# Location & Information

Maarten Löffler
*Utrecht University*

*Based on joint work with*

Kevin Buchin
Ivor van der Hoog
Irina Kostitsyna
Marc van Kreveld
Joe Mitchell
Pat Morin
Wolfgang Mulzer
Jack Snoeyink
Bettina Speckmann

# Chapter 1
*PROXIMITY STRUCTURES*

Point sets and proximity

Computational questions

State of the art in 1D

State of the art in 2D

# Chapter 2
*PARTIAL INFORMATION*

Uncertain locations

Quantifying information

State of the art in 1D

State of the art in 2D

# Chapter 3
*FINE-GRAINED MEASURES*

Ambiguity of regions

Information theory

State of the art in 1D

State of the art in 2D

# Chapter 1
## *PROXIMITY STRUCTURES*

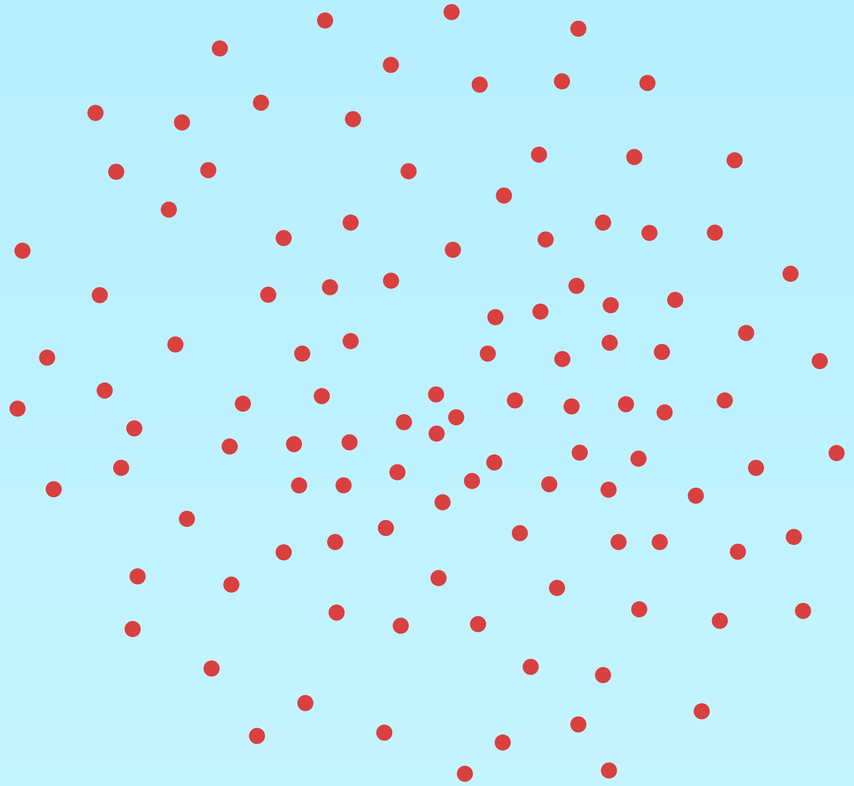
Point sets and proximity

Computational questions

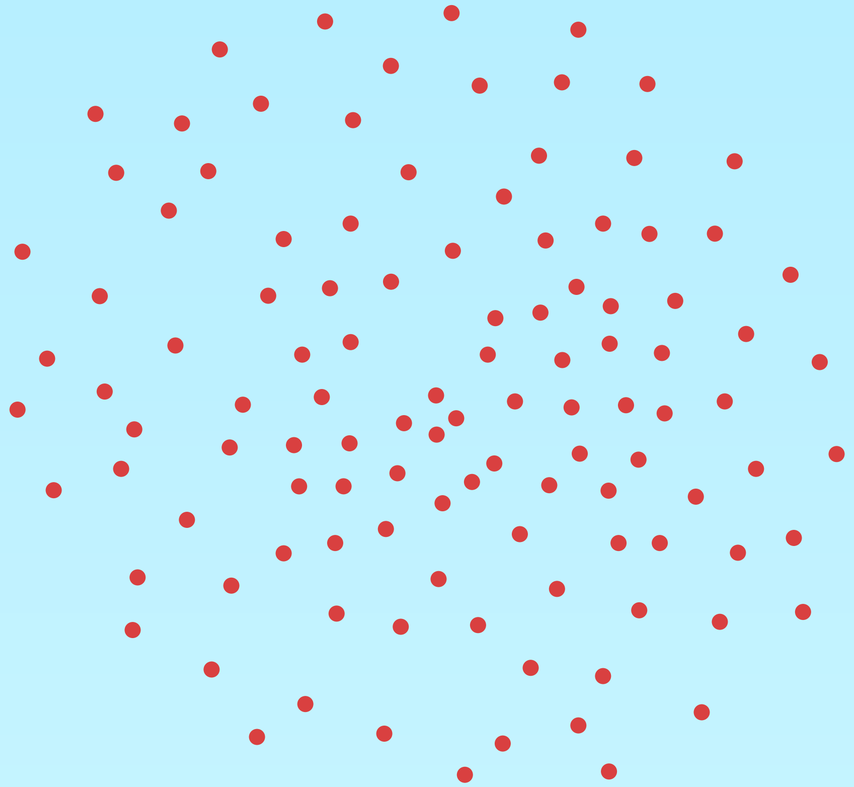
State of the art in 1D

State of the art in 2D

Let $P$ be a set of $n$ points in the plane.

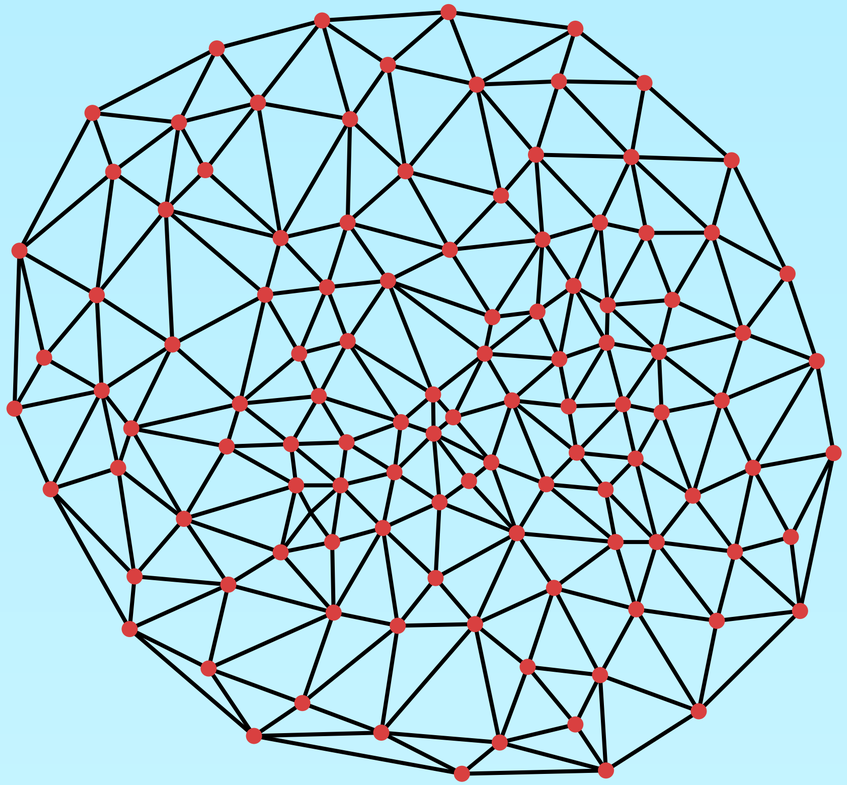


Let $P$ be a set of $n$ points in the plane.



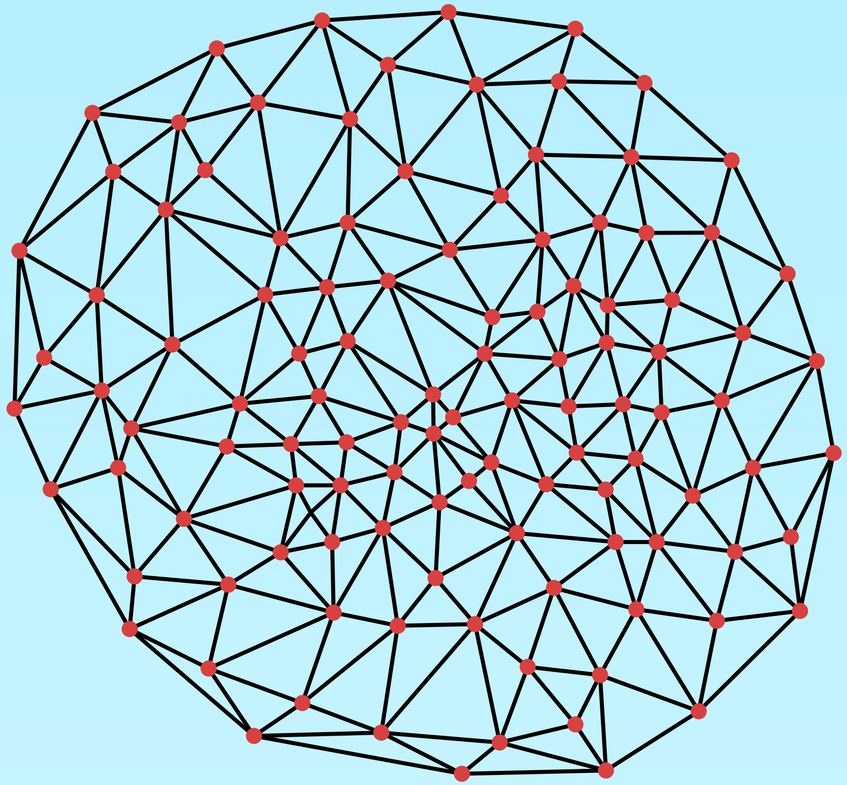
Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.



Let $P$ be a set of $n$ points in the plane.

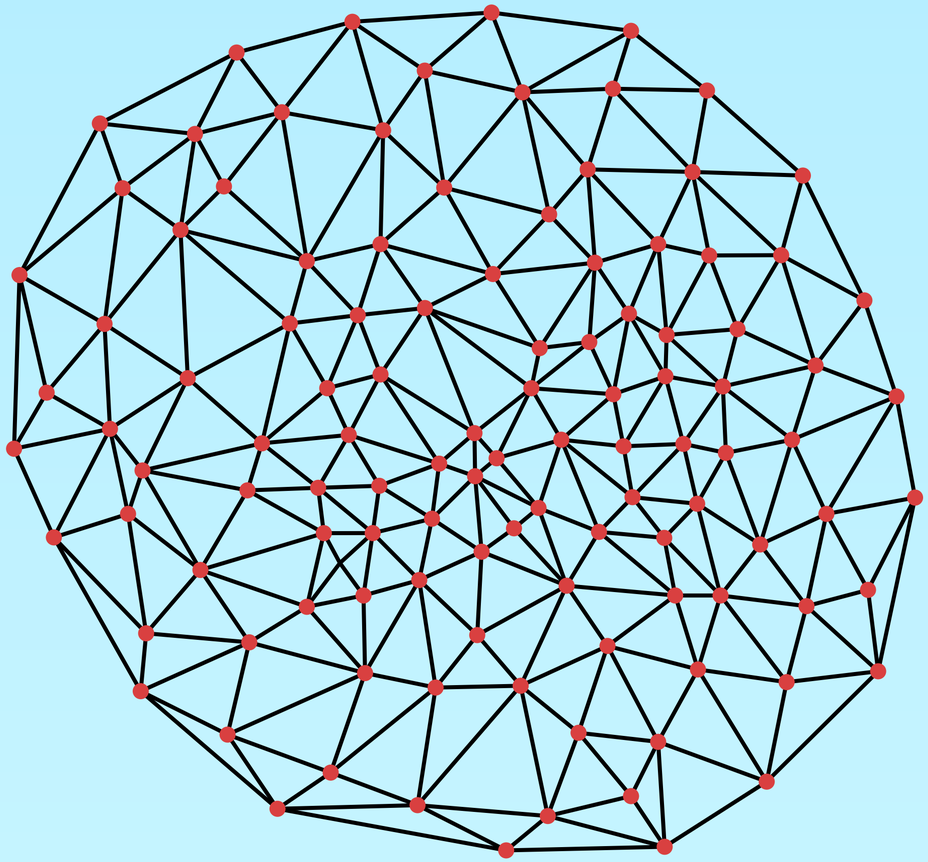
A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

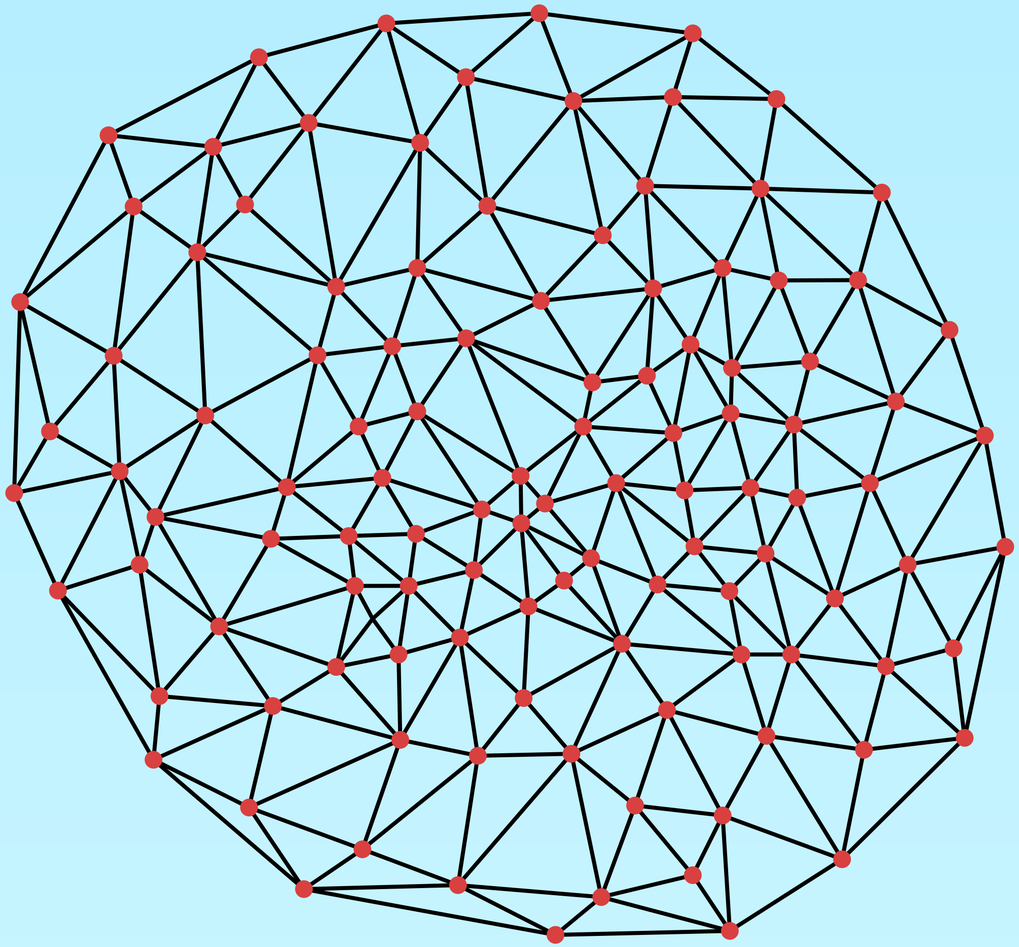
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

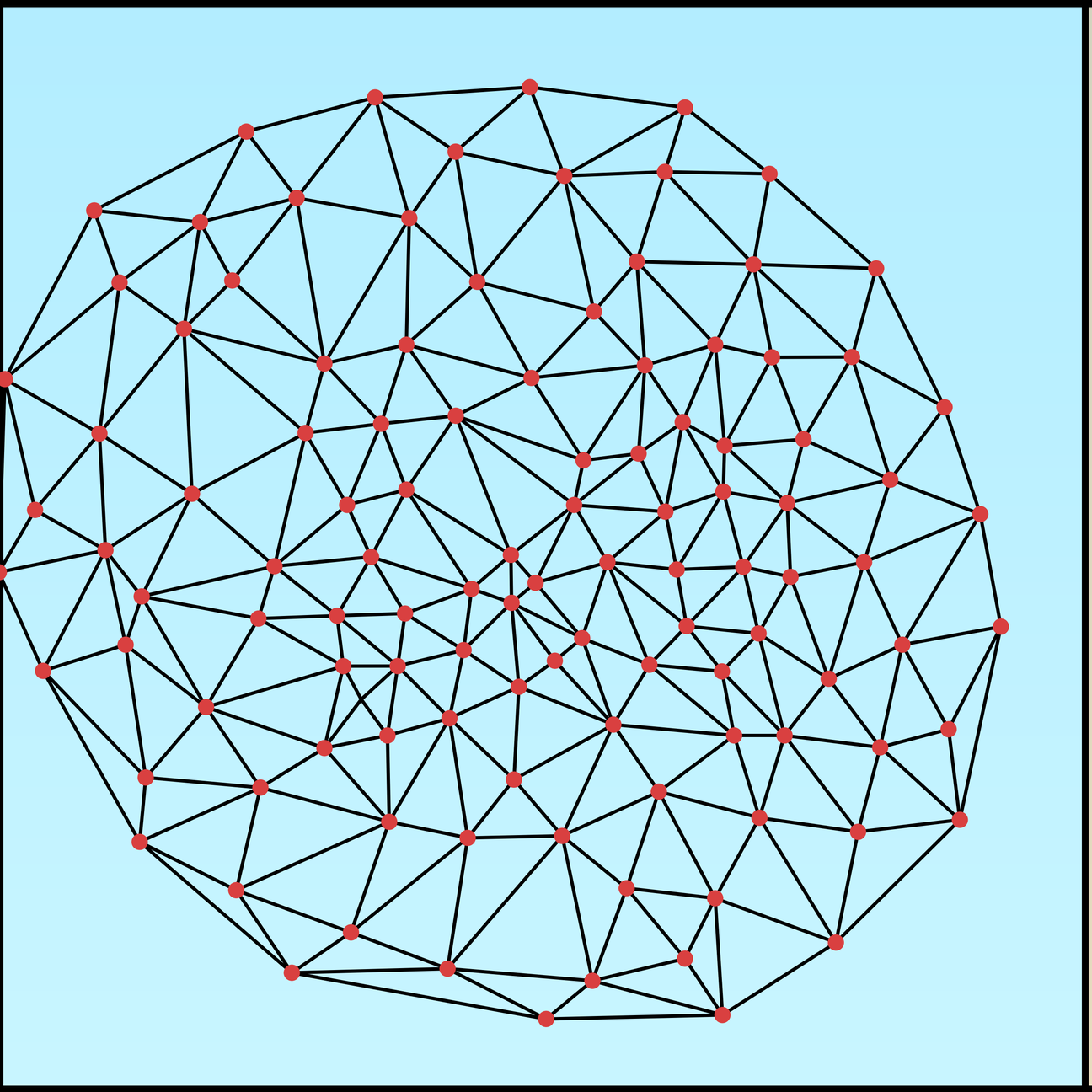
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

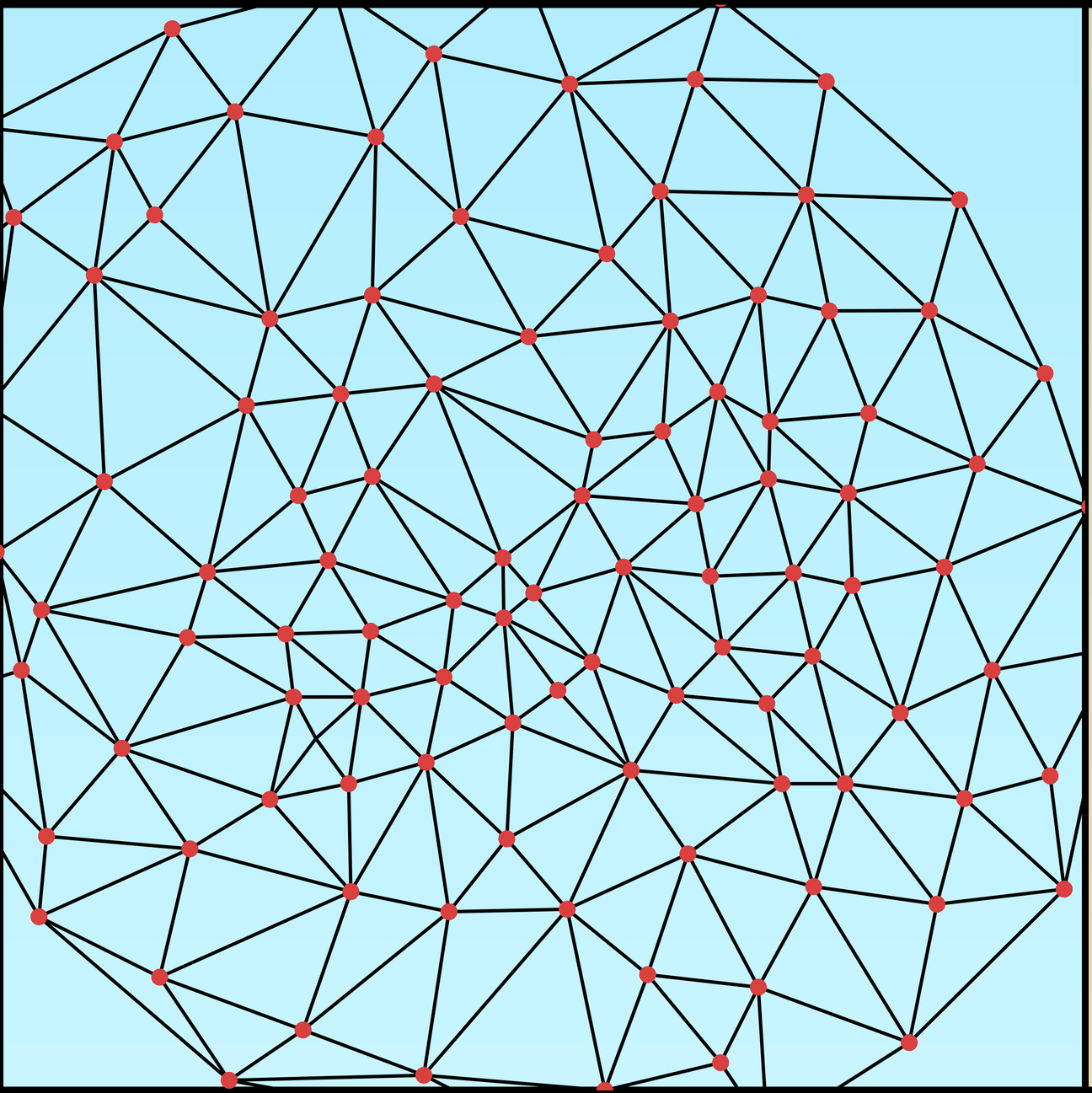
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

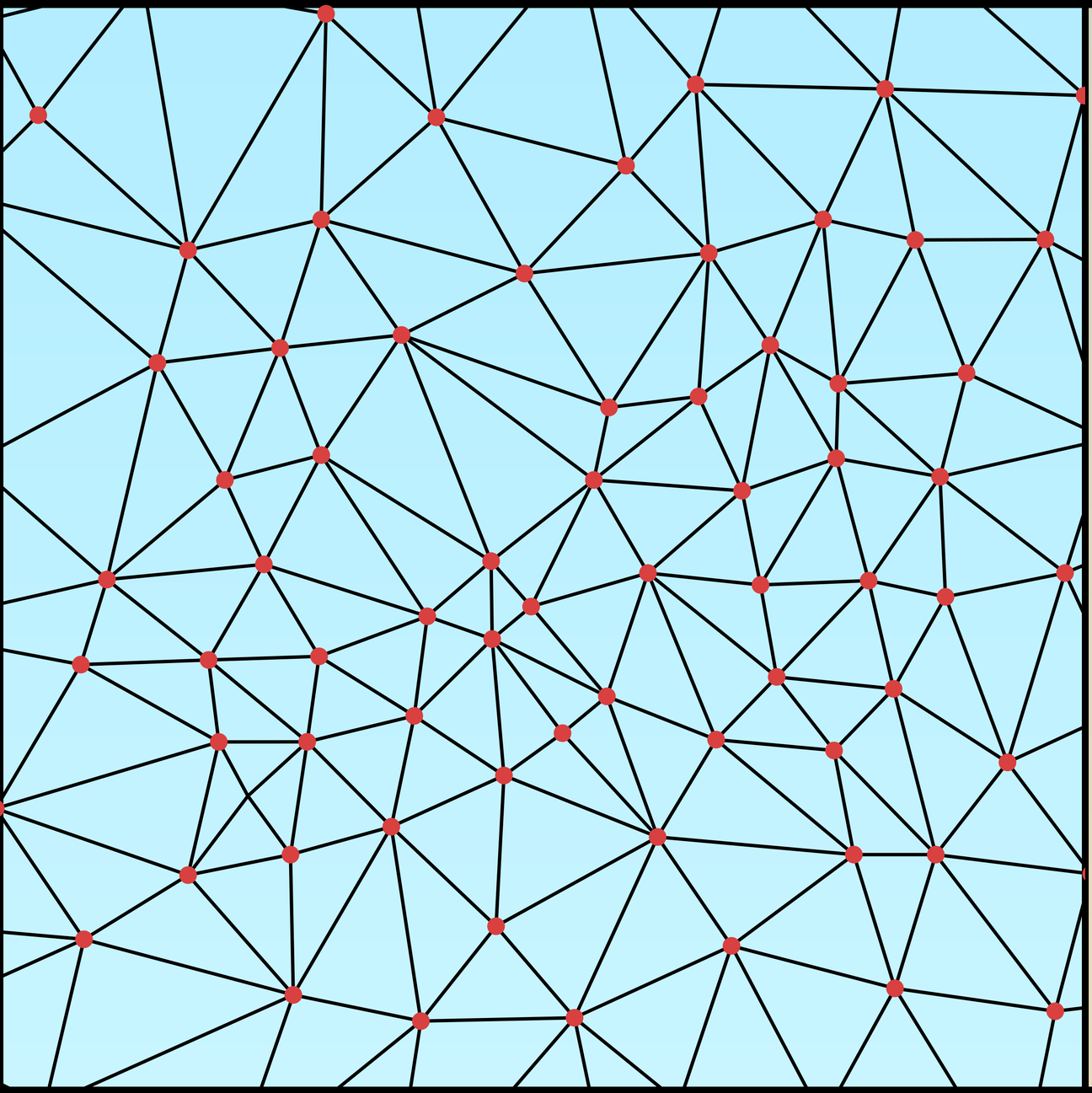
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

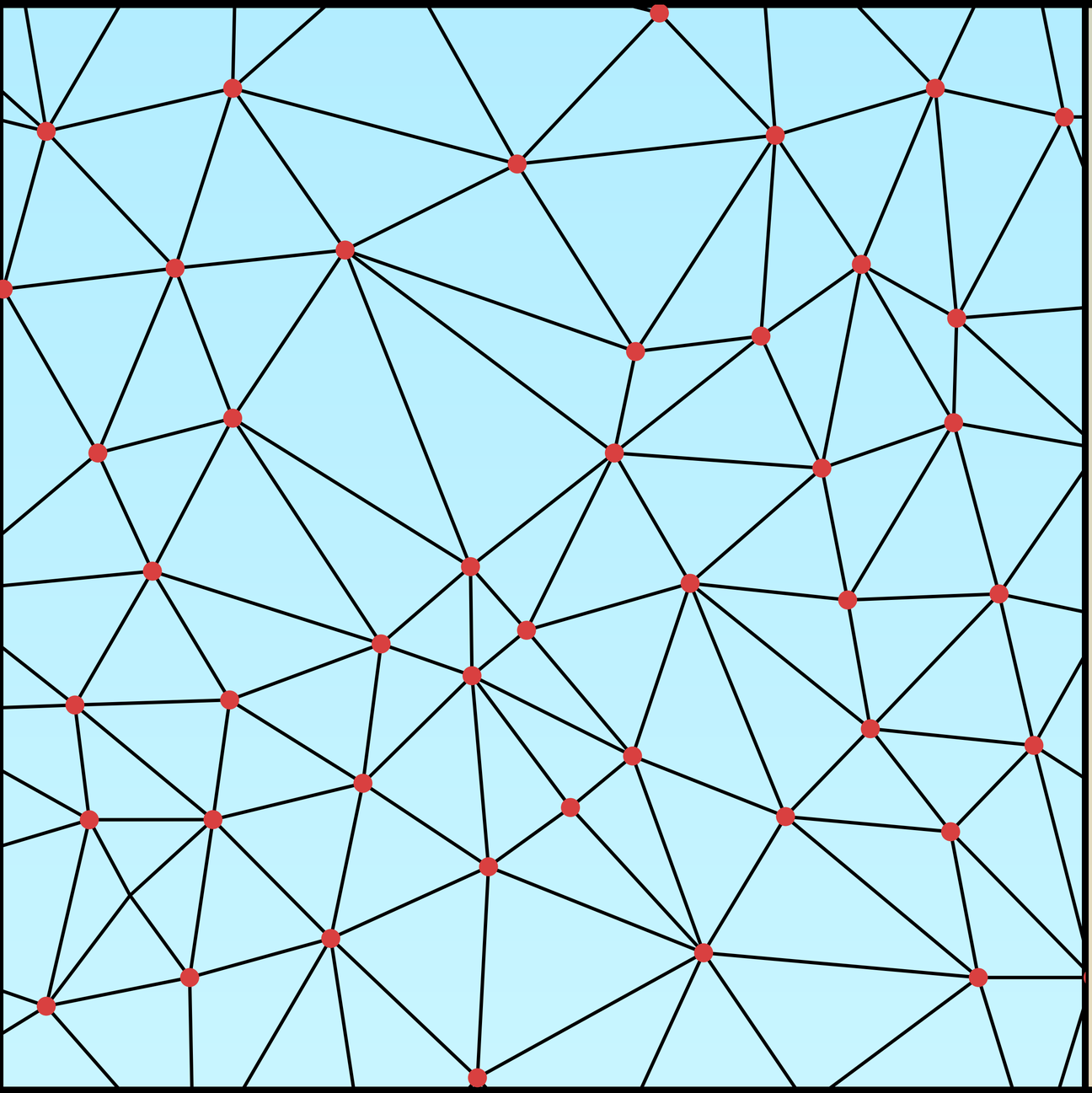
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

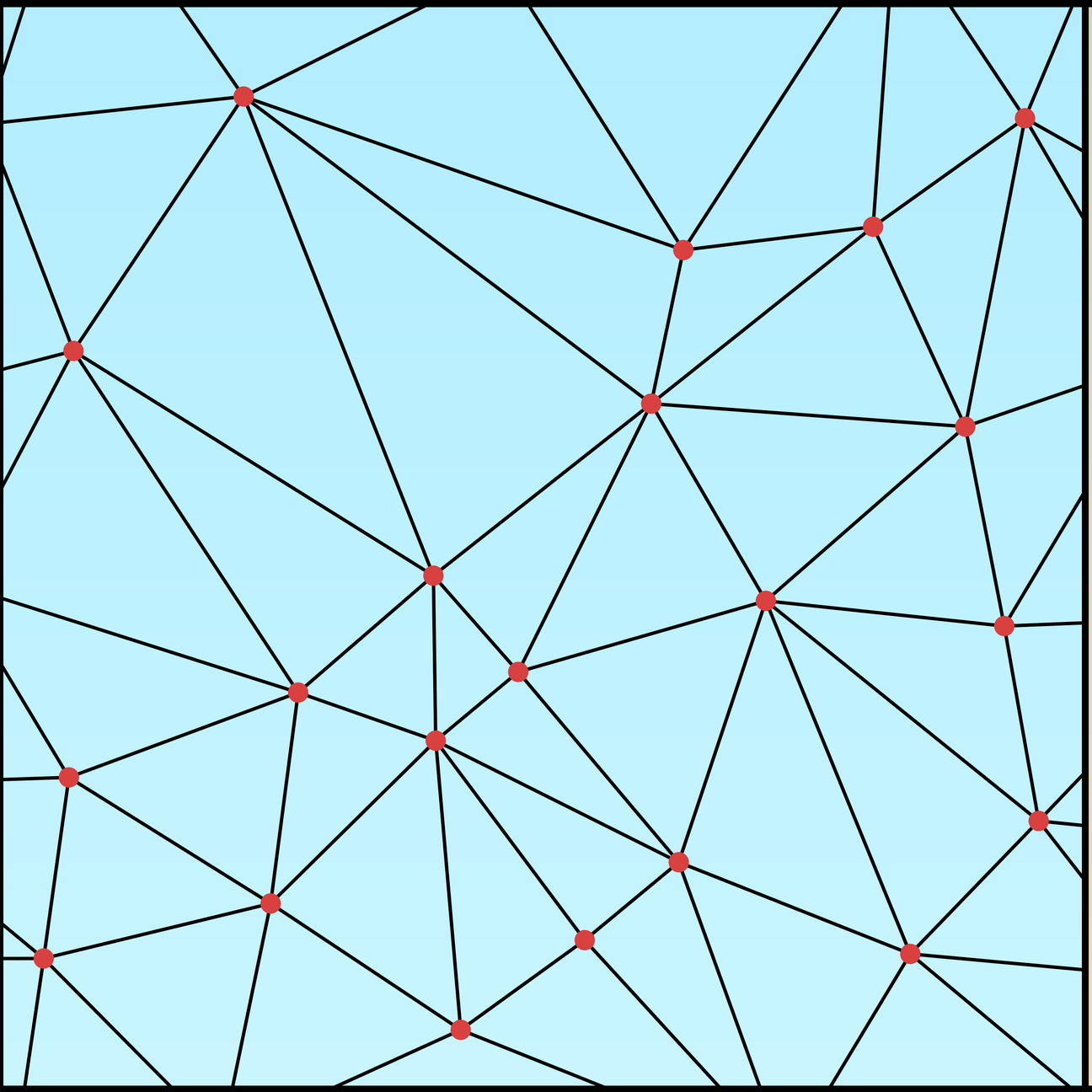
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

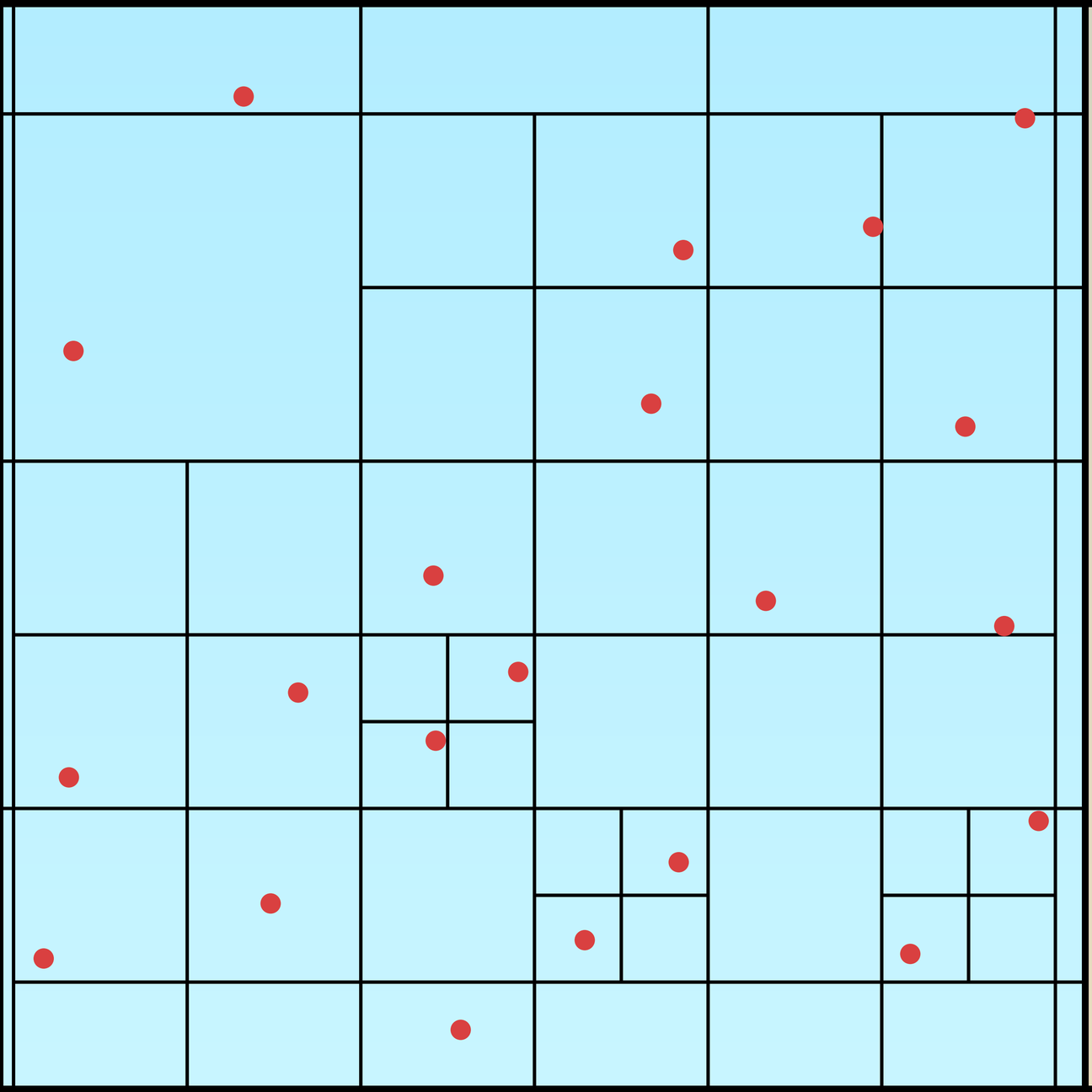
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

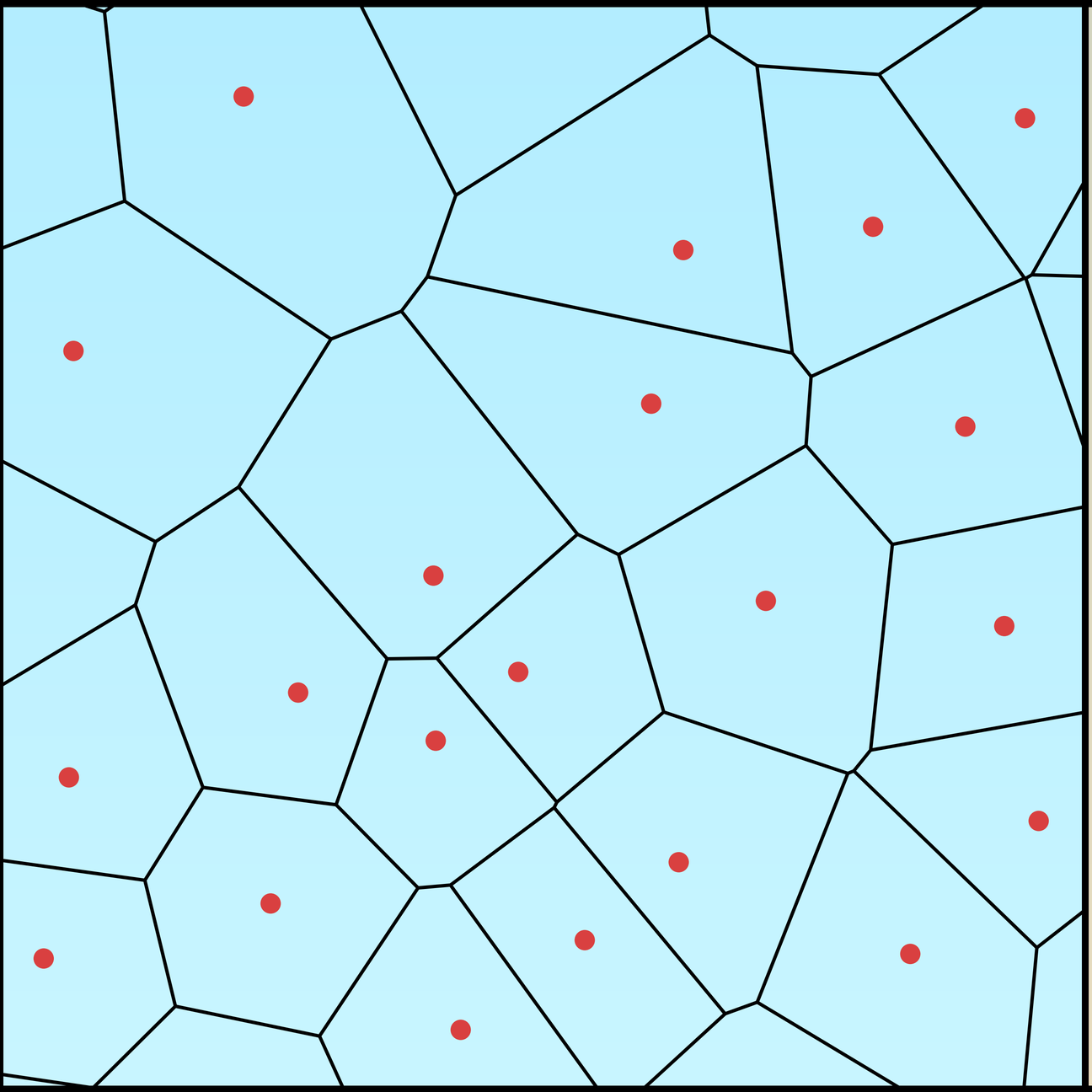
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

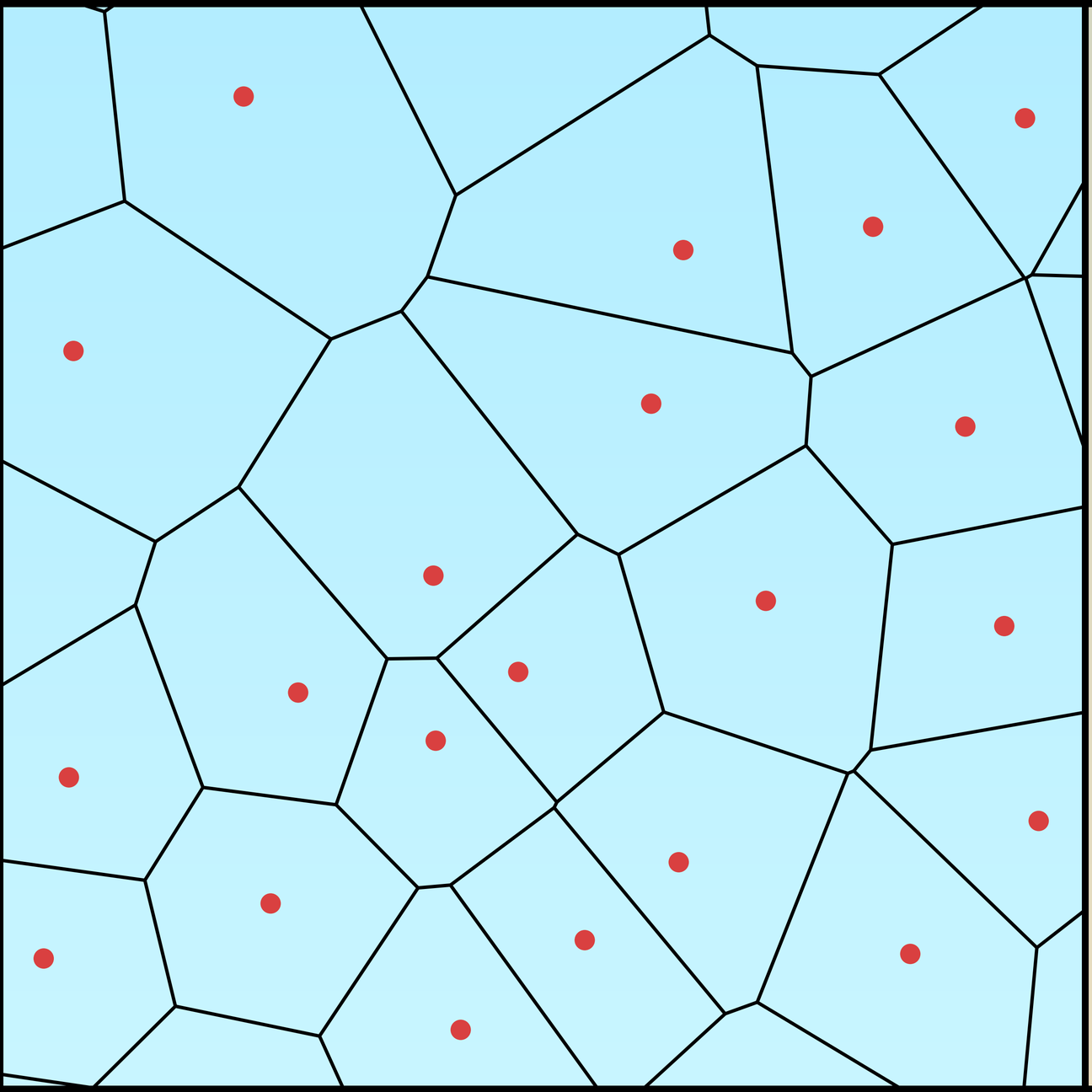
Intuitively: structure does not depend on points that are far away from me.



Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is ‘a structure that stores some kind of useful local information’.

Intuitively: structure does not depend on points that are far away from me.



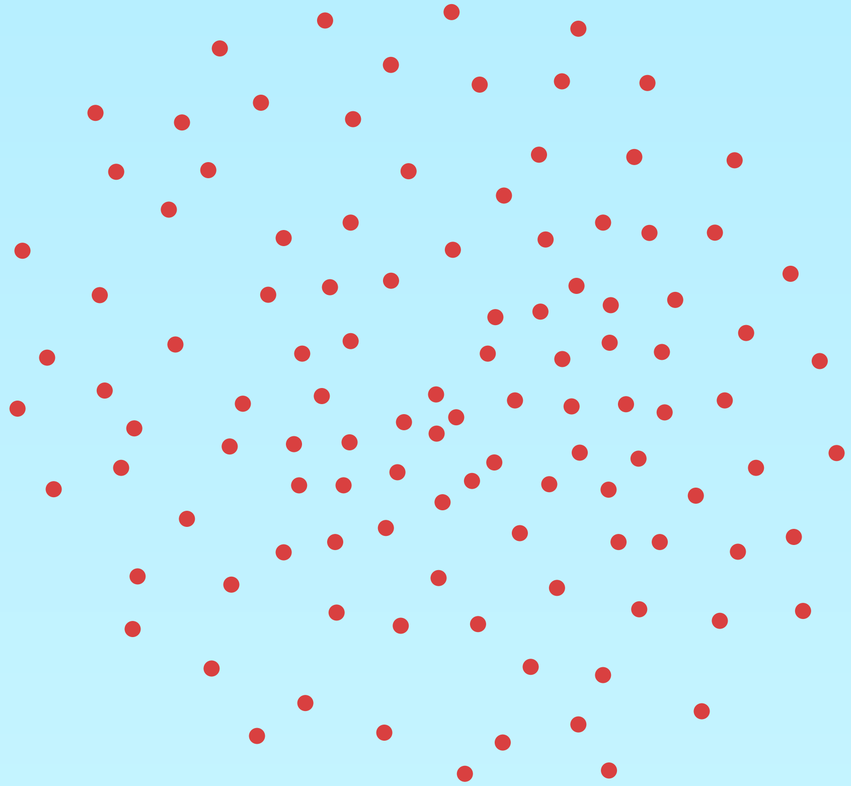
Let $P$ be a set of $n$ points in the plane.

A *proximity structure* on $P$ is 'a structure that stores some kind of useful local information'.

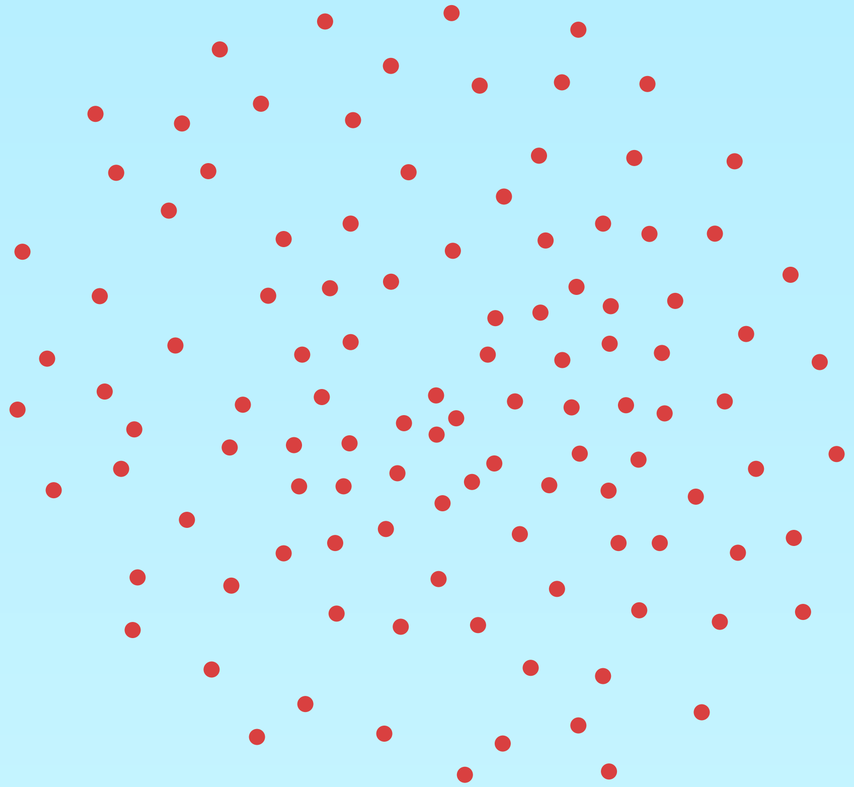
Intuitively: structure does not depend on points that are far away from me.

Because defined locally, complexity should be $O(n)$.

For comparison, there are also global properties of a point set $P$.

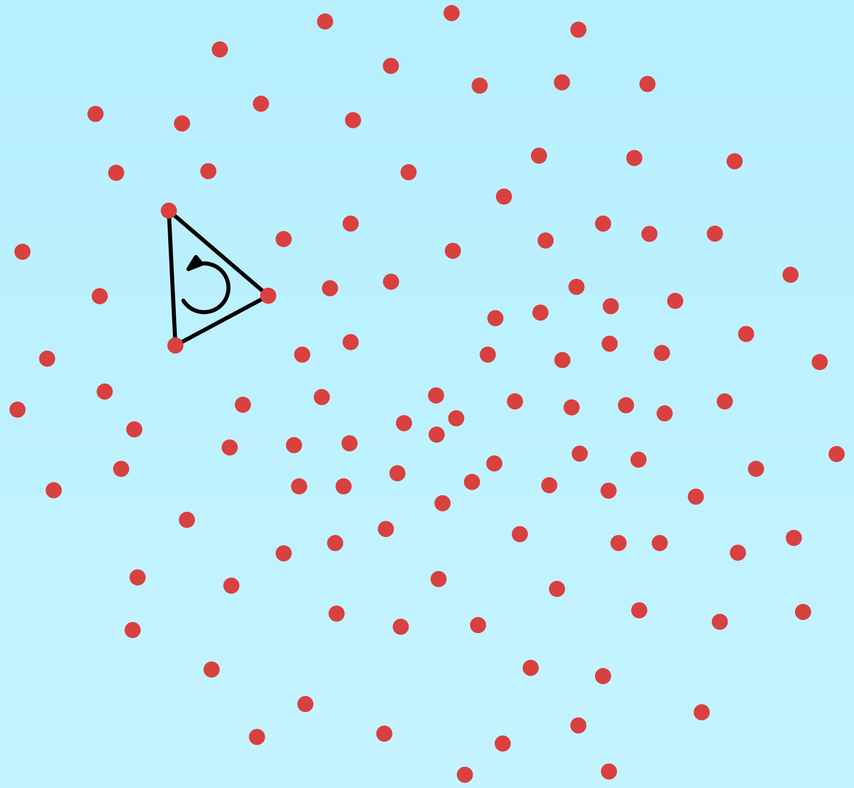


For comparison, there are also global properties of a point set $P$.



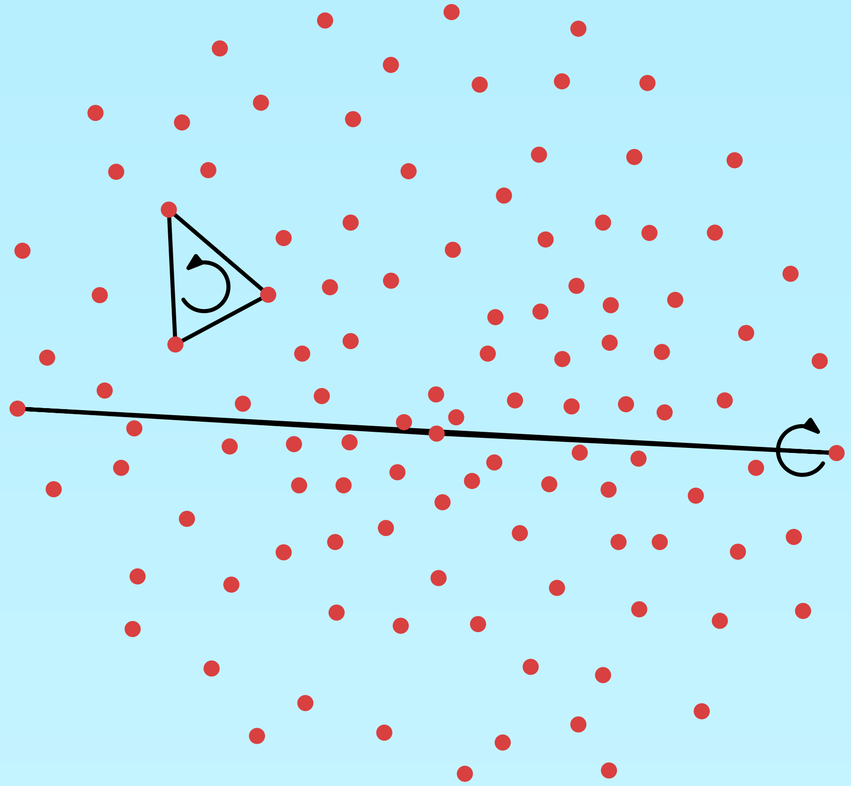
For comparison, there are also global properties of a point set $P$.

The *order type* of $P$ describes for every *triple* of points whether they are in clockwise or counterclockwise order.



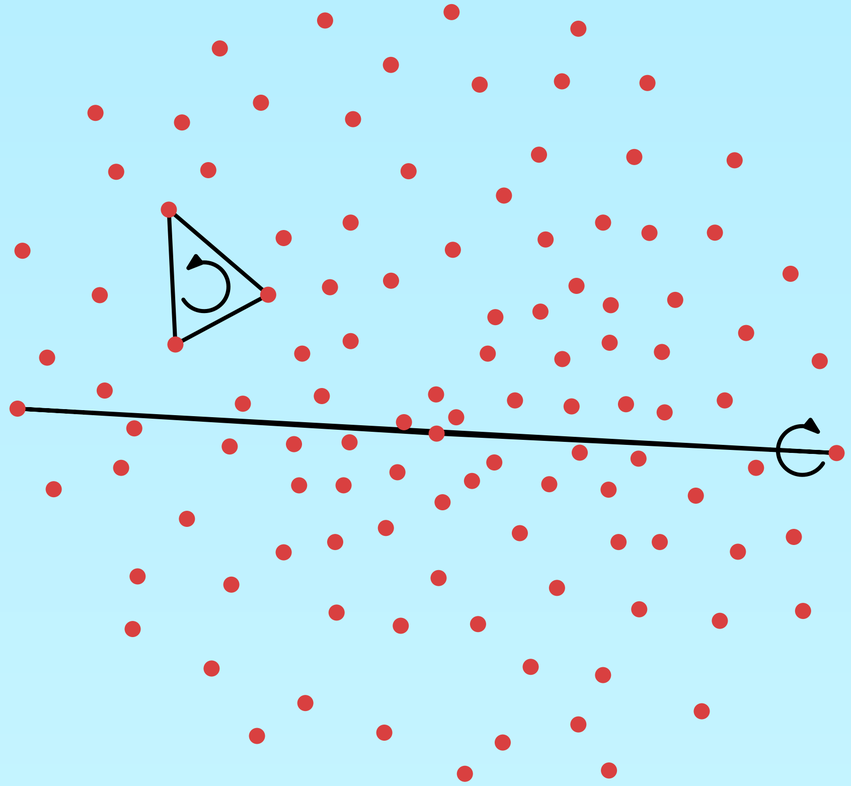
For comparison, there are also global properties of a point set $P$.

The *order type* of $P$ describes for every *triple* of points whether they are in clockwise or counterclockwise order.



For comparison, there are also global properties of a point set $P$.

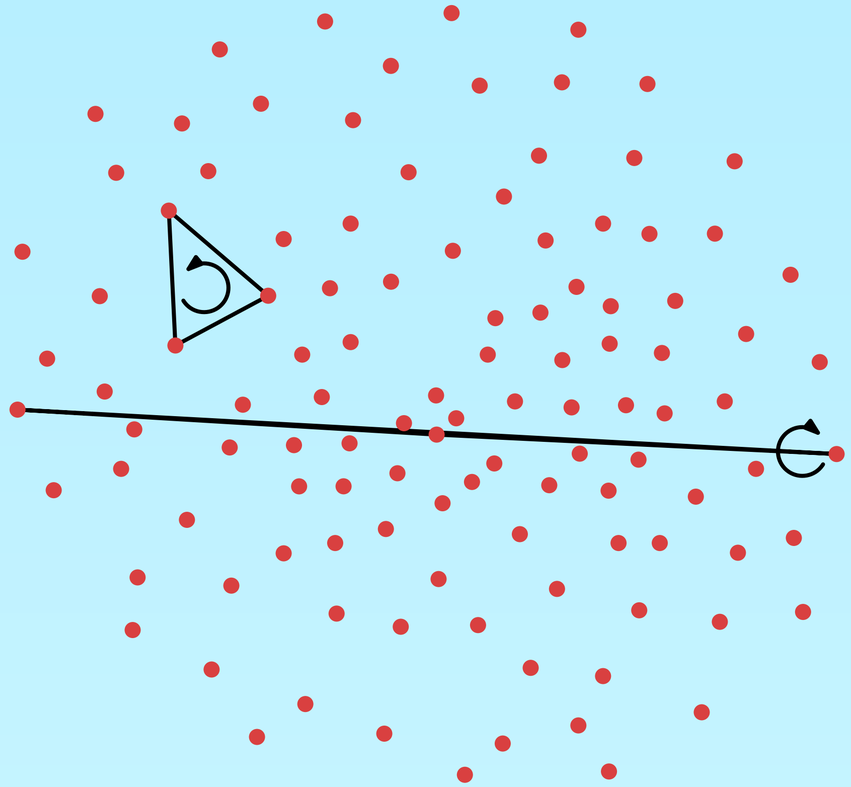
The *order type* of $P$ describes for every *triple* of points whether they are in clockwise or counterclockwise order.



For comparison, there are also global properties of a point set $P$.

The *order type* of $P$ describes for every *triple* of points whether they are in clockwise or counterclockwise order.

The *order type* has $O(n^2)$ (quadratic) complexity.



For comparison, there are also global properties of a point set $P$.

The *order type* of $P$ describes for every *triple* of points whether they are in clockwise or counterclockwise order.

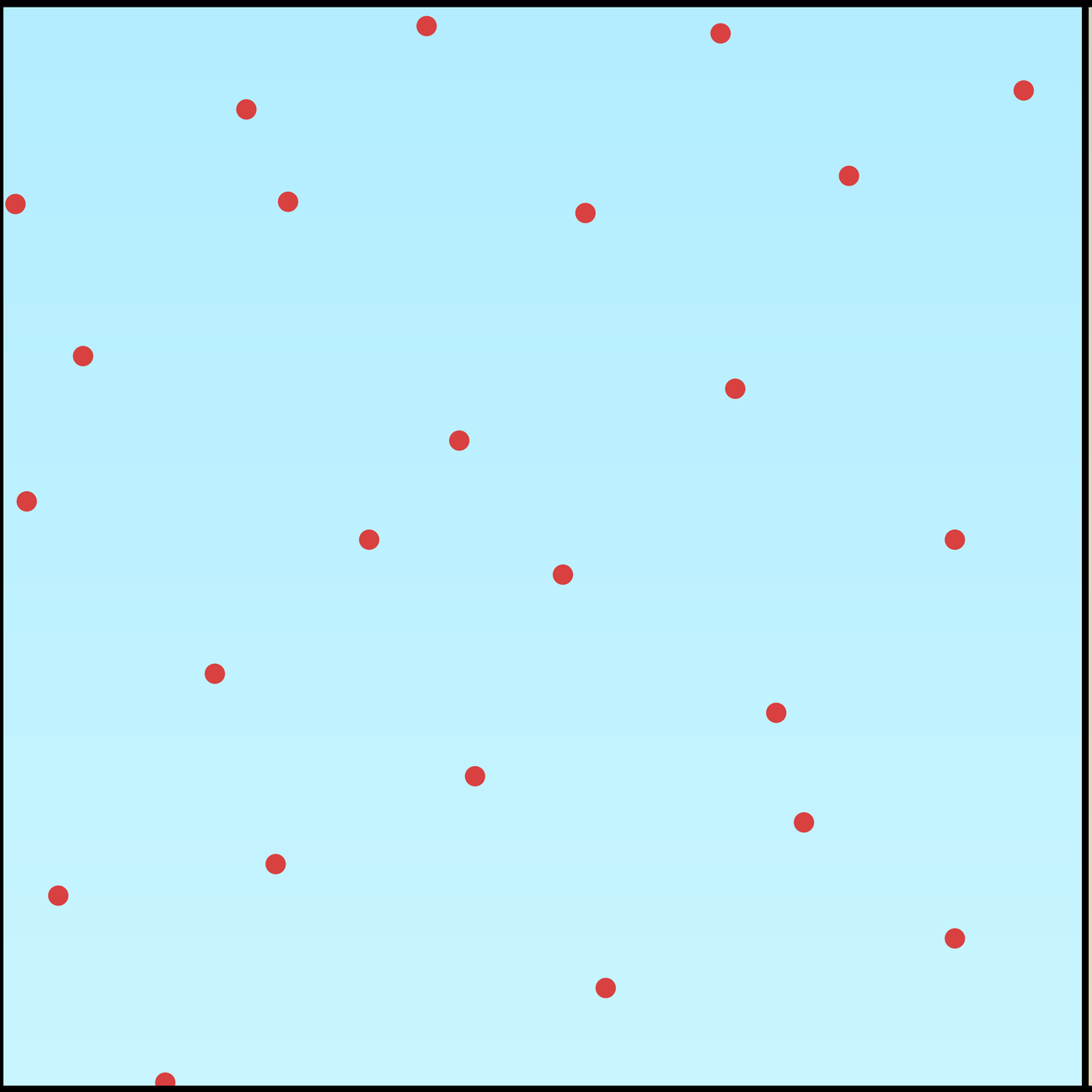
The *order type* has $O(n^2)$ (quadratic) complexity.

Very non-local!

Many useful proximity structures have been introduced over the years.

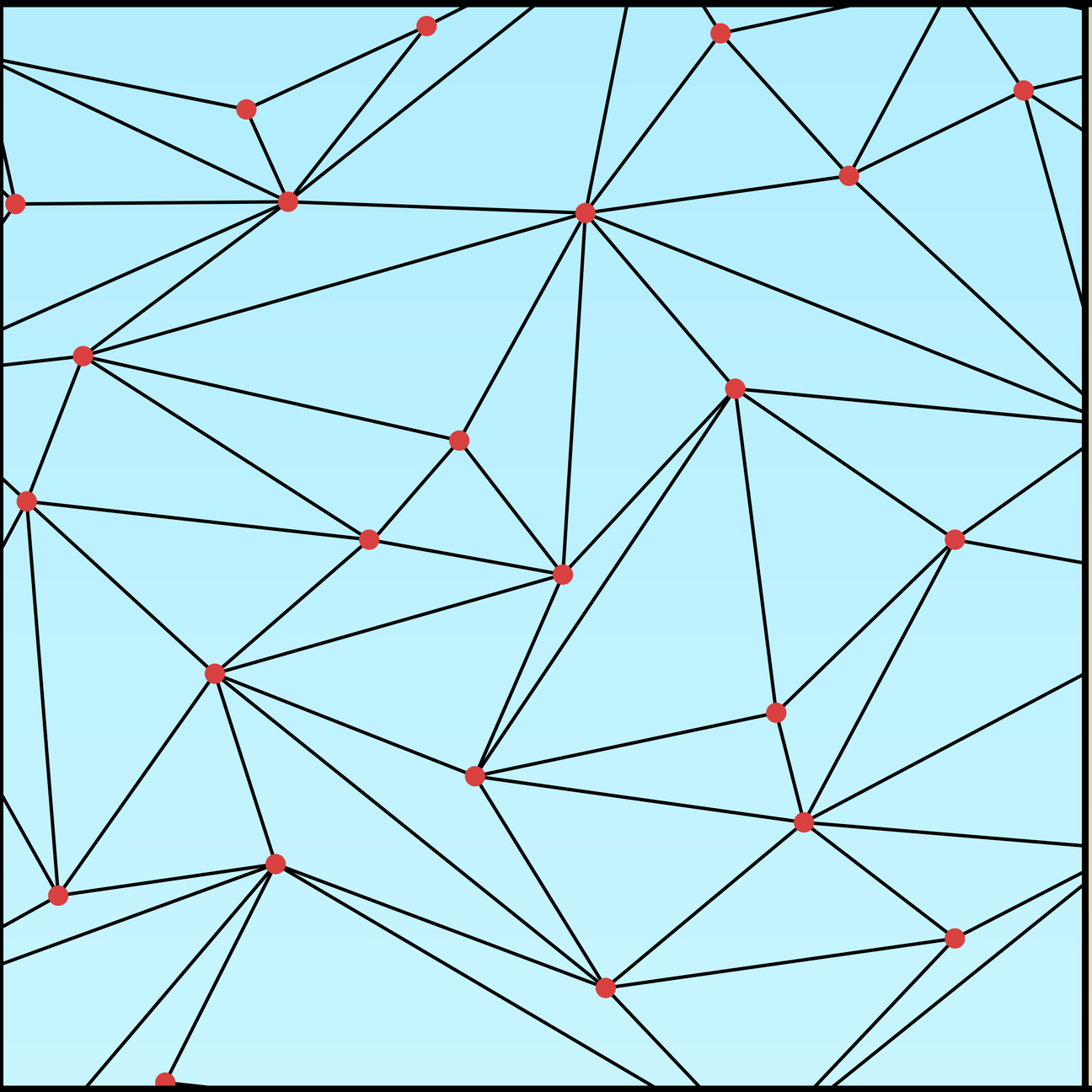
Many useful proximity structures have been introduced over the years.

Let $P$ be a set of $n$ points in the plane.



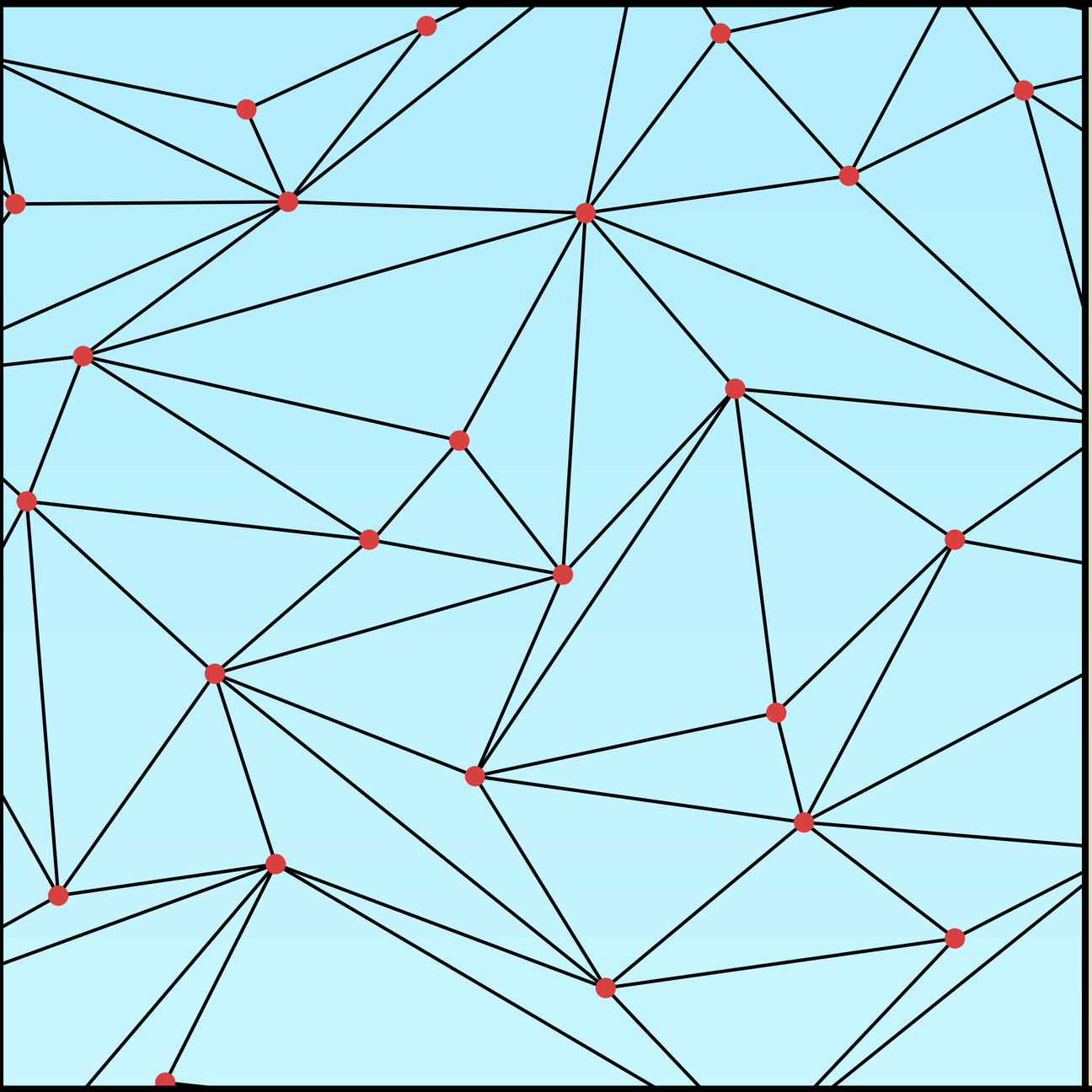
Many useful proximity structures have been introduced over the years.

Let $P$ be a set of $n$ points in the plane.



Many useful proximity structures have been introduced over the years.

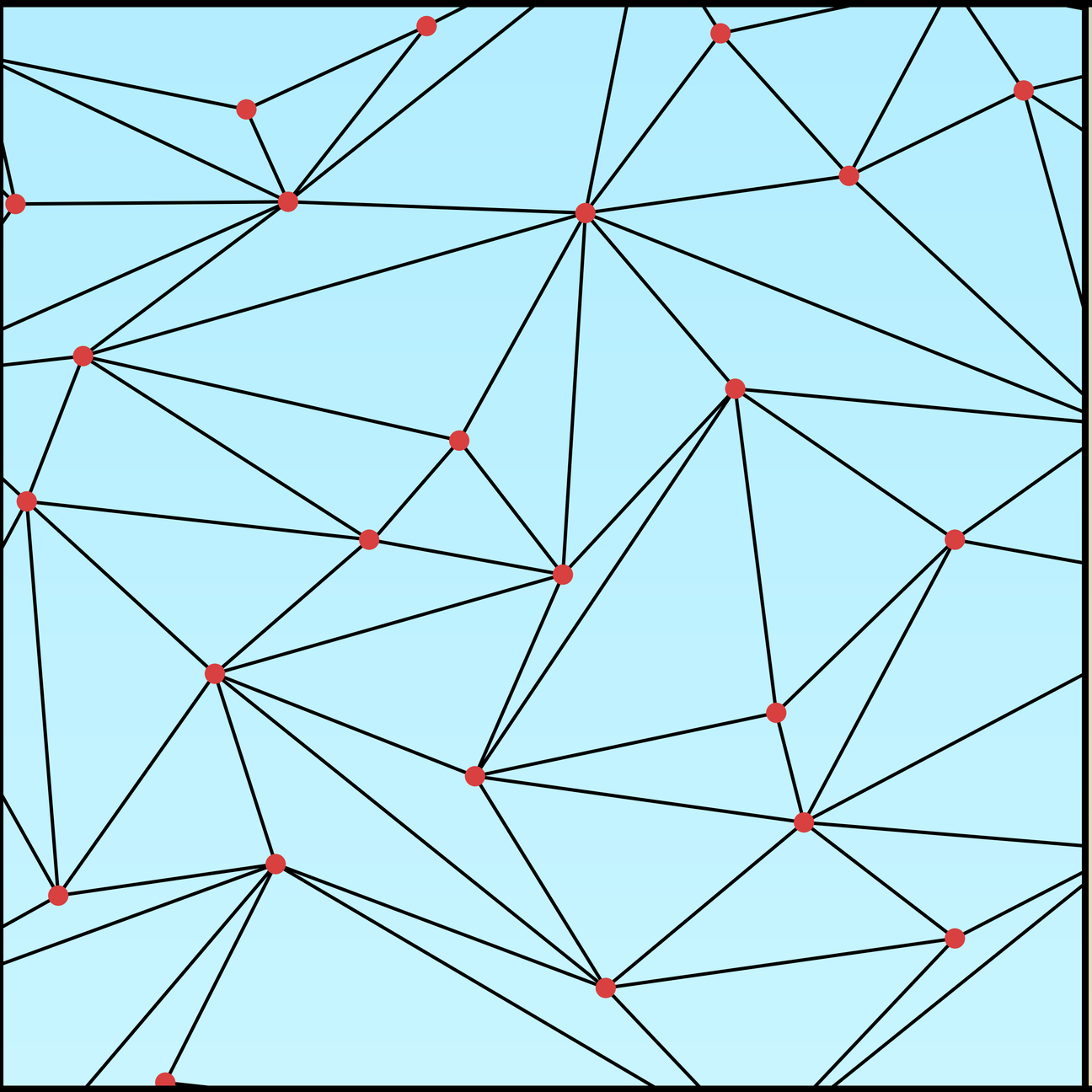
Let $P$ be a set of $n$ points in the plane.



Many useful proximity structures have been introduced over the years.

Let $P$ be a set of $n$ points in the plane.

A *triangulation* of $P$ is a *plane* graph on $P$, such that each face is a triangle.

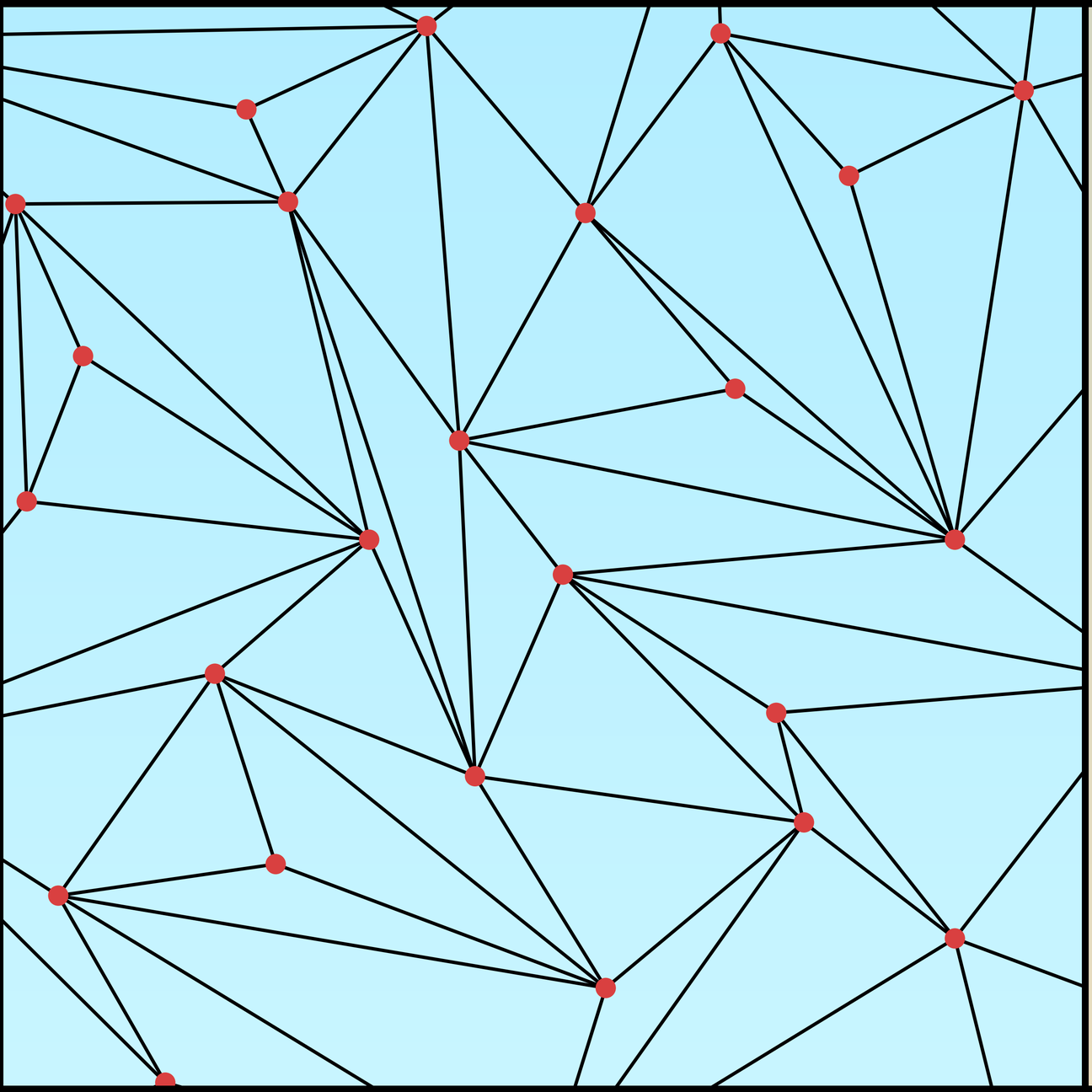


Many useful proximity structures have been introduced over the years.

Let $P$ be a set of $n$ points in the plane.

A *triangulation* of $P$ is a *plane* graph on $P$, such that each face is a triangle.

There are many different triangulations of $P$.

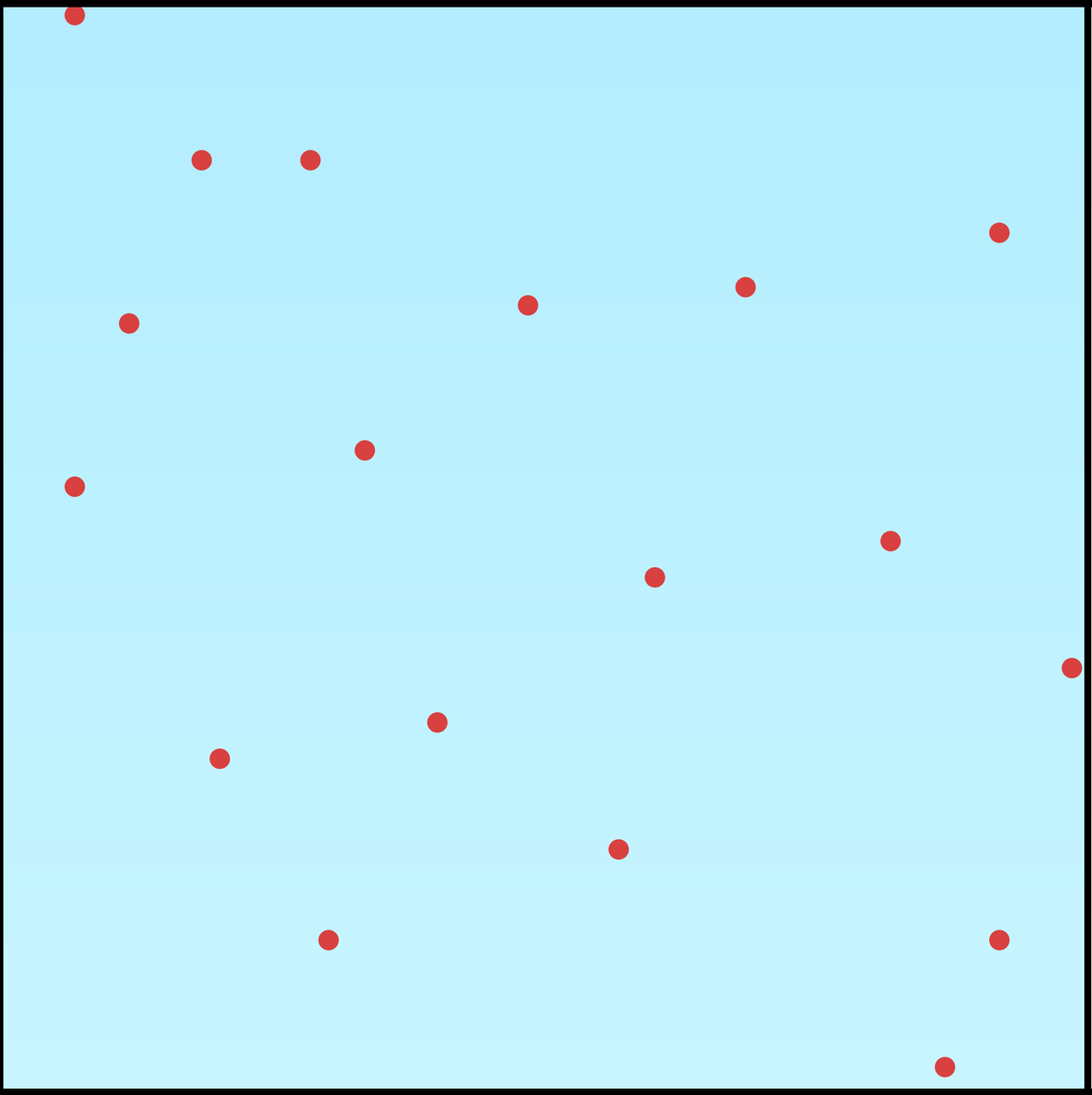


Many useful proximity structures have been introduced over the years.

Let $P$ be a set of $n$ points in the plane.

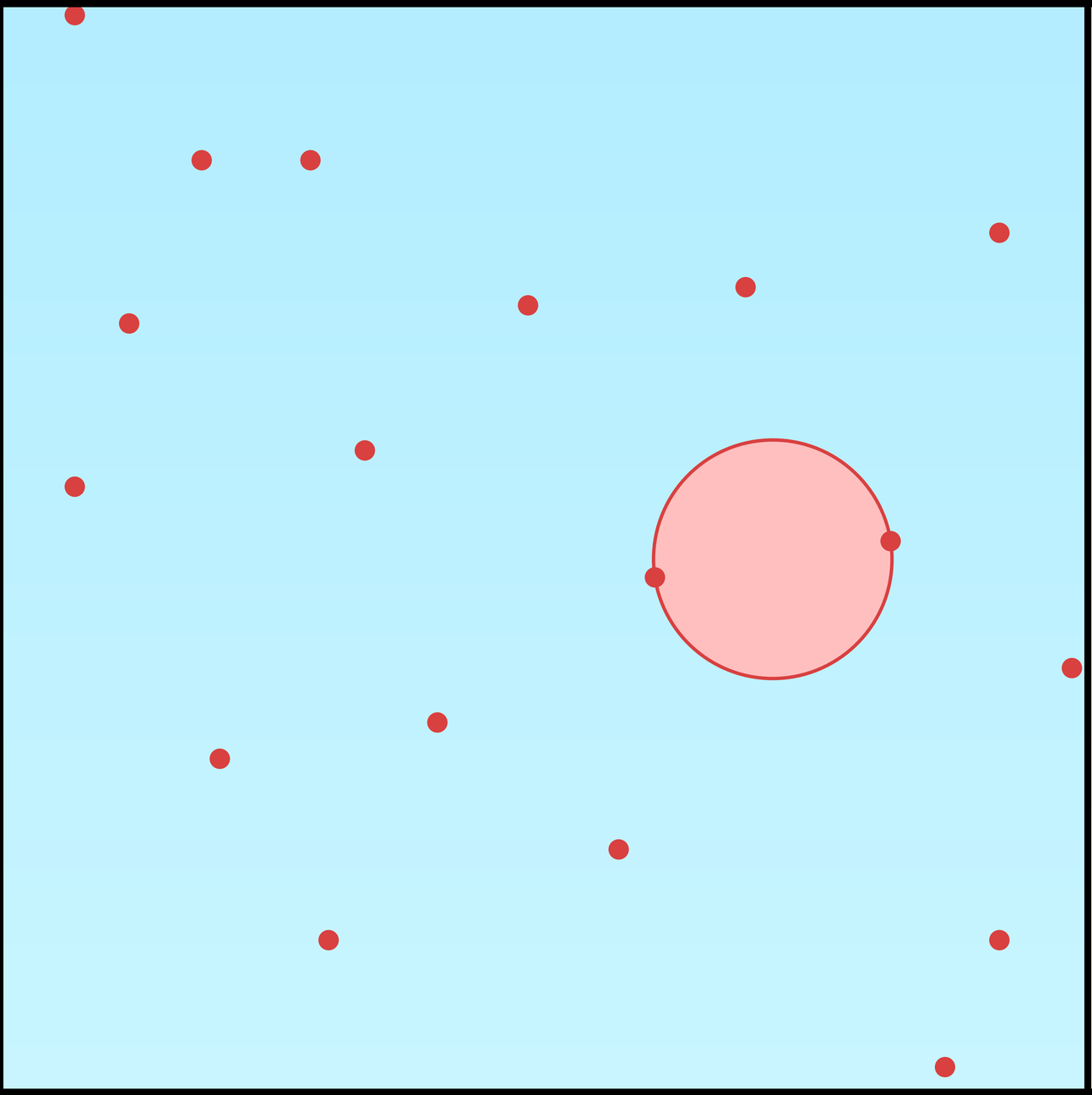
A *triangulation* of $P$ is a *plane* graph on $P$, such that each face is a triangle.

There are many different triangulations of $P$.



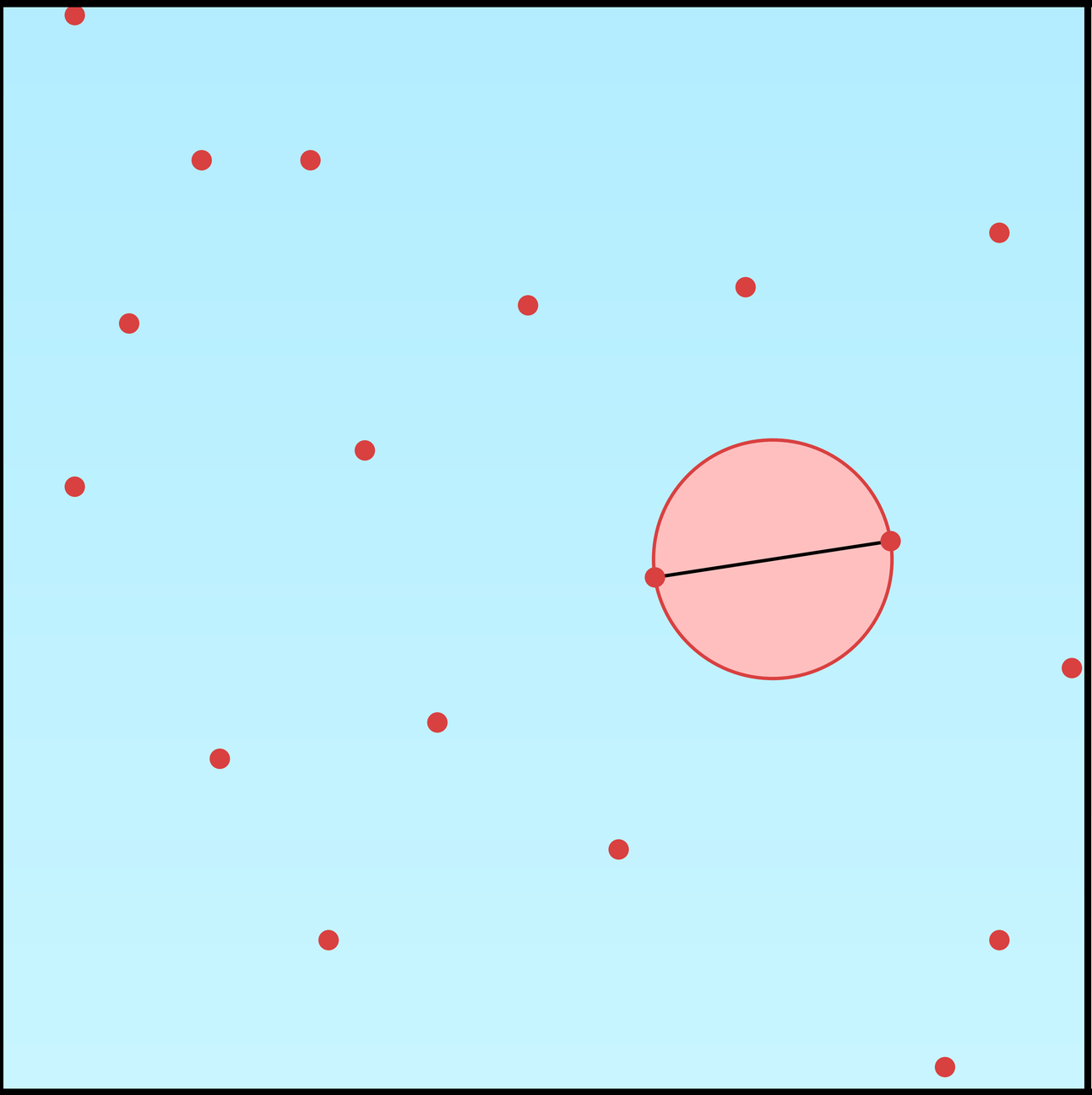
The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

[Delaunay, 1934]



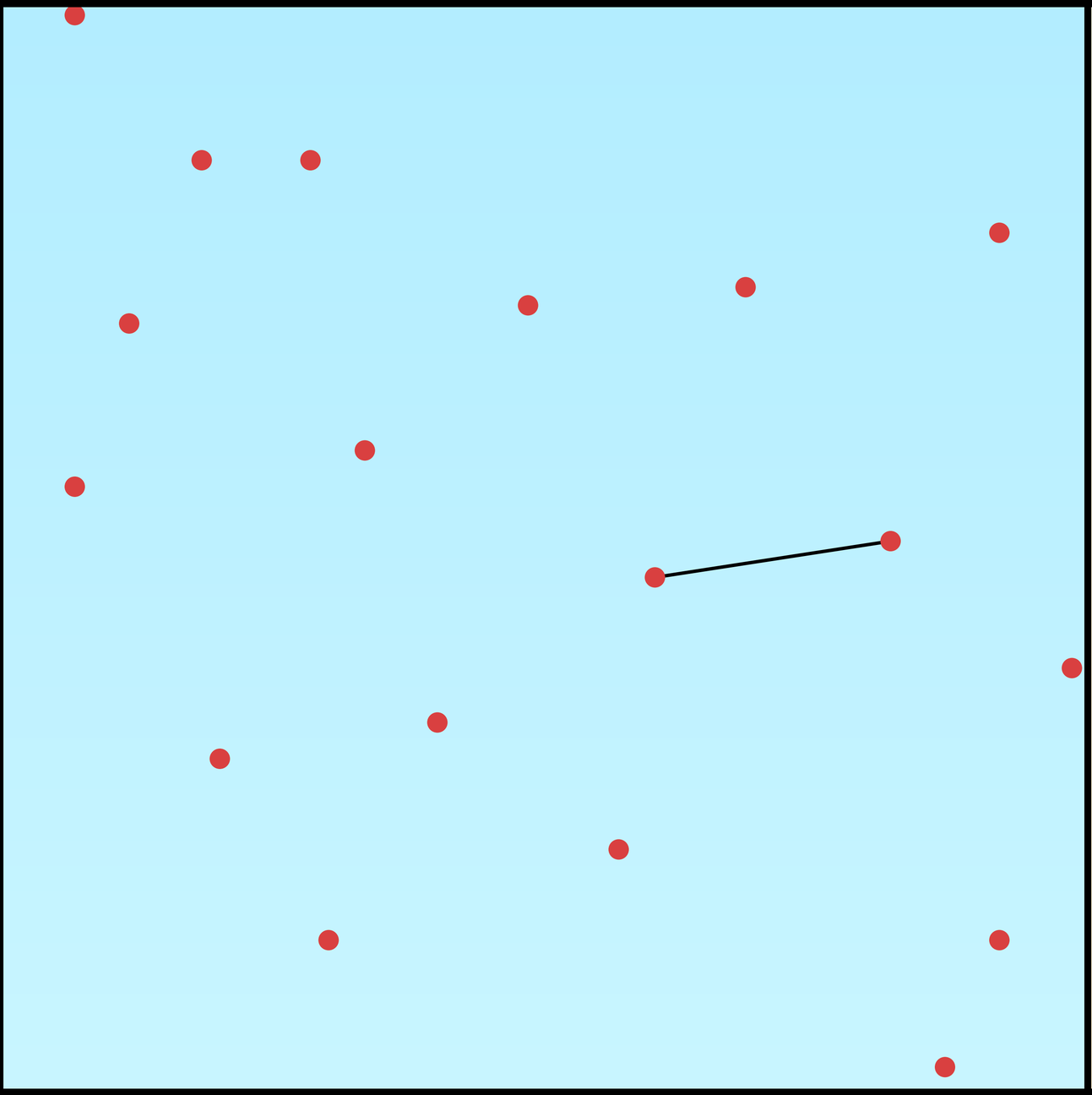
The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

[Delaunay, 1934]



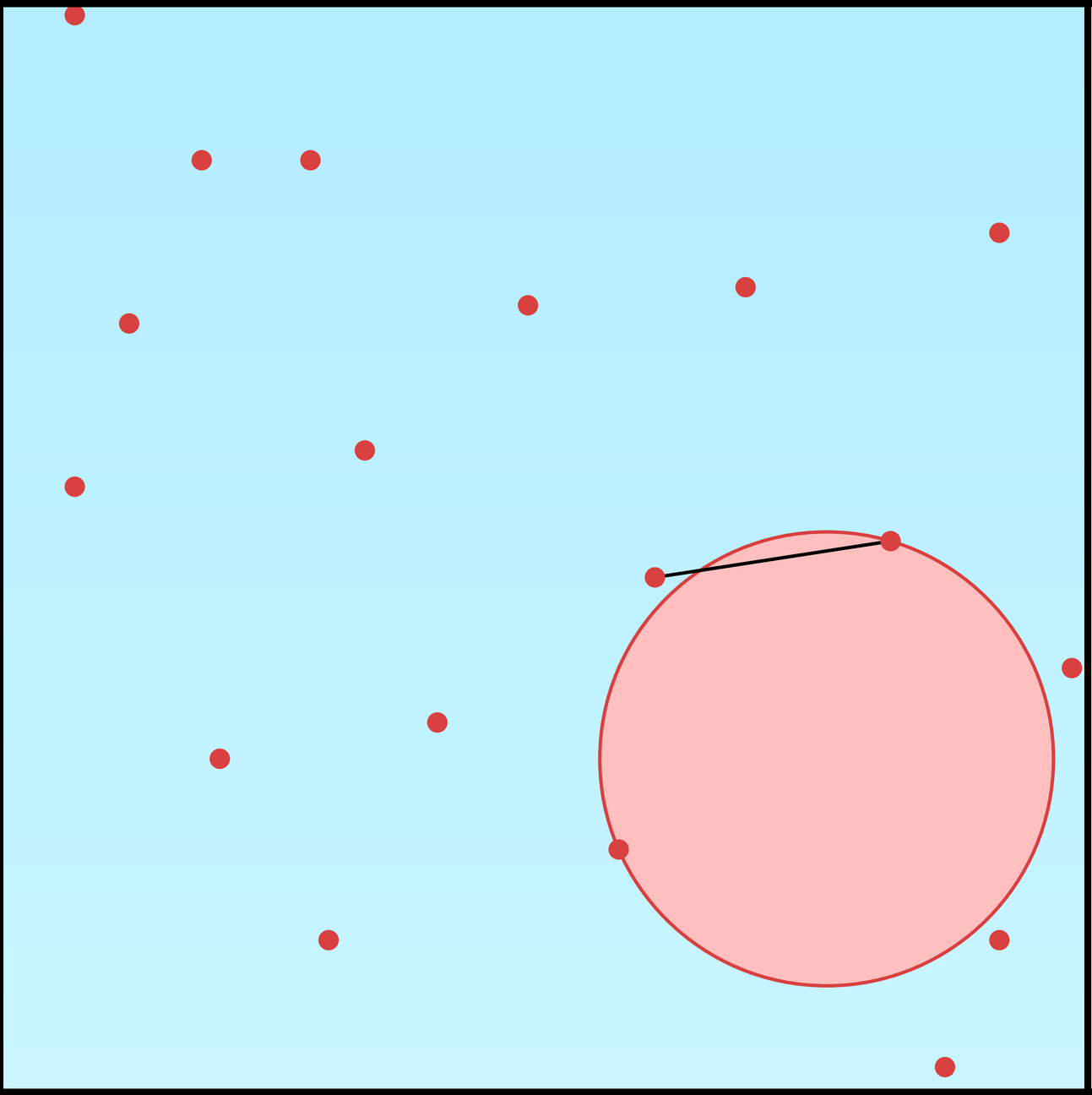
The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

[Delaunay, 1934]



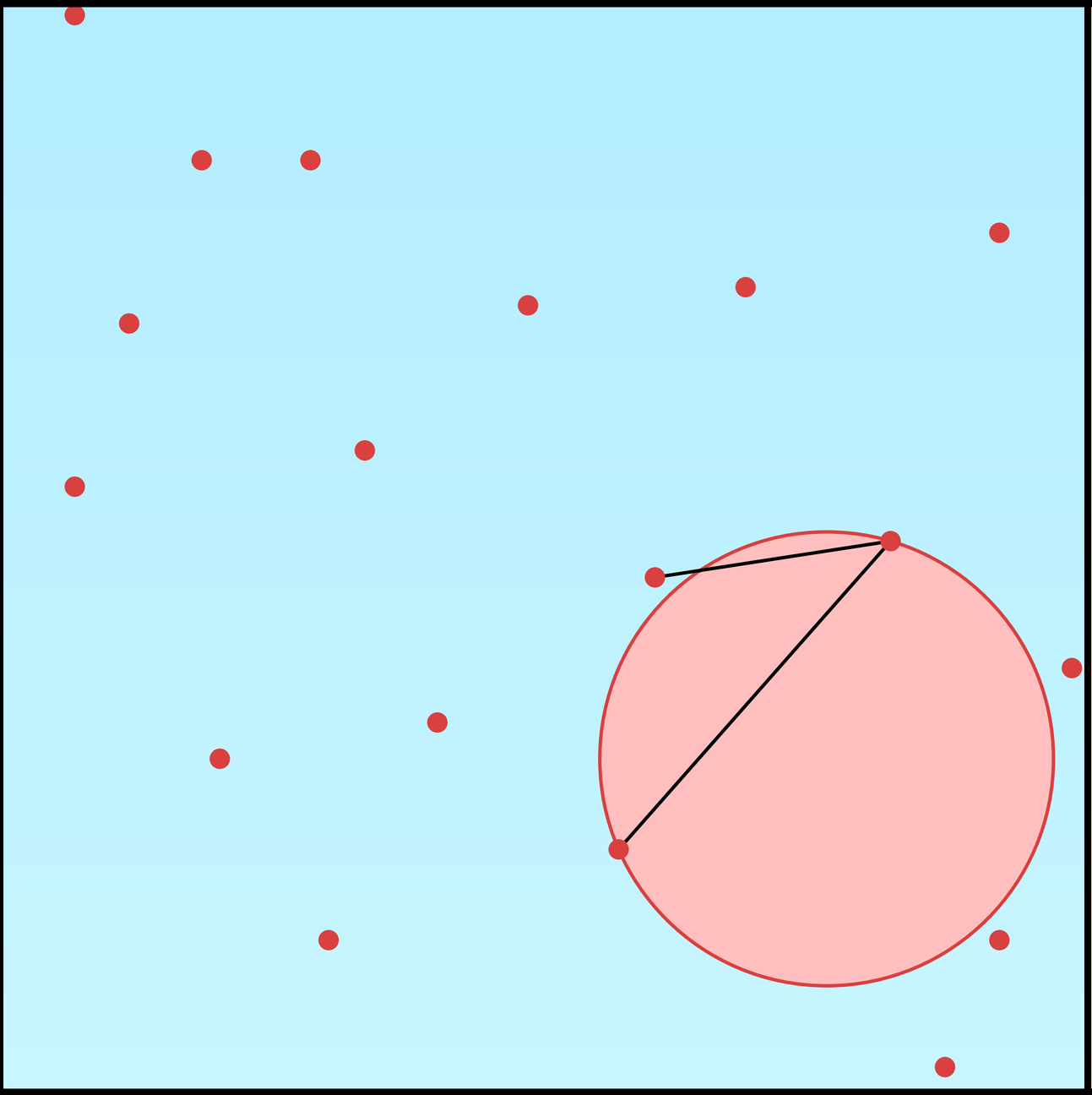
The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

[Delaunay, 1934]



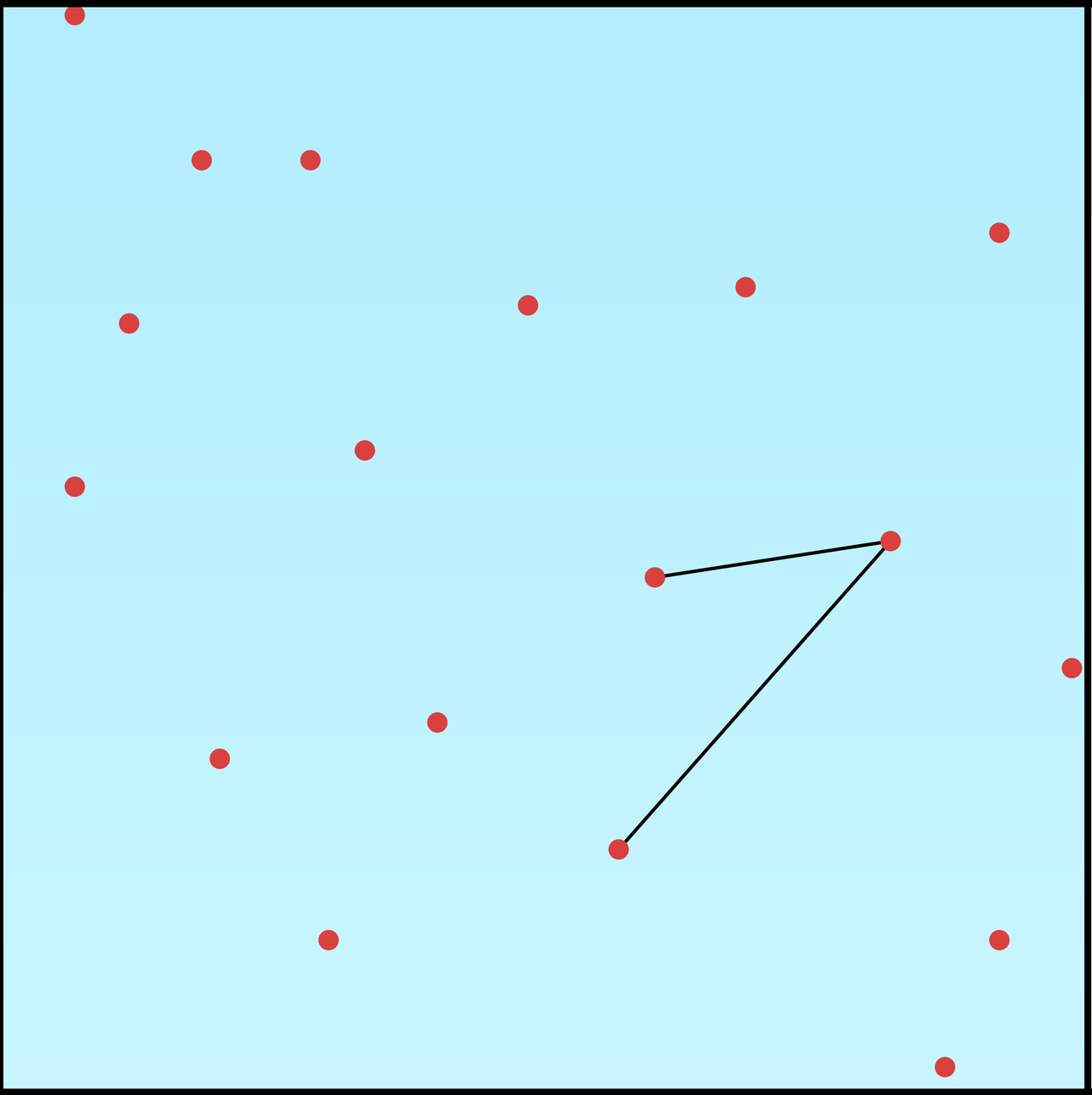
The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

[Delaunay, 1934]



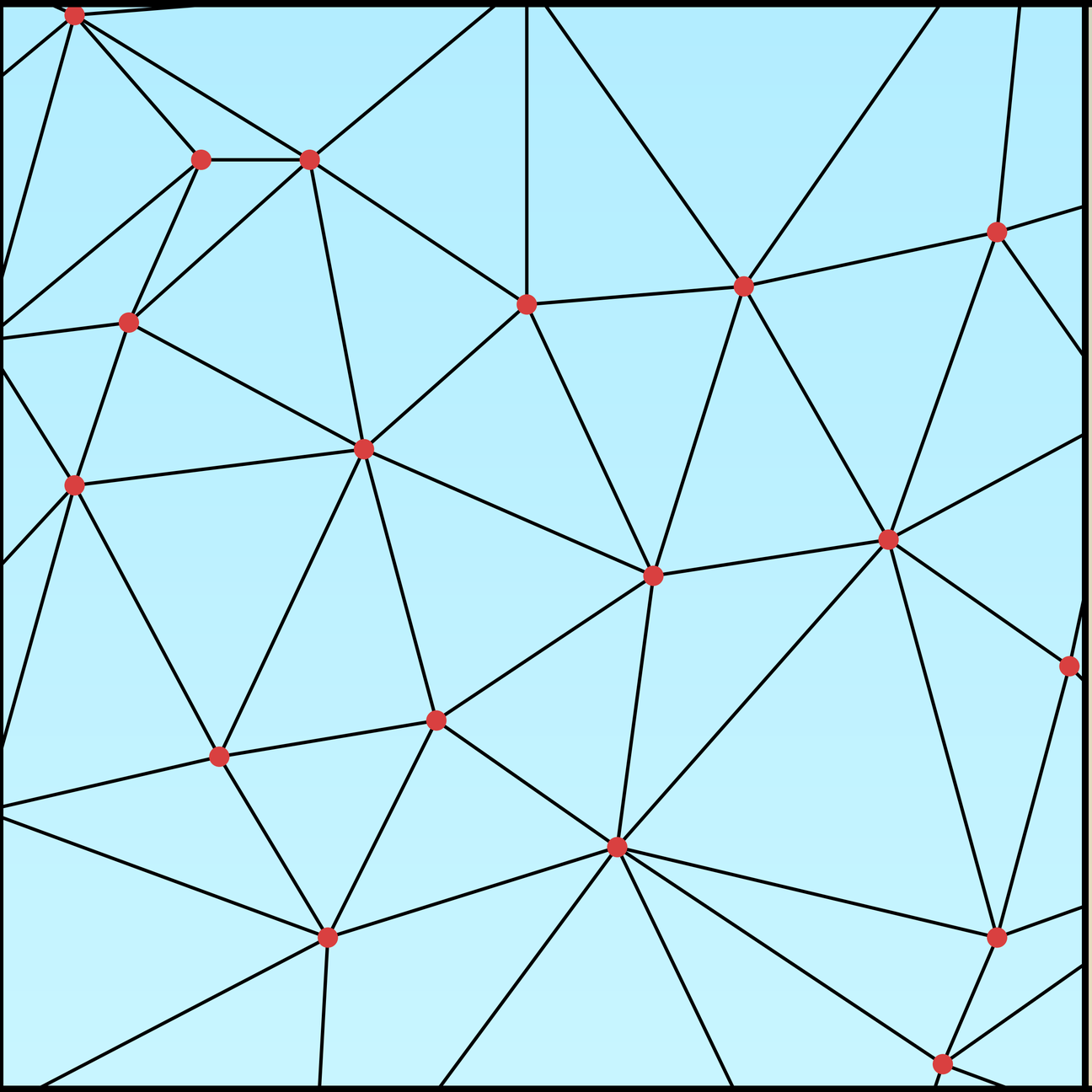
The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

[Delaunay, 1934]



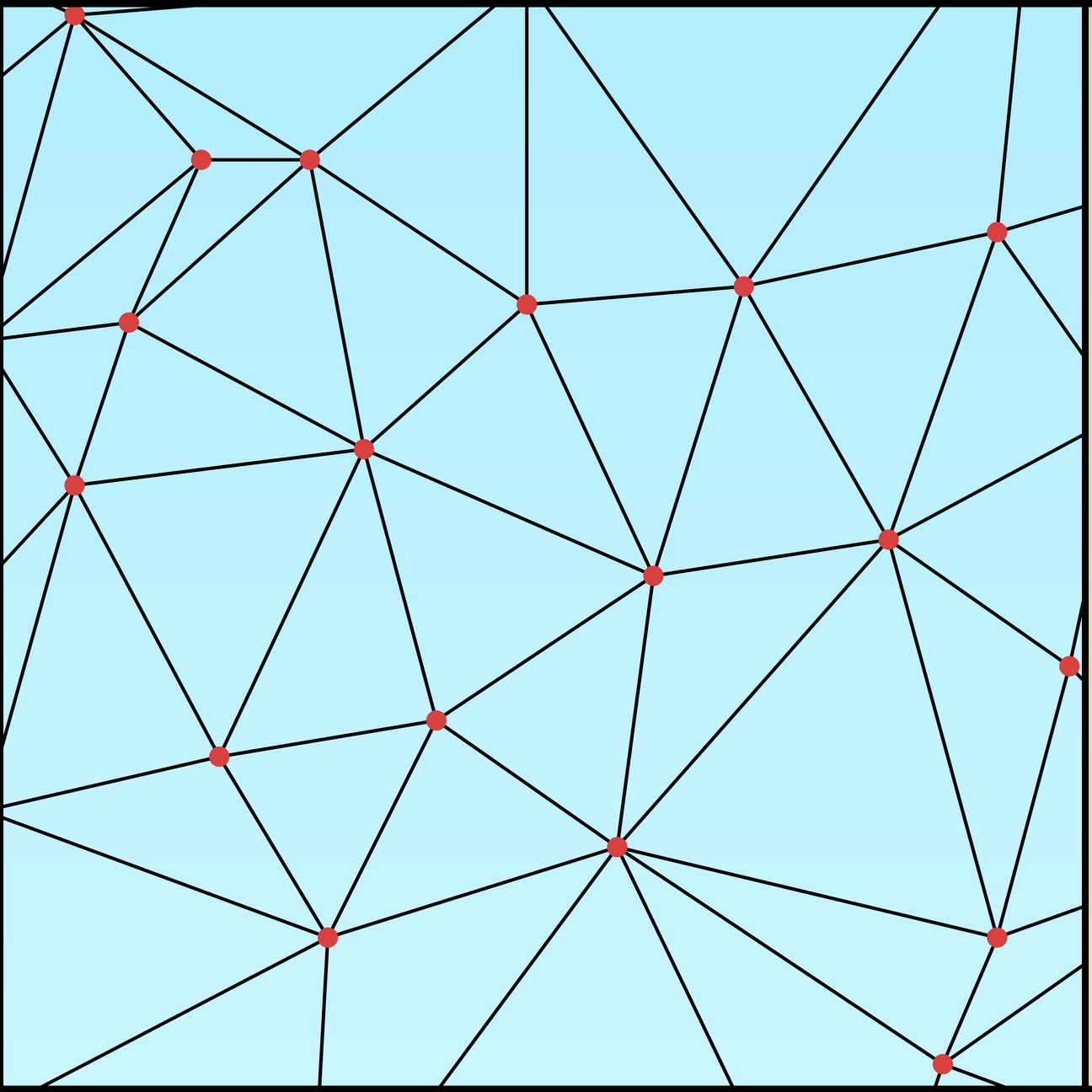
The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

[Delaunay, 1934]



The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

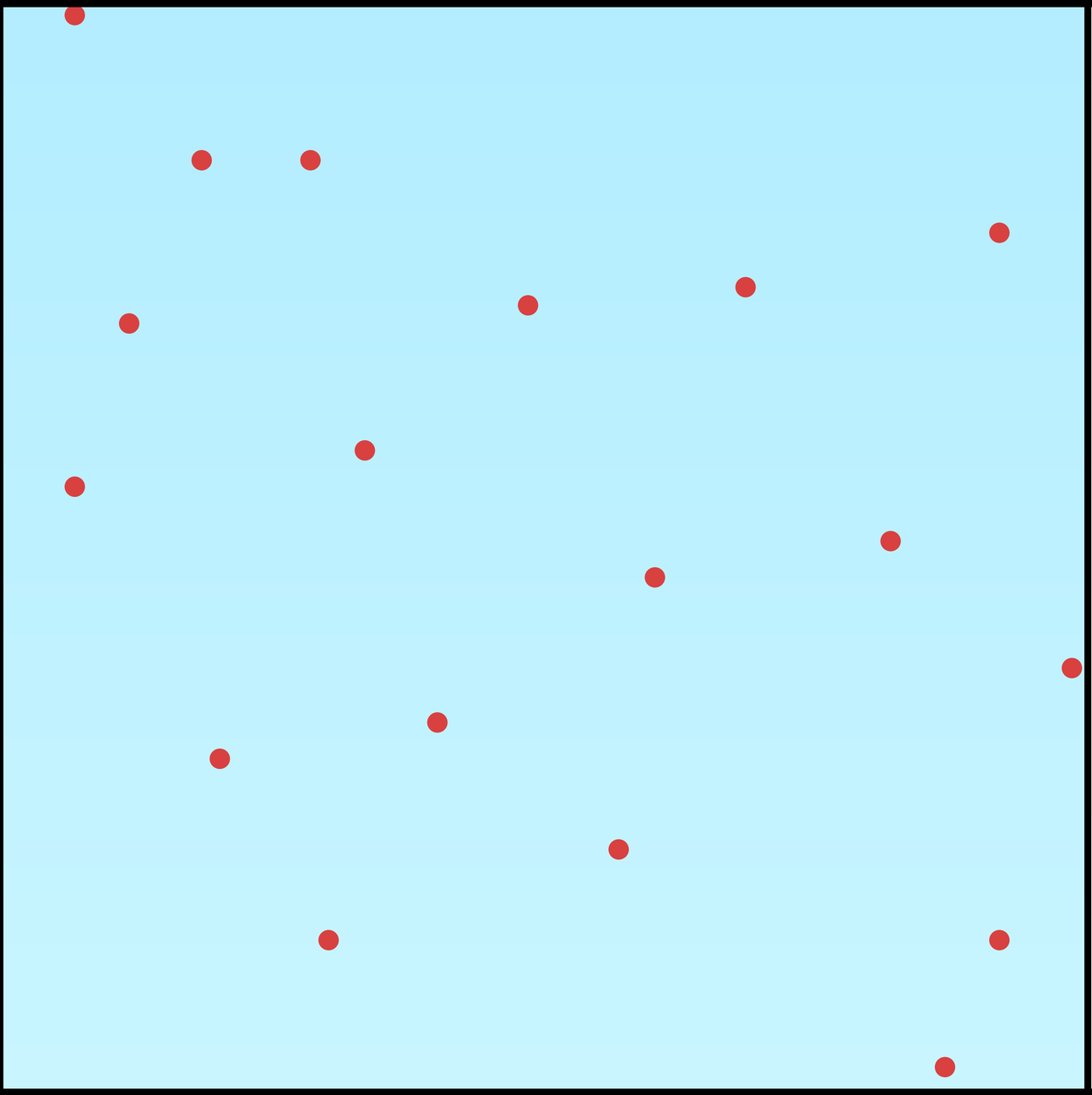
[Delaunay, 1934]



The *Delaunay triangulation (DT)* is a specific triangulation of $P$ that has an edge between two points if there is an empty circle through these points.

