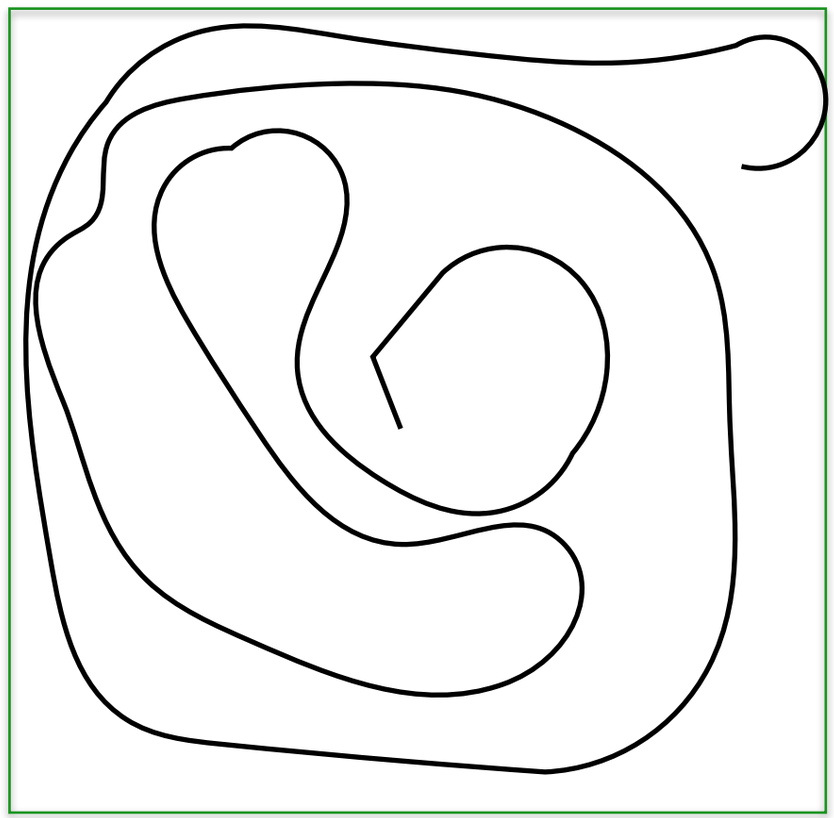
Space Filling Curves

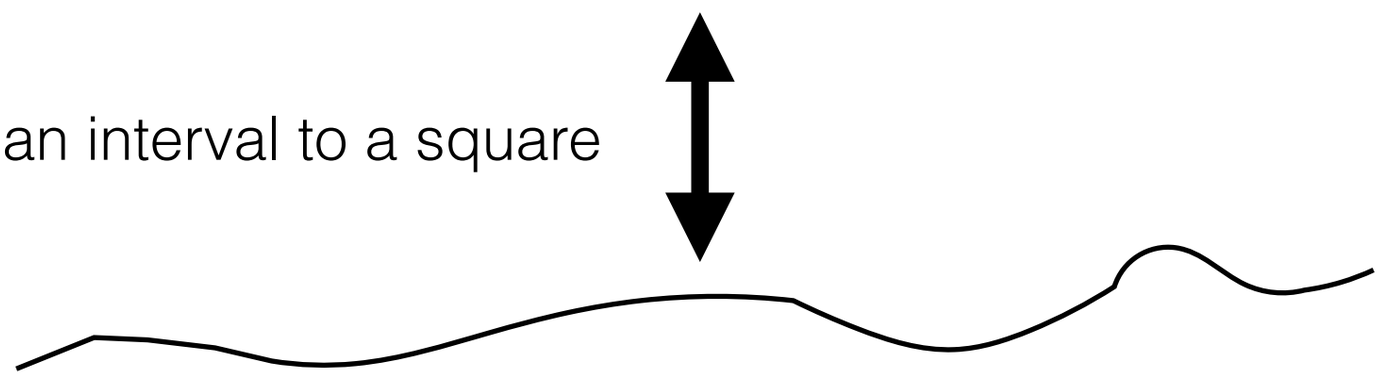
Laura Toma

Bowdoin College

Space filling curves

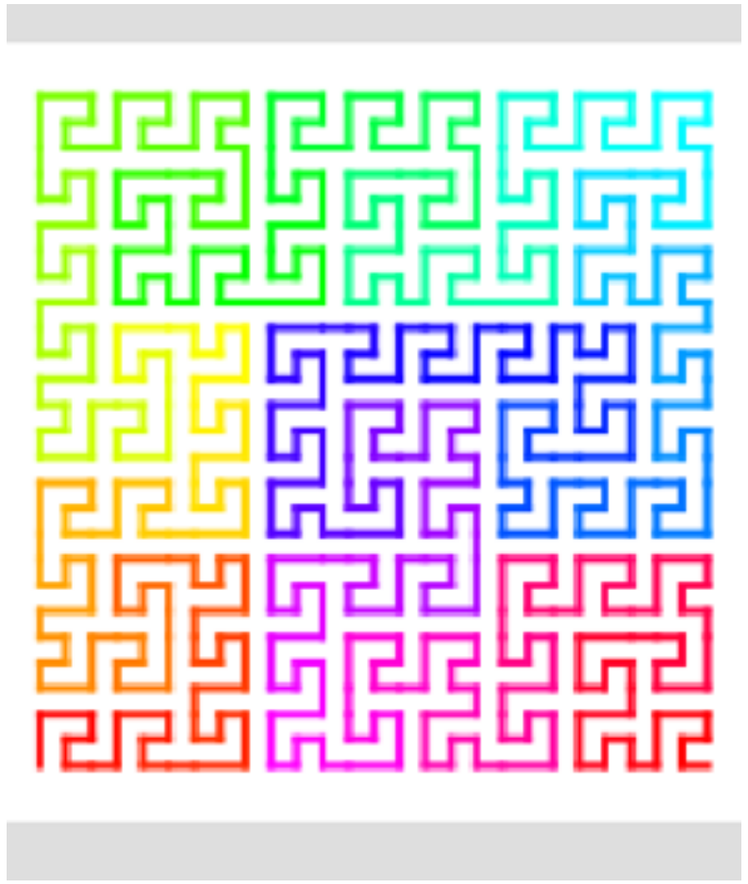


A map from an interval to a square

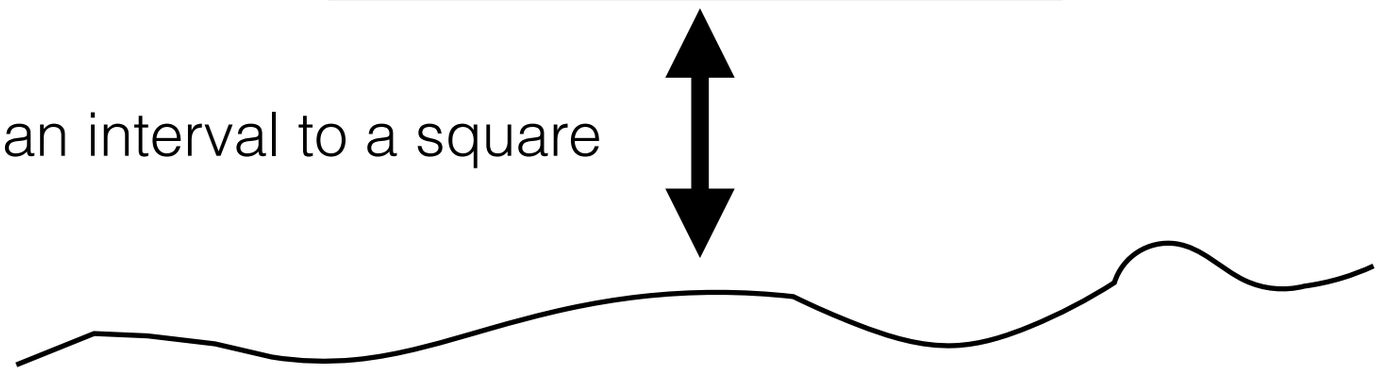


Space filling curves

https://mathsbyagirl.wordpress.com/tag/curve/



A map from an interval to a square



Z-order space filling curves

- Assume 2D points with integer coordinates on k bits

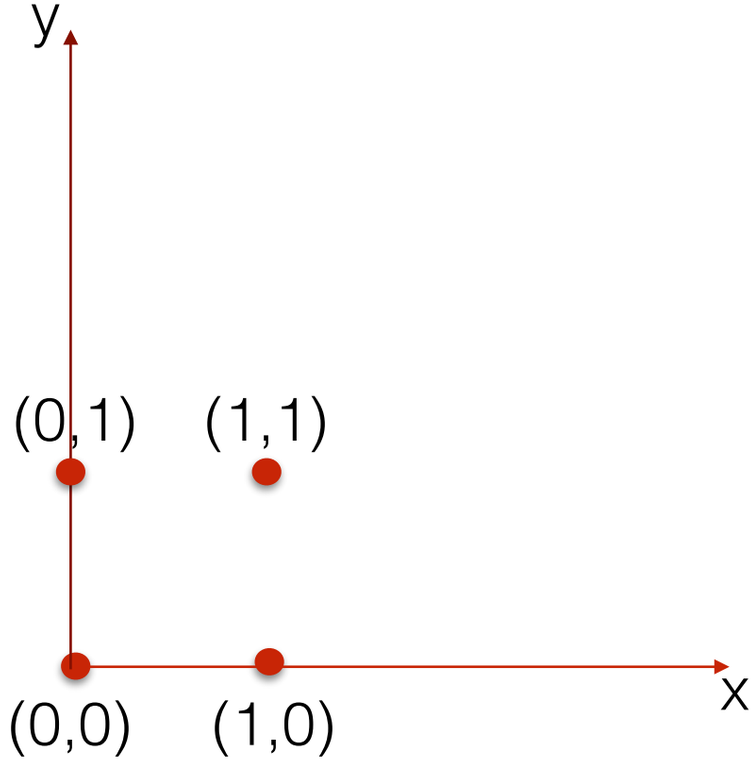
  $p = (x_1x_2x_3\ldots x_k, y_1y_2y_3\ldots y_k)$

- Define the Z-index of a point

  $Z\_index : \{0,..,2^k\text{-}1\} \times \{0,..,2^k\text{-}1\} \rightarrow \{0,..,2^{2k}\text{-}1\}$

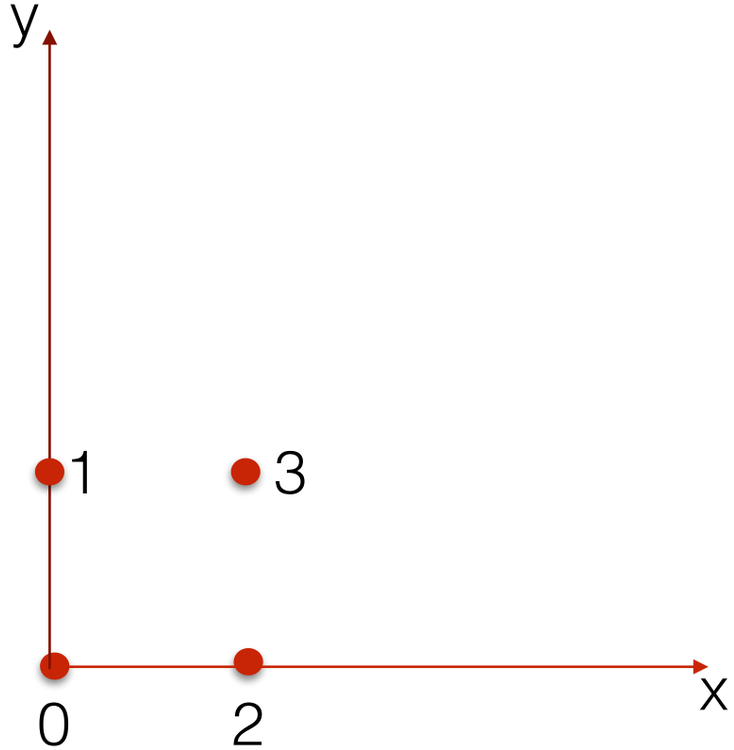
  $Z\_index(p) = x_1y_1x_2y_2\ldots x_ky_k$

k=1 bit



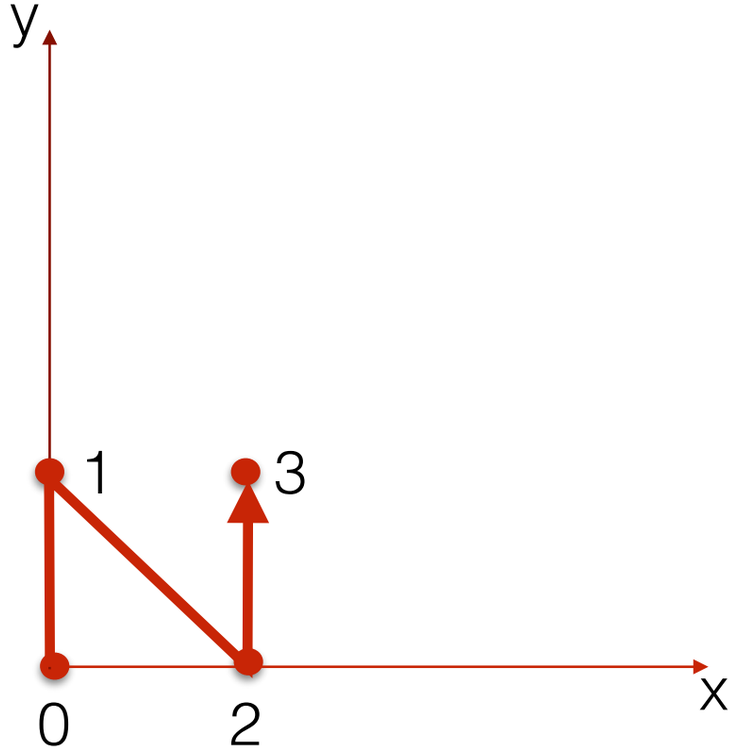
p	Z_index(p)
(0,0)	0
(0,1)	1
(1,0)	2
(1,1)	3

k=1 bit



p	Z_index(p)
(0,0)	0
(0,1)	1
(1,0)	2
(1,1)	3

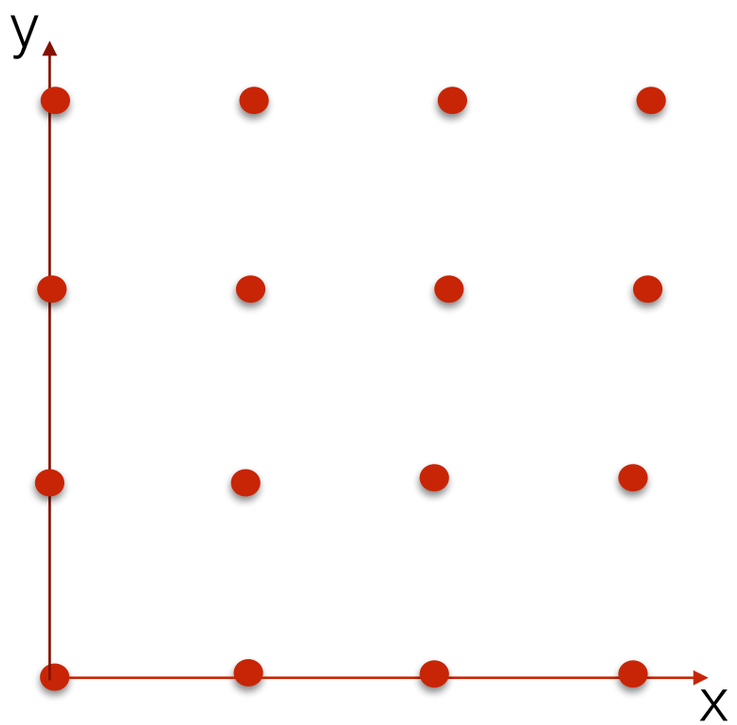
k=1 bit



p	Z_index(p)
(0,0)	0
(0,1)	1
(1,0)	2
(1,1)	3

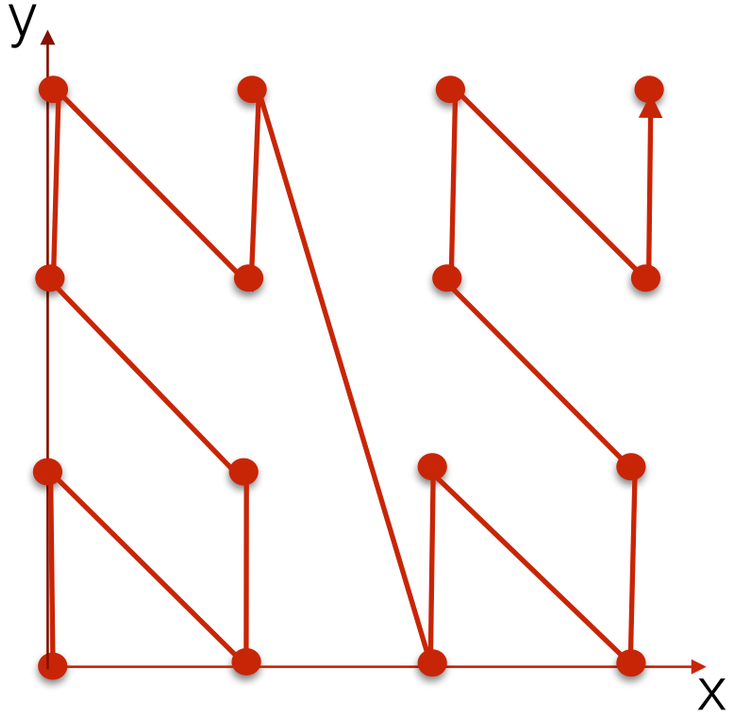
k=2 bits

Find the Z-order!

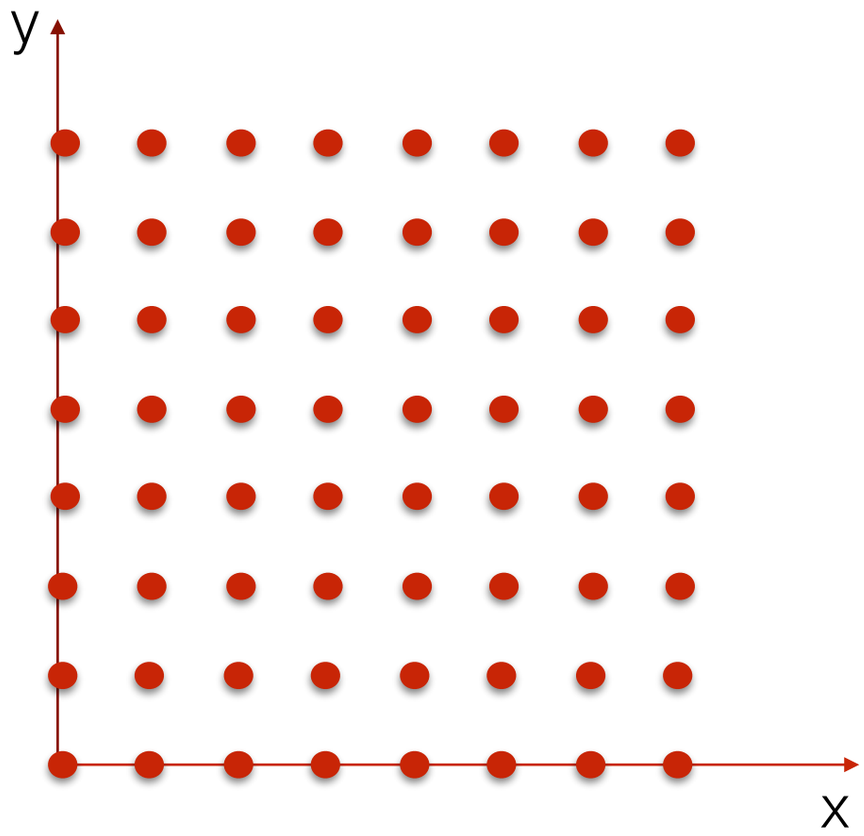


p	Z_index(p)
(00,00)	0000=0
(00,01)	0001=1
(00,10)	0100=4
(00,11)	0101=5
(01,00)	
(01,01)	
(01,10)	
(01,11)	
(10,00)	
(10,01)	
(10,10)	
(10,11)	
(11,00)	
(11,01)	
(11,10)	
(11,11)	

k=2 bits



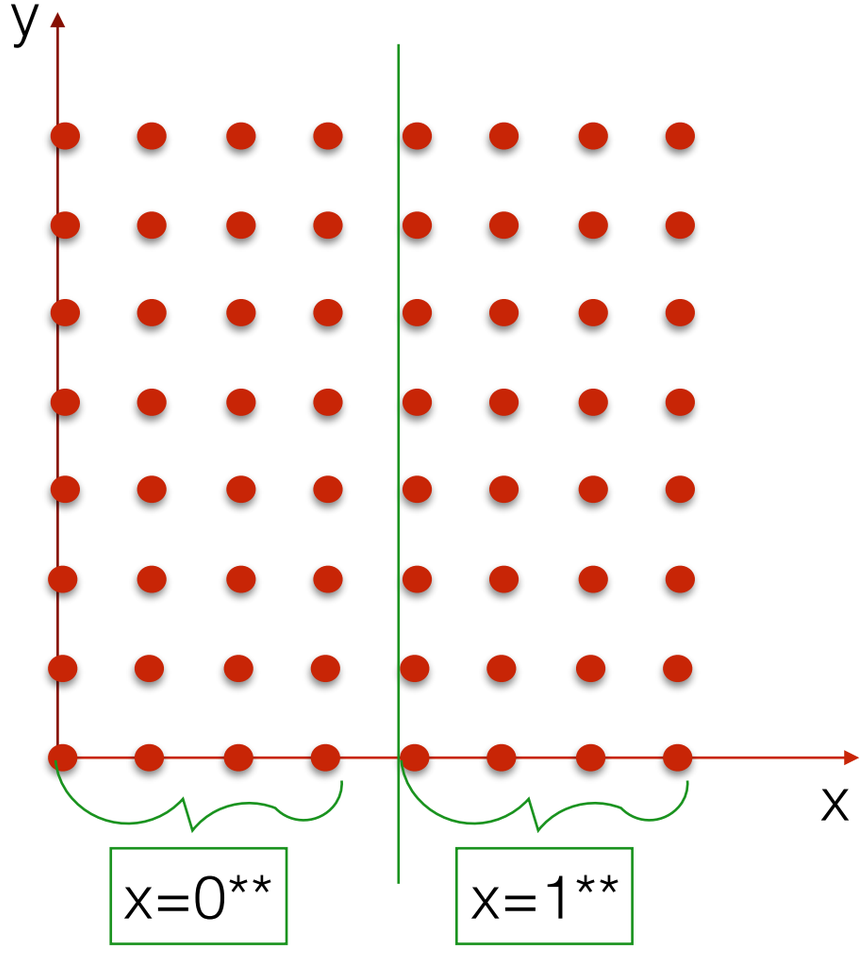
k=3 bits



Find the Z-order!

