CGAL

Computational Geometry Algorithms Library

Monique Teillaud

www.cgal.org

Outline

Introduction

Introduction — The CGAL Open Source Project

Goals

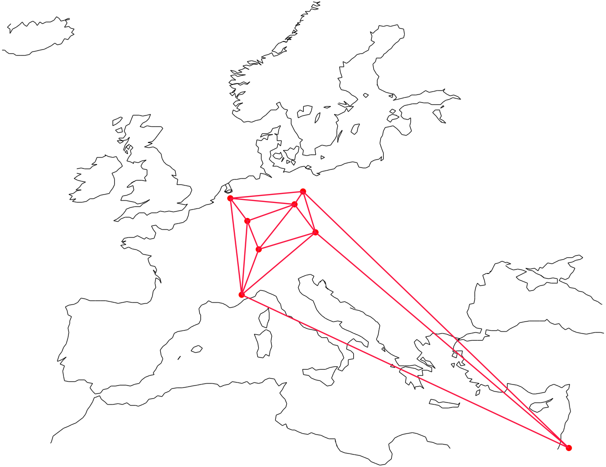
- Promote the research in Computational Geometry (CG)

- *"make the large body of geometric algorithms developed in the field of CG available for industrial applications"*

⇒ robust programs

History

- Development started in 1995



History

- Development started in 1995

- January, 2003: creation of Geometry Factory

INRIA startup

sells commercial licenses, support, customized developments

- November, 2003: Release 3.0 - Open Source Project
  - new contributors

- September, 2017: Release 4.11

License

- a few basic packages under LGPL
- most packages under GPLv3+
  - free use for Open Source code
  - commercial license needed otherwise

Distribution

- from github

- included in Linux distributions (Debian, etc)
- available through macport

- 2009: CGAL triangulations integrated in Matlab

- CGAL-bindings
  - CGAL triangulations, meshes, etc, can be used in Java or Python
  - implemented with SWIG

CGAL in numbers

- N00,000 lines of C++ code
- several platforms
  g++ (Linux MacOS Windows), clang, VC++, etc
- $>$ 1,000 downloads per month
- 50 developers registered on developer list
  ($\sim$ 20 active)

Development process

- New contributions must be submitted to the Editorial board and reviewed.
- Automatic test suites running on all supported compilers/platforms

Users

List of identified users in various fields

- Art
- Architecture, Buildings Modeling, Urban Modeling
- Astronomy
- Computational Geometry and Geometric Computing
- Computer Graphics
- Computational Topology and Shape Matching
- Computer Vision, Image Processing, Photogrammetry
- Games, Virtual Worlds
- Geographic Information Systems
- Geology and Geophysics
- Geometry Processing
- Medical Modeling and Biophysics
- Mesh Generation and Surface Reconstruction
- 2D and 3D Modelers
- Molecular Modeling
- Motion Planning
- Particle Physics, Materials, Nanostructures, Microstructures, Fluid Dynamics
- Peer-to-Peer Virtual Environment
- Sensor Networks

More non-identified users. . .

Customers of Geometry Factory

(2013)

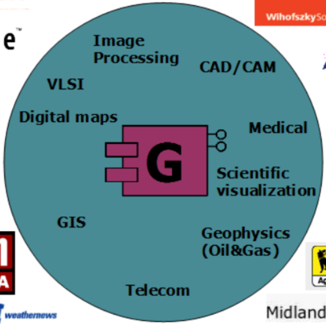


Image Processing

CAD/CAM

VLSI

Digital maps

Medical

G

Scientific visualization

GIS

Geophysics (Oil&Gas)

Telecom

Orbotech
pulsic
VDRC
OHIO STATE UNIVERSITY
NOESIS
protomold
Autodesk
Exa
TOSHIBA
ECI
cādence
Agilent Technologies
BAE SYSTEMS
QinetiQ
erdas
Virtualwind
SAFE SOFTWARE
rm DATA
ARCHIVIDEO
weathernews
ESRI
NAVTEQ
france telecom
ENVITIA
TruePosition
BT
WihofszkySoftware
ZWCAD
MPC
ECL
DASSAULT SYSTEMES
TOMTEC
schaerermayfield
GE Healthcare
ST. JUDE MEDICAL
PRESAGIS
The MathWorks
School of Biomedical Sciences
INDUSTRIAL RESEARCH LIMITED
Agip
TGS
Midland Valley
TOTAL
Schlumberger
IFP
roxar
أرامكو السعودية
Saudi Aramco
Géosciences pour une Terre durable
brgm
ExxonMobil

Introduction — Contents of CGAL

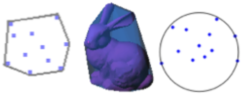
1. Introduction
   - The CGAL Open Source Project
   - Contents of CGAL
   - The CGAL Kernels
2. 2D, 3D Triangulations in CGAL
   - Introduction
   - Functionalities
   - Representation
   - Robustness
   - Software Design

Structure

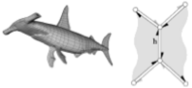
- Kernels
- Various packages
- Support Library

  STL extensions, I/O, generators, timers. . .

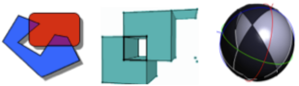
Some packages



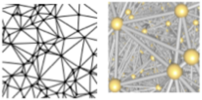
Bounding Volumes



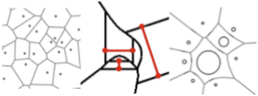
Polyhedral Surface



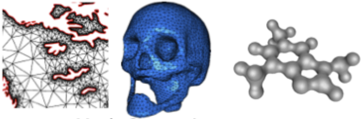
BooleanOperations



Triangulations

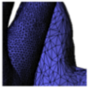


Voronoi Diagrams

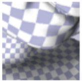


Mesh Generation

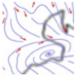
Subdivision



Simplification

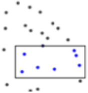


Parameterization



Streamlines

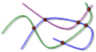
Ridge
Detection



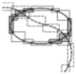
Neighbour
Search

Kinetic
Data structures

Lower Envelope

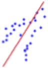


Arrangement



Intersection
Detection

Minkowski
Sum



PCA

Polytope
distance

QP Solver

Introduction — The CGAL Kernels

- 2D, 3D, dD “rational” kernels
- 2D circular and 3D spherical kernels

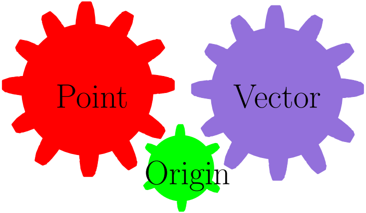
In the kernels

- Elementary geometric objects
- Elementary computations on them

Primitives 2D, 3D, dD	Predicates	Constructions
• Point	• comparison	• intersection
• Vector	• Orientation	• squared distance
• Triangle	• InSphere	. . .
• Circle	. . .	
. . .		

Affine geometry

Point - Origin → Vector
Point - Point → Vector
Point + Vector → Point



Point + Point illegal

midpoint(a,b) = a + 1/2 x (b-a)

Kernels and number types

Cartesian representation

$$\text{Point}\left|\begin{array}{l} x = \frac{hx}{hw} \\ y = \frac{hy}{hw} \end{array}\right.$$

Homogeneous representation

$$\text{Point}\left|\begin{array}{l} hx \\ hy \\ hw \end{array}\right.$$

Kernels and number types

Cartesian representation

$$\text{Point}\left|\begin{array}{l} x = \frac{hx}{hw} \\ y = \frac{hy}{hw} \end{array}\right.$$

Homogeneous representation

$$\text{Point}\left|\begin{array}{l} hx \\ hy \\ hw \end{array}\right.$$

- ex: Intersection of two lines -

$$\begin{cases} a_1 x + b_1 y + c_1 = 0 \\ a_2 x + b_2 y + c_2 = 0 \end{cases}$$

$$\begin{cases} a_1 hx + b_1 hy + c_1 hw = 0 \\ a_2 hx + b_2 hy + c_2 hw = 0 \end{cases}$$

$$(x, y) = \left(\frac{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \right)$$

$$(hx, hy, hw) = \left(\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}, -\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}, \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \right)$$

Kernels and number types

Cartesian representation

$$\text{Point} \left| \begin{array}{l} x = \frac{hx}{hw} \\ y = \frac{hy}{hw} \end{array} \right.$$

Homogeneous representation

$$\text{Point} \left| \begin{array}{l} hx \\ hy \\ hw \end{array} \right.$$

- ex: Intersection of two lines -

$$\begin{cases} a_1 x + b_1 y + c_1 = 0 \\ a_2 x + b_2 y + c_2 = 0 \end{cases}$$

$$\begin{cases} a_1 hx + b_1 hy + c_1 hw = 0 \\ a_2 hx + b_2 hy + c_2 hw = 0 \end{cases}$$

$$(x, y) = \left(\frac{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \right)$$

$$(hx, hy, hw) = \left(\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}, -\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}, \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \right)$$

Field operations

Ring operations

The “rational” Kernels

```
CGAL::Cartesian< FieldType >
CGAL::Homogeneous< RingType >
```

⟶ Flexibility

```
typedef double                        NumberType;
typedef Cartesian< NumberType >       Kernel;
typedef Kernel::Point_2               Point;
```

Arithmetic robustness issues

Rational Kernels:
Predicates = signs of polynomial expressions

Exact Geometric Computation
$\neq$ **exact arithmetics**

Predicates evaluated exactly

Filtering Techniques (interval arithmetics, etc)
exact arithmetics only when needed

`CGAL::Exact_predicates_inexact_constructions_kernel`

Arithmetic robustness issues

```
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt( NT(2) );

Kernel::Point_2 p(0,0), q(sqrt2,sqrt2);
Kernel::Circle_2 C(p,2); // squared radius 2
```

Arithmetic robustness issues

```
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt( NT(2) );

Kernel::Point_2 p(0,0), q(sqrt2,sqrt2);
Kernel::Circle_2 C(p,2); // squared radius 2

assert( C.has_on_boundary(q) );
```

OK if NT gives exact sqrt
assertion violation otherwise

The circular/spherical kernels

Circular/spherical kernels

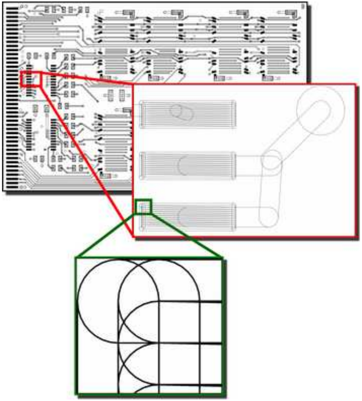
- solve needs for e.g. intersection of circles.
- extend the CGAL (linear) kernels

Exact computations on algebraic numbers of degree 2
= roots of polynomials of degree 2

Algebraic methods reduce comparisons to
computations of signs of polynomial expressions

Application of the 2D circular kernel

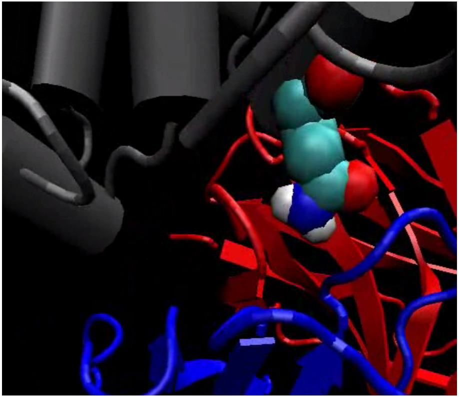
Computation of arrangements
of 2D circular arcs and line segments



Pedro M.M. de Castro, Master internship

Application of the 3D spherical kernel

Computation of arrangements of 3D spheres



Sébastien Loriot, PhD thesis

2D, 3D Triangulations in CGAL

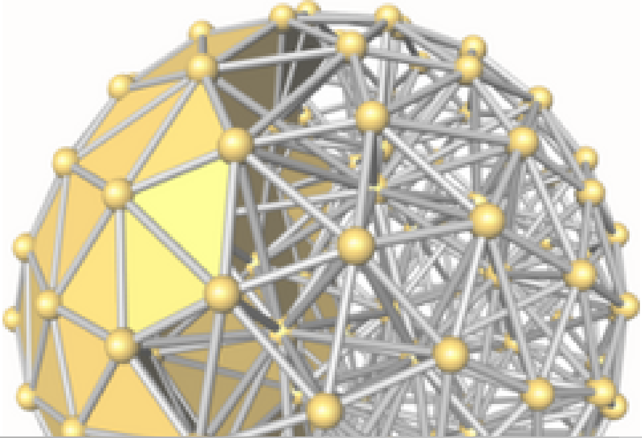
2D, 3D Triangulations in CGAL — Introduction

1. Introduction
   - The CGAL Open Source Project
   - Contents of CGAL
   - The CGAL Kernels
2. 2D, 3D Triangulations in CGAL
   - Introduction
   - Functionalities
   - Representation
   - Robustness
   - Software Design

Simplicial complex

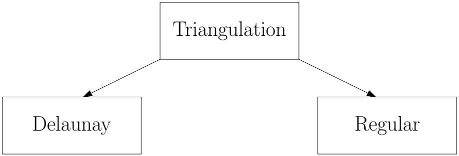
= set $\mathbb{K}$ of 0,1,2,... d-faces such that

- each face is a simplex
- $\sigma \in \mathbb{K}, \tau \leq \sigma \Rightarrow \tau \in \mathbb{K}$
- $\sigma, \sigma' \in \mathbb{K} \Rightarrow \sigma \cap \sigma' \leq \sigma, \sigma'$



Various triangulations

2D, 3D, *d*D Basic triangulations
2D, 3D, *d*D Delaunay triangulations
2D, 3D, *d*D Regular triangulations



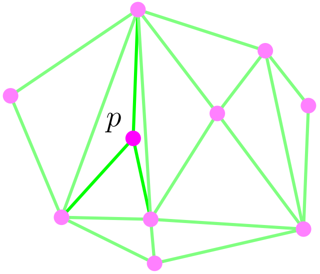
Triangulation

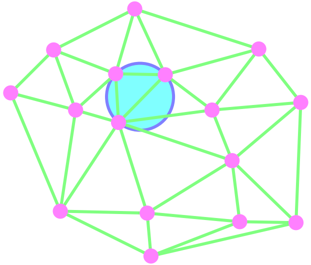
Delaunay

Regular

Basic and Delaunay triangulations

(figures in 2D)





Basic triangulations : incremental construction

Delaunay triangulations: empty sphere property

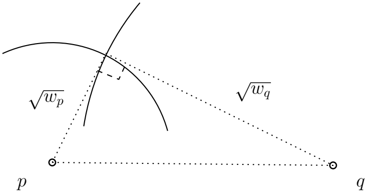
Regular triangulations

weighted point $p^{(w)} = (p, w_p), p \in \mathbb{R}^3, w_p \in \mathbb{R}$
$p^{(w)} = (p, w_p) \simeq$ sphere of center p and radius $\sqrt{w_p}$.
power product between $p^{(w)}$ and $z^{(w)}$

$$\Pi(p^{(w)}, z^{(w)}) = \|p - z\|^2 - w_p - w_z$$

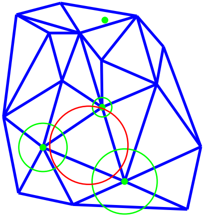
$p^{(w)}$ and $z^{(w)}$ orthogonal iff $\Pi(p^{(w)}, z^{(w)}) = 0$

(2D)



Regular triangulations

Power sphere of 4 weighted points in $\mathbb{R}^3$ =
unique common orthogonal weighted point.
$z^{(w)}$ is regular iff $\forall p^{(w)}, \Pi(p^{(w)}, z^{(w)}) \geq 0$



(2D)

Regular triangulations: generalization of Delaunay triangulations to weighted points. Dual of the power diagram.

The power sphere of all simplices is regular.

2D, 3D Triangulations in CGAL — Functionalities

General functionalities

- Traversal of a 2D (3D) triangulation

- passing from a face (cell) to its neighbors
- iterators to visit all faces (cells) of a triangulation
- circulators (iterators) to visit all faces (cells)
  incident to a vertex
- circulators to visit all cells around an edge

General functionalities

- Traversal of a 2D (3D) triangulation

- passing from a face (cell) to its neighbors
- iterators to visit all faces (cells) of a triangulation
- circulators (iterators) to visit all faces (cells)
  incident to a vertex
- circulators to visit all cells around an edge

- Point location query

- Insertion, removal, flips

General functionalities

- Traversal of a 2D (3D) triangulation

- passing from a face (cell) to its neighbors
- iterators to visit all faces (cells) of a triangulation
- circulators (iterators) to visit all faces (cells)
  incident to a vertex
- circulators to visit all cells around an edge

- Point location query

- Insertion, removal, flips

- is_valid
  checks local validity (sufficient in practice)
  not global validity

Traversal of a 3D triangulation

Iterators

All_cells_iterator	Finite_cells_iterator
All_faces_iterator	Finite_faces_iterator
All_edges_iterator	Finite_edges_iterator
All_vertices_iterator	Finite_vertices_iterator

Circulators

Cell_circulator : cells incident to an edge

Facet_circulator : facets incident to an edge

```
All_vertices_iterator vit;
for (vit = T.all_vertices_begin();
     vit != T.all_vertices_end(); ++vit)
          ...
```

Traversal of a 3D triangulation

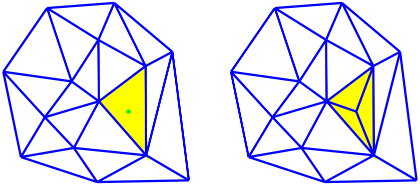
Around a vertex

incident cells and facets, adjacent vertices

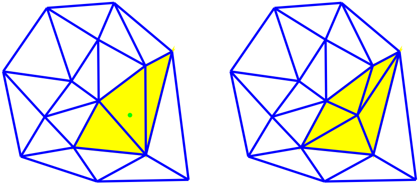
```
template < class OutputIterator >
OutputIterator
     t.incident_cells
          ( Vertex_handle v, OutputIterator cells)
```

Point location, insertion, removal

basic triangulation:

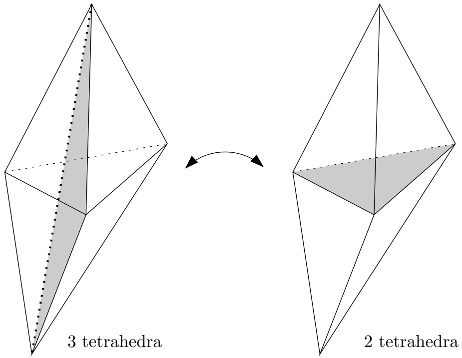


Delaunay triangulation :



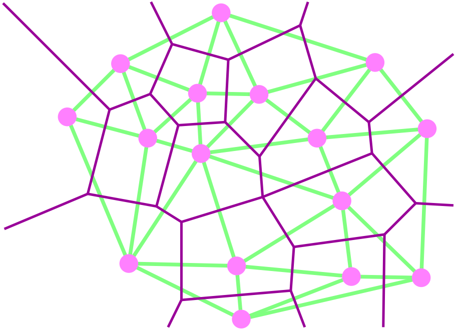
3D Flip

if convex position



Additional functionalities for Delaunay triangulations

Nearest neighbor queries
Voronoi diagram



2D, 3D Triangulations in CGAL — Representation

1. Introduction
   - The CGAL Open Source Project
   - Contents of CGAL
   - The CGAL Kernels
2. 2D, 3D Triangulations in CGAL
   - Introduction
   - Functionalities
   - Representation
   - Robustness
   - Software Design

The main algorithm

Incremental algorithm

- fully dynamic (point insertion, vertex removal)
- any dimension
- easier to implement
- efficient in practice
- . . .

Needs

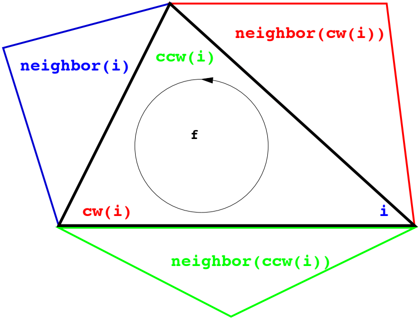
Walking in a triangulation

Access to

- vertices of a simplex
- neighbors of a simplex

in constant time

2D - Representation based on faces



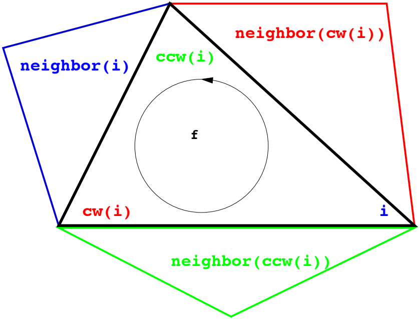
Vertex

Face_handle *v_face*

Face

Vertex_handle *vertex*[3]
Face_handle *neighbor*[3]

2D - Representation based on faces



Vertex

Face_handle *v_face*

Face

Vertex_handle *vertex*[3]
Face_handle *neighbor*[3]

Edges are implicit: `std::pair<` f, i `>`
where $f =$ one of the two incident faces.

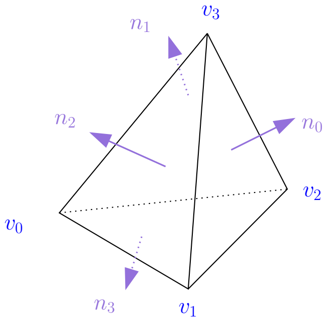
more efficient than half-edges

From one face to another

$\texttt{n = f} \rightarrow \texttt{neighbor(i)}$

$\texttt{j = n} \rightarrow \texttt{index(f)}$

3D - Representation based on cells



Vertex

Cell_handle *v_cell*

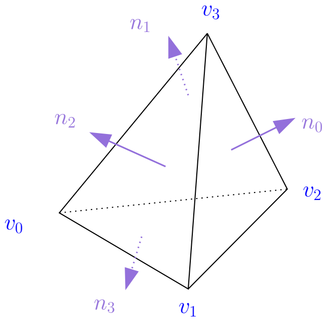
Cell

Vertex_handle *vertex*[4]
Cell_handle *neighbor*[4]

Faces are implicit: `std::pair<` c, i `>`
where c = one of the two incident cells.

Edges are implicit: `std::pair<` u, v `>`
where u, v = vertices.

3D - Representation based on cells



Vertex

Cell_handle	*v_cell*

Cell

Vertex_handle	*vertex*[4]
Cell_handle	*neighbor*[4]

From one cell to another

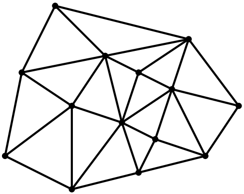
```
n = c → neighbor(i)
j = n → index(c)
```

The infinite region

Triangulation of a set of points =
partition of the convex hull into simplices.

The infinite region has
non-constant size

Add a bounding box?



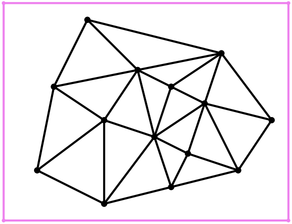
The infinite region

Triangulation of a set of points =
partition of the convex hull into simplices.

The infinite region has
non-constant size

Add a bounding box?

- requires to know points in advance



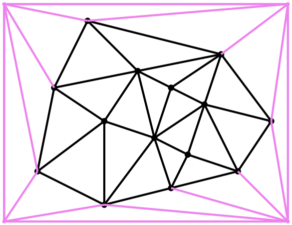
The infinite region

Triangulation of a set of points =
partition of the convex hull into simplices.

The infinite region has
non-constant size

Add a bounding box?

- requires to know points in advance
- creates ugly simplices

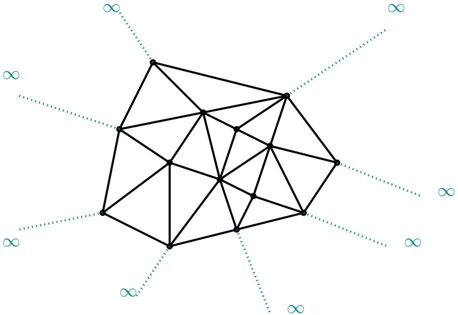


The infinite region

Triangulation of a set of points =
partition of the convex hull into simplices.

Add an infinite vertex
⟶ "triangulation"
of the infinite region

- Every cell is a "simplex".
- Any facet is incident to two cells.

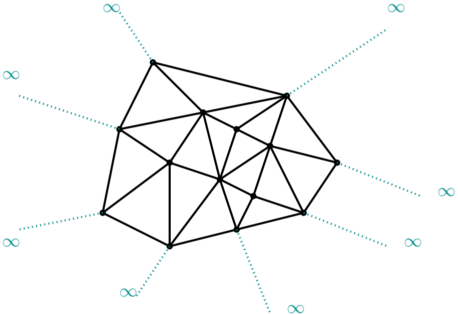


The infinite region

Triangulation of a set of points =
partition of the convex hull into simplices.

Add an infinite vertex
$\longrightarrow$ "triangulation"
of the infinite region

- Every cell is a "simplex".
- Any facet is incident to two cells.



$$\text{Triangulation of } \mathbb{R}^d$$
$$\simeq$$
$$\text{Triangulation of the topological sphere } \mathbb{S}^d.$$

Geometry vs. combinatorics

Each finite vertex stores a point

Geometry vs. combinatorics

Each finite vertex stores a point

There is NO point in the infinite vertex

infinite simplex = half-space

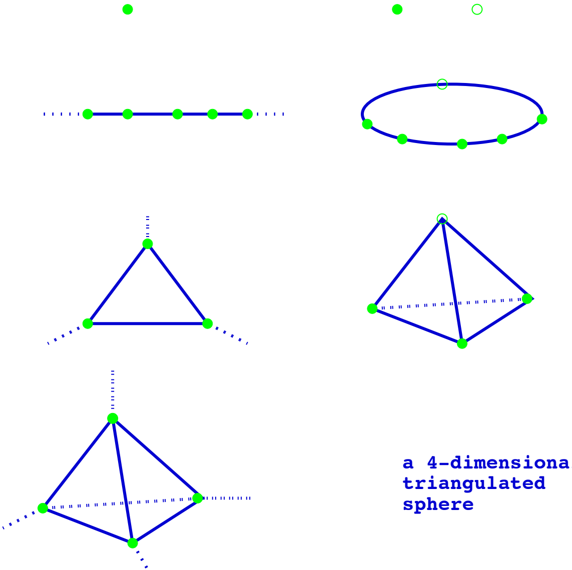
Dimensions in a 3D triangulation

dim 0

dim 1

dim 2

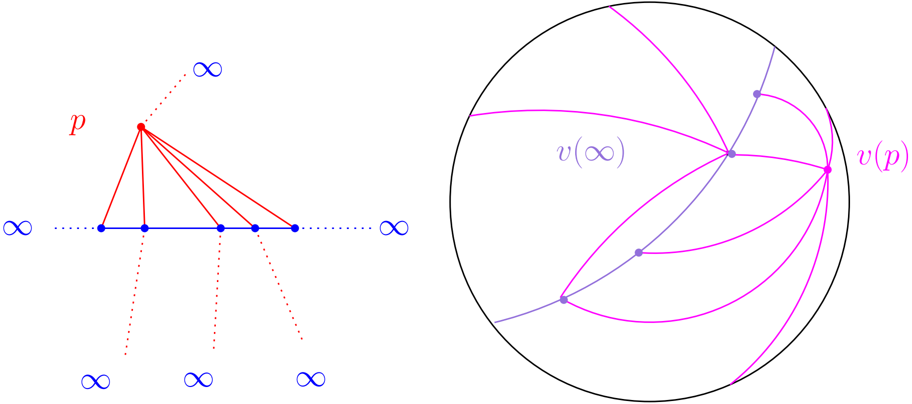
dim 3



a 4-dimensiona
triangulated
sphere

Dimensions

Adding a point outside the current affine hull:
From $d = 1$ to $d = 2$



2D, 3D Triangulations in CGAL — Robustness

Arithmetic robustness

see above

Benchmarks
2.3 GHz, 16 GByte workstation
CGAL 3.9 (Release mode)

Arithmetic robustness

see above

Benchmarks
2.3 GHz, 16 GByte workstation
CGAL 3.9 (Release mode)

Delaunay triangulation - 10 Mpoints

Kernel	2D	3D
`Cartesian < double >`	9.7 sec	75 sec
`Exact_predicates_inexact_constructions_kernel`	10.6 sec	82 sec

Arithmetic robustness

see above

Benchmarks
2.3 GHz, 16 GByte workstation
CGAL 3.9 (Release mode)

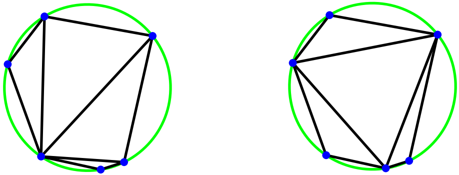
Delaunay triangulation - 10 Mpoints

Kernel	2D	3D
`Cartesian < double >` may loop (or crash) !		
`Exact_predicates_inexact_constructions_kernel`	10.6 sec	82 sec

Degenerate cases

Cospherical points

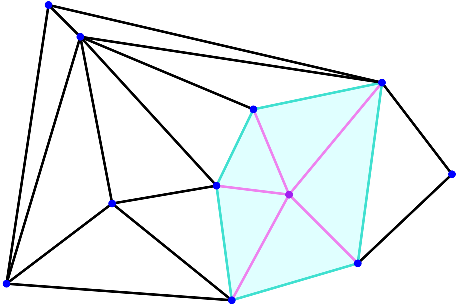
Any triangulation is a Delaunay triangulation



Degenerate cases

Vertex removal

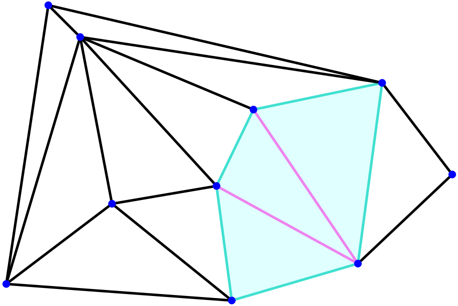
1- remove the tetrahedra incident to $v \longrightarrow$ hole



Degenerate cases

Vertex removal

1- remove the tetrahedra incident to $v \longrightarrow$ hole
2- retriangulate the hole

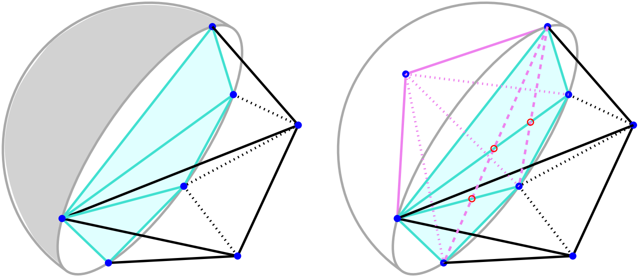


Degenerate cases

Vertex removal

Cocircular points

Several possible Delaunay triangulations of a facet of the hole



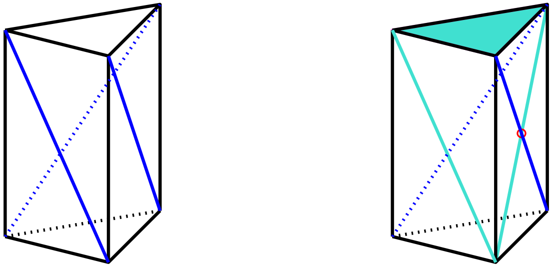
Triangulation of the hole must be
compatible with the rest of the triangulation

Degenerate cases

Remark on the general question:

H given polyhedron with triangulated facets.
Find a Delaunay triangulation of H keeping its facets ?

Not always possible:



Degenerate cases

Allowing flat tetrahedra?

k cocircular points on a facet

2D triangulation of the facet induced by tetrahedra in the hole

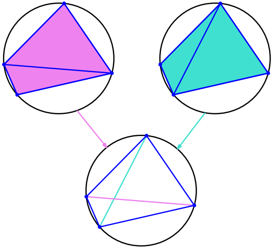
$\vdots$

sequence of $O(k^2)$ edge flips

$\vdots$

2D triangulation of the facet induced by tetrahedra outside the hole

edge flip $\longleftrightarrow$ flat tetrahedron



Degenerate cases

Allowing flat tetrahedra?

k cocircular points on a facet

2D triangulation of the facet induced by tetrahedra in the hole

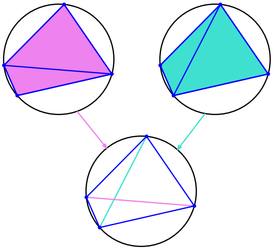
$\vdots$

sequence of $O(k^2)$ edge flips

$\vdots$

2D triangulation of the facet induced by tetrahedra outside the hole

edge flip $\longleftrightarrow$ flat tetrahedron



Unacceptable

Degenerate cases

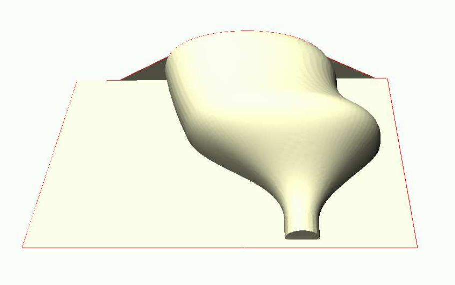
Symbolic perturbation of `in_sphere` predicate

see course robustness

- Algorithm working even in degenerate situations
- No flat tetrahedra
- Perturbed predicate easy to code

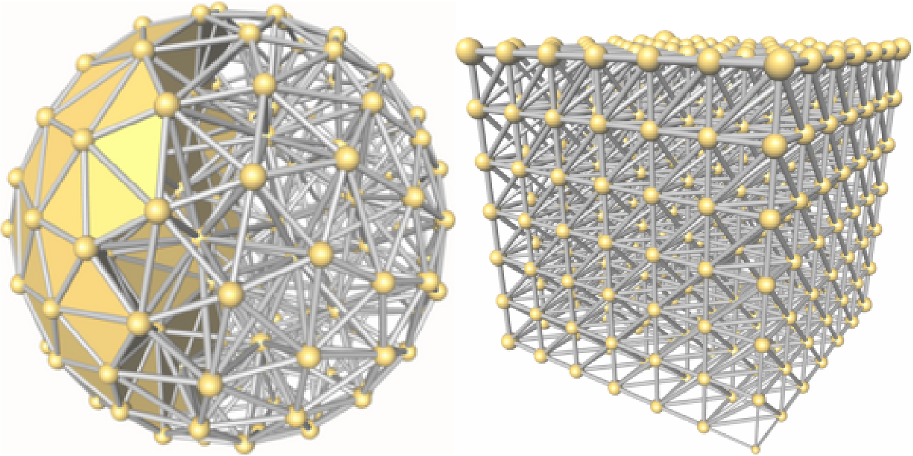
CGAL : only publicly available software
proposing a fully dynamic 3D Delaunay/regular triangulation.

Robustness



Dassault Systèmes

Robustness



Pictures by Pierre Alliez

2D, 3D Triangulations in CGAL — Software Design

1. Introduction
   - The CGAL Open Source Project
   - Contents of CGAL
   - The CGAL Kernels
2. 2D, 3D Triangulations in CGAL
   - Introduction
   - Functionalities
   - Representation
   - Robustness
   - Software Design

Traits class

Triangulation_2<Traits, TDS>

Geometric traits classes provide:

Geometric objects + predicates + constructors

Flexibility:

- The Kernel can be used as a traits class for several algorithms
- Otherwise: Default traits classes provided
- The user can plug his/her own traits class

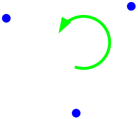
Traits class

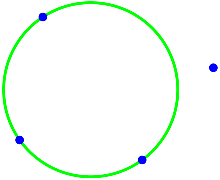
Generic algorithms

`Delaunay_Triangulation_2<Traits, TDS>`

Traits parameter provides:

- Point
- orientation test, in_circle test



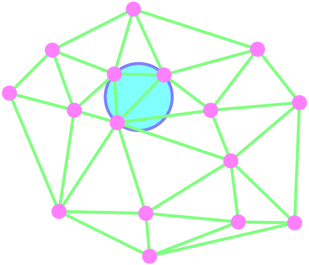


Traits class

2D Kernel used as traits class

```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2< K > Delaunay;
```

- 2D points: coordinates (x, y)
- orientation, in_circle

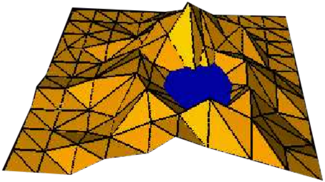


Traits class

Changing the traits class

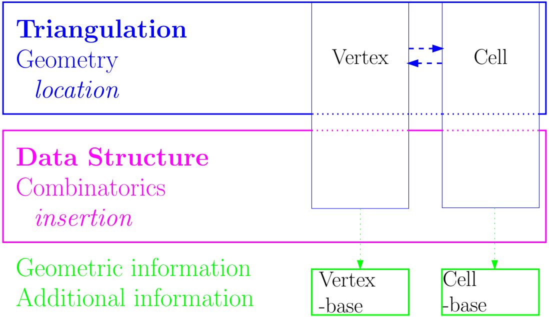
```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Projection_traits_xy_3< K > Traits;
typedef CGAL::Delaunay_triangulation_2< Traits > Terrain;
```

- 3D points: coordinates (x, y, z)
- orientation, in_circle:
  on x and y coordinates only



Layers

Triangulation_3< Traits, TDS >



Triangulation_data_structure_3< Vb, Cb> ;
Vb and Cb have default values.

Layers

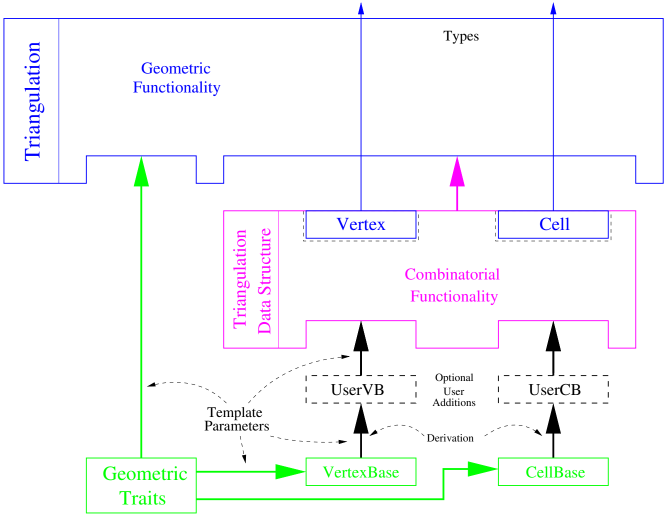
The base level

Concepts VertexBase and CellBase.

Provide
- Point + access function + setting
- incidence and adjacency relations (access and setting)

Several models, parameterised by the traits class.

Changing the Vertex_base and the Cell_base



Changing the Vertex_base and the Cell_base

First option: Triangulation_vertex_base_with_info_3

When the additional information does not depend on the TDS

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/Triangulation_vertex_base_with_info_3.h>
#include <CGAL/IO/Color.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;

typedef CGAL::Triangulation_vertex_base_with_info_3
                                       <CGAL::Color,K> Vb;

typedef CGAL::Triangulation_data_structure_3<Vb>   Tds;
typedef CGAL::Delaunay_triangulation_3<K, Tds>     Delaunay;

typedef Delaunay::Point   Point;
```

Changing the Vertex_base and the Cell_base

First option: Triangulation_vertex_base_with_info_3

When the additional information does not depend on the TDS

```
int main()
{
  Delaunay T;
  T.insert(Point(0,0,0));  T.insert(Point(1,0,0));
  T.insert(Point(0,1,0));  T.insert(Point(0,0,1));
  T.insert(Point(2,2,2));  T.insert(Point(-1,0,1));

     // Set the color of finite vertices of degree 6 to red.
  Delaunay::Finite_vertices_iterator vit;
  for (vit = T.finite_vertices_begin();
                  vit != T.finite_vertices_end(); ++vit)
     if (T.degree(vit) == 6)
         vit->info() = CGAL::RED;

  return 0;
}
```

Changing the Vertex_base and the Cell_base

Third option: write new models of the concepts

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

- Vertex and cell base classes:
  initially given a dummy TDS template parameter:

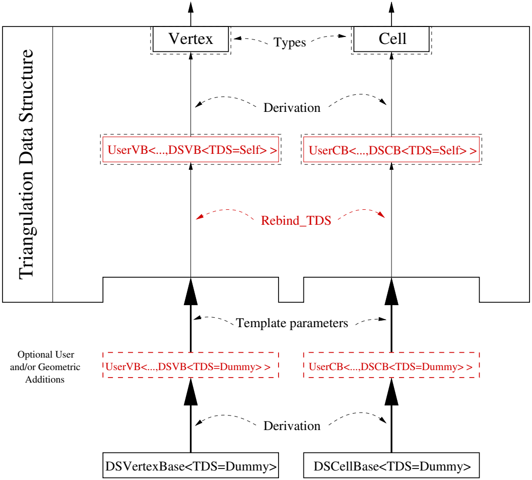
dummy TDS provides the types that can be used
by the vertex and cell base classes (such as handles).

- inside the TDS itself,
  vertex and cell base classes are
  rebound to the real TDS type

→ the same vertex and cell base classes are now
parameterized with the real TDS instead of the dummy one.

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism



Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

```
template< class GT, class Vb= Triangulation_vertex_base<GT> >
class My_vertex : public Vb
{
 typedef typename Vb::Point           Point;
 typedef typename Vb::Cell_handle     Cell_handle;

 template < class TDS2 >
 struct Rebind_TDS {
  typedef typename Vb::template Rebind_TDS<TDS2>::Other Vb2;
  typedef My_vertex<GT, Vb2>                             Other;
 };

 My_vertex() {}
 My_vertex(const Point&p)                : Vb(p) {}
 My_vertex(const Point&p, Cell_handle c) : Vb(p, c) {}
...
}
```

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

Example

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/Triangulation_vertex_base_3.h>
```

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

Example

```
template < class GT, class Vb=CGAL::Triangulation_vertex_base_3<GT> >
class My_vertex_base   : public Vb     {
 typedef typename Vb::Vertex_handle  Vertex_handle;
 typedef typename Vb::Cell_handle    Cell_handle;
 typedef typename Vb::Point          Point;

 template < class TDS2 > struct Rebind_TDS {
  typedef typename Vb::template Rebind_TDS<TDS2>::Other Vb2;
  typedef My_vertex_base<GT, Vb2>                       Other;  };

 My_vertex_base() {}
 My_vertex_base(const Point& p) : Vb(p) {}
 My_vertex_base(const Point& p, Cell_handle c) : Vb(p, c) {}

 Vertex_handle   vh;
 Cell_handle     ch;
};
```

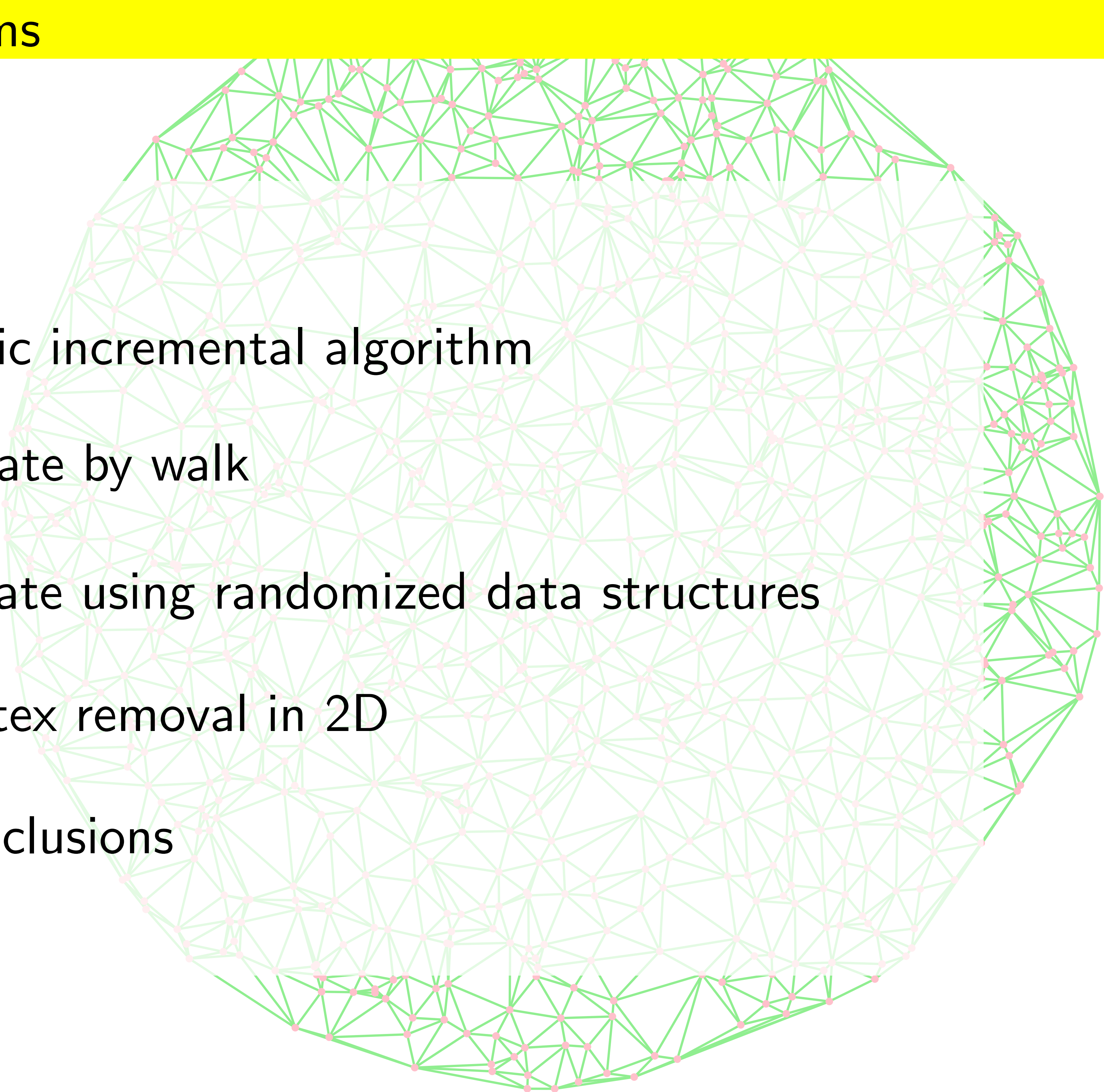
Changing the Vertex_base and the Cell_base

Second option: the "rebind" mechanism

Example

```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::
      Triangulation_data_structure_3< My_vertex_base<K> > Tds;
typedef CGAL::
      Delaunay_triangulation_3< K, Tds >             Delaunay;
typedef Delaunay::Vertex_handle    Vertex_handle;
typedef Delaunay::Point            Point;

int main()
{ Delaunay T;
  Vertex_handle v0 = T.insert(Point(0,0,0));
  ... v1; v2; v3; v4; v5;
   // Now we can link the vertices as we like.
  v0->vh = v1;  v1->vh = v2;
  v2->vh = v3;  v3->vh = v4;
  v4->vh = v5;  v5->vh = v0;
  return 0;
}
```

Algorithms



Basic incremental algorithm

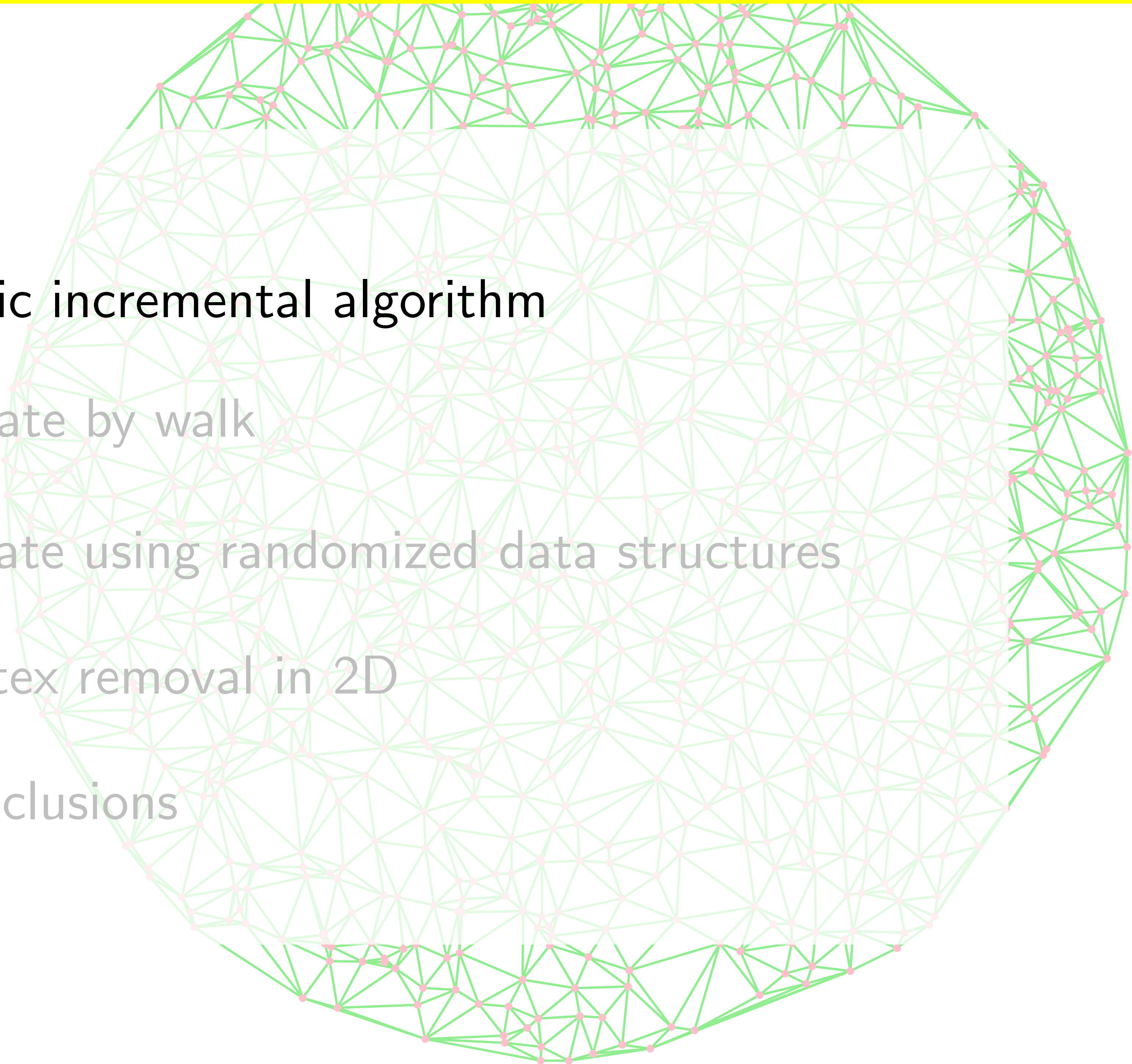
Locate by walk

Locate using randomized data structures

Vertex removal in 2D

Conclusions

Algorithms



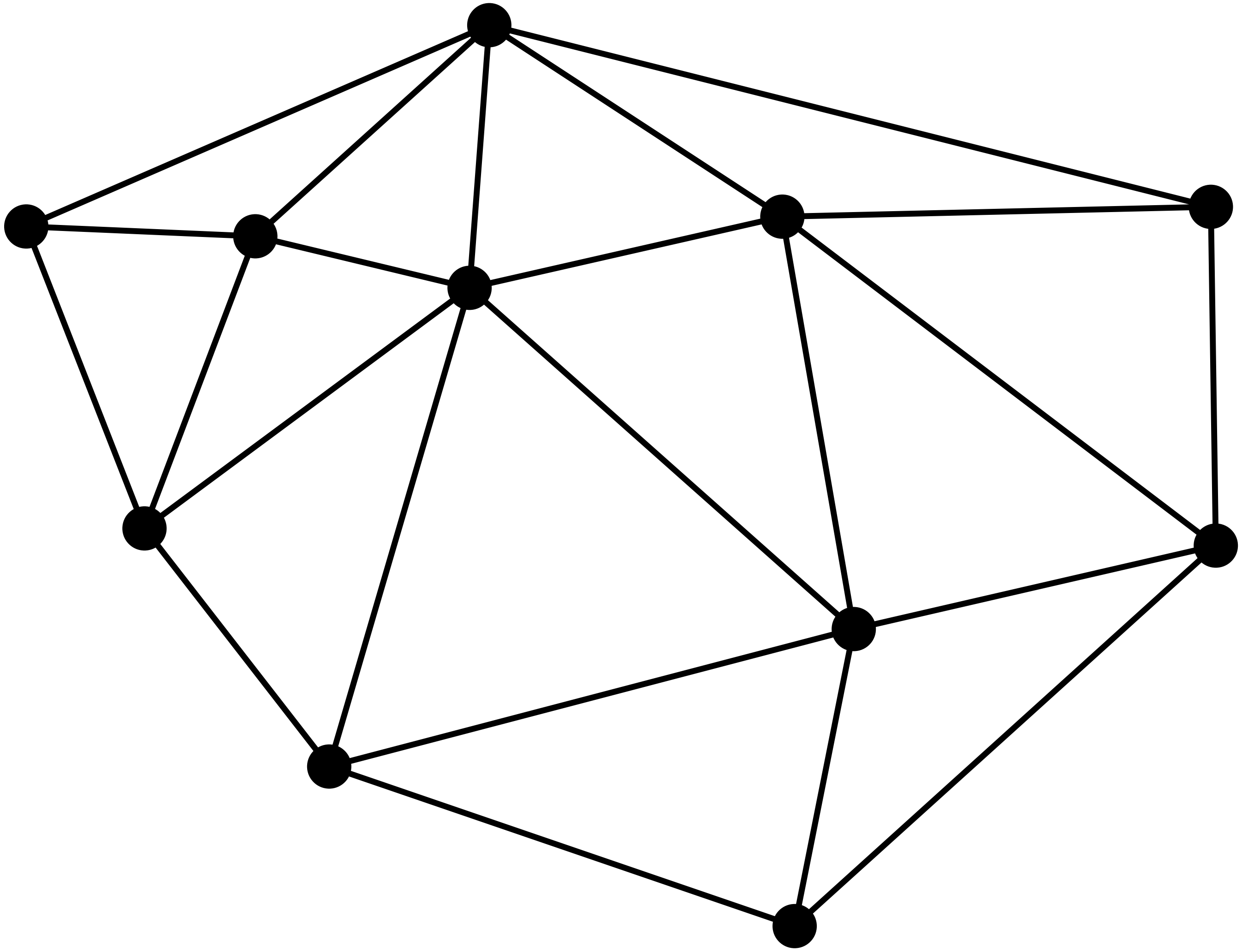
Basic incremental algorithm

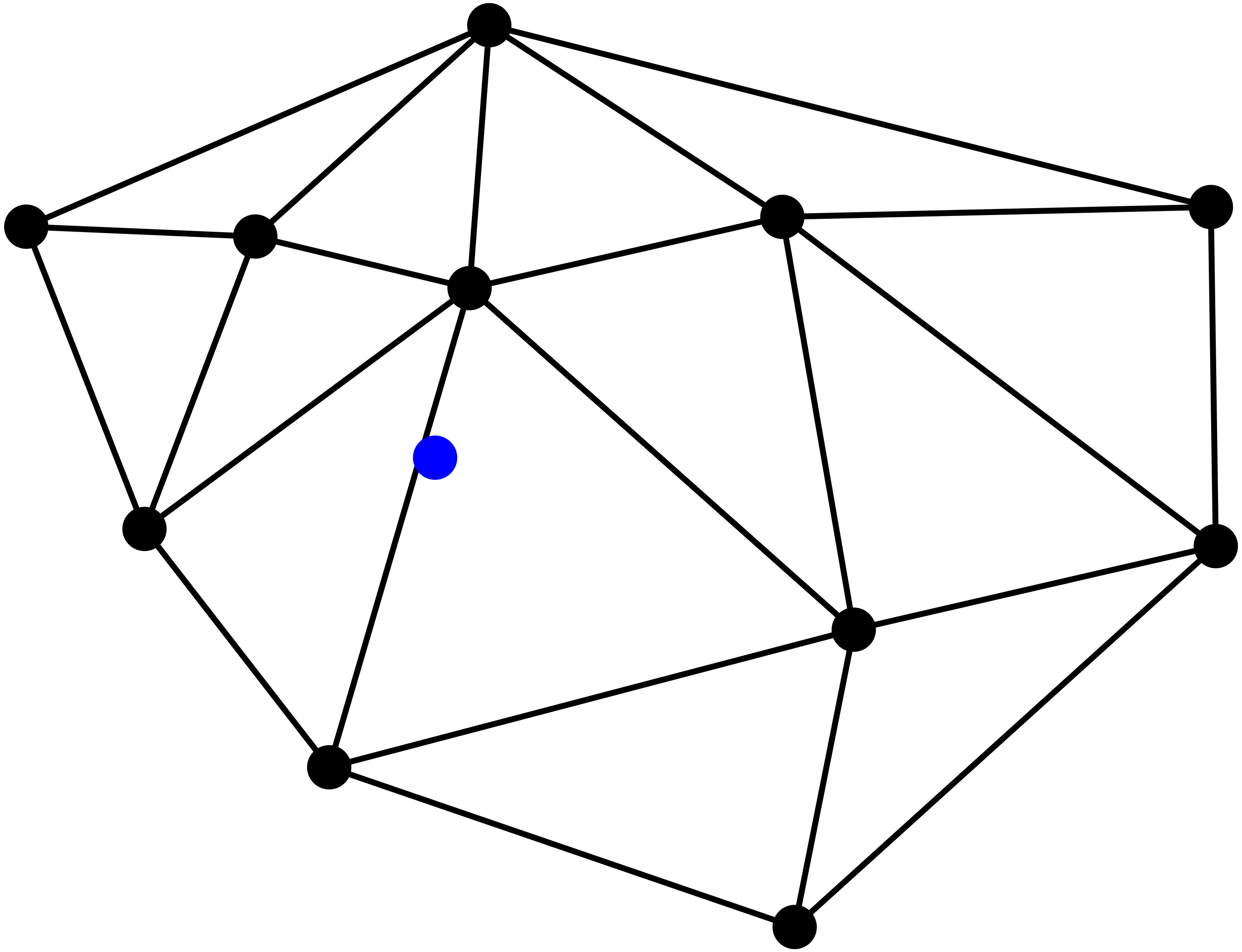
Locate by walk

Locate using randomized data structures

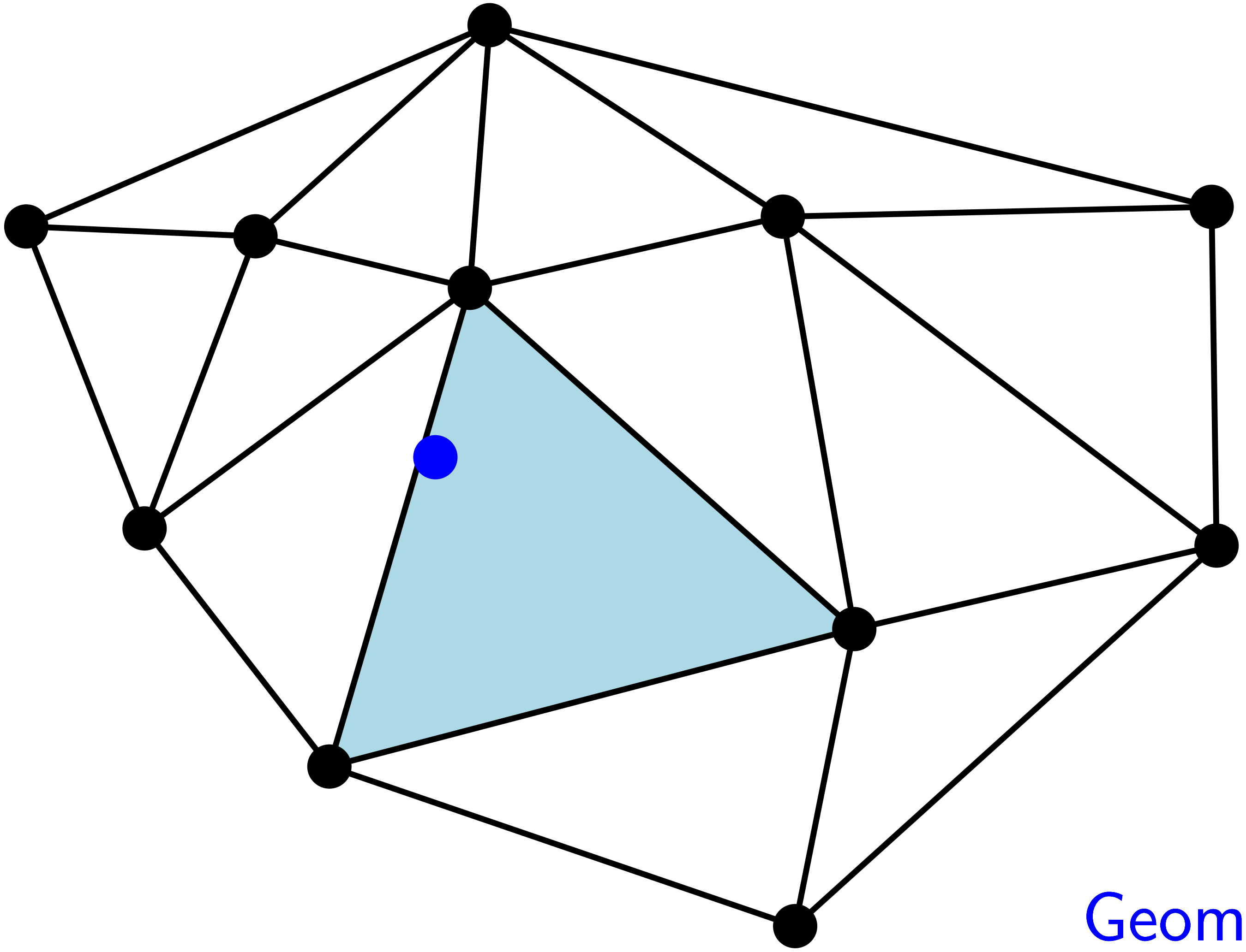
Vertex removal in 2D

Conclusions



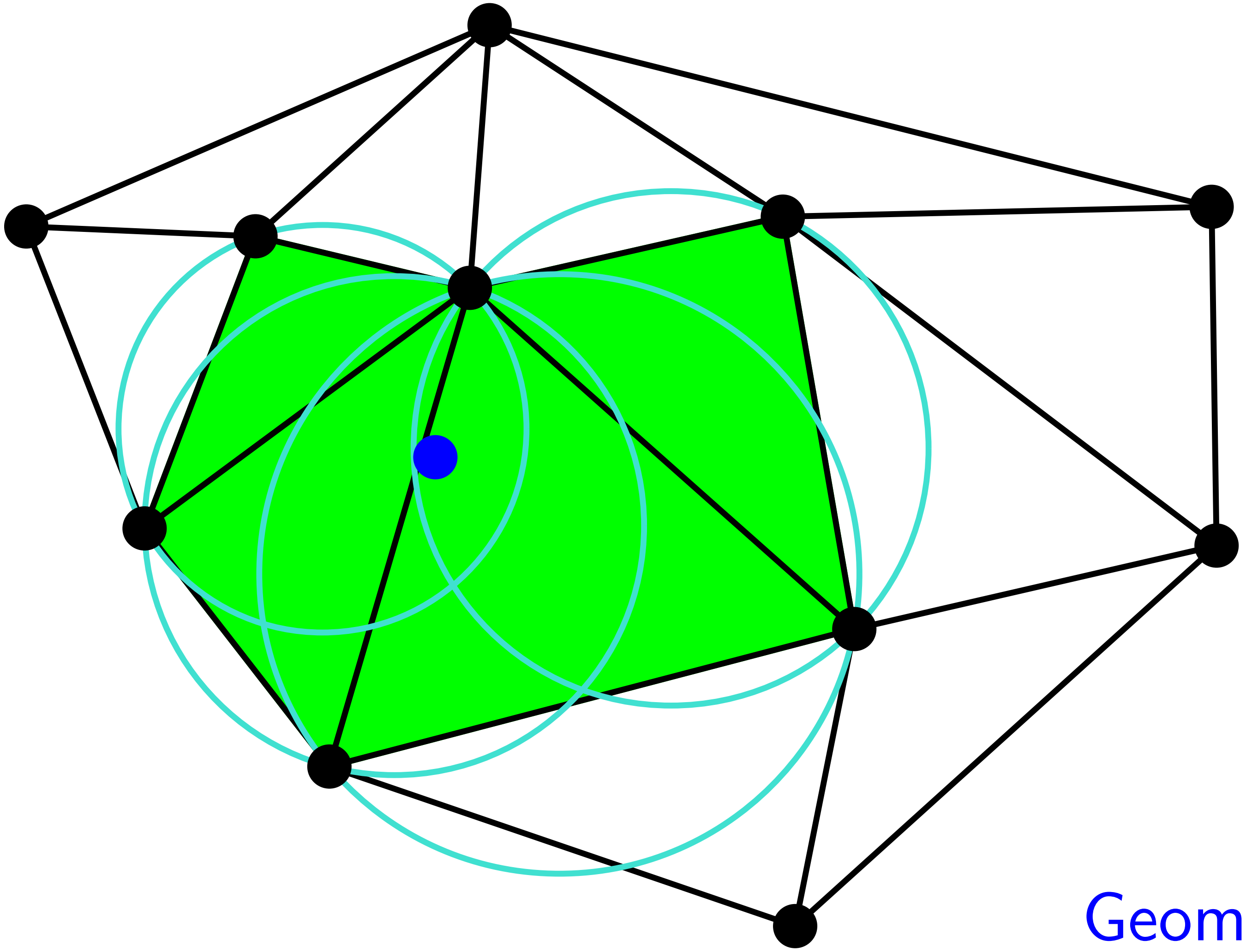


Locate



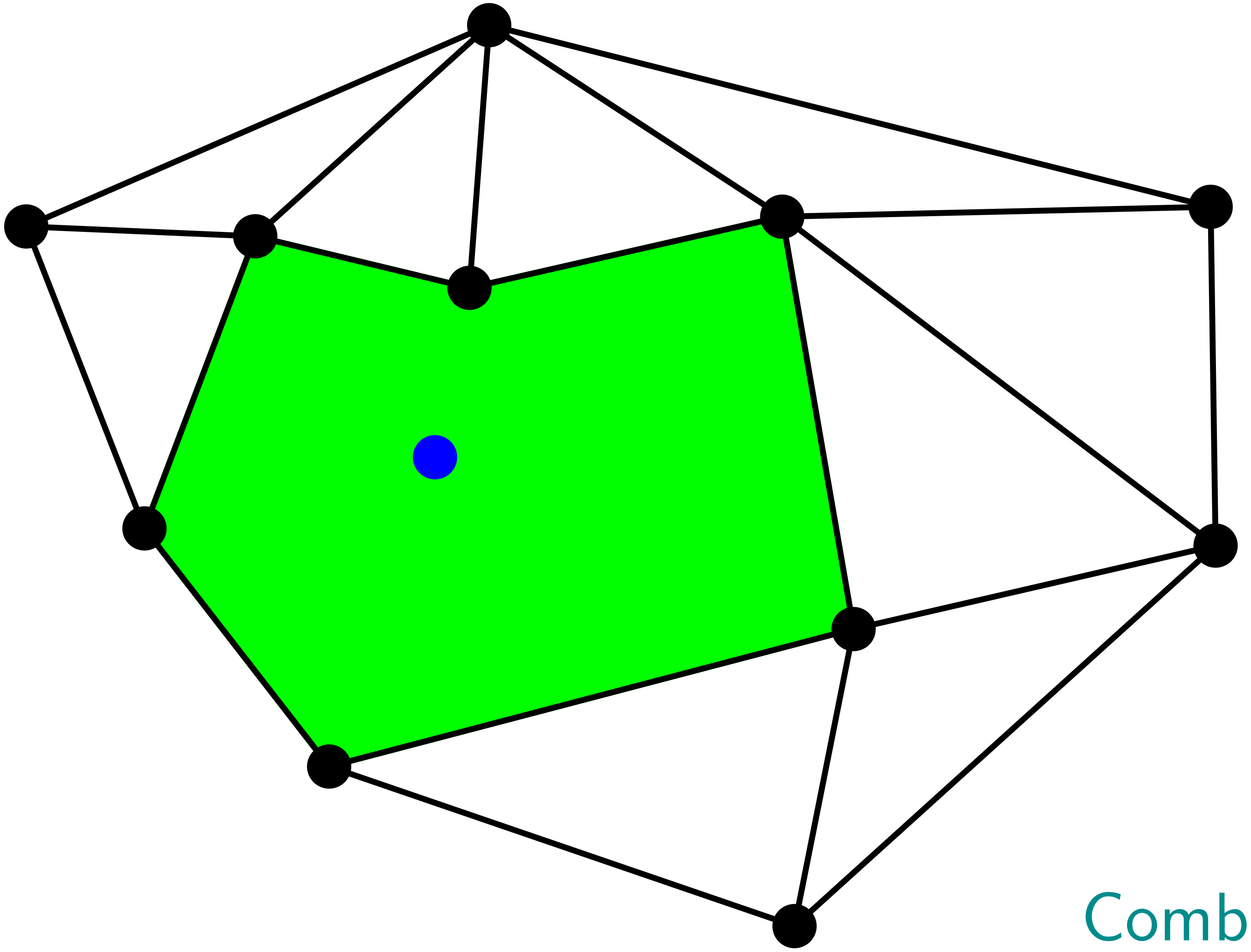
Geometry

Find conflicts



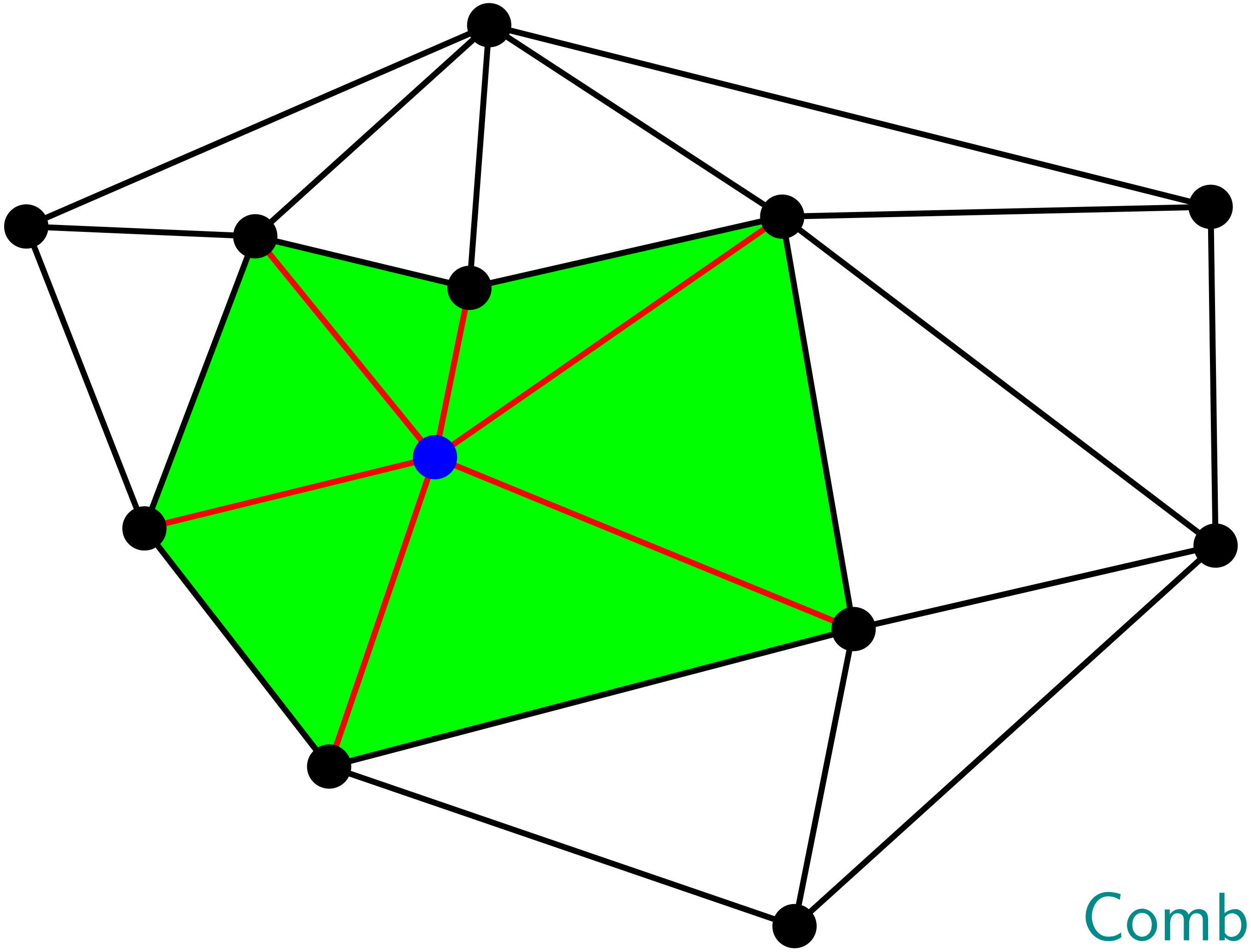
Geometry

Remove triangles



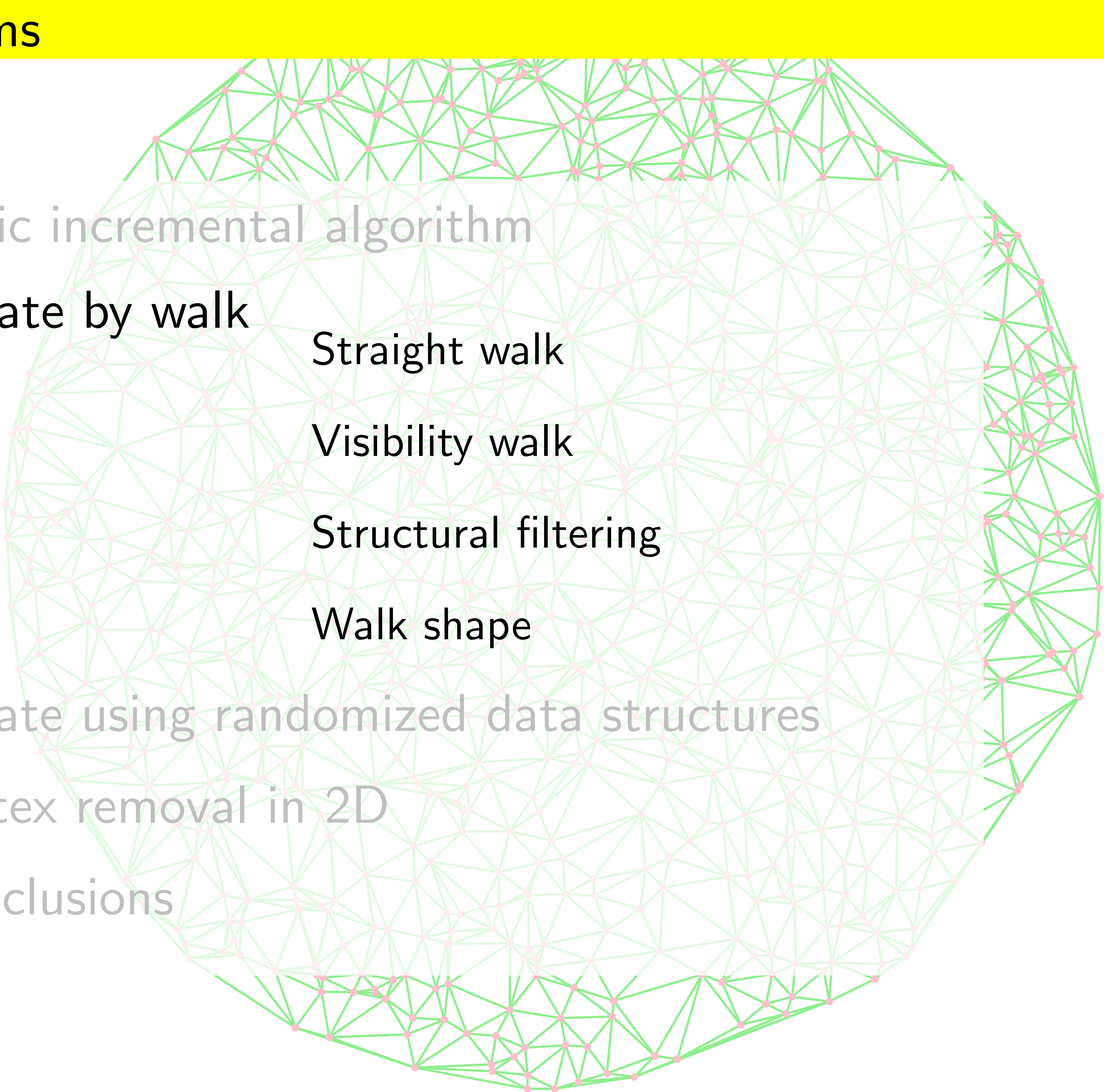
Combinatorics

Star the hole



Combinatorics

Algorithms



Basic incremental algorithm

Locate by walk

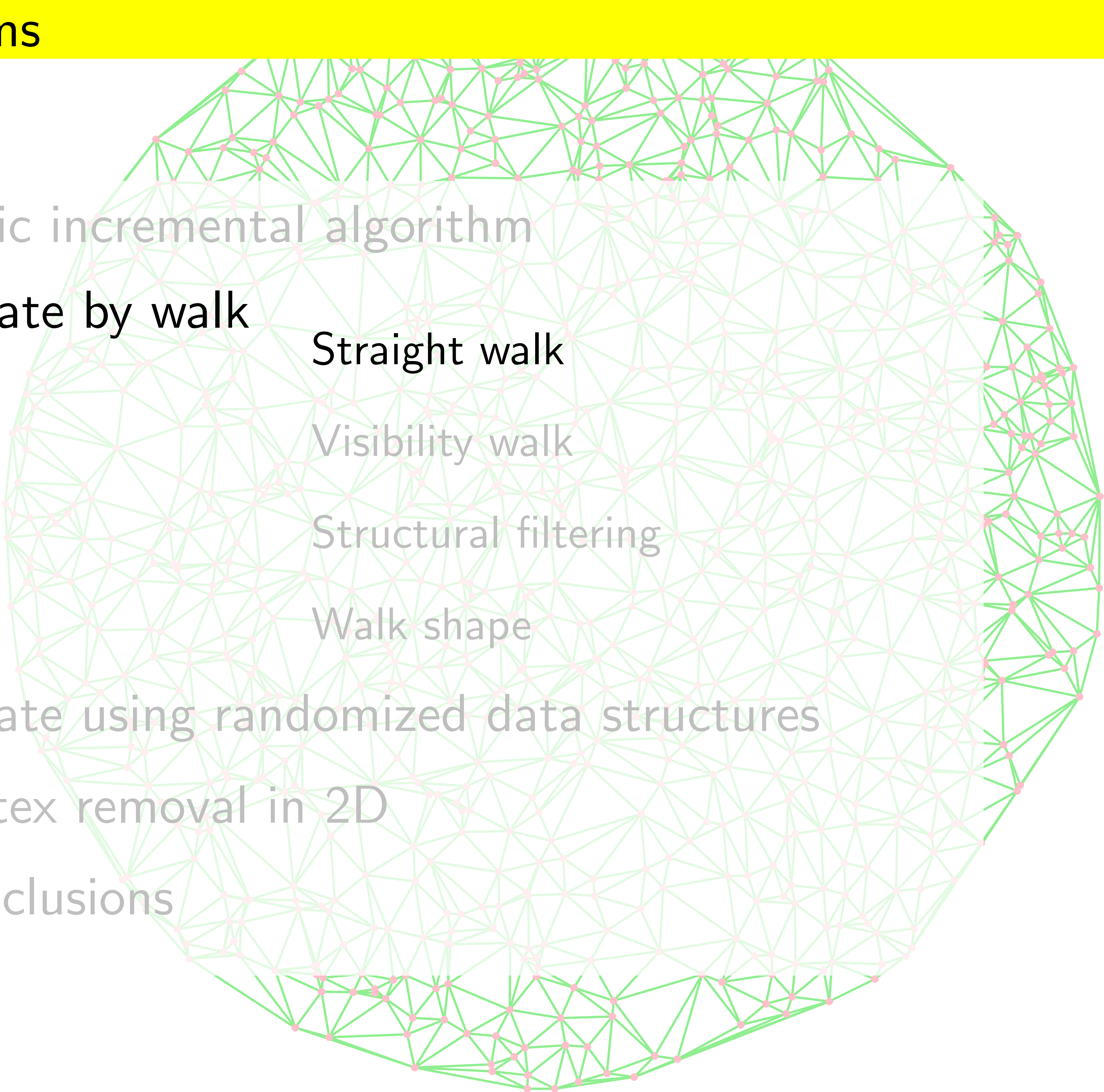
- Straight walk
- Visibility walk
- Structural filtering
- Walk shape

