# Lecture 6:

# Multi Dimensional Data and Queries

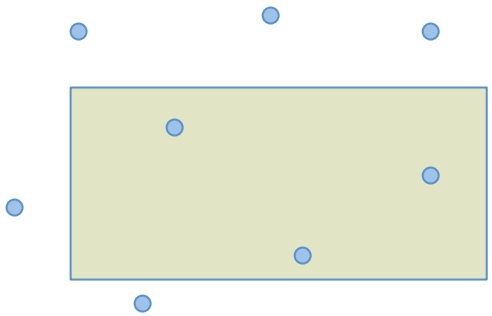
Rik Sarkar

# Data is multi-dimensional

- Locations of nodes have 2 dimensions
- A node can have different data elements
  - User id, files, file size, temperature, pressure, nearby nodes..
- Each data item adds new dimensions

# Multdimensional query

- What are the nodes inside a given rectangle?
- Similarly, what are the sensors at above 30C?
- What are the sensors above 30C and below 50% humidity?  etc



# Range query

- Given a ‘range’ or region in n-dimensional space find the data points in it
- Decision: Are there any data points in the range?
- Counting: How many data points are in the range?
- Reporting: Which data points are in the range?

# Range query in a network

- We need to index data
- To answer range queries quickly

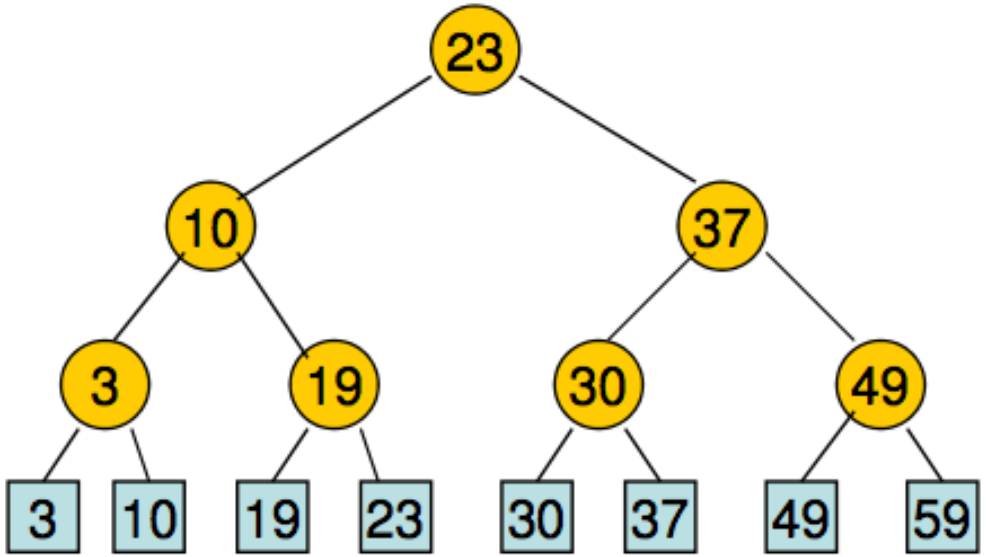
- Papers:
- X. Li, Y. J. Kim, R. Govindan, W. Hong, **Multi-dimensional Range Queries in Sensor Networks**, ACM SenSys 2003.
- J. Gao, L. Guibas, J. Hershberger, L. Zhang, **Fractional Cascaded Information in a Sensor Network**, IPSN'04.

# Theory

- Trees are good for data indexing
- For example: Binary search trees on sorted data
- We can extend trees to higher dimensional data

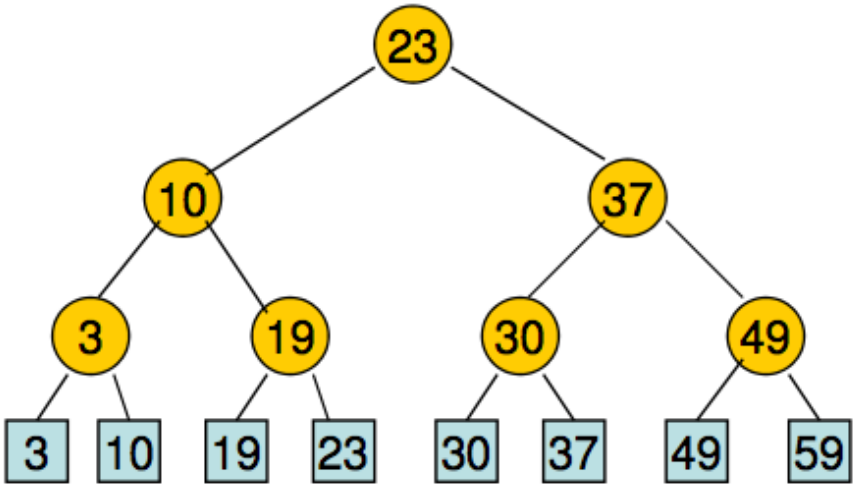
# 1D Search trees and range data

- Imagine data sorted along x-axis
- Find the data in interval $[x_1,x_2]$
- 1D range tree: a balanced binary search tree on a sorted array or list

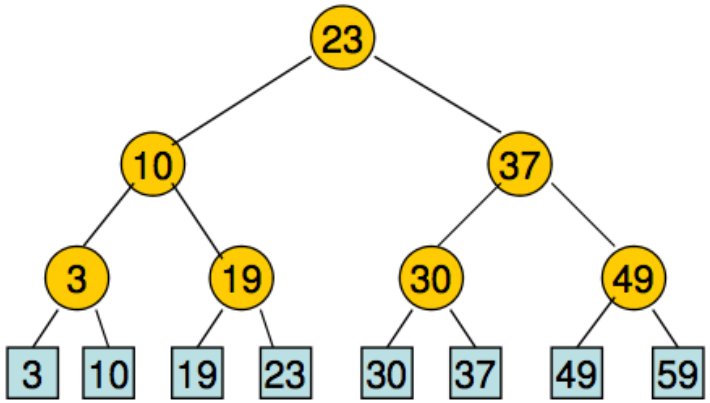


# Search trees and range data

- Find the data in the interval $[x_1,x_2]$
  - Find the location of $x_1$ in the array
  - Find the location of $x_2$ in the array
  - Report all values in between
  - Example: [12, 33]



