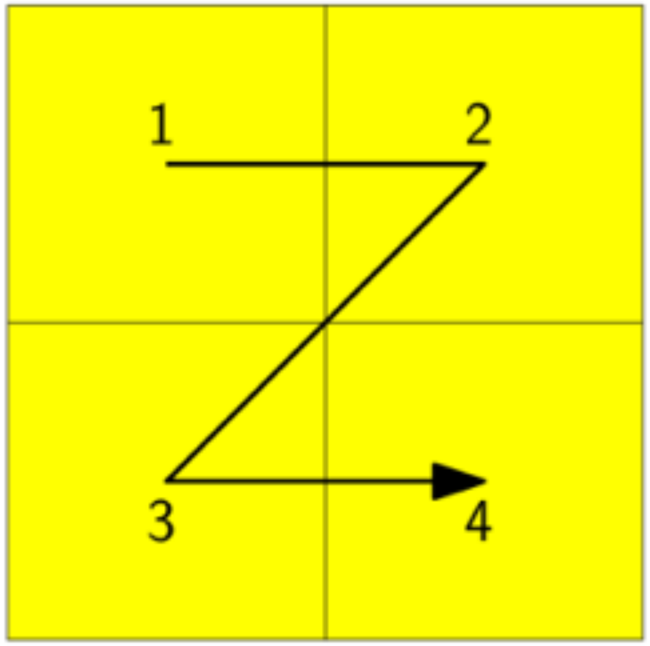
Algorithms for GIS:

Space filling curves

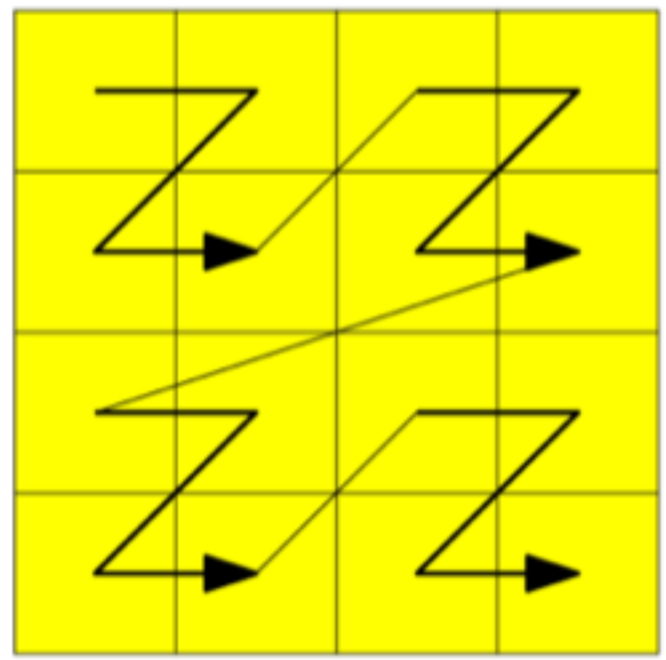
Z-order

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



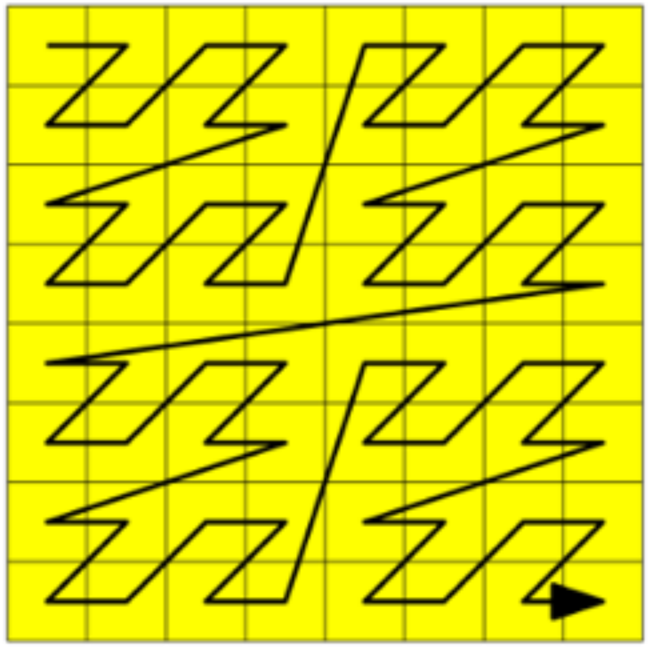
Z-order

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



Z-order

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



Z-order

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

Z-order

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

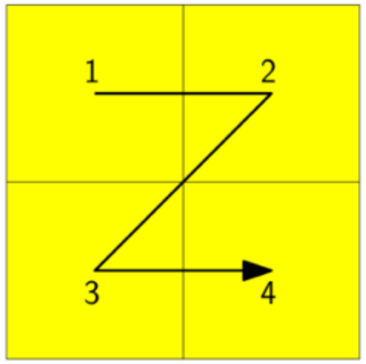
- At the limit, it will reach all points in the square ==> space filling curve

Z-order

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

Where is the very first point visited?

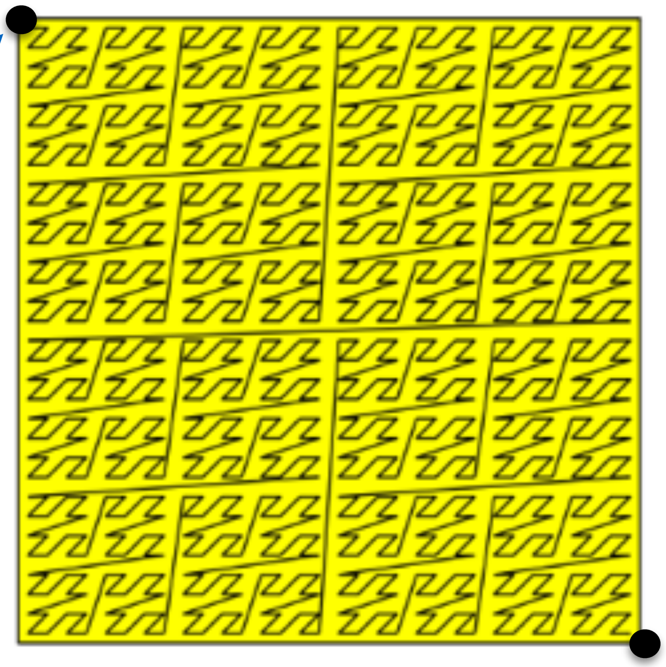
- At the limit, it will reach all points in the square ==> space filling curve



Z-order

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

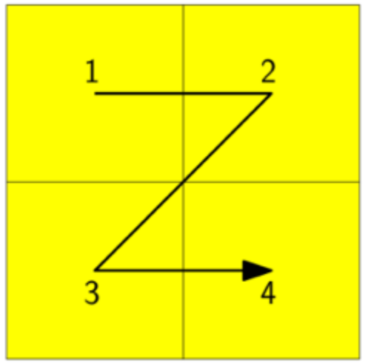
first point in this order



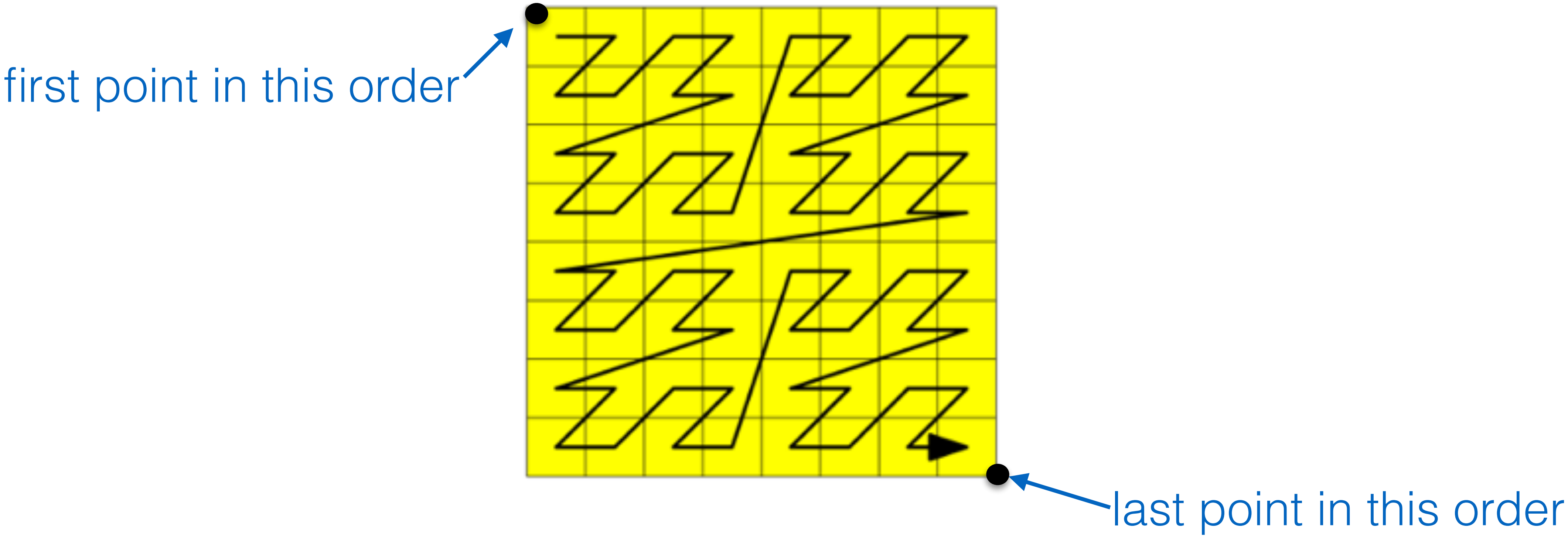
last point in this order

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

Z-order

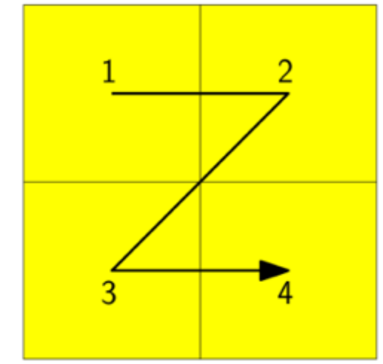


visit quadrants recursively in this order: NW, NE, SW, SE

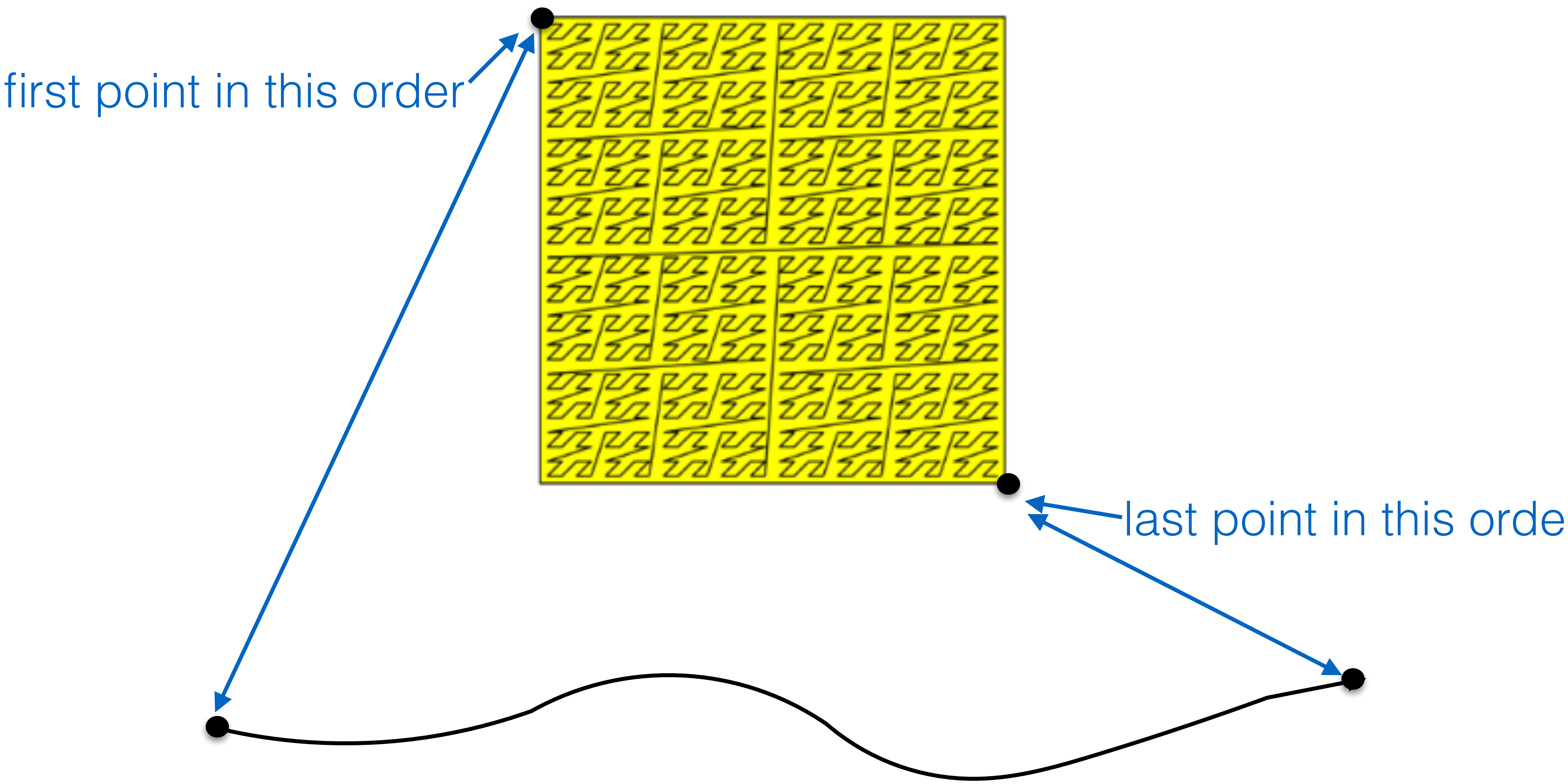


- At the limit, it will reach all points in the square ==> space filling curve

Z-order

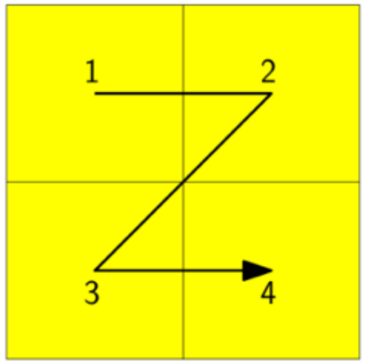


visit quadrants recursively in this order: NW, NE, SW, SE

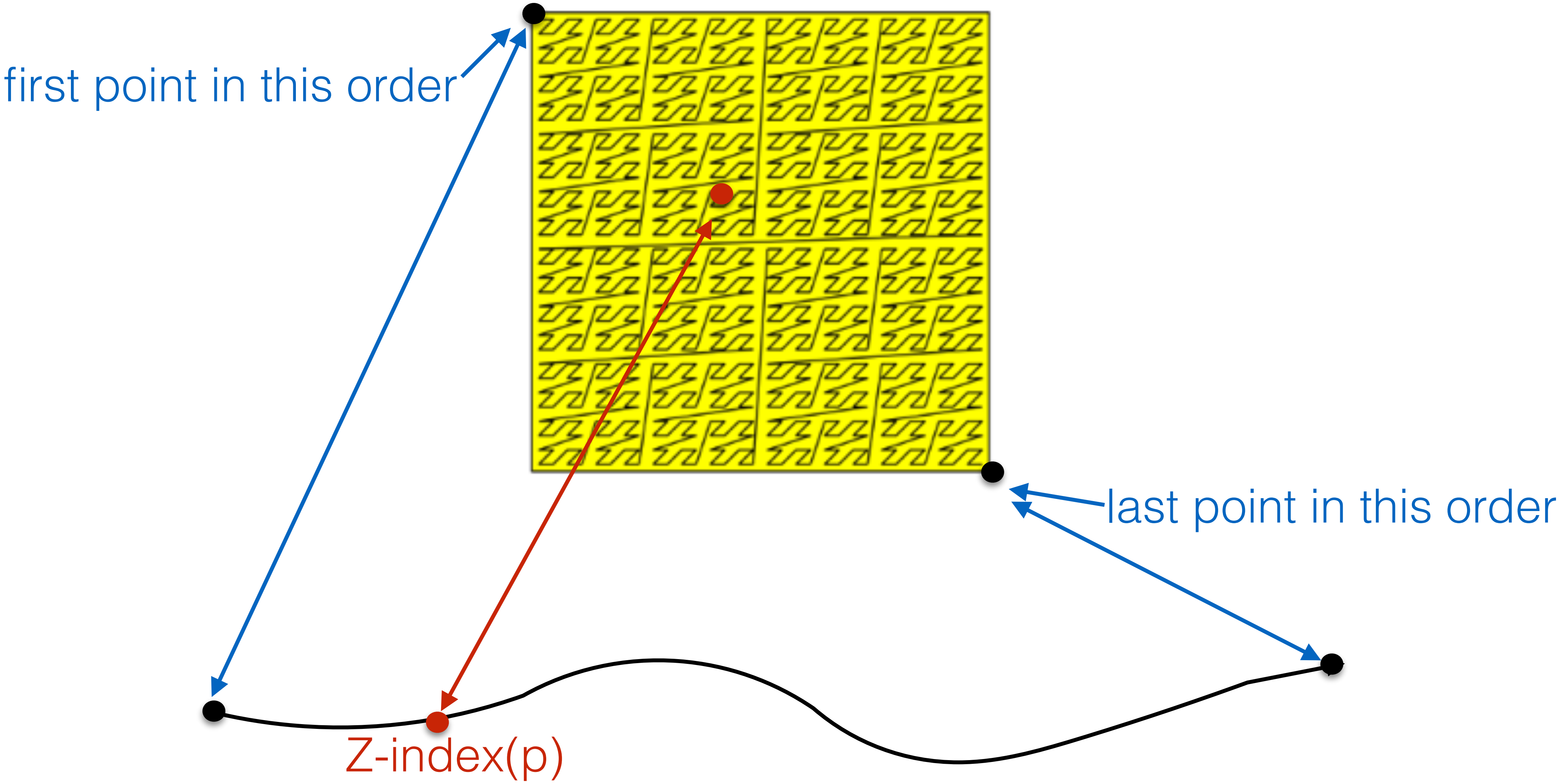


- At the limit, it will reach all points in the square ==> space filling curve
  - Every point in the square will be visited by this curve
  - 2D ==> 1D

Z-order

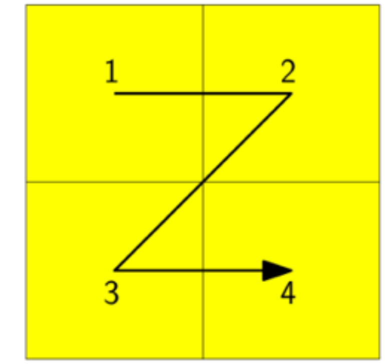


visit quadrants recursively in this order: NW, NE, SW, SE

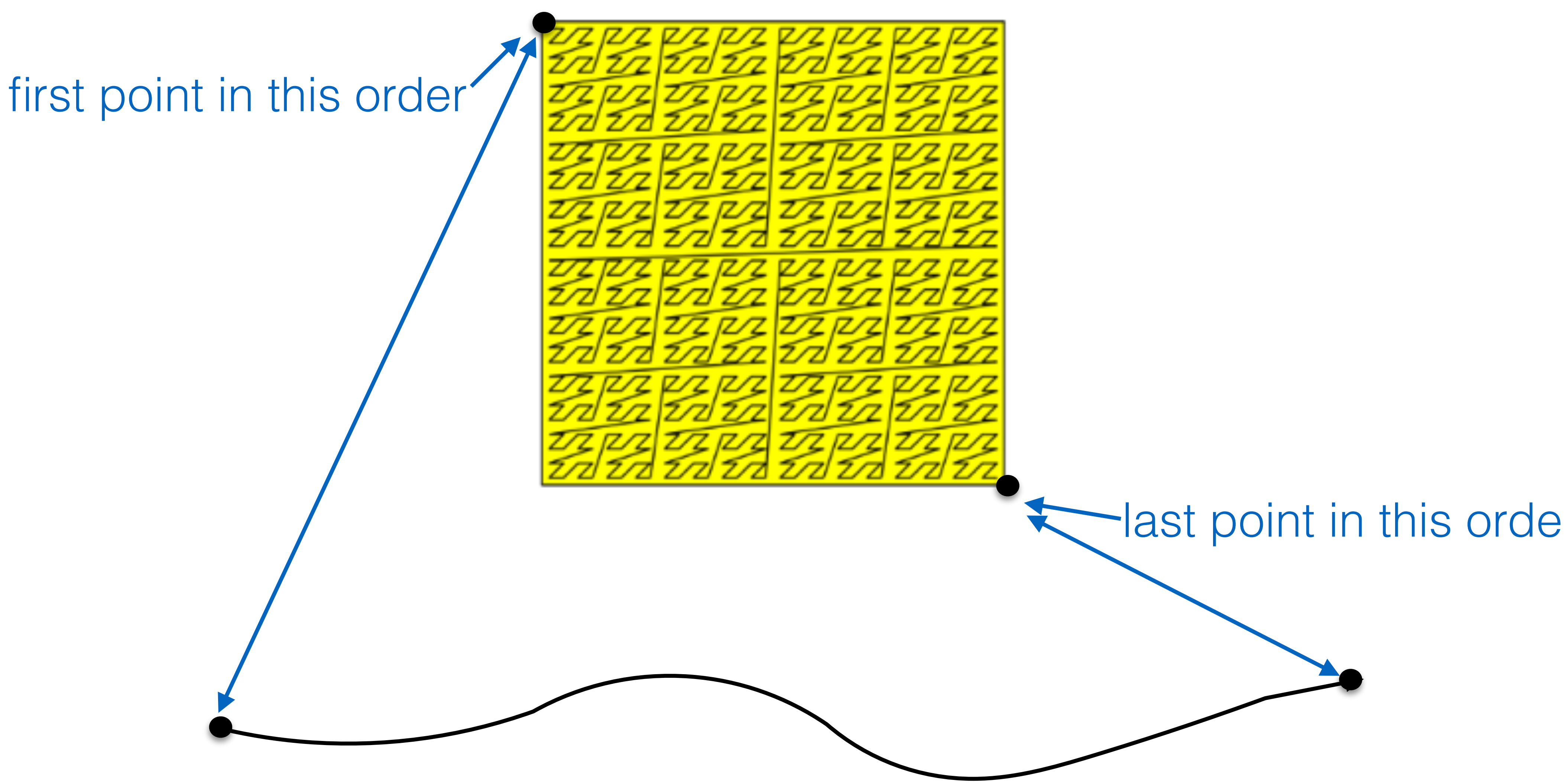


- At the limit, it will reach all points in the square ==> space filling curve
  - Every point in the square will be visited by this curve
  - 2D ==> 1D

Z-order



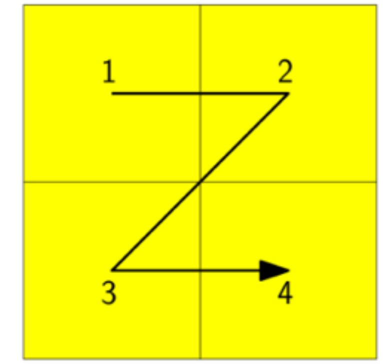
visit quadrants recursively in this order: NW, NE, SW, SE



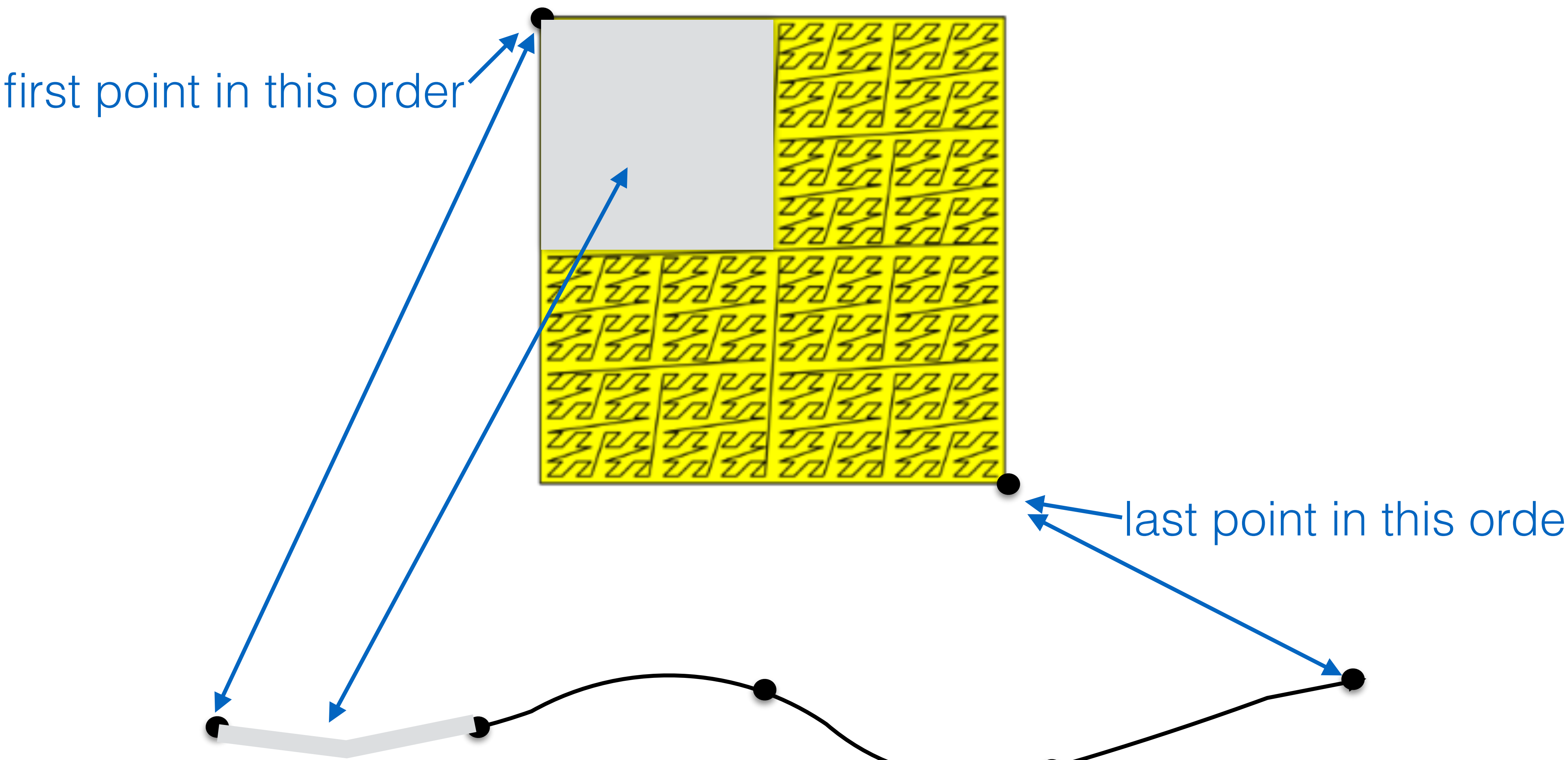
- We visit quadrant 1 before we visit quadrant 2:

