

Location Service

Location service

- Geographical routing requires obtaining the location of the destination.
- What if the sensors move? How to update the location information?
- Internet: domain name server (DNS) translates user-friendly domain name (www.cnn.com) to machine-friendly IP address.

Centralized v.s. distributed location service

- Location server stores the mapping between locations and node IDs.
 - Centralized approach, single point of failure.
 - Communication bottleneck.
 - Location server might be far away.
- Distributed location servers: every node participates and acts as location servers for others.

Locate a mobile user

- “Move” operation:
 - inform system of new location
- “Find” operation:
 - Locate user at his current address.
- Distance-sensitivity: move to **nearby** locations or search for **nearby** users should be cheap.
 - Most moves are local
 - Most queries are local, too.

Model

- Connected, undirected, weighted graph G .
- Weight $w(e)$: cost on edge e .
- $\text{dist}(u, v)$: length of shortest path.
- Diameter $D(G)$: max distance of any two nodes in G .
- Address $\text{Addr}(x)$: current location of x .
- Assume an **efficient routing** scheme.

Model

- Consider a mixed sequence σ of Move and Find operations.
 - $F(\sigma)$: subsequence of all Find operations in σ .
 - $M(\sigma)$: subsequence of all Move operations in σ .
- Cost: message transmissions.
- **Find-stretch**: $\text{cost}(F(\sigma))/\text{OPT}(F(\sigma))$
- **Move-stretch**: $\text{cost}(M(\sigma))/\text{OPT}(M(\sigma))$
- Goal: make Find-stretch and Move-stretch polylogarithmic in n .

A distributed data structure

- Store pointers to locations of each user in various nodes.
- Pointers need to be updated as users move
- Allow some pointers to be inaccurate.

“Pointers at locations nearby to the user, whose update by the user is relatively cheap, are required to be more accurate, whereas pointers at distance locations are updated less often.”

Hierarchical directory server

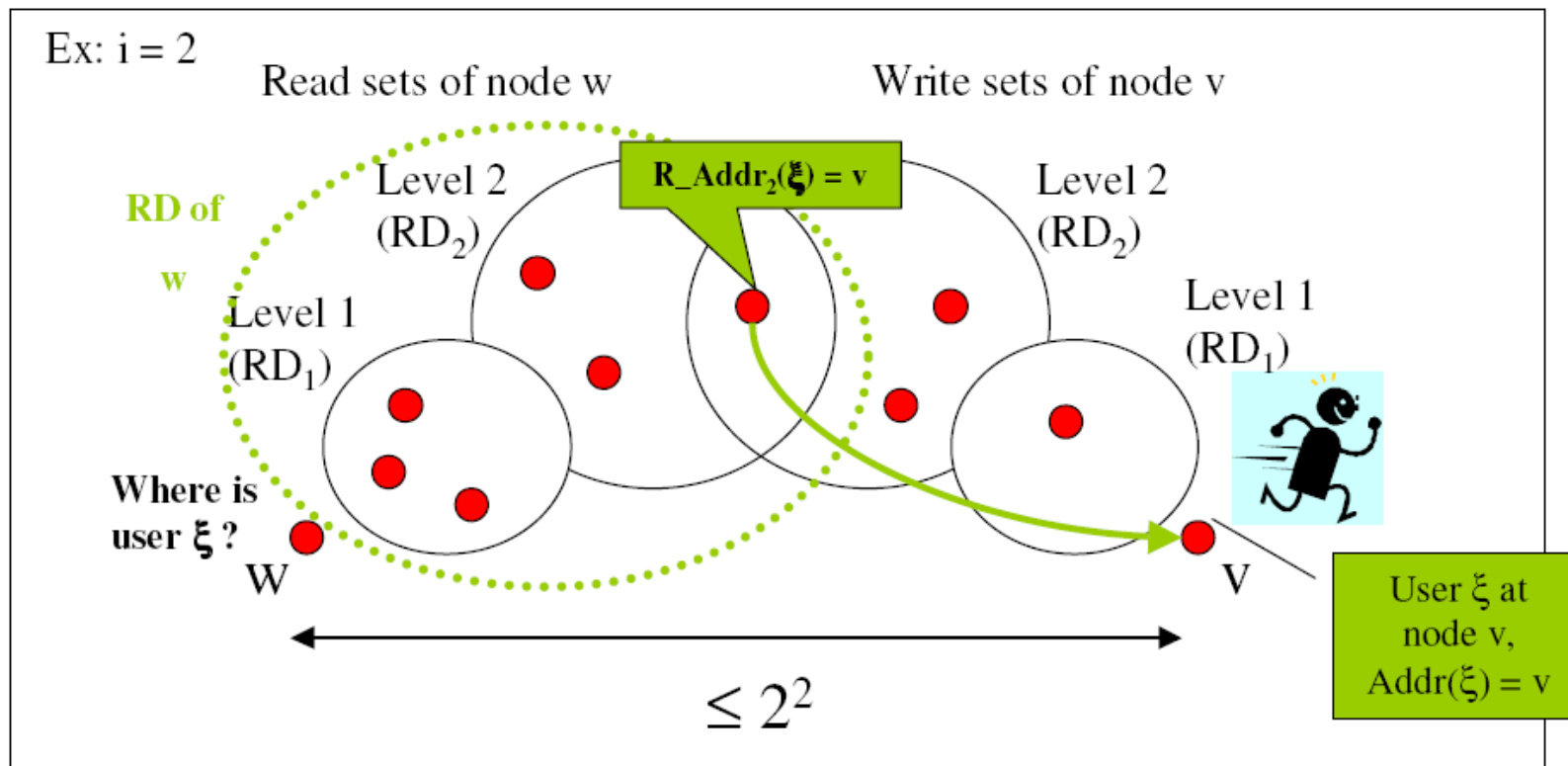
- A hierarchy of $\delta = \lfloor \log D(G) \rfloor$ regional directories RD_i ($1 \leq i \leq \delta$)
- RD_i at level i of the hierarchy enables a searcher to track any user within distance 2^i .
- The address stored for user x at RD_i is called i -th level regional address **$R_addr_i(x)$** --- where x is currently **expected** to be.