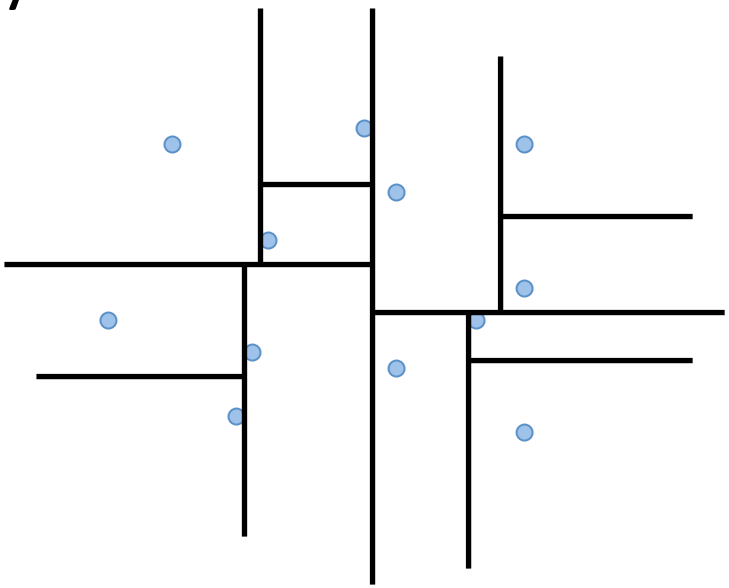
# Costs of Search tree and range data

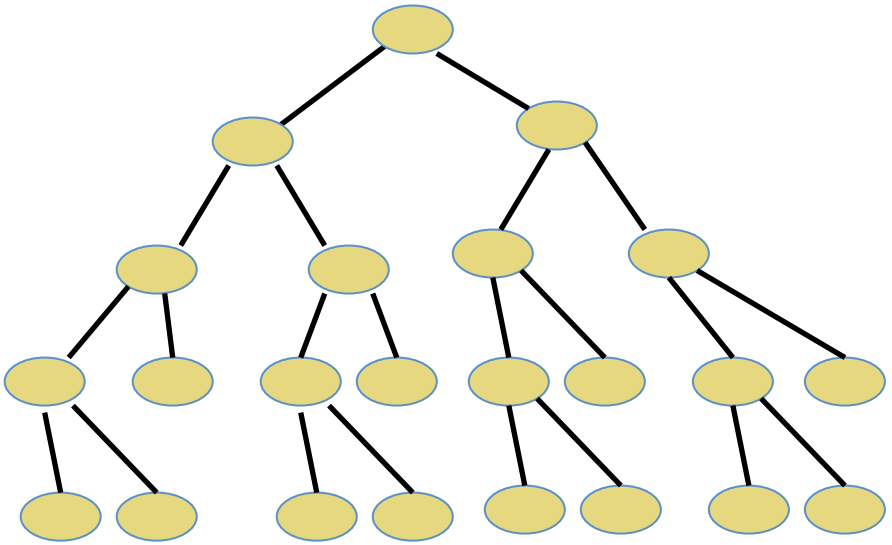



- Storage :
  - n leaf nodes
  - n/2 nodes at next level
  - n/4 nodes at level above…
  - Total : n + n/2 + n/4 + ….+1 = 2n
- Response time:
  - Find $x_1$ in time O(log n) = height of the tree
  - Find $x_2$ in time O(log n) = height of the tree
  - Report k elements in $[x_1,x_2]$
  - Total time : O(log n + k): output sensitive

# K-D Tree

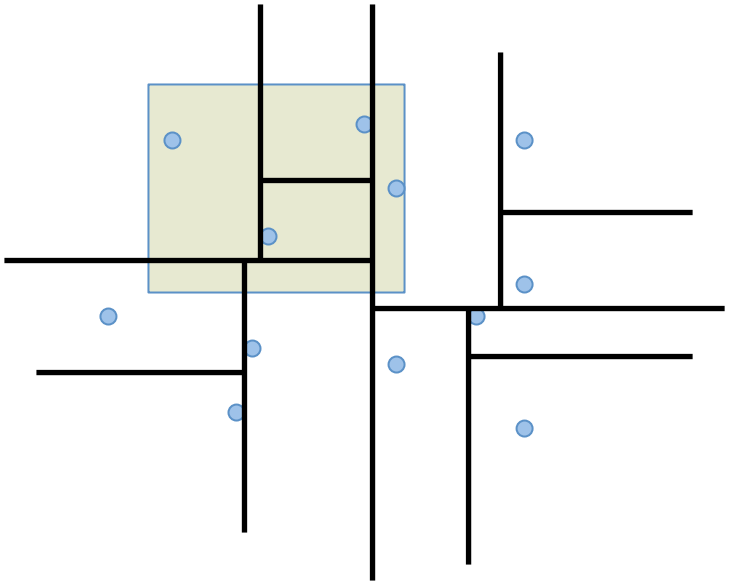
- Recursive space partitioning
- Alternately split along x and y axis
- Each node in tree represents a region split along x or y axis

