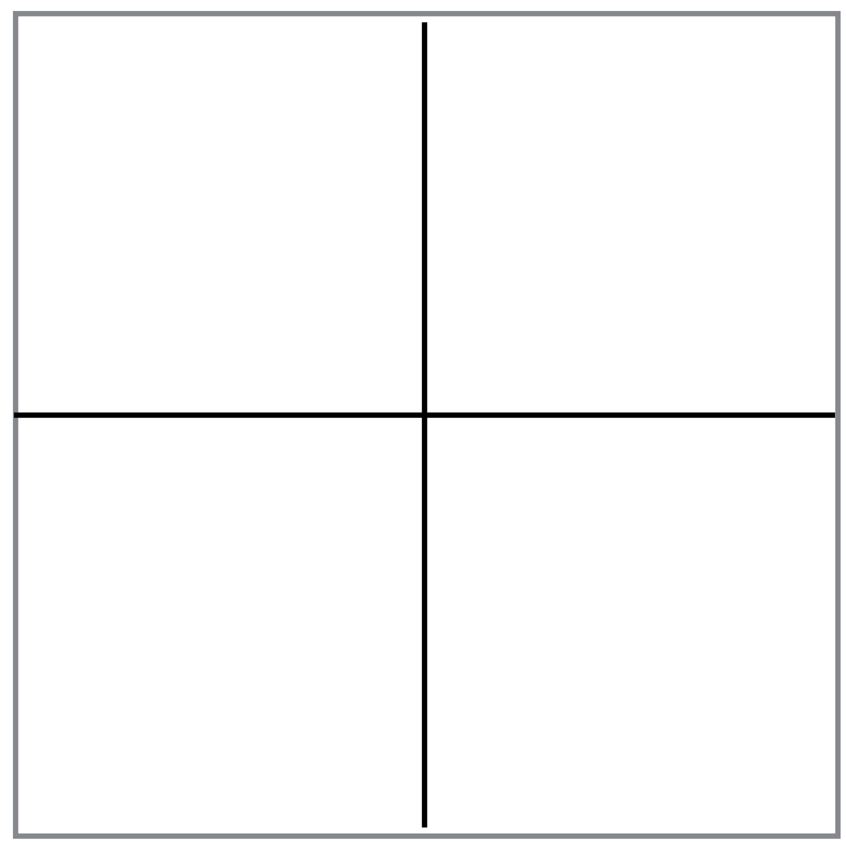
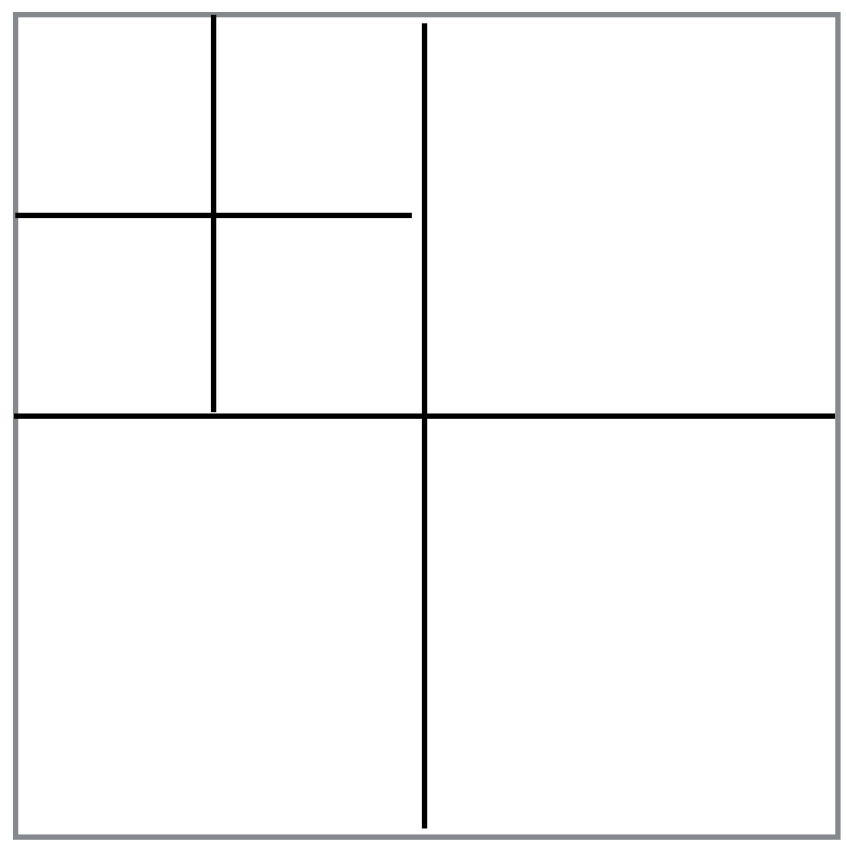
Algorithms for GIS:

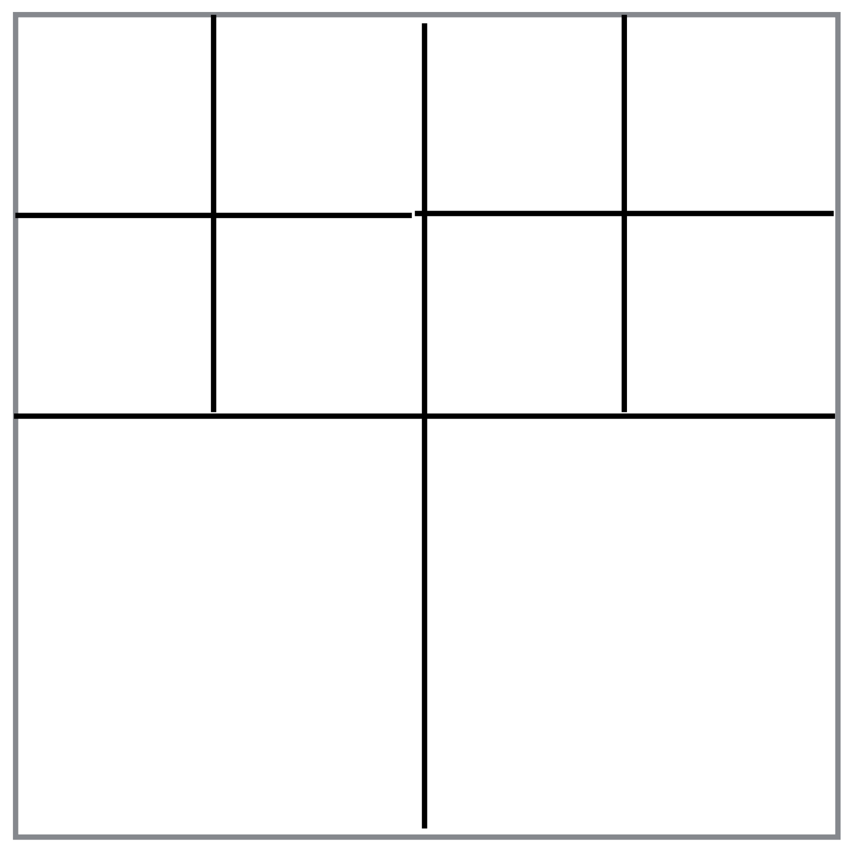
Quadtrees

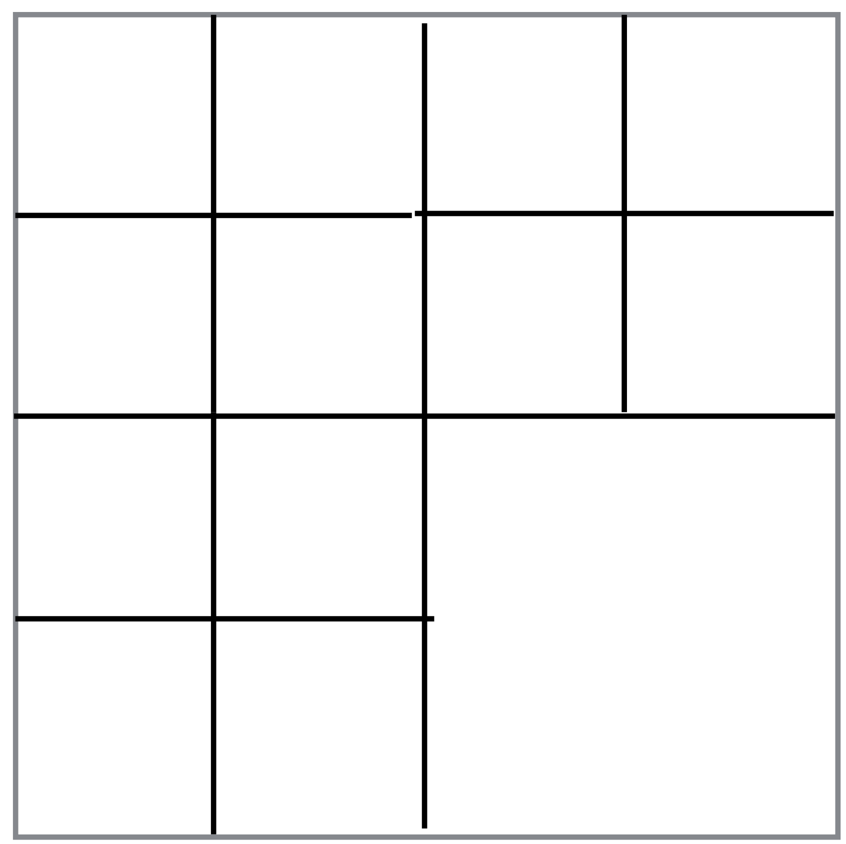
Quadtree

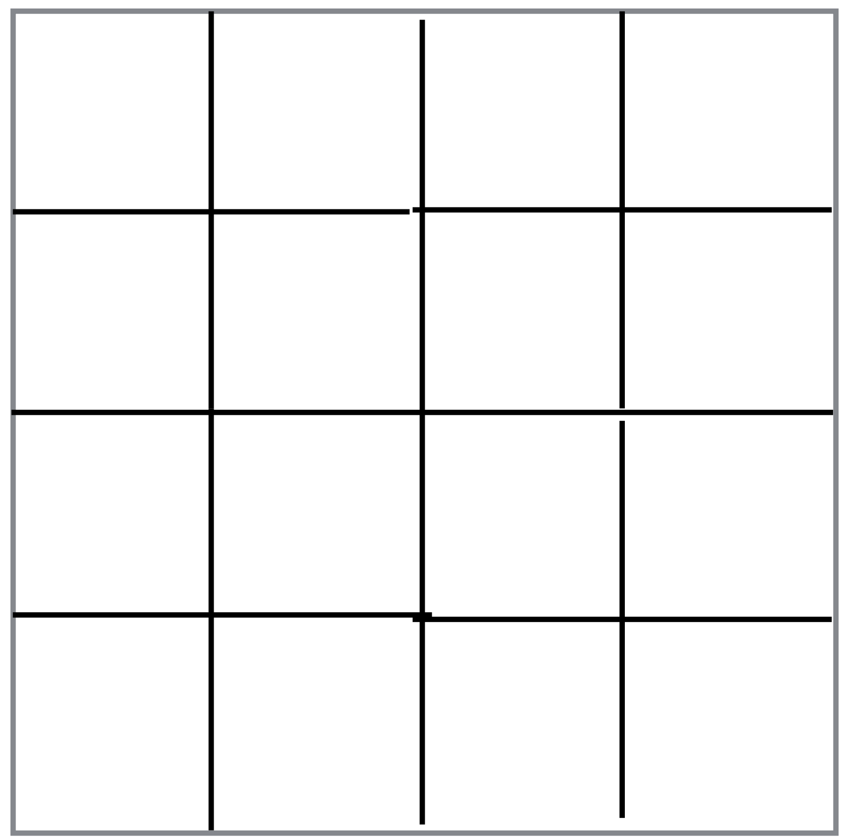
- A data structure that corresponds to a hierarchical subdivision of the plane
- Start with a square (containing inside input data)
    - Divide into 4 equal squares (quadrants)
    - Continue subdividing each quadrant recursively
    - Subdivide a square until it satisfies a stopping condition, usually that a quadrant is “small” enough
        - for e.g. contains at most 1 point

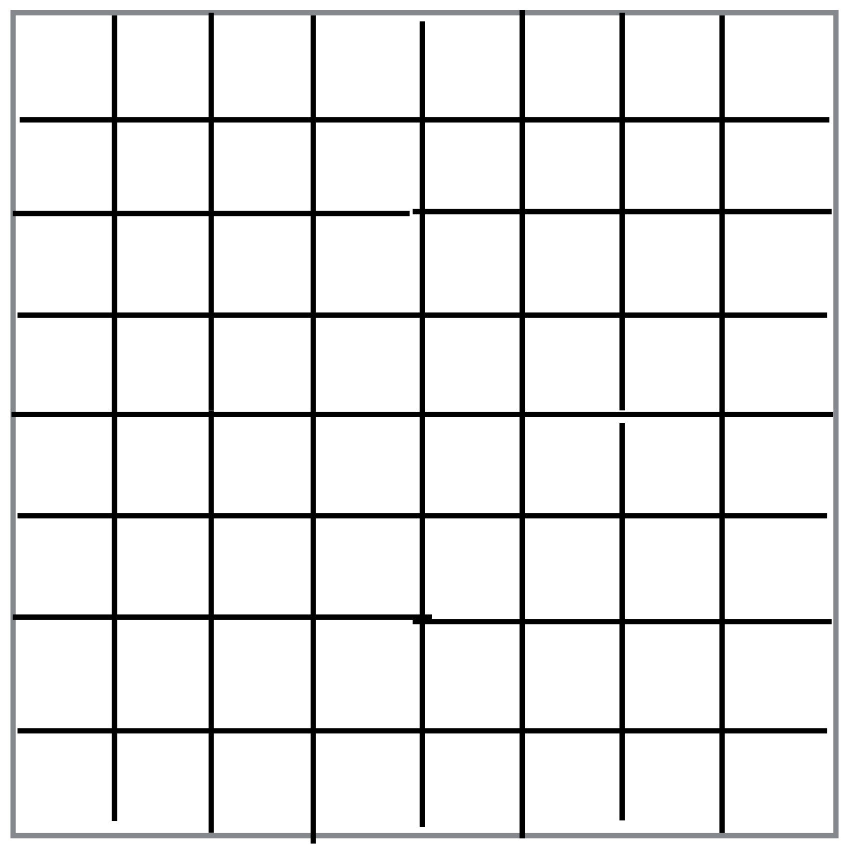


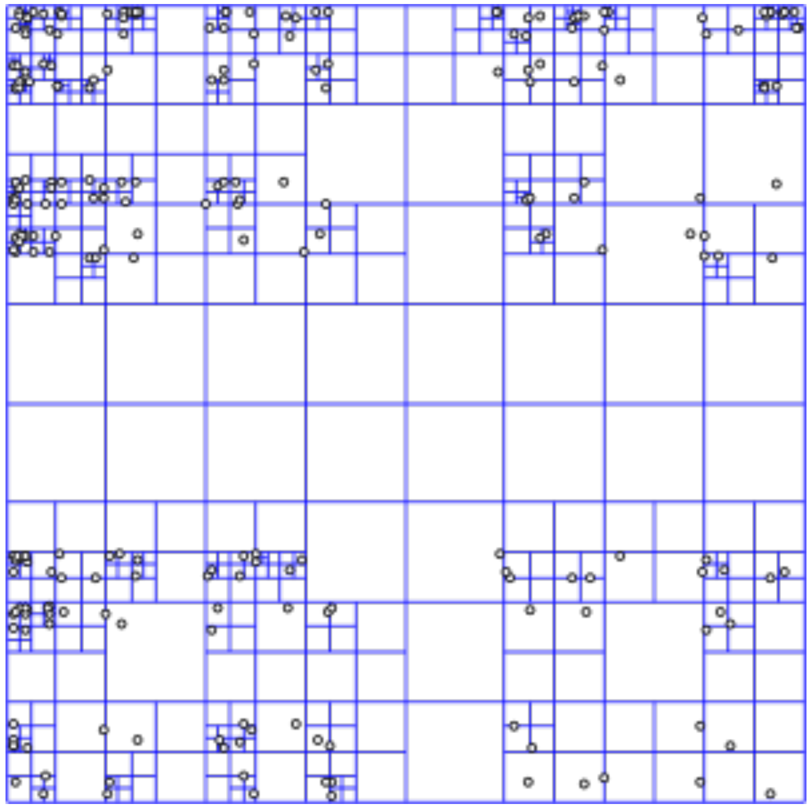












Quadtrees

Quadtrees

- Conceptually simple data structure

Quadtrees

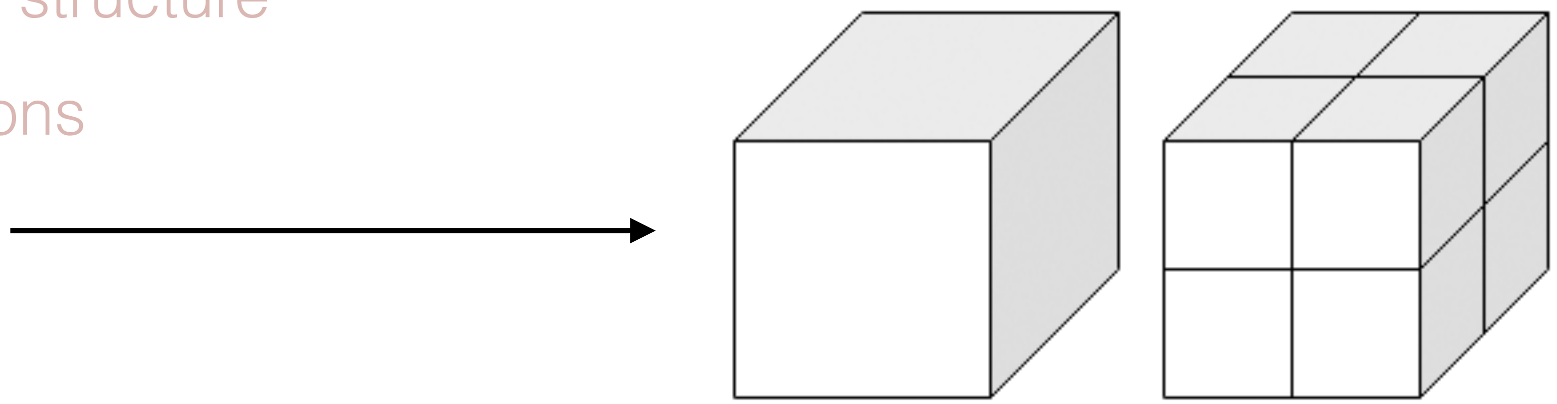
- Conceptually simple data structure
- Generalizes to d dimensions

Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
    - d=3: octree

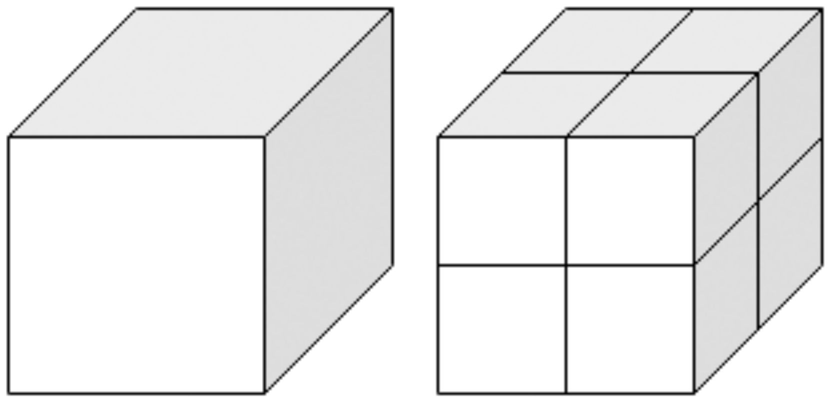
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
    - d=3: octree



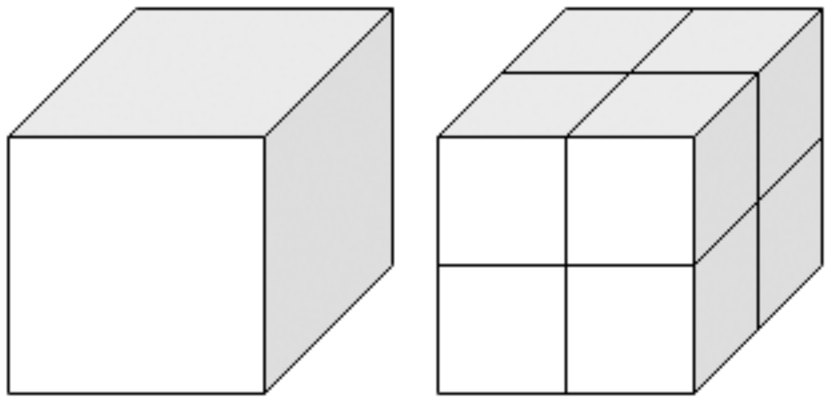
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
  - d=3: octree
- Can be built for many types of data



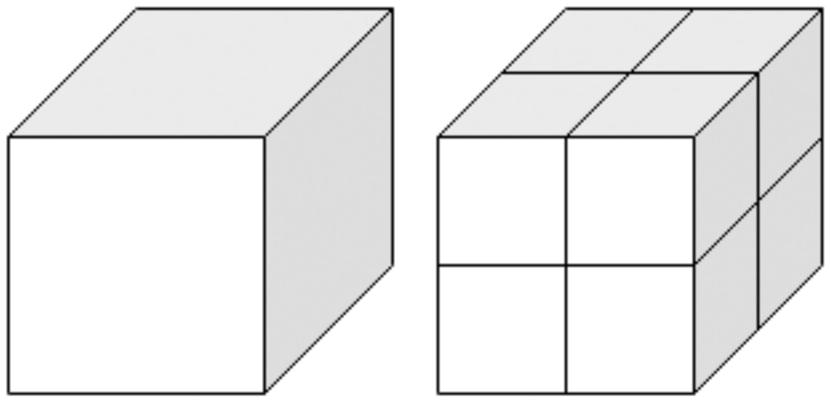
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
  - d=3: octree
- Can be built for many types of data
  - points, edges, polygons, images, etc



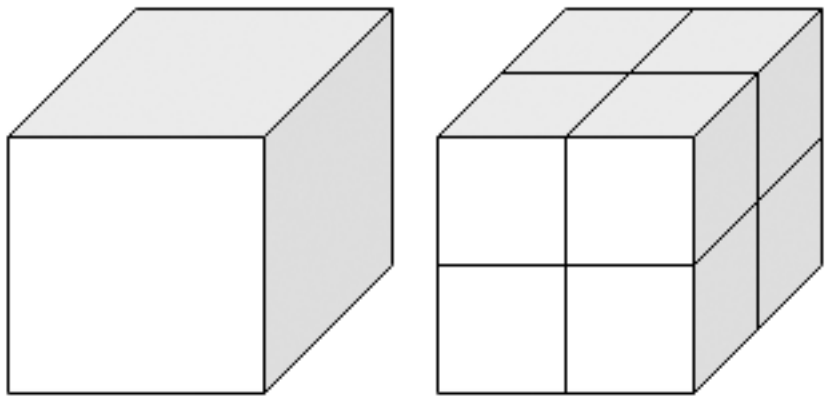
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
  - d=3: octree
- Can be built for many types of data
  - points, edges, polygons, images, etc
- Can be used for many different tasks



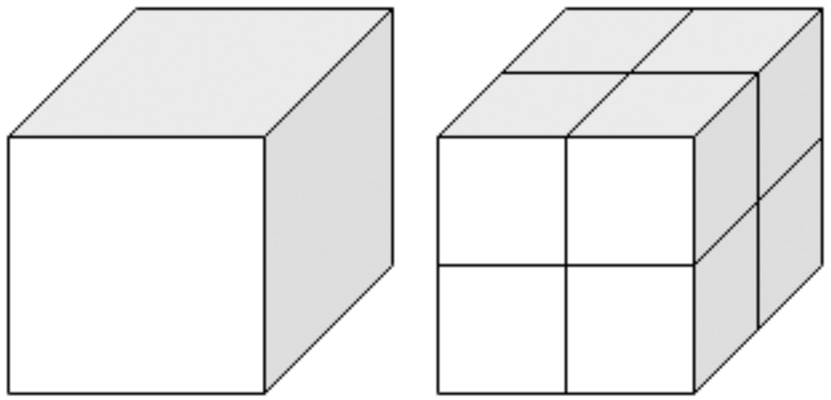
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
    - d=3: octree
- Can be built for many types of data
    - points, edges, polygons, images, etc
- Can be used for many different tasks
    - search, point location, neighbors, joins, unions, etc



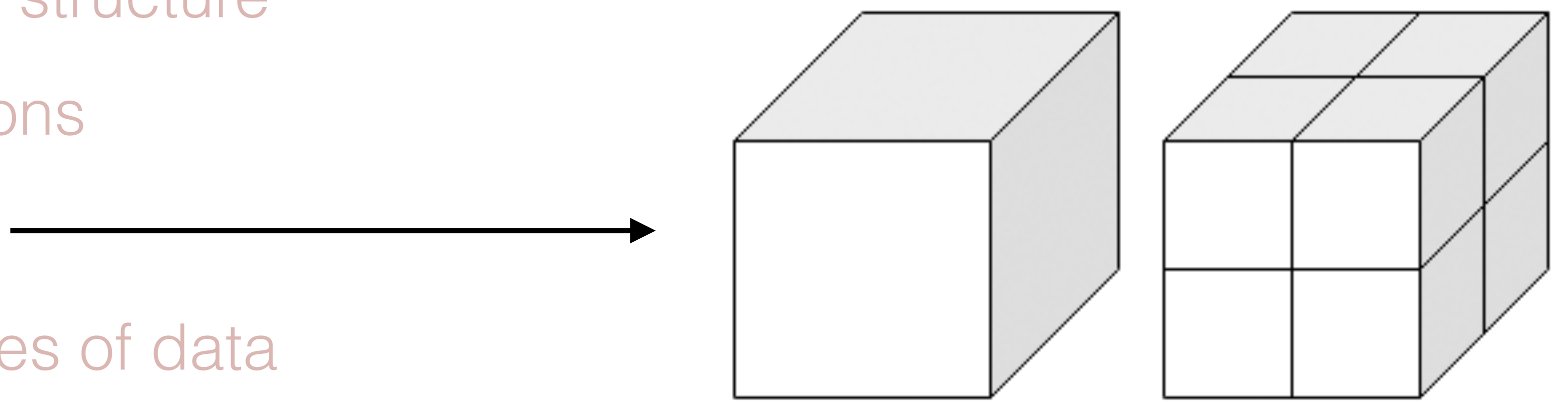
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
  - d=3: octree
- Can be built for many types of data
  - points, edges, polygons, images, etc
- Can be used for many different tasks
  - search, point location, neighbors, joins, unions, etc
  - dynamic



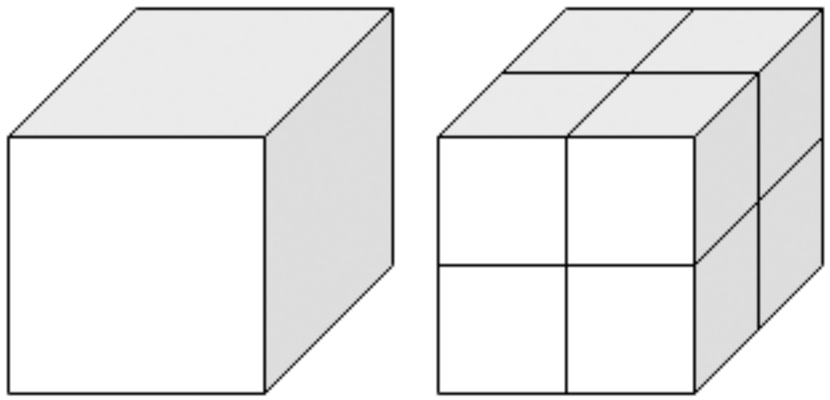
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
  - d=3: octree
- Can be built for many types of data
  - points, edges, polygons, images, etc
- Can be used for many different tasks
  - search, point location, neighbors, joins, unions, etc
  - dynamic
- Theoretical bounds not great, but widely used in practice



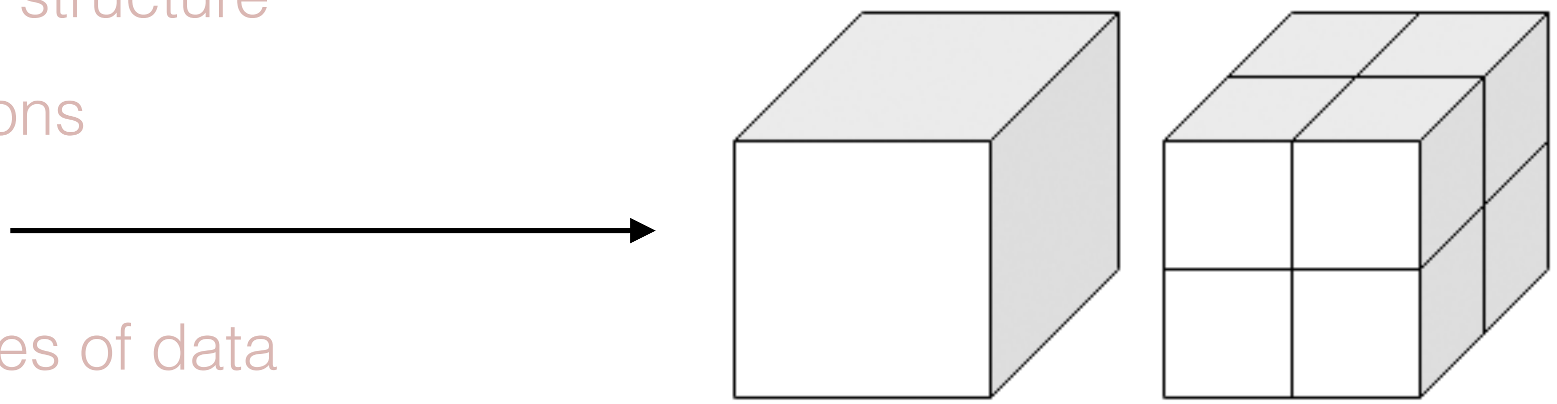
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
    - d=3: octree
- Can be built for many types of data
    - points, edges, polygons, images, etc
- Can be used for many different tasks
    - search, point location, neighbors, joins, unions, etc
    - dynamic
- Theoretical bounds not great, but widely used in practice
- LOTS of applications



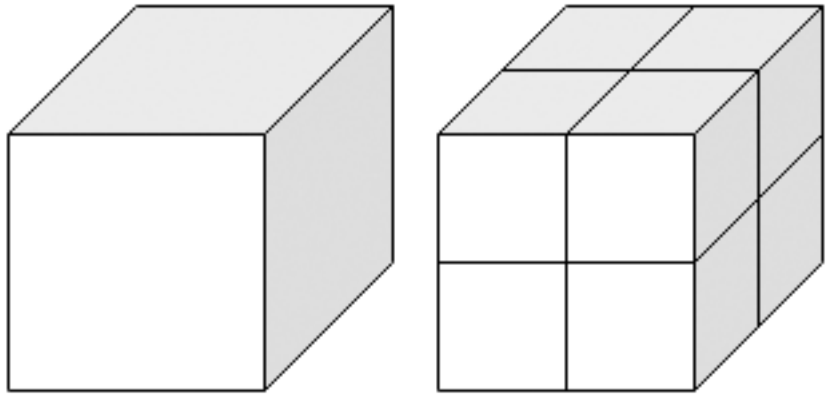
Quadtrees

- Conceptually simple data structure
- Generalizes to d dimensions
  - d=3: octree
- Can be built for many types of data
  - points, edges, polygons, images, etc
- Can be used for many different tasks
  - search, point location, neighbors, joins, unions, etc
  - dynamic
- Theoretical bounds not great, but widely used in practice
- LOTS of applications
  - Many variants of quadtrees have been proposed



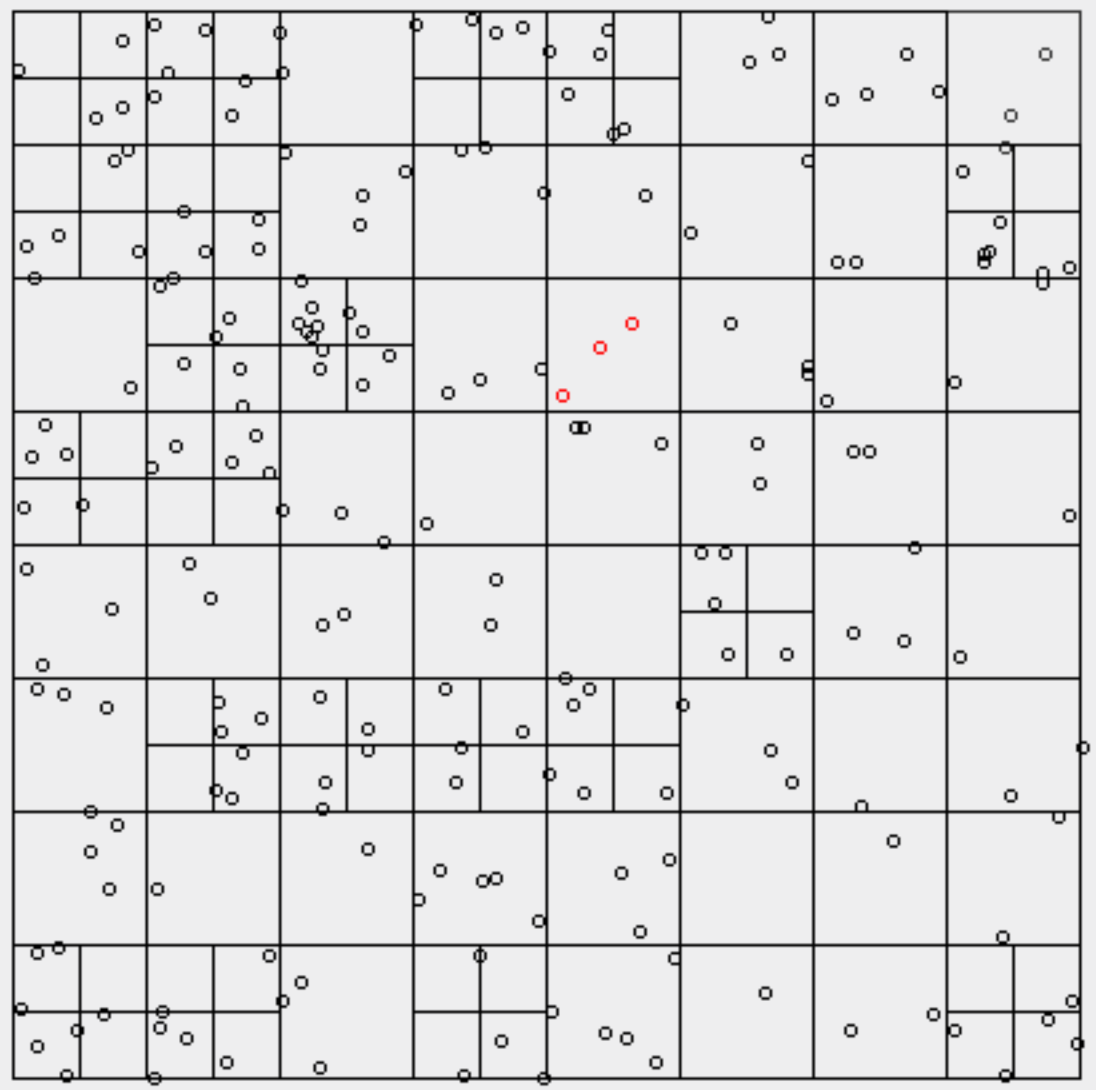
Quadtrees

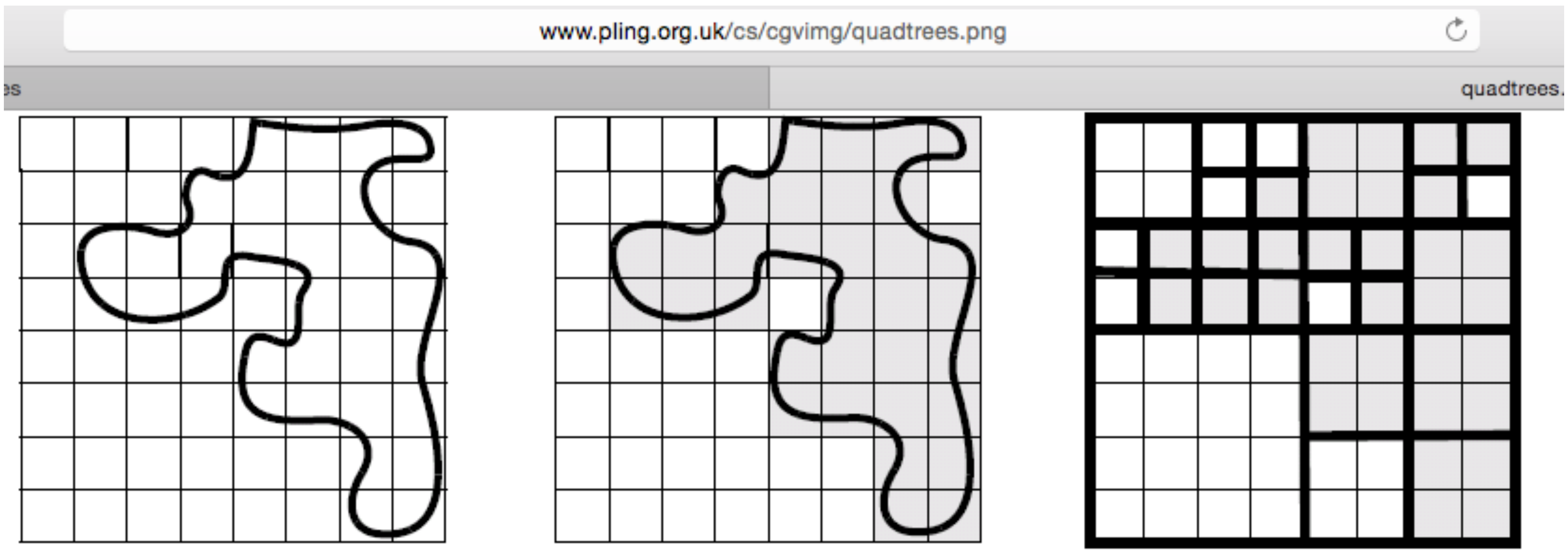
- Conceptually simple data structure
- Generalizes to d dimensions
  - d=3: octree

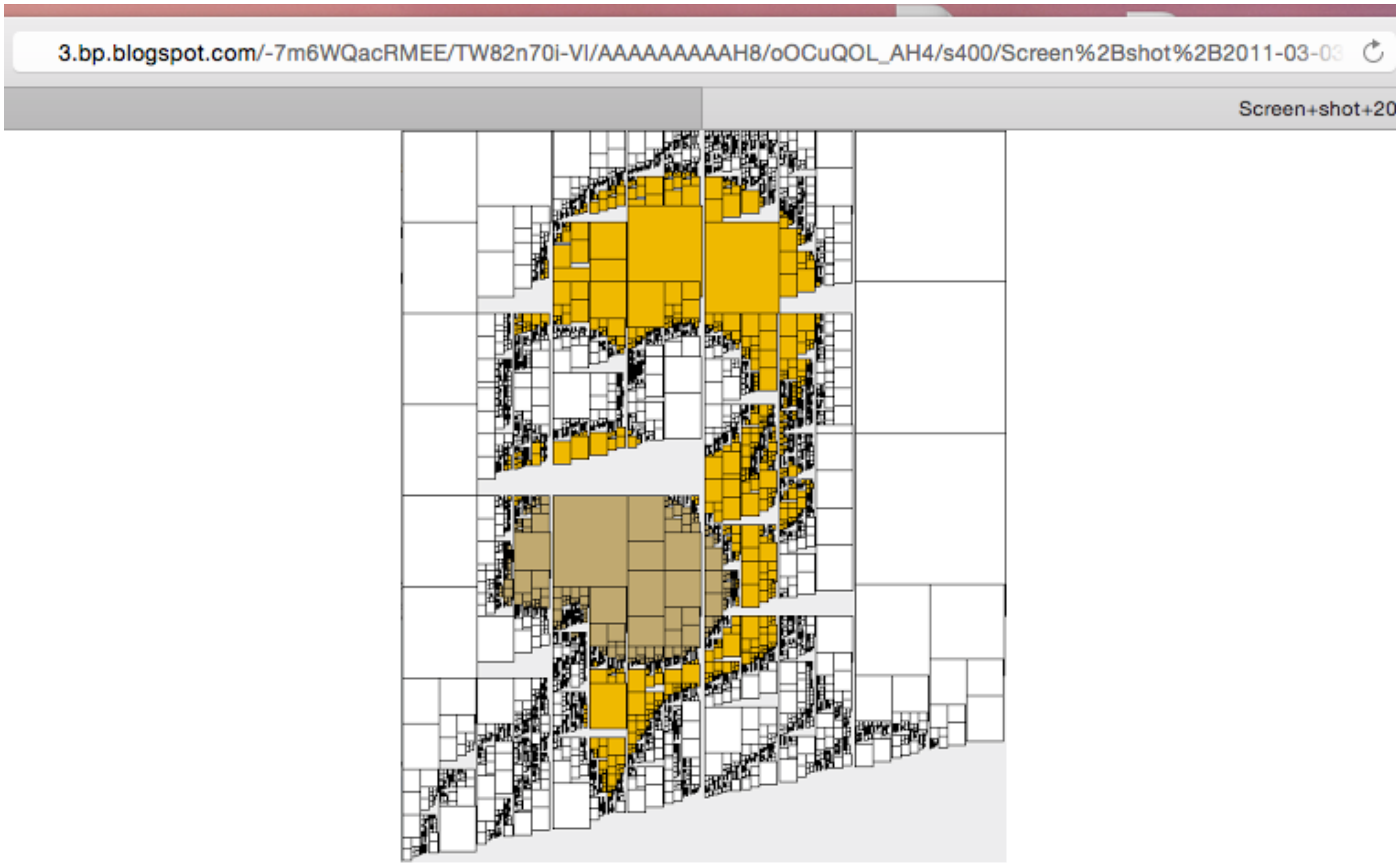


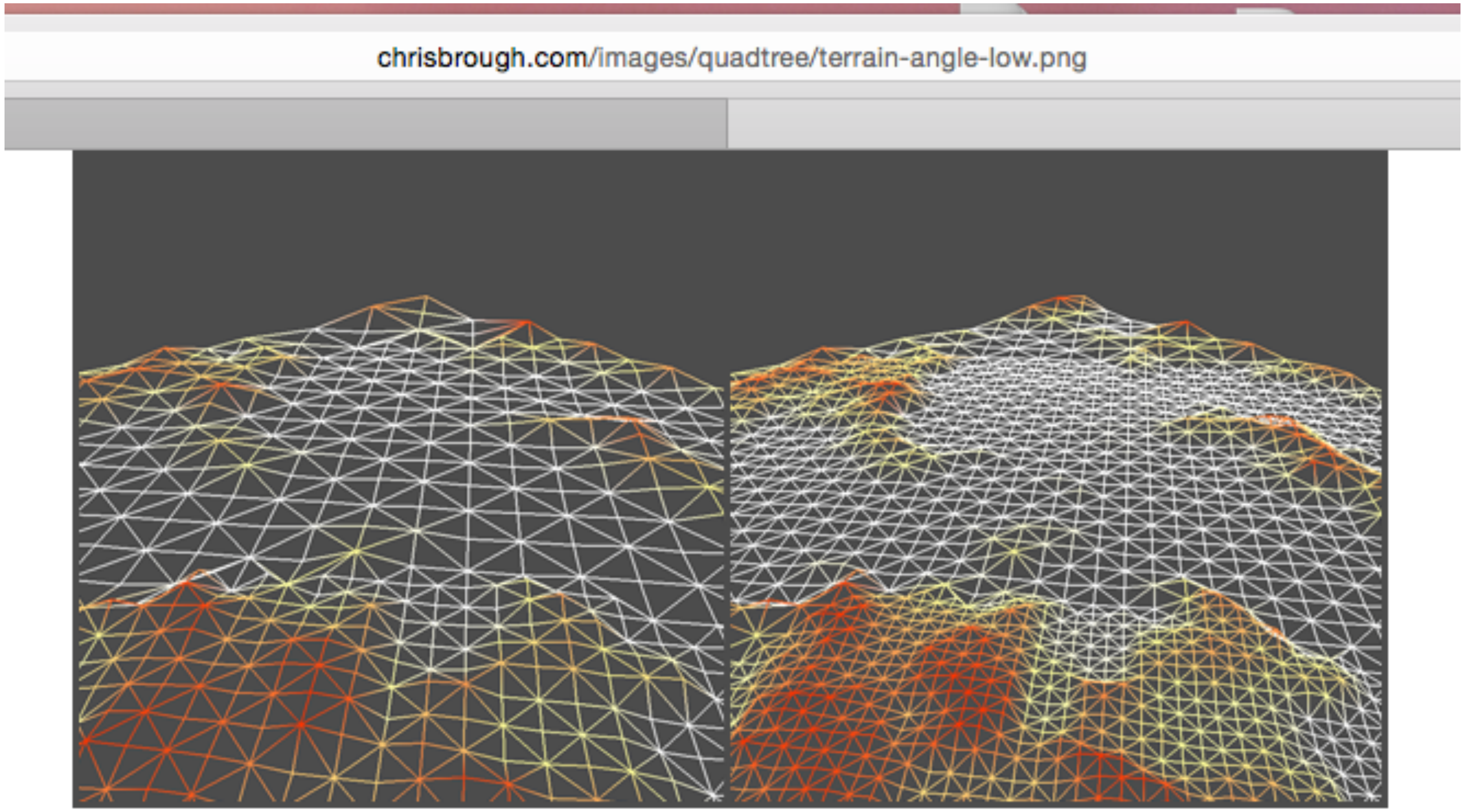
- Can be built for many types of data
  - points, edges, polygons, images, etc
- Can be used for many different tasks
  - search, point location, neighbors, joins, unions, etc
  - dynamic
- Theoretical bounds not great, but widely used in practice
- LOTS of applications
  - Many variants of quadtrees have been proposed
  - Hundreds of papers