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

Z-order



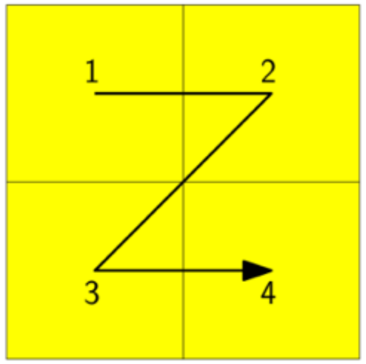
visit quadrants recursively in this order: NW, NE, SW, SE



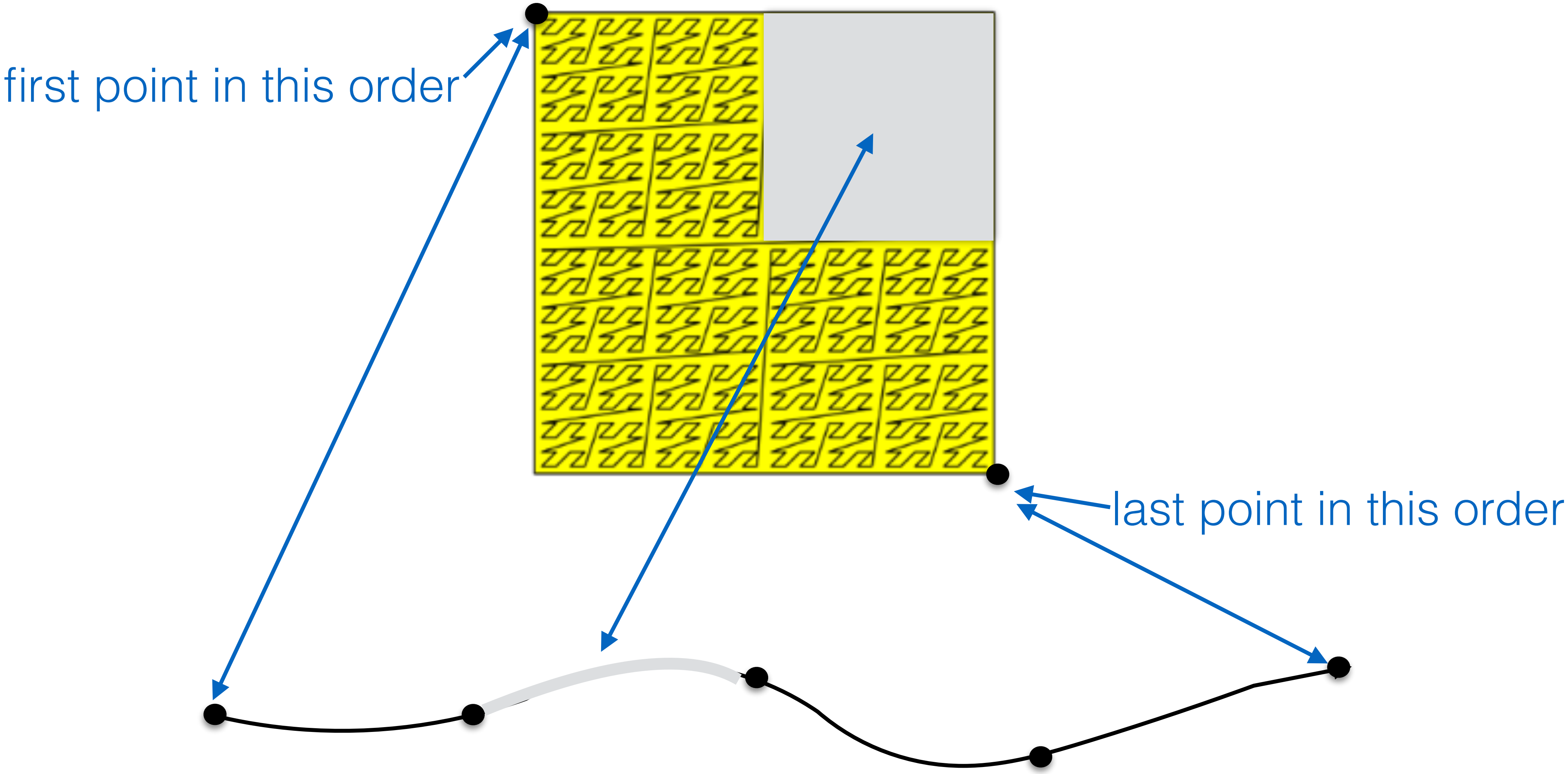
- We visit quadrant 1 before we visit quadrant 2:

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

Z-order



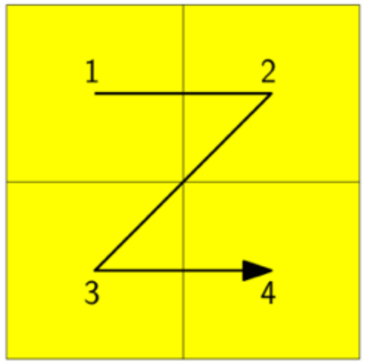
visit quadrants recursively in this order: NW, NE, SW, SE



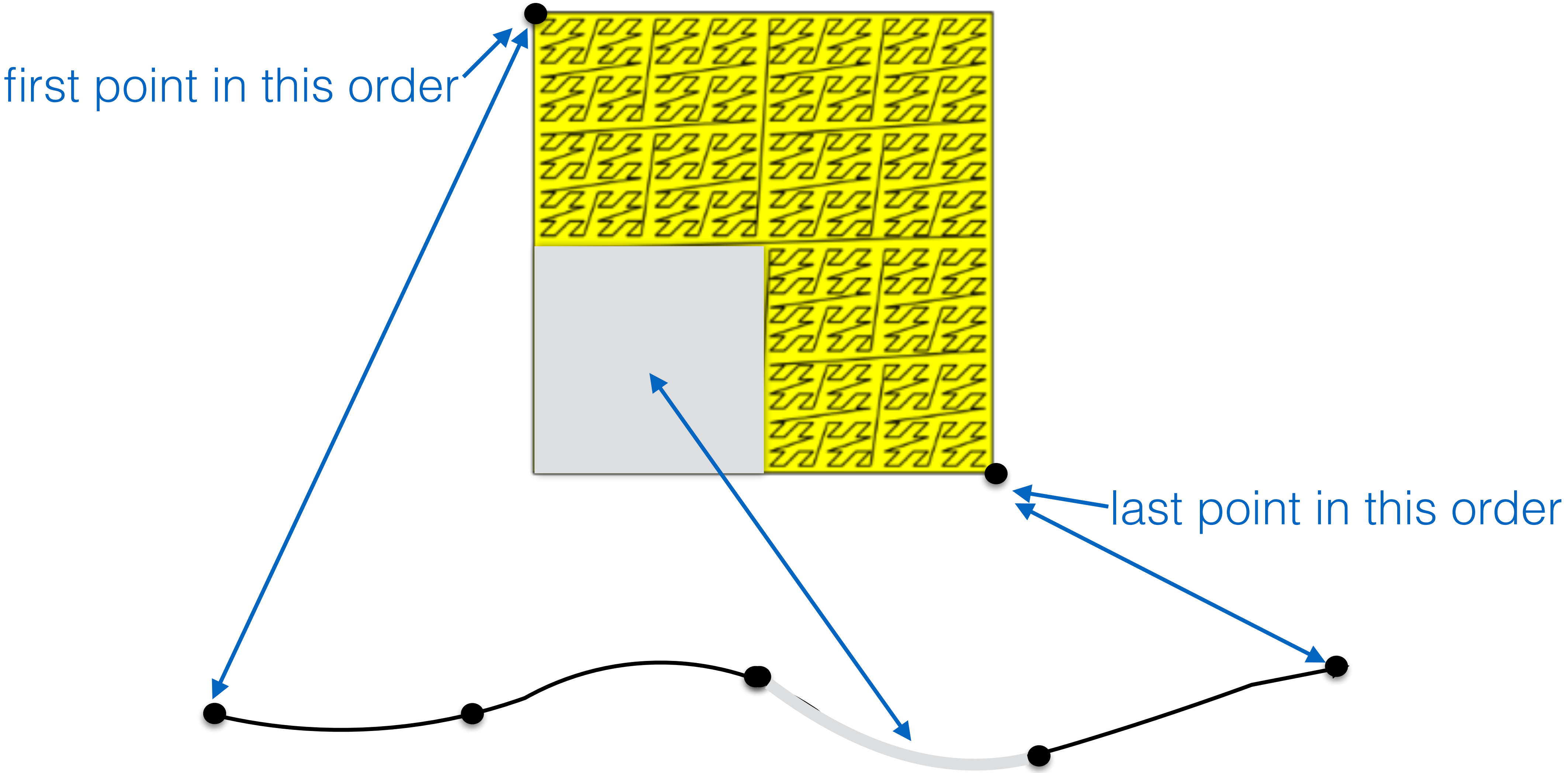
- We visit quadrant 1 before we visit quadrant 2:

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

Z-order

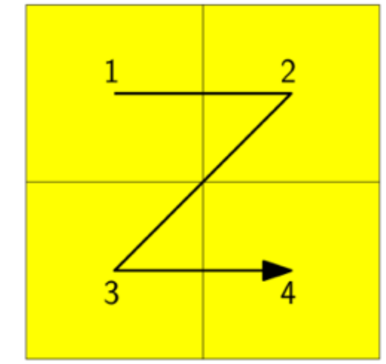


visit quadrants recursively in this order: NW, NE, SW, SE



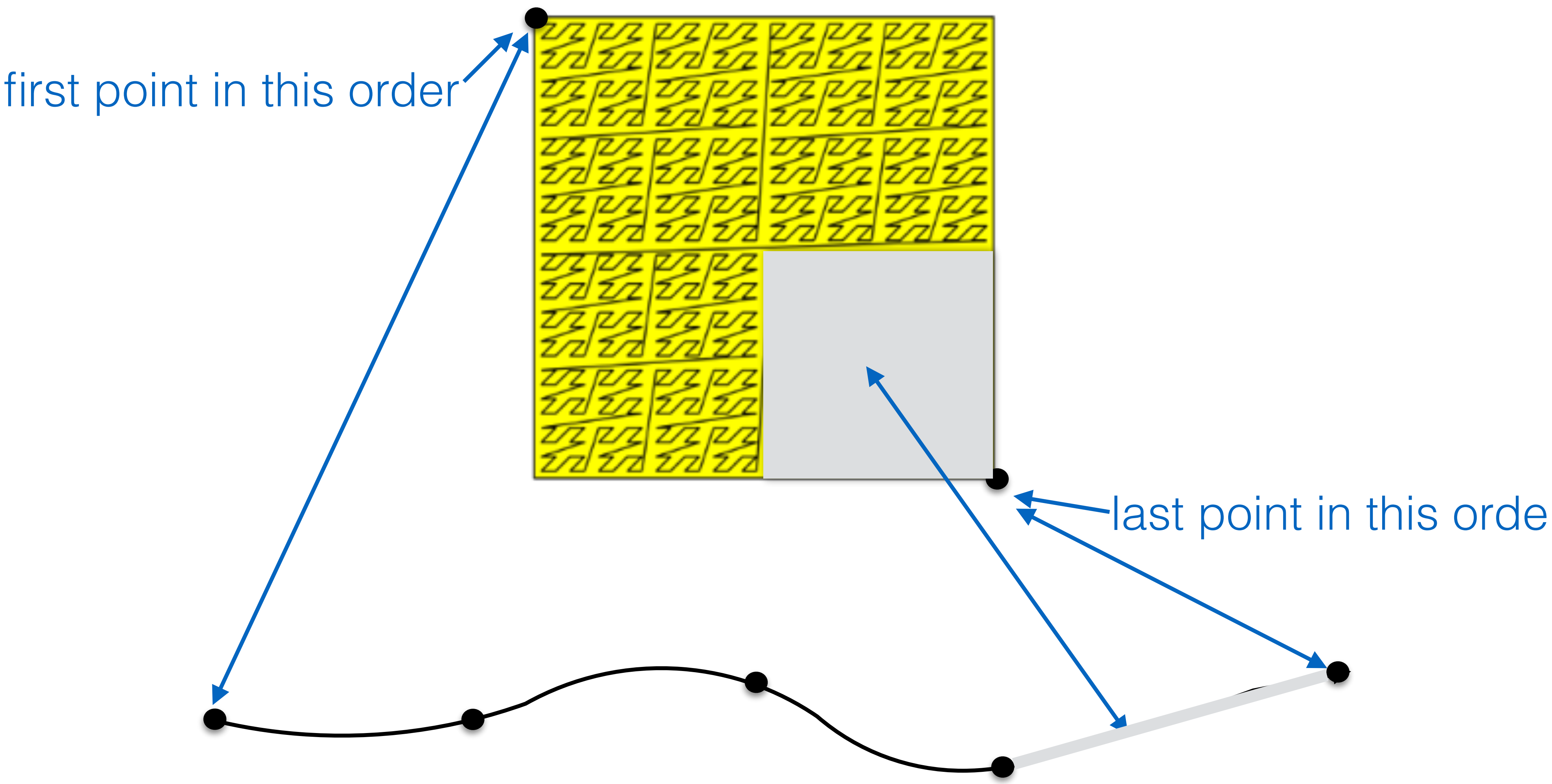
- We visit quadrant 1 before we visit quadrant 2:

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



Z-order

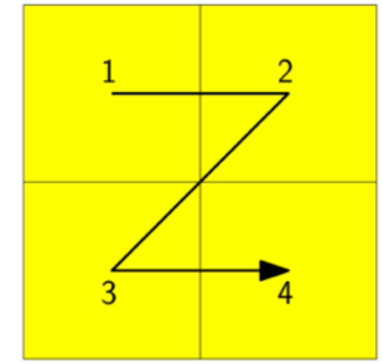
visit quadrants recursively in this order: NW, NE, SW, SE



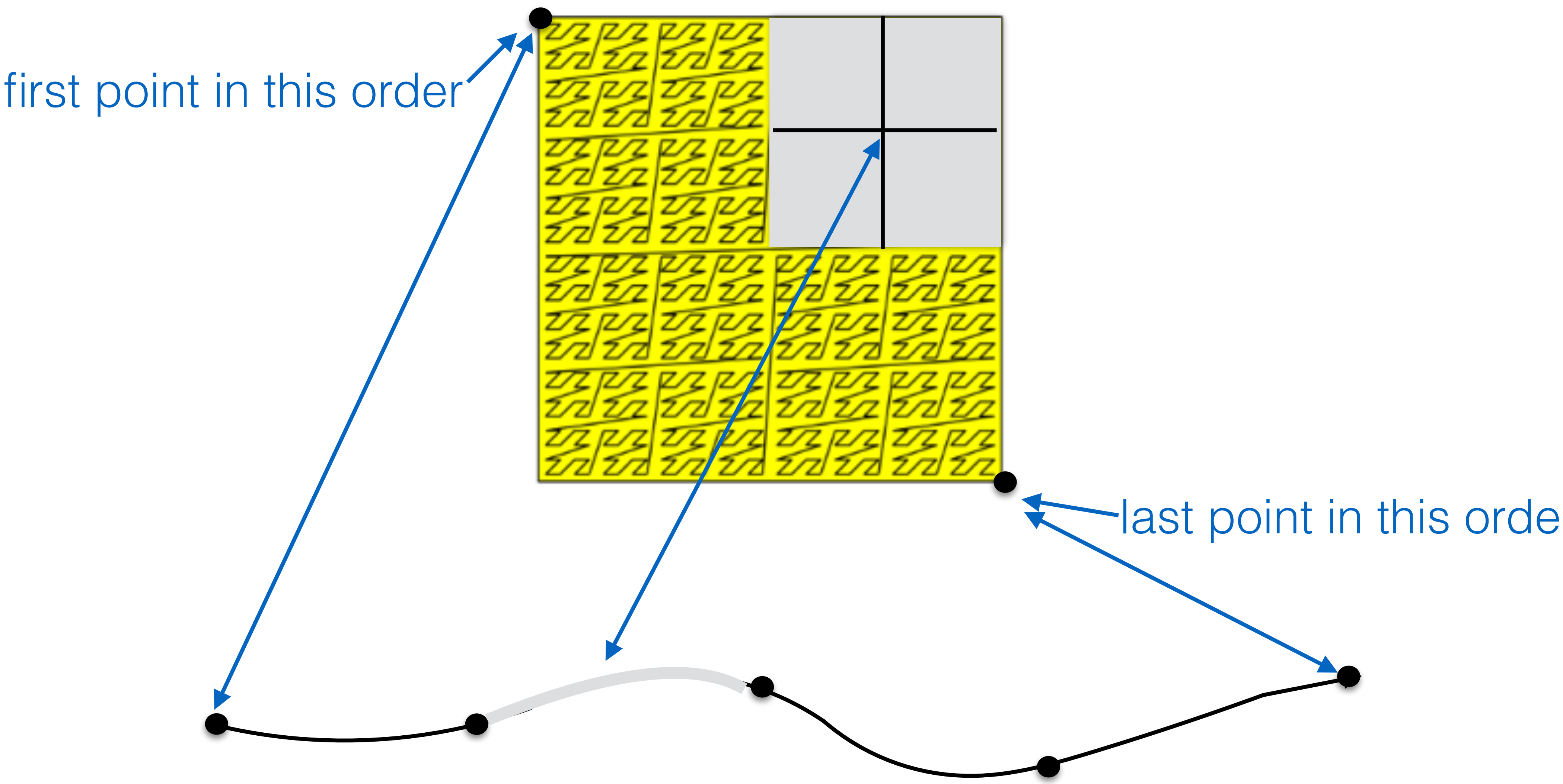
- We visit quadrant 1 before we visit quadrant 2:

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

Z-order

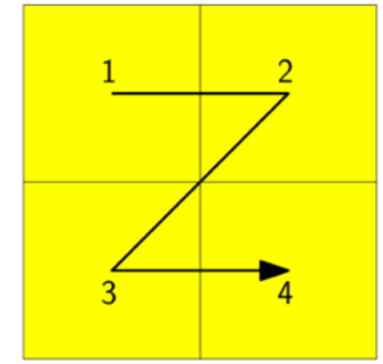


visit quadrants recursively in this order: NW, NE, SW, SE

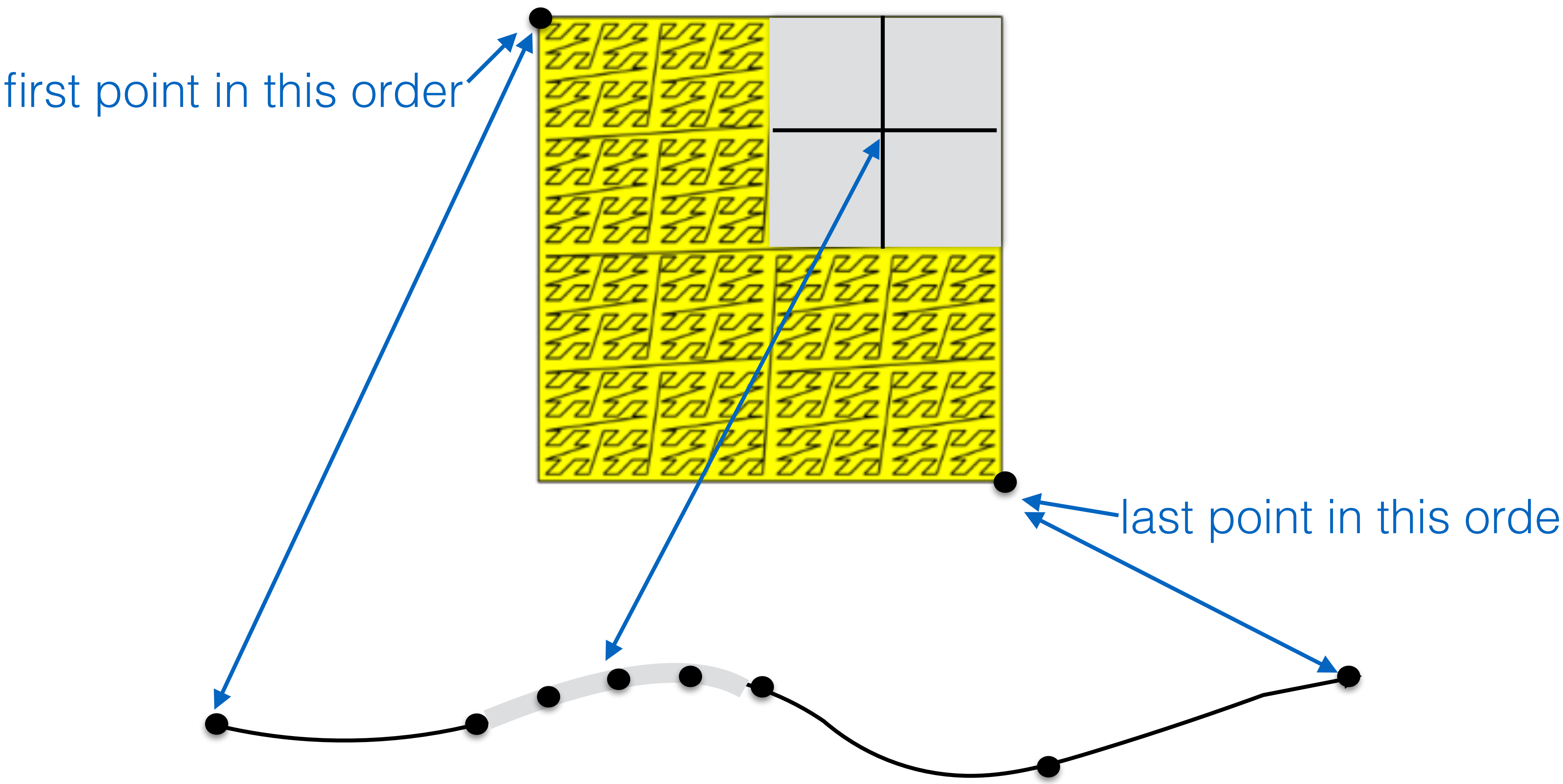


- and so on…..

Z-order

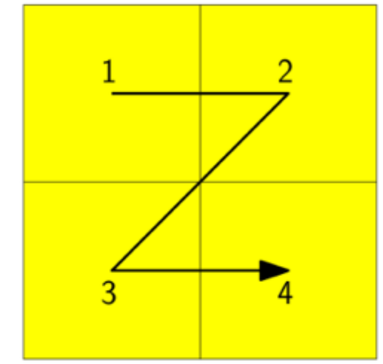


visit quadrants recursively in this order: NW, NE, SW, SE

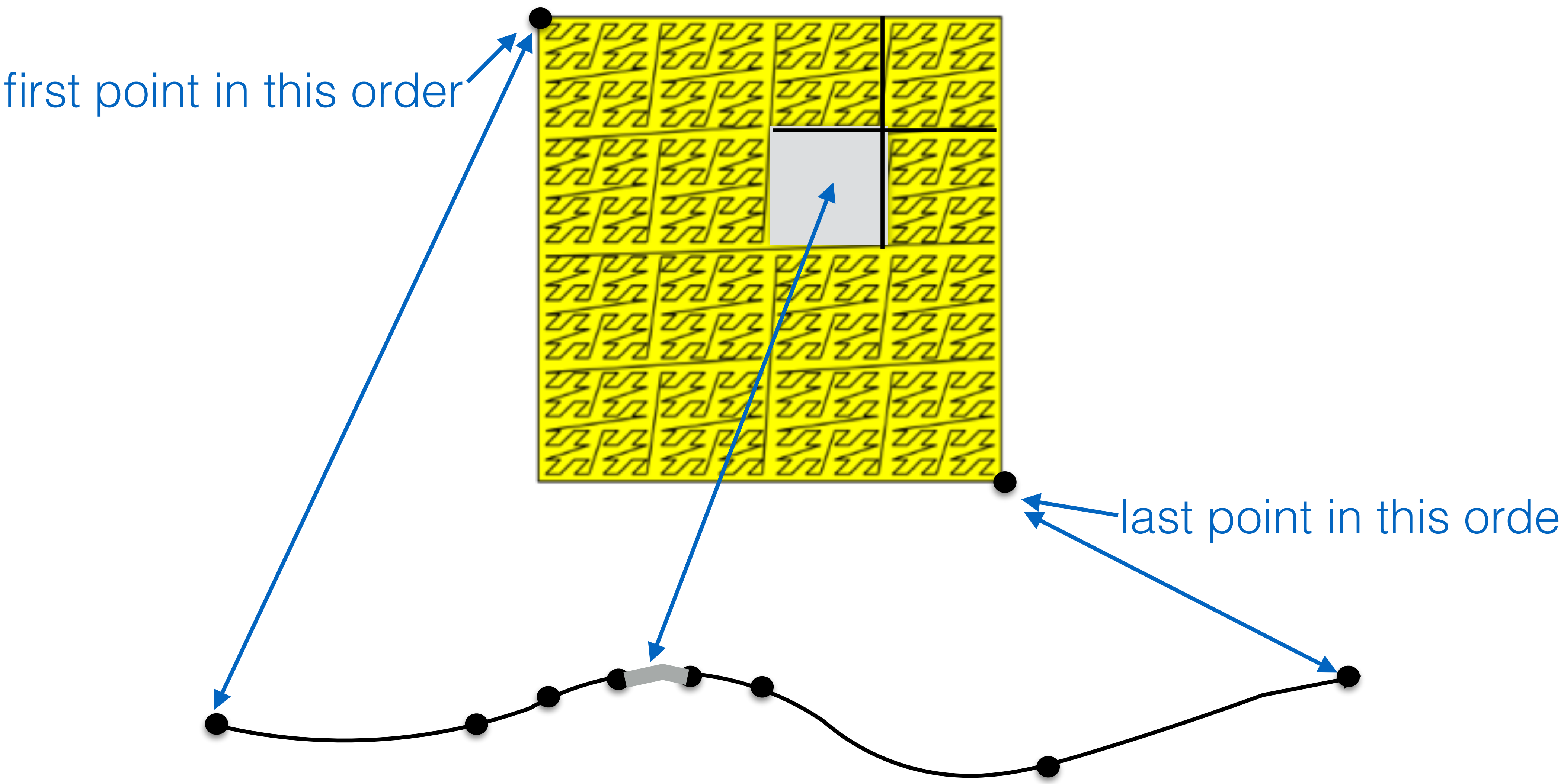


- and so on…..