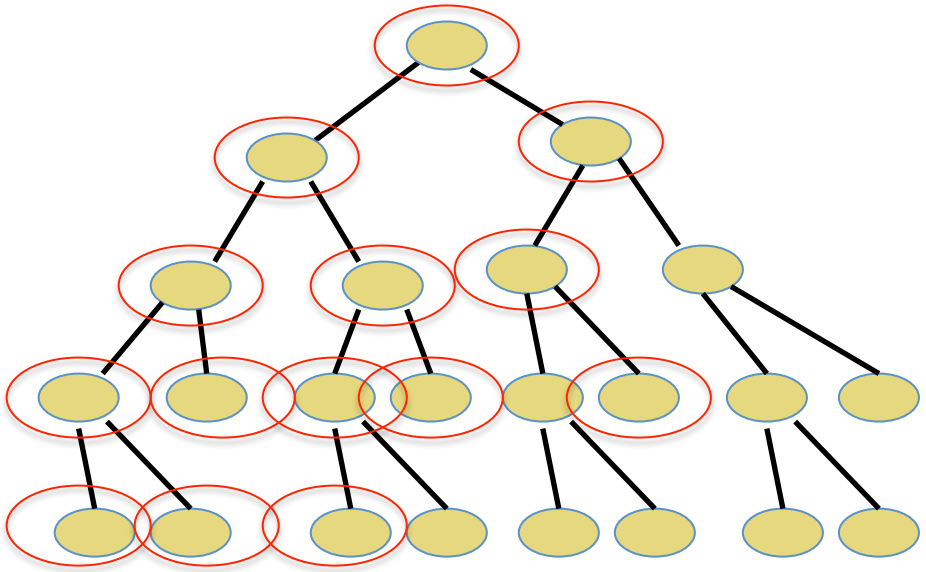




# K-D Tree

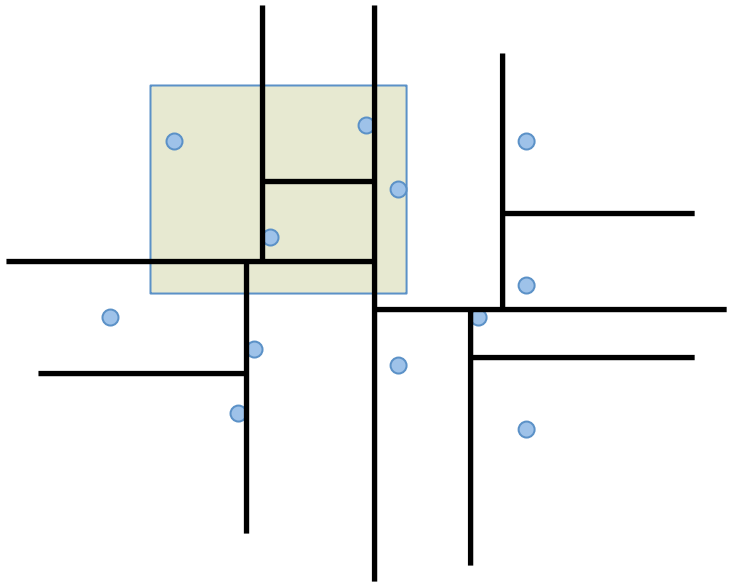
- 2D query R = $[x_1,x_2]X[y_1,y_2]$
- Find the regions that intersect R





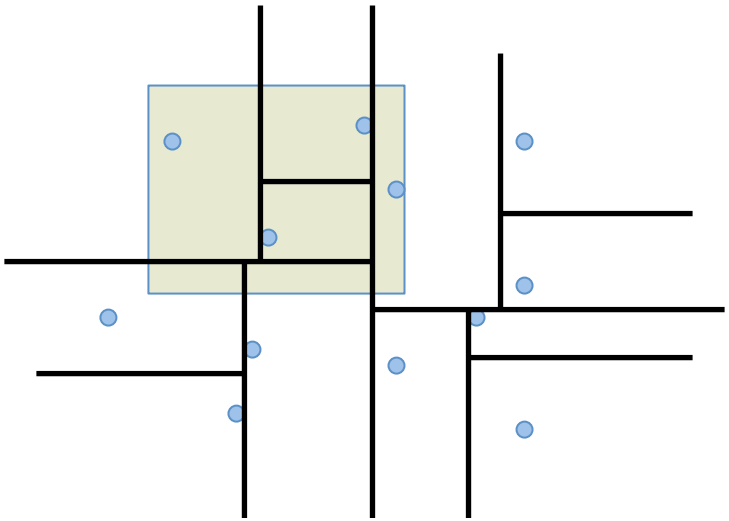
# K-D Tree

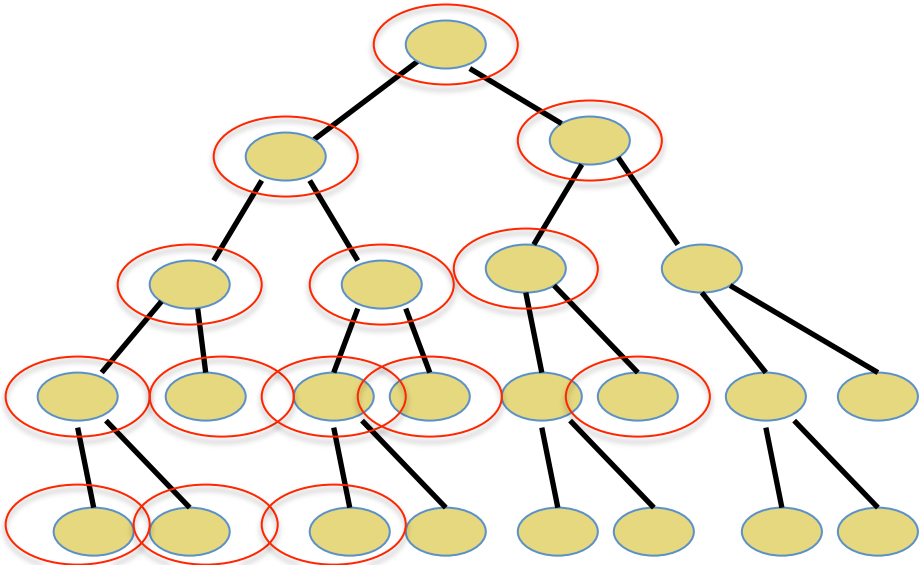
- Storage : O(n) nodes
- Height : O(log n)
- Query cost : $O(n^{1/2} + k)$: k is output size



