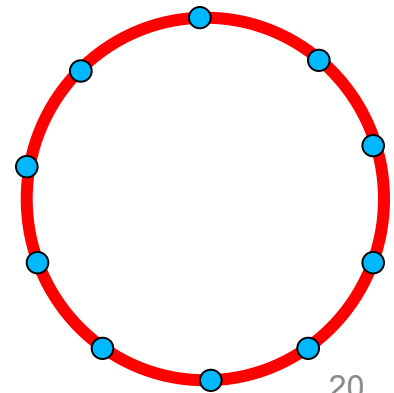


# GLS: Grid Location Service

- Last lecture: location update/query for a mobile user.
- All nodes are possibly mobile.
- Need to support queries for all nodes.
- Objective: balance the load, scalability.

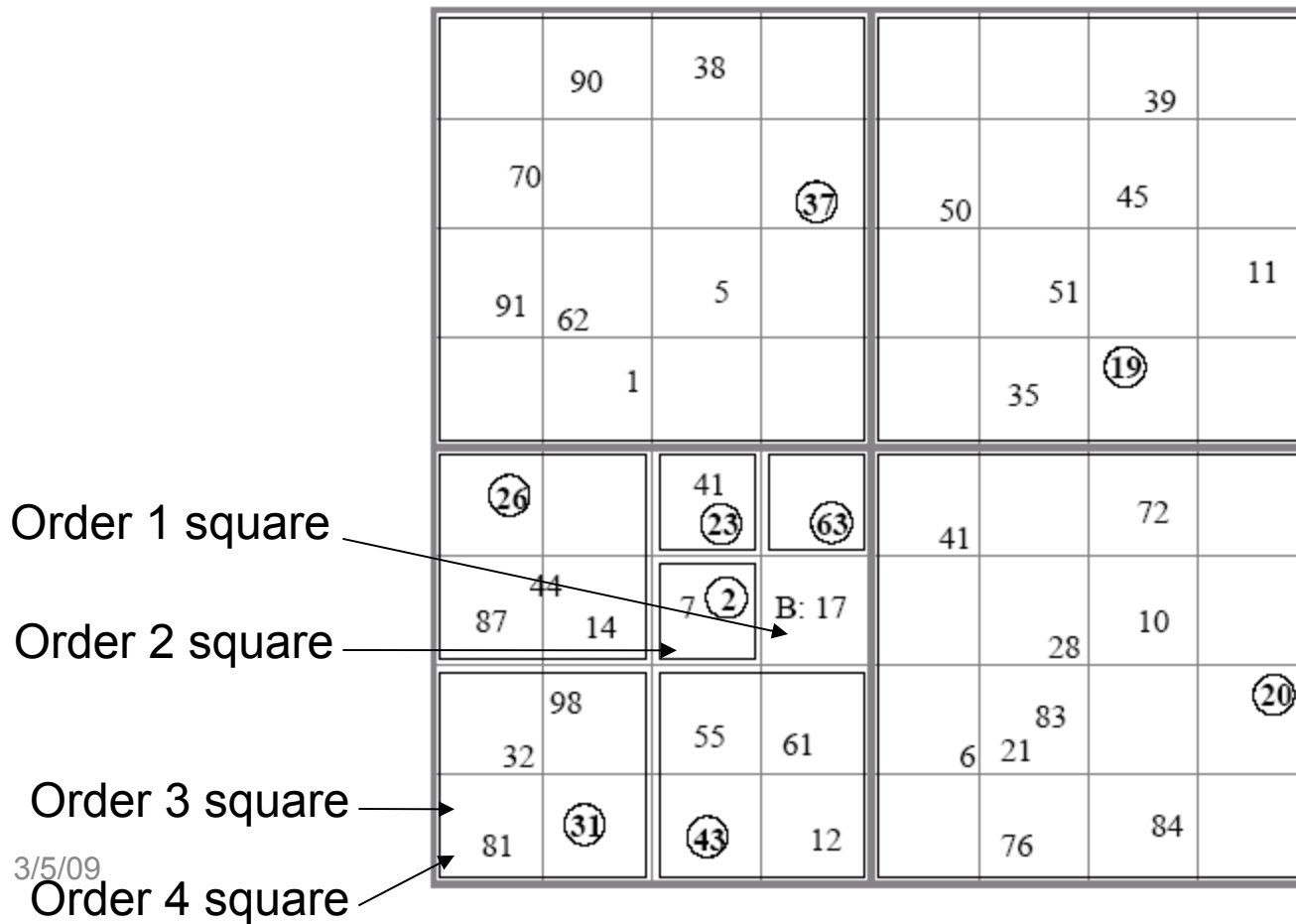
# GLS: Grid Location Service

- Each node is assigned a random ID in a circular space.
- Each node stores/updates its location information at a set of location servers, more at nearby regions, fewer at far away regions.
- Location query uses **nothing beyond the ID**.
- Two operations, FIND, SEARCH

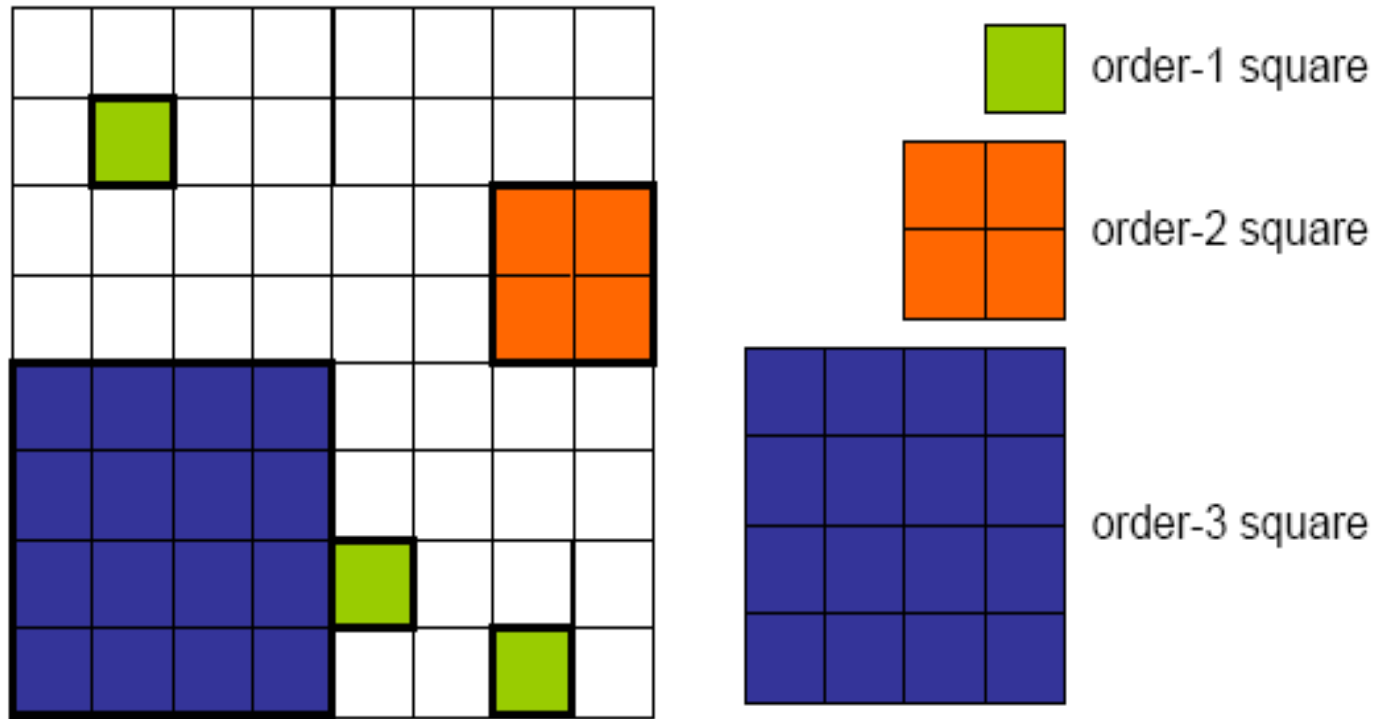


# Recursive partitioning

- Quad-tree partition: each node is inside a unique square on each level.