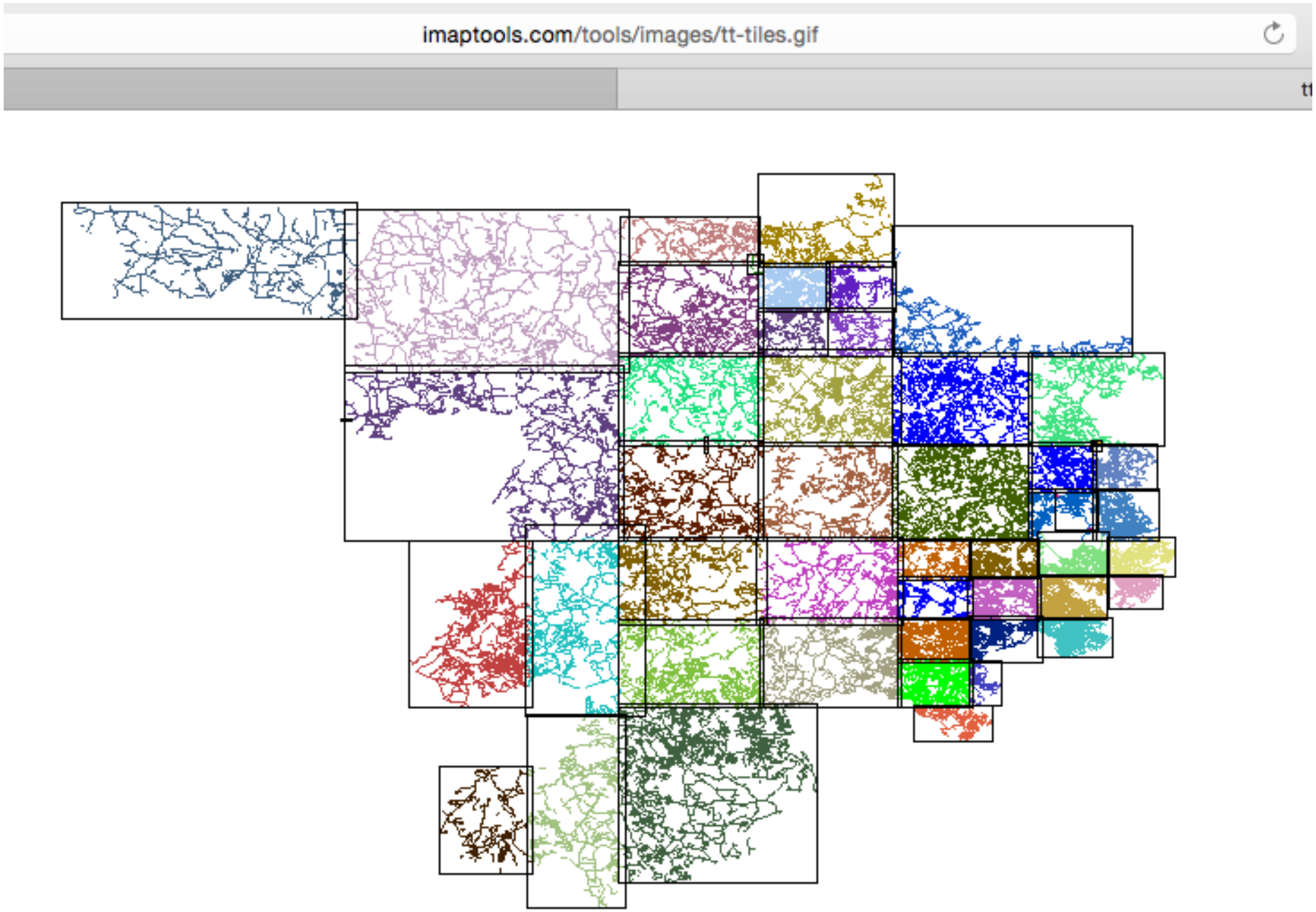
www.jotschi.de/images/quadtree/quadtree.png



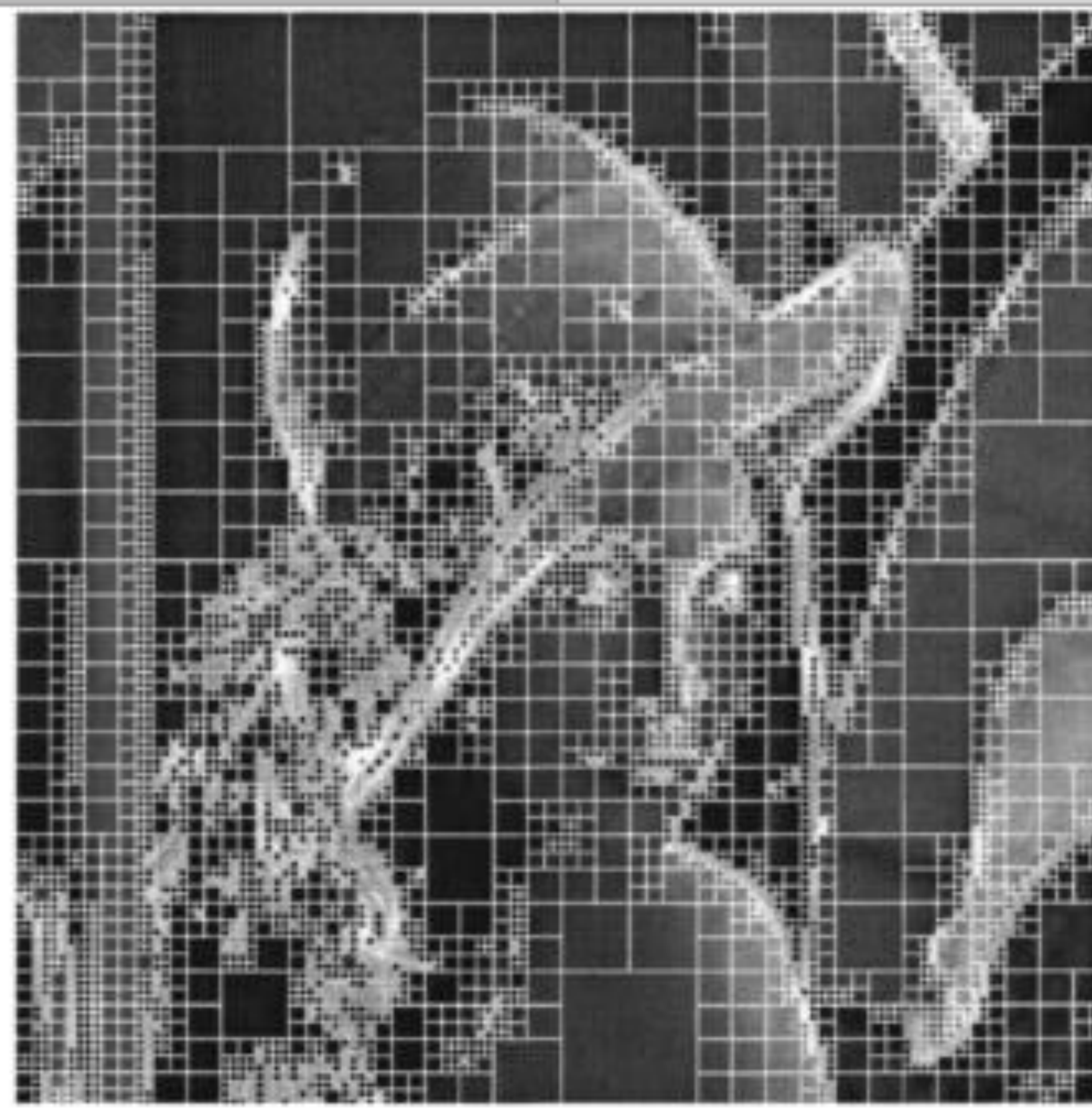

www.pling.org.uk/cs/cgvimg/quadtrees.png
es
quadtrees.

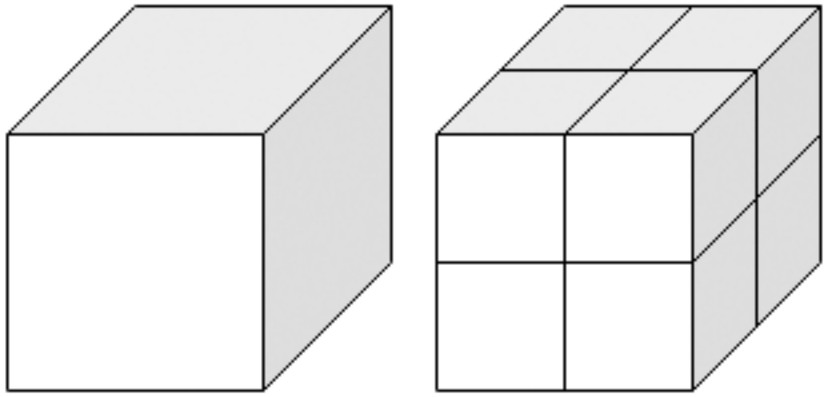

3.bp.blogspot.com/-7m6WQacRMEE/TW82n70i-VI/AAAAAAAAAH8/oOCuQOL_AH4/s400/Screen%2Bshot%2B2011-03-03
Screen+shot+20

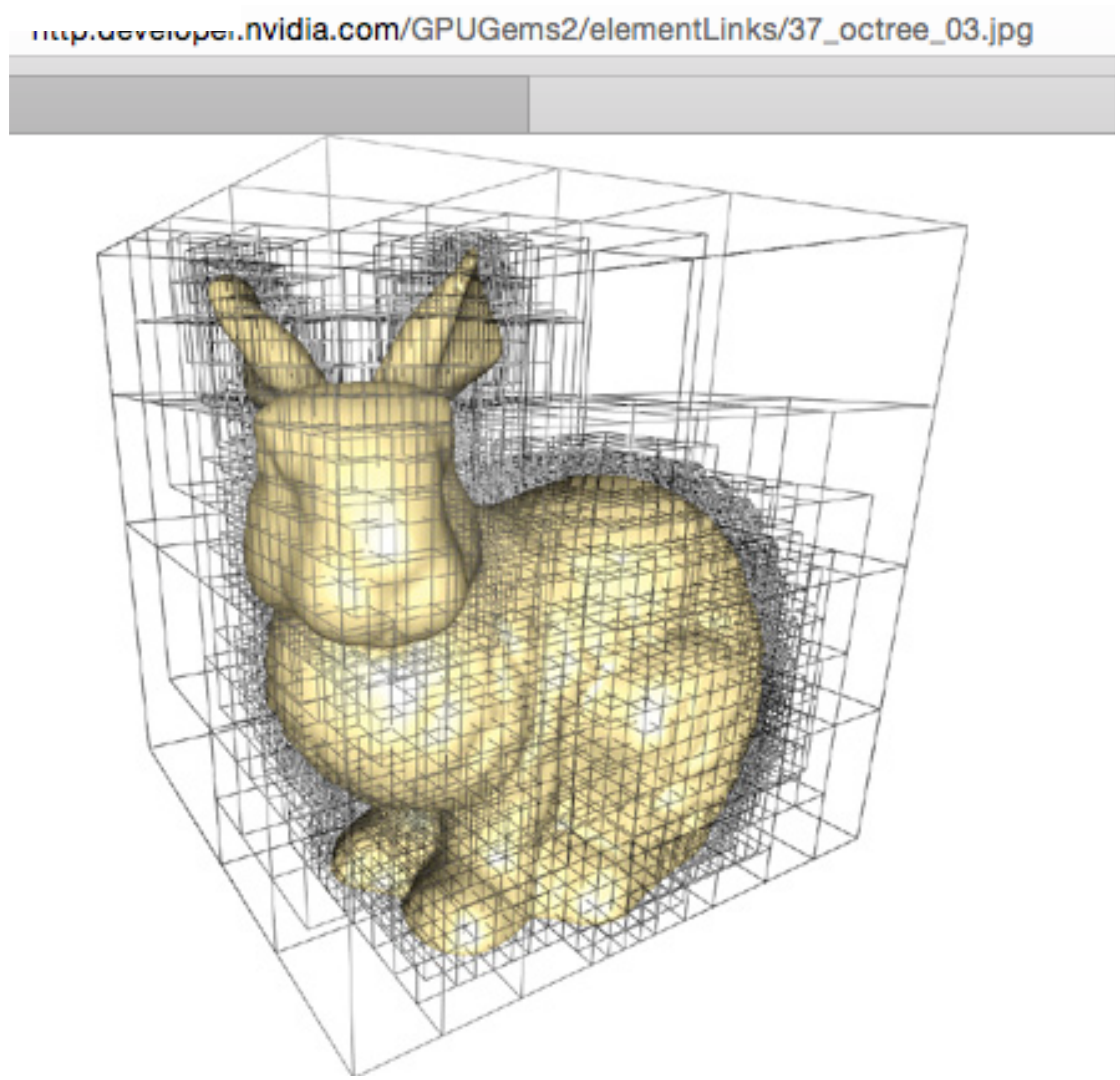

chrisbrough.com/images/quadtree/terrain-angle-low.png


imaptools.com/tools/images/tt-tiles.gif

electronicimaging.spiedigitallibrary.org/data/Journals/ELECTIM/22287/501504jei2.jpeg



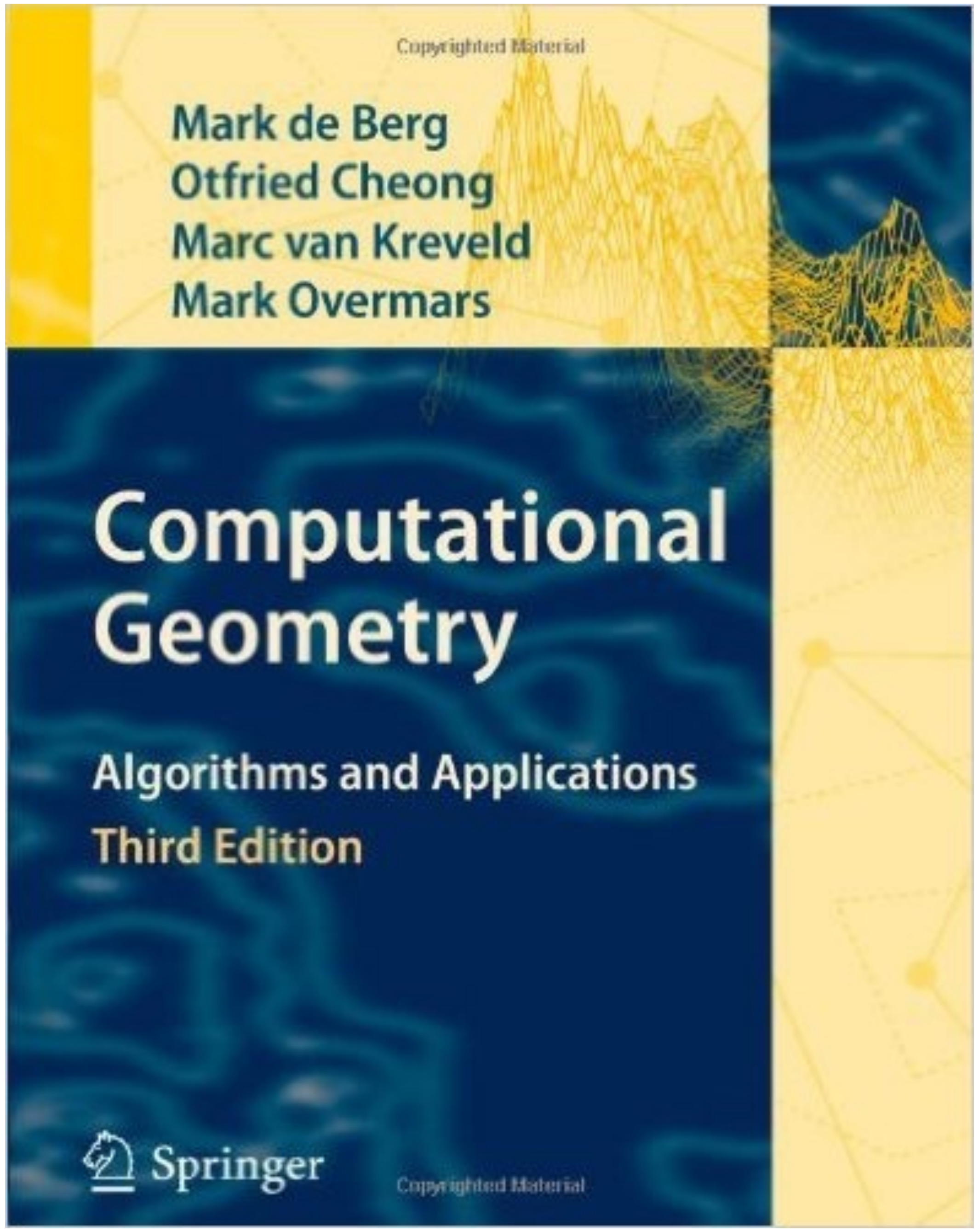



nvidia.com/GPUGems2/elementLinks/37_octree_03.jpg

Outline

- Point quadtrees

  Illustrate the core properties of quadtrees



- Extensions and applications

Point-quadtree

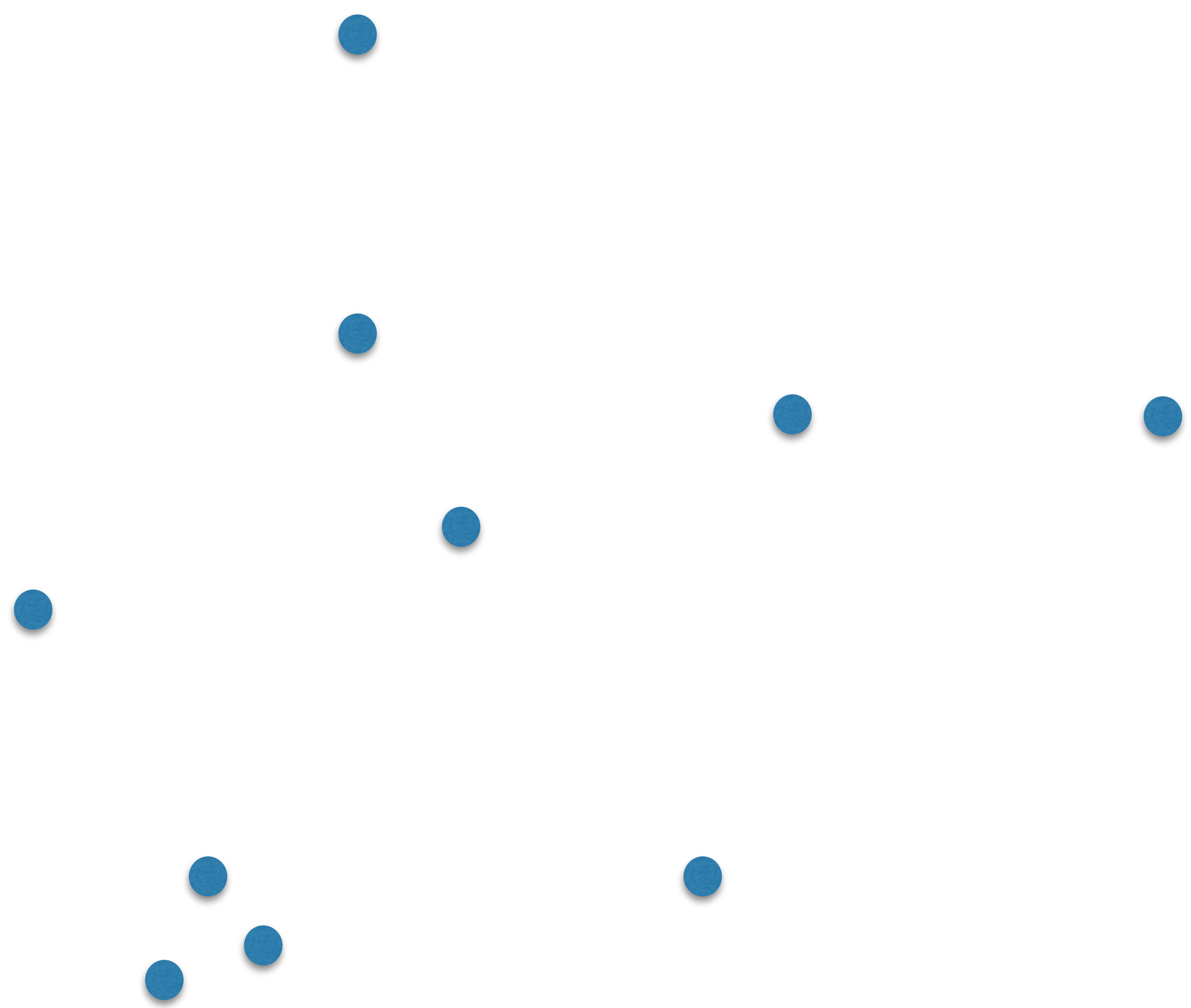
Let P = set of n points in the plane

Problem: Store P in a quadtree such that every square has <= 1 point.

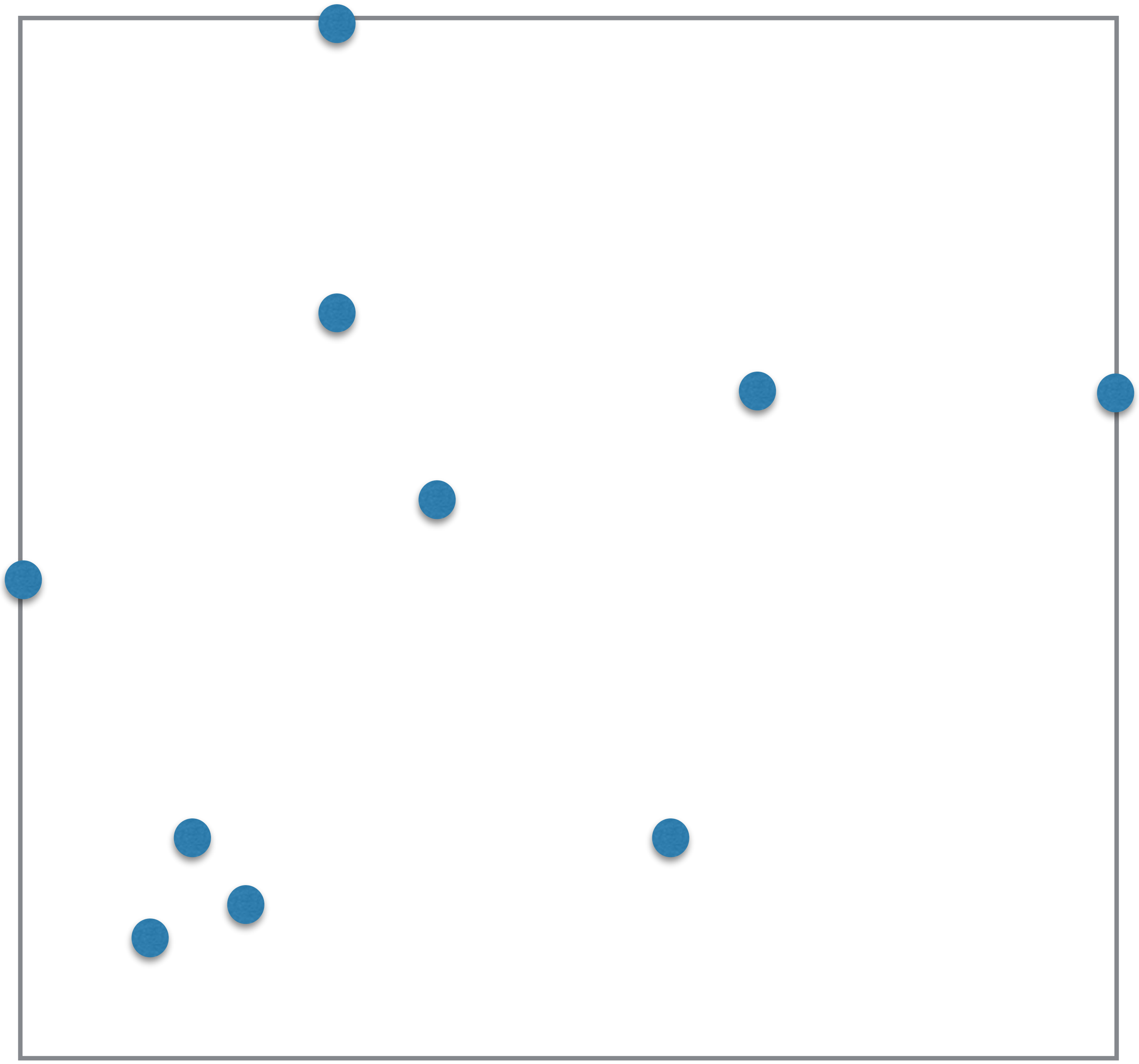
Questions:

1. Size? Height?
2. How to build it and how fast?
3. What can we do with it?

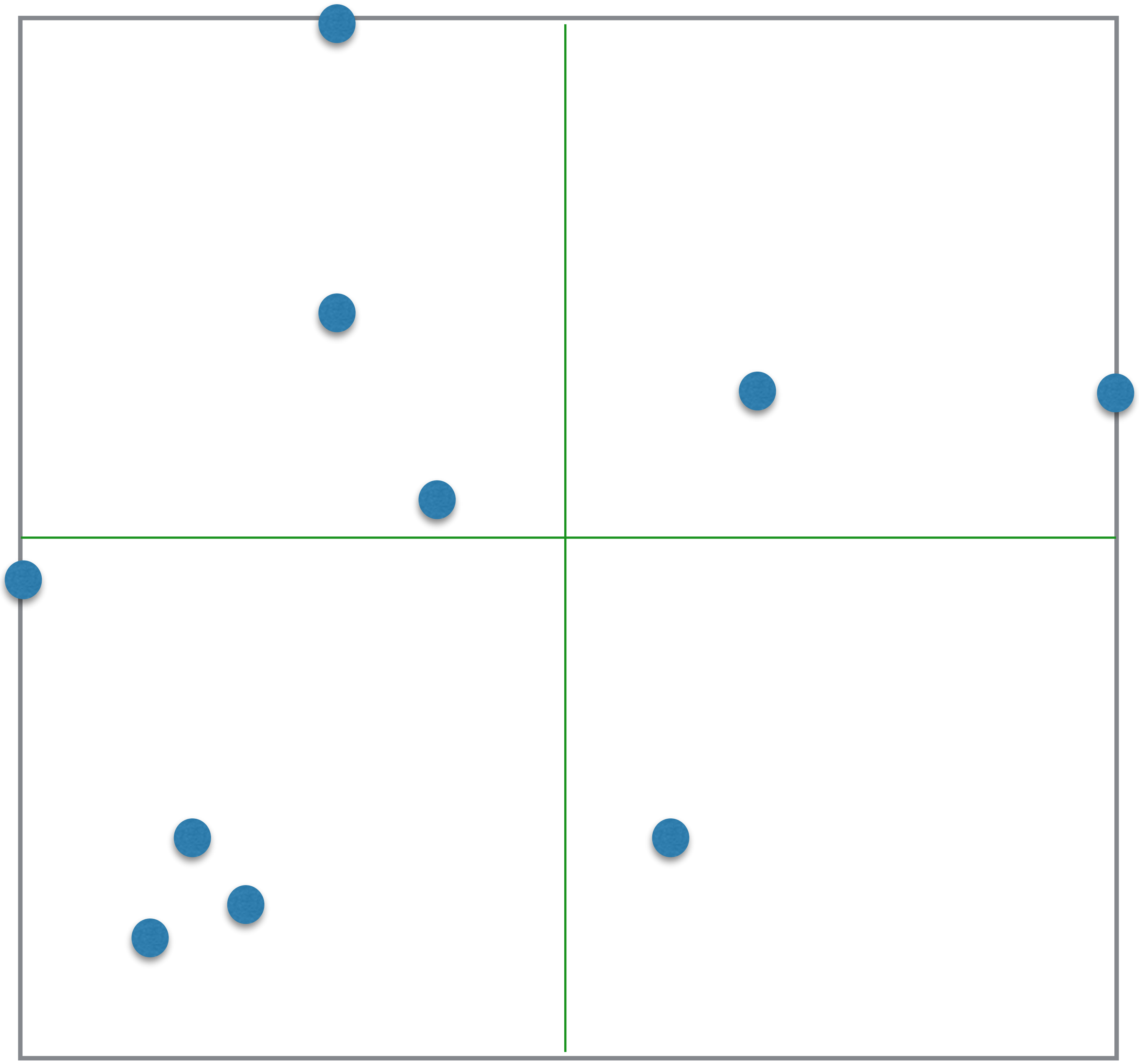
Let P = set of n points in the plane



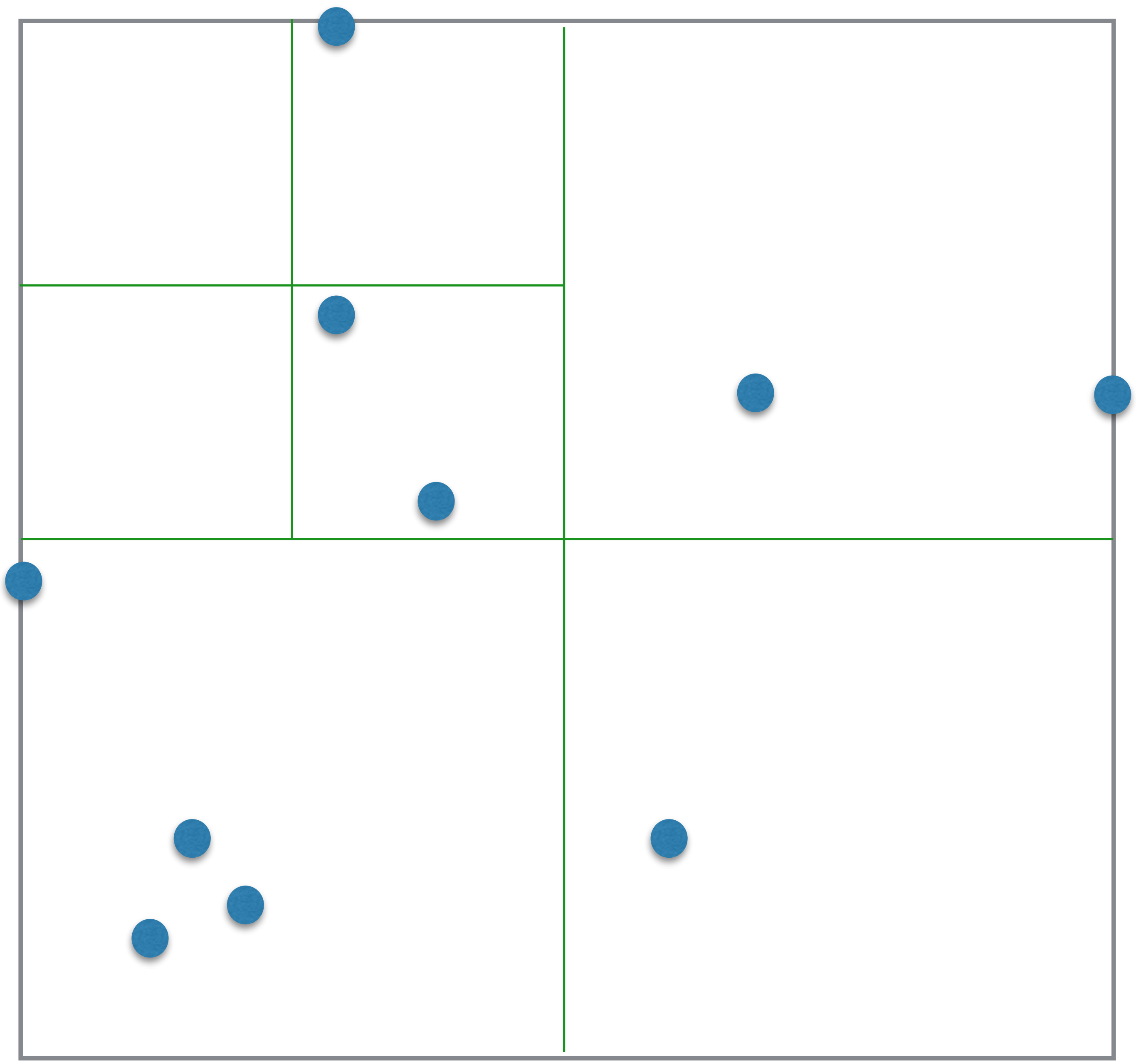
Let P = set of n points in the plane



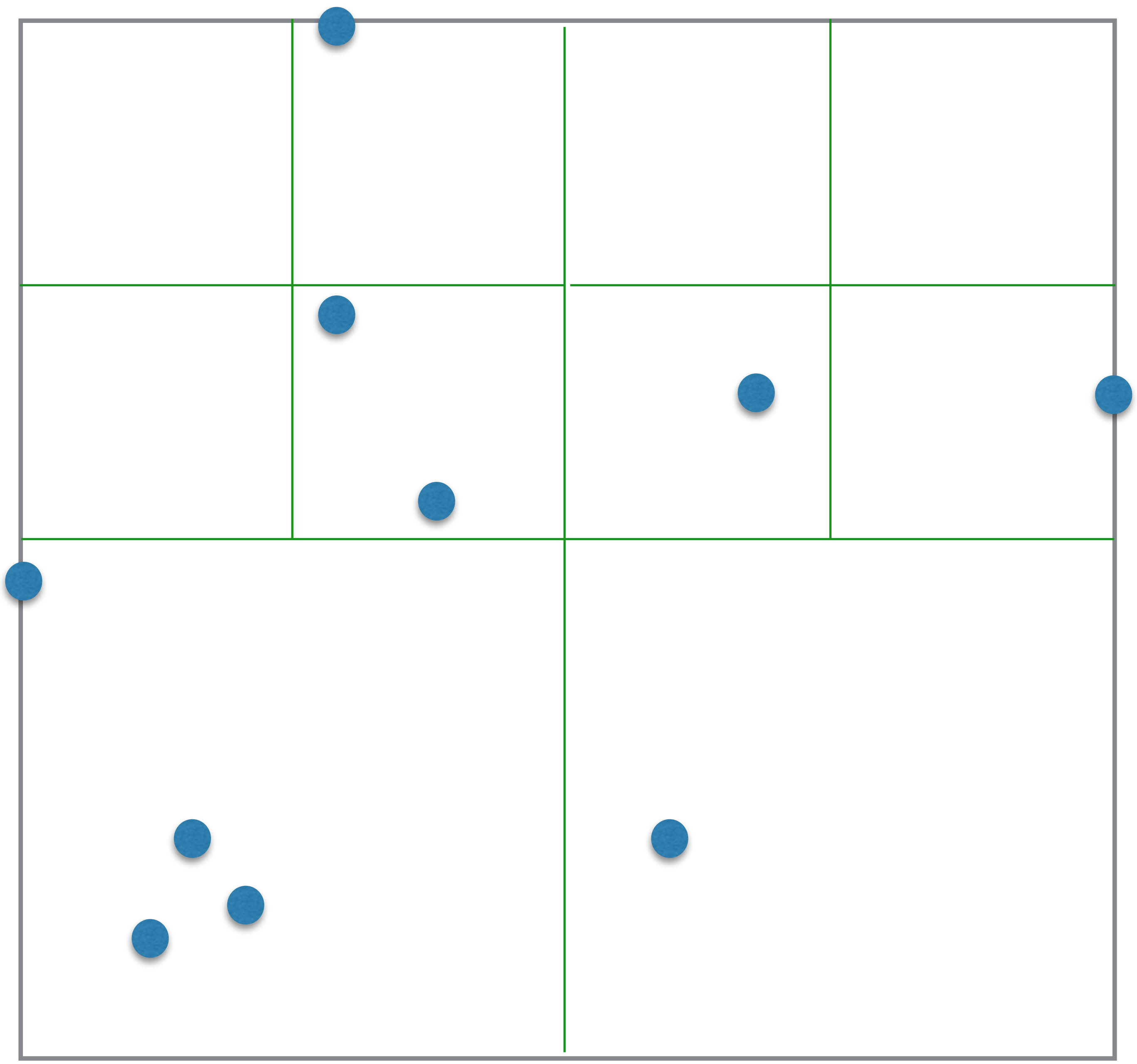
Let P = set of n points in the plane



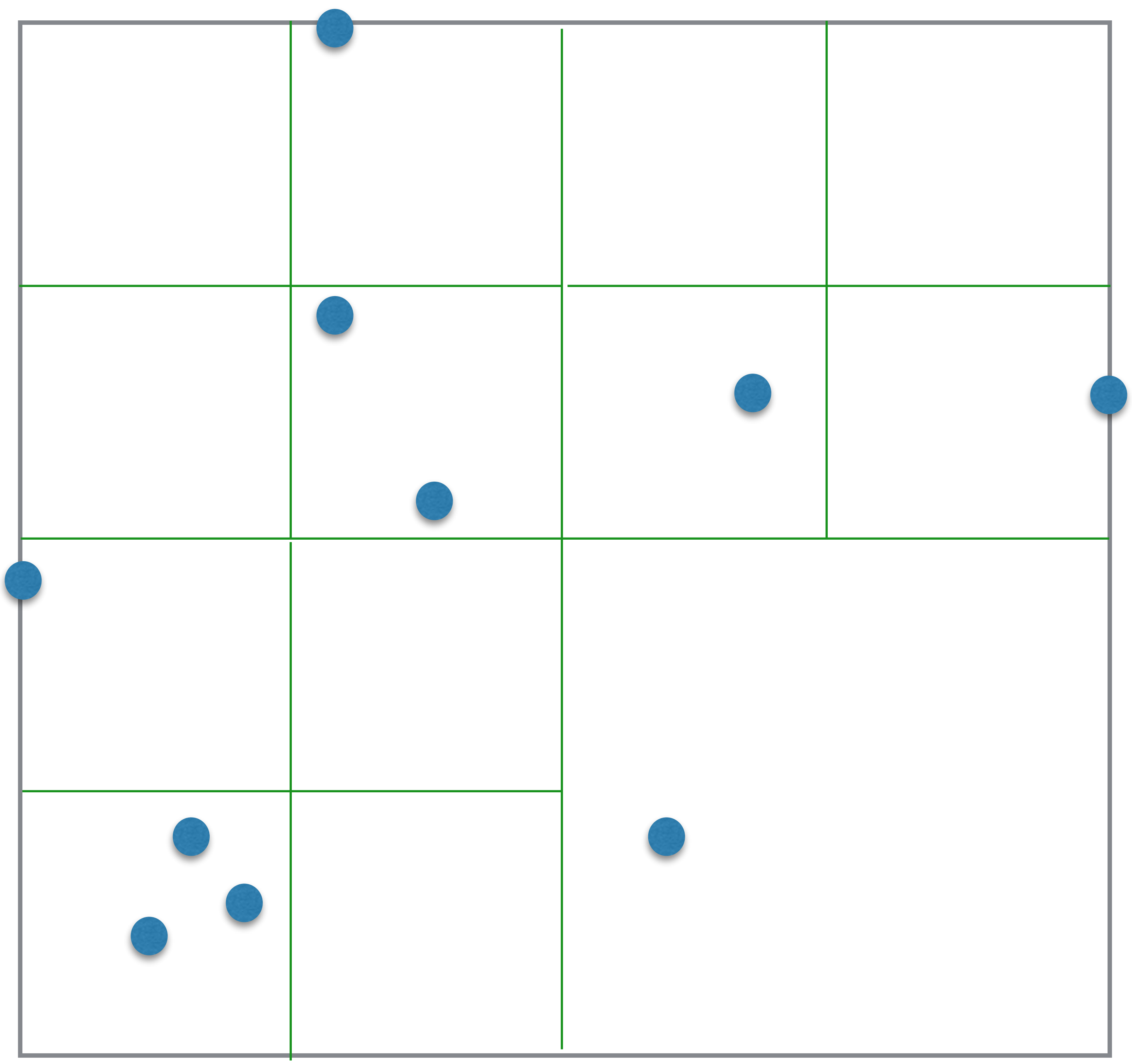
Let P = set of n points in the plane



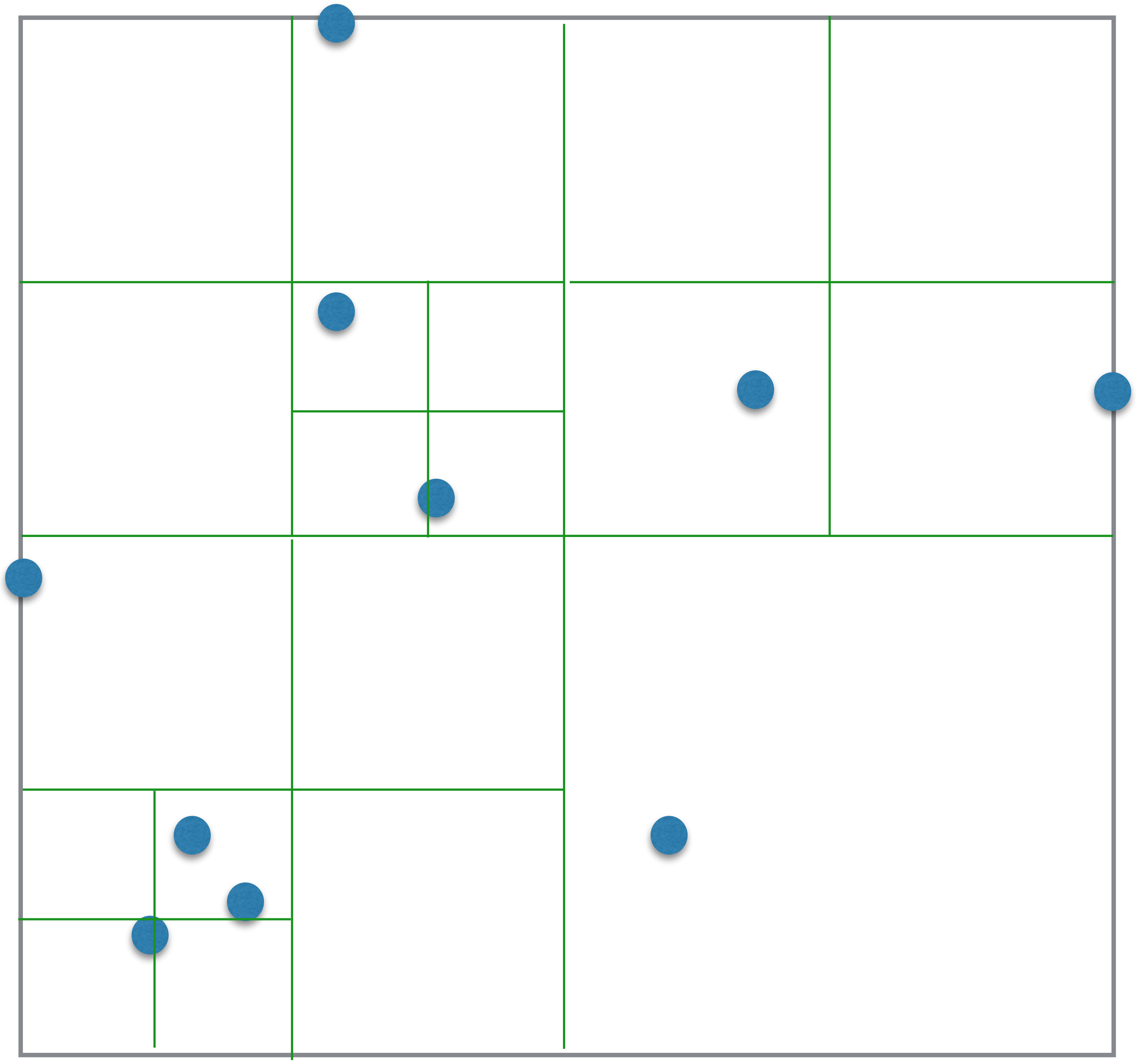
Let P = set of n points in the plane



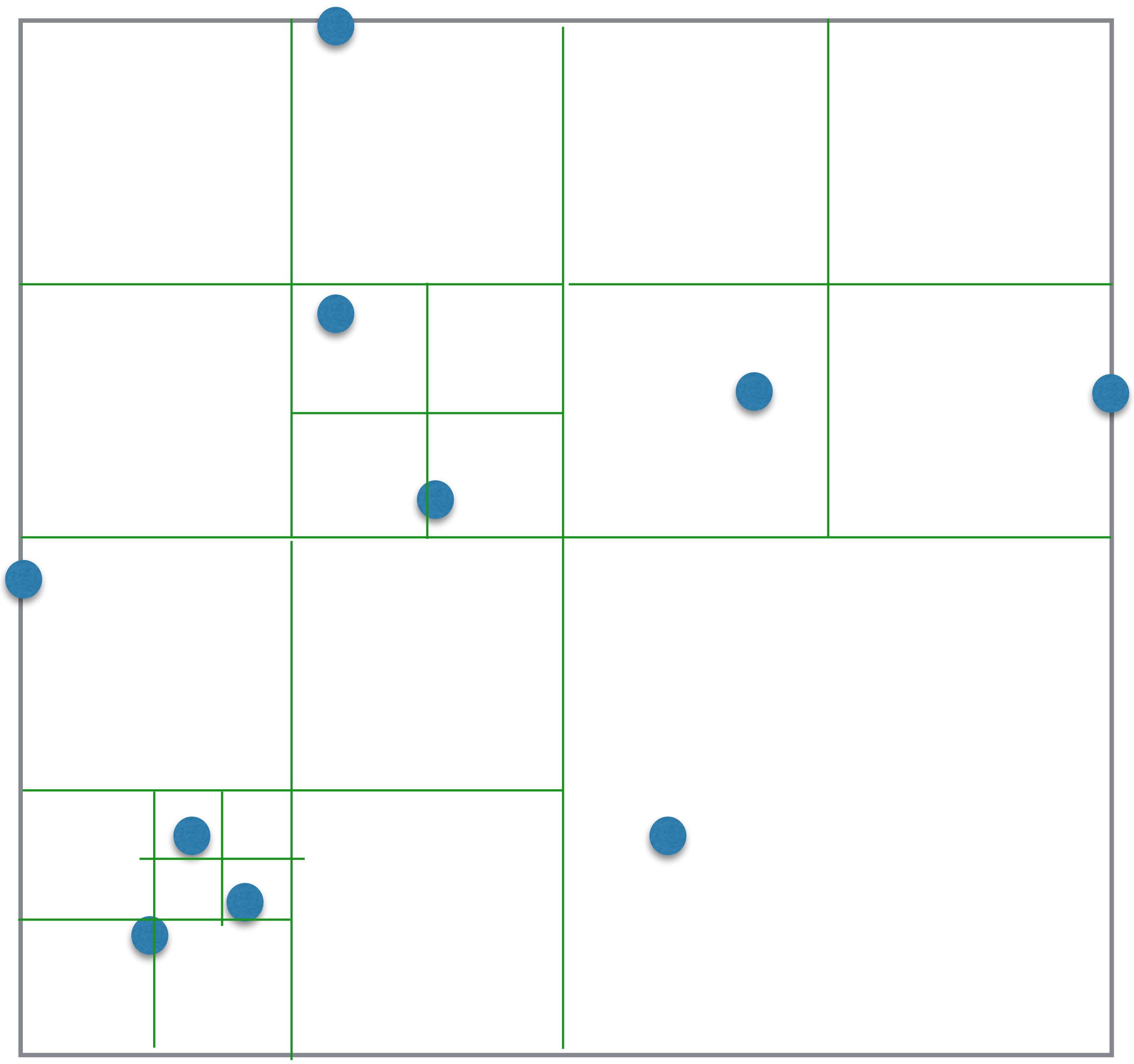
Let P = set of n points in the plane



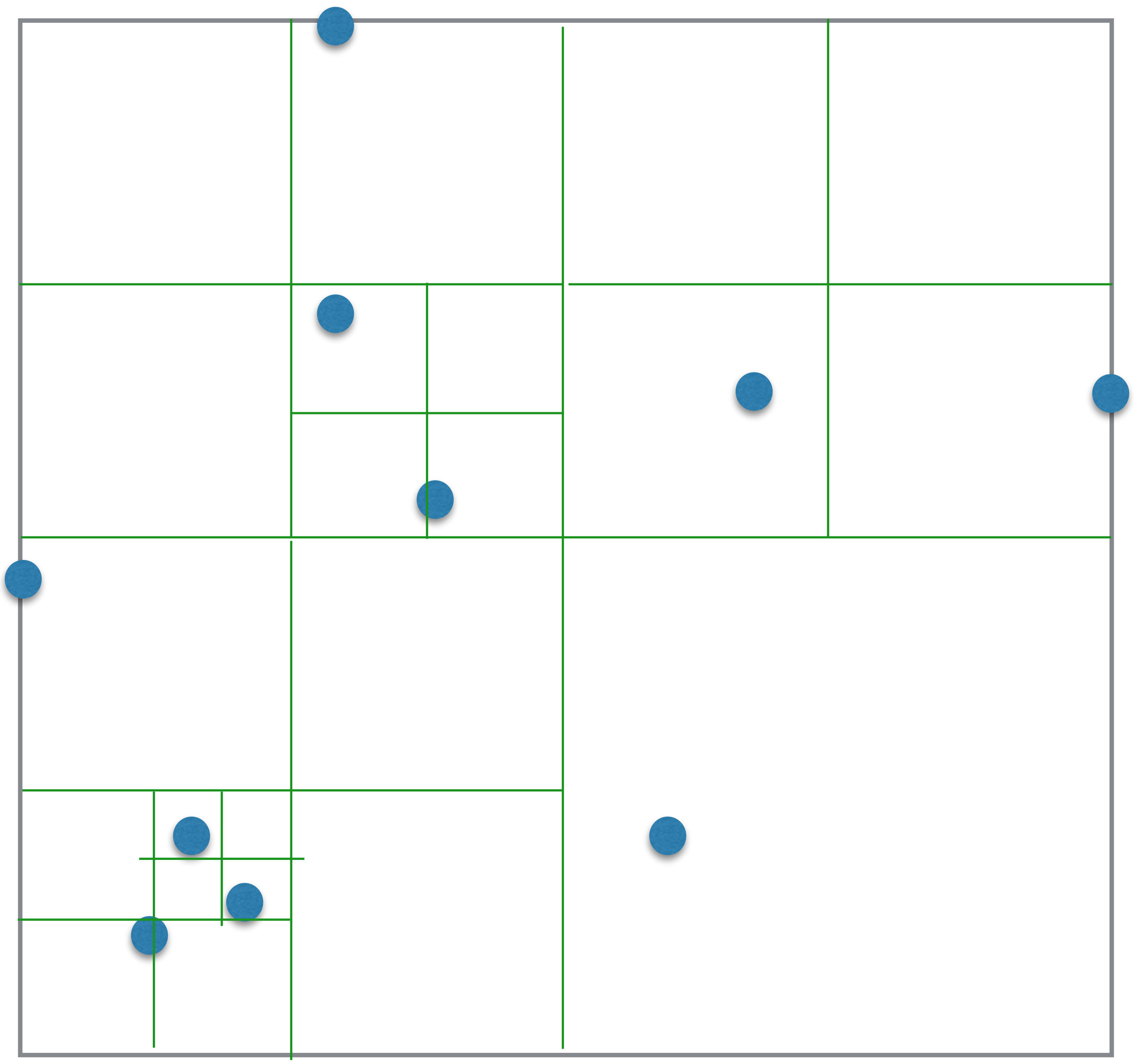
Let P = set of n points in the plane



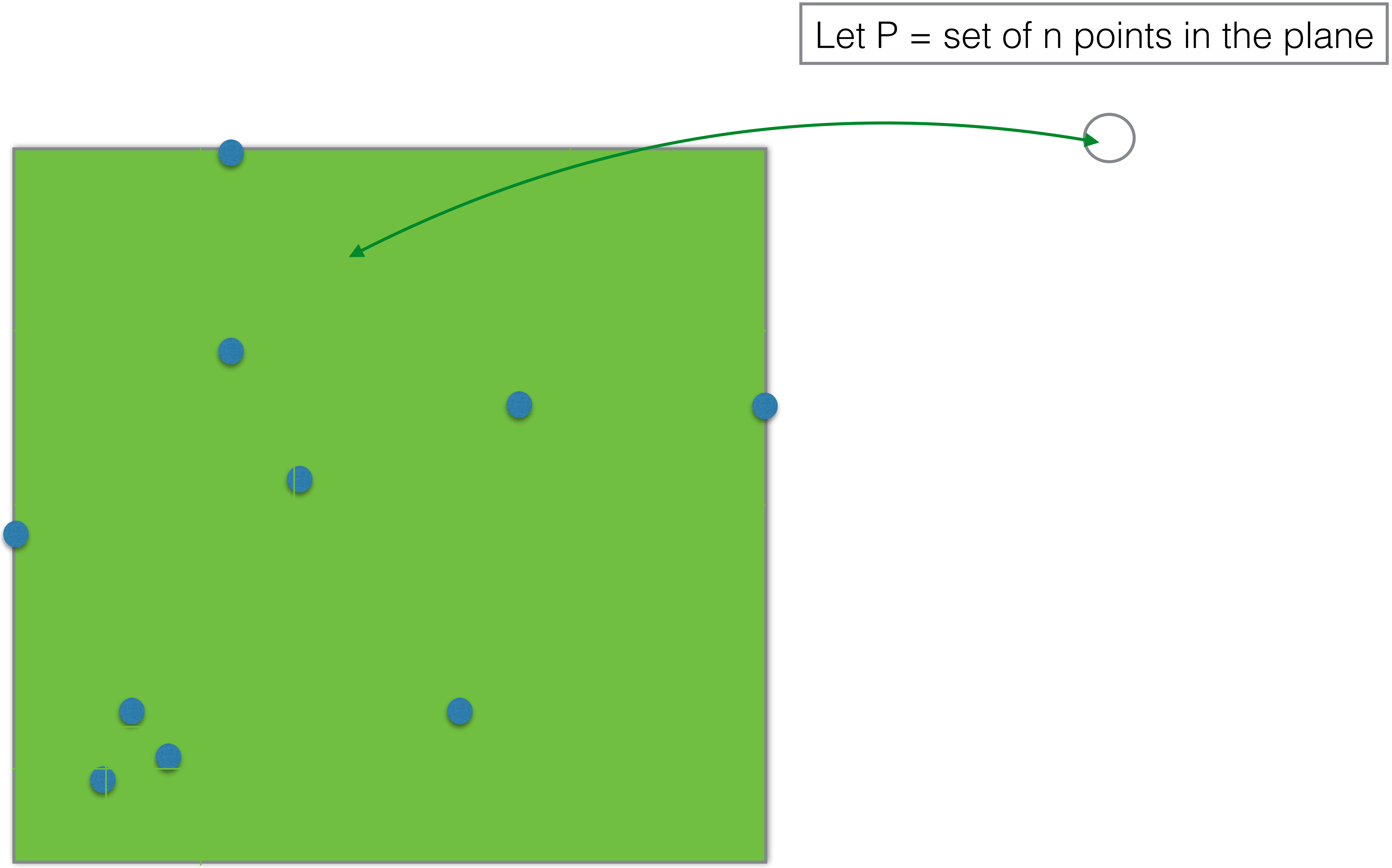
Let P = set of n points in the plane



Let P = set of n points in the plane

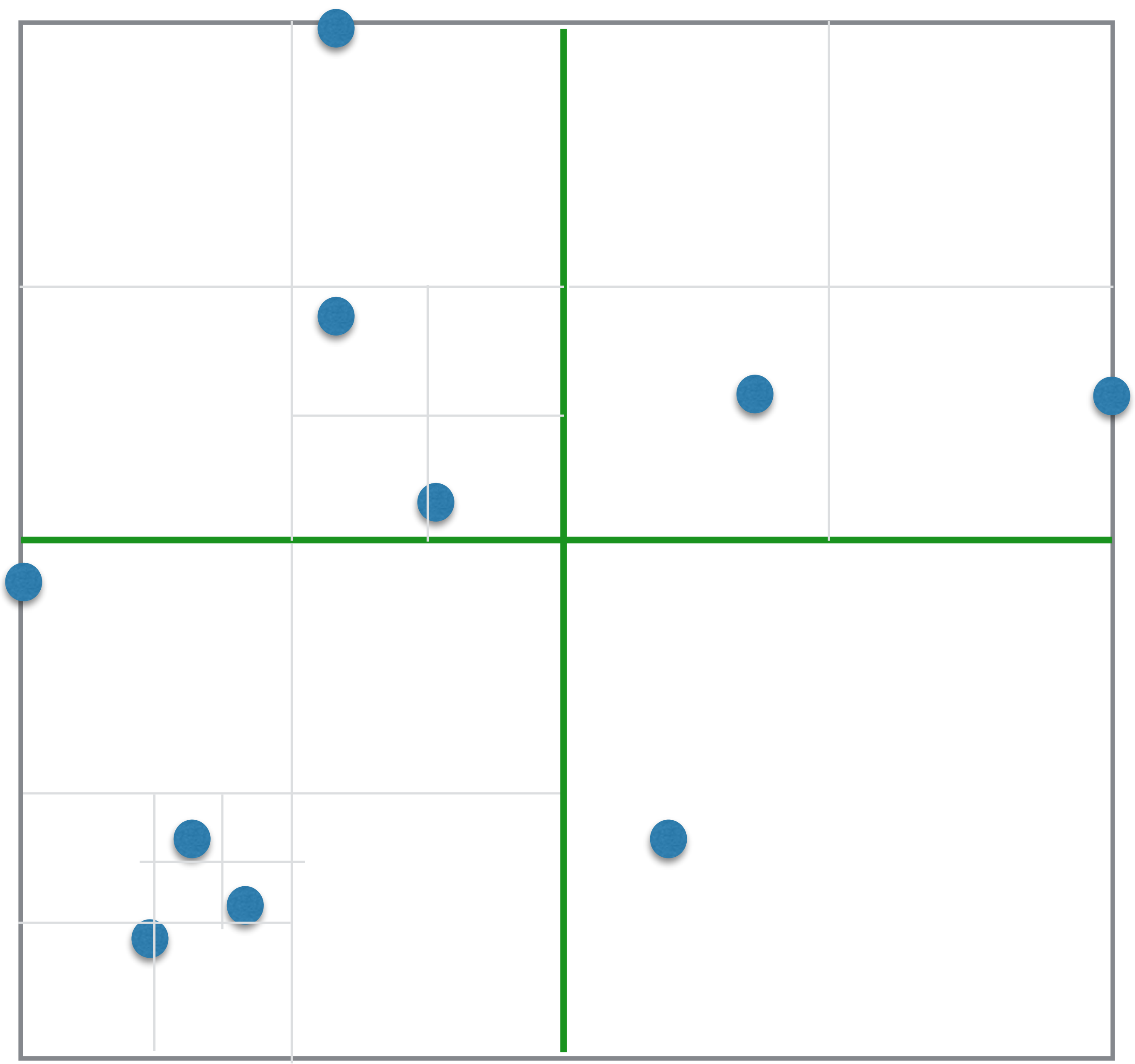


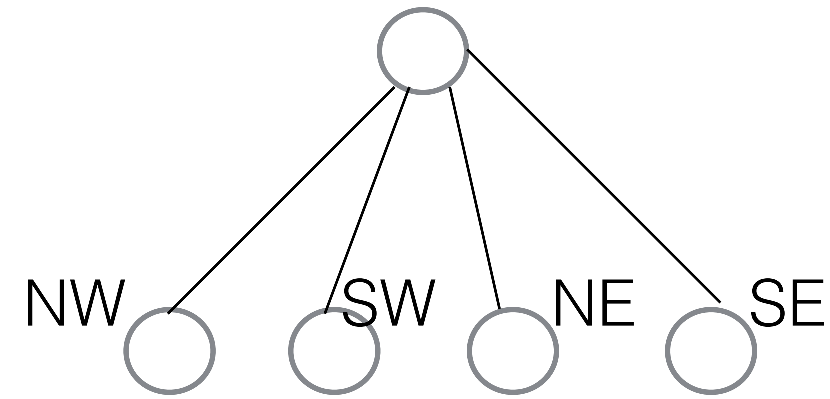
Quadtree: tree corresponding to the subdivision



Quadtree: tree corresponding to the subdivision

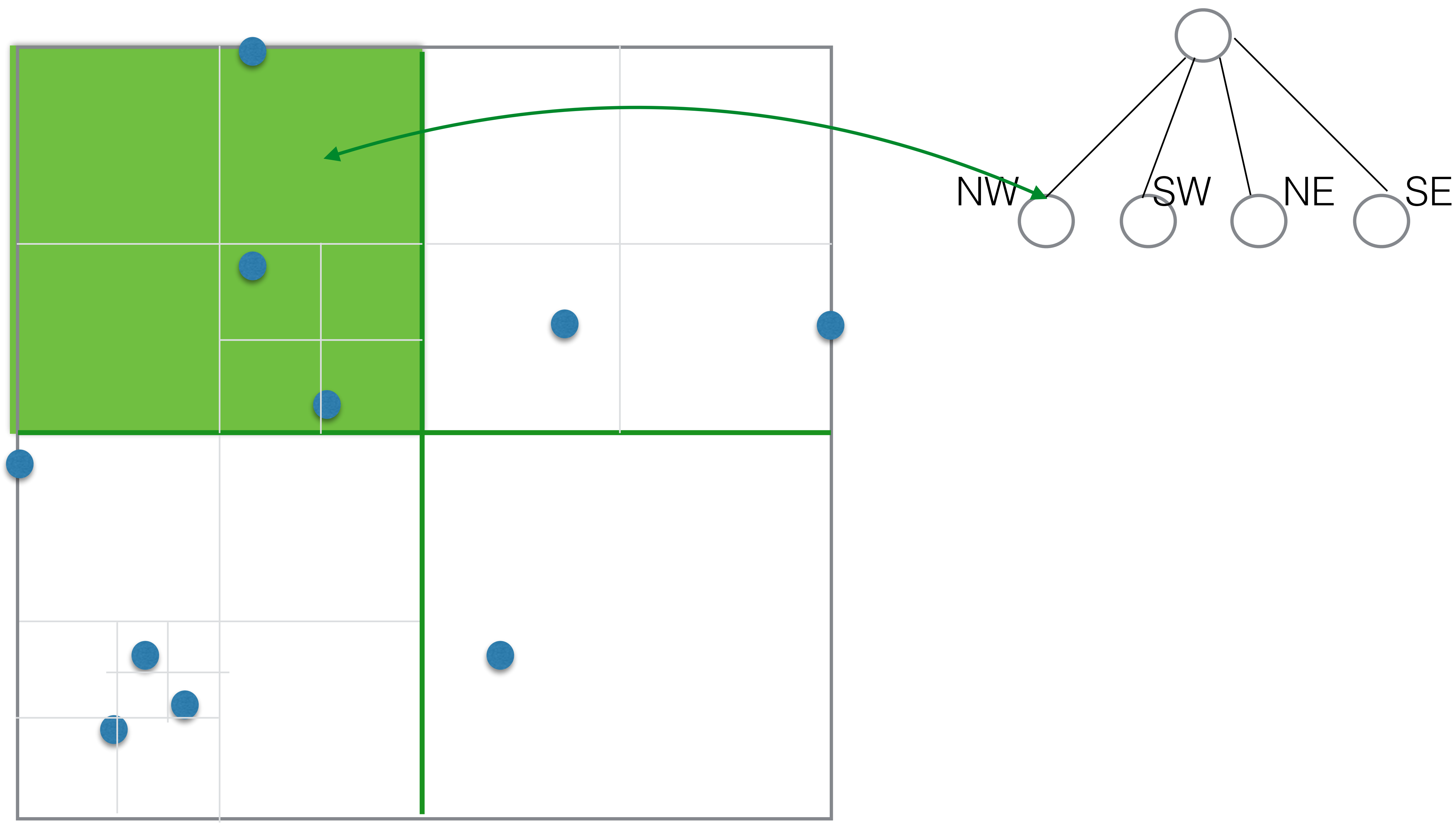
Let P = set of n points in the plane





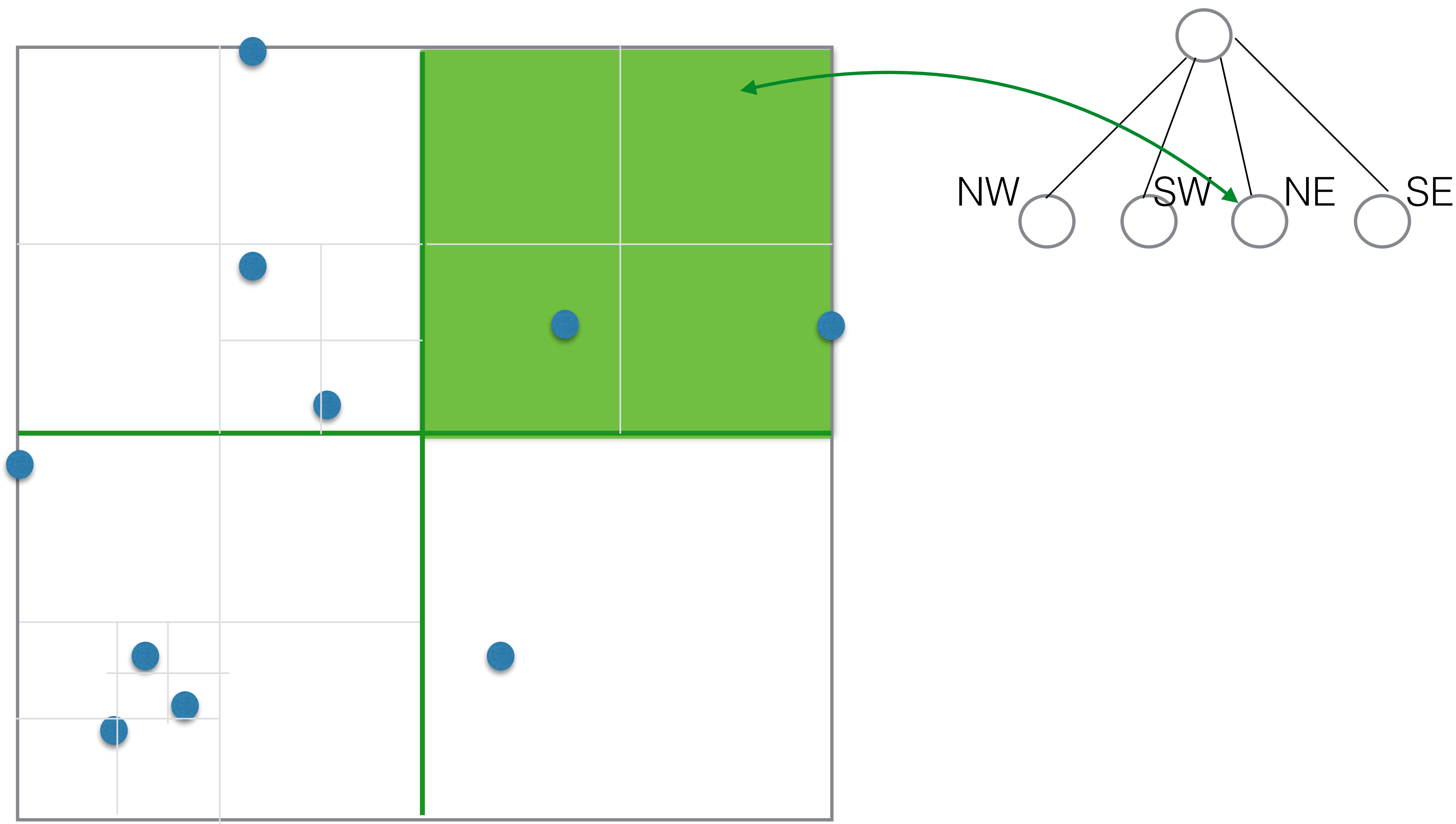
Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



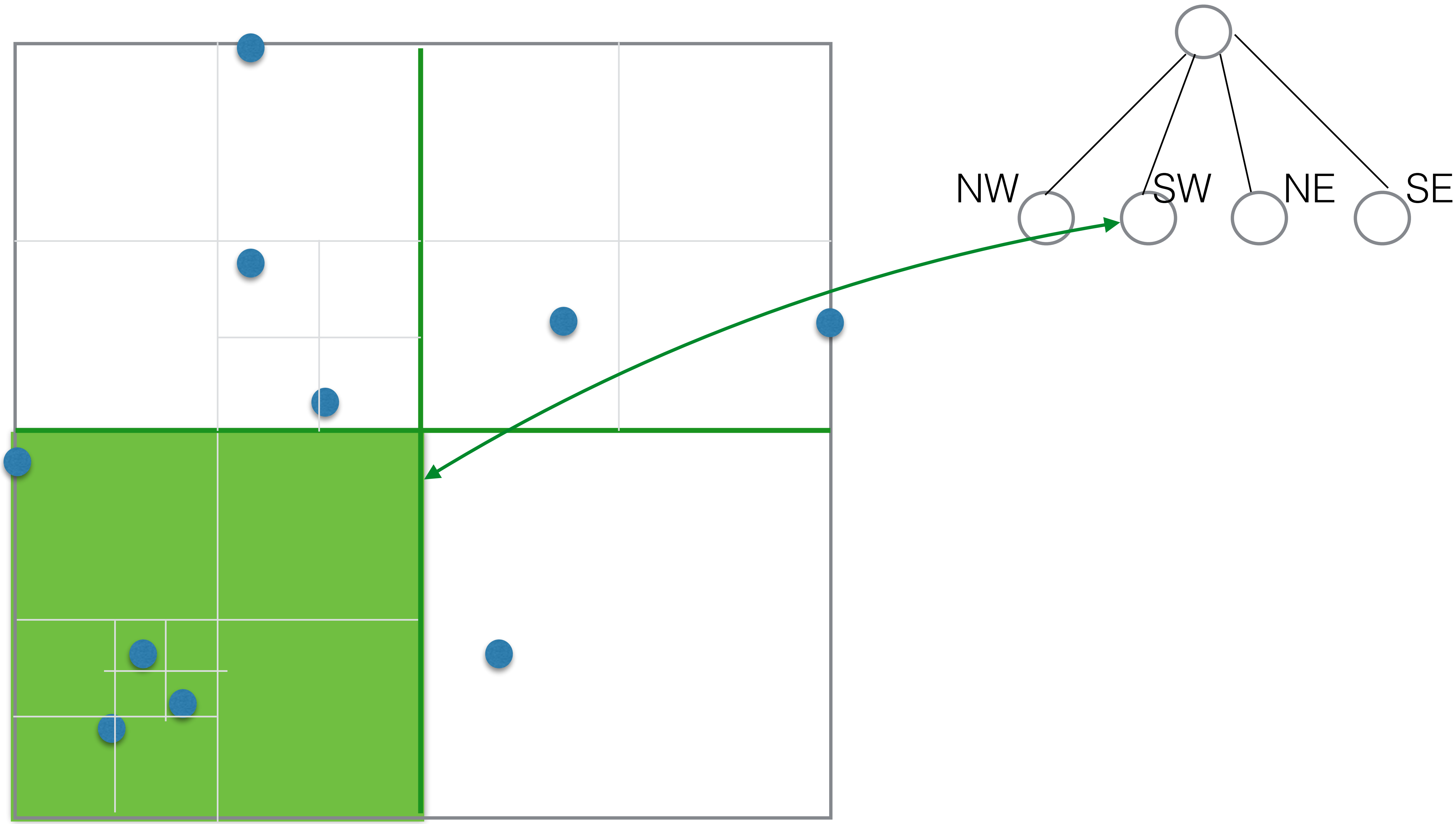
Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



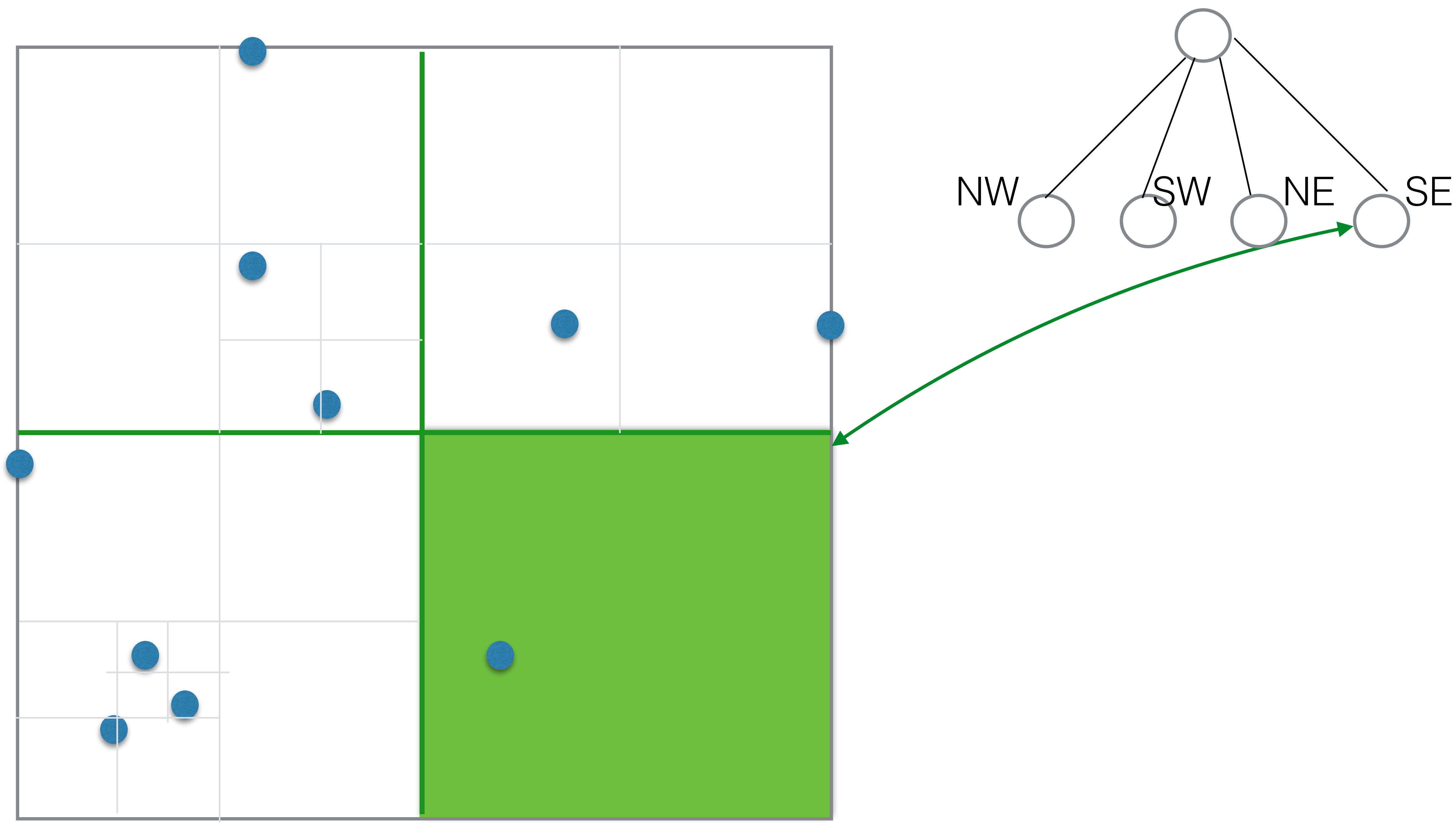
Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



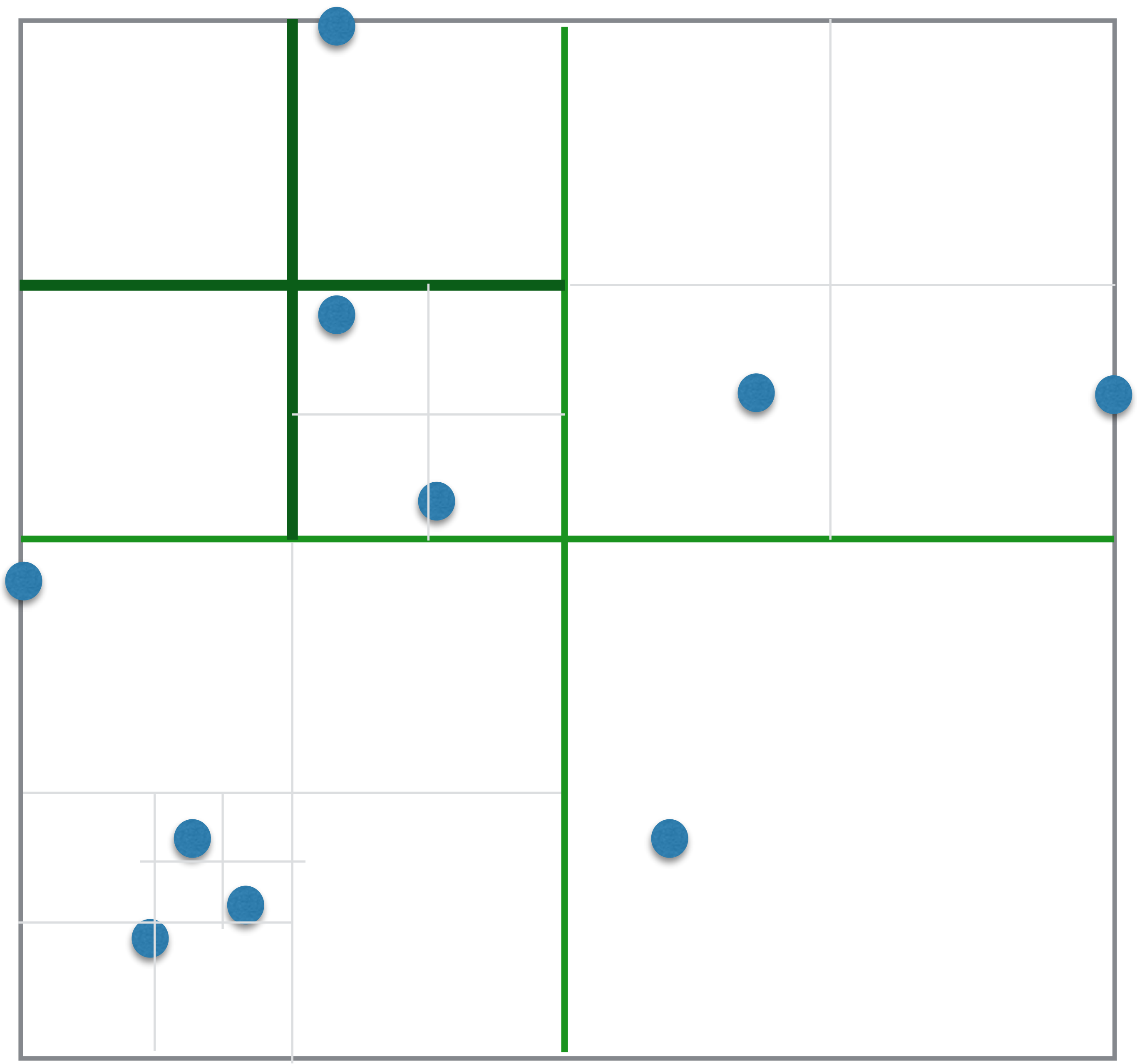
Quadtree: tree corresponding to the subdivision

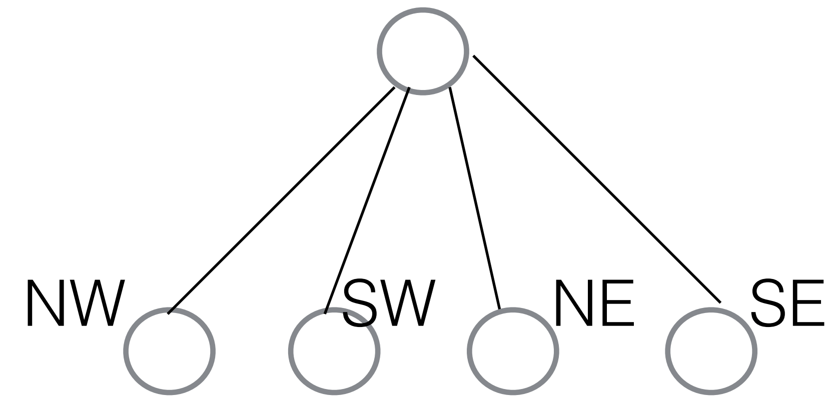
Let P = set of n points in the plane



Quadtree: tree corresponding to the subdivision

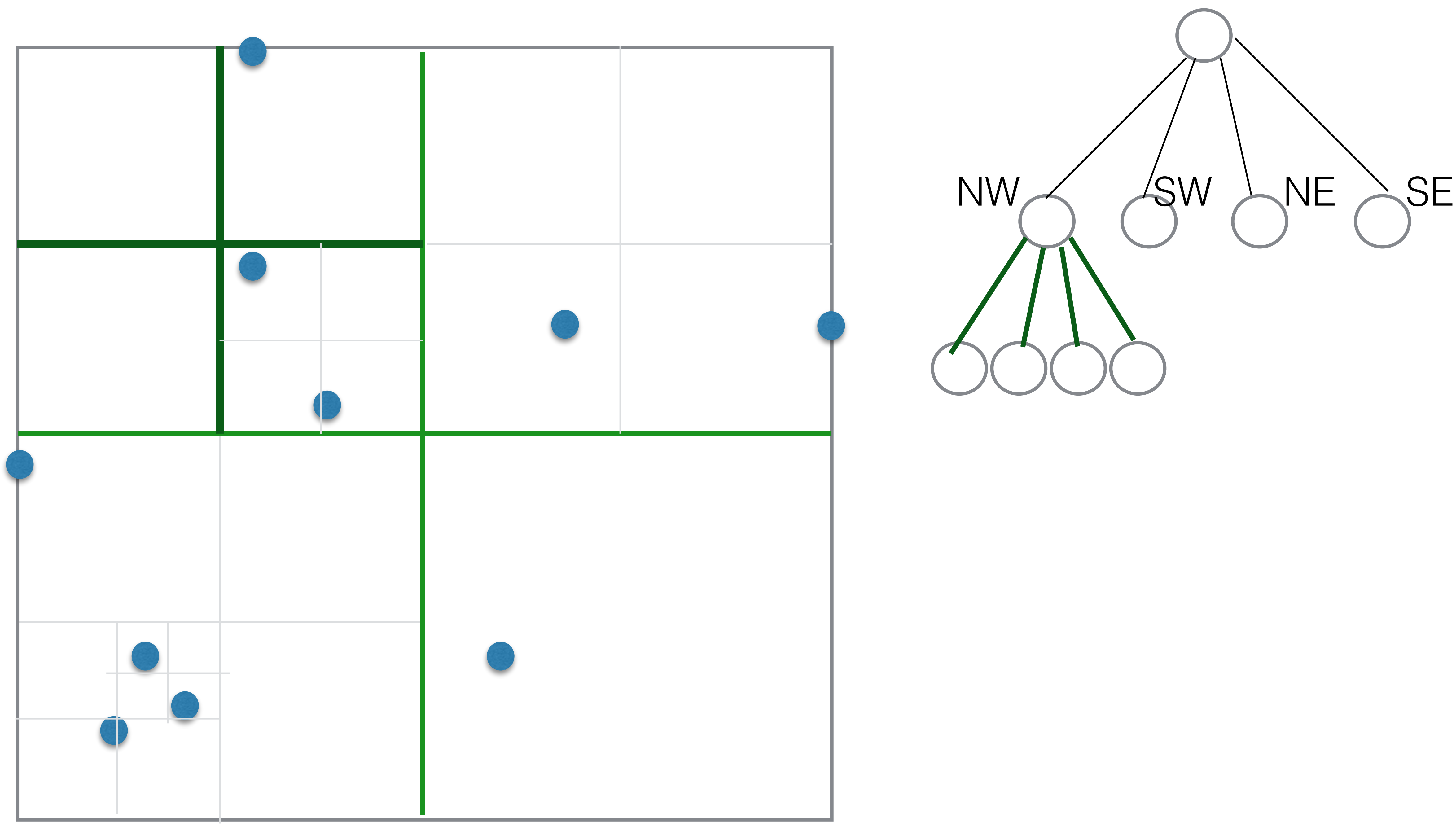
Let P = set of n points in the plane





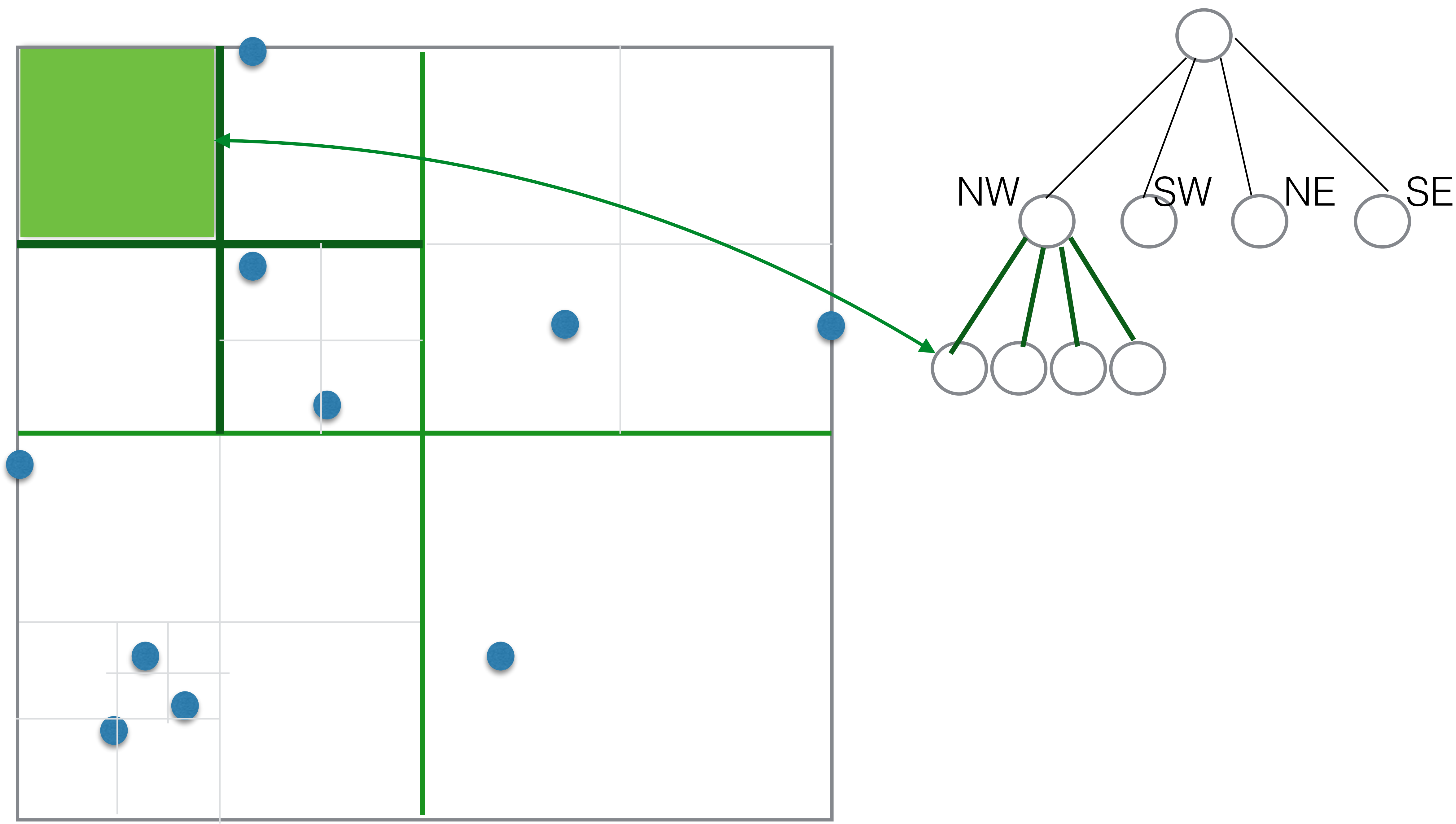
Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



