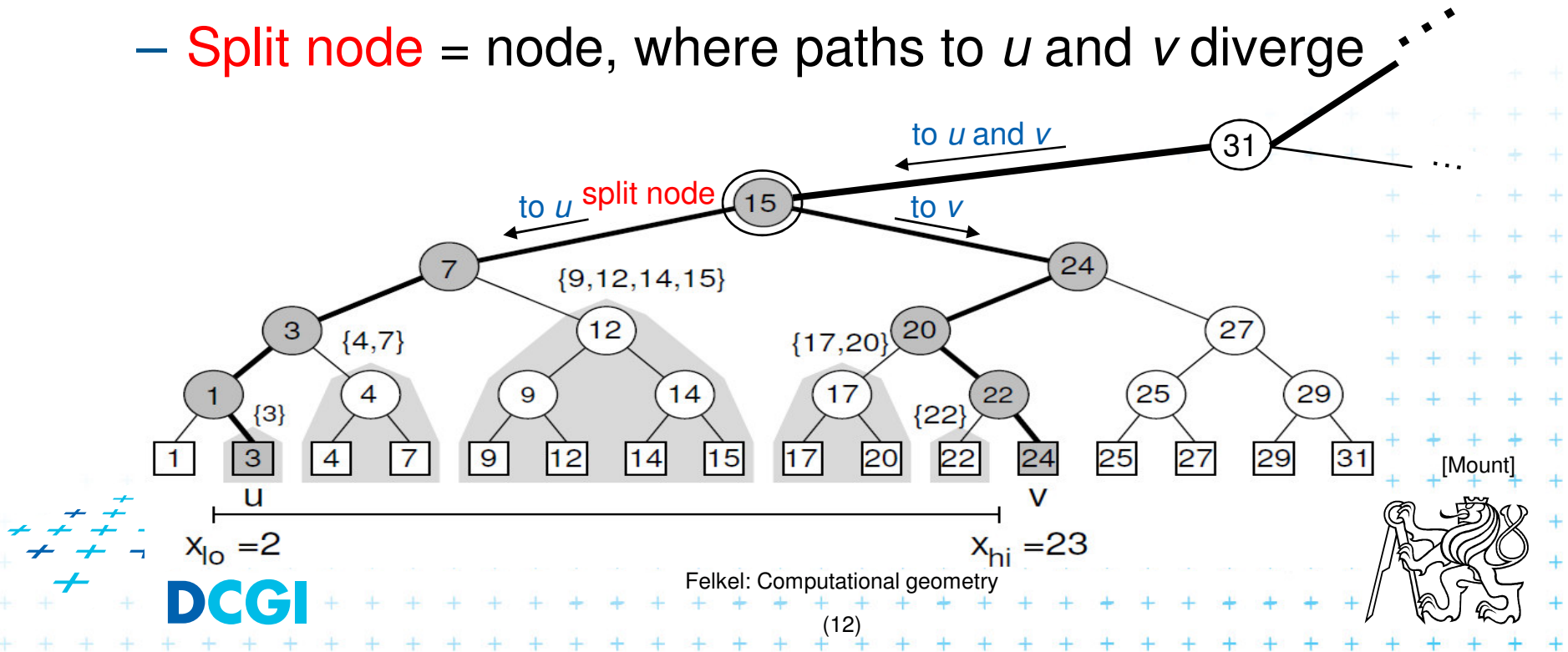
1D range tree search

- 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

split node 15

31

7   {9,12,14,15}   24

3   {4,7}   12   {17,20}   20   27

1   4   9   14   17   22   25   29

{3}   {22}

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

u   v

[Mount]

$x_{lo} = 2$ $x_{hi} = 23$

Felkel: Computational geometry

(13)

1D range tree search complexity

root($\mathcal{T}$)

split node

[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 maximal
    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
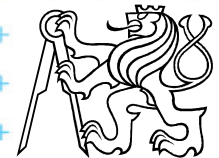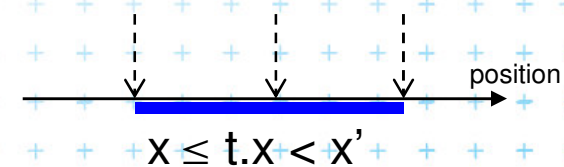
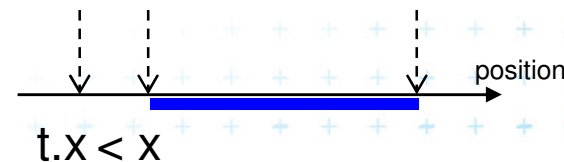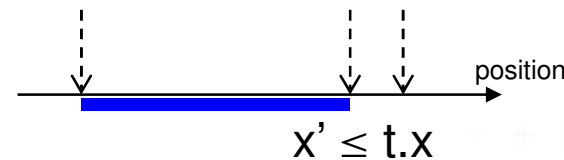- **O(n) storage, O(n log n) preprocessing (sort P)**

DCGI
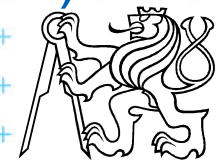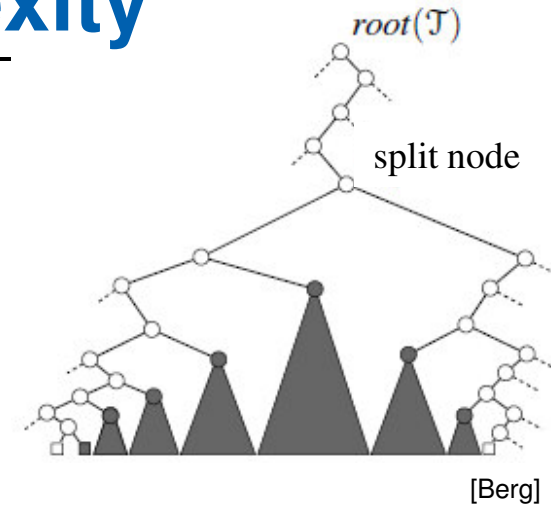
Find split node

FindSplitNode(T, [x:x'])
Input: Tree T and Query range [x:x'], $x \leq x'$
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 (**$x' \leq t.x$ **or** $t.x < x$**)**)   // out of the range [x:x']