Z-order

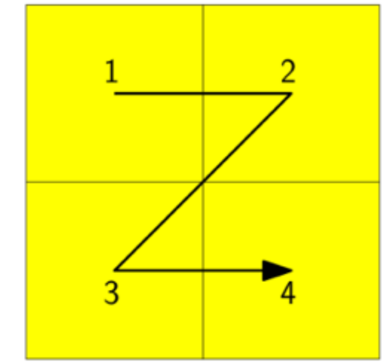


visit quadrants recursively in this order: NW, NE, SW, SE

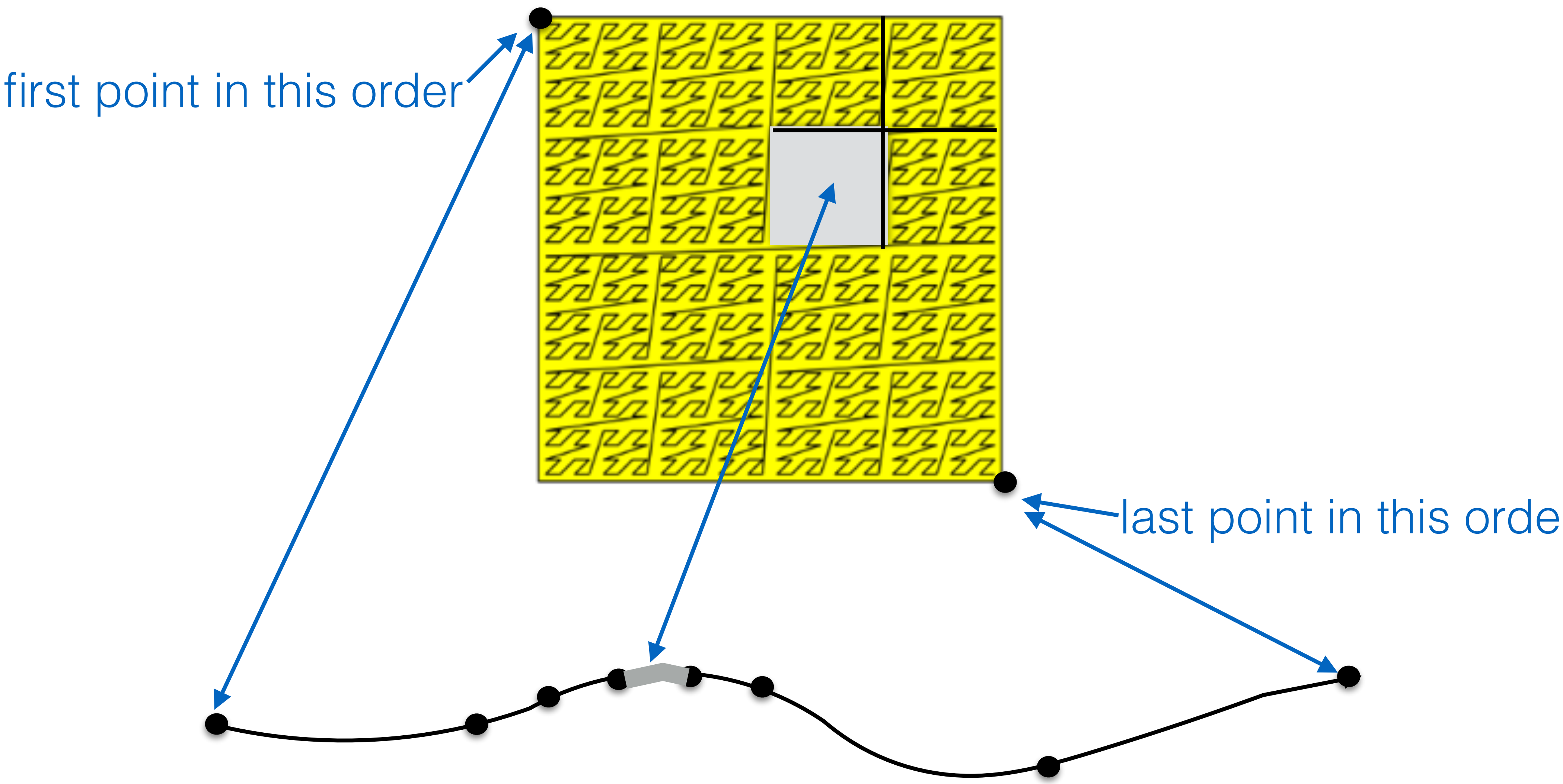


- Every canonical square corresponds to an interval of the z-order curve

Z-order

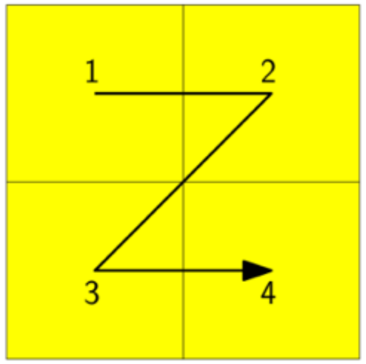


visit quadrants recursively in this order: NW, NE, SW, SE

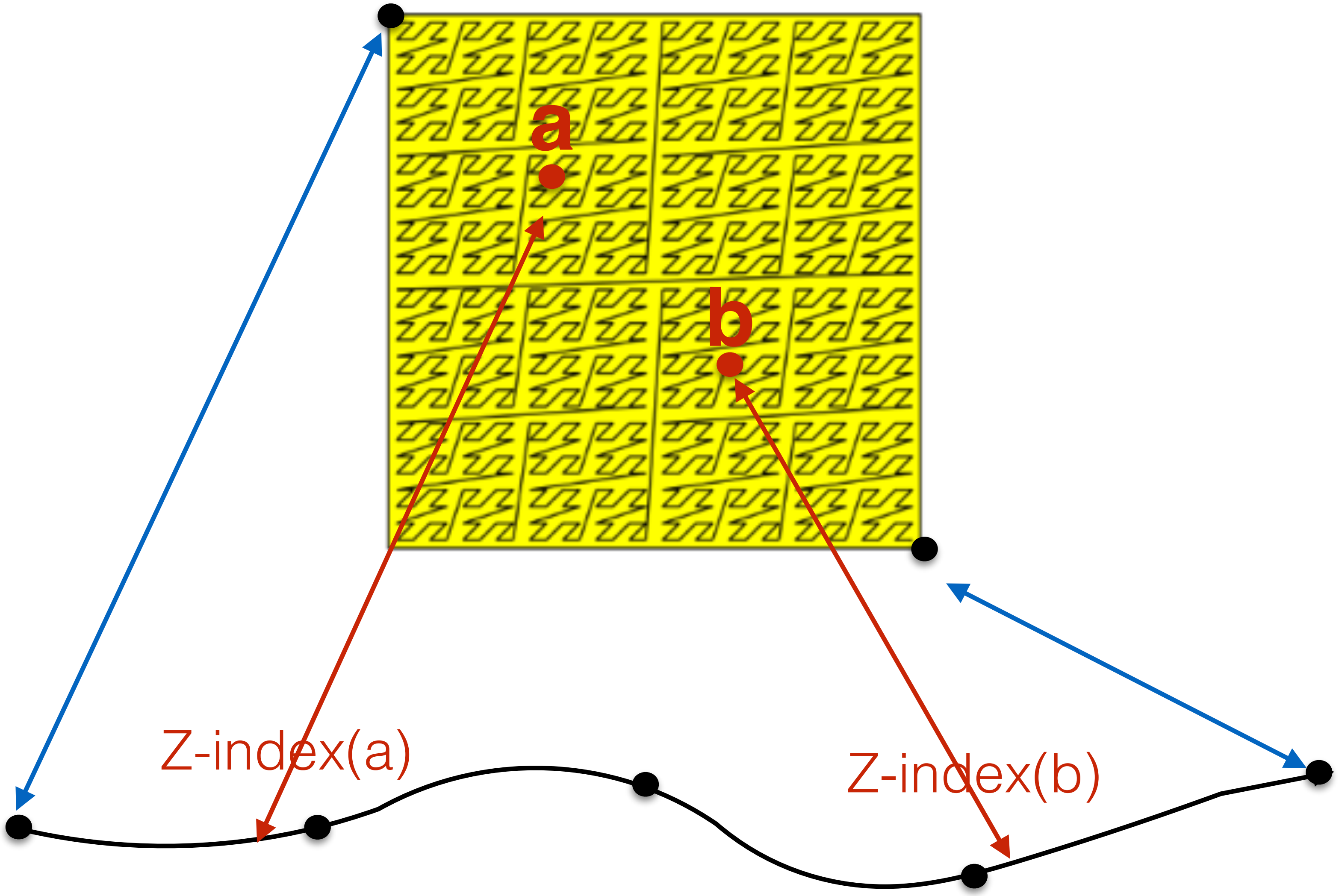


- Two canonical squares are non-intersecting, or one included in the other

Z-order

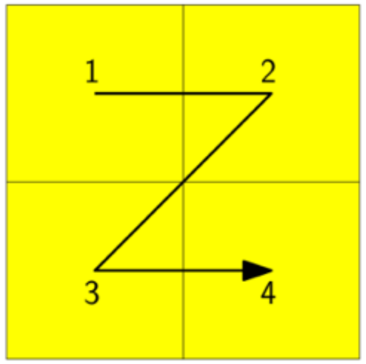


visit quadrants recursively in this order: NW, NE, SW, SE

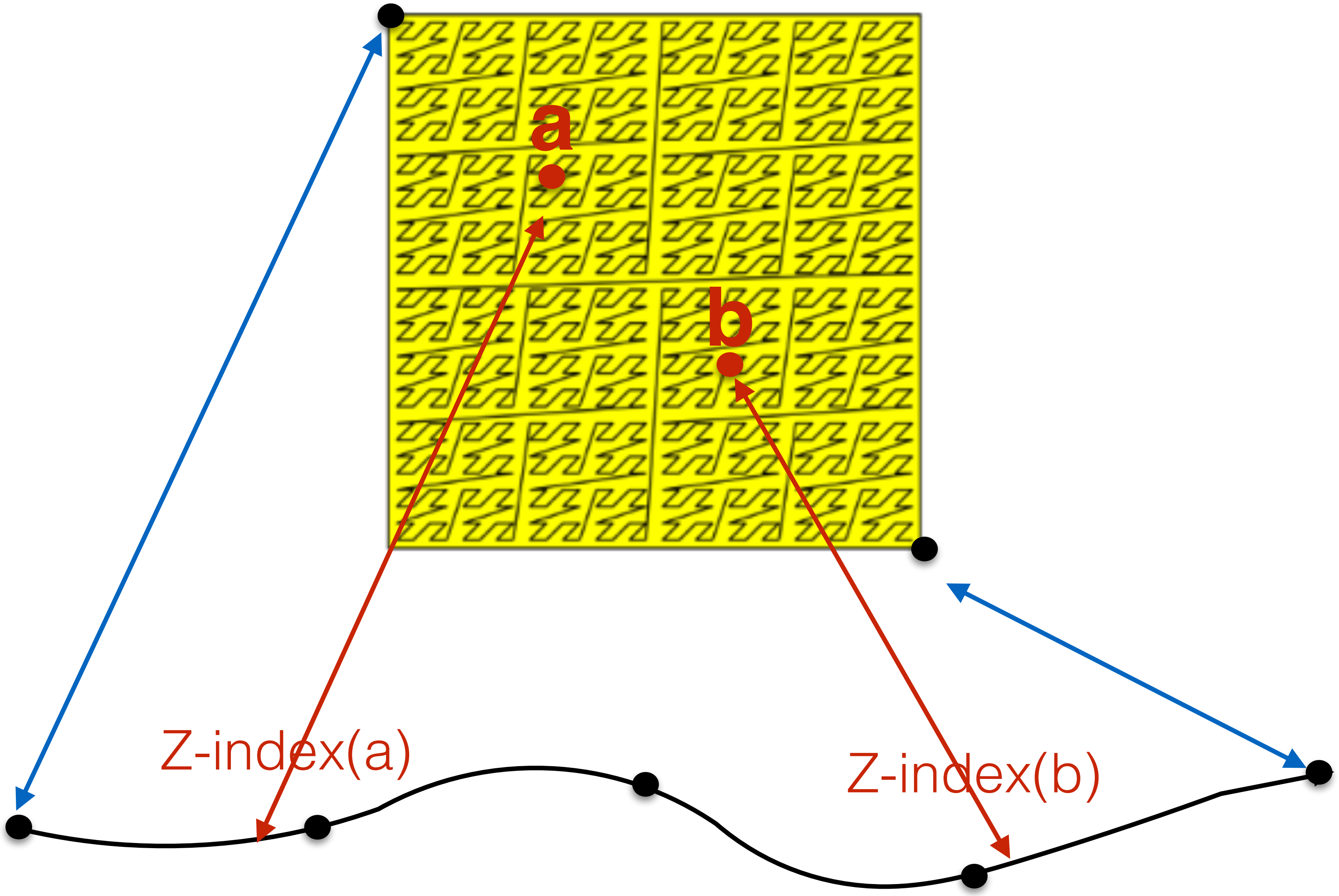


- Any two points can be compared: compare their Z-indices

Z-order



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



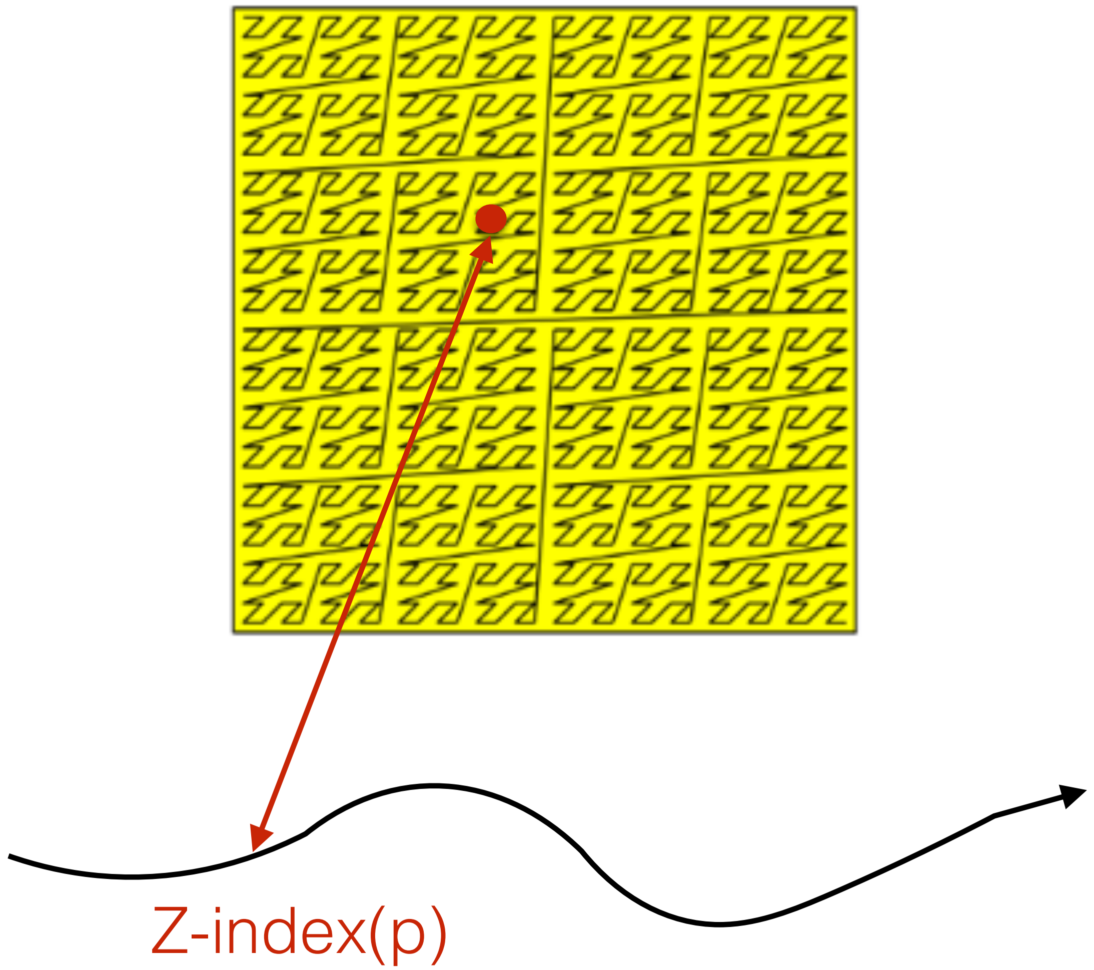
- Any two points can be compared: compare their Z-indices
  - If point a comes before point b on the Z-order curve, it's said that a < b

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?



Computing the Z-index

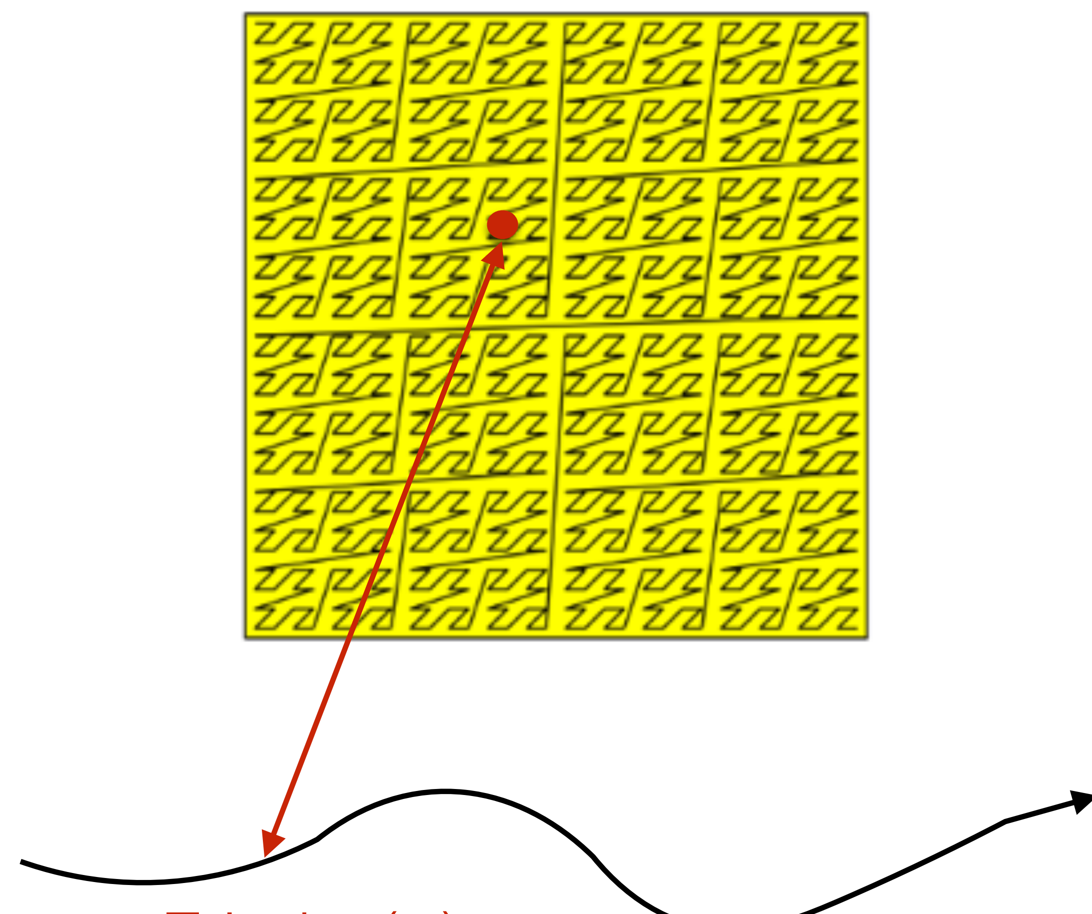
For simplicity assume points with integer coordinates on k bits

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

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

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

What is the largest value representable on 2k bits?



Z-index(p)

Computing the Z-index

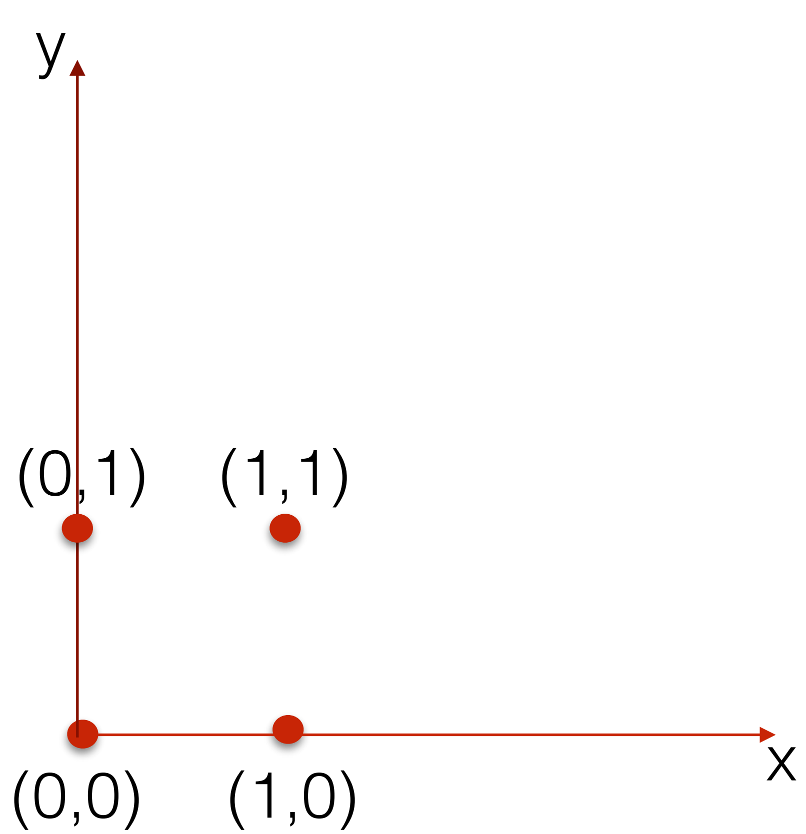
For simplicity assume points with integer coordinates on k bits

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

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

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

k=1 bit



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

Computing the Z-index

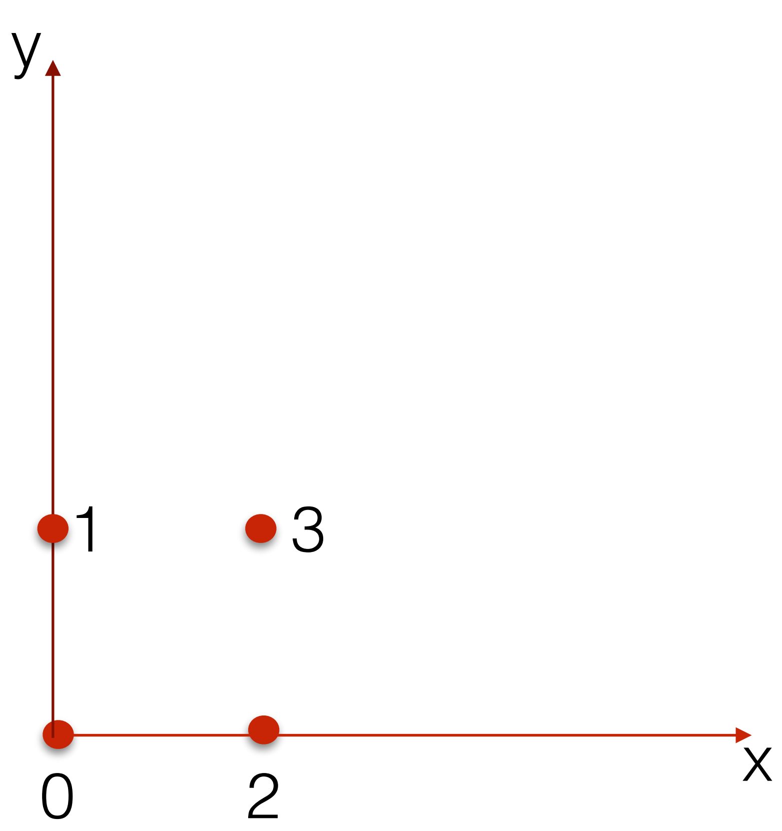
For simplicity assume points with integer coordinates on k bits

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

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

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

k=1 bit



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

Computing the Z-index

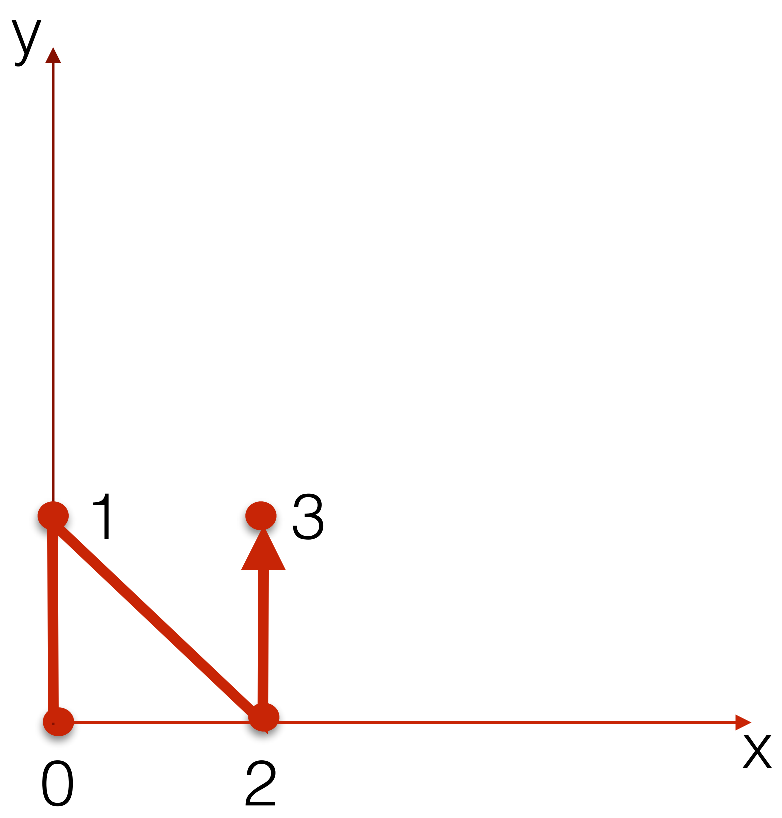
For simplicity assume points with integer coordinates on k bits

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

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

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

k=1 bit



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

Computing the Z-index

For simplicity assume points with integer coordinates on k bits

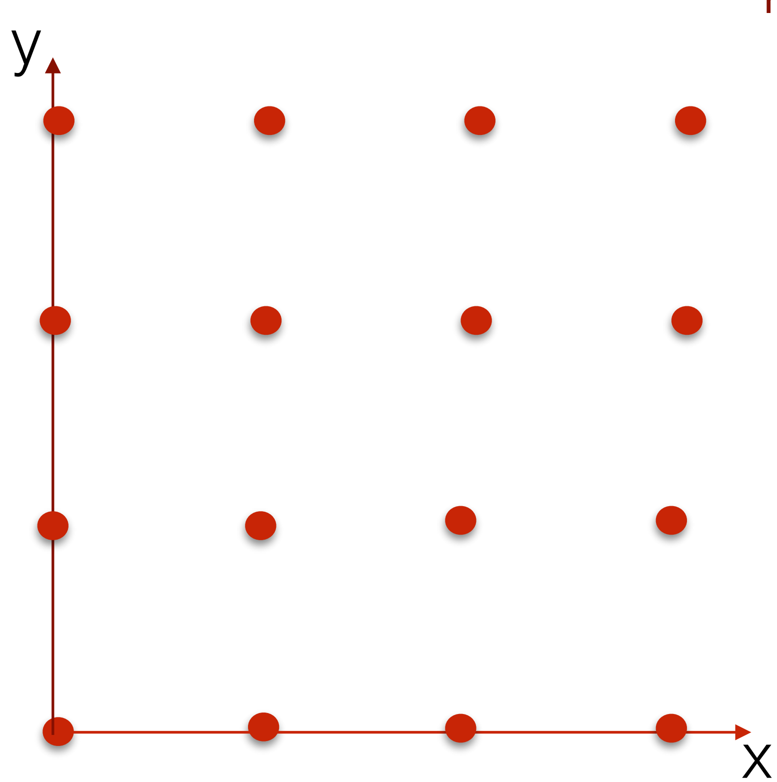
$p = (x_1x_2x_3\ldots x_k,\ y_1y_2y_3\ldots y_k)$

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

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

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



Find the Z-order!

