matfyz

Computer
Graphics
Charles
University

Spatial data structures in 2D

© 1998-2016 Josef Pelikán
CGG MFF UK Praha

pepca@cgg.mff.cuni.cz
http://cgg.mff.cuni.cz/~pepca/

Application areas

- **geographic information systems** (GIS)
  - – area, line and point entities
  - – huge databases (**10^4** to **10^9** objects)
- **image analysis, recognition**
- **computer graphics, games**
  - – 2D & 3D algorithm speedup (ray casting, collisions)
- **industry, CAD**
  - – VLSI design, component positioning, collisions

Elementary tasks I

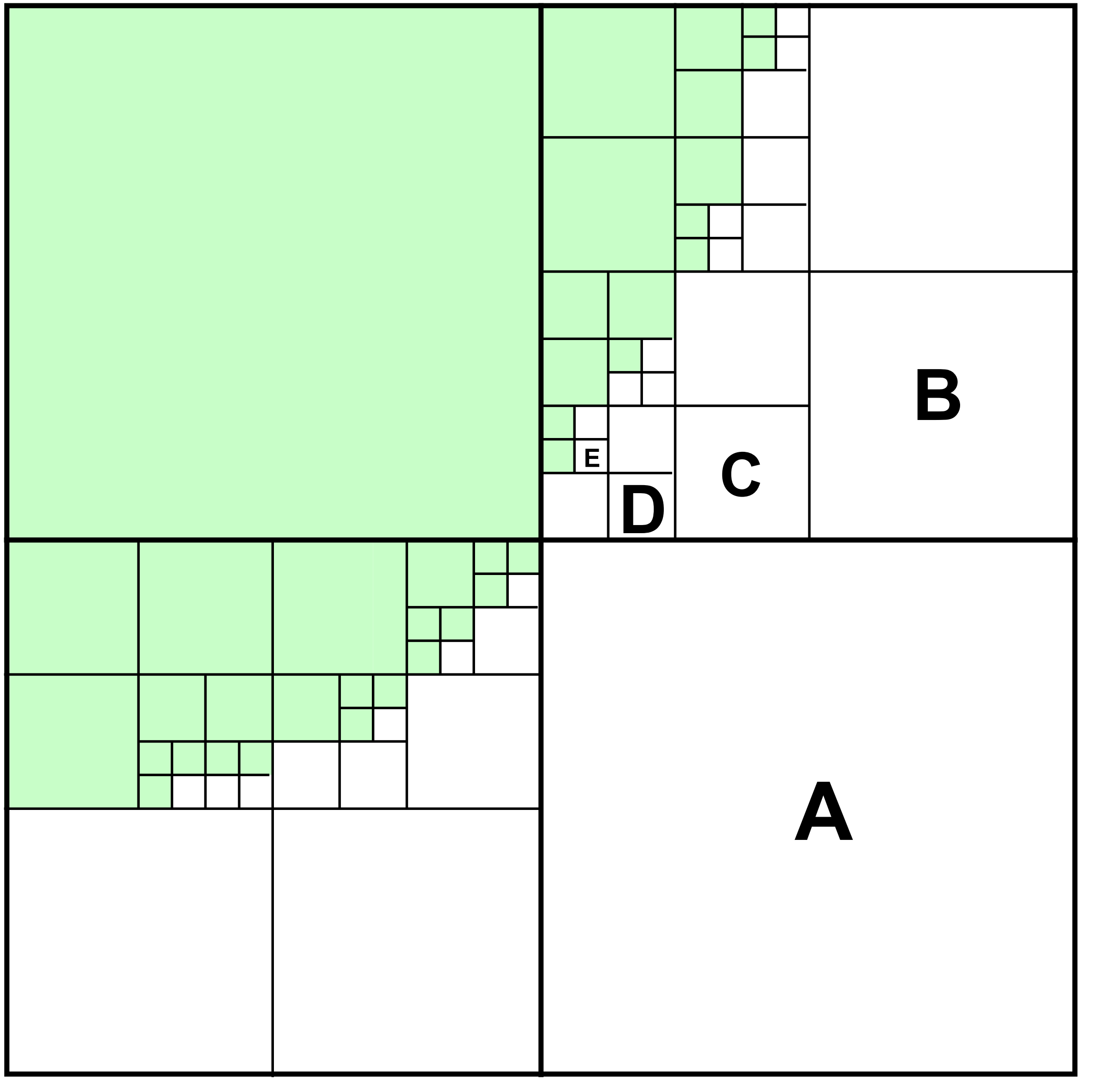
- **point localization** in 2D net
  – looking for an area object which contains the point
- **nearest N points** from the given center
  – global variation: looking for the closest point pair
- **curve intersections** (polylines)
  – collisions in a set of curves (polylines)
- **point objects closest to the given curve** (route)

Elementary tasks II

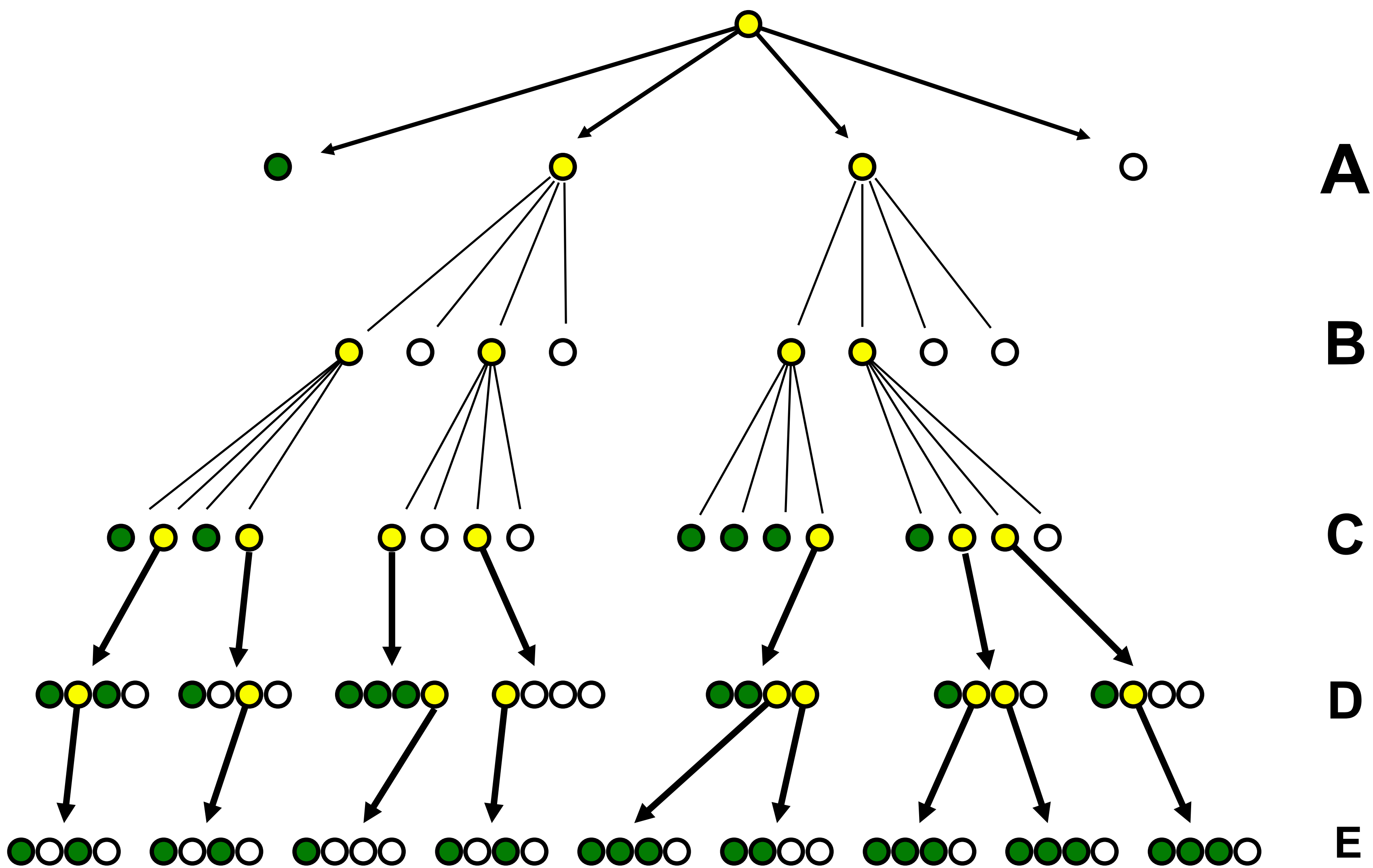
- **interval queries** in 2D space (databases)
- **set operations** on map entities
  – areas, line objects, points
- **collision tests (interferences)**, minimal distances among planar objects (VLSI)
- object processing in some **geometric order**
  - increasing distance from a given center

“Region quadtree”

- **representation of <u>areal regions</u> in 2D**
- **splitting exactly <u>in the middle</u>**
- **information is stored <u>in leaf nodes only</u>**

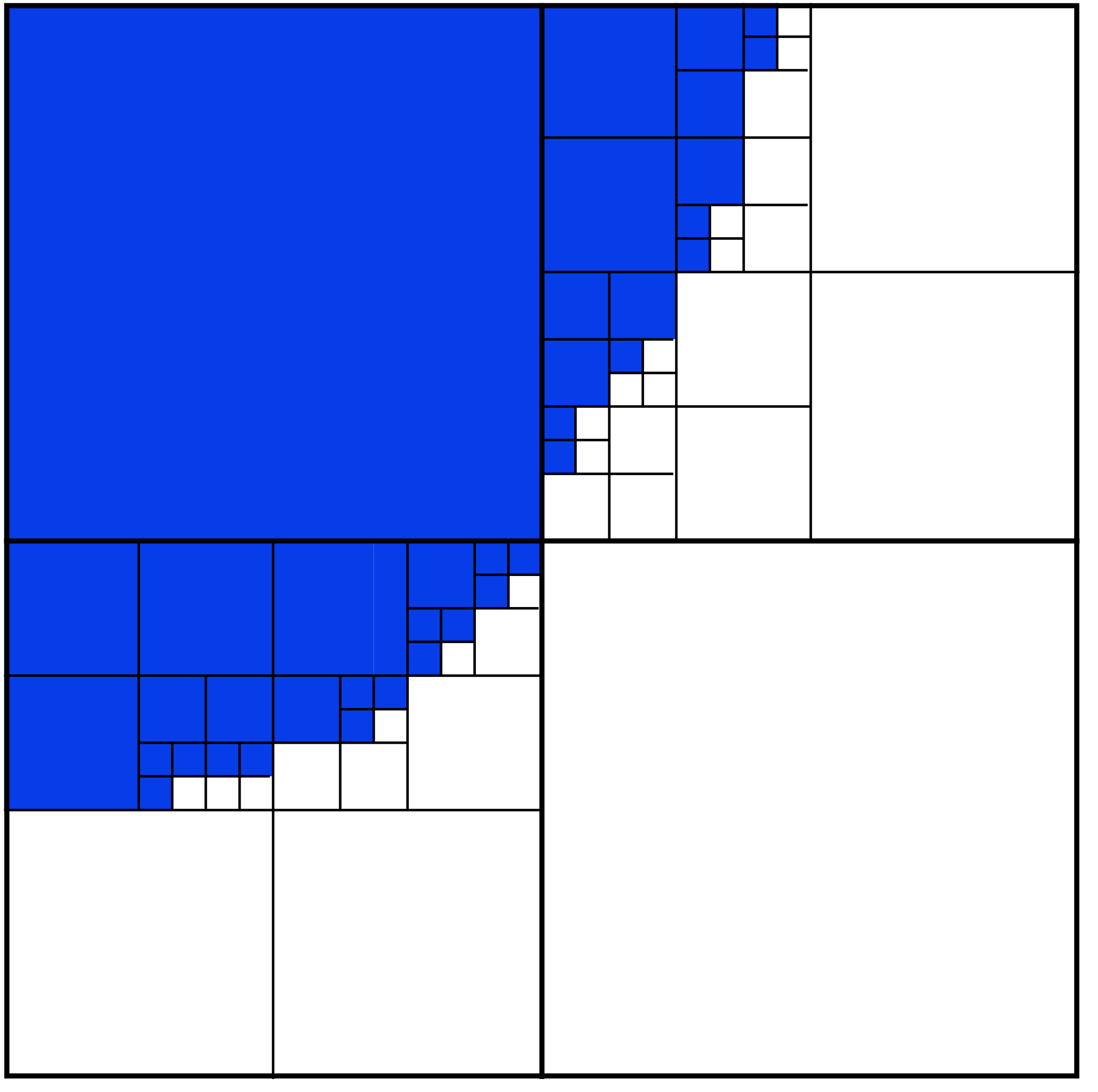


“Region quadtree”