3.   if($x' \leq t.x$) $t = t.left$

4.   else $t = t.right$

5.   return t

root($\mathcal{T}$)

split node

position

$x' \leq t.x$

position

$t.x < x$

position

$x \leq t.x < x'$

[Berg]

DCGI

1dRangeQuery(*t*, [x:x'])
Input: 1d range tree *t* and Query range
Output: All points in *t* lying in the range

1. t_{split} = FindSplitNode(*t, x, x'*)   // find interval point t $\in$ [x:x']
2.   if(t_{split} is leaf)   // e.g. Searching [16:17] or [16:16.5] both stops in the leaf 17 in the previous example
3.   check if the point in t_{split} must be reported   // $t_x \in [x:x']$
4.   else // follow the path to *x*, reporting points in subtrees right of the path
5.   t = t_{split}.left
6.   while(t is not a leaf)
7.   if($x \leq$ t.x)
8.   ReportSubtree(*t.right*)   // any kind of tree traversal
9.   *t = t.left*
10.   else *t = t.right*
11.   check if the point in leaf *t* must be reported
12.   // Symmetrically follow the path to x' reporting points left of the path
       t = t_{split}.right …

DCGI

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
  - Kd tree
  - Orthogonal (Multilevel) range search tree e.g. nd range tree

Kd-tree

- Easy to implement

- Good for different searching problems (counting queries, nearest neighbor,…)

- Designed by Jon Bentley as k-dimensional tree (2-dimensional kd-tree was a 2-d tree, …)

- Not the asymptotically best for orthogonal range search (=> range tree is better)

- Types of queries

  – Reporting – points in range
 – Counting – number of points in range

Kd-tree principle

- Subdivide space according to different dimension (*x*-coord, then *y*-coord, …)

- This subdivides space into rectangular cells => hierarchical decomposition of space

- In node *t* store: cutDim, cutVal, (size (for counting queries))

= Cutting line

Each tree node represents a region

Subdivision

Tree structure

[Mount]

Where is a mistake in the figure?

DCGI

Kd-tree principle

- ## Which dimension to cut?   (cutDim)

  - Cycle through dimensions (round robin)
    - Save storage – cutDim is implicit ~ depth in the tree
    - May produce elongated cells (if uneven data distribution)
  - Greatest spread (the largest difference of coordinates)
    - Adaptive
    - Called "Optimal kd-tree"

- ## Where to cut?   (cutVal)

  - Median, or midpoint between upper and lower median -> $O(n)$

  - Presort coords of points in each dimension (x-, y-,...) for $O(1)$ median – resp. $O(d)$ for all d dimensions

Kd-tree principle

- What about points on the cell boundary?
  - Boundary belongs to the left child
  - Left: $\quad p_{cutDim} \leq cutVal$
  - Right: $\quad p_{cutDim} > cutVal$
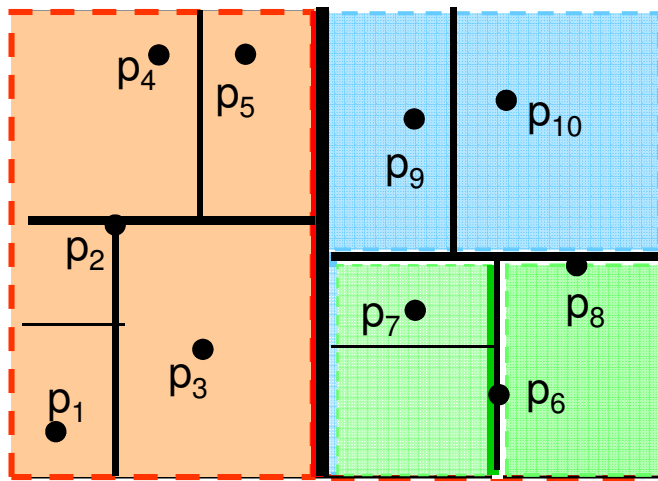
Subdivision
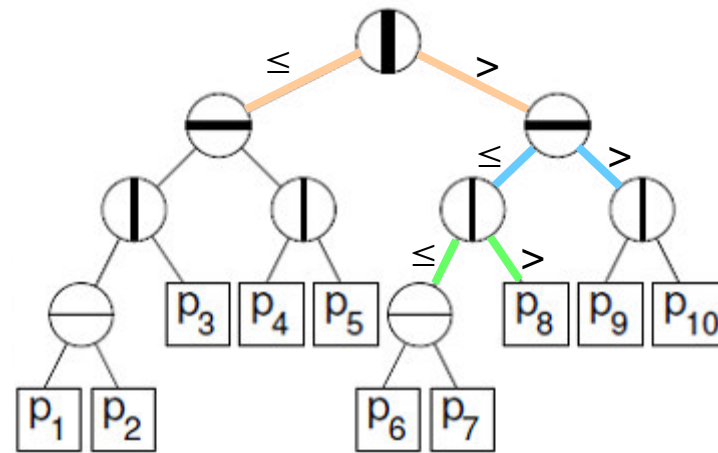
Tree structure   [Mount]

Kd-tree construction in 2-dimensions

BuildKdTree(*P, depth*)
Input: A set of points *P* and current *depth.*
Output: The root of a kD tree storing P.

1. **If (** *P* contains only one point)   [or small set of (10 to 20) points]
2.   **then return** a leaf storing this point
3.   **else if (** *depth* is even)   Split according to (*depth%max_dim)* dimension
4.   **then** split *P* with a vertical line *l* through median *x* into two subsets
                P$_1$ and P$_2$ (left and right from median)
5.   **else** split *P* with a horiz. line *l* through median y into two subsets
                P$_1$ and P$_2$ (below and above the median)
6. t_{left} = BuildKdTree(*P$_1$, depth+1*)
7. t_{right} = BuildKdTree(*P$_2$, depth+1*)
8.   create node *t* storing *l*, t_{left} and t_{right} children   // l = cutDim, cutVal
9.   **return** *t*

