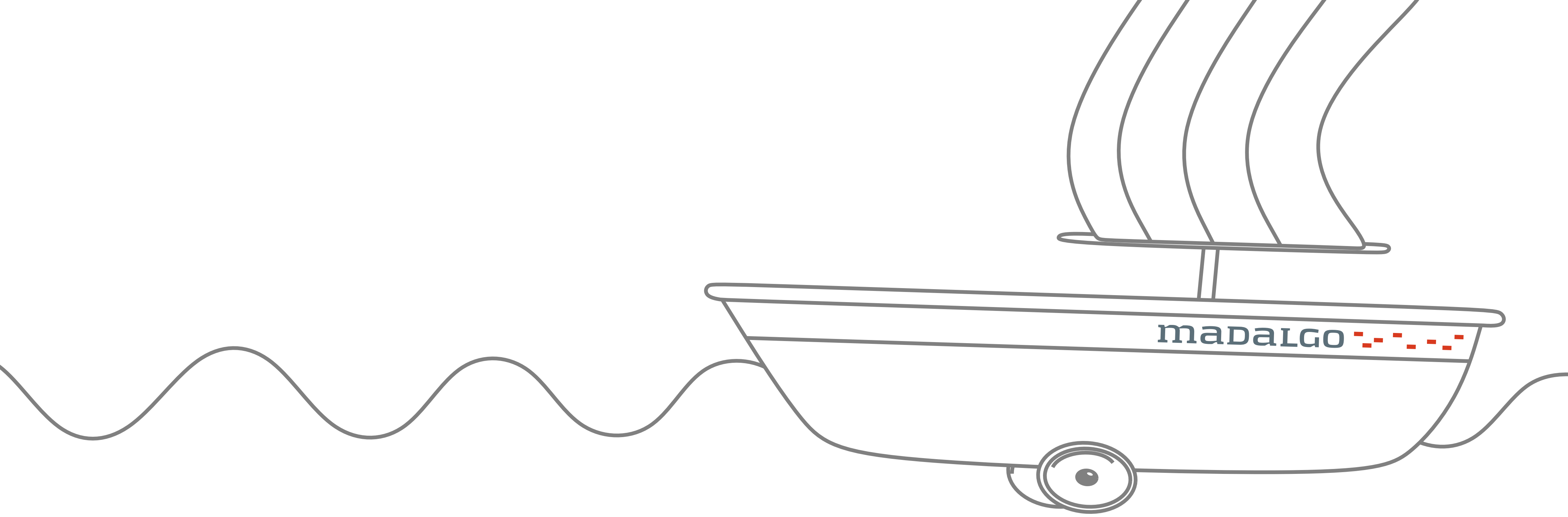


Cleaning massive sonar point clouds

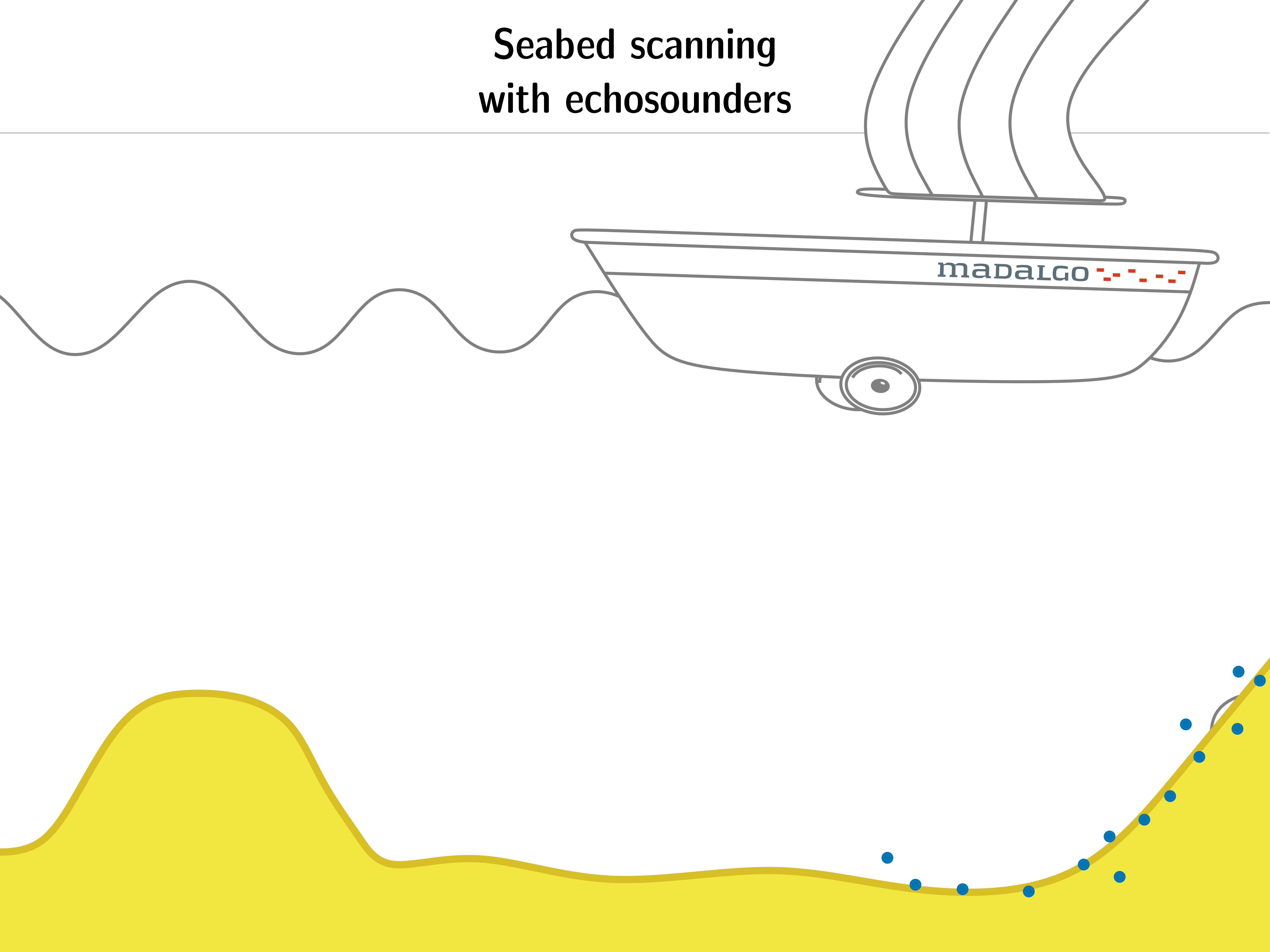
Lars Arge
Aarhus University

Kasper Green Larsen
Aarhus University

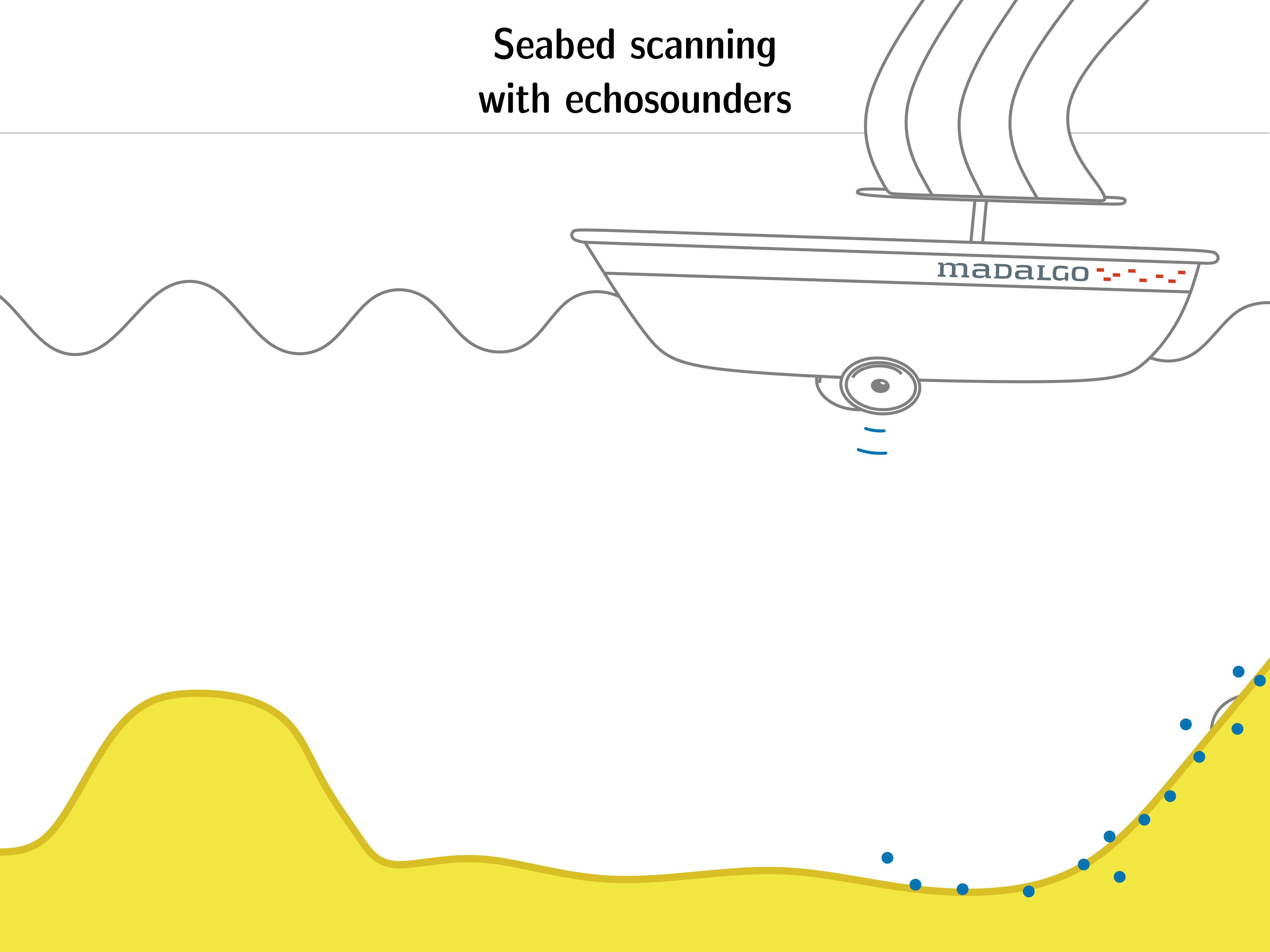
Thomas Mølhave
Duke University

<u>Freek van Walderveen</u>
Aarhus University

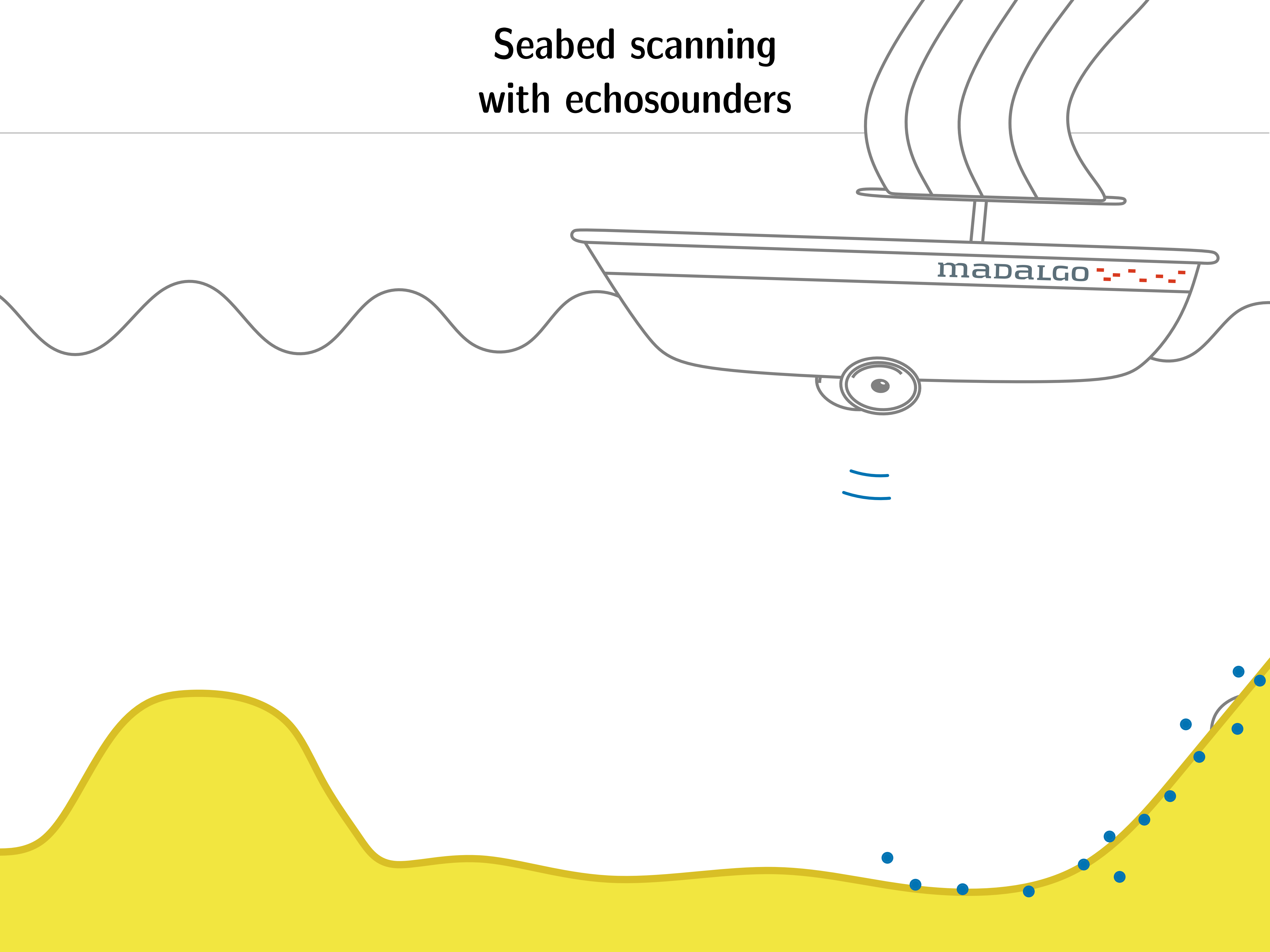
Seabed scanning with echosounders



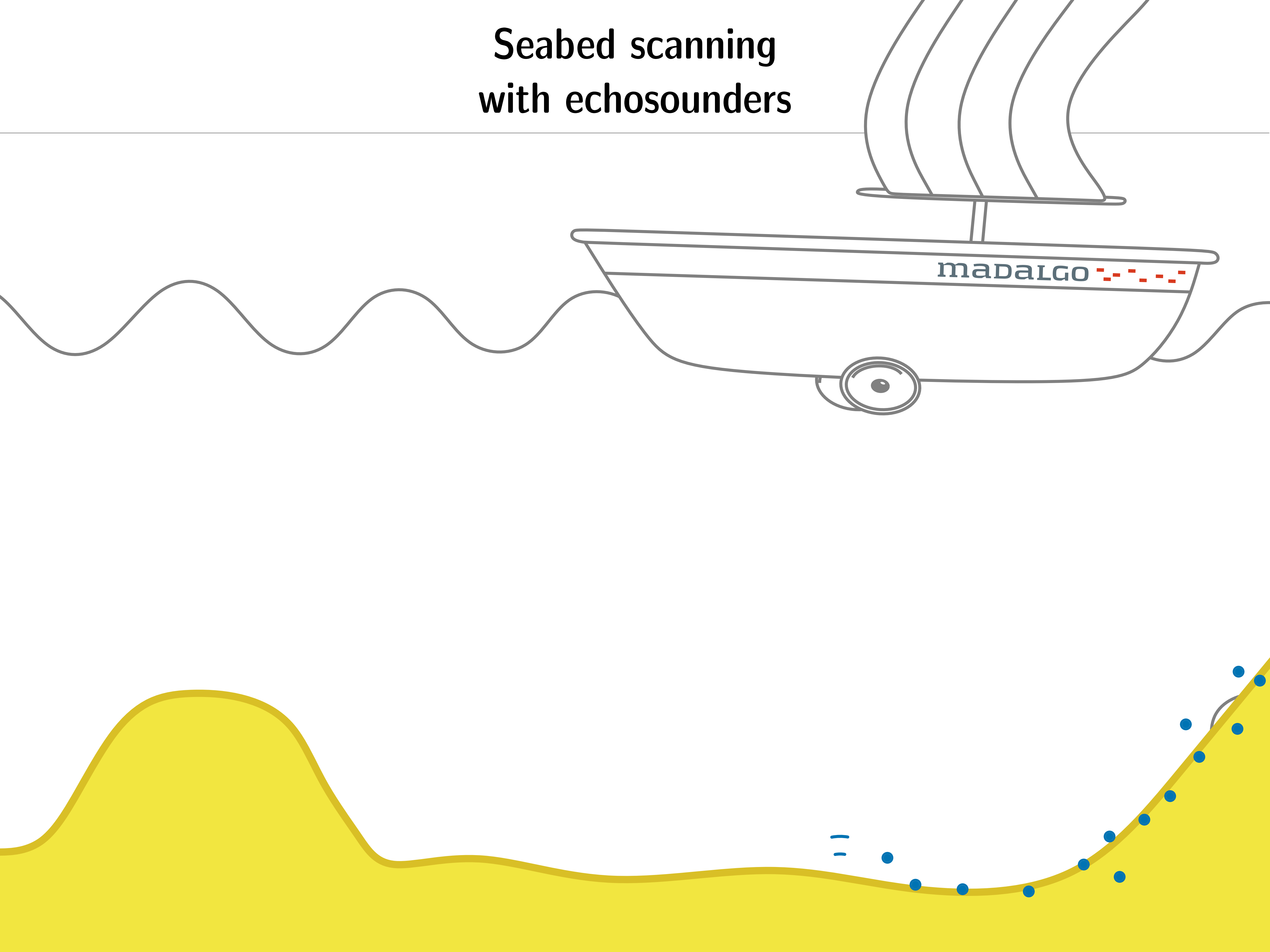
Seabed scanning with echosounders



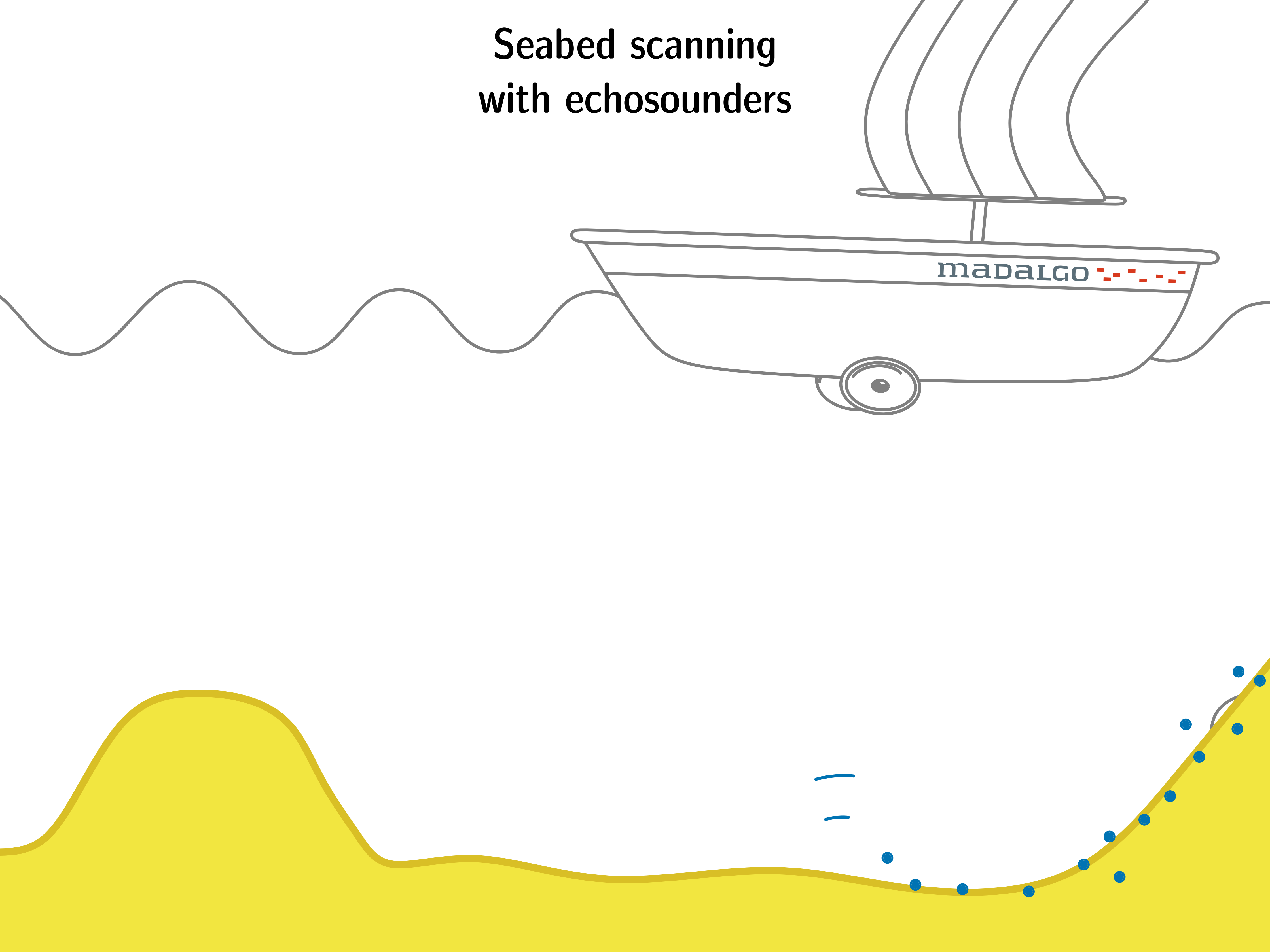
Seabed scanning with echosounders



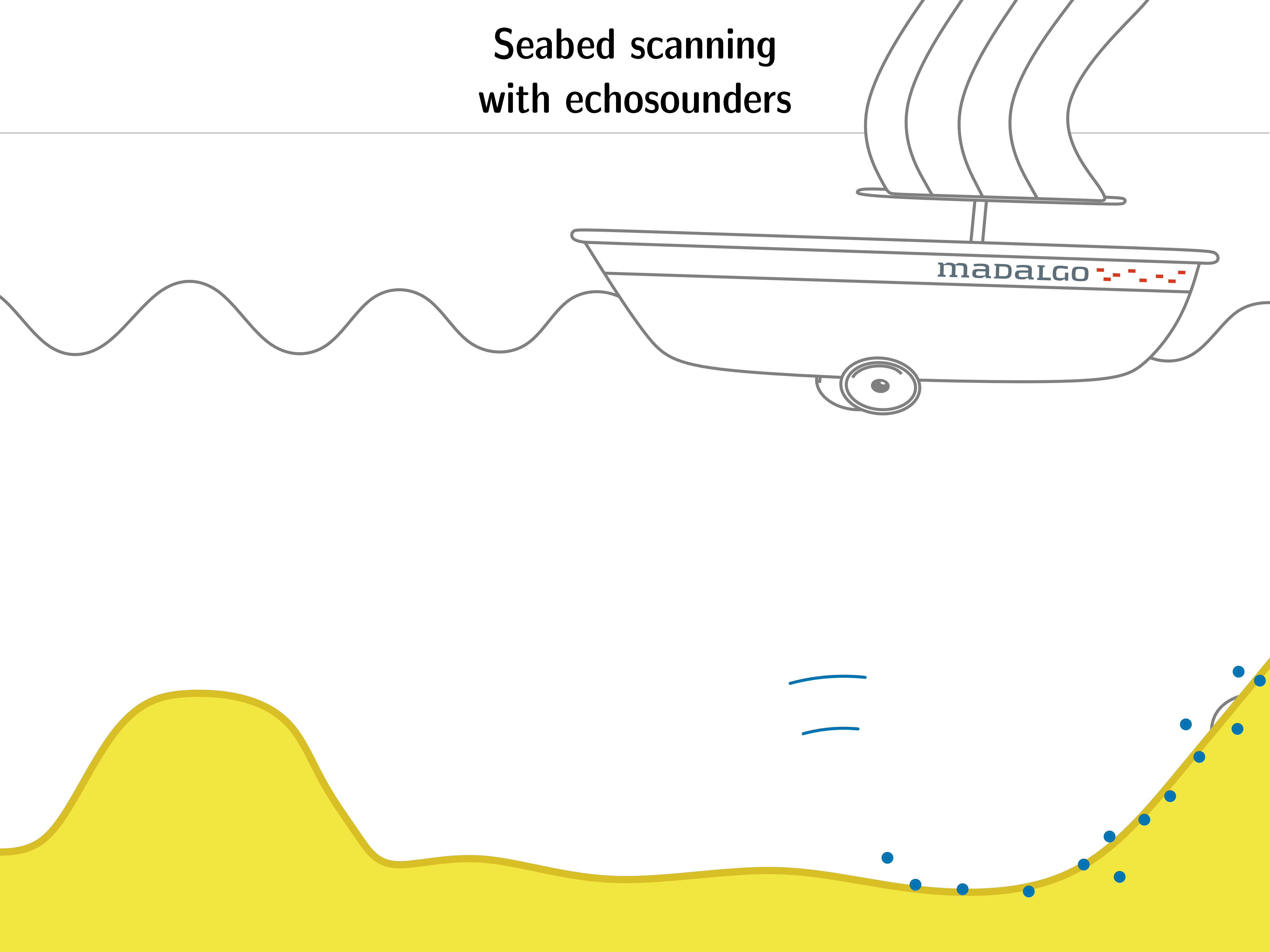
Seabed scanning with echosounders



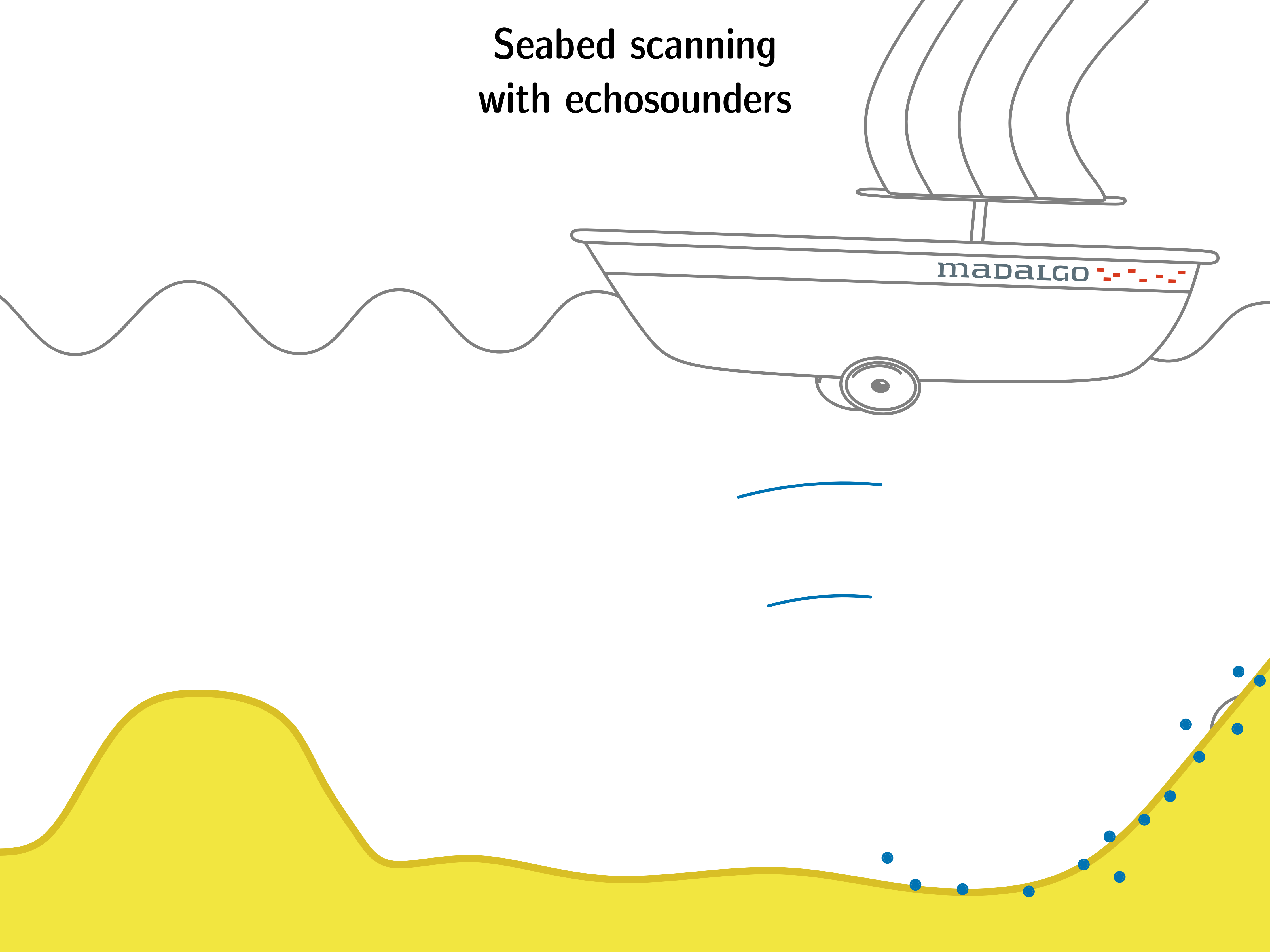
Seabed scanning with echosounders



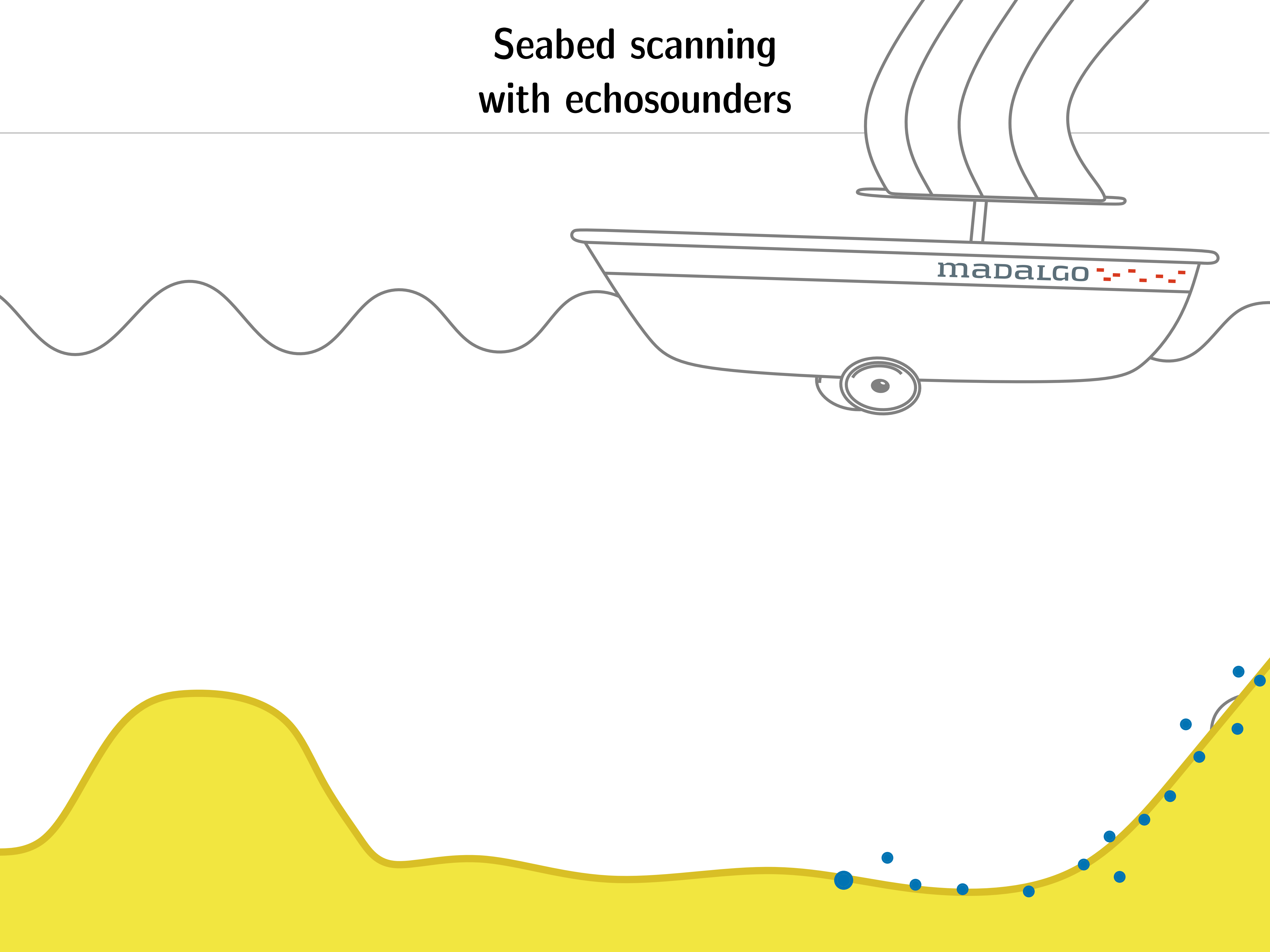
Seabed scanning
with echosounders



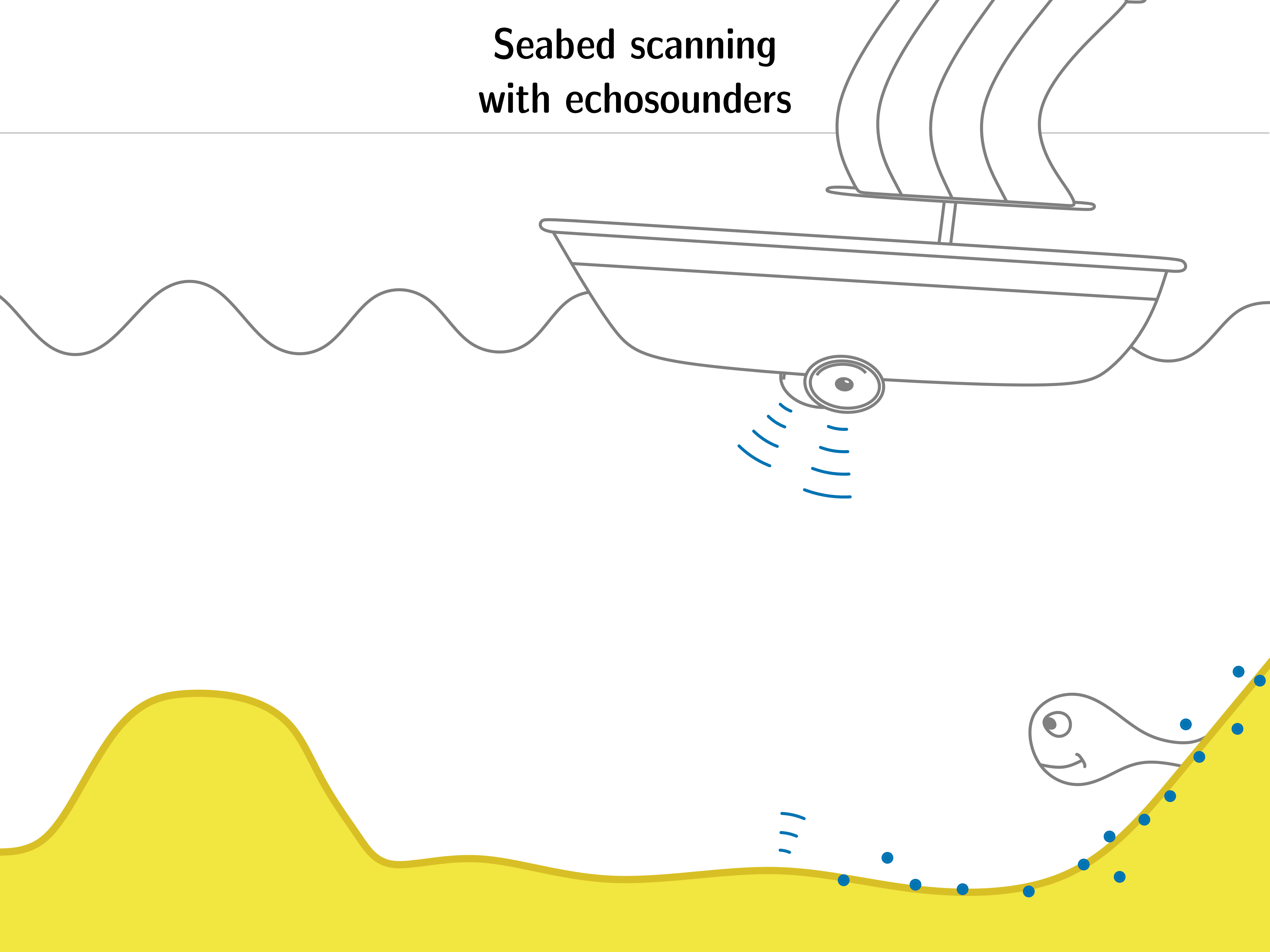
Seabed scanning with echosounders



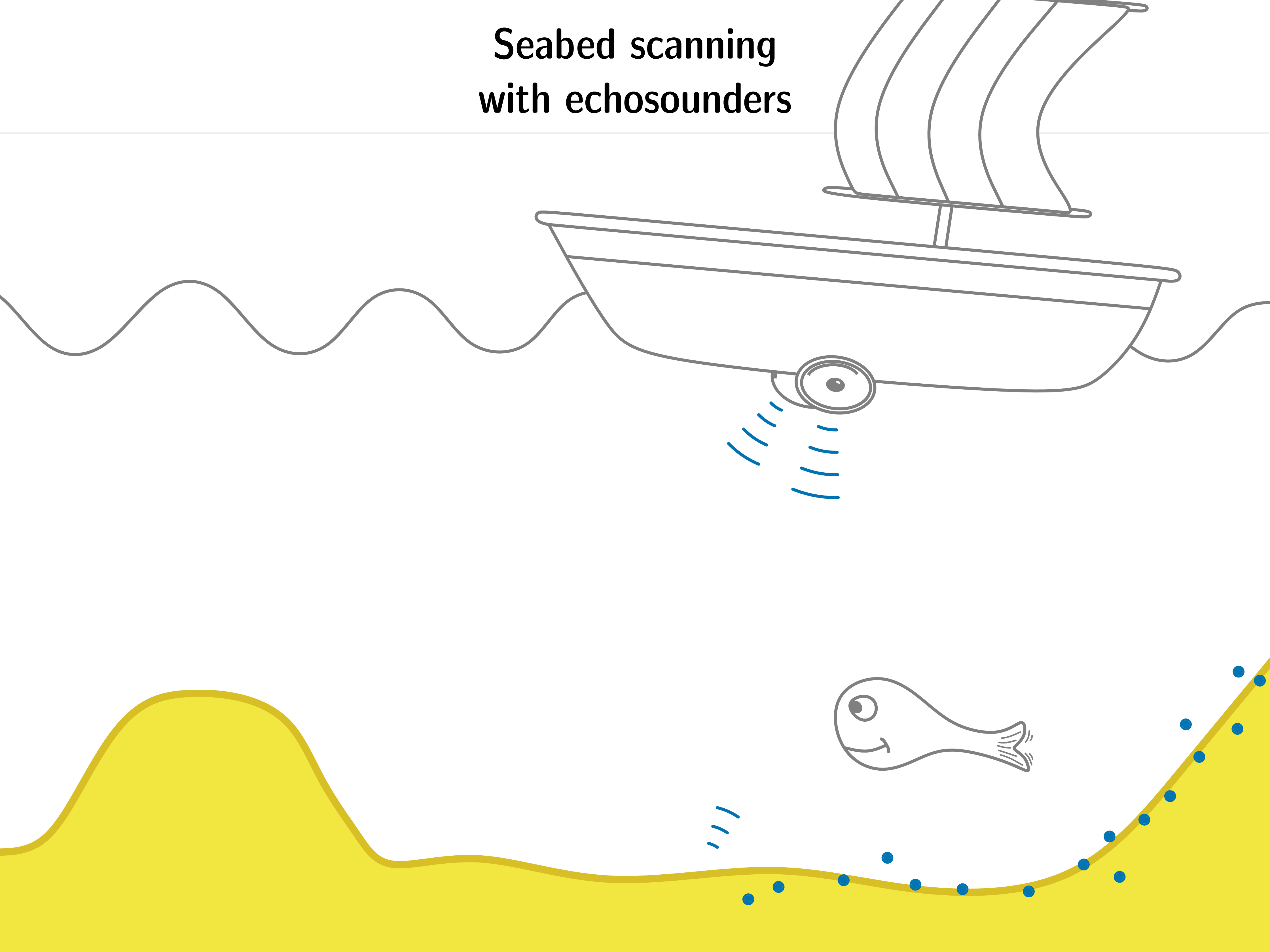
Seabed scanning with echosounders



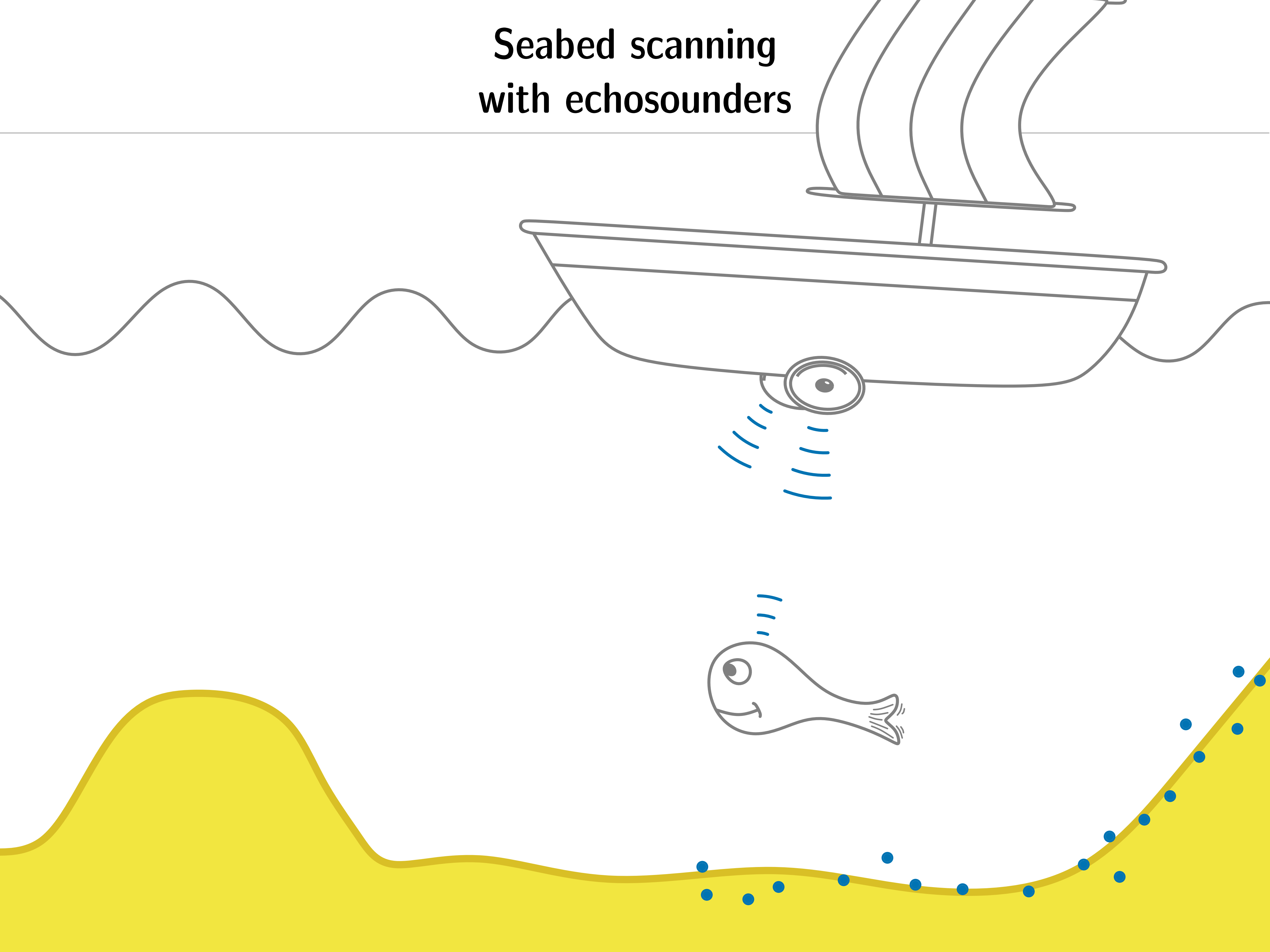
Seabed scanning with echosounders



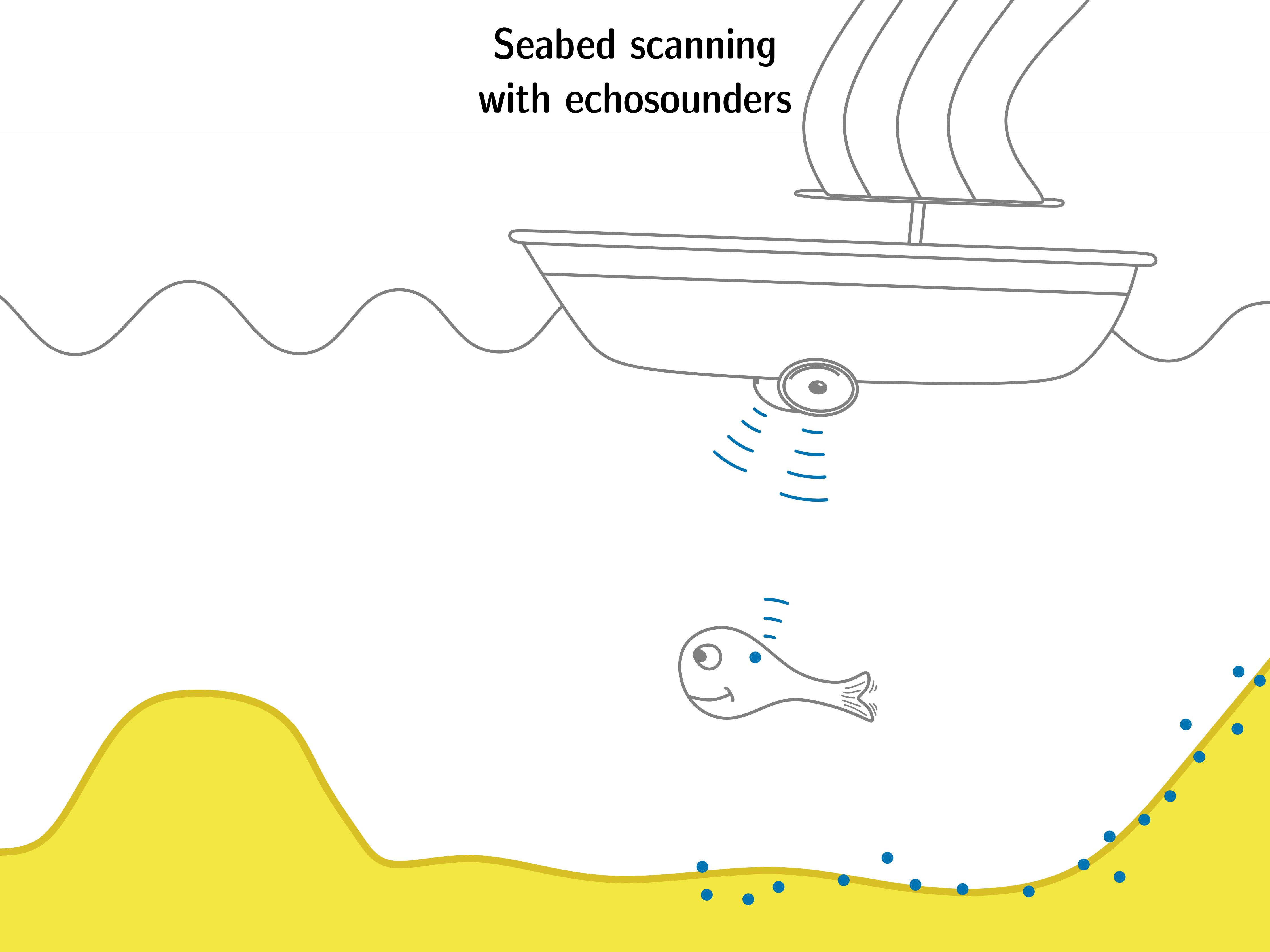
Seabed scanning with echosounders



