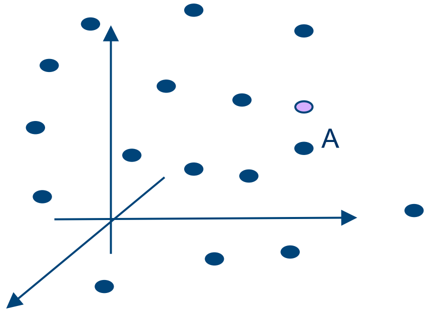
Challenge....

- Traditional data is one dimensional.
- Multimedia data is multi dimensional.
  - Ex. Maps are 2D
  - Ex. Images are (with * height) D
    (assuming that each pixel is a feature)
  - In general, if a given information has k features, it can be represented by a k-dimensional space



What kind of queries we can expect?

- Given a set of point in k-dimensional space
  - Exact match:
    - find if a given point is in the set or not
  - Nearest neihgbor:
    - find the closest point to a given point
  - Range search:
    - Given a region (rectangle or circle), find all the points in the given region

General approach

- Divide the space into regions
- Insert the new object into the corresponding region
- If the region is full, split the region
- retrieval: determine which regions are required to answer a given query and limit the search to these regions

Is there an alternative to multi-dimensional space decomposition?

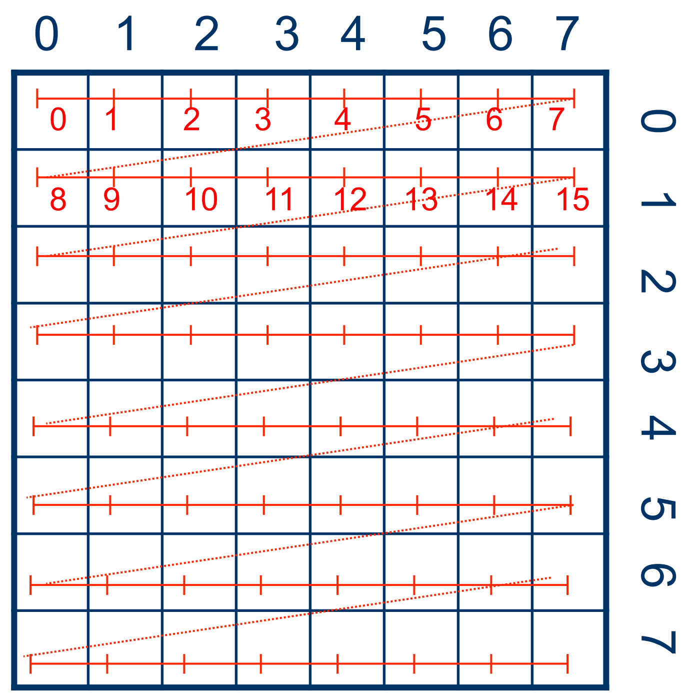
- YES!
  - Convert a given k-D space to 1D space
  - We know how to handle 1D space!!

- Don't we loose information??
  - Yes, but if we are careful, we can minimize the information loss.

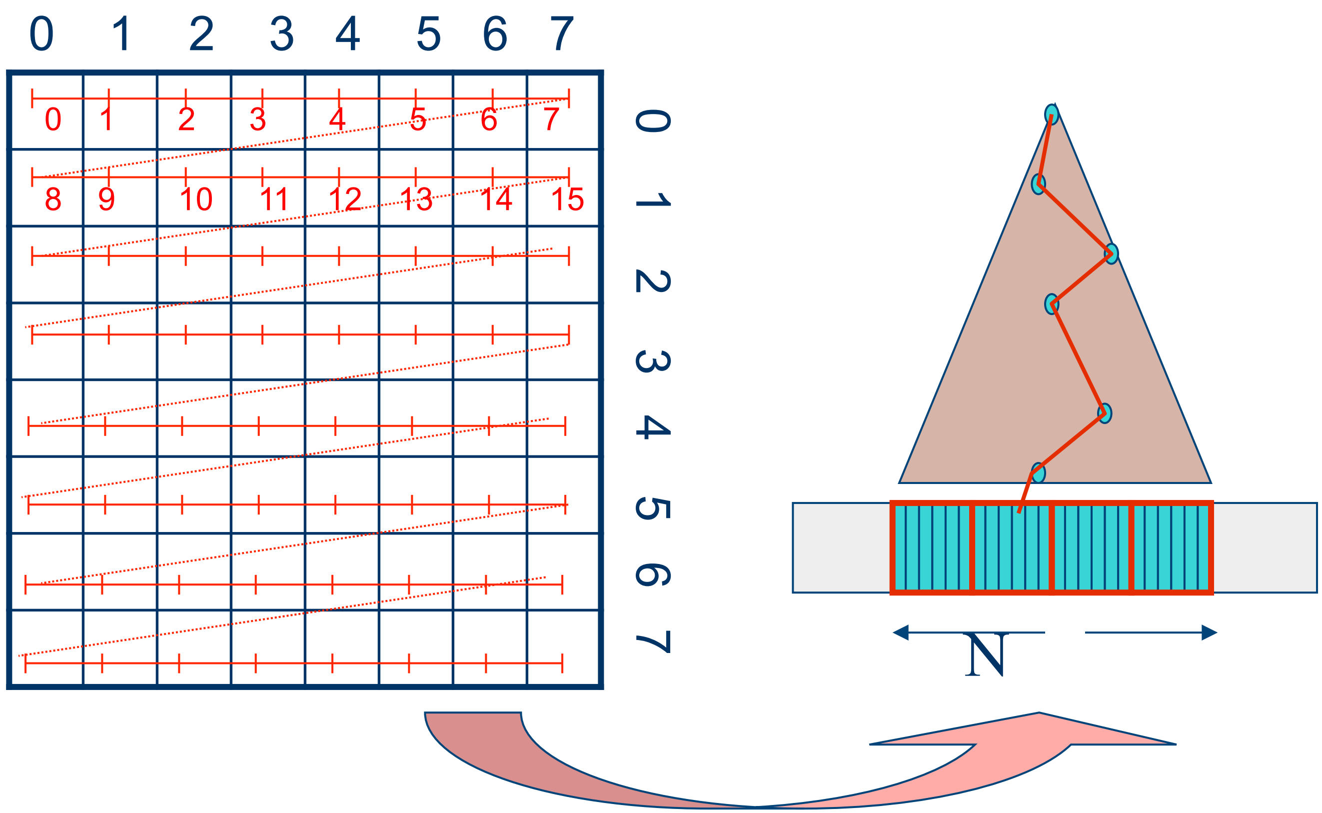
Space filling curves

- Convert a k-D space into 1D space such that points that are close to each other in k-D space are also close to each other in 1-D space

Row order/column order

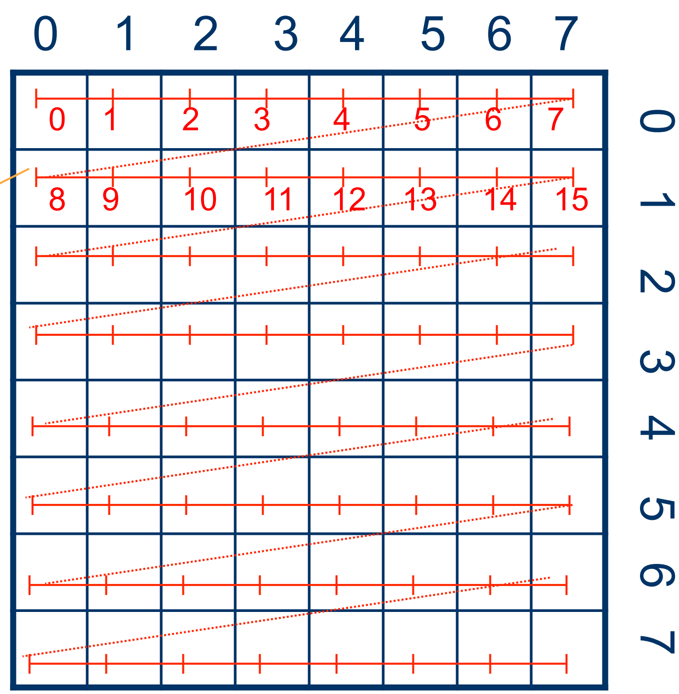


Row order/column order

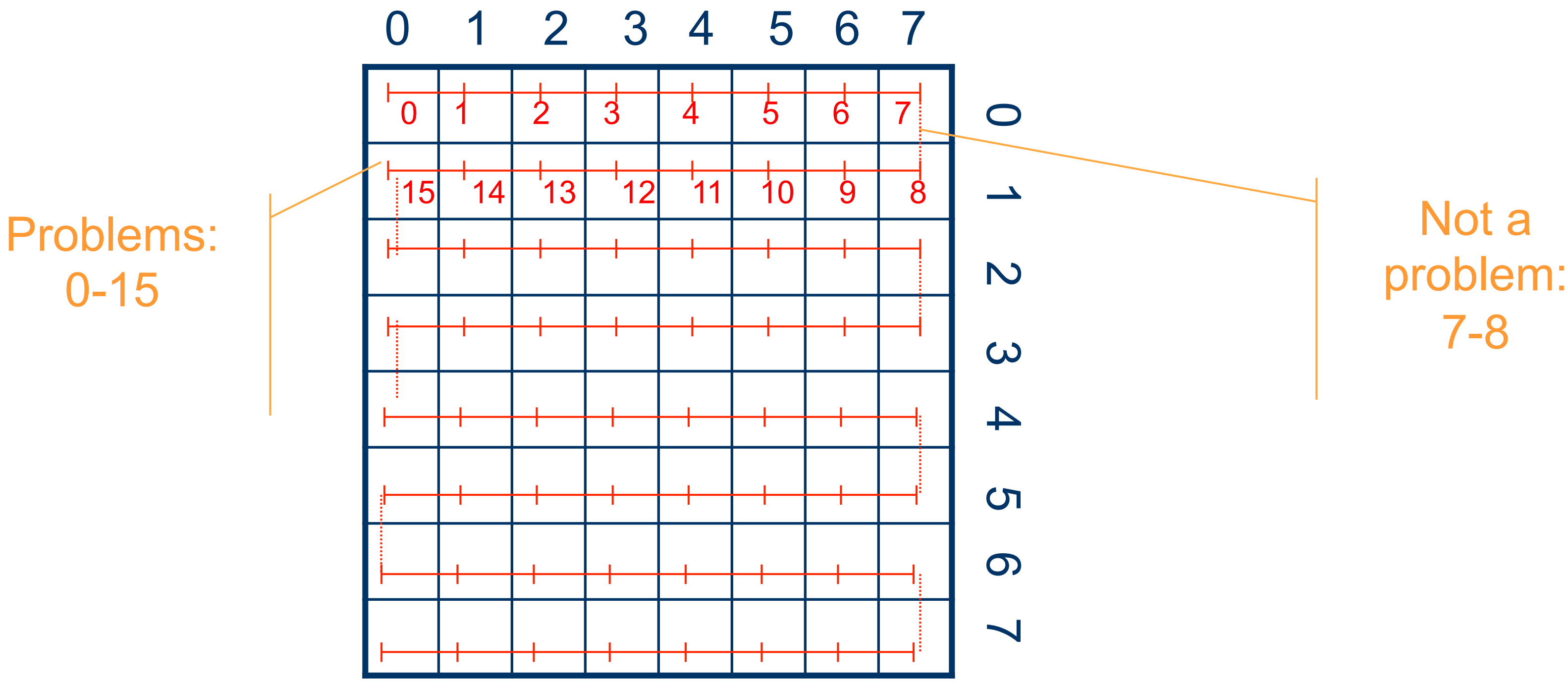


Row order/column order

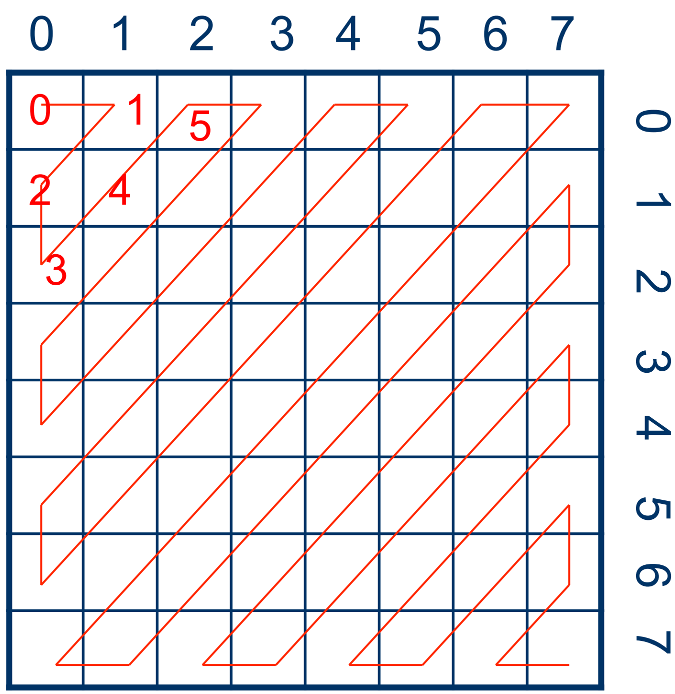
Problems:
0-8
7-8



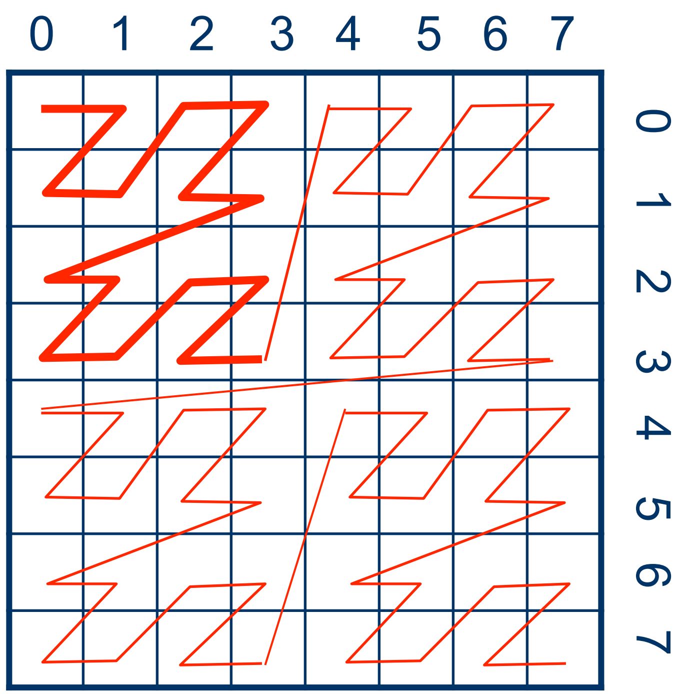
Row prime order/column prime order



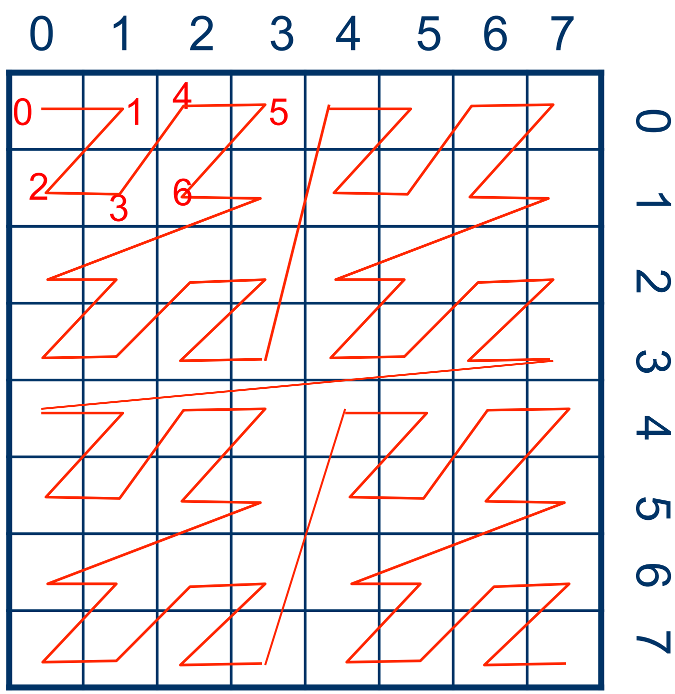
Cantor diagonal order



Z-order curve (hilbert curve)

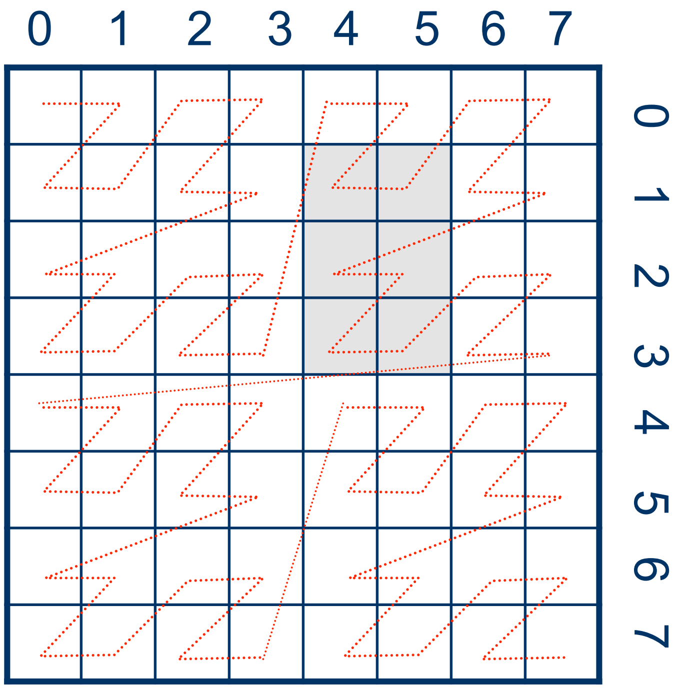


Z-order curve (hilbert curve)

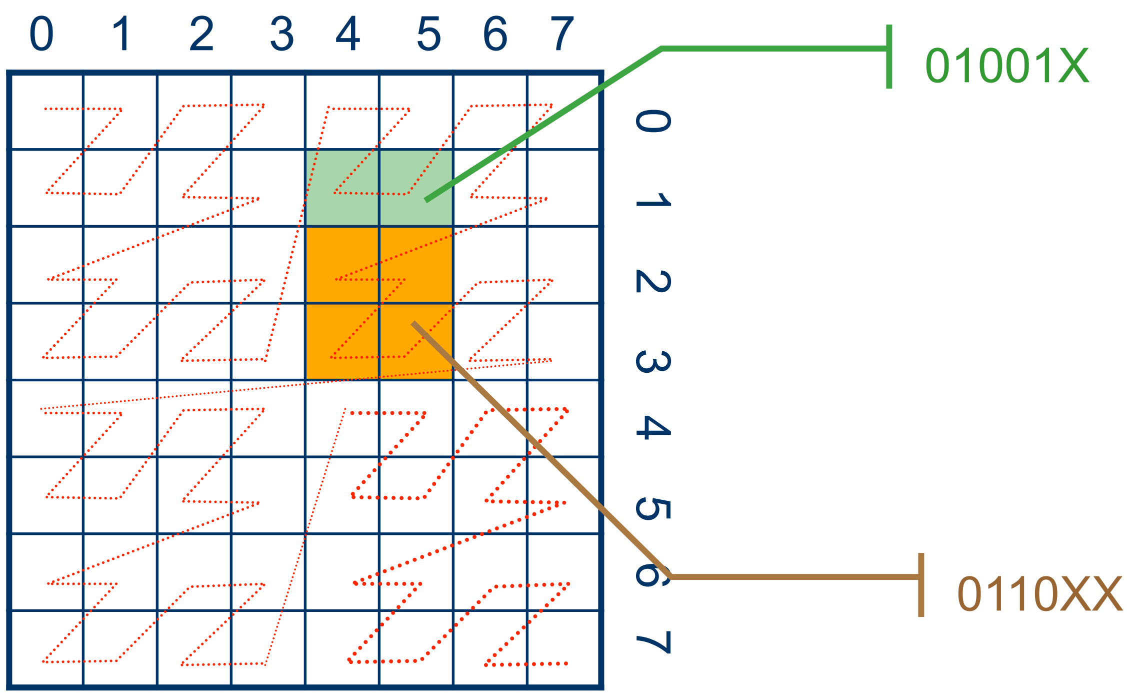


Easy to compute (bit-shuffling): 1(001) X 2 (010) = 6 (000110)

Z-order curve (hilbert curve)

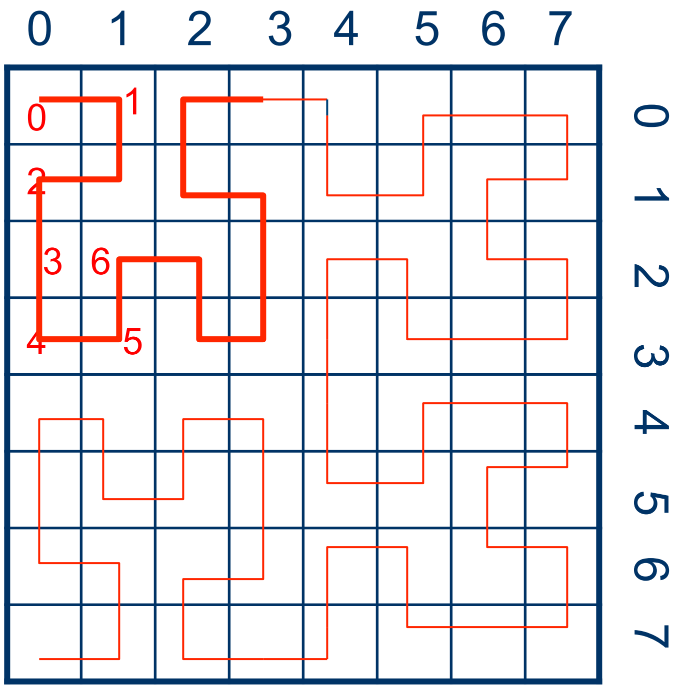


Z-order curve (hilbert curve)



Range search can be implemented using tries…

Peano-hilbert curve



Indexing

- What are we indexing???
  - Text → tries
  - Numbers, text → B-trees, B+ trees, B*trees
  - Images→???????????
- Which feature are we going to index on?
  - Color? Texture? Time? (image series)
- What do we need to specify?
  - Lines? Points? Space?

