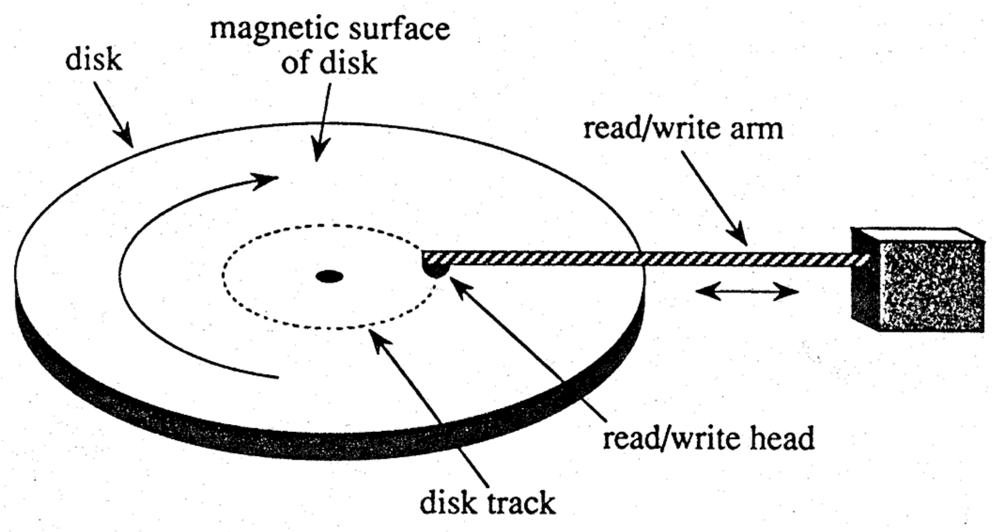
Multidimensional Spatial Data Structures

Nick Kuilema
nicnak@cs.umd.edu

Survey Paper by Jeffery Scott Vitter

Magnetic Disk Drives as Secondary Memory



- ☆ I/O Crisis!
- ☆ Time for rotation ≈ Time for seek.
- ☆ Amortize search time by large block transfer so that
  Time for rotation ≈ Time for seek ≈ Time to transfer data.
- ☆ Parallel disks.

Multidimensional Data Spaces

- Want to provide efficient searching
- Insertion and deletion
- Needed for GIS range search

Desired Complexity

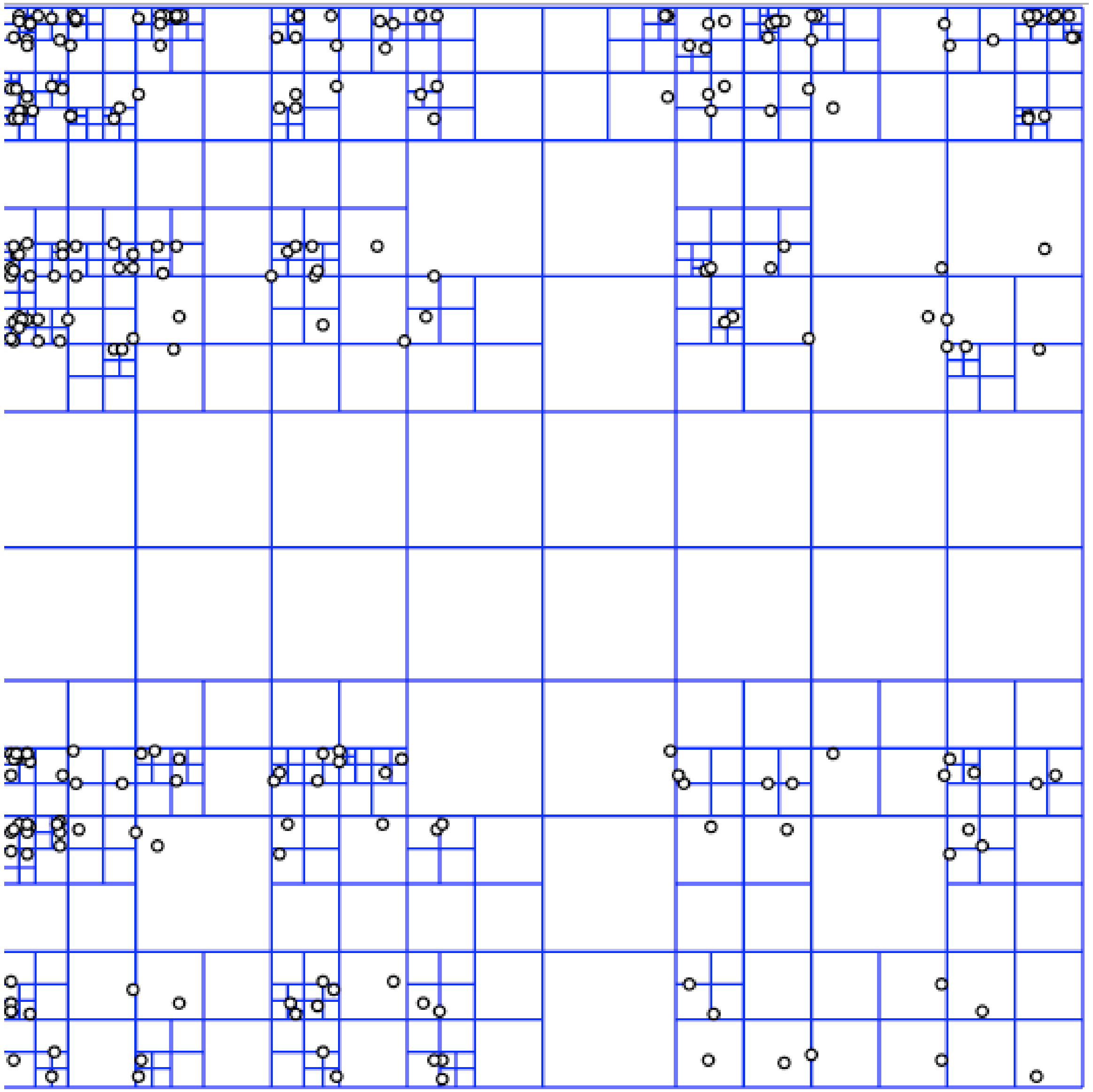
Same at using a B-tree for 1-D range search