Locate using randomized data structures

Vertex removal in 2D

Conclusions

Algorithms



Basic incremental algorithm

Locate by walk

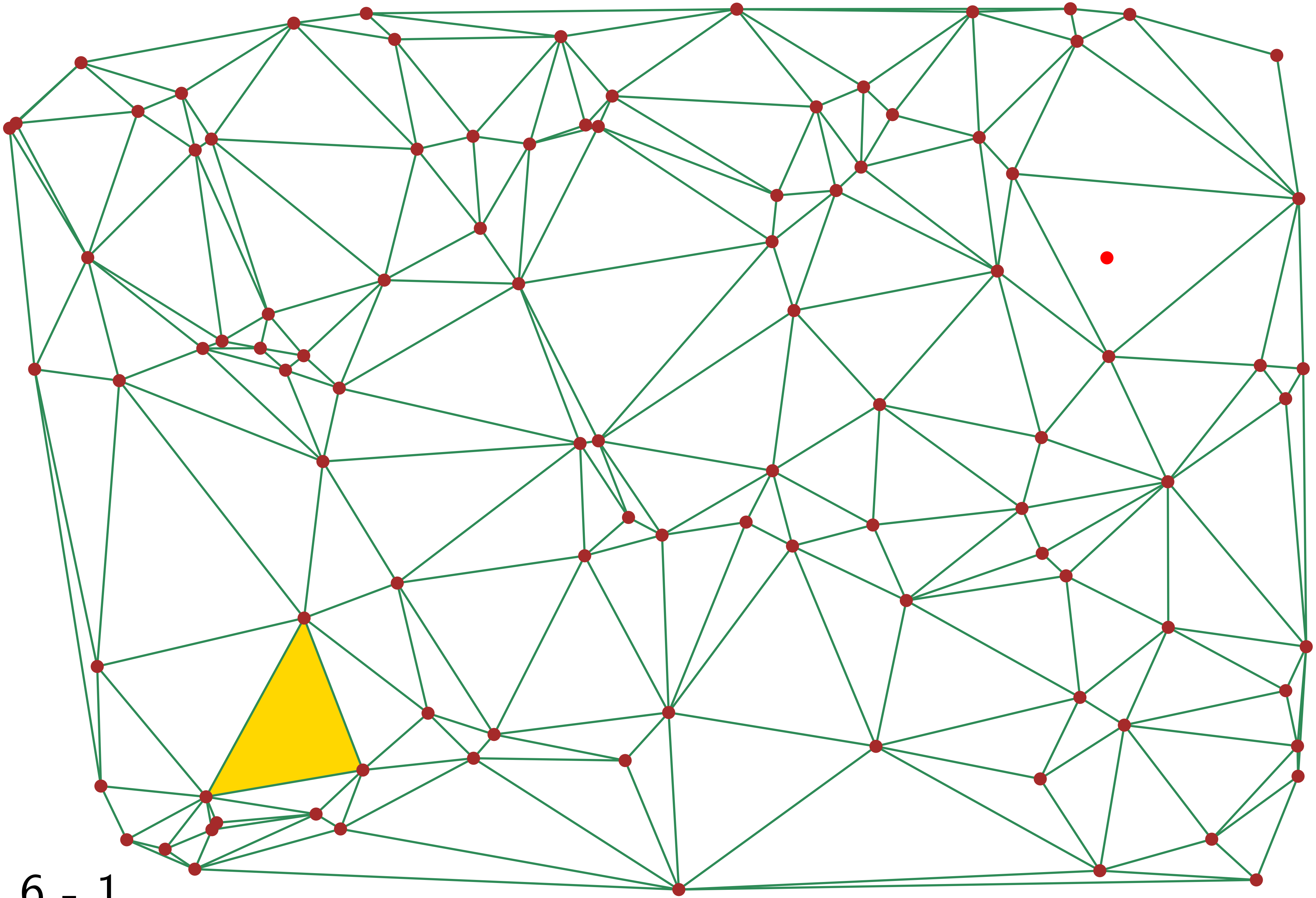
- Straight walk
- Visibility walk
- Structural filtering
- Walk shape

Locate using randomized data structures

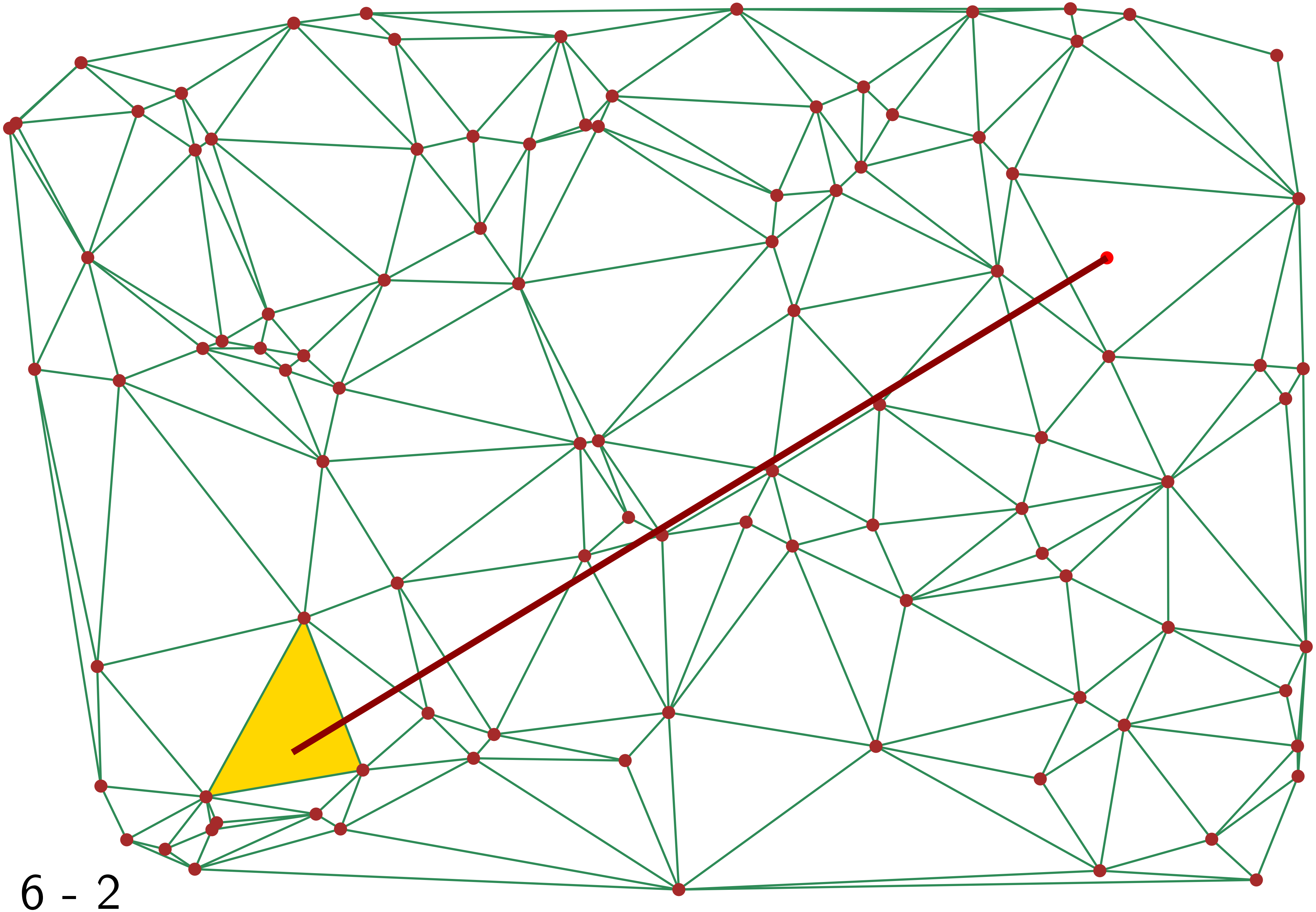
Vertex removal in 2D

Conclusions

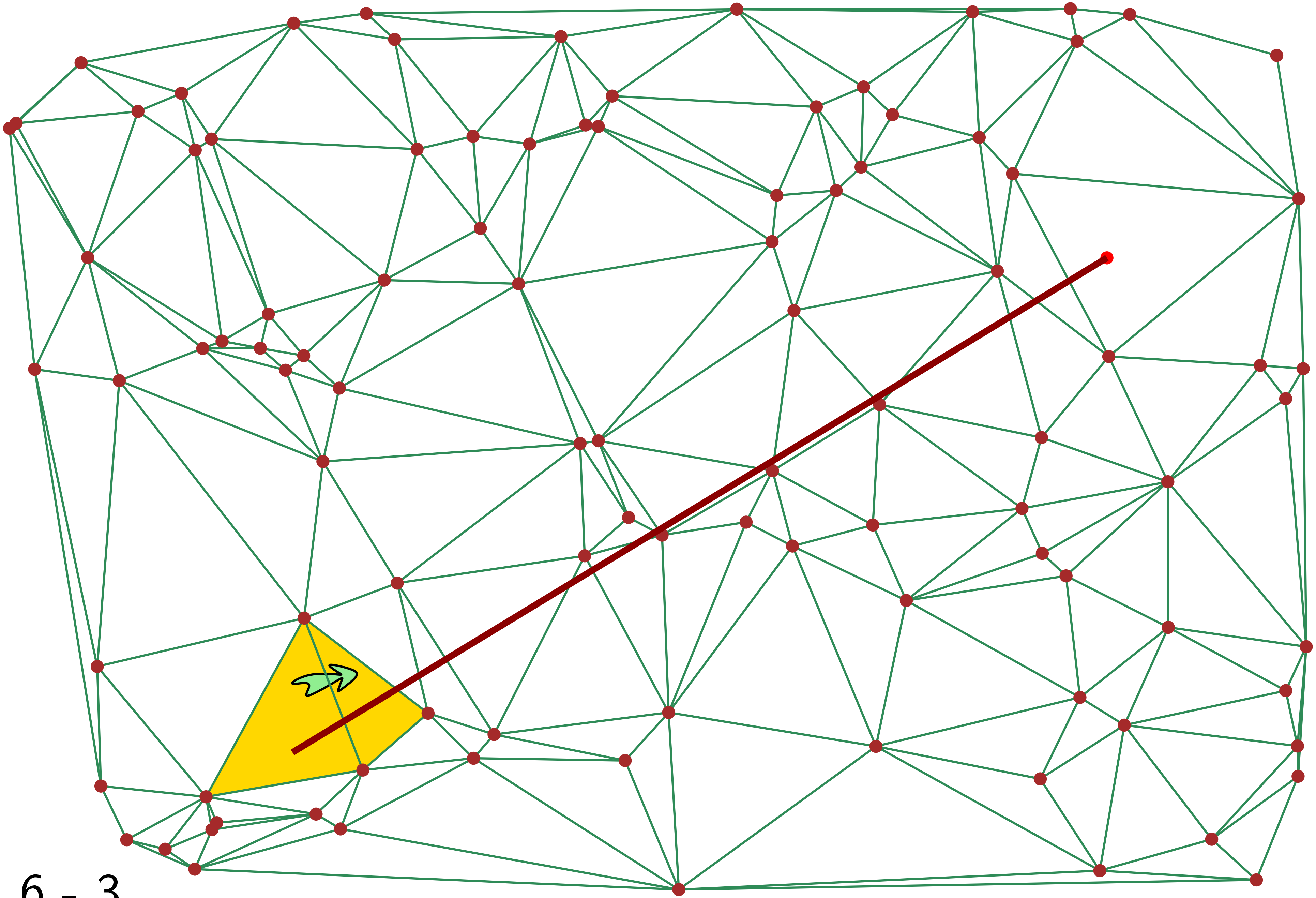
straight walk



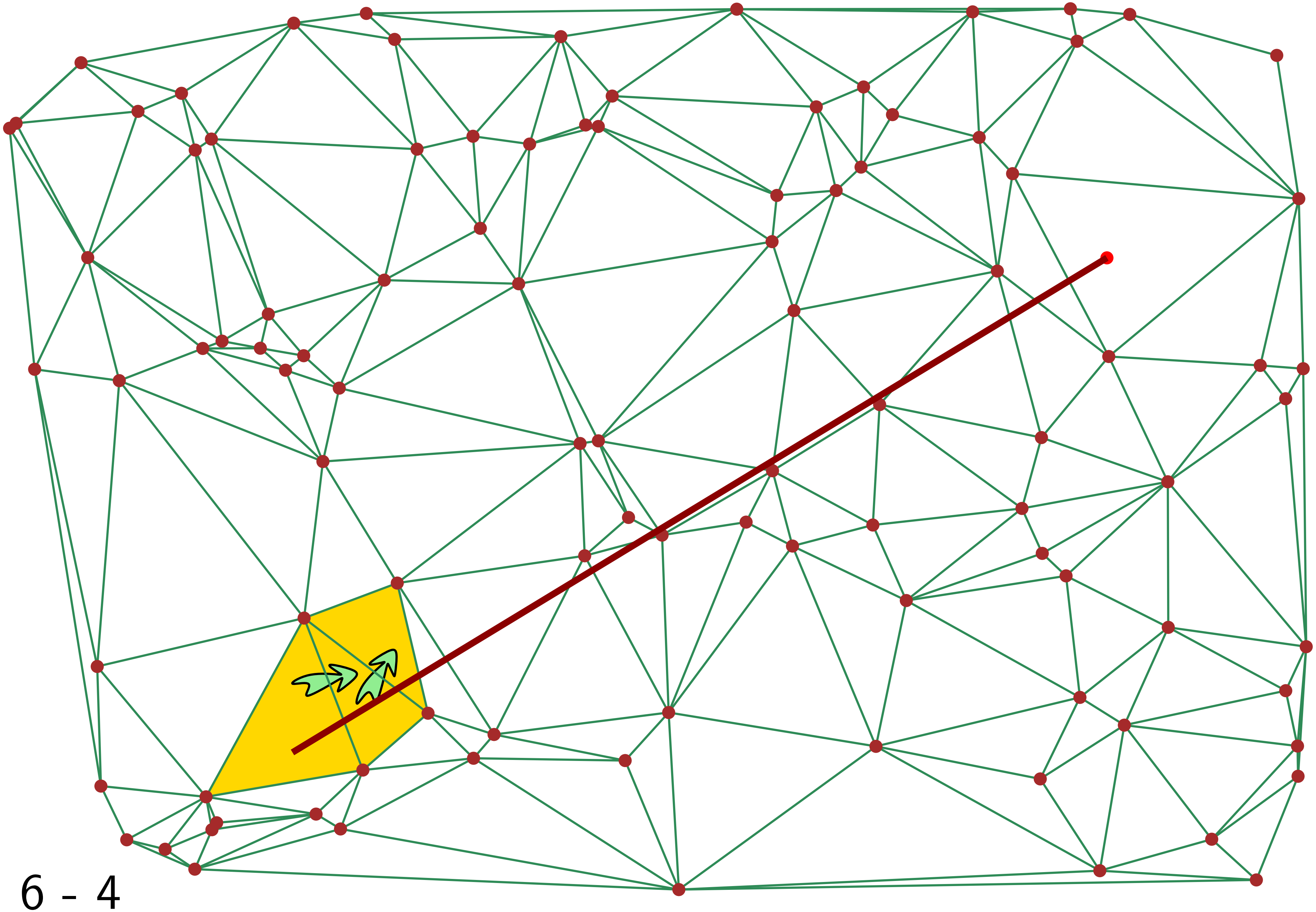
straight walk



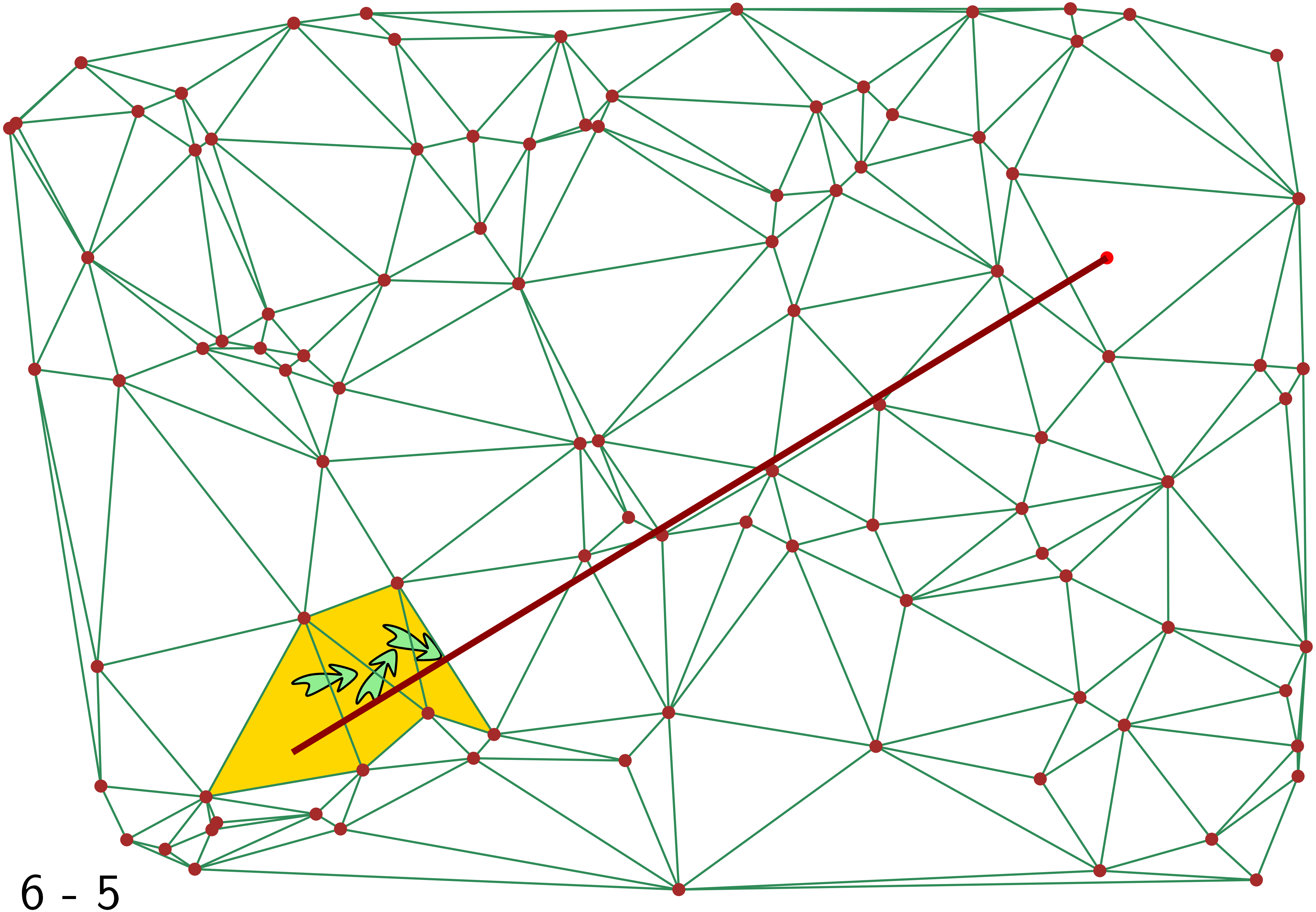
straight walk



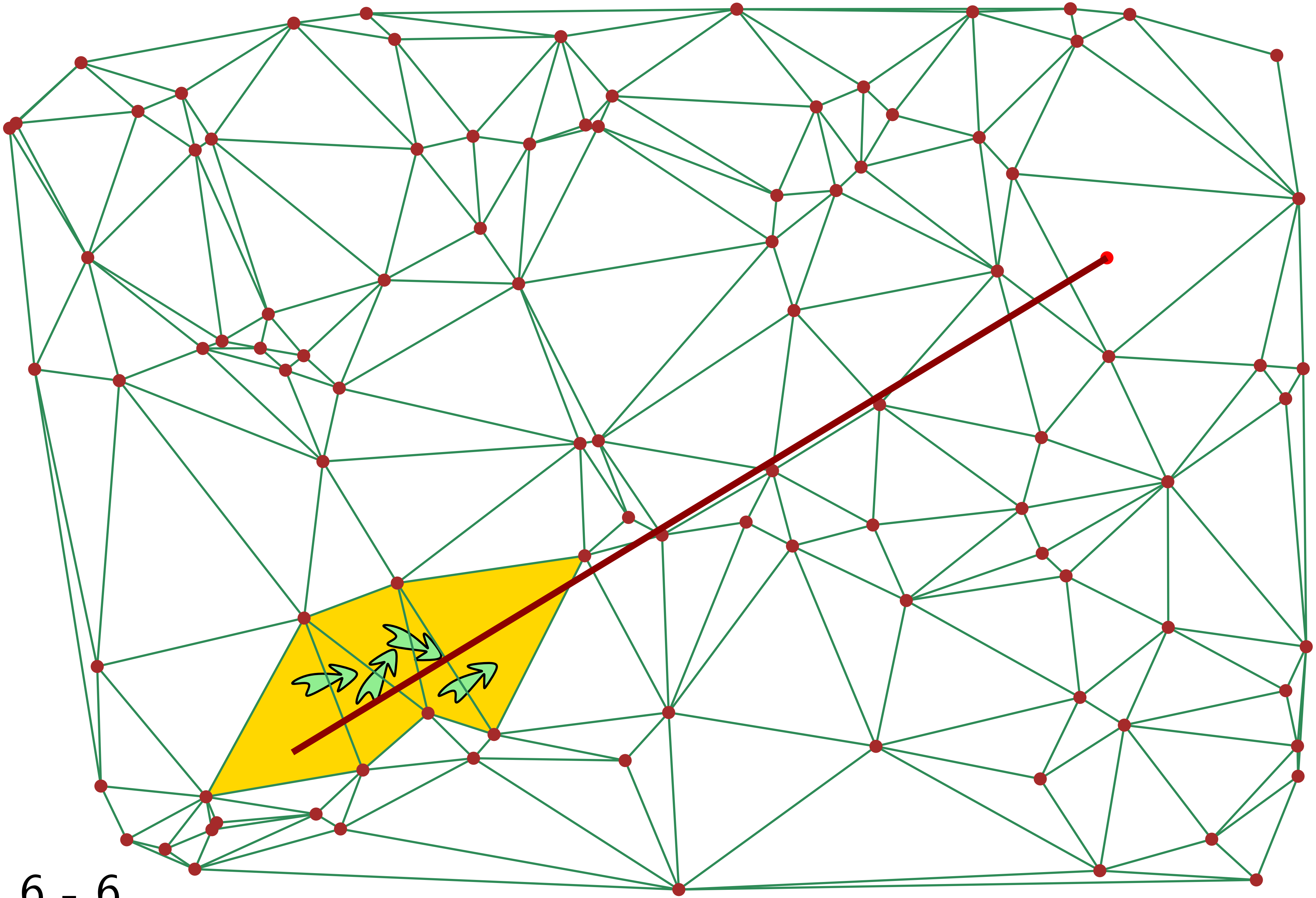
straight walk



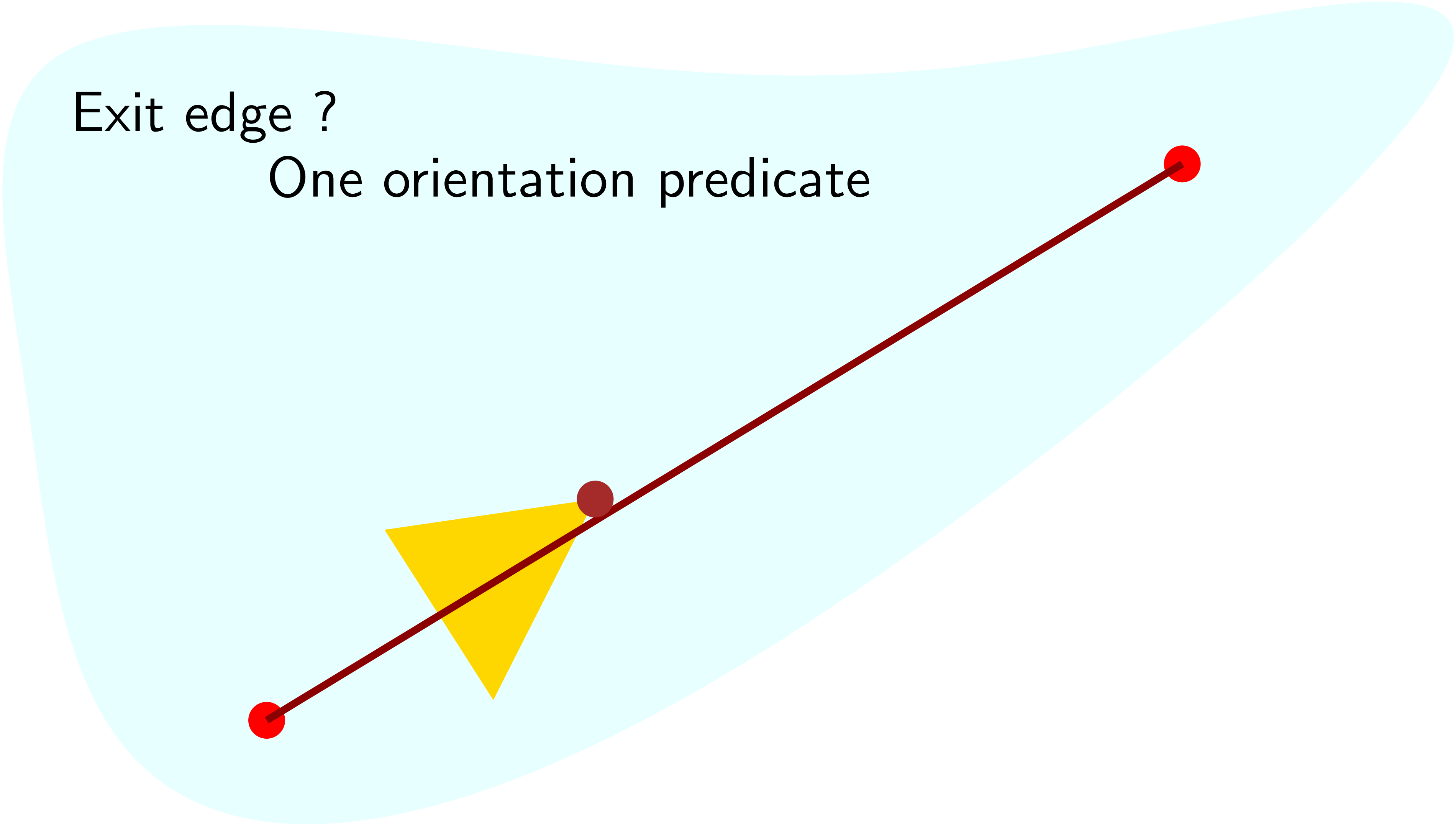
straight walk



straight walk



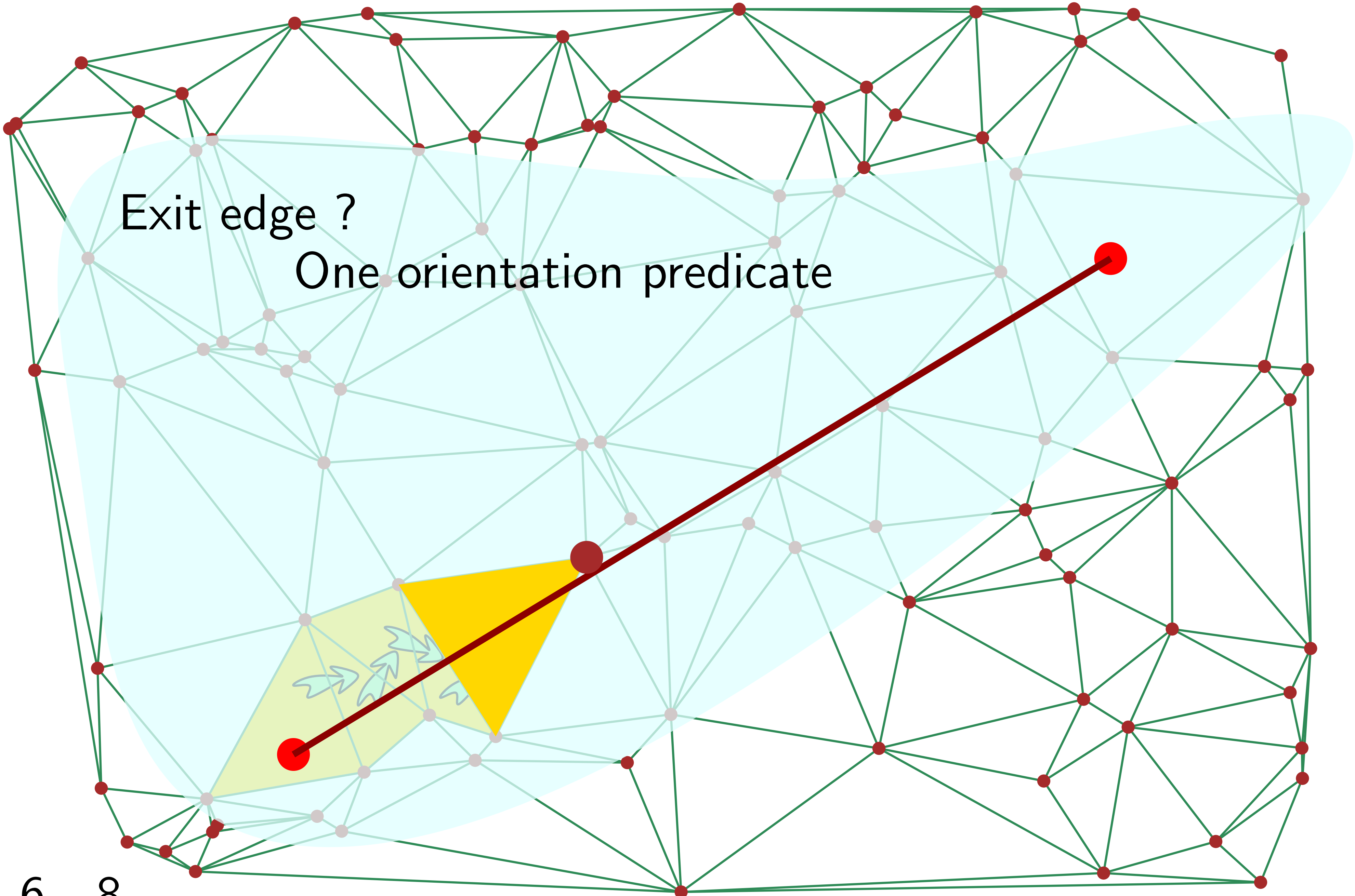
straight walk



Exit edge ?

One orientation predicate

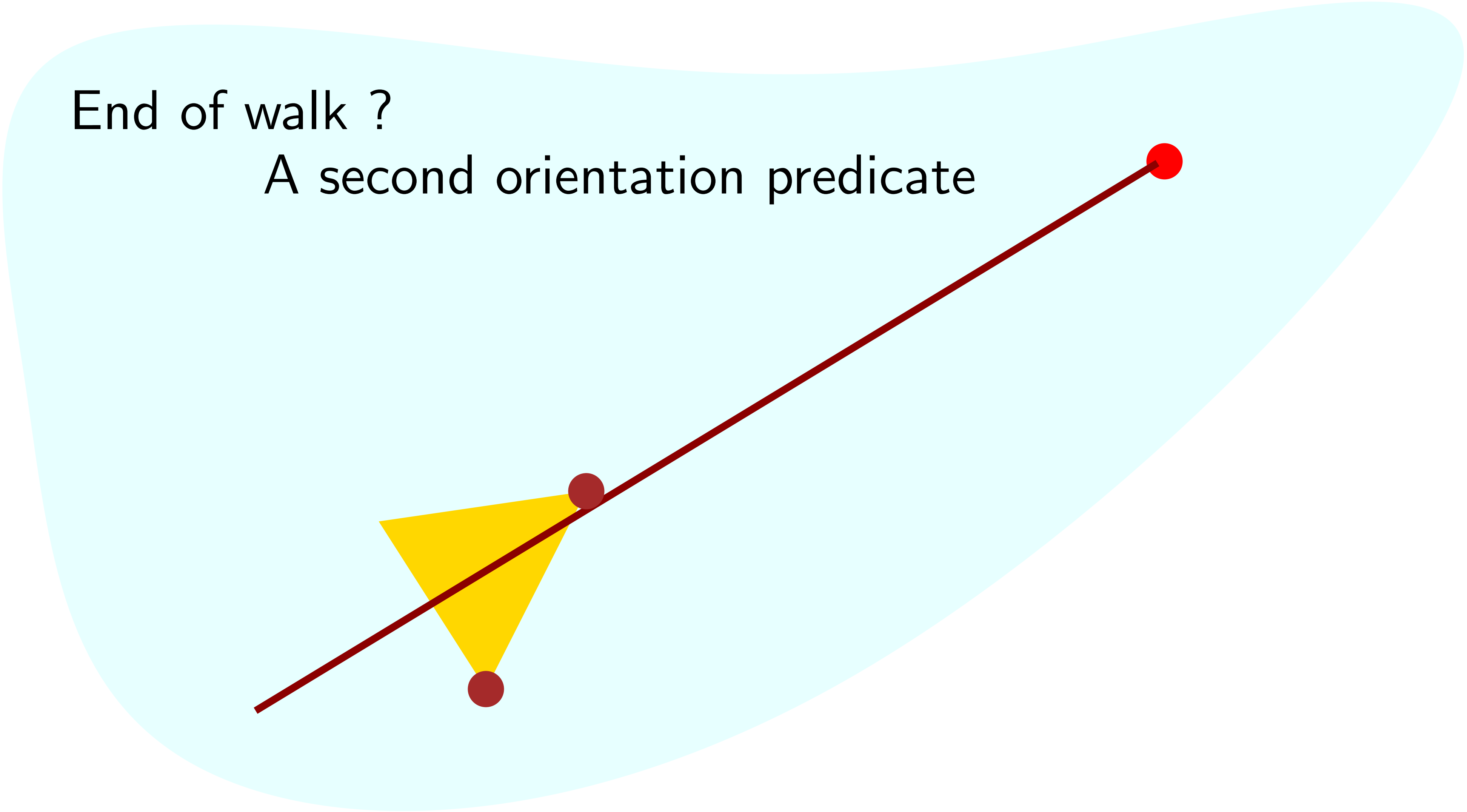
straight walk



Exit edge ?

One orientation predicate

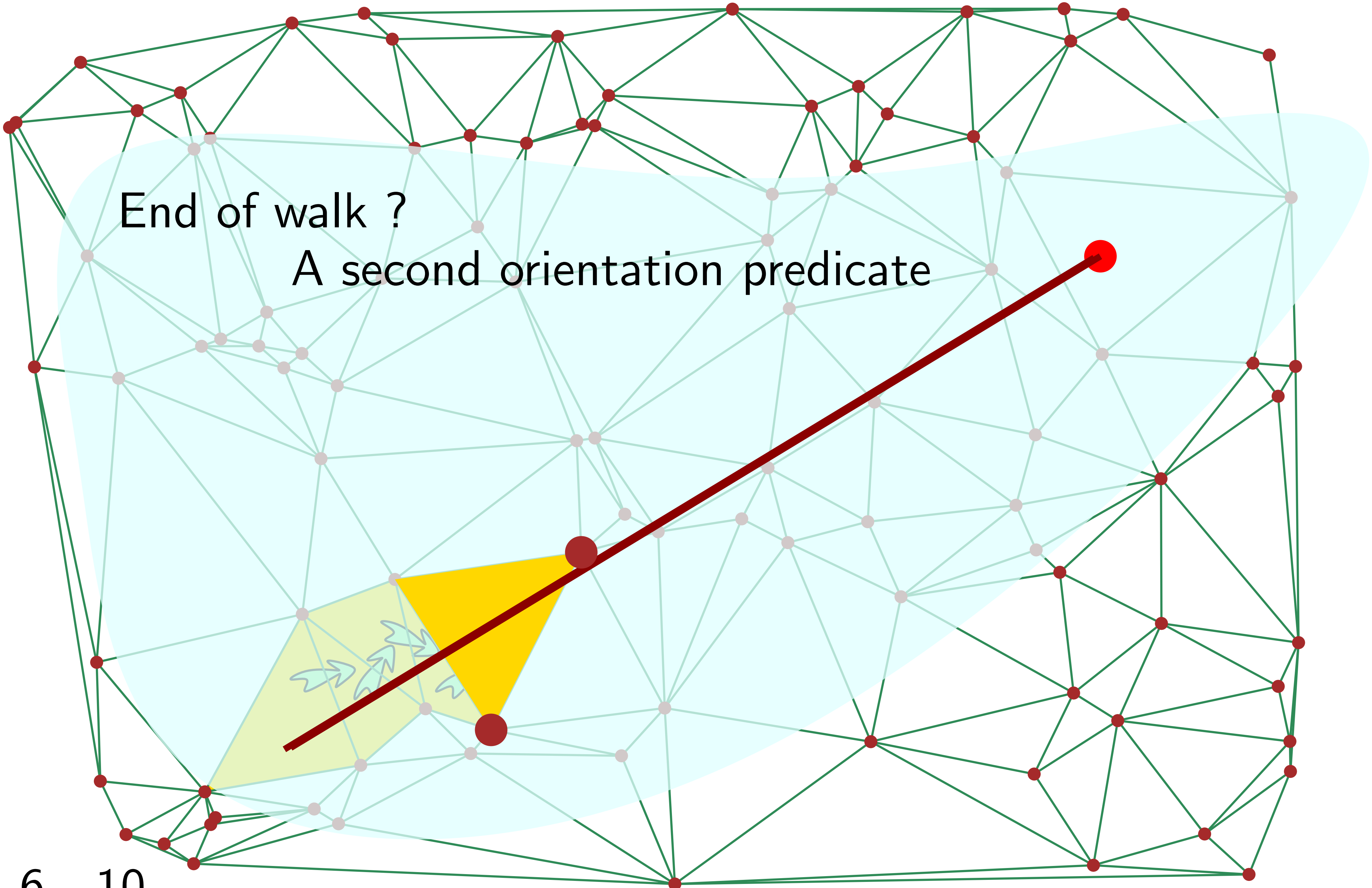
straight walk



End of walk ?

A second orientation predicate

straight walk

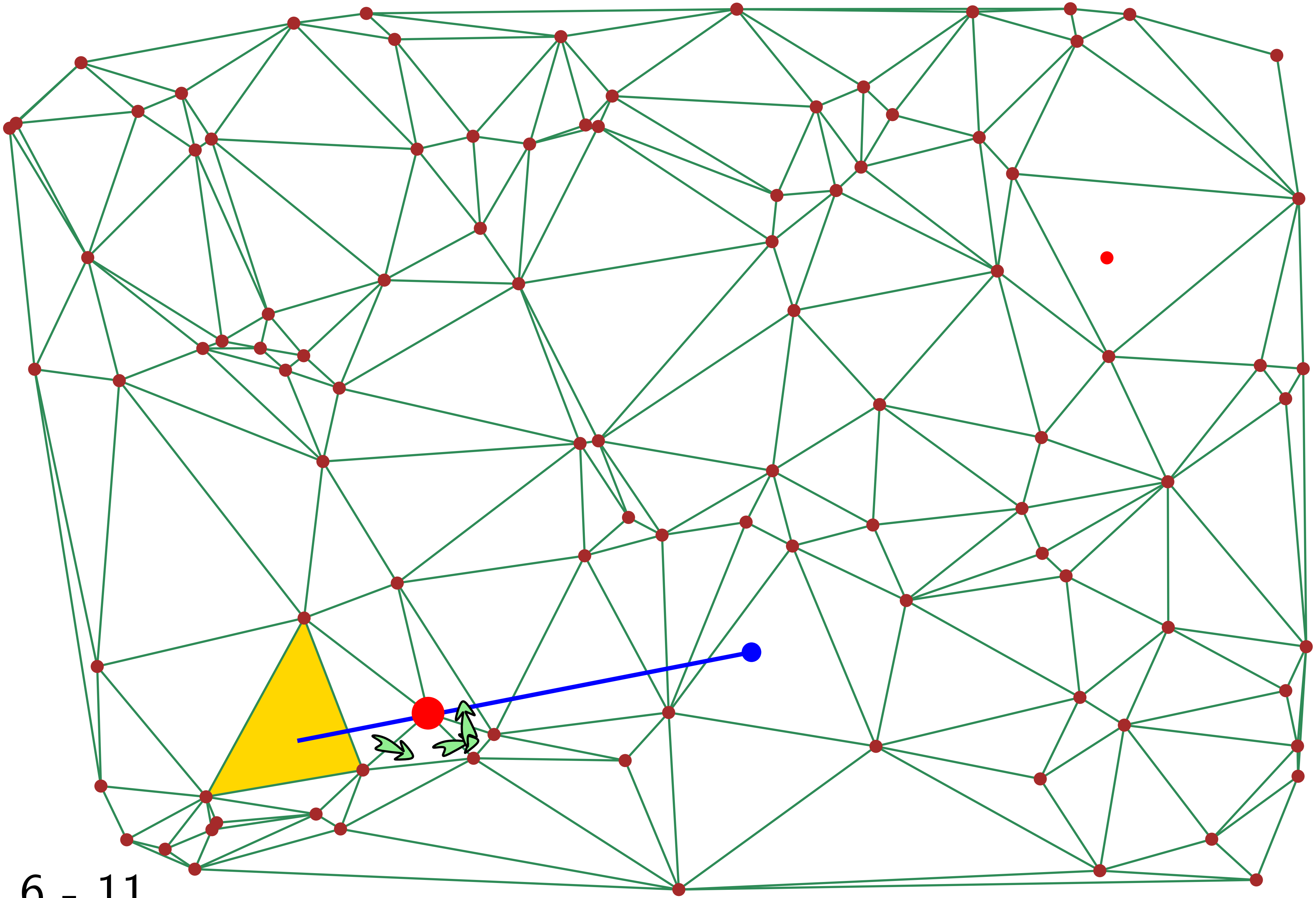


End of walk ?

A second orientation predicate

straight walk

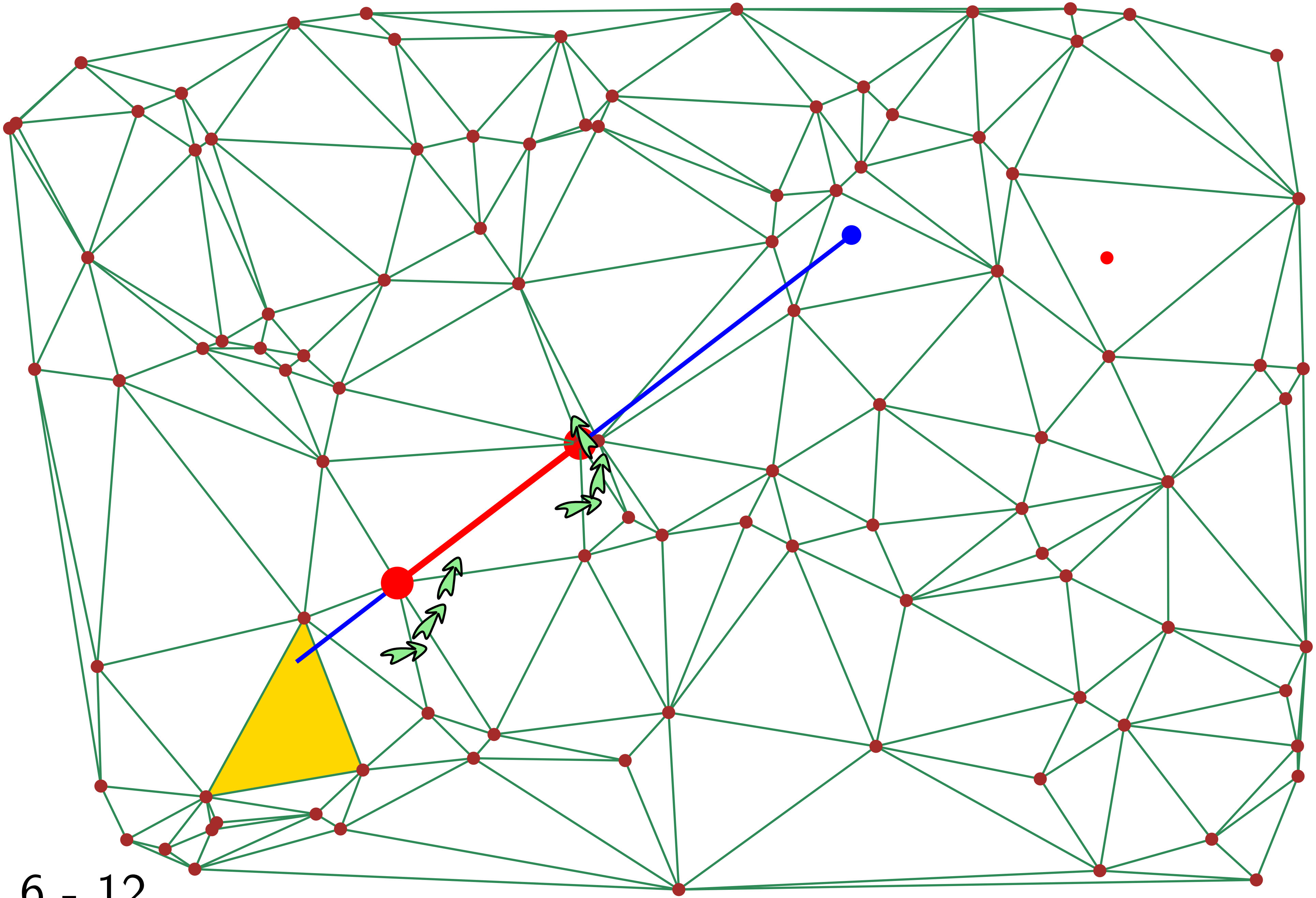
degeneracies



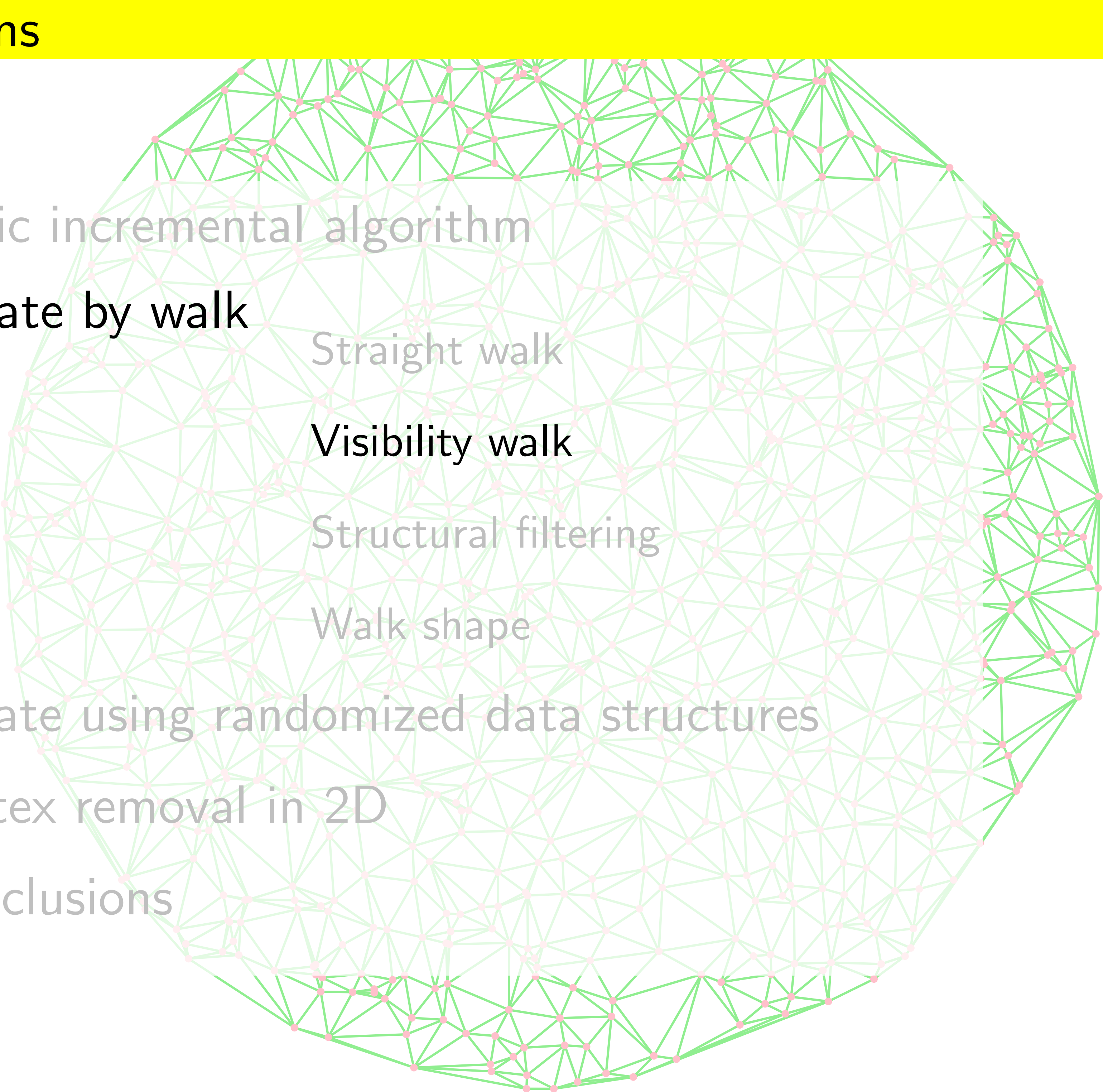
straight walk

degeneracies

(imagine 3D...)

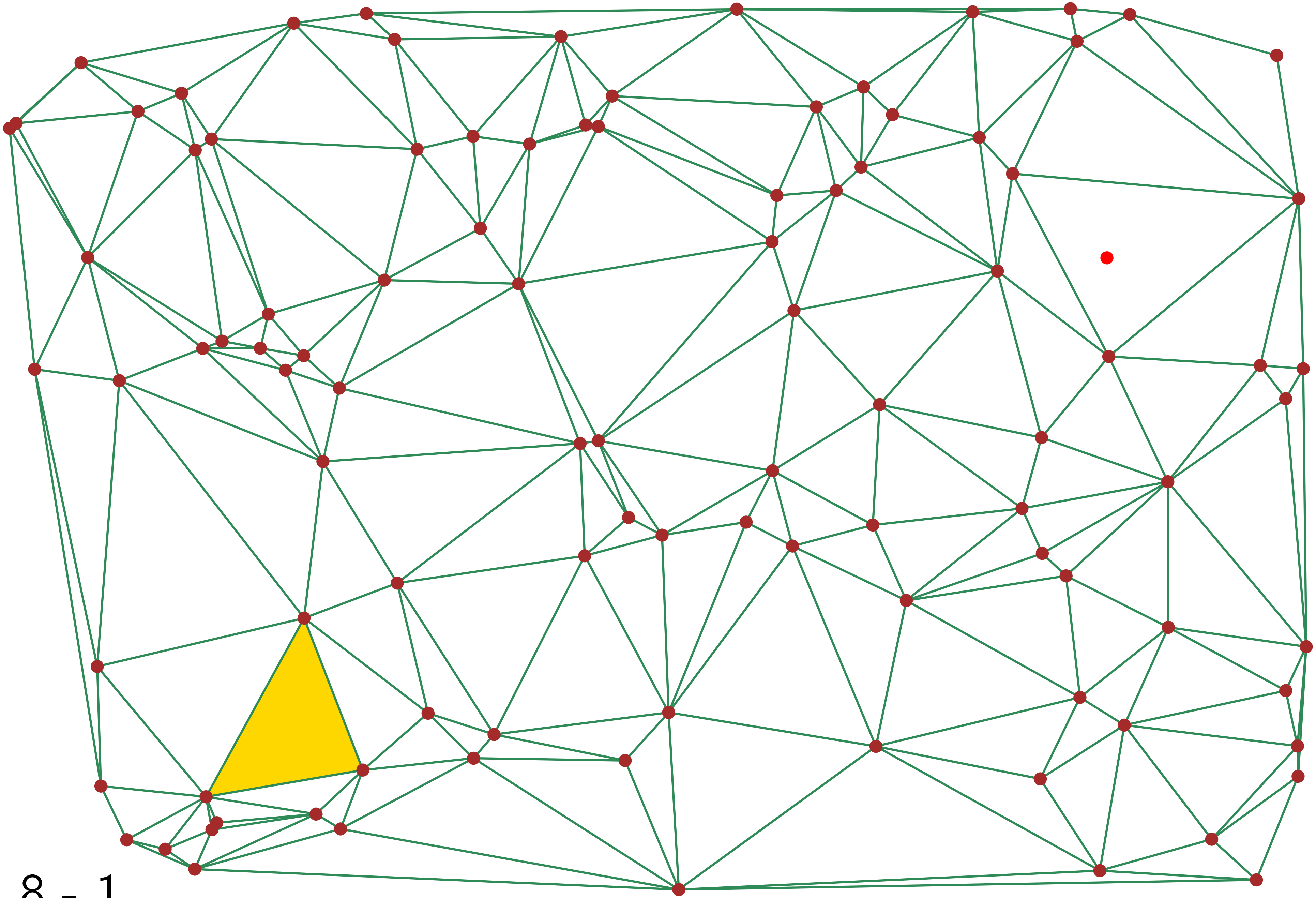


Algorithms

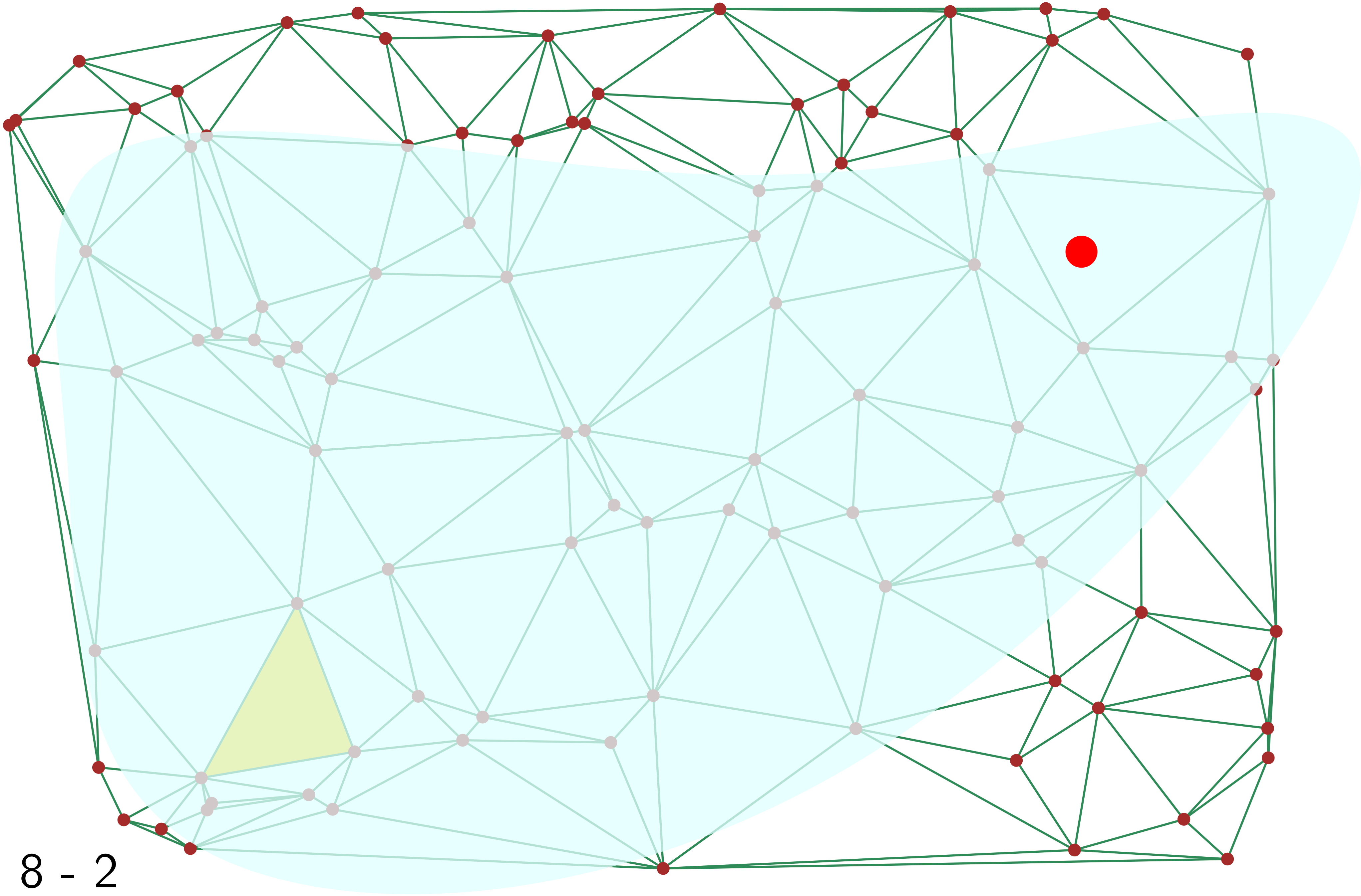


- Basic incremental algorithm
- Locate by walk
  - Straight walk
  - Visibility walk
  - Structural filtering
  - Walk shape
- Locate using randomized data structures
- Vertex removal in 2D
- Conclusions