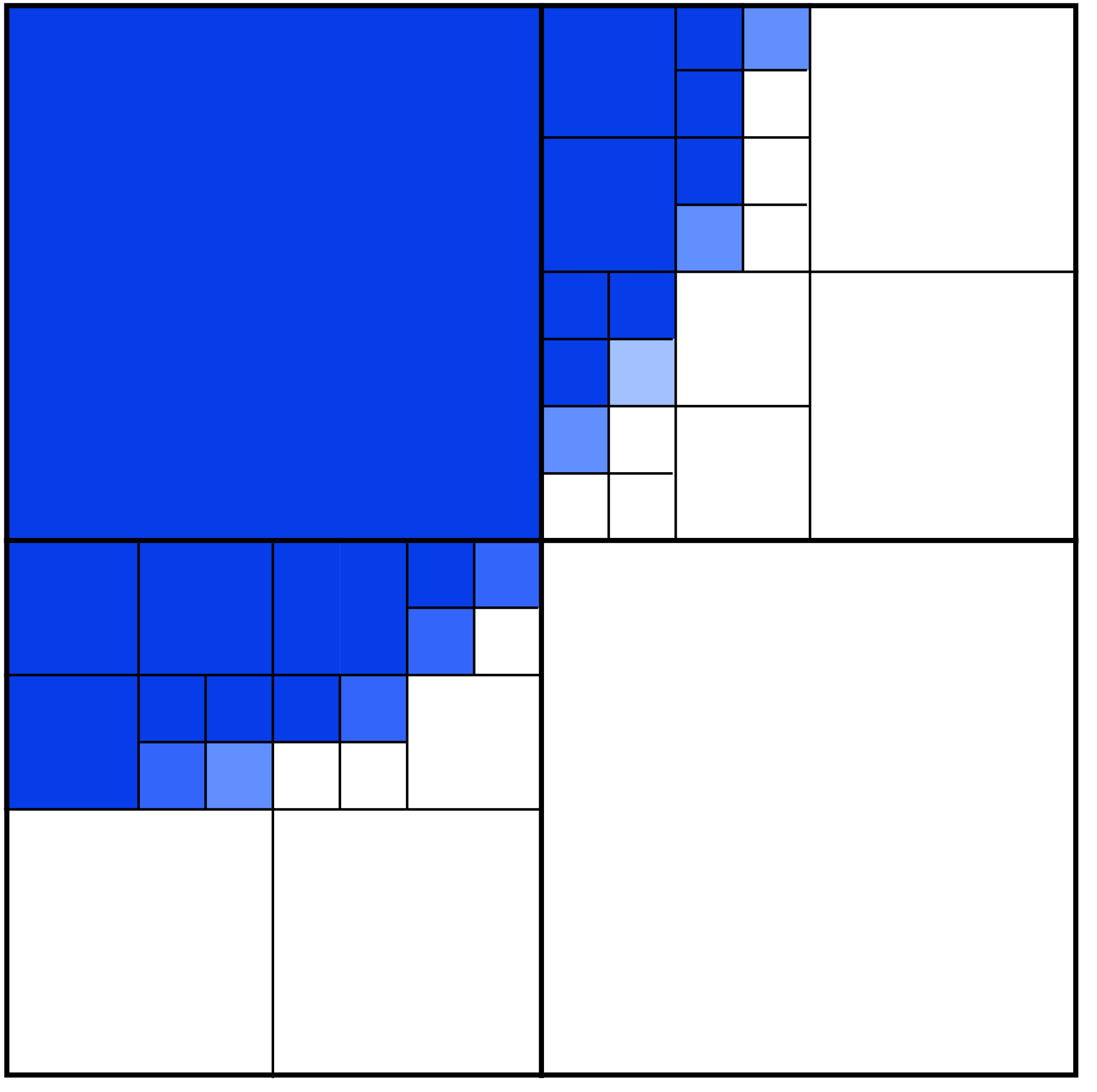
Pyramid

- representation of **<u>areal regions</u>** in 2D
- additional **<u>summary info in inner nodes</u>**
- pyramid nodes up to level: **E**