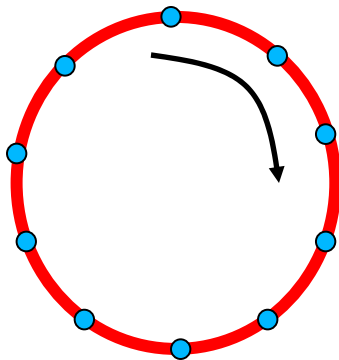
# Partitioning the world



Invariant: a node is located in exactly one square of each size (no overlapping)  
An order- $x$  square contains always 4 order- $(x-1)$  squares

# Location servers

- Node B's location servers: Inside each sibling square on each level, choose B's closest node.
- Node **closest** to B in ID space: node with **least ID greater than B**
- Circular ID space: 2 is closer to 17 than 7 is.



	90	38							
70			37		50		45		
91	62		5			51			11
		1				35	19		
26			41	23	63		41		72
87	44		14	7	2	B: 17		28	10
	98			55	61			83	20
	32					6	21		
81	31		43		12		76		84

# Location queries

- A queries the location of B:
- A's only information about B is the ID of B.
- A does not know who are B's location servers.
- B even doesn't know its location servers.
- How to implement location query?

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82	
	A: 90	38				39	
1,5,16,37,62 63,90,91			16,17 19,21 23,26,28,31	19,35,39,45 51,82		39,41,43	
70			37	50		45	
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50		19,35,39,45 50,51,55,61 62,63,70,72 76,81
91	62	5			51		11
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98	19
	1				35		
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70	72
26		23	63	41			
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84	
87	14	2	B: 17		28	10	
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,21,76	6,10,12,14 16,17,19,84	20
32	98	55	61		6		
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55 61	2,5,6,10,43 55,61,63,81 87,98		6,21,28,41 72	20,21,28,41 72,76,81,82	
81	31	43	12		A: 76	84	

# Location queries

- A queries location of B:
- A stores location information for some other nodes.
- A send the request to the one that is **closest** to B, among those about which A has location information.
- Continue until hit one of B's location servers.
- This works! Why?

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82
	A: 90	38				39
1,5,16,37,62 63,90,91			16,17 19,21 23,26,28,31	19,35,39,45 51,82		39,41,43
70			37	50		45
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50	19,35,39,45 50,51,55,61 62,63,70,72 76,81 11
91	62	5			51	
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98 19
	1				35	
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70 72
	26	23	63	41		
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84
87	14	2	B: 17		28	10
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,21,76	6,10,12,14 16,17,19,84
	32	98	55	61	6	20
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55 61	2,5,6,10,43 55,61,63,81 87,98		6,21,28,41 72	20,21,28,41 72,76,81,82
81	31	43	12		A: 76	84

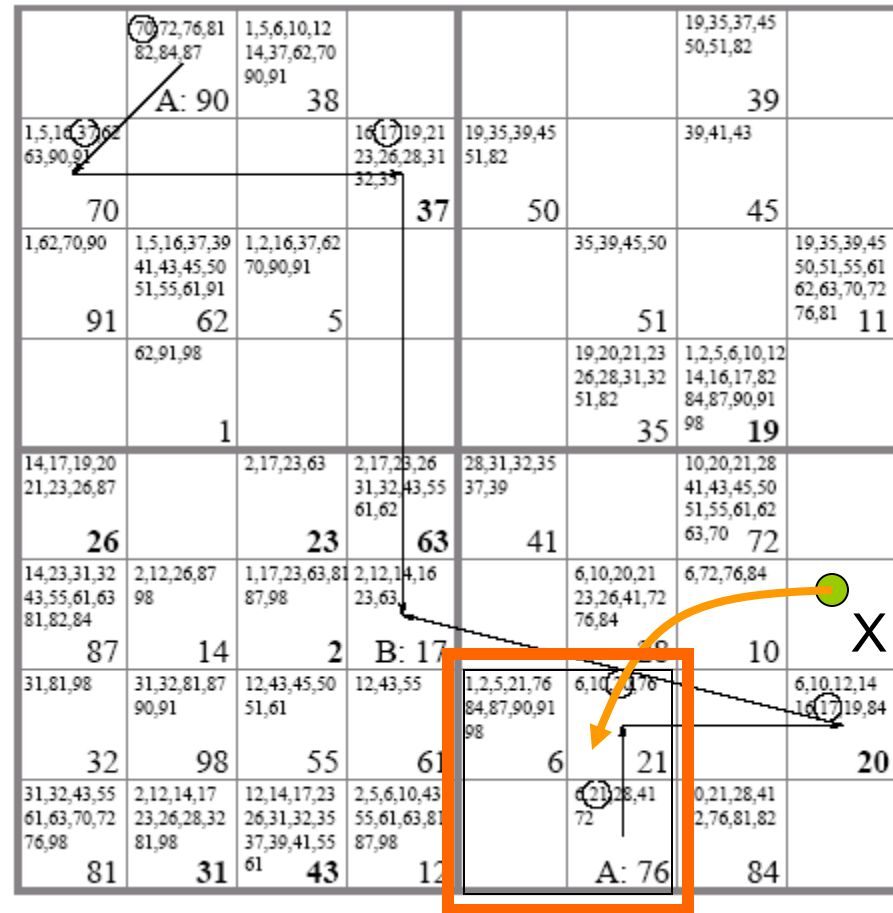
# Location queries

- Claim: the query visits the node closest to B in A's order-i square.
- The query always goes to B's closest node, as the covering scope increases.
- The correctness of the alg: when A's order-i square contains B, the closest node is B itself.
- Proof by induction. It's obvious for order-1 square.

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82
	A: 90	38				39
1,5,16,37,62 63,90,91			16,17 19,21 23,26,28,31	19,35,39,45 51,82		39,41,43
70			37	50		45
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50	19,35,39,45 50,51,55,61 62,63,70,72 76,81 11
91	62	5			51	
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98 19
	1				35	
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70 72
26		23	63	41		
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84
87	14	2	B: 17		28	10
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,21,76	6,10,12,14 16,17,19,84
32	98	55	61		6	20
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55 61 43	2,5,6,10,43 55,61,63,81 87,98		6,21,28,41 72	A: 76 20,21,28,41 72,76,81,82
81	31	61	12		A: 76	84

# Location queries

- Assume 21 is B's closest node in A's order-2 square  
 → no node is between 17 and 21 in order-1 square.
- Suppose a node X in A's order-2 sibling square is between 17 and 21. By the replication rule, X picks 21 as its location server.
- 21 stores the location of **all** the nodes between 17 and 21 in sibling order-2 square, obviously the one closest to 17.



# Inform/update location servers

- A can update its location server inside a square S without knowing its identify.
- A routes to a square with geographical routing.
- The first node in the square S performs a location query of A.
- The query ends up at a node closest to A, who is A's location server!