How do we index points?

- Given
  - a space of N-dimensions
  - M points
  - a distance function between points
- we can use multidimensional index structures
  - k-d trees
  - point quadtrees
  - MX quadtrees
  - R-trees
  - TV-trees
  - X-trees

So…

- we can answer queries of the form
  - Given
    - a point X in N-dimensional space

    Find
    - all points Y that are in its proximity ($d(X,Y) < \varepsilon$)

…thus…

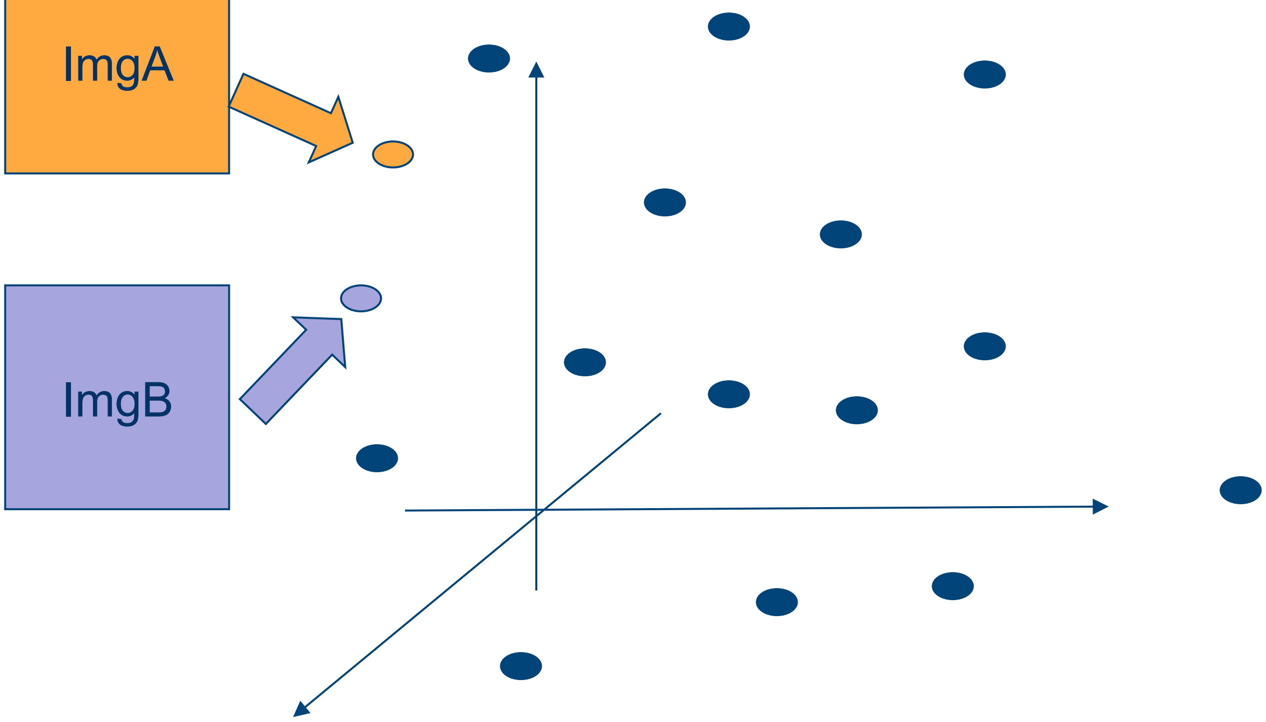
- If
  - we represent any feature as a point in N-dimensional space (color, texture, shape, etc.)
  - we define a distance function between those points
    - (larger distance→lower similarity)
- Then
  - we can find media object with similar properties.

Populate database

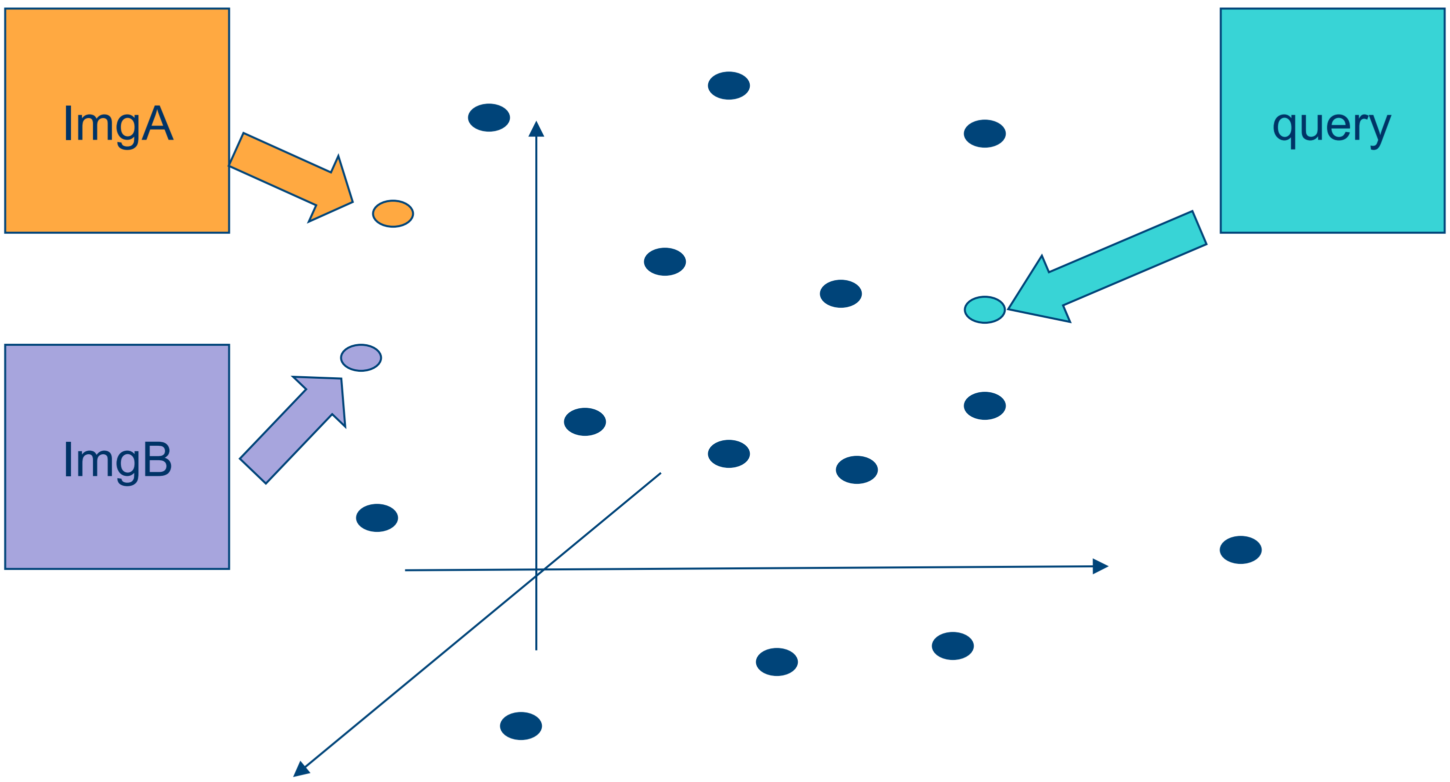
ImgA

ImgB

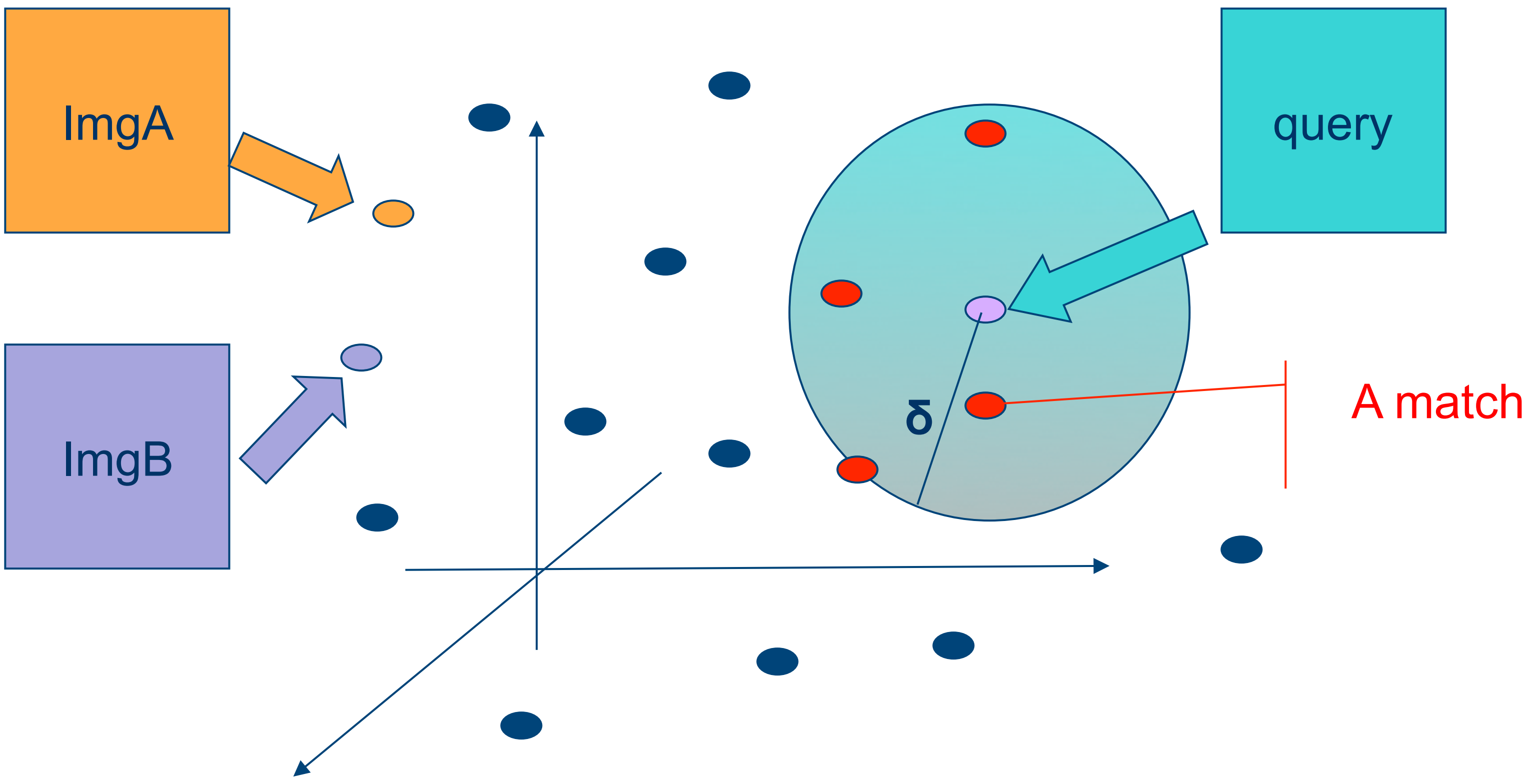
Populate database



Map query image

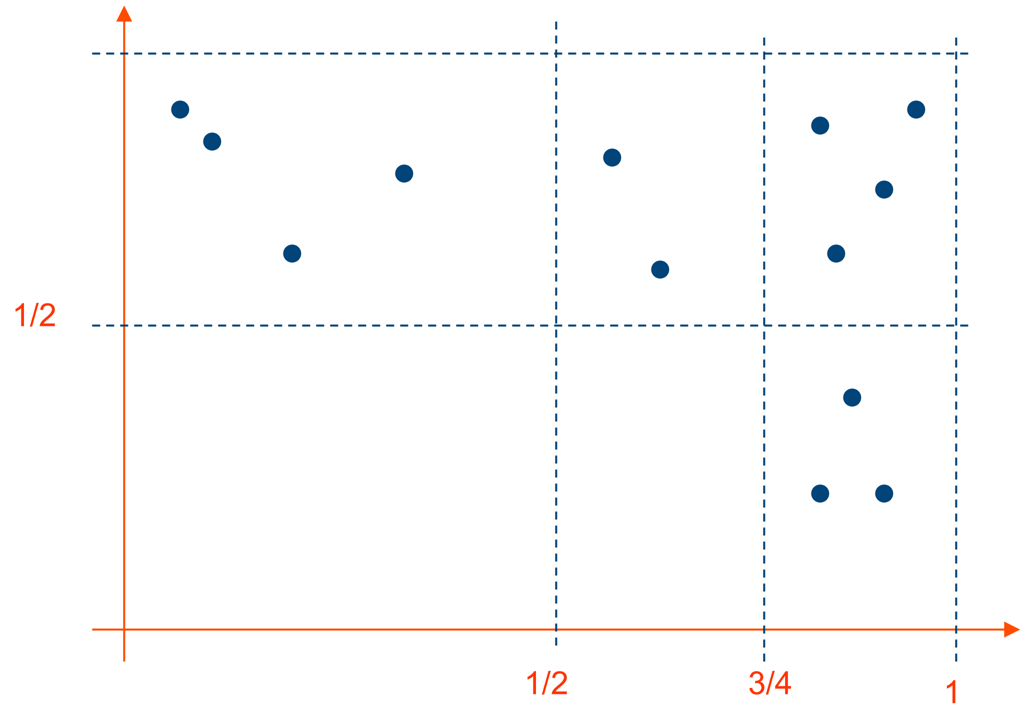


Range search



Grid File

- Every cell is one disk page



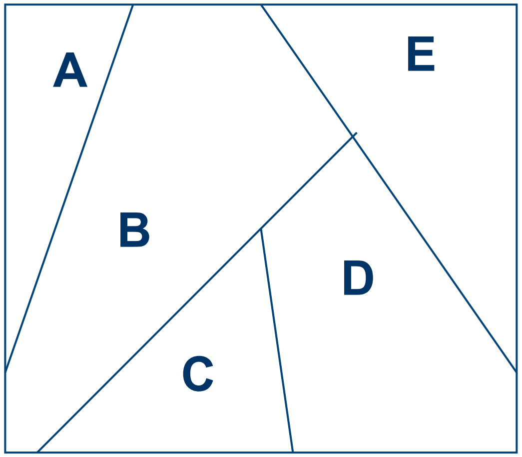
Problems:
1 - Variable page utilization
2 - Dead space

Point Trees

How can we divide space?

- Let us assume that the space is 2-d
- There are many ways to divide the space
  - Fixed size squares
  - Triangles
  - Rectangles
  - Arbitrary space decomposition

- Each line divides the space into two
  - Line: $n_1x+n_2y = c$
  - Regions: $n_1x+n_2y >= c$
    $n_1x+n_2y < c$

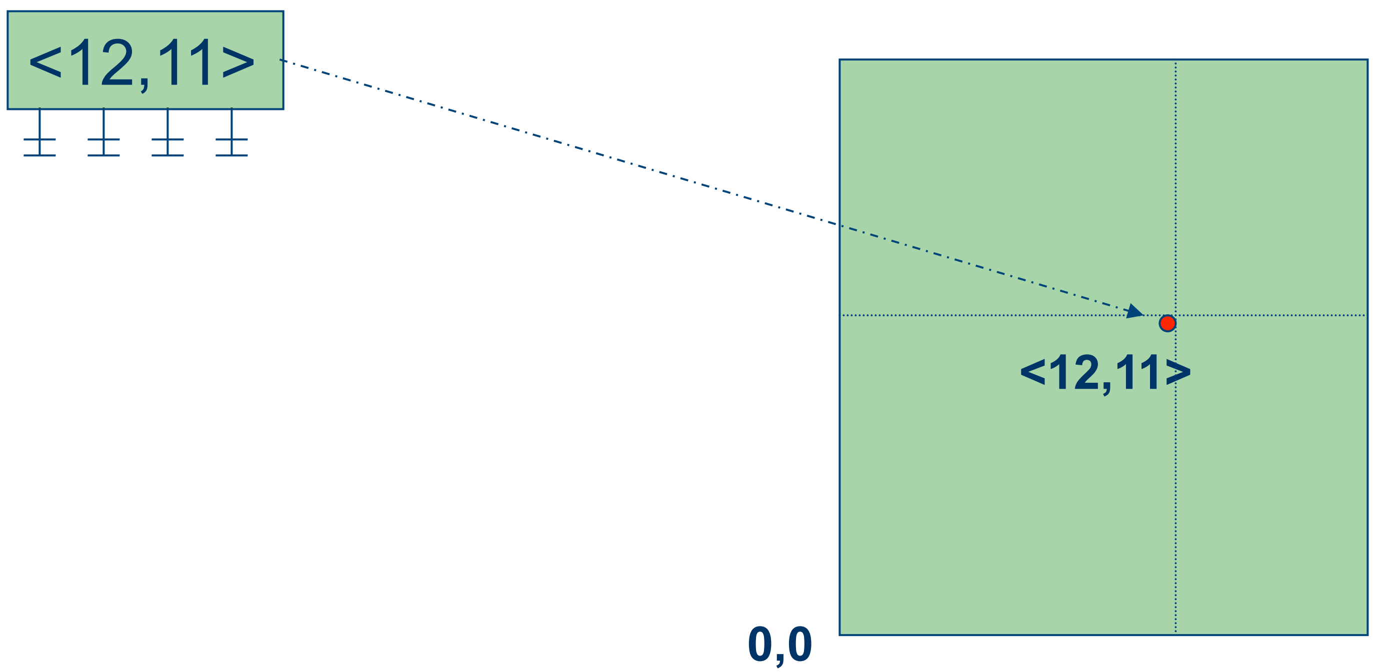


Point quadtrees (Finkel and Bentley 74)

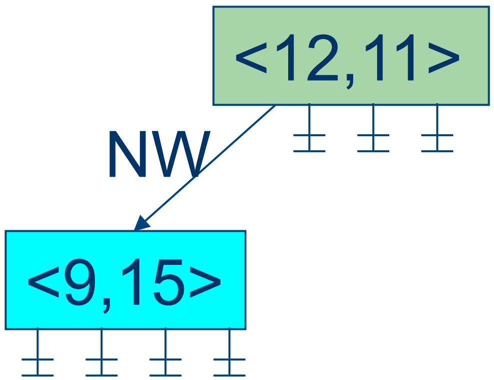
- Key features:
  - Every node in a point quadtree *implicitly* represents a rectangular region.
  - Each node contains an *explicit* point labeling it.
  - Root represents the whole region.
  - Each node's region is split into 4 parts ("quadrants") by drawing a vertical and a horizontal line through the point labeling the node.
  - Each node has 4 children corresponding to the 4 "quadrants" above.

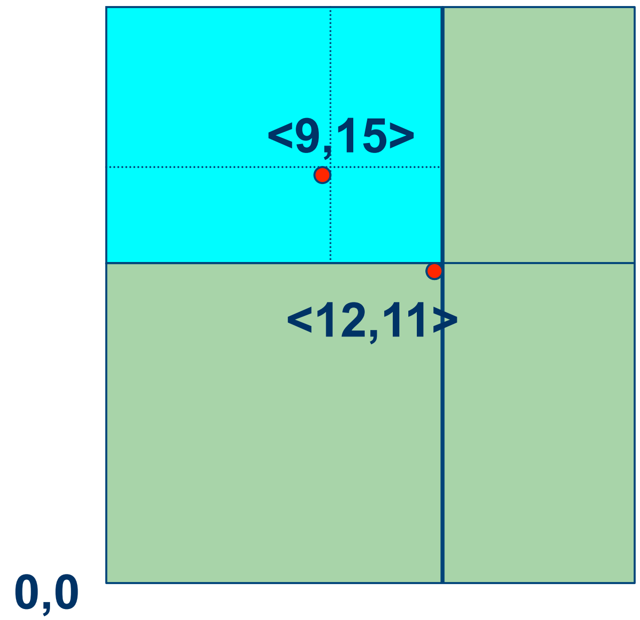
Point quadtrees: example

Represents whole region

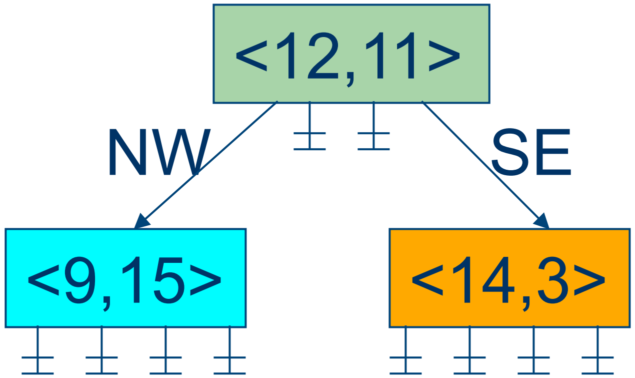


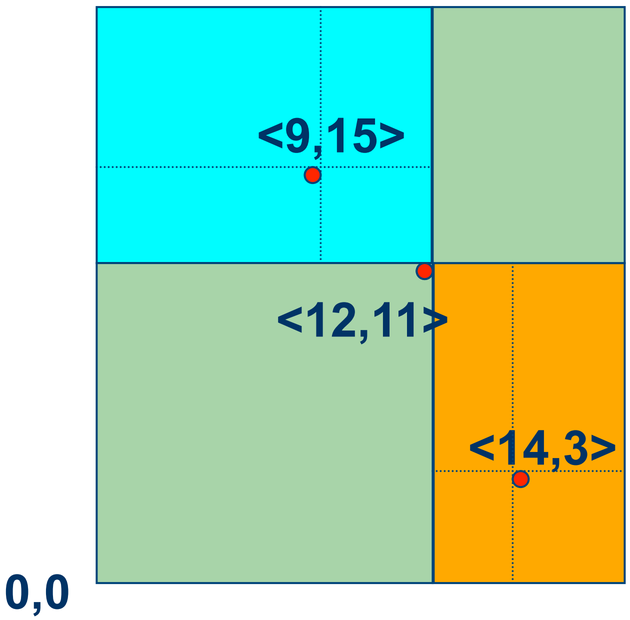
Point quadtrees: example



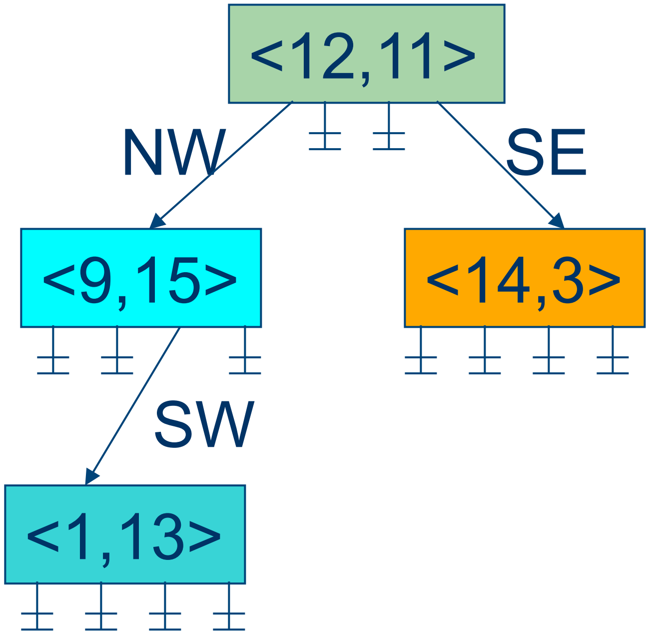


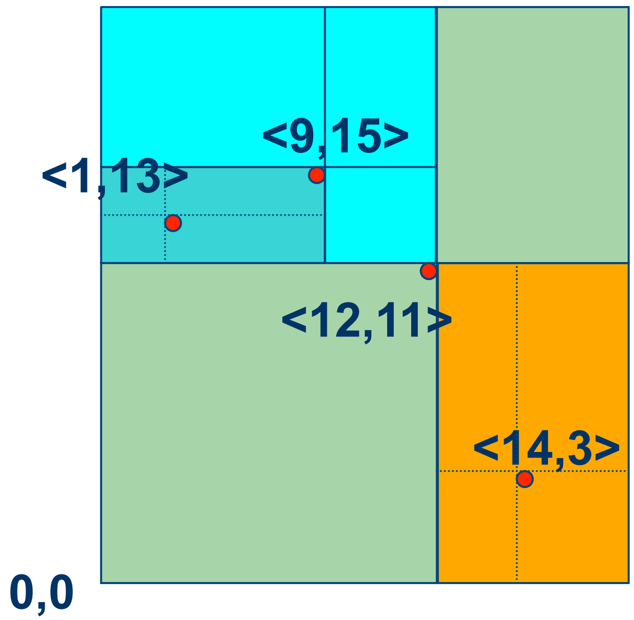
Point quadtrees: example





Point quadtrees: example





Observation

- The structure of the tree depends on the insertion order!!!!

