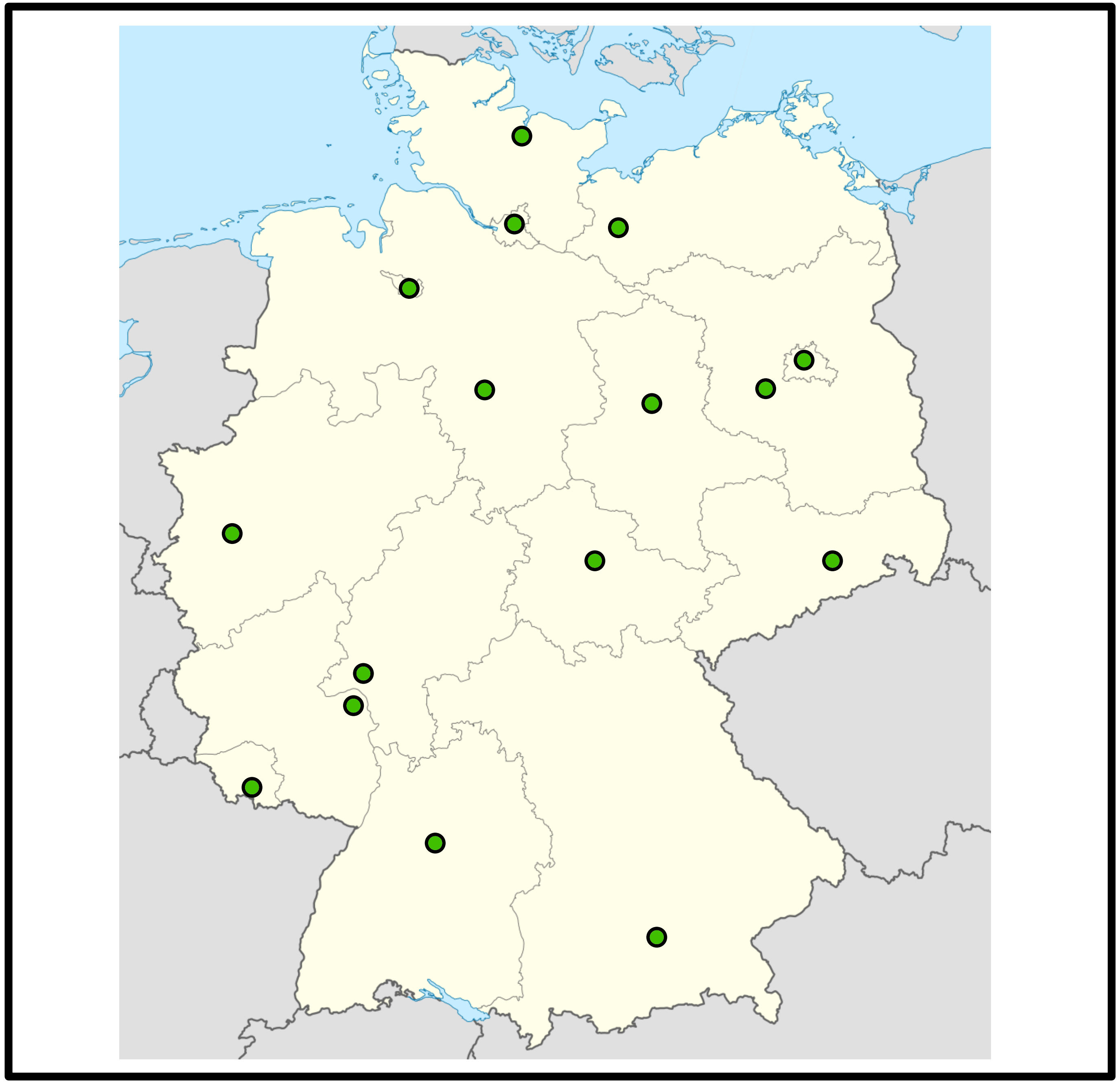
Quadtrees

Geometric Approximation Algorithms

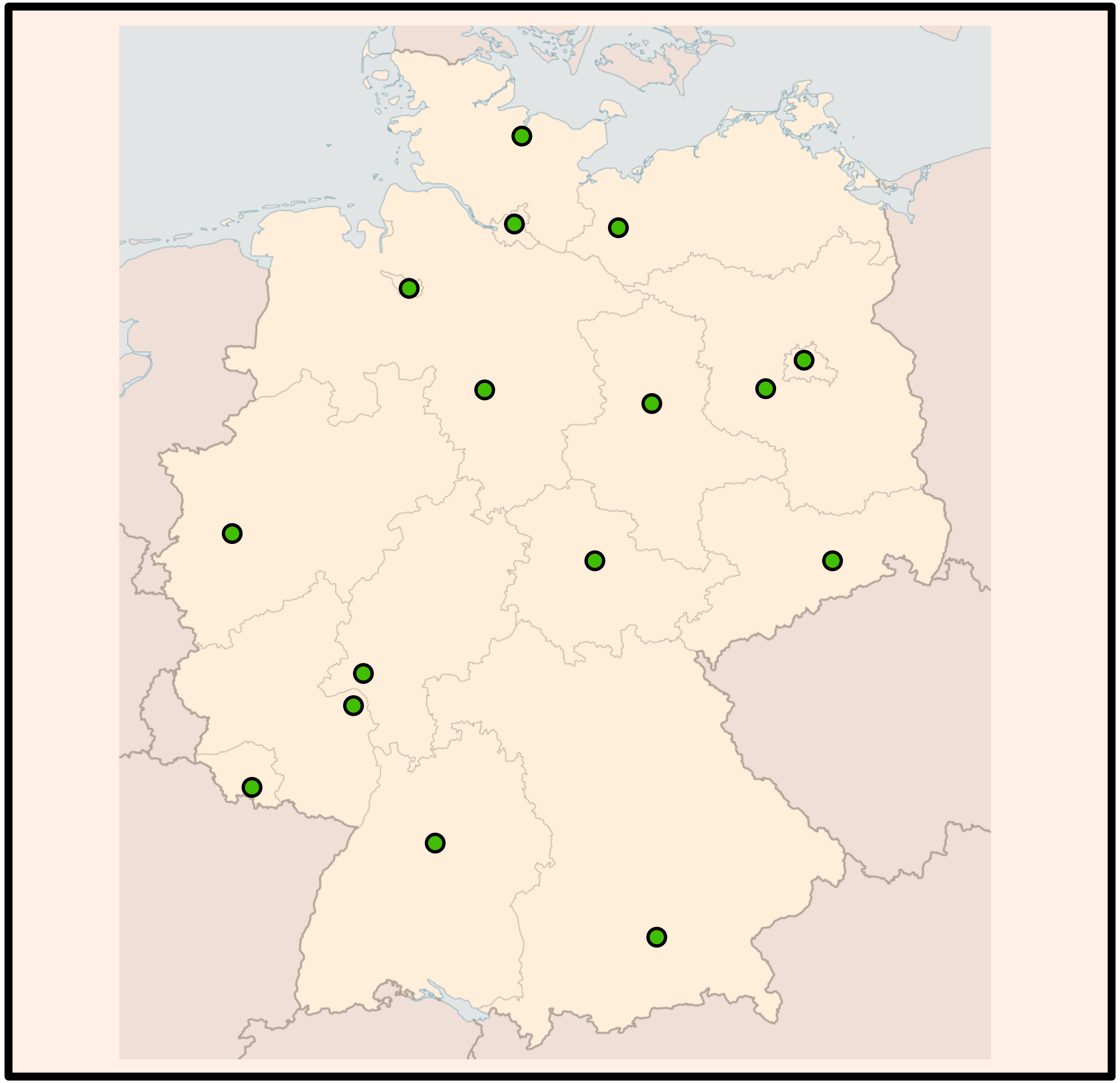
Quadtrees: a simple point-location data structure



Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

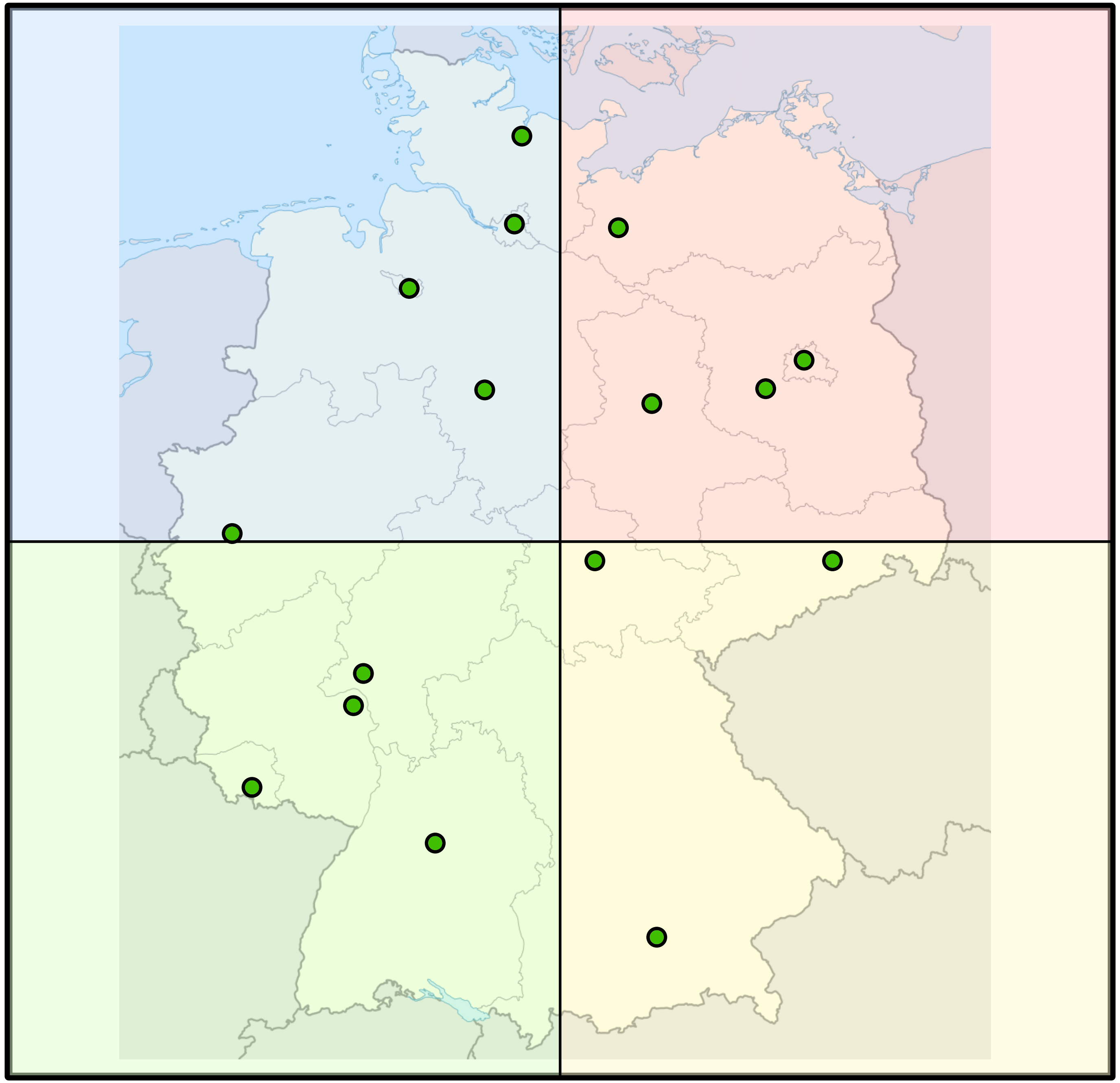
Quadtrees: a simple point-location data structure