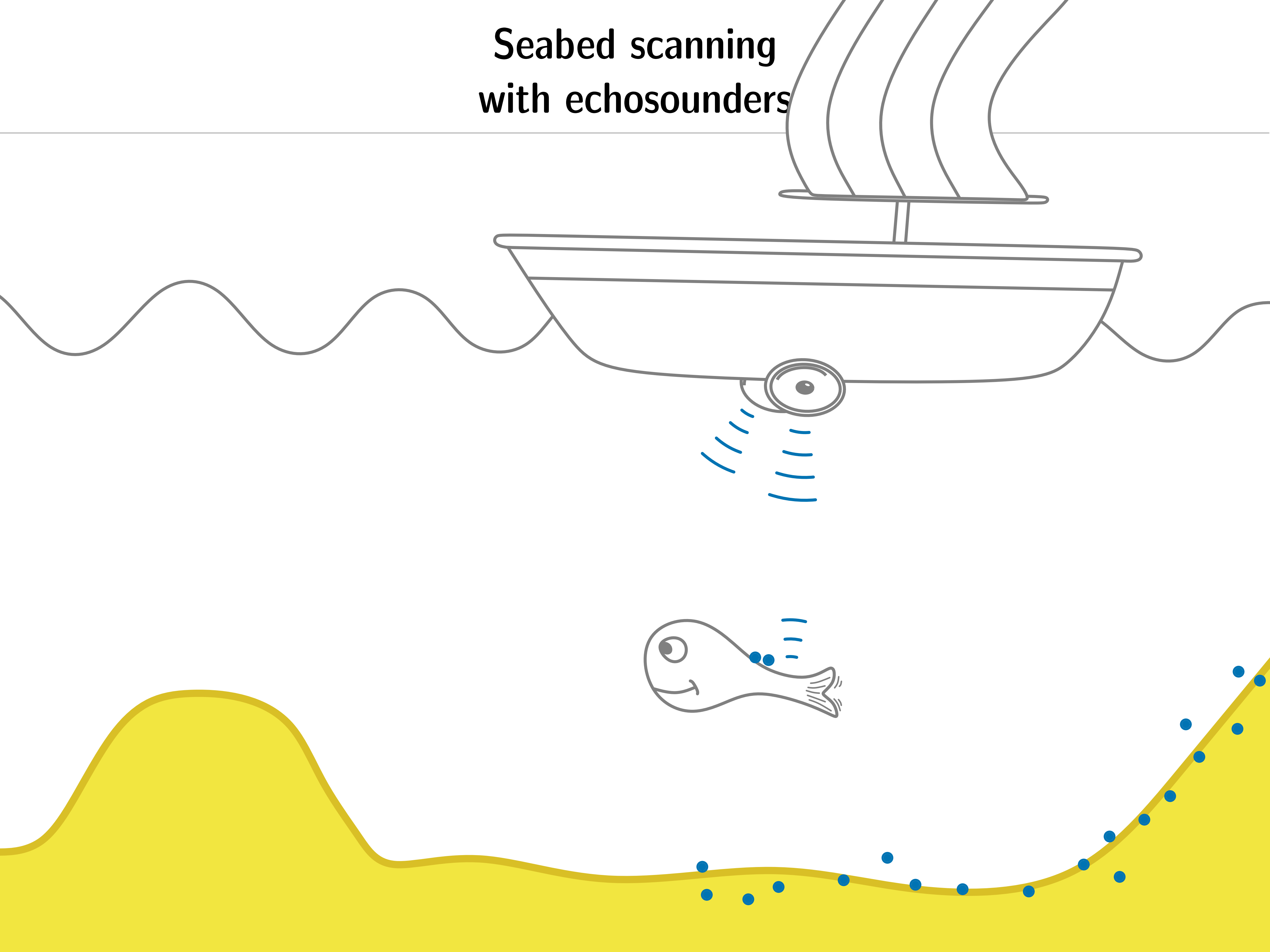
Seabed scanning with echosounders



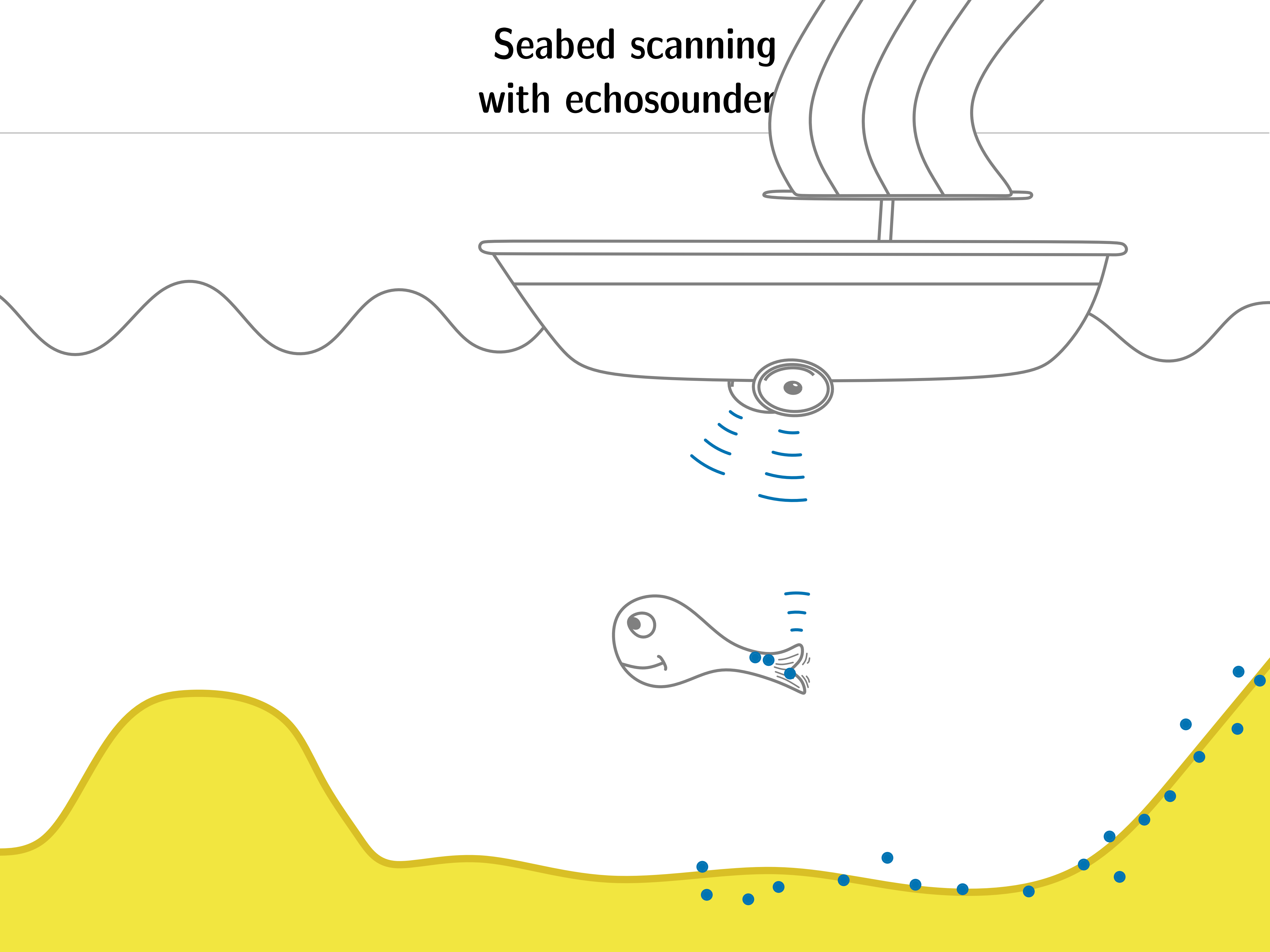
Seabed scanning with echosounders



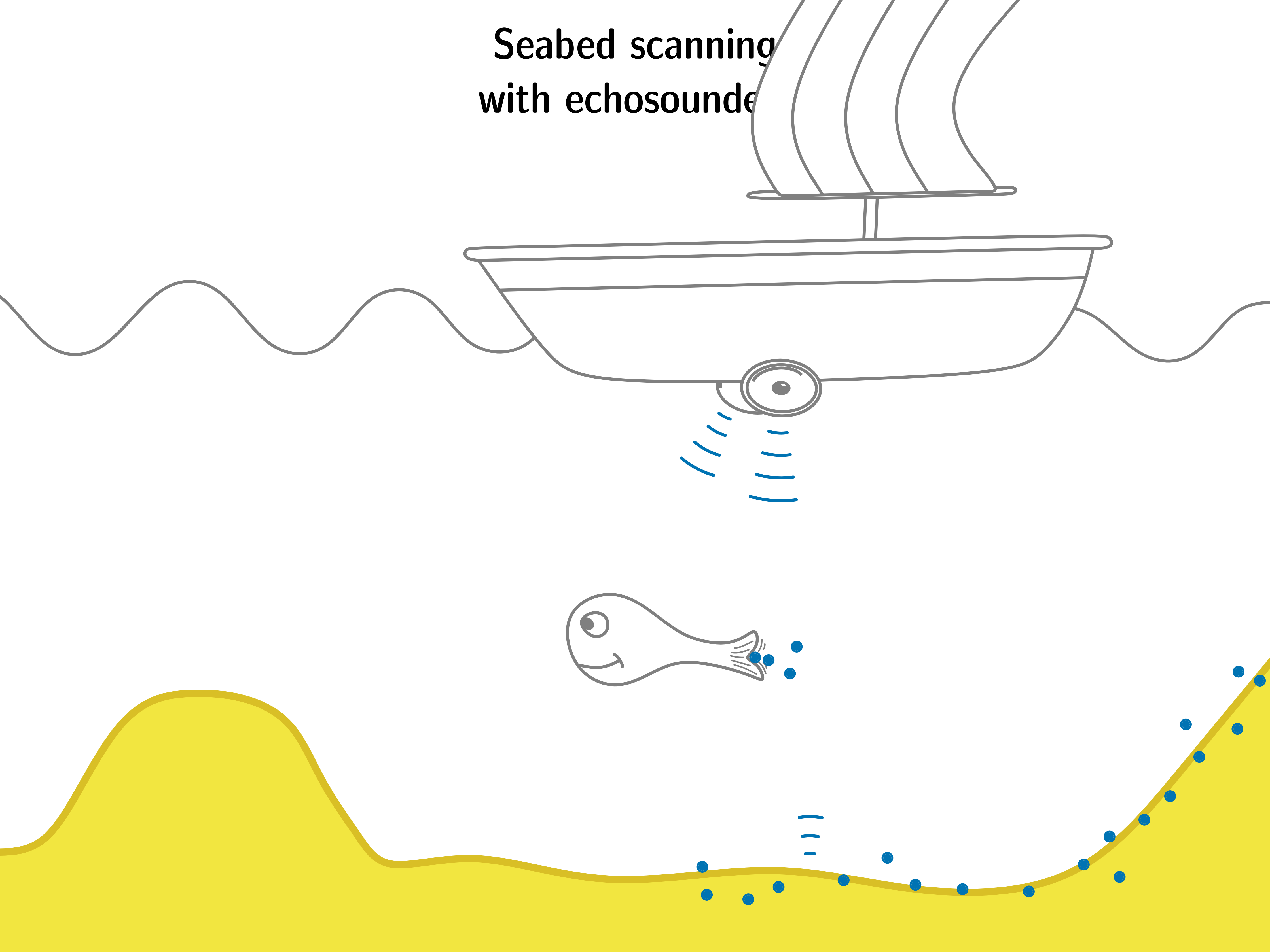
Seabed scanning with echosounders



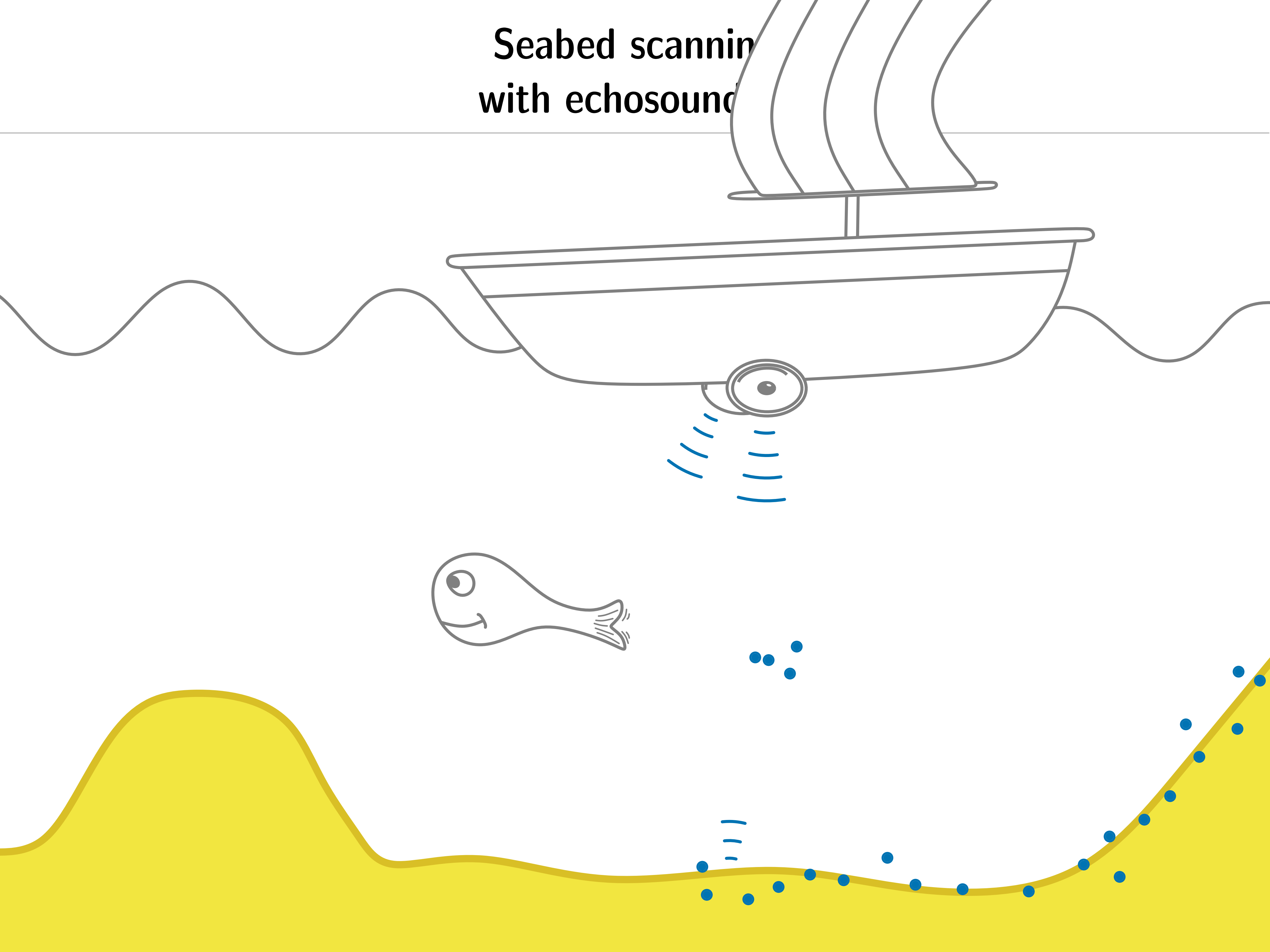
Seabed scanning with echosounder



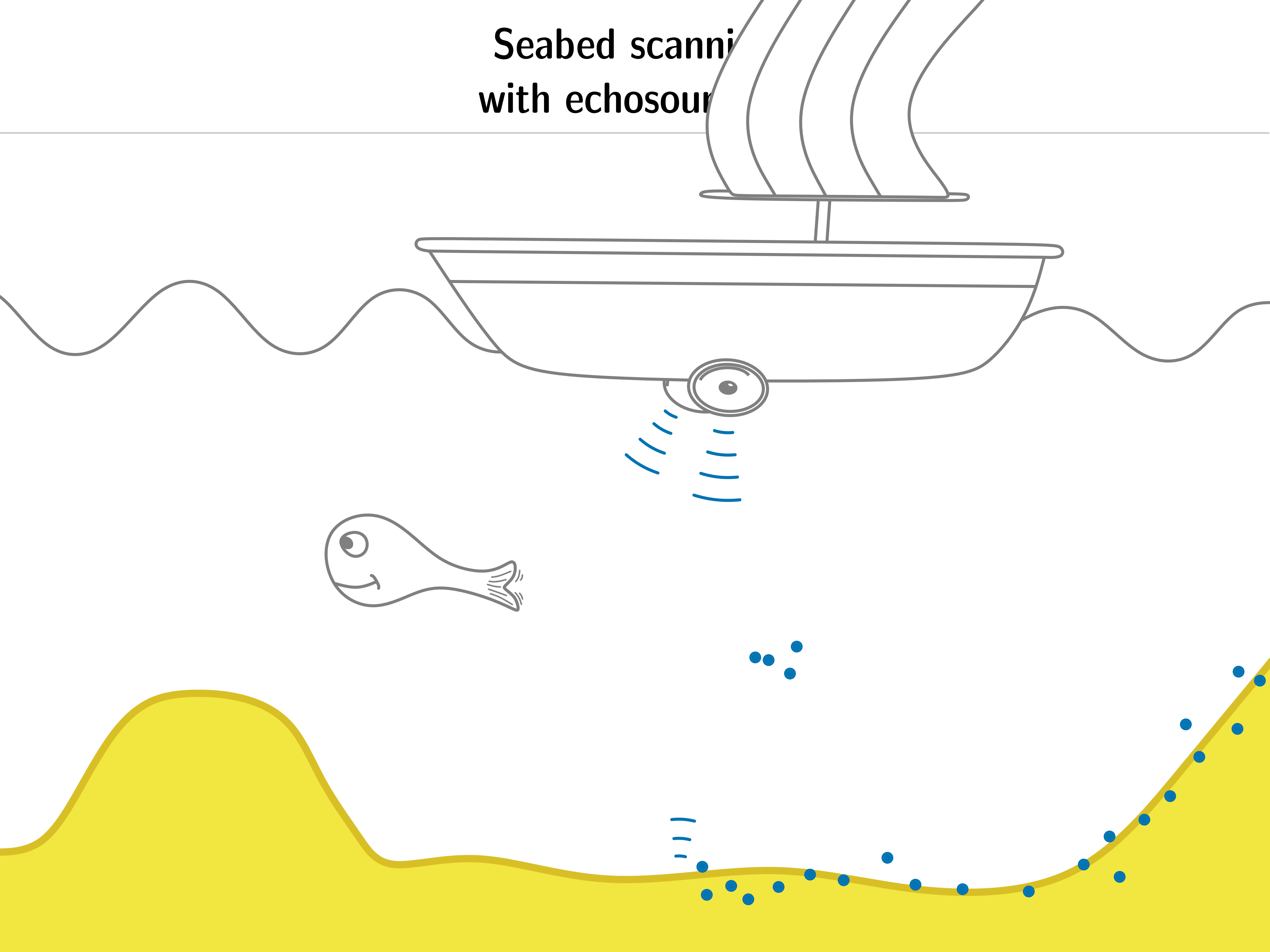
Seabed scanning
with echosounde



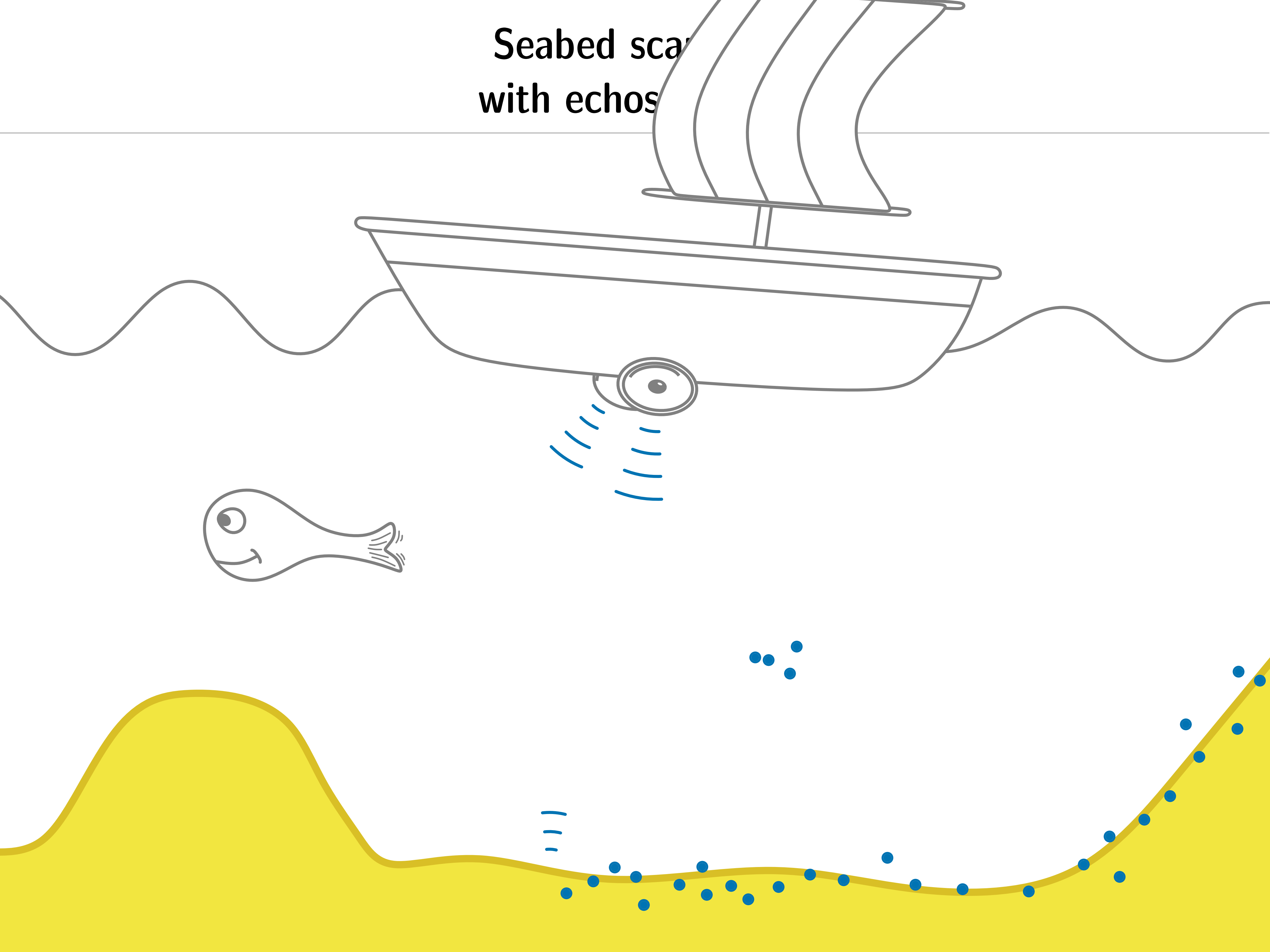
Seabed scannin
with echosound



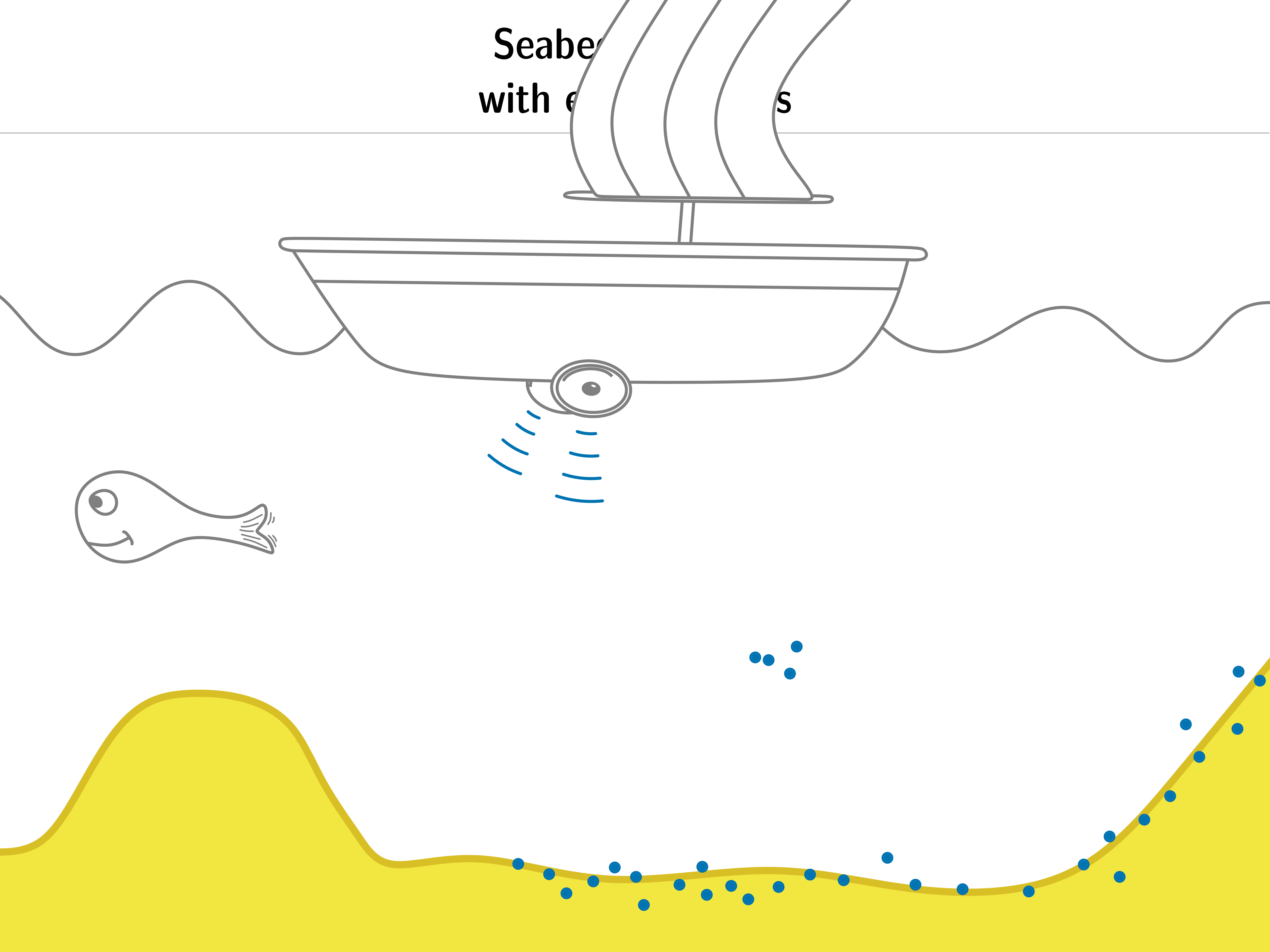
Seabed scanni with echosour



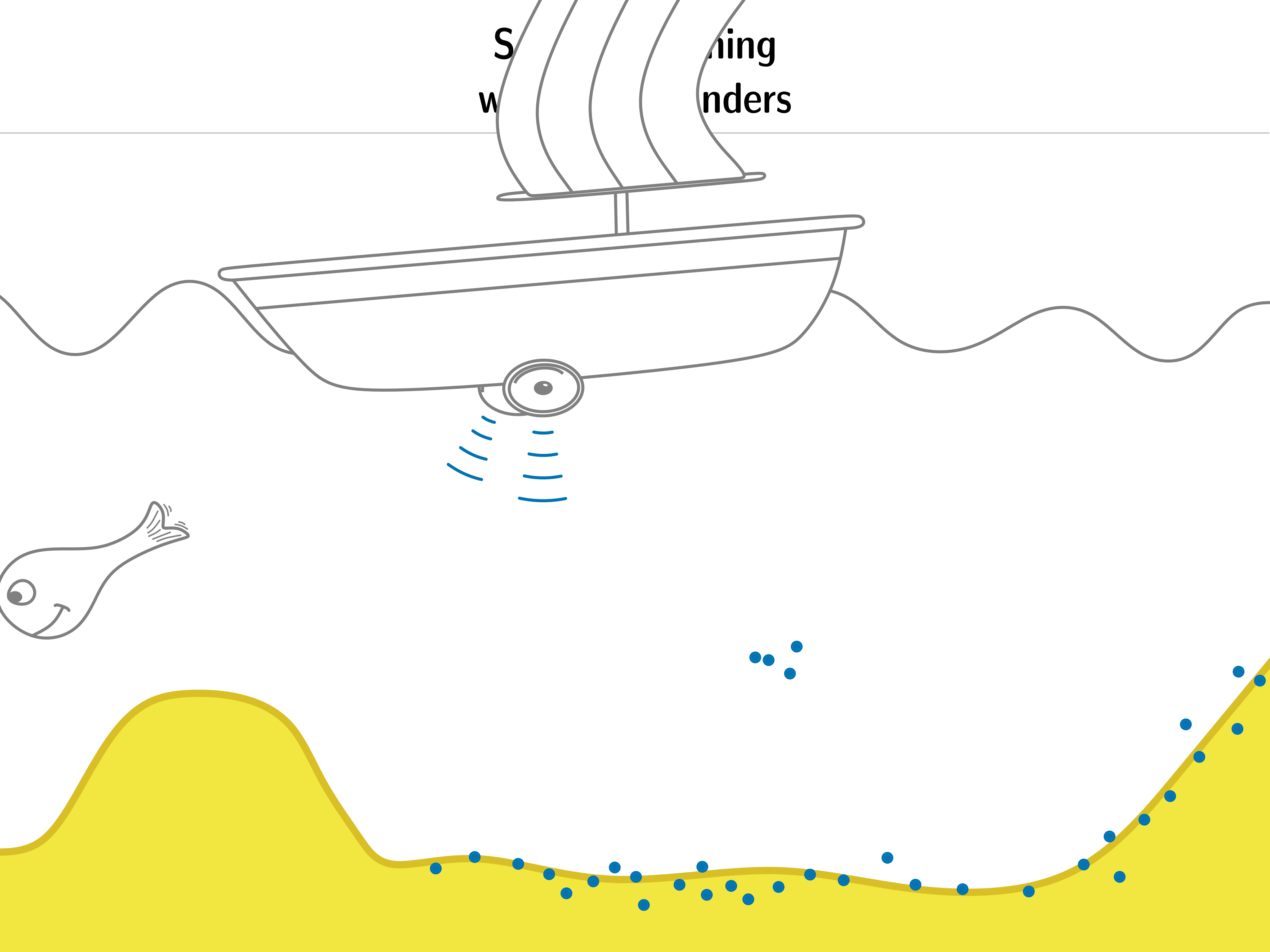
Seabed sca
with echos



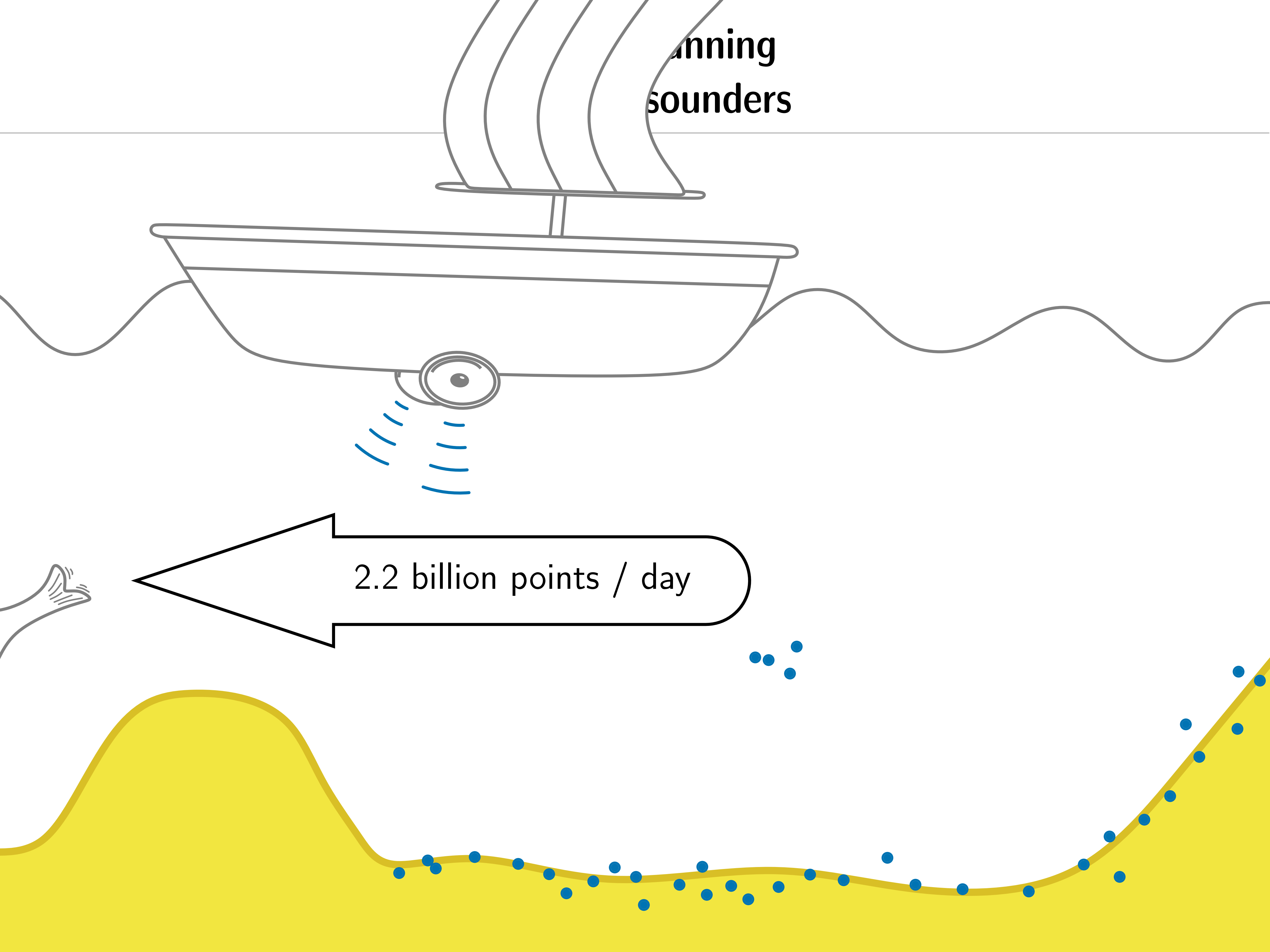
Seabe
with e s



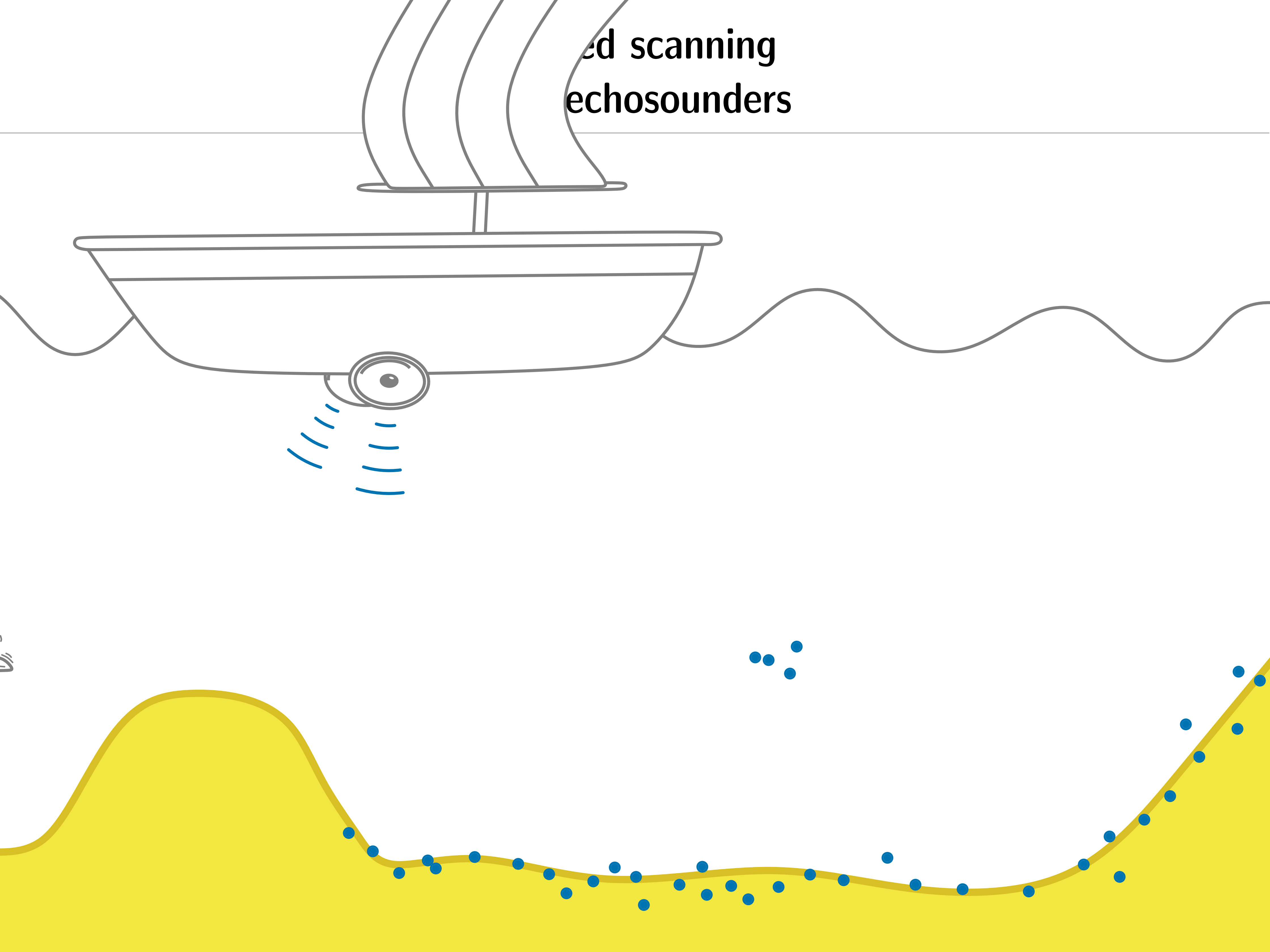
S ning
w nders



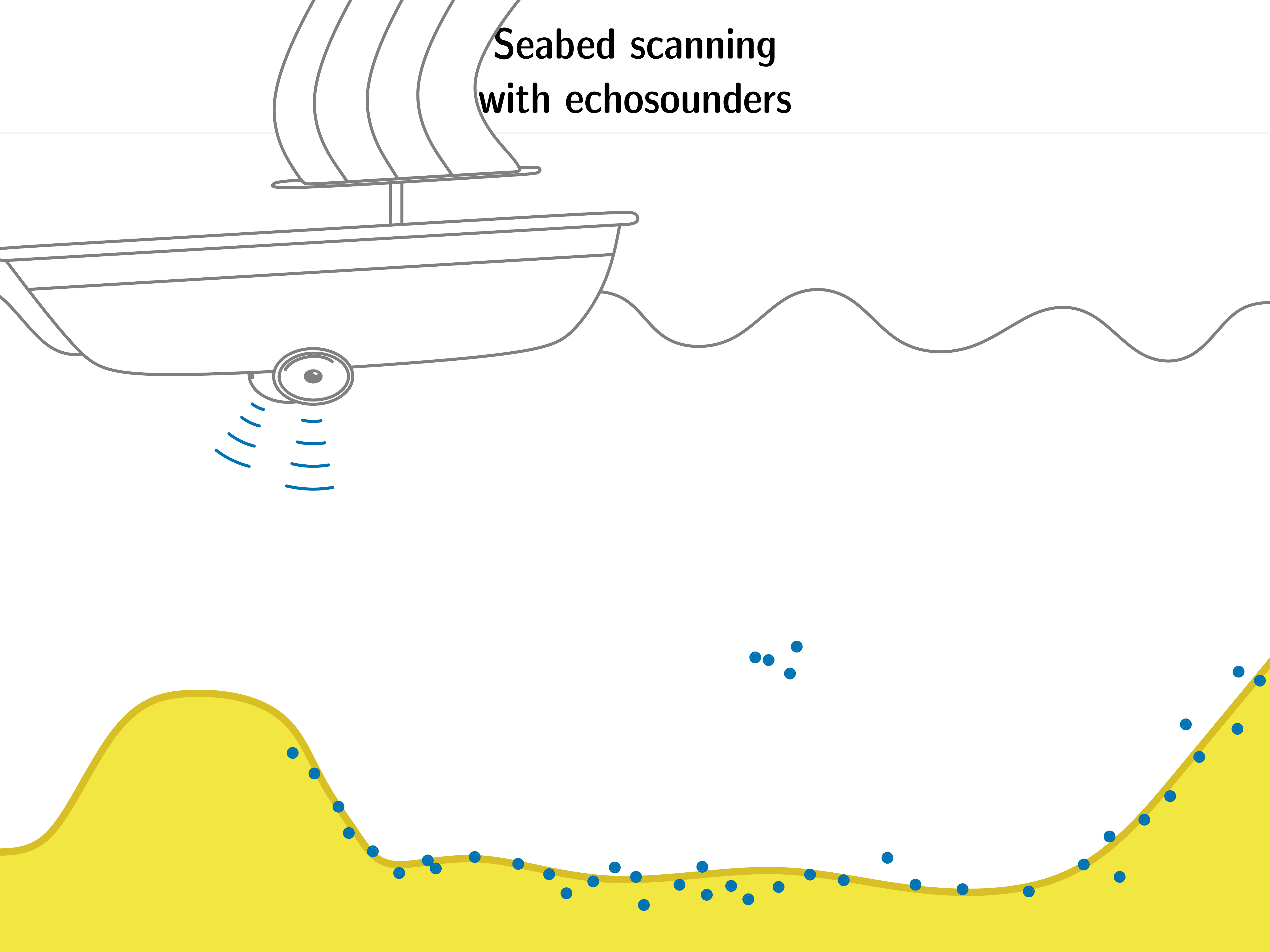
nning sounders



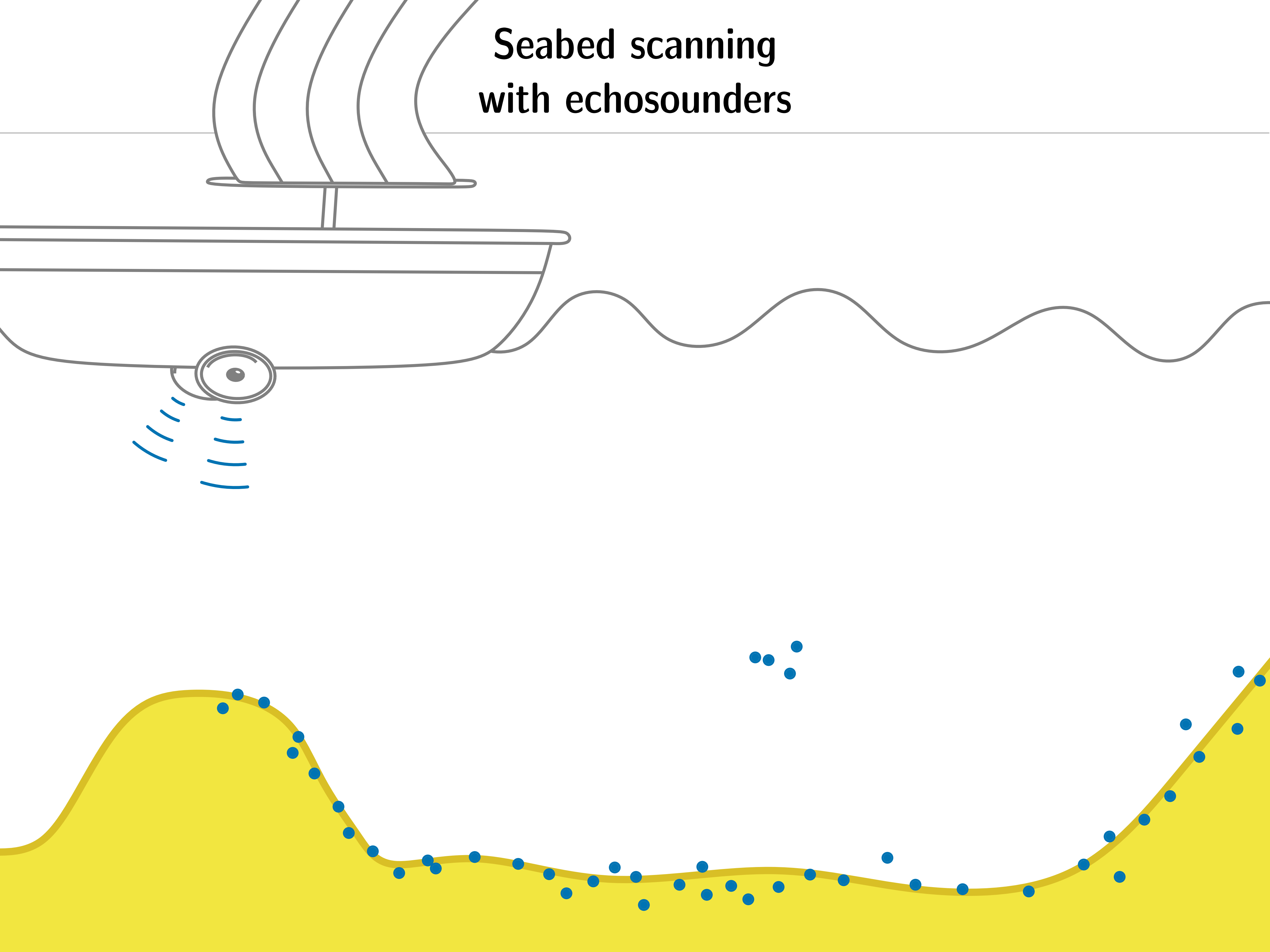
ed scanning
echosounders



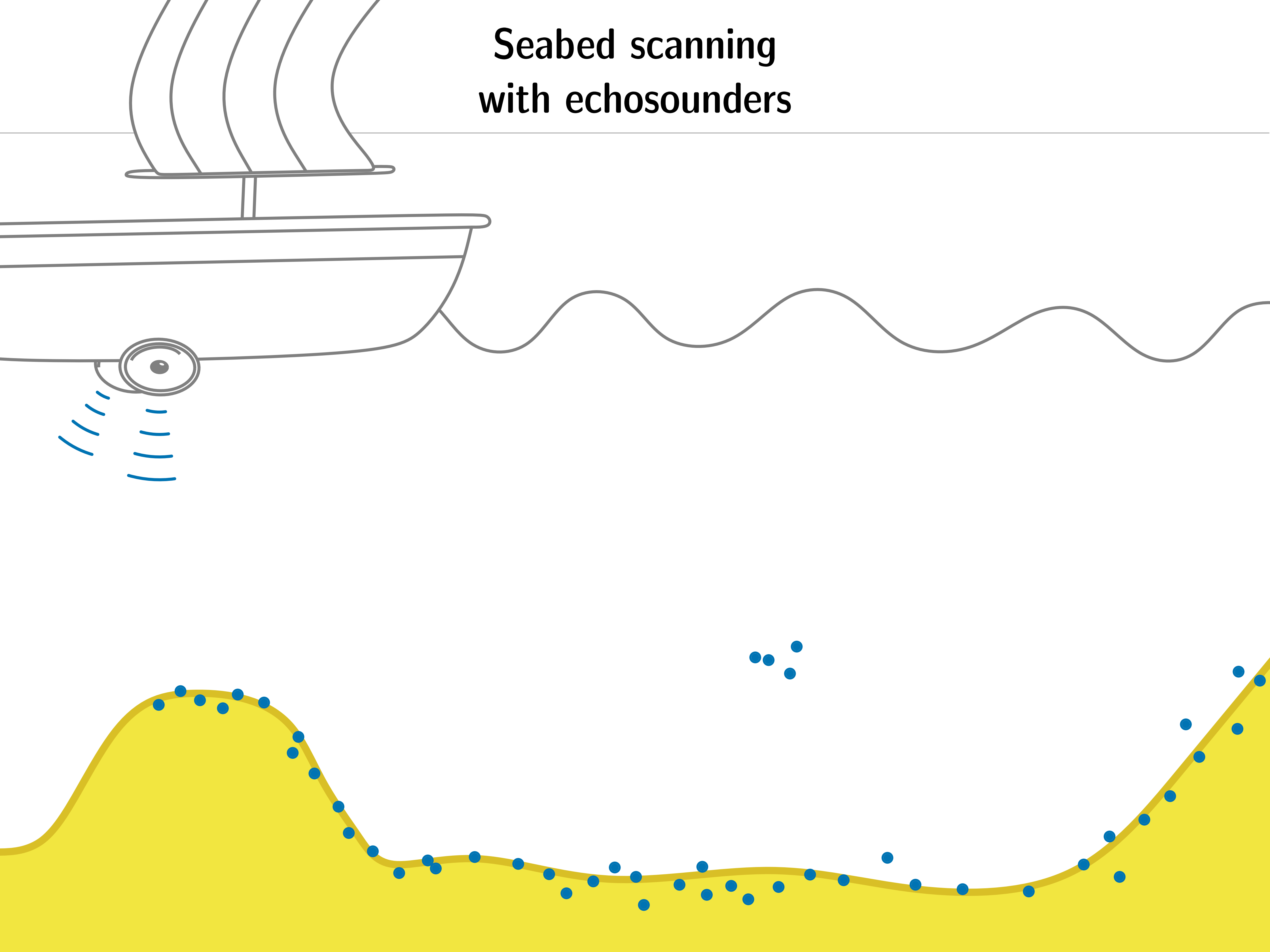
Seabed scanning with echosounders



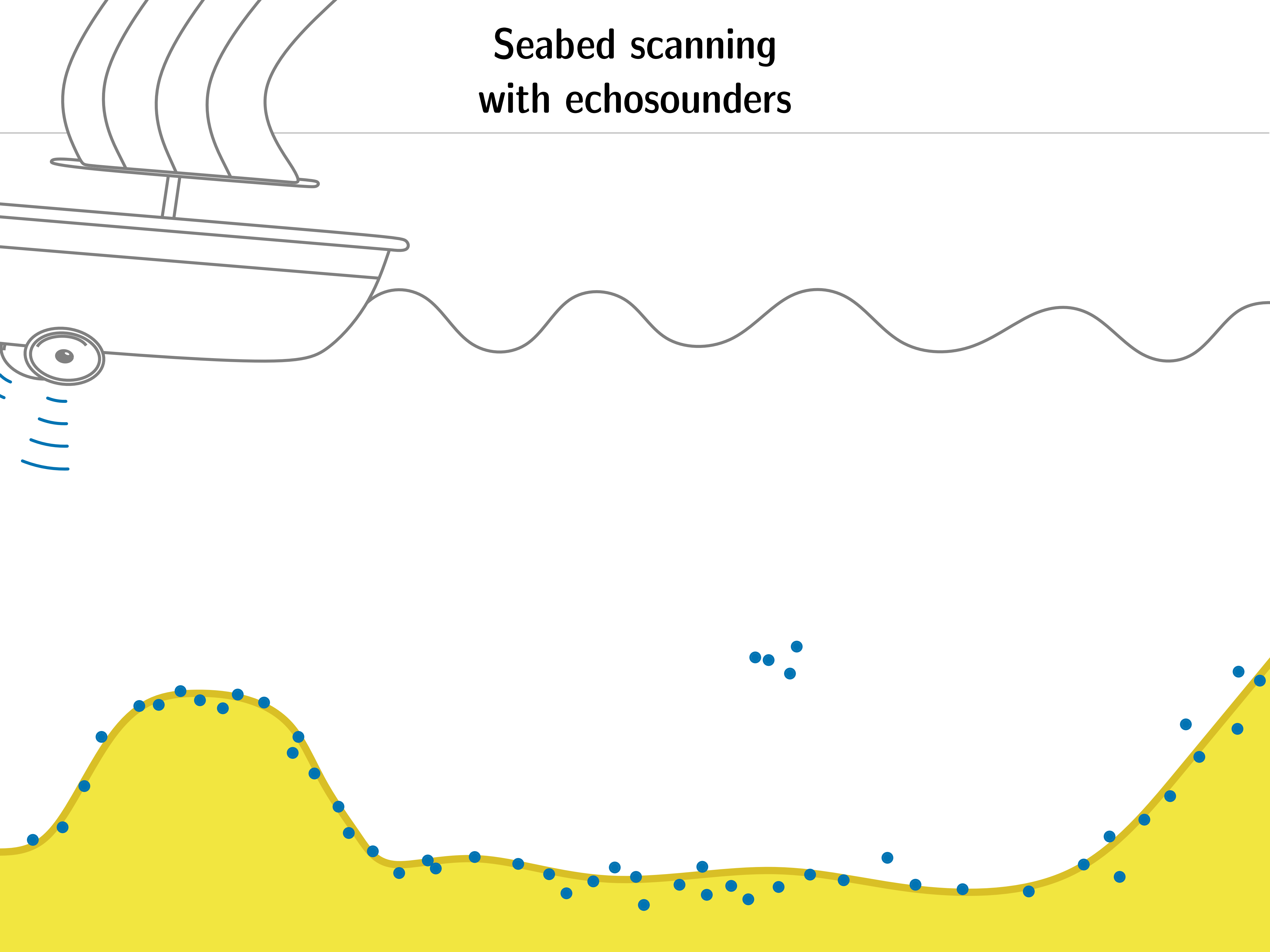
Seabed scanning with echosounders



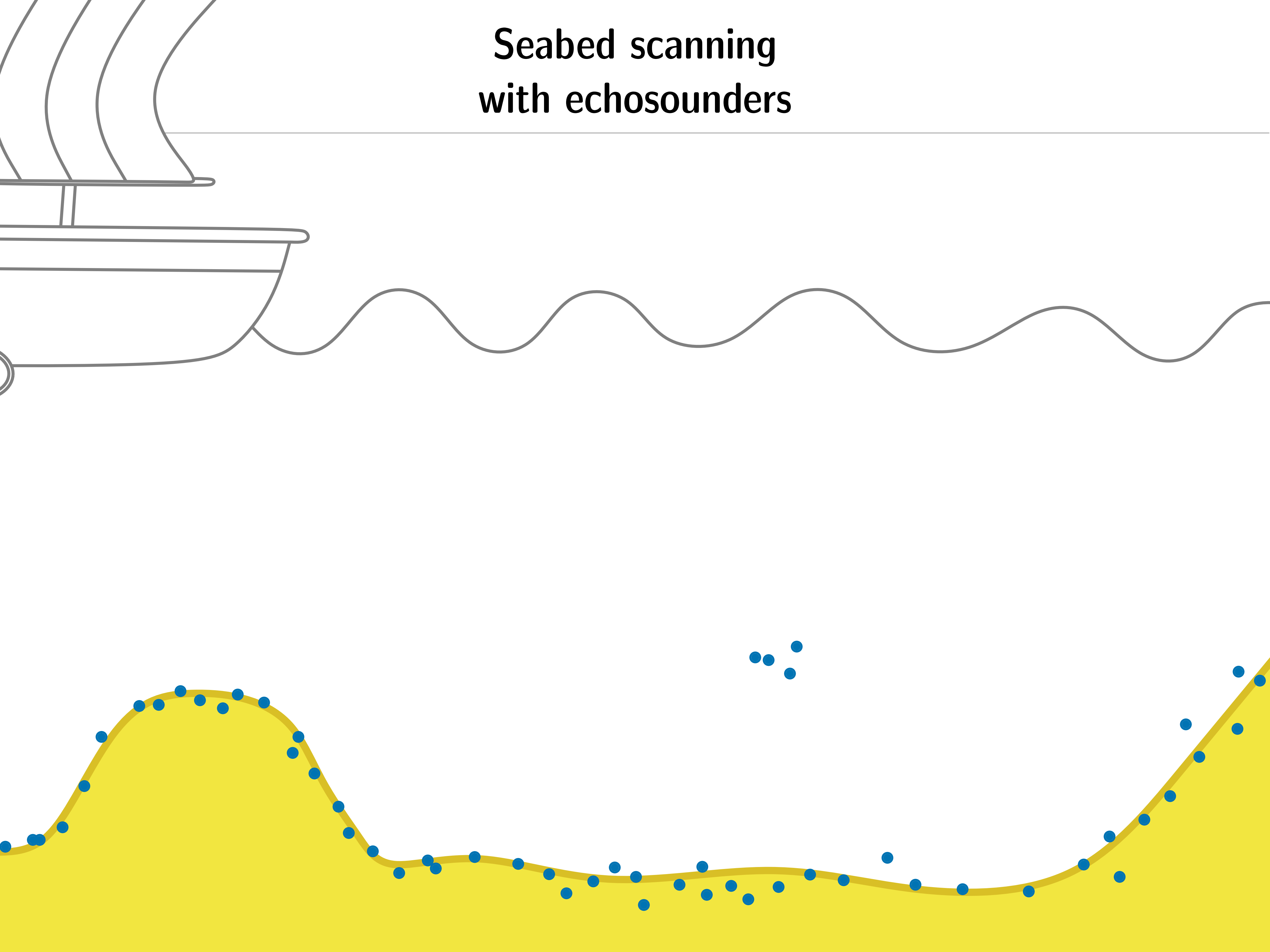
Seabed scanning with echosounders



Seabed scanning with echosounders