If median found in O(1) and array split in O(n)
$T(n) = 2\,T(n/2) + n$ => O(*n* log *n*) construction
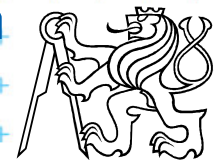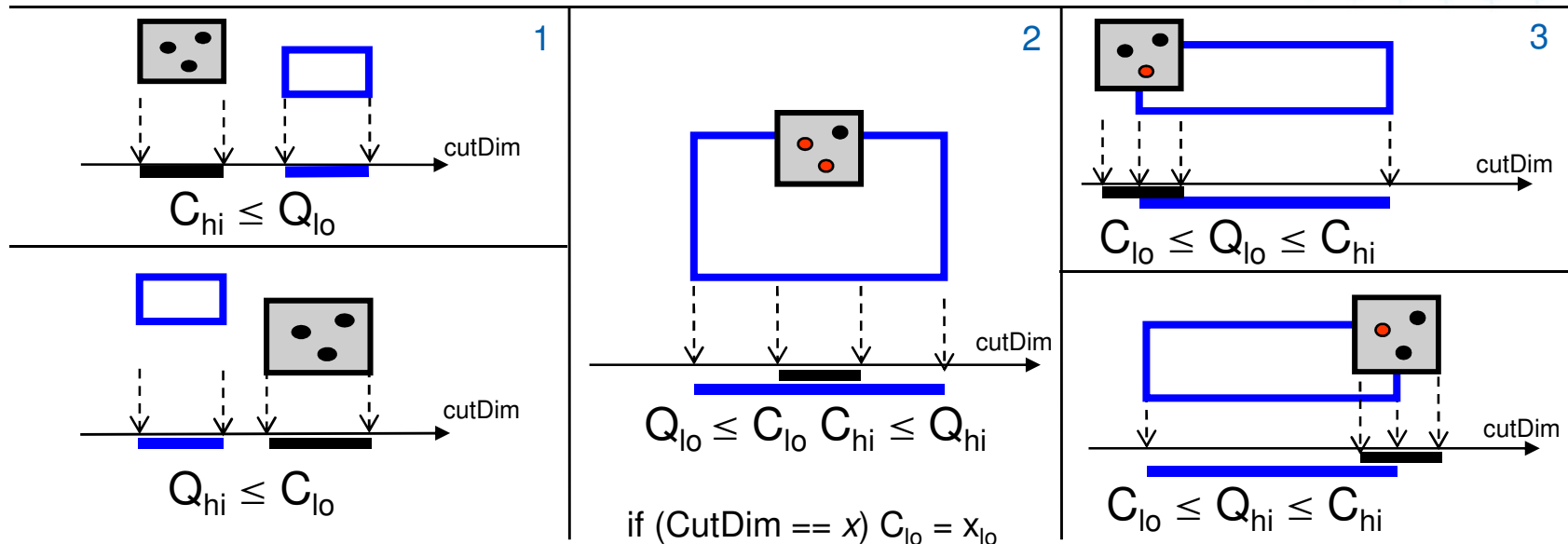
Kd-tree test variants

a) Compare rectang. array Q with rectangular cells C

- Rectangle $C:[x_{lo}, x_{hi}, y_{lo}, y_{hi}]$ – computed on the fly
- Test of kD node cell C against query Q (in one cutDim)
  1. if cell is disjoint with Q   … $C \cap Q = \emptyset$ … stop
  2. If cell C completely inside Q … $C \subseteq Q$ … stop and report cell points
  3. else cell C overlaps Q   … recurse on both children
- Recursion stops on the largest subtree (in/out)

1

$C_{hi} \leq Q_{lo}$

cutDim

$Q_{hi} \leq C_{lo}$

cutDim

2

cutDim

$Q_{lo} \leq C_{lo} \ C_{hi} \leq Q_{hi}$

if (CutDim == x) $C_{lo} = x_{lo}$

3

cutDim

$C_{lo} \leq Q_{lo} \leq C_{hi}$

cutDim

$C_{lo} \leq Q_{hi} \leq C_{hi}$

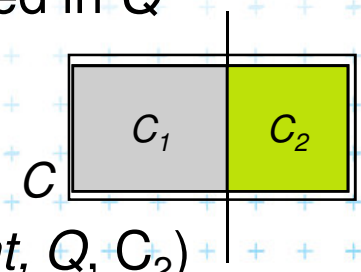# Kd-tree rangeCount (with rectangular cells)

int rangeCount(*t, Q, C*)
Input: The root *t* of kD tree, query range *Q and t's* cell C.
Output: Number of points at leaves below *t* that lie in the range.