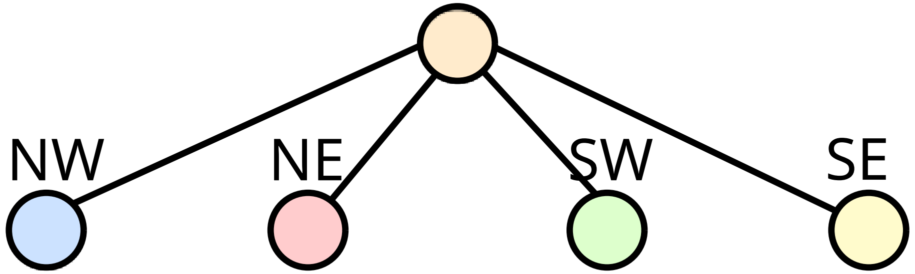


Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

Quadtrees: a simple point-location data structure

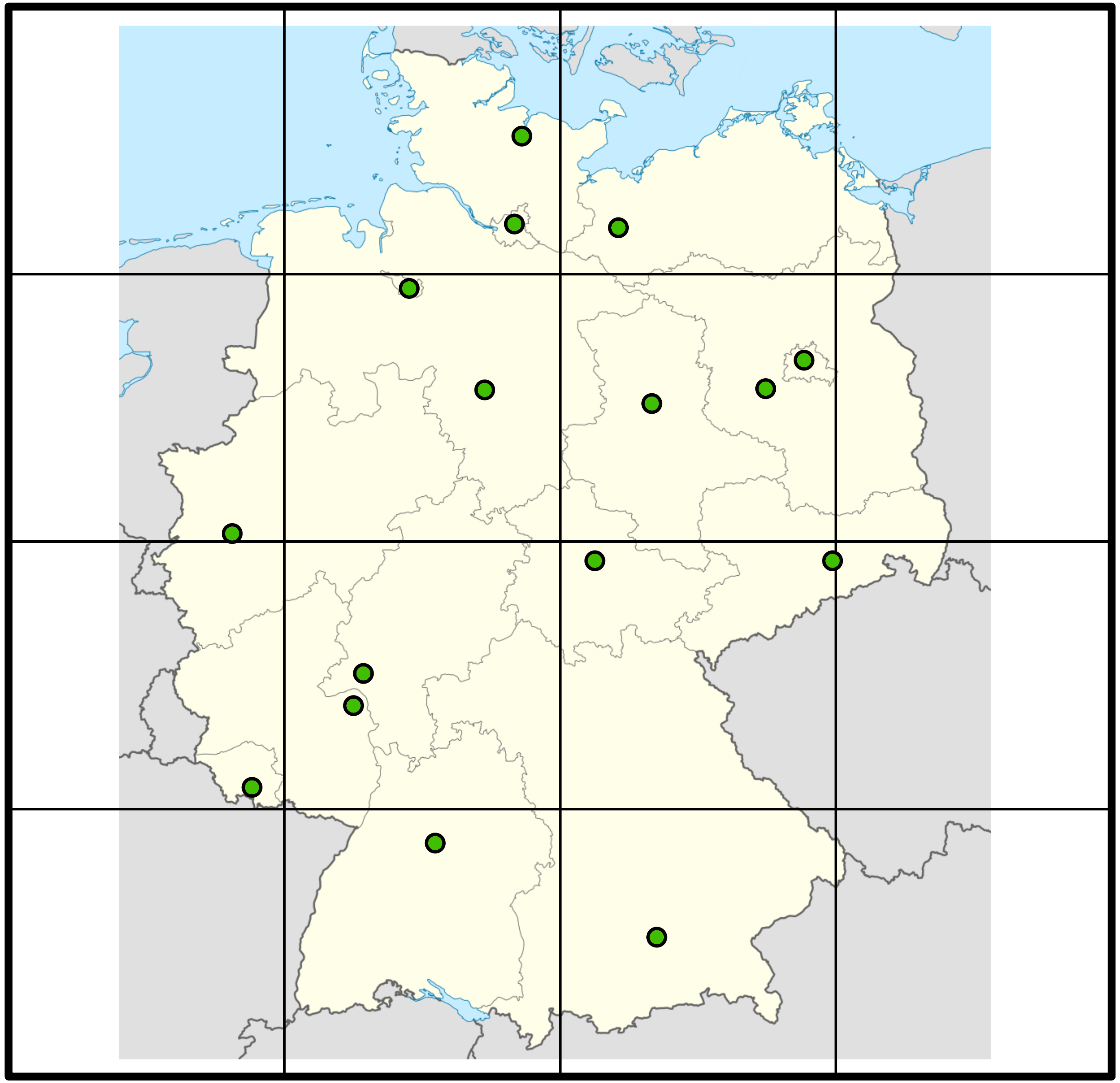


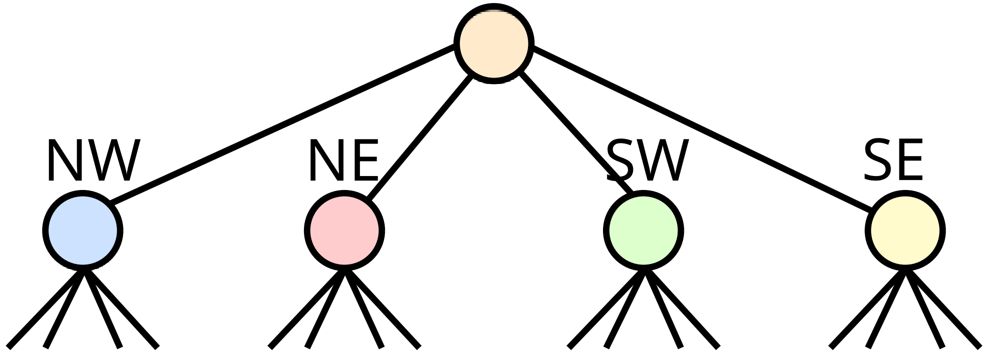


Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

Quadtrees: a simple point-location data structure

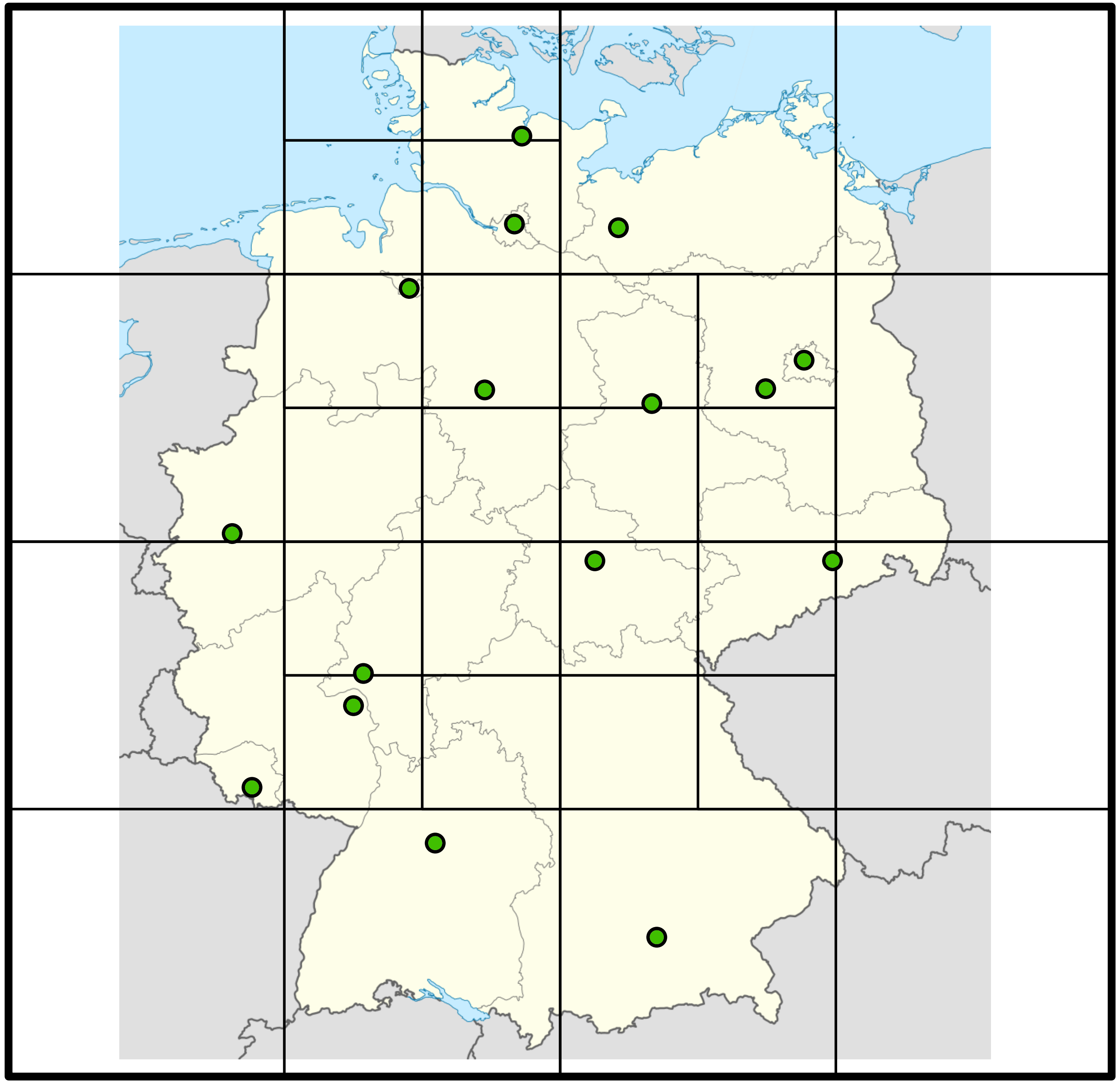


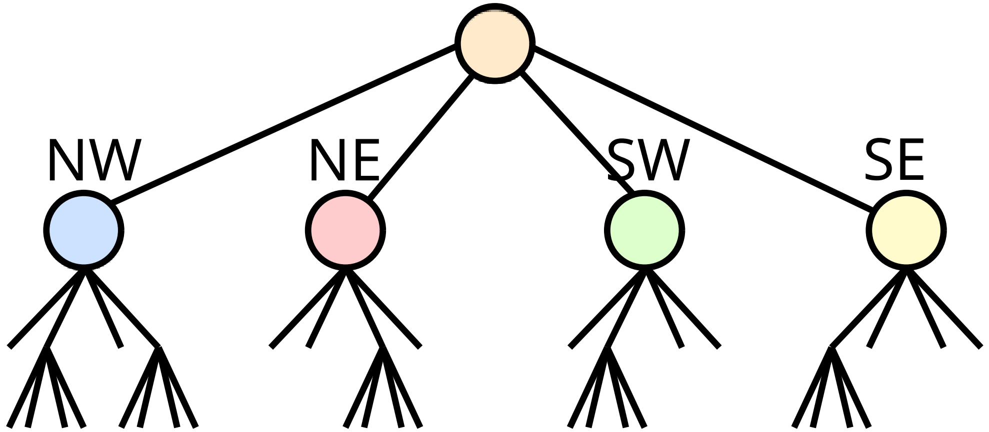


Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

Quadtrees: a simple point-location data structure

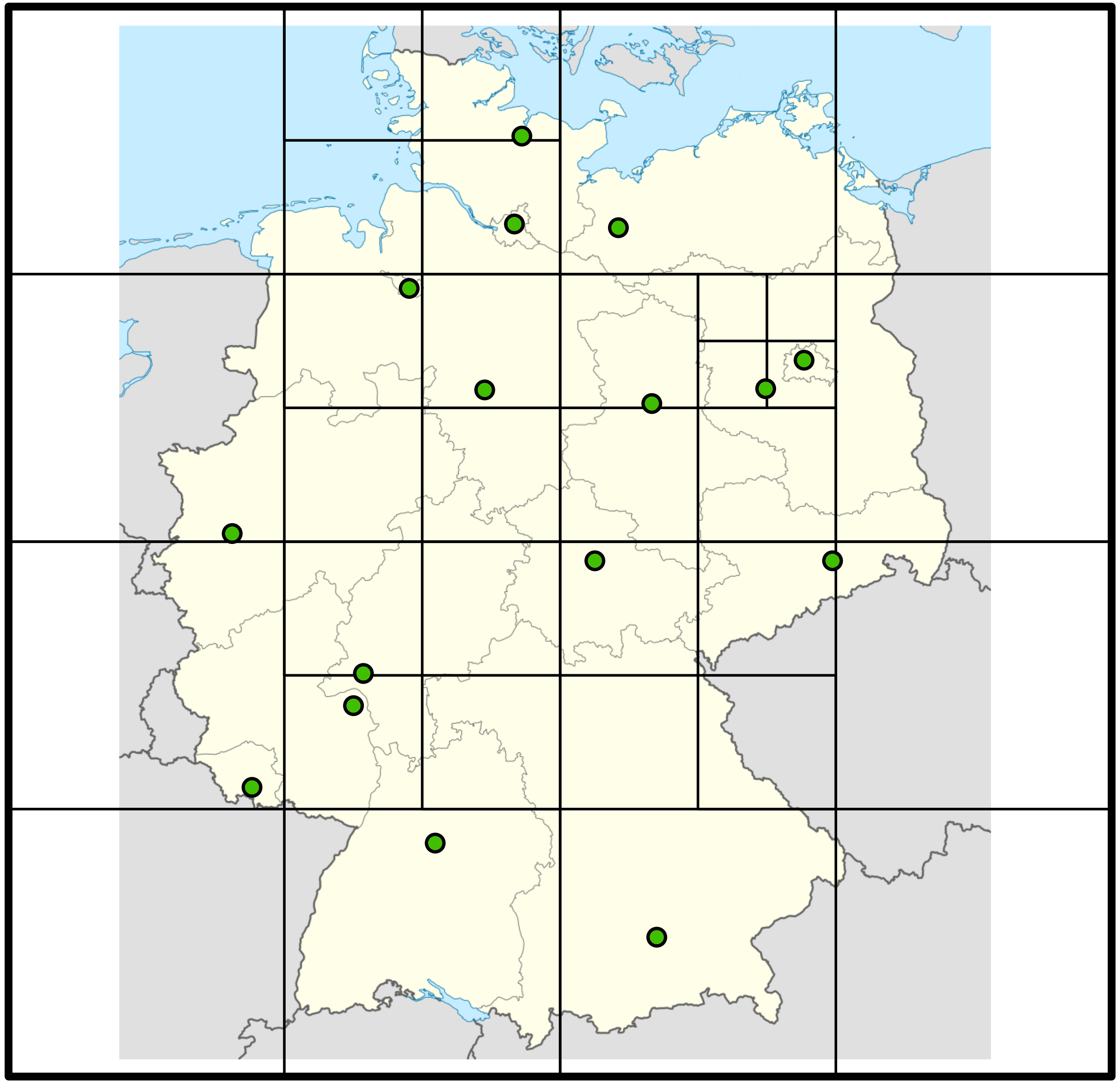


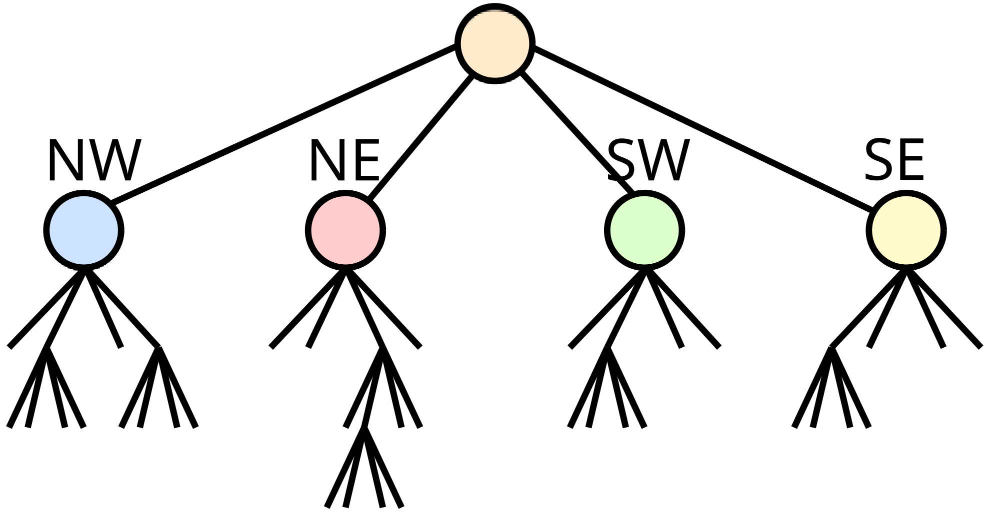


Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

Quadtrees: a simple point-location data structure

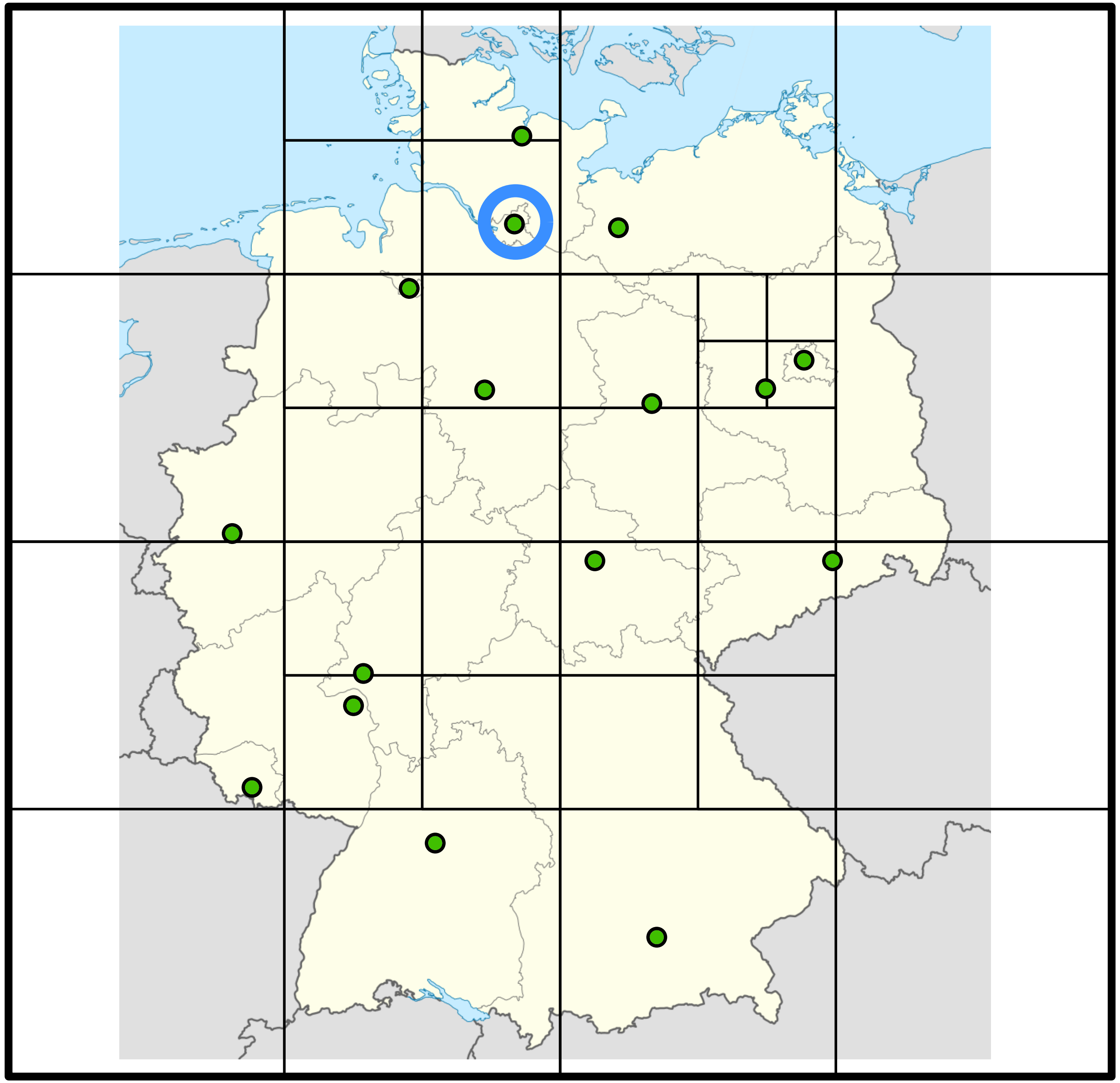


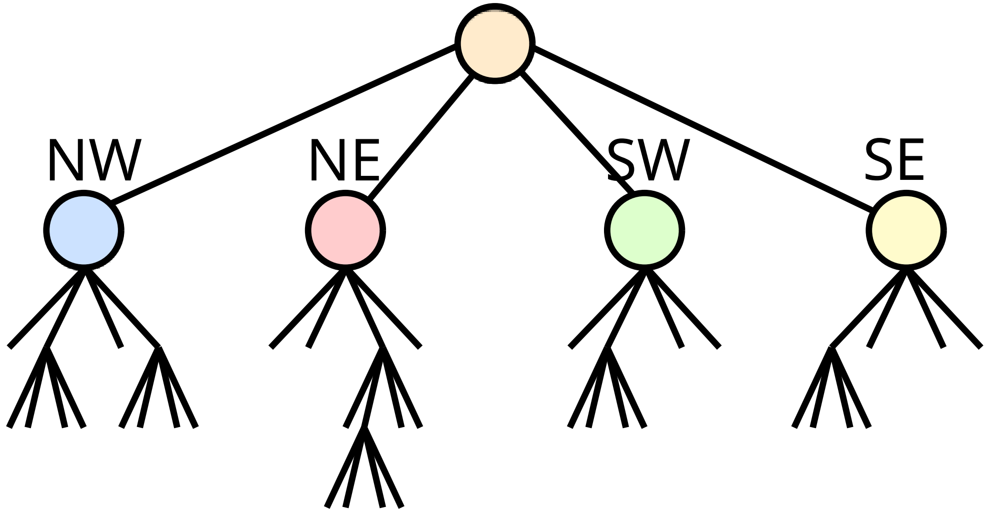


Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

Quadtrees: a simple point-location data structure

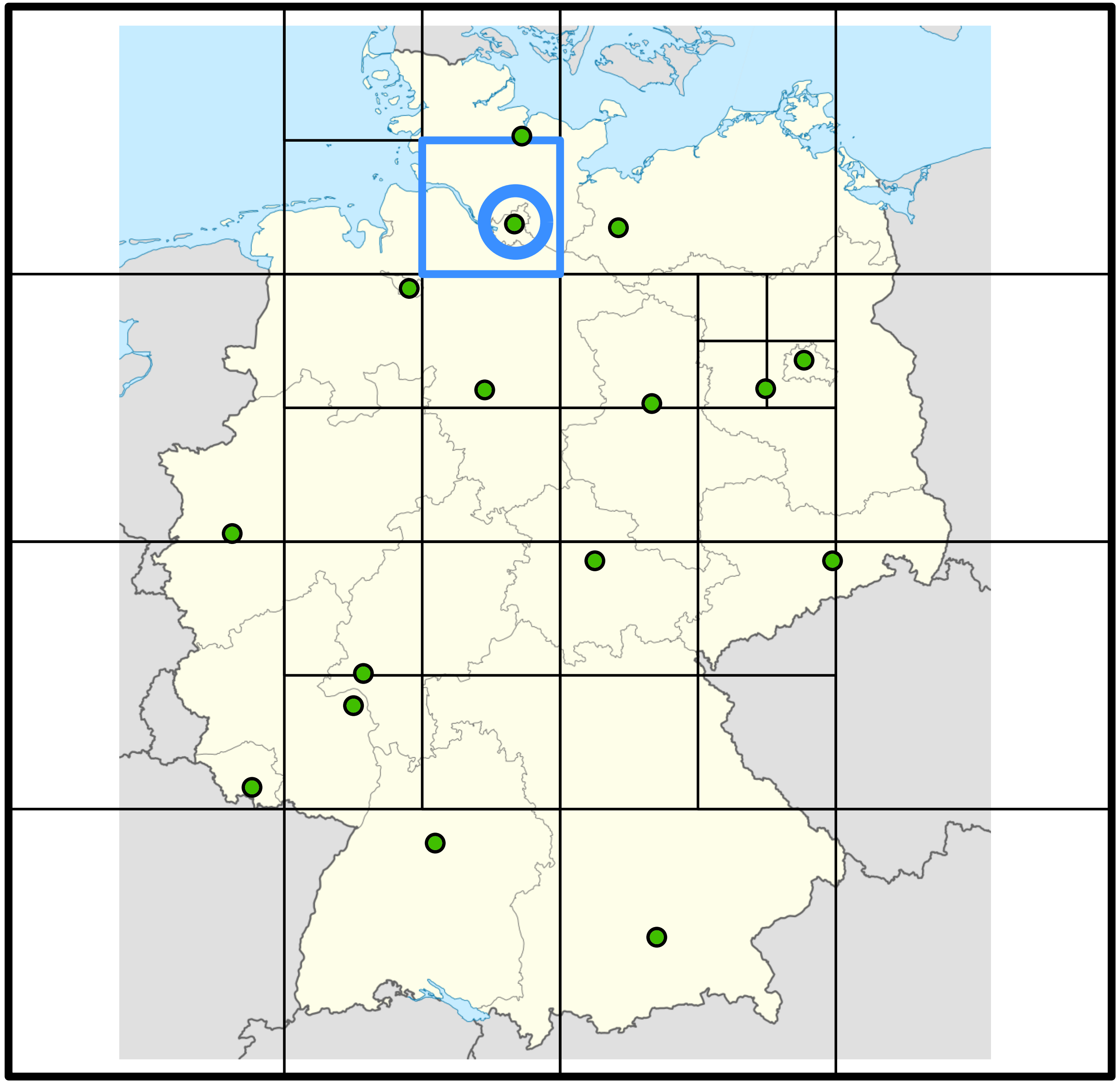


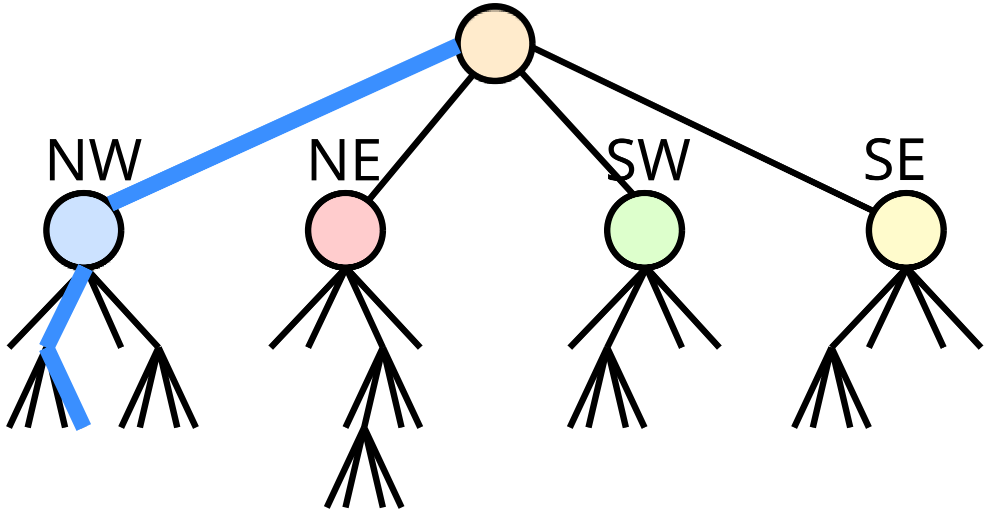


Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

Quadtrees: a simple point-location data structure

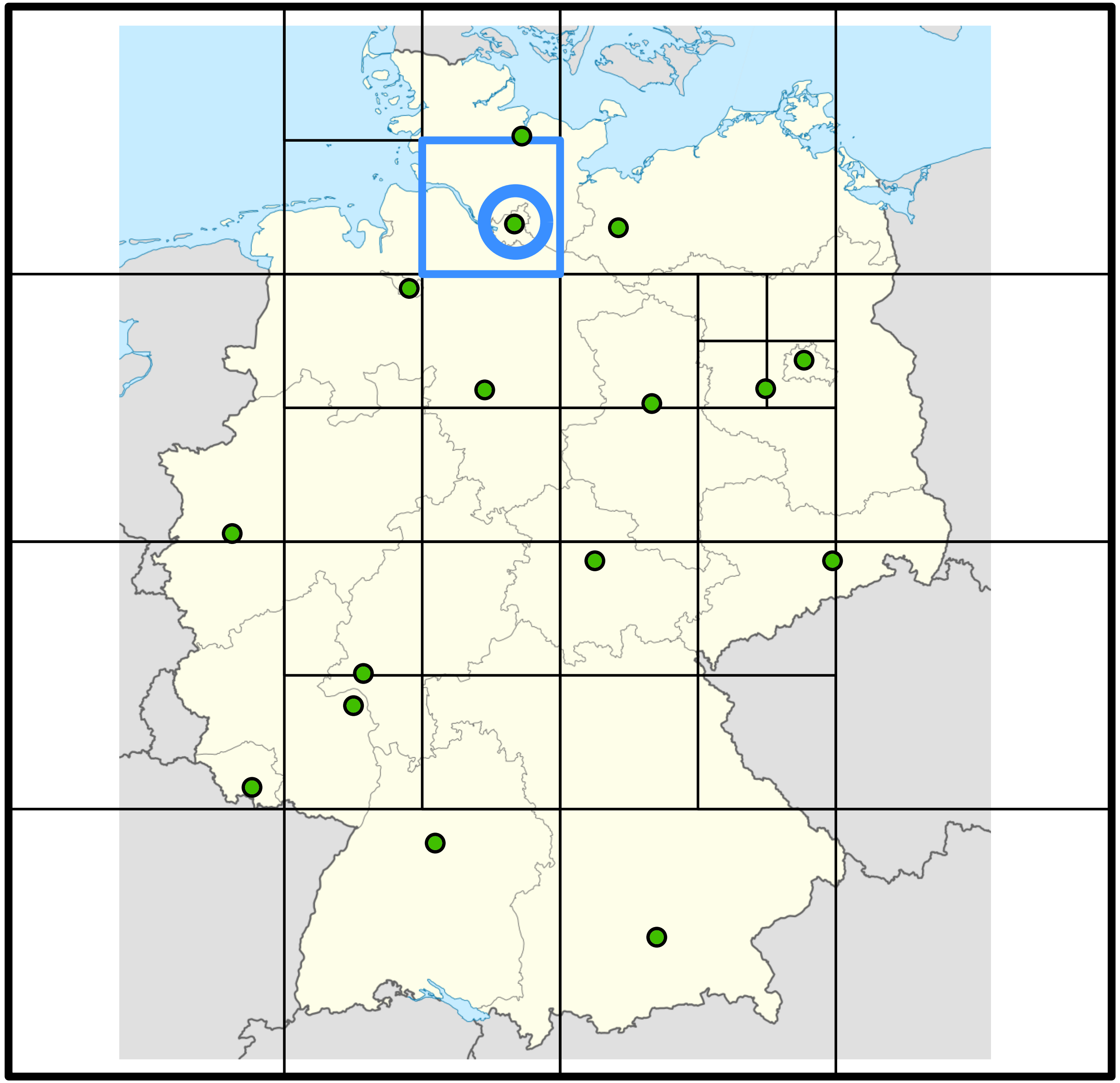


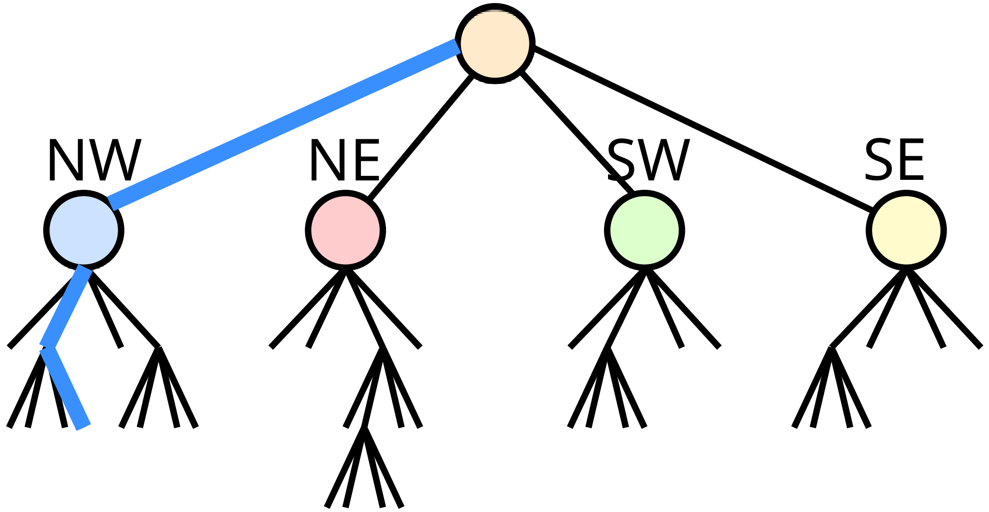


Main idea: Use a **tree structure** to be able to quickly find location of point

- nodes represent squares
- recursively subdivide squares into 4 until 1 point per square

Quadtrees: a simple point-location data structure





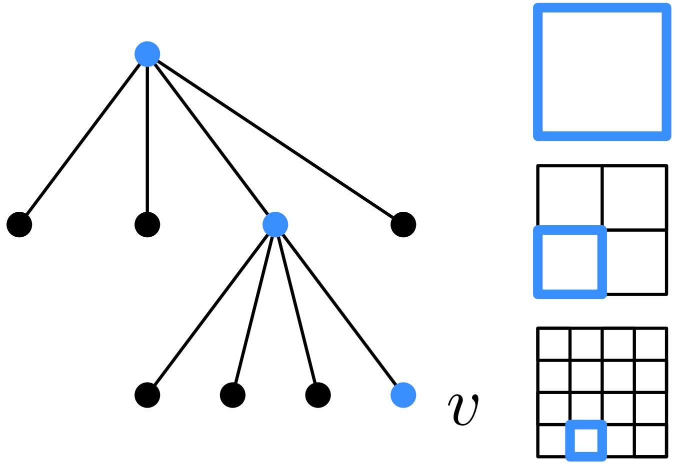
Simple point location: $O(d)$
for n points and quadtree of depth d

(+ check regions overlapping with leaf square)

Quadtrees: multi-scale grids

node v at depth i:

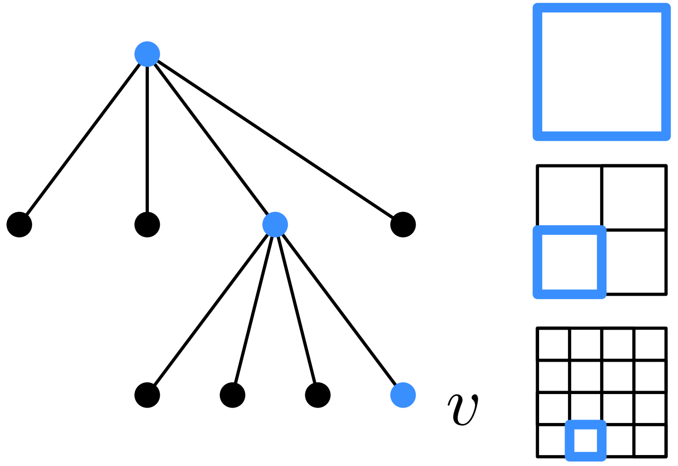
- square $S_v \rightarrow$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v



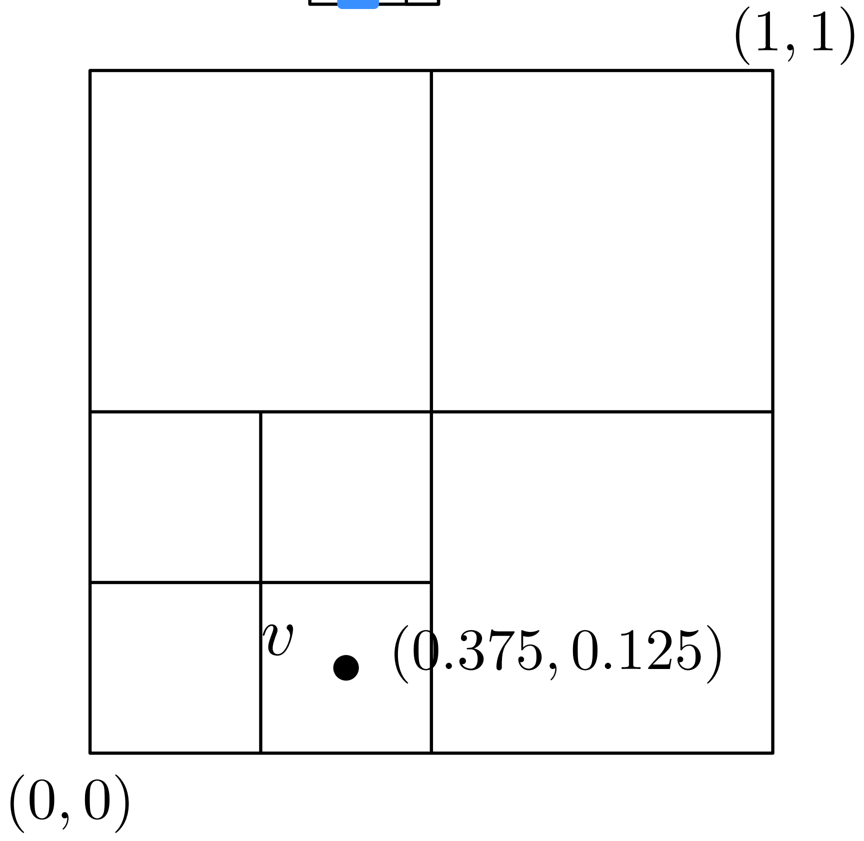
Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \rightarrow$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v



Quiz What is $id(v)$ in this example?