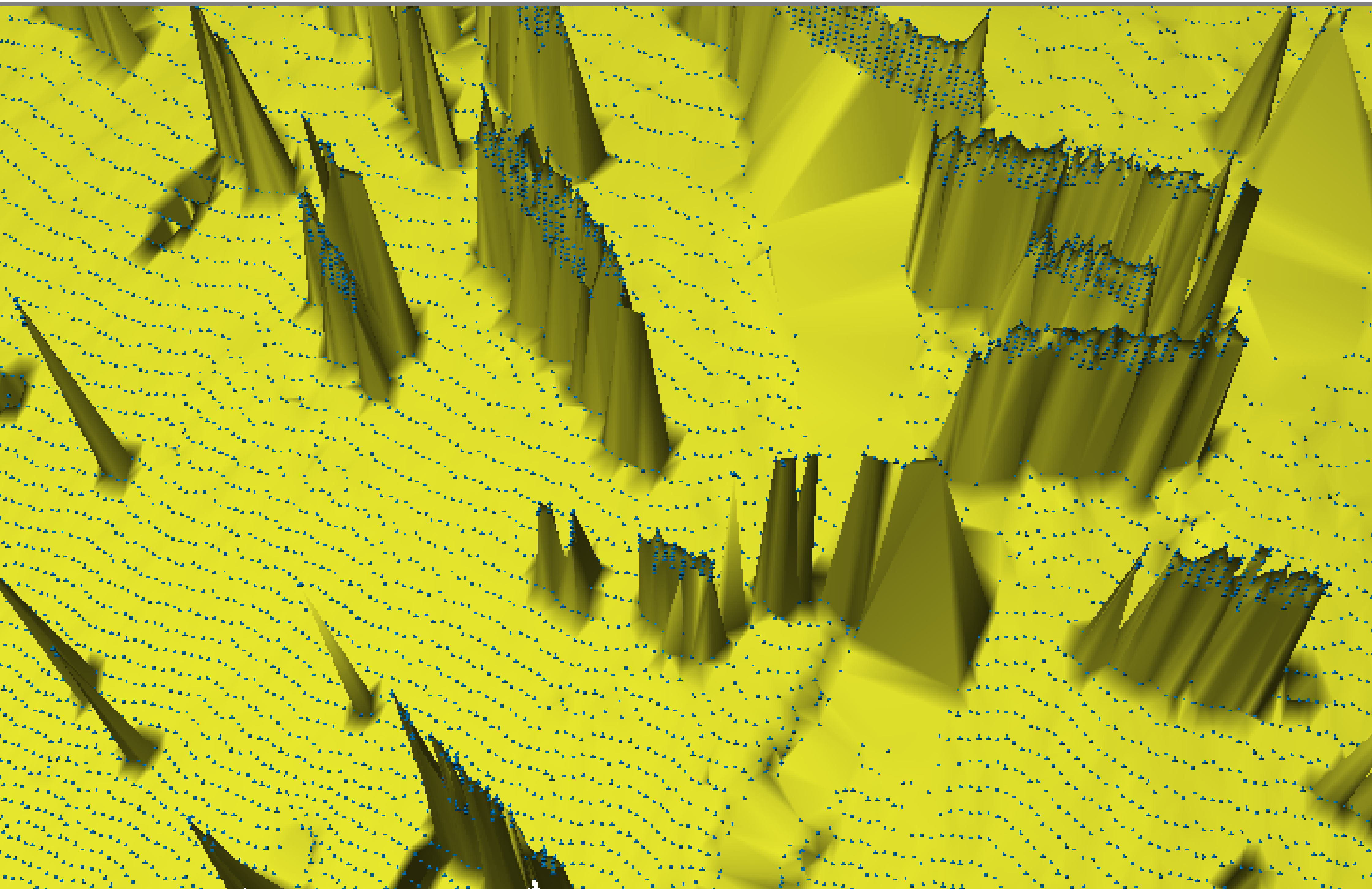


Seabed scanning with echosounders



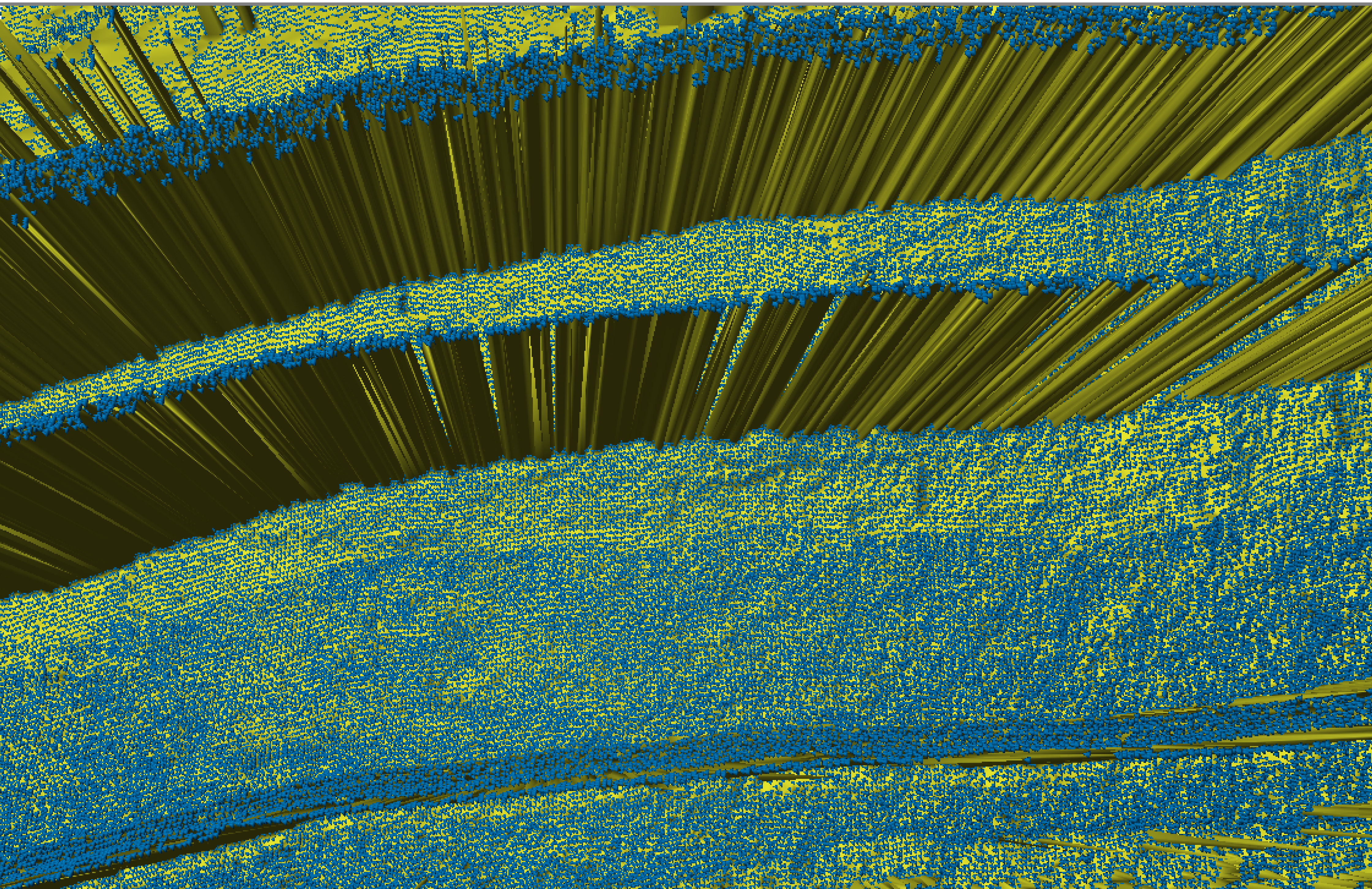
Real-world examples

Data: StatoilHydro



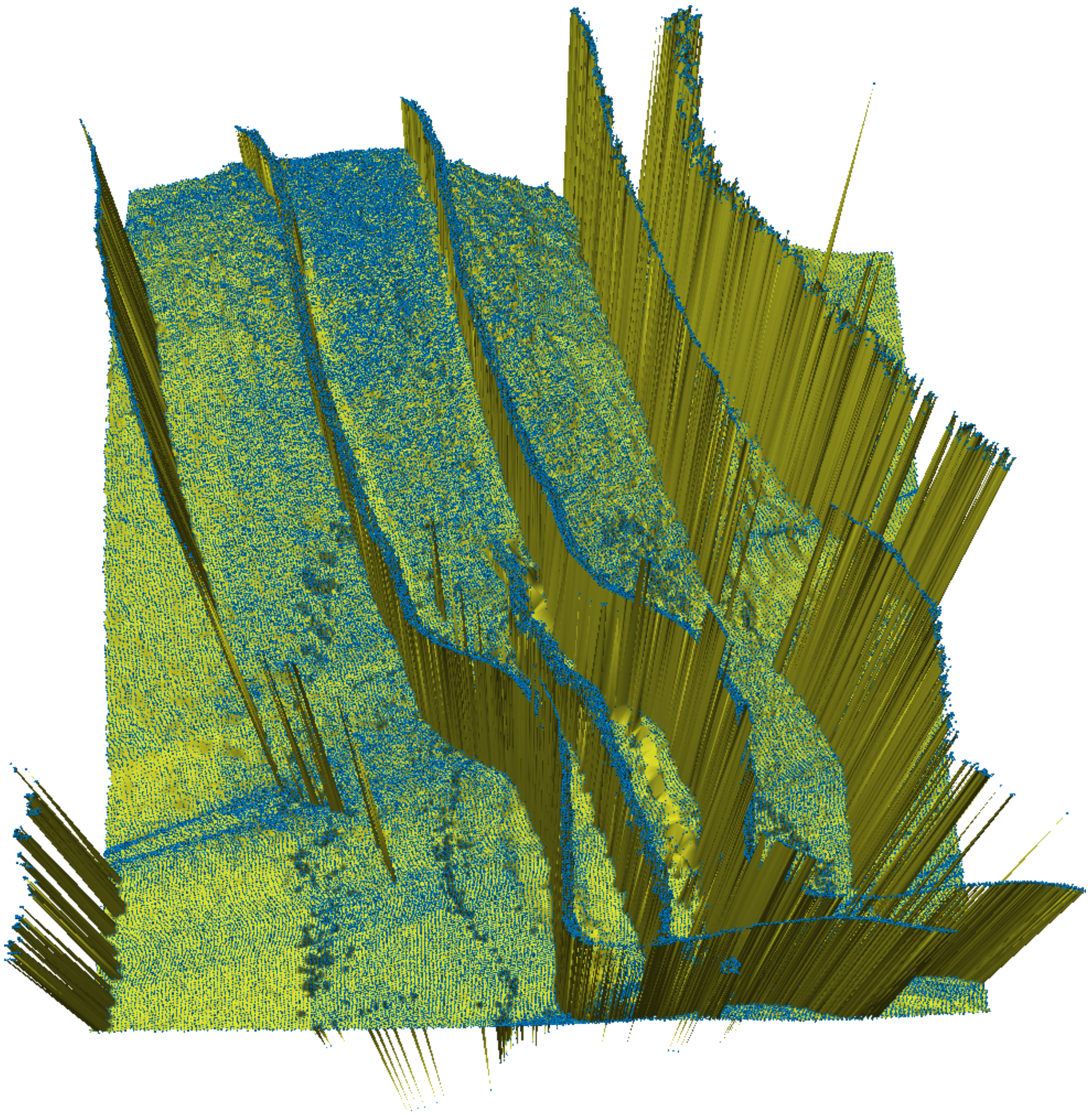
Real-world examples

Data: StatoilHydro



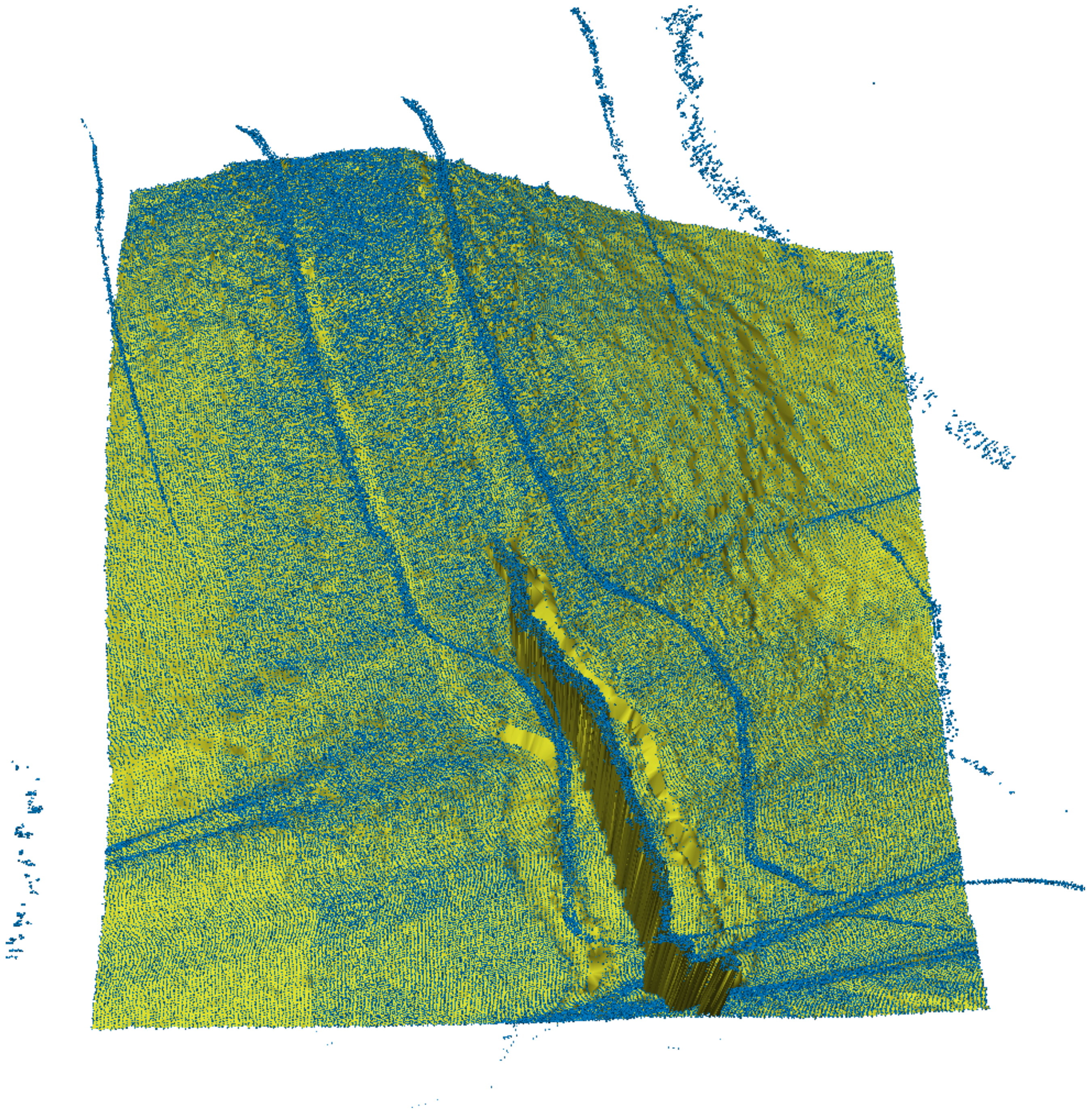
Real-world examples

Data: StatoilHydro



Real-world examples

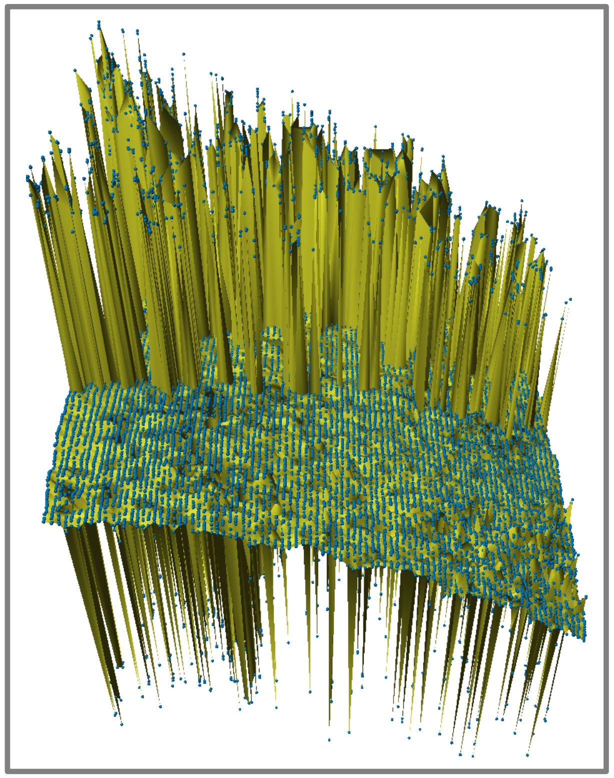
Data: StatoilHydro



Noise types

Data: StatoilHydro

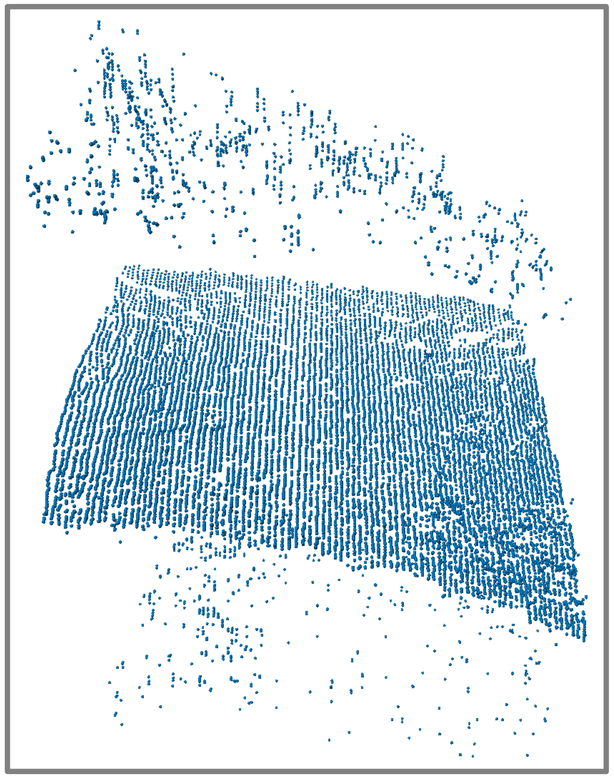
1. Random spikes, possibly clustered



Noise types

Data: StatoilHydro

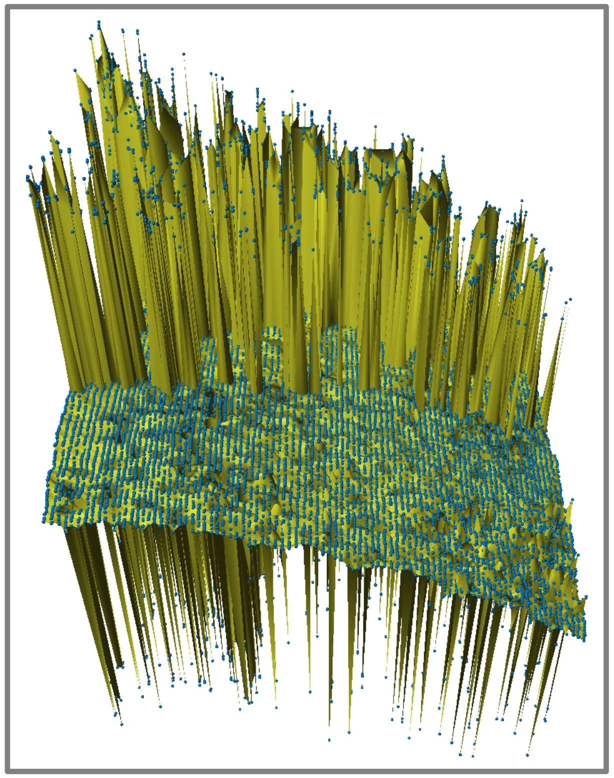
1. Random spikes, possibly clustered



Noise types

Data: StatoilHydro

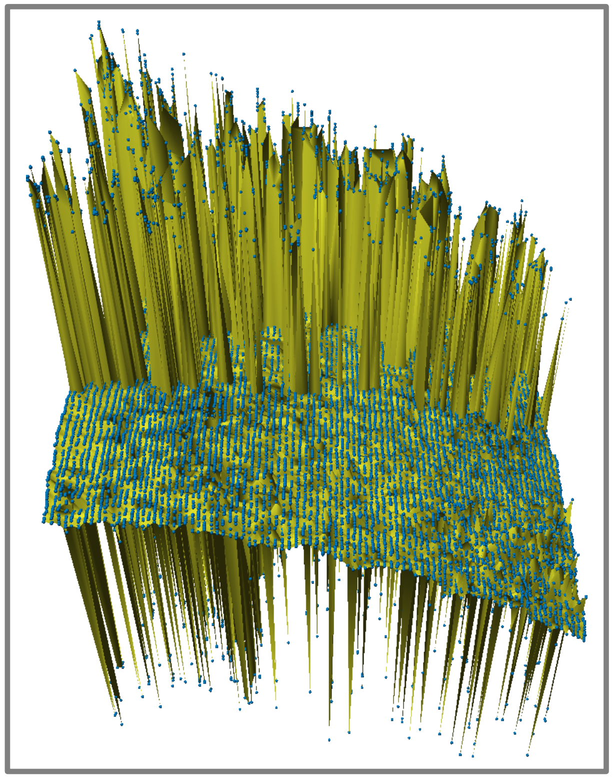
1. Random spikes, possibly clustered

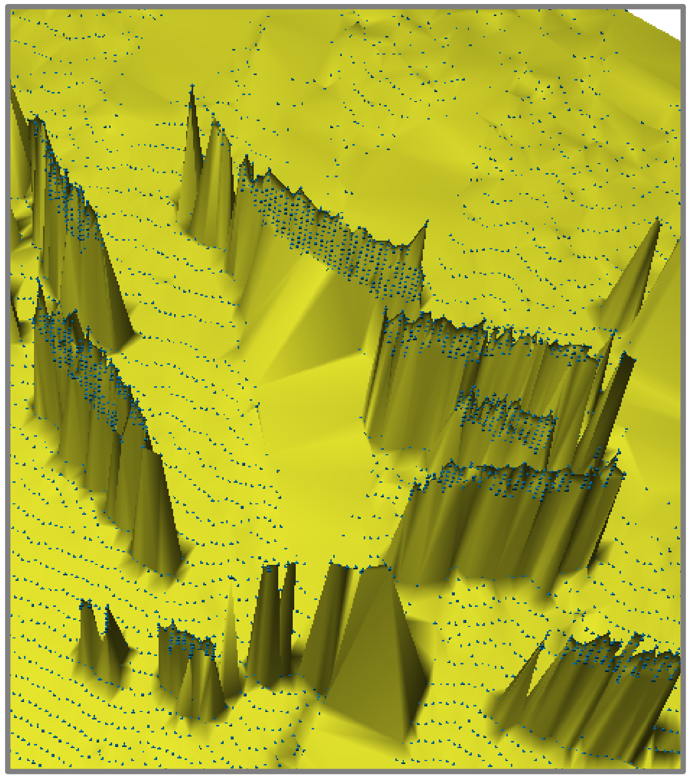


Noise types

Data: StatoilHydro

1. Random spikes, possibly clustered
2. Non-permanent physical objects (e.g. fish)

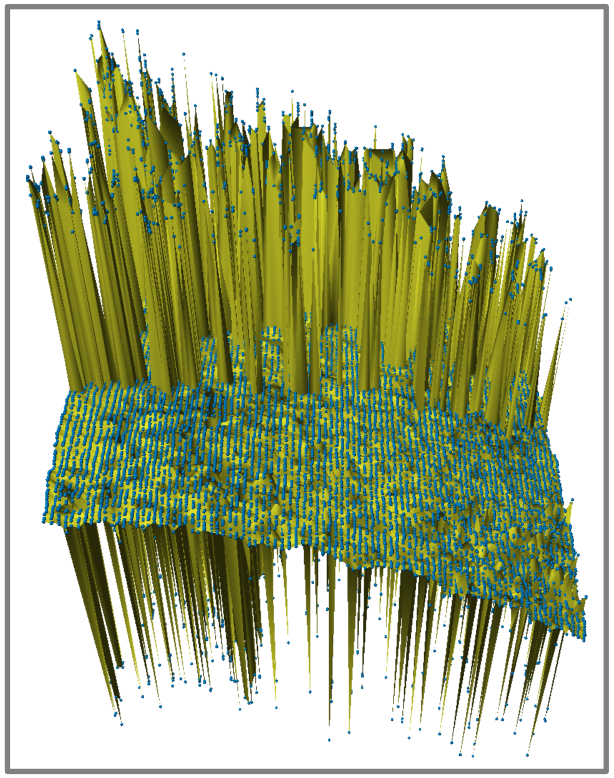


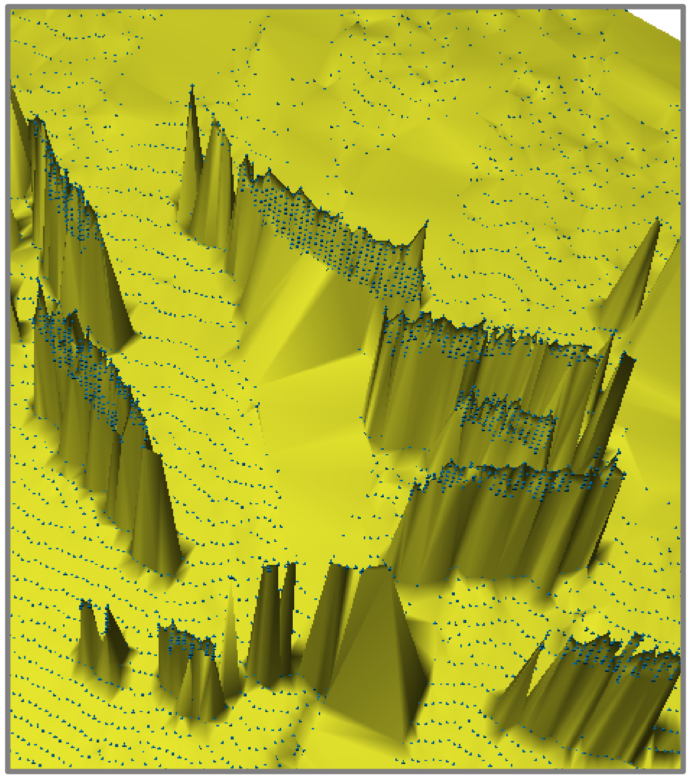


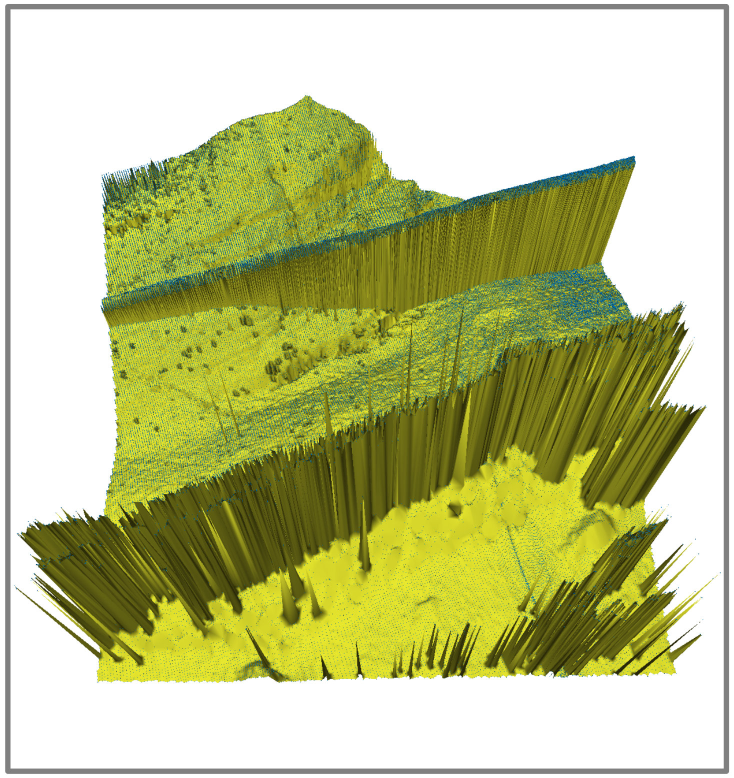
Noise types

Data: StatoilHydro

1. Random spikes, possibly clustered