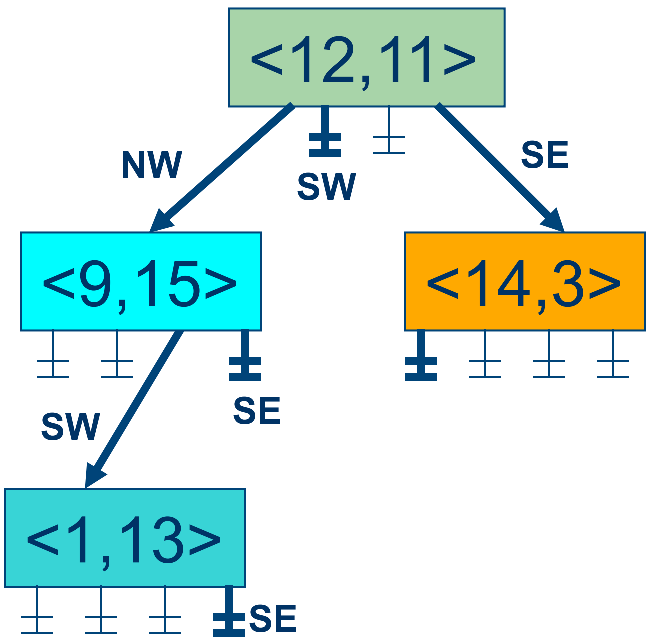
- Exercise: try to insert nodes in the following order
  <14,3> <12,11>, <1,13> <9,15>

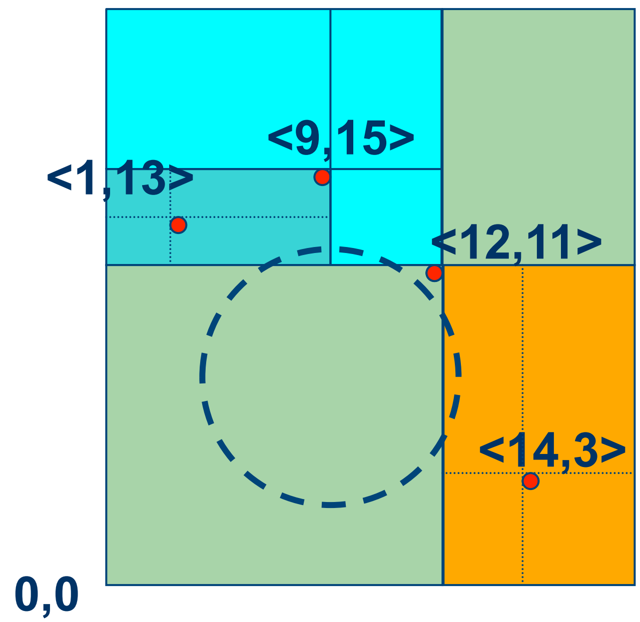
  and compare the resulting tree with the previous one.

Key Points

- Suppose a point quadtree has N nodes in it.
- Worst case height = N.
- Worst case insertion time = N.
- Other operations are:
  - Deletion: delete a point
  - Range query: find all points within a given region
  - NN query: find the nearest neighbor (or M nearest neighbors) of a given point.

Point quadtrees: range search

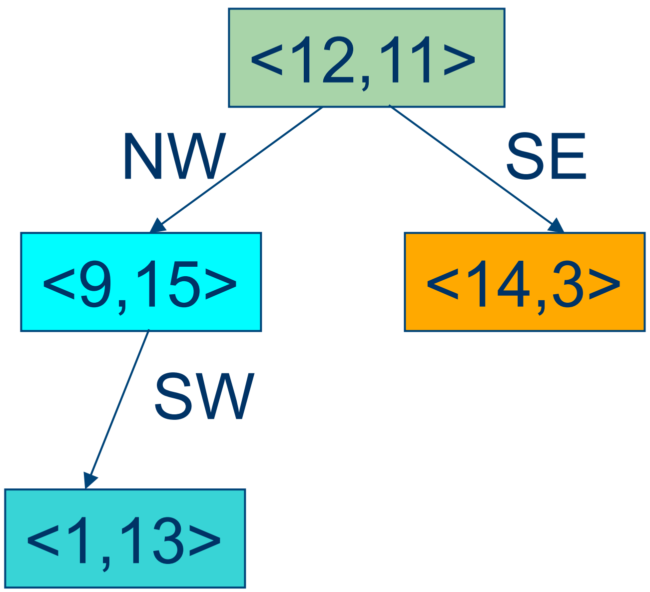


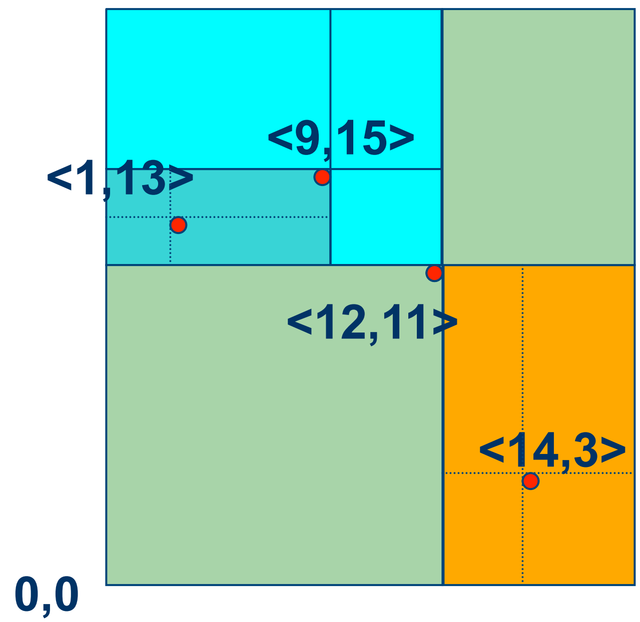


Deletion

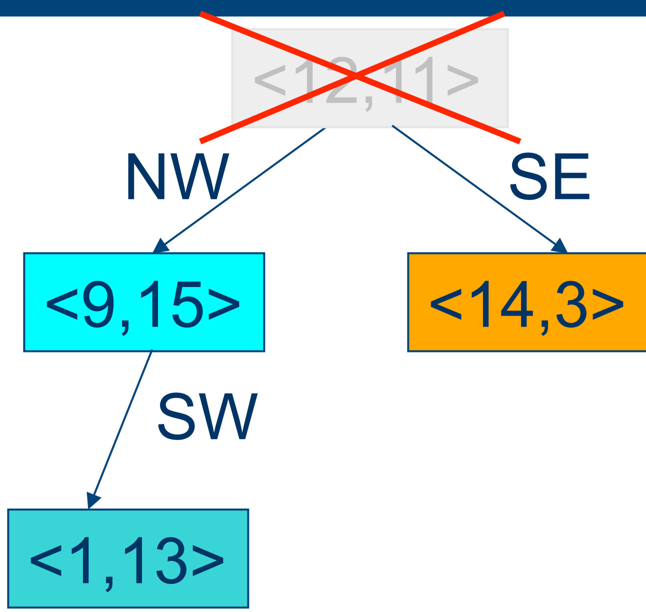
- Suppose T is the root of a point quadtree and you want to delete <x,y>.
- Steps:
  - Find <x,y> by doing a search.
  - If it is a leaf node, then simply set the appropriate link field of its parent to nil (and return the node to available storage).
  - What if it is not a leaf ?

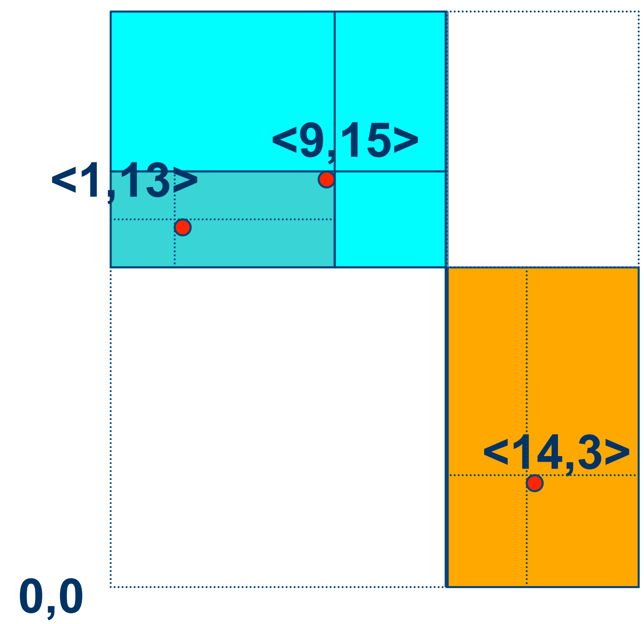
Delete <12,11>



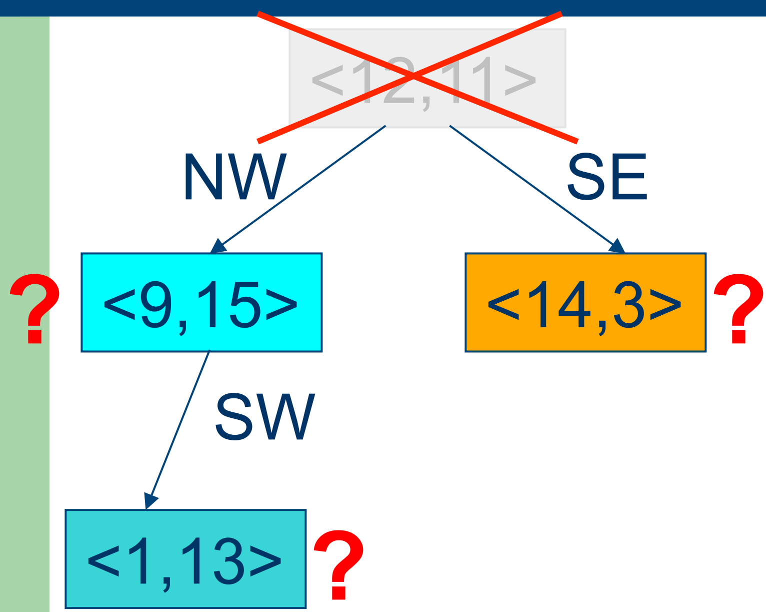


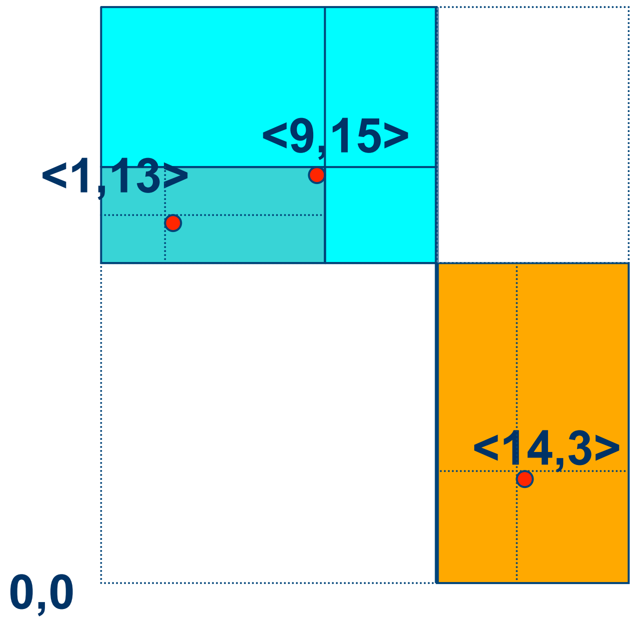
Delete <12,11>



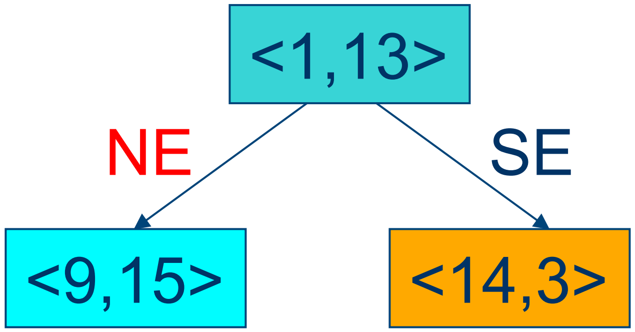


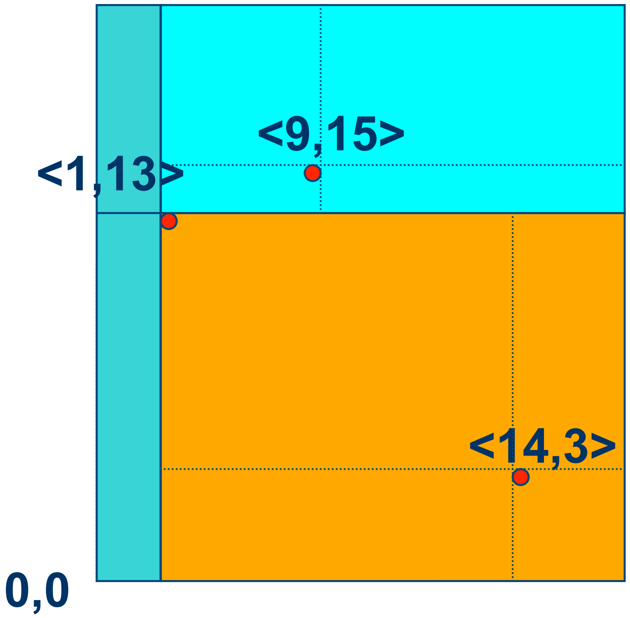
Delete <12,11>





Let us choose <1,13>





Problems with Point Quadtrees

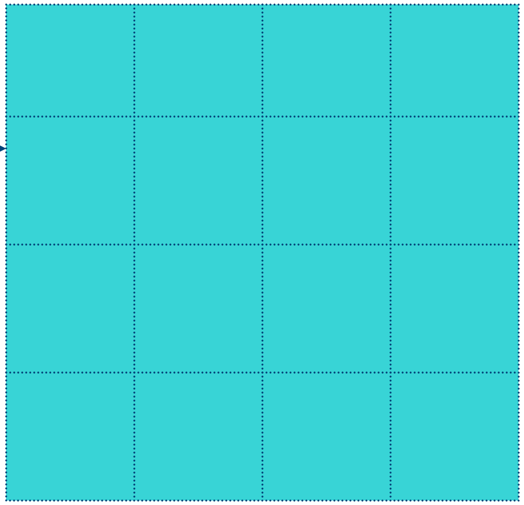
- Deletion is slow.
- Tree can be highly unbalanced.
- Size of regions associated with nodes can vary dramatically.
- All these factors make the time taken to compute NN and range queries unpredictable.

MX quadtrees

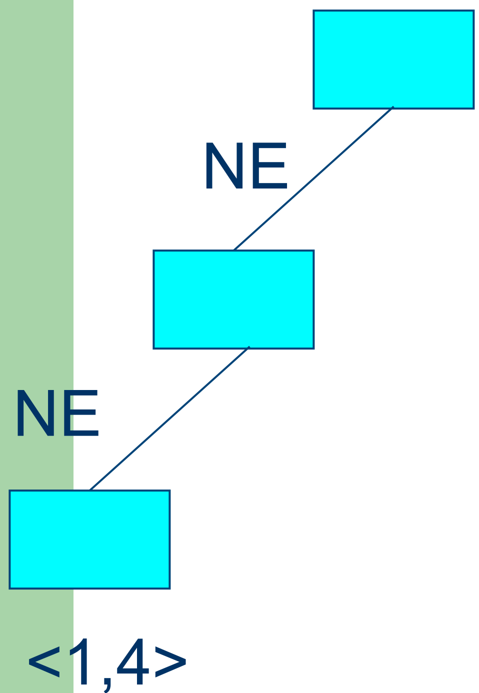
- In point quadtrees, the region is split by drawing a vertical and a horizontal line through the point labeling node N.
- In MX-quadtrees,
  - the entire space is a $2^n \times 2^n$ matrix.
  - region is split by drawing a vertical and a horizontal line through the center of the region.

