Algorithms for GIS

Quadtrees

II

Laura Toma

Bowdoin College

Applications of quadtrees

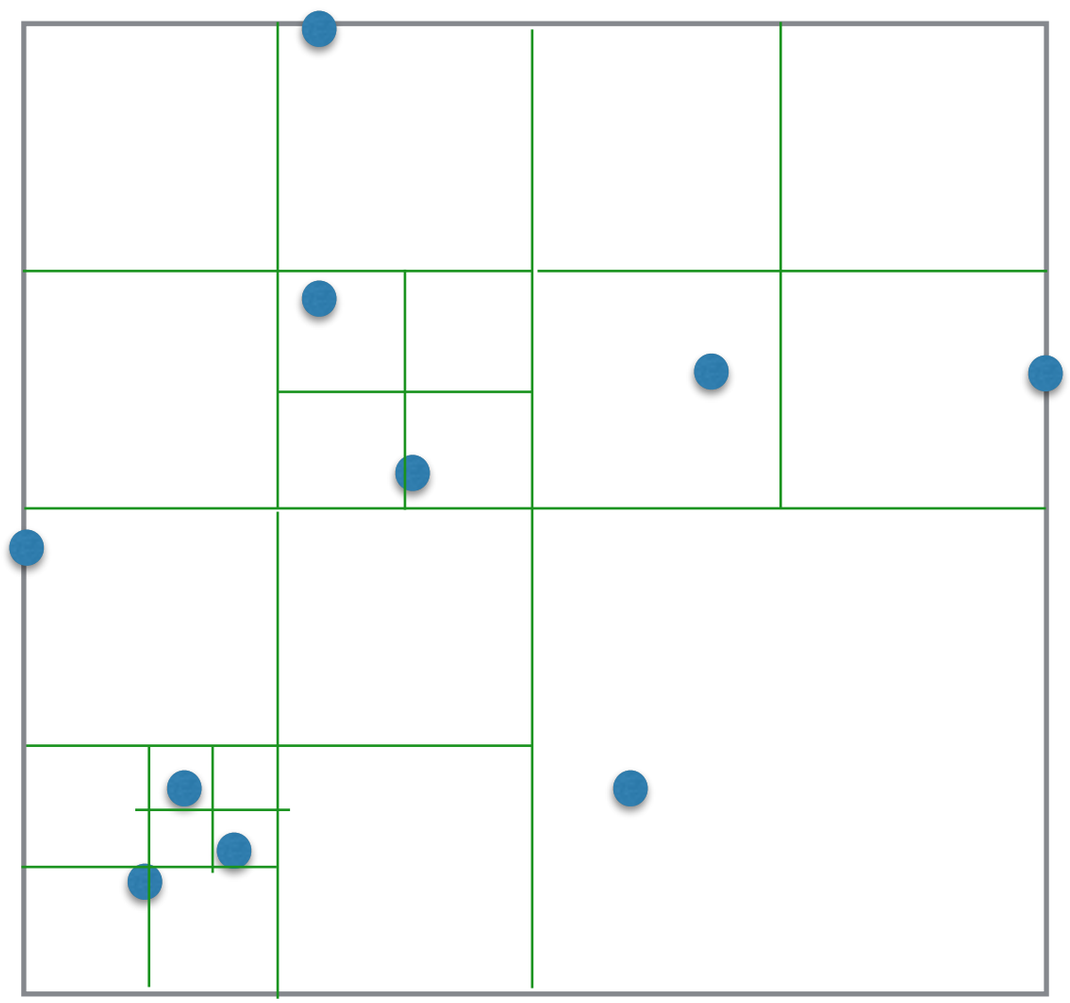
Applications of quadtrees

- 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.

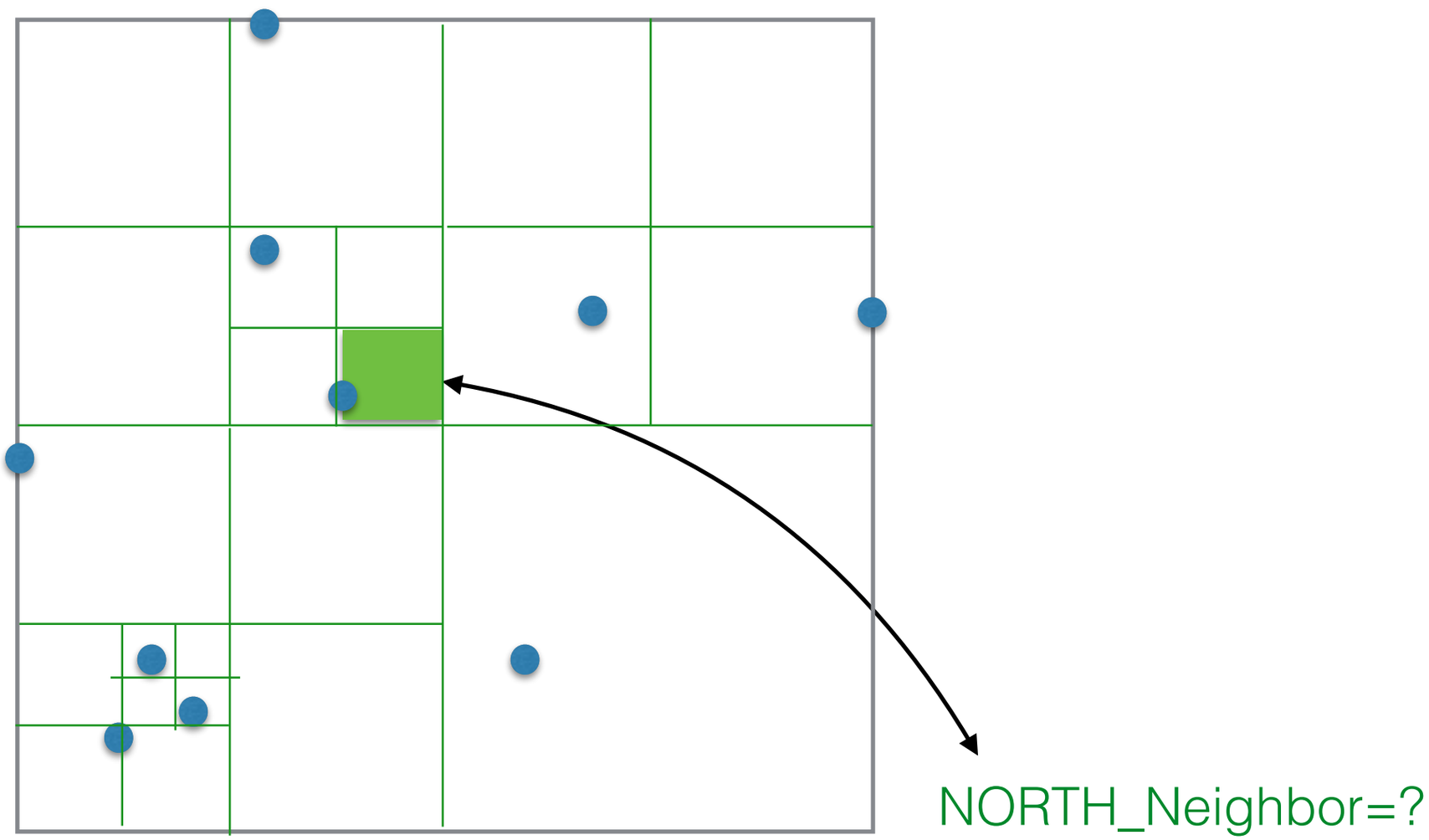
- two regions (squares) are adjacent iff they share an edge



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.

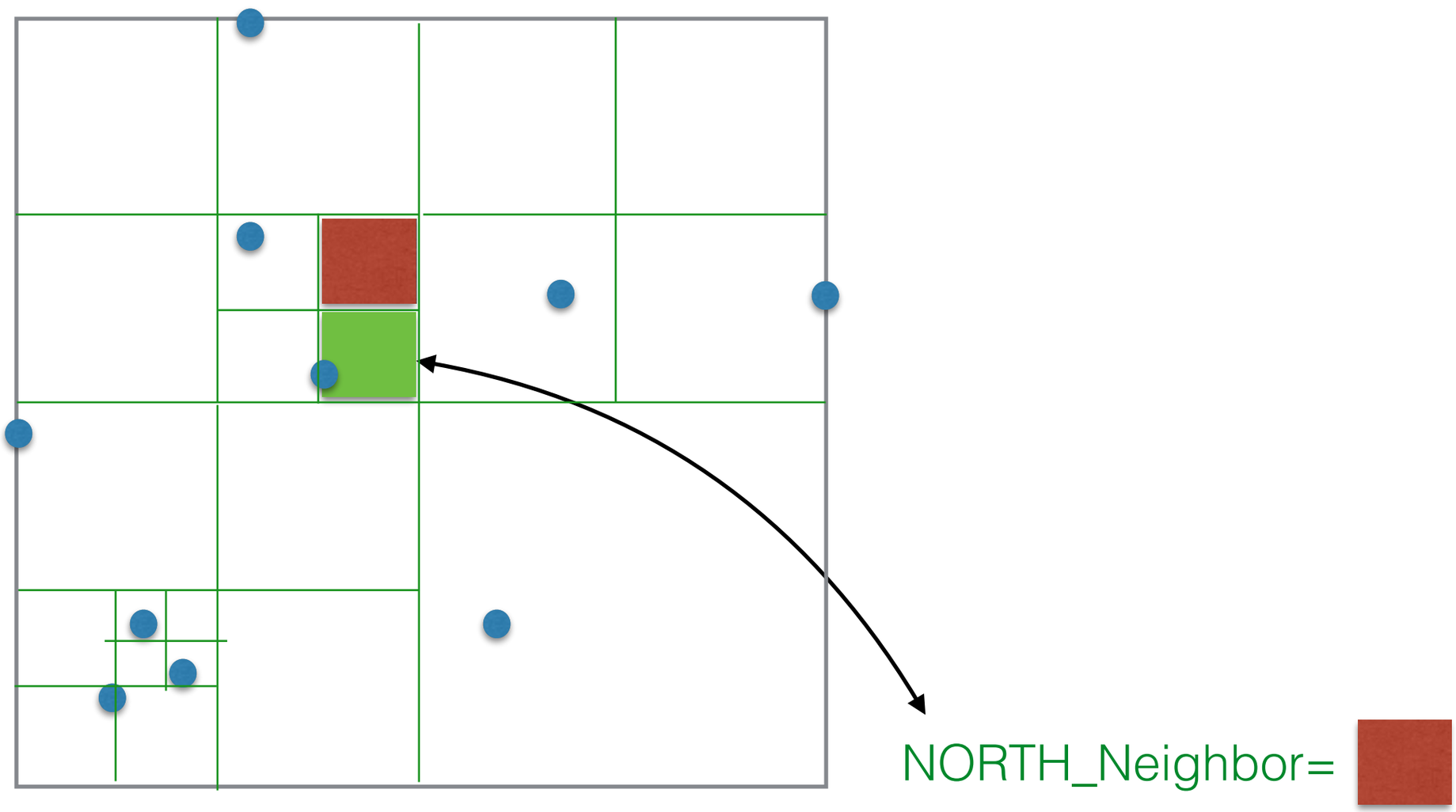
- two regions (squares) are adjacent iff they share an edge



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.

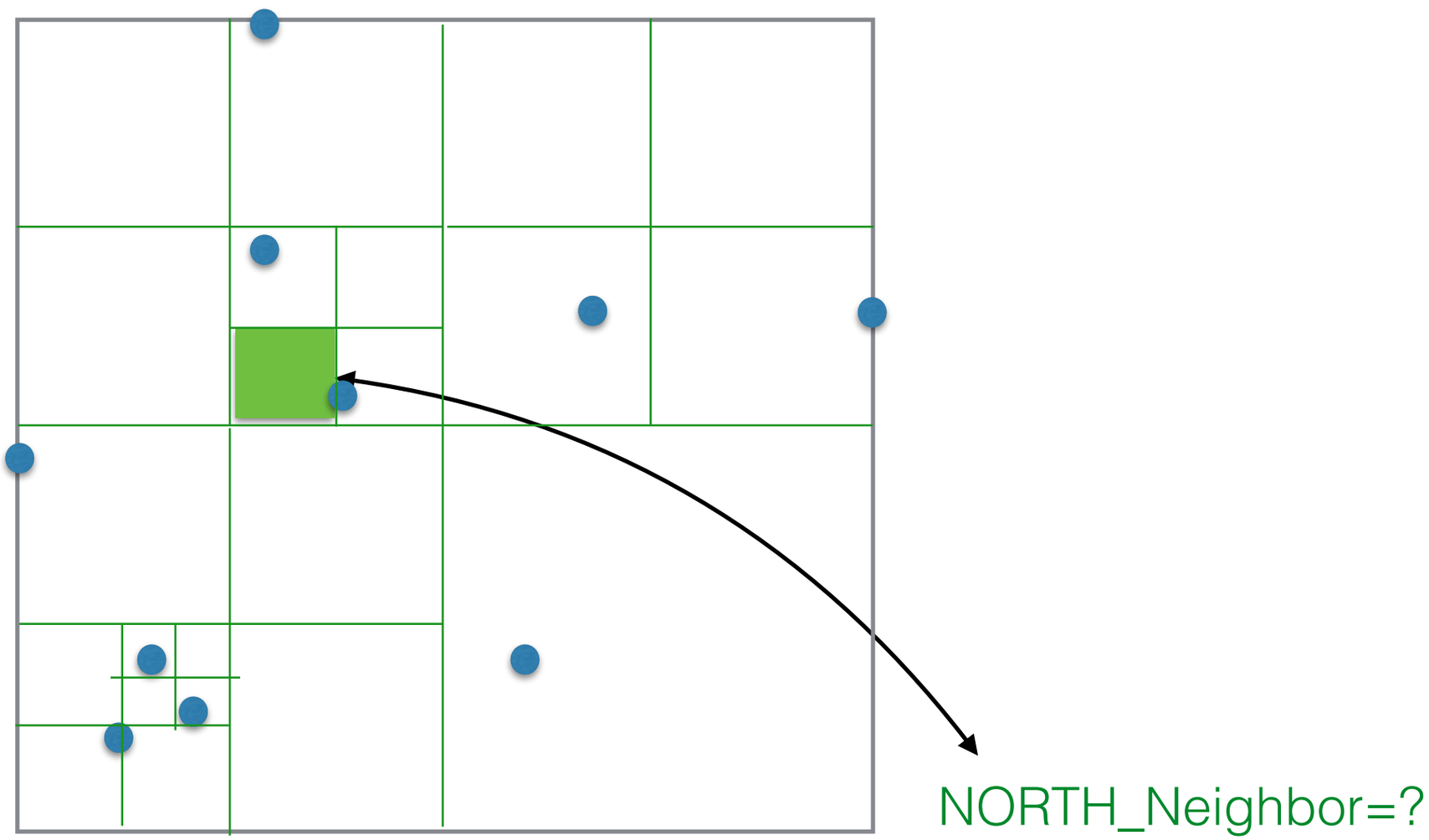
- two regions (squares) are adjacent iff they share an edge



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.

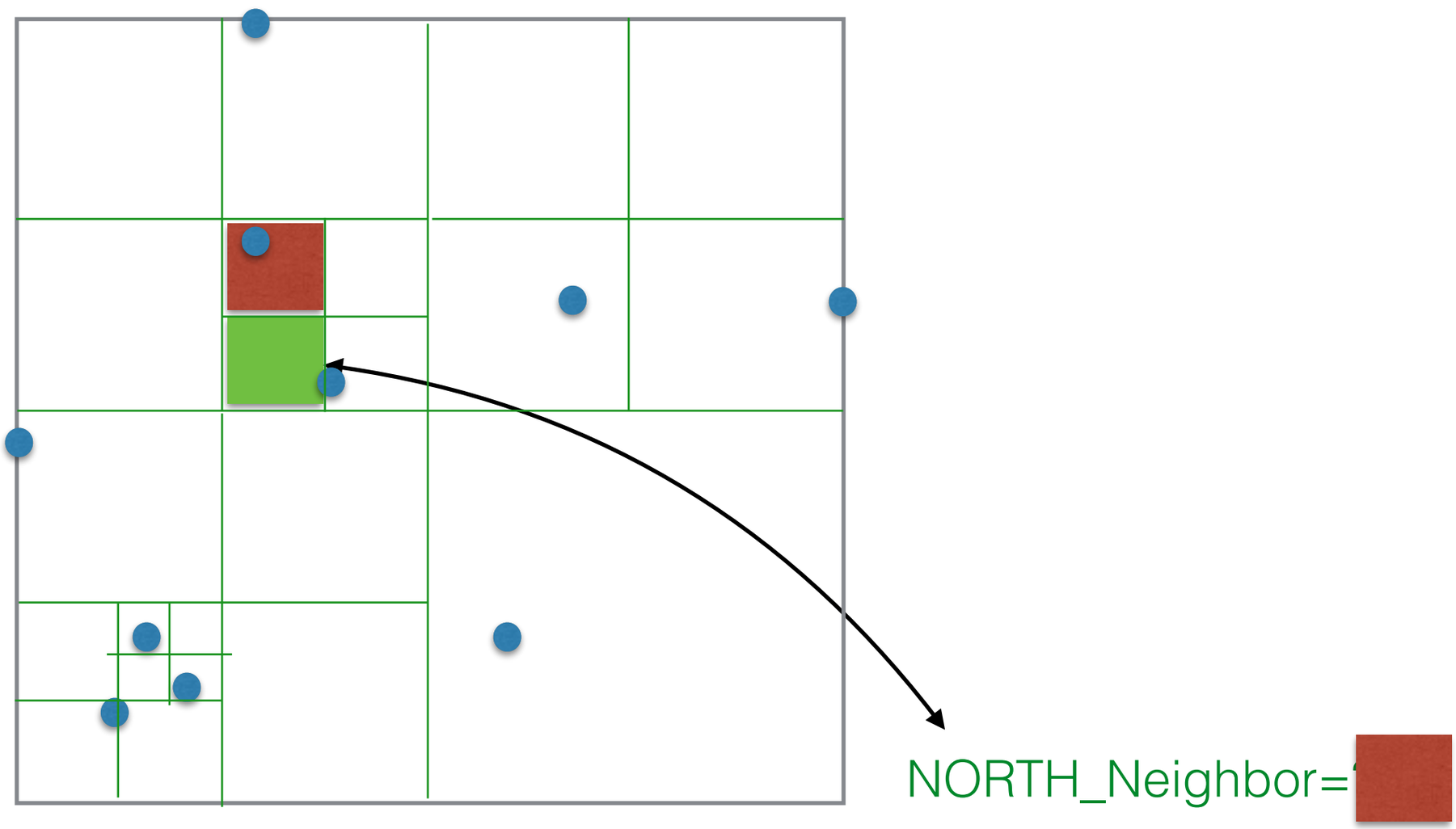
- two regions (squares) are adjacent iff they share an edge



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.