2. Non-permanent physical objects (e.g. fish)
3. Structural noise







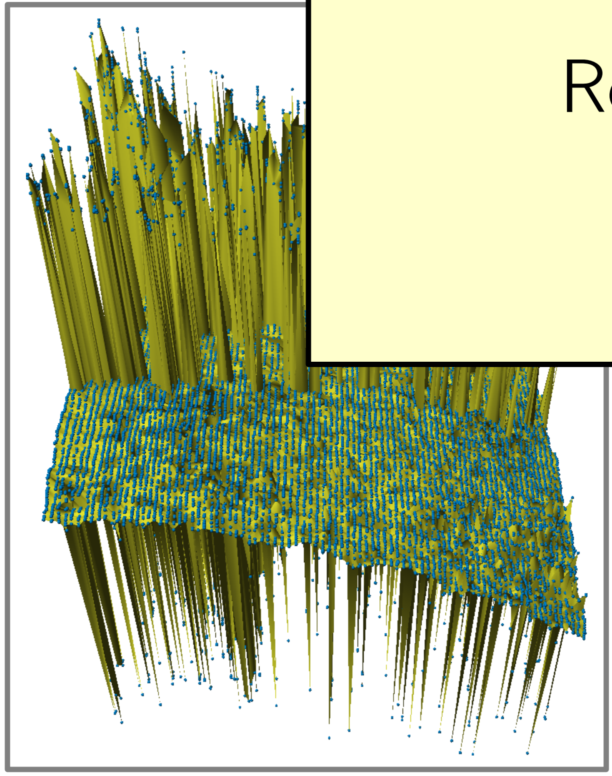
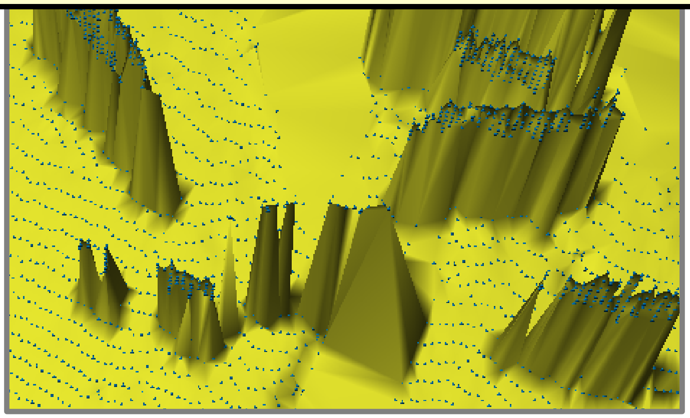
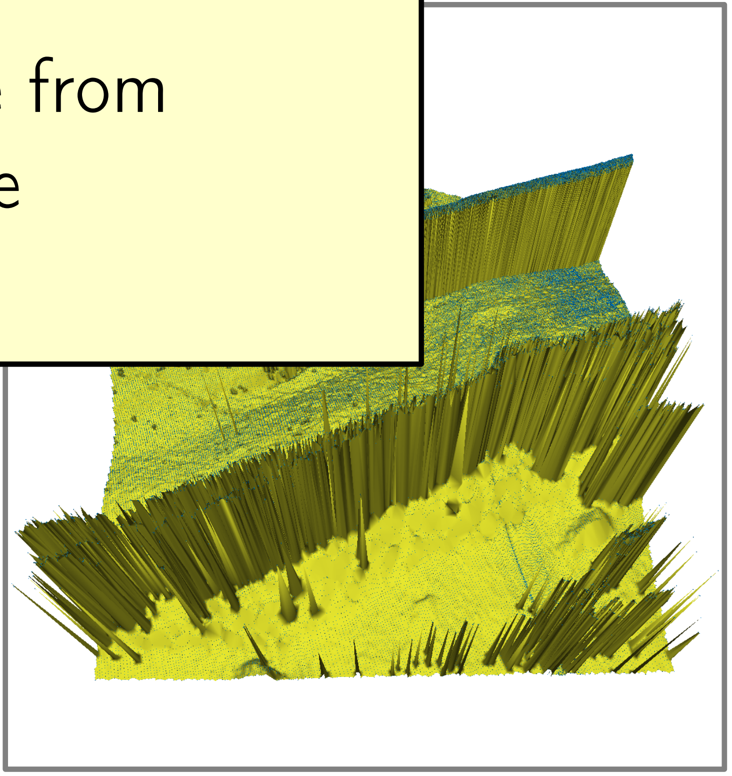
Noise types

Data: StatoilHydro

1. Random spikes, possibly clustered
2. Non-permanent physical objects (e.g. fish)
3. Structural noise

Problem

Remove these types of noise from massive point sets while keeping features intact

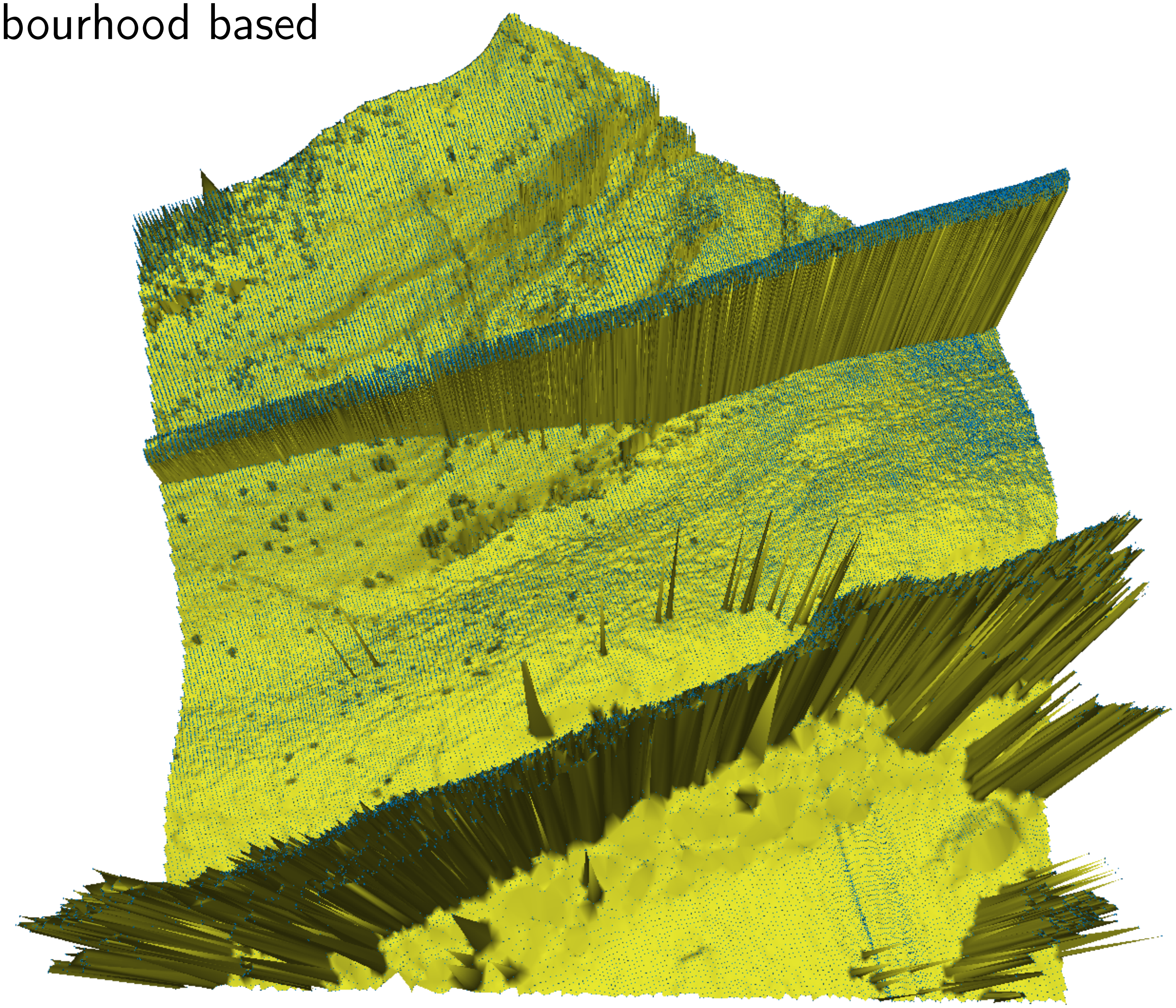




Previous work

Local-neighbourhood based

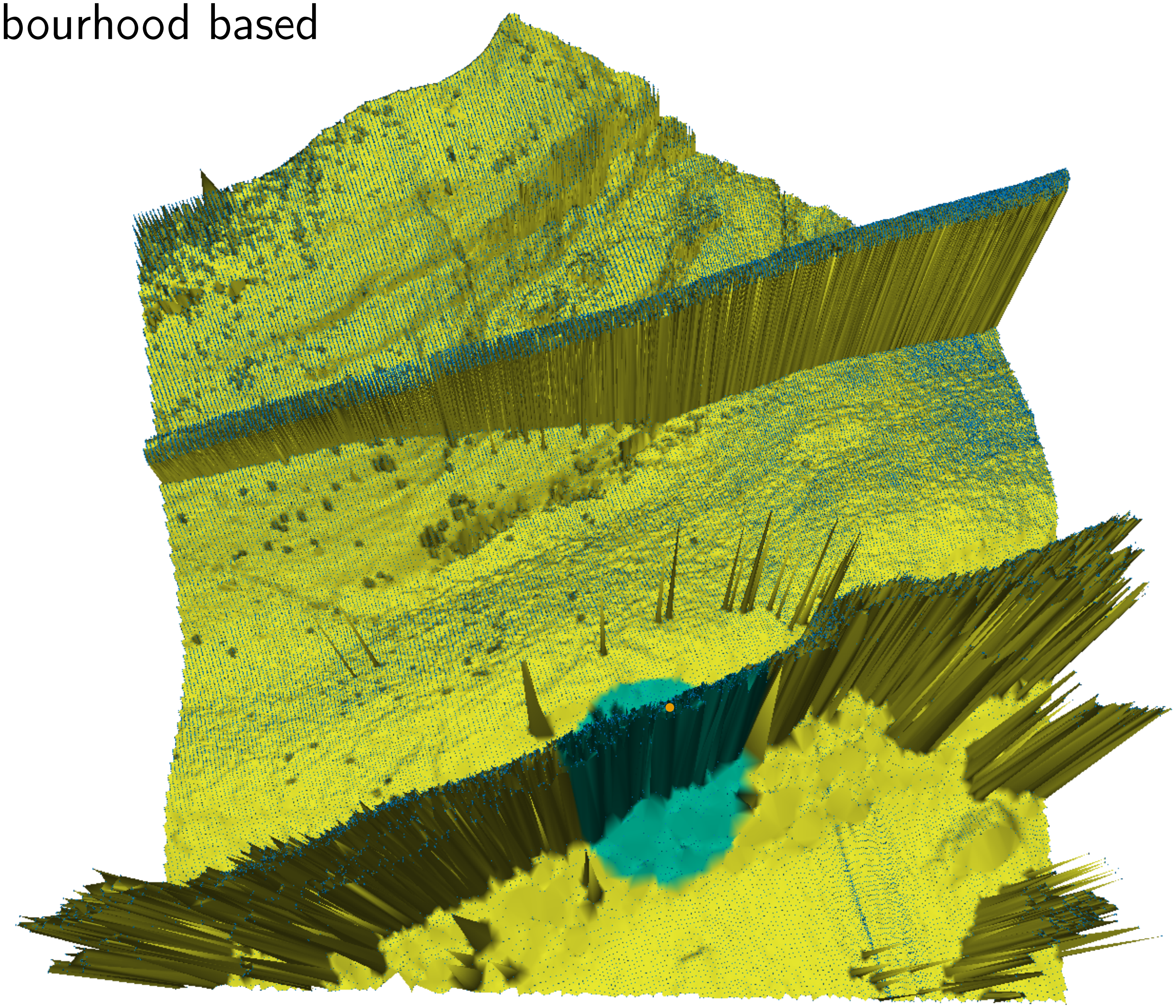
Previous work

Local-neighbourhood based



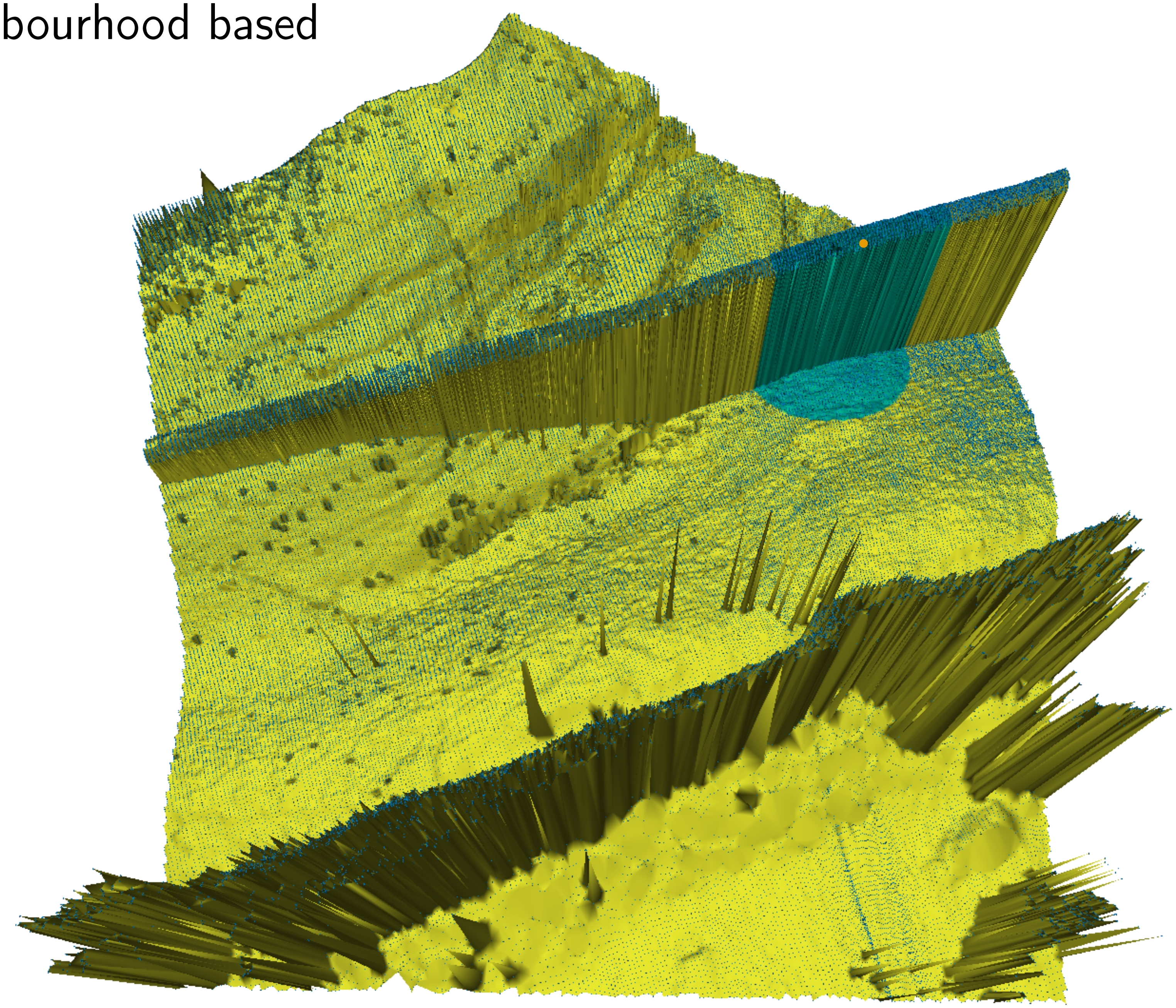
Previous work

Local-neighbourhood based



Previous work

Local-neighbourhood based

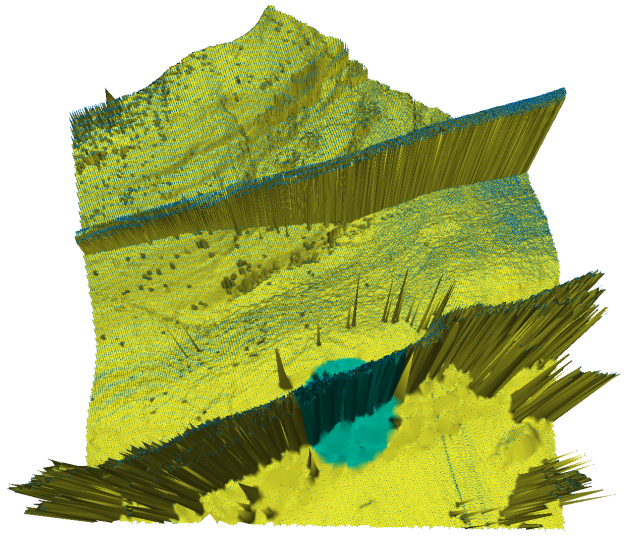


Previous work

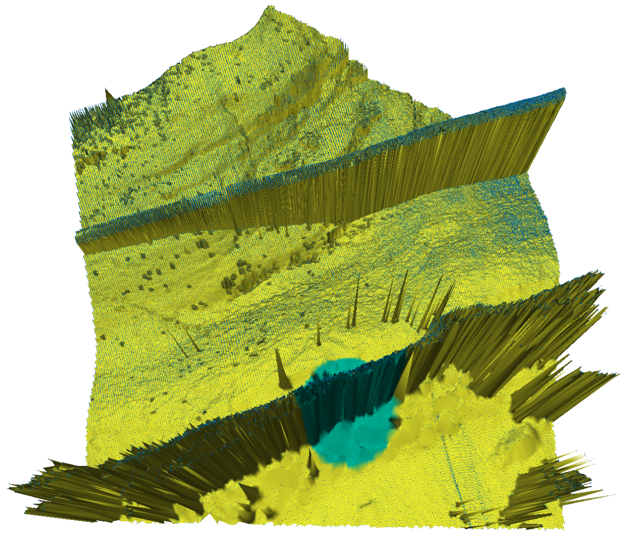
Local-neighbourhood based

E.g. CUBE [Calder & Mayer 2003], industry standard

- Place grid over points
- Estimate heights at grid nodes
  - Stastical analysis of points in neighbourhood
- Remove points far away from estimated surface



Previous work