1. Get a combined search and answer cost for queries of $O(\log_B N + z)$ I/Os,
2. Use only a linear amount (namely, $O(n)$ blocks) of disk storage space, and
3. Support dynamic updates in $O(\log_B N)$ I/Os (in the case of dynamic data structures).

Try for Linear Space

- Cross Tree
  - Upper levels - Weight balanced B-Tree
  - Lower levels - Quad Trees
- O-Tree

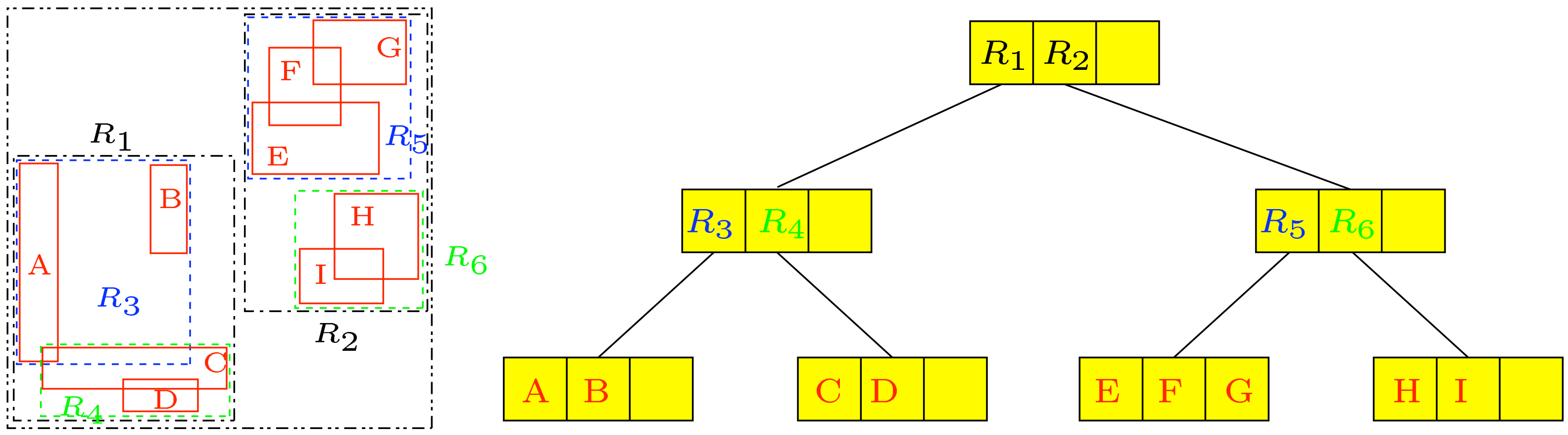
Quadtree



R-Trees

- Internal nodes have degree $\Theta(B)$
- Leaves store $\Theta(B)$ items
- Each node has a bounding box
- Points may be in more then one child node