1. **if (***t* is a leaf)
2.   **if** (*t.point* lies in *Q) return 1* ❶ // or loop this test for all points in leaf
3.   *else return 0*   // visited, not counted
4. **else** // (*t* is not a leaf)
5.   **if (**$C \cap Q = \varnothing$) return 0   ... *disjoint*
6.   **else if (**$C \subseteq Q$) return t.size ❹ ... *C is fully contained in Q*
7.   **else**
8.   split C along *t's* cutting value and dimension,
      creating two rectangles C_1 and C_2.
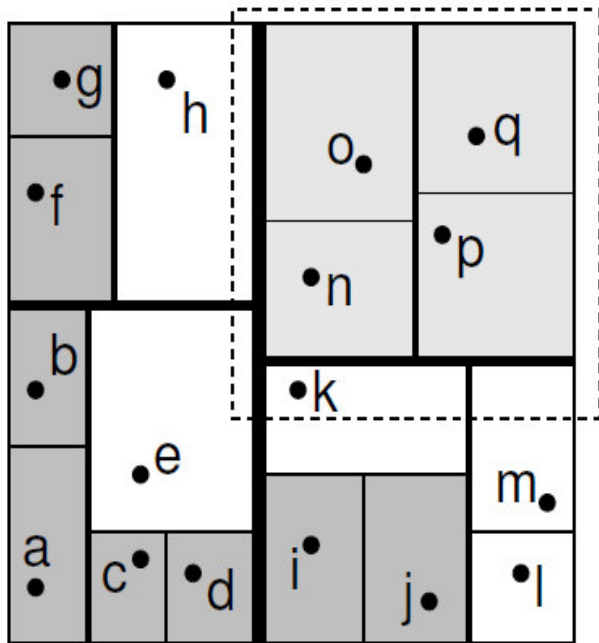9.   **return** rangeCount(*t.left, Q*, C_1) + rangeCount(*t.right, Q*, C_2)
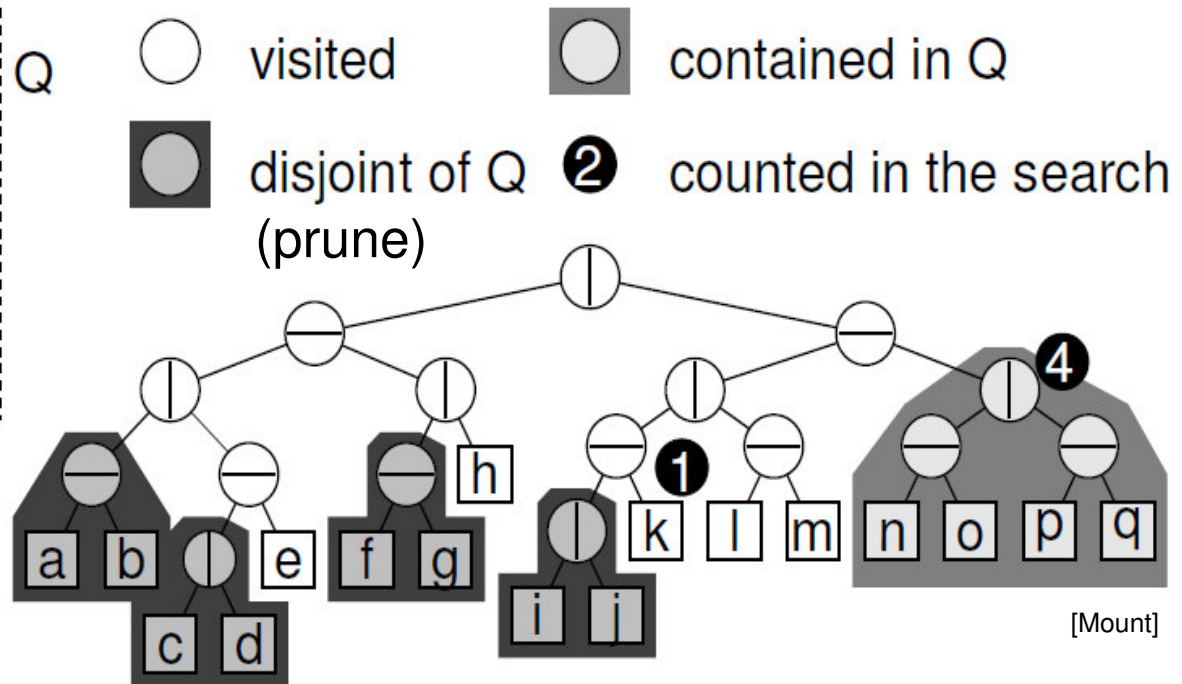
// (pictograms refer to the next slide)

Kd-tree rangeCount example

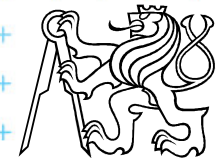Tree node (rectangular region)

○ visited   ◐ contained in Q

◕ disjoint of Q   ❷ counted in the search
(prune)

kd–tree subdivision

Nodes visited in range search

[Mount]

Kd-tree test variants

b) Compare Q with cutting lines

– Line = Splitting value *p* in one of the dimensions

– Test of single position given by dimension against Q

   1. Line is right from Q   … recurse on left child only (prune right child)
   2. Line is left from Q   … recurse on right child only (prune left ch.)
   3. Line intersects Q   … recurse on both children

– Recursion stops in leaves - traverses the whole tree

[Havran]

position   position   position

$Q_{hi} \leq p$ $Q_{lo} \leq p \leq Q_{hi}$ $Q_{lo} \leq p$
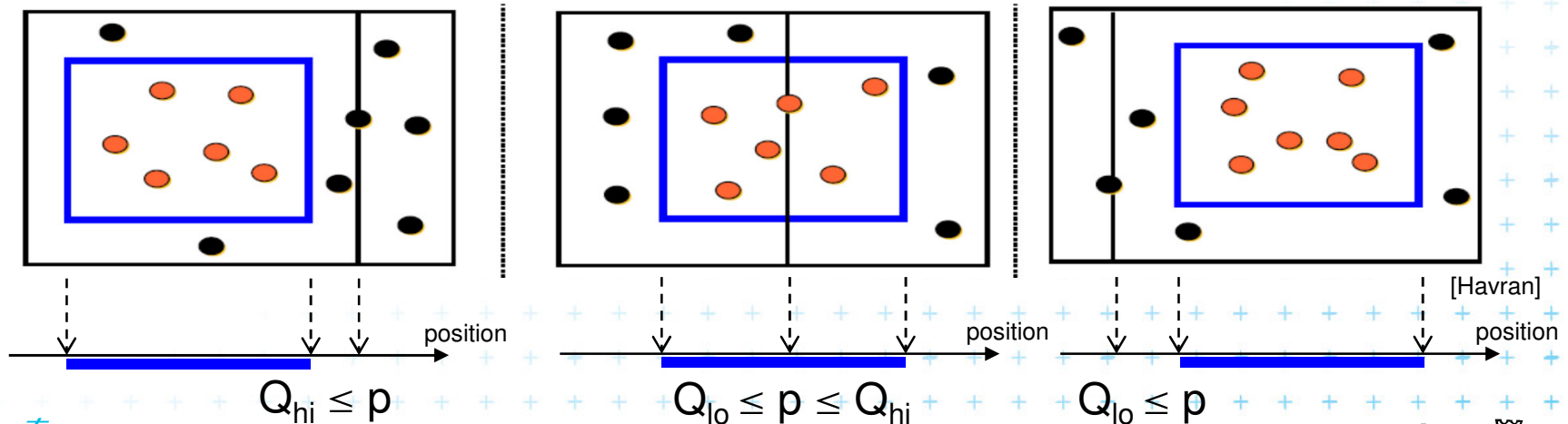
Kd-tree rangeSearch (with cutting lines)

Input:
Output:

int rangeSearch(t, Q)
The root t of (a subtree of a) kD tree and query range Q.
Points at leaves below t that lie in the range.