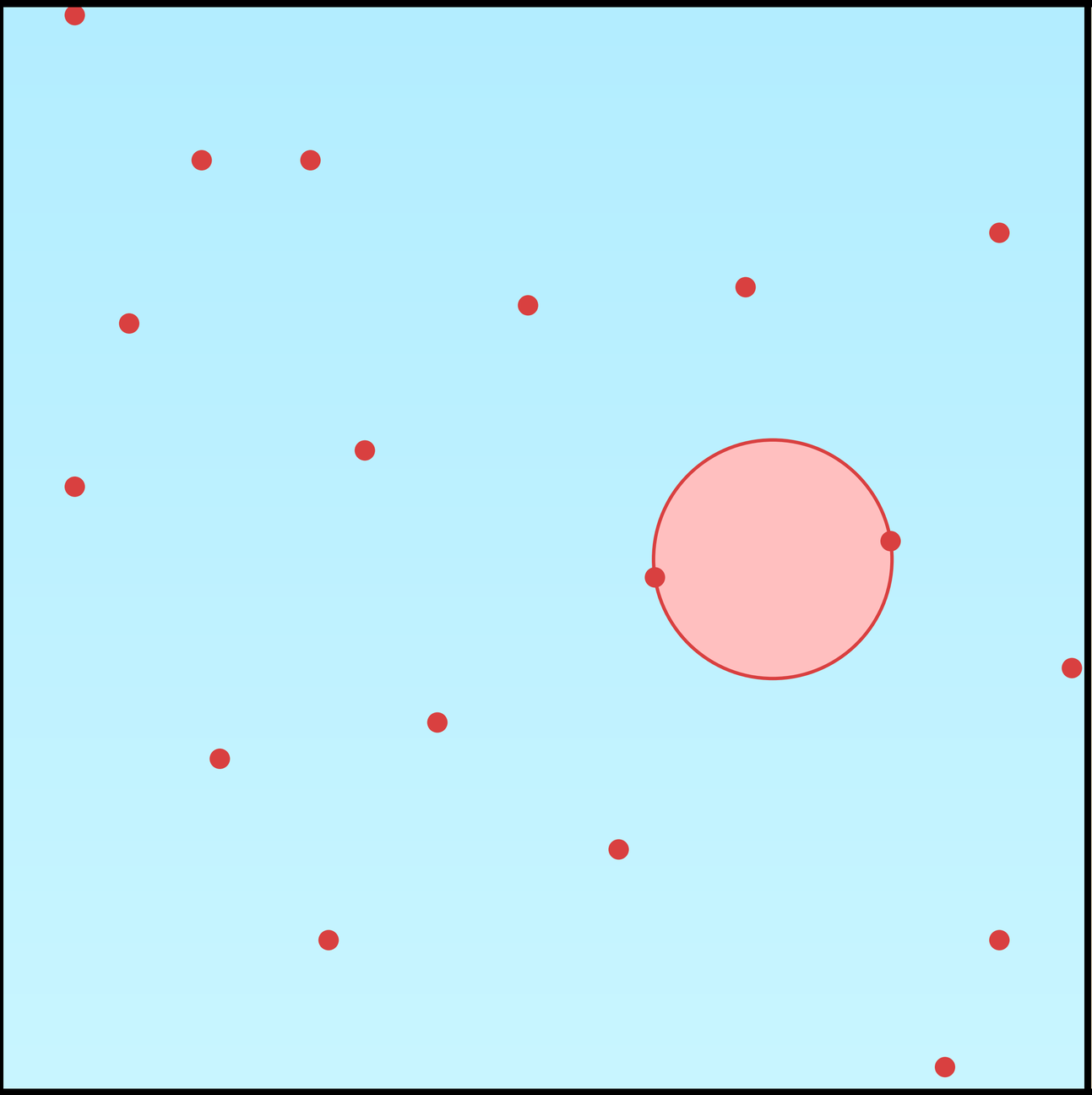
If $P$ is in *general position*, the Delaunay triangulation is uniquely defined.

[Delaunay, 1934]



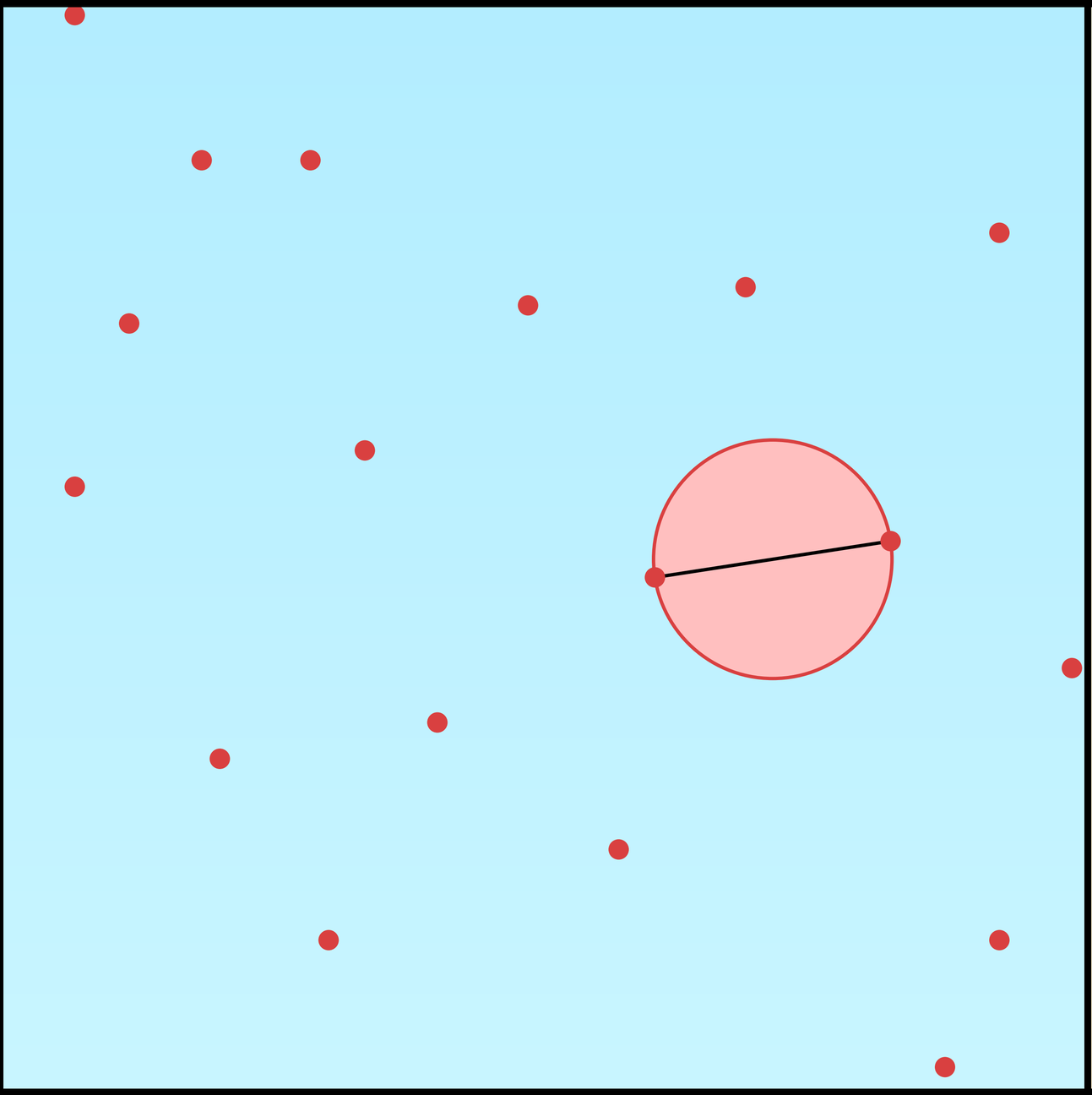
The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

[Gabriel, 1969]



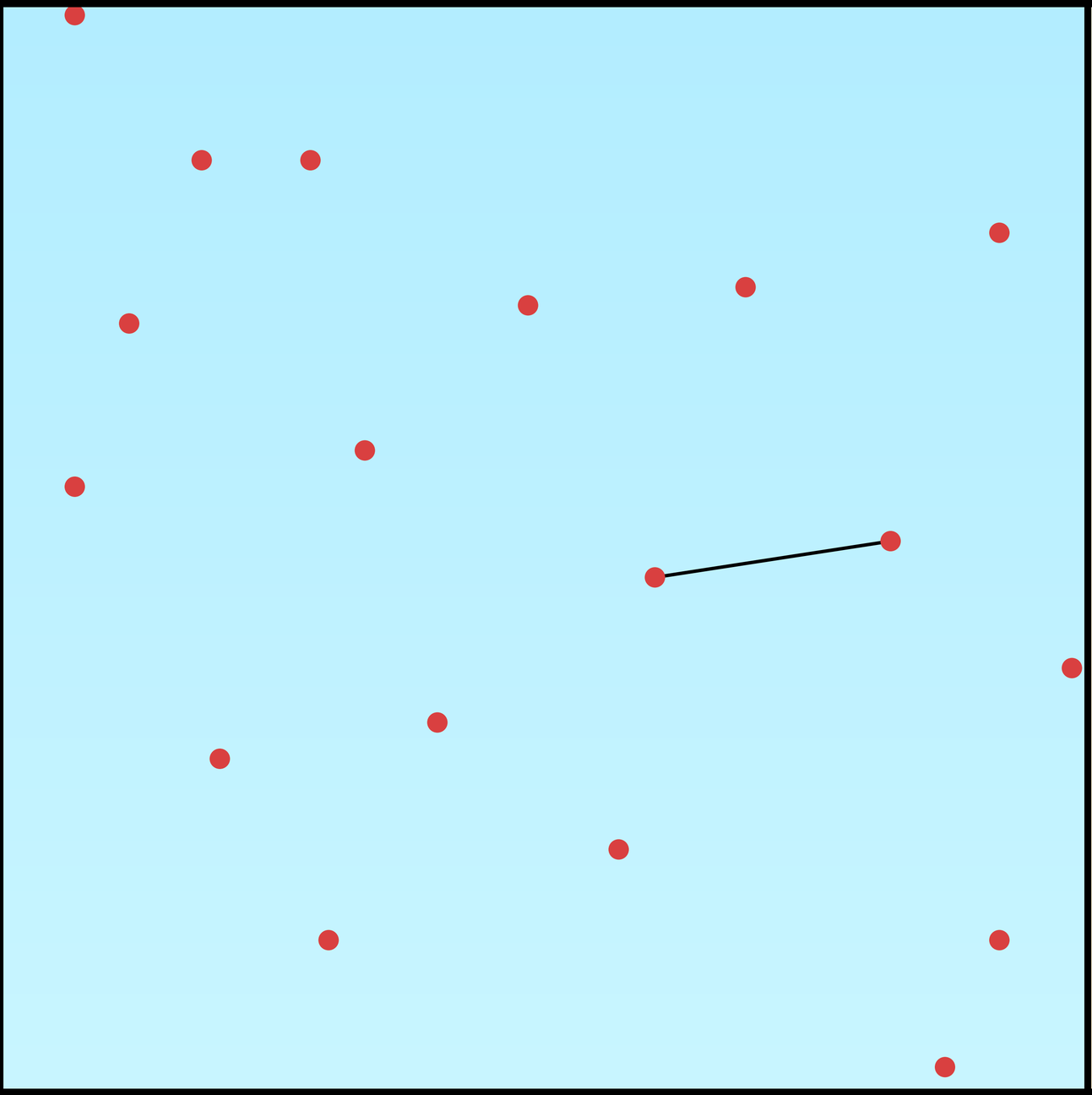
The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

[Gabriel, 1969]



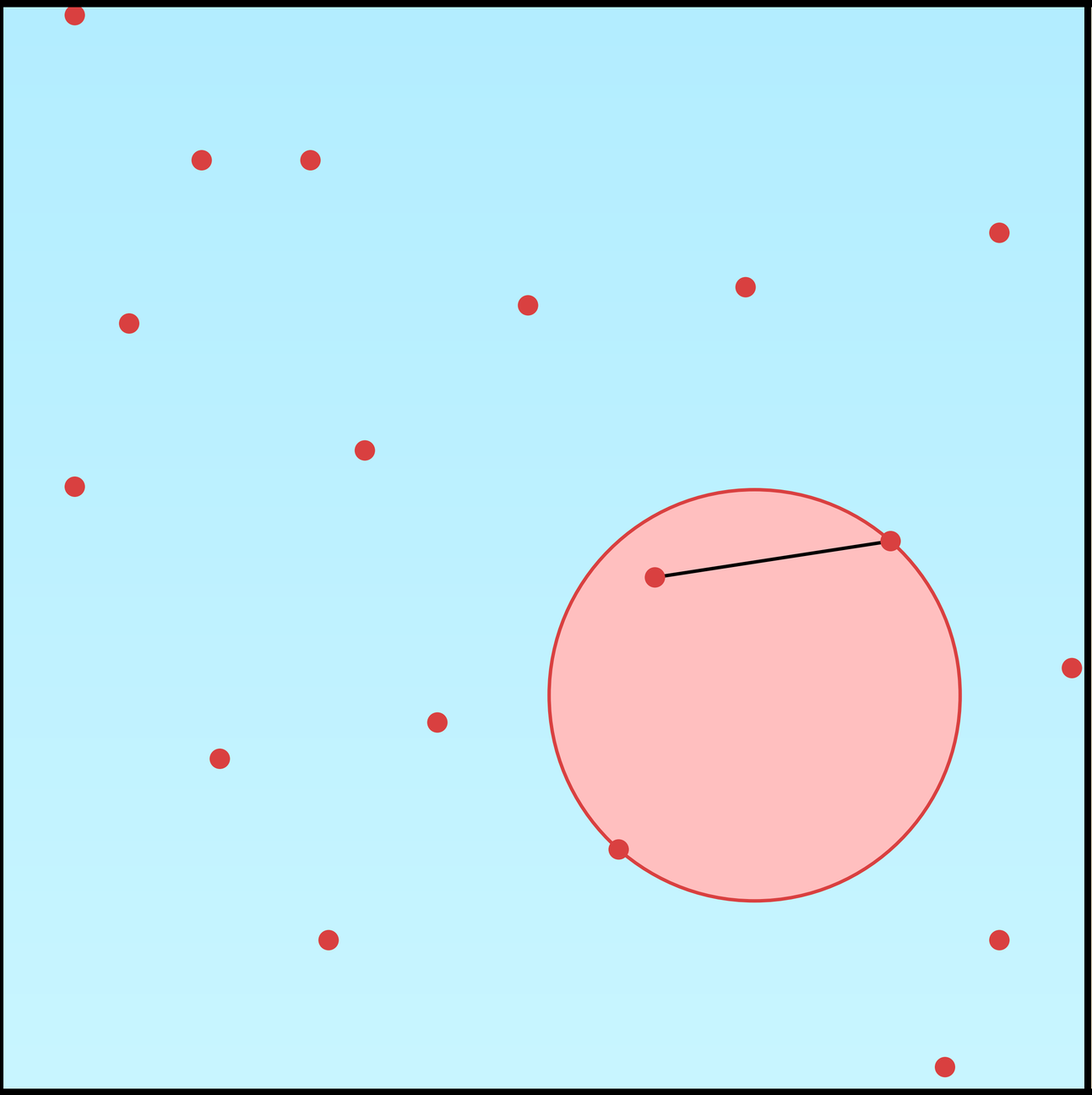
The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

[Gabriel, 1969]



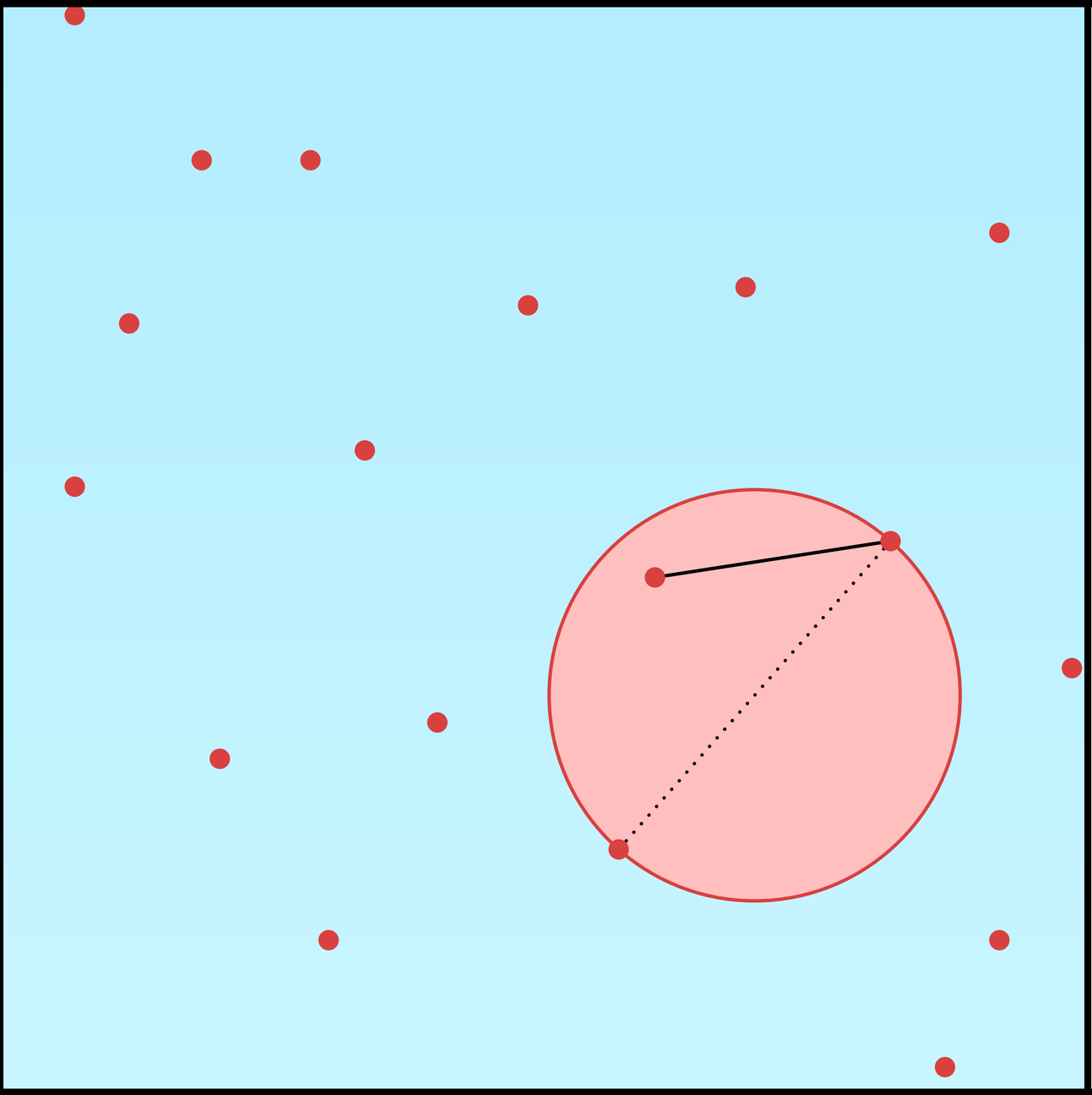
The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

[Gabriel, 1969]



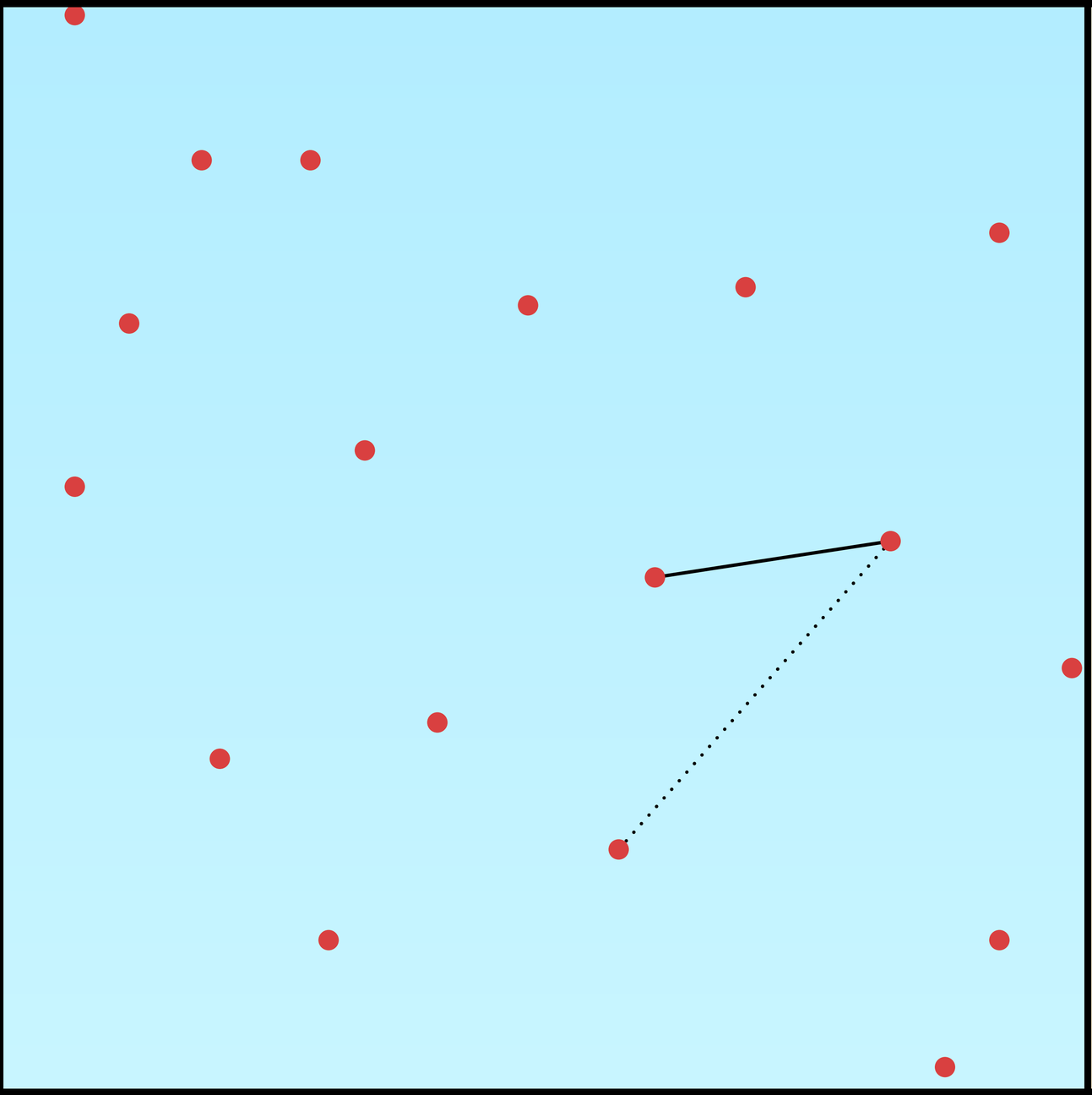
The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

[Gabriel, 1969]



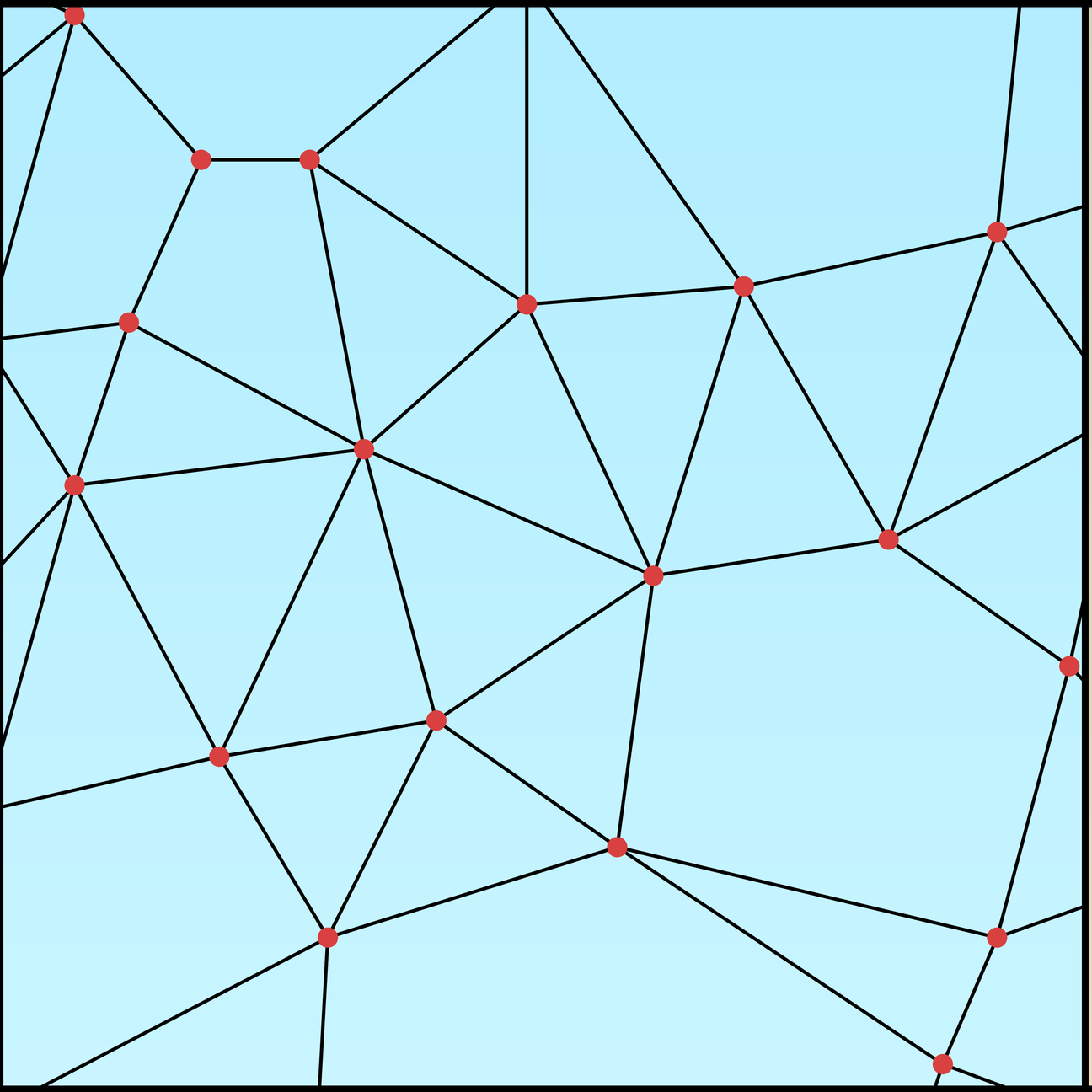
The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

[Gabriel, 1969]



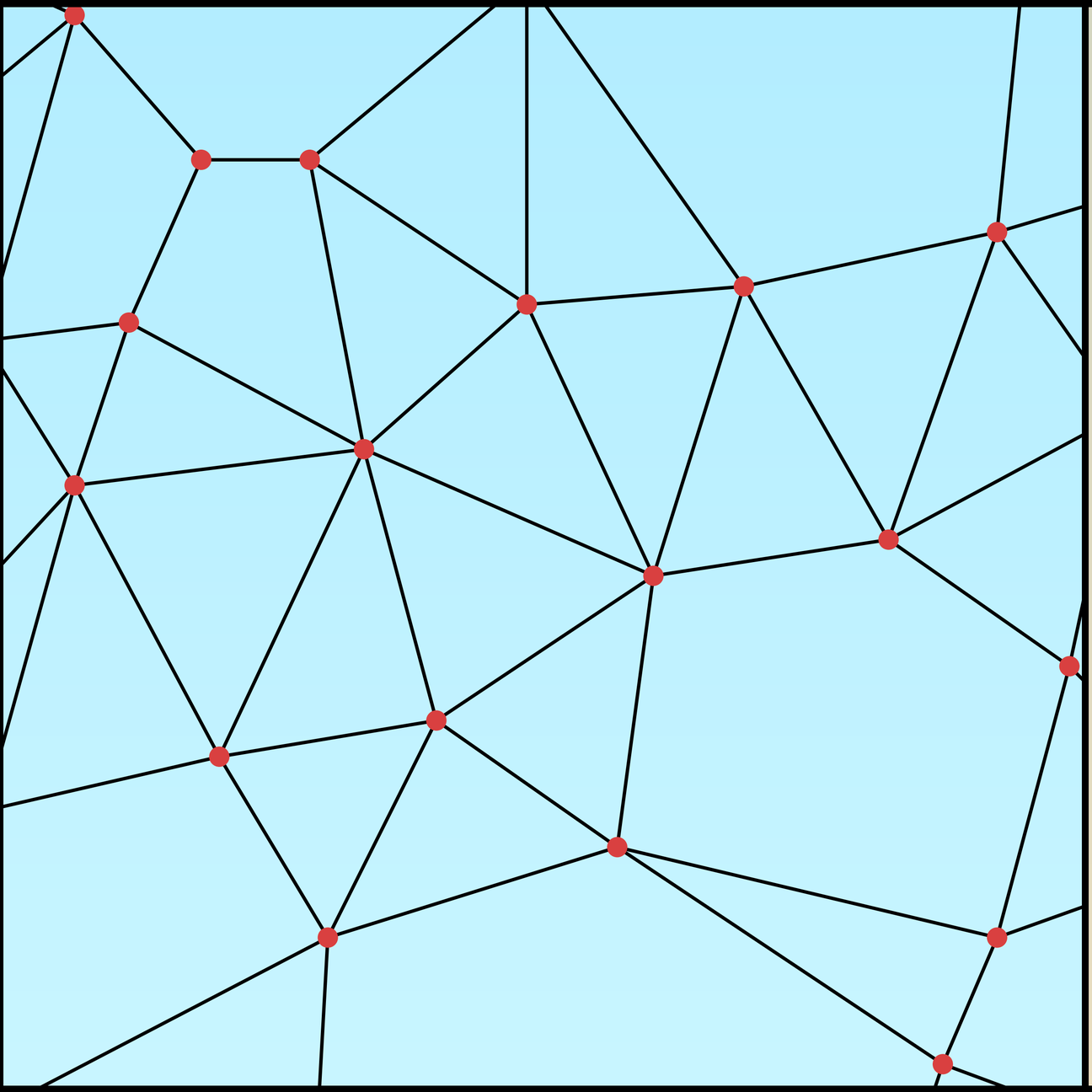
The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

[Gabriel, 1969]



The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

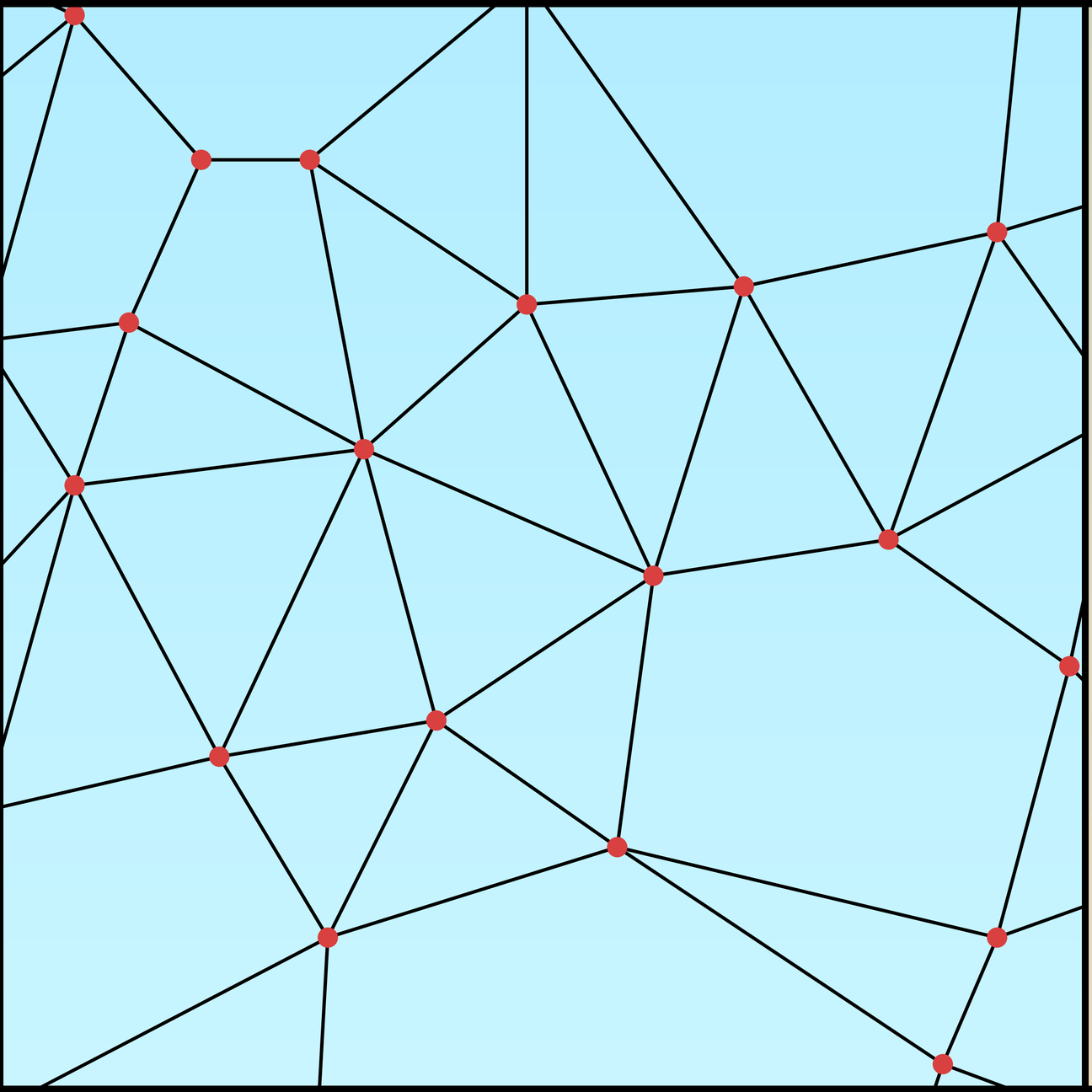
[Gabriel, 1969]



The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

The Gabriel graph is a *subgraph* of the Delaunay triangulation.

[Gabriel, 1969]

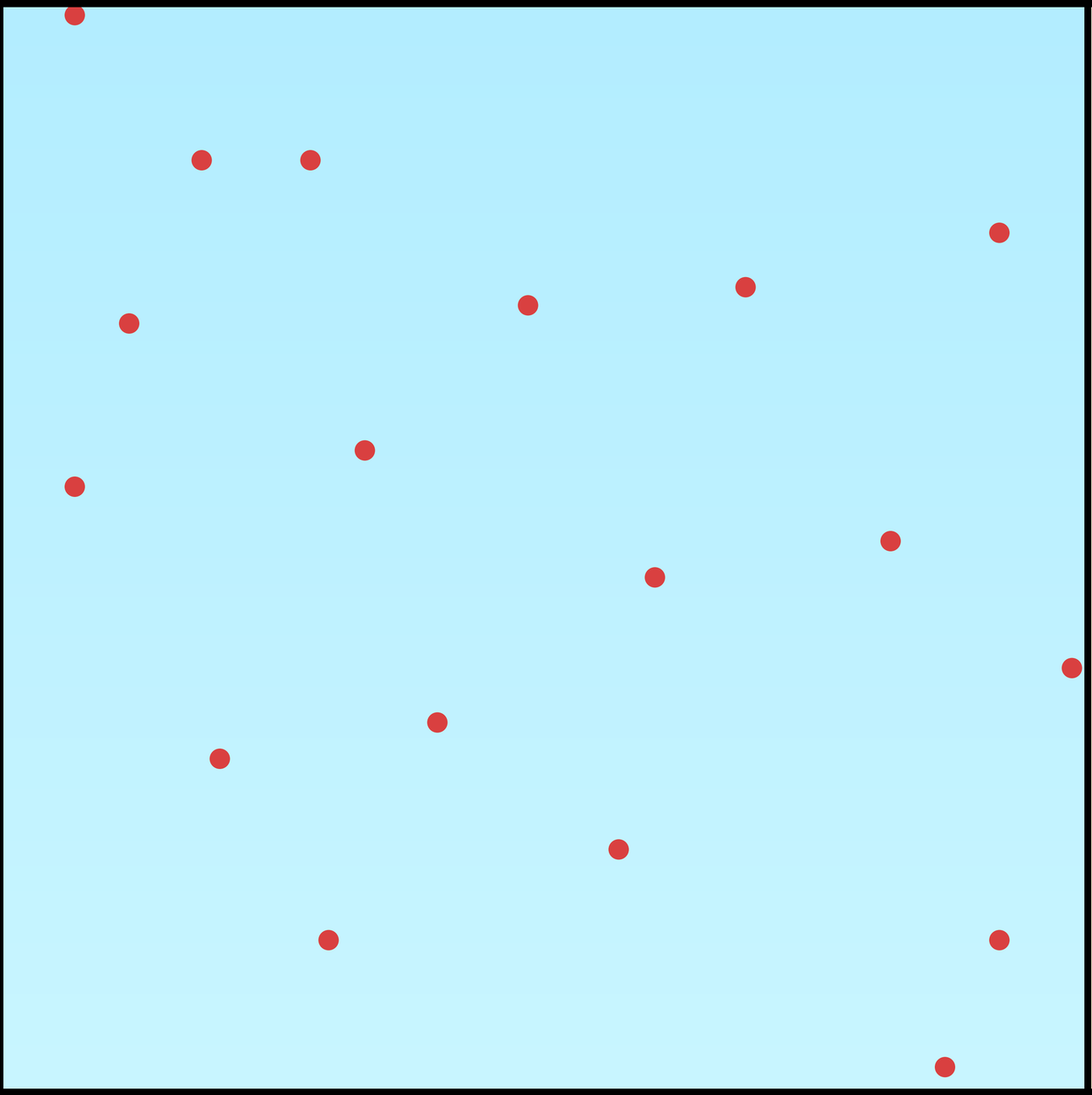


The *Gabriel graph* is the graph that has an edge between two points of $P$ if the *diametrical* circle is empty.

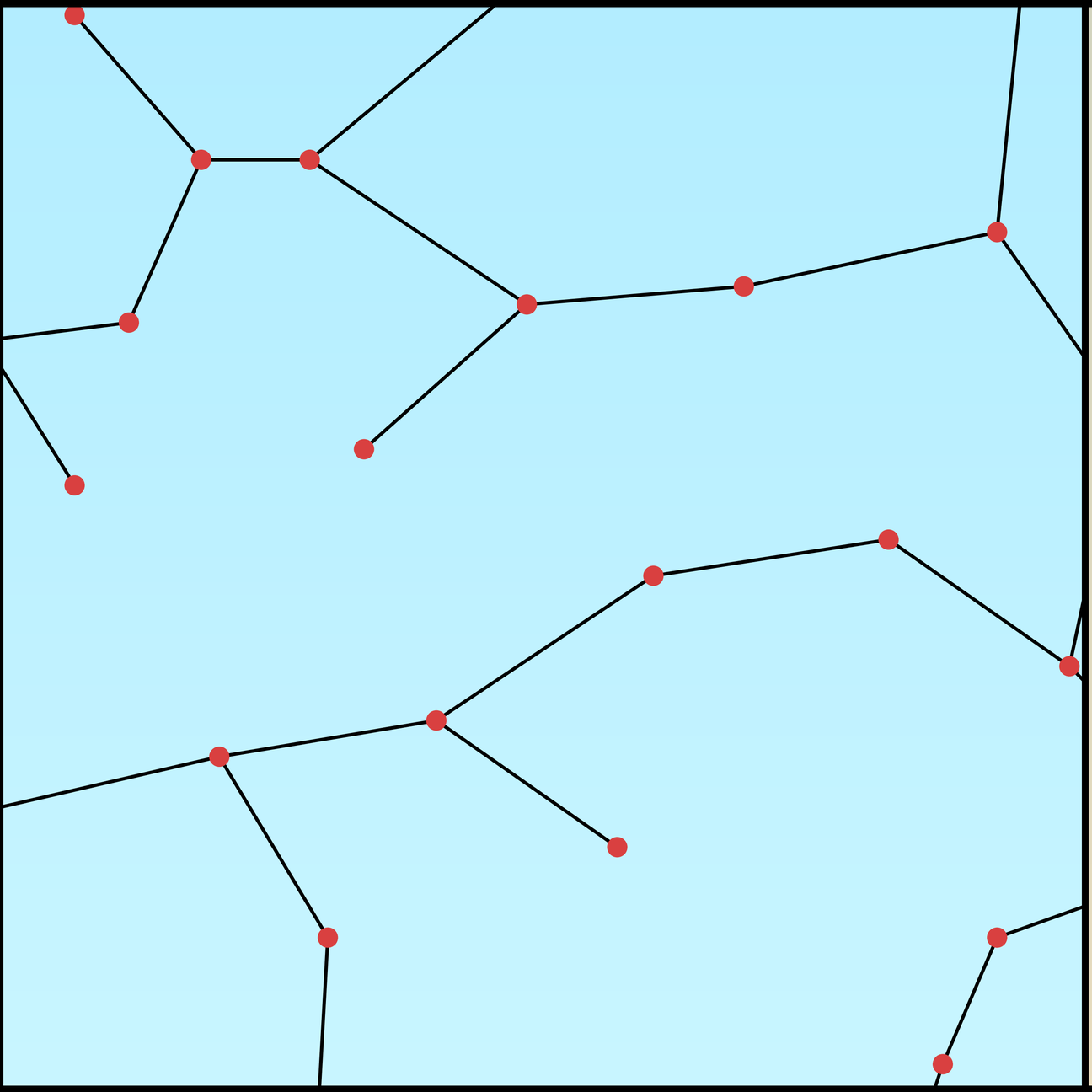
The Gabriel graph is a *subgraph* of the Delaunay triangulation.

The *Gabriel graph* is always connected!

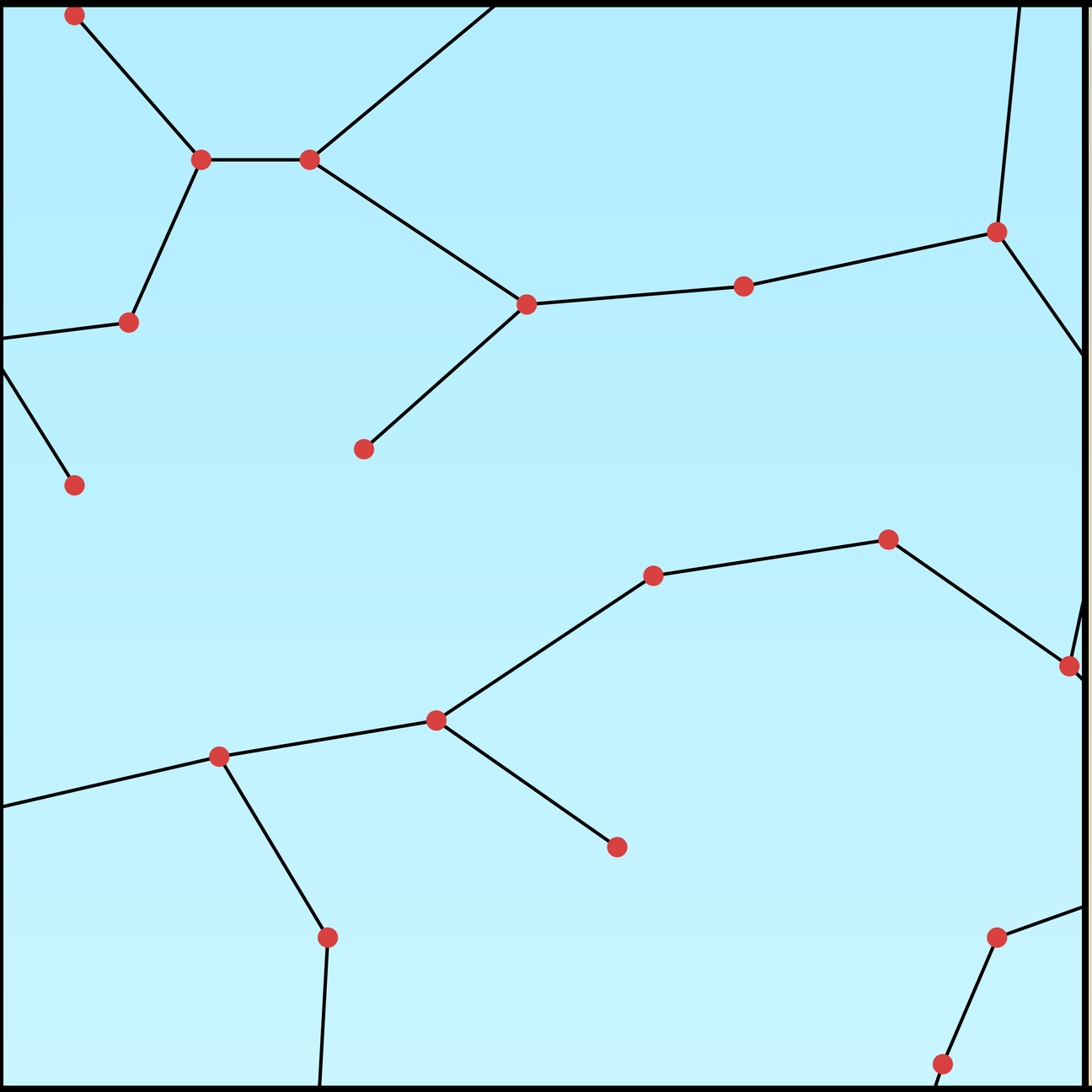
[Gabriel, 1969]



The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

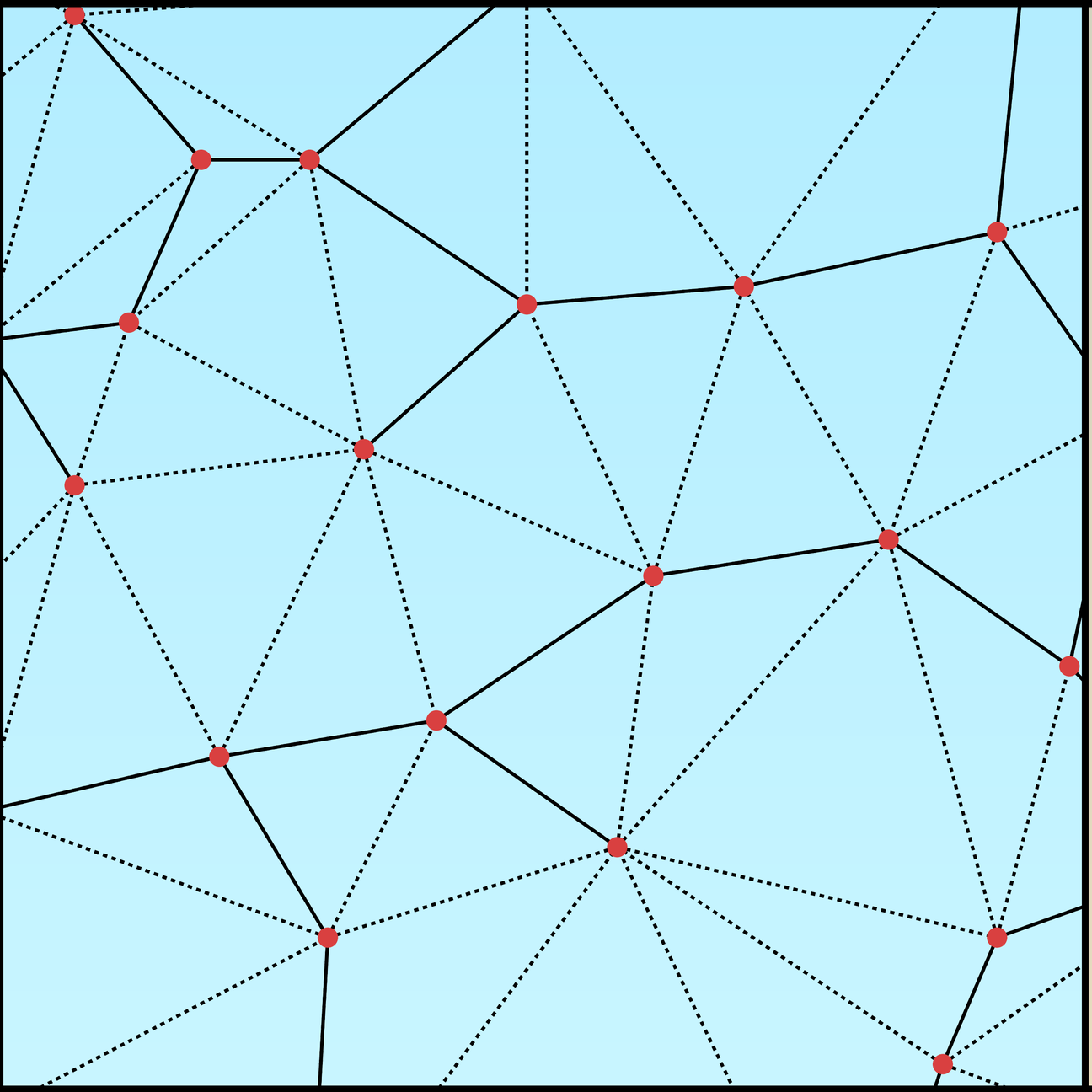


The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.



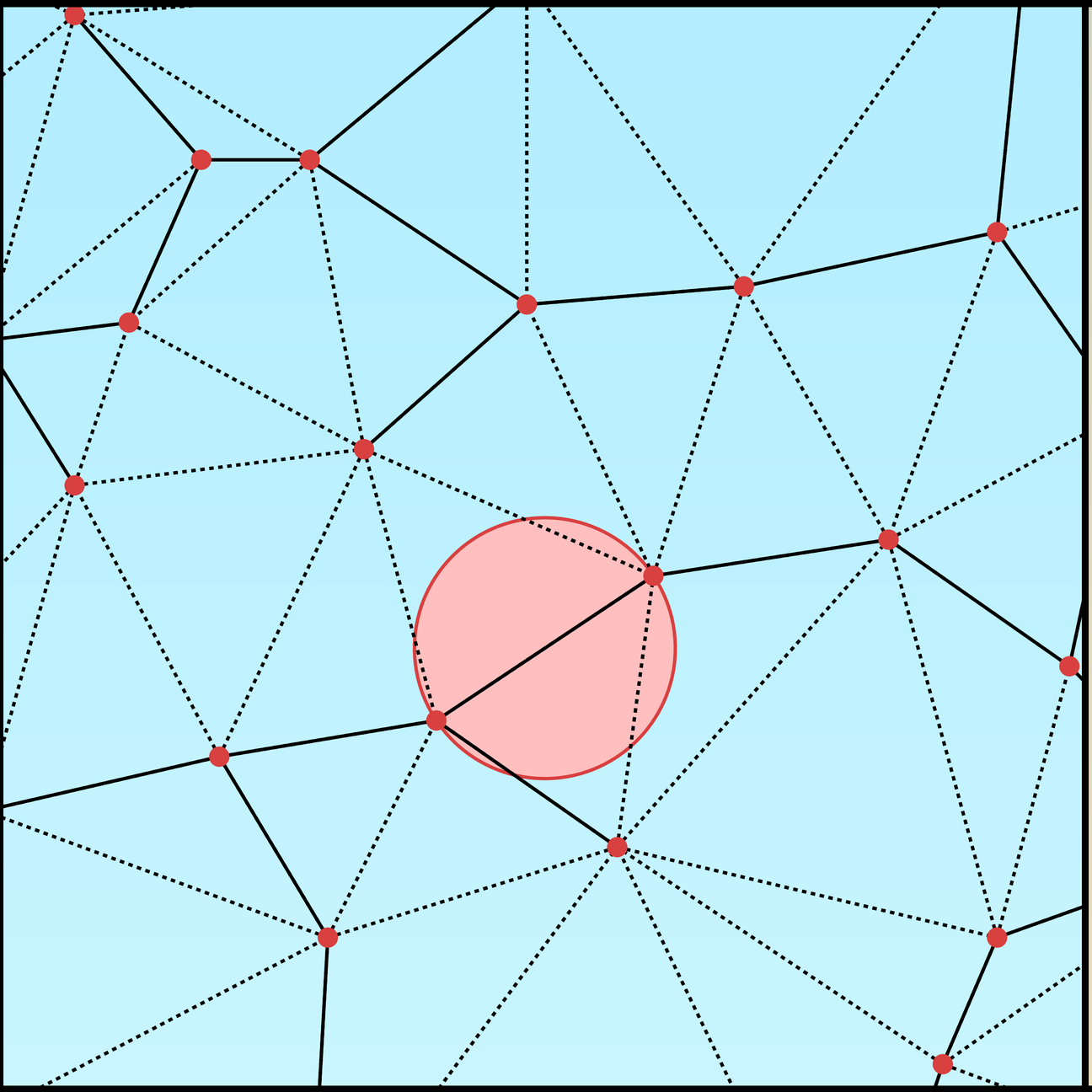
The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

The EMST is a subgraph of the Gabriel graph.



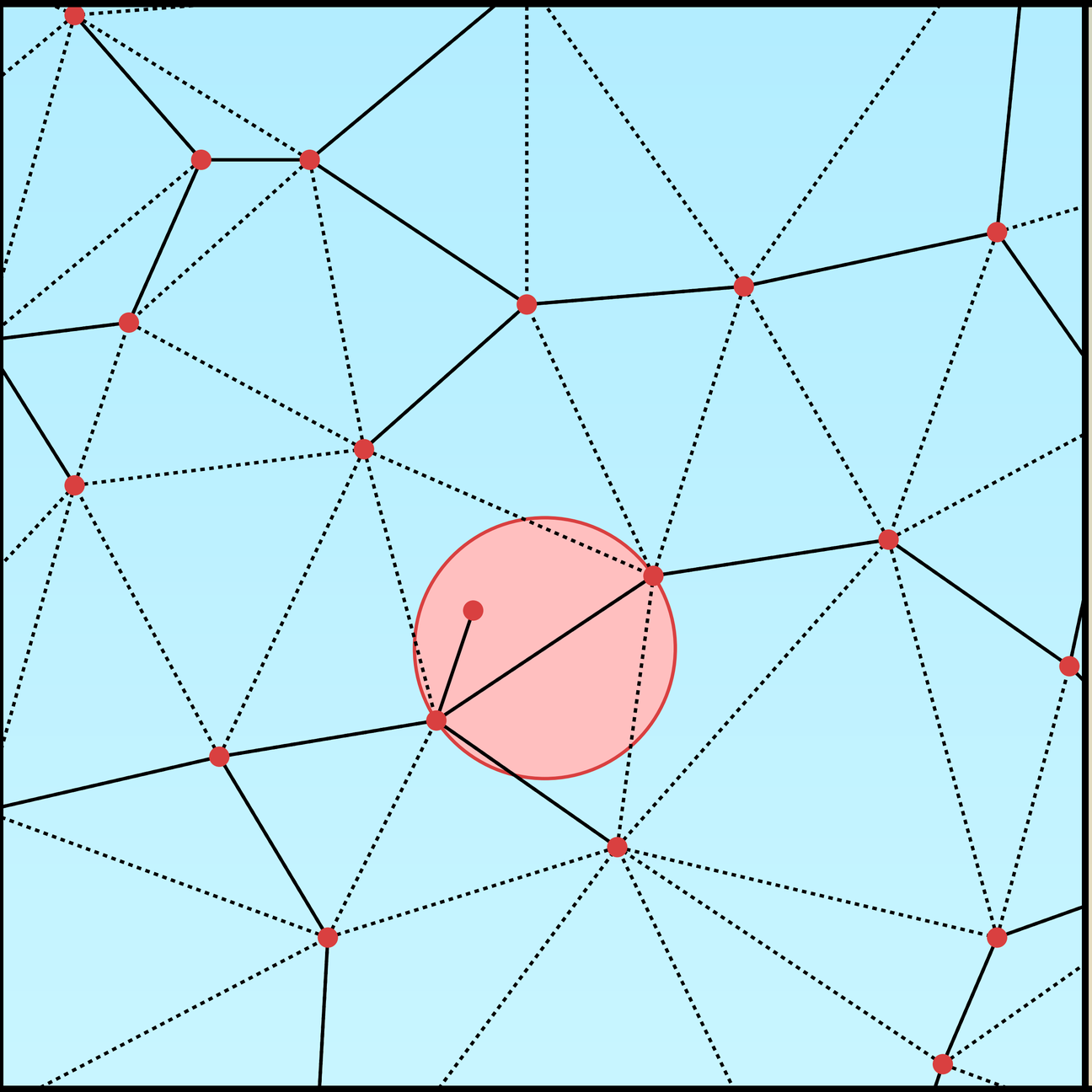
The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

The EMST is a subgraph of the Gabriel graph.



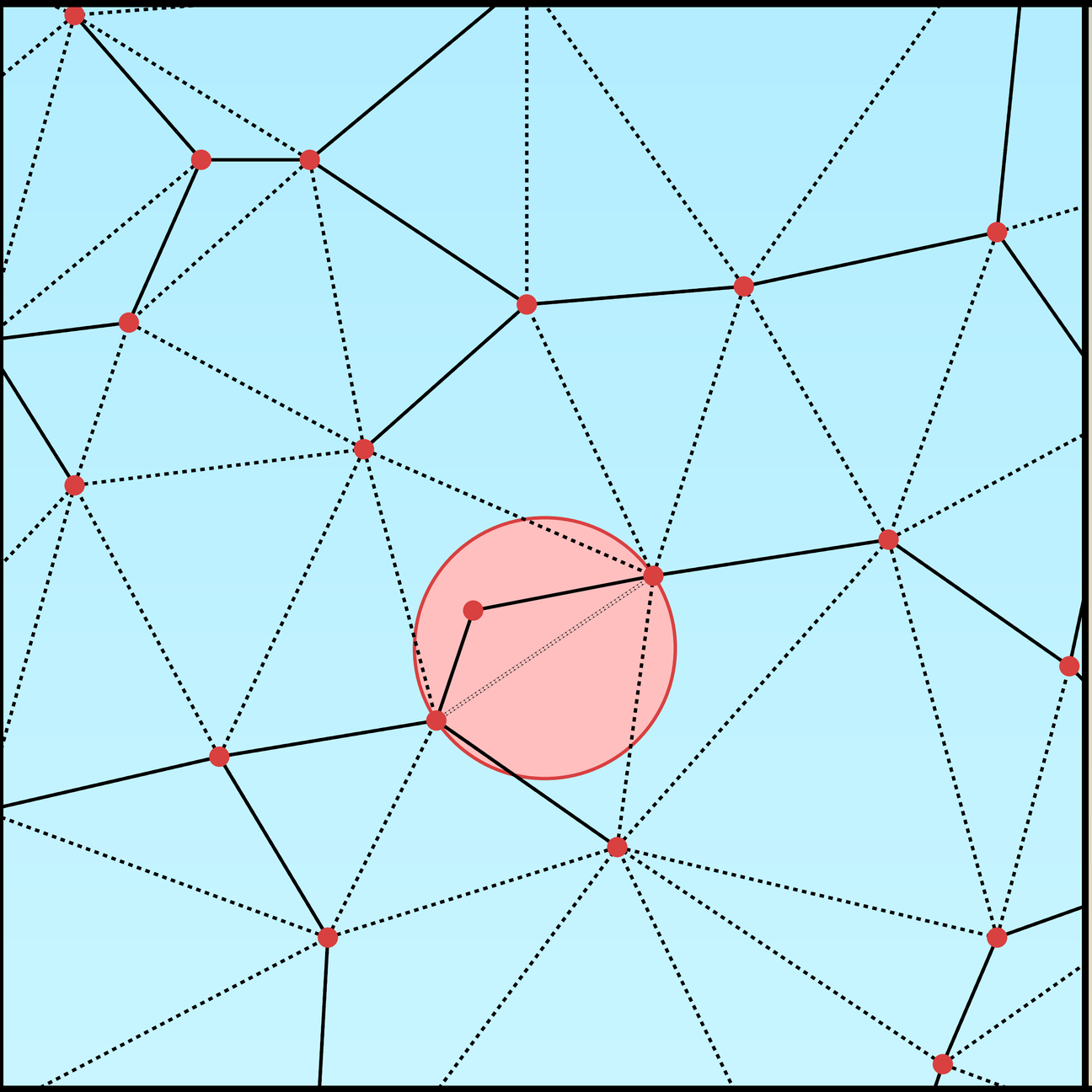
The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

The EMST is a subgraph of the Gabriel graph.



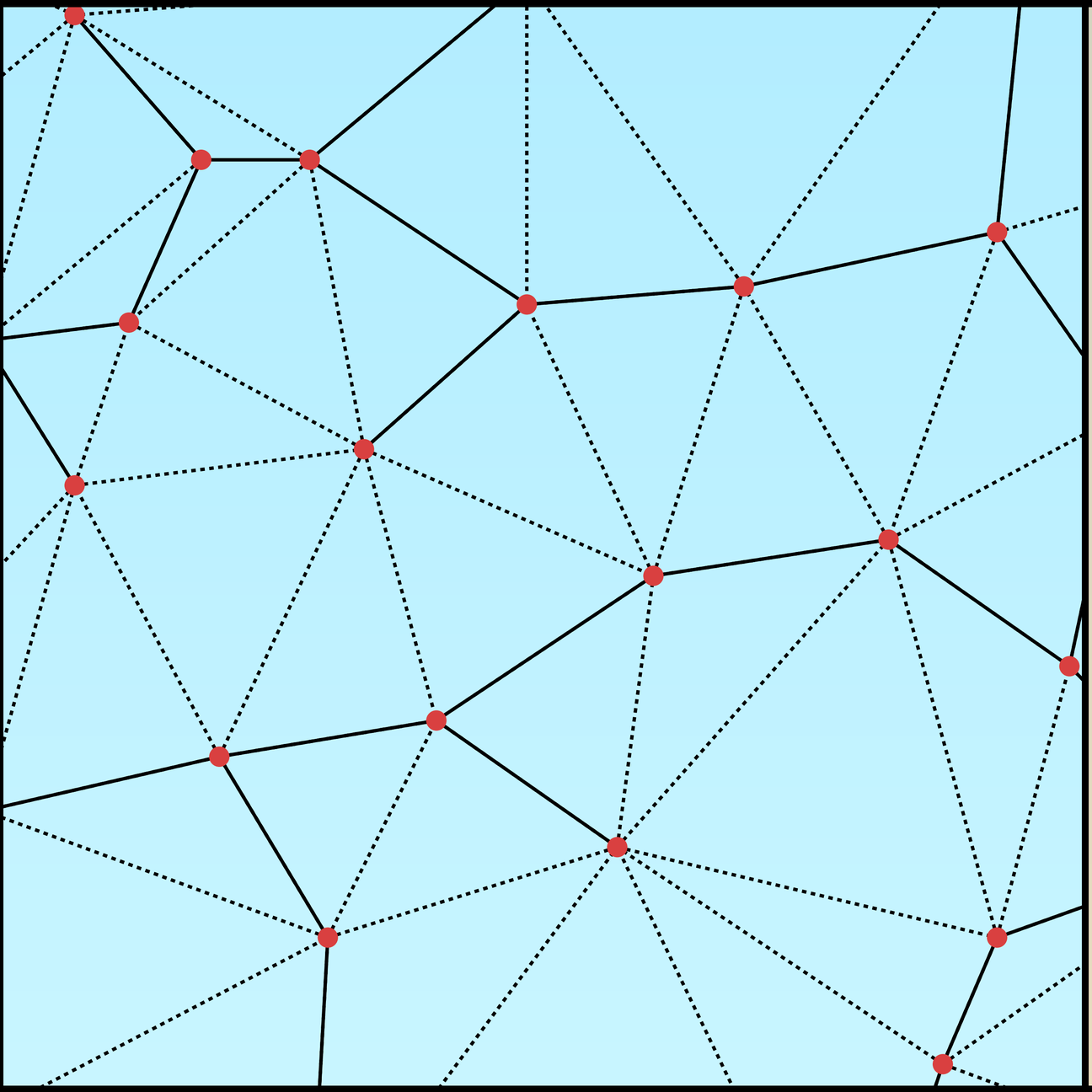
The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

The EMST is a subgraph of the Gabriel graph.



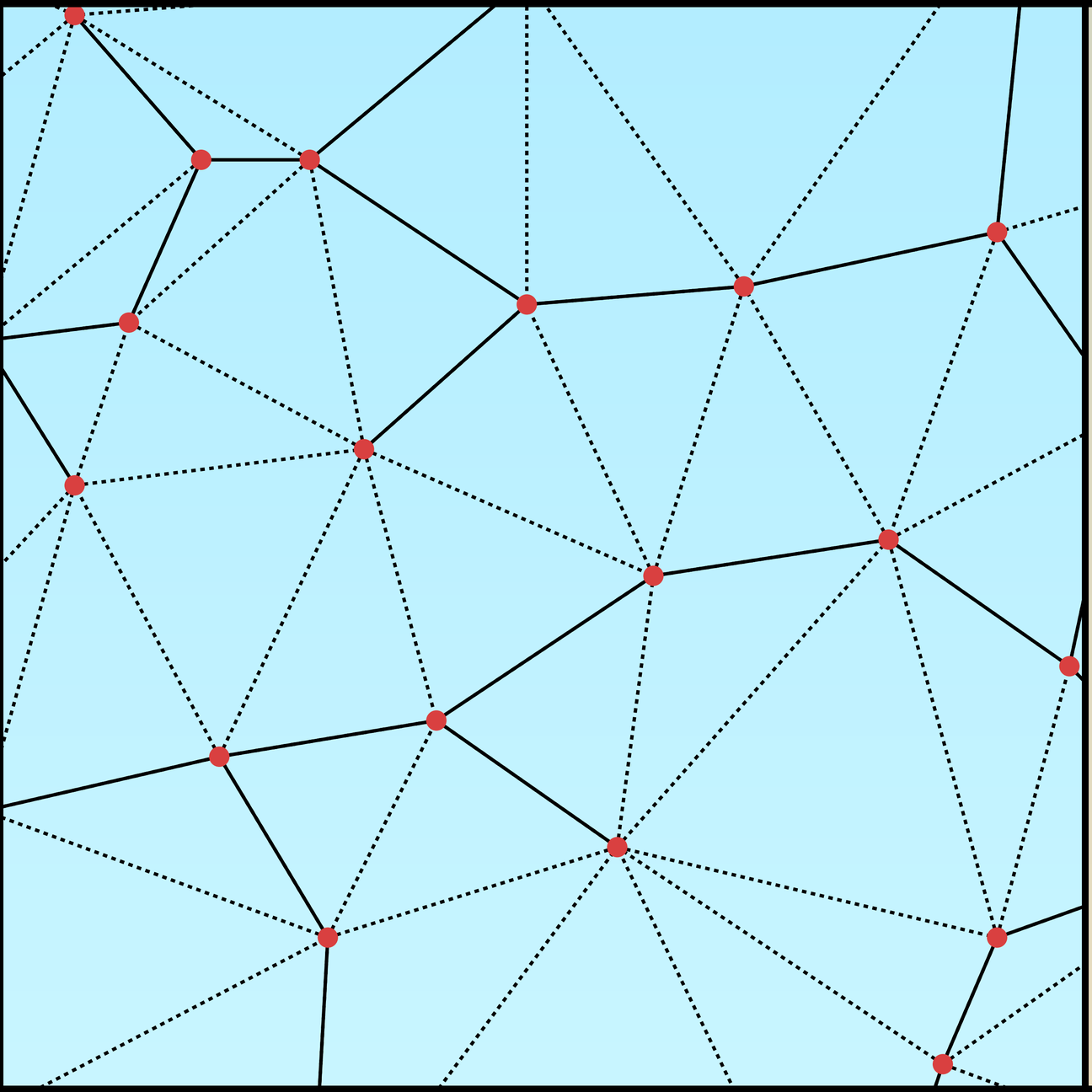
The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

The EMST is a subgraph of the Gabriel graph.



The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

The EMST is a subgraph of the Gabriel graph.

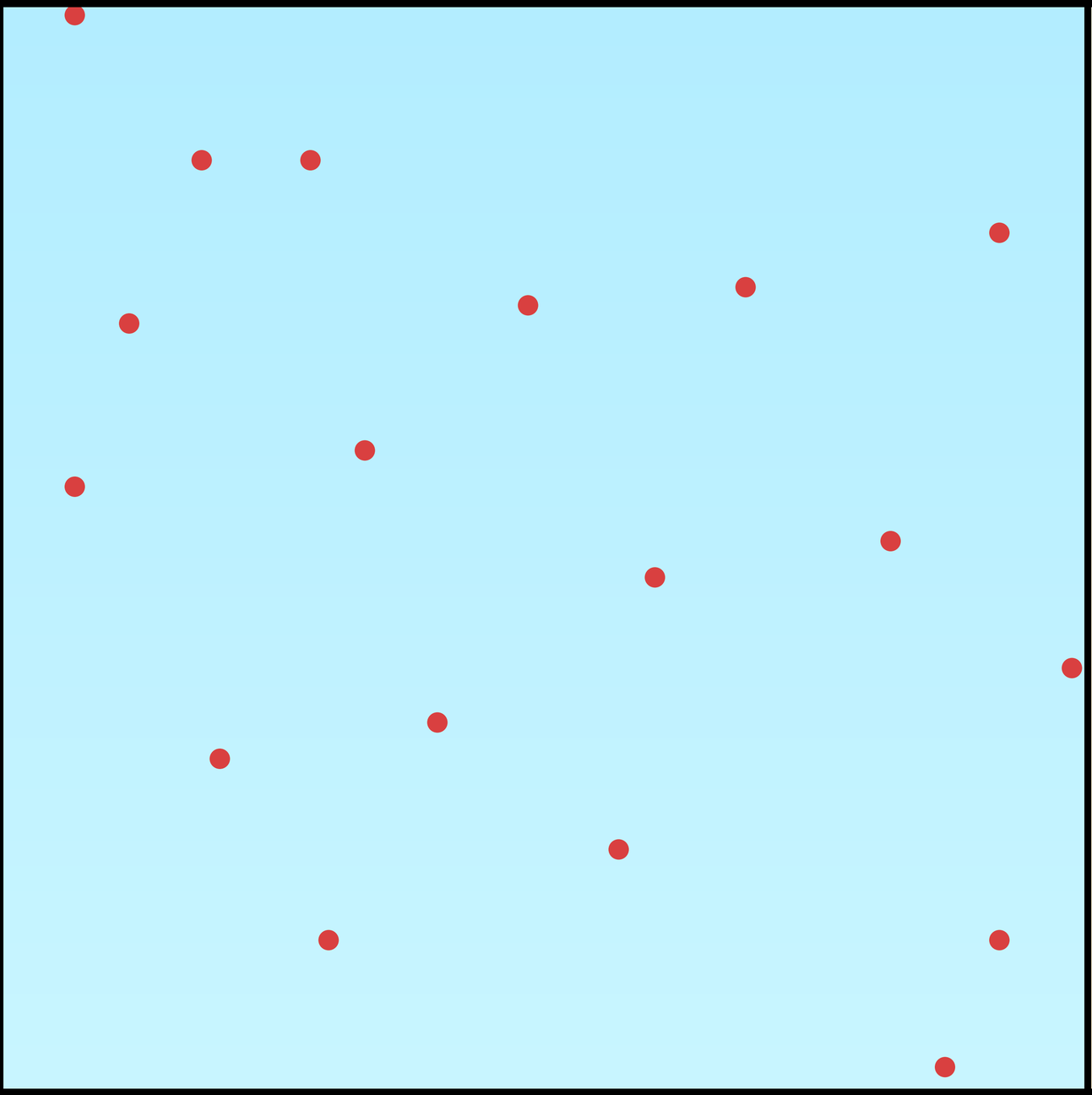


The *Euclidean minimum spanning tree (EMST)* is the minimum spanning tree of the complete graph on a point set $P$.

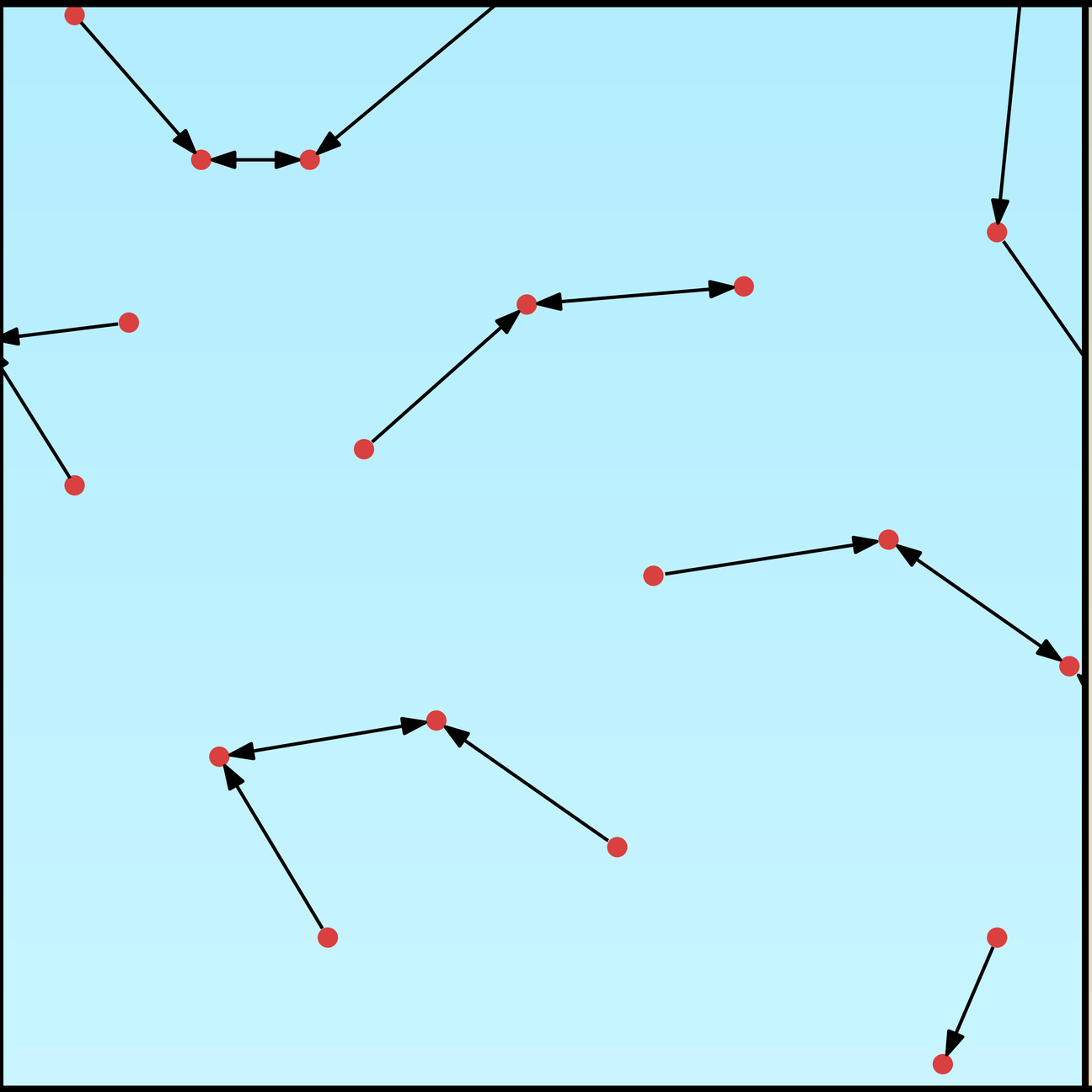
The EMST is a subgraph of the Gabriel graph.

The EMST is not really a proximity structure, since it is not completely defined locally!

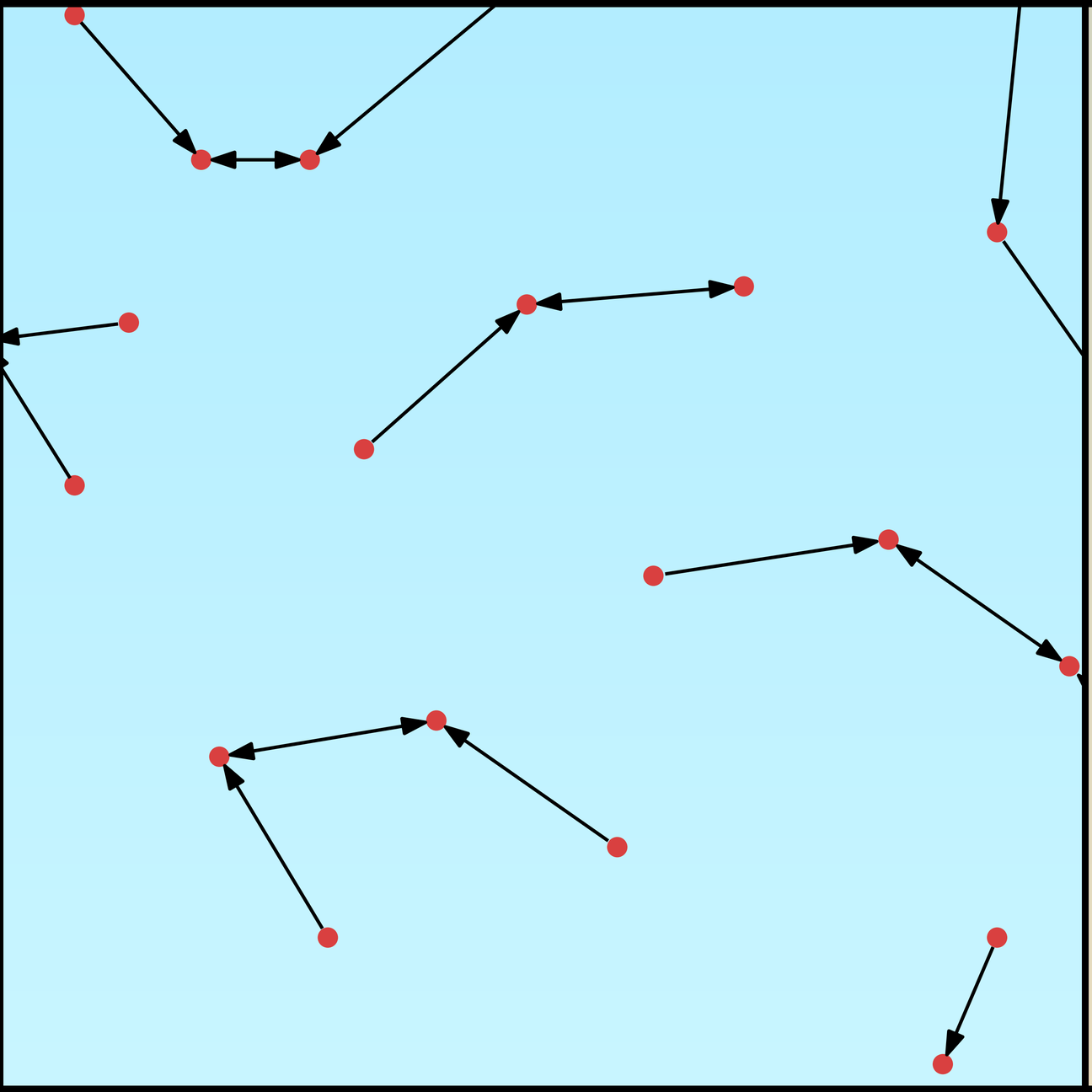
The *Nearest-Neighbour graph* is the graph that stores for every point in a point set $P$ its nearest neighbour.



The *Nearest-Neighbour graph* is the graph that stores for every point in a point set $P$ its nearest neighbour.

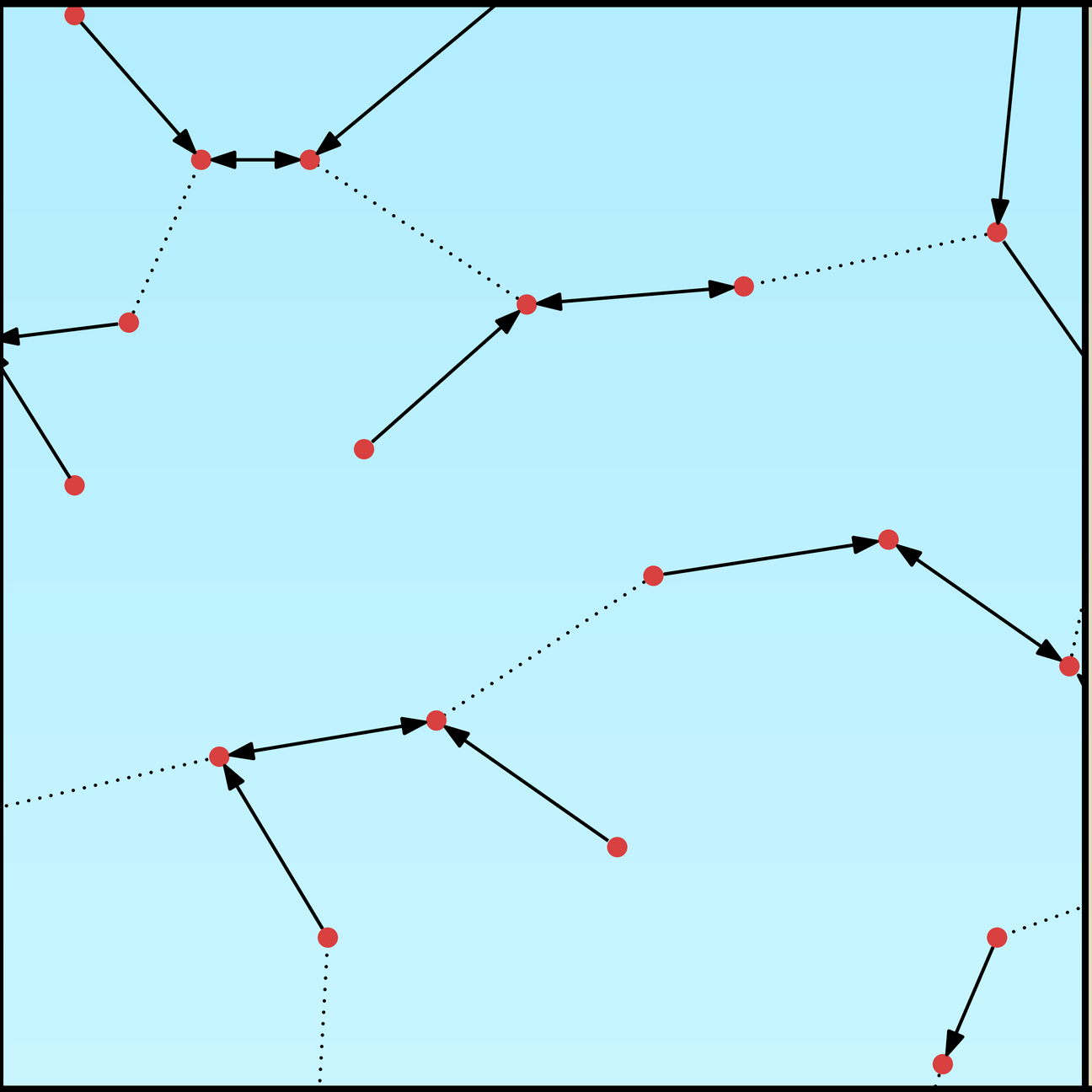


The *Nearest-Neighbour graph* is the graph that stores for every point in a point set $P$ its nearest neighbour.



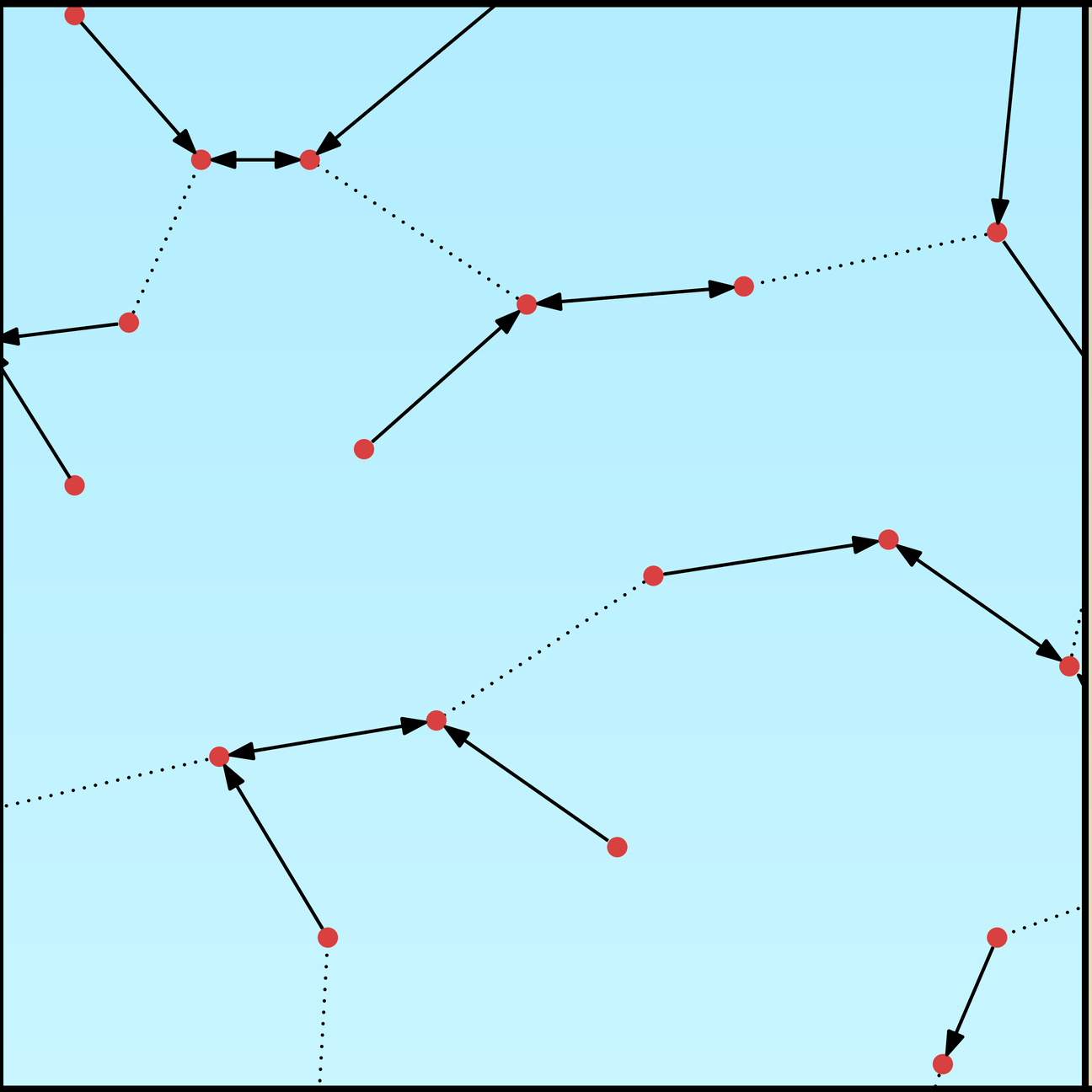
The *Nearest-Neighbour graph* is the graph that stores for every point in a point set $P$ its nearest neighbour.

The Nearest-Neighbour graph is a subgraph of the EMST.



The *Nearest-Neighbour graph* is the graph that stores for every point in a point set $P$ its nearest neighbour.

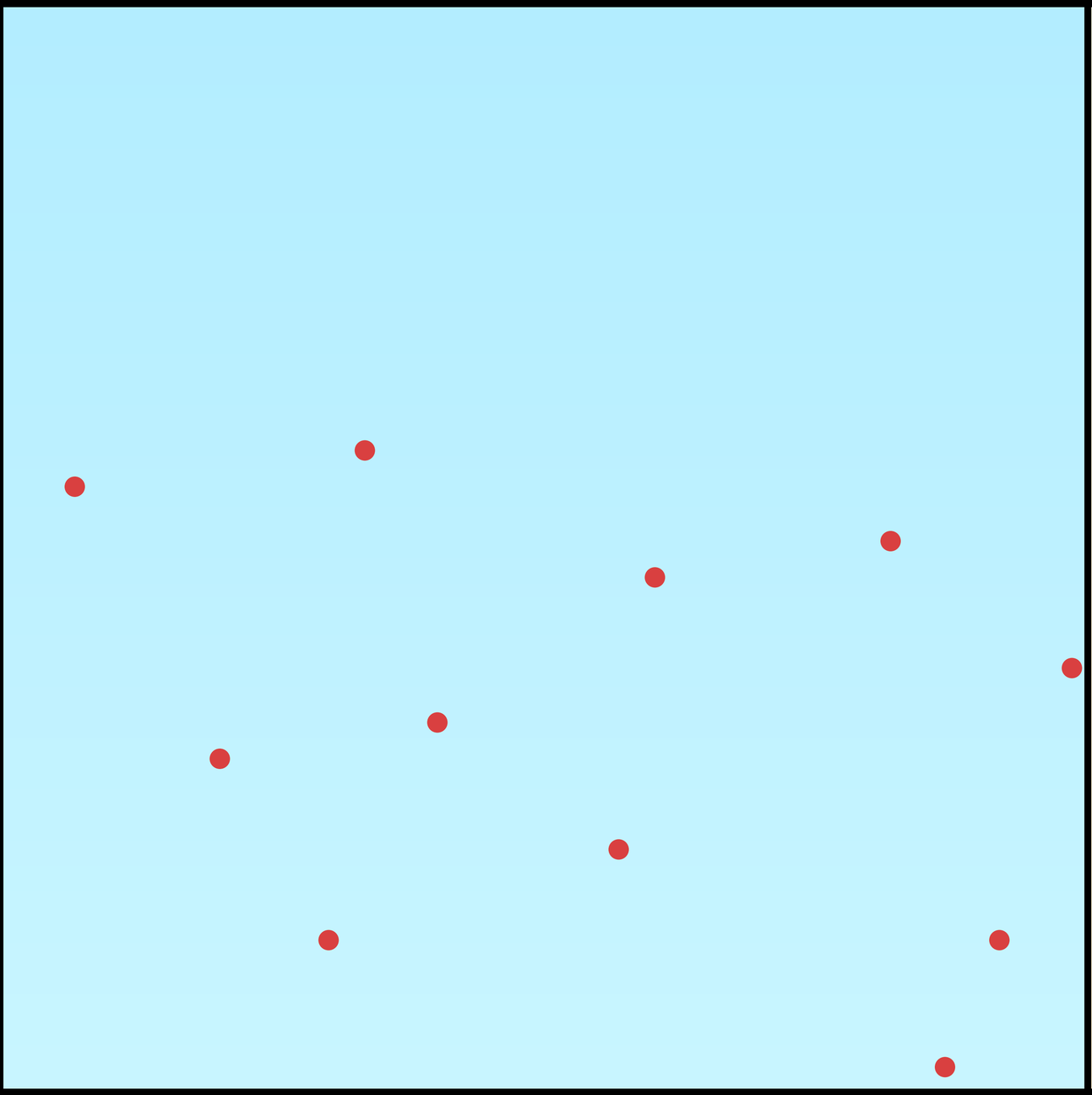
The Nearest-Neighbour graph is a subgraph of the EMST.