Local-neighbourhood based

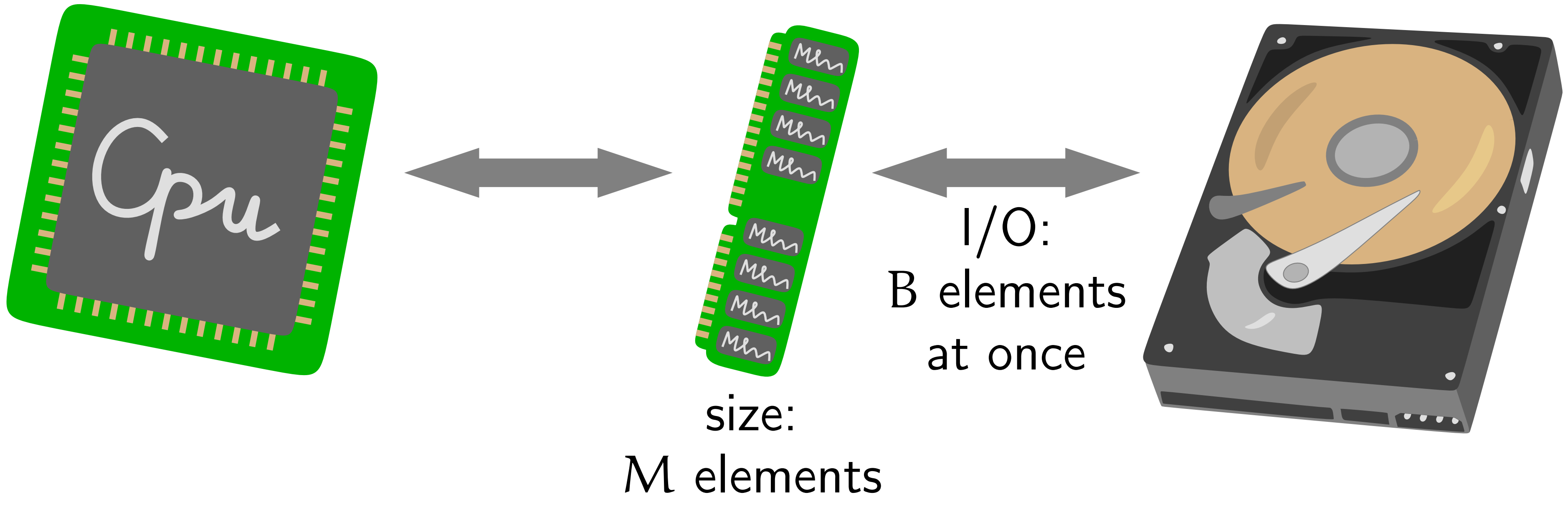
E.g. CUBE [Calder & Mayer 2003], industry standard

- Place grid over points
- Estimate heights at grid nodes
  - Stastical analysis of points in neighbourhood
- Remove points far away from estimated surface

$\longrightarrow$ Problems handling large clusters of noise and structural noise (types 2 and 3)

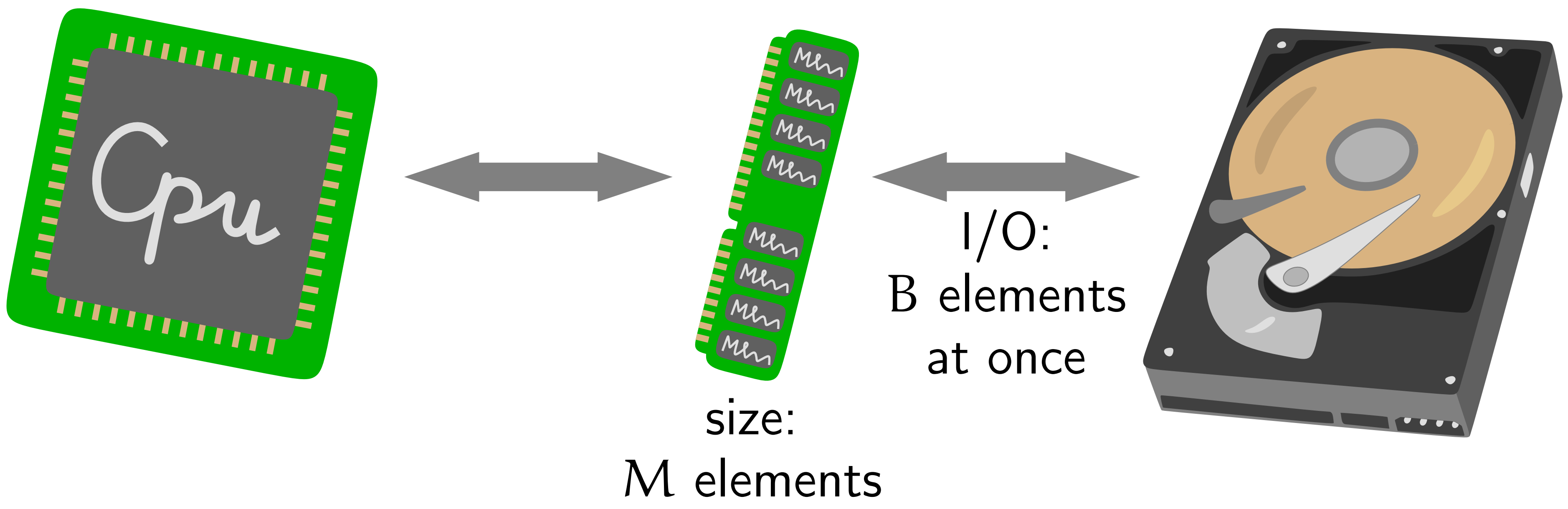
I/O-efficient algorithms

I/O model: analyze number of data transfers between internal and external memory



I/O-efficient algorithms

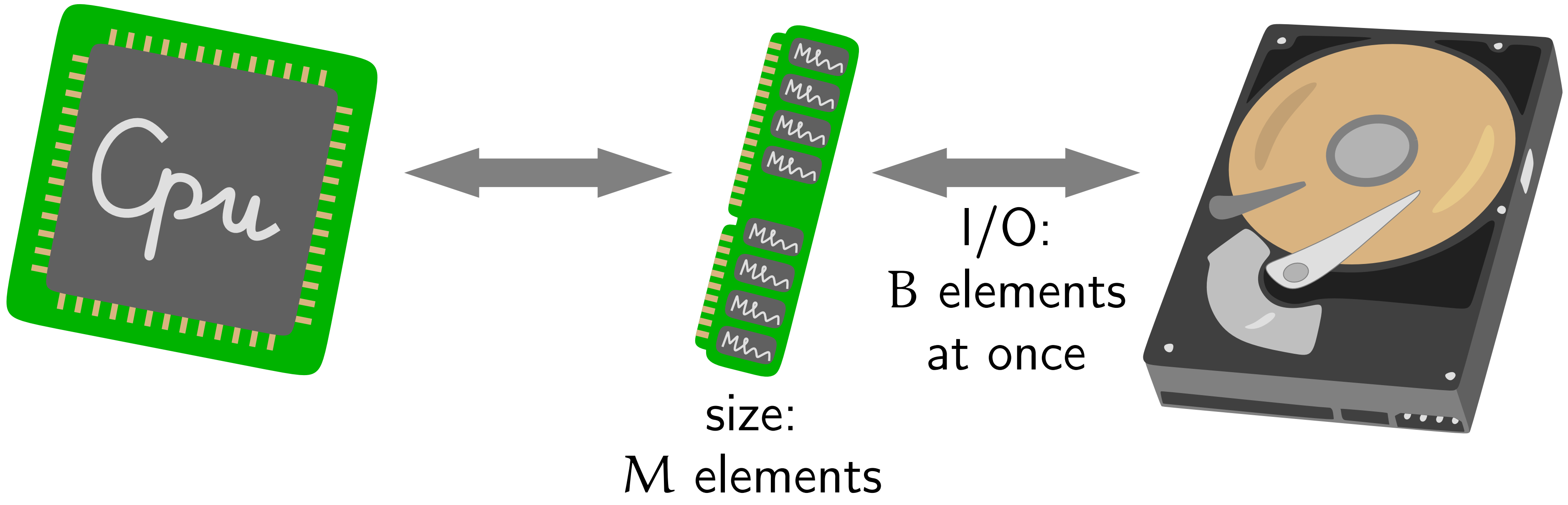
I/O model: analyze number of data transfers between internal and external memory



- Scanning N elements:
  $\Theta(\mathrm{scan}(N)) = \Theta(N/B)$ I/Os

I/O-efficient algorithms

I/O model: analyze number of data transfers between internal and external memory

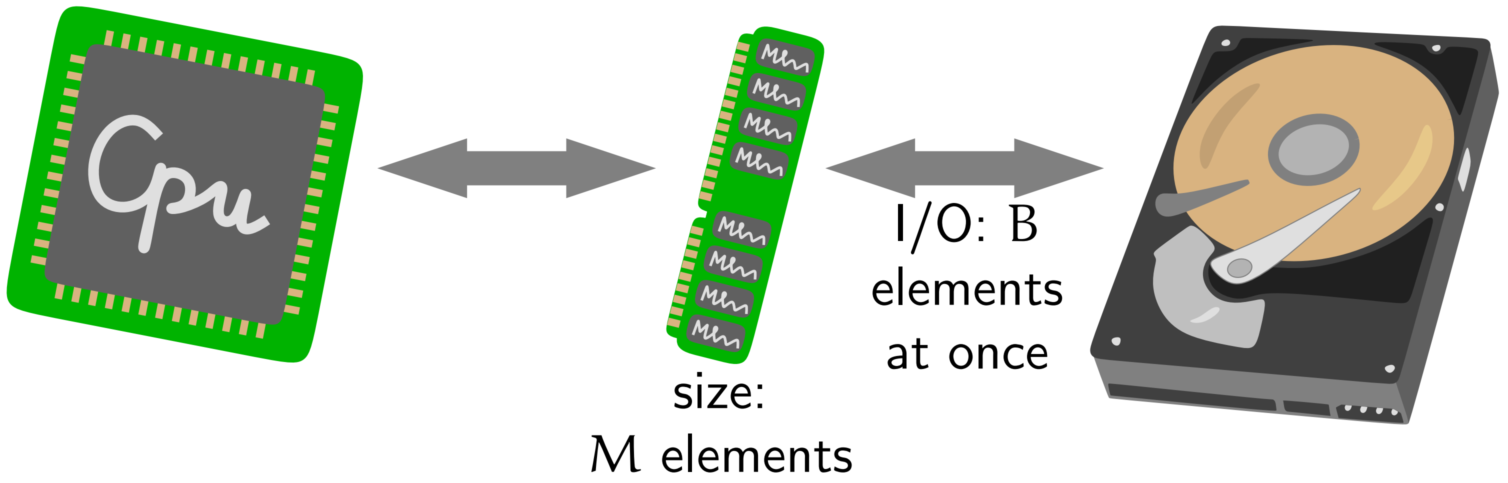


- Scanning N elements:
  $\Theta(\mathtt{scan}(N)) = \Theta(N/B)$ I/Os

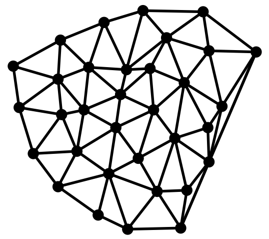
- Sorting N elements:
  $\Theta(\mathtt{sort}(N)) = \Theta(N/B \ \mathtt{log}_{M/B} N/B)$ I/Os