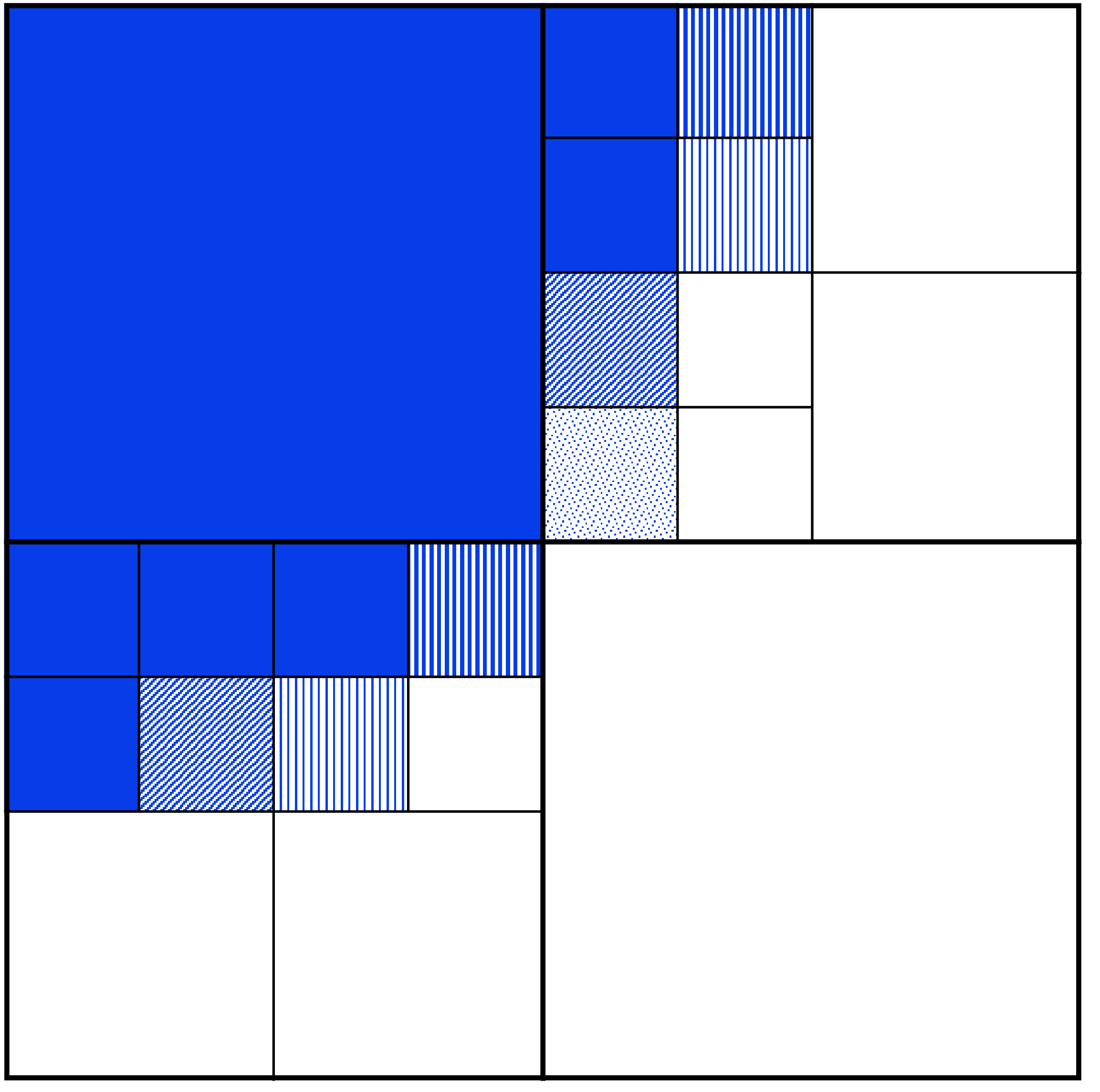
Pyramid

- **pyramid nodes up to level: D**



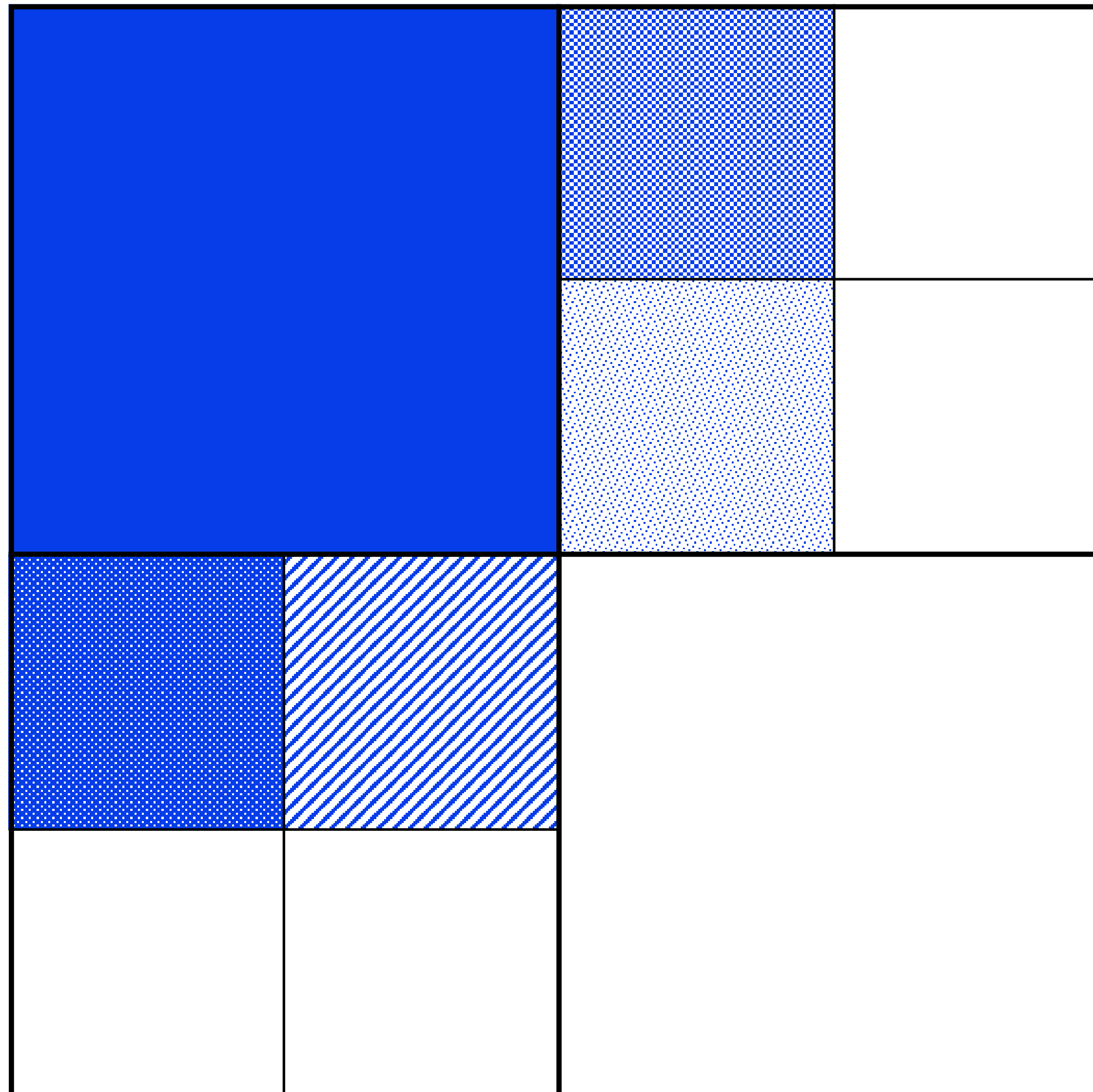
Pyramid

- **pyramid nodes up to level: C**



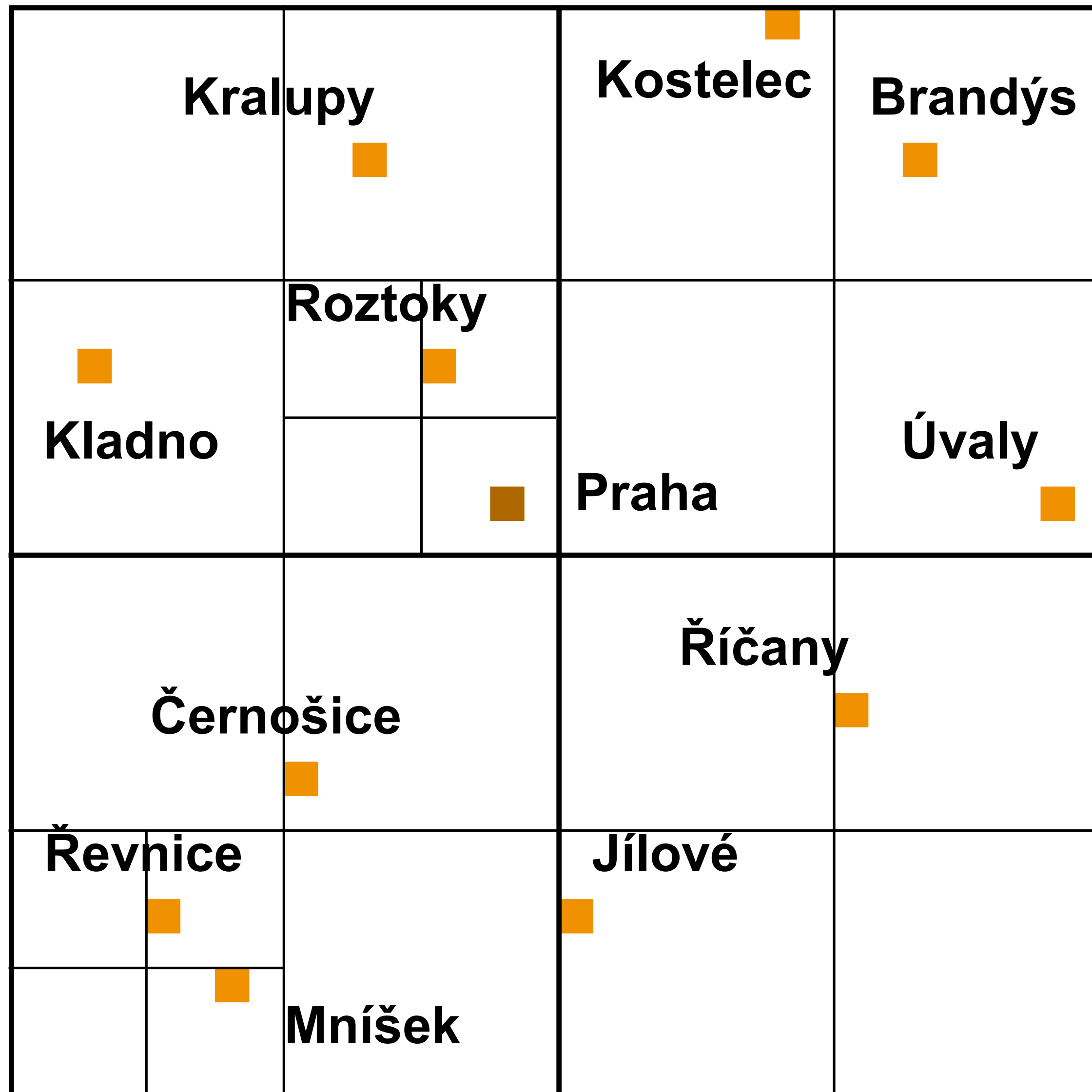
Pyramid

- **pyramid nodes up to level: B**

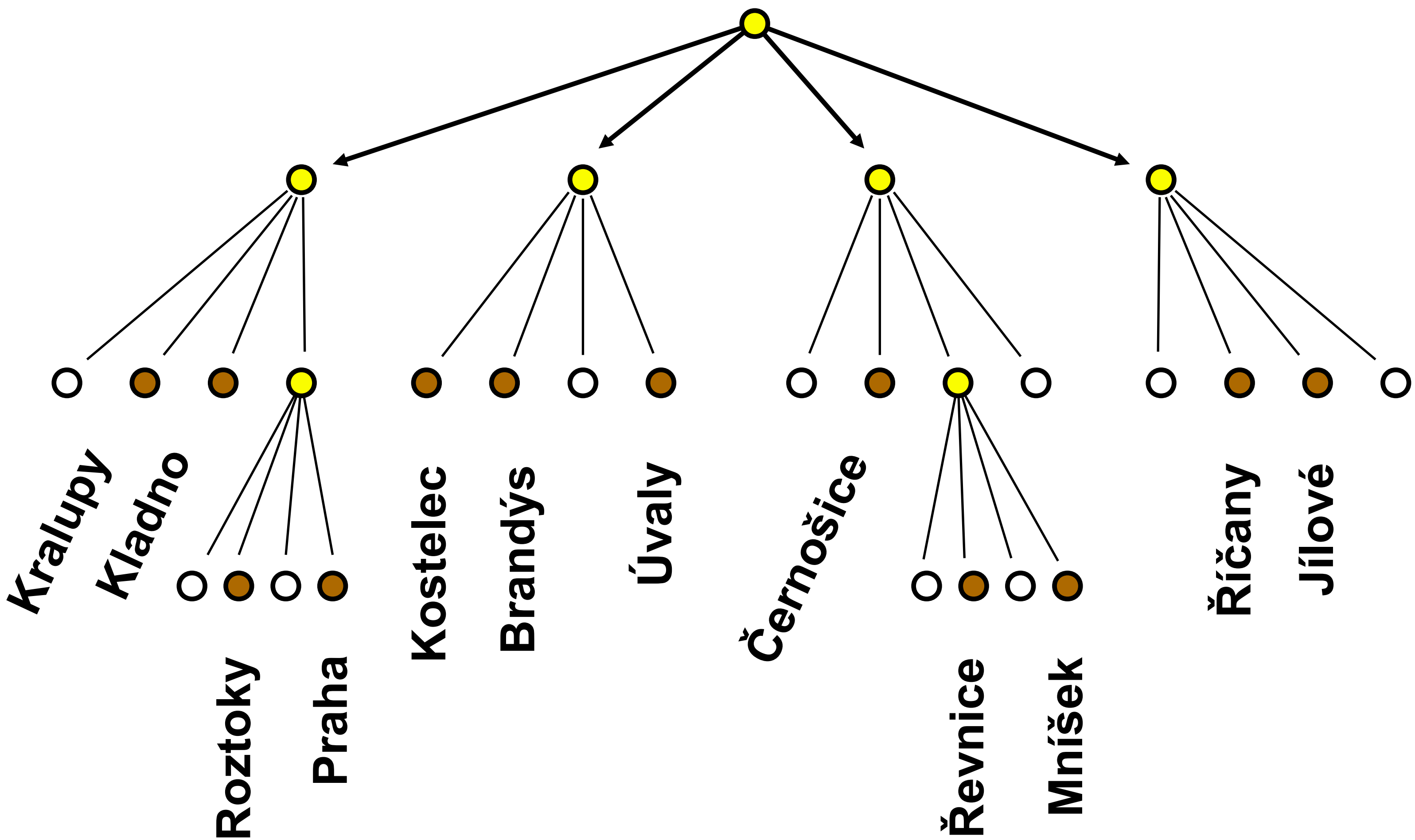


“PR quadtree” (Point-Region)

- representation <u>of point objects</u>
- splitting exactly <u>in the middle</u>
- object information stored <u>in leaf nodes</u> (<u>all levels, max. one object per node</u>)



"PR quadtree" (Orenstein)



Bucket methods

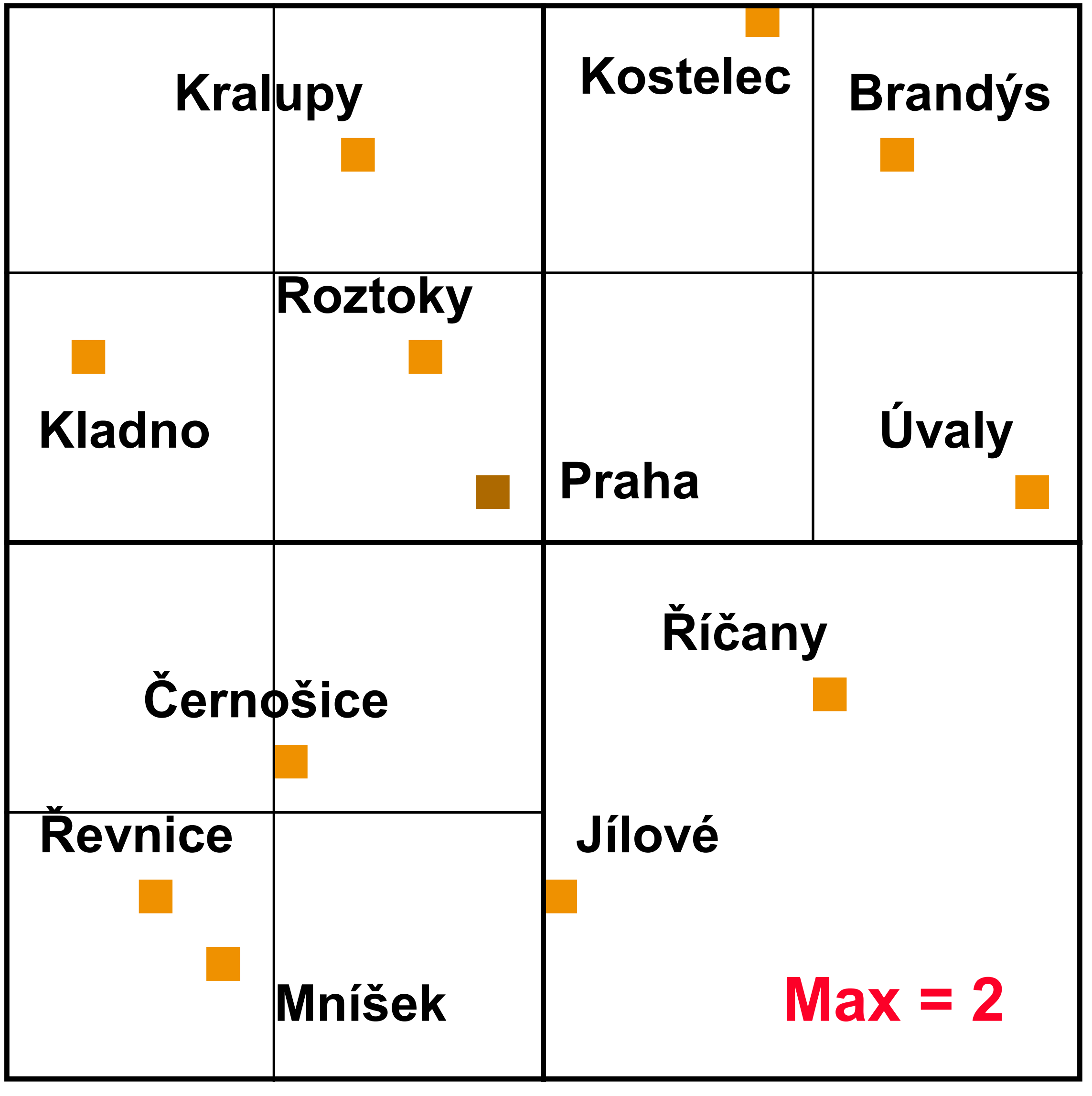
- node can store information about **more than one object**