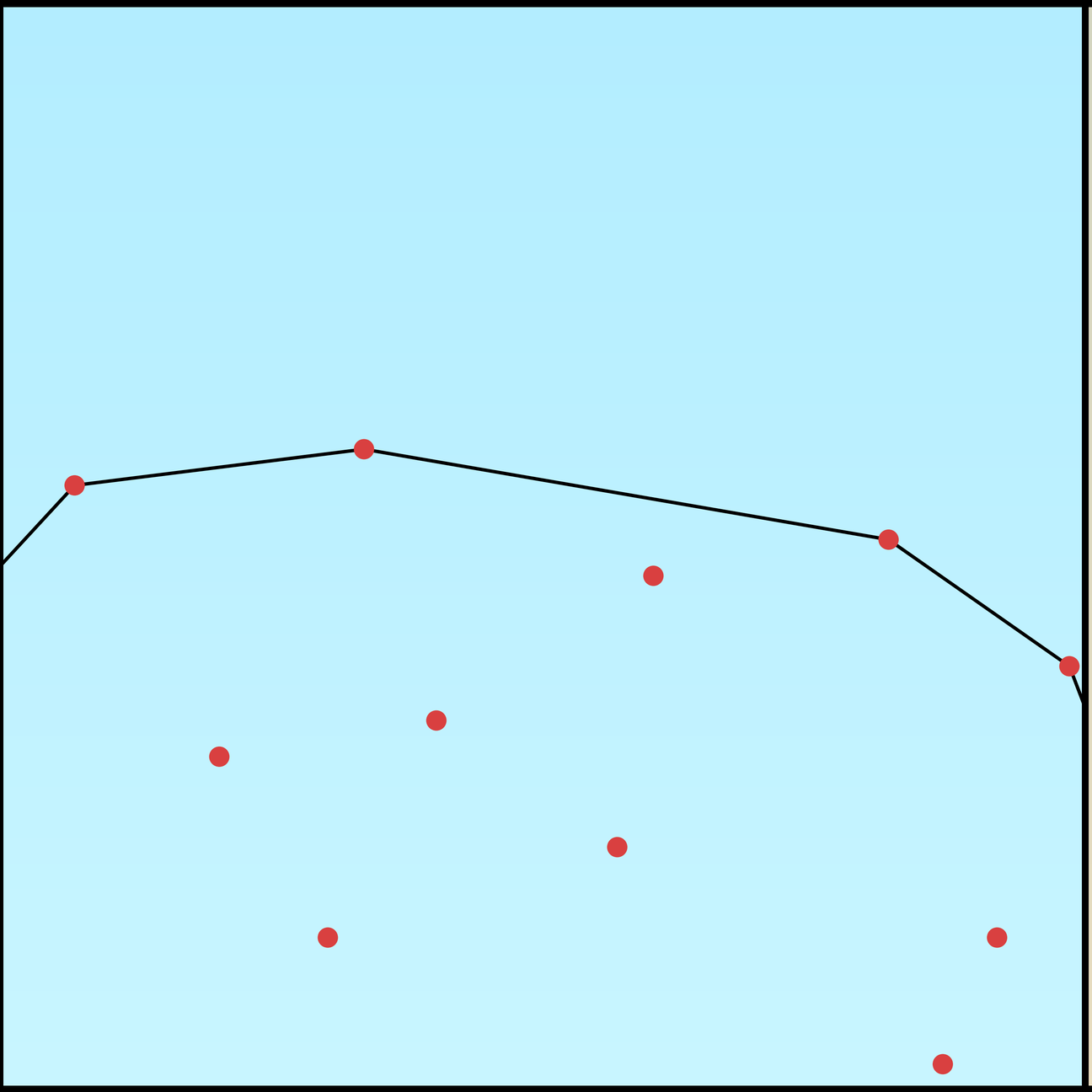
The *Nearest-Neighbour graph* is the graph that stores for every point in a point set $P$ its nearest neighbour.

The Nearest-Neighbour graph is a subgraph of the EMST.

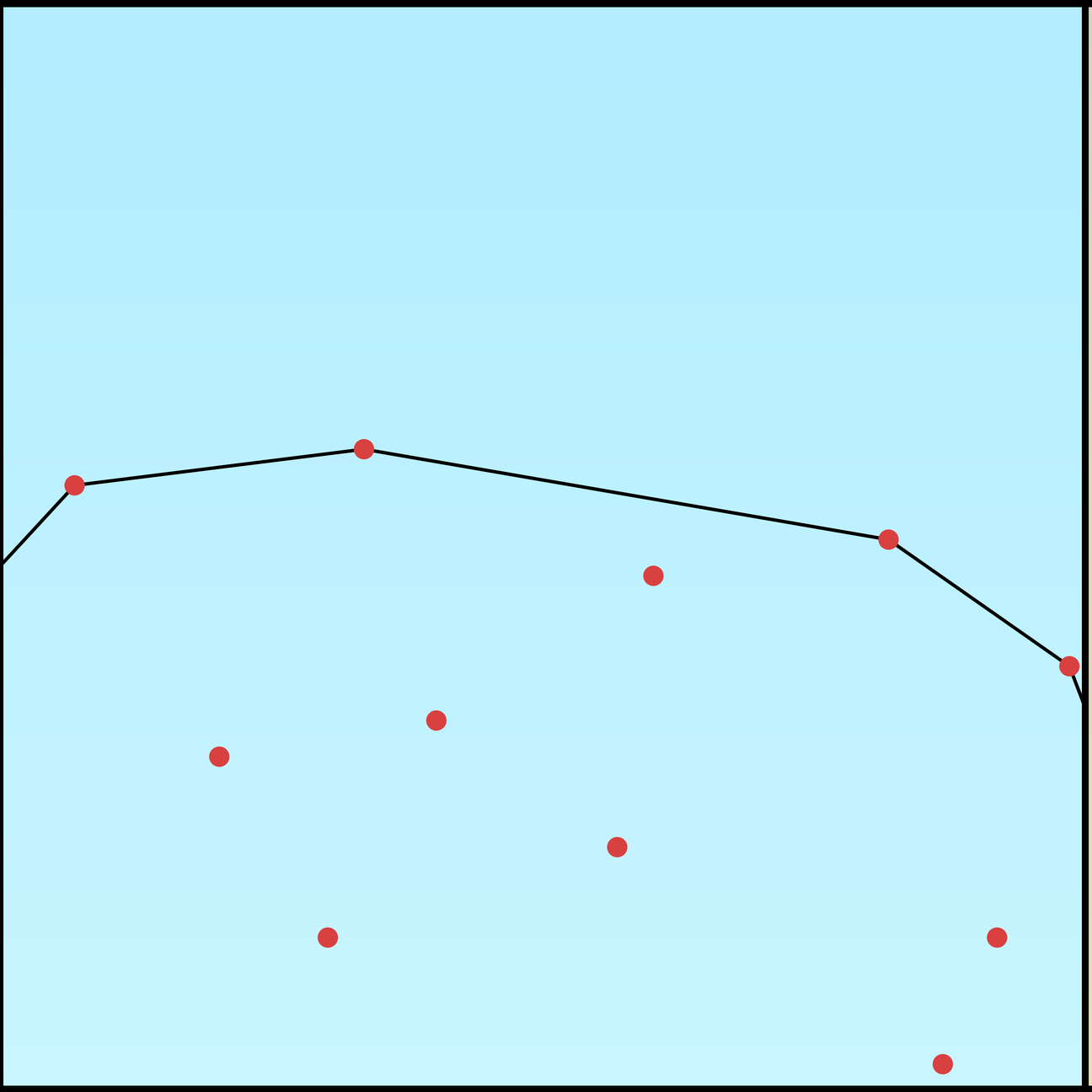
The NN graph is locally defined, but not necessarily connected.



Slight different: the *convex hull* of a point set $P$ is the smallest convex set that contains $P$.

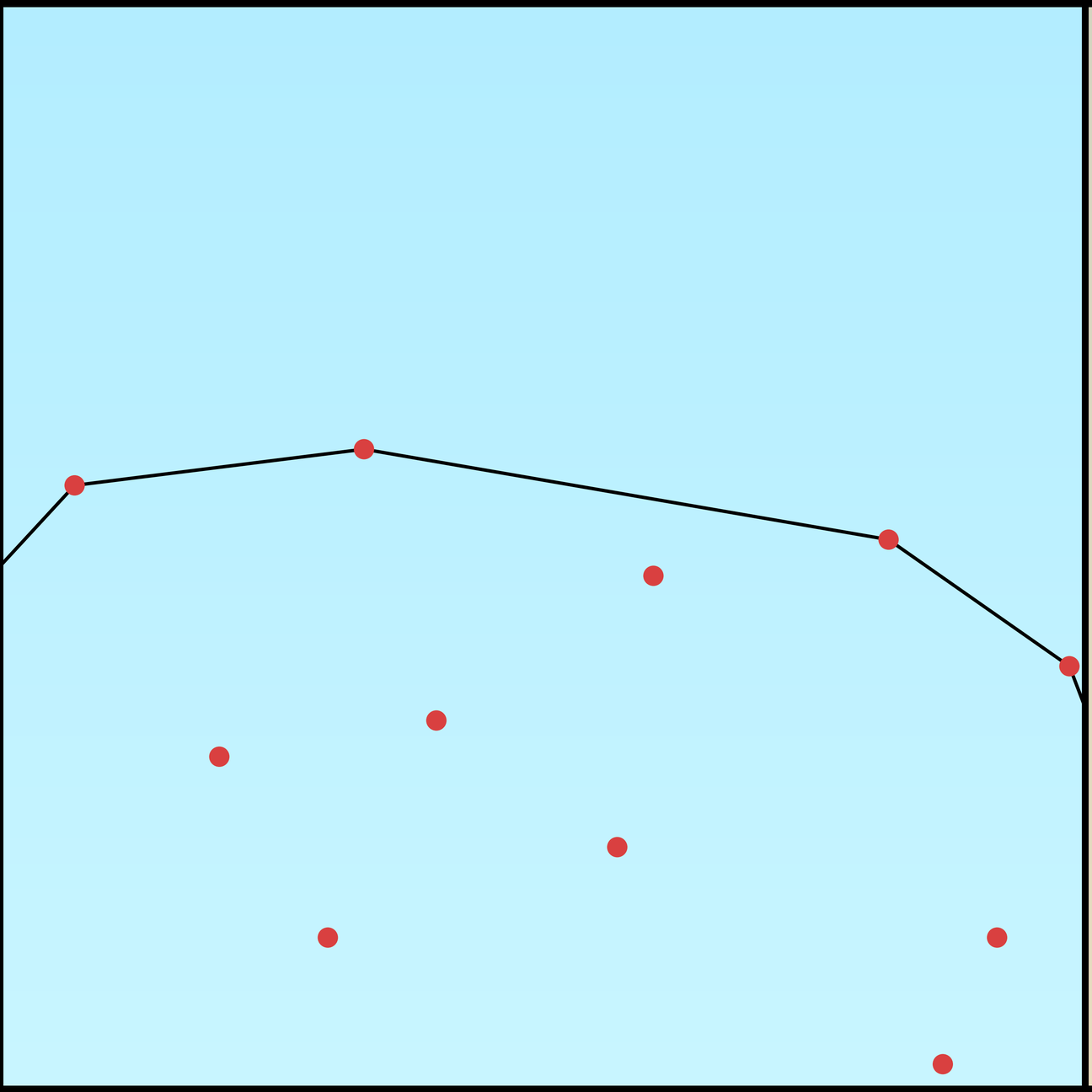


Slight different: the *convex hull* of a point set $P$ is the smallest convex set that contains $P$.



Slight different: the *convex hull* of a point set $P$ is the smallest convex set that contains $P$.

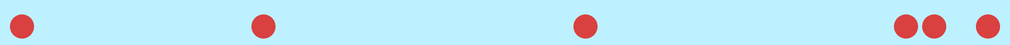
The convex hull is a local structure, but only on the boundary of the set.



Slight different: the *convex hull* of a point set $P$ is the smallest convex set that contains $P$.

The convex hull is a local structure, but only on the boundary of the set.

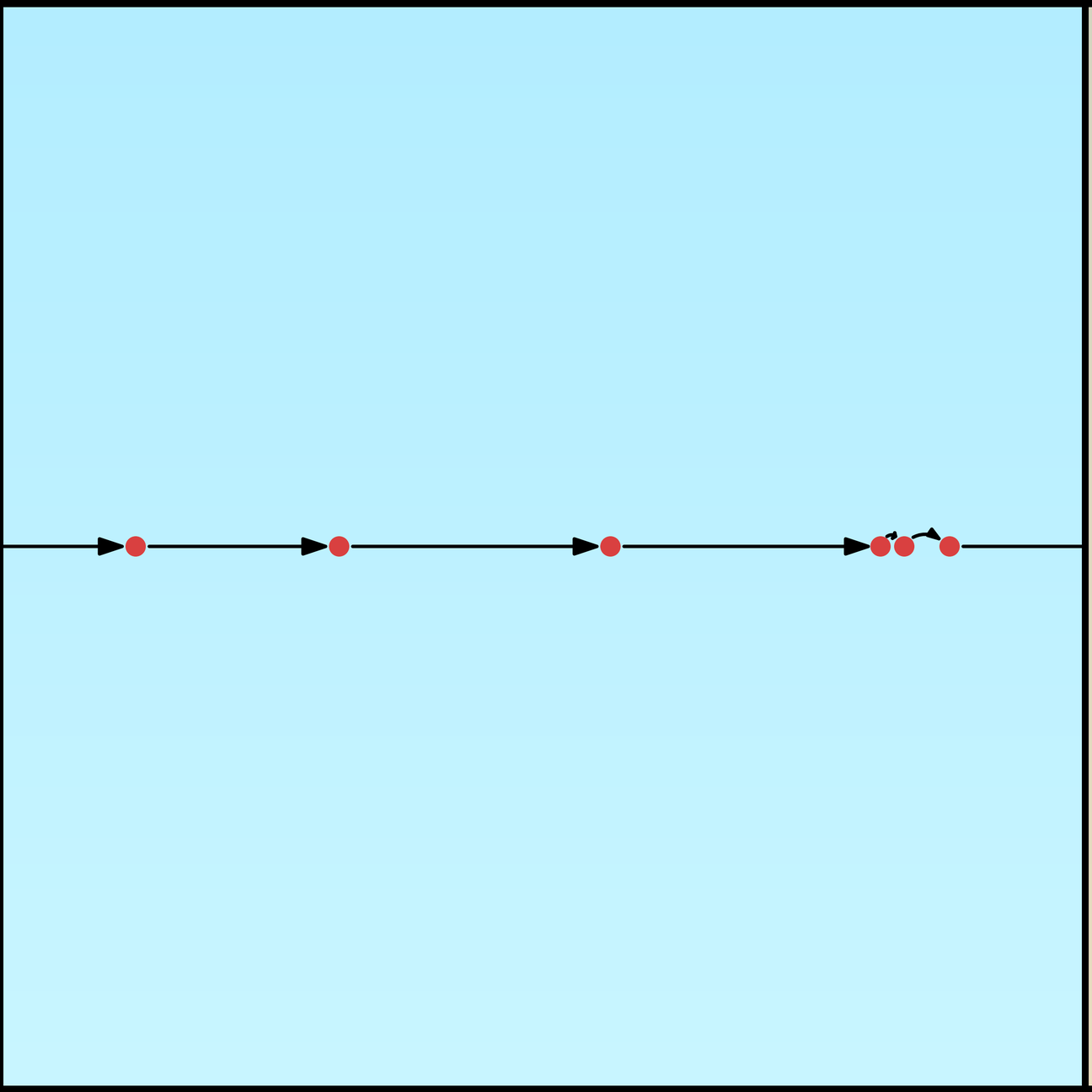
Related to the *sorted order* of a "point set" $P$ in $\mathbb{R}^1$.



Slight different: the *convex hull* of a point set $P$ is the smallest convex set that contains $P$.

The convex hull is a local structure, but only on the boundary of the set.

Related to the *sorted order* of a “point set” $P$ in $\mathbb{R}^1$.

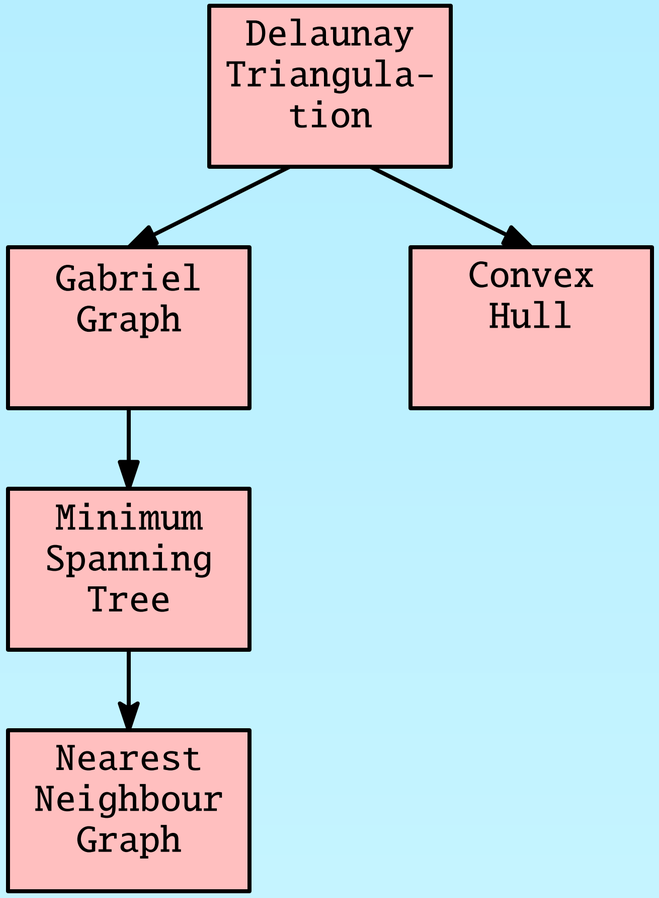


Slight different: the *convex hull* of a point set $P$ is the smallest convex set that contains $P$.

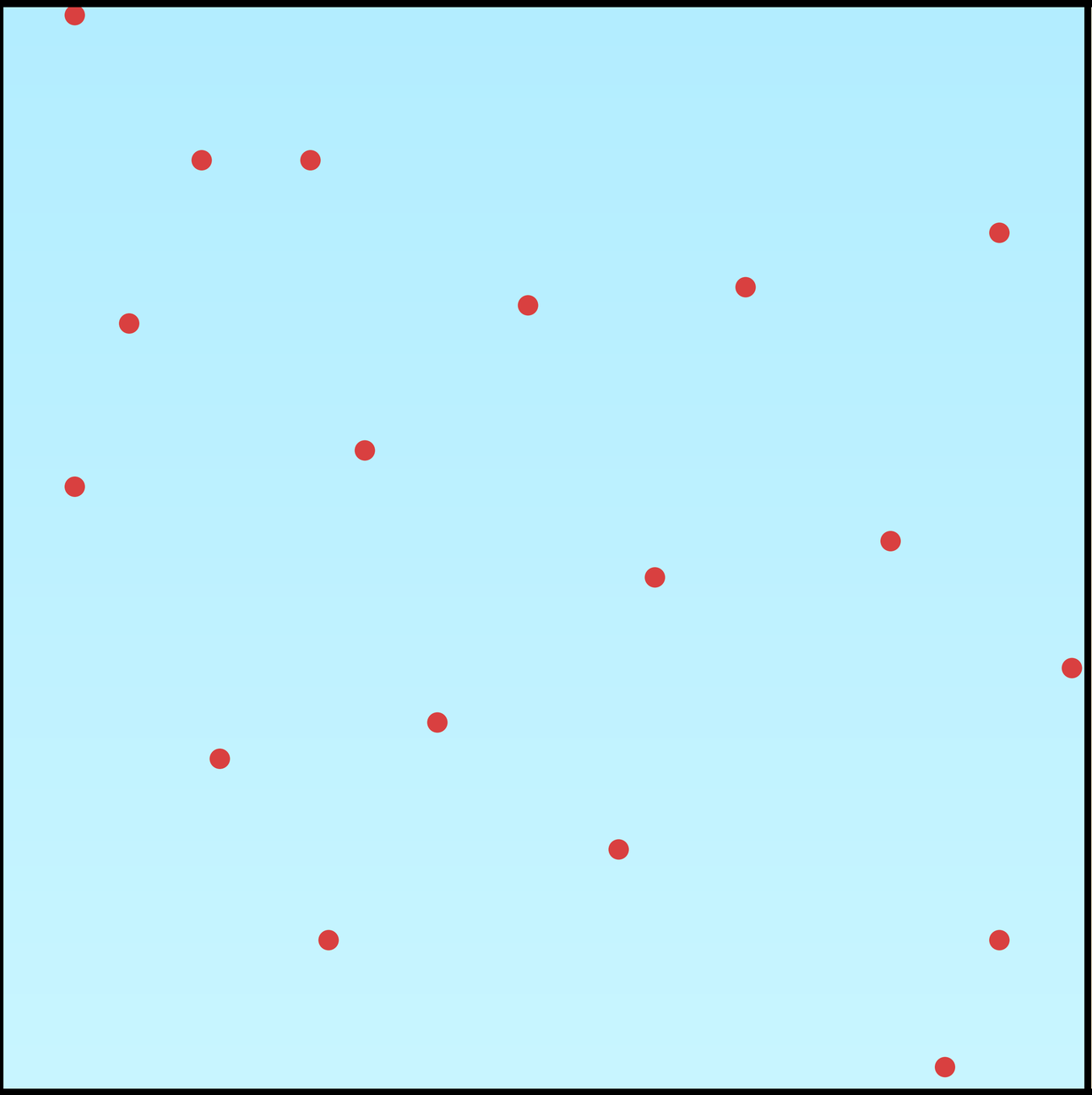
The convex hull is a local structure, but only on the boundary of the set.

Related to the *sorted order* of a “point set” $P$ in $\mathbb{R}^1$.

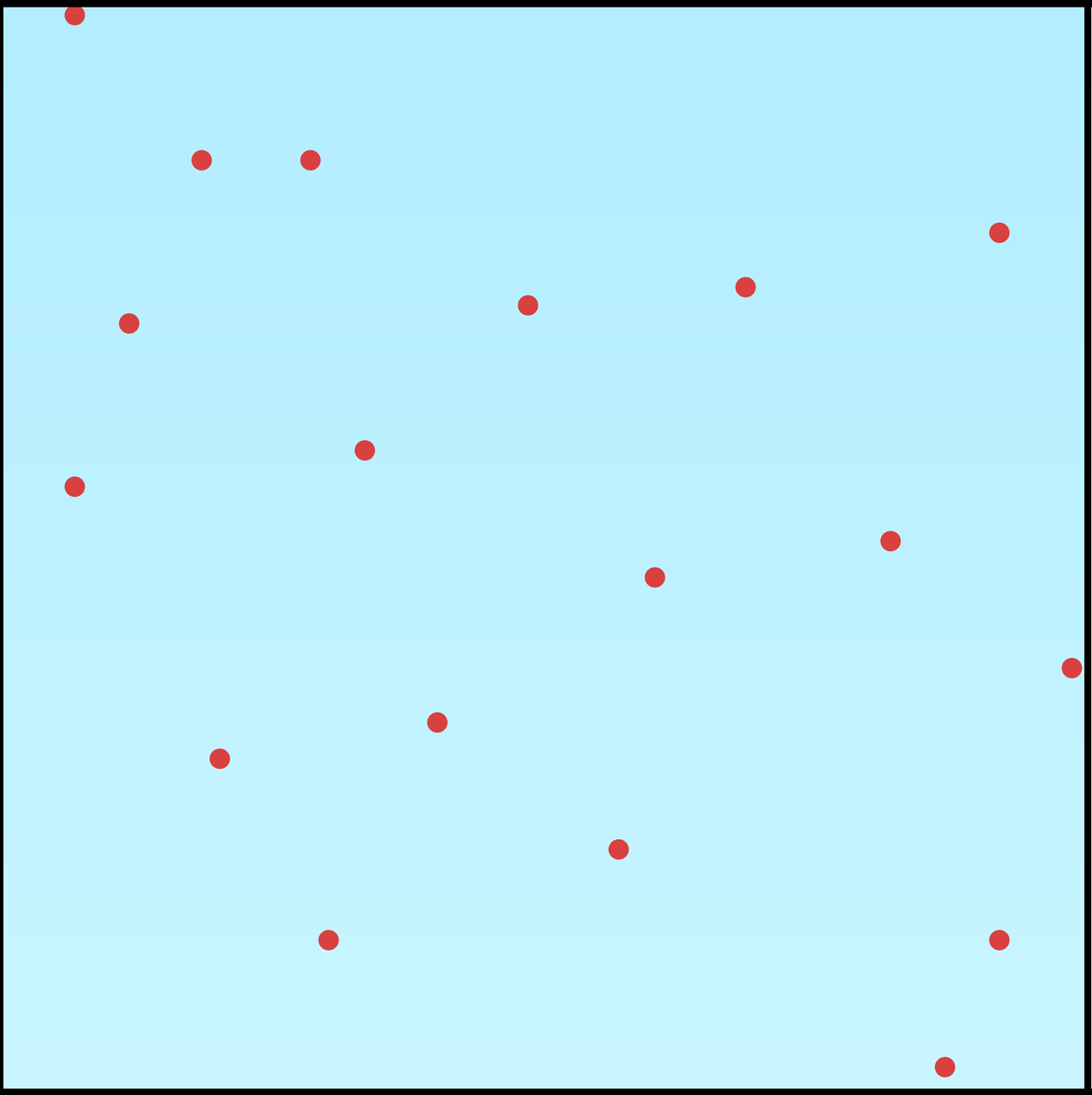
We can consider a “meta-graph” of sub-graph relations between proximity structures.




We can consider a “meta-graph” of sub-graph relations between proximity structures.



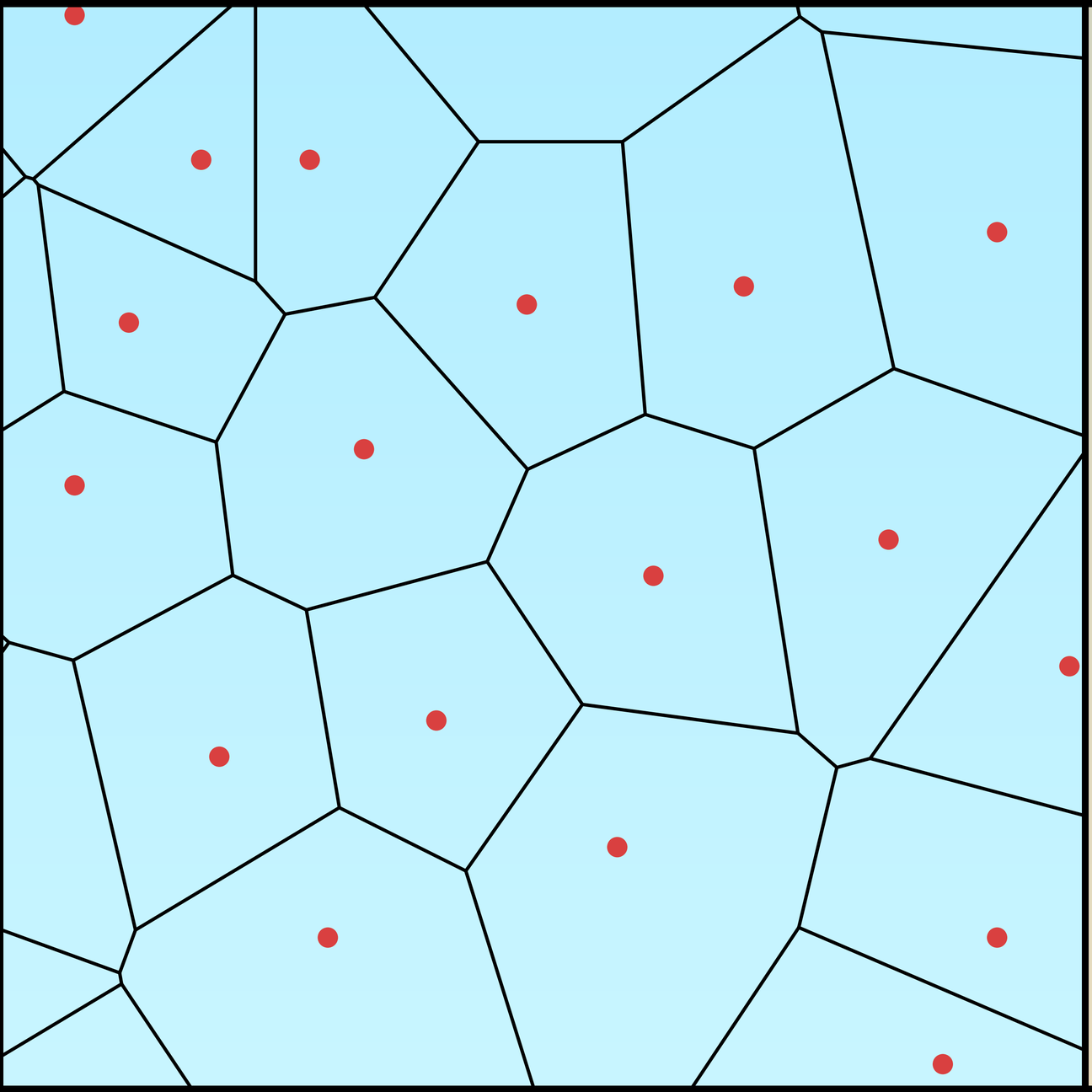
A few more examples.



A few more examples.

The *Voronoï diagram* of $P$ is the subdivision of $\mathbb{R}^2$ into cells for which the same point of $P$ is the closest.

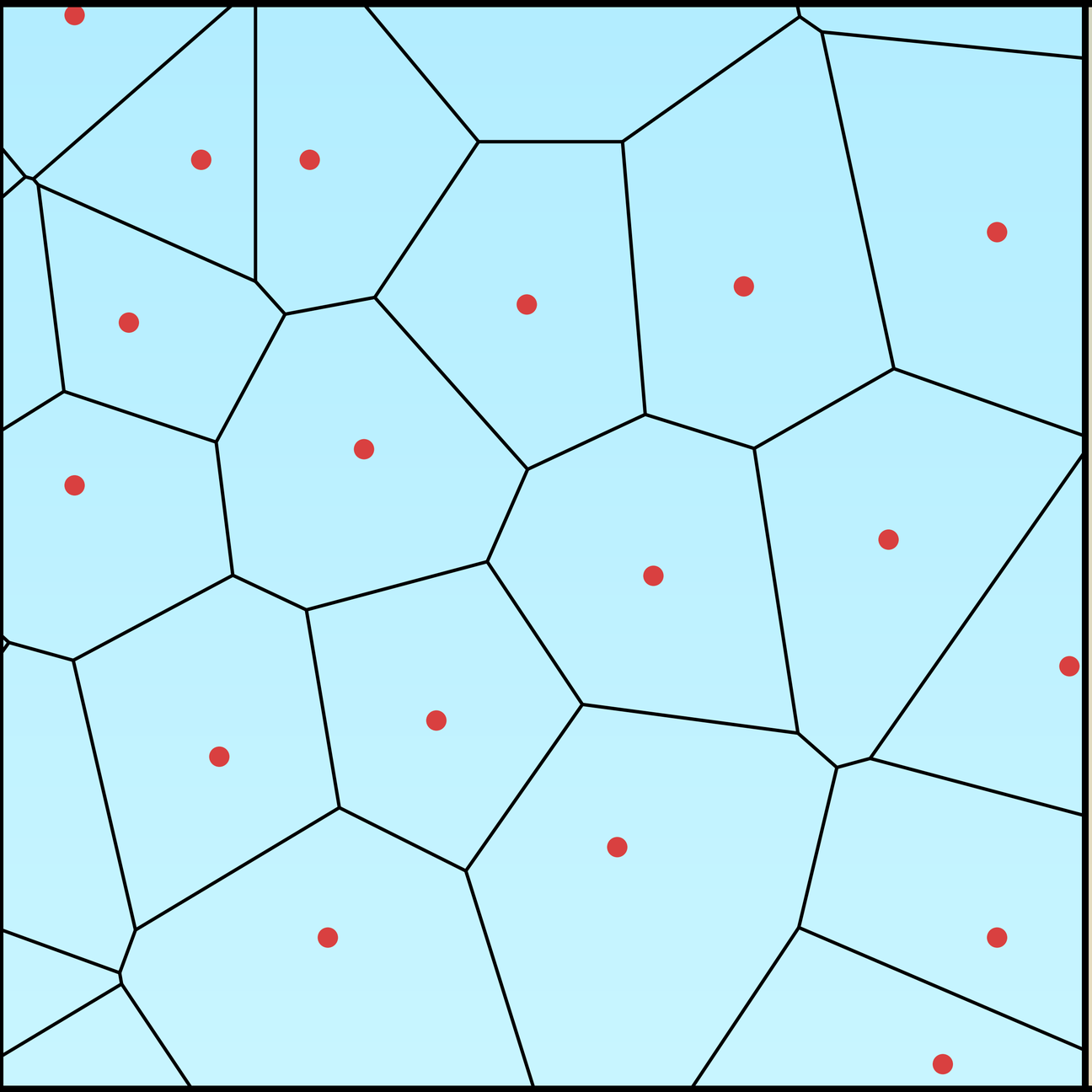
[Dirichlet, 1850]



A few more examples.

The *Voronoï diagram* of $P$ is the subdivision of $\mathbb{R}^2$ into cells for which the same point of $P$ is the closest.

[Dirichlet, 1850]

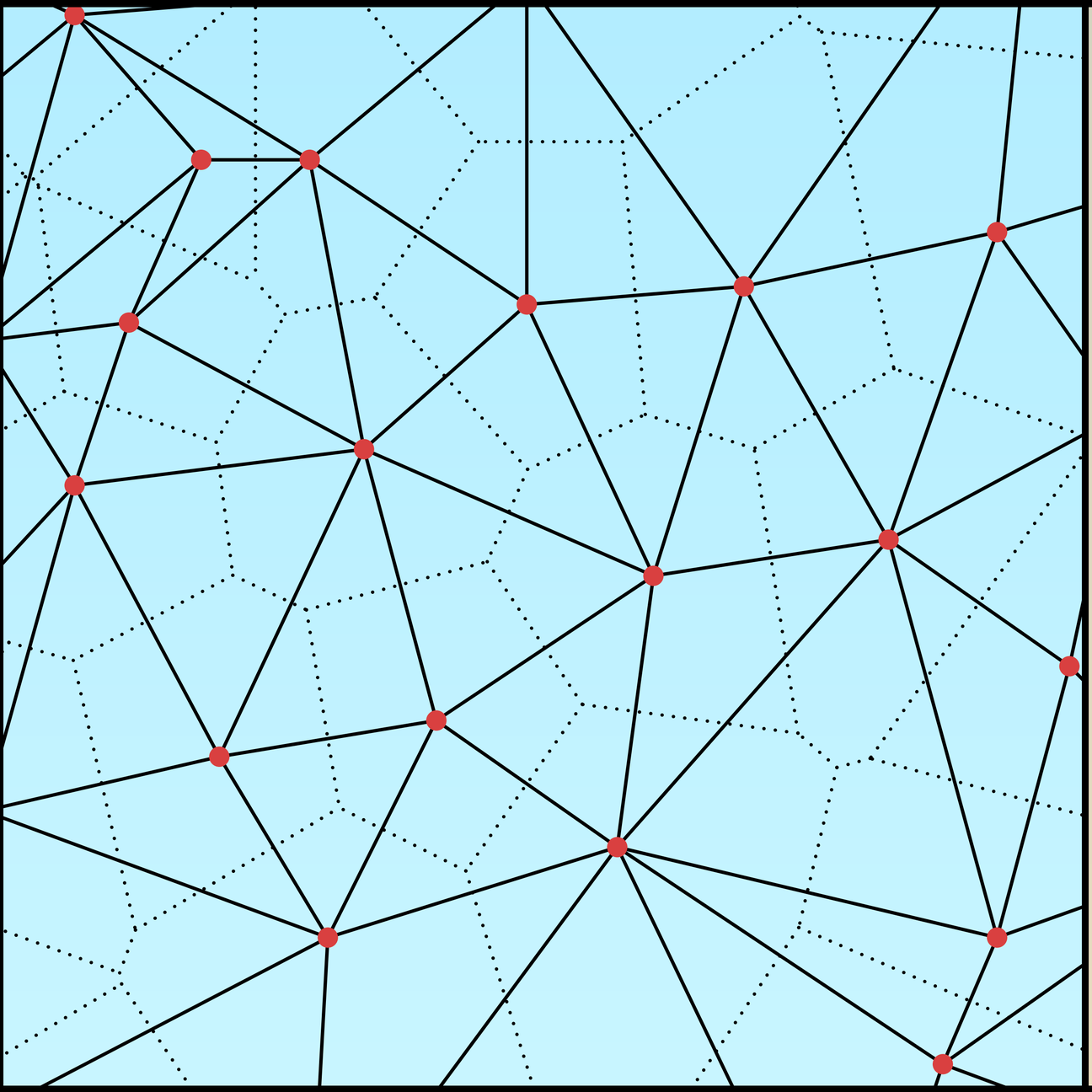


A few more examples.

The *Voronoï diagram* of $P$ is the subdivision of $\mathbb{R}^2$ into cells for which the same point of $P$ is the closest.

De Voronoï diagram is the *dual graph* of the Delaunay triangulation.

[Dirichlet, 1850]

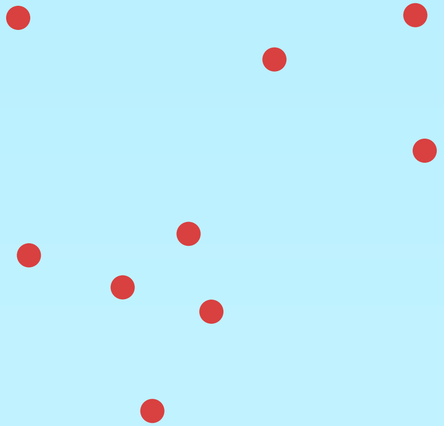


A few more examples.

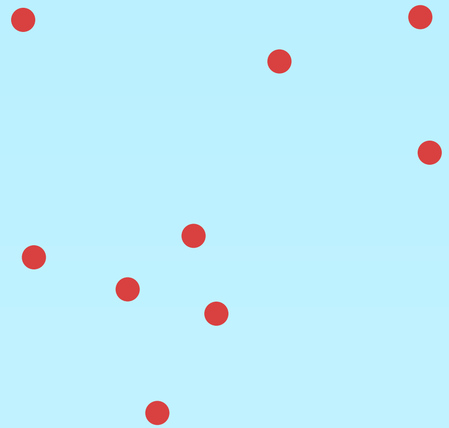
The *Voronoï diagram* of $P$ is the subdivision of $\mathbb{R}^2$ into cells for which the same point of $P$ is the closest.

De Voronoï diagram is the *dual graph* of the Delaunay triangulation.

[Dirichlet, 1850]



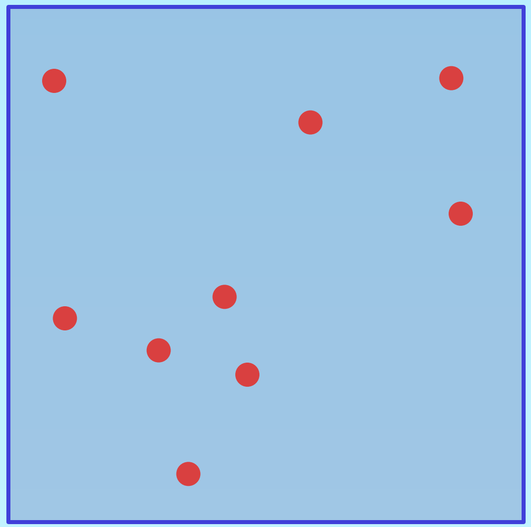
A few more examples.



A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

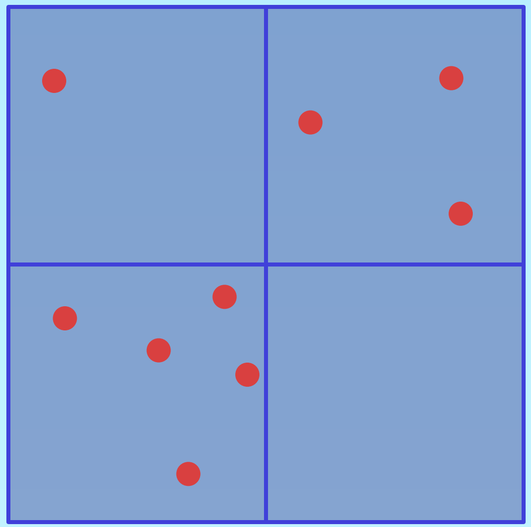
[Finkel & Bentley, 1974]



A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

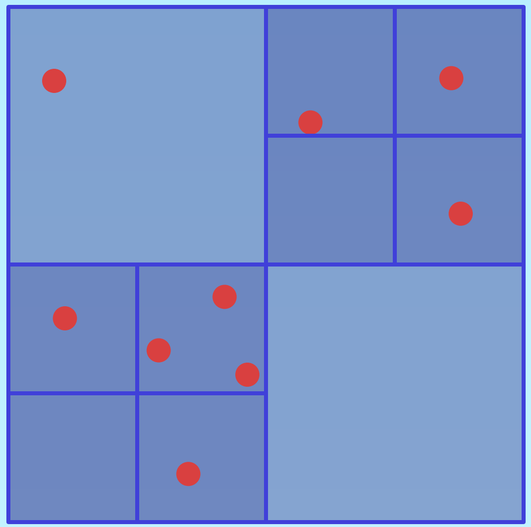
[Finkel & Bentley, 1974]



A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

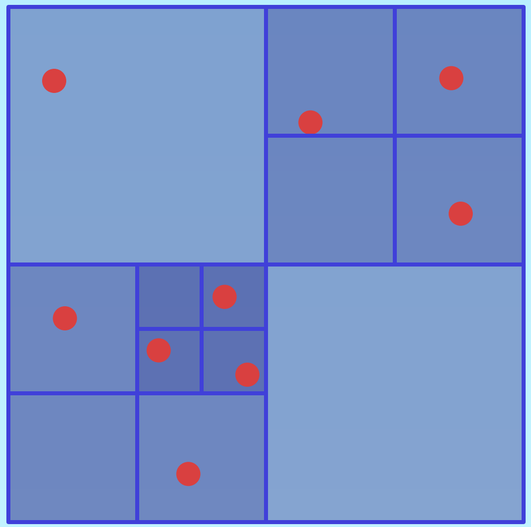
[Finkel & Bentley, 1974]



A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

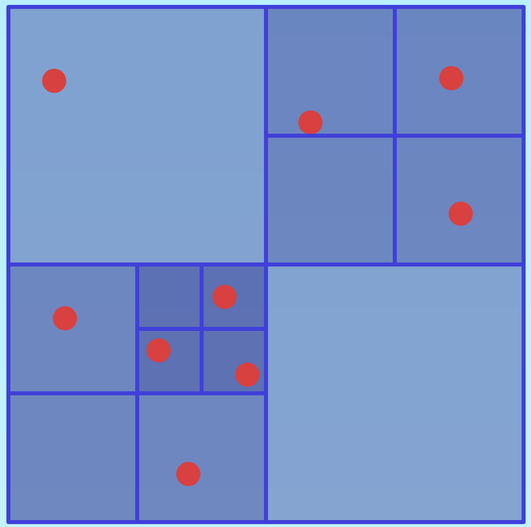
[Finkel & Bentley, 1974]



A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

[Finkel & Bentley, 1974]

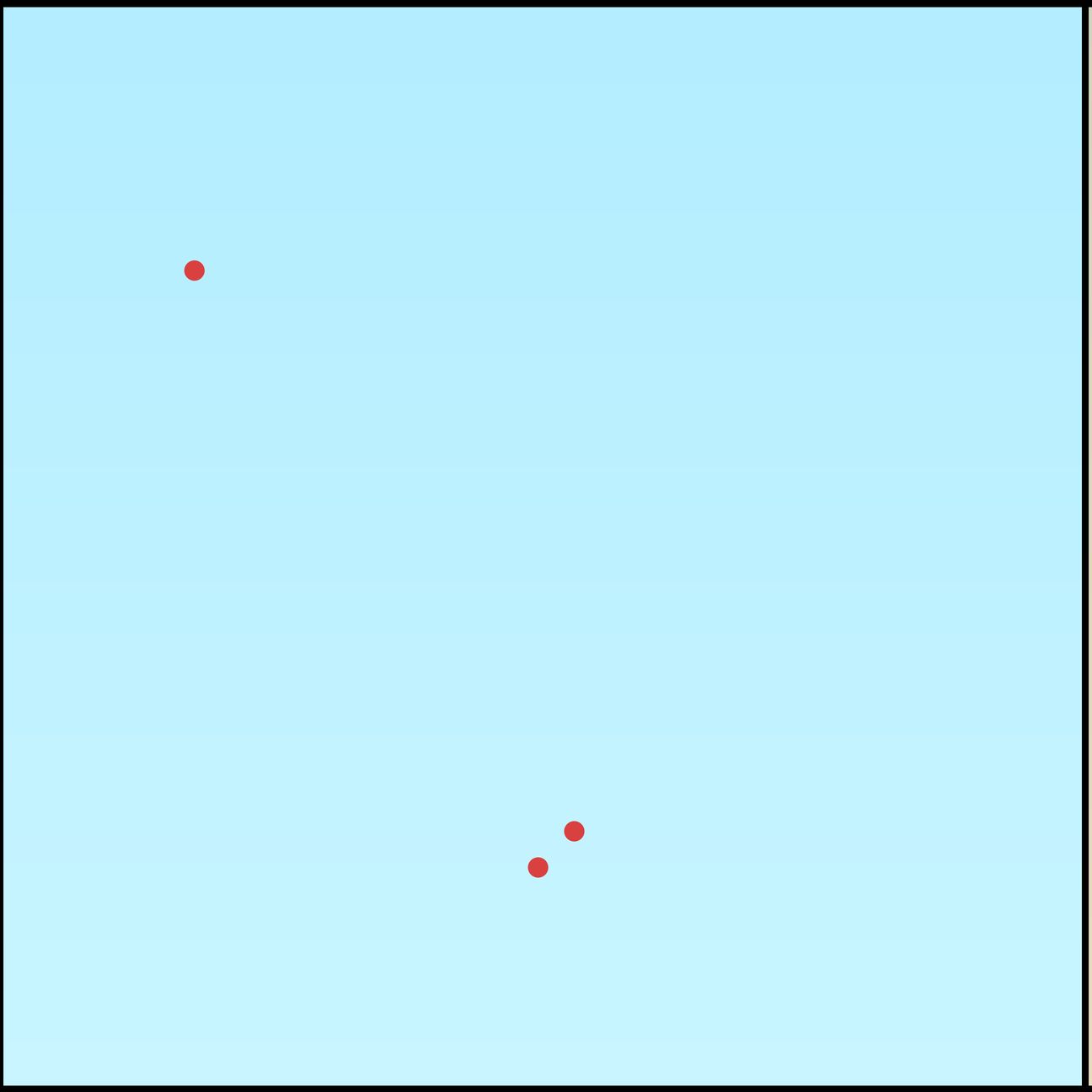


A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

For point sets with large spread, a quadtree needs to be *compressed*.

[Finkel & Bentley, 1974]
[Clarkson, 1983]

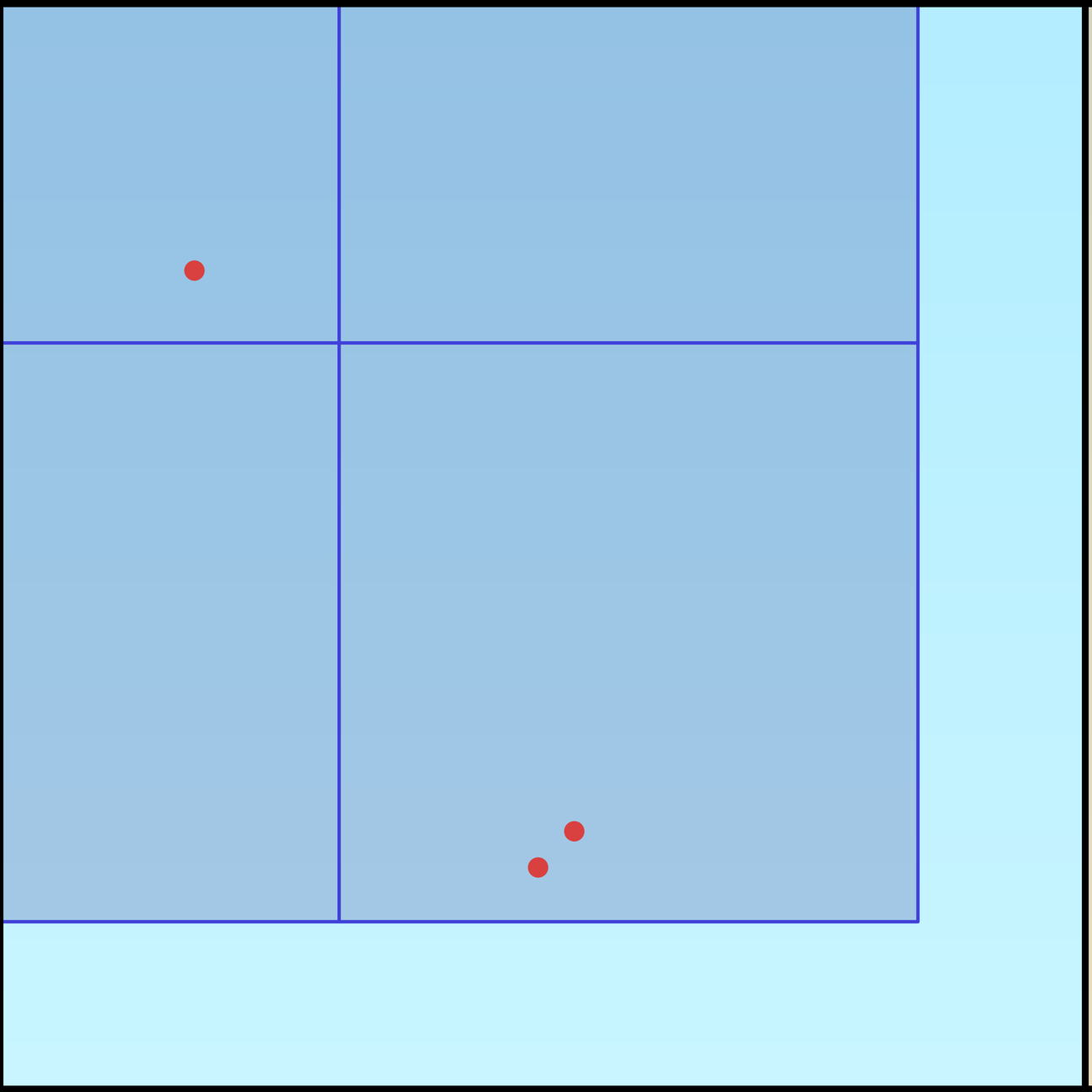


A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

For point sets with large spread, a quadtree needs to be *compressed*.

[Finkel & Bentley, 1974]
[Clarkson, 1983]

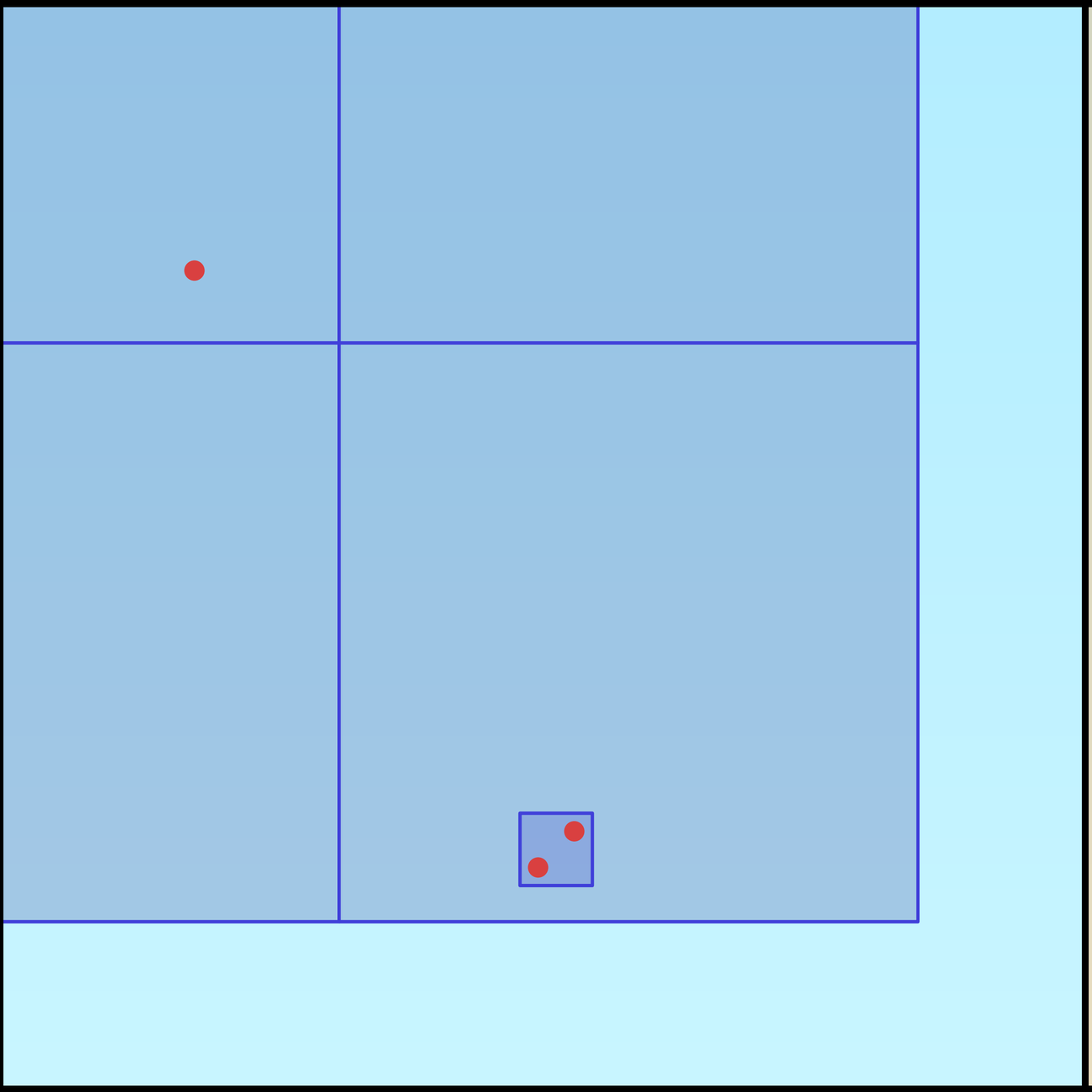


A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

For point sets with large spread, a quadtree needs to be *compressed*.

[Finkel & Bentley, 1974]
[Clarkson, 1983]

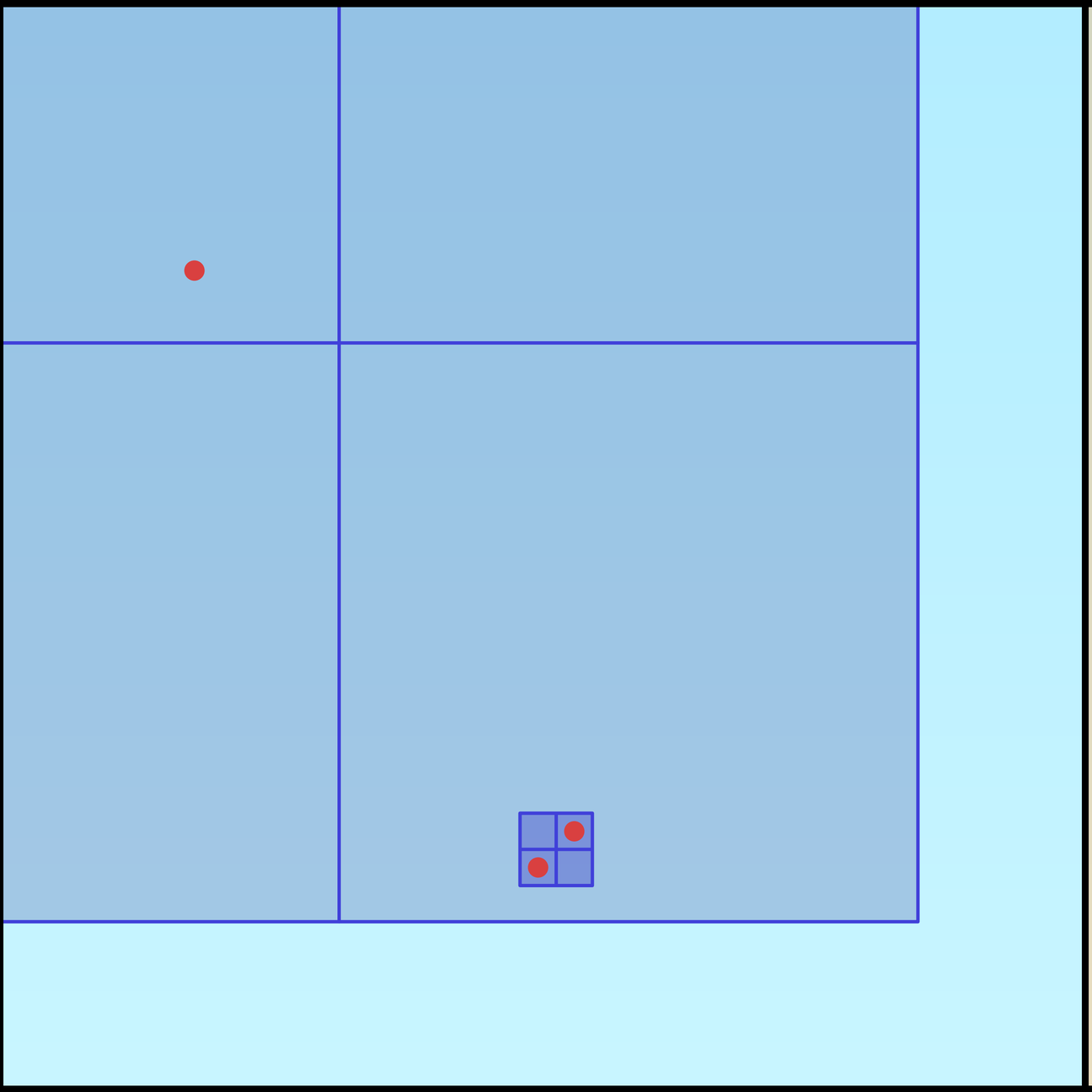


A few more examples.

A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

For point sets with large spread, a quadtree needs to be *compressed*.

[Finkel & Bentley, 1974]
[Clarkson, 1983]

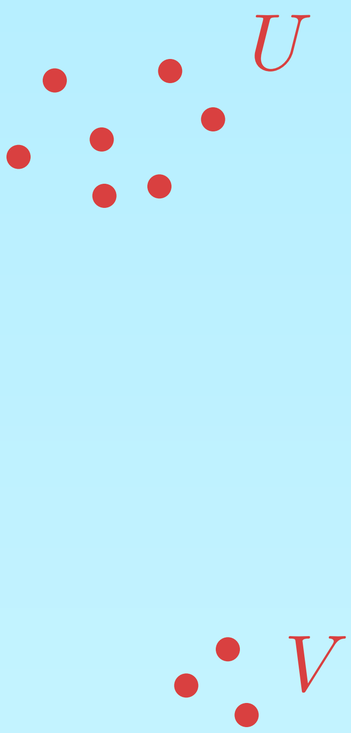


A few more examples.

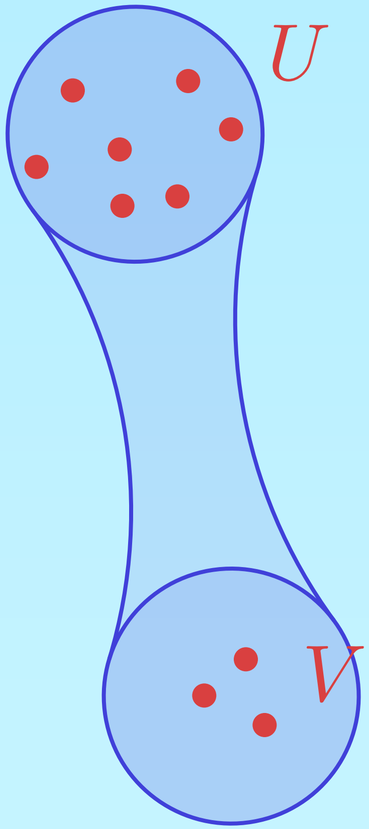
A *quadtree* $T$ for a set of points $P$ is a hierarchical subdivision of a square into smaller squares that separates $P$.

For point sets with large spread, a quadtree needs to be *compressed*.

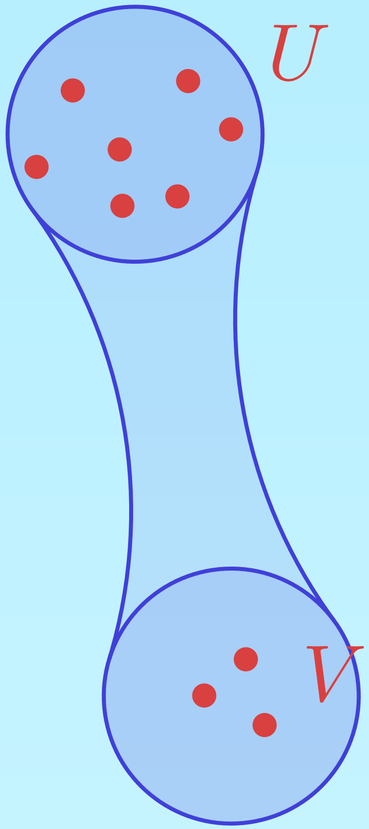
[Finkel & Bentley, 1974]
[Clarkson, 1983]




A *$c$-well-separated pair* is a pair of point sets $U, V$ whose distance to each other is at least $c$ times their diameters.

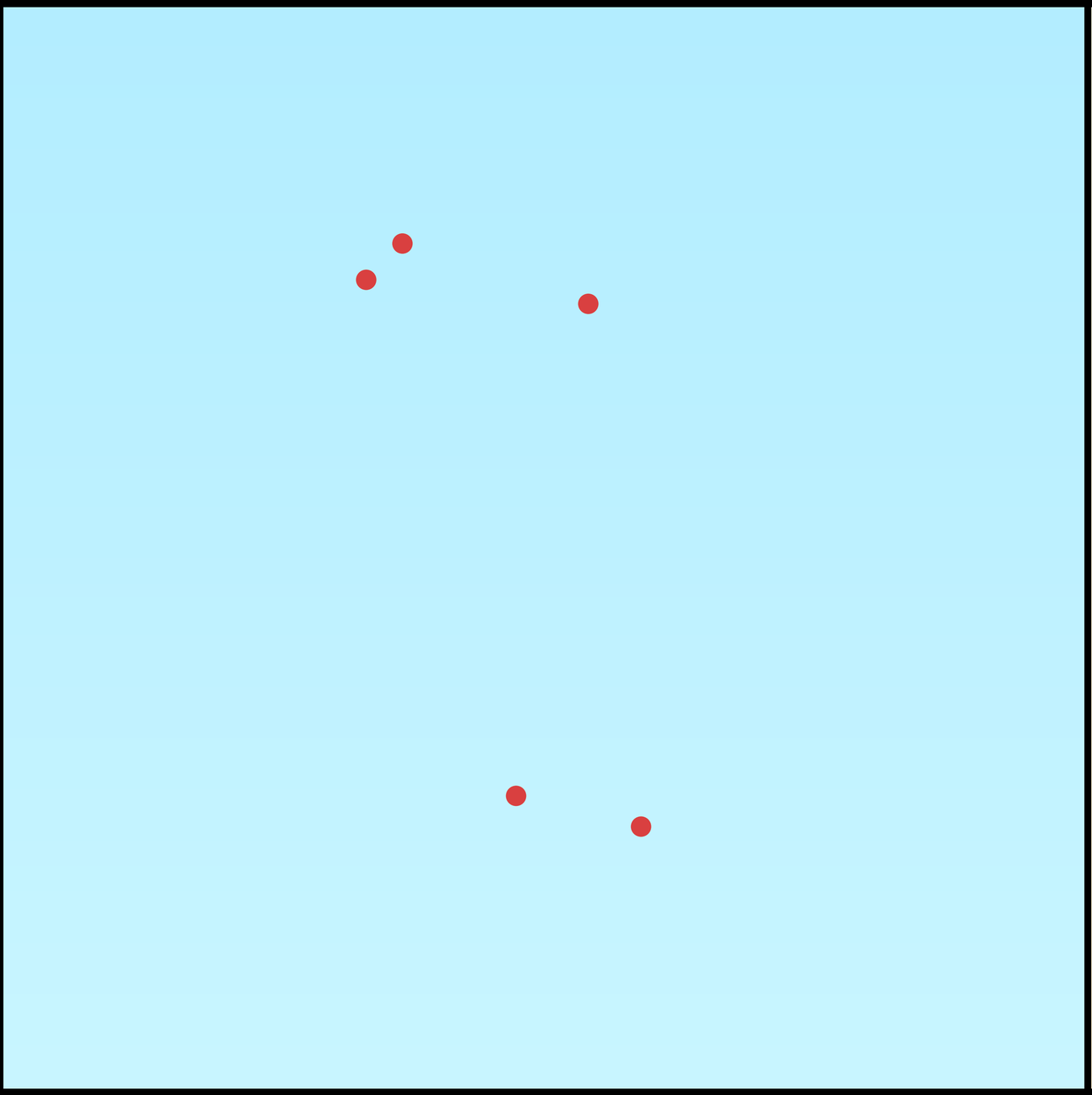



A *$c$-well-separated pair* is a pair of point sets $U, V$ whose distance to each other is at least $c$ times their diameters.



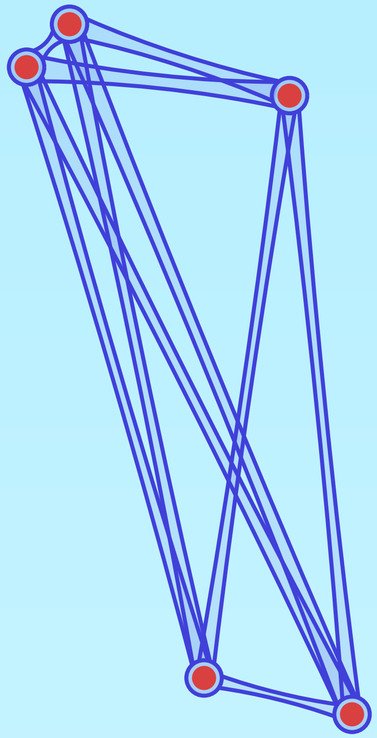

A *$c$-well-separated pair* is a pair of point sets $U, V$ whose distance to each other is at least $c$ times their diameters.

A *$c$-well-separated pair decomposition (WSPD)* of a set of points $P$ is a set of well-separated pairs of subsets of $P$ such that every pair of points $p, q \in P$ appear in exactly one pair of subsets.



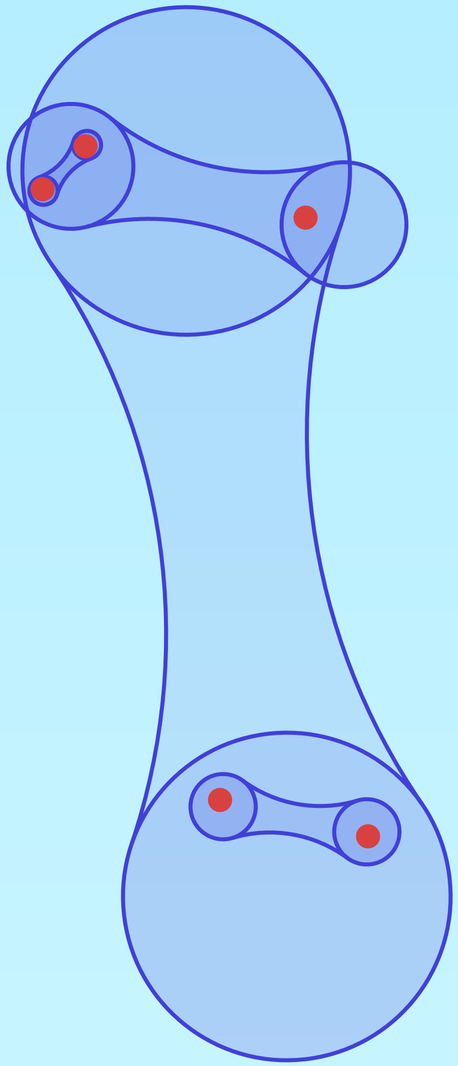
A *$c$-well-separated pair* is a pair of point sets $U, V$ whose distance to each other is at least $c$ times their diameters.

A *$c$-well-separated pair decomposition (WSPD)* of a set of points $P$ is a set of well-separated pairs of subsets of $P$ such that every pair of points $p, q \in P$ appear in exactly one pair of subsets.



A *$c$-well-separated pair* is a pair of point sets $U, V$ whose distance to each other is at least $c$ times their diameters.

A *$c$-well-separated pair decomposition (WSPD)* of a set of points $P$ is a set of well-separated pairs of subsets of $P$ such that every pair of points $p, q \in P$ appear in exactly one pair of subsets.



A *$c$-well-separated pair* is a pair of point sets $U, V$ whose distance to each other is at least $c$ times their diameters.

A *$c$-well-separated pair decomposition (WSPD)* of a set of points $P$ is a set of well-separated pairs of subsets of $P$ such that every pair of points $p, q \in P$ appear in exactly one pair of subsets.

# Chapter 1
## *PROXIMITY STRUCTURES*

Point sets and proximity

Computational questions

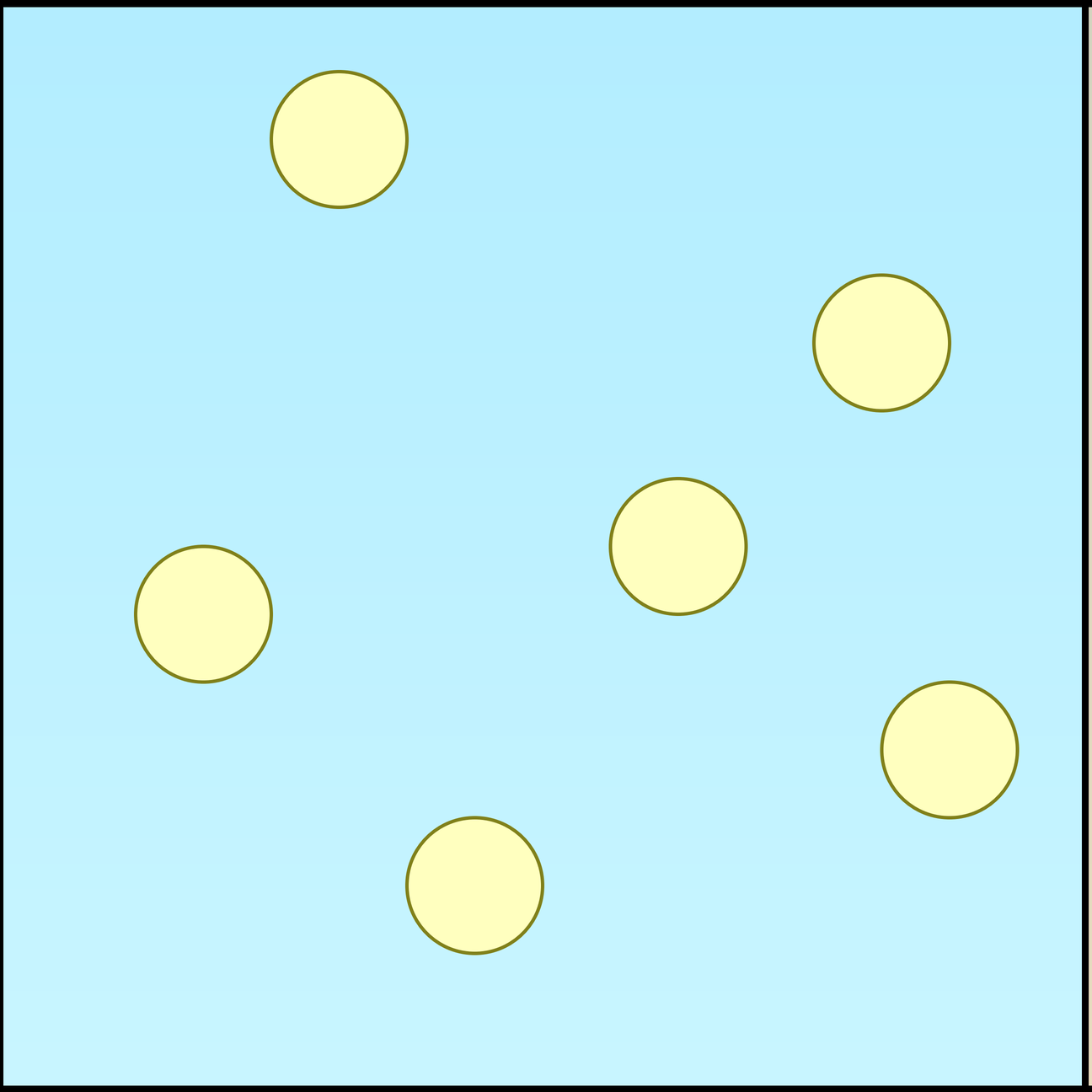
State of the art in 1D

State of the art in 2D

A *pointer machine* models how a computer can access and manipulate *data*.

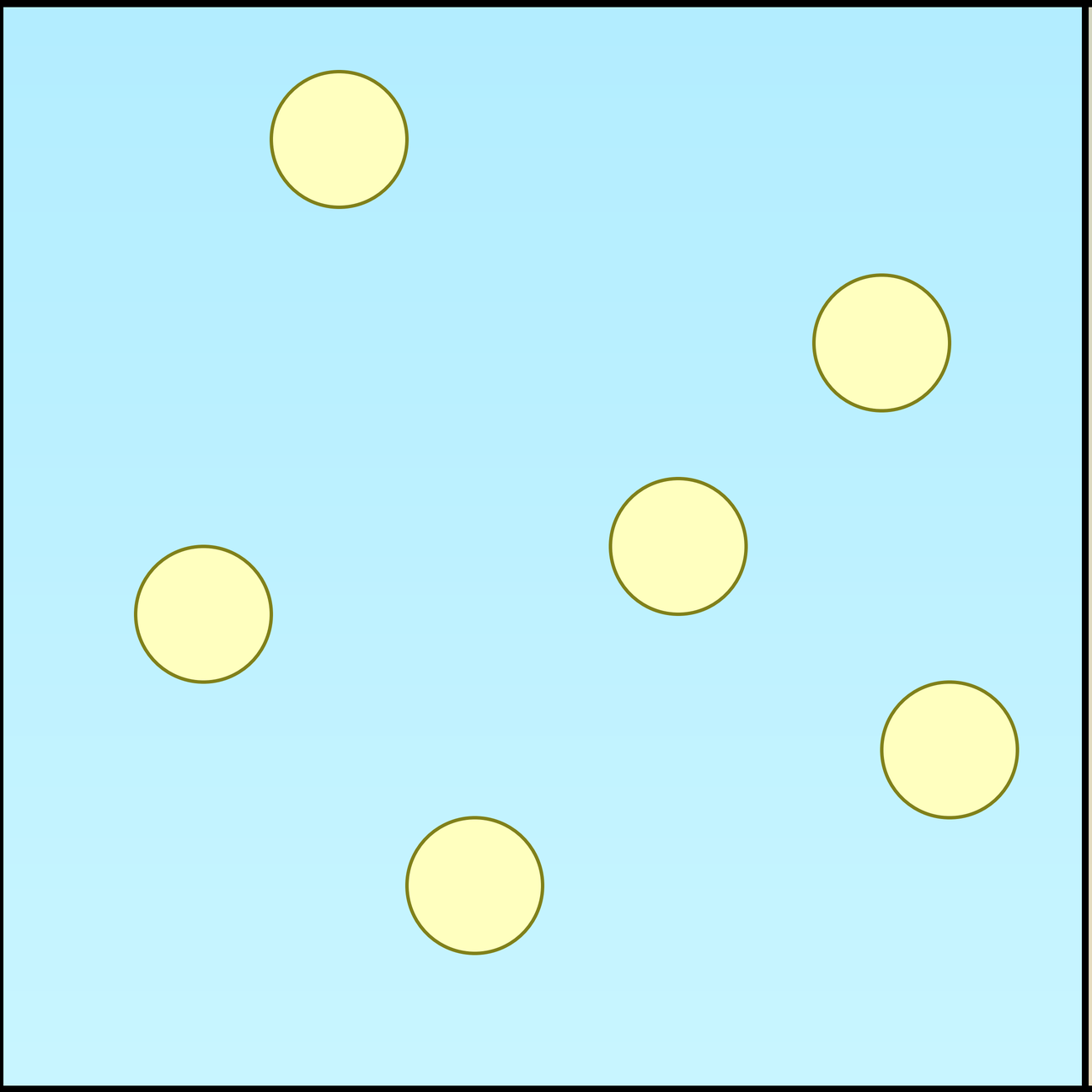
A *pointer machine* models how a computer can access and manipulate *data*.

Data is stored in *cells*.



A *pointer machine* models how a computer can access and manipulate *data*.

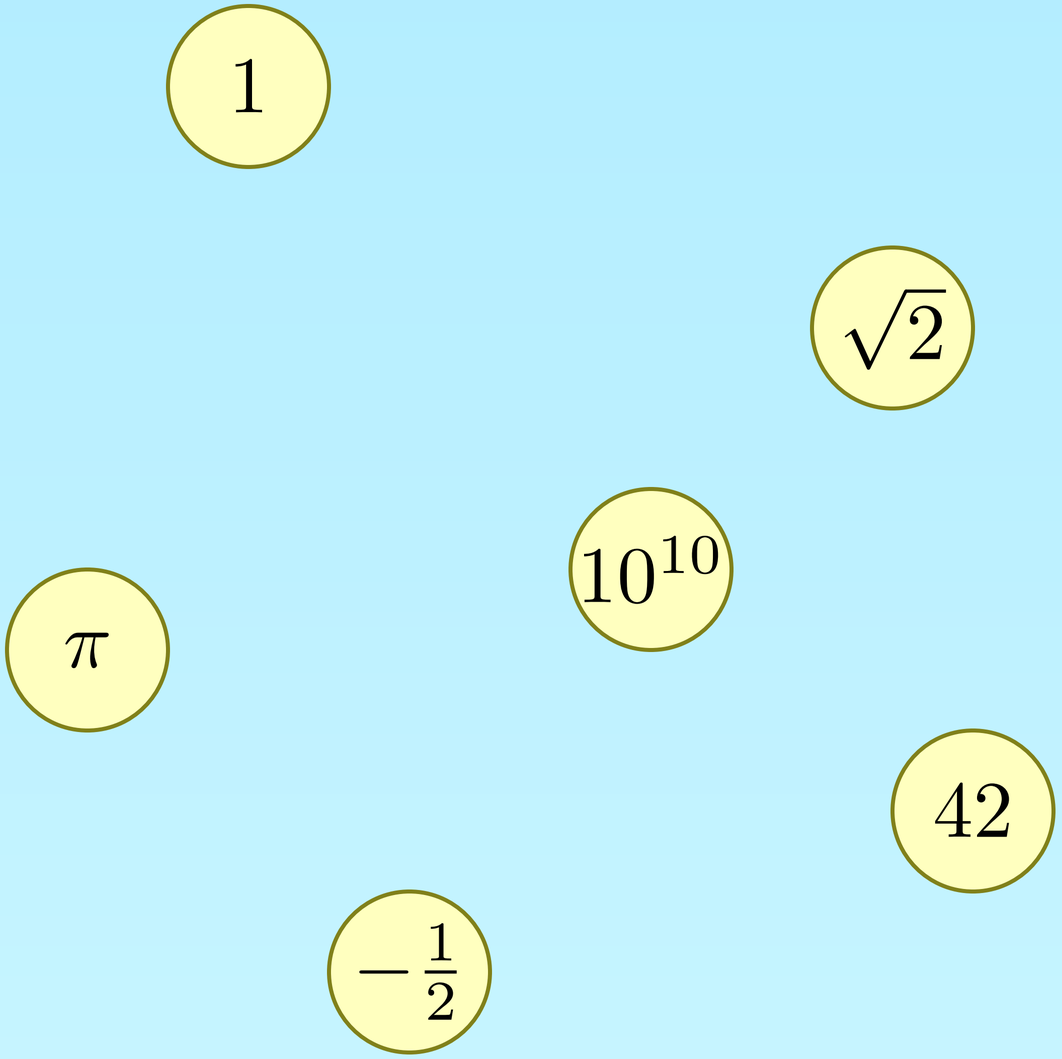
Data is stored in *cells*.



A *pointer machine* models how a computer can access and manipulate *data*.

Data is stored in *cells*.

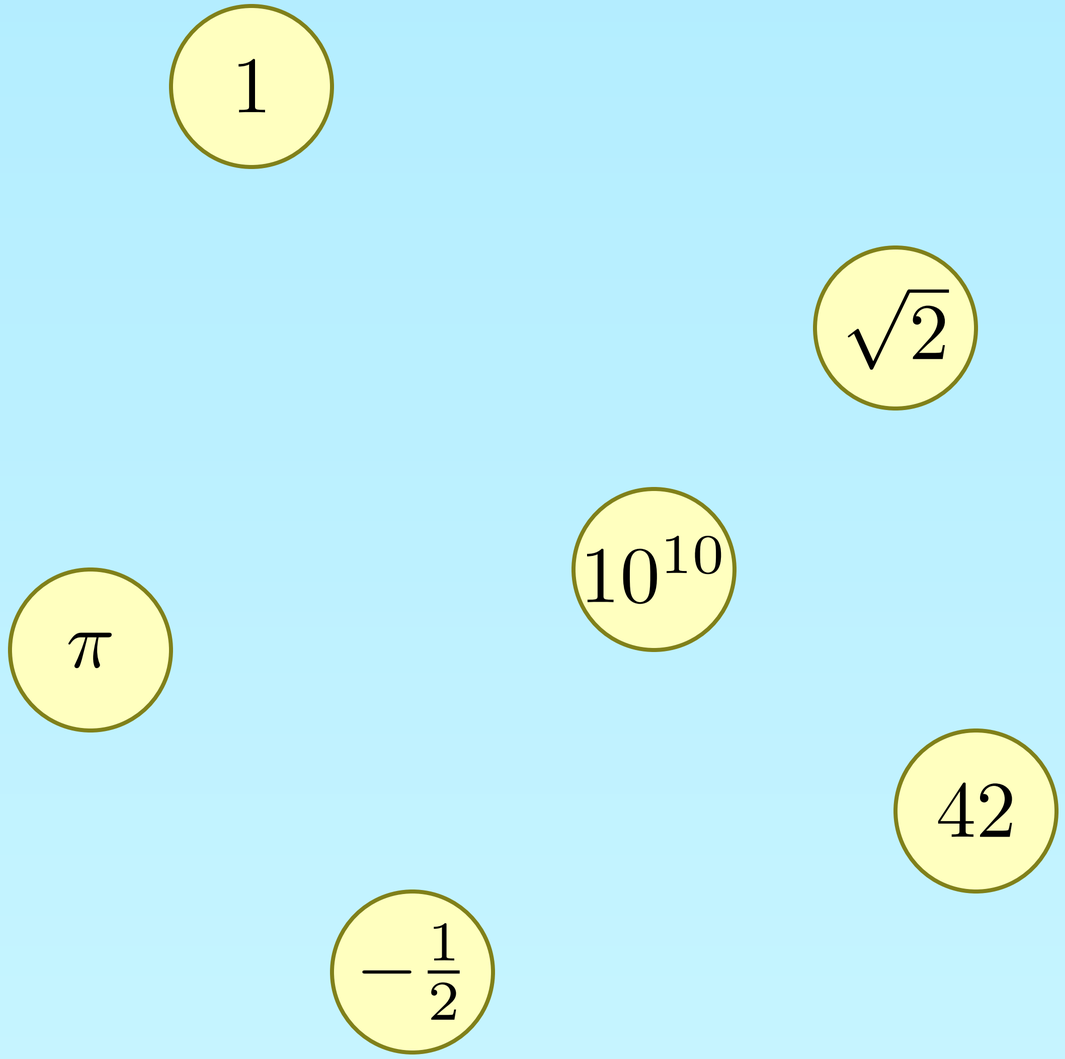
On a real-valued pointer machine, each cell can store any *real number*.




A *pointer machine* models how a computer can access and manipulate *data*.

Data is stored in *cells*.

On a real-valued pointer machine, each cell can store any *real number*.

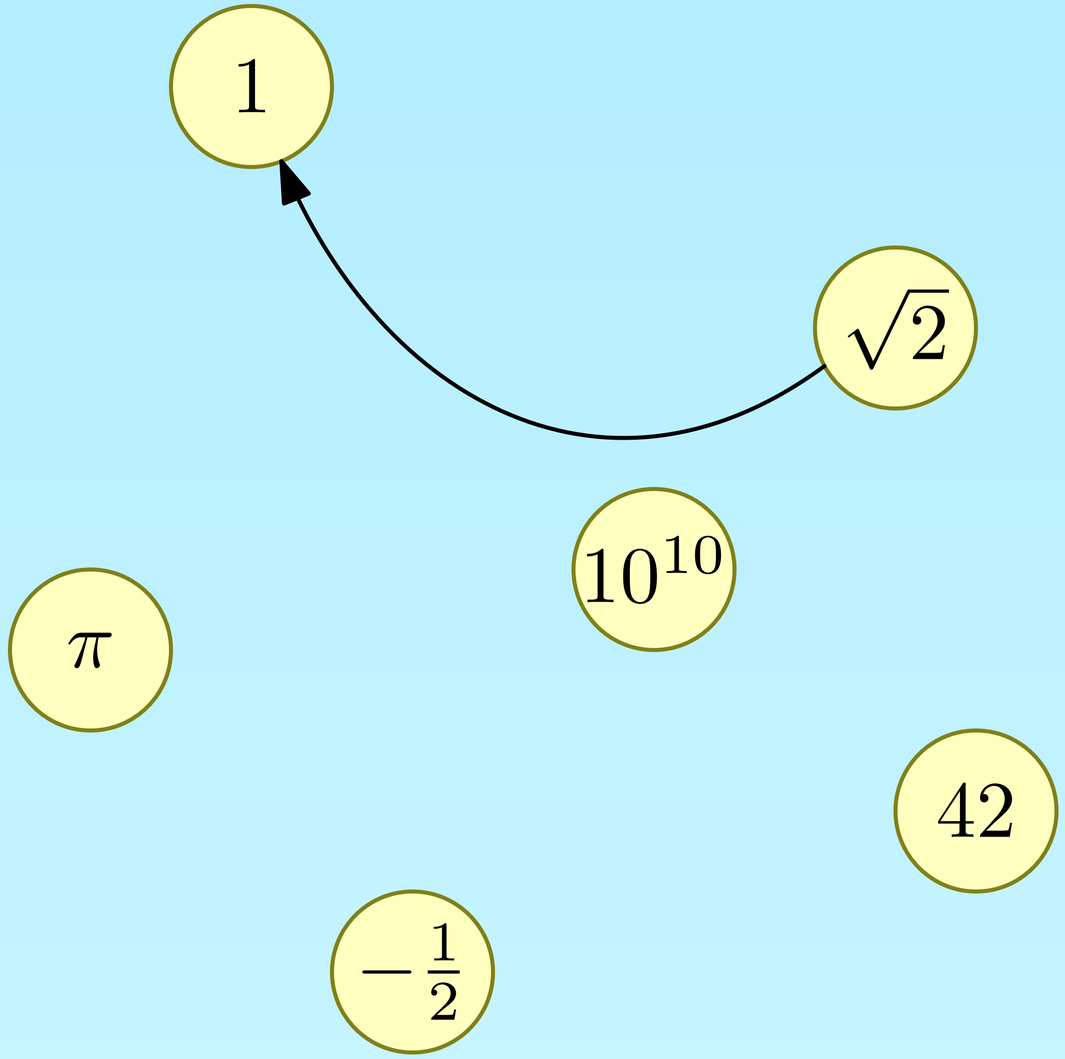



A *pointer machine* models how a computer can access and manipulate *data*.

Data is stored in *cells*.

On a real-valued pointer machine, each cell can store any *real number*.

In addition, each cell can have $O(1)$ *pointers* to other cells.

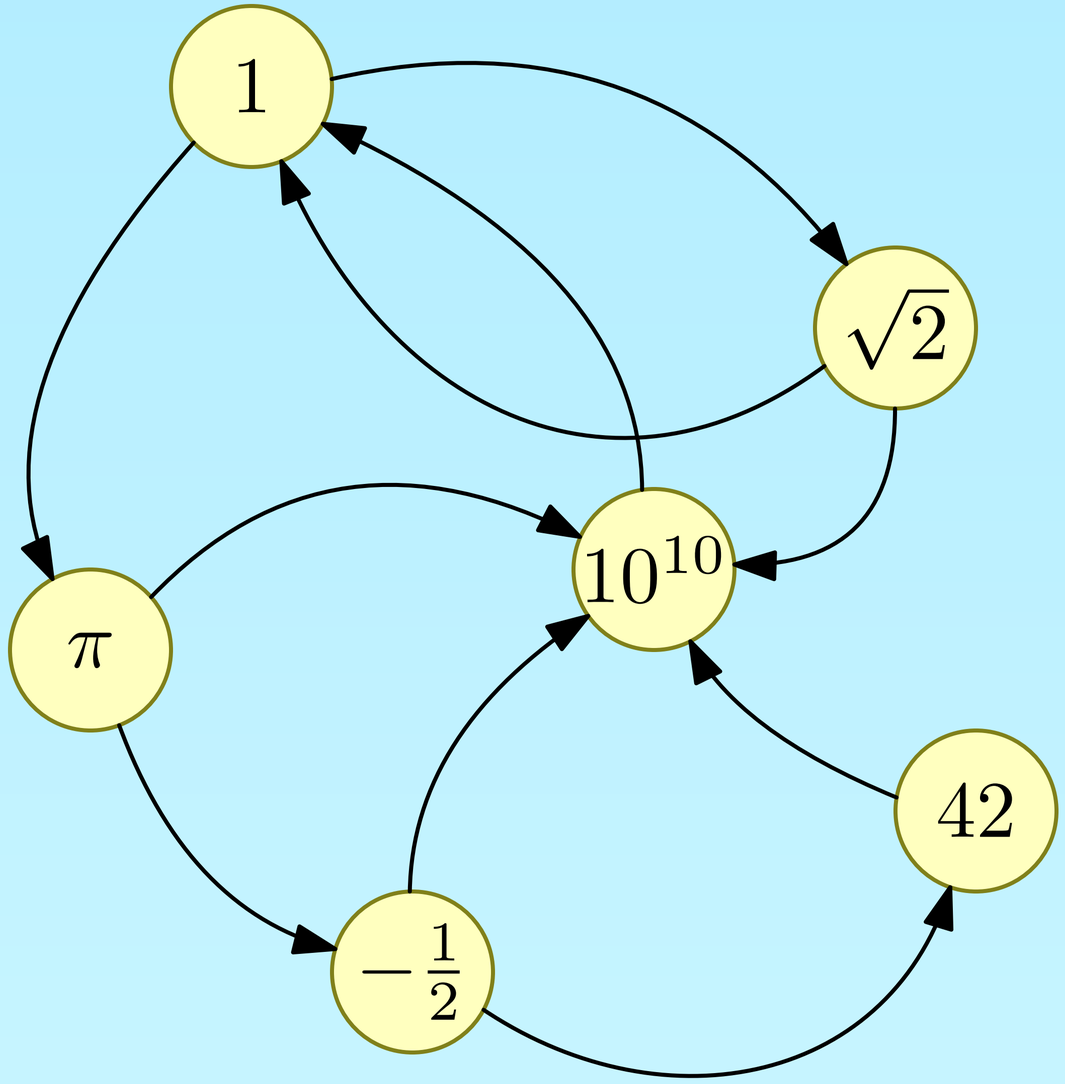



A *pointer machine* models how a computer can access and manipulate *data*.

Data is stored in *cells*.

On a real-valued pointer machine, each cell can store any *real number*.

In addition, each cell can have $O(1)$ *pointers* to other cells.

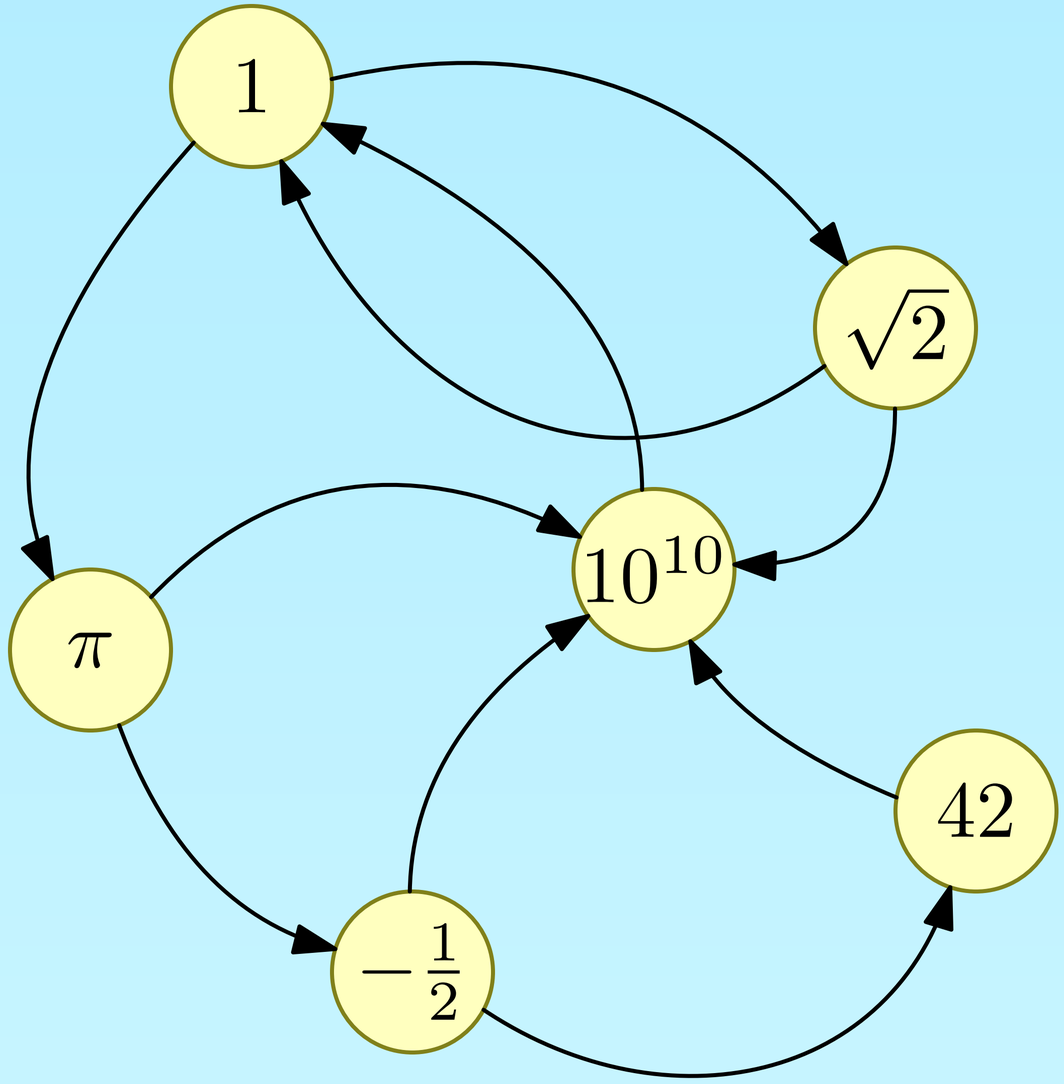



A *pointer machine* models how a computer can access and manipulate *data*.

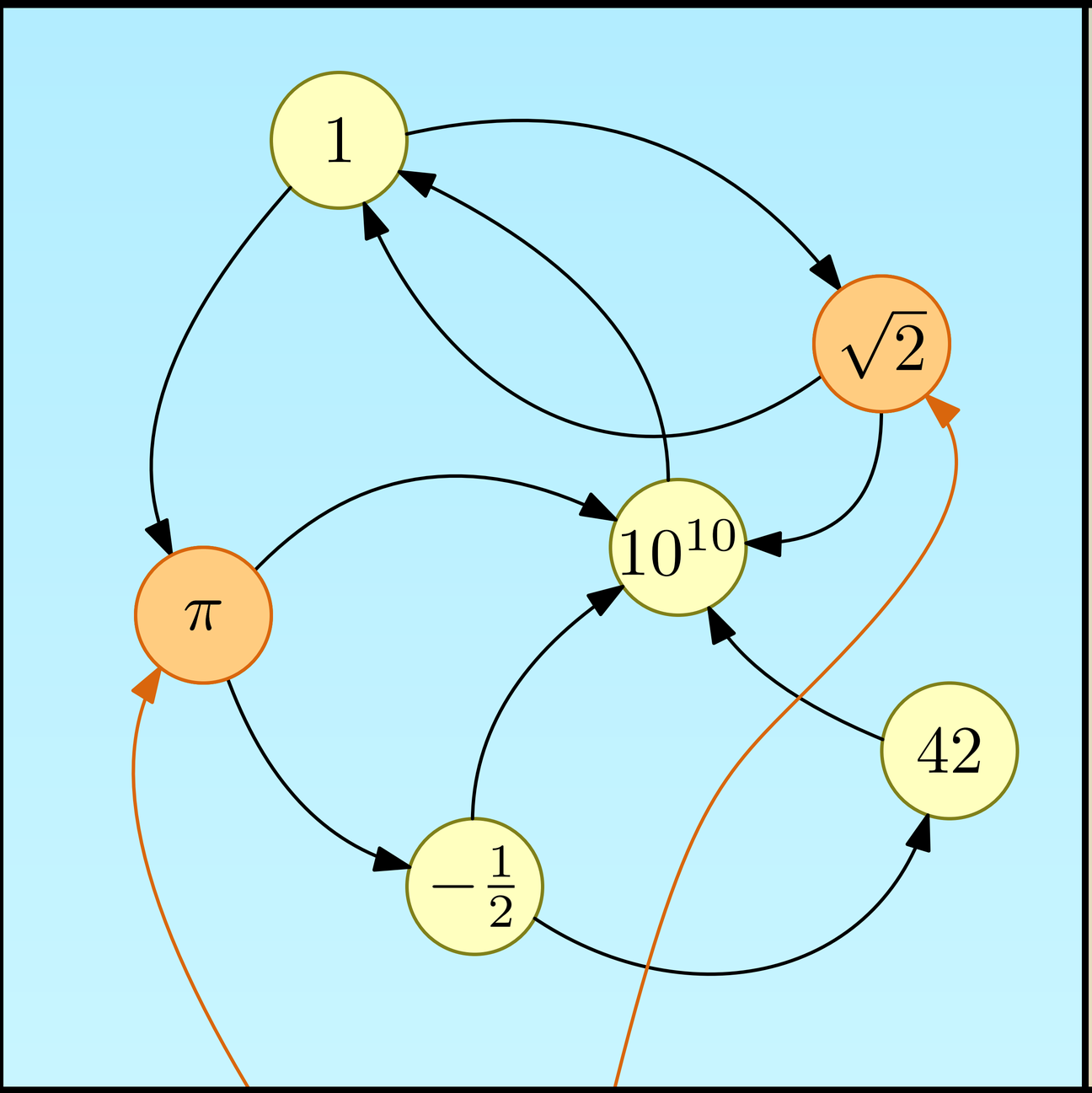
Data is stored in *cells*.

On a real-valued pointer machine, each cell can store any *real number*.

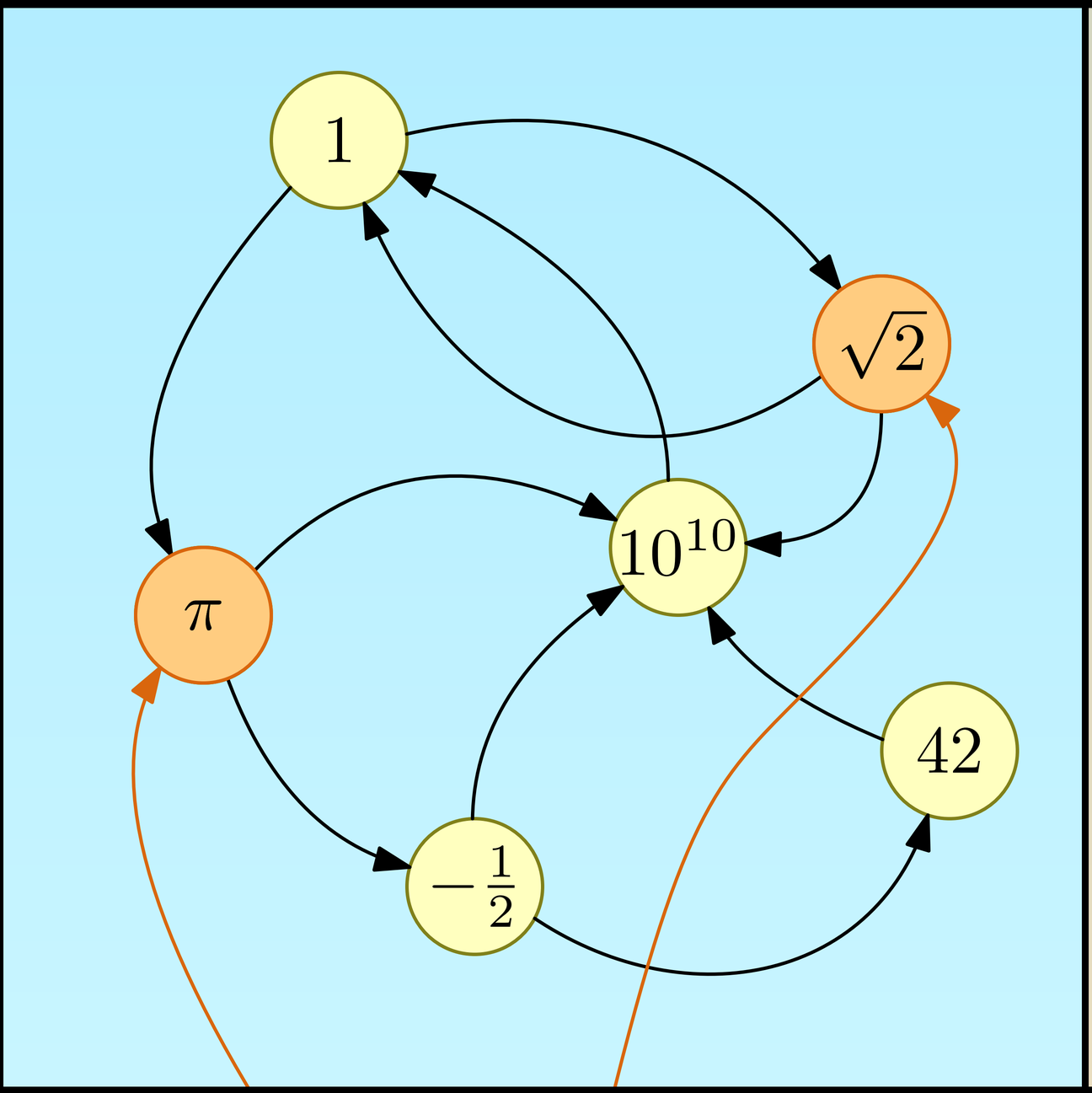
In addition, each cell can have $O(1)$ *pointers* to other cells.




The computer also has a *working memory* which can hold $O(1)$ real numbers and $O(1)$ pointers.