I/O-efficient algorithms

Previous work

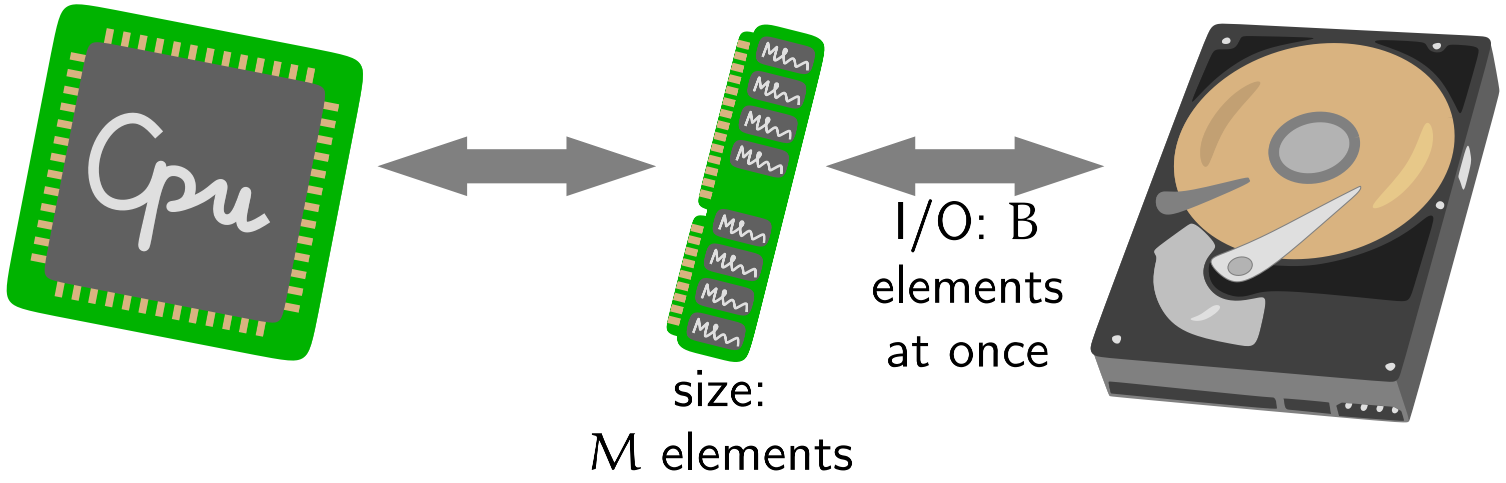


Delaunay triangulation for computing a TIN DEM

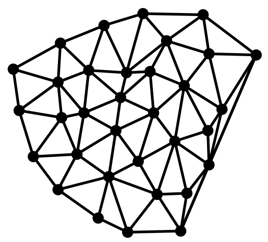


I/O-efficient algorithms

Previous work



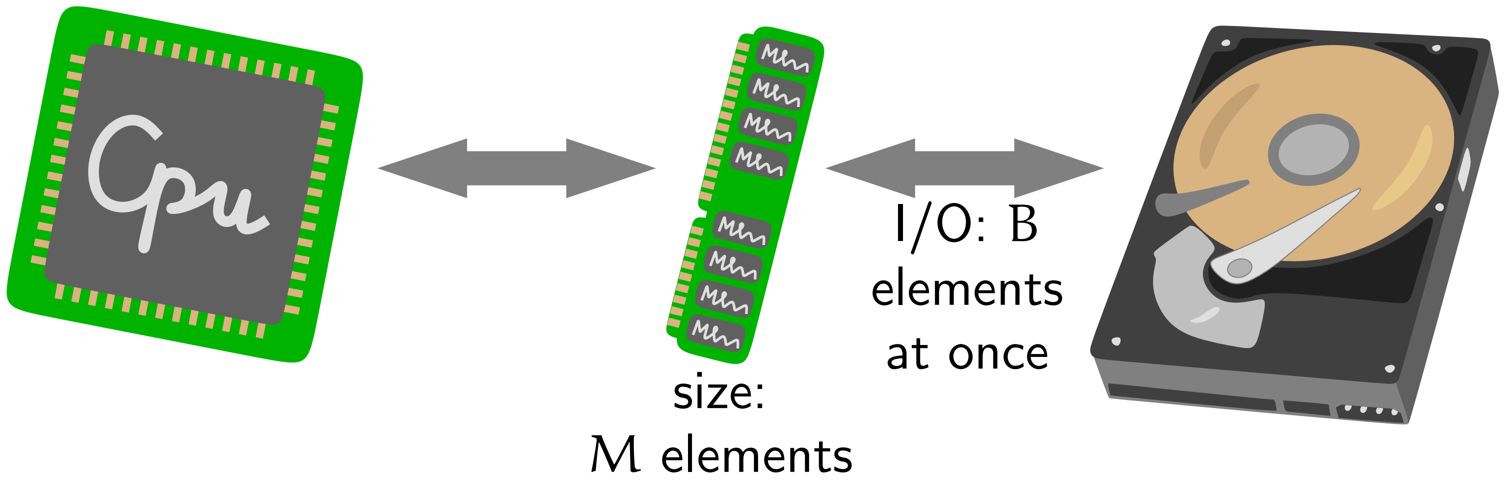
Delaunay triangulation for computing a TIN DEM



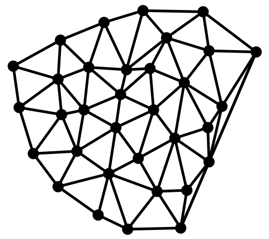
- $O(\mathtt{sort}(N))$ [Goodrich et al. 1993, Kumar & Ramos 2002]
- Practical $O(\mathtt{sort}(N))$ [Agarwal et al. 2005]

I/O-efficient algorithms

Previous work

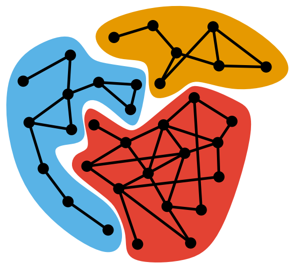


Delaunay triangulation for computing a TIN DEM



- $O(\mathtt{sort}(N))$ [Goodrich et al. 1993, Kumar & Ramos 2002]
- Practical $O(\mathtt{sort}(N))$ [Agarwal et al. 2005]

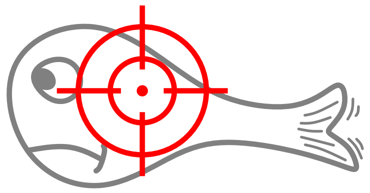
Connected components



- $O(\mathtt{sort}(|E|)\,\mathtt{log}_2\mathtt{log}_2(B\frac{|V|}{|E|}))$ [Munagala and Ranade 1999]
- Practical $O(\mathtt{sort}(N)\,\mathtt{log}_2(N/M))$ [Agarwal et al. 2006] (batched union–find)

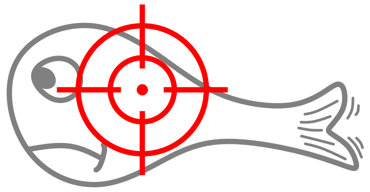
Our results

- Cleaning algorithm for MBES data
  - Identifies both random, local and structural noise
  - Theoretically I/O-efficient
  - Practically efficient and implementable

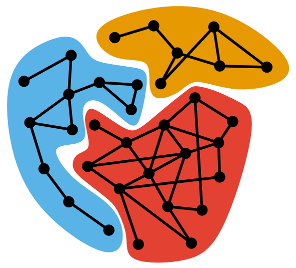


Our results

- Cleaning algorithm for MBES data
  - Identifies both random, local and structural noise
  - Theoretically I/O-efficient
  - Practically efficient and implementable

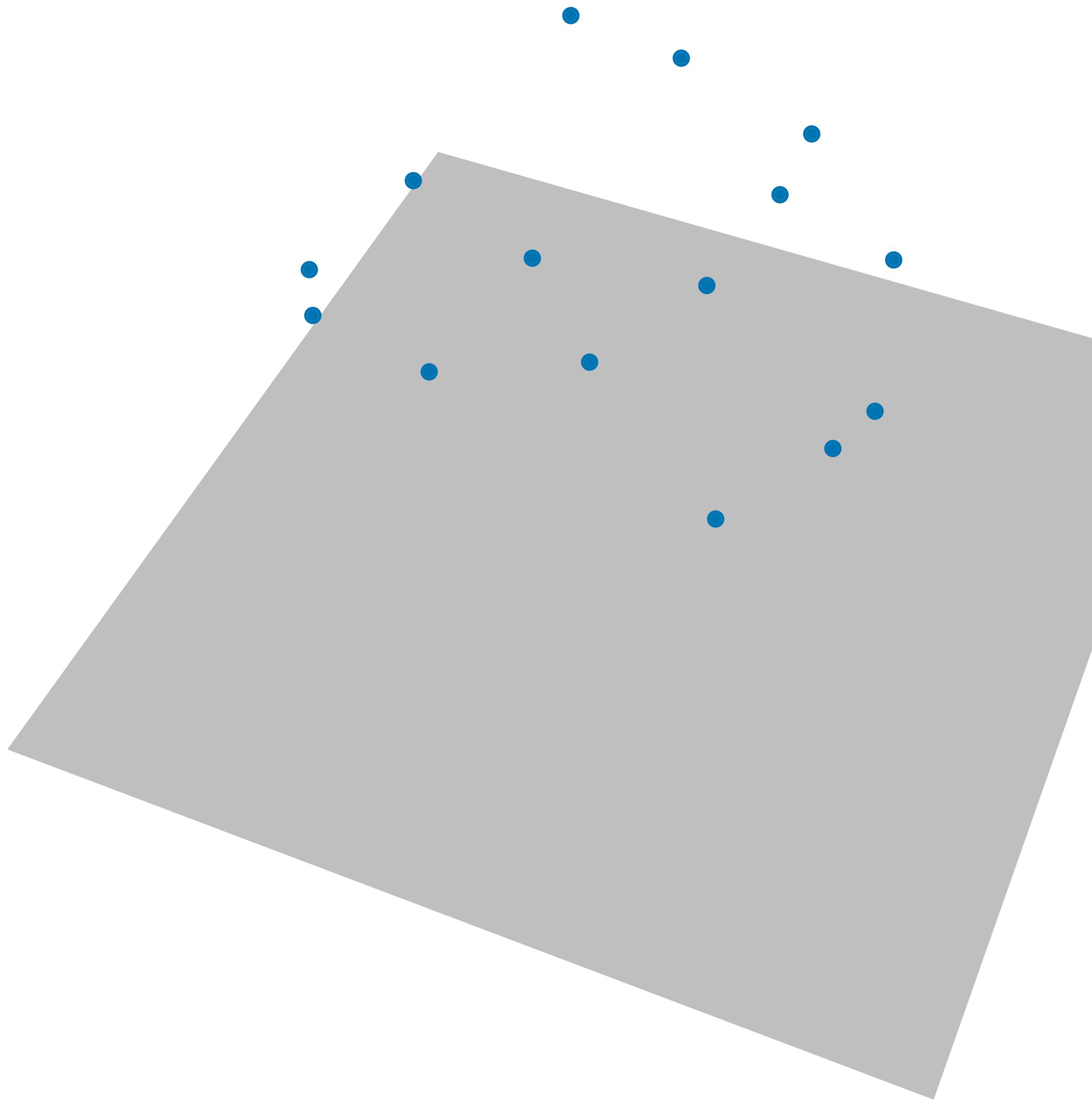


- Connected component algorithm
  - $O(\mathtt{sort}(N))$ I/Os under a natural assumption
  - Practically efficient and implementable



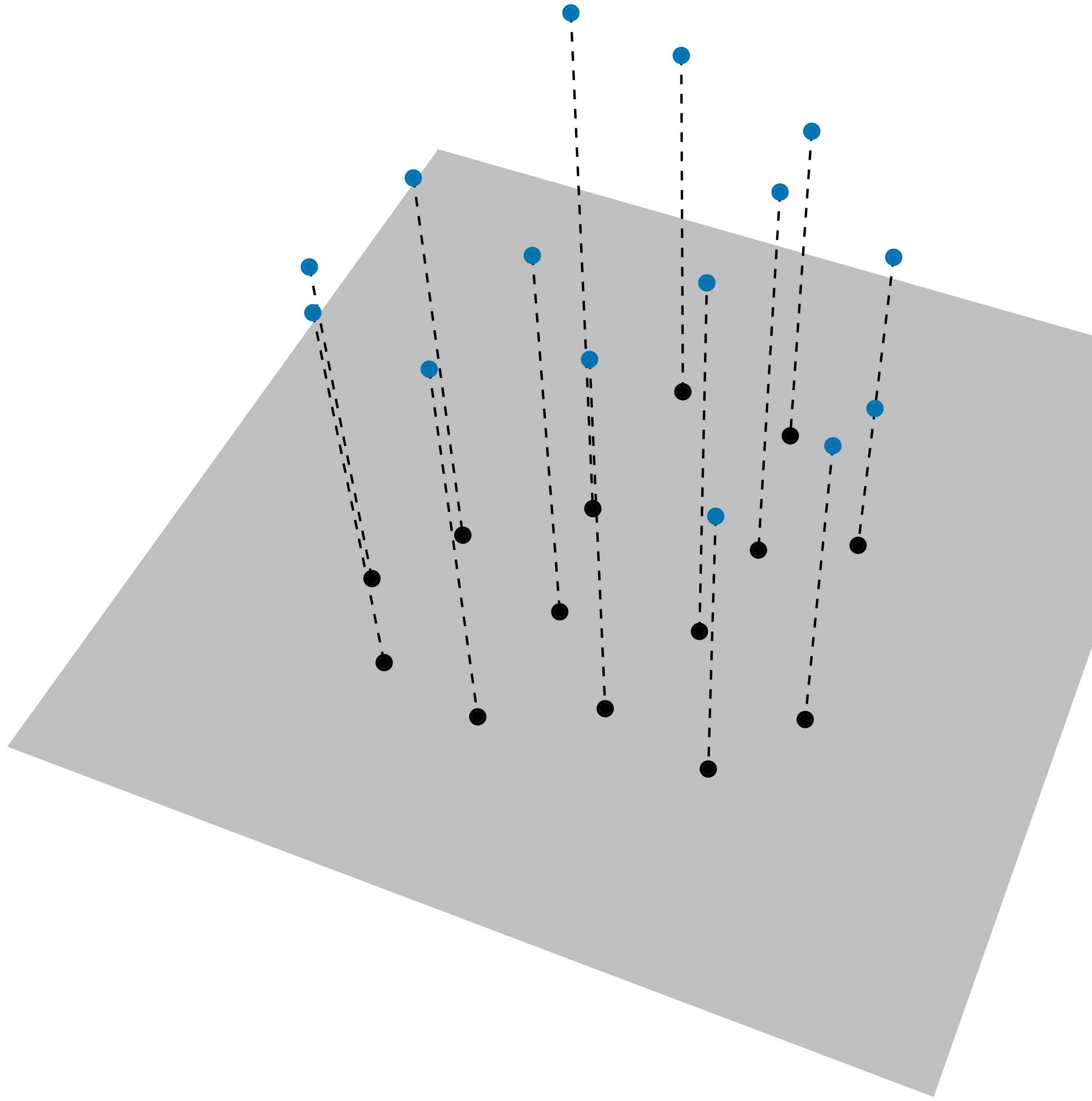
Our cleaning algorithm

1. Perturb "xy-duplicate" points



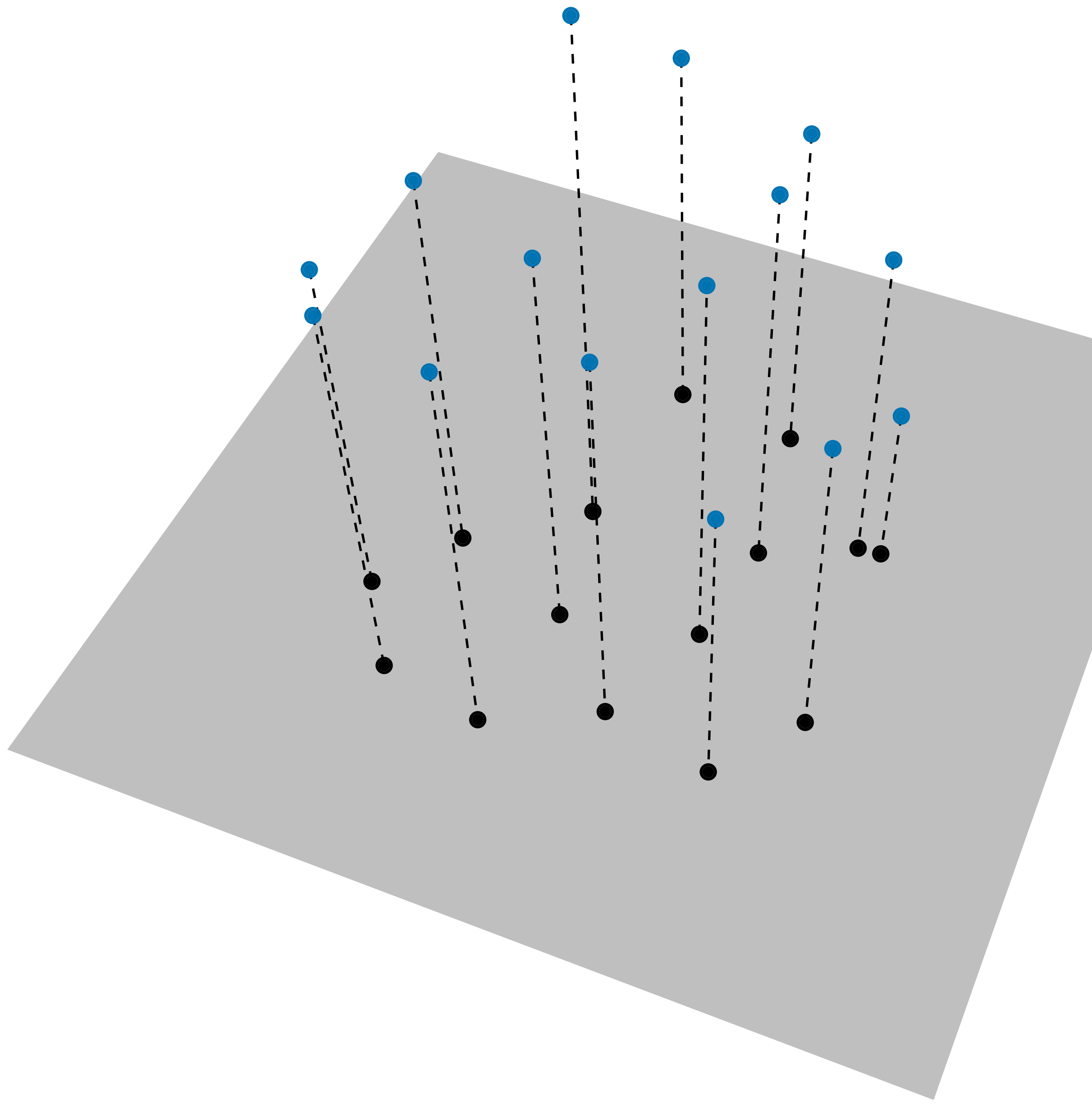
Our cleaning algorithm

1. Perturb "xy-duplicate" points



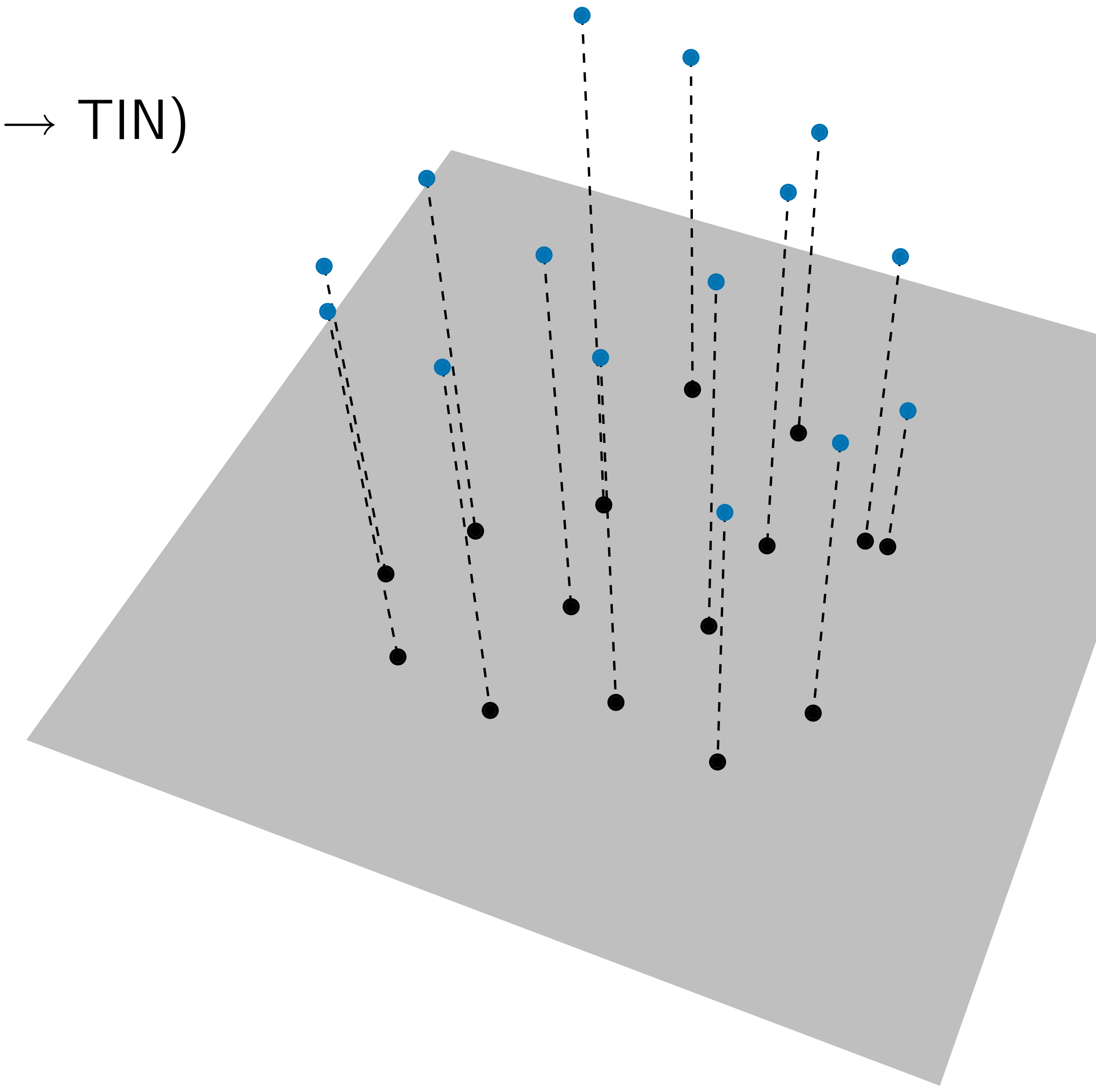
Our cleaning algorithm

1. Perturb "xy-duplicate" points



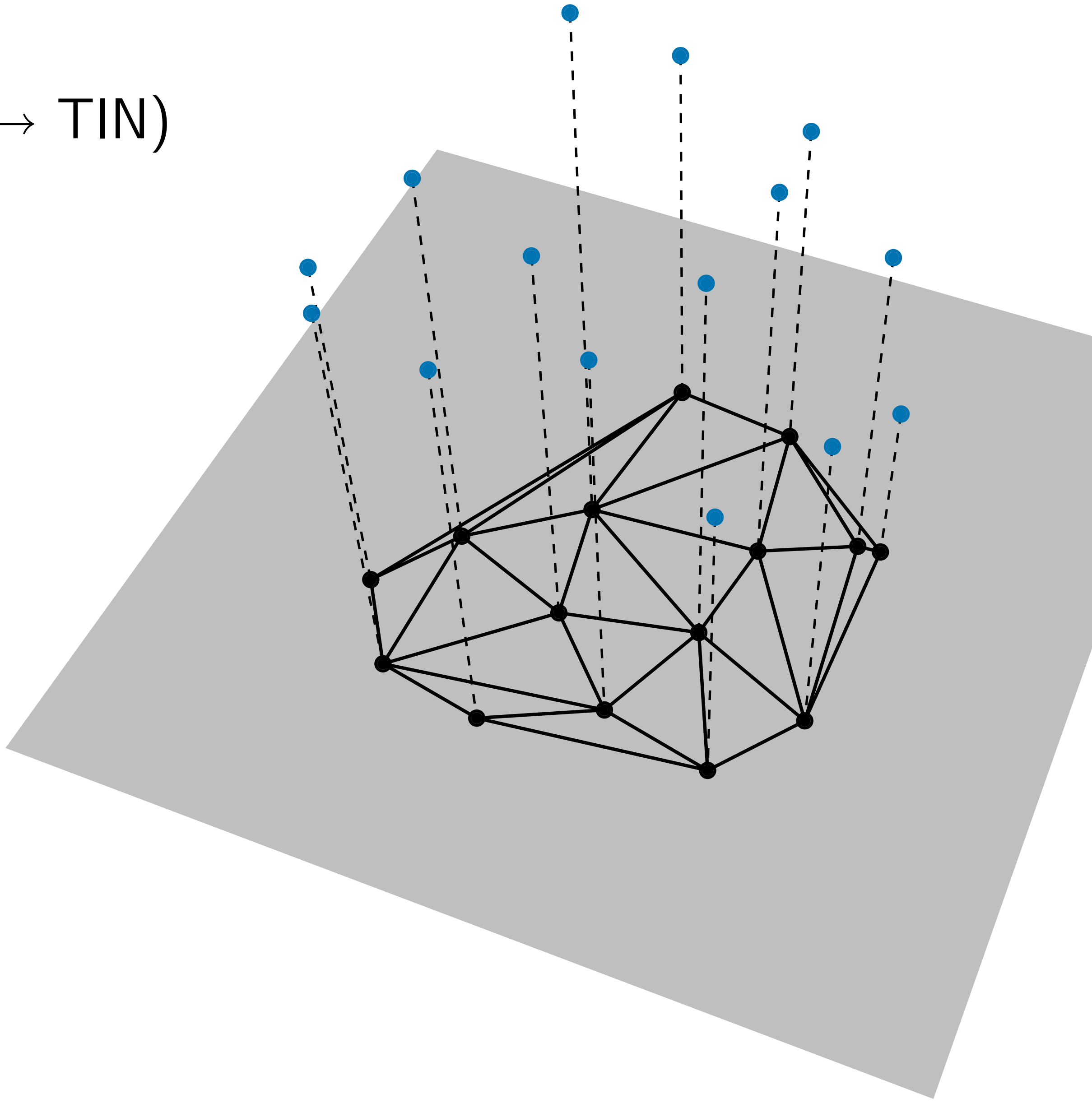
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)



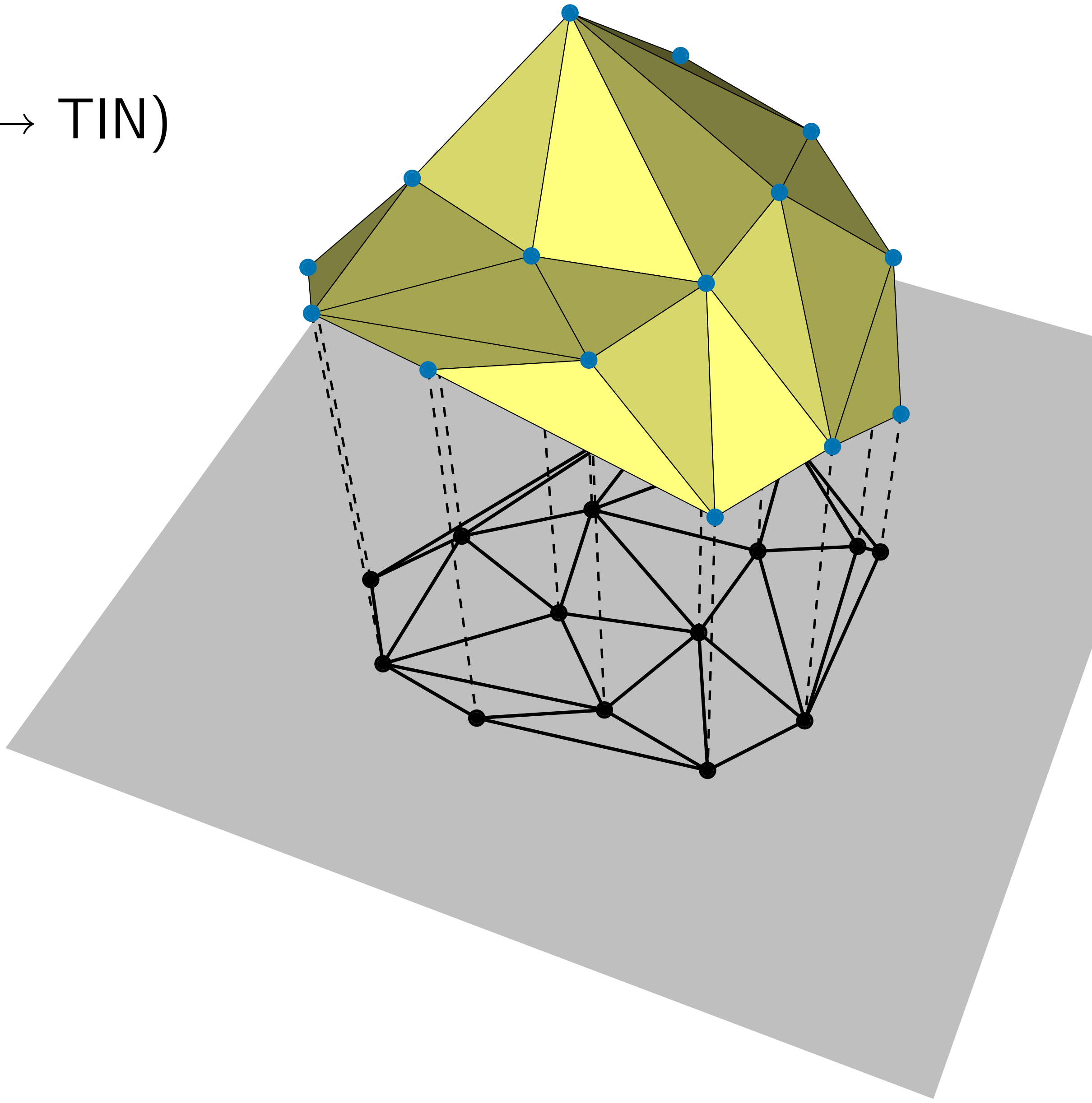
Our cleaning algorithm

1. Perturb “xy-duplicate” points
2. Create Delaunay triangulation ($\rightarrow$ TIN)



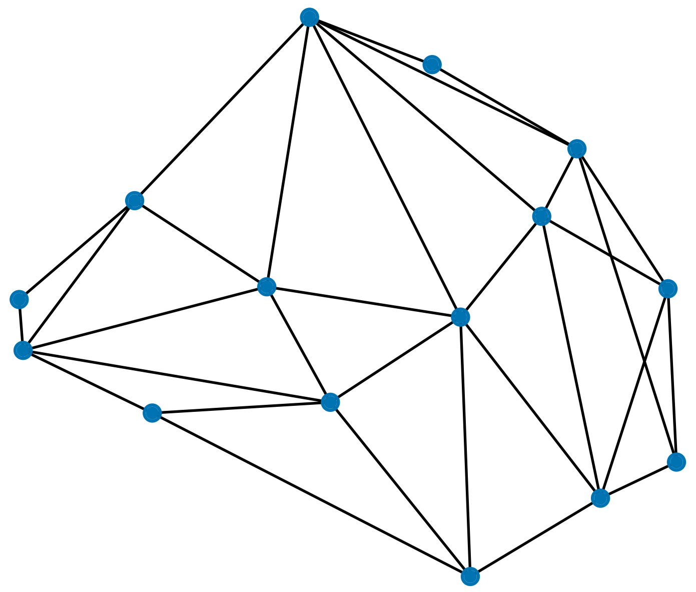
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)