1. **if (** t is a leaf)
2.   **if** (*t.point* lies in *Q*) *report t.point*   // or loop test for all points in leaf
3.   *else return*
4. **else (** t is not a leaf)
5.   **if (Q$_{hi}$** $\leq$ *t.cutVal)* rangeSearch(*t.left*, *Q*)   // go left only
6.   **if (Q$_{lo}$** $>$ *t.cutVal)* rangeSearch(*t.right*, *Q*) // go right only
7.   **else**
8.   rangeSearch(*t.left*, *Q*)   // go to both
9.   rangeSearch(*t.right*, *Q*)

Kd-tree - summary

- **Orthogonal range queries in the plane (in balanced 2d-tree)**

  - Counting queries $O(\sqrt{n})$ time

  - Reporting queries $O(\sqrt{n} + k)$ time,
    where k = No. of reported points

  - Space $O(n)$

  - Preprocessing: Construction $O(n \log n)$ time
    (Proof: if presorted points to arrays in dimensions. Median in $O(1)$
    and split in $O(n)$ per level, log n levels of the tree)

- **For d≥2:**

  - Construction $O(d\, n \log n)$, space $O(dn)$, Search $O(d\, n^{(1-1/d)} + k)$

Orthogonal range tree (RT)

- DS highly tuned for orthogonal range queries
- Query times in plane

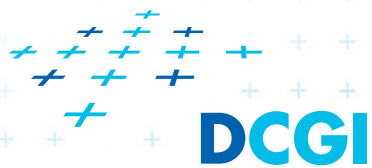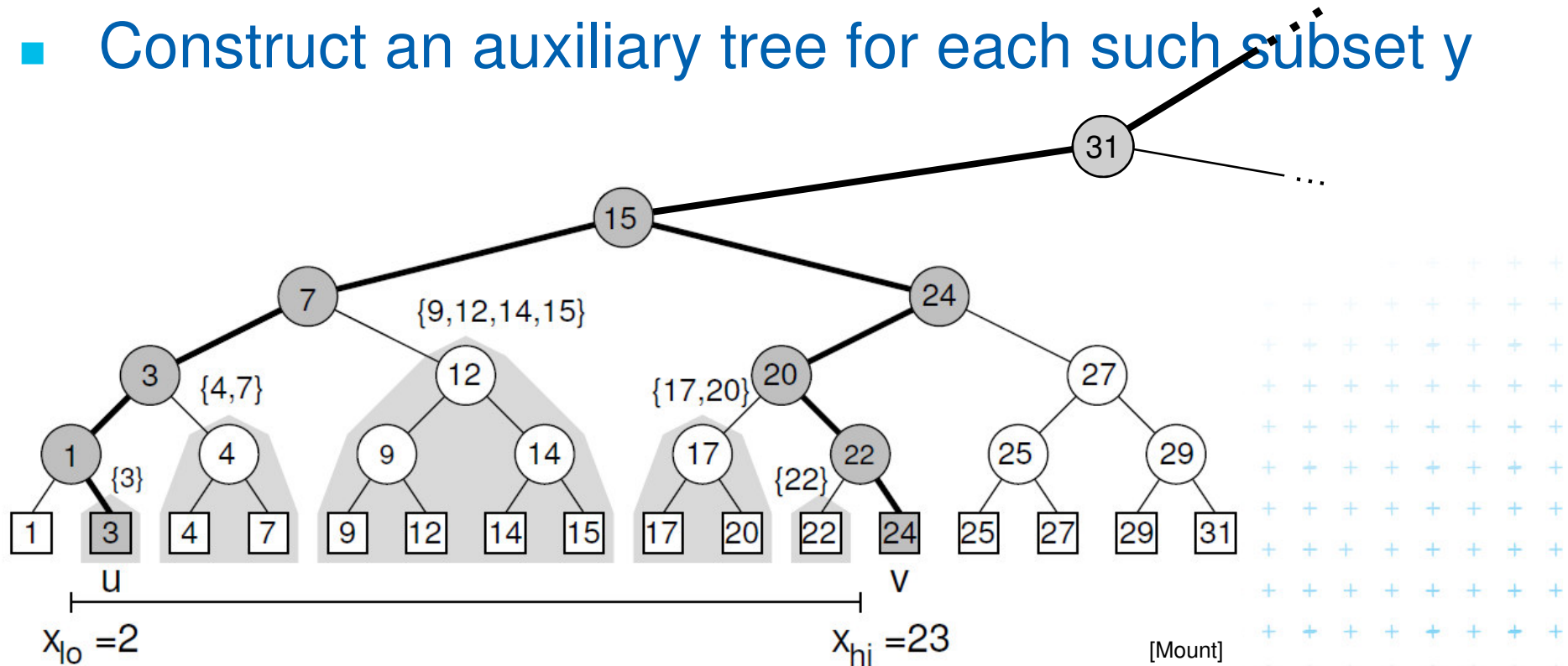| 2d tree versus	range tree
$O(\sqrt{n} + k)$ time of Kd >	$O(\log n)$ time query
$O(n)$ space of Kd <	$O(n \log n)$ space