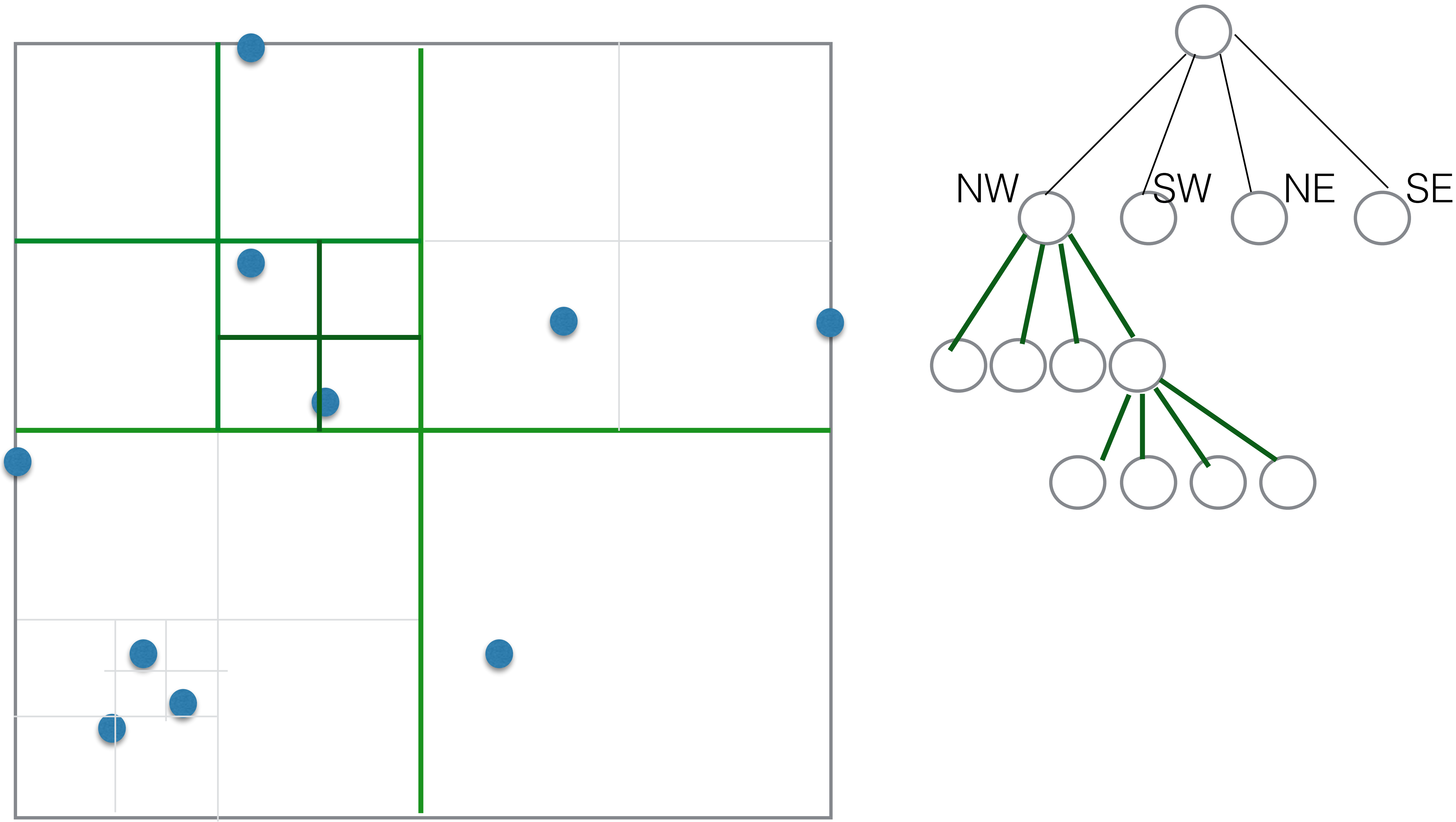
Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



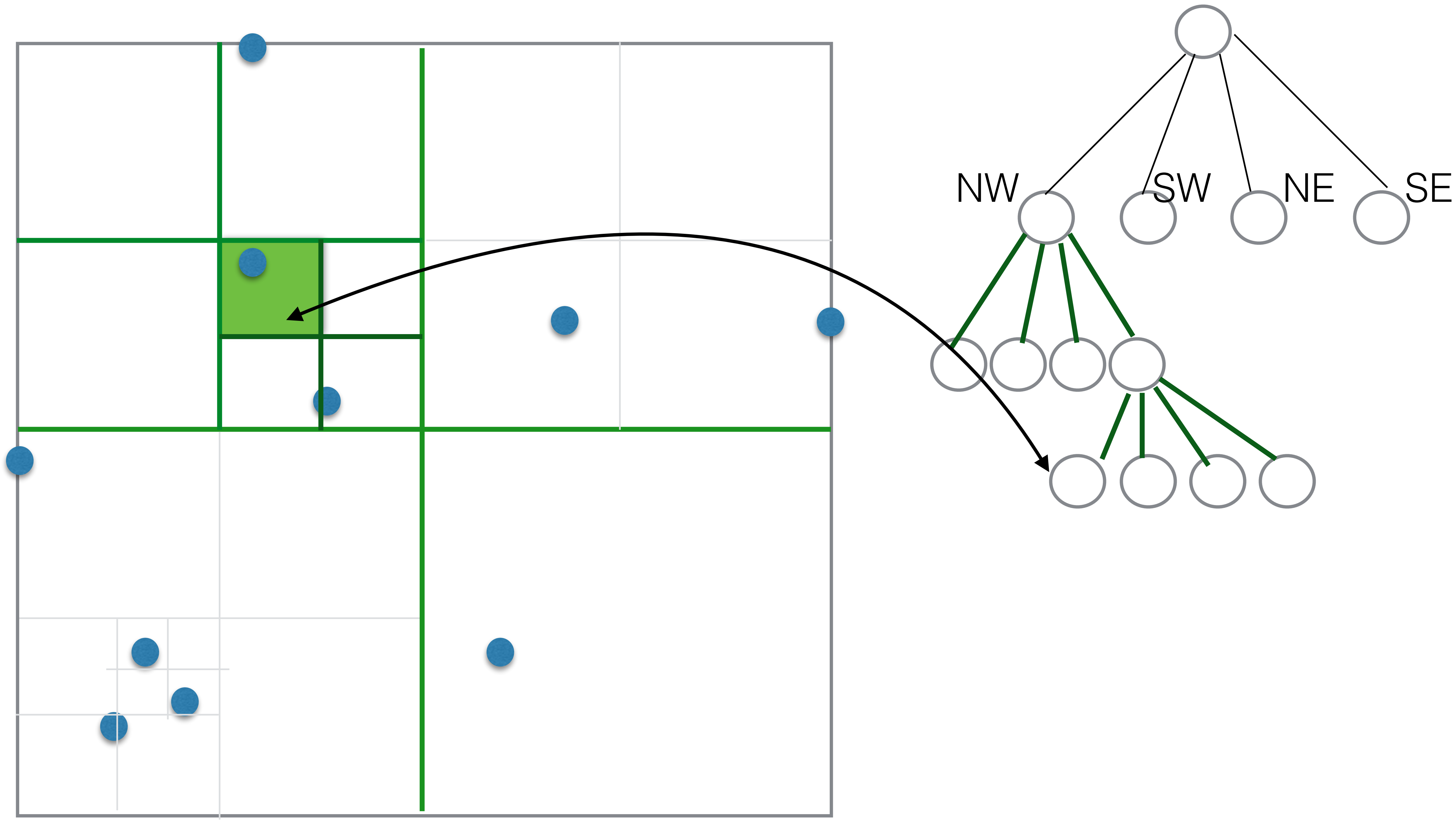
Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



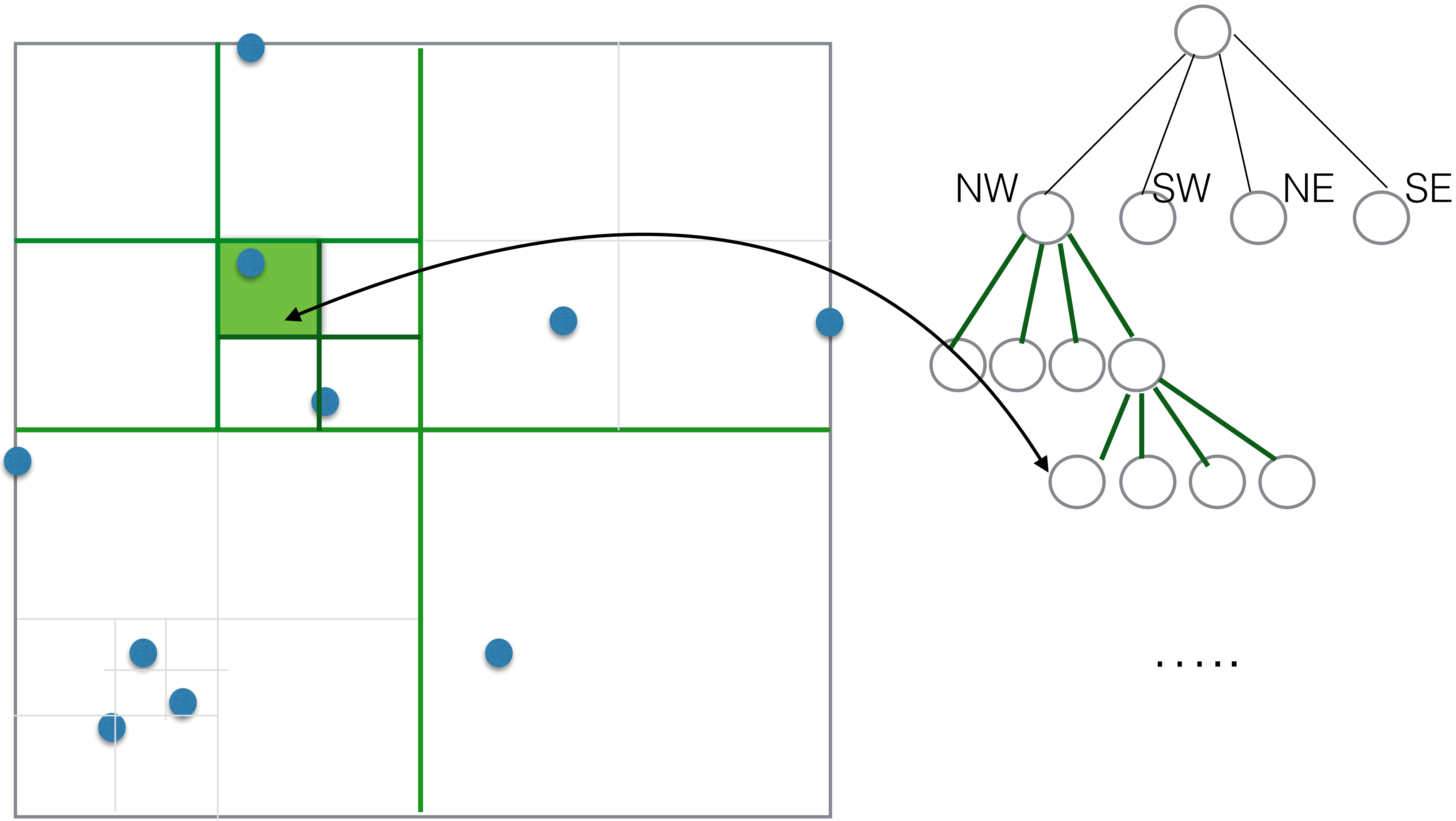
Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



Quadtree: tree corresponding to the subdivision

Let P = set of n points in the plane



Quadtree: tree corresponding to the subdivision

Exercises

Let P = set of n points in the plane

- Pick n=10 points in the plane and draw their quadtree.
- Show a set of (10) points that have a balanced quadtree.
- Show a set of (10) points that have an unbalanced quadtree.
- Draw the quadtree corresponding to a regular grid
  - how many nodes does it have?
  - how many leaves? height?
- Consider a set of points with a uniform distribution. What can you say about the quadtree ?
- Let's look at sets of 2 points in the plane.
  - Sketch the smallest possible quad tree for two points in the plane.
  - Sketch the largest possible quad tree for two points in the plane.
  - An upper bound for the height of a quadtree for 2 points ????

Quadtree size

P = set of n points in the plane

Quadtree size

P = set of n points in the plane

Theorem:

Quadtree size

P = set of n points in the plane

Theorem:

The height of a quadtree storing P is at most $\lg (s/d) + 3/2$, where s is the side of the original square and d is the distance between the closest pair of points in P.

Quadtree size

P = set of n points in the plane

Theorem:

The height of a quadtree storing P is at most $\lg(s/d) + 3/2$, where s is the side of the original square and d is the distance between the closest pair of points in P.

Proof:

- Each level divides the side of the quadrant into two. After i levels, the side of the quadrant is $s/2^i$
- A quadrant will be split as long as the two closest points will fit inside it.
- In the worst case the closest points will fit diagonally in a quadrant and the "last" split will happen at depth i such that $s\ \mathrm{sqrt}(2)/2^i = d$…
- The height of the tree is i+1

Quadtree size

P = set of n points in the plane

Theorem:

The height of a quadtree storing P is at most $\lg(s/d) + 3/2$, where s is the side of the original square and d is the distance between the closest pair of points in P.

Proof:

- Each level divides the side of the quadrant into two. After i levels, the side of the quadrant is $s/2^i$
- A quadrant will be split as long as the two closest points will fit inside it.
- In the worst case the closest points will fit diagonally in a quadrant and the "last" split will happen at depth i such that $s\,\mathrm{sqrt}(2)/2^i = d$...
- The height of the tree is i+1

- What does this mean?
  - The distance between points can be arbitrarily small, so the height of a quadtree can be arbitrarily large in the worst case

Building a quadtree

Let P = set of n points in the plane

Building a quadtree

Let P = set of n points in the plane

- Let's come up with a (recursive) algorithm to build quadtree of P

Building a quadtree

Let P = set of n points in the plane

- Let's come up with a (recursive) algorithm to build quadtree of P

  //create quadtree of P and return its root

Building a quadtree

Let P = set of n points in the plane

- Let's come up with a (recursive) algorithm to build quadtree of P

  //create quadtree of P and return its root

  buildQuadtree(set of points P, square S)

Building a quadtree

Let P = set of n points in the plane

Building a quadtree

Let P = set of n points in the plane

```
//create quadtree of P and return its root
```

Building a quadtree

Let P = set of n points in the plane

```
//create quadtree of P and return its root
buildQuadtree(set of points P, square S)
```

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
    - build a leaf node , store P in it, and return node

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
  - build a leaf node , store P in it, and return node
- else

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
    - build a leaf node , store P in it, and return node
- else
    - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
  - build a leaf node , store P in it, and return node
- else
  - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4
  - create a node

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
    - build a leaf node , store P in it, and return node
- else
    - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4
    - create a node
    - node ->child1 = buildQuadtree(P1, S1)

Building a quadtree

Let P = set of n points in the plane

```
//create quadtree of P and return its root
```

buildQuadtree(set of points P, square S)

- if P has at most one point:
  - build a leaf node , store P in it, and return node
- else
  - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4
  - create a node
  - node ->child1 = buildQuadtree(P1, S1)
  - node ->child2 = buildQuadtree(P2, S2)

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
  - build a leaf node , store P in it, and return node
- else
  - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4
  - create a node
  - node ->child1 = buildQuadtree(P1, S1)
  - node ->child2 = buildQuadtree(P2, S2)
  - node ->child3 = buildQuadtree(P3, S3)

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
  - build a leaf node , store P in it, and return node
- else
  - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4
  - create a node
  - node ->child1 = buildQuadtree(P1, S1)
  - node ->child2 = buildQuadtree(P2, S2)
  - node ->child3 = buildQuadtree(P3, S3)
  - node ->child4 = buildQuadtree(P4, S4)

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
    - build a leaf node , store P in it, and return node
- else
    - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4
    - create a node
    - node ->child1 = buildQuadtree(P1, S1)
    - node ->child2 = buildQuadtree(P2, S2)
    - node ->child3 = buildQuadtree(P3, S3)
    - node ->child4 = buildQuadtree(P4, S4)
    - return node

Building a quadtree

Let P = set of n points in the plane

//create quadtree of P and return its root

buildQuadtree(set of points P, square S)

- if P has at most one point:
  - build a leaf node , store P in it, and return node
- else
  - partition S into 4 quadrants S1, S2, S3, S4 and use them to partition P into P1, P2, P3, P4
  - create a node
  - node ->child1 = buildQuadtree(P1, S1)
  - node ->child2 = buildQuadtree(P2, S2)
  - node ->child3 = buildQuadtree(P3, S3)
  - node ->child4 = buildQuadtree(P4, S4)
  - return node

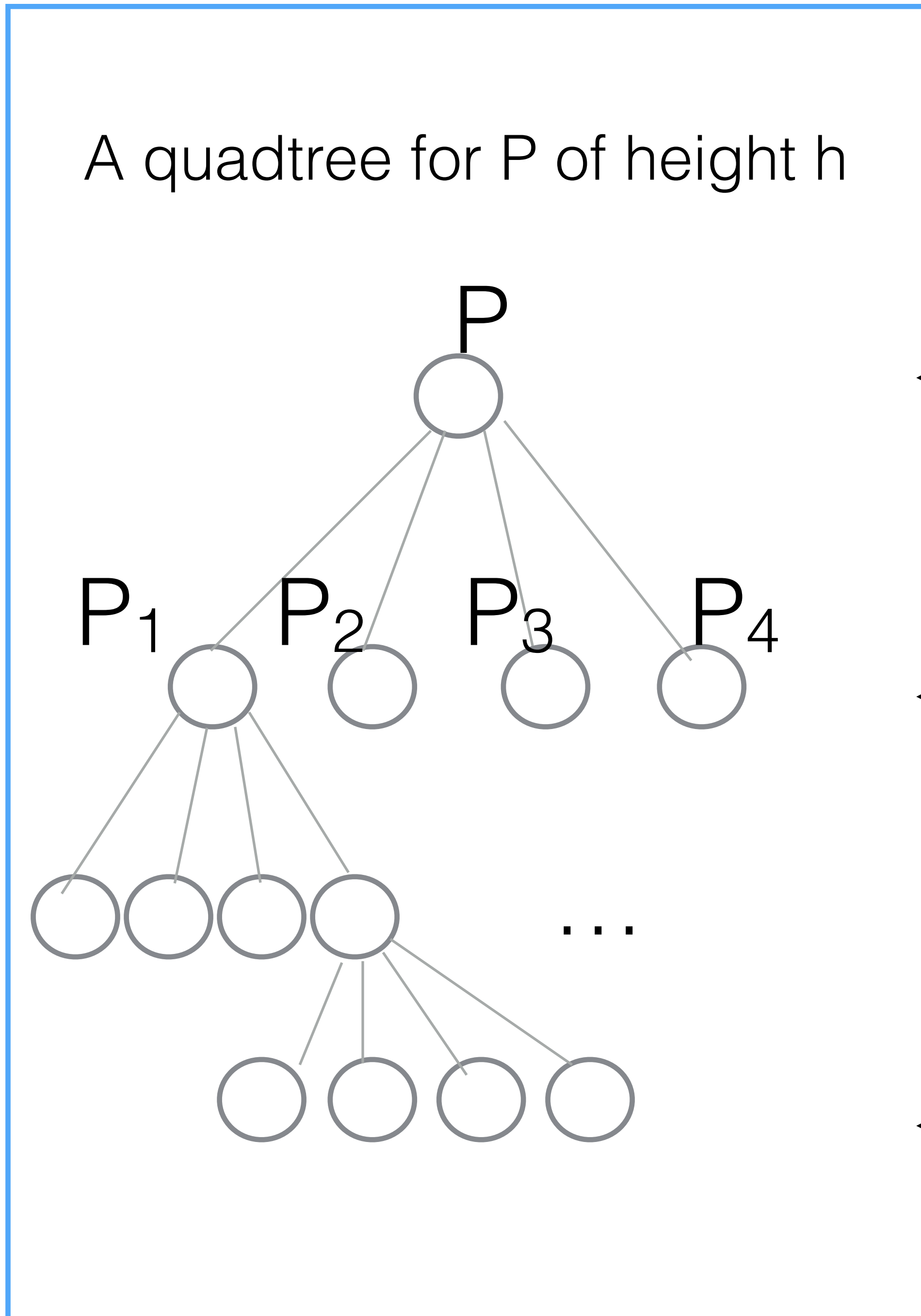
How long does this take, function of n and height h?

Building a quadtree

- Total time = total time in partitioning + total time in recursion

Partitioning

Let P = set of n points in the plane



partition P into P_1, P_2, P_3 P_4 takes $O(|P|) = O(n)$

partition P_1, P_2, P_3 P_4 into their quadrants takes $O(|P_1|) + O(|P_2|) + O(|P_3|) + O(|P_4|) = O(|P|) = O(n)$

…

Partitioning

Let P = set of n points in the plane

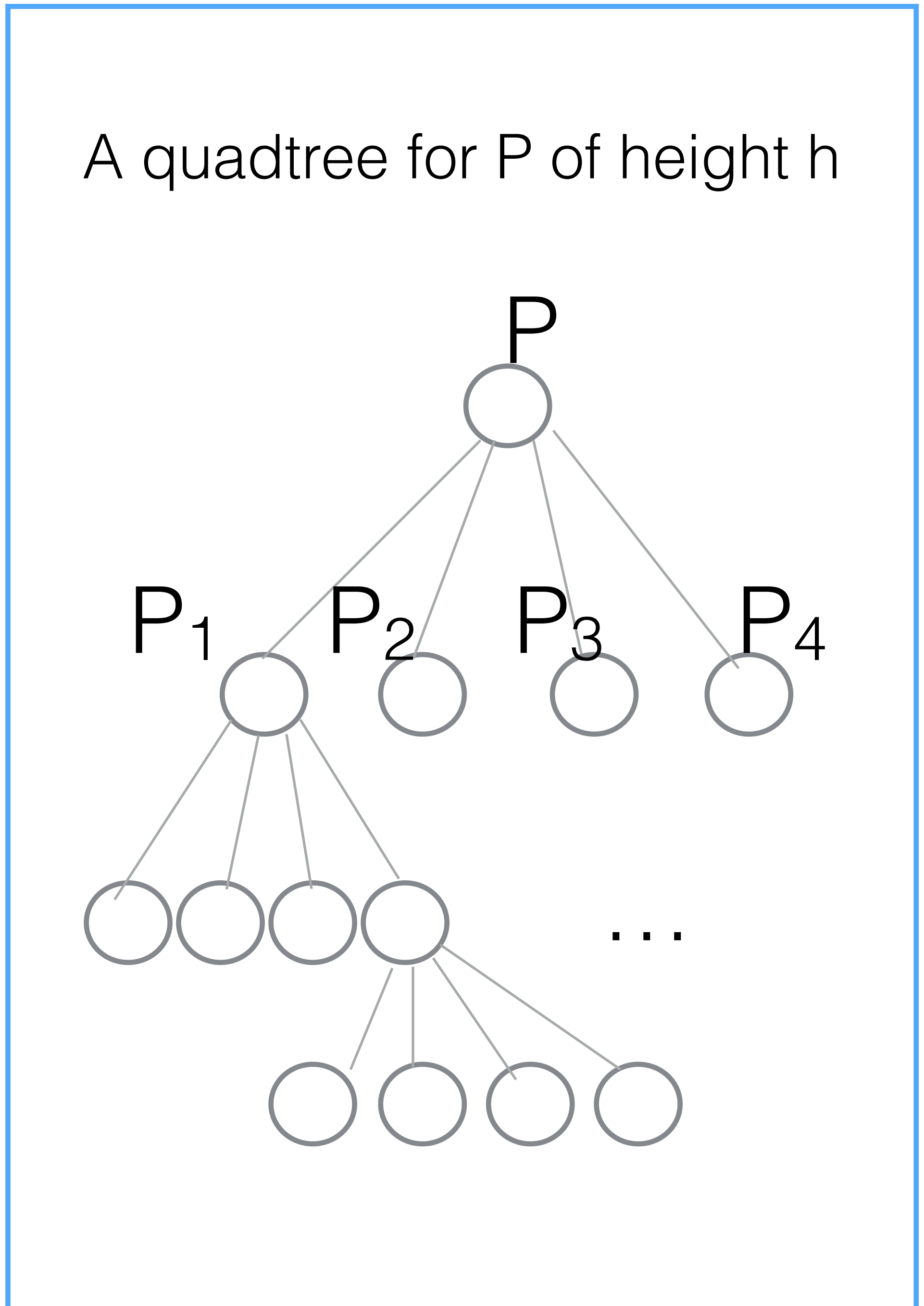
- Partitioning P into P1, P2, P3, P4 runs in time O(|P|)
- We cannot bound P1, P2, P3, P4 (each can have anywhere between 0 points and n points)
- But if we look at all nodes at same level in the quadtree: together they form a partition of the input square and the union of their points is P

==> The time to partition, at every level, is O(n)

==> Summed over the entire quadtree partition will take O(h x n) in total

Building a quadtree

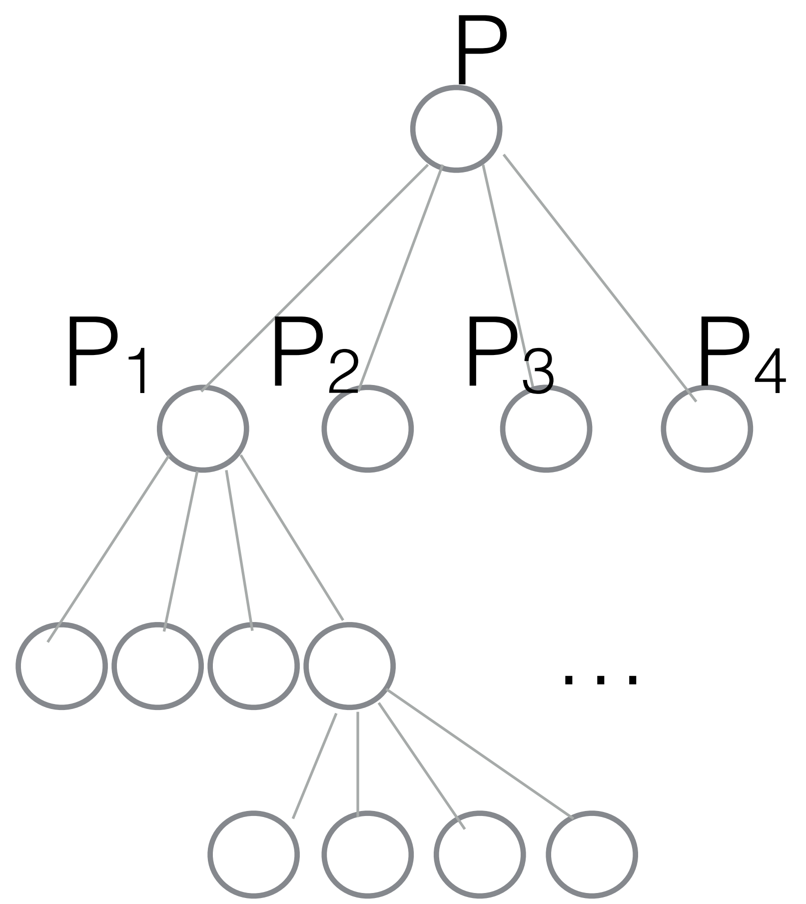
Let P = set of n points in the plane



- Recursion
  - Every recursive call creates a node
  - How many nodes?

Building a quadtree

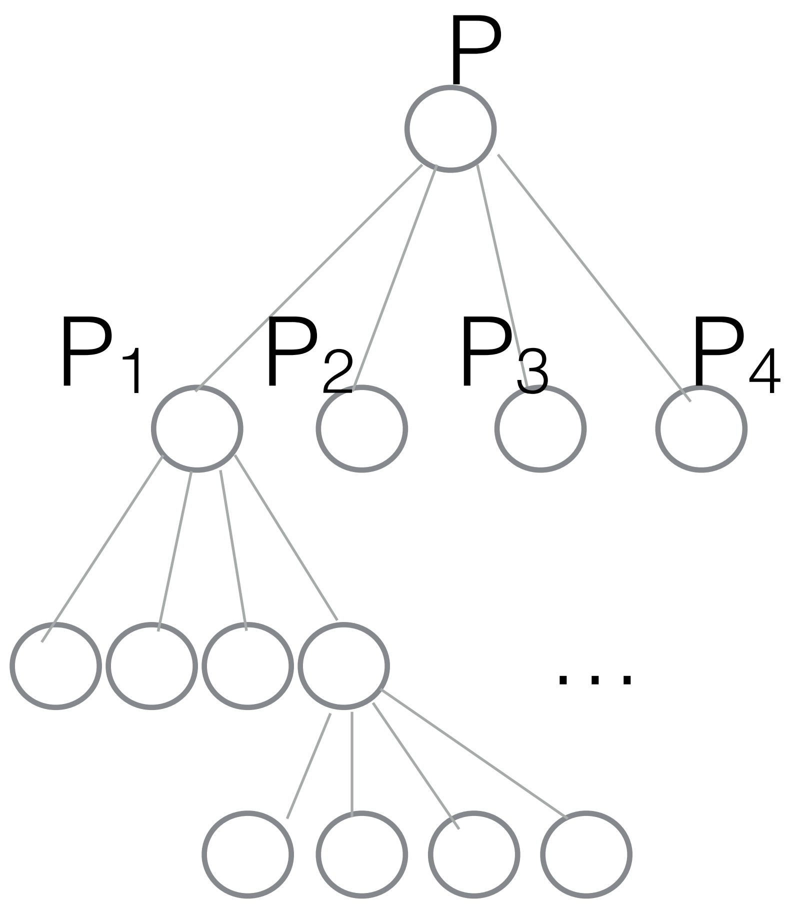
A quadtree for P of height h



How many nodes?

Building a quadtree

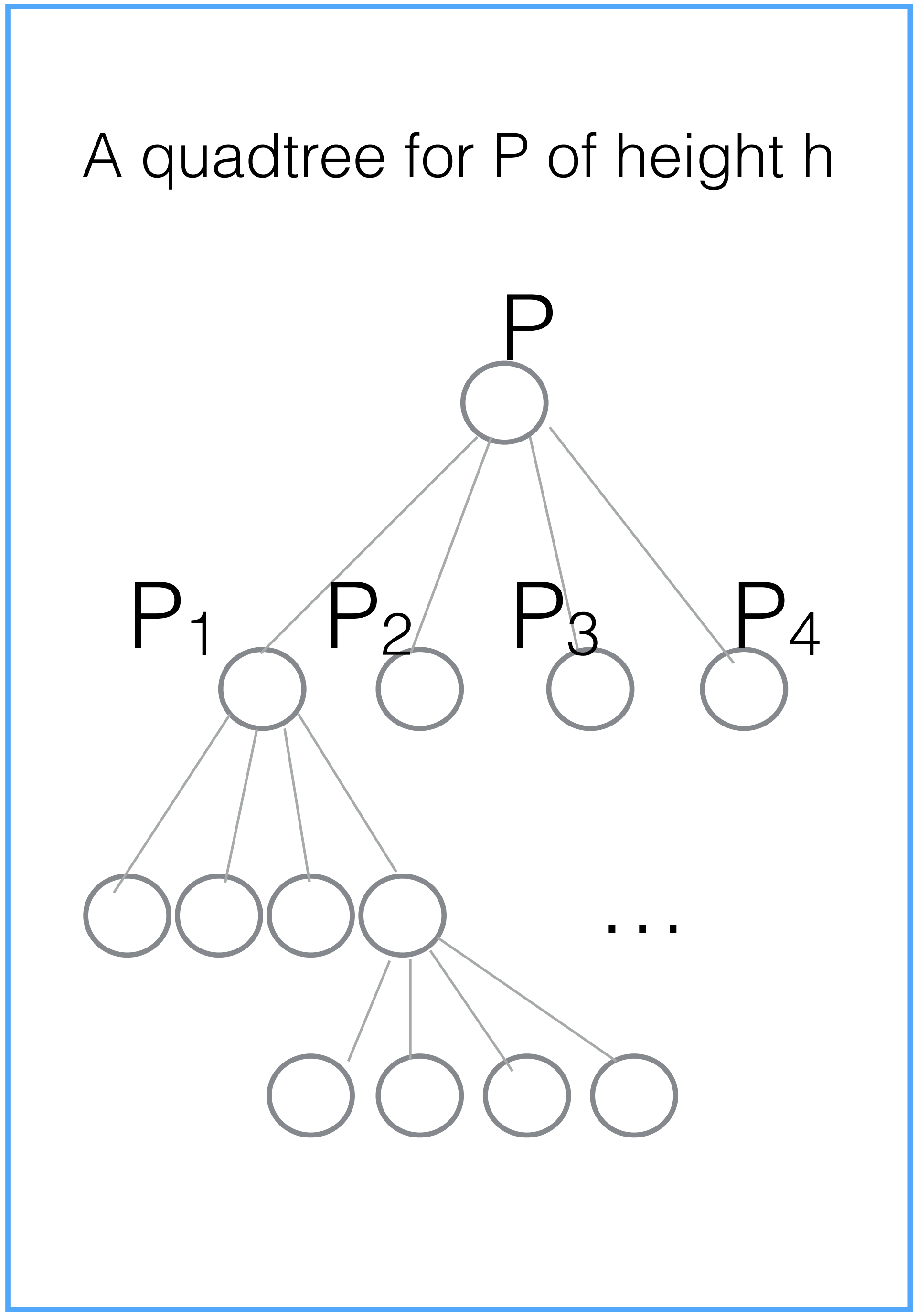
A quadtree for P of height h



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)

Building a quadtree

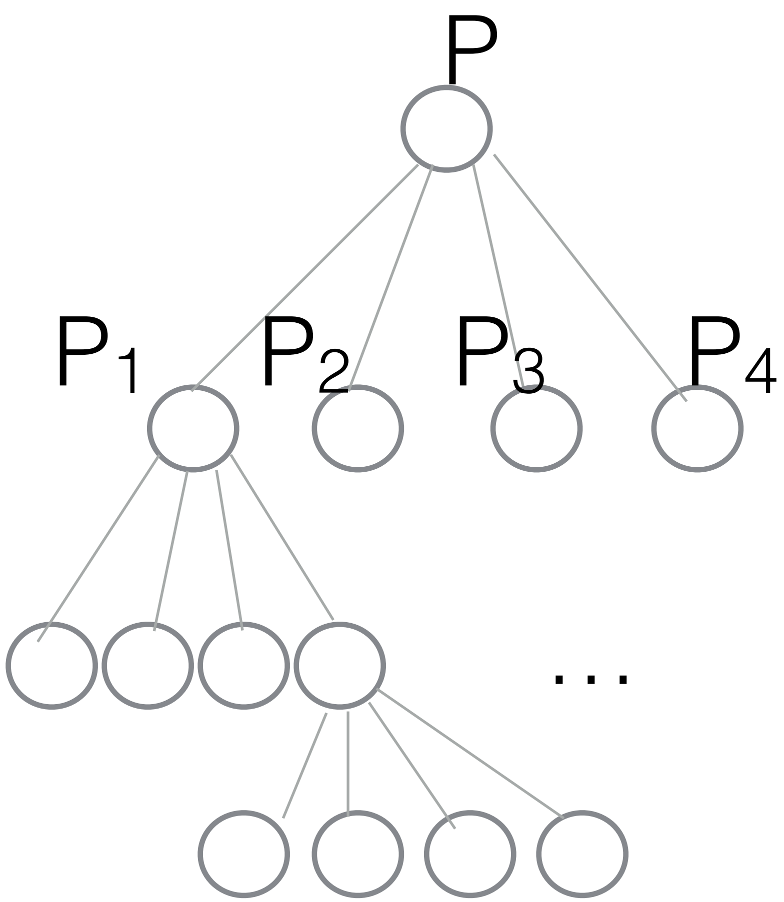


How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)

Building a quadtree

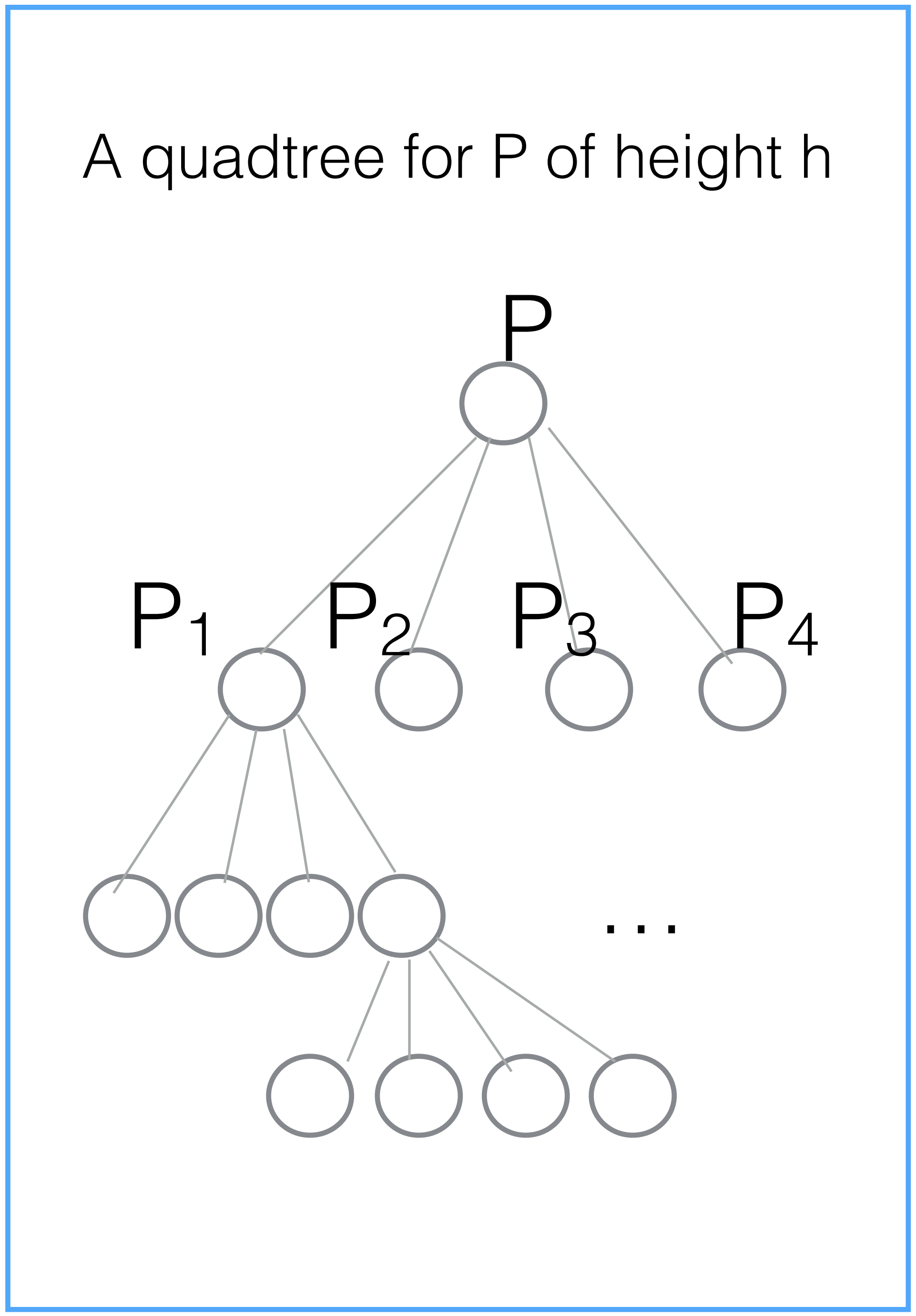
A quadtree for P of height h



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- Each node has 0 or 4 children

Building a quadtree

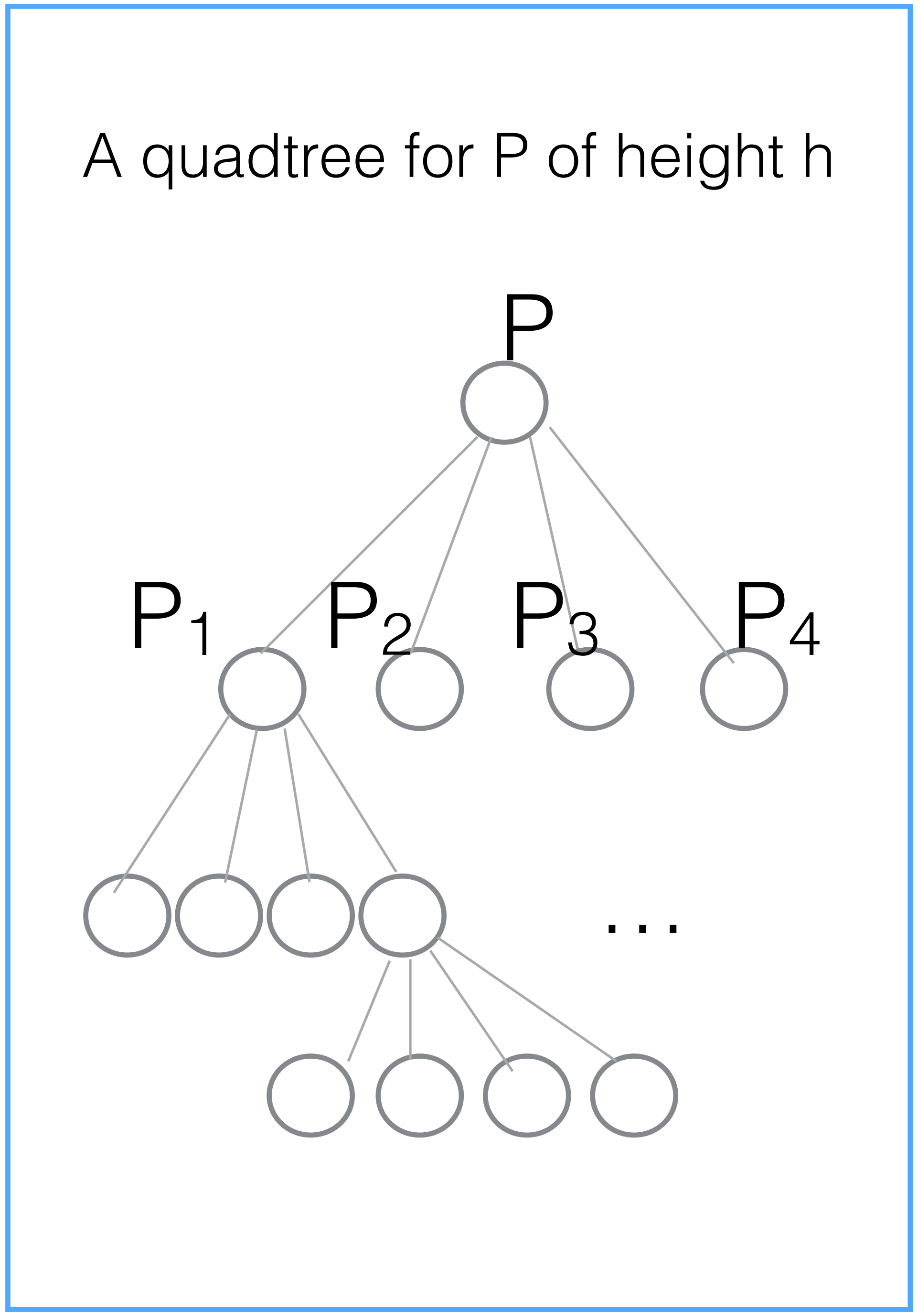


How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)

- Each node has 0 or 4 children

Building a quadtree

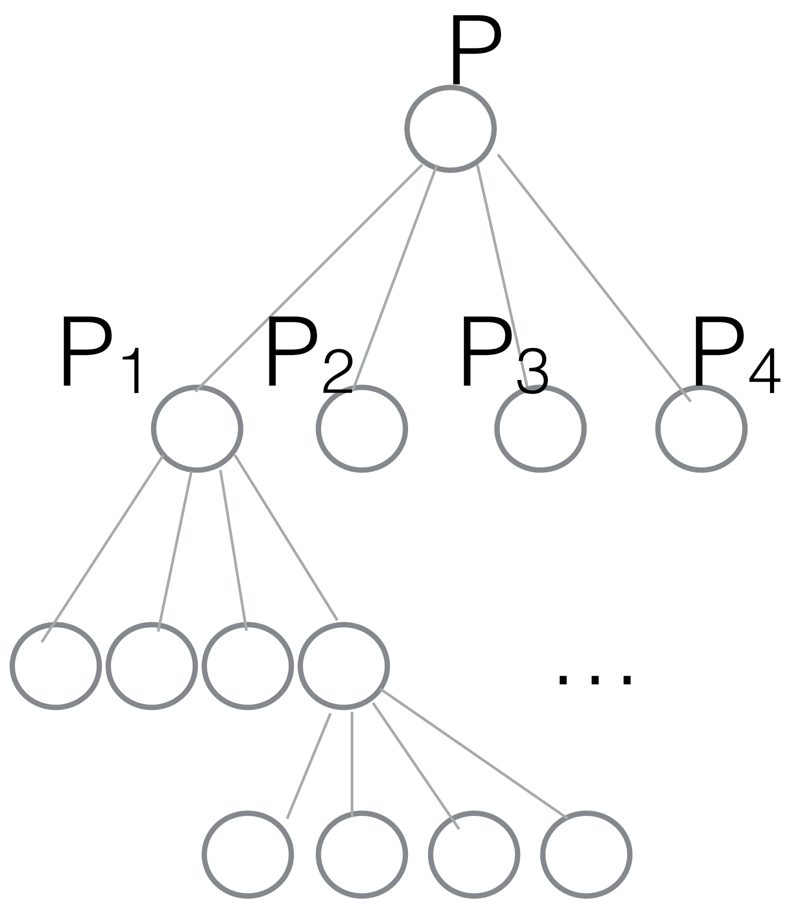


How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- Each node has 0 or 4 children
- A relation between I and L?