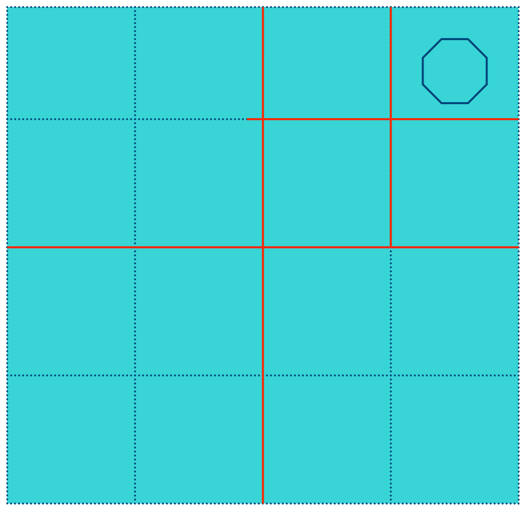
MX quadtrees: example

Empty region



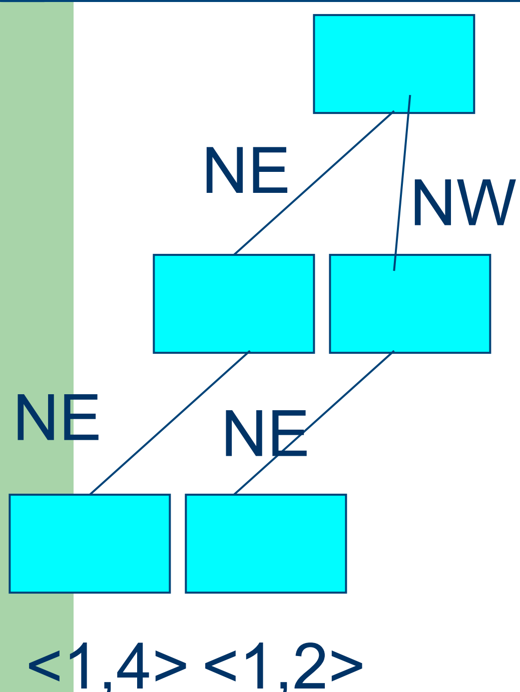
MX quadtrees: example

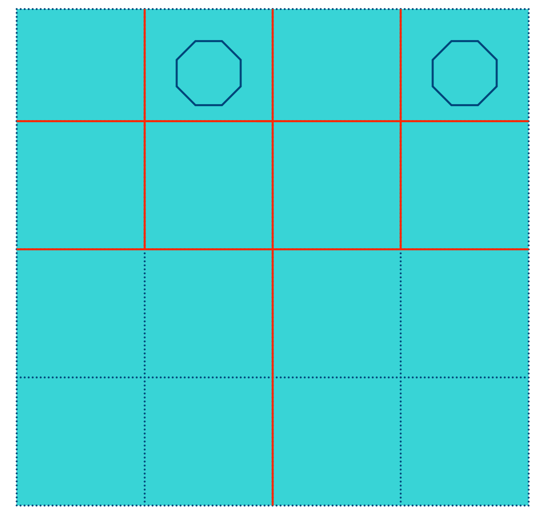




Insert <1,4>

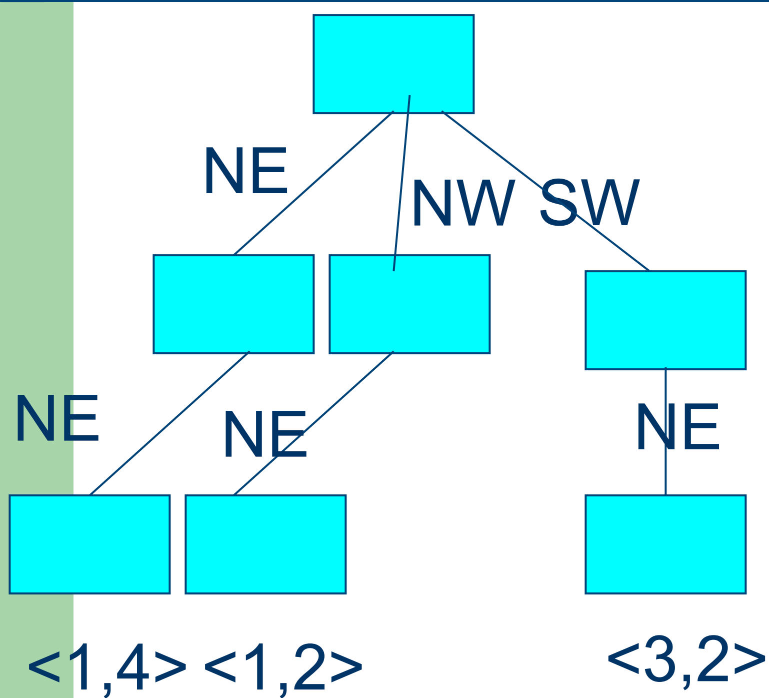
MX quadtrees: example

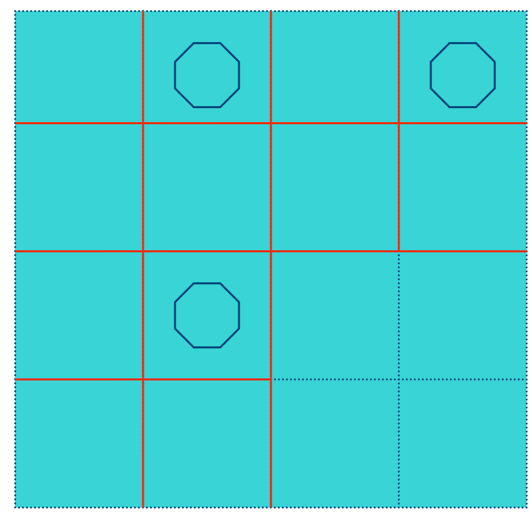




Insert <1,2>

MX quadtrees: example





Insert <3,2>

MX-quadtrees: salient features

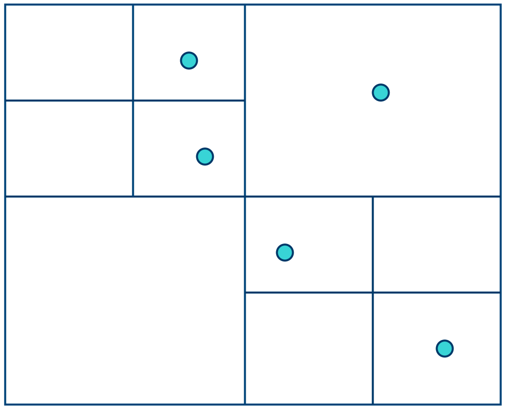
- Each node represents a region.
- Root (level 0) represents 2^n x 2^n region.
- Nodes at level j represent 2^{n-j} x 2^{n-j} region.
- Points label leaf nodes (at level n).
- Insertion takes time O(n).
- So does search for a point.

MX-Quadtrees: deletion

- Very easy to delete a point.
- First search for the point (which must be a leaf) and delete the leaf.
- If the parent now has 4 empty child fields, then delete the parent. And repeat as long as possible. This process is termed “collapsing”.

PR-quadtrees

- MX-quadtree works well if the data is discrete
  - otherwise, it may need to use buckets, which may increase search time
- PR-quadtree (point region quadtree) assumes a continous space.



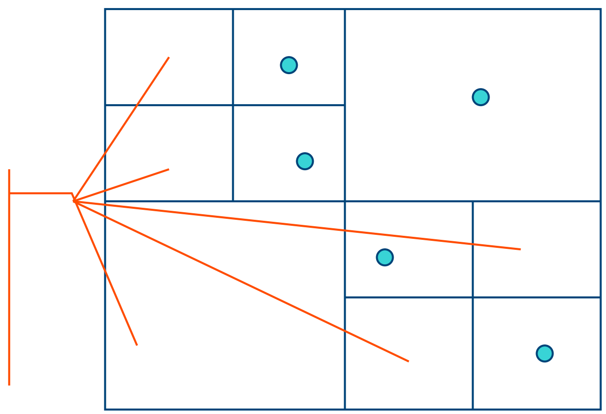
Structure is independent of insertion order

Deletion is easy

PR-quadtrees

- MX-quadtree works well if the data is discrete
  - otherwise, it may need to use buckets, which may increase search time
- PR-quadtree (point region quadtree) assumes a continous space.

BD-trees minimize the waste of space



Structure is independent of insertion order

Deletion is easy

KD-trees

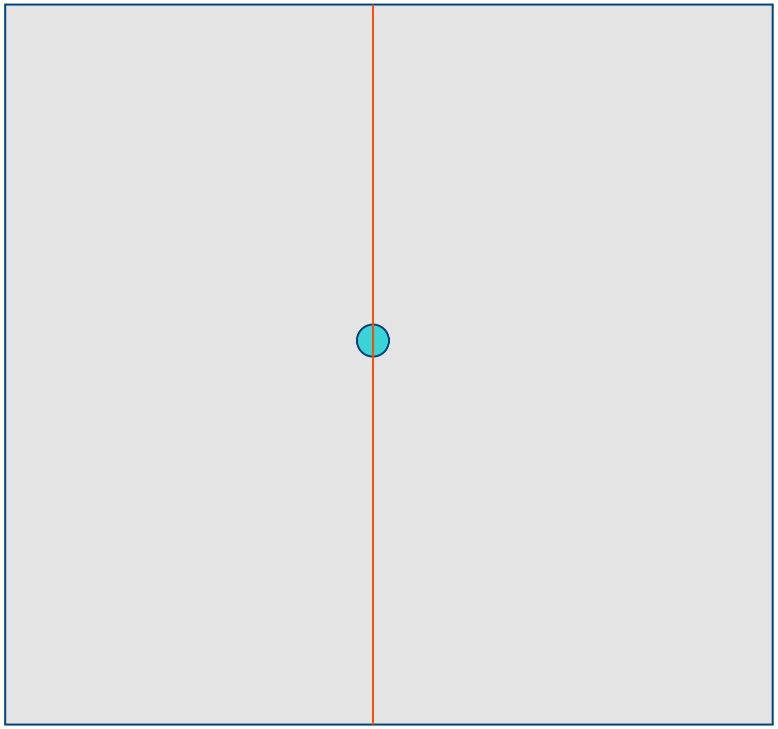
- Deficiencies of quadtree:
  - each node requires k comparisons
  - each leaf contains k null pointers
  - node size gets larger as k increases

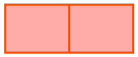
KD-trees

- Deficiencies of quadtree:
  - each node requires k comparisons
  - each leaf contains k null pointers
  - node size gets larger as k increases
- Solution: KD-tree
  - the tree is binary whatever k is!!!
  - each node has two pointers only

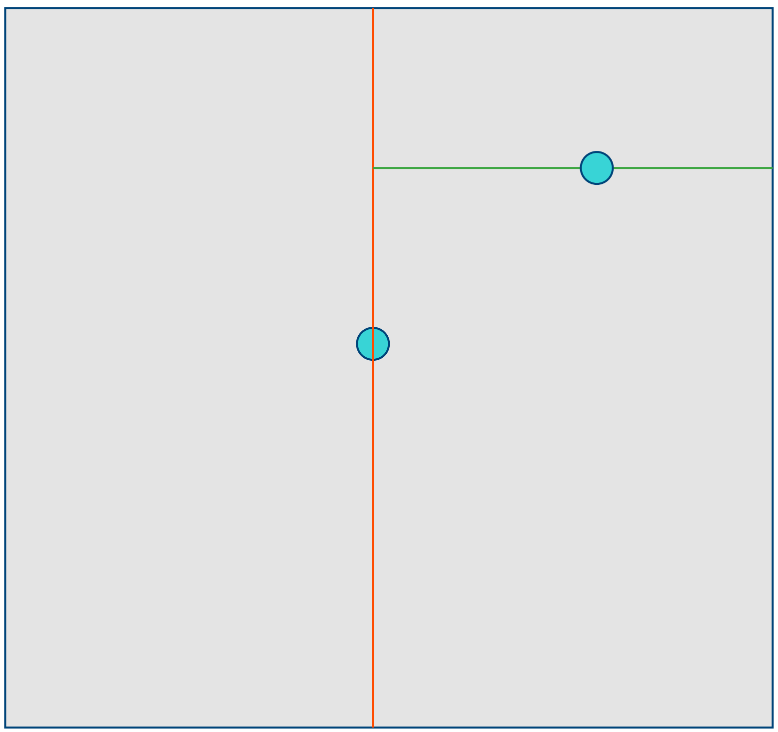
KD-trees

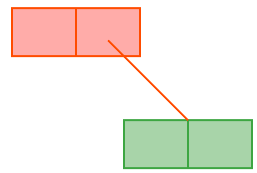
KD-trees



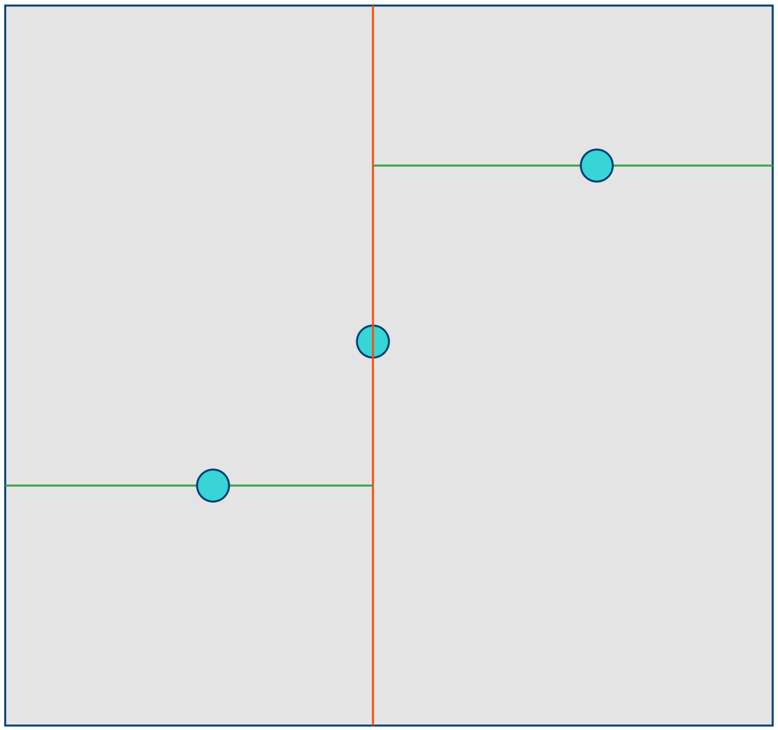


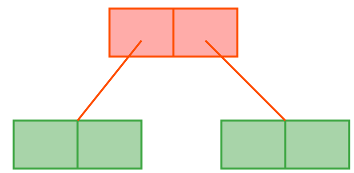
KD-trees



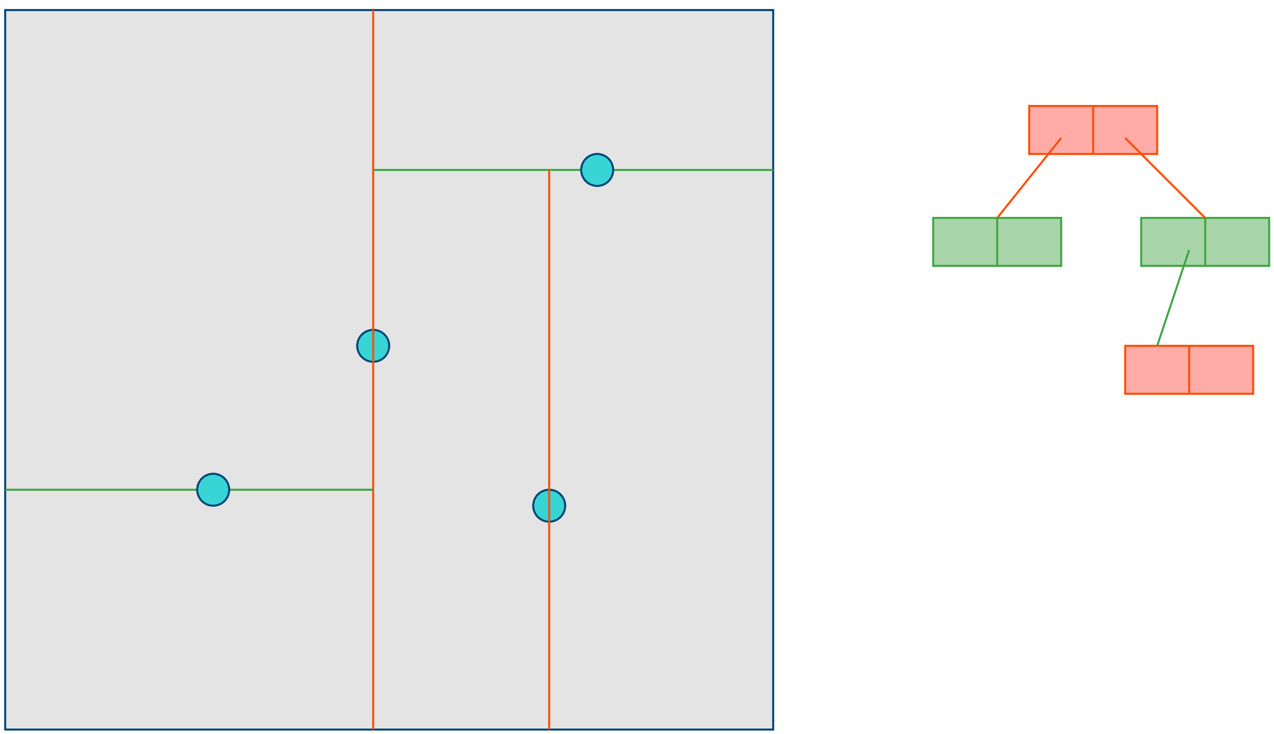


KD-trees

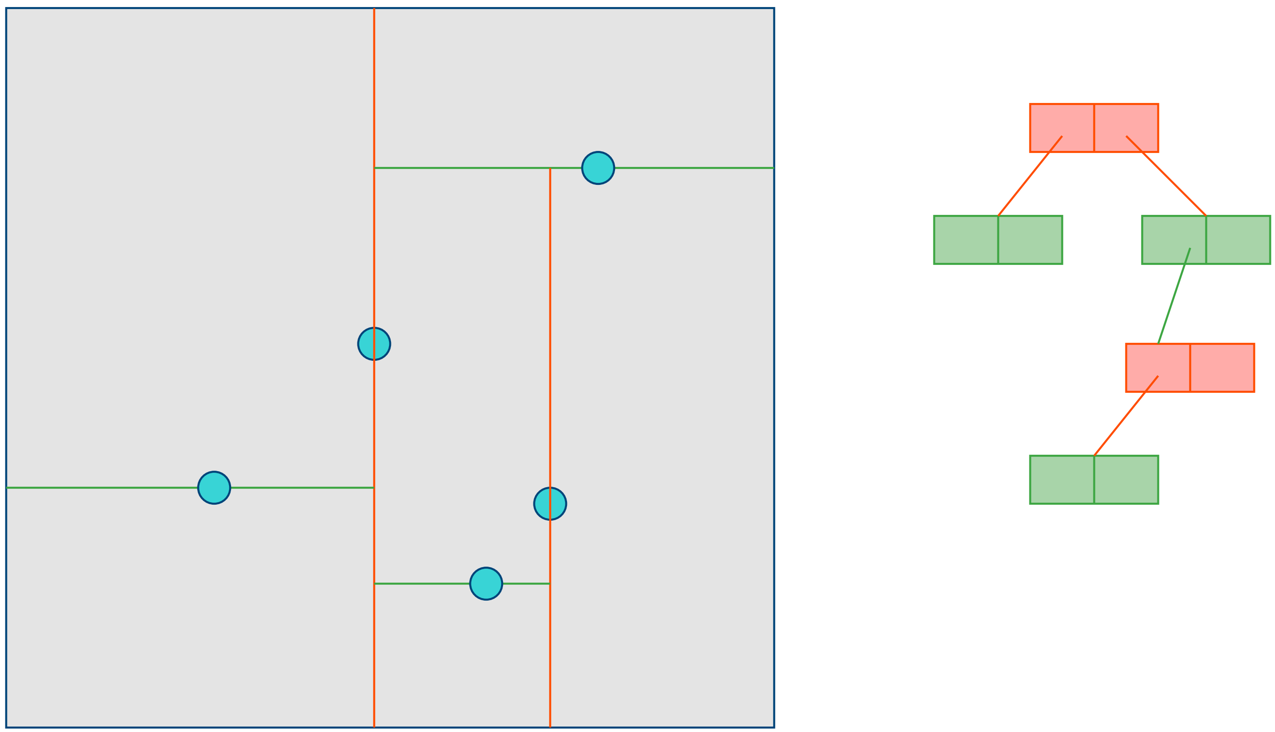




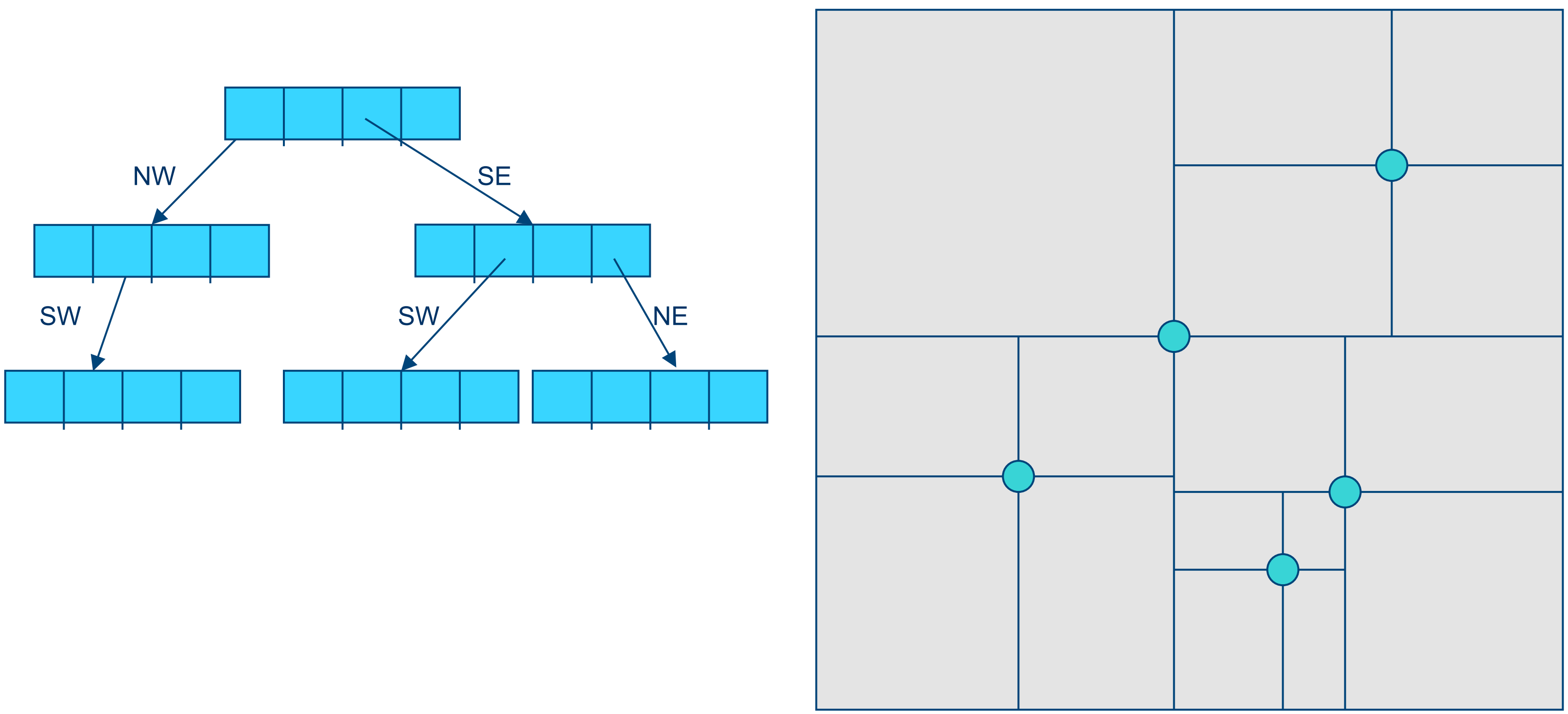
KD-trees



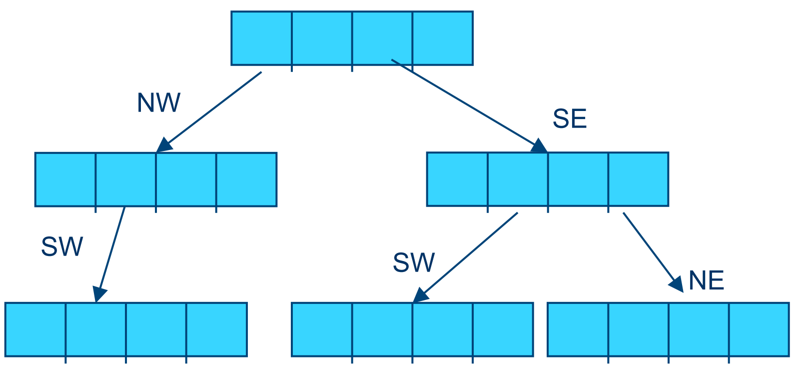
KD-trees

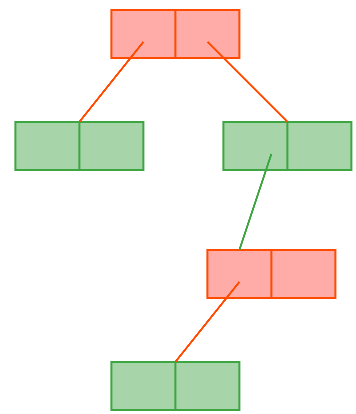


KD-trees

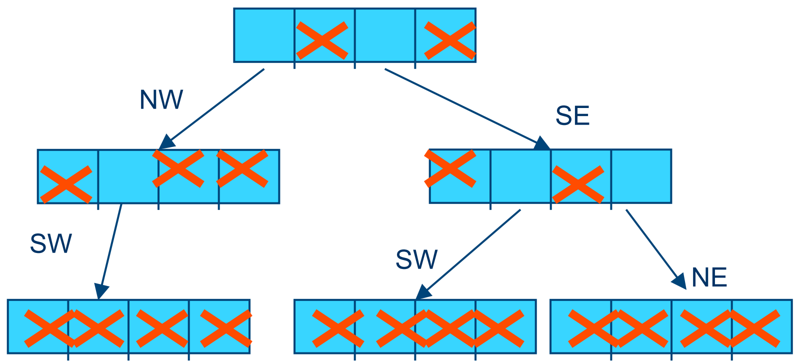


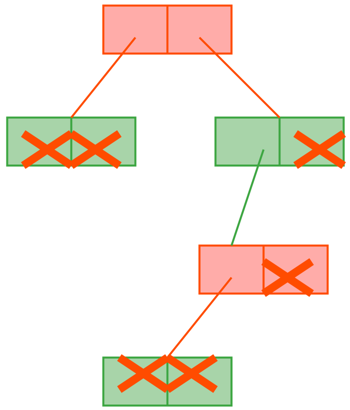
KD-trees





KD-trees





11

- Deletion in k-d trees:
  - delete node A.
  - if both subtrees are empty delete A
  - otherwise
    - find a suitable replacement node B in one of the subtrees of A.
    - recursively delete B
    - replace A with B.