Hidden assumption: the nodes in S have distributed their locations inside S!

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91			19,35,37,45 50,51,82	
	A: 90	38			39	
1,5,16,37,62 63,90,91			16,17,19,21 23,26,28,31	19,35,39,45 51,82	39,41,43	
70			37	50	45	
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50	19,35,39,45 50,51,55,61 62,63,70,72 76,81
91	62	5			51	11
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98
	1				35	19
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70
26		23	63	41		72
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84
87	14	2	B: 17		28	10
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,20,76	6,10,12,14 16,17,19,84
32	98	55	61		6	21
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55	2,5,6,10,43 55,61,63,81 87,98		6,10,20,41 72	20,21,28,41 72,76,81,82
81	31	61	43	12	A: 76	84

# The bootstrapping

- When the entire system is turned on, order-1 squares exchange their information with local protocol, then nodes recruit their order-2 location servers and so on.
- No flooding needed. The location service is constructed by geographical unicast routing only.

	70 72,76,81 82,84,87	1,5,6,10,12 14,37,62,70 90,91				19,35,37,45 50,51,82	
	A: 90	38				39	
1,5,16,37,62 63,90,91			16,17 19,21 23,26,28,31	19,35,39,45 51,82		39,41,43	
70			37	50		45	
1,62,70,90	1,5,16,37,39 41,43,45,50 51,55,61,91	1,2,16,37,62 70,90,91			35,39,45,50		19,35,39,45 50,51,55,61 62,63,70,72 76,81
91	62	5			51		11
	62,91,98				19,20,21,23 26,28,31,32 51,82	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98	19
	1				35		
14,17,19,20 21,23,26,87		2,17,23,63	2,17,23,26 31,32,43,55 61,62	28,31,32,35 37,39		10,20,21,28 41,43,45,50 51,55,61,62 63,70	72
26		23	63	41		72	
14,23,31,32 43,55,61,63 81,82,84	2,12,26,87 98	1,17,23,63,81 87,98	2,12,14,16 23,63		6,10,20,21 23,26,41,72 76,84	6,72,76,84	
87	14	2	B: 17		28	10	
31,81,98	31,32,81,87 90,91	12,43,45,50 51,61	12,43,55	1,2,5,21,76 84,87,90,91 98	6,10,20,76		6,10,12,14 16,17,19,84
32	98	55	61		6	21	20
31,32,43,55 61,63,70,72 76,98	2,12,14,17 23,26,28,32 81,98	12,14,17,23 26,31,32,35 37,39,41,55	2,5,6,10,43 55,61,63,81 87,98		6,10,20,41 72	20,21,28,41 72,76,81,82	
81	31	61	43	12	A: 76	84	

# Take a rest and enjoy the beauty of this algorithm

- It solves location service problem by using geographical routing.
- More locality sensitive: a node acquires the location from a nearby server.
- Load balancing: location servers are spatially distributed.
- Simple rule, simple construction and maintenance.
- Worst-case query behavior is not bounded, however.  
☹

# Handle mobility

- For the geographic forwarding protocol:
  - Refresh neighbor information using Hello messages, timeouts.
- Updating location servers when a node moves:
  - Not every time—that would be too much communication.
  - Rather, they use the Awerbuch-Peleg idea: Update order- $i$  servers when you've moved  $2^{i-1}$  distance.
  - Thus, updates are sent more frequently to local servers than more distant ones.

# Handle mobility

- Invalidating location table entries:
  - They have a timeout value.
  - Even when a node doesn't move, it thus has to periodically send its location to its servers, in order to refresh the location info.
- Caching:
  - Nodes can cache locations they hear about, to use in sending data via geographic routing—but these entries aren't put into the same table used by FIND—recall that it's important that certain entries NOT be in these tables.

# Handle mobility

- Two types of failures
  - A node receives a query but does not know the location of any node with an ID closer.  
Location update is not soon enough.
  - A location server forwards a packet to the next node's square but the node is not there.  
When a node moves to a nearby square, it leaves a forwarding pointer.
  - Can be improved.

# Follow-up work

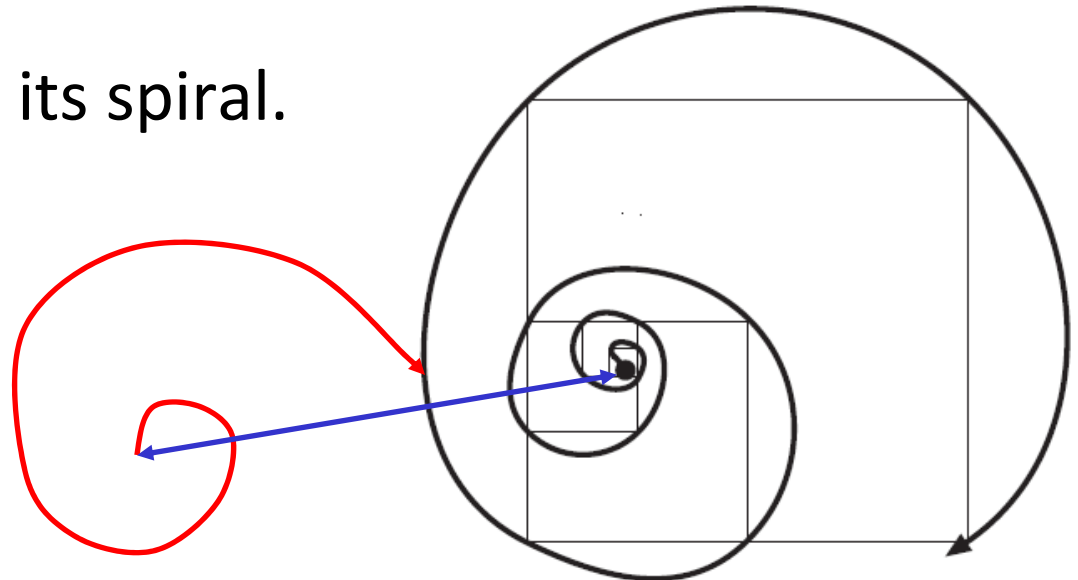
- I. Abraham, D. Dolev, D. Malkhi , LLS : a Locality Aware Location Service for Mobile Ad Hoc Networks, DIALM-POMC 2004.

# Main idea of LLS

- Handle node mobility
- Worst-case guarantee of FIND & MOVE
- Recall for Grid:
  - Cost for MOVE can be high, if a node crosses quad-tree boundaries.
  - Cost for FIND can be high, if a node is just at the neighboring square separated far in the hierarchy.

