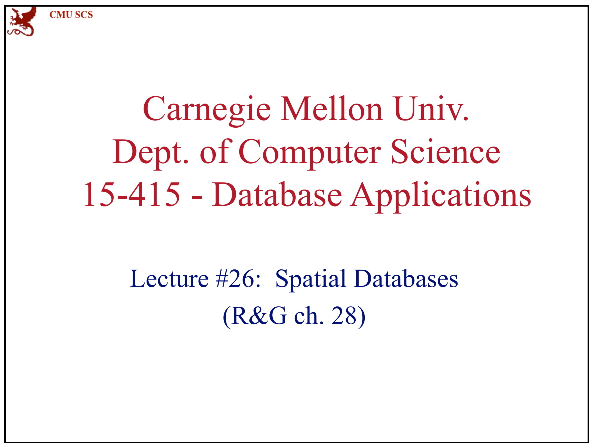
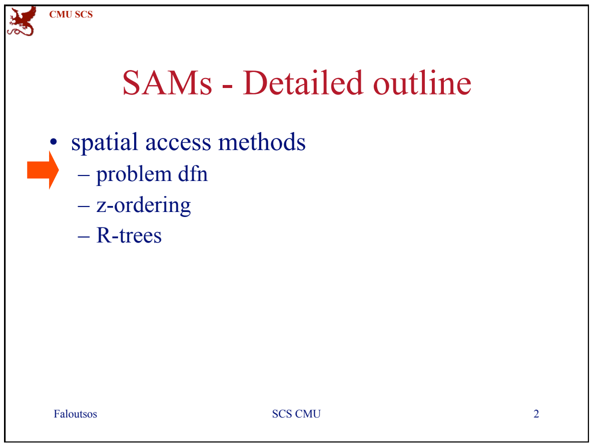


CMU SCS

Carnegie Mellon Univ. Dept. of Computer Science 15-415 - Database Applications

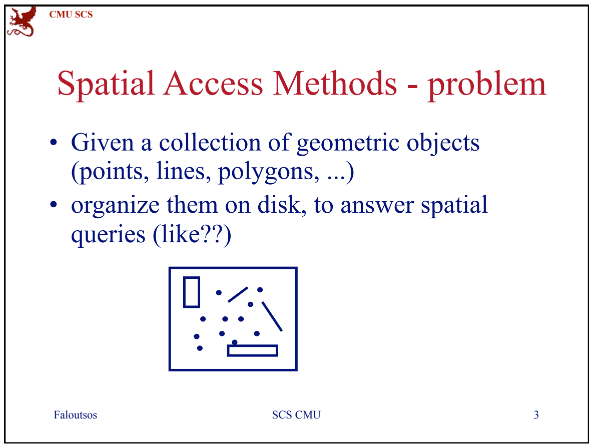
Lecture #26: Spatial Databases
(R&G ch. 28)



CMU SCS

SAMs - Detailed outline

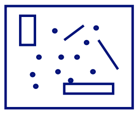
- spatial access methods
  - problem dfn
  - z-ordering
  - R-trees

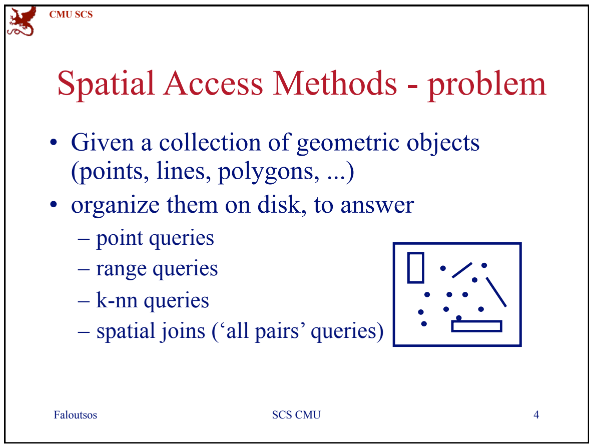


CMU SCS

Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer spatial queries (like??)

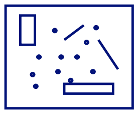




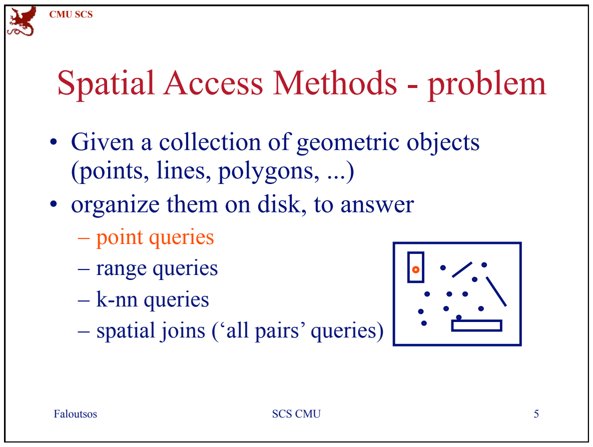
CMU SCS

Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



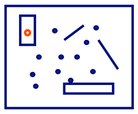
Faloutsos SCS CMU 4



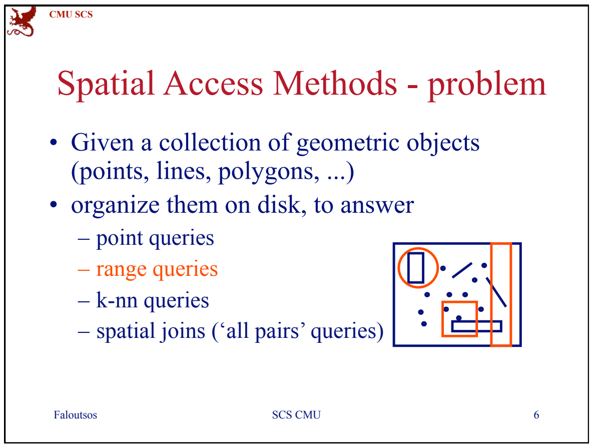
CMU SCS

Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



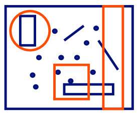
Faloutsos SCS CMU 5



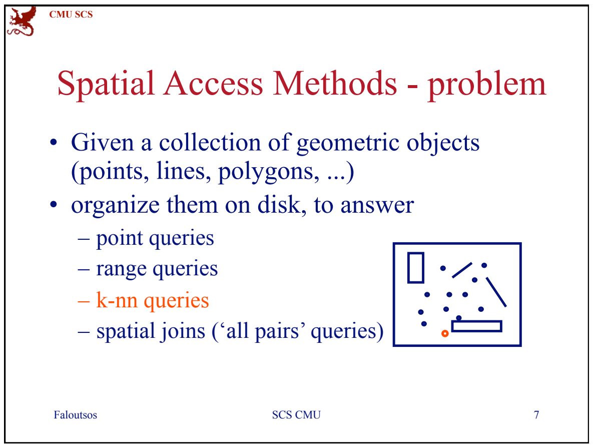
CMU SCS

Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



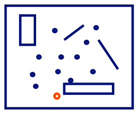
Faloutsos SCS CMU 6



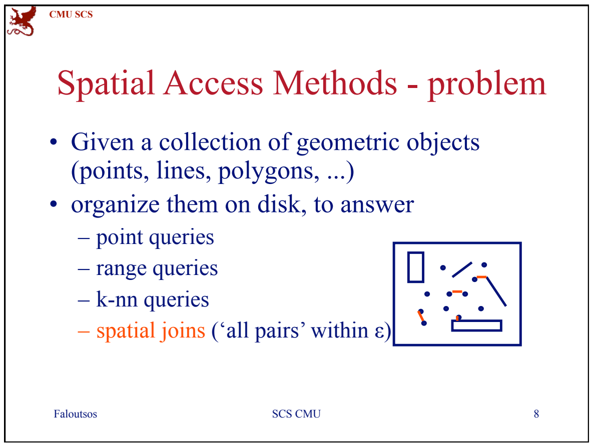
CMU SCS

Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



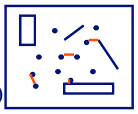
Faloutsos SCS CMU 7



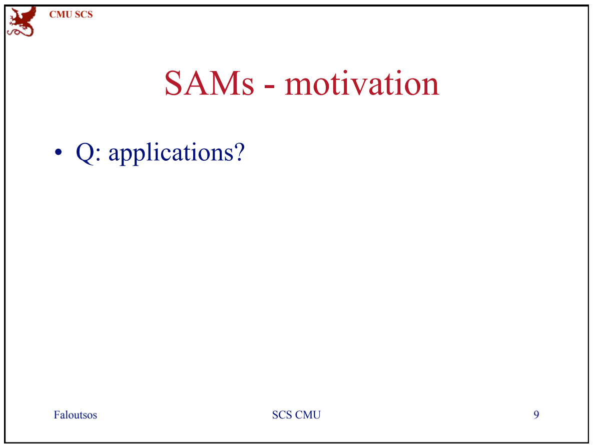
CMU SCS

Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' within ε)



Faloutsos SCS CMU 8

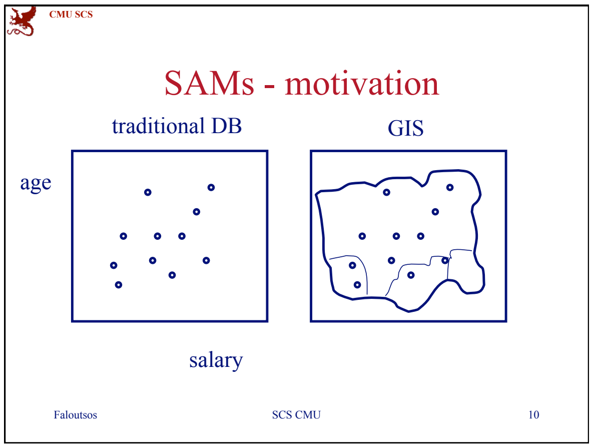


CMU SCS

SAMs - motivation

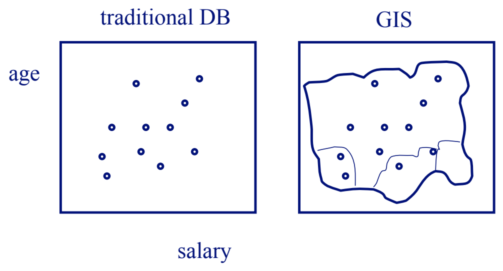
- Q: applications?