Adaptive k-d trees.

- all ~~items~~ data are stored at the leaves
- the split dimension is chosen in a way that the spread is maximized
- split is performed at the median or mean.
  - deletion is hard.

R-trees

- R-trees are used to store *two* dimensional rectangle data.
- They can be easily generalized to higher dimensions.
- R-trees themselves generalize the well known B-trees.

Node capacity

- Each node in a R-tree can contain upto N rectangles.
- But in addition, each node must contain *at least* N/2 rectangles.
- We will assume henceforth that N >= 4.

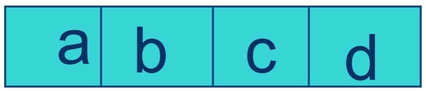
Node structure

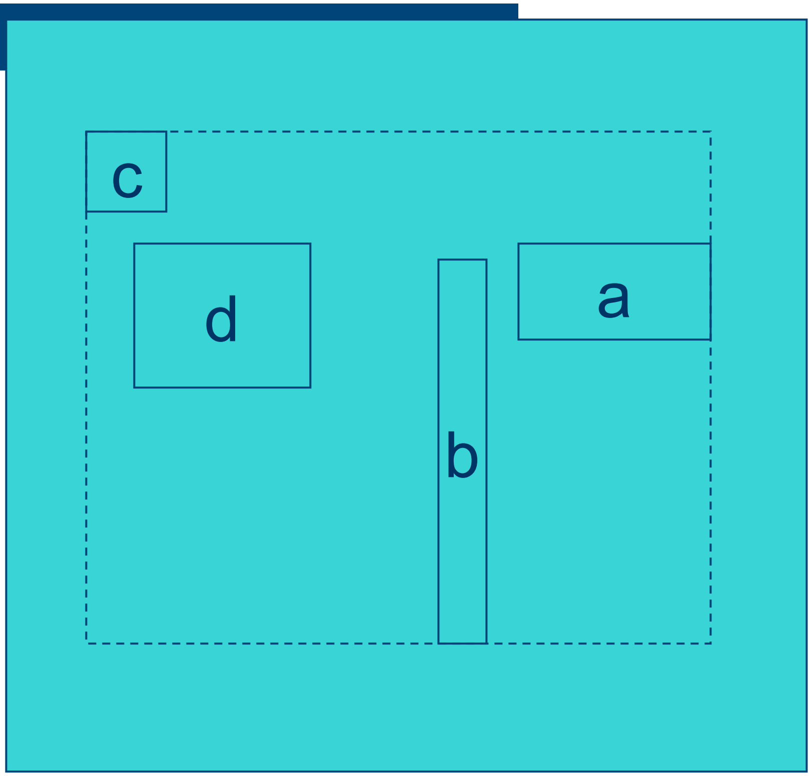
- Each node has between N/2 and N rectangles.
- Like a B-tree:
  - All leaves are at the same level
  - Root has at least two children unless it's a leaf

Node properties

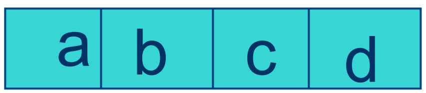
- Each node *implicitly* represents a region.
- Root represents the whole space.
- The region of a node N, N.reg, is the bounding box of the rectangles stored at that node.
- Unlike quadtrees, it is possible for regions of siblings to intersect.

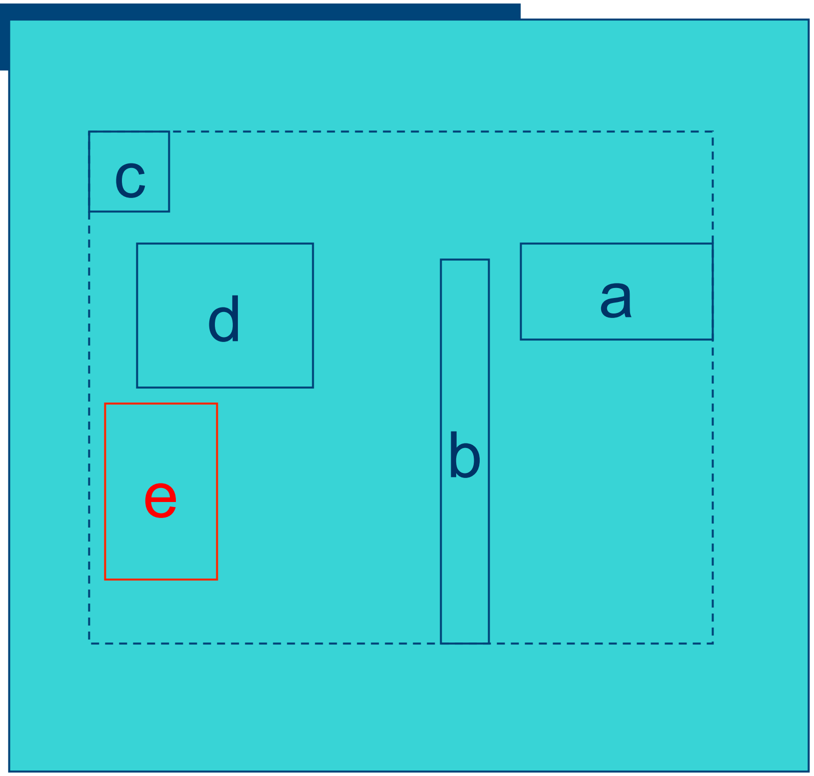
Example R-tree





Example R-tree

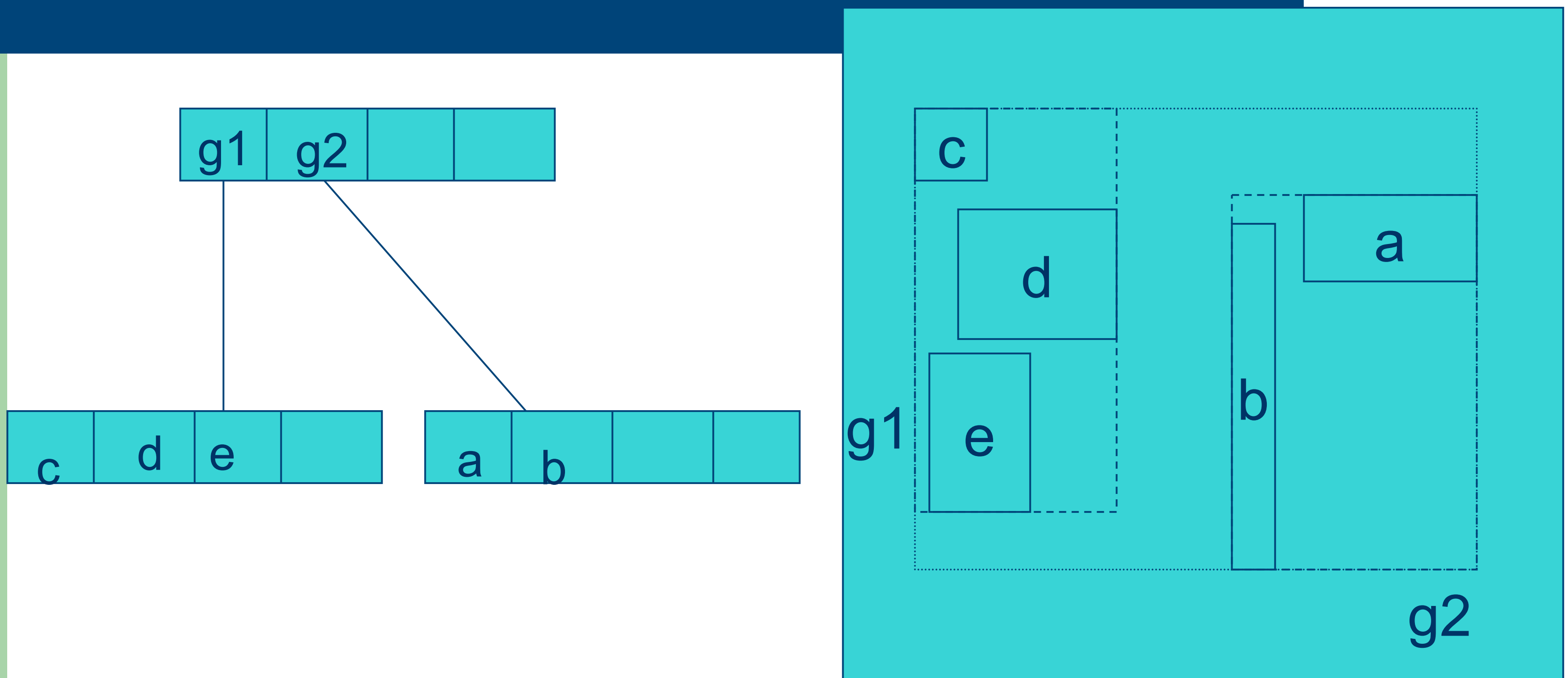




No space in root ! Must split

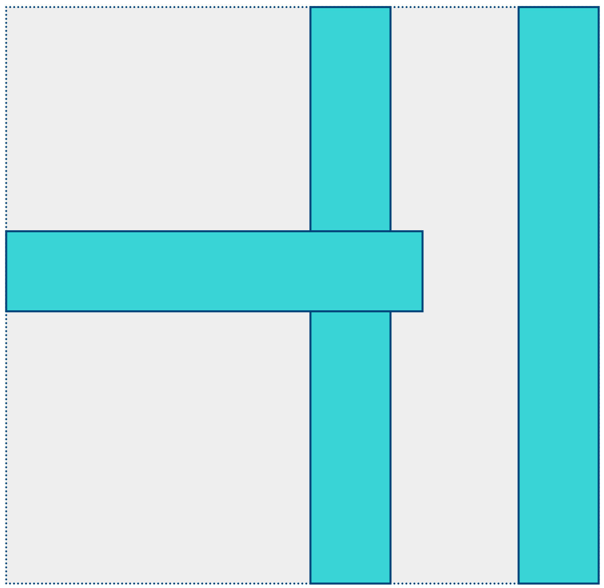
K. Selcuk Candan (CSE515)

Example R-tree

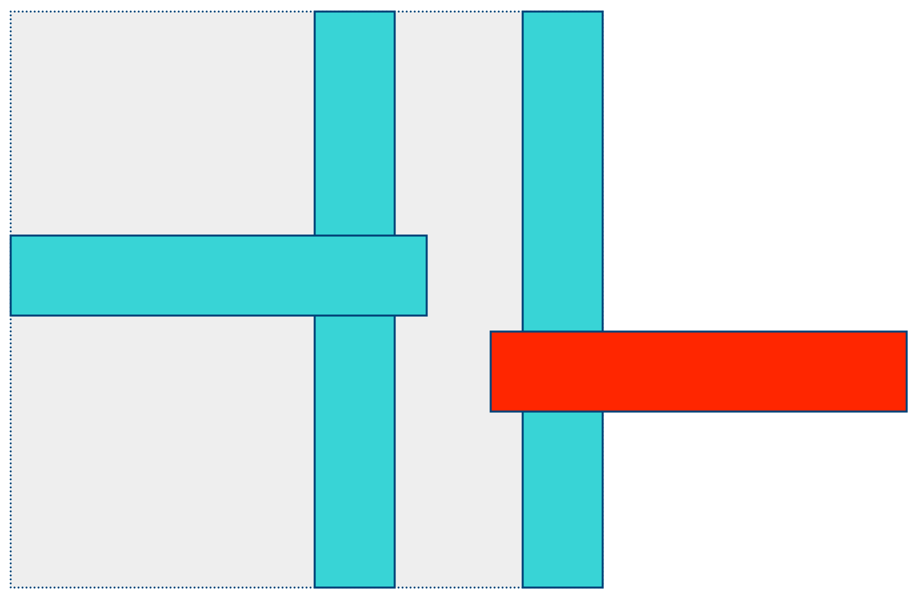


Minimize the sum of the areas of the two BRs.

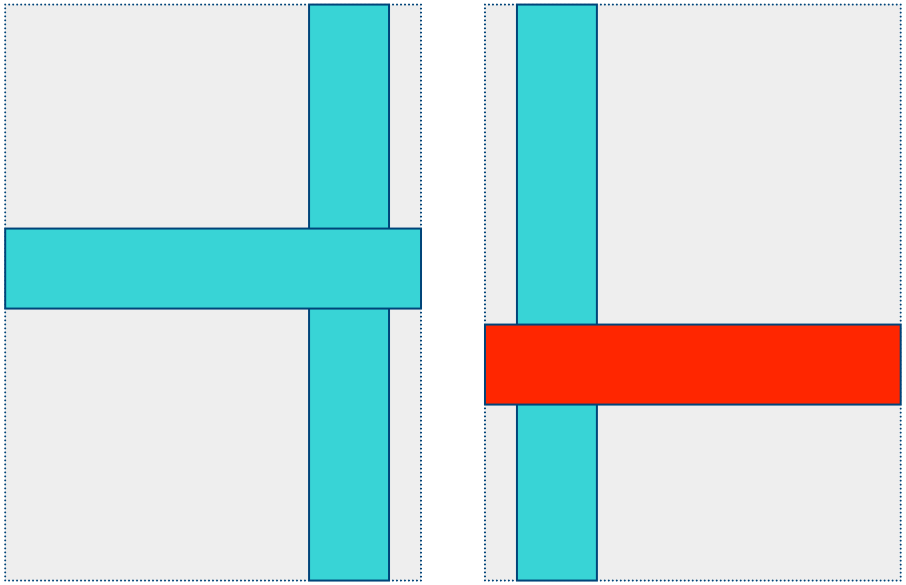
Example: Let max size be 3



How do we split???

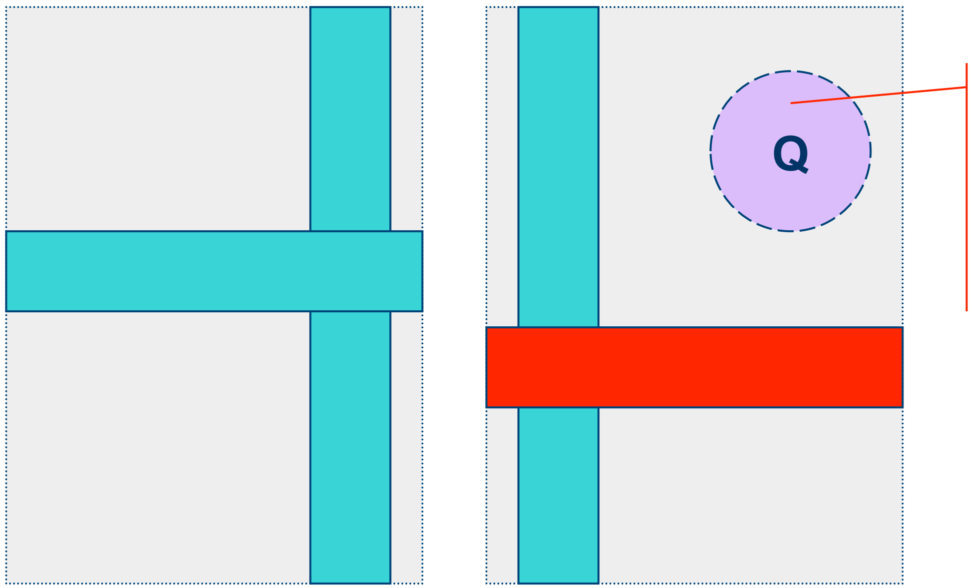


How do we split???



Minimize overlap of the BRs

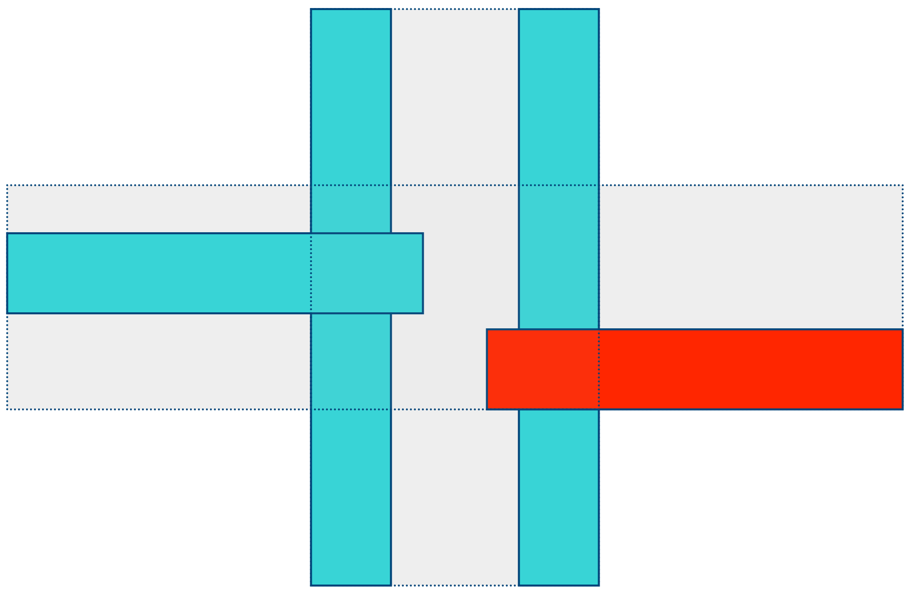
How do we split???



This search range can not be quickly pruned

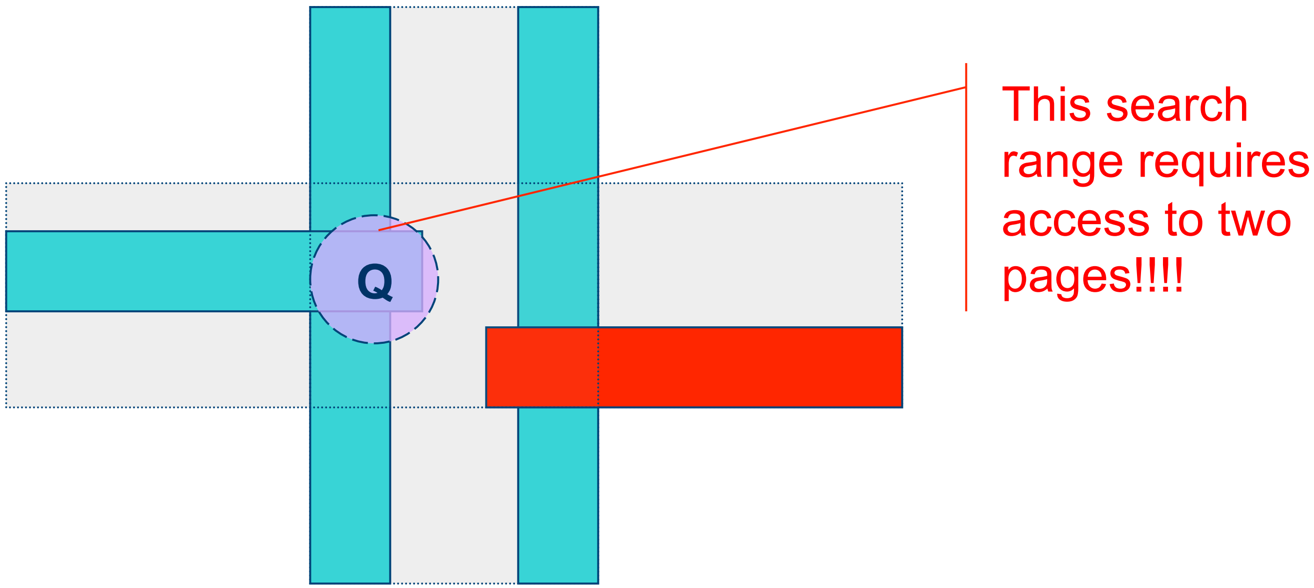
Minimize overlap of the BRs

How do we split???