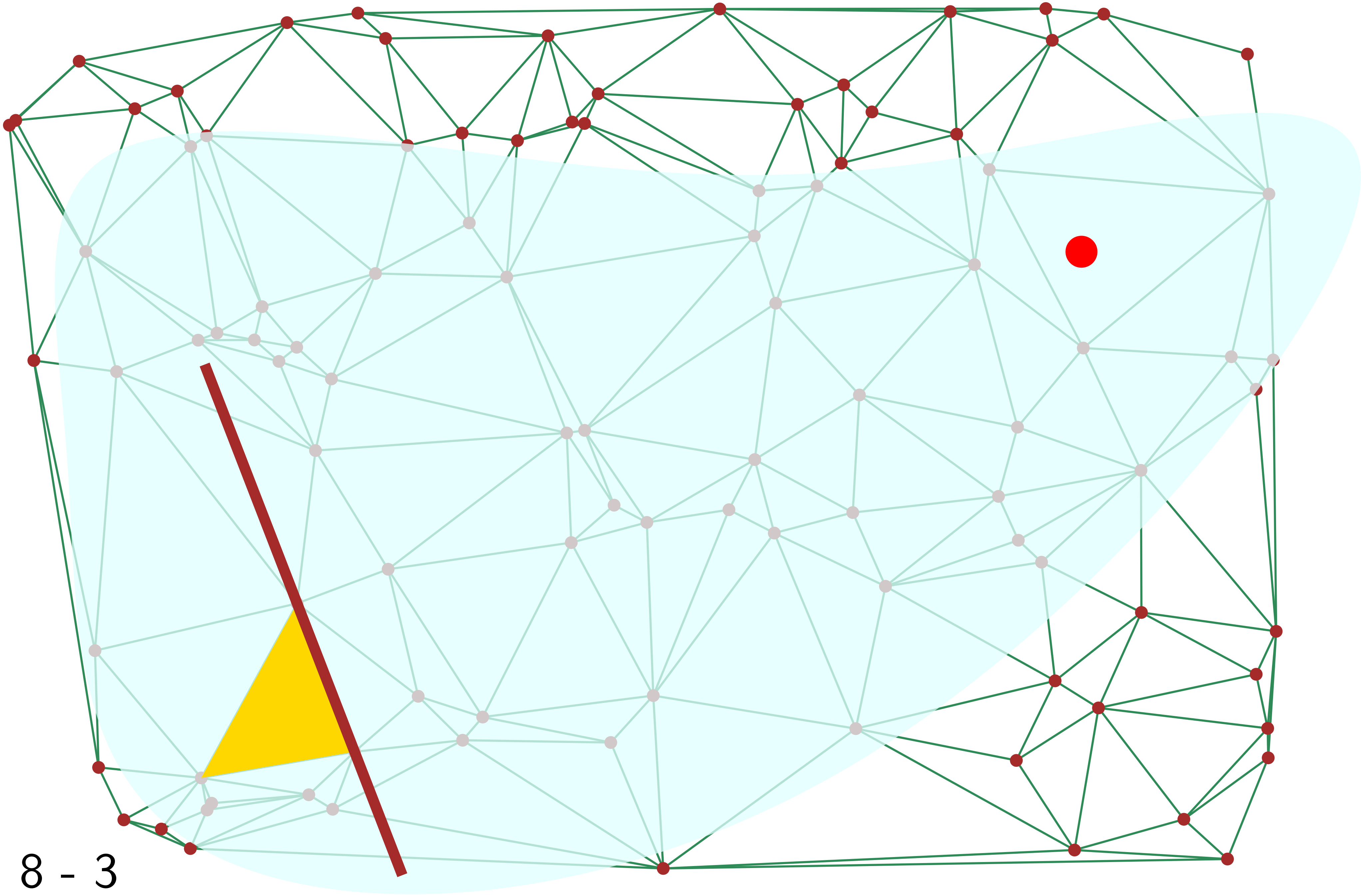
visibility walk



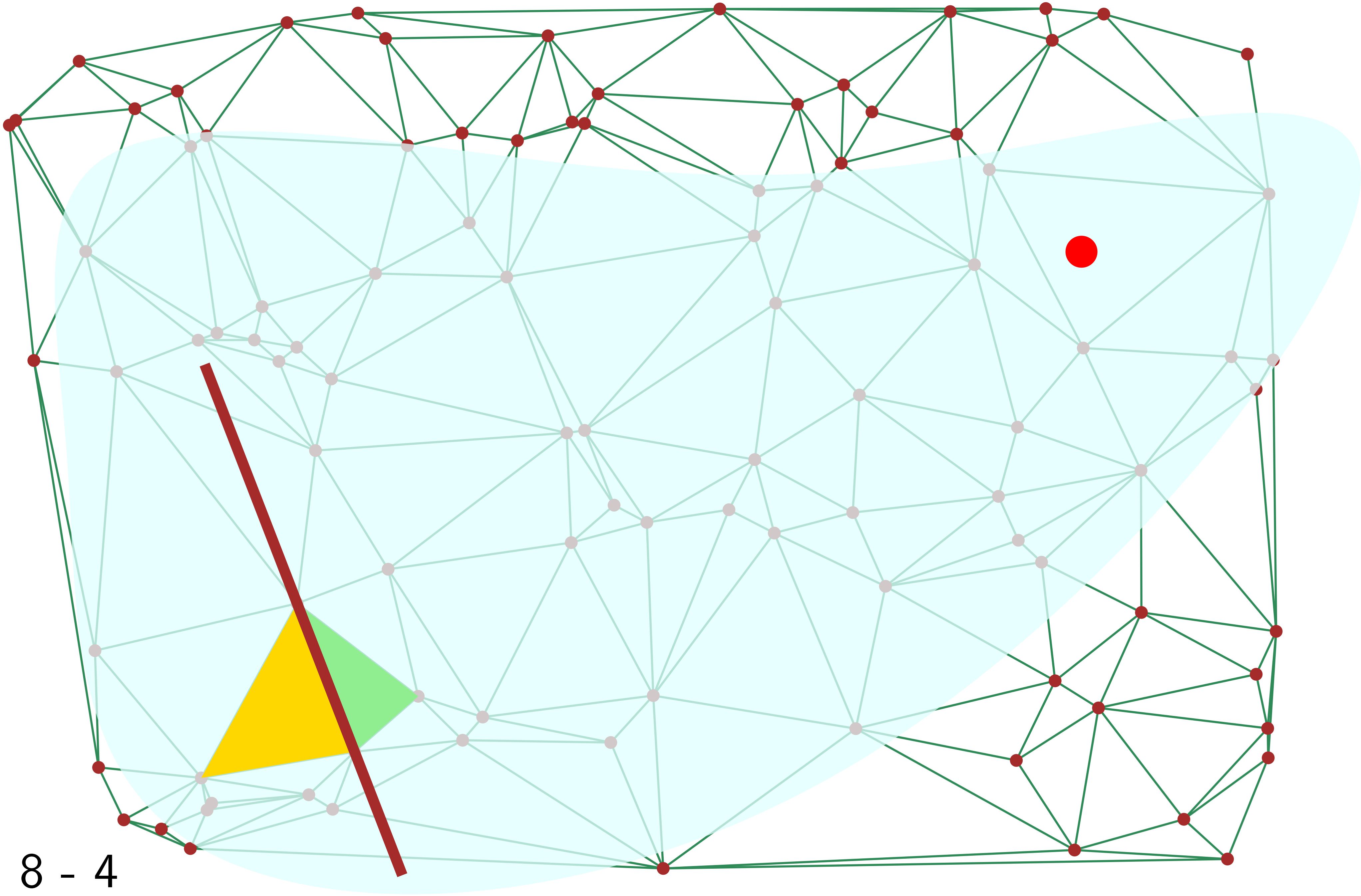
visibility walk



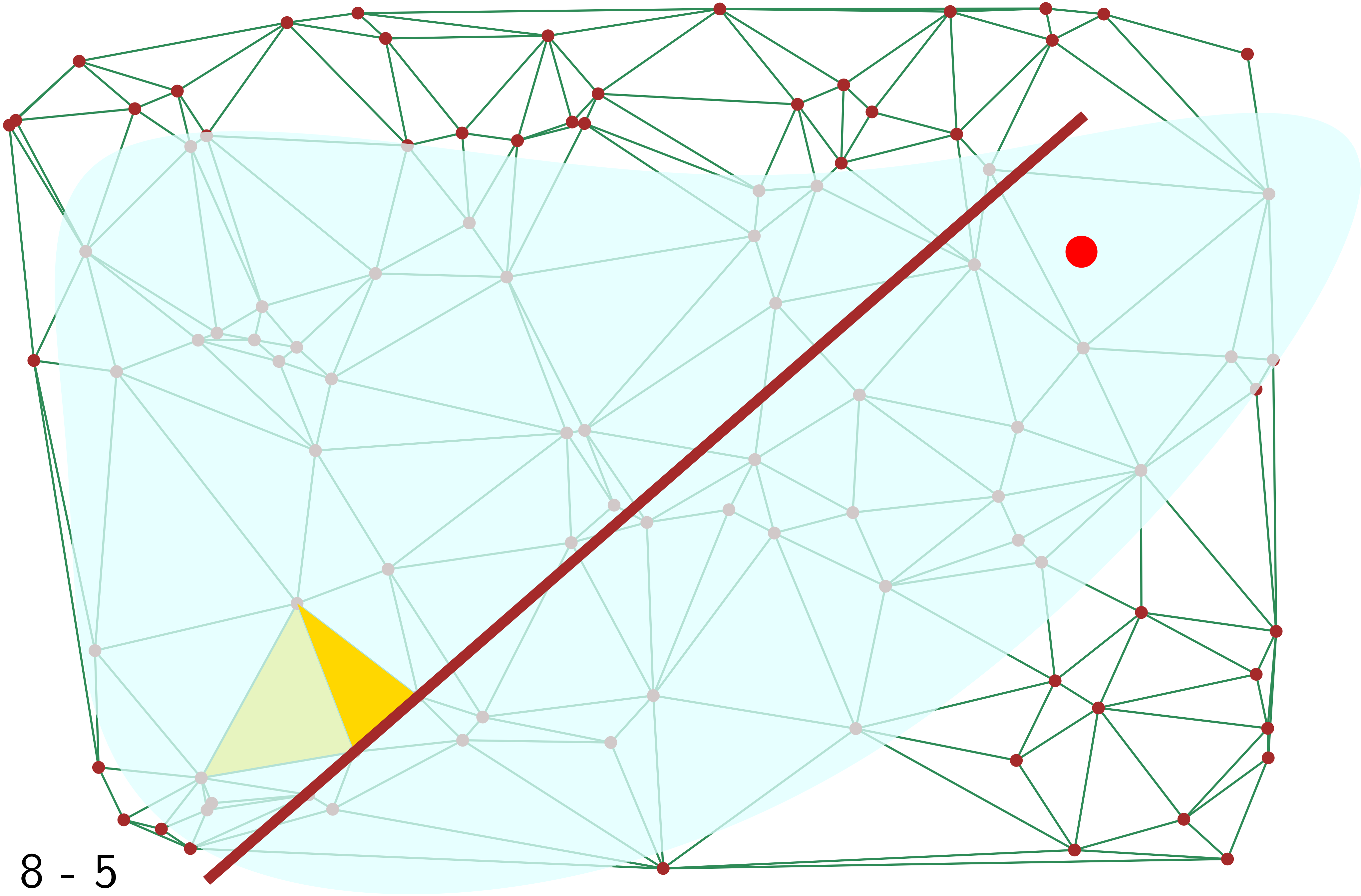
visibility walk



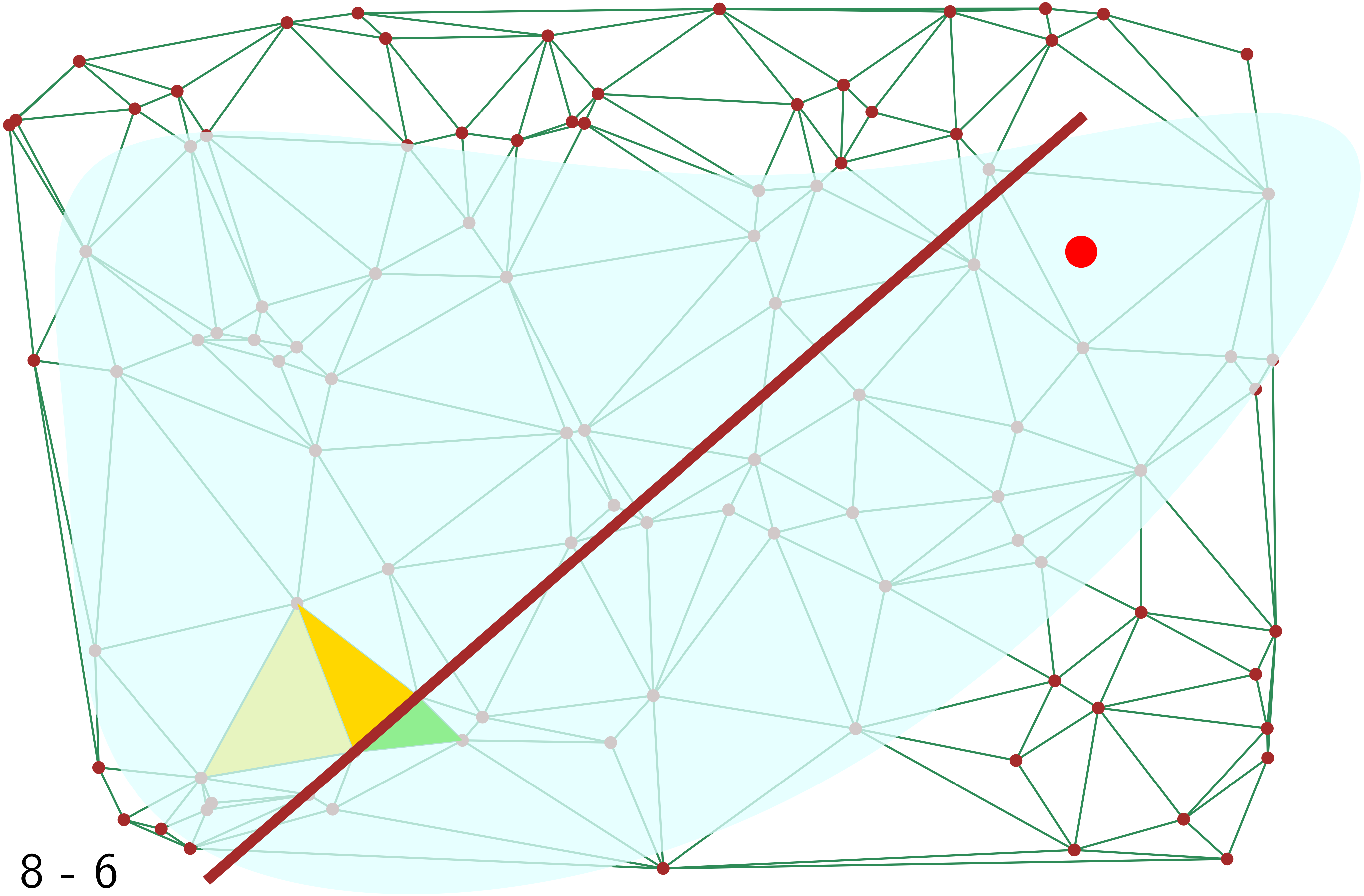
visibility walk



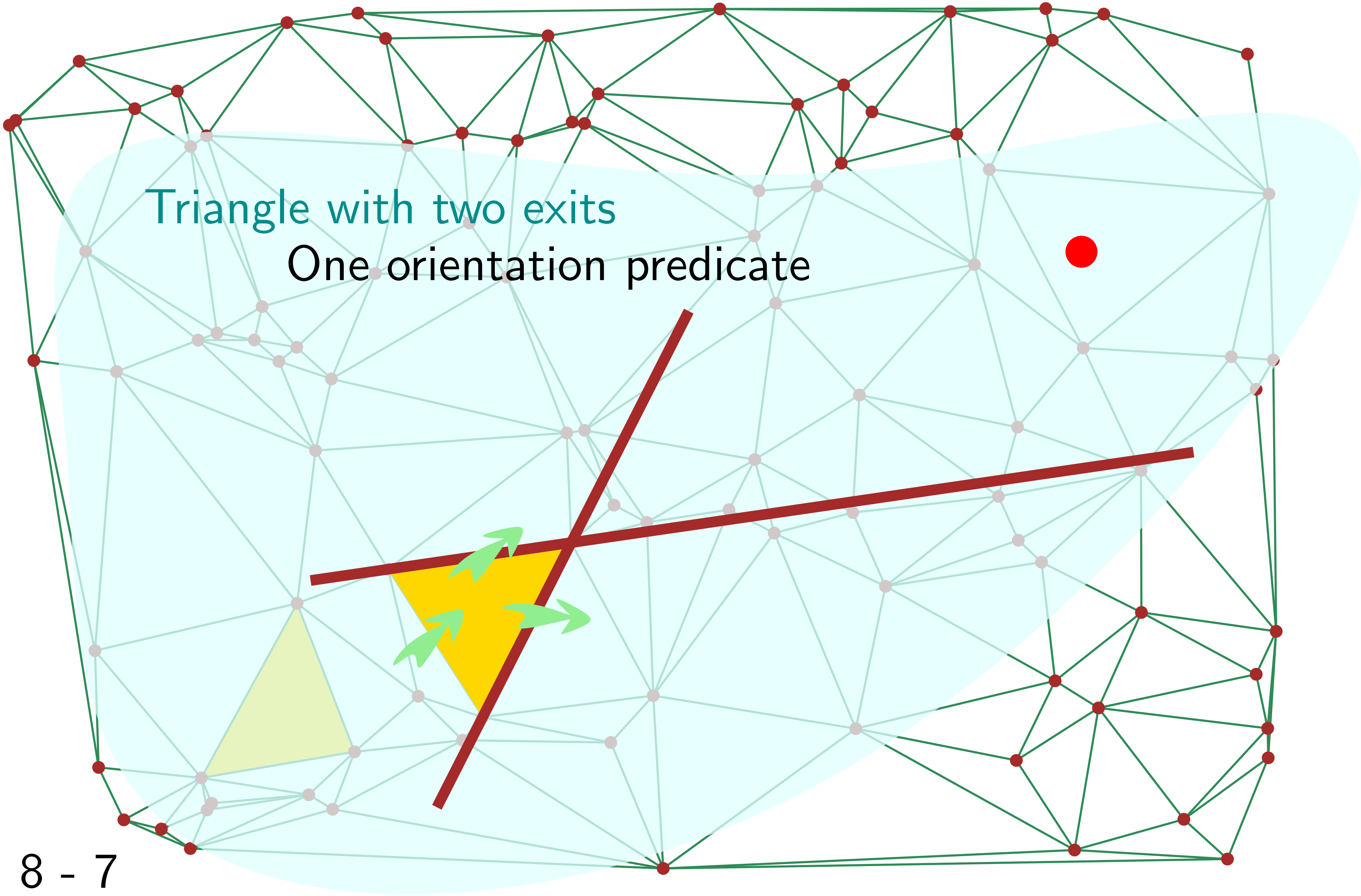
visibility walk



visibility walk



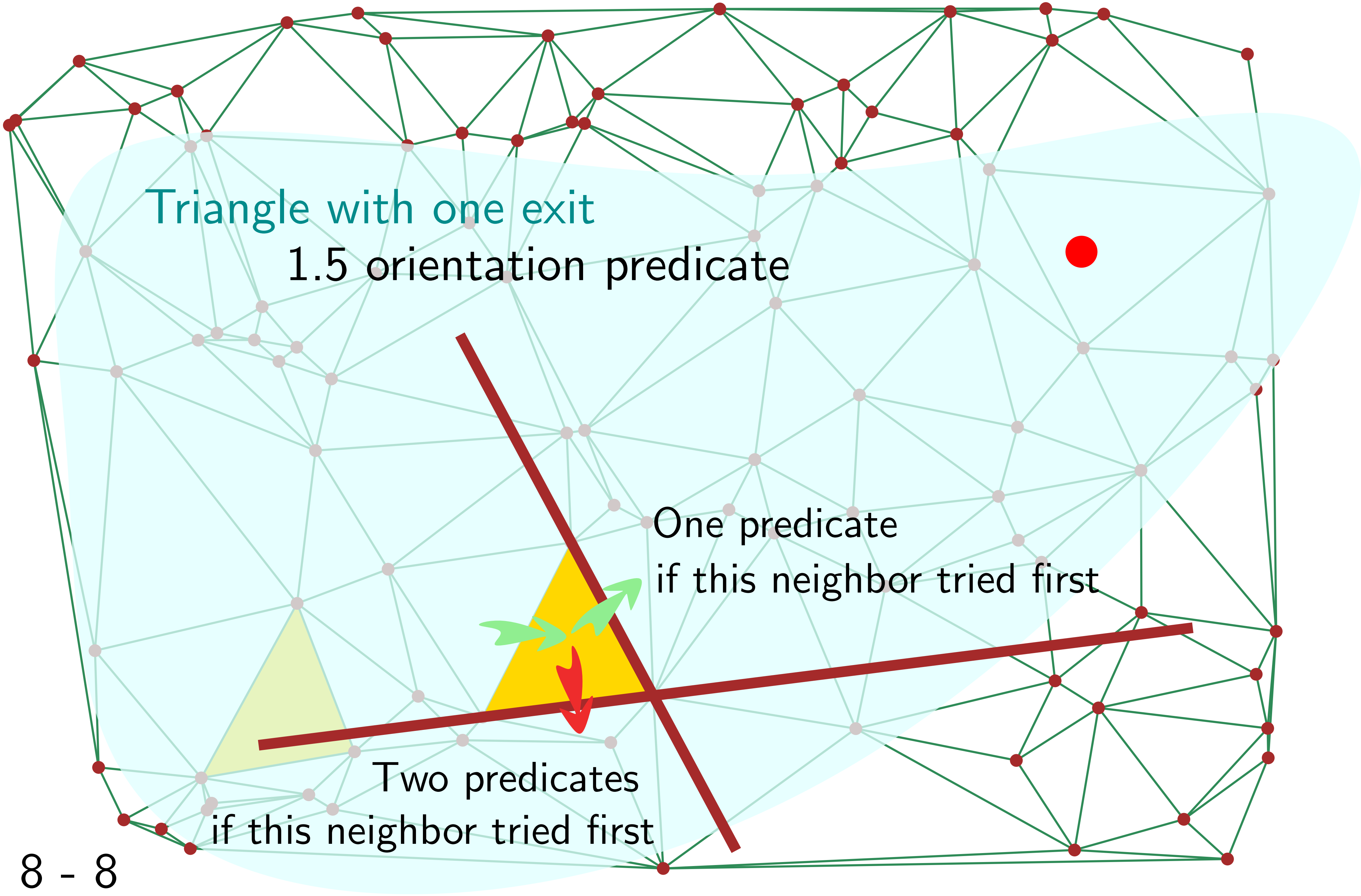
visibility walk



Triangle with two exits

One orientation predicate

visibility walk



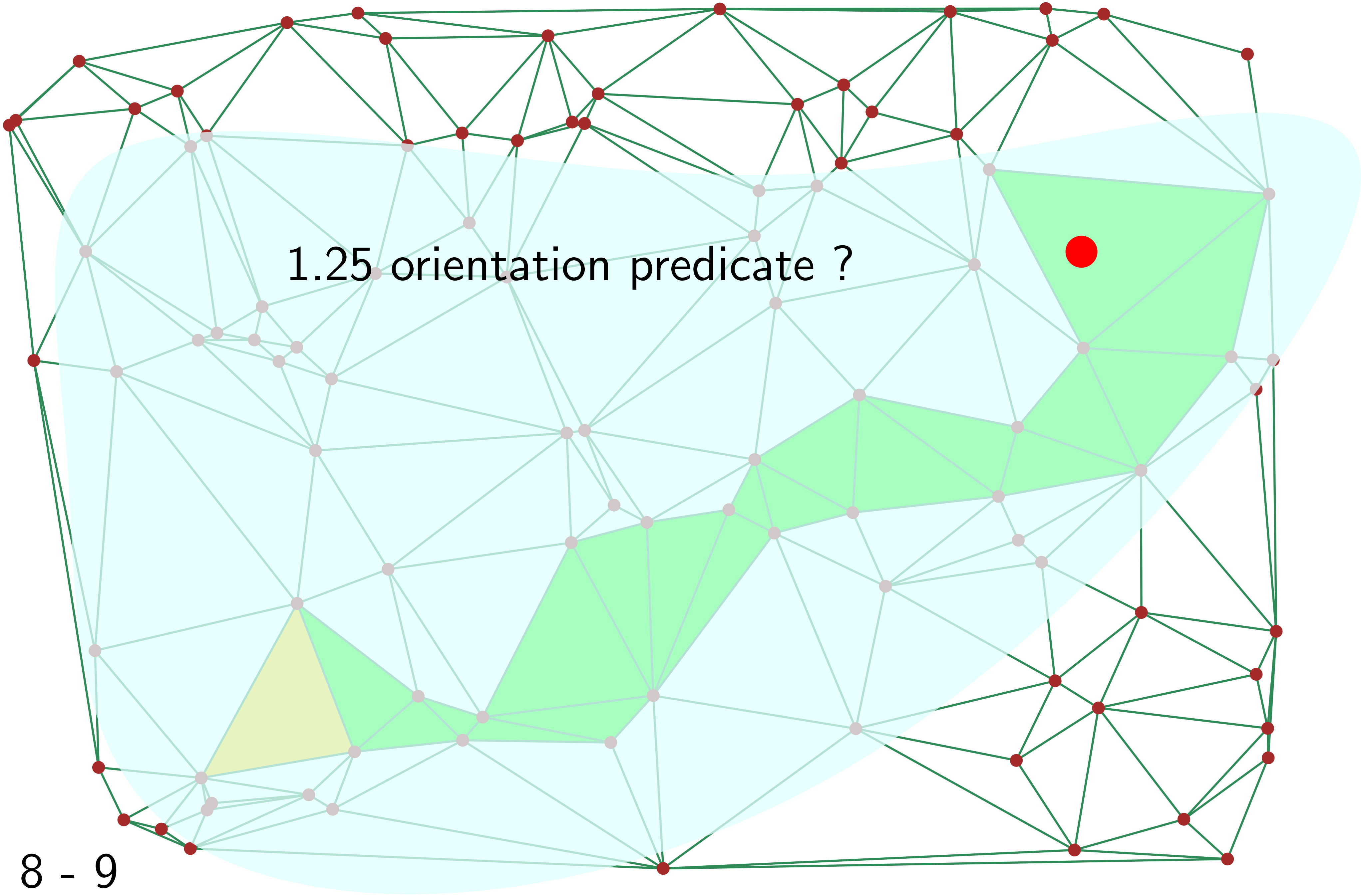
Triangle with one exit

1.5 orientation predicate

One predicate
if this neighbor tried first

Two predicates
if this neighbor tried first

visibility walk



1.25 orientation predicate ?

Visibility vs straight walk

Visibility vs straight walk 2D and 3D

fewer predicates per crossed edge

similar number of crossed edges

experimental / theoretical

Visibility vs straight walk

Speed improvement ?

Visibility vs straight walk

Speed improvement ?

Walk in Delaunay 1 Mpoints

Straight: 324 μs

Visibility: 285 μs

3D: 97 μs

Visibility vs straight walk

Speed improvement ?

Walk in Delaunay 1 Mpoints

Straight: 324 μs

Visibility: 285 μs

3D: 97 μs

Much easier to code

no degeneracies to handle!

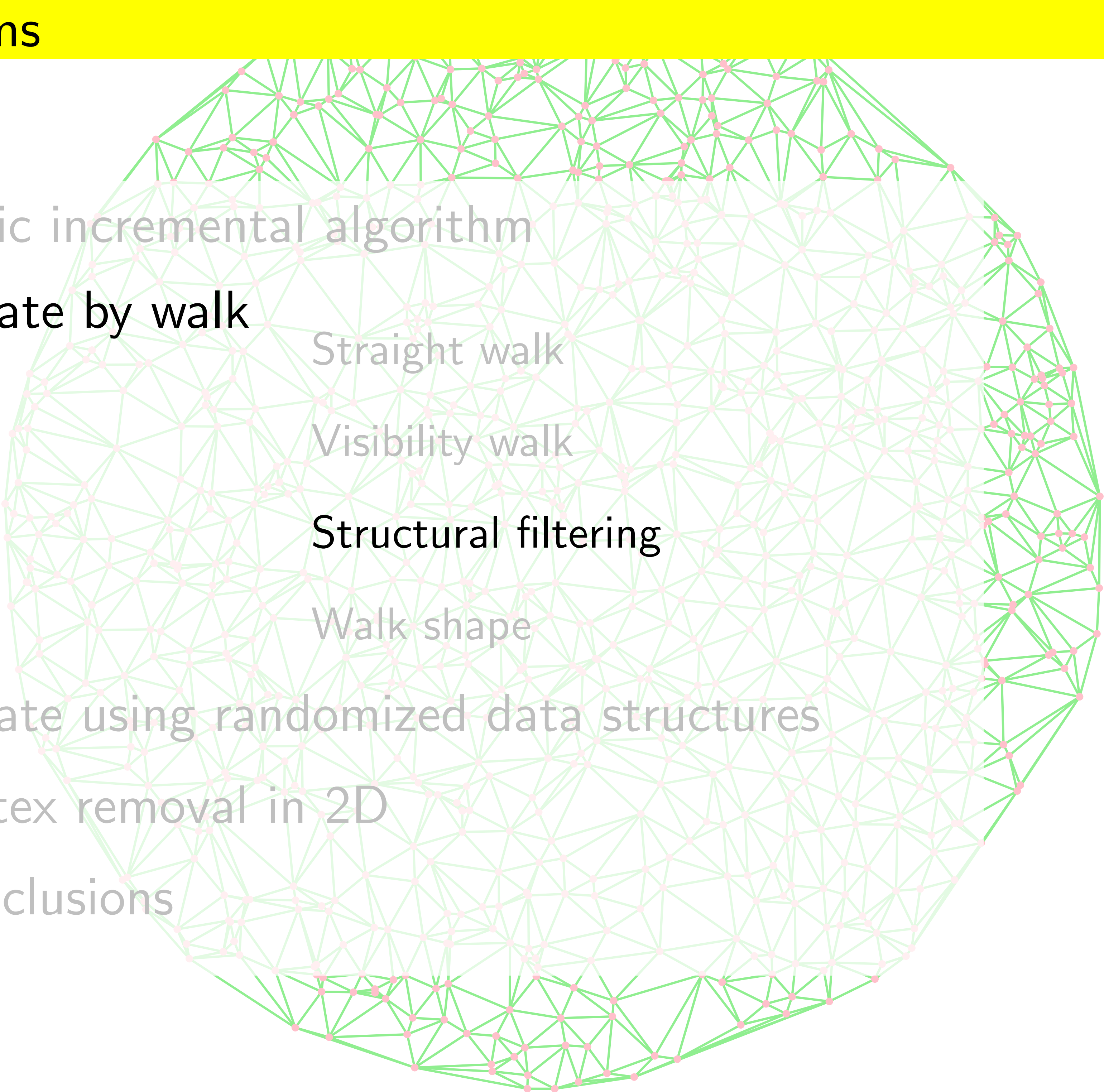
```
#include
   <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Triangulation_3.h>

#include <iostream> #include <fstream>
#include <cassert>
#include <list>     #include <vector>

typedef
   CGAL::Exact_predicates_inexact_constructions_kernel K;

typedef CGAL::Triangulation_3<K> Triangulation;

typedef Triangulation::Cell_handle Cell_handle;
typedef Triangulation::Vertex_handle Vertex_handle;
typedef Triangulation::Locate_type Locate_type;
typedef Triangulation::Point Point;
```

```
int main()
{
   std::list<Point> L;
   L.push_front(Point(0,0,0));
   L.push_front(Point(1,0,0));
   L.push_front(Point(0,1,0));
   Triangulation T(L.begin(), L.end());
   int n = T.number_of_vertices();

   std::vector<Point> V(3);
   V[0] = Point(0,0,1);
   V[1] = Point(1,1,1);
   V[2] = Point(2,2,2);
   n = n + T.insert(V.begin(), V.end());

   assert( n == 6 );
   assert( T.is_valid() );
```

```
Locate_type lt;
int li, lj;
Point p(0,0,0);
Cell_handle c = T.locate(p, lt, li, lj);
assert( lt == Triangulation::VERTEX );
assert( c->vertex(li)->point() == p );

Vertex_handle v = c->vertex( (li+1)&3 );
Cell_handle nc = c->neighbor(li);
int nli;
assert( nc->has_vertex( v, nli ) );
```

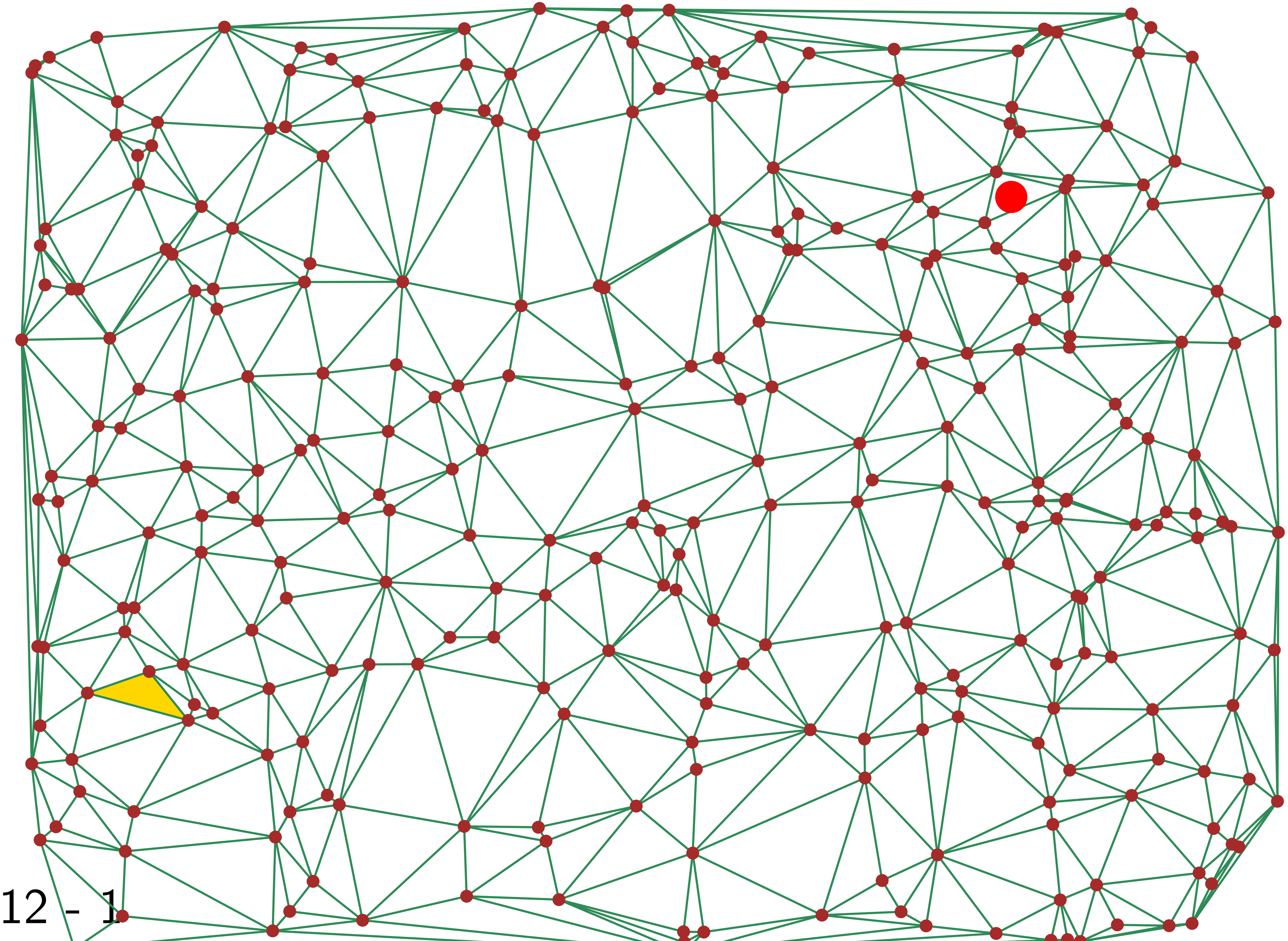
```
  std::ofstream oFileT("output",std::ios::out);
  oFileT << T;
  Triangulation T1;
  std::ifstream iFileT("output",std::ios::in);
  iFileT >> T1;
  assert( T1.is_valid() );
  assert(T1.number_of_vertices() == T.number_of_vertices());
  assert( T1.number_of_cells() == T.number_of_cells() );
  return 0;
}
```

Algorithms

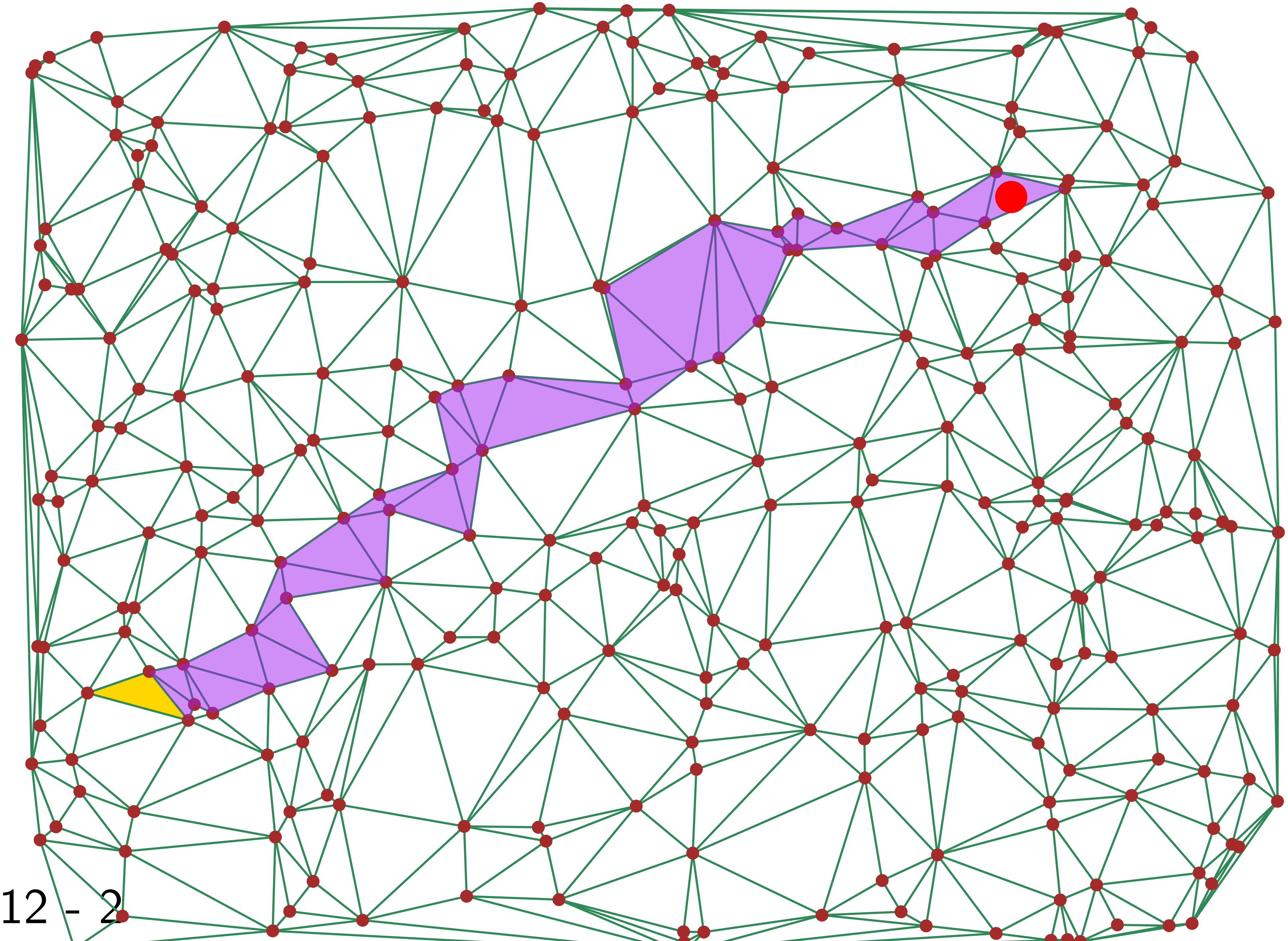


- Basic incremental algorithm
- Locate by walk
  - Straight walk
  - Visibility walk
  - Structural filtering
  - Walk shape
- Locate using randomized data structures
- Vertex removal in 2D
- Conclusions

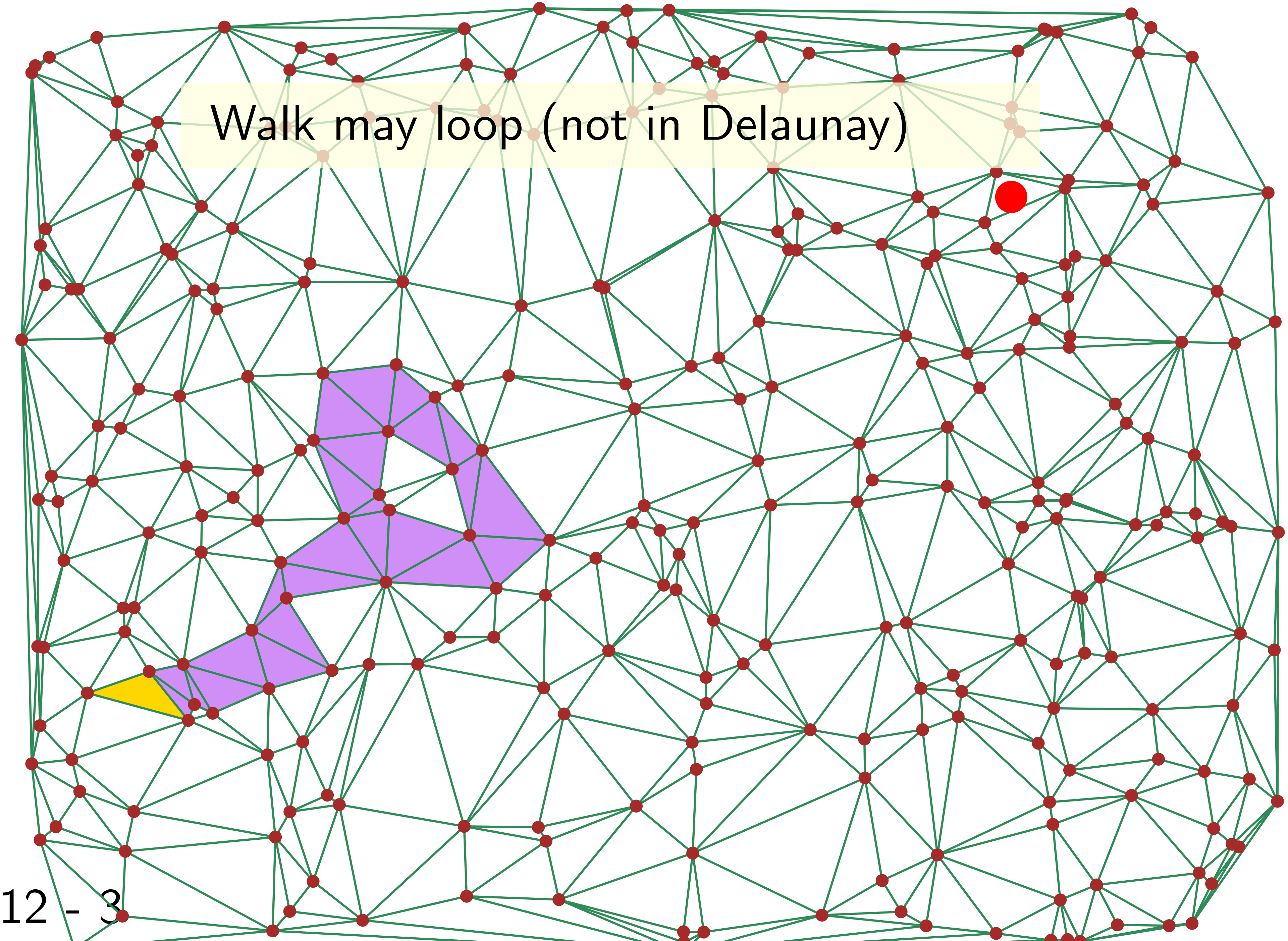
visibility walk - structural filtering



visibility walk - structural filtering

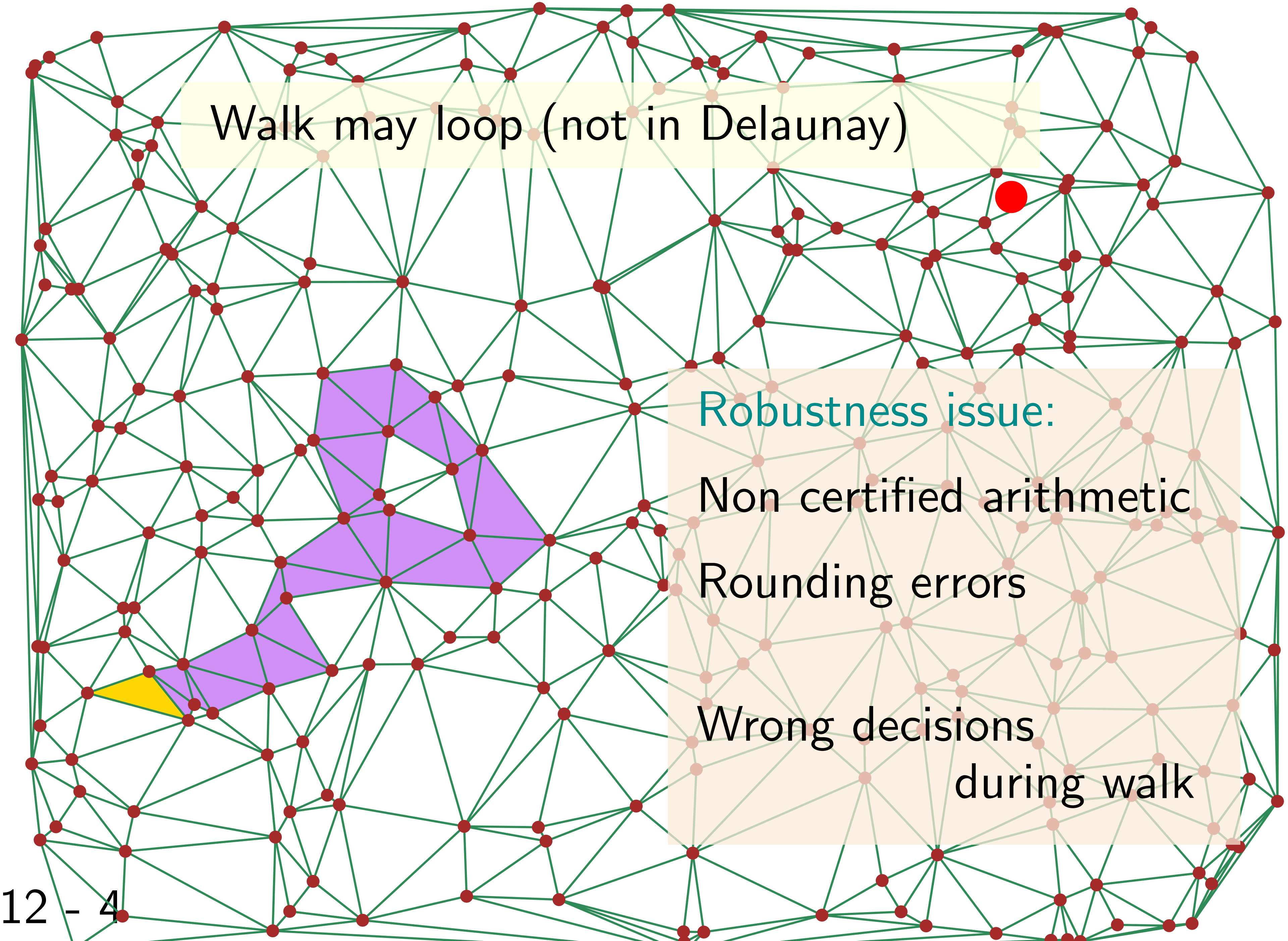


visibility walk - structural filtering



Walk may loop (not in Delaunay)

visibility walk - structural filtering



Walk may loop (not in Delaunay)

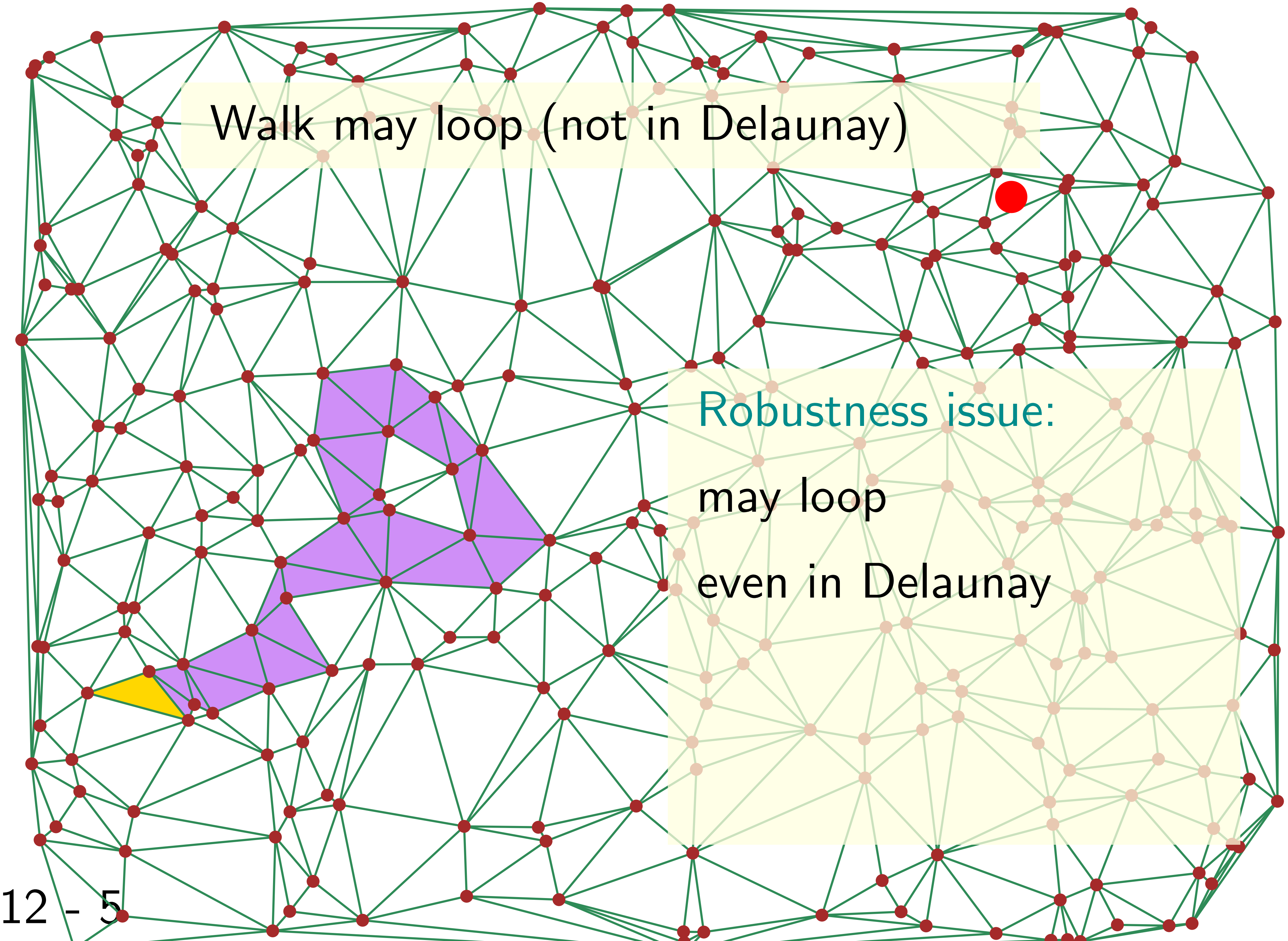
Robustness issue:

Non certified arithmetic

Rounding errors

Wrong decisions
during walk

visibility walk - structural filtering



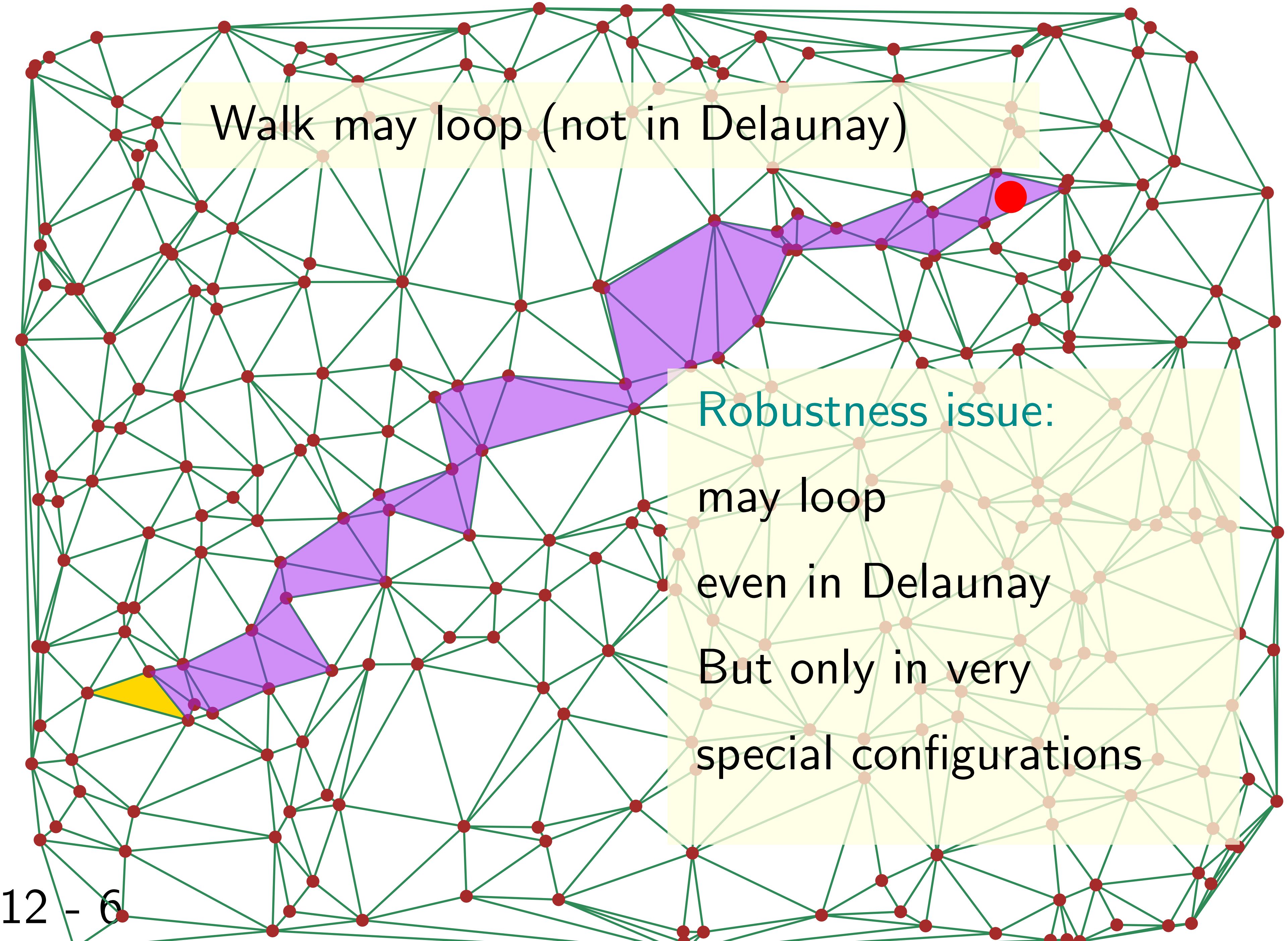
Walk may loop (not in Delaunay)

Robustness issue:

may loop

even in Delaunay

visibility walk - structural filtering



Walk may loop (not in Delaunay)

Robustness issue:

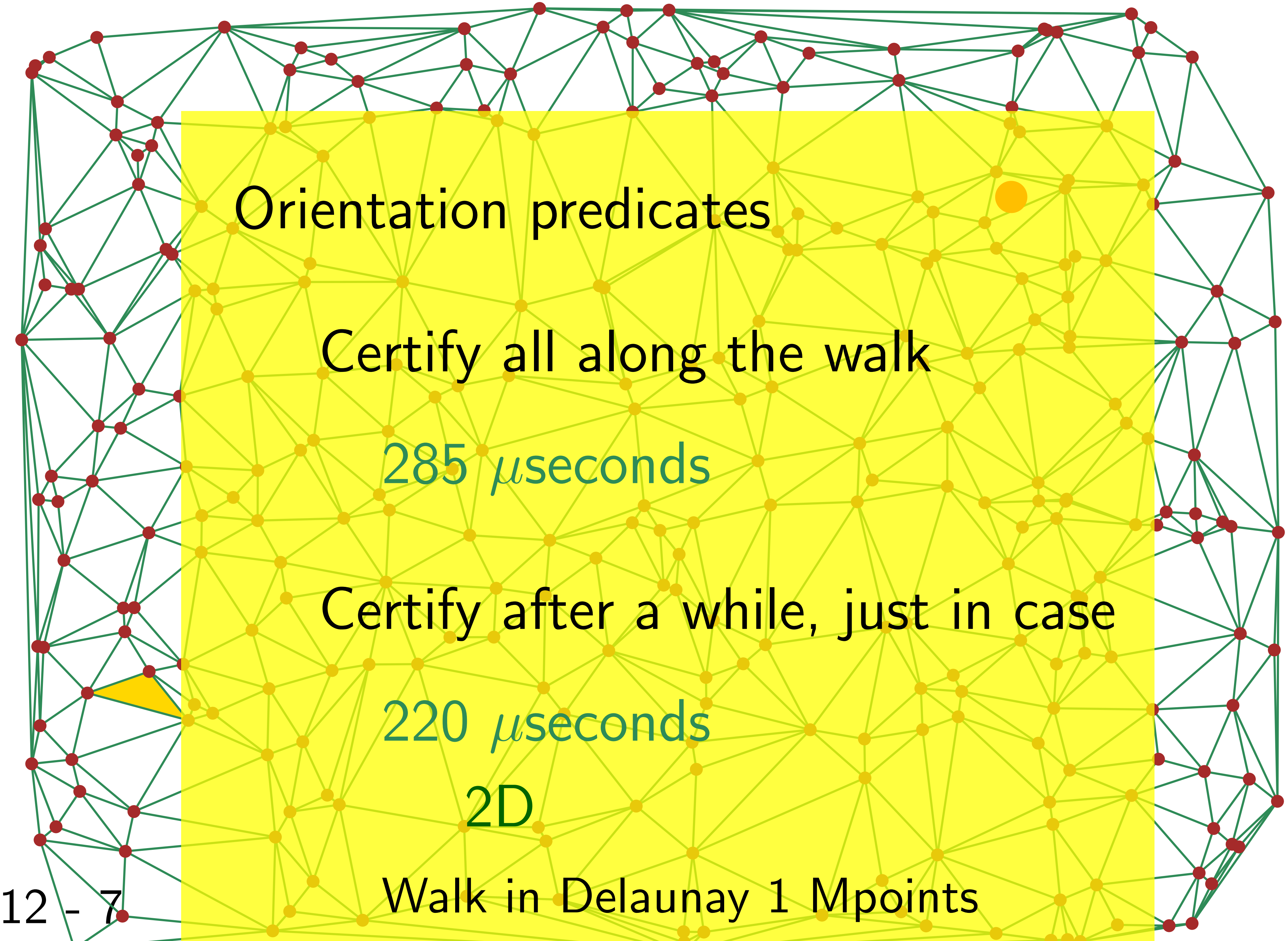
may loop

even in Delaunay

But only in very

special configurations

visibility walk - structural filtering



Orientation predicates

Certify all along the walk

285 μseconds

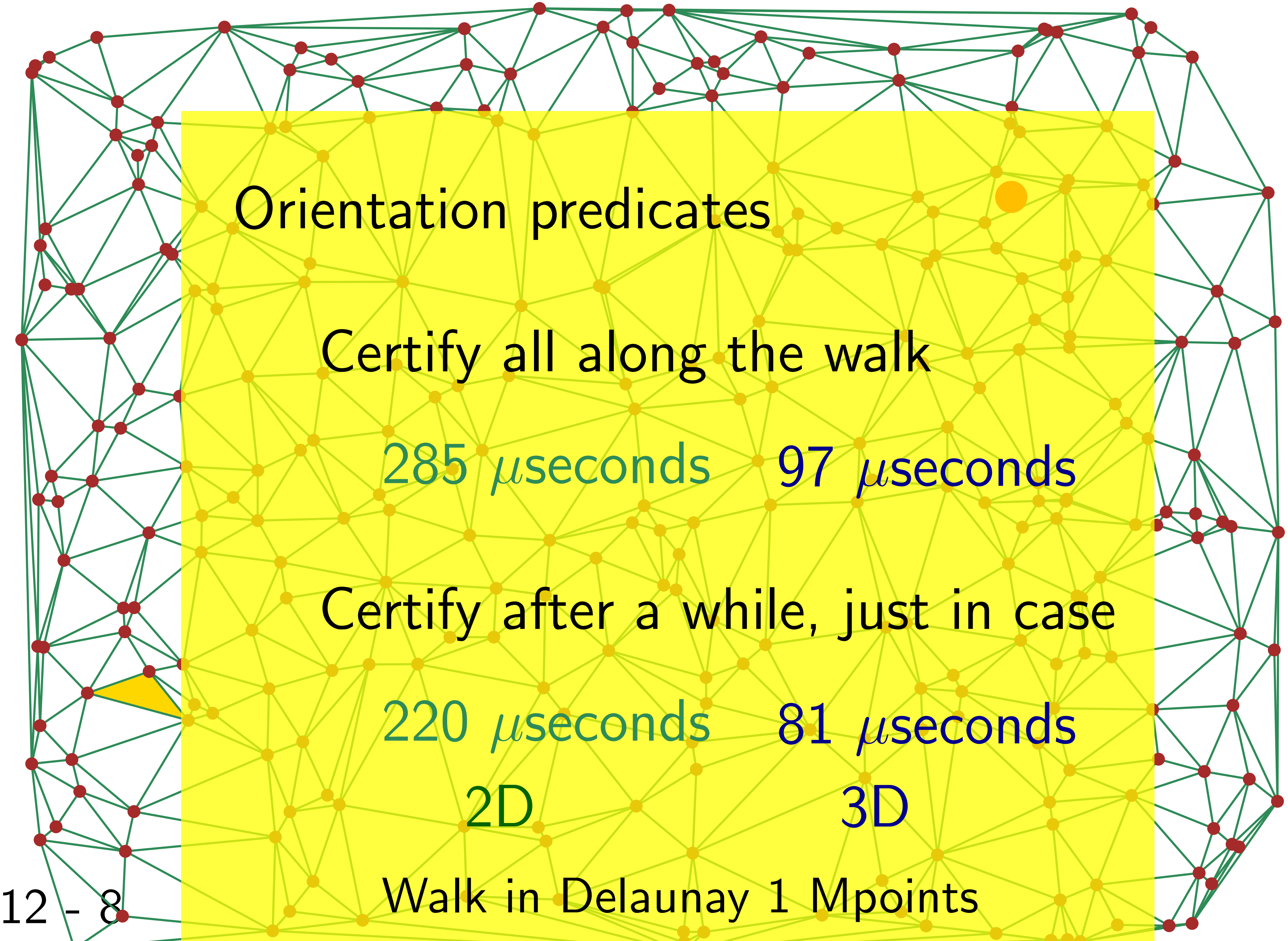
Certify after a while, just in case

220 μseconds

2D

Walk in Delaunay 1 Mpoints

visibility walk - structural filtering



Orientation predicates

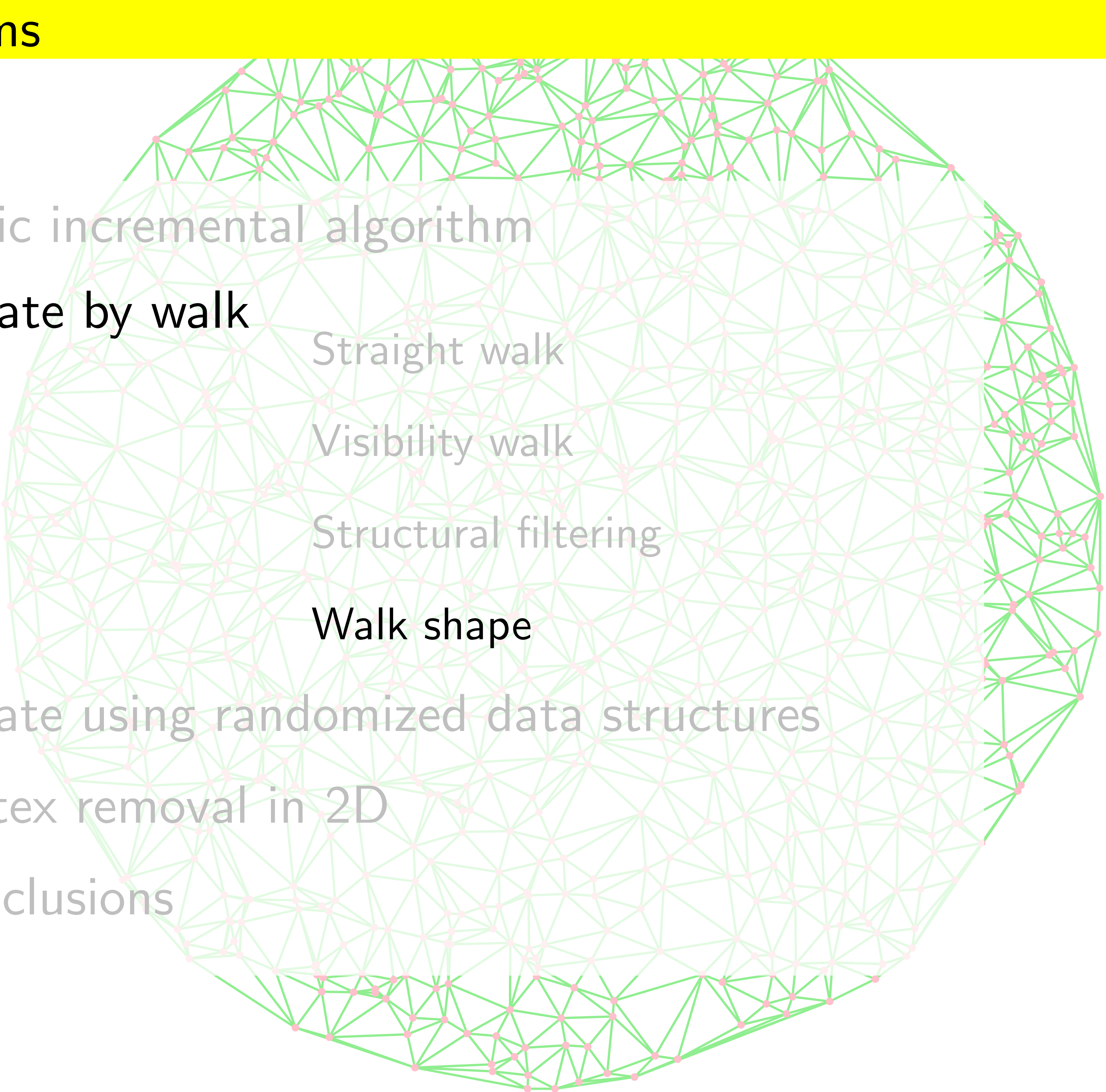
Certify all along the walk

2D	3D
285 μseconds	97 μseconds
220 μseconds	81 μseconds

Certify after a while, just in case

Walk in Delaunay 1 Mpoints

Algorithms



Basic incremental algorithm

Locate by walk

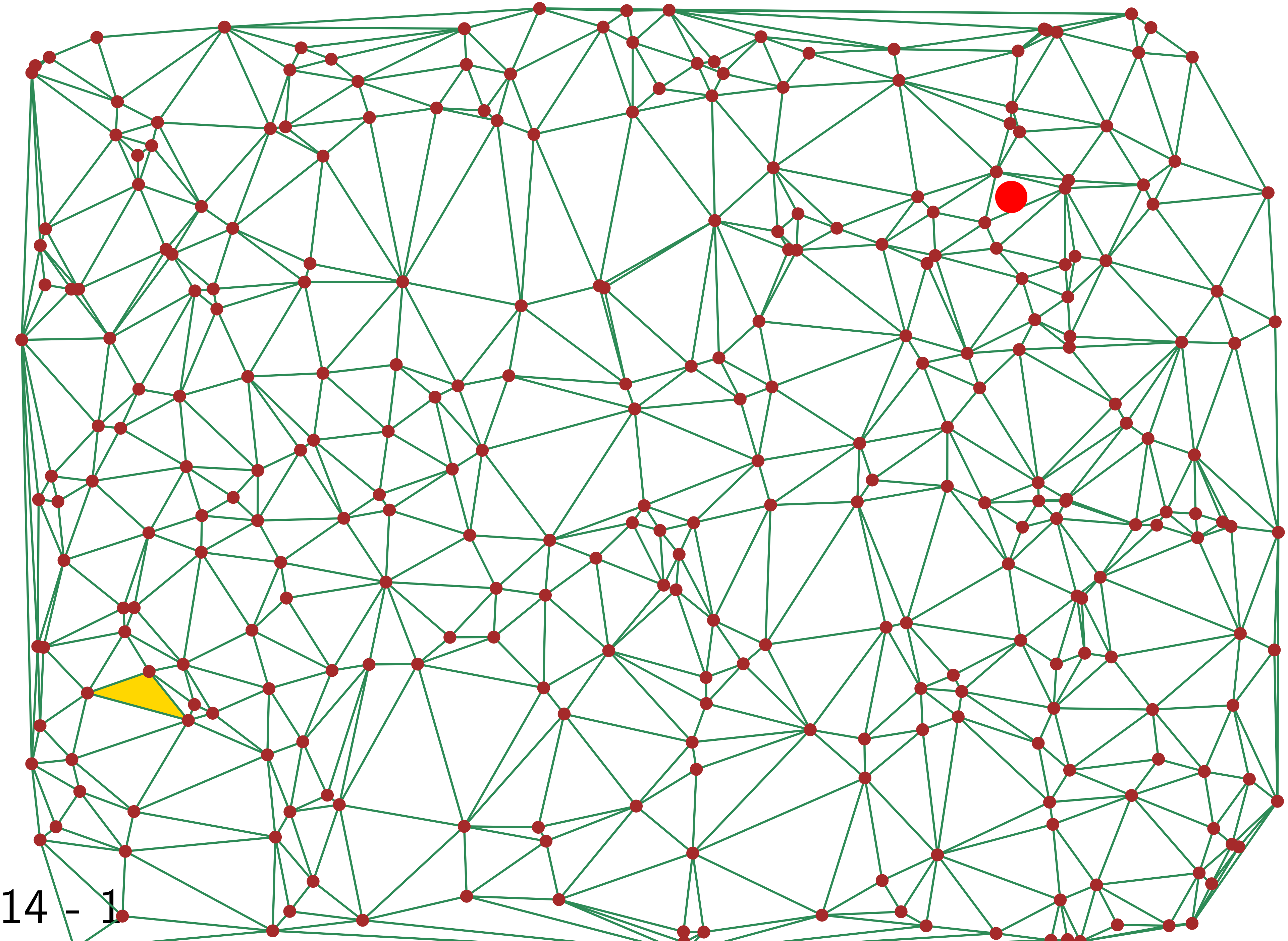
- Straight walk
- Visibility walk
- Structural filtering
- Walk shape