R-trees



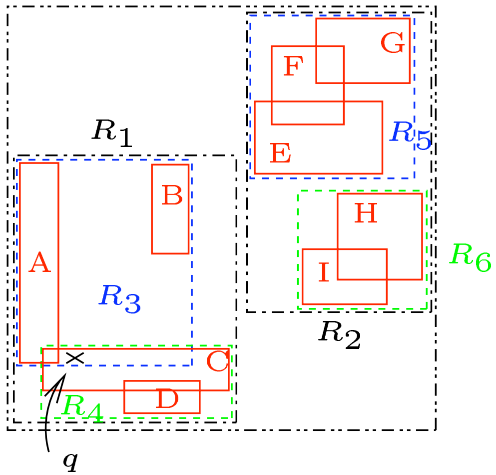
☆ Structure for storing d-dimensional rectangles.

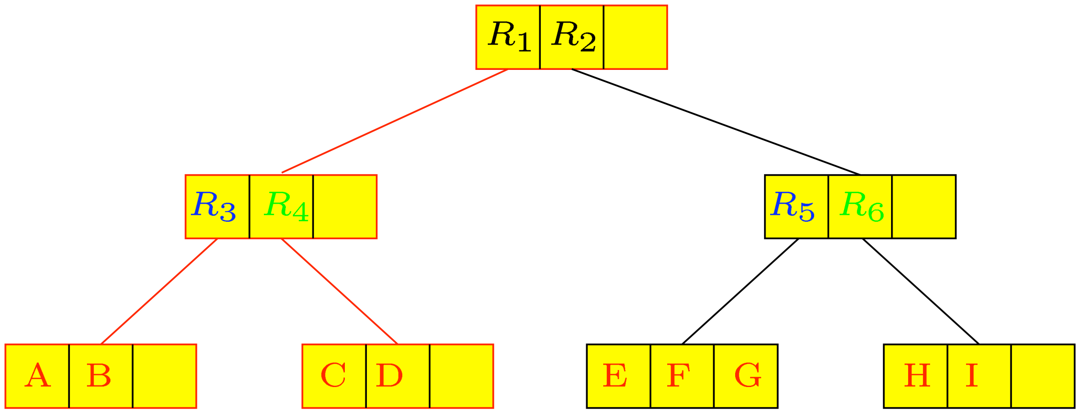
☆ Structured like B-tree:

- Data in leaves.
- Fan-out B.
- Rebalancing basically like in B-trees.

☆ Internal node holds minimal bounding rectangle of each subtree.

Querying R-trees





☆ Query with point q:

- Visits all nodes with minimal bounding rectangle containing q.

☆ Minimal bounding rectangles allowed to overlap:

- Small overlap or perimeter desirable.
- Several insert/split heuristics (R^+-trees, R*-trees, Hilbert trees, …) have been proposed, surveyed in [G89, GG98].

R*-Tree

- Seems to give best performance
- Heuristic for Insertion
    - Recursively insert into bounding box that expands the least
    - If tie, add to subtree with smallest bounding box
- When a node is full remove a percentage of elements and re-insert
    - If node is still full, split it
    - Improves packing and query times

Constructing ("Bulk Loading") an R-tree

☆ Using repeated insertion takes $O(N \log_B n)$ I/Os.

☆ Bottom-up algorithms [RL85, KF93, DKLPY94, LLE96, vdBSW97]

- Rectangles are sorted (using space-filling curve)
  $\Longrightarrow \quad O(n \log_m n)$-I/O algorithm.
- Can only handle construction—not e.g. "bulk updates."
- Questionable query performance, esp. in high dimensions.

☆ Buffer technique immediately applies:

- Conceptually simple (algorithm unchanged).
- Modular design (all R-tree insert heuristics can be used).
- Handles all "bulk" operations.