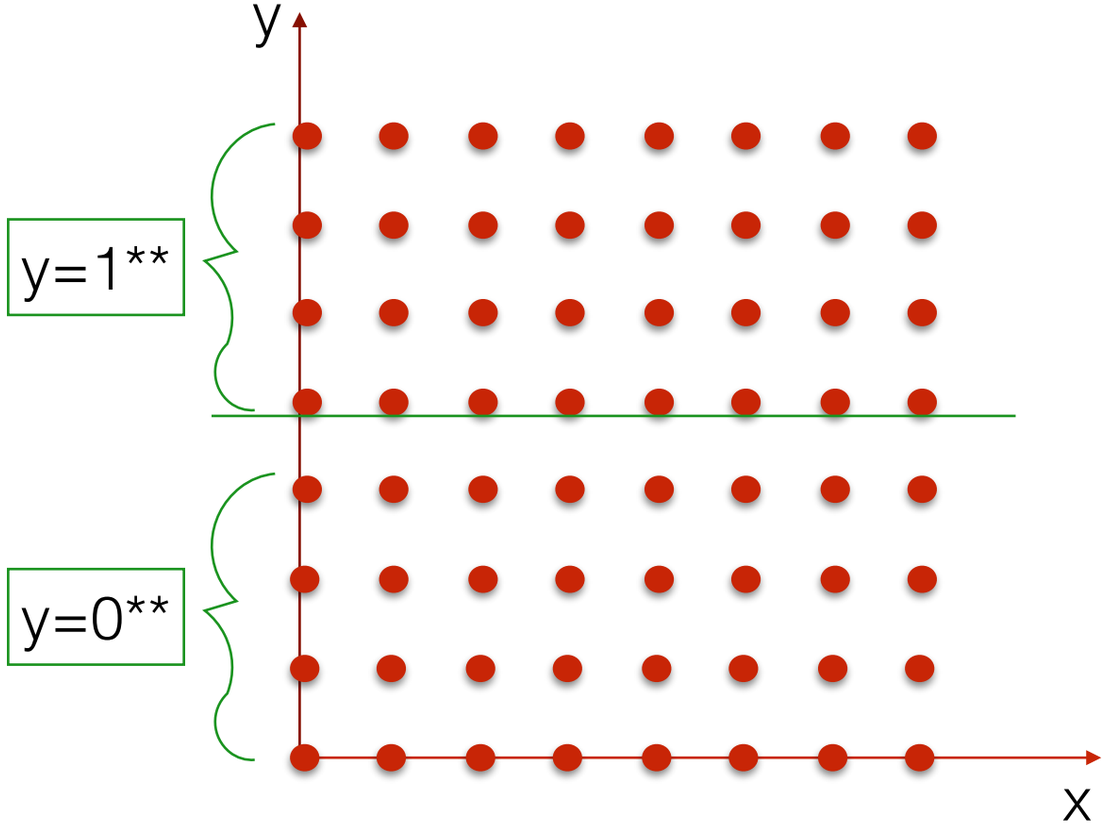
Computing the Z-index

- Consider an x-coordinate $x_1x_2x_3$ in the square [0,…8)
  - x_1=0 means the point will reside in the first half
  - x_1=1 means the point will reside in the second half



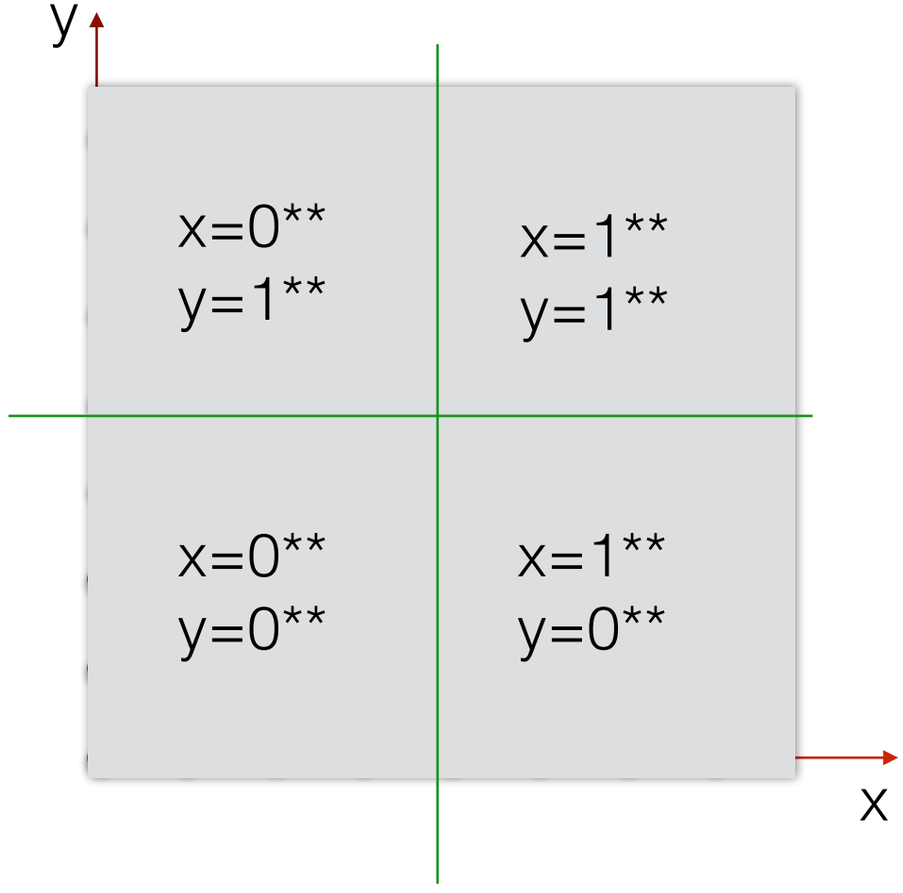
Computing the Z-index

- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half



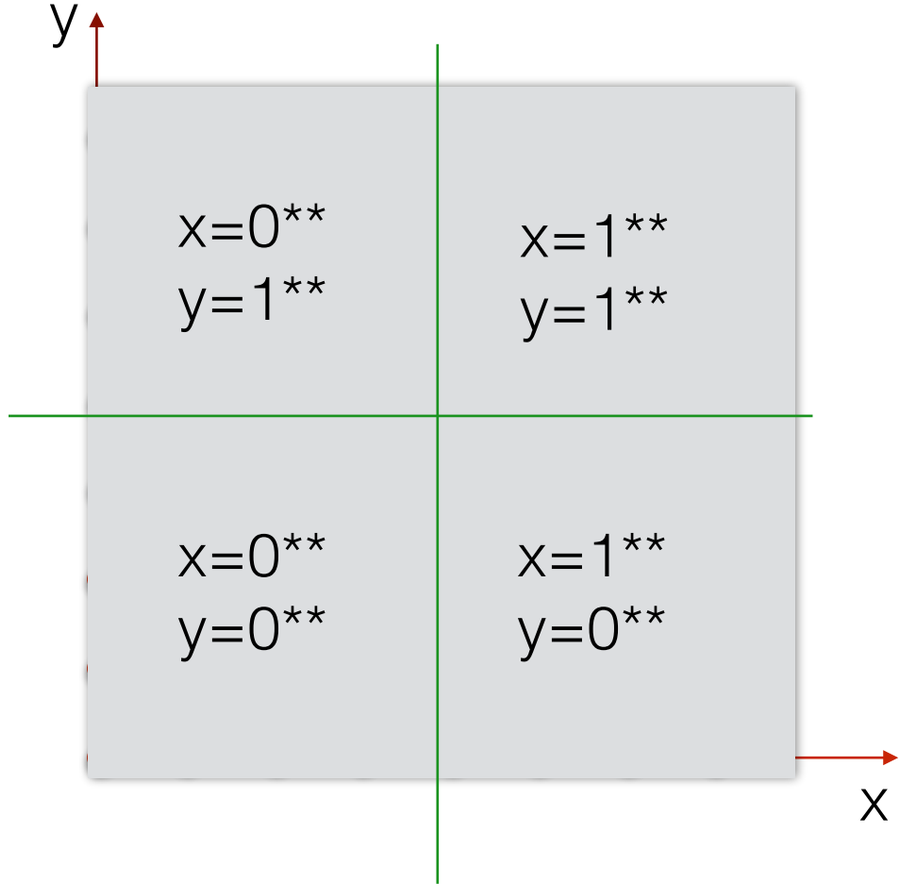
Computing the Z-index

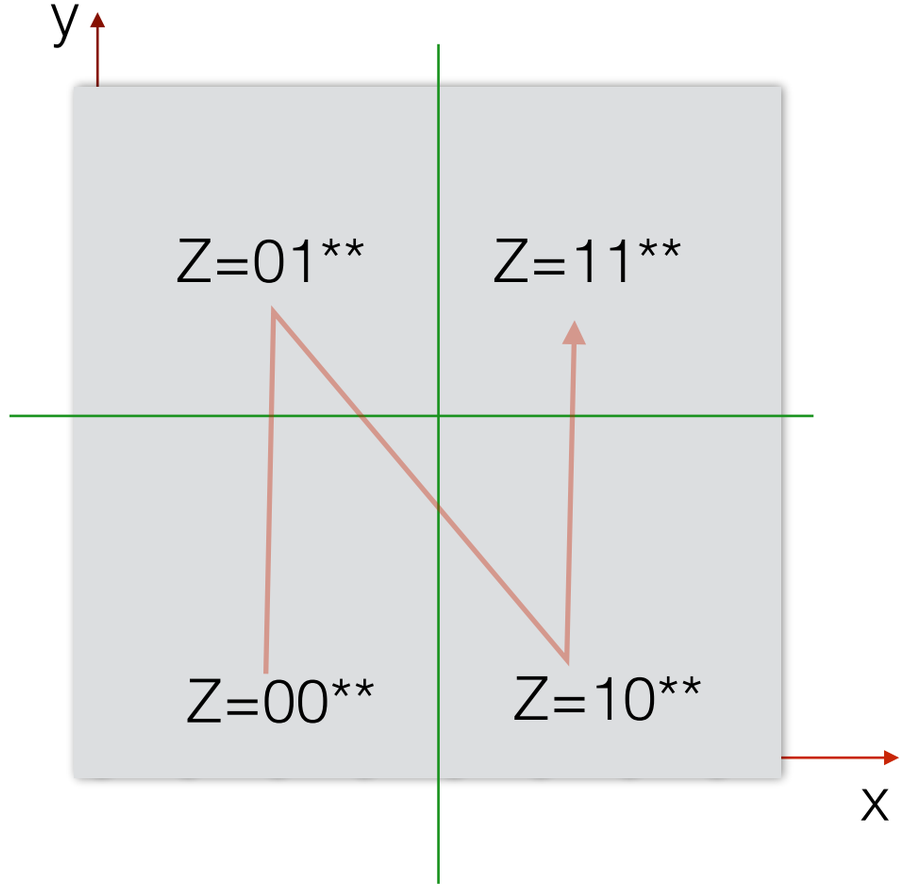
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half



Computing the Z-index

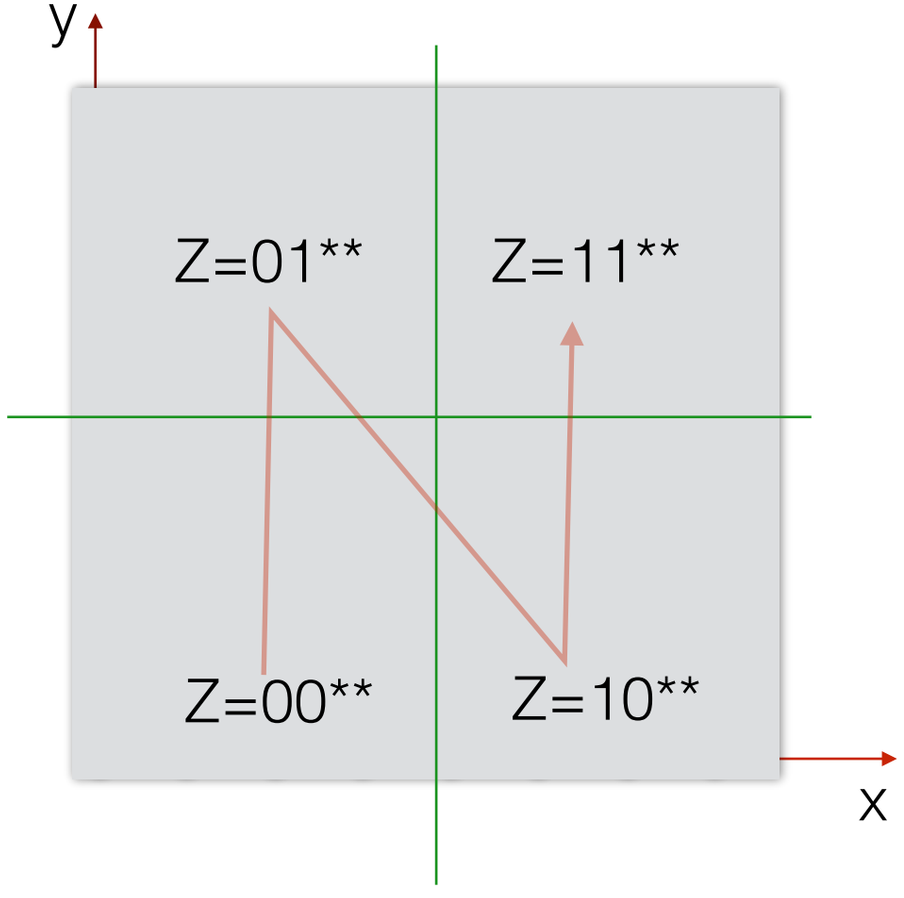
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

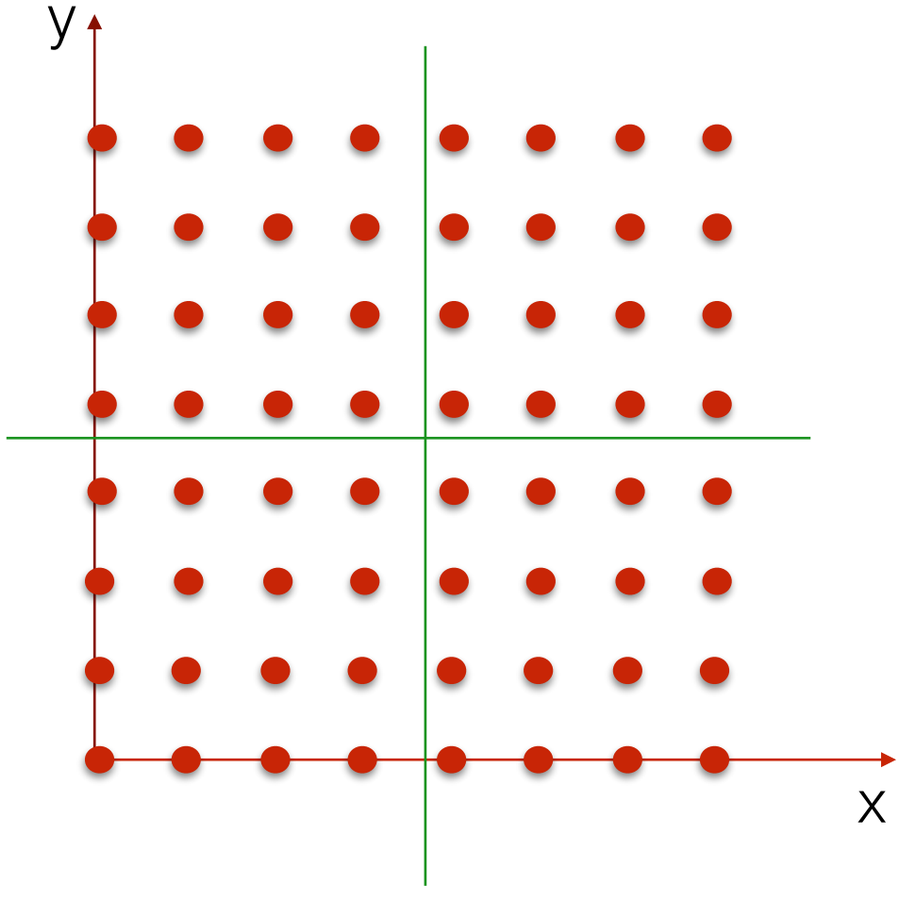




Computing the Z-index

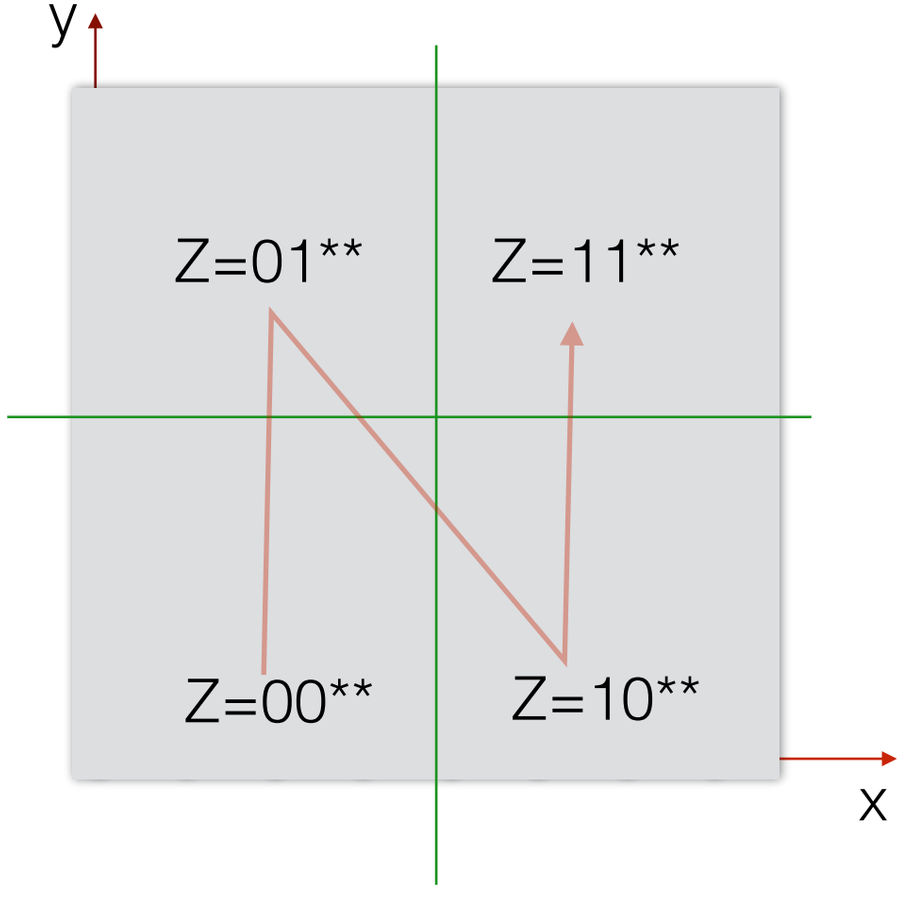
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

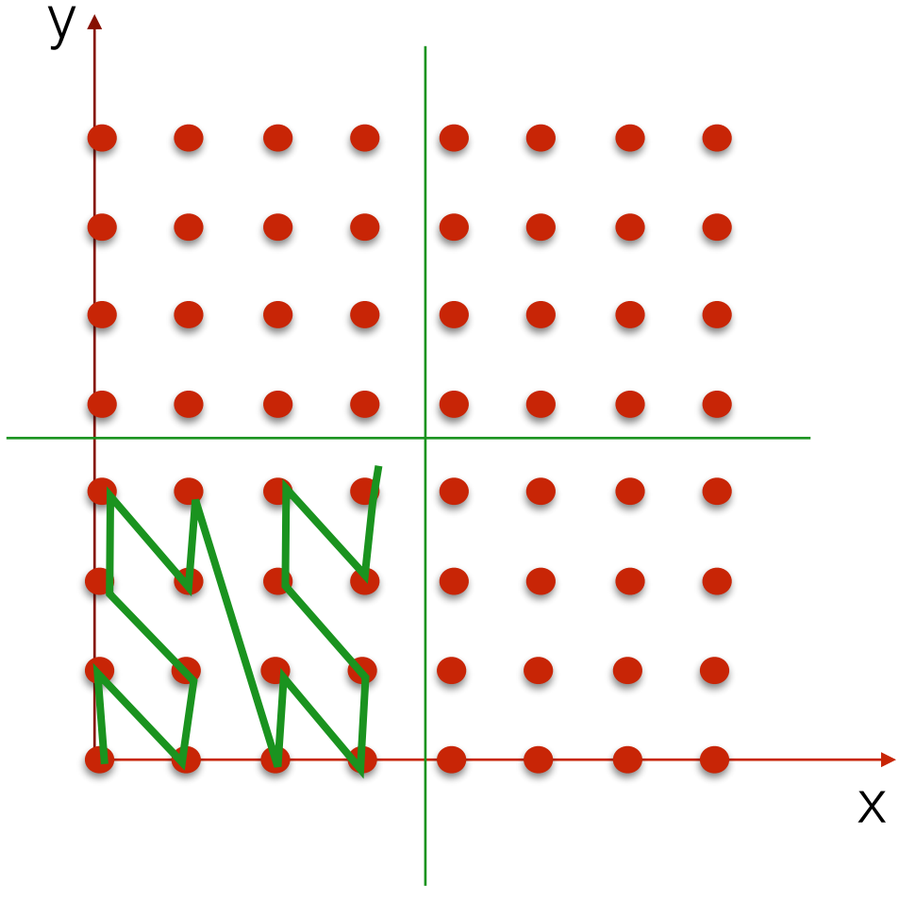




Computing the Z-index

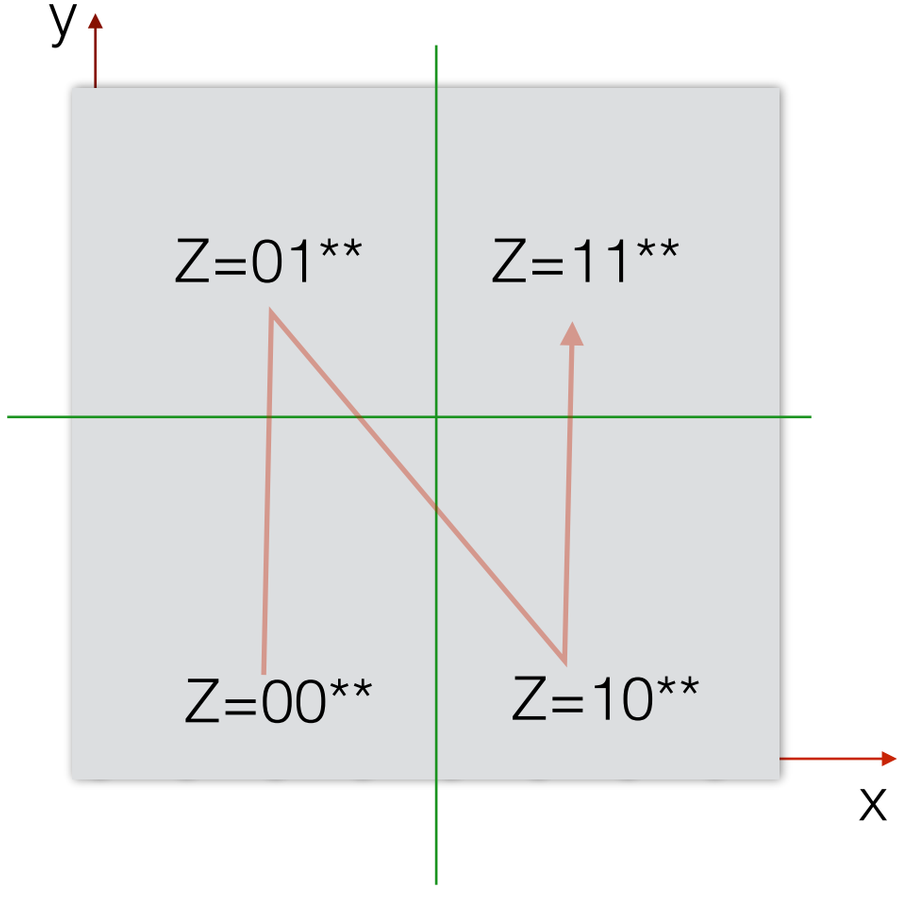
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