Locate using randomized data structures

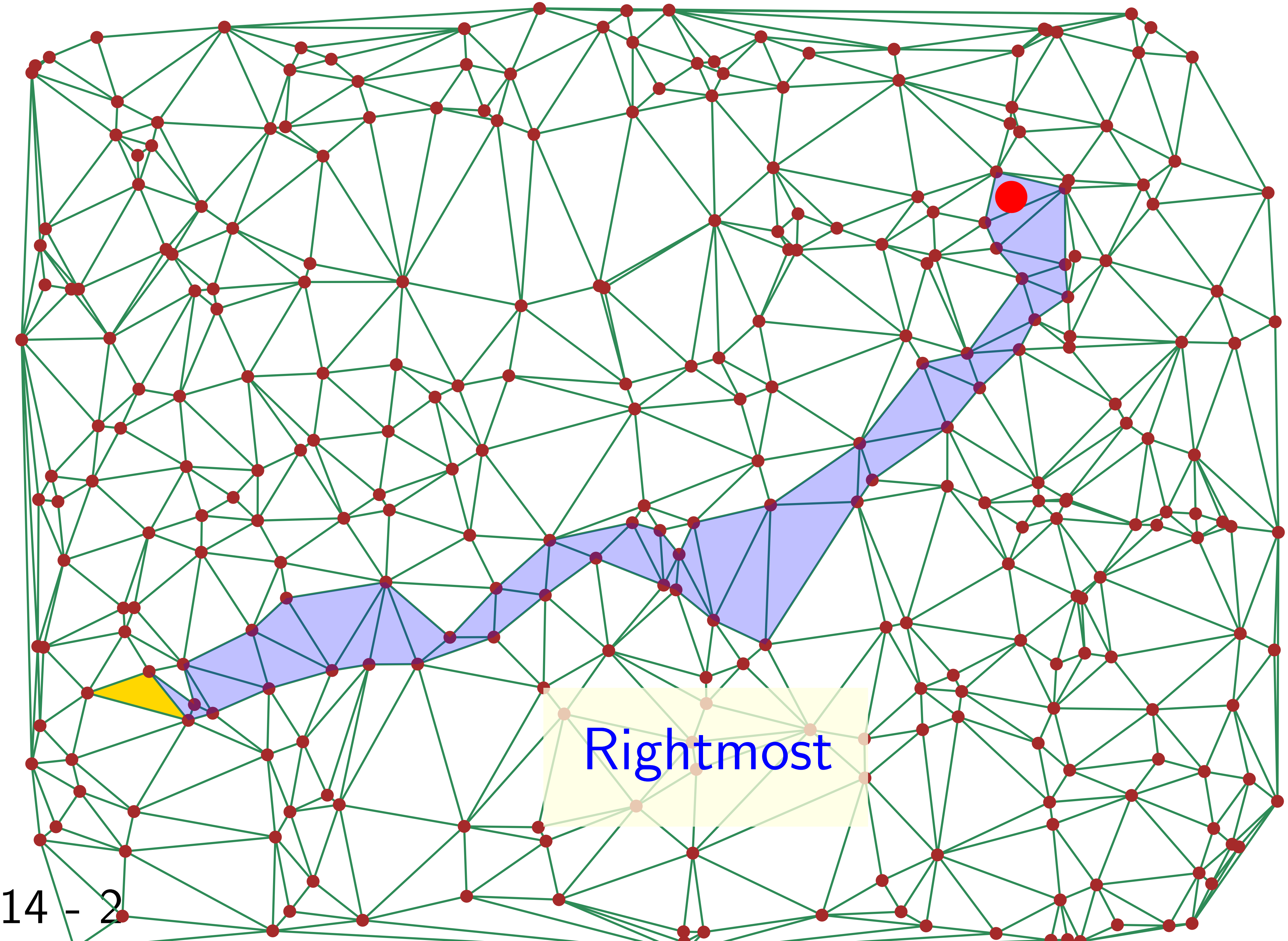
Vertex removal in 2D

Conclusions

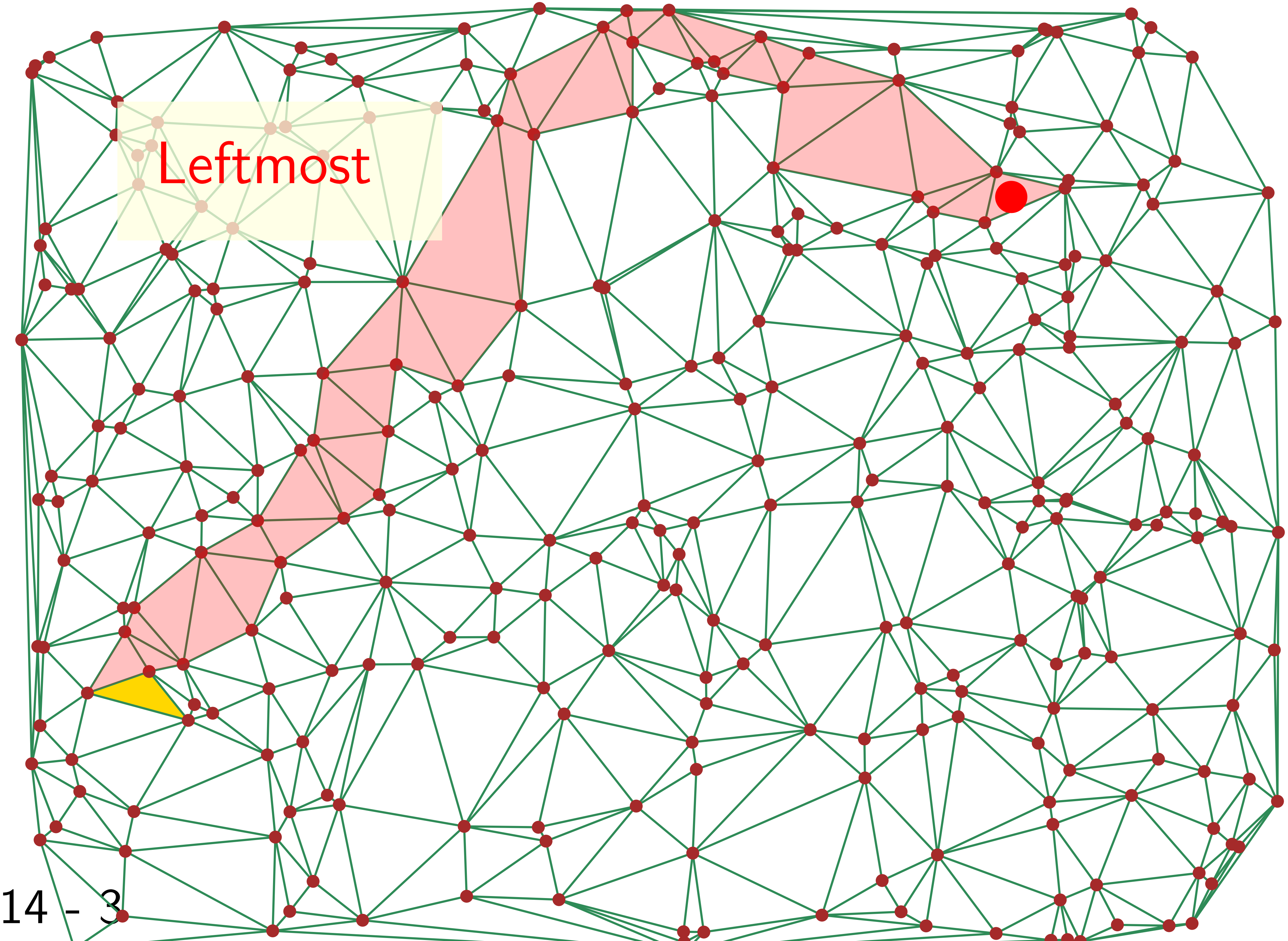
visibility walk - walk shape



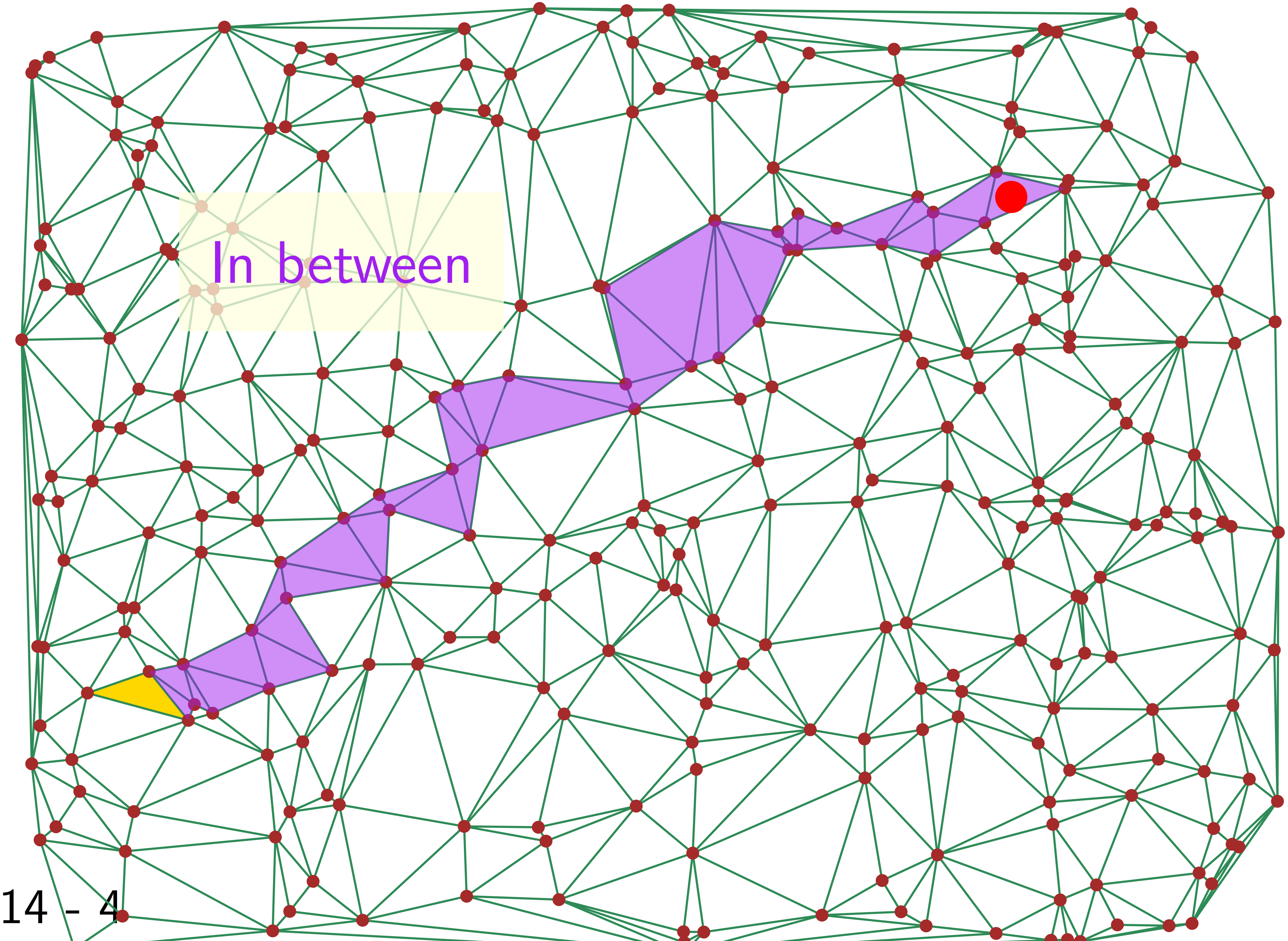
visibility walk - walk shape



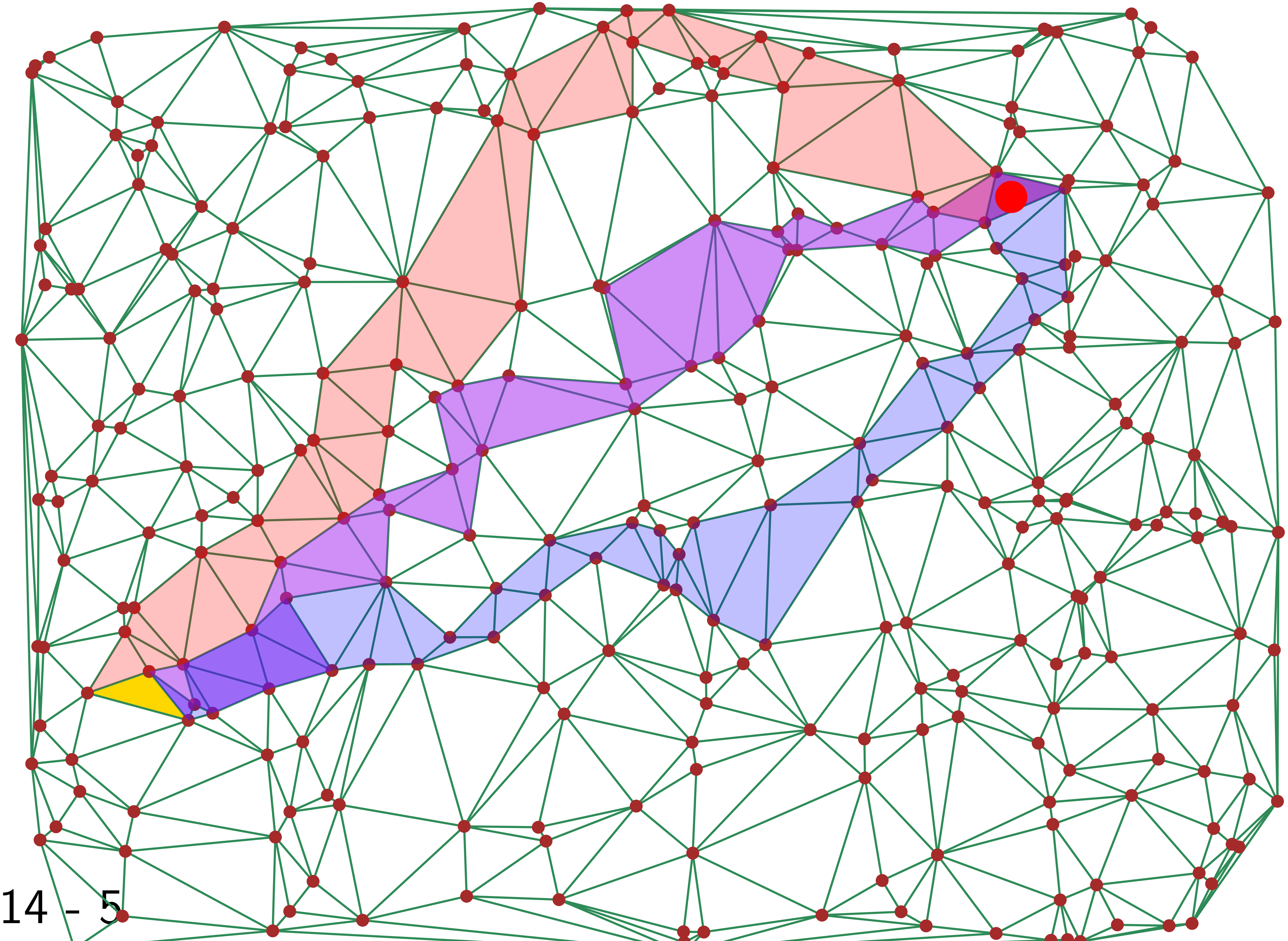
visibility walk - walk shape



visibility walk - walk shape



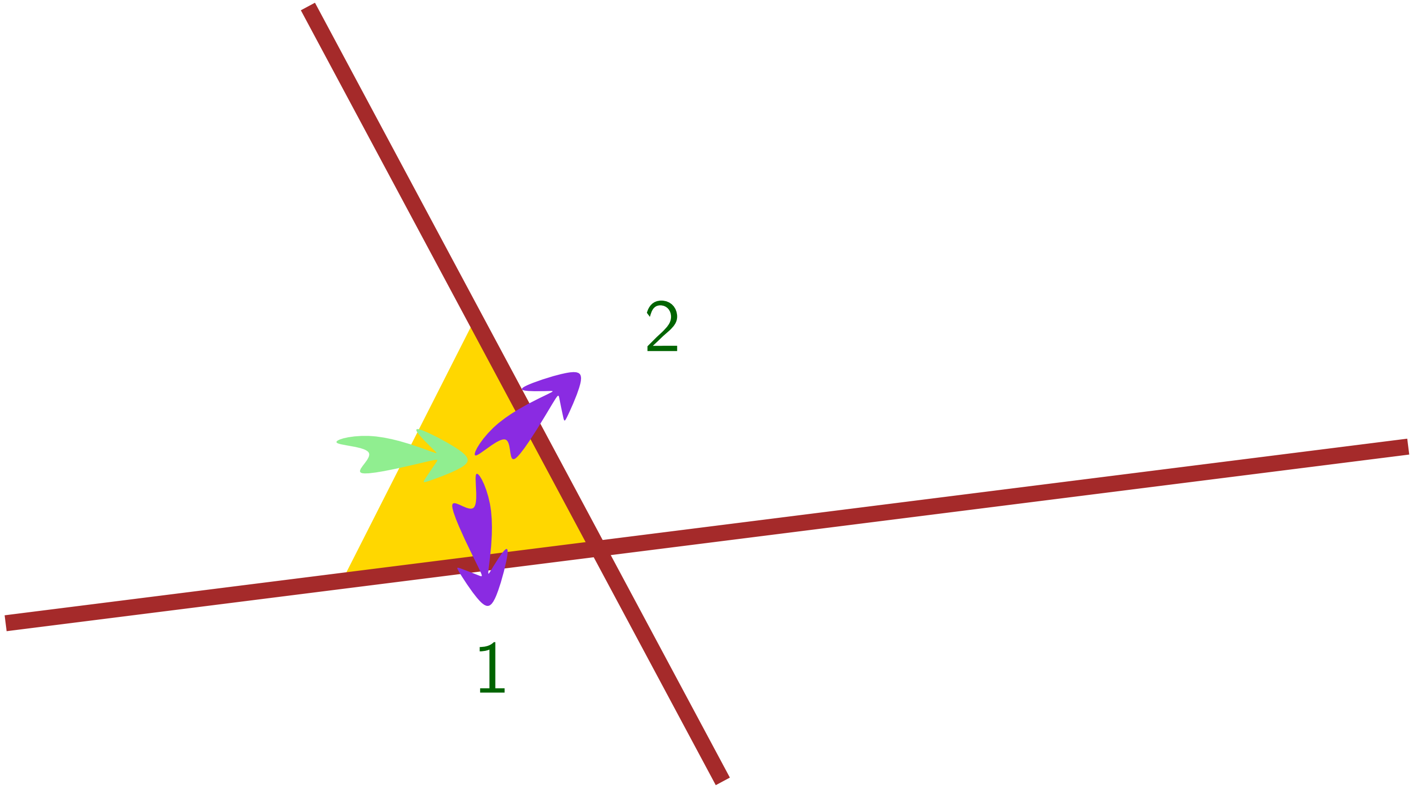
visibility walk - walk shape



visibility walk - walk shape

Turn counterclockwise from previous

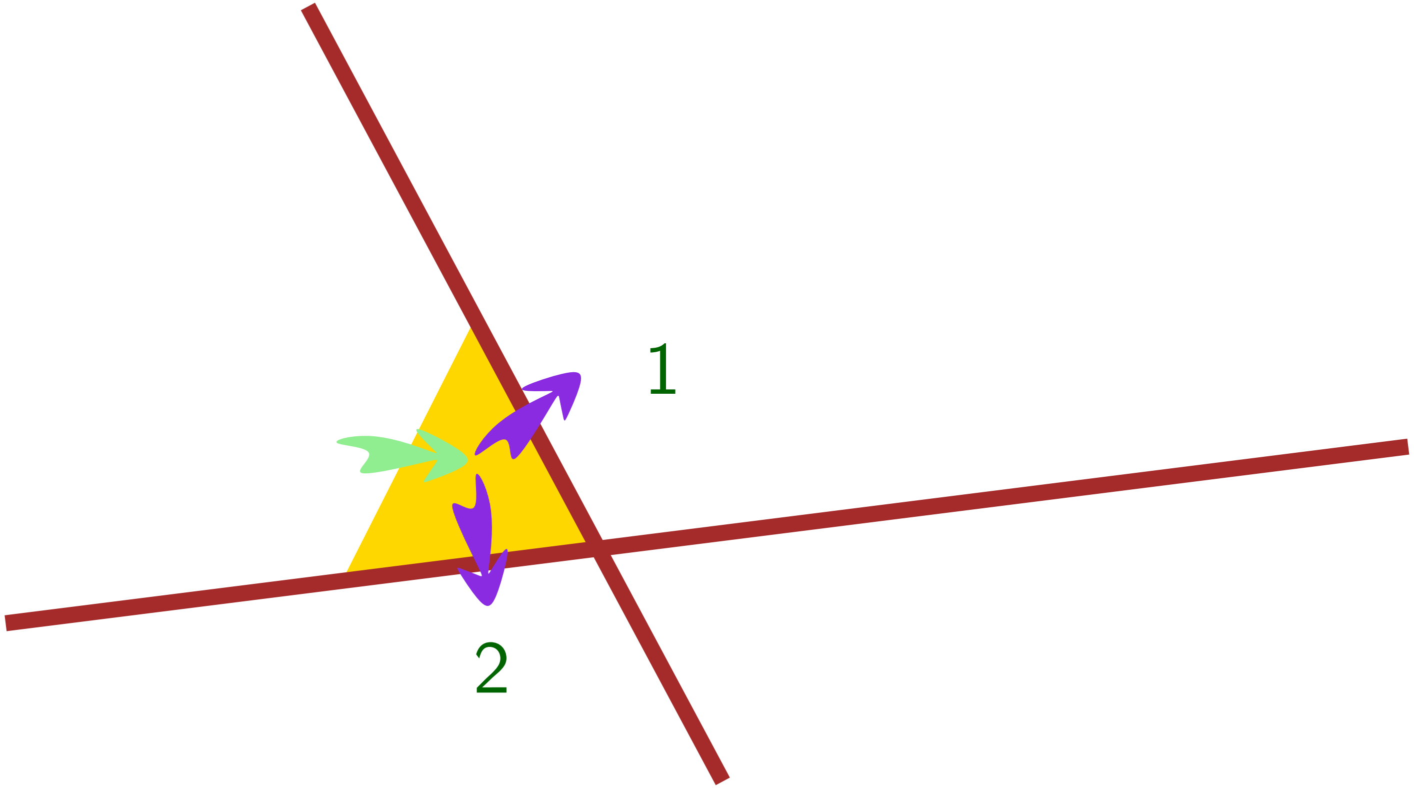
Rightmost



visibility walk - walk shape

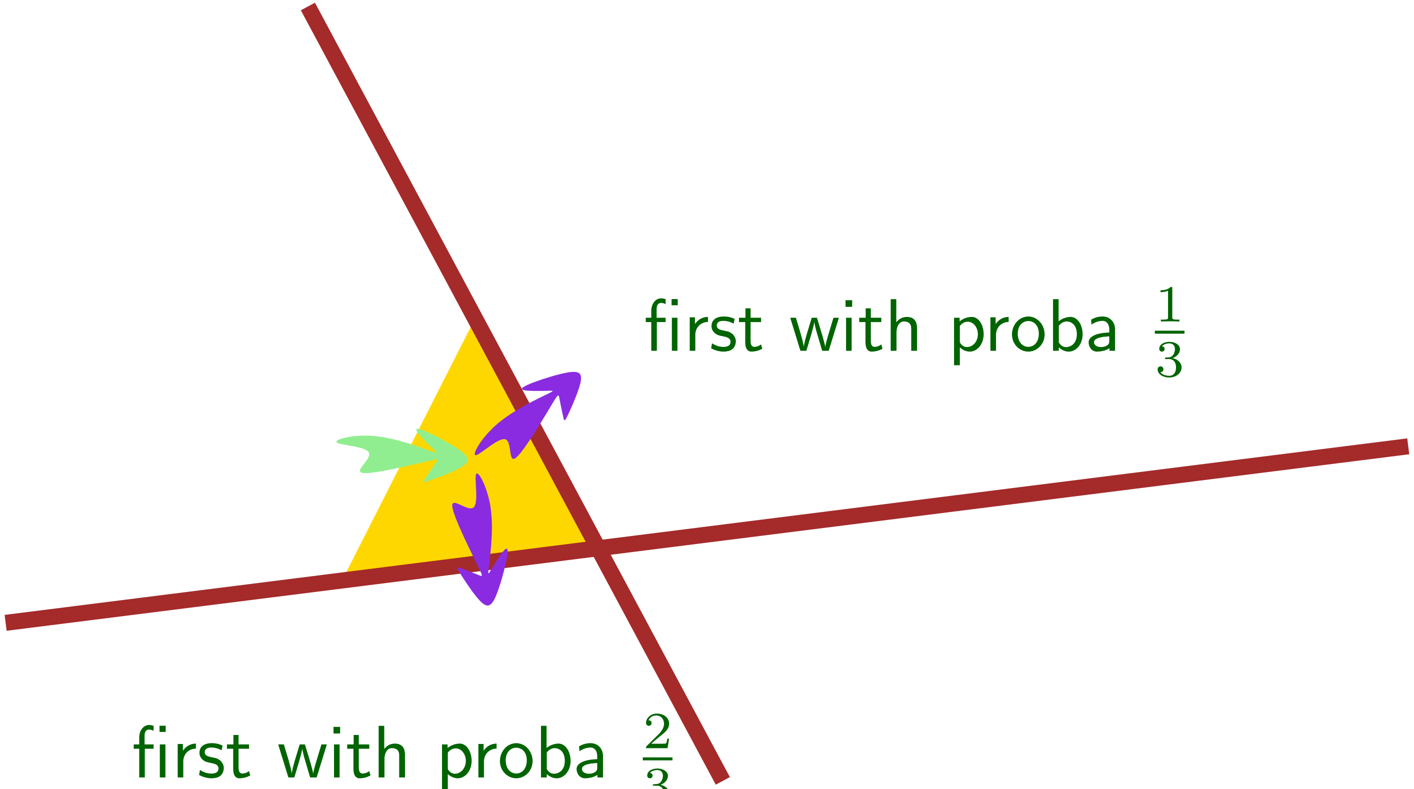
Turn clockwise from previous

Leftmost



visibility walk - walk shape

Balance left and right turns



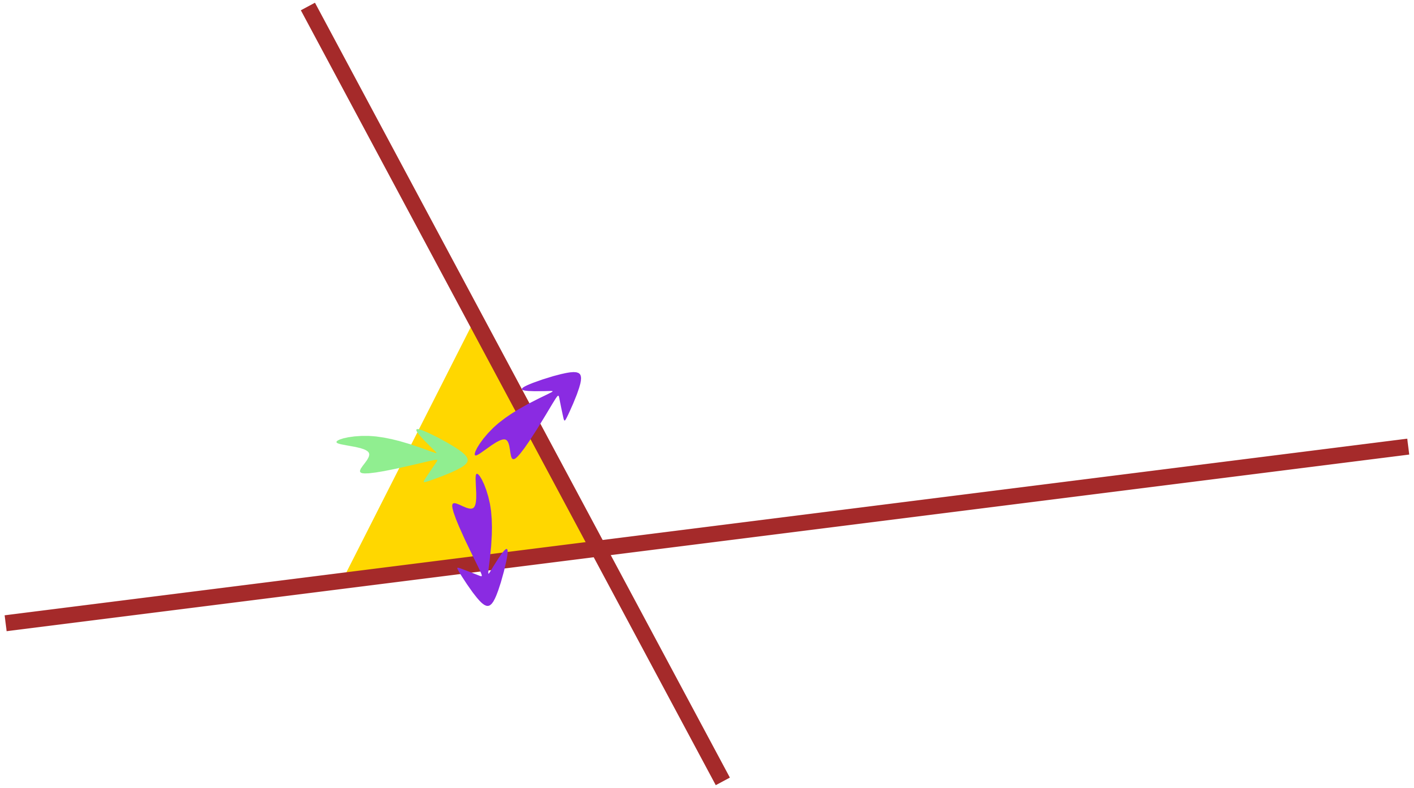
first with proba $\frac{1}{3}$

first with proba $\frac{2}{3}$

visibility walk - walk shape

Walk in Delaunay 1 Mpoints

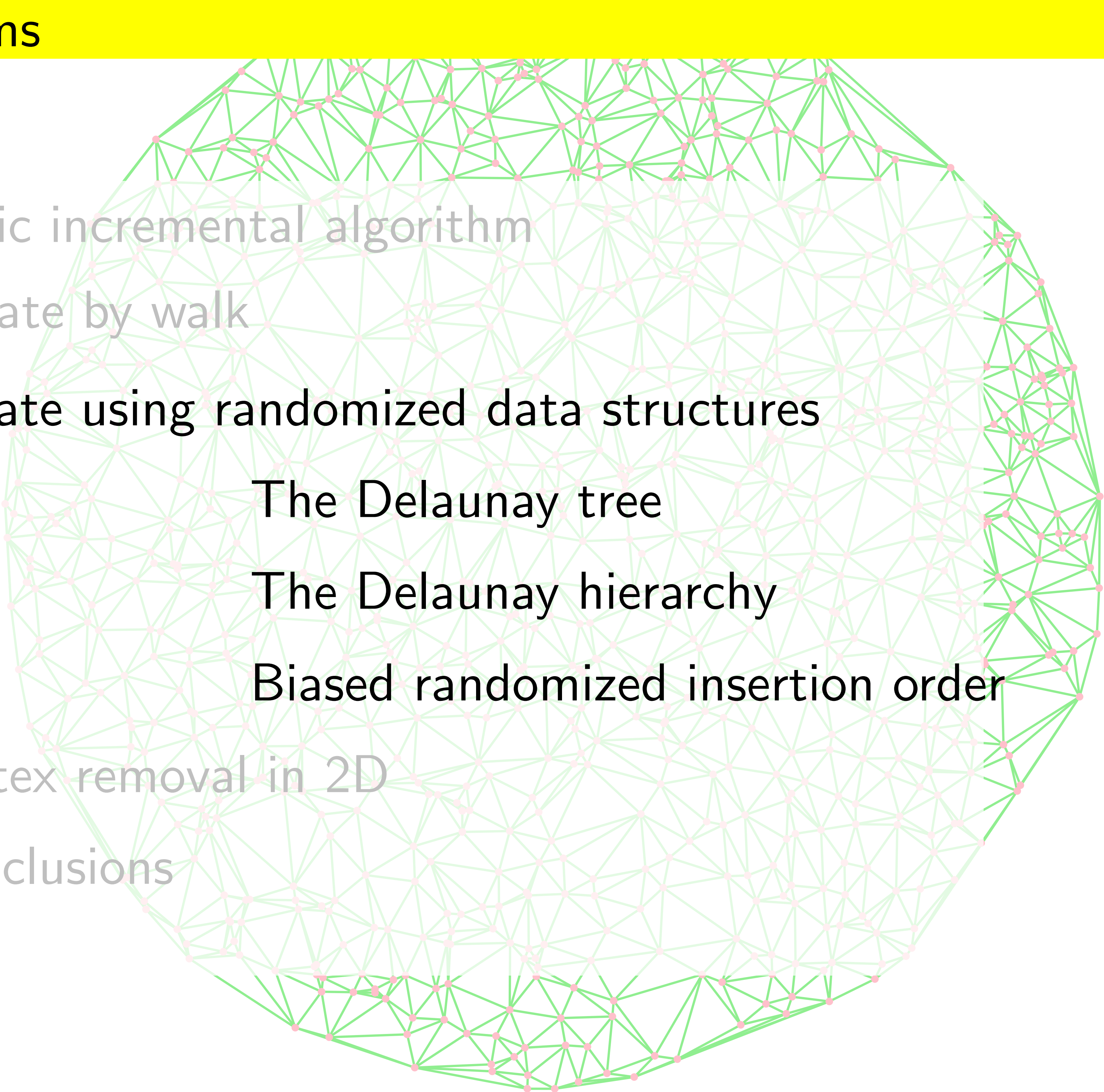
Leftmost	220 μseconds
Balanced	188 μseconds



Walk in Delaunay 1 Mpoints

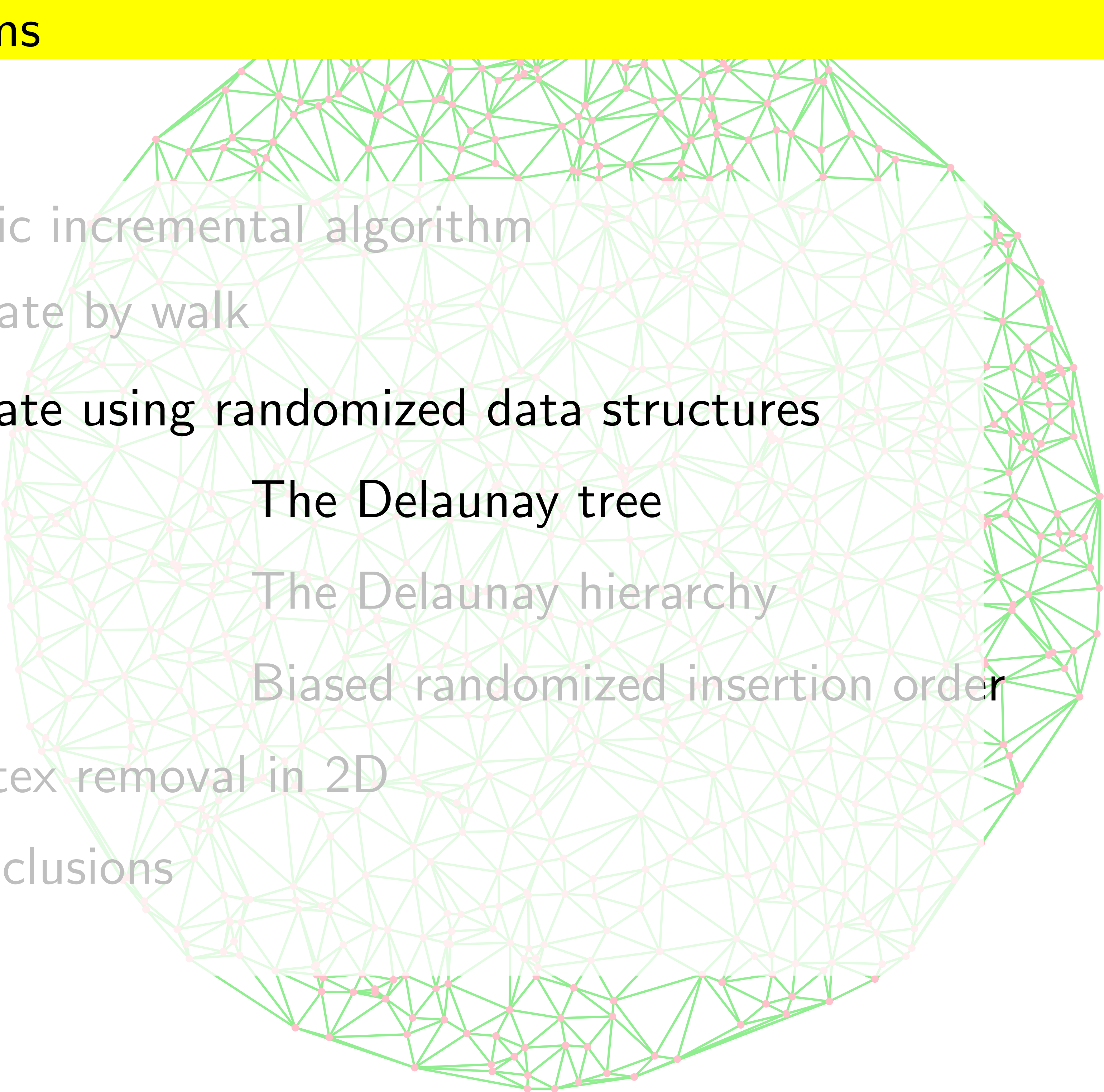
Straight walk	324 μseconds
Visibility walk	285 μseconds
Structural filtering	220 μseconds
Balanced walk	188 μseconds

Algorithms



- Basic incremental algorithm
- Locate by walk
- Locate using randomized data structures
  - The Delaunay tree
  - The Delaunay hierarchy
  - Biased randomized insertion order
- Vertex removal in 2D
- Conclusions

Algorithms



Basic incremental algorithm

Locate by walk

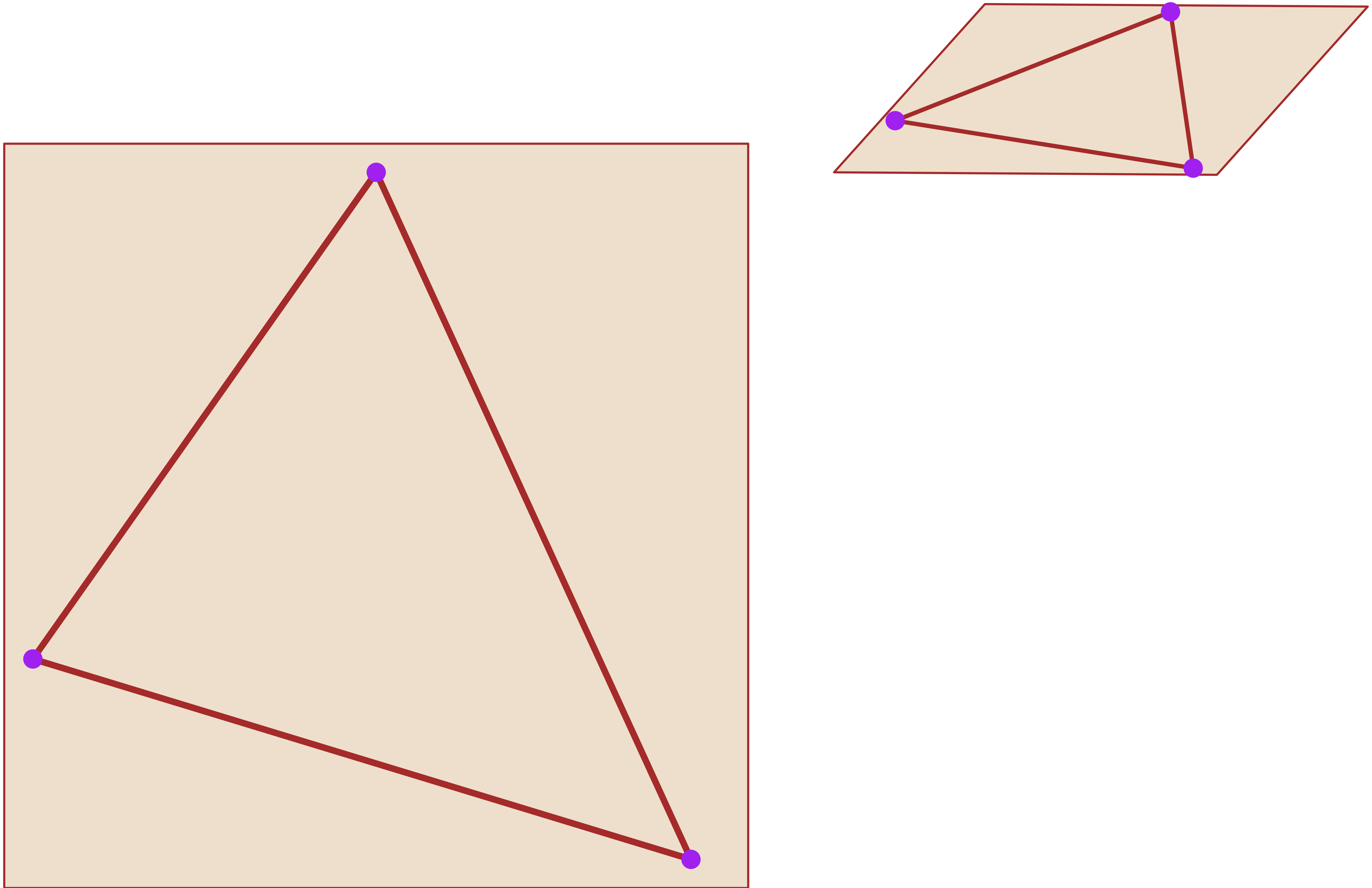
Locate using randomized data structures

- The Delaunay tree
- The Delaunay hierarchy
- Biased randomized insertion order

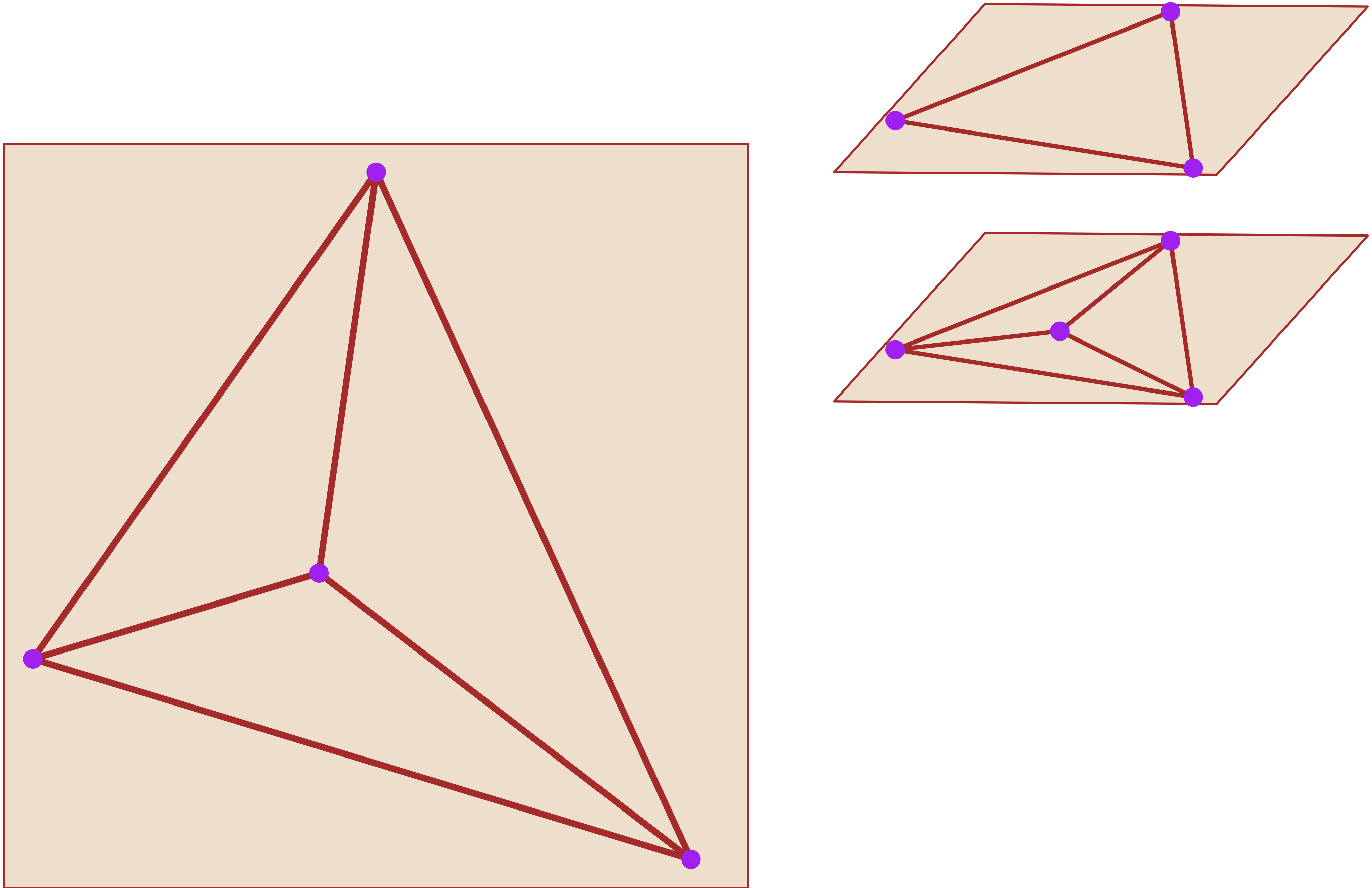
Vertex removal in 2D

Conclusions

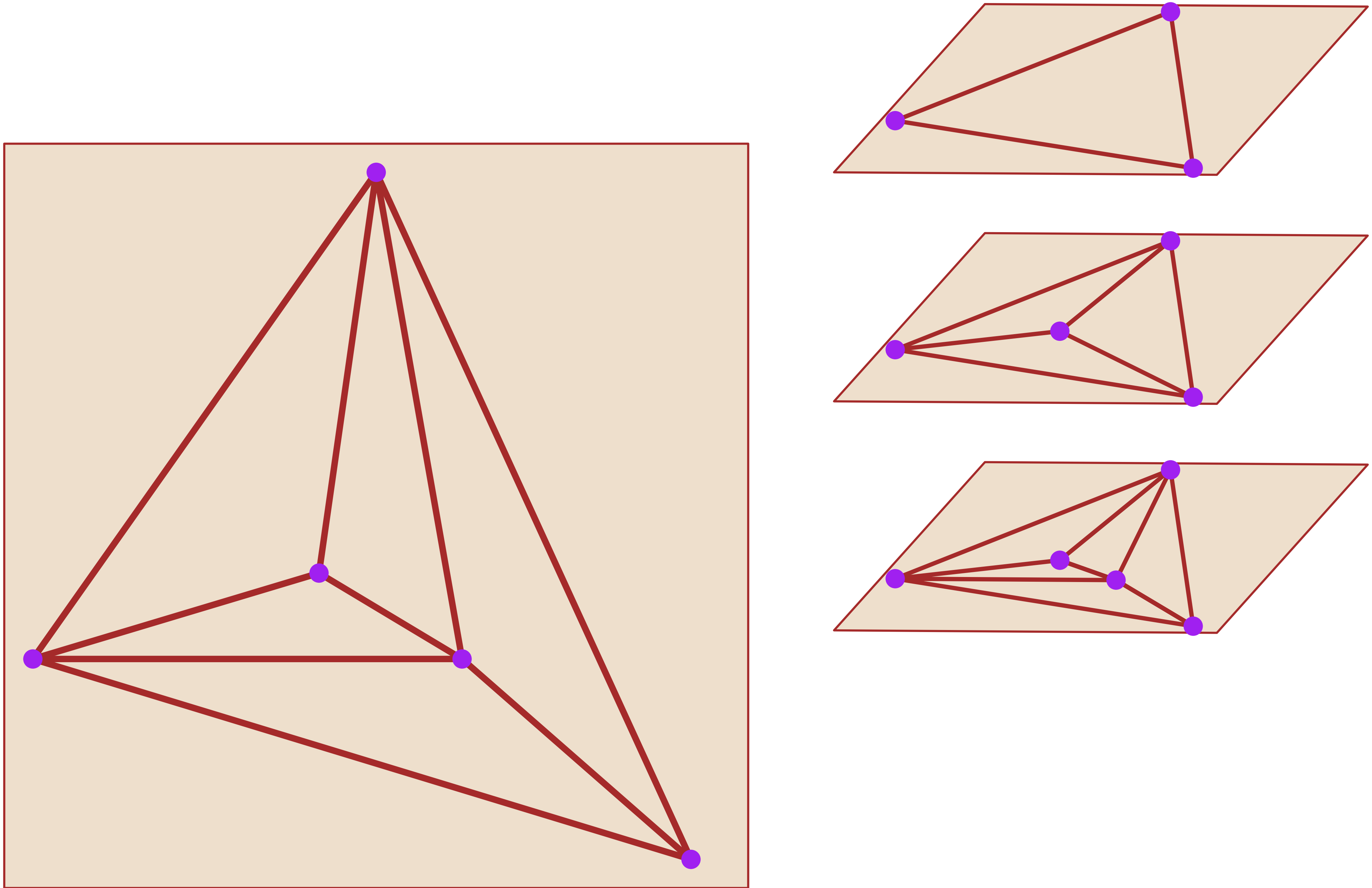
the Delaunay tree



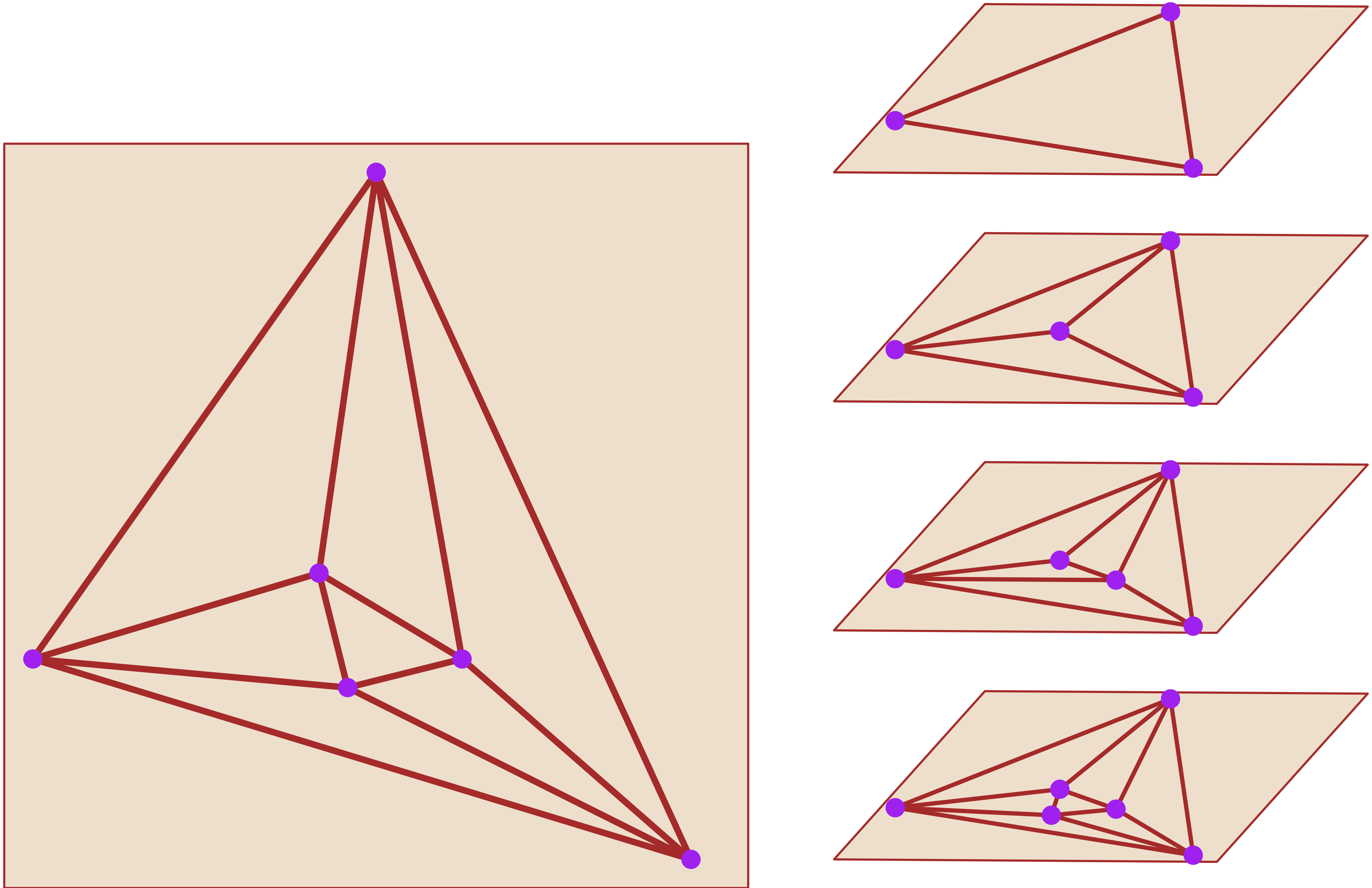
the Delaunay tree



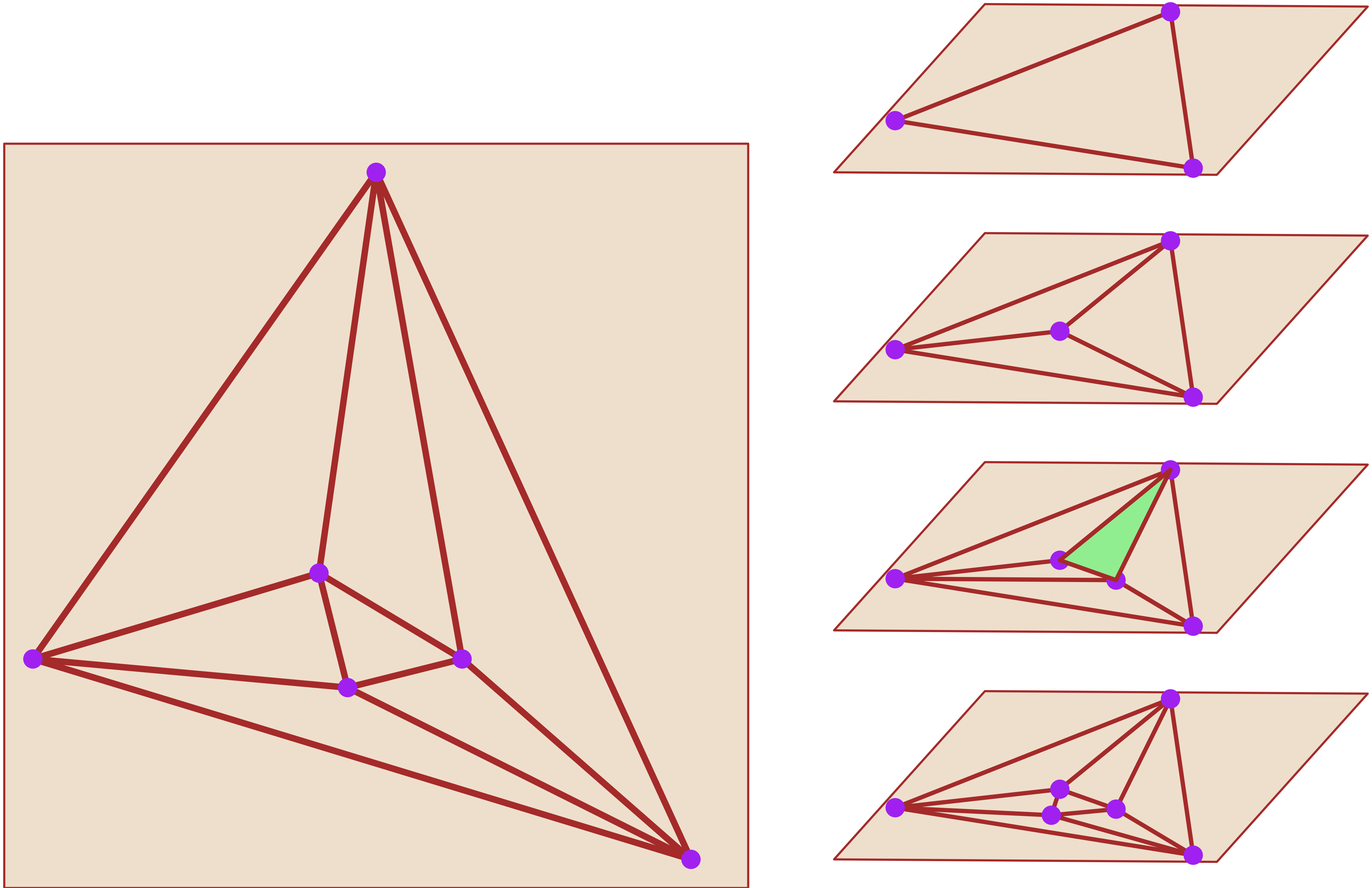
the Delaunay tree



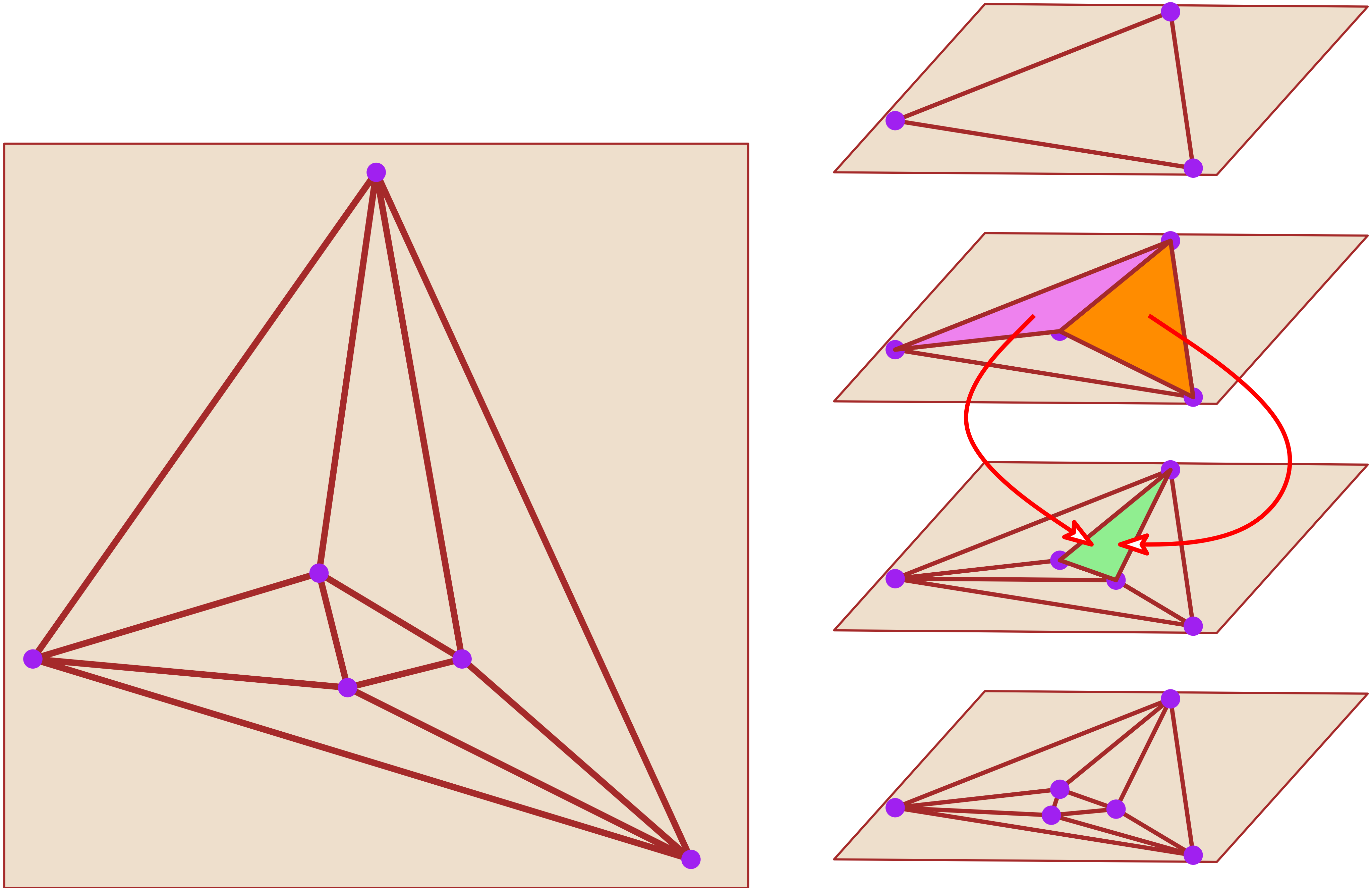
the Delaunay tree



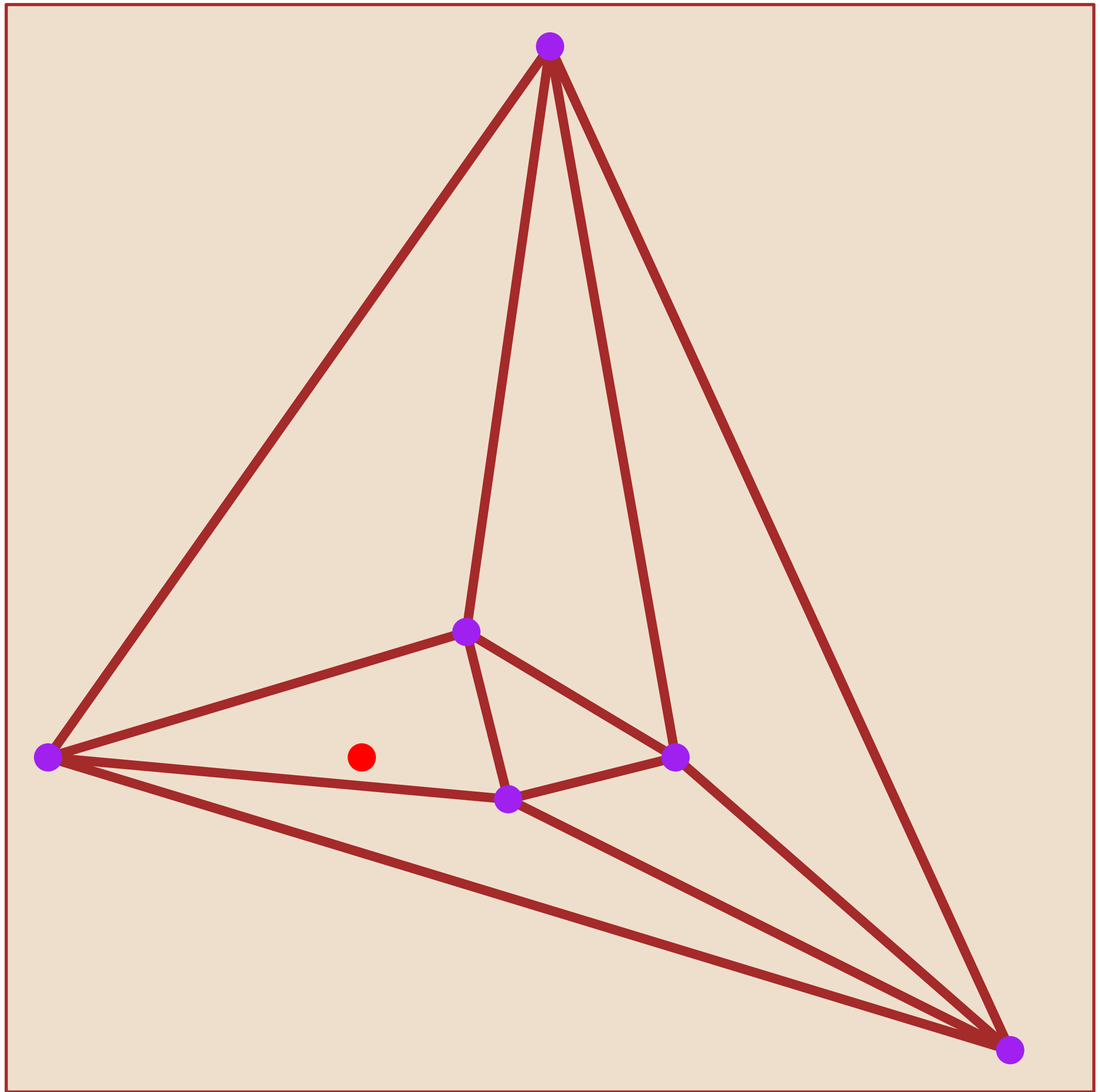
the Delaunay tree

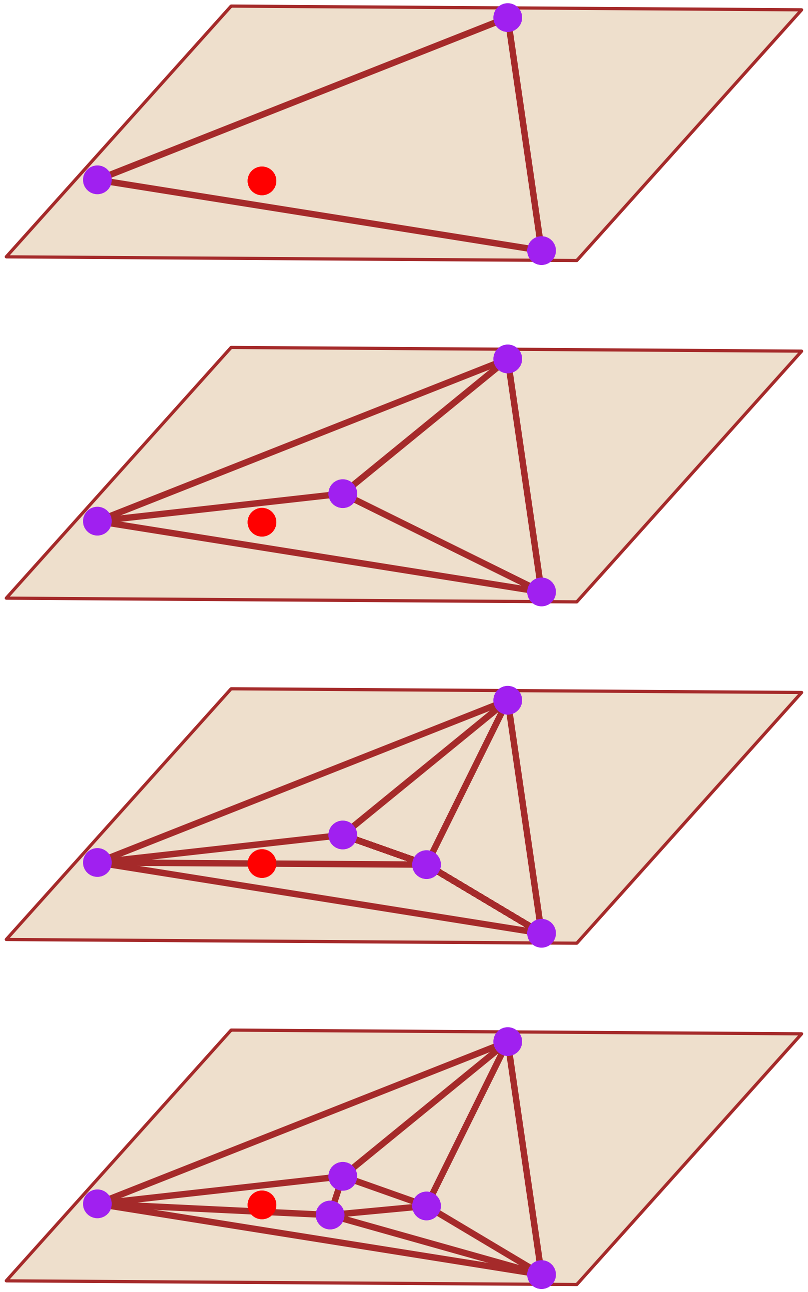


the Delaunay tree

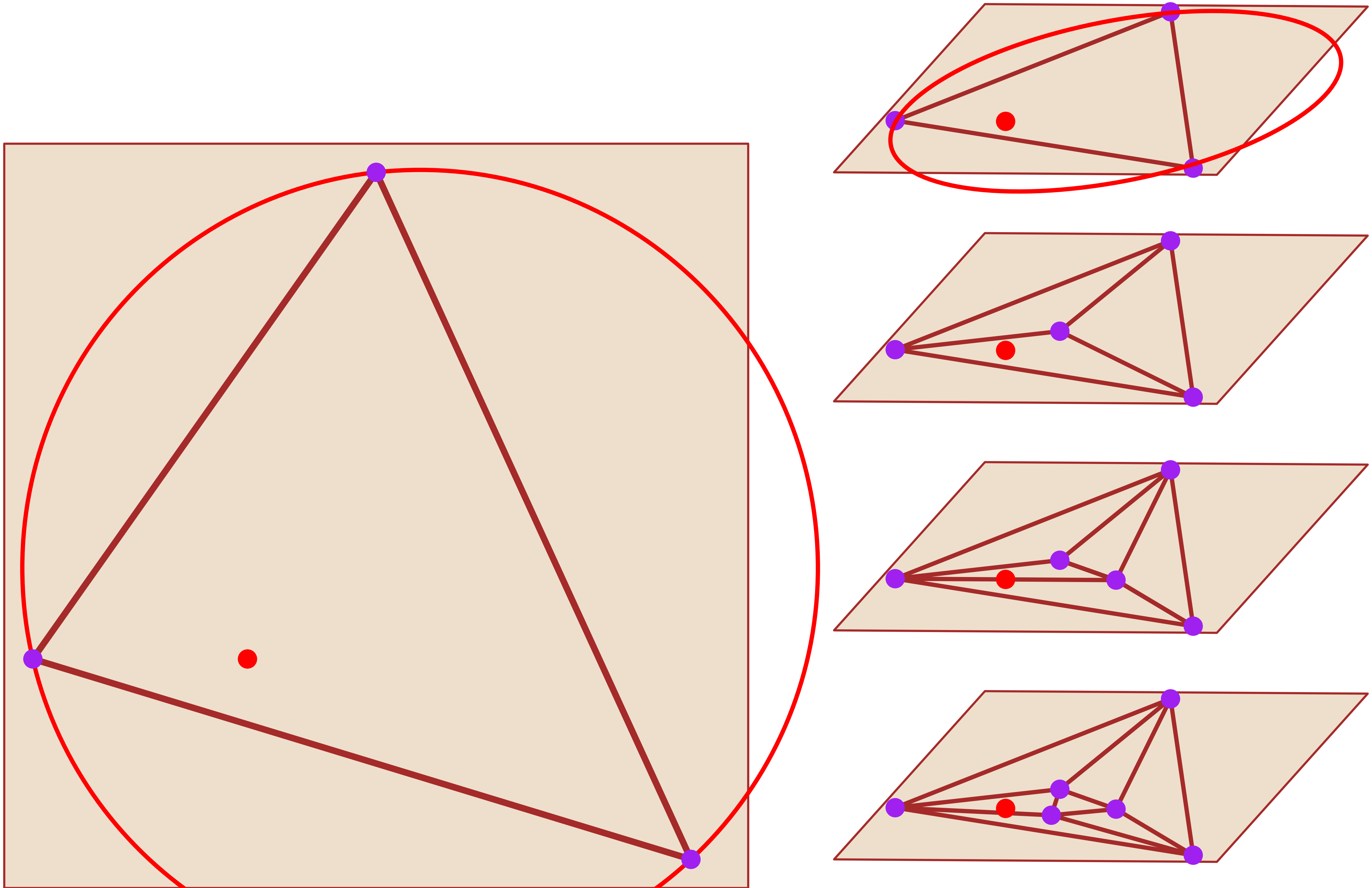


the Delaunay tree

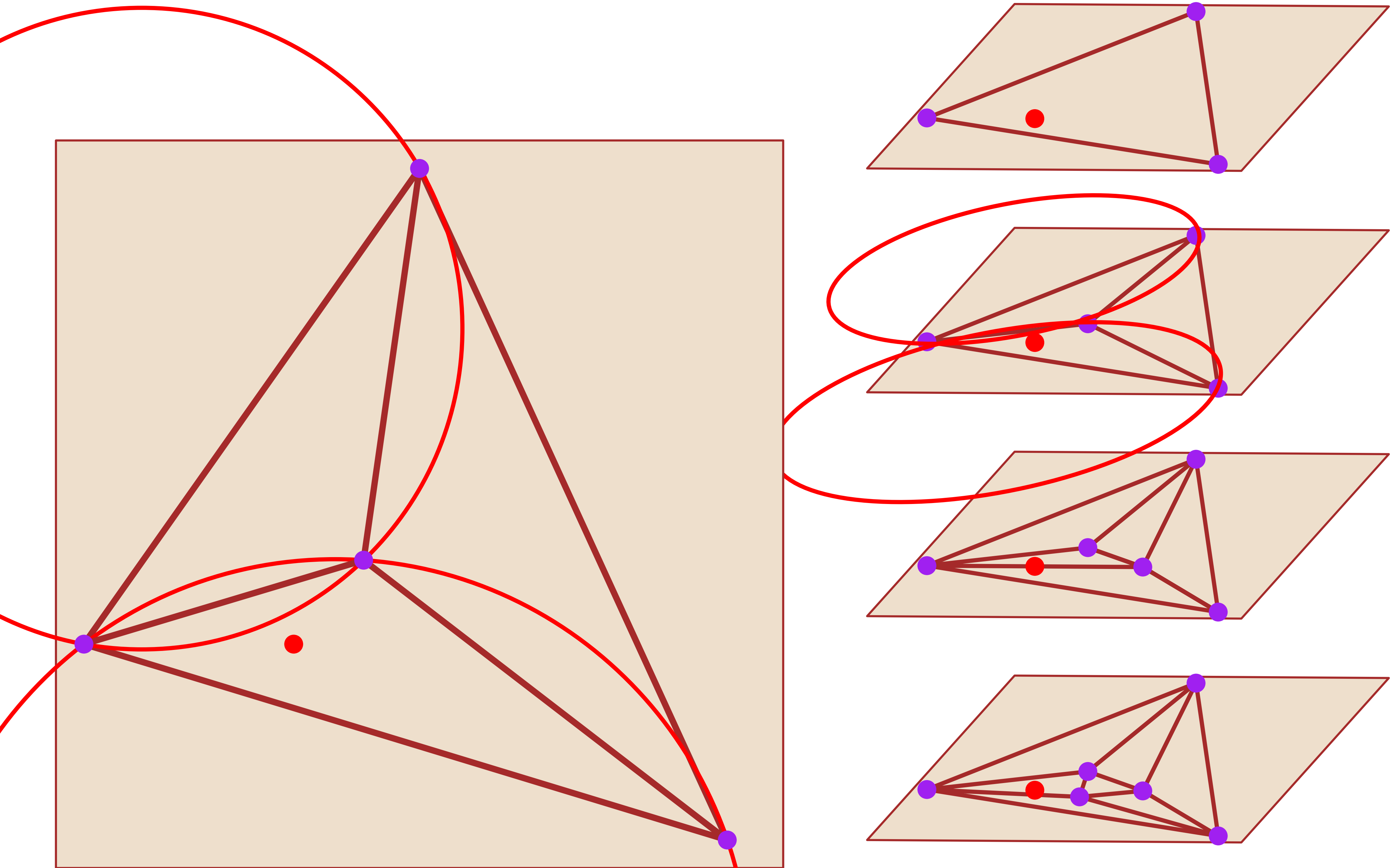




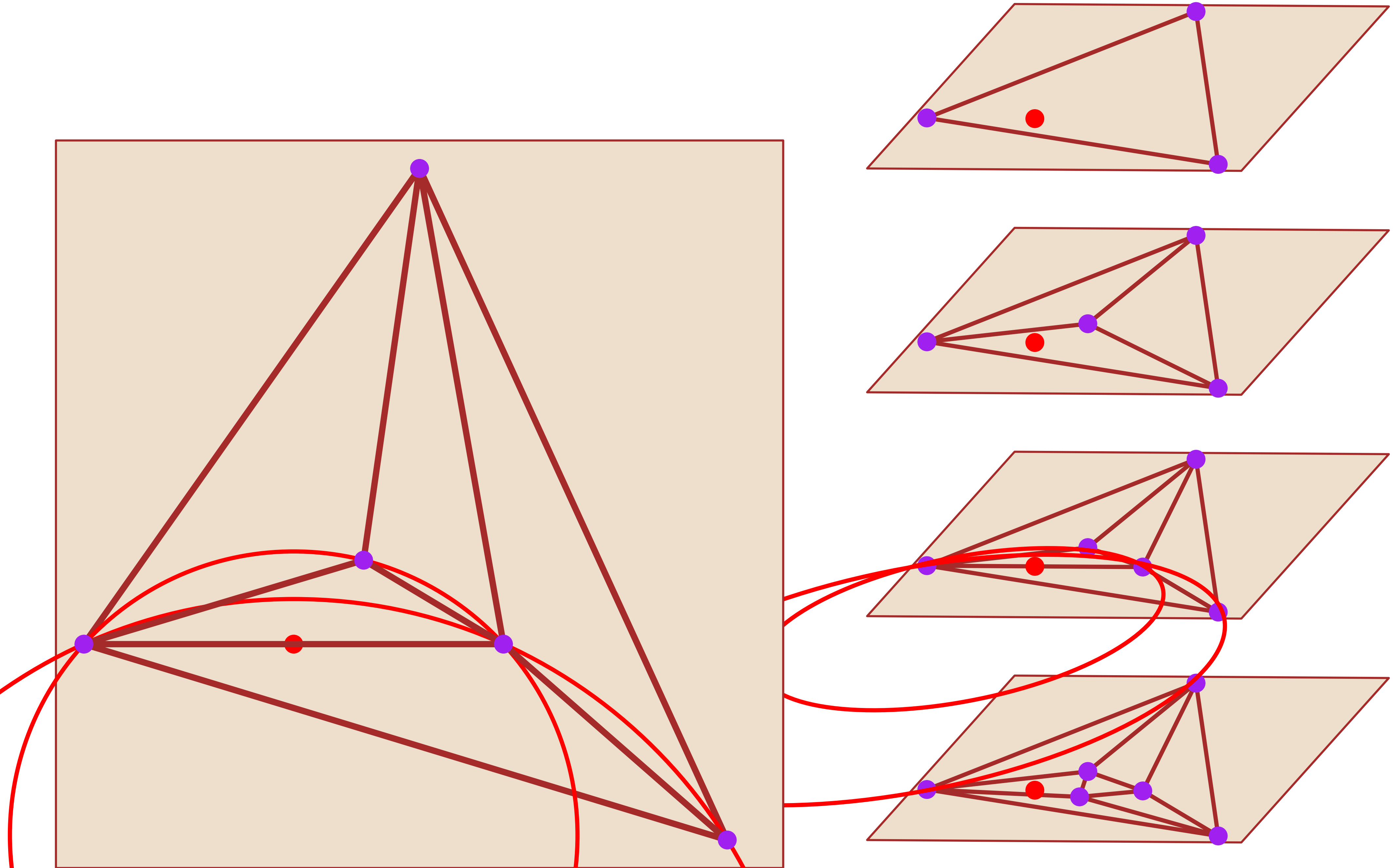
the Delaunay tree



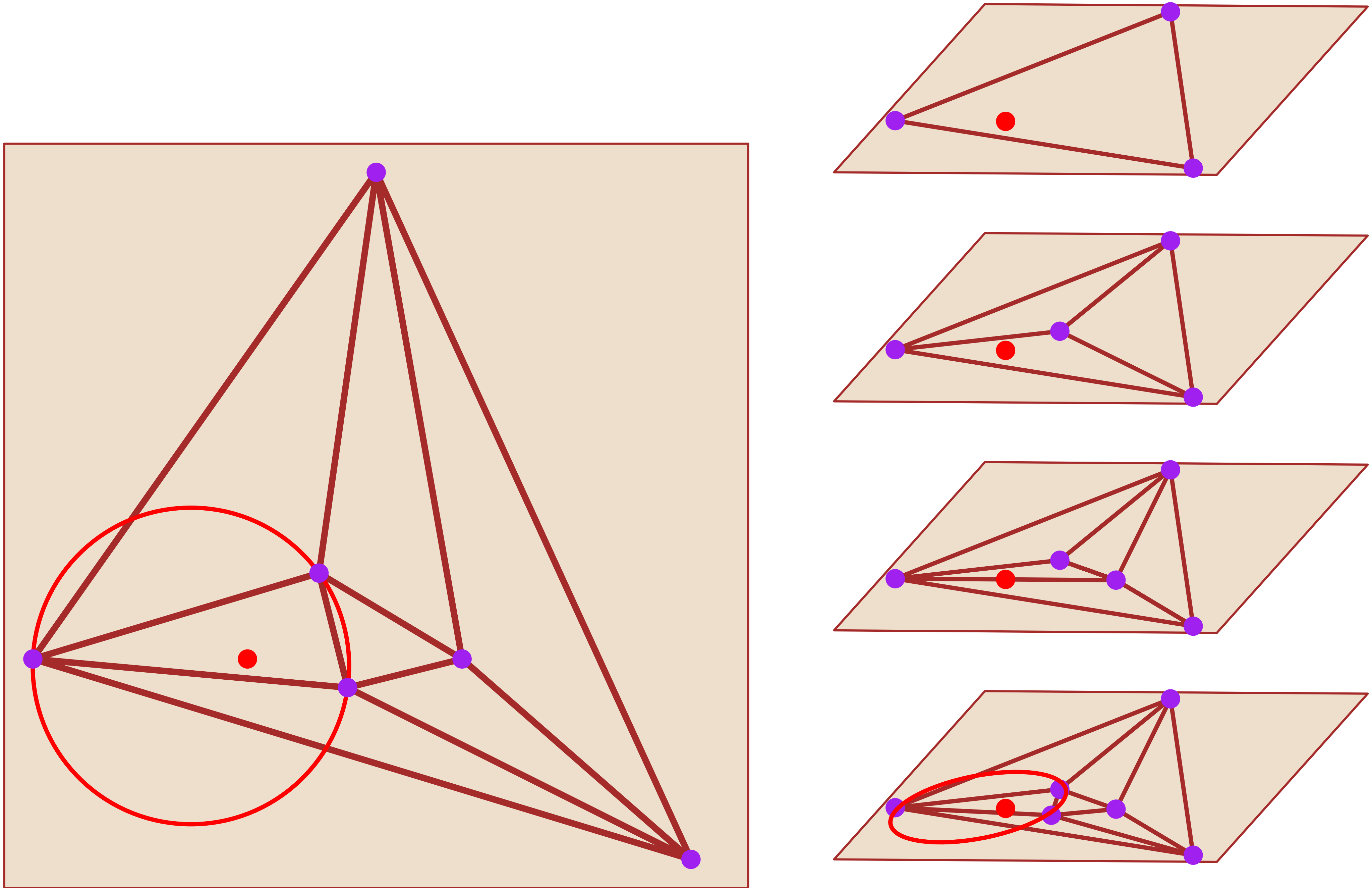
the Delaunay tree



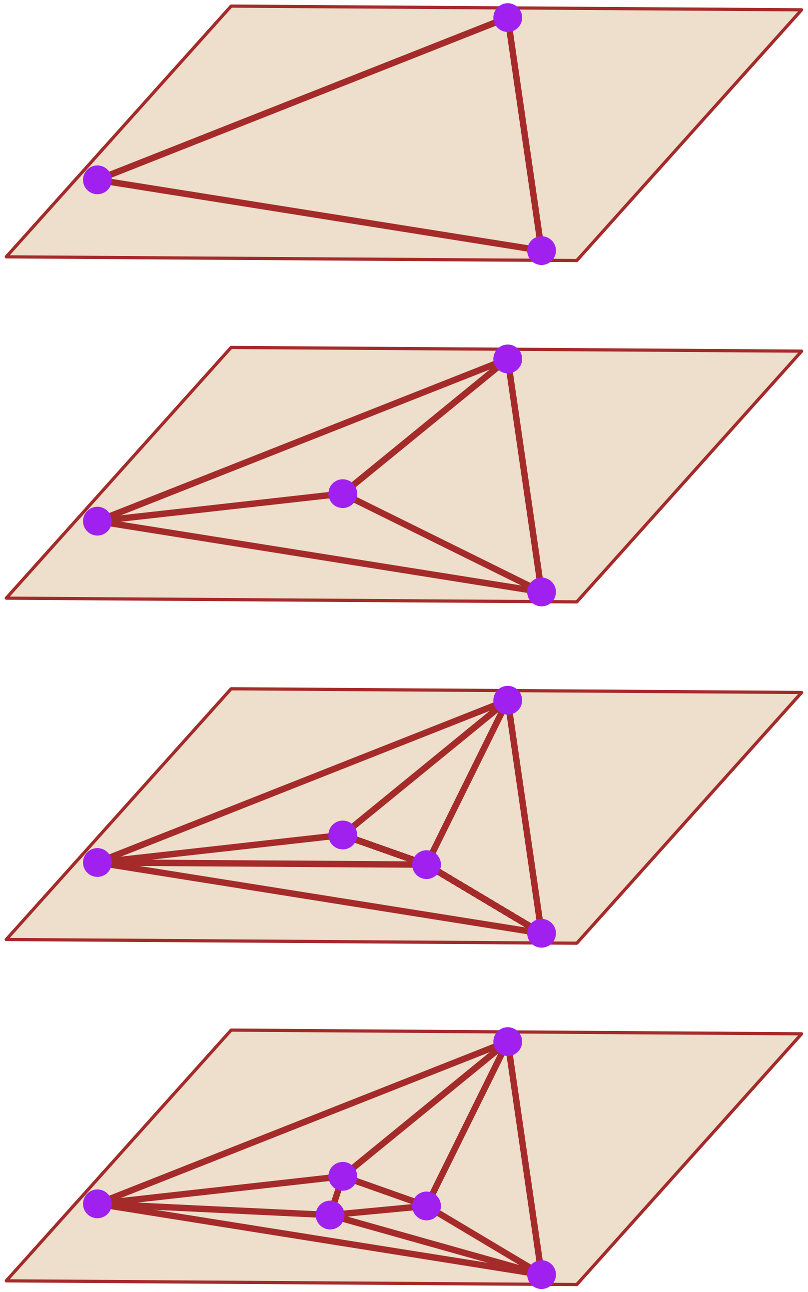
the Delaunay tree



the Delaunay tree



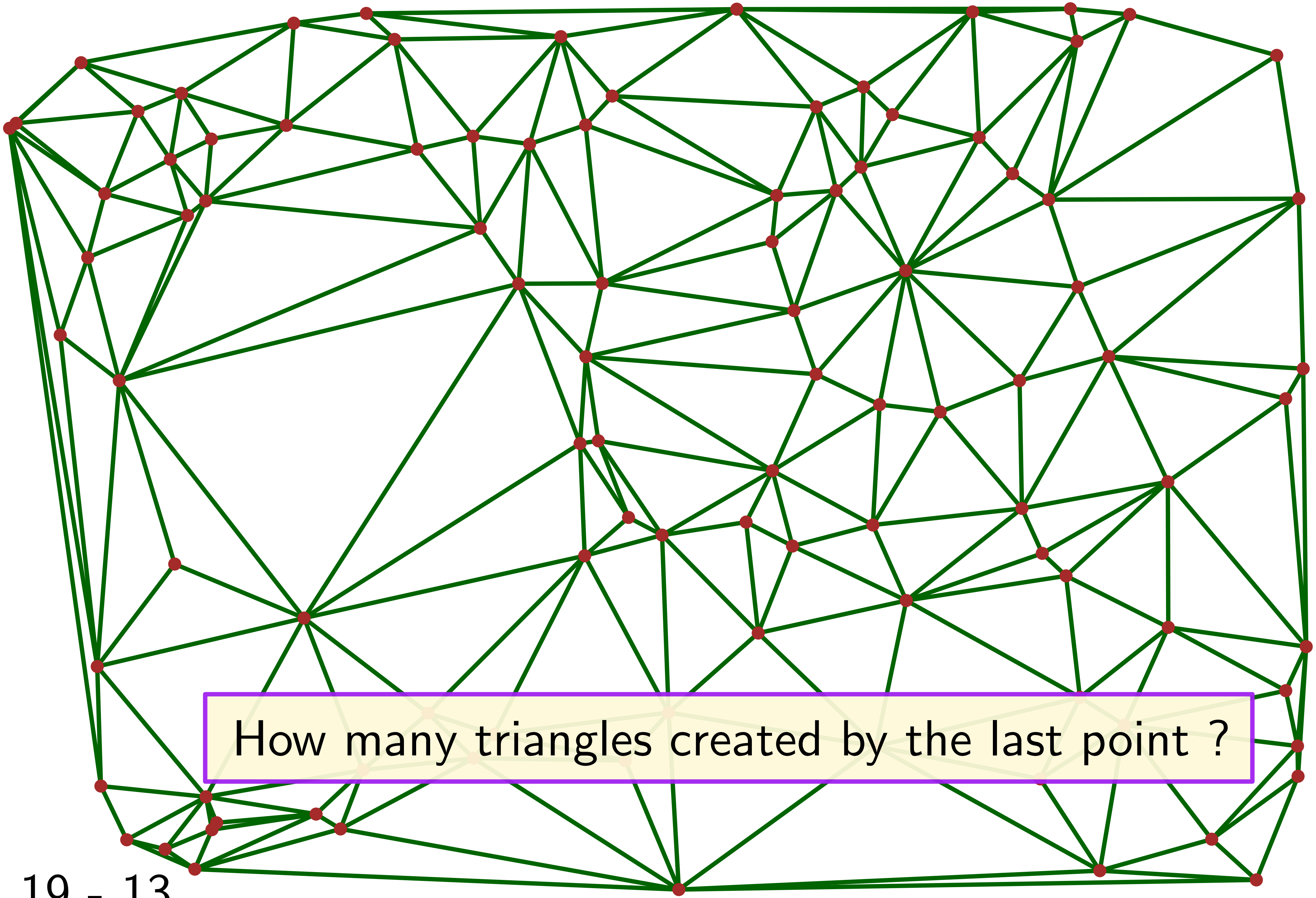
the Delaunay tree



locate based on incircle predicate

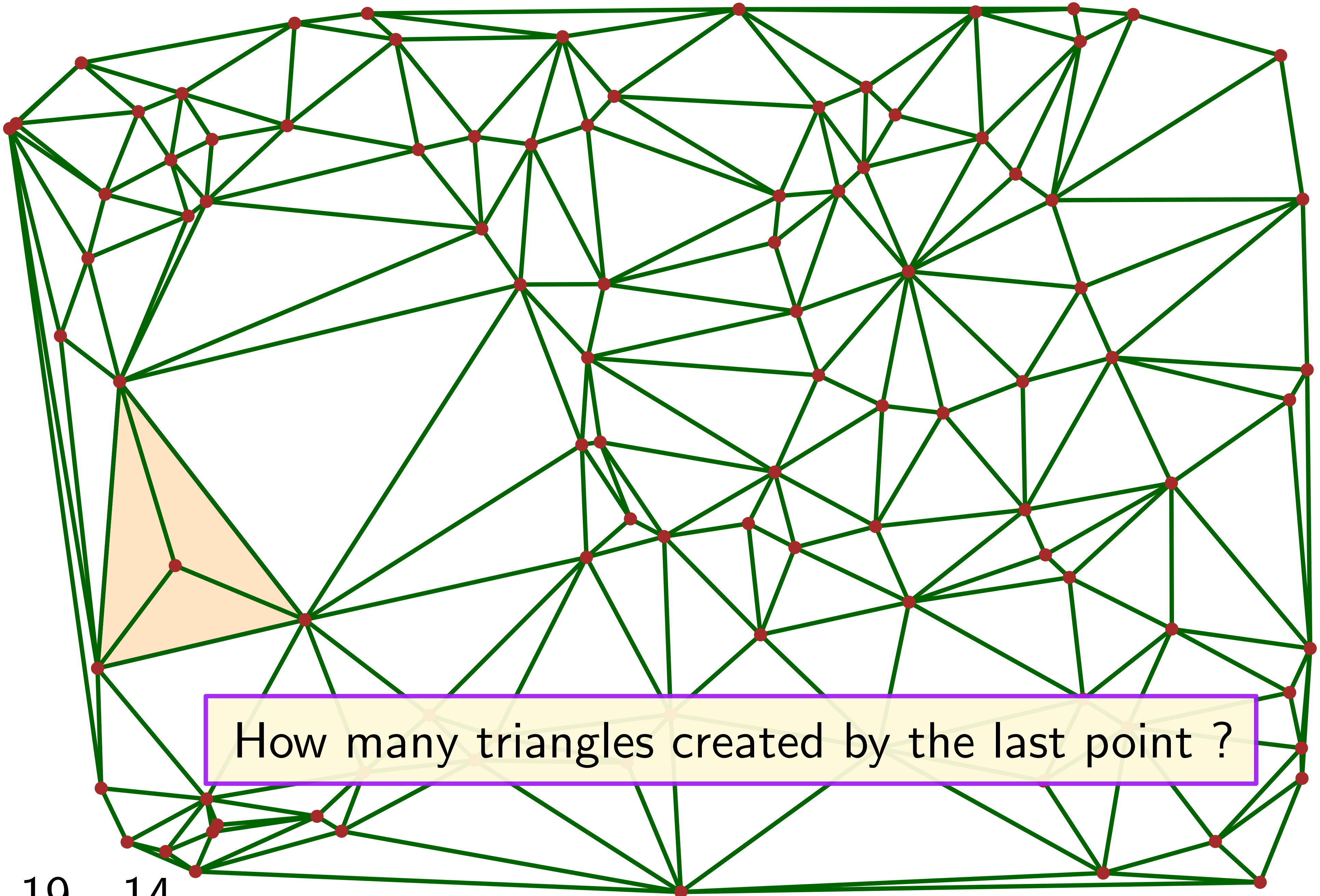
♯ triangles in the Delaunay tree

the Delaunay tree



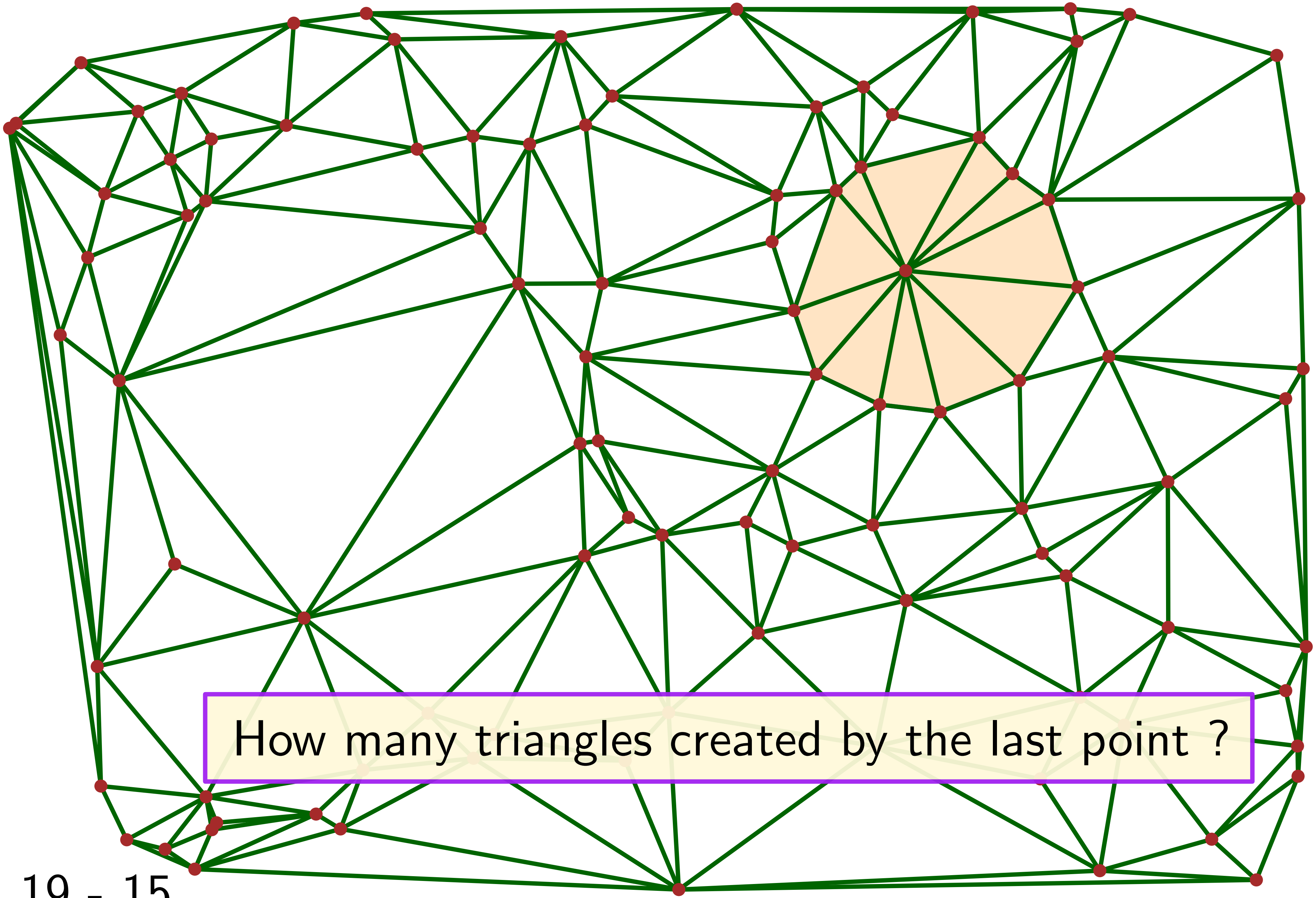
How many triangles created by the last point ?