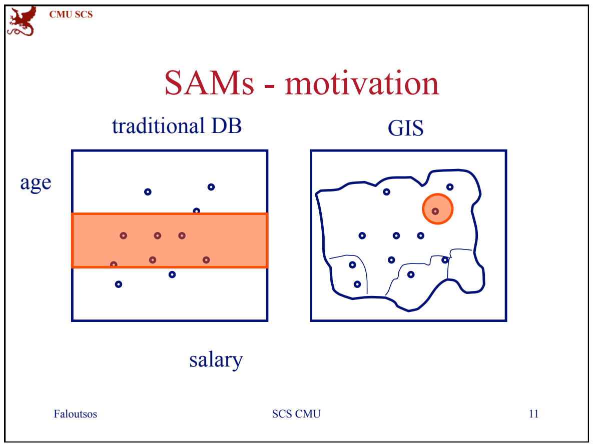
Faloutsos SCS CMU 9



CMU SCS

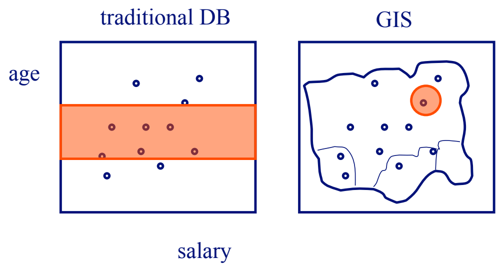
SAMs - motivation

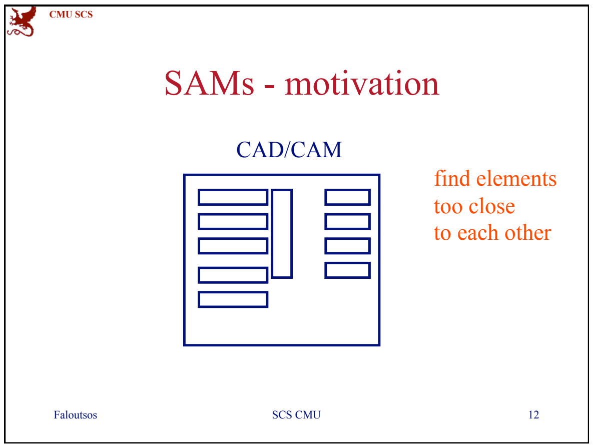




CMU SCS

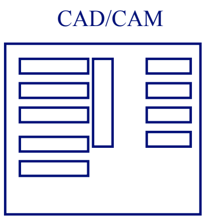
SAMs - motivation



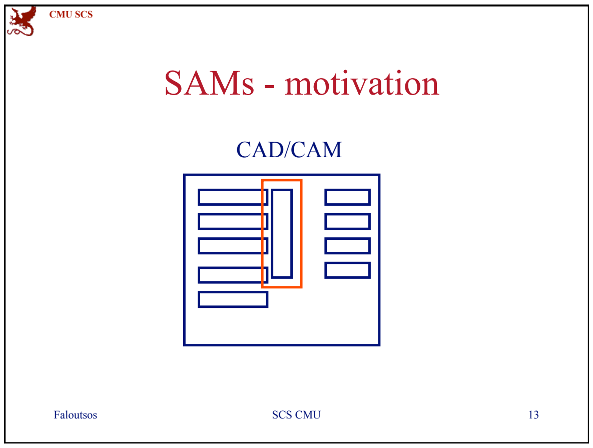


CMU SCS

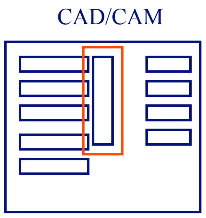
SAMs - motivation

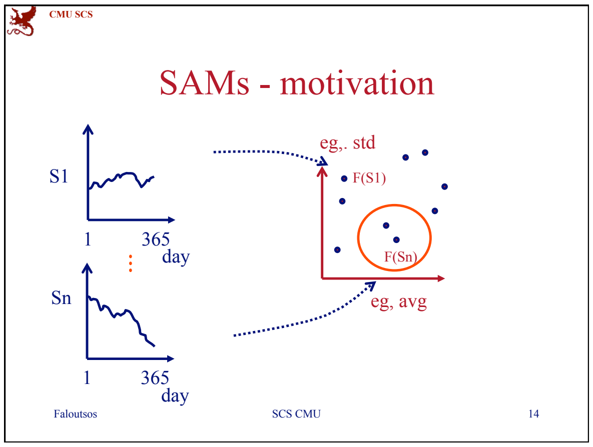


find elements
too close
to each other

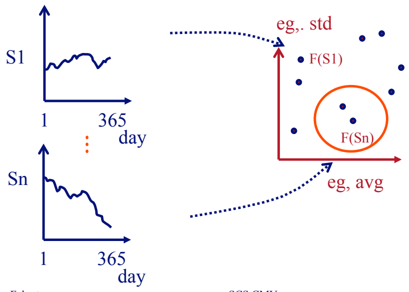


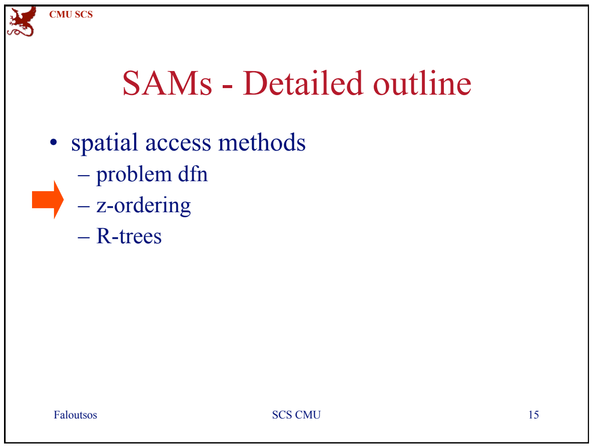
SAMs - motivation





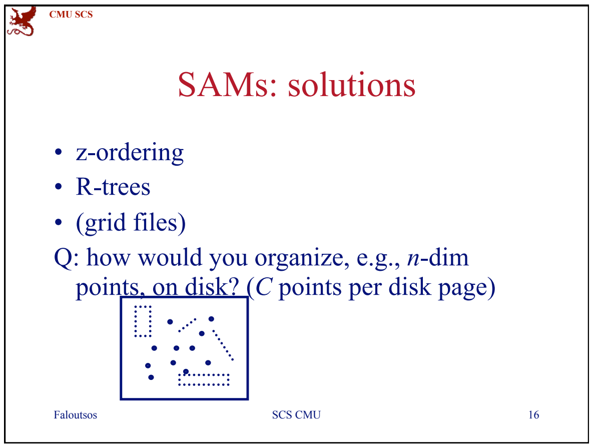
SAMs - motivation





SAMs - Detailed outline

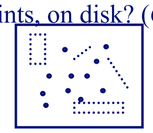
- spatial access methods
  - problem dfn
  - z-ordering
  - R-trees


CMU SCS

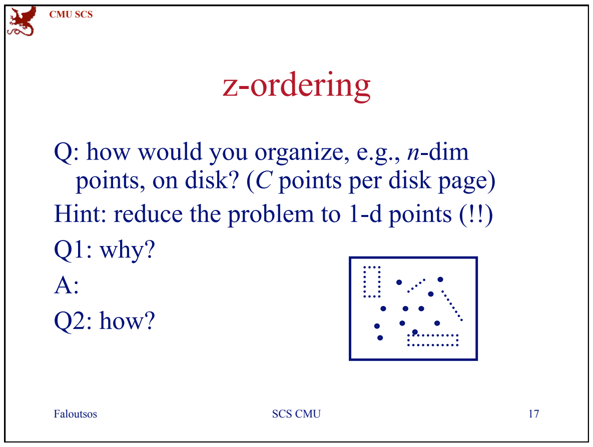
SAMs: solutions

- z-ordering
- R-trees
- (grid files)

Q: how would you organize, e.g., *n*-dim points, on disk? (*C* points per disk page)



Faloutsos SCS CMU 16


CMU SCS

z-ordering

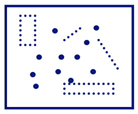
Q: how would you organize, e.g., *n*-dim points, on disk? (*C* points per disk page)

Hint: reduce the problem to 1-d points (!!)

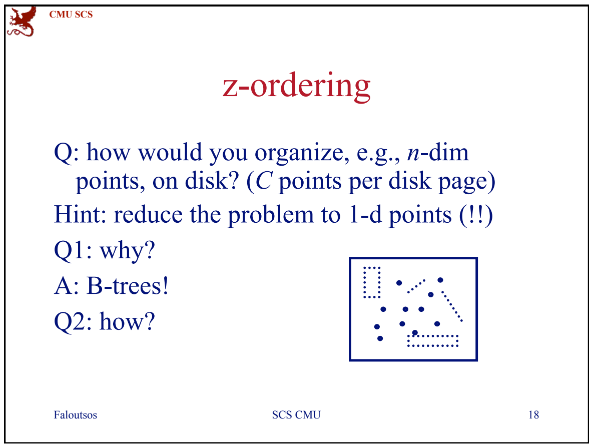
Q1: why?

A:

Q2: how?



Faloutsos SCS CMU 17


CMU SCS

z-ordering

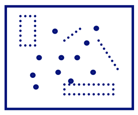
Q: how would you organize, e.g., *n*-dim points, on disk? (*C* points per disk page)

Hint: reduce the problem to 1-d points (!!)

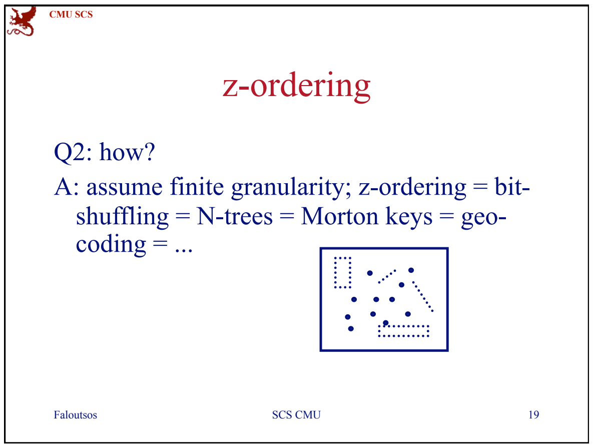
Q1: why?

A: B-trees!

Q2: how?



Faloutsos SCS CMU 18

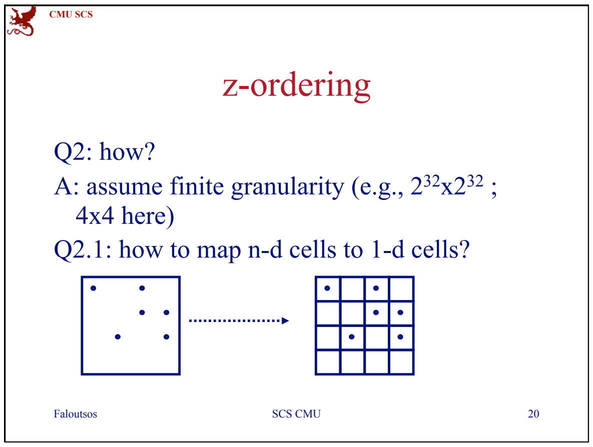

CMU SCS

z-ordering

Q2: how?

A: assume finite granularity; z-ordering = bit-shuffling = N-trees = Morton keys = geo-coding = ...

Faloutsos SCS CMU 19


CMU SCS

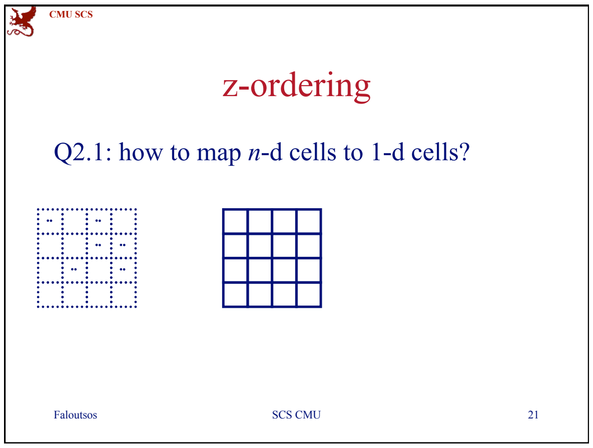
z-ordering

Q2: how?

A: assume finite granularity (e.g., 2^{32}x2^{32} ; 4x4 here)

Q2.1: how to map n-d cells to 1-d cells?