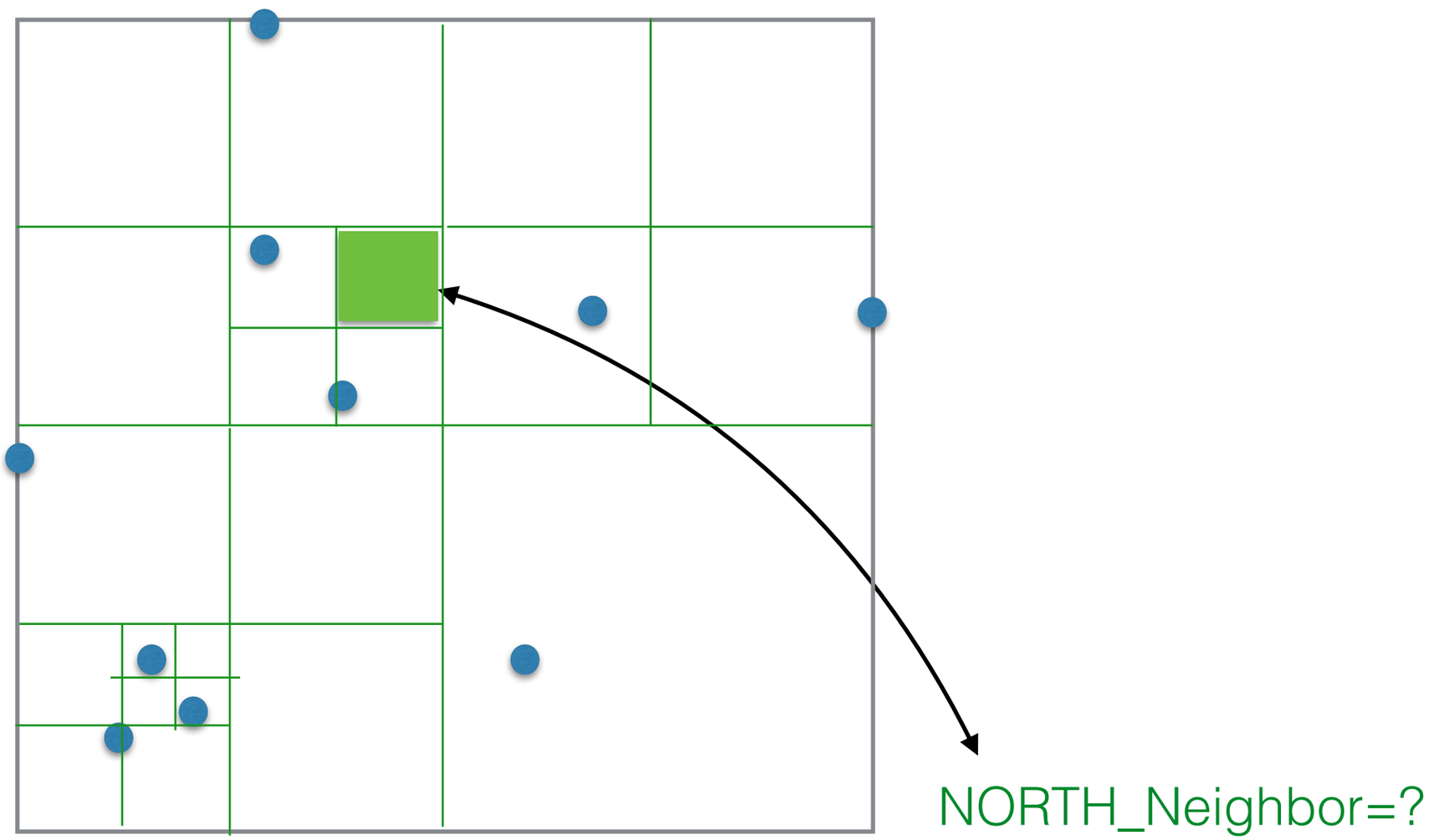
- two regions (squares) are adjacent iff they share an edge



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.

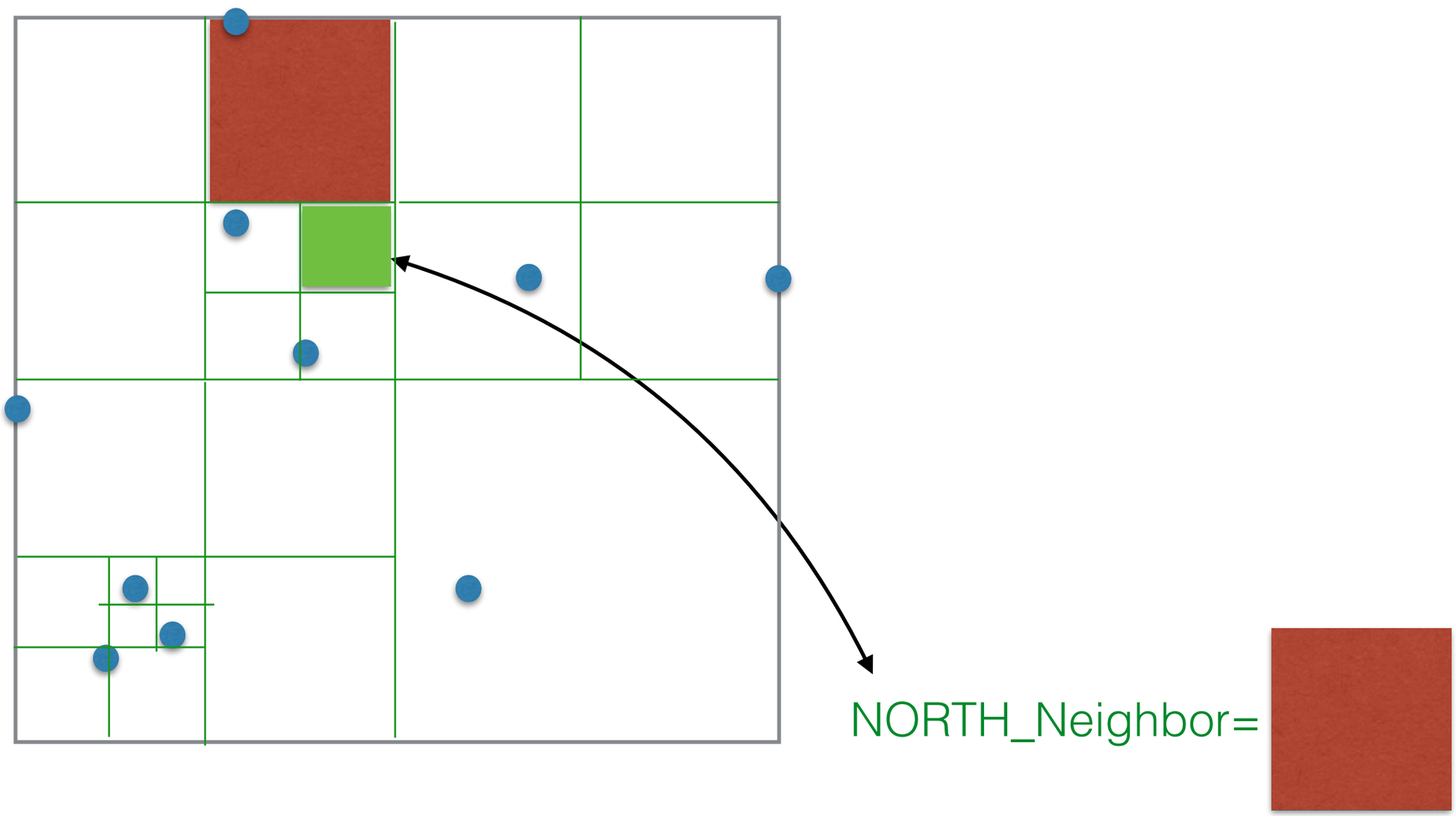
- two regions (squares) are adjacent iff they share an edge



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.

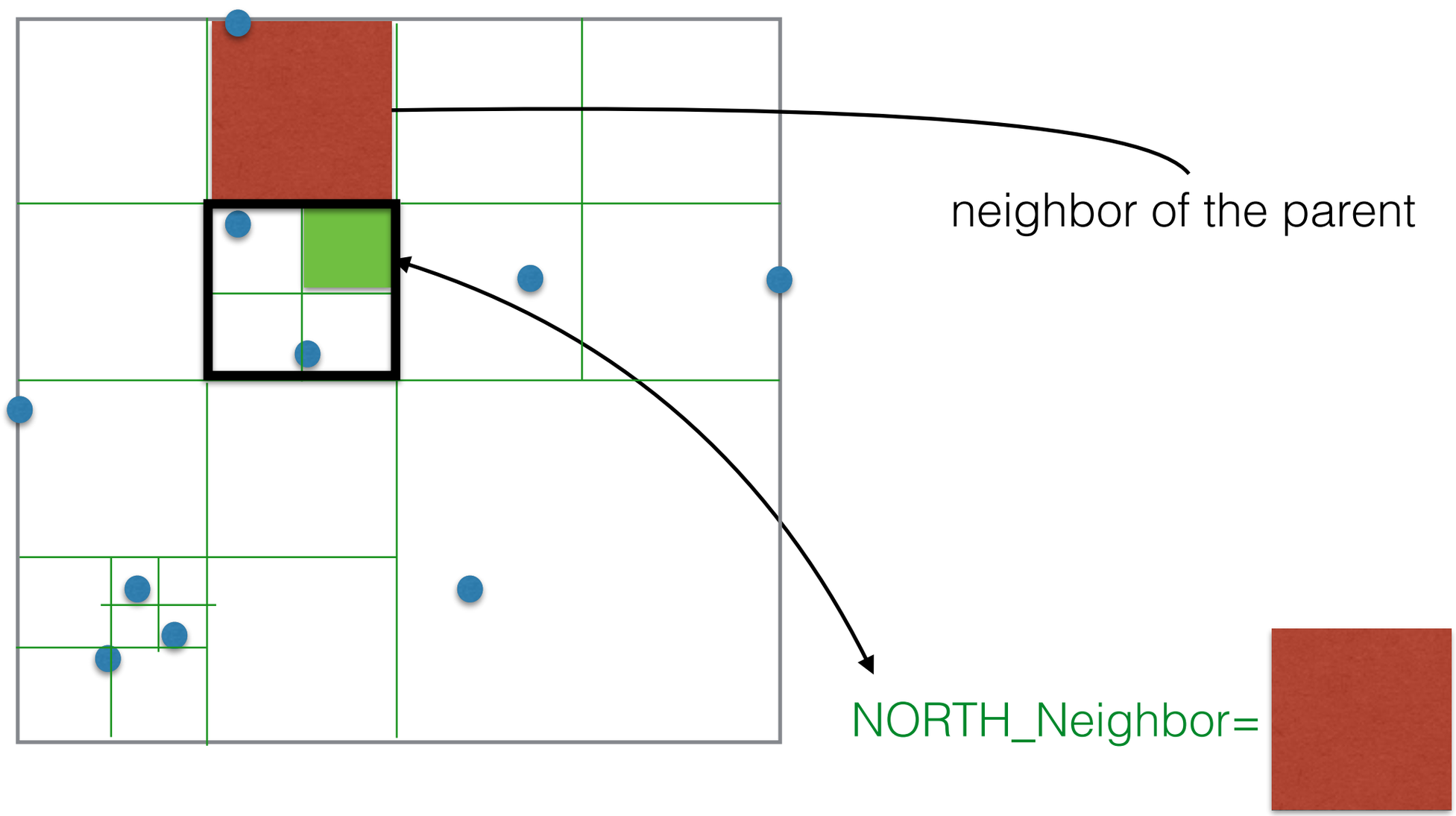
- two regions (squares) are adjacent iff they share an edge



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.

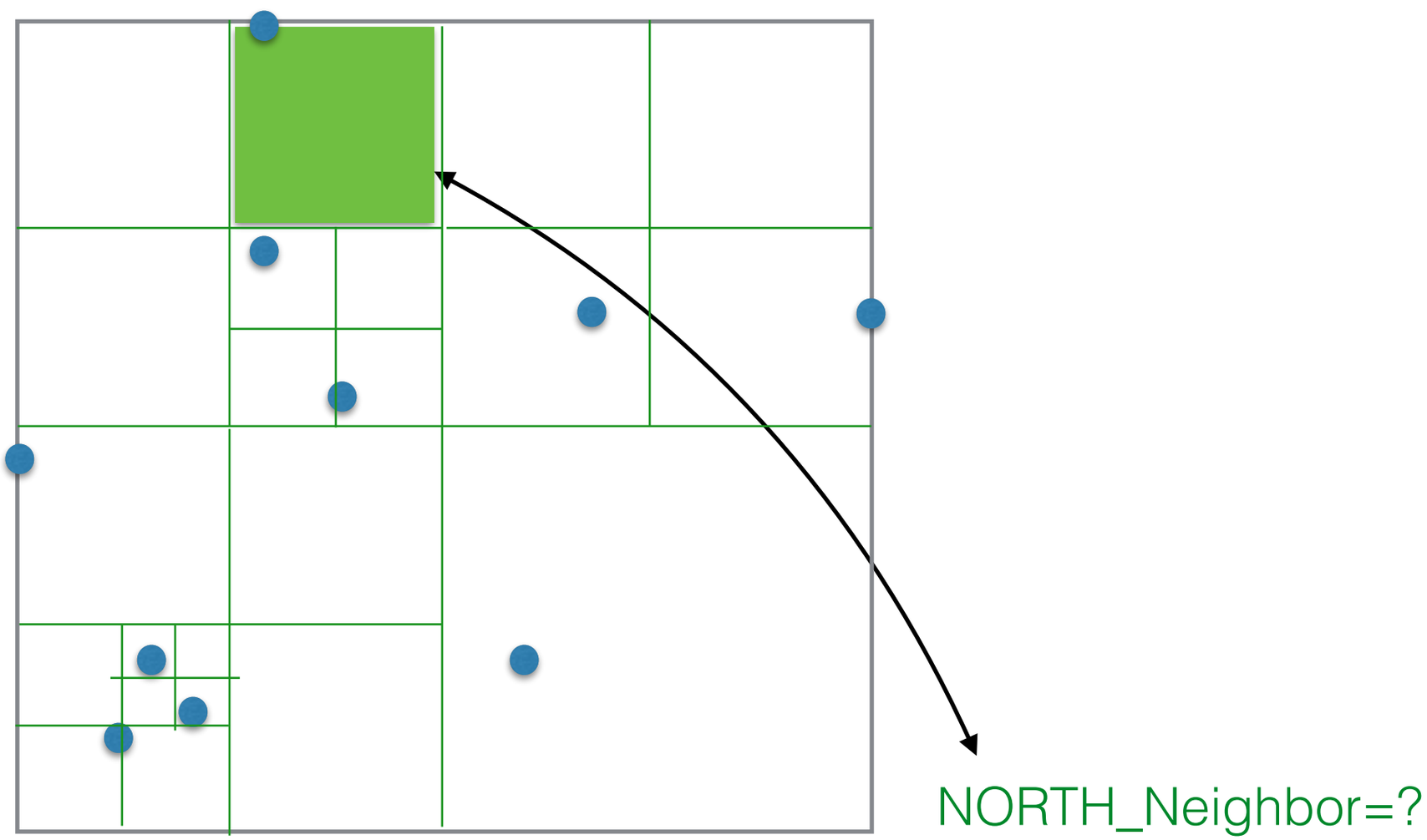
- two regions (squares) are adjacent iff they share an edge



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.