Computing the Z-index

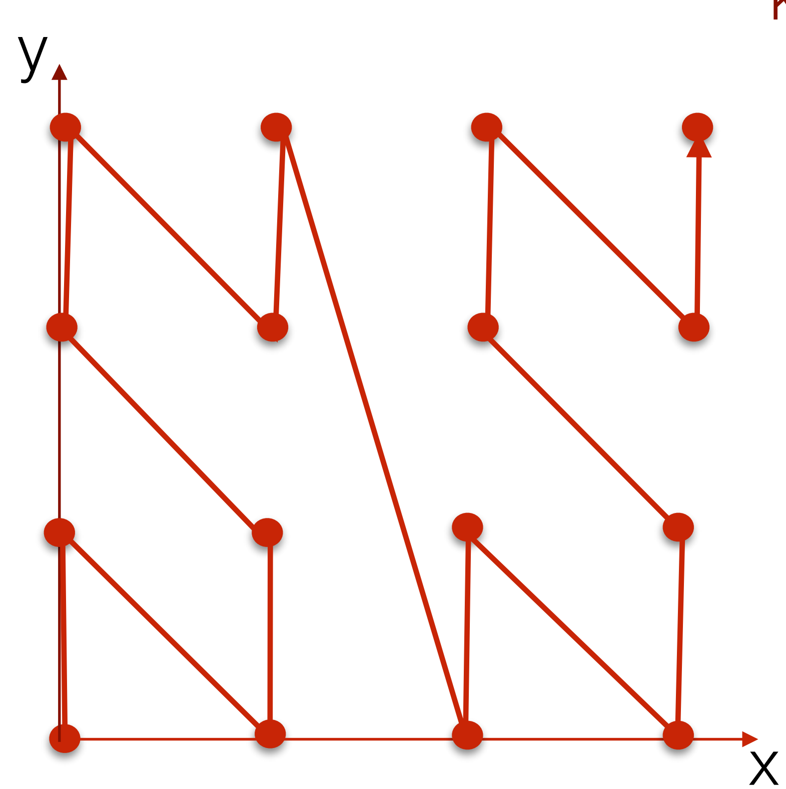
For simplicity assume points with integer coordinates on k bits

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

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

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

k=2 bits



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)	

Computing the Z-index

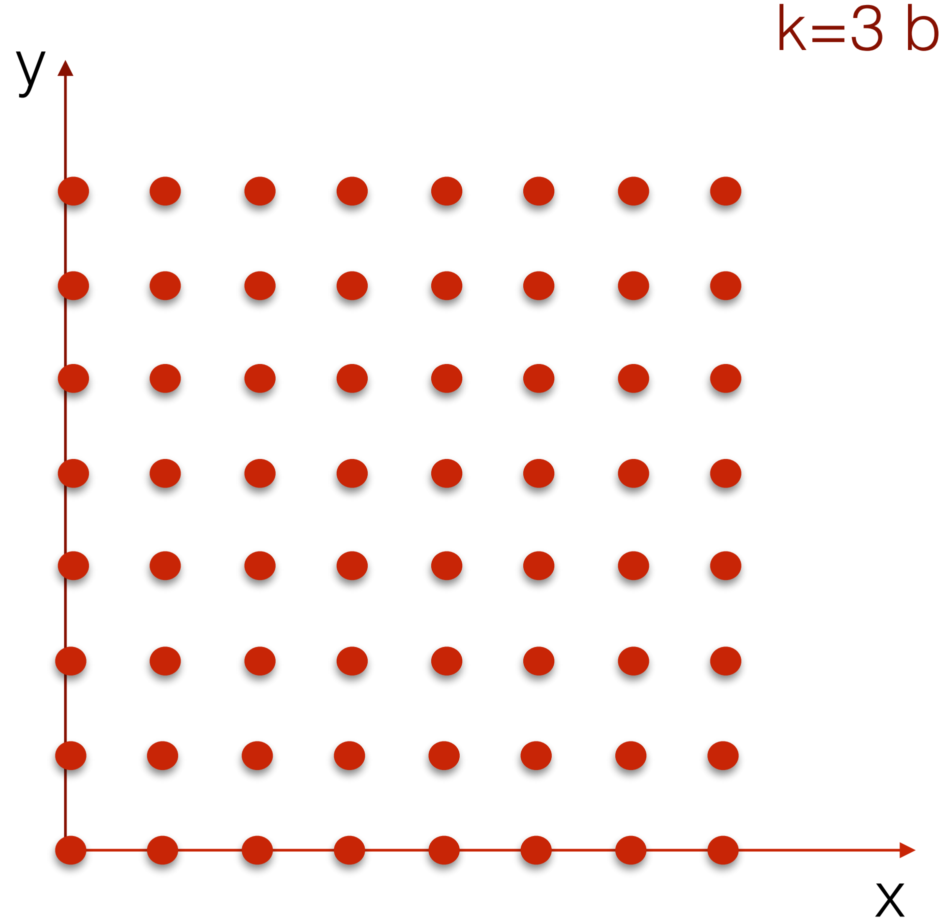
For simplicity assume points with integer coordinates on k bits

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

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

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

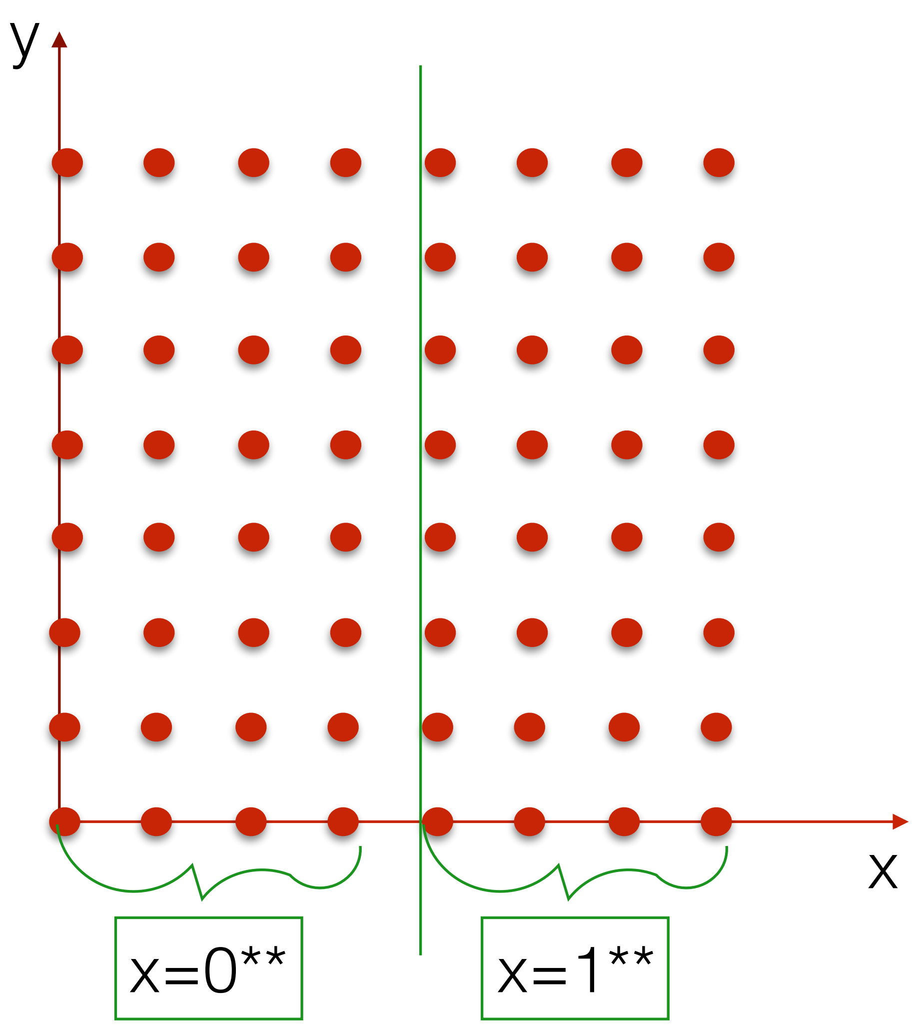
k=3 bits



Find the Z-order!

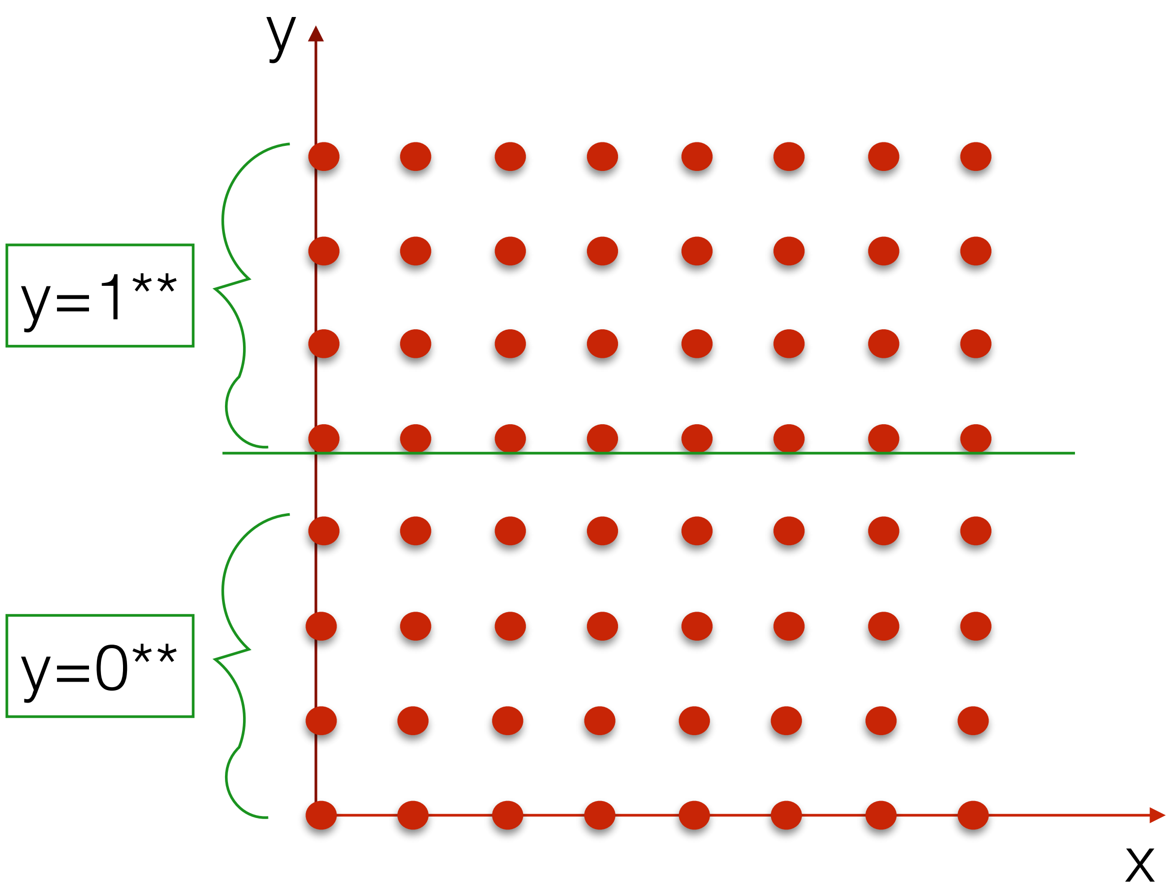
Computing the Z-index

- 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



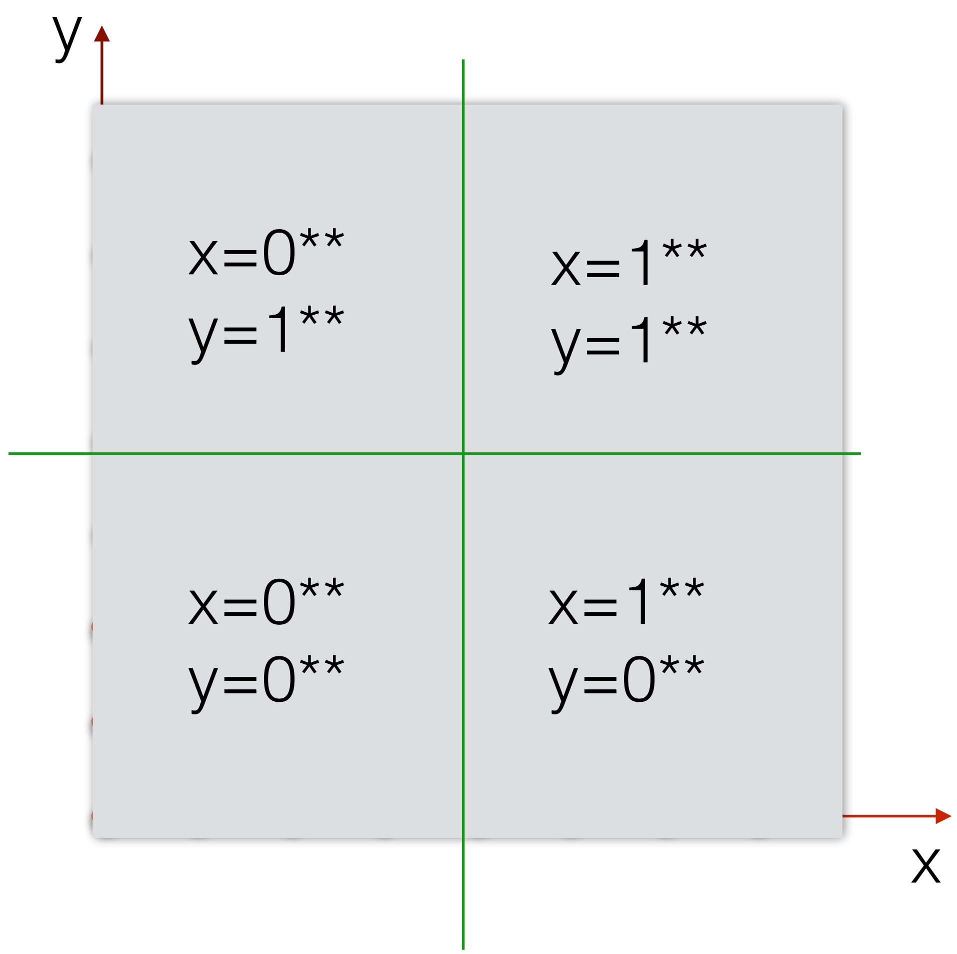
Computing the Z-index

- 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



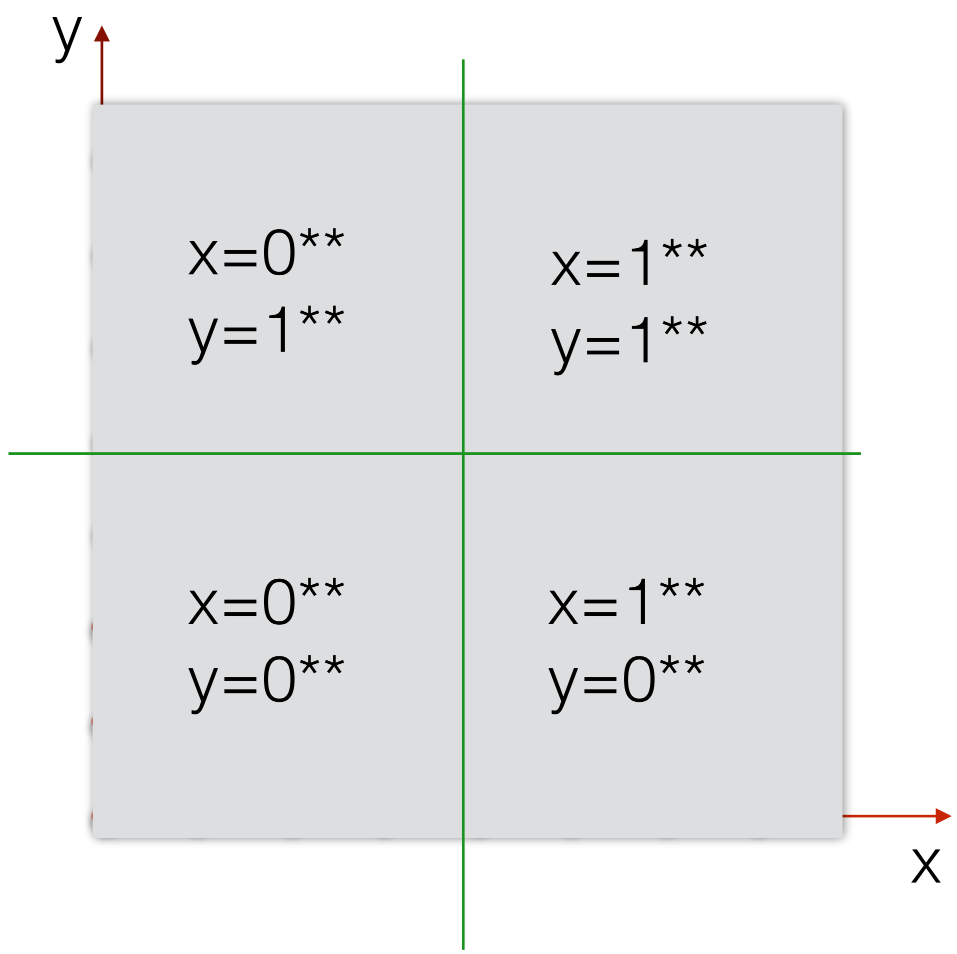
Computing the Z-index

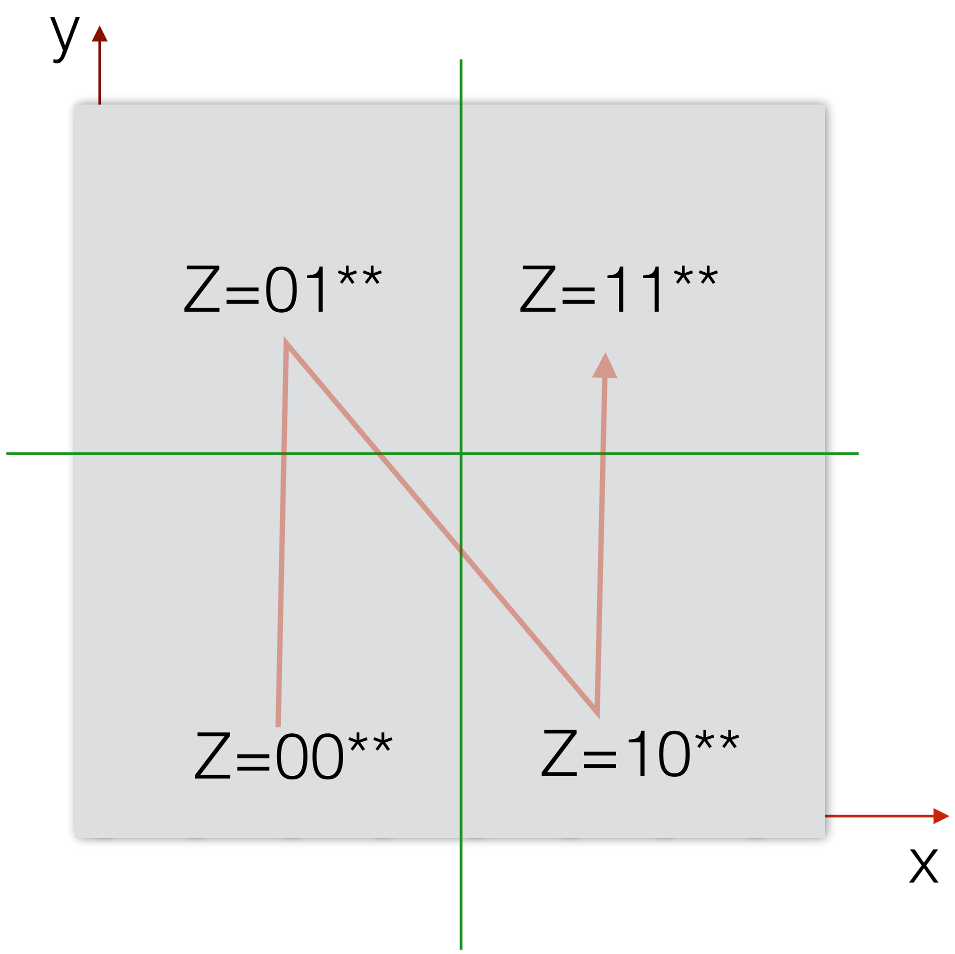
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half



Computing the Z-index

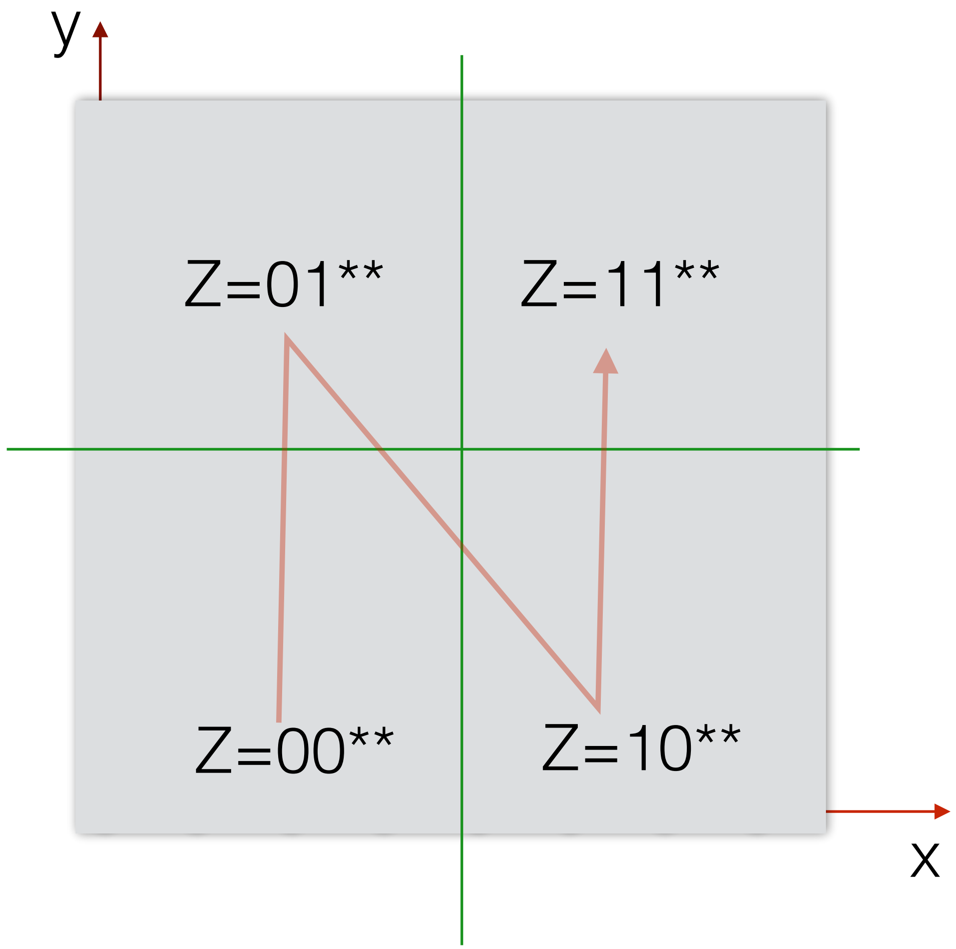
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

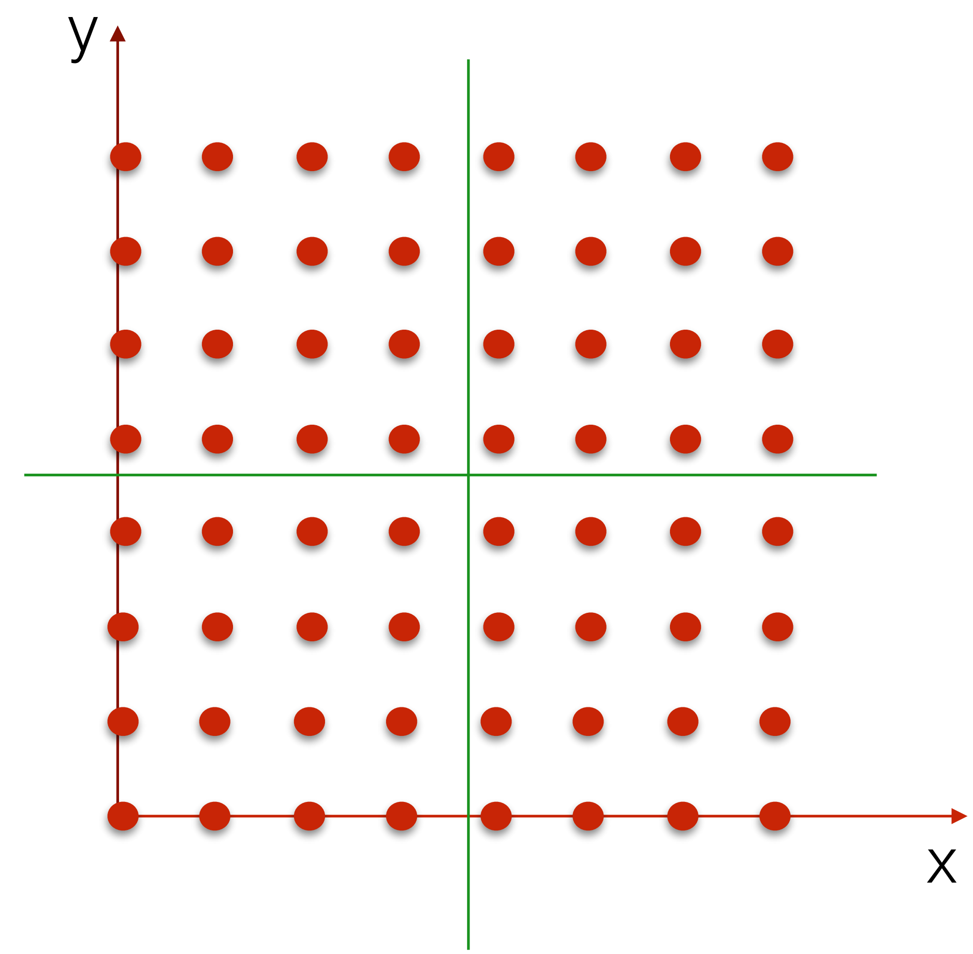




Computing the Z-index

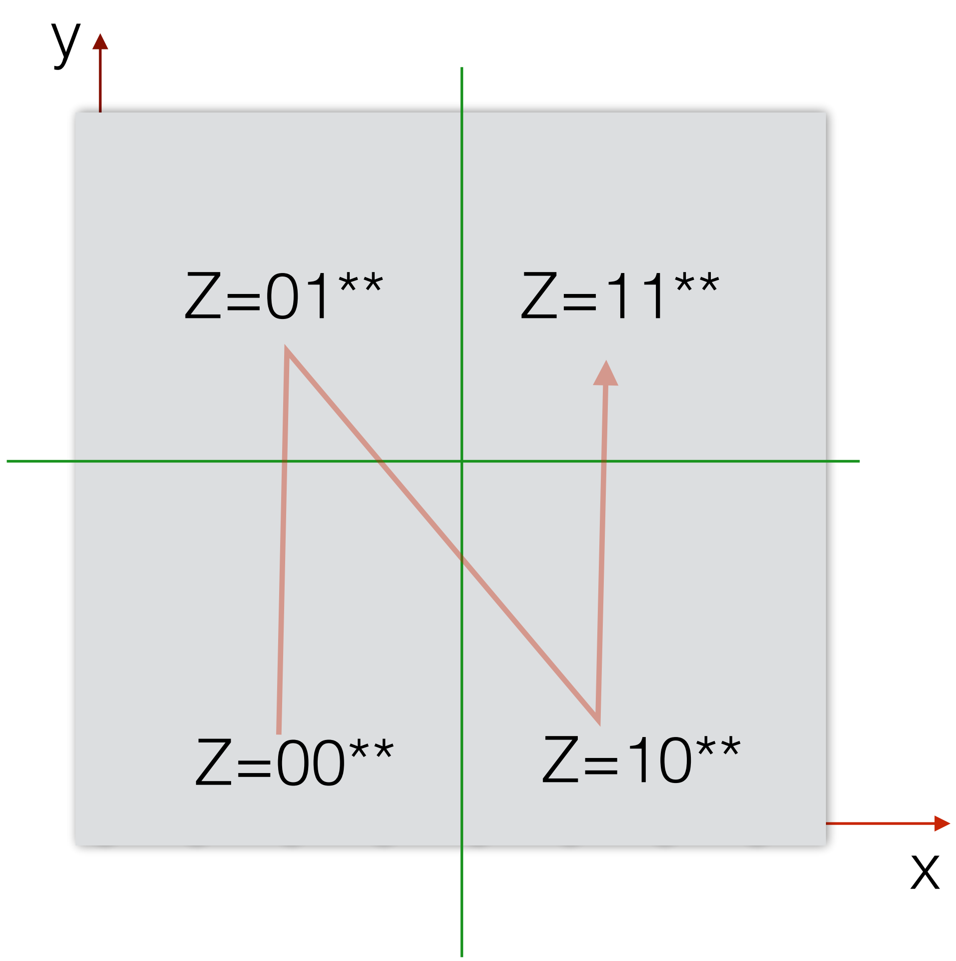
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

