ΠΟΛΥΔΙΑΣΤΑΤΑ ΕΥΡΕΤΗΡΙΑ

Γιάννης Κωτίδης

Που βρισκόμαστε

- Ευρετήρια ενός (ίσως σύνθετου) γνωρίσματος
  - Απλά ευρετήρια
  - B^+-trees
  - Κατακερματισμός
    - Στατικός
    - Δυναμικός (Extendible, Linear Hashing)
- Πολυδιάστατα ευρετήρια

Παράδειγμα

- Βρες τις εγγραφές της σχέσης Employee που ικανοποιούν τις συνθήκες

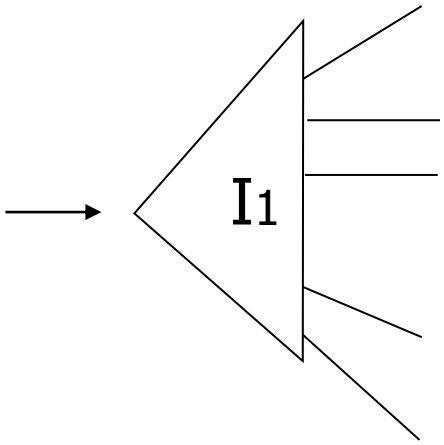
DEPT = “Toy” AND SAL > 50k

Παραδοχή

- Στη γενική περίπτωση, ένα ευρετήριο μου προσφέρει (με γρήγορο τρόπο) δείκτες (pointers) προς τη θέση όσων εγγραφών στο δίσκο ικανοποιούν μία συνθήκη

Στρατηγική Ι:

- Χρησιμοποίησε ευρετήριο το γνώρισμα Dept.
  - Από το ευρετήριο βρίσκω τις εγγραφές όπου Dept = “Toy” και στη συνέχεια ελέγχω αν SAL>50K



Τι γίνεται αν οι περισσότεροι υπάλληλοι δουλεύουν στο Dept=“Toy”, ενώ πολύ λίγοι από αυτούς έχουν SAL>50K?

Στρατηγική ΙΙ:

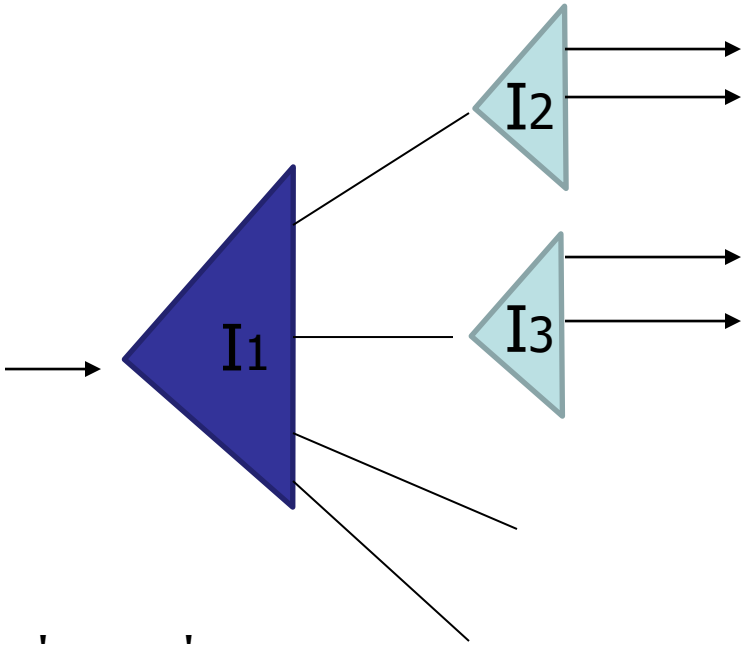
- Χρησιμοποίησε 2 ανεξάρτητα ευρετήρια
  - Πάρε την τομή των λιστών από pointers

SAL>50K

- η μία από τις δύο λίστες μπορεί να είναι μεγάλη (παρότι η τομή τους είναι μικρή)
- ψάχνω σε δύο ευρετήρια αντί για 1
- συχνά δεν υλοποιείται

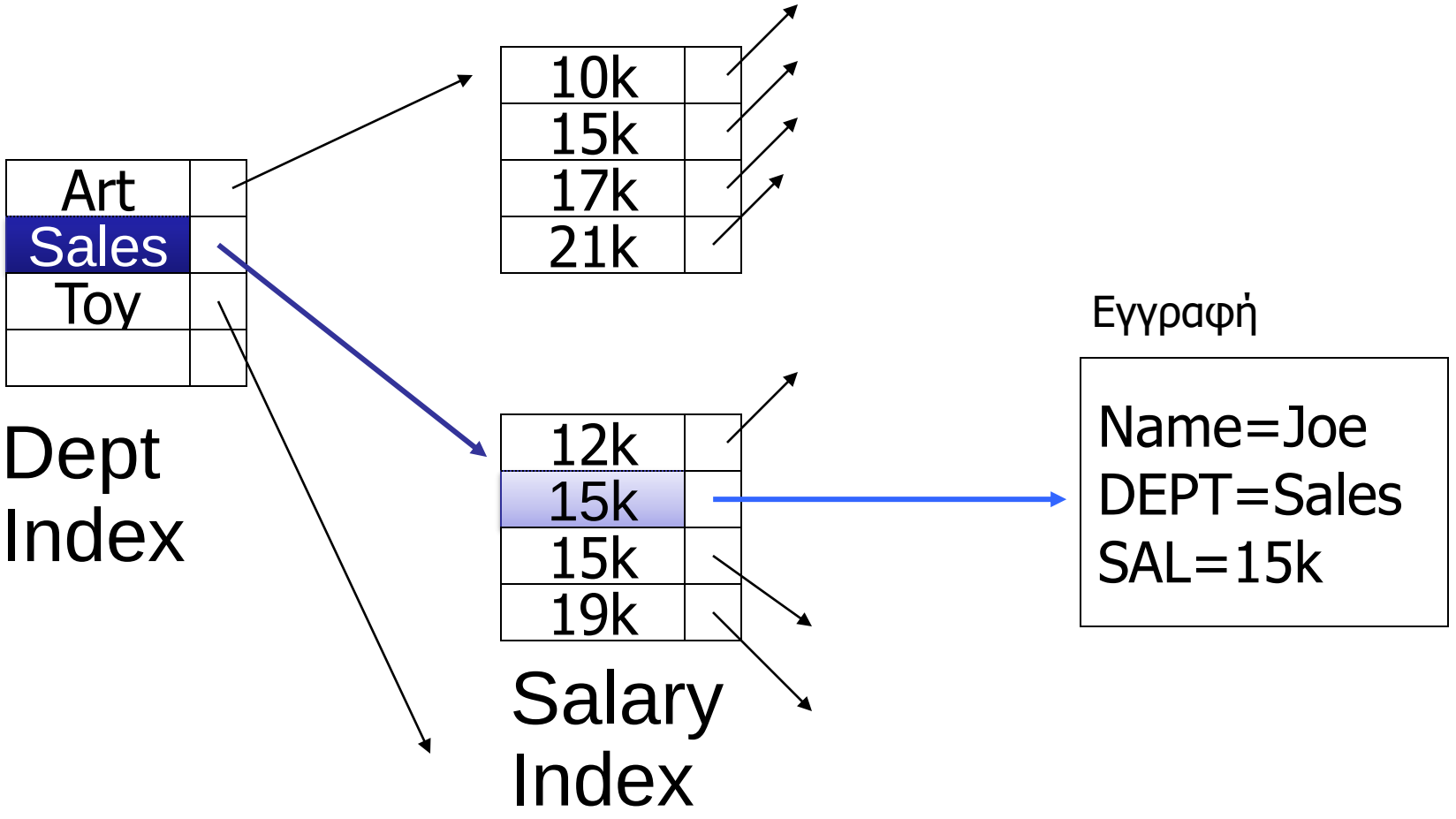
Στρατηγική III-a:

- Ευρετήριο σε πολλαπλά γνωρίσματα



I1: ευρετήριο στο πρώτο γνώρισμα
I2,I3: ευρετήρια στο δεύτερο γνώρισμα

Παράδειγμα


Art
Sales
Toy
Dept
Index
10k
15k
17k
21k
12k
15k
15k
19k
Salary
Index
Εγγραφή
Name=Joe
DEPT=Sales
SAL=15k

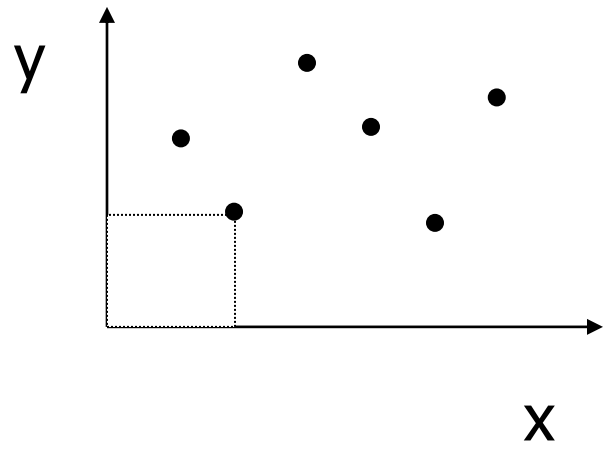
Για ποιες ερωτήσεις είναι το προηγούμενο ευρετήριο αποδοτικό;

- ☐ Dept = “Sales” AND SAL=15k
- ☐ Dept = “Sales” AND SAL $\geq$ 15k
- ☐ Dept = “Sales”
- ☐ SAL = 15k

Στρατηγική III-b

- Ευρετήριο σε σύνθετο κλειδί (Dept,SAL)
  - Πχ (Sales,10K)

Άλλο παράδειγμα: Χωρικές Βάσεις Δεδομένων



Σχέση Restaurants:

<X1,Y1, Attributes>

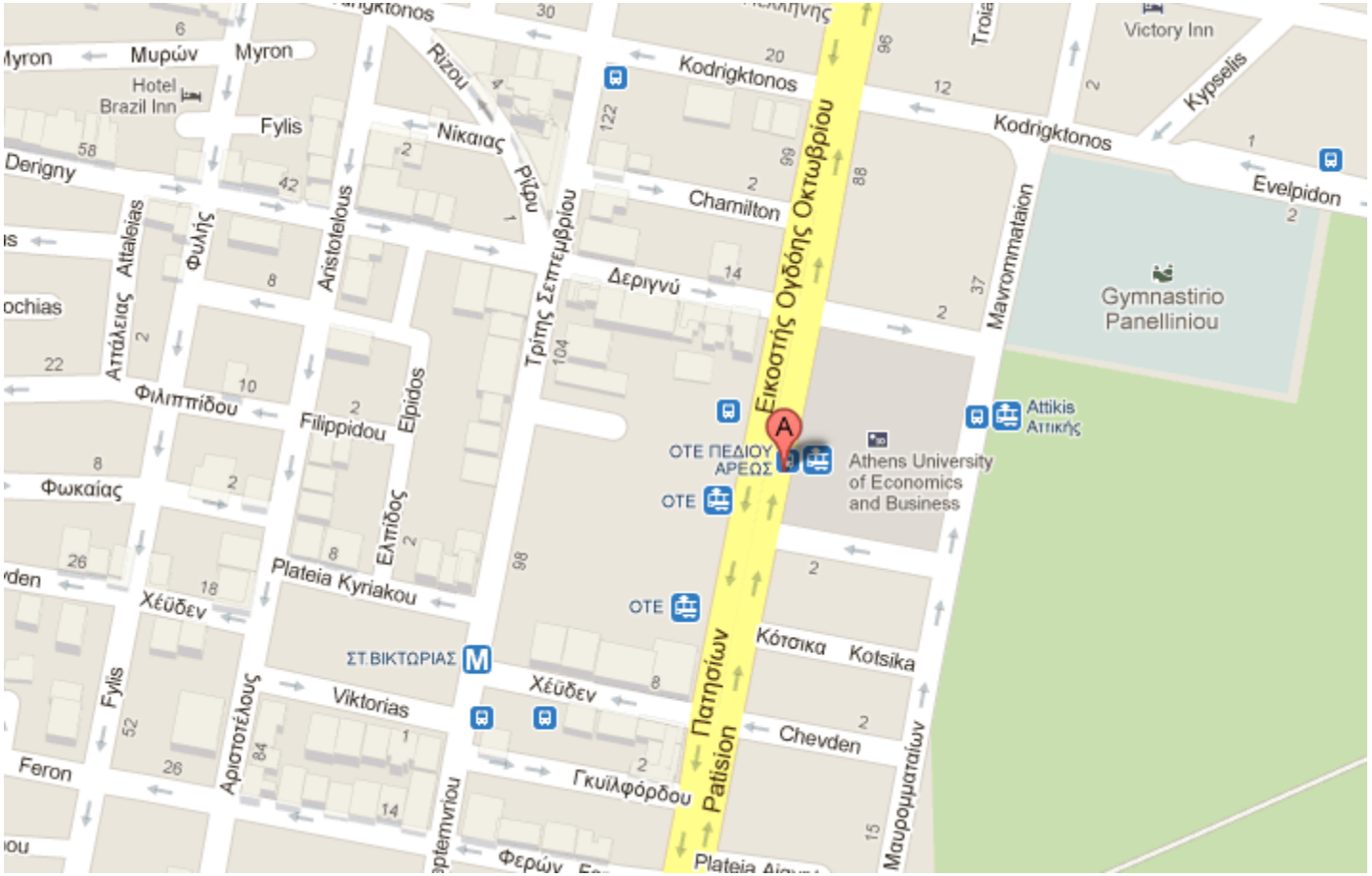
<X2,Y2, Attributes>

⋮

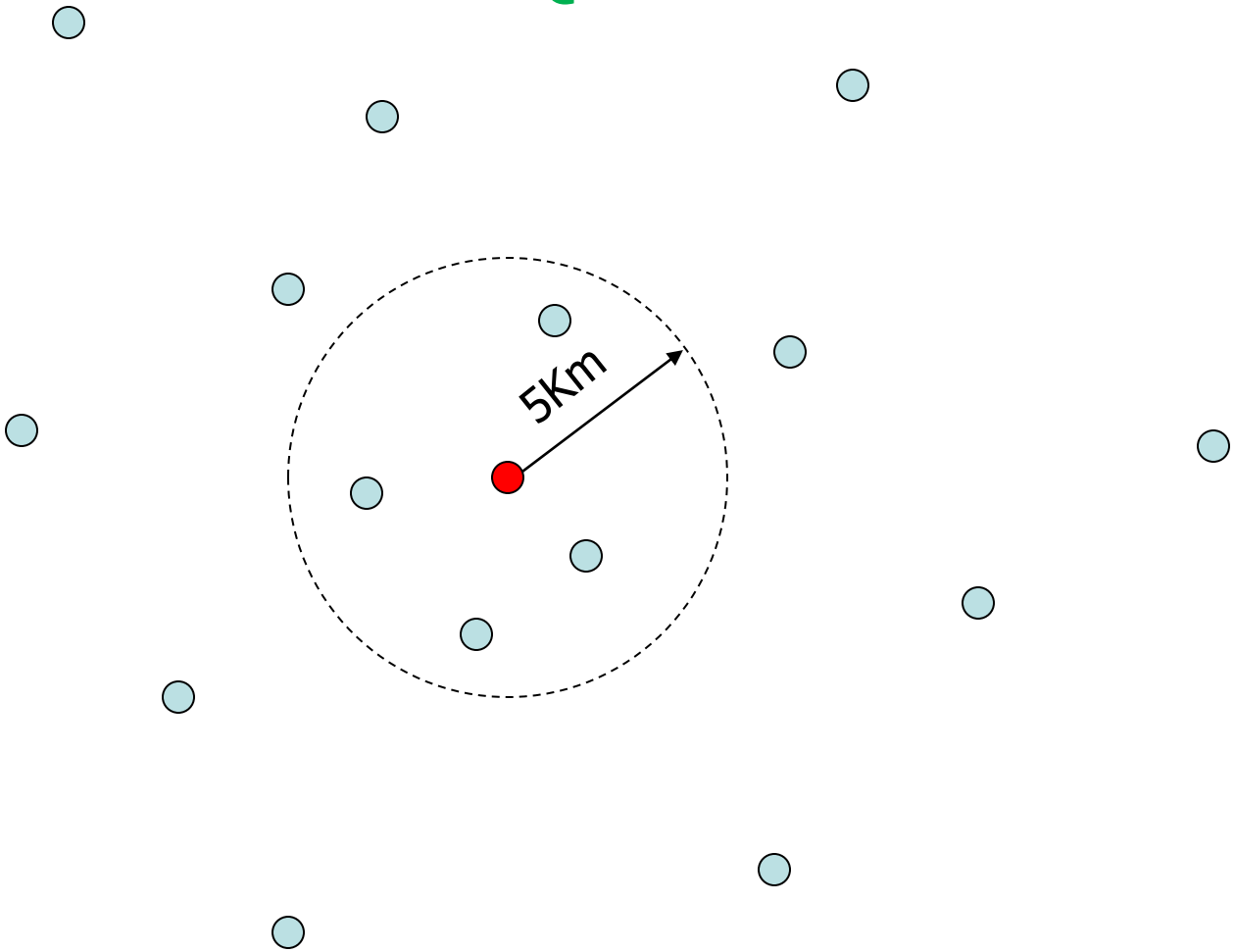
Παραδείγματα χωρικών επερωτήσεων

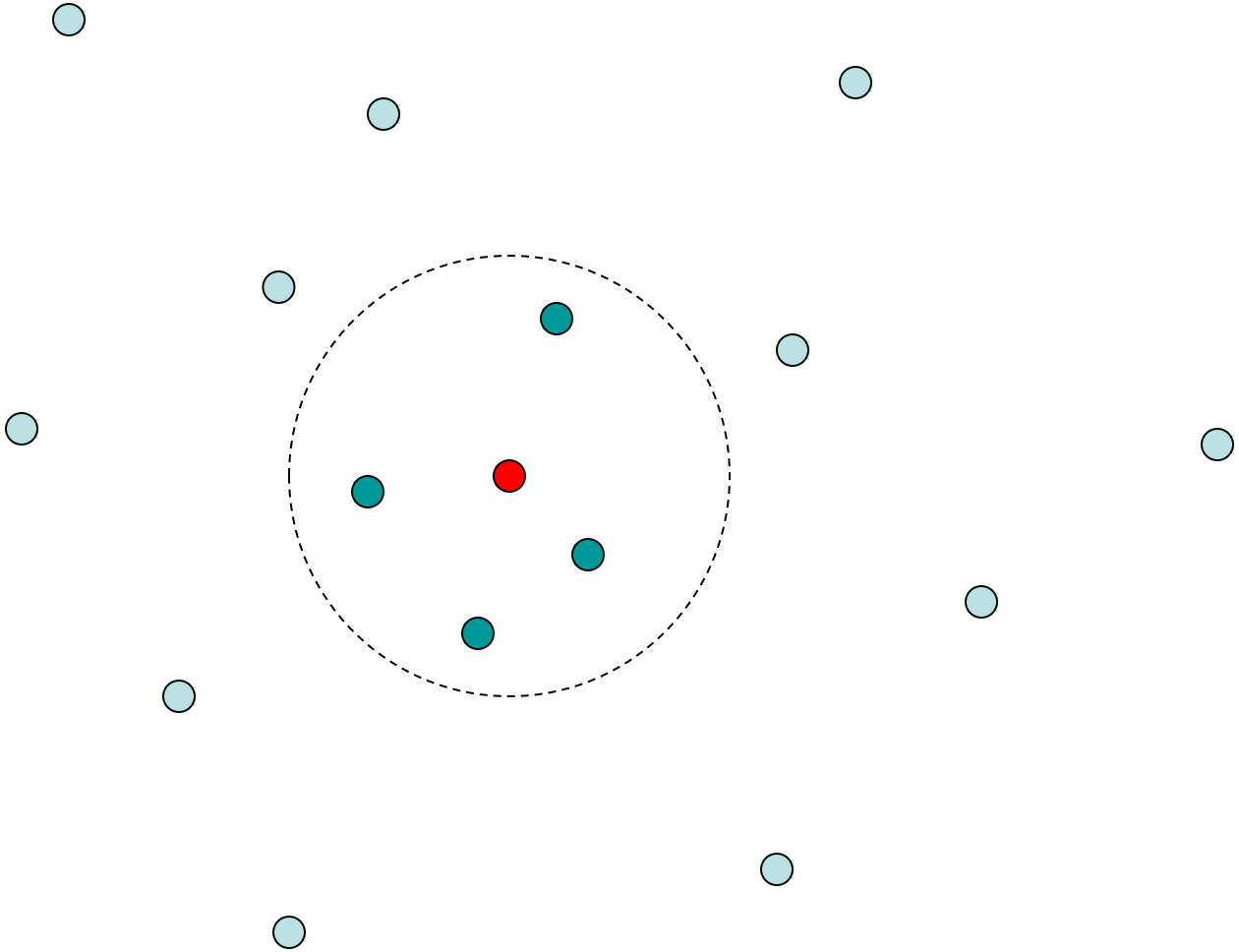
- POINT QUERY (αναζήτηση σημείου)
  - Ποιο είναι το εστιατόριο στη θέση <Xi,Yi>;
- RANGE QUERY(αναζήτηση εύρους)
  - Βρες όλα τα εστιατόρια σε απόσταση μέχρι και 5 χιλιόμετρα από τη θέση μου <Xi,Yi>
- NEAREST NEIGHBOR QUERY (αναζήτηση κοντινότερου γείτονα/ων)
  - Ποιο είναι το κοντινότερο εστιατόριο από την τρέχουσα θέση μου <Xi,Yi>;

Χάρτης

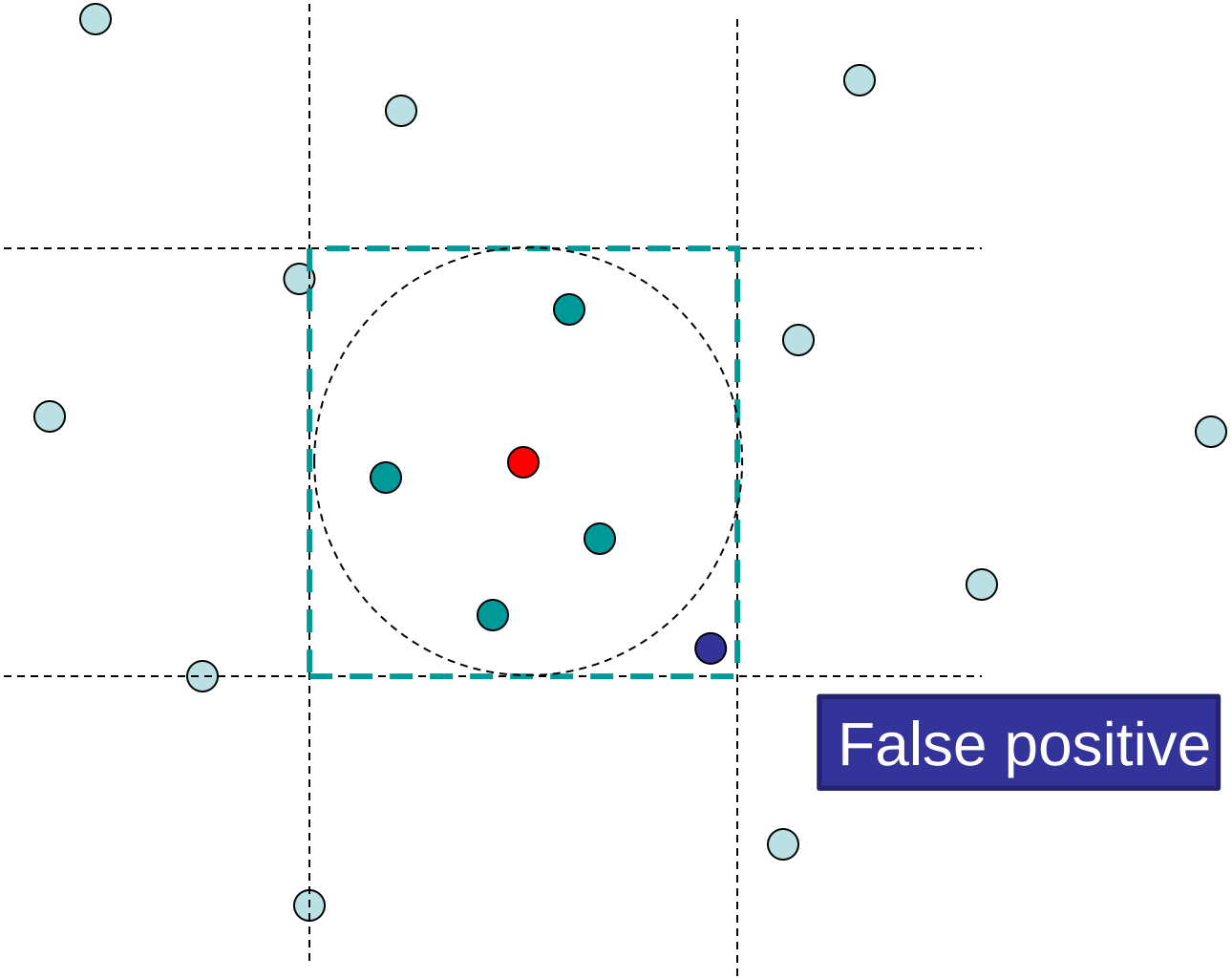

Victory Inn
Kodrigktonos
Kodrigktonos
Kypselis
Evelpidon
Myron
Μυρών
Myron
Hotel
Brazil Inn
Fylis
Rizou
Νικαιας
Derigny
Chamilton
Δεριγνύ
Φυλής
Aristotelous
Τρίτης Σεπτεμβρίου
Εικοστής Ογδόης Οκτωβρίου
Mavrommataion
Gymnastirio
Panelliniou
Attaleias
Αττάλειας
Φιλιππίδου
Filippidou
Elpidos
Ελπίδος
Φωκαίας
Plateia Kyriakou
Χέϋδεν
ΟΤΕ ΠΕΔΙΟΥ ΑΡΕΩΣ
ΟΤΕ
ΟΤΕ
Athens University
of Economics
and Business
Attikis
Αττικής
ΣΤ.ΒΙΚΤΩΡΙΑΣ
Viktorias
Χέϋδεν
Κότσικα Kotsika
Chevden
Γκυϊλφόρδου
Patision
Πατησίων
Feron
Aristotelous
Φερών
Μαυρομματαίων
Plateia Aigyptou