n = number of points

k = number of reported points

DCGI
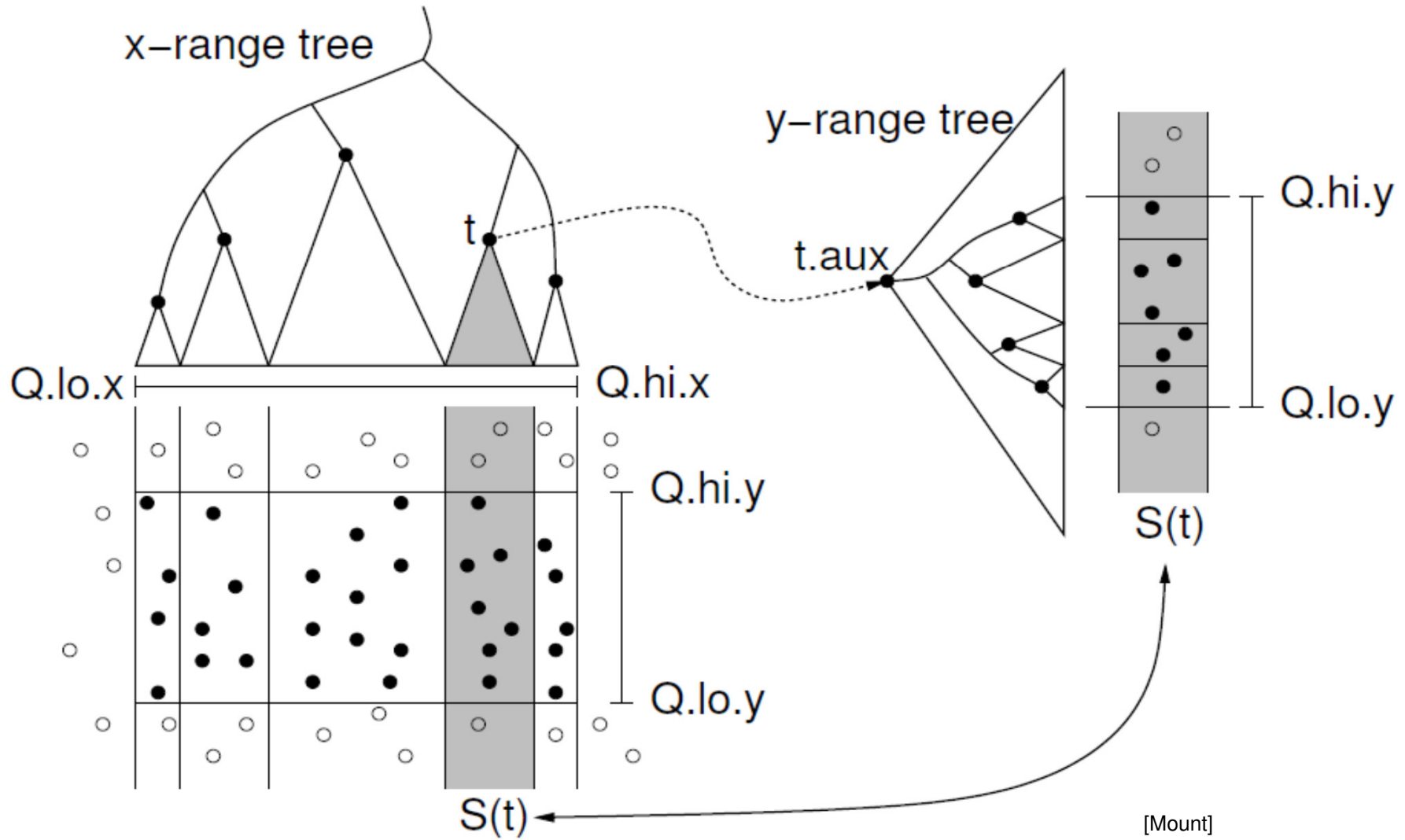
From 1D to 2D range tree

- 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 auxiliary tree for each such subset y

[Mount]

DCGI

2D range tree

x–range tree

y–range tree

t

t.aux

Q.lo.x

Q.hi.x

Q.hi.y

Q.lo.y

Q.hi.y

Q.lo.y

S(t)

S(t)

[Mount]

DCGI

2D range search

2dRangeQuery(t, [x:x'] $\times$ [y:y'])
Input: 2d range tree t and Query range
*Output:*   All points in t laying in the range
1. t_{split} = FindSplitNode(t, x, x')
2.   if(t_{split} is leaf)
3.   check if the point in t_{split} must be reported   … t.x $\in$ [x:x'], t.y $\in$ [y:y']
4.   else // follow the path to x, calling 1dRangeQuery on y
5. $t = t_{split}$.left   // path to the left
6.   while(t is not a leaf)
7.   if($x \leq$ t.x)
8.   1dRangeQuerry(t_{assoc}($t.right$), [$y:y'$]) // check associated subtree
9. $t = t.left$
10.   else $t = t.right$
11. check if the point in leaf t must be reported   … t.x $\leq$ x', t.y $\in$ [y:y']
12. Similarly for the path to x'   … // path to the right
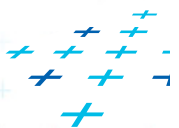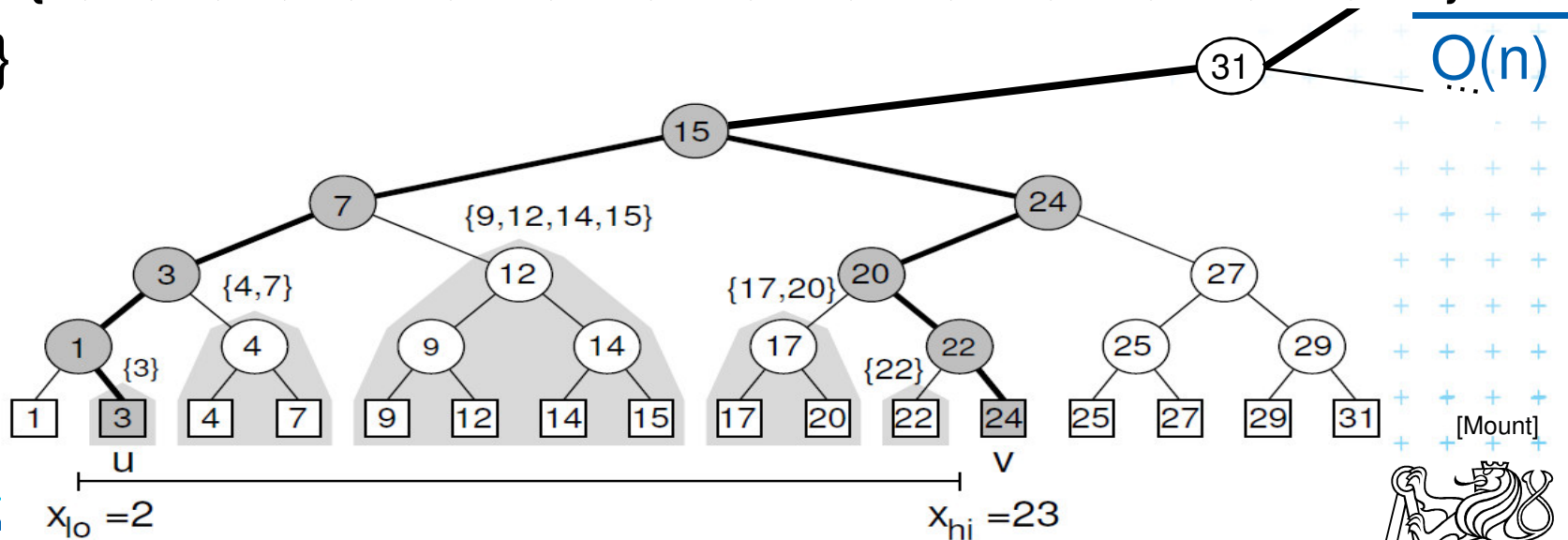
DCGI