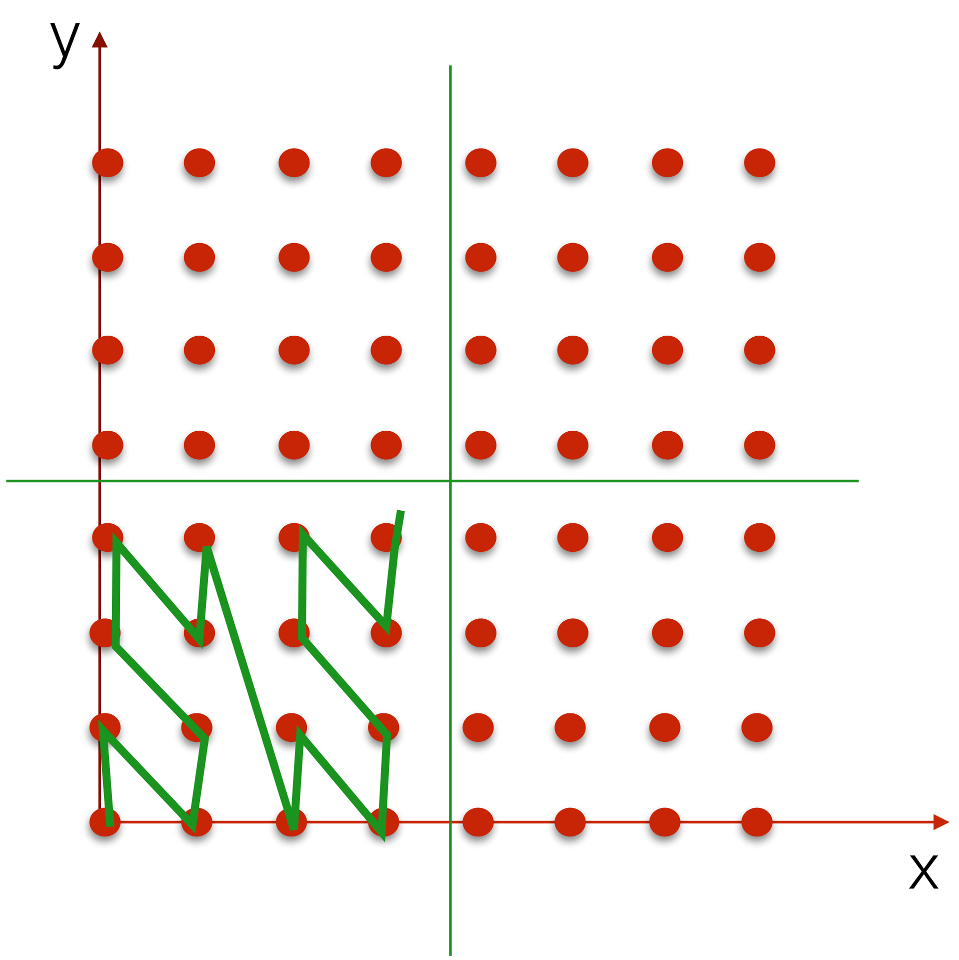




Computing the Z-index

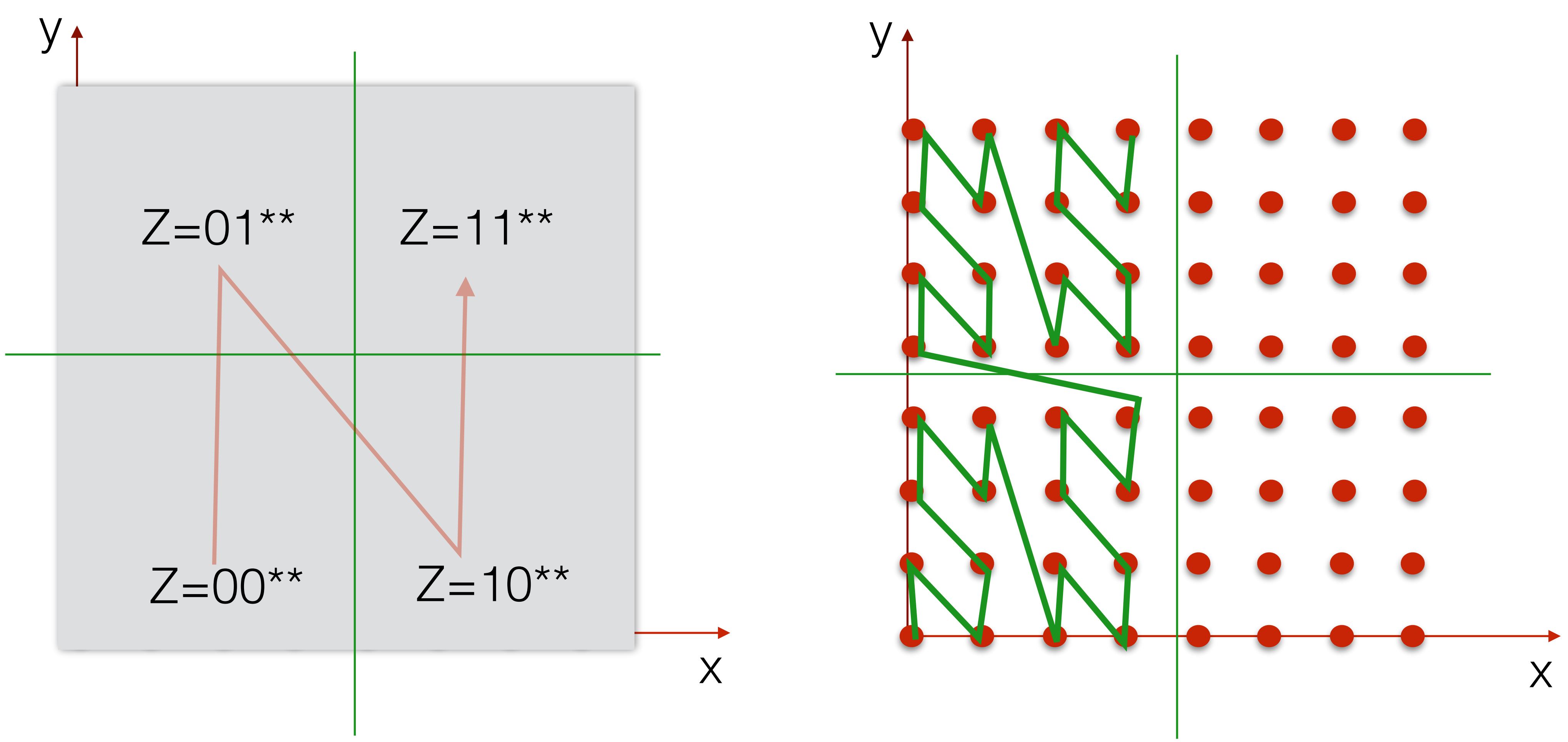
- Consider an y-coordinate $y_1 y_2 y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half





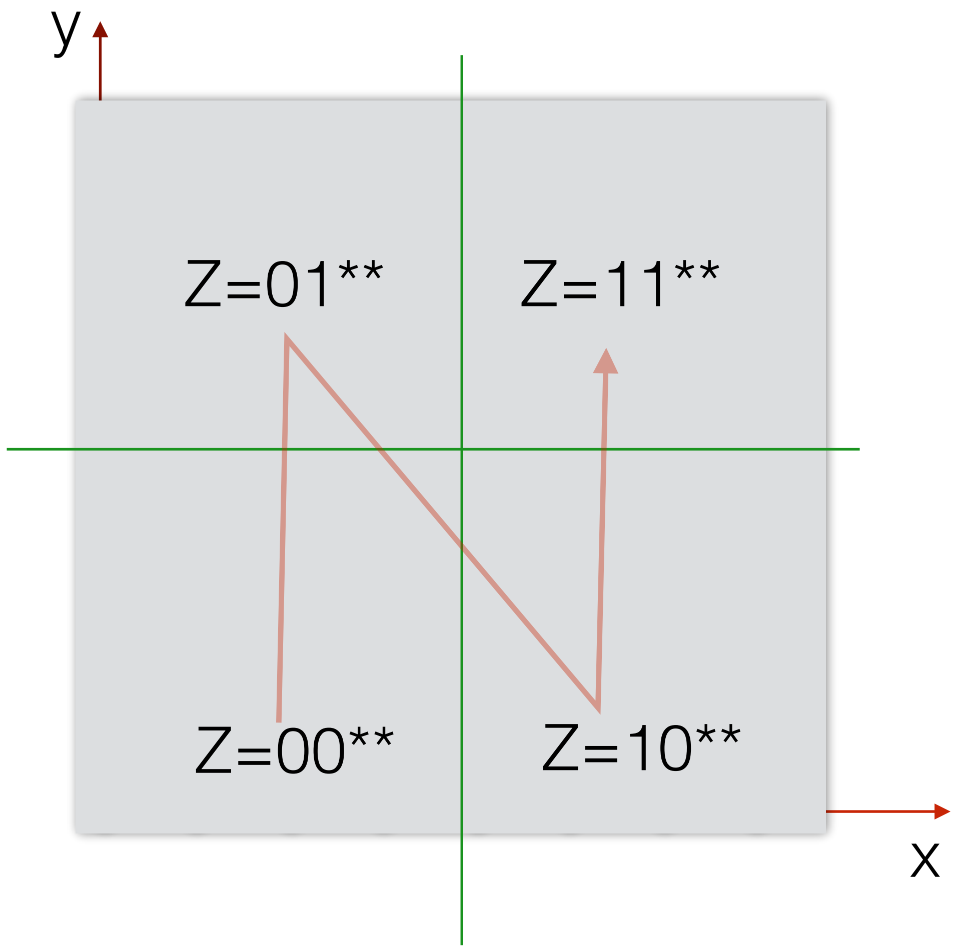
Computing the Z-index

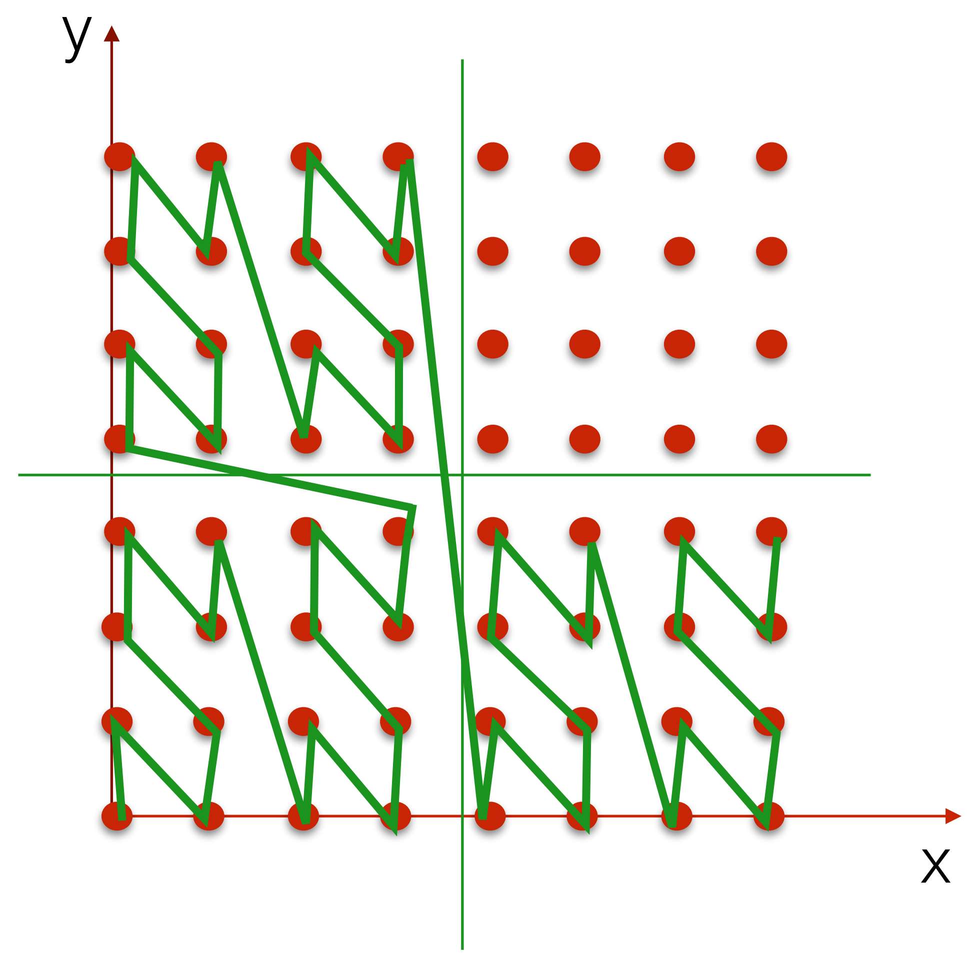
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - $y_1=0$ means the point will reside in the first half
  - $y_1=1$ means the point will reside in the second half



Computing the Z-index

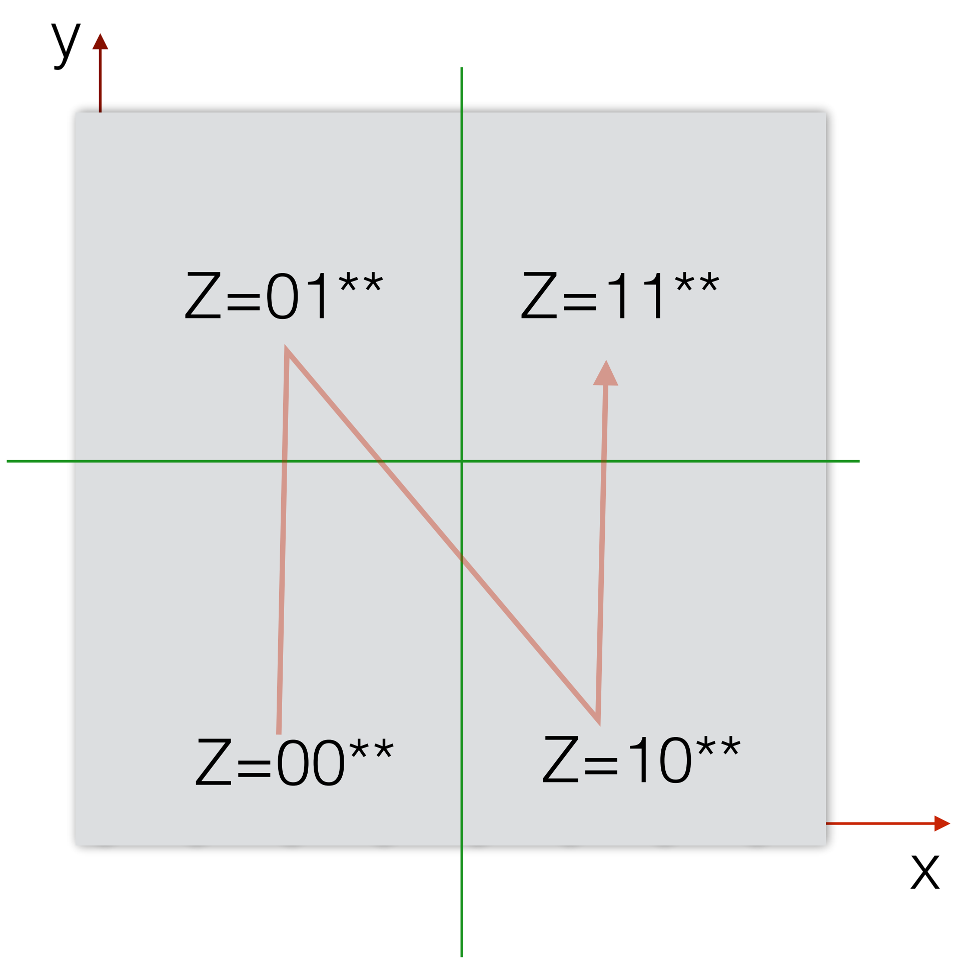
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half

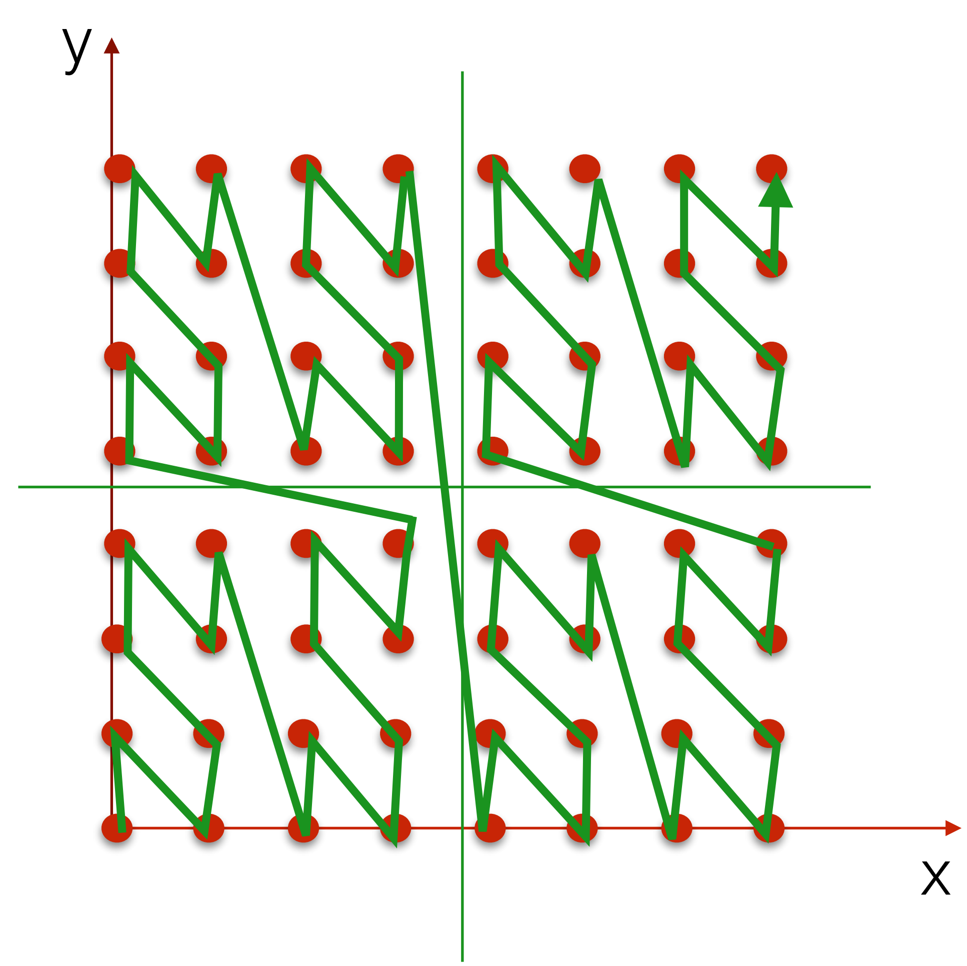




Computing the Z-index

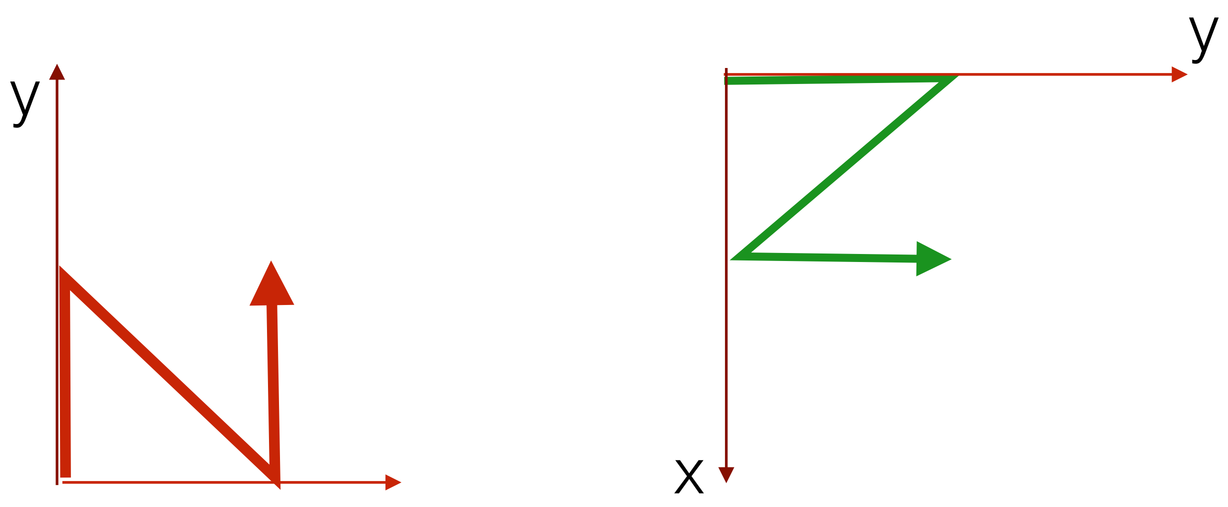
- Consider an y-coordinate $y_1y_2y_3$ in the square [0,…8)
  - y_1=0 means the point will reside in the first half
  - y_1=1 means the point will reside in the second half





Z-order

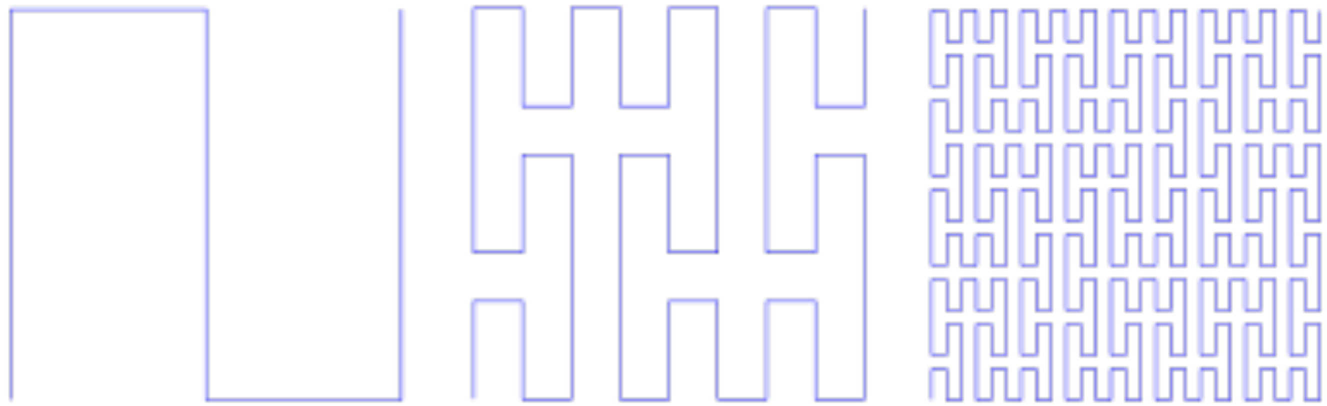
- other Z-orders can be obtained similarly



- 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)

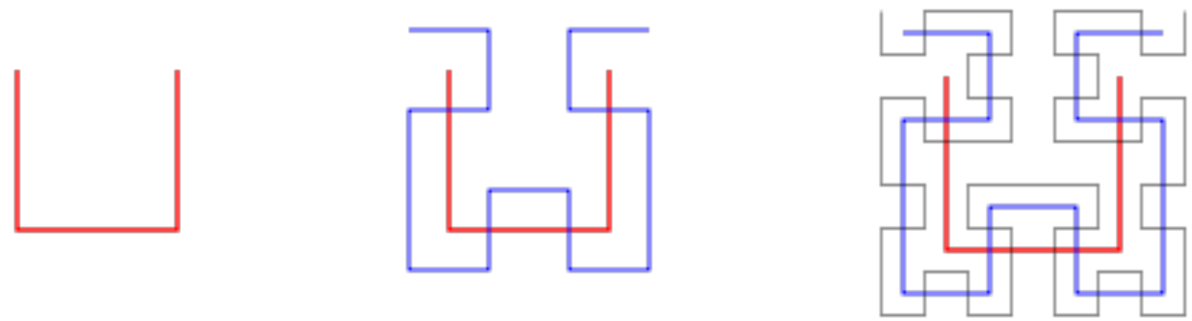
Space-filling curves

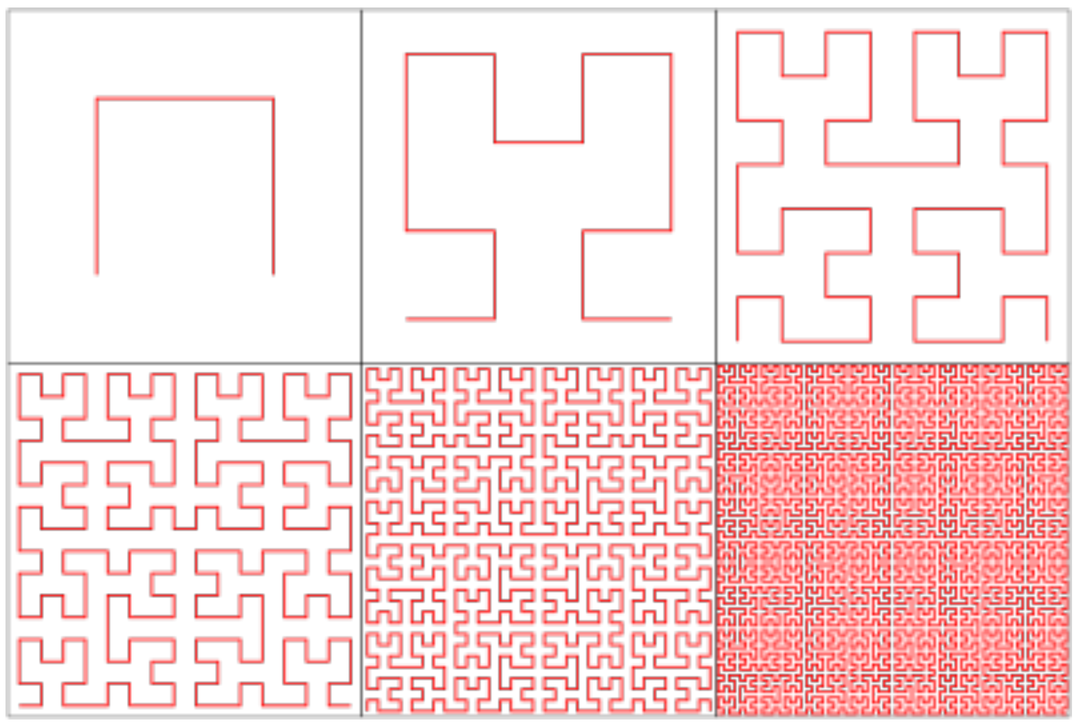
- Z-order curves are a special type of space-filling curves
- First SFC were described by Peano and Hilbert