Building a quadtree

A quadtree for P of height h



How many nodes?

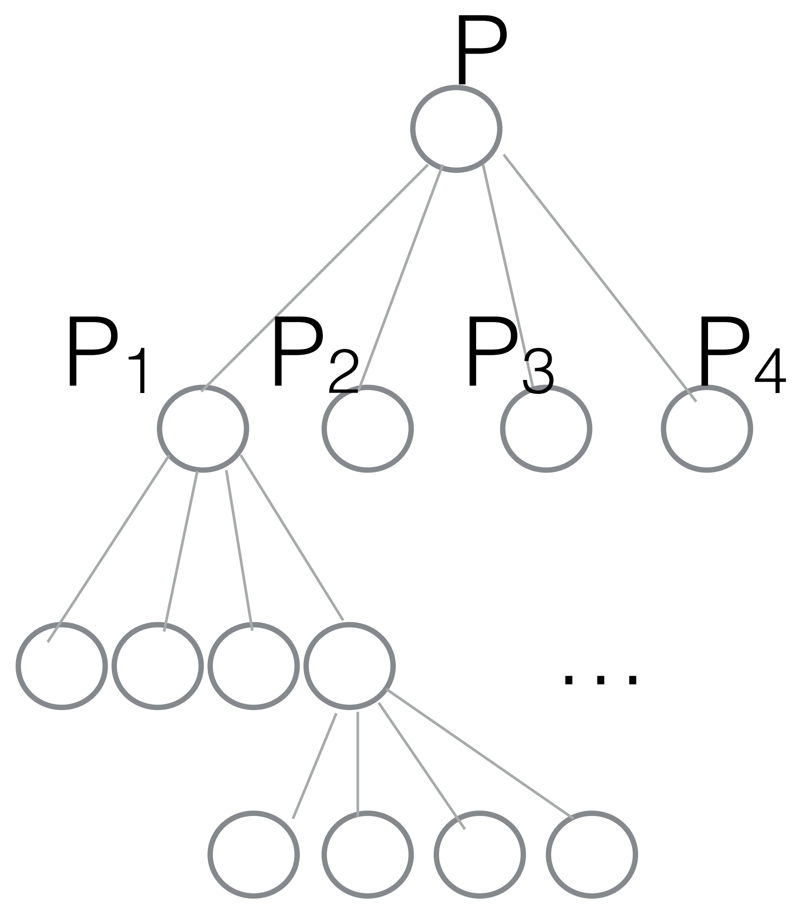
- nodes (N) = internal nodes (I)+ leaves (L)
- Each node has 0 or 4 children
- A relation between I and L?

$$L = 3\,I + 1$$

(Proof: by induction)

Building a quadtree

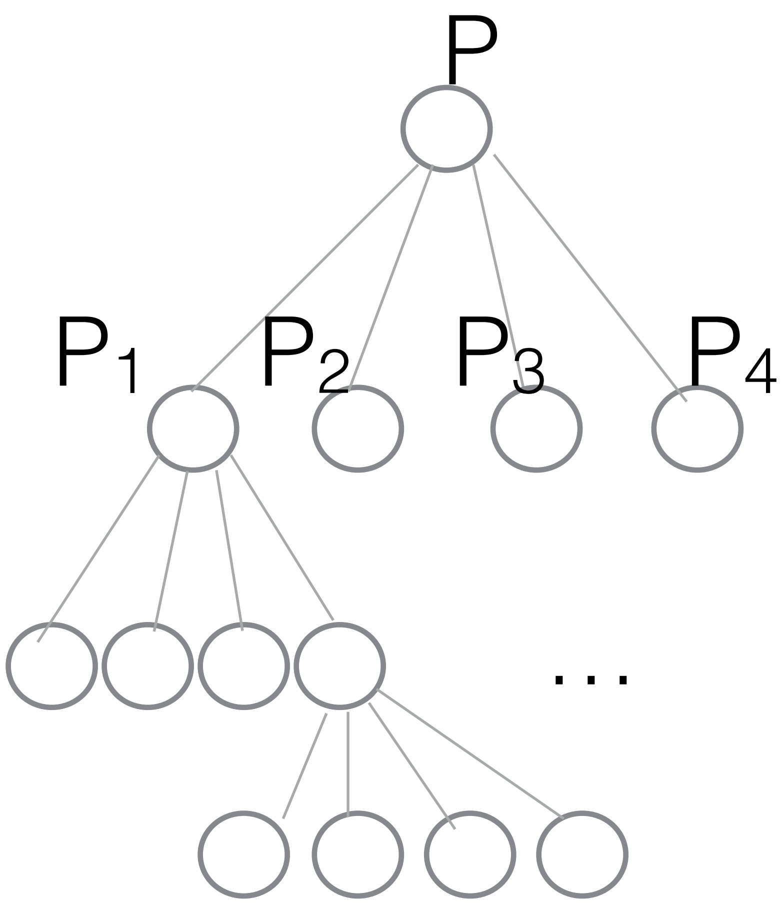
A quadtree for P of height h



How many nodes?

Building a quadtree

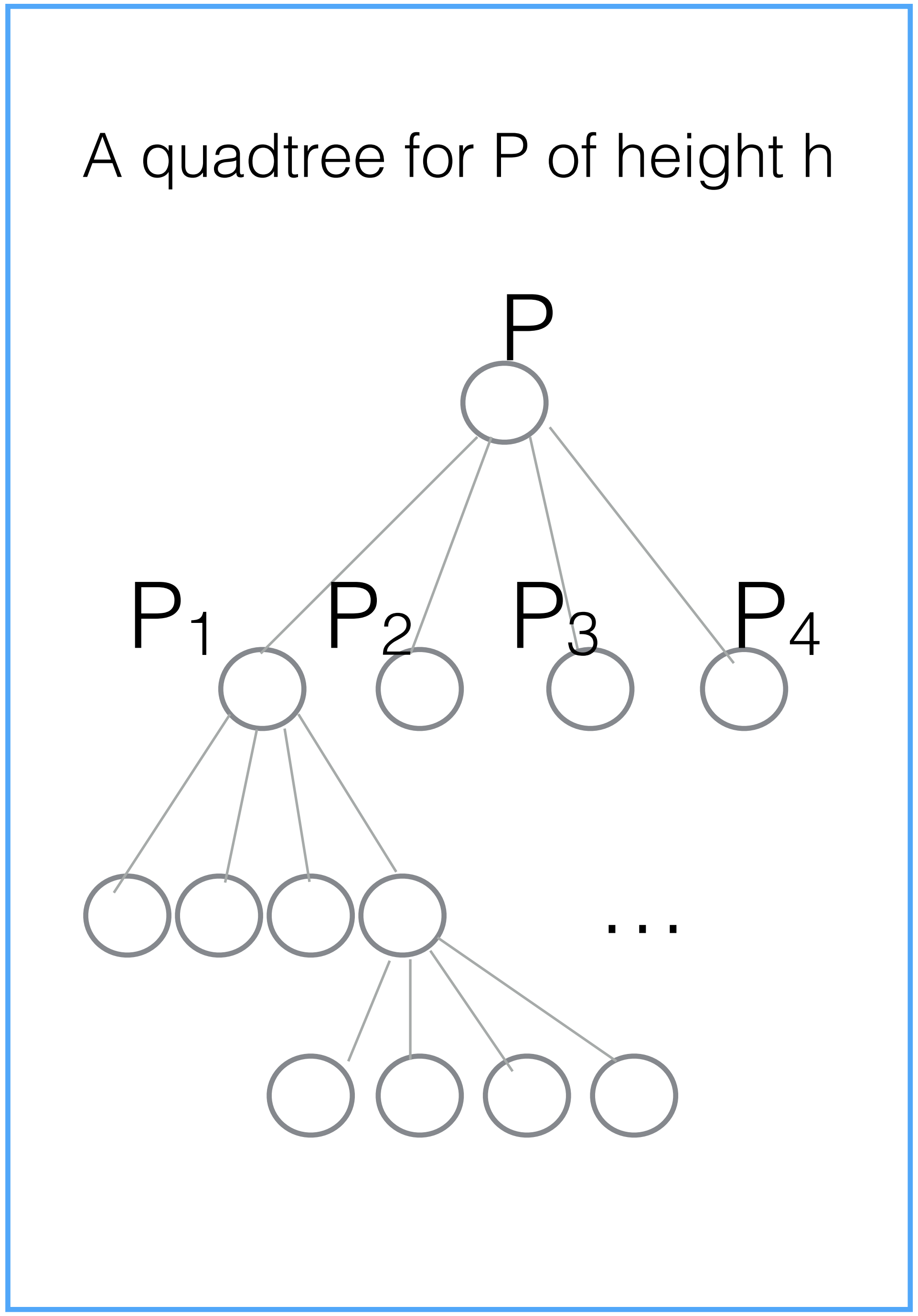
A quadtree for P of height h



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)

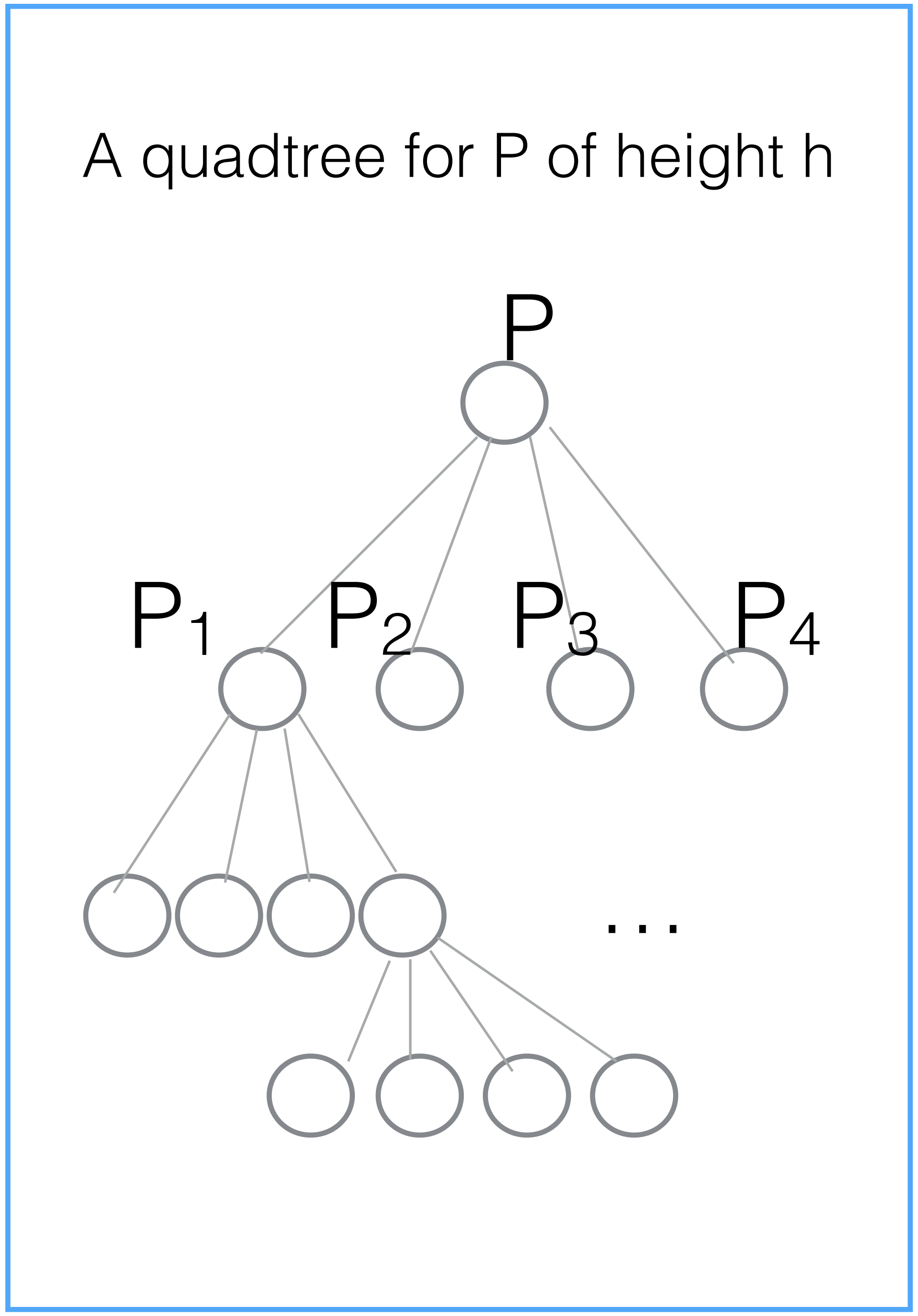
Building a quadtree



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1

Building a quadtree

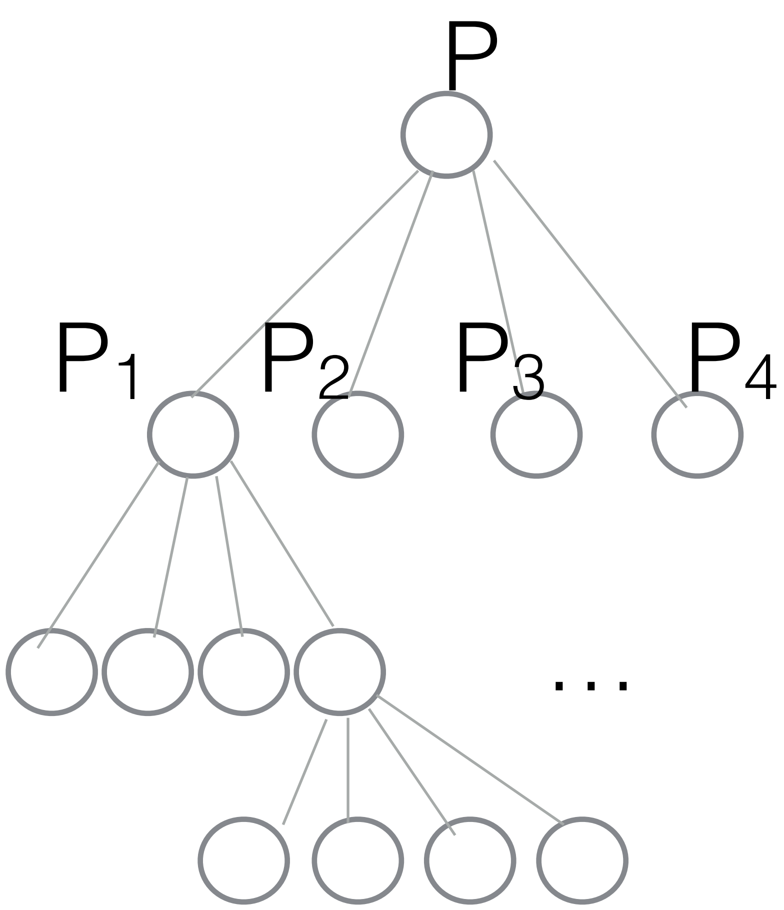


How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?

Building a quadtree

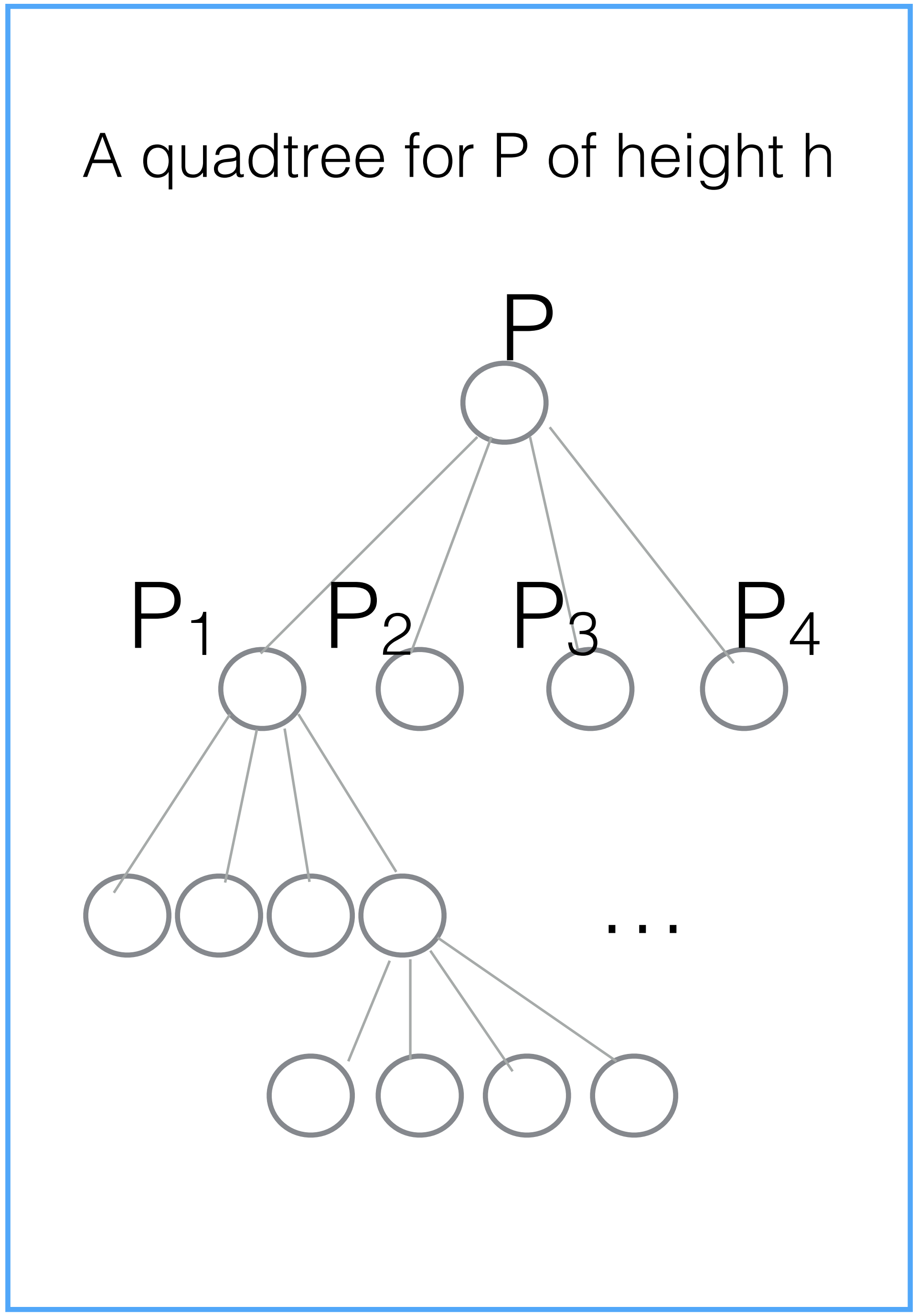
A quadtree for P of height h



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?
    - can be unbounded

Building a quadtree

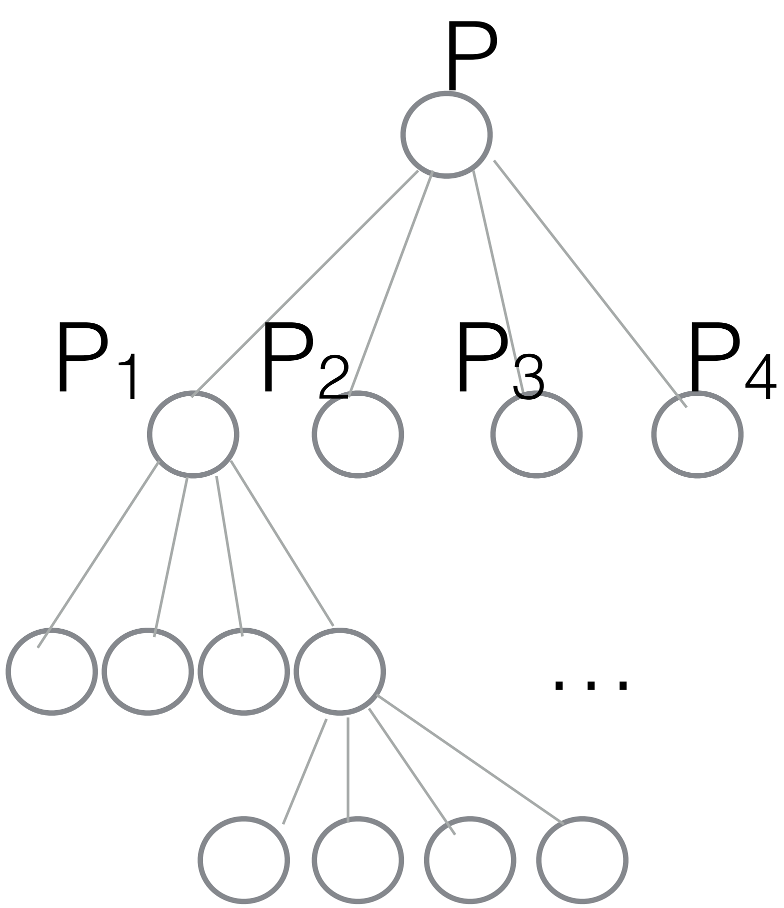


How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?
    - can be unbounded
    - want to express function of h

Building a quadtree

A quadtree for P of height h

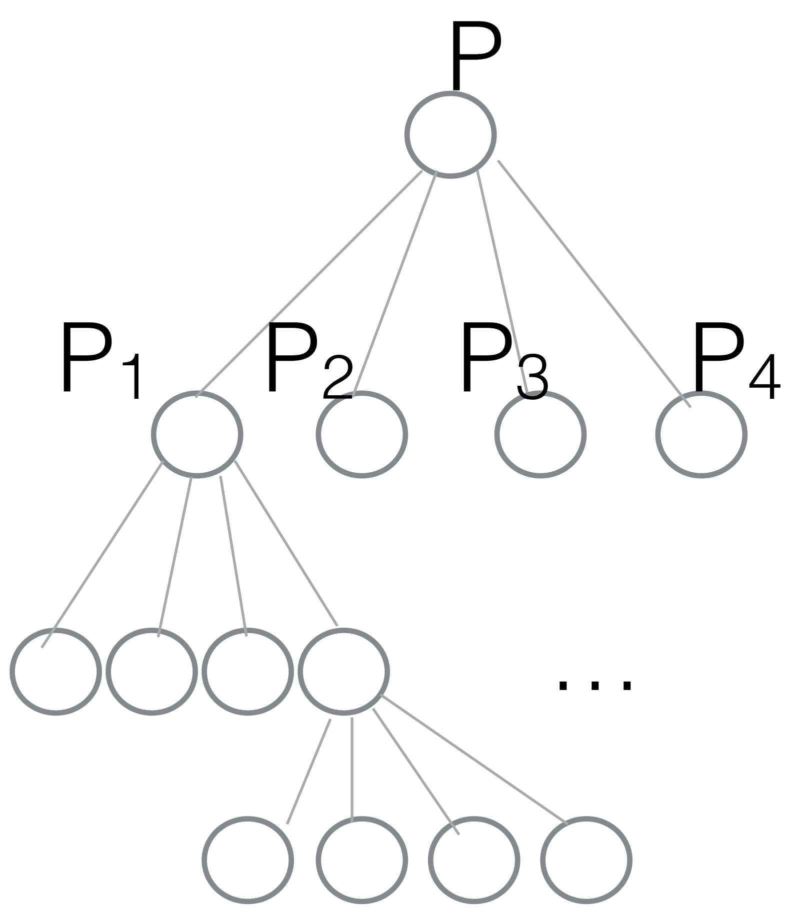


How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?
    - can be unbounded
    - want to express function of h

Building a quadtree

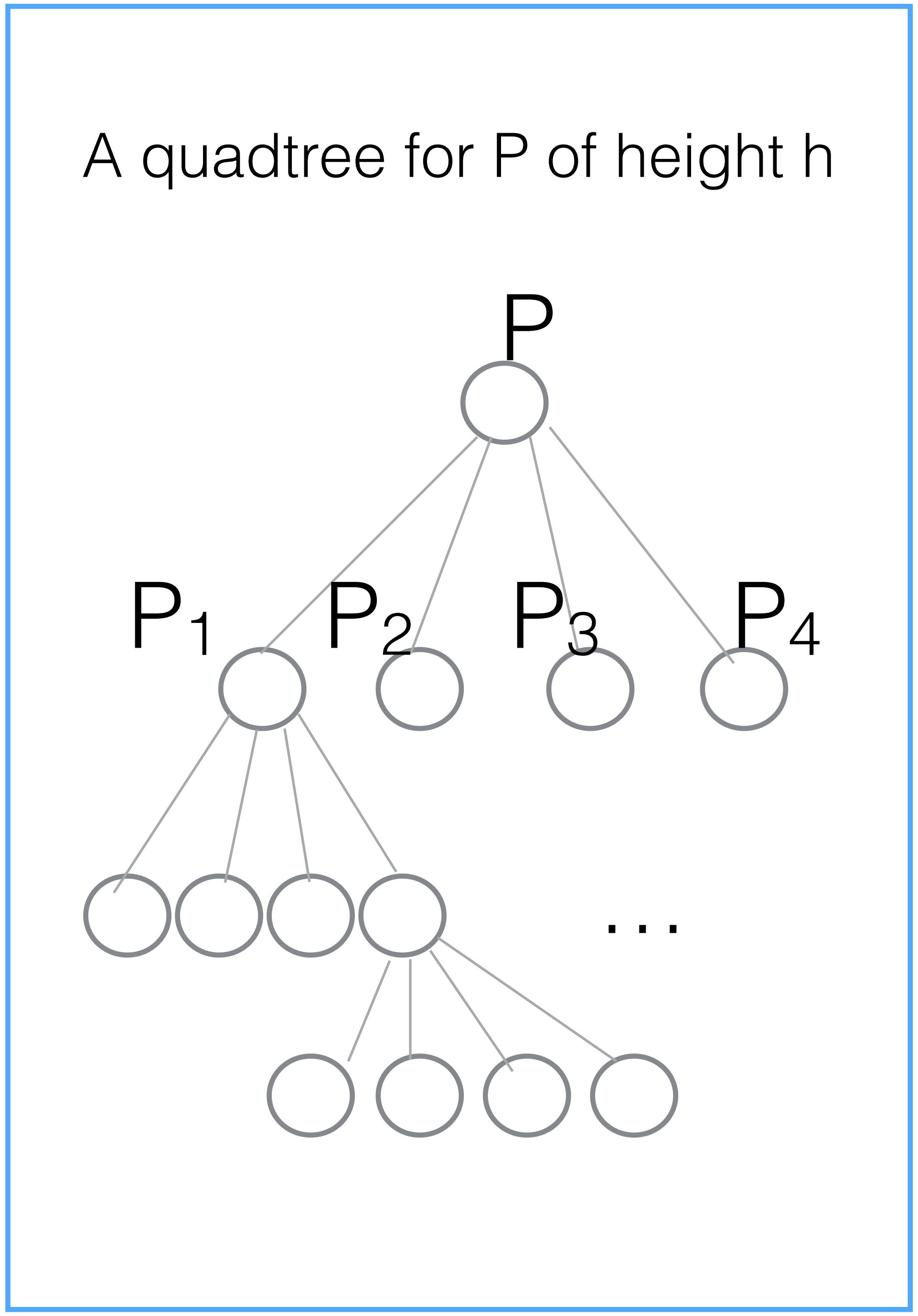
A quadtree for P of height h



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?
    - can be unbounded
    - want to express function of h
    - in the best case each leaf contains one point (no empty leaves); in the worst case there can be many empty leaves ==> many internal nodes

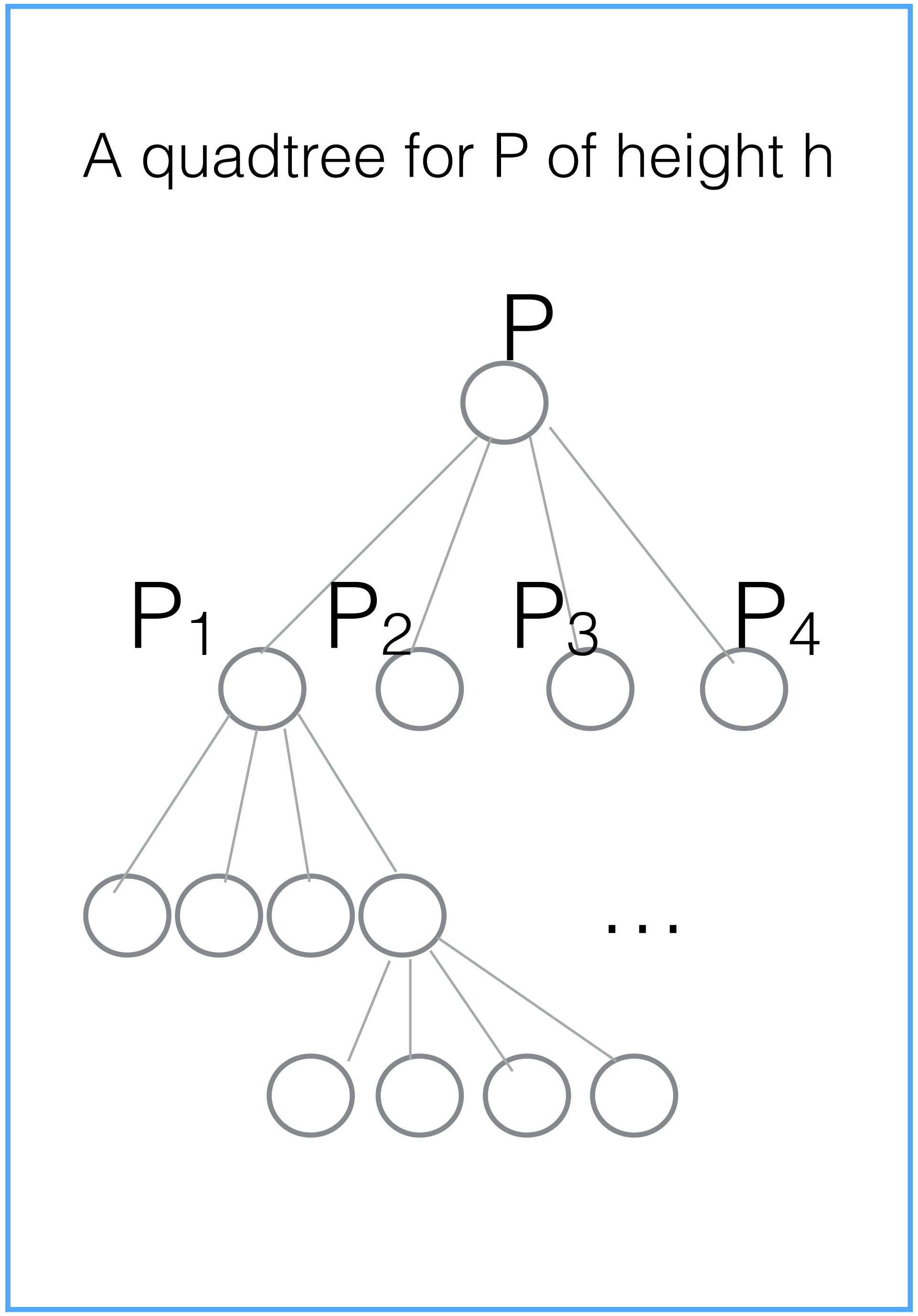
Building a quadtree



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?
    - can be unbounded
    - want to express function of h
    - in the best case each leaf contains one point (no empty leaves); in the worst case there can be many empty leaves ==> many internal nodes

Building a quadtree

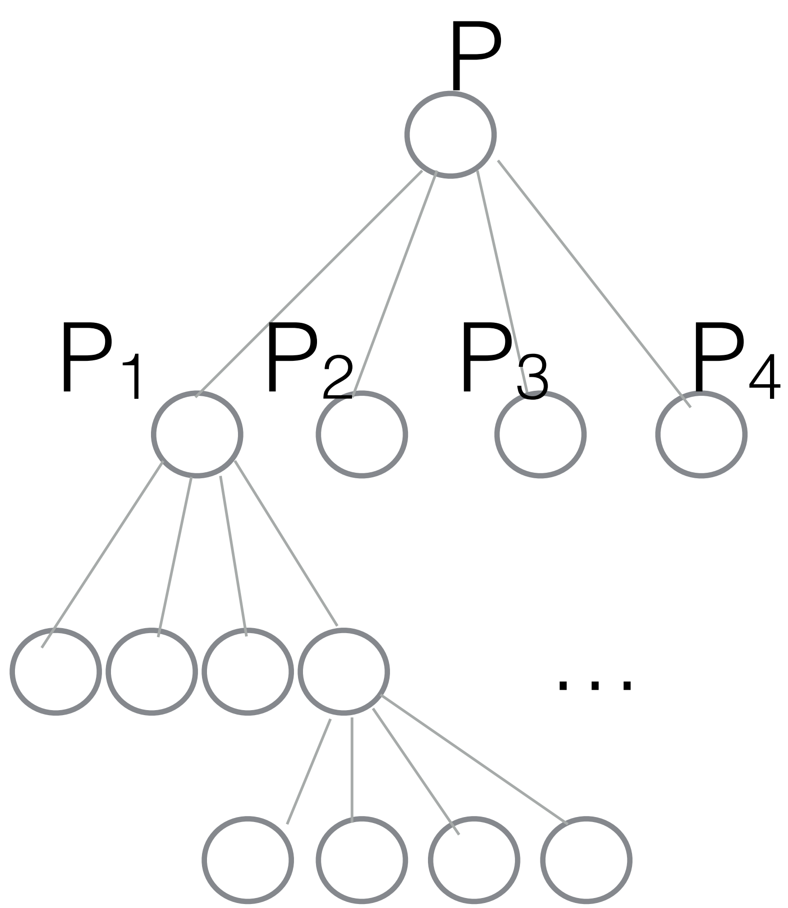


How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?
    - can be unbounded
    - want to express function of h
    - in the best case each leaf contains one point (no empty leaves); in the worst case there can be many empty leaves ==> many internal nodes
    - at each level, the internal nodes partition the original square and each internal node contains at least 2 points ==> O(n) internal nodes per level —> O(n x h)

Building a quadtree

A quadtree for P of height h



How many nodes?

- nodes (N) = internal nodes (I)+ leaves (L)
- L = 3 I + 1 —> N = I + 3I + 1 = 4I + 1
- How many internal nodes?
    - can be unbounded
    - want to express function of h
    - in the best case each leaf contains one point (no empty leaves); in the worst case there can be many empty leaves ==> many internal nodes
    - at each level, the internal nodes partition the original square and each internal node contains at least 2 points ==> O(n) internal nodes per level —> $O(n \times h)$

$\boxed{O(n \times h) \text{ nodes}}$

Summary

Theorem:

A quadtree for a set P of points in the plane:

- has height $h = O(\lg(1/d))$ (where d is closest distance)
- has $O(h \times n)$ nodes; and
- can be built in $O(h \times n)$ time.

Summary

Theorem:

A quadtree for a set P of points in the plane:

- has height $h = O(\lg(1/d))$ (where d is closest distance)
- has $O(h \times n)$ nodes; and
- can be built in $O(h \times n)$ time.

- Theoretical worst case:

Summary

Theorem:

A quadtree for a set P of points in the plane:

- has height $h = O(\lg(1/d))$ (where d is closest distance)
- has $O(h \times n)$ nodes; and
- can be built in $O(h \times n)$ time.

- Theoretical worst case:
  - height and size are unbounded

Summary

Theorem:

A quadtree for a set P of points in the plane:

- has height $h = O(\lg(1/d))$ (where d is closest distance)
- has $O(h \times n)$ nodes; and
- can be built in $O(h \times n)$ time.

- Theoretical worst case:
    - height and size are unbounded
- In practice:

Summary

Theorem:

A quadtree for a set P of points in the plane:

- has height $h = O(\lg (1/d))$ (where d is closest distance)
- has $O(h \times n)$ nodes; and
- can be built in $O(h \times n)$ time.

- Theoretical worst case:
  - height and size are unbounded
- In practice:
  - often $h=O(n)$ ==> size = $O(n^2)$, build time is $O(n^2)$

Summary

Theorem:

A quadtree for a set P of points in the plane:

- has height $h = O(\lg (1/d))$ (where d is closest distance)
- has $O(h \times n)$ nodes; and
- can be built in $O(h \times n)$ time.

- Theoretical worst case:
  - height and size are unbounded
- In practice:
  - often $h=O(n)$ ==> size = $O(n^2)$, build time is $O(n^2)$
  - For sets of points that are uniformly distributed, quadtrees have height $h = O(\lg n)$, size $O(n)$ and can be built in $O(n \lg n)$ time.

Compressed (point) quadtrees

Exercise

Let P = set of n points in the plane

- Draw a quadtree of arbitrarily large size corresponding to a small set of points in the plane (pick n=2 or n=3).
    - How many leaves are empty / non-empty?
    - Why is the size of the quadtree super-linear?

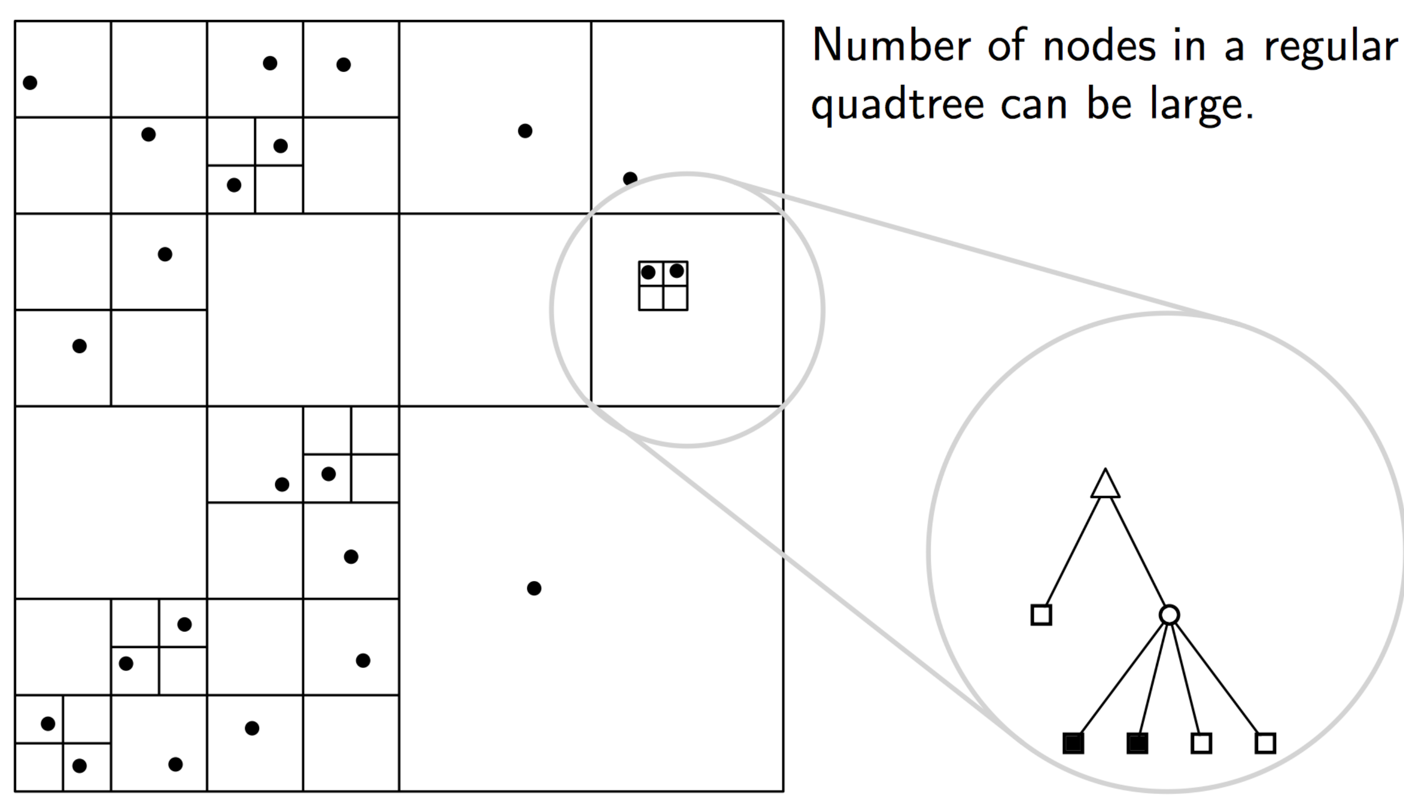
- Compress the quadtree as follows:
- compress paths of nodes with 3 empty children into one node
- this node is called a *donut*
- a node may have 5 children, an empty *donut* + 4 regular quadrants

Compressed quadtrees

Let P = set of n points in the plane

A compressed quadtree is a regular quadtree where paths of nodes with 3 empty children are compressed into one node (called: donut)

- a node may have 5 children, an empty *donut* + 4 regular quadrants



Number of nodes in a regular quadtree can be large.

Compressed quadtrees

Let P = set of n points in the plane

A compressed quadtree is a regular quadtree where paths of nodes with 3 empty children are compressed into one node (called: donut)

- a node may have 5 children, an empty *donut* + 4 regular quadrants

Compressed quadtrees

Let P = set of n points in the plane

A compressed quadtree is a regular quadtree where paths of nodes with 3 empty children are compressed into one node (called: donut)

- a node may have 5 children, an empty *donut* + 4 regular quadrants

- What does this mean in terms of size?

Compressed quadtrees

Let P = set of n points in the plane

A compressed quadtree is a regular quadtree where paths of nodes with 3 empty children are compressed into one node (called: donut)

- a node may have 5 children, an empty *donut* + 4 regular quadrants

- What does this mean in terms of size?

Theorem: A compressed quadtree has O(n) nodes and h=O(n) height.

Compressed quadtrees

Let P = set of n points in the plane

A compressed quadtree is a regular quadtree where paths of nodes with 3 empty children are compressed into one node (called: donut)

- a node may have 5 children, an empty *donut* + 4 regular quadrants

- What does this mean in terms of size?

Theorem: A compressed quadtree has O(n) nodes and h=O(n) height.

- Can you argue why..?