Faloutsos SCS CMU 20


CMU SCS

z-ordering

Q2.1: how to map *n*-d cells to 1-d cells?

Faloutsos SCS CMU 21

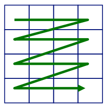
CMU SCS

z-ordering

Q2.1: how to map *n*-d cells to 1-d cells?

A: row-wise

Q: is it good?



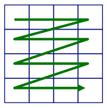
Faloutsos SCS CMU 22

CMU SCS

z-ordering

Q: is it good?

A: great for ‘x’ axis; bad for ‘y’ axis

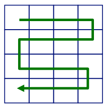


Faloutsos SCS CMU 23

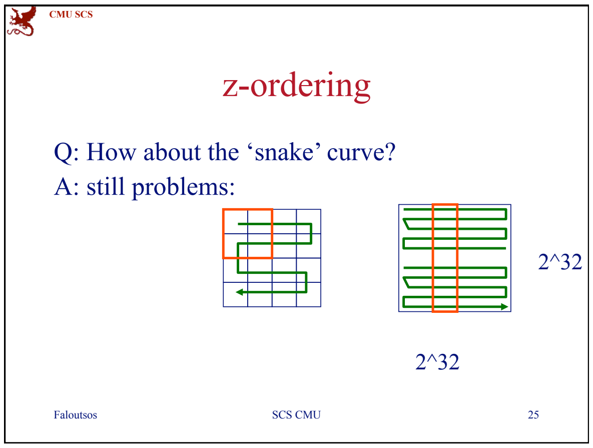
CMU SCS

z-ordering

Q: How about the ‘snake’ curve?



Faloutsos SCS CMU 24

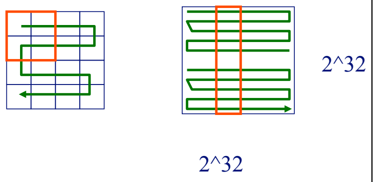


CMU SCS

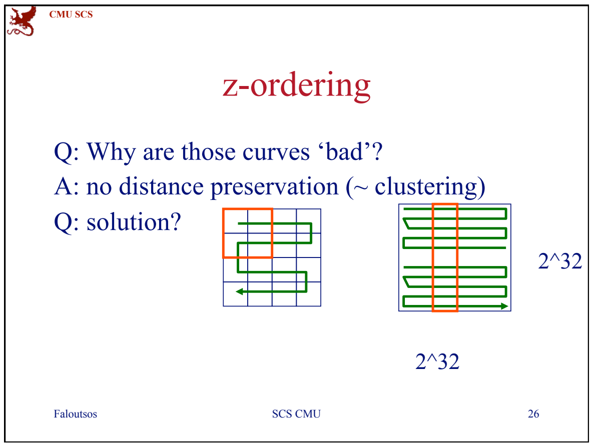
z-ordering

Q: How about the ‘snake’ curve?

A: still problems:



Faloutsos SCS CMU 25



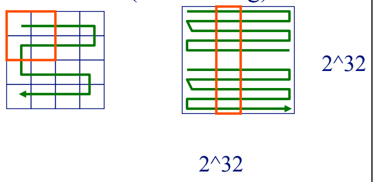
CMU SCS

z-ordering

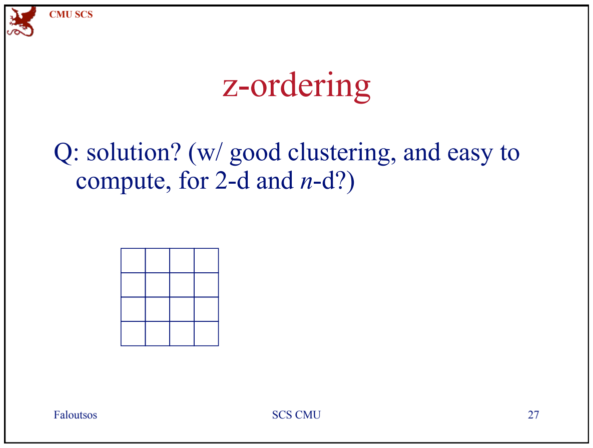
Q: Why are those curves ‘bad’?

A: no distance preservation (~ clustering)

Q: solution?



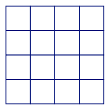
Faloutsos SCS CMU 26



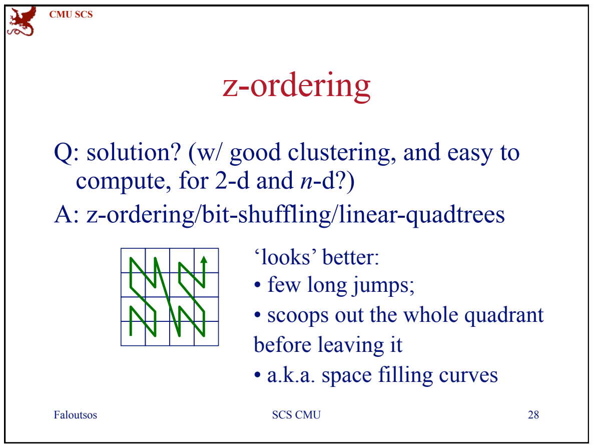
CMU SCS

z-ordering

Q: solution? (w/ good clustering, and easy to compute, for 2-d and *n*-d?)



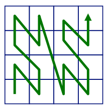
Faloutsos SCS CMU 27



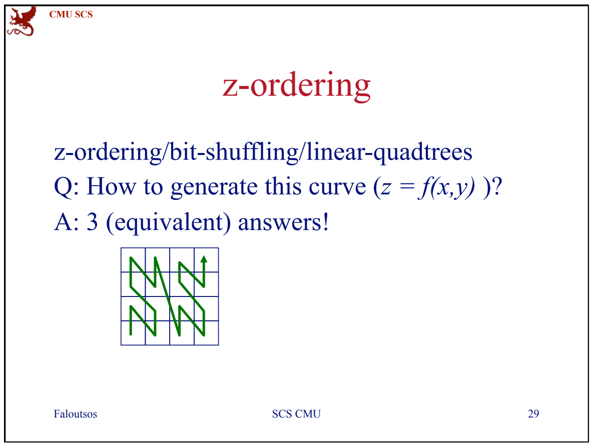
z-ordering

Q: solution? (w/ good clustering, and easy to compute, for 2-d and *n*-d?)

A: z-ordering/bit-shuffling/linear-quadtrees



'looks' better:
- few long jumps;
- scoops out the whole quadrant before leaving it
- a.k.a. space filling curves

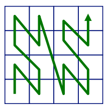


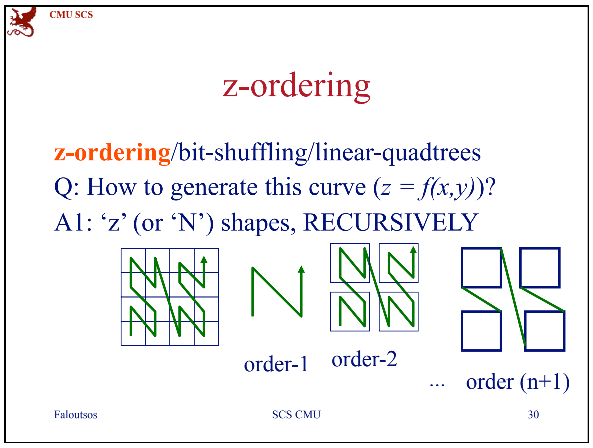
z-ordering

z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve (*z = f(x,y)*)?

A: 3 (equivalent) answers!



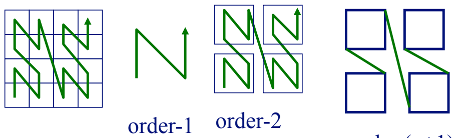


z-ordering

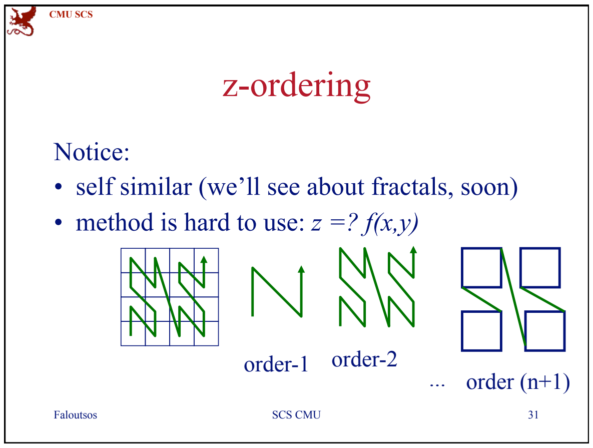
z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve (*z = f(x,y)*)?

A1: 'z' (or 'N') shapes, RECURSIVELY



order-1 order-2 ... order (n+1)



CMU SCS

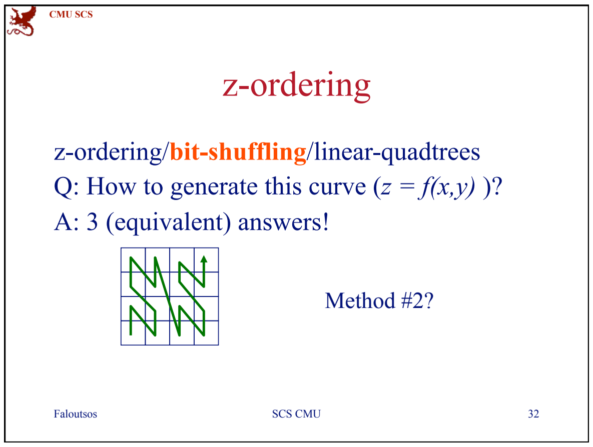
z-ordering

Notice:

- self similar (we'll see about fractals, soon)
- method is hard to use: *z =? f(x,y)*

order-1
order-2
...
order (n+1)

Faloutsos SCS CMU 31



CMU SCS

z-ordering

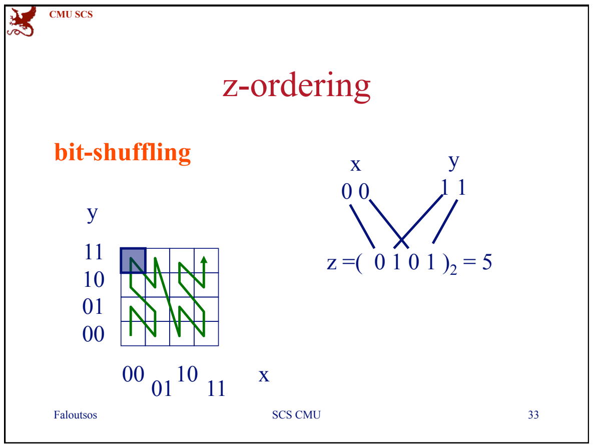
z-ordering/**bit-shuffling**/linear-quadtrees

Q: How to generate this curve (*z = f(x,y)*)?

A: 3 (equivalent) answers!

Method #2?