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 - $O(n \log n)$

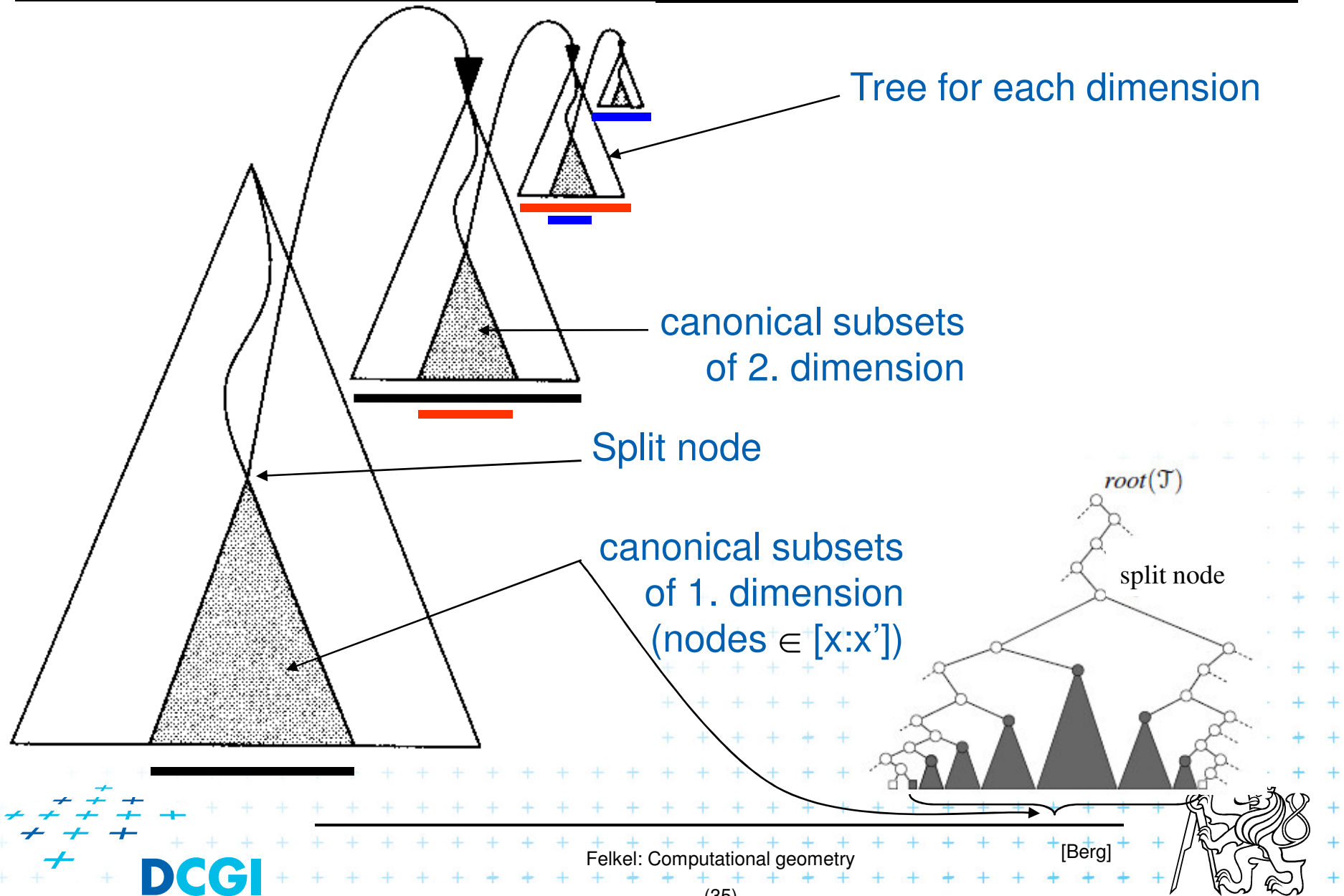 - Sort points (by *x* and by *y*). Bottom up construction

DCGI
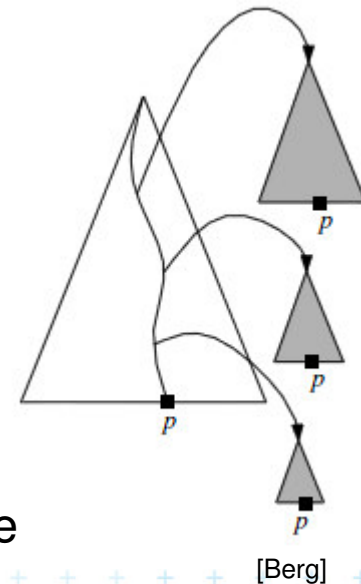
Canonical subsets

- 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]

$x_{lo} = 2$ $x_{hi} = 23$

nD range tree (multilevel search tree)

Tree for each dimension

canonical subsets
of 2. dimension

Split node

canonical subsets
of 1. dimension
(nodes $\in [x:x']$)

$root(\mathcal{T})$

split node

[Berg]

DCGI

Fractional cascading - principle

- Two sets S_1, S_2 stored in sorted arrays A_1, A_2

- Report objects in both whose keys in [y:y']

- Naïve approach
  - $O(\log n_1 + k_1)$ – search in A_1 + report k_1 elements
  - $O(\log n_2 + k_2)$ – search in A_2 + report k_2 elements