Applications of quadtrees

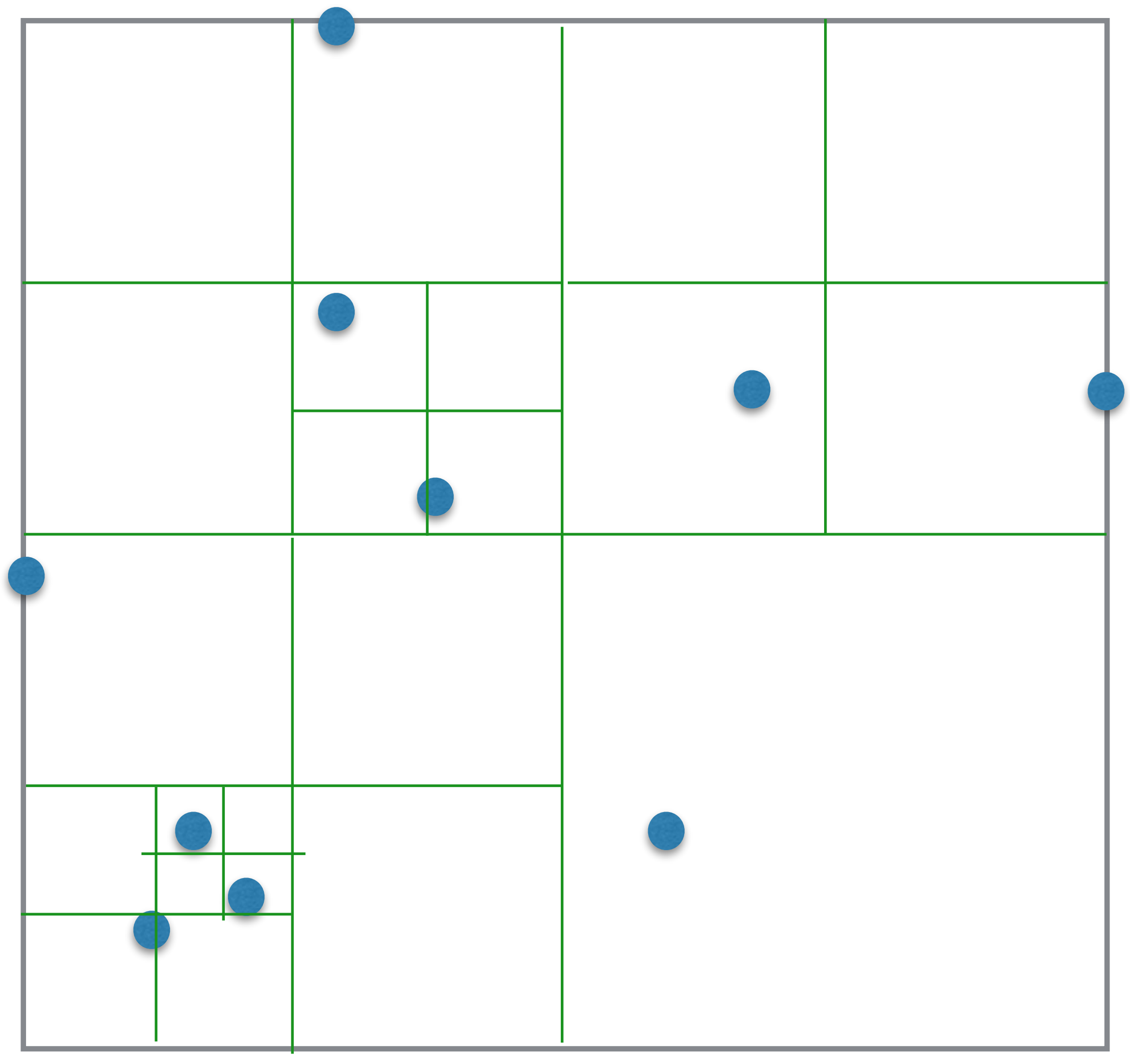
Applications of quadtrees

- Hundreds of papers
- Specialized quadtrees
  - customized for specific types of data (images, edges, polygons)
  - customized for specific applications
  - customized for large data
- Used to answer queries on spatial data such as:
  - point location
  - nearest neighbor (NN)
  - k-NNs
  - range searching
  - find all segments intersecting a given segment
  - meshing
  - …

Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

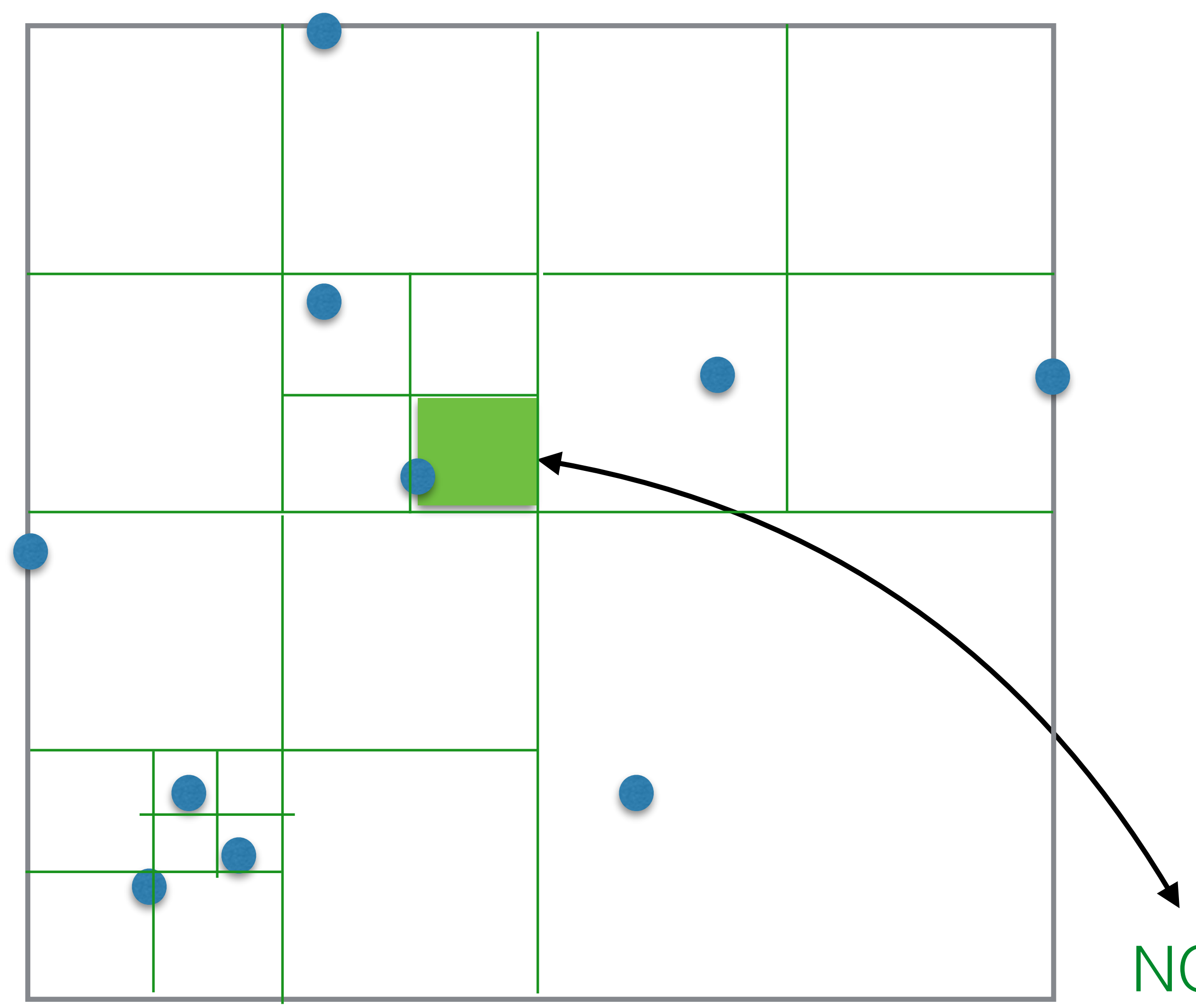
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v’ such that region(v’) is adjacent to region(v) in the given direction.

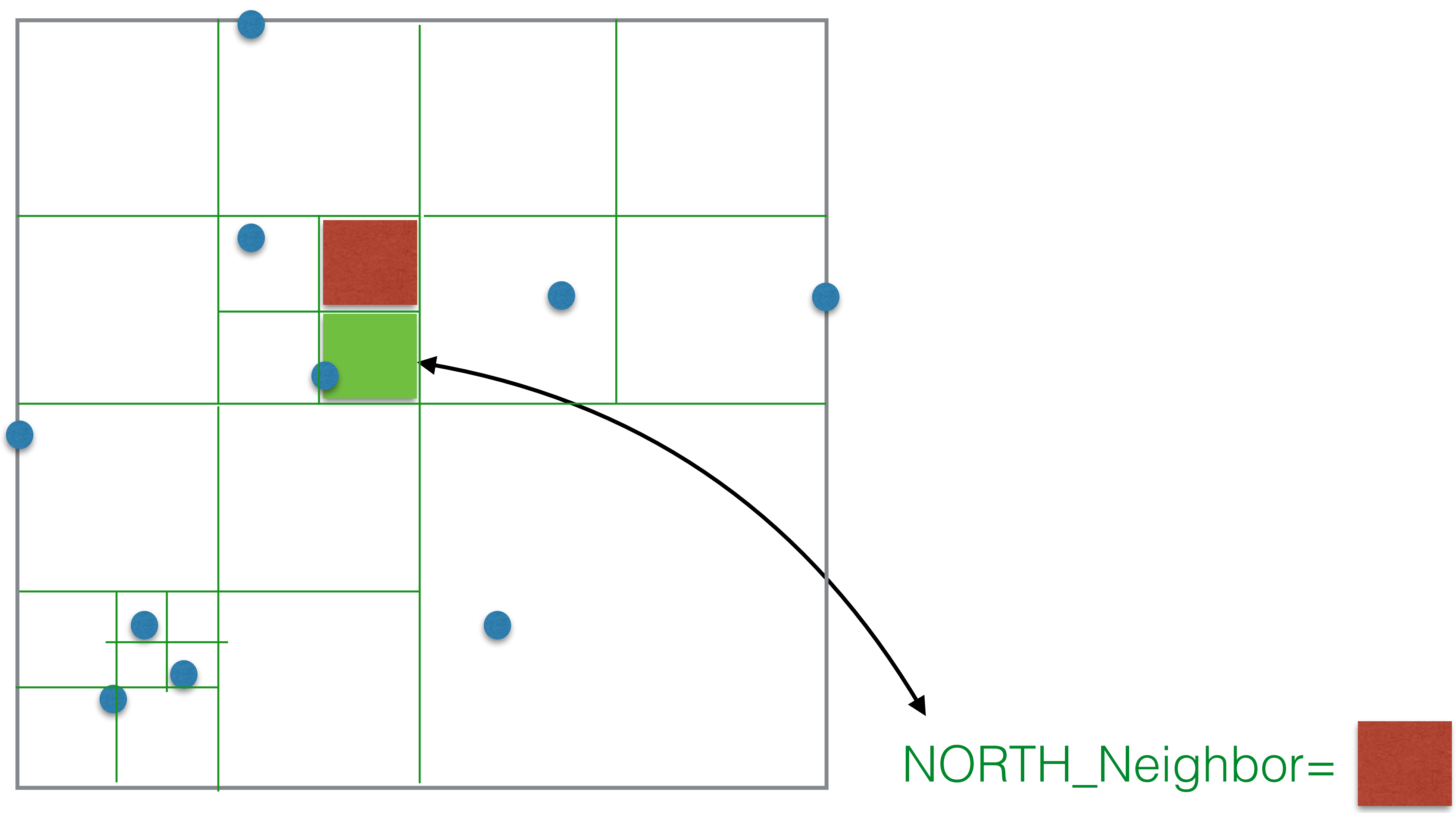
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

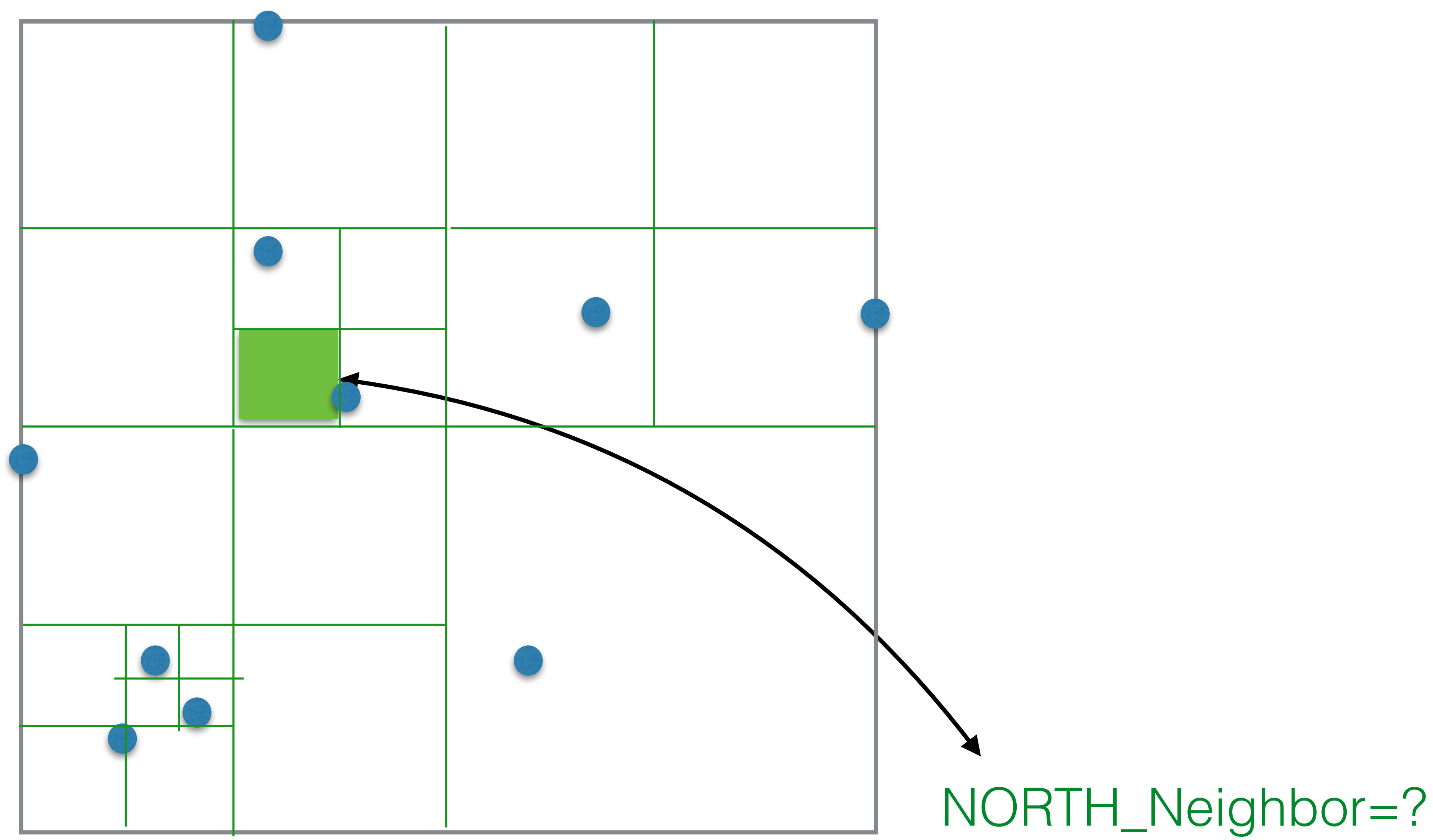
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

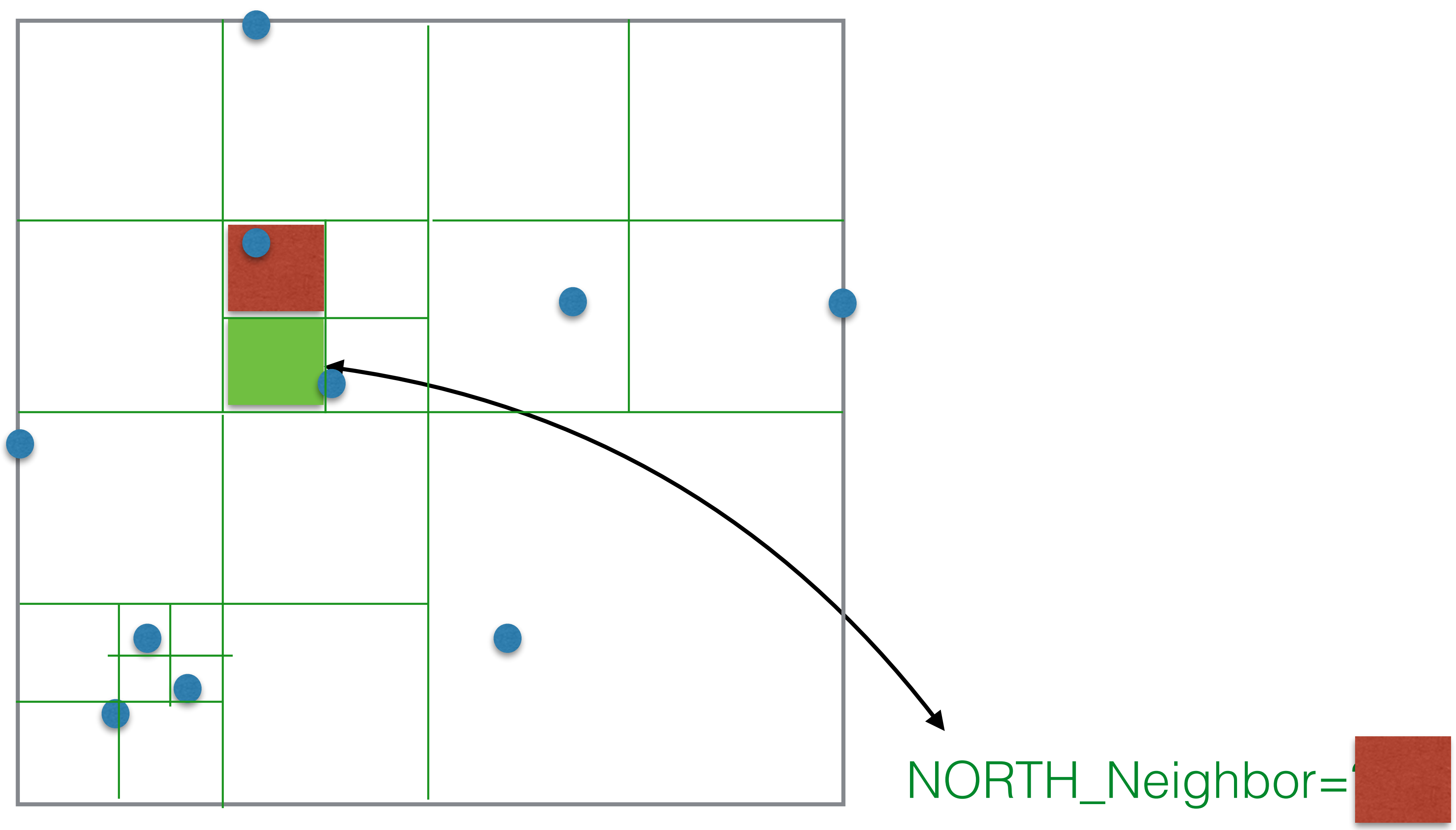
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

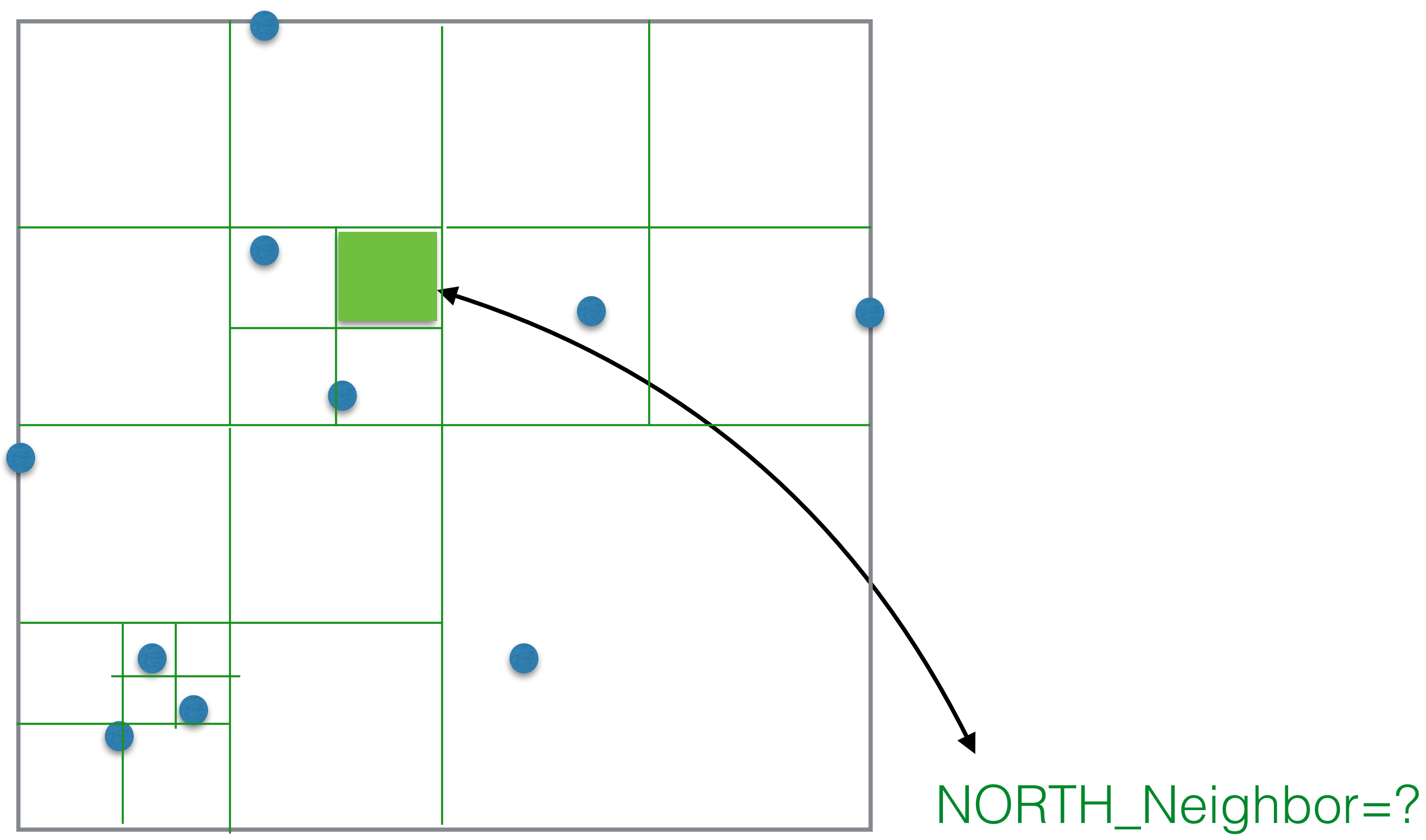
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

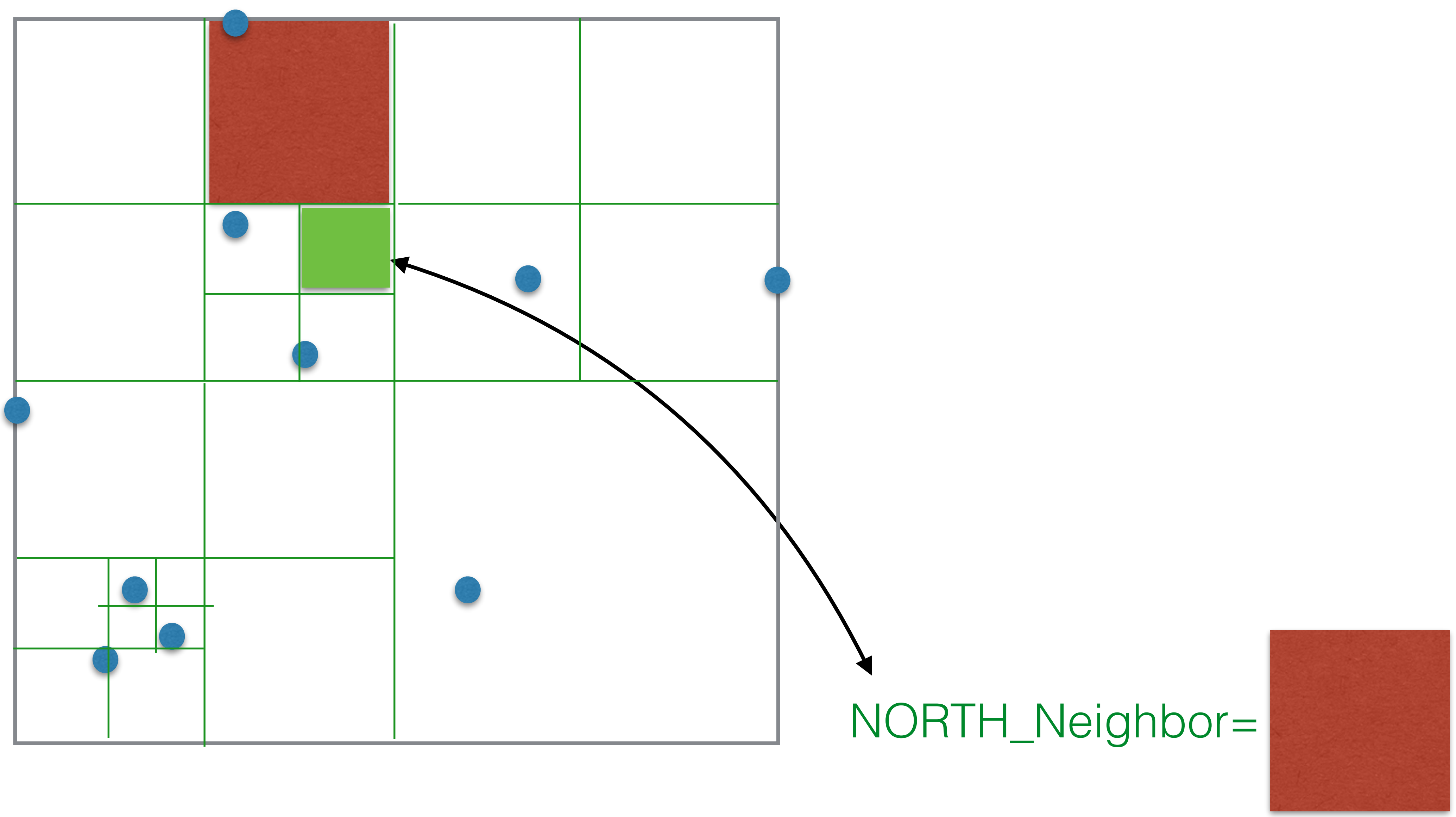
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

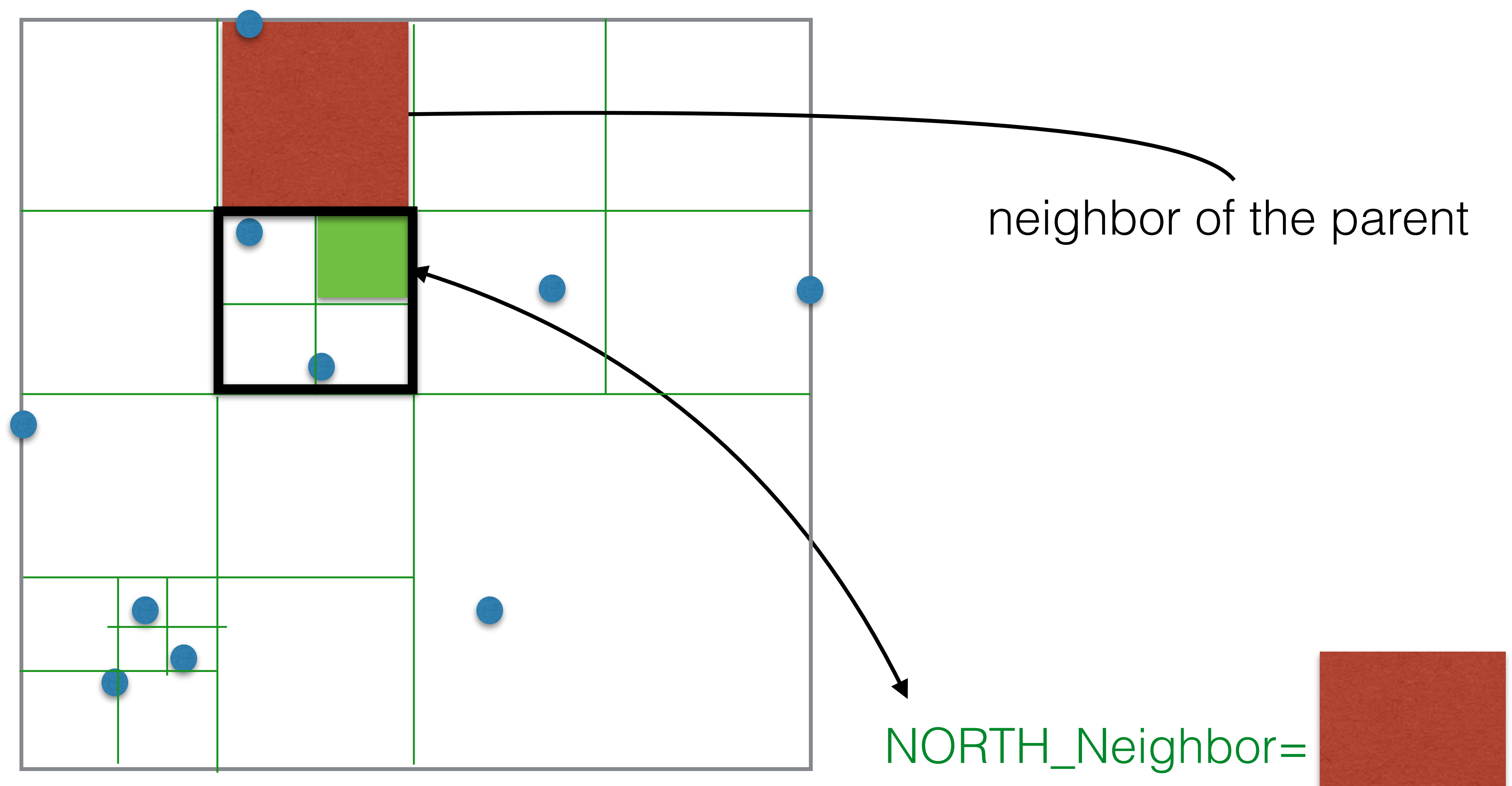
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

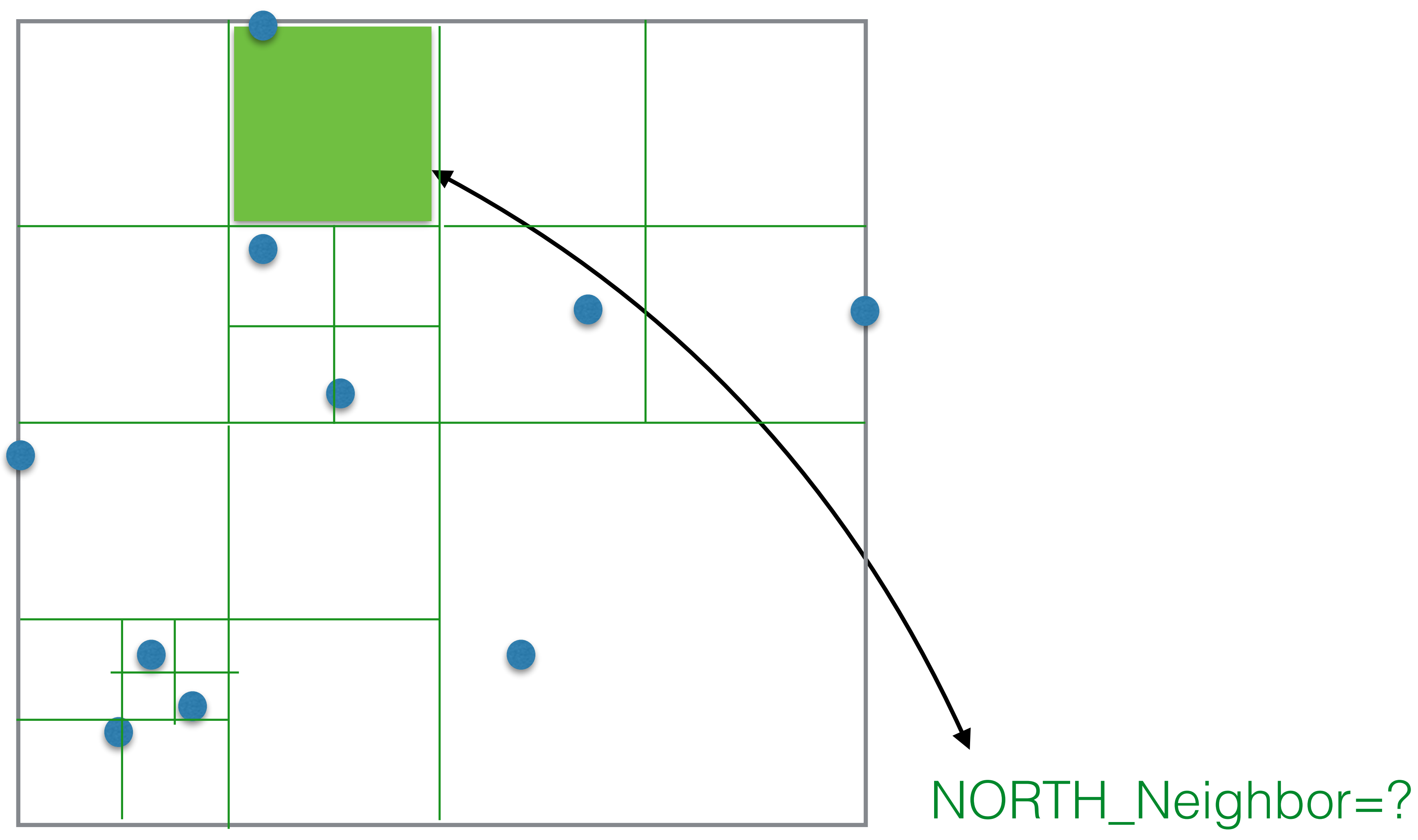
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

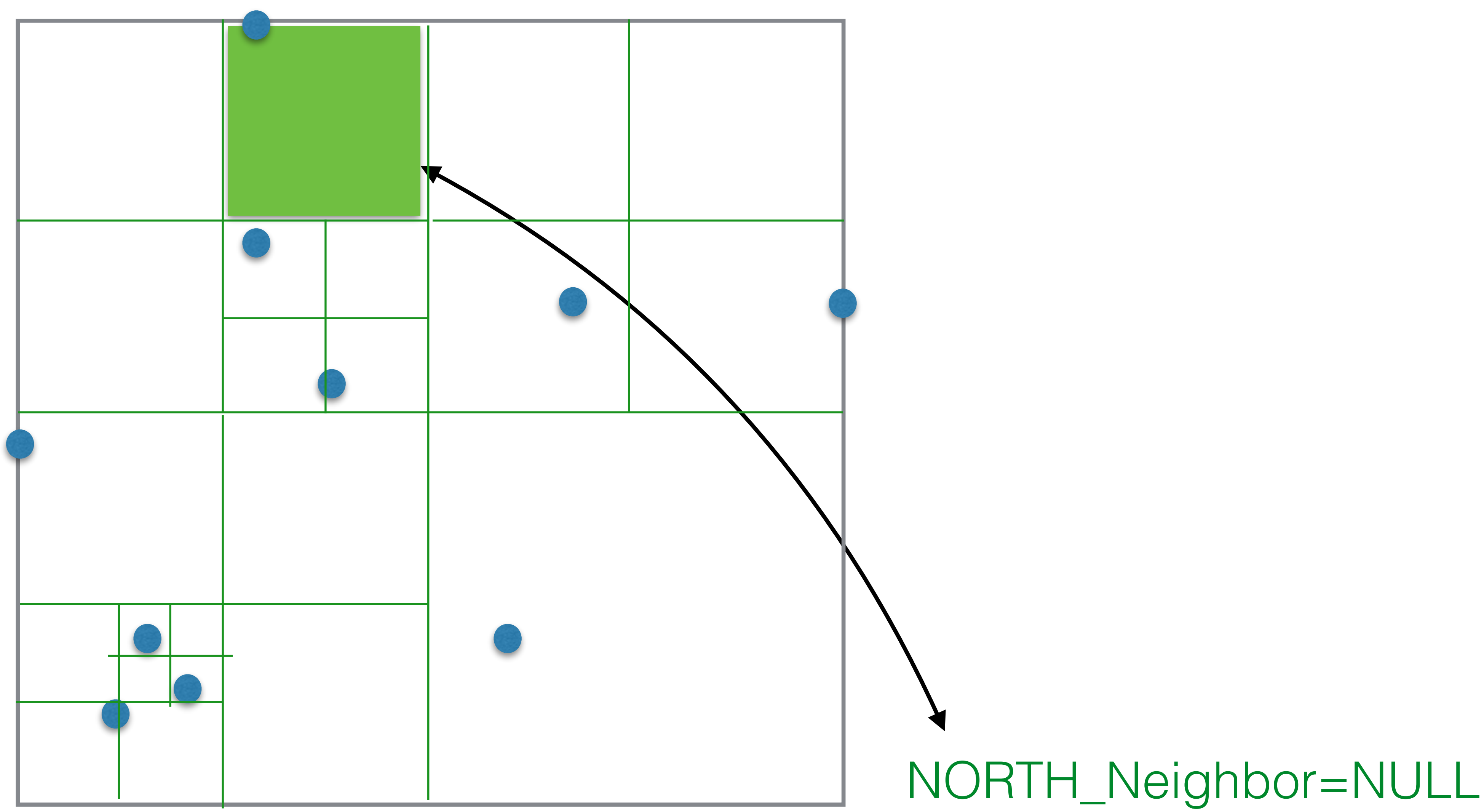
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

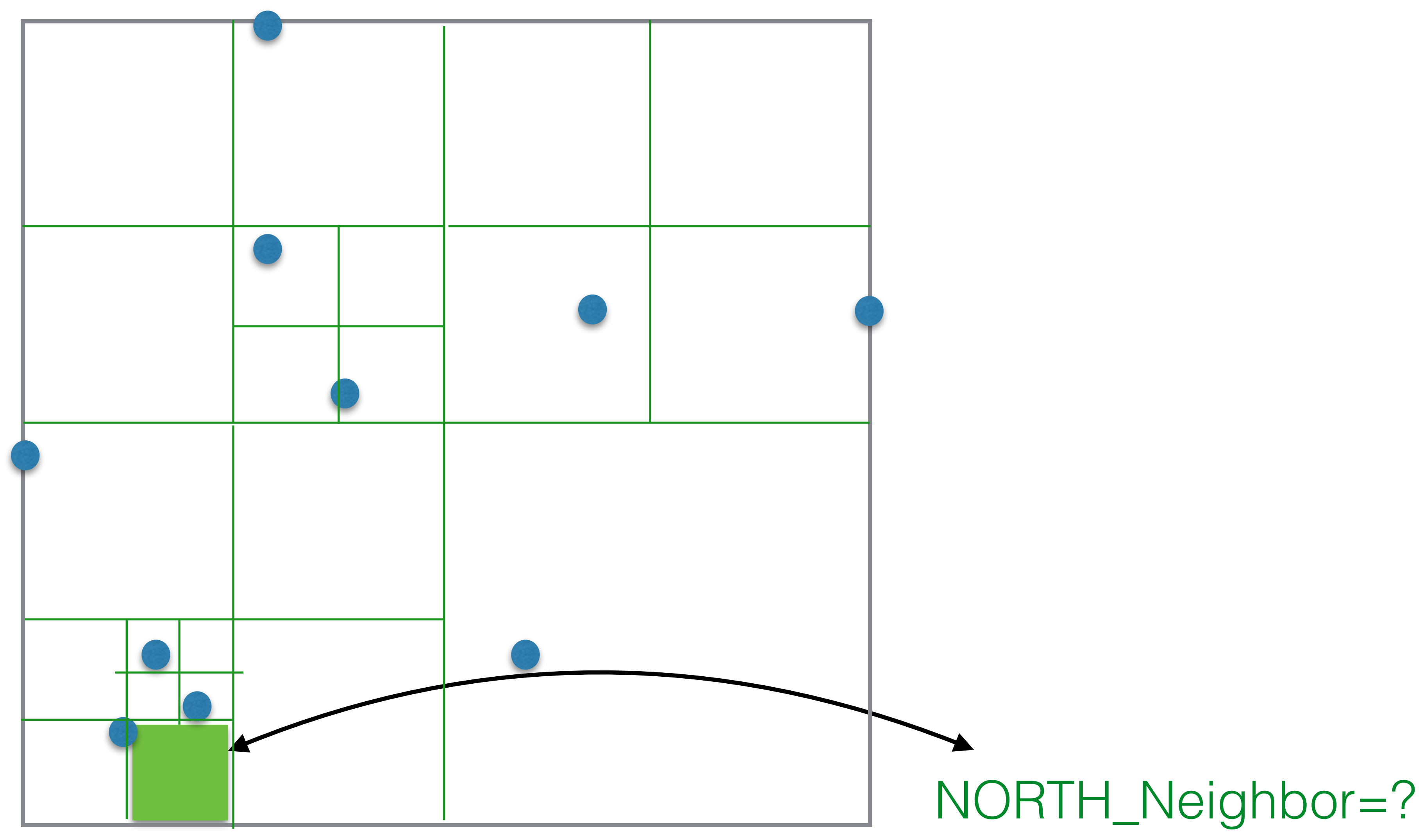
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

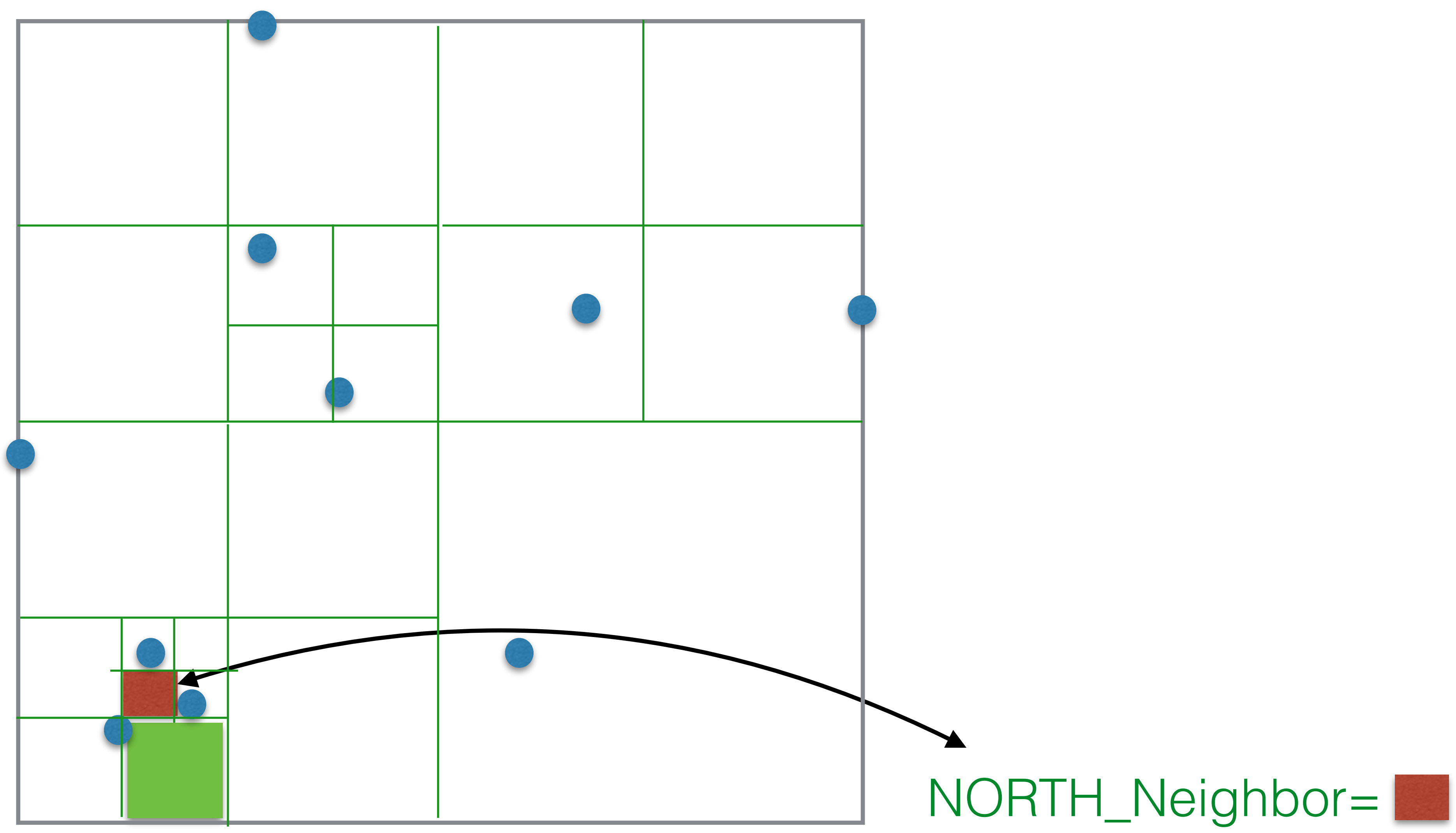
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