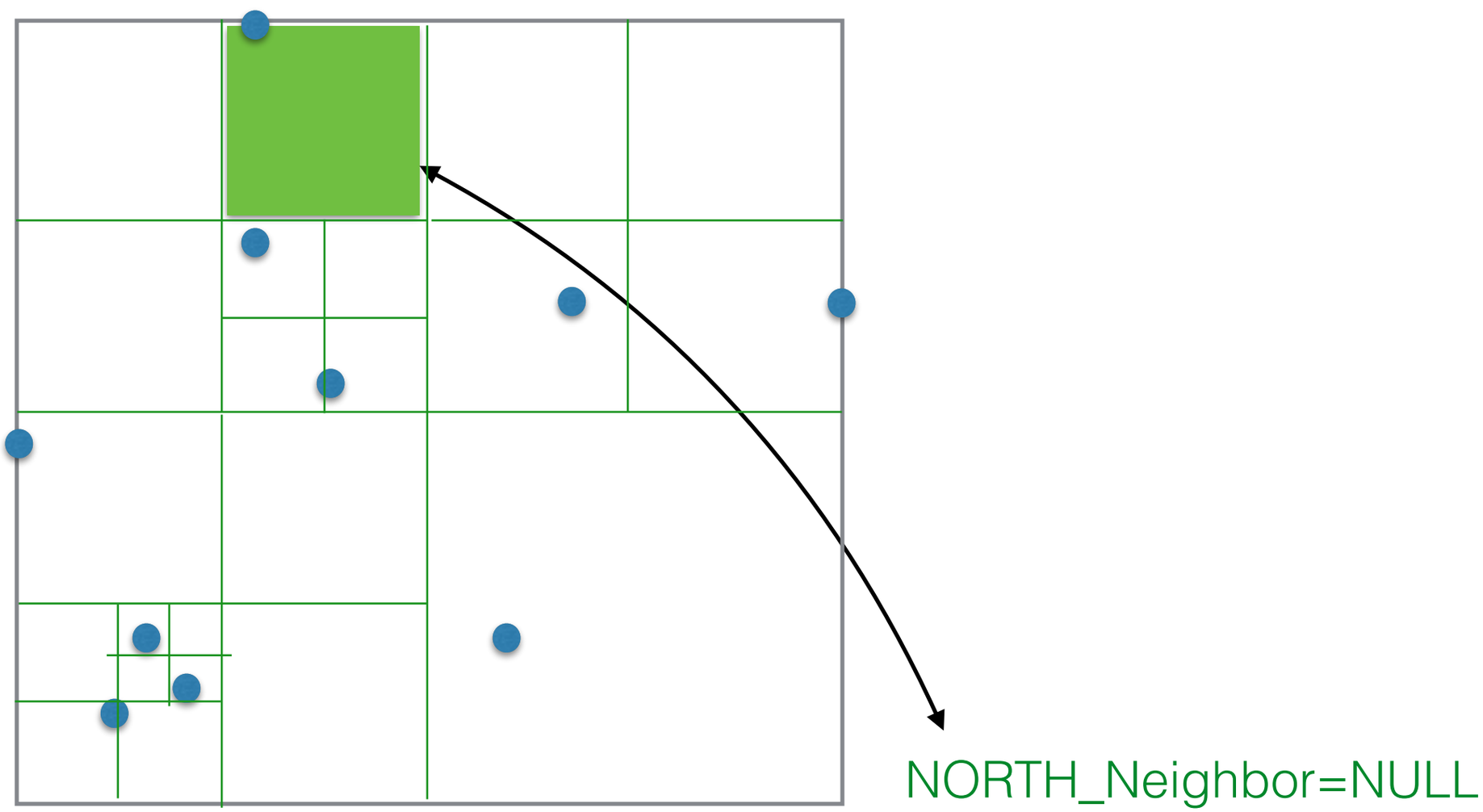
- two regions (squares) are adjacent iff they share an edge



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.

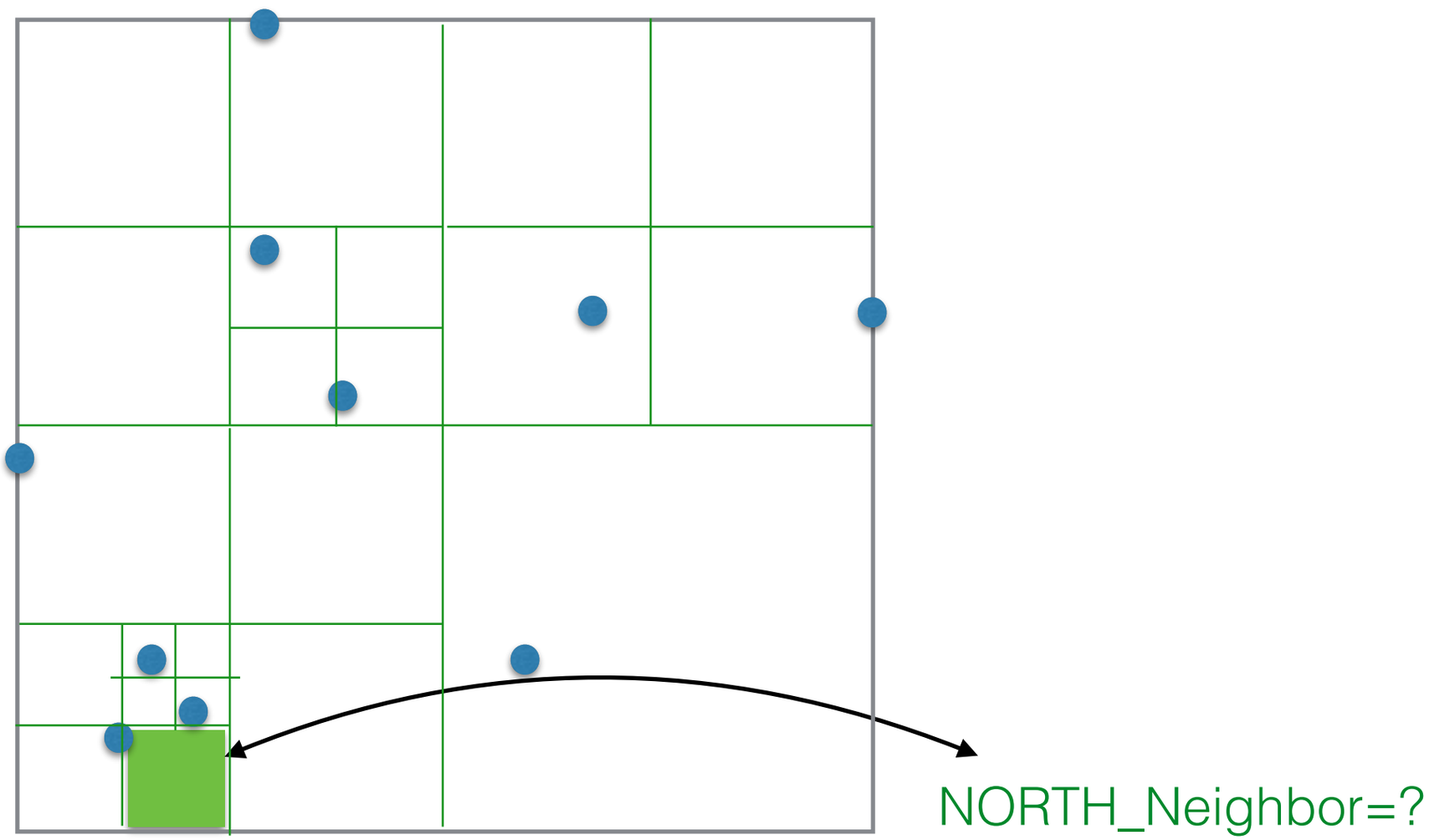
- two regions (squares) are adjacent iff they share an edge



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.

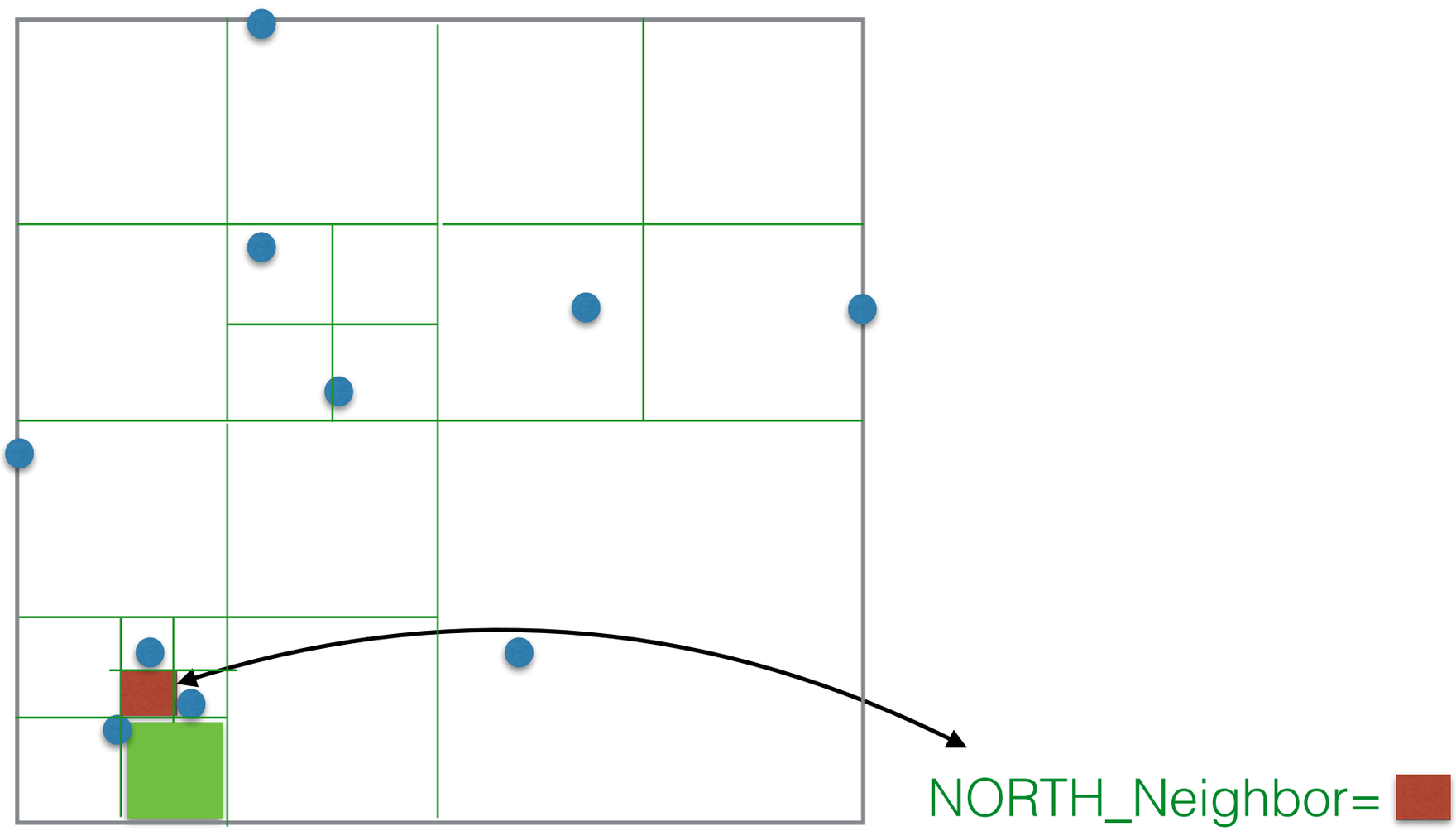
- two regions (squares) are adjacent iff they share an edge



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.

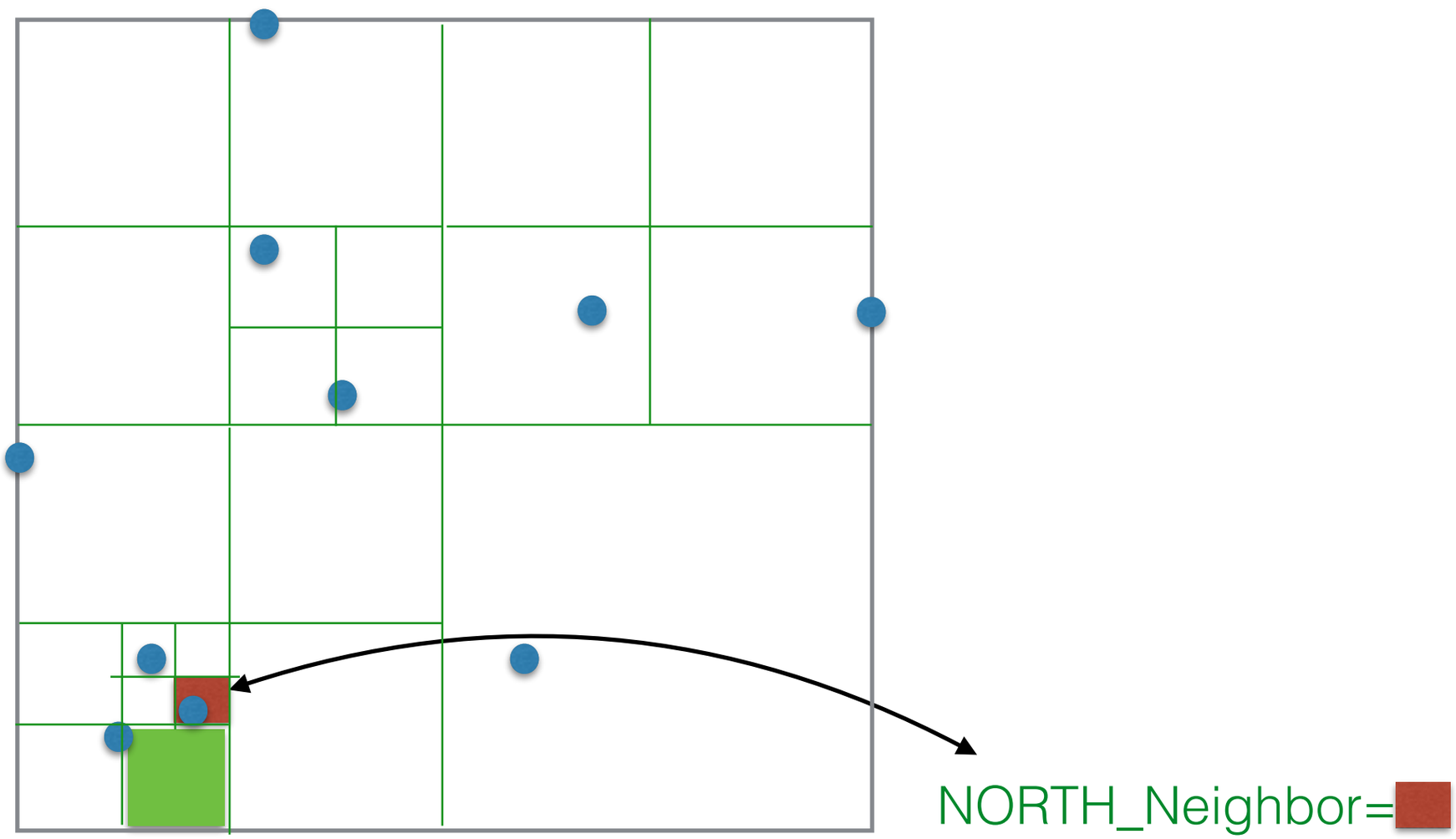
- two regions (squares) are adjacent iff they share an edge



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.

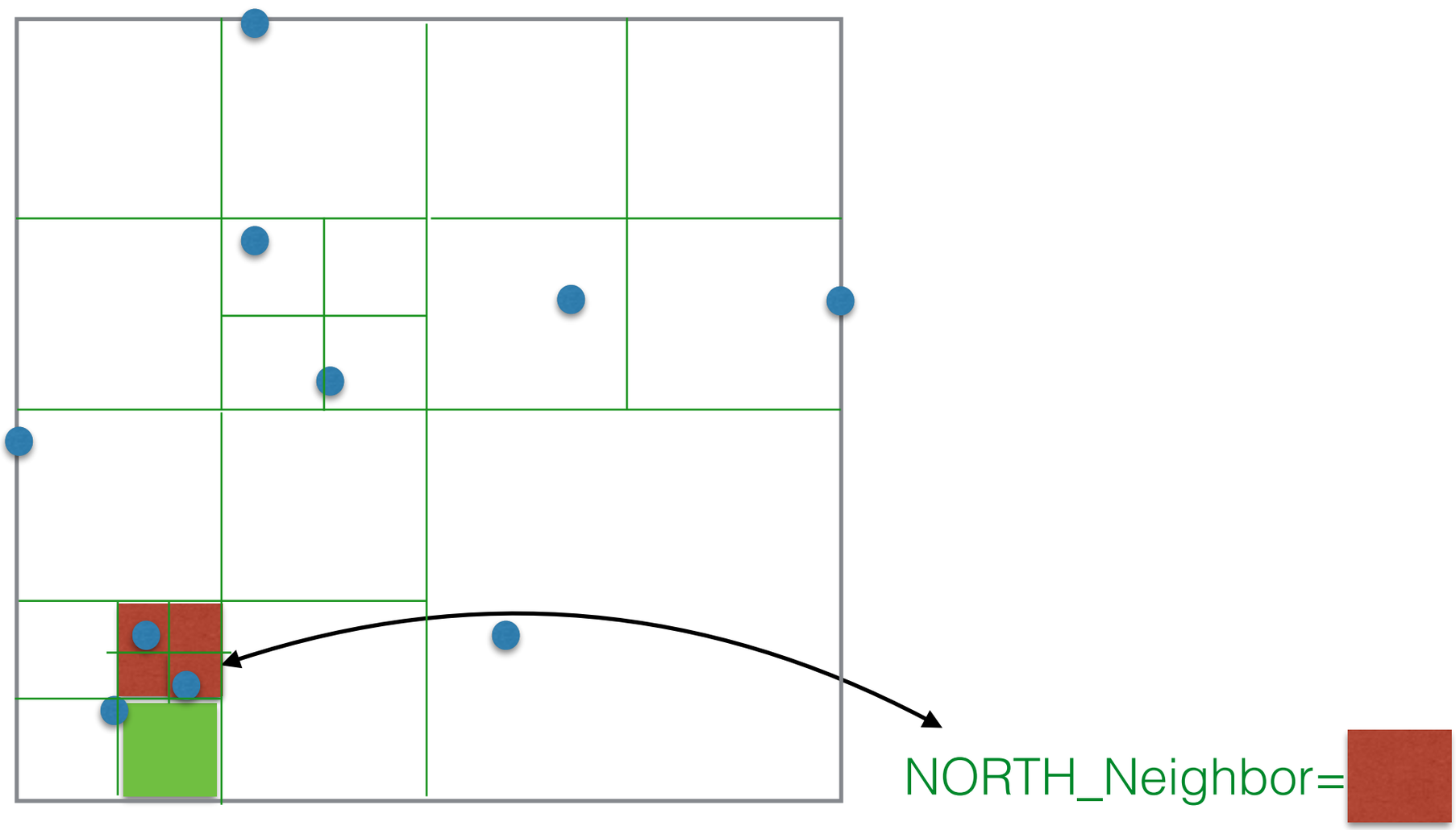
- two regions (squares) are adjacent iff they share an edge



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.