the Delaunay tree



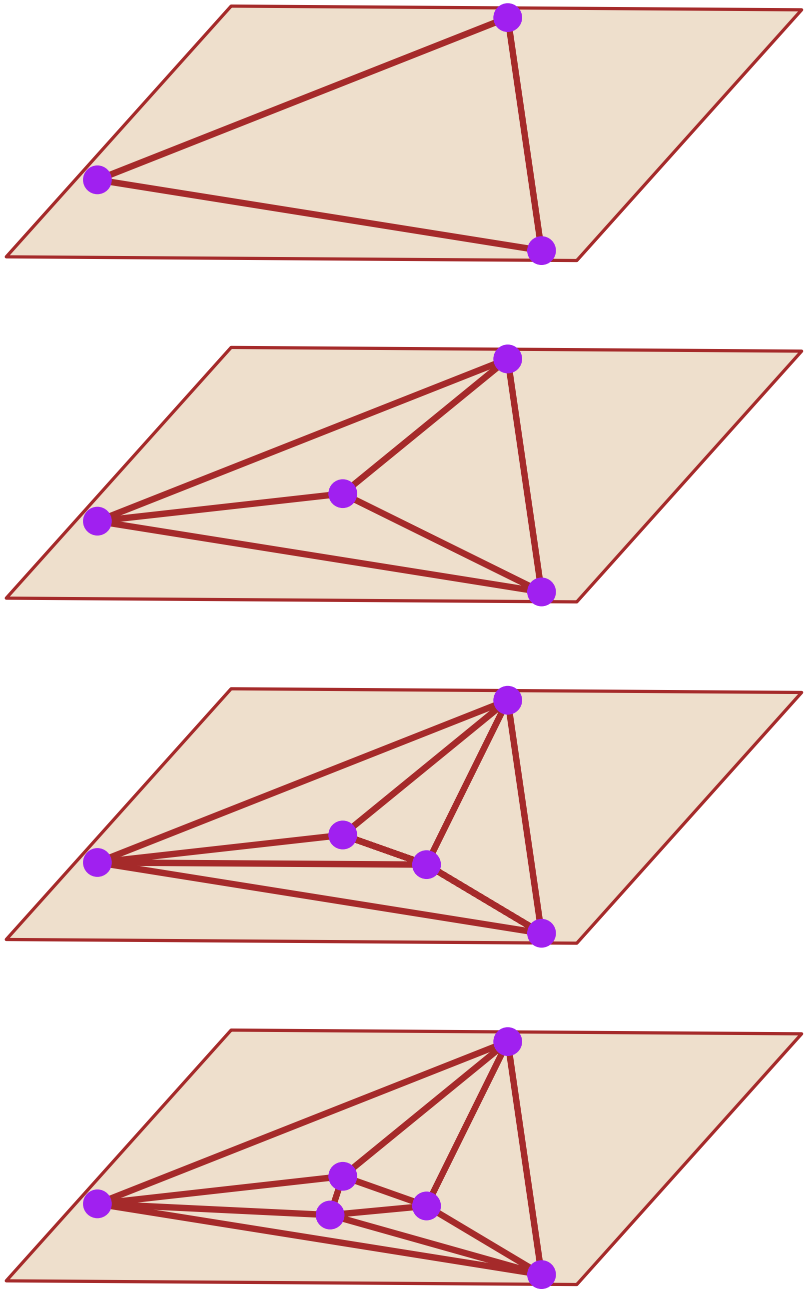
How many triangles created by the last point ?

the Delaunay tree



How many triangles created by the last point ?

the Delaunay tree

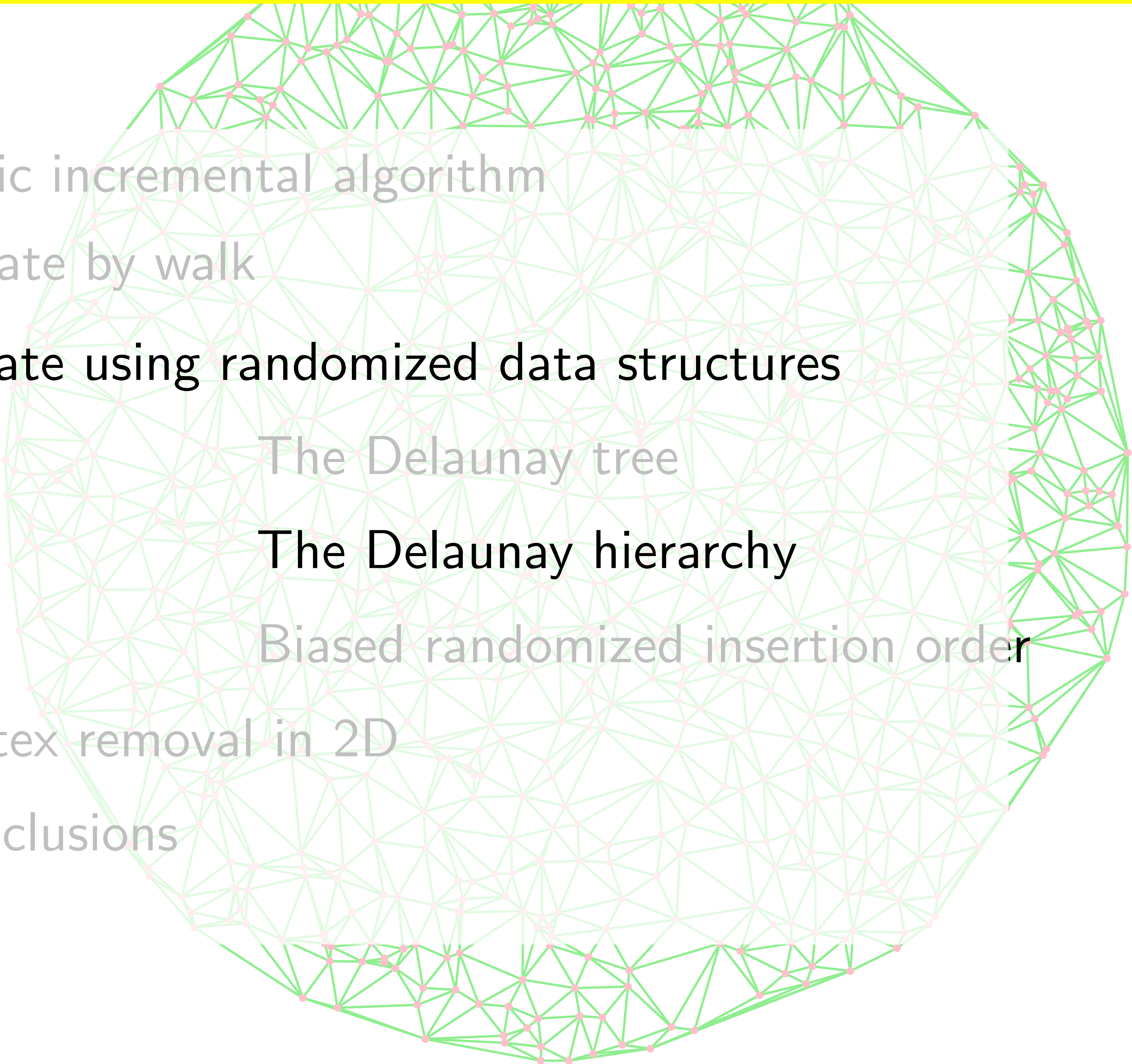


locate based on incircle predicate

$\sharp$ triangles in the Delaunay tree

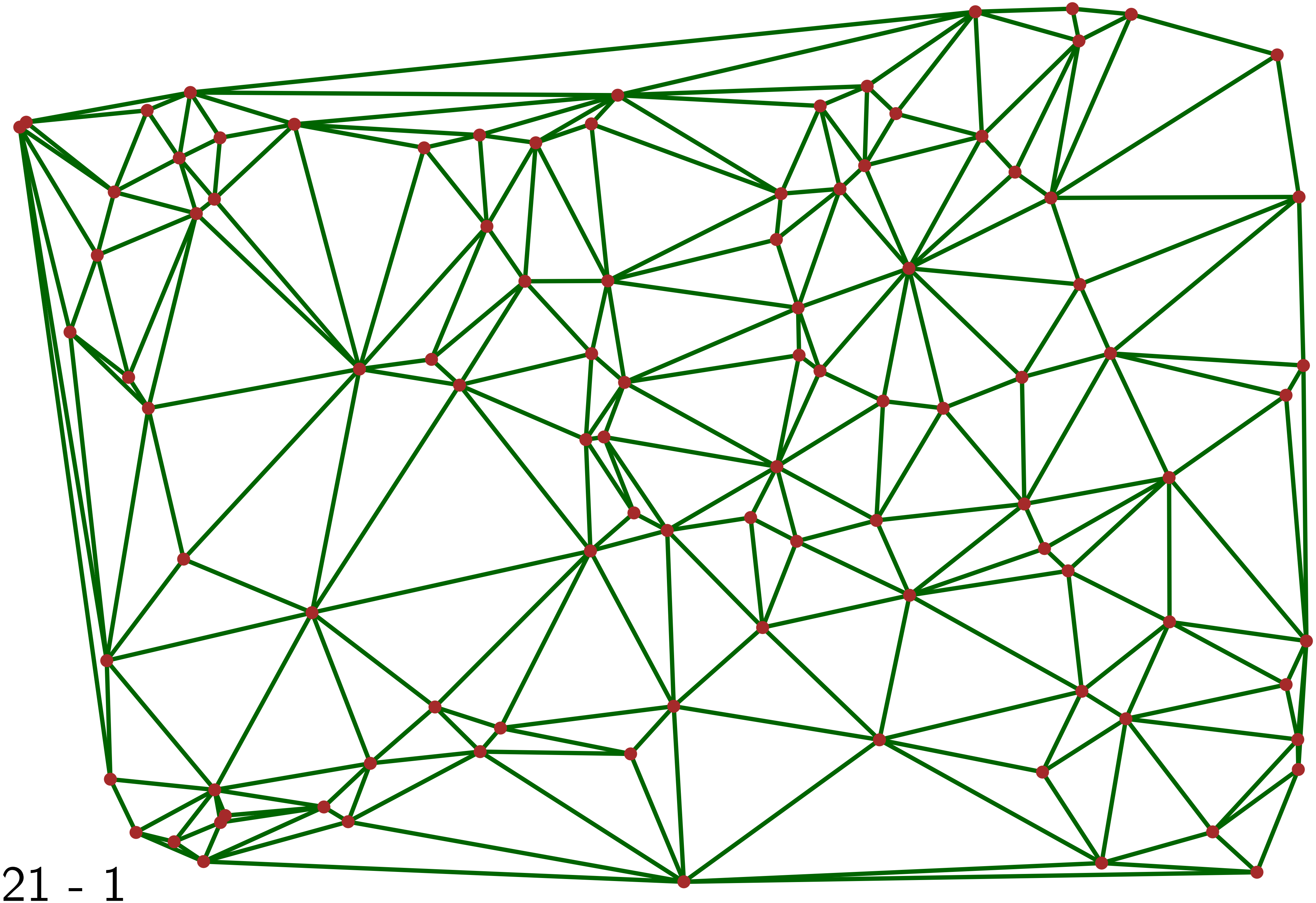
$= 6n$ (randomized)

Algorithms

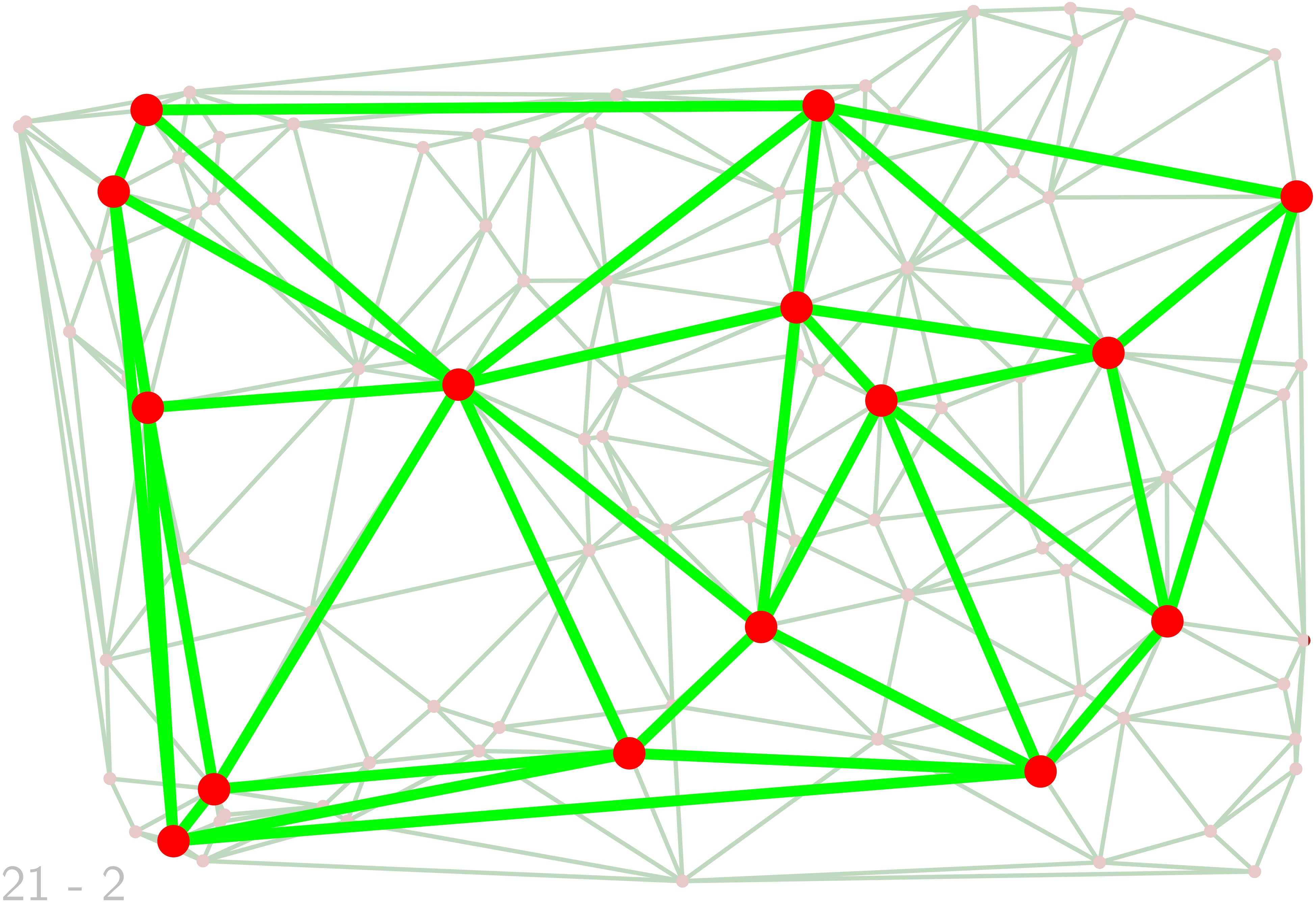


- Basic incremental algorithm
- Locate by walk
- Locate using randomized data structures
  - The Delaunay tree
  - The Delaunay hierarchy
  - Biased randomized insertion order
- Vertex removal in 2D
- Conclusions

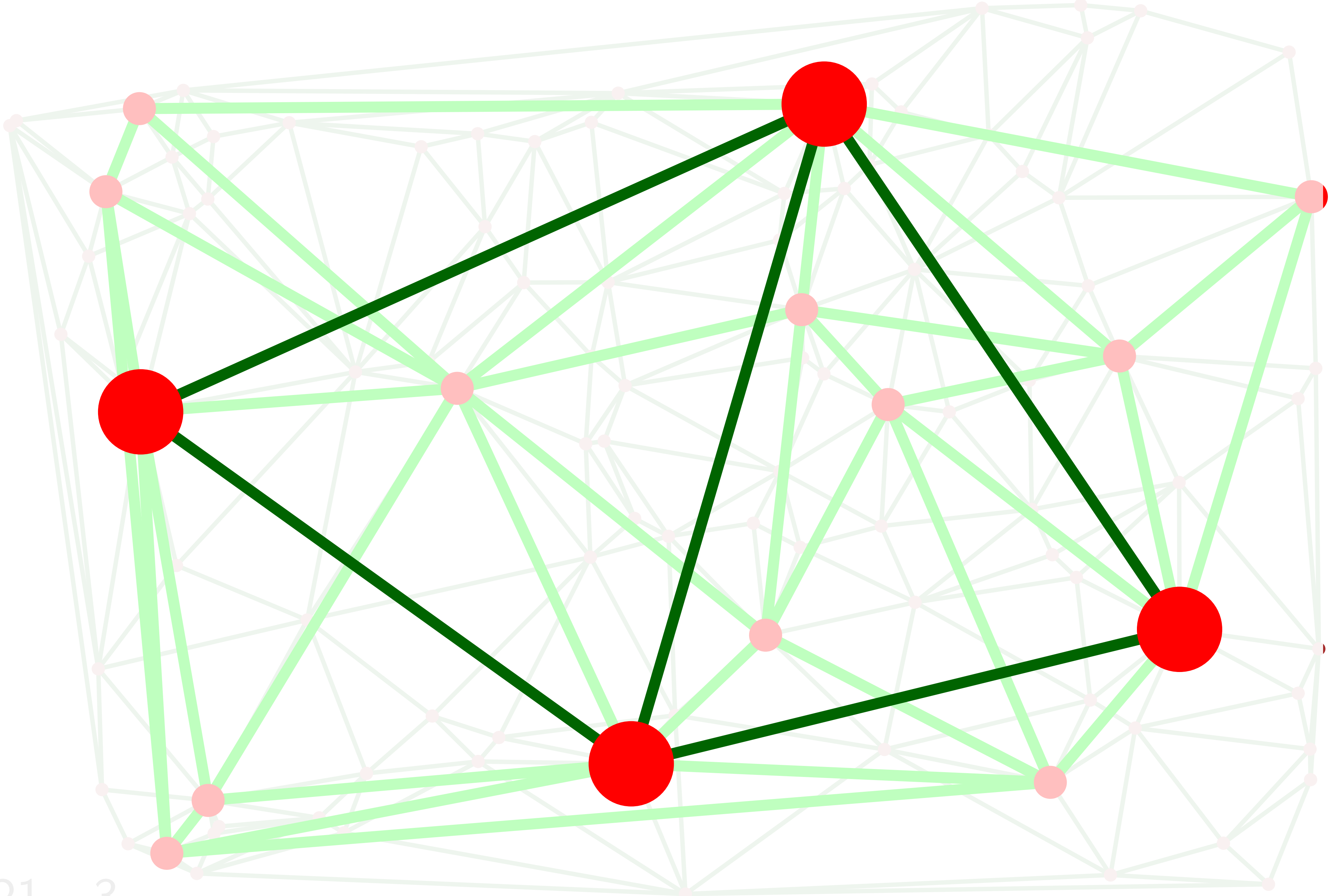
the Delaunay hierarchy



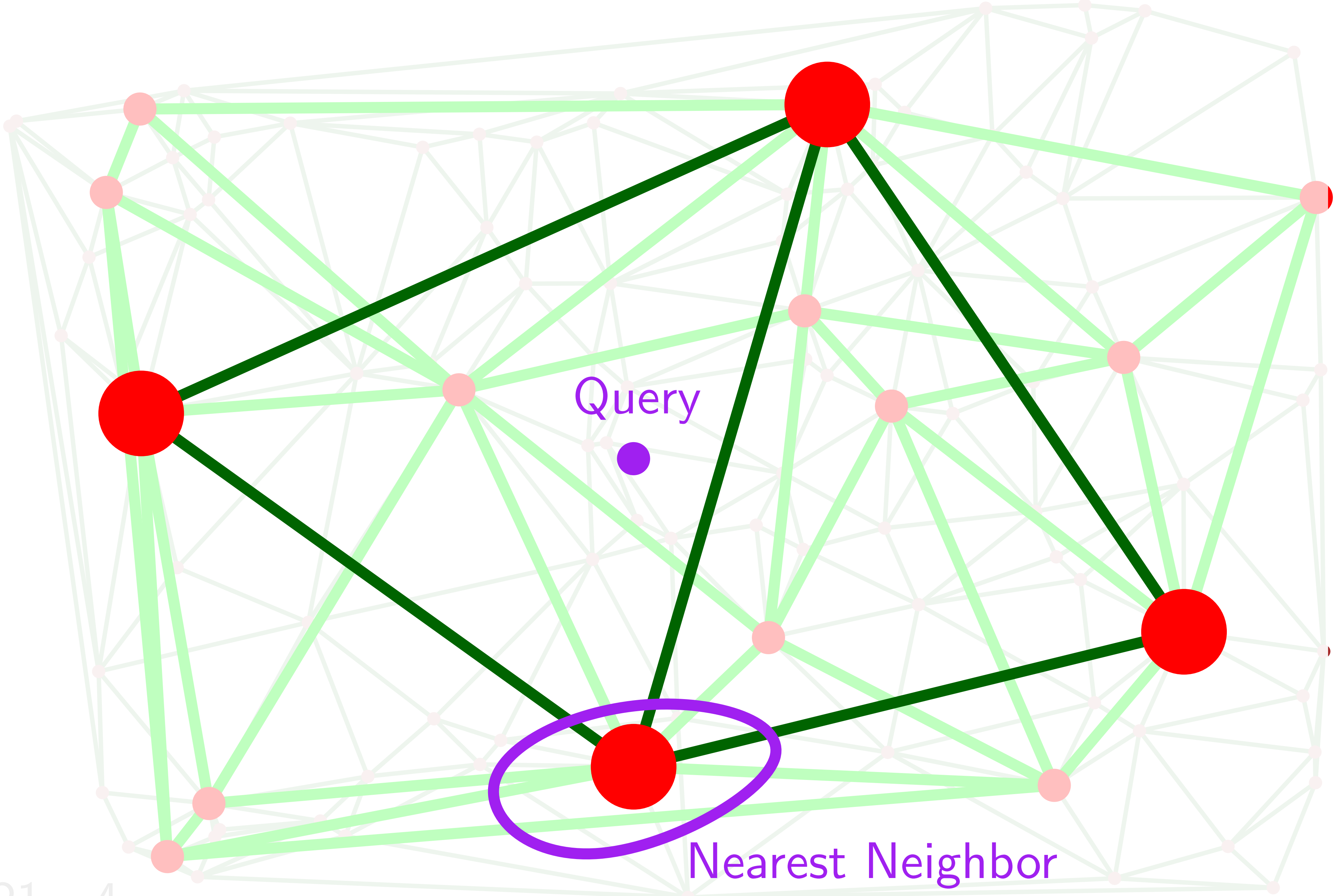
the Delaunay hierarchy



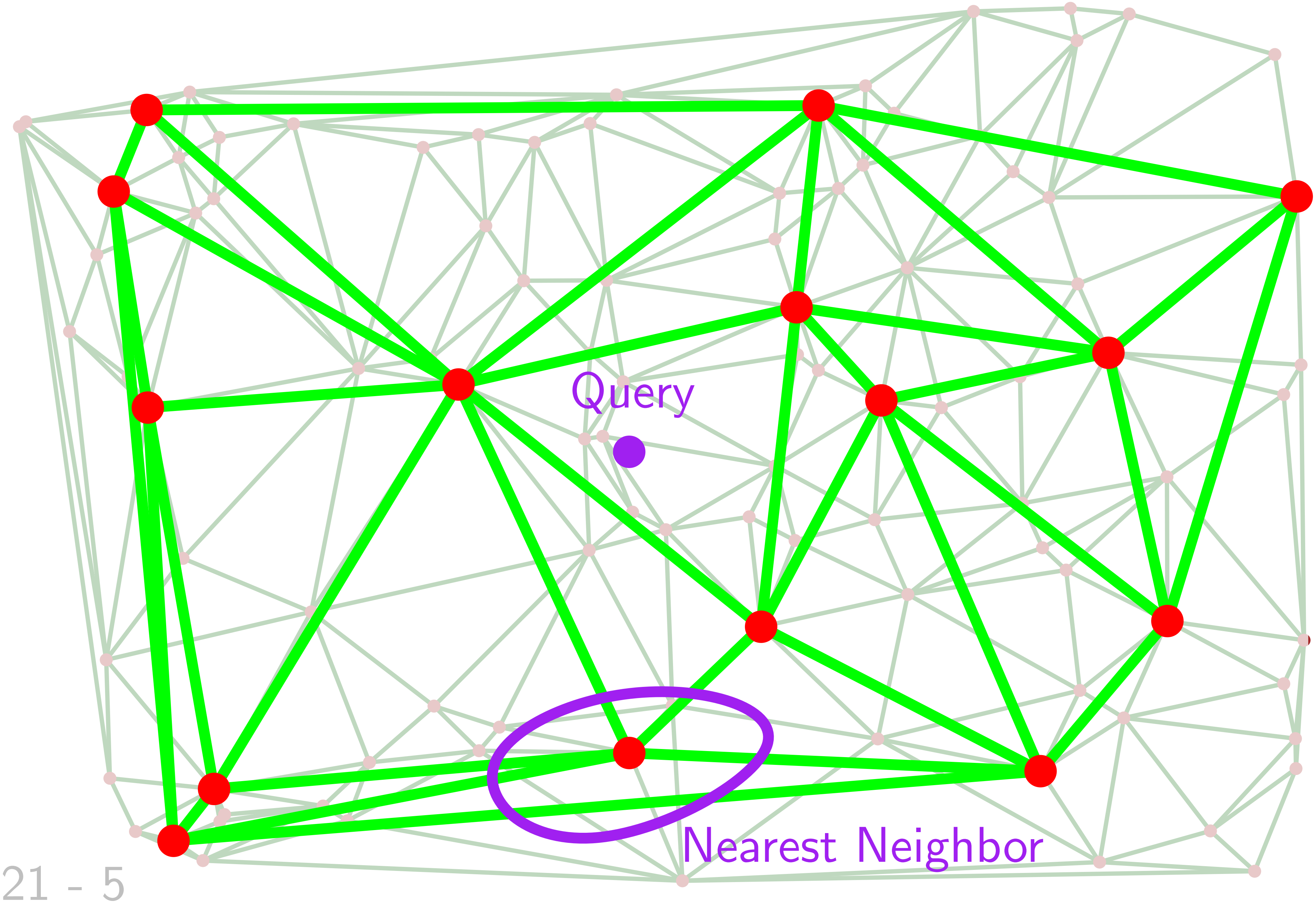
the Delaunay hierarchy



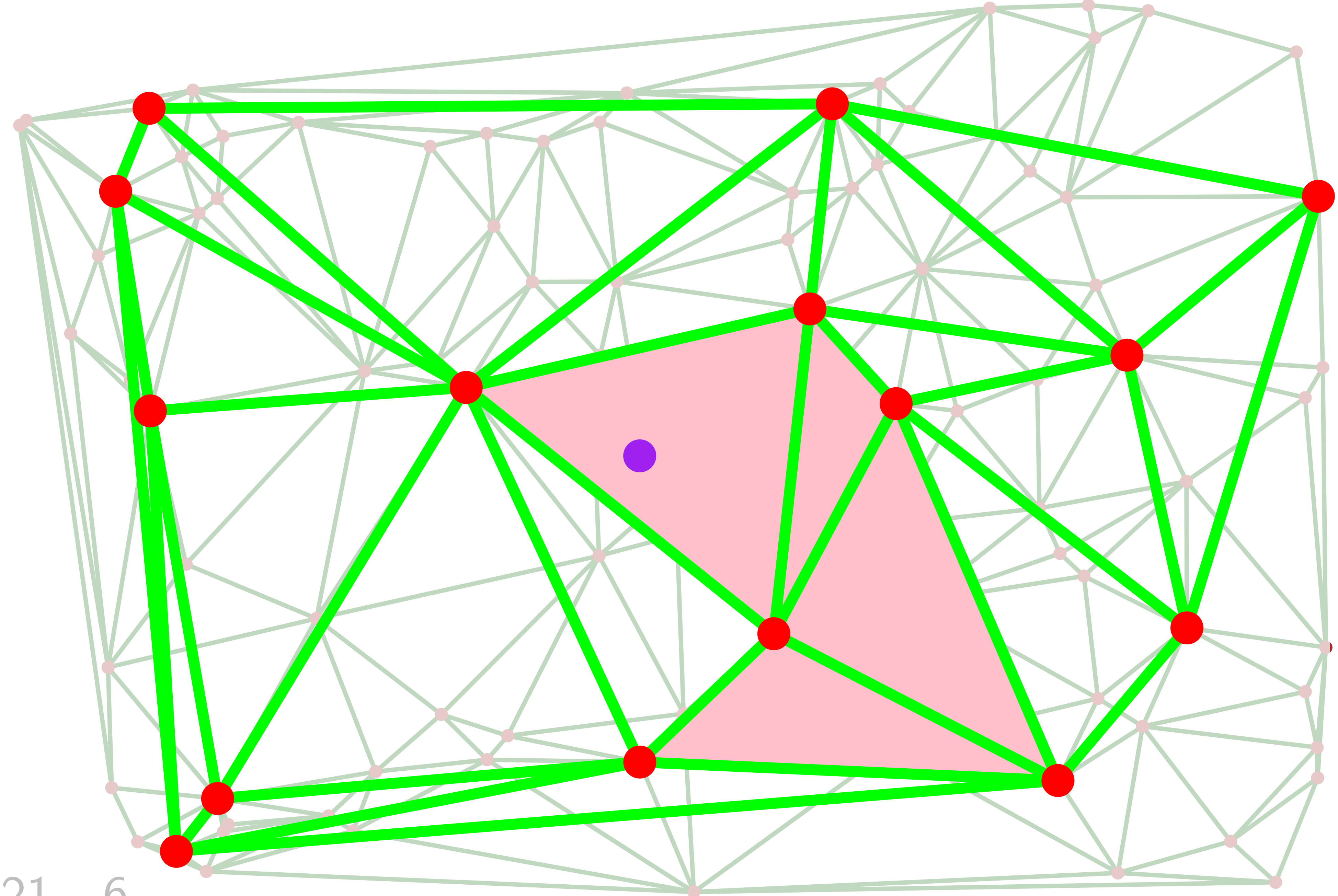
the Delaunay hierarchy



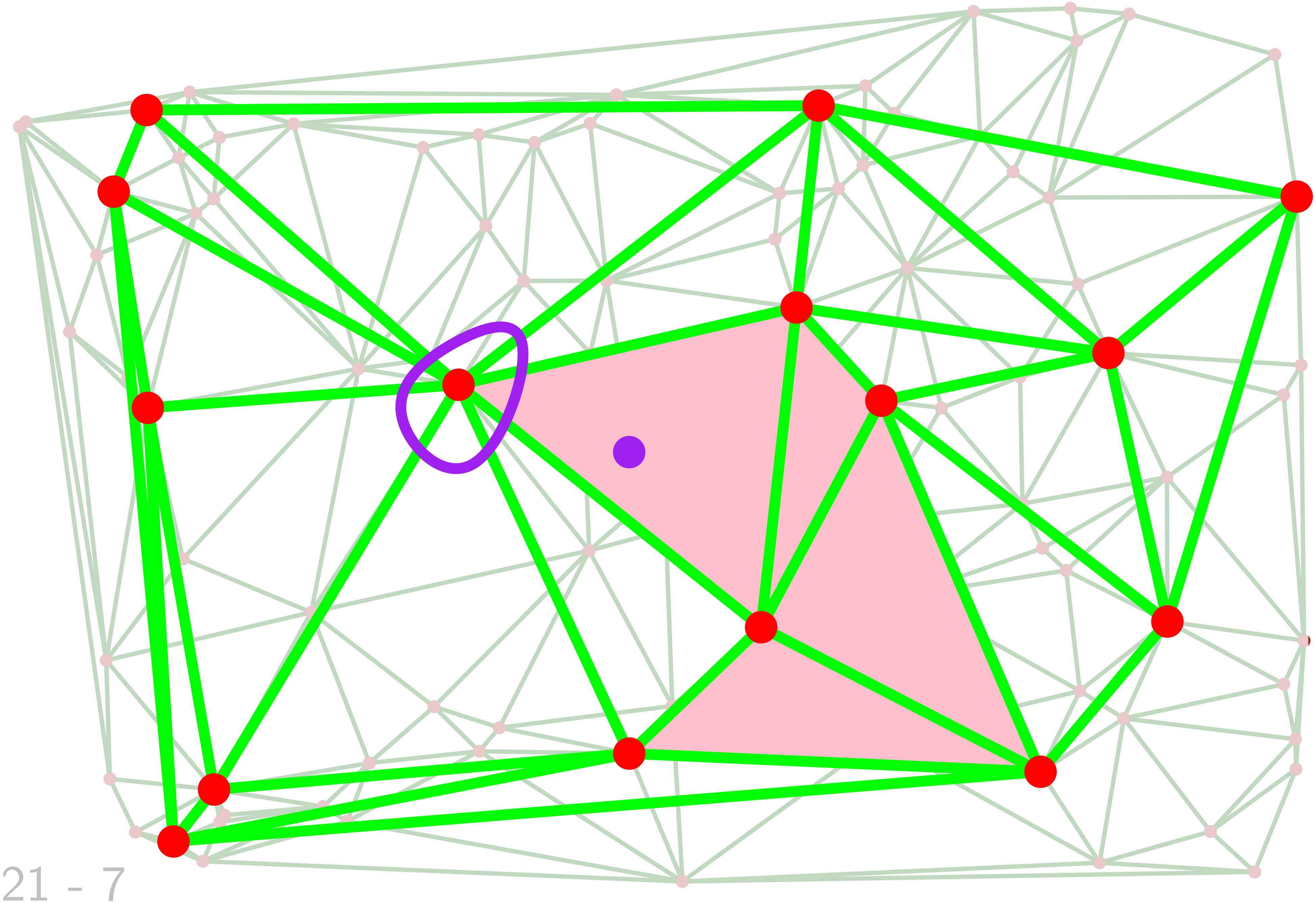
the Delaunay hierarchy



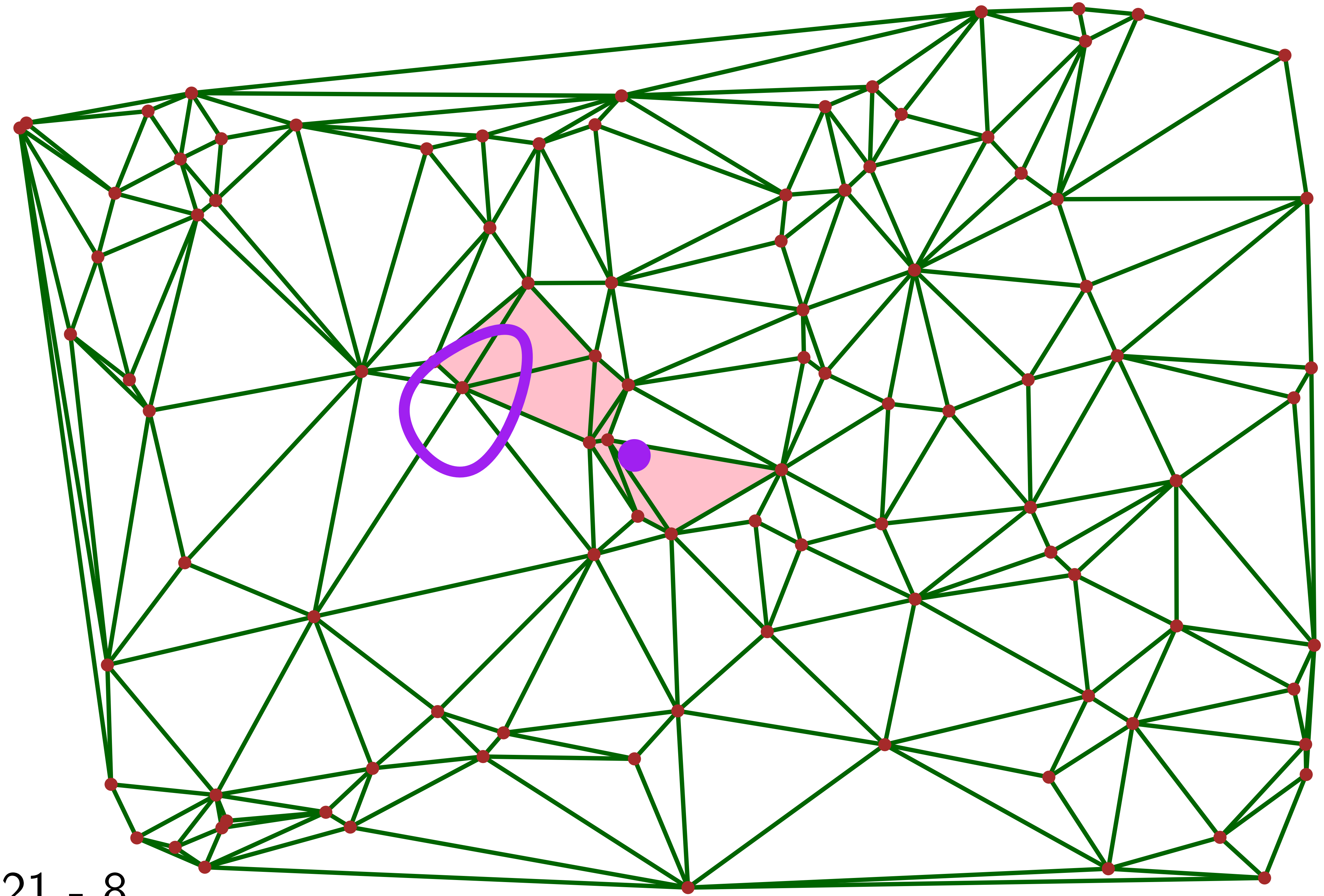
the Delaunay hierarchy



the Delaunay hierarchy



the Delaunay hierarchy



The Delaunay tree

locate based on incircle predicate

$\sharp$ triangles in the Delaunay tree

$= 6n$ (randomized)

The Delaunay hierarchy

based on orientation predicate

$\sharp$ triangles in the hierarchy

can be chosen

$= 1.03 \times 2n$ (expected)

The Delaunay tree

locate based on incircle predicate

$\sharp$ triangles in the Delaunay tree

$= 6n$ (randomized)

The Delaunay hierarchy

based on orientation predicate

$\sharp$ triangles in the hierarchy

can be chosen

$= 1.03 \times 2n$ (expected)

The Delaunay tree

locate based on incircle predicate

$\sharp$ triangles in the Delaunay tree

$= 6n$ (randomized)

$O(n \log n)$

The Delaunay hierarchy

based on orientation predicate

$\sharp$ triangles in the hierarchy

can be chosen

$= 1.03 \times 2n$ (expected)

The Delaunay tree

locate based on incircle predicate

$\sharp$ triangles in the Delaunay tree

$= 6n$ (randomized)

2.3 seconds

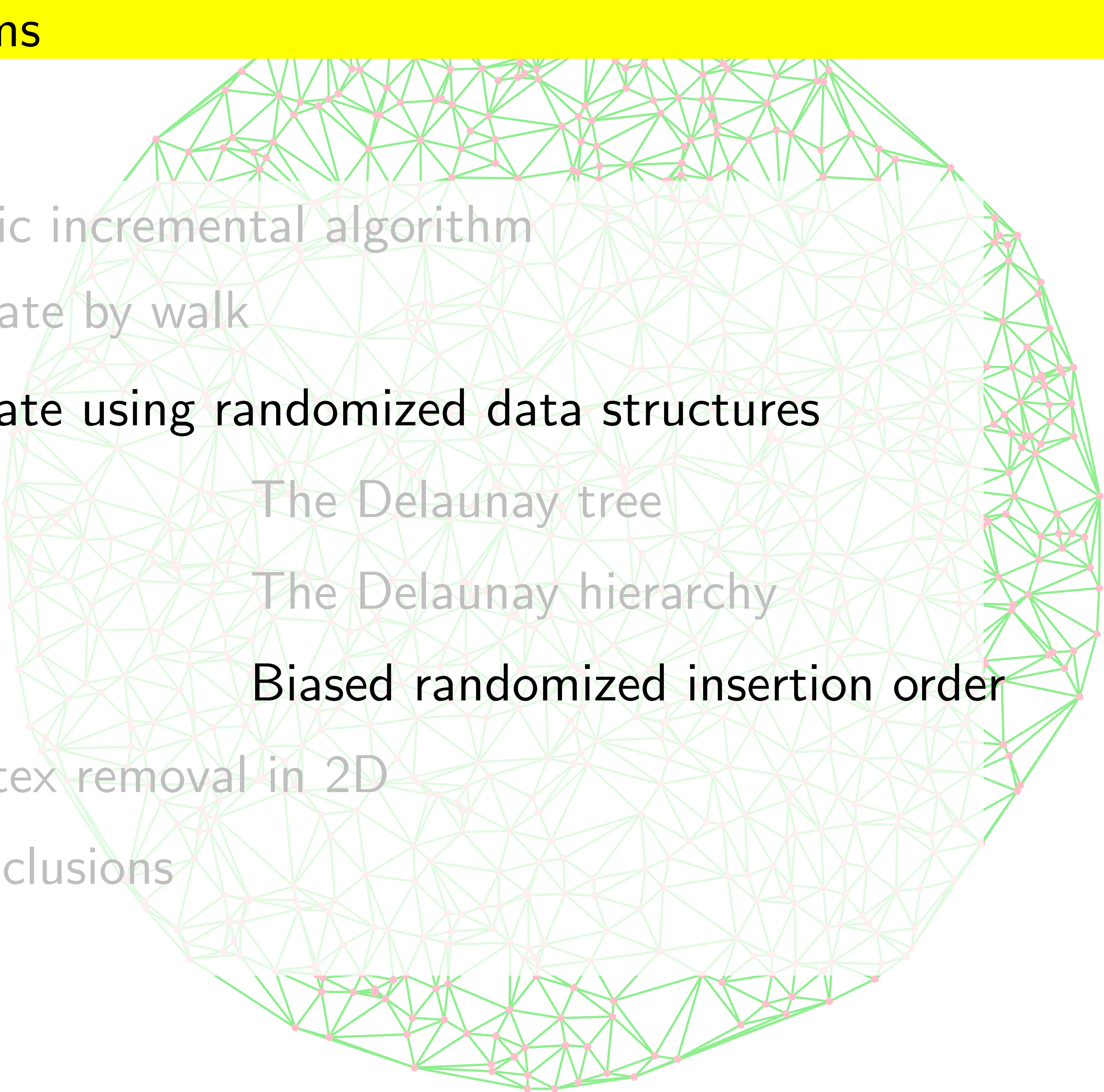
17 seconds

50000 random points (original benchmarks in 2000).

```
#include
   <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/Random.h>

#include <vector>
#include <cassert>

typedef
   CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_3<K,
                                    CGAL::Fast_location> Delaunay;
typedef Delaunay::Point Point;
```

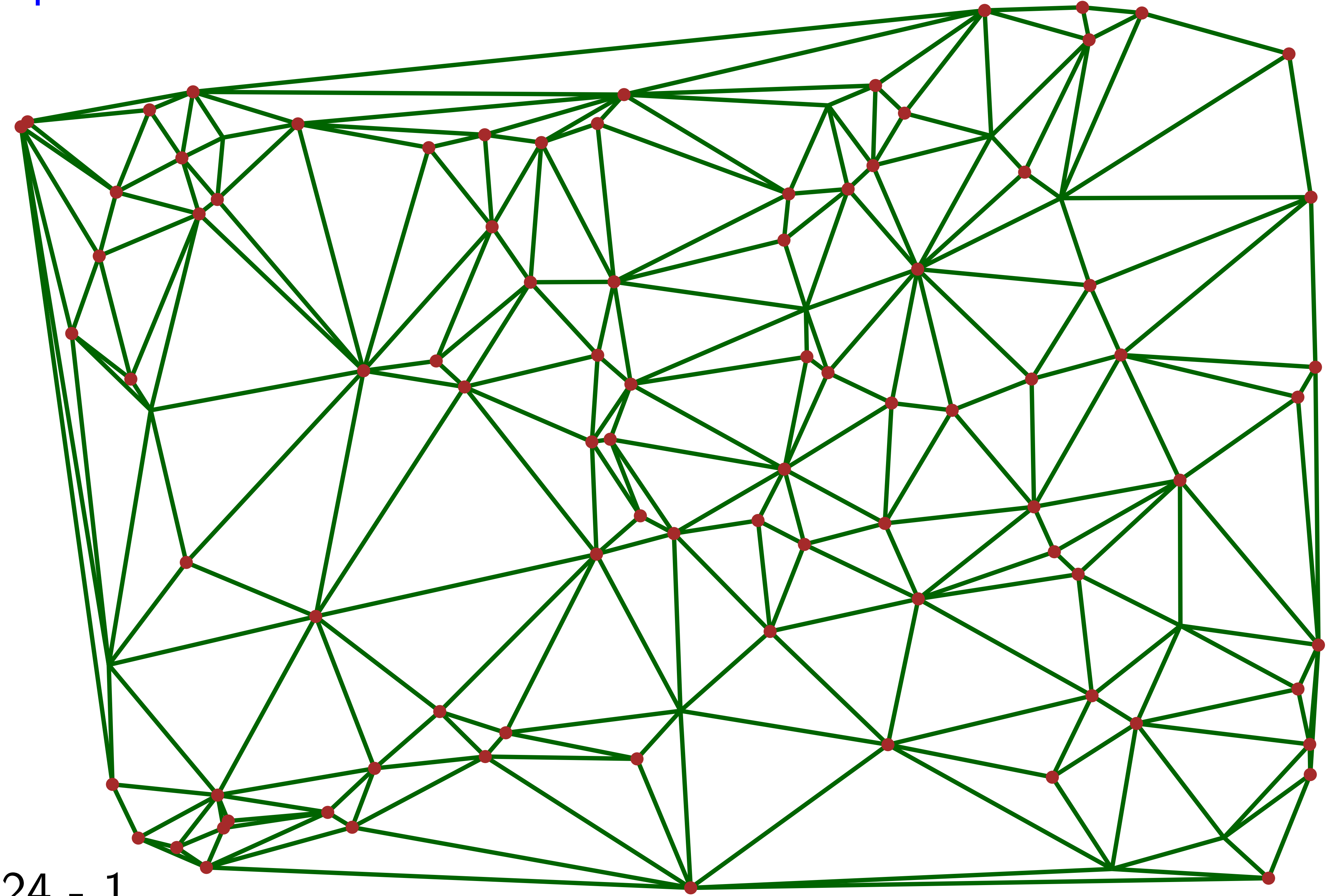
```
int main()
{ Delaunay T;
   std::vector<Point> P;
   for (int z=0 ; z<20 ; z++)
    for (int y=0 ; y<20 ; y++)
     for (int x=0 ; x<20 ; x++)
      P.push_back(Point(x,y,z));

   Delaunay T(P.begin(), P.end());
   assert( T.number_of_vertices() == 8000 );

   for (int i=0; i<10000; ++i)
   T.nearest_vertex
     ( Point(CGAL::default_random.get_double(0,20),
        CGAL::default_random.get_double(0,20),
        CGAL::default_random.get_double(0,20)) );

   return 0; }
```

Algorithms

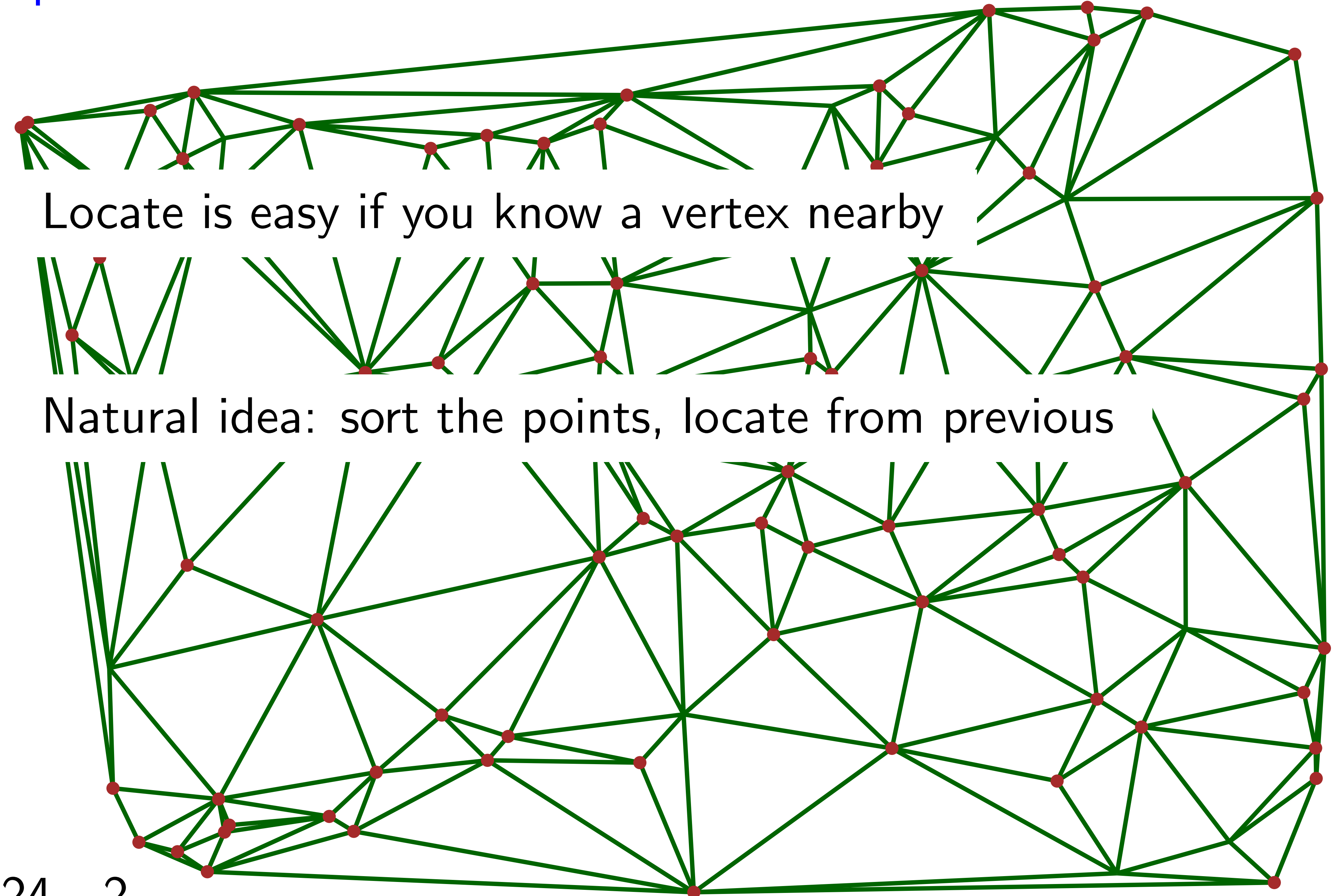


- Basic incremental algorithm
- Locate by walk
- Locate using randomized data structures
  - The Delaunay tree
  - The Delaunay hierarchy
  - Biased randomized insertion order
- Vertex removal in 2D
- Conclusions

Efficiency of incremental algorithms depends on the order of insertion

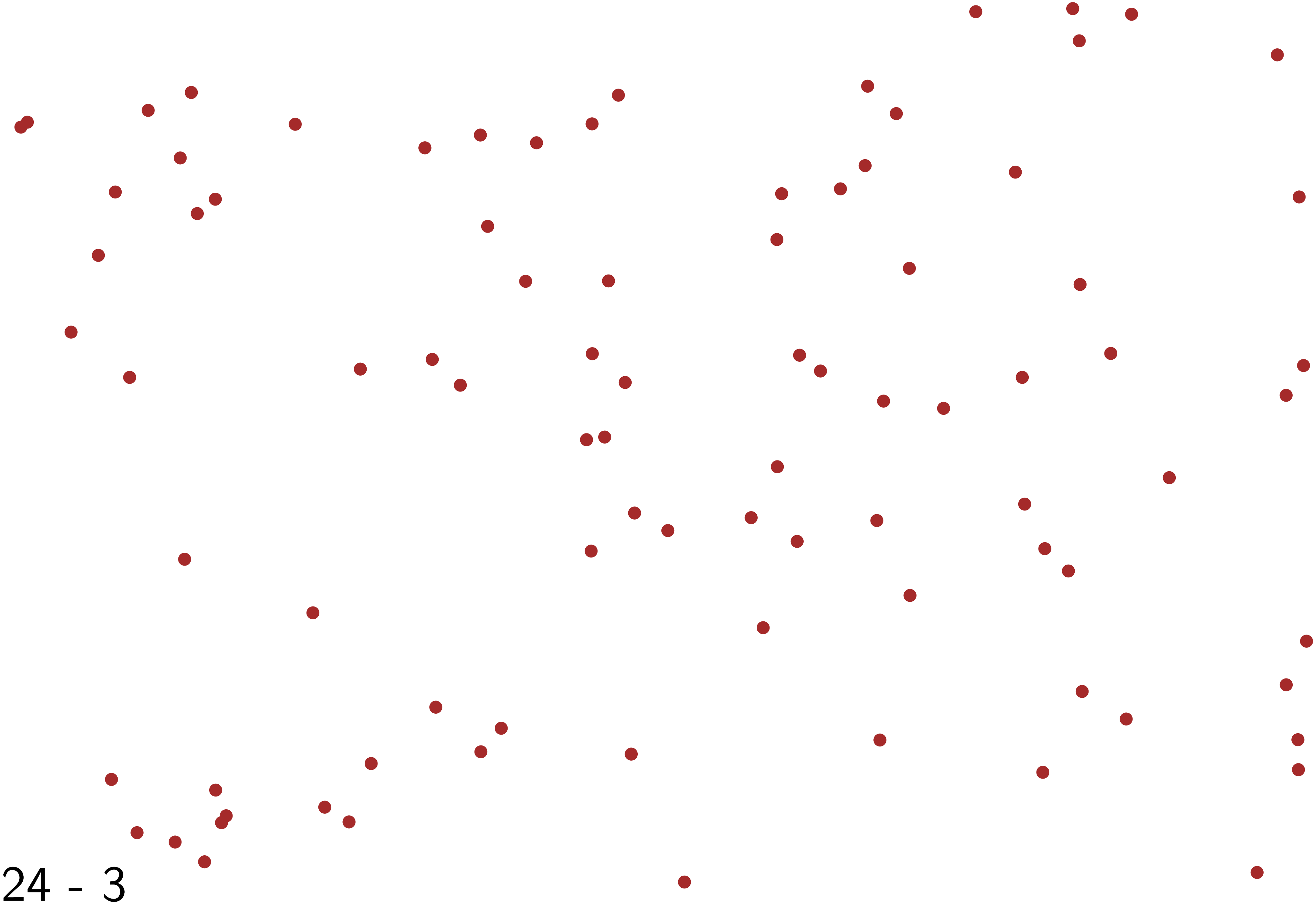


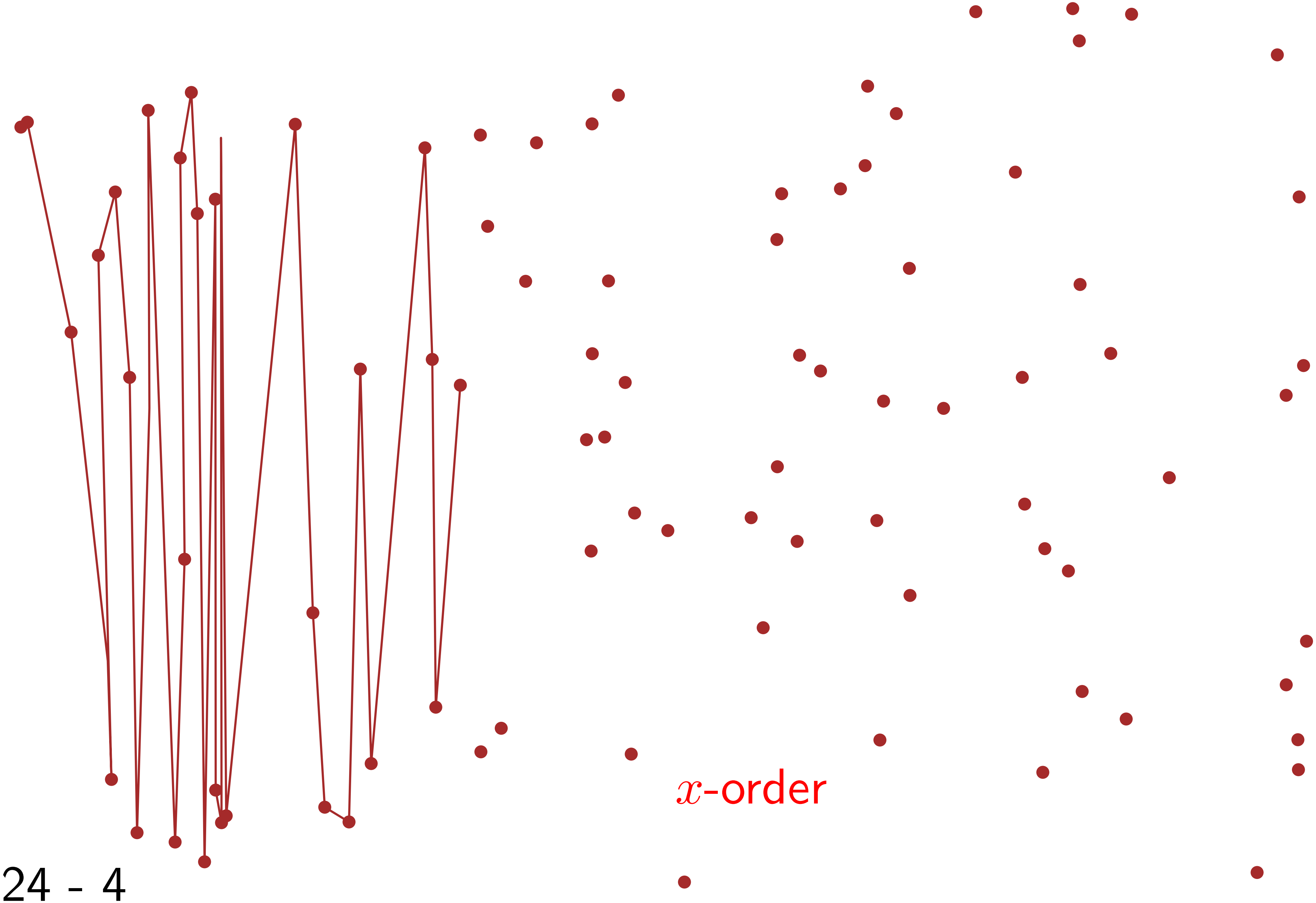
Efficiency of incremental algorithms depends on the order of insertion



Locate is easy if you know a vertex nearby

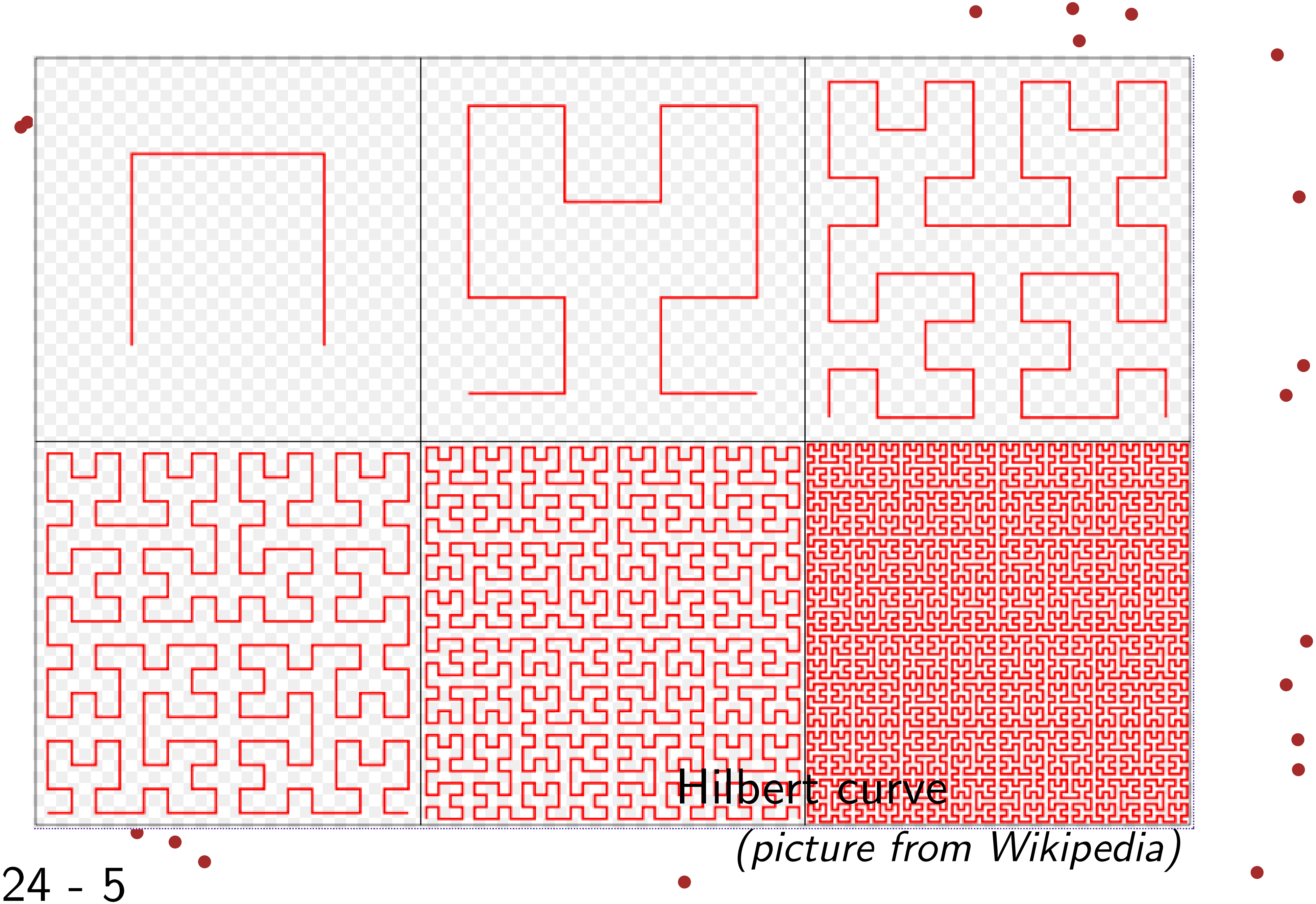
Natural idea: sort the points, locate from previous





x-order

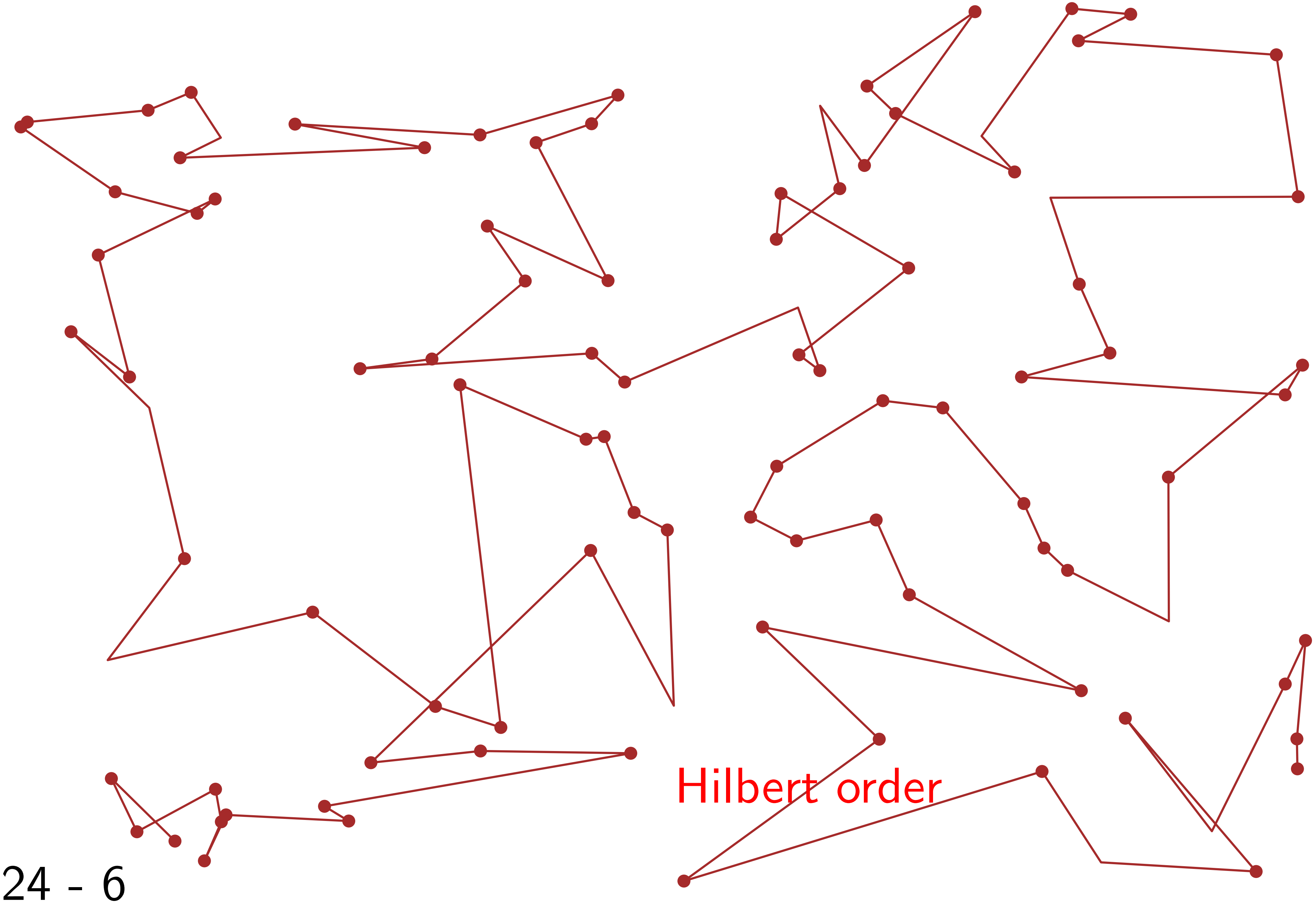
biased random insertion order



Hilbert curve

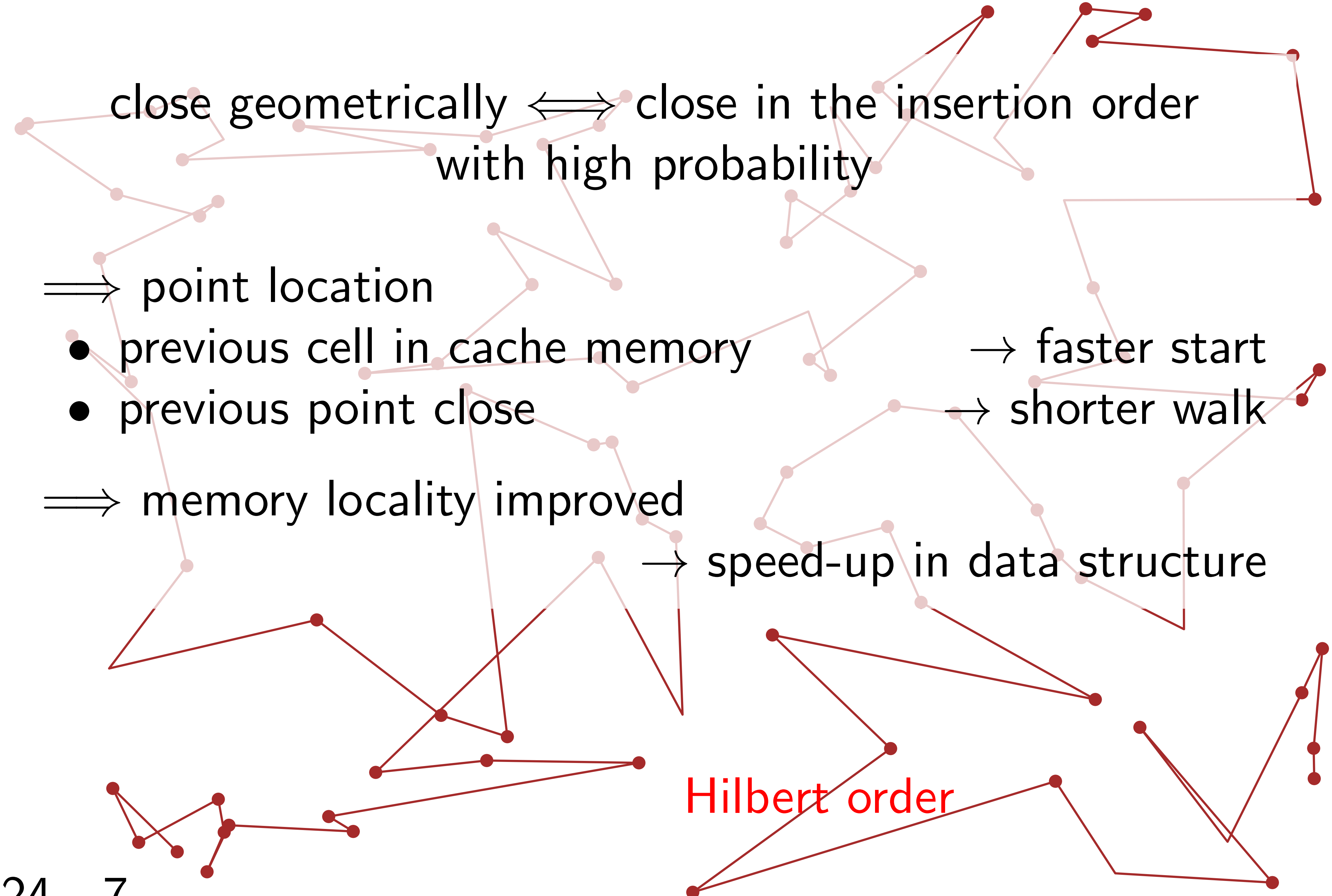
(picture from Wikipedia)

biased random insertion order



Hilbert order

biased random insertion order



close geometrically $\Longleftrightarrow$ close in the insertion order
with high probability