A (-1,2,1)

B (-1,3,4)

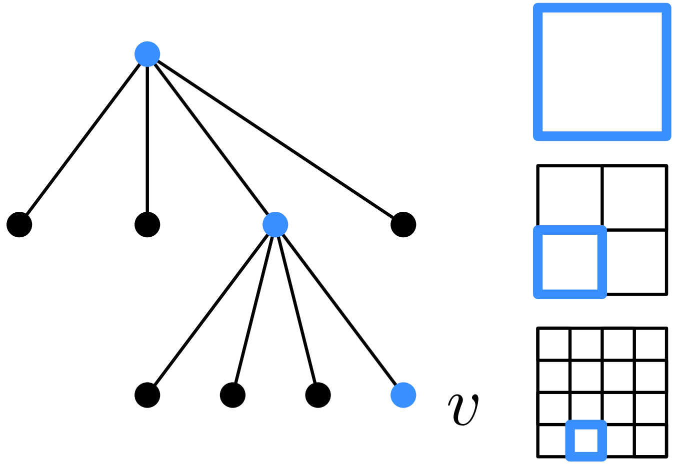
C (-2,2,2)

D (-2,1,0)

Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \rightarrow$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v



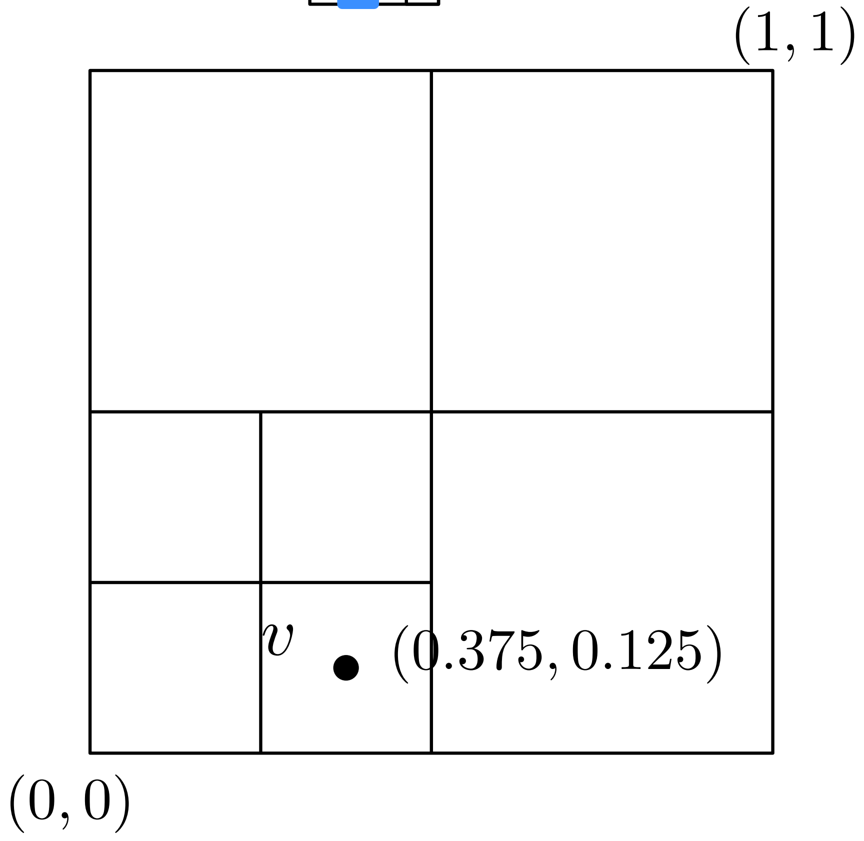
Quiz   What is $id(v)$ in this example?

A (-1,2,1)

B (-1,3,4)

C (-2,2,2)

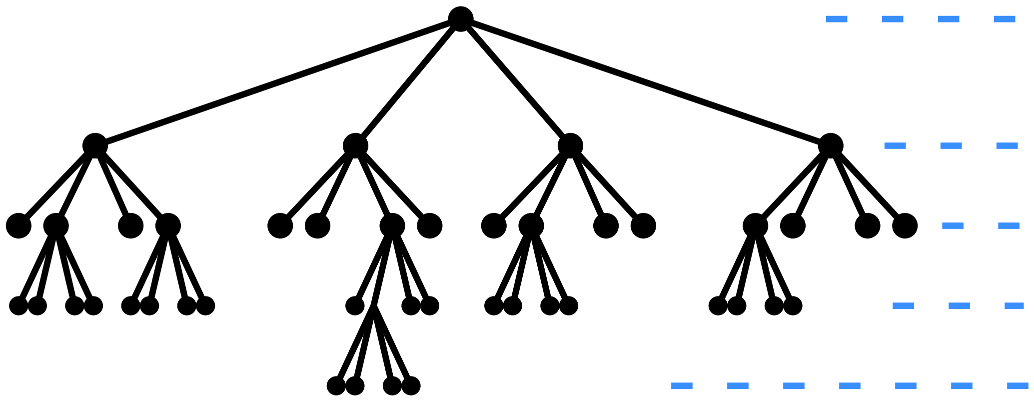
D (-2,1,0)

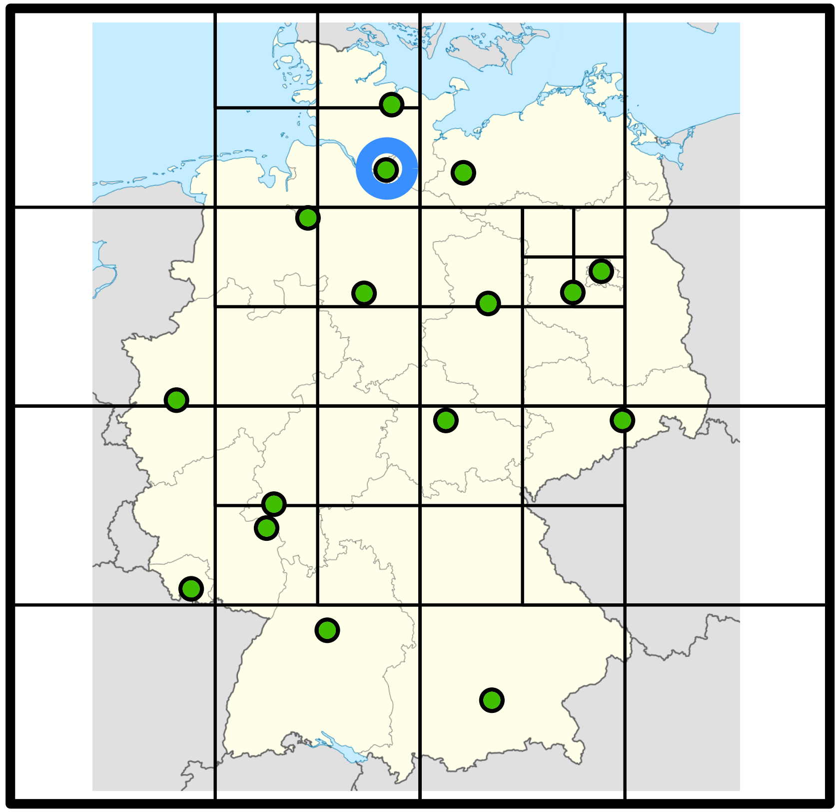


Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \to$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v





Faster point location:

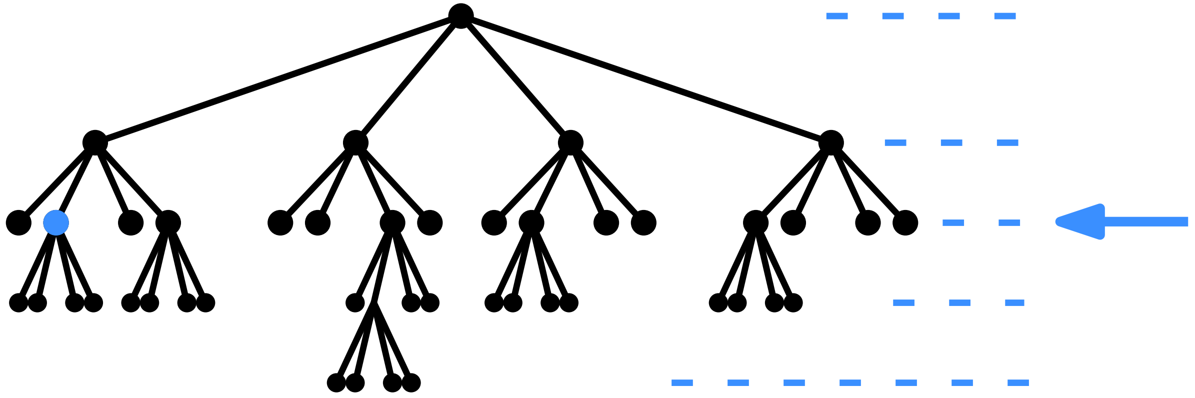
preprocessing: build hash table using $id(v)$

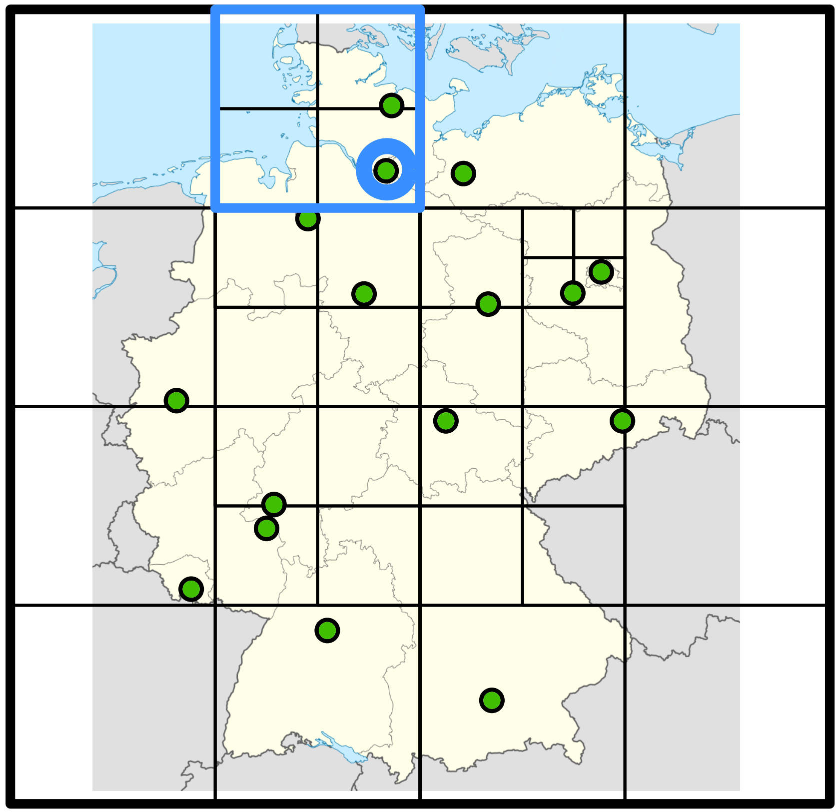
query: binary search on levels

Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \rightarrow$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v





Faster point location:

preprocessing: build hash table using $id(v)$

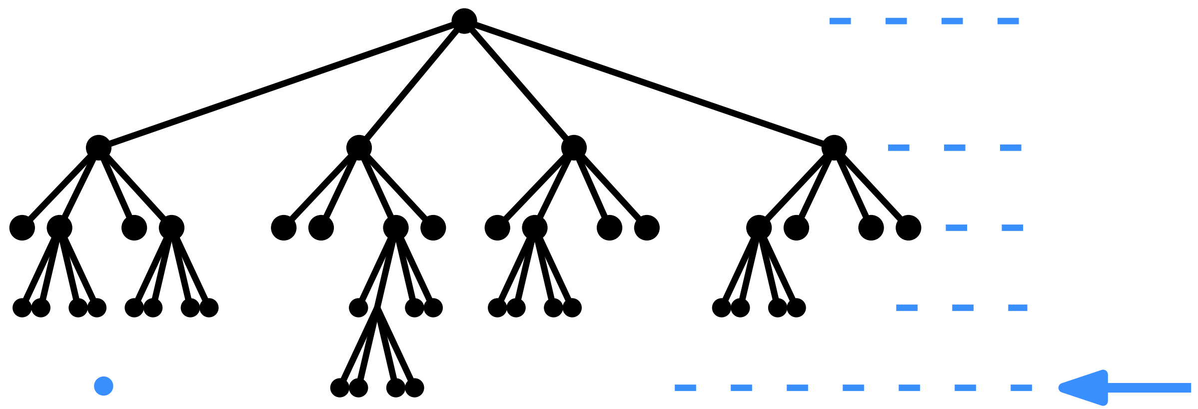
query: binary search on levels

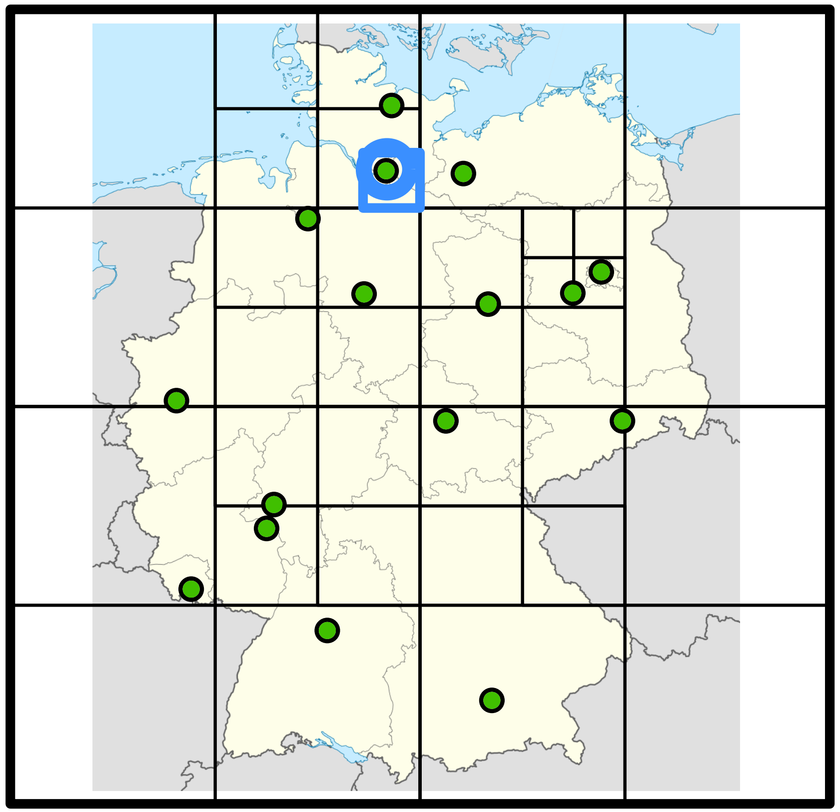
- if inner node: recurse in lower half

Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \to$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v





Faster point location:

preprocessing: build hash table using $id(v)$

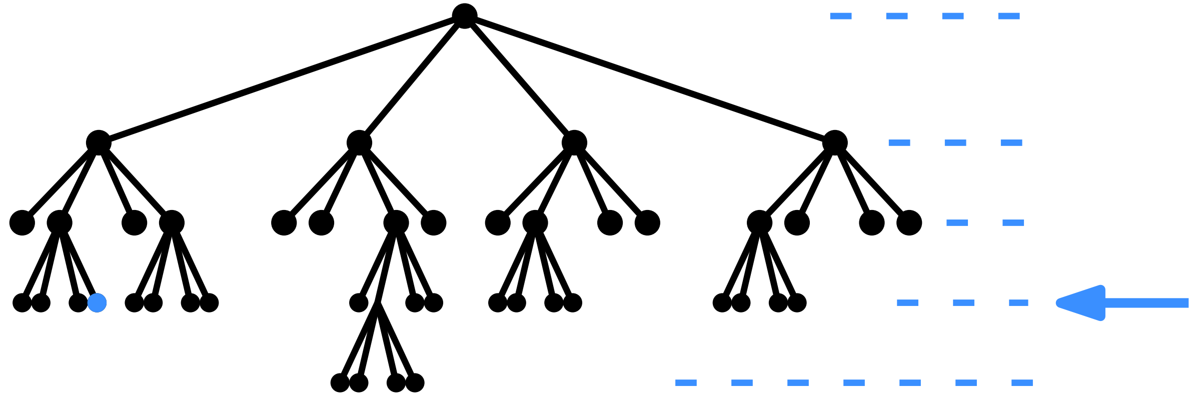
query: binary search on levels

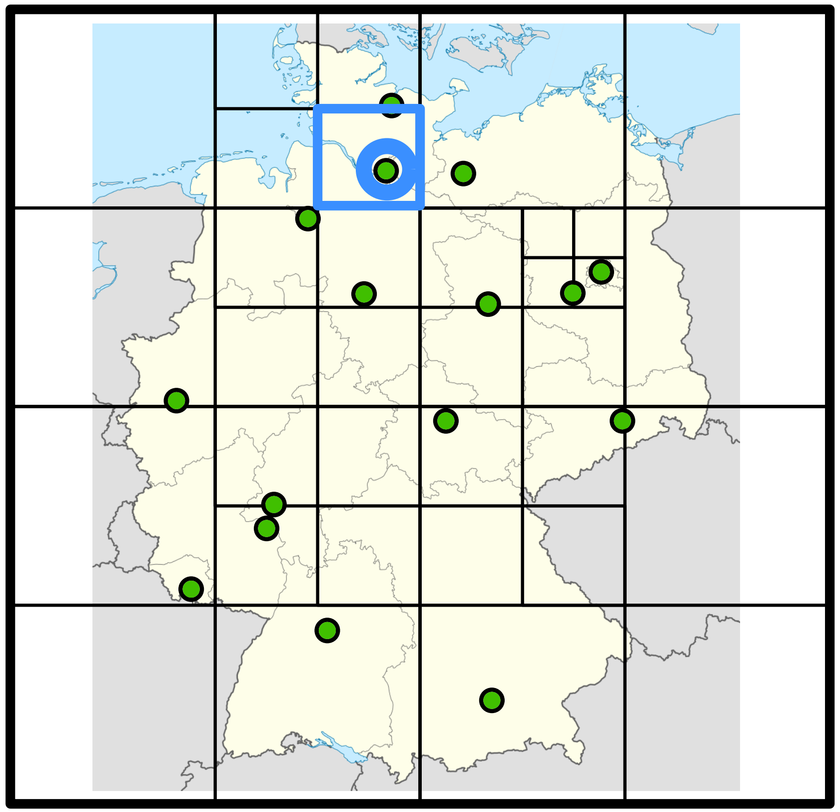
- if inner node: recurse in lower half
- if no node: recurse in upper half

Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \rightarrow$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v





Faster point location:

preprocessing: build hash table using $id(v)$

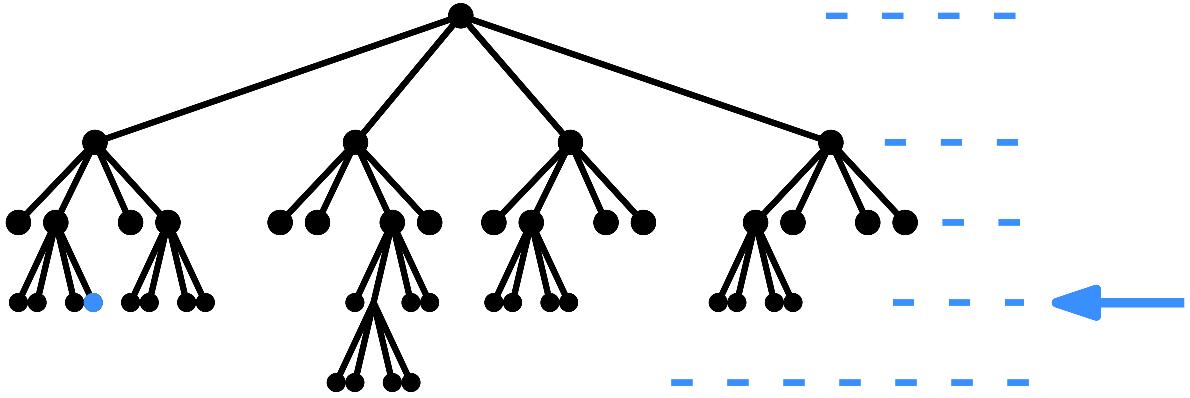
query: binary search on levels

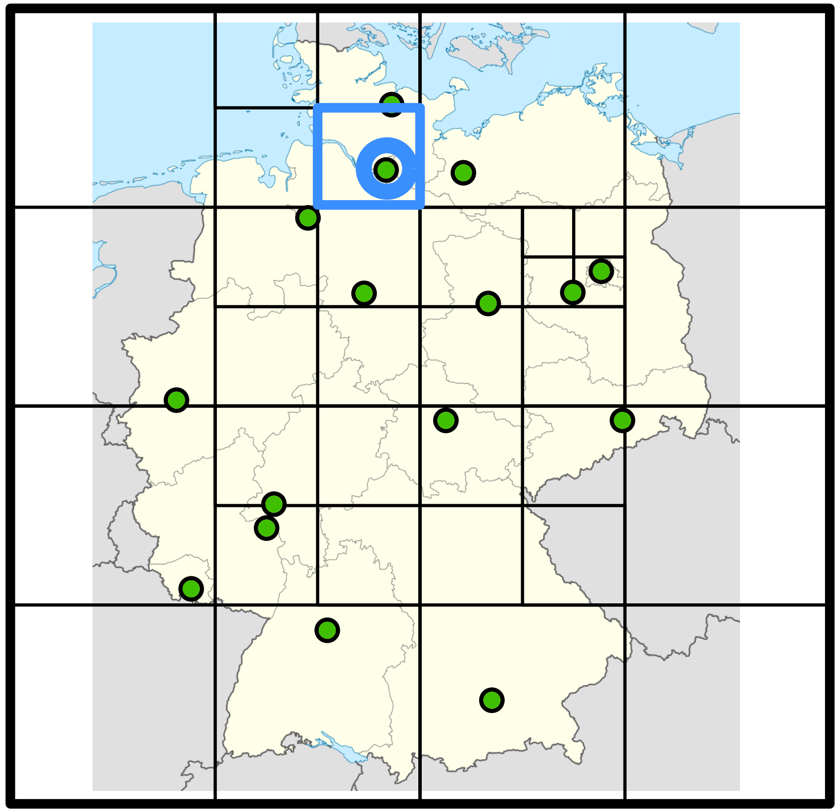
- if inner node: recurse in lower half
- if no node: recurse in upper half
- if leaf: ✓

Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \to$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v





Faster point location:

preprocessing: build hash table using $id(v)$

query: binary search on levels

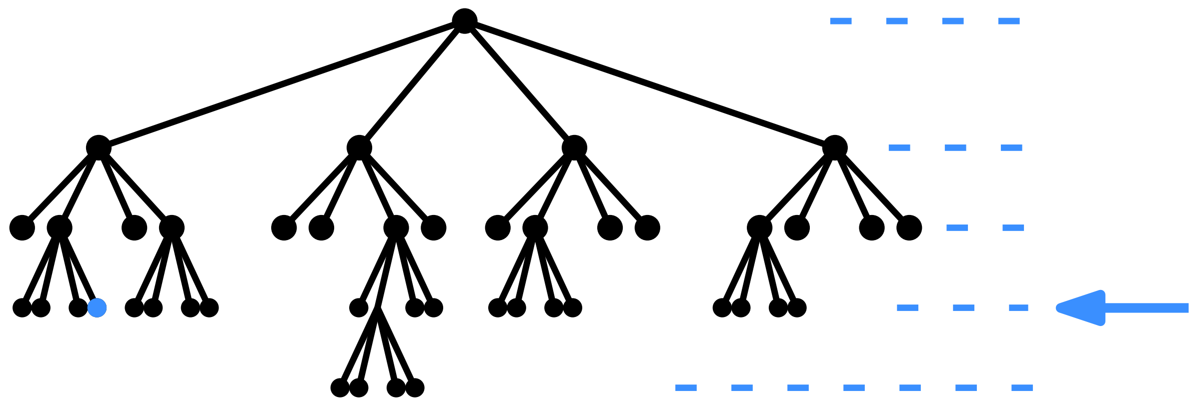
- if inner node: recurse in lower half
- if no node: recurse in upper half
- if leaf: ✓

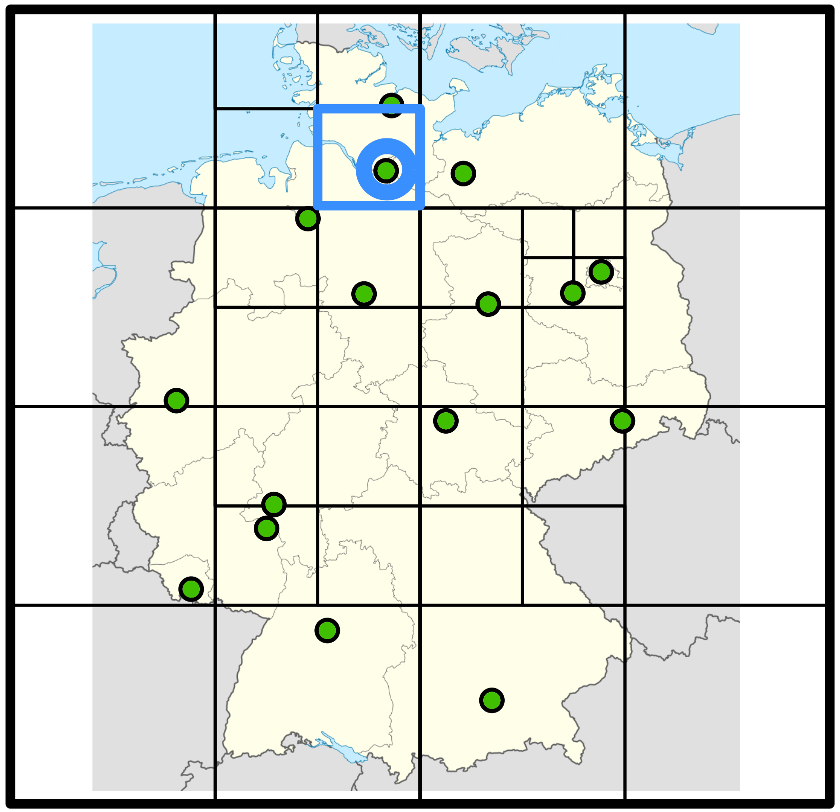
query time: $O(\log d)$

Quadtrees: multi-scale grids

node v at depth i:

- square $S_v \to$ side length $= 2^{-i}$
- in grid $G_{2^{-i}}$
- level $\ell(v) = -i$
- $id(v) = (\ell(v), \lfloor x/2^{\ell(v)} \rfloor, \lfloor y/2^{\ell(v)} \rfloor)$, with (x, y) a point in S_v





Faster point location:

preprocessing: build hash table using $id(v)$

query: binary search on levels

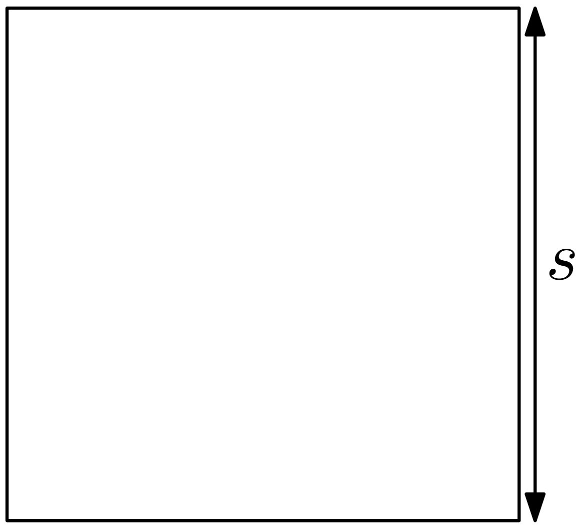
- if inner node: recurse in lower half
- if no node: recurse in upper half
- if leaf: ✓

query time: $O(\log d)$

How large is d? How large is the quadtree?

Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

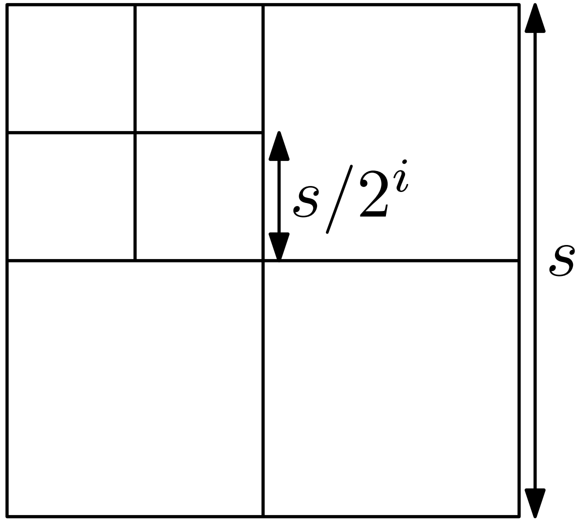


Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Proof:

- consider square σ of depth i with side length $s/2^i$

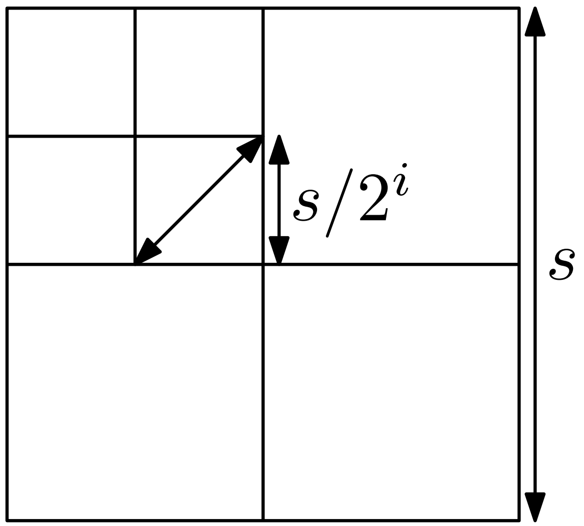


Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Proof:

- consider square σ of depth i with side length $s/2^i$
- maximum distance between two points in σ: $\sqrt{2}s/2^i$



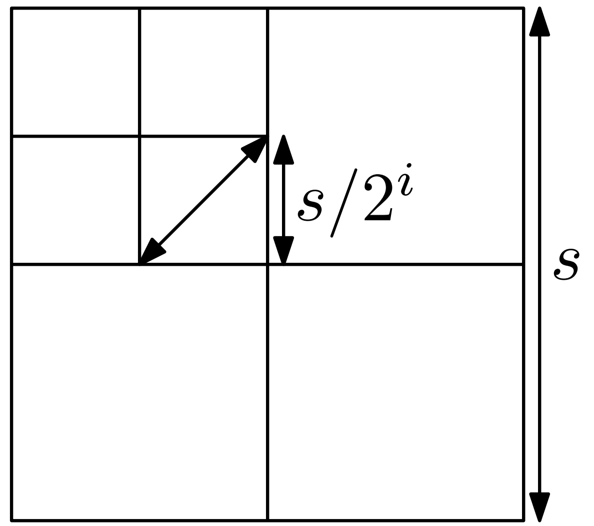
Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Proof:

- consider square σ of depth i with side length $s/2^i$
- maximum distance between two points in σ: $\sqrt{2}s/2^i$

$\Rightarrow$ if depth of cell with ≥ 2 points is i, $\sqrt{2}s/2^i \geq c$



Quadtree: depth and size

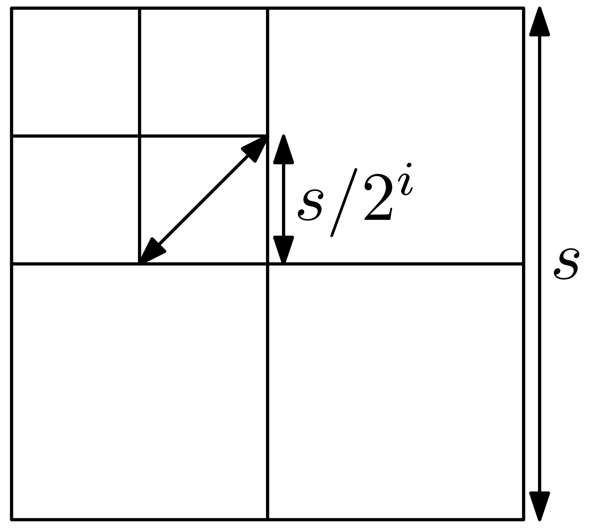
Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Proof:

- consider square σ of depth i with side length $s/2^i$
- maximum distance between two points in σ: $\sqrt{2}s/2^i$

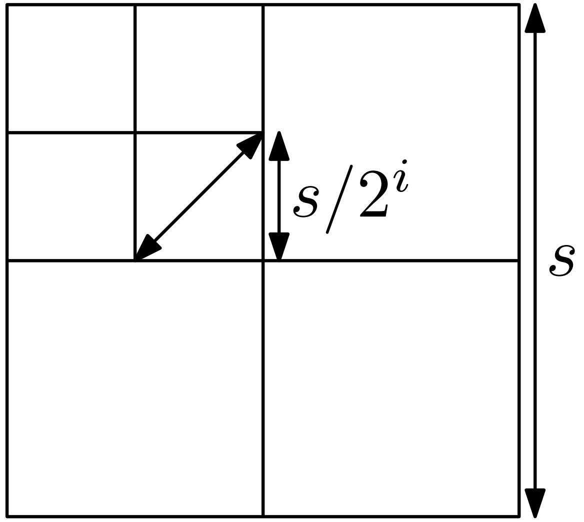
$\Rightarrow$ if depth of cell with ≥ 2 points is i, $\sqrt{2}s/2^i \geq c$

$$\Rightarrow i \leq \log(\sqrt{2}s/c) = \log(s/c) + 1/2$$



Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.



Proof:

- consider square σ of depth i with side length $s/2^i$
- maximum distance between two points in σ: $\sqrt{2}s/2^i$

$\Rightarrow$ if depth of cell with ≥ 2 points is i, $\sqrt{2}s/2^i \geq c$

$$\Rightarrow i \leq \log(\sqrt{2}s/c) = \log(s/c) + 1/2$$

$\Rightarrow$ depth of quadtree $\leq \log(s/c) + 1/2 + 1$, since nodes with ≤ 1 points have no children

Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

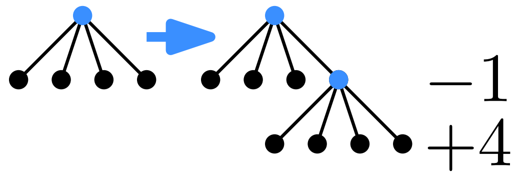
Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Proof:

- Inner nodes have 4 children $\Rightarrow$ #leaves $= 1 + 3\cdot$#inner nodes



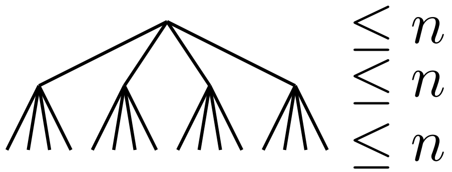
Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Proof:

- Inner nodes have 4 children $\quad \Rightarrow \quad$ #leaves $= 1 + 3\cdot$#inner nodes
- Inner nodes correspond to disjoint squares with ≥ 2 points
  $\Rightarrow \quad \leq n$ squares per layer corresponding to inner nodes



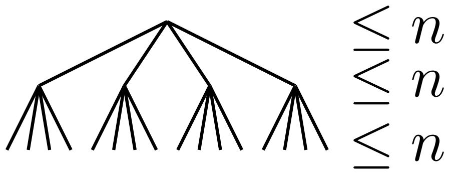
Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Proof:

- Inner nodes have 4 children $\quad \Rightarrow \quad$ #leaves $= 1 + 3\cdot$#inner nodes
- Inner nodes correspond to disjoint squares with ≥ 2 points
  $\Rightarrow \quad \leq n$ squares per layer corresponding to inner nodes



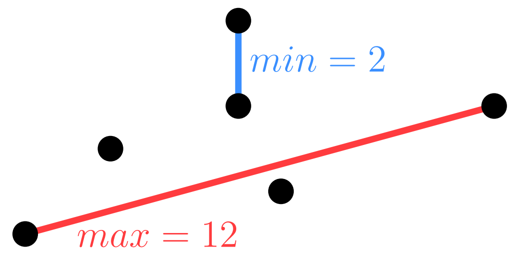
$\Rightarrow$ for depth d overall $O((d+1)n)$ nodes.

Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Definition: The spread of point set P is $\Phi(P) = \frac{\max_{p,q \in P} ||p-q||}{\min_{p,q \in P, p \neq q} ||p-q||}$



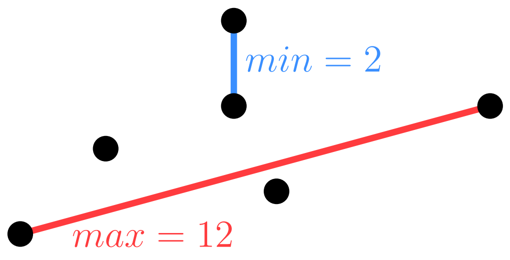
$$\Phi = \frac{max}{min} = \frac{12}{2} = 6$$

Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Definition: The spread of point set P is $\Phi(P) = \frac{\max_{p,q \in P} ||p-q||}{\min_{p,q \in P, p \neq q} ||p-q||}$



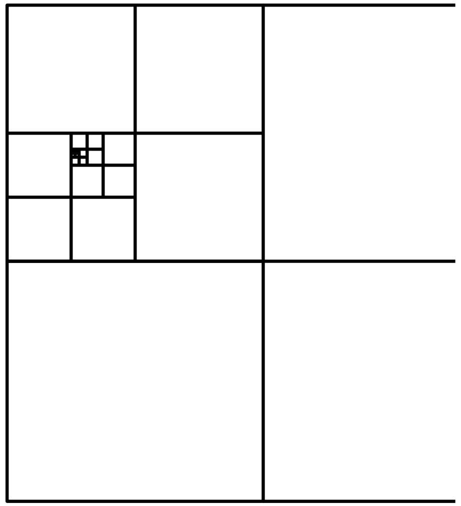
$$\Phi = \frac{max}{min} = \frac{12}{2} = 6$$

Observation: The depth of a quadtree is in $O(\log(\Phi(P)))$ and the size in $O(n \log \Phi(P))$.

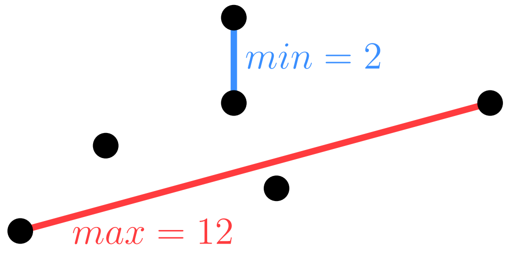
Quadtree: depth and size

Lemma: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.



Definition: The spread of point set P is $\Phi(P) = \frac{\max_{p,q \in P} ||p-q||}{\min_{p,q \in P, p \neq q} ||p-q||}$

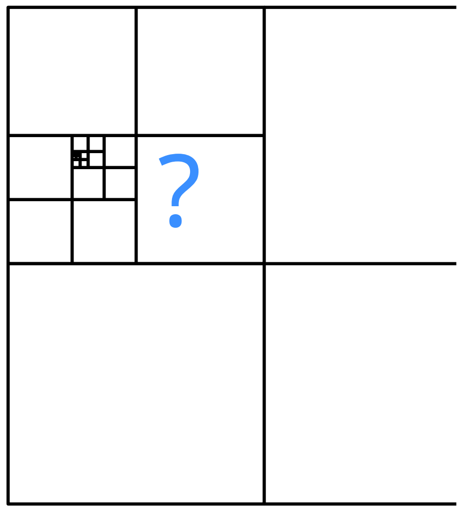


$$\Phi = \frac{max}{min} = \frac{12}{2} = 6$$

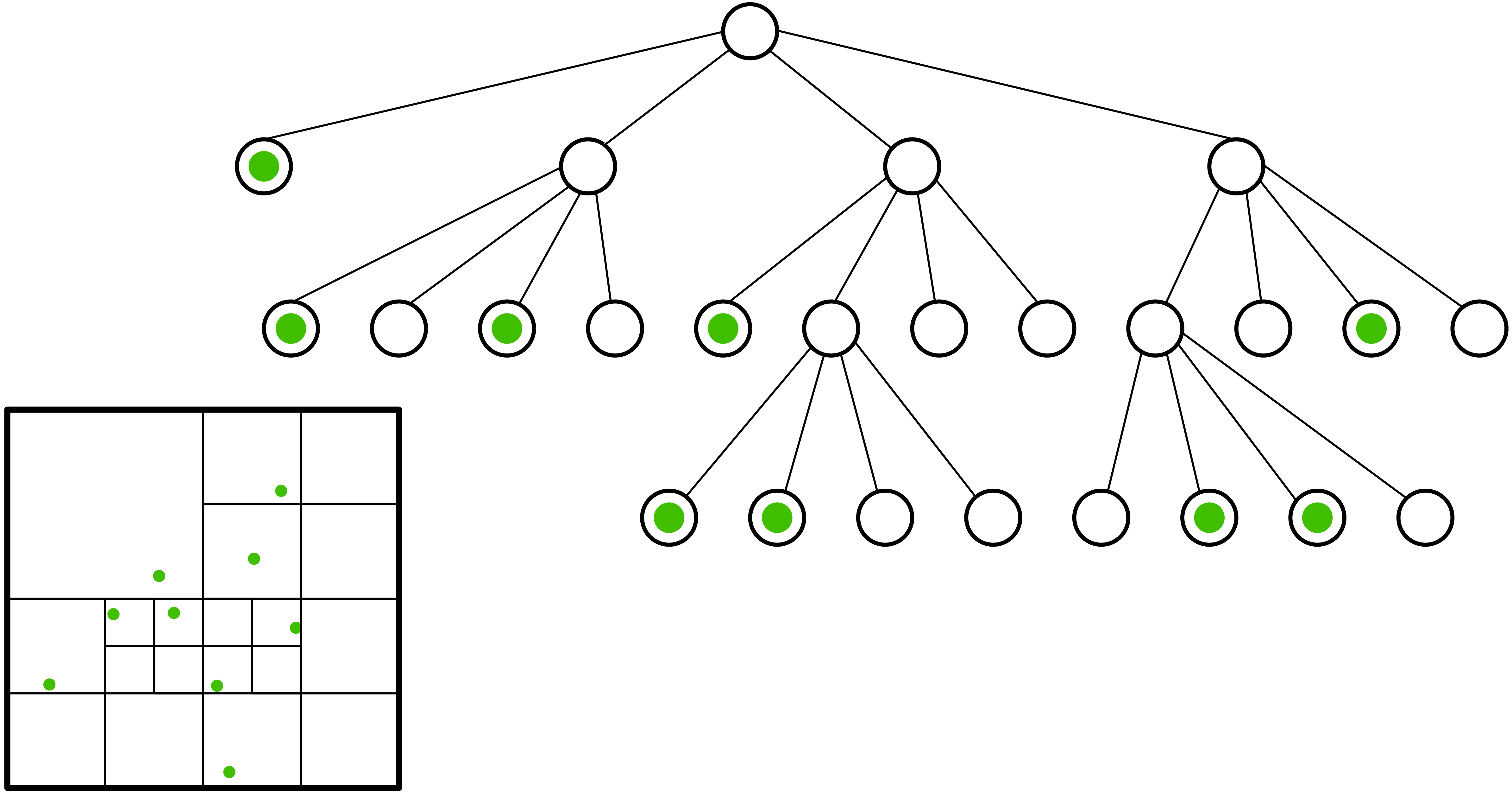
Observation: The depth of a quadtree is in $O(\log(\Phi(P)))$ and the size in $O(n \log \Phi(P))$.

How can we handle the case when $\Phi(P)$ is not bounded by a polynomial in n?   Can we get a linear-size data structure?