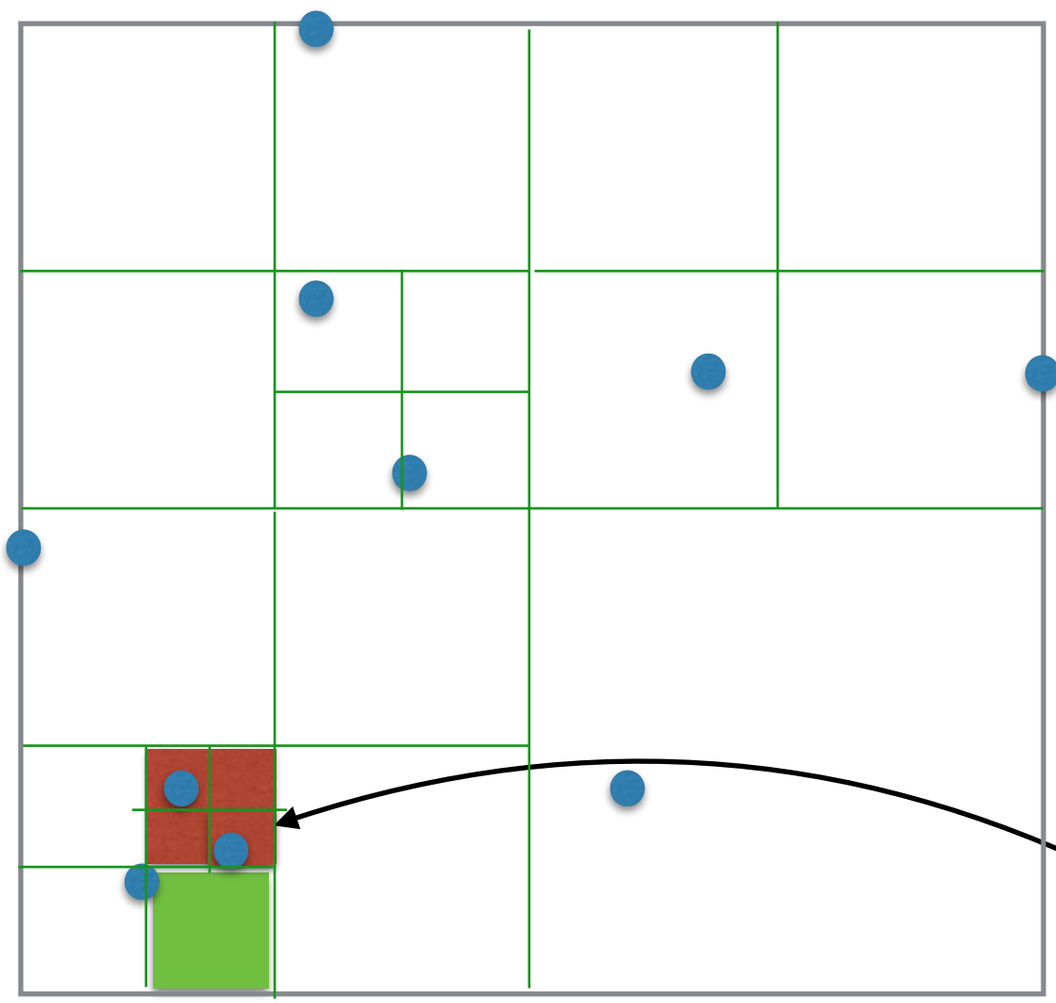
- two regions (squares) are adjacent iff they share an edge



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.

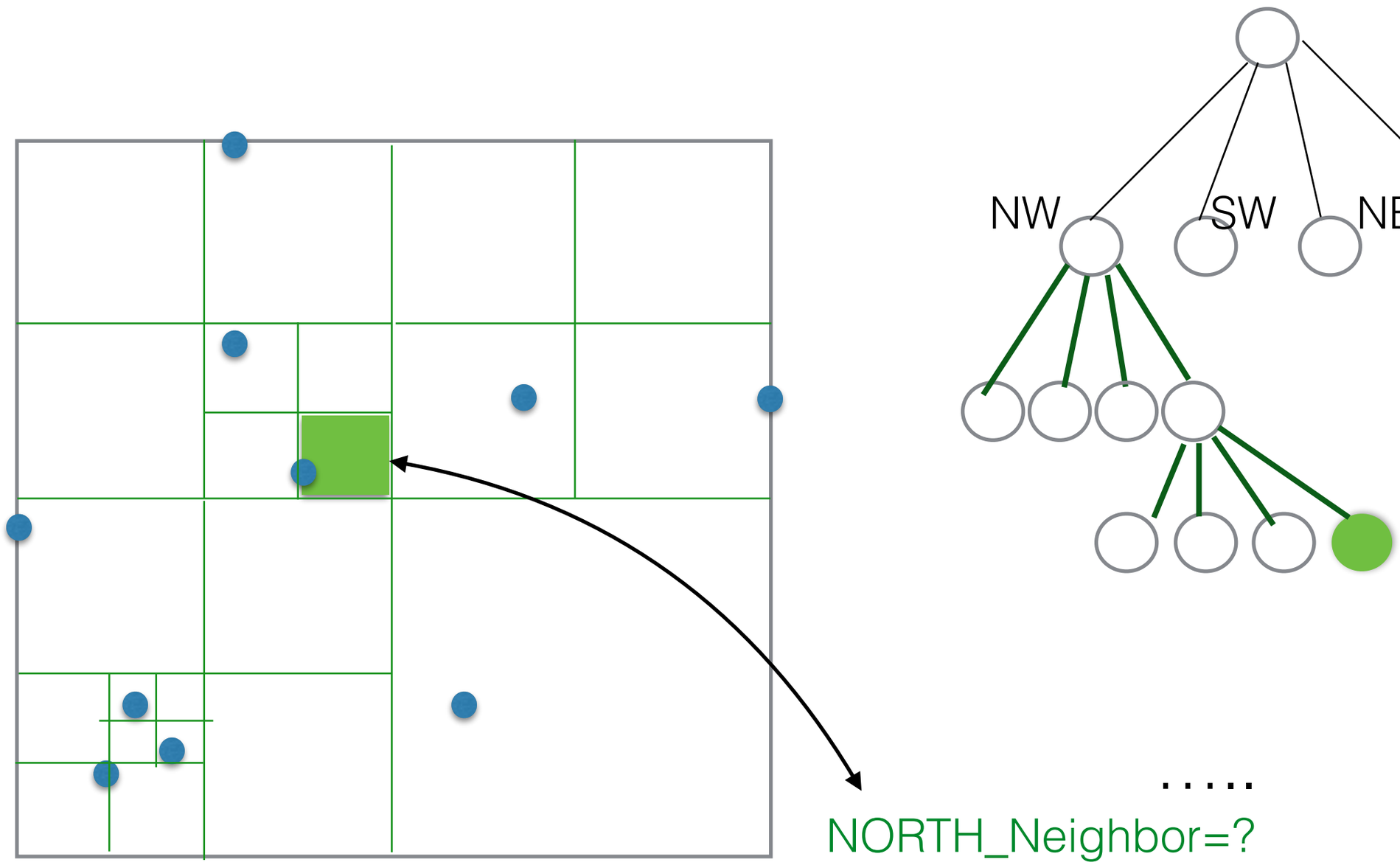
- two regions (squares) are adjacent iff they share an edge



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

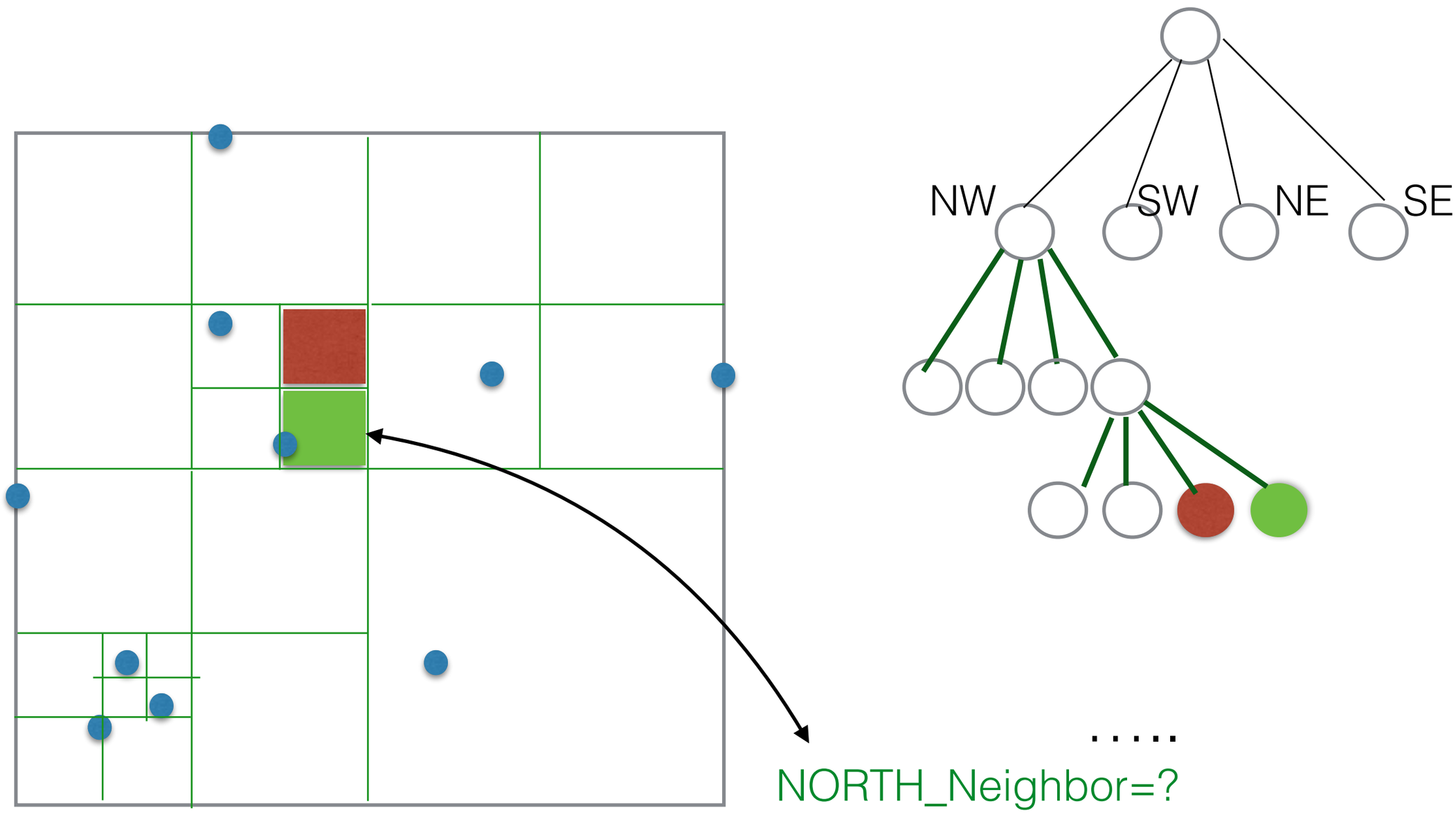
NORTH_Neighbor=

Visualizing it on the tree..



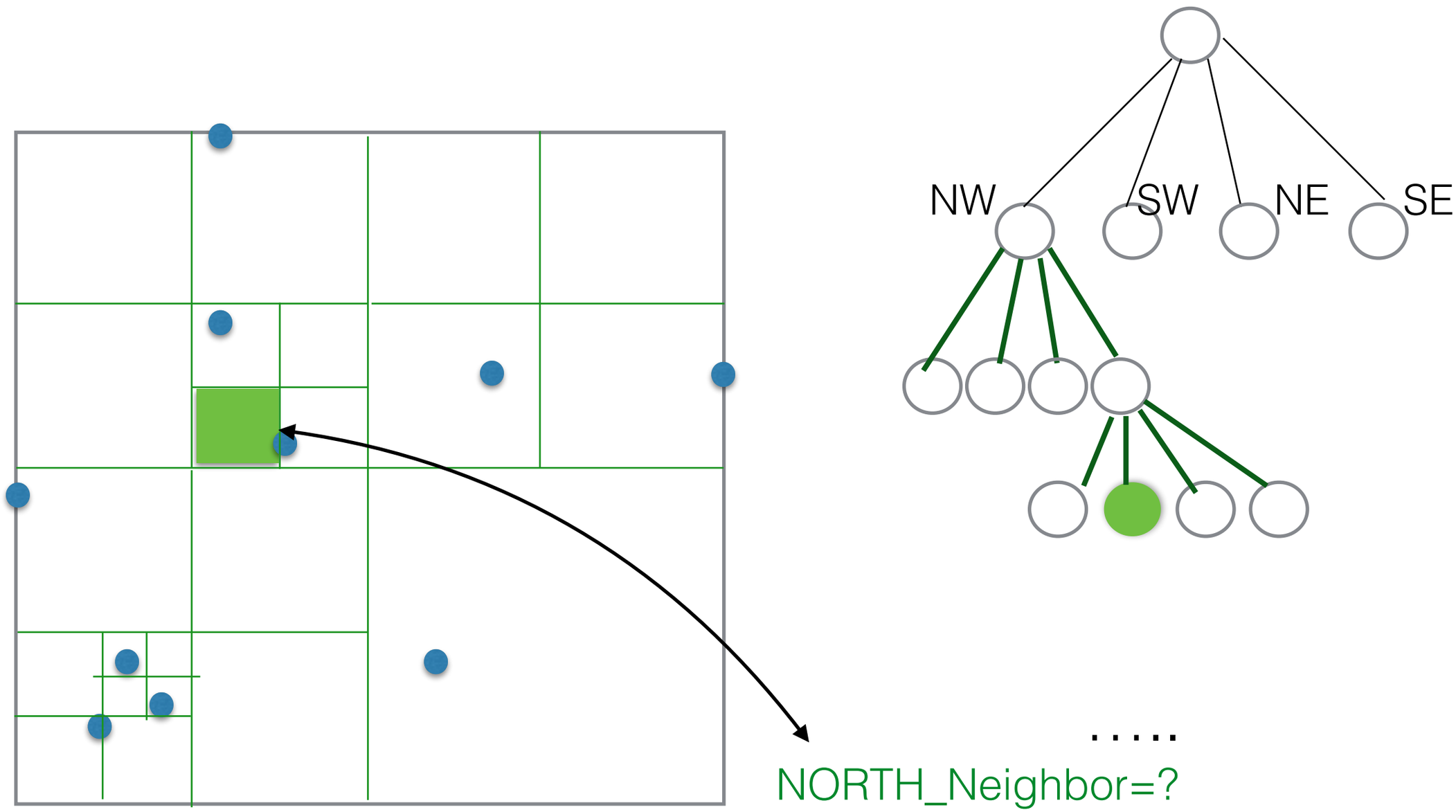
- try to find a node v' at the same depth as v
- if not possible, find the deepest

Visualizing it on the tree..



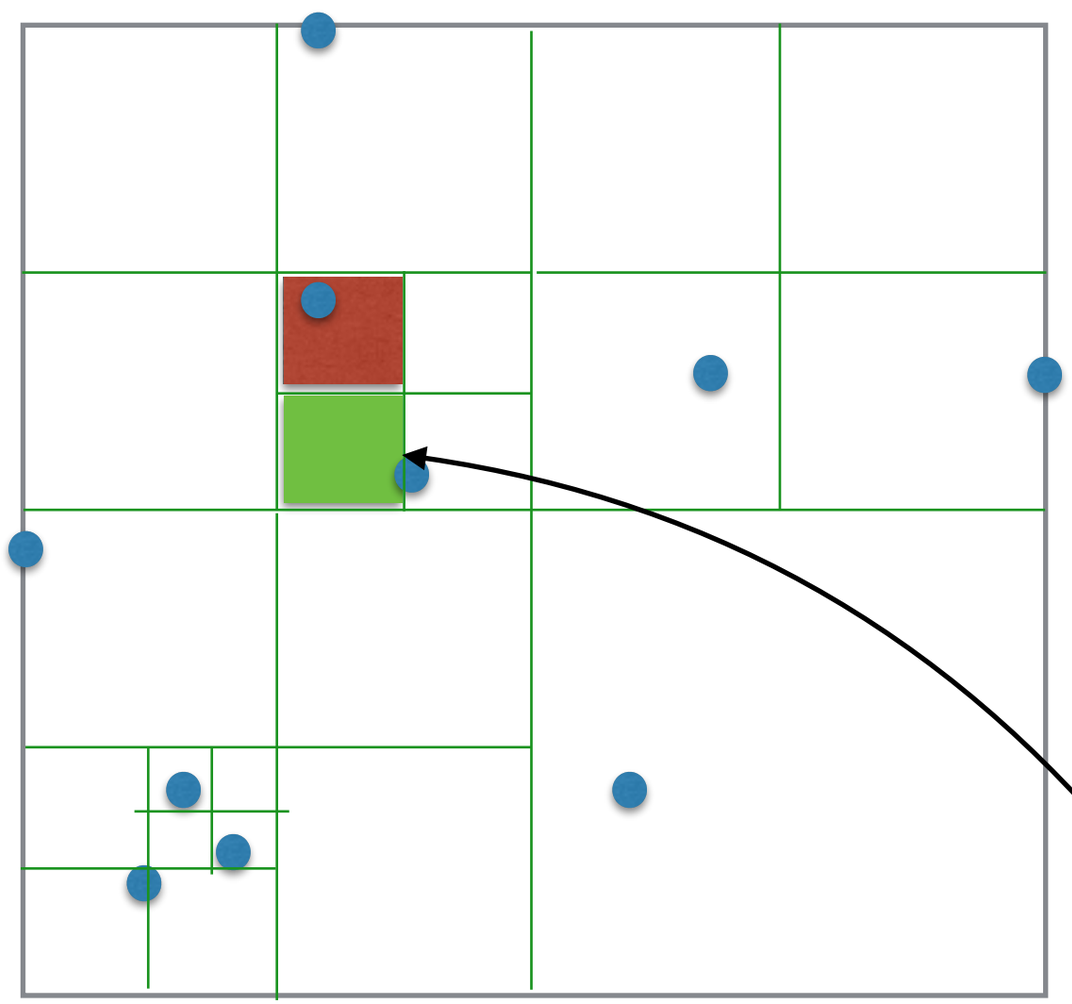
- try to find a node v' at the same depth as v
- if not possible, find the deepest

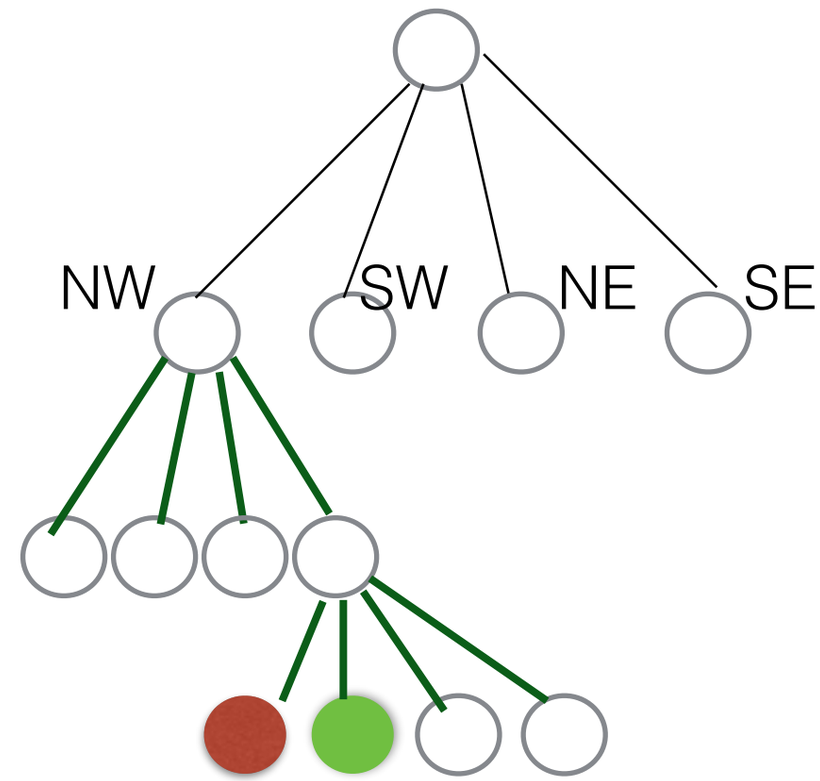
Visualizing it on the tree..