$\Longrightarrow$ point location

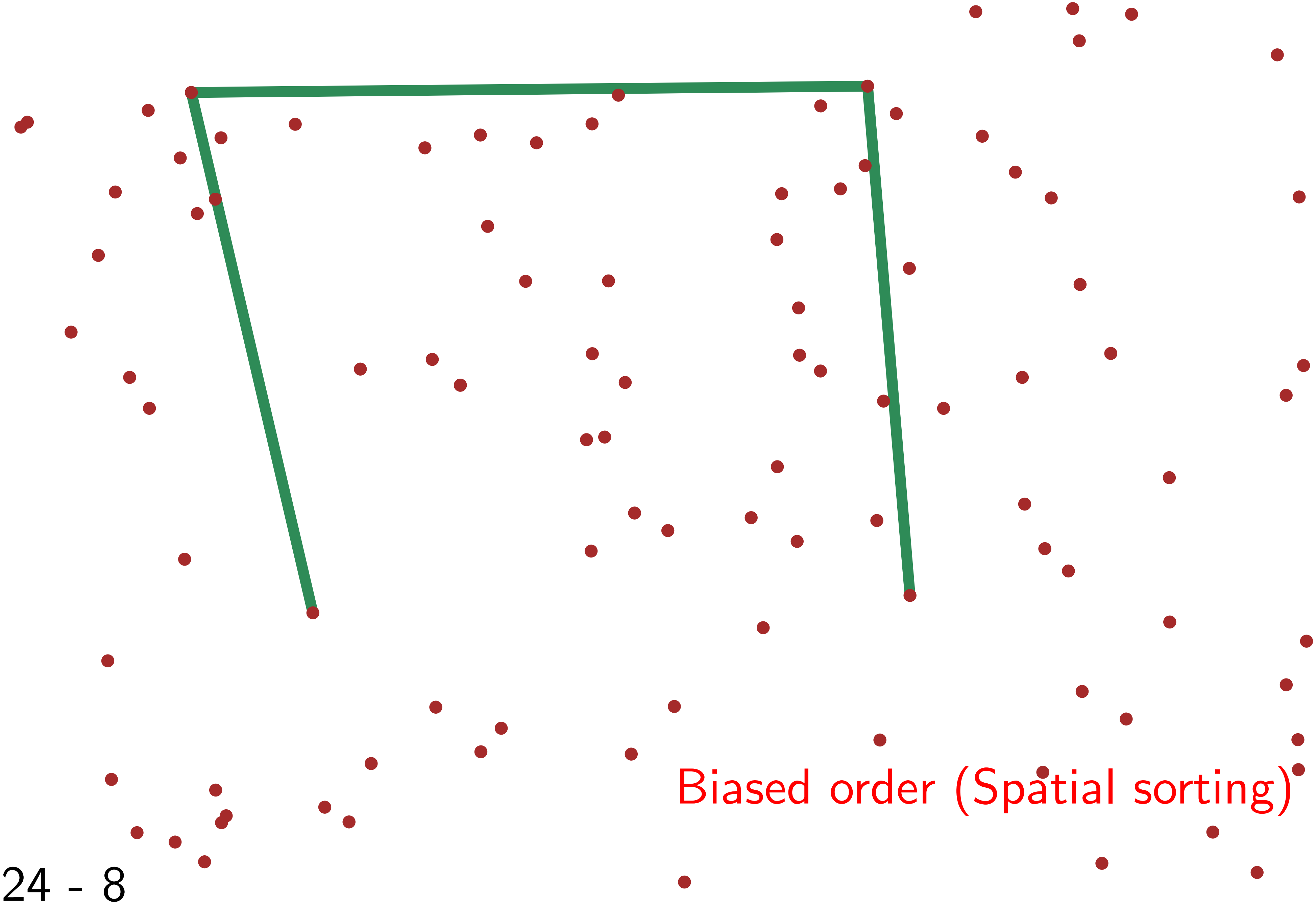
- previous cell in cache memory → faster start
- previous point close → shorter walk

$\Longrightarrow$ memory locality improved

→ speed-up in data structure

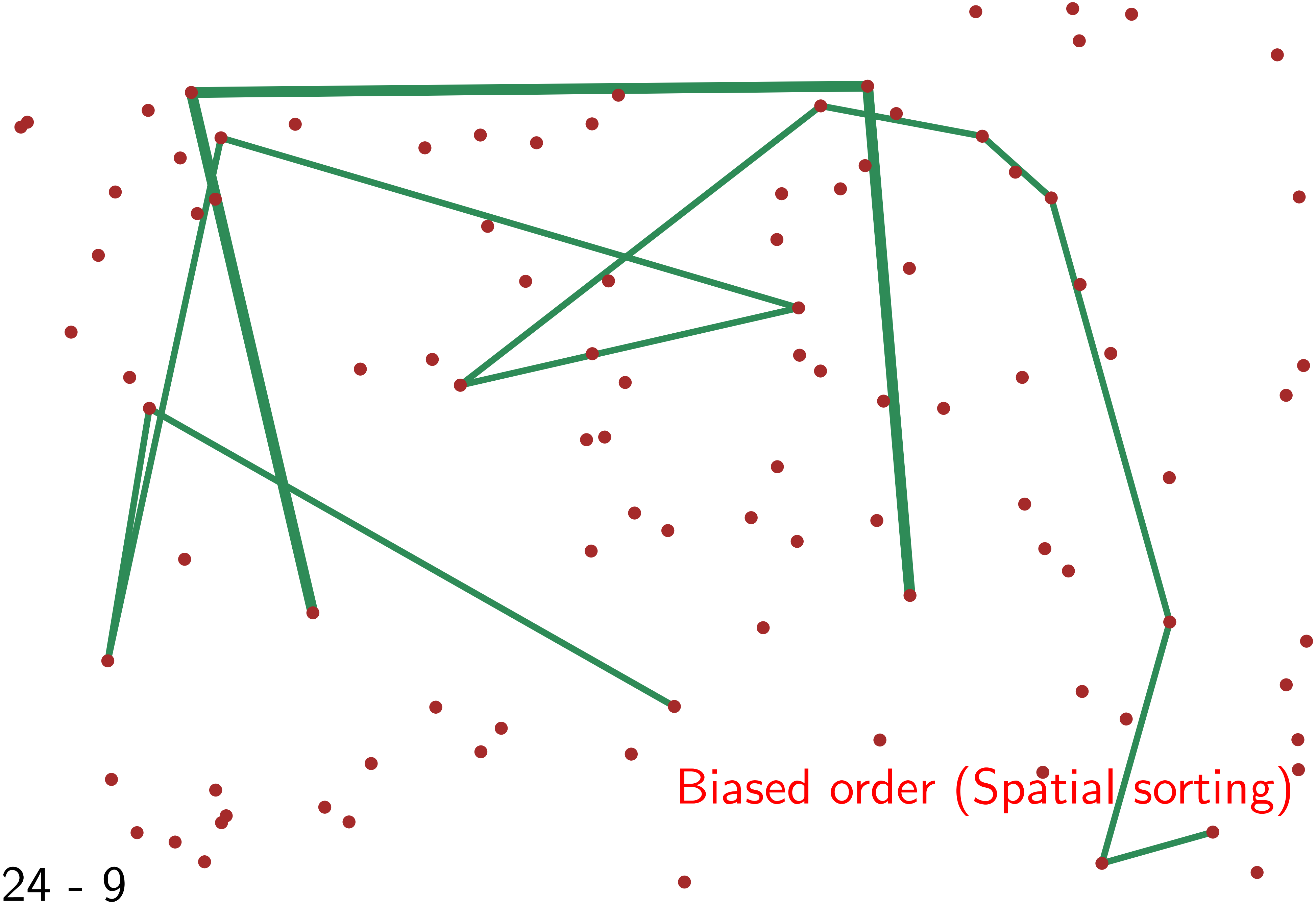
Hilbert order

biased random insertion order



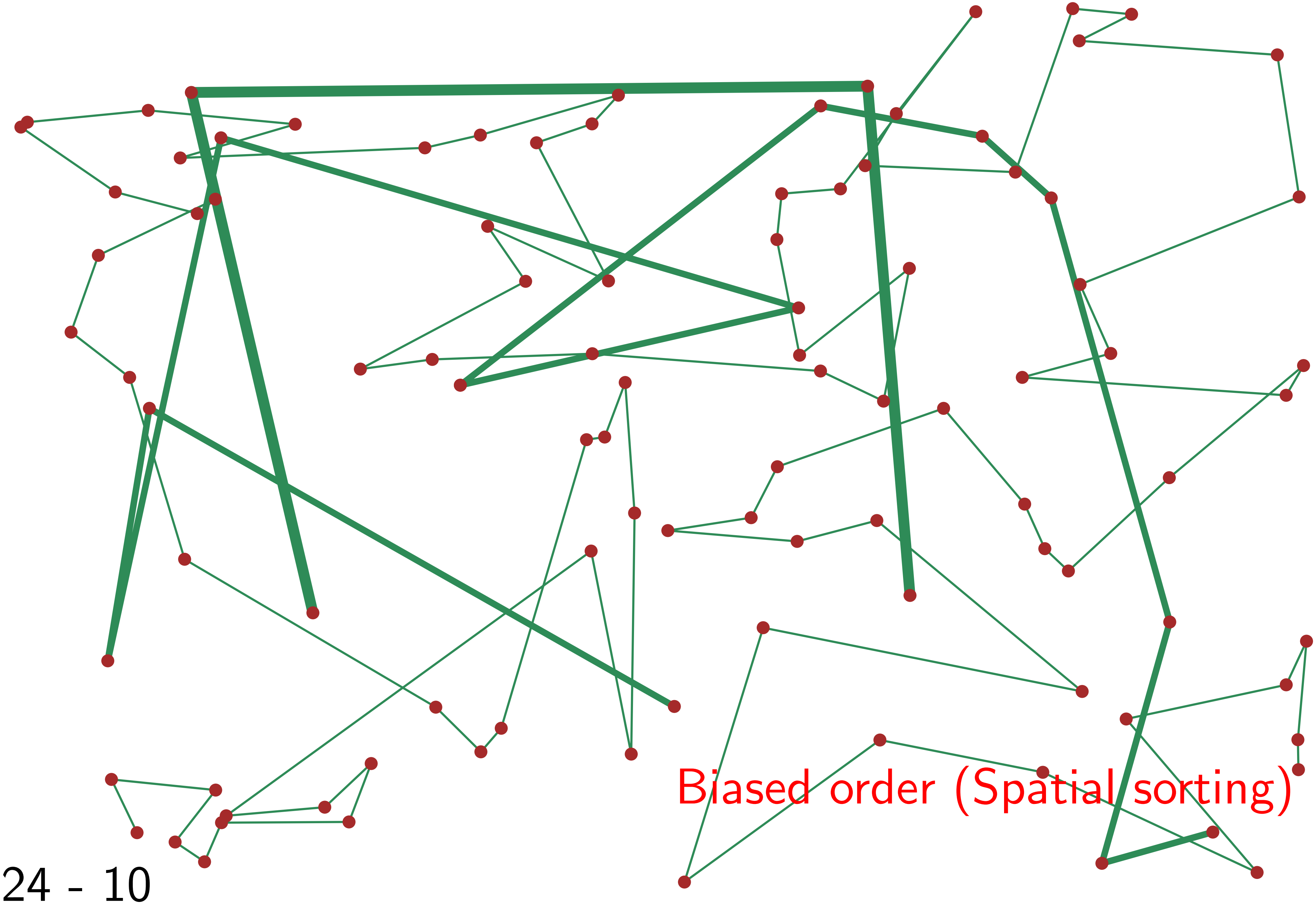
Biased order (Spatial sorting)

biased random insertion order



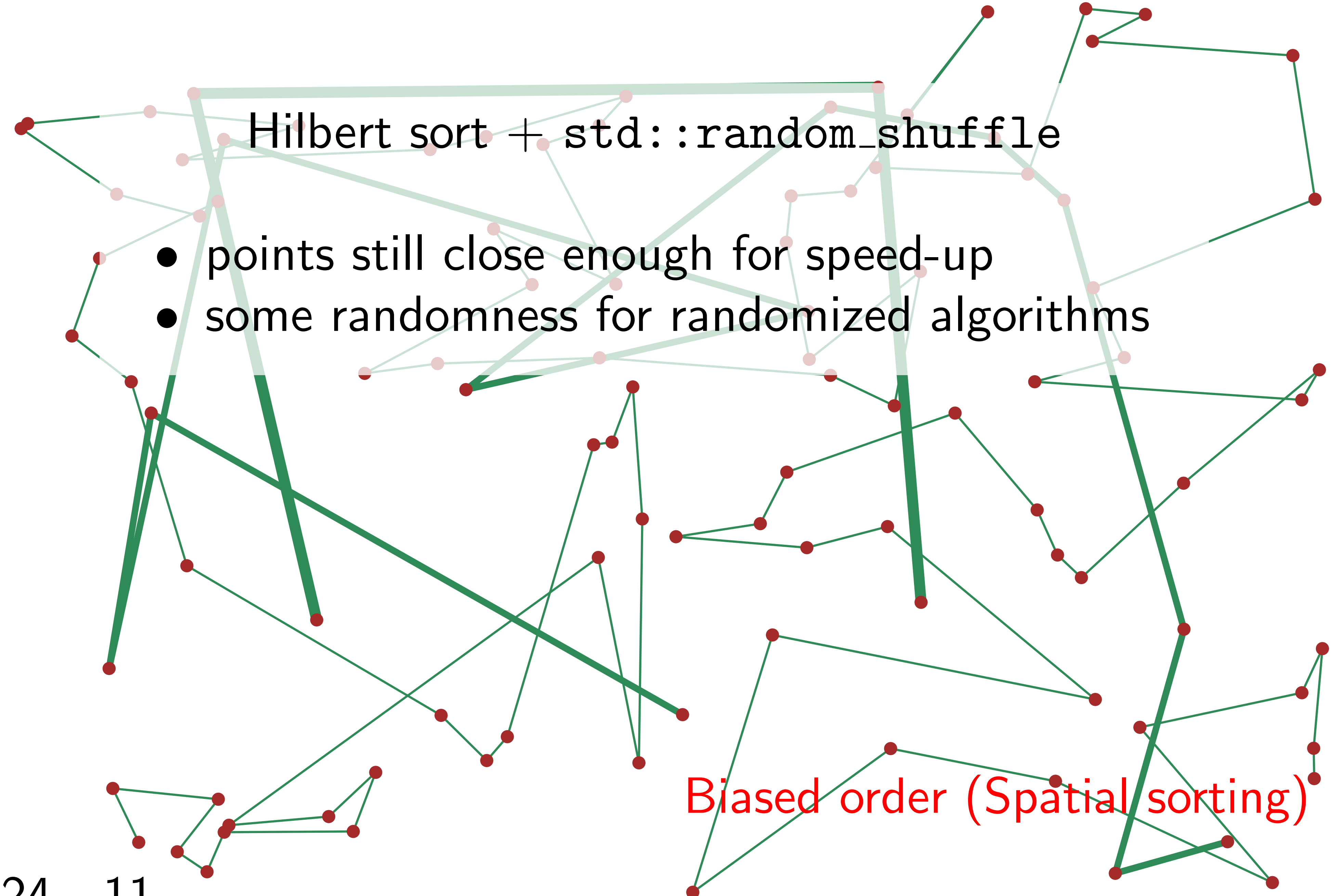
Biased order (Spatial sorting)

biased random insertion order



Biased order (Spatial sorting)

biased random insertion order

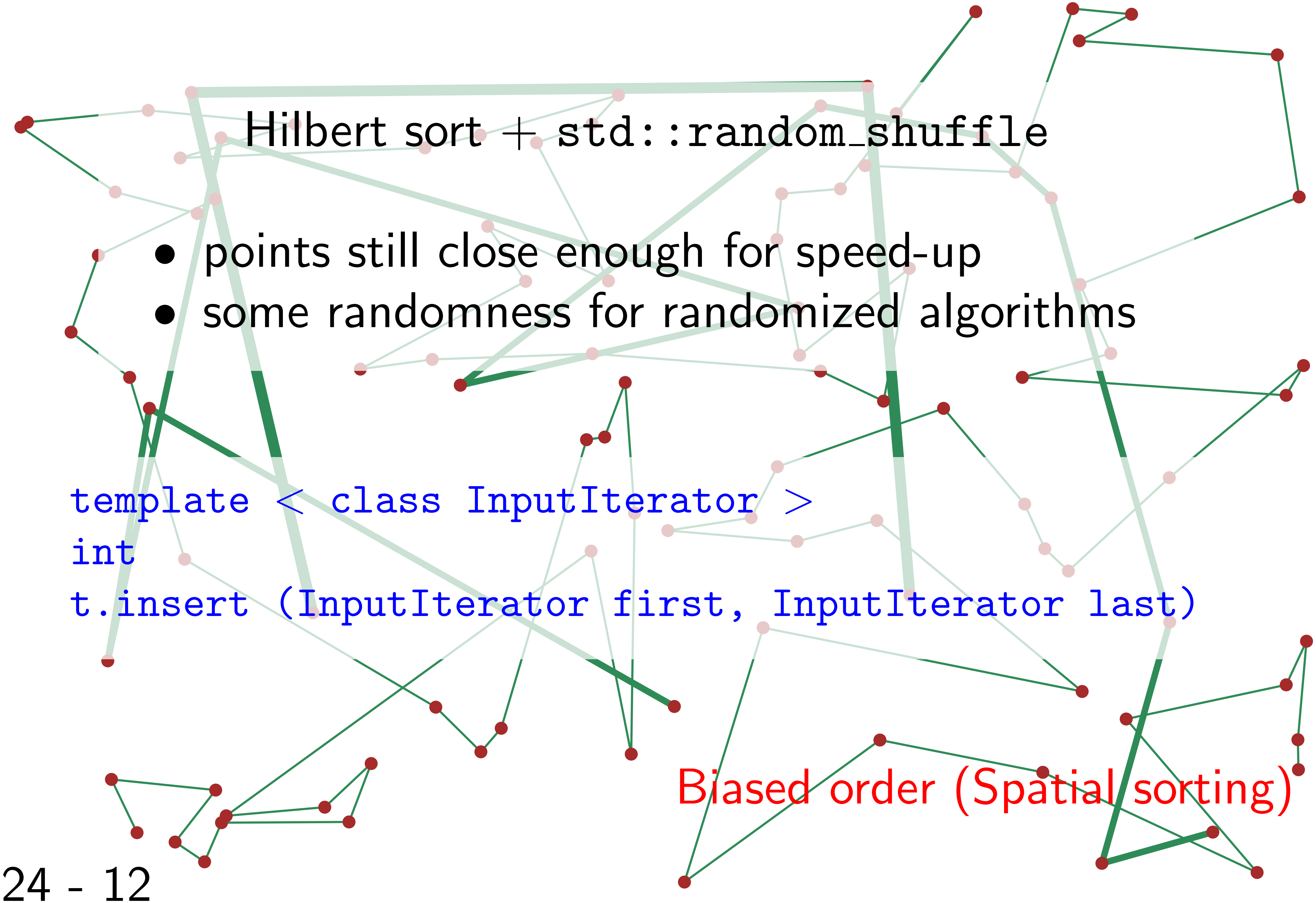


Hilbert sort + `std::random_shuffle`

- points still close enough for speed-up
- some randomness for randomized algorithms

Biased order (Spatial sorting)

biased random insertion order

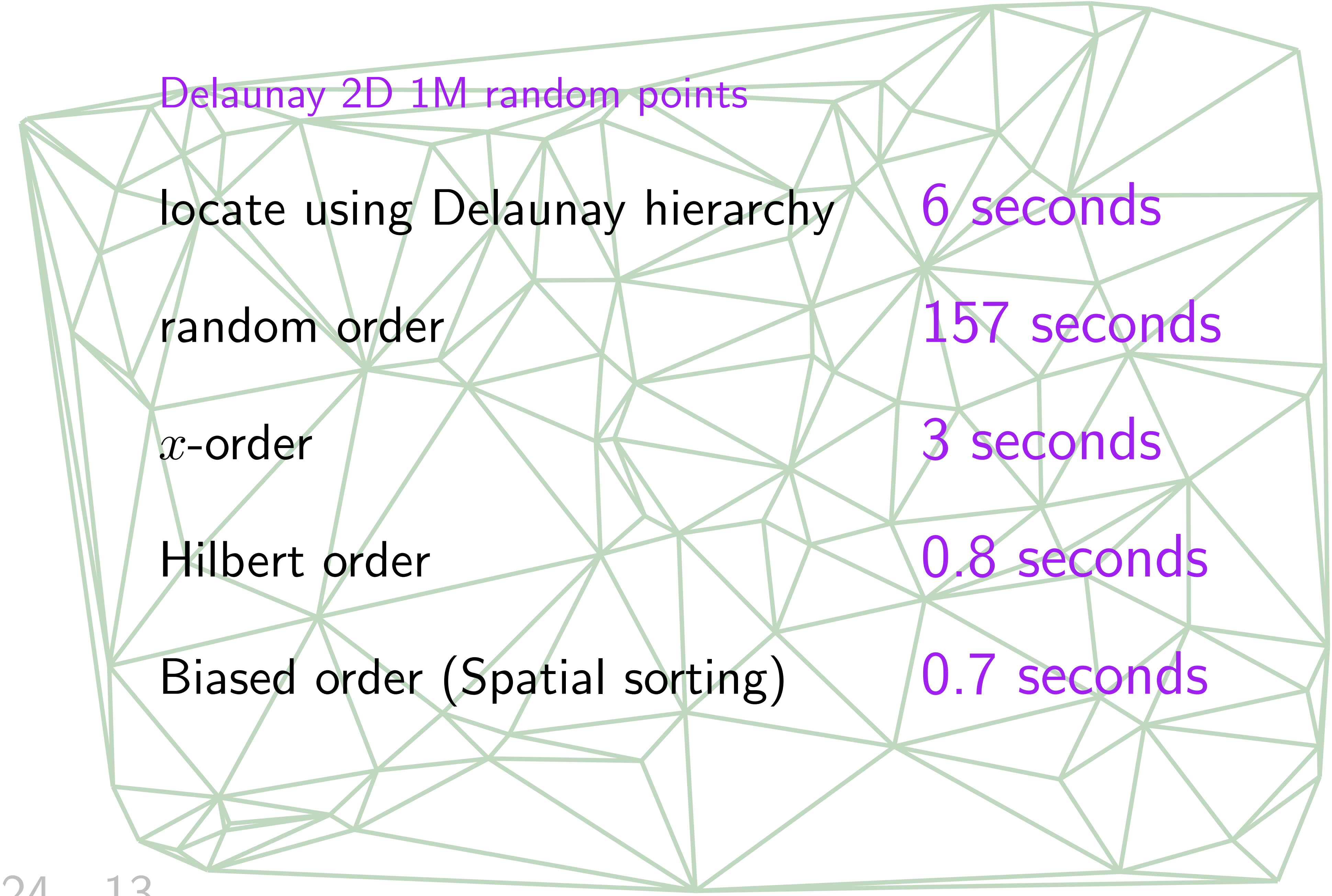


Hilbert sort + `std::random_shuffle`

- points still close enough for speed-up
- some randomness for randomized algorithms

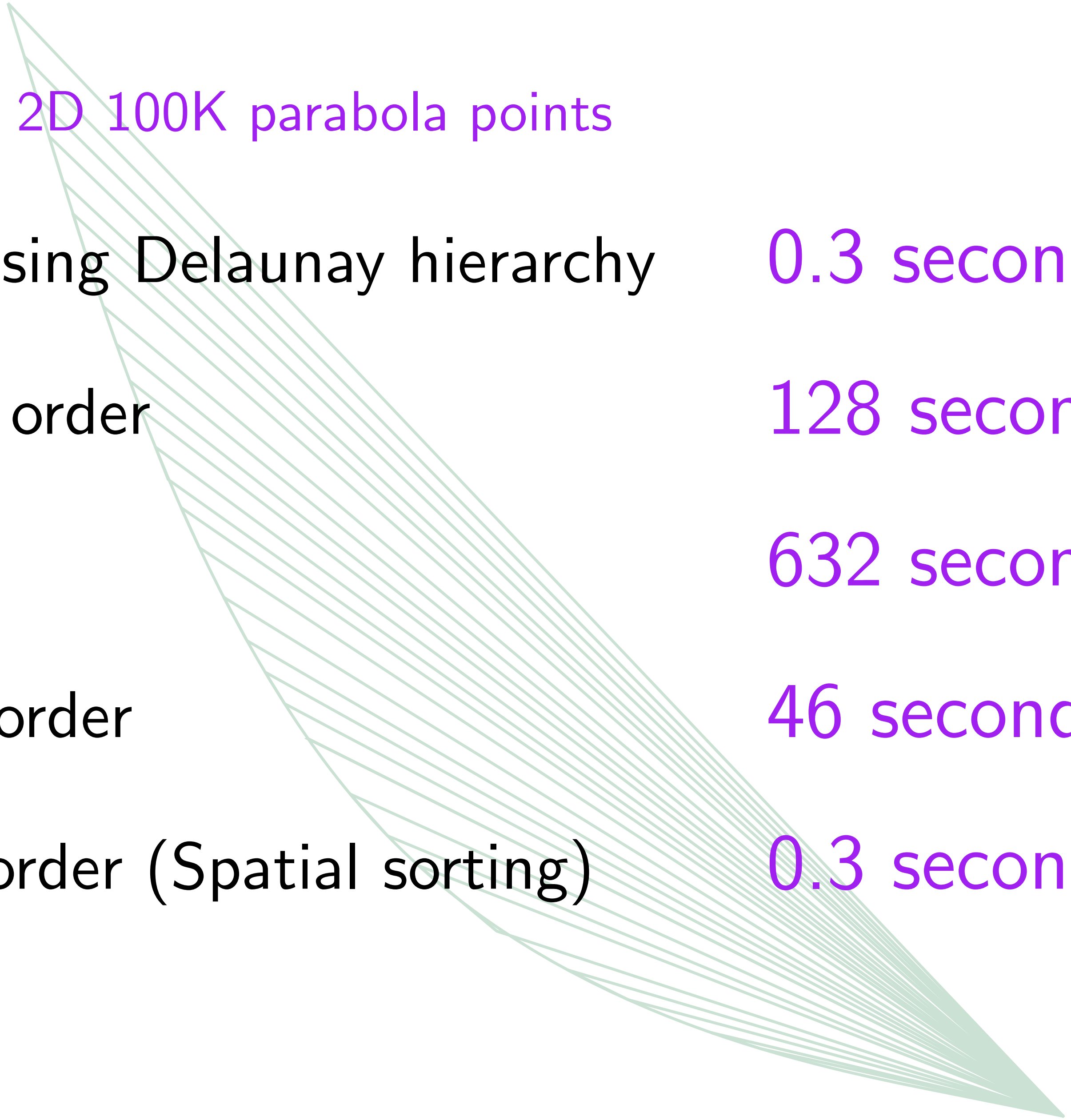
```
template < class InputIterator >
int
t.insert (InputIterator first, InputIterator last)
```

Biased order (Spatial sorting)



Delaunay 2D 1M random points

locate using Delaunay hierarchy	6 seconds
random order	157 seconds
x-order	3 seconds
Hilbert order	0.8 seconds
Biased order (Spatial sorting)	0.7 seconds



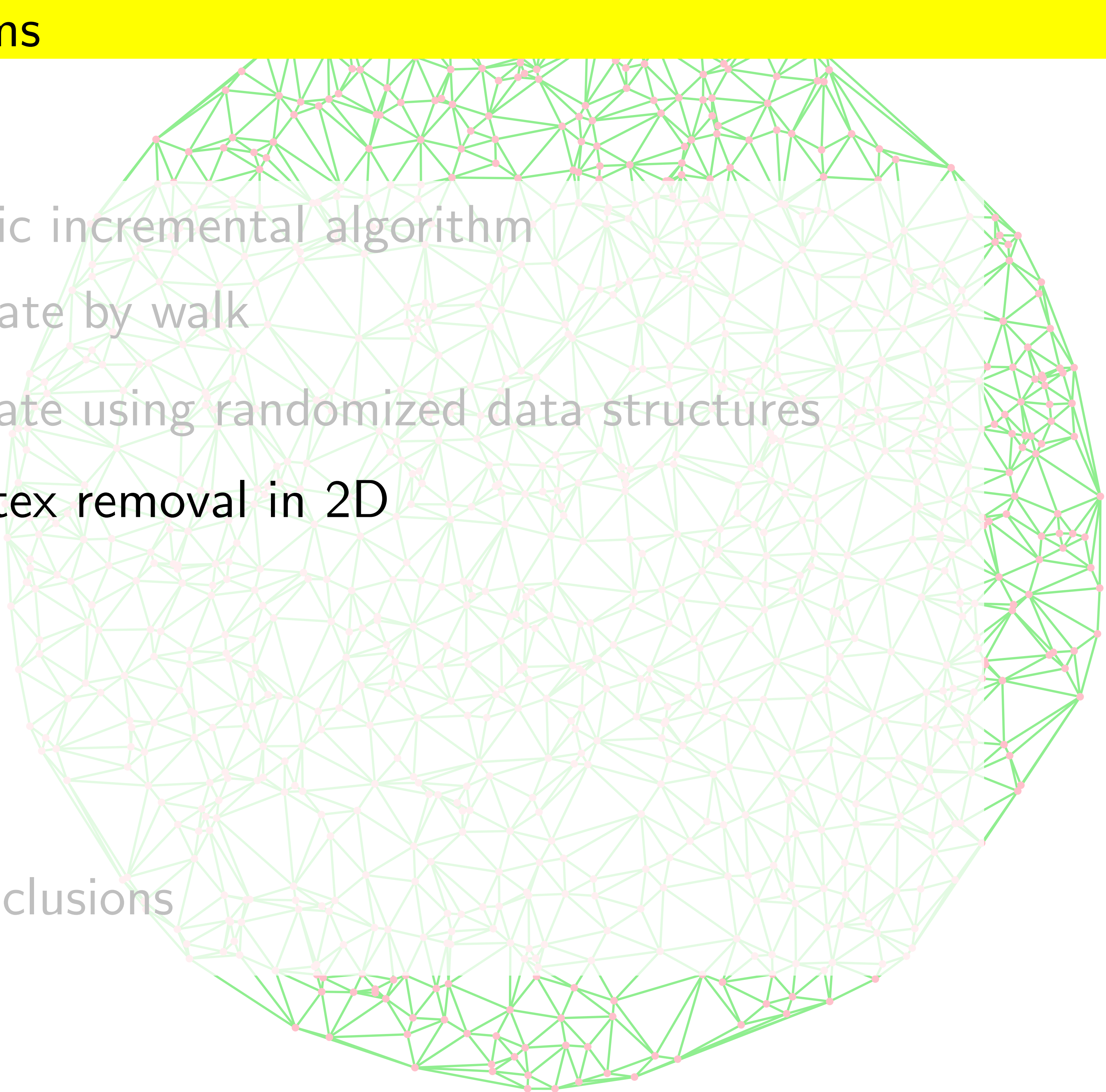
Delaunay 2D 100K parabola points

locate using Delaunay hierarchy	0.3 seconds
random order	128 seconds
x-order	632 seconds
Hilbert order	46 seconds
Biased order (Spatial sorting)	0.3 seconds

Construction of Delaunay 10 M random points

Delaunay tree	$\sim$ 10 mn (estimate)
Delaunay hierarchy	90 seconds
Biased random order	10.6 seconds

Algorithms



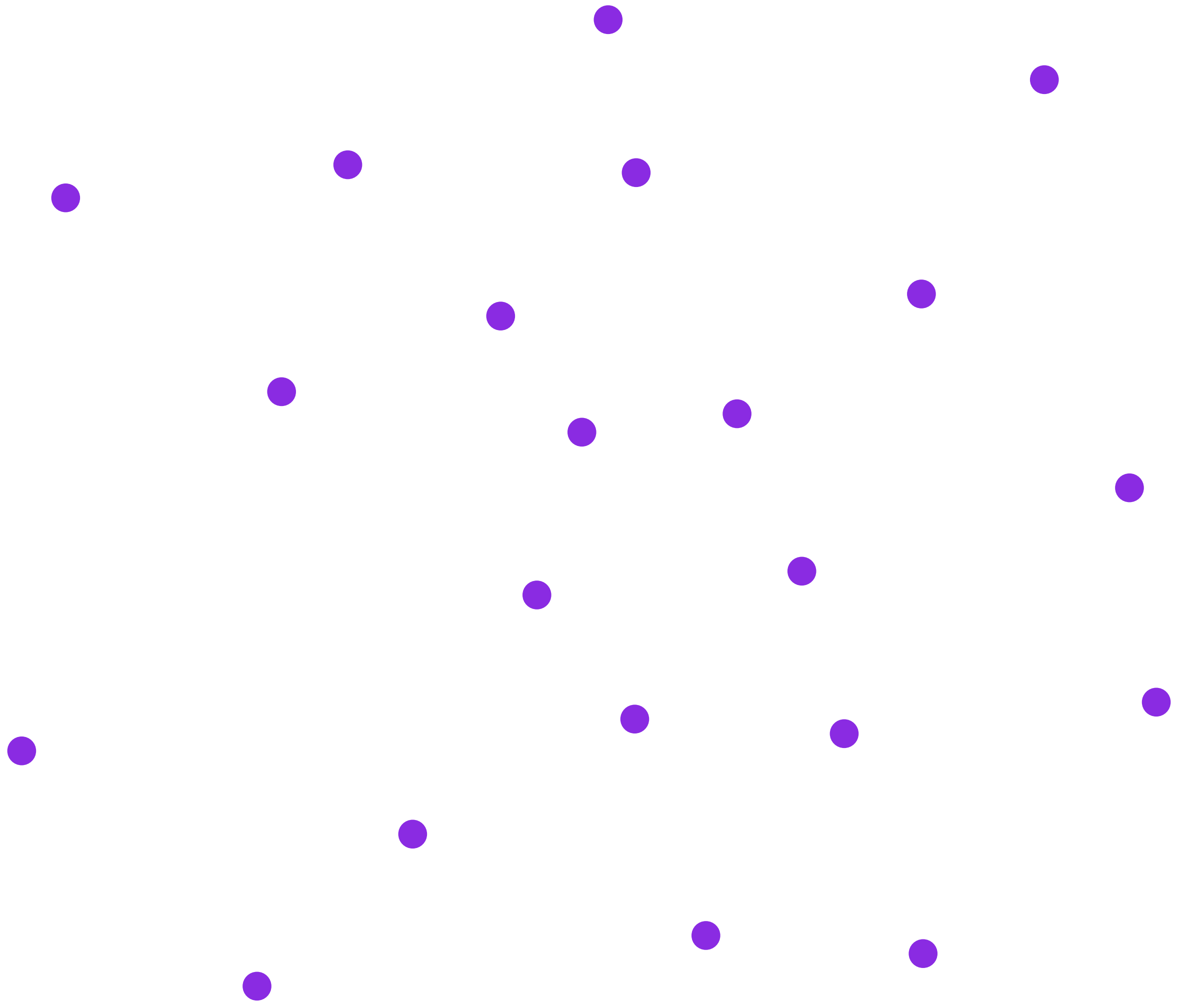
Basic incremental algorithm

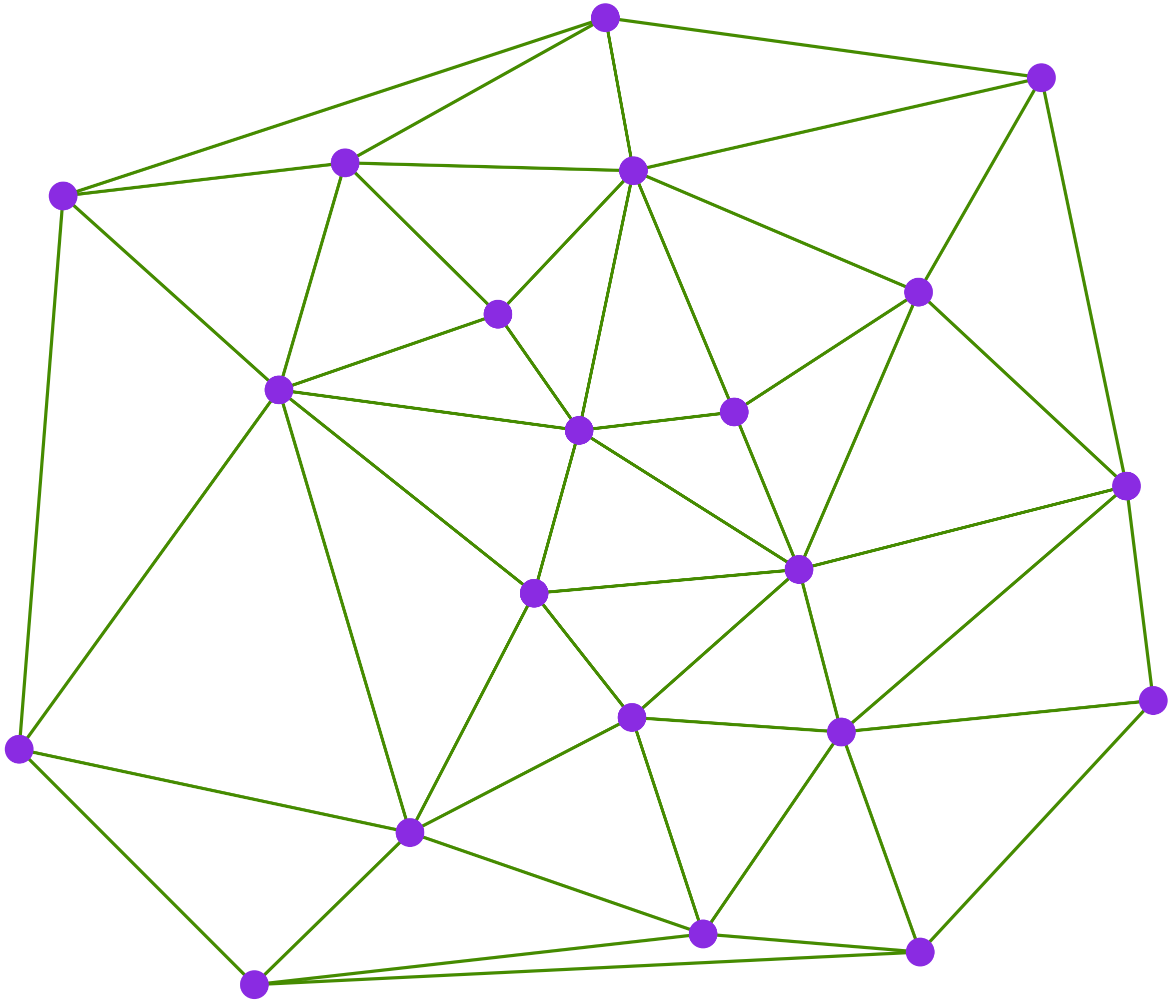
Locate by walk

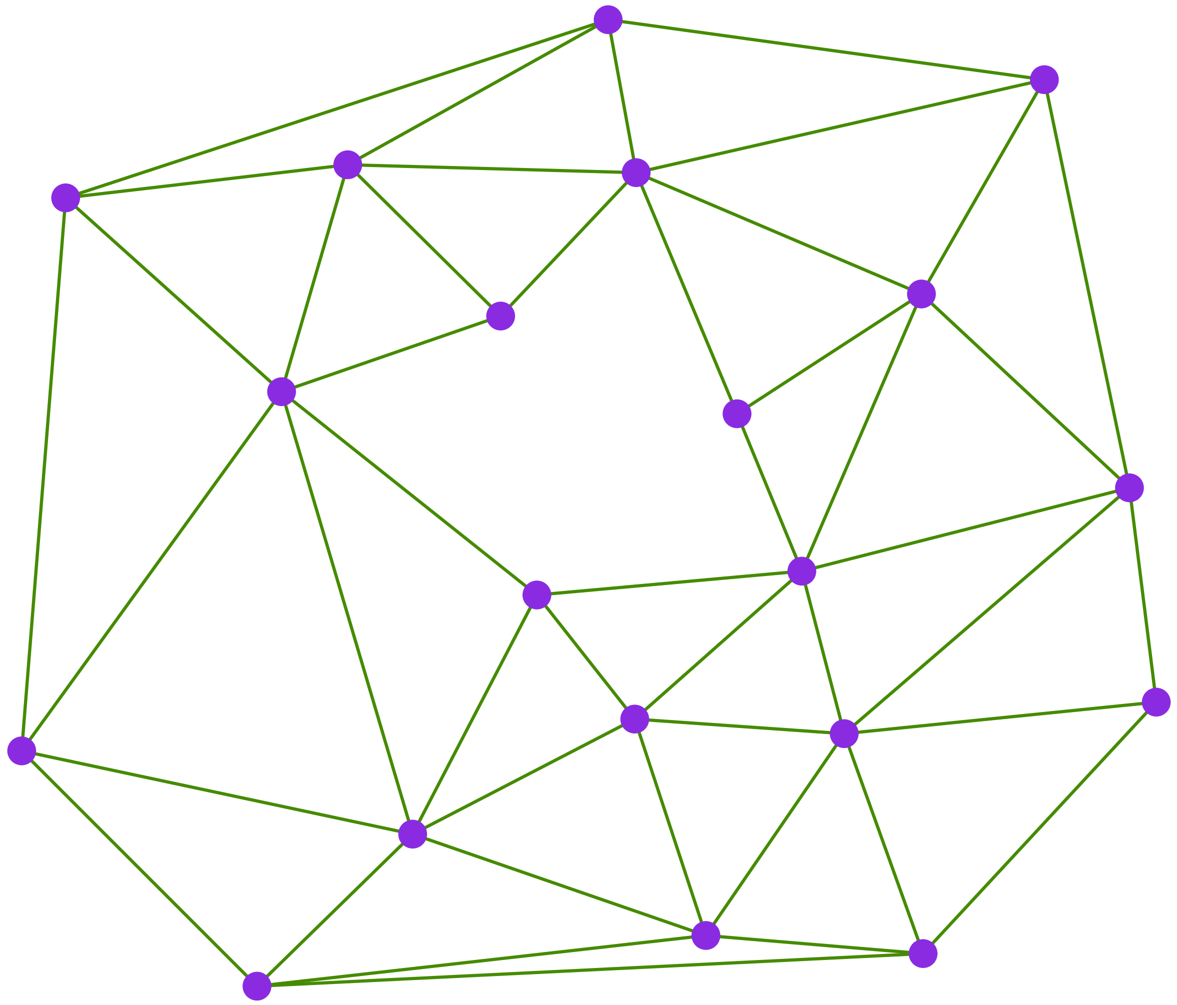
Locate using randomized data structures

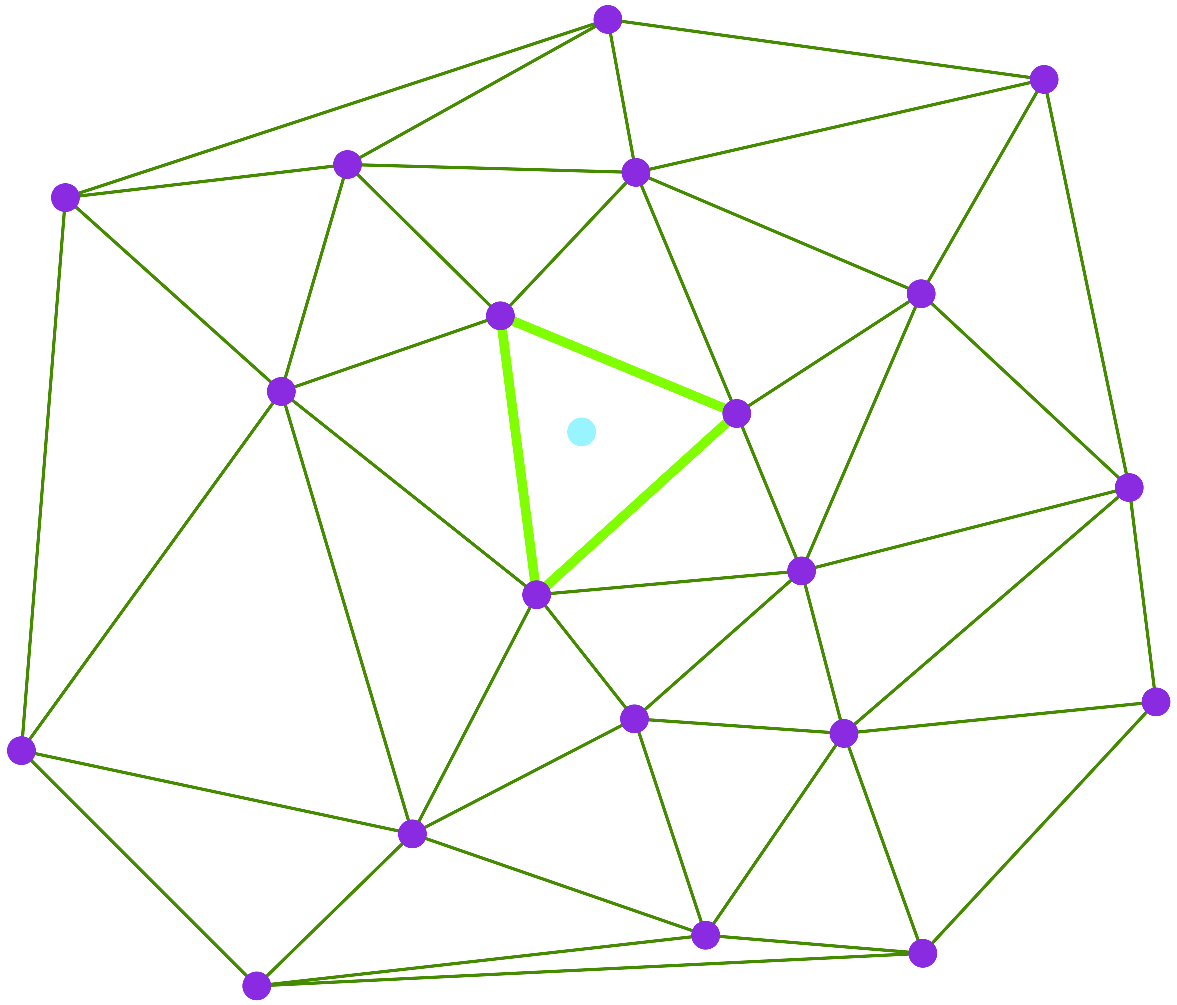
Vertex removal in 2D

Conclusions

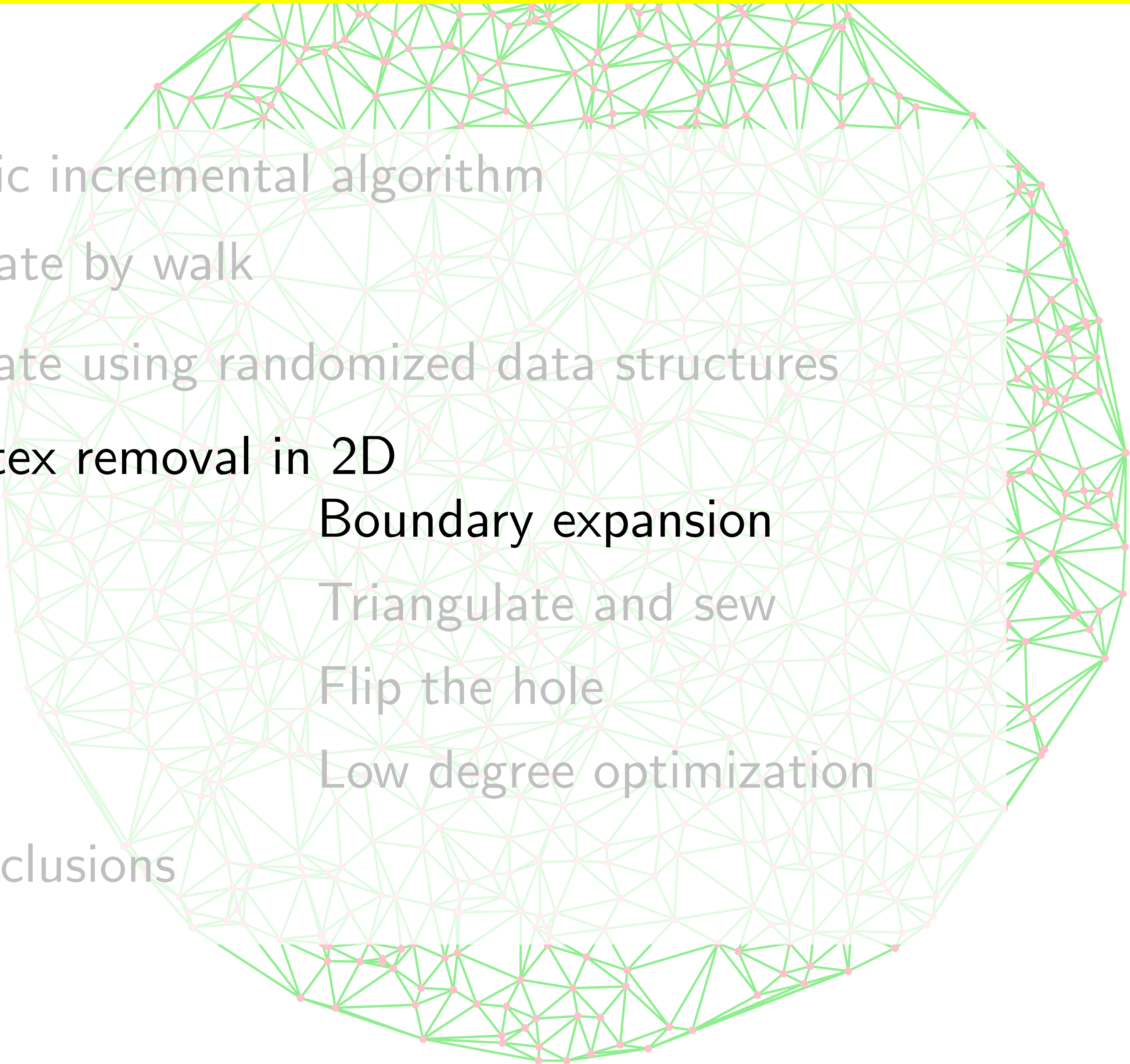








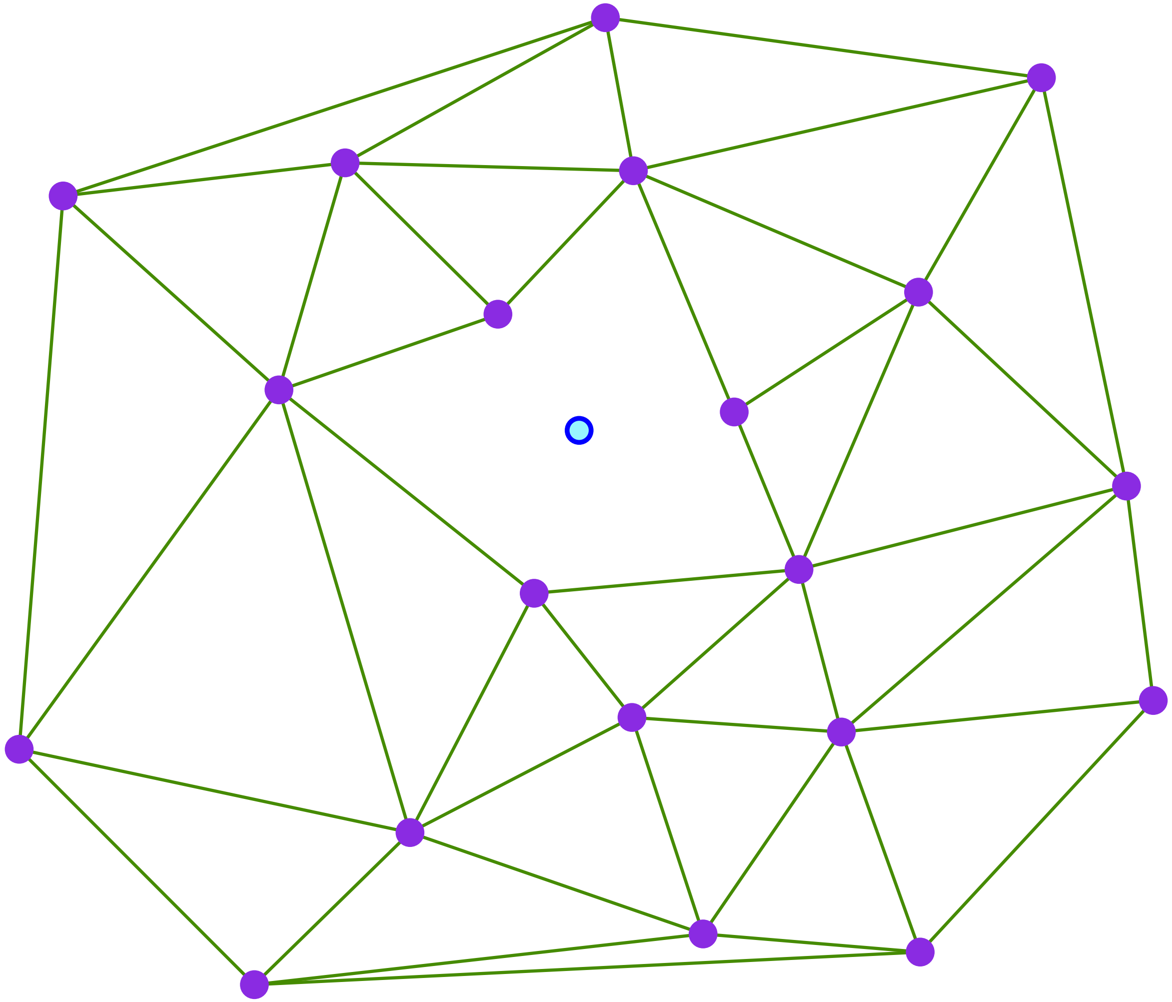
Algorithms



- Basic incremental algorithm
- Locate by walk
- Locate using randomized data structures
- Vertex removal in 2D
  - Boundary expansion
  - Triangulate and sew
  - Flip the hole
  - Low degree optimization
- Conclusions

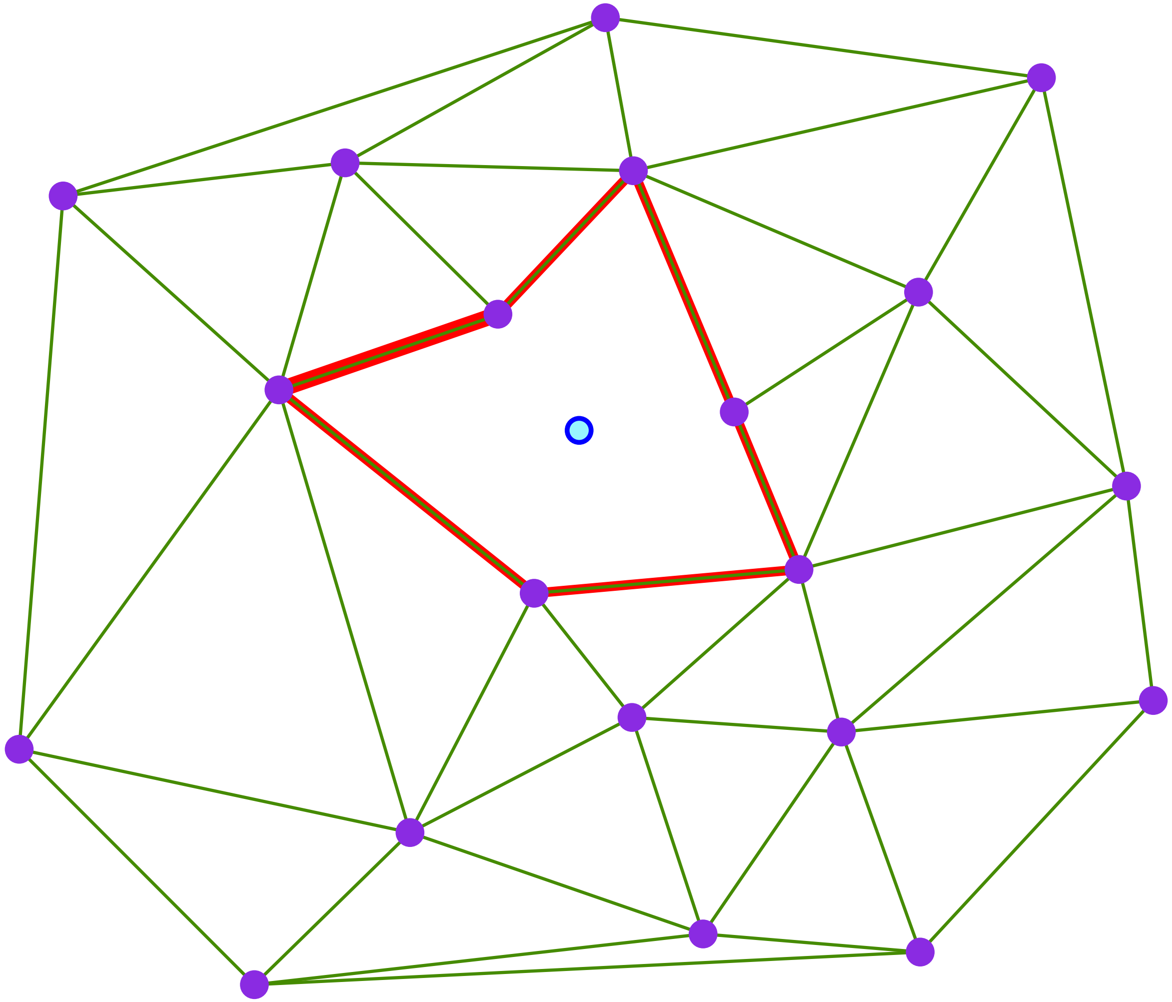
boundary expansion

release 3.5, 2D implementation



boundary expansion

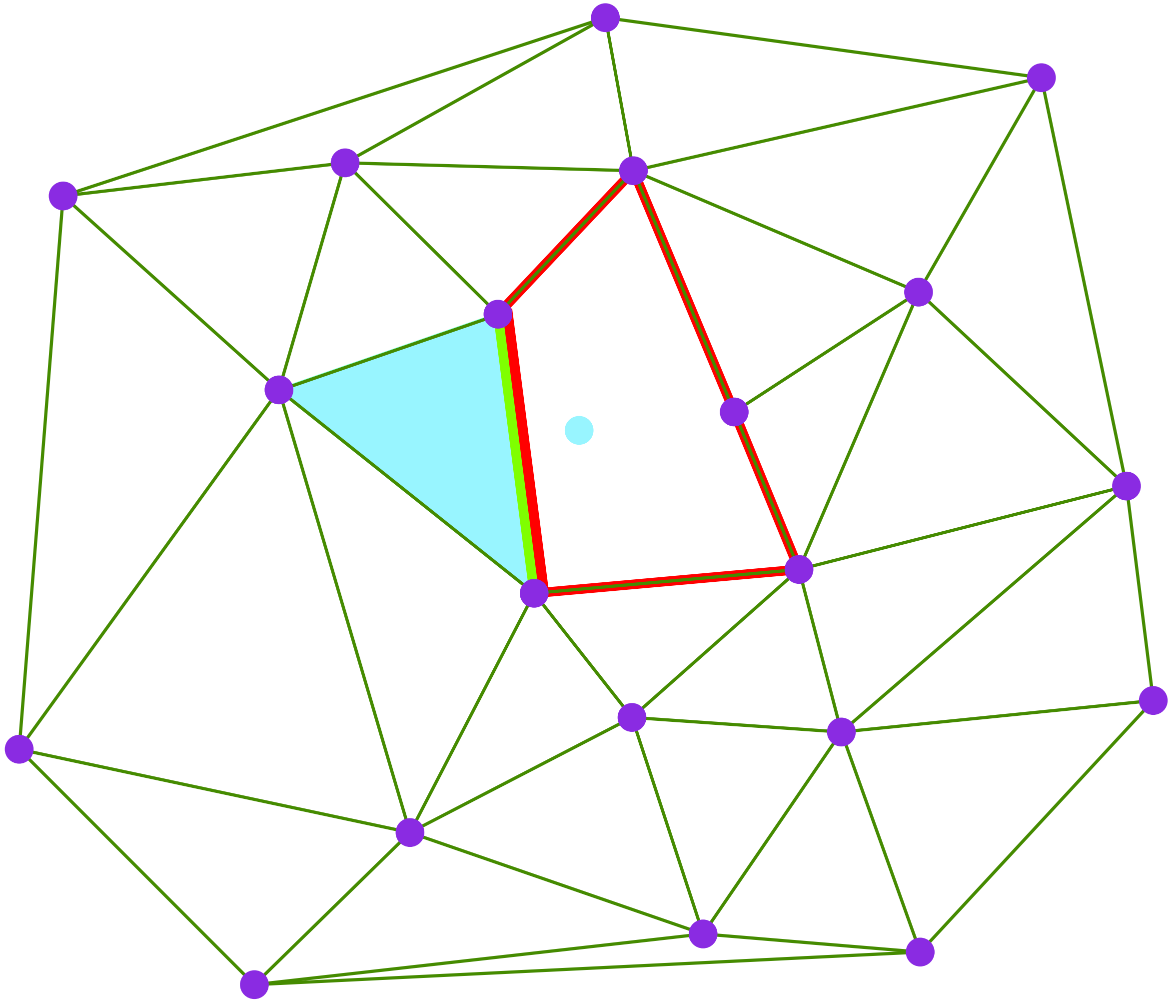
release 3.5, 2D implementation



hole boundary = queue

boundary expansion

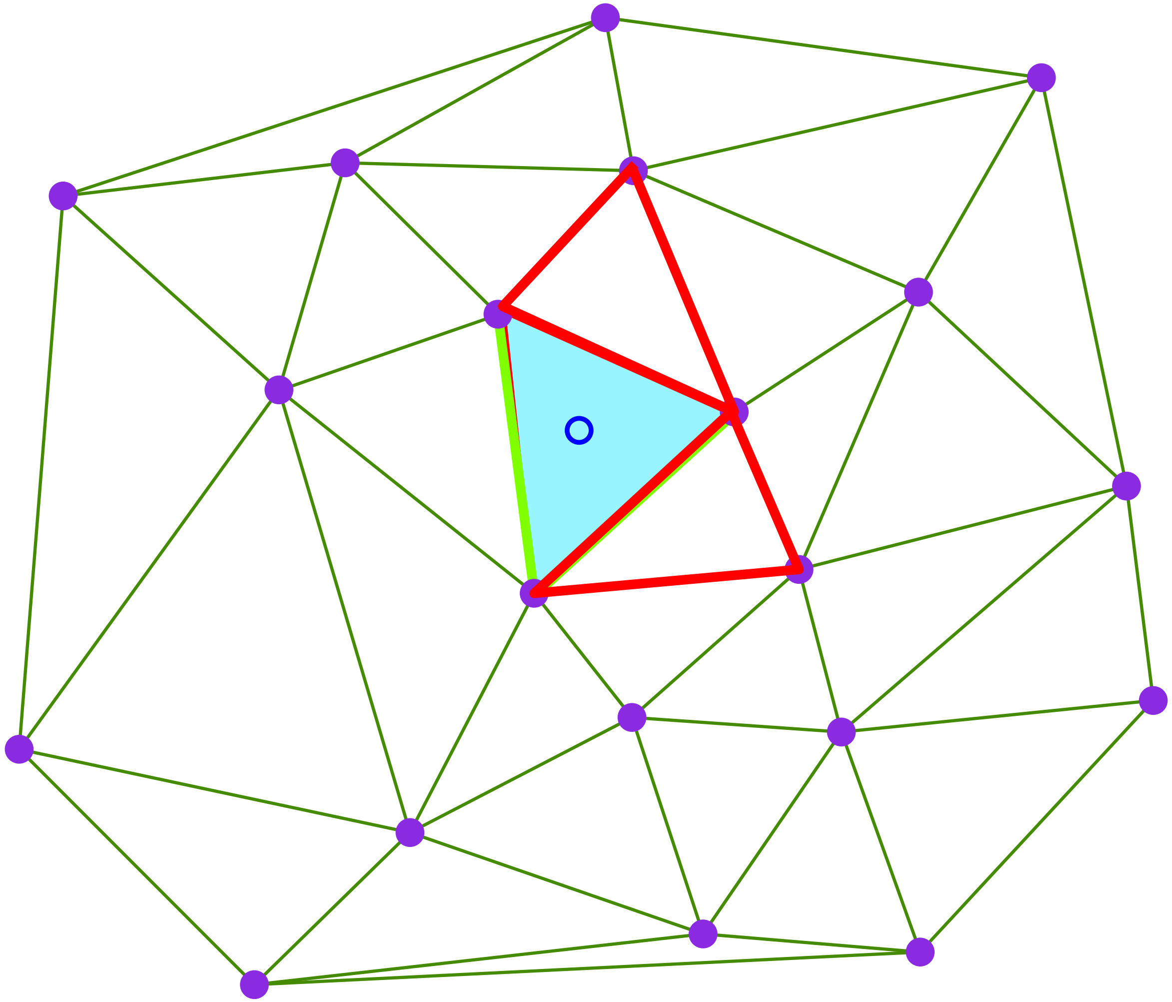
release 3.5, 2D implementation



find new incident triangle in linear time

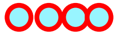
boundary expansion

release 3.5, 2D implementation

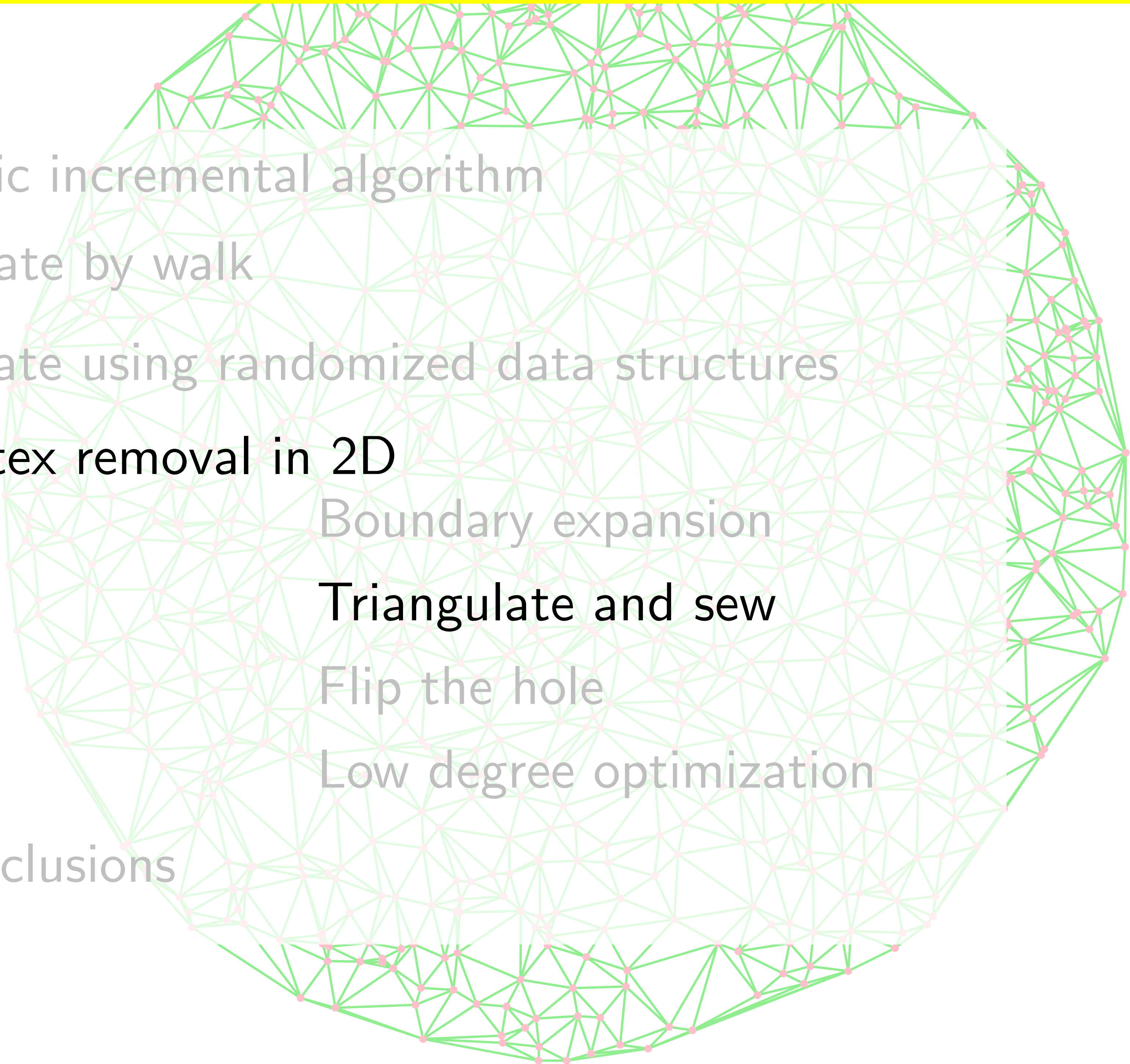


boundary expansion

release 3.5, 2D implementation



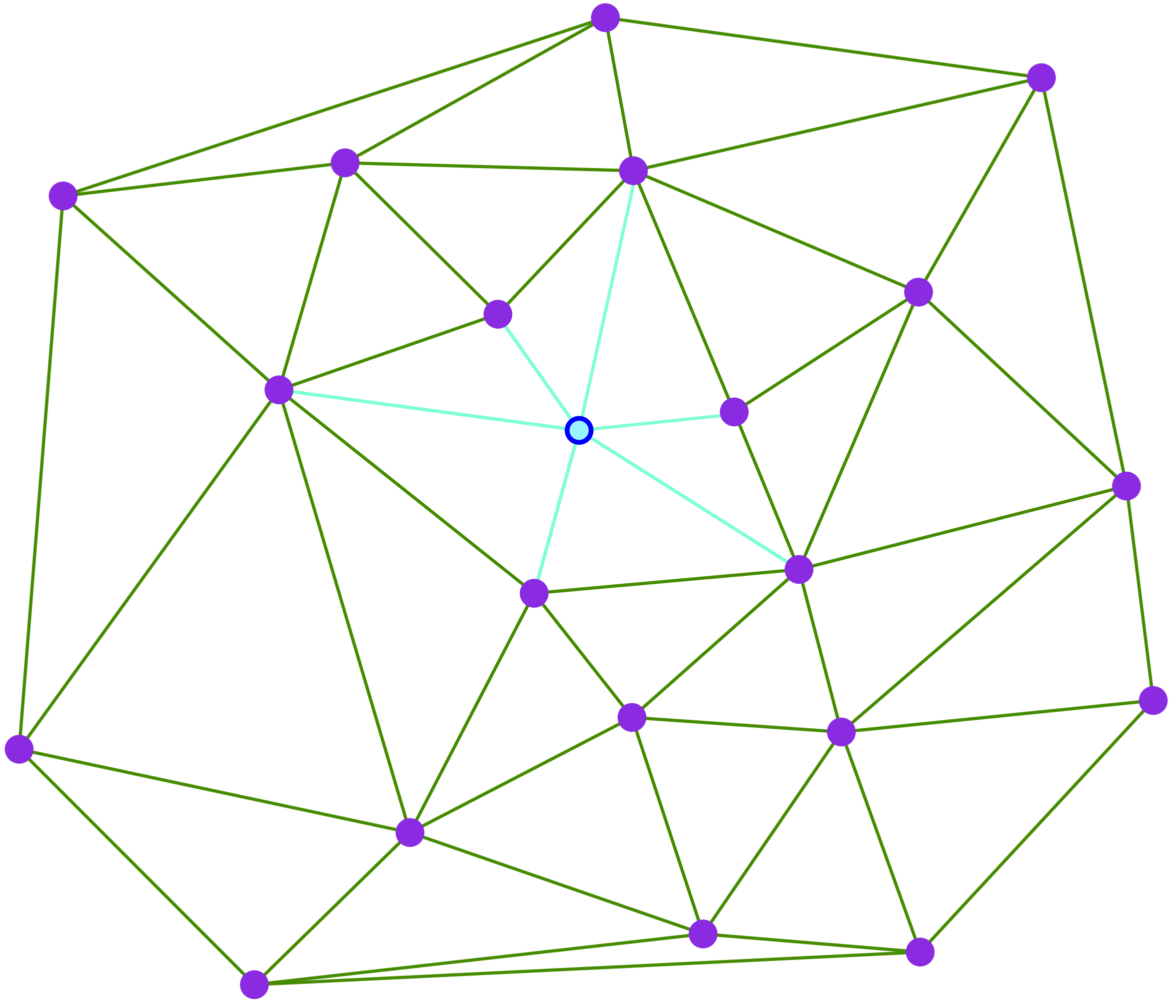
Algorithms



- Basic incremental algorithm
- Locate by walk
- Locate using randomized data structures
- Vertex removal in 2D
  - Boundary expansion
  - Triangulate and sew
  - Flip the hole
  - Low degree optimization
- Conclusions