RANGE QUERY



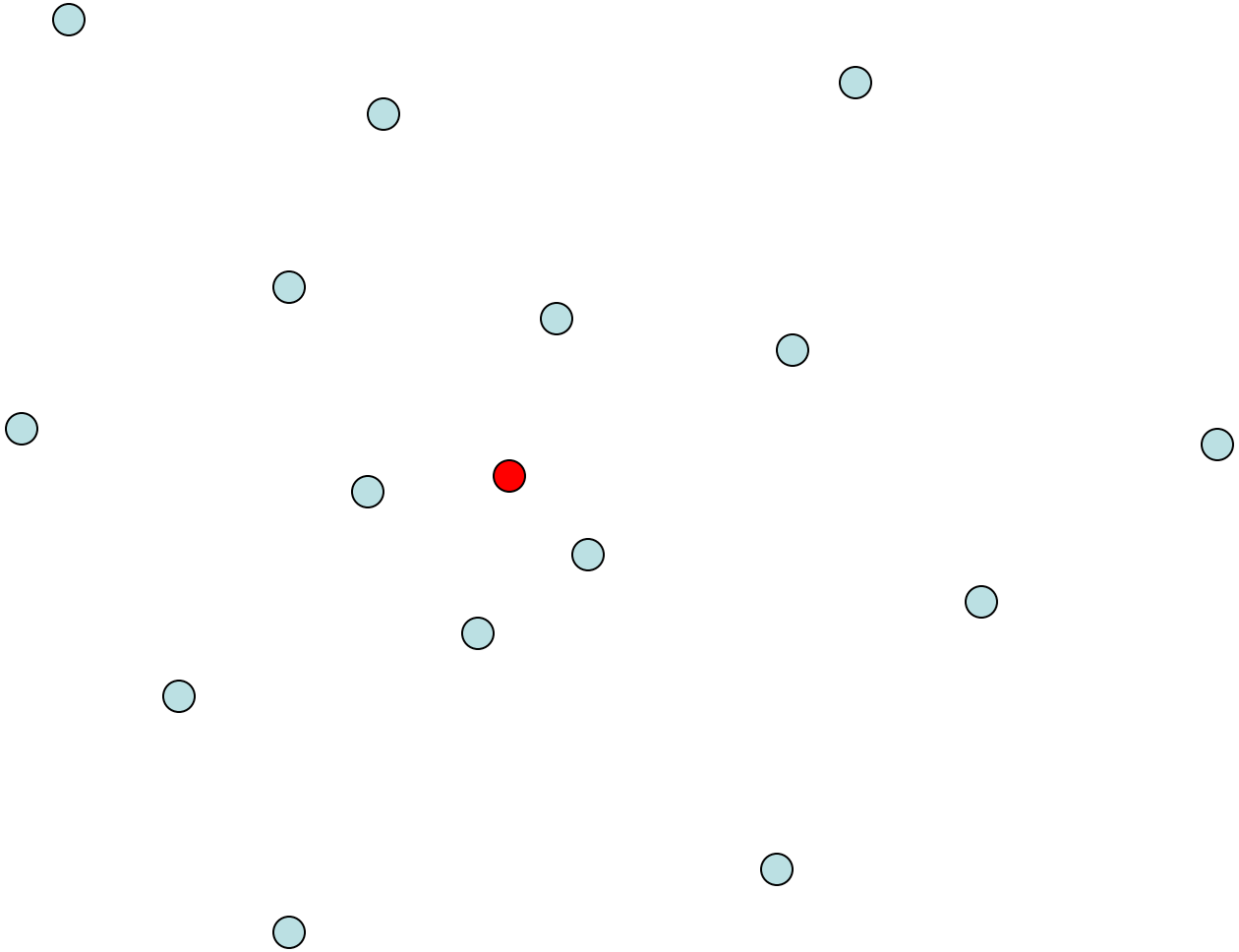


Transformed Range Query

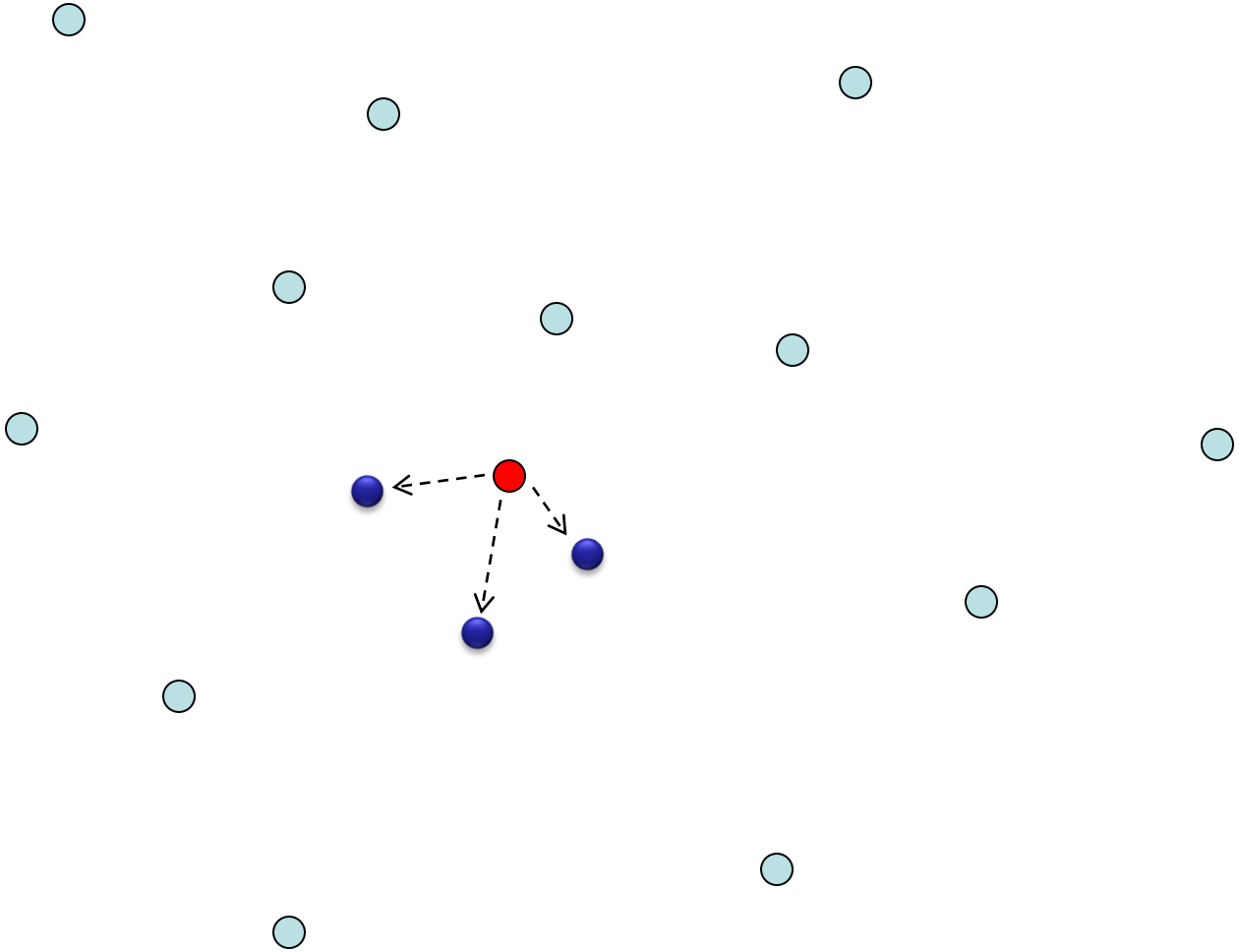


NN-Queries

- Βρες τα 3 κοντινότερα εστιατόρια σε μια τοποθεσία q
  - Αναφέρεται ως $NN_κ(q)$ (κ=3) αναζήτηση
- Το ερώτημα δέχεται 2 παραμέτρους
  - Τη θέση q
  - Τον αριθμό των κοντινότερων γειτόνων που αναζητώ κ

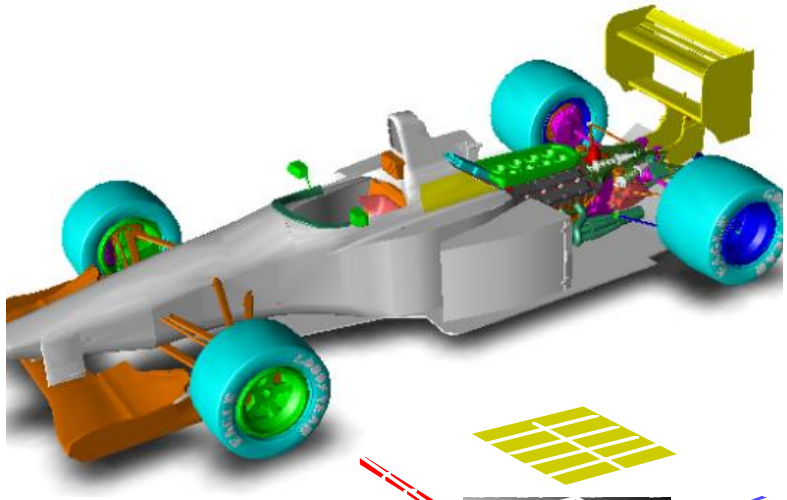


Nearest Neighbor Query Results (k=3)



Άλλες εφαρμογές με πολυδιάστατα δεδομένα

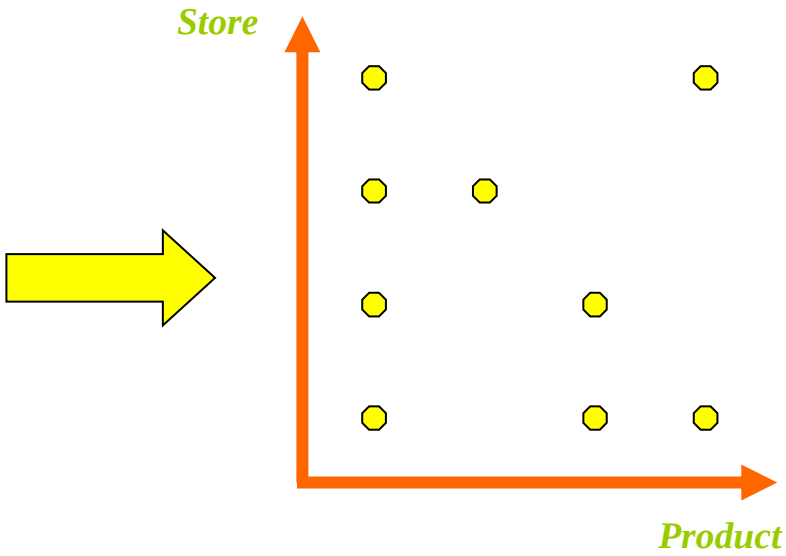
- Mechanical CAD
- VLSI
- Bio Medical Imaging
- OLAP
- …



On-Line Analytical Processing (OLAP)

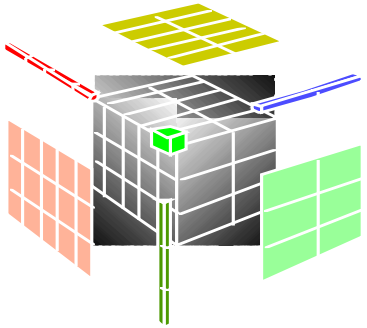
- Ανάλυση πωλήσεων (πχ συνολικές πωλήσεις ανά μαγαζί και προϊόν)

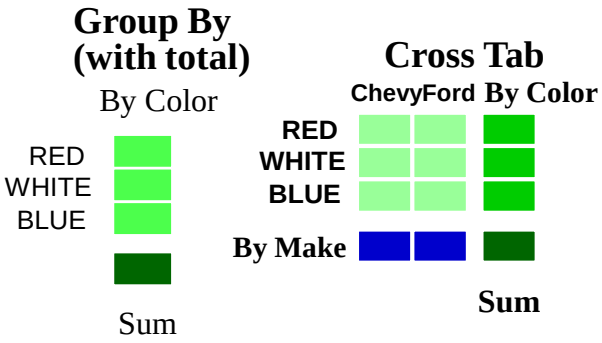
Total Sales		Product			
		1	2	3	4
STORE	1	$454	-	-	$925
	2	$468	$800	-	-
	3	$296	-	$240	-
	4	$652	-	$540	$745



Αποθήκες Δεδομένων

- Multi-dimensional space
  - Συνήθως 20-30 διαστάσεις
  - 2-6 επίπεδα ιεραρχίας σε κάθε διάσταση
  - Πχ Τοποθεσία: Μαγαζί→πόλη→νομός→χώρα





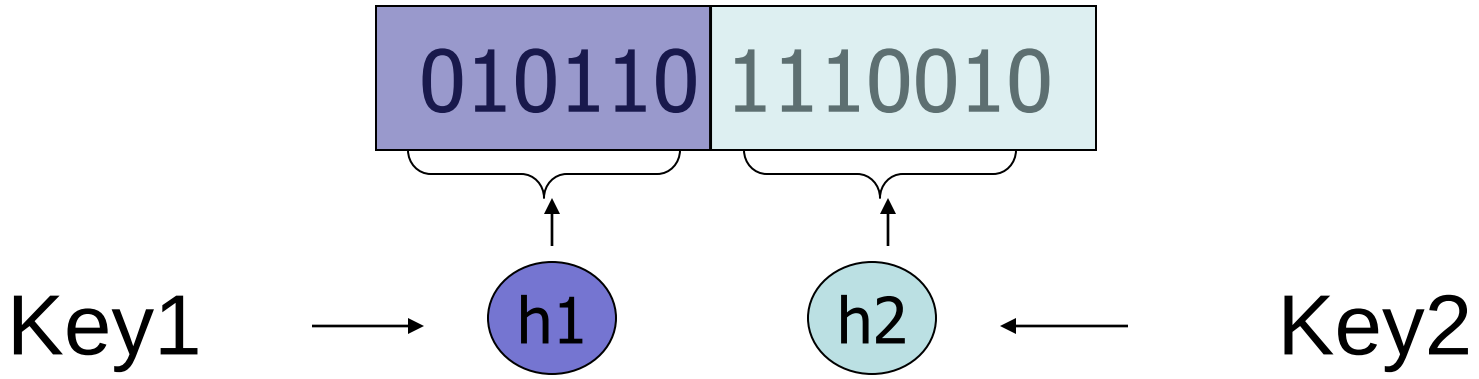
Πολυδιάστατα Ευρετήρια

- Διαμερισμένος Κατακερματισμός
- Space filling curves
- K-D-Tree
- Quad-Tree
- R-tree

Χρήση κατακερματισμού;

- Χρειάζομαι ευρετήριο στα (Dept,SAL)
- Μπορώ να χρησιμοποιήσω μία συνάρτηση h(Dept,SAL) με τιμές στο διάστημα [0,b-1];
  - OK για Dept="TOY" AND SAL=15K
  - Τι γίνεται αν ψάχνω για Dept="SALES"?

Διαμερισμένος κατακερματισμός



Partitioned hash function
h(key1,key2)=“h1(key1)h2(key2)”

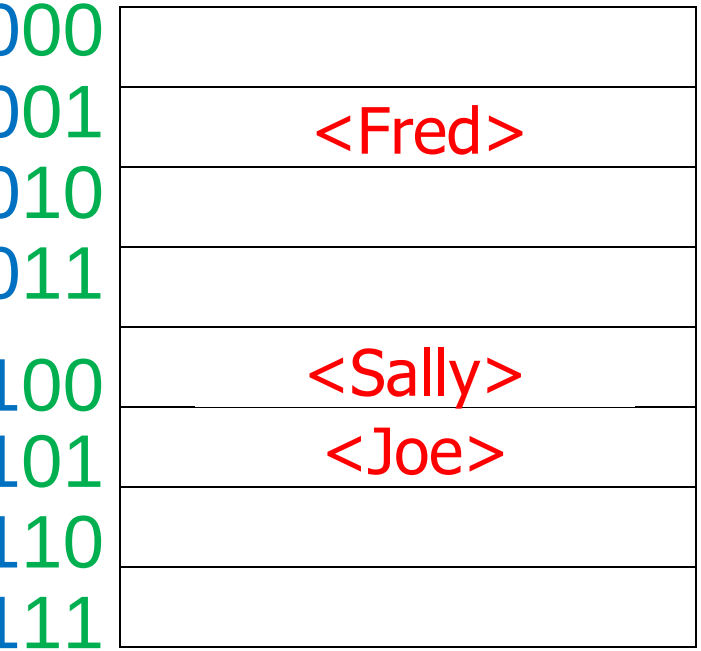
Παράδειγμα (Dept,SAL)

h1(toy) =0
h1(sales) =1
h1(art) =1

.
.

h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00

.
.

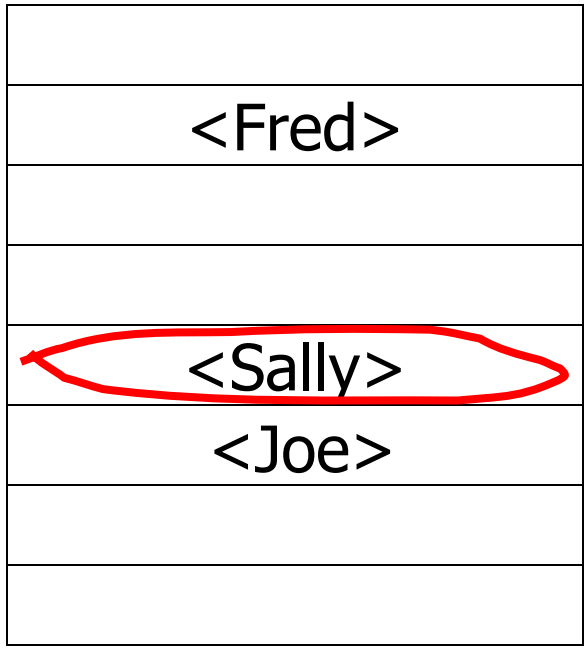


Insert

<Fred,toy,10k>,<Joe,sales,10k>
<Sally,art,40k>

Βρες υπαλλήλους όπου Dept=Sales AND SAL=40k

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.

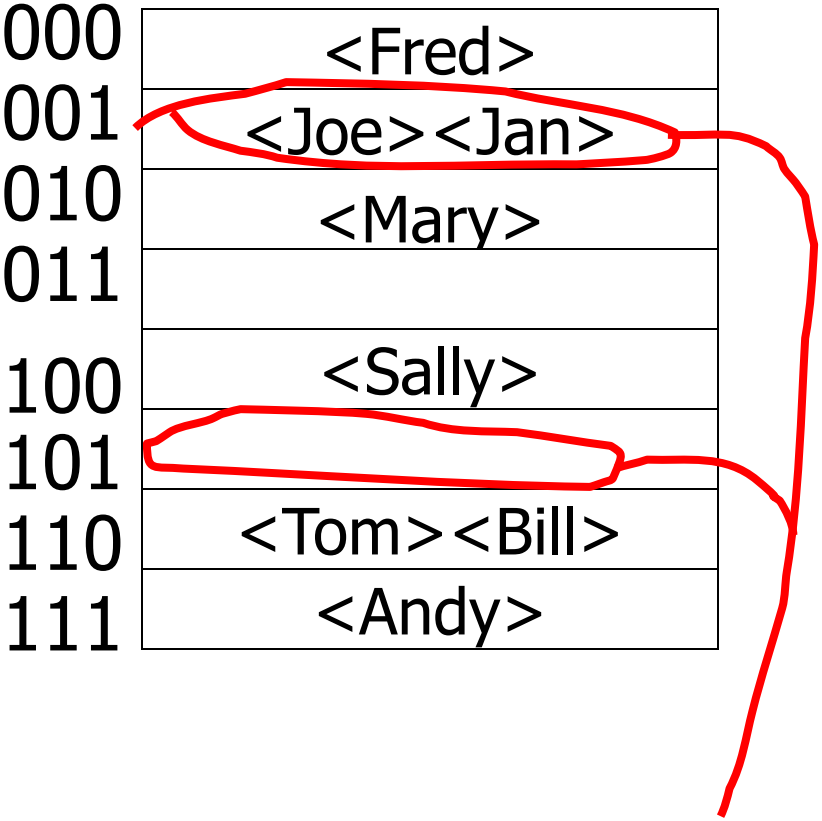


Record: <Sally,art,40k>

False Positive: Θα διαβάσει την εγγραφή και θα την απορρίψει

Βρες υπαλλήλους όπου SAL=30k

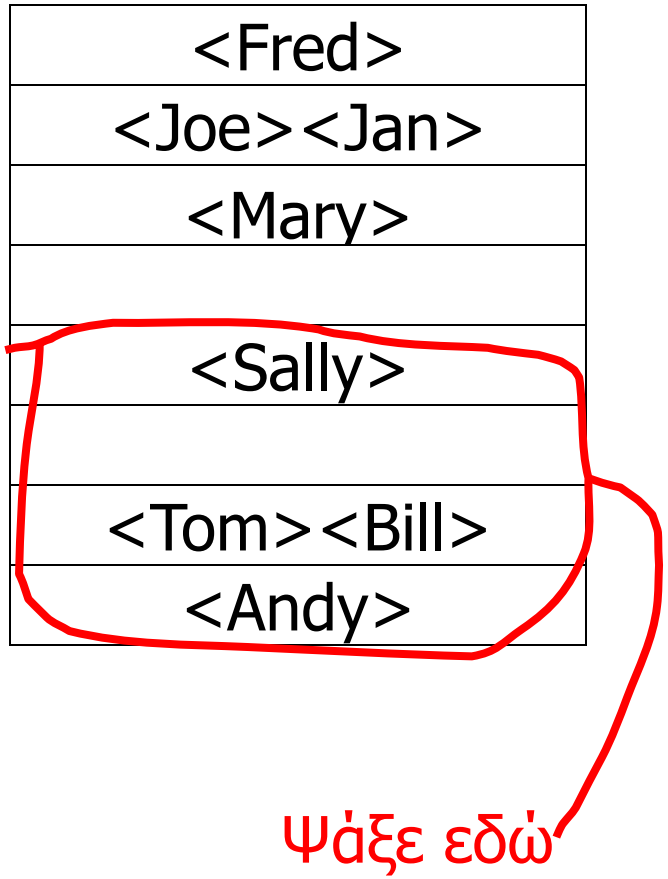
h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.



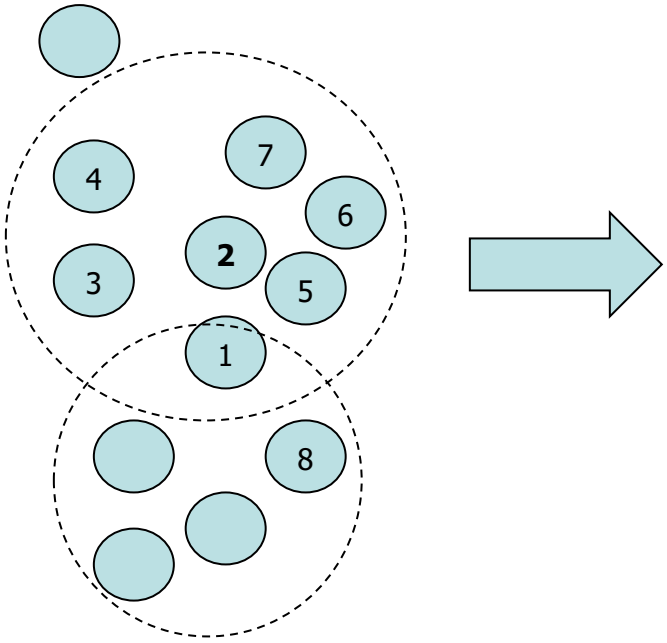
Ψάξε αυτά τα buckets

Βρες υπαλλήλους όπου Dept=Sales

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.



Why Multi-dimensional indexing is hard?



Disk pages

Rec1 **Rec2** Rec3	Rec4 Rec5 Rec6	Rec7 Rec8

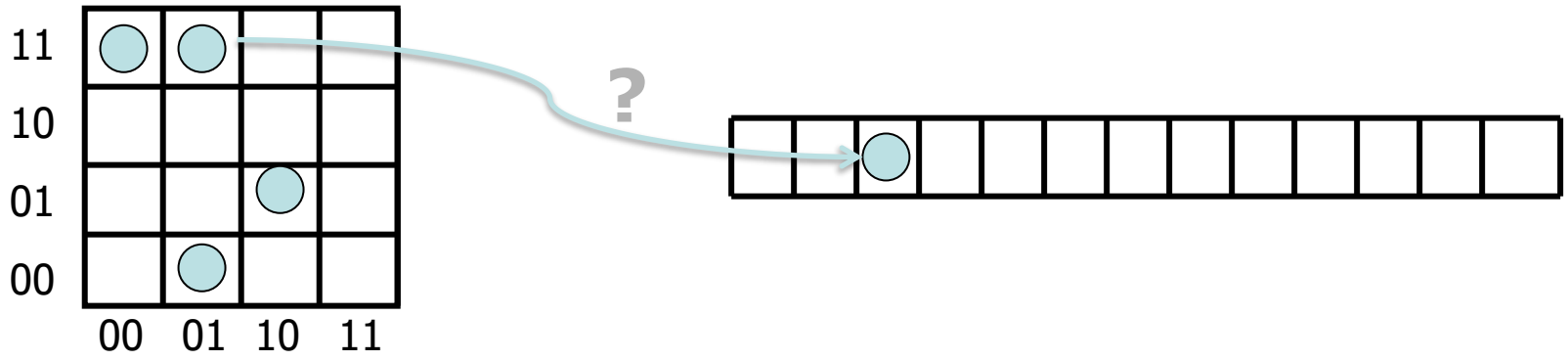
Ideally, we want nearby points to be stored in the same or in adjacent pages on disk

It becomes harder to maintain locality of multidimensional data, when dimensionality increases

Space filling curves

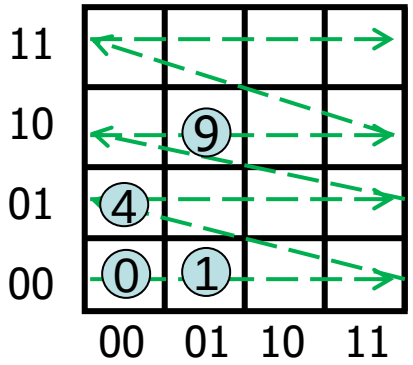
- Objective: map n-dim points into 1-dim space
- Assumption: space has finite granularity
  - e.g. 2^{32} x 2^{32} grid

- Example with 4x4 grid :



Simple Idea

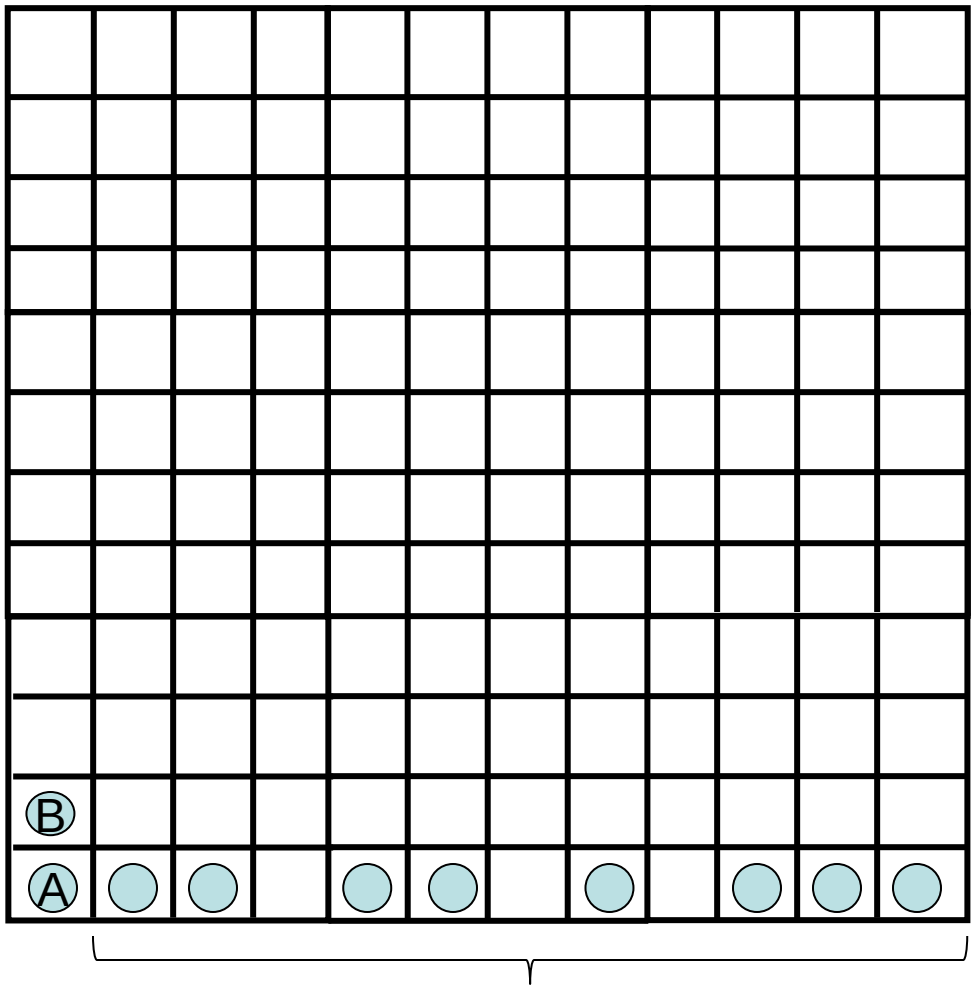
- Map data row-wise



Is it good?