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

Visualizing it on the tree..



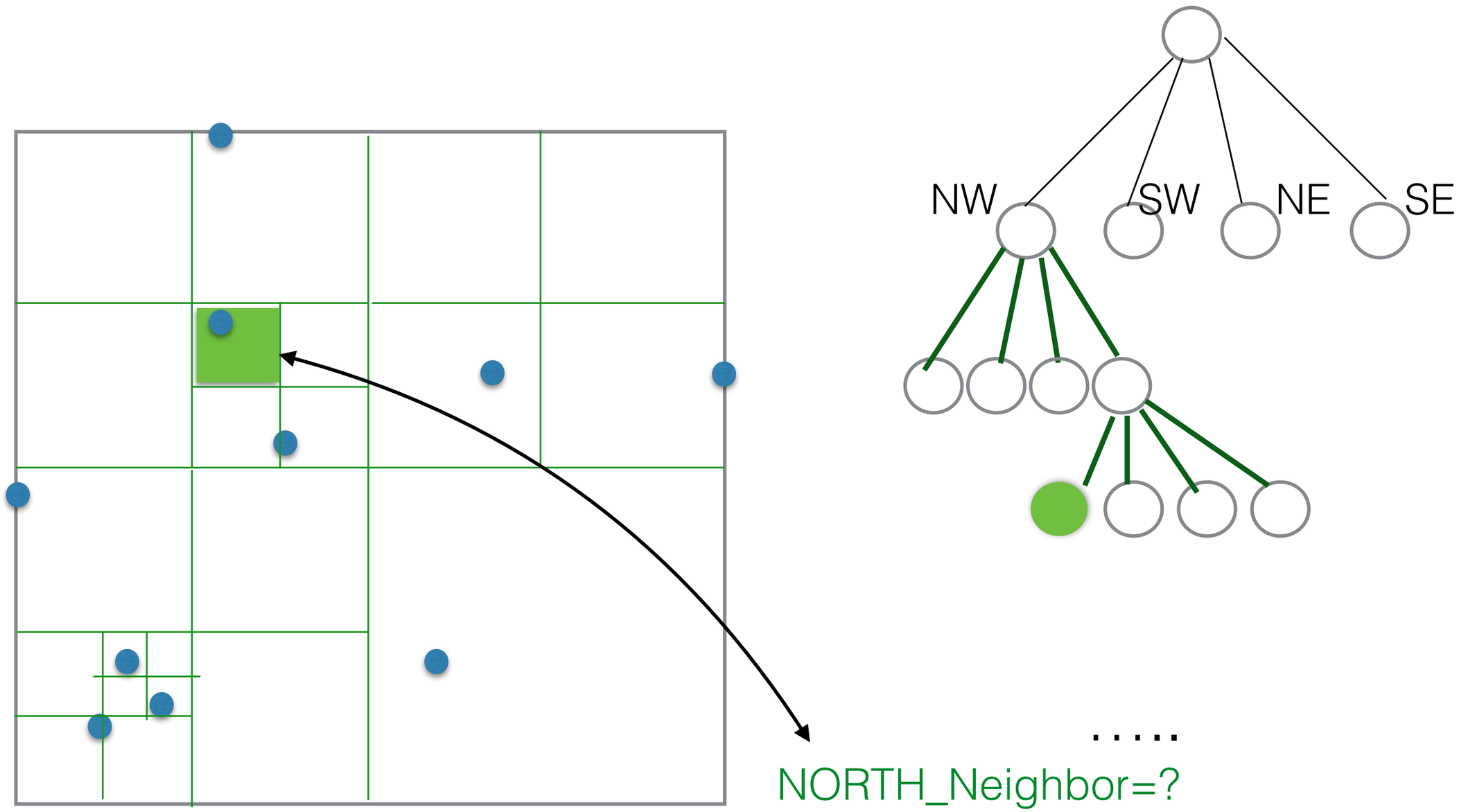


.....

NORTH_Neighbor=?

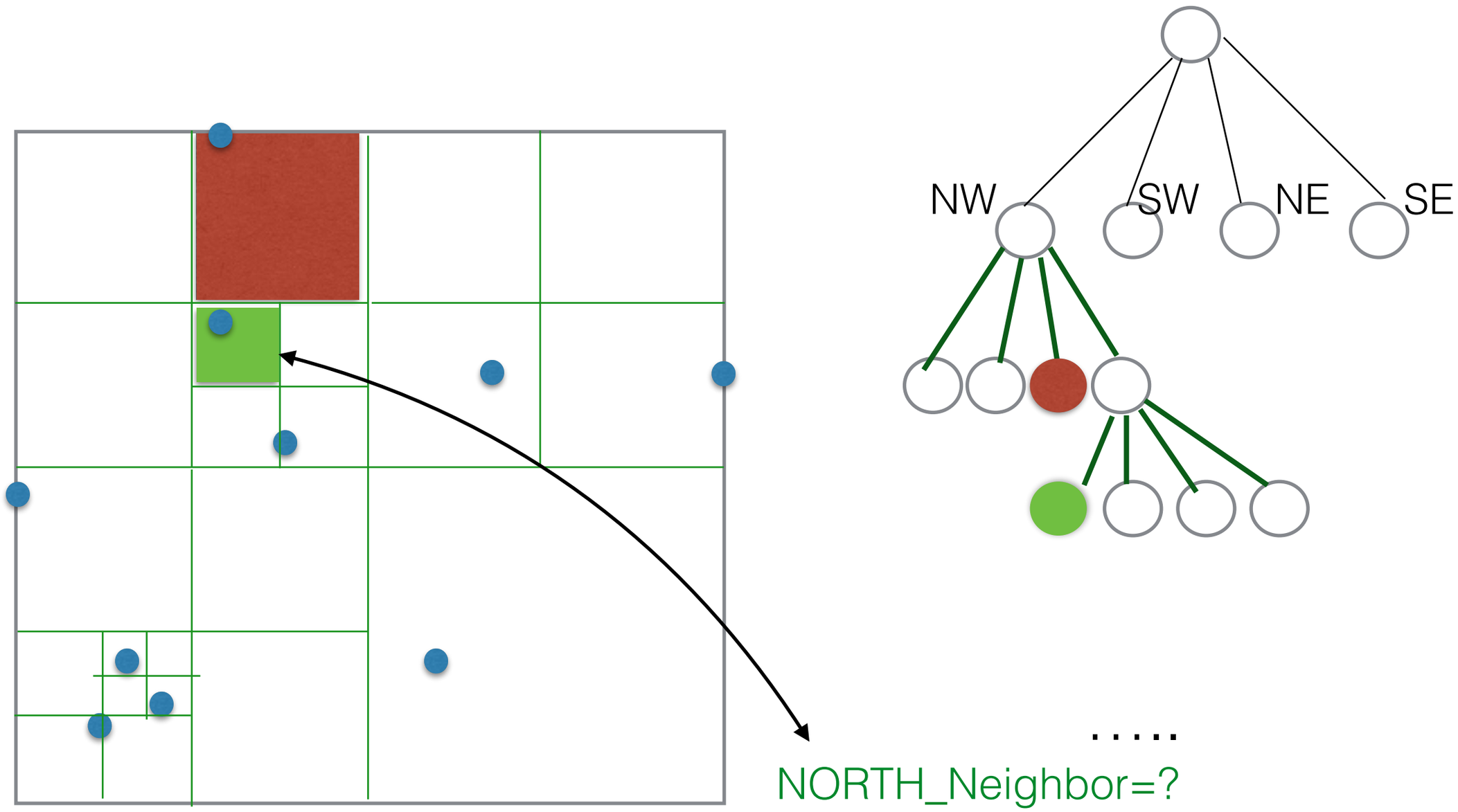
- try to find a node v' at the same depth as v
- if not possible, find the deepest

Visualizing it on the tree..



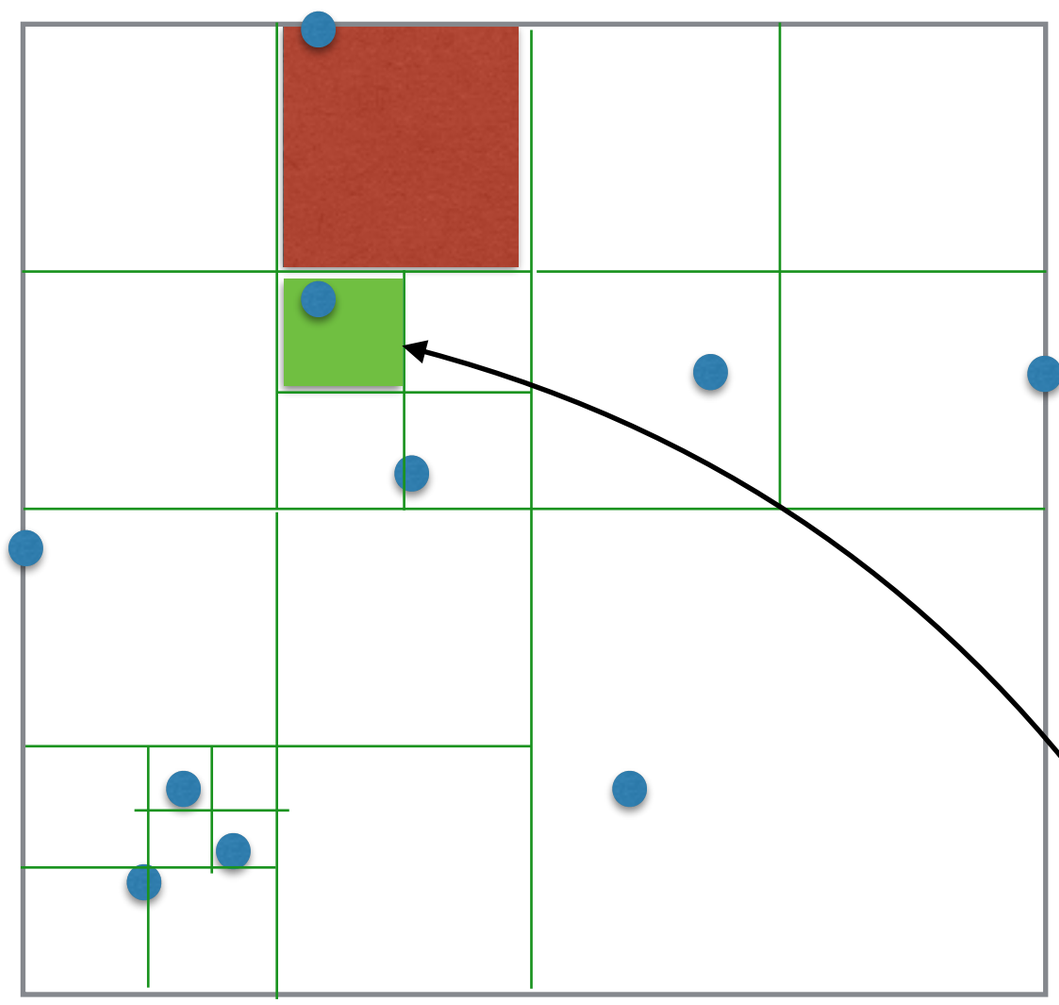
- try to find a node v' at the same depth as v
- if not possible, find the deepest

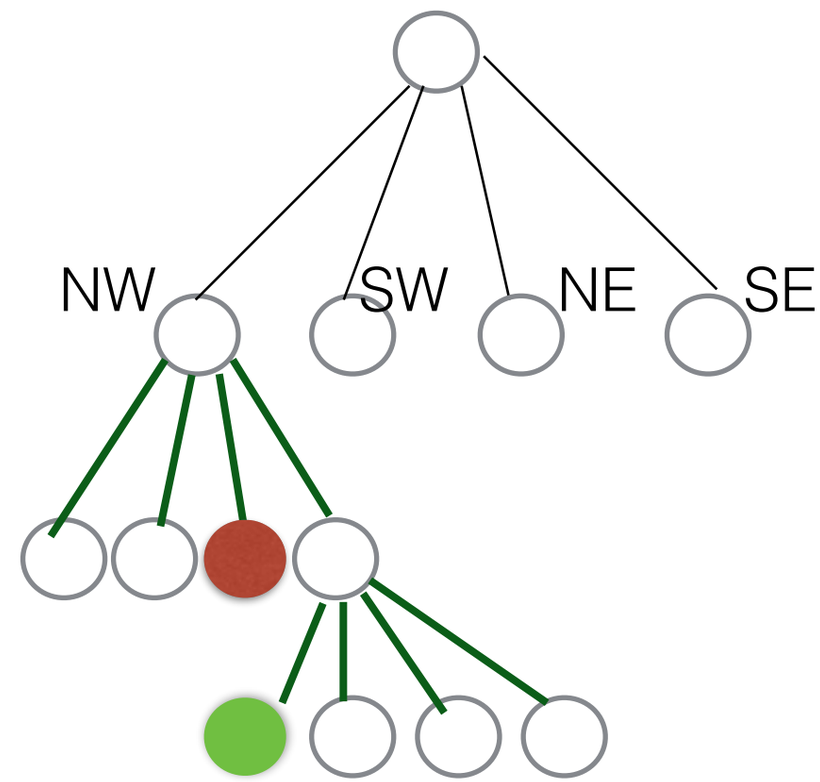
Visualizing it on the tree..



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

Visualizing it on the tree..





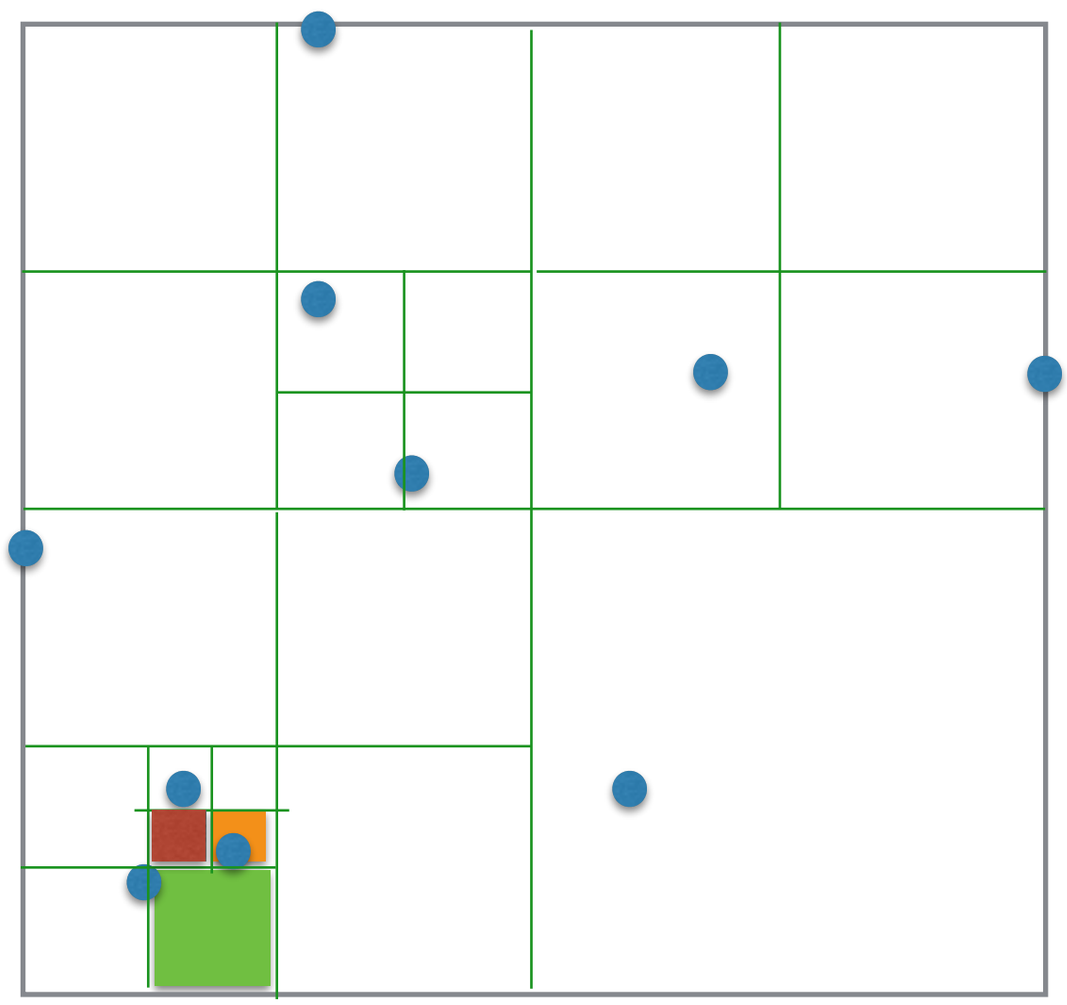
.....