# K-D Tree query cost

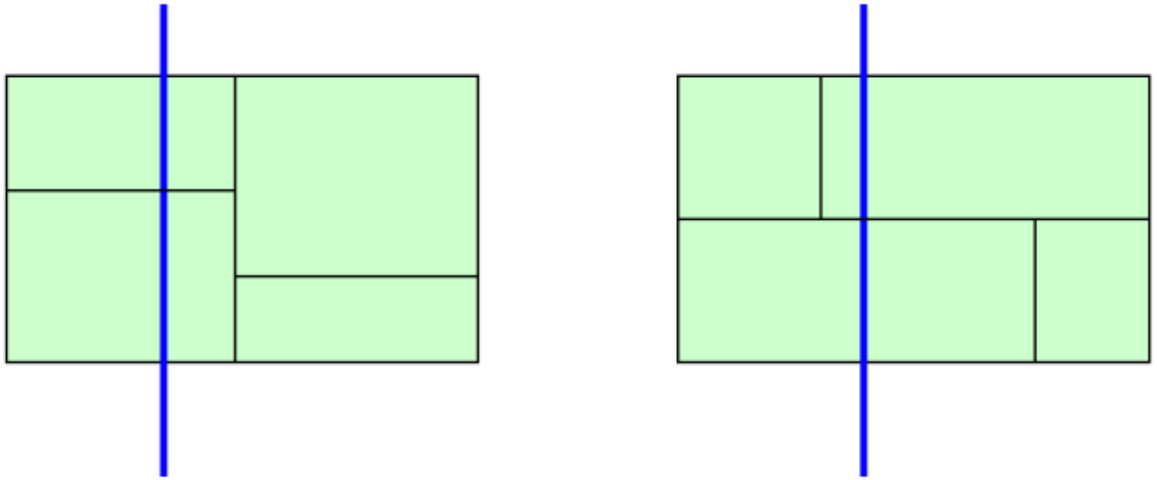
- $O(n^{1/2} + k)$
- Say r(v) is the region covered by node v in KD tree
- We visit 2 types of nodes in the tree
  - r(v) may be completely inside R
    - Then we simply take the nodes inside r(v)
    - This cost is accounted by k
  - r(v) may not be completely inside R
    - We may be visiting v, without counting the vertex inside r (v) : is not accounted in k





# Cost of K-D tree query

- How many nodes Q(n) can we visit, without counting anything in them?
- Suppose the boundary of R intersects r(v)
- Look at the 4 grandchildren of r(v).
- The boundary can intersect at most 2 of them
- Therefore $Q(n) = 2Q(n/4)+O(1)=O(n^{1/2})$
- Total query cost is $O(k) + 4Q(n)=O(n^{1/2} + k)$

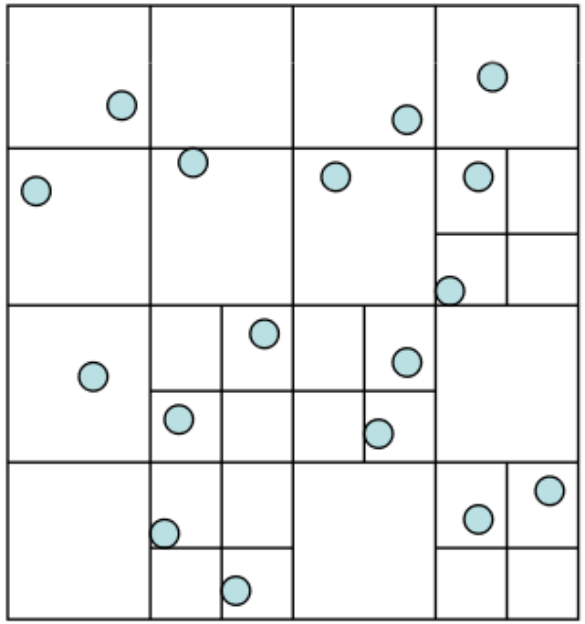


# Higher dimensional K-D Trees

- If dimension is d
- Storage O(n)
- Query cost $O(n^{1-1/d} + k)$

# Quad Trees

- Recursive space partition
- Without trying to balance number of nodes
- May need Ω(n) depth
- For uniform point distribution :
  depth is O(log n)