- Fractional cascading – adds pointers from A_1 to A_2
  - $O(\log n_1 + k_1 + 1 + k_2)$ – search in A_1 + report k_1 elements
  - $O(1 + k_2)$ – jump to A_2 + report k_2 elements
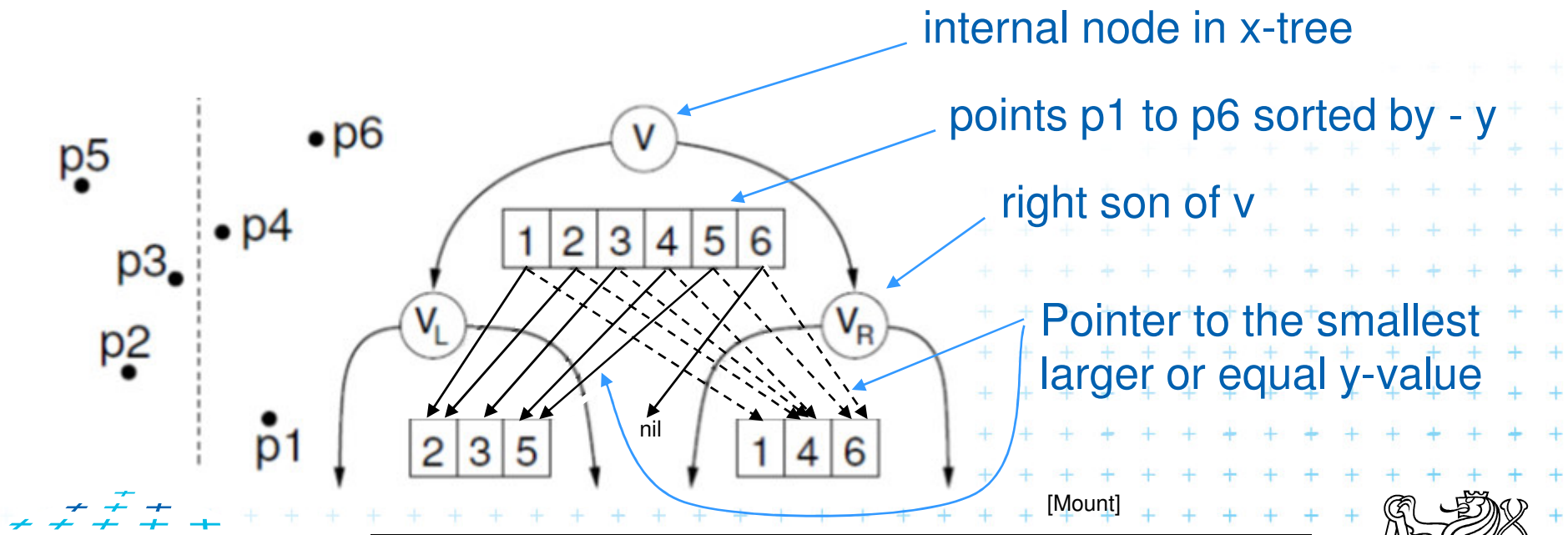 - Saves the $O(\log n_2)$ – search

Fractional cascading – principle for arrays

- Add pointers from A_1 to A_2
  - From element in A_1 with a key y_i point to the element in A_2 with the smallest key larger or equal to y_i

- Example query with the range [20 : 65]

[Berg]
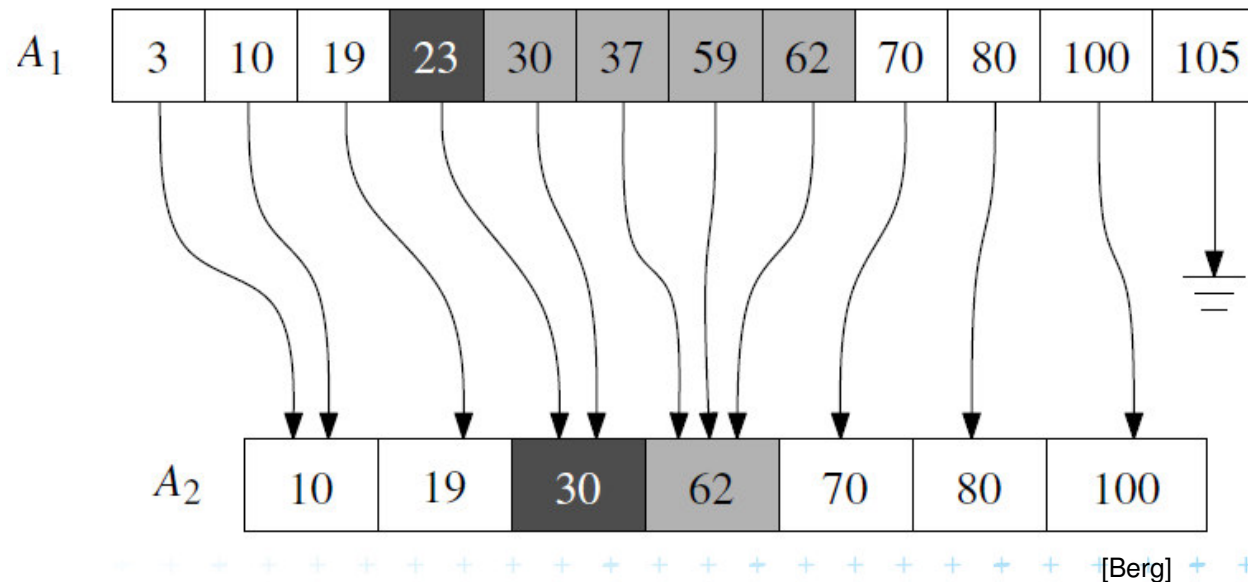
Fractional cascading in the 2D range tree

- How to save one log n during last dim. search?
  - Store canonical subsets in arrays sorted by y
  - Pointers to subsets for both child nodes v_L and v_R
  - O(1) search in lower levels => in two dimensional search O($\log^2 n$) time -> O(2 log n)

internal node in x-tree

points p1 to p6 sorted by - y

right son of v

Pointer to the smallest larger or equal y-value

p5

p6

p4

p3

p2

p1

v

1 2 3 4 5 6

V_L V_R

2 3 5   nil   1 4 6

[Mount]

DCGI

Orthogonal range tree - summary

- **Orthogonal range queries in plane**
  - Counting queries $O(\log^2 n)$ time,
    or with fractional cascading $O(\log n)$ time
  - Reporting queries plus $O(k)$ time, for k reported points
  - Space $O(n \log n)$
  - Construction $O(n \log n)$

- **Orthogonal range queries in d-dimensions, $d \geq 2$**
  - Counting queries $O(\log^d n)$ time,
    or with fractional cascading $O(\log^{(d-1)} n)$ time
  - Reporting queries plus $O(k)$ time, for k reported points
  - Space $O(n \log^{(d-1)} n)$
  - Construction $O(n \log^{(d-1)} n)$ time

References

- **[Berg] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars: Computational Geometry: *Algorithms and Applications*, Springer-Verlag, 3rd rev. ed. 2008. 386 pages, 370 fig. ISBN: 978-3-540-77973-5, Chapter 5, http://www.cs.uu.nl/geobook/**

- **[Mount] David Mount, - CMSC 754: Computational Geometry, Lecture Notes for Spring 2007, University of Maryland , Lectures 17 and 18. http://www.cs.umd.edu/class/spring2007/cmsc754/lectures.shtml**

- **[Havran] Vlastimil Havran, Materiály k předmětu Datové struktury pro počítačovou grafiku, přednáška č. 6, Proximity search and its Applications 1, CTU FEL, 2007**

DCGI