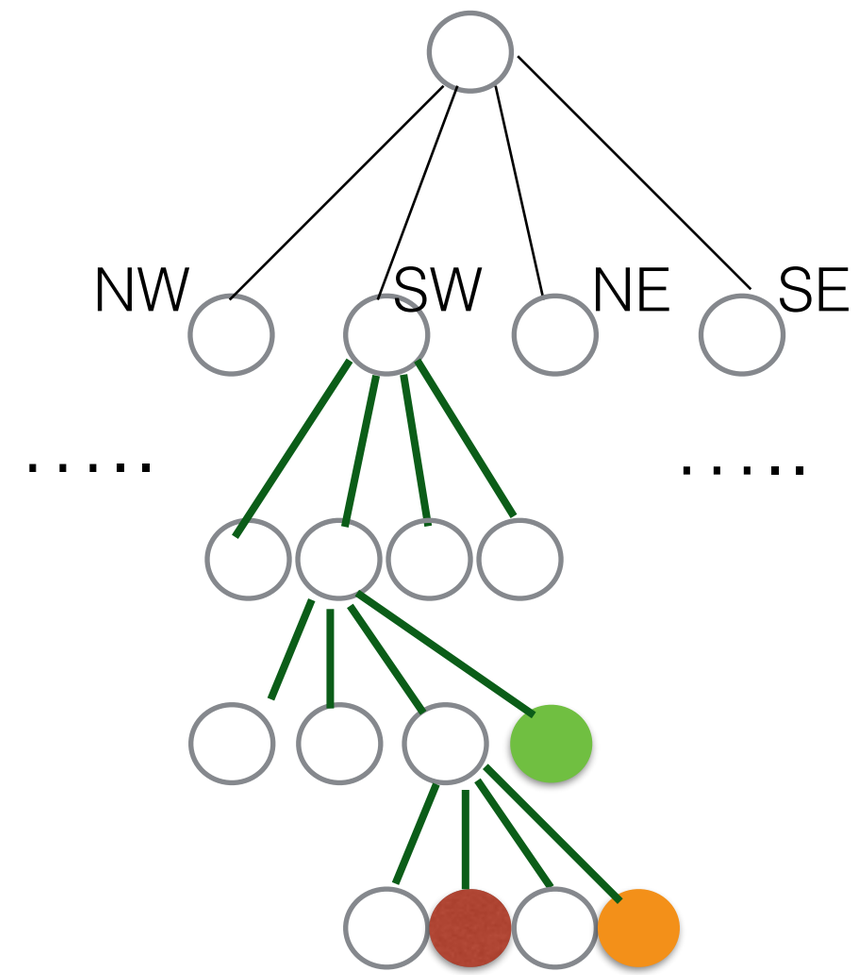
NORTH_Neighbor=?

- 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..

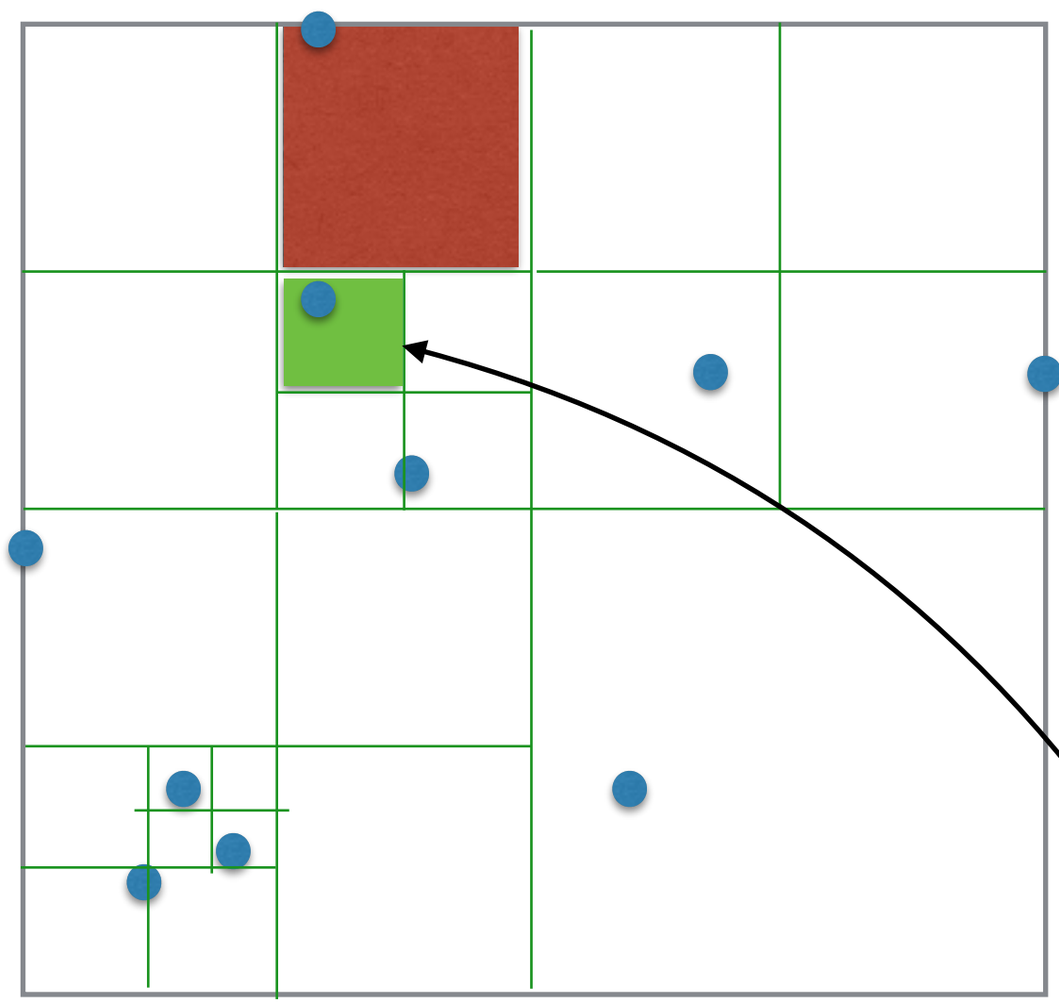


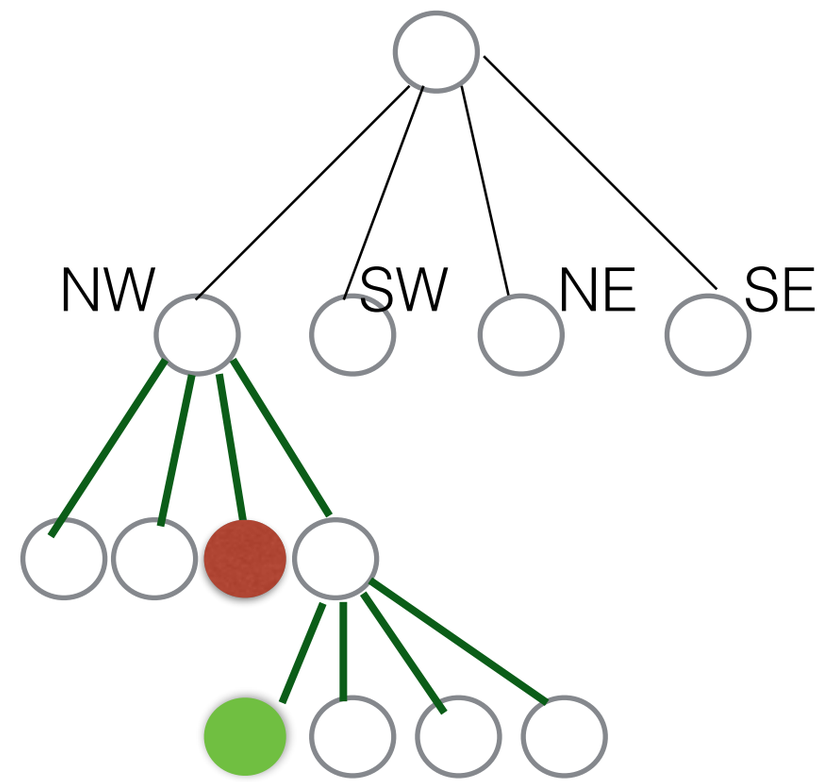


- try to find a node v' at the same depth as v
- if not possible, find the deepest

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

Visualizing it on the tree..





.....

NORTH_Neighbor=?

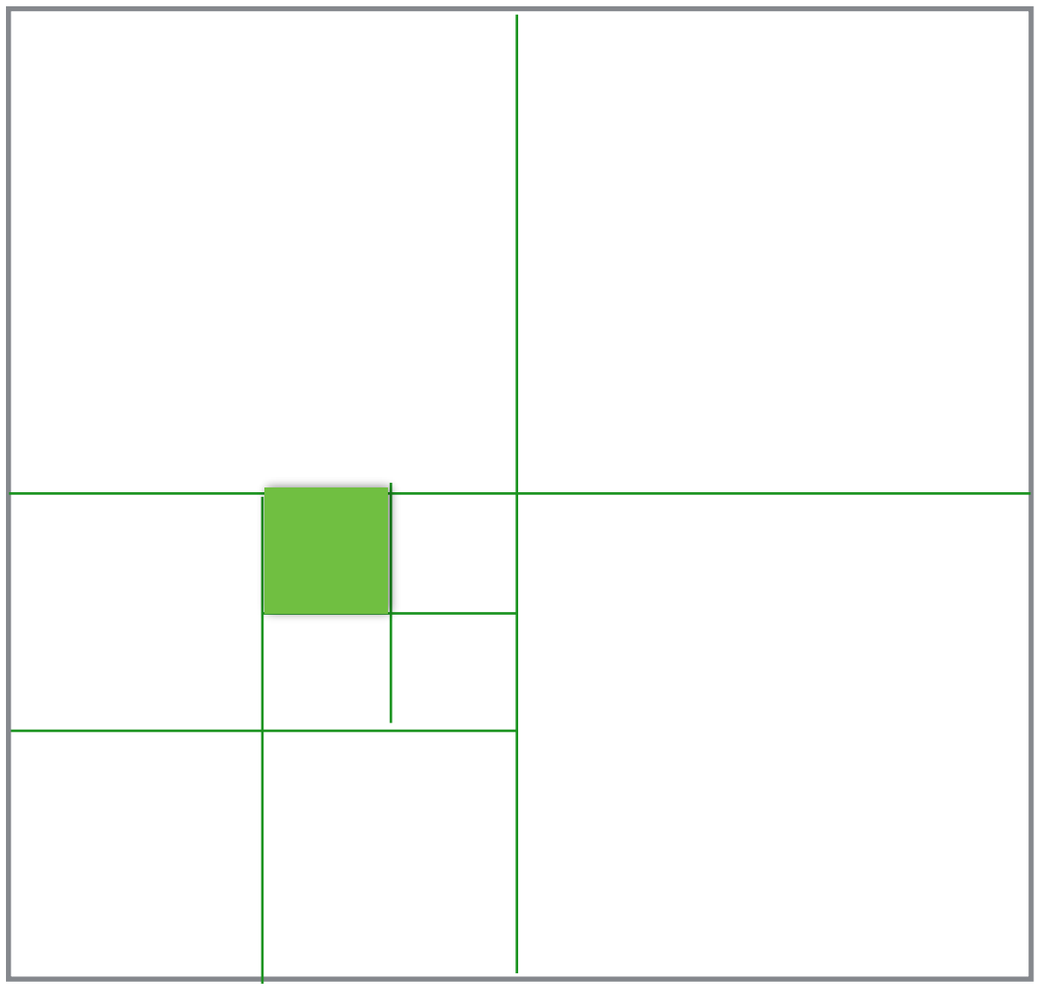
- try to find a node v' at the same depth as v
- if not possible, find the deepest

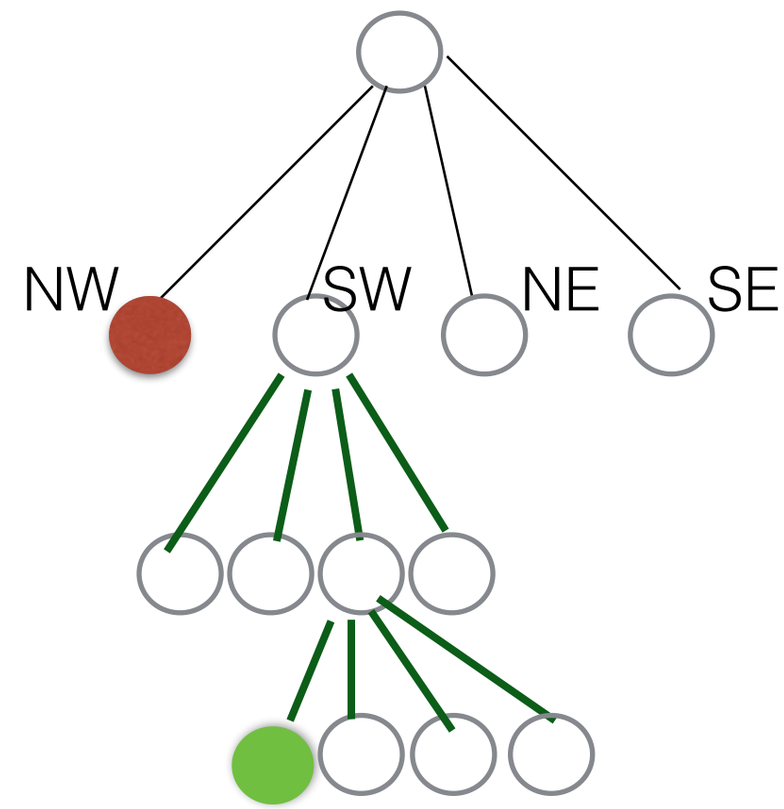
Come up with an example where the search for a North_neighbor is a great-uncle