$$0 < \text{object\#} \leq \text{Max}$$

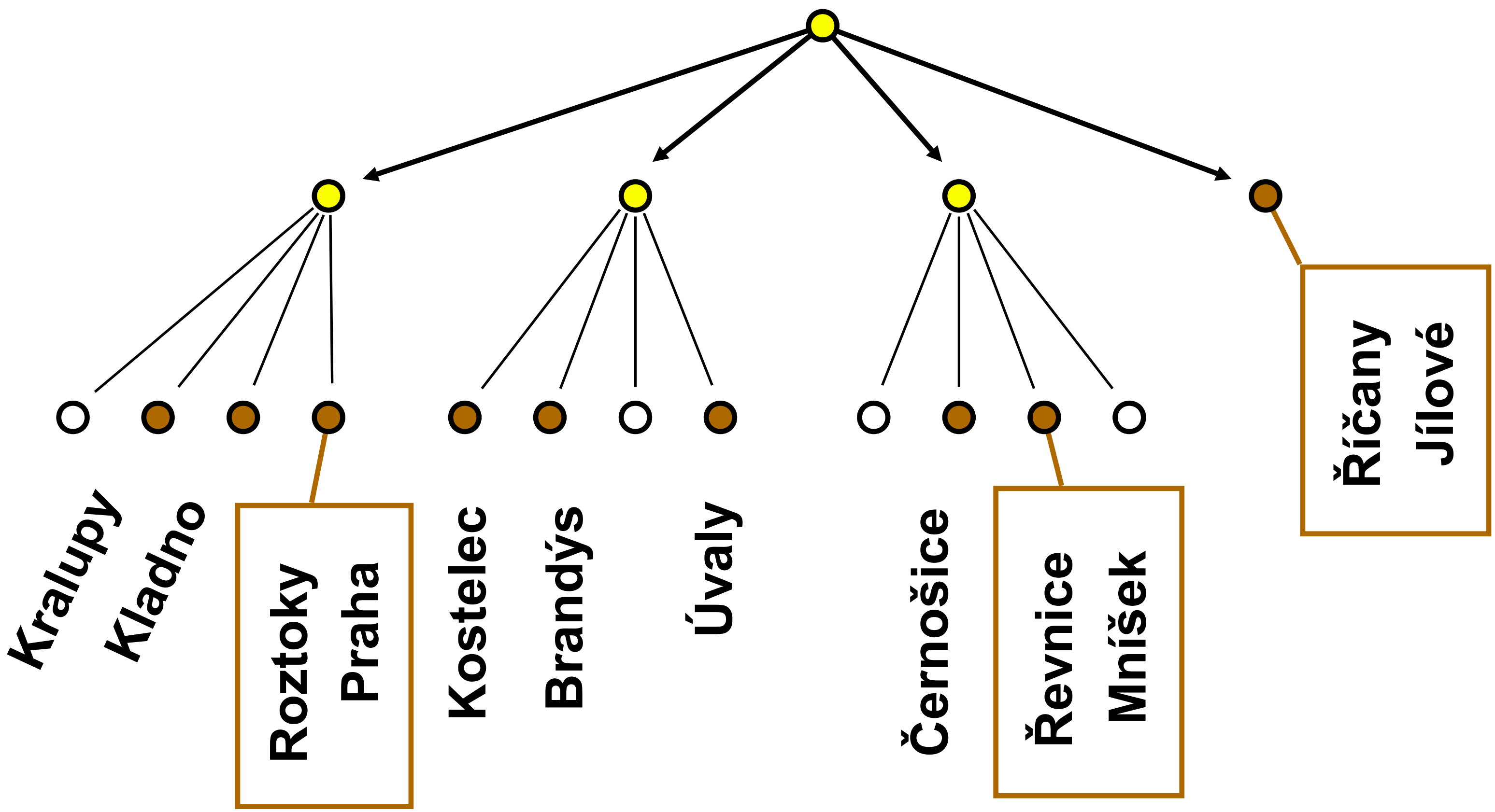
  - **Max** is constant value tuned for storage efficiency (record size..)
- **memory & time** efficiency
  - less overhead in pointer management
- **AB-tree** analogy

"Bucket PR quadtree"

- representation **of point objects**
- splitting exactly **in the middle**
- object information stored **in leaf nodes** (**all levels, ≤ Max object per node**)



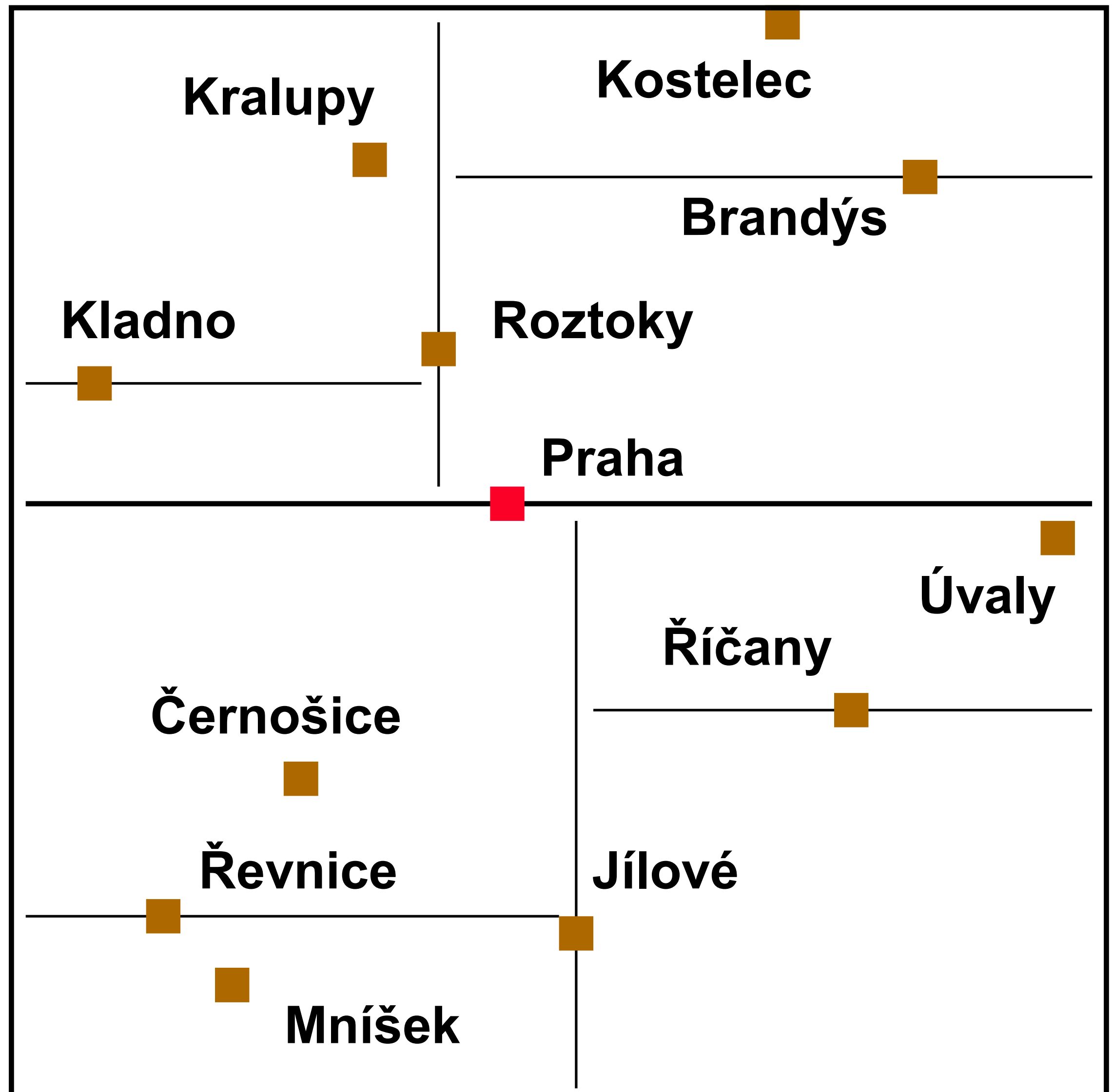
“Bucket PR quadtree”



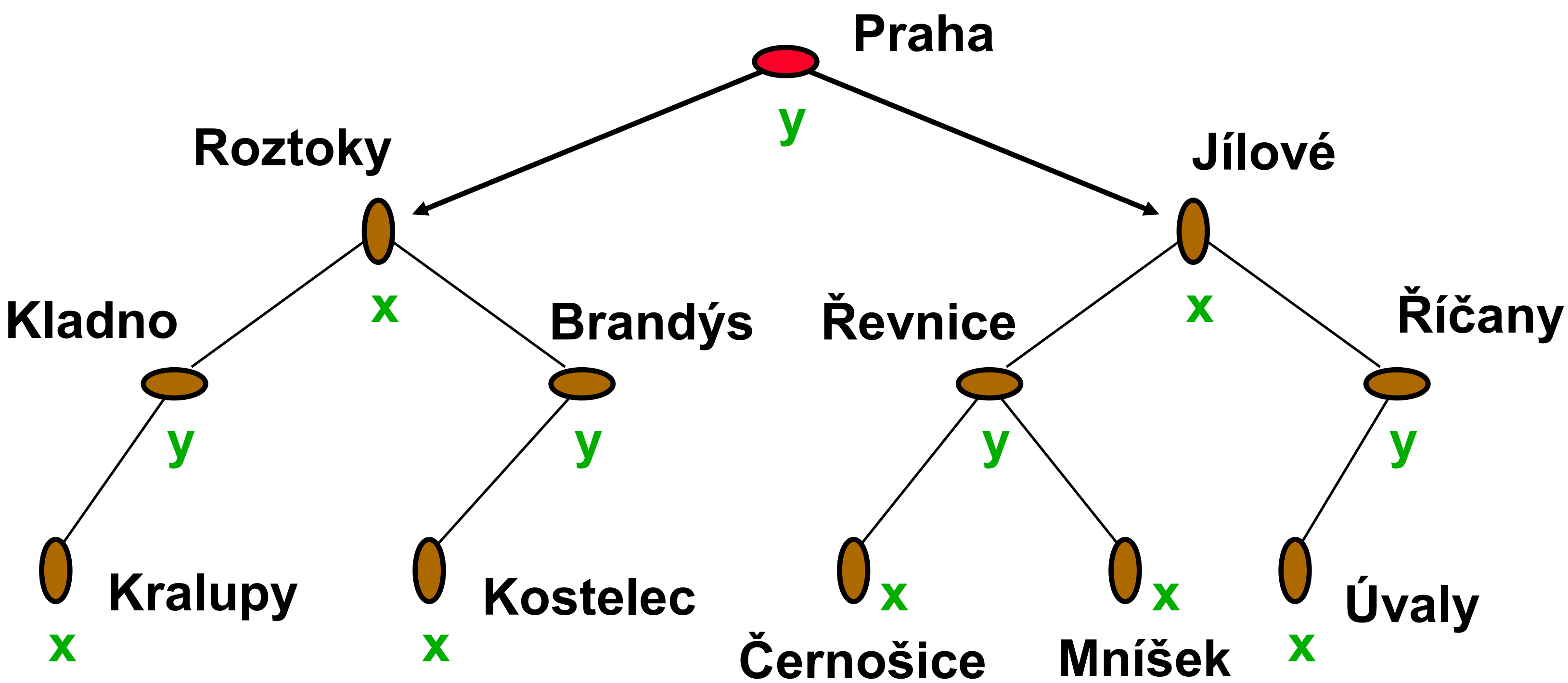
Max = 2

“K-D tree” (Bentley)