Consider a larder grid

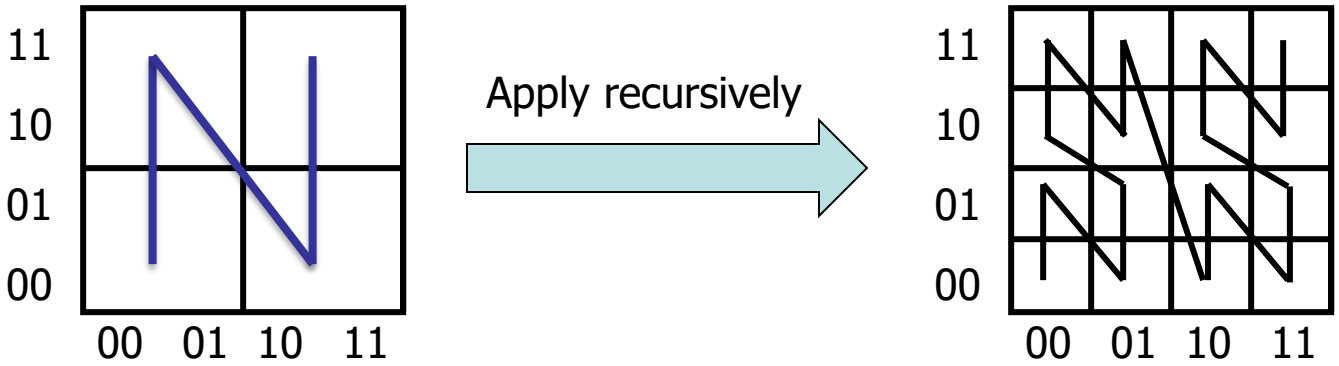
Preserving locality?



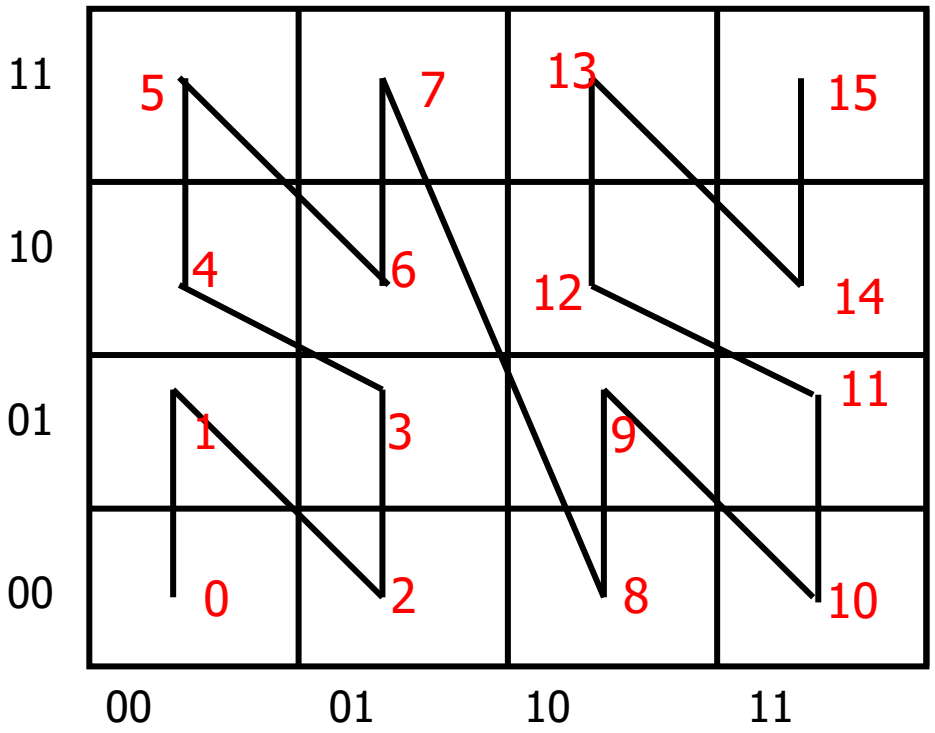
All these points will be mapped/stored in between A and B

Z-ordering

- A.k.a. bit-shuffling/linear-quadtrees
  - few long jumps
  - scoops out a whole quadrant before leaving it
  - thus the name space filling curves

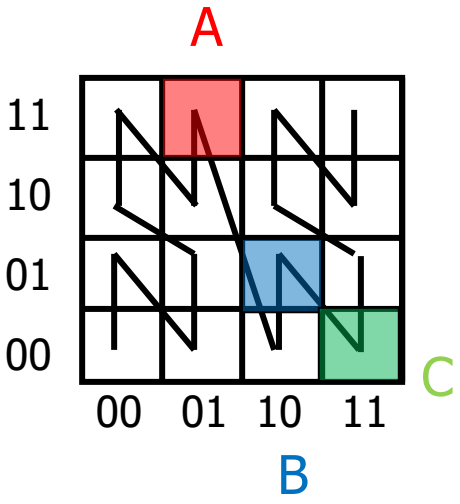


Z-values



Z-ordering

- How to compute z-value ?
  - recursive computation, bit-shuffling, linear quad-trees



$Z_A = shuffle(x_A, y_A) = shuffle(\text{“}01\text{”}, \text{“}11\text{”})$

$= 0111 = (7)_{10}$

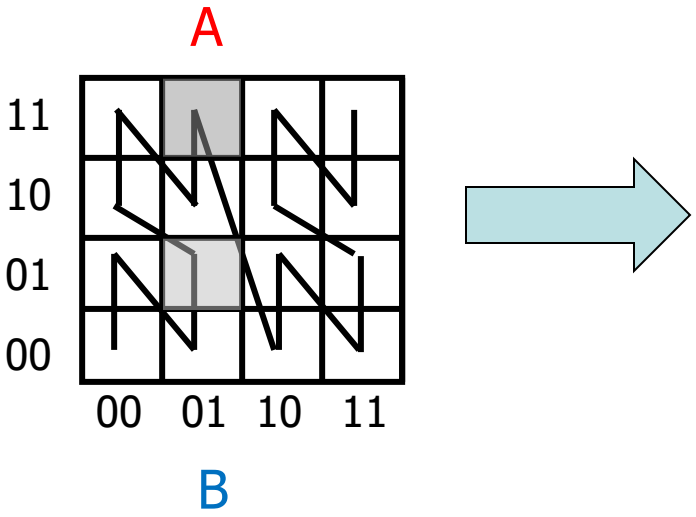
$Z_B = shuffle(\text{“}10\text{”}, \text{“}01\text{”}) = 1001 = (9)_{10}$

$Z_C = shuffle(\text{“}11\text{”}, \text{“}00\text{”}) = 1010 = (10)_{10}$

Similar computation for d>2 dims

Implementation

- Use a B-tree to index the z-values
  - E.g. use z-value as a key in order to index the multidimensional records

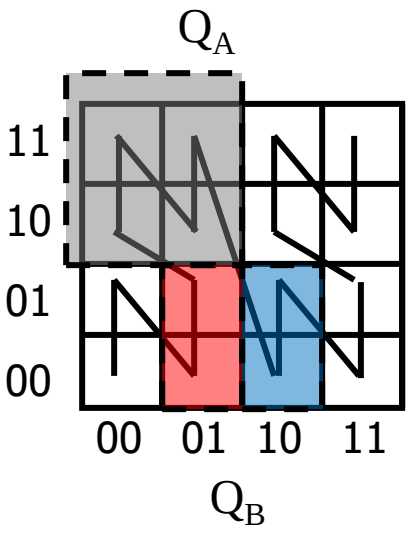


Z-value	Record
3	B
7	A

  - Can be also utilized in (key,value)-stores

Range Queries

- A range (rectangular) query in 2-d is mapped to a set of ranges in 1-d
  - Trick: decompose only non-full quadrants



Q_A → range [4,7]

Q_B → ranges [2,3] and [8,9]

Space filling curves

- We want points that are close in 2-d to be close in the 1-d
- Z-curve has some “jumps” that we would like to avoid
- Hilbert curve avoids the jumps: recursive definition

Hilbert Curve

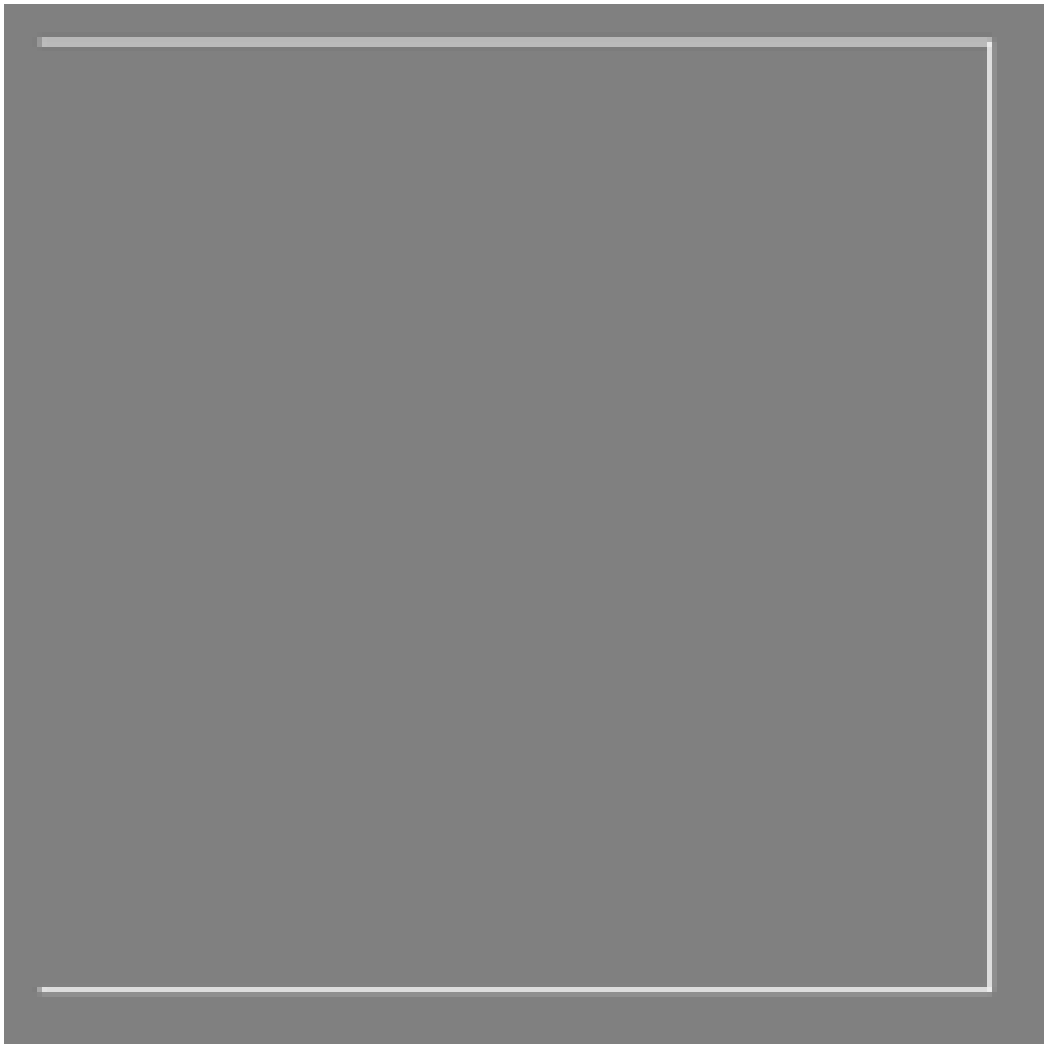
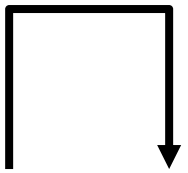


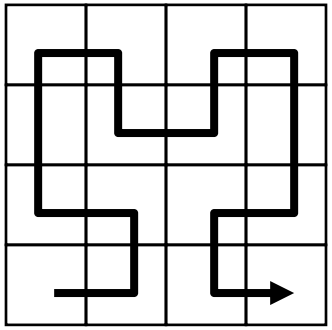
Image from Wikipedia

Hilbert Curve - details

- It has been shown that in general Hilbert is better than the other space filling curves for retrieval [Jag90]
- H_i (order-i) Hilbert curve for $2^i x 2^i$ array

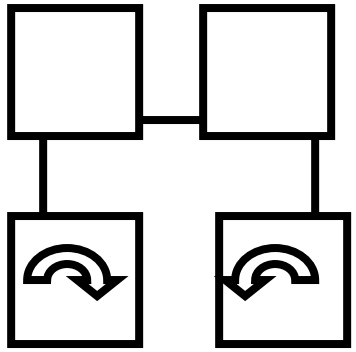


H_1



H_2

...



$H_{(n+1)}$

References

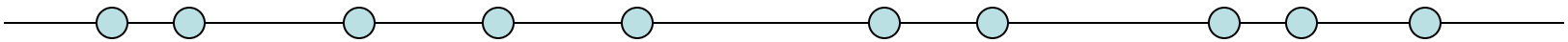
- H. V. Jagadish: Linear Clustering of Objects with Multiple Atributes. ACM SIGMOD Conference 1990: 332-342

- Walid G. Aref, Hanan Samet: A Window Retrieval Algorithm for Spatial Databases Using Quadtrees. ACM-GIS 1995: 69-77

Στη συνέχεια…

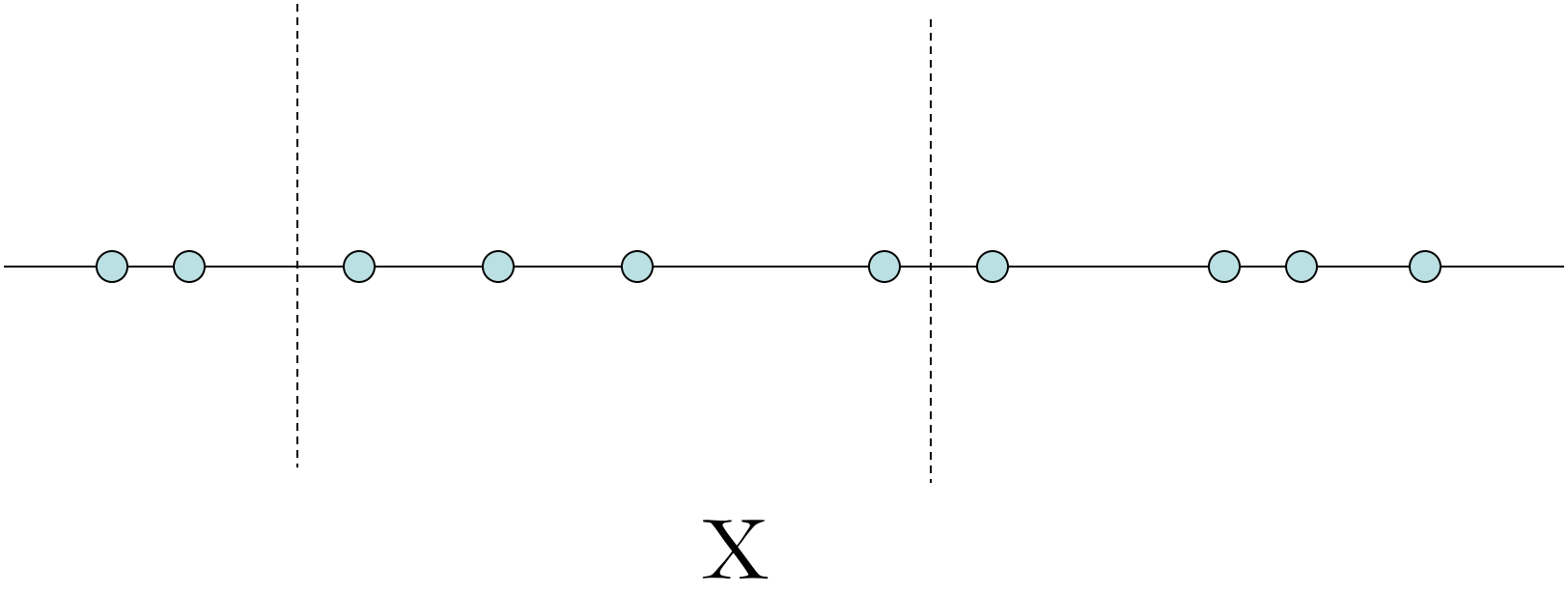
- Hierarchical (tree-based) multi-dimensional indexes
  - K-D Tree
  - Quad Tree
  - R-Tree

1-Dimensional Range Searching

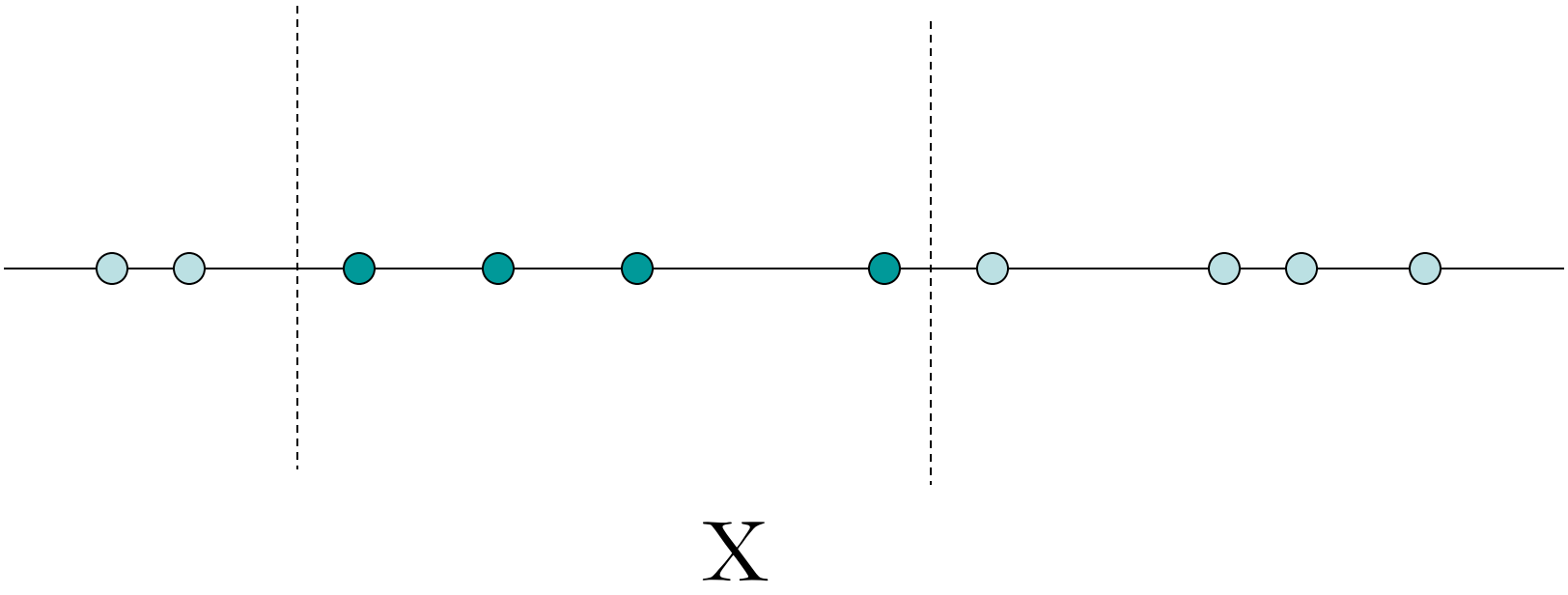


X

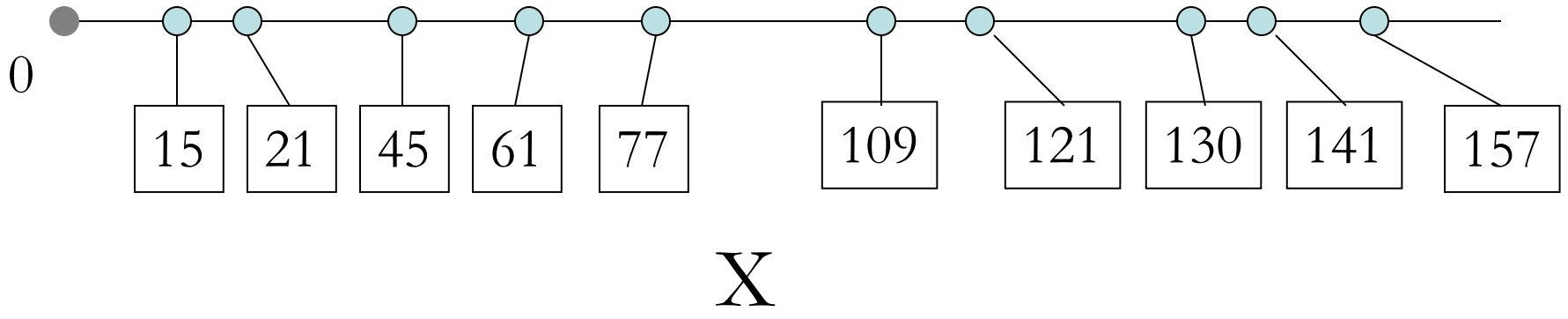
1-Dimensional Range Searching



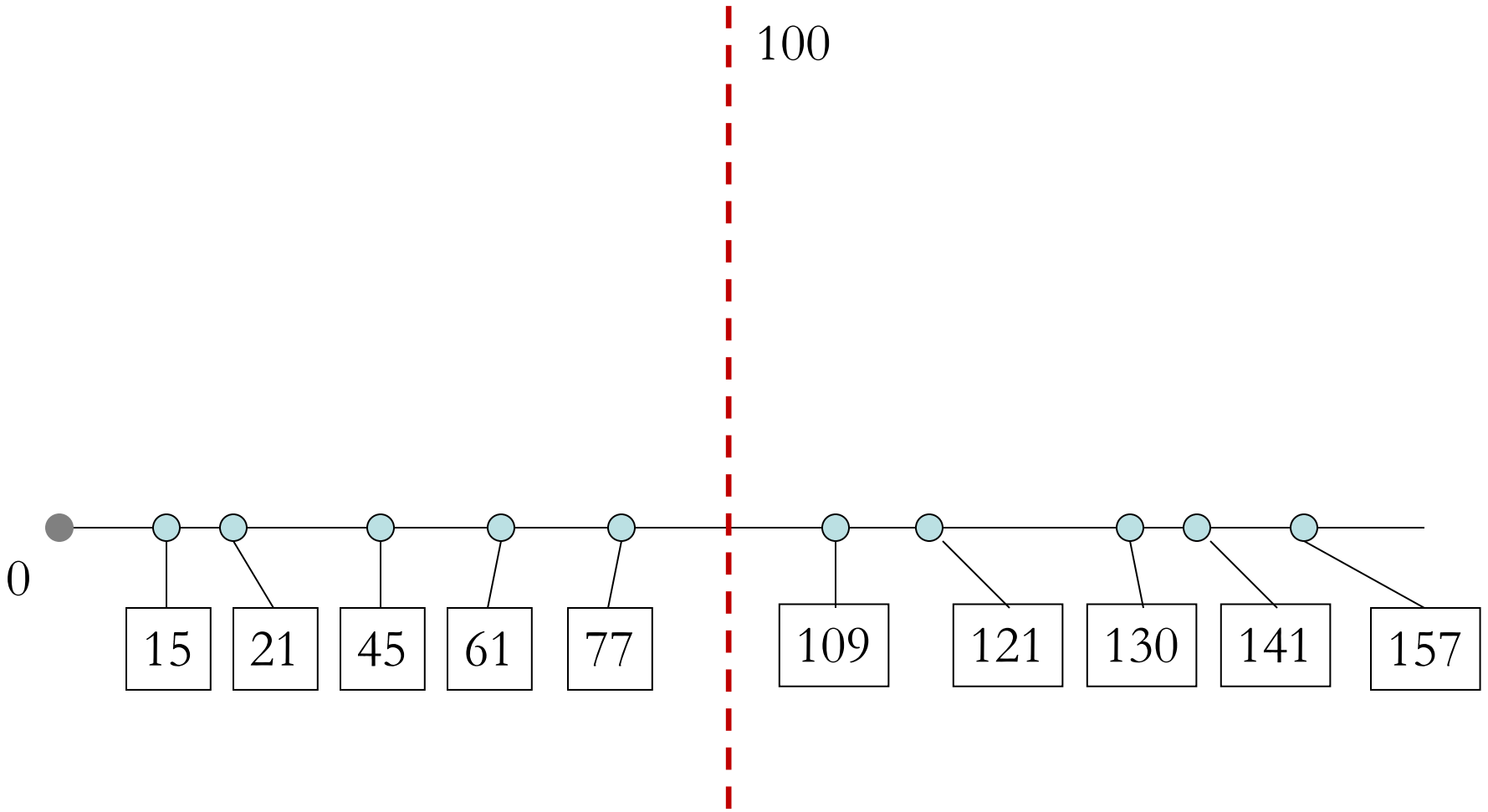
1-Dimensional Range Searching



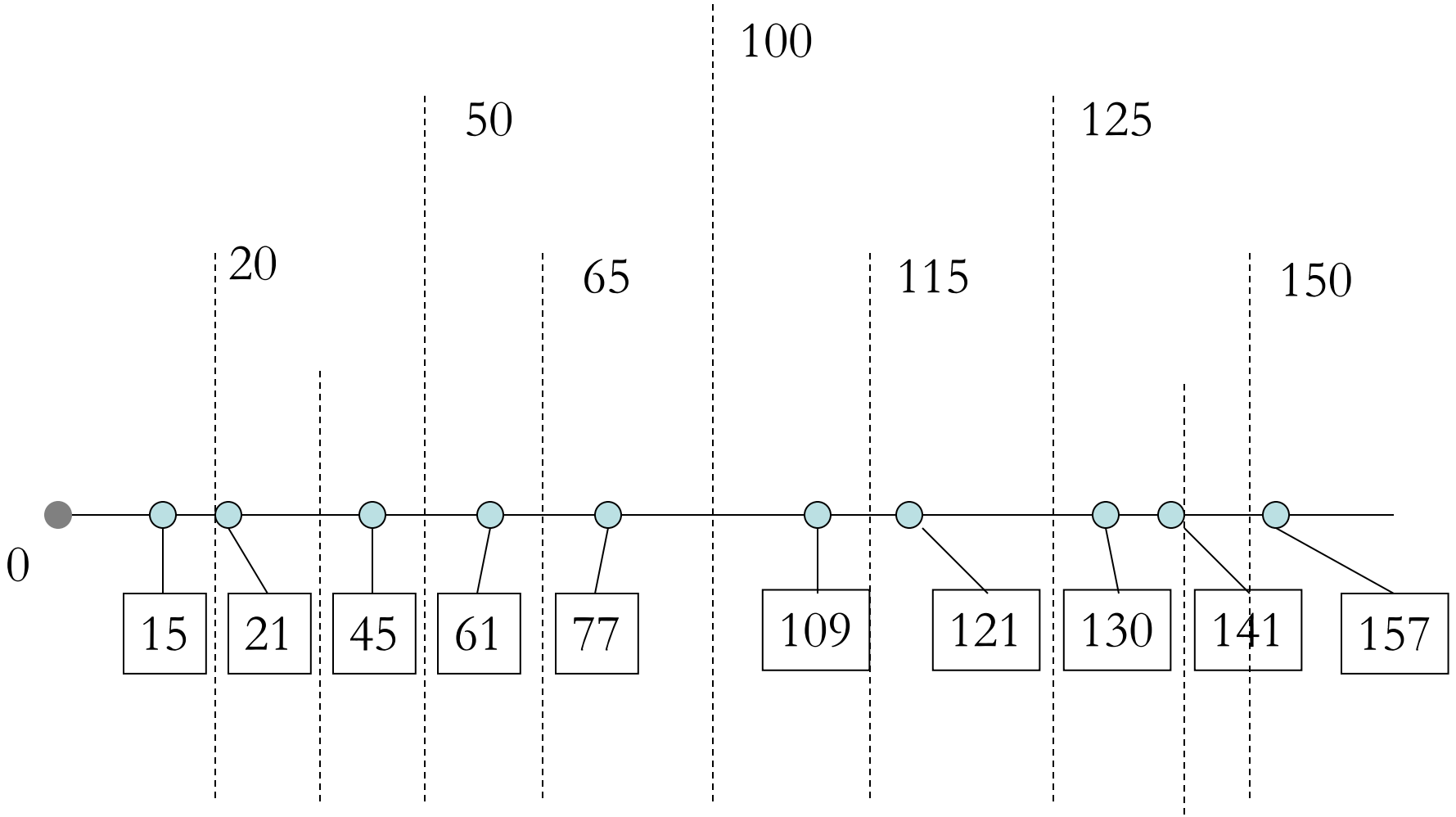
1-Dimensional Range Searching

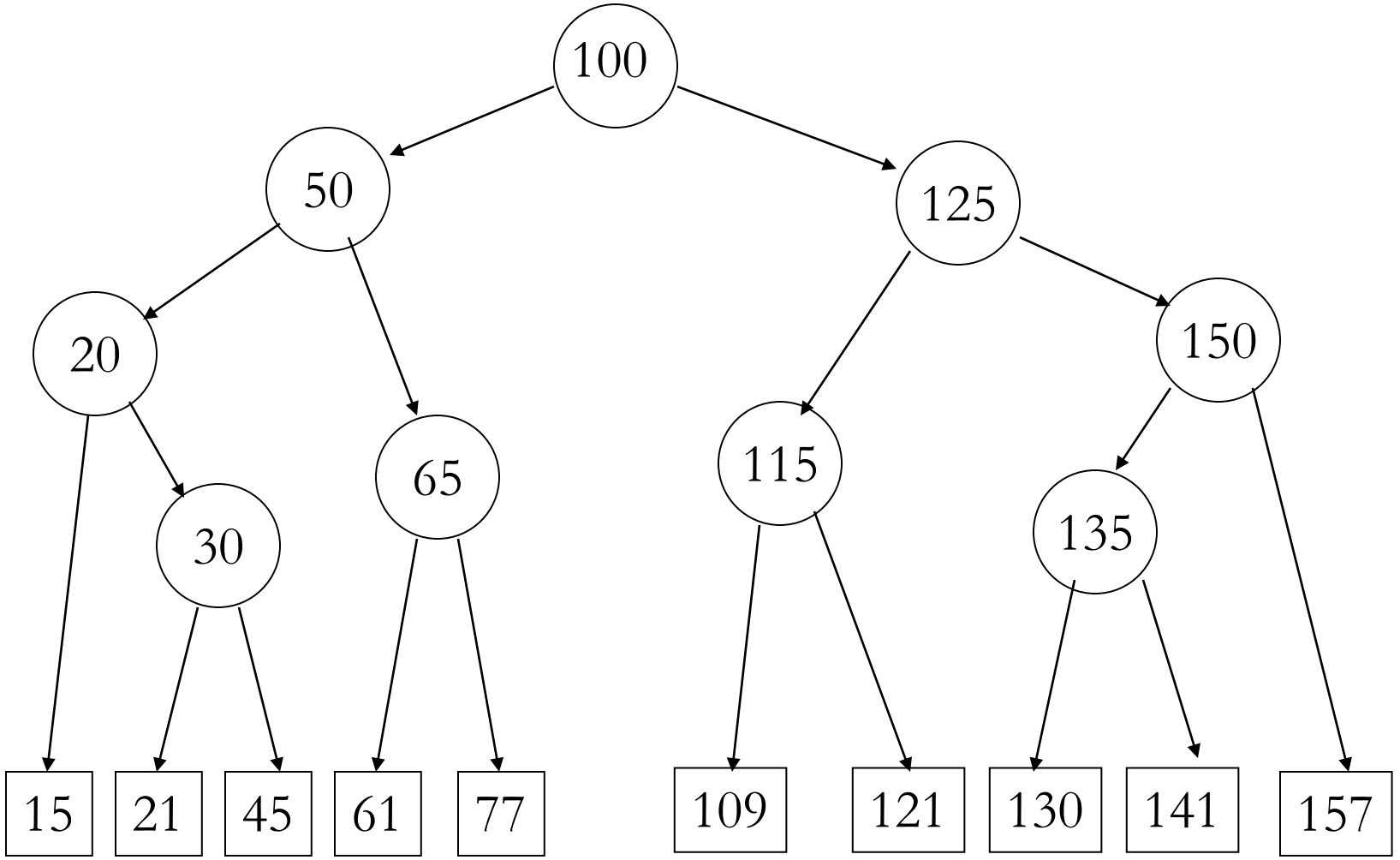


Find “Splitter”