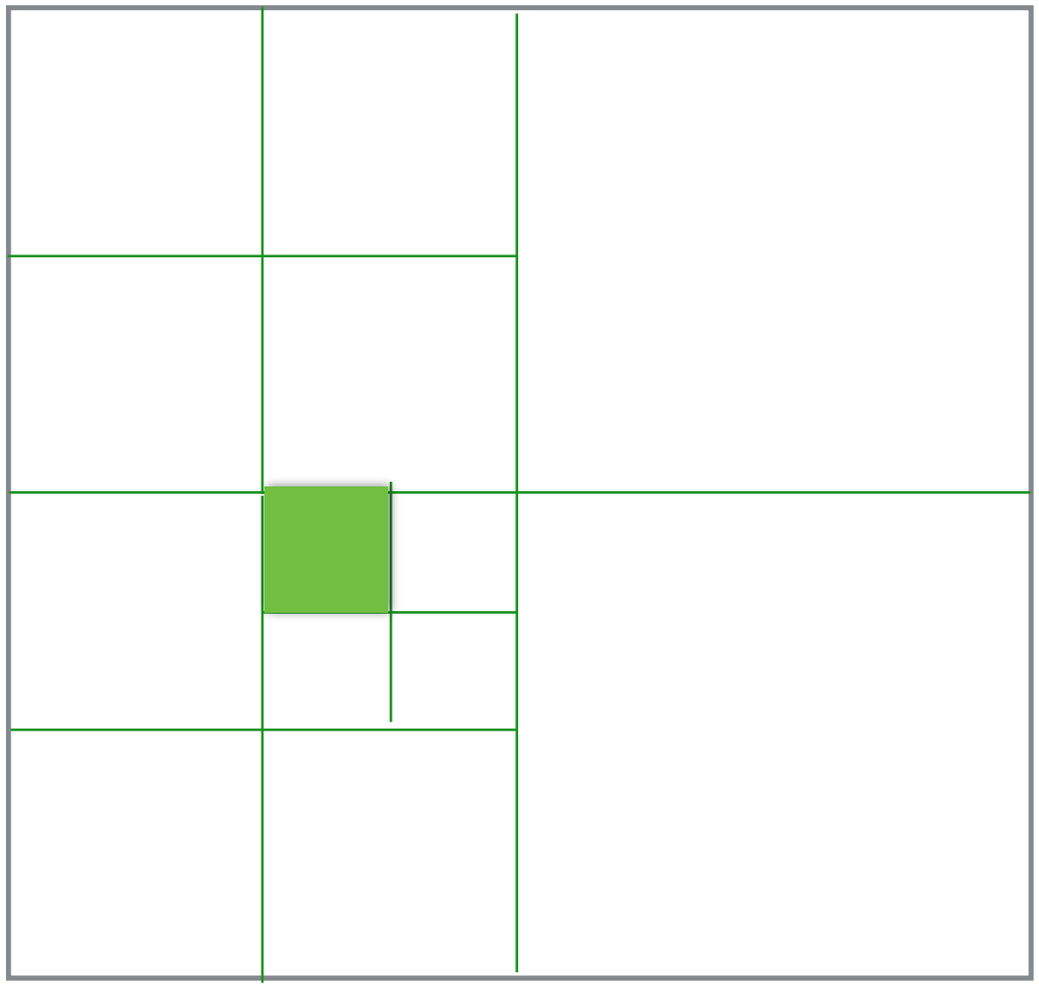
Example: Neighbor finding

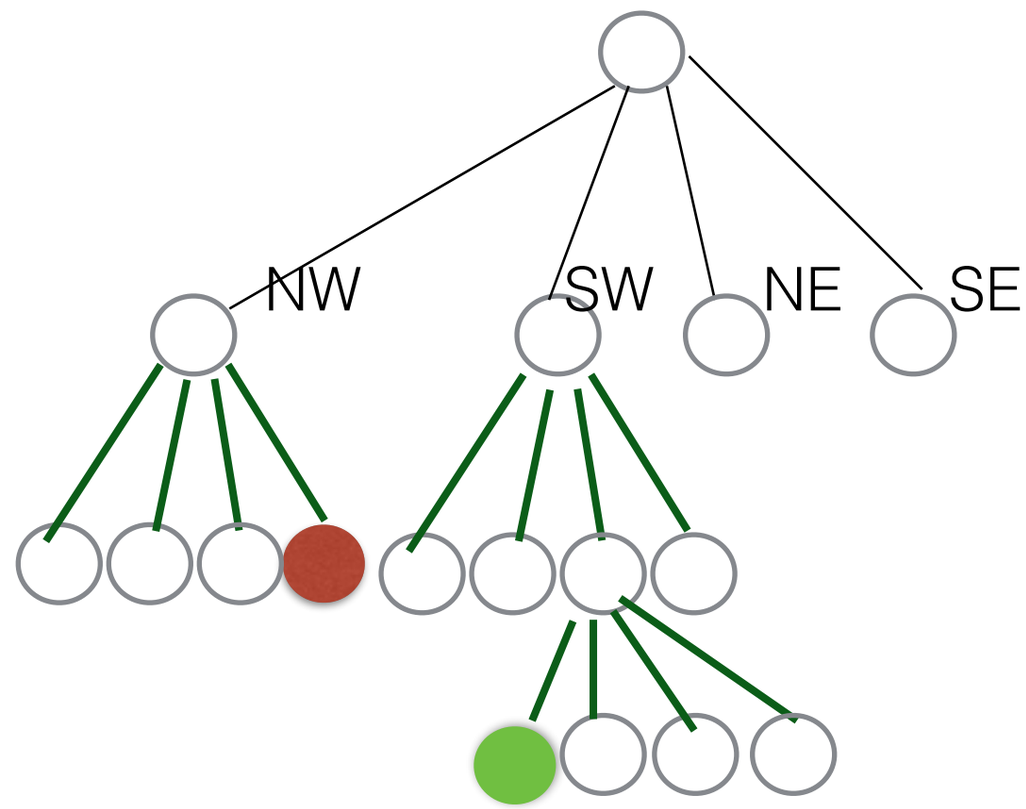
Come up with an example where the North_neighbor is a

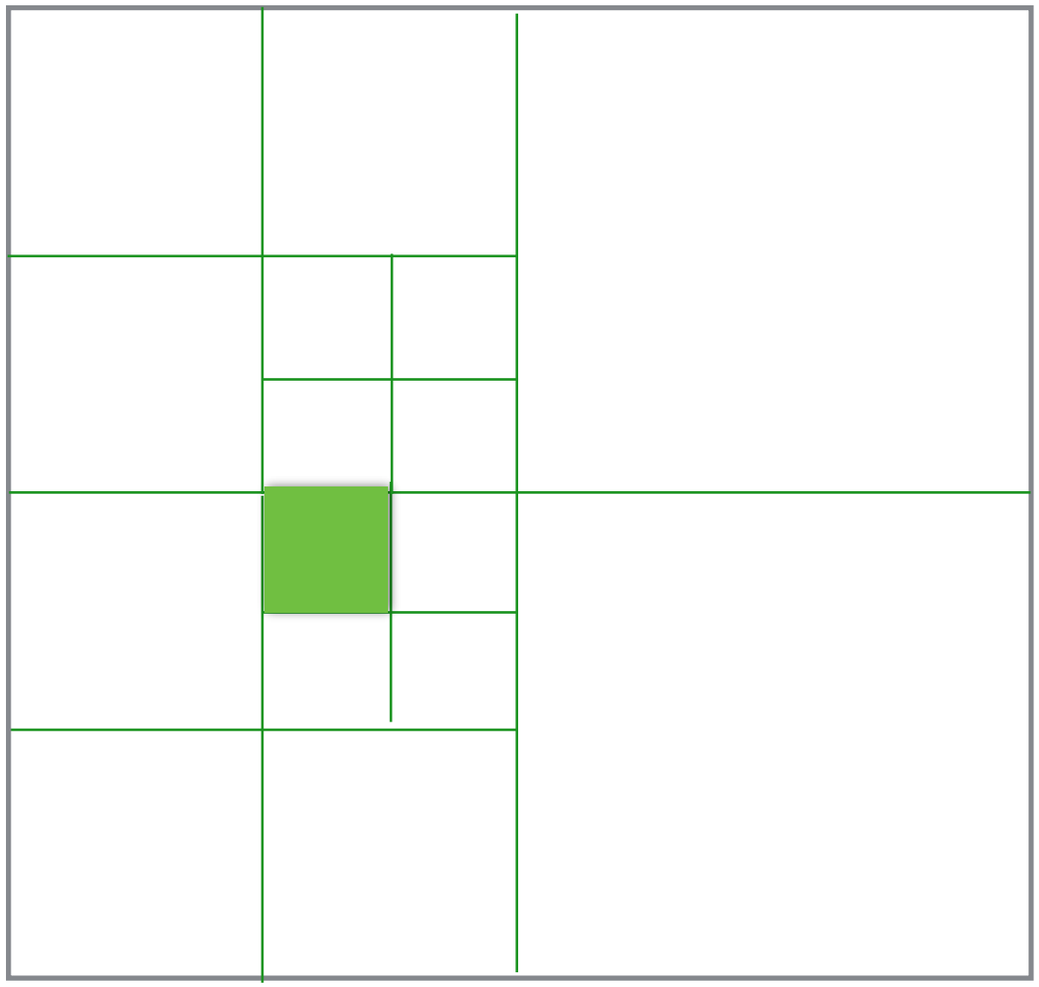
- great-uncle.
- great-great-uncle
- …

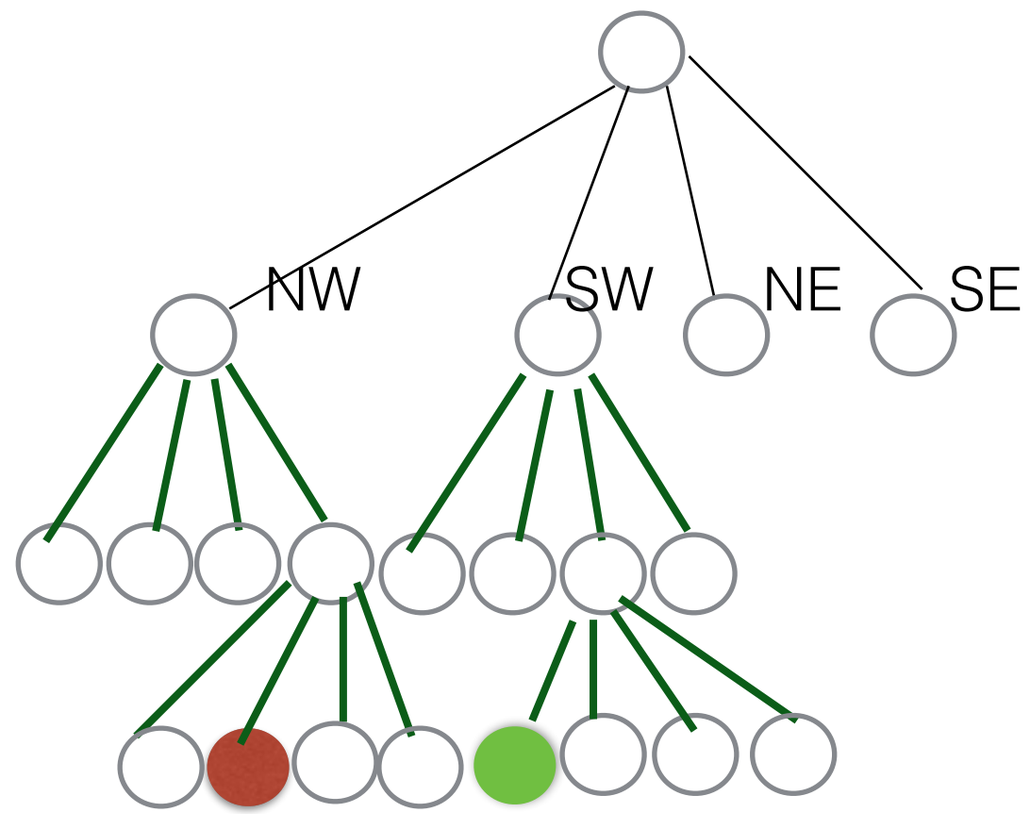



NW
SW
NE
SE




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)

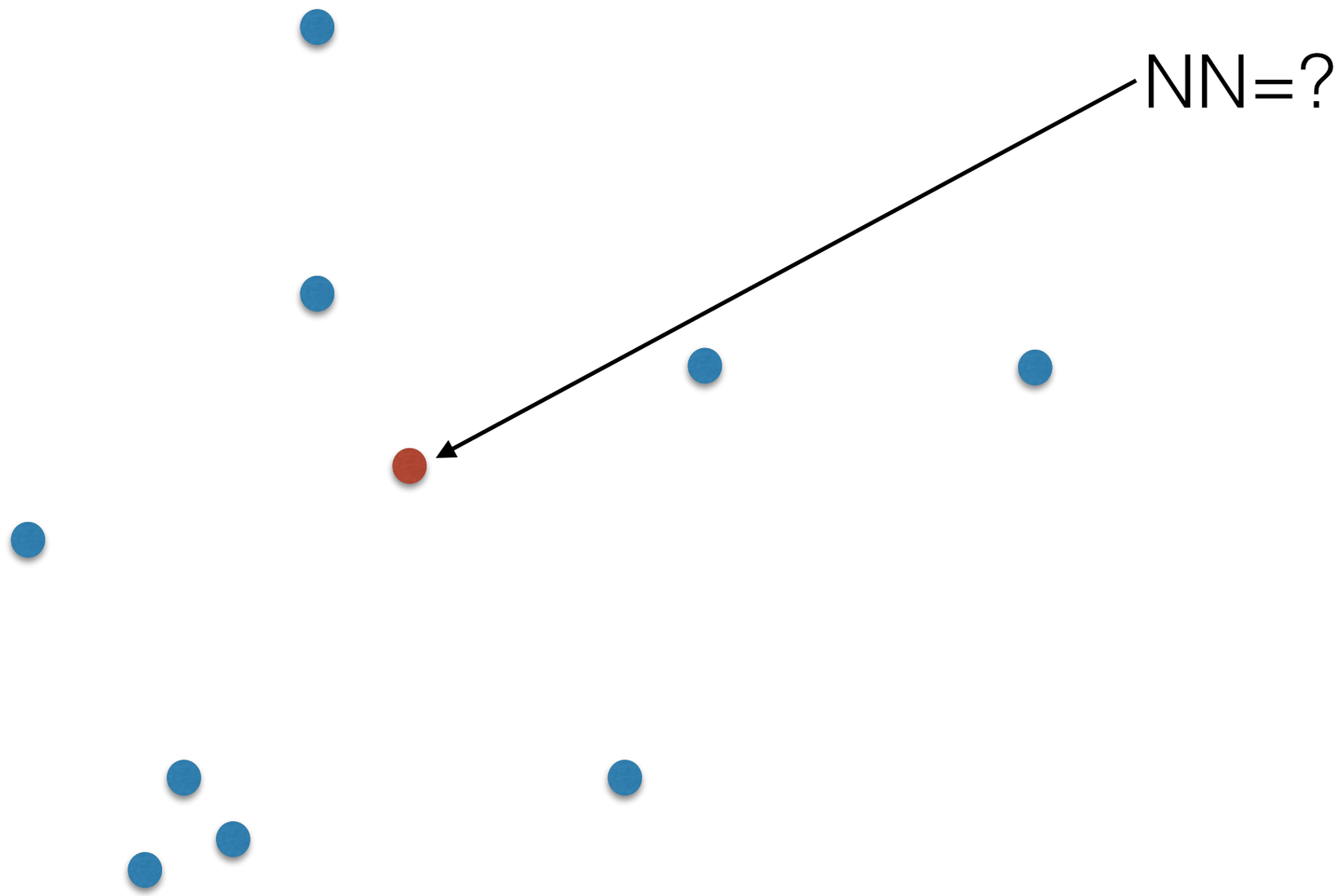
//if we reached here, v must be NW or NE child

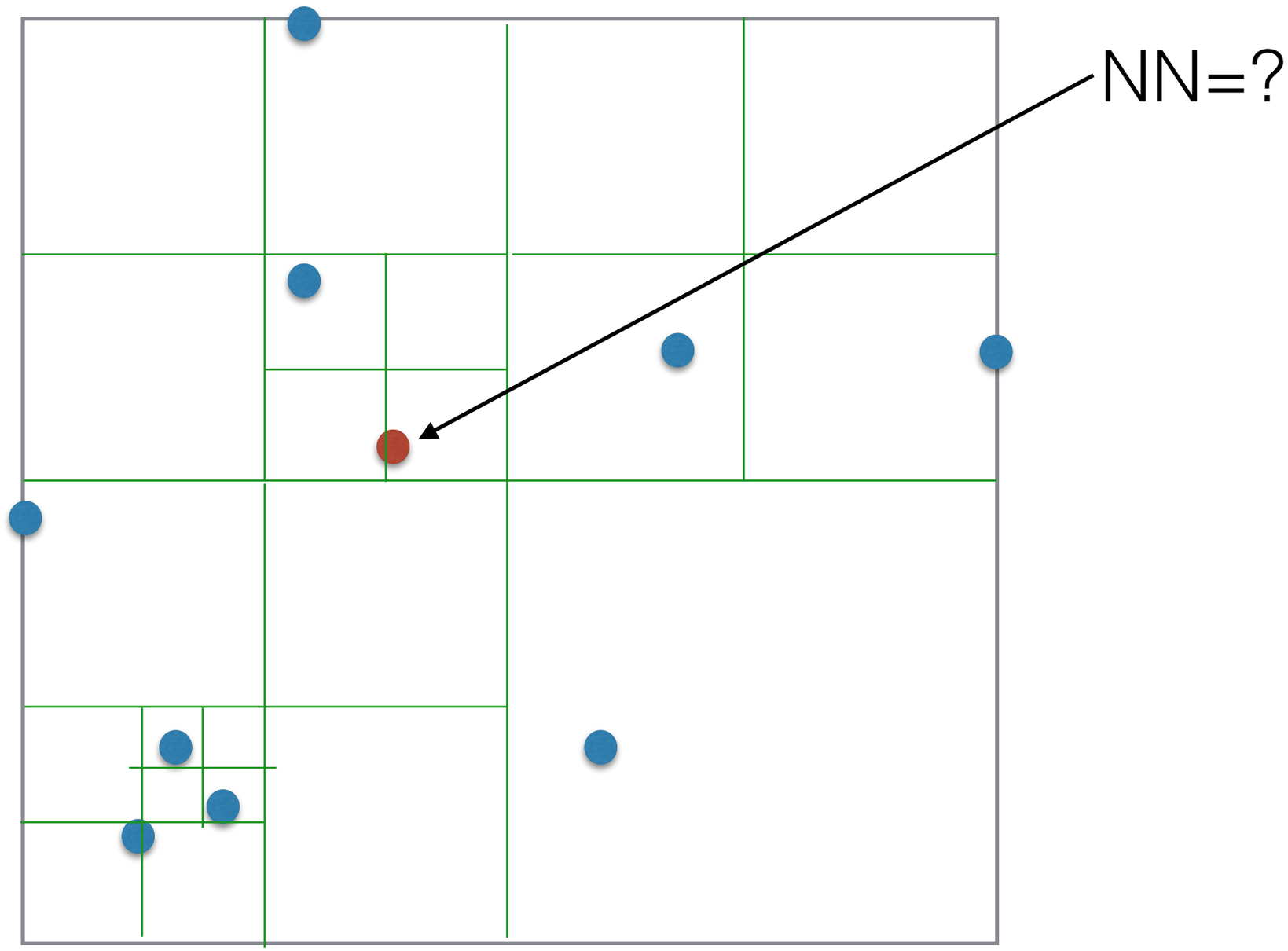
- x <—- North_Neighbor(parent(v)) ← give an example that would trigger several recursive calls
  - if x is NULL or a leaf: return x
  - else:
    - if v ==NW-child of parent(v): return SW-child(x)
    - else: return SE-child(x)