Faloutsos SCS CMU 32



CMU SCS

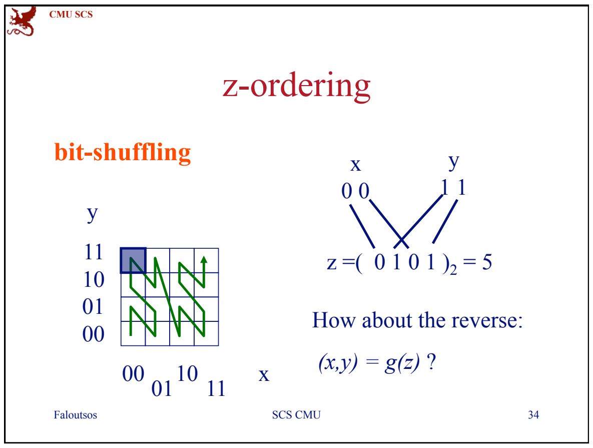
z-ordering

bit-shuffling

y
11
10
01
00
00
01
10
11
x
x
y
0 0
1 1

$z = (0\,1\,0\,1)_2 = 5$

Faloutsos SCS CMU 33


y
11
10
01
00
00
01
10
11
x

CMU SCS

z-ordering

bit-shuffling

x y

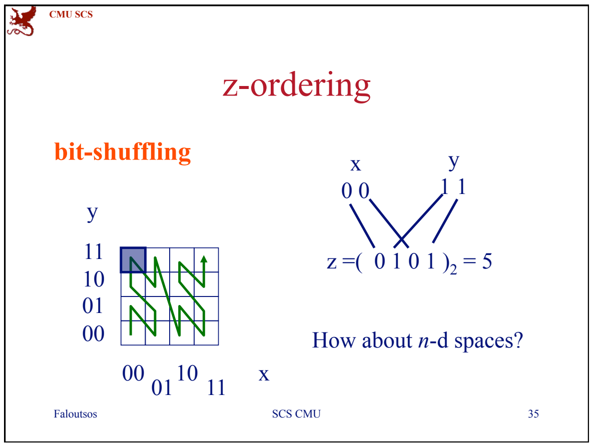
0 0 1 1

$z = (0\ 1\ 0\ 1)_2 = 5$

How about the reverse:

(x,y) = g(z) ?

Faloutsos SCS CMU 34


y
11
10
01
00
00
01
10
11
x

CMU SCS

z-ordering

bit-shuffling

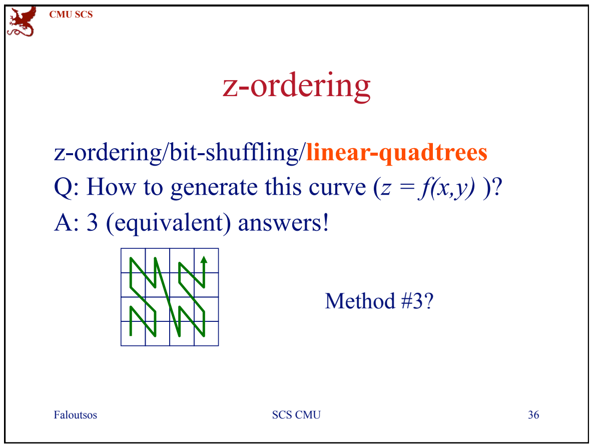
x y

0 0 1 1

$z = (0\ 1\ 0\ 1)_2 = 5$

How about *n*-d spaces?

Faloutsos SCS CMU 35



CMU SCS

z-ordering

z-ordering/bit-shuffling/**linear-quadtrees**

Q: How to generate this curve (*z = f(x,y)*)?

A: 3 (equivalent) answers!

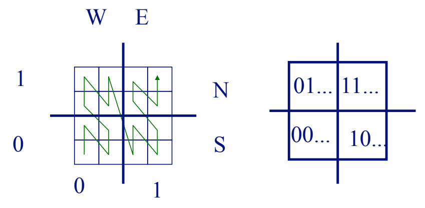
Method #3?

Faloutsos SCS CMU 36

CMU SCS

z-ordering

linear-quadtrees : assign N->1, S->0 e.t.c.

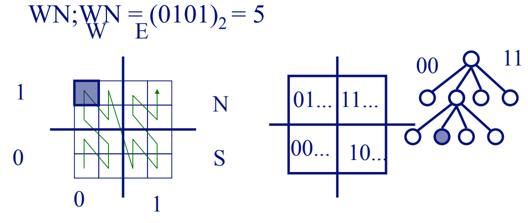


Faloutsos SCS CMU 37

CMU SCS

z-ordering

... and repeat recursively. Eg.: $z_{blue\text{-}cell}$ = WN;WN = $(0101)_2$ = 5

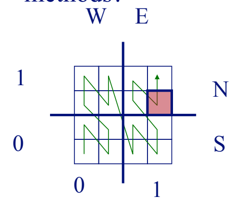


Faloutsos SCS CMU 38

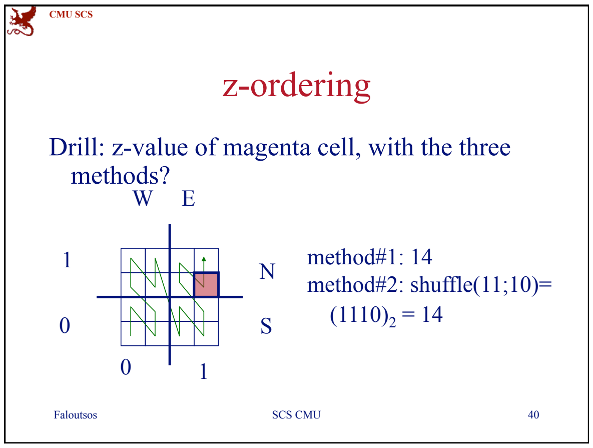
CMU SCS

z-ordering

Drill: z-value of magenta cell, with the three methods?

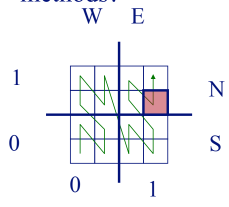


Faloutsos SCS CMU 39

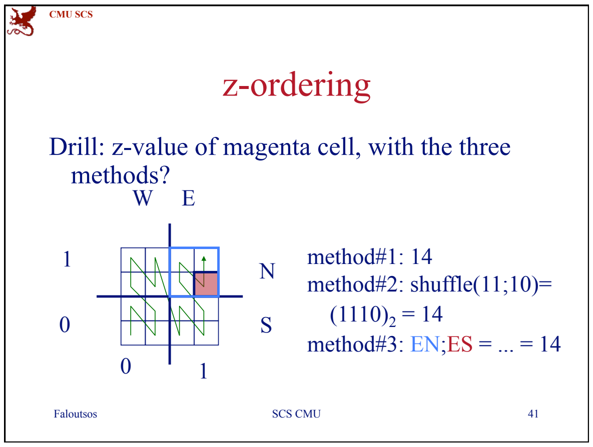


z-ordering

Drill: z-value of magenta cell, with the three methods?

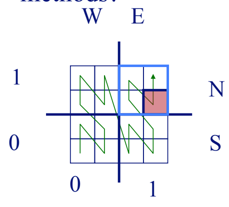


method#1: 14
method#2: shuffle(11;10)= $(1110)_2$ = 14

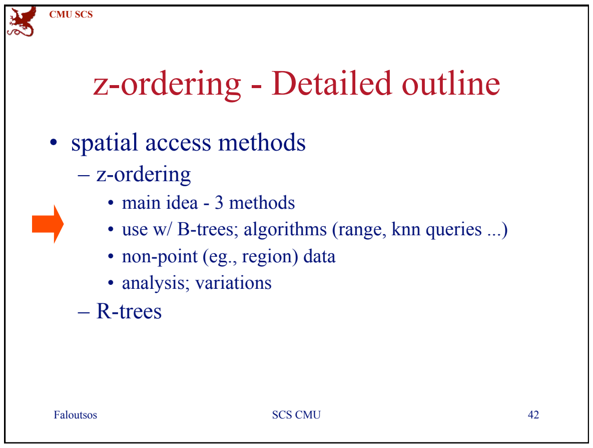


z-ordering

Drill: z-value of magenta cell, with the three methods?

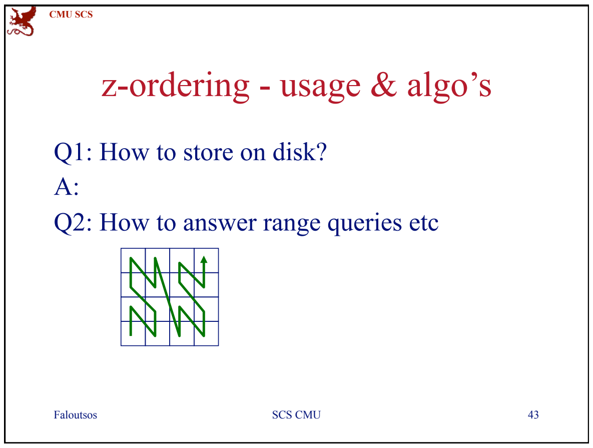


method#1: 14
method#2: shuffle(11;10)= $(1110)_2$ = 14
method#3: EN;ES = ... = 14



z-ordering - Detailed outline

- spatial access methods
  - z-ordering
    - main idea - 3 methods
    - use w/ B-trees; algorithms (range, knn queries ...)
    - non-point (eg., region) data
    - analysis; variations
  - R-trees



CMU SCS

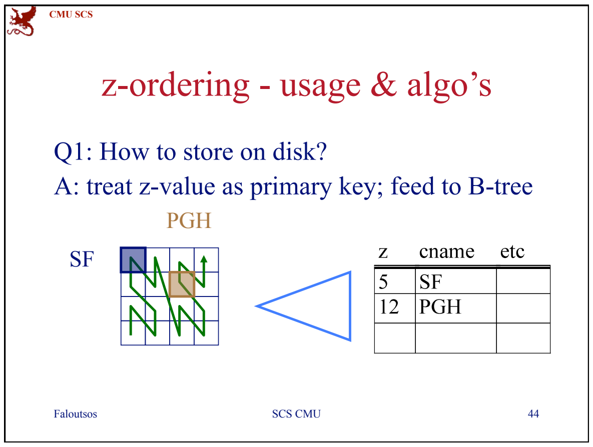
z-ordering - usage & algo's

Q1: How to store on disk?

A:

Q2: How to answer range queries etc

Faloutsos SCS CMU 43



CMU SCS

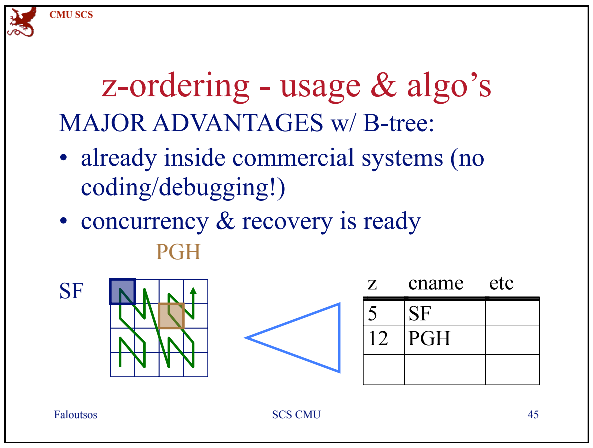
z-ordering - usage & algo's

Q1: How to store on disk?