Regional directory RD_i

- $Write_i(v)$
 - A node v reports every user it hosts to all nodes in the write set.
- $Read_i(w)$
 - A searching node w queries all nodes in some read set.
- $Read_i(w)$ and $Write_i(v)$ are guaranteed to intersect whenever v and w are within distance 2^i of each other.

2^i -regional matching

- $\text{Read}_i(w)$ and $\text{Write}_i(v)$ are guaranteed to intersect whenever v and w are within distance 2^i of each other.



2^i -regional matching

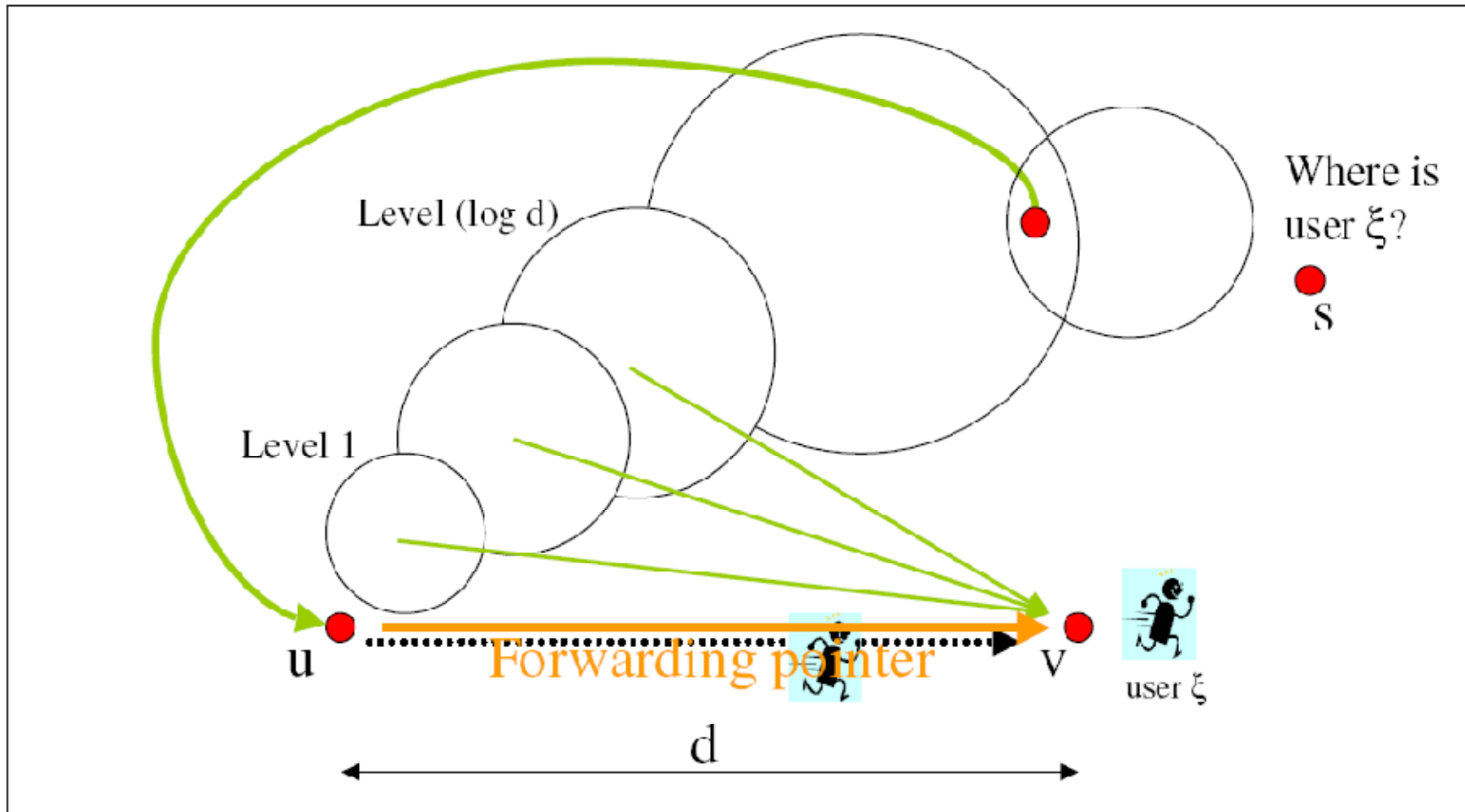
- Find operation invoked at node w will succeed at the lowest possible level enabled by the distance from w to v .
- At the highest level this operation will always succeed.

- Two questions:
- What if the network diameter grows?
- What if the address maintained at the location directory is out of date?

“Forwarding addresses”

- Whenever a user x moves, it update its regional directory **on all levels**.
 - Too expensive!
- Update only **$\log d$** lowest levels
 - The lower the level, the more up-to-date is the regional address
 - Low communication cost
 - The address **$R_addr_i(x)$** might point to an old address.

“Forwarding addresses”



The reachability invariant

- Define the tuple of regional addresses

$$A(x) = \langle R_Addr_1(x), \dots, R_Addr_\delta(x) \rangle$$

- $R_Addr_1(x) = Addr(x)$ the true address.
- The reachability invariant: if at any time, $R_Addr_i(x)$ stores a pointer **Forward (x)** to $R_Addr_{i-1}(x)$.
- This may result in a long chain of forwarding pointers.

The proximity invariant

- The reachability invariant: if at any time, the distance travelled by x since the last time $R_Addr_i(x)$ was updated satisfies

$$| \text{Migrate}_i(x) | \leq 2^{i-1} - 1$$

- $\text{Migrate}_i(x)$: the actual migration path from $R_Addr_i(x)$ to its current location.
- A node currently hosting user x maintains **Tuple of migration counters**: $C(x) = \langle C_1(x), \dots, C_\delta(x) \rangle$
 - $C_i(x)$: distance travelled since the last update.