Minimize total area

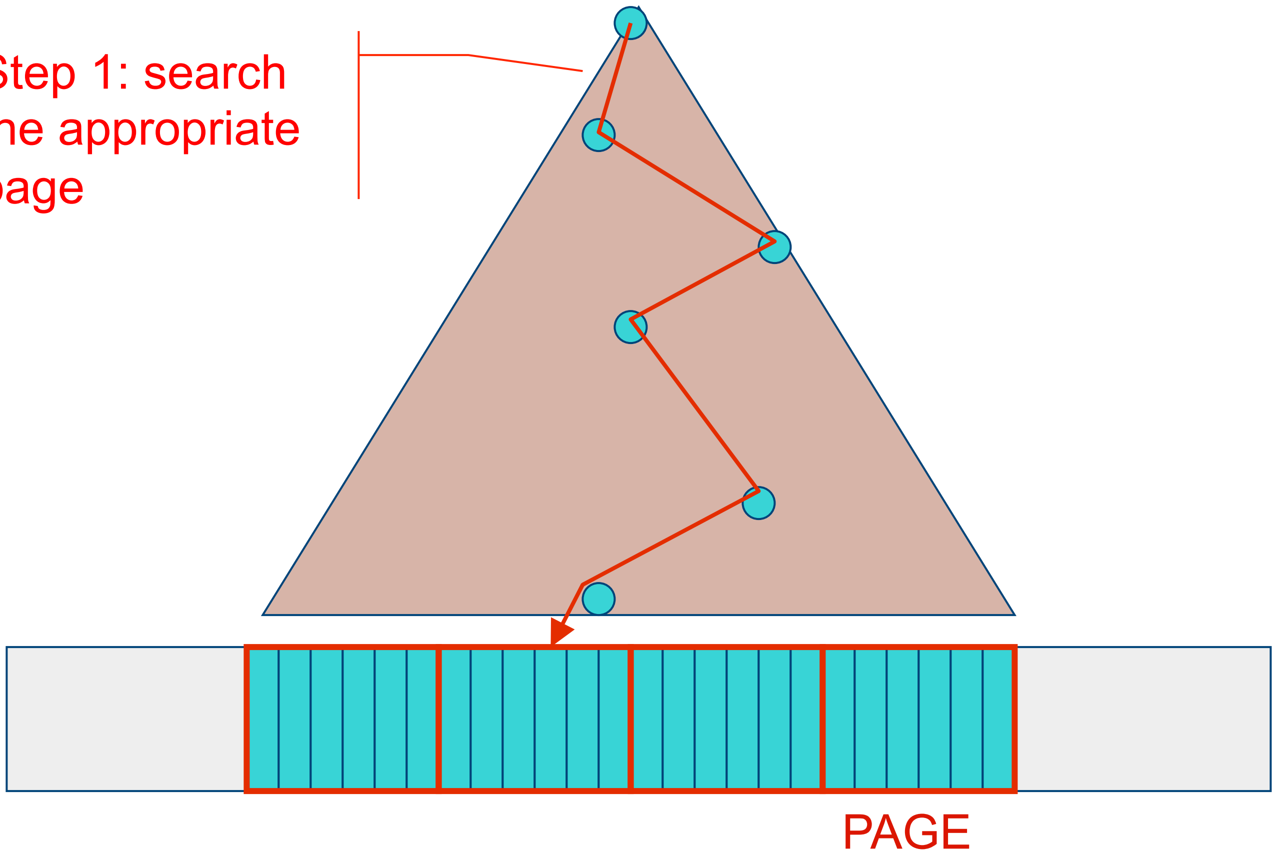
How do we split???



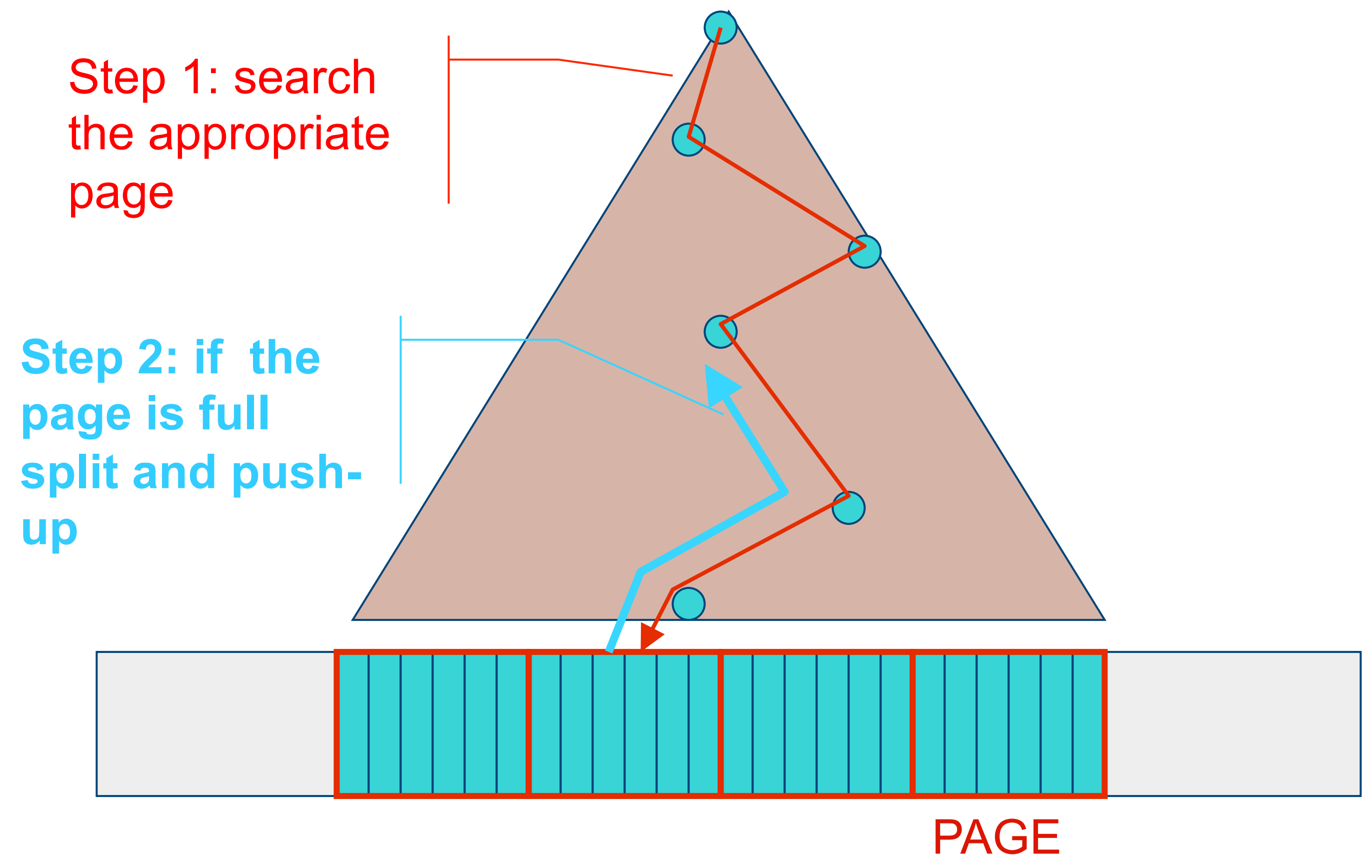
Minimize total area

Insertion (similar to B-trees)

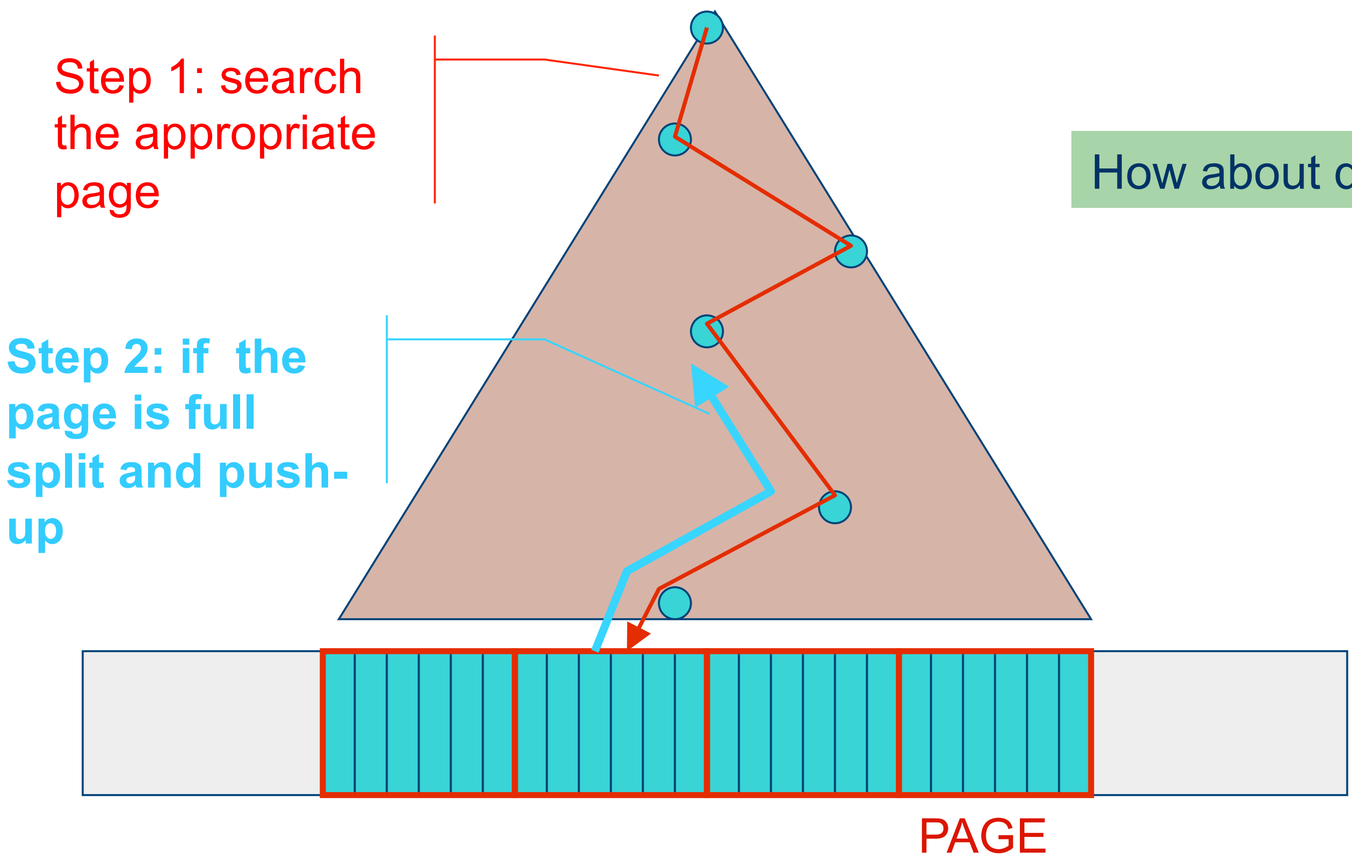
Step 1: search the appropriate page



Insertion (similar to B-trees)



Insertion (similar to B-trees)

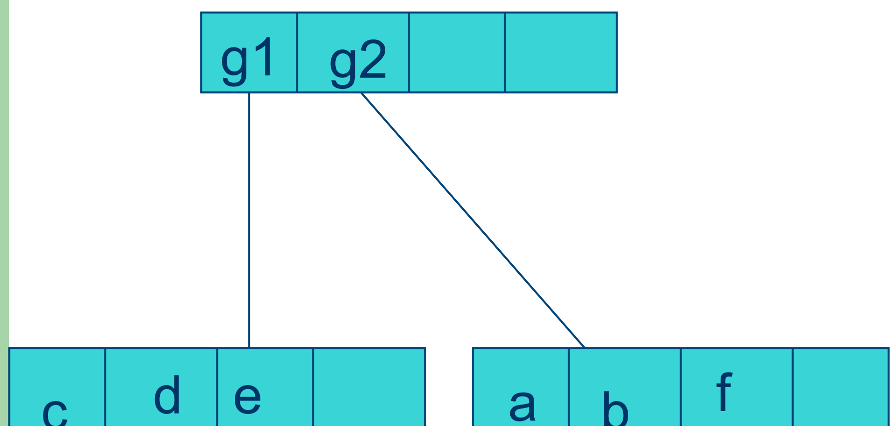


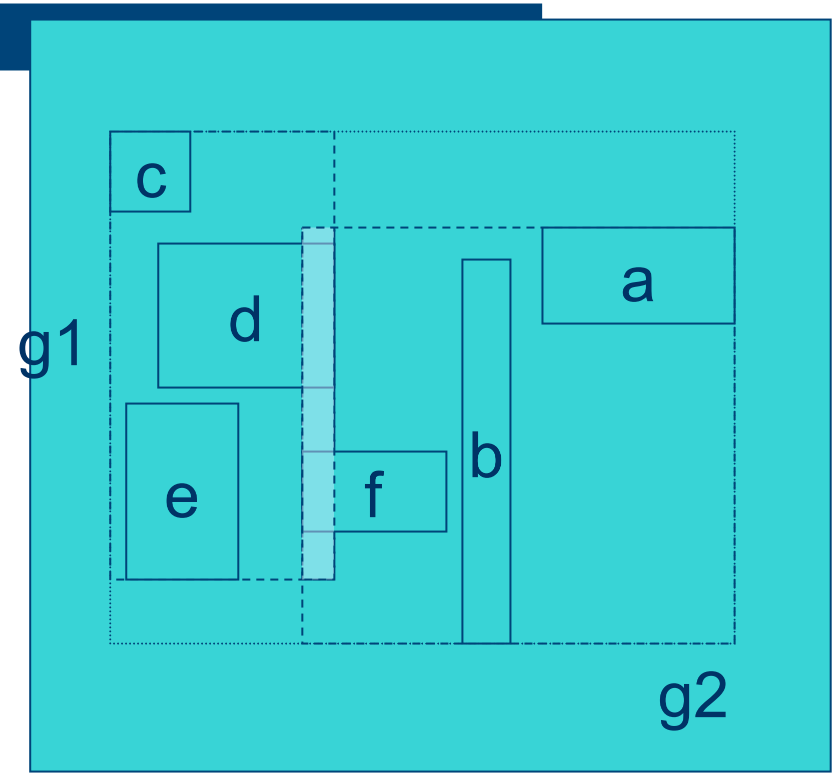
How about deletion??

R+-tree

- Overlaps are bad…..
- ..so, let's eliminate overlaps

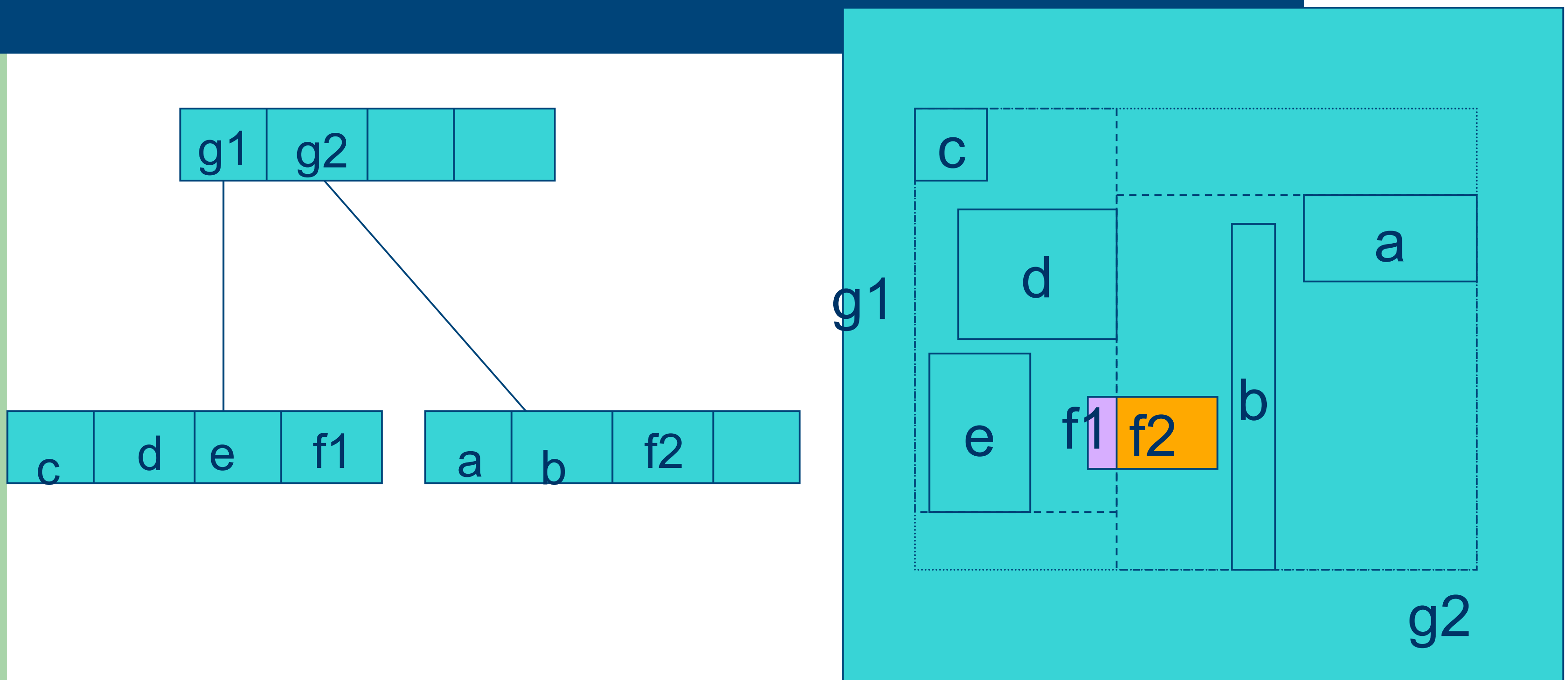
Overlap in R-tree





The two BRs are overlapping.

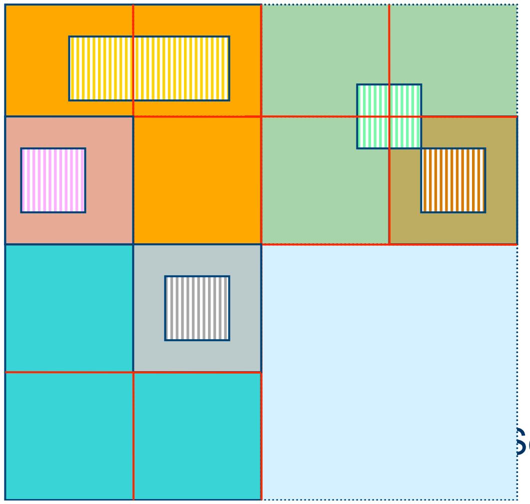
No-overlap in R+-tree



The two BRs are not-overlapping.

Other range/region index structures

- Range-tree, 2D-range tree
  - Precise, too much overhead
- MX-CIF quadtree
  - Regular division
  - Each rectangle is associated with the quadtree-page which covers it entirely



TV trees (telescopic vector trees)

(Lin, Jagadish, Faloutsos, VLDB Journal, 1994)

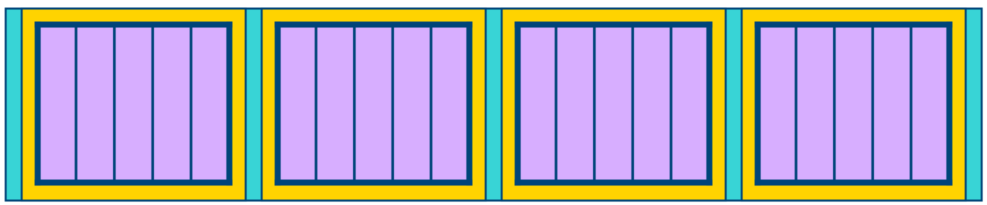
- Dimensionality curse: R-trees do not work for large numbers of dimensions
- Idea:
  - not all features are equally important
  - order features based on importance (discrimination power)
  - use as little features as possible
  - “contract” and “extend” feature vectors based on need

Cost of a dimension

- Every rectangle has to have values describing all its dimensions

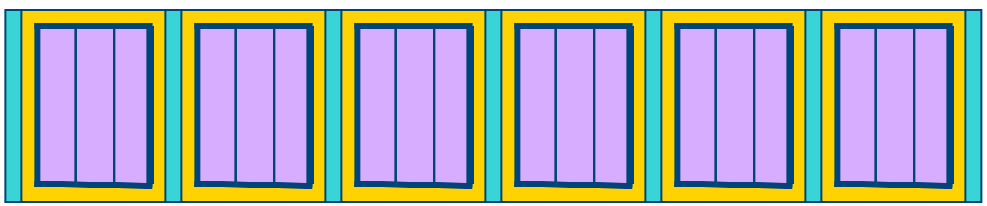
Cost of a dimension

- Every rectangle has to have values describing all its dimensions



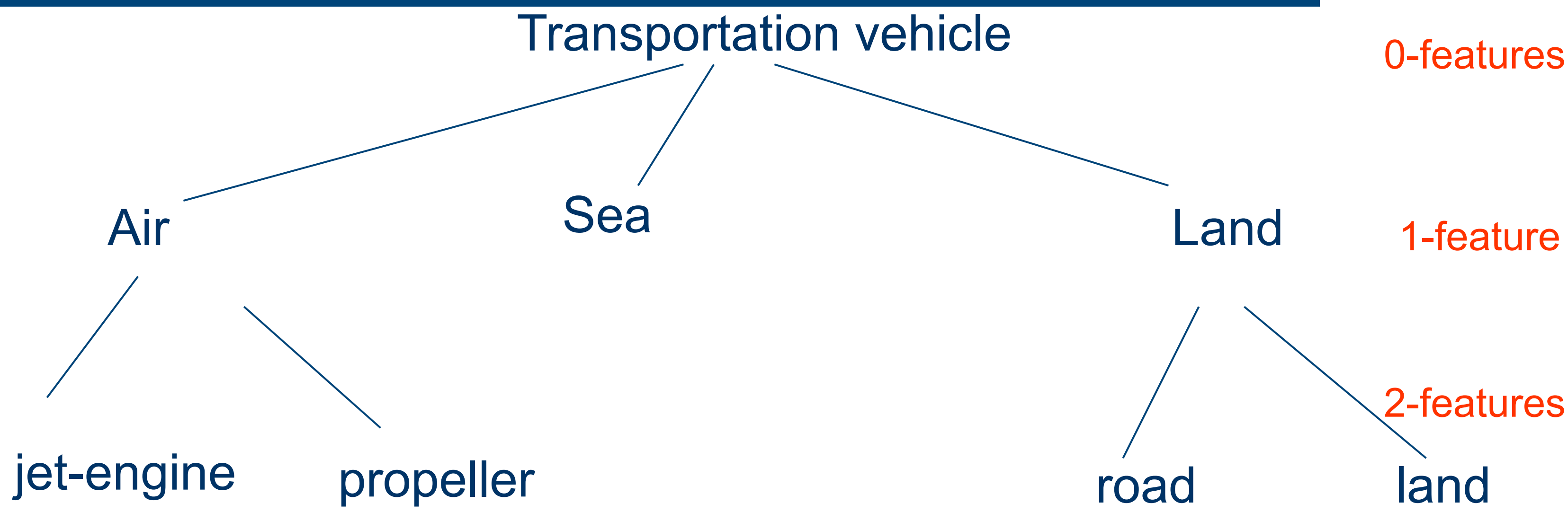
Disk Page

vs.