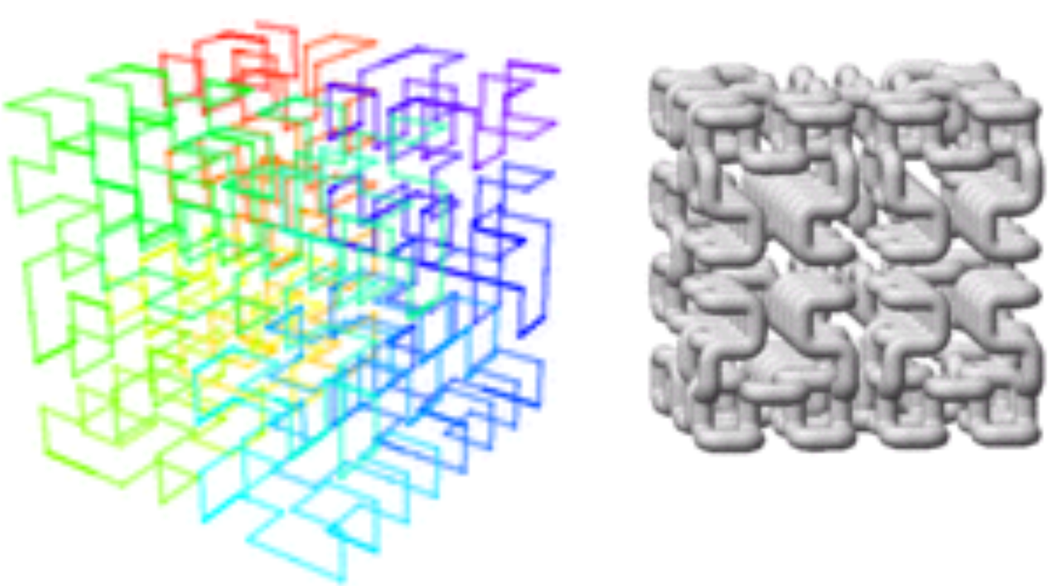


Peano curve

Hilbert curve

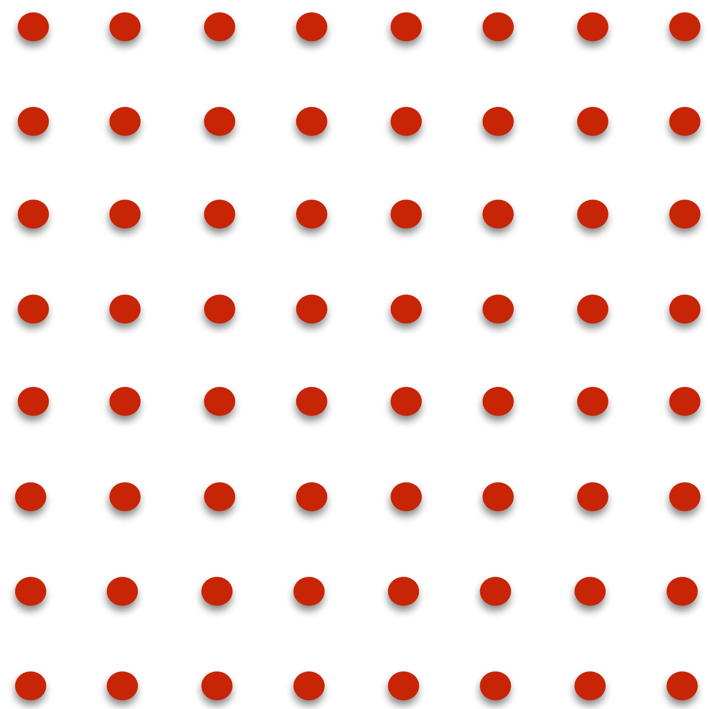






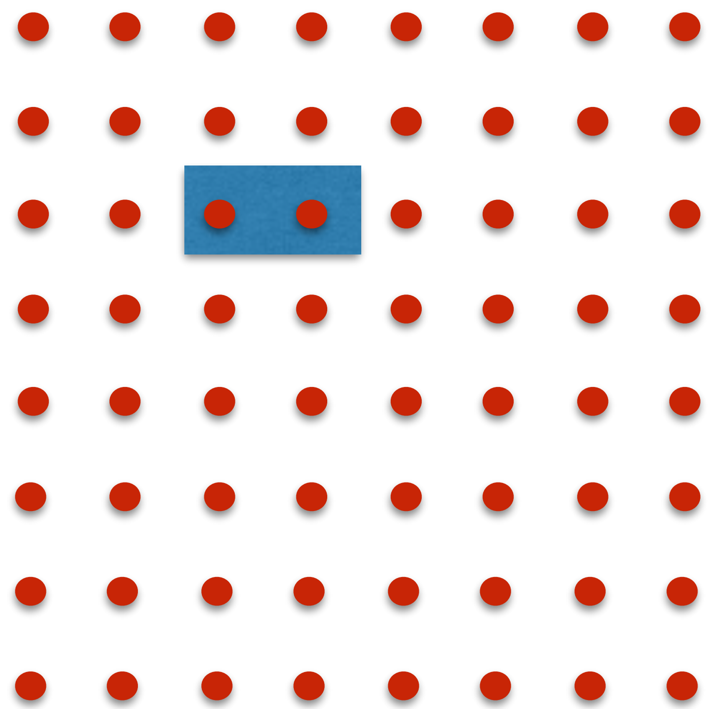
Spatial locality

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



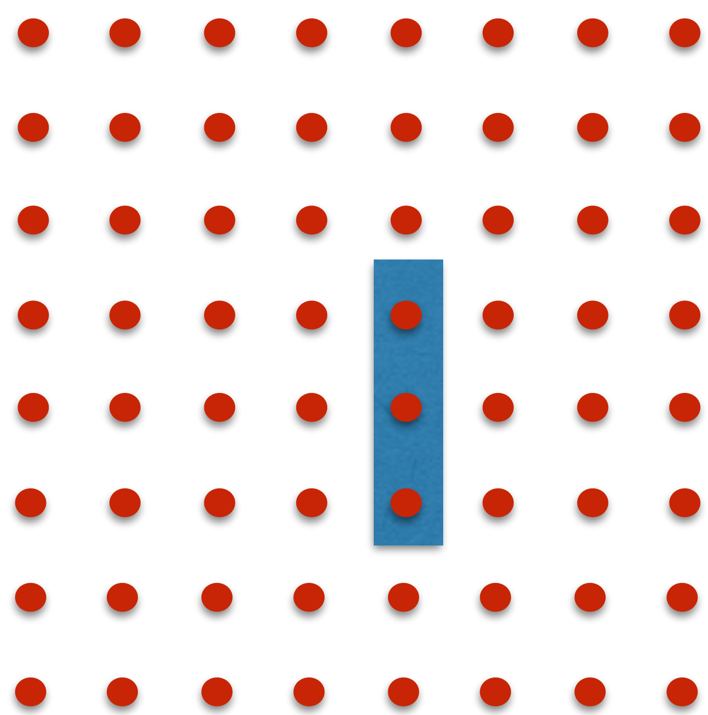
Spatial locality

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



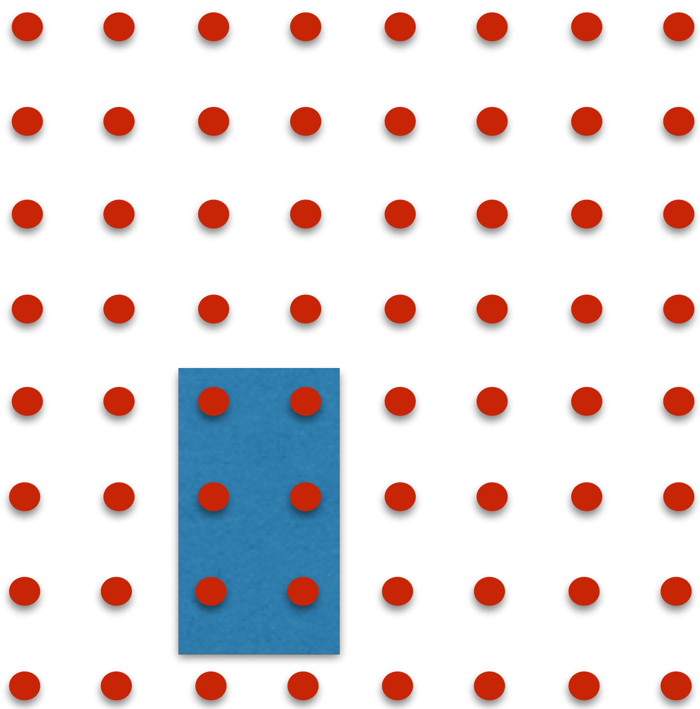
Spatial locality

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



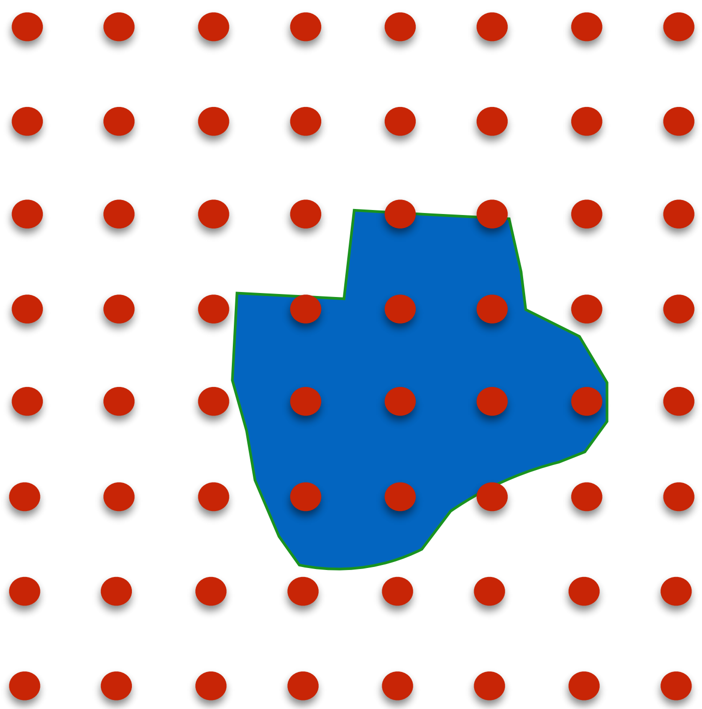
Spatial locality

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



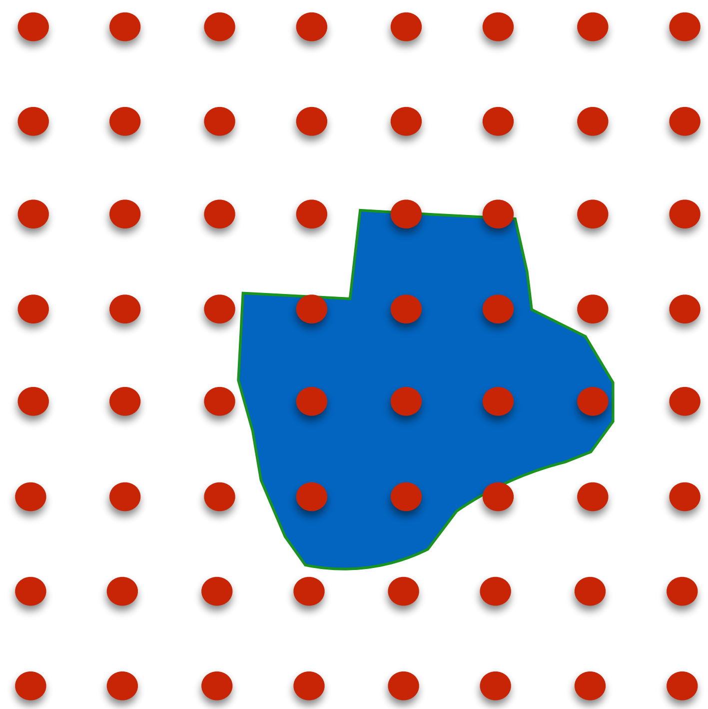
Spatial locality

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



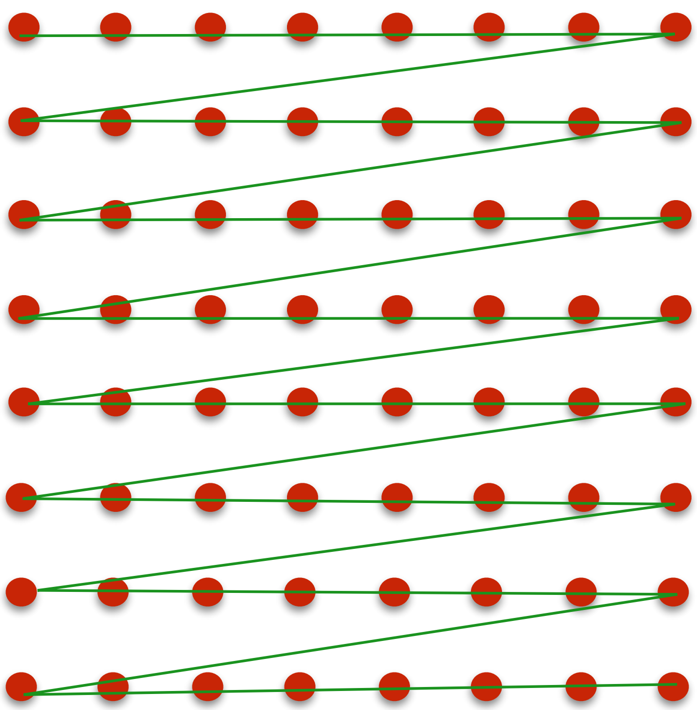
Spatial locality

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



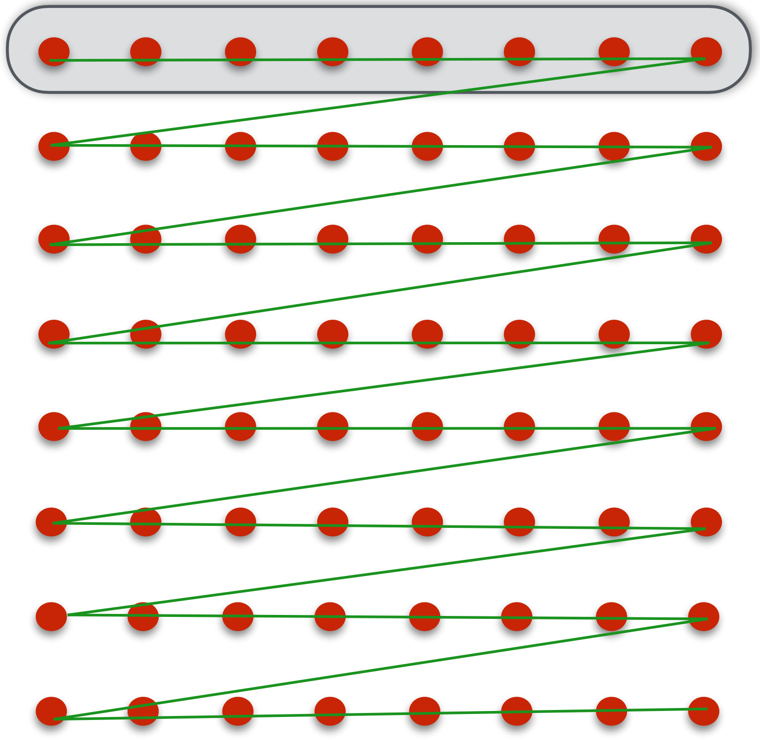
We would like points “close” in 2D to be stored “close” to each other in the data structure

Spatial locality

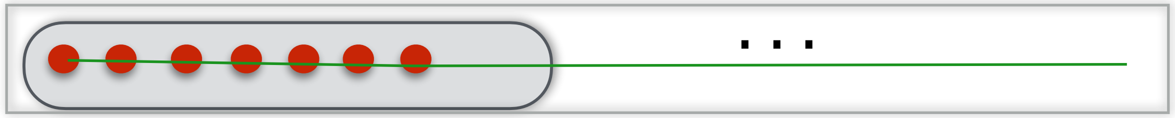


grid in default row-major order

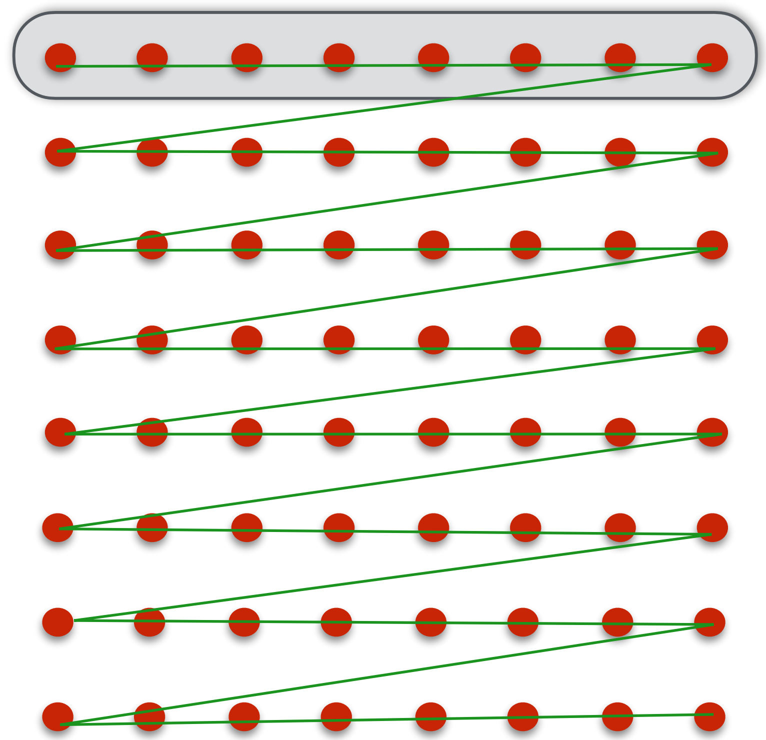
Spatial locality



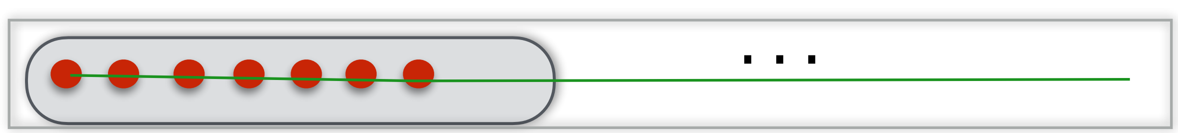
grid in default row-major order



Spatial locality



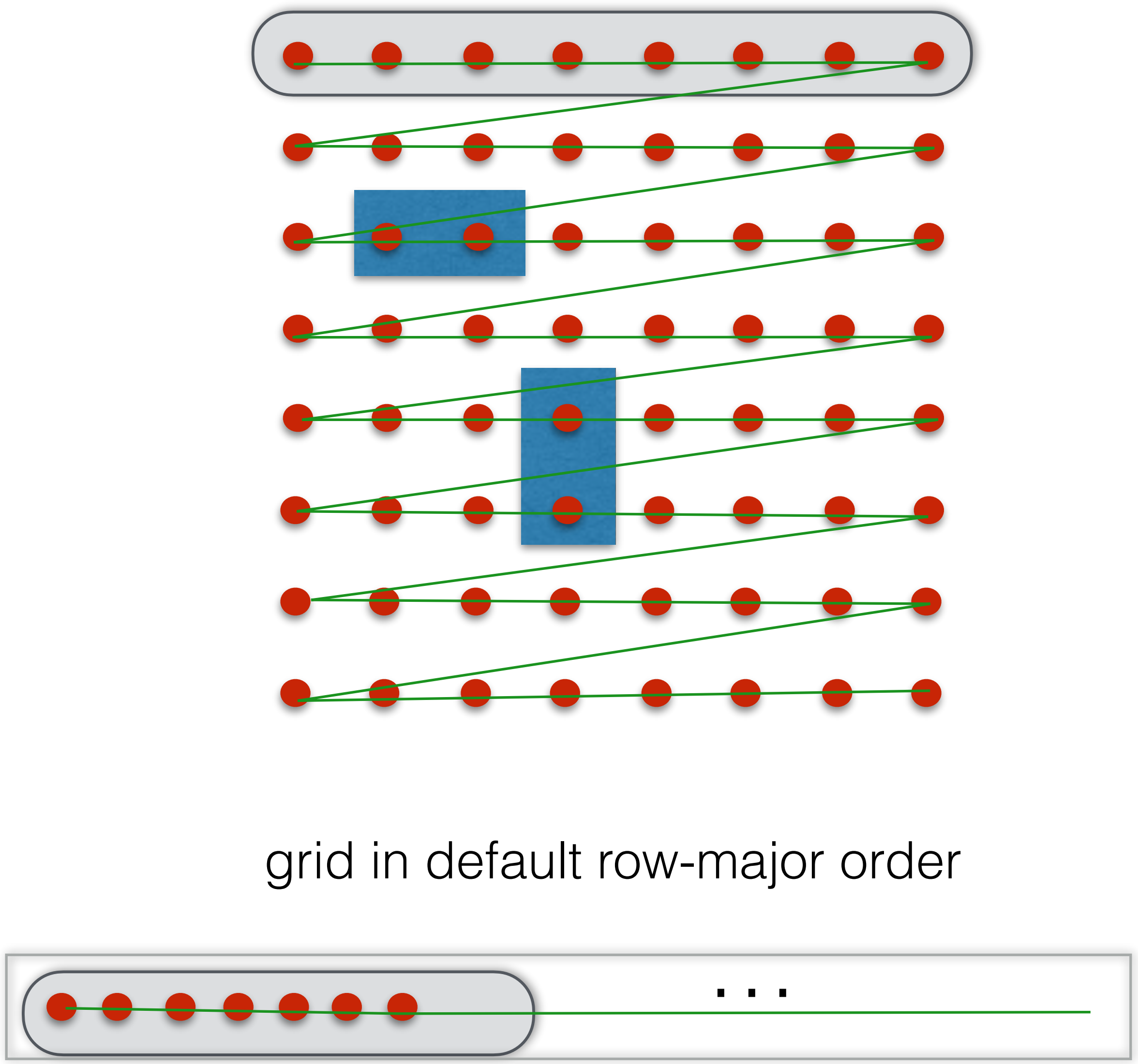
grid in default row-major order



Does this layout have good spatial locality?

- points (r,c) (r, c+/-1): how far are they in the array?
- points (r,c), (r+1,c): how far are they in the array?

Spatial locality

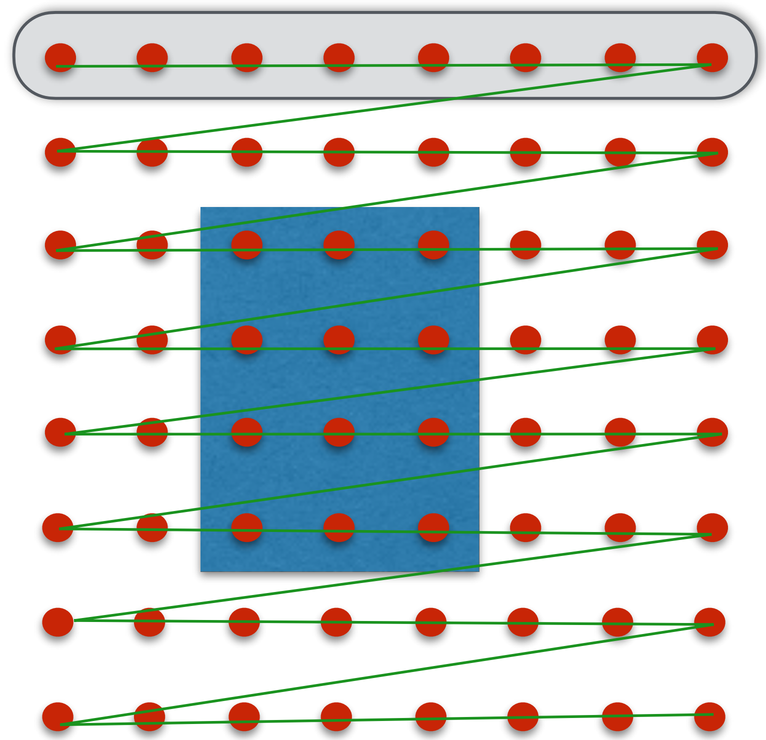


grid in default row-major order

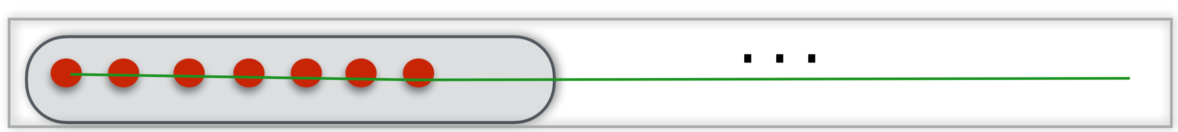
Does this layout have good spatial locality?

- points (r,c) (r, c+/-1): how far are they in the array?
- points (r,c), (r+1,c): how far are they in the array?