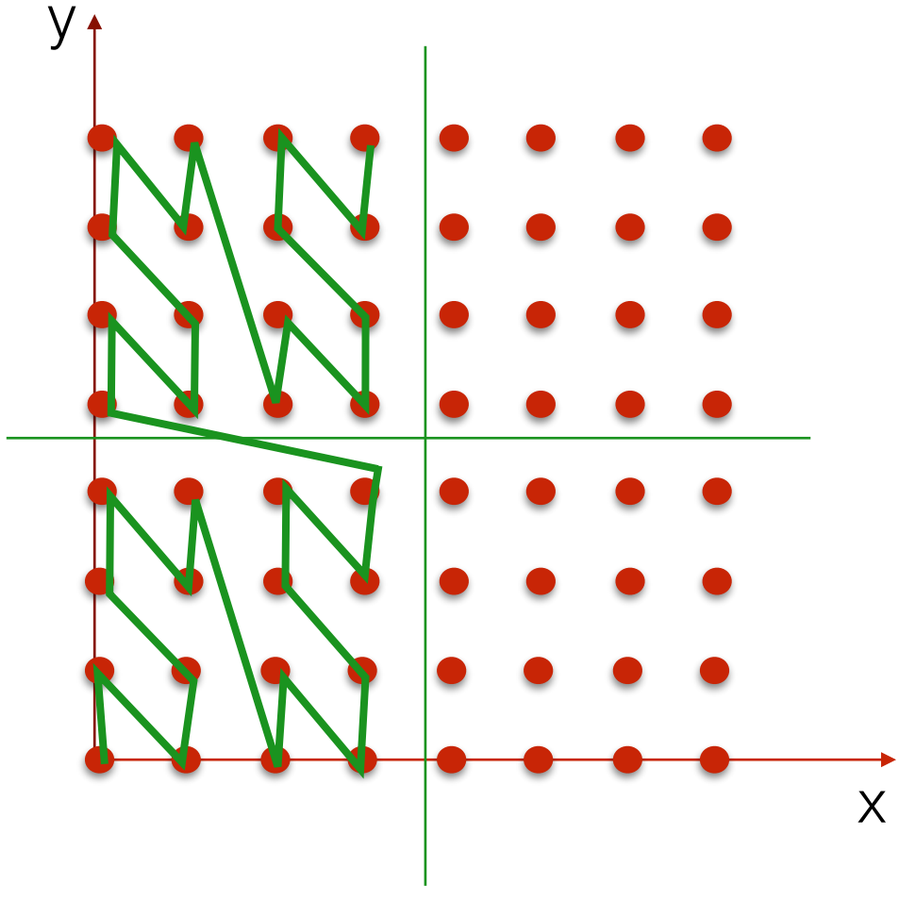




Computing the Z-index

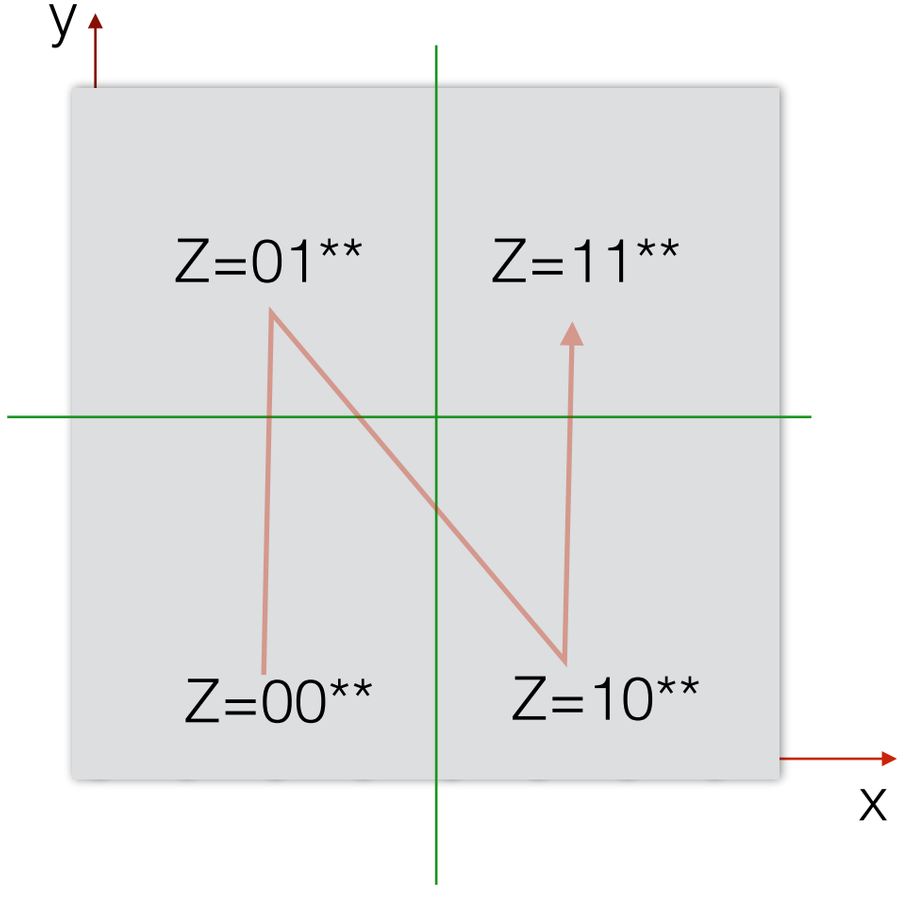
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

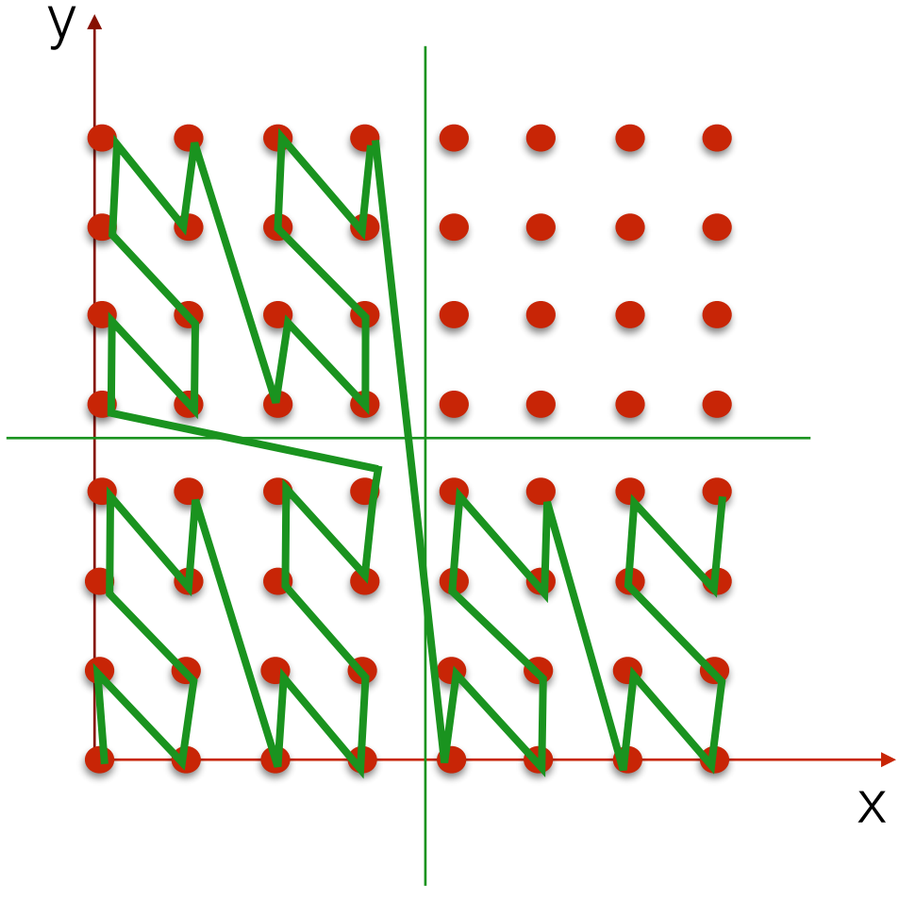




Computing the Z-index

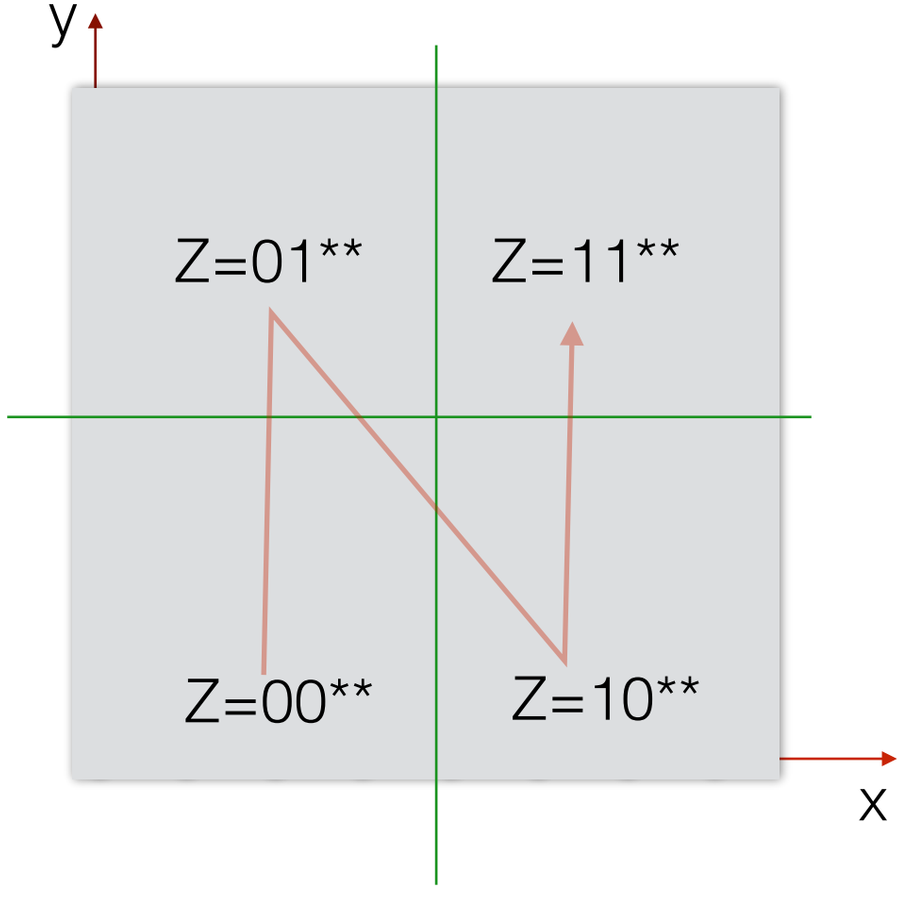
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

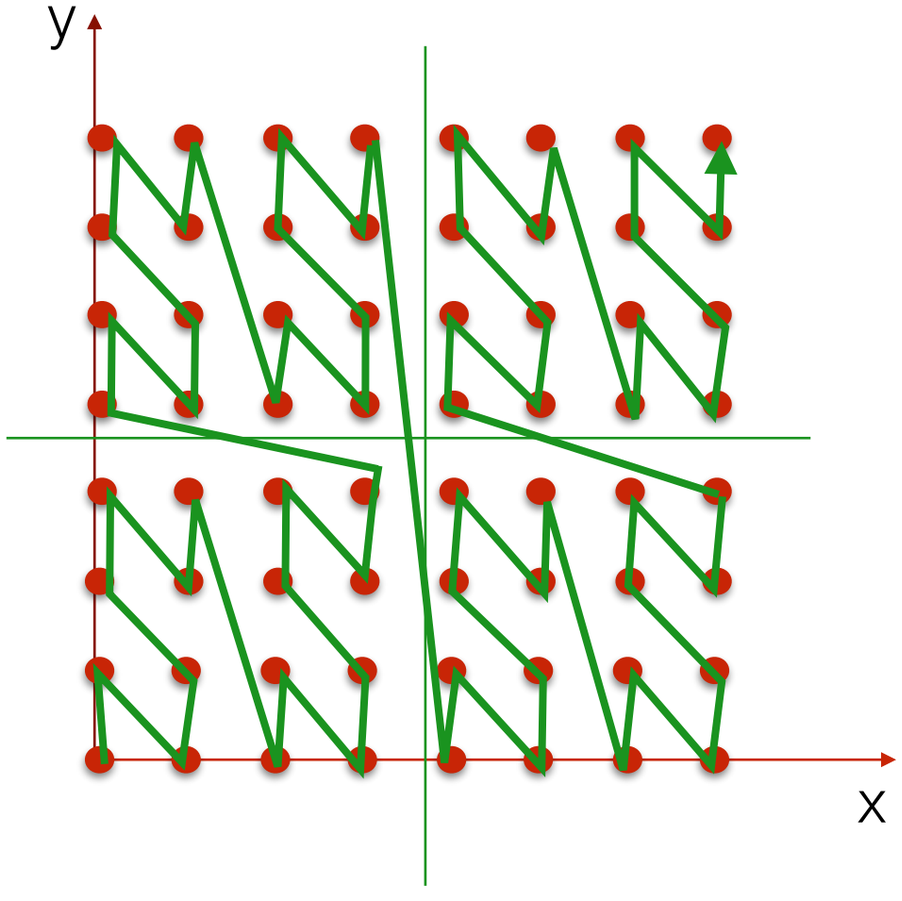




Computing the Z-index

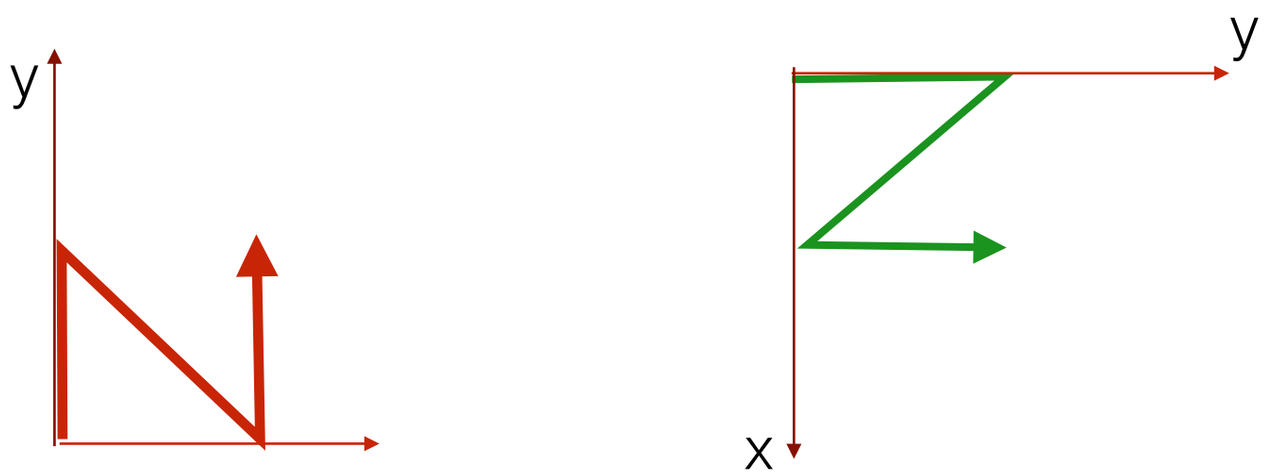
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half





Z-order space filling curves

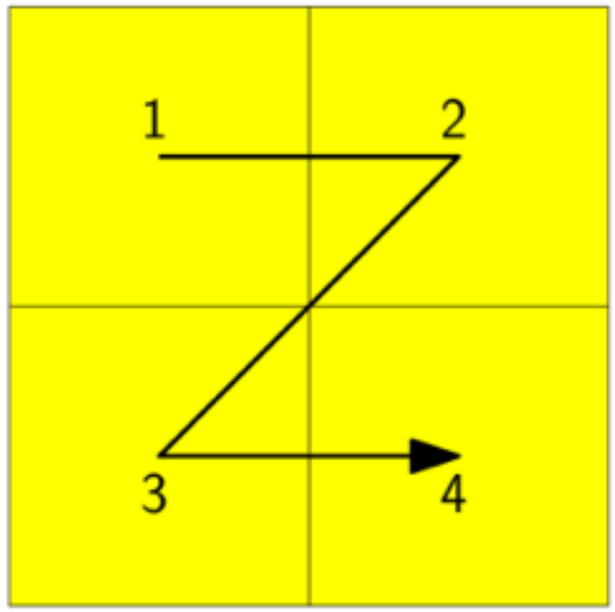
- Other Z-orders can be obtained similarly



- Can be extended to work with decimal numbers in [0,1)
  - make values positive (add smallest value)
  - divide all values by max value
  - ==> now we got values in [0,1)p=(.1100, .0101)

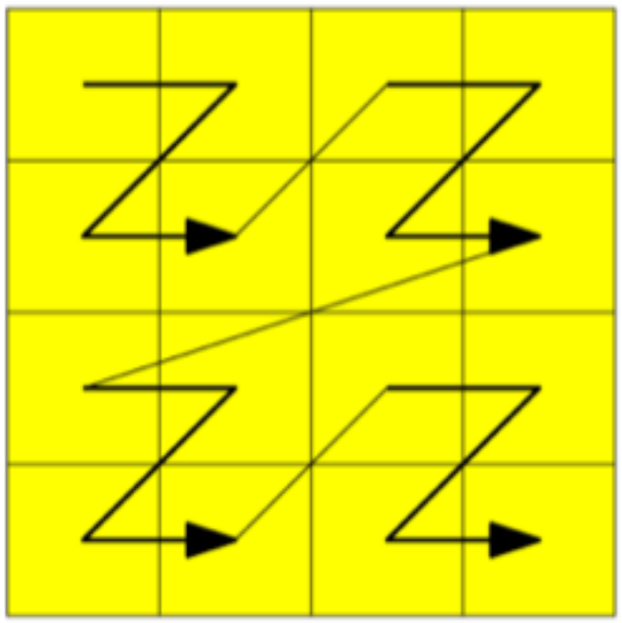
Z-order and quadtrees

visit quadrants recursively in this order: NW, NE, SW, SE



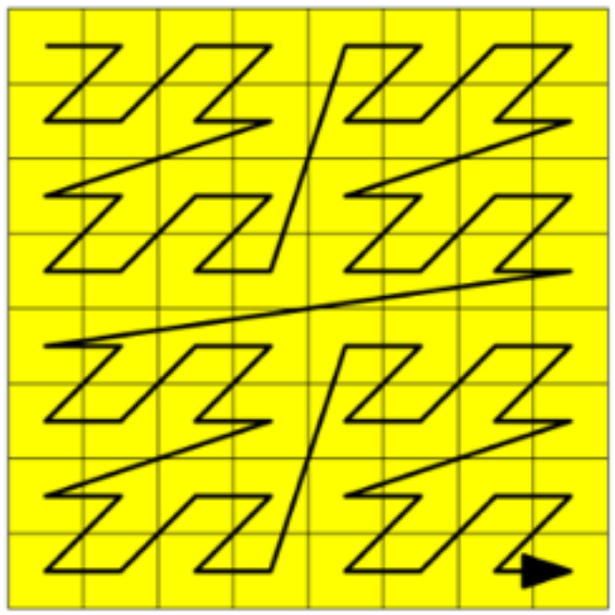
Z-order and quadtrees

visit quadrants recursively in this order: NW, NE, SW, SE



Z-order and quadtrees

visit quadrants recursively in this order: NW, NE, SW, SE



Z-order and quadtrees

visit quadrants recursively in this order: NW, NE, SW, SE

Z-order and quadtrees

visit quadrants recursively in this order: NW, NE, SW, SE

- At the limit, it will reach all points in the square ==> space filling curve

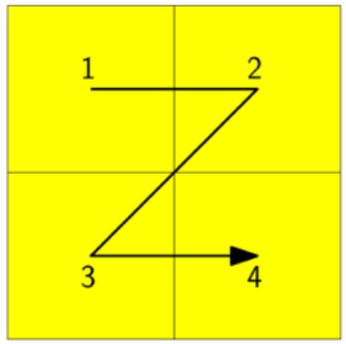
Z-order and quadtrees

visit quadrants recursively in this order: NW, NE, SW, SE

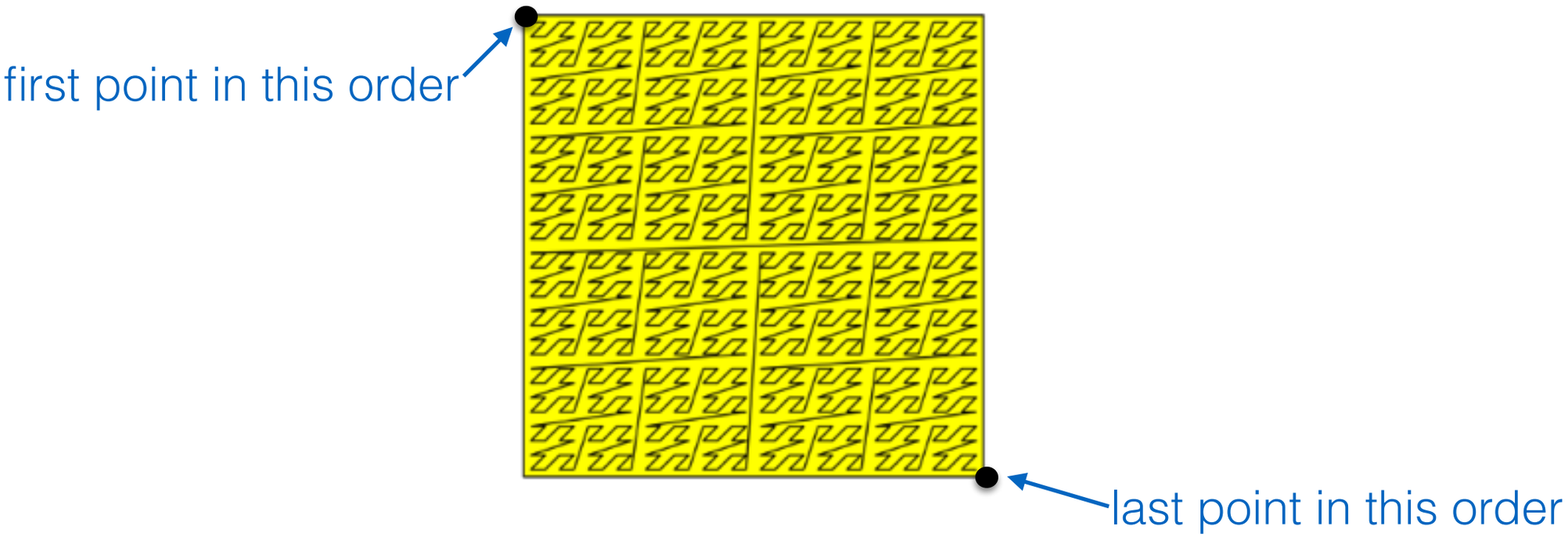
Where is the very first point visited?