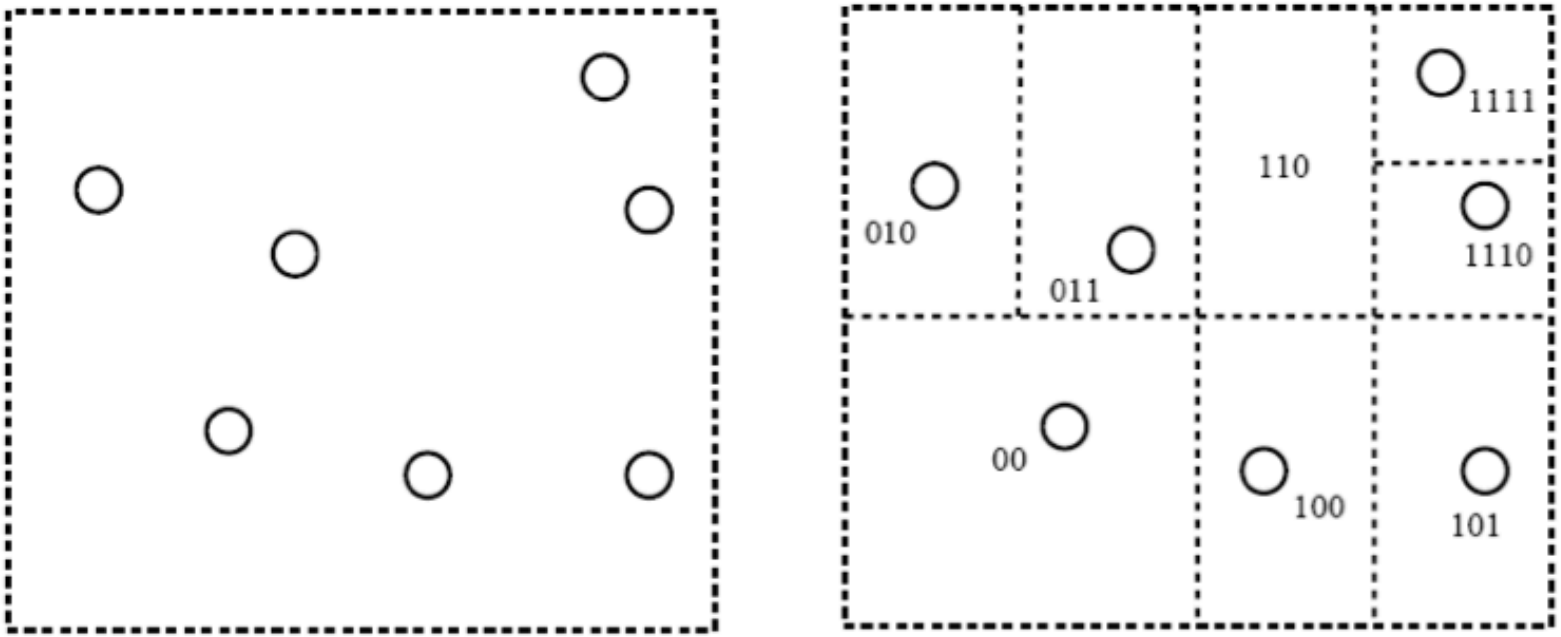


# Distributed Storage in Sensor Networks

- Where should we store data?
- How should we search for the data?

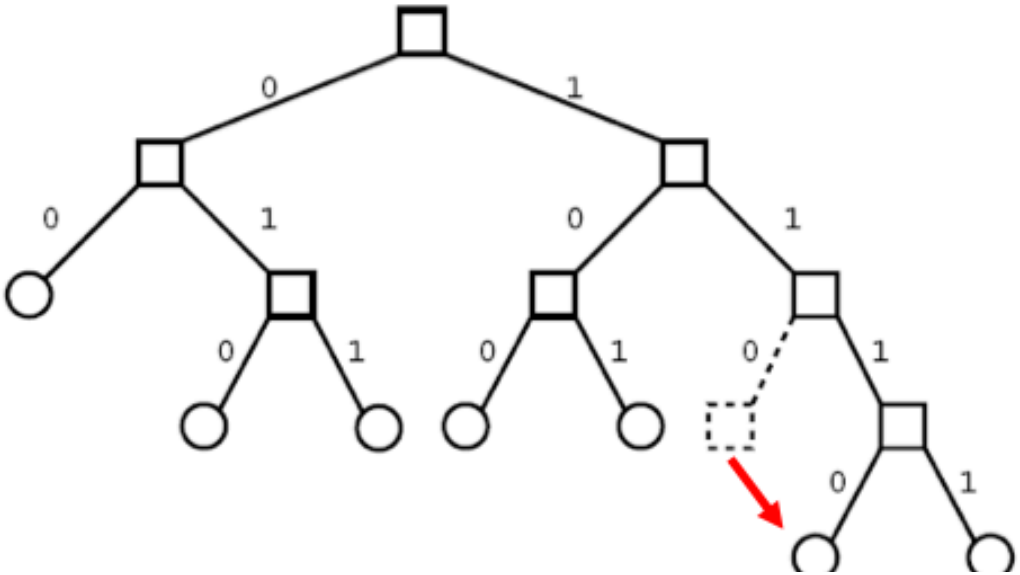
# DIM: Use K-D Trees

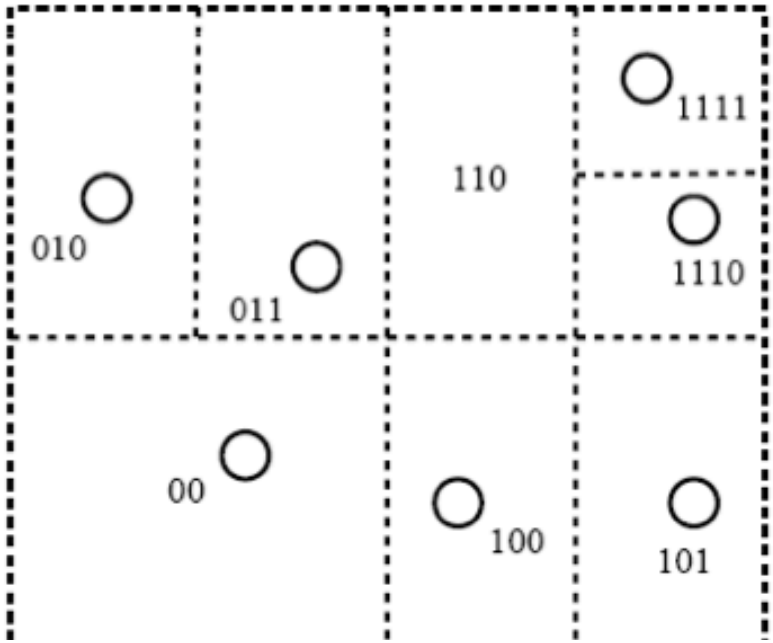
- Split network into equal zones alternately along x and y axes (we do not knonw locations of all nodes)
- Each cell has a zone code: left(bottom) = 0; right(top) = 1



# Zone tree

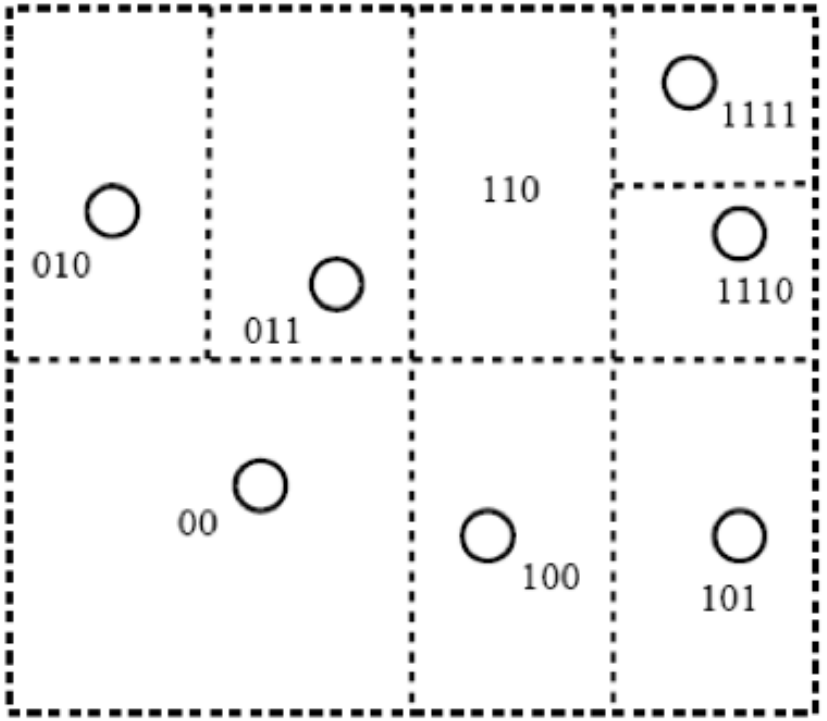
- Each node owns the largest zone that contains it and no other node
- If a zone is empty, it is owned by the leftmost node in the right subtree