Updating regional addresses

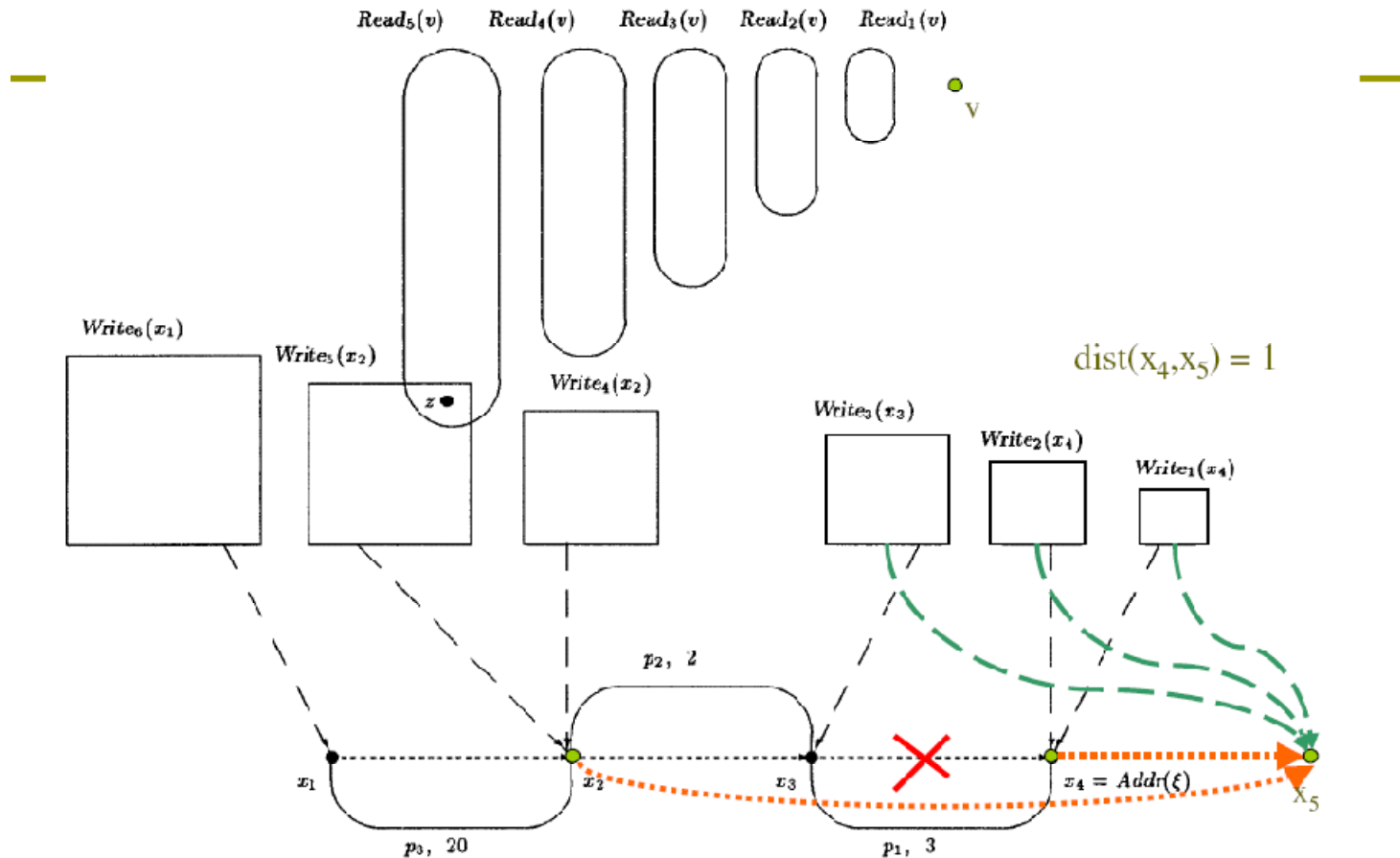
Whenever user moves from a node s to a node t :

- Increase all migration counters C_i by $\text{dist}(s,t)$.

If the highest level counter C_j reaches the upper limit $(2^{j-1} - 1)$

- Update the regional directory at levels 1 to j : set to t .
- Set forwarding pointer at $R_Addr_{j+1}(x)$ leading to t .
- Relocate user x together with its tuples $A(x)$ and $C(x)$.