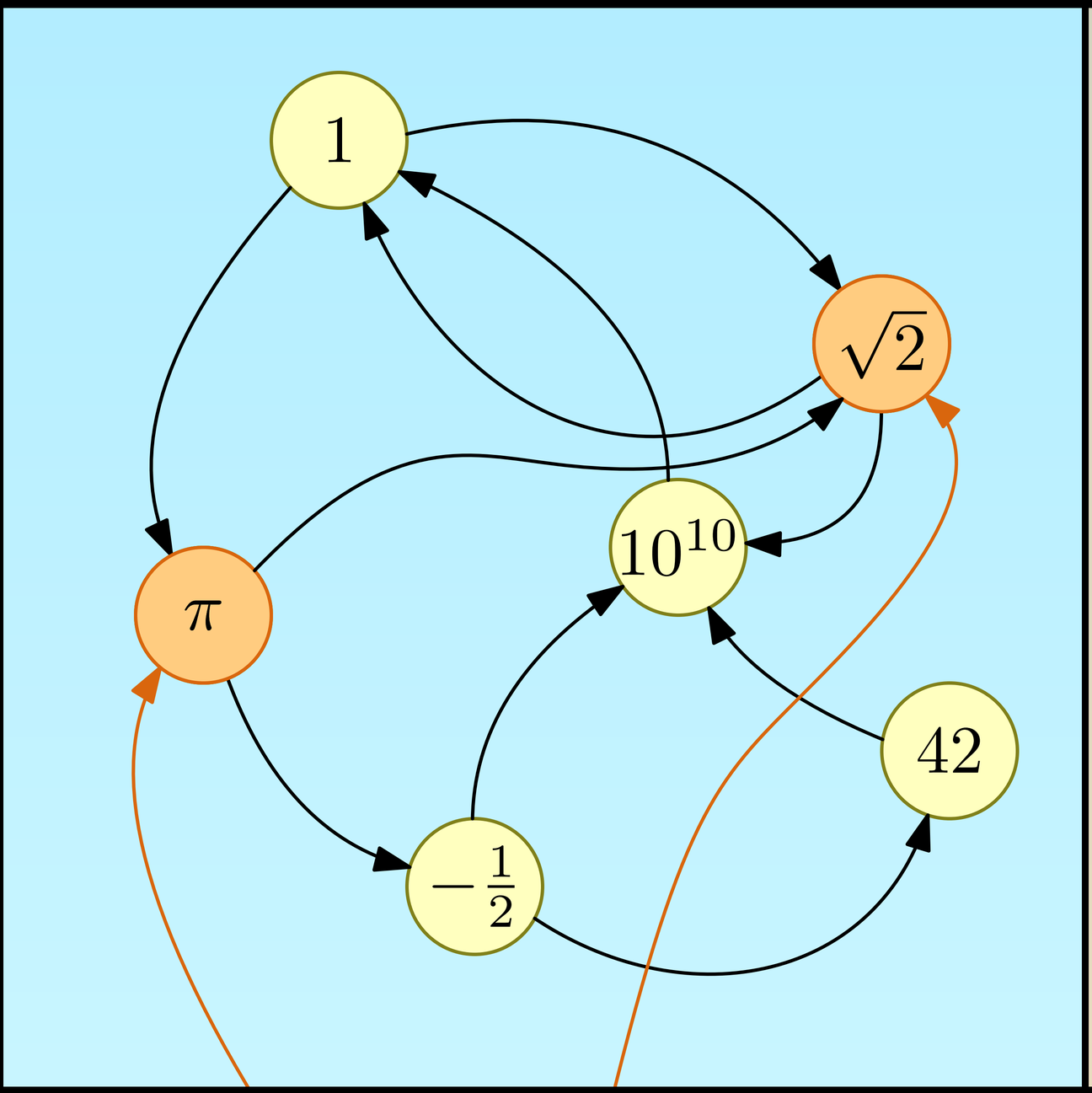



The computer also has a *working memory* which can hold $O(1)$ real numbers and $O(1)$ pointers.



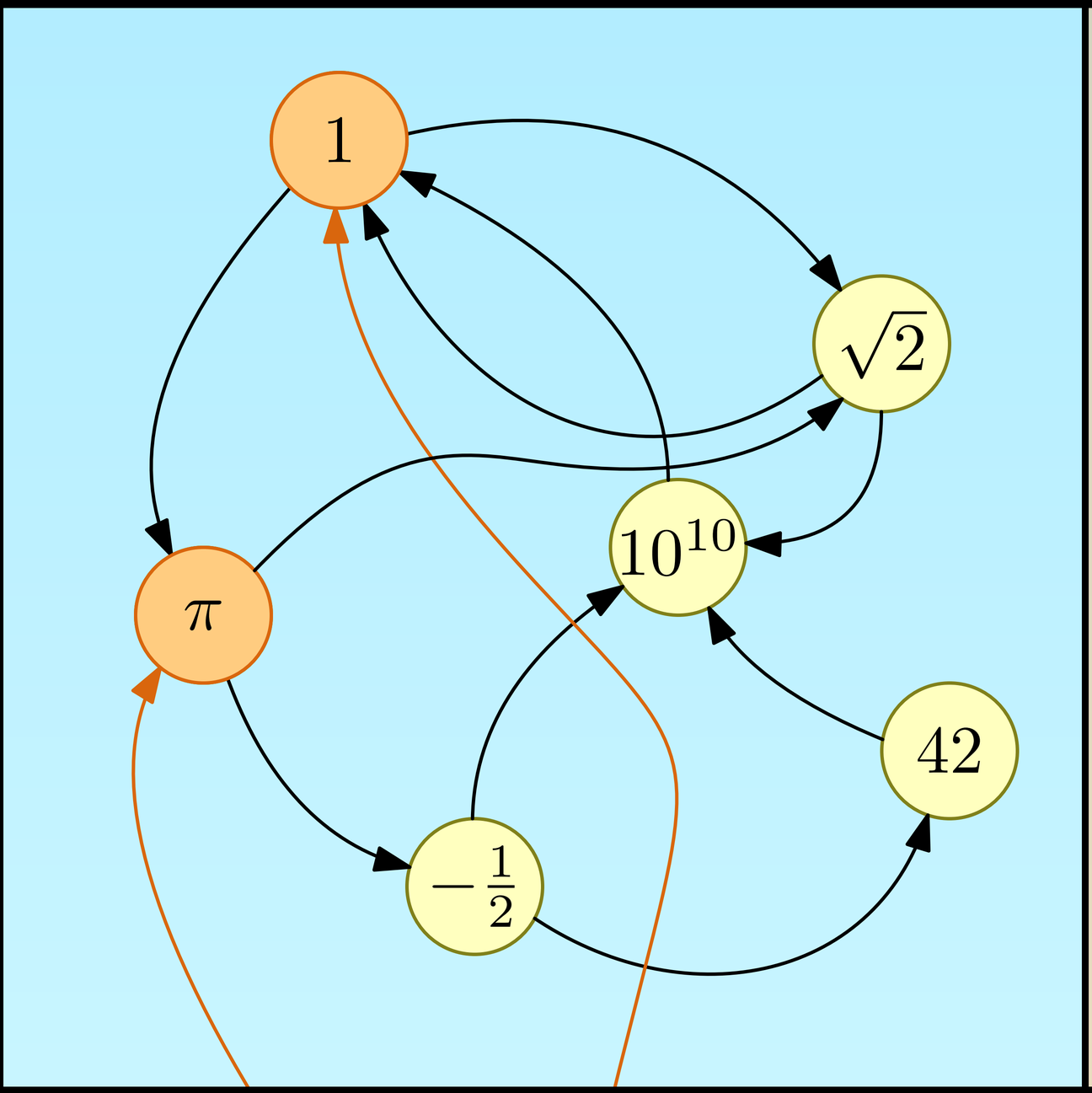

The computer also has a *working memory* which can hold $O(1)$ real numbers and $O(1)$ pointers.

An algorithm can follow or duplicate pointers, create new cells, and manipulate real numbers.



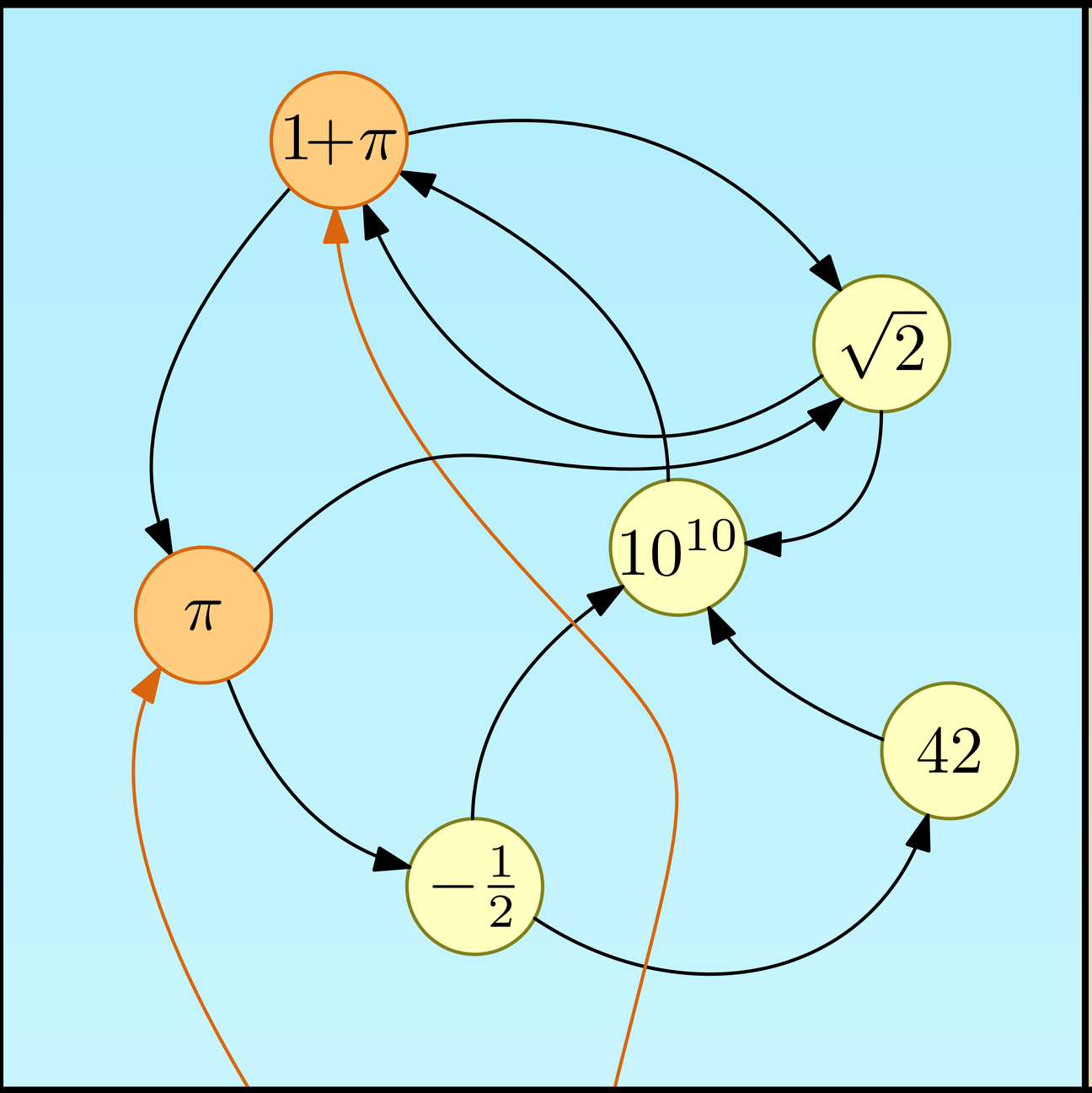

The computer also has a *working memory* which can hold $O(1)$ real numbers and $O(1)$ pointers.

An algorithm can follow or duplicate pointers, create new cells, and manipulate real numbers.



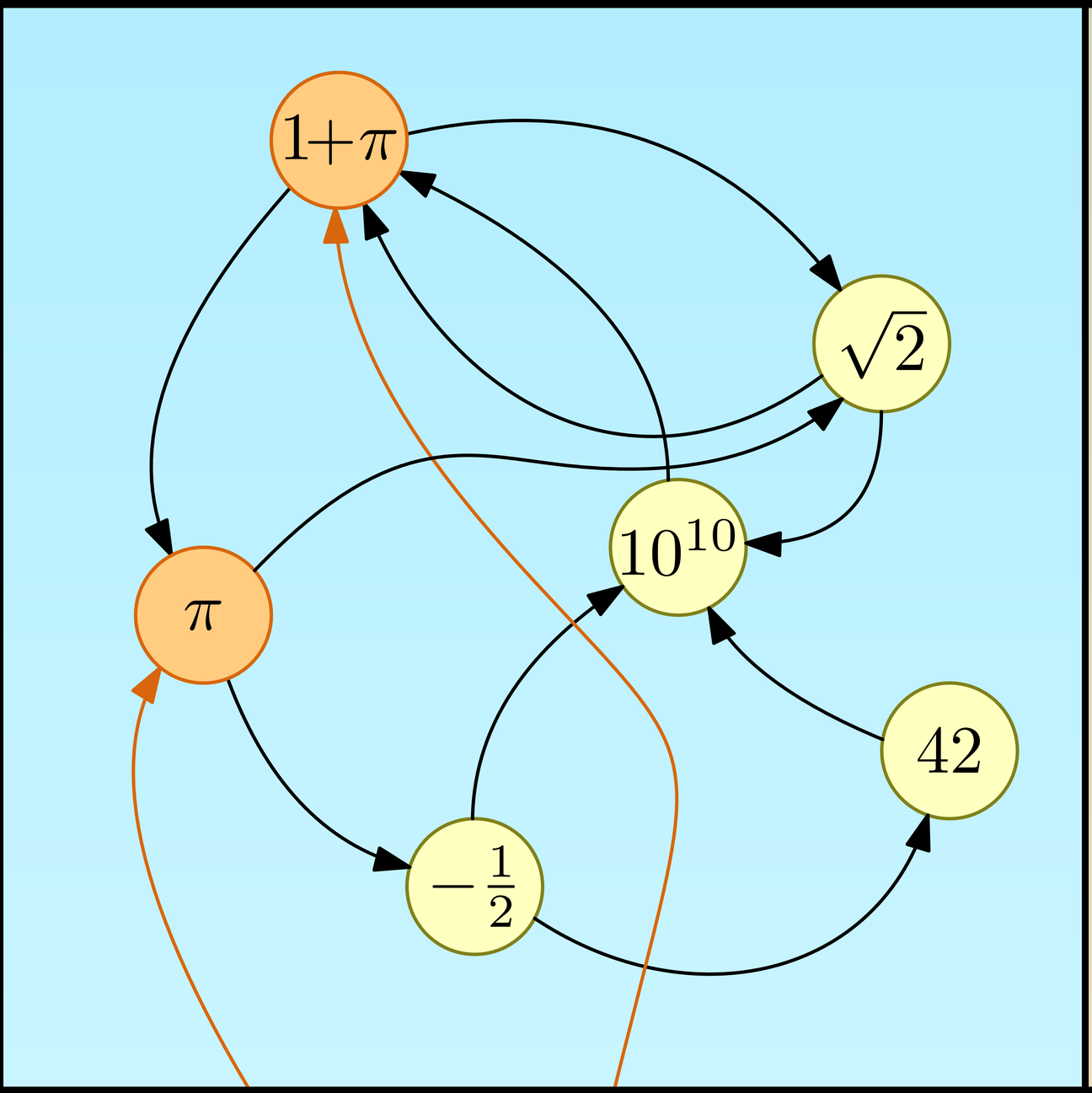

The computer also has a *working memory* which can hold $O(1)$ real numbers and $O(1)$ pointers.

An algorithm can follow or duplicate pointers, create new cells, and manipulate real numbers.




The computer also has a *working memory* which can hold $O(1)$ real numbers and $O(1)$ pointers.

An algorithm can follow or duplicate pointers, create new cells, and manipulate real numbers.




The computer also has a *working memory* which can hold $O(1)$ real numbers and $O(1)$ pointers.

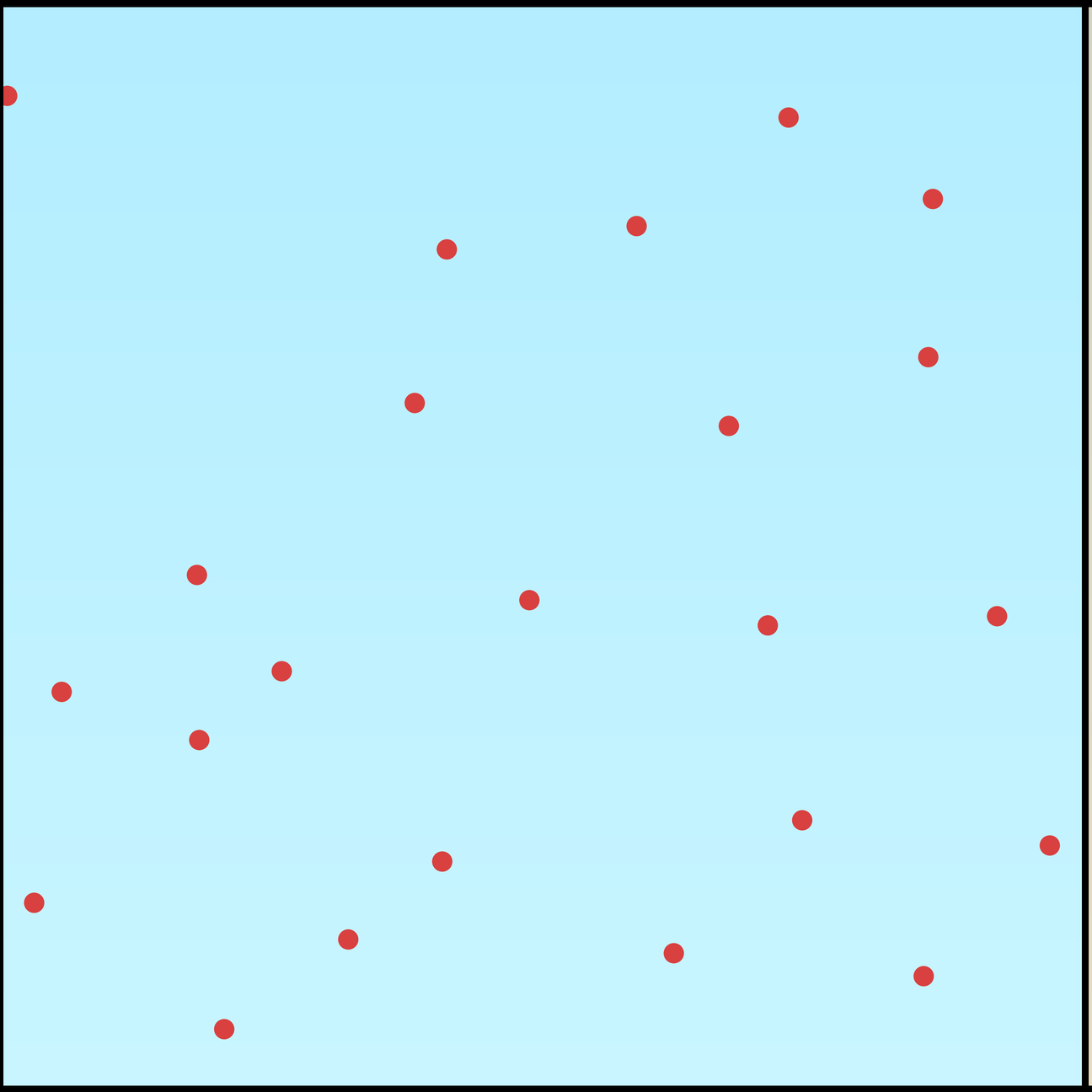
An algorithm can follow or duplicate pointers, create new cells, and manipulate real numbers.

In this model of computation, pointers are *not* numbers!

All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

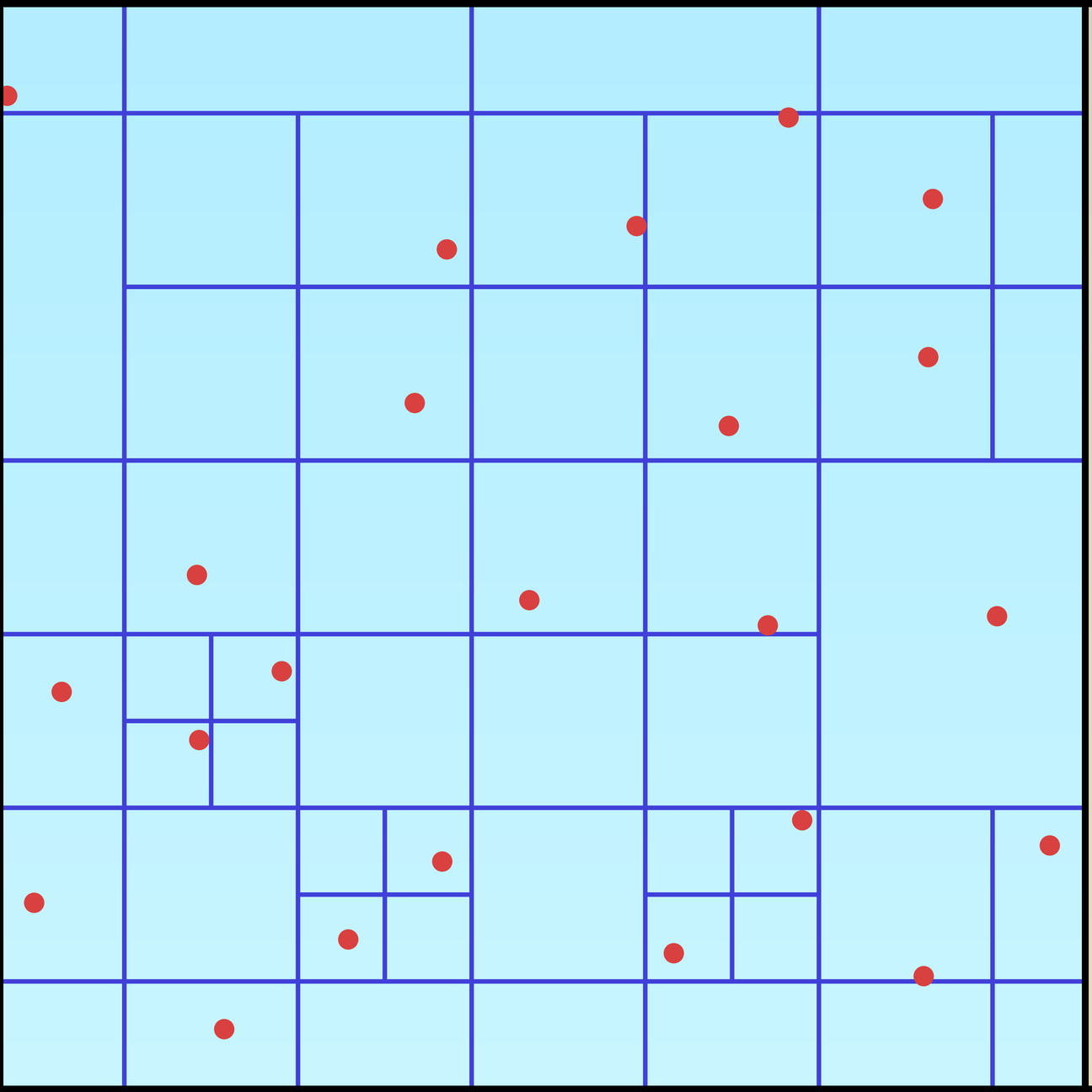
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



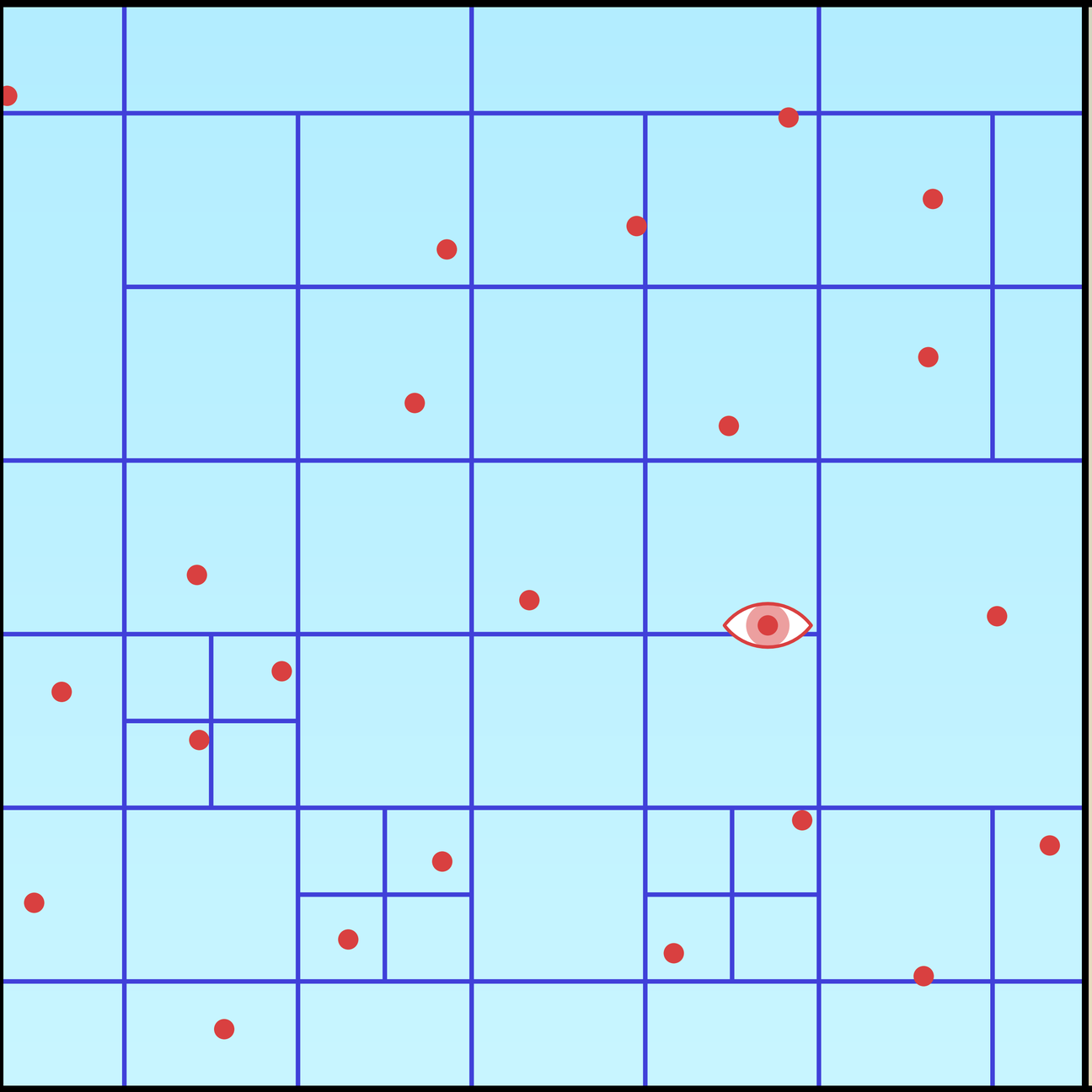
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



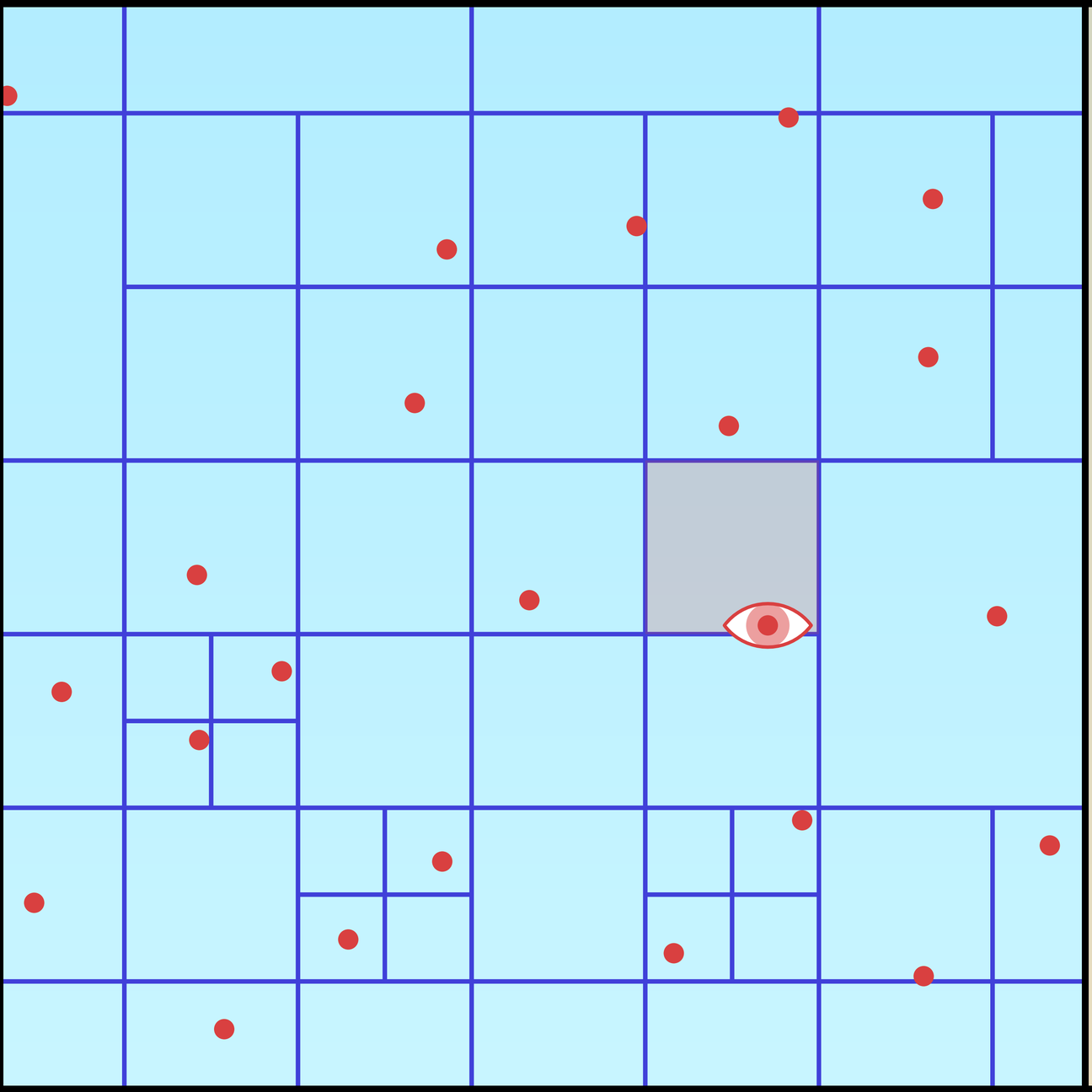
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



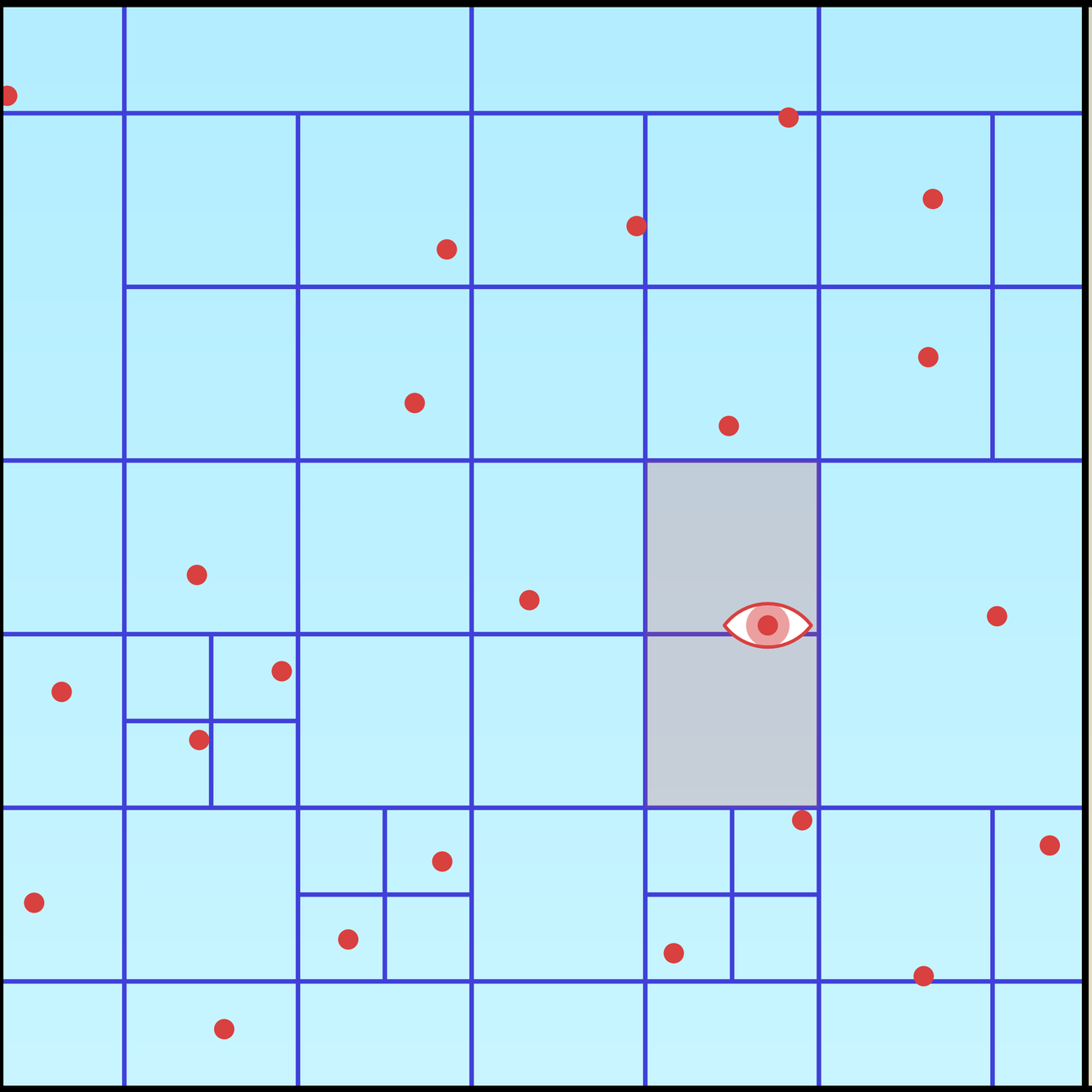
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



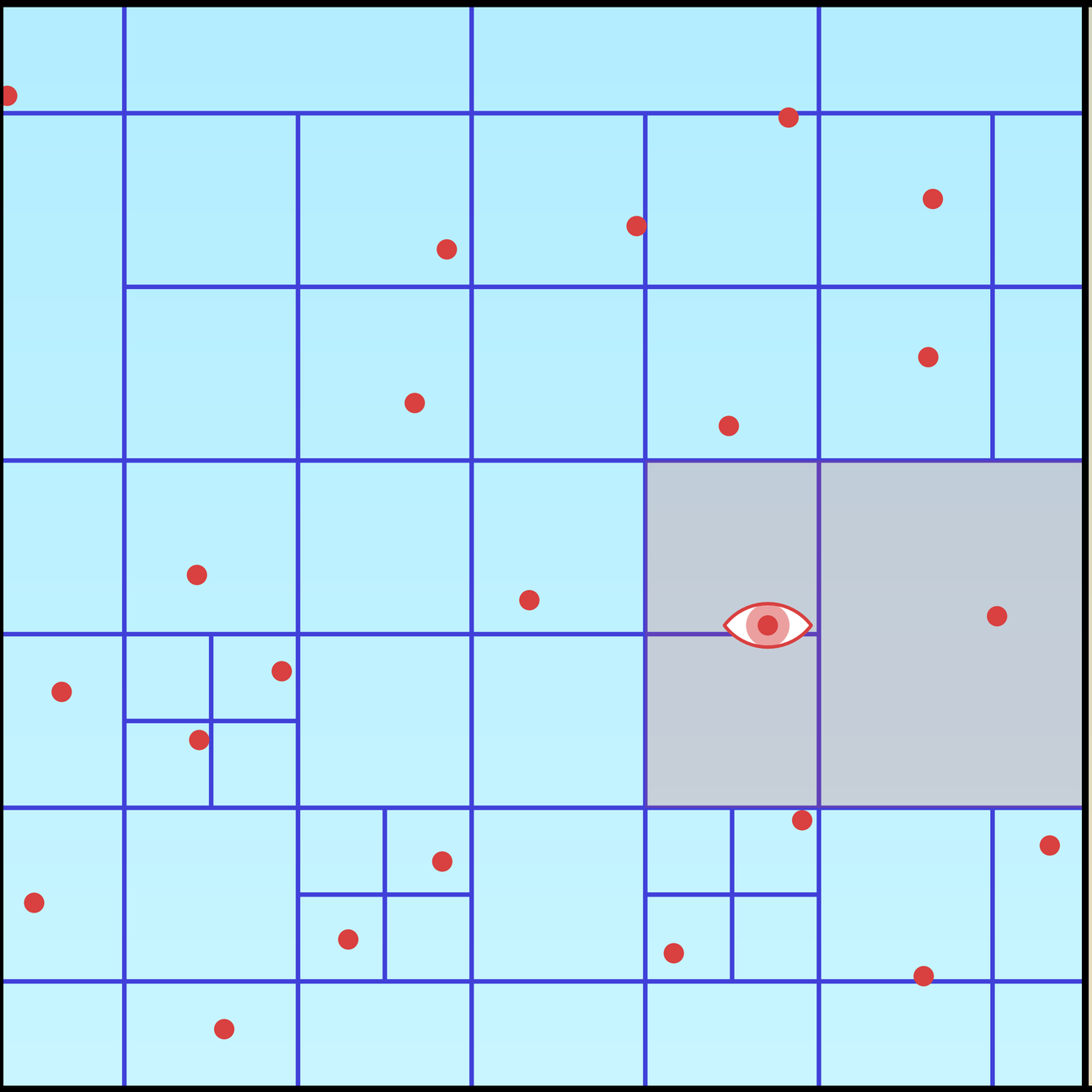
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



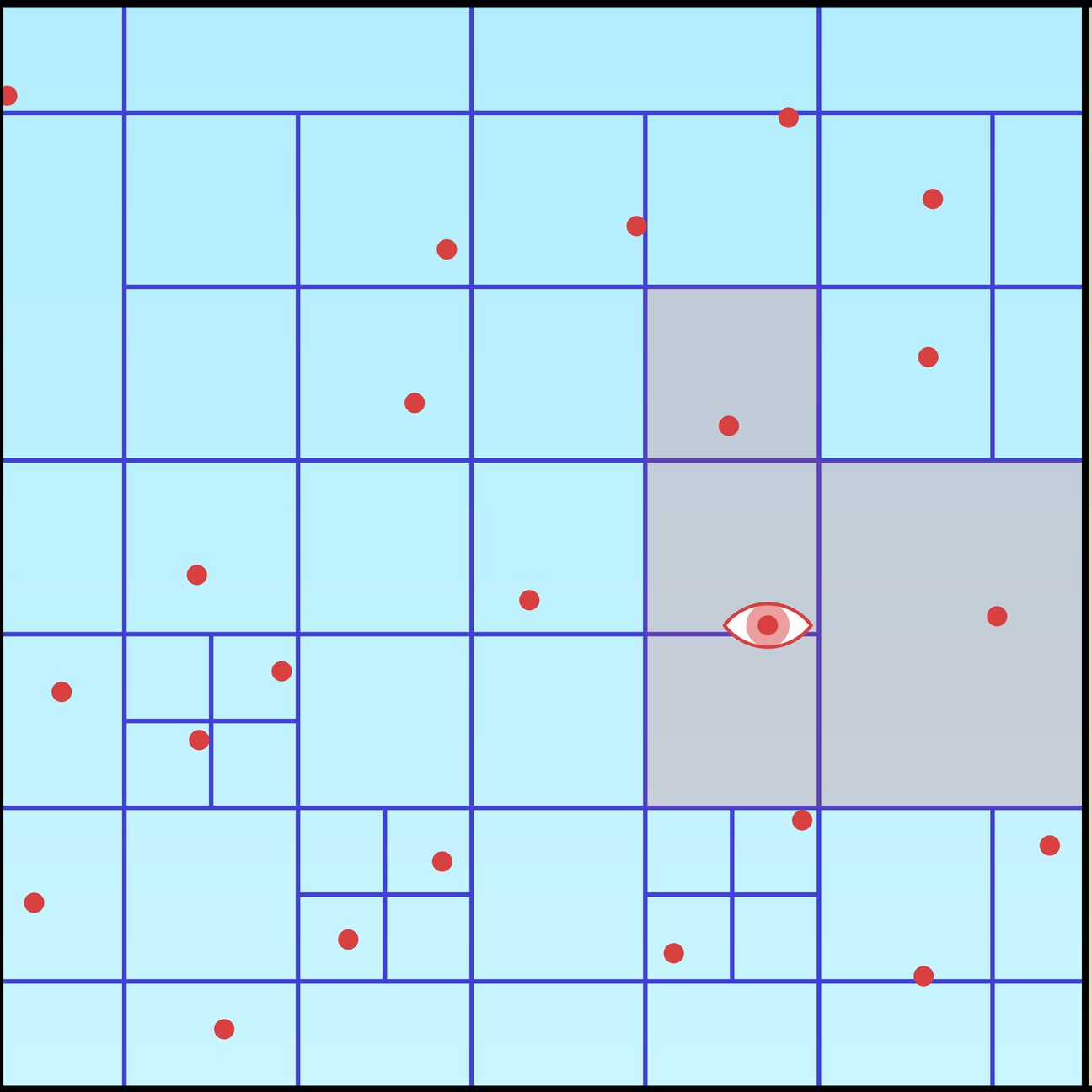
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



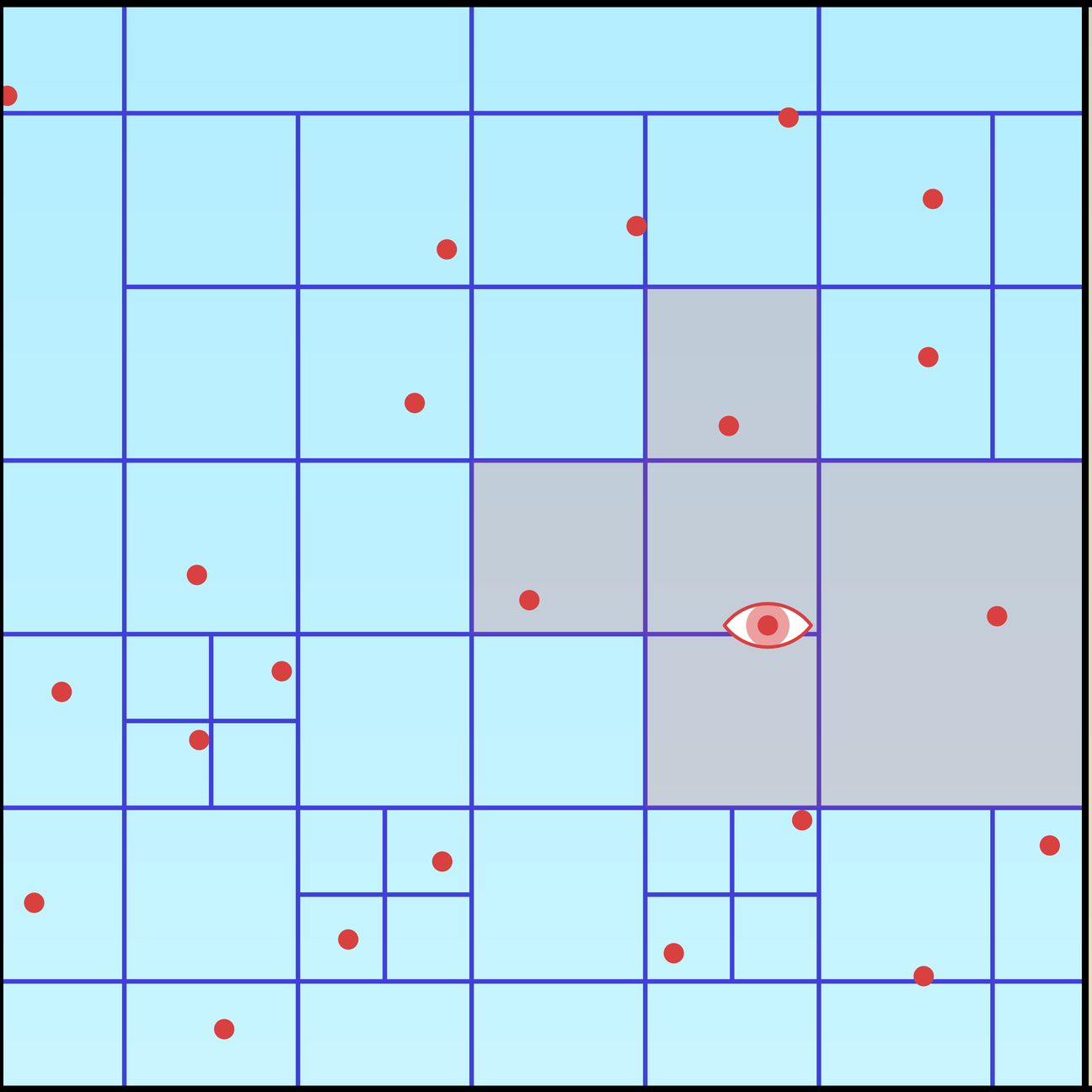
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



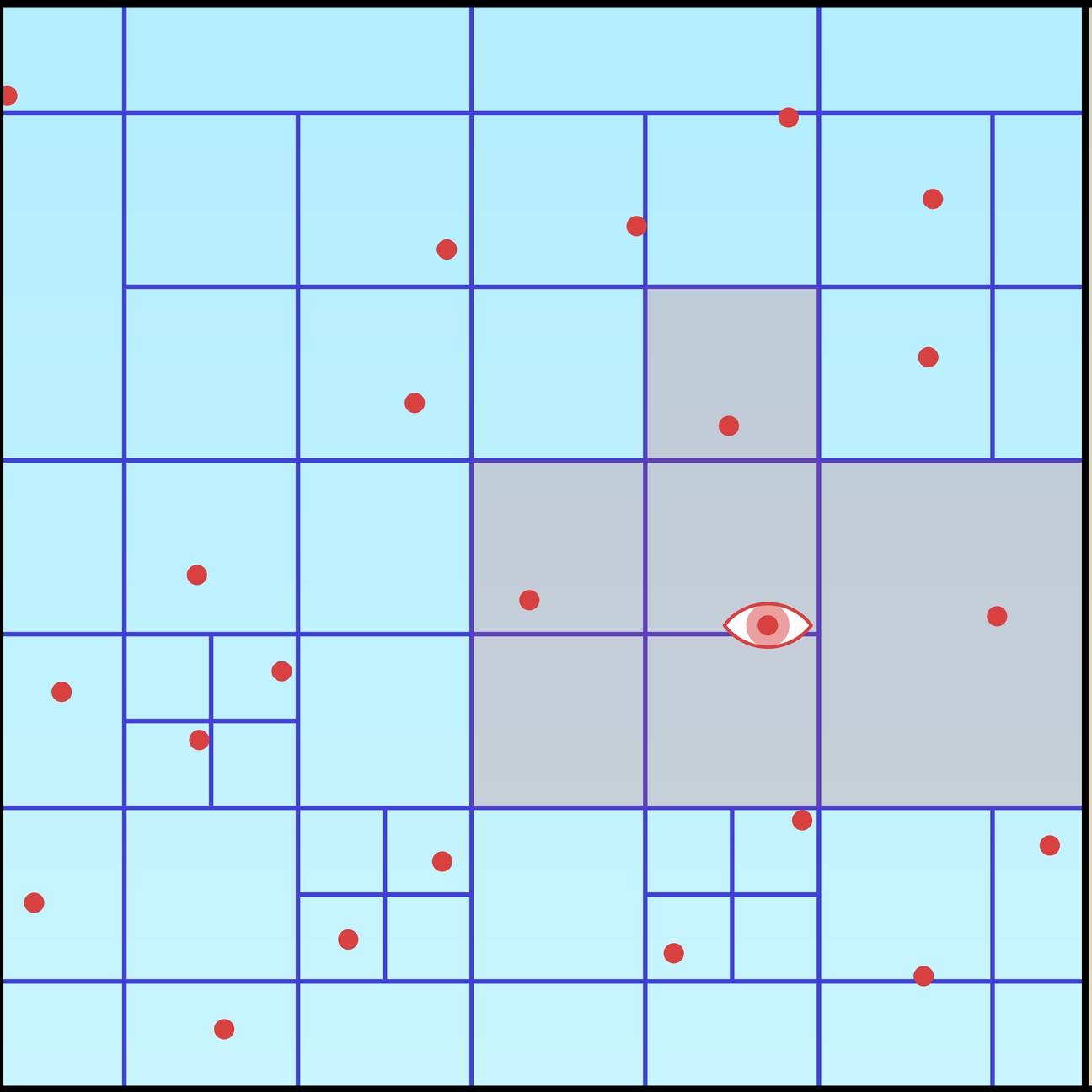
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



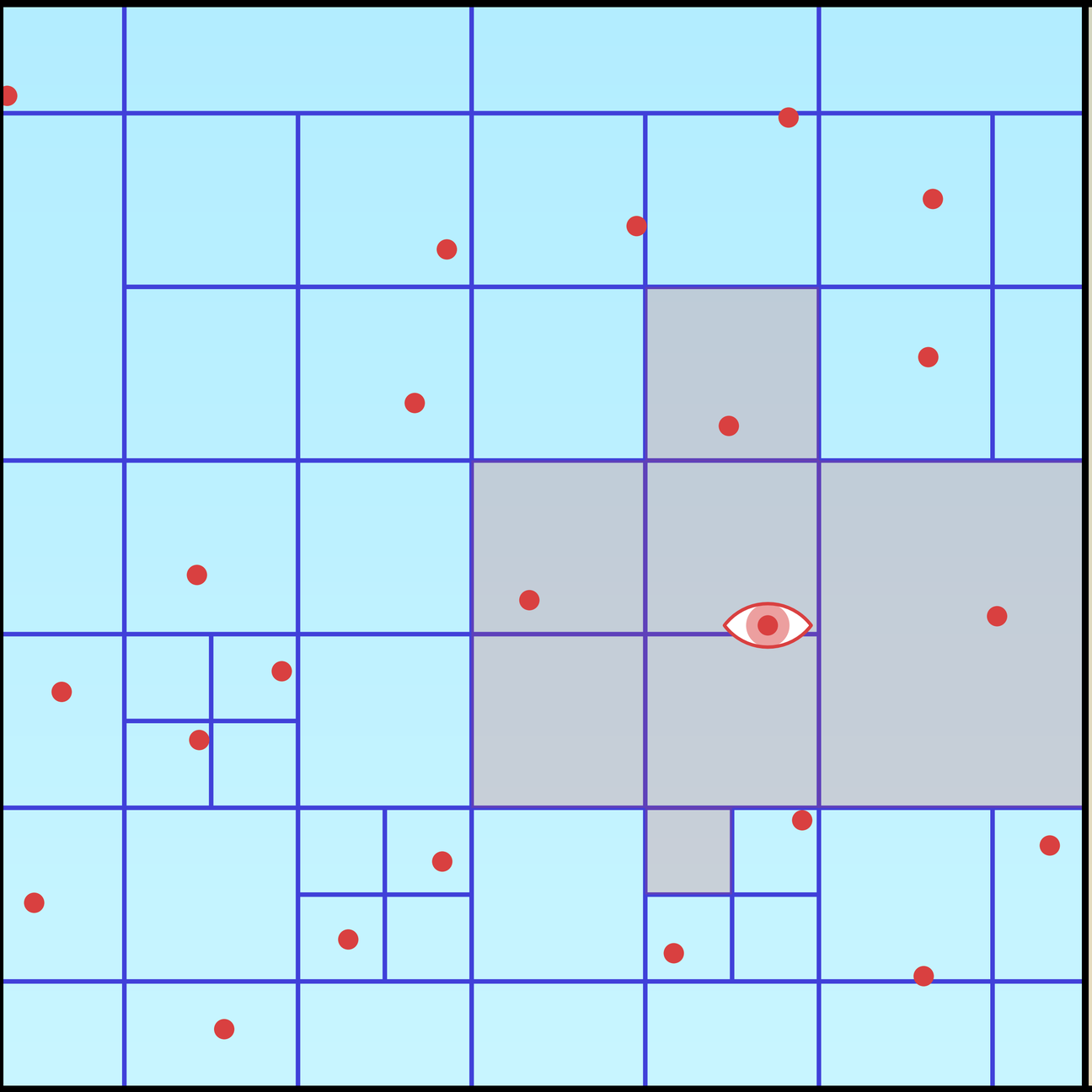
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



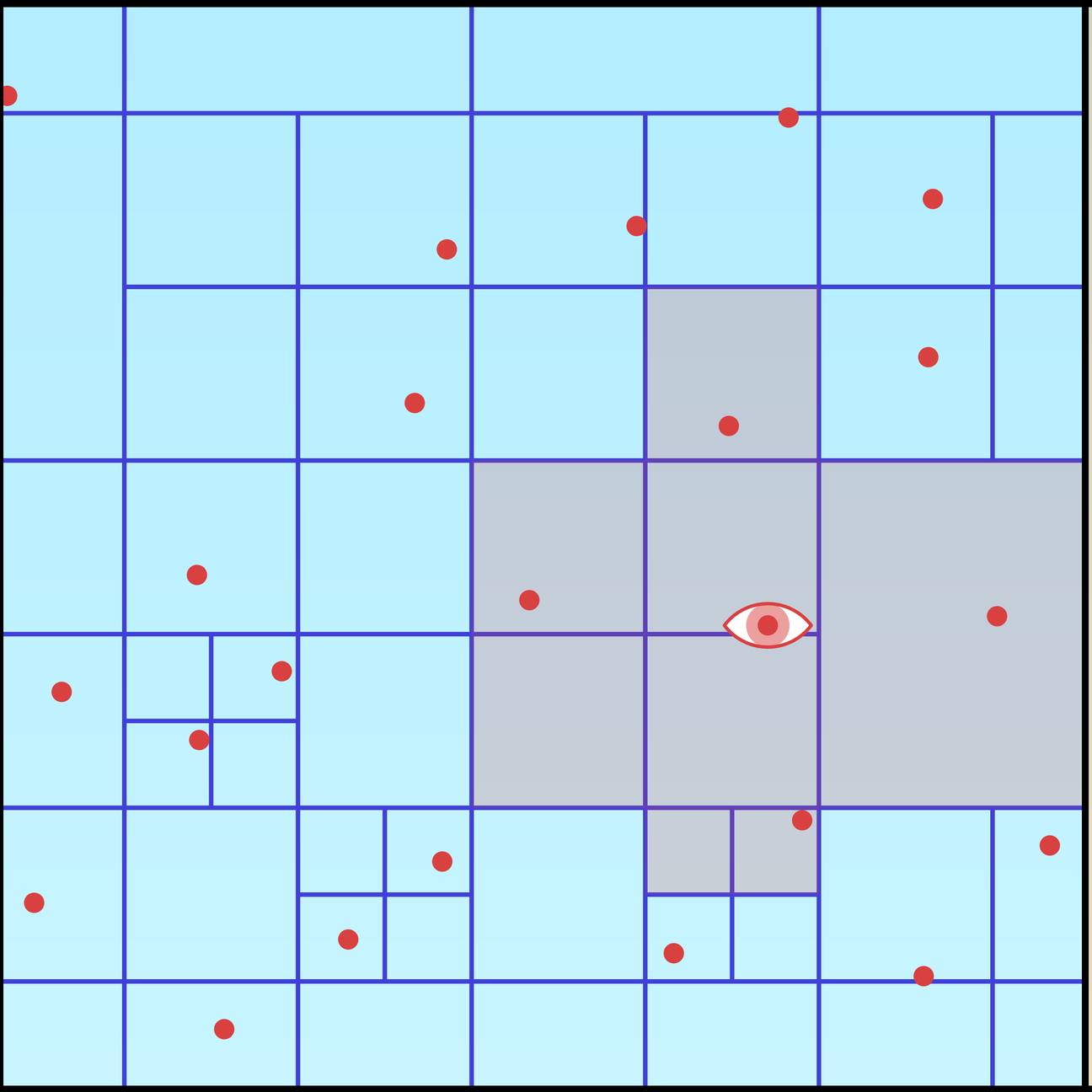
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



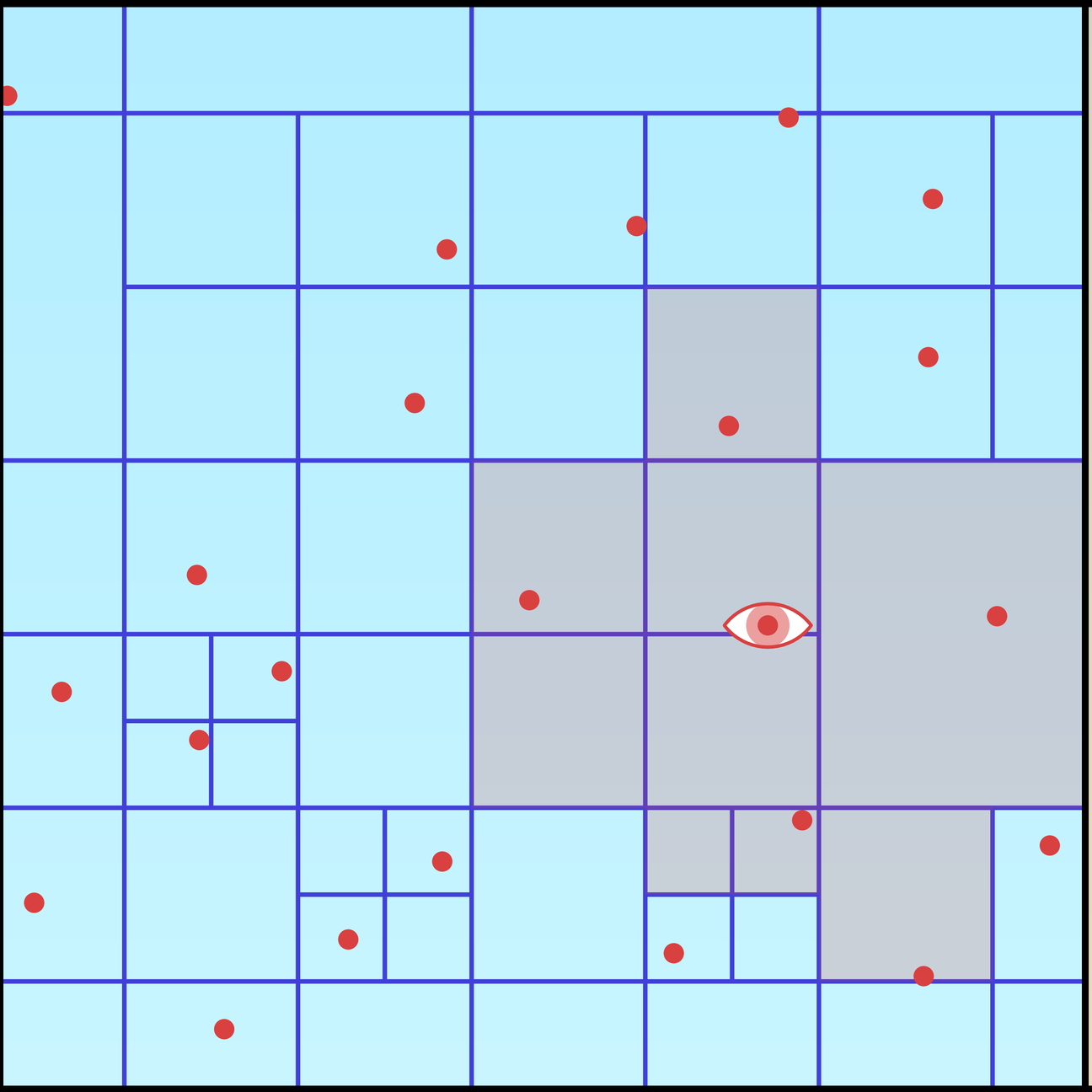
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



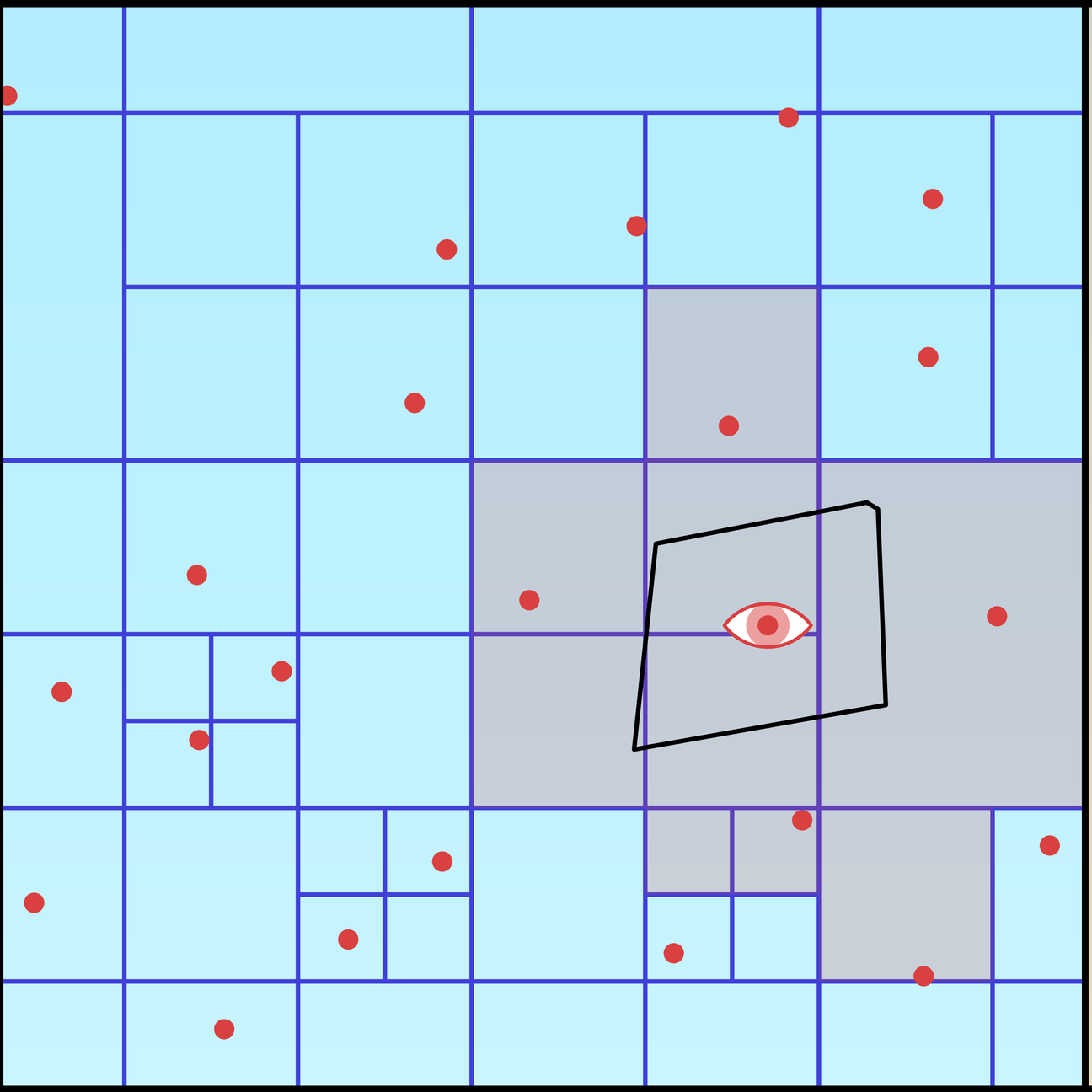
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



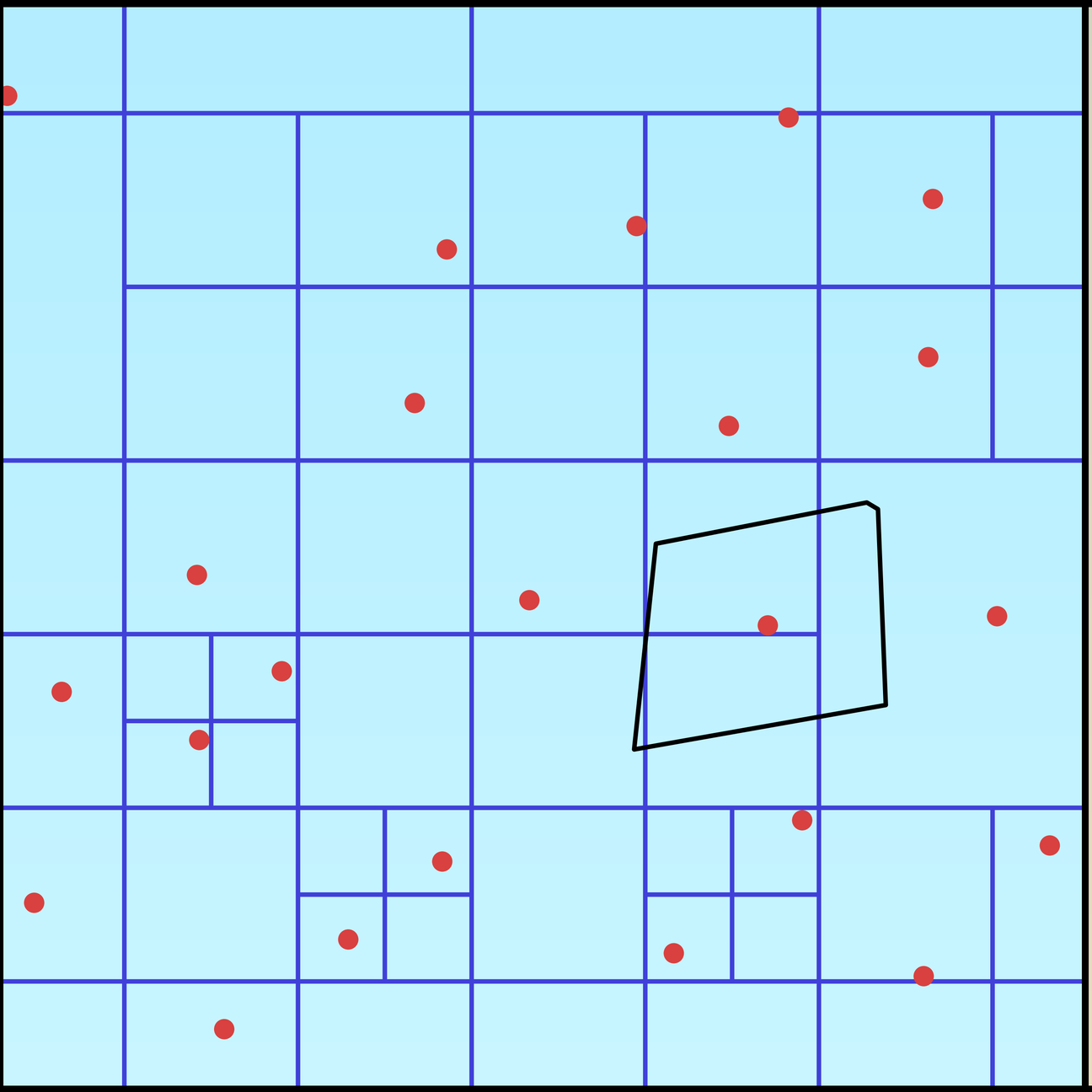
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



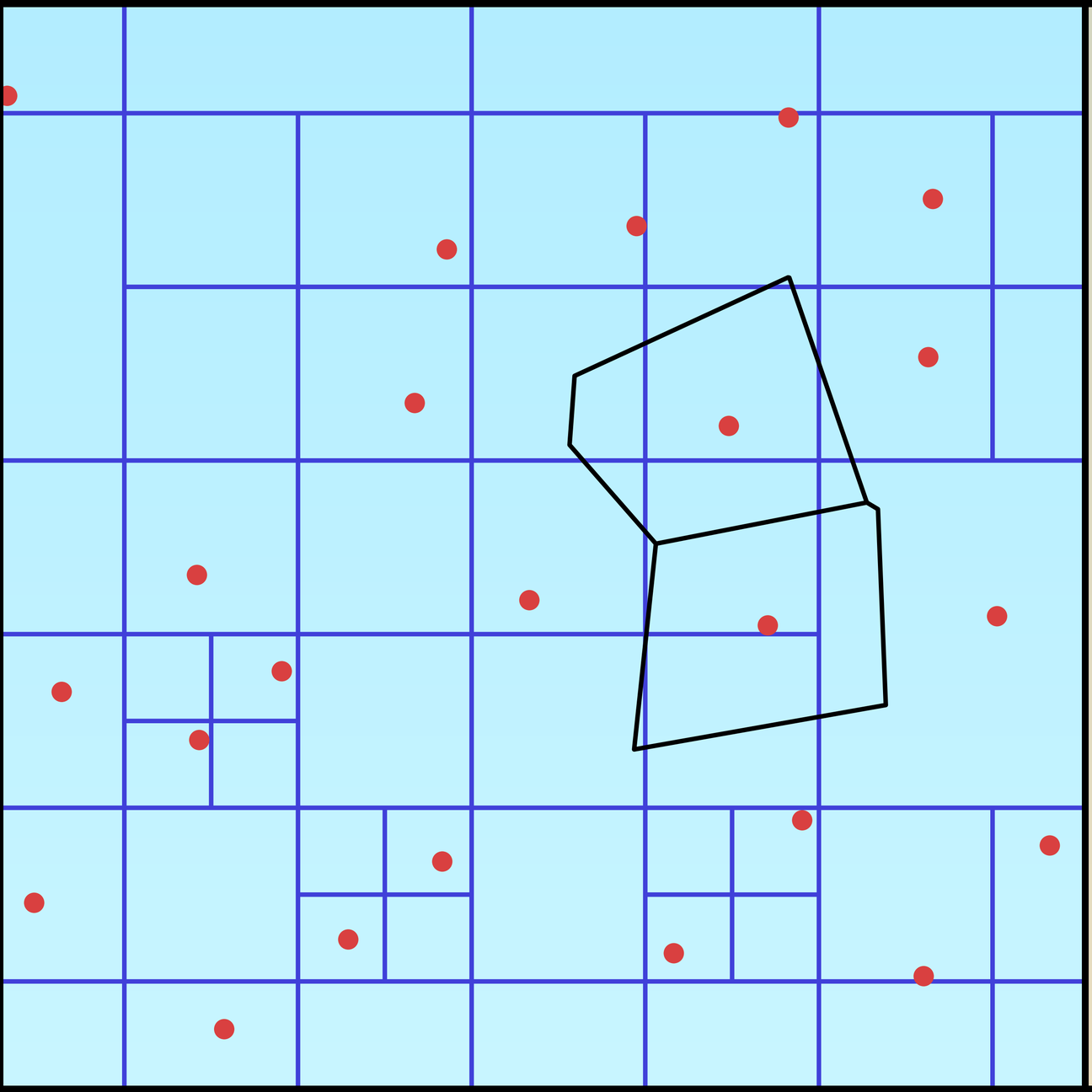
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



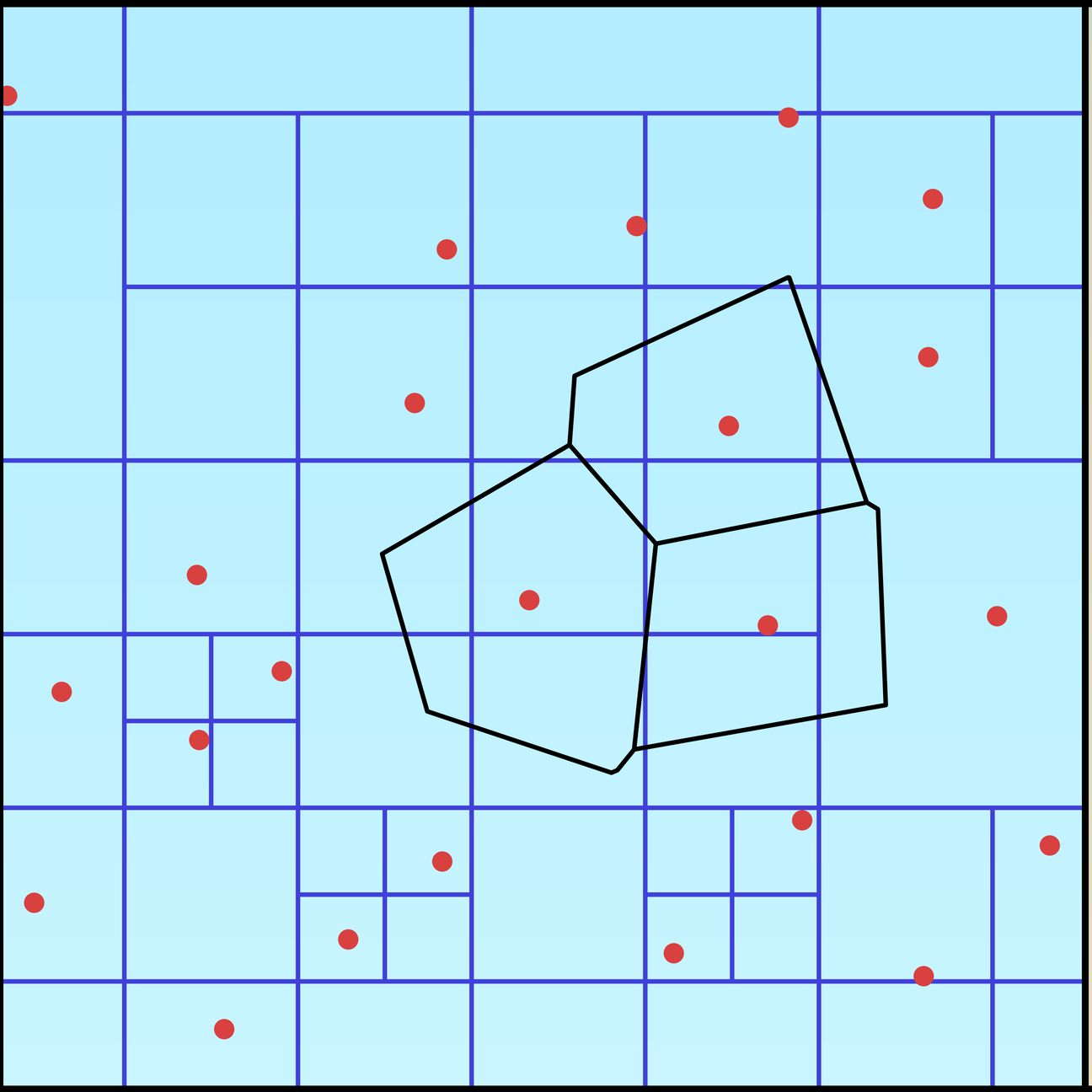
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



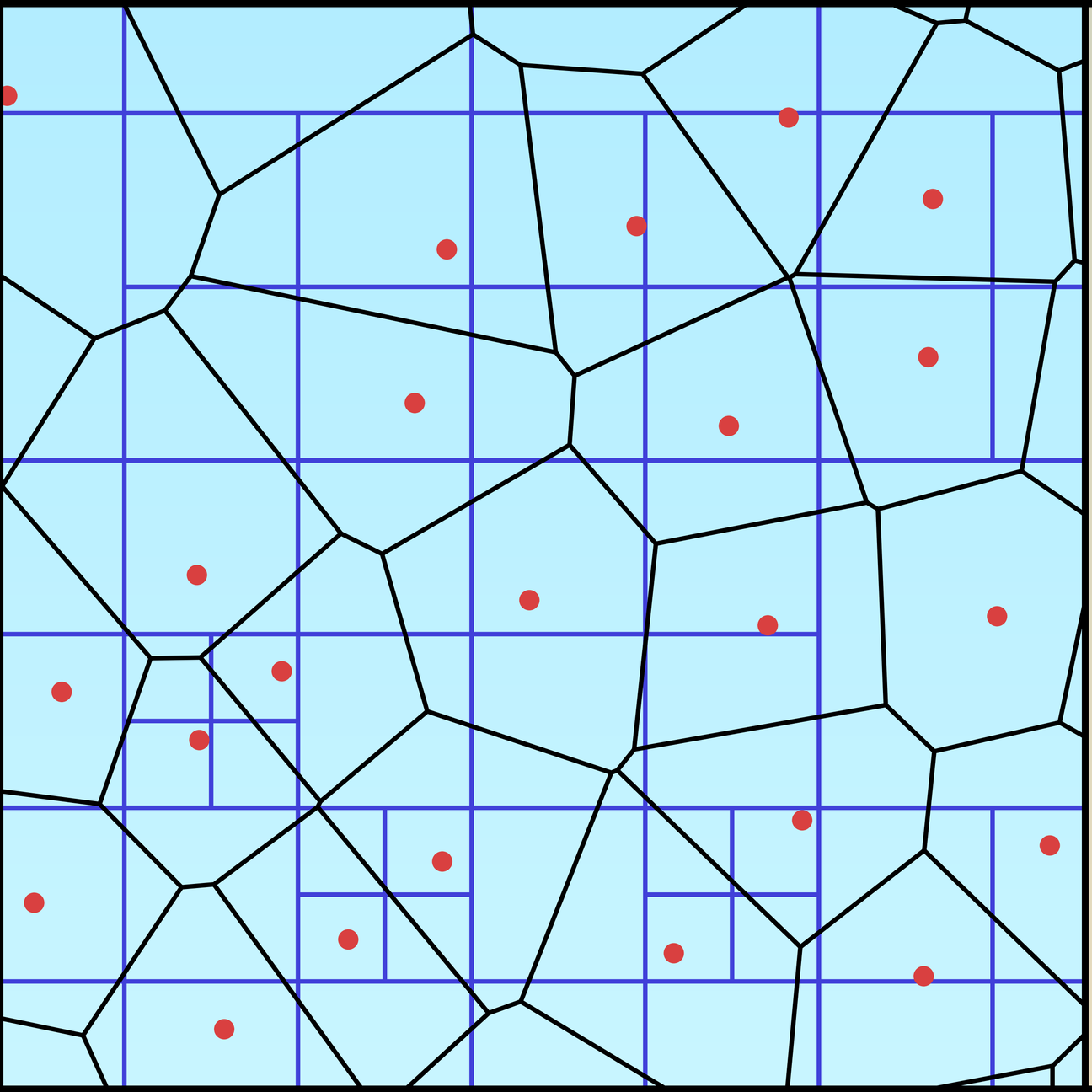
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



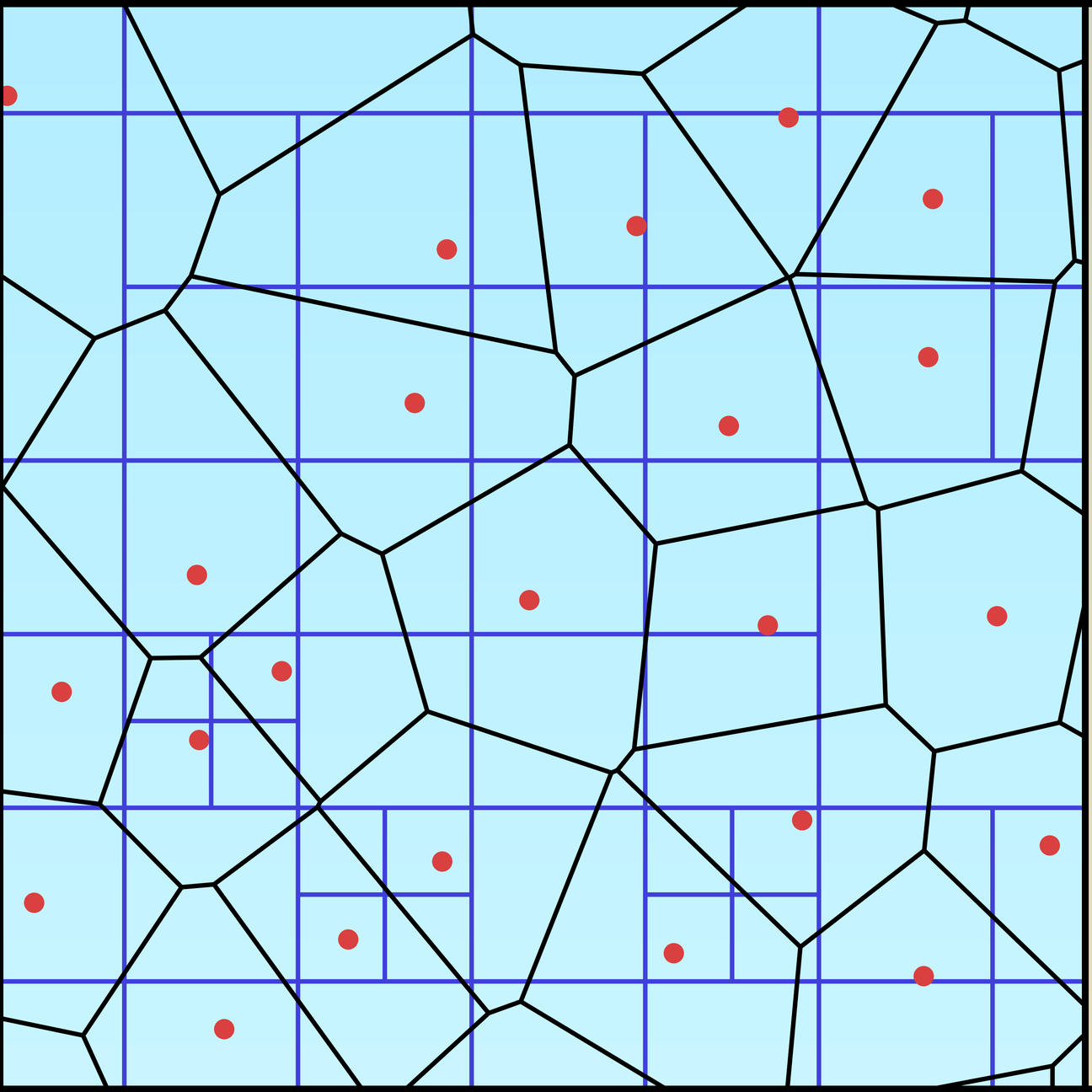
All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.



All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

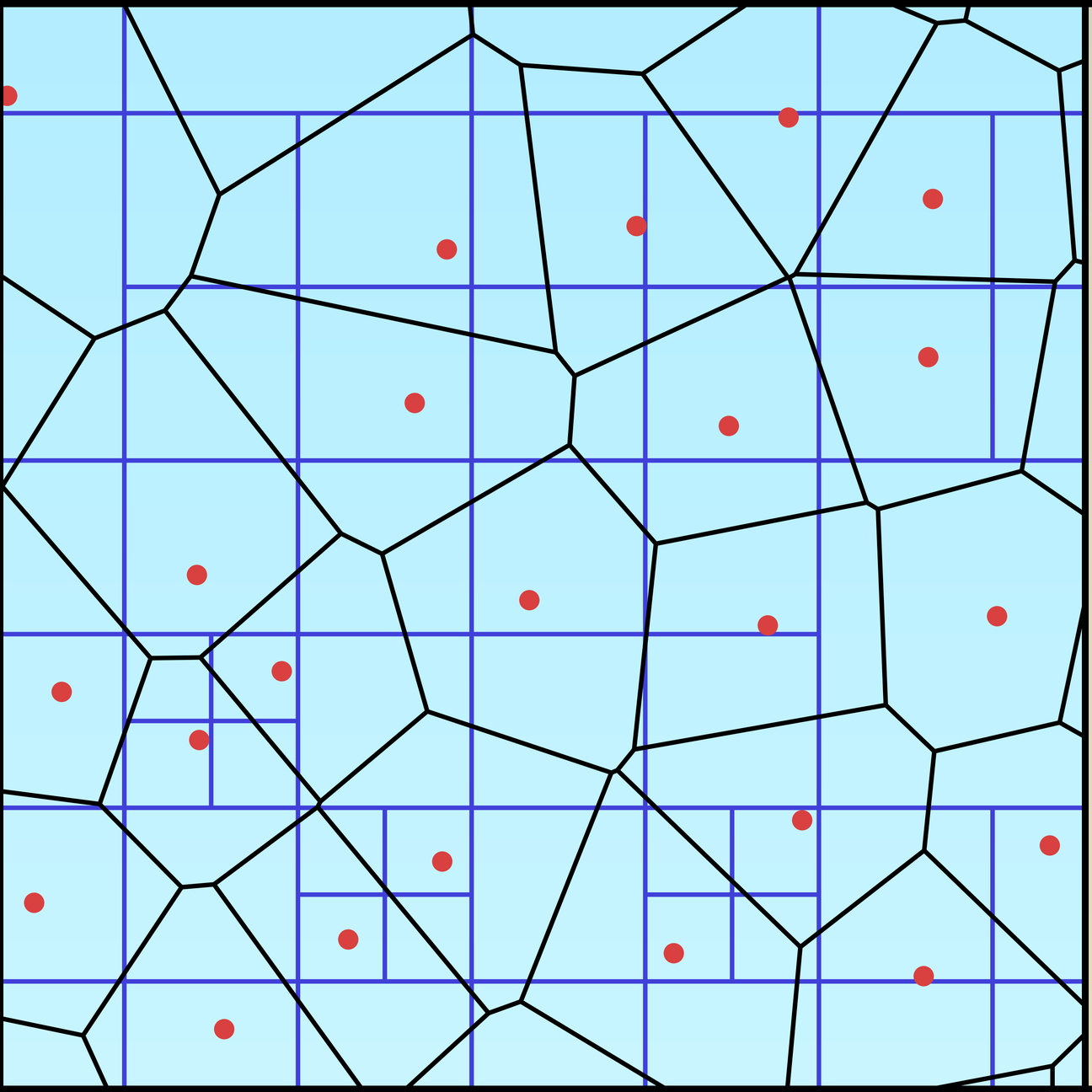
Intuitively, though, once you built one, you should be able to *derive* the others.



All structures I mentioned have $\Theta(n)$ size and on a real-valued pointer machine take $\Theta(n \log n)$ time to build.

Intuitively, though, once you built one, you should be able to *derive* the others.

Since information is *local*, such a conversion should take $O(n)$ time.



We turn things around,
and define what we mean
by a proximity structure
“capturing the local
information” of $P$ in terms
of *computation*.

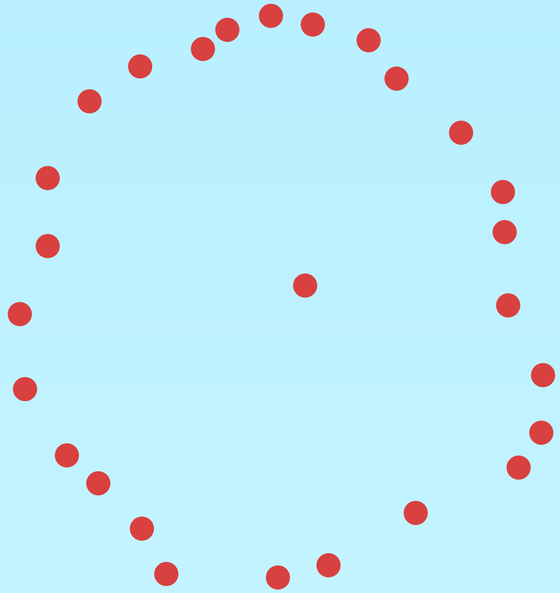
Unfortunately, not all point sets behave themselves.

Unfortunately, not
all point sets behave
themselves.

A single point may have
more than $O(1)$ neighbours.

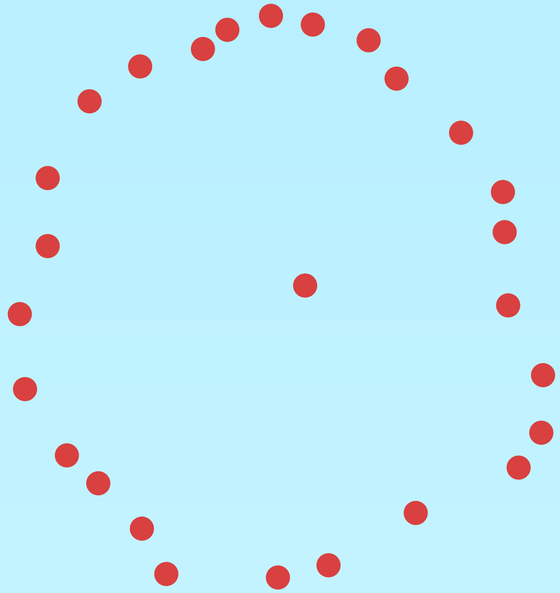
Unfortunately, not
all point sets behave
themselves.

A single point may have
more than $O(1)$ neighbours.



Unfortunately, not
all point sets behave
themselves.

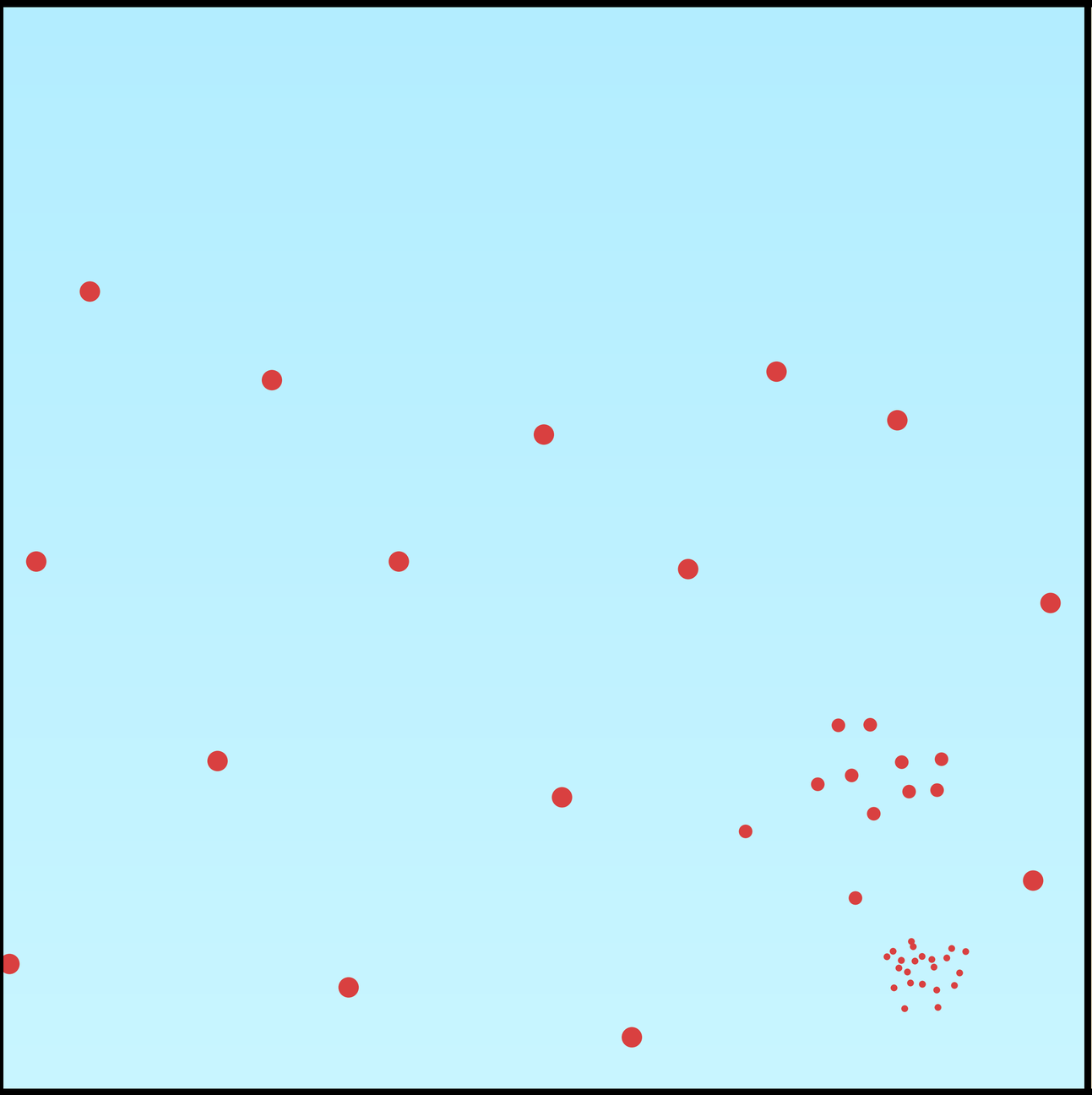
A single point may have
more than $O(1)$ neighbours.



Unfortunately, not all point sets behave themselves.

A single point may have more than $O(1)$ neighbours.

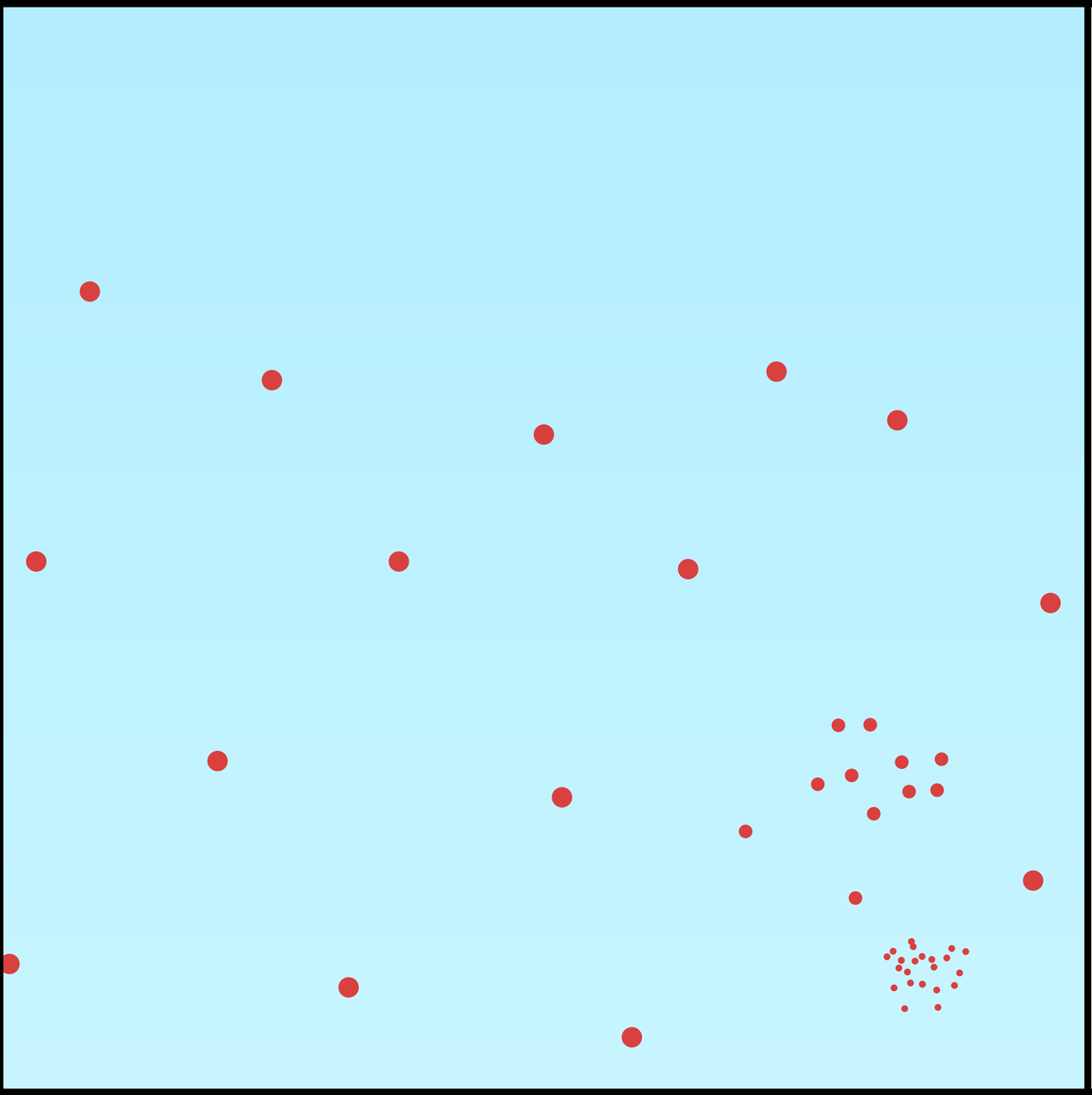
The scale of a point set need not be uniform.



Unfortunately, not all point sets behave themselves.

A single point may have more than $O(1)$ neighbours.

The scale of a point set need not be uniform.



Unfortunately, not all point sets behave themselves.

A single point may have more than $O(1)$ neighbours.

The scale of a point set need not be uniform.

What does “local” mean exactly?

Do we assume that in practice, point sets are well-behaved, and design useable algorithms that only work (or are only efficient) on certain classes of point sets?

Do we assume that in practice, point sets are well-behaved, and design useable algorithms that only work (or are only efficient) on certain classes of point sets?

*Or do we insist on being general, and design devilishly complicated algorithms that nobody will ever use, but that are able to handle arbitrary point sets?*

# Chapter 1
## *PROXIMITY STRUCTURES*

Point sets and proximity

Computational questions

State of the art in 1D

State of the art in 2D

Points in $\mathbb{R}^1$ are numbers.

Points in $\mathbb{R}^1$ are numbers.

The obvious “proximity structure” is their order.

Points in $\mathbb{R}^1$ are numbers.

The obvious “proximity structure” is their order.

Computing this order (a.k.a. *sorting*) has an $\Omega(n \log n)$ lower bound on a real-valued pointer machine.

[Shannon, 1948]

Points in $\mathbb{R}^1$ are numbers.

The obvious “proximity structure” is their order.

Computing this order (a.k.a. *sorting*) has an $\Omega(n \log n)$ lower bound on a real-valued pointer machine.

Any other structure can be derived from the order in $O(n)$ time.

[Shannon, 1948]

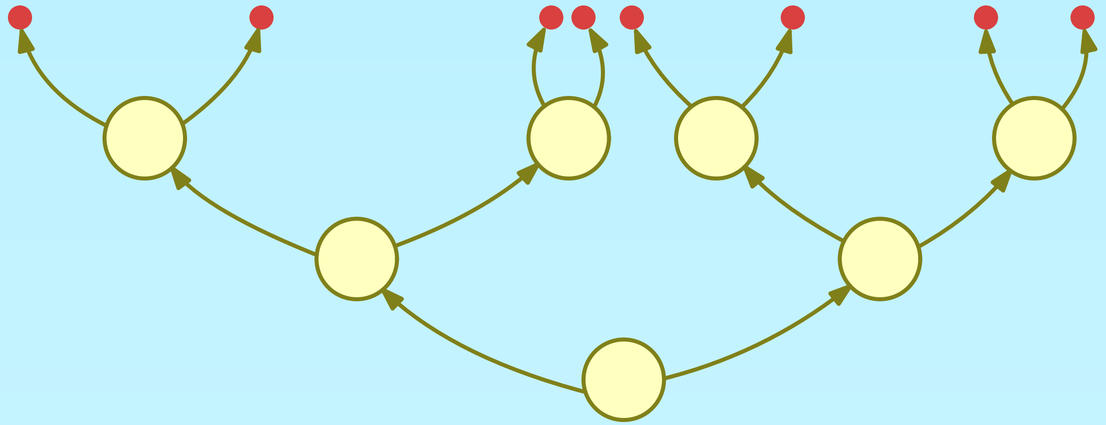
Points in $\mathbb{R}^1$ are numbers.

The obvious “proximity structure” is their order.

Computing this order (a.k.a. *sorting*) has an $\Omega(n \log n)$ lower bound on a real-valued pointer machine.

Any other structure can be derived from the order in $O(n)$ time.

[Shannon, 1948]



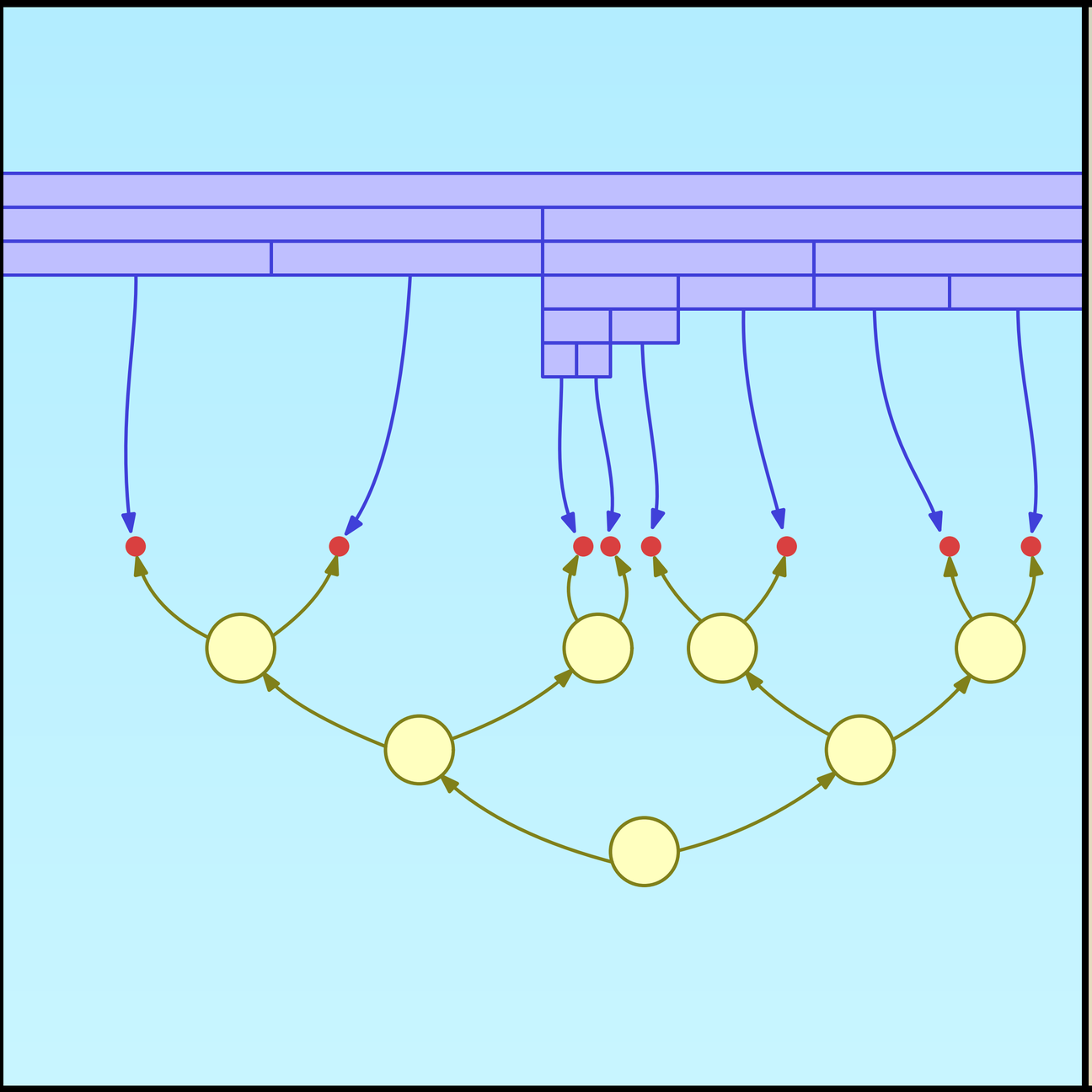
Points in $\mathbb{R}^1$ are numbers.

The obvious “proximity structure” is their order.

Computing this order (a.k.a. *sorting*) has an $\Omega(n \log n)$ lower bound on a real-valued pointer machine.

Any other structure can be derived from the order in $O(n)$ time.

[Shannon, 1948]



Points in $\mathbb{R}^1$ are numbers.

The obvious “proximity structure” is their order.

Computing this order (a.k.a. *sorting*) has an $\Omega(n \log n)$ lower bound on a real-valued pointer machine.

Any other structure can be derived from the order in $O(n)$ time.