# Data dependent hashing

- Hash a multi-dimensional event to a zone
- Event $\{A_i\}$, i=1,…,m, $0 \le A_i \le 1$
- Suppose zone codes have k bits
- For i=1 to m, if $A_i < 0.5$ i-th bit is 0, otherwise 1
- For i =m+1 to 2m, if $A_i < 0.25$ or $0.5 < A_i < 0.75$ i-th bit is 0 otherwise it is 1
- For example: [0.9, 0.6] is stored at 1110



# Data routing

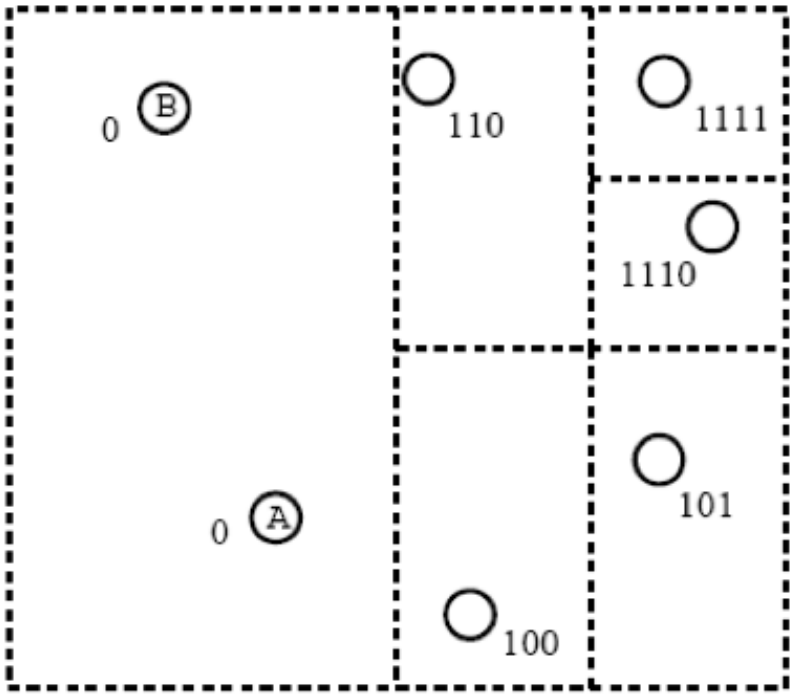
- The code for event E tells us the location where it should be stored
- Node A generates even and sends it by GPSR (greedy+face routing)
- Message arrives at node that owns the zone for E

# Query routing

- Searching a point event is same as routing an event
- A range query is routed to a larger zone that contains it, and successively refined

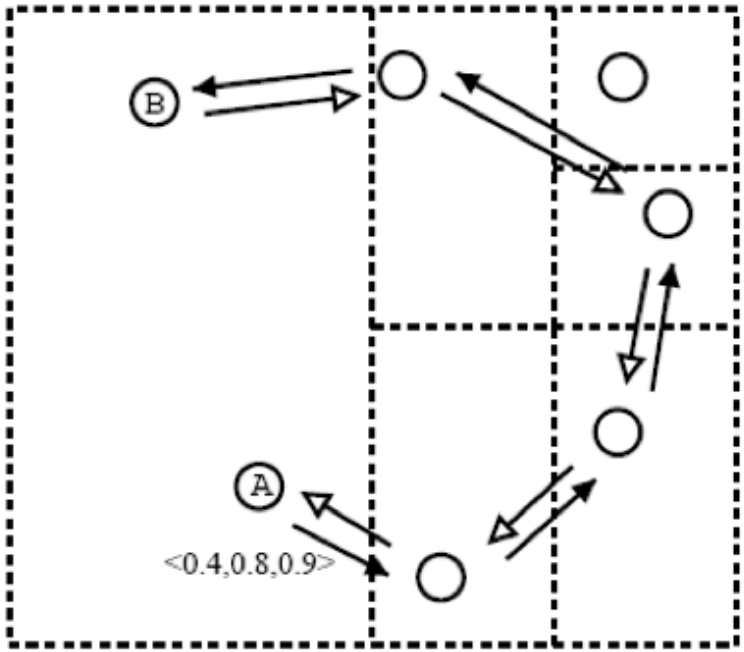
# Finding zones with event routing

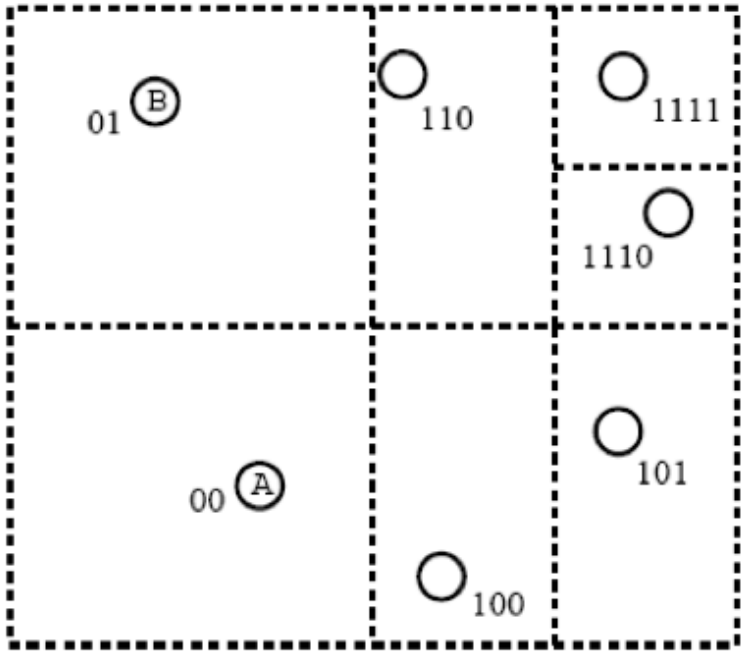
- How does  a node know its own zone?
- Assume everyone knows the outer boundary box
- A node checks its 1-hop neighbors and decides a largest zone that contains only itself
- This may be incorrect