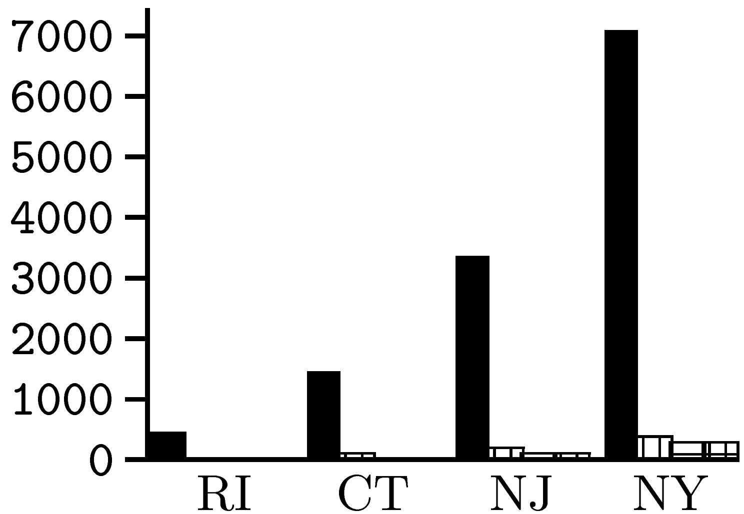
TIGER/Line Data

☆ TIGER/Line data from U.S. Census Bureau
(standard benchmark data for spatial databases)

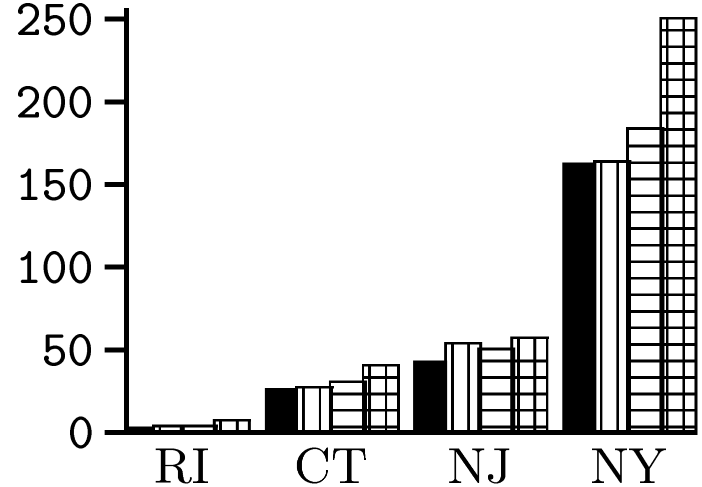
State	Category	Size	Objects
Rhode Island (RI)	Roads	4.3 MB	68,278
	Hydrography	0.4 MB	7,013
Connecticut (CT)	Roads	12.0 MB	188,643
	Hydrography	1.8 MB	28,776
New Jersey (NJ)	Roads	26.5 MB	414,443
	Hydrography	3.2 MB	50,854
New York (NY)	Roads	55.7 MB	870,413
	Hydrography	10.0 MB	156,568
All	Roads	98.5 MB	1541,777
	Hydrography	15.4 MB	243,211

Experimental Results: R-tree

- Buffers on all nodes for simplicity (buffer size $\Theta(B)$)

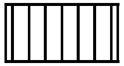
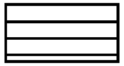
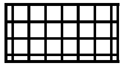


Building R-tree on road data
(I/Os in thousands)



Query with (1/10) hydro data
(I/Os in thousands)

- Naive repeated insertion: 