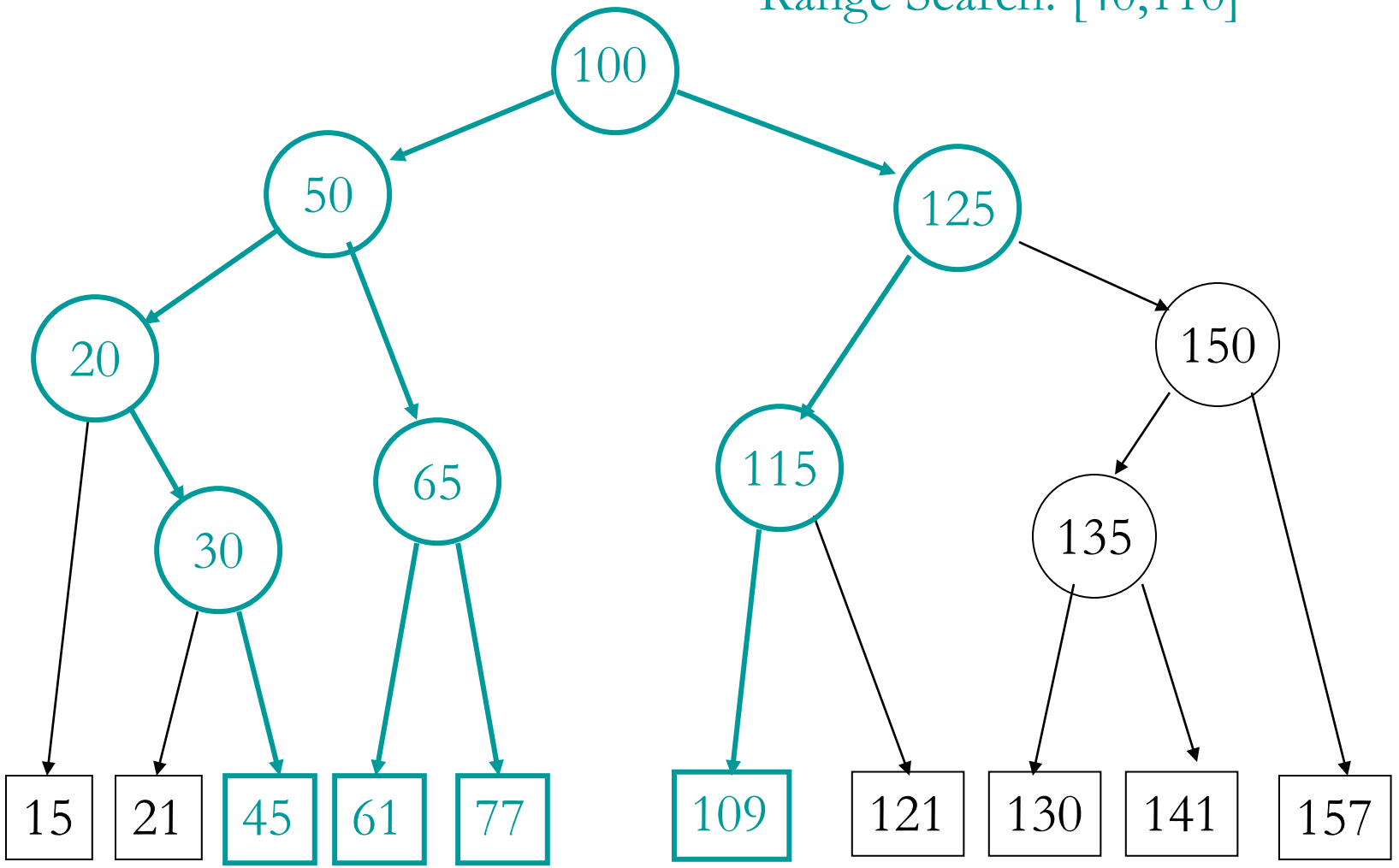


Binary Search Tree




100
50
125
20
150
65
115
30
135
15
21
45
61
77
109
121
130
141
157

Range Search: [40,110]



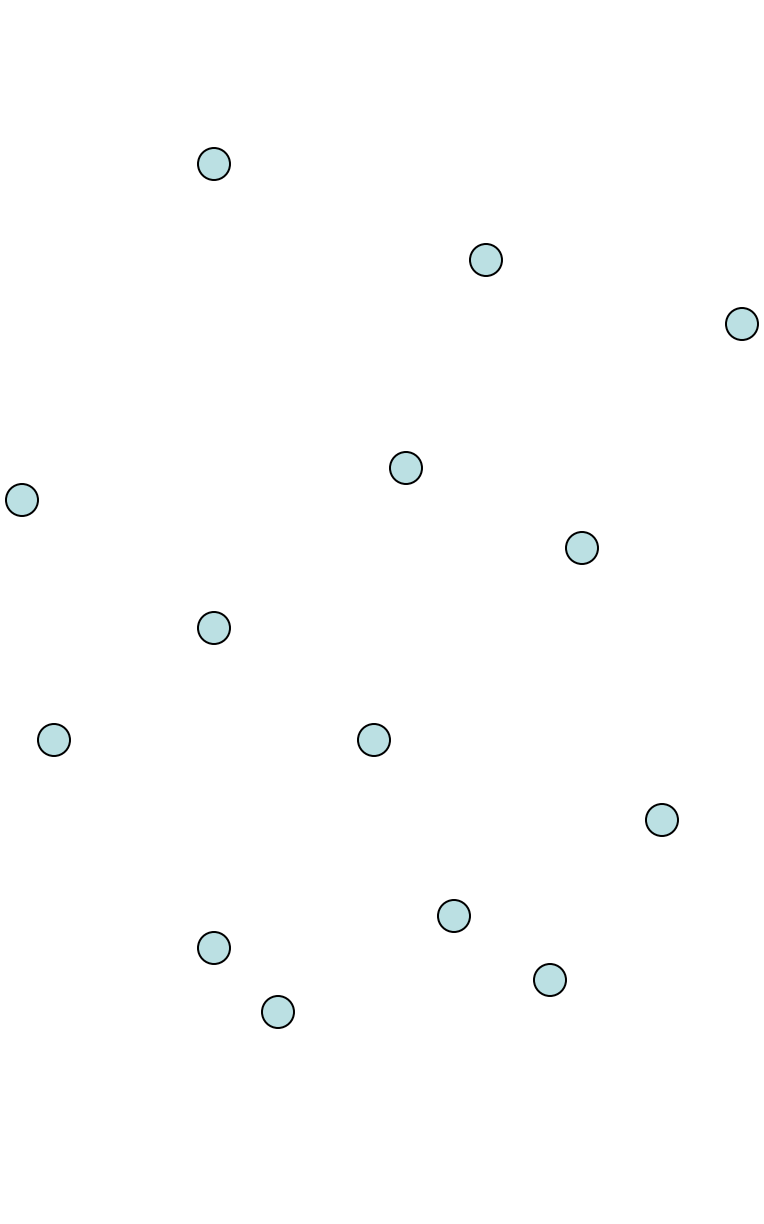
1-Dimensional Range Searching

Efficiency:

$O(\log N) + \text{output size}$

$N = \#$ indexed points

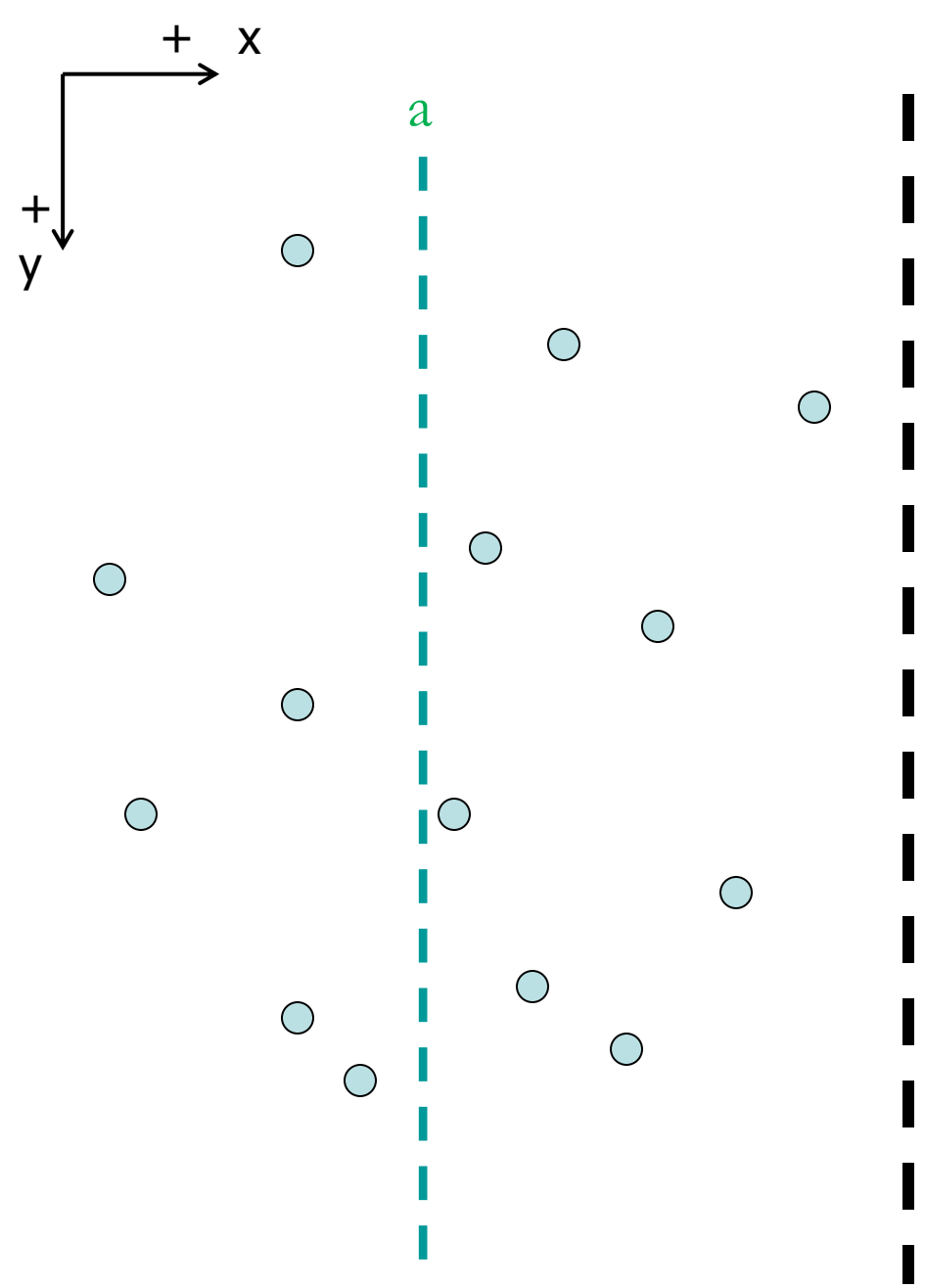
K-D Tree

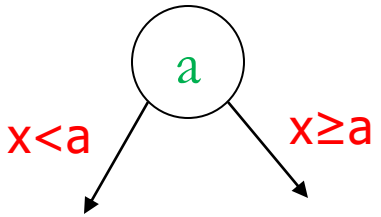


Data

K-D Tree

Split on X

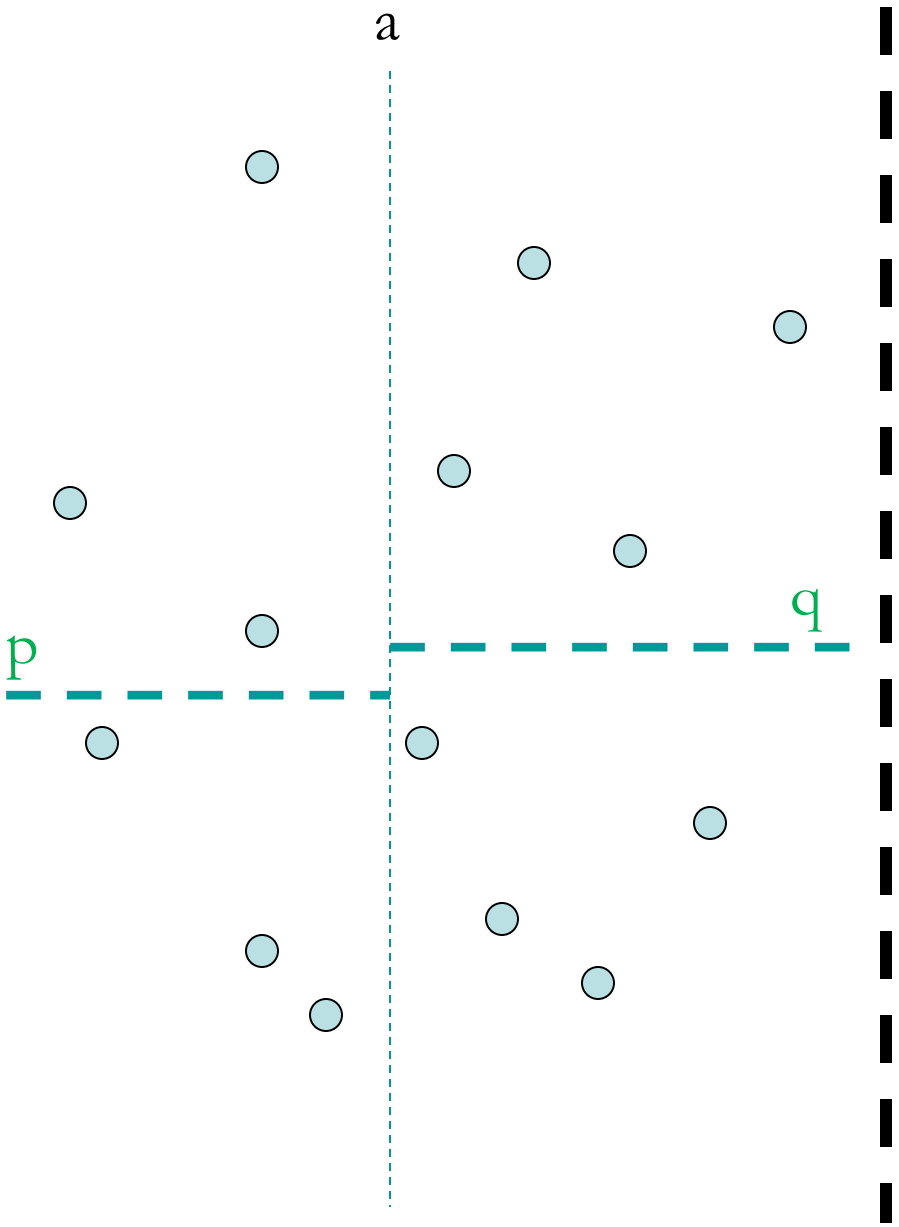




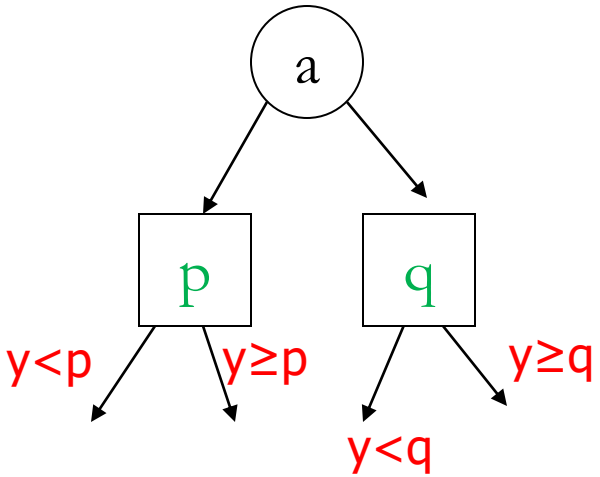
Data

K-D Tree

Split on Y

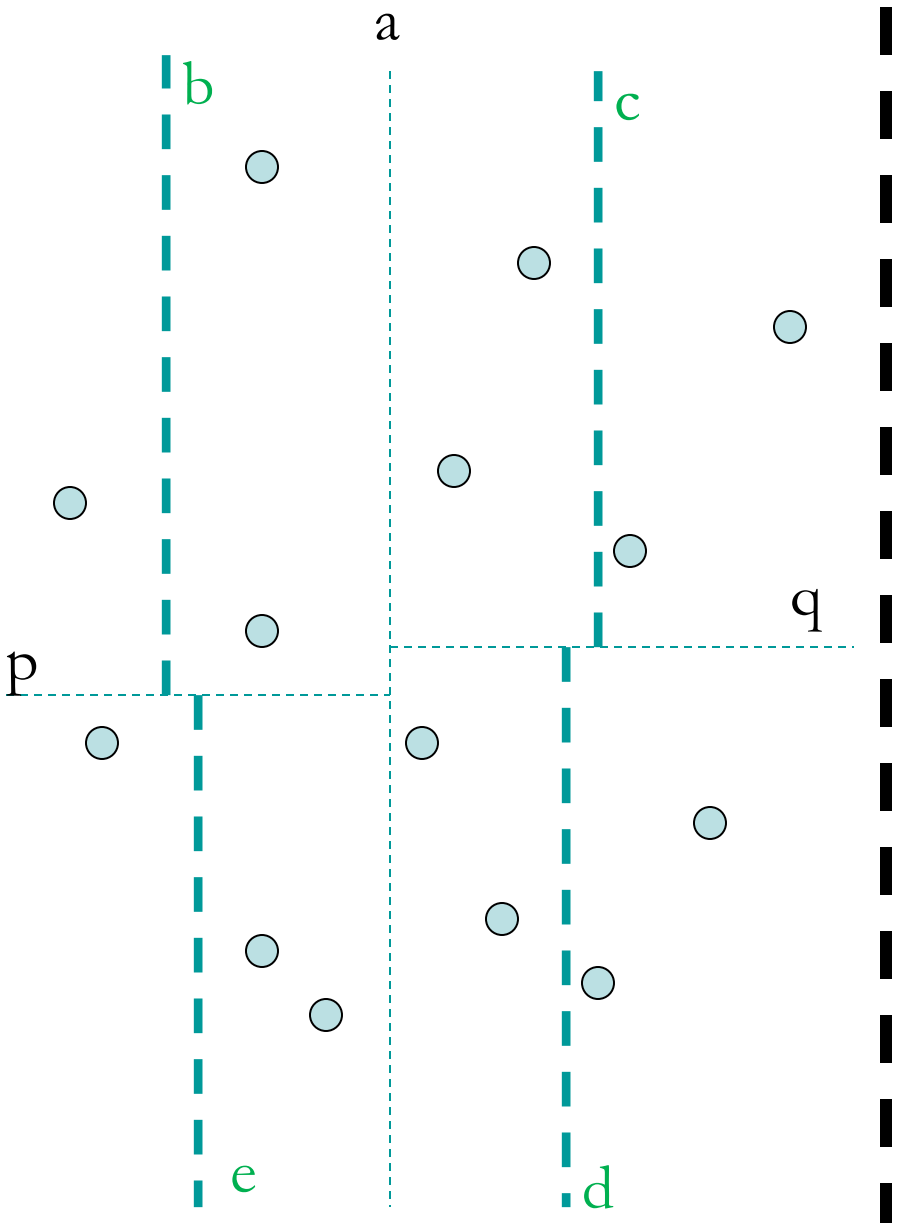


Data

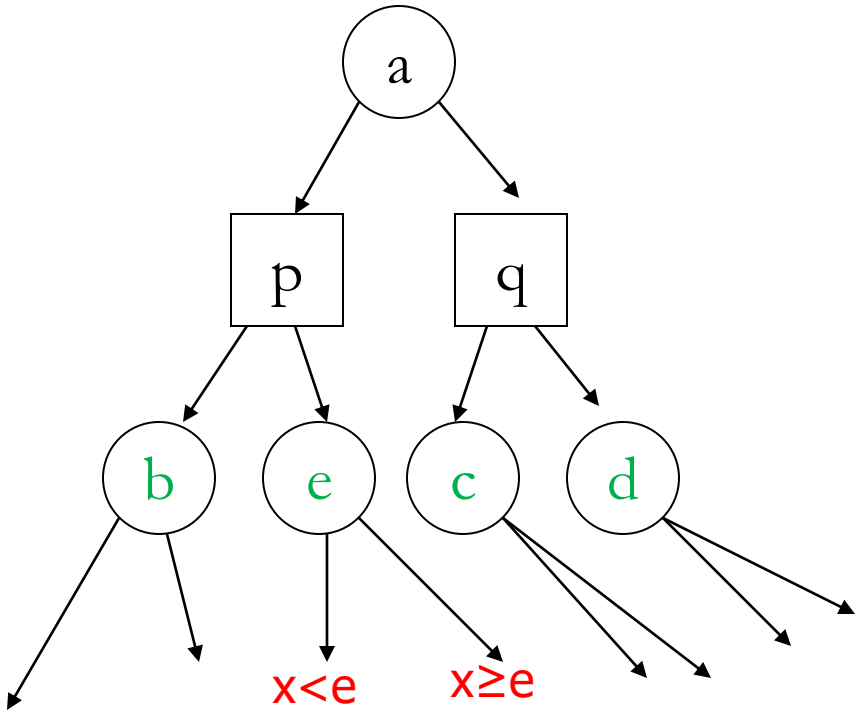


K-D Tree

Split on X

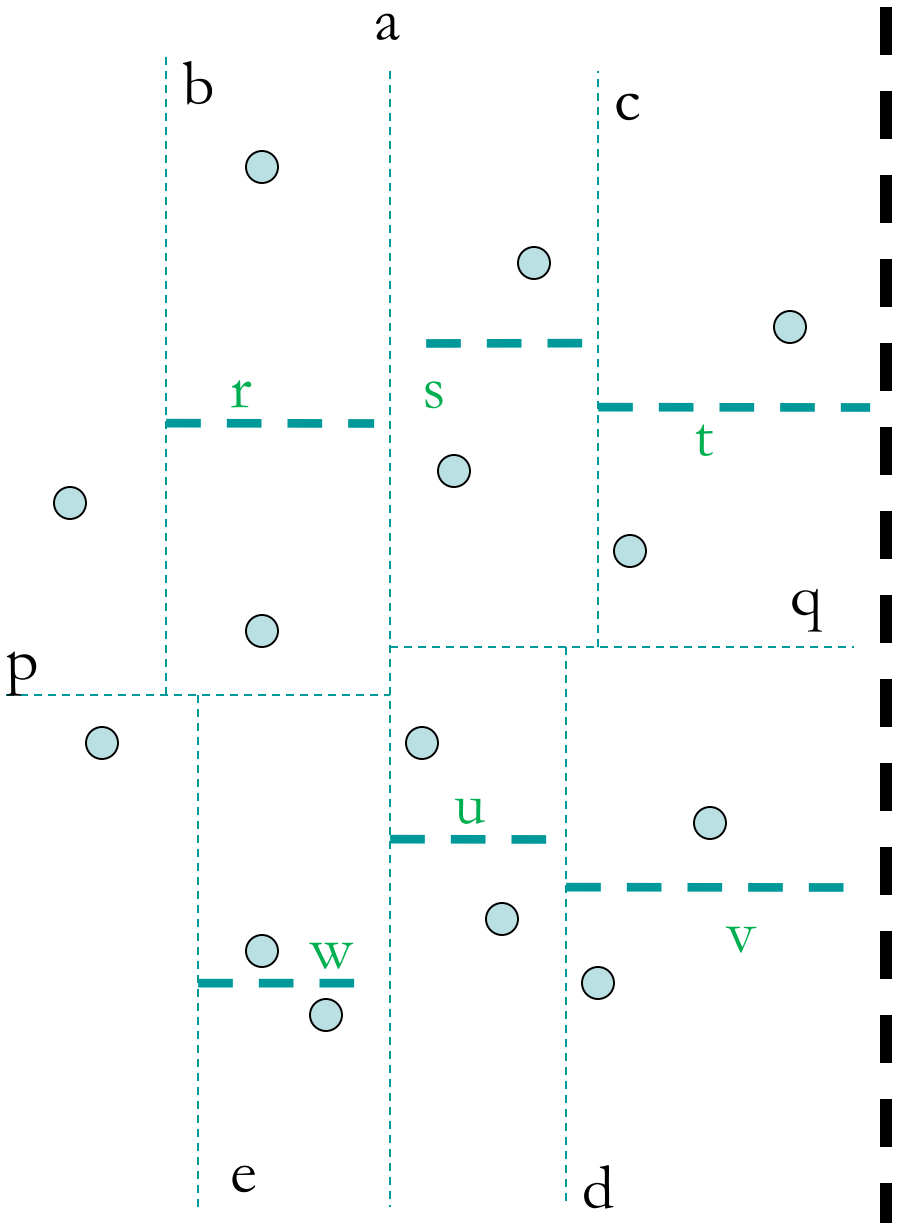


Data

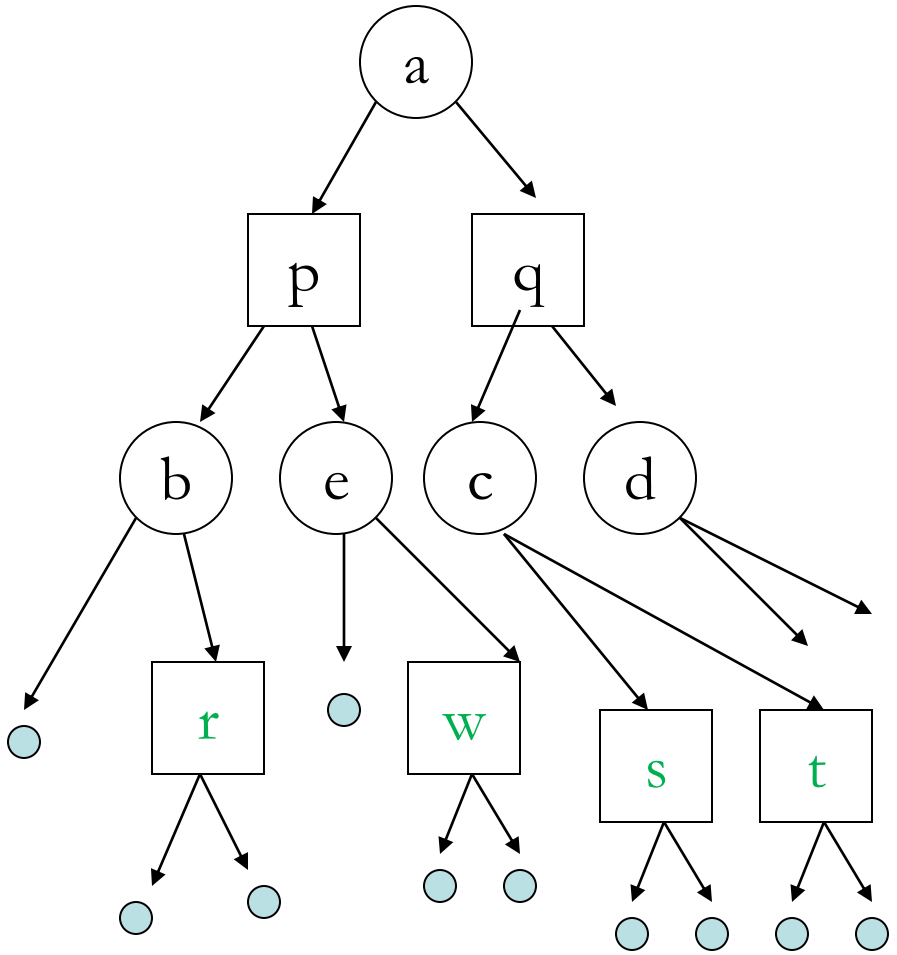


K-D Tree

Split on Y

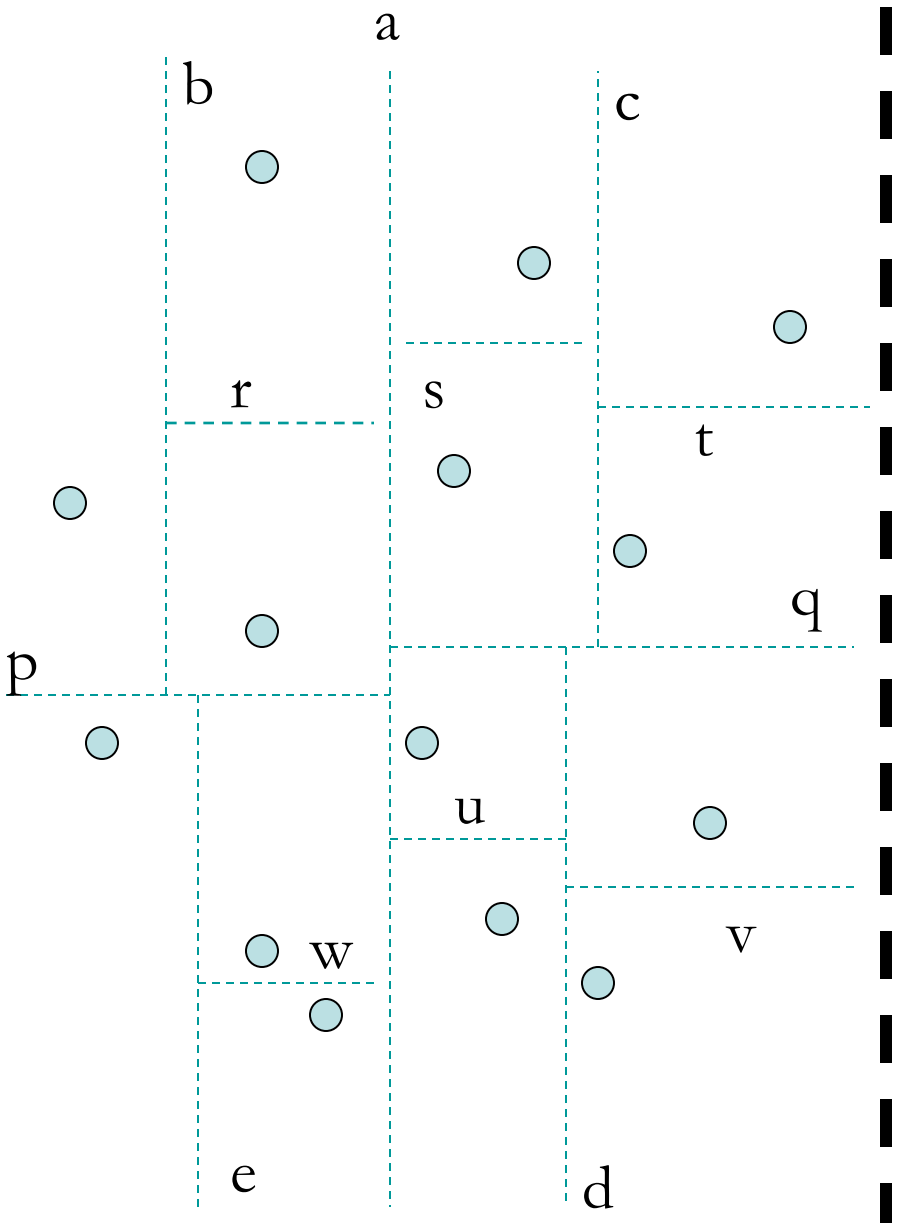


Data

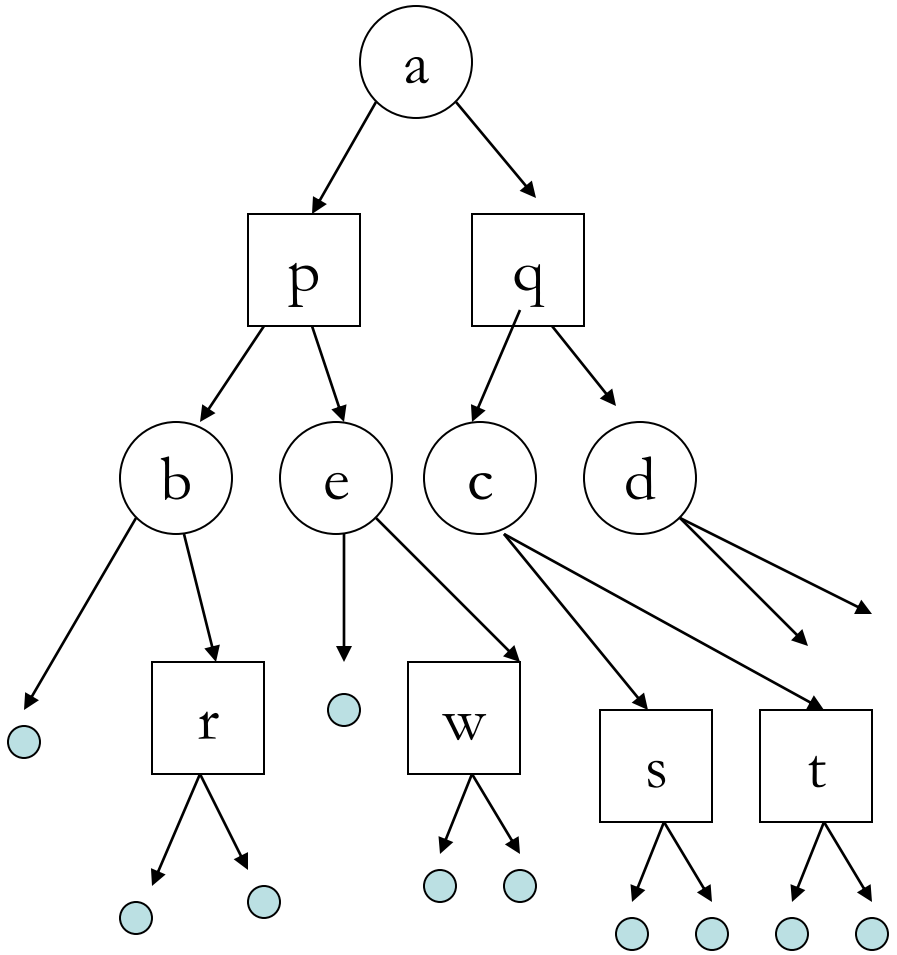


K-D Tree

K-D Tree

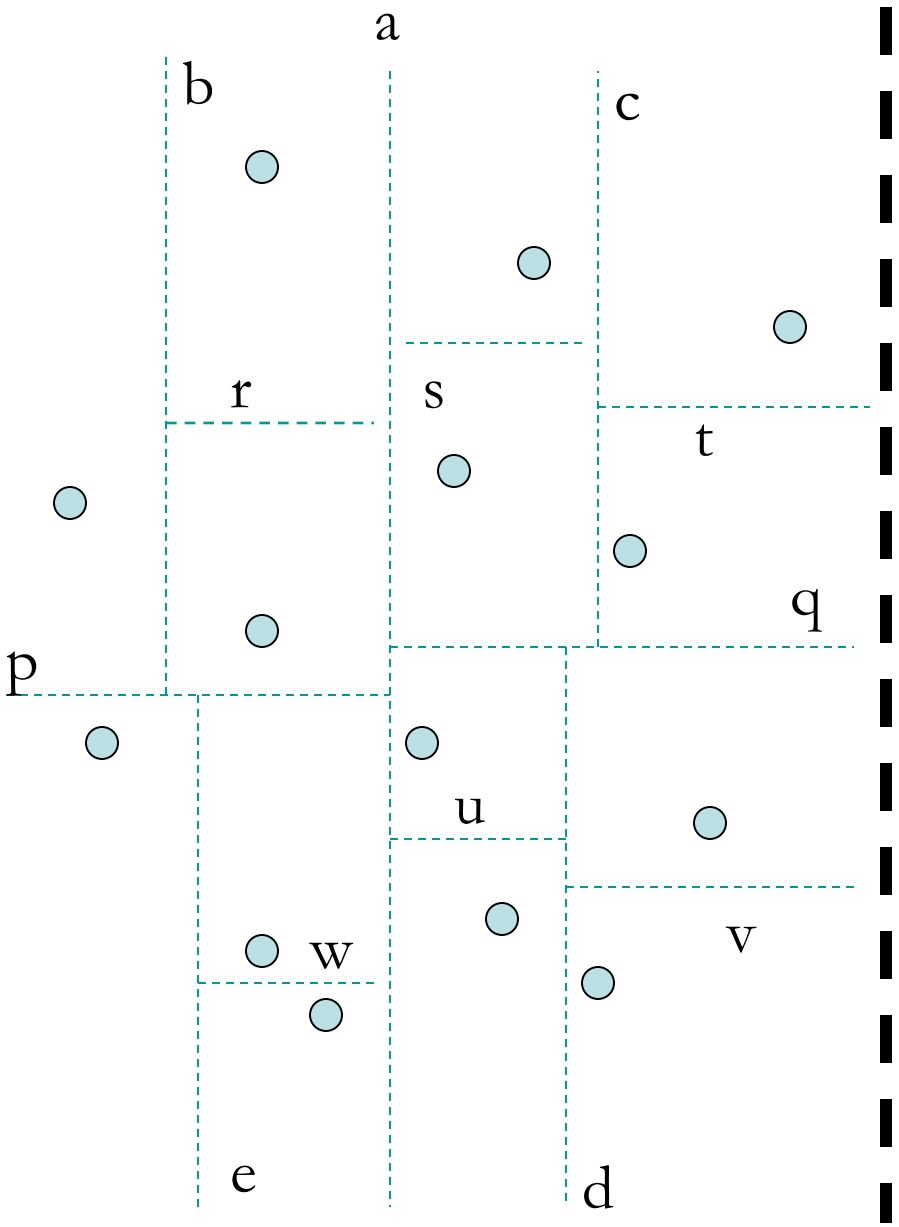


Data

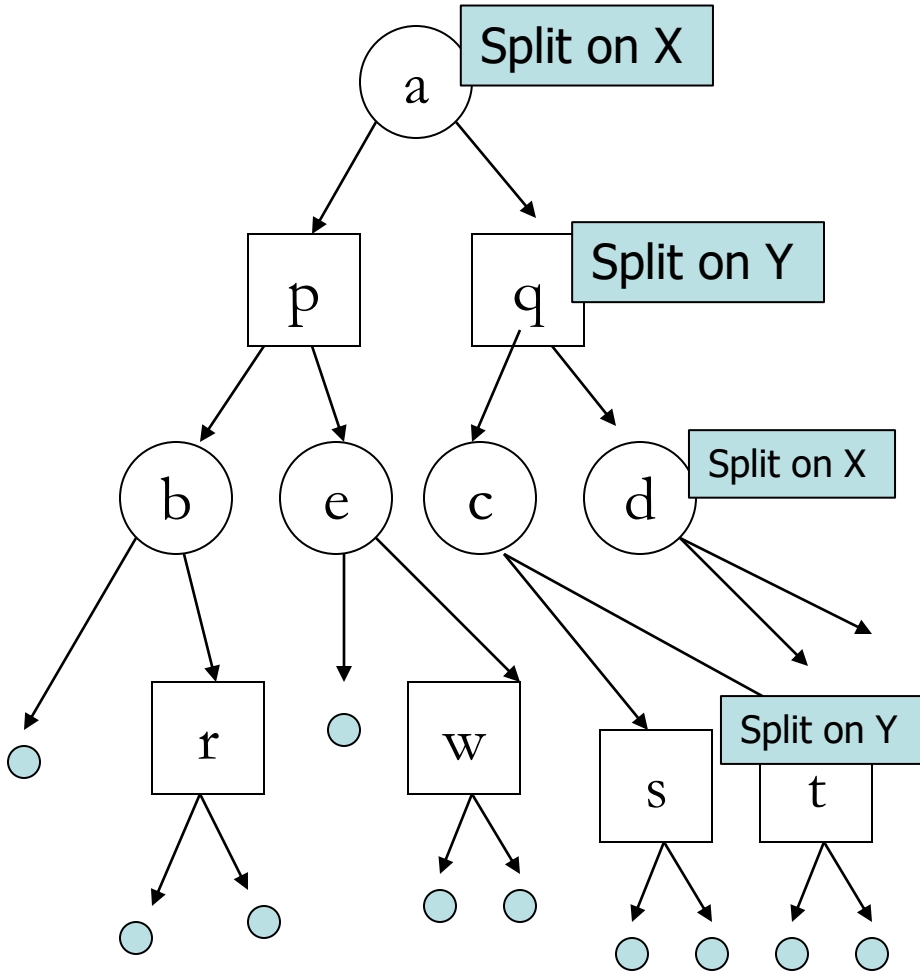


K-D Tree

K-D Tree

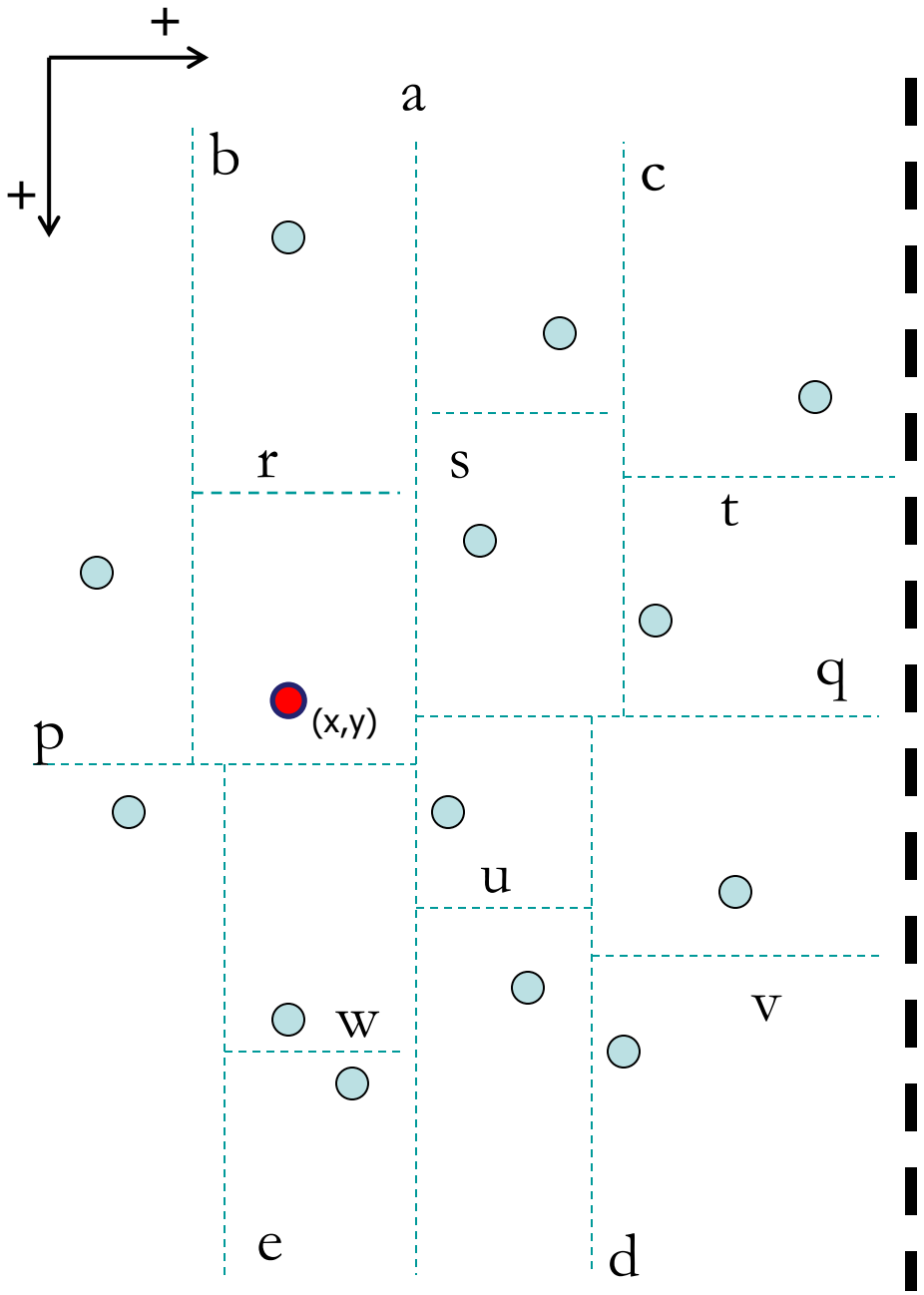


Data