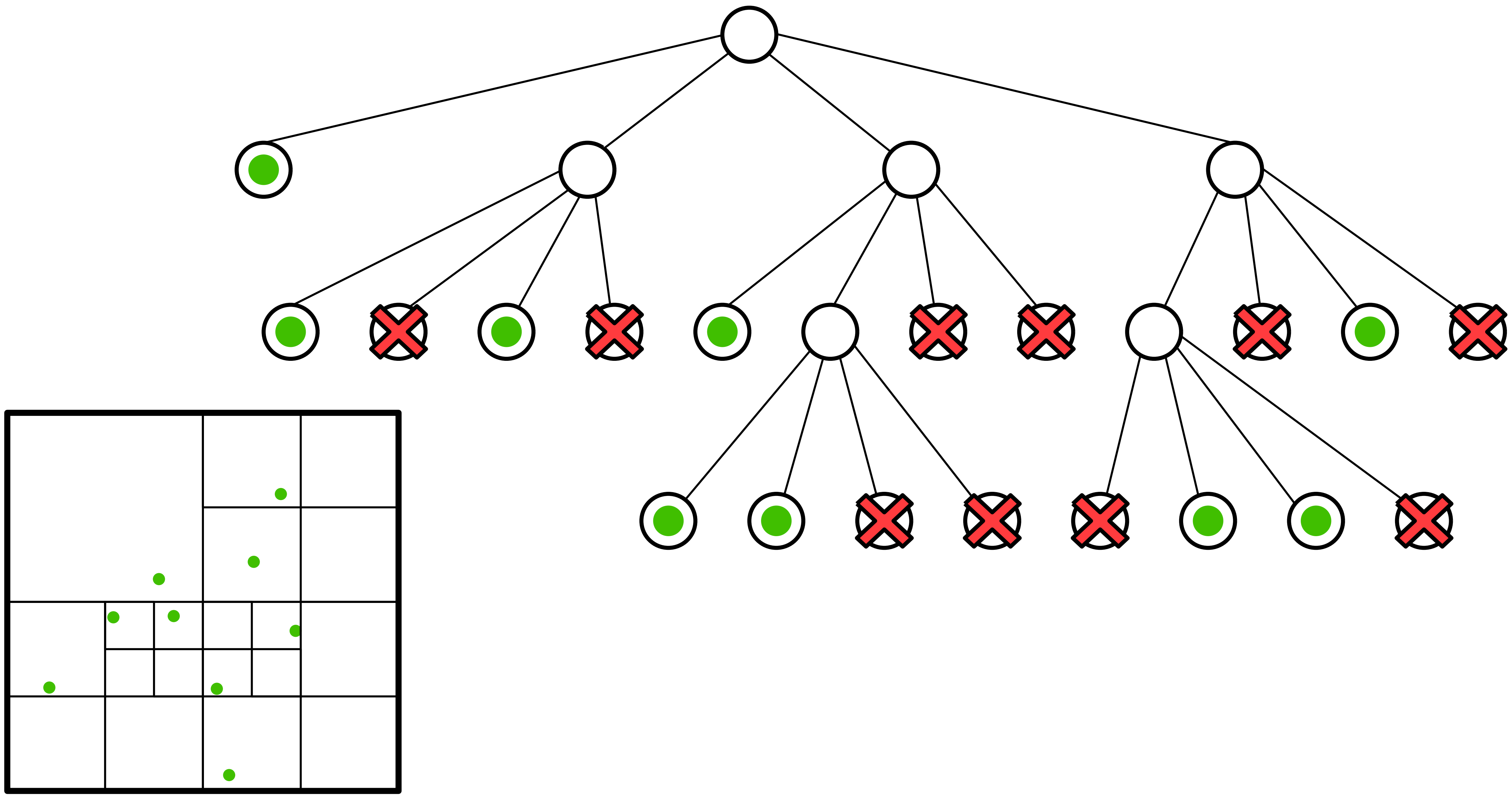
Compressed Quadtrees



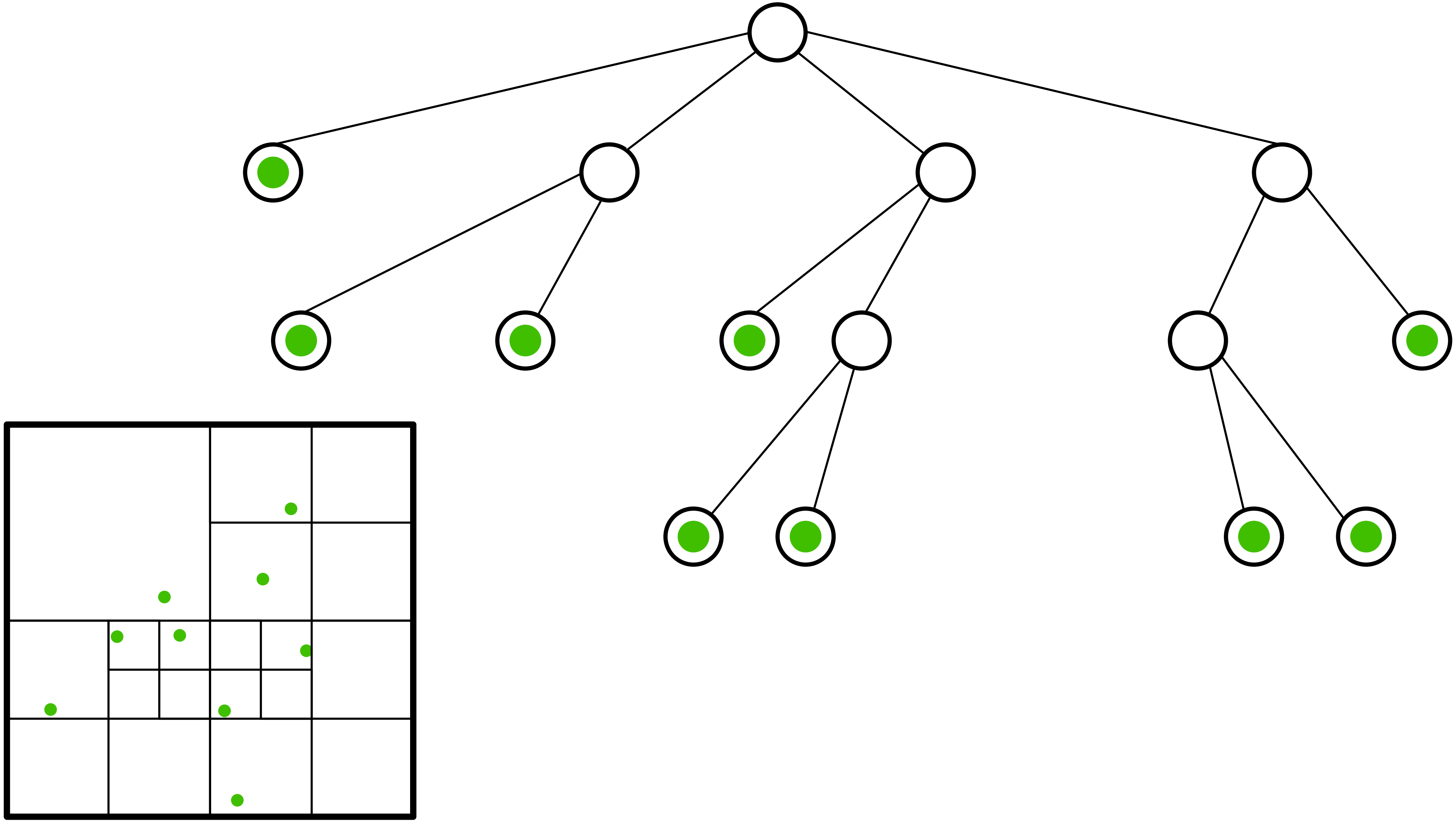
Improving the size, step 0



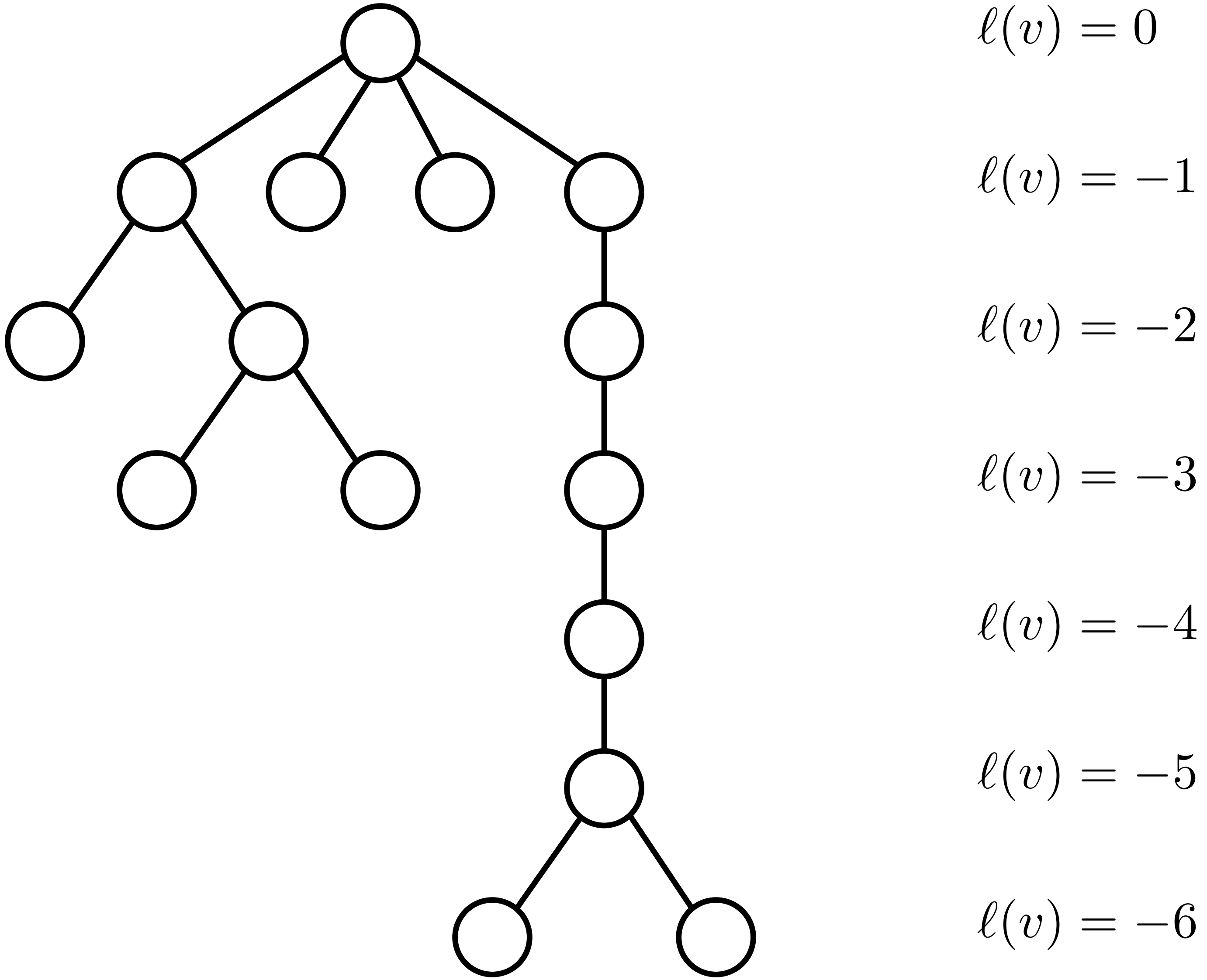
Improving the size, step 0



Improving the size, step 0



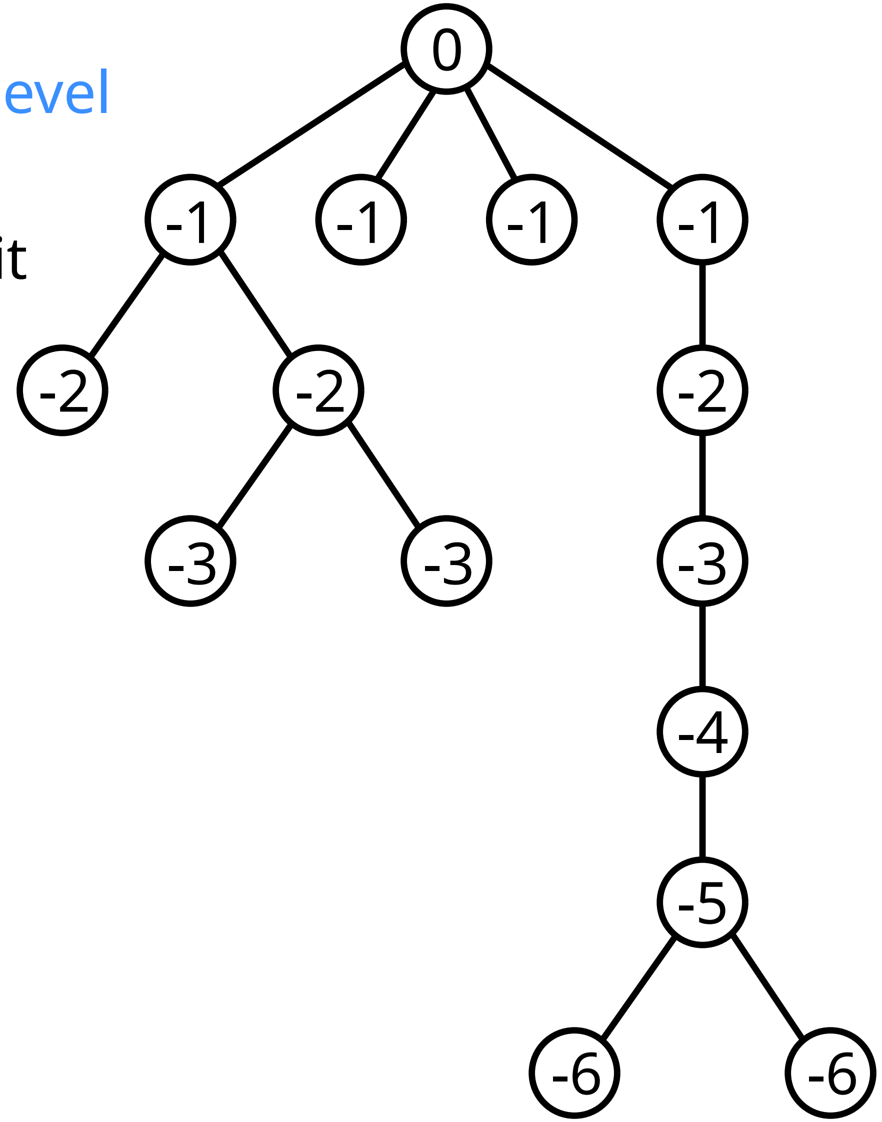
Compressed Quadtrees



Compressed Quadtrees

Each node gets:

- An integer denoting its level in the original quadtree
- A pointer to the square it represents.



$$\ell(v) = 0$$

$$\ell(v) = -1$$

$$\ell(v) = -2$$

$$\ell(v) = -3$$

$$\ell(v) = -4$$

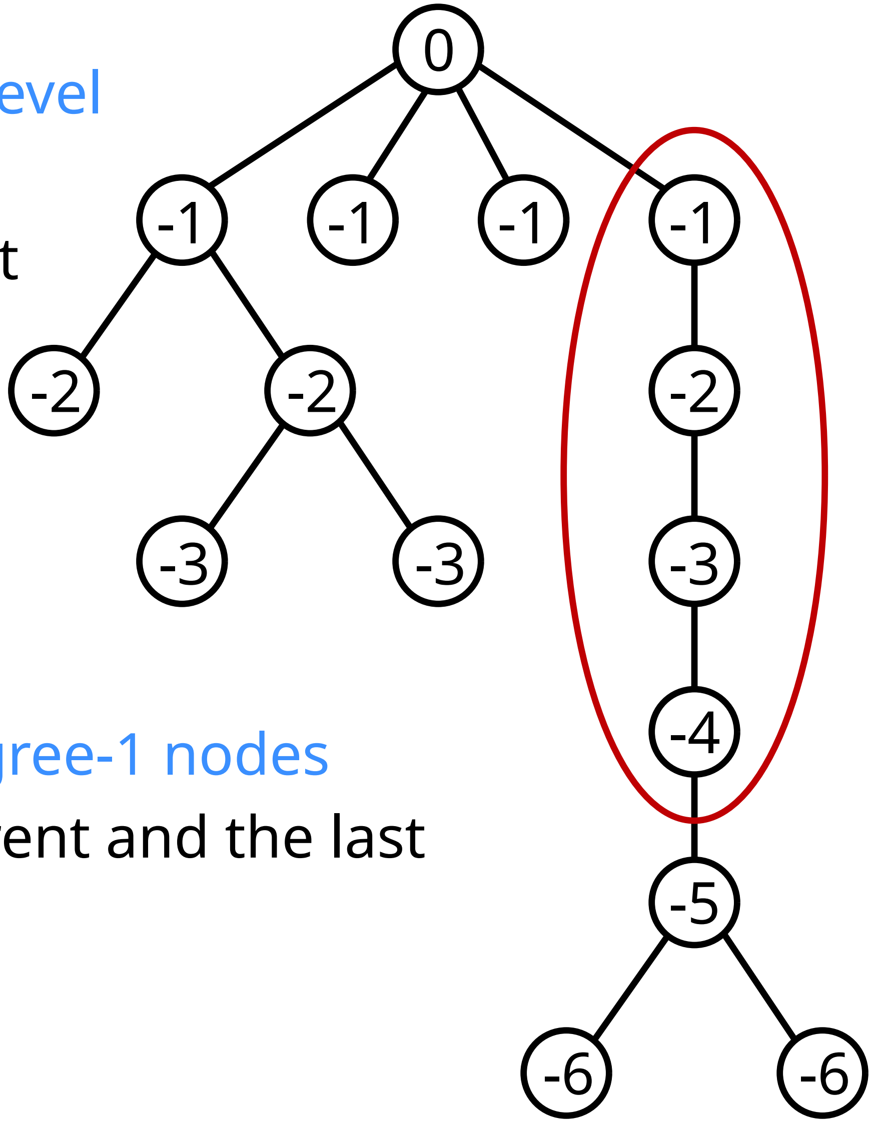
$$\ell(v) = -5$$

$$\ell(v) = -6$$

Compressed Quadtrees

Each node gets:

- An integer denoting its level in the original quadtree
- A pointer to the square it represents.



$\ell(v) = 0$

$\ell(v) = -1$

$\ell(v) = -2$

$\ell(v) = -3$

$\ell(v) = -4$

$\ell(v) = -5$

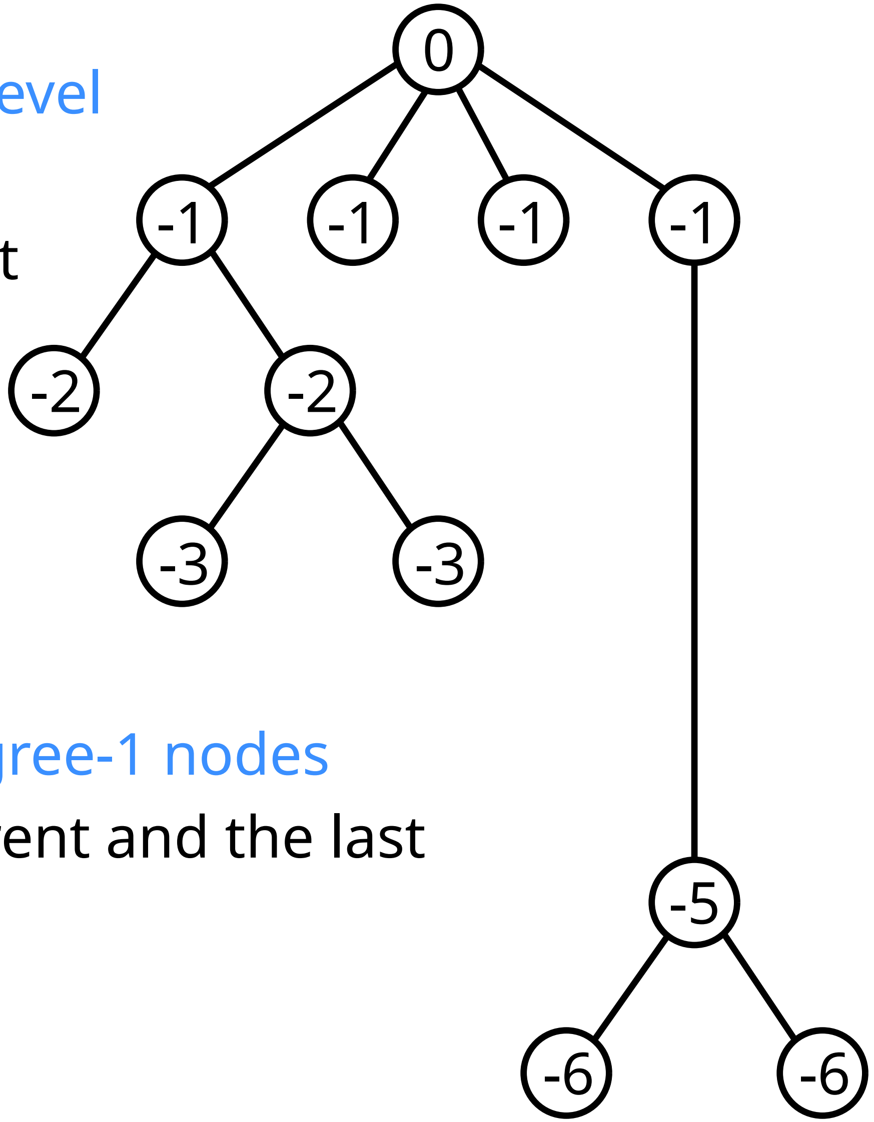
$\ell(v) = -6$

Paths consisting of only degree-1 nodes $\Longrightarrow$ replace by the first parent and the last child on the path.

Compressed Quadtrees

Each node gets:

- An integer denoting its level in the original quadtree
- A pointer to the square it represents.



Paths consisting of only degree-1 nodes $\Longrightarrow$ replace by the first parent and the last child on the path.