Example



x4 stores:

$$\overline{A}(\xi) = \langle x_4, x_4, x_3, x_2, x_2, x_1 \rangle$$

$$\overline{C}(\xi) = \langle 0, 0, 3, 5, 5, 25 \rangle$$

x5 stores:

$$\overline{A}(\xi) = \langle x_5, x_5, x_5, x_2, x_2, x_1 \rangle$$

$$\overline{C}(\xi) = \langle 0, 0, 0, 6, 6, 26 \rangle$$

Discussion

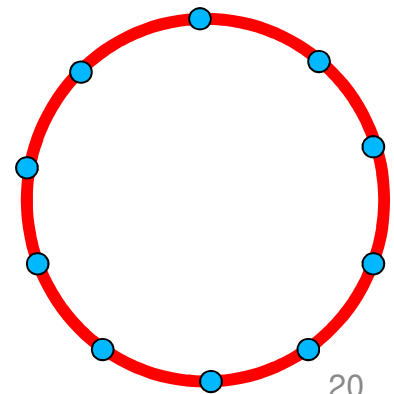
- Proof of the cost in the full paper.
- Location service for one mobile user.
- What if **all** the nodes in the network are mobile?

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.

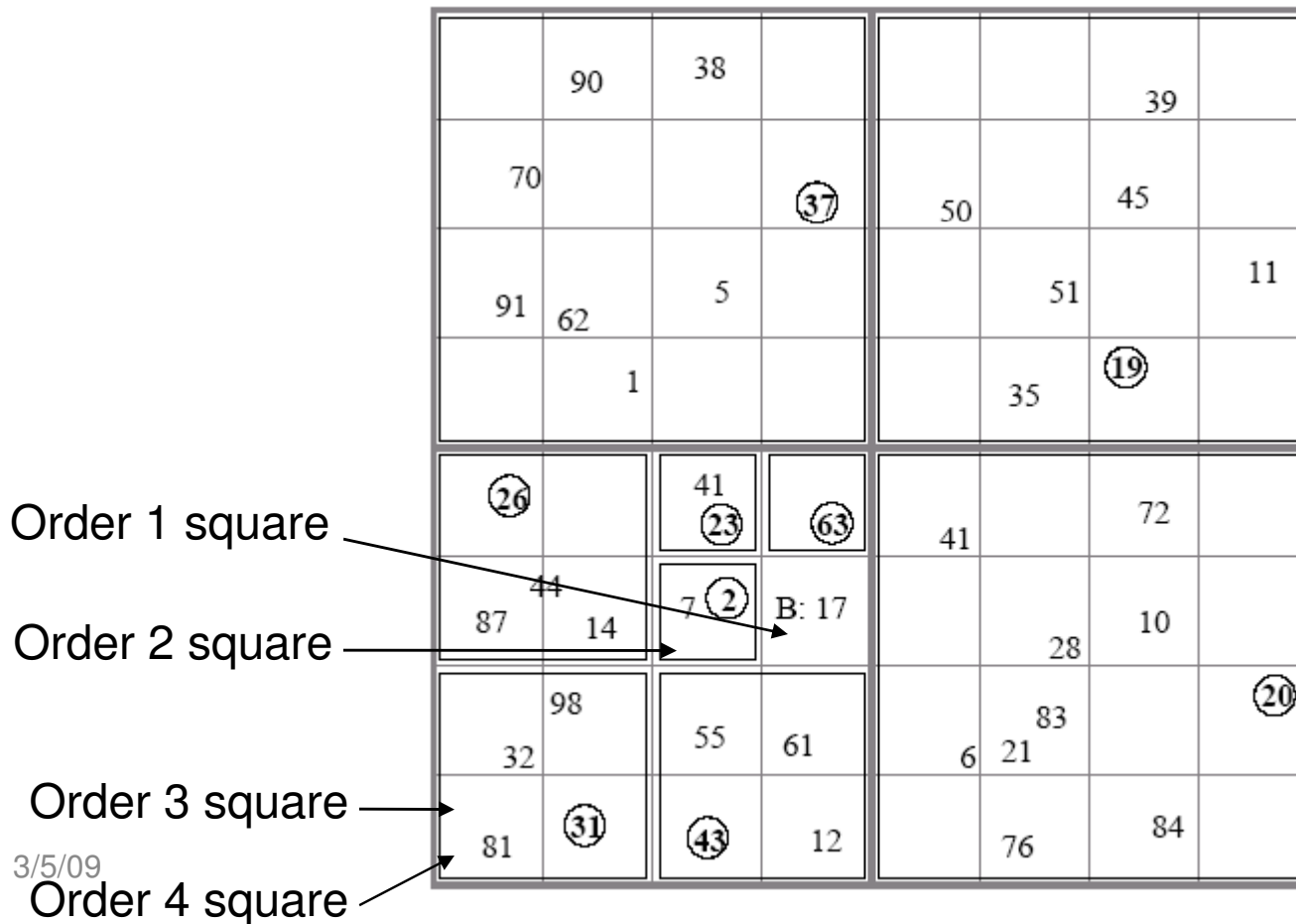
GLS: Grid Location Service

- 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