A: treat z-value as primary key; feed to B-tree

z	cname	etc
5	SF	
12	PGH	

Faloutsos SCS CMU 44



CMU SCS

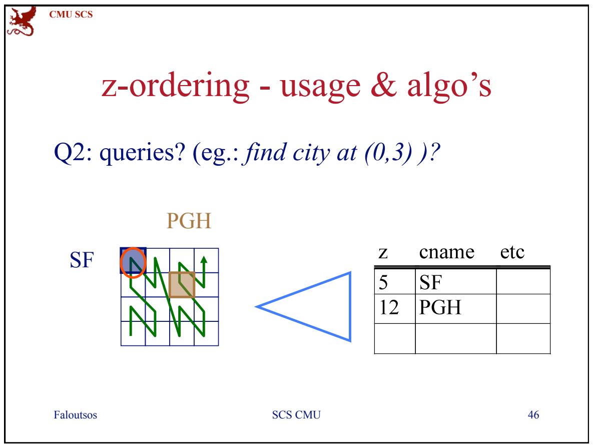
z-ordering - usage & algo's

MAJOR ADVANTAGES w/ B-tree:

- already inside commercial systems (no coding/debugging!)
- concurrency & recovery is ready

z	cname	etc
5	SF	
12	PGH	

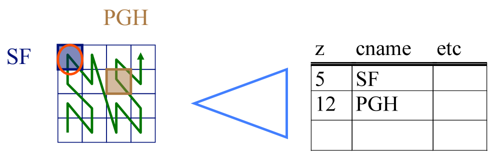
Faloutsos SCS CMU 45



CMU SCS

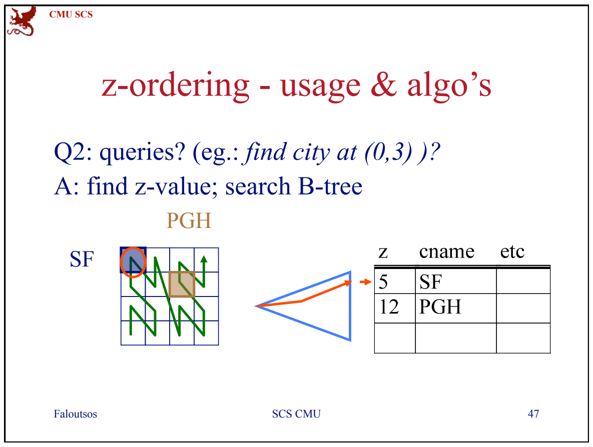
z-ordering - usage & algo's

Q2: queries? (eg.: *find city at (0,3))?*



z	cname	etc
5	SF	
12	PGH	

Faloutsos SCS CMU 46

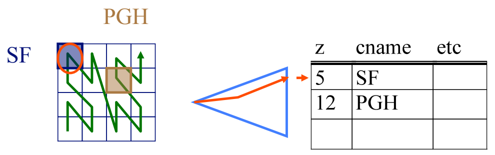


CMU SCS

z-ordering - usage & algo's

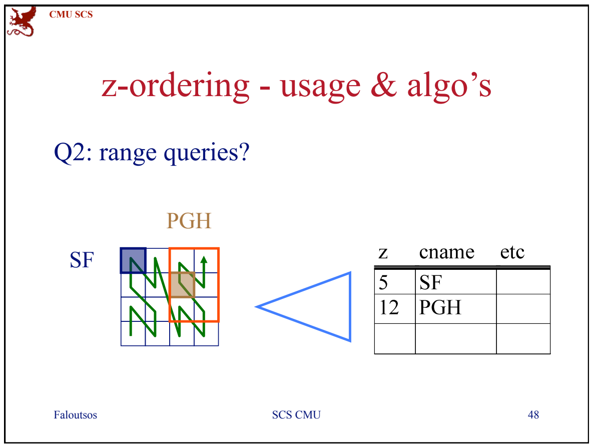
Q2: queries? (eg.: *find city at (0,3))?*

A: find z-value; search B-tree



z	cname	etc
5	SF	
12	PGH	

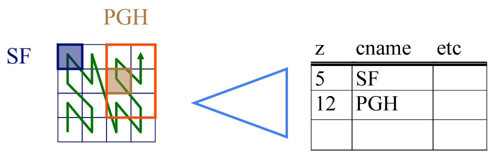
Faloutsos SCS CMU 47



CMU SCS

z-ordering - usage & algo's

Q2: range queries?



z	cname	etc
5	SF	
12	PGH	

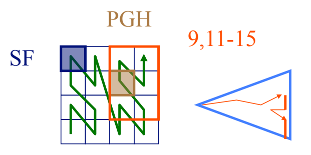
Faloutsos SCS CMU 48

CMU SCS

z-ordering - usage & algo’s

Q2: range queries?

A: compute ranges of z-values; use B-tree



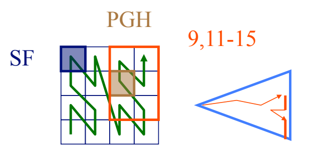
z	cname	etc
5	SF	
12	PGH	

Faloutsos SCS CMU 49

CMU SCS

z-ordering - usage & algo’s

Q2’: range queries - how to reduce # of qualifying of ranges?



z	cname	etc
5	SF	
12	PGH	

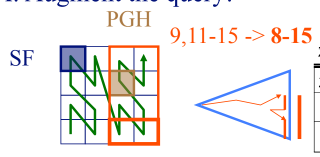
Faloutsos SCS CMU 50

CMU SCS

z-ordering - usage & algo’s

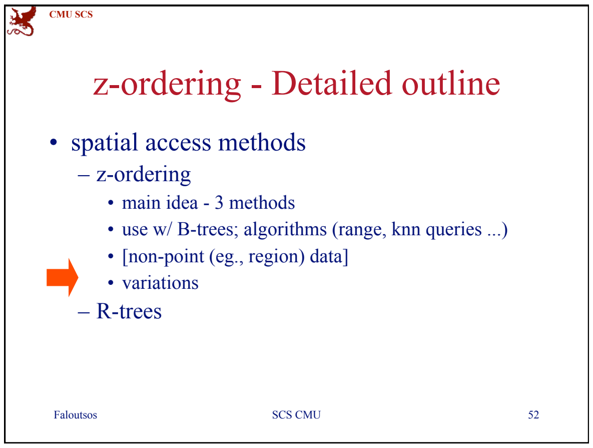
Q2’: range queries - how to reduce # of qualifying of ranges?

A: Augment the query!



z	cname	etc
5	SF	
12	PGH	

Faloutsos SCS CMU 51

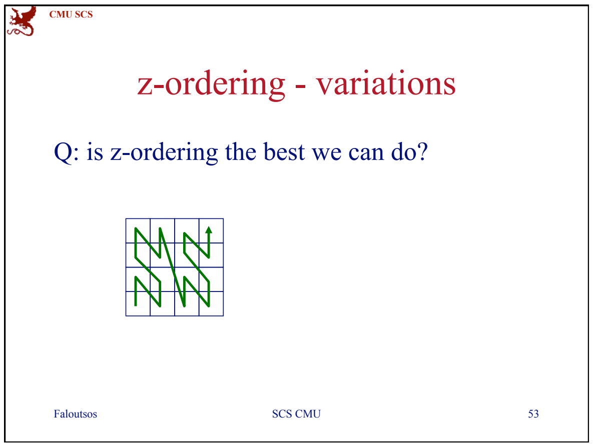


CMU SCS

z-ordering - Detailed outline

- spatial access methods
  - z-ordering
    - main idea - 3 methods
    - use w/ B-trees; algorithms (range, knn queries ...)
    - [non-point (eg., region) data]
    - variations
  - R-trees

Faloutsos SCS CMU 52

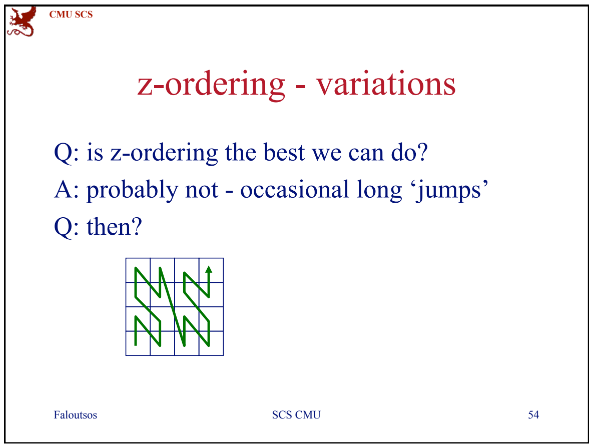


CMU SCS

z-ordering - variations

Q: is z-ordering the best we can do?

Faloutsos SCS CMU 53



CMU SCS

z-ordering - variations

Q: is z-ordering the best we can do?

A: probably not - occasional long ‘jumps’

Q: then?

Faloutsos SCS CMU 54

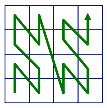
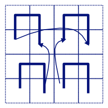
CMU SCS

z-ordering - variations

Q: is z-ordering the best we can do?

A: probably not - occasional long 'jumps'

Q: then? A1: Gray codes

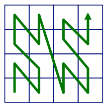
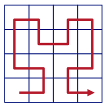



Faloutsos SCS CMU 55

CMU SCS

z-ordering - variations

A2: Hilbert curve! (a.k.a. Hilbert-Peano curve)

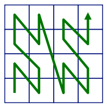
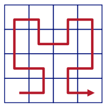



Faloutsos SCS CMU 56

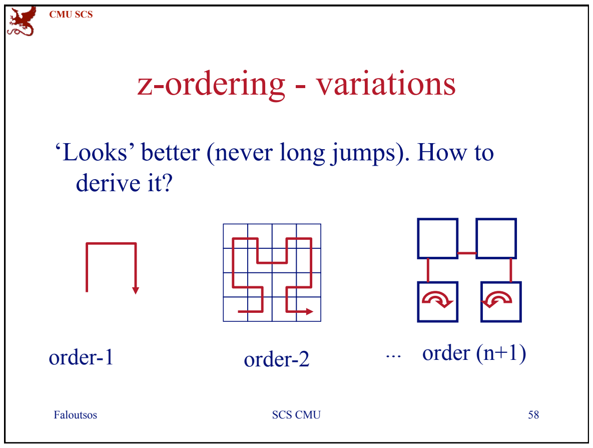
CMU SCS

z-ordering - variations

'Looks' better (never long jumps). How to derive it?

Faloutsos SCS CMU 57



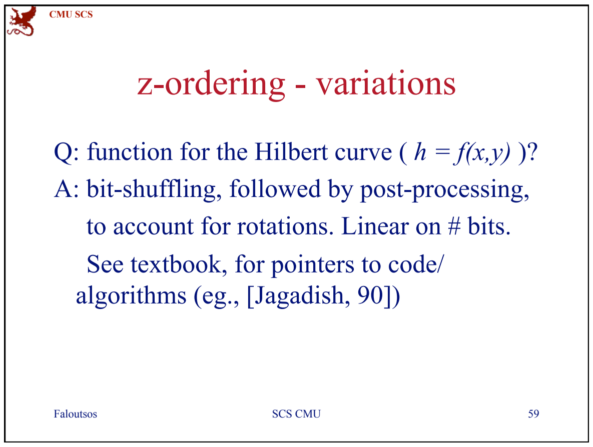
CMU SCS

z-ordering - variations

‘Looks’ better (never long jumps). How to derive it?

order-1 order-2 … order (n+1)

Faloutsos SCS CMU 58



CMU SCS

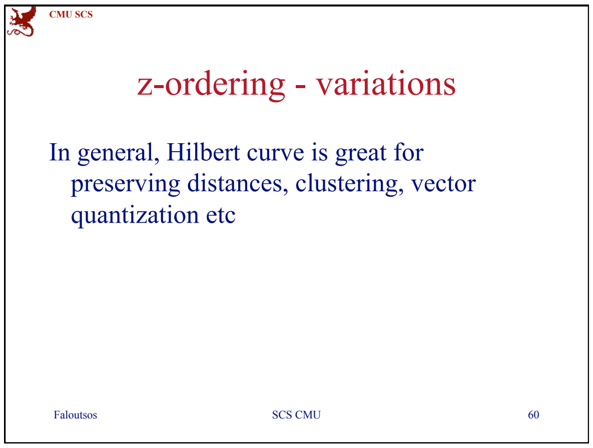
z-ordering - variations

Q: function for the Hilbert curve (*h = f(x,y)*)?