$\ell(v) = 0$

$\ell(v) = -1$

$\ell(v) = -2$

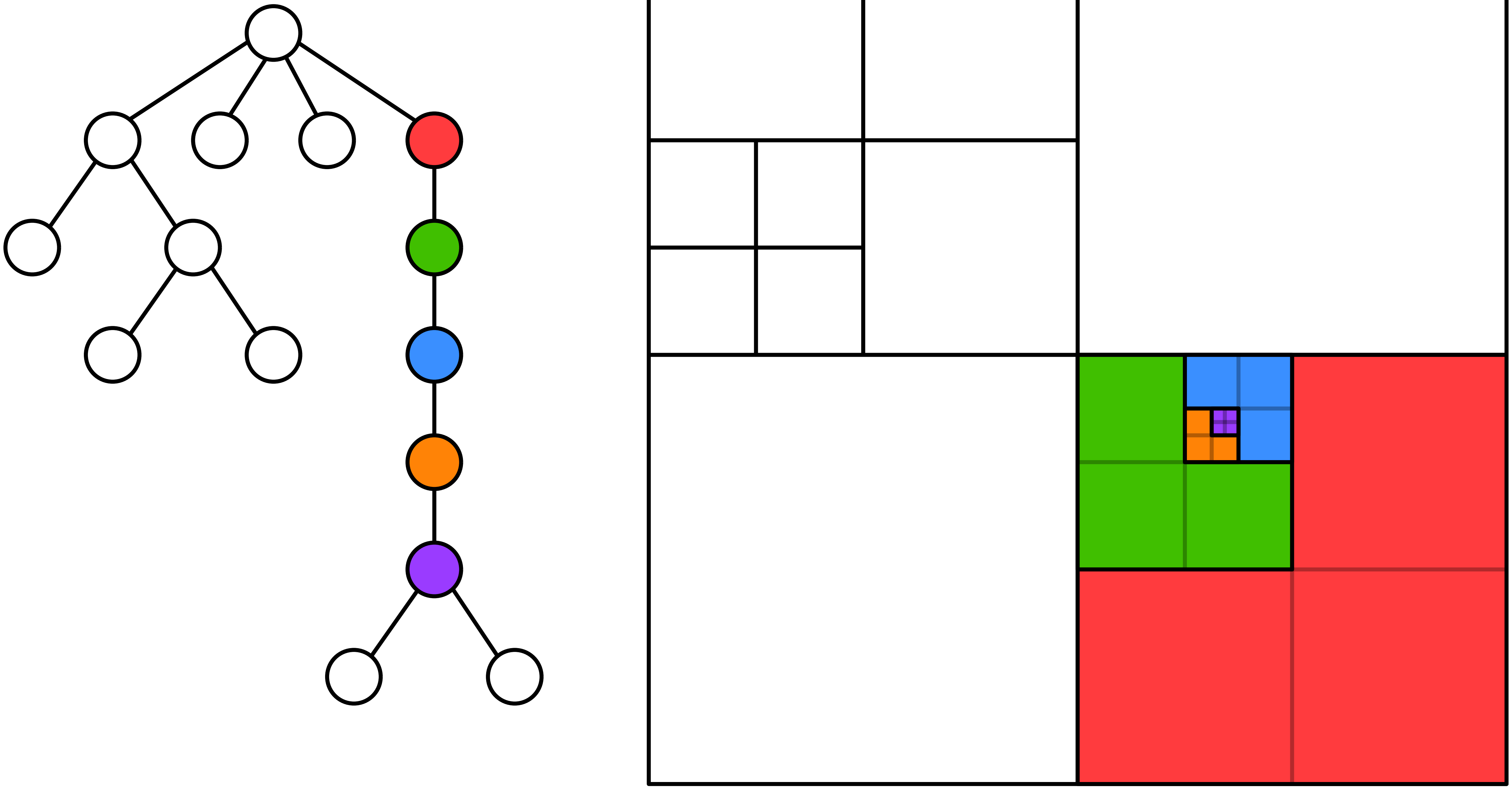
$\ell(v) = -3$

$\ell(v) = -4$

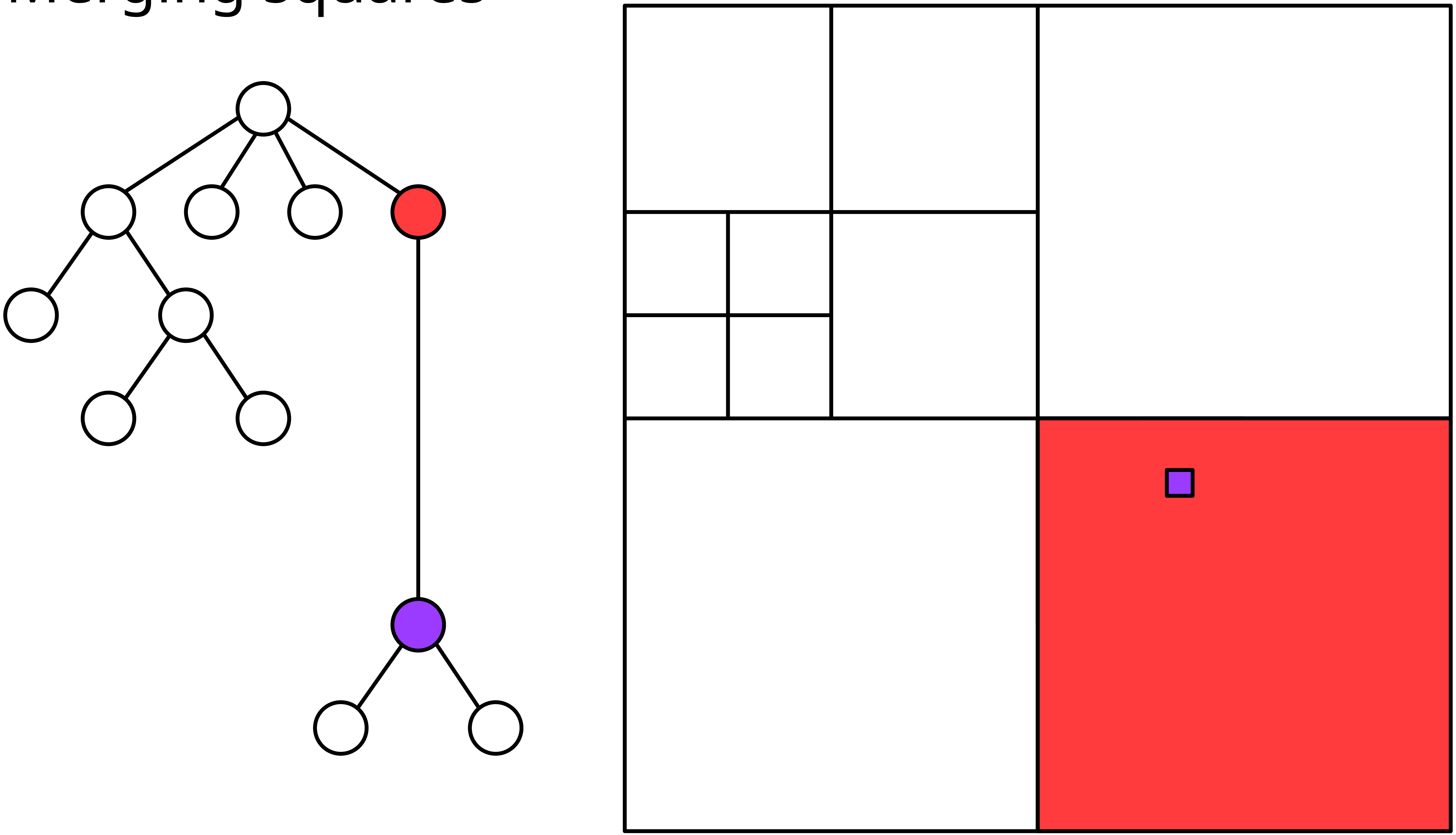
$\ell(v) = -5$

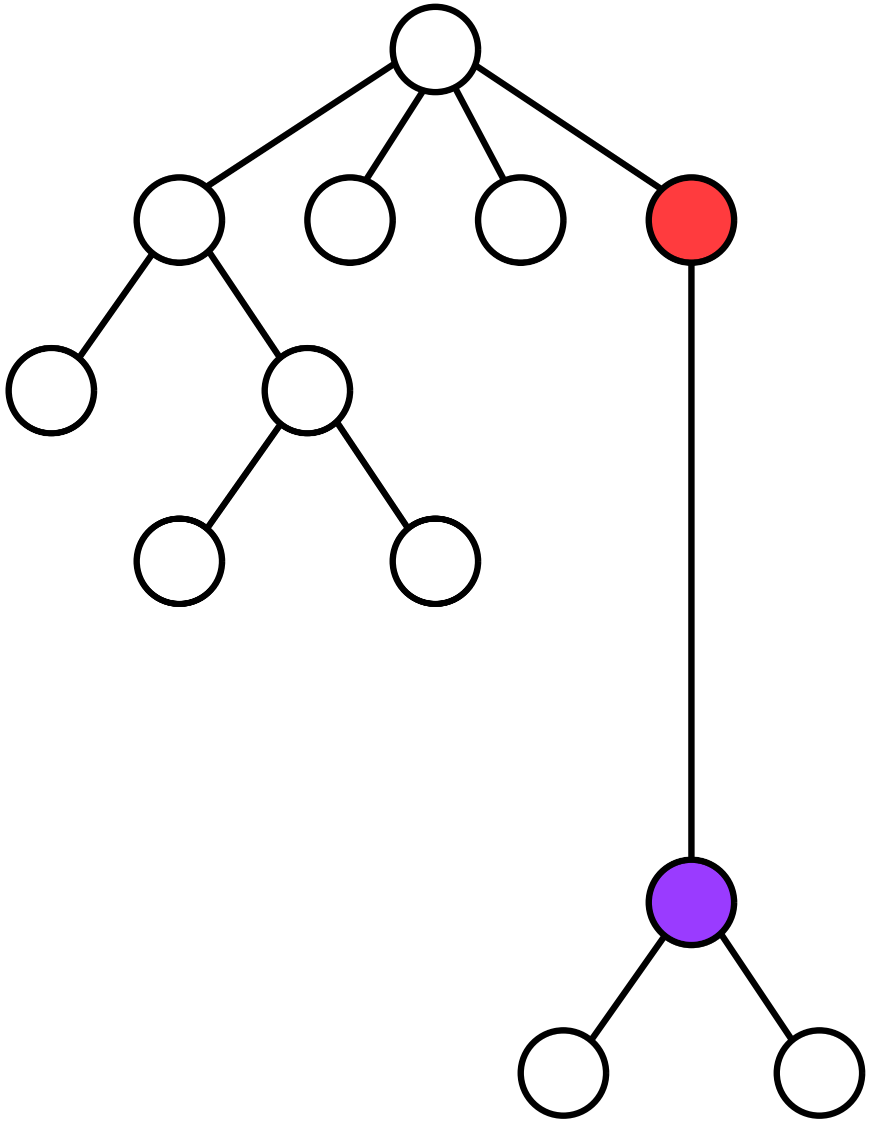
$\ell(v) = -6$

Merging squares



Merging squares

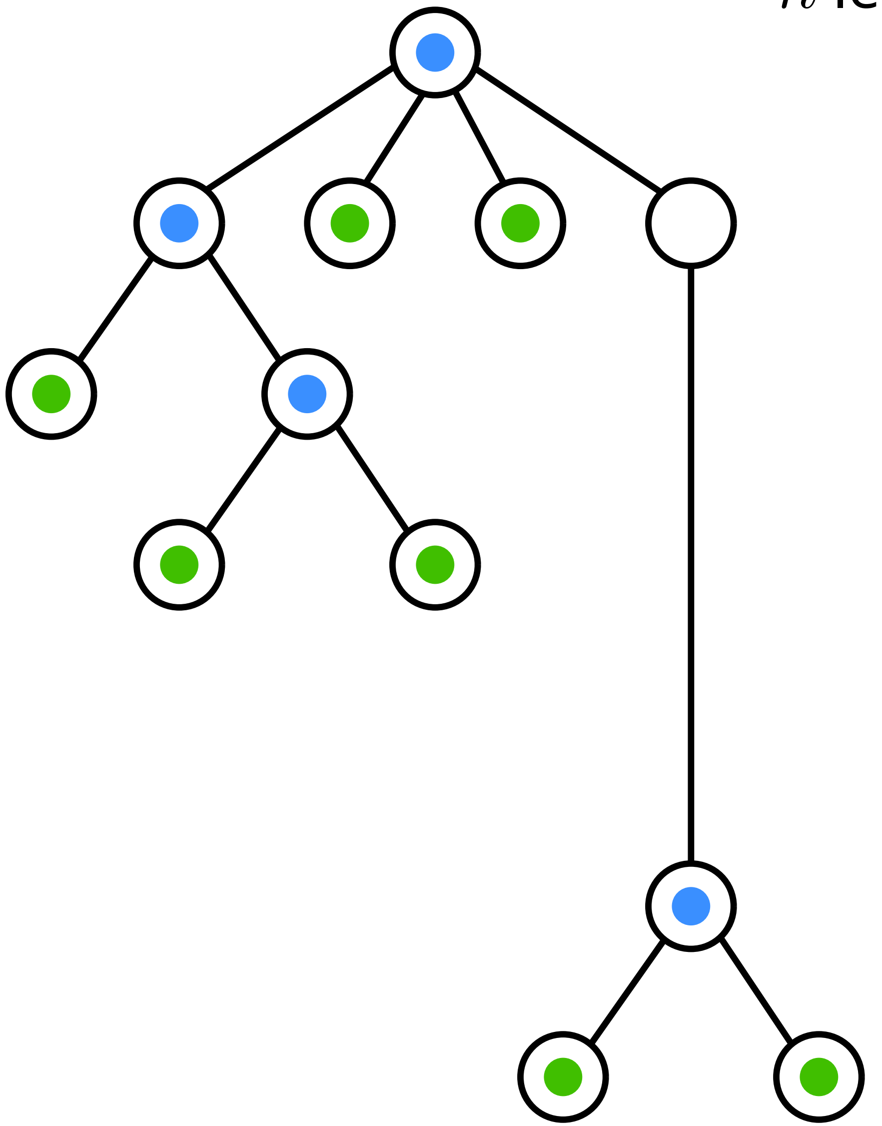




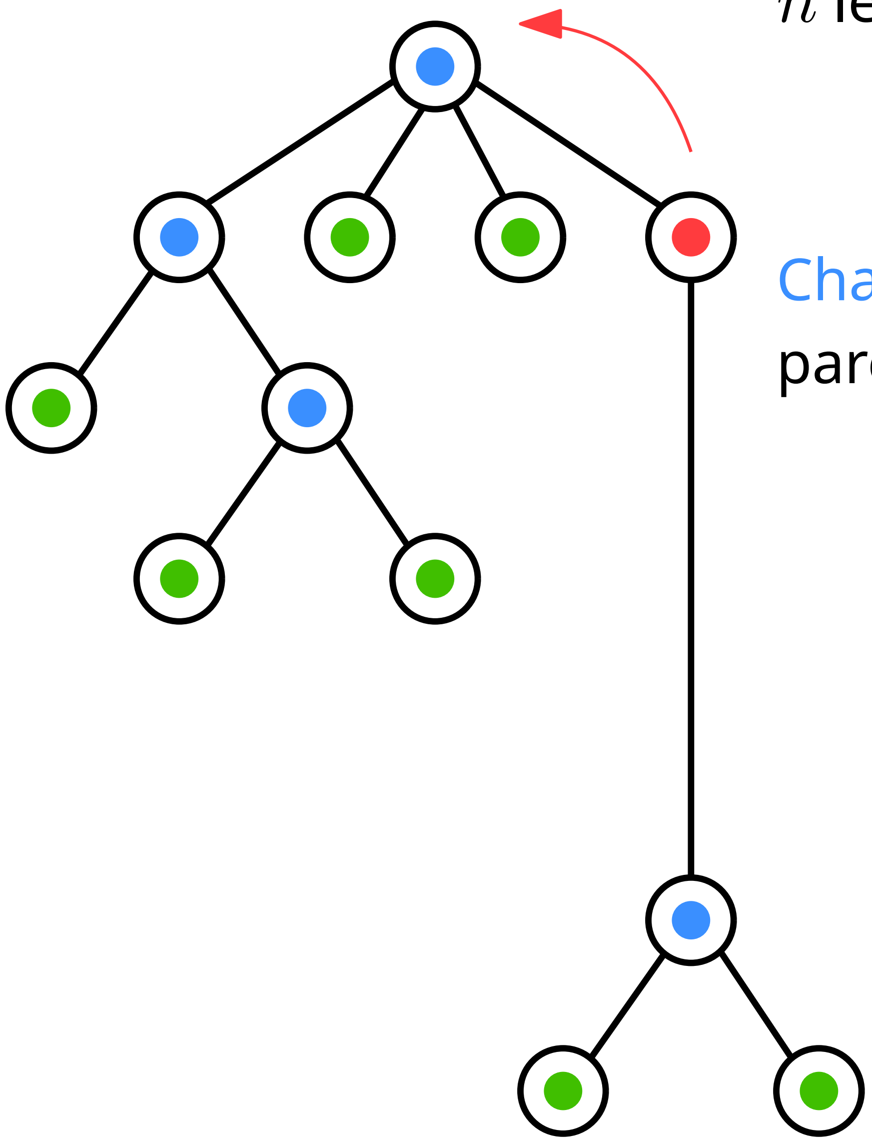
What the size of the compressed quadtree?

Size of compressed quadtree

n leaves $\Rightarrow$ at most $n-1$ internal nodes with degree ≥ 2



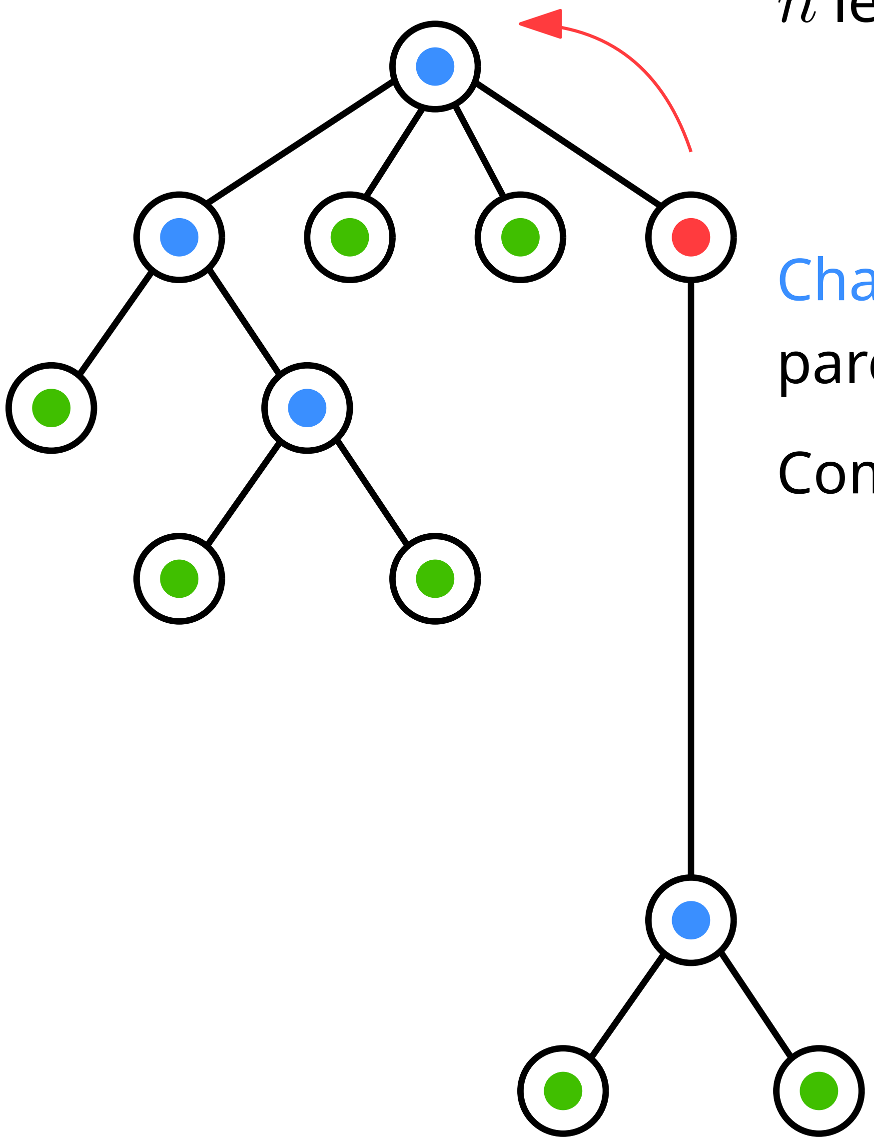
Size of compressed quadtree



n leaves $\Rightarrow$ at most $n-1$ internal nodes with degree ≥ 2

Charging argument: Charge any node with a single child to its parent, which has 2 or more children because of compression

Size of compressed quadtree



n leaves $\Rightarrow$ at most $n - 1$ internal nodes with degree ≥ 2

Charging argument: Charge any node with a single child to its parent, which has 2 or more children because of compression

Compressed quadtrees have linear size!

Efficient construction

Simple recursive construction on compressed quadtrees has unbounded time complexity when the spread of the point set is unbounded.

We can do better with a divide and conquer approach!

Efficient construction

Simple recursive construction on compressed quadtrees has unbounded time complexity when the spread of the point set is unbounded.

We can do better with a divide and conquer approach!

Idea: Find a square (in a grid $G_{2^{-i}}$) that contains a constant fraction of the points.

Efficient construction

Simple recursive construction on compressed quadtrees has unbounded time complexity when the spread of the point set is unbounded.

We can do better with a divide and conquer approach!

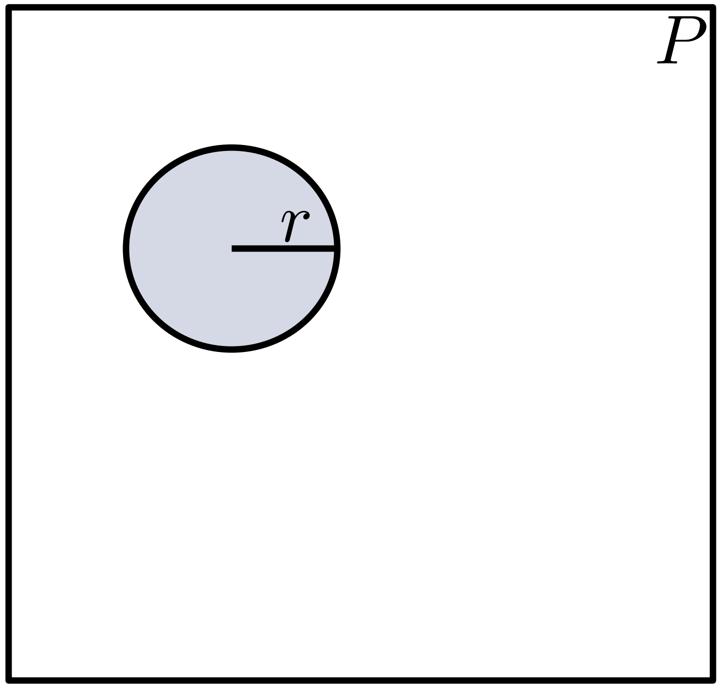
Idea: Find a square (in a grid $G_{2^{-i}}$) that contains a constant fraction of the points.

Theorem: In linear time we can compute a disk D containing $n/10$ points with radius $r_D \leq 2r_{OPT}$, where r_{OPT} is the radius of the smallest disk containing $n/10$ points.

Question: Which algorithm(s) do you know to compute this disk?

Efficient construction

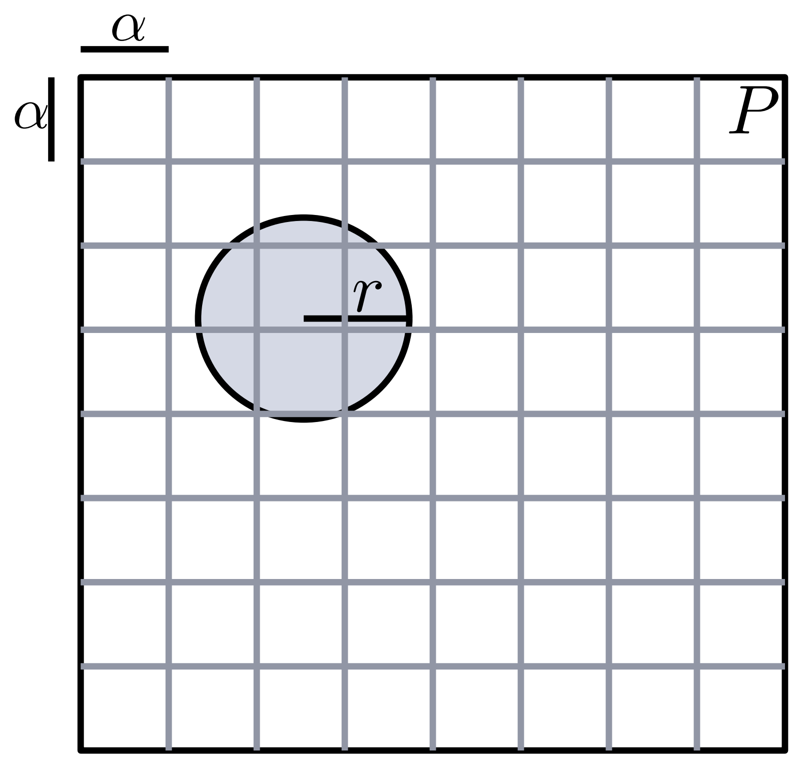
Theorem: In linear time we can compute a disk D containing $n/10$ points with radius $r_D \leq 2r_{OPT}$, where r_{OPT} is the radius of the smallest disk containing $n/10$ points.