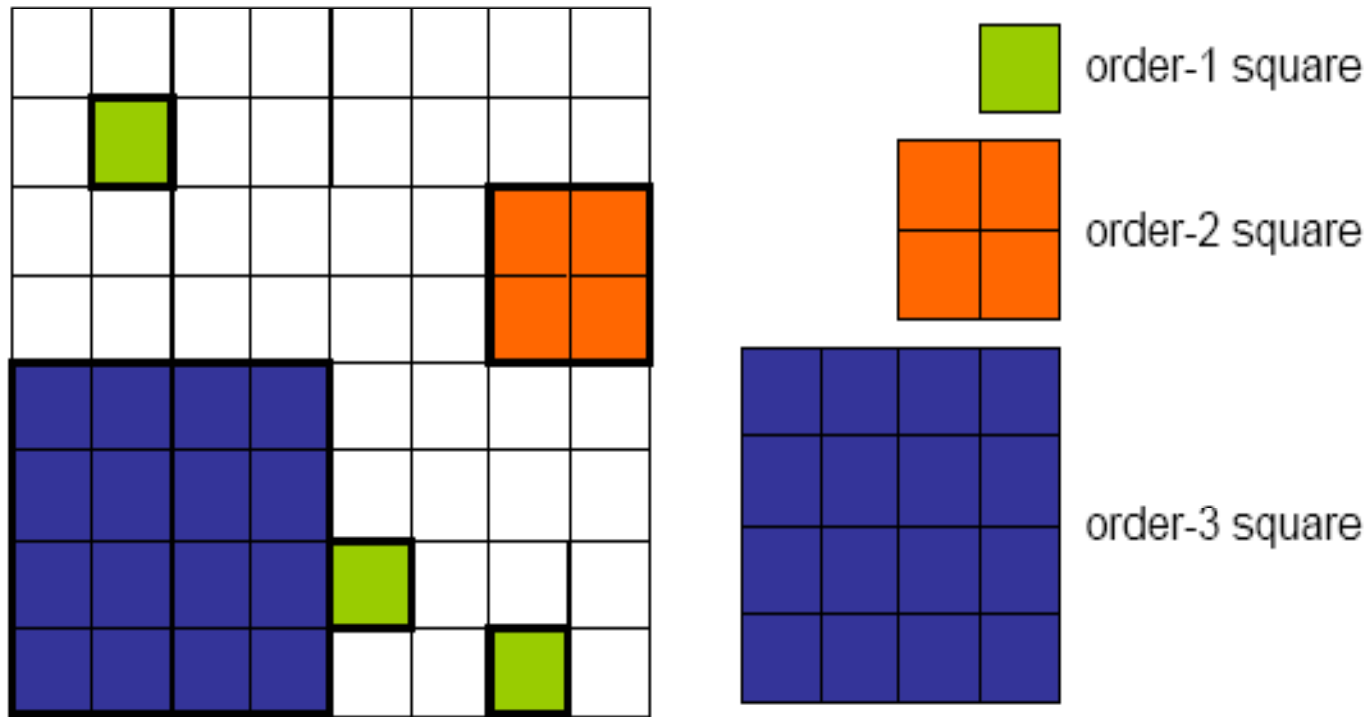


Recursive partitioning

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



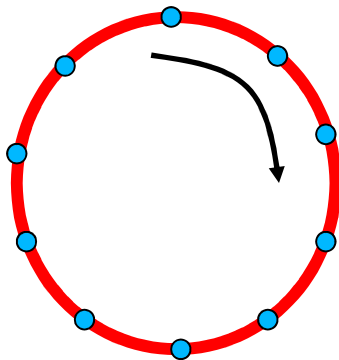
Partitioning the world



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					39
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		10
	98					28	
32		55	61		6	21	83
81	31	43	12		76		84
							20

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 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	6,10,20,76	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 61	2,5,6,10,43 55,61,63,81 87,98		6,20,28,41 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	6,10,20,76	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		21	