[Shannon, 1948]

# Chapter 1
## *PROXIMITY STRUCTURES*

Point sets and proximity

Computational questions

State of the art in 1D

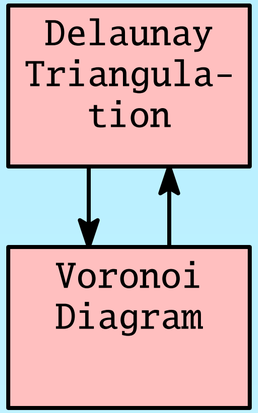
State of the art in 2D

In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

Voronoi Diagram

In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

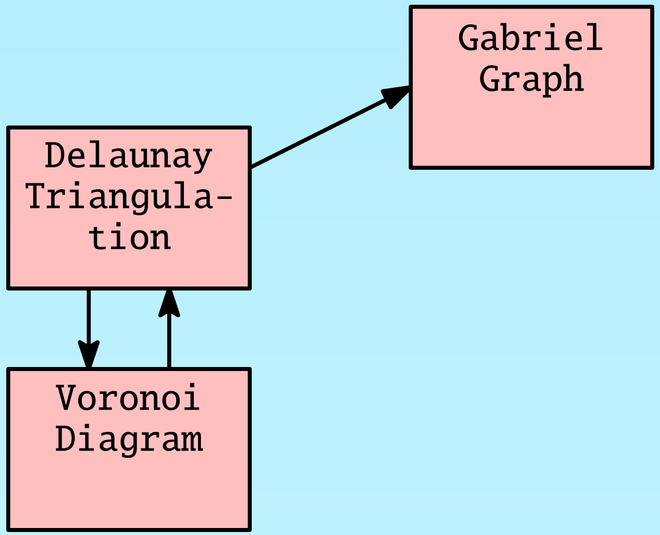
[Dirichlet, 1850]




In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

⟶ Deterministic linear time

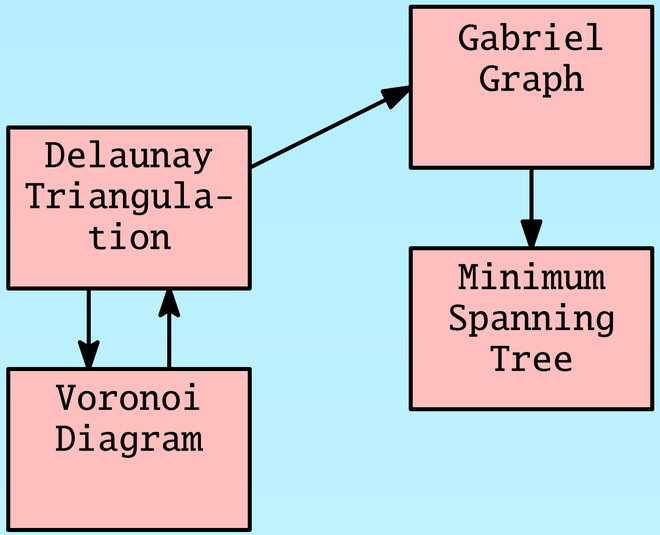
[Dirichlet, 1850]
[Delaunay, 1934]




In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

→ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]

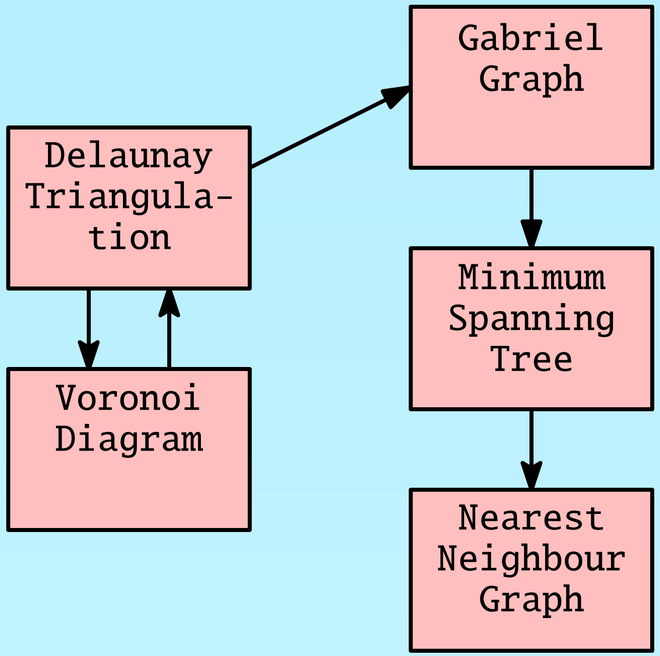



# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

$\longrightarrow$ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]

[Matsui, 1995]



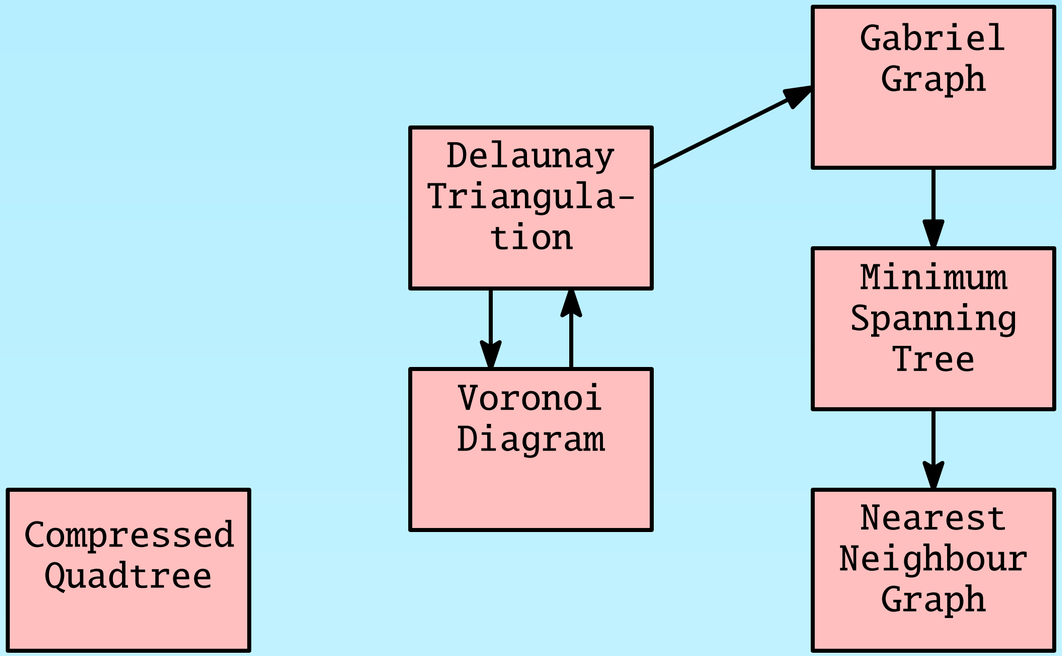

# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

$\longrightarrow$ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]

[Preparata & Shamos, 1985]

[Matsui, 1995]

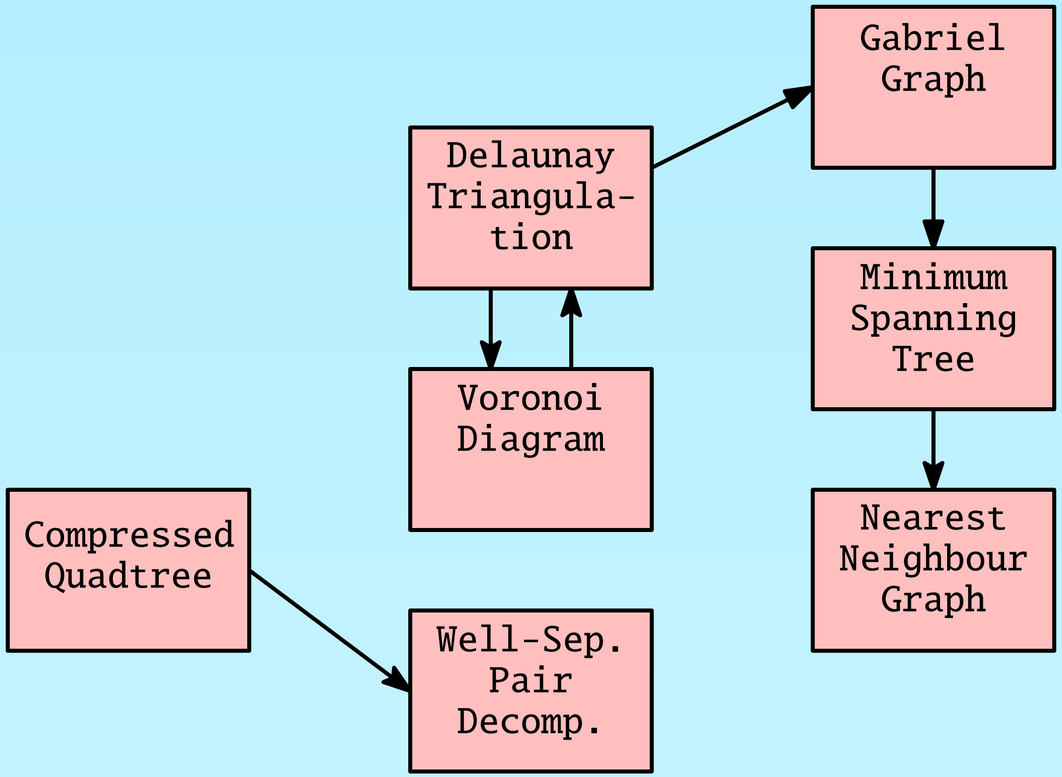



# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

→ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]

[Matsui, 1995]

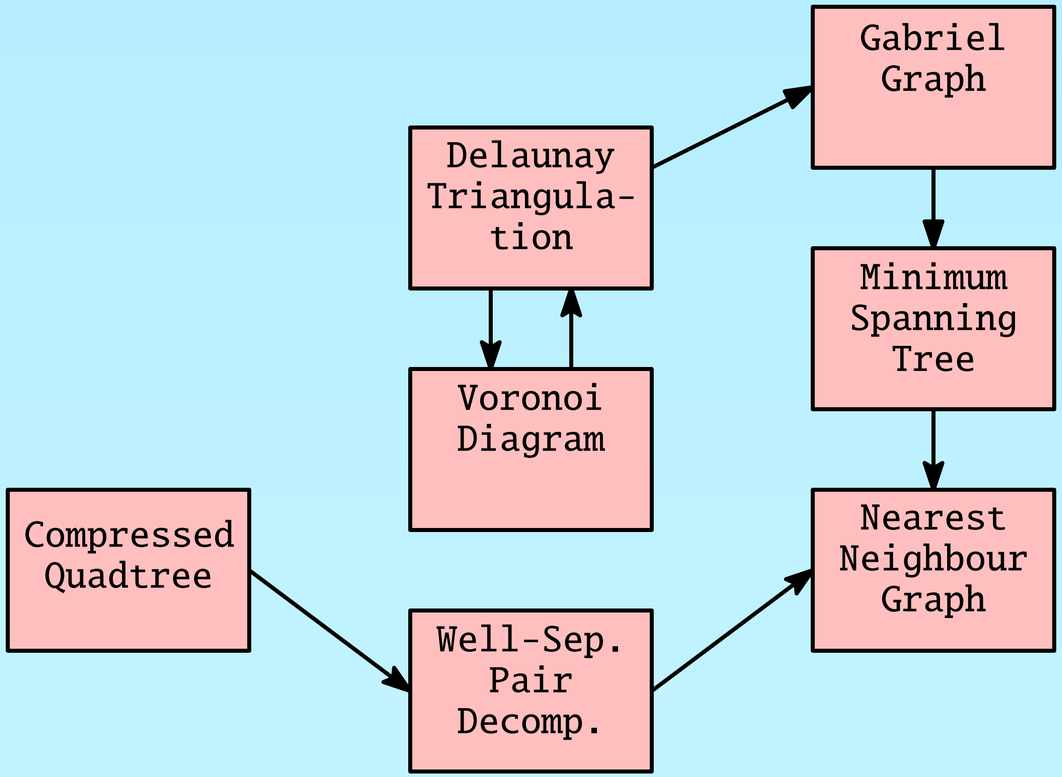



# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

→ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]

[Matsui, 1995]
[Callahan & Kosaraju, 1995]

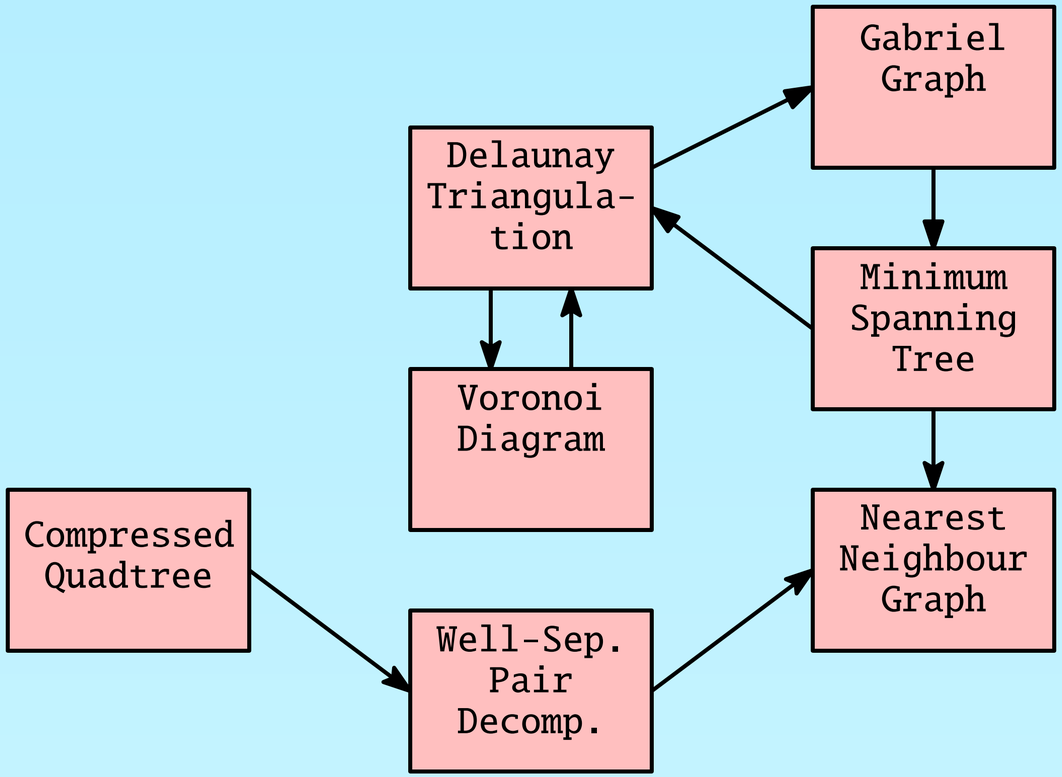



# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

⟶ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]

[Matsui, 1995]
[Callahan & Kosaraju, 1995]

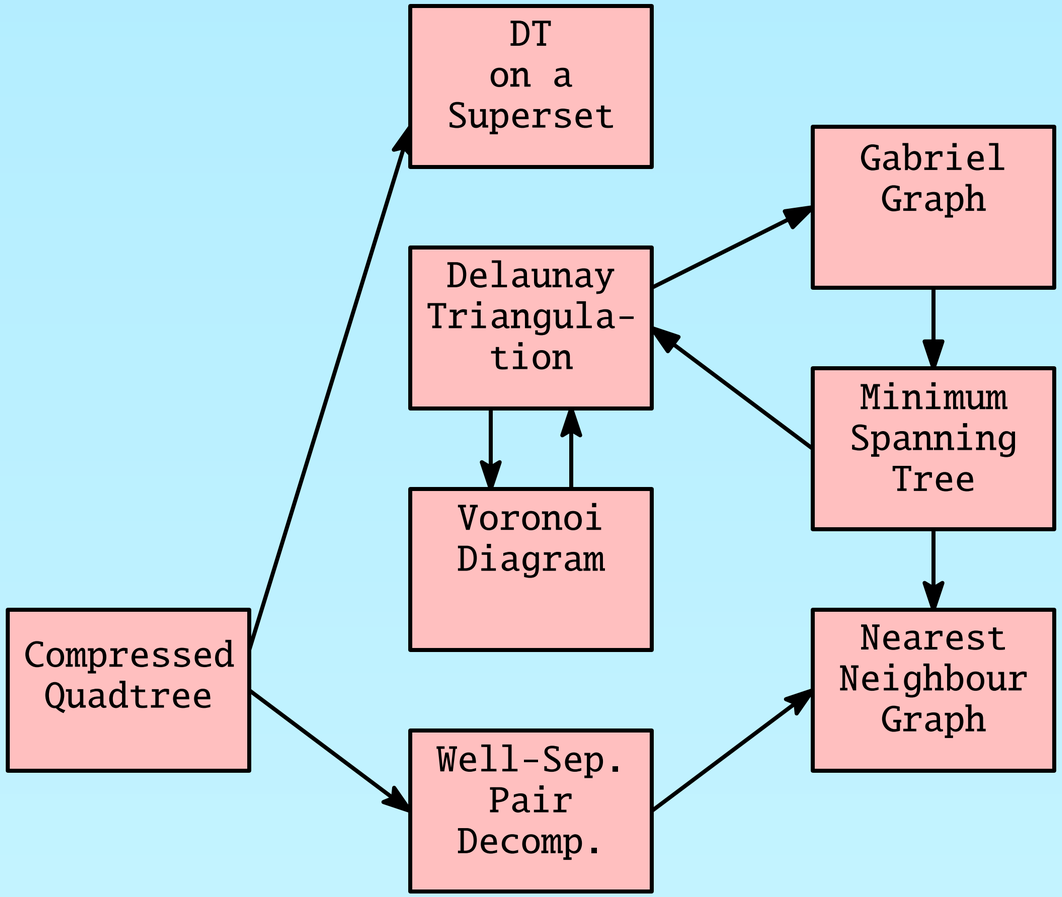



# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

⟶ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]

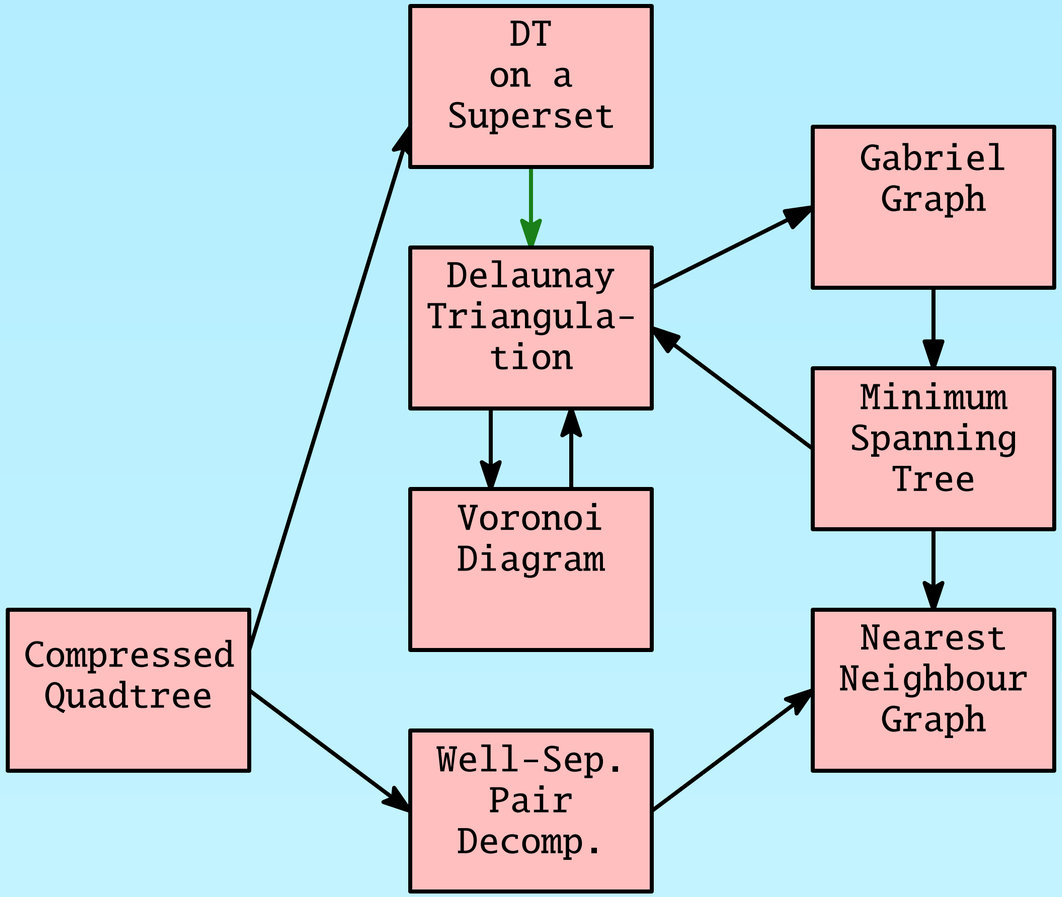
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]




# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

→ Deterministic linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]

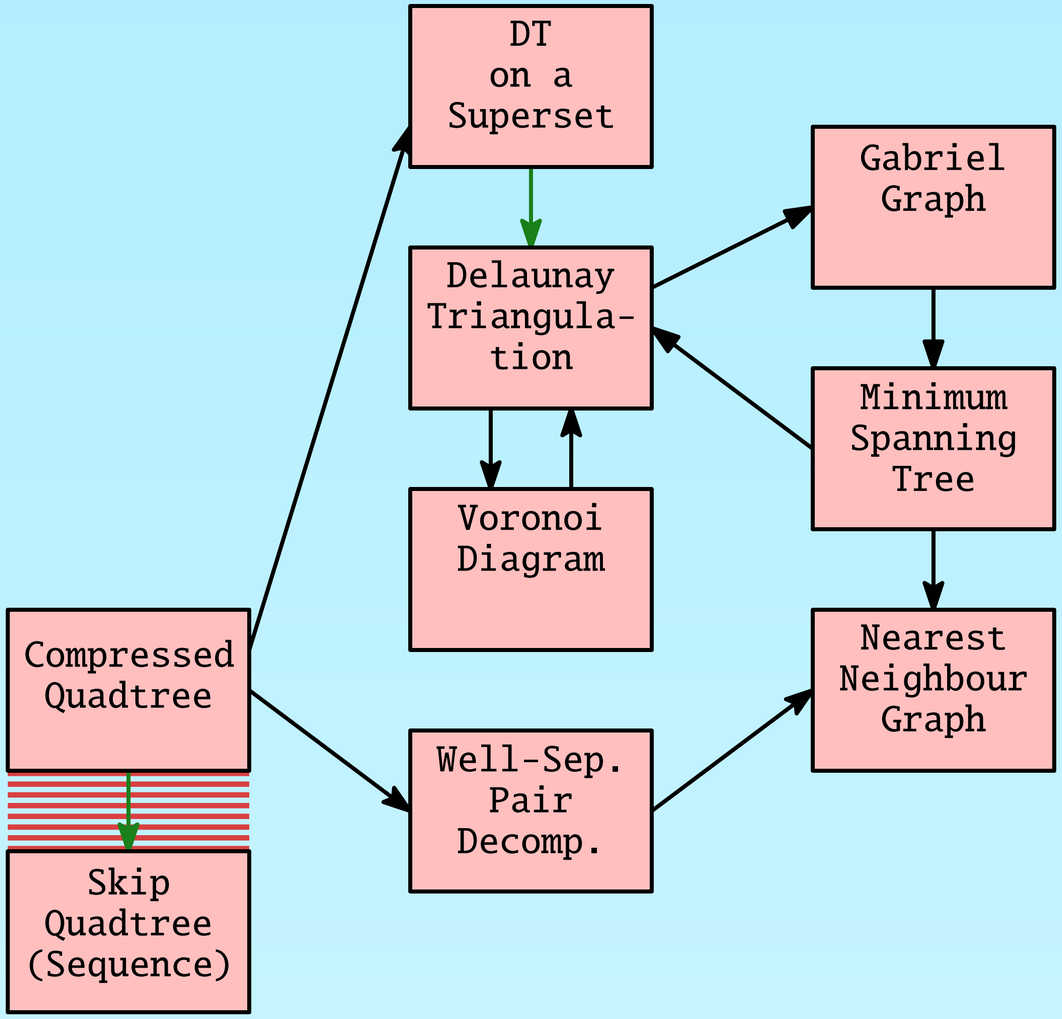



# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

→ Deterministic linear time
→ Randomised linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]

[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]

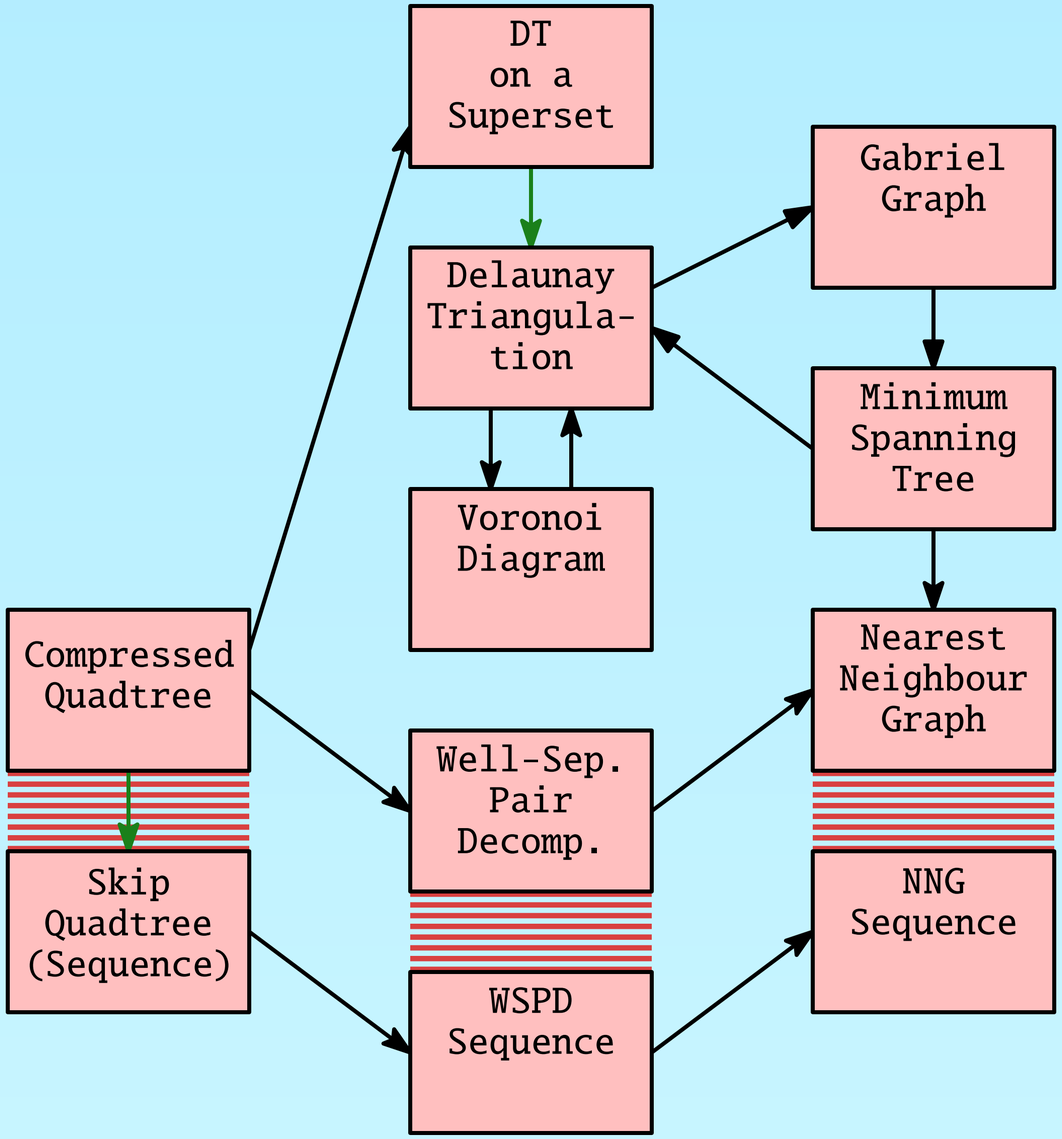



# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

→ Deterministic linear time
→ Randomised linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]

[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]

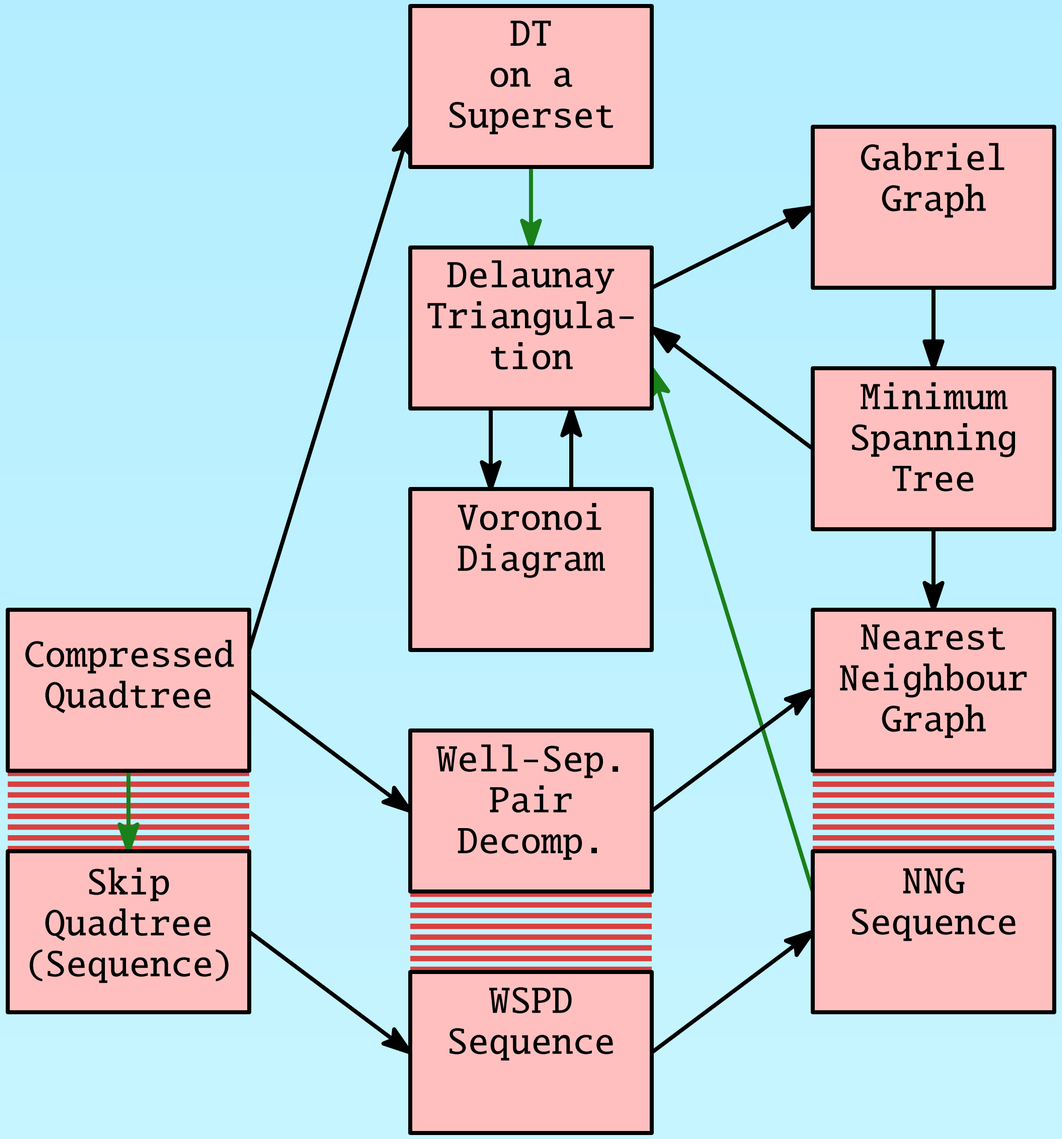



# In $\mathbb{R}^2$, there is a long history of "linear-time conversion" algorithms.

→ Deterministic linear time
→ Randomised linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]

[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]



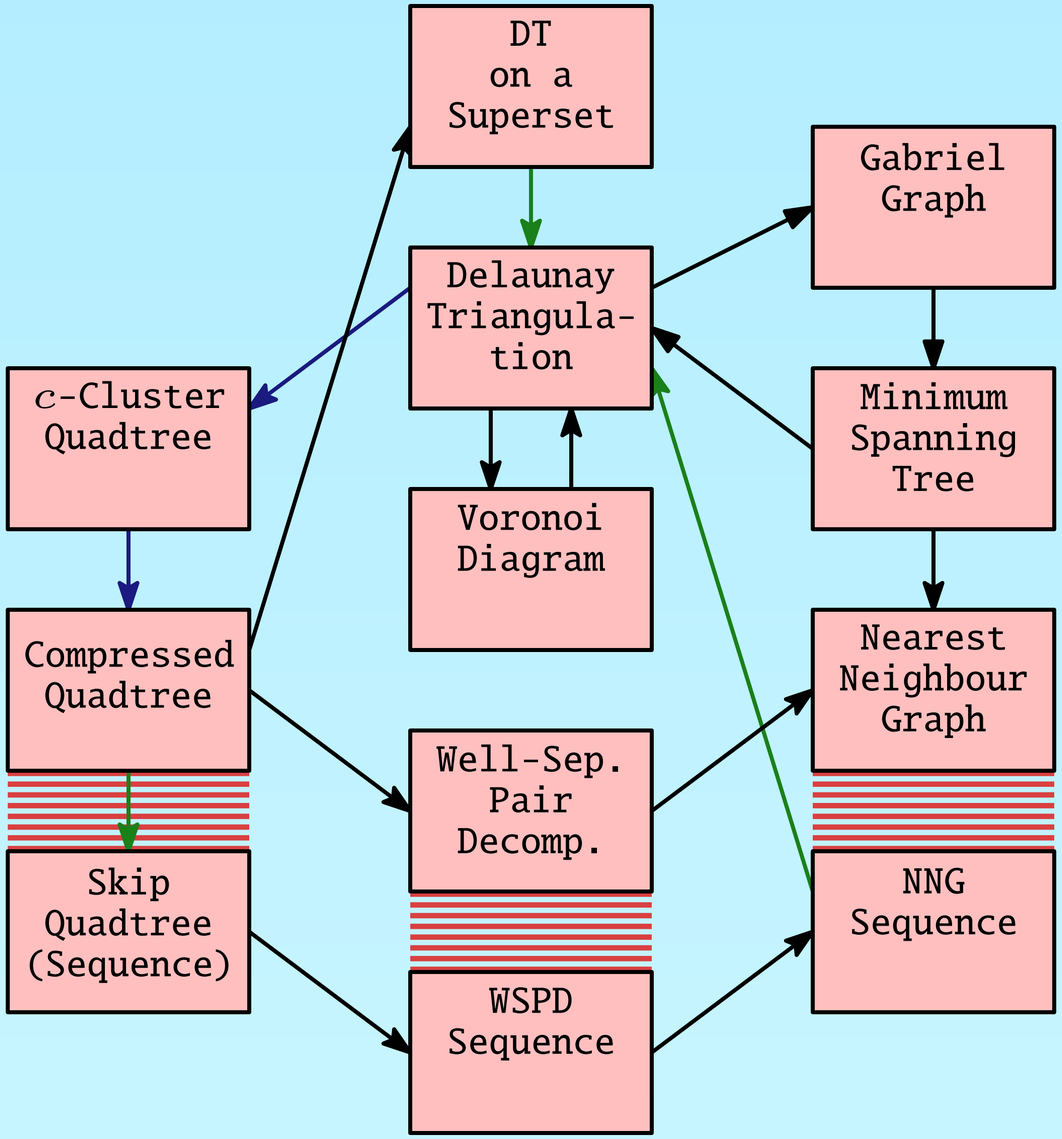

# In $\mathbb{R}^2$, there is a long history of “linear-time conversion” algorithms.

→ Deterministic linear time

→ Randomised linear time

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]

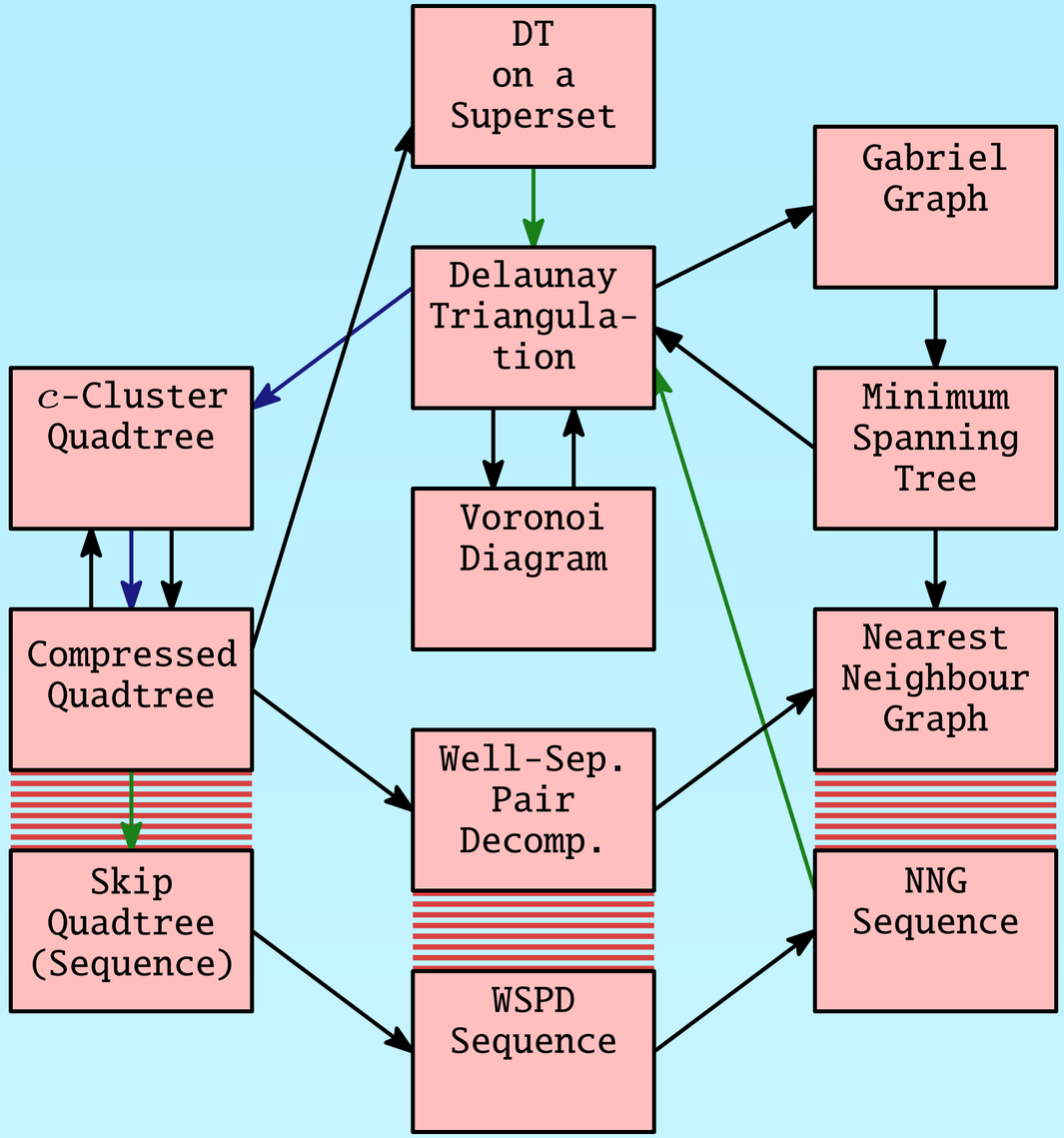
[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]
[Buchin & Mulzer, 2009]




# In $\mathbb{R}^2$, there is a long history of "linear-time conversion" algorithms.

→ Deterministic linear time
→ Randomised linear time
→ Linear time with floor operation

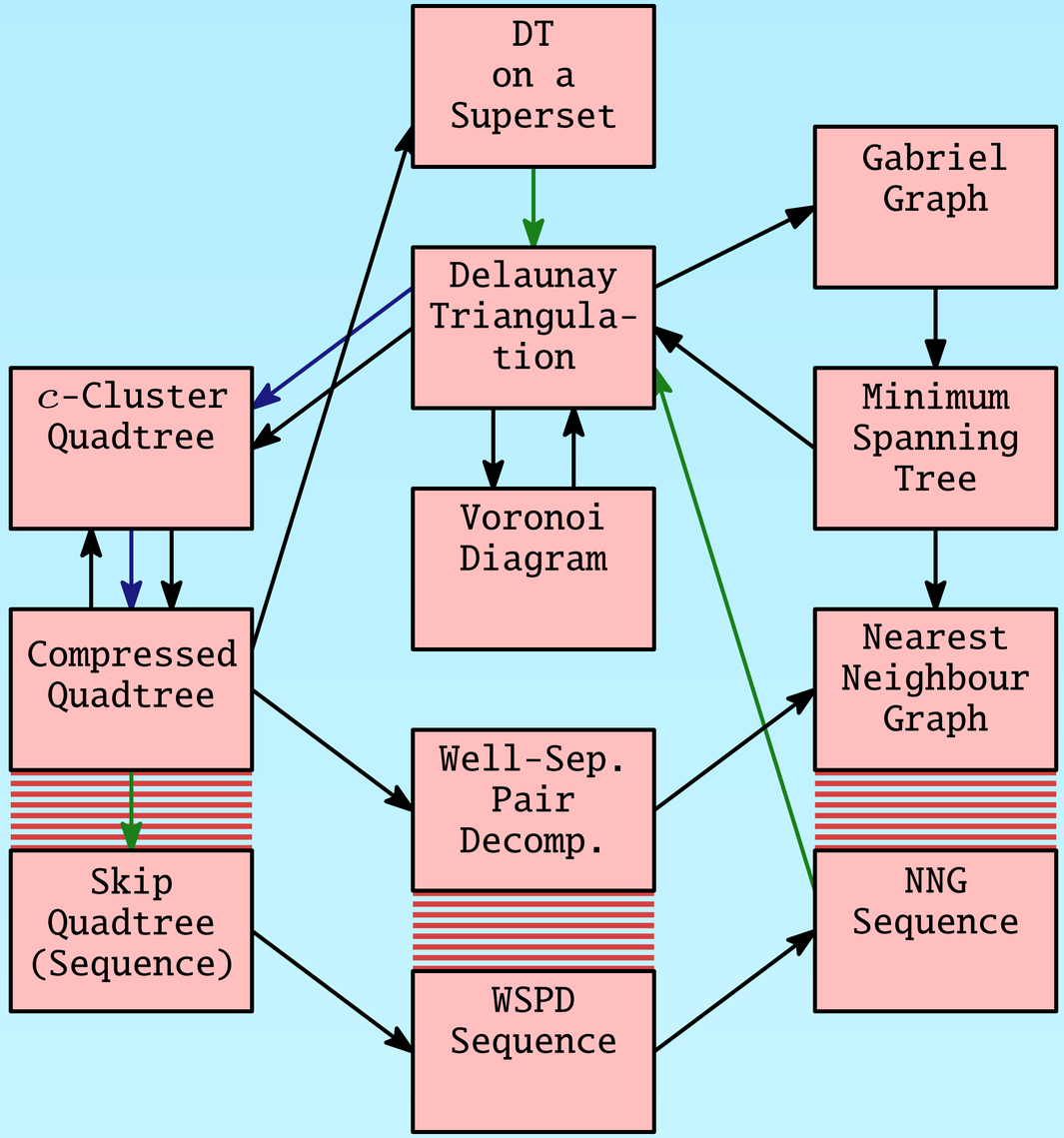
[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]
[Krznaric & Levcopoulos, 1998]
[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]
[Buchin & Mulzer, 2009]




# In $\mathbb{R}^2$, there is a long history of "linear-time conversion" algorithms.

→ Deterministic linear time
→ Randomised linear time
→ Linear time with floor operation

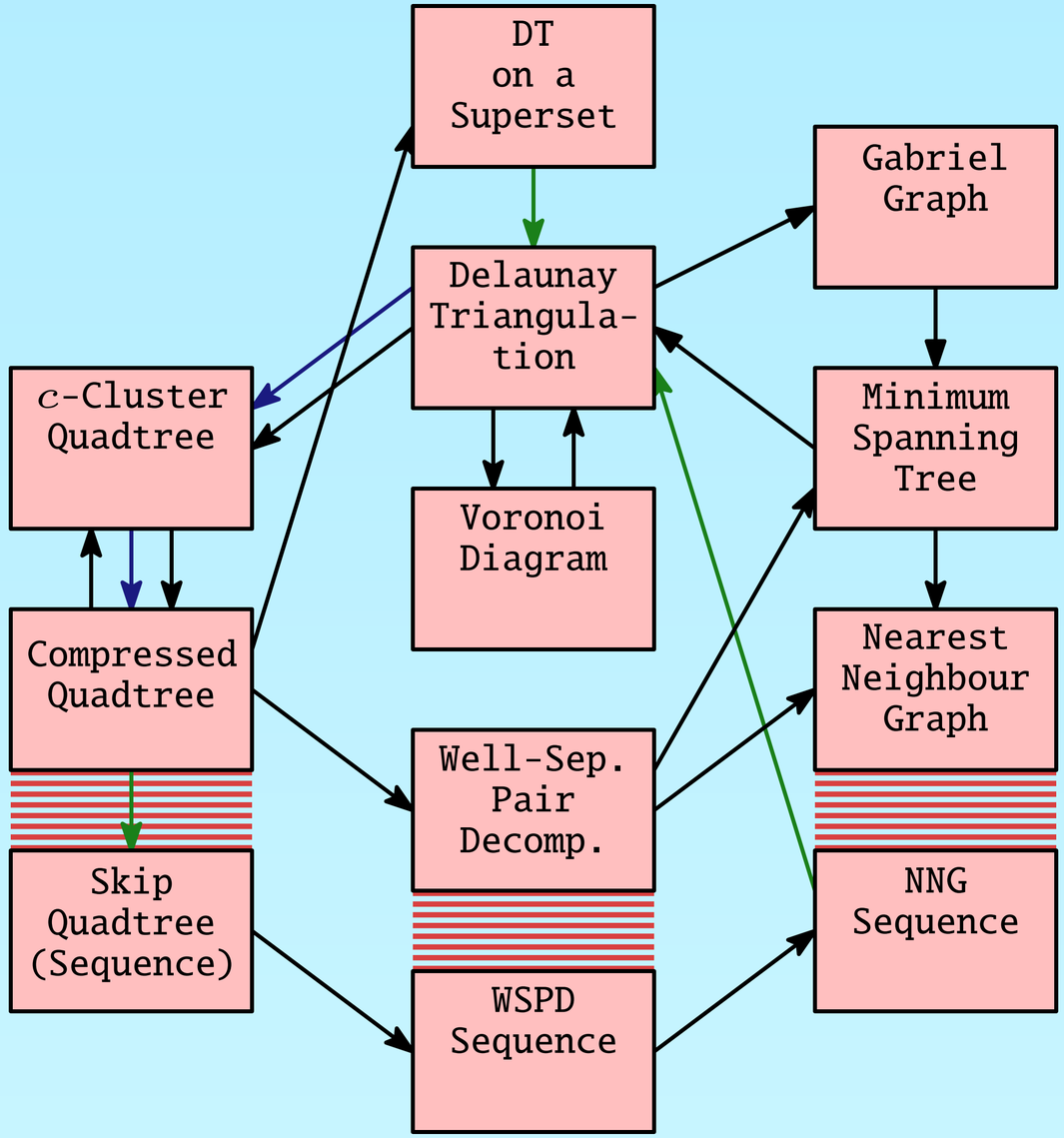
[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]
[Krznaric & Levcopoulos, 1998]
[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]
[Buchin & Mulzer, 2009]
[Löffler & Mulzer, 2011]




# In $\mathbb{R}^2$, there is a long history of "linear-time conversion" algorithms.

→ Deterministic linear time
→ Randomised linear time
→ Linear time with floor operation

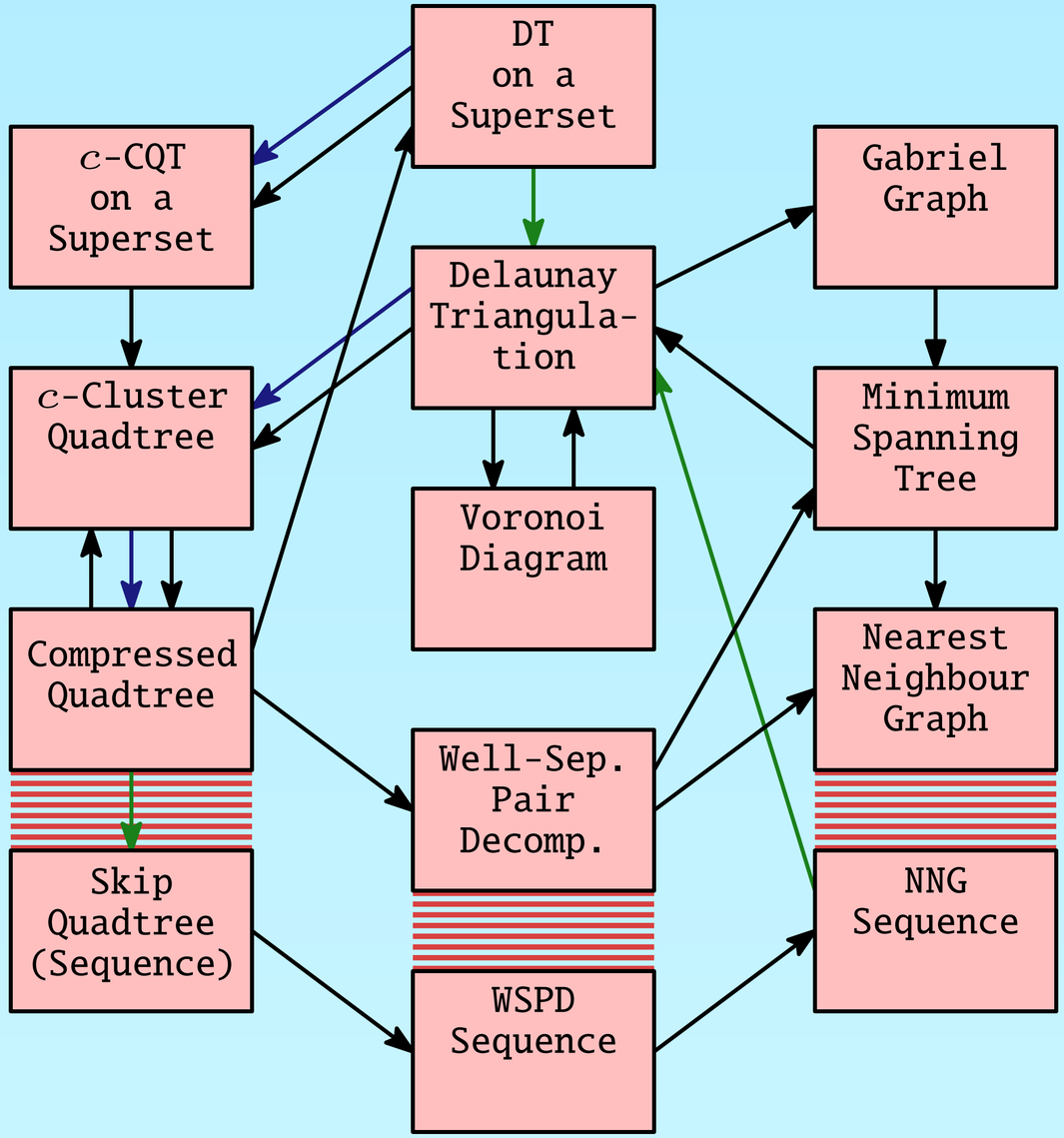
[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]
[Krznaric & Levcopoulos, 1998]
[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]
[Buchin & Mulzer, 2009]
[Löffler & Mulzer, 2011]




# In $\mathbb{R}^2$, there is a long history of "linear-time conversion" algorithms.

→ Deterministic linear time
→ Randomised linear time
→ Linear time with floor operation

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]
[Krznaric & Levcopoulos, 1998]
[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]
[Buchin & Mulzer, 2009]
[Löffler & Mulzer, 2011]




# In $\mathbb{R}^2$, there is a long history of "linear-time conversion" algorithms.

→ Deterministic linear time
→ Randomised linear time
→ Linear time with floor operation

[Dirichlet, 1850]
[Delaunay, 1934]
[Gabriel, 1969]
[Clarkson, 1983]
[Preparata & Shamos, 1985]
[Bern, Eppstein & Gilbert, 1990]
[Chazelle, 1991]
[Matsui, 1995]
[Callahan & Kosaraju, 1995]
[Chin & Wang, 1998]
[Krznaric & Levcopoulos, 1998]
[Chazelle, Devillers, Hurtado, Mora, Sacristán & Teillaud, 2001]
[Eppstein, Goodrich & Sun, 2005]
[Buchin & Mulzer, 2009]
[Löffler & Mulzer, 2011]

# Chapter 2
*PARTIAL INFORMATION*

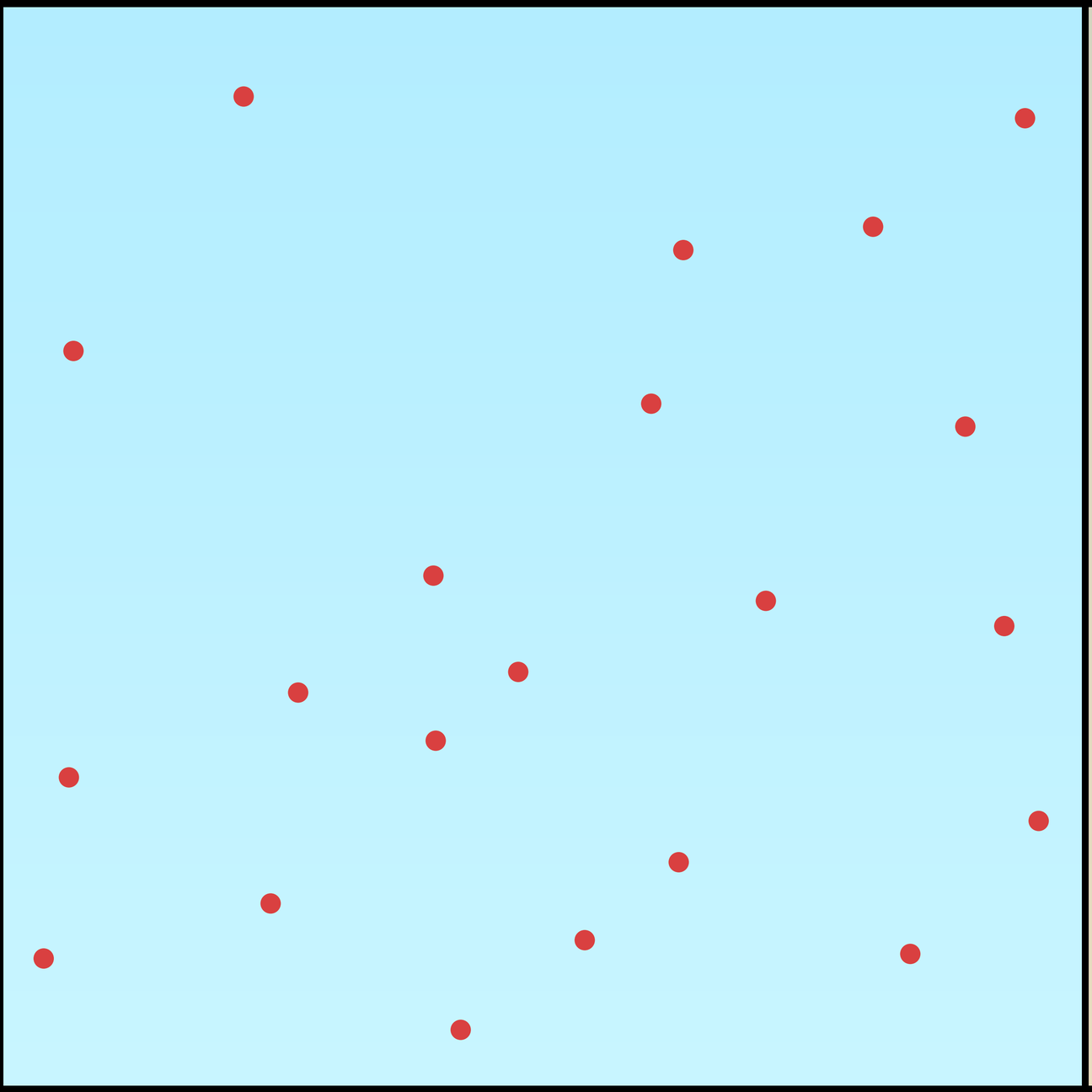
Uncertain locations

Quantifying information

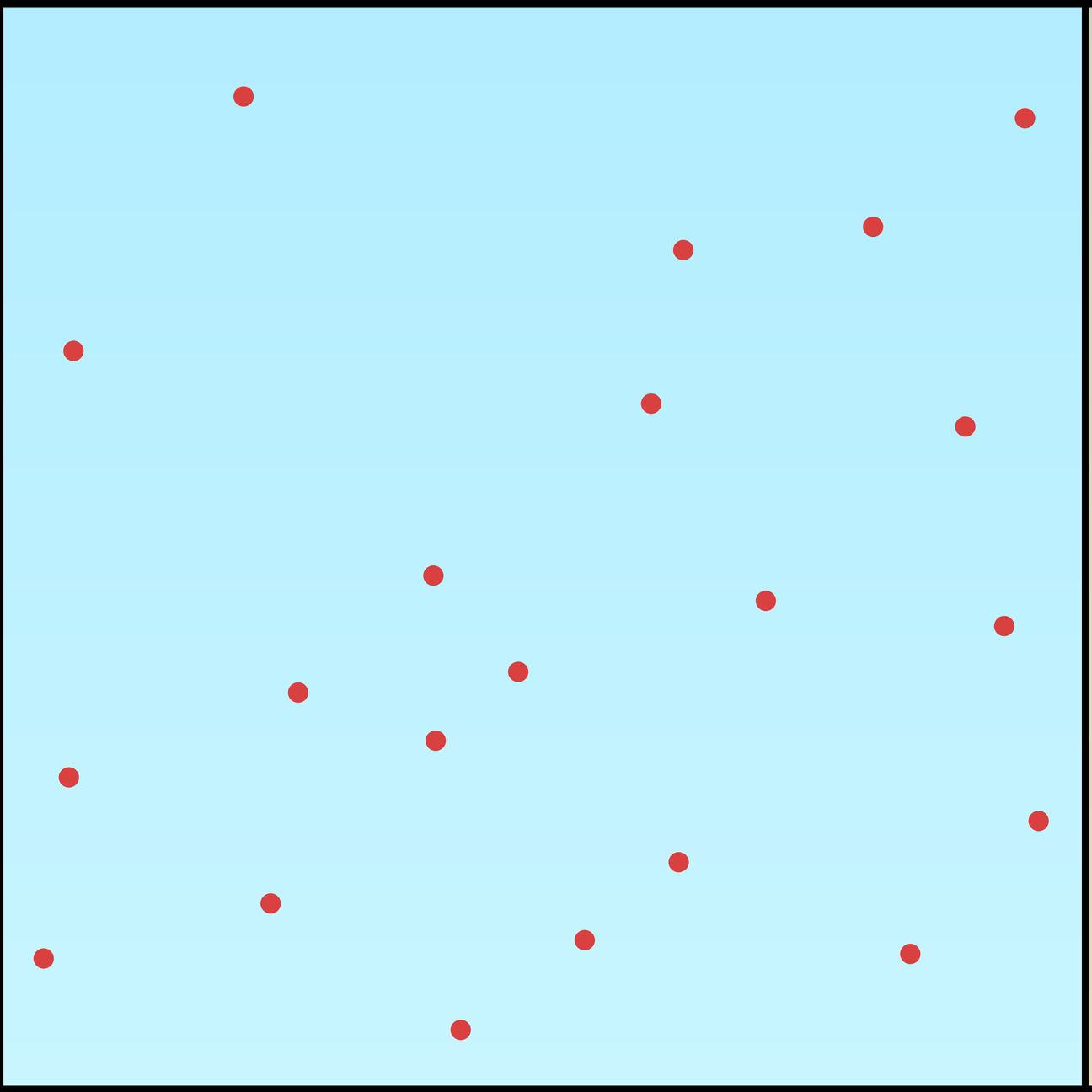
State of the art in 1D

State of the art in 2D

Let $P$ be a set of $n$ points in the plane.



Let $P$ be a set of $n$ points in the plane.



Let $P$ be a set of $n$ points in the plane.

What if we don't know where they are?

Let $P$ be a set of $n$ points in the plane.

What if we don't know where they are?

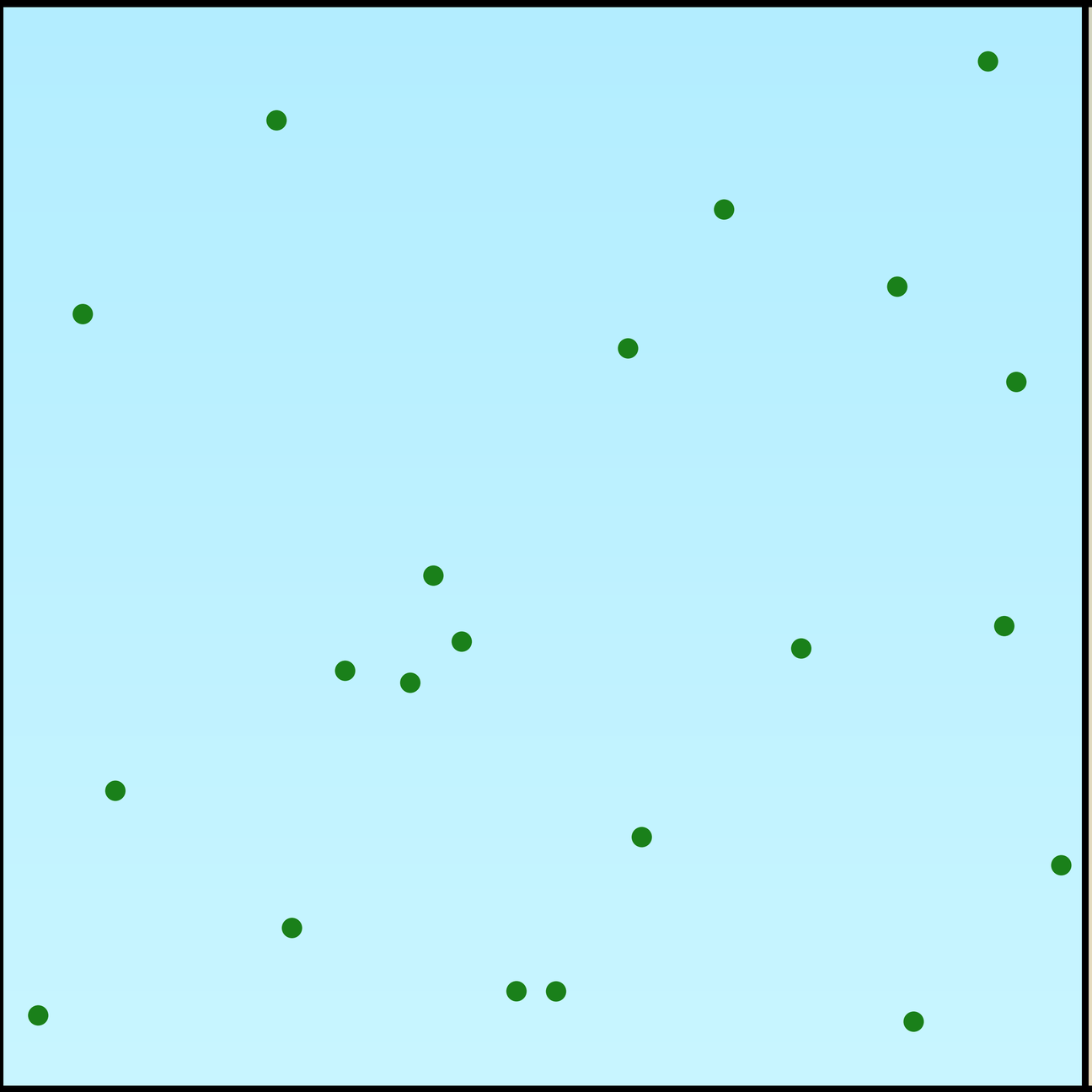
Let $P$ be a set of $n$ points in the plane.

What if we don't know *precisely* where they are?

Let $P$ be a set of $n$ points in the plane.

What if we don't know *precisely* where they are?

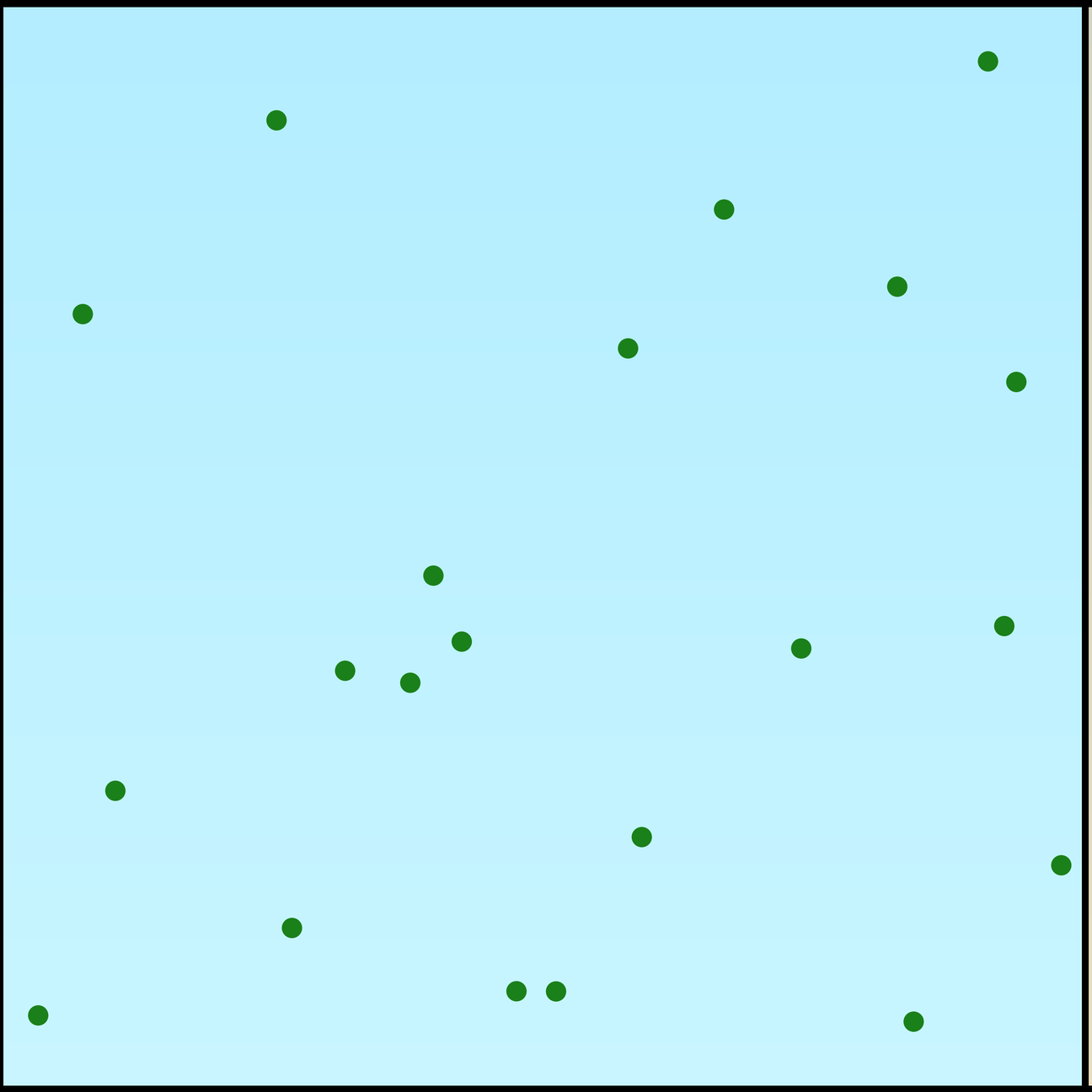
We have a measured set $P'$



Let $P$ be a set of $n$ points in the plane.

What if we don't know *precisely* where they are?

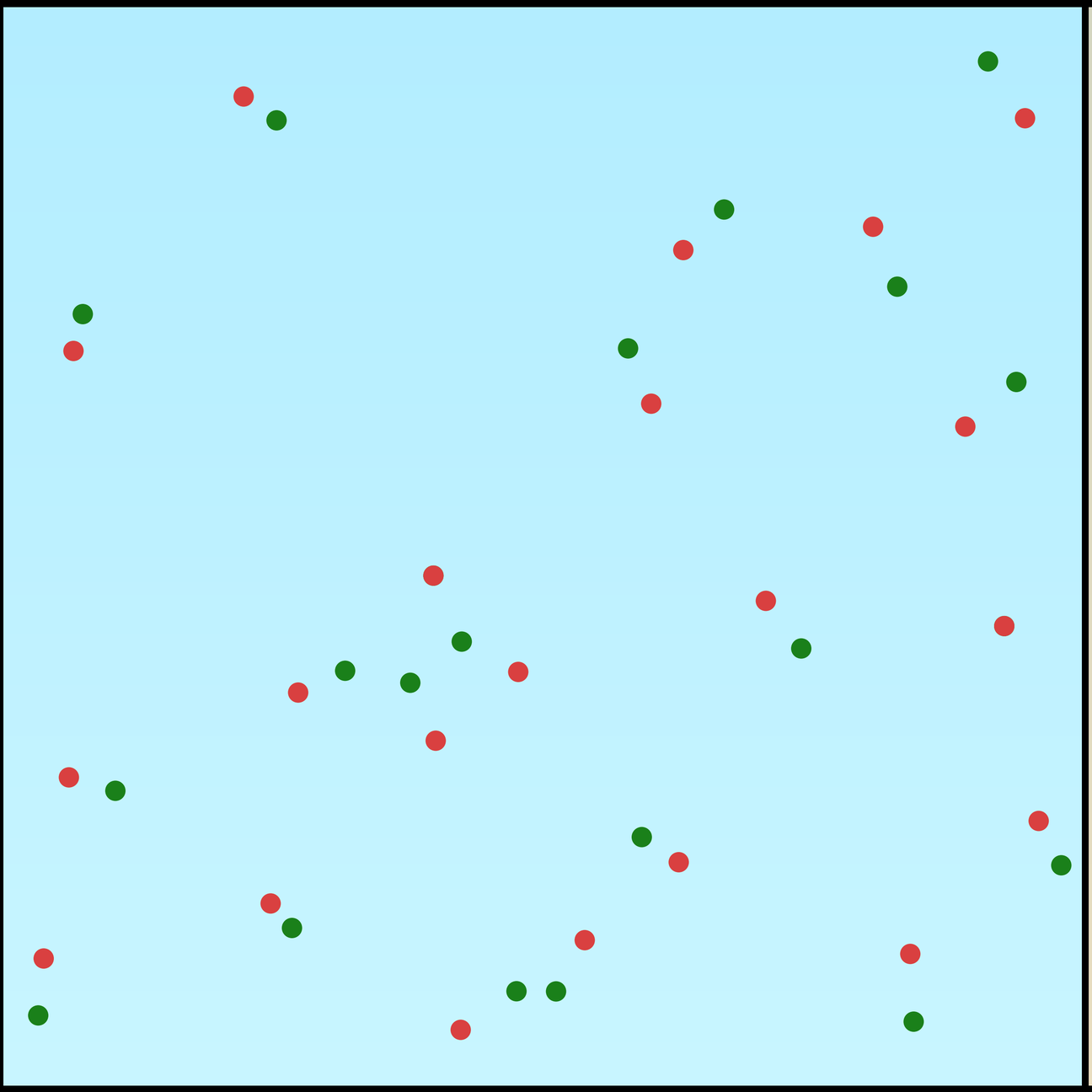
We have a measured set $P'$



Let $P$ be a set of $n$ points in the plane.

What if we don't know *precisely* where they are?

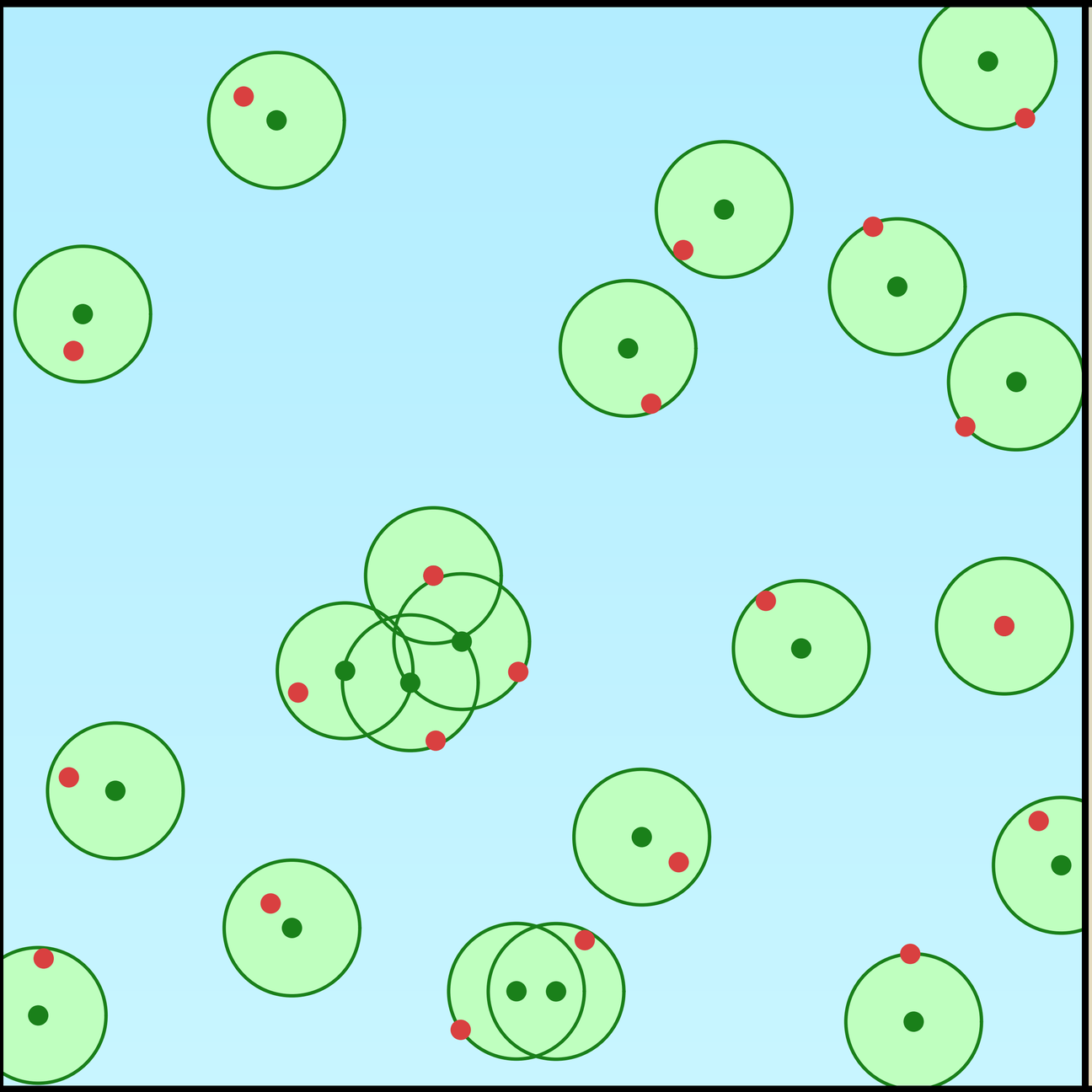
We have a measured set $P'$ at distance $\leq \varepsilon$ from $P$.



Let $P$ be a set of $n$ points in the plane.

What if we don't know *precisely* where they are?

We have a measured set $P'$ at distance $\leq \varepsilon$ from $P$.

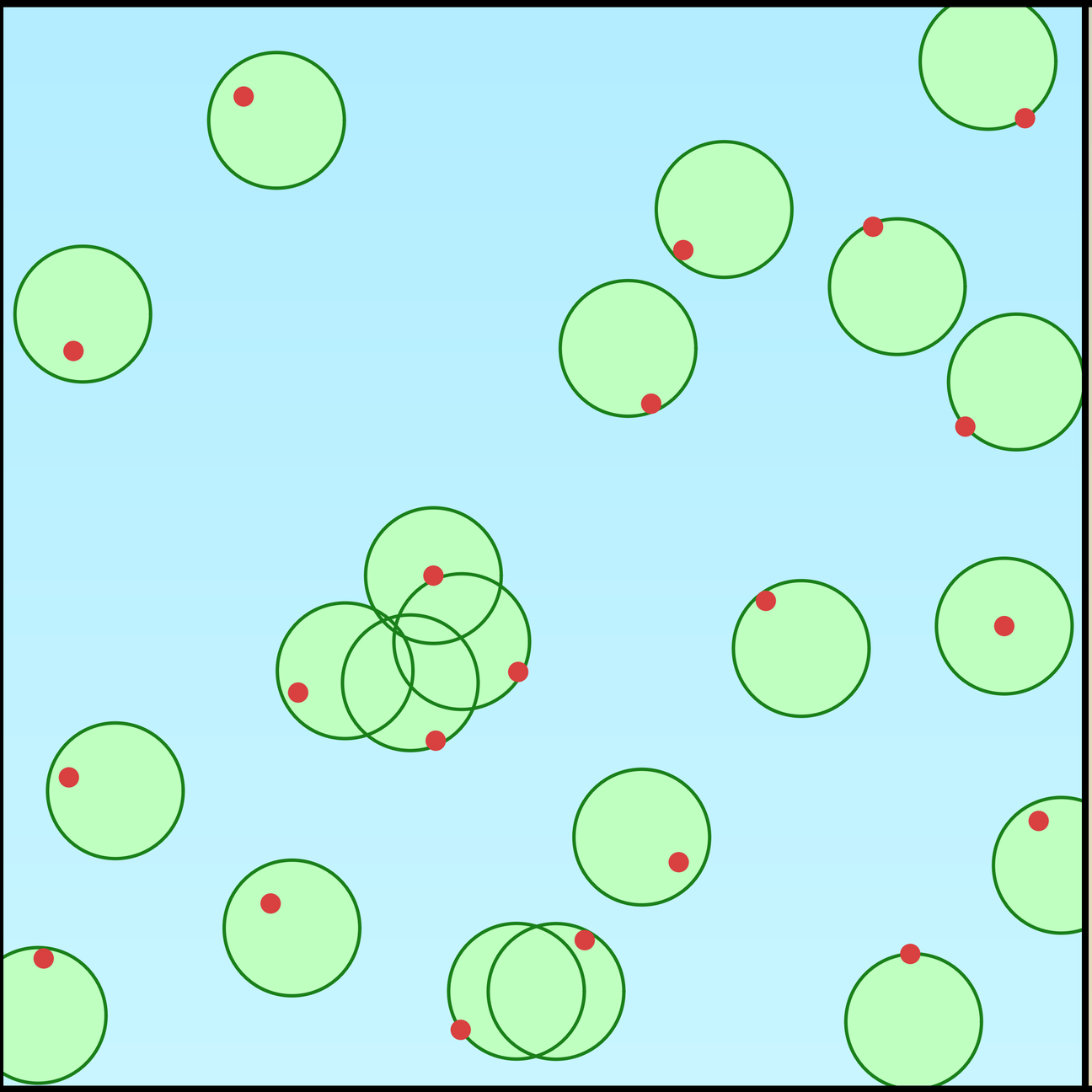


Let $P$ be a set of $n$ points in the plane.

What if we don't know *precisely* where they are?

We have a measured set $P'$ at distance $\leq \varepsilon$ from $P$.

We can model the set of *potential* locations of $P$ by a set of *disks* $\mathcal{R}$.



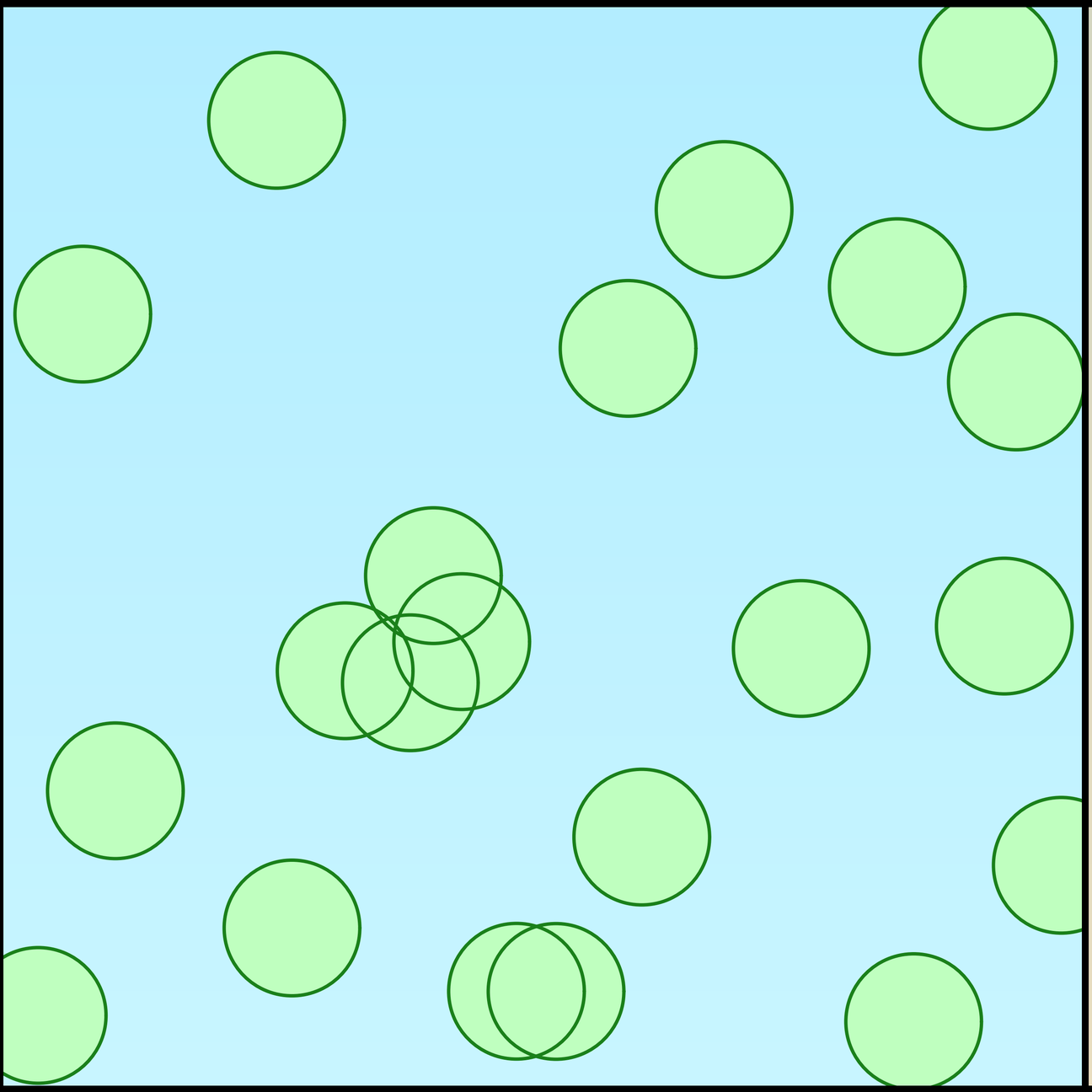
Let $P$ be a set of $n$ points in the plane.

What if we don't know *precisely* where they are?

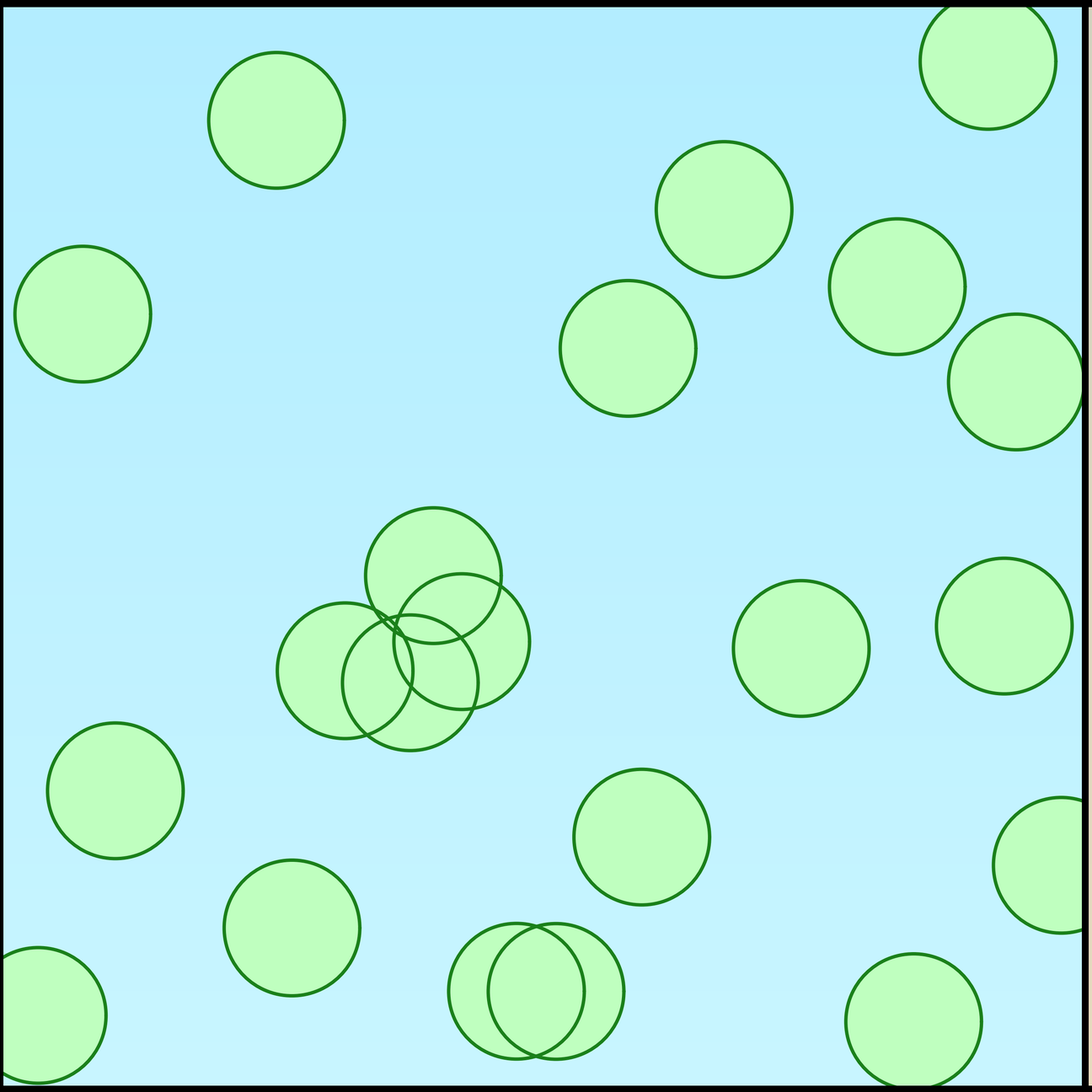
We have a measured set $P'$ at distance $\leq \varepsilon$ from $P$.

We can model the set of *potential* locations of $P$ by a set of *disks* $\mathcal{R}$.

$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.

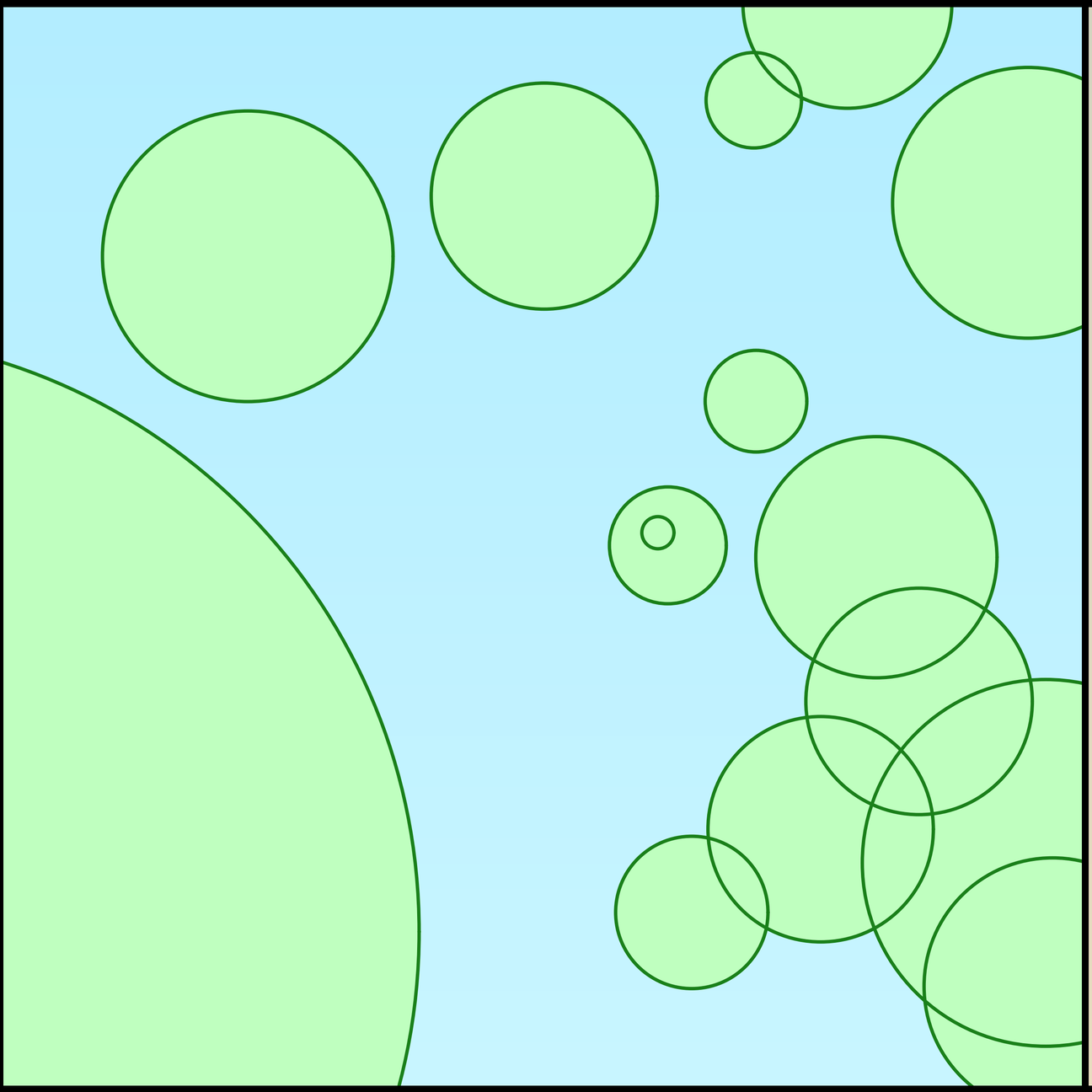


$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.



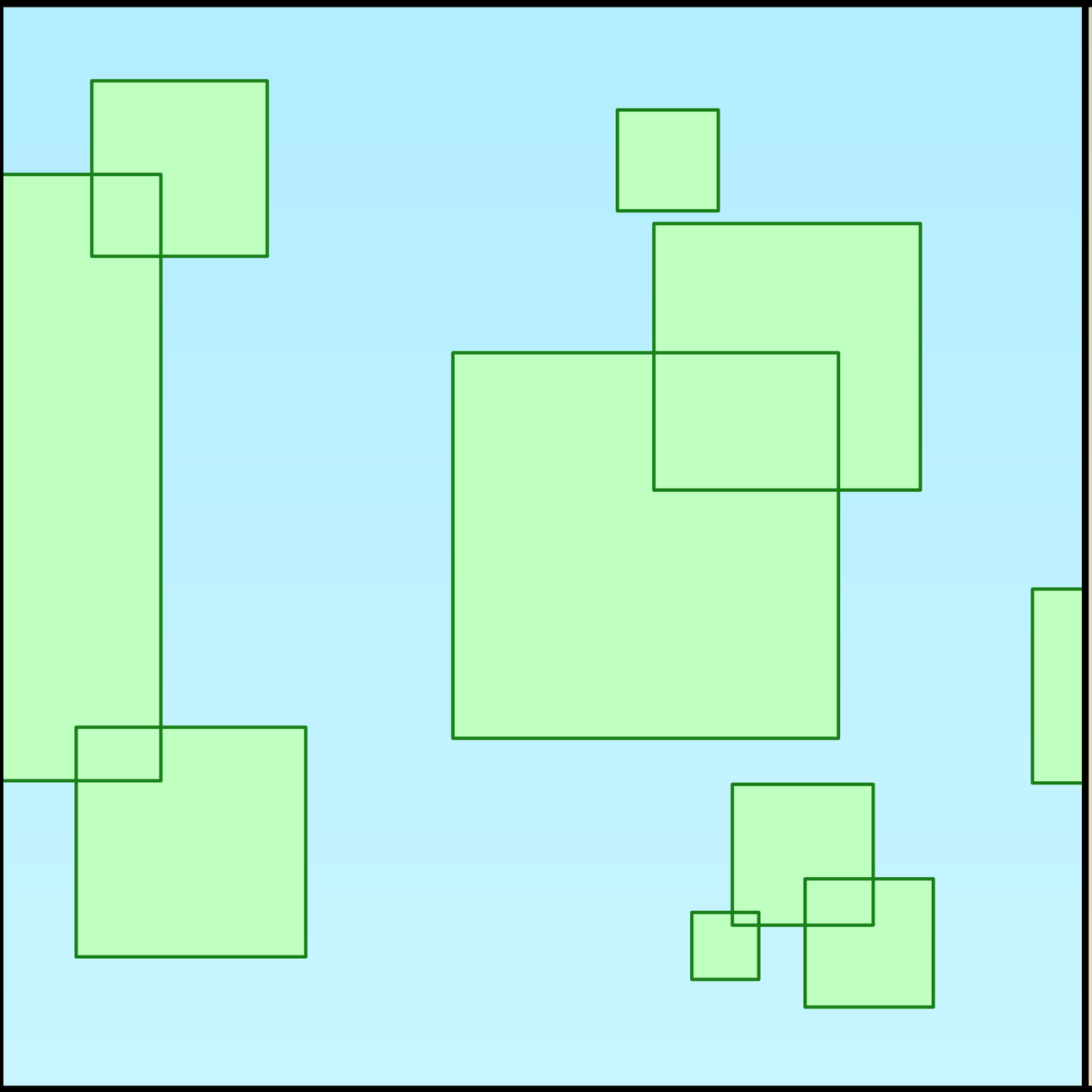
$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.

In principle, there is no reason to restrict $\mathcal{R}$ to a set of $\varepsilon$-disks.



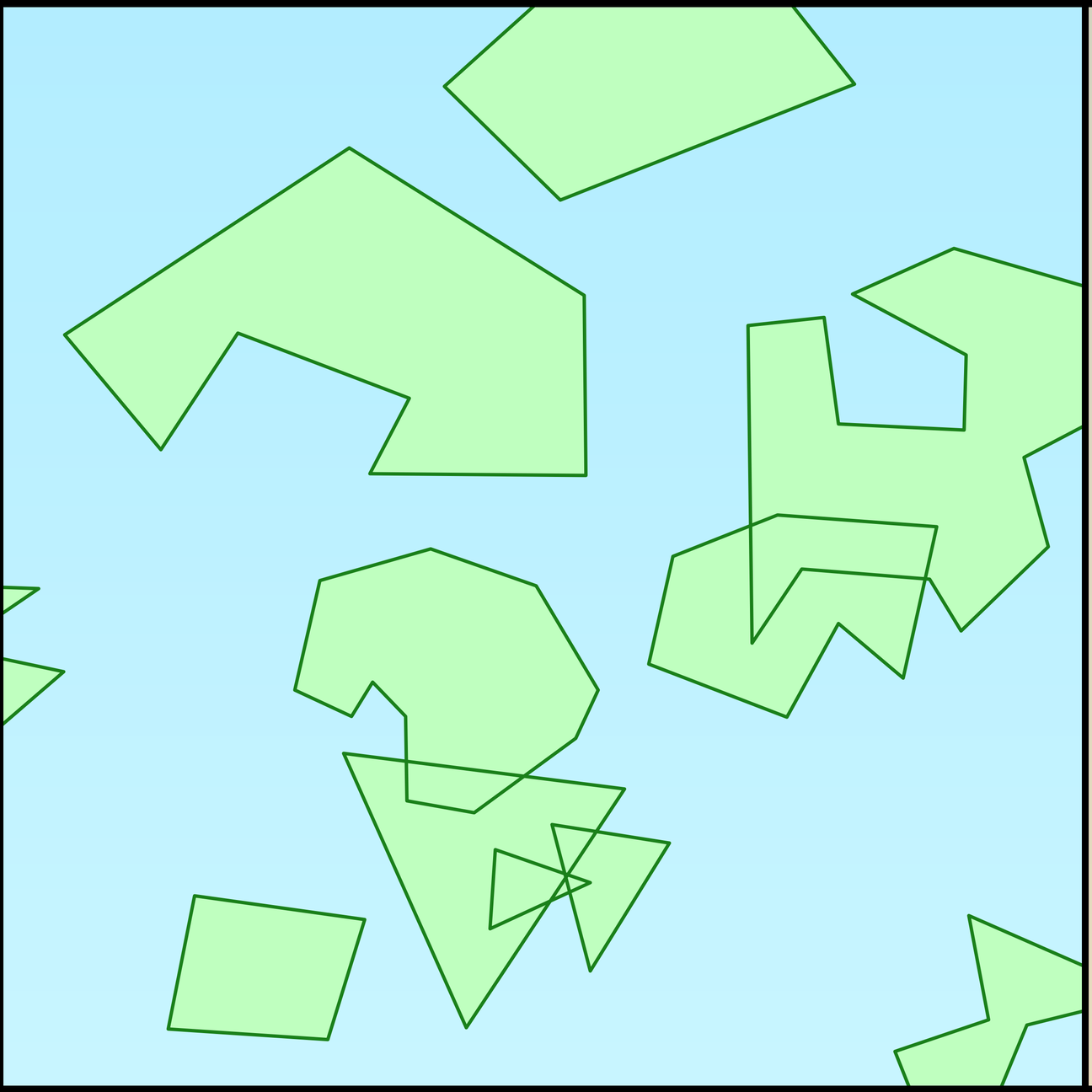
$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.

In principle, there is no reason to restrict $\mathcal{R}$ to a set of $\varepsilon$-disks.



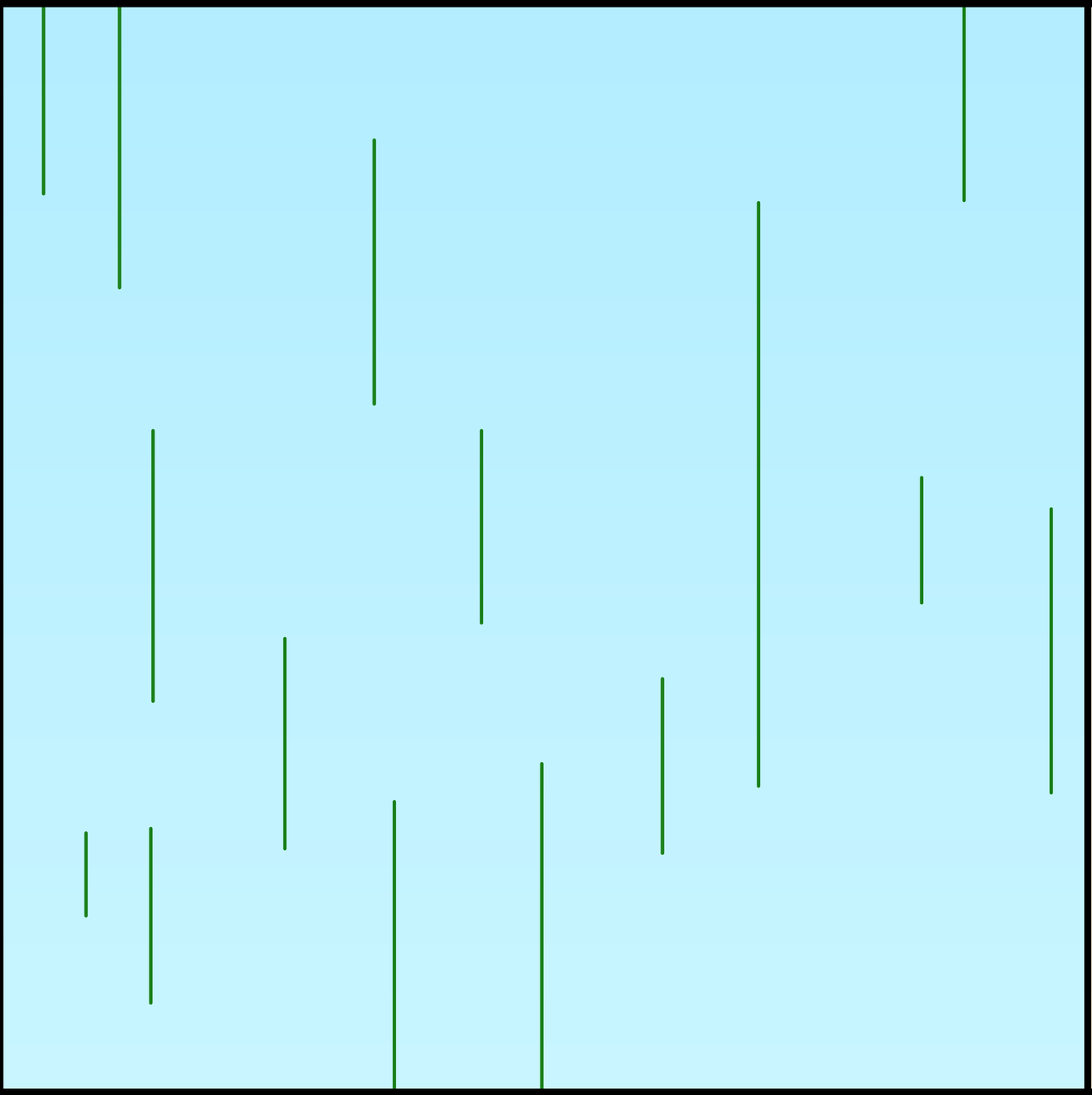
$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.

In principle, there is no reason to restrict $\mathcal{R}$ to a set of $\varepsilon$-disks.



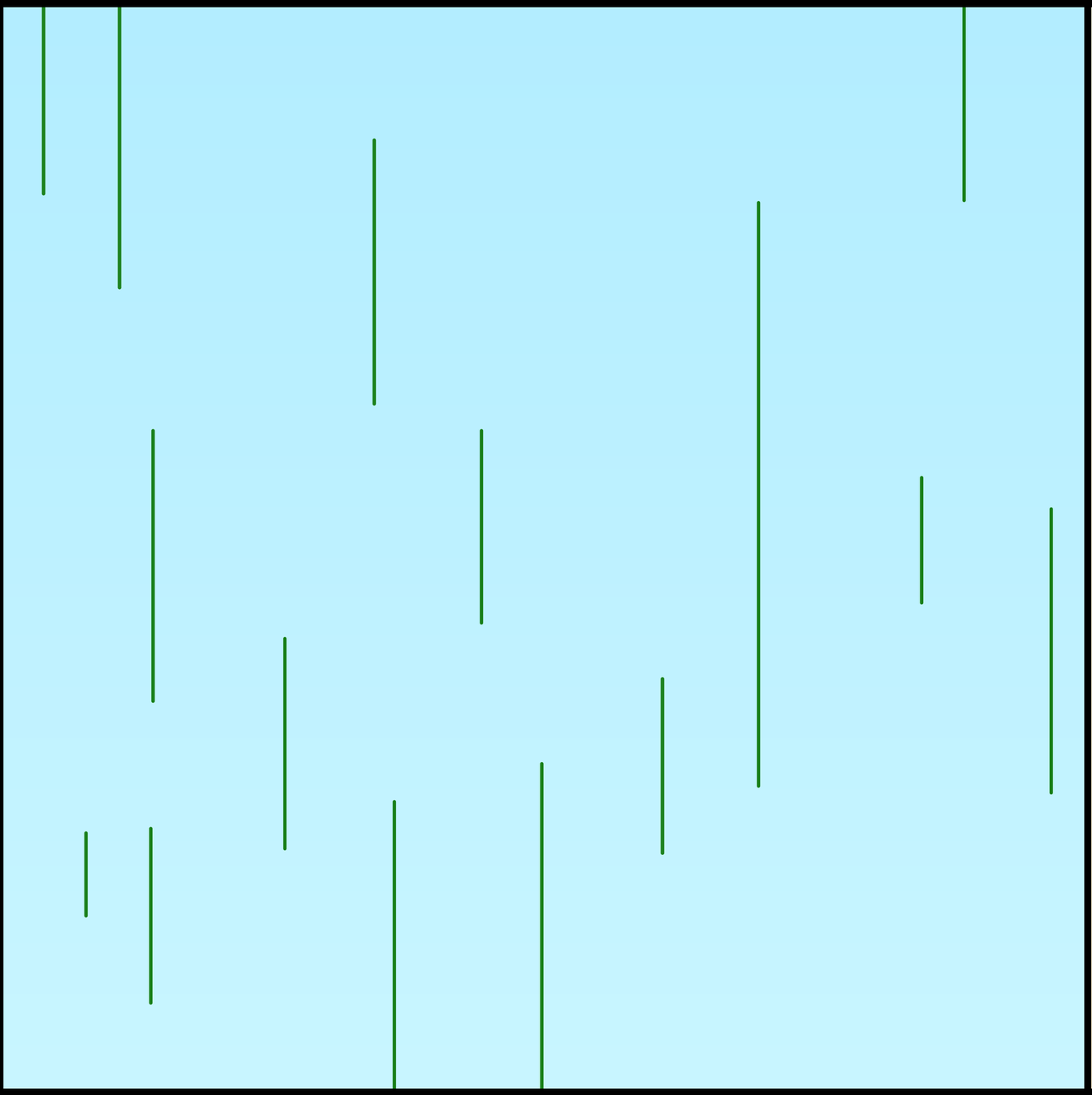
$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.