Disk Page

Intuition



Classification requires less features at the higher levels than it uses at the lower levels

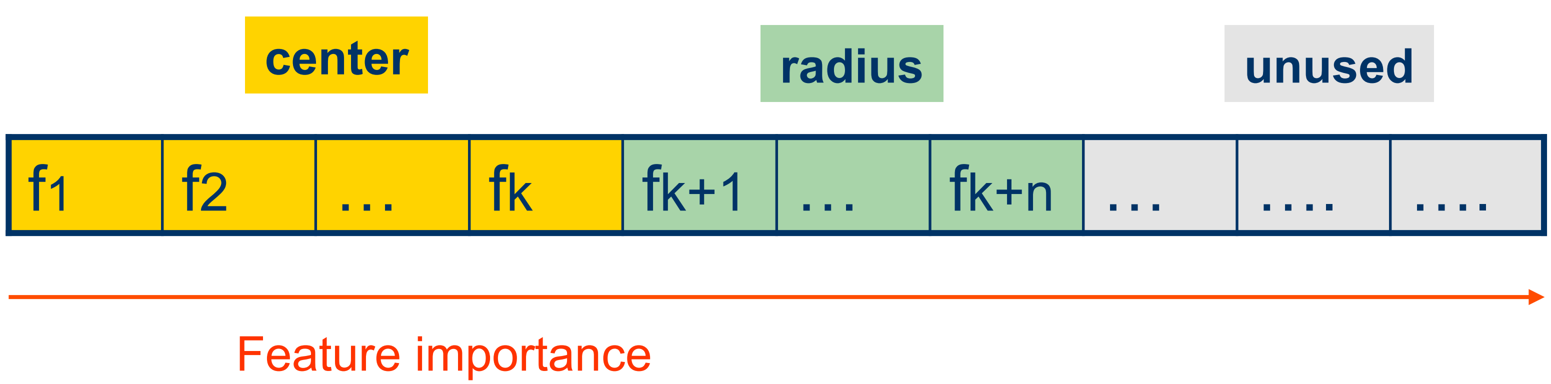
TV-trees

- Hierarchical
  - Leaves: objects (documents)
  - Internal nodes: Minimum Bounding Regions
    - Higher fan-out at the root
    - Lower fan-out at the leaves (or lower levels)

Node structure in TV-trees

- In R-trees, every node is a hyper-rectangle

- In TV-trees, every node has
  - a center (in k-dimensions)
  - a radius (defined in n-dimensions)

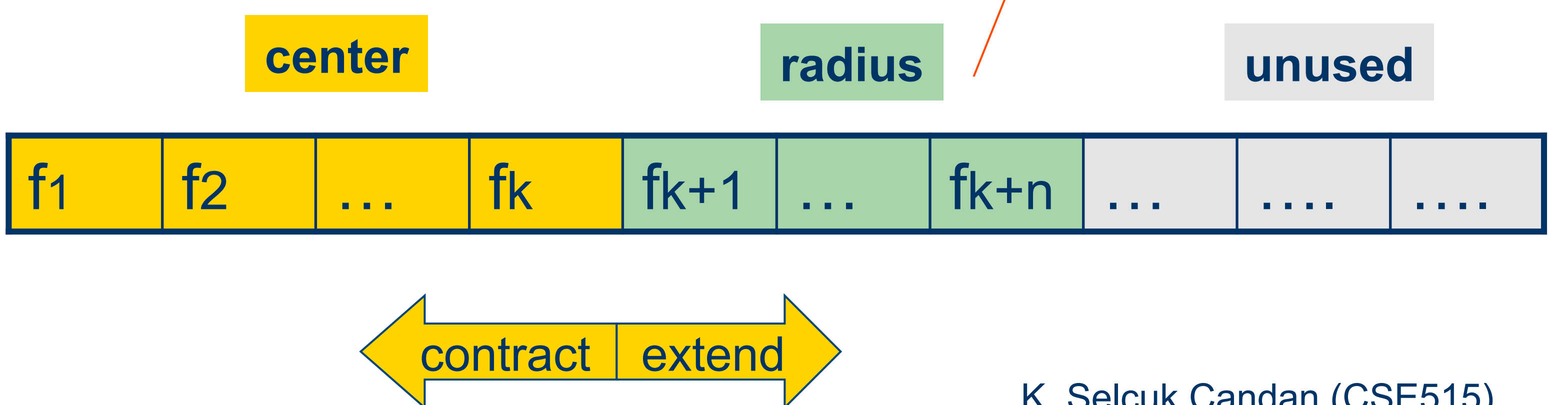


Node structure in TV-trees

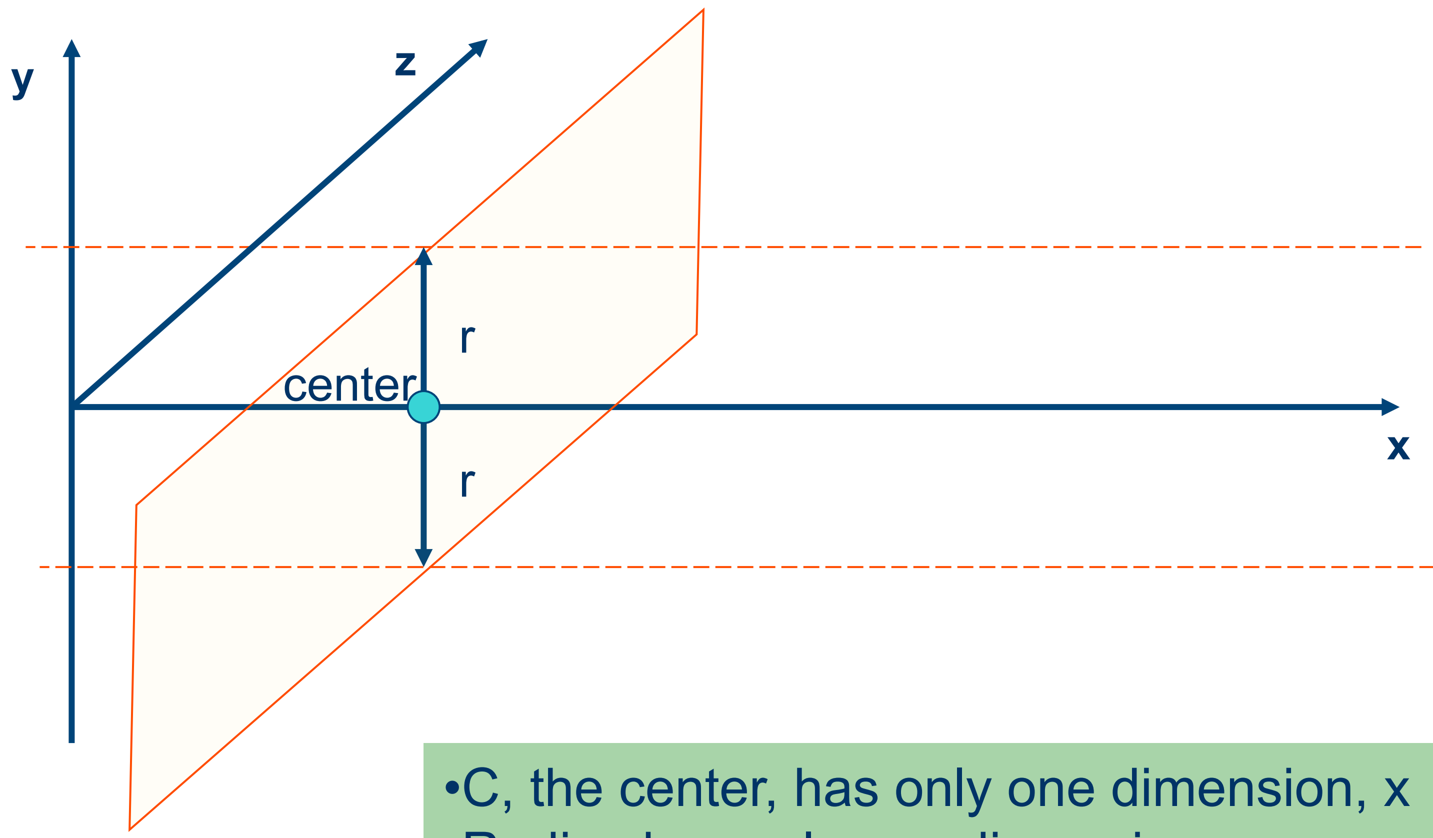
- In R-trees, every node is a hyper-rectangle

- In TV-trees, every node has
  - a center (in k-dimensions)
  - a radius (defined in n-dimensions)

n stays constant!!!

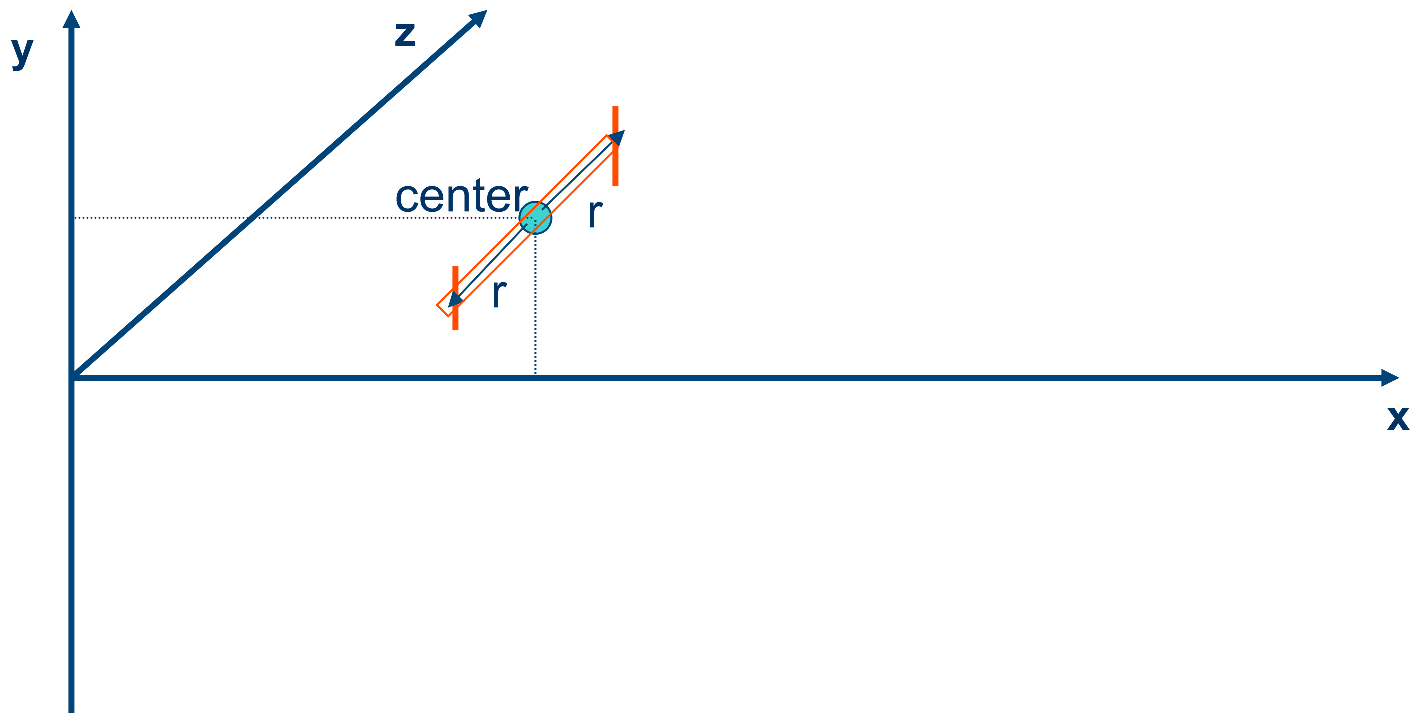


TV trees: example



- C, the center, has only one dimension, x
- Radius has only one dimension, y
- .....any z is okay (z is unused)

TV trees: extension example



- C, the center, has only two dimensions, x,y
- Radius has only one dimension, z
- ....any z is not okay!!!!!

Drawback

- Information about the behaviour of single attributes, e.g., their selectivity, is required

X-tree

- Like R-trees, but
  - change the page size based on the depth to ensure that there is larger fanout higher in the tree structure
- A larger page size means multiple disk pages that are consecutively stored
  - so, no “page seek” penalty during disk access.

Dimensionality curse

- Exponential growth in the number of pointers needed, wasted storage,
- Exponential suqueries (quadtrees)
- Larger MBRs means smaller fanout in trees and this is bad

Pyramid trees (Berchtold, Bohm, Kriegel, SIGMOD98)

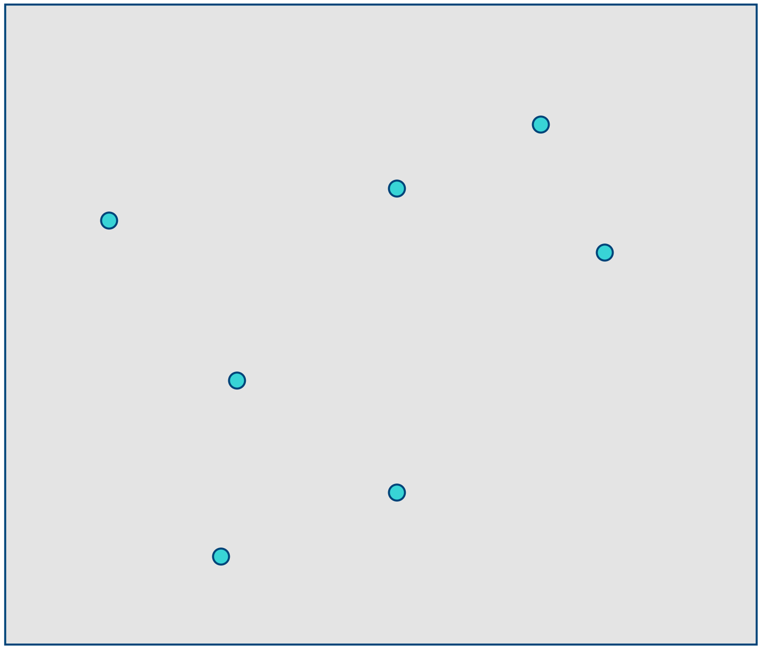
- Motivation: drawbacks of already existing multidimensional index structures
  - Querying and indexing techniques which provide good results on
    - low-dimensional data do not perform sufficiently well on multi-dimensional data (curse of dimensionality)
  - high cost for insert/delete operations
  - Poor support for concurrency control/recovery

Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves

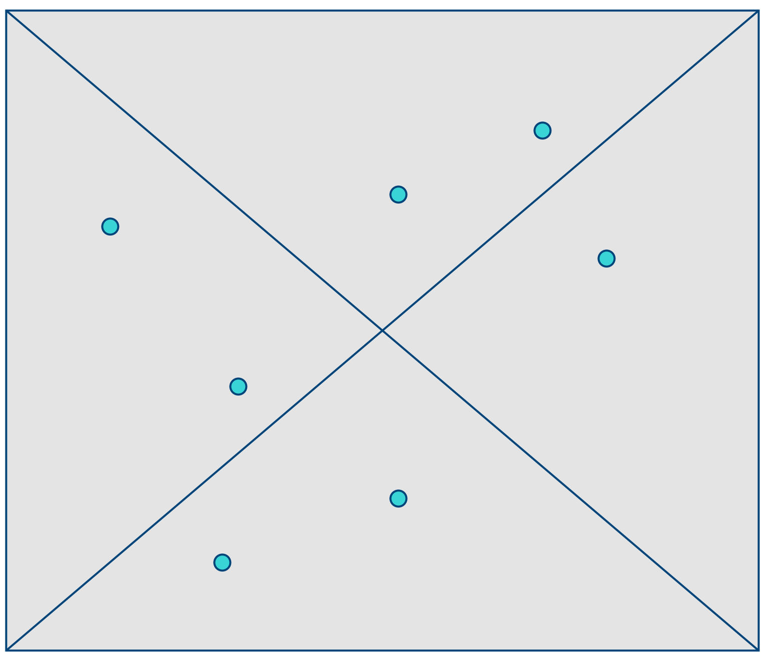
Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



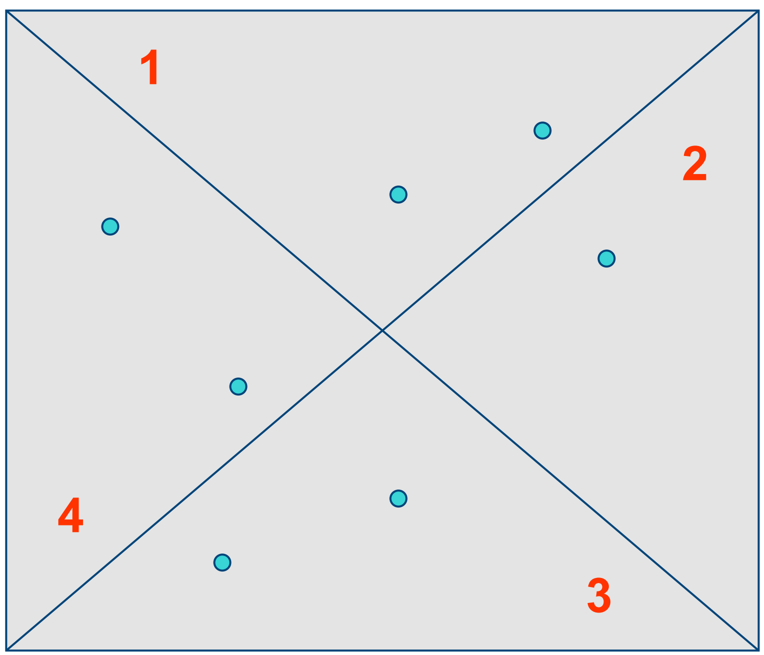
Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



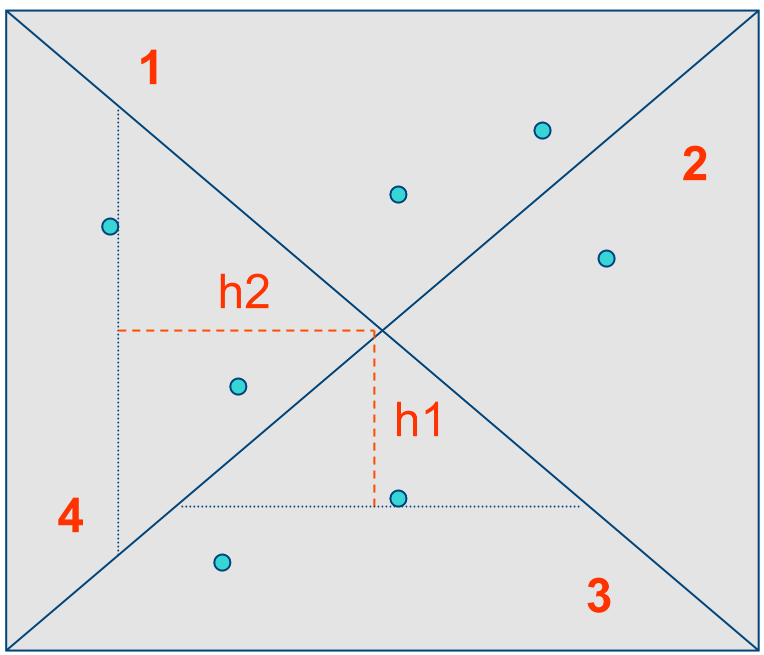
Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