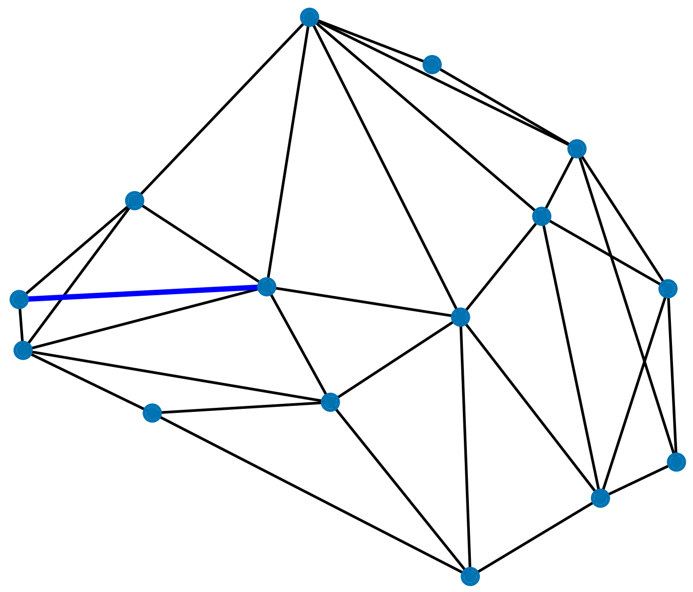
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*



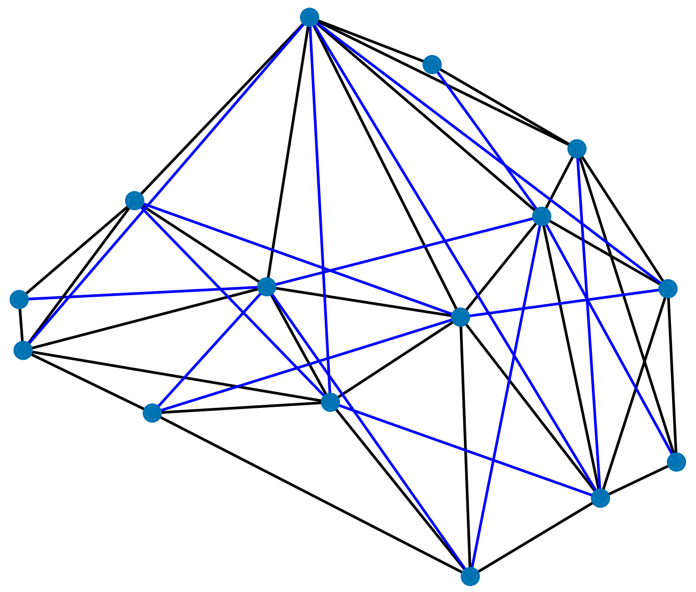
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*



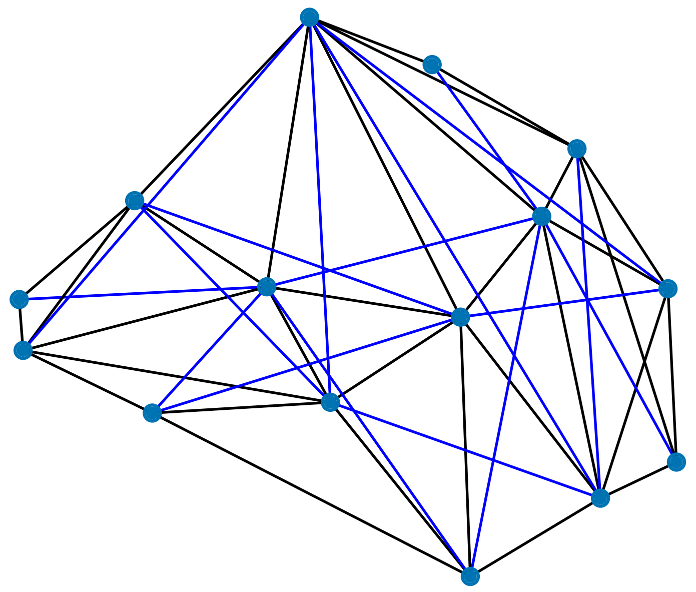
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*



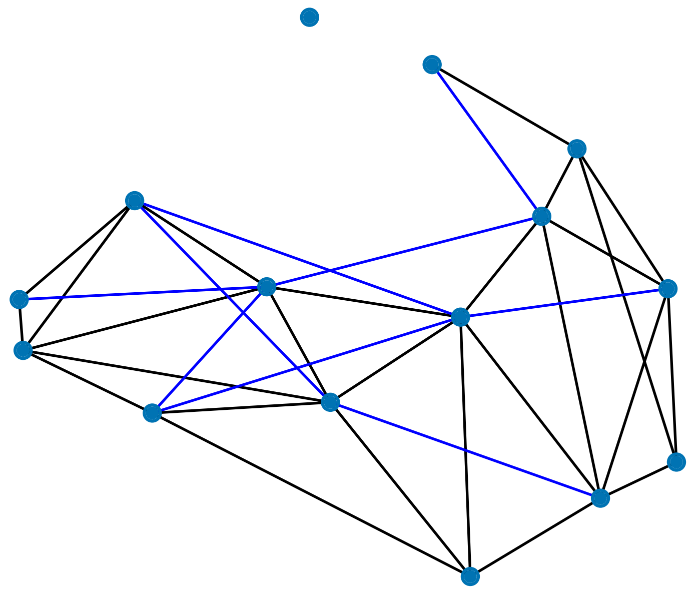
Our cleaning algorithm

1. Perturb “xy-duplicate” points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*
4. Remove edges with z-difference $>$ threshold



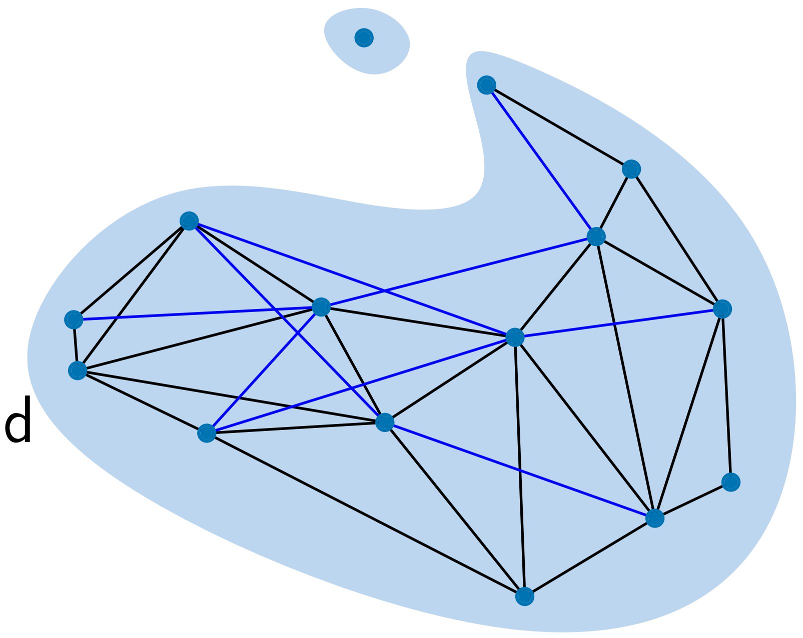
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*
4. Remove edges with z-difference $>$ threshold



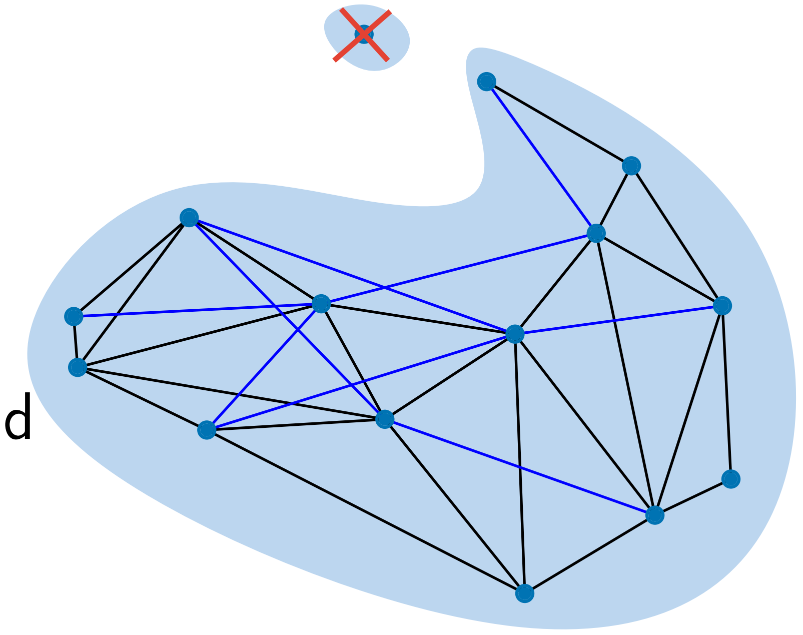
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*
4. Remove edges with z-difference $>$ threshold
5. Find largest connected component



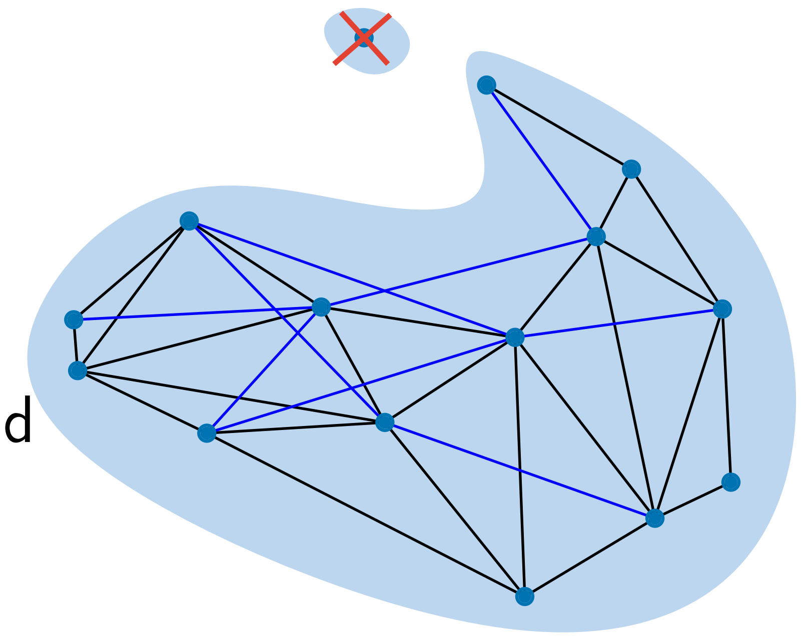
Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*
4. Remove edges with z-difference $>$ threshold
5. Find largest connected component
6. Remove all points not in largest component



Our cleaning algorithm

1. Perturb "xy-duplicate" points
2. Create Delaunay triangulation ($\rightarrow$ TIN)
3. Add *diagonals*
4. Remove edges with z-difference $>$ threshold
5. Find largest connected component
6. Remove all points not in largest component

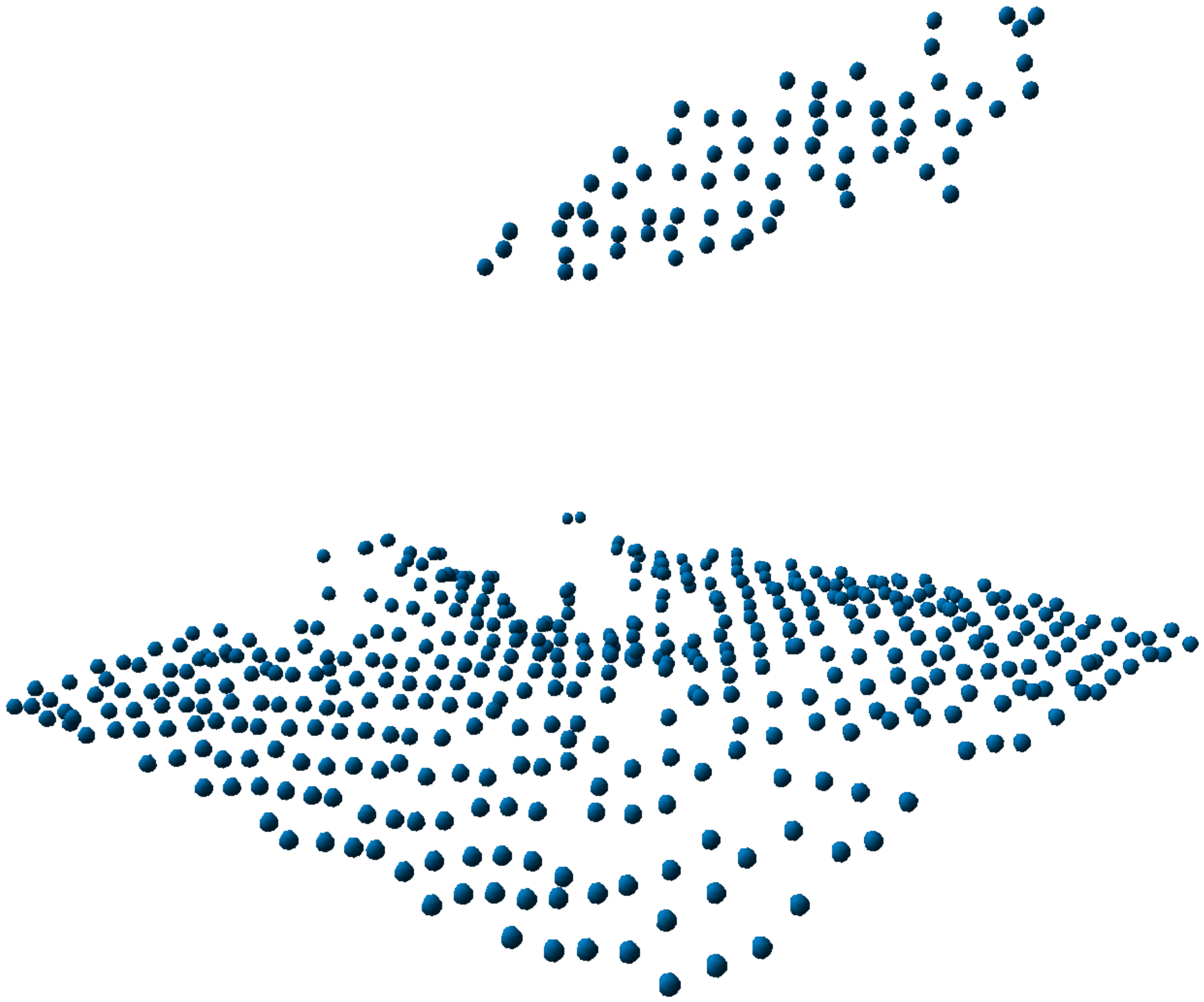


$O(\texttt{sort}(N))$ I/Os + connected components = $O(\texttt{sort}(N) \log\log B)$ I/Os, or $O(\texttt{sort}(N))$ I/Os under a practical assumption