81	31	43	12		A: 76	84

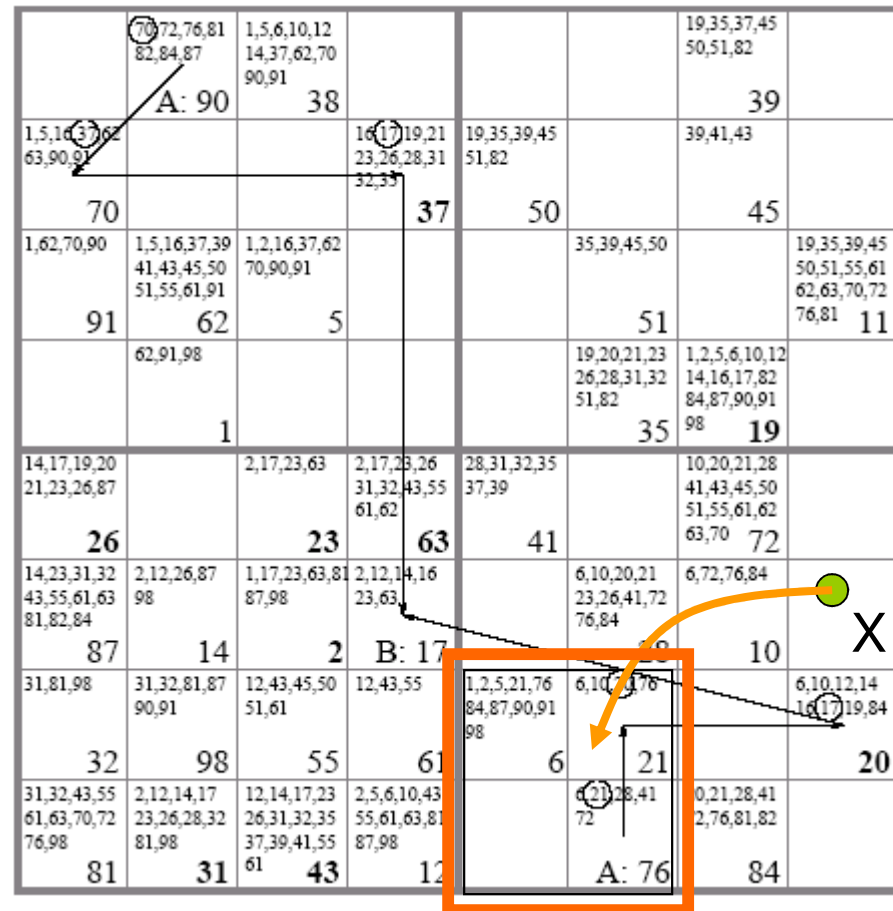
Location queries

- 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 E 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				35	1,2,5,6,10,12 14,16,17,82 84,87,90,91 98 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 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	6,10,20,76	6,10,12,14 16,17,19,84
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	2,5,6,10,43 55,61,63,81 87,98		6,20,28,41 72	20,21,28,41 72,76,81,82
81	31	61 43	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.



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.