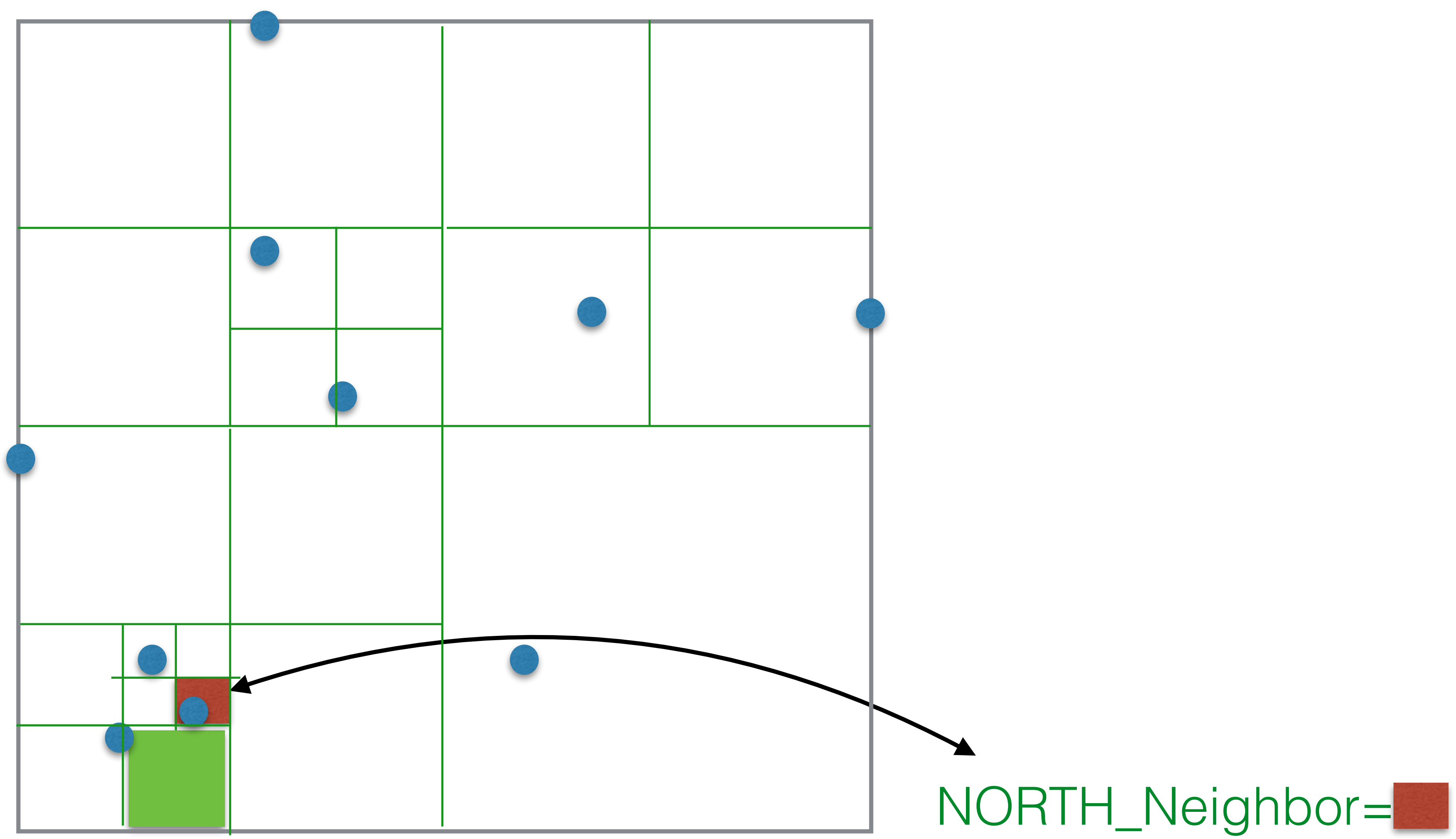
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

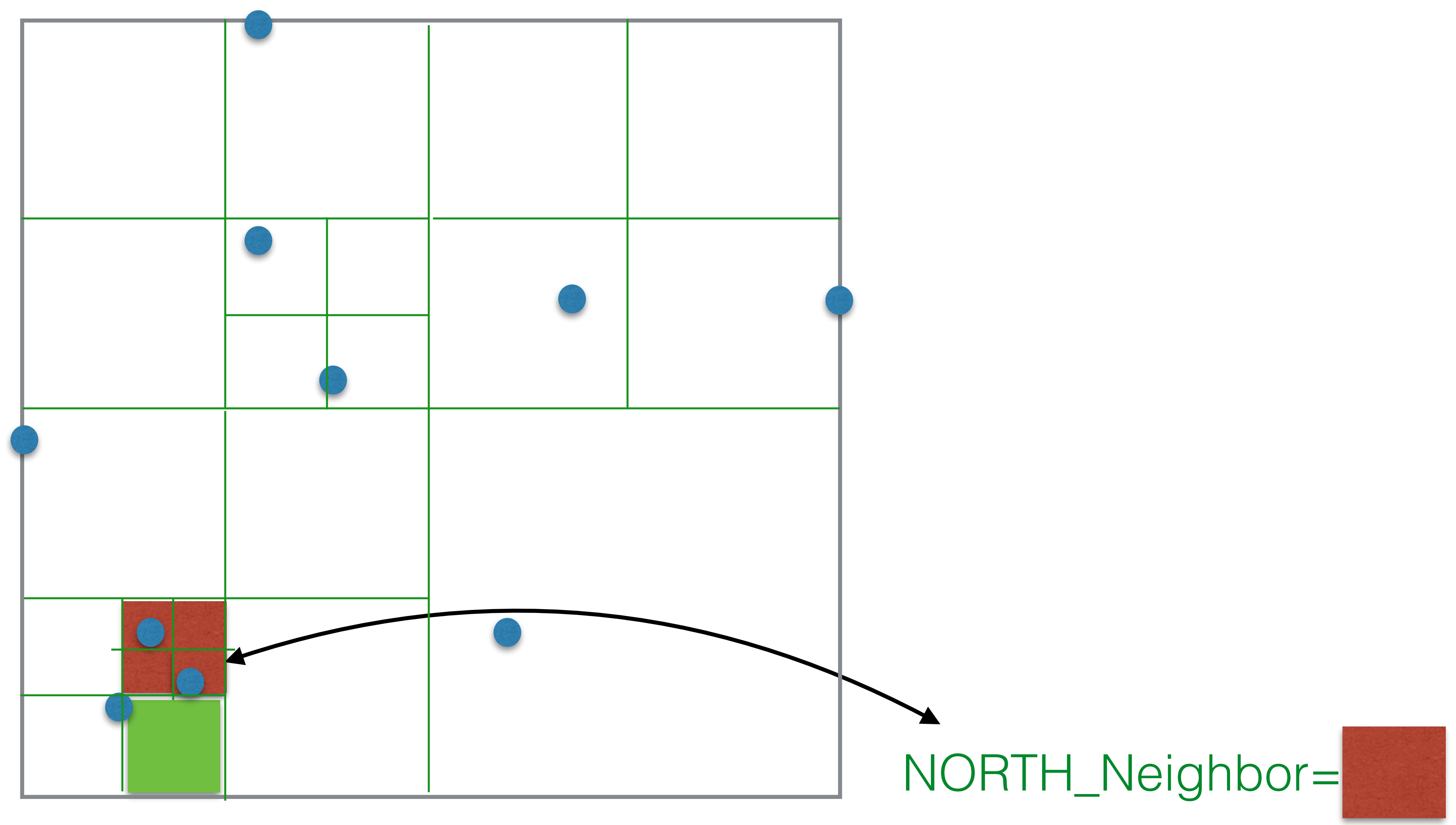
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

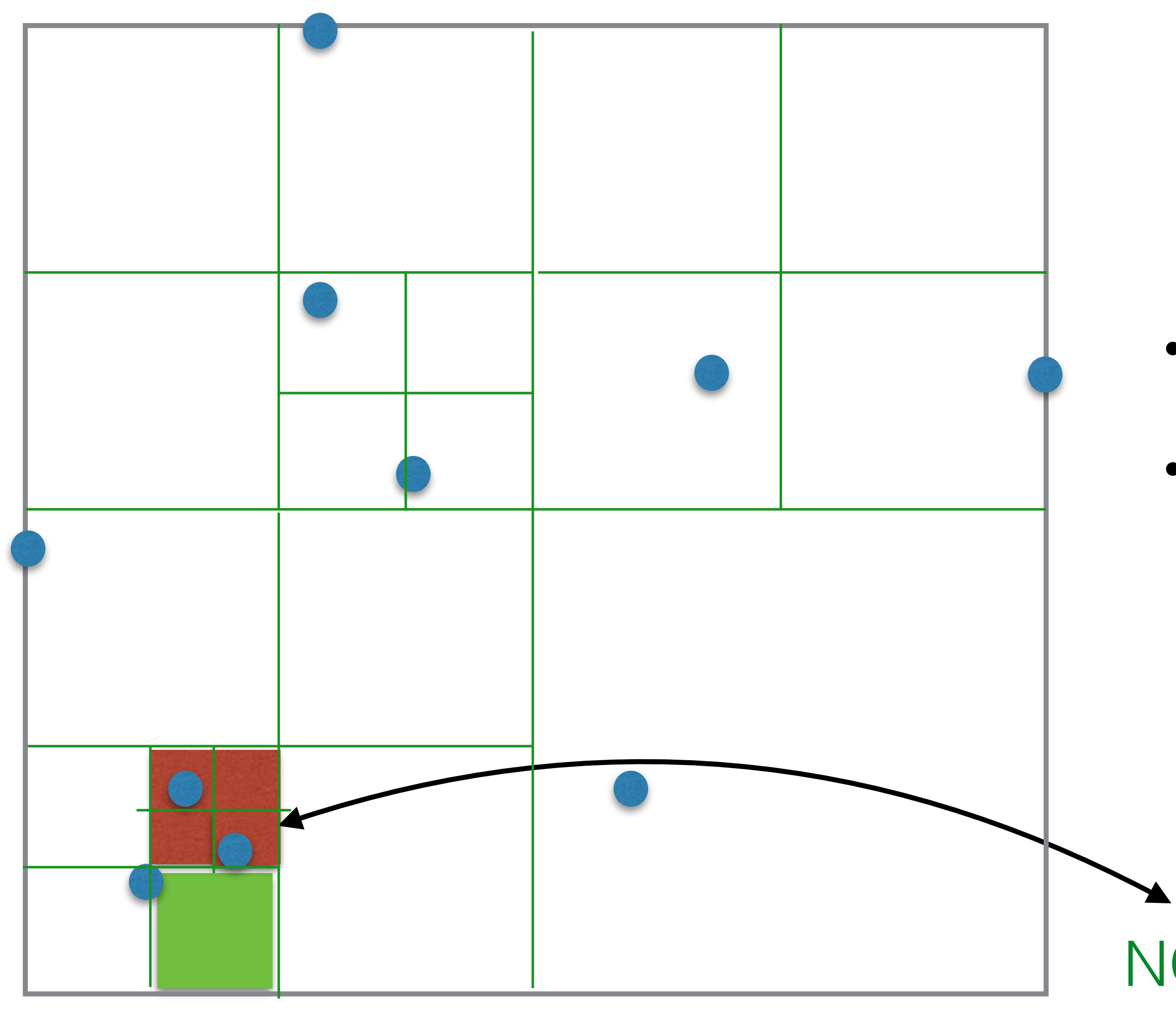
- two regions (squares) are adjacent iff they share an edge



Example: Neighbor finding

Given a node v and a direction (N, S, E, W) find a node v' such that region(v') is adjacent to region(v) in the given direction.

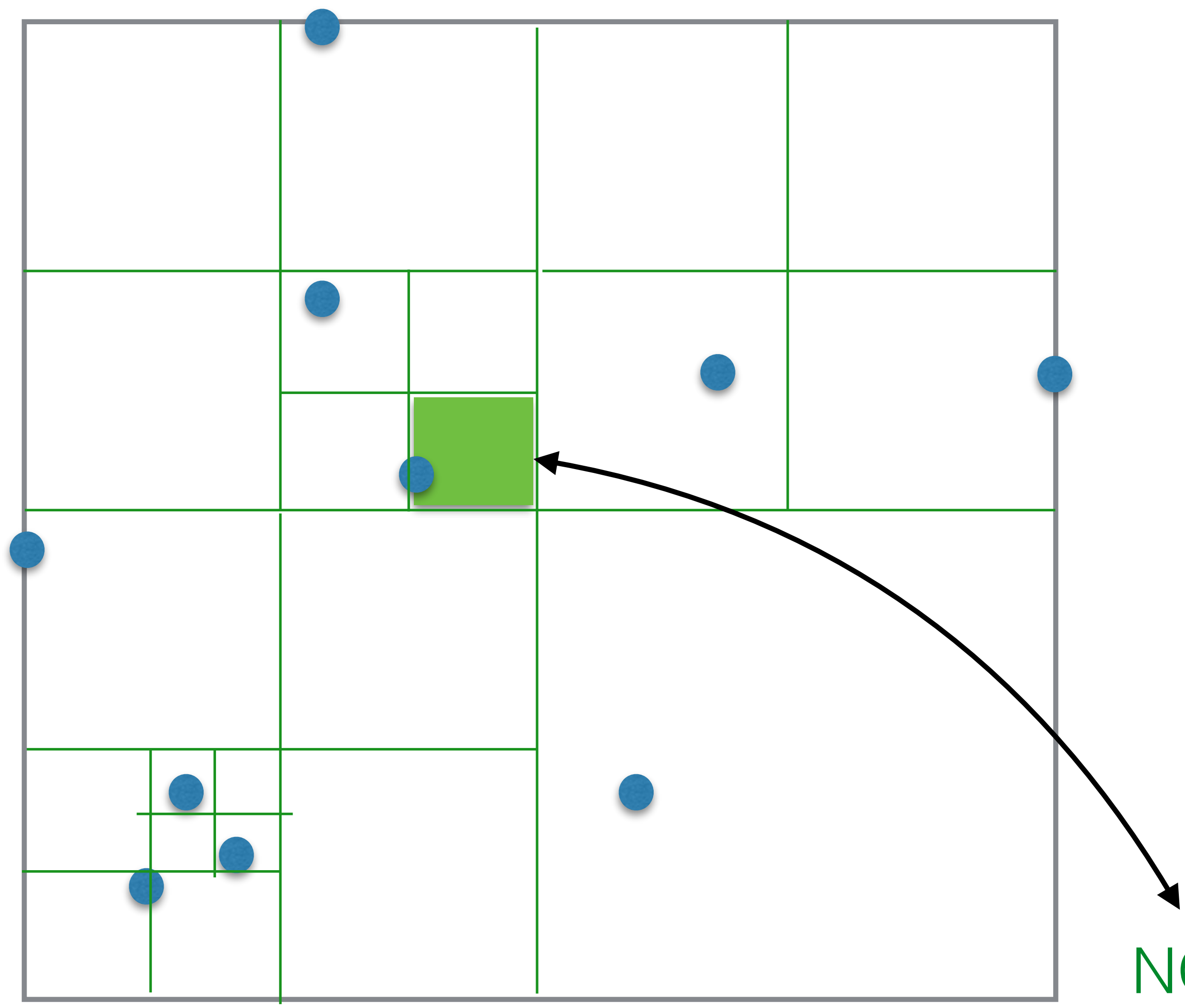
- two regions (squares) are adjacent iff they share an edge

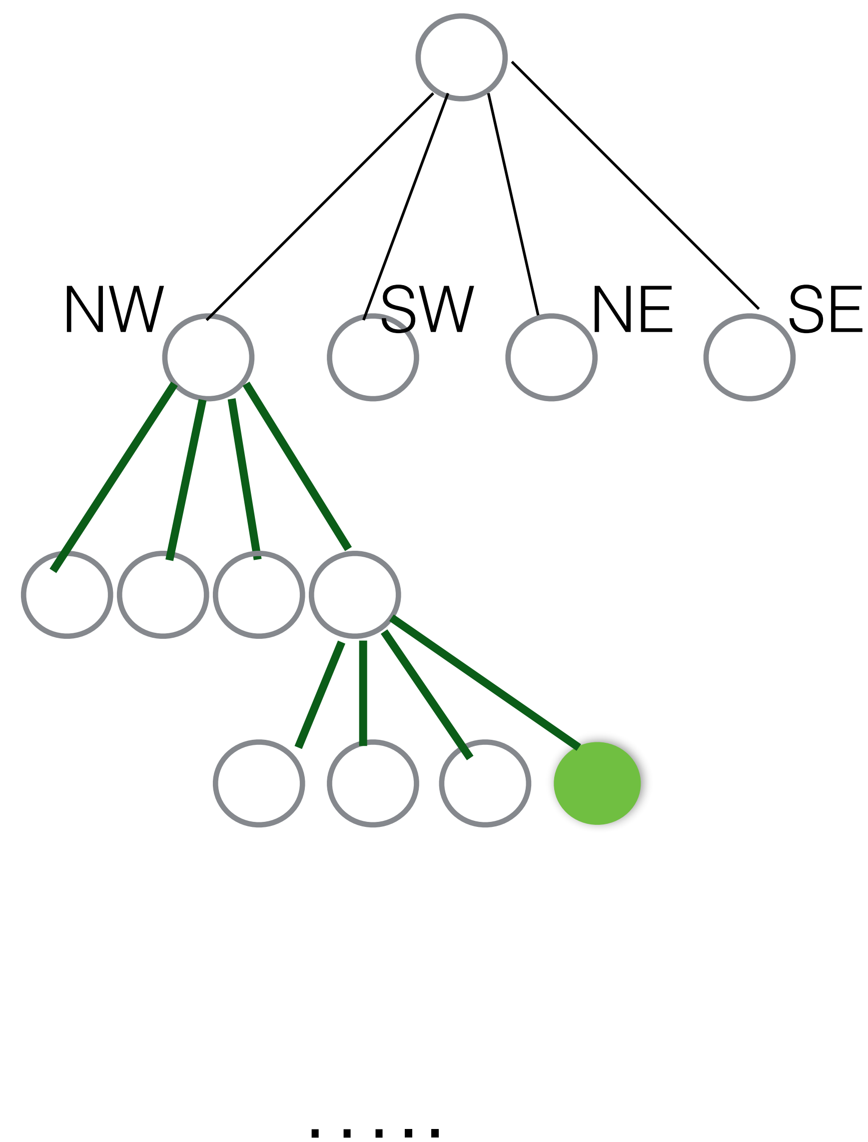


- try to find a node v' at the same depth as v
- if not possible, find the deepest

NORTH_Neighbor=

Visualizing it on the tree..

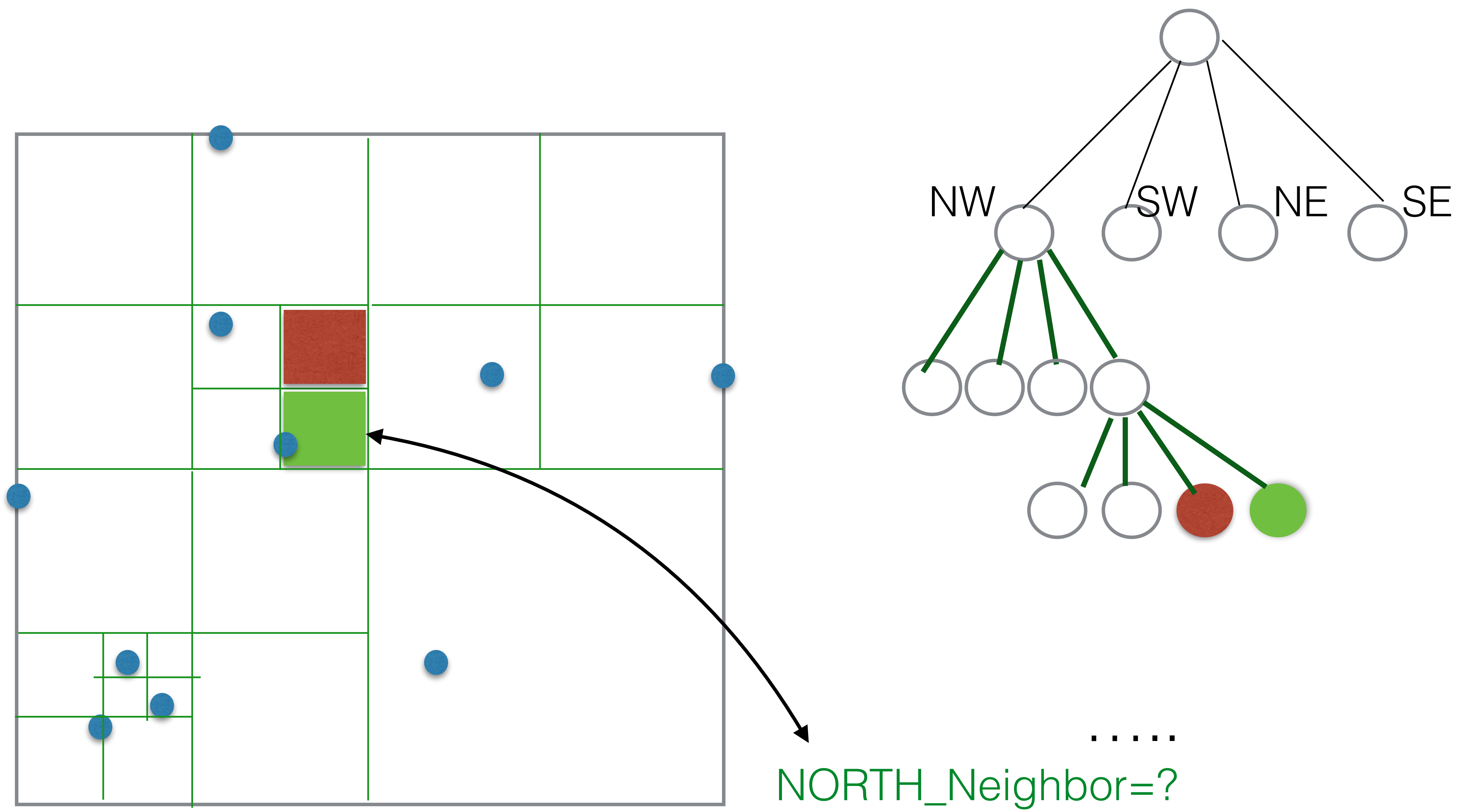




NORTH_Neighbor=?

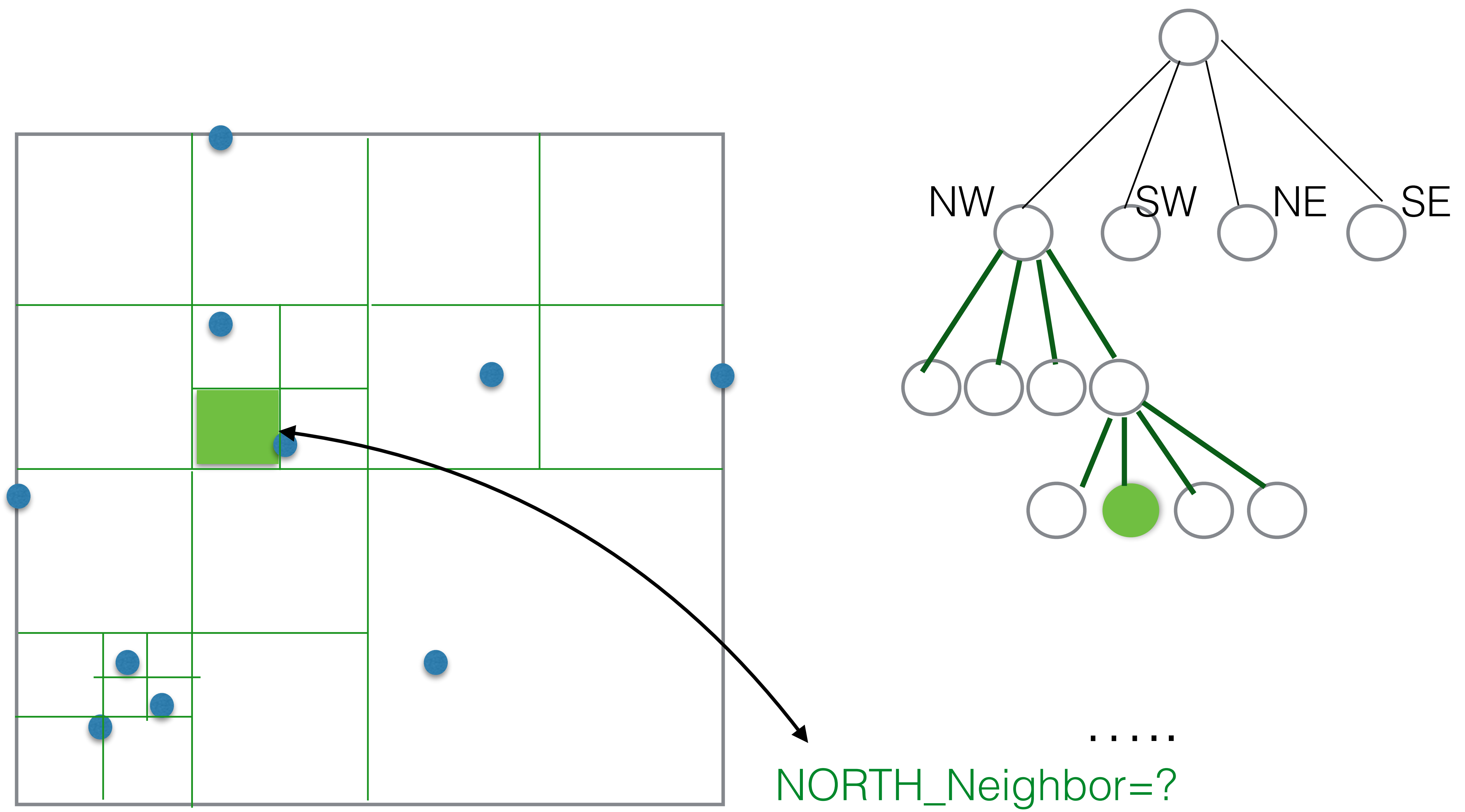
- try to find a node v' at the same depth as v
- if not possible, find the deepest

Visualizing it on the tree..



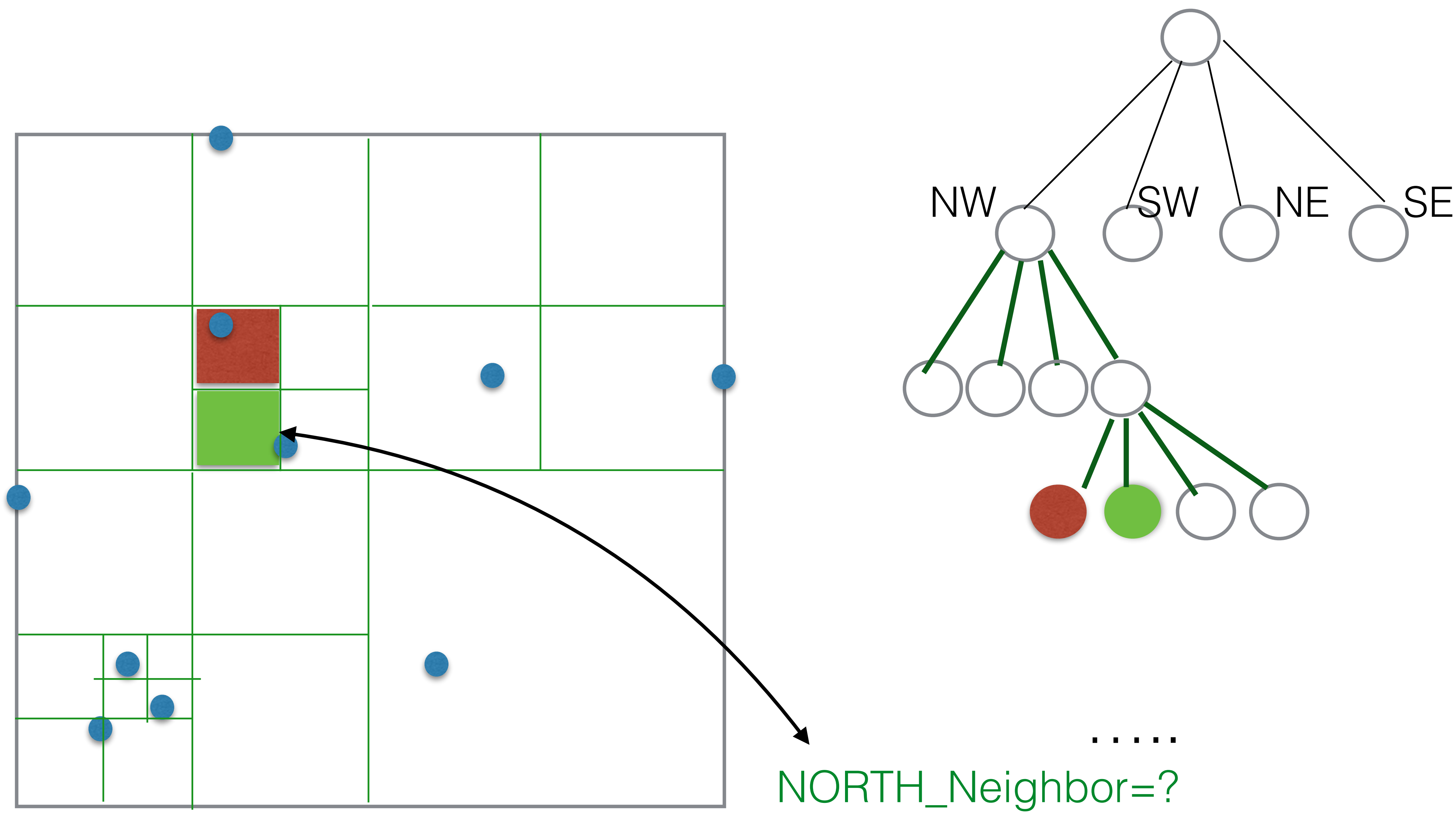
- try to find a node v’ at the same depth as v
- if not possible, find the deepest

Visualizing it on the tree..



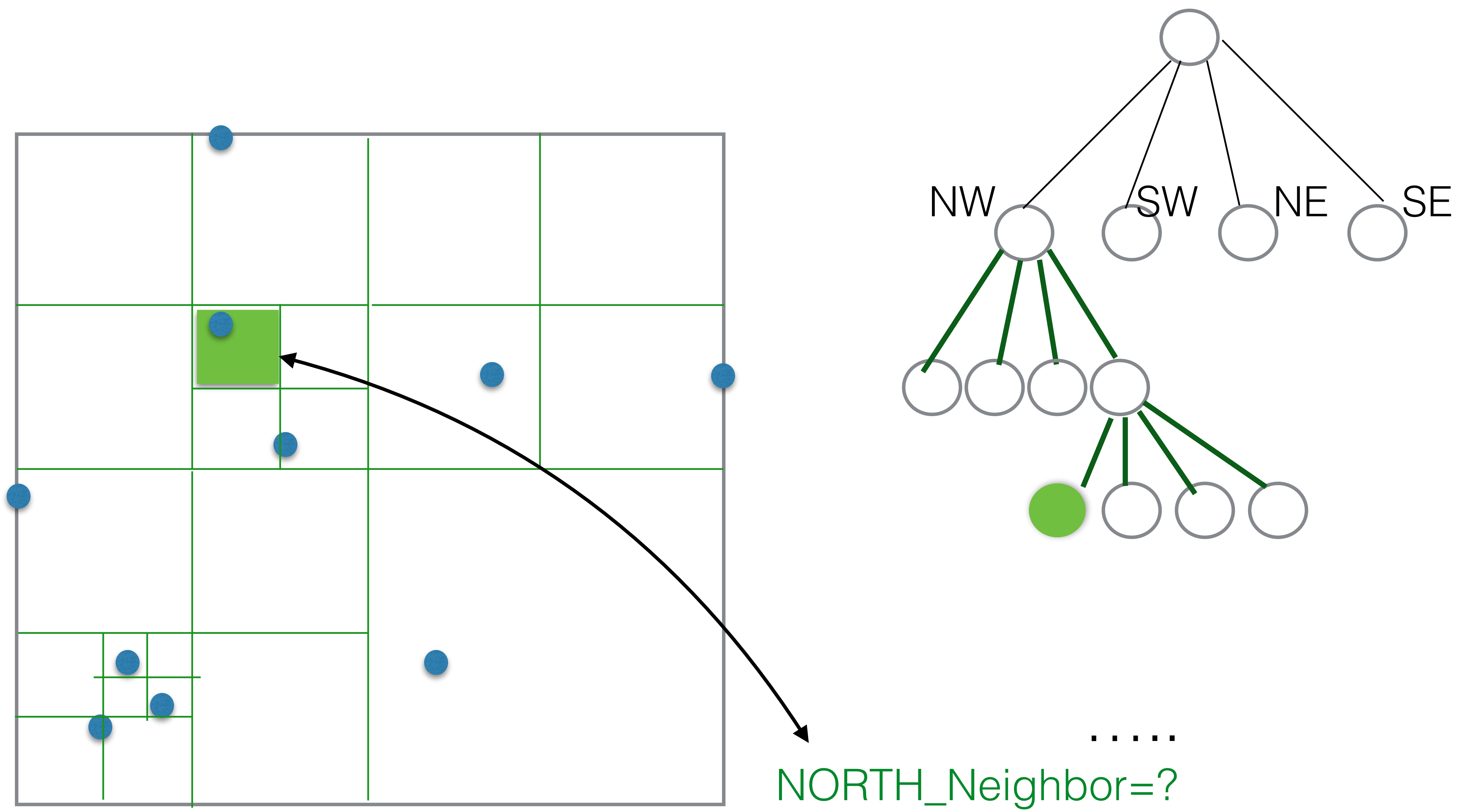
- try to find a node v' at the same depth as v
- if not possible, find the deepest

Visualizing it on the tree..



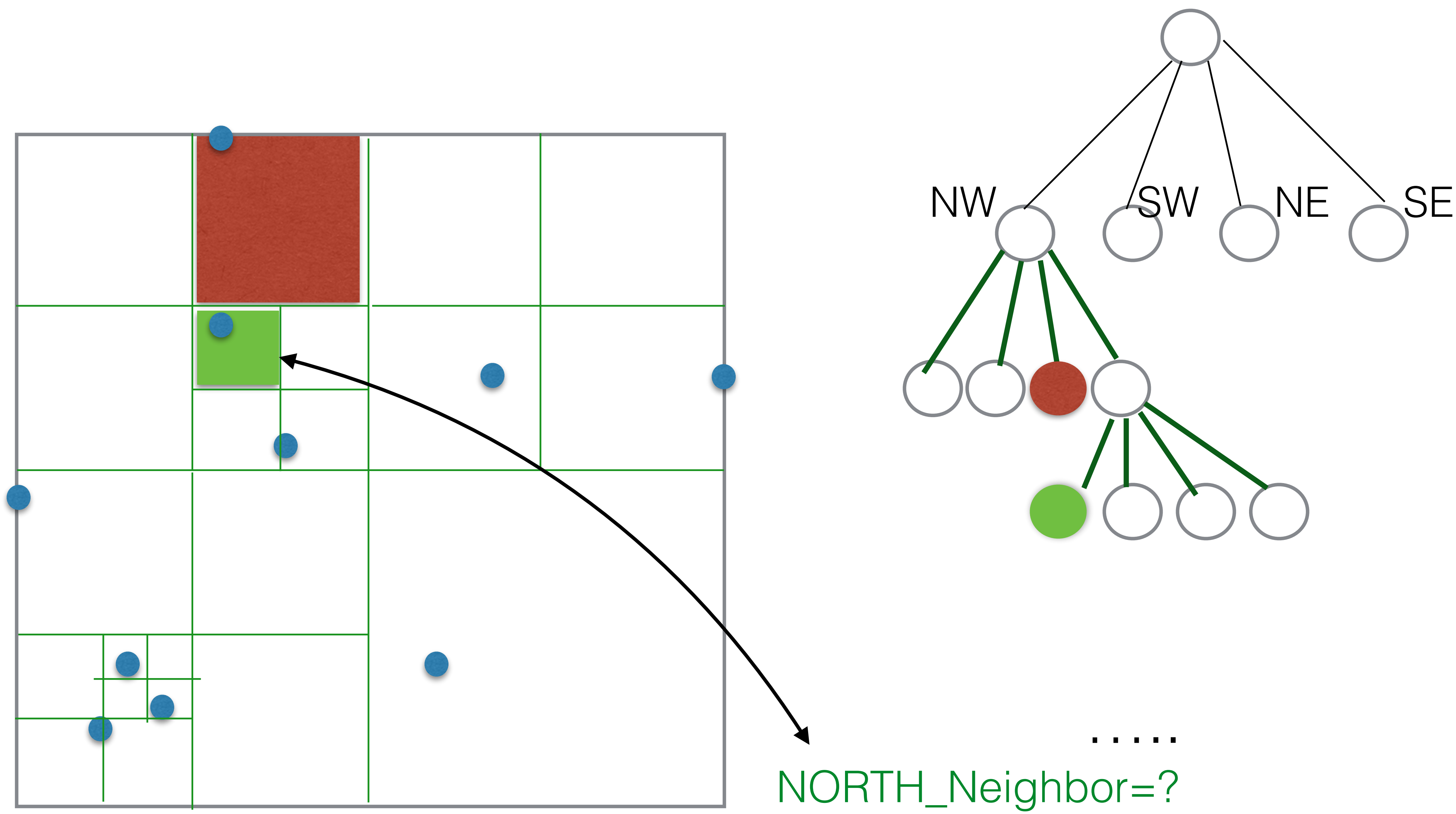
- try to find a node v’ at the same depth as v
- if not possible, find the deepest

Visualizing it on the tree..



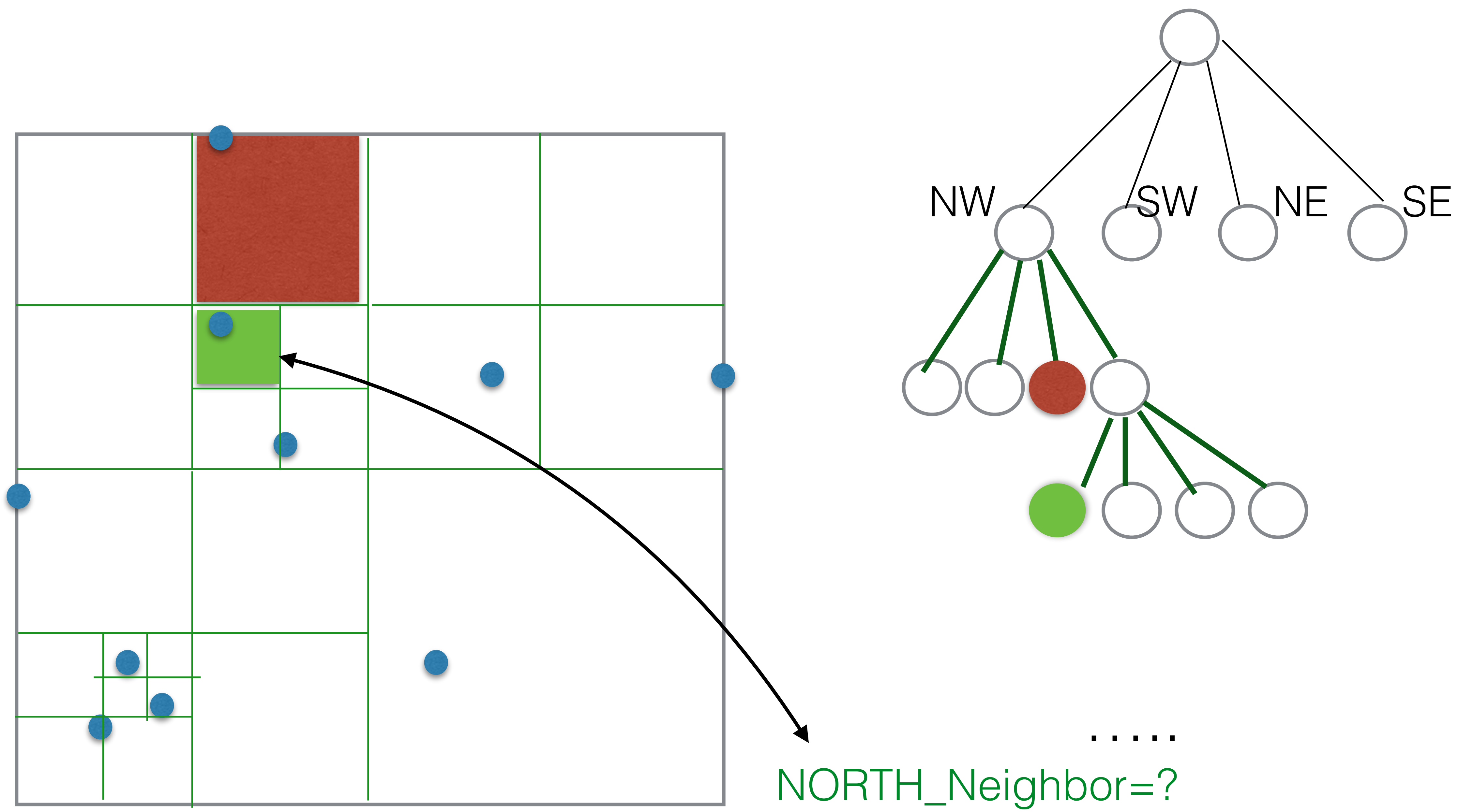
- try to find a node v’ at the same depth as v
- if not possible, find the deepest

Visualizing it on the tree..



- try to find a node v' at the same depth as v
- if not possible, find the deepest

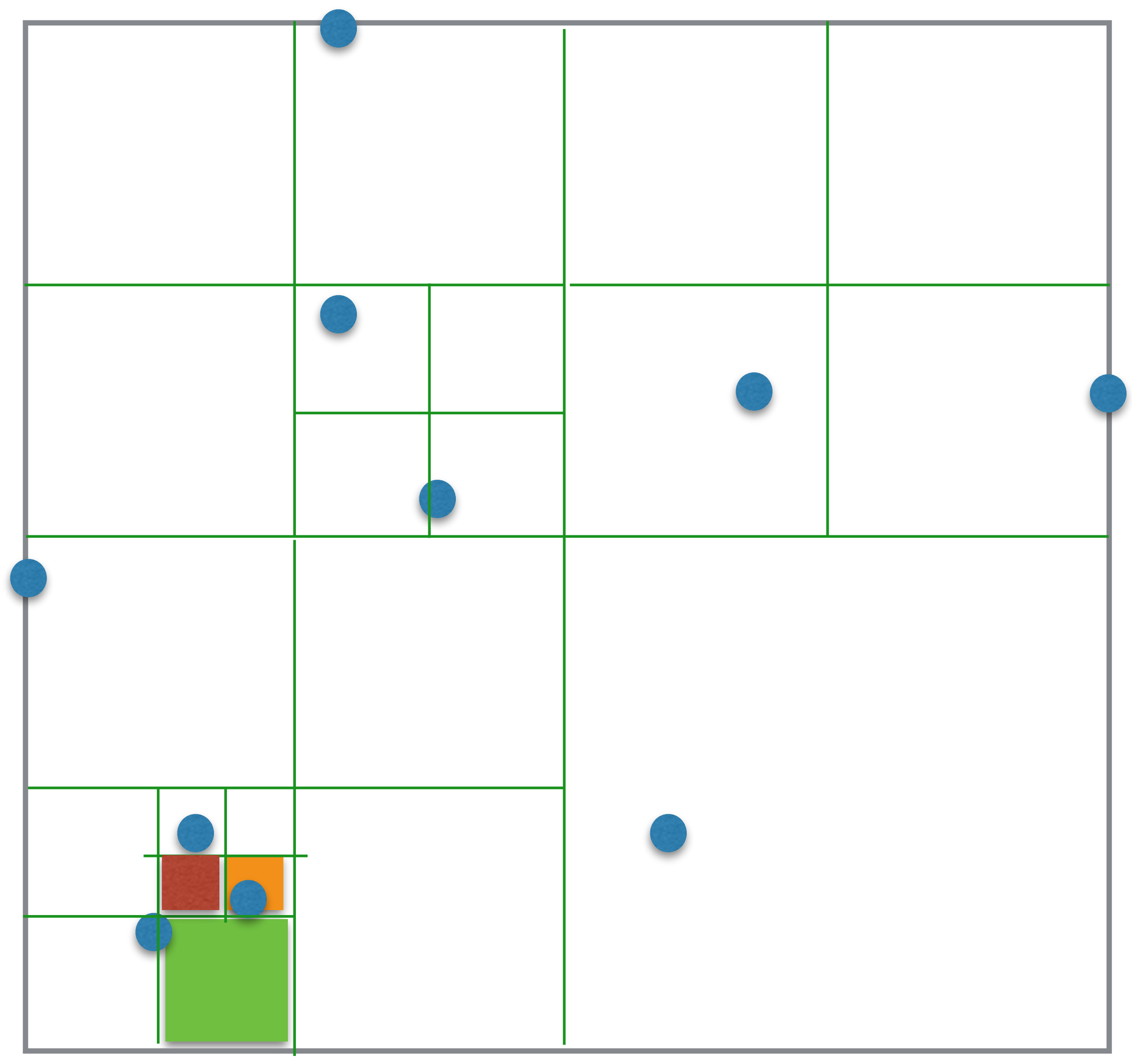
Visualizing it on the tree..