# Zones could be corrected by event routing

- Suppose A is not sure if E is in its zone
- A sends E to a location near itself by GPSR
- This traverses the face
- B receives the event and notifies A that the zone should be split



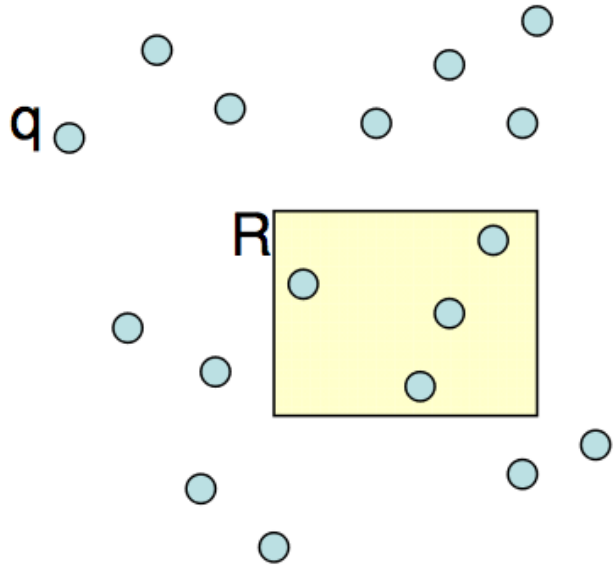



# Summary of DIM

- Similar data stored nearby: helpful for range query
- Storage may be far from where events are generated
- Each event costs about $O(n^{1/2})$ communications to store
- If data is not uniform, storage can be unbalanced

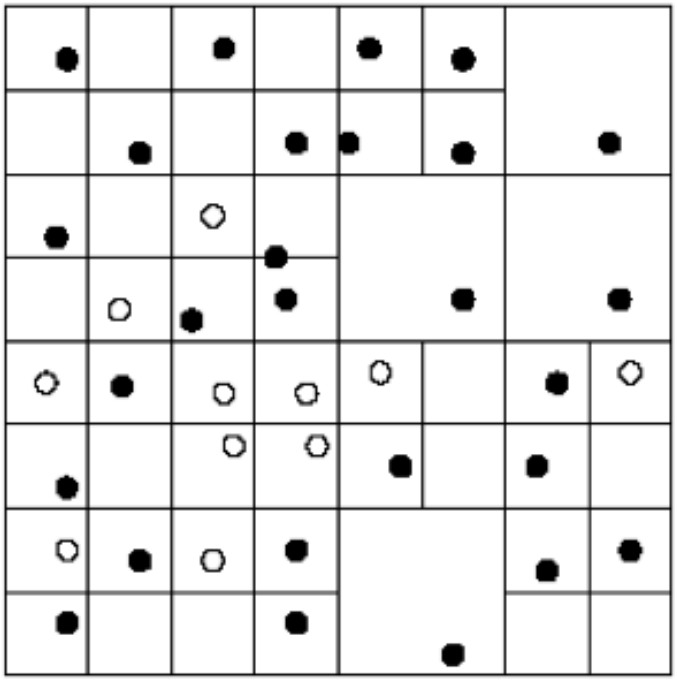
# Fractional cascading

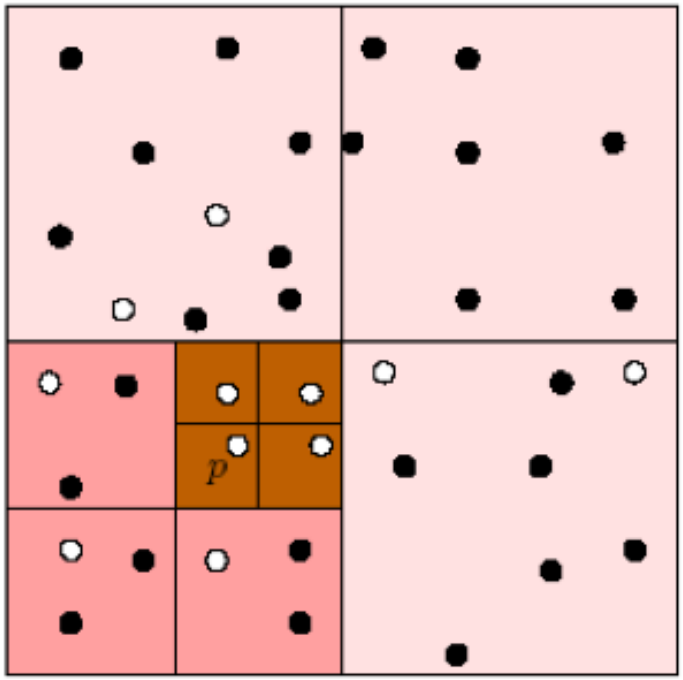
- Geographic query (q, R, T)
- Node q asks for all nodes in region R with temperature above T