- At the limit, it will reach all points in the square ==> space filling curve

Z-order and quadtrees

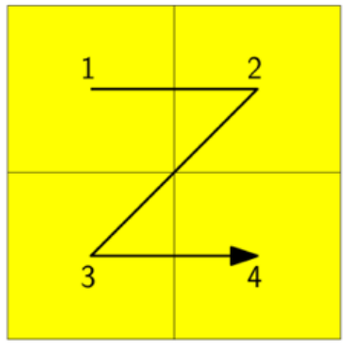


visit quadrants recursively in this order: NW, NE, SW, SE

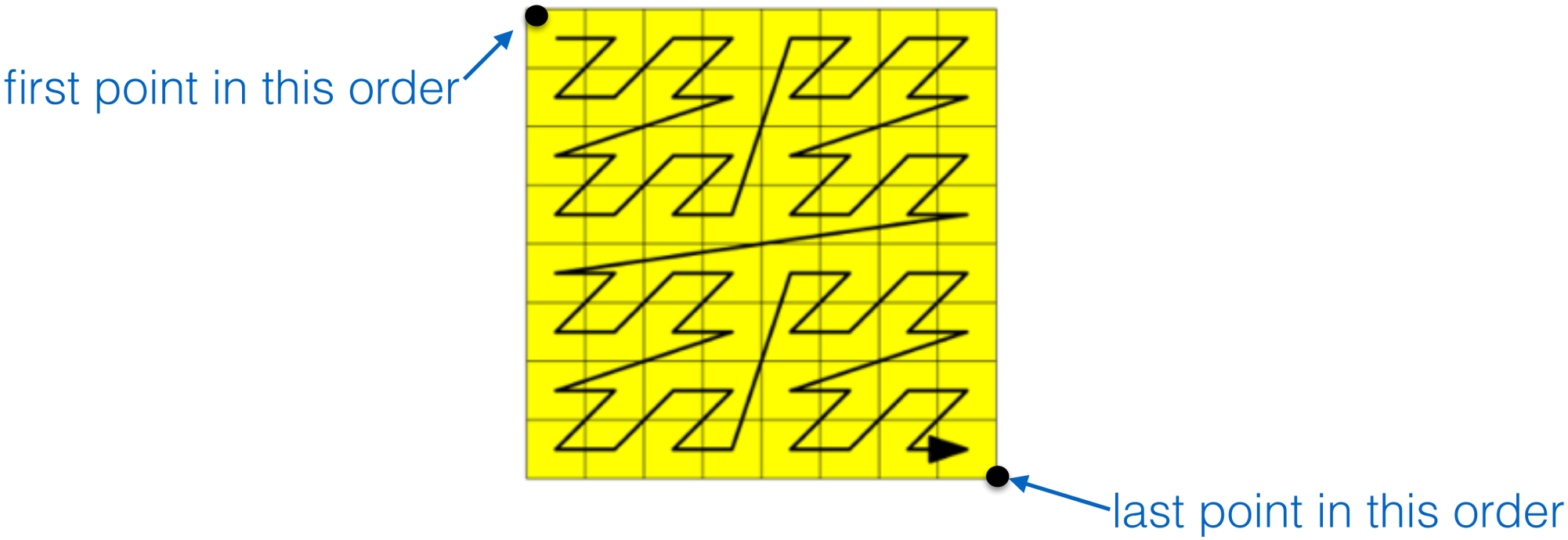


- At the limit, it will reach all points in the square ==> space filling curve

Z-order and quadtrees

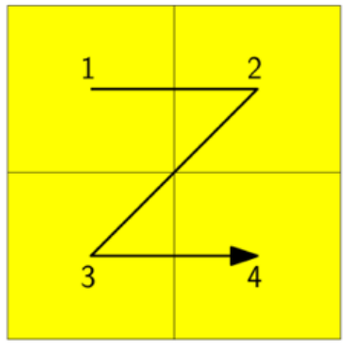


visit quadrants recursively in this order: NW, NE, SW, SE

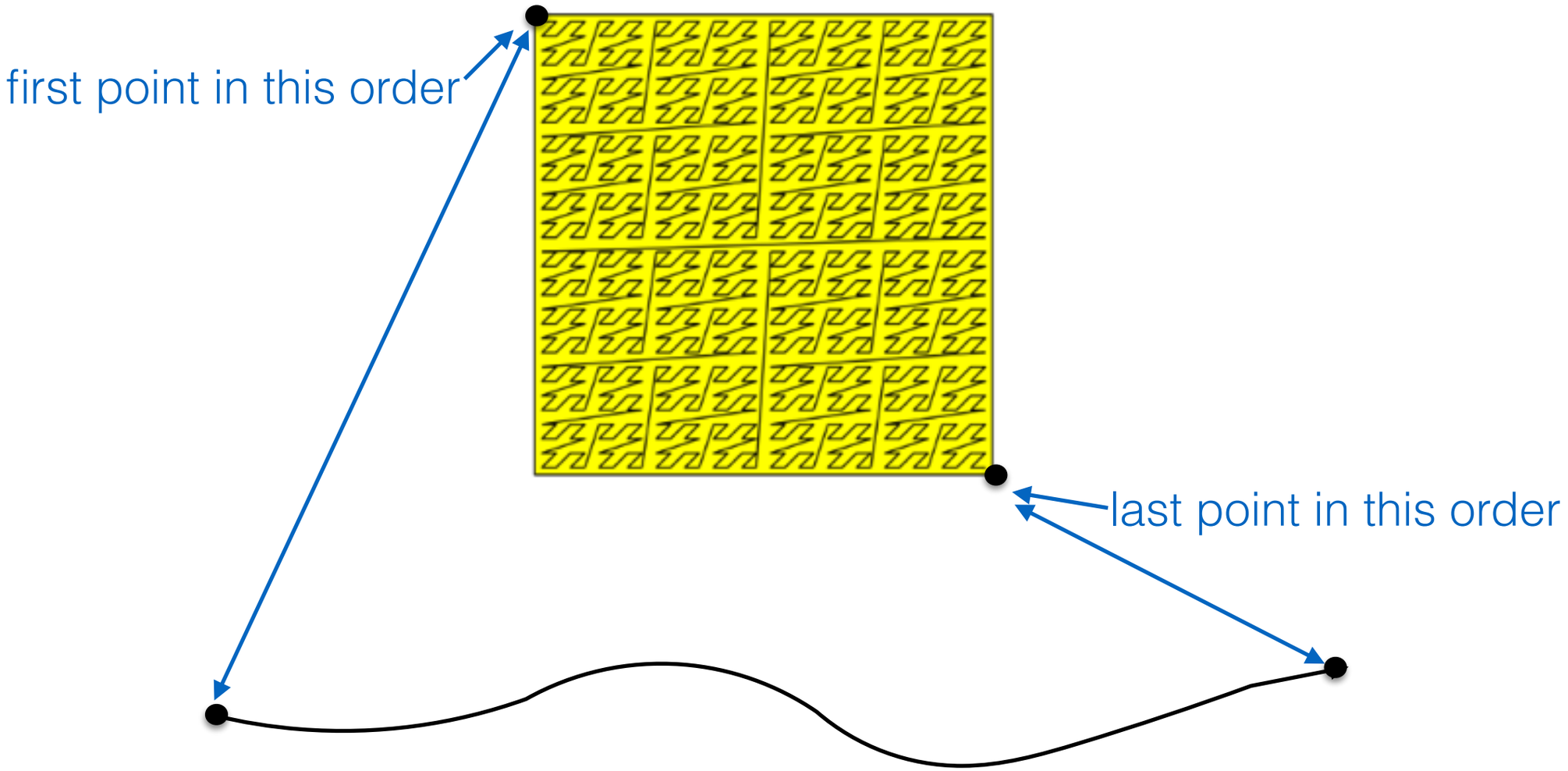


- At the limit, it will reach all points in the square ==> space filling curve

Z-order and quadtrees

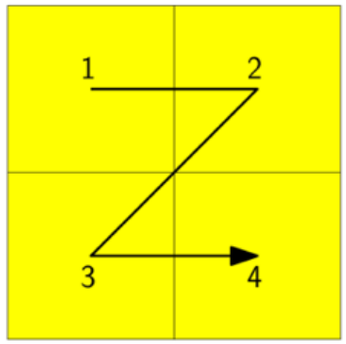


visit quadrants recursively in this order: NW, NE, SW, SE

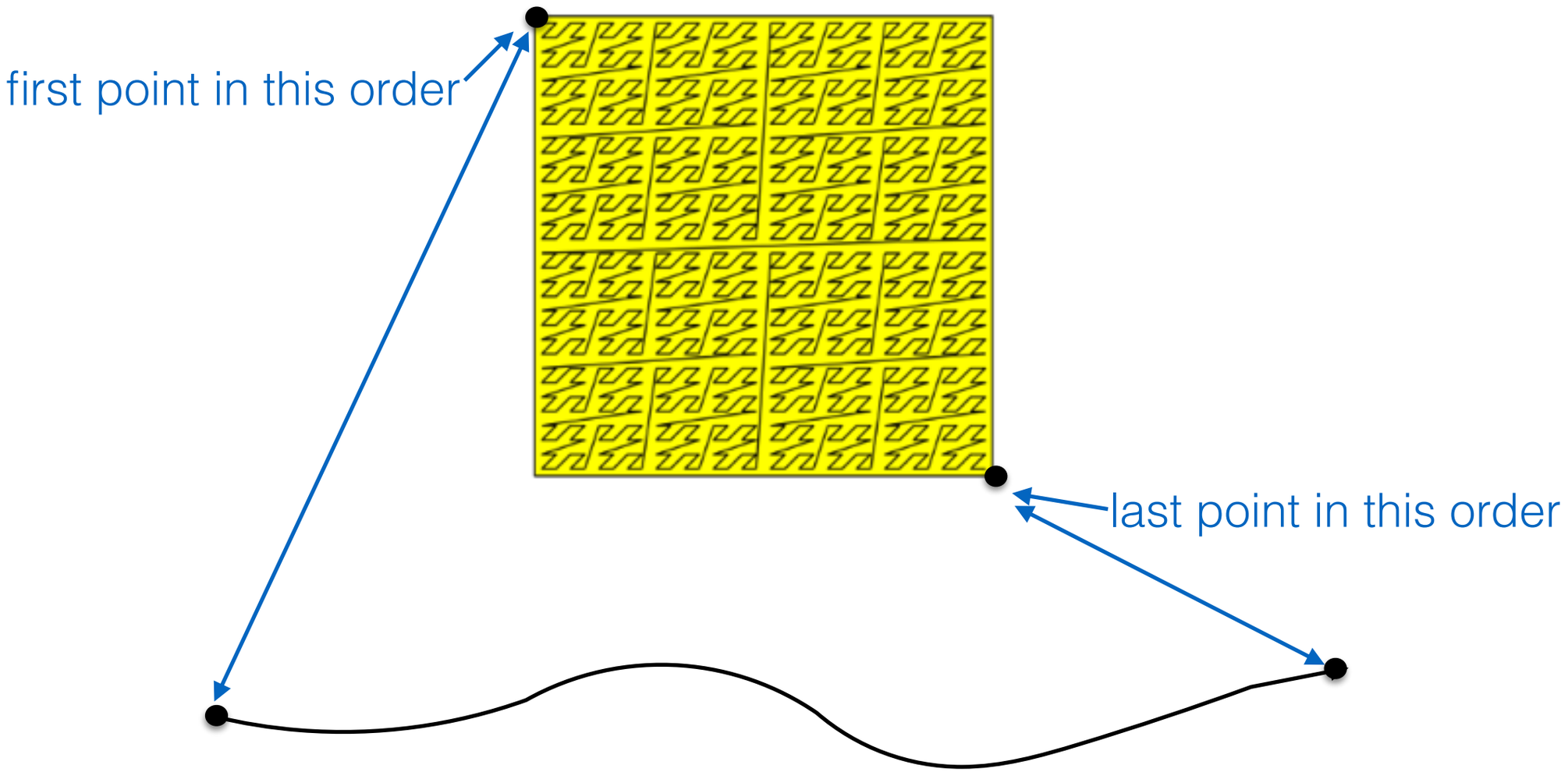


- Every point in the square will be visited by this curve
- 2D ==> 1D

Z-order and quadtrees



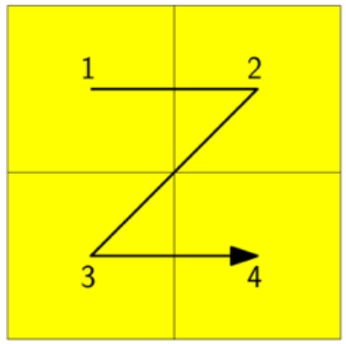
visit quadrants recursively in this order: NW, NE, SW, SE



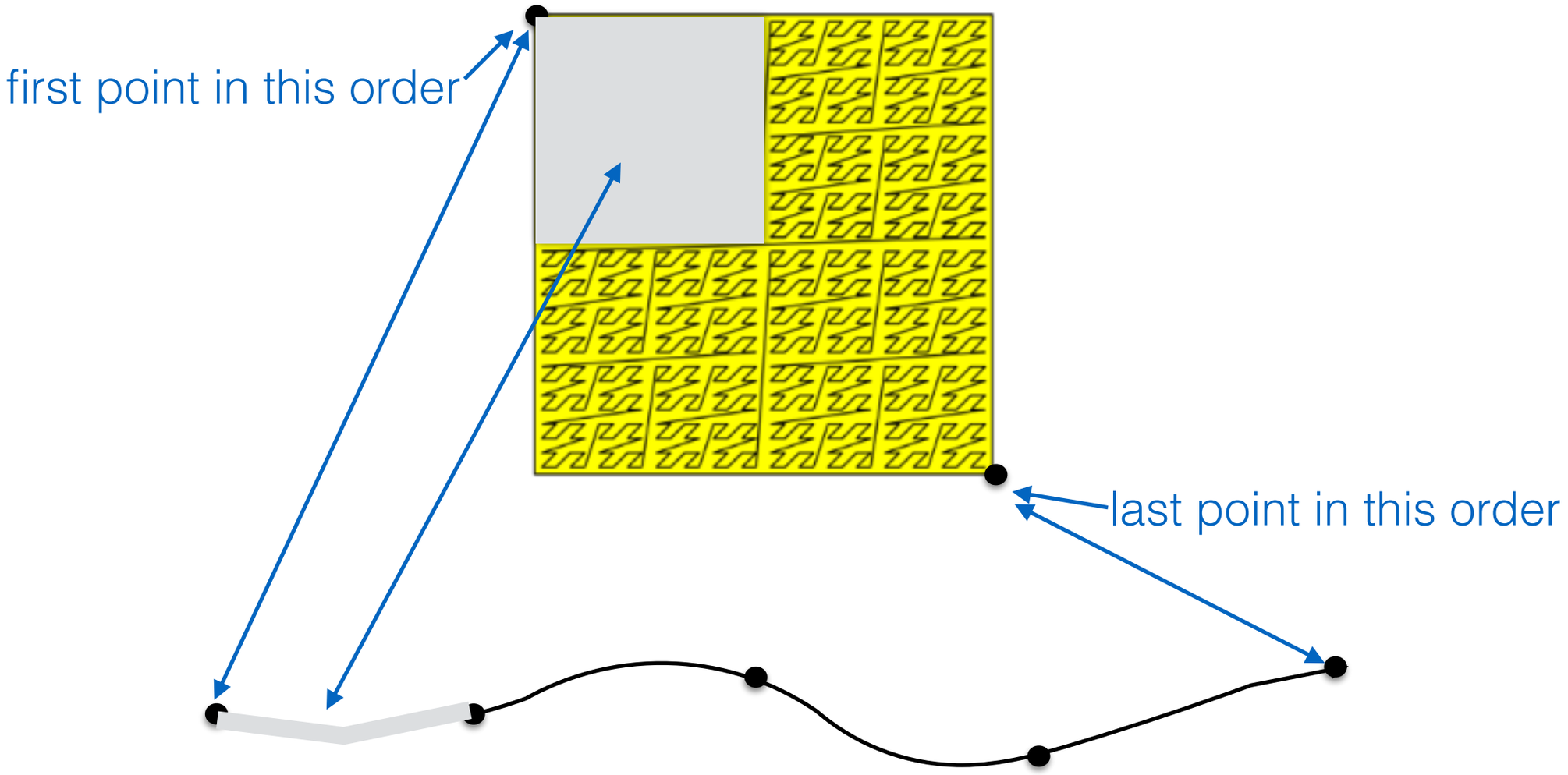
- We visit quadrant 1 before we visit quadrant 2:

  ==> All points in quadrant 1 comes before all points in quadrant 2

Z-order and quadtrees



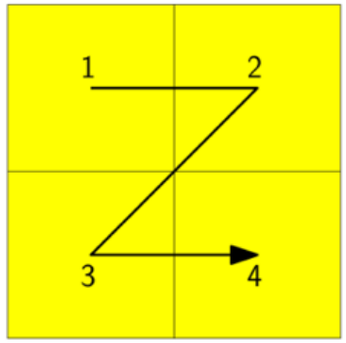
visit quadrants recursively in this order: NW, NE, SW, SE



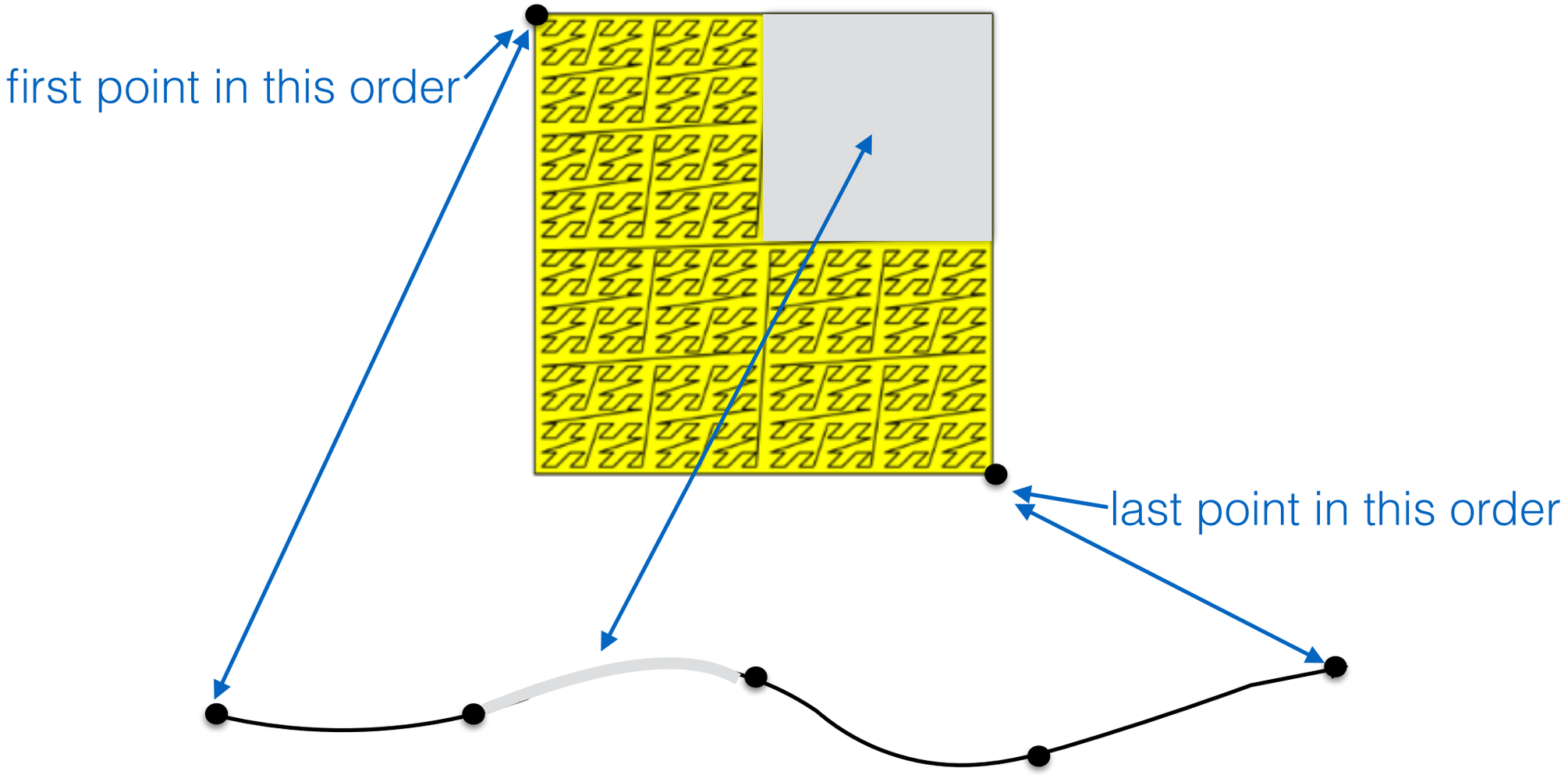
- We visit quadrant 1 before we visit quadrant 2:

  ==> All points in quadrant 1 comes before all points in quadrant 2

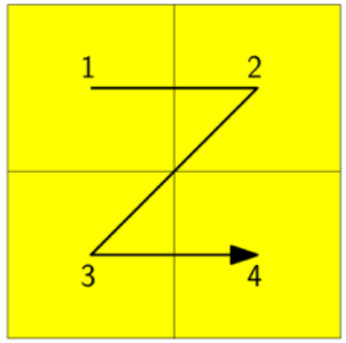
Z-order and quadtrees



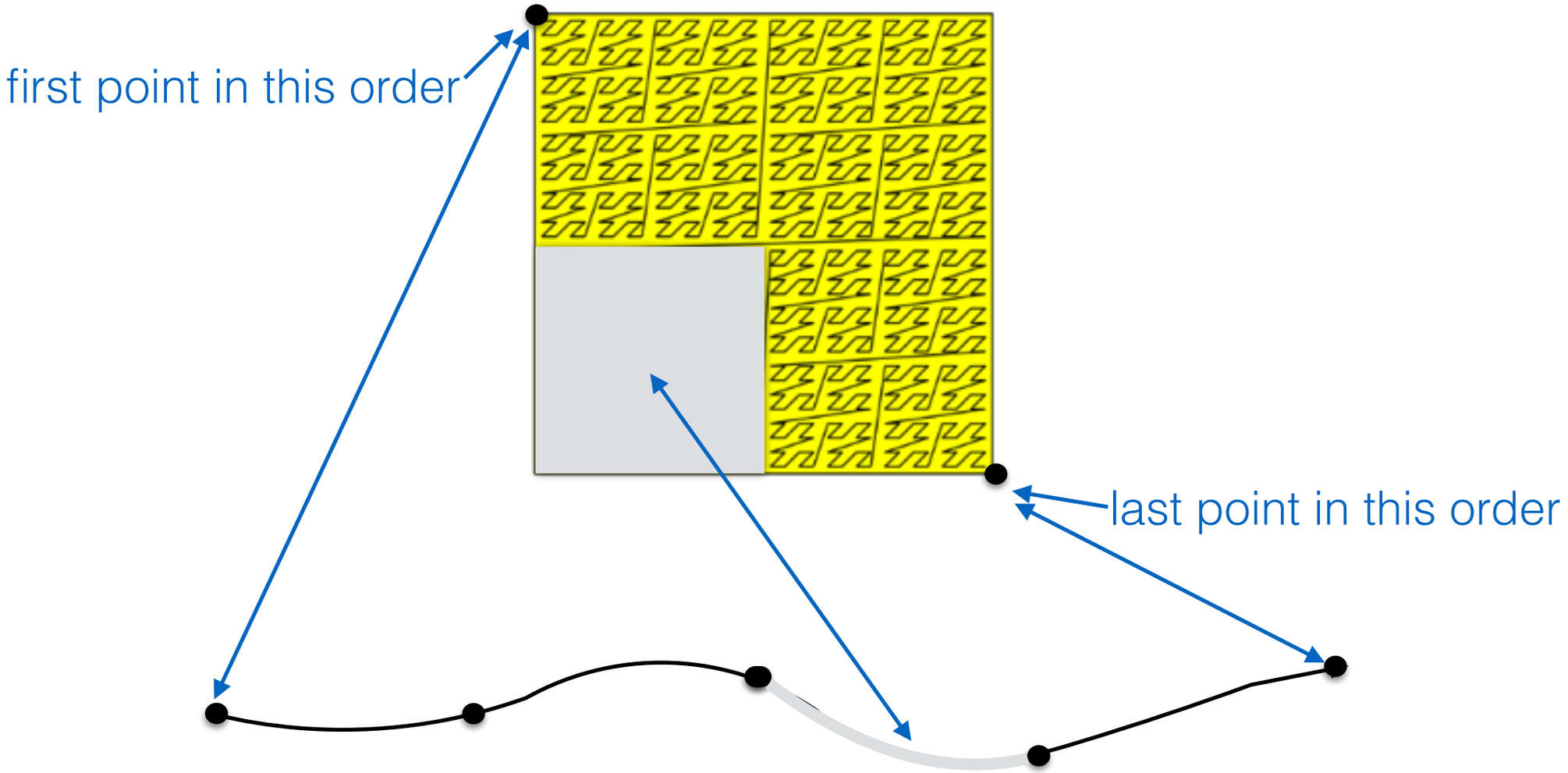
visit quadrants recursively in this order: NW, NE, SW, SE



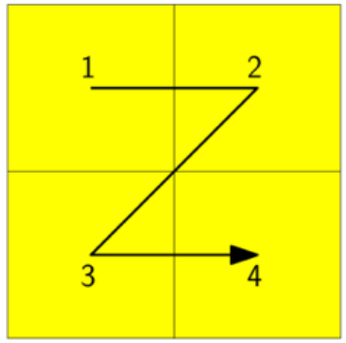
Z-order and quadtrees



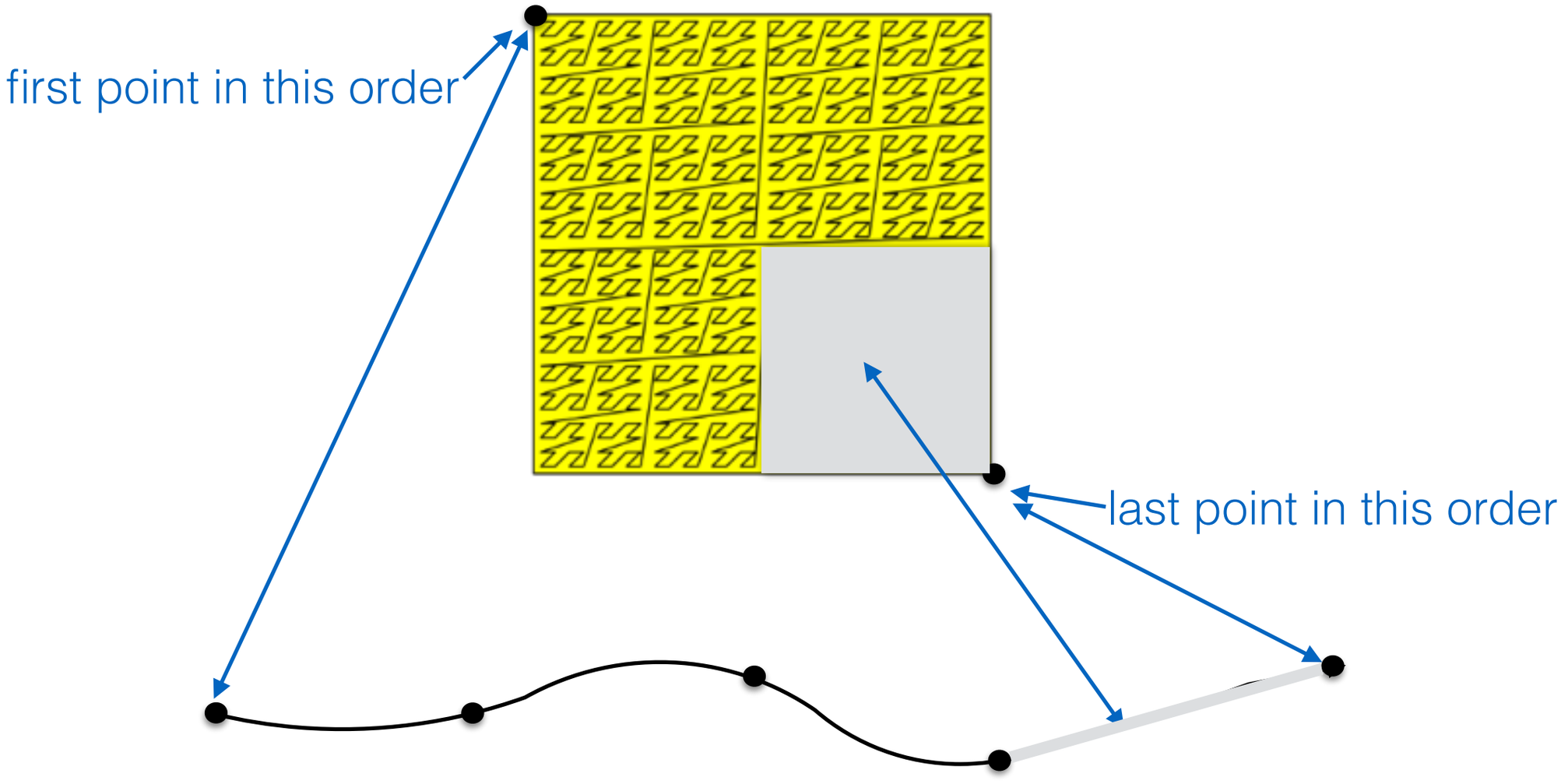
visit quadrants recursively in this order: NW, NE, SW, SE



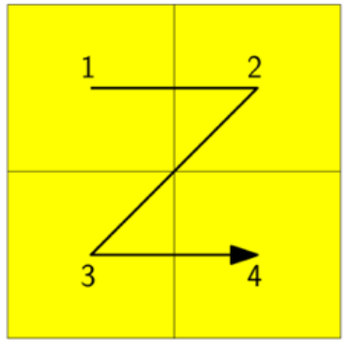
Z-order and quadtrees



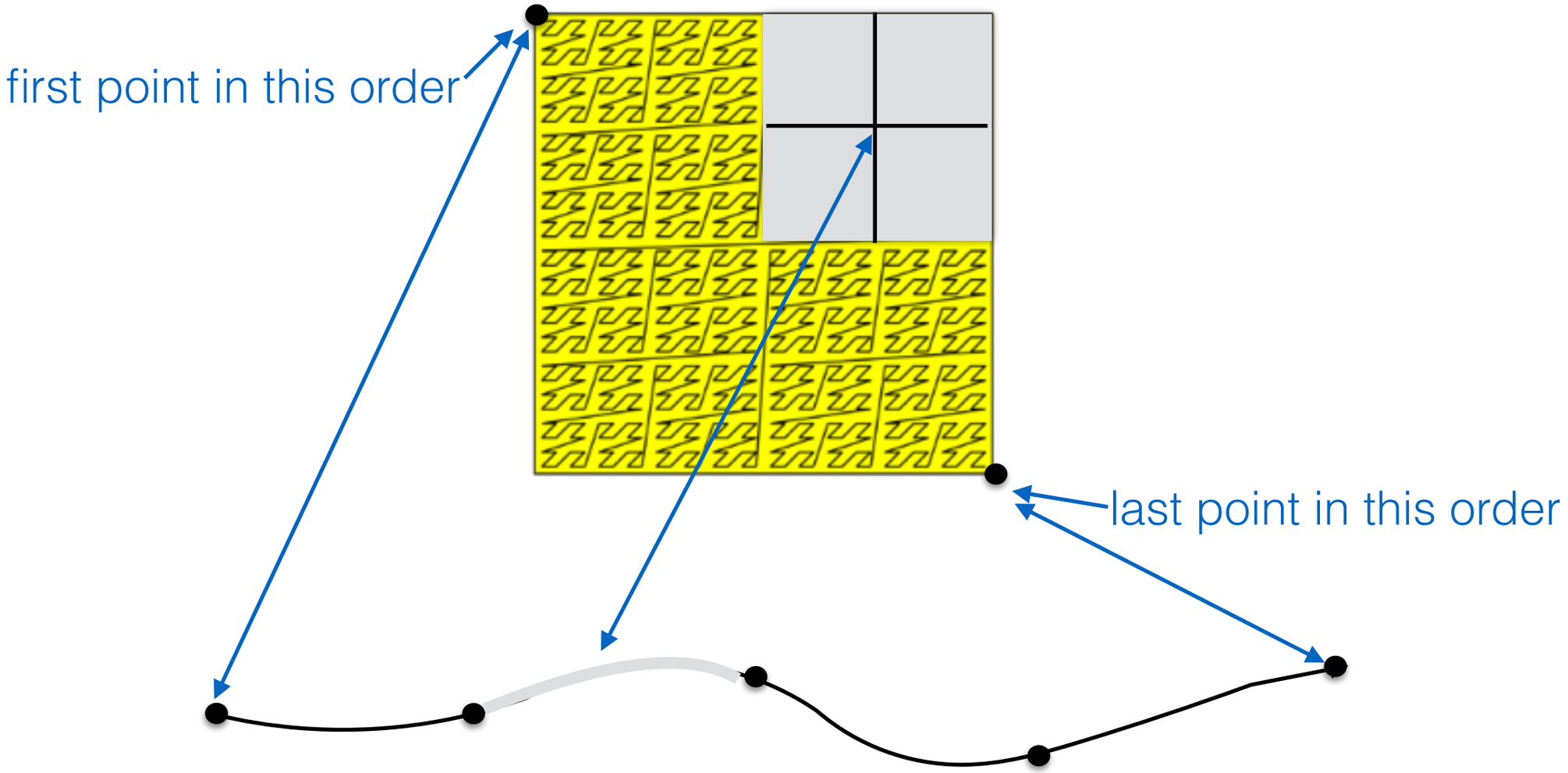
visit quadrants recursively in this order: NW, NE, SW, SE



Z-order and quadtrees

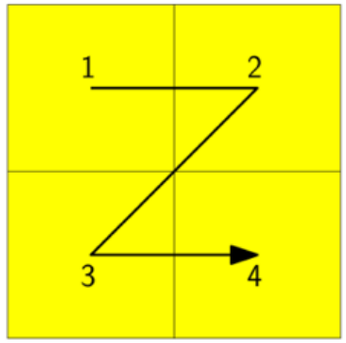


visit quadrants recursively in this order: NW, NE, SW, SE

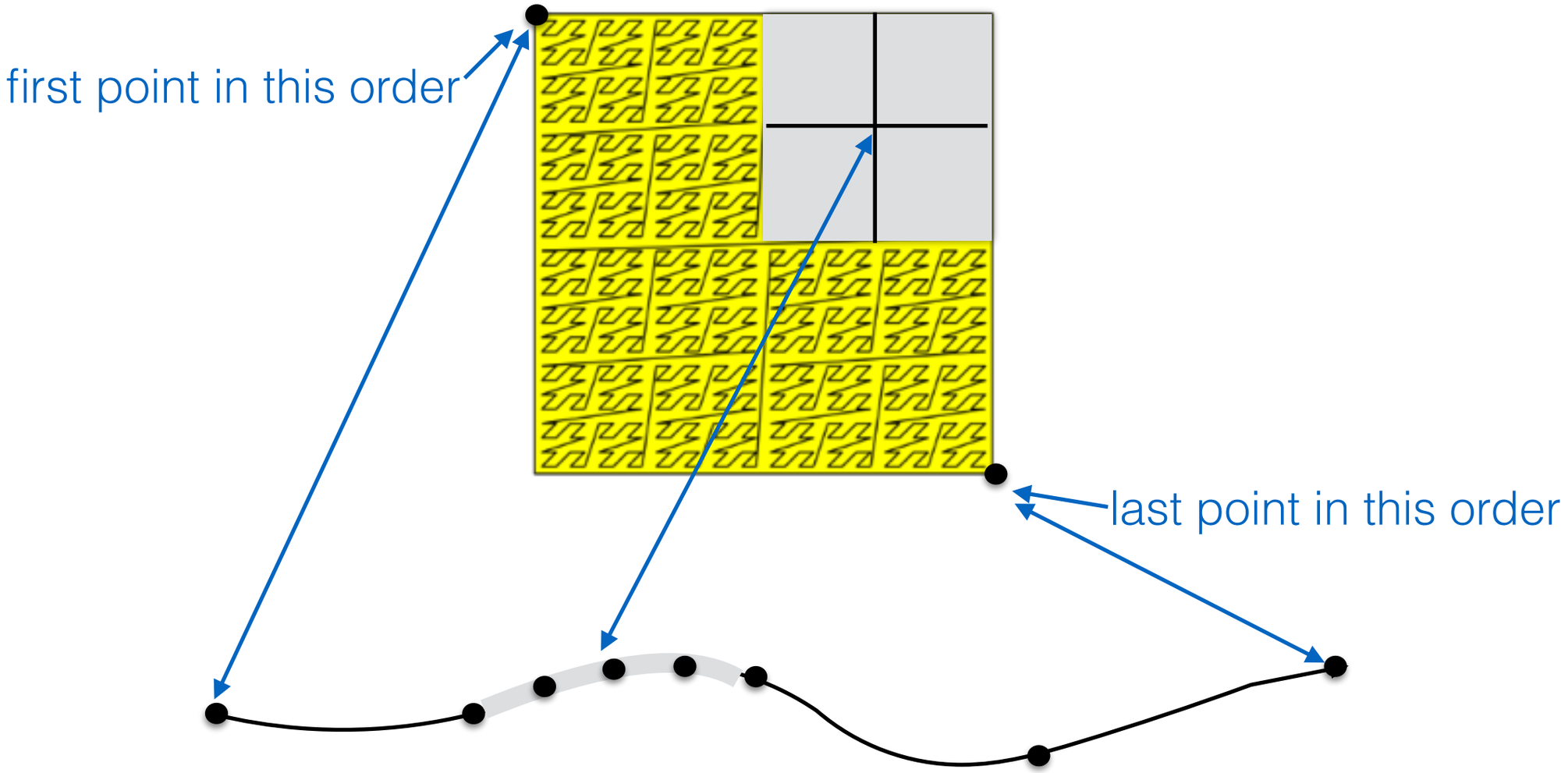


- and so on…..

Z-order and quadtrees

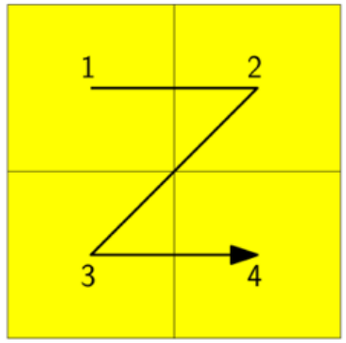


visit quadrants recursively in this order: NW, NE, SW, SE

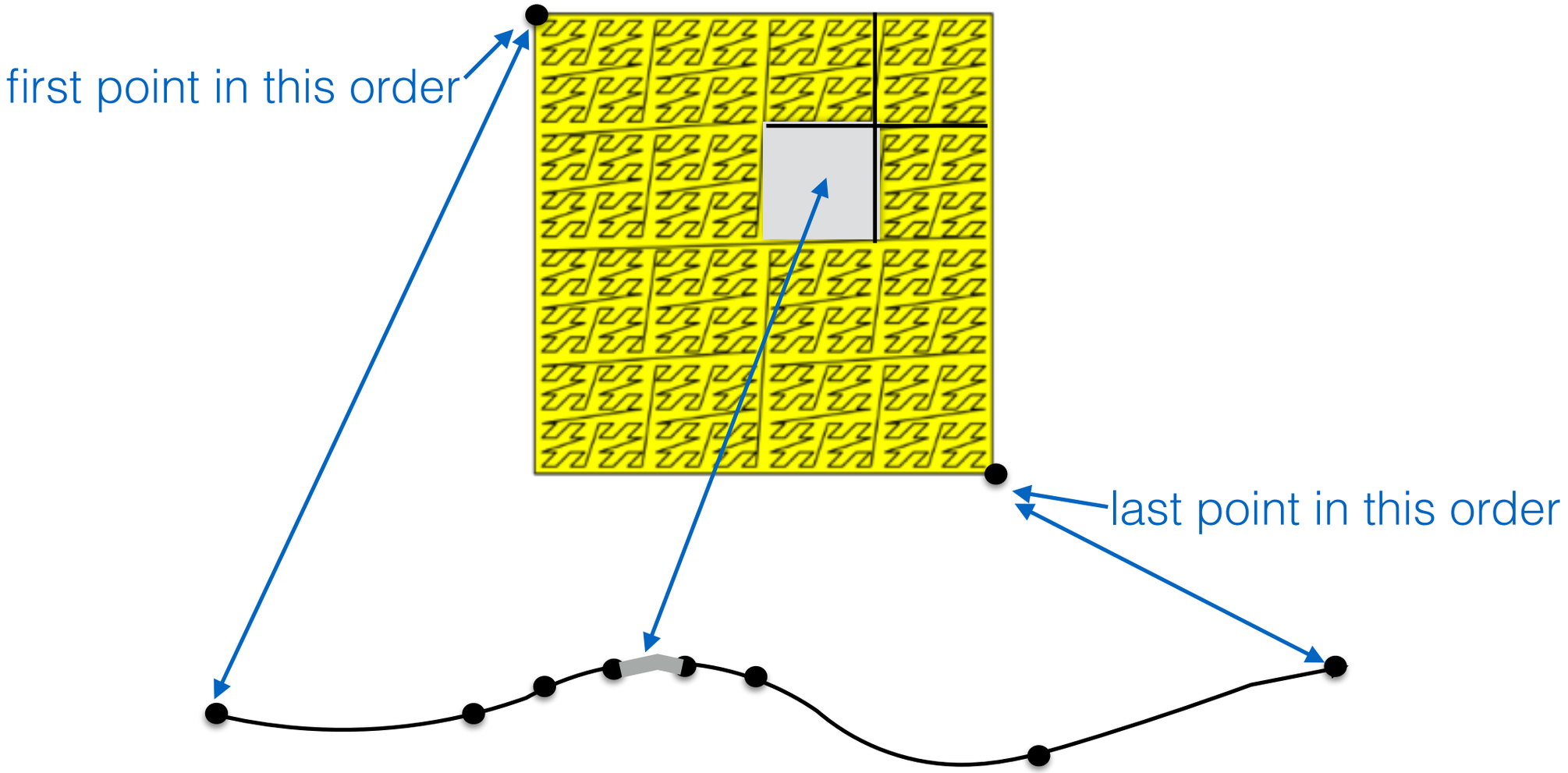


- and so on…..