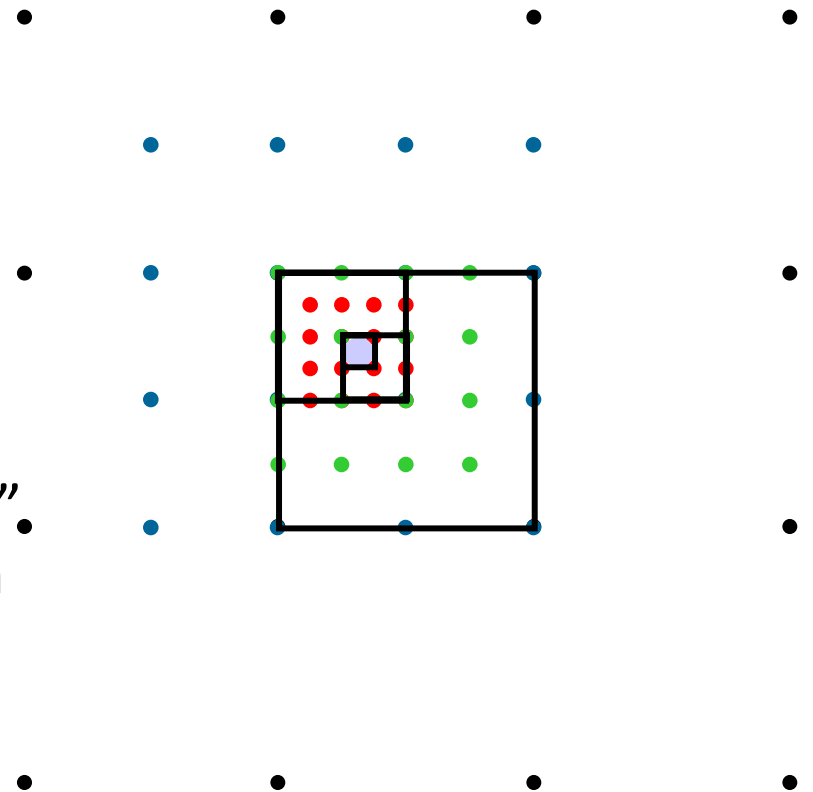
# LLS

- Each node defines a square partitioning **centered at itself**.
- Spiral: store the location at the corners of the squares.
- MOVE is heavy --- erase the old spiral and draw the new spiral
- FIND is efficient: use its spiral.
- $\text{Cost}(\text{FIND}) = O(d)$ .



# LLS --- improve MOVE behavior

- Store location in the 8 squares surrounding each square.
  - Some flexibility to allow “lazy” update.
  - When a node moves inside the 9 squares, nothing happens.
  - When a node moves outside the 9 squares, it has moved “sufficiently” far away from its previous location and can drag the 9 squares with it.
  - Amortized update cost is low.



# LLS performance

- MOVE:  $O(d \log d)$
- FIND:  $O(d)$
- Amortized cost.

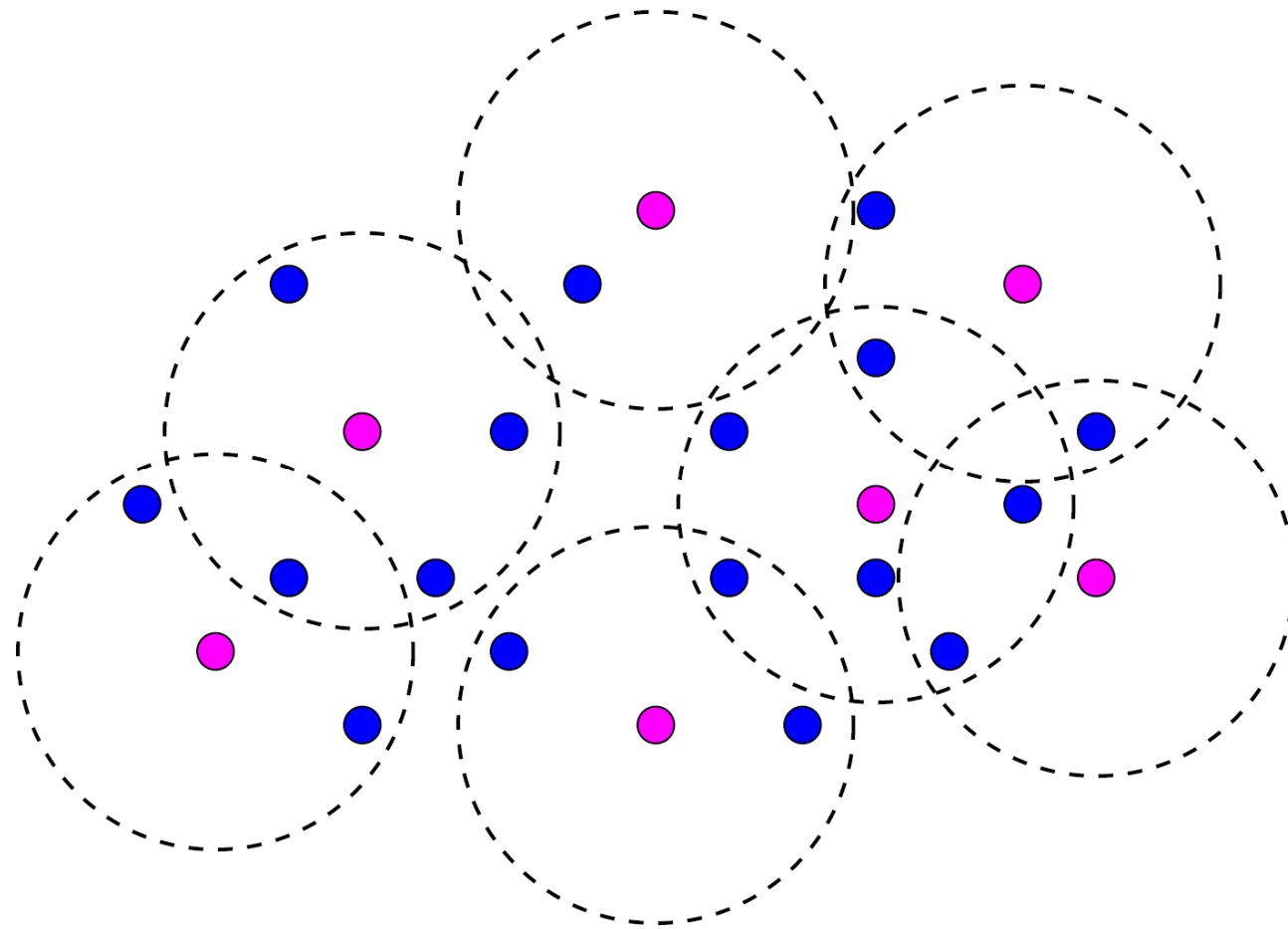
# What is next?

- The hierarchies use location-based hierarchies, quadtree, etc.
- In a graph setting, we use a hierarchy of landmarks.
- Landmark-based routing and location service.