- Buffer: Size of buffer  $\frac{B^2}{4}$  $\frac{B^2}{2}$  $2B^2$

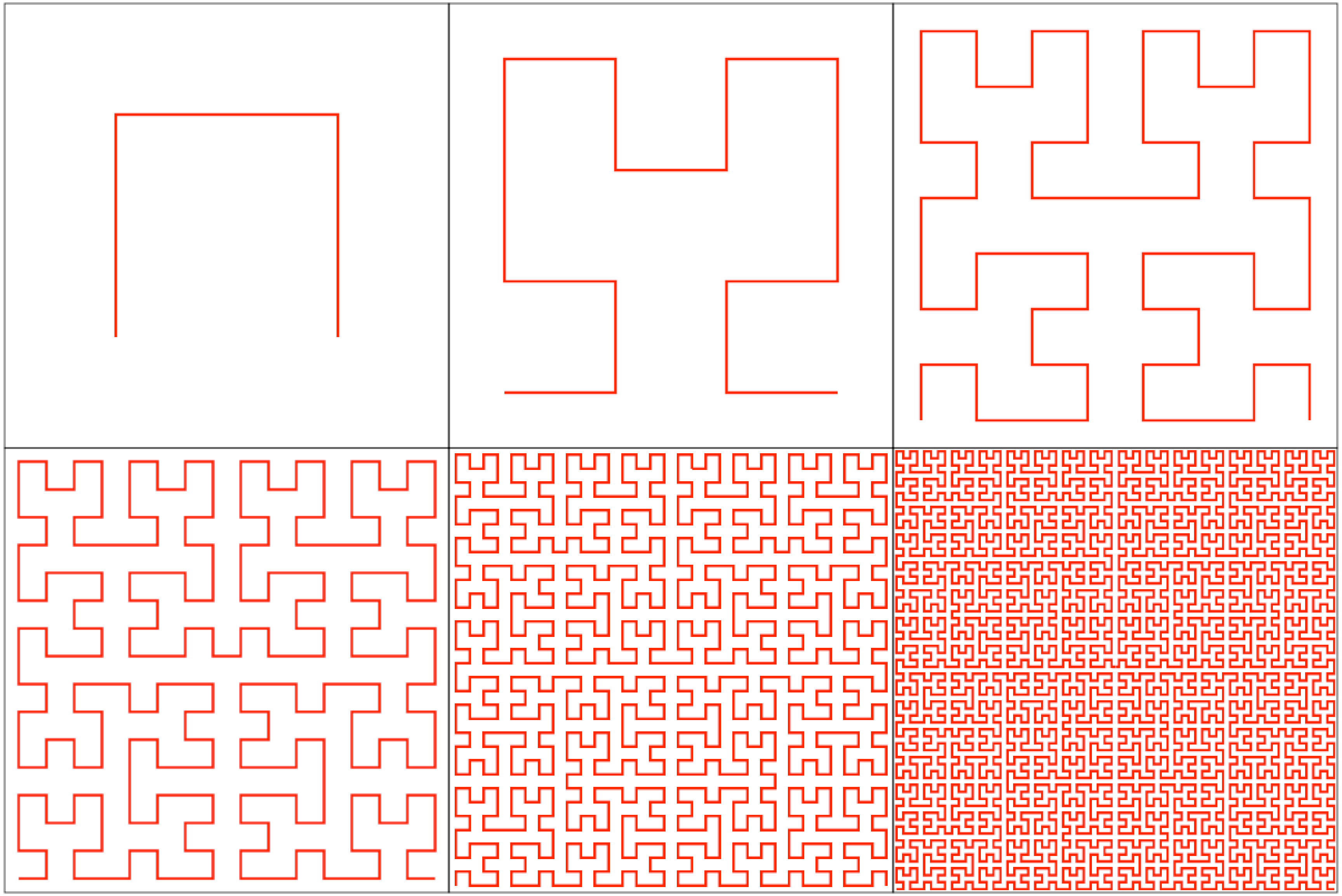
Experimental Results: I/O Costs for R-trees

Data Set	Update Method	Update with 25% of the data		
		Building	Querying	Packing
RI	naive	259,263	6,670	64%
	Hilbert	15,865	7,262	92%
	buffer	13,484	5,485	90%
CT	naive	805,749	40,910	66%
	Hilbert	51,086	40,593	92%
	buffer	42,774	37,798	90%
NJ	naive	1,777,570	70,830	66%
	Hilbert	120,034	69,798	92%
	buffer	101,017	65,898	91%
NY	naive	3,736,601	224,039	66%
	Hilbert	246,466	230,990	92%
	buffer	206,921	227,559	90%