Z-order and quadtrees

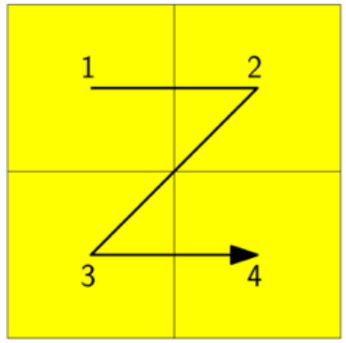


visit quadrants recursively in this order: NW, NE, SW, SE

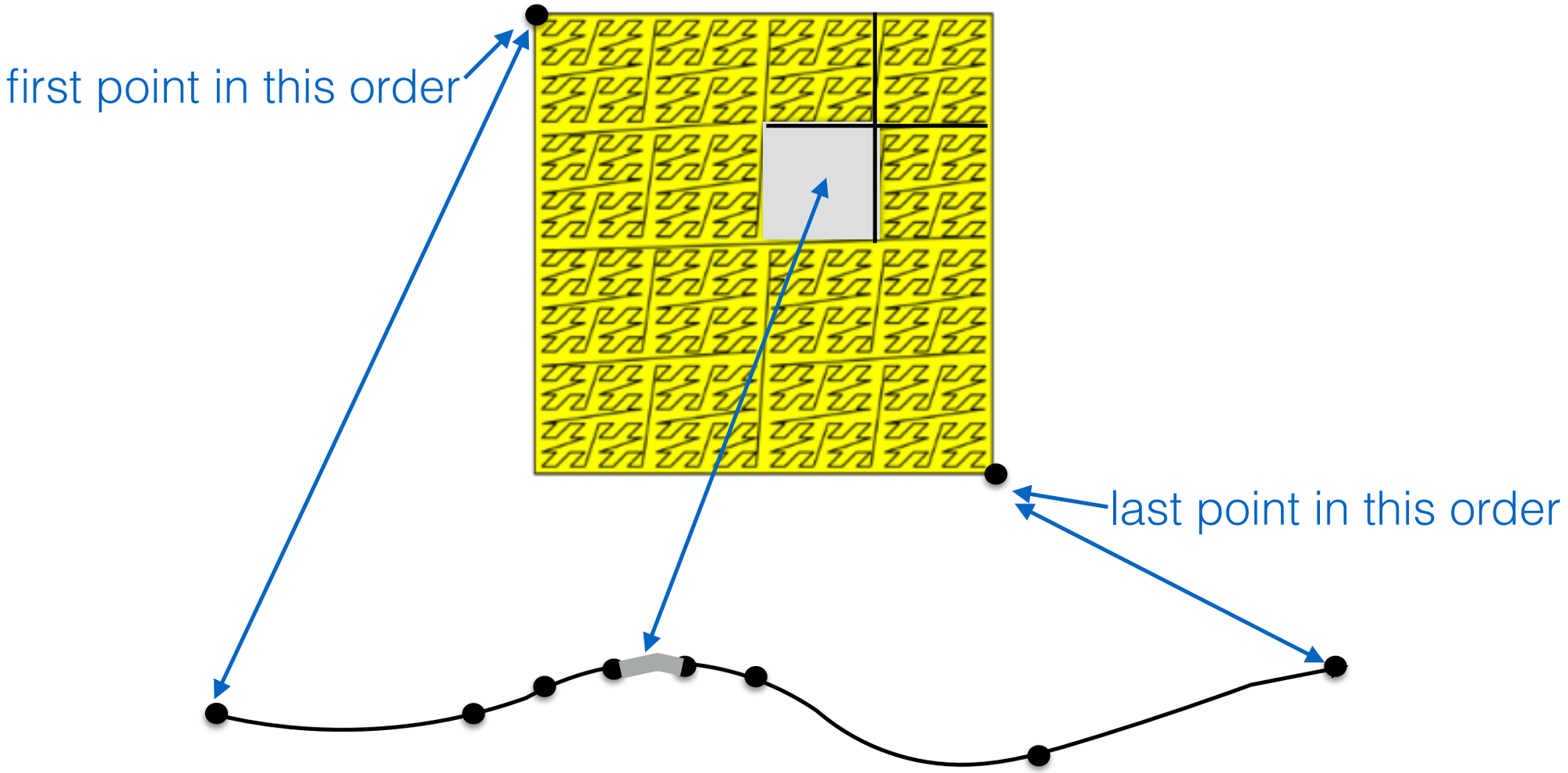


- Every canonical square corresponds to an interval of the z-order curve

Z-order and quadtrees

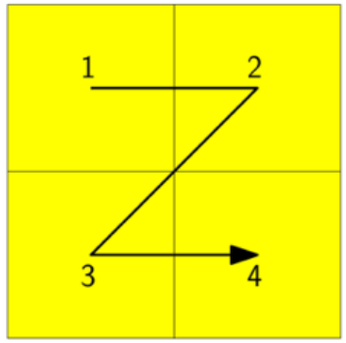


visit quadrants recursively in this order: NW, NE, SW, SE

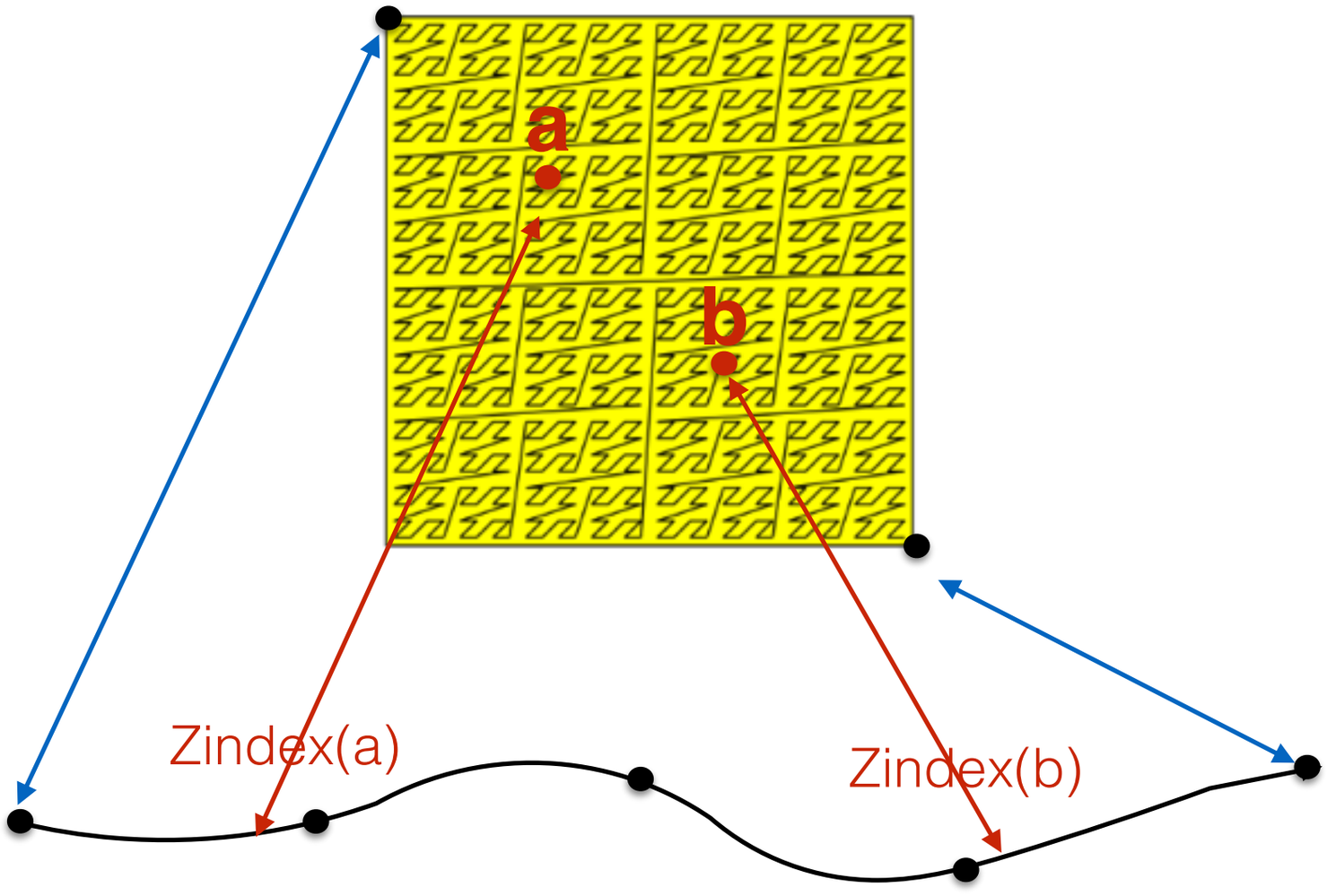


- Two canonical squares are non-intersecting, or one included in the other

Z-order and quadtrees



visit quadrants recursively in this order: NW, NE, SW, SE



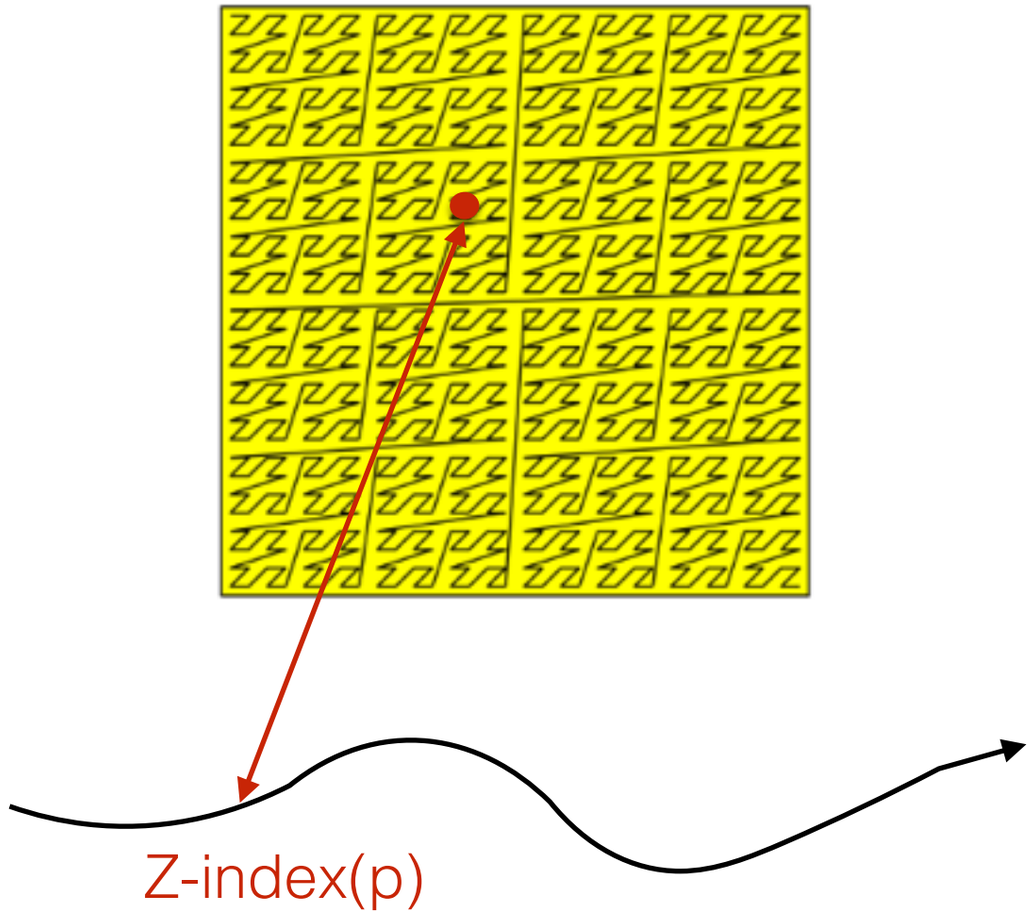
- If Zindex(a) < Zindex(b), we say that a < b
- Any two points a,b can be compared

Computing the Z-index

Z_index : R^2 —> R

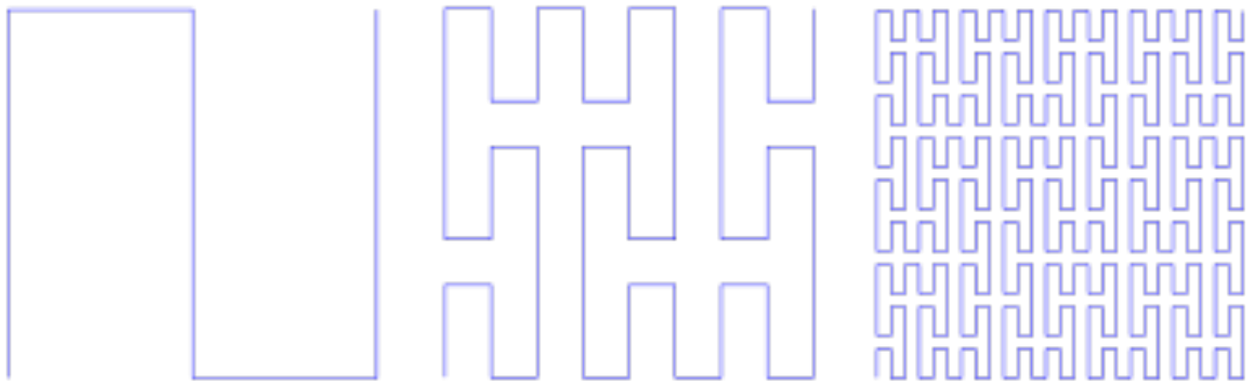
For simplicity assume points with integer coordinates on k bits

- What is the largest integer representable on k bits?



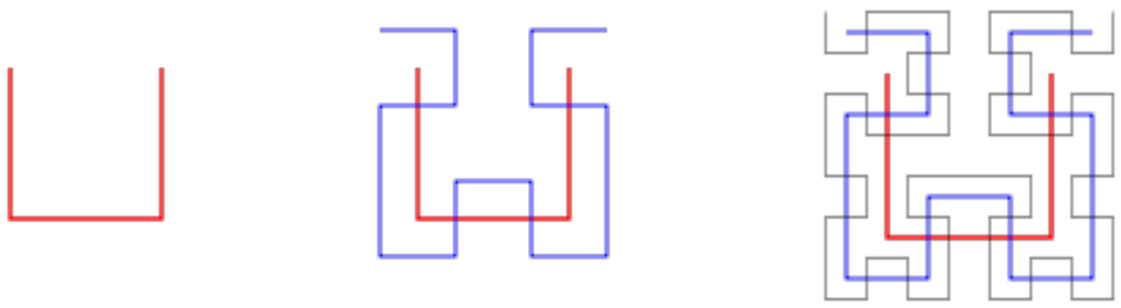
Space-filling curves

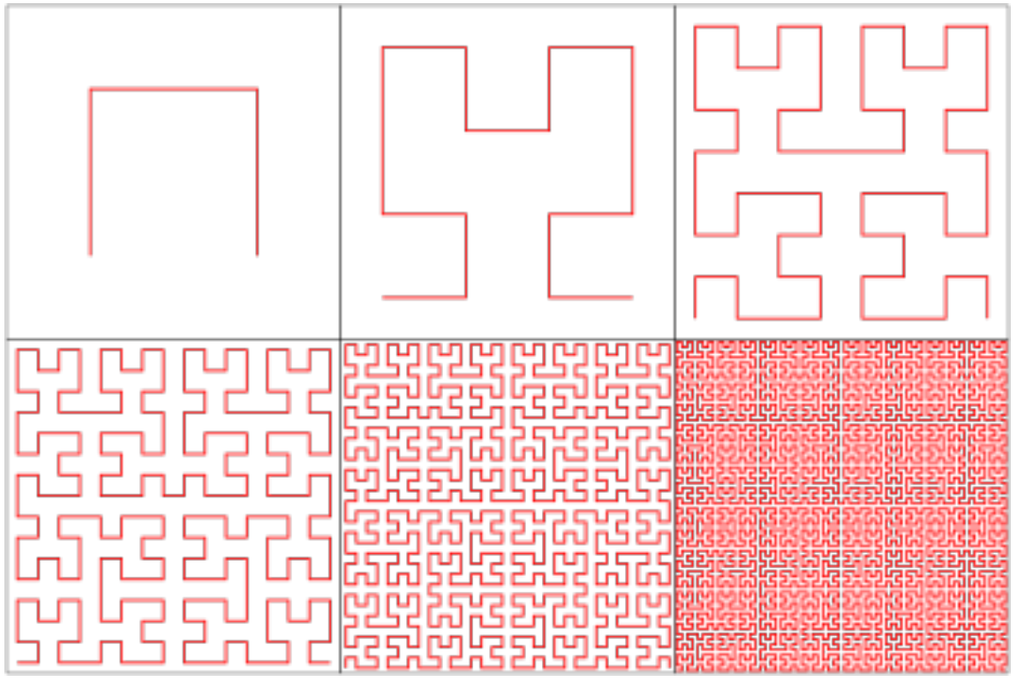
- Z-order curves are a special type of space-filling curves
- First SFC were described by Peano and Hilbert

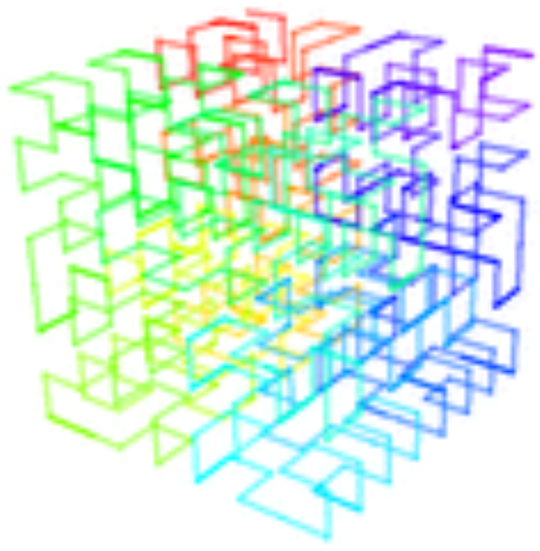


Peano curve

Hilbert curve



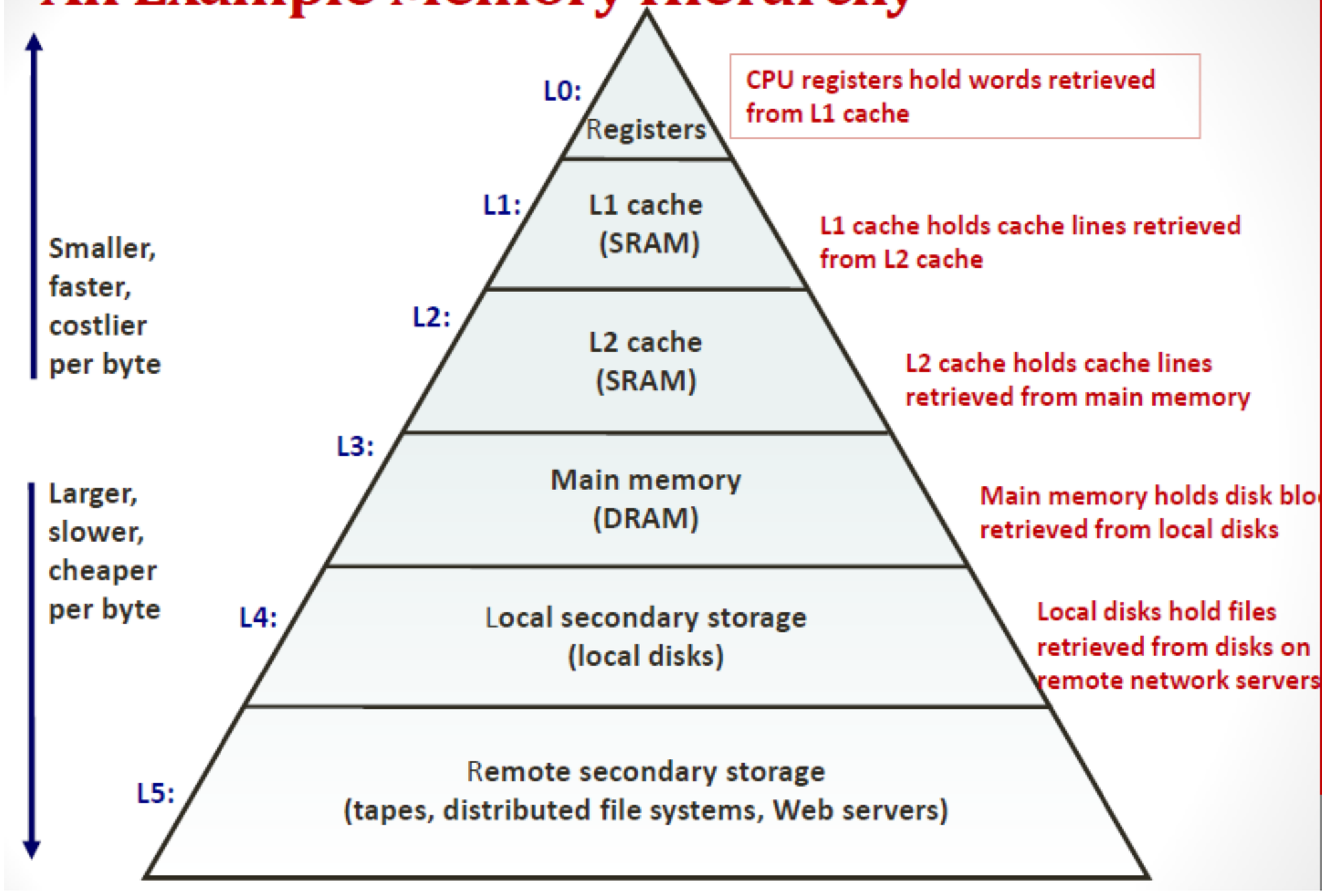




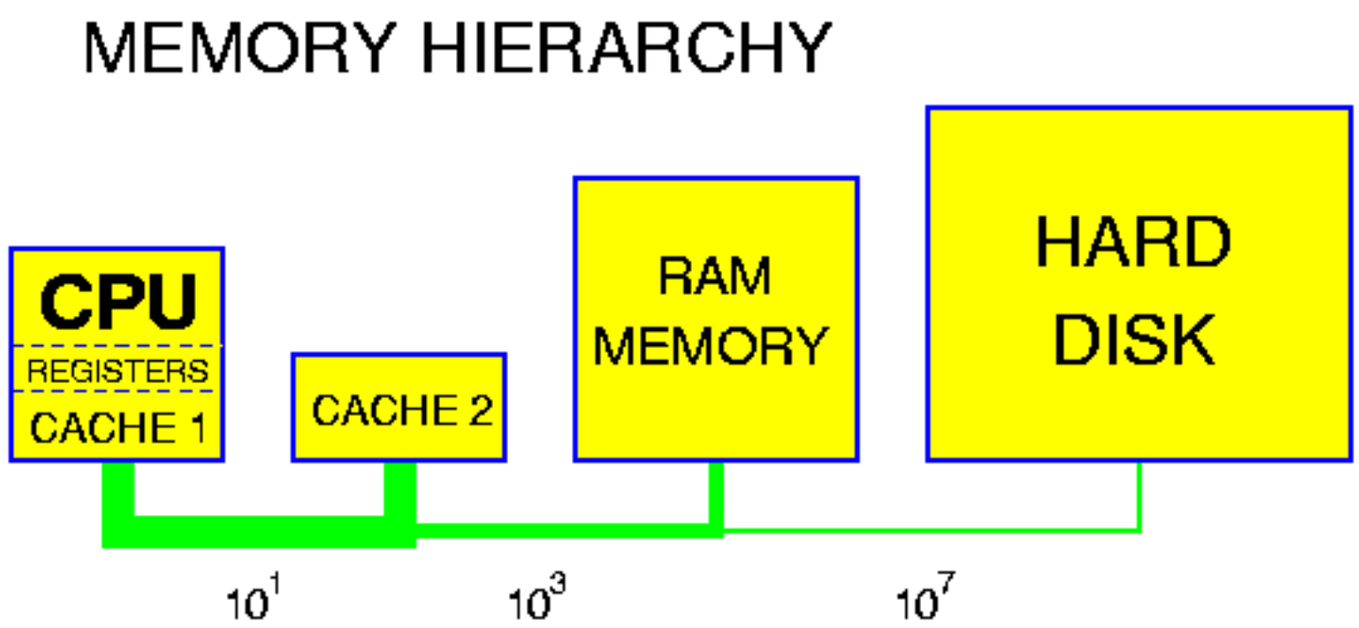
Spatial locality

- Big-Oh analysis does not have the final word
- Two algorithms that have the same big-Oh can differ a lot in performance depending on their cache efficiency
- To analyze and fine tune the algorithm we need to look at the performance across all levels of the memory hierarchy