In principle, there is no reason to restrict $\mathcal{R}$ to a set of $\varepsilon$-disks.



$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.

In principle, there is no reason to restrict $\mathcal{R}$ to a set of $\varepsilon$-disks.

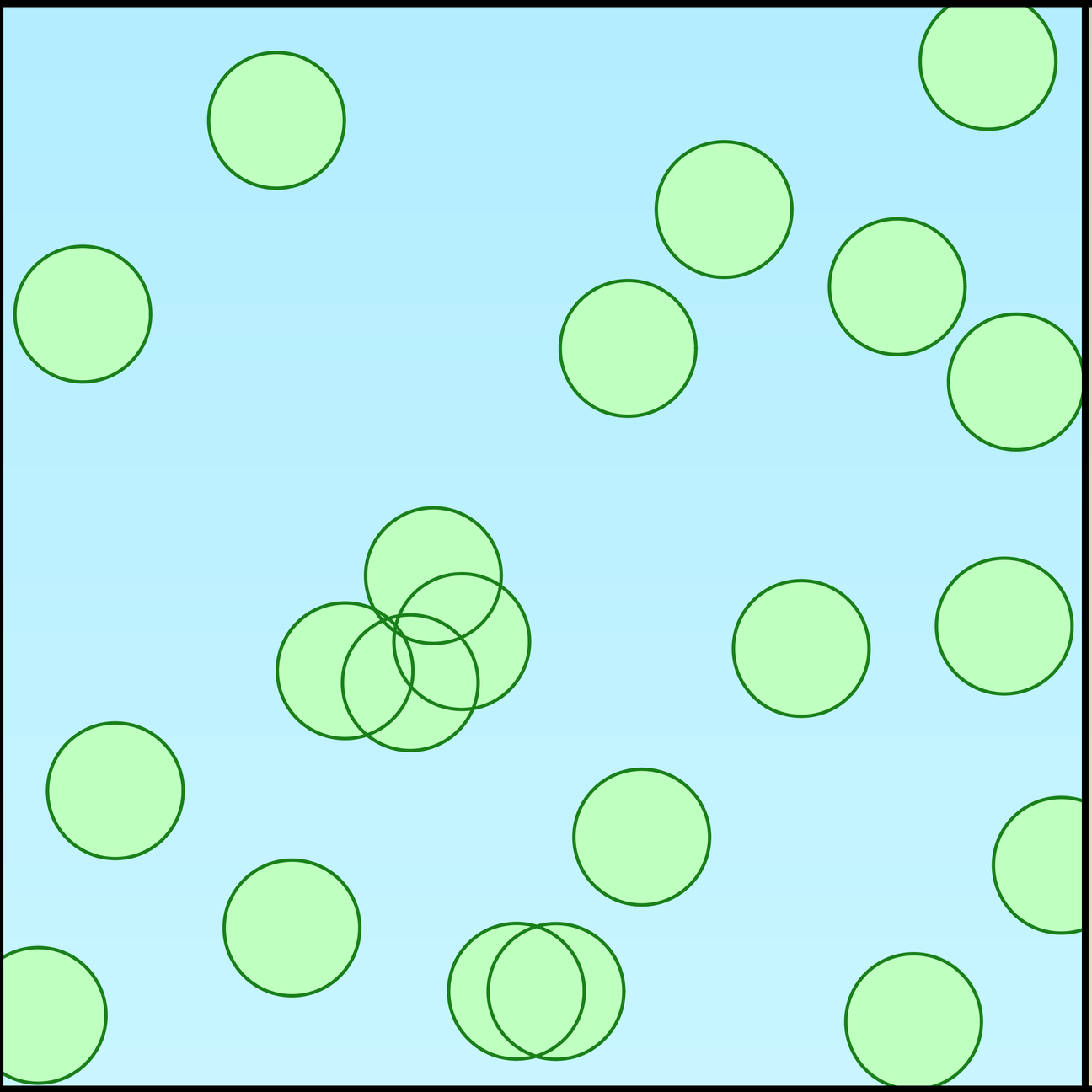


$\mathcal{R}$ is a set of *regions* which represent *partial information* about the *location* of a point set.

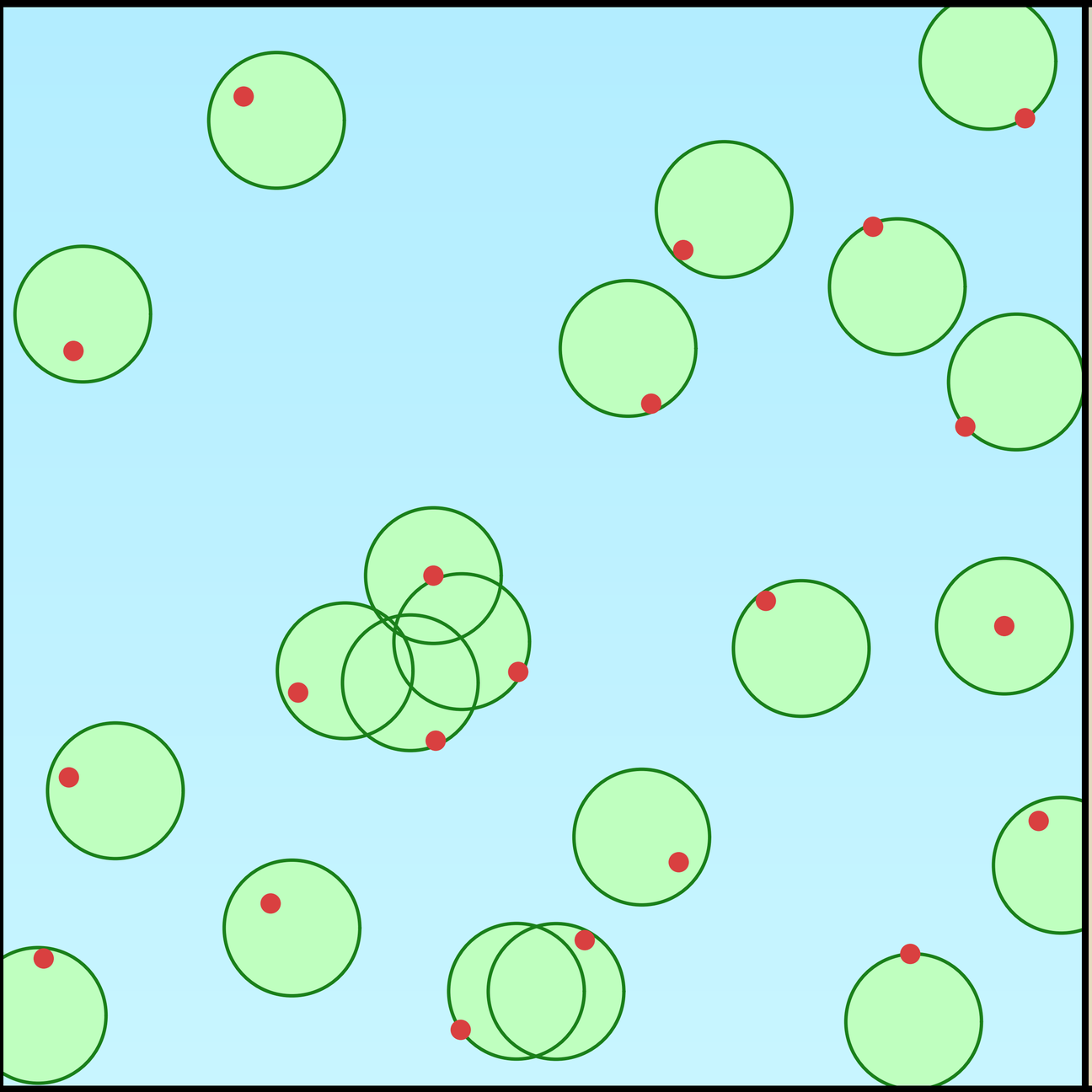
In principle, there is no reason to restrict $\mathcal{R}$ to a set of $\varepsilon$-disks.

$\mathcal{R}$ is a set of *imprecise* (or *uncertain*) points.

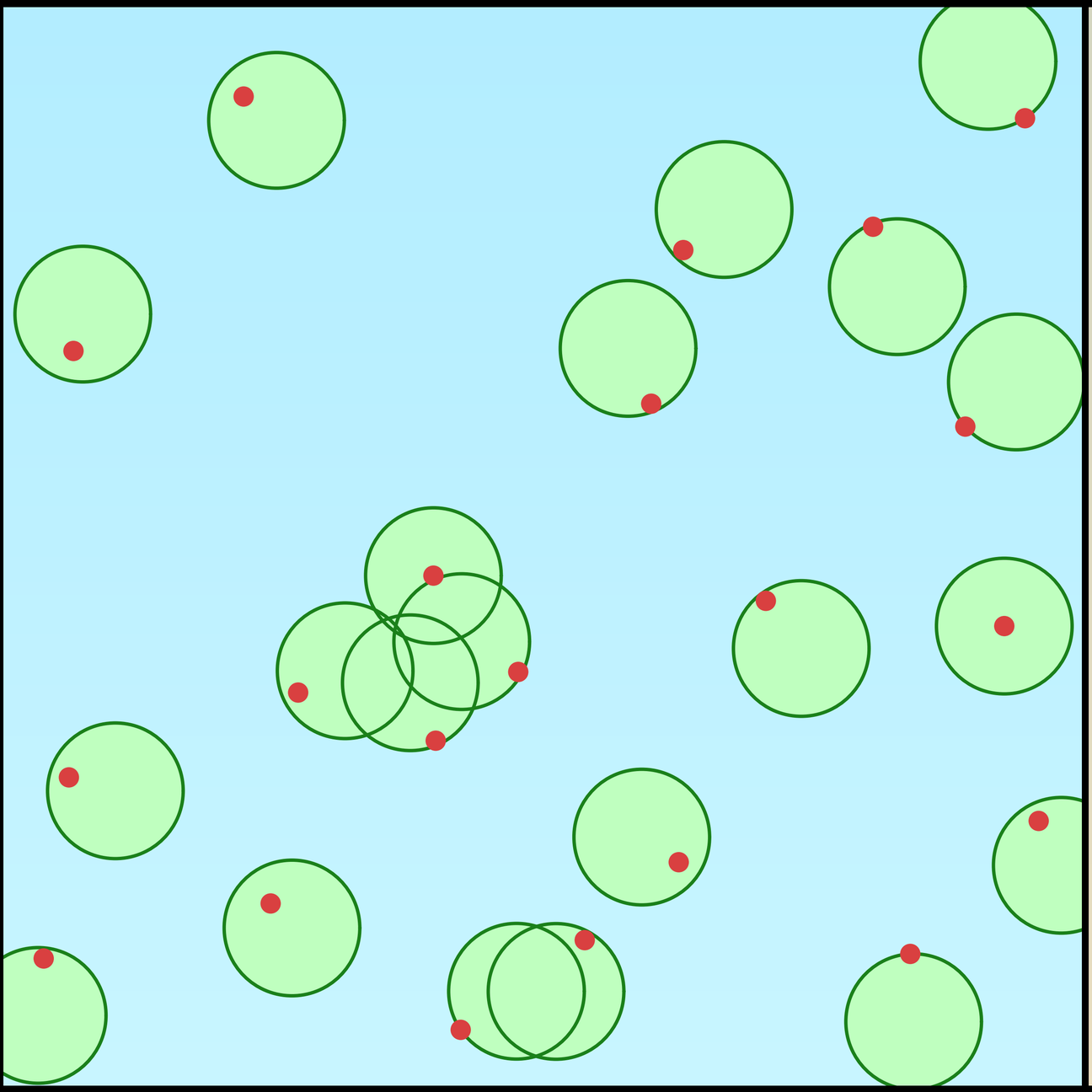
Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.



Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.

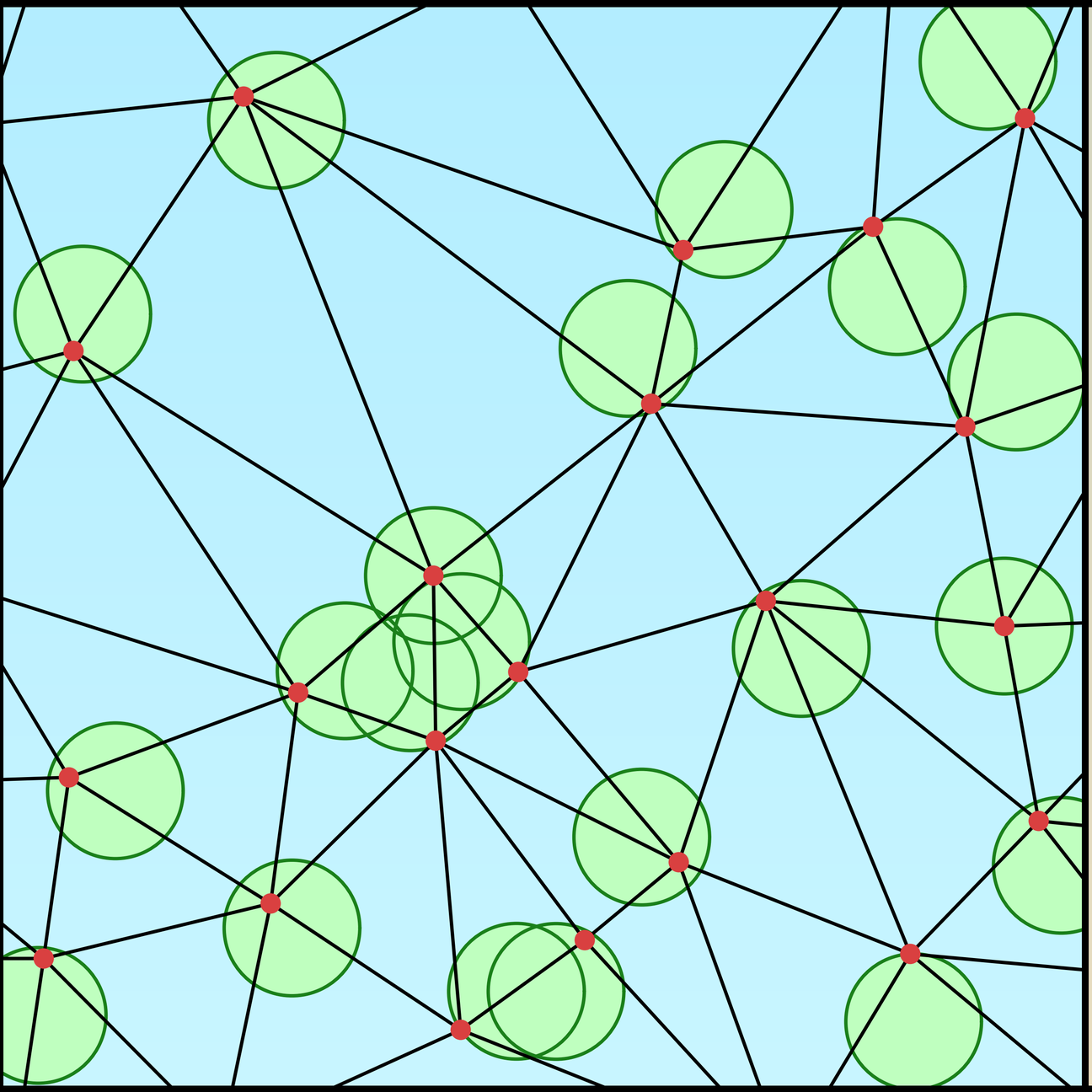


Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.



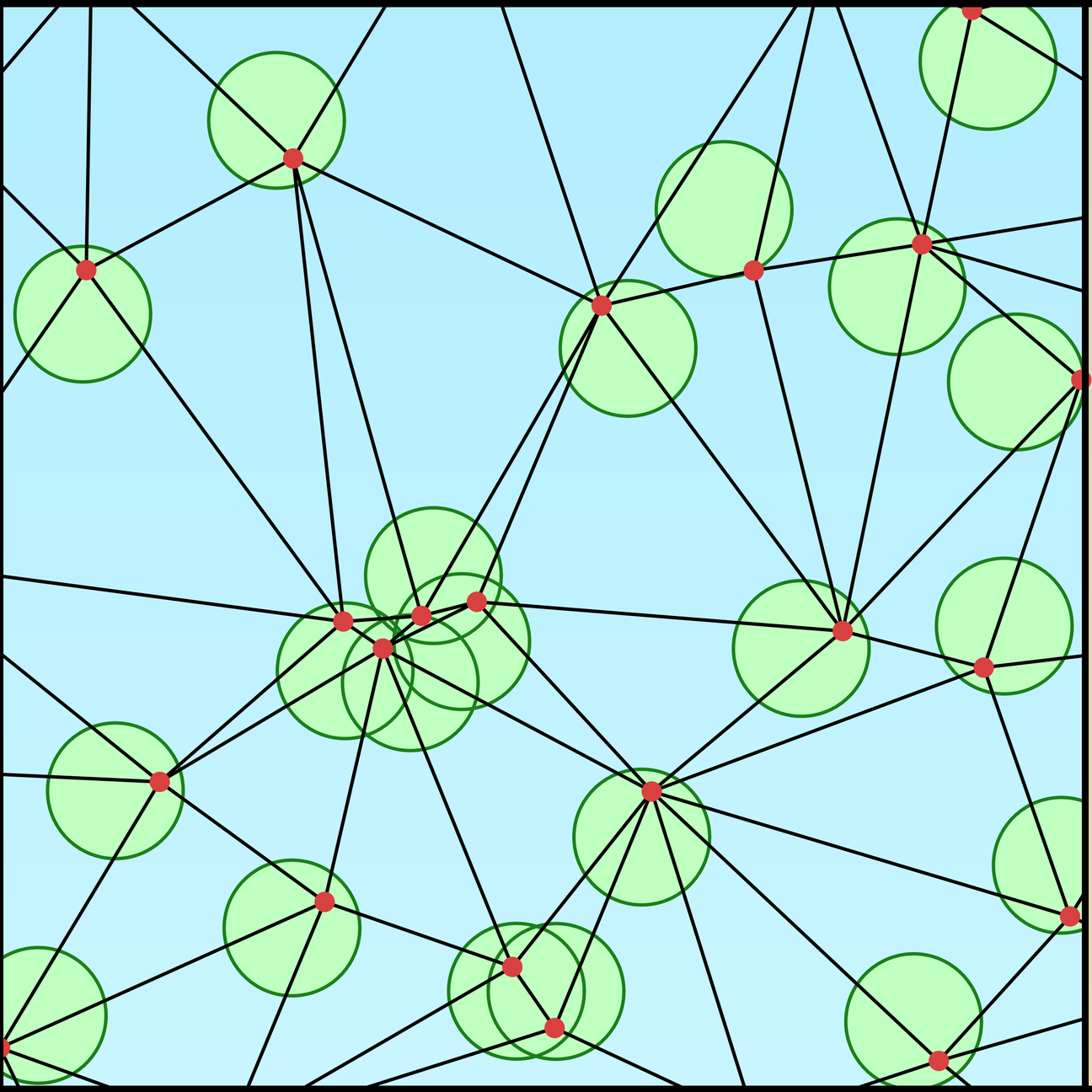
Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.

How much information about a *proximity structure* on $P$ is already present in $\mathcal{R}$?



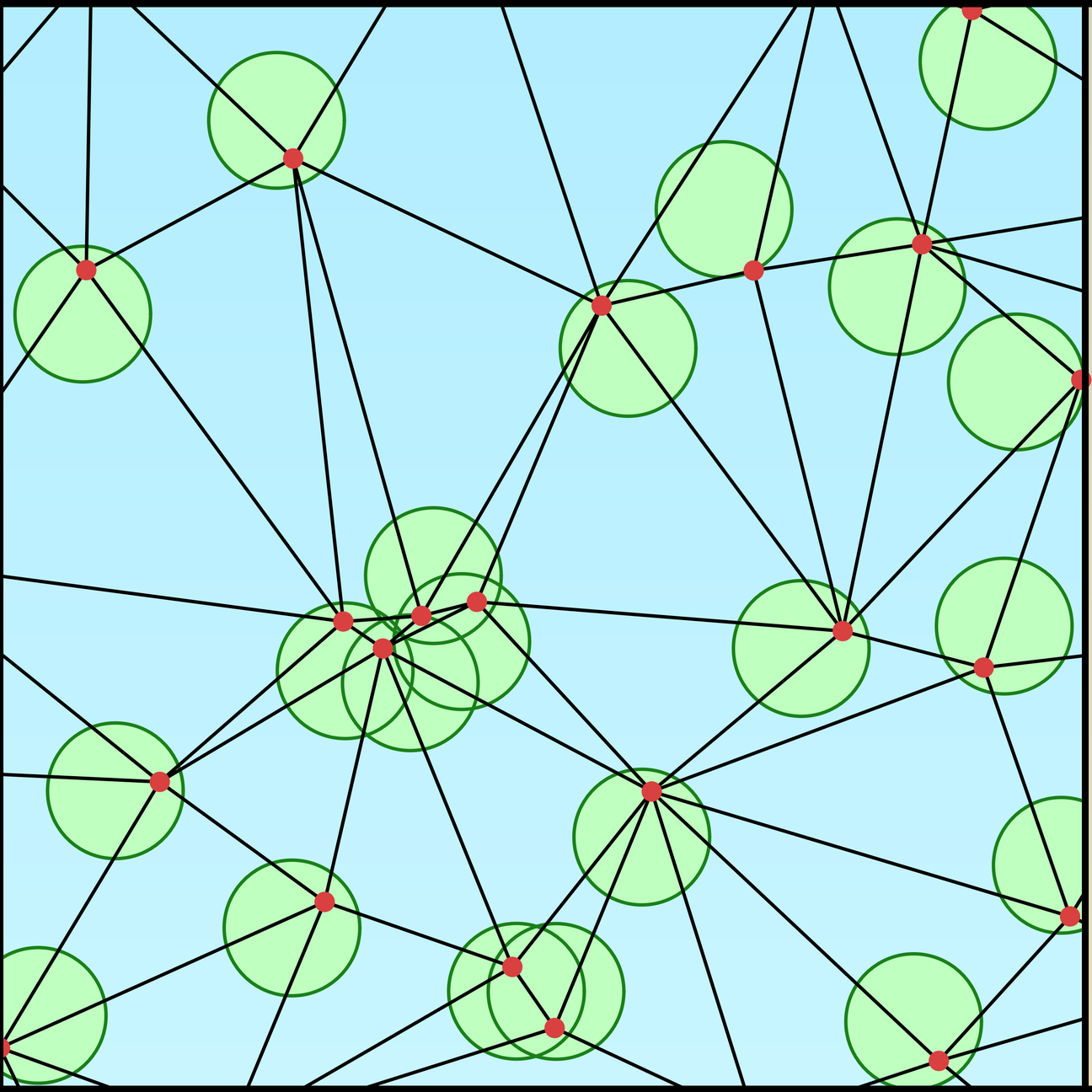
Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.

How much information about a *proximity structure* on $P$ is already present in $\mathcal{R}$?



Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.

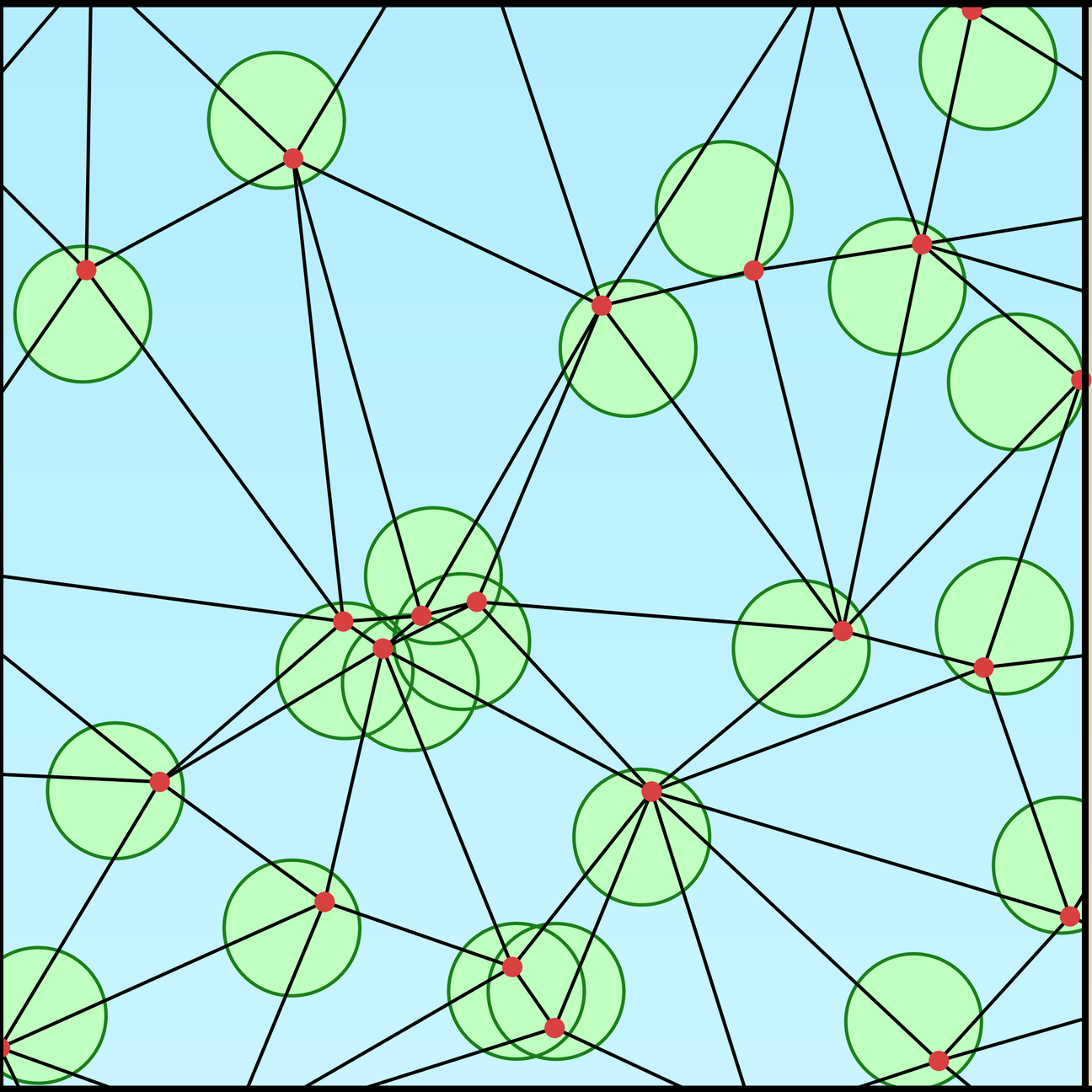
How much information about a *proximity structure* on $P$ is already present in $\mathcal{R}$?



Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.

How much information about a *proximity structure* on $P$ is already present in $\mathcal{R}$?

Intuitively, things can change a lot *locally*, but the *global* structure should not change much.



Let $\mathcal{R}$ be a set of regions representing an (unknown) point set $P$.

How much information about a *proximity structure* on $P$ is already present in $\mathcal{R}$?

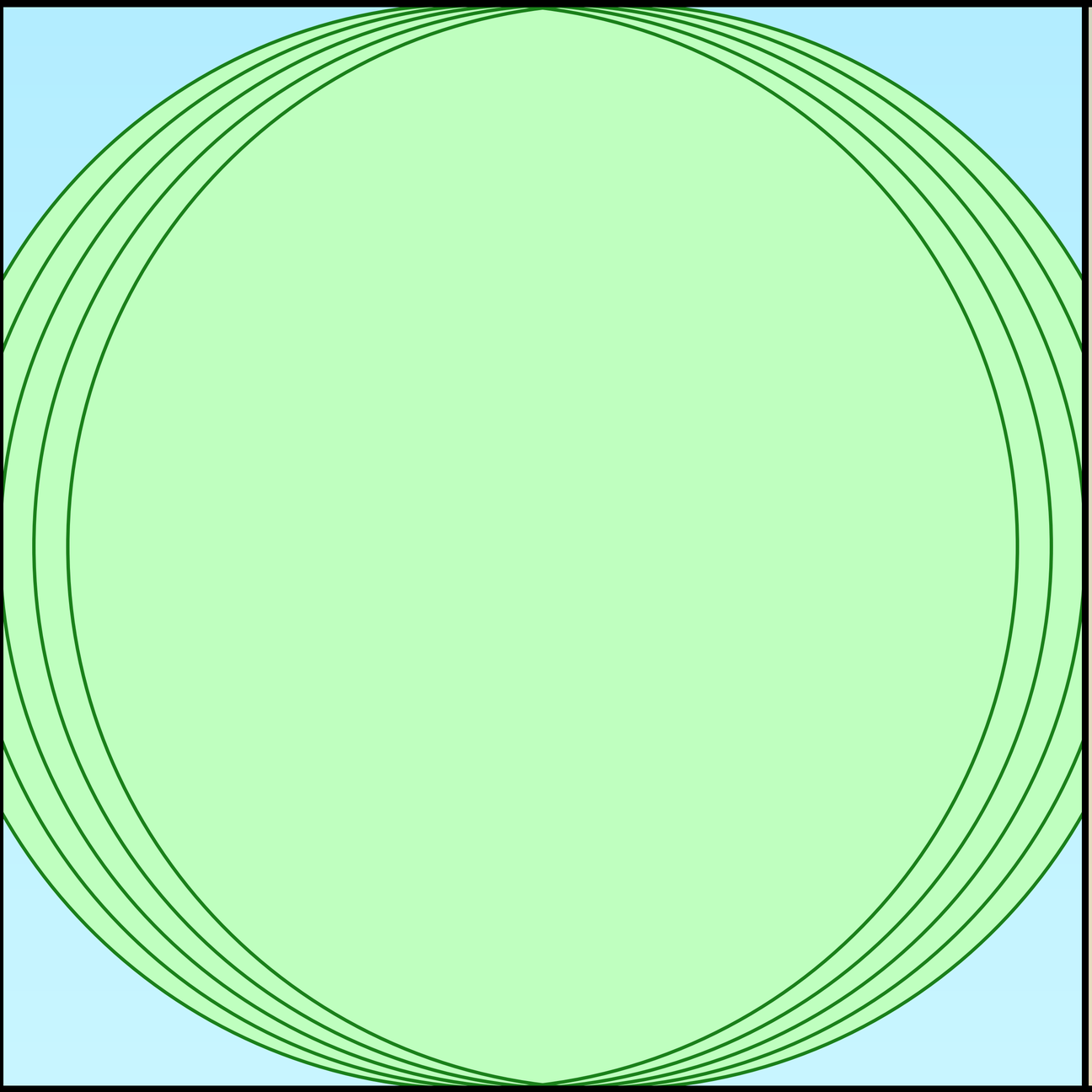
Intuitively, things can change a lot *locally*, but the *global* structure should not change much.

But proximity is local!

The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

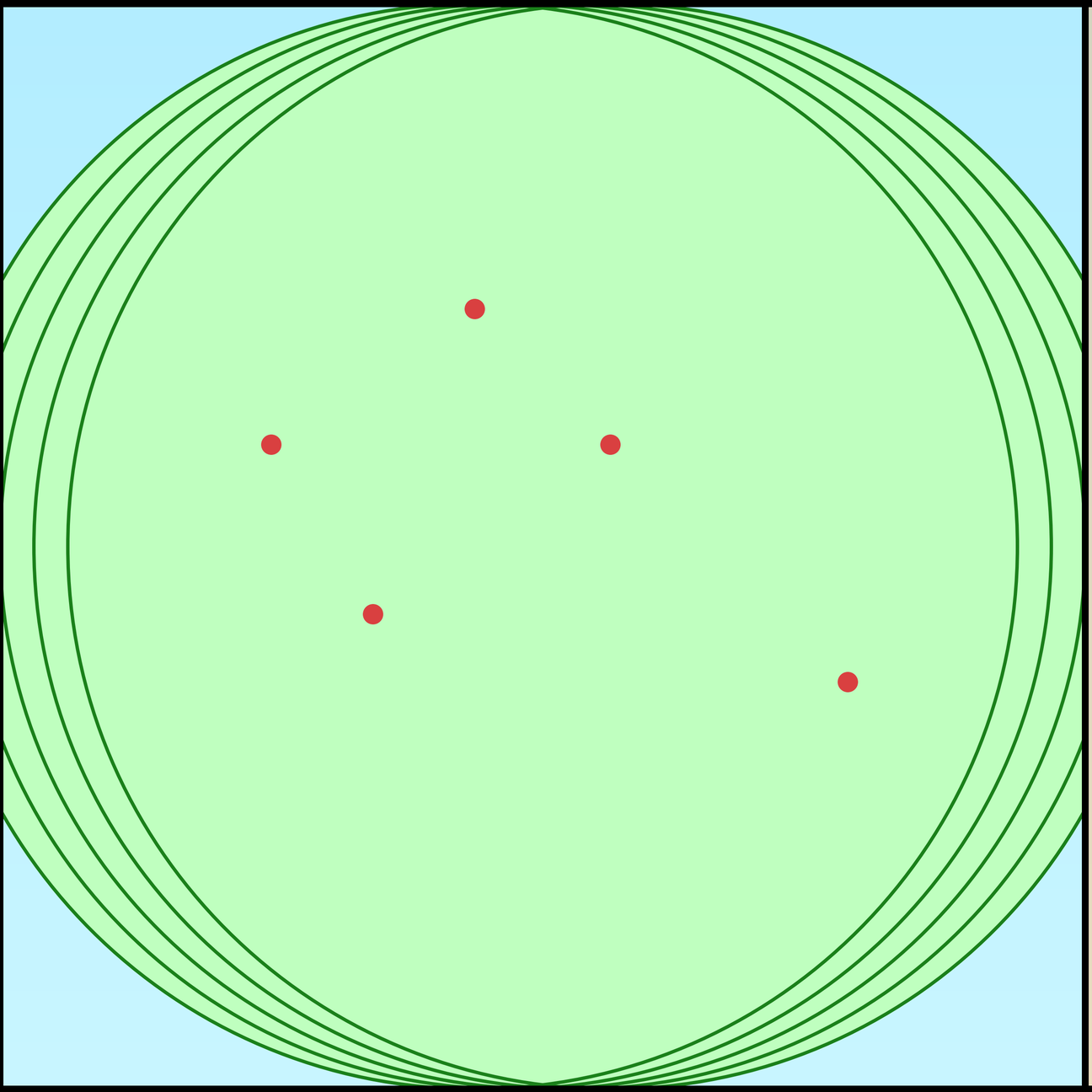
The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

If all regions in $\mathcal{R}$ overlap each other, there is no information.



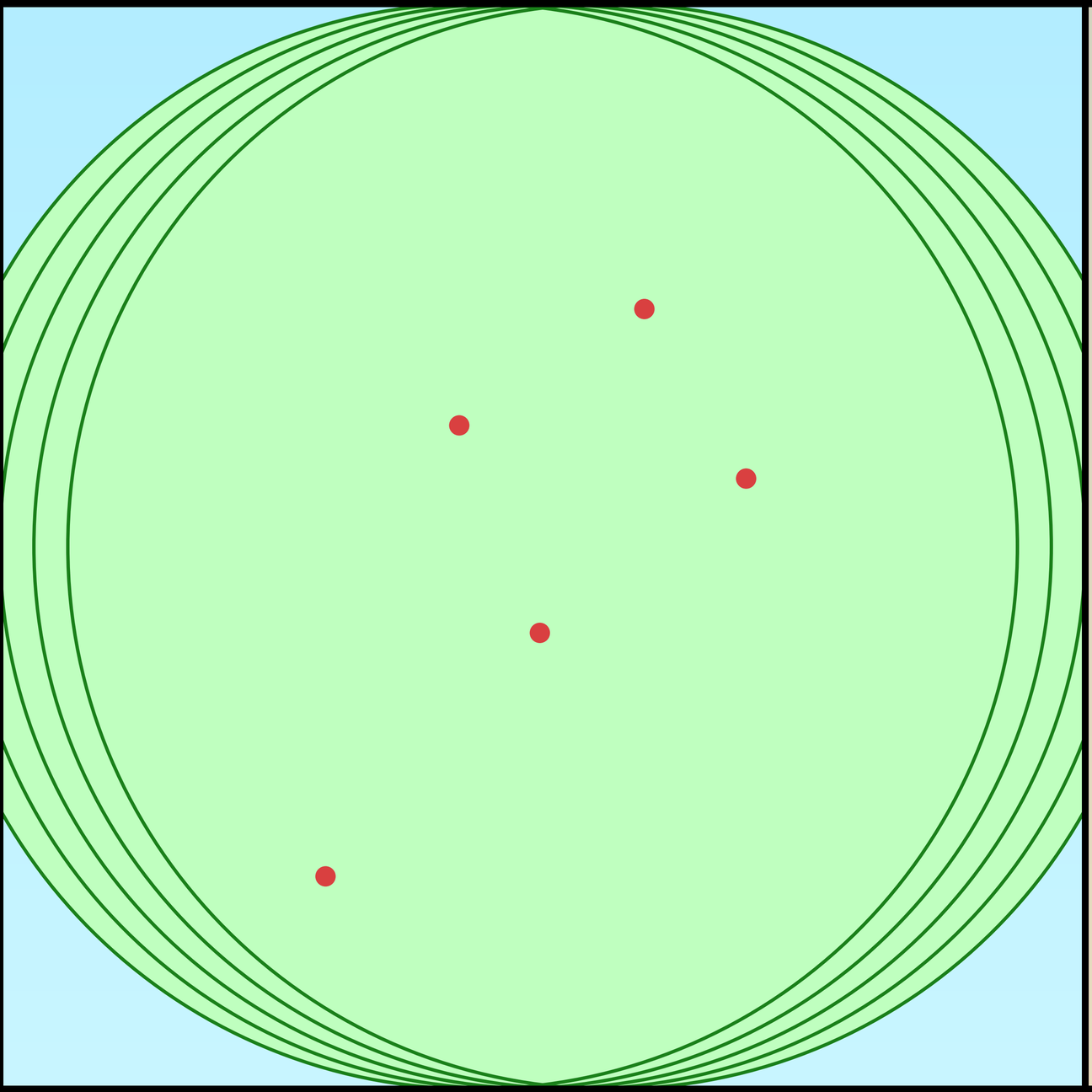
The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

If all regions in $\mathcal{R}$ overlap each other, there is no information.



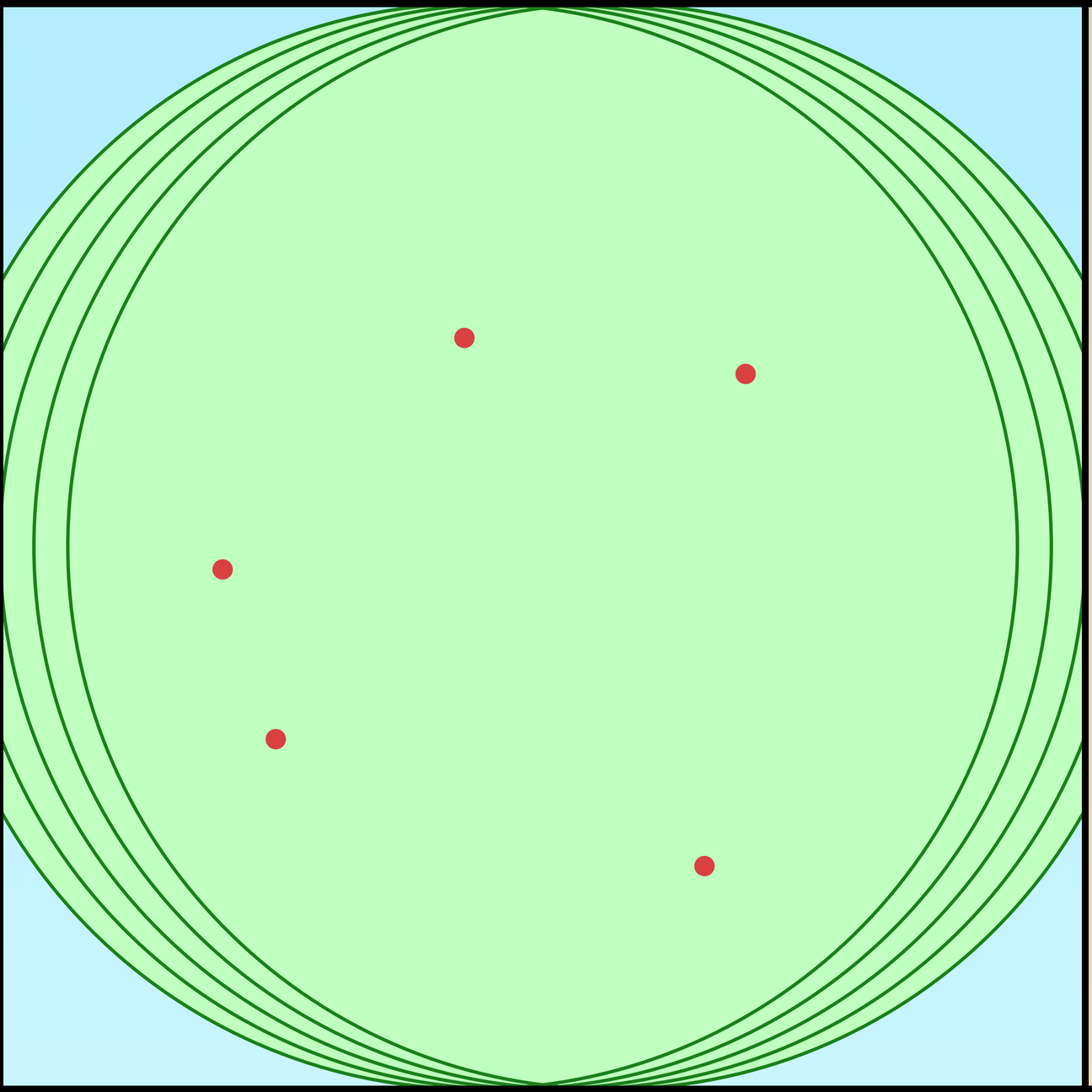
The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

If all regions in $\mathcal{R}$ overlap each other, there is no information.



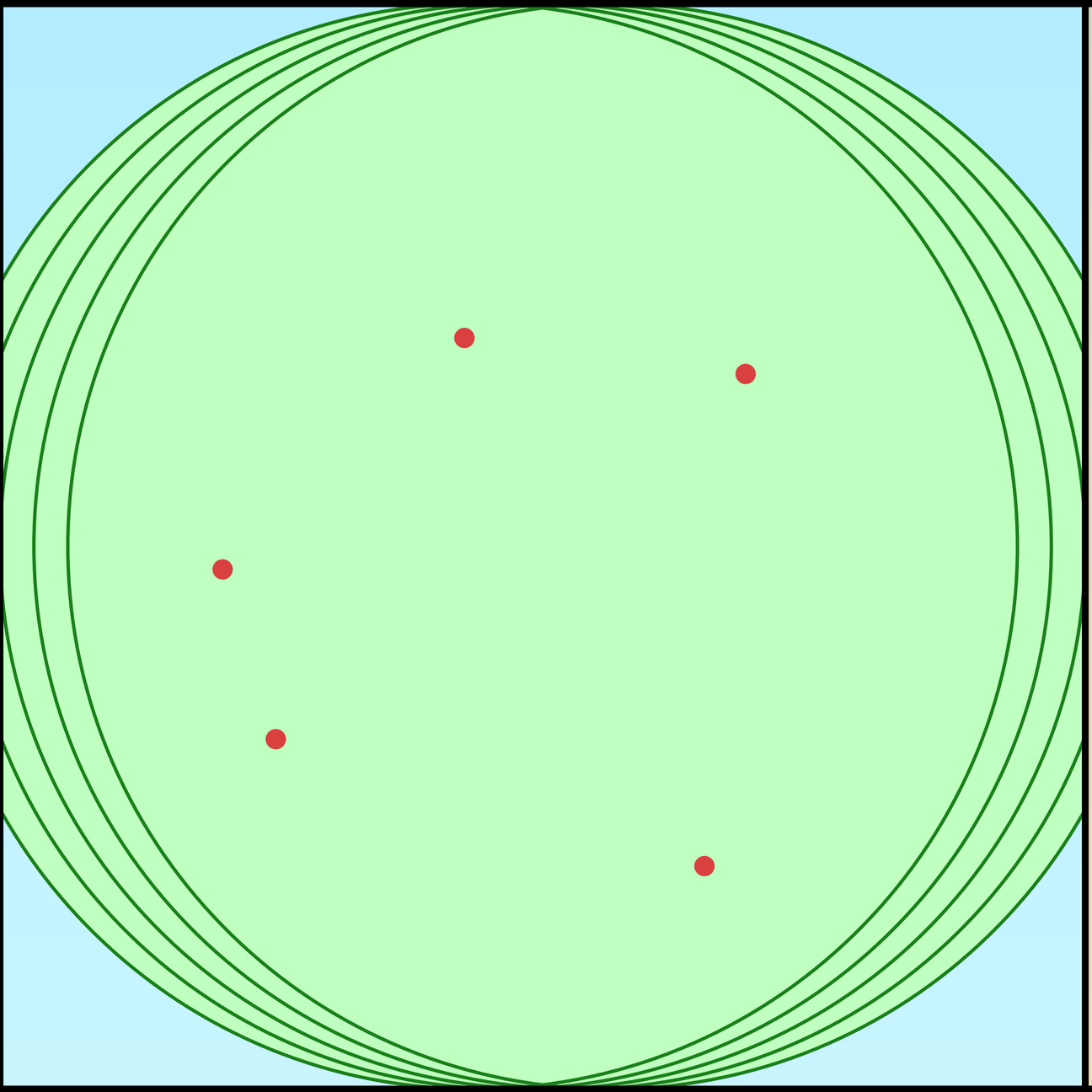
The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

If all regions in $\mathcal{R}$ overlap each other, there is no information.



The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

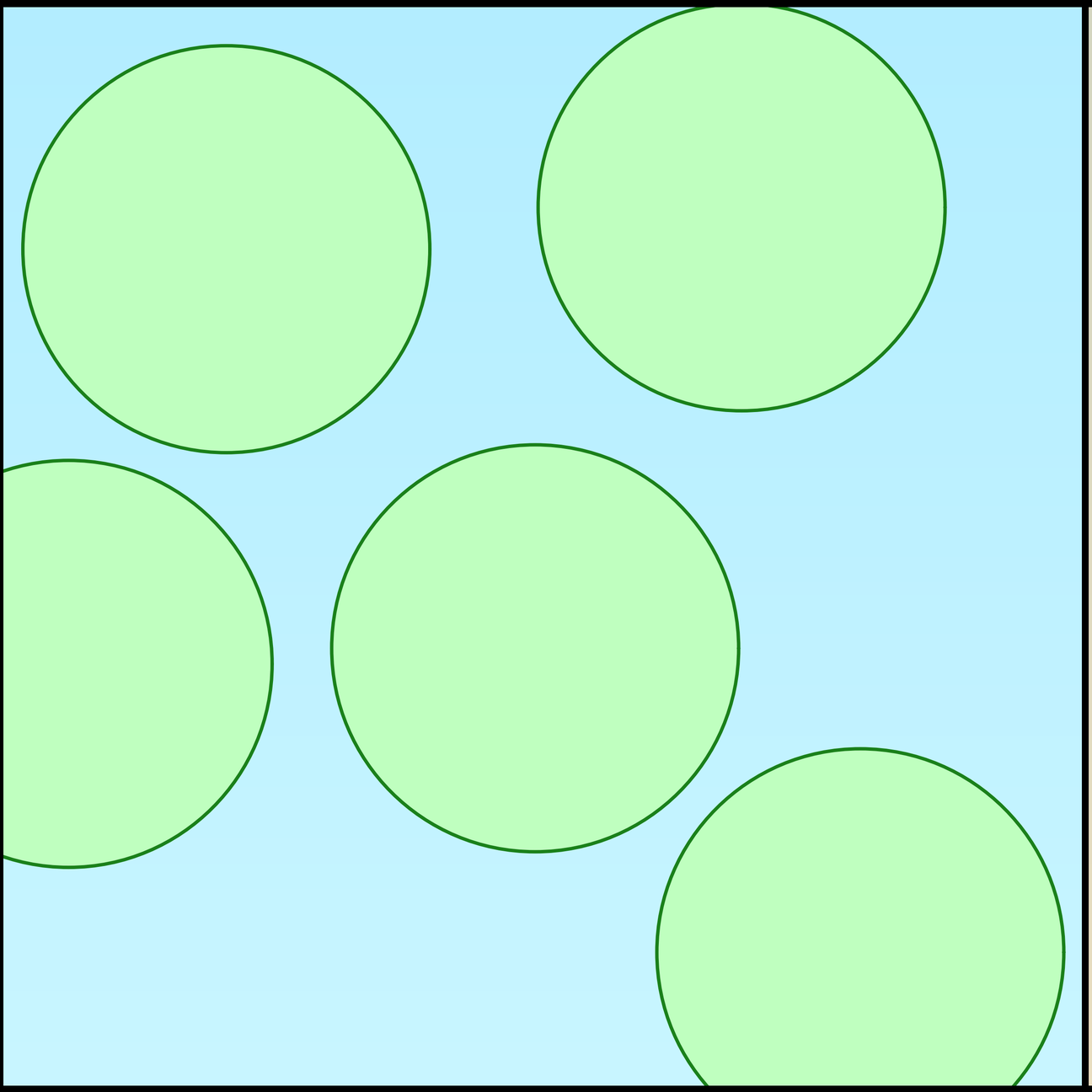
If all regions in $\mathcal{R}$ overlap each other, there is no information.



The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

If all regions in $\mathcal{R}$ overlap each other, there is no information.

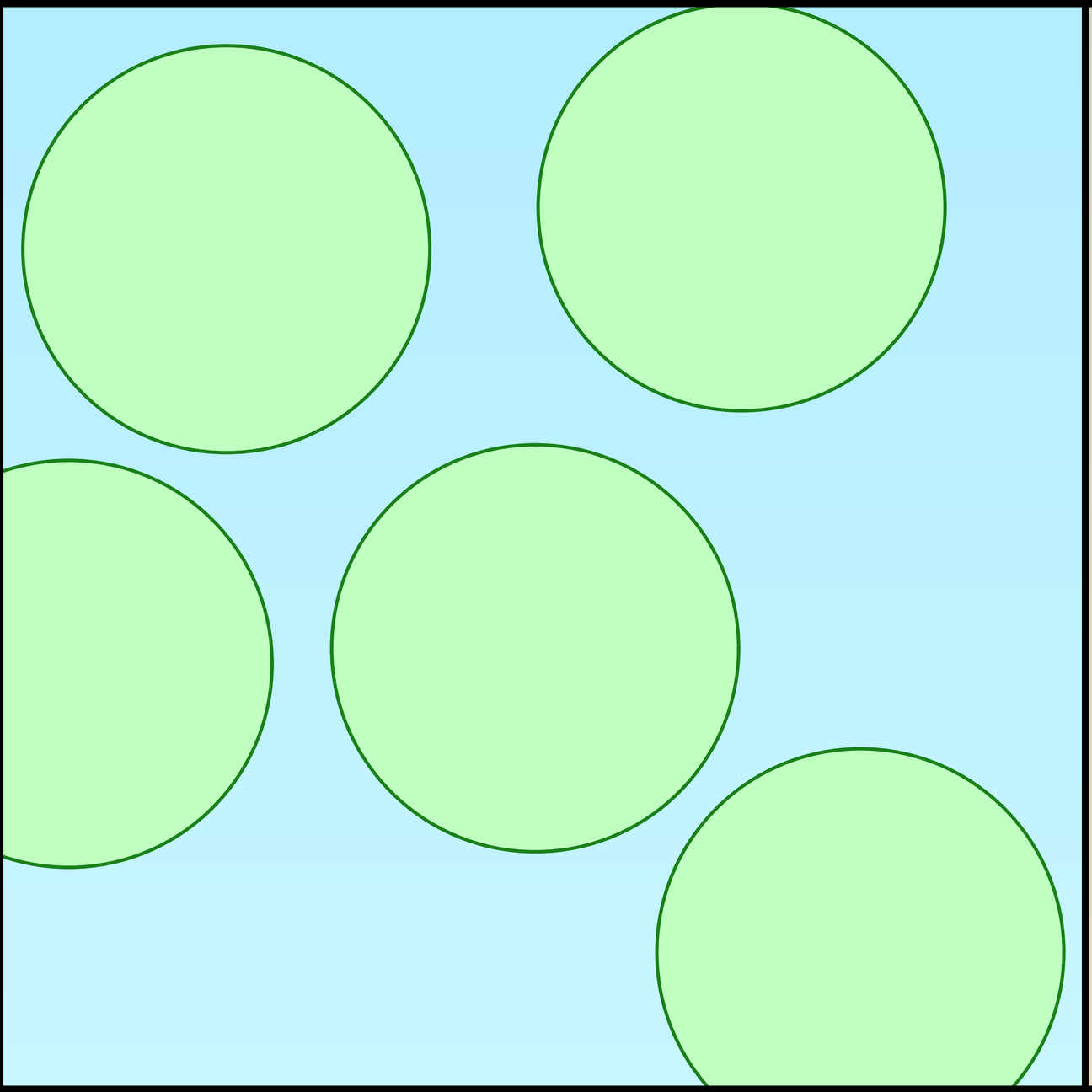
What if the regions are *disjoint*?



The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

If all regions in $\mathcal{R}$ overlap each other, there is no information.

What if the regions are *disjoint*?



The answer may depend on what proximity structure we want, but also on the class of regions $\mathcal{R}$.

If all regions in $\mathcal{R}$ overlap each other, there is no information.

What if the regions are *disjoint*?

How can we even formalize this question?

# Chapter 2
*PARTIAL INFORMATION*

Uncertain locations

Quantifying information

State of the art in 1D

State of the art in 2D

We can capture *information* in terms of *computation*.

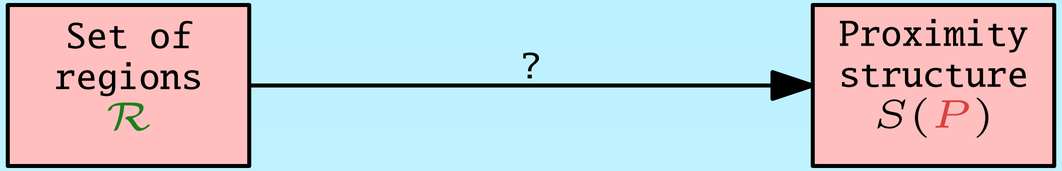
Set of regions $\mathcal{R}$

We can capture *information* in terms of *computation*.

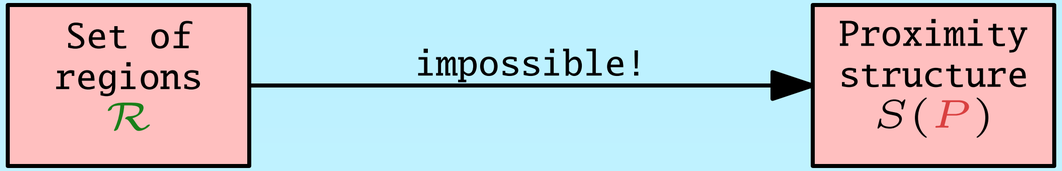
Set of regions $\mathcal{R}$

Proximity structure $S(P)$

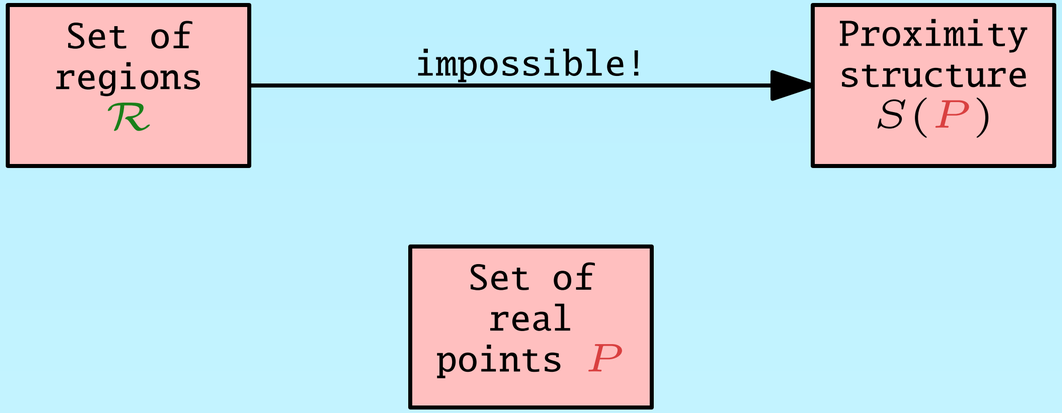
We can capture *information* in terms of *computation*.



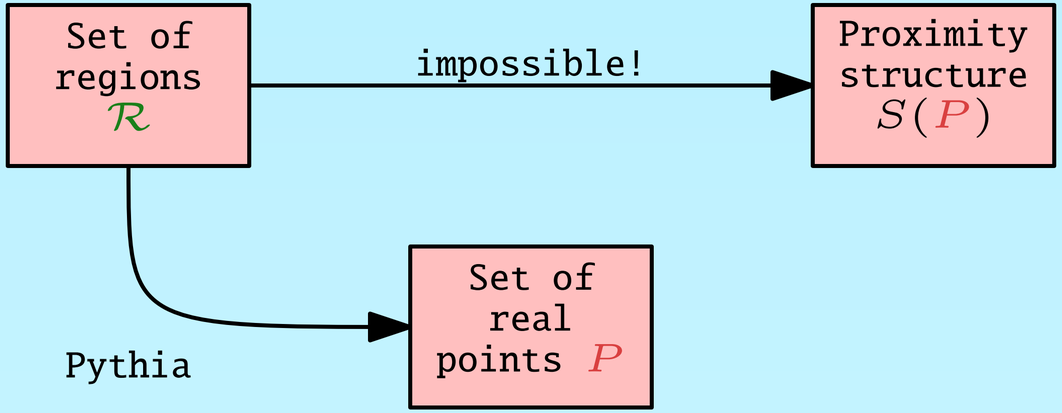

We can capture *information* in terms of *computation*.



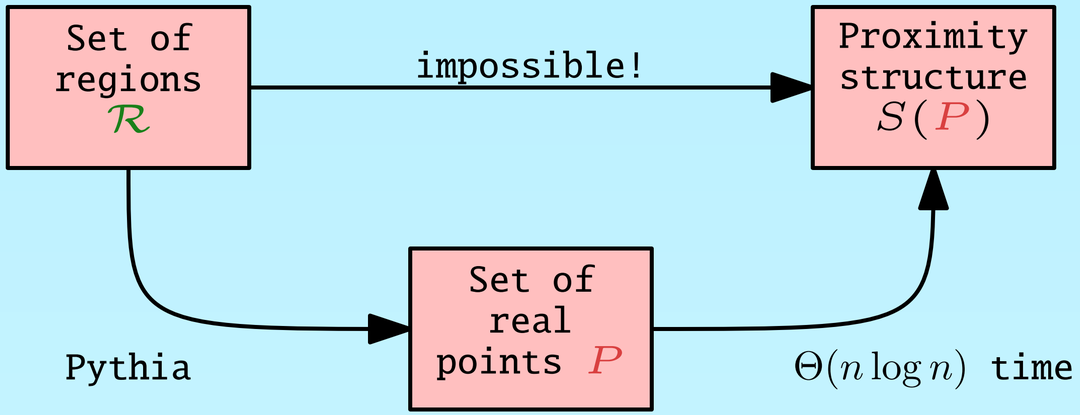

We can capture *information* in terms of *computation*.



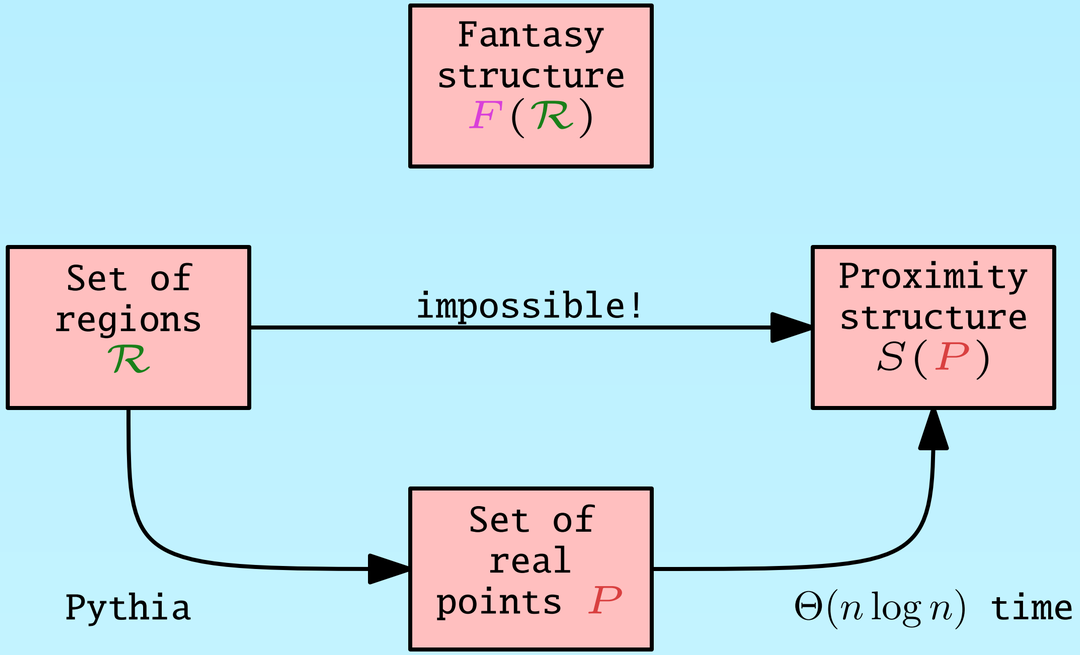

We can capture *information* in terms of *computation*.



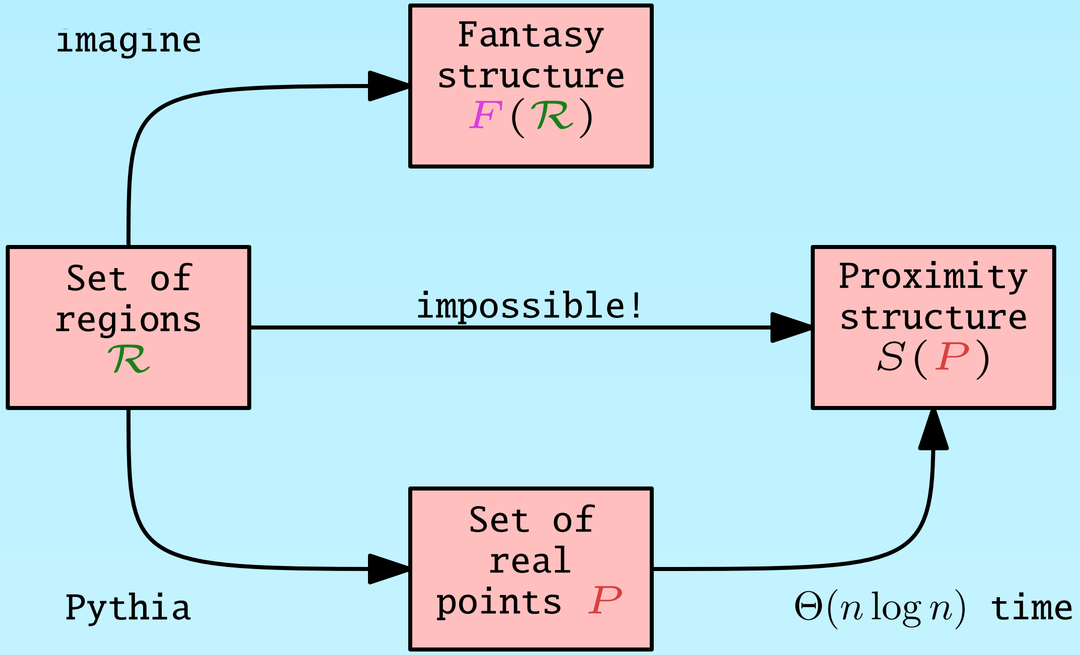

We can capture *information* in terms of *computation*.



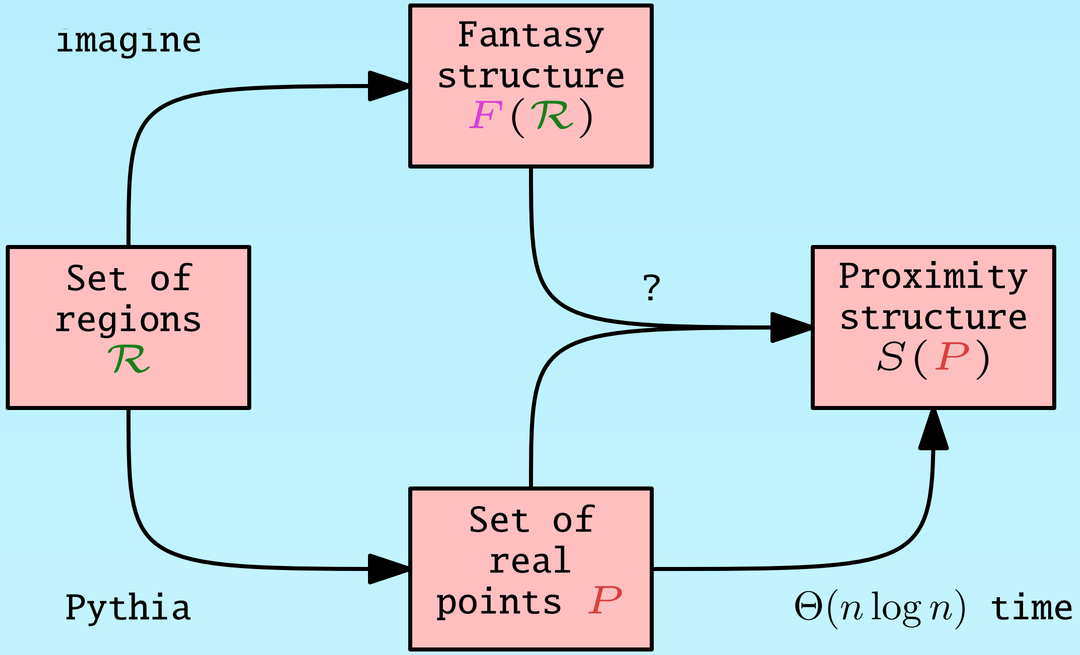

We can capture *information* in terms of *computation*.



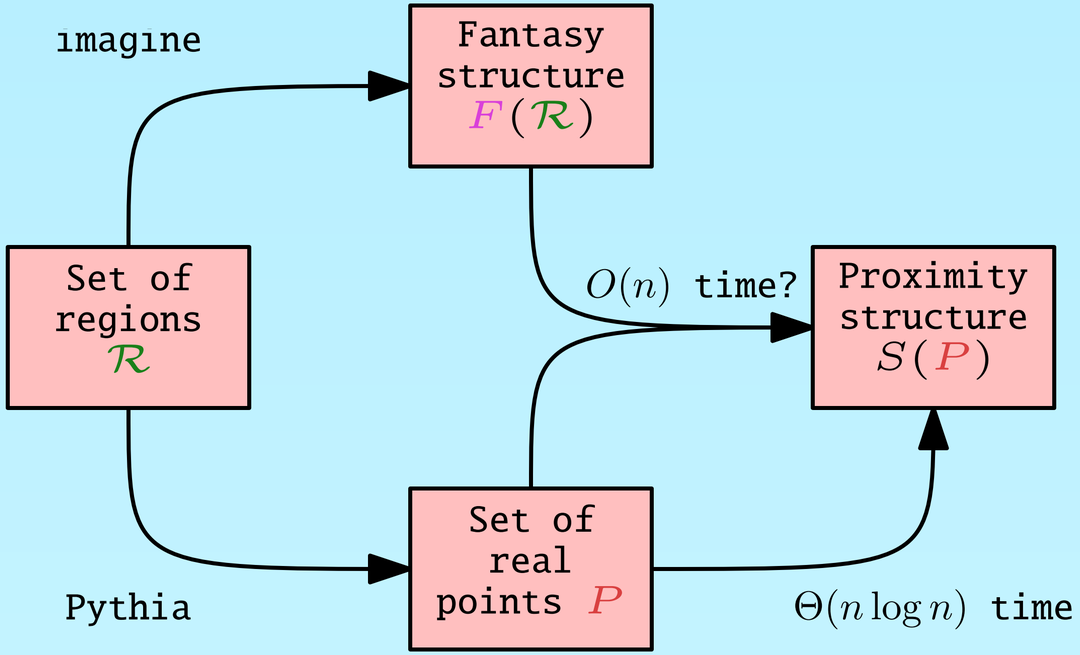

We can capture *information* in terms of *computation*.



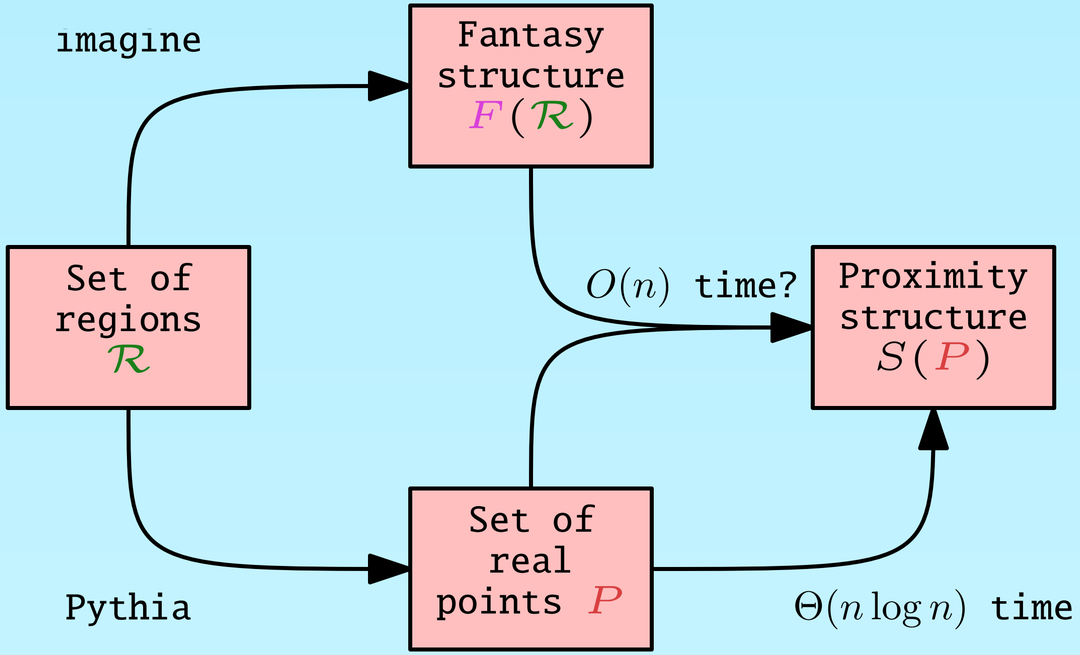

We can capture *information* in terms of *computation*.




We can capture *information* in terms of *computation*.

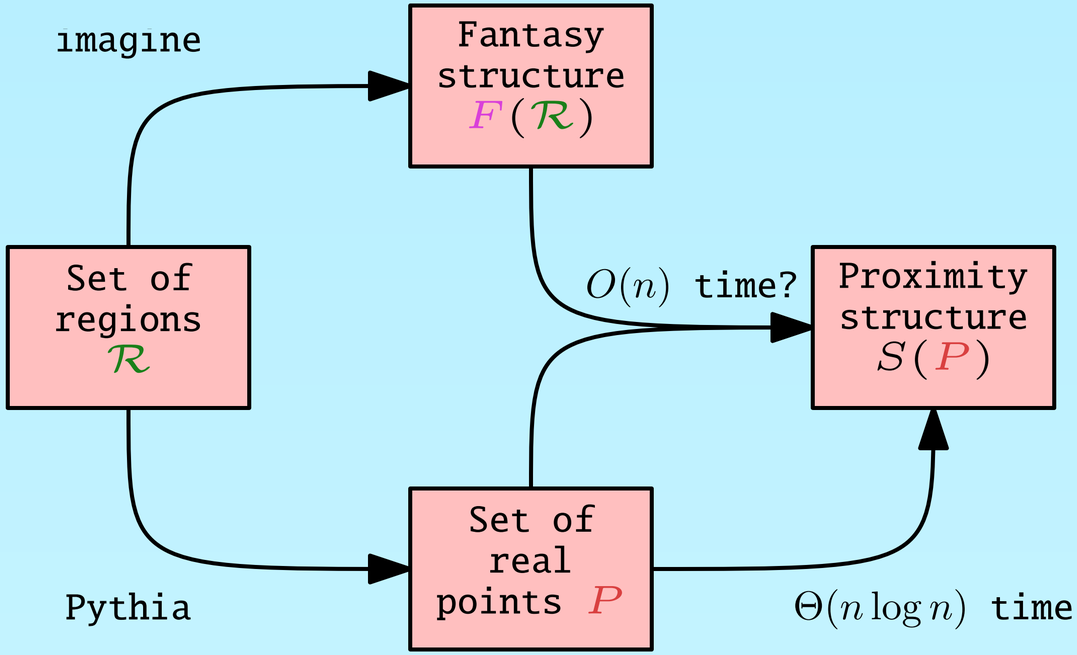



We can capture *information* in terms of *computation*.




We can capture *information* in terms of *computation*.

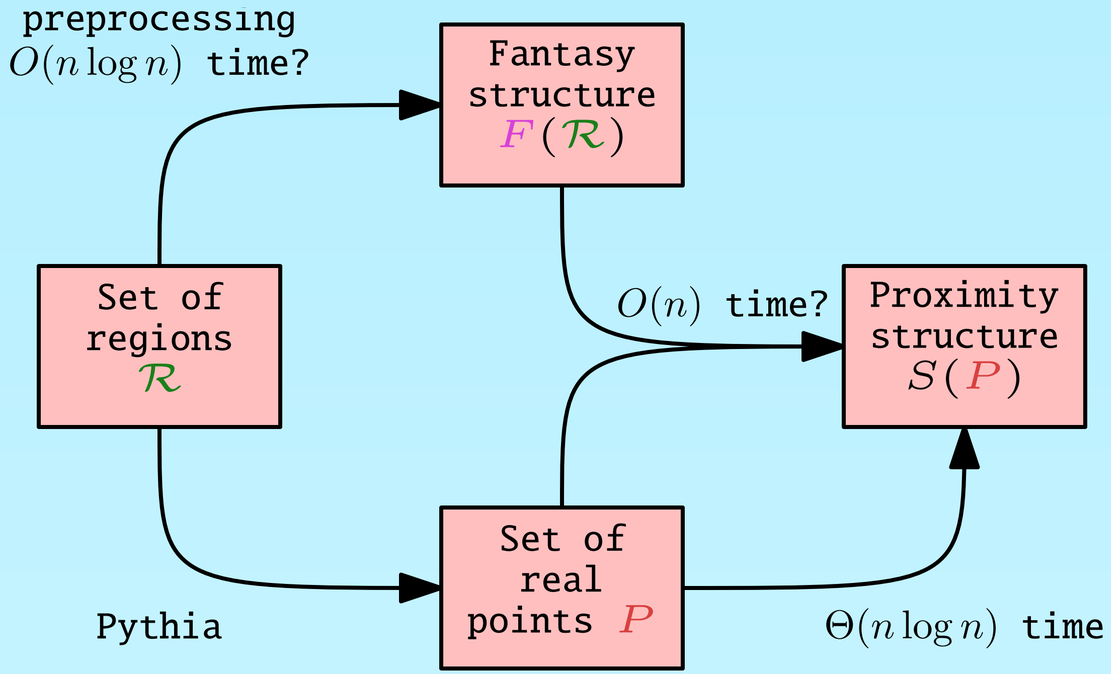
We can say $\mathcal{R}$ contains the information of $S(P)$ if there *exists* a fantasy structure $F(\mathcal{R})$.




We can capture *information* in terms of *computation*.

We can say $\mathcal{R}$ contains the information of $S(P)$ if there *exists* a fantasy structure $F(\mathcal{R})$.

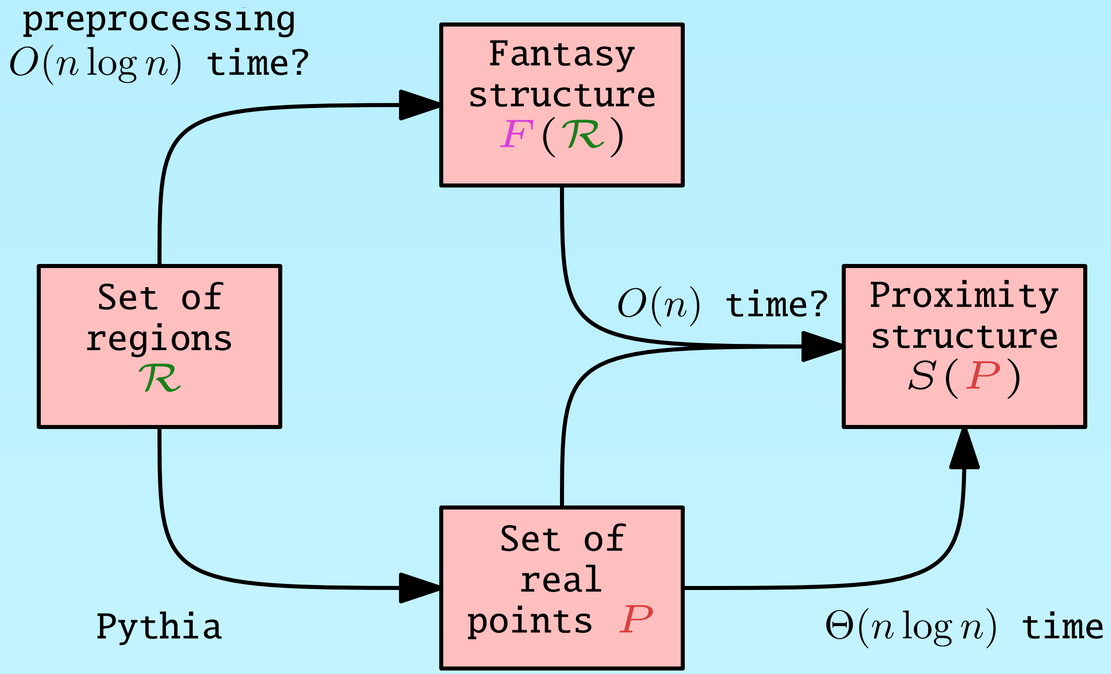
We compute $F(\mathcal{R})$ in a *preprocessing* phase.




We can capture *information* in terms of *computation*.

We can say $\mathcal{R}$ contains the information of $S(P)$ if there *exists* a fantasy structure $F(\mathcal{R})$.

We compute $F(\mathcal{R})$ in a *preprocessing* phase.

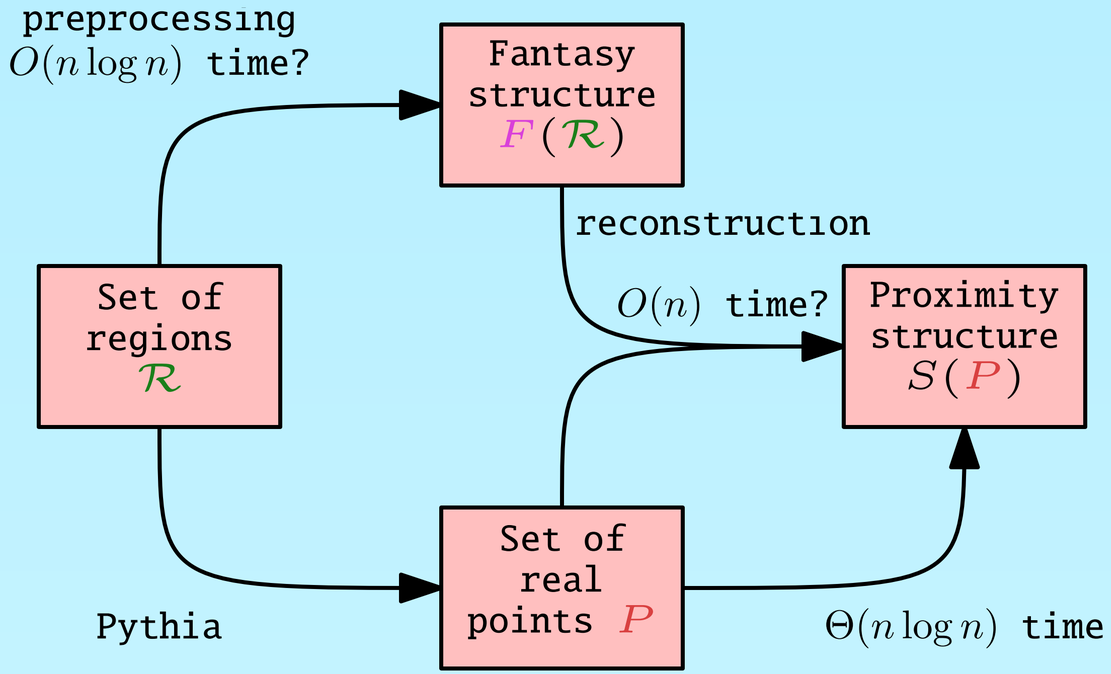



We can capture *information* in terms of *computation*.

We can say $\mathcal{R}$ contains the information of $S(P)$ if there *exists* a fantasy structure $F(\mathcal{R})$.

We compute $F(\mathcal{R})$ in a *preprocessing* phase.

We compute $S(P)$ in a *reconstruction* phase.



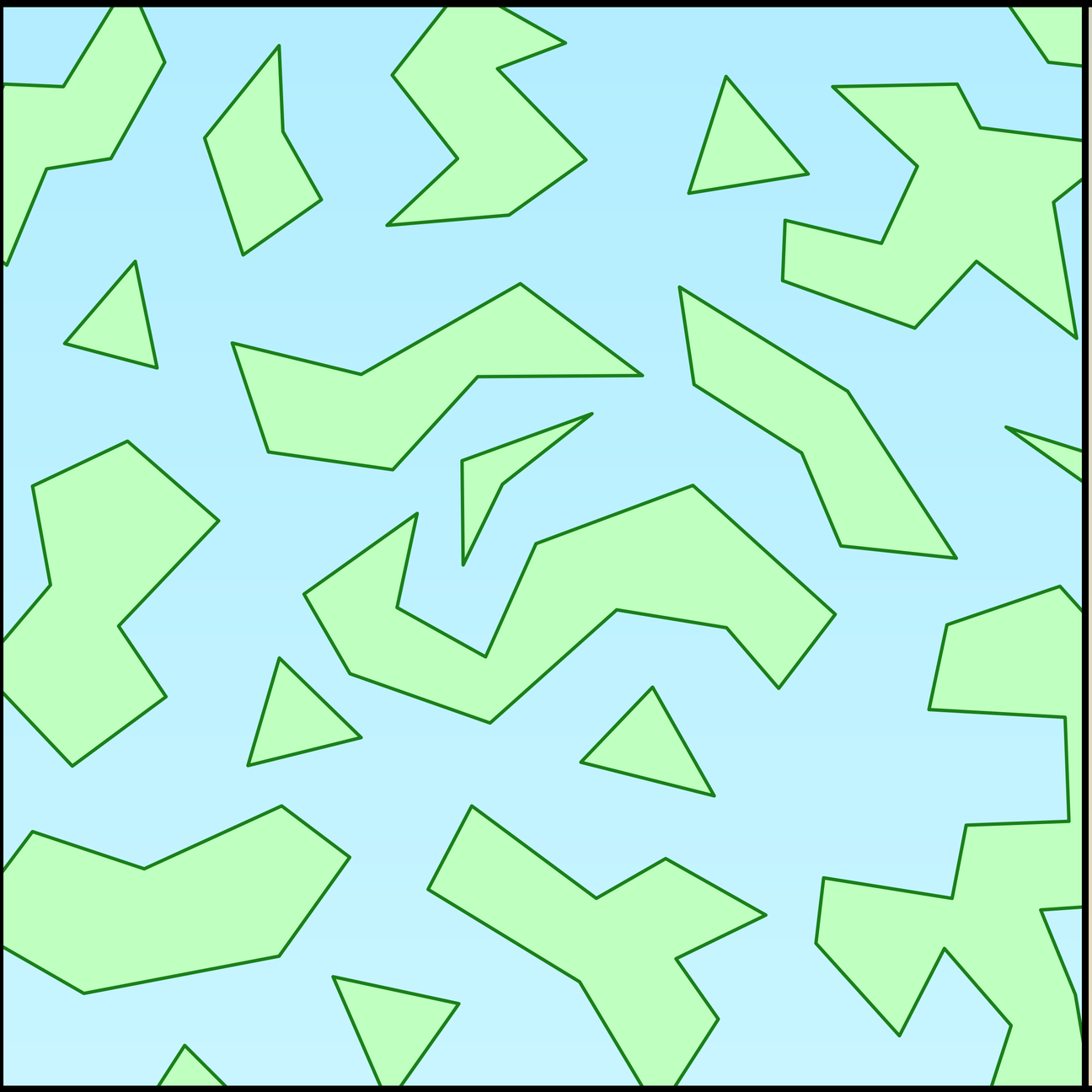

We can capture *information* in terms of *computation*.

We can say $\mathcal{R}$ contains the information of $S(P)$ if there *exists* a fantasy structure $F(\mathcal{R})$.

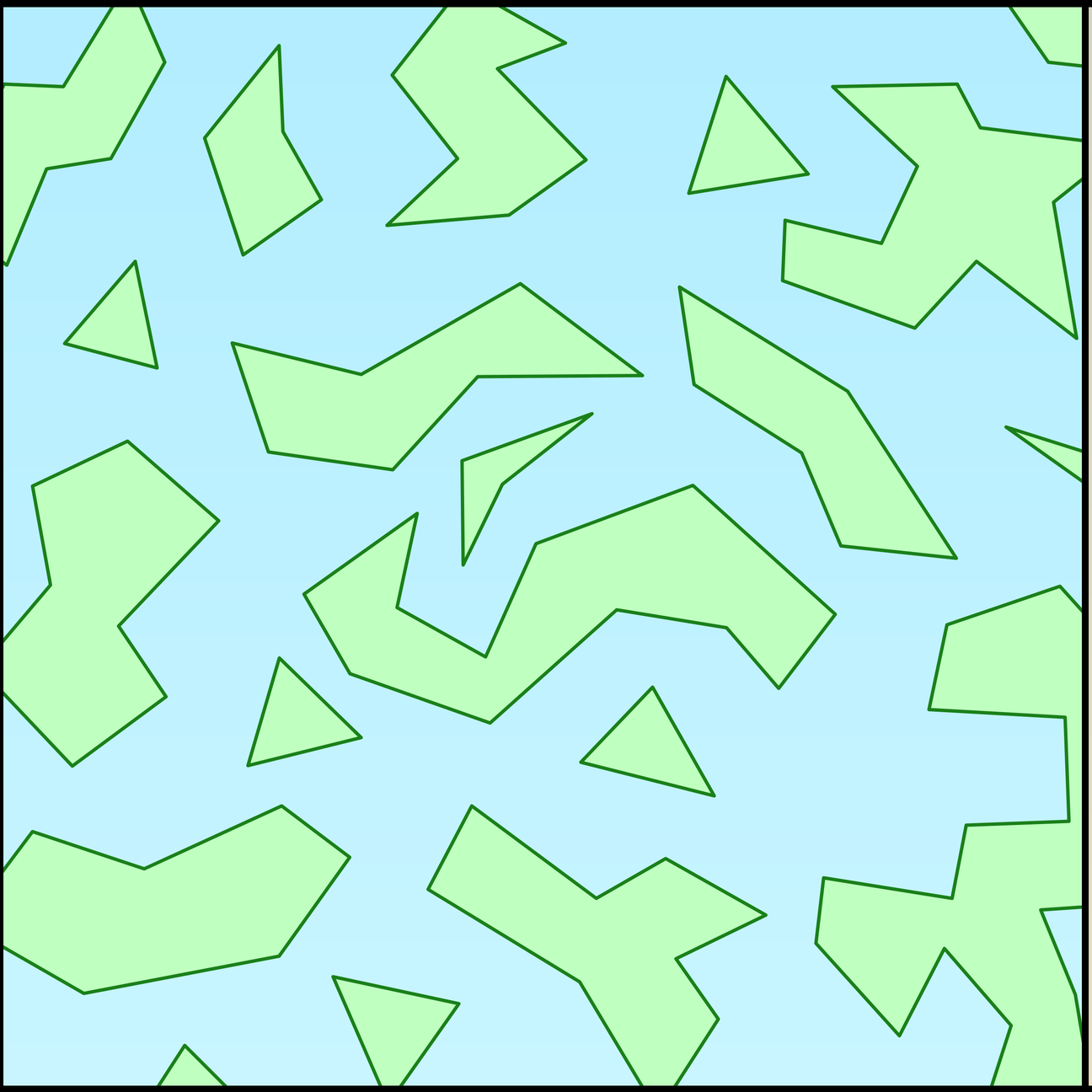
We compute $F(\mathcal{R})$ in a *preprocessing* phase.

We compute $S(P)$ in a *reconstruction* phase.

For example, let $\mathcal{R}$ be a set of disjoint polygons.

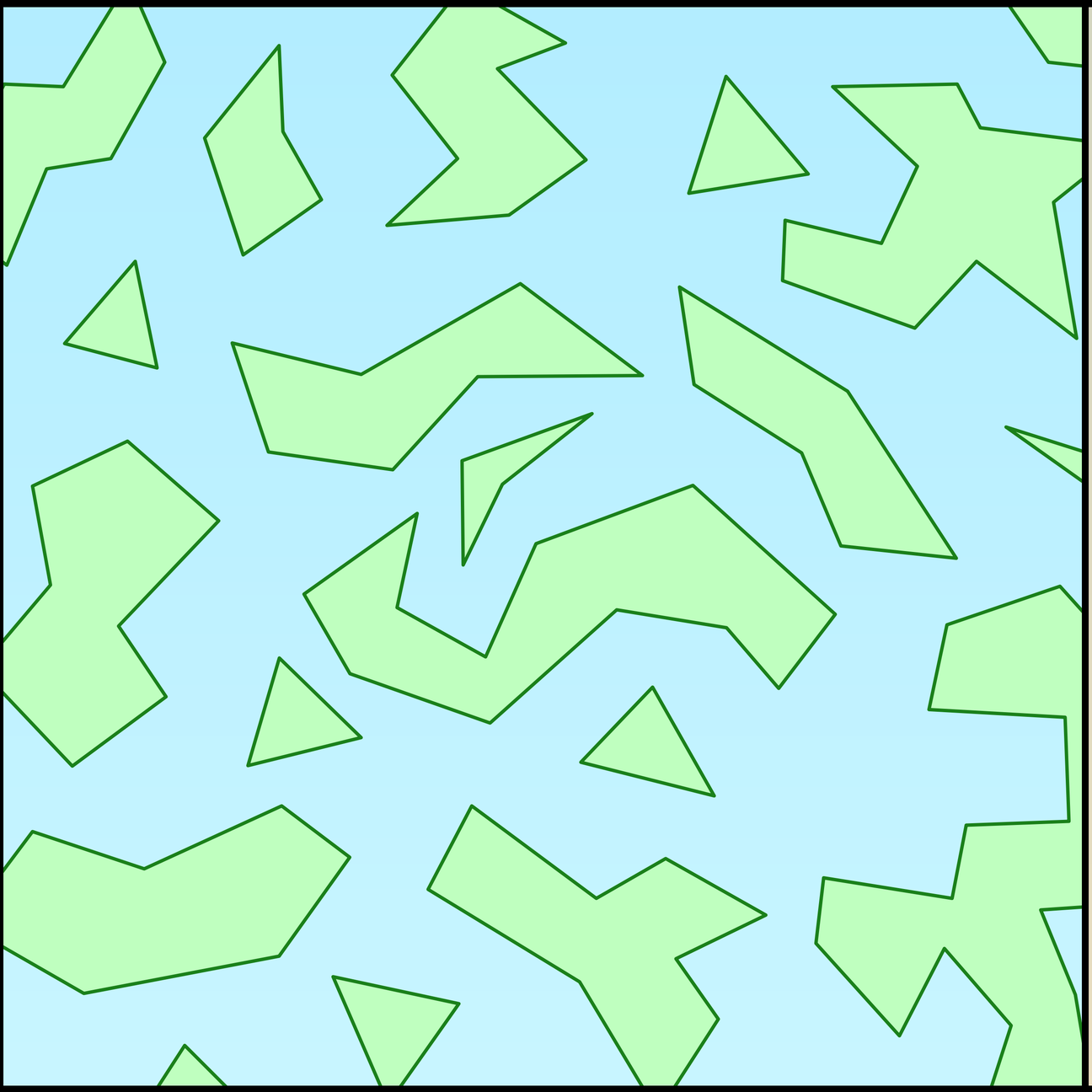


For example, let $\mathcal{R}$ be a set of disjoint polygons.



For example, let $\mathcal{R}$ be a set of disjoint polygons.

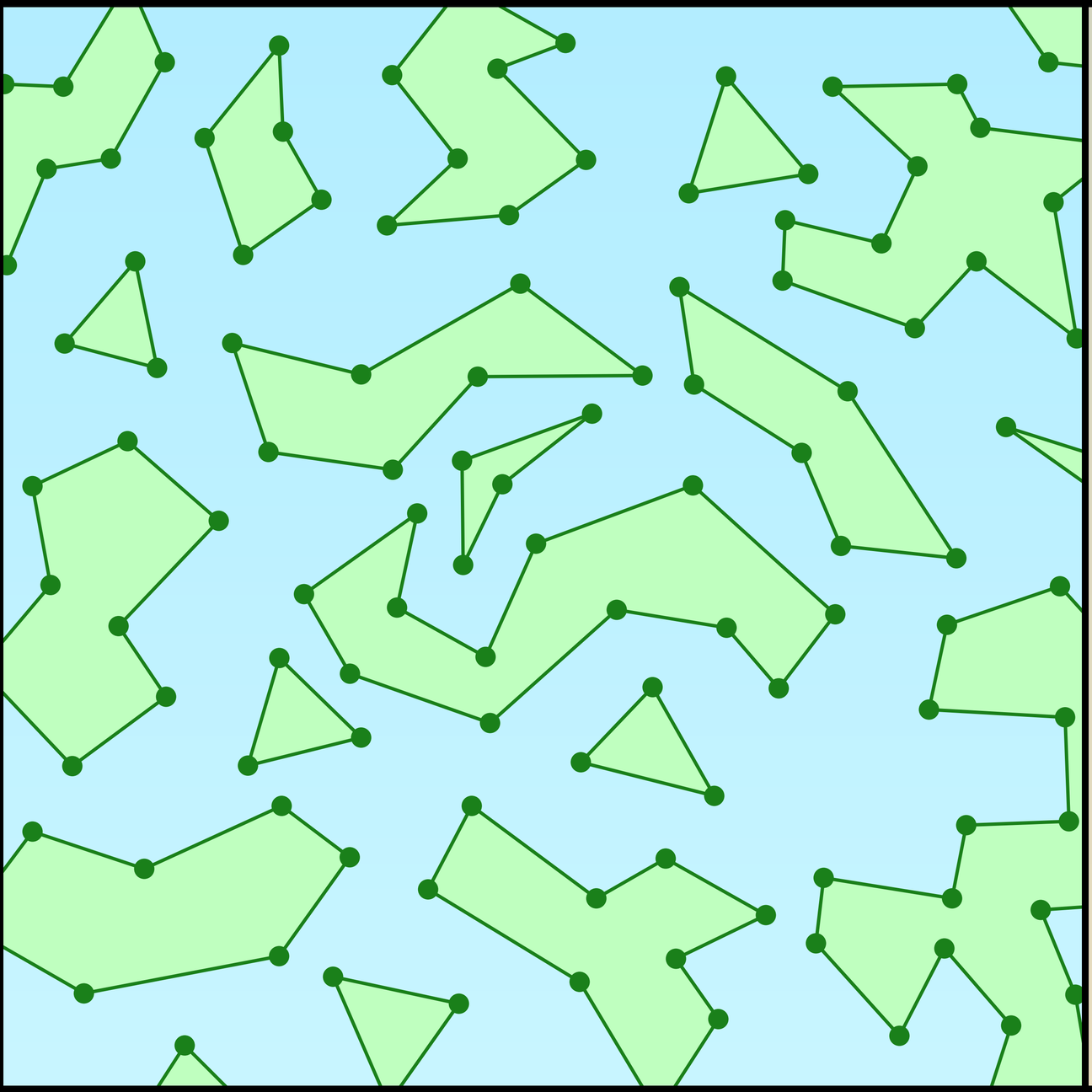
Suppose we want a *triangulation* of $P$.



For example, let $\mathcal{R}$ be a set of disjoint polygons.

Suppose we want a *triangulation* of $P$.

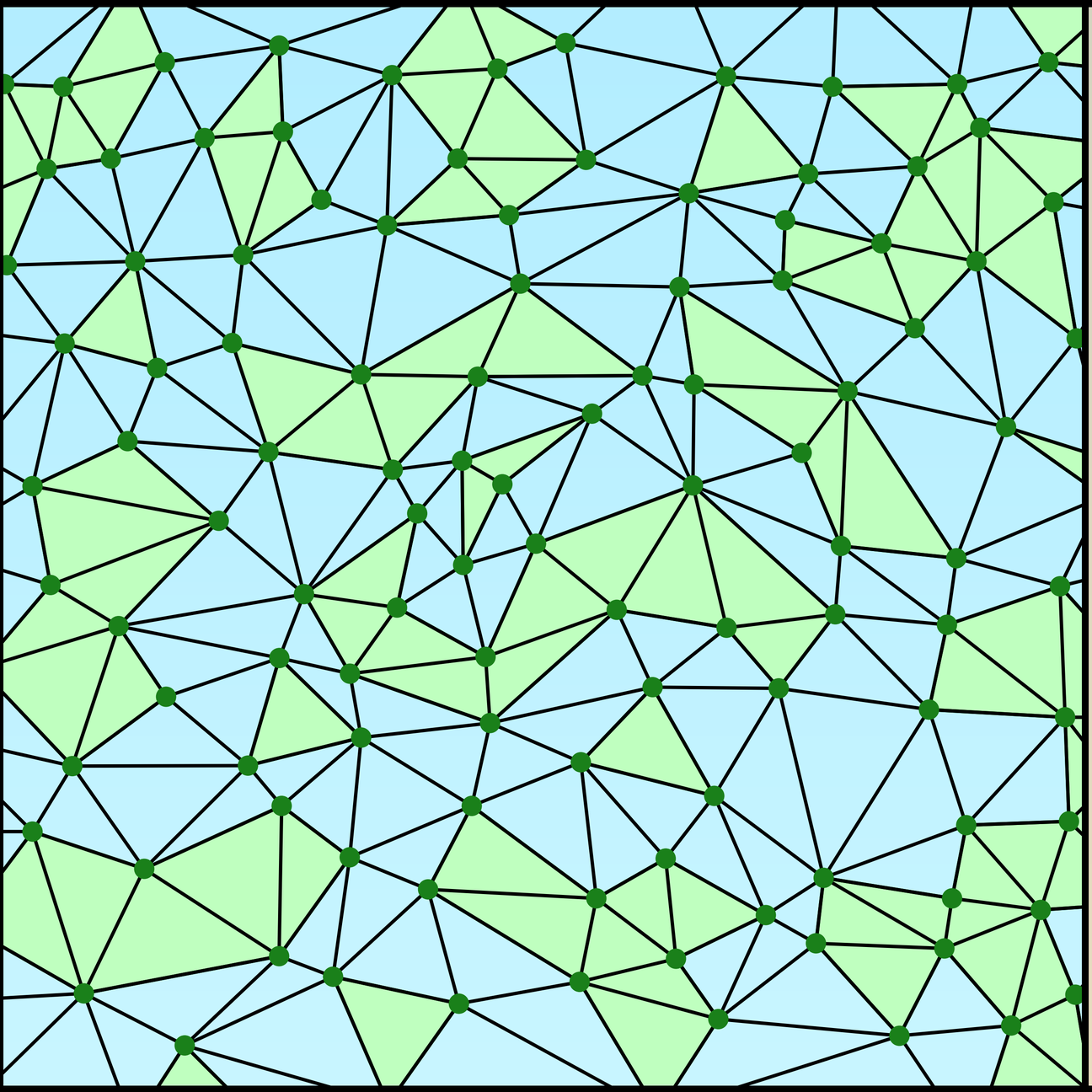
Fantasy $F(\mathcal{R})$ is a *constrained triangulation* of the vertices of $\mathcal{R}$.



For example, let $\mathcal{R}$ be a set of disjoint polygons.

Suppose we want a *triangulation* of $P$.

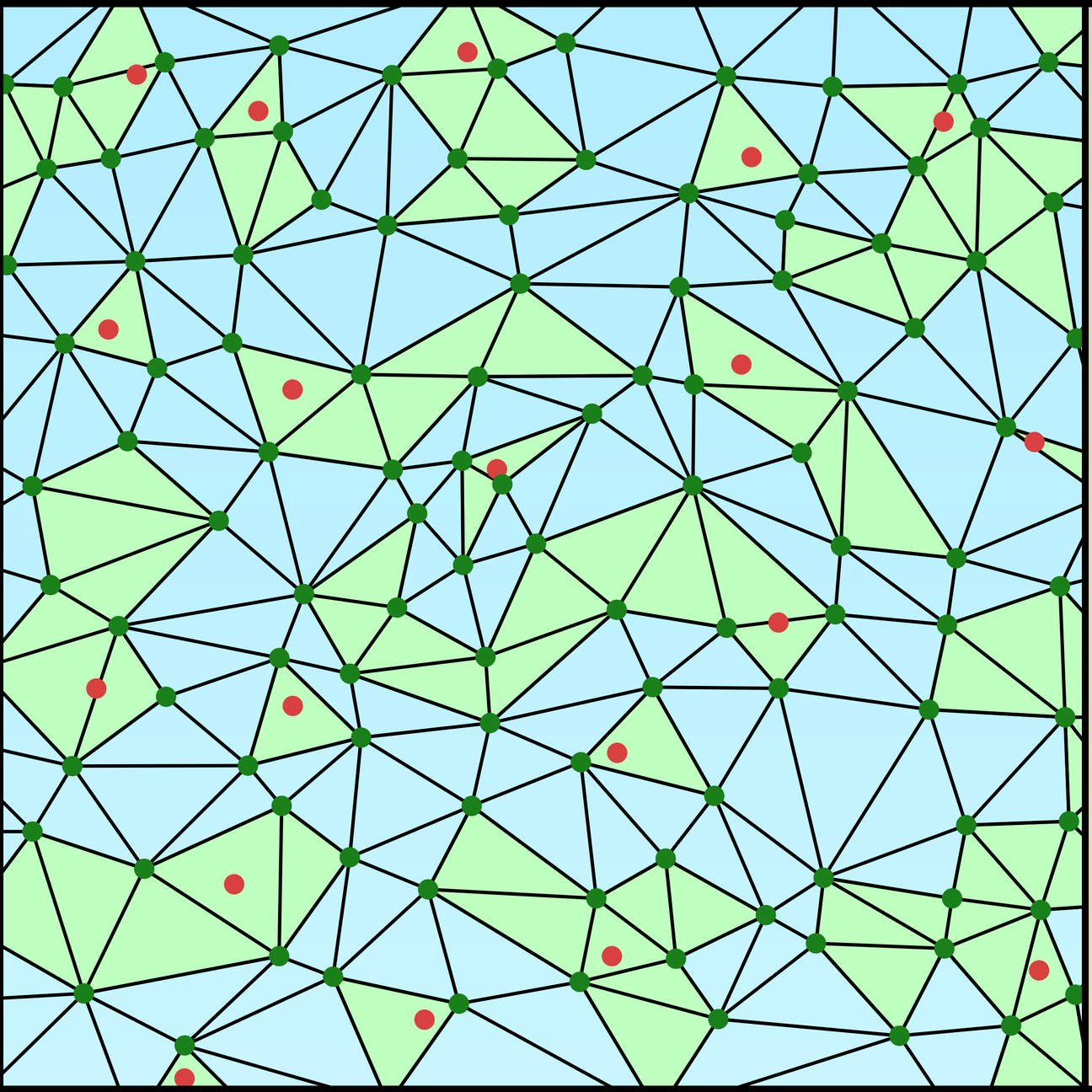
Fantasy $F(\mathcal{R})$ is a *constrained triangulation* of the vertices of $\mathcal{R}$.