A: bit-shuffling, followed by post-processing, to account for rotations. Linear on # bits.

See textbook, for pointers to code/ algorithms (eg., [Jagadish, 90])

Faloutsos SCS CMU 59

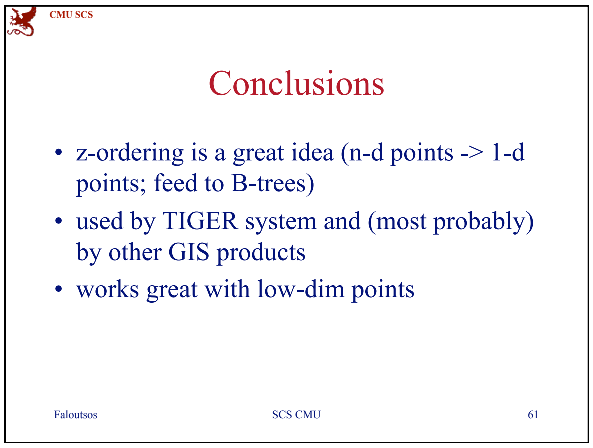


CMU SCS

z-ordering - variations

In general, Hilbert curve is great for preserving distances, clustering, vector quantization etc

Faloutsos SCS CMU 60

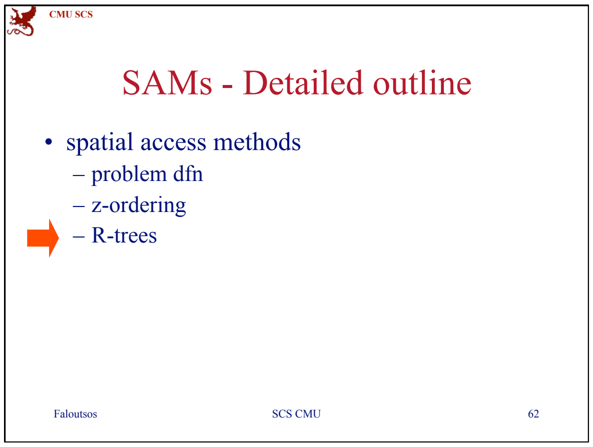


CMU SCS

Conclusions

- z-ordering is a great idea (n-d points -> 1-d points; feed to B-trees)
- used by TIGER system and (most probably) by other GIS products
- works great with low-dim points

Faloutsos SCS CMU 61

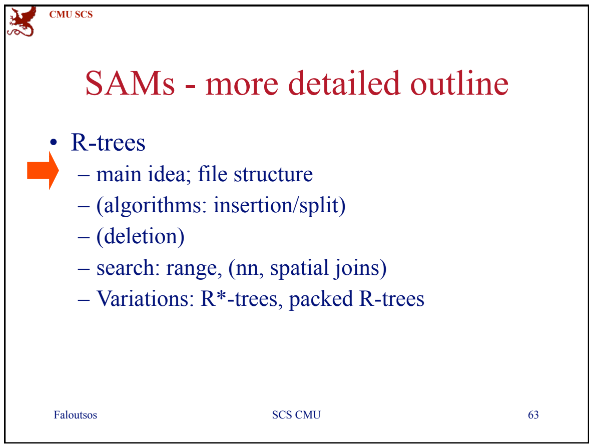


CMU SCS

SAMs - Detailed outline

- spatial access methods
  – problem dfn
  – z-ordering
  – R-trees

Faloutsos SCS CMU 62

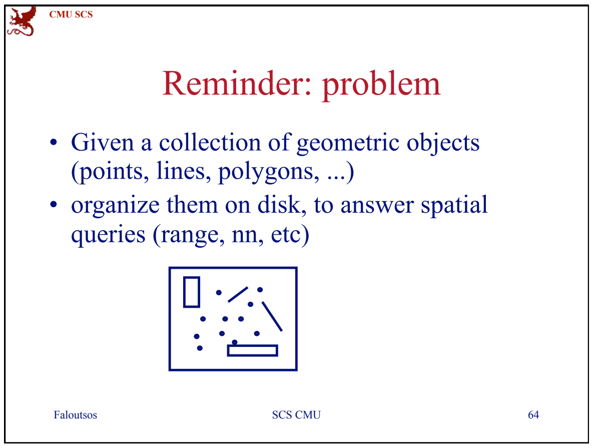


CMU SCS

SAMs - more detailed outline

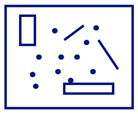
- R-trees
  – main idea; file structure
  – (algorithms: insertion/split)
  – (deletion)
  – search: range, (nn, spatial joins)
  – Variations: R*-trees, packed R-trees

Faloutsos SCS CMU 63

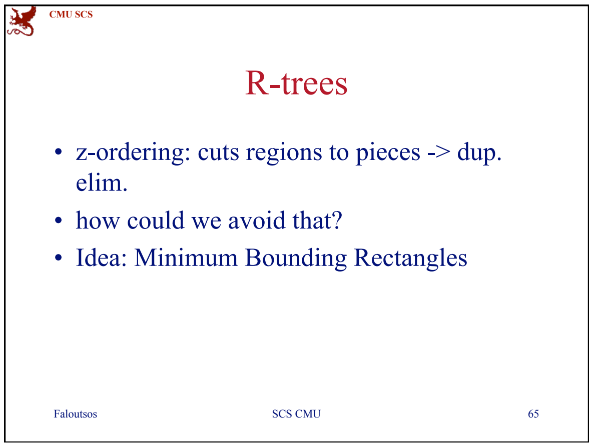

CMU SCS

Reminder: problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer spatial queries (range, nn, etc)



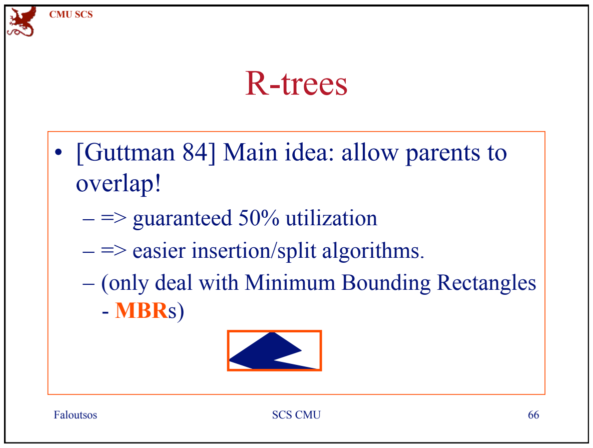
Faloutsos SCS CMU 64


CMU SCS

R-trees

- z-ordering: cuts regions to pieces -> dup. elim.
- how could we avoid that?
- Idea: Minimum Bounding Rectangles

Faloutsos SCS CMU 65


CMU SCS

R-trees

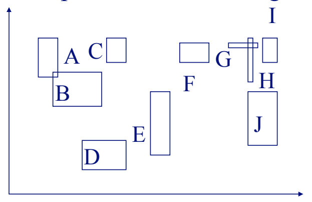
- [Guttman 84] Main idea: allow parents to overlap!
  - => guaranteed 50% utilization
  - => easier insertion/split algorithms.
  - (only deal with Minimum Bounding Rectangles - **MBR**s)

Faloutsos SCS CMU 66

CMU SCS

R-trees

- eg., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page

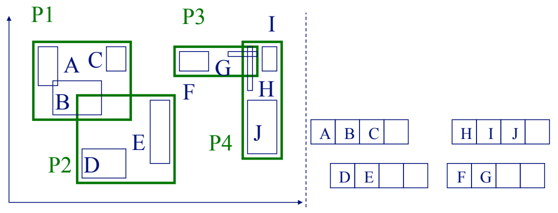


Faloutsos SCS CMU 67

CMU SCS

R-trees

- eg., w/ fanout 4:

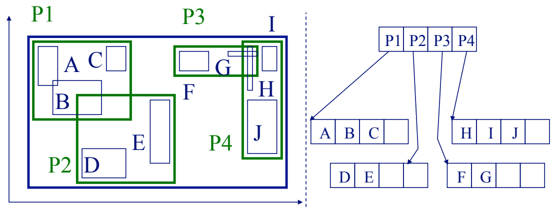


Faloutsos SCS CMU 68

CMU SCS

R-trees

- eg., w/ fanout 4:

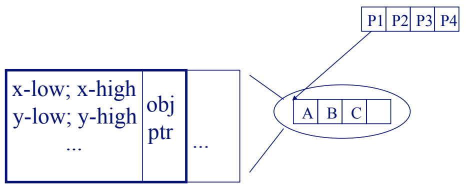


Faloutsos SCS CMU 69

CMU SCS

R-trees - format of nodes

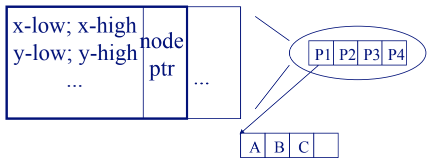
- {(MBR; obj-ptr)} for leaf nodes



CMU SCS

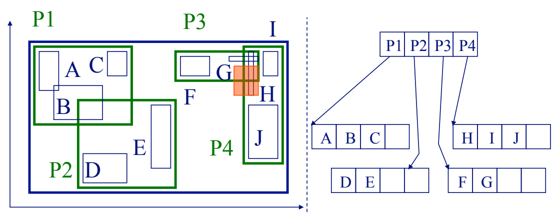
R-trees - format of nodes

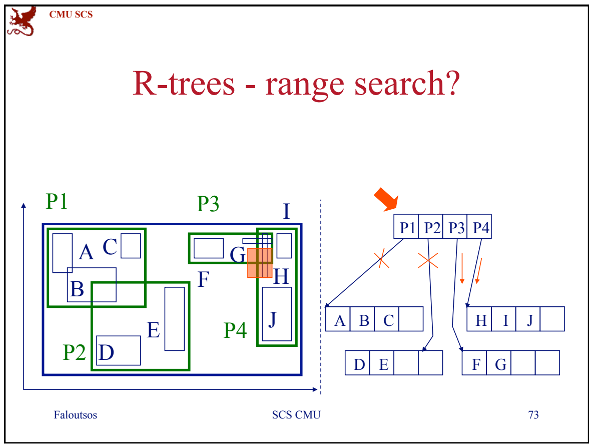
- {(MBR; node-ptr)} for non-leaf nodes



CMU SCS

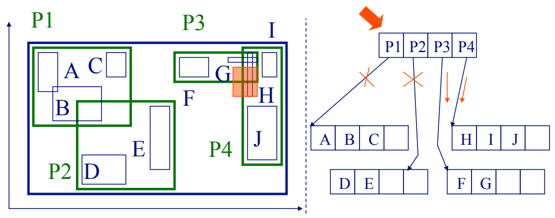
R-trees - range search?