Spatial locality



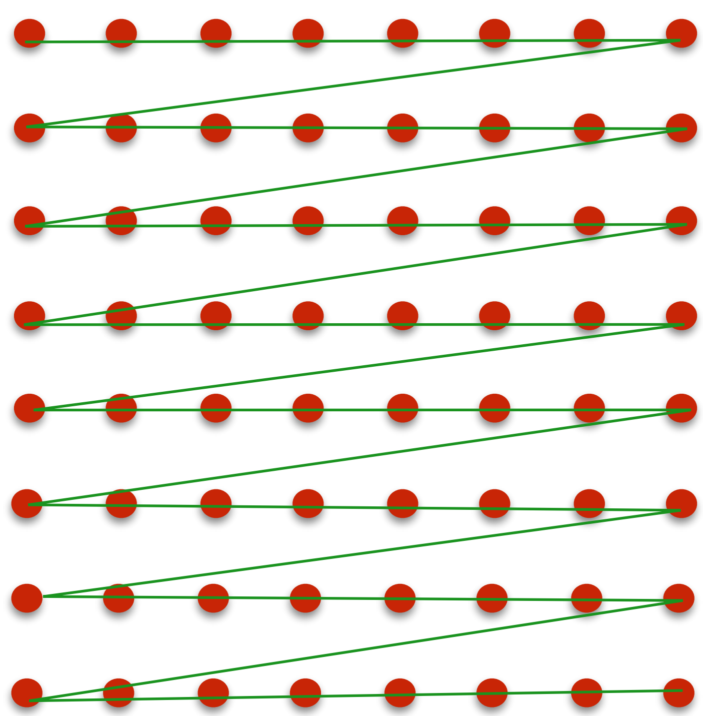
grid in default row-major order



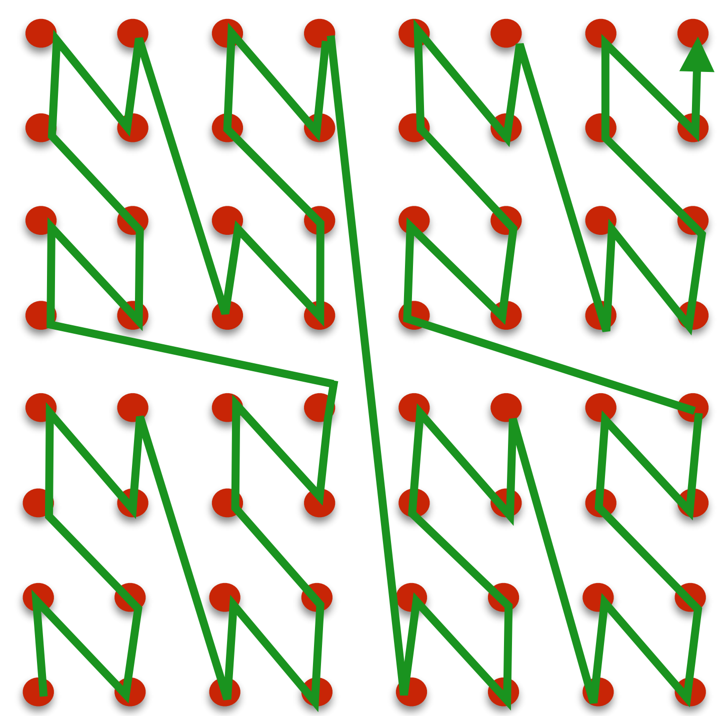
Does this layout have good spatial locality?

- points (r,c) (r, c+/-1): how far are they in the array?
- points (r,c), (r+1,c): how far are they in the array?