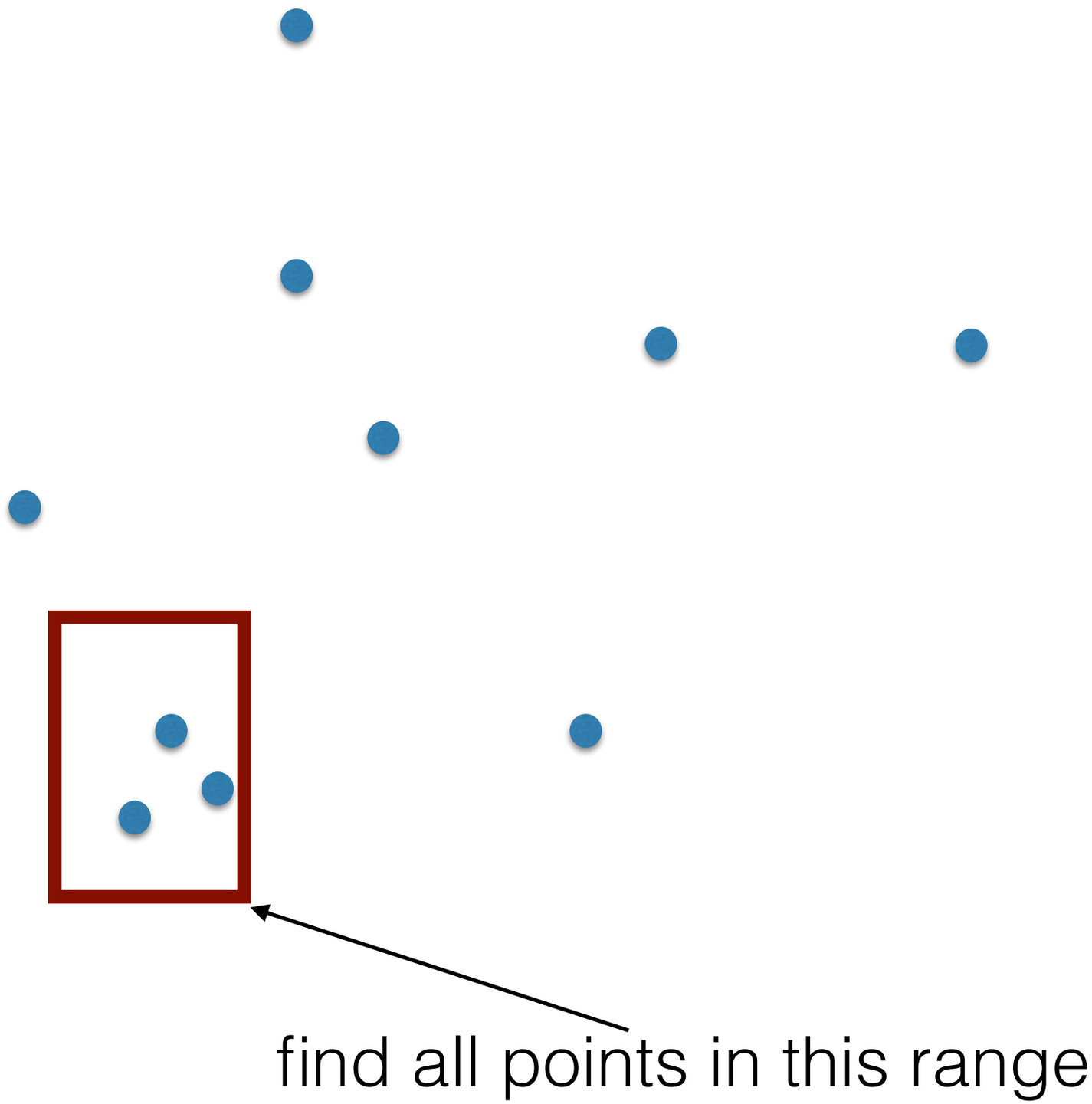
More applications

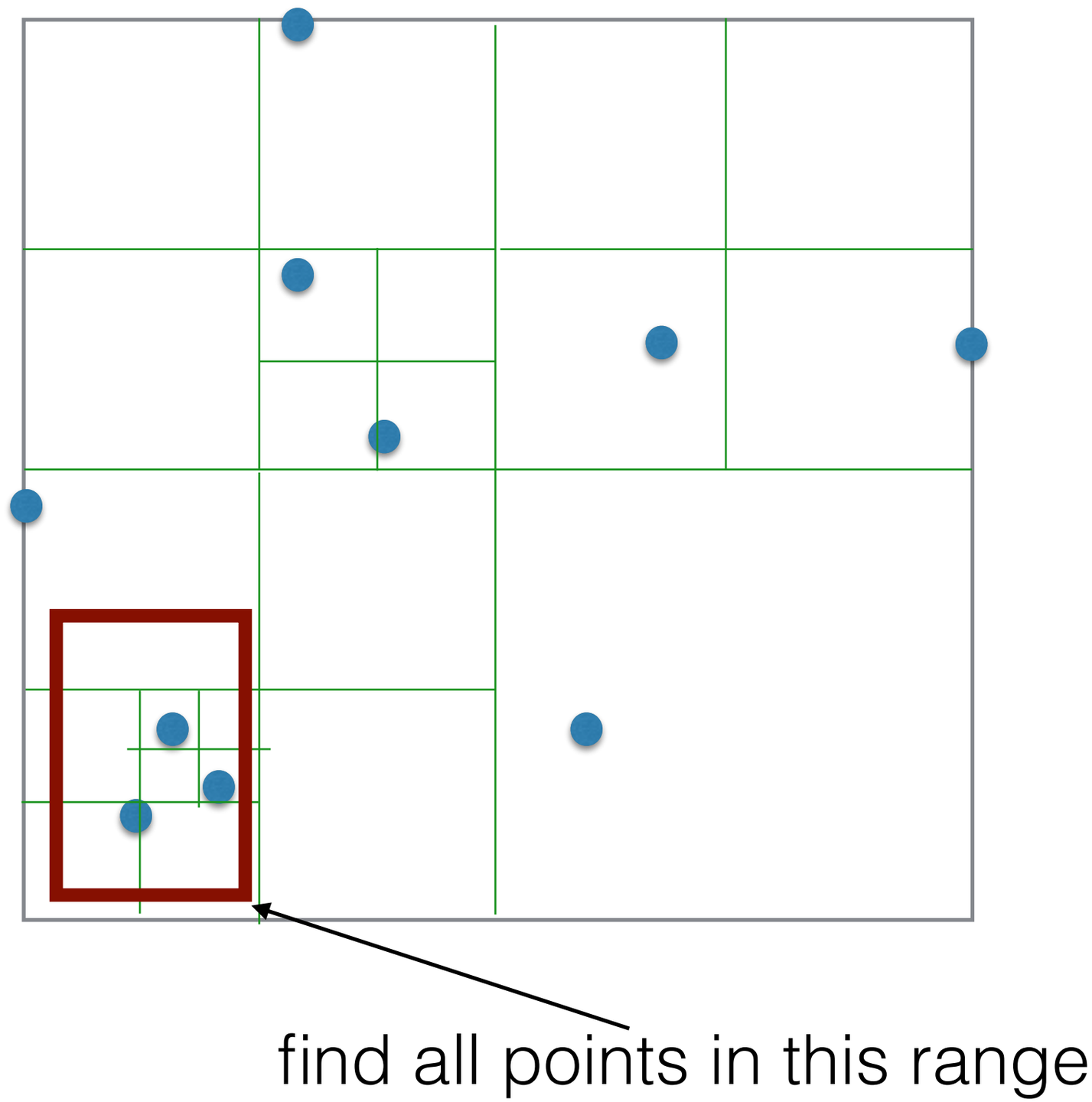
- Used to answer queries on spatial data such as:
  - point location
  - nearest neighbor (NN)
  - k-NNs
  - range searching
  - find all segments intersecting a given segment
  - meshing
  - …

How would you do these?


NN=?

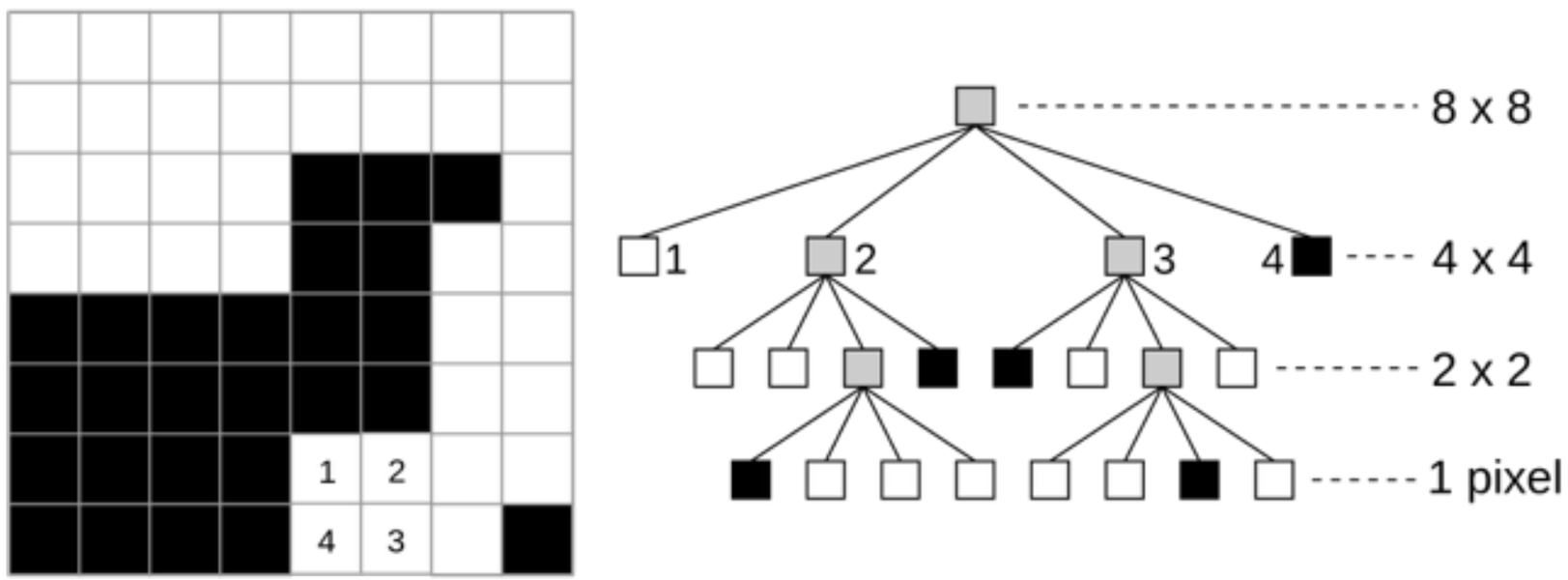

NN=?


find all points in this range


find all points in this range

Applications

- Image analysis/compression



Applications

- Used for fast rendering (LOD)
  - Store data at various levels of detail, using a quadtree
    - Bottom level has full resolution, level above it has lower resolution, and so on
    - This can be done so that the total amount of data stored is still O(n) (that is, no blowup due to storing multiple levels)
  - Render scene at a resolution dependent on its distance from the viewpoint
    - when rendering an object, select the appropriate level based on its distance from viewpoint

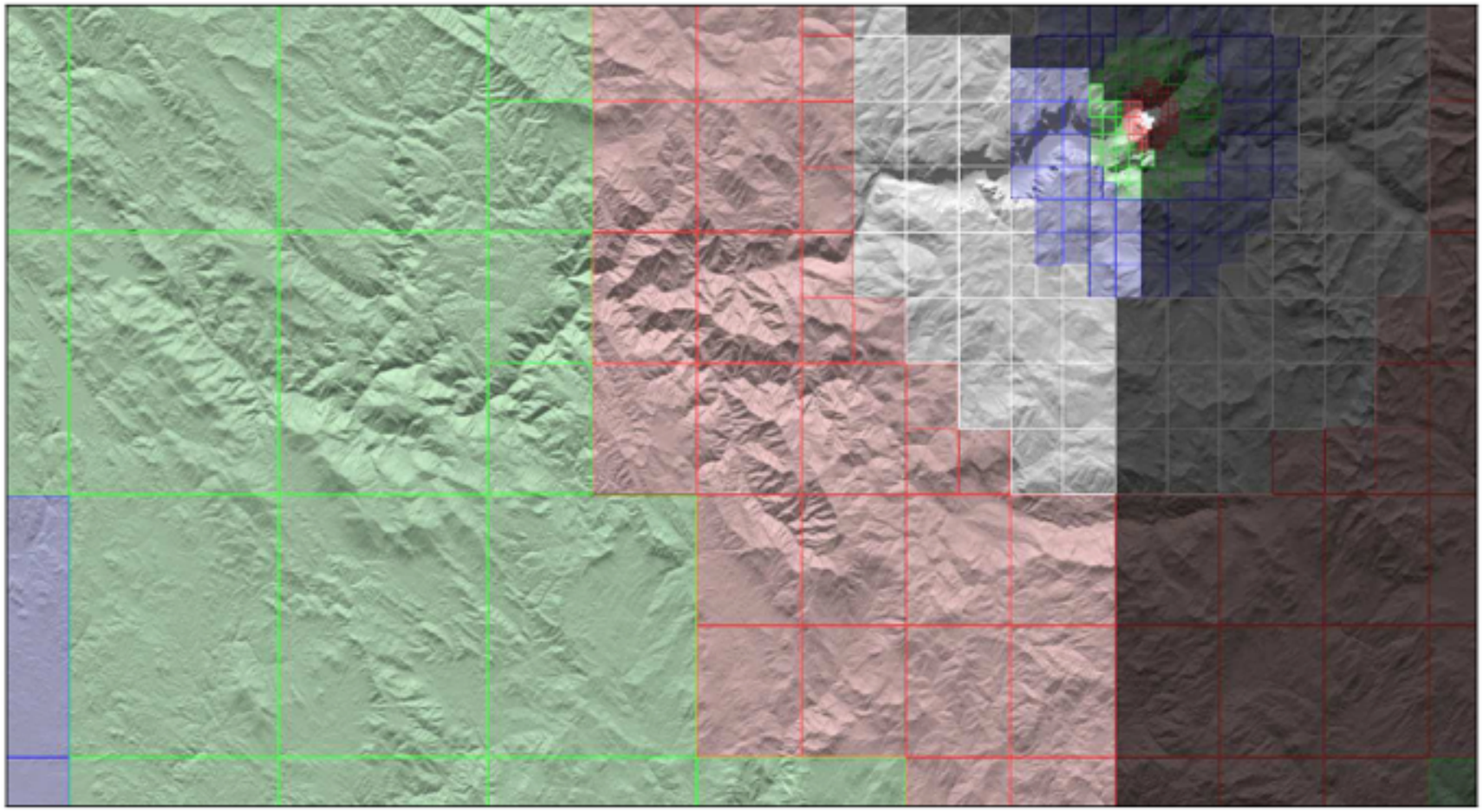


Figure 3 *LOD selection of quadtree nodes (the frustum culled section is shaded in dark).*

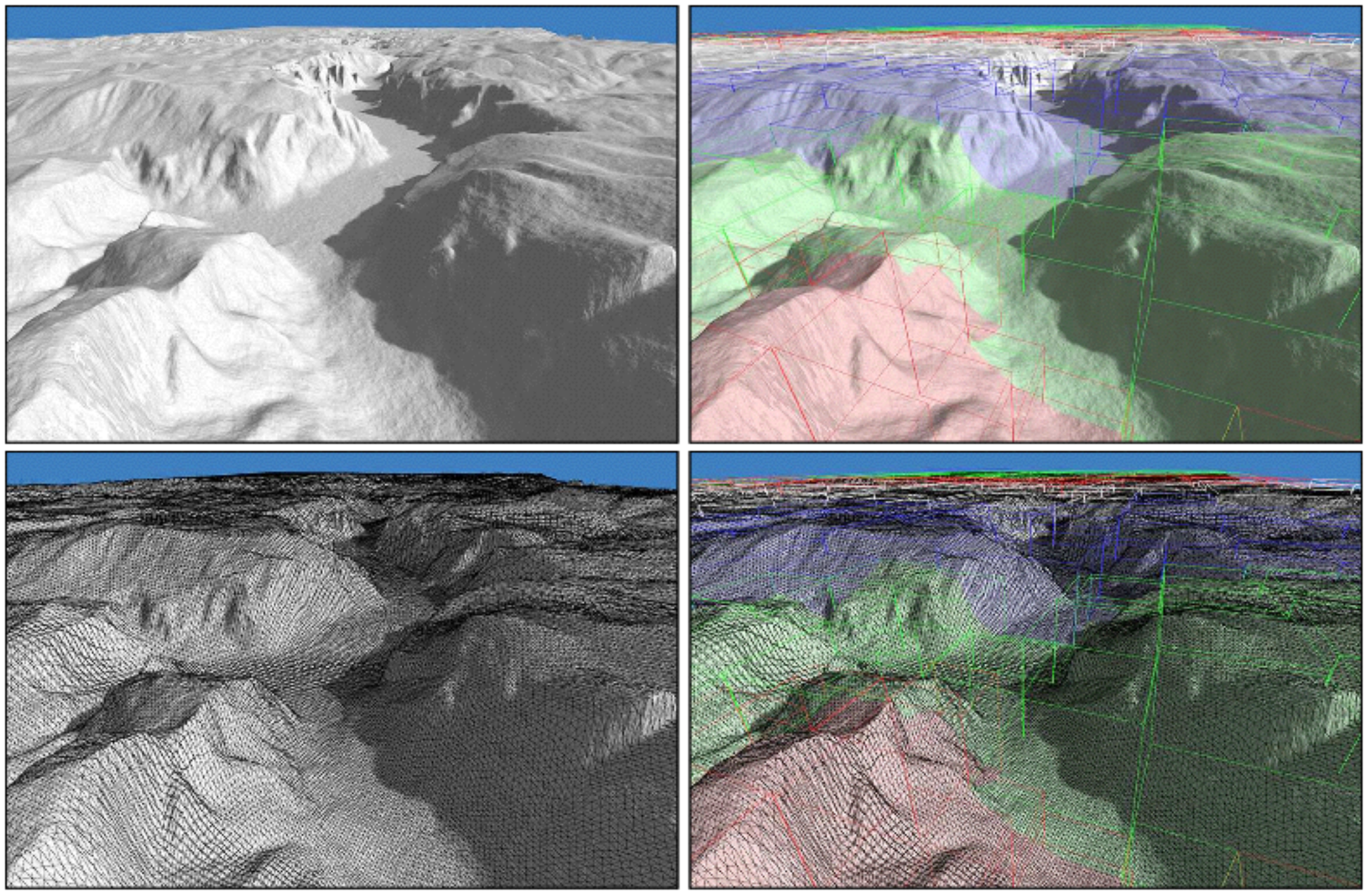


Figure 5 *Distribution of LOD levels and nodes (different colors represent different layers).*