For example, let $\mathcal{R}$ be a set of disjoint polygons.

Suppose we want a *triangulation* of $P$.

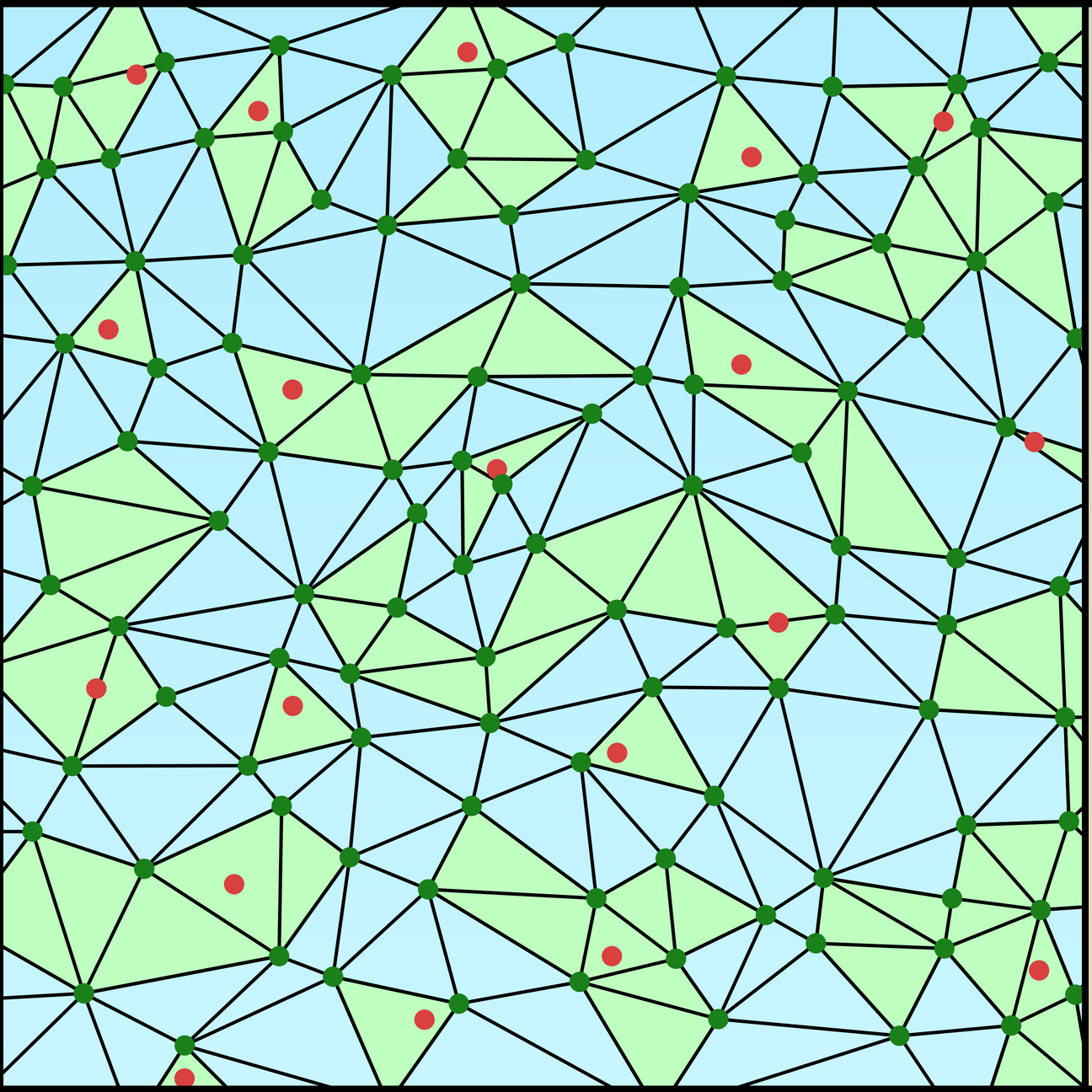
Fantasy $F(\mathcal{R})$ is a *constrained triangulation* of the vertices of $\mathcal{R}$.



For example, let $\mathcal{R}$ be a set of disjoint polygons.

Suppose we want a *triangulation* of $P$.

Fantasy $F(\mathcal{R})$ is a *constrained triangulation* of the vertices of $\mathcal{R}$.

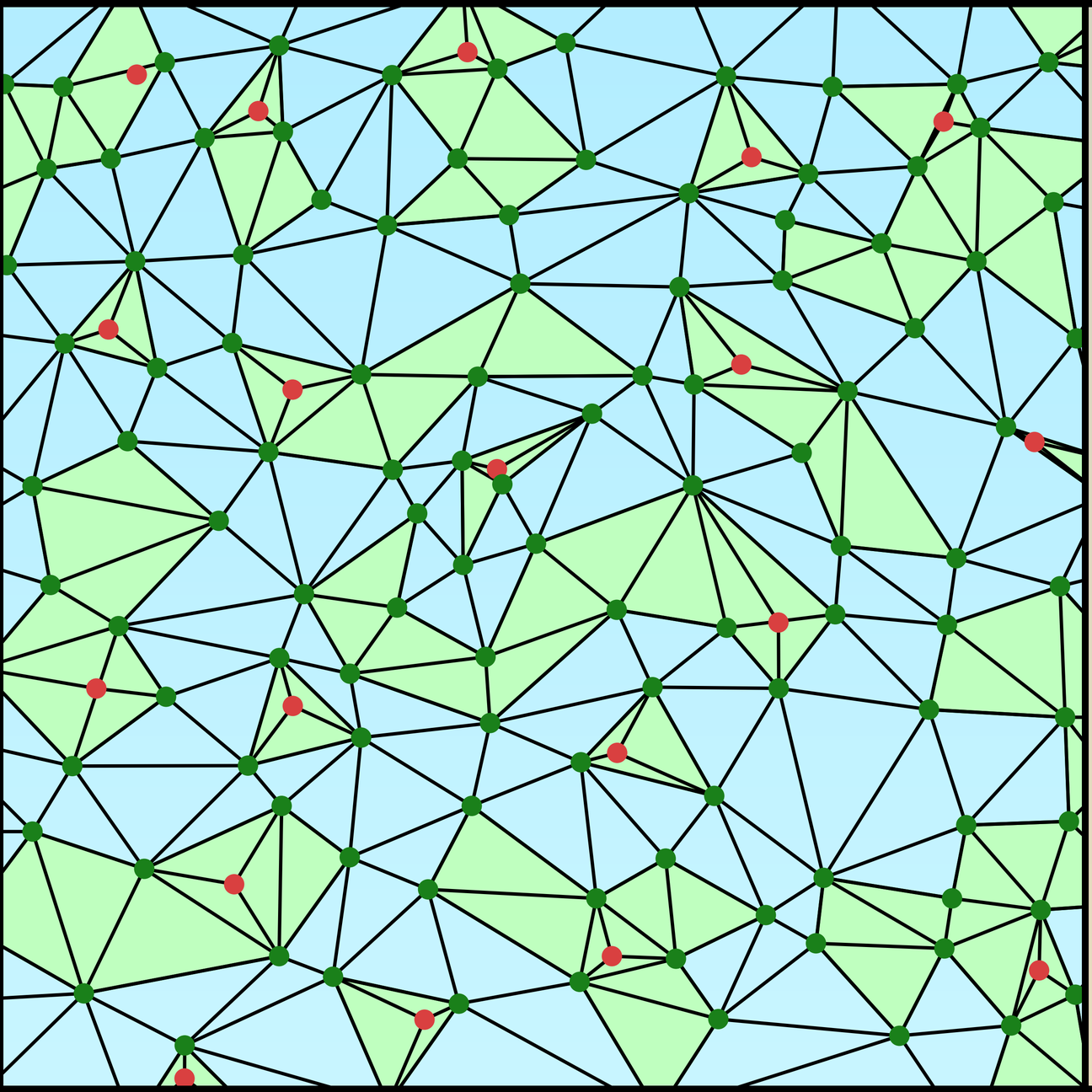


For example, let $\mathcal{R}$ be a set of disjoint polygons.

Suppose we want a *triangulation* of $P$.

Fantasy $F(\mathcal{R})$ is a *constrained triangulation* of the vertices of $\mathcal{R}$.

During reconstruction, we insert $P$ into the triangulation, and then remove the old vertices.

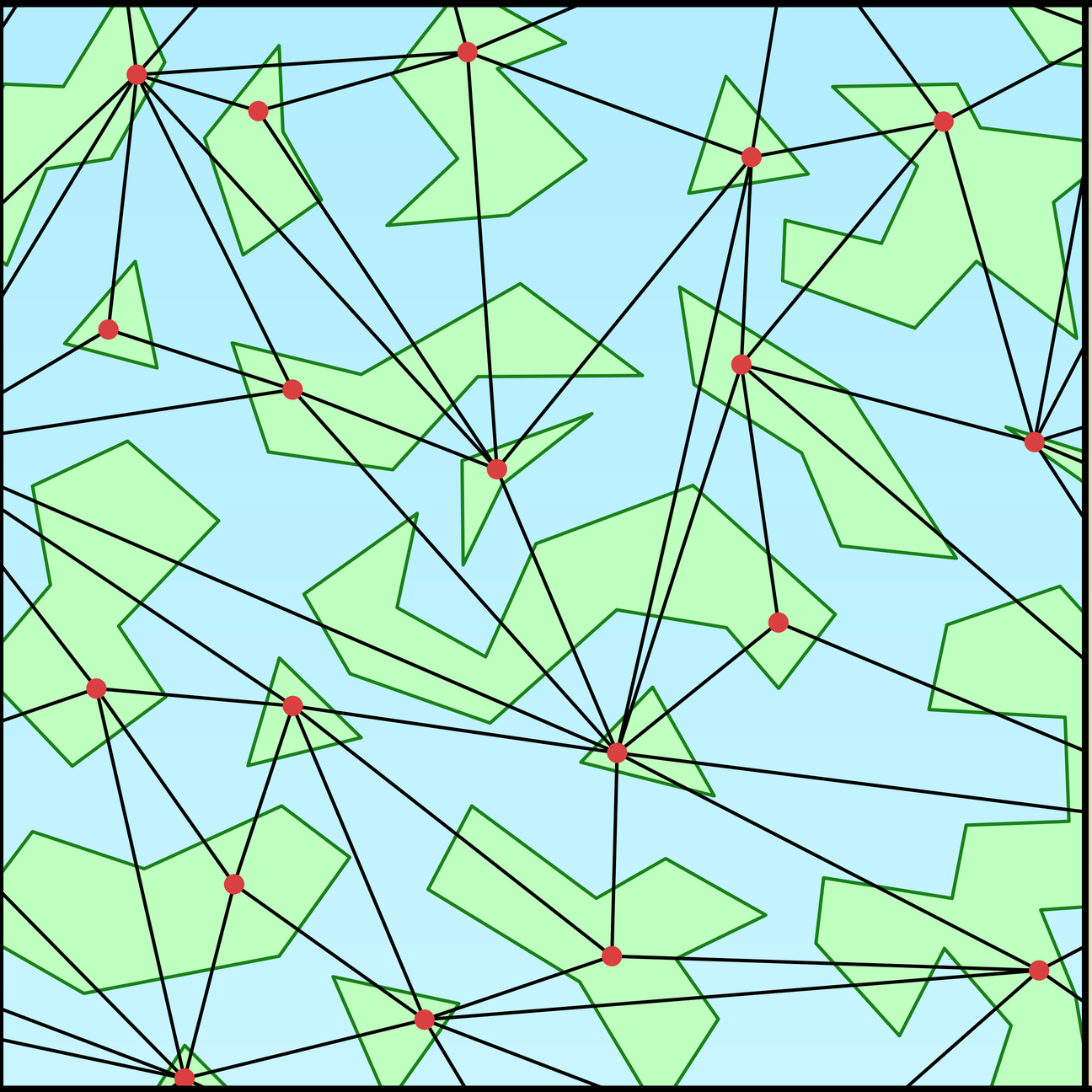


For example, let $\mathcal{R}$ be a set of disjoint polygons.

Suppose we want a *triangulation* of $P$.

Fantasy $F(\mathcal{R})$ is a *constrained triangulation* of the vertices of $\mathcal{R}$.

During reconstruction, we insert $P$ into the triangulation, and then remove the old vertices.

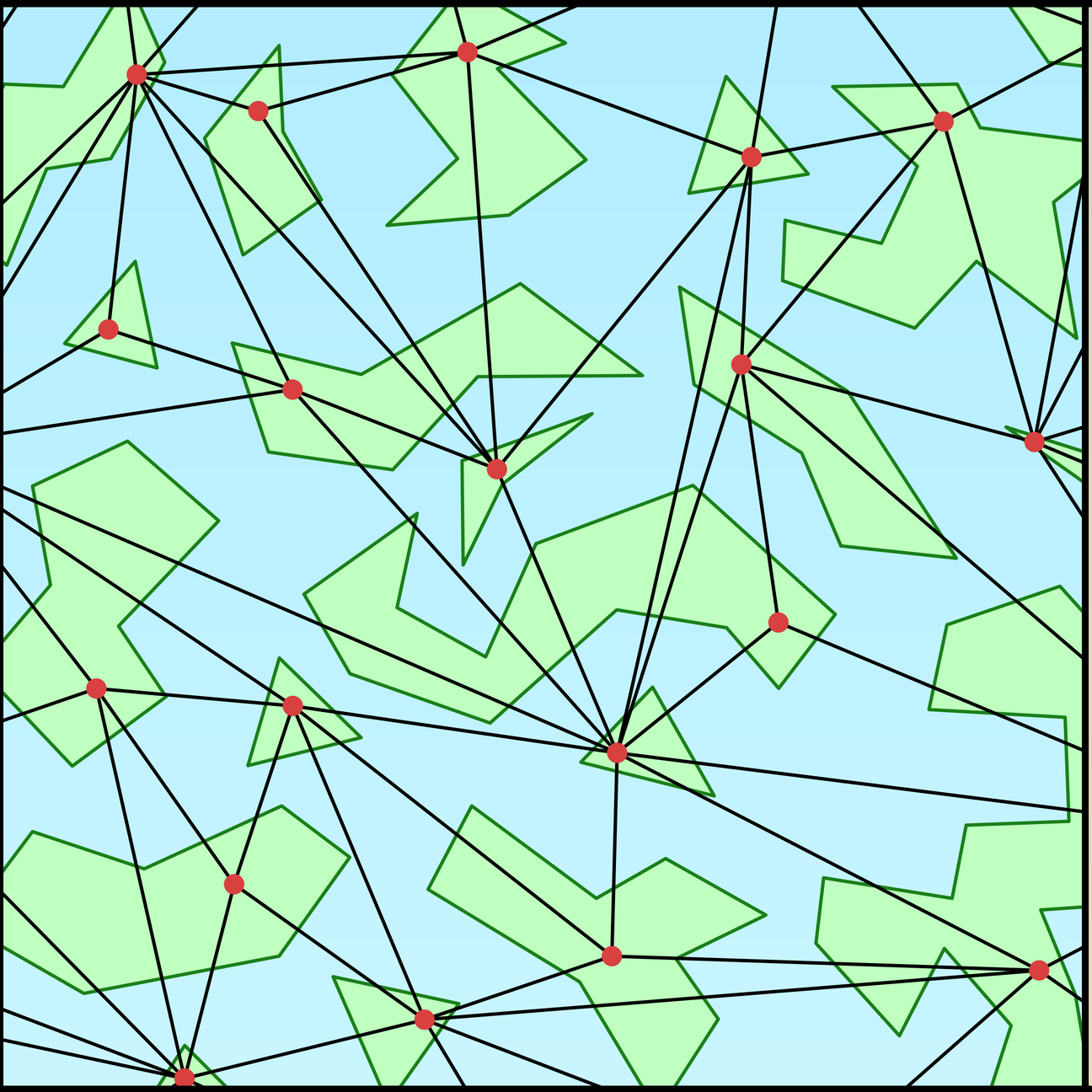


For example, let $\mathcal{R}$ be a set of disjoint polygons.

Suppose we want a *triangulation* of $P$.

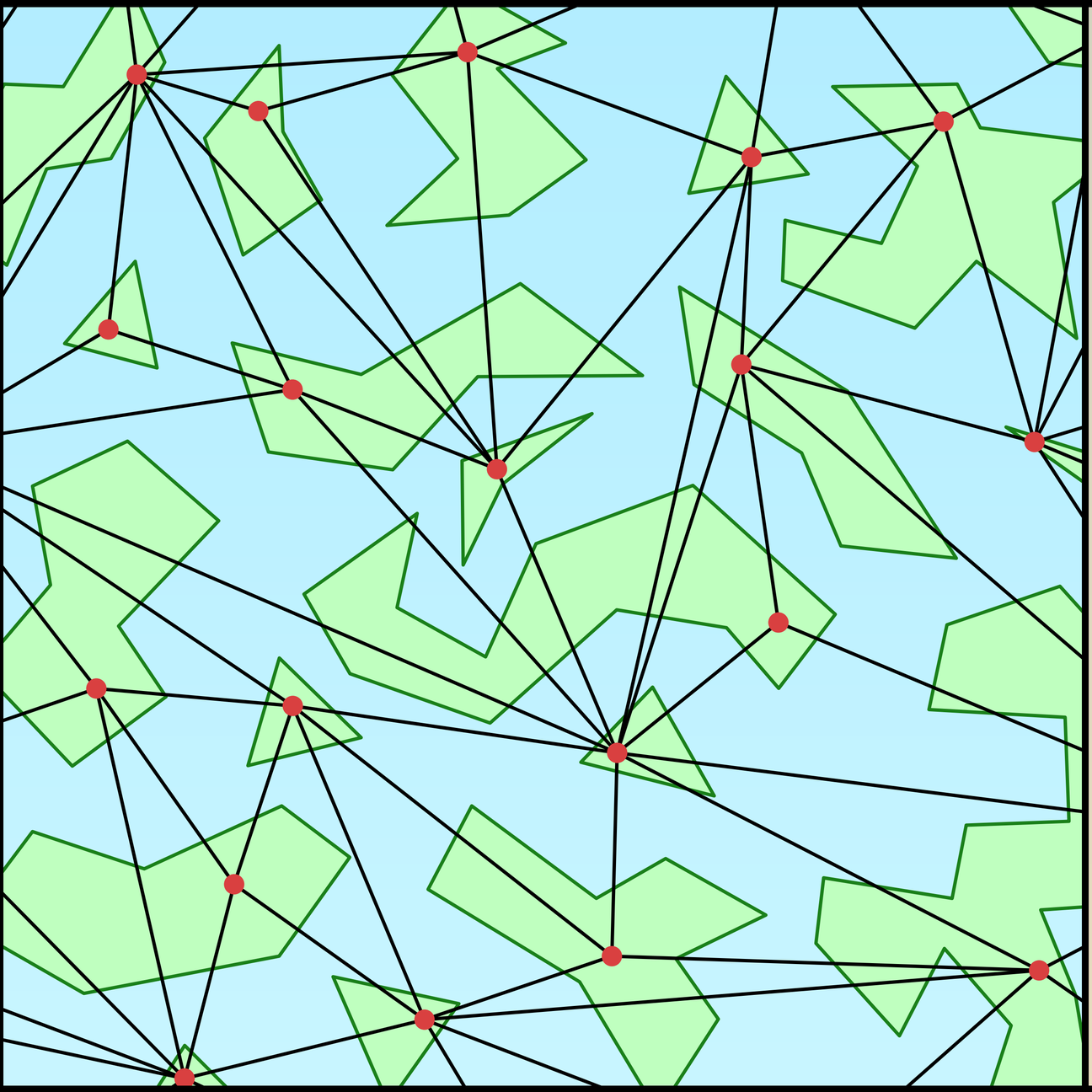
Fantasy $F(\mathcal{R})$ is a *constrained triangulation* of the vertices of $\mathcal{R}$.

During reconstruction, we insert $P$ into the triangulation, and then remove the old vertices.



The answer is *yes*:
a set of *disjoint polygons* $\mathcal{R}$ contains enough information to reconstruct a *triangulation* of $P$.

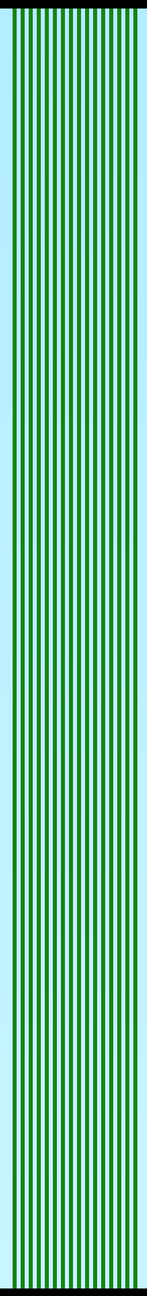
[Löffler, Mitchell & van Kreveld, 2010]



The answer is *yes*:
a set of *disjoint polygons* $\mathcal{R}$ contains enough information to reconstruct a *triangulation* of $P$.

In contrast:
a set of disjoint polygons $\mathcal{R}$ does *not* contain enough information to reconstruct the *Delaunay triangulation* of $P$.

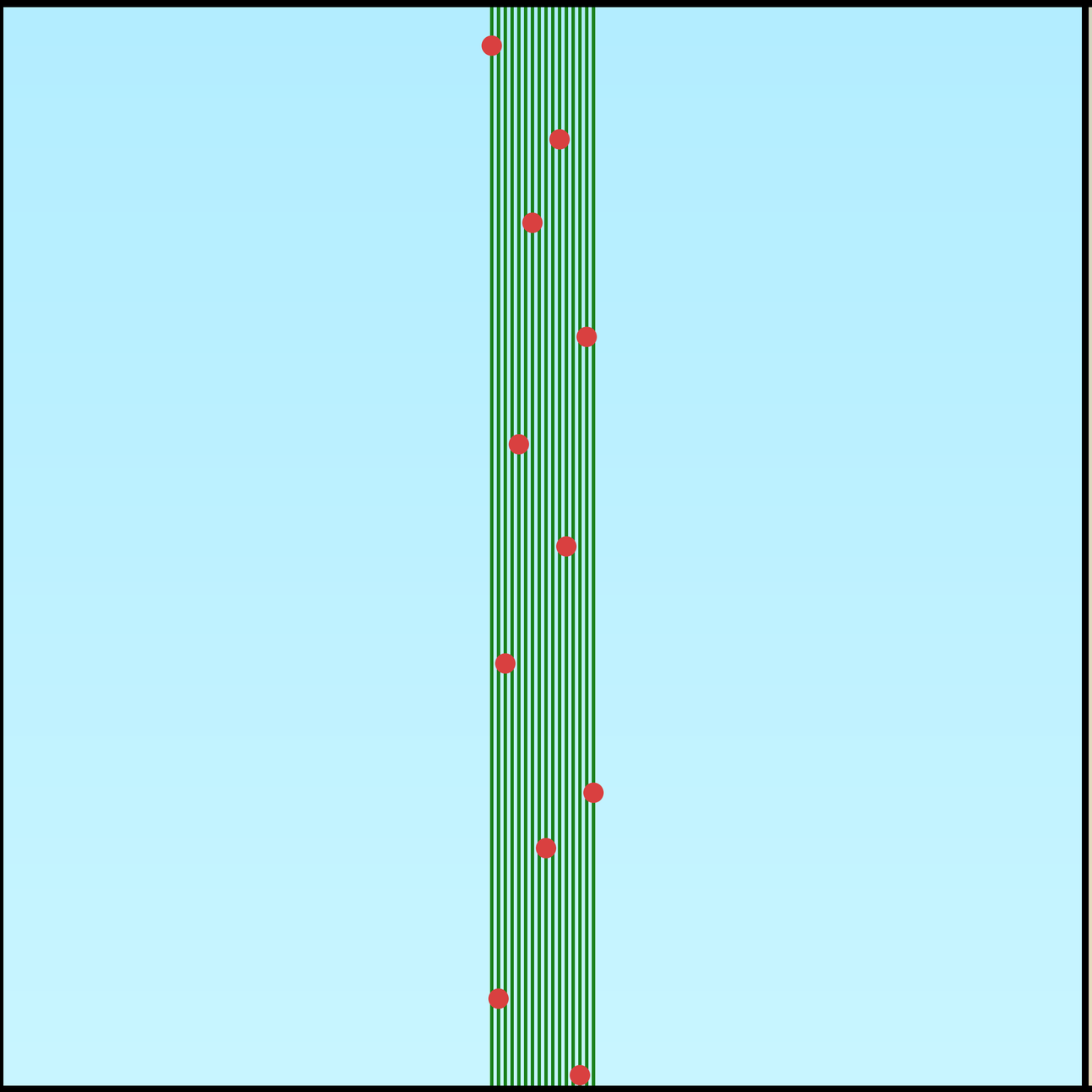
[Djidjev & Lingas, 1995]
[Löffler, Mitchell & van Kreveld, 2010]



The answer is *yes*:
a set of *disjoint polygons* $\mathcal{R}$ contains enough information to reconstruct a *triangulation* of $P$.

In contrast:
a set of disjoint polygons $\mathcal{R}$ does *not* contain enough information to reconstruct the *Delaunay triangulation* of $P$.

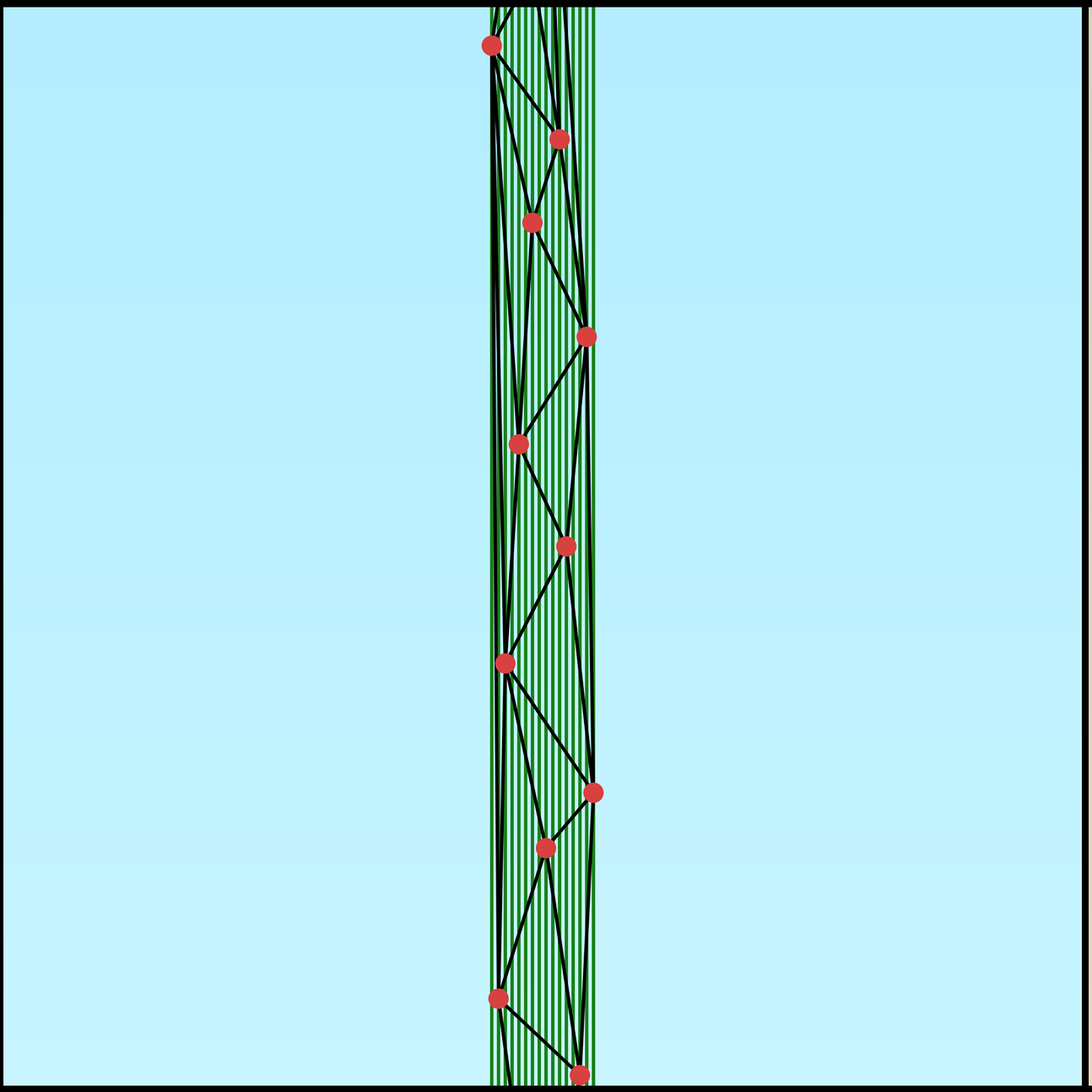
[Djidjev & Lingas, 1995]
[Löffler, Mitchell & van Kreveld, 2010]



The answer is *yes*:
a set of *disjoint polygons* $\mathcal{R}$ contains enough information to reconstruct a *triangulation* of $P$.

In contrast:
a set of disjoint polygons $\mathcal{R}$ does *not* contain enough information to reconstruct the *Delaunay triangulation* of $P$.

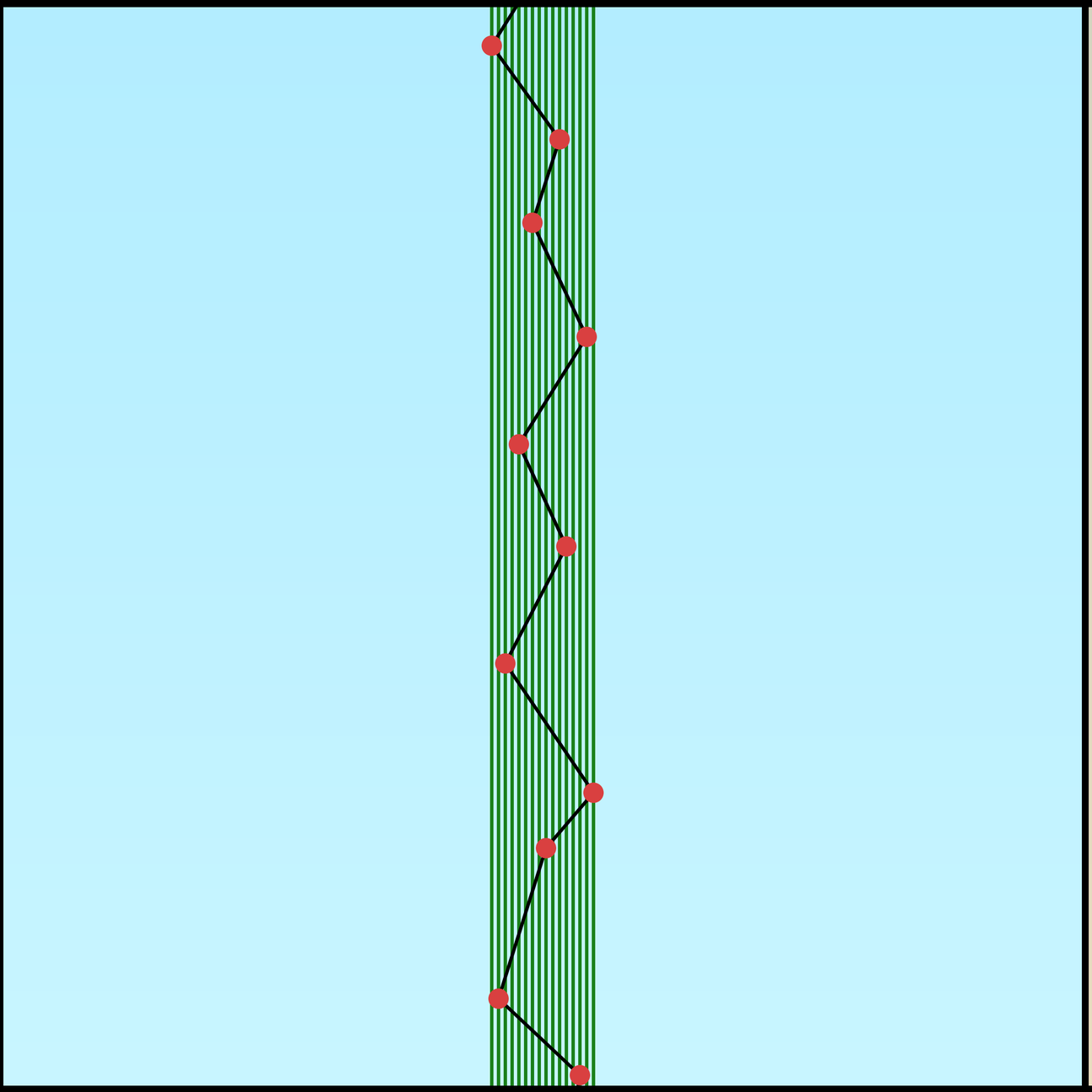
[Djidjev & Lingas, 1995]
[Löffler, Mitchell & van Kreveld, 2010]



The answer is *yes*:
a set of *disjoint polygons* $\mathcal{R}$ contains enough information to reconstruct a *triangulation* of $P$.

In contrast:
a set of disjoint polygons $\mathcal{R}$ does *not* contain enough information to reconstruct the *Delaunay triangulation* of $P$.

[Djidjev & Lingas, 1995]
[Löffler, Mitchell & van Kreveld, 2010]



The answer is *yes*:
a set of *disjoint polygons* $\mathcal{R}$ contains enough information to reconstruct a *triangulation* of $P$.

In contrast:
a set of disjoint polygons $\mathcal{R}$ does *not* contain enough information to reconstruct the *Delaunay triangulation* of $P$.

[Djidjev & Lingas, 1995]
[Löffler, Mitchell & van Kreveld, 2010]

# Chapter 2
*PARTIAL INFORMATION*

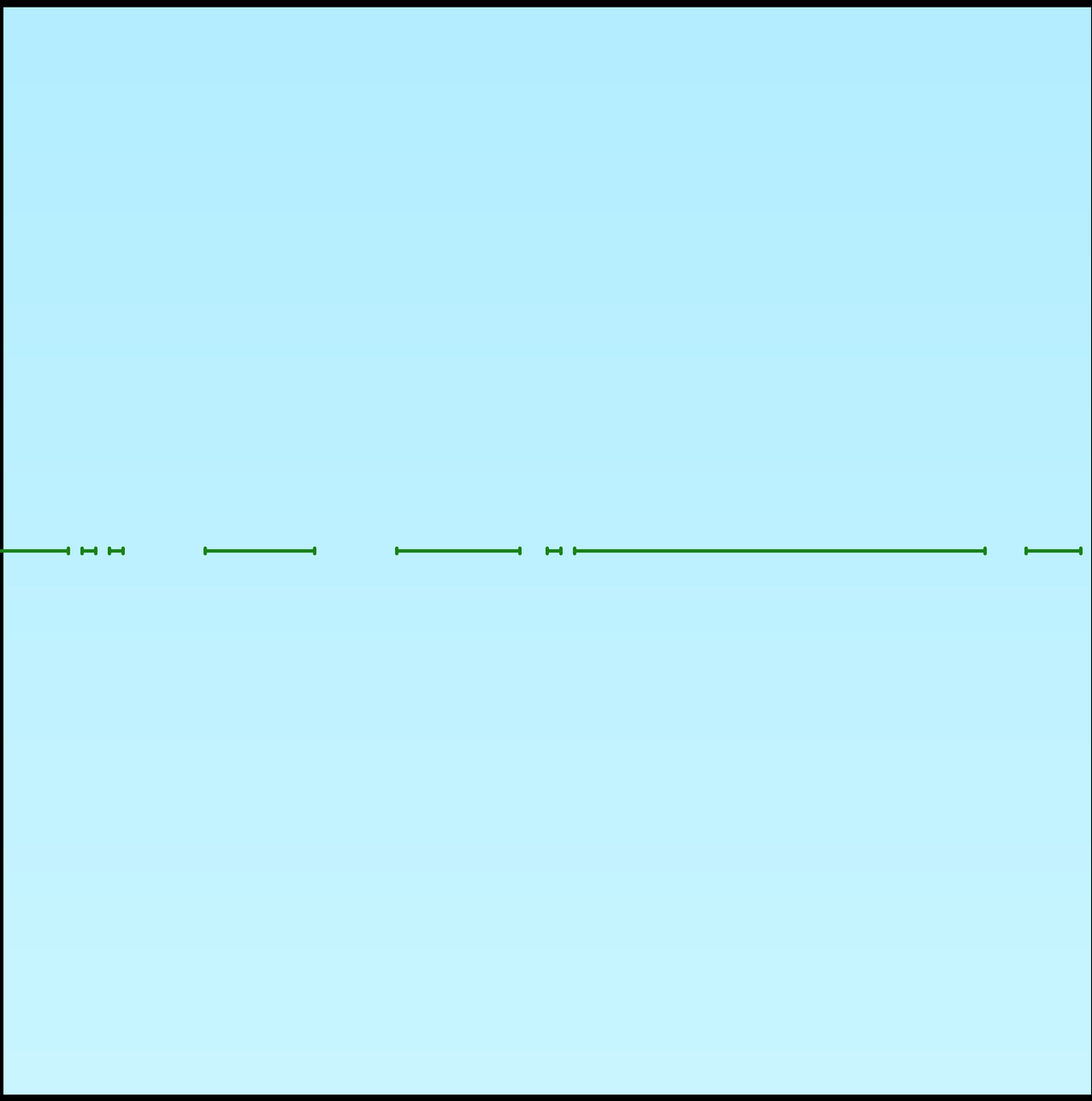
Uncertain locations

Quantifying information

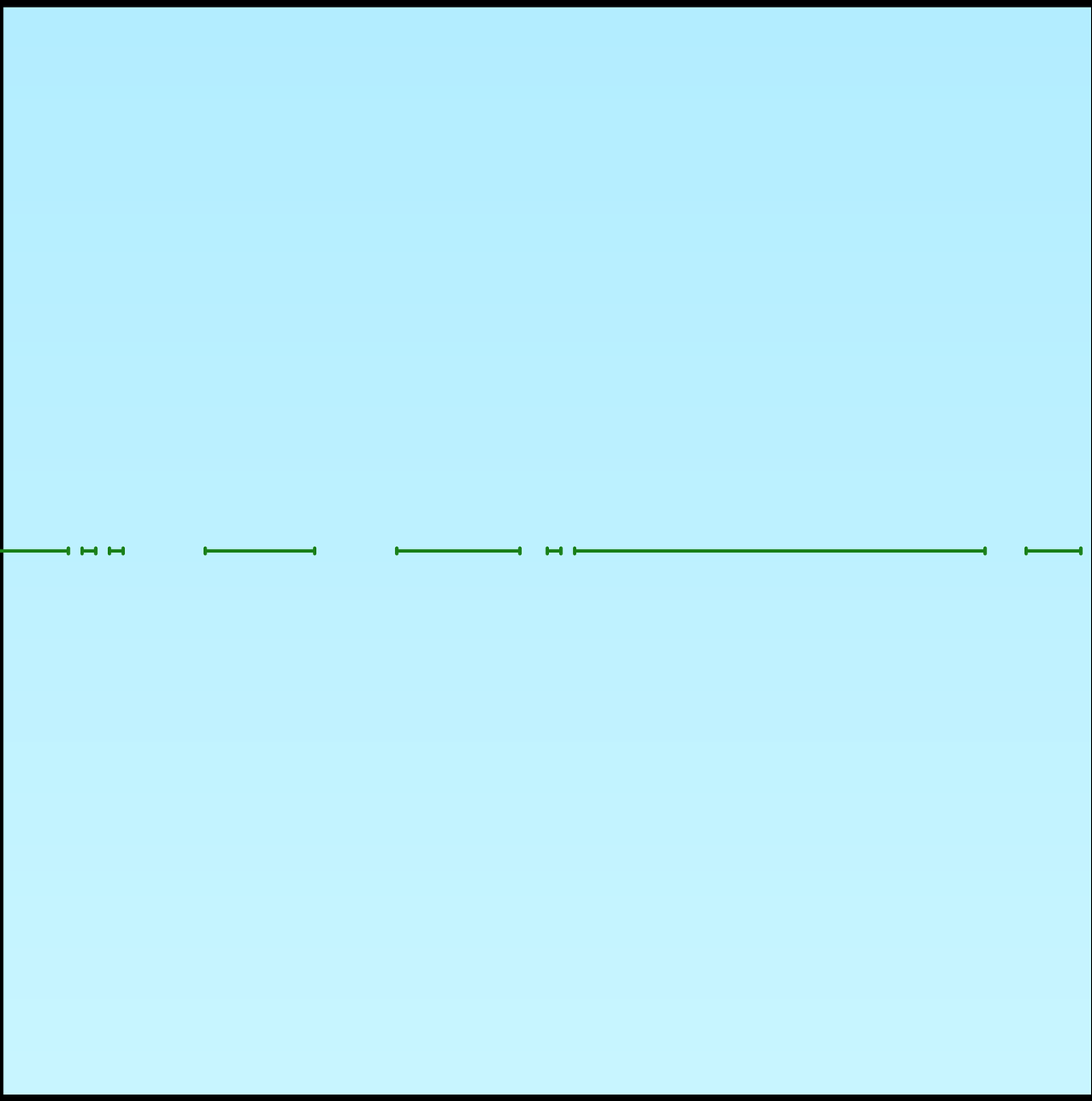
State of the art in 1D

State of the art in 2D

In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?

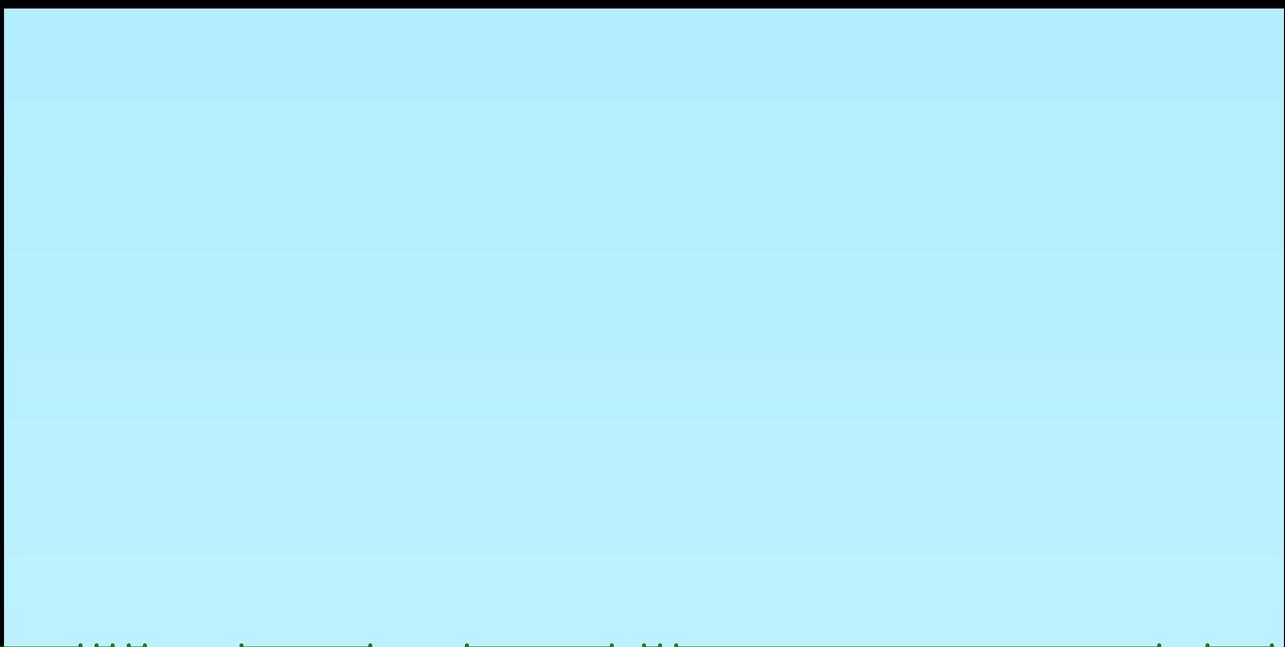


In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?



In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?

Of course we can!



In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?

Of course we can!

During reconstruction, we simply scan the points.

In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?

Of course we can!

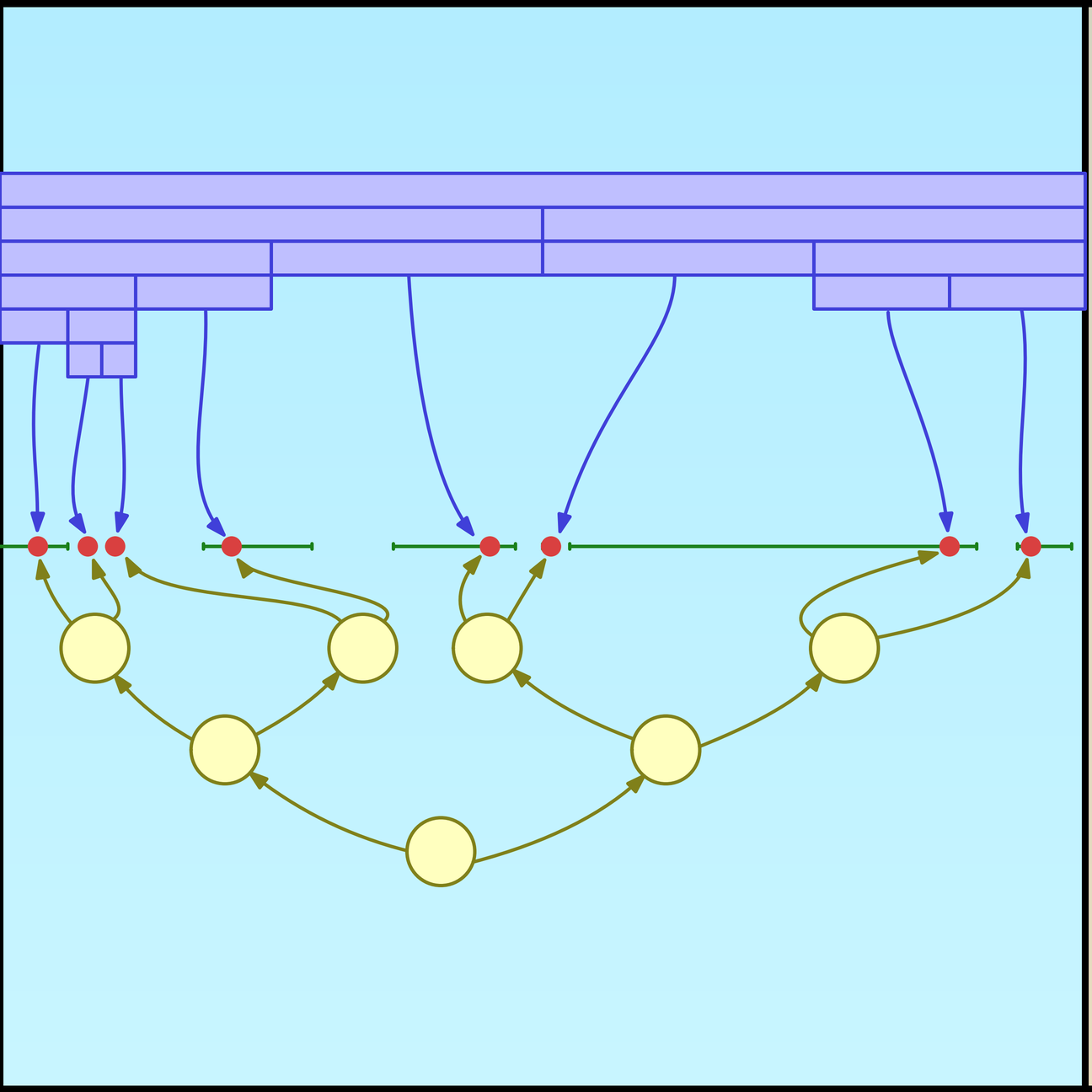
During reconstruction, we simply scan the points.

In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?

Of course we can!

During reconstruction, we simply scan the points.

Since any other structure can be derived from the order in $O(n)$ time...

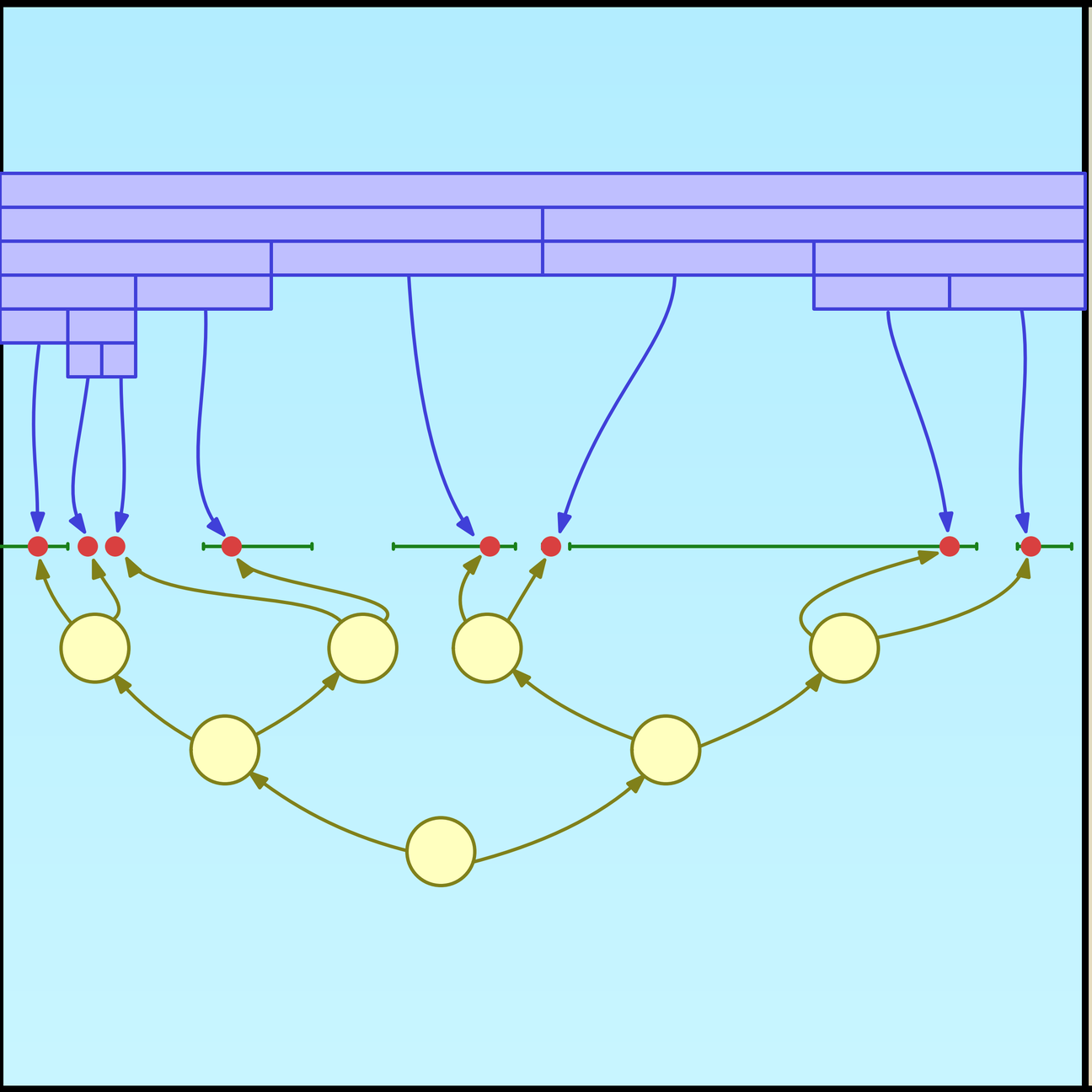


In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?

Of course we can!

During reconstruction, we simply scan the points.

Since any other structure can be derived from the order in $O(n)$ time...



In $\mathbb{R}^1$, this is question: can we presort a set of disjoint intervals?

Of course we can!

During reconstruction, we simply scan the points.

Since any other structure can be derived from the order in $O(n)$ time...

...everything is easy.

# Chapter 2

*PARTIAL INFORMATION*

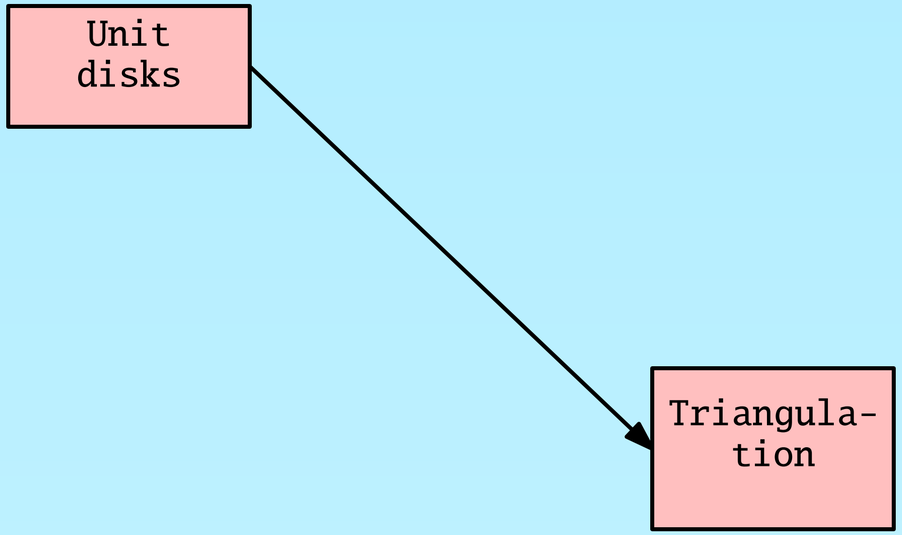
Uncertain locations

Quantifying information

State of the art in 1D

State of the art in 2D

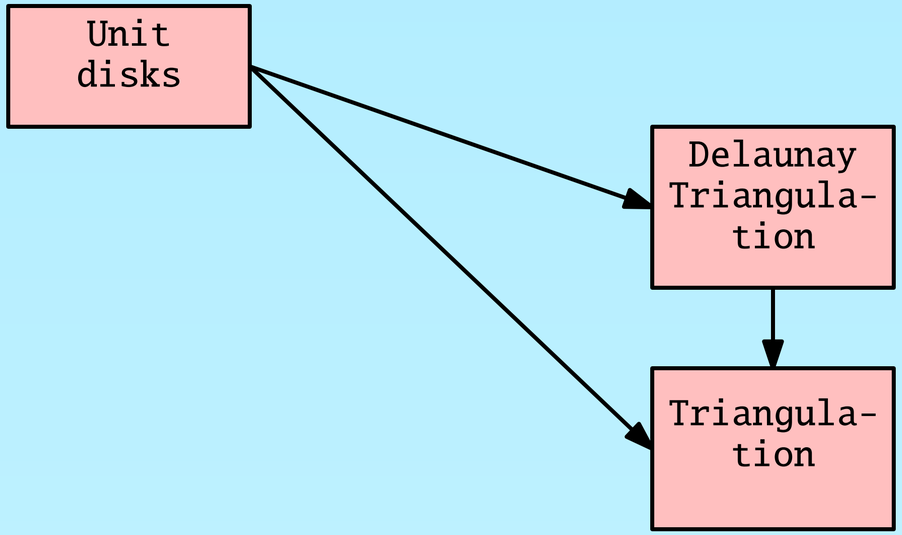
In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.




In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

⟶ Deterministic linear time

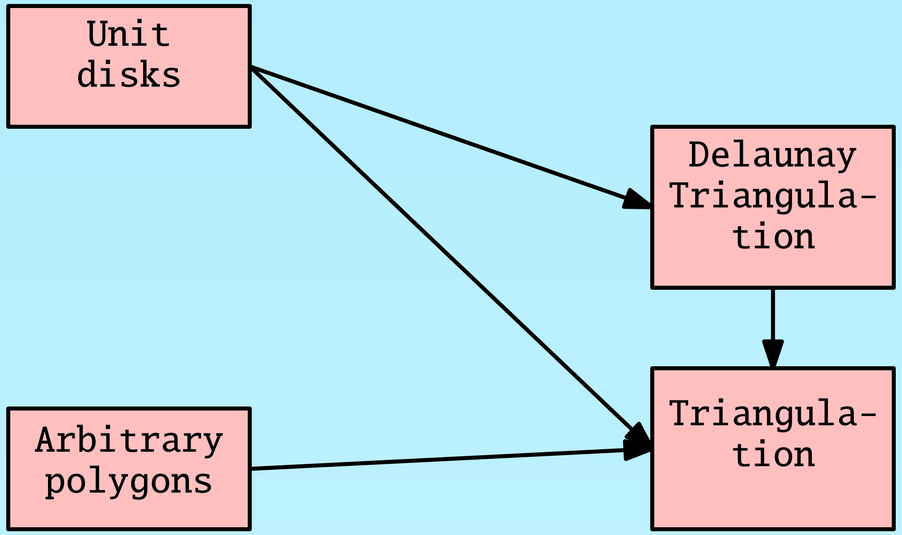
[Held & Mitchell, 2008]




In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

$\longrightarrow$ Deterministic linear time

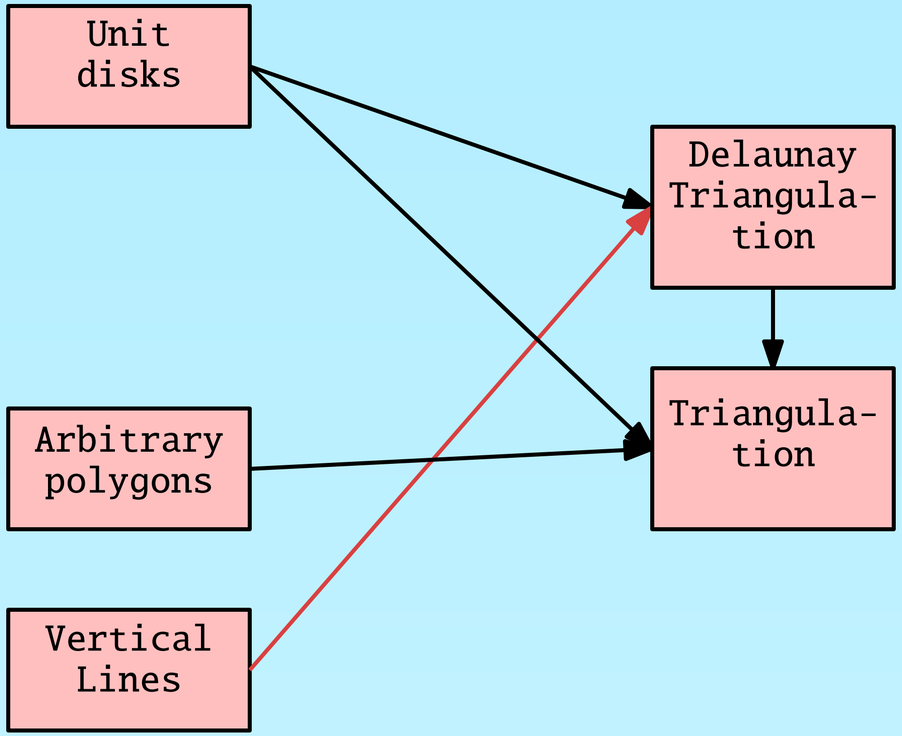
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]




In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

$\longrightarrow$ Deterministic linear time

[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]

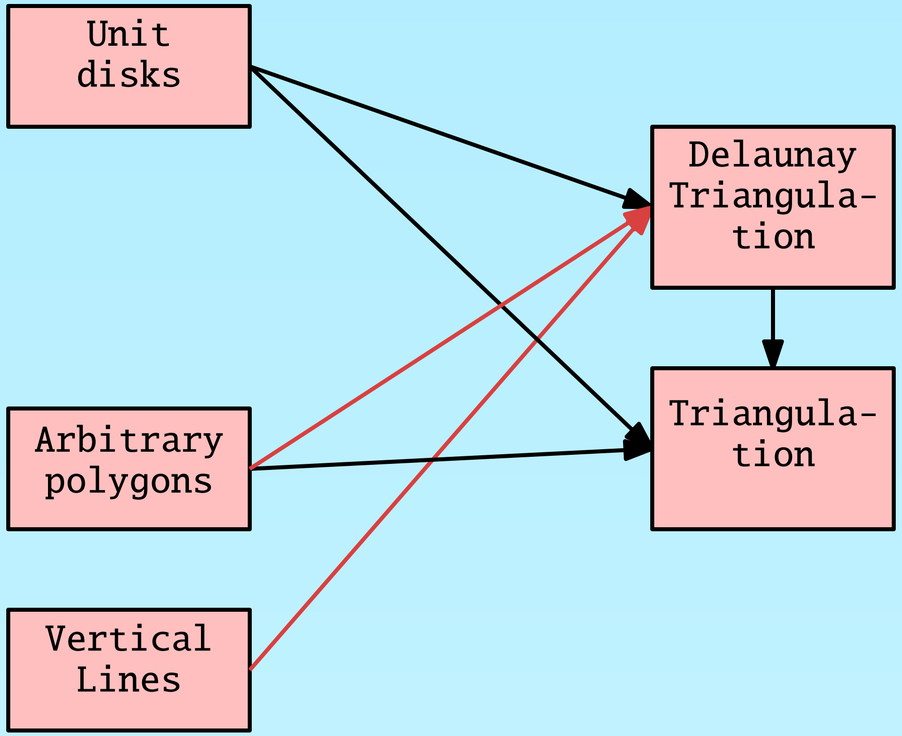



In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

→ Deterministic linear time

→ o(n log n) not possible

[Djidjev & Lingas, 1995]
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]

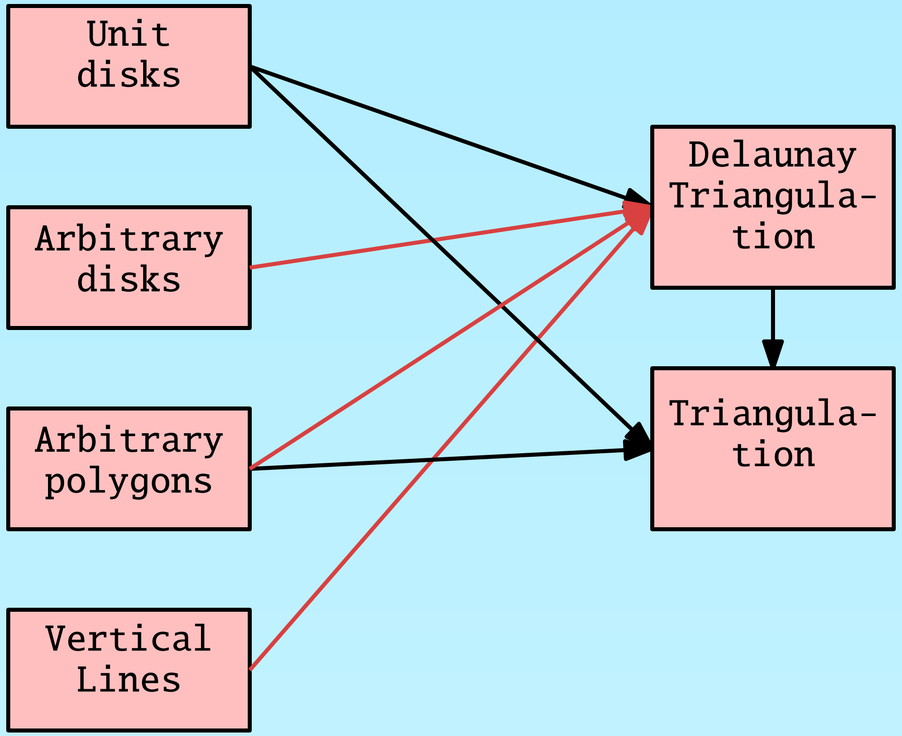



In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

→ Deterministic linear time

→ o(n log n) not possible

[Djidjev & Lingas, 1995]
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]

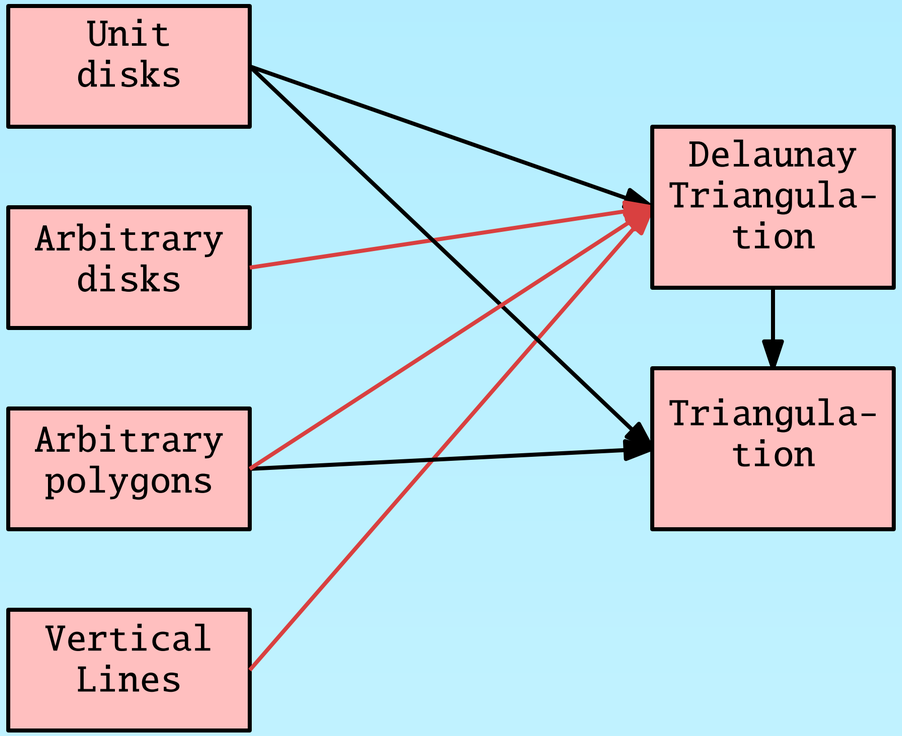



In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

→ Deterministic linear time

→ o(n log n) not possible

[Djidjev & Lingas, 1995]
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]
[Buchin, Löffler, Morin & Mulzer, 2011]

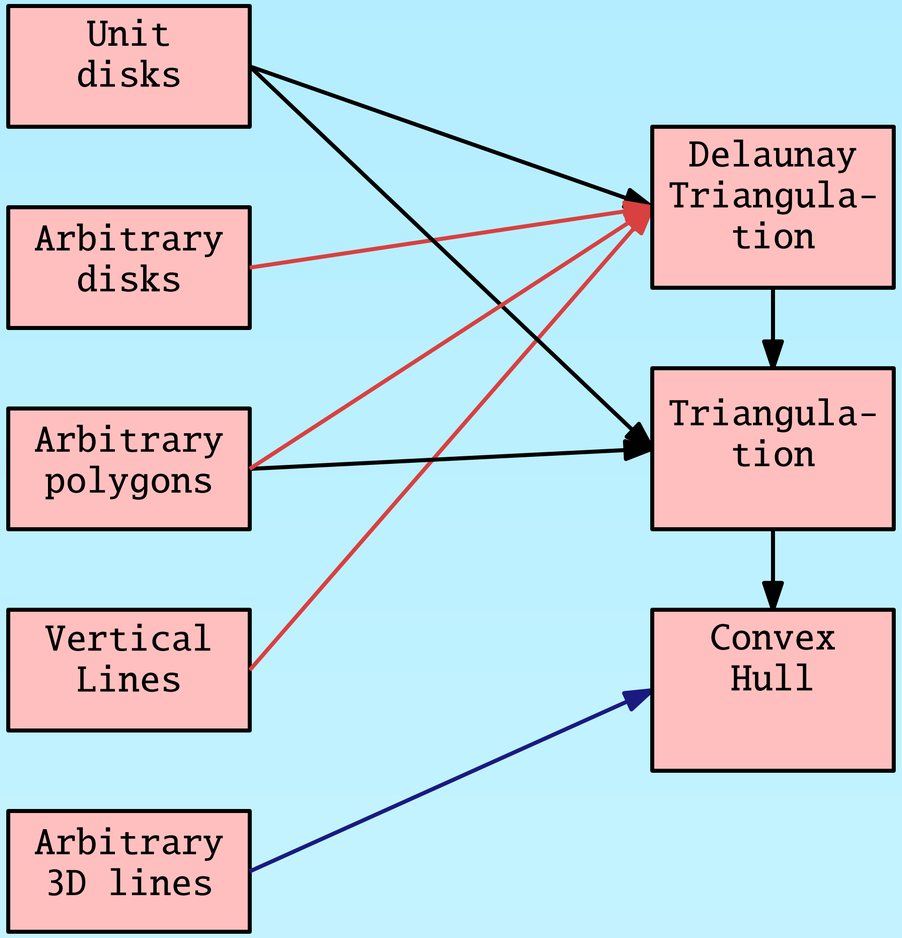



In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

→ Deterministic linear time

→ o(n log n) not possible

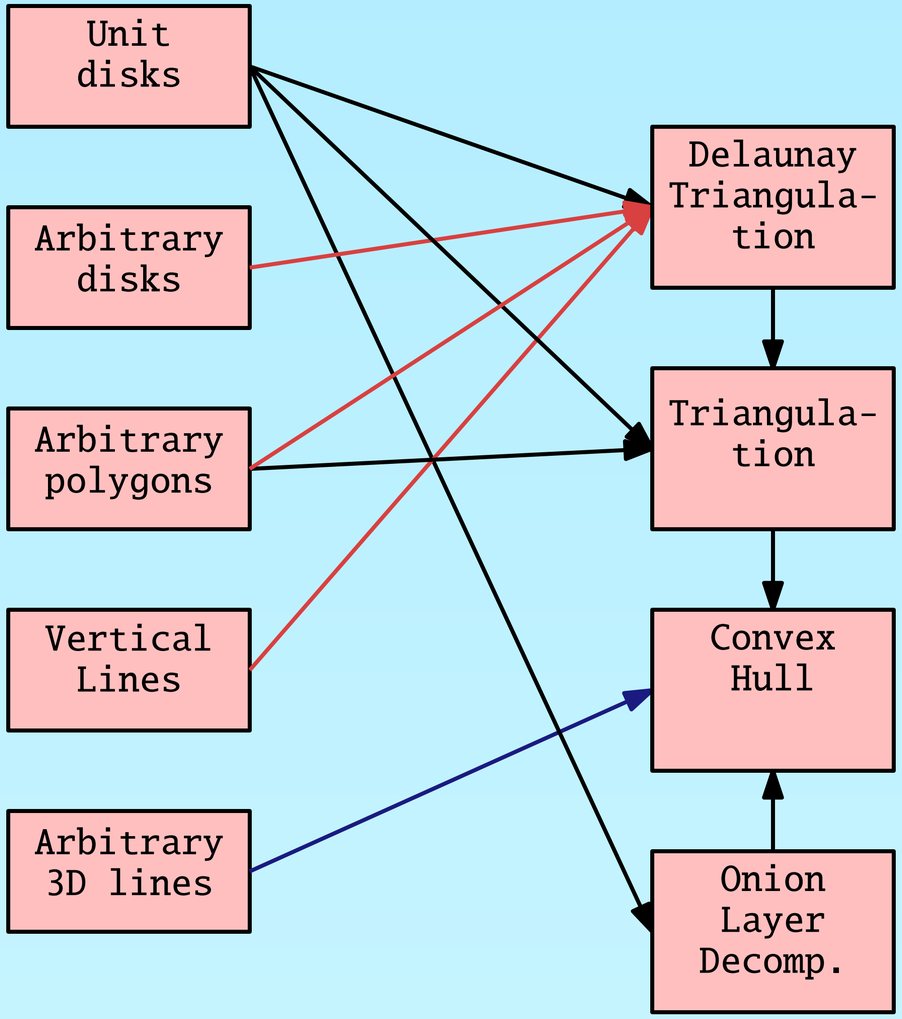
[Djidjev & Lingas, 1995]
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]
[Buchin, Löffler, Morin & Mulzer, 2011]
[Devillers, 2011]




In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

→ Deterministic linear time
→ o(n log n) but not linear
→ o(n log n) not possible

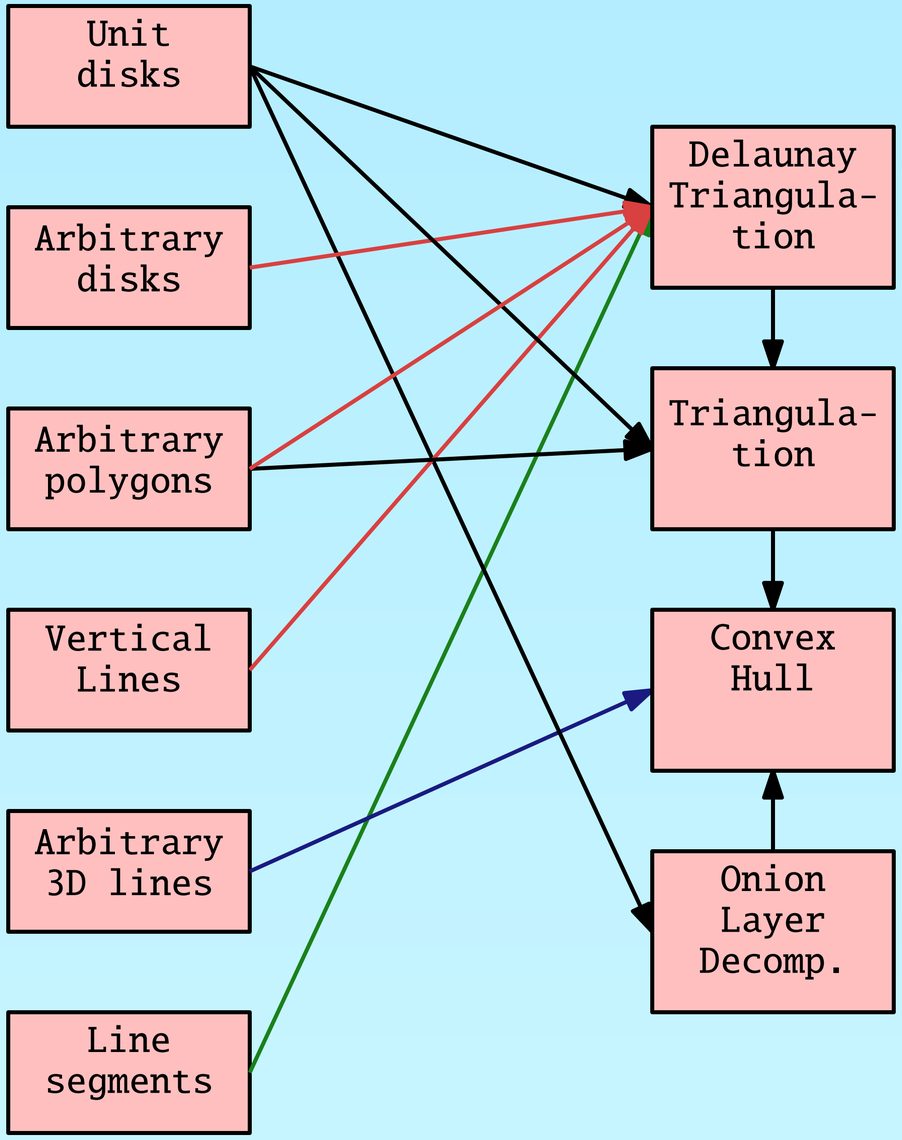
[Djidjev & Lingas, 1995]
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]
[Buchin, Löffler, Morin & Mulzer, 2011]
[Devillers, 2011]
[Ezra & Mulzer, 2013]




In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

→ Deterministic linear time
→ o(n log n) but not linear
→ o(n log n) not possible

[Djidjev & Lingas, 1995]
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]
[Buchin, Löffler, Morin & Mulzer, 2011]
[Devillers, 2011]
[Ezra & Mulzer, 2013]
[Löffler & Mulzer, 2014]




In $\mathbb{R}^2$, there is also a (somewhat more recent) long history of $O(n)$ time reconstruction algorithms for disjoint regions.

→ Deterministic linear time
→ o(n log n) but not linear
→ o(n log n) not possible
→ expected linear time (expectation over $P$)

[Djidjev & Lingas, 1995]
[Held & Mitchell, 2008]
[Löffler & Snoeyink, 2010]
[Löffler, Mitchell & van Kreveld, 2010]
[Buchin, Löffler, Morin & Mulzer, 2011]
[Devillers, 2011]
[Ezra & Mulzer, 2013]
[Löffler & Mulzer, 2014]
[Keikha, Mohades & Davoodi, 2016]

# Chapter 3
## *FINE-GRAINED MEASURES*

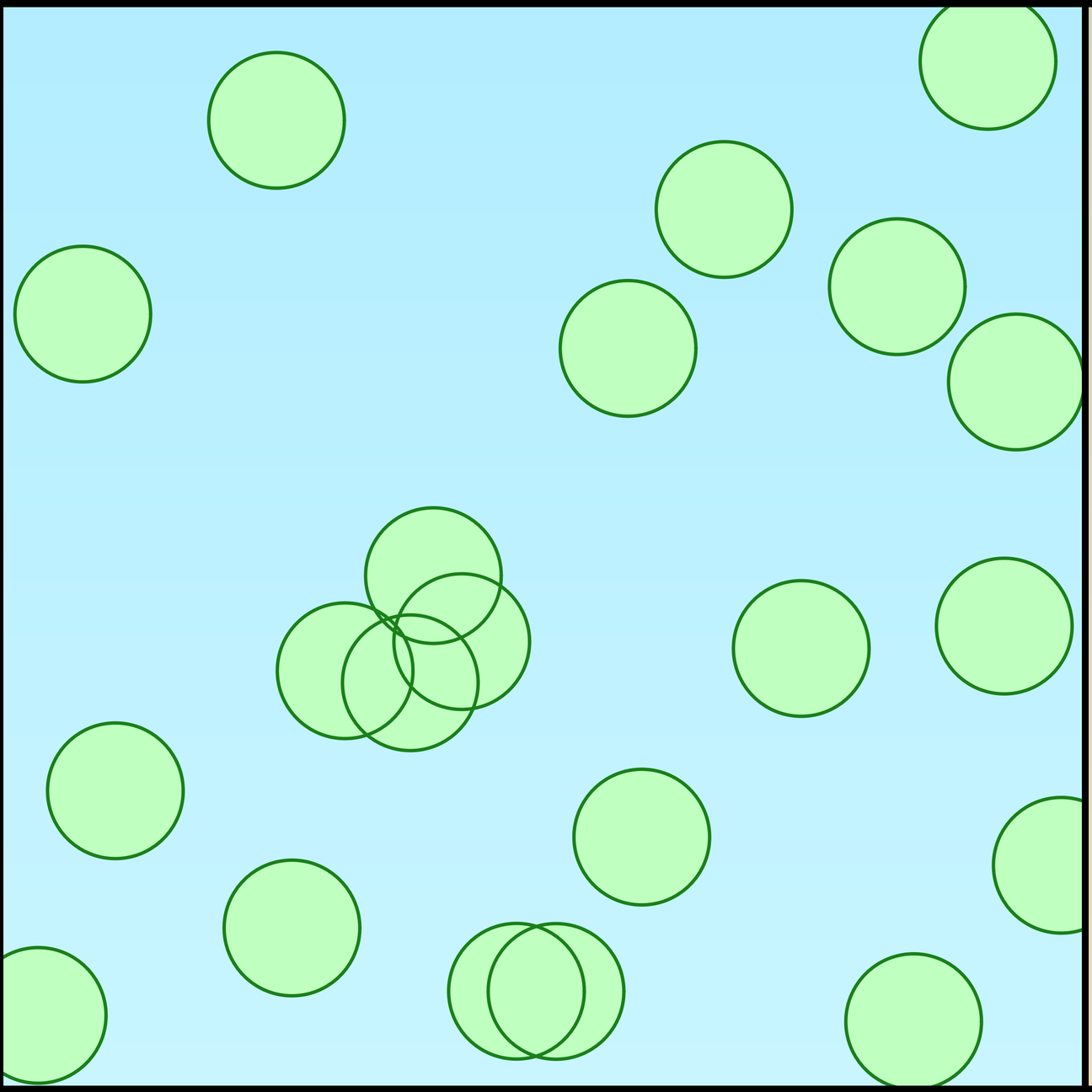
Ambiguity of regions

Information theory

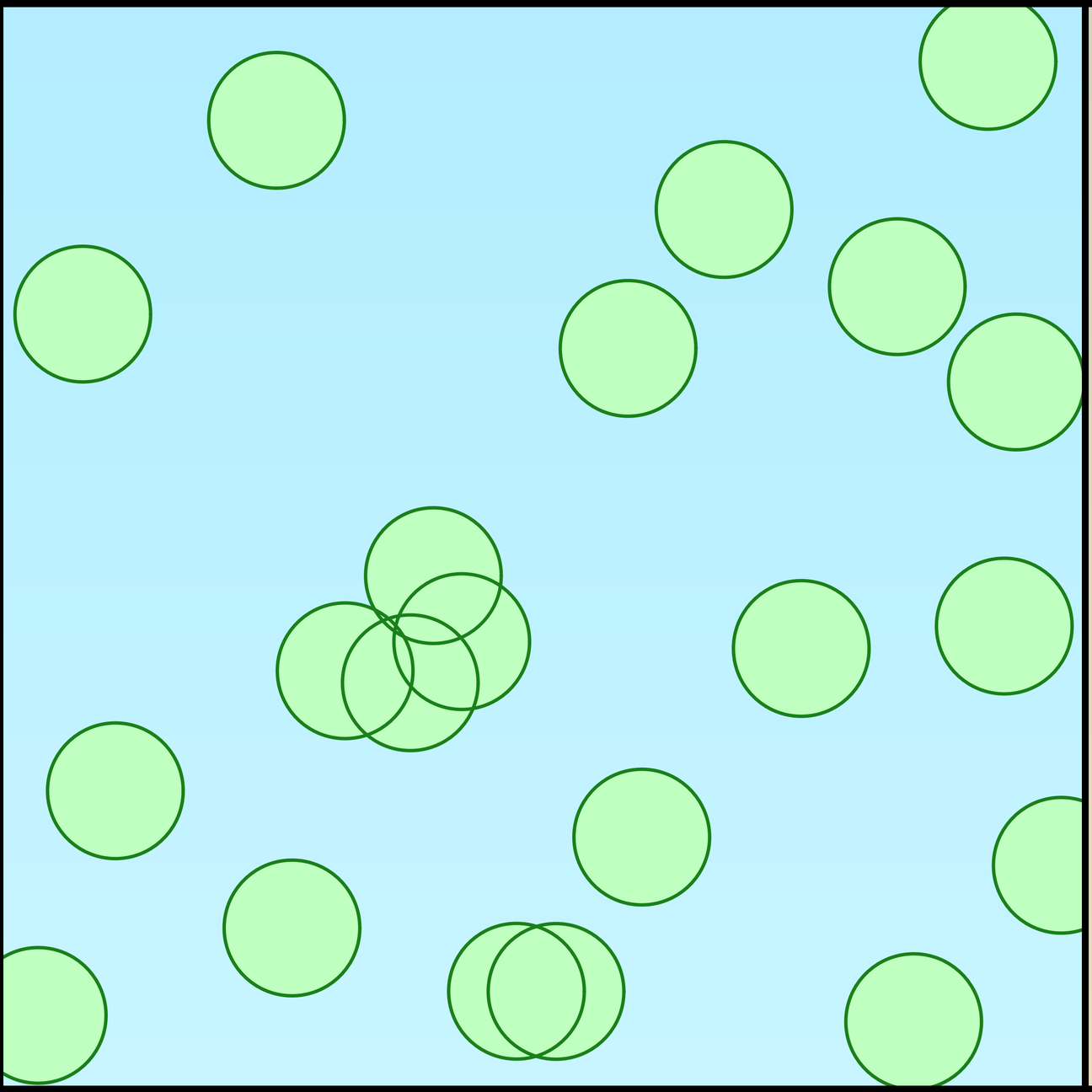
State of the art in 1D

State of the art in 2D

What if the regions in $\mathcal{R}$ are not disjoint, but only overlap a little bit?

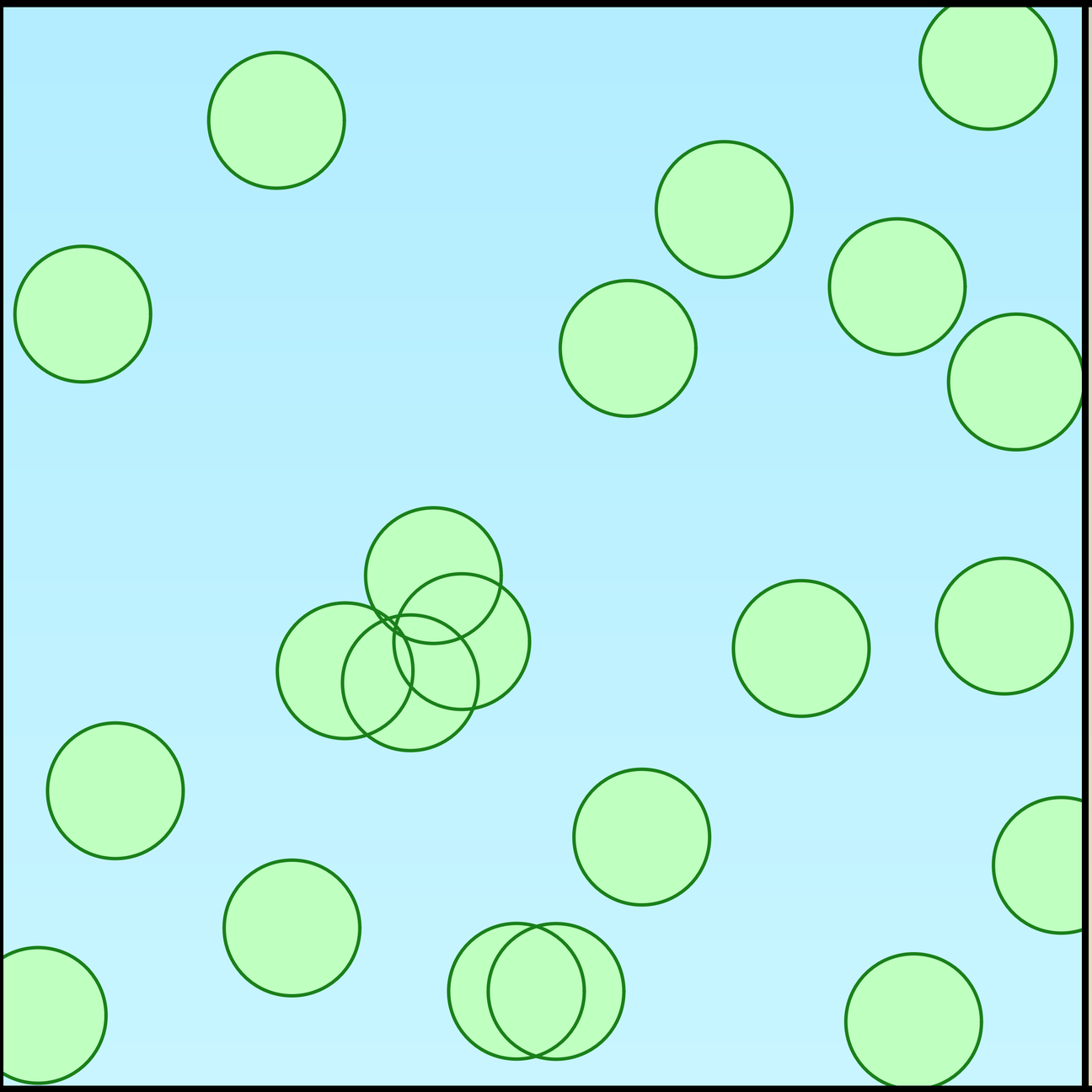


What if the regions in $\mathcal{R}$ are not disjoint, but only overlap a little bit?



What if the regions in $\mathcal{R}$ are not disjoint, but only overlap a little bit?

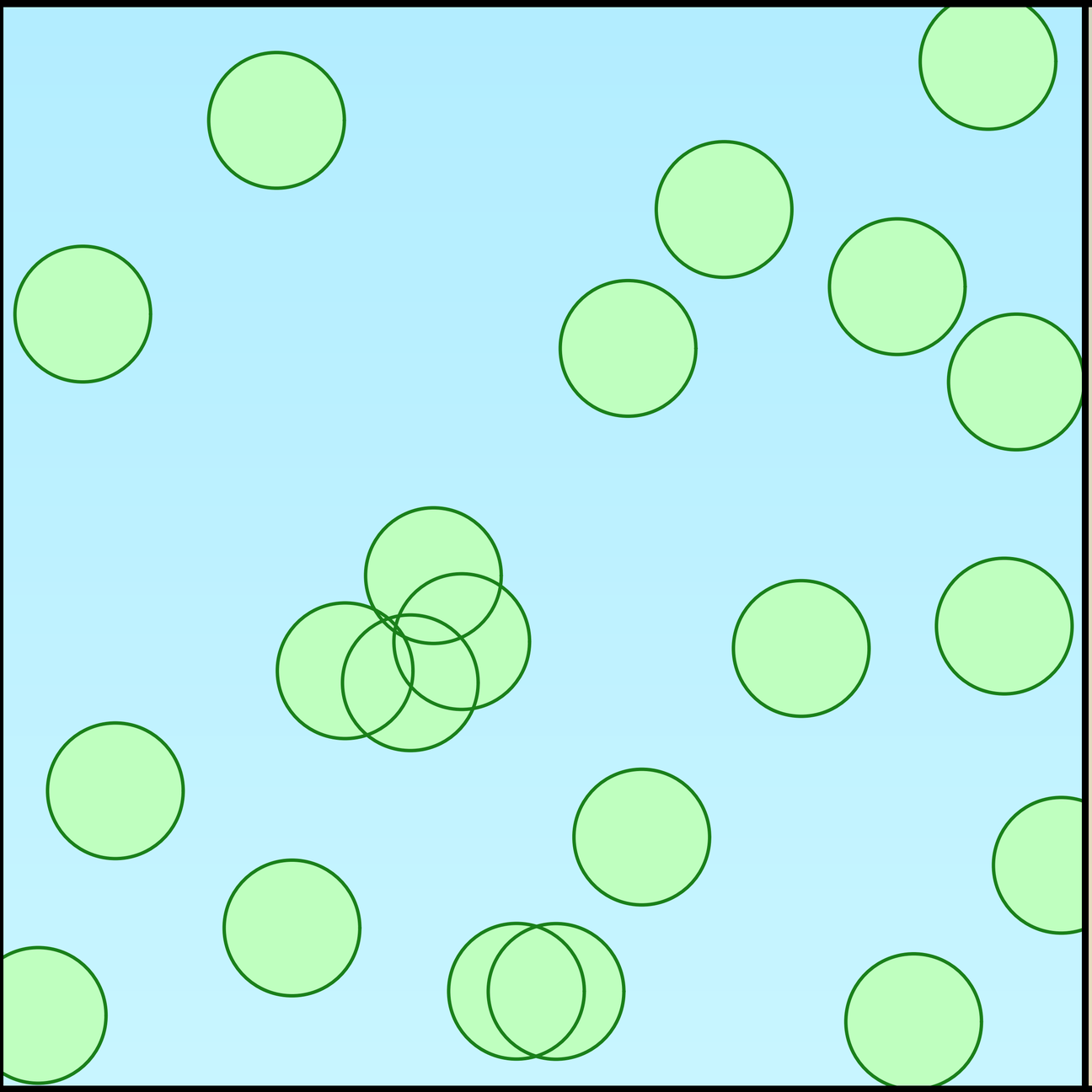
According to our theory so far, $\mathcal{R}$ does not contain information about any proximity structure.



What if the regions in $\mathcal{R}$ are not disjoint, but only overlap a little bit?

According to our theory so far, $\mathcal{R}$ does not contain information about any proximity structure.

Intuitively, this is not true.

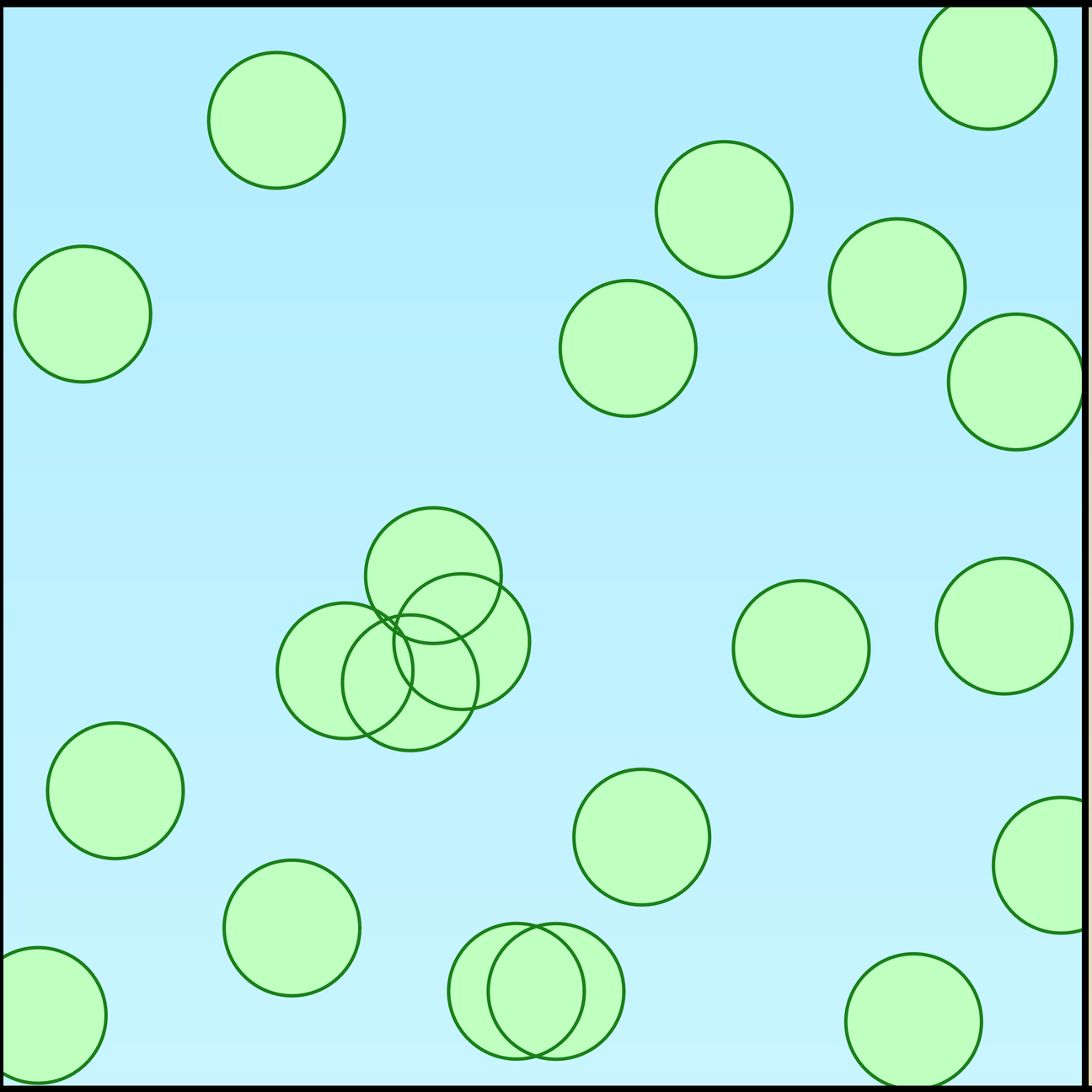


What if the regions in $\mathcal{R}$ are not disjoint, but only overlap a little bit?

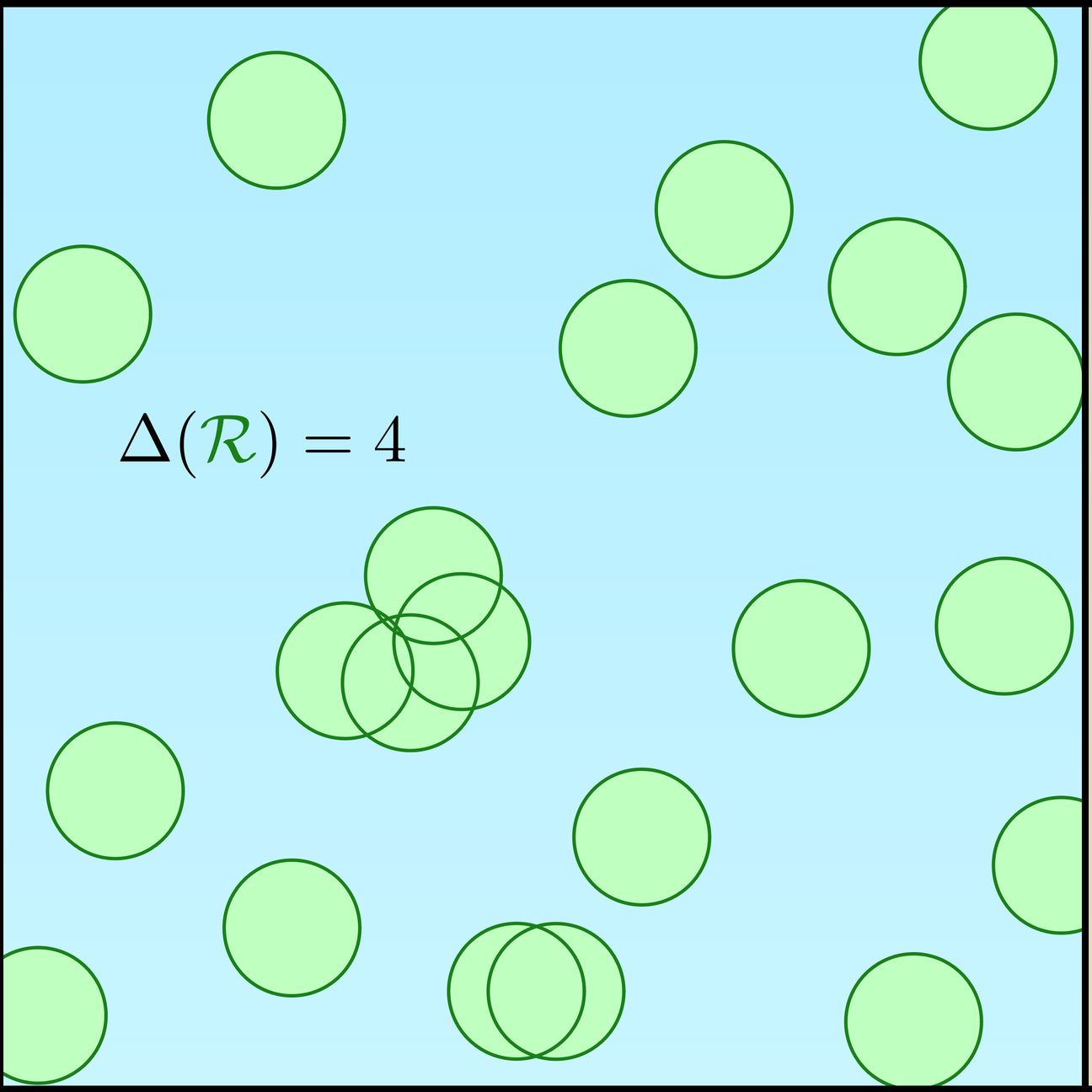
According to our theory so far, $\mathcal{R}$ does not contain information about any proximity structure.

Intuitively, this is not true.

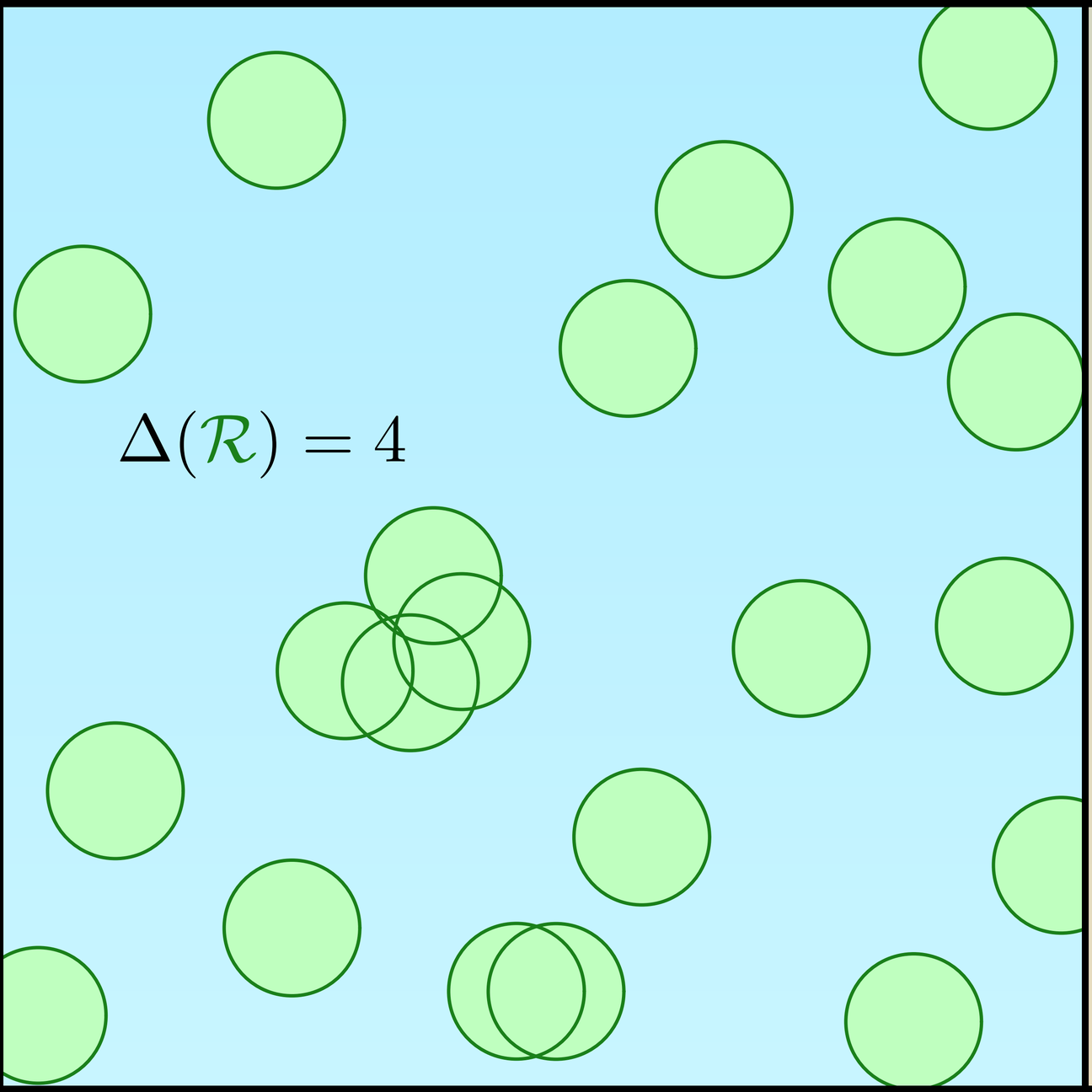
Can we measure the amount of overlap in $\mathcal{R}$?



The *ply* $\Delta(\mathcal{R})$ measures the maximum number of regions that contain a common point.