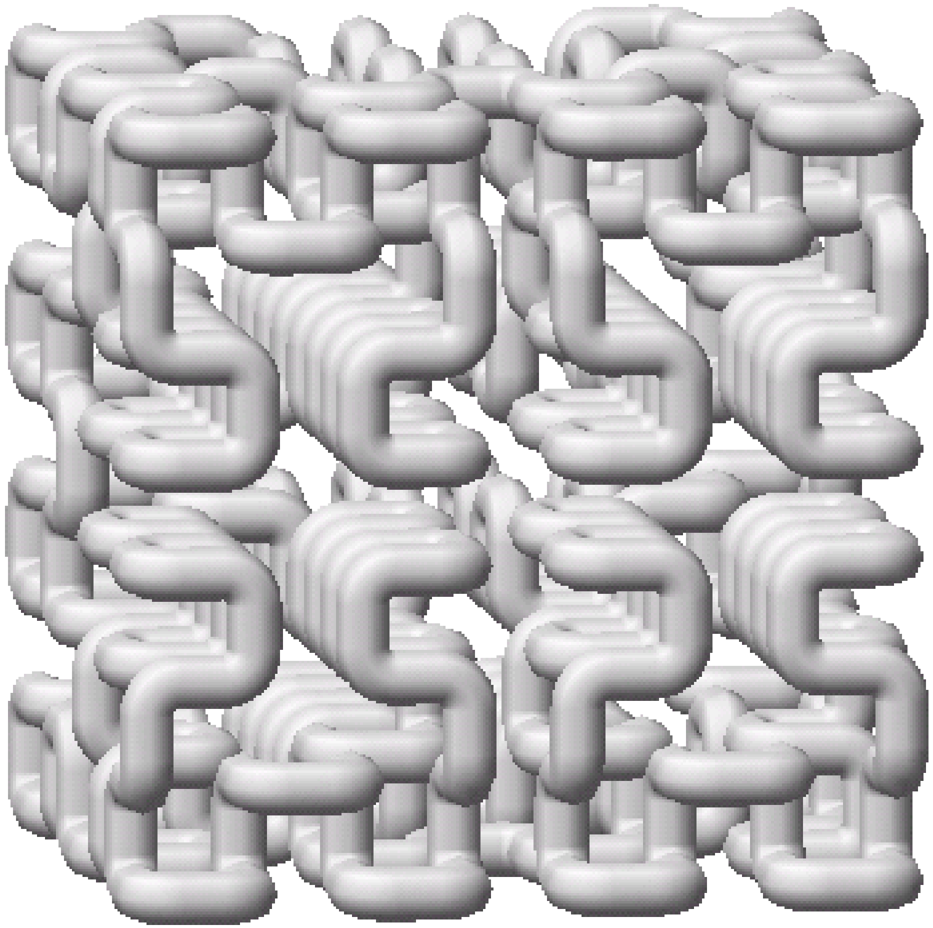
Space Filling Curves

- Impose a total ordering over a multi-dimensional space
- Hilbert curve
- Z-order
- Sierpiński curve
- Peano curve

2-D Hilbert Curve



3-D Hilbert Curve



HPC application Moldyn

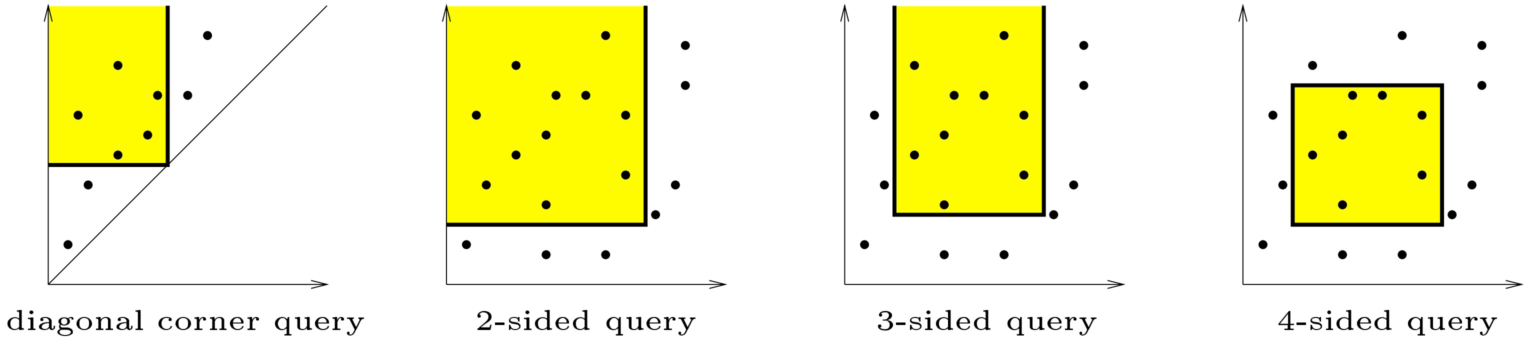
- Molecular dynamics simulation
- Computes interactions between each pair of particles
- Hundreds of thousands of particles
- Millions of interactions