- representation **of point objects**
- adaptive splitting - **one coordinate at a time (binary tree)**. regular alternation of the coordinates
- object information stored in **all nodes**

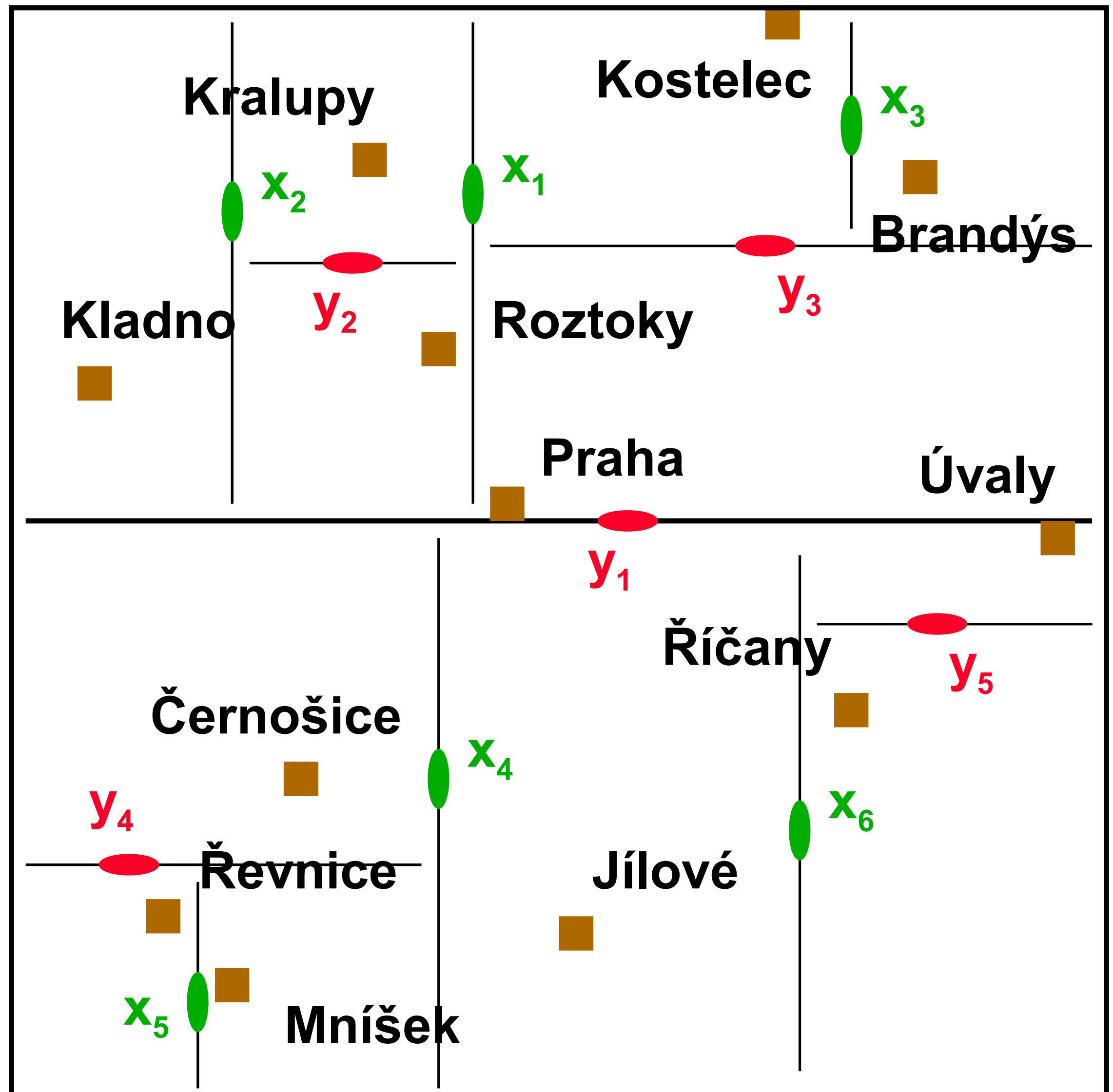


"K-D tree"

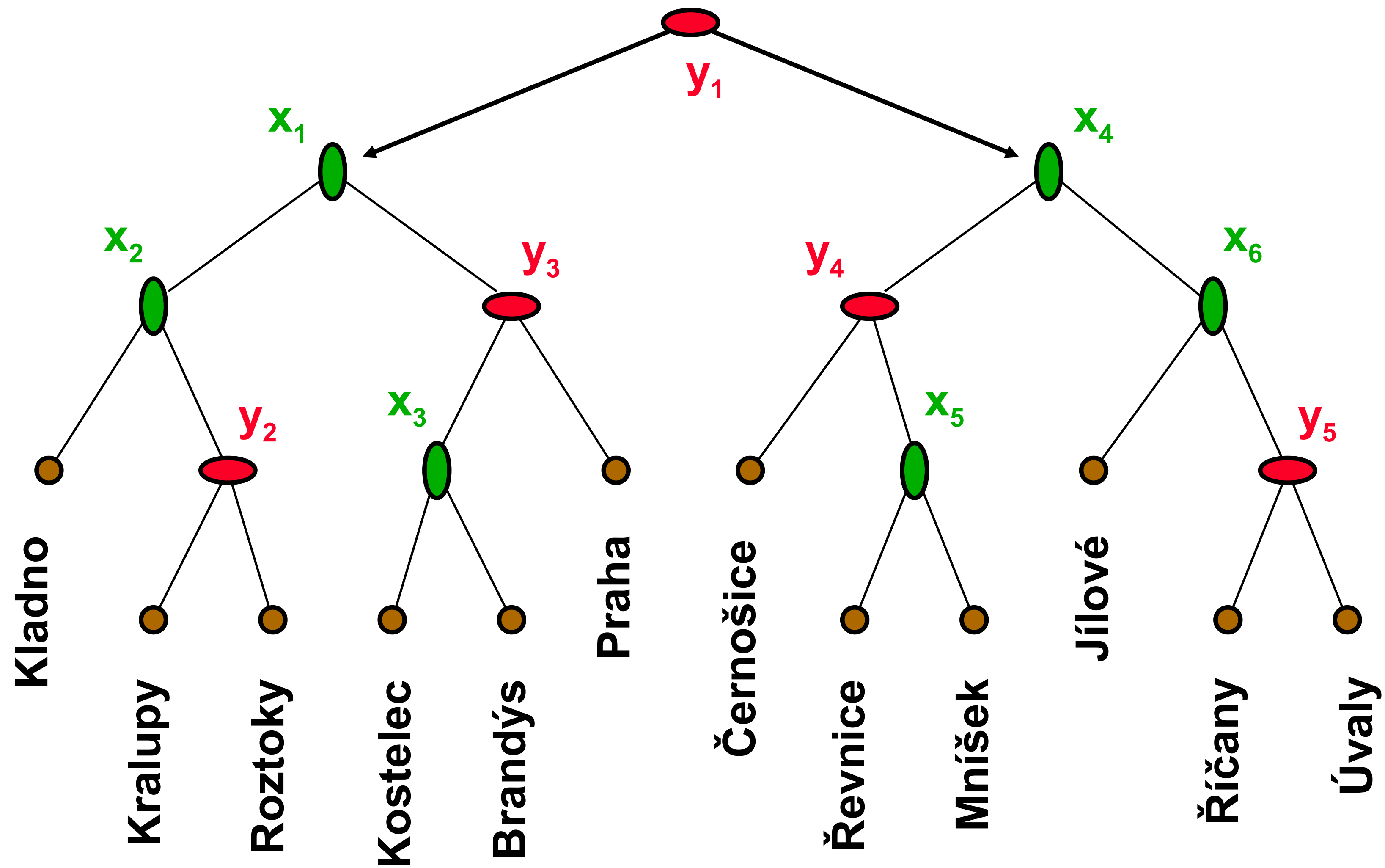

Praha
y
Roztoky
Jílové
Kladno
x
Brandýs
Řevnice
x
Říčany
y
y
y
y
Kralupy
Kostelec
x
x
Úvaly
x
x
x
Černošice
Mníšek

"Adaptive K-D tree" (Friedman)

- representation **of point objects**
- adaptive splitting - **one coordinate at a time (binary tree)**. Exact object coordinates not used
- object information stored **in leafs only**



“Adaptive K-D tree”