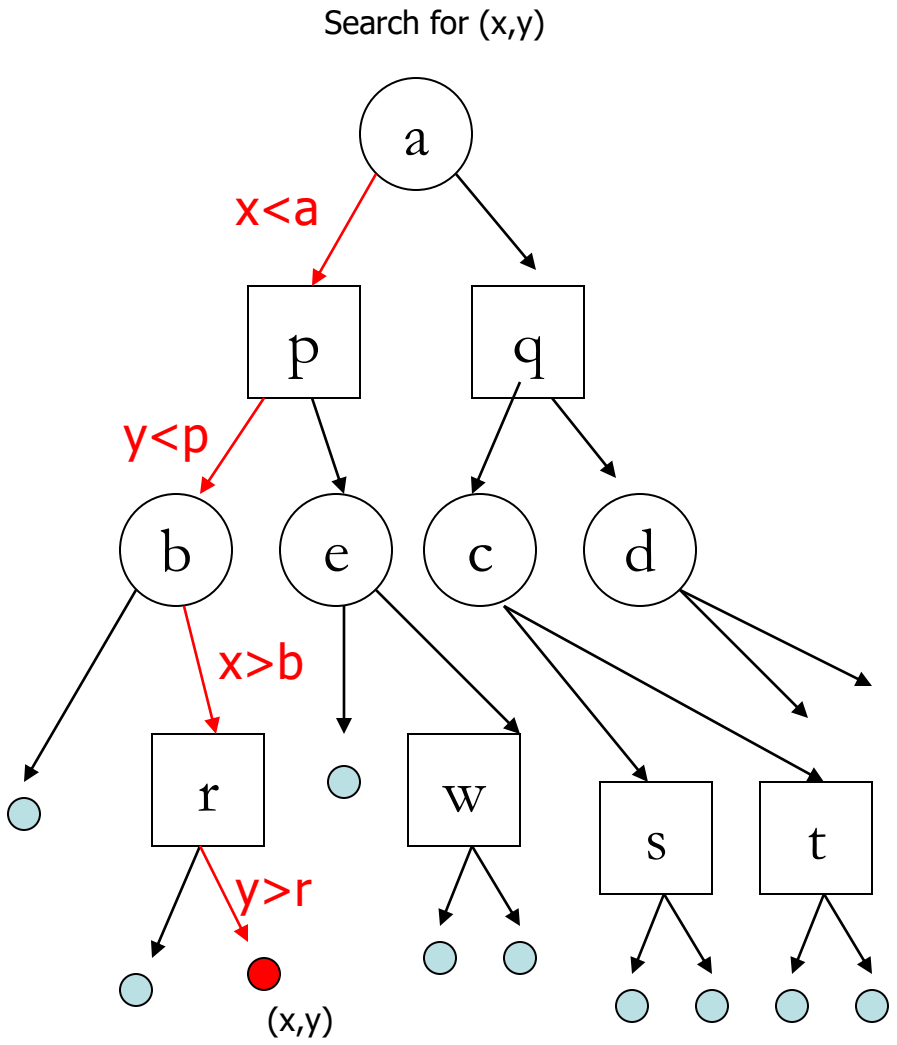


K-D Tree

Point Query

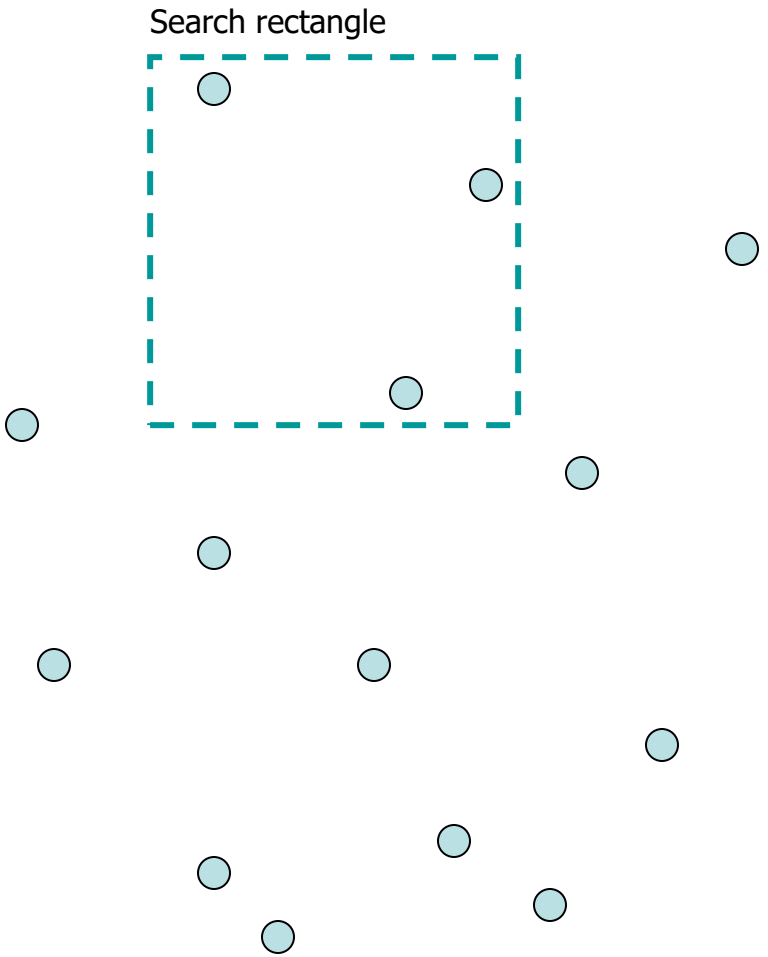


Data

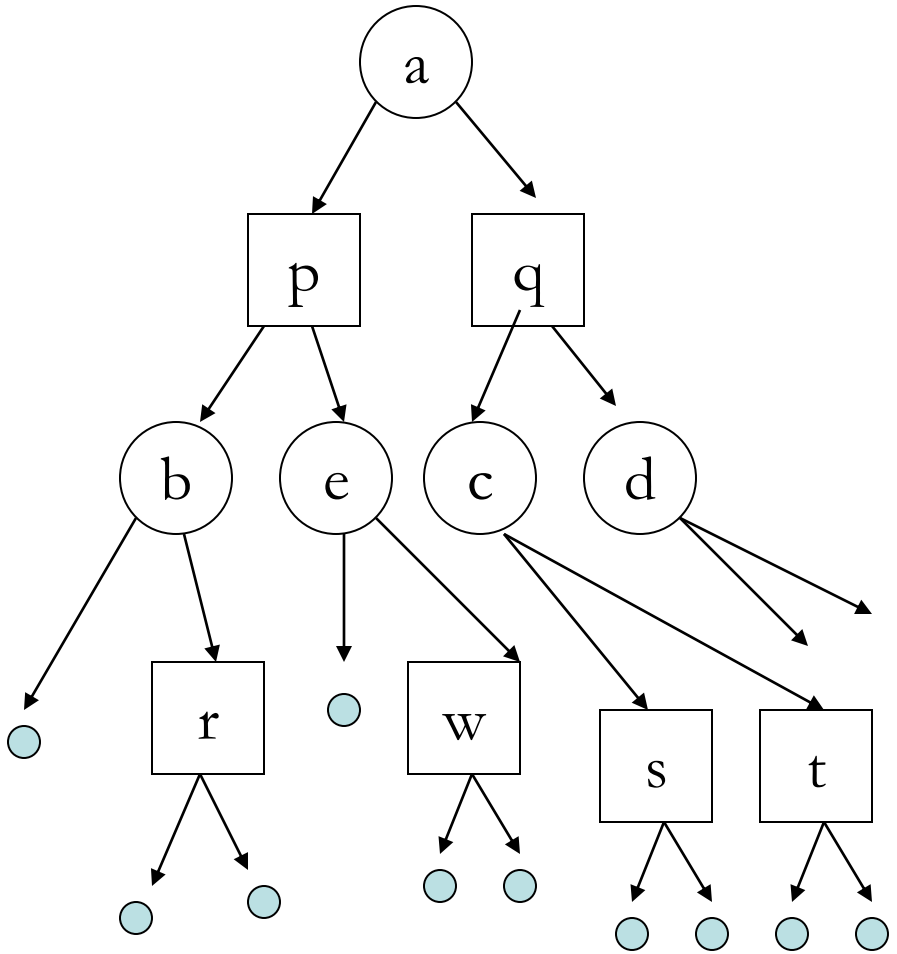


K-D Tree

Range Query

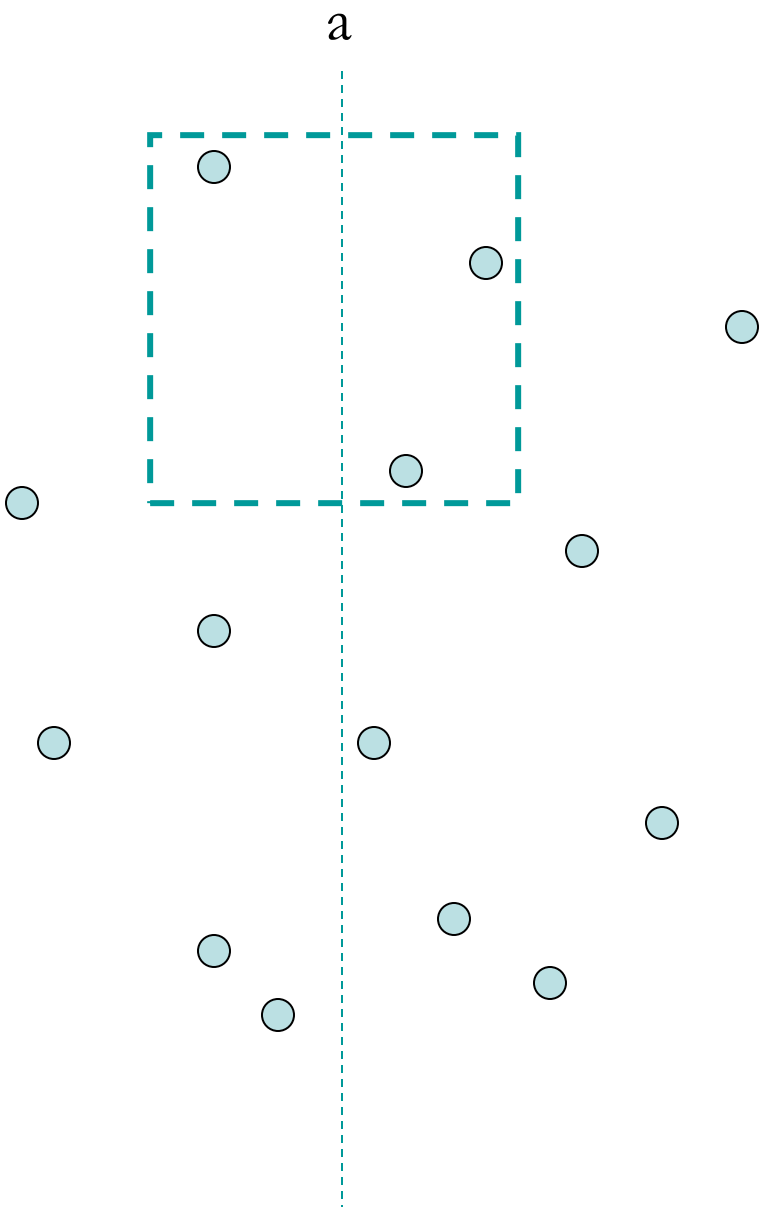


Data

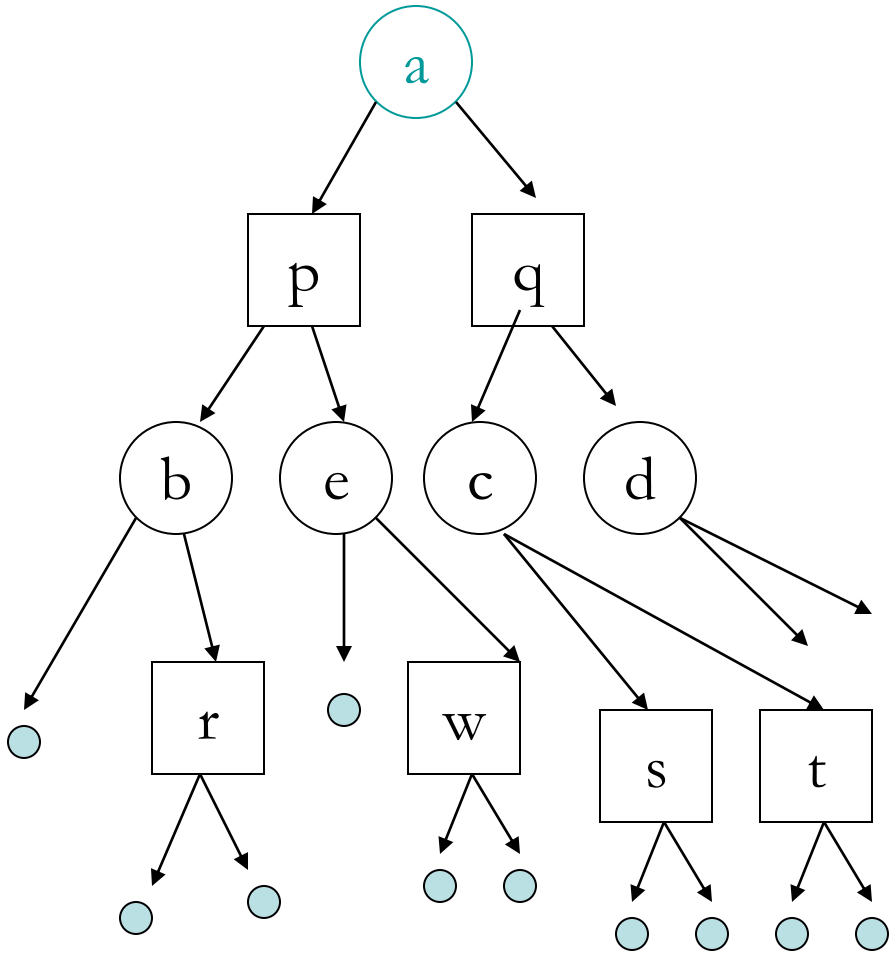


K-D Tree

K-D Tree

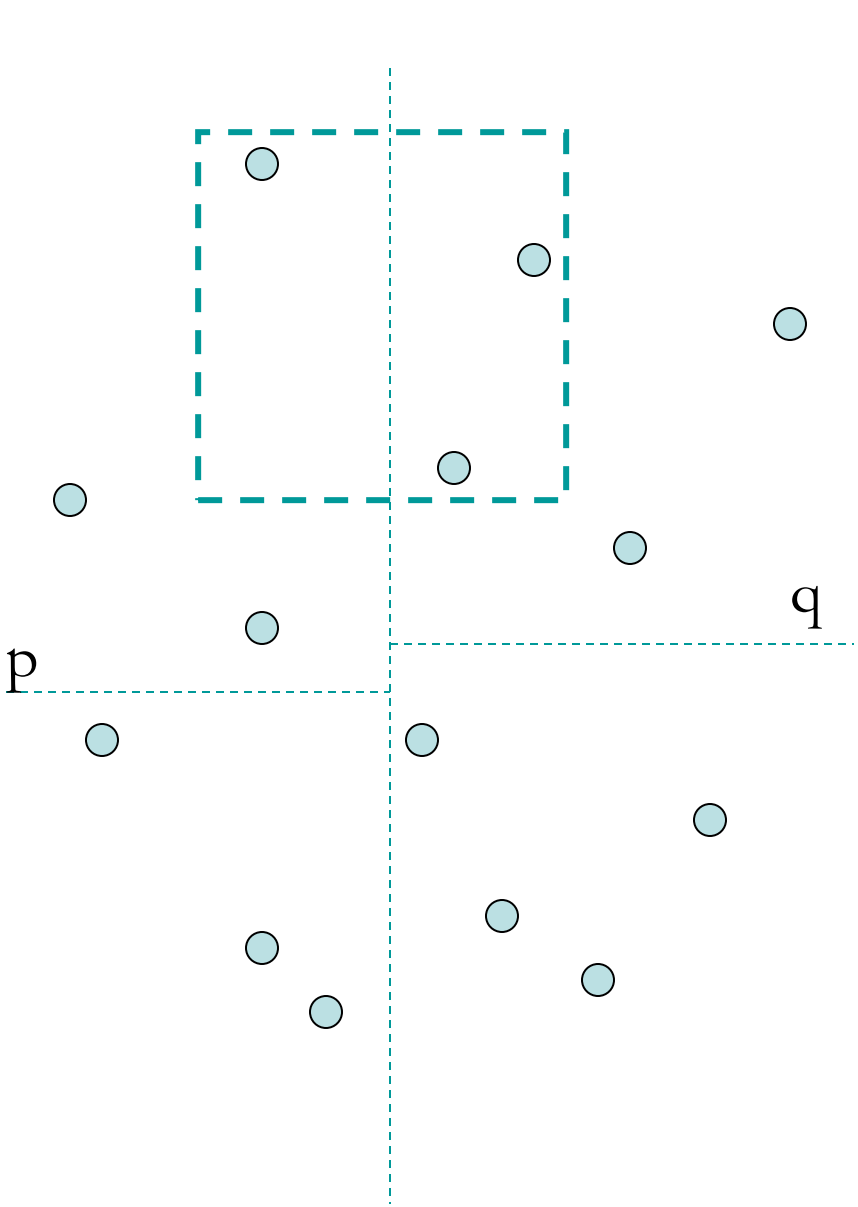


Data

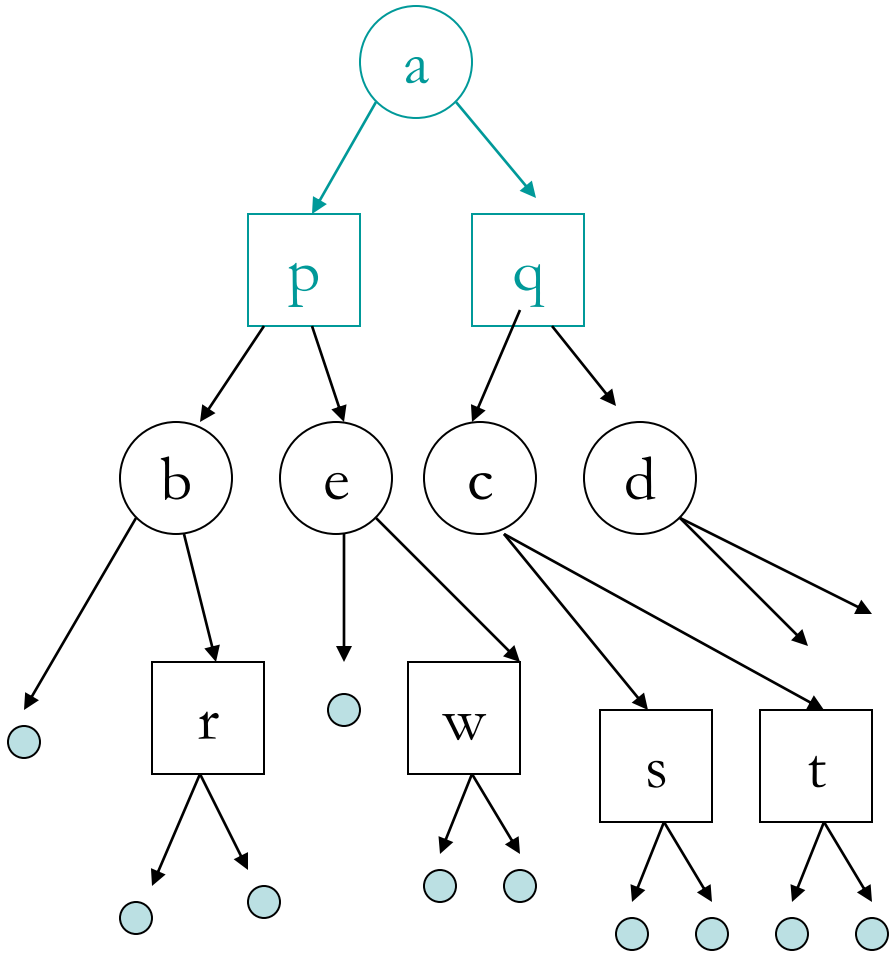


K-D Tree

K-D Tree

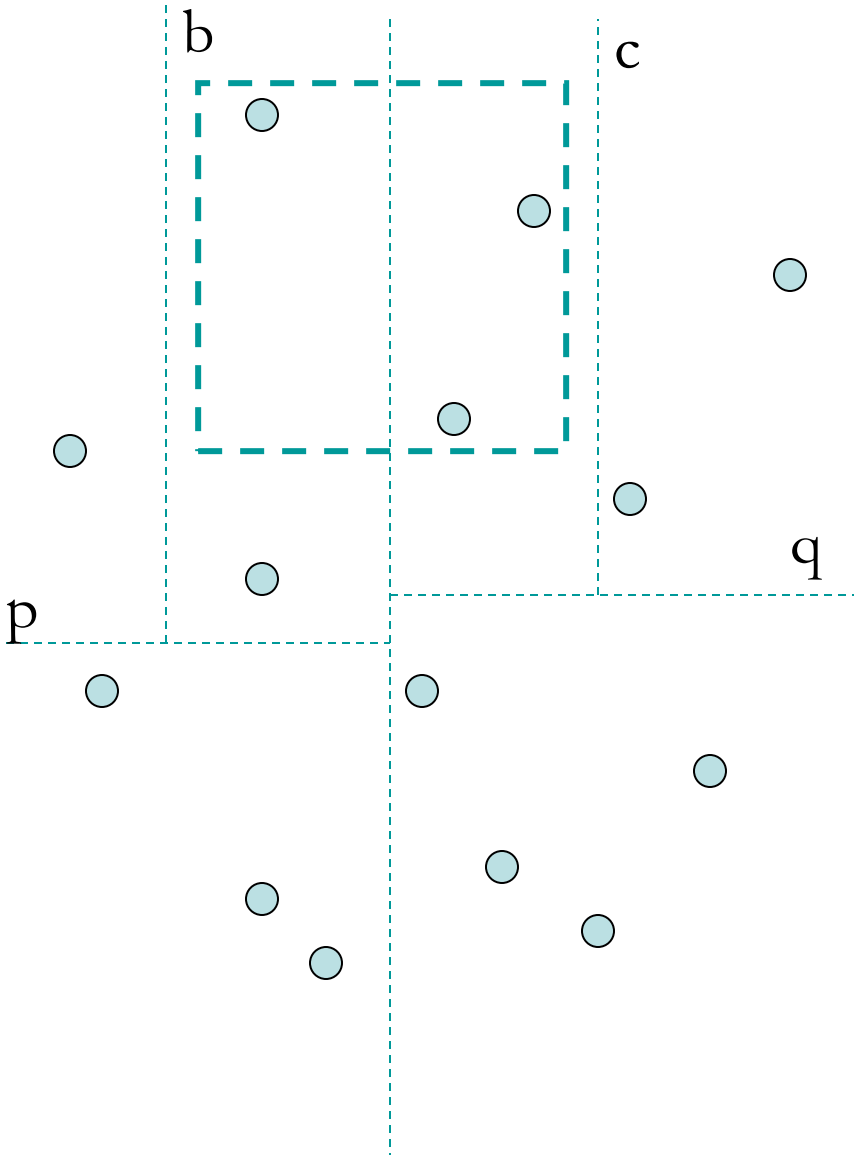


Data

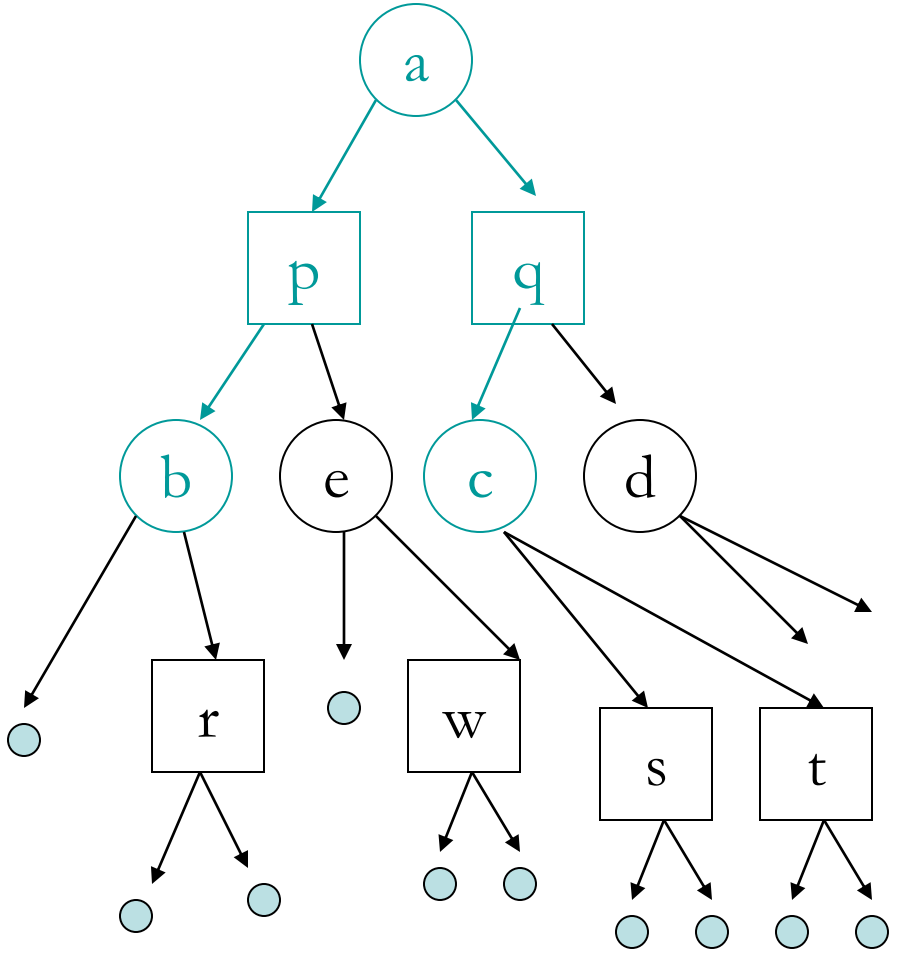


K-D Tree

K-D Tree

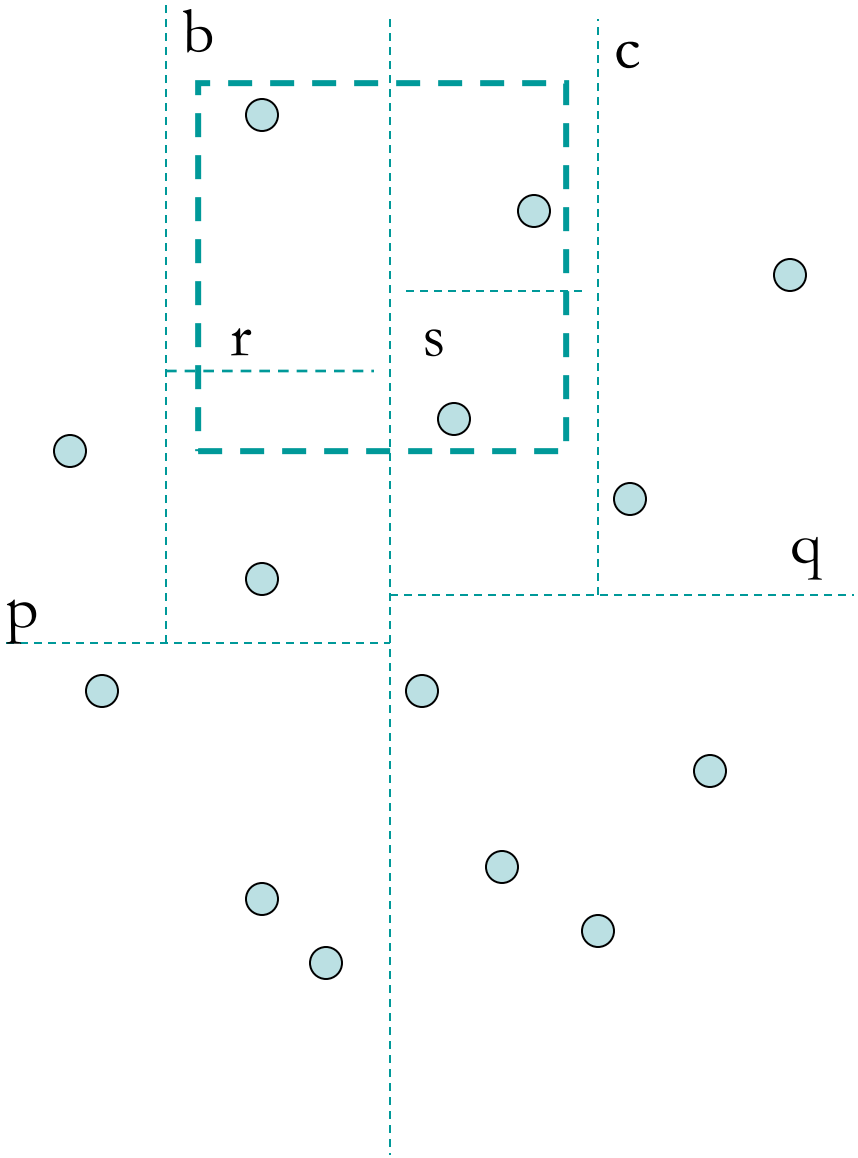


Data

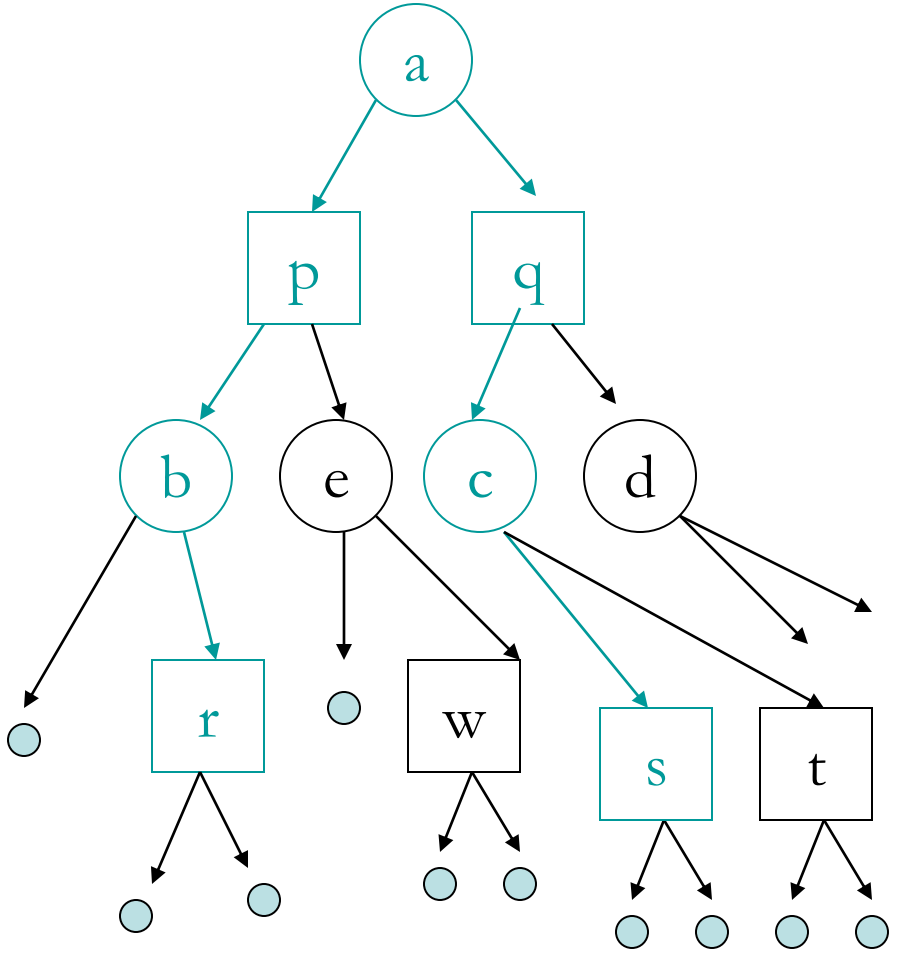


K-D Tree

K-D Tree

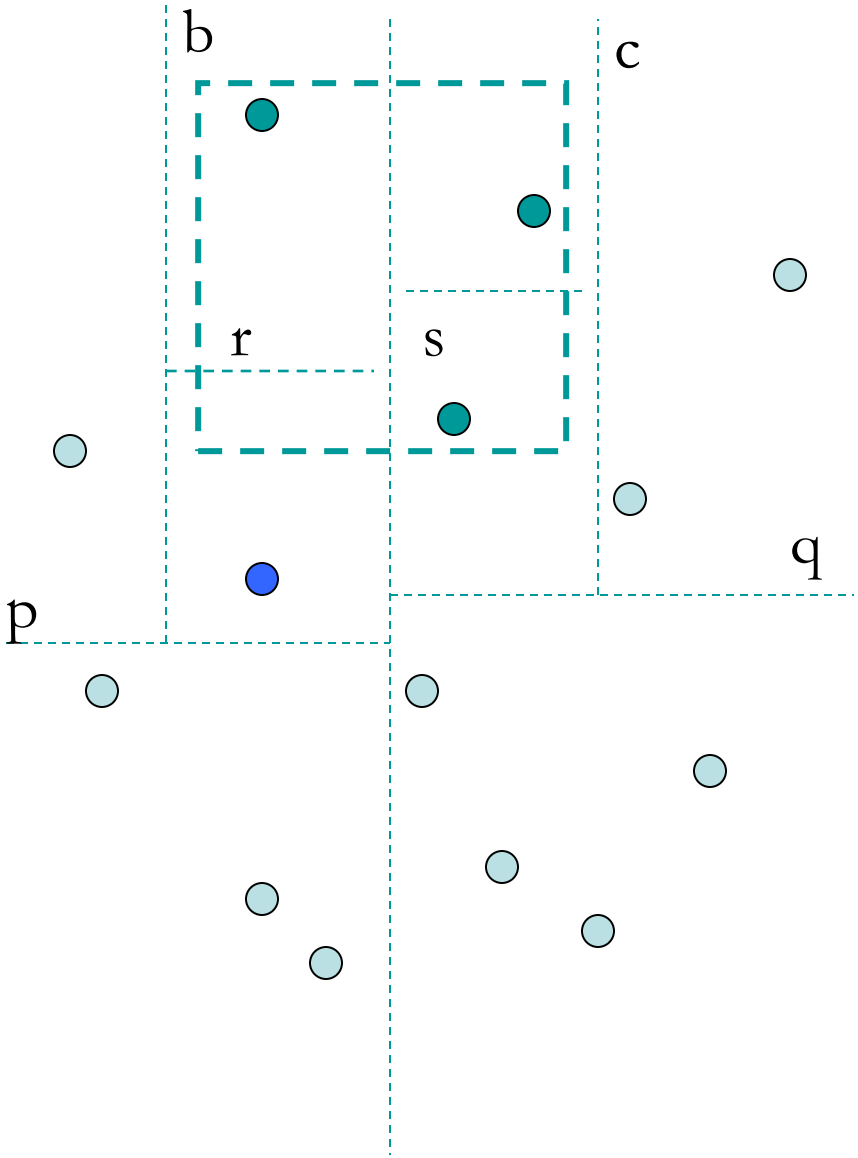


Data

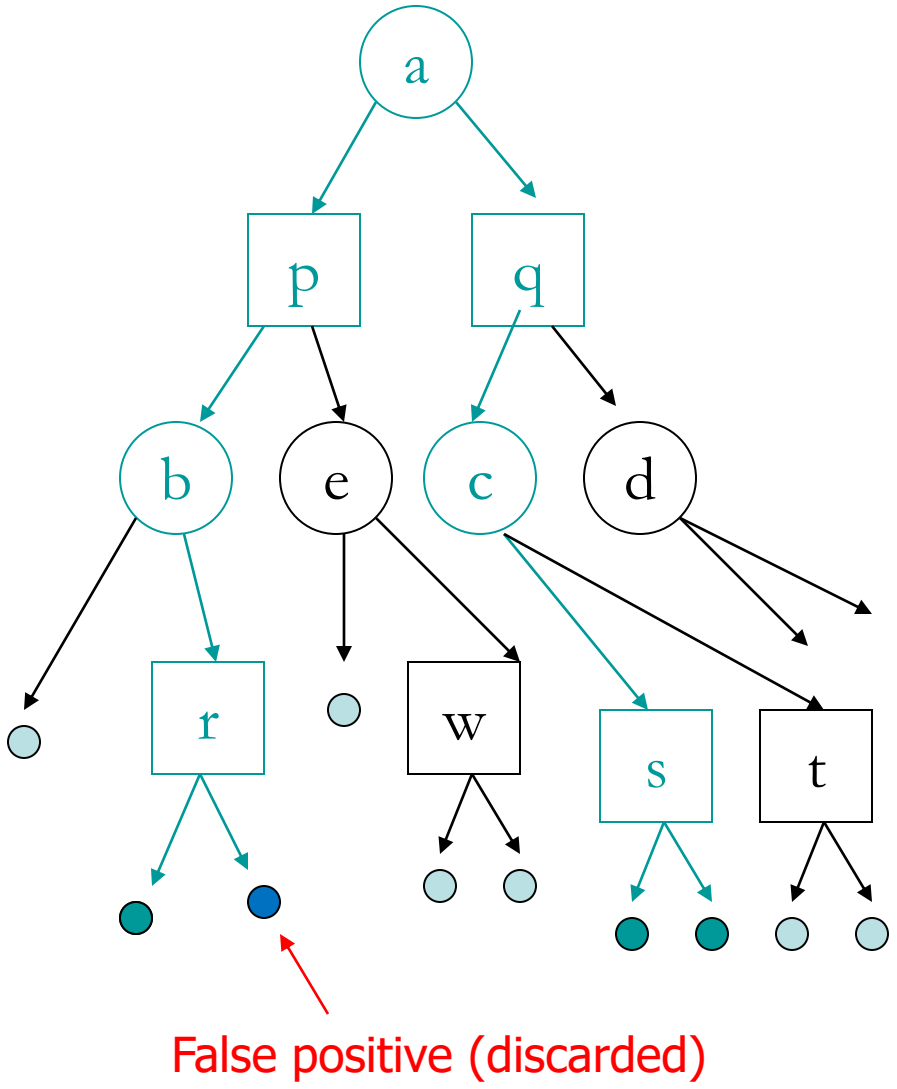


K-D Tree

K-D Tree



Data



K-D Tree

K-D Tree: Range Search

Efficiency:

$O(\sqrt{N} + \text{output size})$ (2 dimensions)

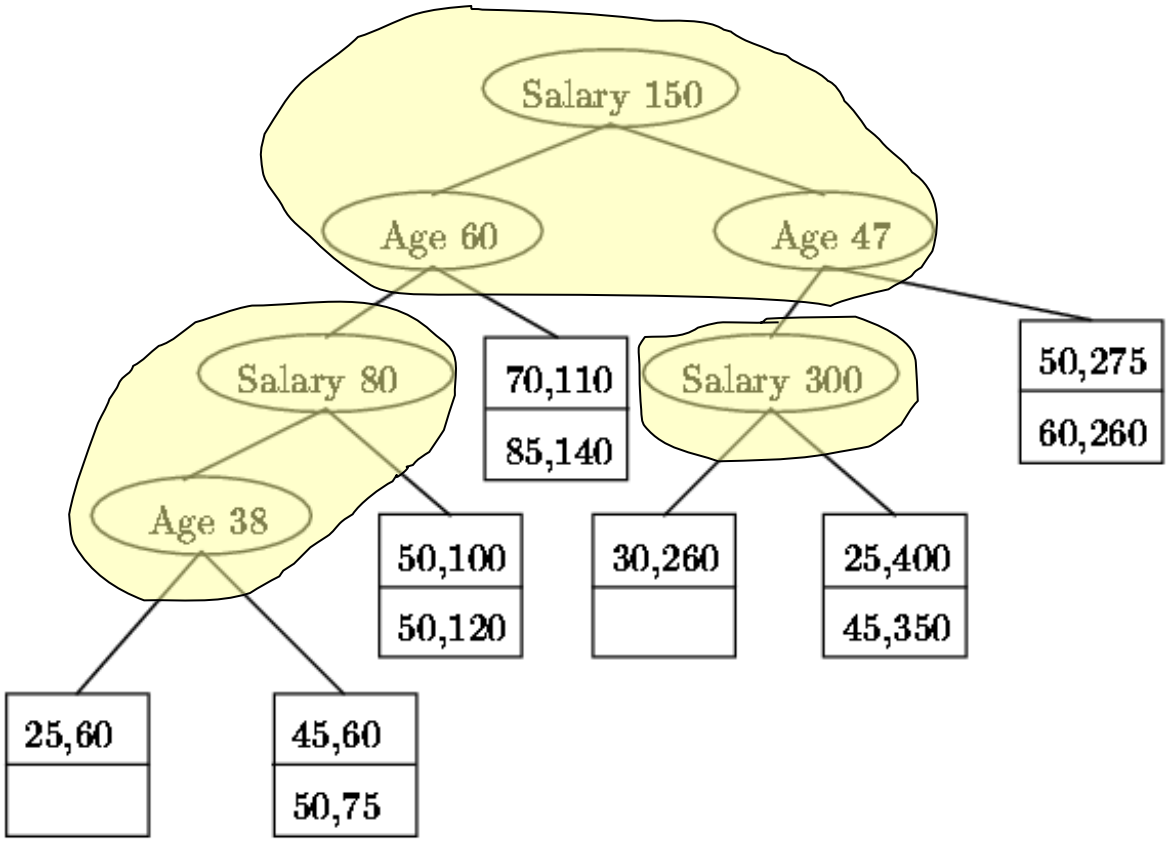
$O(N^{1-1/d} + \text{output size})$ (d dimensions)

Same as linear search (plus cost of index) in higher dimensions

N: # of indexed points

KD-trees in secondary storage

- 1000 leaves ➔ $\log_2(1000)$ = 10 levels.
- If each internal node is stored in one block then too many disk I/O's
- Solution: Group nodes into blocks



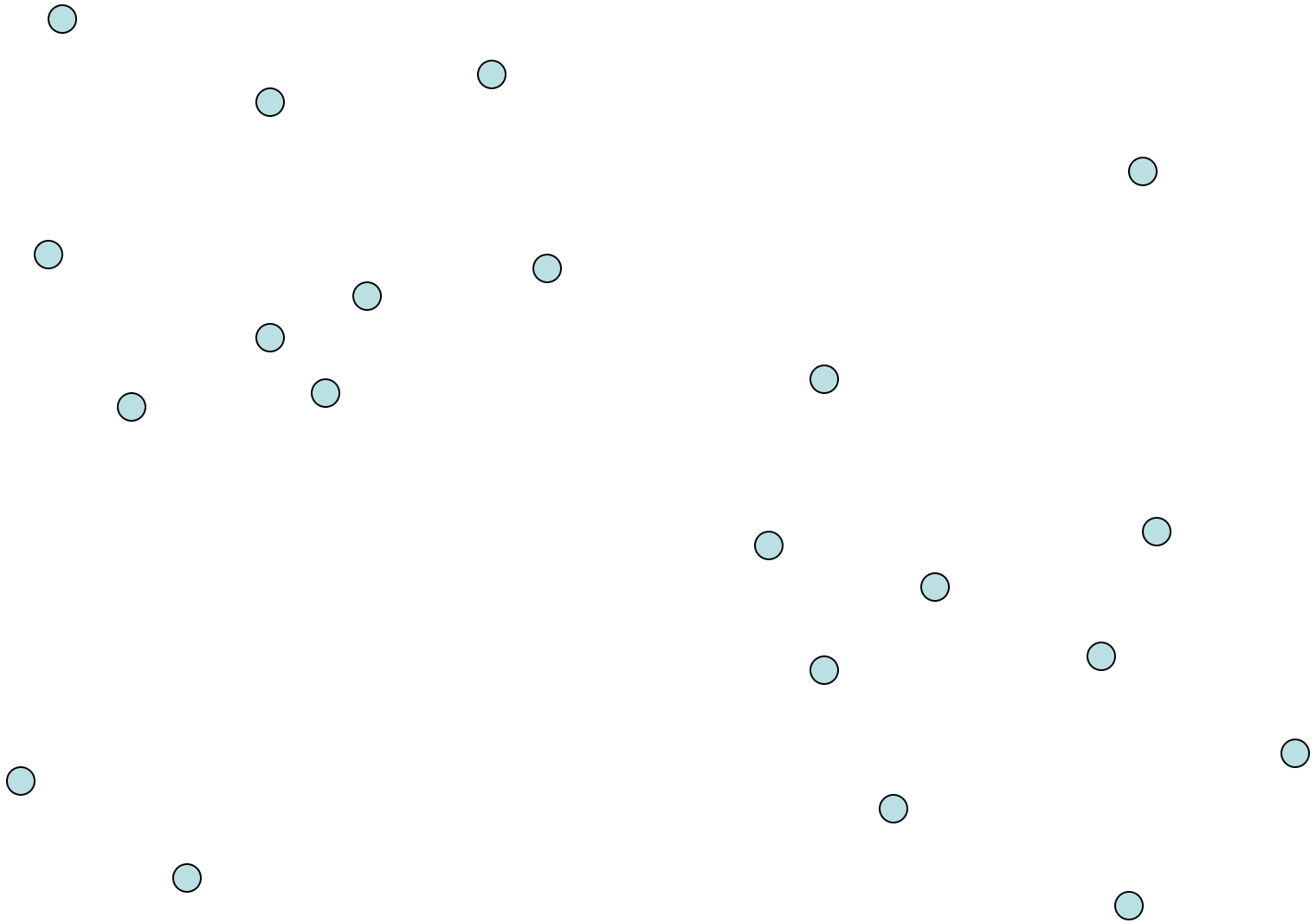
K-D-B Tree: Disk based K-D Tree

- More than one points per leaf
- 1 Leaf = 1 Disk block
- Pack multiple internal nodes into 1 disk block

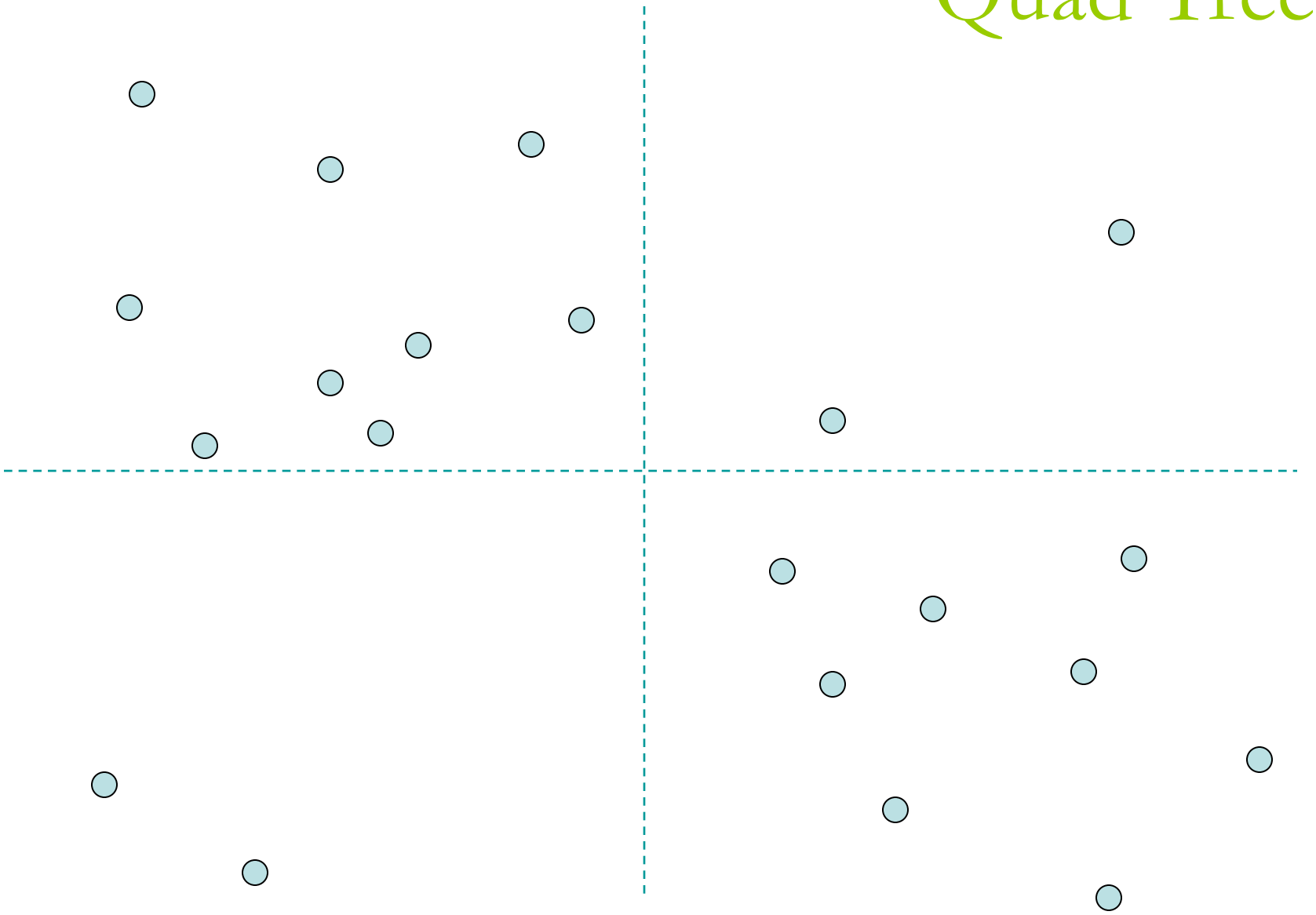
Not very clean

No strong guarantees about space

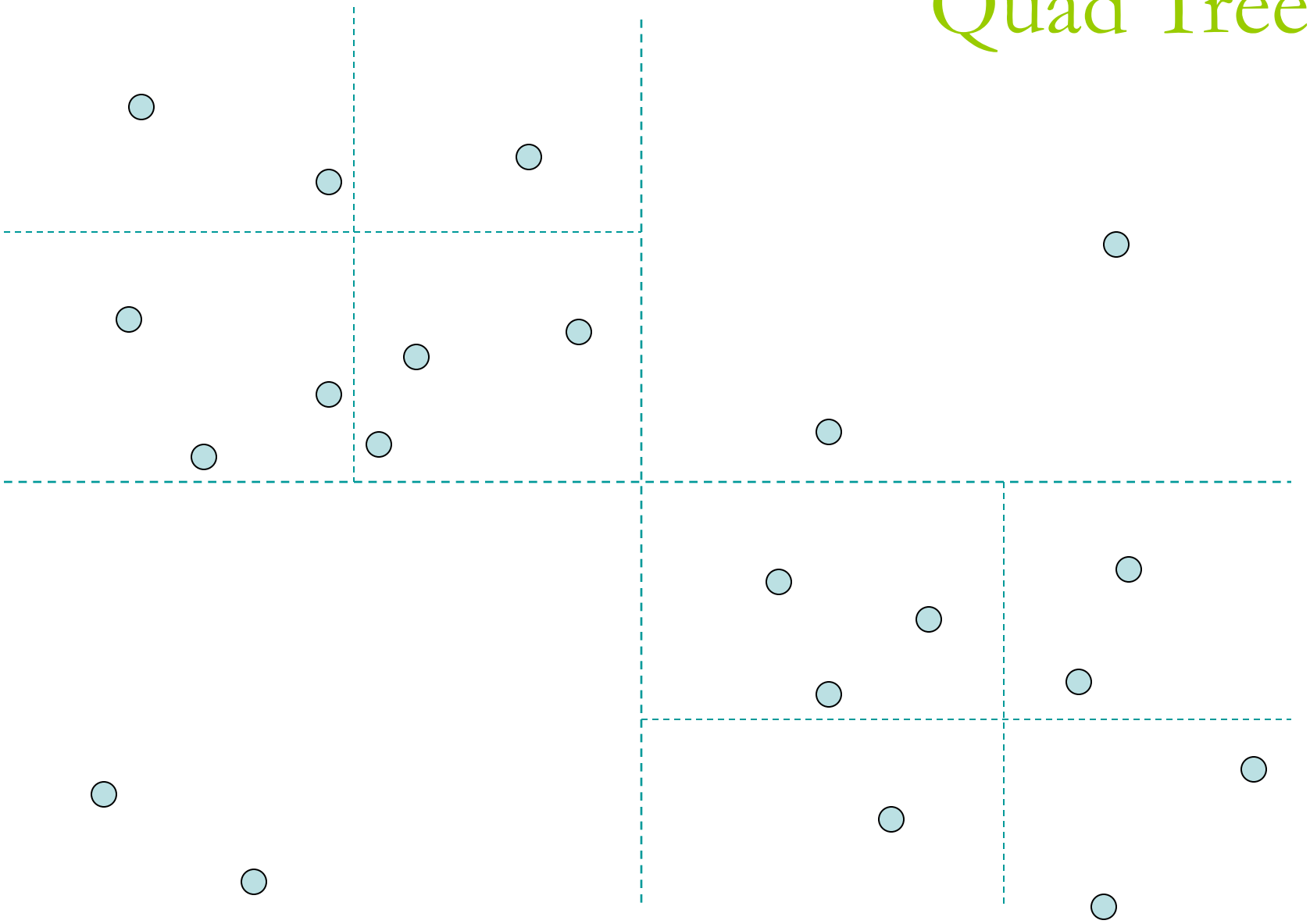
Quad Trees



Quad Trees

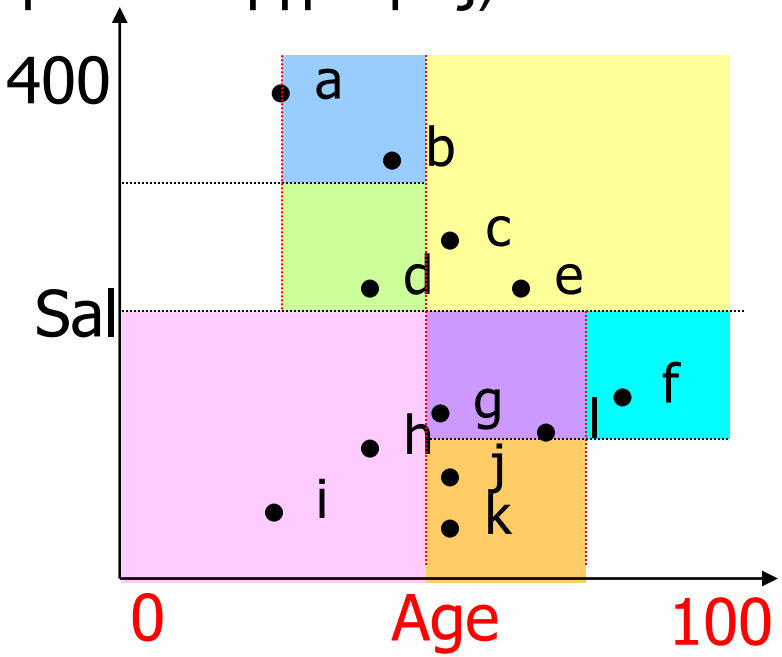


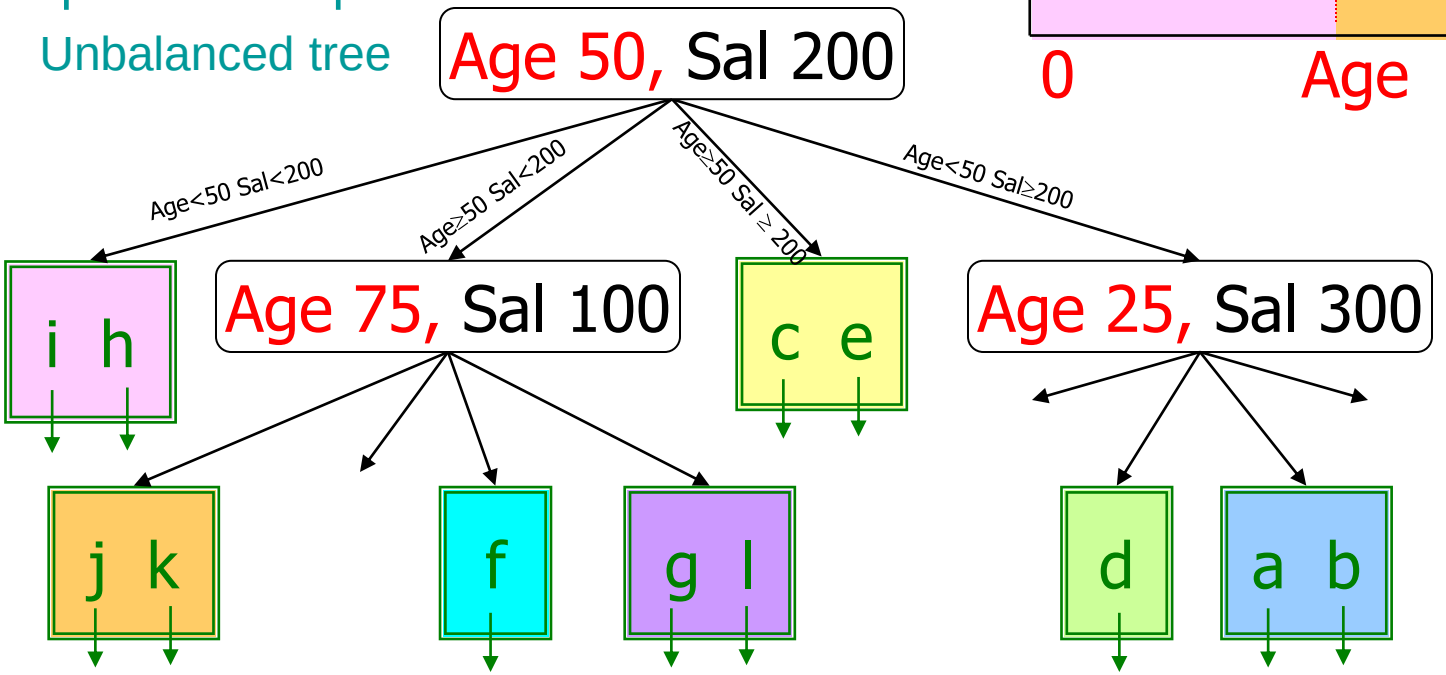
Quad Trees



Quad trees Παράδειγμα 2
(υποθέτω κάθε φύλλο χωράει 2 εγγραφές)

- Σε αντίθεση με το K-D-tree οι κόμβοι σπάνε <u>ταυτόχρονα</u> σε όλες τις διαστάσεις
- Για ν διαστάσεις κάθε εσωτερικός κόμβος έχει 2^v παιδιά
- Σε κάθε διάσταση σπάω στη μέση το πεδίο τιμών
  - Unbalanced tree





Roadmap

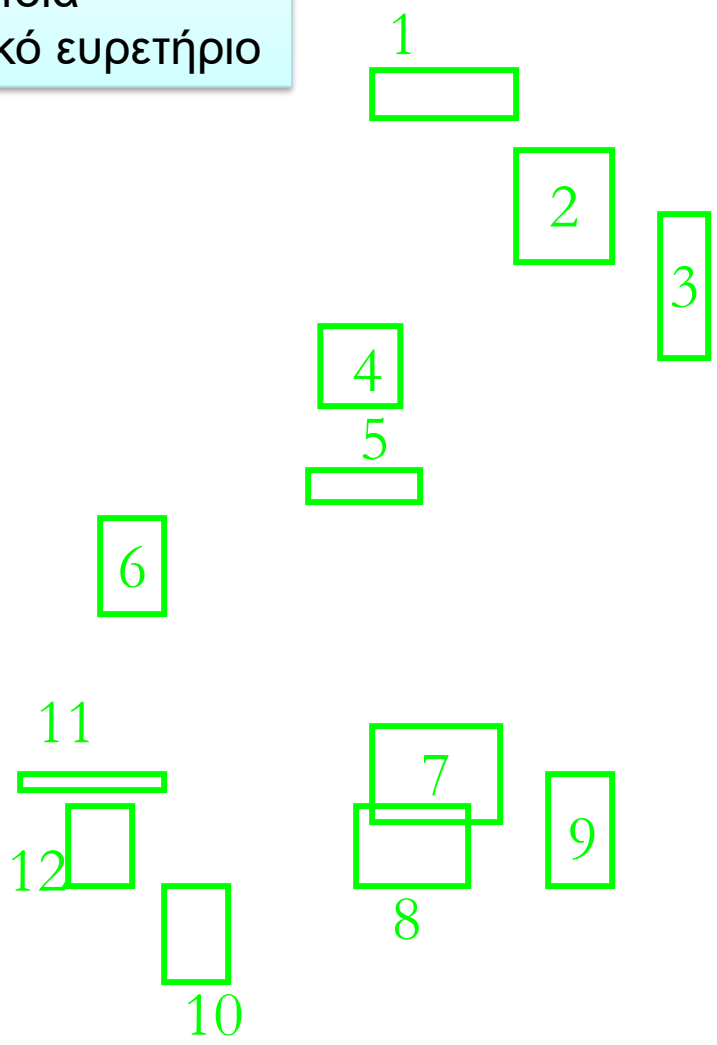
- Multi-Dimensional Indexes
  - Grid Files
  - K-D Tree
  - Quad Tree
  - R-Tree

R-Tree

- Χρησιμοποιείται για αντικείμενα d-διαστάσεων (d-dimensional boxes)
- Υποστηρίζει πλήθος ερωτημάτων
  - Range queries (containment)
  - Intersection queries (overlap)
  - “Proximity” queries
    - NN-Queries, Reverse NN-Queries
  - Πλήθος άλλων εφαρμογών: skylines, cubetrees, etc

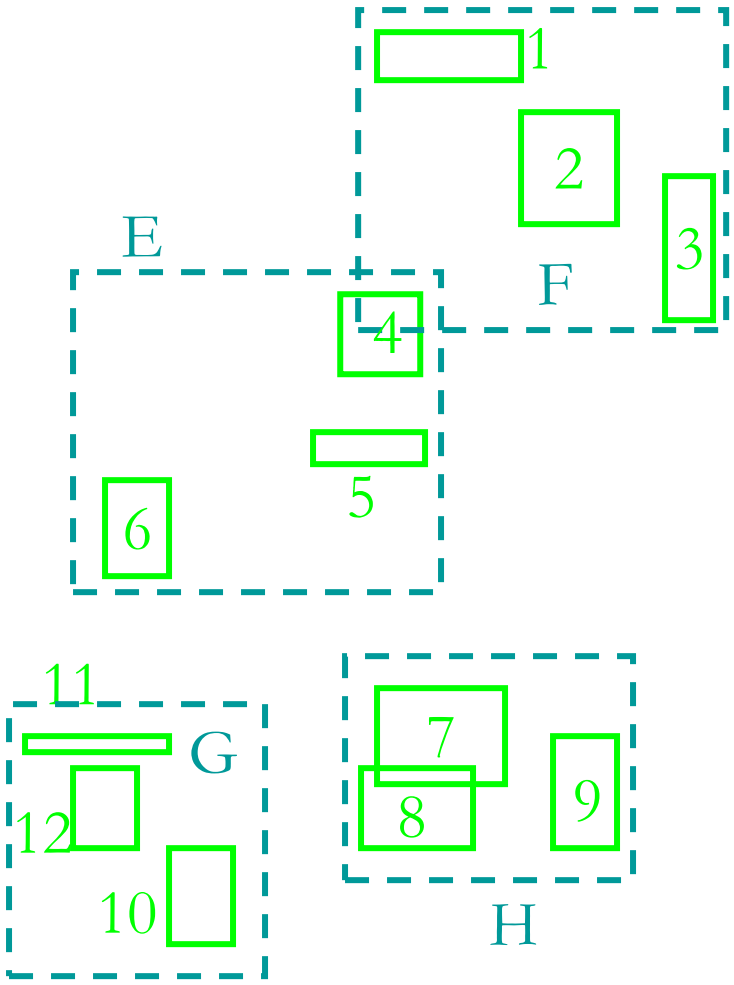
R-Tree

Τα 1..12 είναι δισδιάστατα αντικείμενα για τα οποία χρειάζομαι ένα χωρικό ευρετήριο