An Example Memory Hierarchy

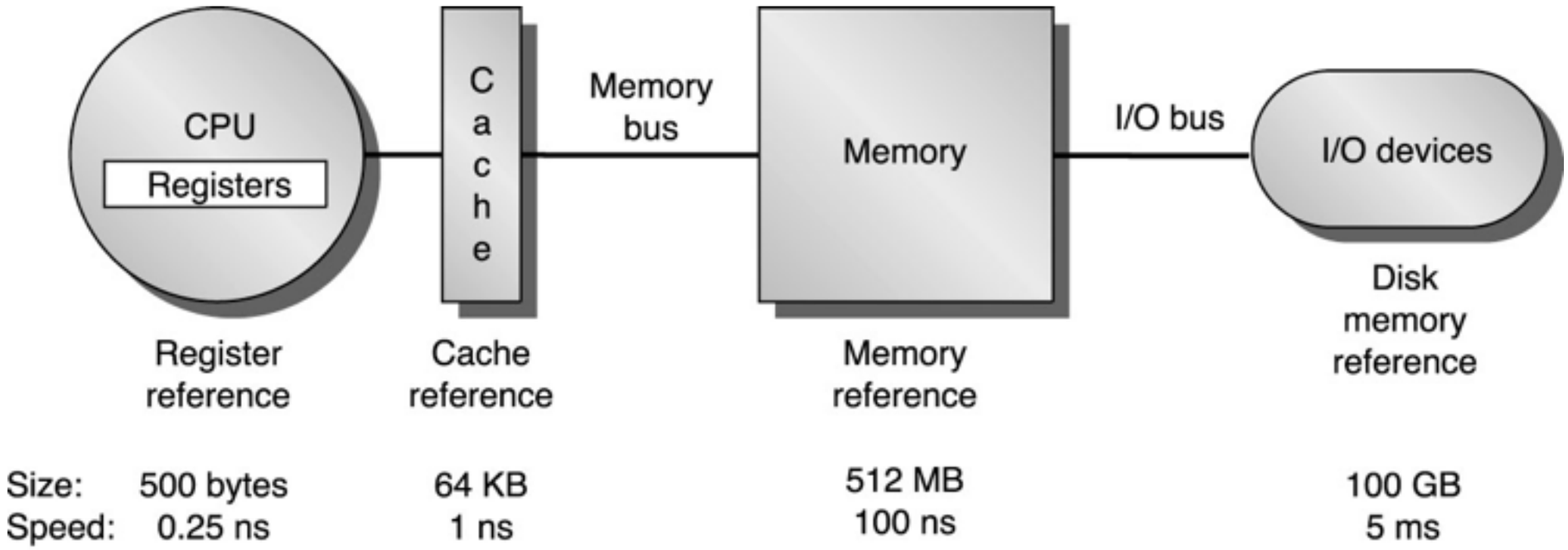


http://classconnection.s3.amazonaws.com/149/flashcards/3088149/png/memory_hierarchy1367201501848.png



Indicated are approximate numbers of clock cycles to access the various elements of the memory hierarchy

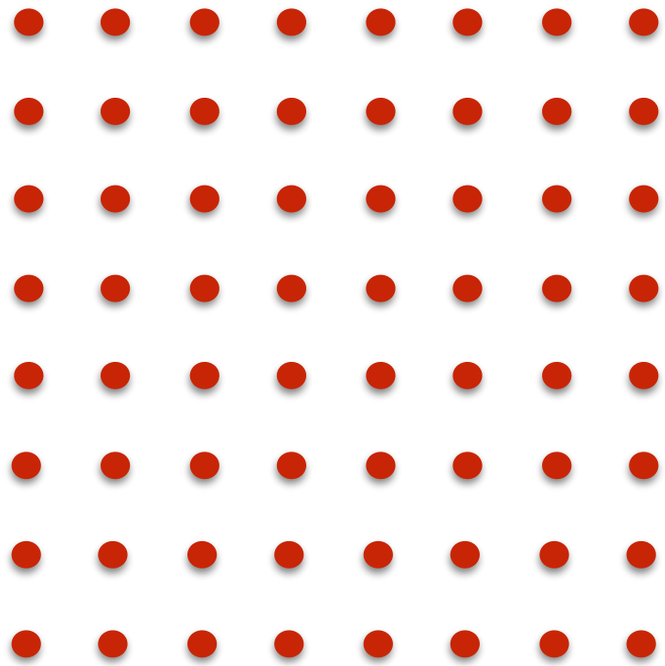
http://users.informatik.uni-halle.de/~jopsi/dinf104/memory_hierarchy.gif



- At all levels, data is organized and moved in blocks/pages
- Each level acts as a “cache” for the next level: stores most recently used blocks
- Applications that access data that’s stored in a “recent” block will find it in cache
  - 1ns vs 100ns <—- SIGNIFICANT!

Spatial locality

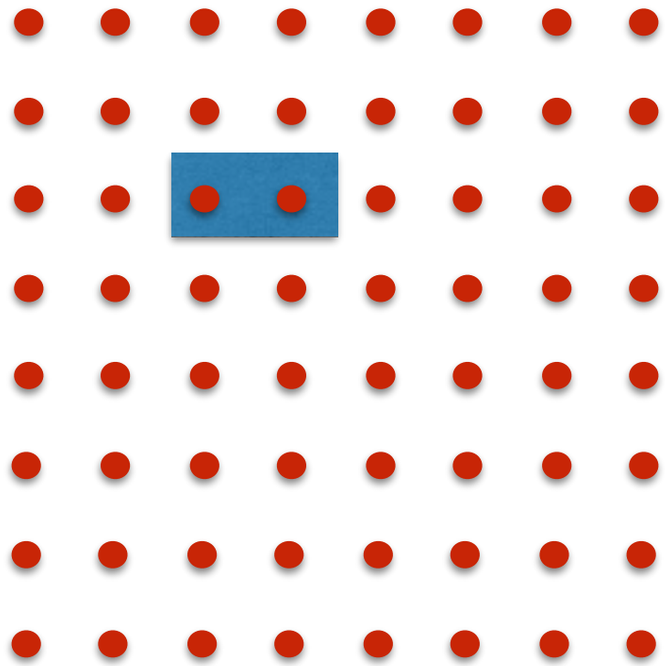
Spatial applications usually have spatial locality in their access to data, i.e. they are likely to access together points that are close to each other in space



We would like points “close” in 2D to be stored “close” to each other in the data structure

Spatial locality

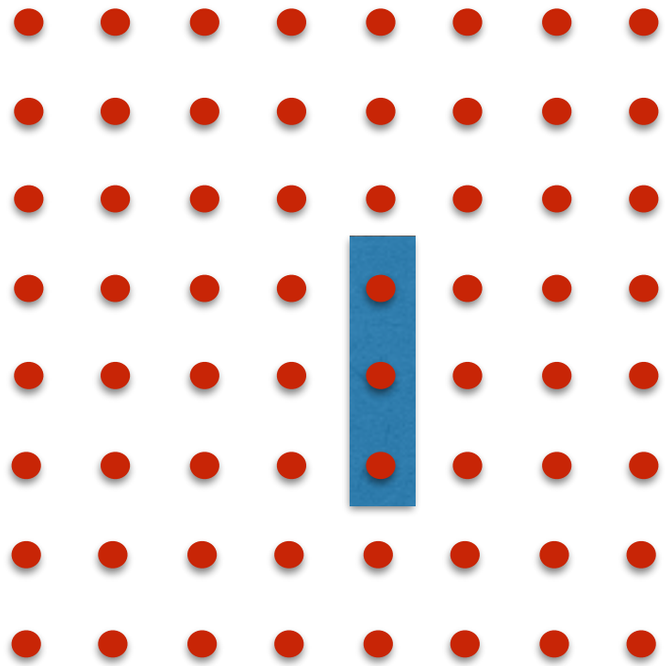
Spatial applications usually have spatial locality in their access to data, i.e. they are likely to access together points that are close to each other in space



We would like points “close” in 2D to be stored “close” to each other in the data structure

Spatial locality

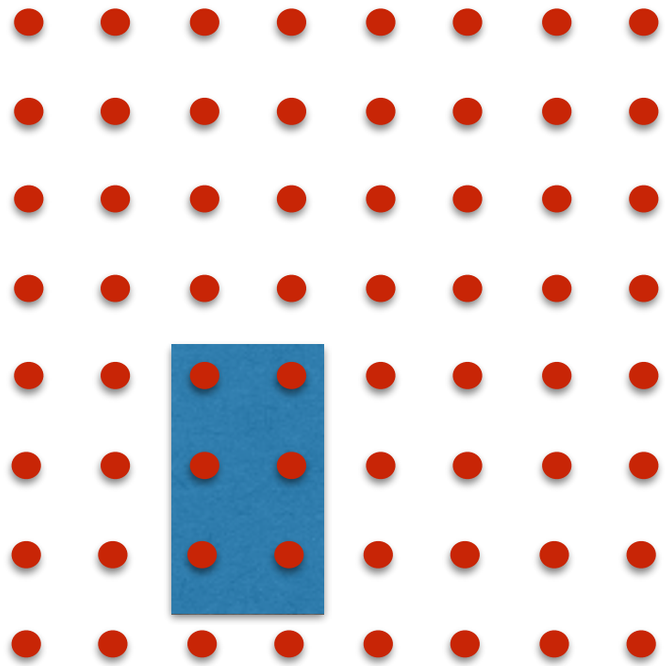
Spatial applications usually have spatial locality in their access to data, i.e. they are likely to access together points that are close to each other in space



We would like points “close” in 2D to be stored “close” to each other in the data structure

Spatial locality

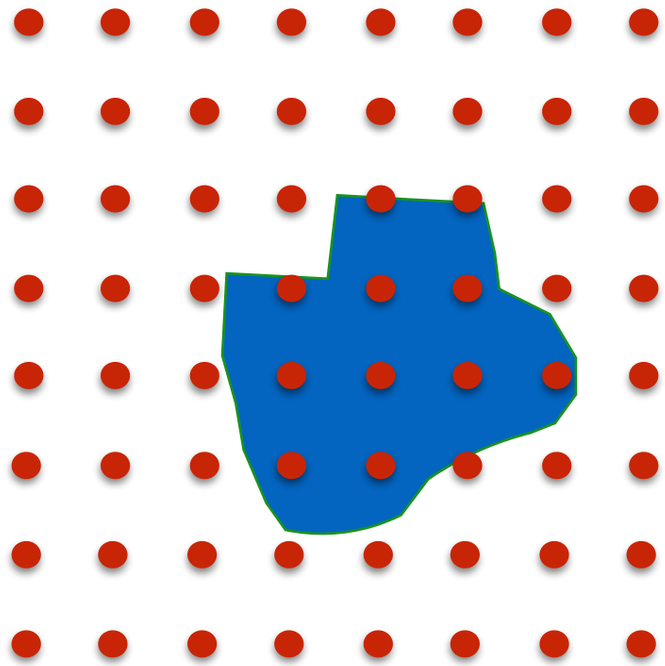
Spatial applications usually have spatial locality in their access to data, i.e. they are likely to access together points that are close to each other in space



We would like points “close” in 2D to be stored “close” to each other in the data structure

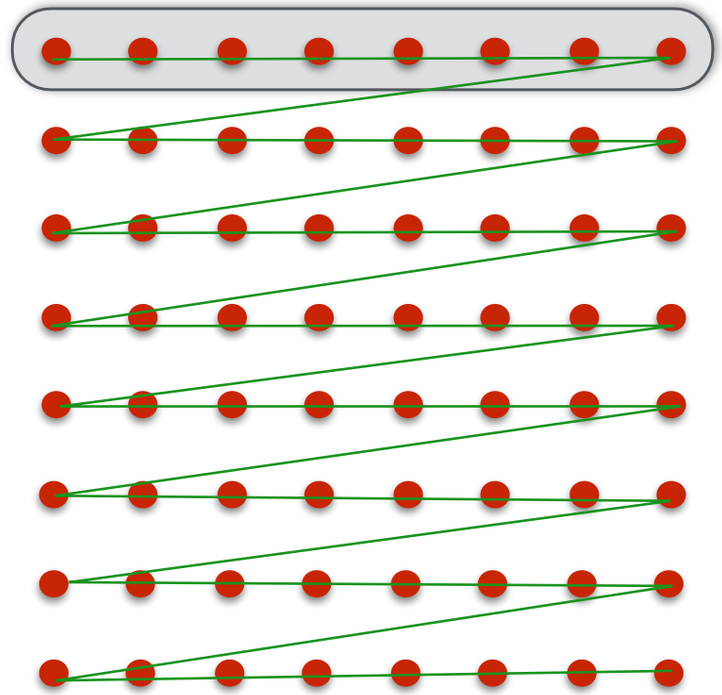
Spatial locality

Spatial applications usually have spatial locality in their access to data, i.e. they are likely to access together points that are close to each other in space

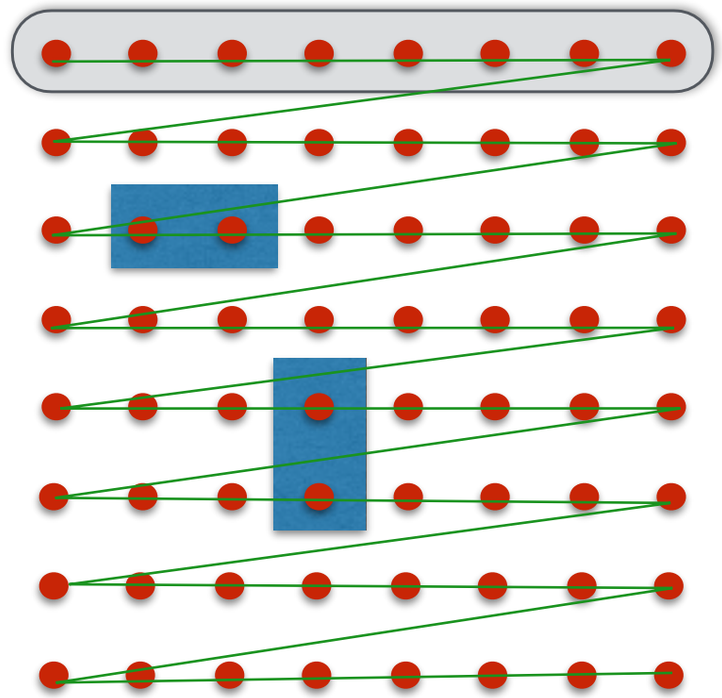


We would like points “close” in 2D to be stored “close” to each other in the data structure

Grid default layout: row-major order

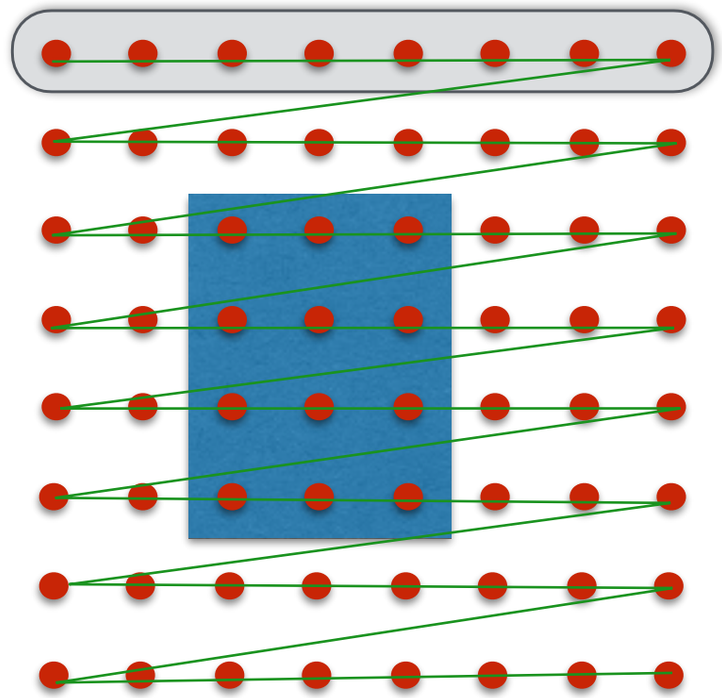


Grid default layout: row-major order



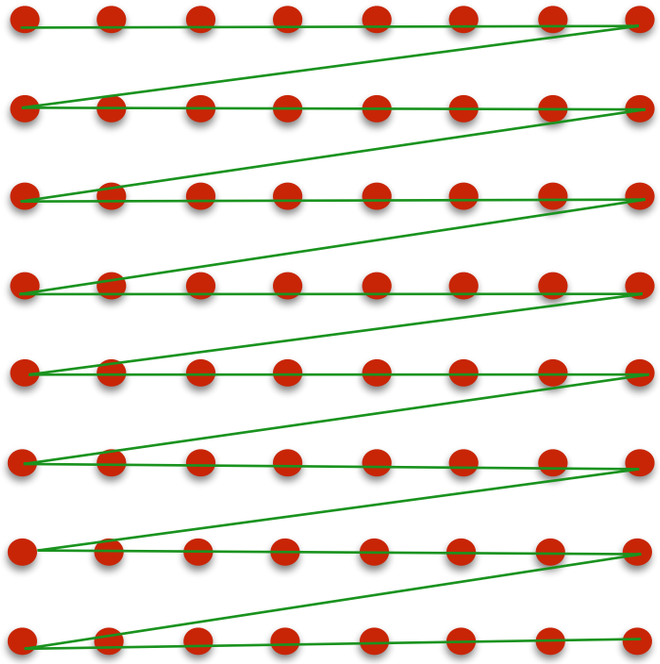
- Does this layout have good spatial locality?
- How far are these points in the array?

Grid default layout: row-major order

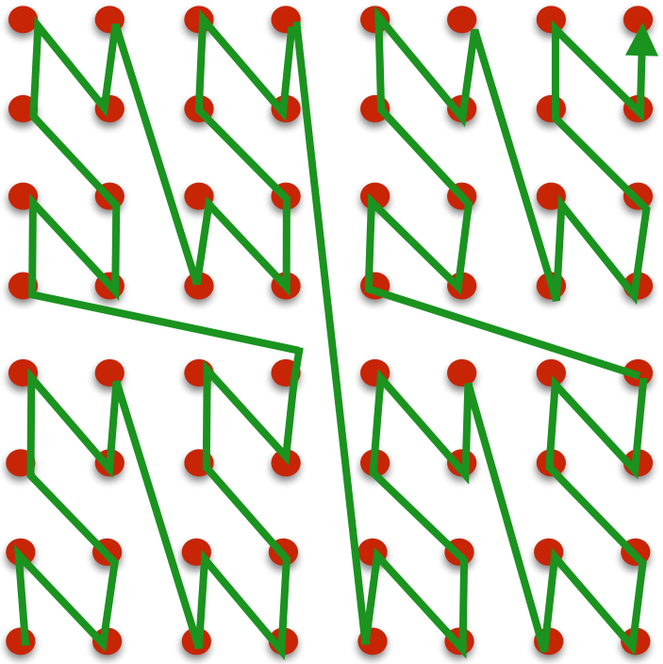


- Does this layout have good spatial locality?
- How far are these points in the array?