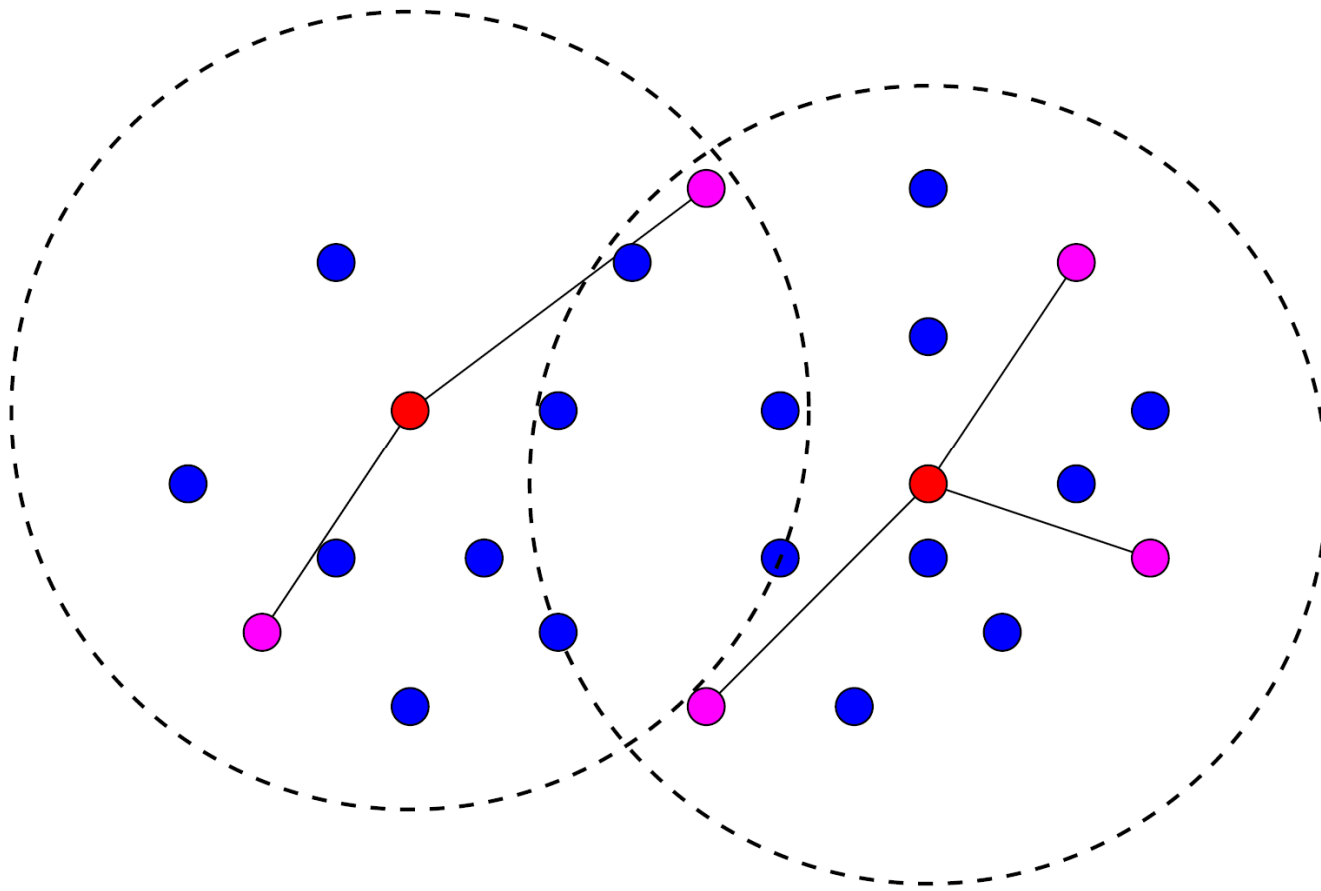
# Landmark hierarchy

- Nodes are organized as landmarks of different levels.
- All nodes are level-0 landmarks.
- Some nodes are selected as level-1 landmarks and so on.
  - Covering: each level  $i-1$  landmark is within distance  $2^i$  from **at least one** level  $i$  landmark and select one as its **parent**;
  - Packing: every two level  $i$  landmarks are of distance  $2^i$  away.