# Storage

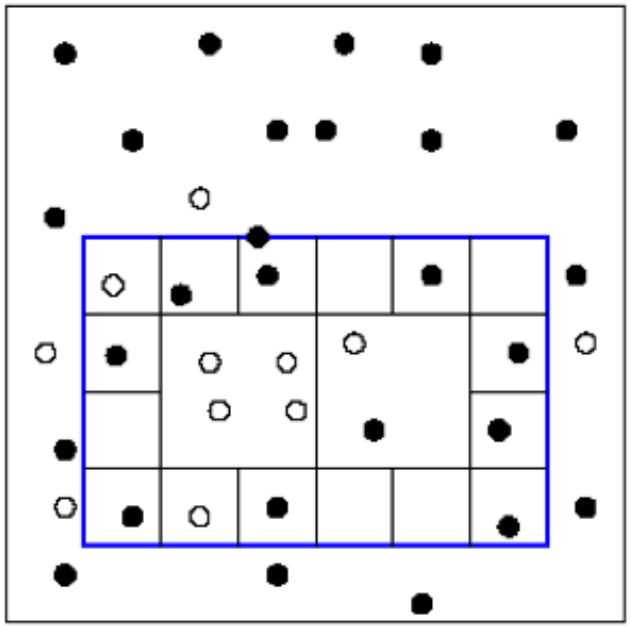
- Aggregated (max) value of a quad node is stored in all sensors in the parent subtree
- Each node stores O(log n) data
- Construction: O(log n)

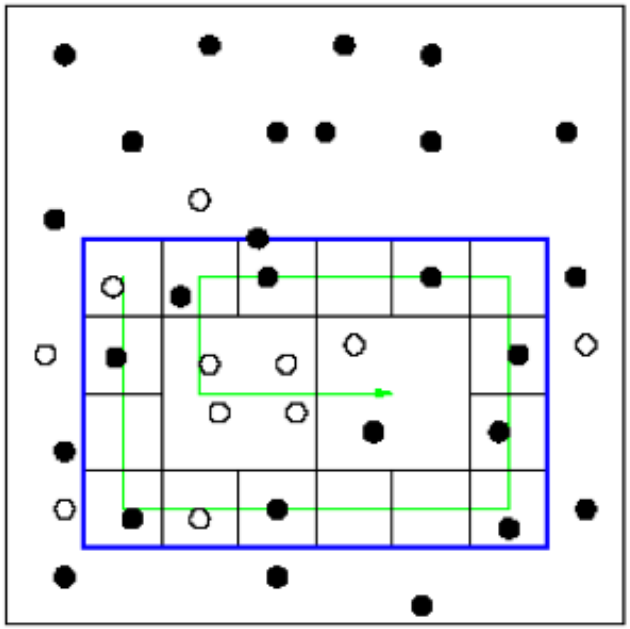




# Query

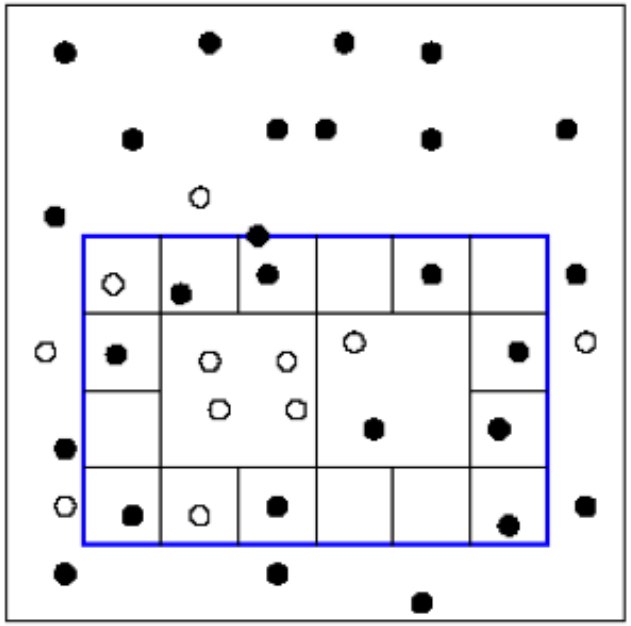
- Query region R is partitioned into canonical regions : maximal quads completely inside R
- Walk spirally to visit all quads

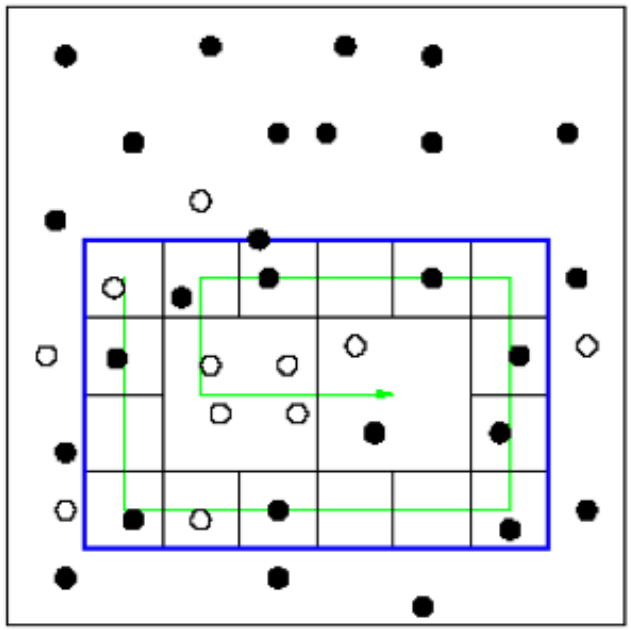




# Query cost

- Query cost for (q, R, T) is $O(D + \sqrt{Ak} + P \log P)$
- A is area, P is perimeter of R, D is distance(q,R)
- k is output size
- PlogP is cost of spiral visit





# Query cost

- Cost of recursively evaluating a canonical region with length L(u) and output k(u) is $O(L(u)\sqrt{k(u)})$

Total cost

$$\sum_{u \in C} L(u)\sqrt{k(u)} \leq \sqrt{\sum_{u \in C} L(u)^2 \sum_{u \in C} (\sqrt{k(u)})^2}$$

$$\leq \sqrt{Ak}$$

# Summary

- DIM: Store similar data close
  - Focus on data value
  - May need to travel far to store data
- Fractional cascading
  - Store data near source: Focus on location
  - Low cost to store
  - Range search is harder

# Paper Presentation 6/12

- A. Savvides, C.-C. Han, and M. B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. MobiCom 01.

- Yi Shang, Wheeler Ruml, Ying Zhang, Markus P.J. Fromherz. Localization from Mere Connectivity, MobiHoc 03.

- D. Moore, J. Leonard, D. Rus, S. Teller. Robust distributed network localization with noisy range measurements, SenSys 2004.