Moldyn results

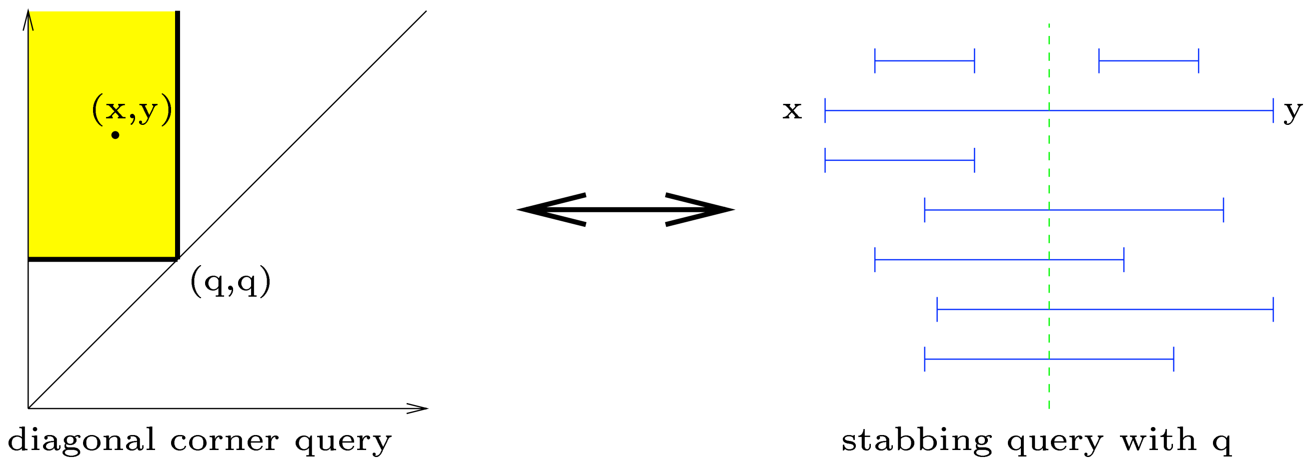
First touch re-ordering is reordering data set as it is visited

Data Reordering	Computation Reordering	L1 Cache Misses	L2 Cache Misses	TLB Misses	Cycles
RCM	None	0.96441	0.81847	0.49658	0.86650
First Touch	None	0.87487	0.76548	0.31928	0.79069
Hilbert	None	0.87978	0.78074	0.26397	0.80731
None	Hilbert	0.45053	0.12157	0.74006	0.37778
None	Blocking	0.30376	0.23557	0.19278	0.61910
First Touch	Hilbert	0.33735	0.14314	0.00806	0.38773
Hilbert	Hilbert	0.25816	0.10139	0.00624	0.26550

Online Range Searching



For example, indexing constraints in constraint query languages:



External Range Searching Results

- ☆ *Corner (Interval tree):* $O(n)$ space, $O(\log_B n + z)$ I/Os query, $O(\log_B)$ I/Os update [AV96]

- ☆ *2-sided:* $O(n \log\log B)$ space, $O(\log_B n + z)$ I/Os query, $O(\log_B n)$ amortized updates [RS94]

- ☆ *3-sided:* $O(n)$ space, $O\big(\log_B n + z + IL^*(B)\big)$ I/Os query, $O\big(\log_B n + \frac{1}{B}(\log_B n)^2\big)$ I/Os amortized updates [SR95]

- ☆ *4-sided:* $O\big(n(\log N)/\log\log_B n\big)$ space, $O(\log_B n + z + IL^*(B))$ I/Os query [SR95]

- ☆ 3-d range queries: $O\big((\log\log\log_B n)\log_B n + z\big)$ I/Os query [VV96]

- ☆ Halfspace queries: $O(n \log n)$ space, $O(\log_B n + z)$ I/Os query [AAEFV98]

- ☆ Lower bounds: [SR95] Cannot achieve simultaneously $O\big(n(\log n)/\log\log_B n\big)$ space, $O\big((\log_B n)^c + z\big)$ I/Os query.

Acknowledgements

Jeffery Scott Vitter, Duke. Paper and slides at

http://www.cs.duke.edu/~jsv/Papers/catalog/node38.html

John Mellor-Crummey, David Whalley, Ken Kennedy

http://www.cs.rice.edu/~johnmc/papers/reordering-ijpp01.pdf