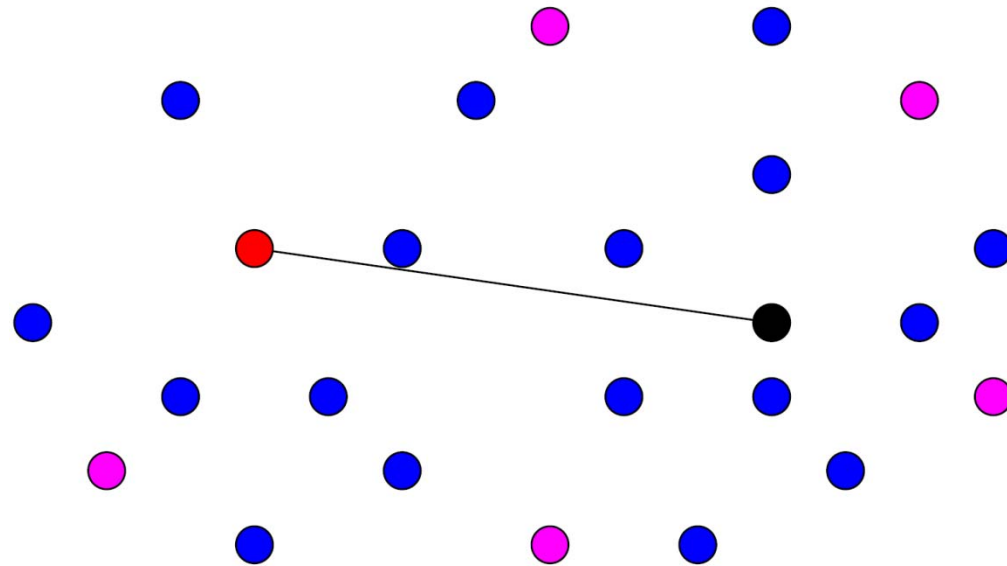
# 1<sup>st</sup> level landmarks



## 2<sup>nd</sup> level landmarks

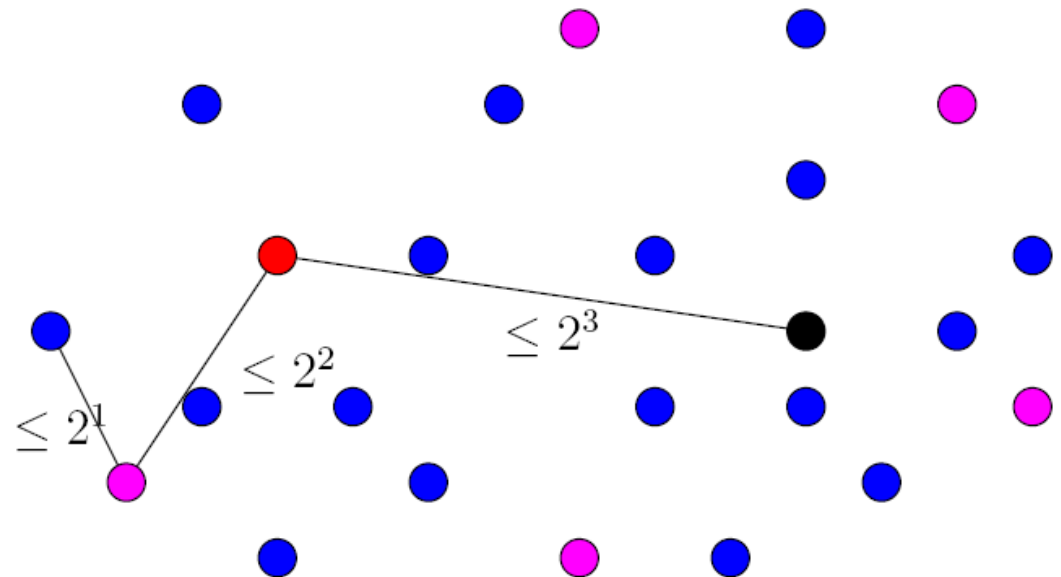


# 3<sup>rd</sup> level landmarks



# Landmark hierarchy

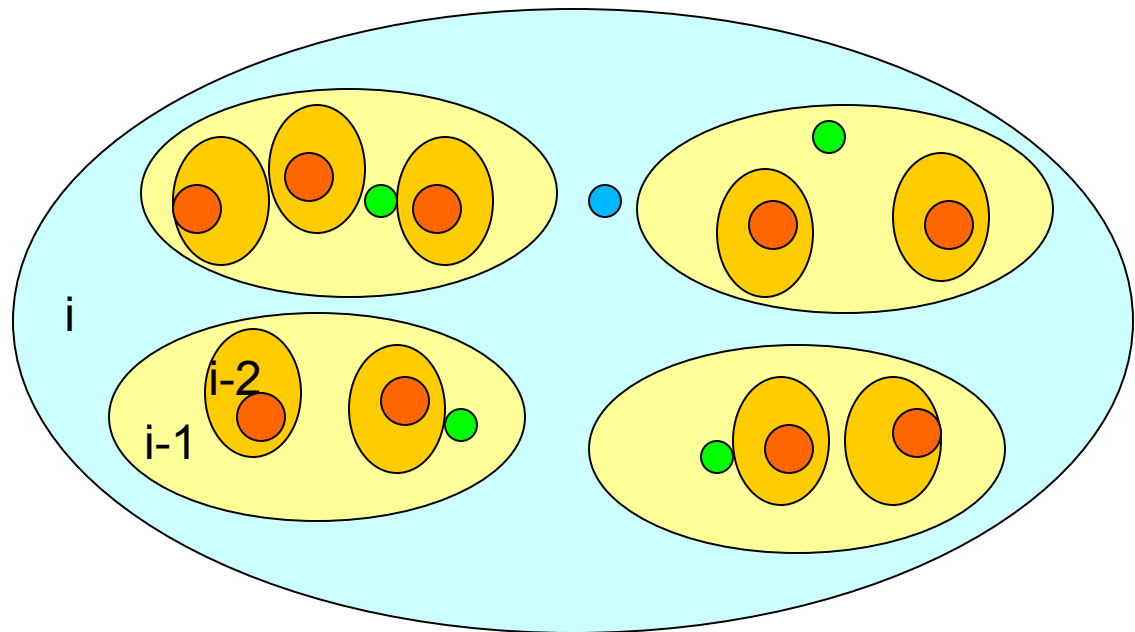
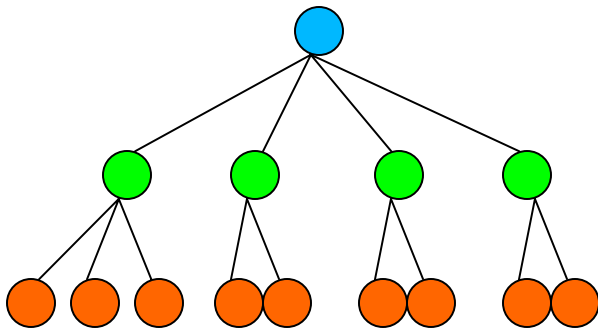
- # levels:  $h = \log n$ .
- A node is within distance  $2^{i+1}$  from its level  $i$  ancestor.
  - The distances along the path to the ancestor is at most  $1+2+4+\dots+2^i = 2^{i+1}-1$
- A landmark at level  $i$  has a **cluster** = its descendants.



# Landmark hierarchy

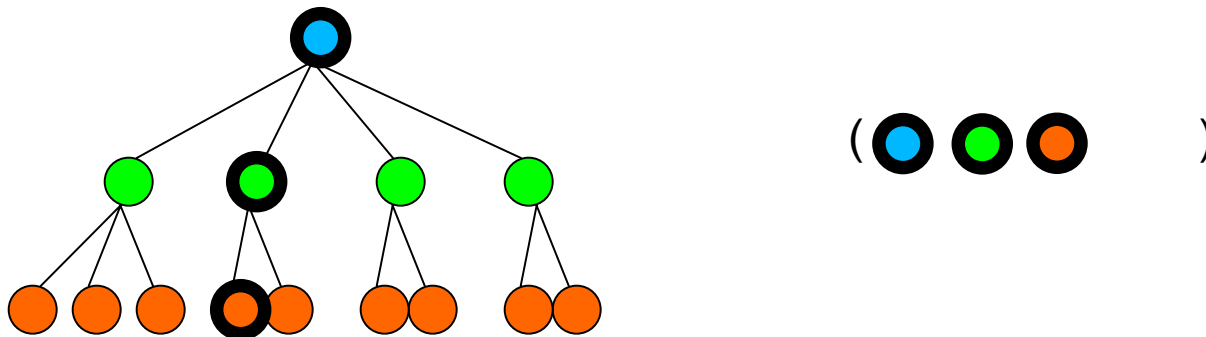
- Each landmark has  $O(1)$  children if the graph has **constant doubling dimension**.
  - Recall constant doubling dimension: each ball of radius  $r$  can be covered by  $\beta$  balls of radius  $r/2$ .
  - Packing argument: a landmark  $u$  at level  $i$  has  $k$  children within distance  $r$ . The ball at this landmark can be covered by  $\beta^3$  balls of radius  $r/8$ . Call them  $B$ . Take balls at  $u$ 's children with radius  $r/8$ , these balls intersect disjoint sets of balls in  $B$ . Thus there are  $O(1)$  children.

# Example



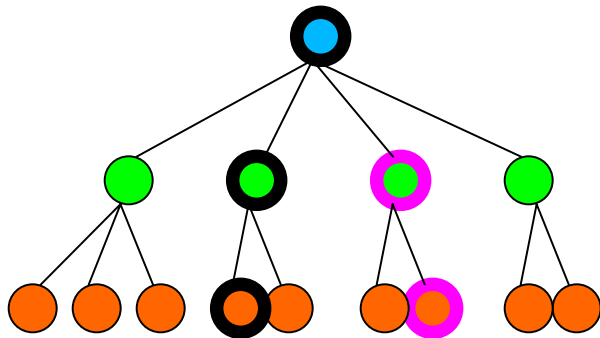
# Addresses/labels/virtual coordinates

- Each node keeps its position in the tree.
- A landmark at level  $i$  has a tuple of length  $h$  including all its ancestors from level  $i$  up and 0 from level  $i$  down.



# How to route?

- We can route on the landmark hierarchy (which is a tree).
- A node p routes to the lowest common ancestor of p, q.
- One can identify the lowest common ancestor from their addresses.
- But how to route down the tree?

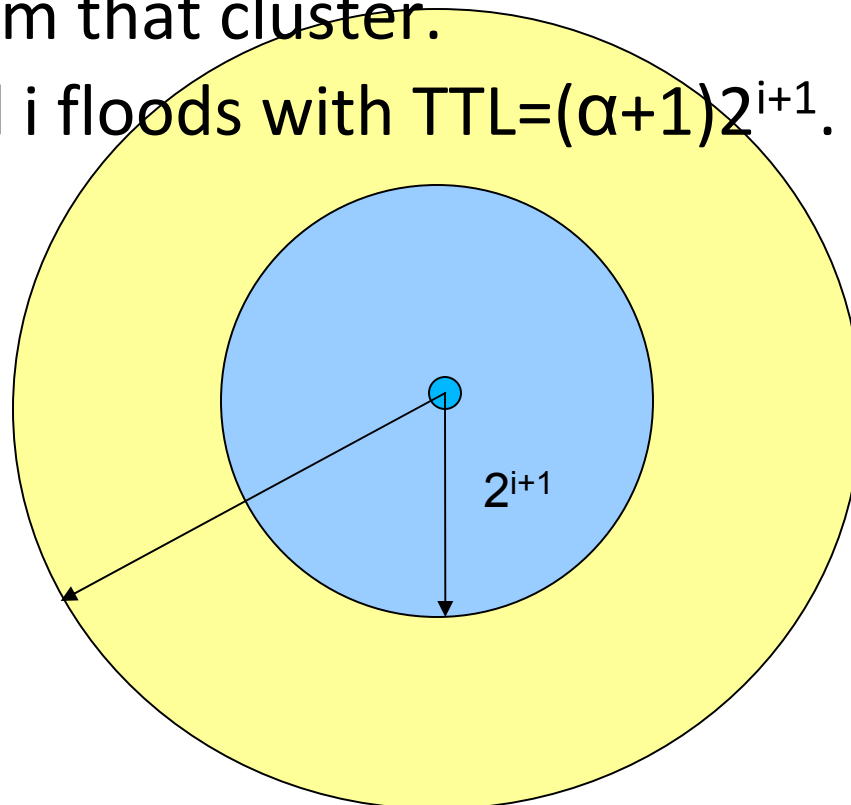


p=(   )

q=(   )