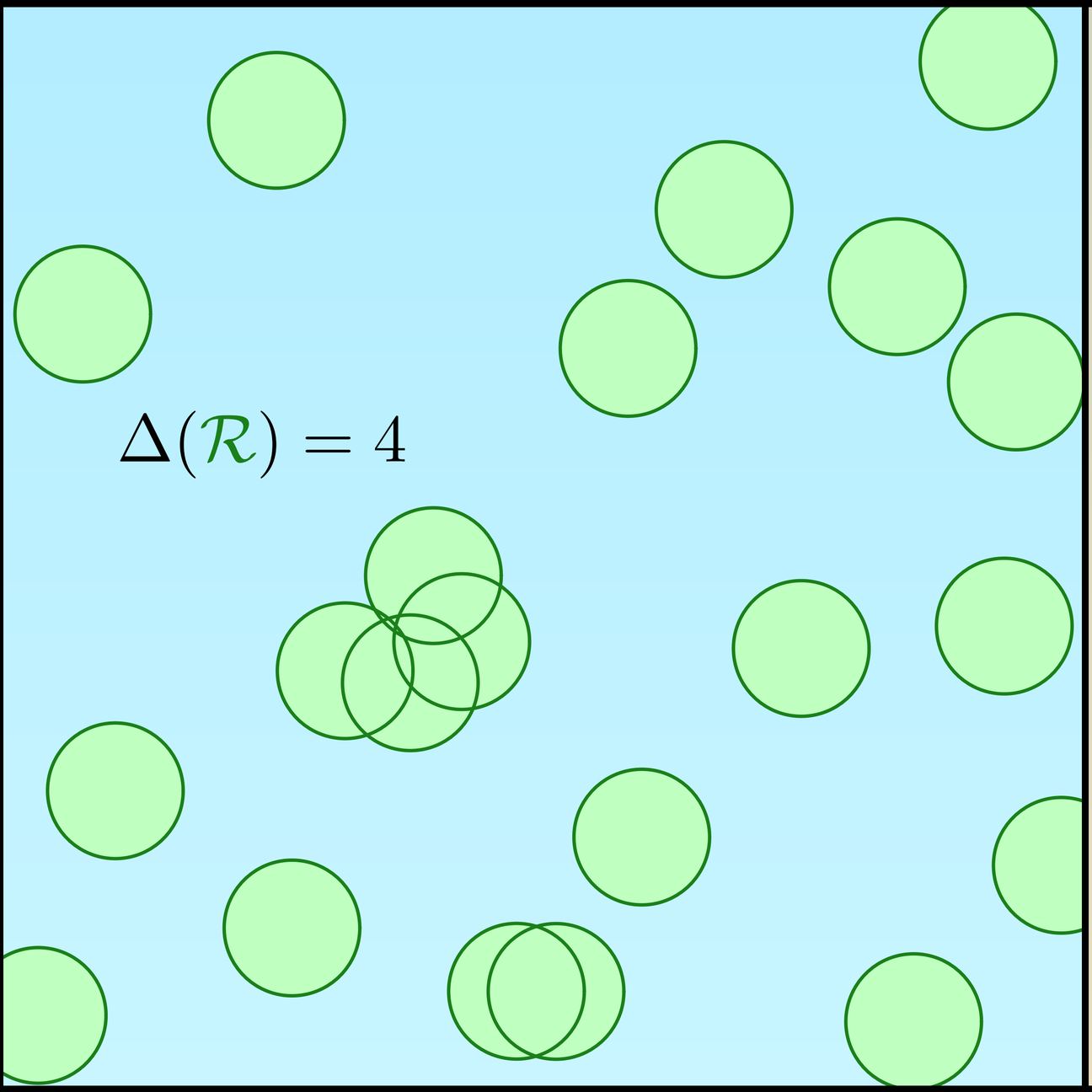



The *ply* $\Delta(\mathcal{R})$ measures the maximum number of regions that contain a common point.




The *ply* $\Delta(\mathcal{R})$ measures the maximum number of regions that contain a common point.

We may hope for reconstruction in time that depends on $\Delta$.

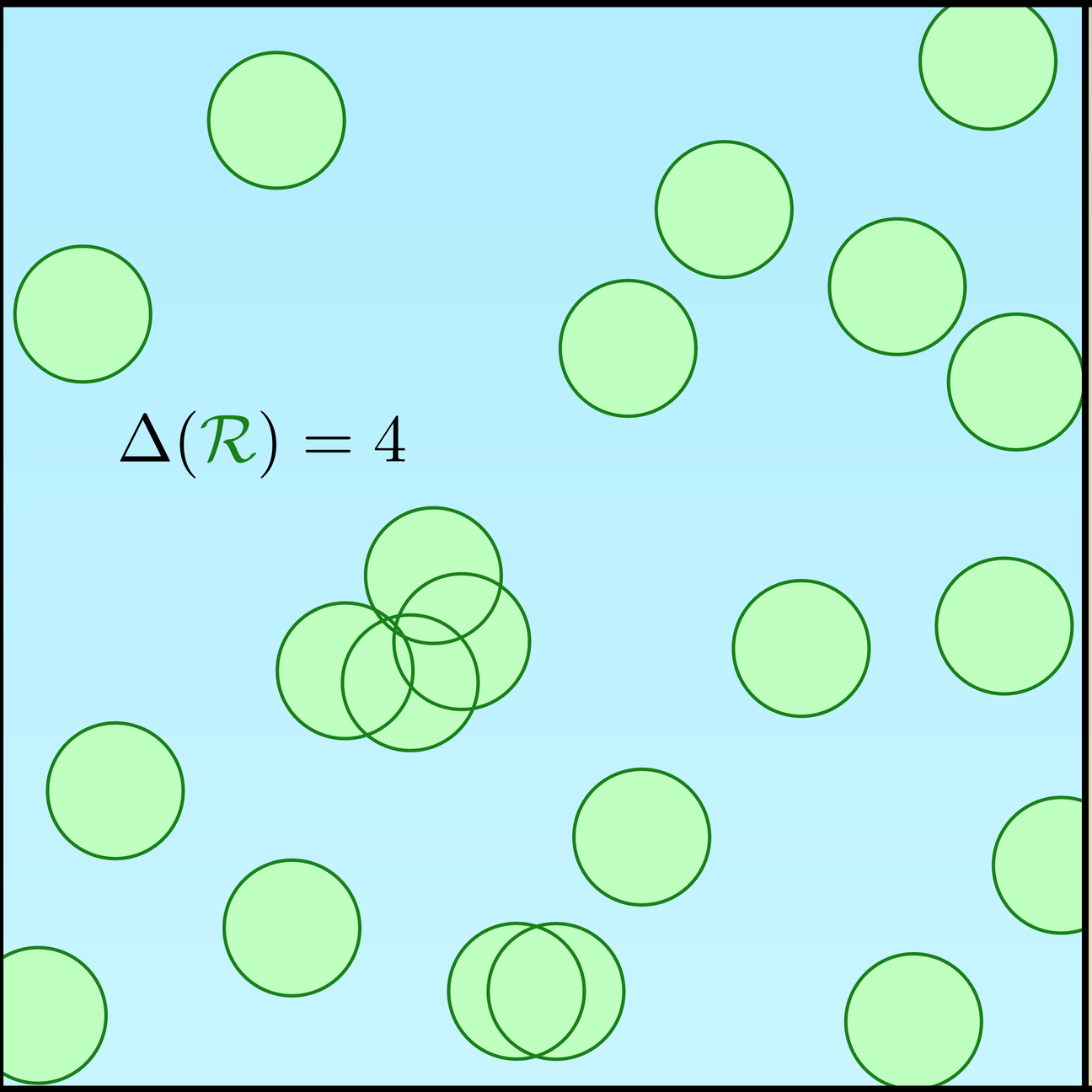



The *ply* $\Delta(\mathcal{R})$ measures the maximum number of regions that contain a common point.

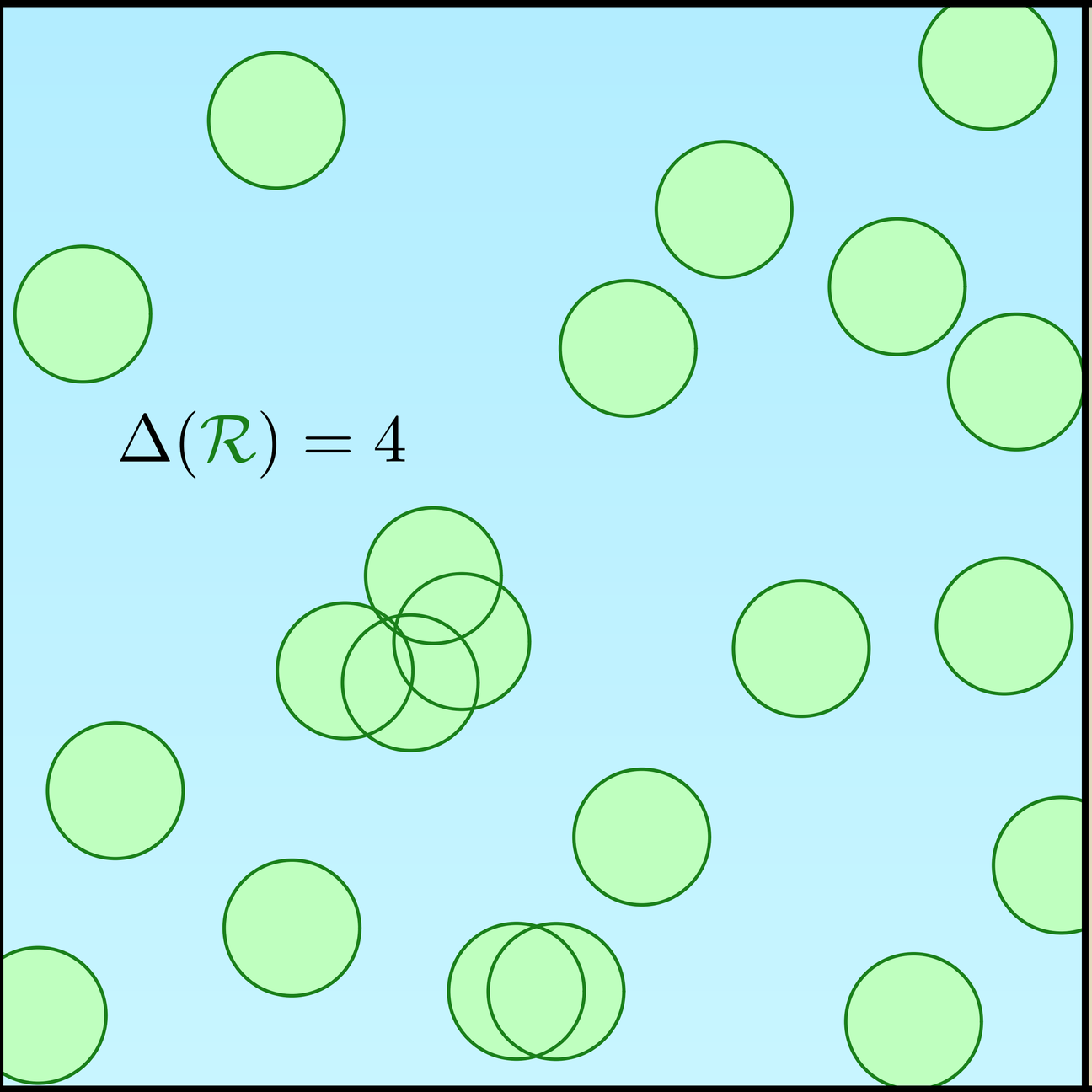
We may hope for reconstruction in time that depends on $\Delta$.

For example: the Delaunay triangulation can actually be reconstructed in $O(n \log \Delta)$ time.

[Buchin, Löffler, Morin & Mulzer, 2011]

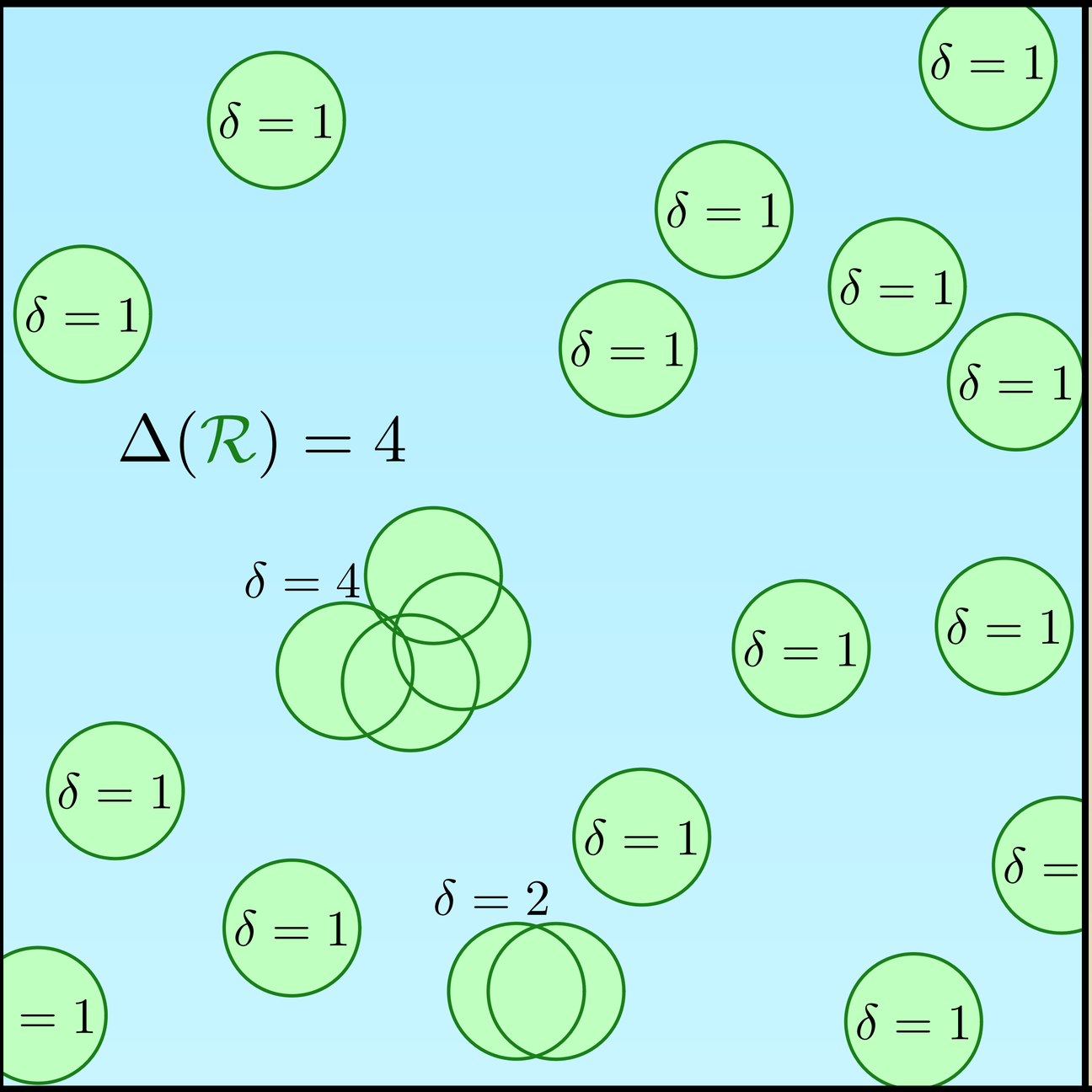



The ply is a *global* measure of overlap.



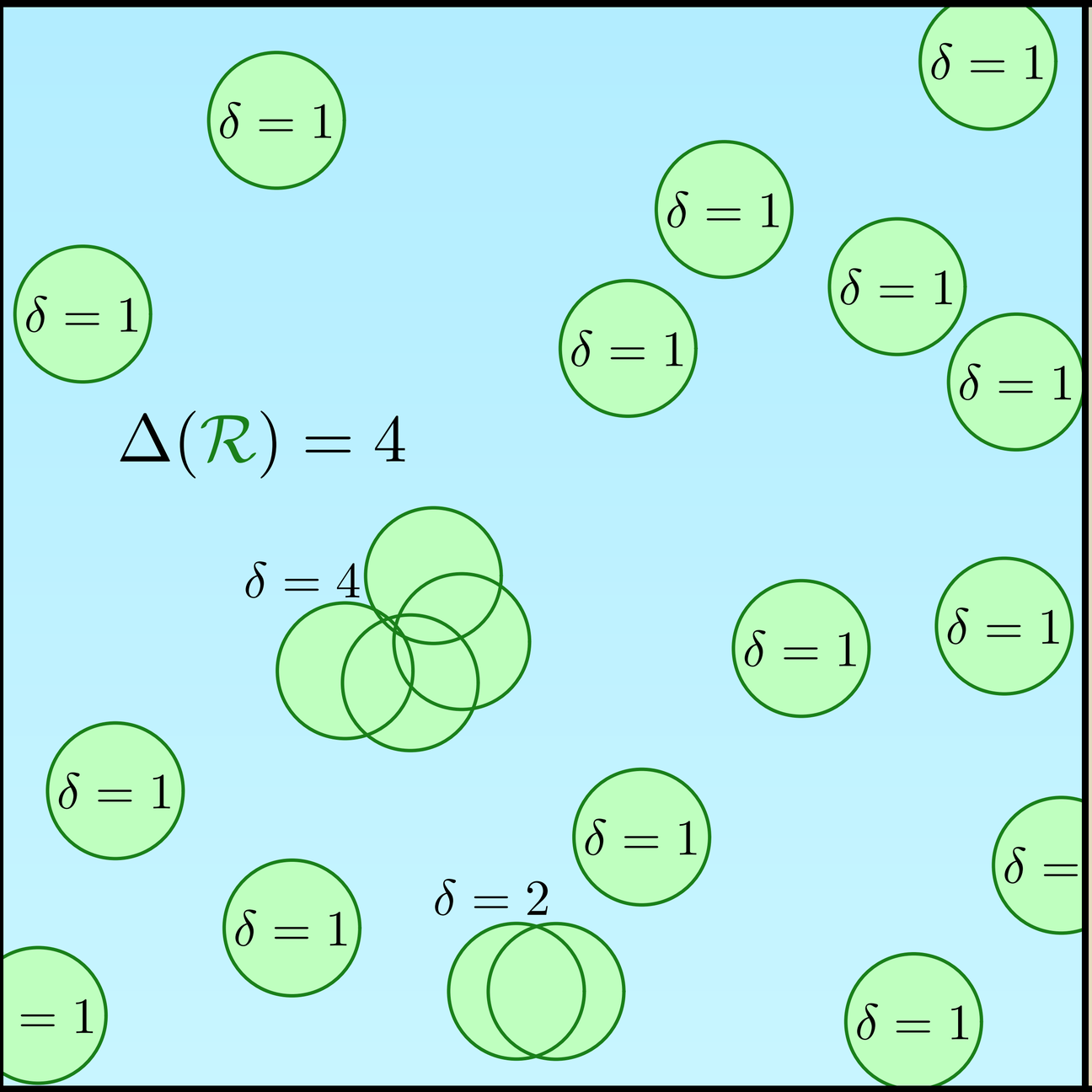

The ply is a *global* measure of overlap.

Instead, we can consider the *local* ply $\delta(R)$ of each region $R \in \mathcal{R}$.




The ply is a *global* measure of overlap.

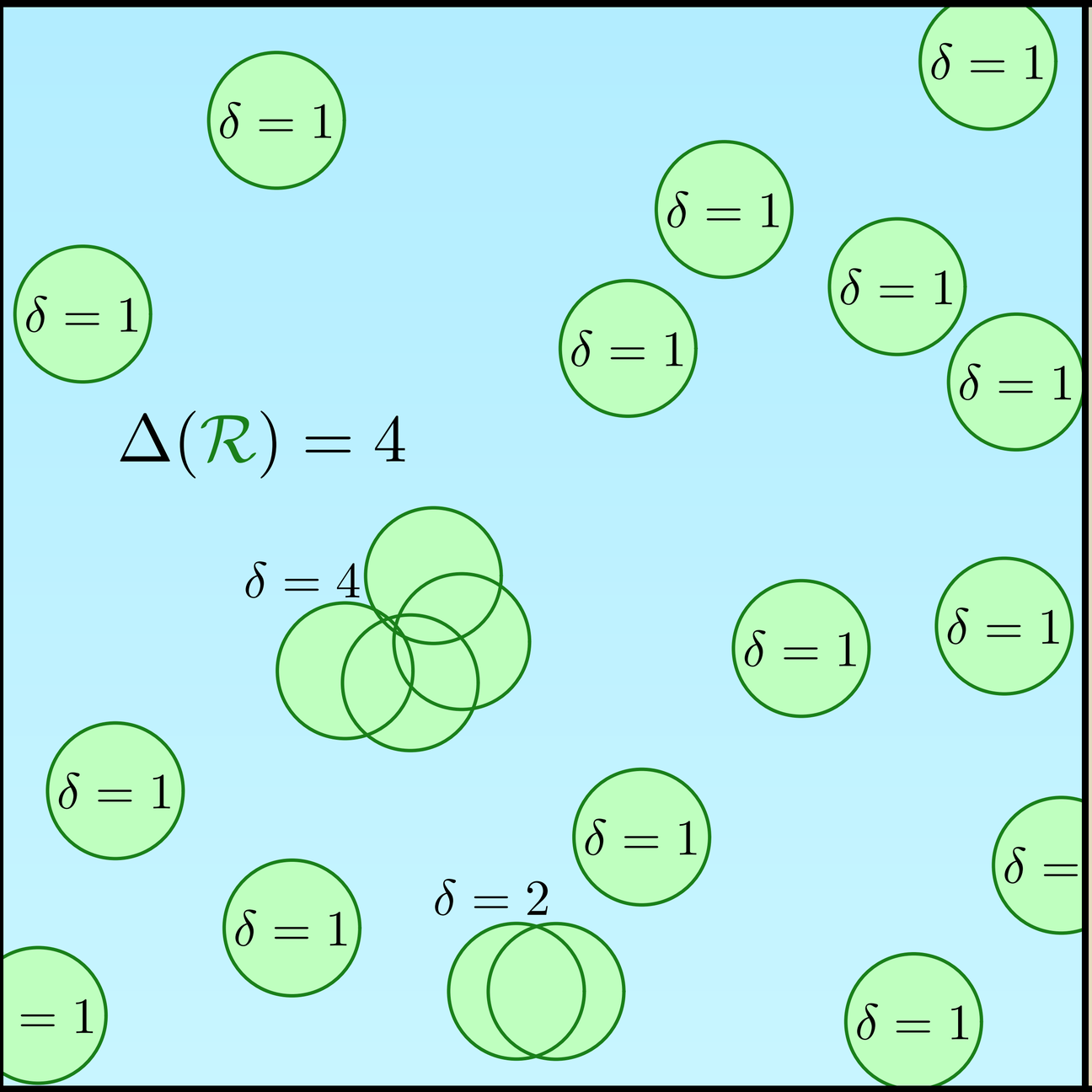
Instead, we can consider the *local* ply $\delta(R)$ of each region $R \in \mathcal{R}$.




The ply is a *global* measure of overlap.

Instead, we can consider the *local* ply $\delta(R)$ of each region $R \in \mathcal{R}$.

We want reconstruction time to only "locally" depend on $\delta(R)$.

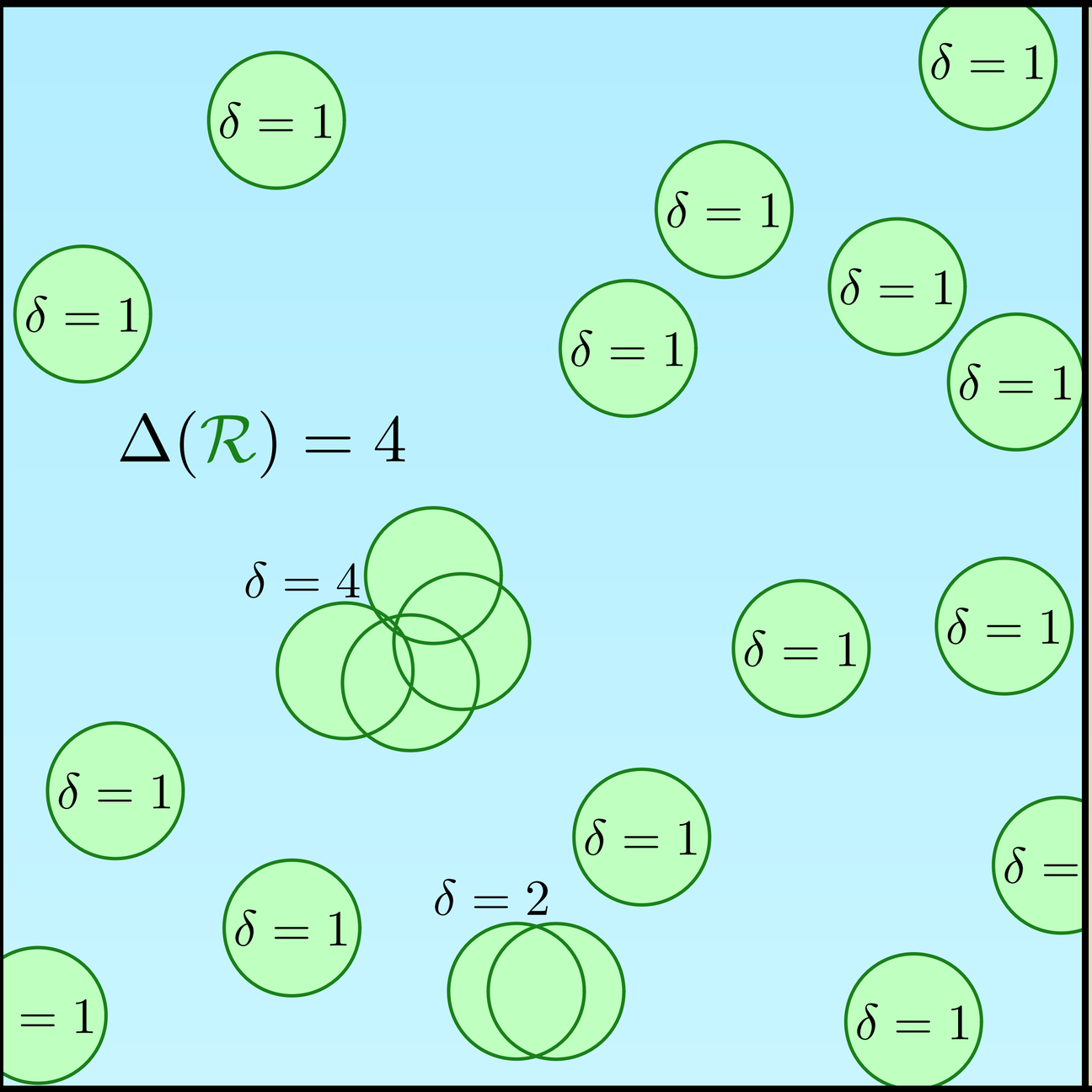



The ply is a *global* measure of overlap.

Instead, we can consider the *local* ply $\delta(R)$ of each region $R \in \mathcal{R}$.

We want reconstruction time to only "locally" depend on $\delta(R)$.

$$A(\mathcal{R}) = \sum \log \delta(R).$$




The ply is a *global* measure of overlap.

Instead, we can consider the *local* ply $\delta(R)$ of each region $R \in \mathcal{R}$.

We want reconstruction time to only "locally" depend on $\delta(R)$.

$$A(\mathcal{R}) = \sum \log \delta(R).$$

Is $O(A(\mathcal{R}))$ time possible?

# Chapter 3

*FINE-GRAINED MEASURES*

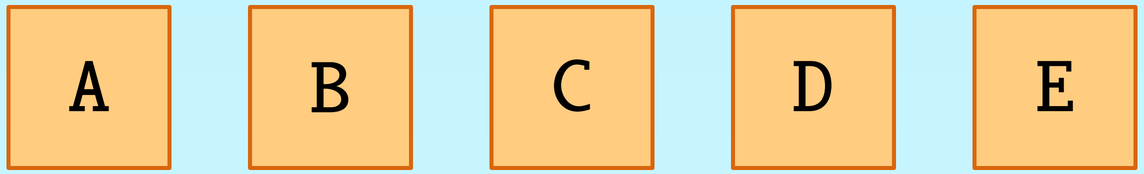
Ambiguity of regions

Information theory

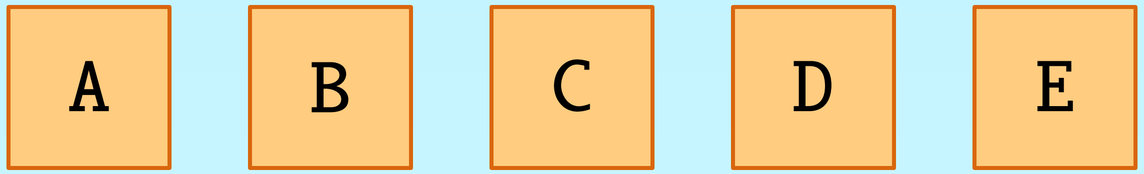
State of the art in 1D

State of the art in 2D

Consider all $N$ possible outputs of a computational problem.

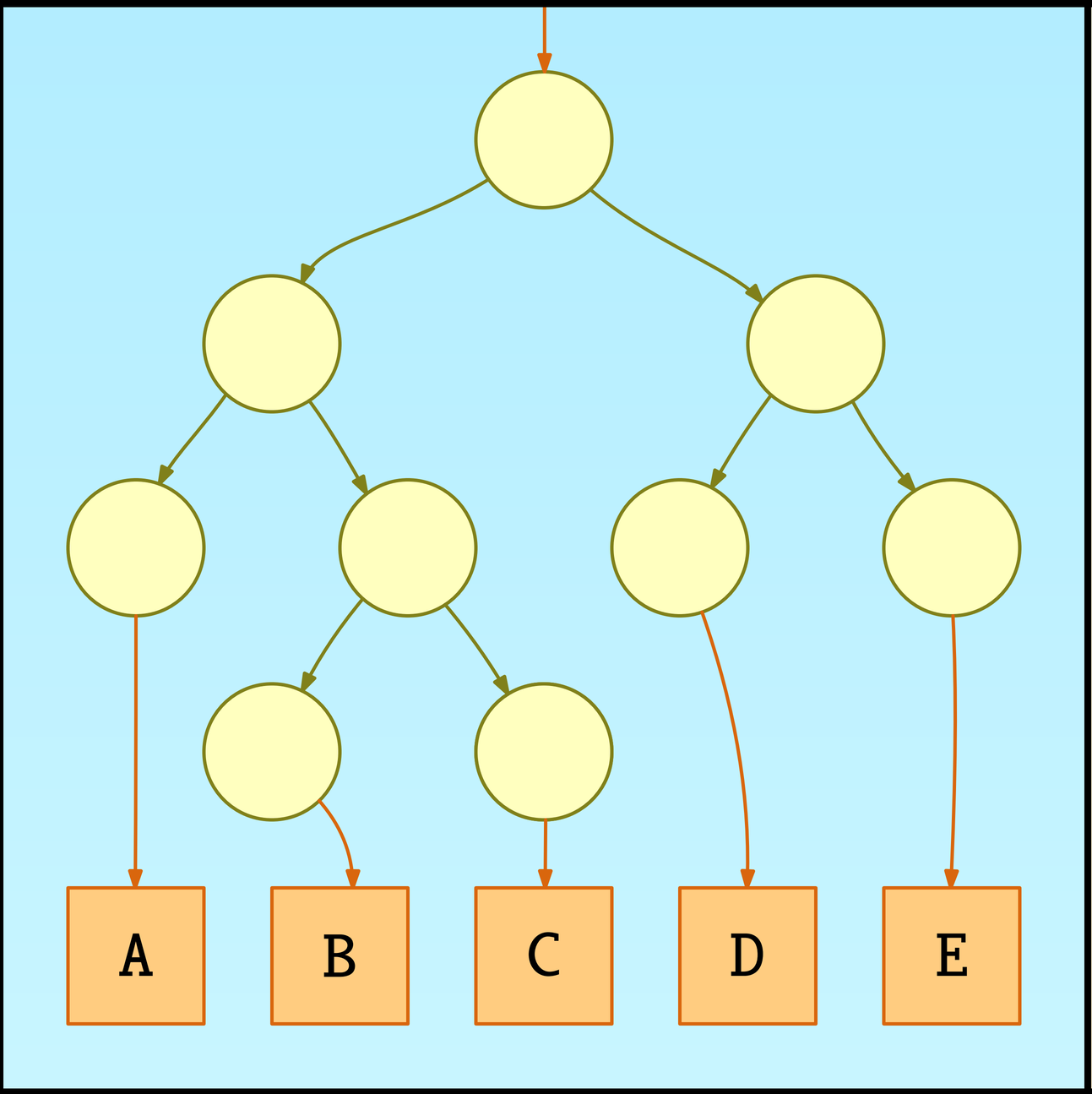



Consider all $N$ possible outputs of a computational problem.



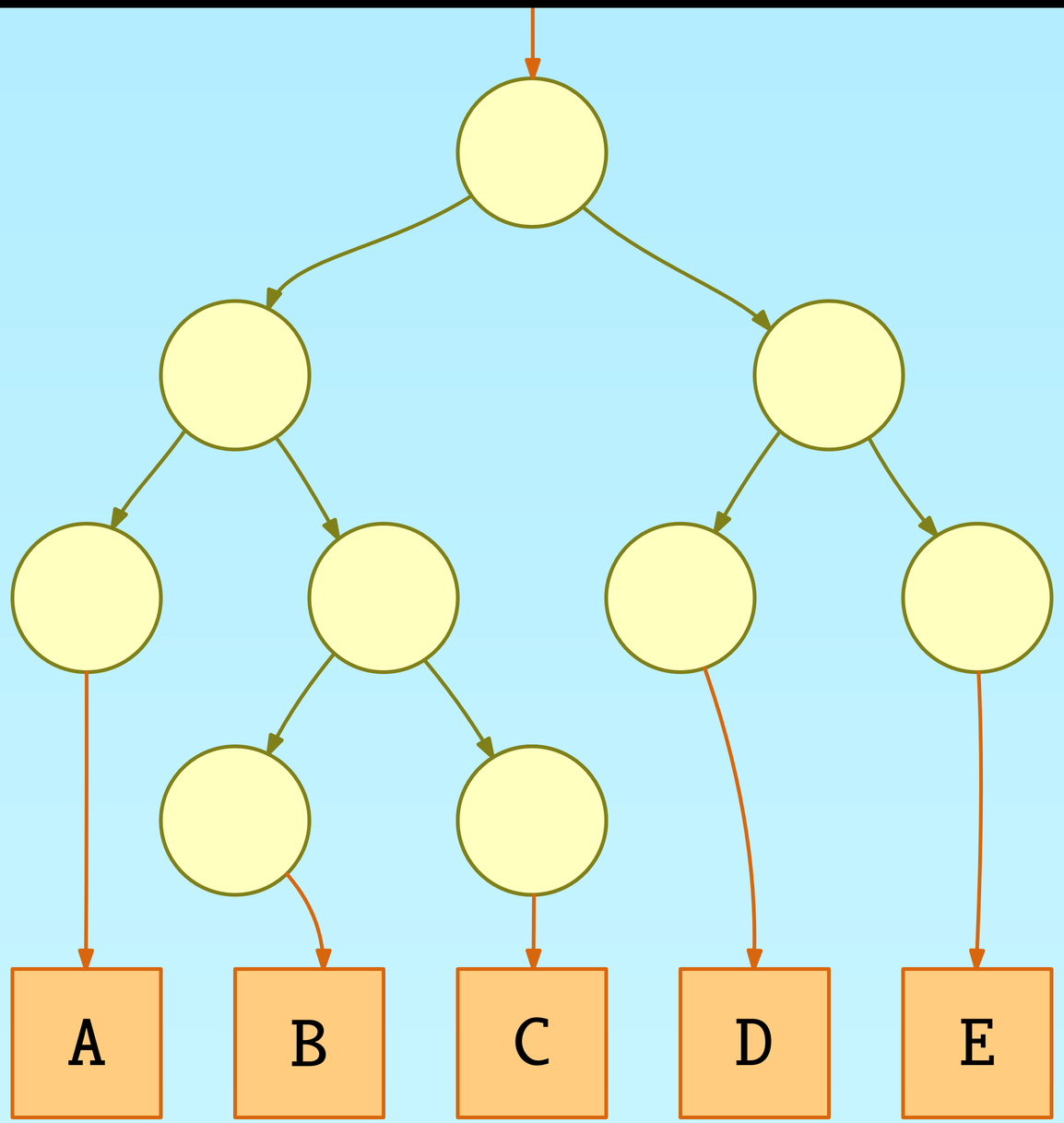

Consider all $N$ possible outputs of a computational problem.

Now consider the decisions an algorithm (on a pointer machine) must make to reach these outputs.




Consider all $N$ possible outputs of a computational problem.

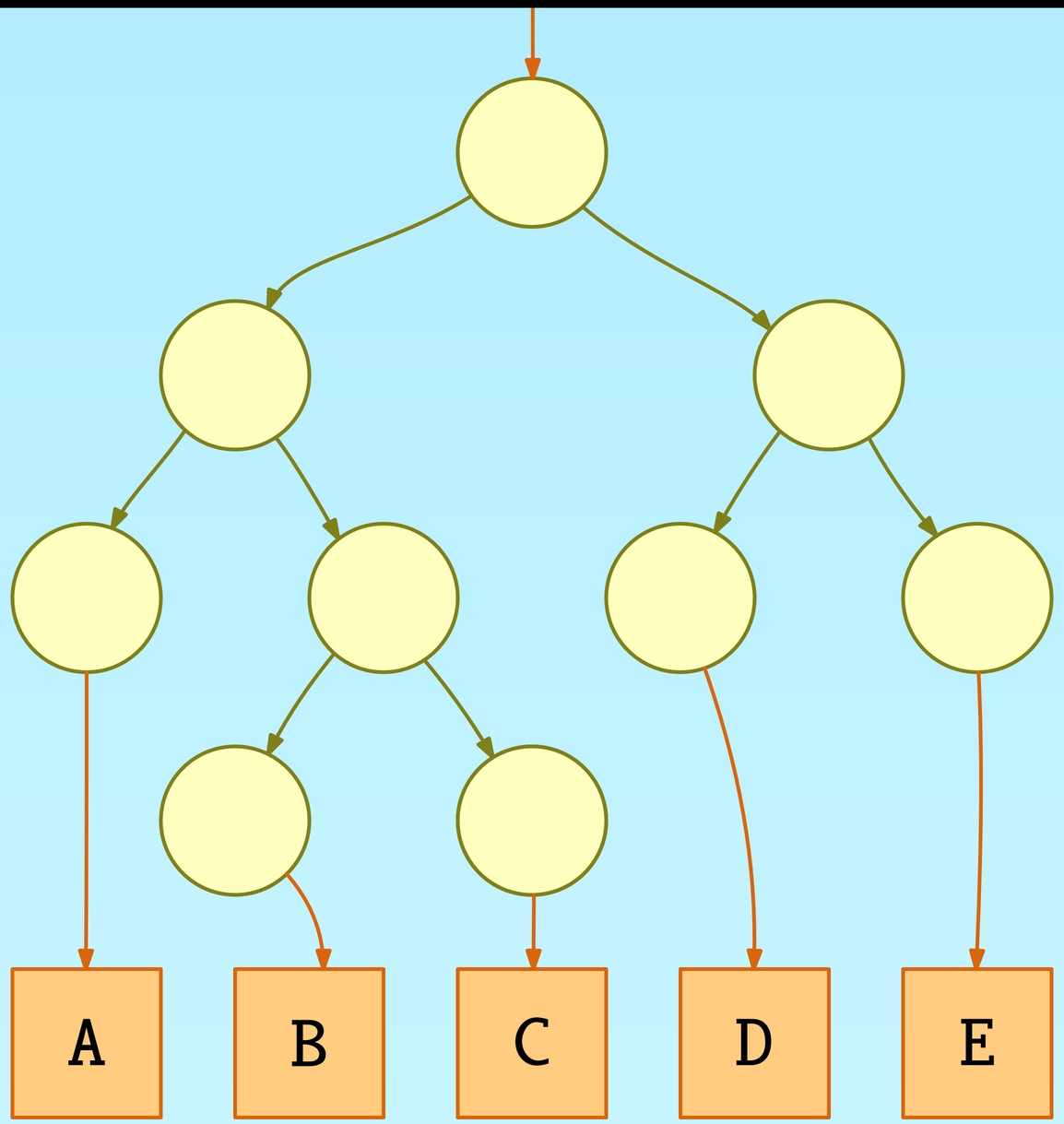
Now consider the decisions an algorithm (on a pointer machine) must make to reach these outputs.




Consider all $N$ possible outputs of a computational problem.

Now consider the decisions an algorithm (on a pointer machine) must make to reach these outputs.

The *height* of this tree is at least $\log N$.



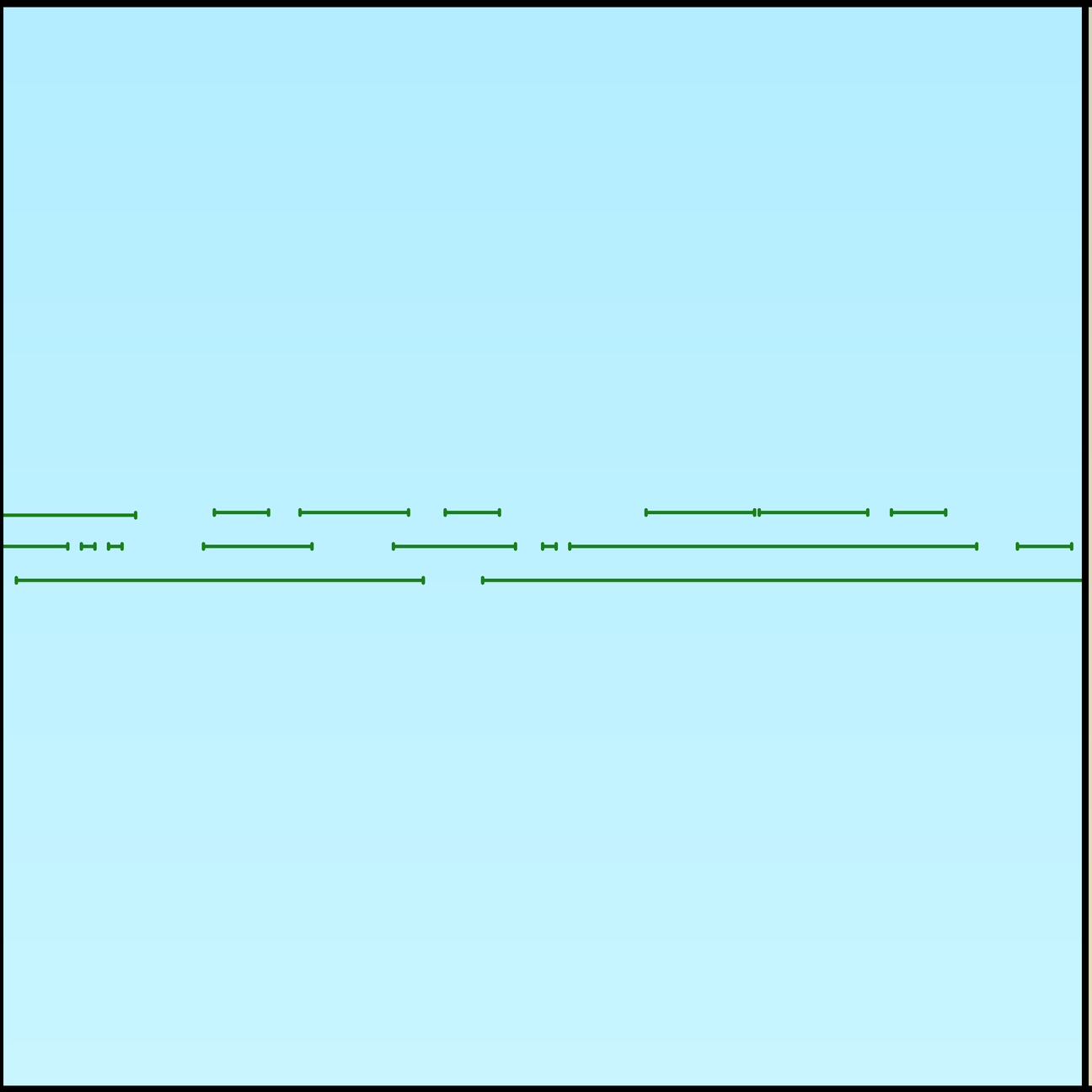

Consider all $N$ possible outputs of a computational problem.

Now consider the decisions an algorithm (on a pointer machine) must make to reach these outputs.

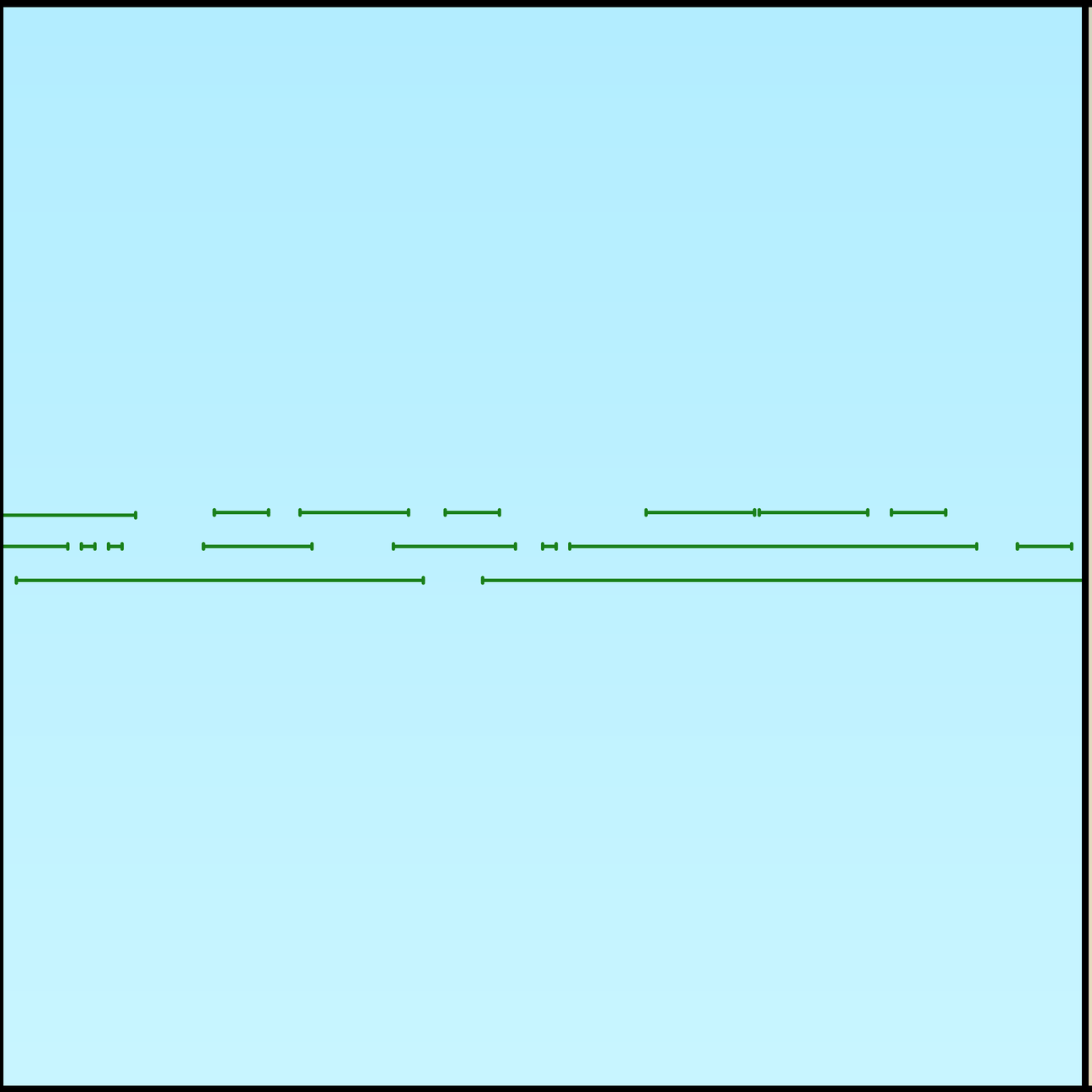
The *height* of this tree is at least $\log N$.

So, the worst-case running time is at least $\log N$.

Now, consider a set of overlapping regions in $\mathbb{R}^1$.

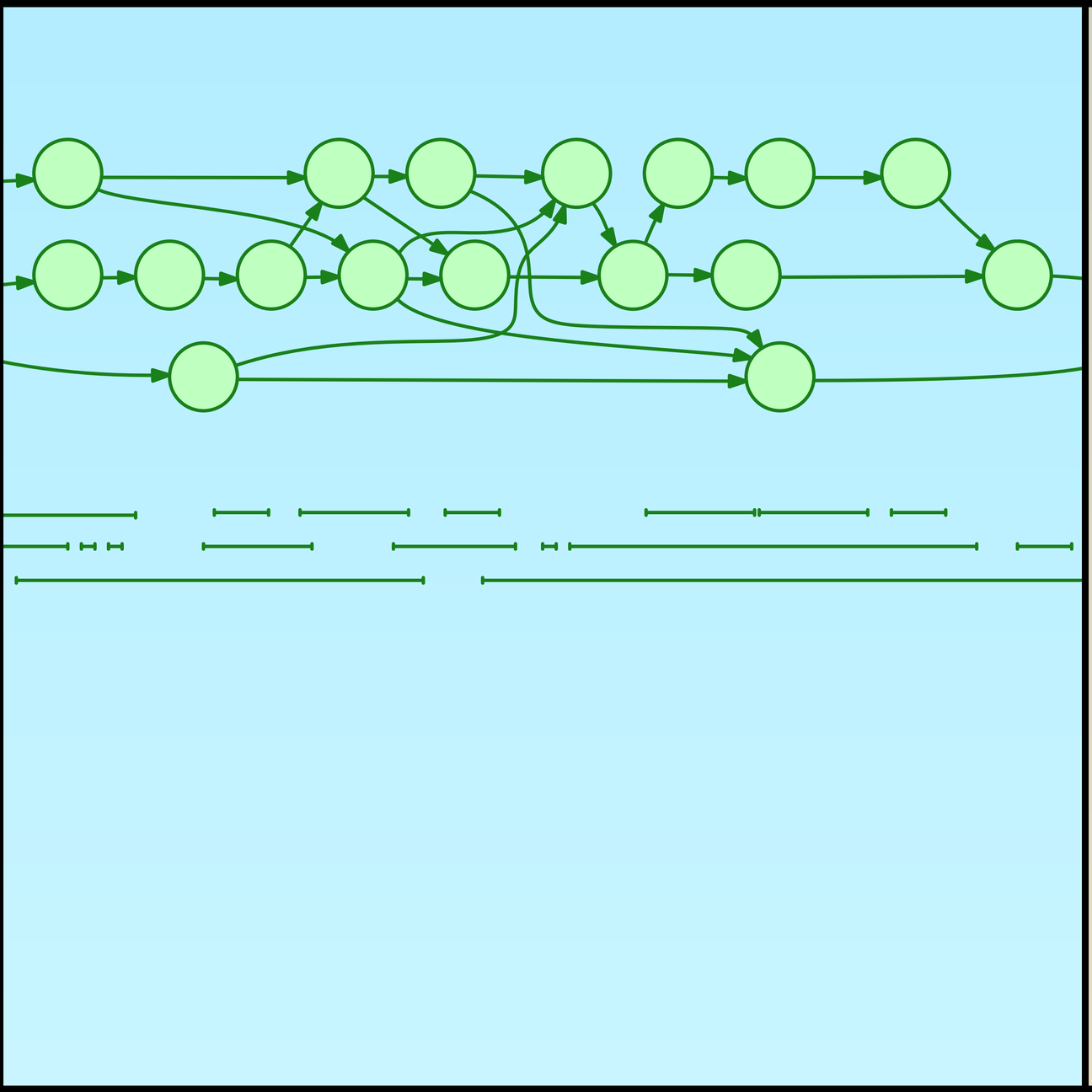


Now, consider a set of overlapping regions in $\mathbb{R}^1$.



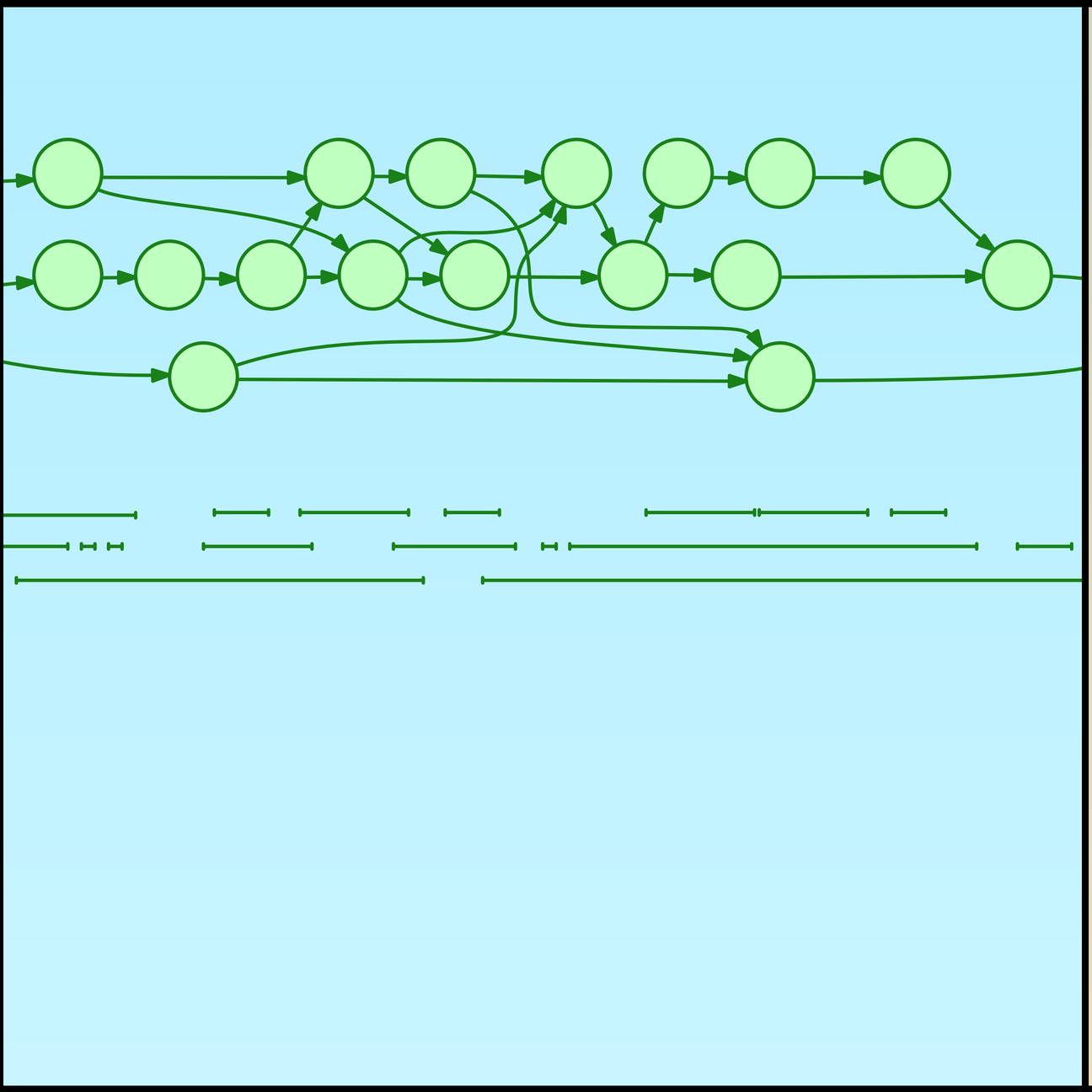
Now, consider a set of overlapping regions in $\mathbb{R}^1$.

These intervals induce a *partial order*.



Now, consider a set of overlapping regions in $\mathbb{R}^1$.

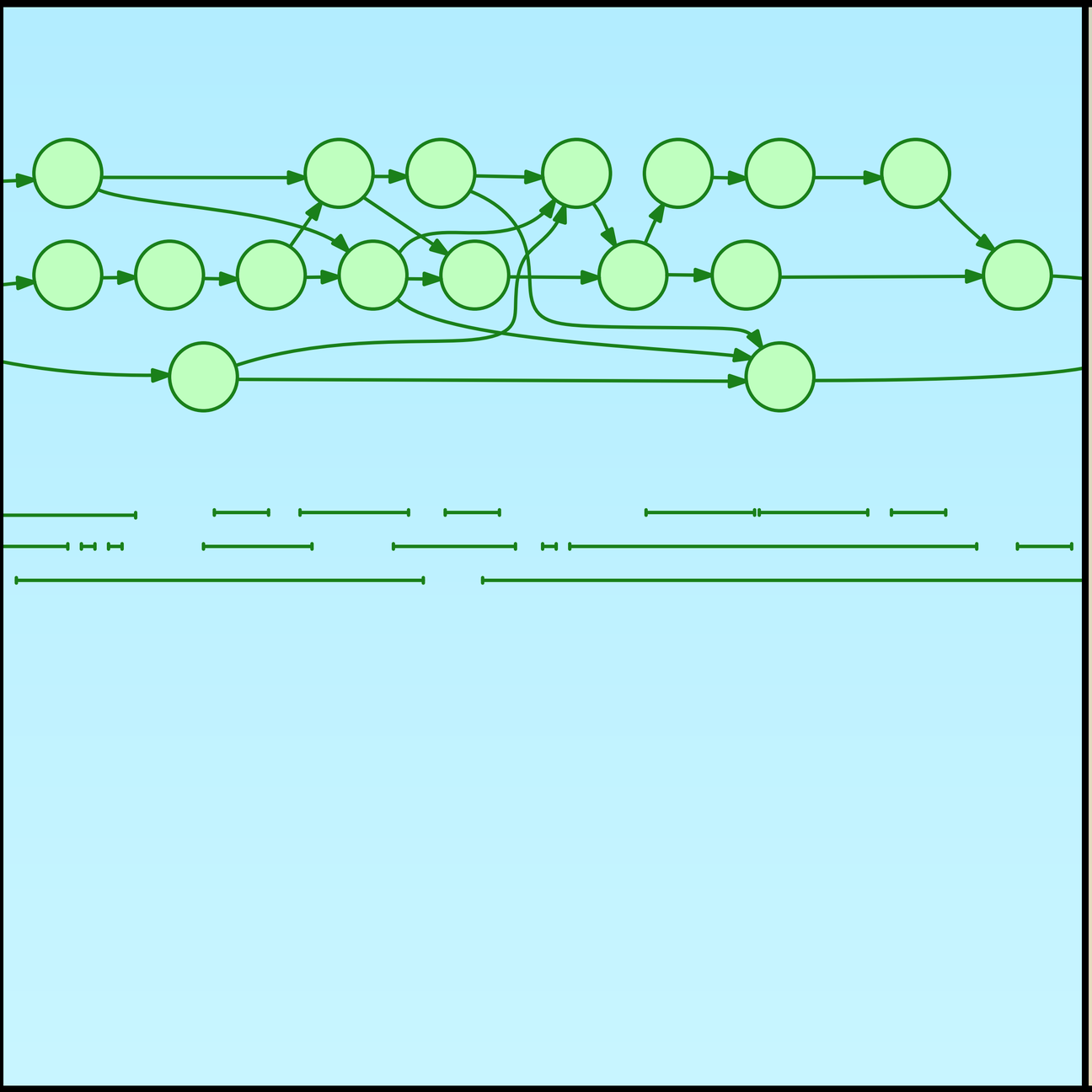
These intervals induce a *partial order*.



Now, consider a set of overlapping regions in $\mathbb{R}^1$.

These intervals induce a *partial order*.

The number of possible outputs is the number of total orders compatible with this partial order.



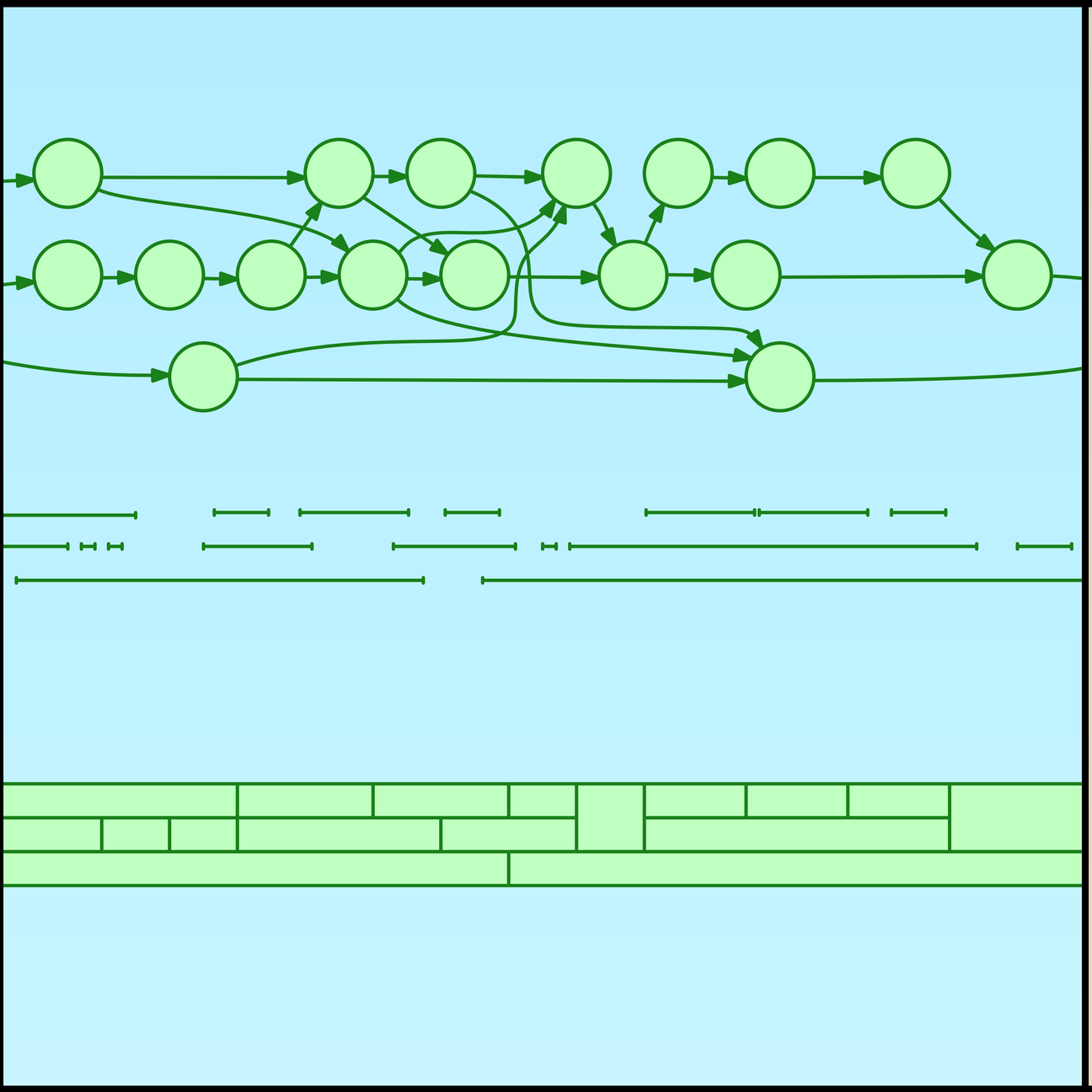
Now, consider a set of overlapping regions in $\mathbb{R}^1$.

These intervals induce a *partial order*.

The number of possible outputs is the number of total orders compatible with this partial order.

This number is roughly the *entropy* of the order.

[Cardinal, Fiorini, Joret, Jungers & Munro, 2013]



Now, consider a set of overlapping regions in $\mathbb{R}^1$.

These intervals induce a *partial order*.

The number of possible outputs is the number of total orders compatible with this partial order.

This number is roughly the *entropy* of the order.

[Cardinal, Fiorini, Joret, Jungers & Munro, 2013]

# Chapter 3

*FINE-GRAINED MEASURES*

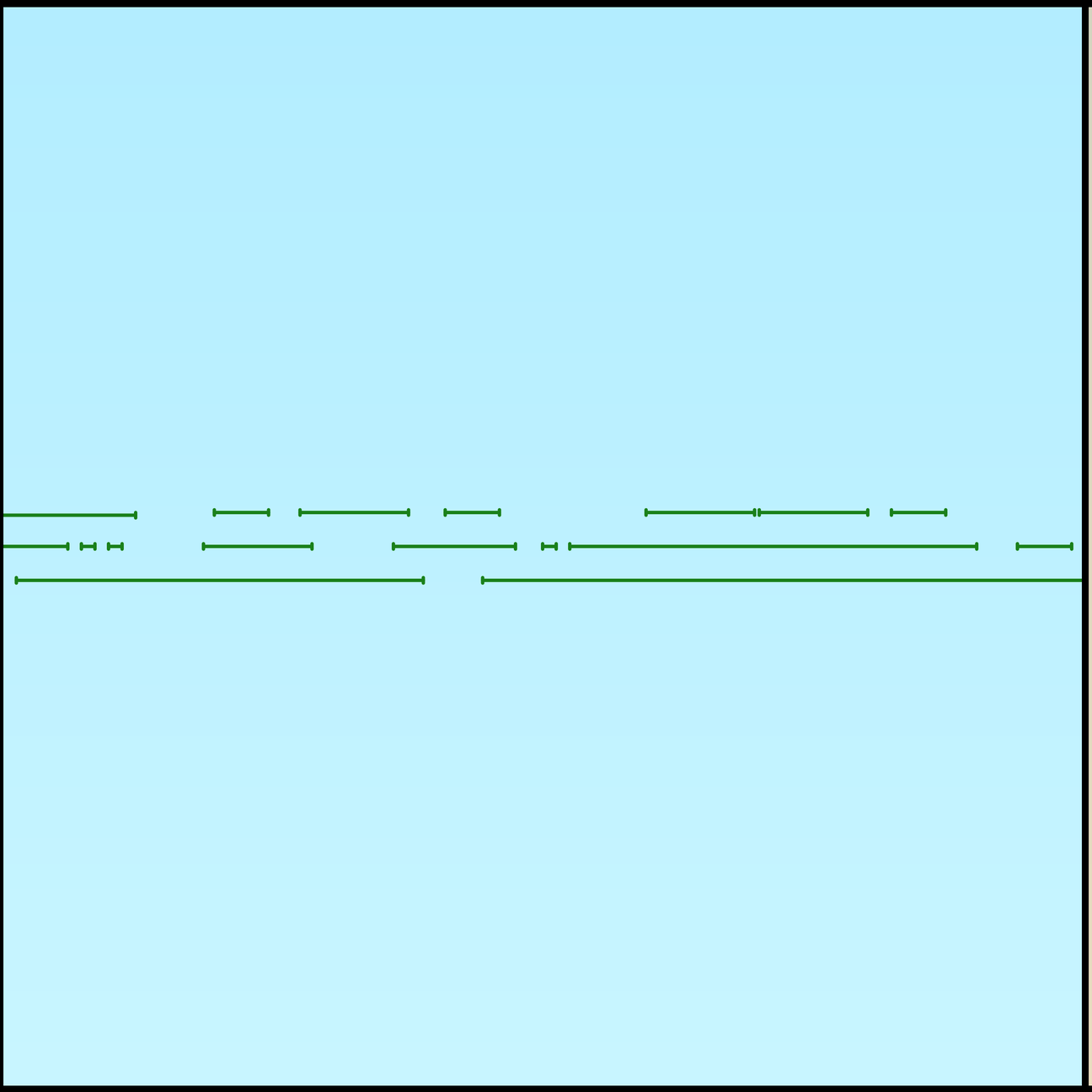
Ambiguity of regions

Information theory

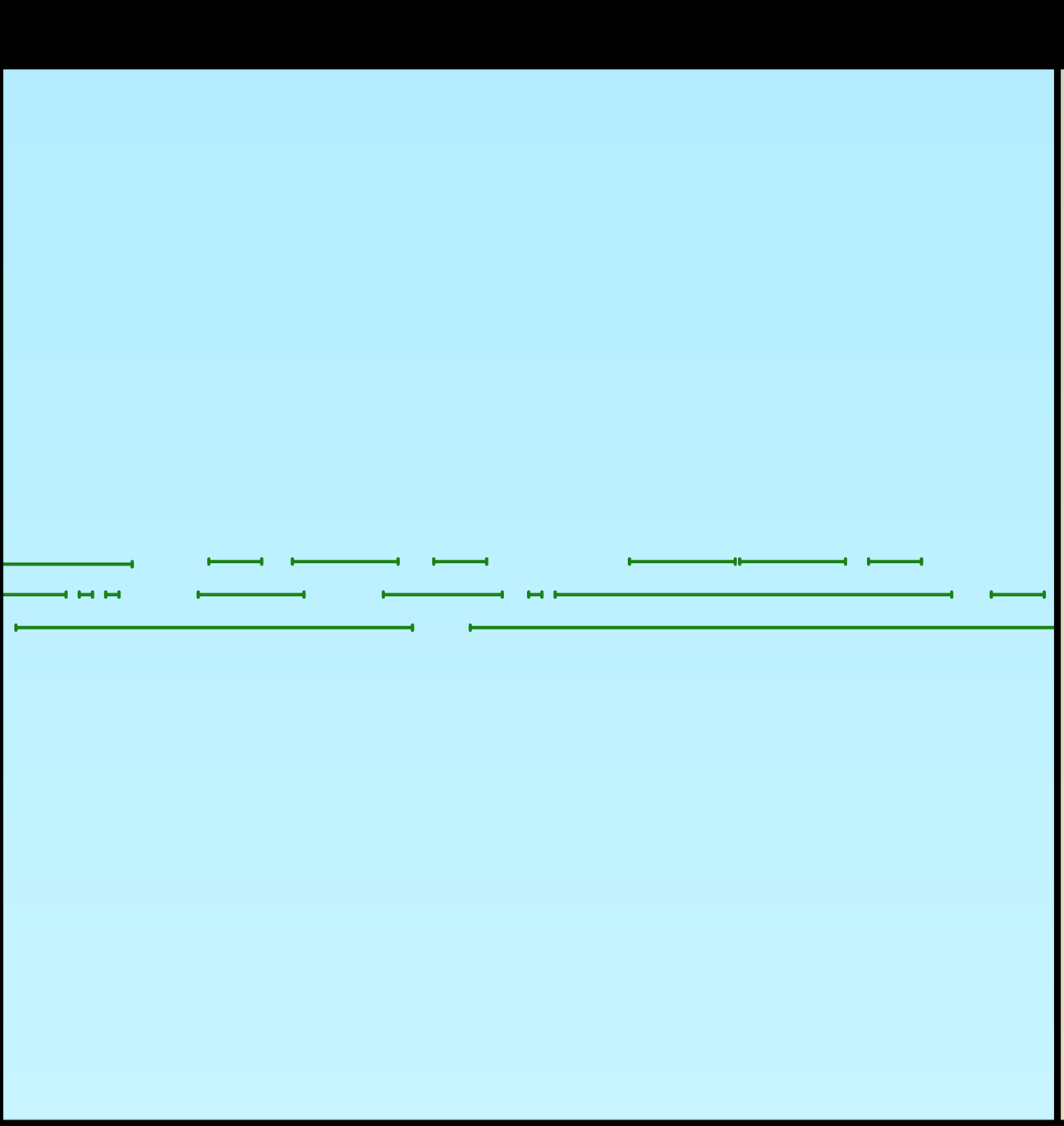
State of the art in 1D

State of the art in 2D

Let $\mathcal{R}$ be a set of partially overlapping intervals.

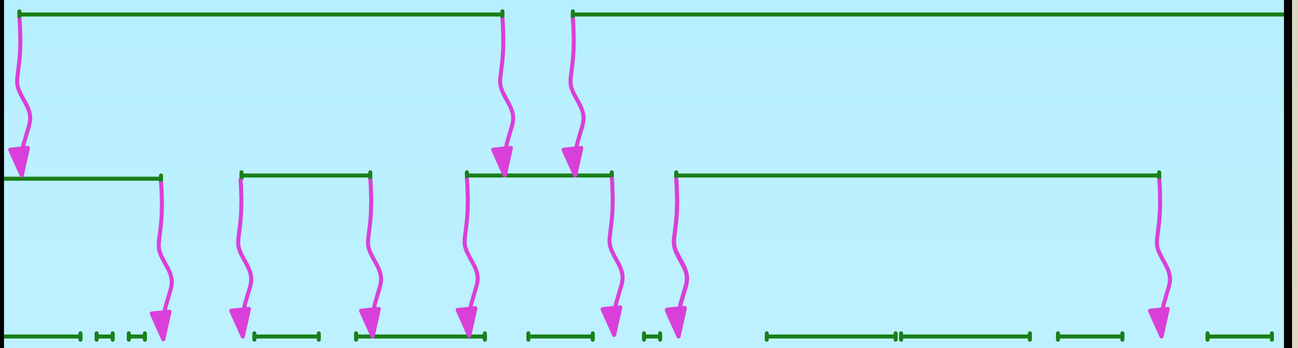


Let $\mathcal{R}$ be a set of partially overlapping intervals.



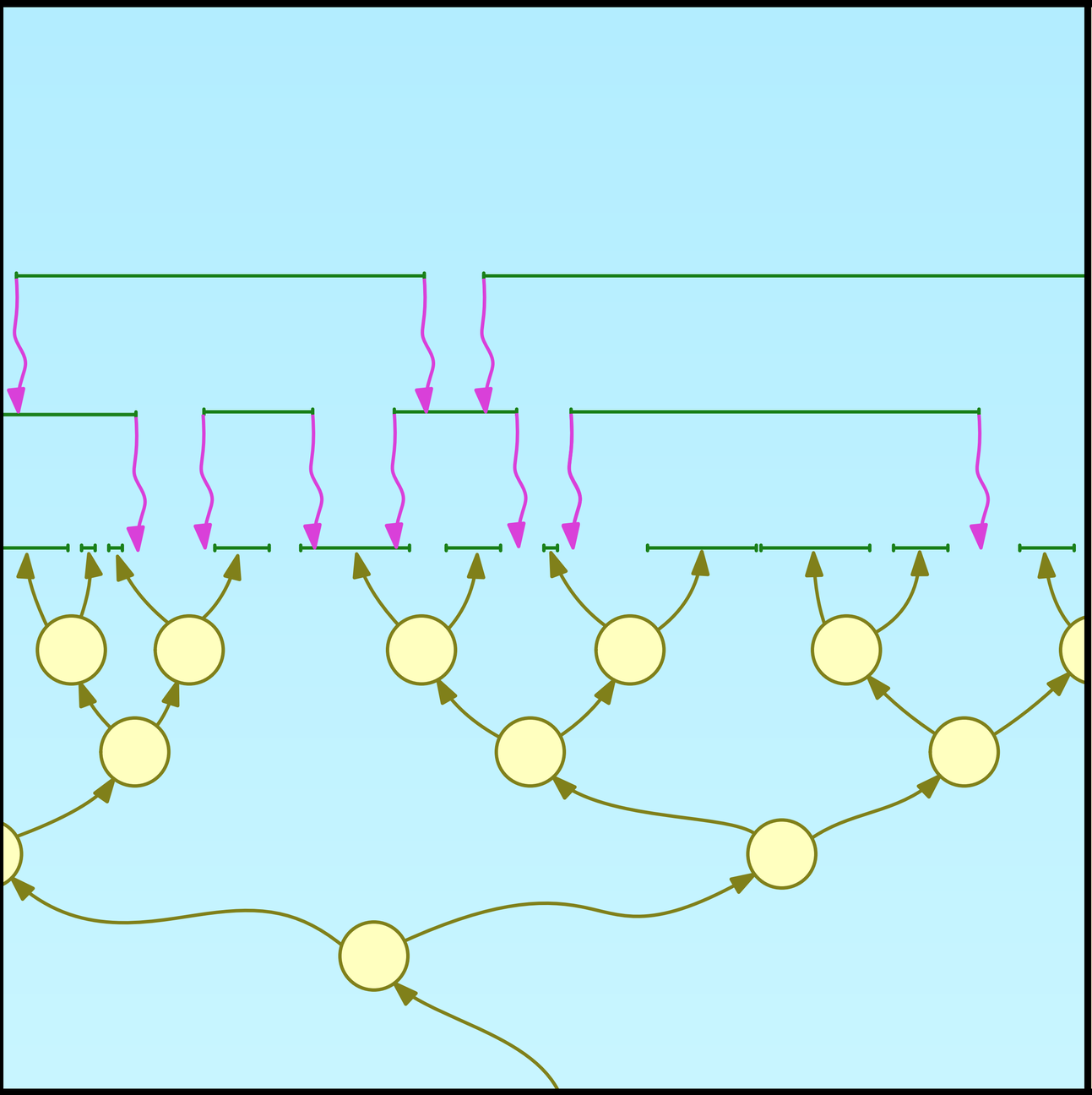
Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.



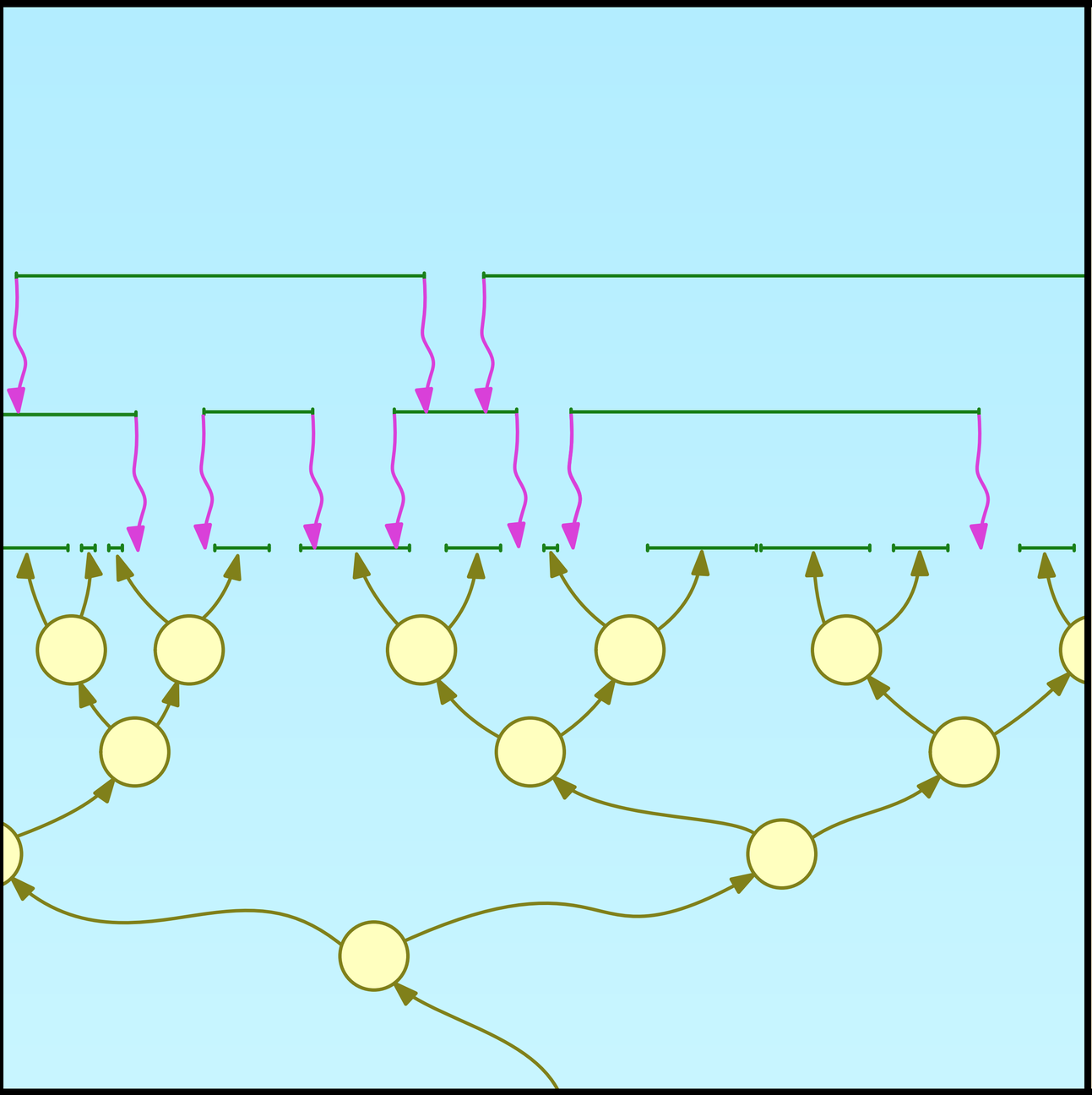
Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.



Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.

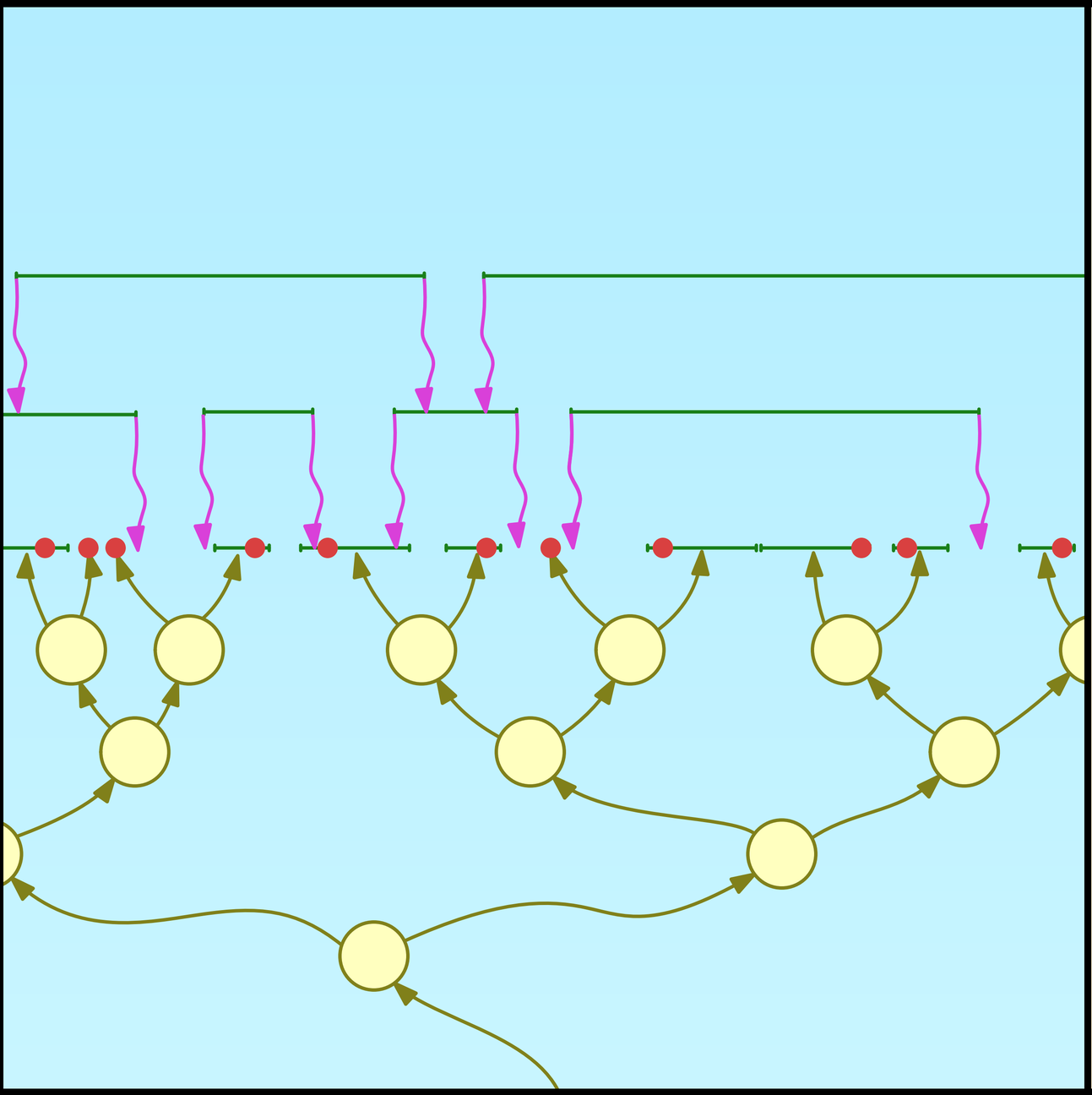


Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.

Using $F(\mathcal{R})$, we can sort $P$ in $A(\mathcal{R}) + n$ time.

[van der Hoog, Kostitsyna, Löffler & Speckmann, 2019]

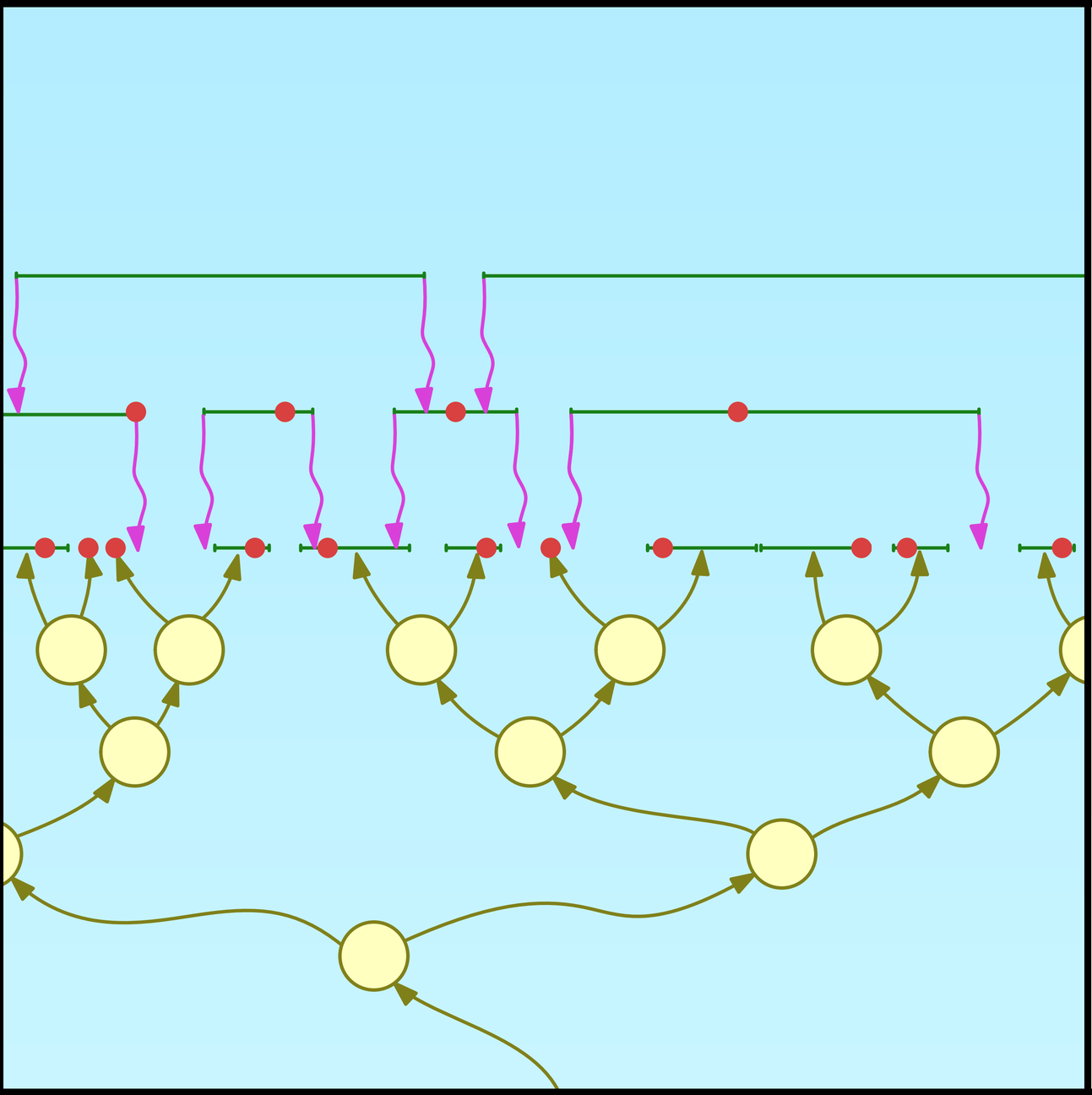


Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.

Using $F(\mathcal{R})$, we can sort $P$ in $A(\mathcal{R}) + n$ time.

[van der Hoog, Kostitsyna, Löffler & Speckmann, 2019]

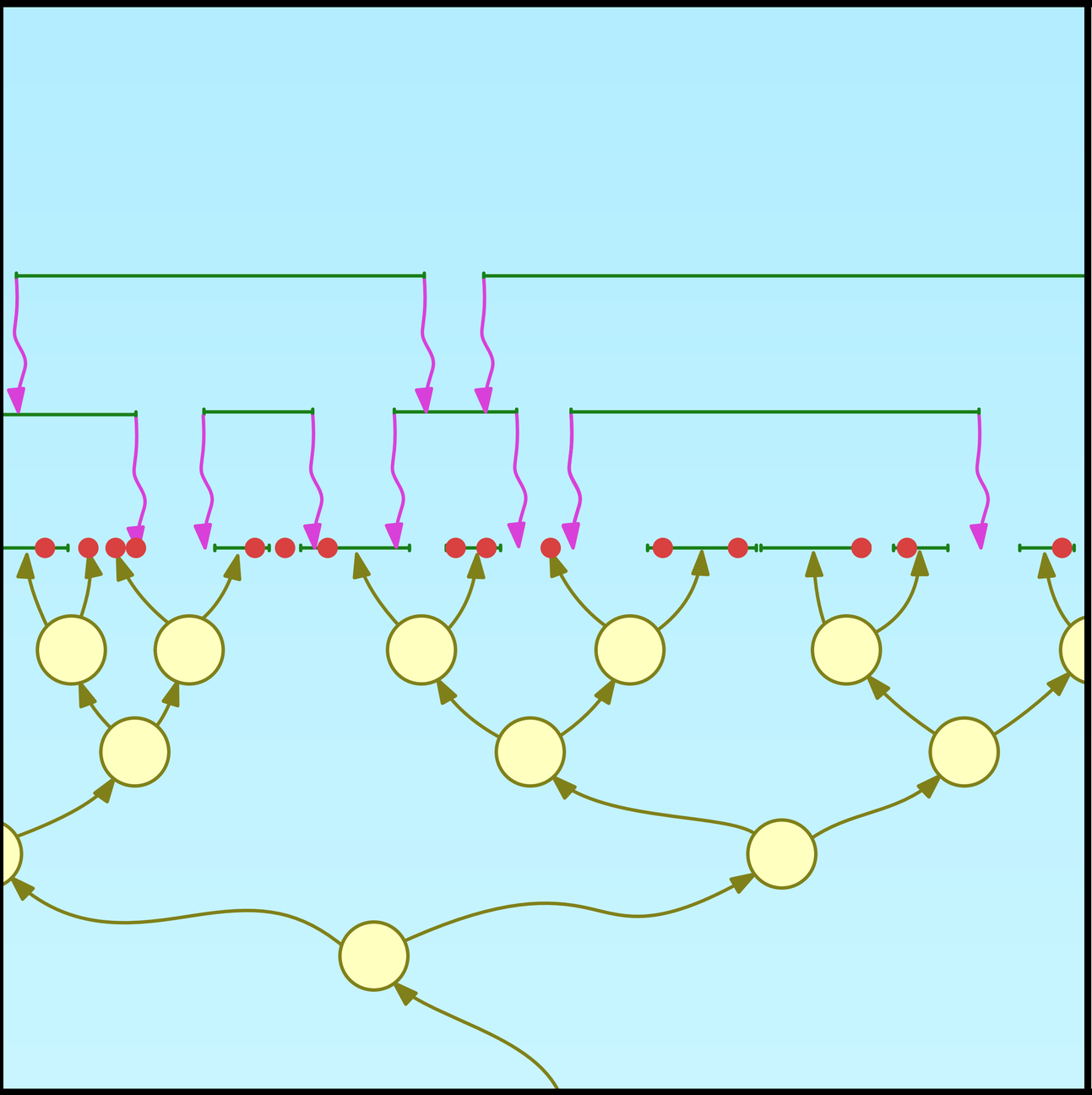


Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.

Using $F(\mathcal{R})$, we can sort $P$ in $A(\mathcal{R}) + n$ time.

[van der Hoog, Kostitsyna, Löffler & Speckmann, 2019]

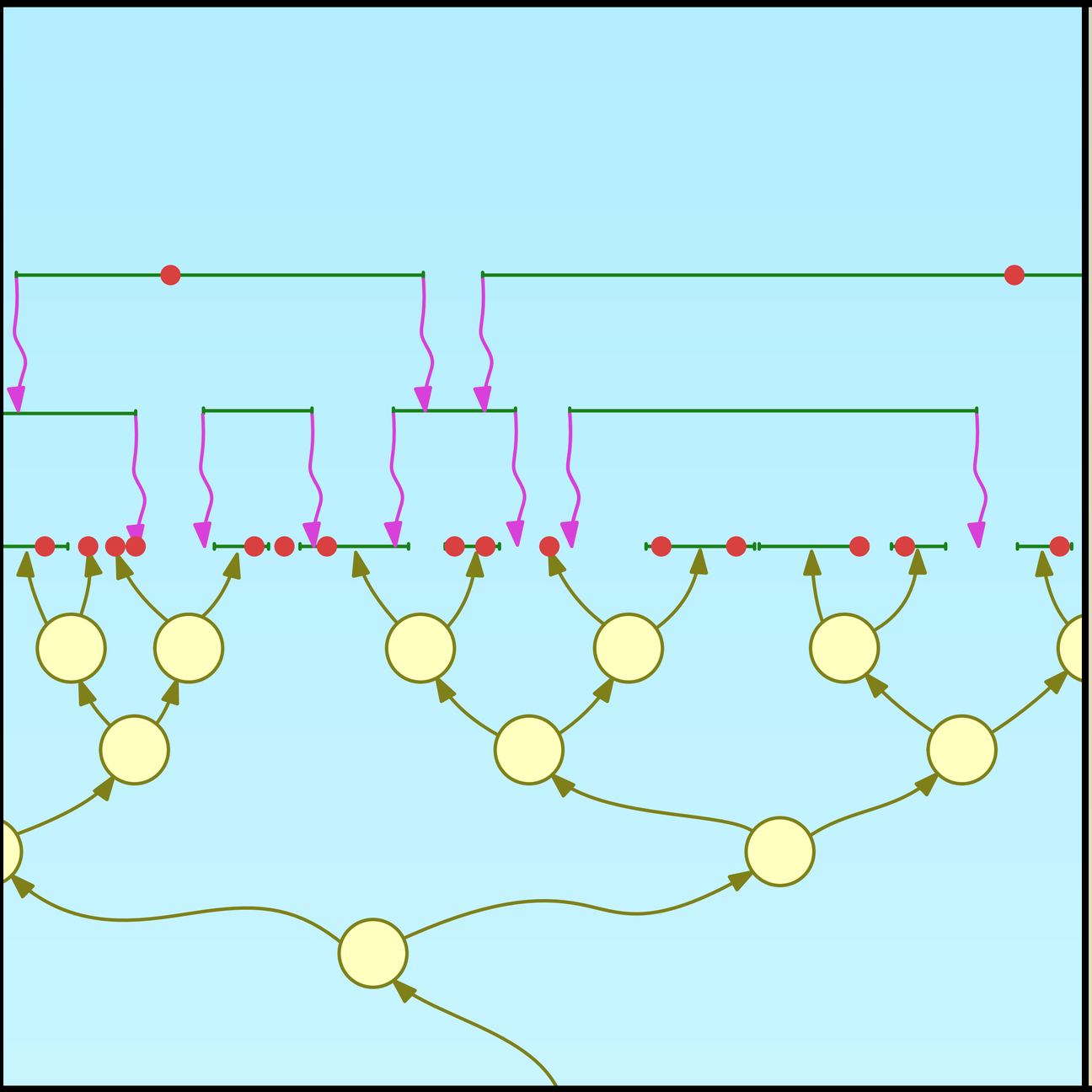


Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.

Using $F(\mathcal{R})$, we can sort $P$ in $A(\mathcal{R}) + n$ time.

[van der Hoog, Kostitsyna, Löffler & Speckmann, 2019]

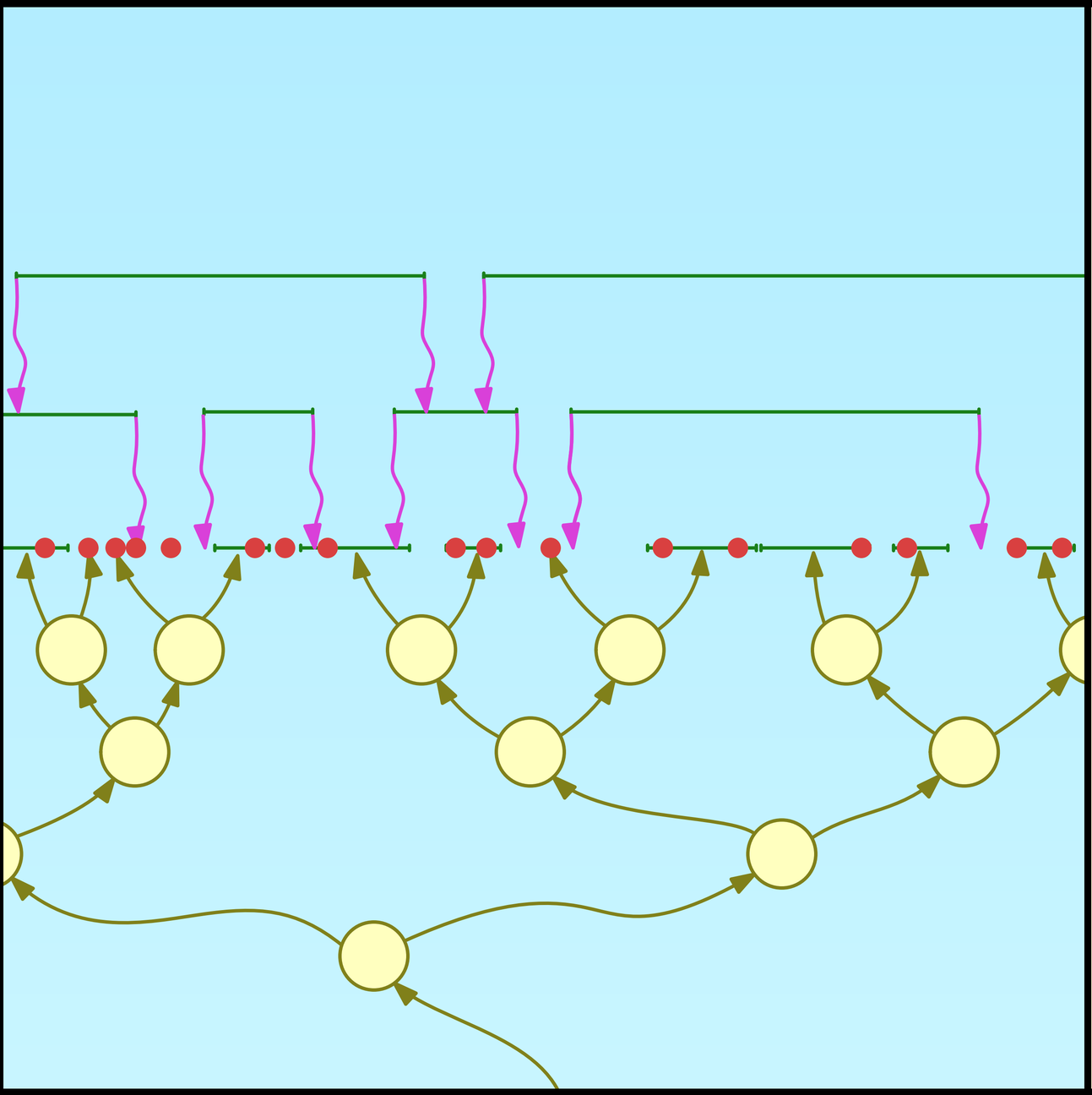


Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.

Using $F(\mathcal{R})$, we can sort $P$ in $A(\mathcal{R}) + n$ time.

[van der Hoog, Kostitsyna, Löffler & Speckmann, 2019]



Let $\mathcal{R}$ be a set of partially overlapping intervals.

There is a fantasy structure $F(\mathcal{R})$ which we can compute from $\mathcal{R}$ in $O(n \log n)$ time.

Using $F(\mathcal{R})$, we can sort $P$ in $A(\mathcal{R}) + n$ time.

[van der Hoog, Kostitsyna, Löffler & Speckmann, 2019]

# Chapter 3

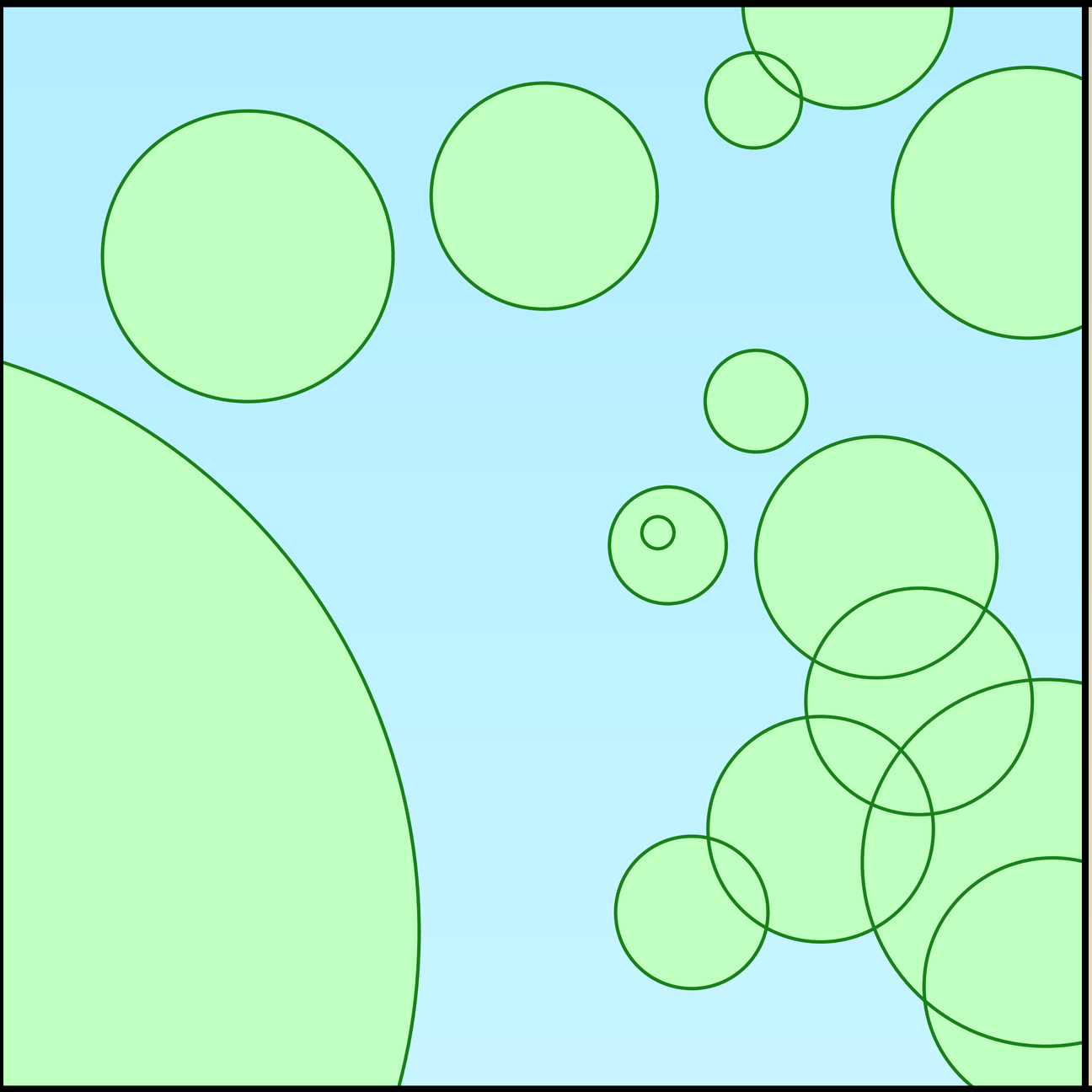
*FINE-GRAINED MEASURES*

Ambiguity of regions

Information theory

State of the art in 1D

State of the art in 2D



?

# Thank You!

## Location & Information

Maarten Löffler
*Utrecht University*

*Based on joint work with*

Kevin Buchin
Ivor van der Hoog
Irina Kostitsyna
Marc van Kreveld
Joe Mitchell
Pat Morin
Wolfgang Mulzer
Jack Snoeyink
Bettina Speckmann