Efficient construction

Theorem: In linear time we can compute a disk D containing $n/10$ points with radius $r_D \leq 2r_{OPT}$, where r_{OPT} is the radius of the smallest disk containing $n/10$ points.

$$\alpha = 2^{\lfloor \log_2(r) \rfloor}$$



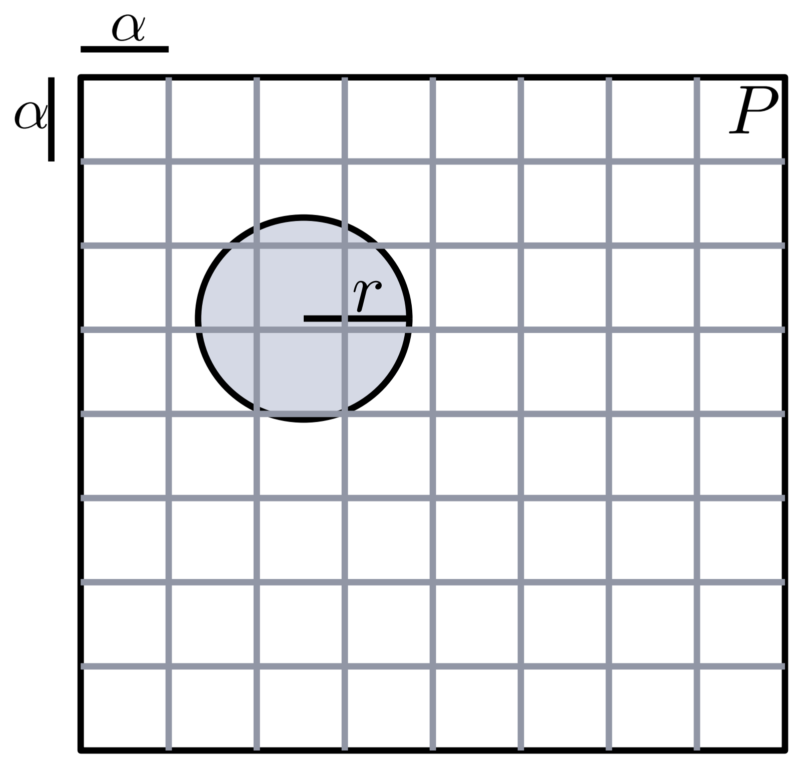
Efficient construction

Theorem: In linear time we can compute a disk D containing $n/10$ points with radius $r_D \leq 2r_{OPT}$, where r_{OPT} is the radius of the smallest disk containing $n/10$ points.

$$\alpha = 2^{\lfloor \log_2(r) \rfloor}$$

$$r \geq \alpha \geq r/2$$

$\implies$ D is covered by at most 25 grid cells



Efficient construction

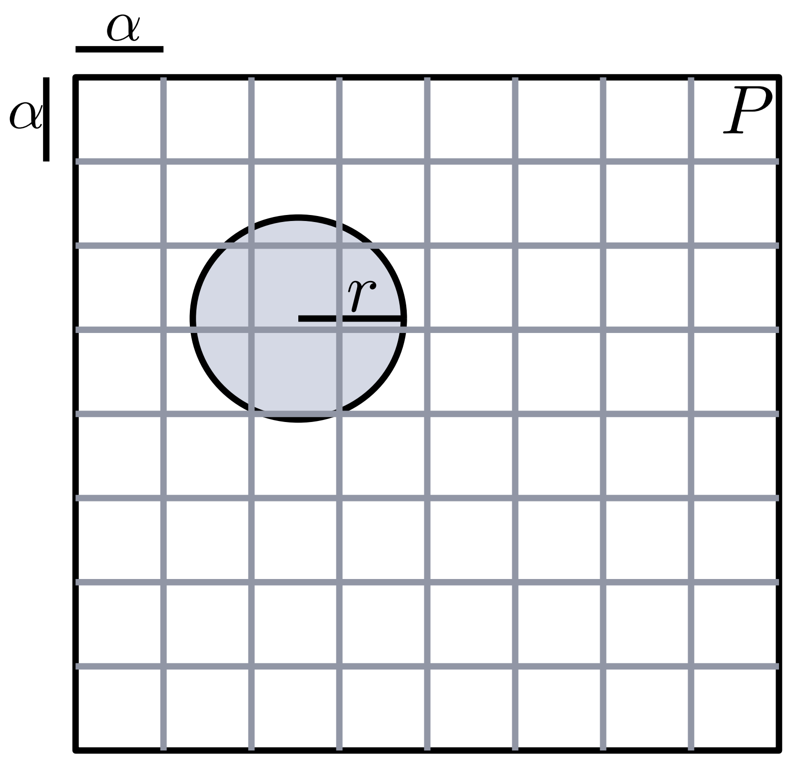
Theorem: In linear time we can compute a disk D containing $n/10$ points with radius $r_D \leq 2r_{OPT}$, where r_{OPT} is the radius of the smallest disk containing $n/10$ points.

$\alpha = 2^{\lfloor \log_2(r) \rfloor}$

$r \geq \alpha \geq r/2$

$\implies$ D is covered by at most 25 grid cells

$\implies$ $\exists$ a cell c containing at least $\frac{n/10}{25}$ points



Efficient construction

Theorem: In linear time we can compute a disk D containing $n/10$ points with radius $r_D \leq 2r_{OPT}$, where r_{OPT} is the radius of the smallest disk containing $n/10$ points.

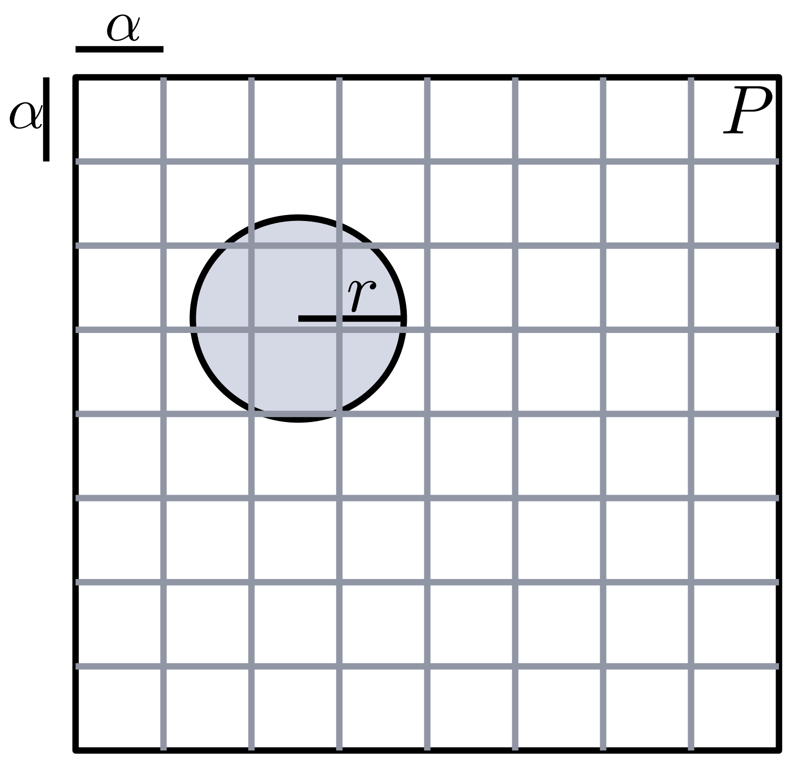
$\alpha = 2^{\lfloor \log_2(r) \rfloor}$

$r \geq \alpha \geq r/2$

$\implies$ D is covered by at most 25 grid cells

$\implies$ $\exists$ a cell c containing at least $\frac{n/10}{25}$ points

Lemma: No cell contains more than $5 \cdot n/10 = n/2$ points



Efficient construction

Theorem: In linear time we can compute a disk D containing $n/10$ points with radius $r_D \leq 2r_{OPT}$, where r_{OPT} is the radius of the smallest disk containing $n/10$ points.

$\alpha = 2^{\lfloor \log_2(r) \rfloor}$

$r \geq \alpha \geq r/2$

$\implies$ D is covered by at most 25 grid cells

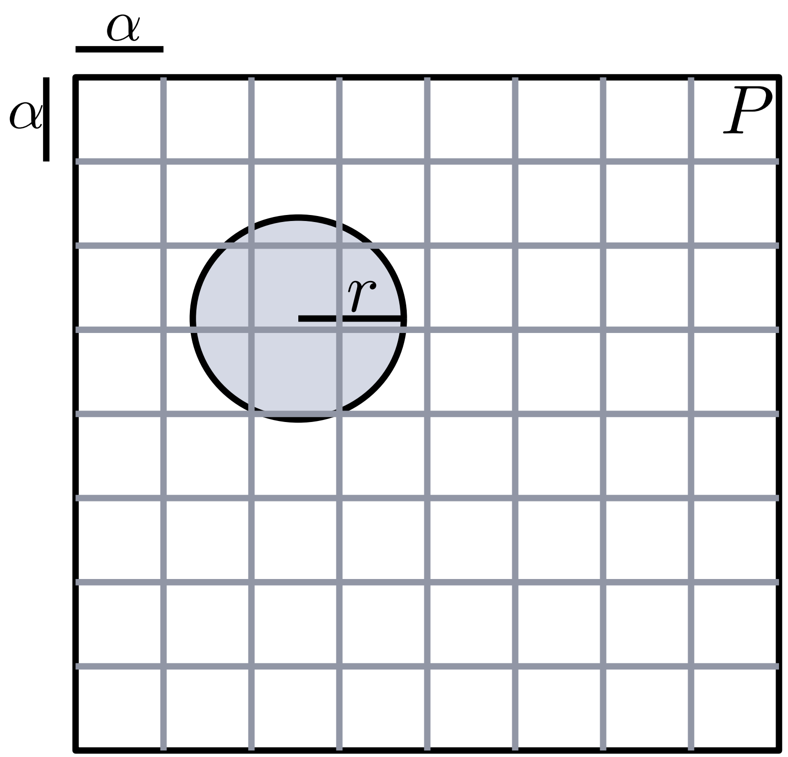
$\implies$ $\exists$ a cell c containing at least $\frac{n/10}{25}$ points

Lemma: No cell contains more than $5 \cdot n/10 = n/2$ points

Let $\square$ denote the cell containing the largest number of points.

$$P_{in} = P \cap \square \text{ and } P_{out} = P \setminus P_{in}$$

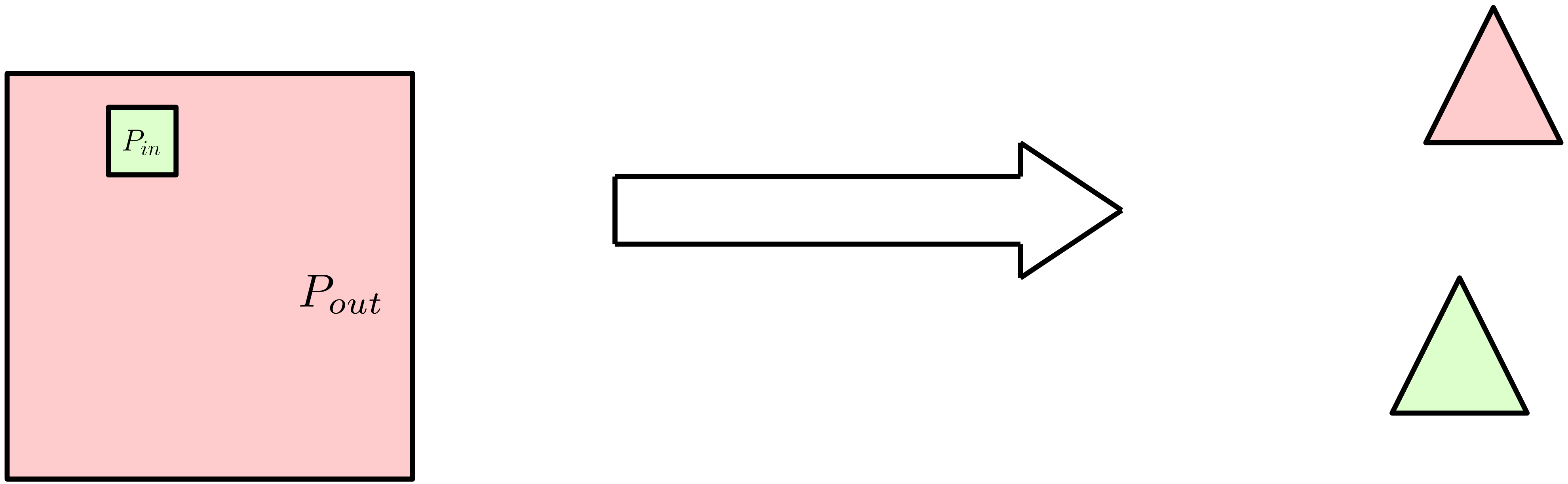
Note that $|P_{in}| \geq n/250$ and $|P_{out}| \geq n/2$



Efficient construction

$P_{in} = P \cap \square$ and $P_{out} = P \setminus P_{in}$

Recursively construct quadtrees for P_{in} and P_{out}

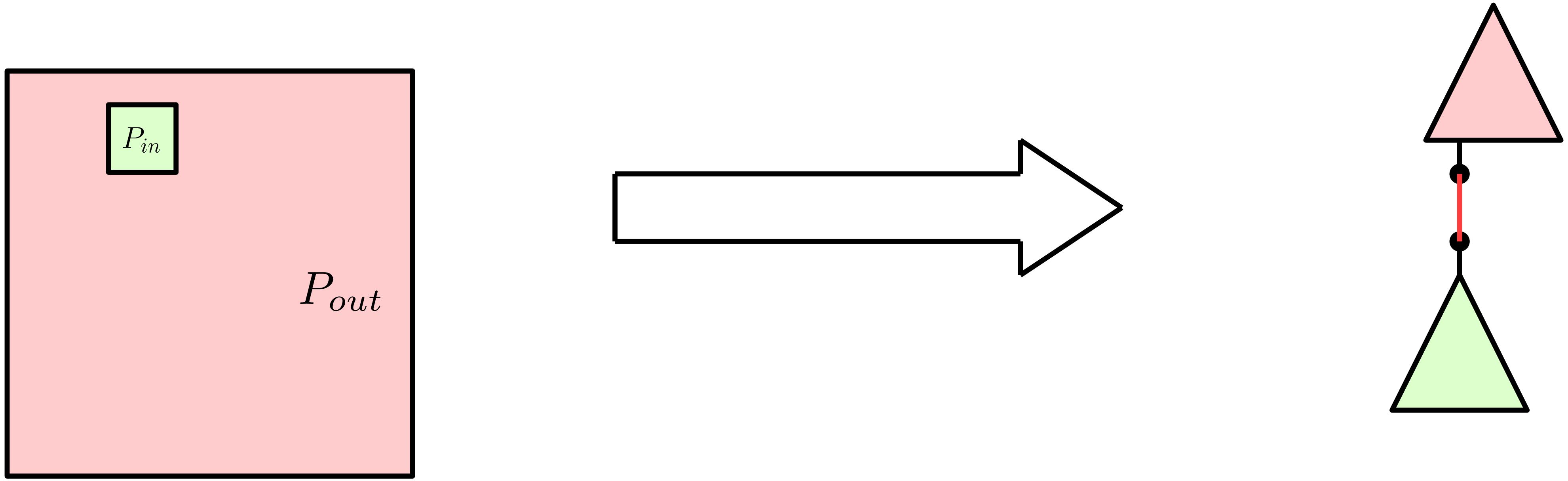


Efficient construction

$$P_{in} = P \cap \square \text{ and } P_{out} = P \setminus P_{in}$$

Recursively construct quadtrees for P_{in} and P_{out}

Create a node representing $\square$ in both quadtrees.

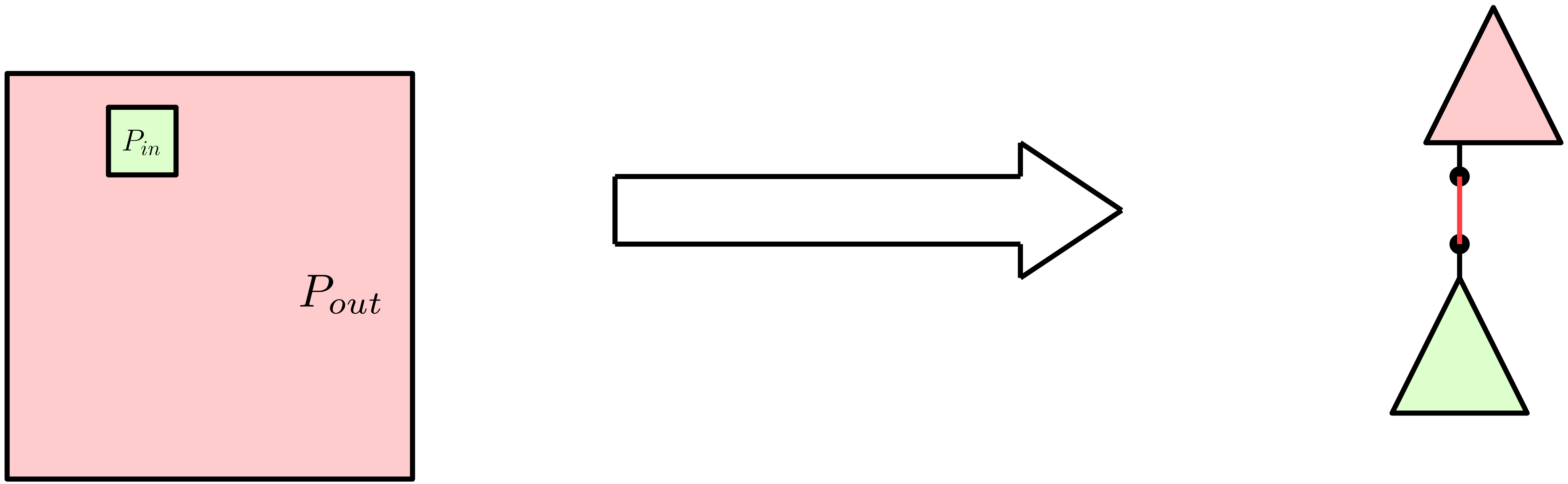


Efficient construction

$P_{in} = P \cap \square$ and $P_{out} = P \setminus P_{in}$

Recursively construct quadtrees for P_{in} and P_{out}

Create a node representing $\square$ in both quadtrees.



Construction time: $T(n) = O(n) + T(|P_{in}|) + T(|P_{out}|) = O(n \log n)$

Quiz

What is the maximum depth that a quadtree on n points can have?

A $\Theta(\log n)$

B $\Theta(\sqrt{n})$

C $\Theta(n)$

Quiz

What is the maximum depth that a quadtree on n points can have?

A $\Theta(\log n)$

B $\Theta(\sqrt{n})$

C $\Theta(n)$

Question: How does such a quadtree look like?

Point-location on compressed quadtrees

Given a compressed quadtree T of size n, find lowest node in the tree containing point q.

Point-location on compressed quadtrees

Given a compressed quadtree T of size n, find lowest node in the tree containing point q.

May take $\Omega(n)$ time!

Point-location on compressed quadtrees

Given a compressed quadtree T of size n, find lowest node in the tree containing point q.

May take $\Omega(n)$ time!

Alternative: preprocess T into a balanced tree T' with cross-pointers to T.

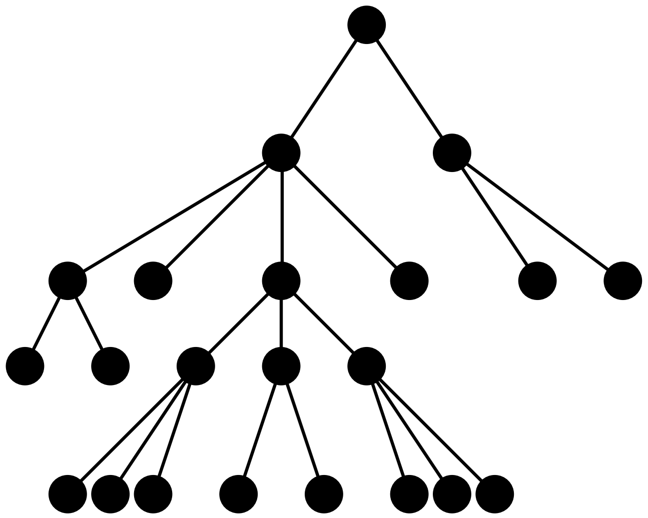
Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

Claim: Any tree T always contains a separator.

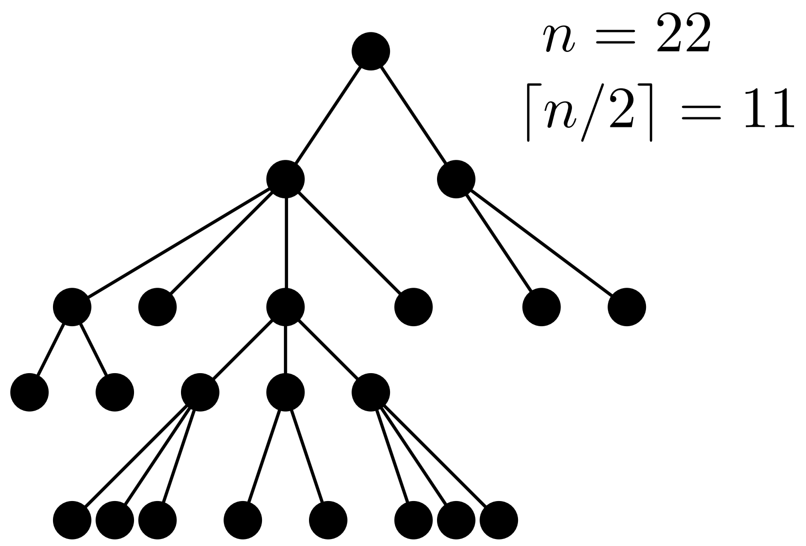


Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

Claim: Any tree T always contains a separator.

Walk through the tree starting at root, going into the subtree that contains $\geq \lceil n/2 \rceil$ nodes.

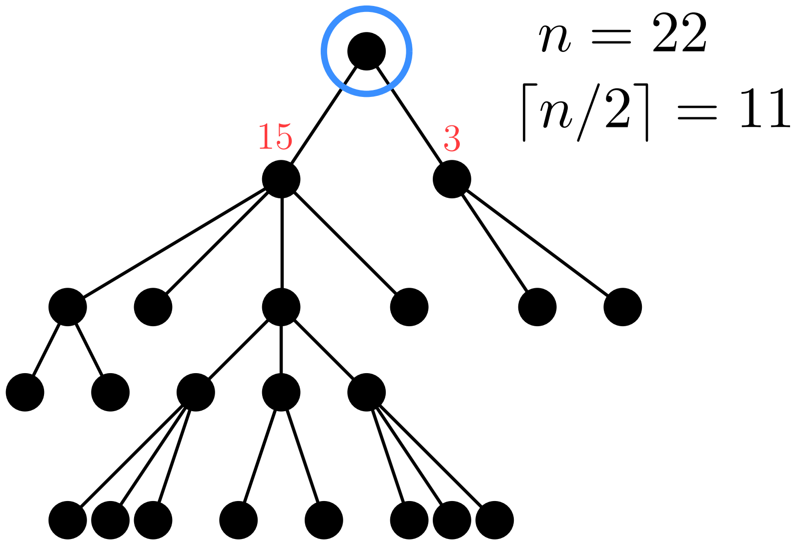


Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

Claim: Any tree T always contains a separator.

Walk through the tree starting at root, going into the subtree that contains $\geq \lceil n/2 \rceil$ nodes.