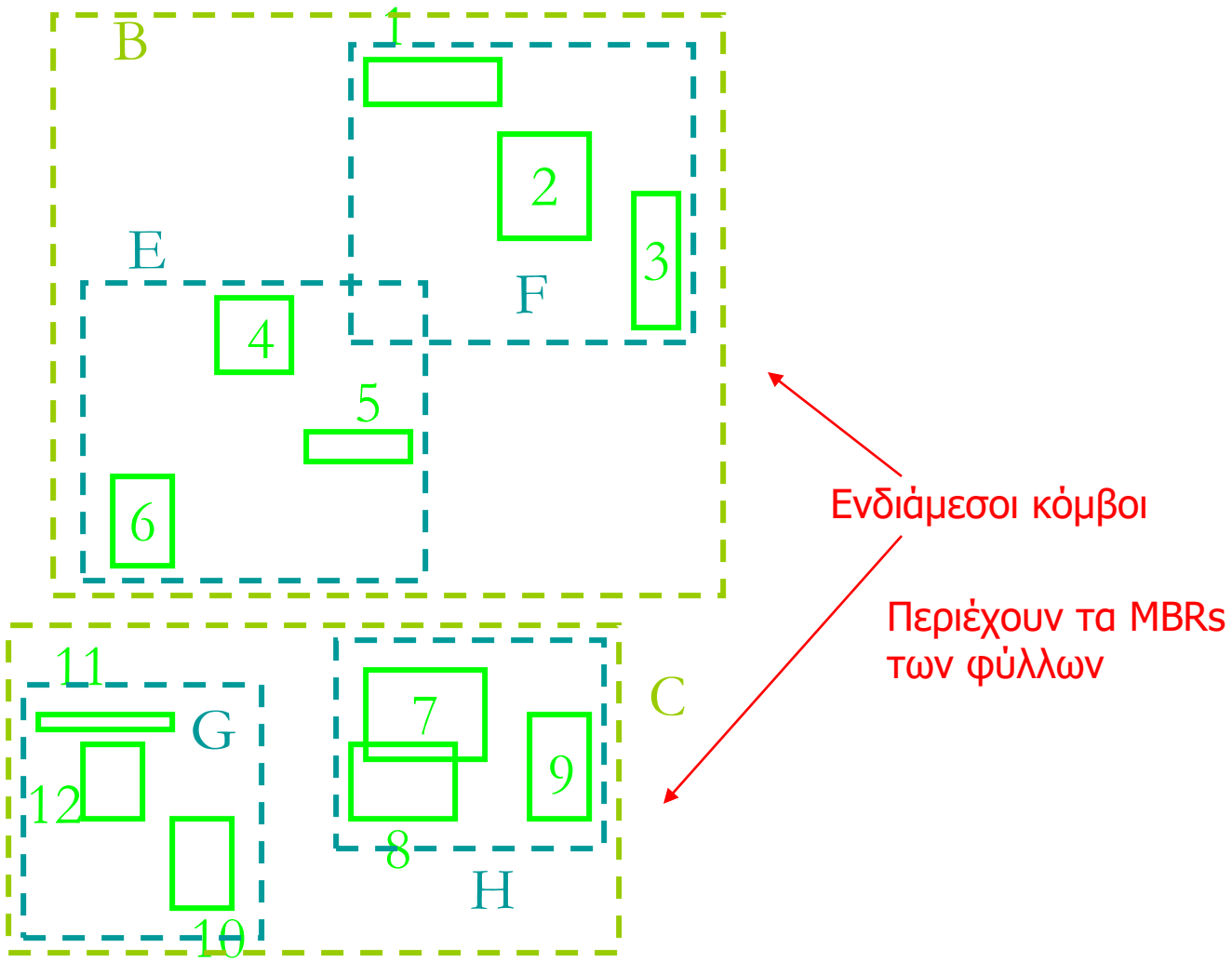


Minimum Bounding Rectangles

- Τα χωρικά δεδομένα ομαδοποιούνται στα φύλλα
  - Κάθε ομάδα περιγράφεται μέσω ενός νέου ορθογωνίου (MBR) που την περικλείει
- Τα MBRs ομαδοποιούνται με τη σειρά τους στα ανώτερα επίπεδα του R-tree
- Στο παράδειγμα τα E,F,G,H είναι τα MBRs των φύλλων του δέντρου

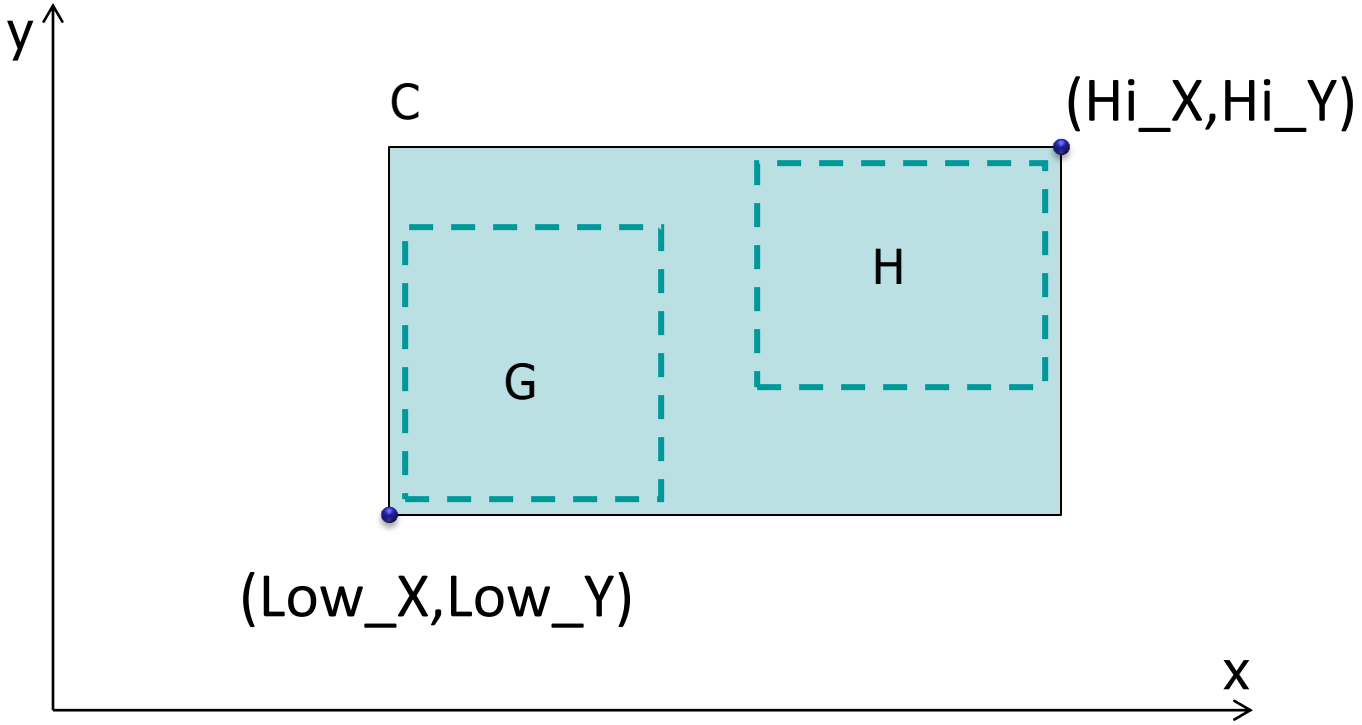


R-Tree


B
1
2
3
E
F
4
5
6
Ενδιάμεσοι κόμβοι
Περιέχουν τα MBRs
των φύλλων
11
G
12
10
7
9
8
H
C

Minimum Bounding Rectangle (MBR)

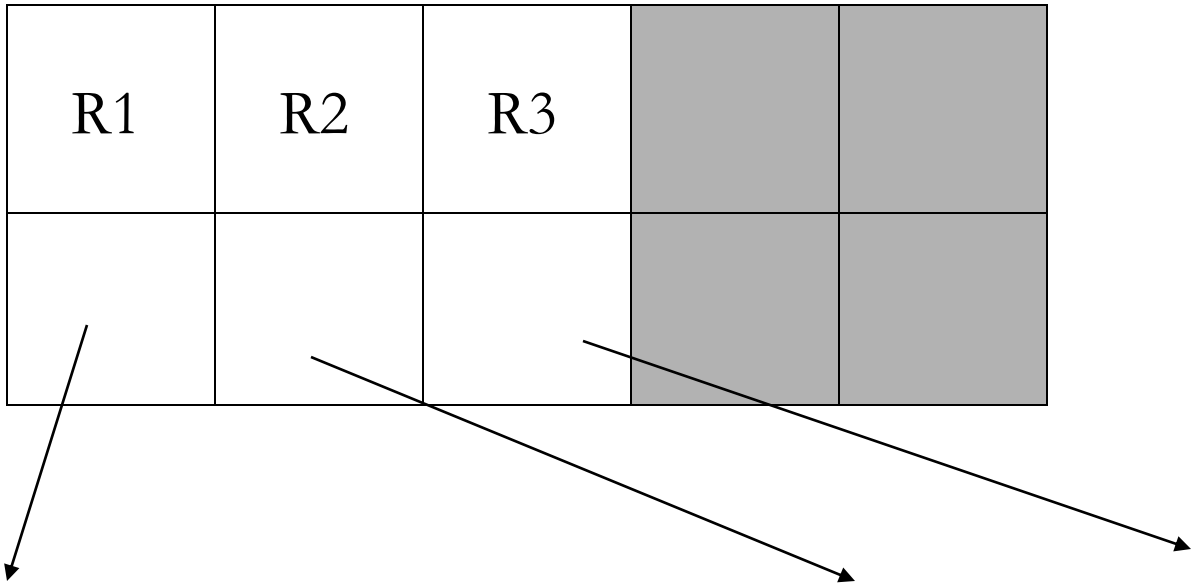
- C = {Low_X,Low_Y,Hi_X,Hi_Y)



R-Tree: Εσωτερικοί κόμβοι

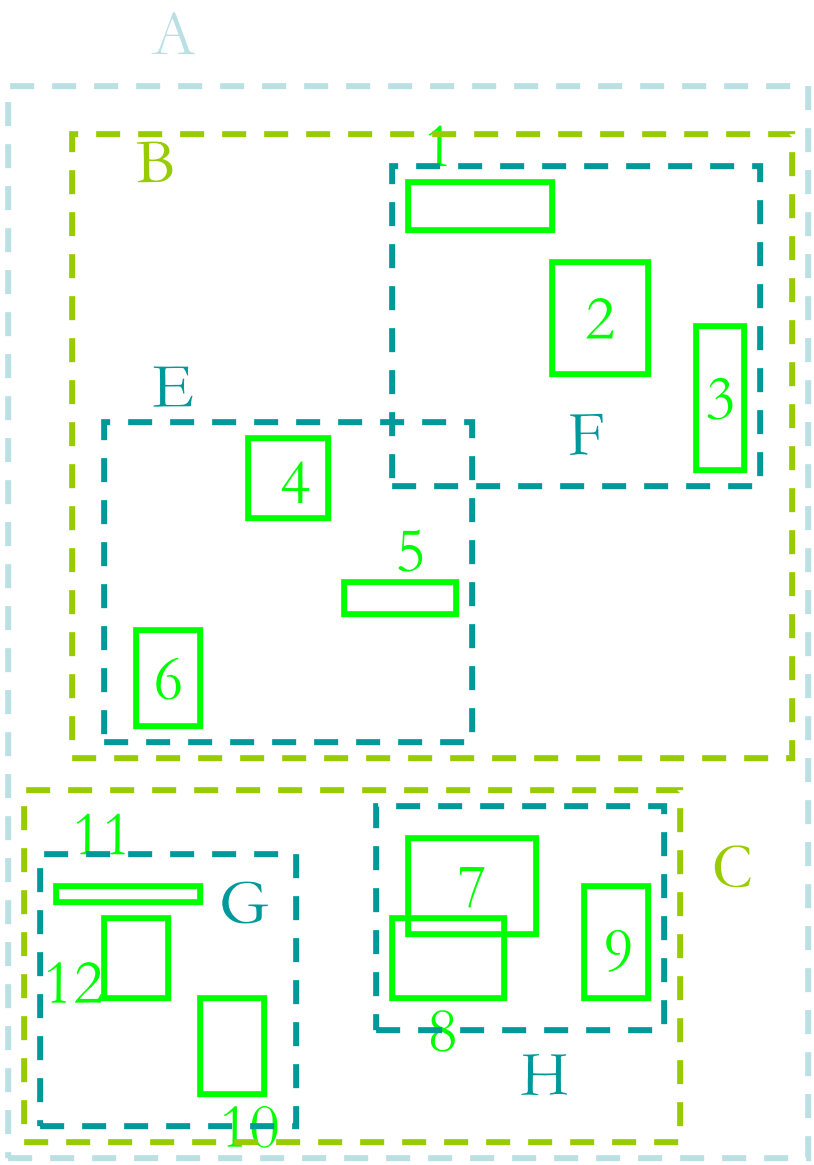
n "key" slots (MBRs)

n pointer slots



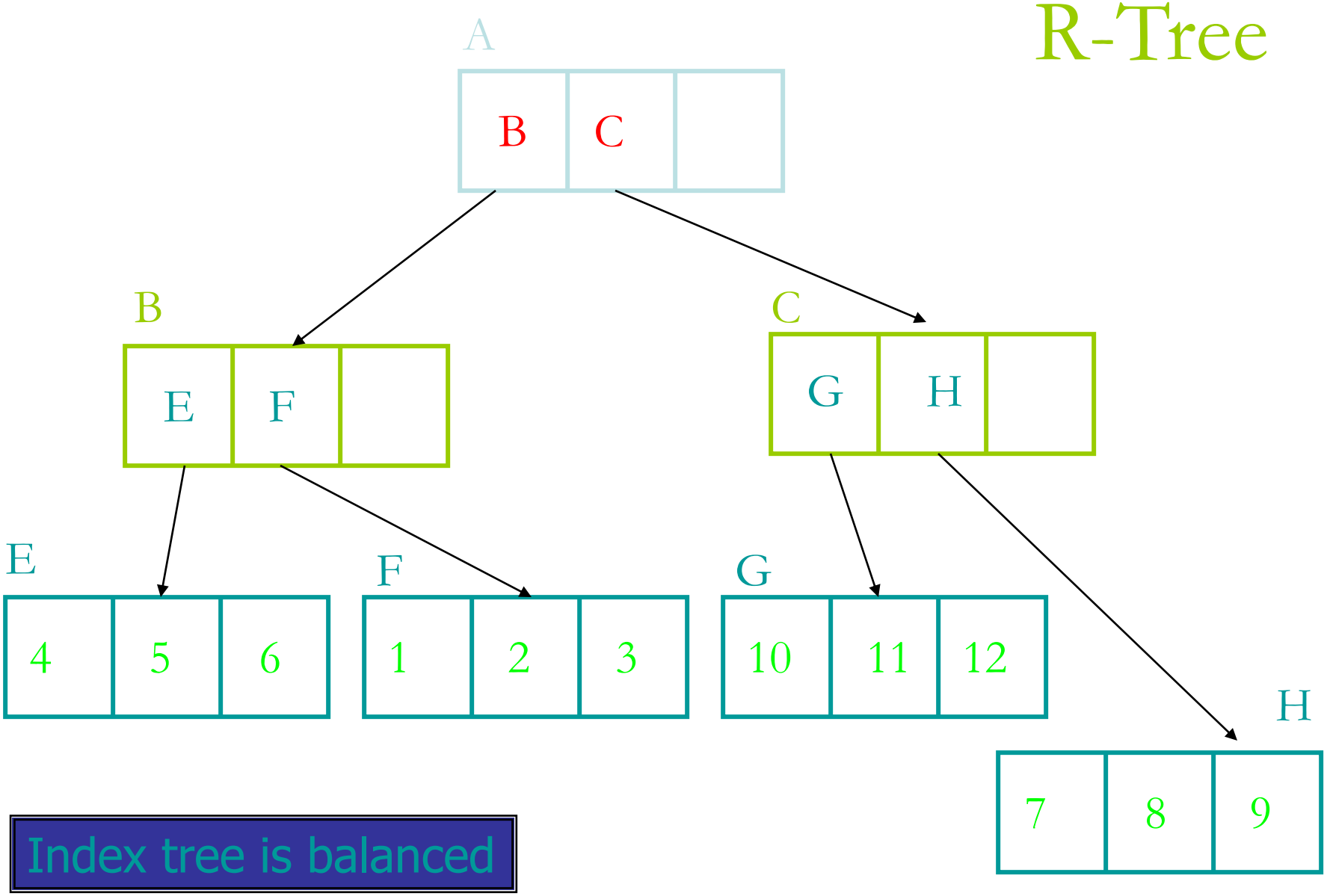
Προς εσωτερικούς κόμβους ή φύλλα του δέντρου με αντικείμενα που εσωκλείονται στο MBR R1

R-Tree



Ρίζα του δέντρου, αποθηκεύει τα MBRs των ενδιάμεσων κόμβων

R-Tree



Index tree is balanced

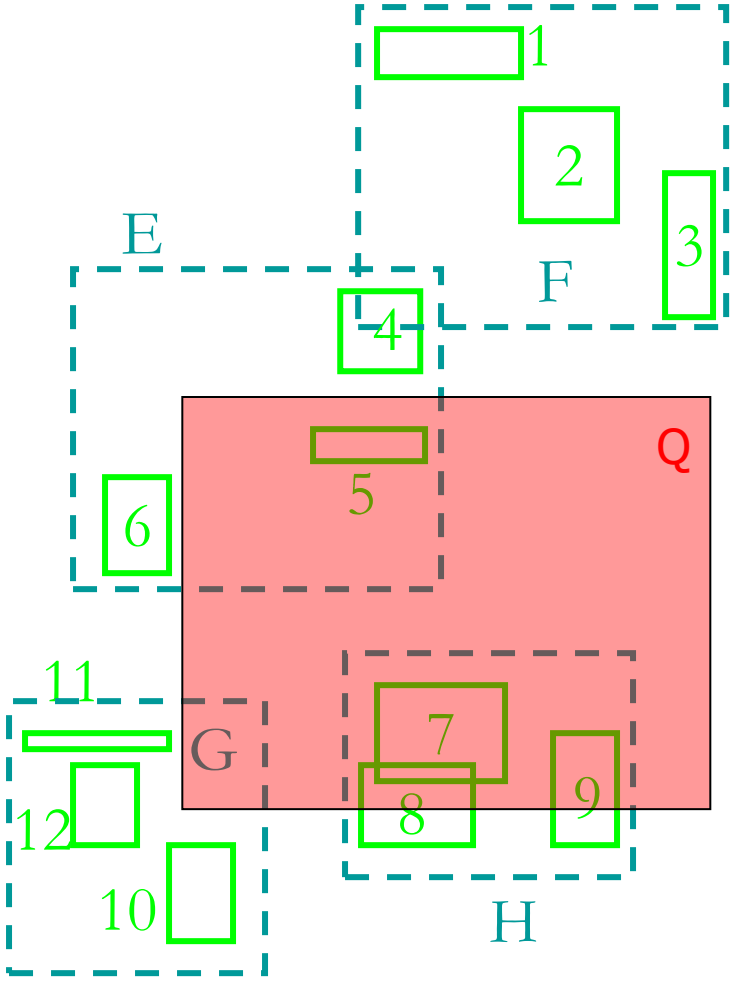
Σημαντική Παρατήρηση

Οι εσωτερικοί κόμβοι μπορούν να επικαλύπτονται!!!

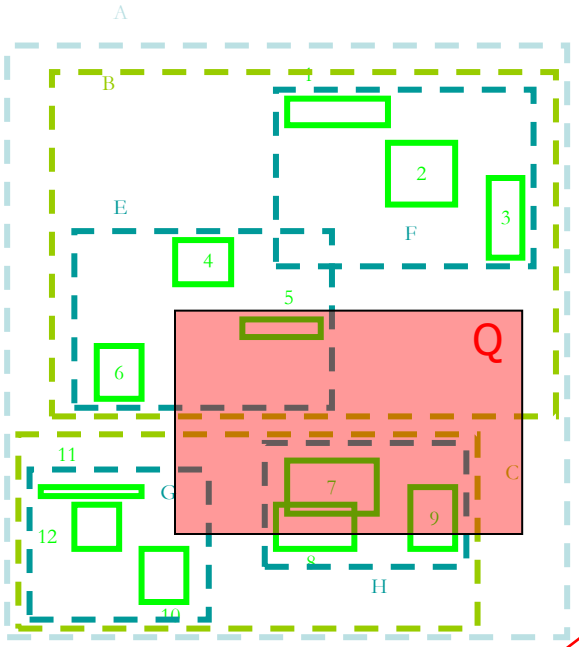
Διαφορά από όλα τα ευρετήρια
που είδαμε ως τώρα!

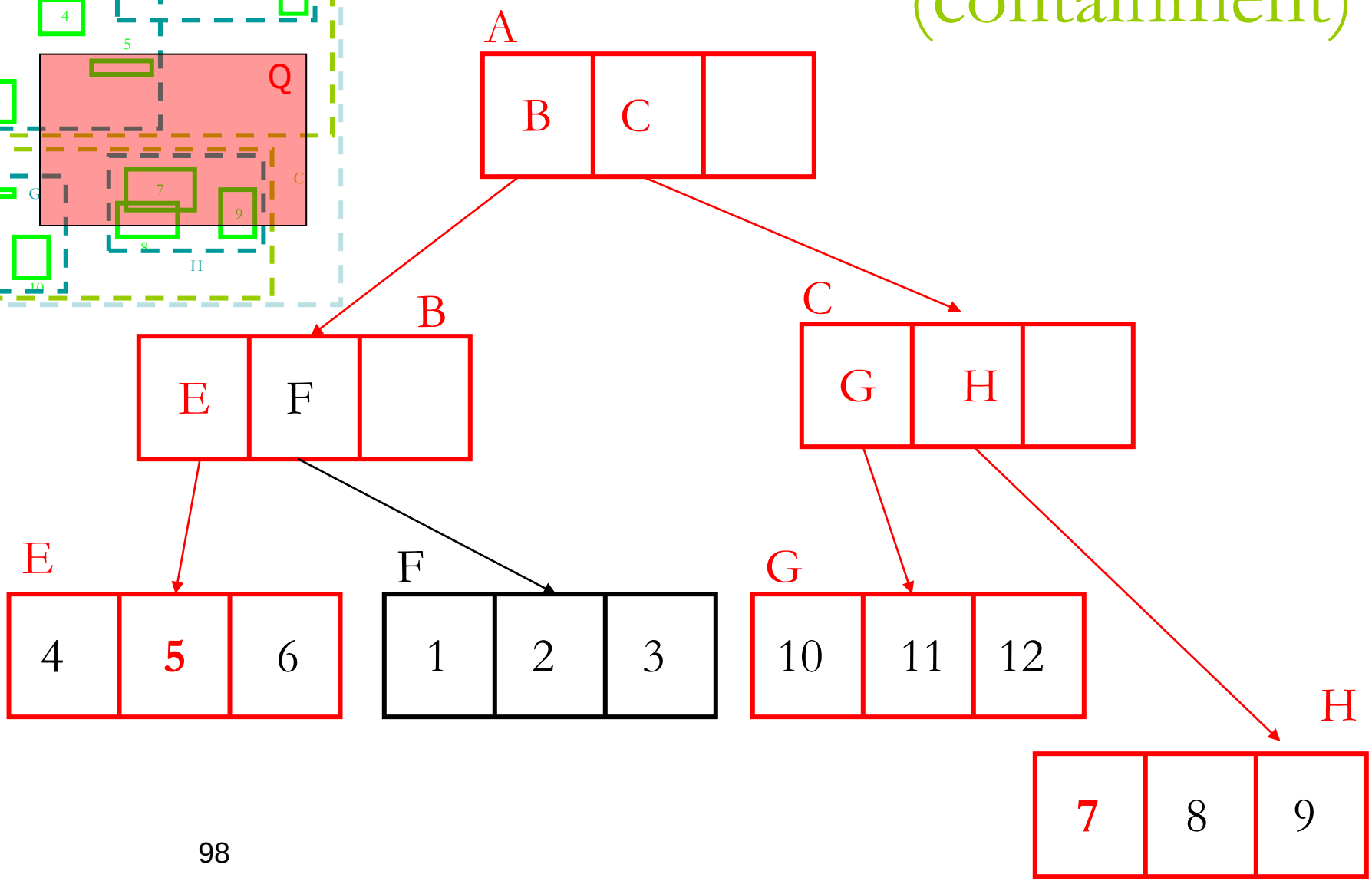
Intersection, containment Queries

- Ως απάντηση σε ένα χωρικό ερώτημα Q το R-tree ανακαλεί όλους τους κόμβους, το MBRs των οποίων έχει μη μηδενική επικάλυψη με το ερώτημα
- Στα φύλλα επιστρέφονται όλα τα δεδομένα που έχουν επικάλυψη (intersection query) ή εμπεριέχονται πλήρως στο ερώτημα (containment query)
- Στο παράδειγμα, το containment query Q ανακαλεί τα φύλλα E,G,H και μέσα από αυτά τα δεδομένα 5,7
  - Ένα intersection query θα επέστρεφε (επιπλέον των 5,7) και τα 8,9



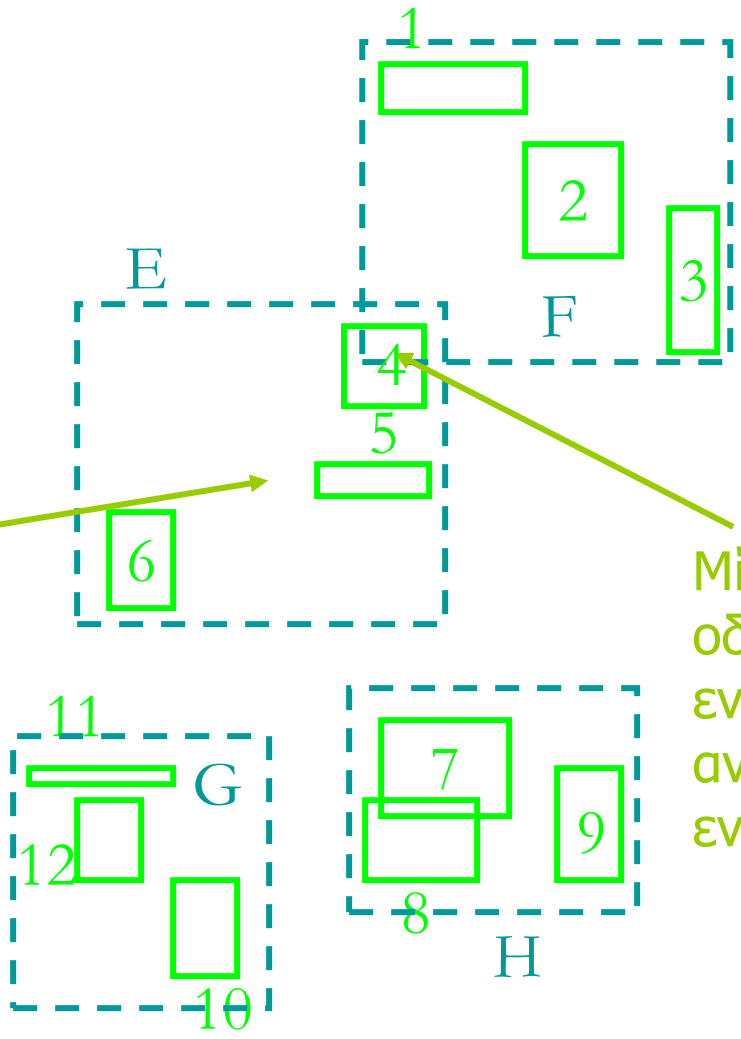
Αναζήτηση (containment)





Construction

- Δύο επιθυμητά χαρακτηριστικά των MBRs
  - Ελαχιστοποίηση επικάλυψης (overlap)
  - Ελαχιστοποίηση κενού χώρου (dead space)



Μία αναζήτηση σε αυτό το σημείο θα μας οδηγήσει στον κόμβο E ενώ δεν υπάρχει αντικείμενο στο συγκεκριμένο σημείο

Μία αναζήτηση εδώ θα μας οδηγήσει στους κόμβους E,F ενώ μόνο ο πρώτος περιέχει αντικείμενο (4) που μας ενδιαφέρει

Περισσότερες λεπτομέρειες

R-Trees: A Dynamic Index Structure for Spatial Searching; A. Guttman, SIGMOD '84