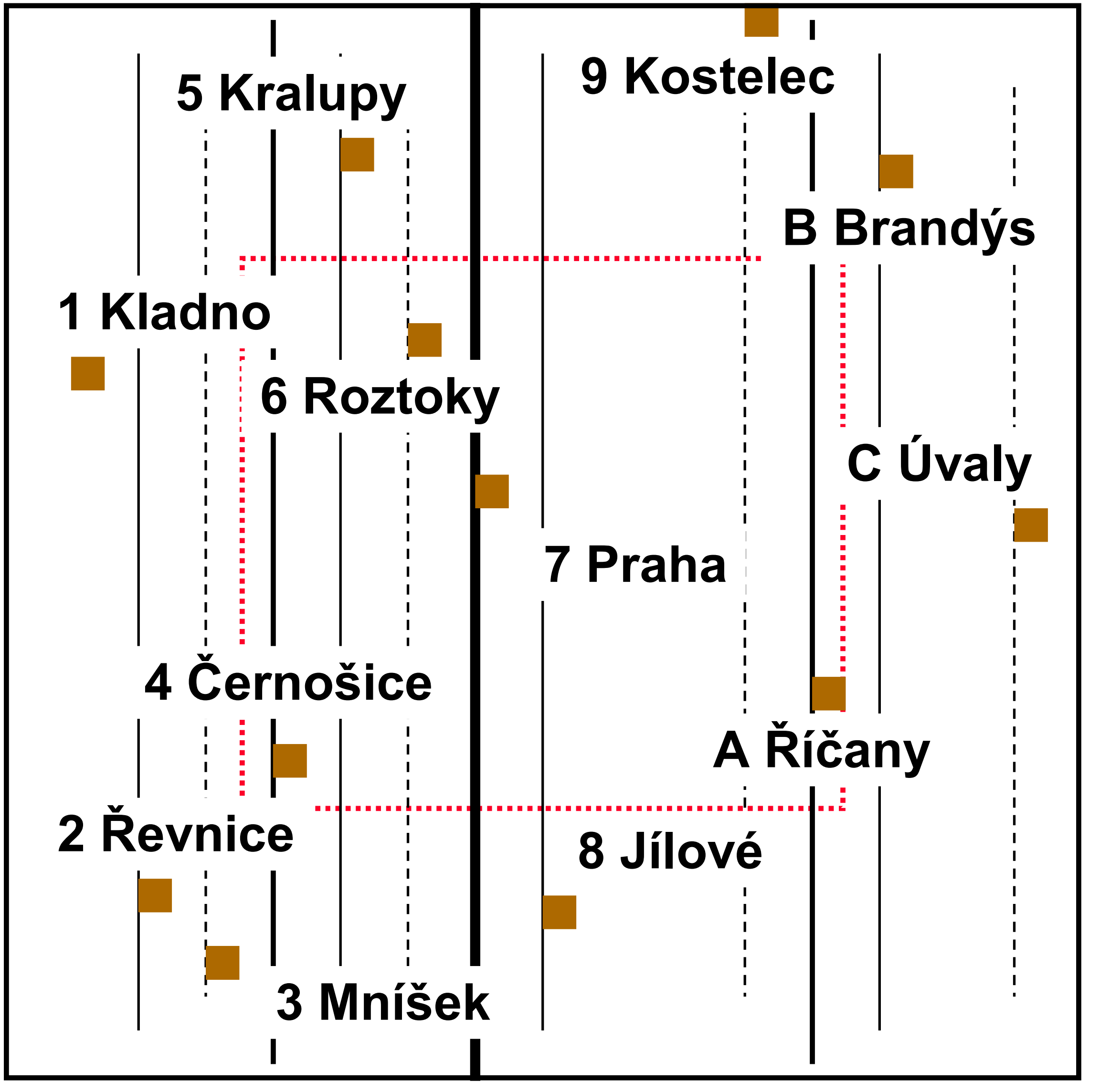


“Range trees”

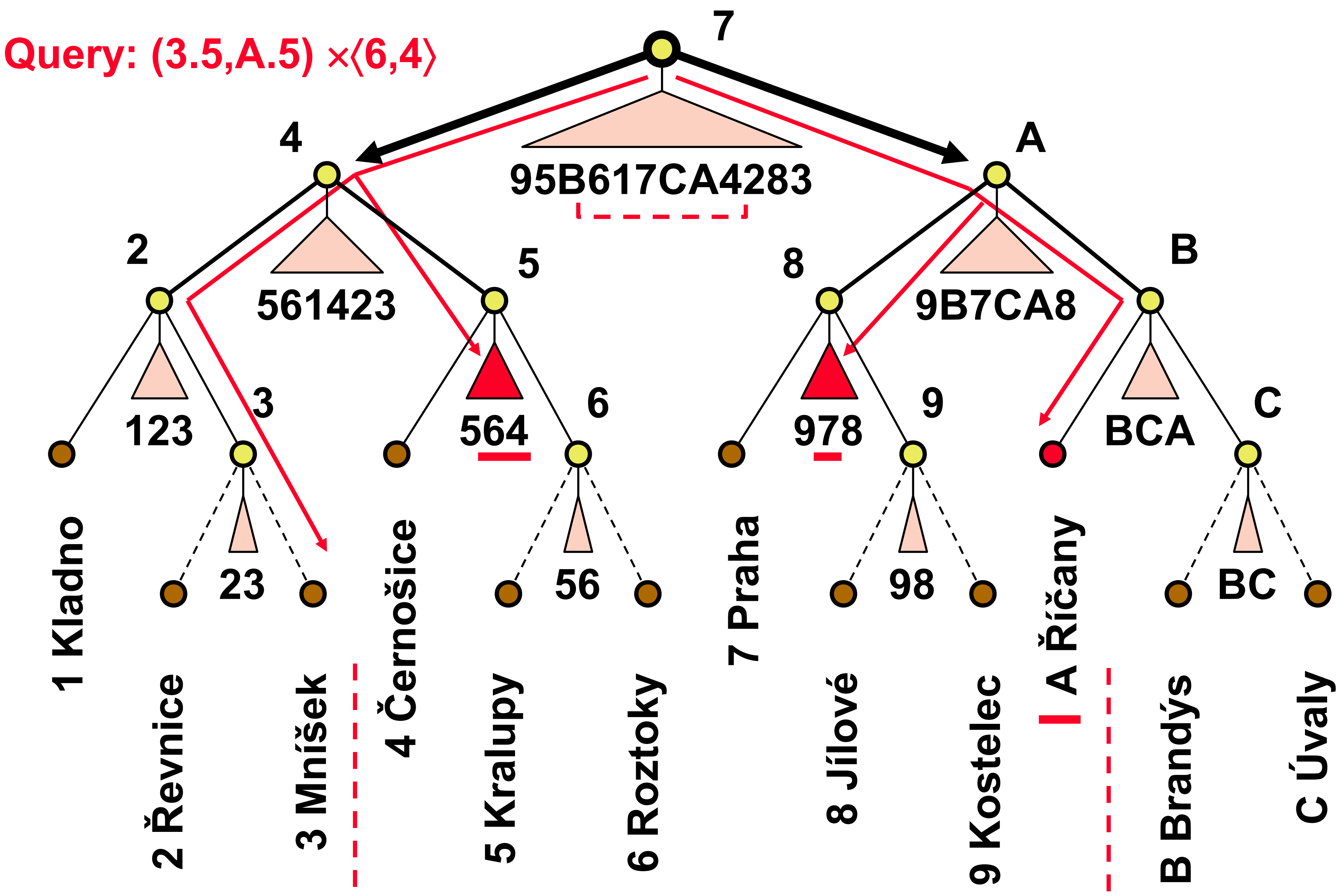
- efficient implementation of **range queries**
  - $\langle \mathbf{x}_{min}, \mathbf{x}_{max} \rangle \times \langle \mathbf{y}_{min}, \mathbf{y}_{max} \rangle$ ve 2D
  - complexity: $\mathbf{O(\log_2 N + F)}$ in 1D, $\mathbf{O(\log_2^2 N + F)}$ in 2D
- **”1D range tree”**
  - balanced binary search tree, leaves are connected by a double-linked list
- **”2D range tree”**
  - balanced binary search tree for the **x** coordinate
  - every inner node contains “1D range tree” (**y** coord.) for all points in the relevant region

"2D range tree"

- balanced binary tree for <u>x-coordinate</u>
- object information <u>in leaves only</u>
- <u>inner nodes:</u> 1D range trees for <u>y-coordinate</u>

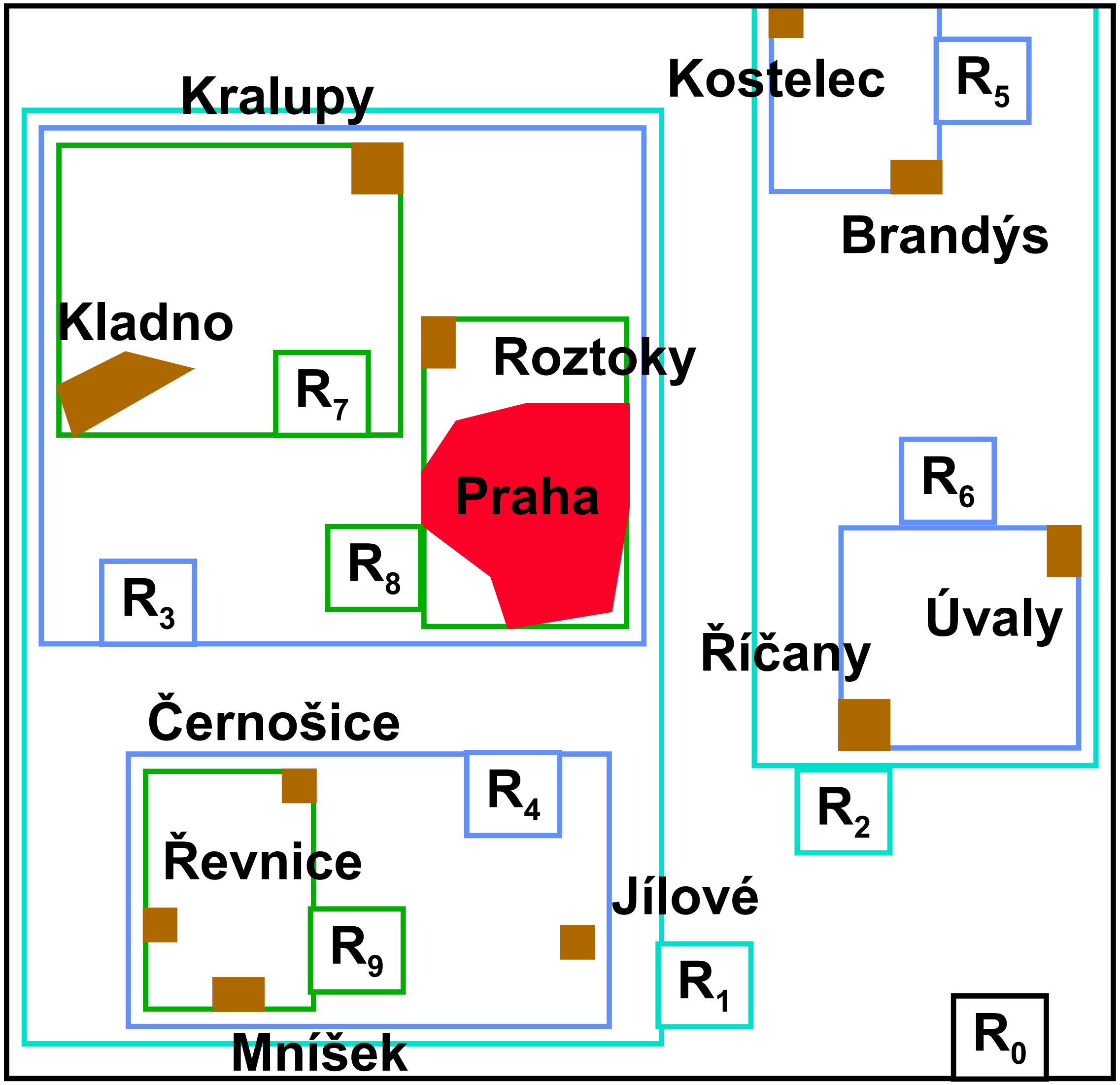


“2D range tree”