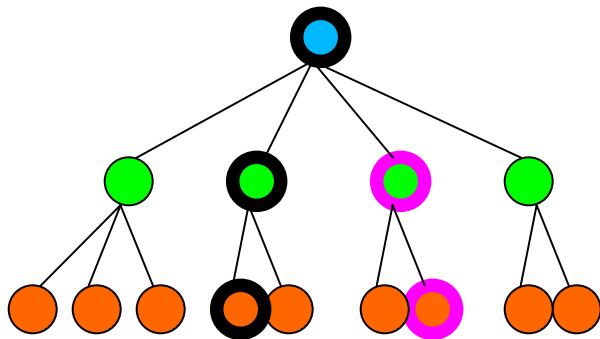
# Routing

- To aid routing, we add **cross-branch** links.
  - A node maintains routing table to reach the cluster of a landmark at level  $i$  if it is within distance  $\alpha 2^{i+1}$  from that cluster.
  - Landmark at level  $i$  floods with  $TTL=(\alpha+1)2^{i+1}$ .



# How to route?

- A node p routes to the **lowest level cluster** that contains q.
  - Can possibly be lower than their common ancestor.
  - Can be identified with q's address.
- At each step, route to a **lower level cluster** containing q.

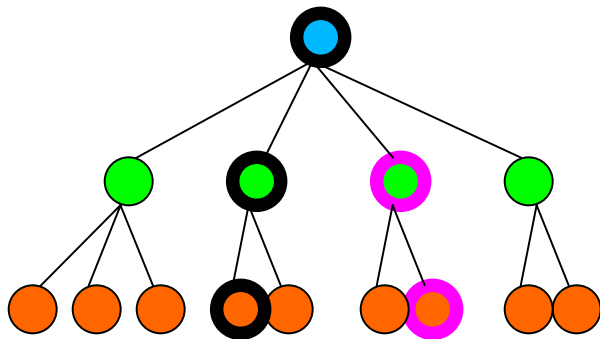


p=(    )

q=(    )

# How to route?

- A node  $p$  routes to the **lowest level cluster  $w$**  (at level  $i$ ) that contains  $q$ .
- First leg to  $w$ 's cluster: at most  $\alpha 2^{i+1}$ .
- Note that  $w$  is also at level  $i-1$ .
- Since  $w$  at level  $i-1$  is **not** a neighboring cluster,  $|pq| > \alpha 2^i$ . Thus the first leg is at most  $2|pq|$ . The next leg is halved in length.
- Total length is at most  $4|pq|$ .

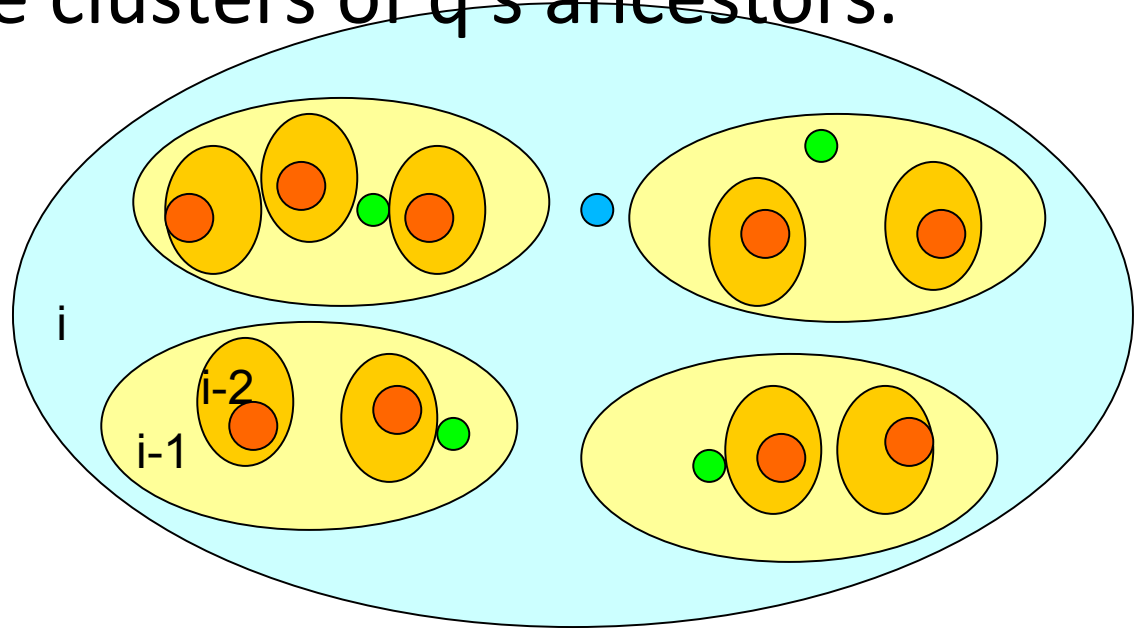
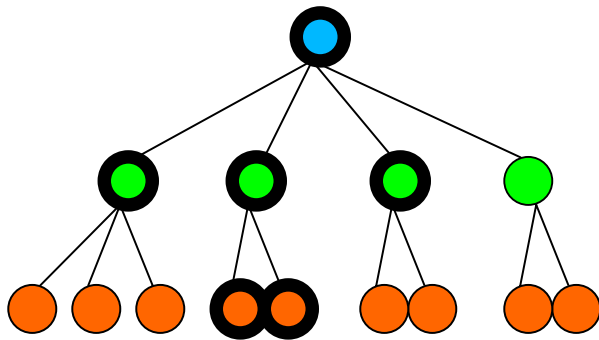


$$p = (\text{blue circle with black outline} \quad \text{green circle with black outline} \quad \text{orange circle with black outline} \quad )$$

$$q = (\text{blue circle with black outline} \quad \text{pink circle with black outline} \quad \text{orange circle with black outline} \quad )$$