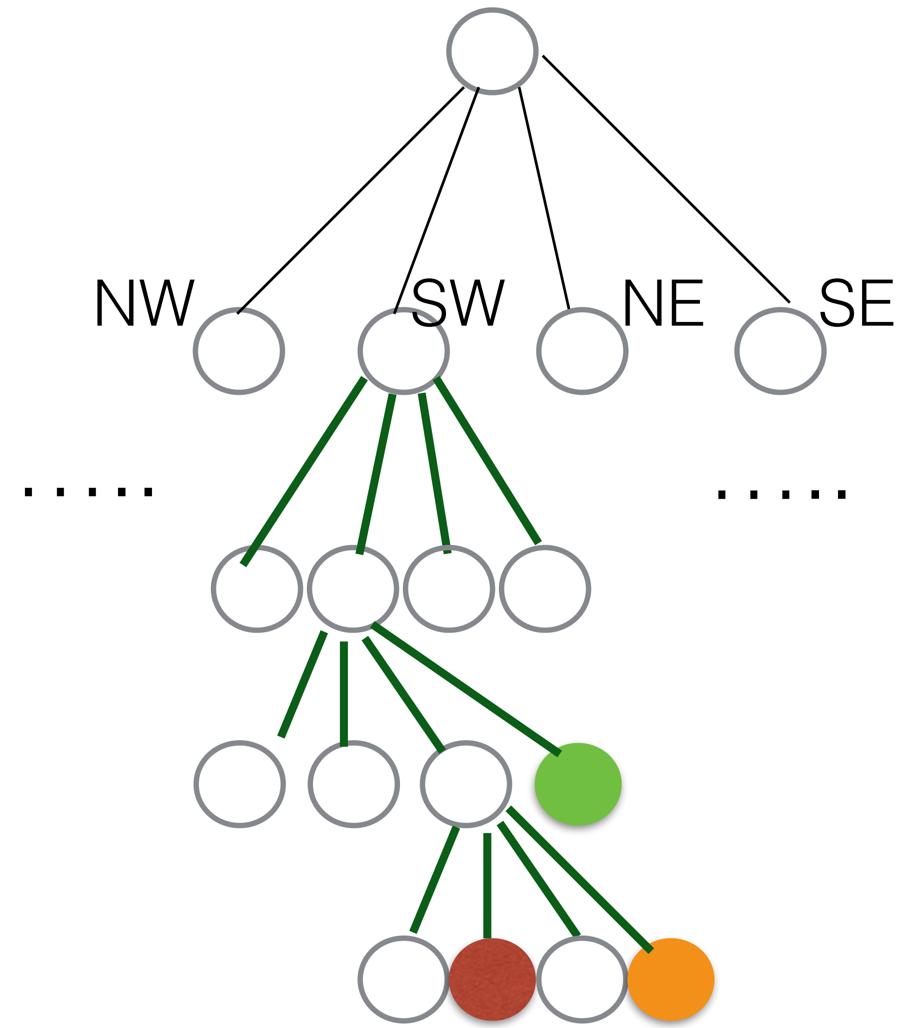


- try to find a node v' at the same depth as v
- if not possible, find the deepest

Is the North_neighbor always a sibling or an uncle?

Visualizing it on the tree..

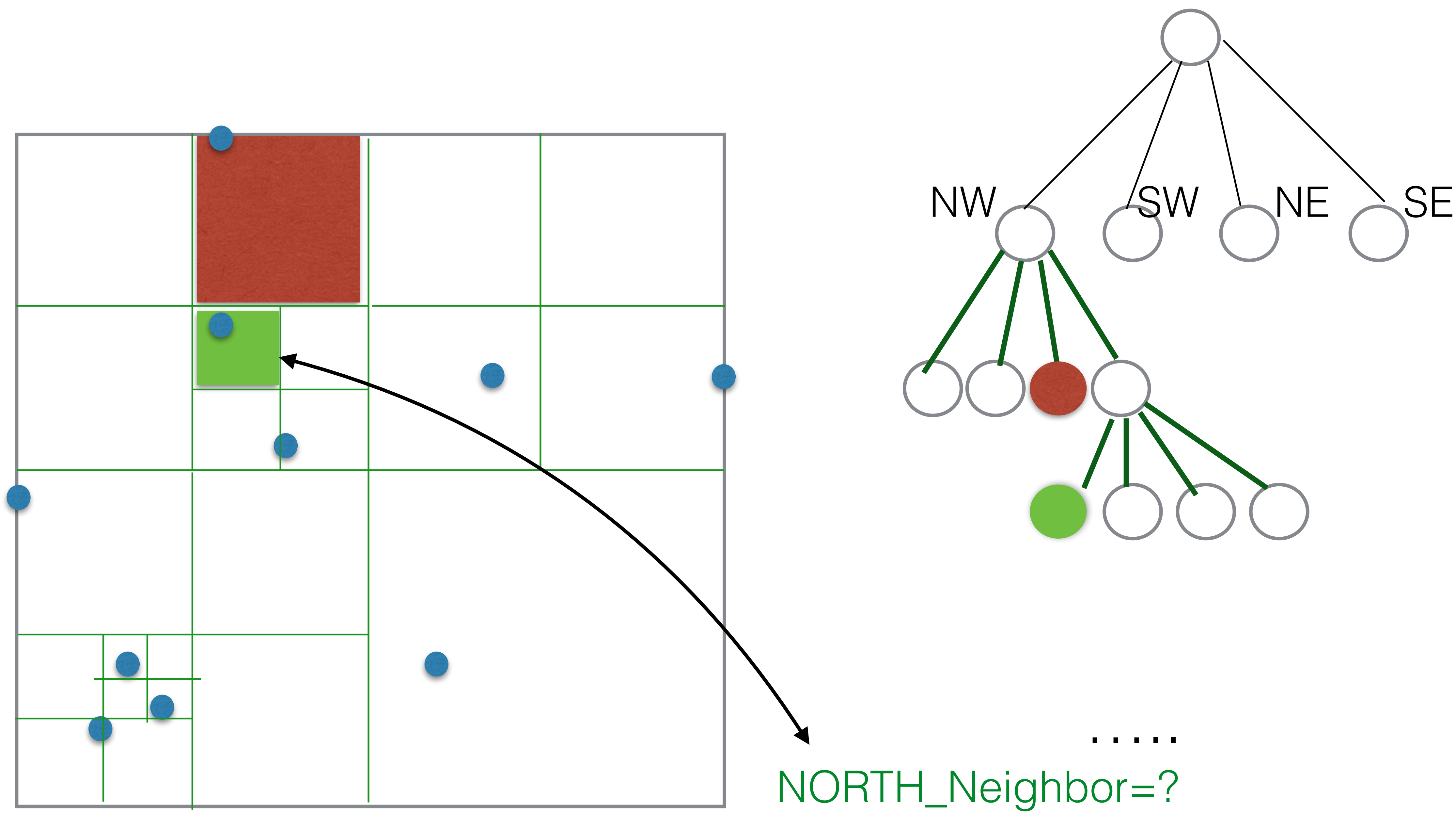




- try to find a node v’ at the same depth as v
- if not possible, find the deepest

Could be a nephew/niece, but we prefer the sibling..

Visualizing it on the tree..



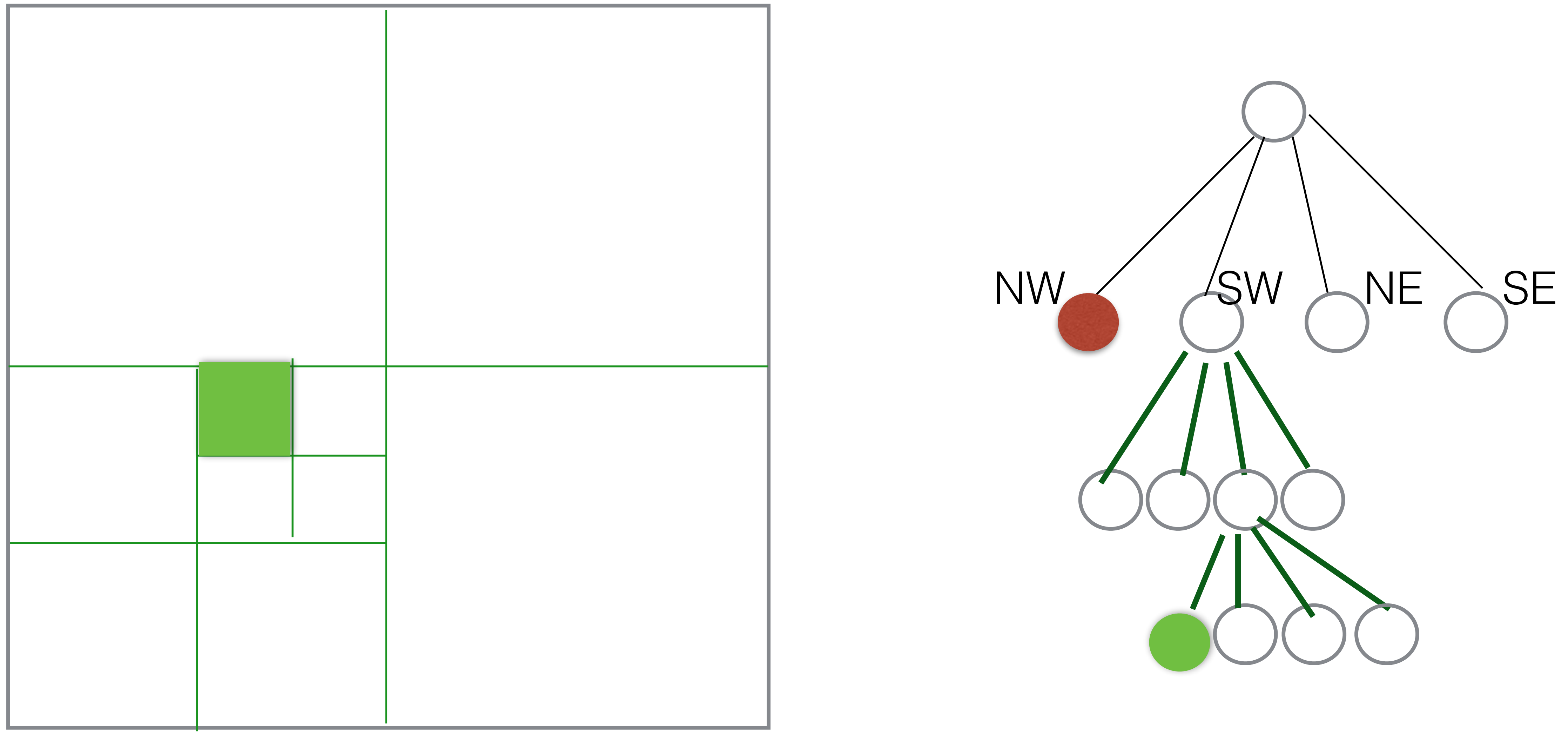
- try to find a node v’ at the same depth as v
- if not possible, find the deepest

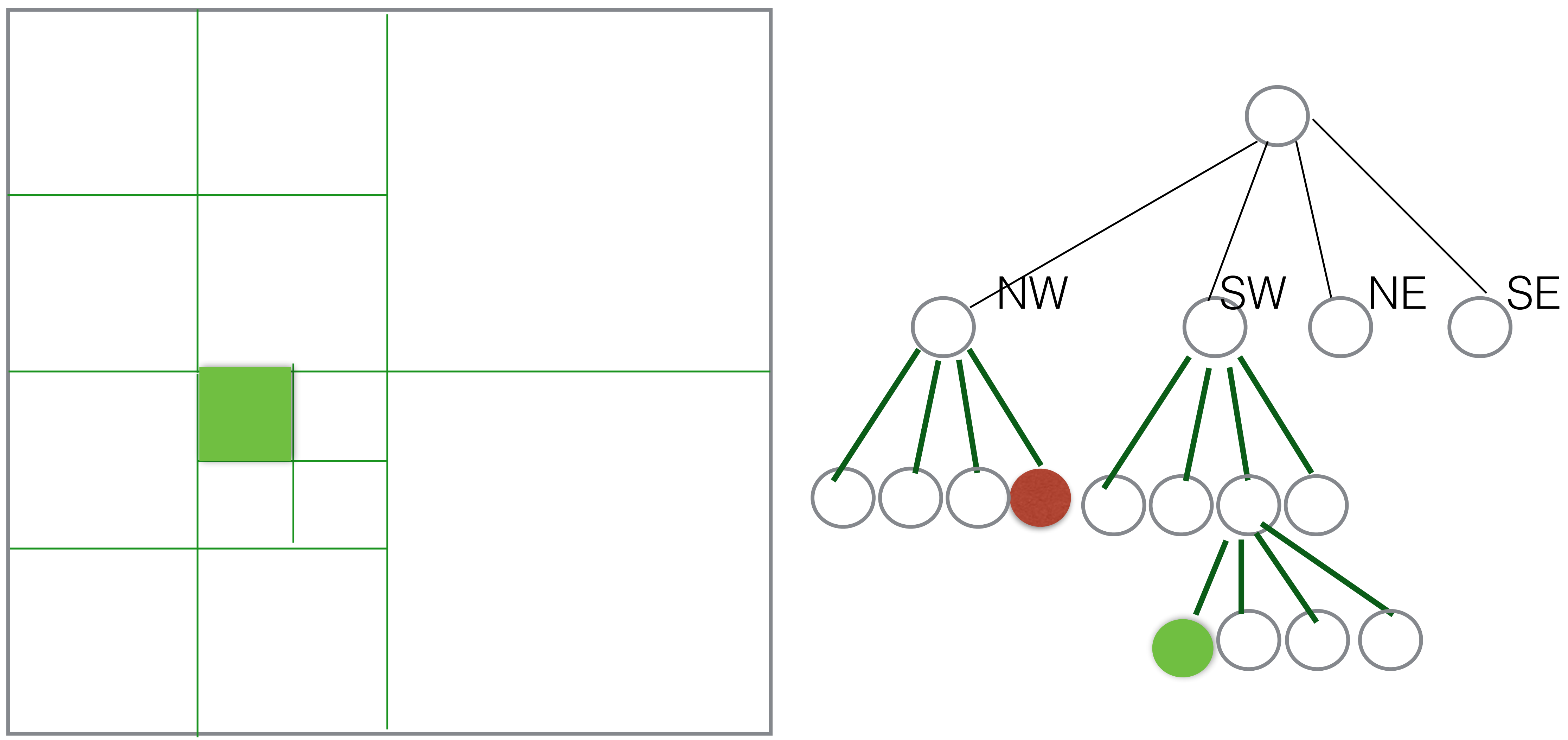
Come up with an example where the search for a North_neighbor is a great-uncle

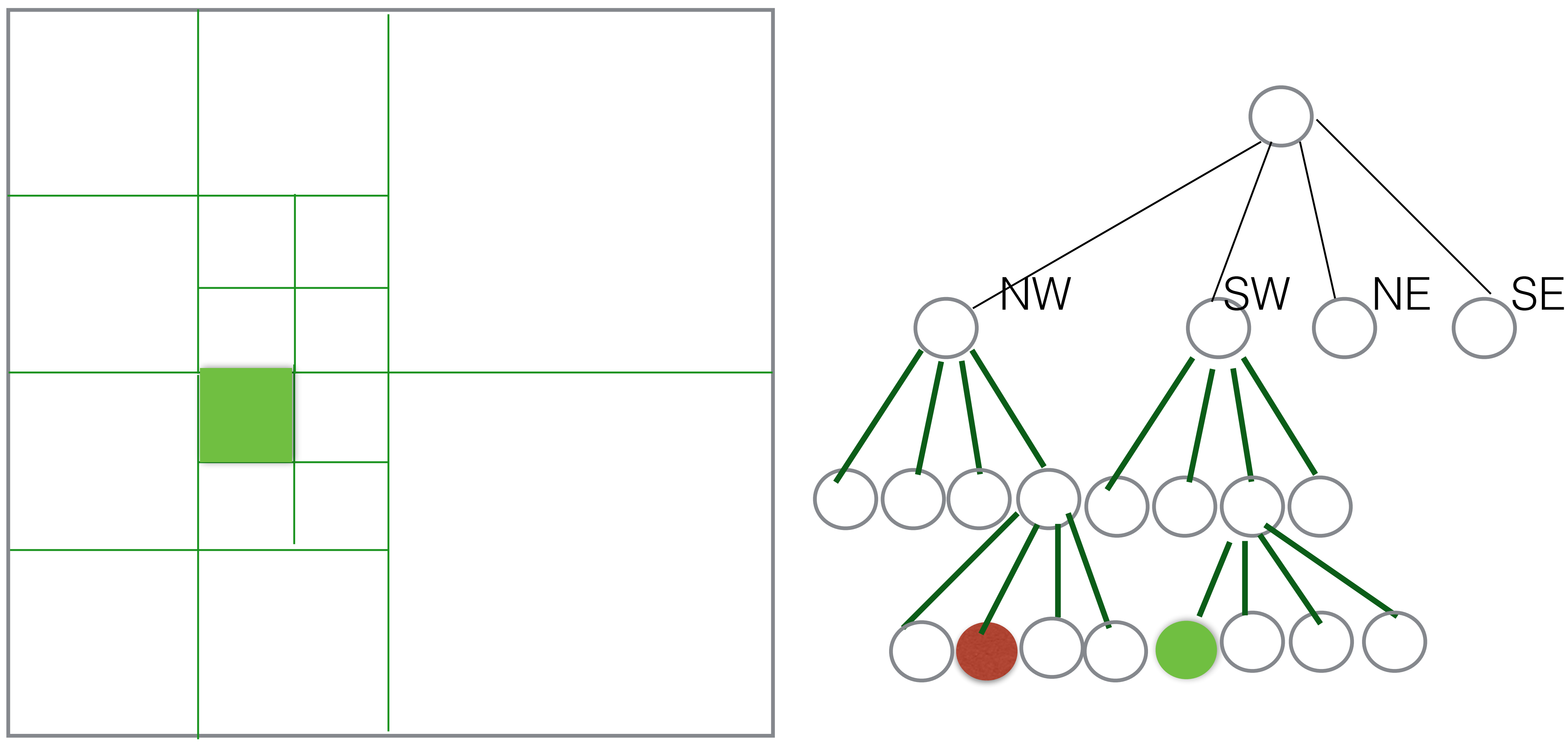
Example: Neighbor finding

Come up with an example where the North_neighbor is a

- great-uncle.
- great-great-uncle
- …


NW
SW
NE
SE


NW
SW
NE
SE


NW
SW
NE
SE

Example: Neighbor finding

//input: a node v in a quadtree

//output: the deepest node v' whose depth is at most the depth of v such that region(v') is a north-neighbor of region(v), and NULL if there is no such node

North_Neighbor(v)

- if v==root: …
- if v==SW-child of parent(v):…
- if v==SE-child of parent(v): …

//if we reached here, v must be NW or NE child

- x <—- North_Neighbor(parent(v))
  - if x is NULL or a leaf:
    - ….
  - else:
    - …..

Example: Neighbor finding

//input: a node v in a quadtree

//output: the deepest node v' whose depth is at most the depth of v such that region(v') is a north-neighbor of region(v), and NULL if there is no such node

North_Neighbor(v)

- if v==root: return NULL
- if v==SW-child of parent(v): return NW-child of parent(v)
- if v==SE-child of parent(v): return NE-child of parent(v)

  //if we reached here, v must be NW or NE child
- x <—- North_Neighbor(parent(v))
  - if x is NULL or a leaf: return x
  - else:
    - if v ==NW-child of parent(v): return SW-child(x)
    - else: return SE-child(x)

Example: Neighbor finding

//input: a node v in a quadtree

//output: the deepest node v' whose depth is at most the depth of v such that region(v') is a north-neighbor of region(v), and NULL if there is no such node

North_Neighbor(v)

- if v==root: return NULL
- if v==SW-child of parent(v): return NW-child of parent(v)
- if v==SE-child of parent(v): return NE-child of parent(v)

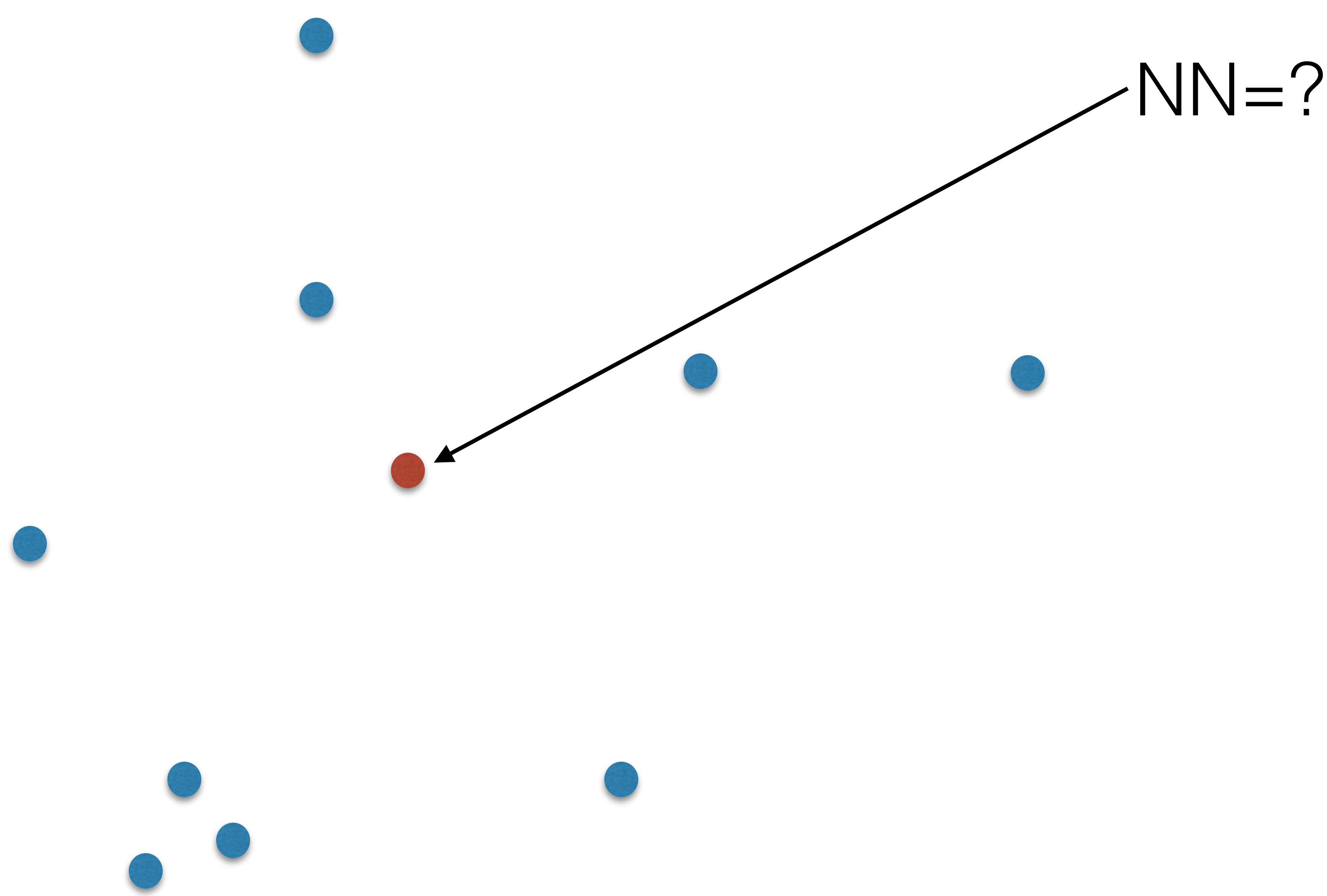
//if we reached here, v must be NW or NE child

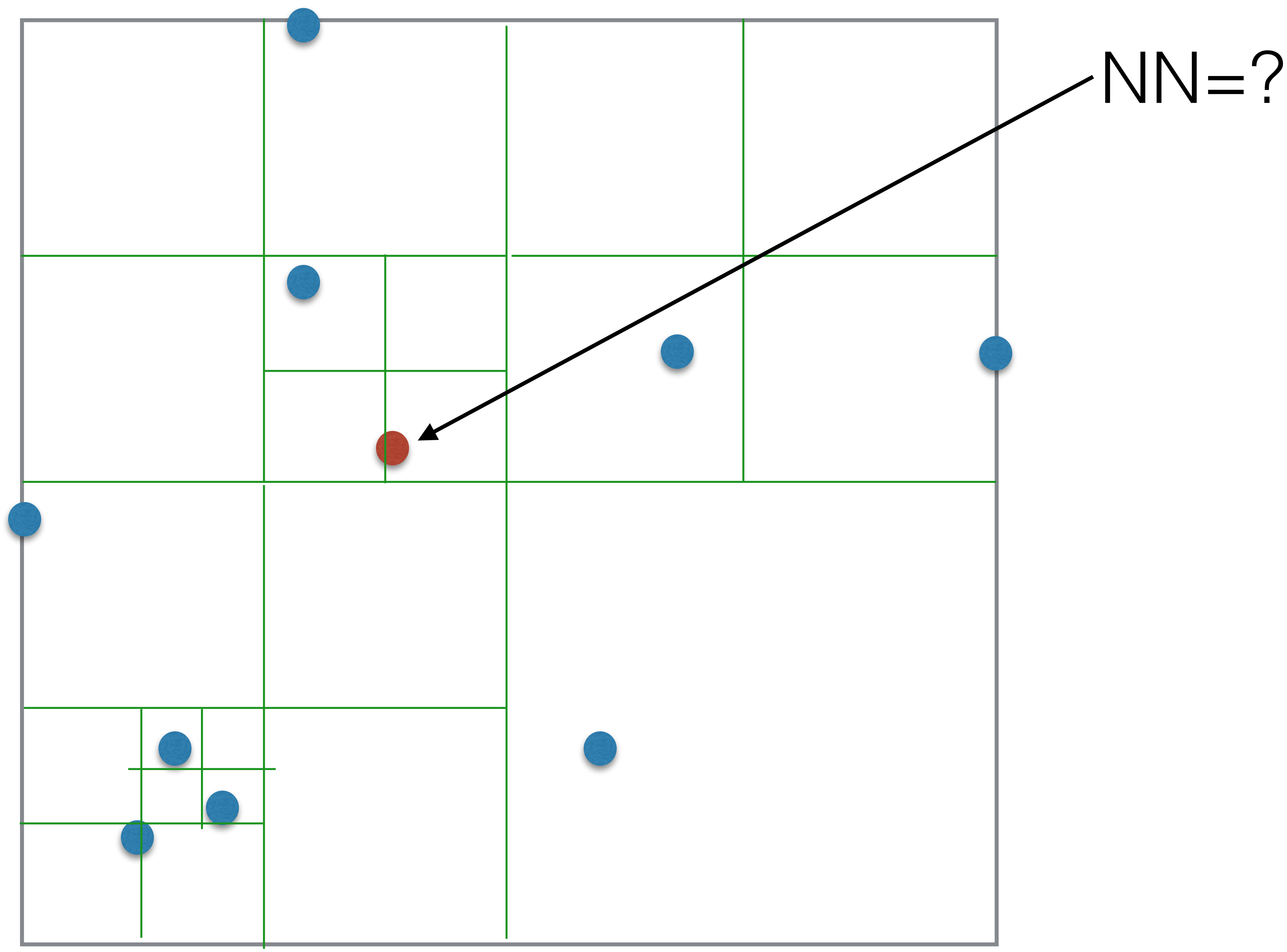
- x <—- North_Neighbor(parent(v)) ← give an example that would trigger several recursive calls
    - if x is NULL or a leaf: return x
    - else:
        - if v ==NW-child of parent(v): return SW-child(x)
        - else: return SE-child(x)

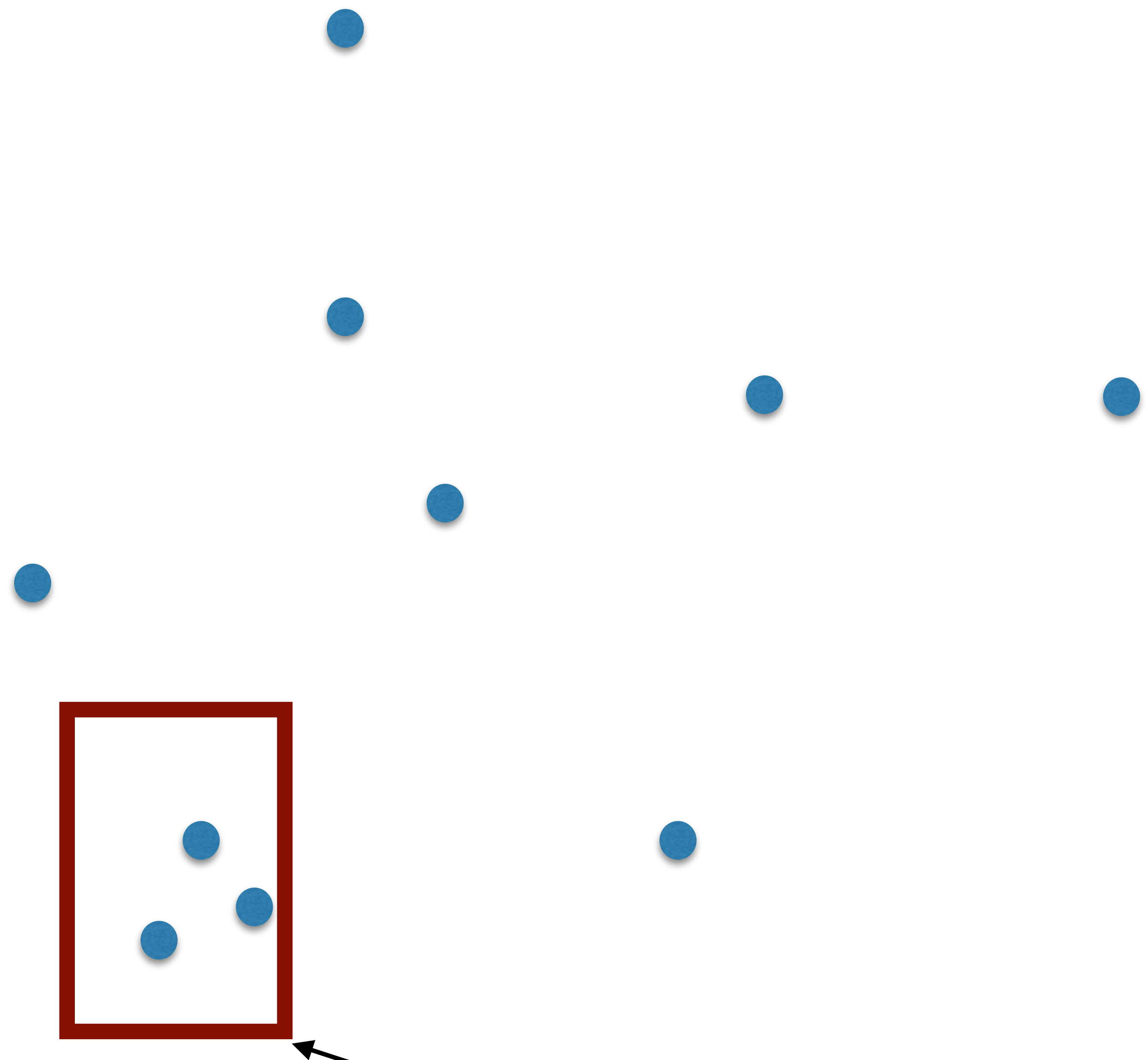
More applications

- Used to answer queries on spatial data such as:
    - point location
    - nearest neighbor (NN)
    - k-NNs
    - range searching
    - find all segments intersecting a given segment
    - meshing
    - …

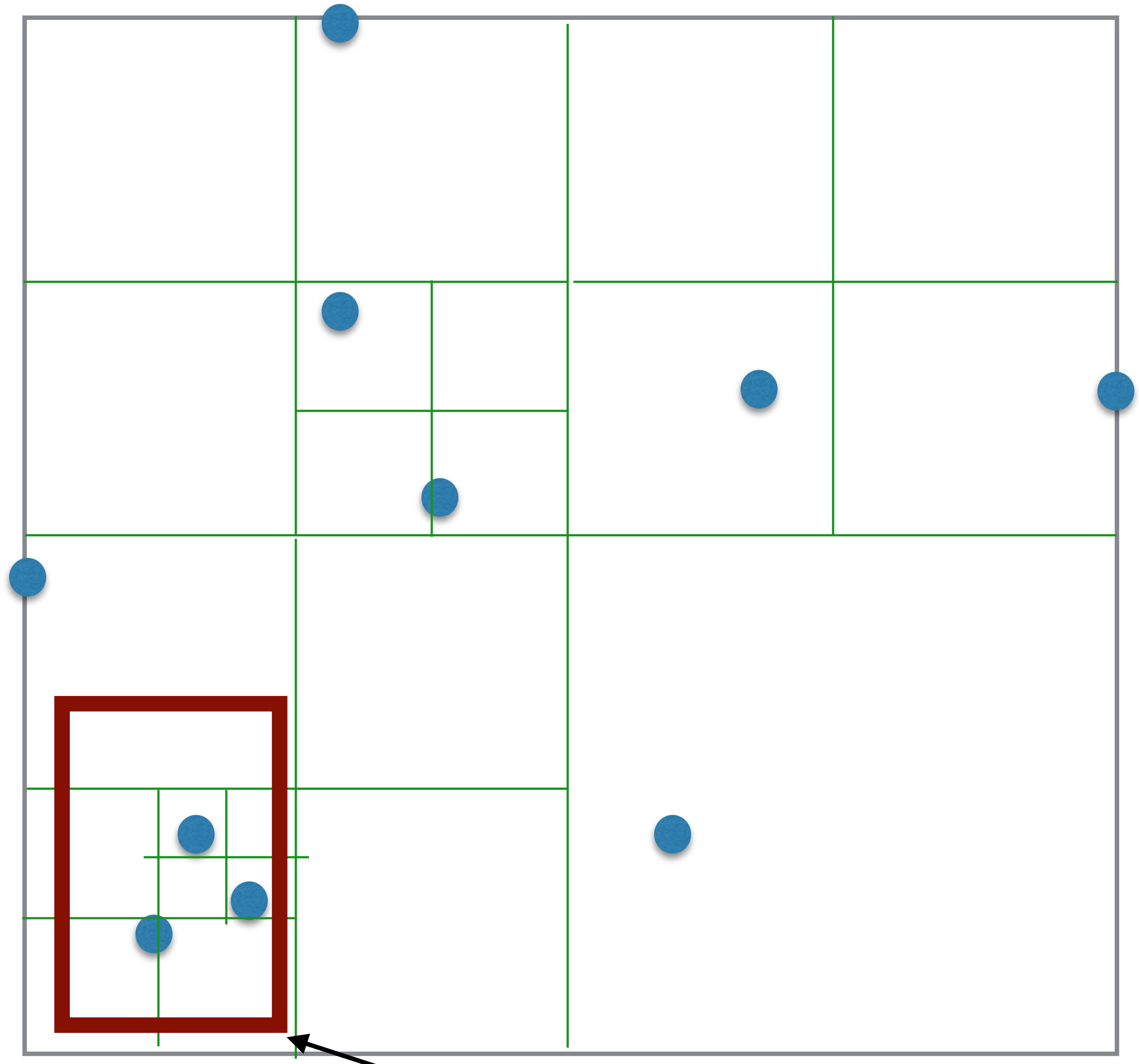
How would you do these?


NN=?


NN=?



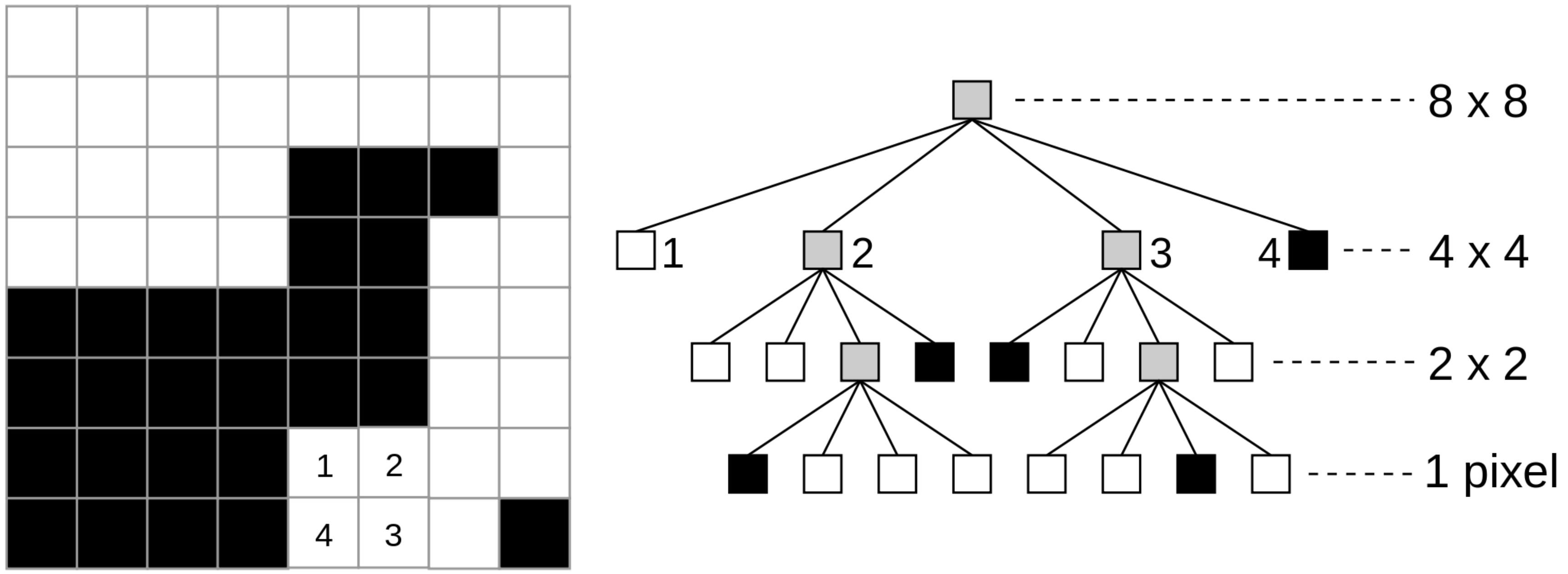
find all points in this range



find all points in this range

Applications

- Image analysis/compression



Applications

- Used for fast rendering (LOD)
  - Level i in the qdt —> scene at a certain resolution
  - bottom level has full resolution
  - render scene at a resolution dependent on its distance from the viewpoint

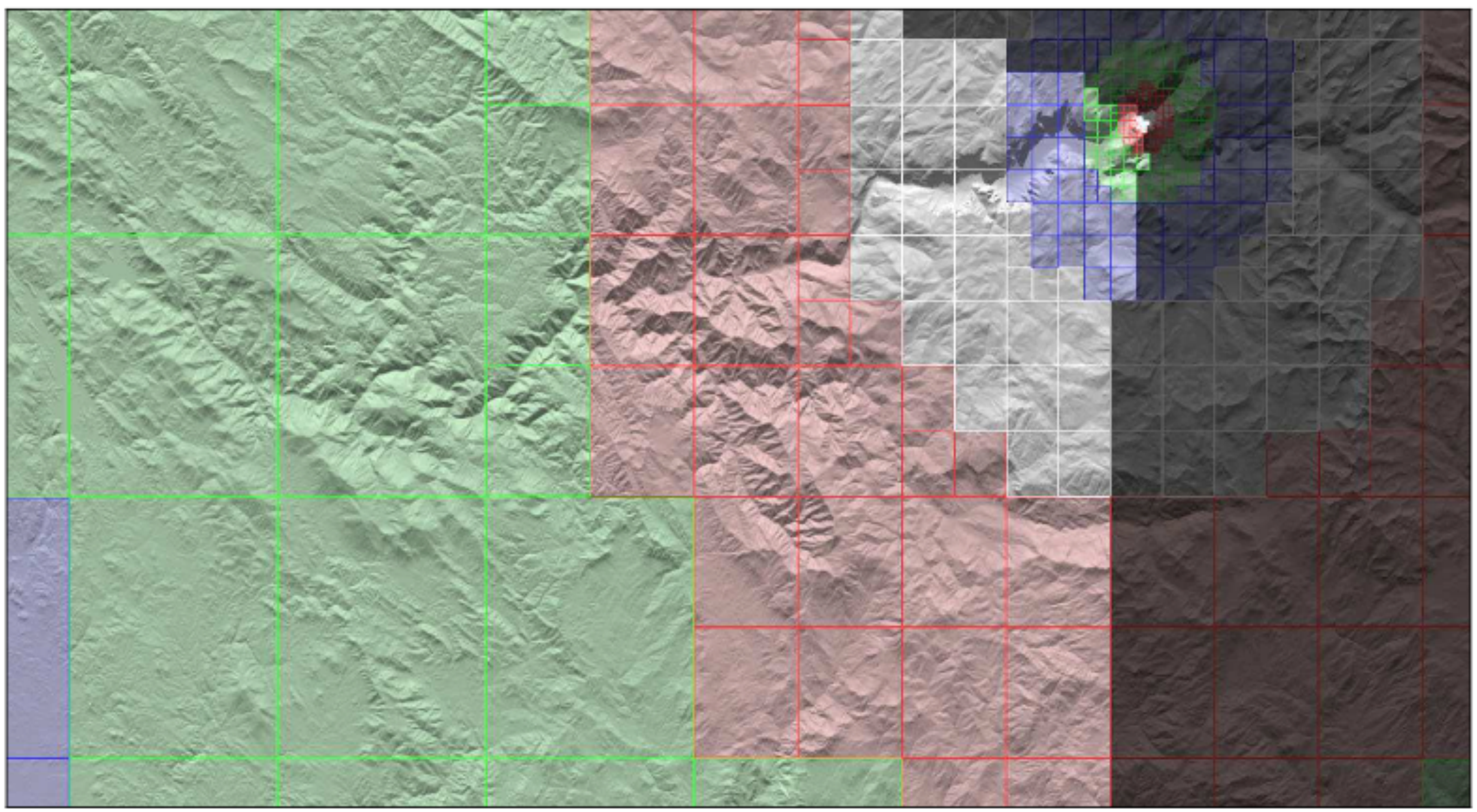


Figure 3 *LOD selection of quadtree nodes (the frustum culled section is shaded in dark).*

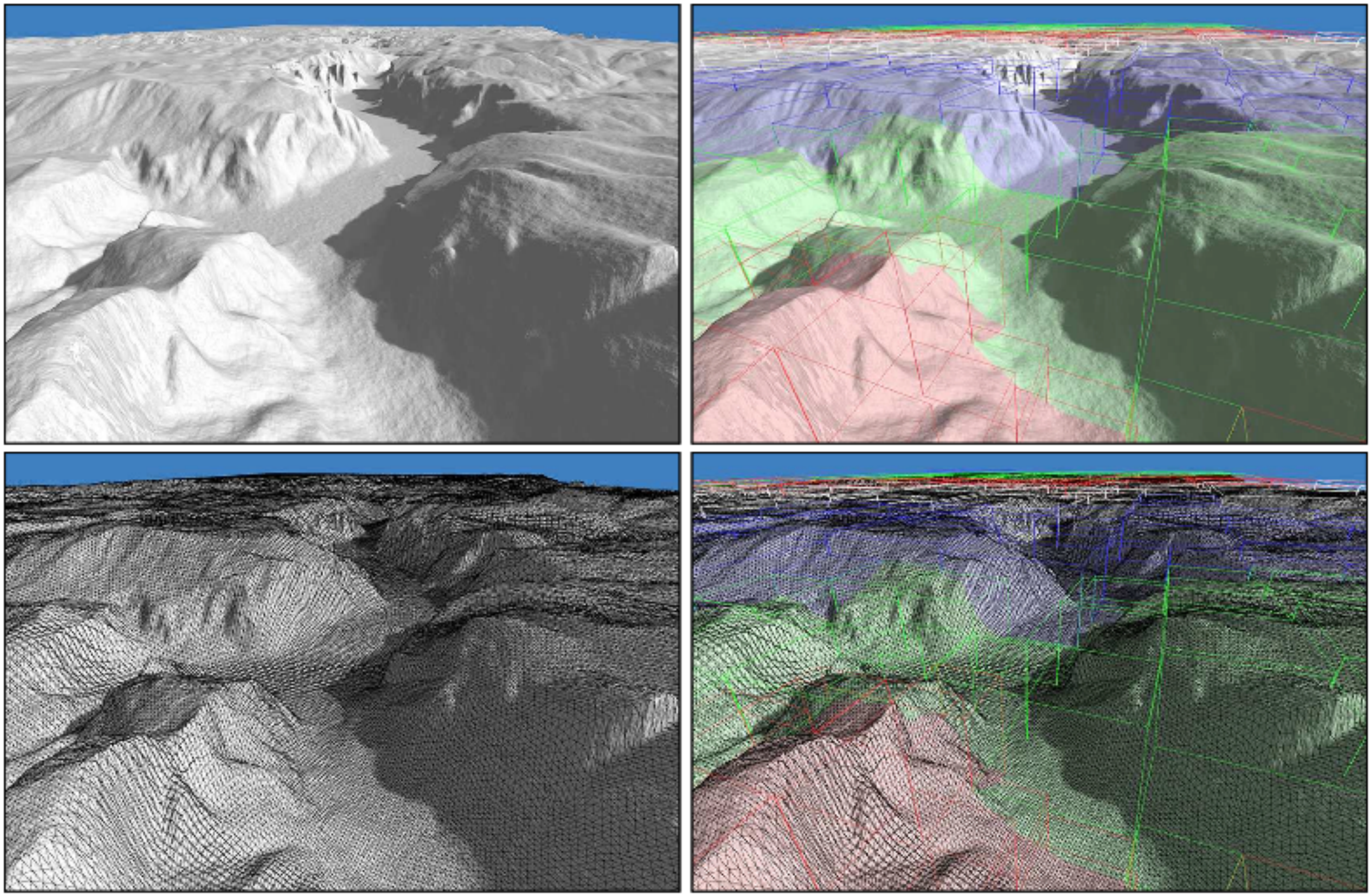


Figure 5 *Distribution of LOD levels and nodes (different colors represent different layers).*