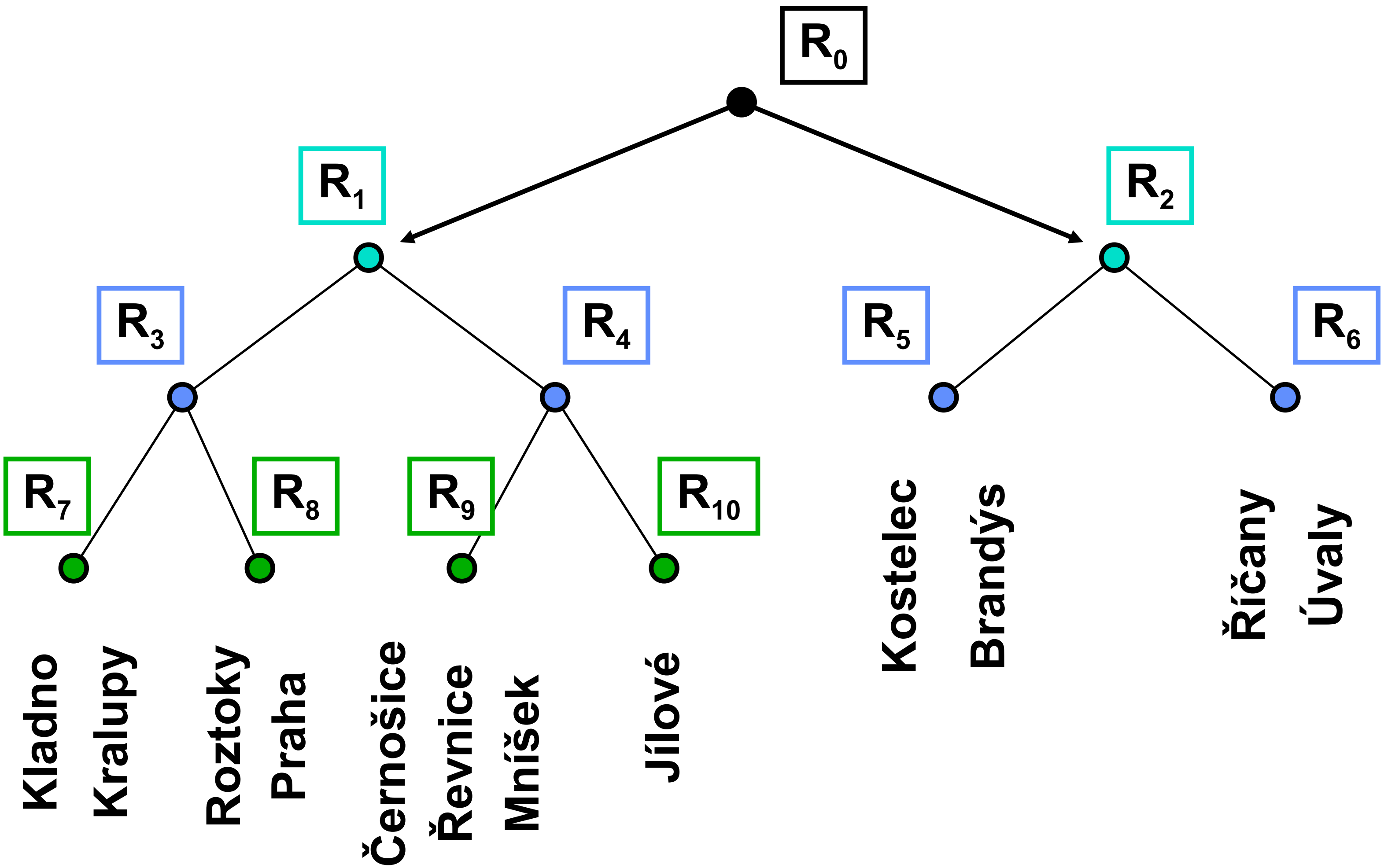


"R-tree" (Guttman)

- objects can be <u>areal</u>
- <u>bounding boxes</u> in inner nodes (the whole subtree must fit into the bounding box)
- object references <u>in leaf nodes</u>



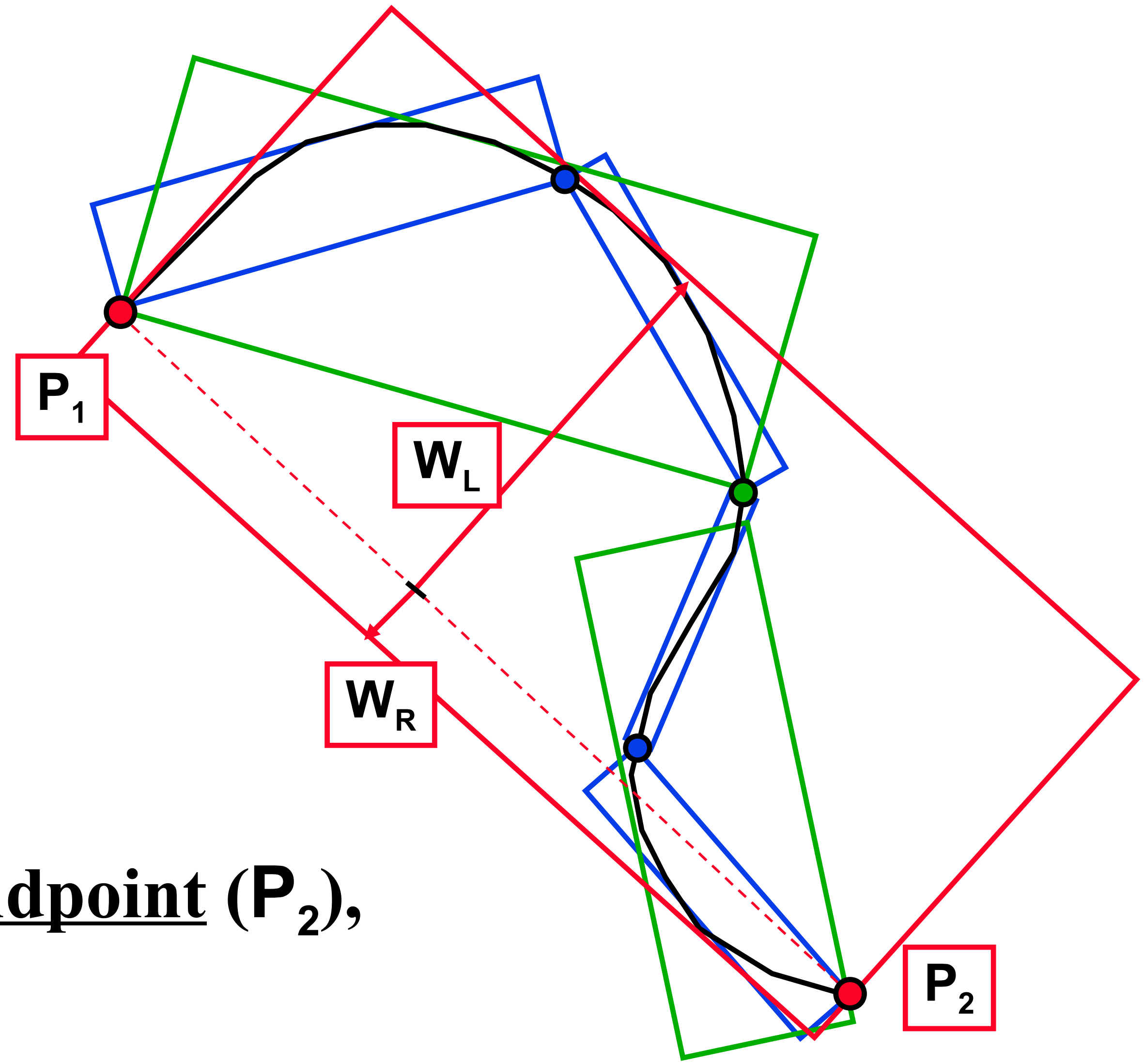
“R-tree”



R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
Kladno
Kralupy
Roztoky
Praha
Černošice
Řevnice
Mníšek
Jílové
Kostelec
Brandýs
Říčany
Úvaly

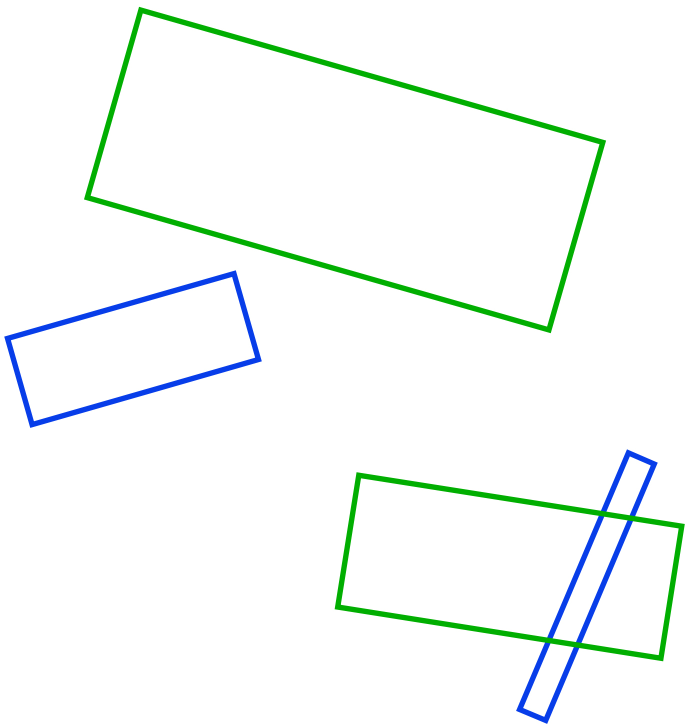
"Strip tree" (Ballard)

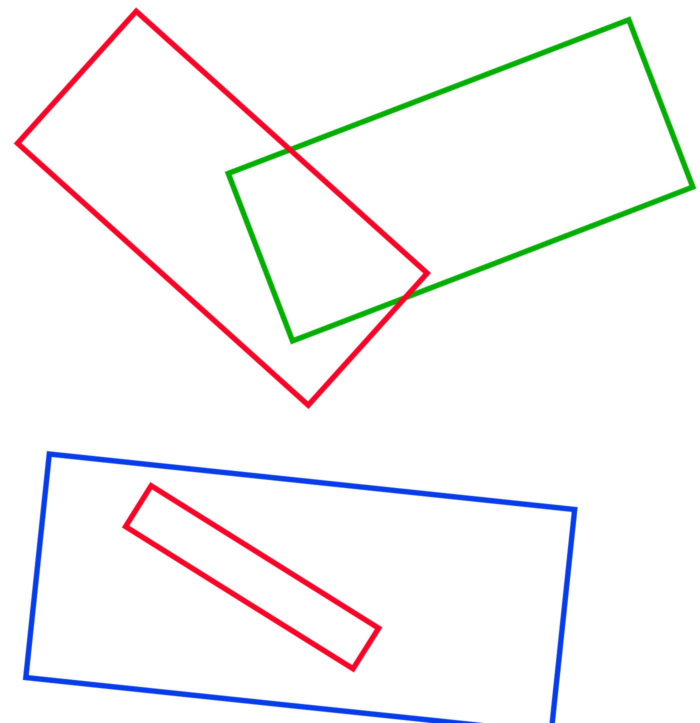
- **planar curve** (polyline)
- **adaptive splitting induced by the curvature**
- **oriented bounding rectangle defined by: starting point (P_1), endpoint (P_2), widths (W_L, W_R)**



Intersection of two curves

- **"strip trees"** were built for both curves
  - **trivial cases**
  - **subdivision is necessary**





Bounding system hierarchies

- ”**Sphere tree**” (Palmer, Grimsdale, 1995)
  – simple tests & transform, approximation not so good
- ”**AABB tree**” (Held, Klosowski, Mitchell, 1995)
  – simple test, more complicated transform
- ”**OBB tree**” (Gottschalk, Lin, Manocha, 1996)
  – simple transform, more complicated test, good approximation
- ”**K-dop tree**” (Klosowski, Held, Mitchell, 1998)
  – complicated transform & test, excellent aproximation

Directional pass

- data pass using specific **directional order**
  - visibility (front-to-back or vice versa) in orthographic projection
  - ”plane sweep” pass (“sweeping”)
- pass from the **center point**
  - visibility (form-factors, ..) in perspective projection
  - kNN search (“k-Nearest Neighbors”)

Presumption

- **hierarchical** space decomposition
  - inclusion conditions on objects only, not needed for bounding boxes (e.g. ”strip tree”)
- effective computation of **minimal distance of each cell / box** ... **d(B)**
  - distance from sweep plane or center point of the pass
  - distance need not be an infimum (but has to be an lower bound)