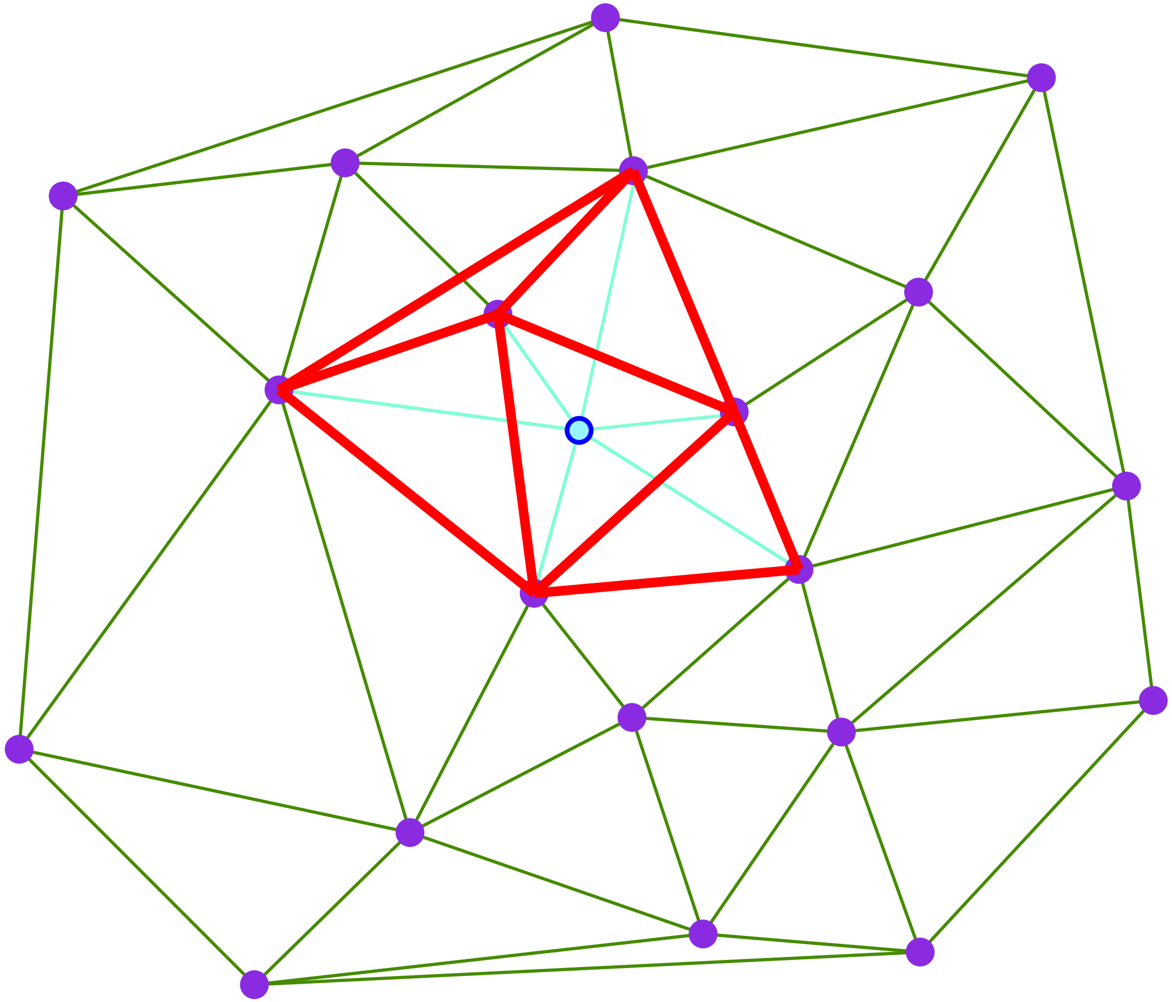
triangulate and sew

current CGAL implementation in 3D



triangulate and sew

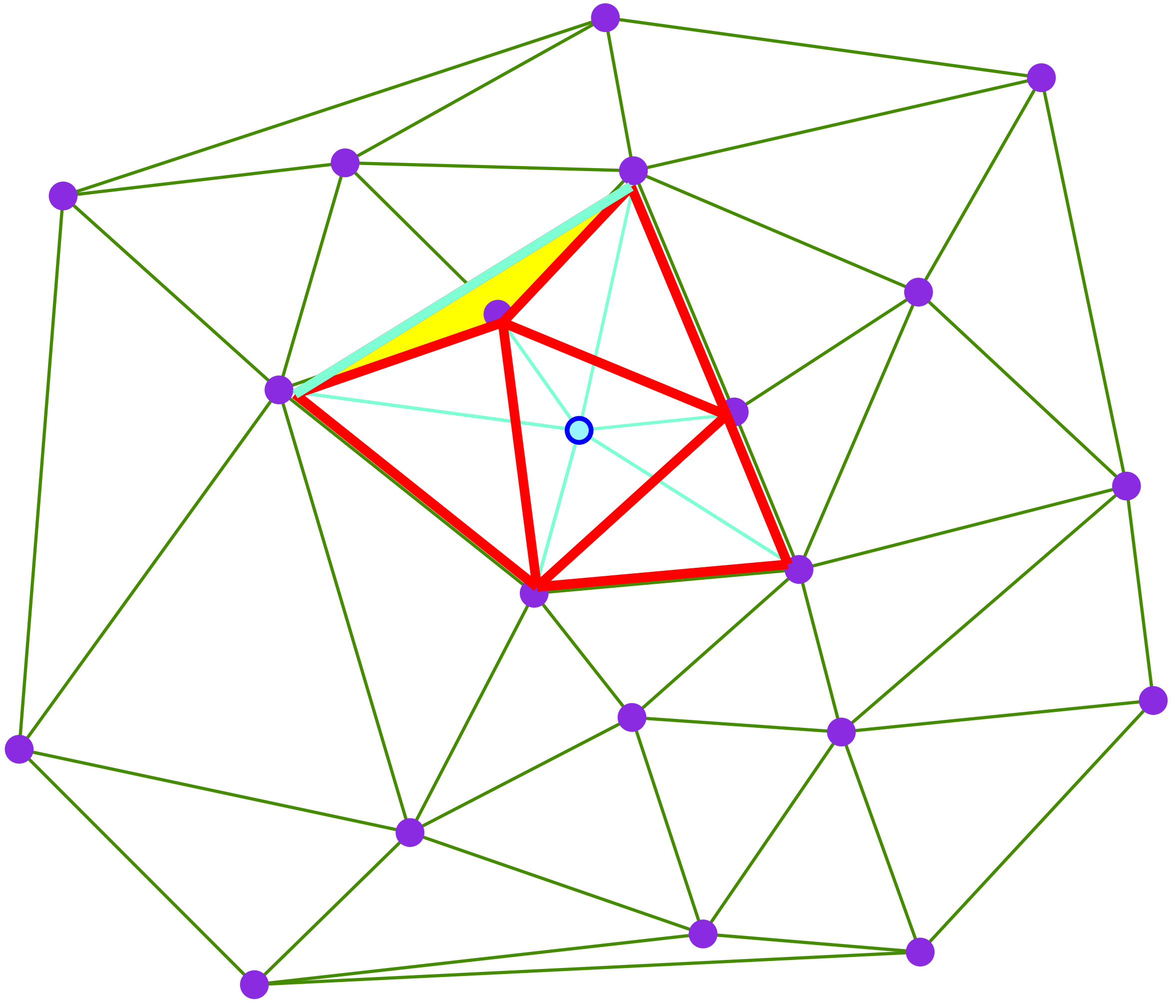
current CGAL implementation in 3D



Delaunay of neighbors

triangulate and sew

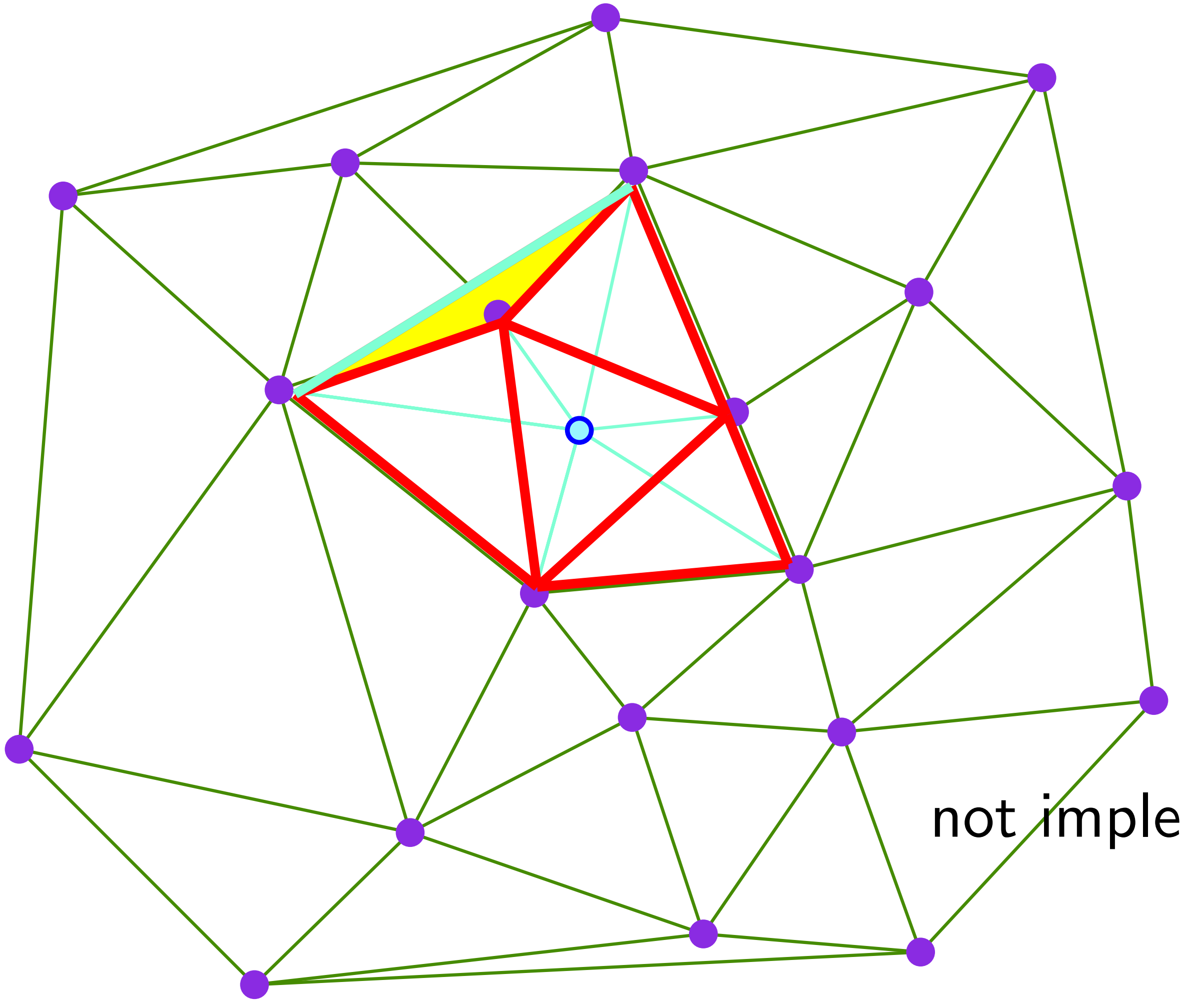
current CGAL implementation in 3D



delete extra triangles and sew

triangulate and sew

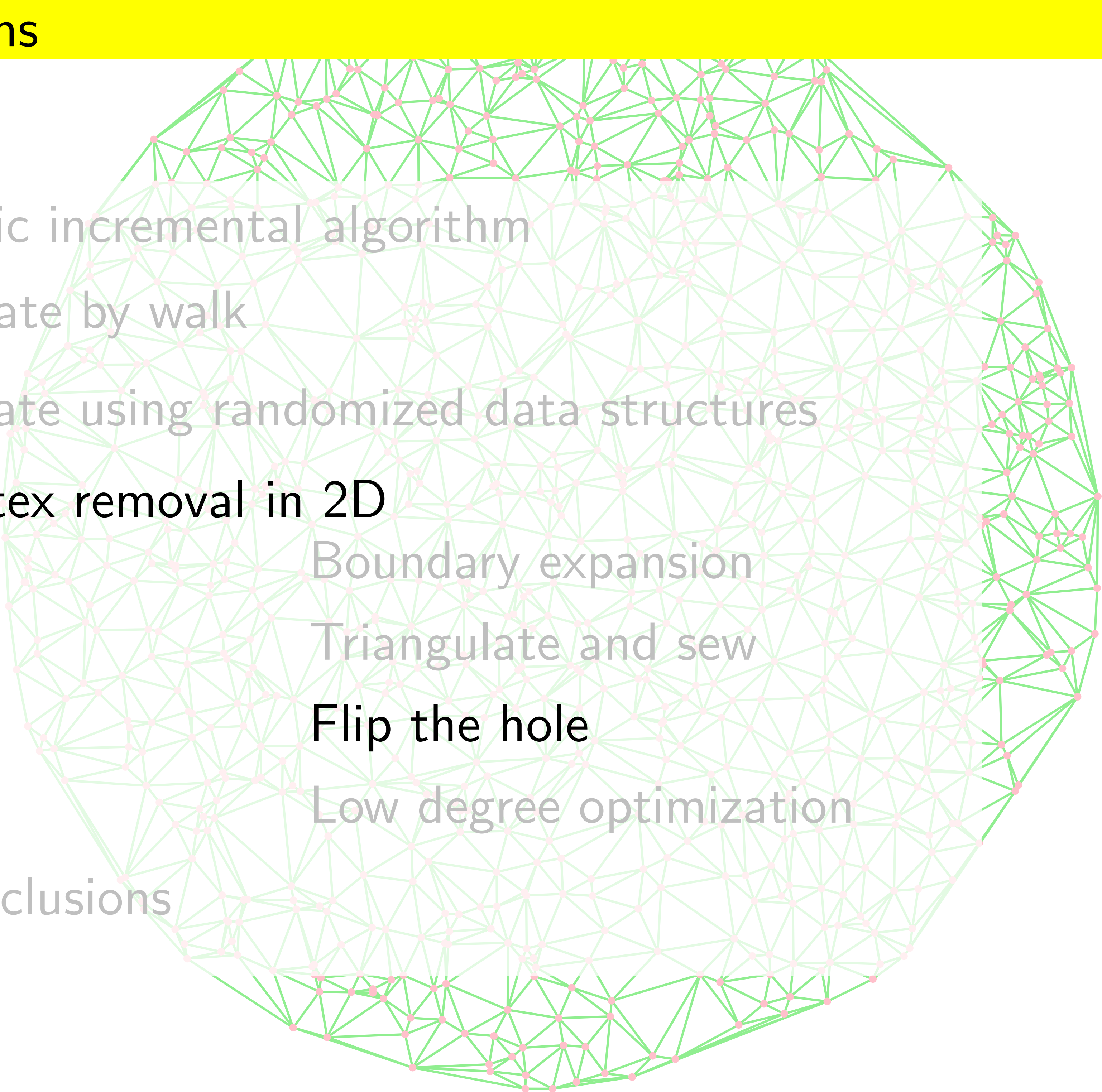
current CGAL implementation in 3D



not implemented in 2D

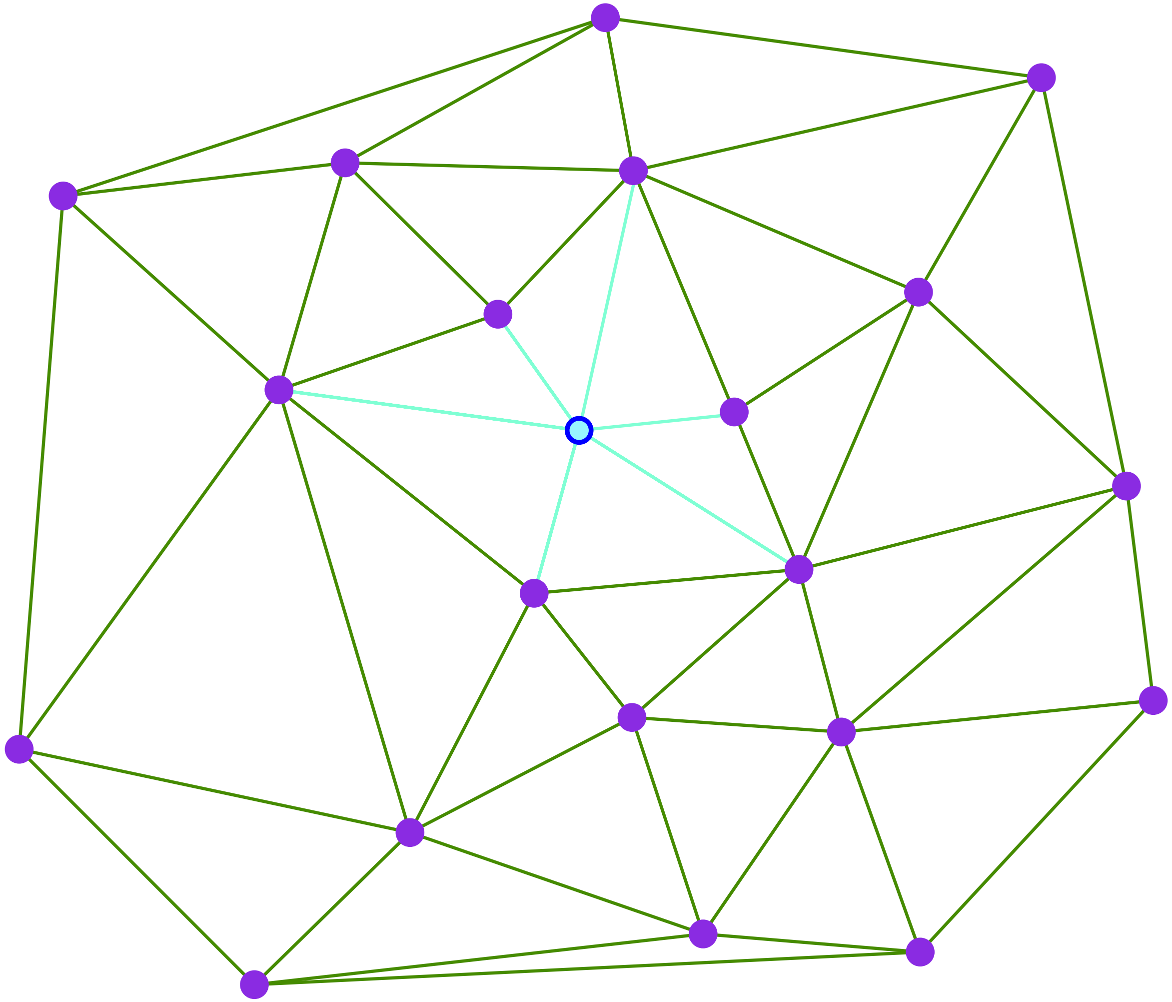
delete extra triangles and sew

Algorithms

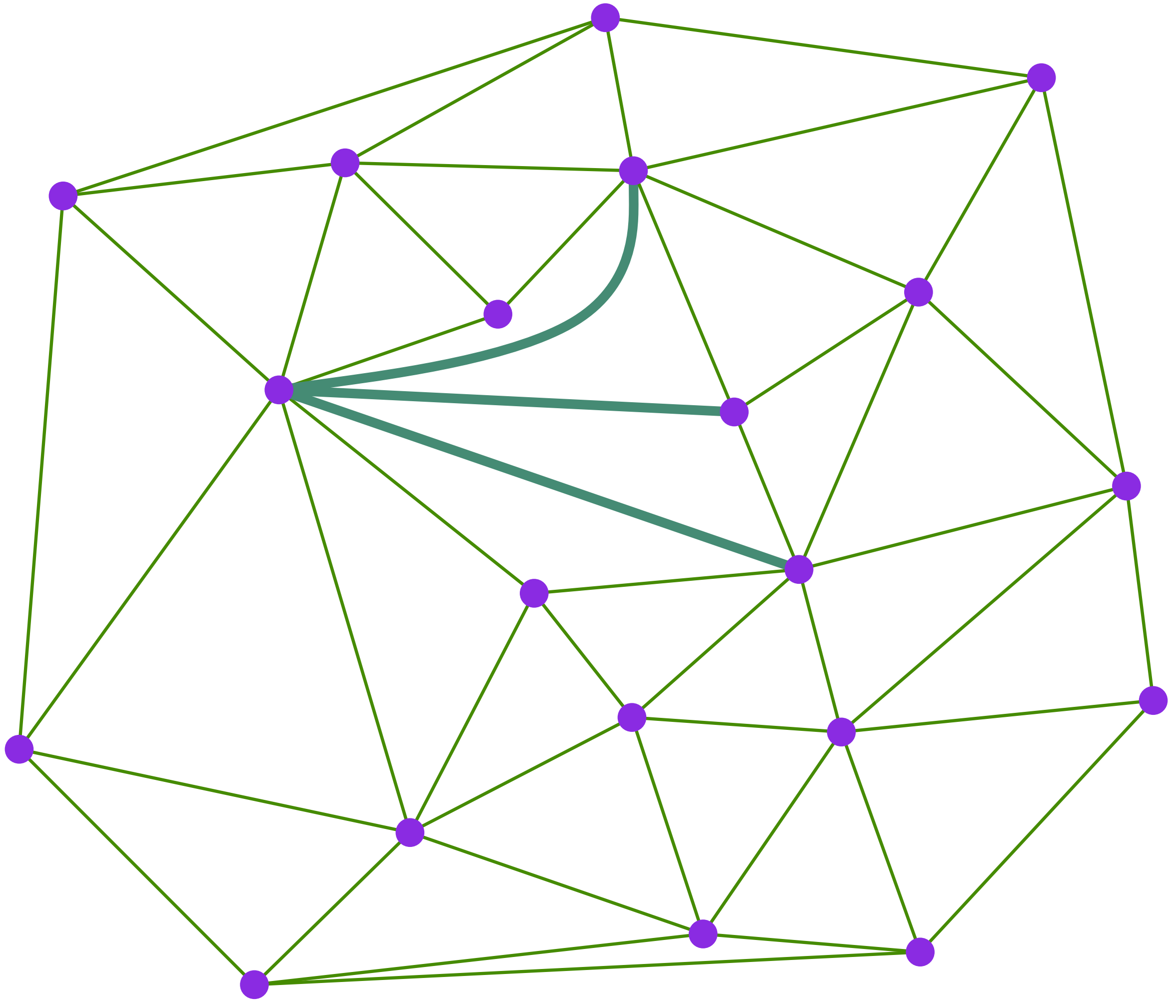


- Basic incremental algorithm
- Locate by walk
- Locate using randomized data structures
- Vertex removal in 2D
  - Boundary expansion
  - Triangulate and sew
  - Flip the hole
  - Low degree optimization
- Conclusions

flip the hole

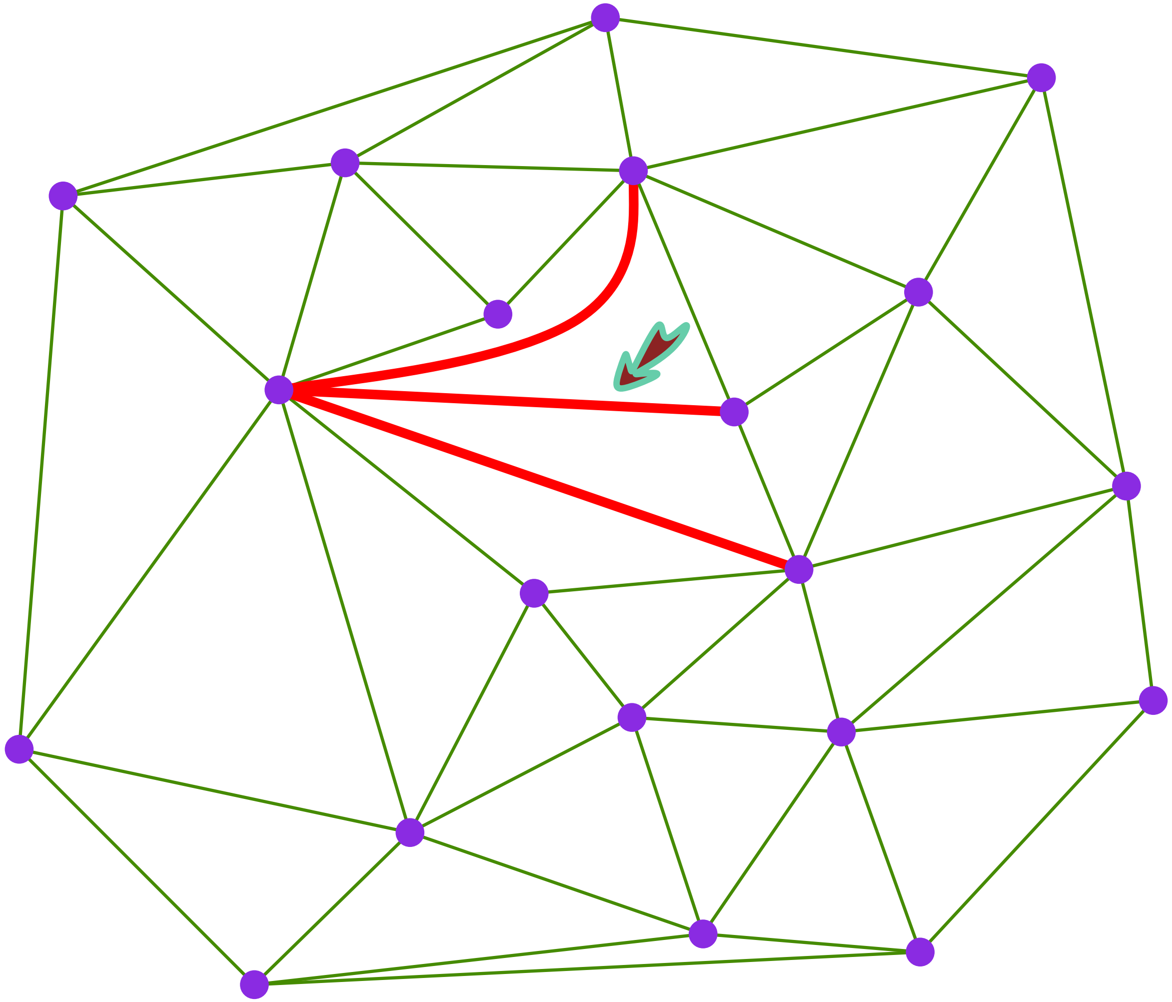


flip the hole



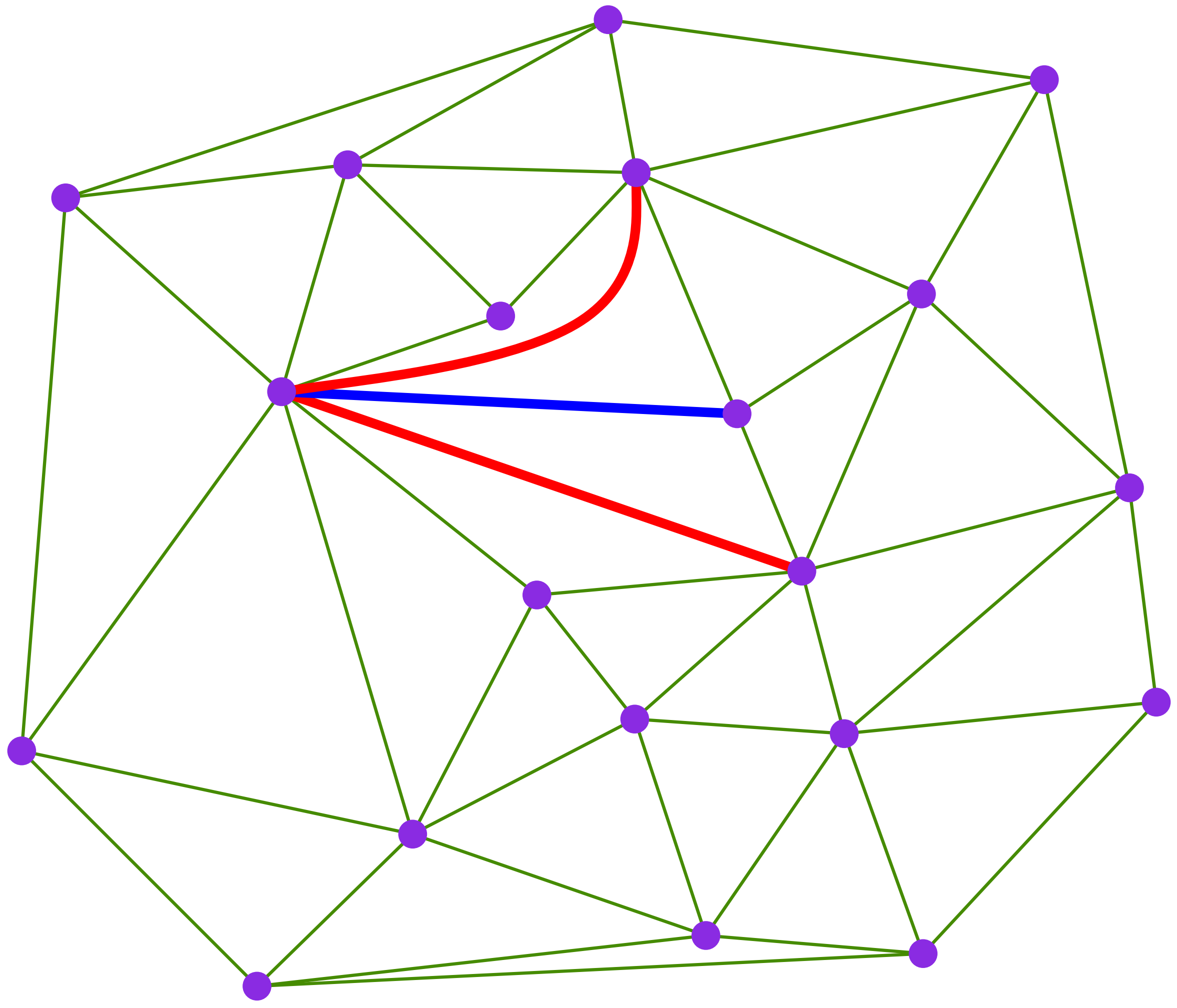
triangulate from any vertex

flip the hole

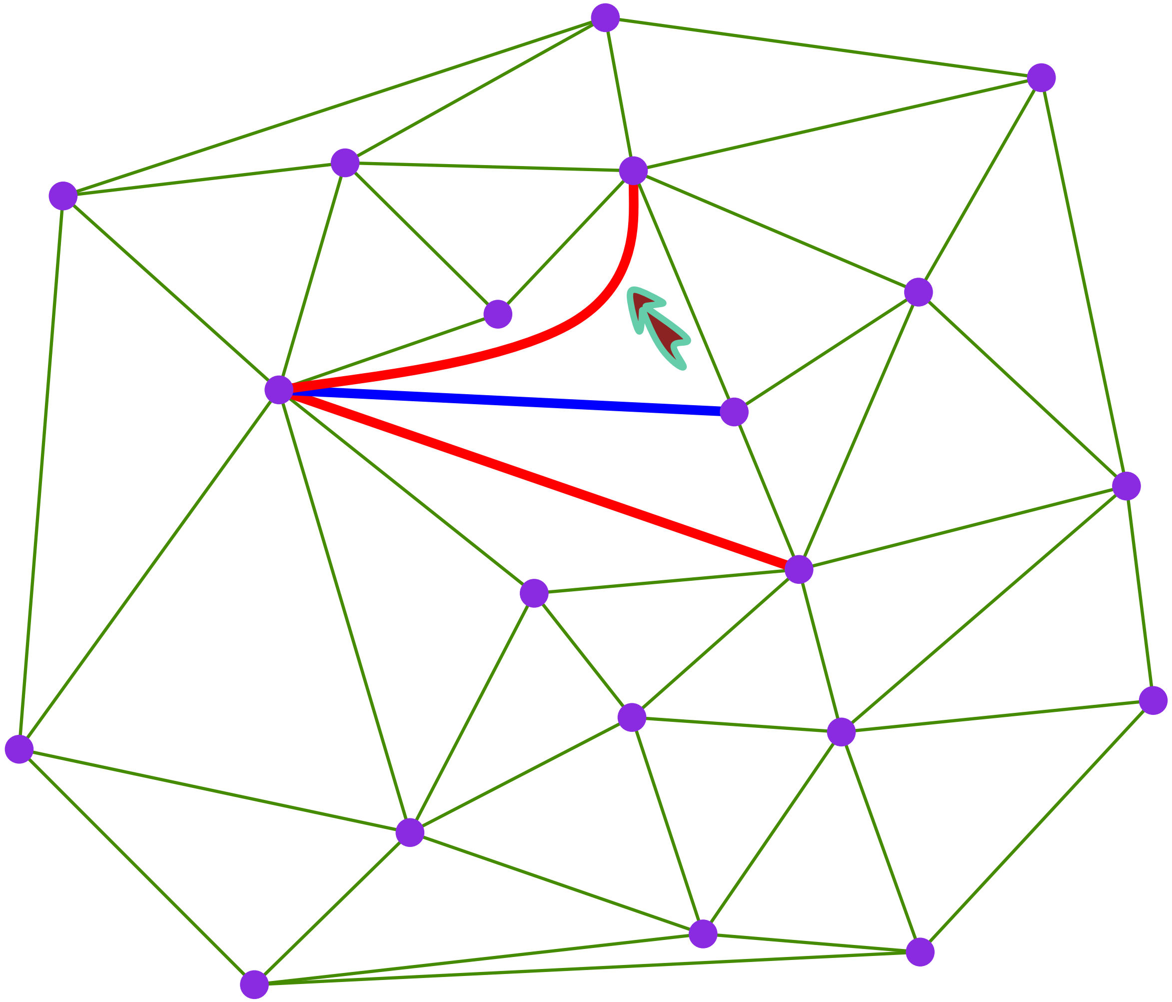


queue of edges to be checked

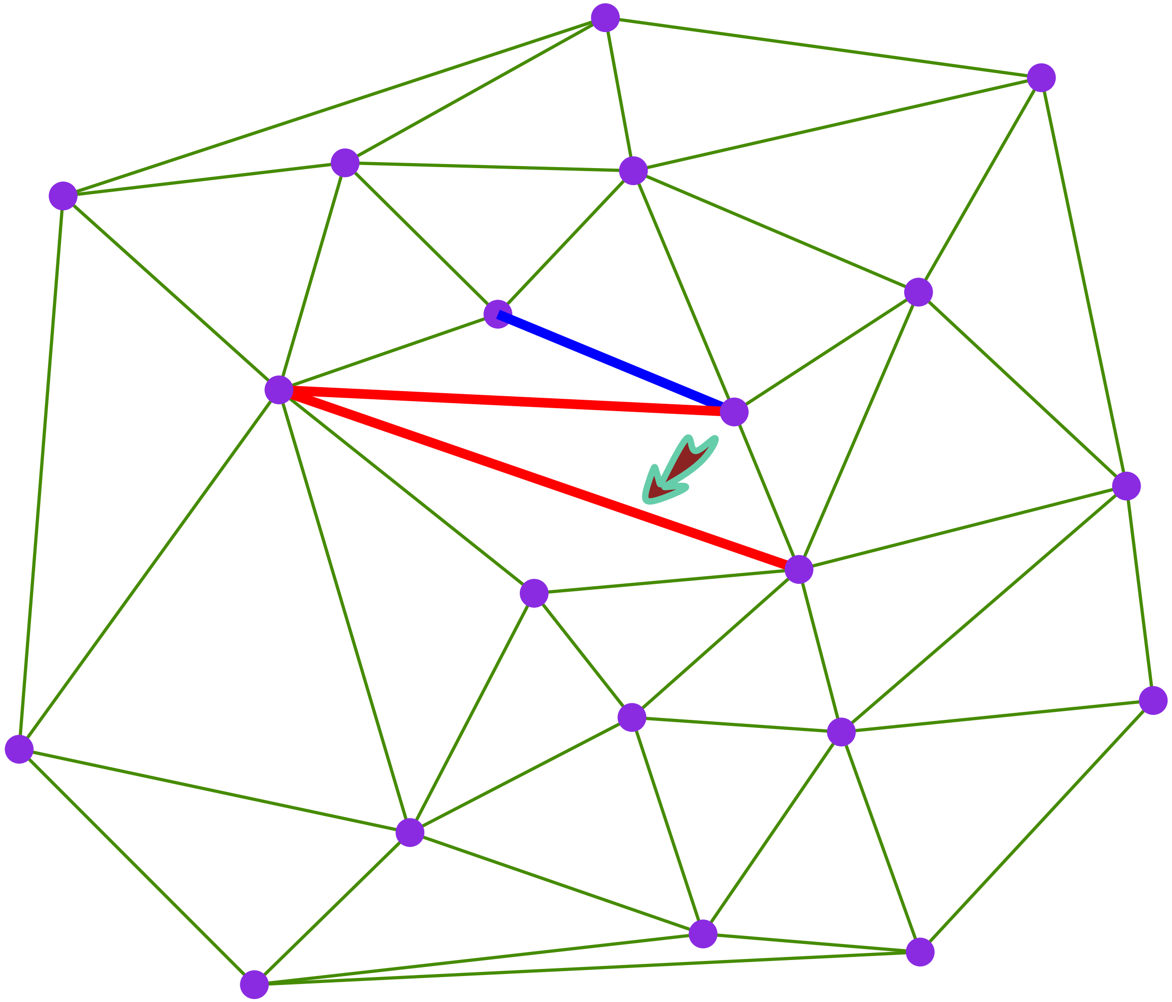
flip the hole



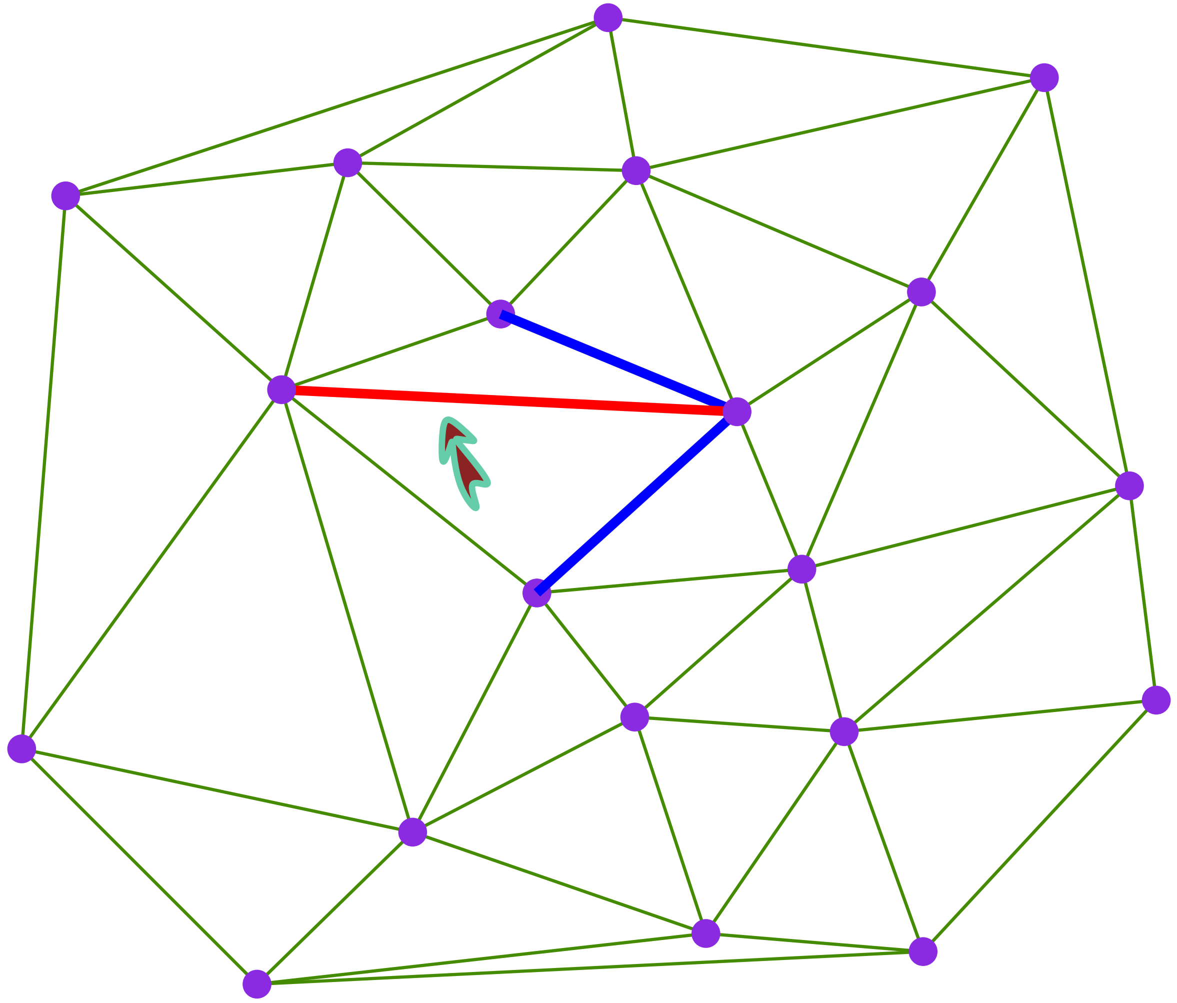
flip the hole



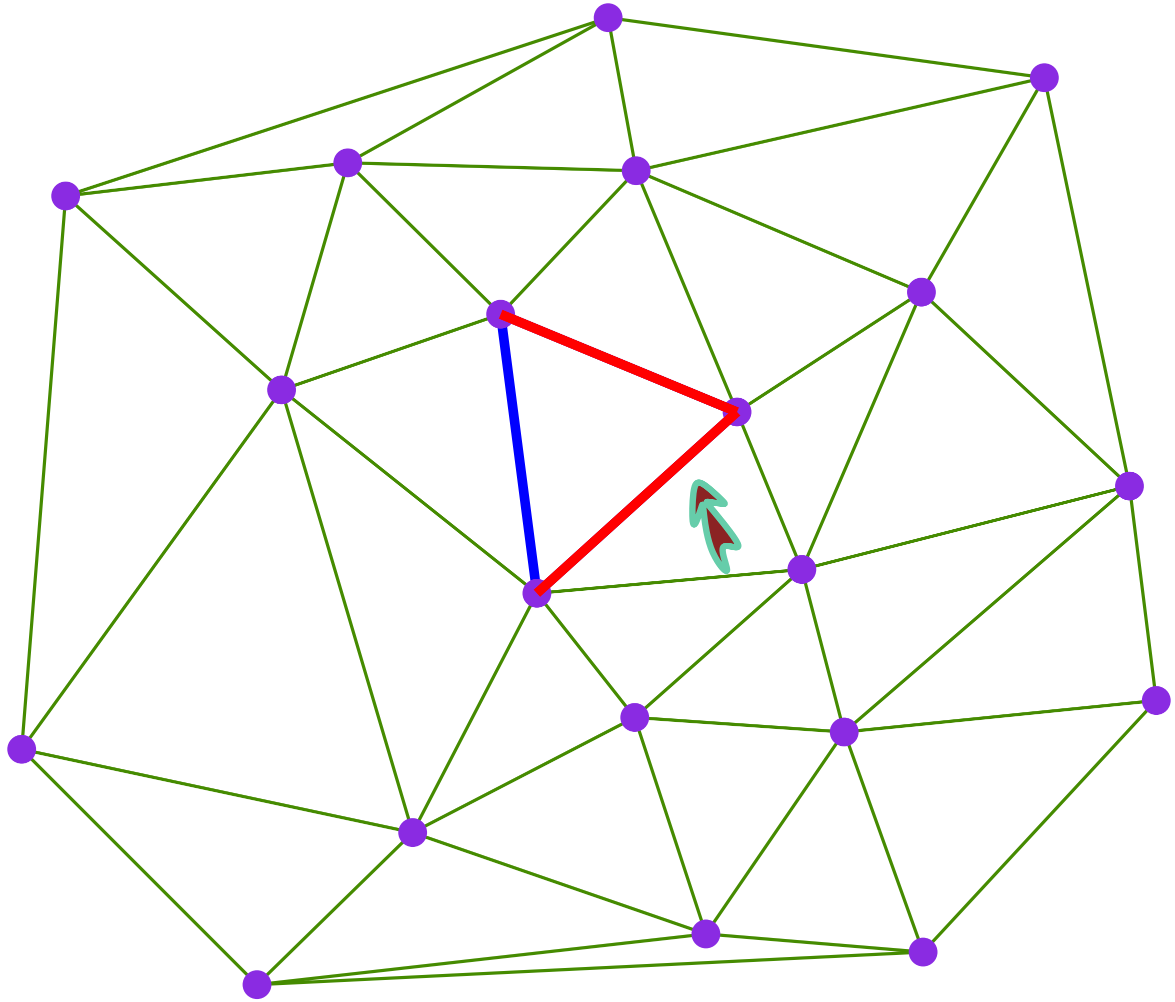
flip the hole



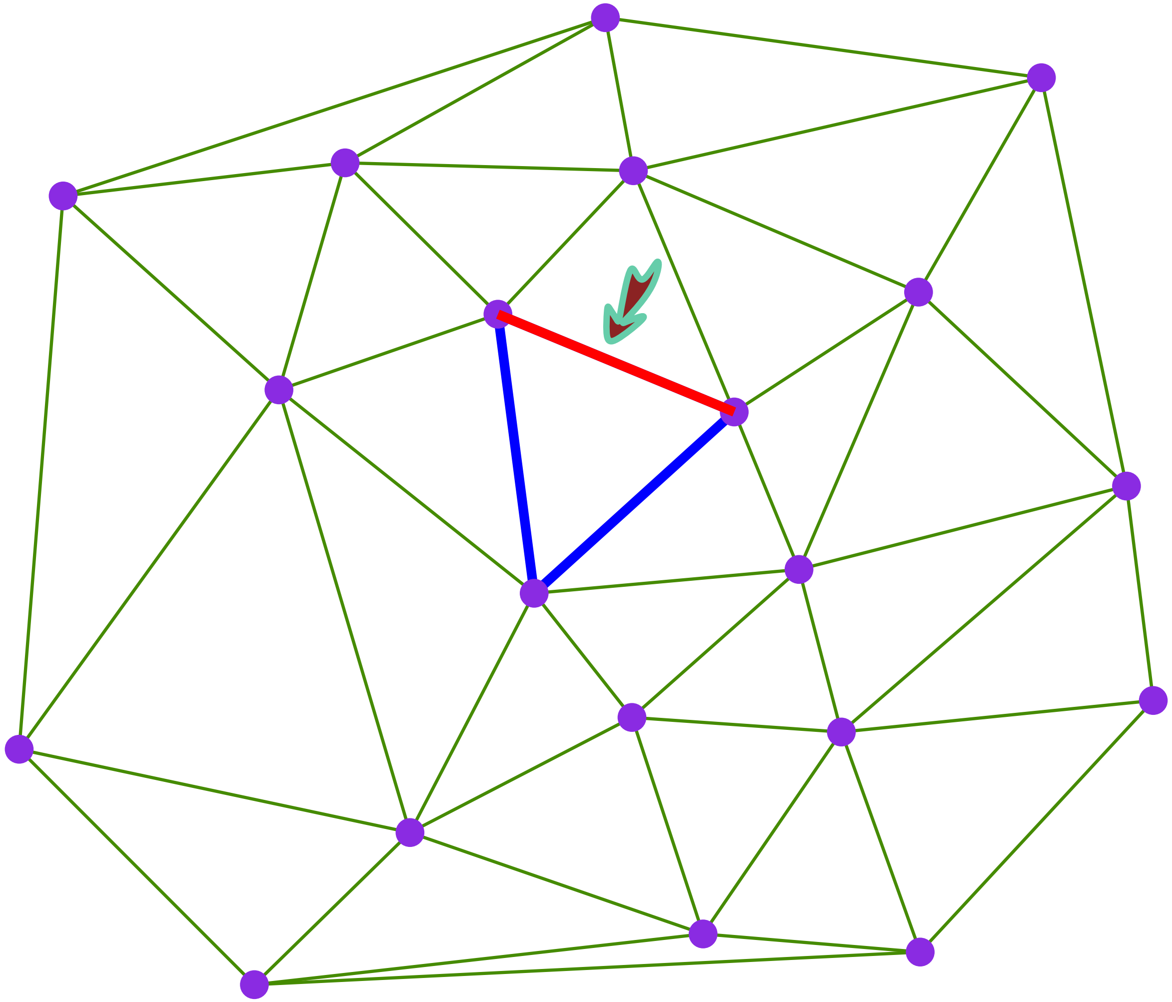
flip the hole



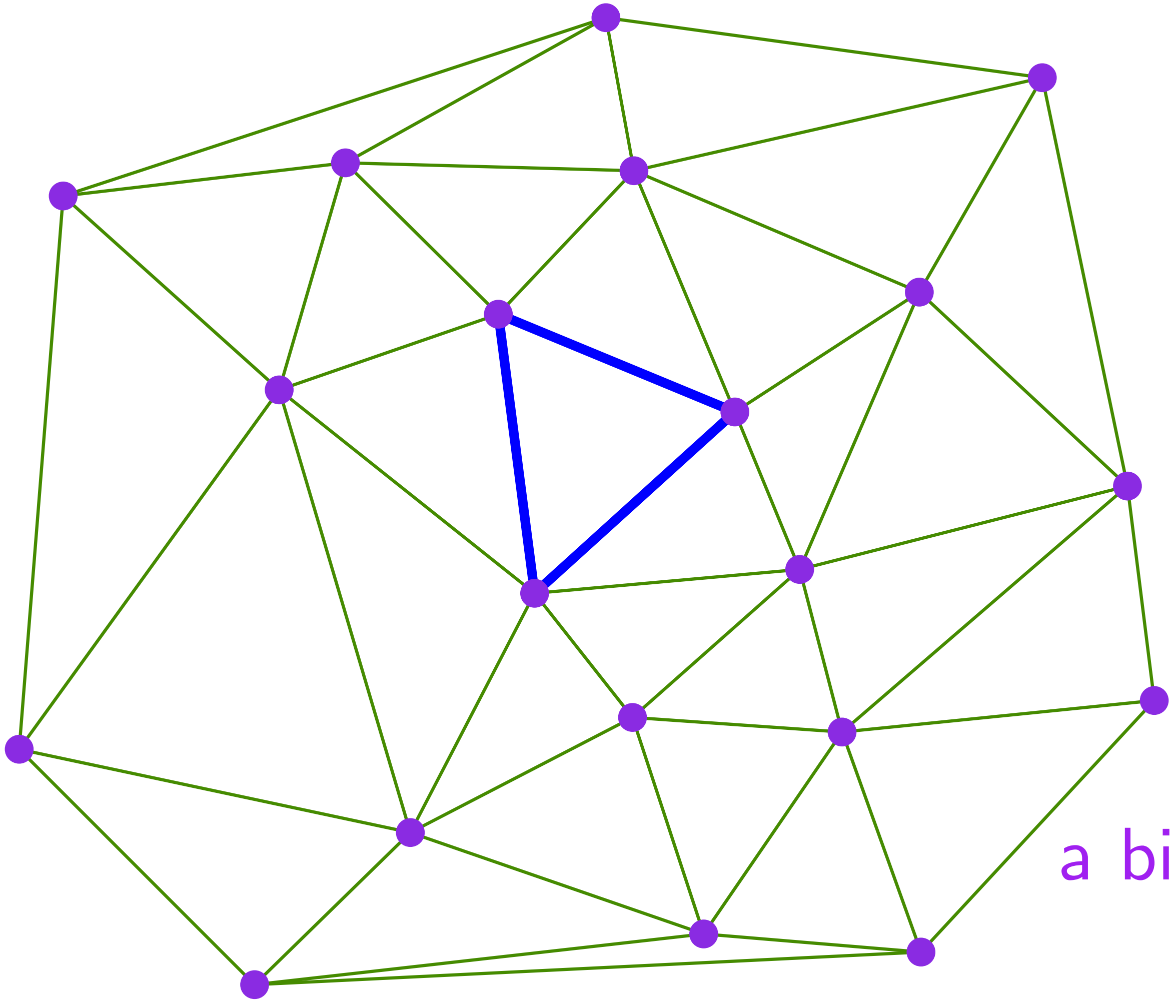
flip the hole



flip the hole

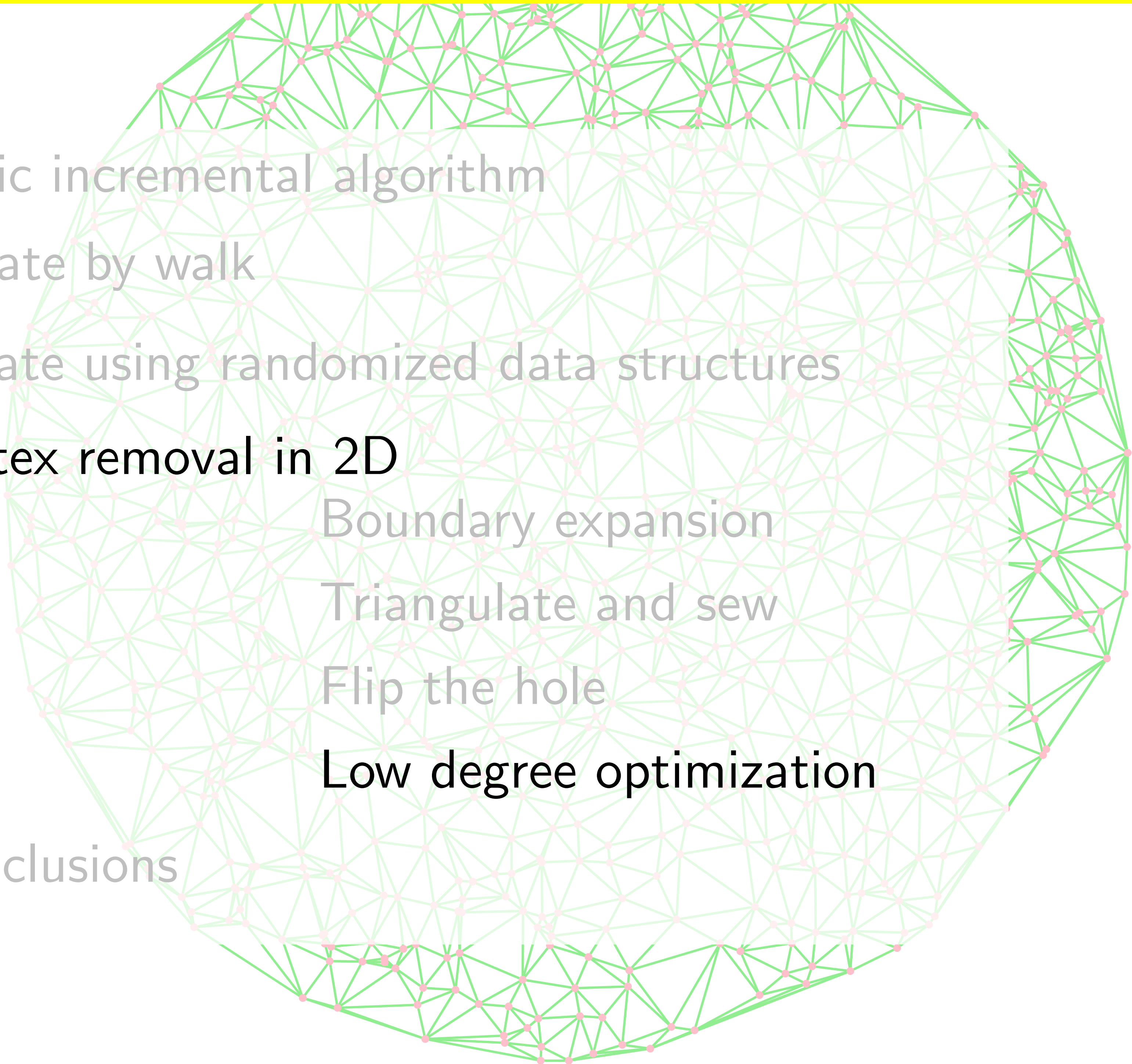


flip the hole



a bit faster

Algorithms



Basic incremental algorithm

Locate by walk

Locate using randomized data structures

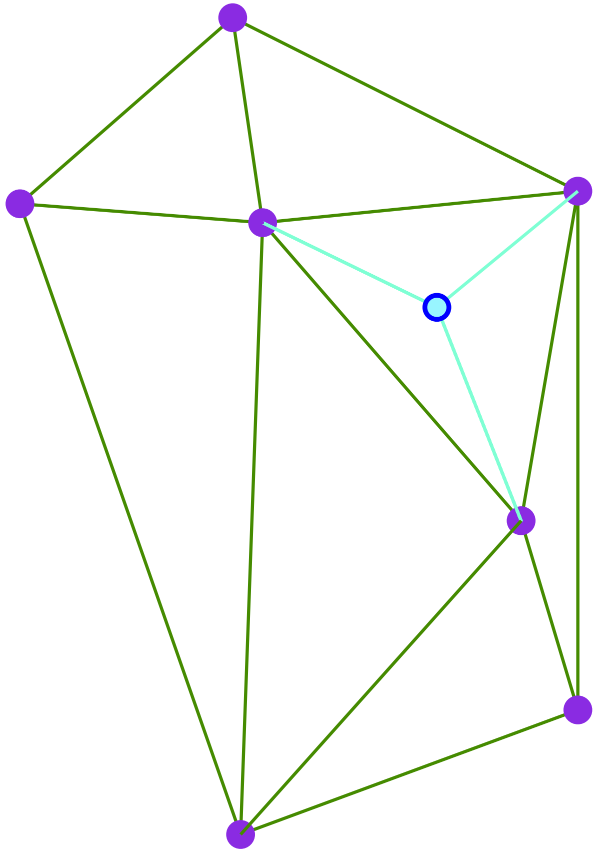
Vertex removal in 2D

- Boundary expansion
- Triangulate and sew
- Flip the hole
- Low degree optimization