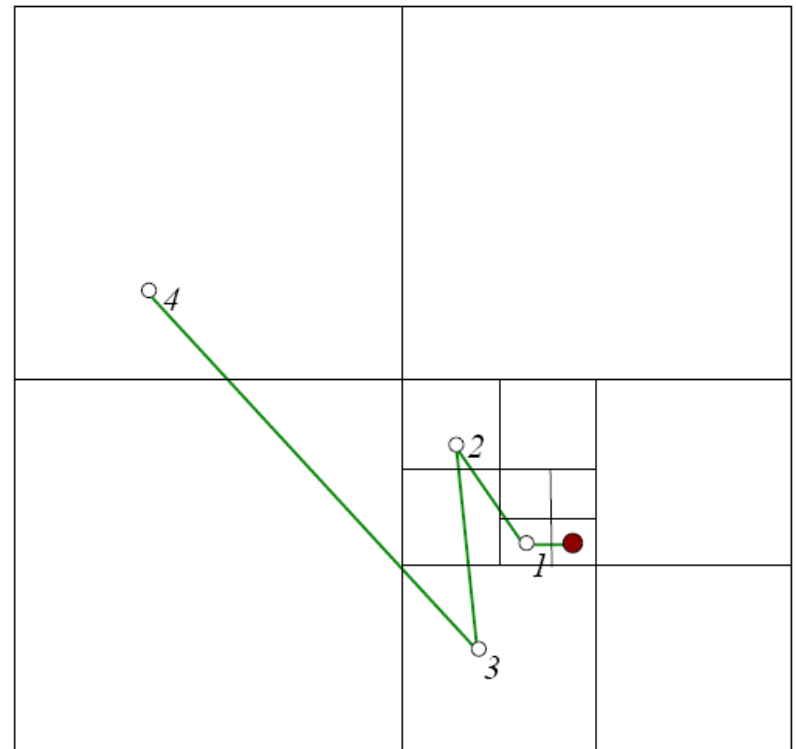
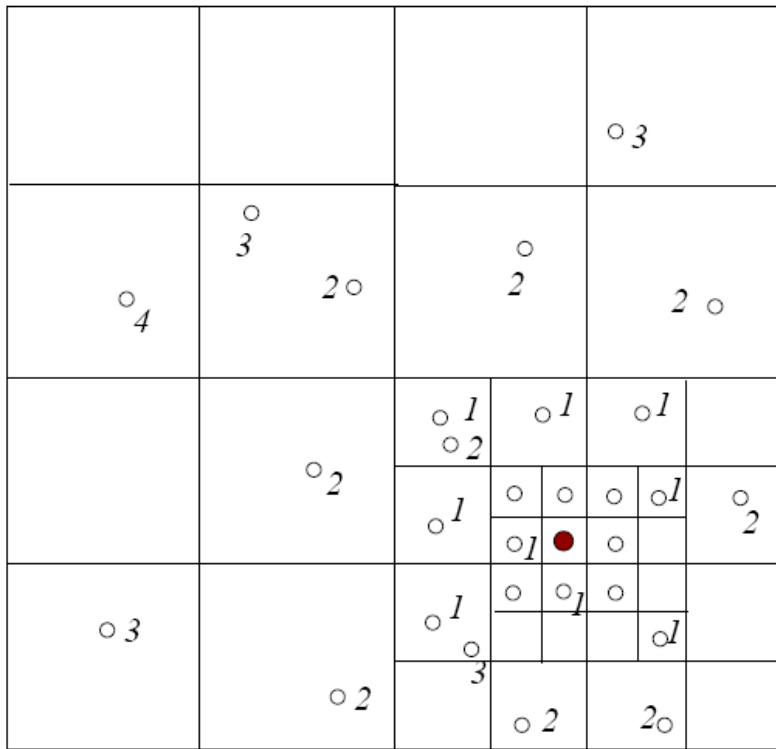
# Data-centric routing, aka, location service

- Each node  $p$  stores its location at a hashed node inside the cluster of all the neighboring clusters.
- FIND: search the clusters of  $q$ 's ancestors.



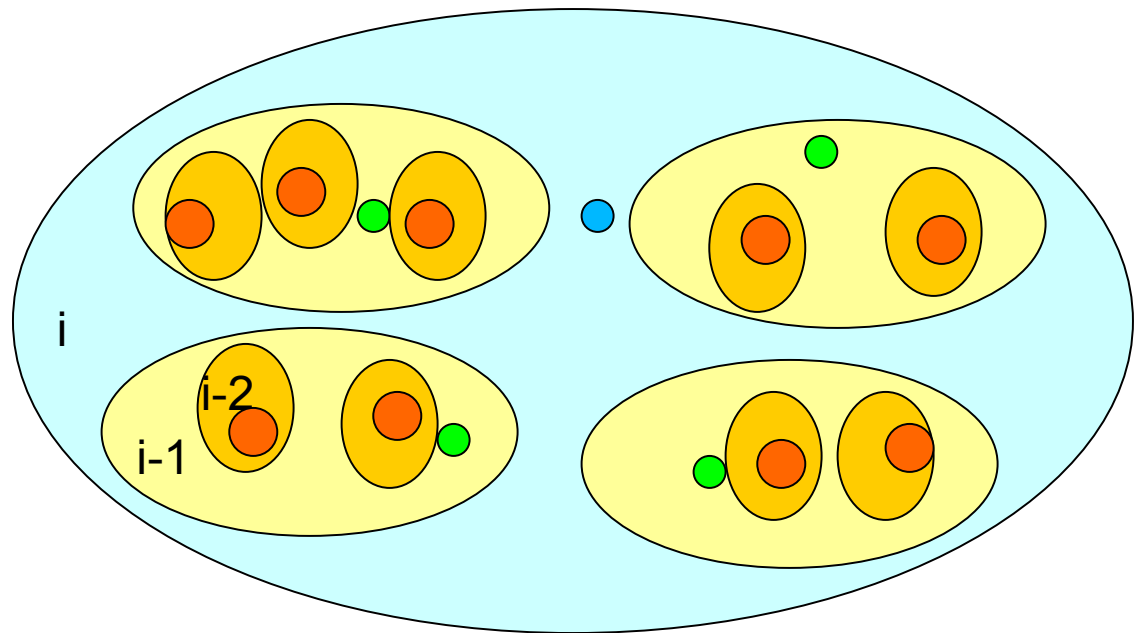
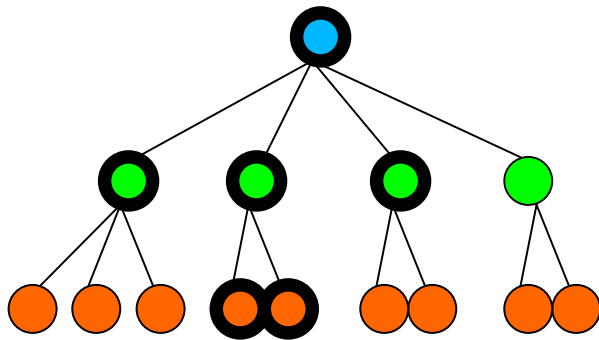
# Data-centric routing, aka, location service

- Example in a quad-tree setting.



# FIND cost

- $q$  finds  $p$  in its ancestor cluster  $w$  at level  $i$ .
- $w$  at level  $i-1$  is not a neighboring cluster of  $p$ . That means,  $|pq| > \alpha 2^i$ .
- $q$ 's cost is at most  $\alpha 2^{i+2}$ , thus at most  $4|pq|$ .



# Open issues on location service

- Make use of node mobility?
  - When two nodes pass by, they keep each other's info.
- Security issue with location service?

# Papers

- **[Li00]** Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger and Robert Morris, [A scalable location service for geographic ad hoc routing](#), MobiCom'00. *Hierarchical structure with universal hashing. Application in location service.*
- **[Abraham04]** I. Abraham, D. Dolev, D. Malkhi , [LLS : a Locality Aware Location Service for Mobile Ad Hoc Networks](#), DIALM-POMC 2004. *Improve the above algorithm in terms of worst-case bound.*
- **[Tsuchiya88]** P. F. Tsuchiya. [The landmark hierarchy: a new hierarchy for routing in very large networks](#). In Sigcomm88. *Landmark hierarchy for routing.*
- **[Funke05a]** S. Funke, L. Guibas, A. Nguyen, Y. Wang, [Distance Sensitive Routing and Information Brokerage in Sensor Networks](#), DCOSS'06. *Improved landmark hierarchy for routing with worse-case guarantee.*