Spatial locality

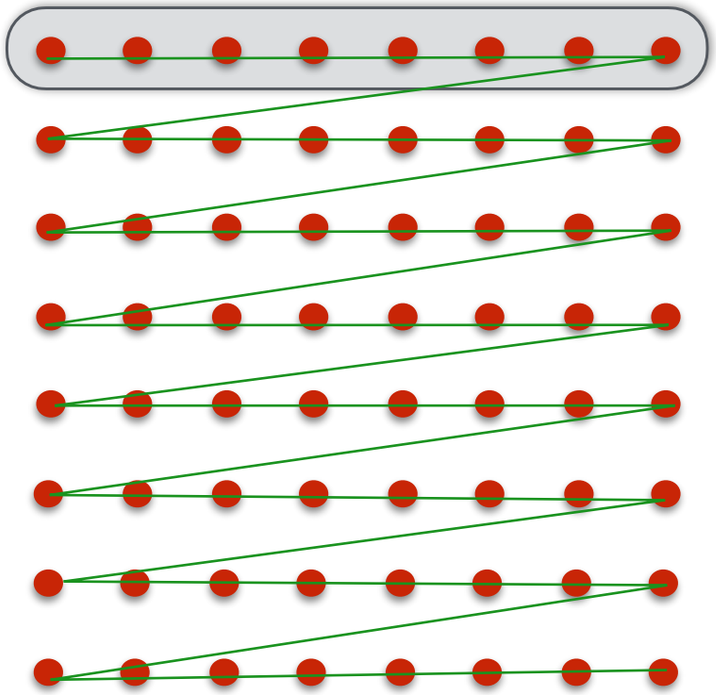


grid in default row-major order

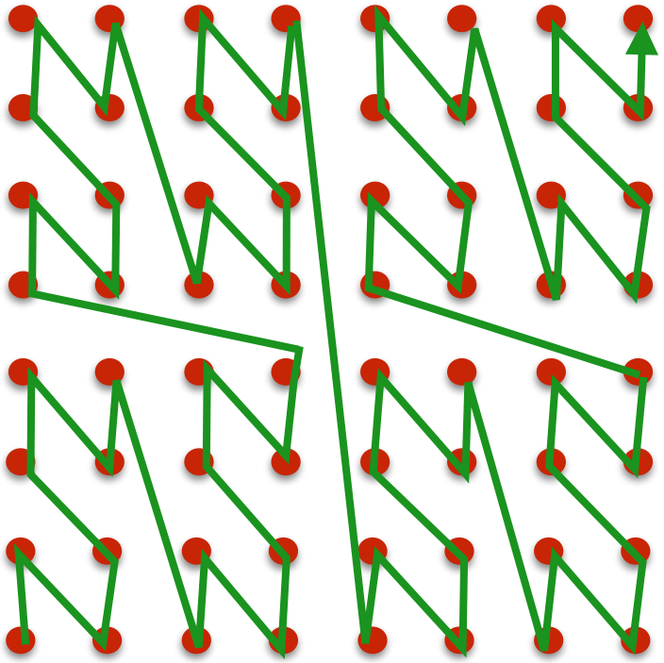


grid stored in Z-order

Spatial locality

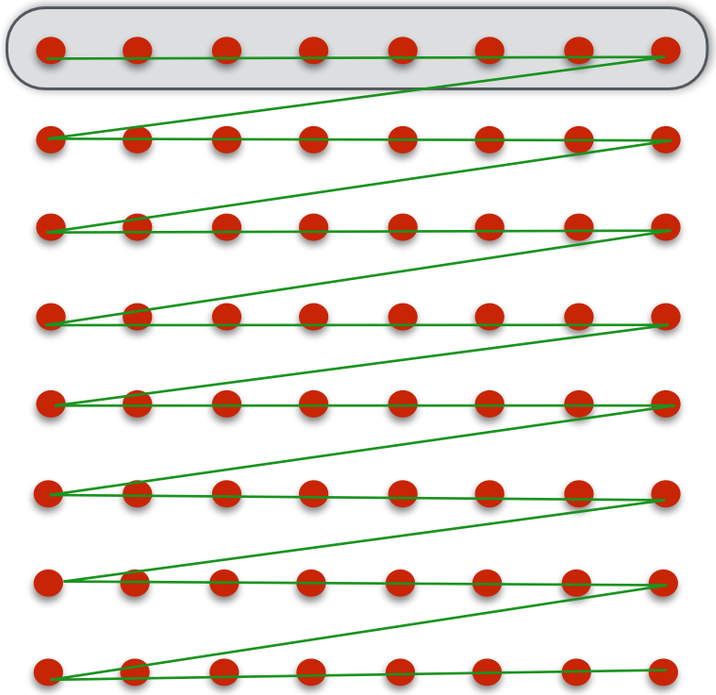


grid in default row-major order

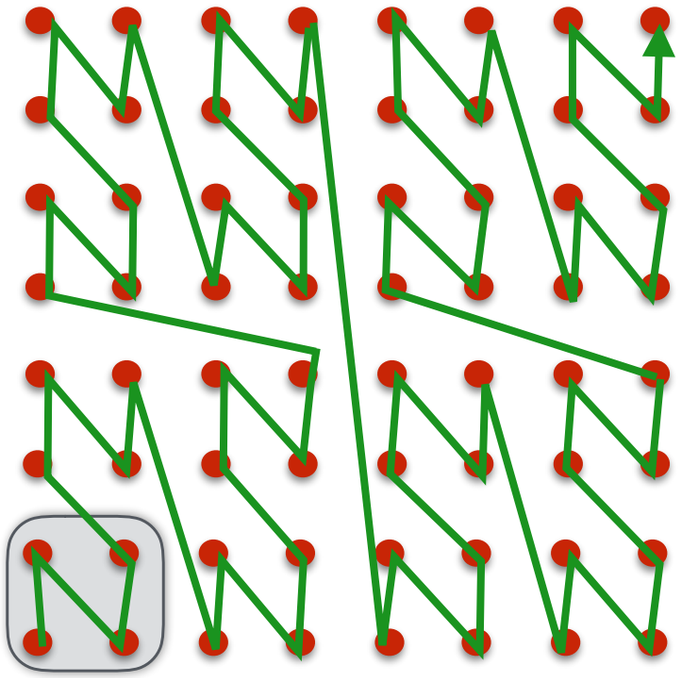


grid stored in Z-order

Spatial locality

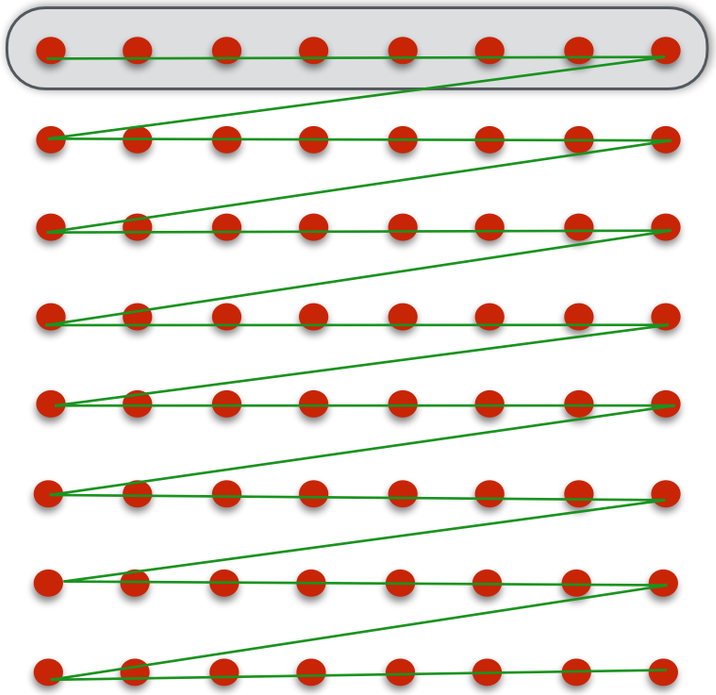


grid in default row-major order

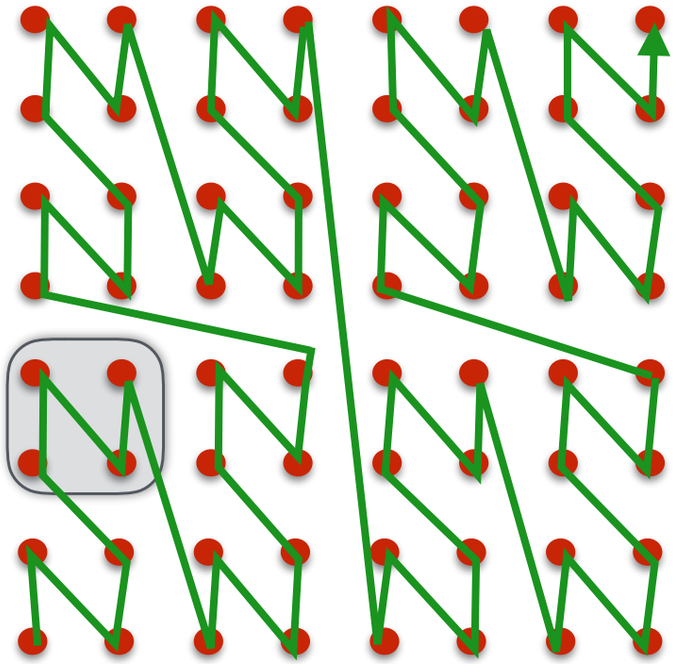


grid stored in Z-order

Spatial locality

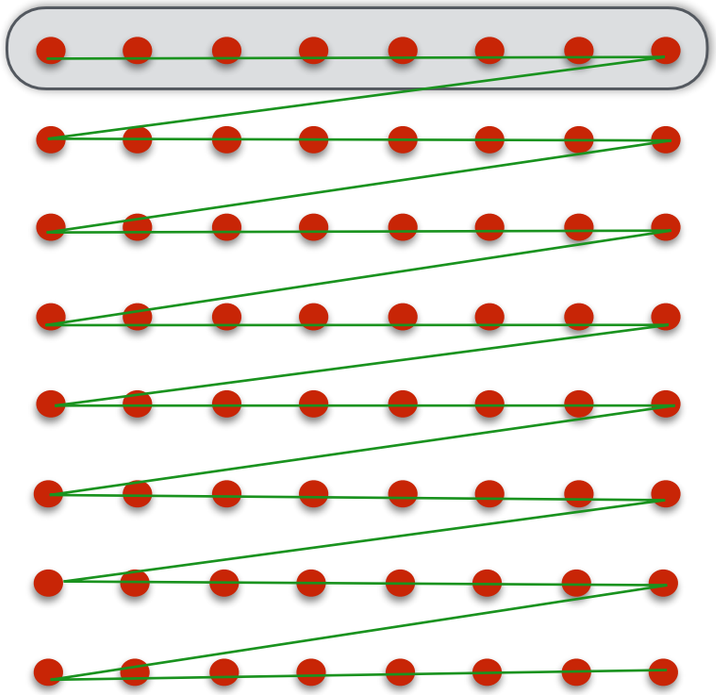


grid in default row-major order

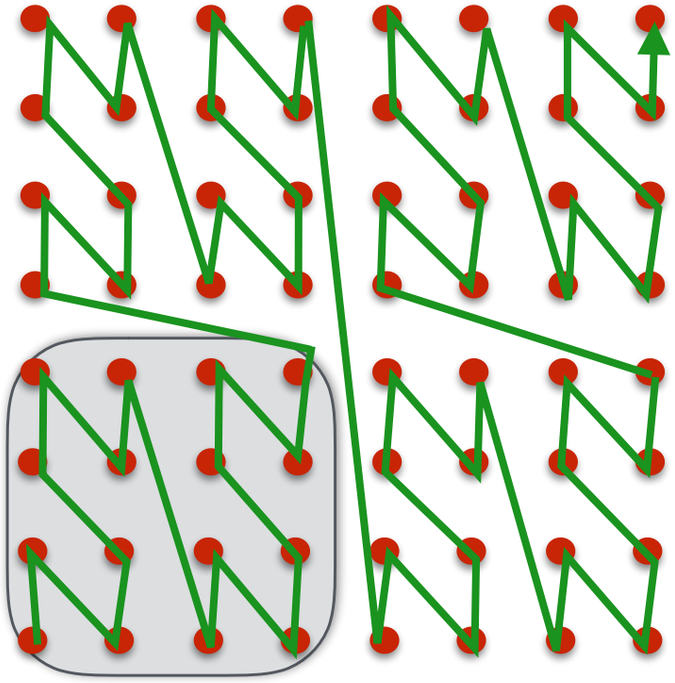


grid stored in Z-order

Spatial locality

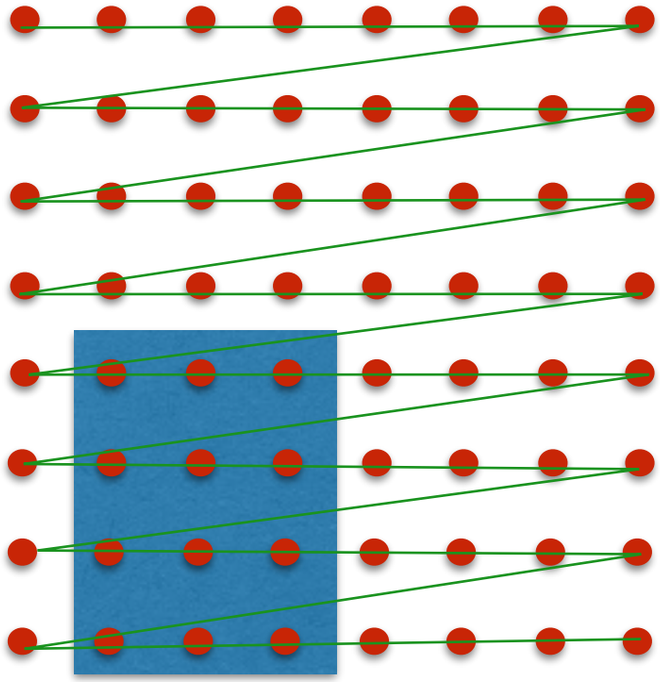


grid in default row-major order

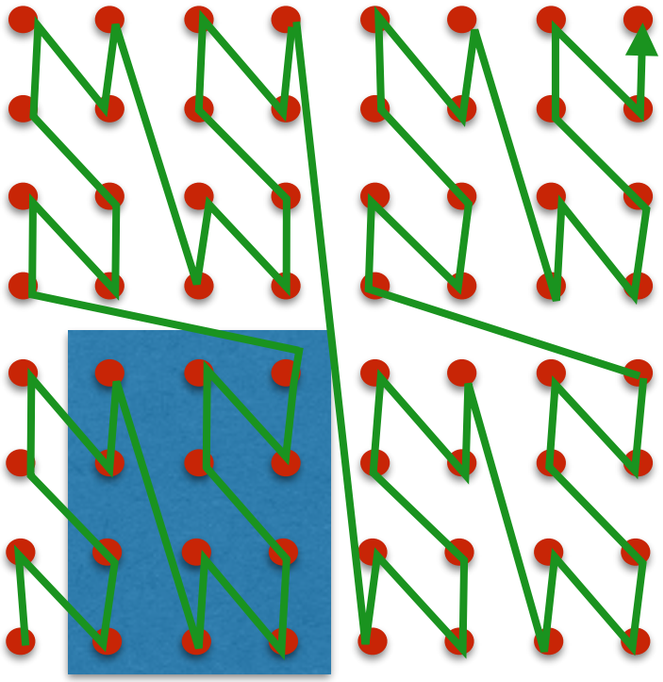


grid stored in Z-order

Spatial locality

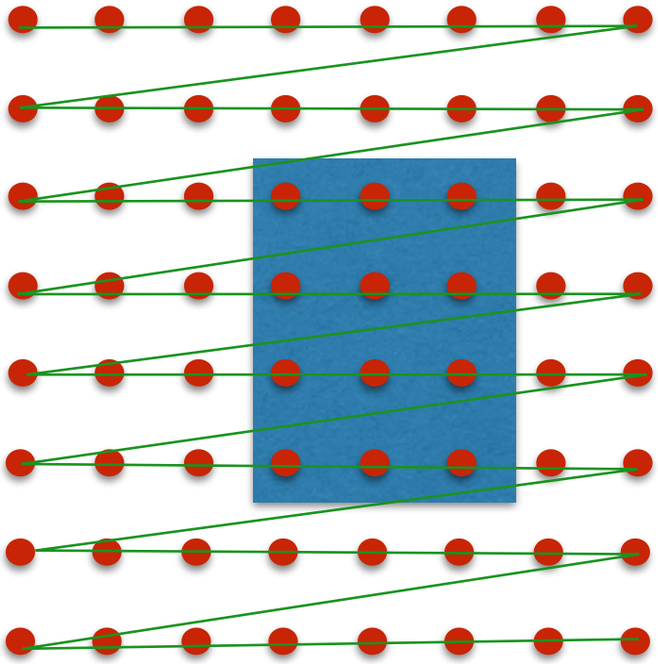


grid in default row-major order

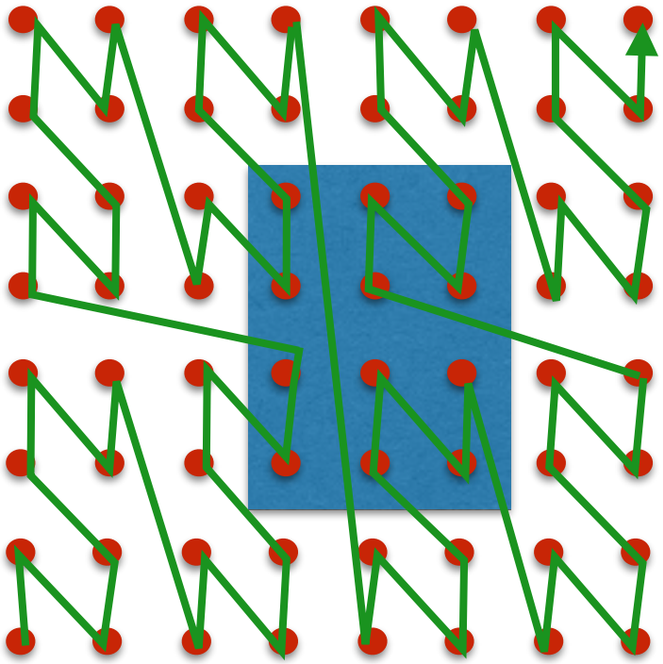


grid stored in Z-order

Spatial locality

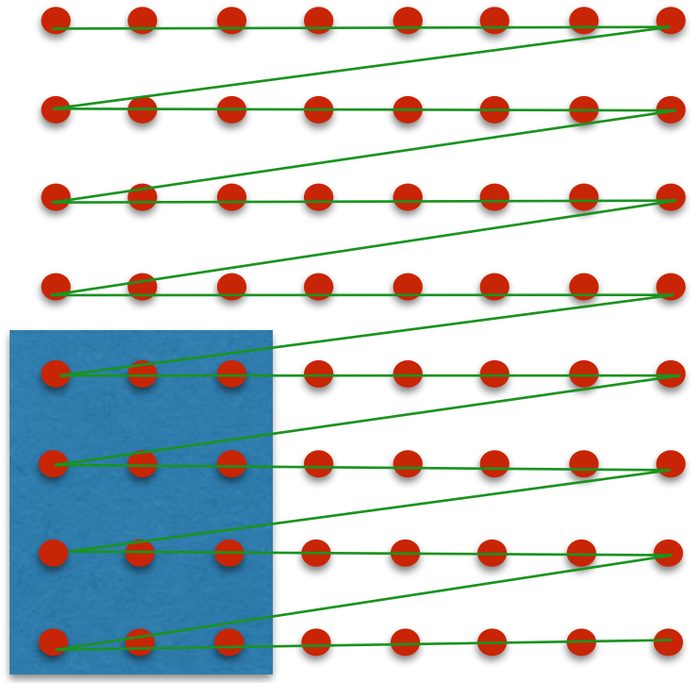


grid in default row-major order

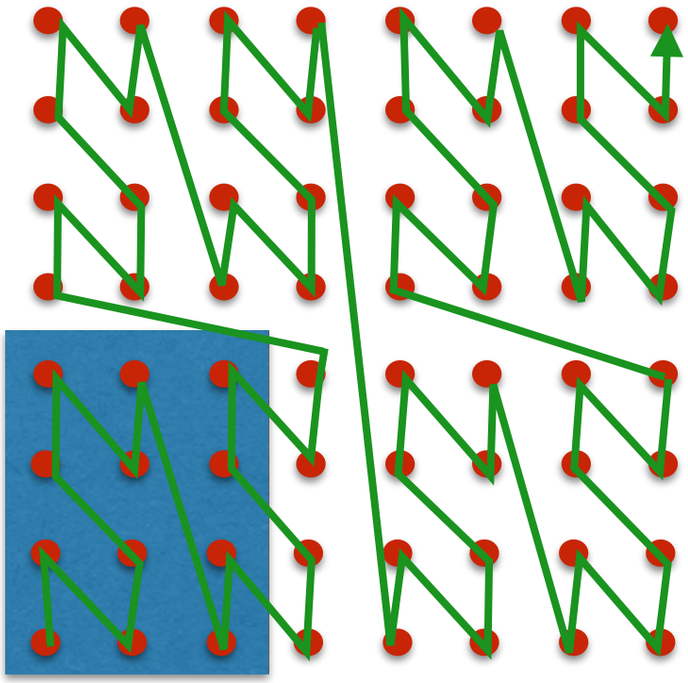


grid stored in Z-order

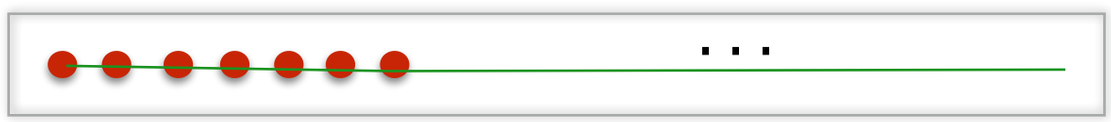
Spatial locality



grid in default row-major order



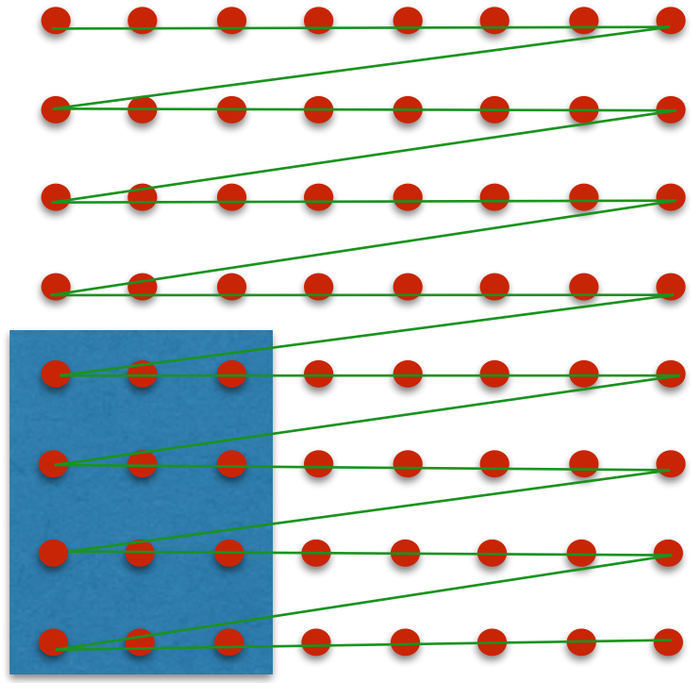
grid stored in Z-order



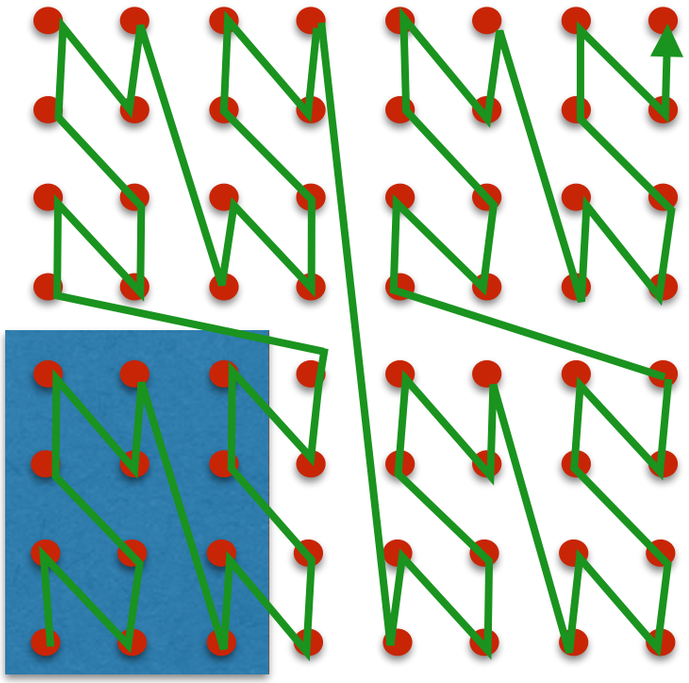
Spatial locality

Arranging data in order of a space-filling curve improves spatial locality

- points that are close together in space, will be stored close to each other



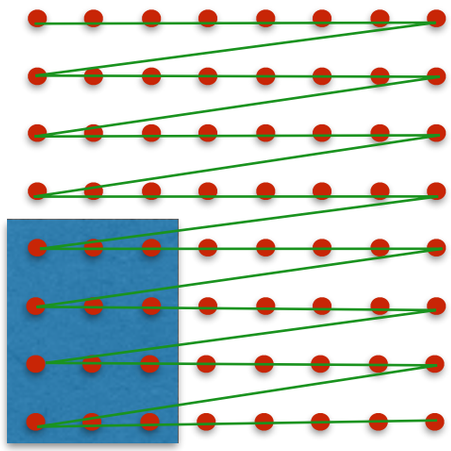
grid in default row-major order

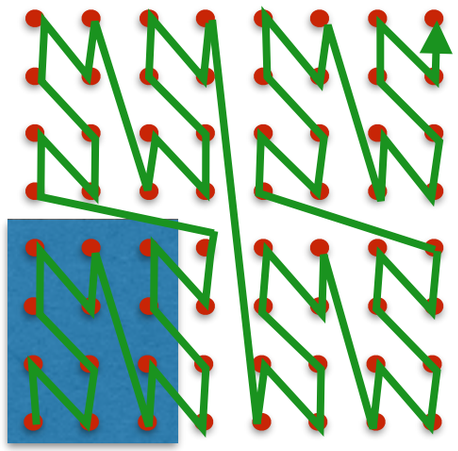


grid stored in Z-order

Spatial locality

- Arranging data in order of a space-filling curve improves spatial locality
  - points that are close together in space, will be stored close to each other
  - data will be in the same blocks as previous data
  - data will be found in cache
  - improvements at all levels of the memory hierarchy





- Hilbert curve has better locality than z-order, but slower to compute
- Z-order used with Strassen's algorithm —> speedups (2002)

SFC in art

Don Relyea, artist futurist and tehnologist

- http://www.donrelyea.com/site2015/space-filling-curve-art-2004-2014-wide-format/