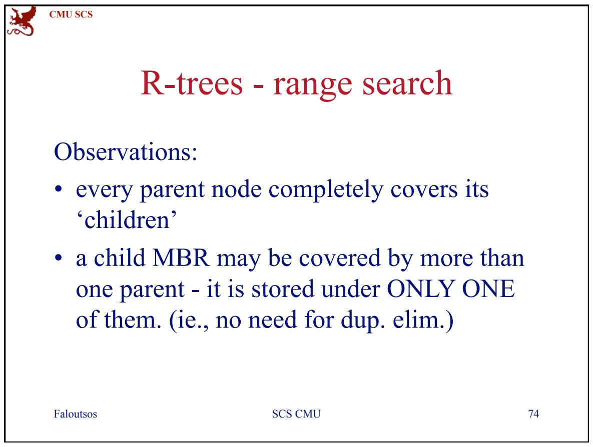


CMU SCS

R-trees - range search?



Faloutsos SCS CMU 73



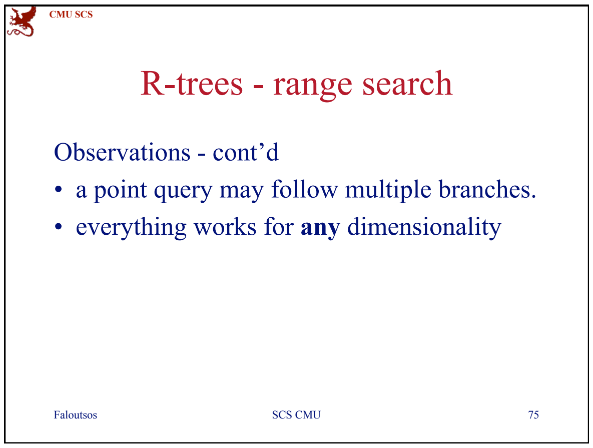
CMU SCS

R-trees - range search

Observations:

- every parent node completely covers its 'children'
- a child MBR may be covered by more than one parent - it is stored under ONLY ONE of them. (ie., no need for dup. elim.)

Faloutsos SCS CMU 74



CMU SCS

R-trees - range search

Observations - cont'd

- a point query may follow multiple branches.
- everything works for **any** dimensionality

Faloutsos SCS CMU 75

CMU SCS

SAMs - more detailed outline

- R-trees
  - main idea; file structure
  - (algorithms: insertion/split)
  - (deletion)
  - search: range, (nn, spatial joins)
  - Variations: R*-trees, packed R-trees

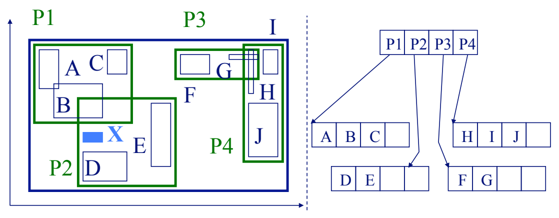
Faloutsos SCS CMU 76

CMU SCS

SKIP

R-trees - insertion

- eg., rectangle ‘X’



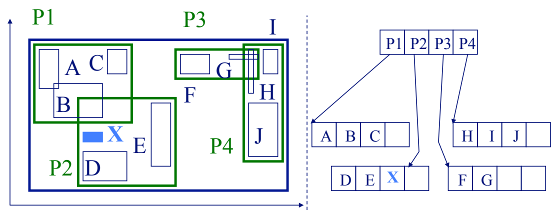
Faloutsos SCS CMU 77

CMU SCS

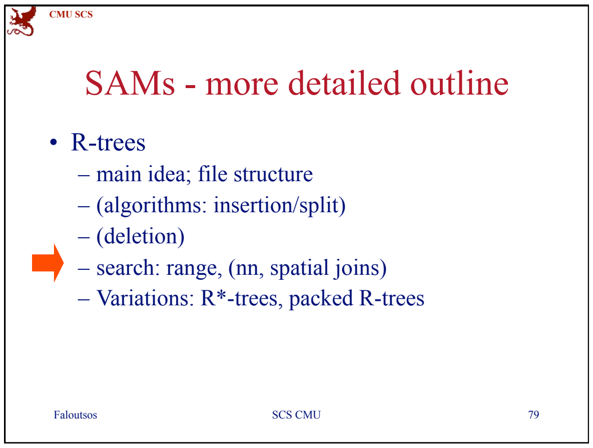
SKIP

R-trees - insertion

- eg., rectangle ‘X’



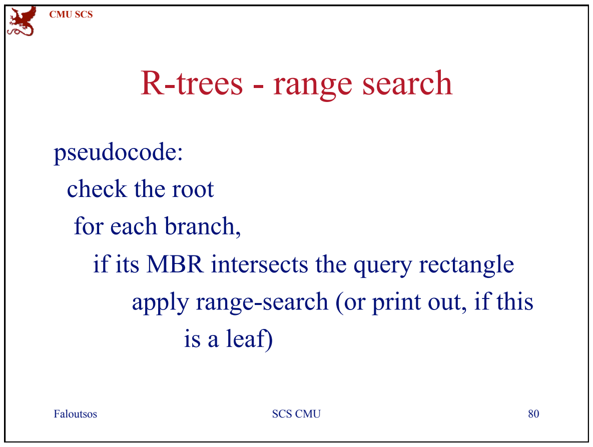
Faloutsos SCS CMU 78


CMU SCS

SAMs - more detailed outline

- R-trees
  - main idea; file structure
  - (algorithms: insertion/split)
  - (deletion)
  - search: range, (nn, spatial joins)
  - Variations: R*-trees, packed R-trees

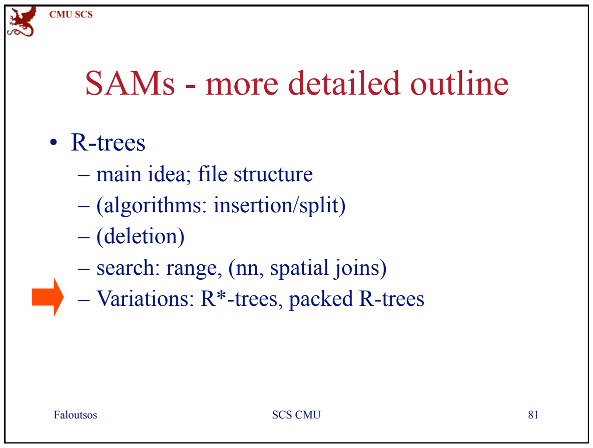
Faloutsos SCS CMU 79


CMU SCS

R-trees - range search

```
pseudocode:
  check the root
   for each branch,
      if its MBR intersects the query rectangle
            apply range-search (or print out, if this
                    is a leaf)
```

Faloutsos SCS CMU 80


CMU SCS

SAMs - more detailed outline

- R-trees
  - main idea; file structure
  - (algorithms: insertion/split)
  - (deletion)
  - search: range, (nn, spatial joins)
  - Variations: R*-trees, packed R-trees

Faloutsos SCS CMU 81

CMU SCS

SKIP

R-trees - variations

Guttman's R-trees sparked **much** follow-up work

- can we do better splits?
- what about static datasets (no ins/del/upd)?
- what about other bounding shapes?

Faloutsos SCS CMU 82

CMU SCS

SKIP

R-trees - variations

Guttman's R-trees sparked much follow-up work

- can we do better splits?
  - i.e, defer splits?

Faloutsos SCS CMU 83

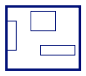
CMU SCS

SKIP

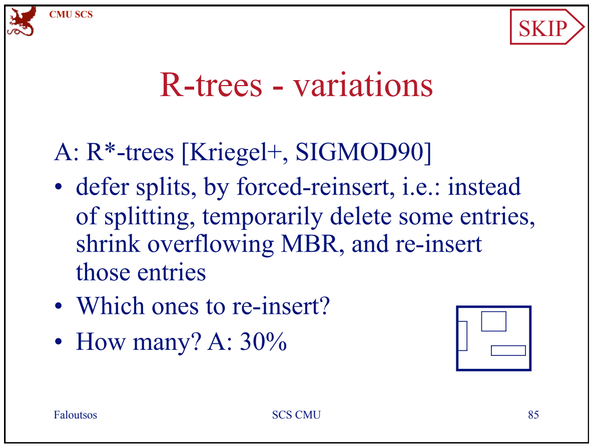
R-trees - variations

A: R*-trees [Kriegel+, SIGMOD90]

- defer splits, by forced-reinsert, i.e.: instead of splitting, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries
- Which ones to re-insert?
- How many?



Faloutsos SCS CMU 84



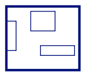
CMU SCS

SKIP

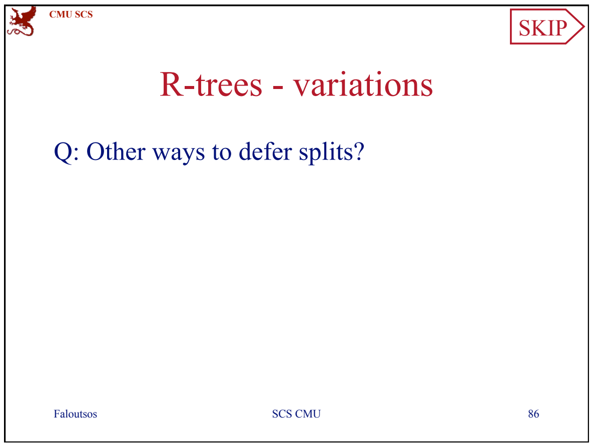
R-trees - variations

A: R*-trees [Kriegel+, SIGMOD90]

- defer splits, by forced-reinsert, i.e.: instead of splitting, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries
- Which ones to re-insert?
- How many? A: 30%



Faloutsos SCS CMU 85



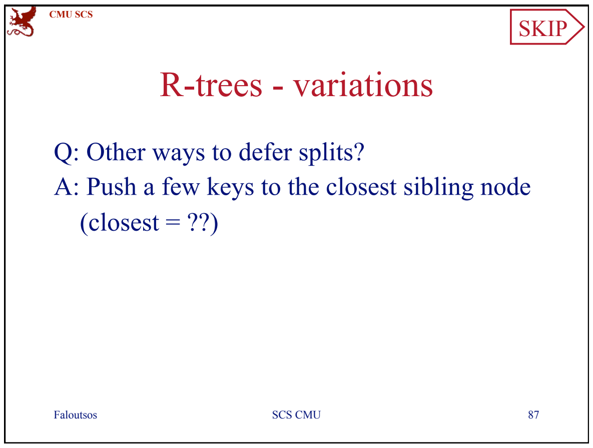
CMU SCS

SKIP

R-trees - variations

Q: Other ways to defer splits?

Faloutsos SCS CMU 86



CMU SCS

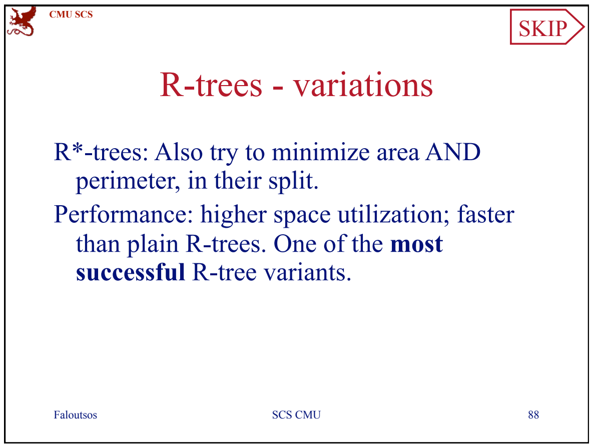
SKIP

R-trees - variations

Q: Other ways to defer splits?