Conclusions

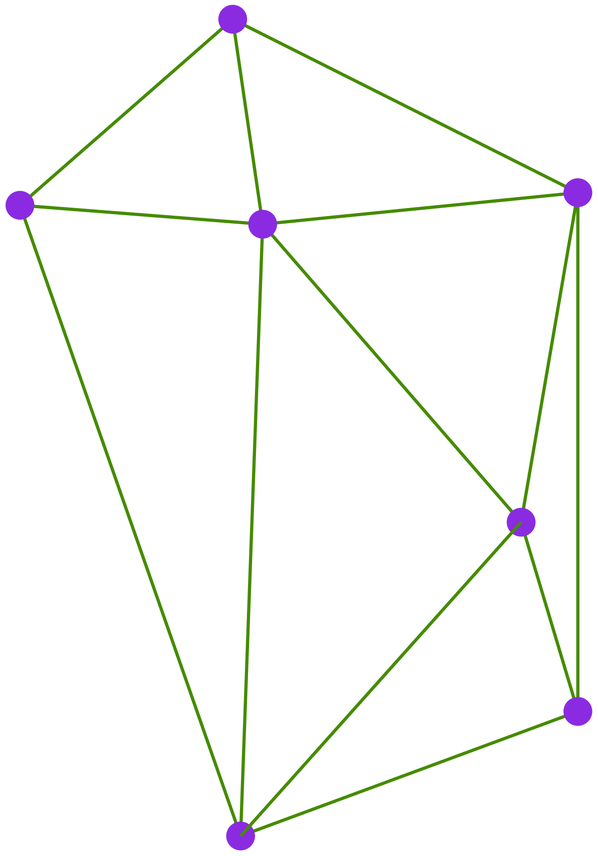
low degree optimization

degree 3



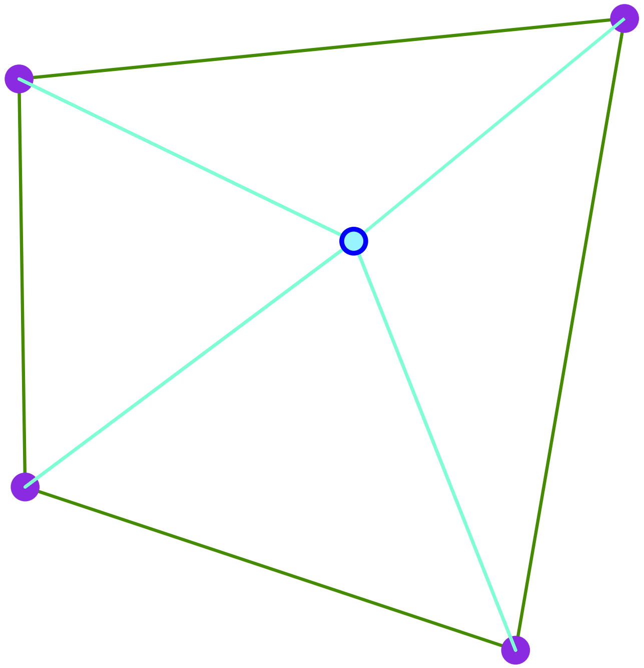
low degree optimization

degree 3



almost nothing to do

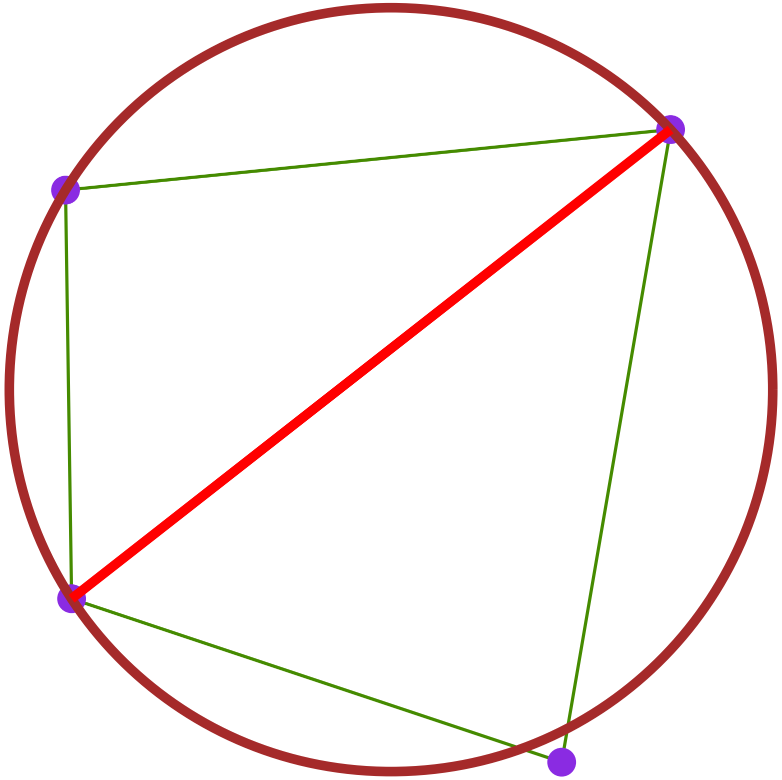
low degree optimization



degree 4

low degree optimization

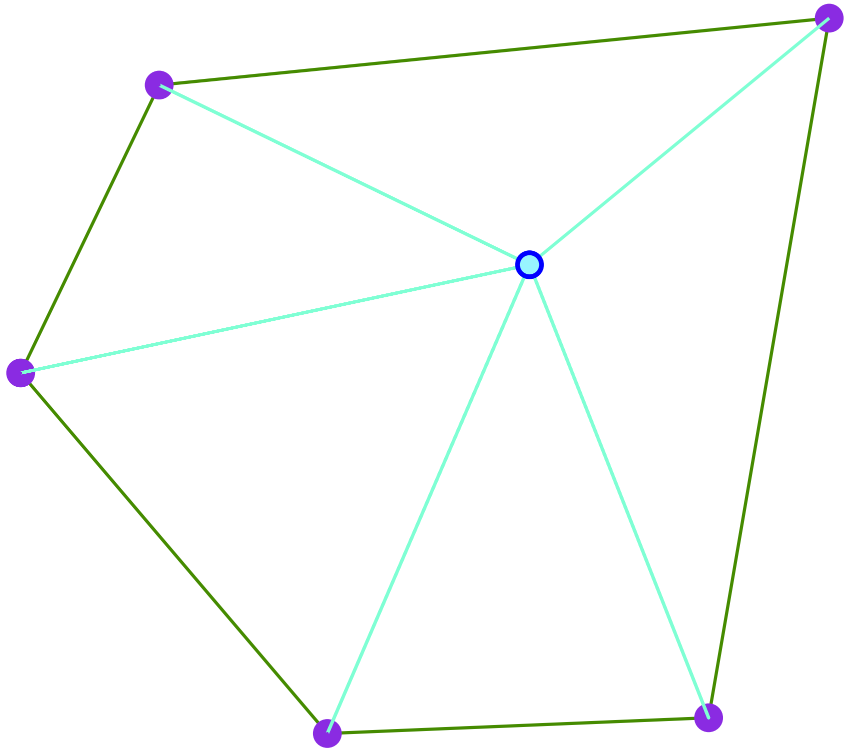
degree 4



just one incircle test to decide

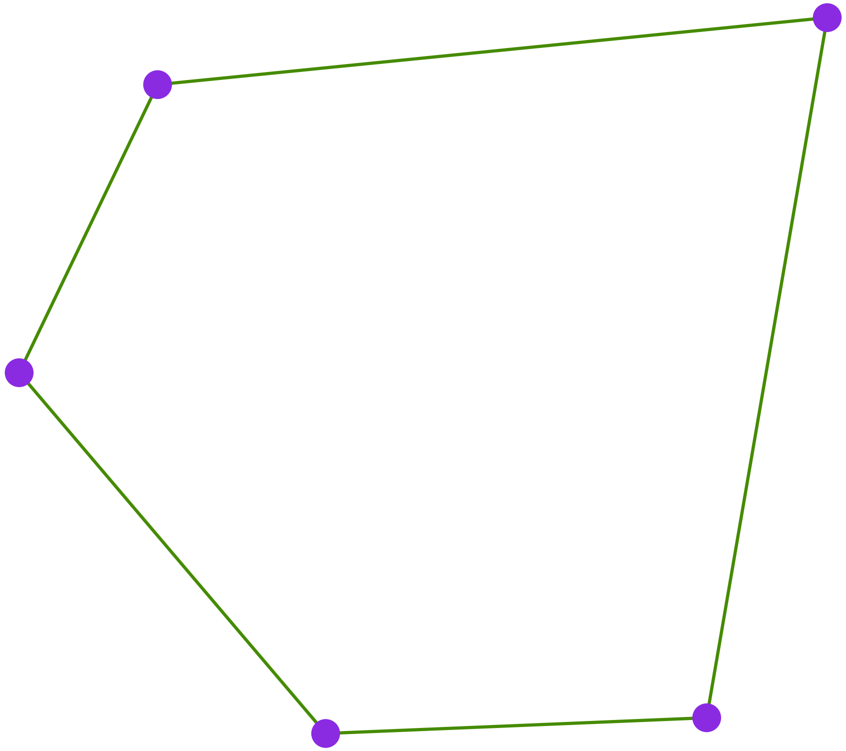
low degree optimization

degree 5



low degree optimization

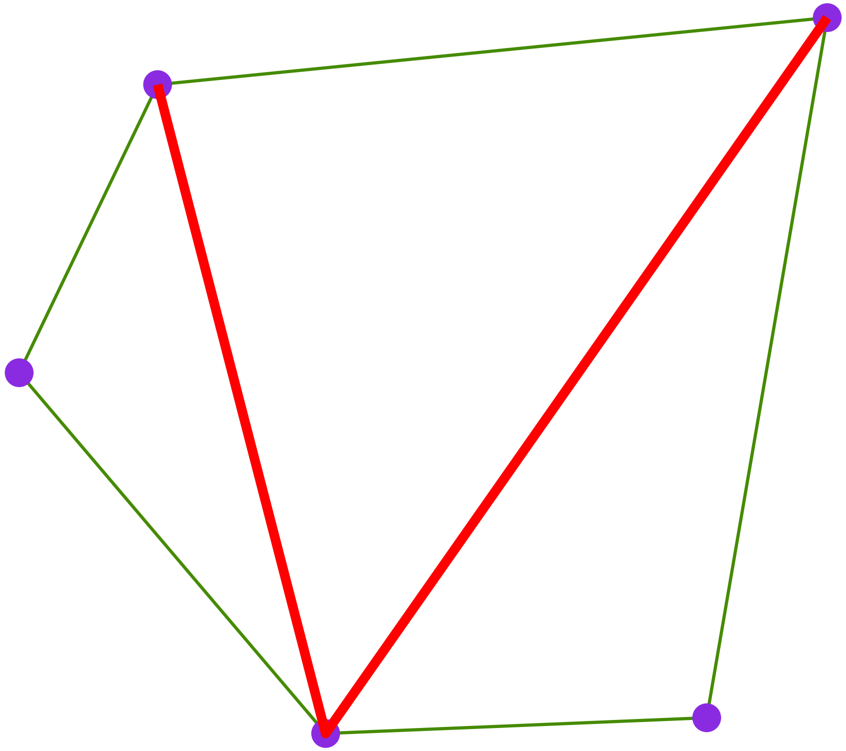
degree 5



”star” the pentagon from the right vertex

low degree optimization

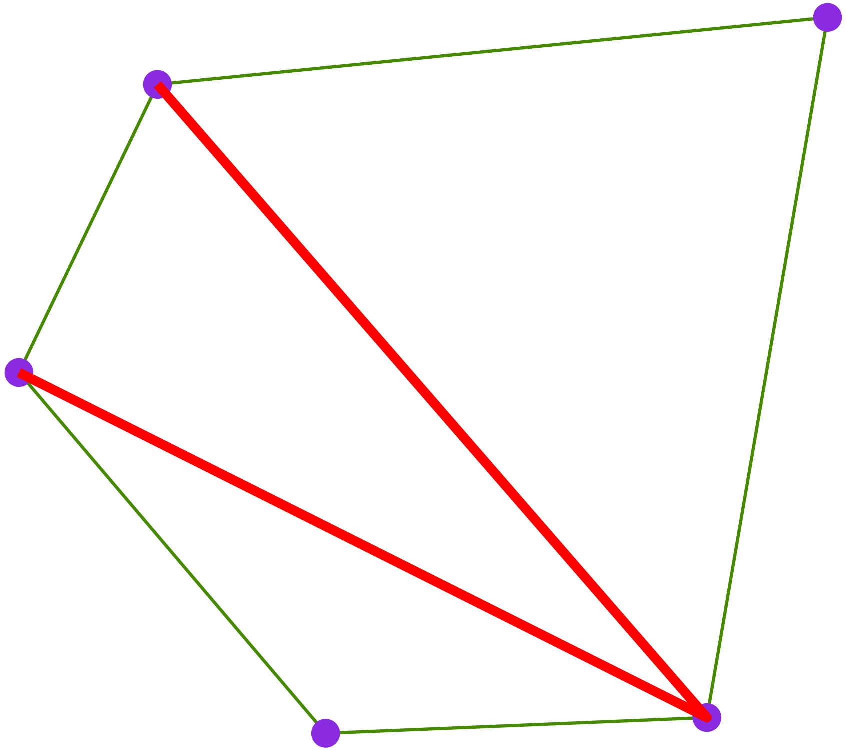
degree 5



"star" the pentagon from the right vertex

low degree optimization

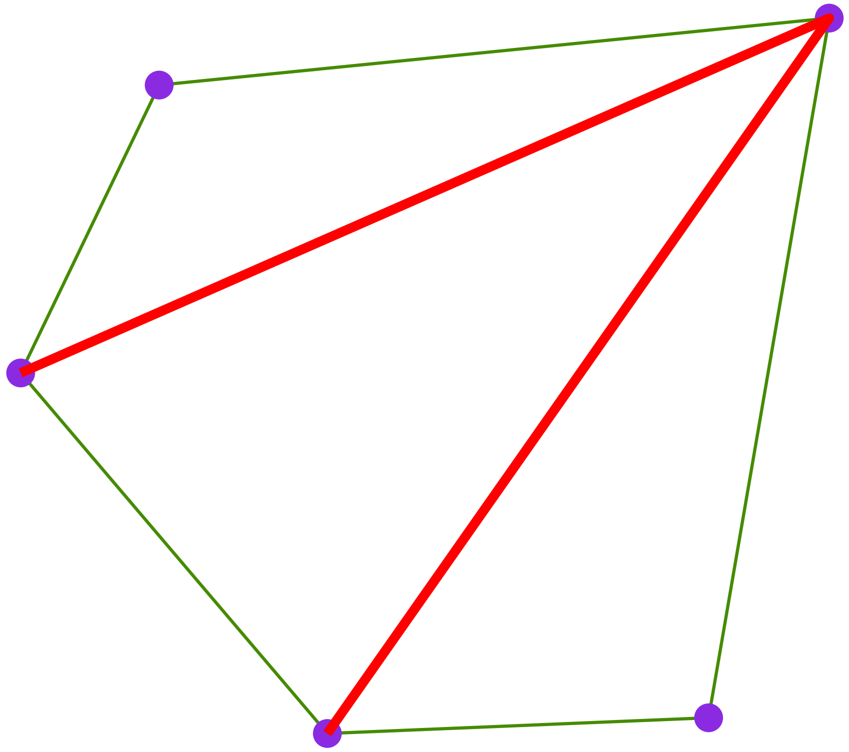
degree 5



"star" the pentagon from the right vertex

low degree optimization

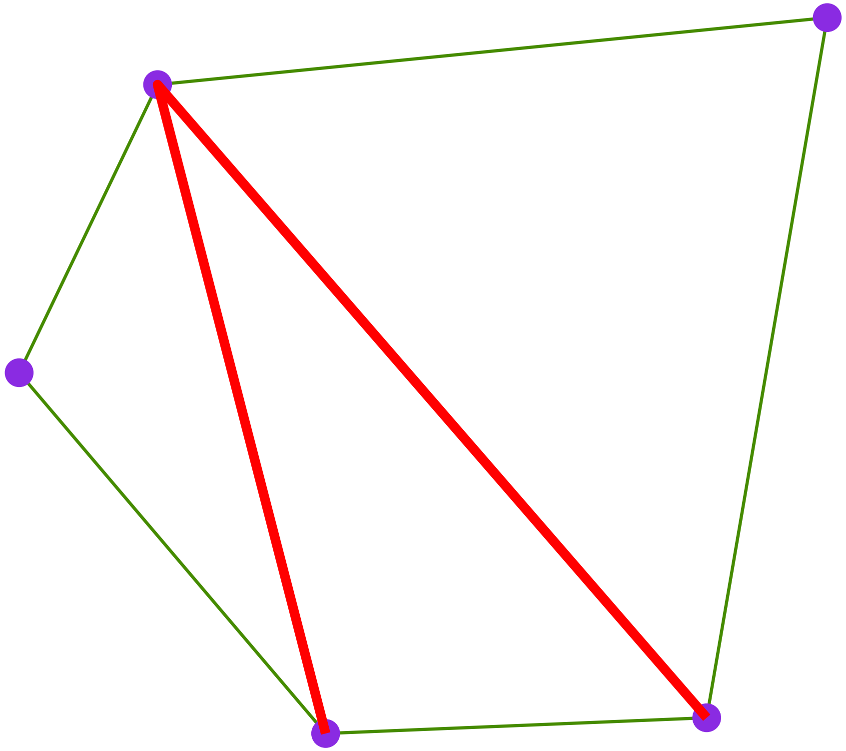
degree 5



”star” the pentagon from the right vertex

low degree optimization

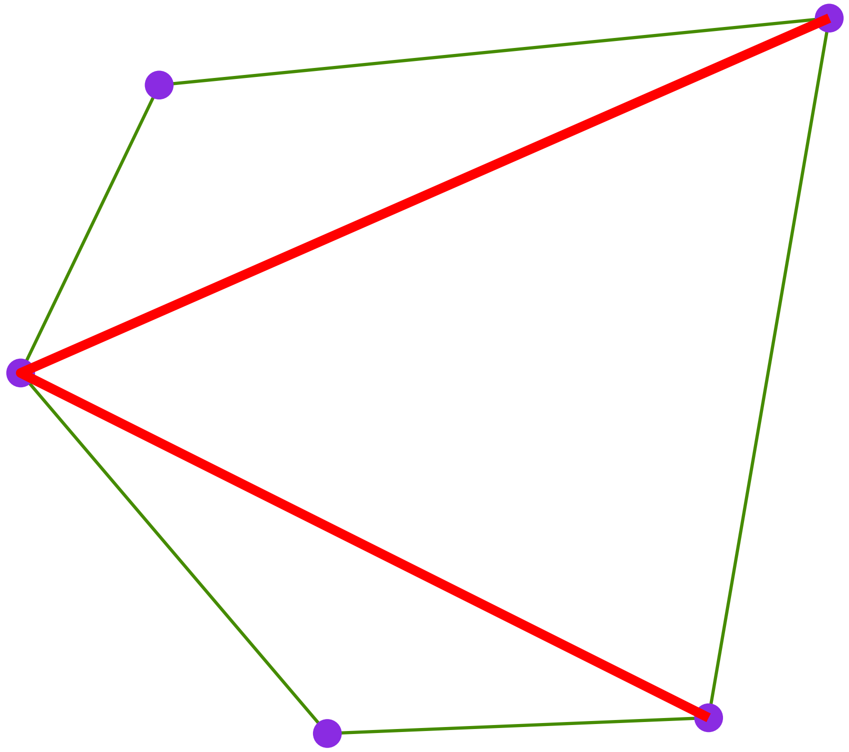
degree 5



"star" the pentagon from the right vertex

low degree optimization

degree 5



"star" the pentagon from the right vertex

low degree optimization

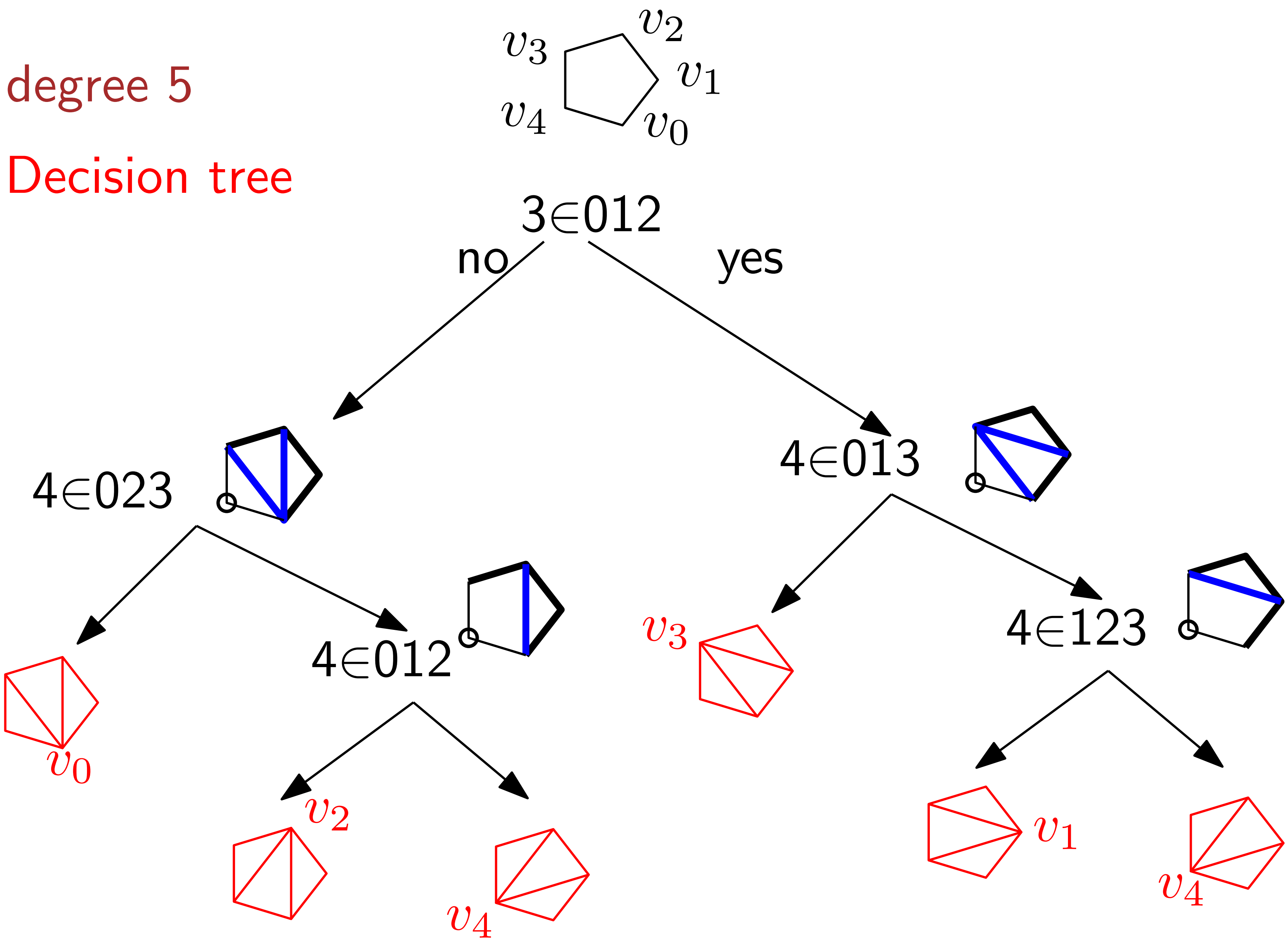
degree 5

Decision tree

low degree optimization

degree 5

Decision tree



v_0 v_1 v_2 v_3 v_4

3∈012

no

yes

4∈023

4∈013

4∈012

4∈123

v_0

v_2

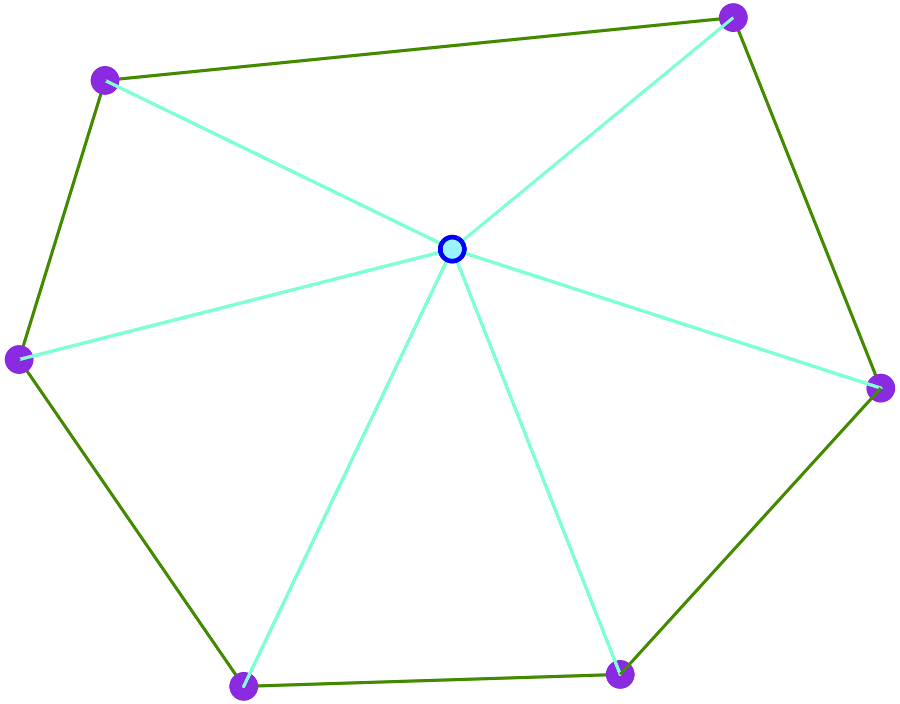
v_4

v_3

v_1

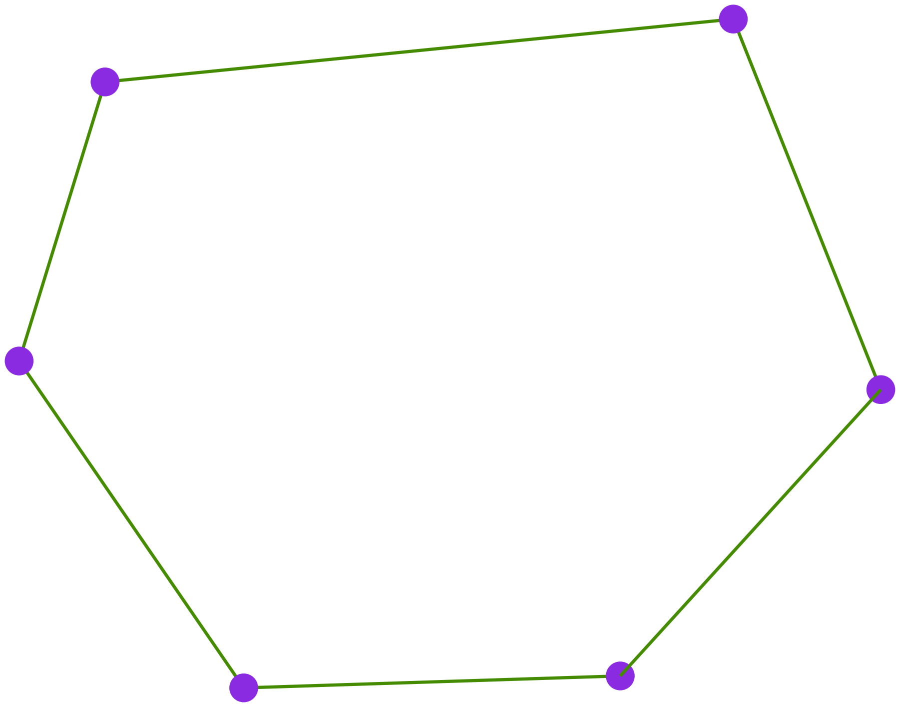
v_4

low degree optimization



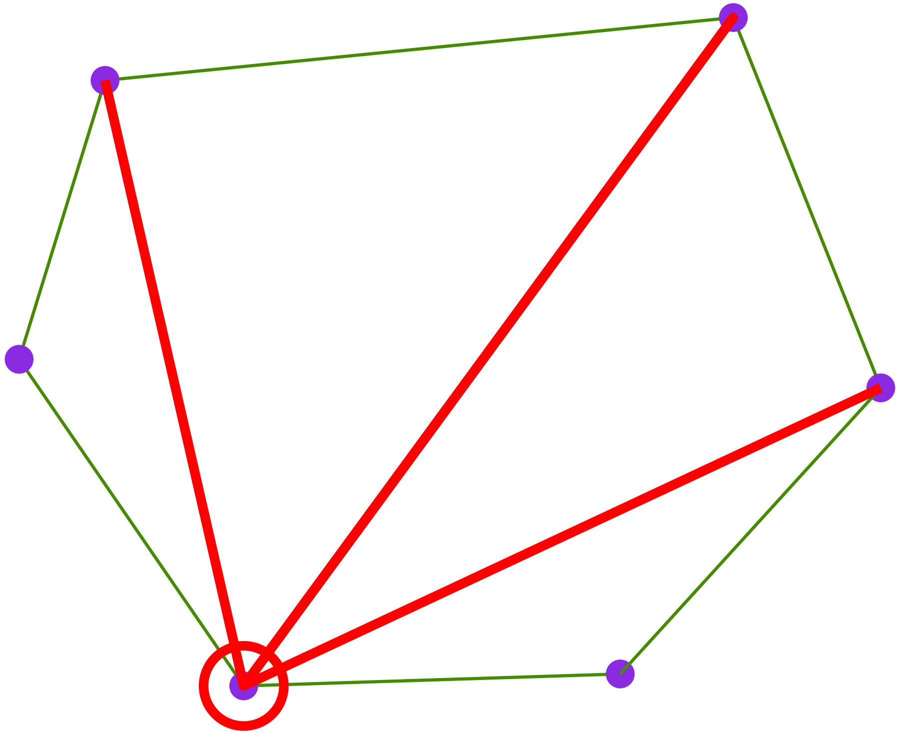
degree 6

low degree optimization



degree 6

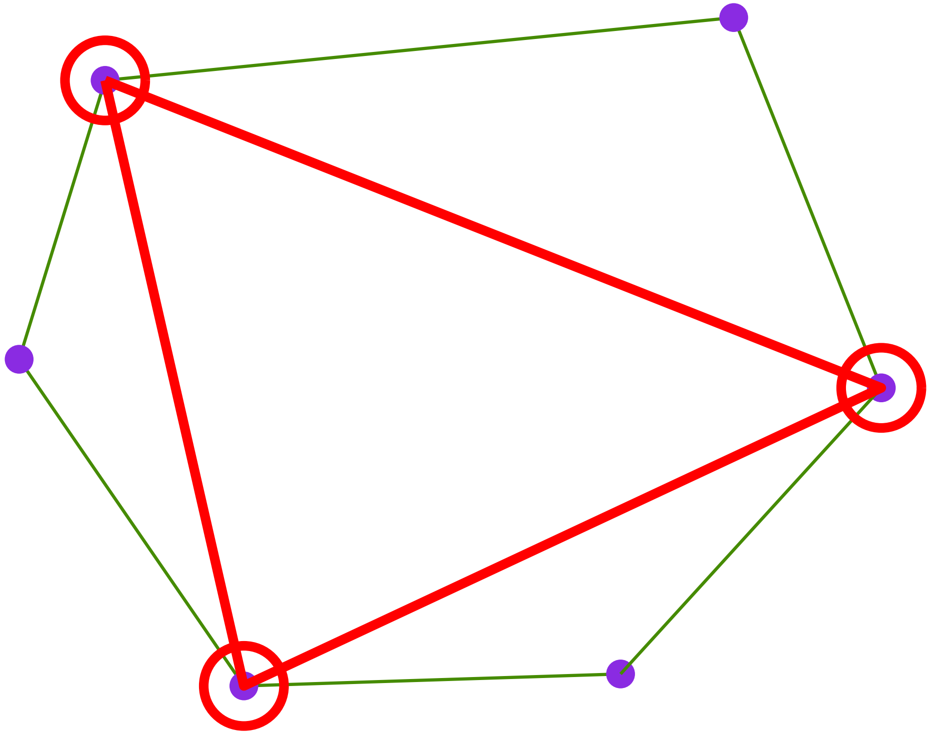
low degree optimization



degree 6

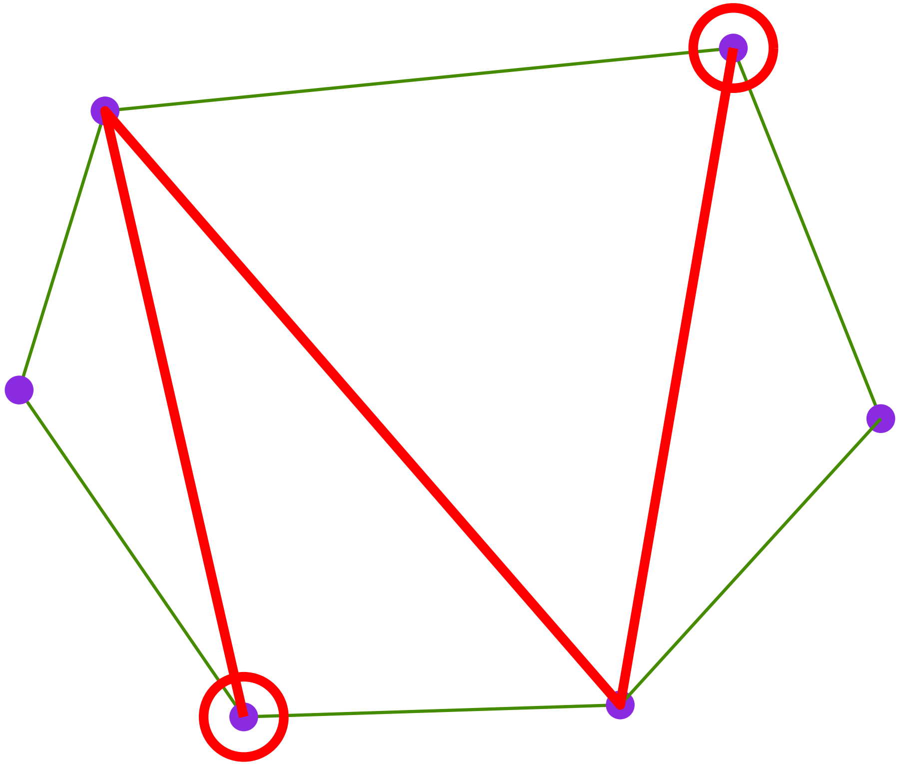
low degree optimization

degree 6

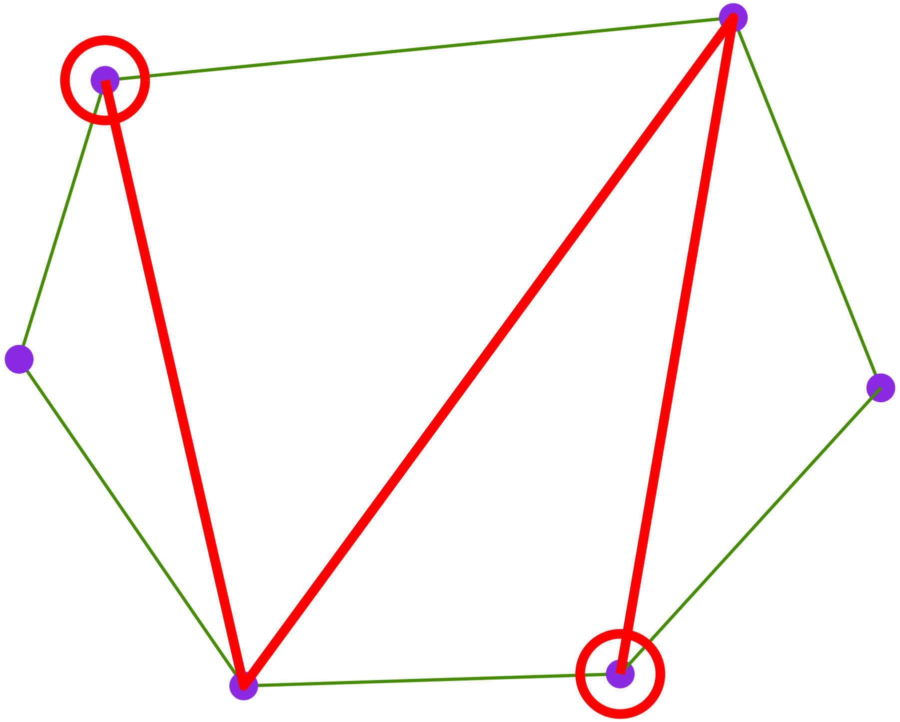


low degree optimization

degree 6



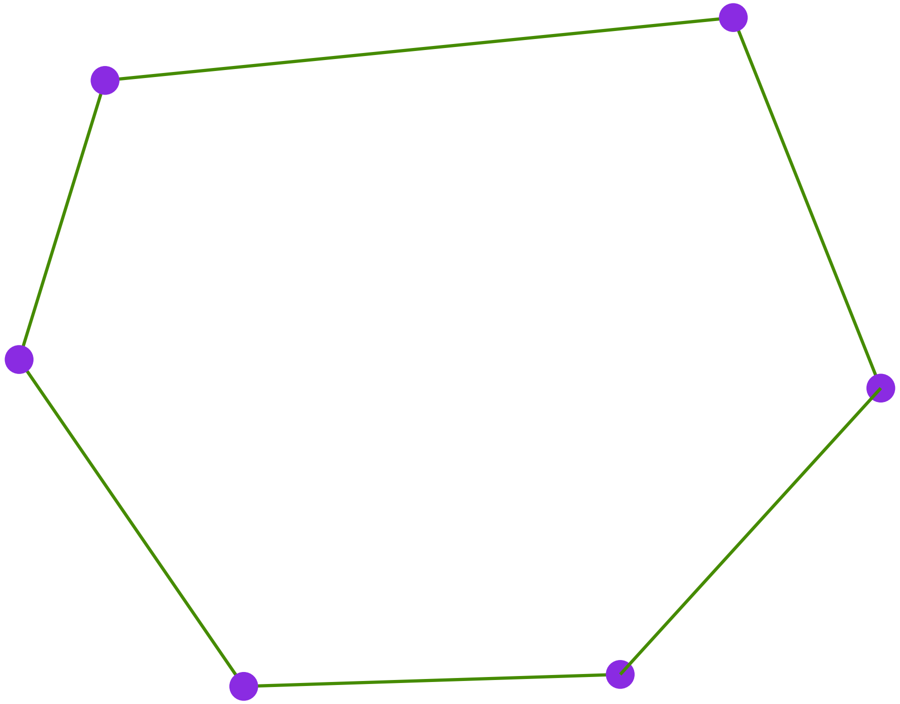
low degree optimization



degree 6

low degree optimization

degree 6



14 results

low degree optimization

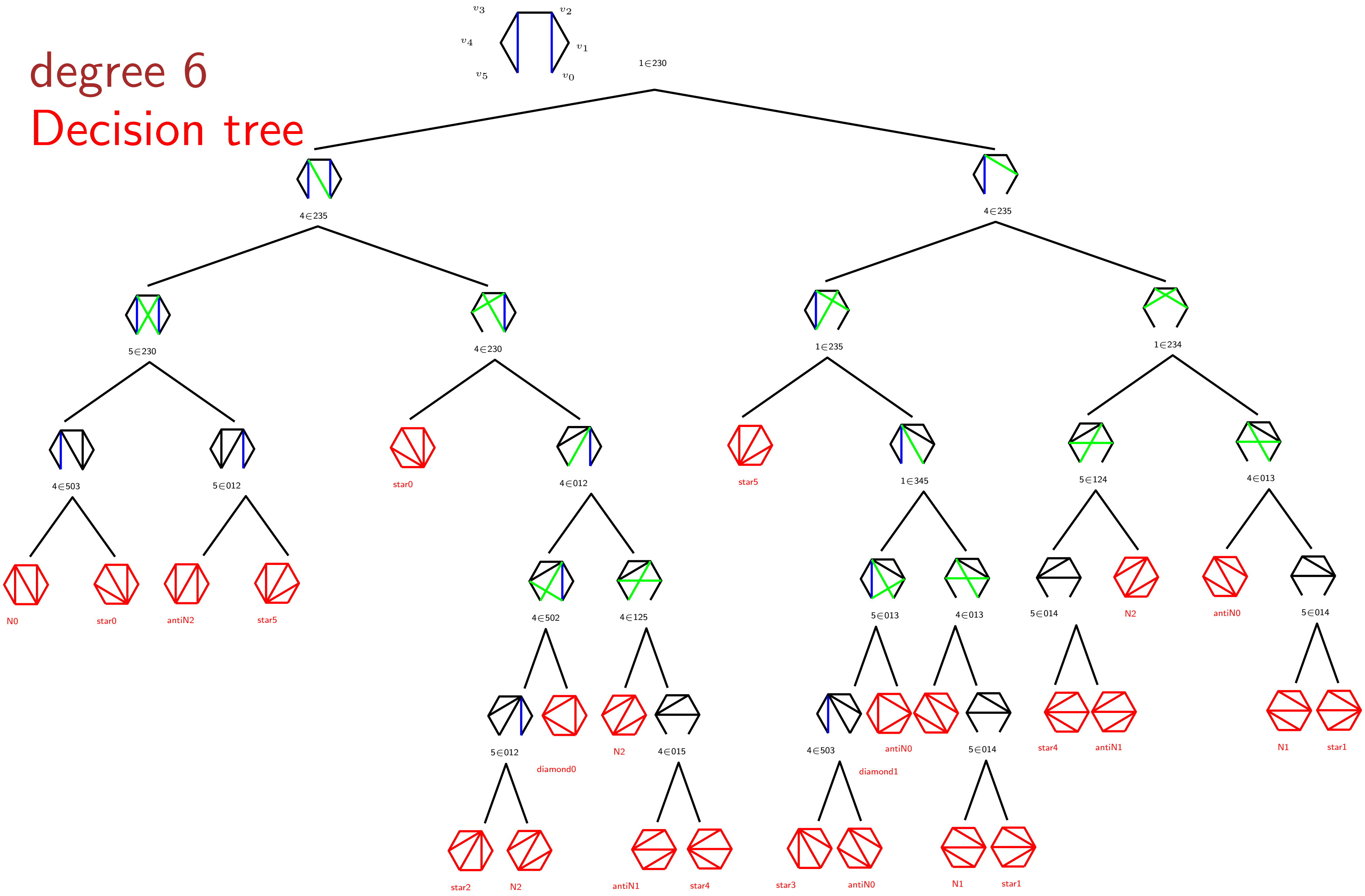
degree 6

Decision tree

low degree optimization

degree 6

Decision tree

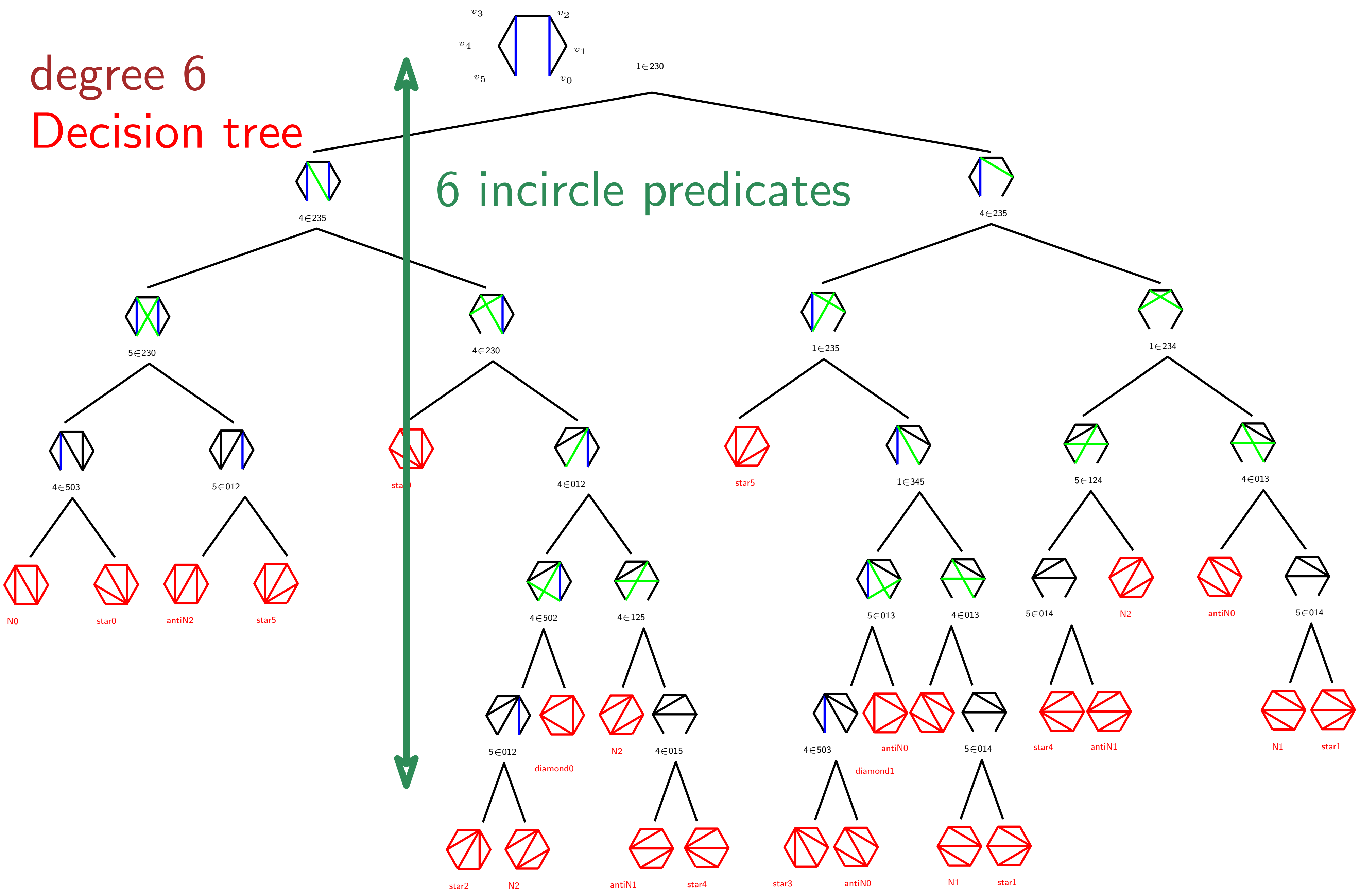


low degree optimization

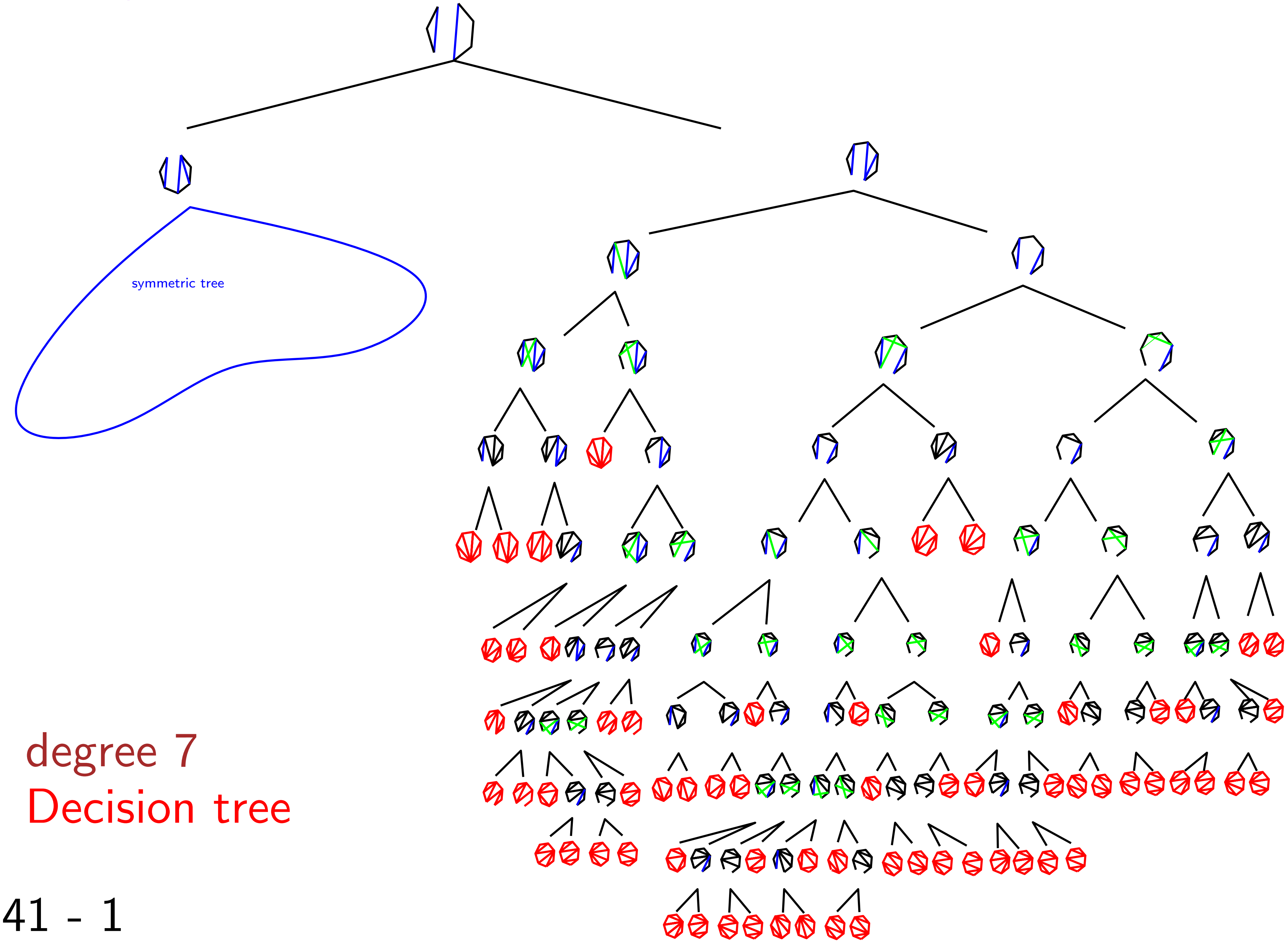
degree 6

Decision tree

6 incircle predicates

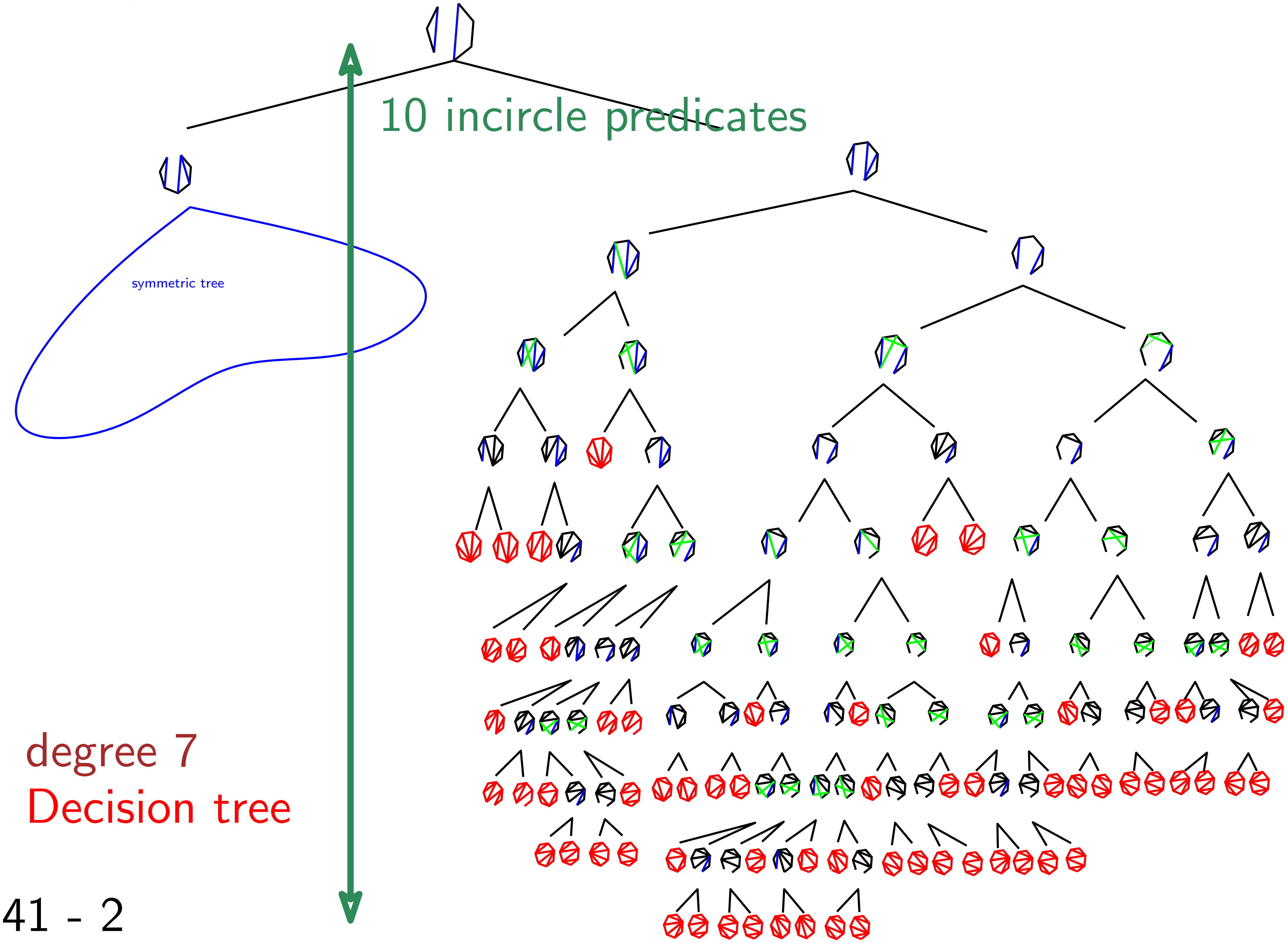


low degree optimization



degree 7
Decision tree

low degree optimization



degree 7
Decision tree

low degree optimization

degree	3	4	5	6	7	$8^\star$	9
♯ results	1	2	5	14	42	132	429
♯ leaves	1	2	6	24	130	$\simeq$*500*	
$\lceil \log_2 \sharp results \rceil$	0	1	3	4	6	8	9
tree height	0	1	3	6	10	$\simeq$*14*	
♯ lines of code	30	40	90	280	700	$\simeq$*2500*	

* not implemented. The sizes of the tree and the code are estimated

low degree optimization

Remarks on implementation

limited memory allocation, use old faces ”in place”

re-use as many neighbor links as possible

low degree optimization

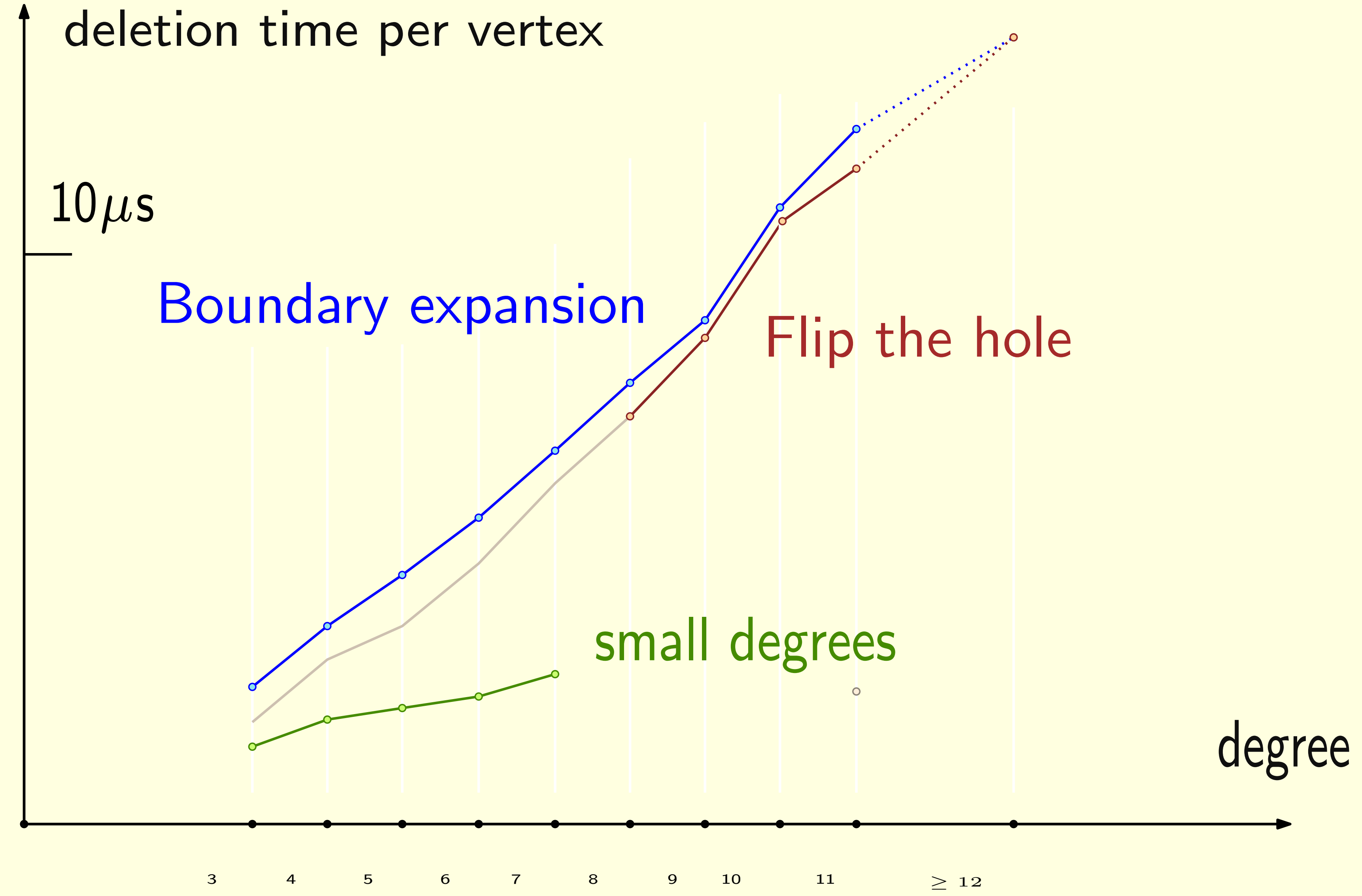
Remarks on implementation

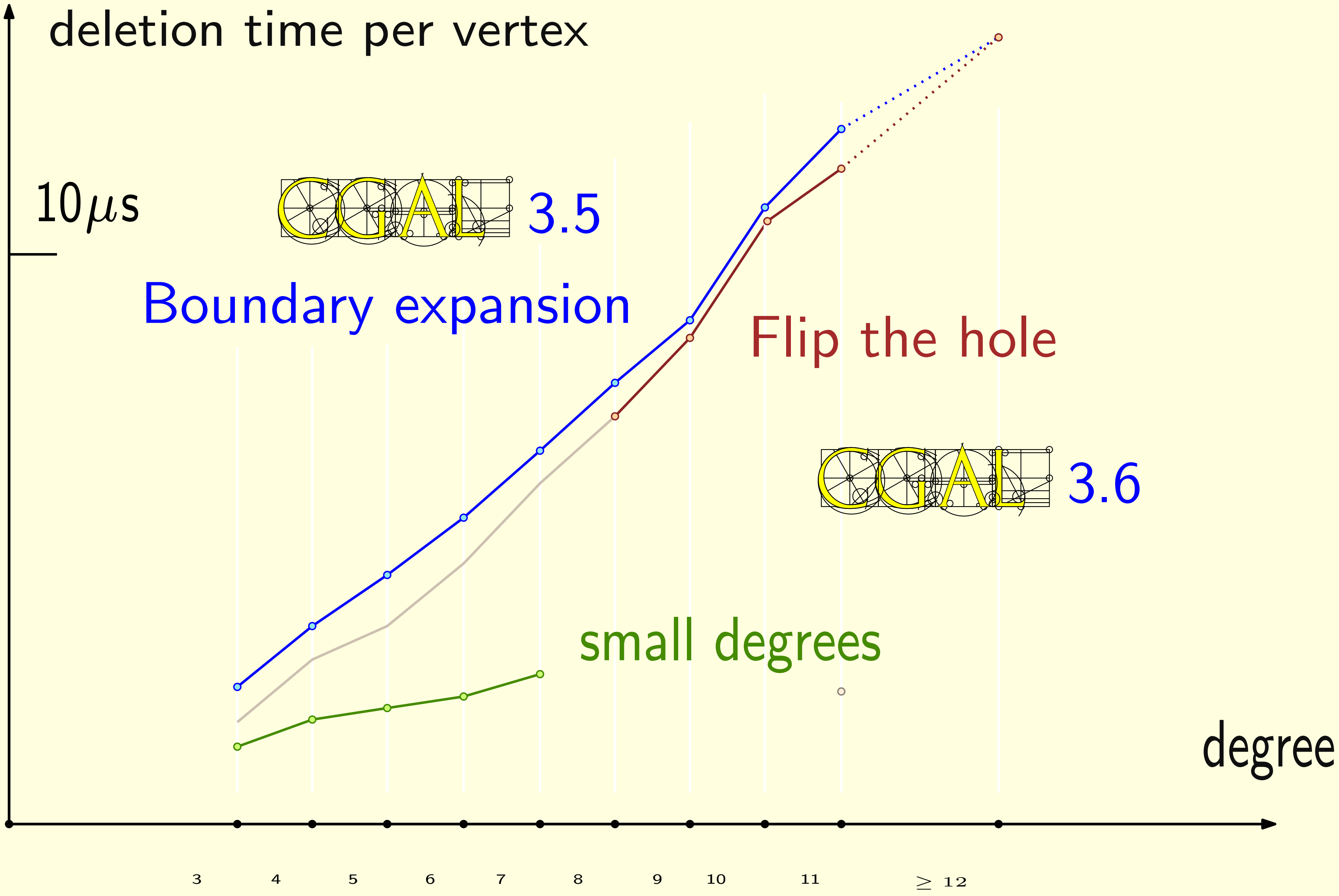
limited memory allocation, use old faces ”in place”

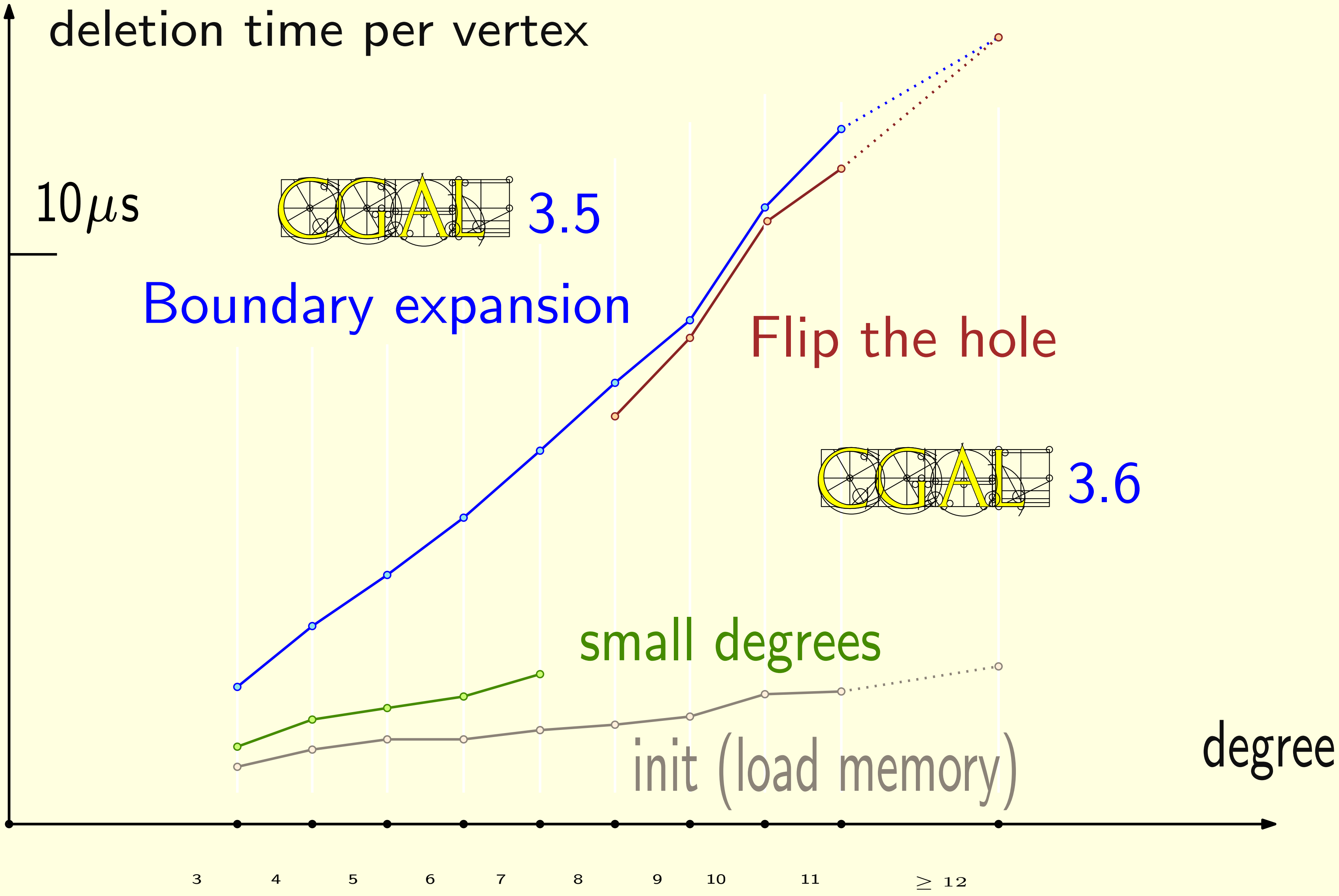
re-use as many neighbor links as possible

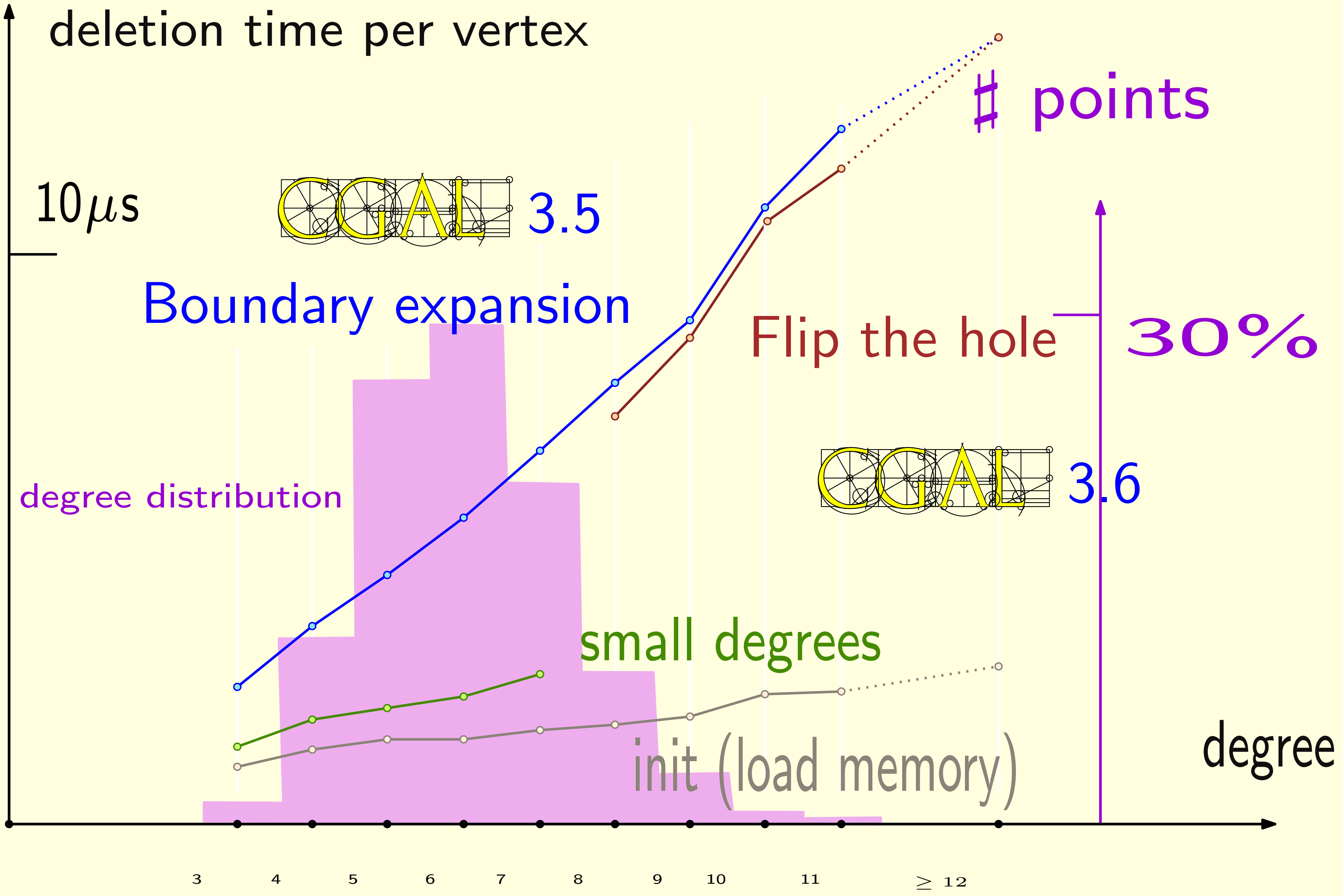
tree implementation

```
if incircle(...)
    if incircle(...)
        if incircle(...)  use_this_shape(face0,face1,face2...)
        else               use_other_shape(face2,face3,face4...)
......
```

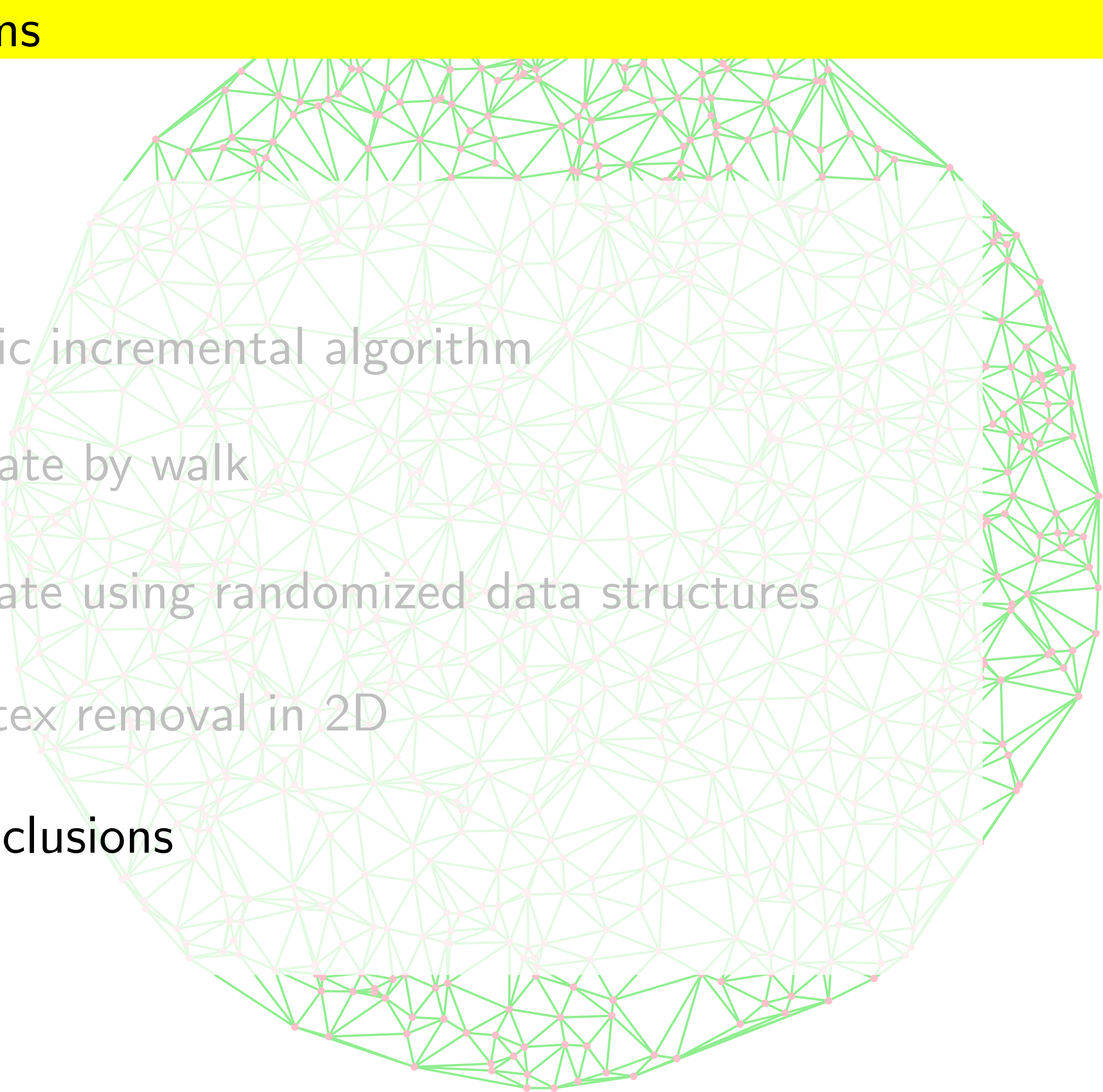

deletion time per vertex
10μs
Boundary expansion
Flip the hole
small degrees
degree
3
4
5
6
7
8
9
10
11
$\geq$ 12


deletion time per vertex
10μs
CGAL 3.5
Boundary expansion
Flip the hole
CGAL 3.6
small degrees
degree
3
4
5
6
7
8
9
10
11
$\geq$ 12


deletion time per vertex
10μs
CGAL 3.5
Boundary expansion
Flip the hole
CGAL 3.6
small degrees
init (load memory)
degree
3
4
5
6
7
8
9
10
11
$\geq$ 12


deletion time per vertex
10μs
CGAL 3.5
Boundary expansion
♯ points
Flip the hole
30%
CGAL 3.6
degree distribution
small degrees
init (load memory)
degree
3
4
5
6
7
8
9
10
11
≥ 12

Algorithms



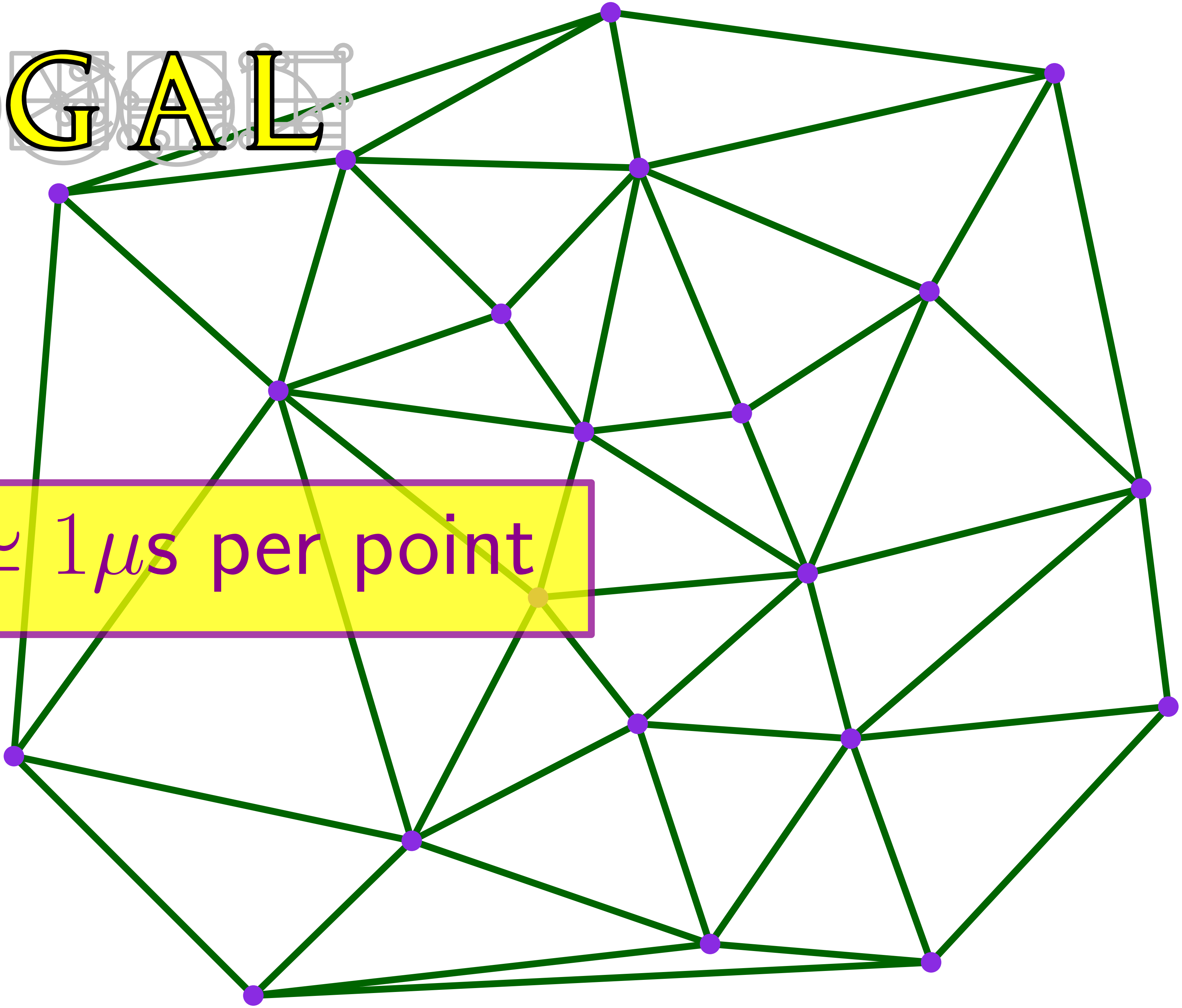
Basic incremental algorithm

Locate by walk

Locate using randomized data structures

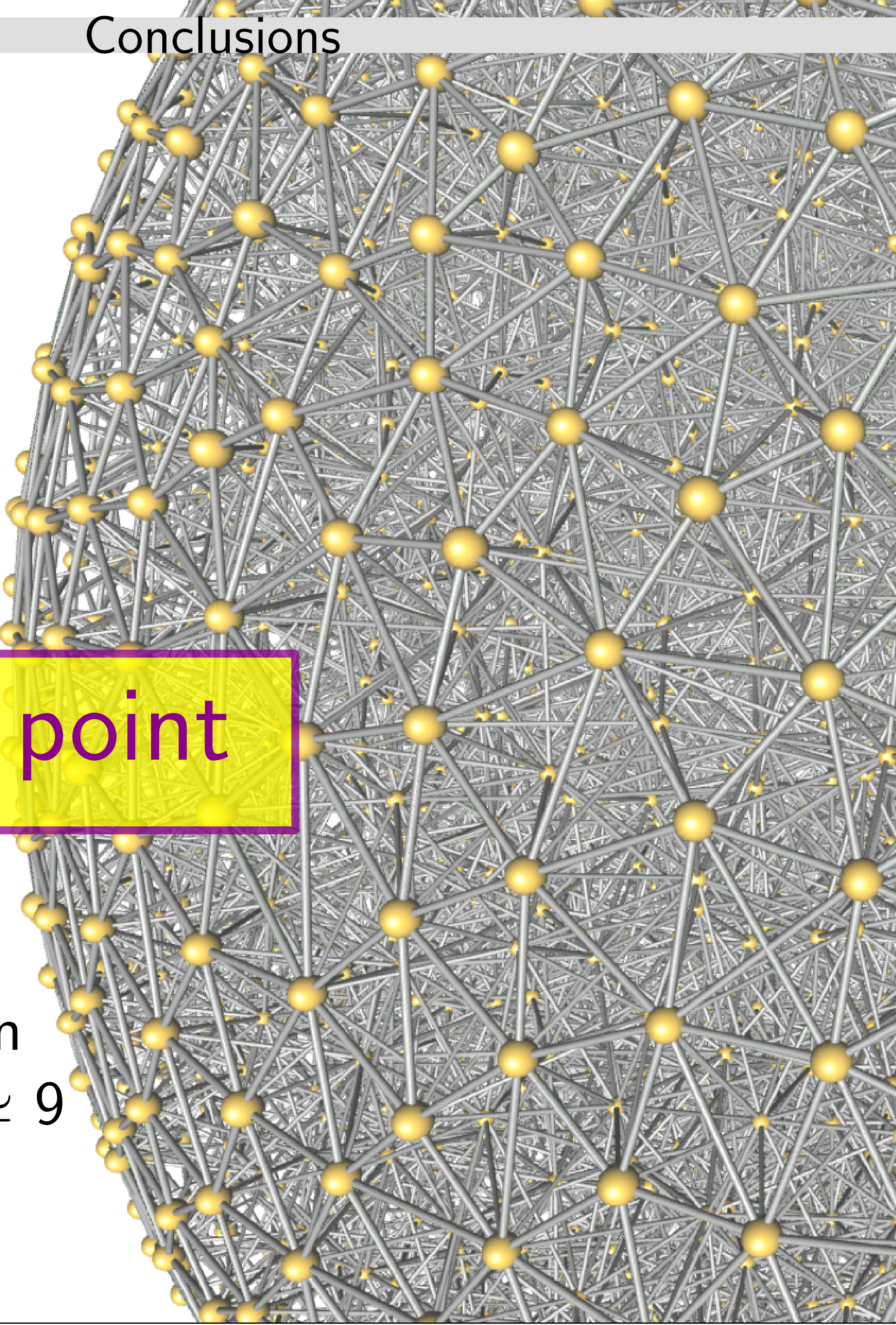
Vertex removal in 2D

Conclusions



$\simeq 1\mu$s per point

CGAL



$\simeq 8\mu$s per point

CGAL $\geq$ 4.5: multicore option
10 cores $\mapsto$ speed up factor $\simeq 9$

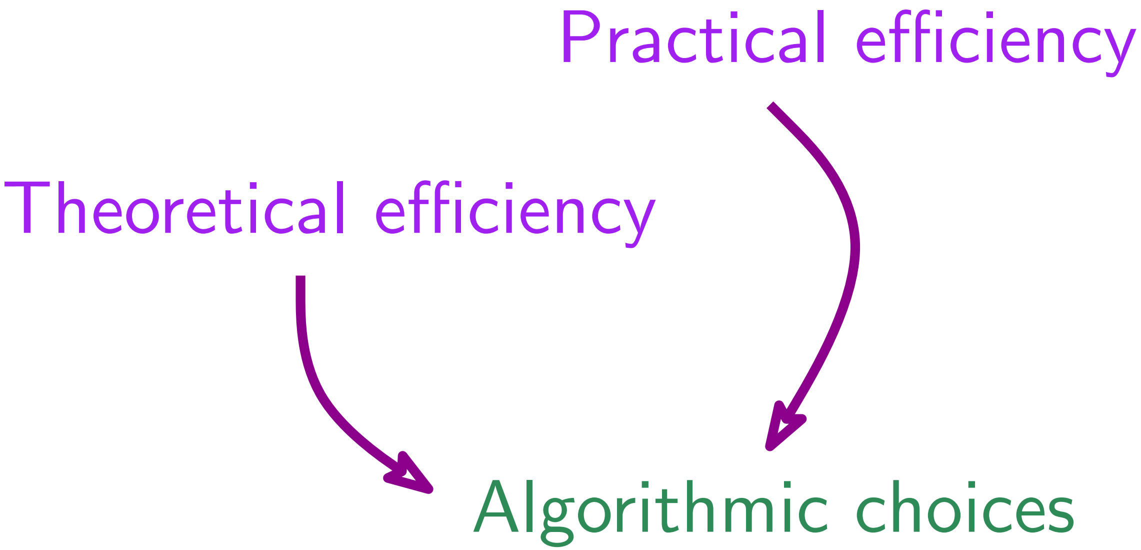
Algorithmic choices

CGAL

Theoretical efficiency

Algorithmic choices

CGAL

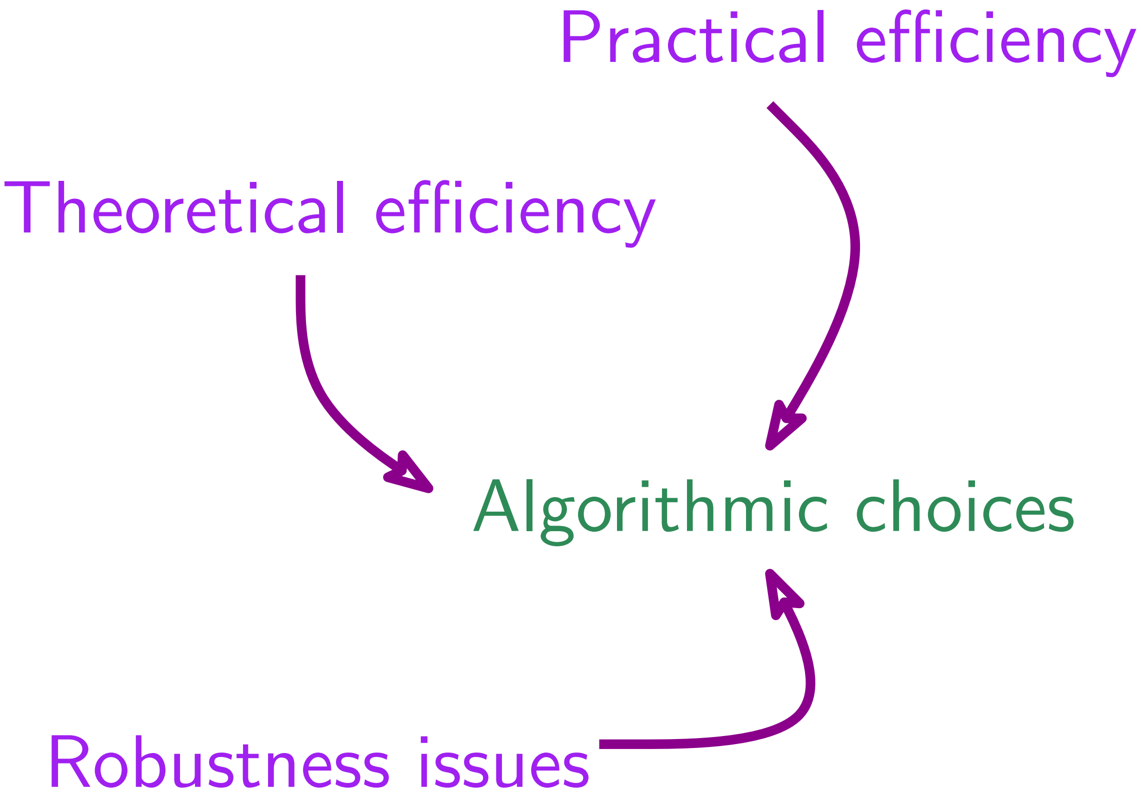


Practical efficiency

Theoretical efficiency

Algorithmic choices

CGAL



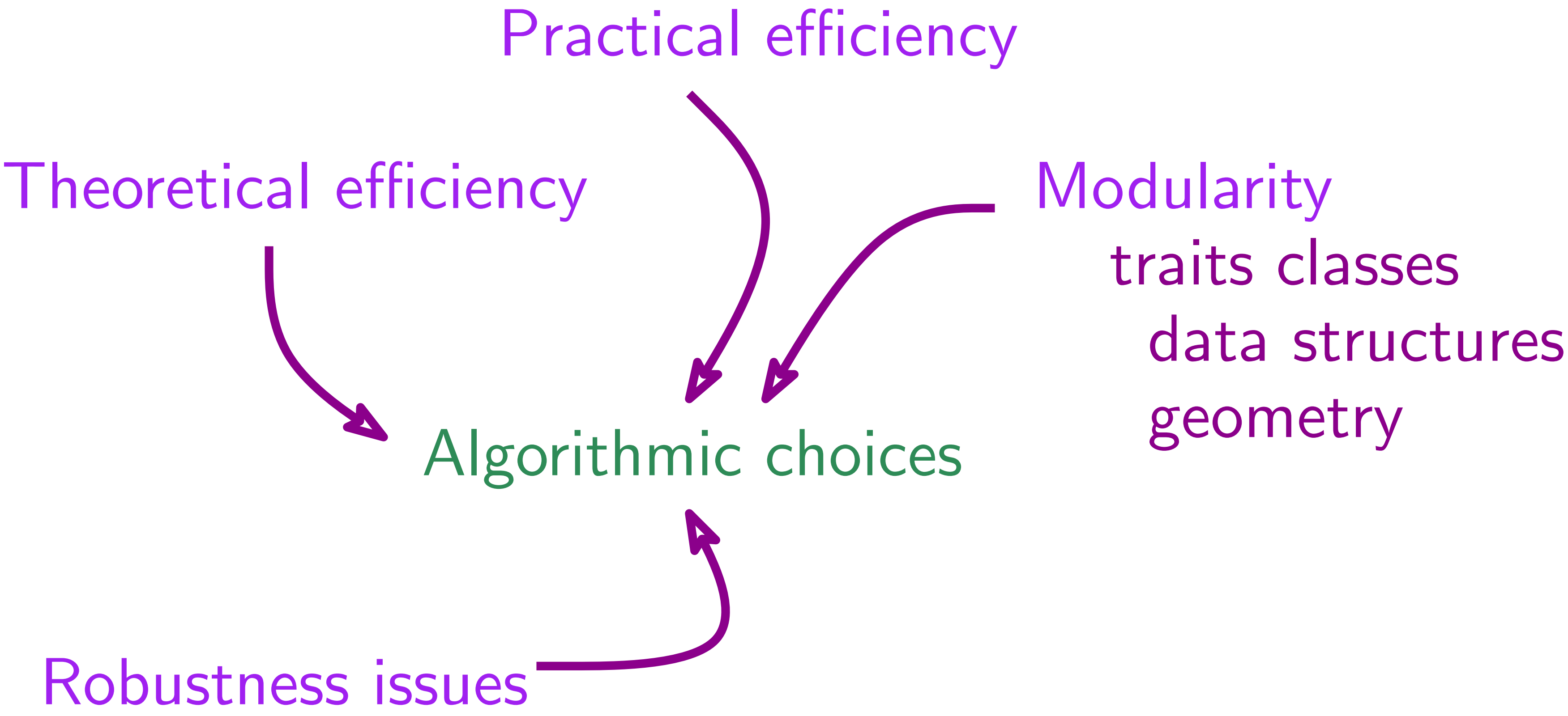
Practical efficiency

Theoretical efficiency

Algorithmic choices

Robustness issues

CGAL



Practical efficiency

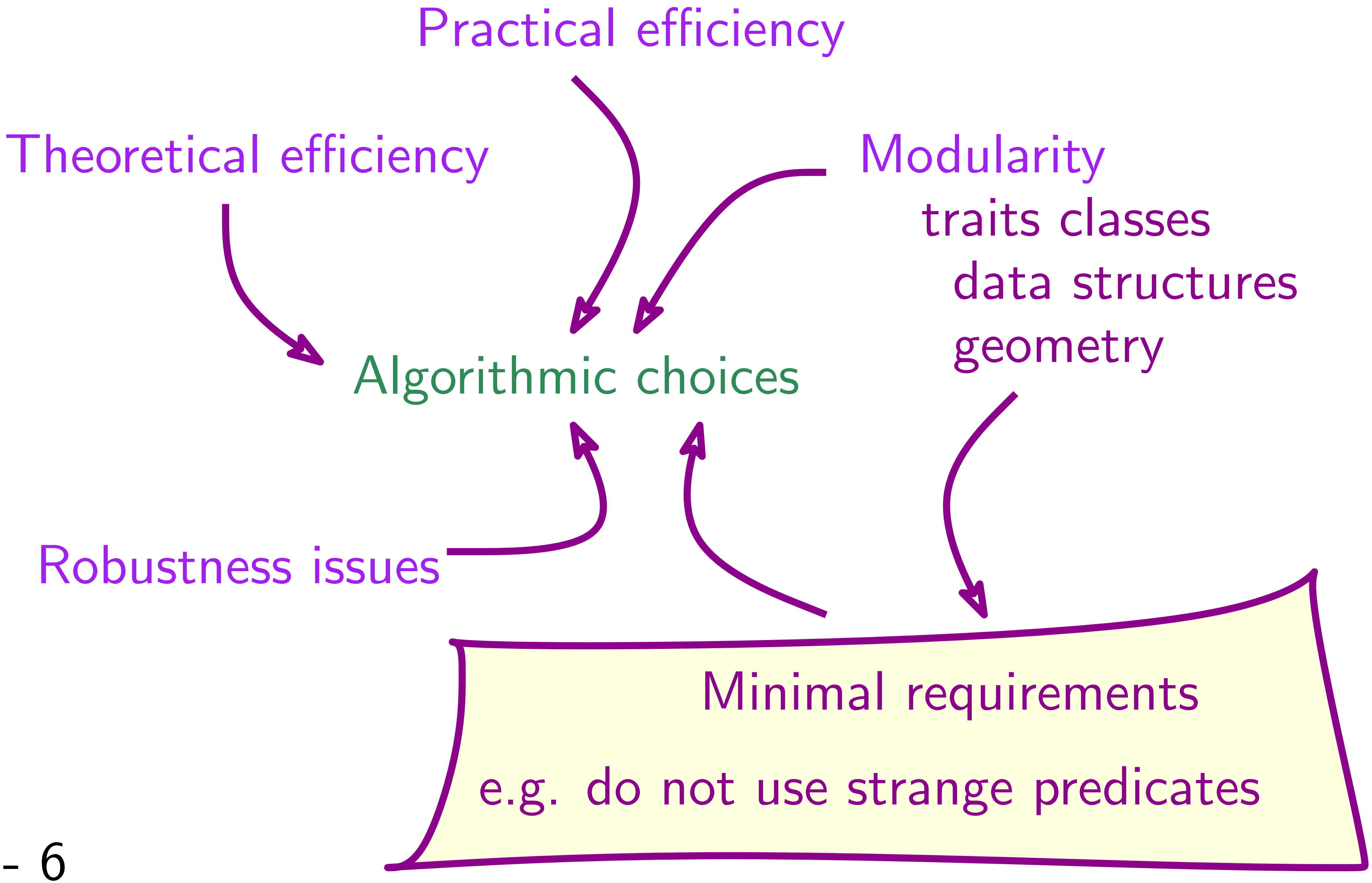
Theoretical efficiency

Modularity
- traits classes
  - data structures
  - geometry

Algorithmic choices

Robustness issues

CGAL



Practical efficiency

Theoretical efficiency

Modularity

traits classes

data structures

geometry

Algorithmic choices

Robustness issues

Minimal requirements

e.g. do not use strange predicates

Usable software subsumes

- clean mathematical foundations
- good algorithms
- adapted programming choices
- (some programming tricks)

- requires people with various skills
- raises interesting research questions

some challenges

Practical vs worst case size of Delaunay 3D

some challenges

Practical vs worst case size of Delaunay 3D

Known results

$\Theta(n^2)$ worst case

$\Theta(n)$ random in ball

$\Omega(n) O(n \log n)$ random on polyhedron

$O(n \log n)$ good sample of smooth generic surface

$\Theta(n \log n)$ random on cylinder

some challenges

Practical vs worst case size of Delaunay 3D

Known results

$\Theta(n^2)$ worst case

Find good models of practical data

$\Theta(n)$ random in ball

(Smooth analysis)

$\Omega(n)O(n \log n)$ random on polyhedron

$O(n \log n)$ good sample of smooth generic surface

$\Theta(n \log n)$ random on cylinder

some challenges

Practical vs worst case size of Delaunay 3D

Better algorithm for 3D deletion

$10\ \mu$s to insert

$100\ \mu$s to delete

some challenges

Practical vs worst case size of Delaunay 3D

Better algorithm for 3D deletion

One billion points

- Needs memory efficient algorithms
- Cache effects are already important

CGAL

demos

web site `www.cgal.org`