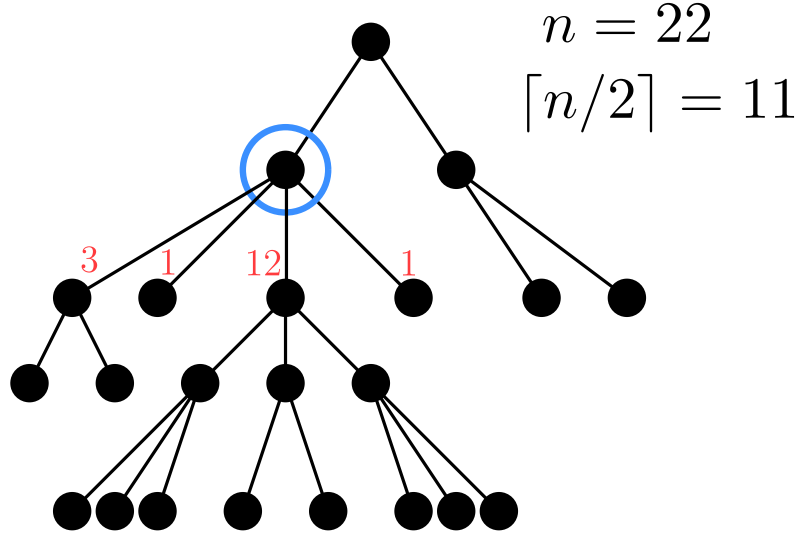


Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

Claim: Any tree T always contains a separator.

Walk through the tree starting at root, going into the subtree that contains $\geq \lceil n/2 \rceil$ nodes.

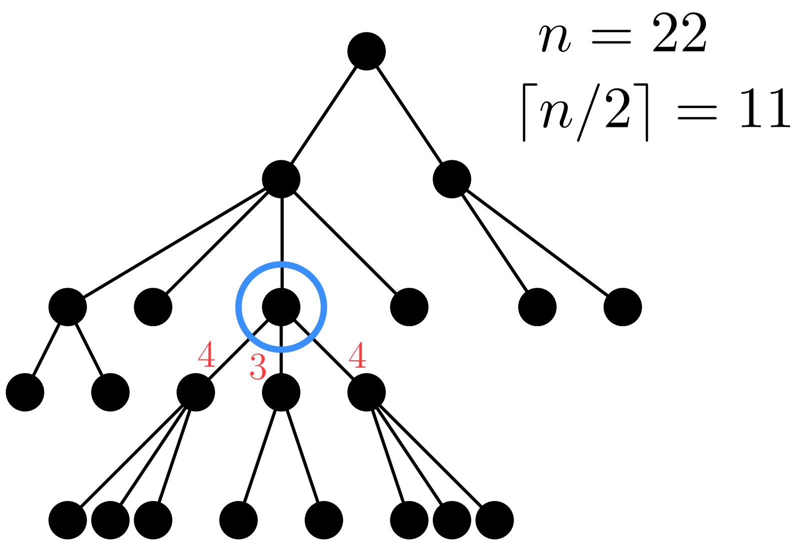


Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

Claim: Any tree T always contains a separator.

Walk through the tree starting at root, going into the subtree that contains $\geq \lceil n/2 \rceil$ nodes.

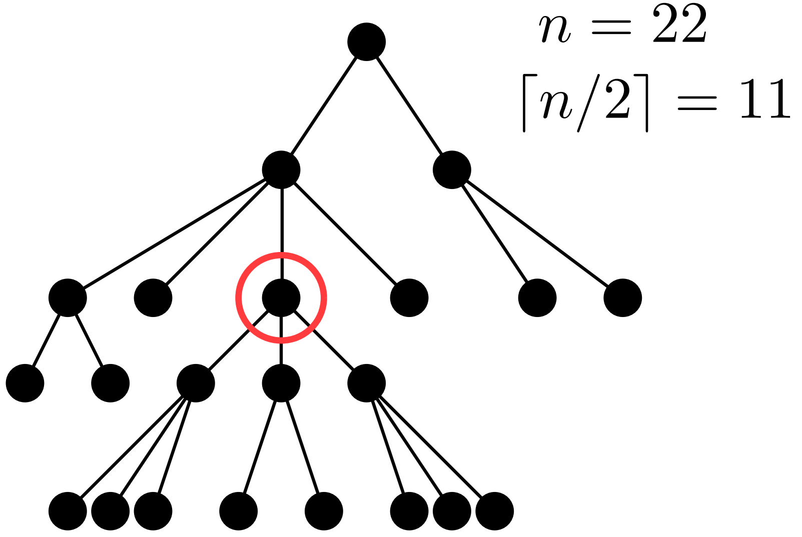


Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

Claim: Any tree T always contains a separator.

Walk through the tree starting at root, going into the subtree that contains $\geq \lceil n/2 \rceil$ nodes.

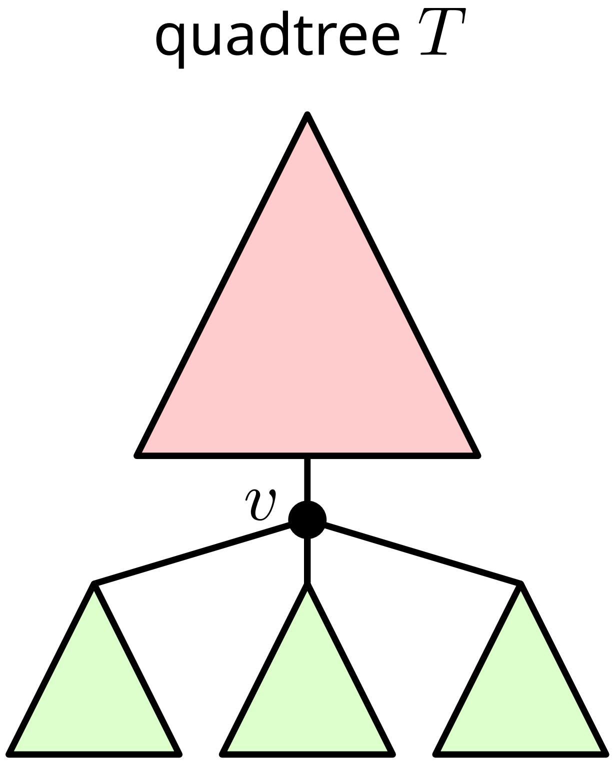


Once we get stuck:

- child subtree sizes $< \lceil n/2 \rceil$
- rooted subtree size $\leq n - \lceil n/2 \rceil \leq \lfloor n/2 \rfloor$

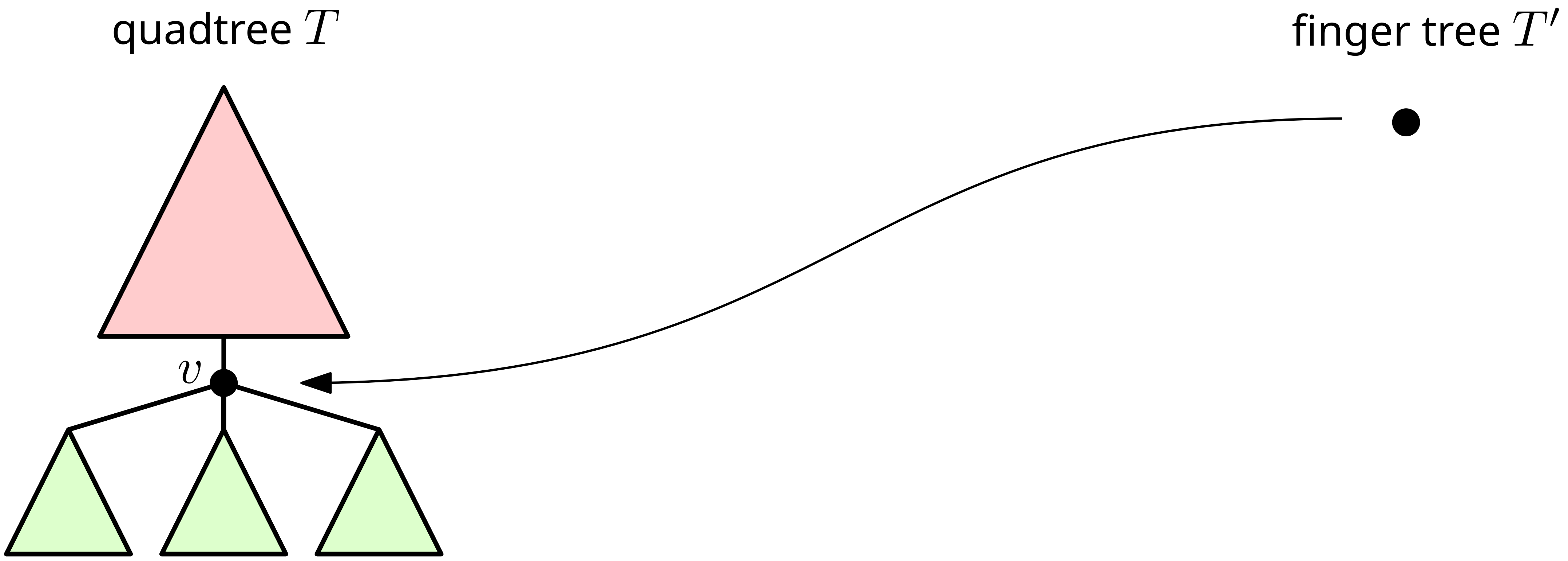
Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.



Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.



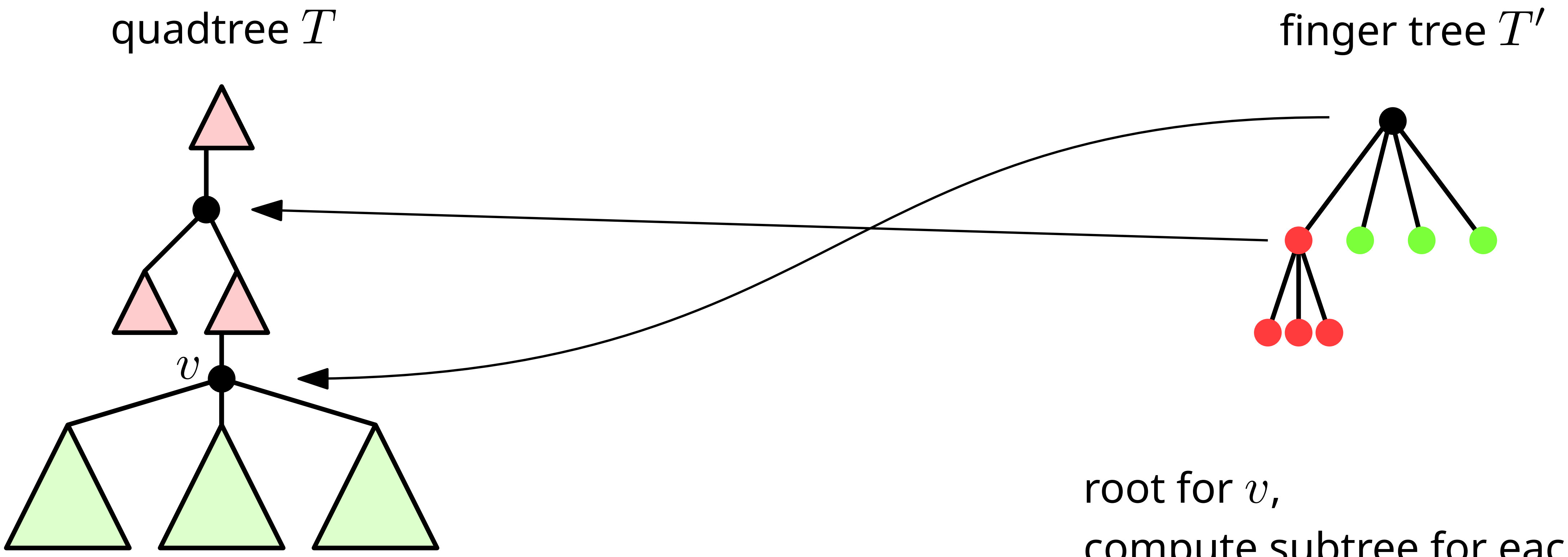
Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.

root for v,
compute subtree for each component recursively

Fast point-location - Fingering the quadtree

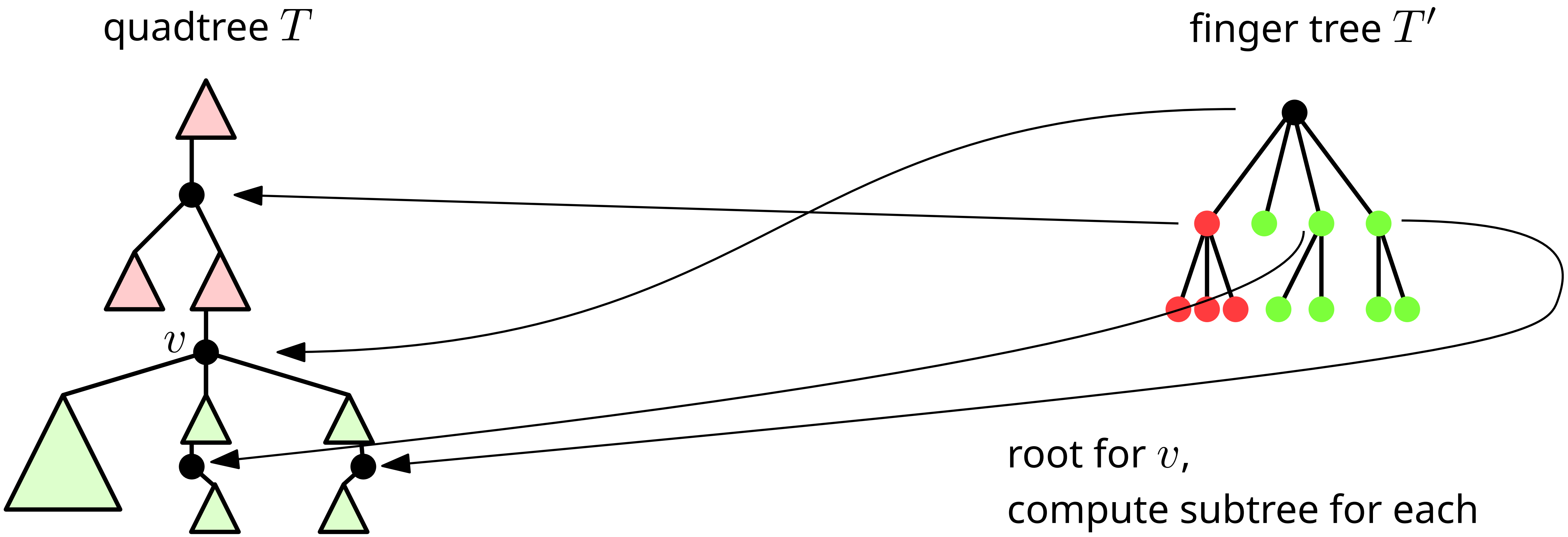
Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.



root for v,
compute subtree for each component recursively

Fast point-location - Fingering the quadtree

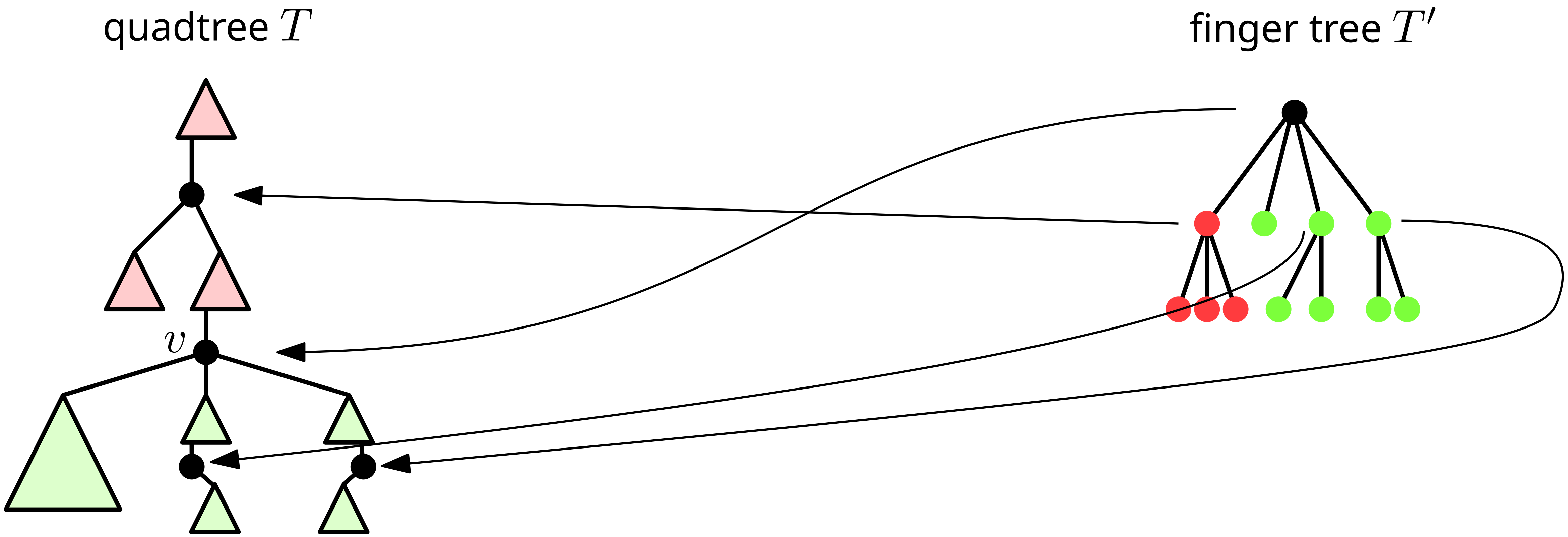
Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.



root for v,
compute subtree for each
component recursively

Fast point-location - Fingering the quadtree

Definition: A separator of a tree T with n nodes is a node $v \in T$ such that removing v results in a forest of which every tree has at most $\lceil n/2 \rceil$ nodes.



To query for point q, recursively, in time $O($height of $T')$:

- go into red subtree if $q \notin \square_v$
- search all $O(1)$ green subtrees if $q \in \square_v$

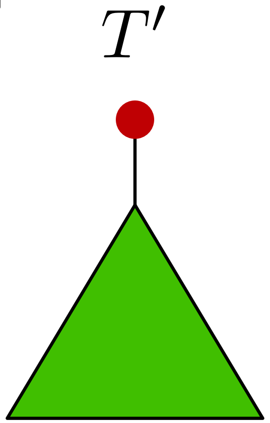
What are the height/query time and the construction time of the finger tree?

Finger trees

Recall that the separator splits T into subtrees of size $\leq \lceil n/2 \rceil$

recurrence for height:

$$\implies H(n) \leq 1 + H(\lceil n/2 \rceil) = O(\log n)$$

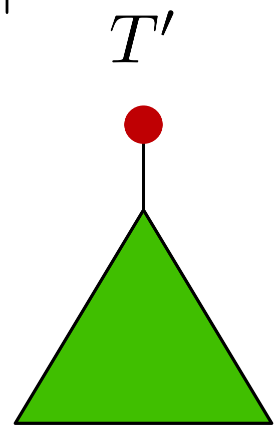


Finger trees

Recall that the separator splits T into subtrees of size $\leq \lceil n/2 \rceil$

recurrence for height:

$$\implies H(n) \leq 1 + H(\lceil n/2 \rceil) = O(\log n)$$

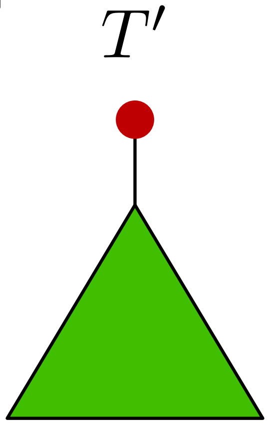


Finger trees

Recall that the separator splits T into subtrees of size $\leq \lceil n/2 \rceil$

recurrence for height:

$$\implies H(n) \leq 1 + H(\lceil n/2 \rceil) = O(\log n)$$

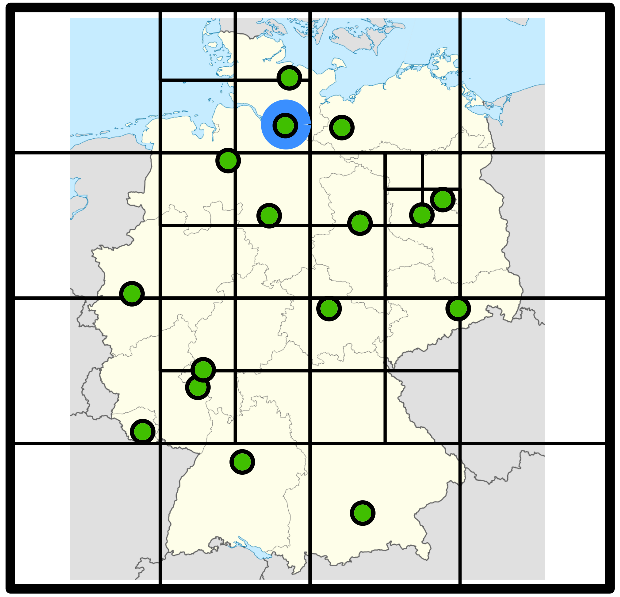


Construction time $T(n) = O(n) + \sum_{i=1}^{t} T(n_i)$ where $n_1 \ldots n_t$ are the sizes of the t subtrees formed after removing the separator.

Since $t = O(1)$ and $n_i \leq \lceil n/2 \rceil$, we have $T(n) = O(n \log n)$

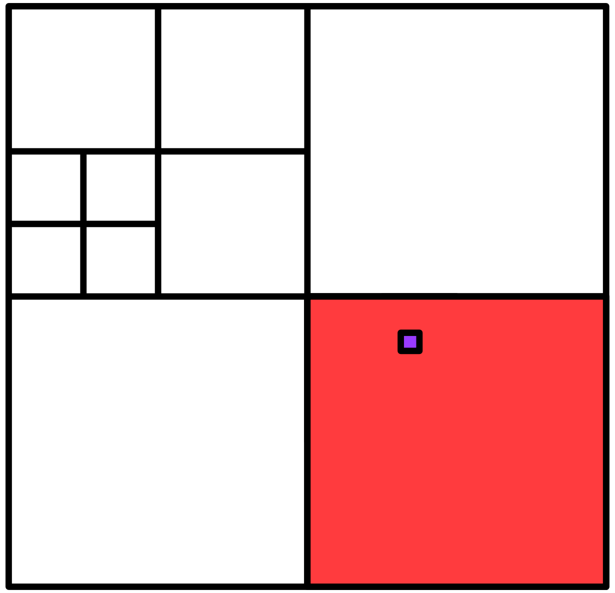
Summary

Normal quadtrees



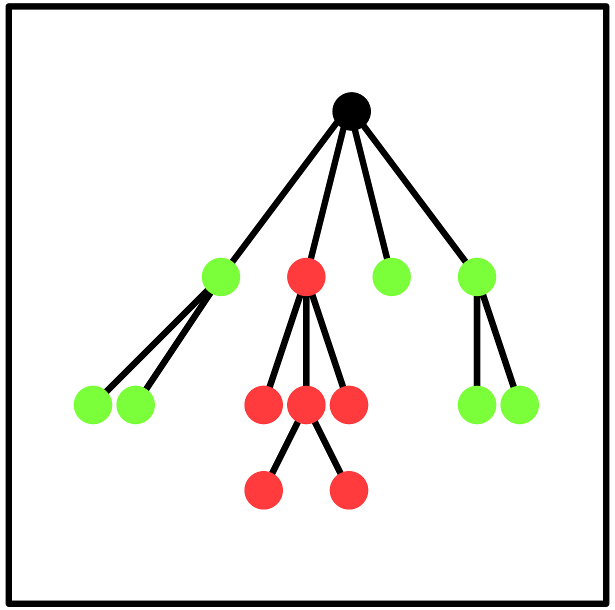
Bounded by spread

Compressed quadtrees



Bounded by number of points

Finger trees



Fast query time

more in book: dynamic quadtrees