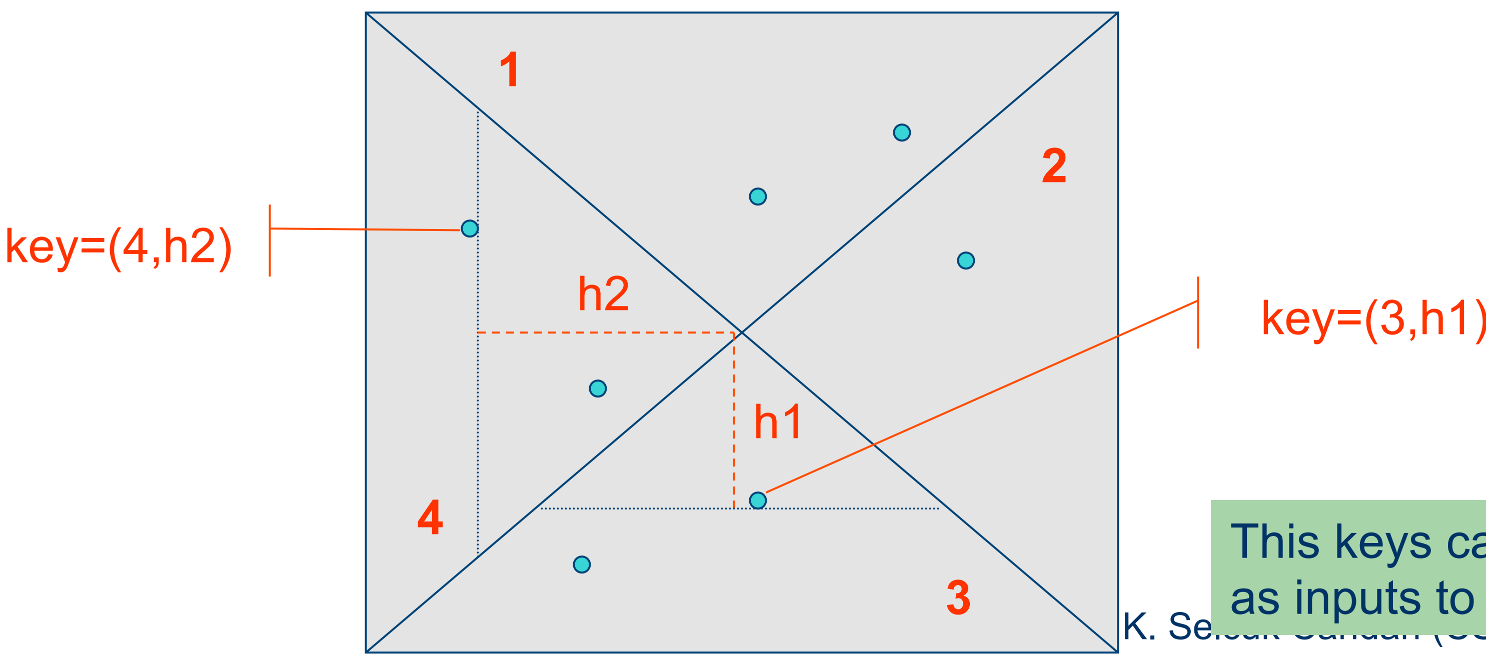
Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



Pyramid tree

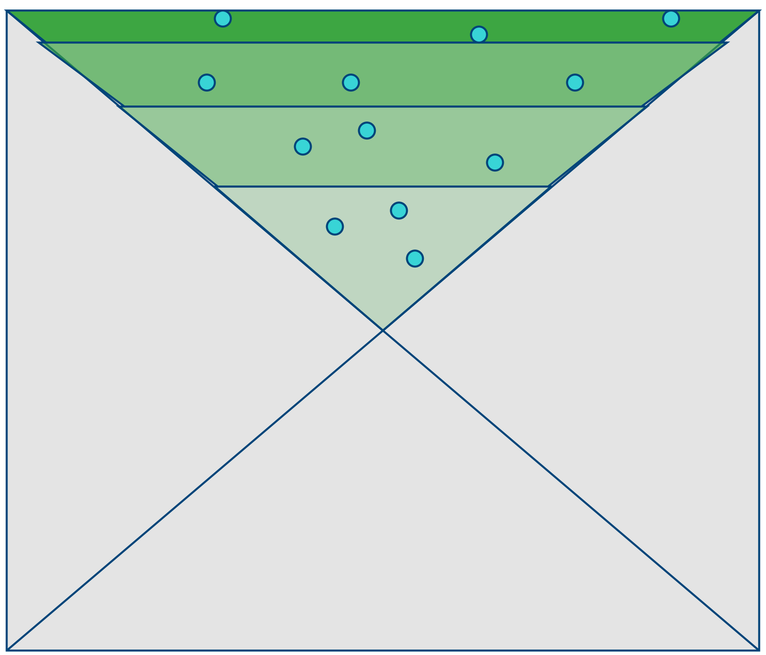
- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



This keys can be used as inputs to a B-tree!!!

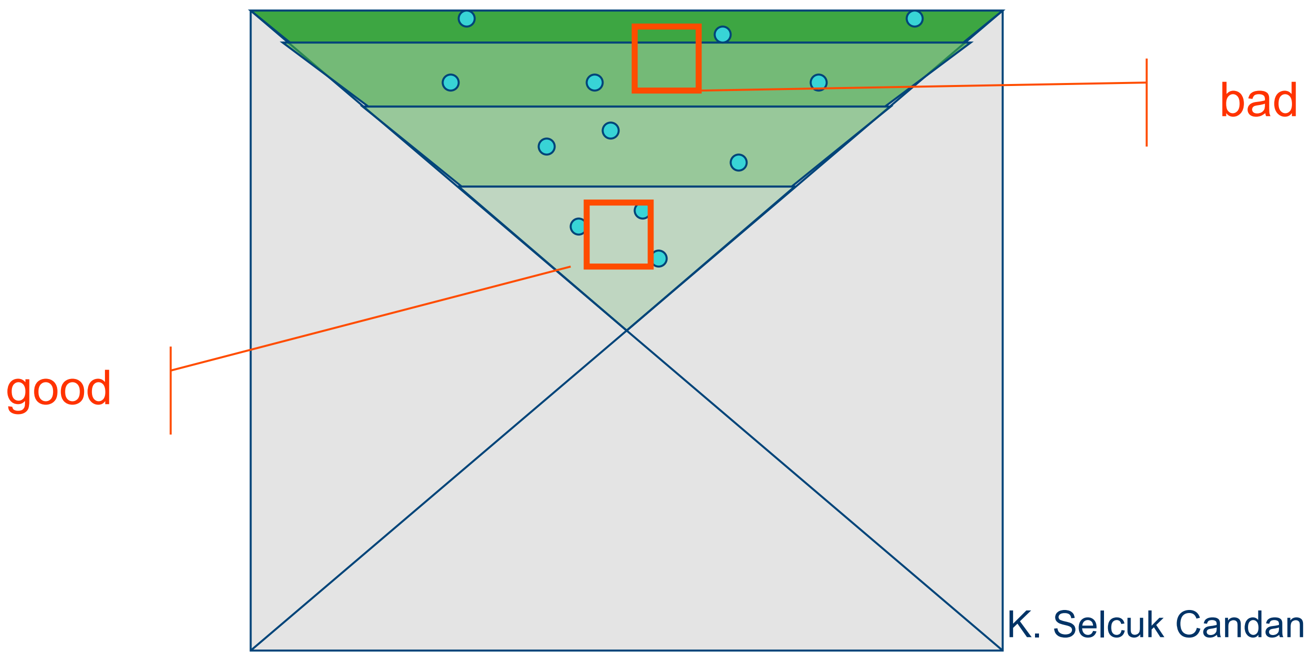
Pyramid tree

- If data is uniformly distributed, pages are likely to be of the same volume



Pyramid tree

- If data is uniformly distributed, queries likely to avoid thin pages, reducing the average access time



Other index structures

- Grids
- VA-files
  - extension of the grid idea..
- SR-, SS-trees
  - like R-trees
  - use spheres instead of rectangles
- X-trees
  - like R-trees
  - change the page size based on the depth

Nearest neighbor search

- ..no range given
  - first pick a (random) object $o \in D$ *and compute the distance dist(q, o)*….this is the first nearest neighbor candidate.
  - start a range search on the hierarchy using the range, $r = dist(q, o)$.
  - whenever you find a data object o such that $dist(q, o) < r$, where r is the current nearest neighbor range, pick o is as the new nearest neighbor candidate
    - Set $dist(q, o)$ as the new range, r'

Nearest neighbor search

- ..great, but in which order do we visit the pages?

- ..how can we prune the pages that we have not visited yet most effectively??

Nearest neighbor search

- Given q and MBR M
  - minDist(q,M)
  - minimum possible distance between the query and the objects contained within the MBR
  - minimum distance between q and any of the faces of M

- optimistic; yet minDist based ordering of the MBRs provides good pruning opportunities.

Nearest neighbor search

- Given q and MBR M
  - minMaxDist(q,M)
  - upperbound on the distances
  - minimum among
    - maximum distances
      - on the closest faces of M on each dimension

minDist(q,M) <= r <= minMaxDist(q,M)

Nearest neighbor search

- Cannot prune an MBR as long as

  minDist(q,M) <= r <= minMaxDist(q,M)

- downward pruning: discard M if there exists M' s.t.

  minDist(q,M) > minMaxDist(q,M')
- downward pruning: prune candidate object o if there exists M s.t.

  dist(q,o) =r > minMaxDost(q,M)
- upward pruning: M is discarded if the current candidate is s.t.

  minDist(q,M) > r = dist(q,o)

Nearest neighbor search

- What is we are looking for more than one, say *k*, nearest neighbors?
  - Maintain a list of *k* candidates in the memory
  - Always prune the search space using the current k^{th} best candidate
  - When you find an object better than the current k^{th} best candidate
    - Drop the current k^{th} best candidate
    - Include the new object in the list of *k* candidates
    - Identify the new k^{th} best candidate

Hashing for nearest neighbor search

- Hashing generally works for "equality searches"
- ..can we use "hashes" for nearest-neighbor searches???

- ….if they are locality sensitive, then "yes"!

Locality Sensitive Hashing (LSH)

- What is "locality sensitive hashing"?
  - ...a "grid" is a locality sensitive hash
  - ...a space filling curve is a locality sensitive hash

  More specifically, these are deterministic functions that tend to map nearby points to the same or nearby values.
- Can we develop randomized locality sensitive hashes?

Locality Sensitive Hashing (LSH)

- Let *sim()* be a similarity function
- A locality sensitive hash corresponding to *sim()* is a function, *h()*, such that

$$prob(h(o1) = h(o2)) \quad = \quad sim(o1,o2)$$

- The challenge is to find the appropriate *h()* for a given *sim()*

Locality Sensitive Hashing (LSH)

- An LSH family, H, is (r, cr, P_1, P_2)-sensitive, if for any two objects o_i and o_j and for a randomly selected $h() \in H$
  - if $dist(o_i, o_j) \leq r$ then $prob\ (h(o_i) = h(o_j)) \geq P_1$,
  - if $dist(o_i, o_j) \geq cr$ then $prob\ (h(o_i) = h(o_j)) \leq P_2$ and
  - $P_1 > P_2$.

Locality Sensitive Hashing (LSH)

- Consider a (r, cr, P_1, P_2)-sensitive hash family, H
- Let's create L composite hash functions

$$g_j(o) = (h_{1,j}(o), \ldots, h_{k,j}(o)),$$

by picking $L \times k$ hash functions, $h_{i,j} \in H$, independently and uniformly at random from H.

Locality Sensitive Hashing (LSH)

- Let us be given $g_1()$ through $g_L()$ and database, D,
- Hash object o in D using $g_1()$ through $g_L()$ and include o in all matching hash buckets

$$g_1(o) = (h_{1,1}(o), \ldots, h_{k,1}(o)),$$

$$\ldots.$$

$$g_L(o) = (h_{1,L}(o), \ldots, h_{k,L}(o))$$

Locality Sensitive Hashing (LSH)

- Hash the query q in also using $g_1()$ through $g_L()$ and consider all objects in these hash buckets

$$g_1(q) = (h_{1,1}(q), \ldots, h_{k,1}(q)),$$

$$\ldots.$$

$$g_L(q) = (h_{1,L}(q), \ldots, h_{k,L}(q))$$

- Key result:
  - if $L = \log_{1-P_1^k}\delta$, then any object within range r is returned with probability at least $1-\delta$.

Locality Sensitive Hashing (LSH)

- Then, how do we create a *(r, cr, P_1, P_2)-sensitive* hash family, *H ??*

- ….depends on the underlying *sim()* or *δ()* function…

Locality Sensitive Hashing (LSH)

- Assume d-dimensional binary vector; e.g. (0,1,1,1,0,…, 1)
- Let δ() be the hamming distance (number of differing dimensions between two vectors)

- H contains all projections of the input point x on one of the coordinates; i.e., $h_i(x) = x_i$

Locality Sensitive Hashing (LSH)

- Let
  - p and q be two vectors in d-dimensional binary vector space
  - δ() is the hamming distance
  - H contains $h_i(x) = x_i$
- Note that *prob[h(q) = h(p)]* is equal to the fraction of coordinates on which *p* and *q* agree.
- Then, if we select
  - $P_1 = 1 - (r/d)$ and $P_2 = 1 - c(r/d)$
  - such that $c>1$

  we have $P_1 > P_2$.

Locality Sensitive Hashing (LSH)

- L1-distance in d-dimensional space:
  - pick a $w >> r$
  - impose a randomly shifted grid with cells of width w
    - pick random $s_1, s_2, \ldots, s_d$ in [0, w)
    - define $h_{s1, s2, \ldots sd}(x) = (\lfloor (x_1 - s_1)/w \rfloor, \ldots, \lfloor (x_d - s_d)/w \rfloor)$.

Locality Sensitive Hashing (LSH)

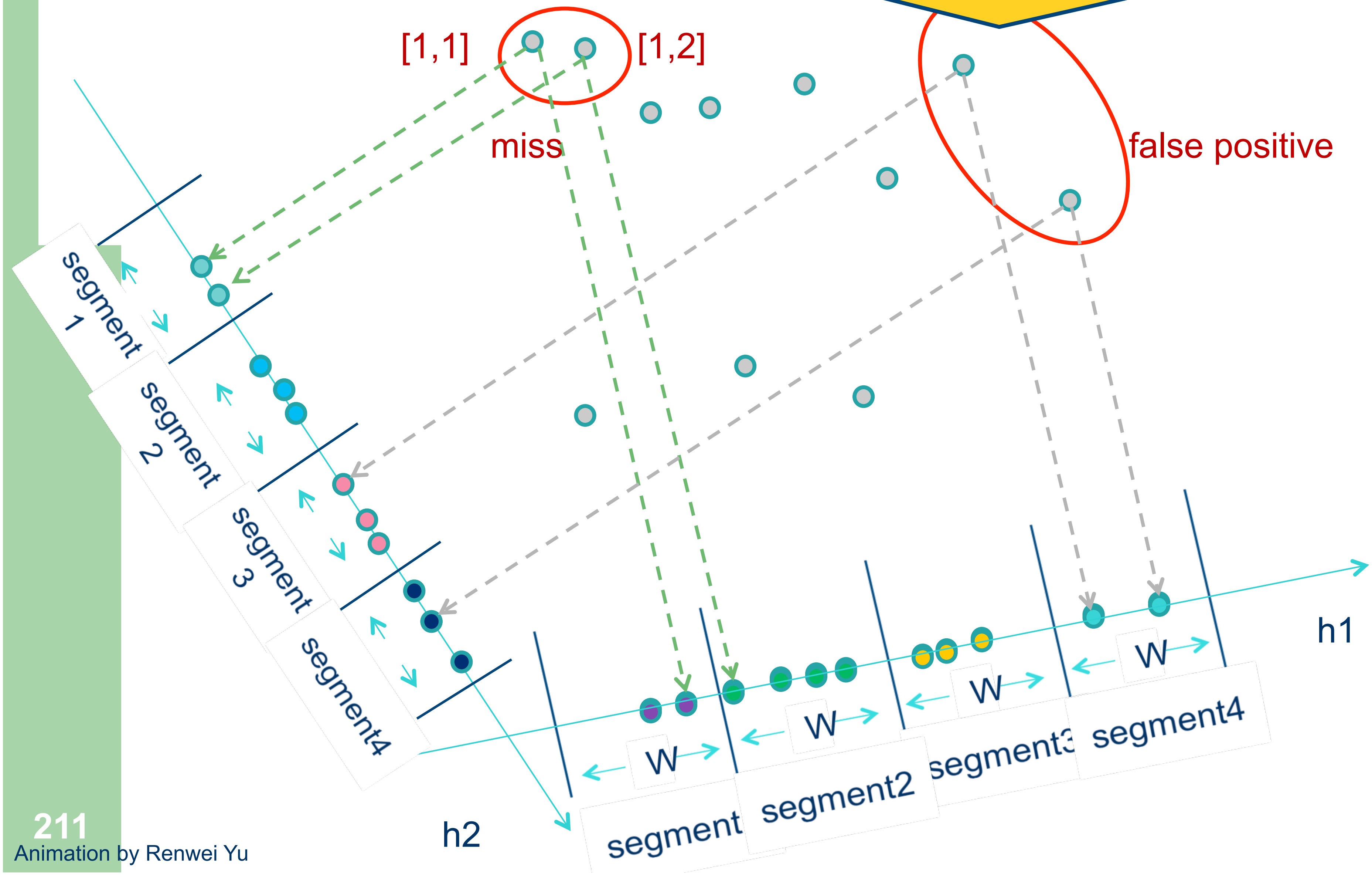
- Ls-distance in d-dimensional space:
  - pick a $w >> r$
  - pick a random projection, p, of the space onto a 1-dimensional line by picking each coordinate of p from the Gaussian distribution.
  - chop the line into segments of length w, shifted by a random value b in [0, w); i.e., given vector x

$$h_{r,b}(x) = \lfloor (p \cdot x + b)/w \rfloor,$$

To eliminate false positives, hash function of each table is the intersection of *multiple* element hashes h()

$g()=[h_1(),\ldots\ldots,h_c()]$

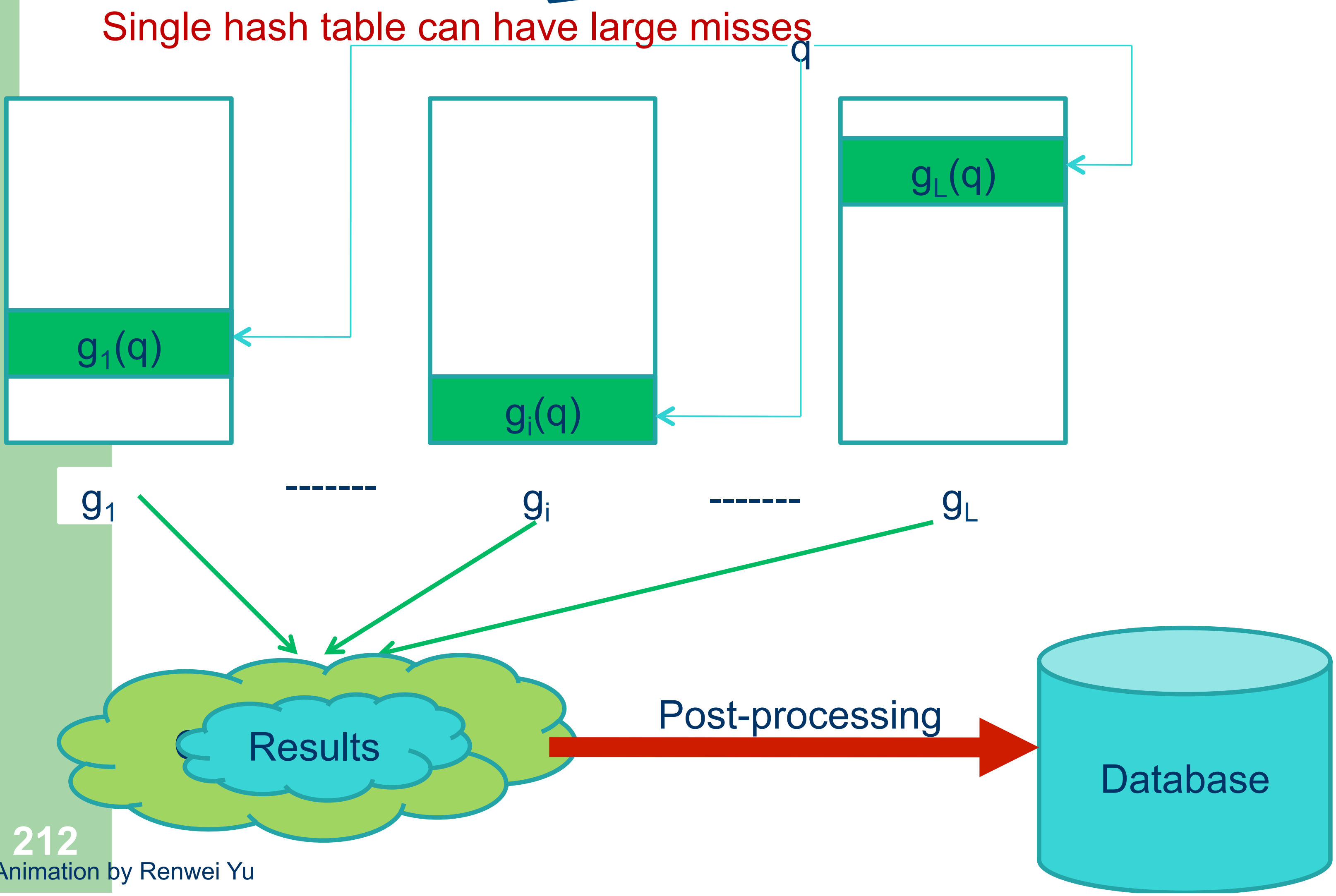
…but this can increase the number of misses!!!!!



Animation by Renwei Yu

To reduce misses, union of L hash tables are used

$G=\{g_1,\ldots\ldots,g_L\}$



Animation by Renwei Yu

Issues of Basic LSH

- Large number of tables to achieve good search quality
  - L>580 in [Buhler01]

- Impractical for large datasets, need reduce hash tables
  - Entropy-based LSH [Panigrahy06]
  - Multi-Probe LSH [Qin07]

..this slide by Renwei Yu

Issues of Basic LSH (continued…)

- Data dependent parameters need hand-tuning
  - Different bucket size is required to collect enough candidates to answer different KNN queries
  - LSH-Forest [Bawa05]
  - Multi-Probe LSH can also be self-tuning to answer different KNN queries

..this slide by Renwei Yu