A: Push a few keys to the closest sibling node (closest = ??)

Faloutsos SCS CMU 87



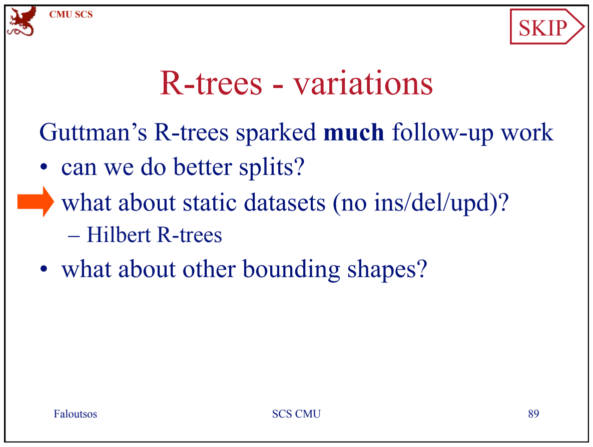
CMU SCS

SKIP

R-trees - variations

R*-trees: Also try to minimize area AND perimeter, in their split.

Performance: higher space utilization; faster than plain R-trees. One of the **most successful** R-tree variants.



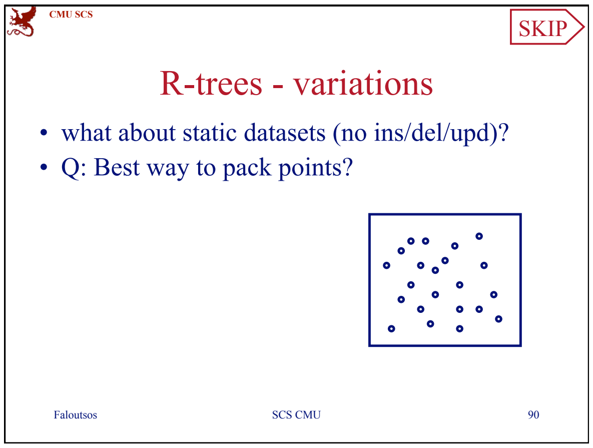
CMU SCS

SKIP

R-trees - variations

Guttman's R-trees sparked **much** follow-up work

- can we do better splits?
- ➡ what about static datasets (no ins/del/upd)?
  - Hilbert R-trees
- what about other bounding shapes?

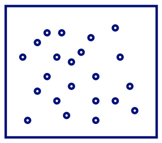


CMU SCS

SKIP

R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?

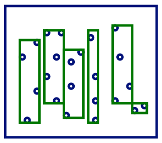


CMU SCS

SKIP

R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep
  great for queries on ‘x’;
  terrible for ‘y’

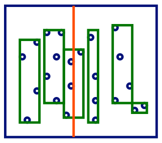


CMU SCS

SKIP

R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep
  great for queries on ‘x’;
  bad for ‘y’

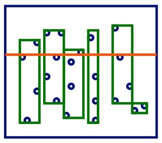


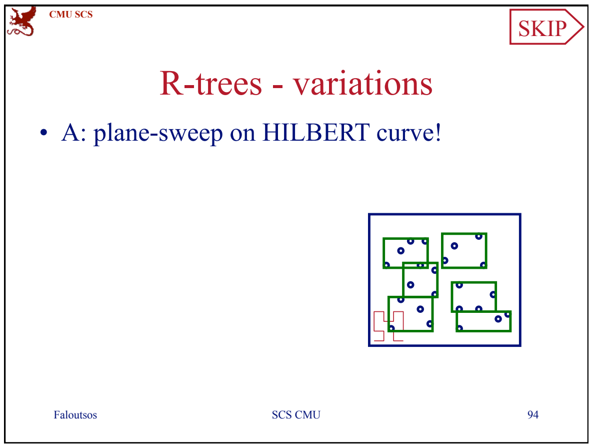
CMU SCS

SKIP

R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep
  great for queries on ‘x’;
  terrible for ‘y’
- Q: how to improve?



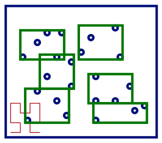


CMU SCS

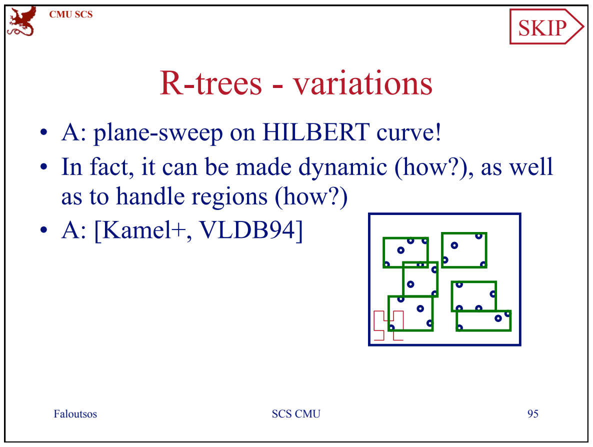
SKIP

R-trees - variations

- A: plane-sweep on HILBERT curve!



Faloutsos SCS CMU 94

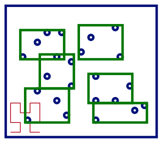


CMU SCS

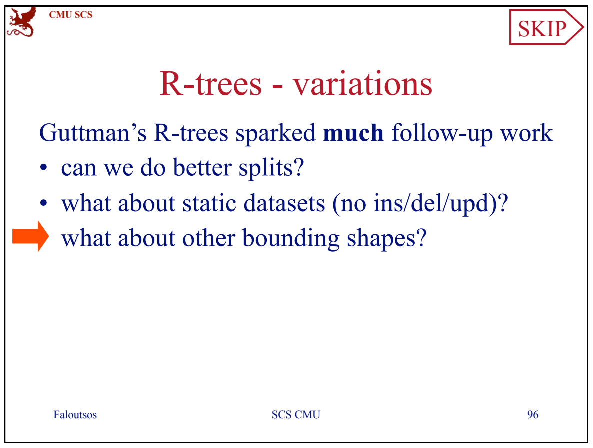
SKIP

R-trees - variations

- A: plane-sweep on HILBERT curve!
- In fact, it can be made dynamic (how?), as well as to handle regions (how?)
- A: [Kamel+, VLDB94]



Faloutsos SCS CMU 95



CMU SCS

SKIP

R-trees - variations

Guttman's R-trees sparked **much** follow-up work

- can we do better splits?
- what about static datasets (no ins/del/upd)?
- ➡ what about other bounding shapes?

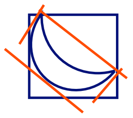
Faloutsos SCS CMU 96

CMU SCS

SKIP

R-trees - variations

- what about other bounding shapes? (and why?)
- A1: arbitrary-orientation lines (cell-tree, [Guenther]
- A2: P-trees (polygon trees) (MB polygon: 0, 90, 45, 135 degree lines)



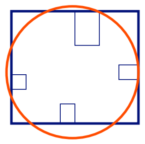
Faloutsos SCS CMU 97

CMU SCS

SKIP

R-trees - variations

- A3: L-shapes; holes (hB-tree)
- A4: TV-trees [Lin+, VLDB-Journal 1994]
- A5: SR-trees [Katayama+, SIGMOD97] (used in Informedia)



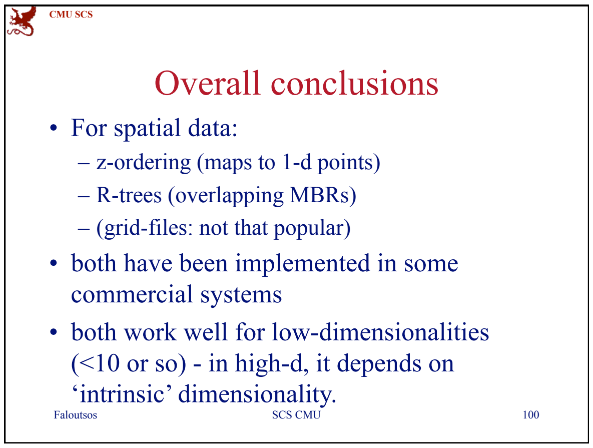
Faloutsos SCS CMU 98

CMU SCS

R-trees - conclusions

- Popular method; like multi-d B-trees
- guaranteed utilization
- good search times (for low-dim. at least)
- R*-, Hilbert- and SR-trees: still used
- Informix/DB2 ships DataBlade with R-trees
  - Also in postgres and sqlite3 (separate module)

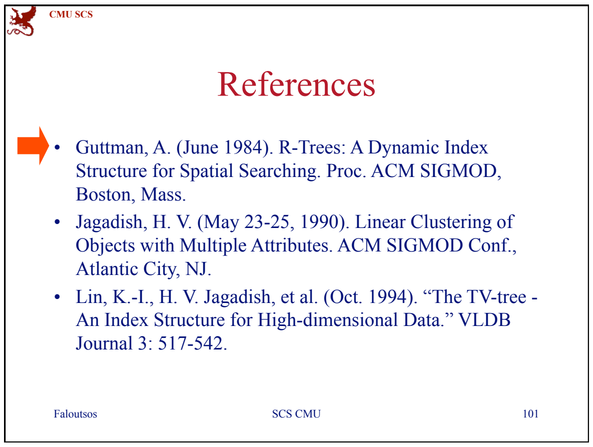
Faloutsos SCS CMU 99



CMU SCS

Overall conclusions

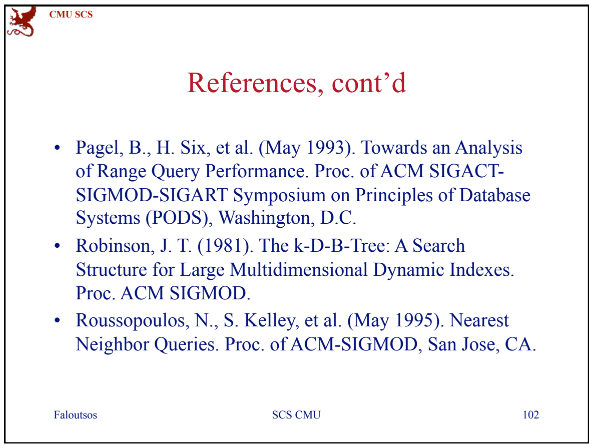
- For spatial data:
  - z-ordering (maps to 1-d points)
  - R-trees (overlapping MBRs)
  - (grid-files: not that popular)
- both have been implemented in some commercial systems
- both work well for low-dimensionalities (<10 or so) - in high-d, it depends on 'intrinsic' dimensionality.



CMU SCS

References

- Guttman, A. (June 1984). R-Trees: A Dynamic Index Structure for Spatial Searching. Proc. ACM SIGMOD, Boston, Mass.
- Jagadish, H. V. (May 23-25, 1990). Linear Clustering of Objects with Multiple Attributes. ACM SIGMOD Conf., Atlantic City, NJ.
- Lin, K.-I., H. V. Jagadish, et al. (Oct. 1994). "The TV-tree - An Index Structure for High-dimensional Data." VLDB Journal 3: 517-542.



CMU SCS

References, cont'd

- Pagel, B., H. Six, et al. (May 1993). Towards an Analysis of Range Query Performance. Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Washington, D.C.
- Robinson, J. T. (1981). The k-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. Proc. ACM SIGMOD.
- Roussopoulos, N., S. Kelley, et al. (May 1995). Nearest Neighbor Queries. Proc. of ACM-SIGMOD, San Jose, CA.