Algorithm

- auxiliary data structure **H** (heap)
  - efficient operations: **Min**, **DeleteMin**, **Insert**

❶ put the root node into **H**
  - **d(B)** is used for **heap sorting**

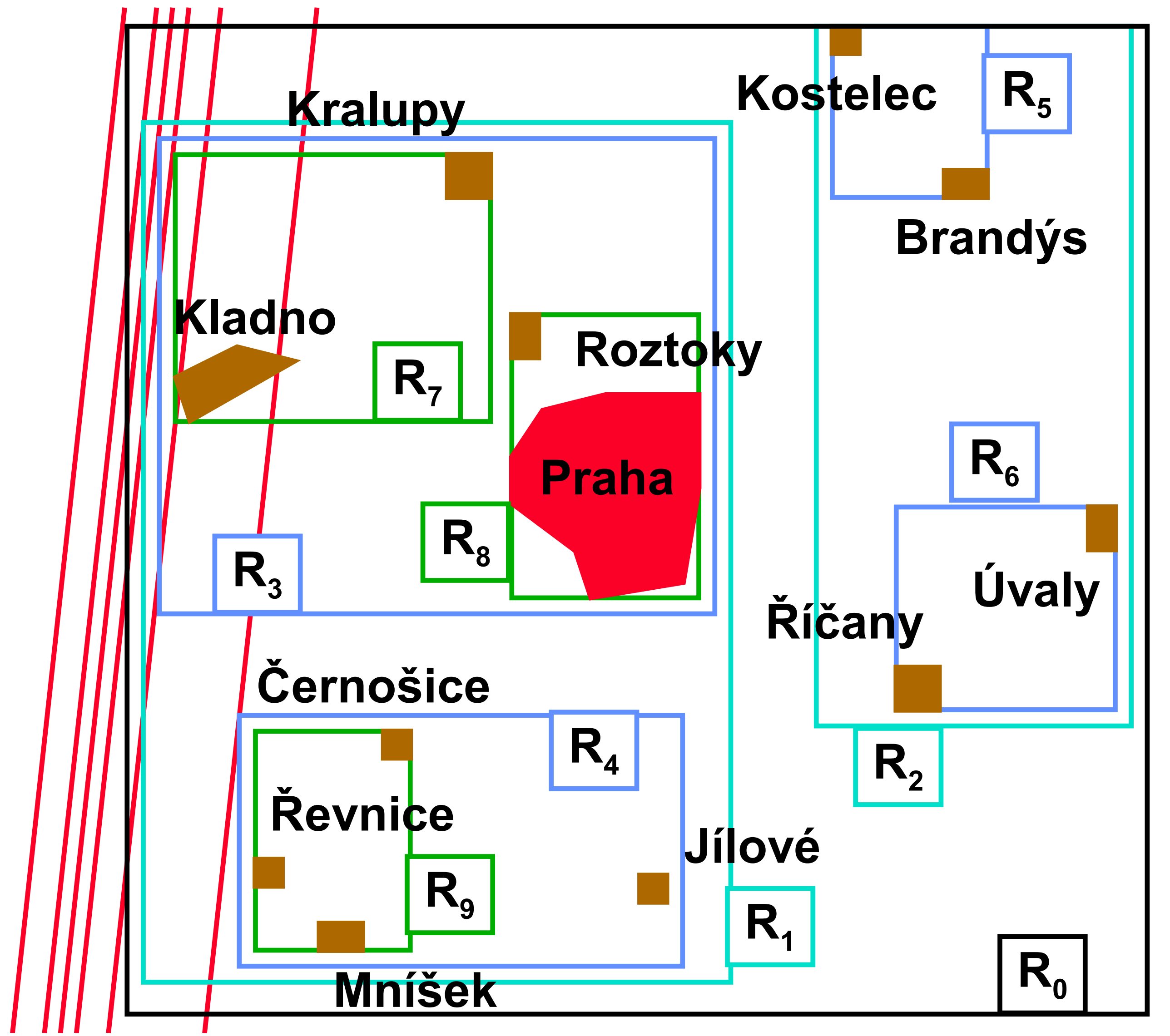
❷ if **Min(H)** is **an object**, process it and remove it

❸ if **Min(H)** is **a hierarchy cell**, remove it and put all its children back into the heap

❹ repeat steps ❷ and ❸ until the heap **H** is empty or until all required output objects are processed (kNN)

Example I

- $\{R_0\} \rightarrow \{R_1, R_2\}$
- $\{R_1, R_2\} \rightarrow \{R_3, R_4, R_2\}$
- $\{R_3, R_4, R_2\}$
  $\rightarrow \{R_7, R_4, R_8, R_2\}$
- $\{R_7, R_4, R_8, R_2\}$
  $\rightarrow$ {**Kladno**, R_4,
  Kralupy, R_8, R_2}
- Kladno
- {R_4, Kralupy, R_8, R_2}
  $\rightarrow$ {R_9, Kralupy, R_8,
  Jílové, R_2}



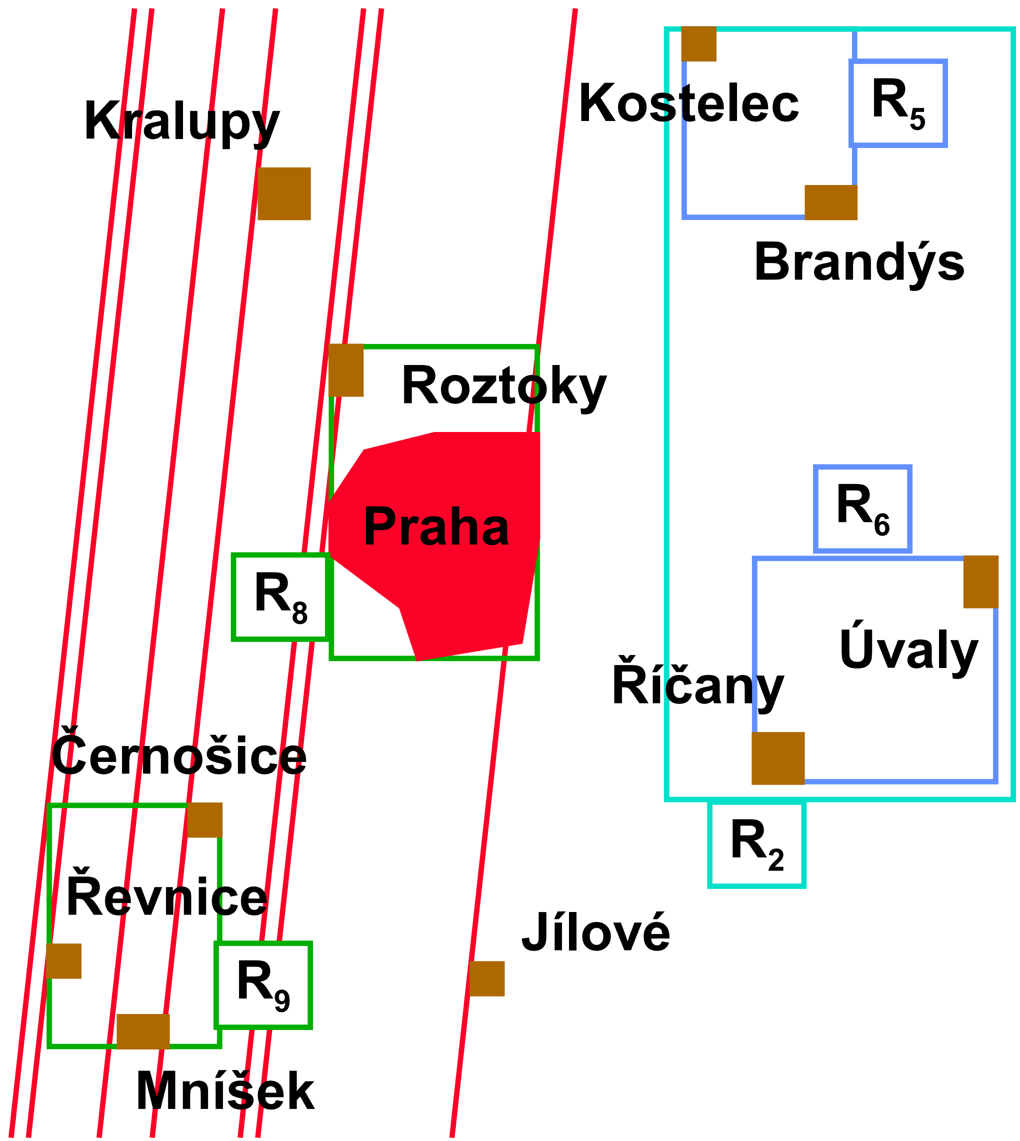
Example II

- {R$_9$,Kralupy,R$_8$,Jílové,R$_2$} →{Řevnice,Mníšek, Kralupy,Černošice, R$_8$,Jílové,R$_2$}

❷-❺ Řevnice,Mníšek, Kralupy,Černošice

- {R$_8$,Jílové,R$_2$} →{Roztoky,Praha, Jílové,R$_2$}

❻-❽ Roztoky,Praha, Jílové



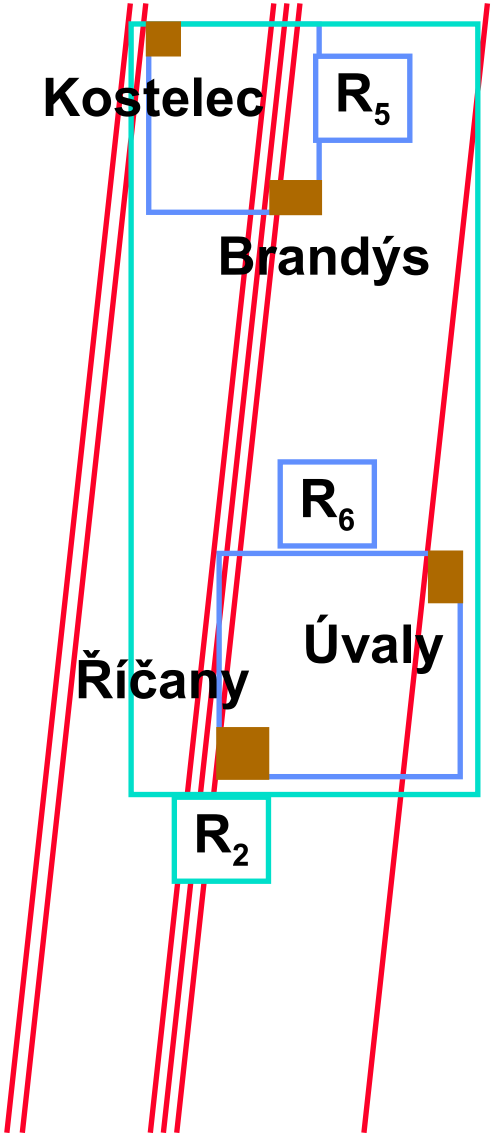
Example III

- $\{R_2\} \rightarrow \{R_5, R_6\}$
- $\{R_5, R_6\} \rightarrow \{$Kostelec$, R_6,$ Brandýs$\}$

❾ **Kostelec**

- $\{R_6,$ Brandýs$\}$
  $\rightarrow \{$Brandýs, Říčany, Úvaly$\}$

⑩-❶❷ **Brandýs,Říčany,Úvaly**



The End

More information:

- **H. Samet**: ***The Design and Analysis of Spatial Data Structures***, Addison-Wesley, 1990
- **H. Samet**: ***Foundations of Multidimensional and Metric Data Structures***, Morgan Kaufmann, 2006
- **F. Preparata, M. Shamos**: ***Computational Geometry, An Introduction***, Springer-Verlag, 1985