Spatial locality

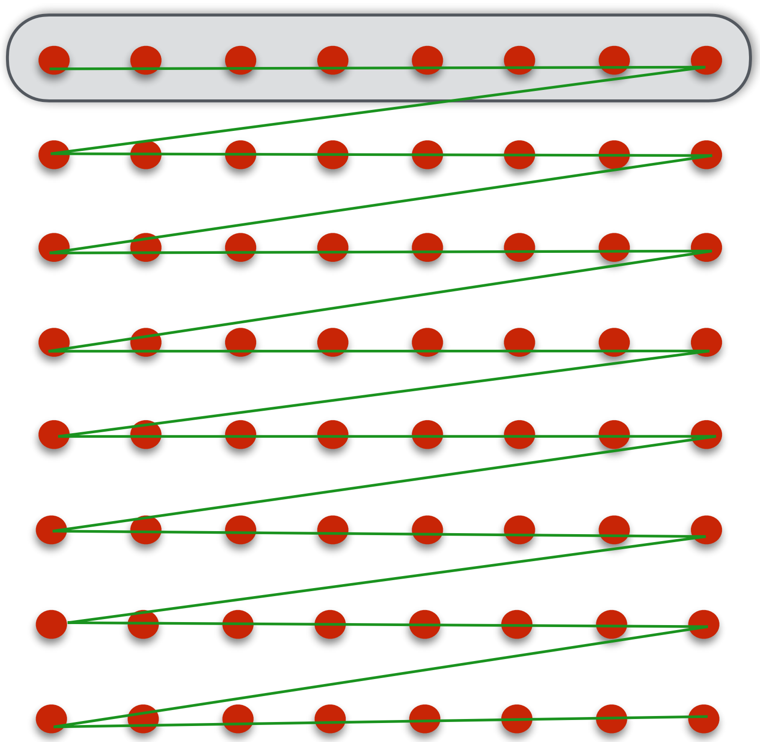


grid in default row-major order

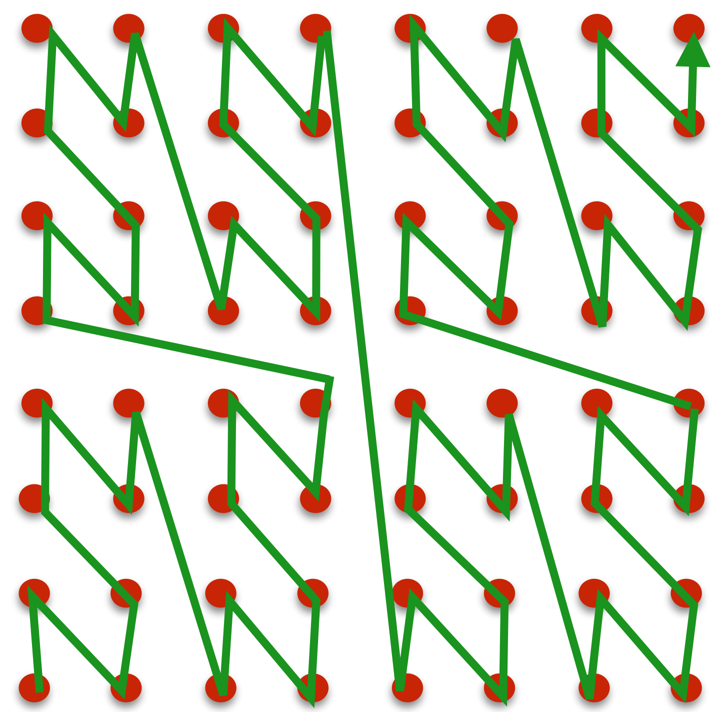


grid stored in Z-order

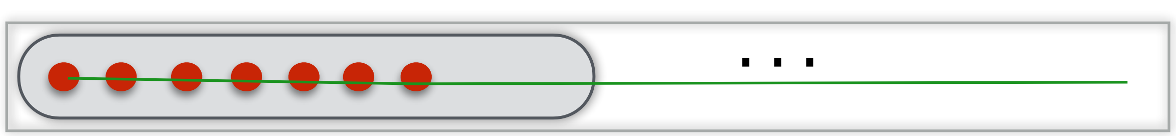
Spatial locality

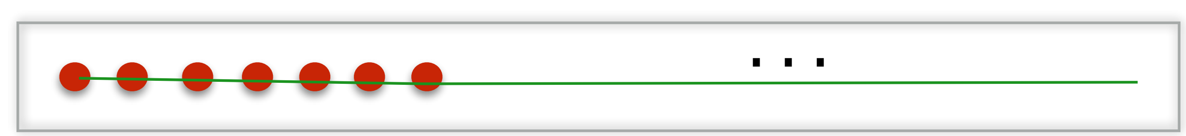


grid in default row-major order

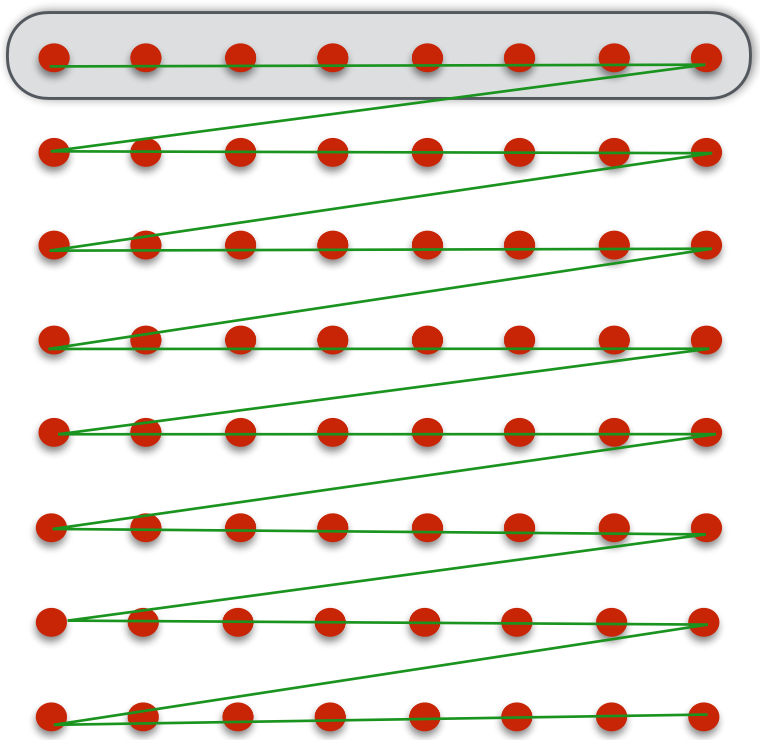


grid stored in Z-order

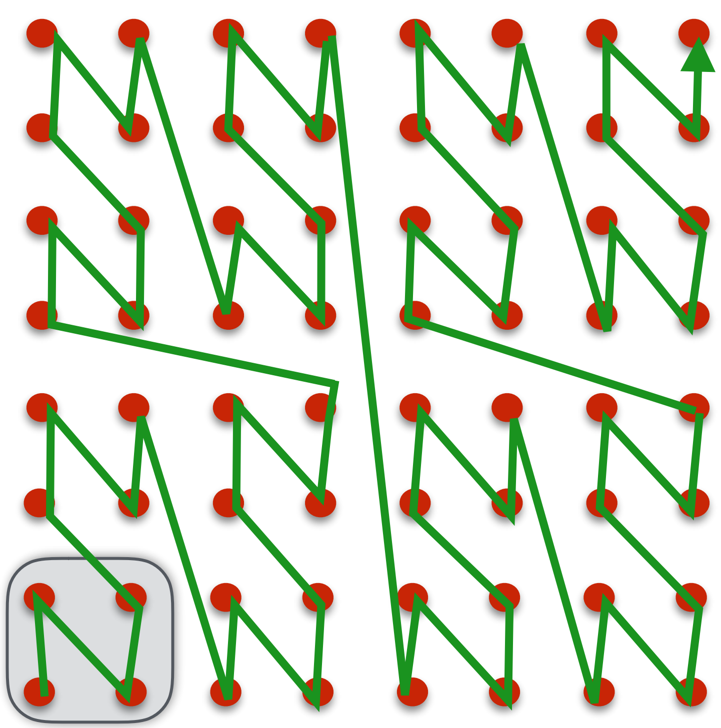




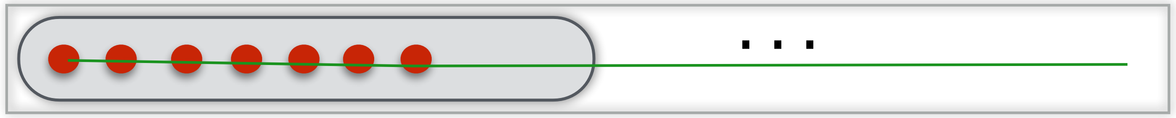
Spatial locality



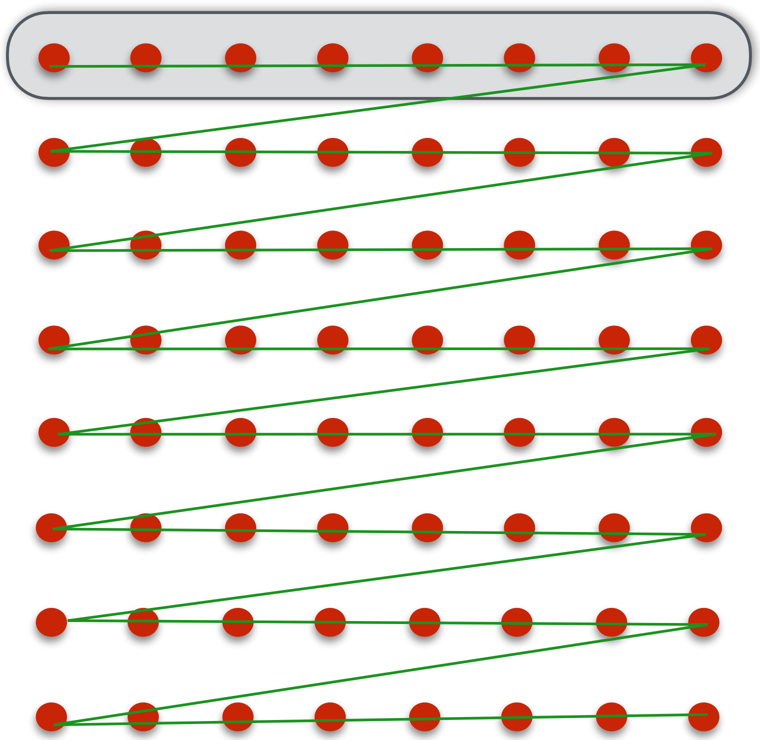
grid in default row-major order



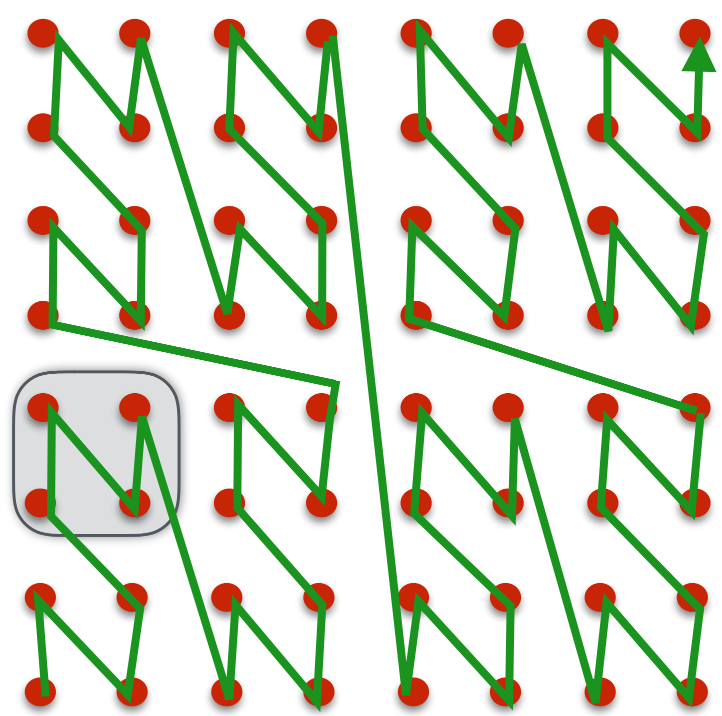
grid stored in Z-order



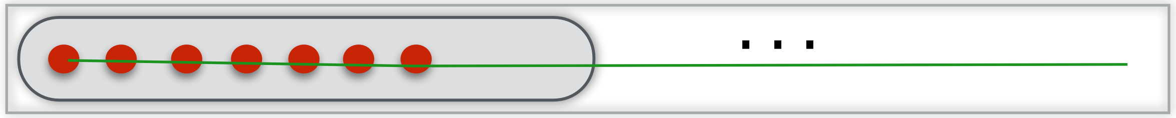
Spatial locality

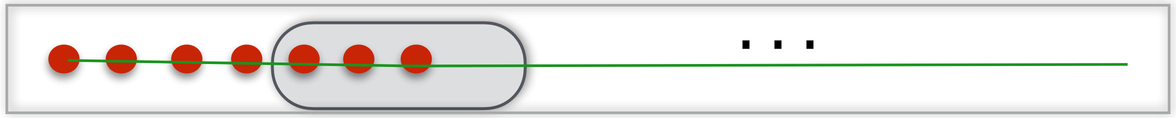


grid in default row-major order

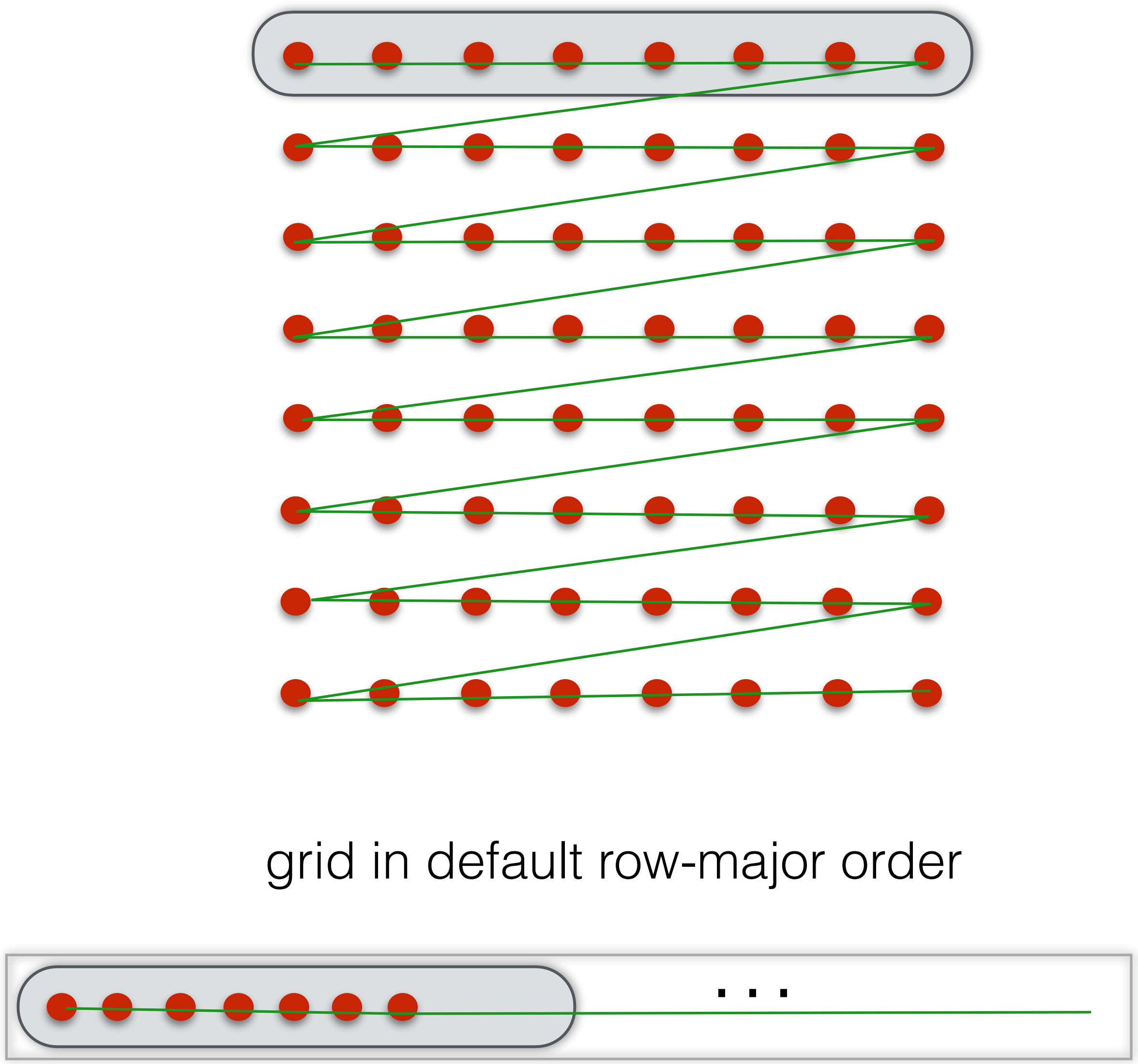


grid stored in Z-order

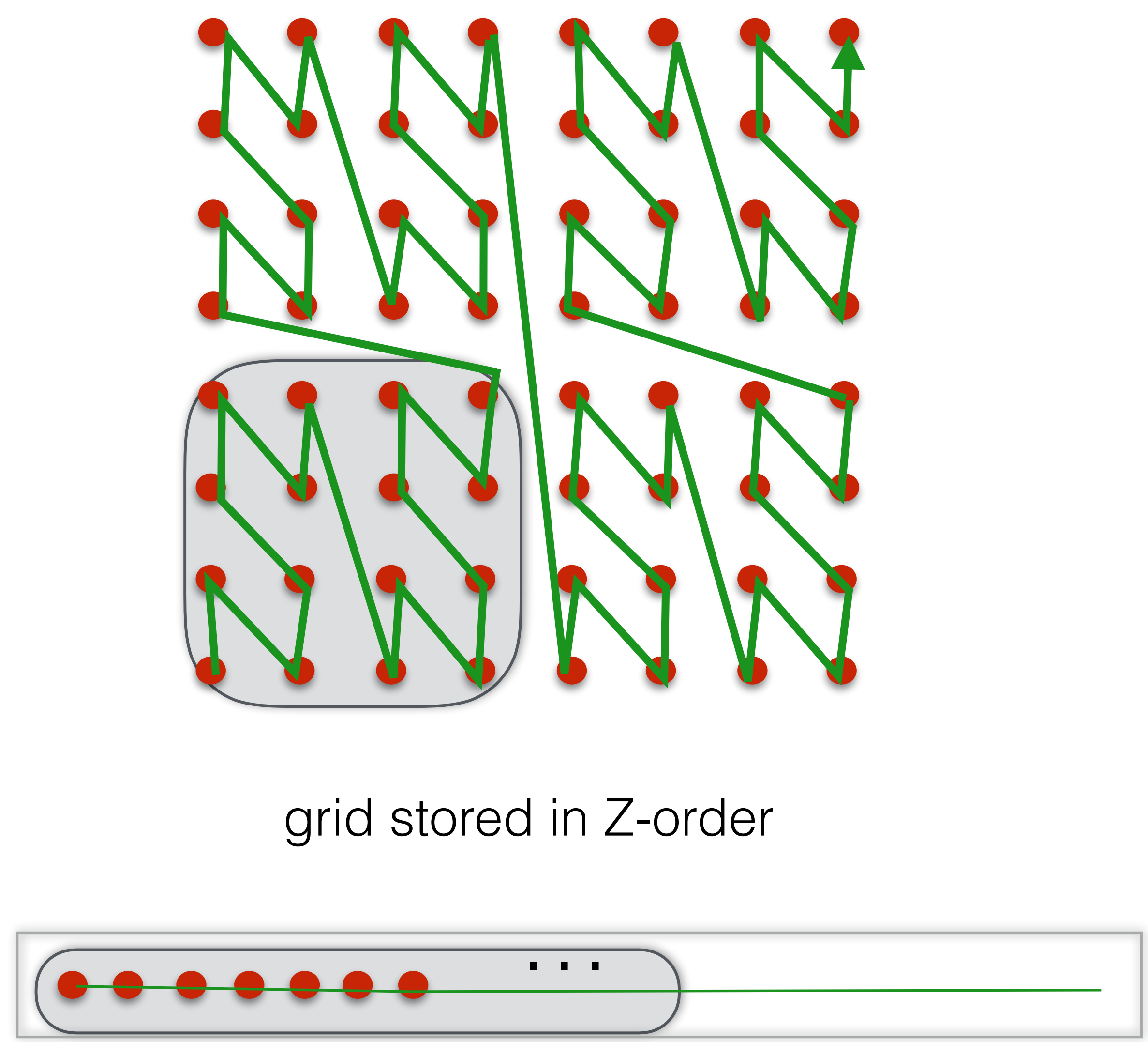




Spatial locality

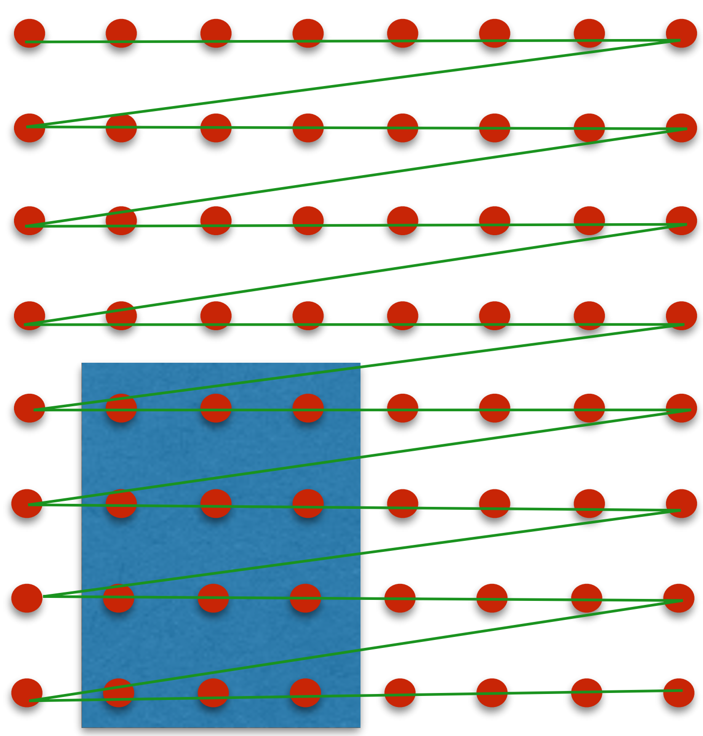


grid in default row-major order

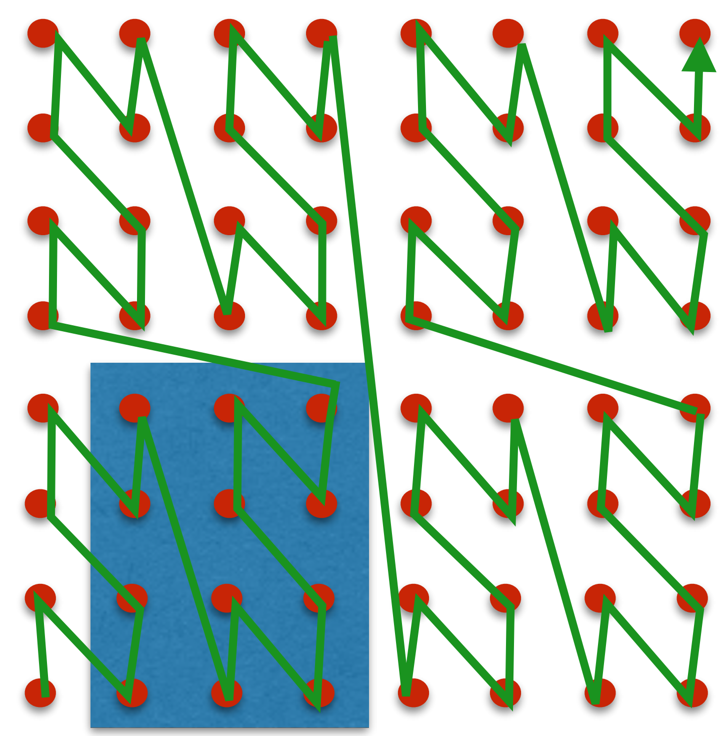


grid stored in Z-order

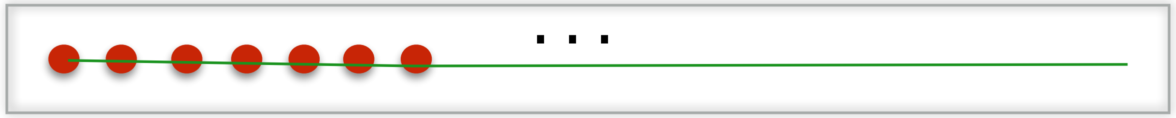
Spatial locality



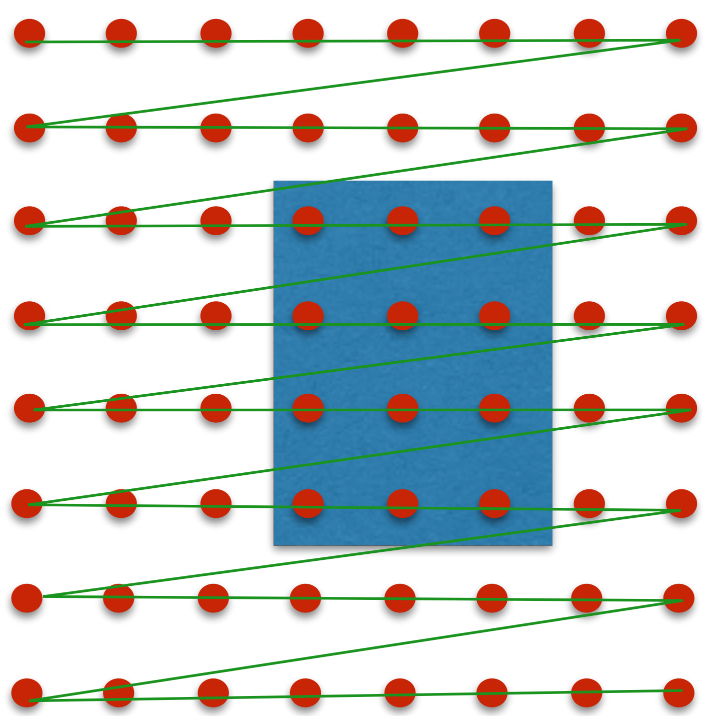
grid in default row-major order



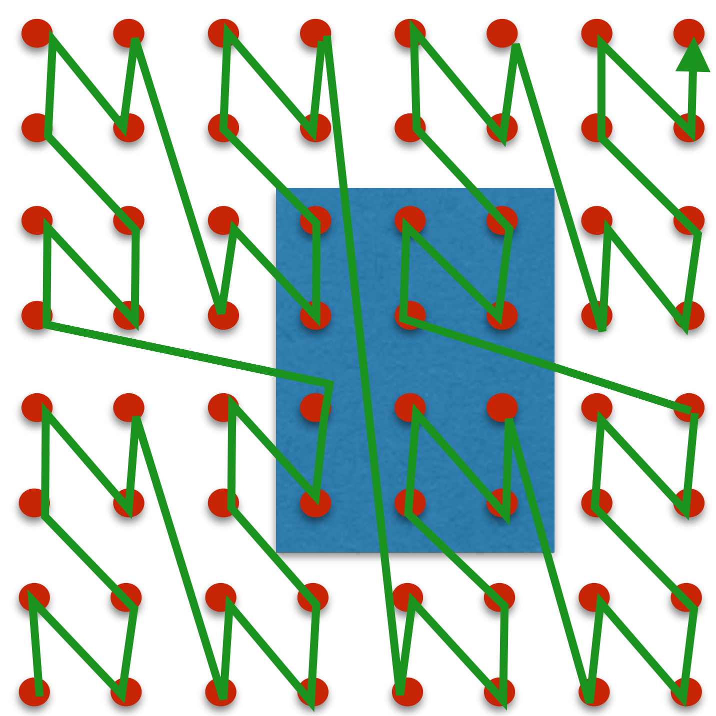
grid stored in Z-order



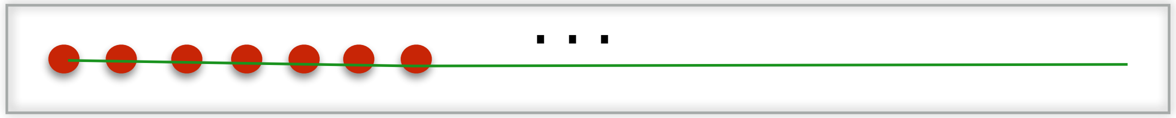
Spatial locality



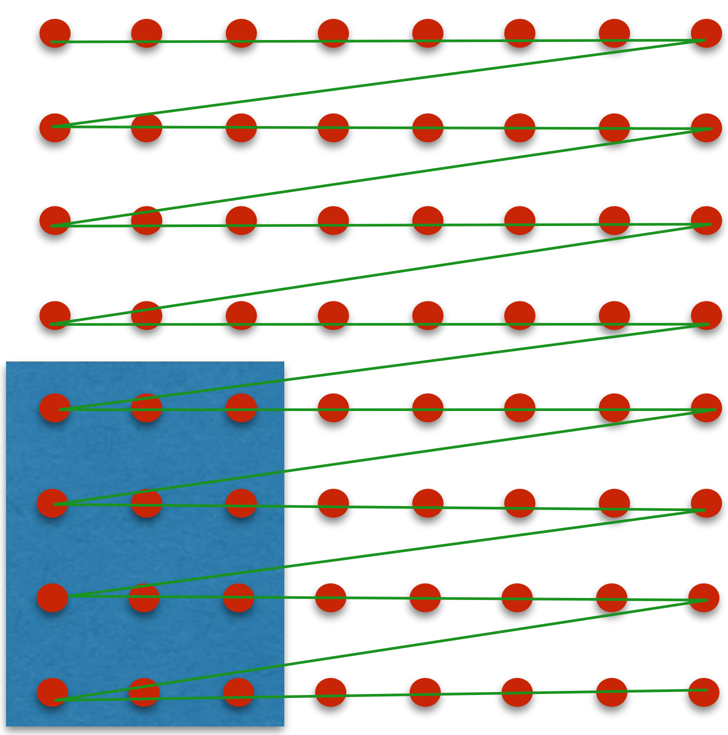
grid in default row-major order



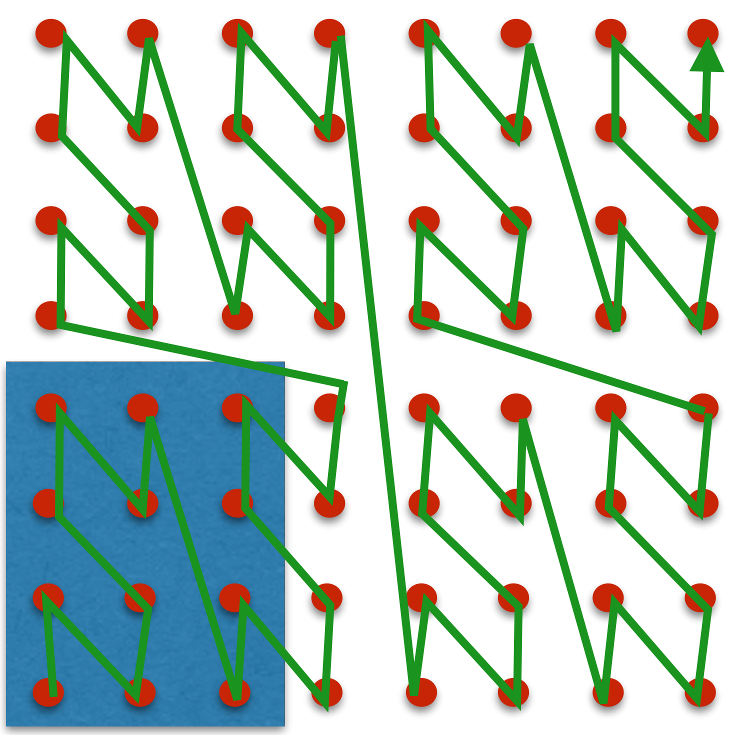
grid stored in Z-order



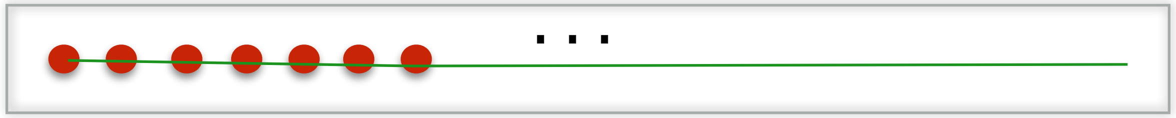
Spatial locality



grid in default row-major order



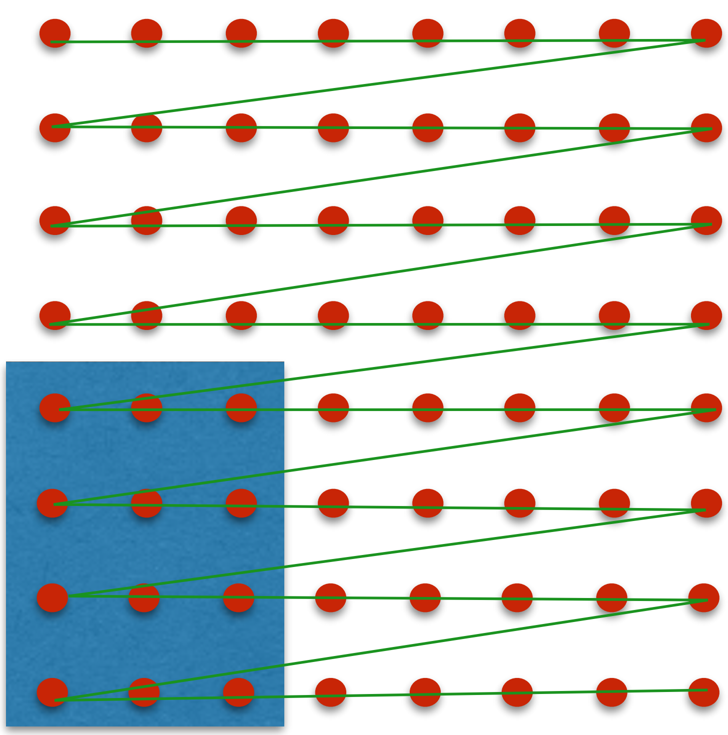
grid stored in Z-order



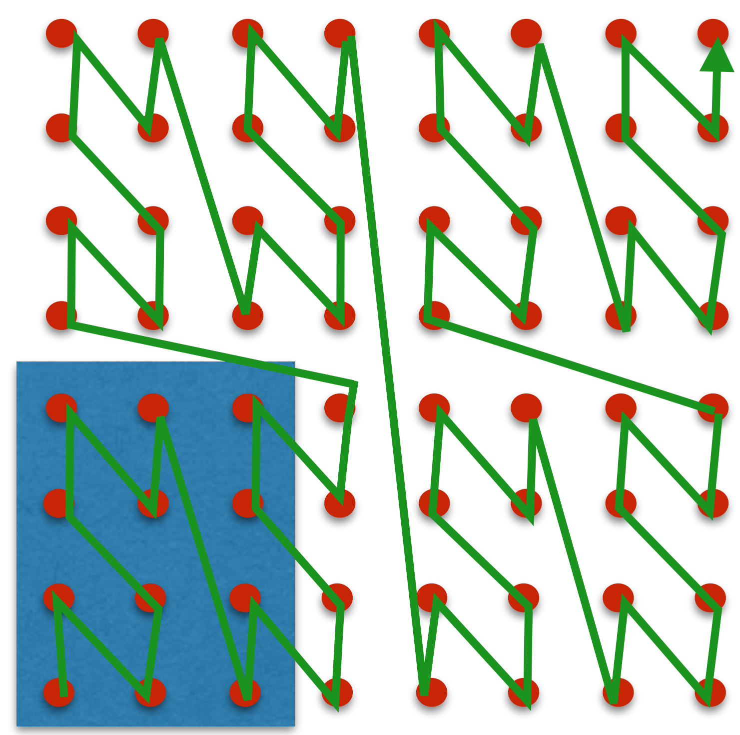
Spatial locality

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

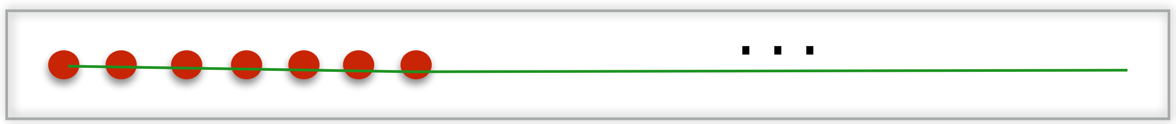
- points that are close together in space, will be stored close to each other

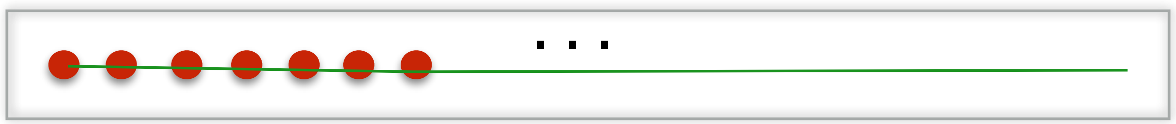


grid in default row-major order



grid stored in Z-order

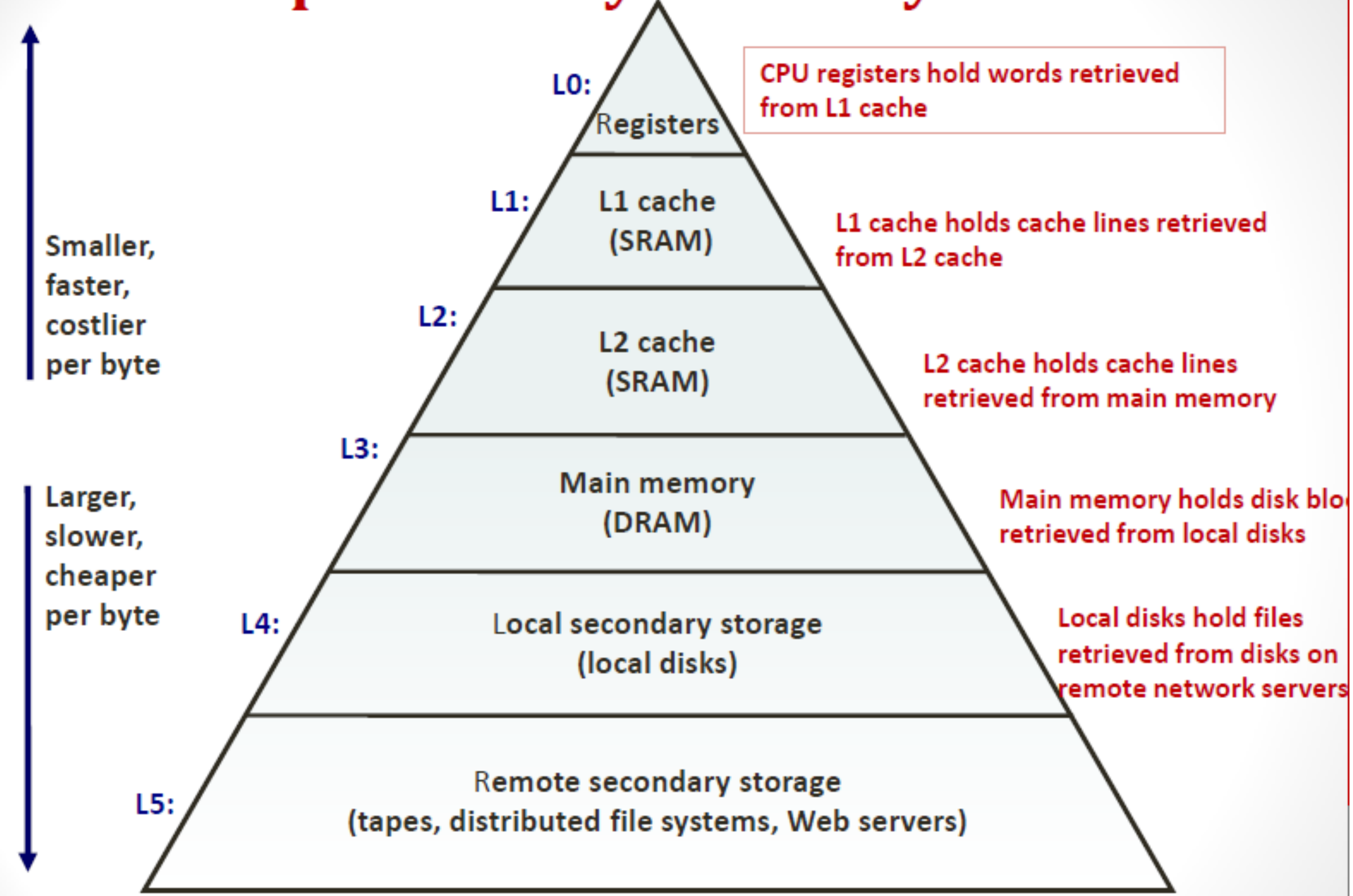




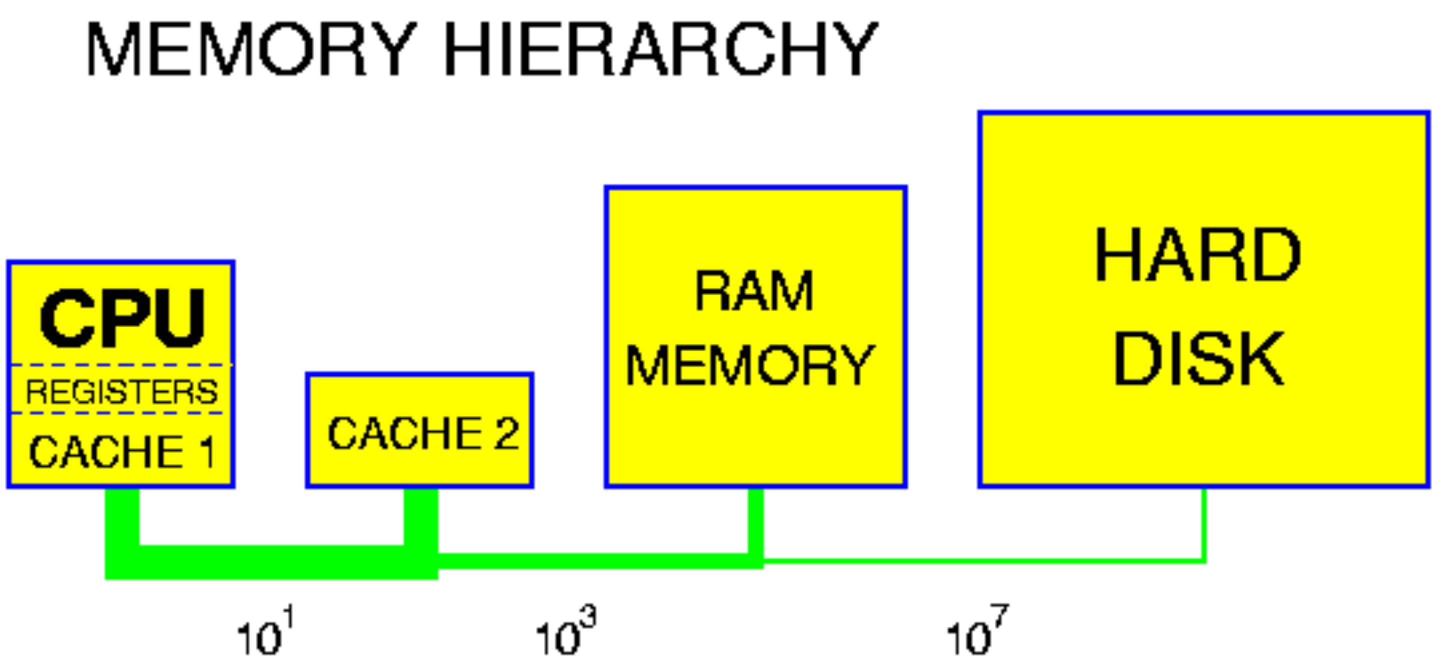
Spatial locality

- 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

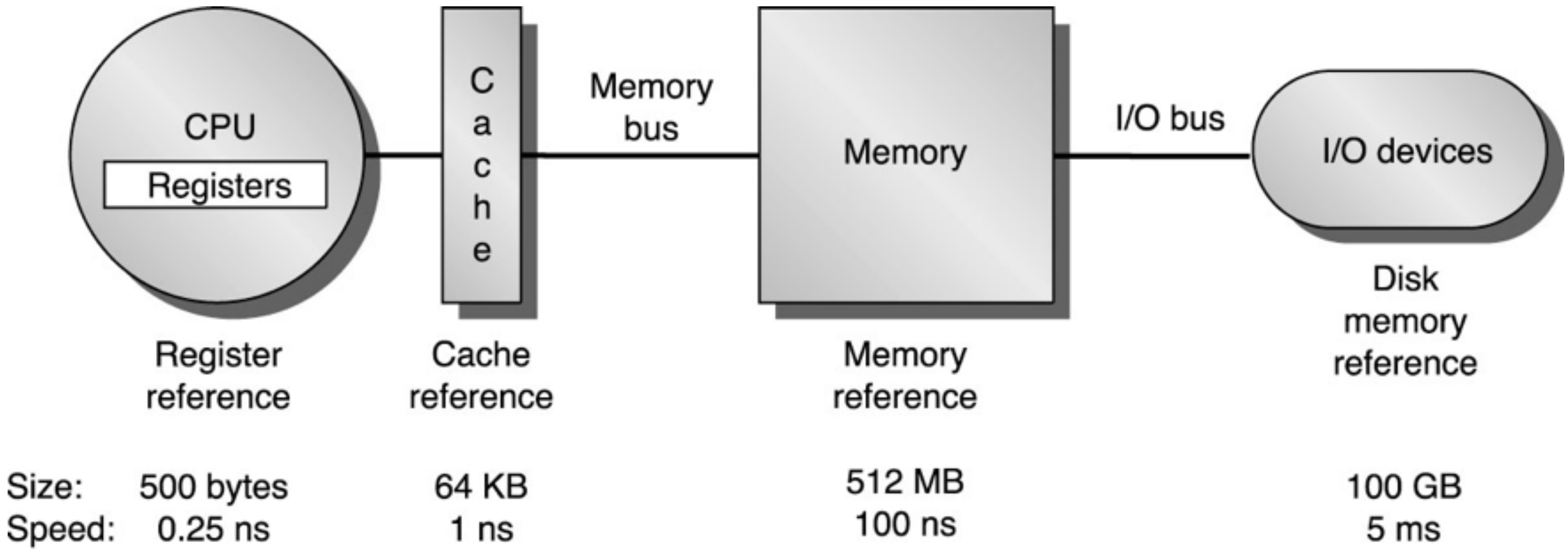


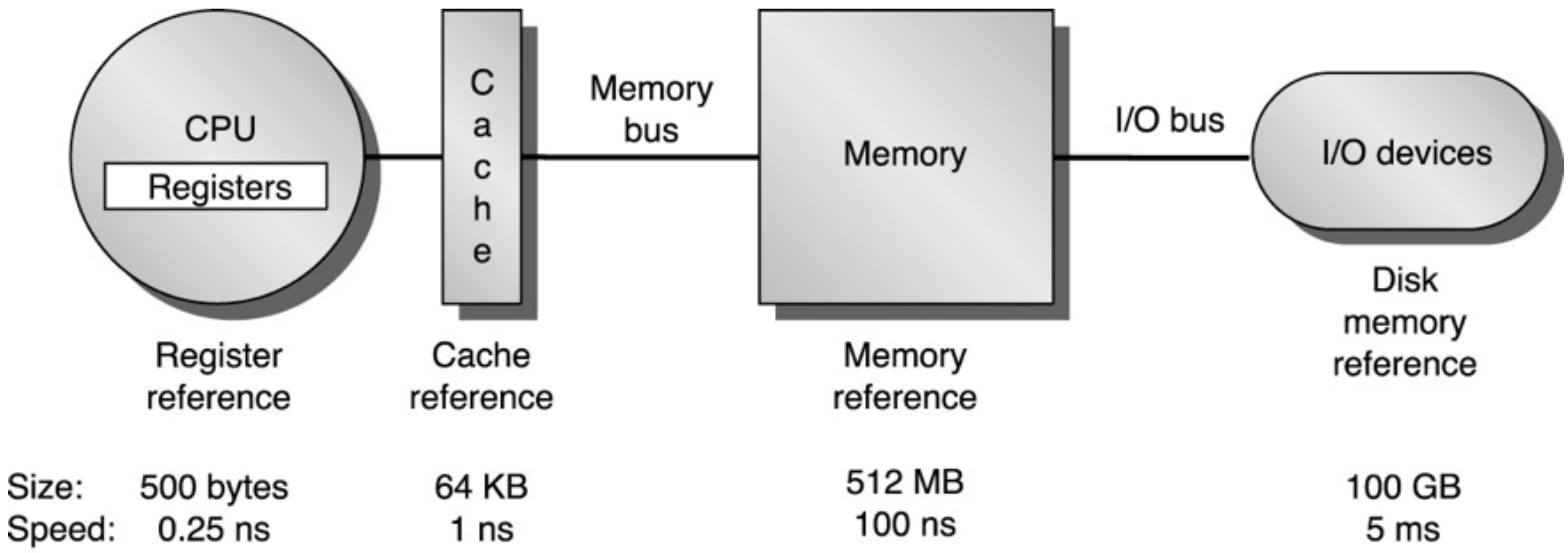
http://classconnection.s3.amazonaws.com/149/flashcards/3088149/png/memory_hierarchy1367201501848.png



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


CPU
Registers
C
a
c
h
e
Memory
bus
Memory
I/O bus
I/O devices
Register
reference
Cache
reference
Memory
reference
Disk
memory
reference
Size:
500 bytes
64 KB
512 MB
100 GB
Speed:
0.25 ns
1 ns
100 ns
5 ms



- 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!

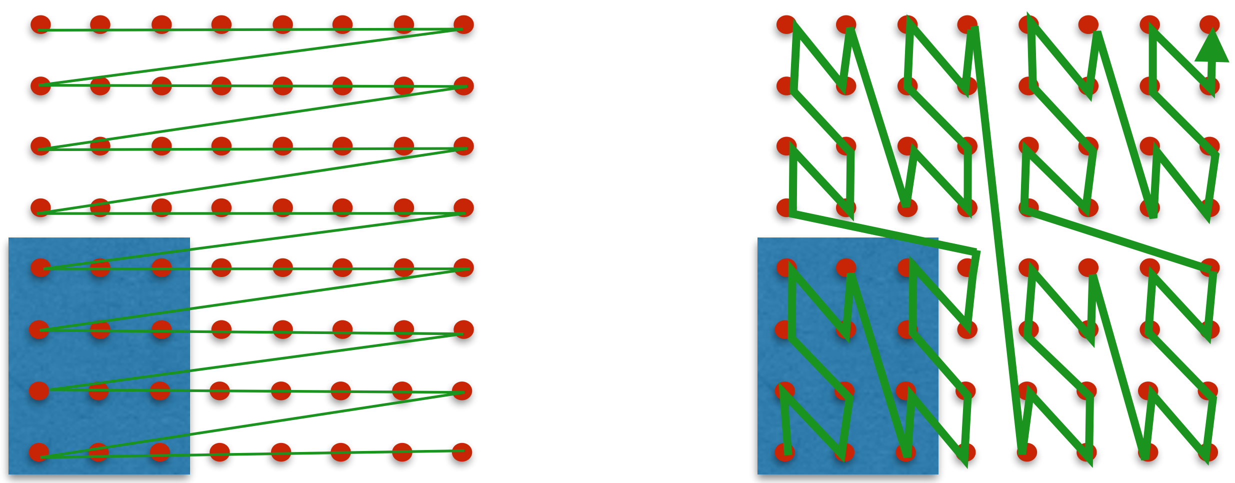
Spatial locality

- 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/