Why it works

fish

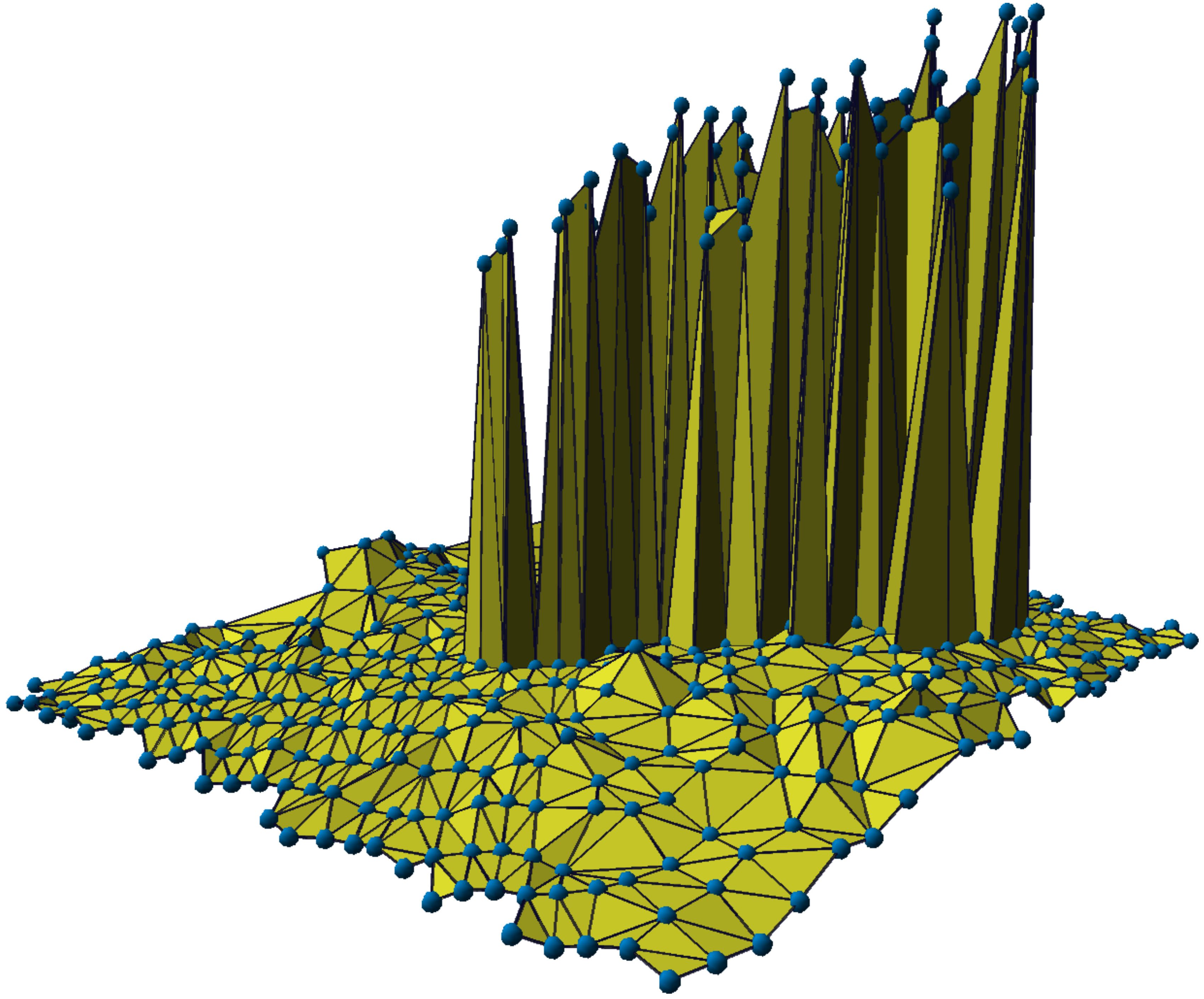
Data: StatoilHydro



Why it works

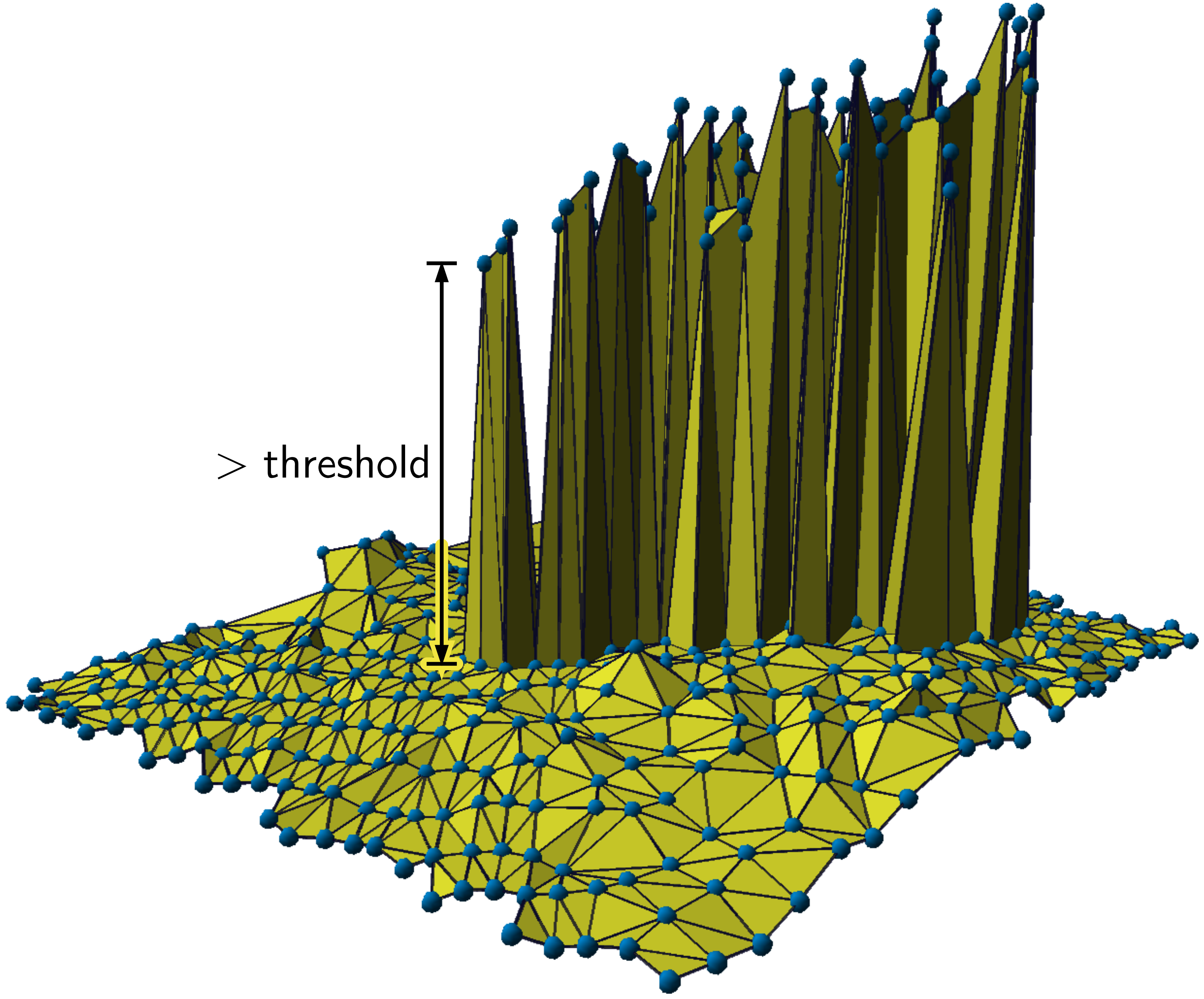
fish

Data: StatoilHydro



Why it works
fish

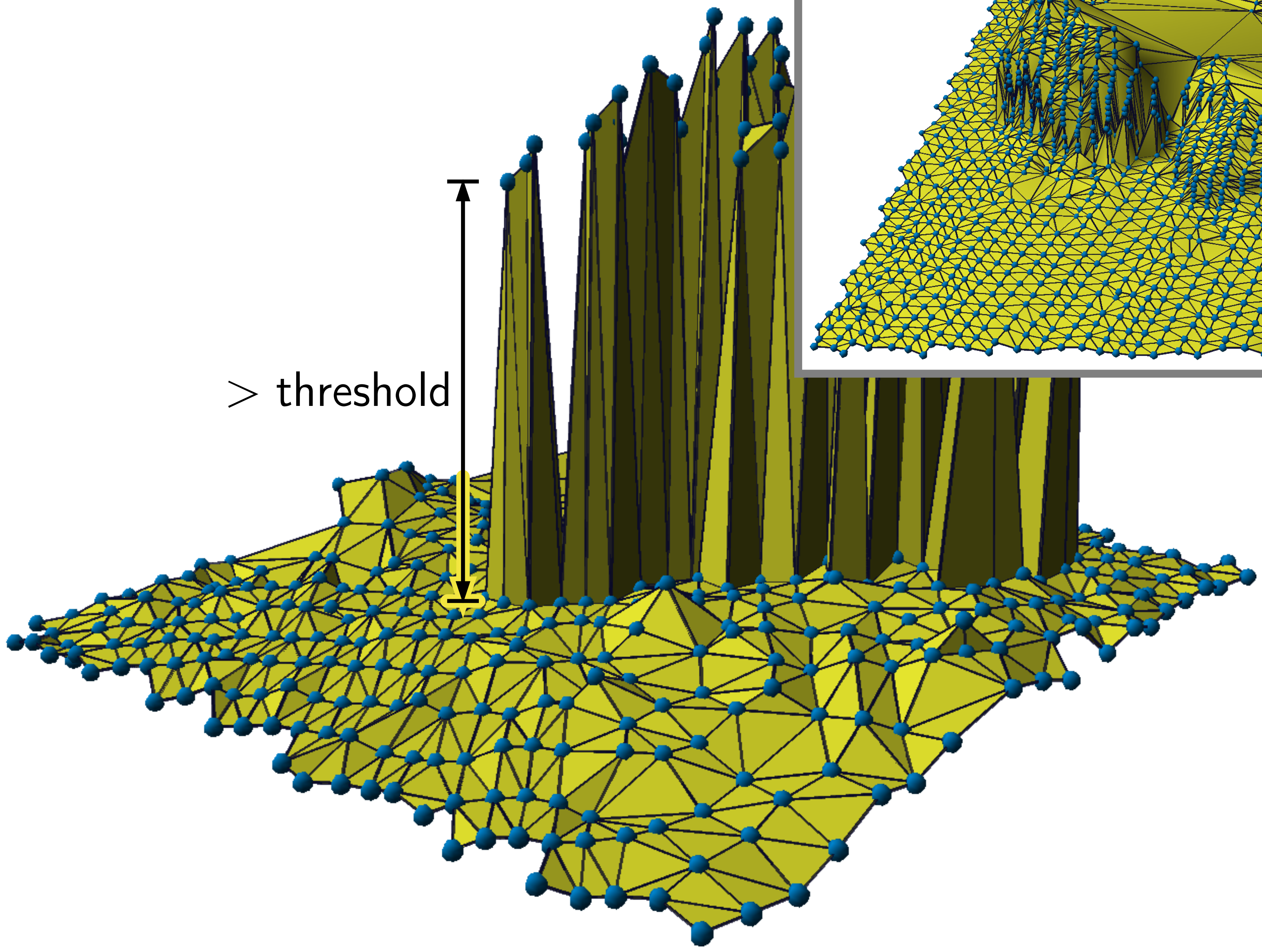
Data: StatoilHydro

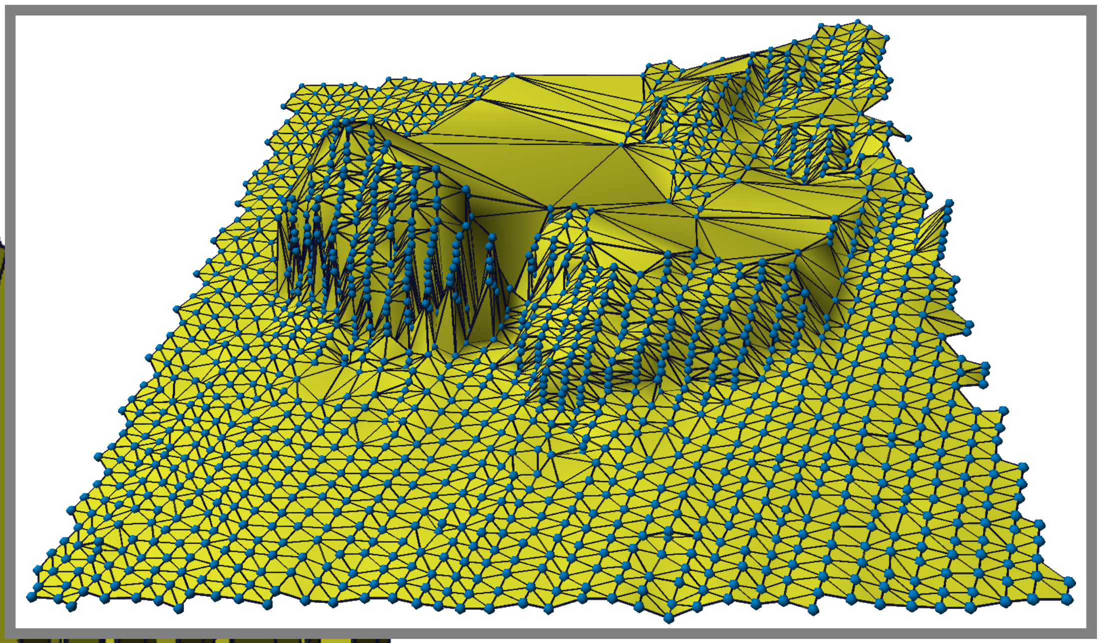


Why it works

fish

Data: StatoilHydro

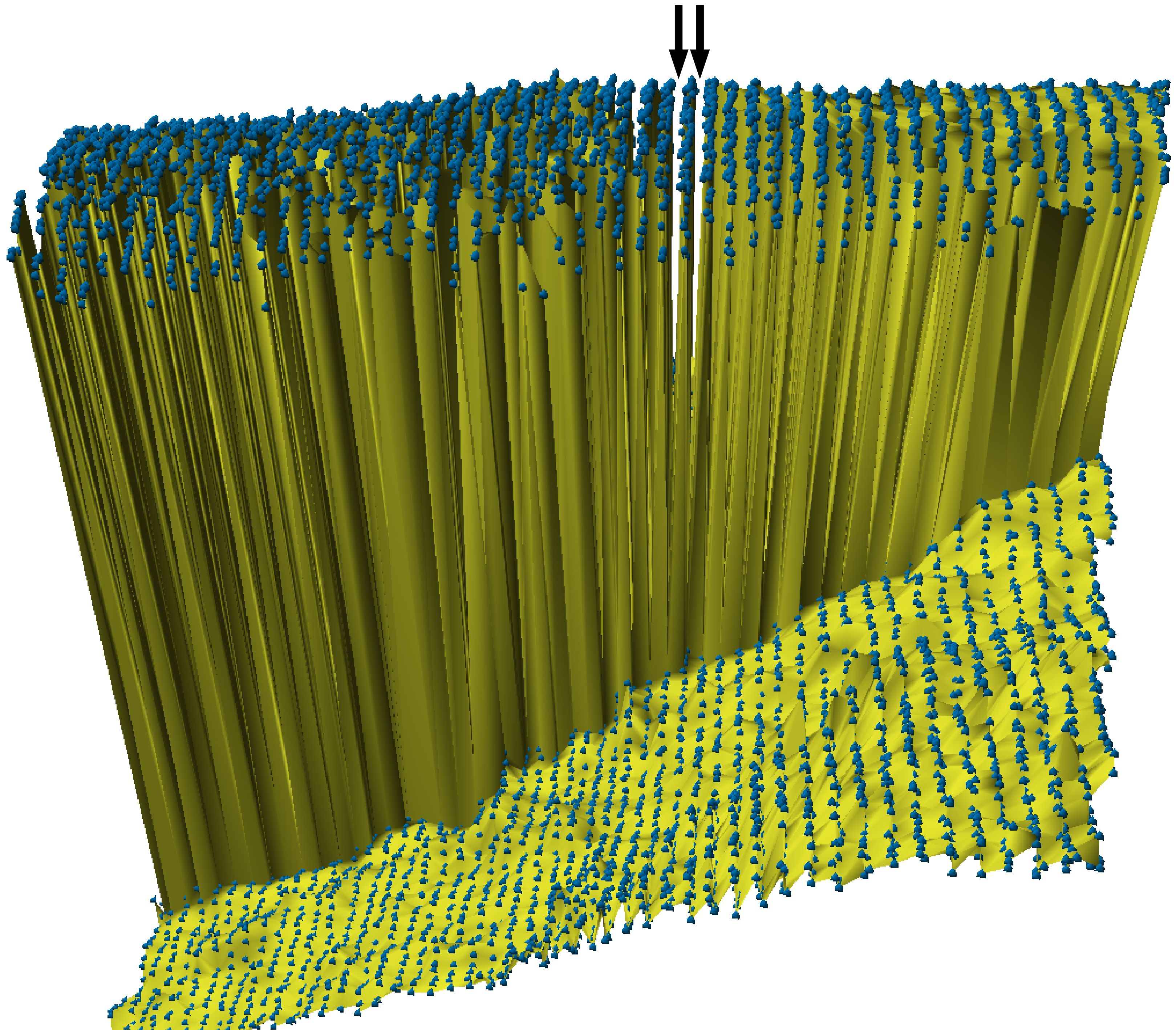




Why it works

no diagonals $\Rightarrow$ pipeline disconnected

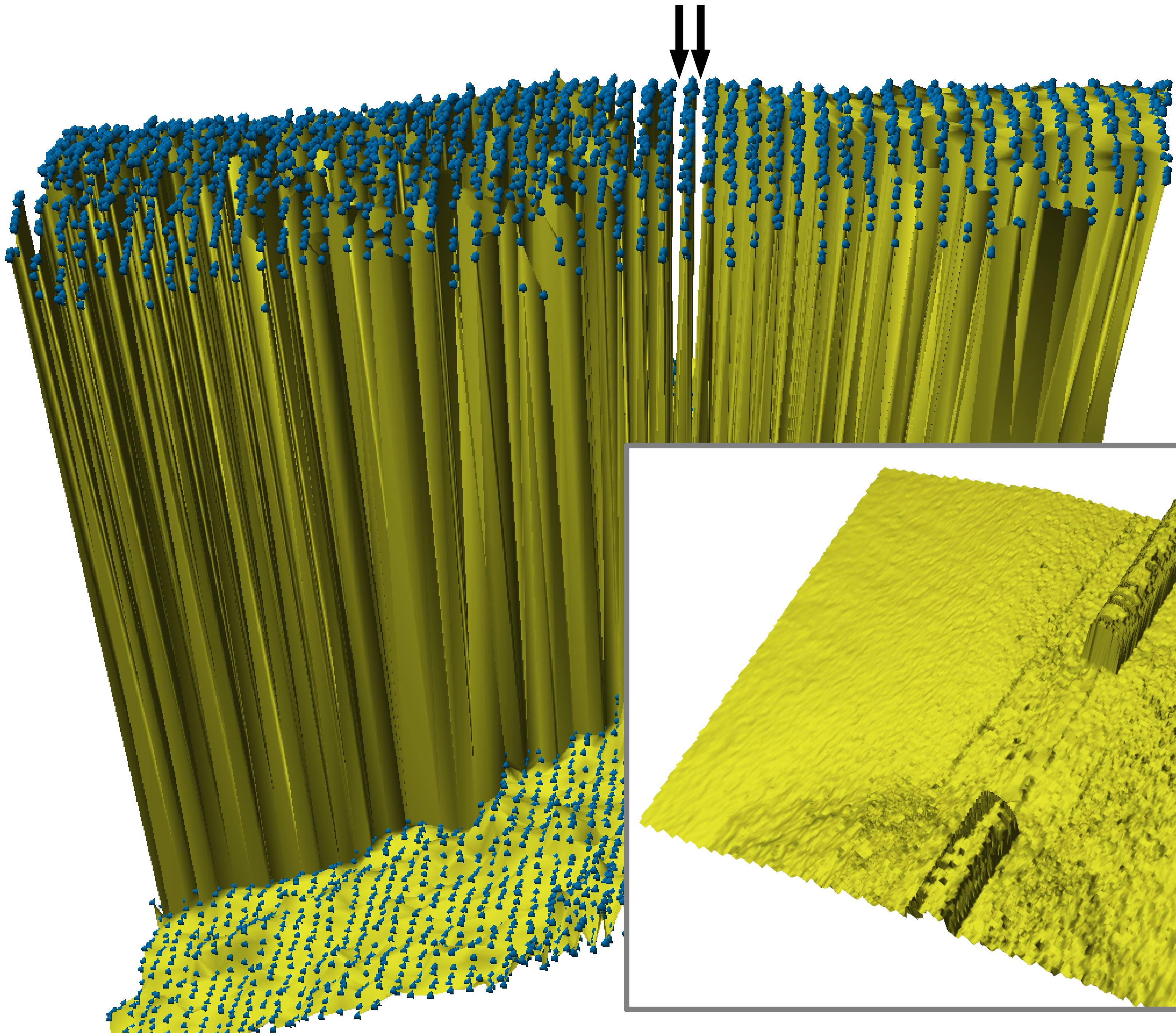
Data: StatoilHydro

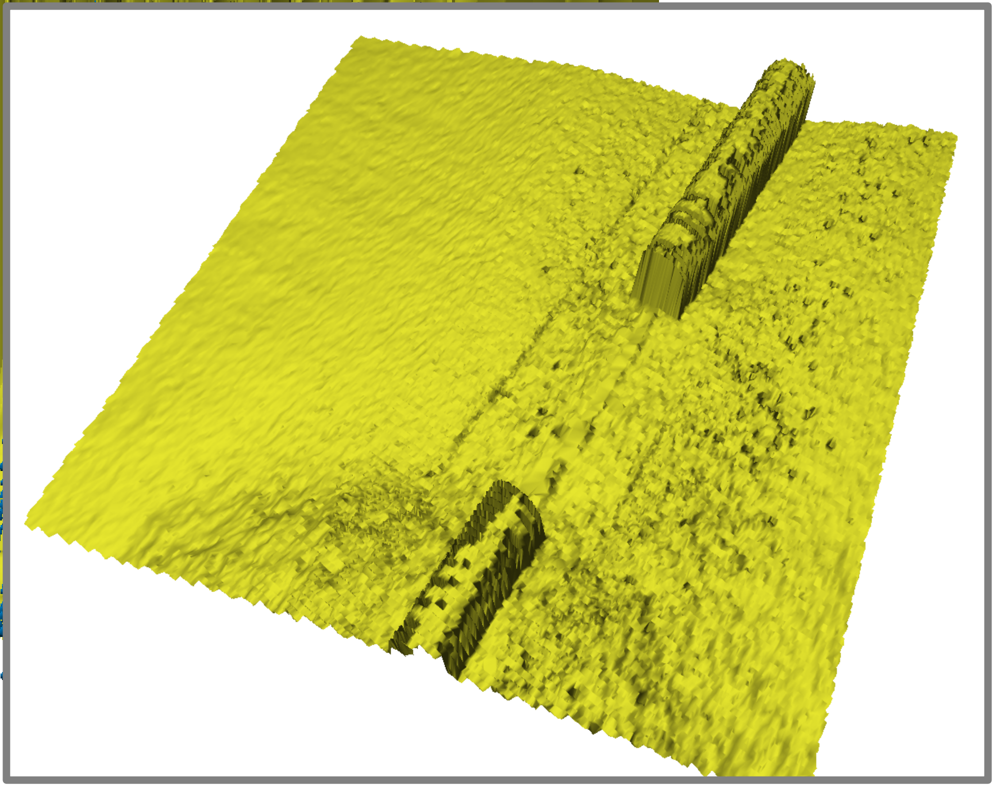


Why it works

no diagonals $\Rightarrow$ pipeline disconnected

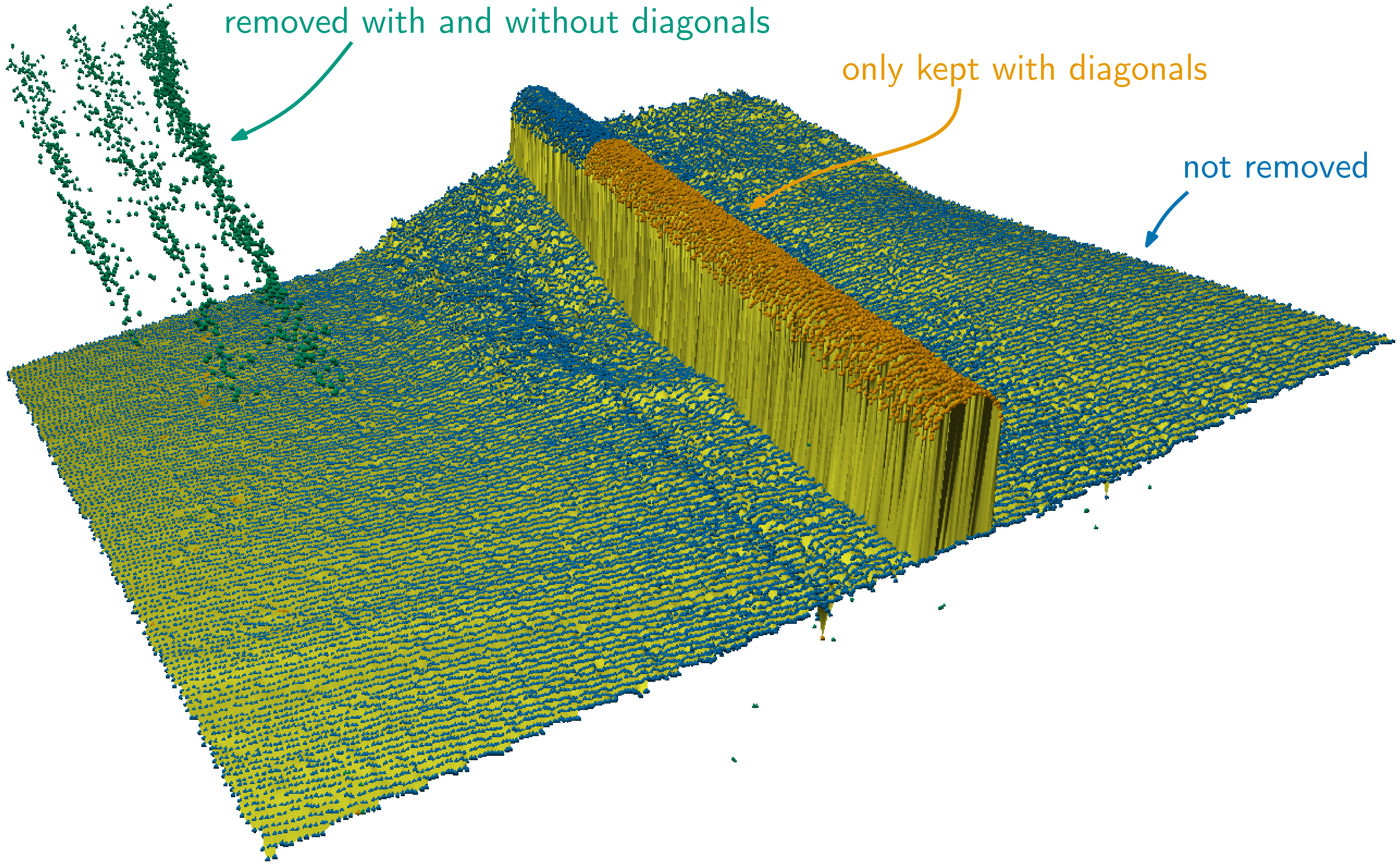
Data: StatoilHydro





Why it works
with vs. without diagonals

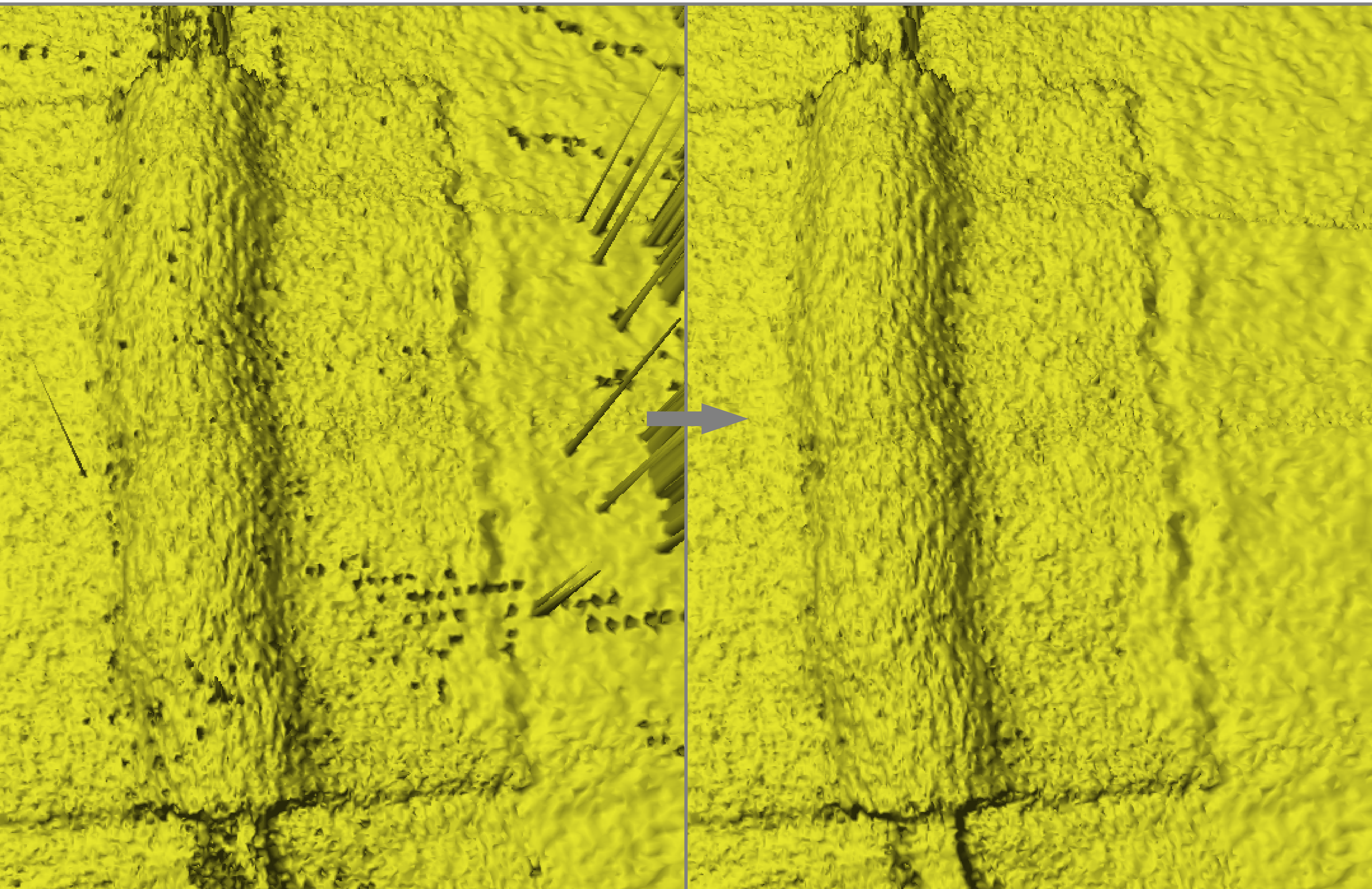
Data: StatoilHydro



Results

type-1 noise

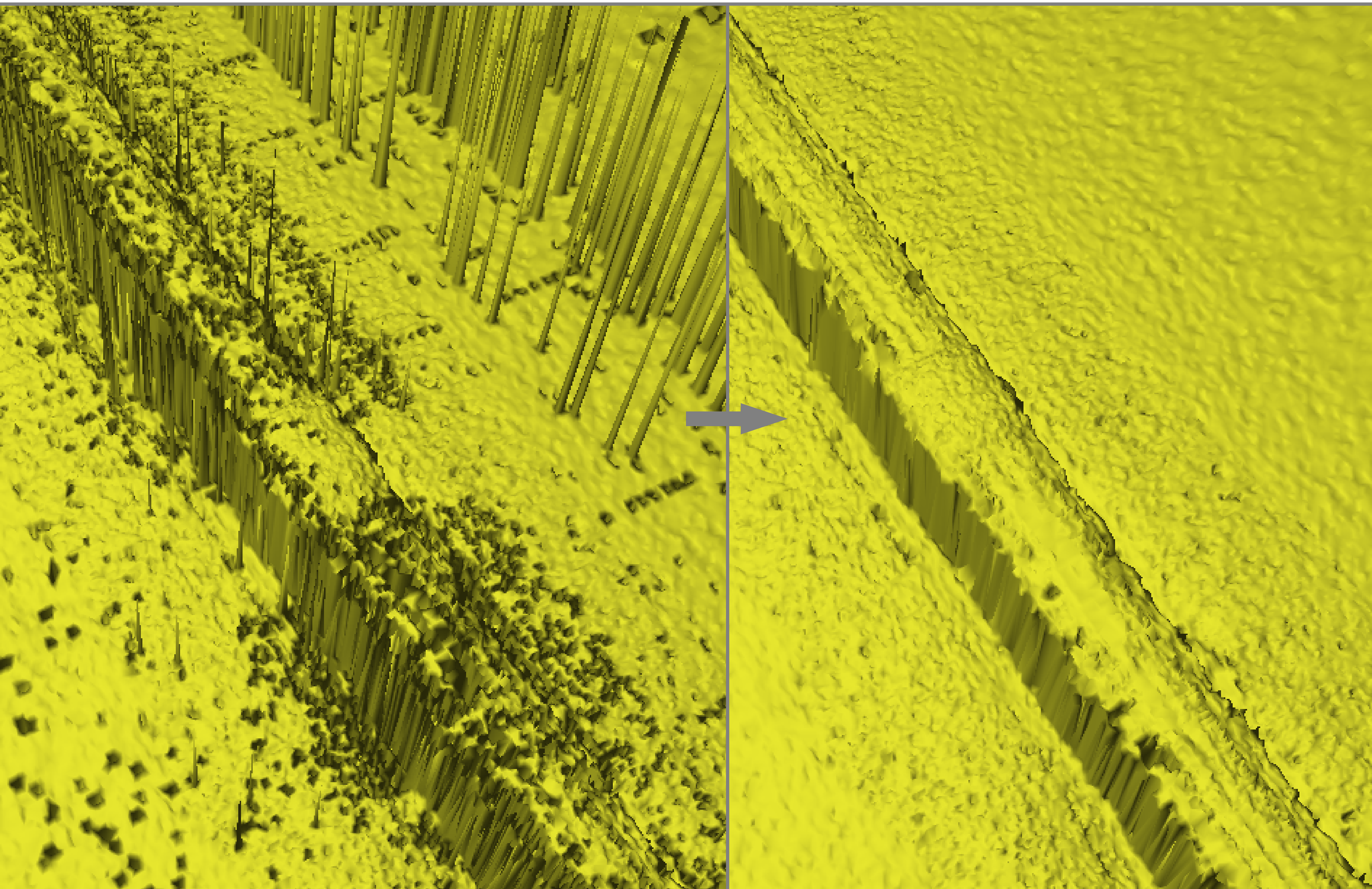
Data: EIVA



Results

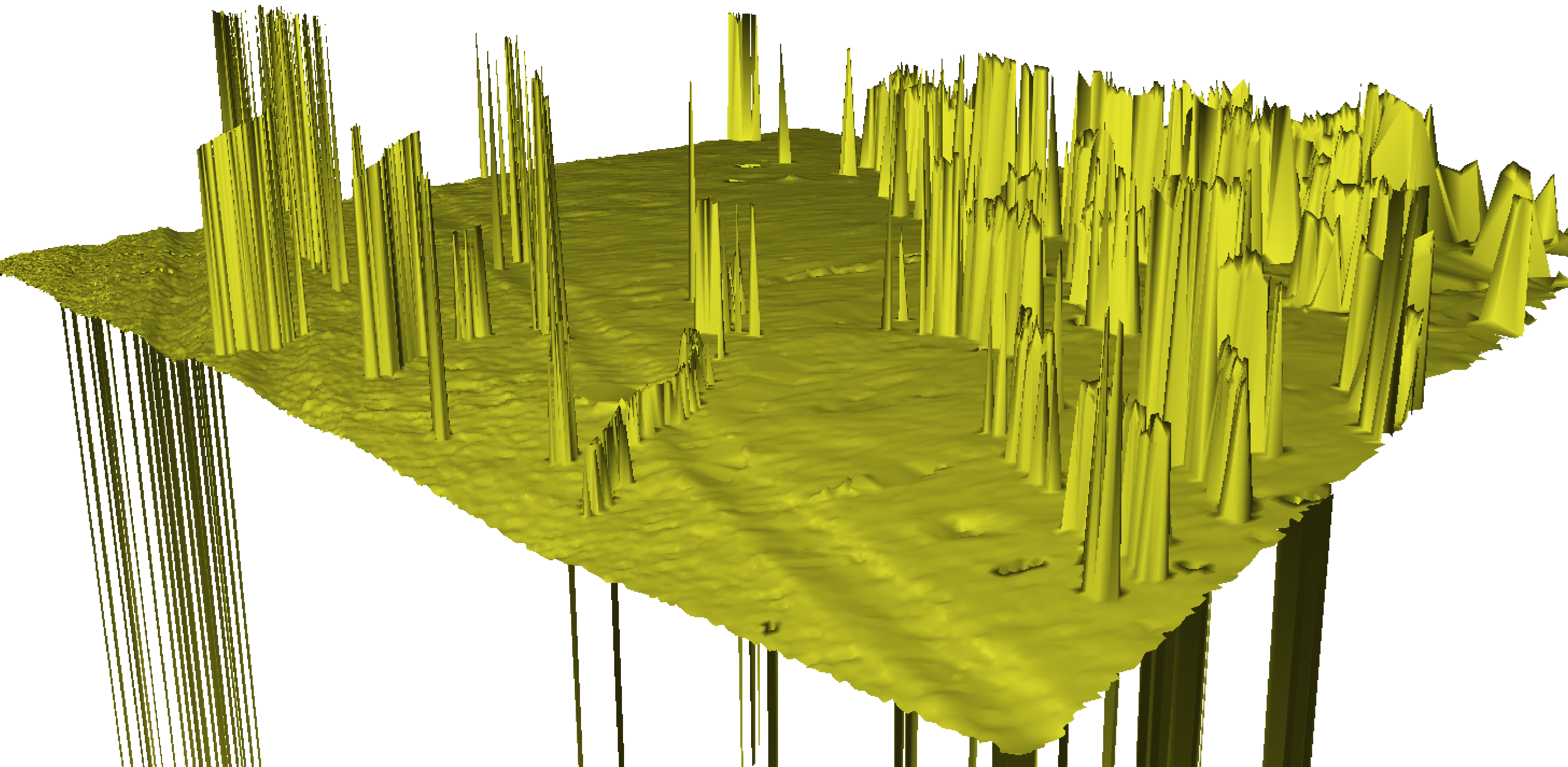
type-1 noise

Data: EIVA



Results
type-2 noise

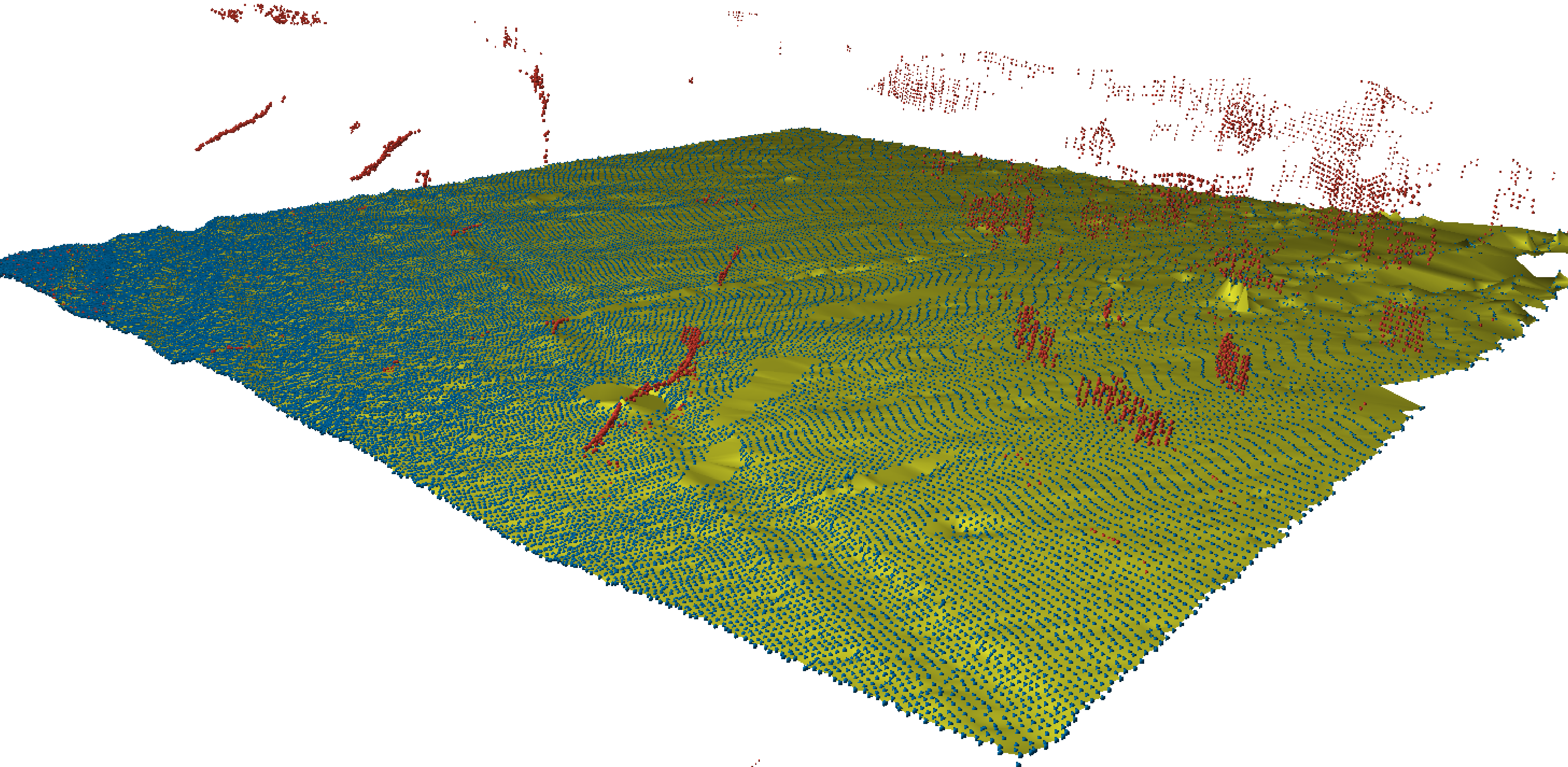
Data: StatoilHydro



Results

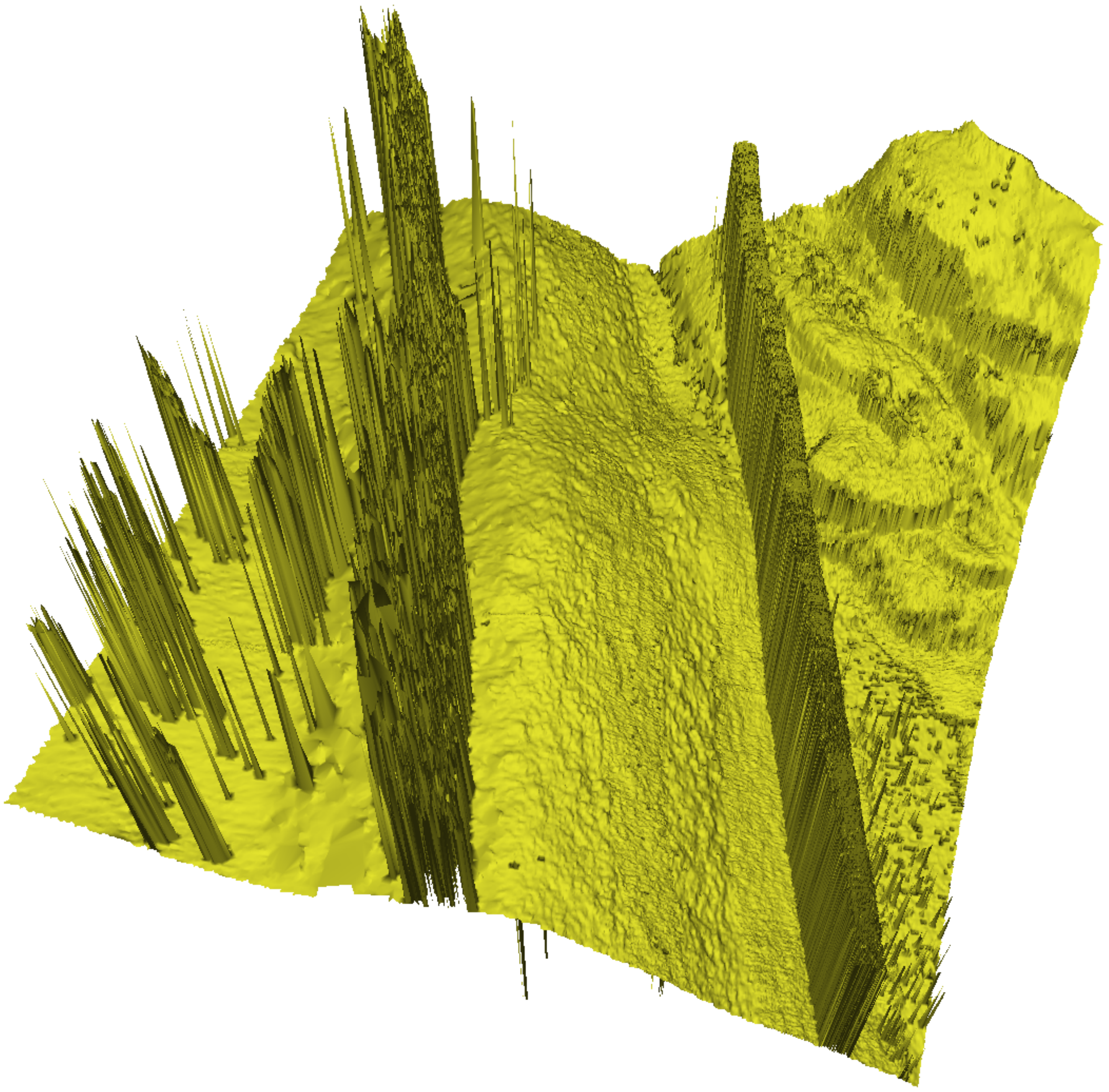
type-2 noise

Data: StatoilHydro



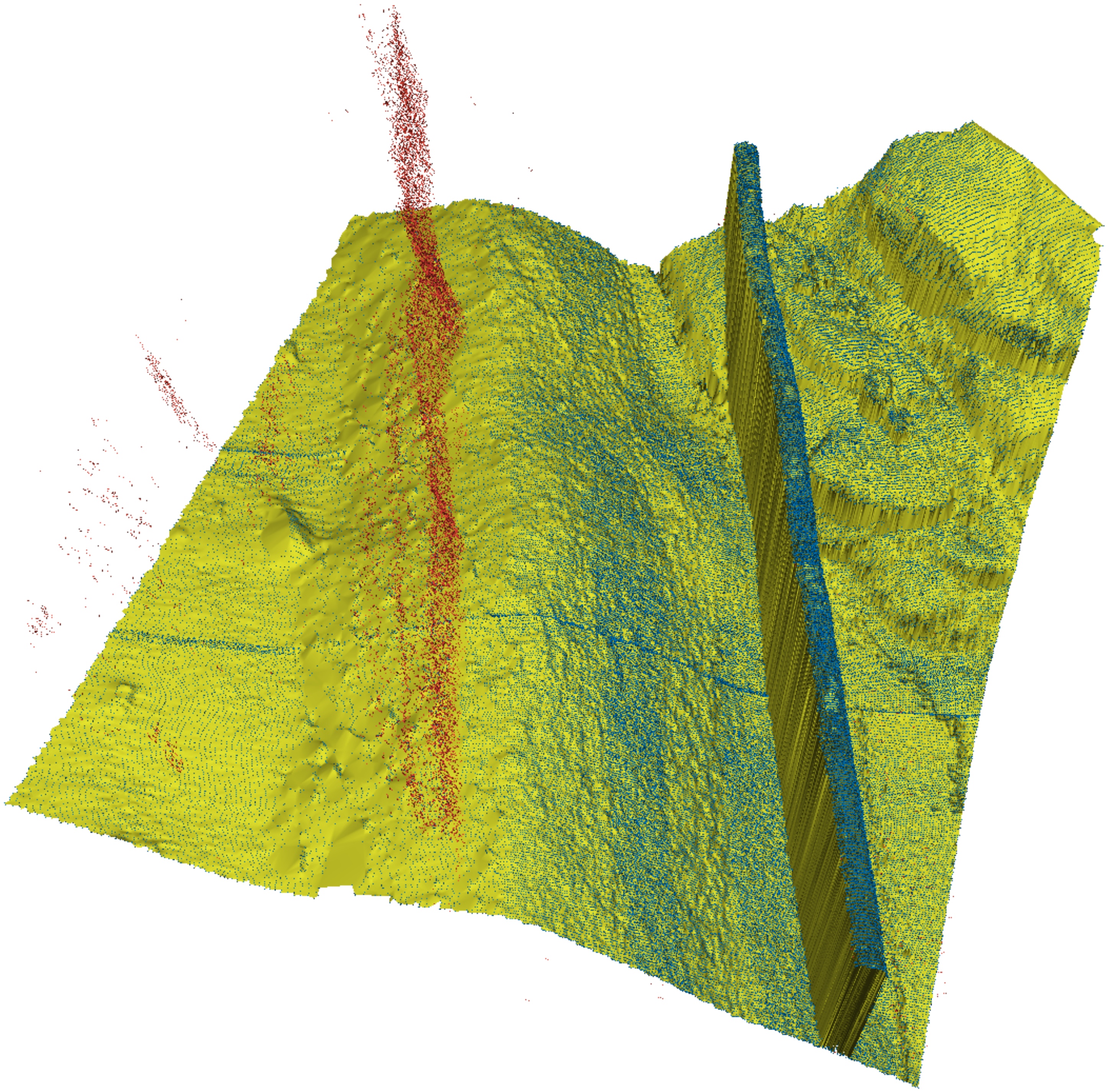
Results
type-3 noise

Data: StatoilHydro



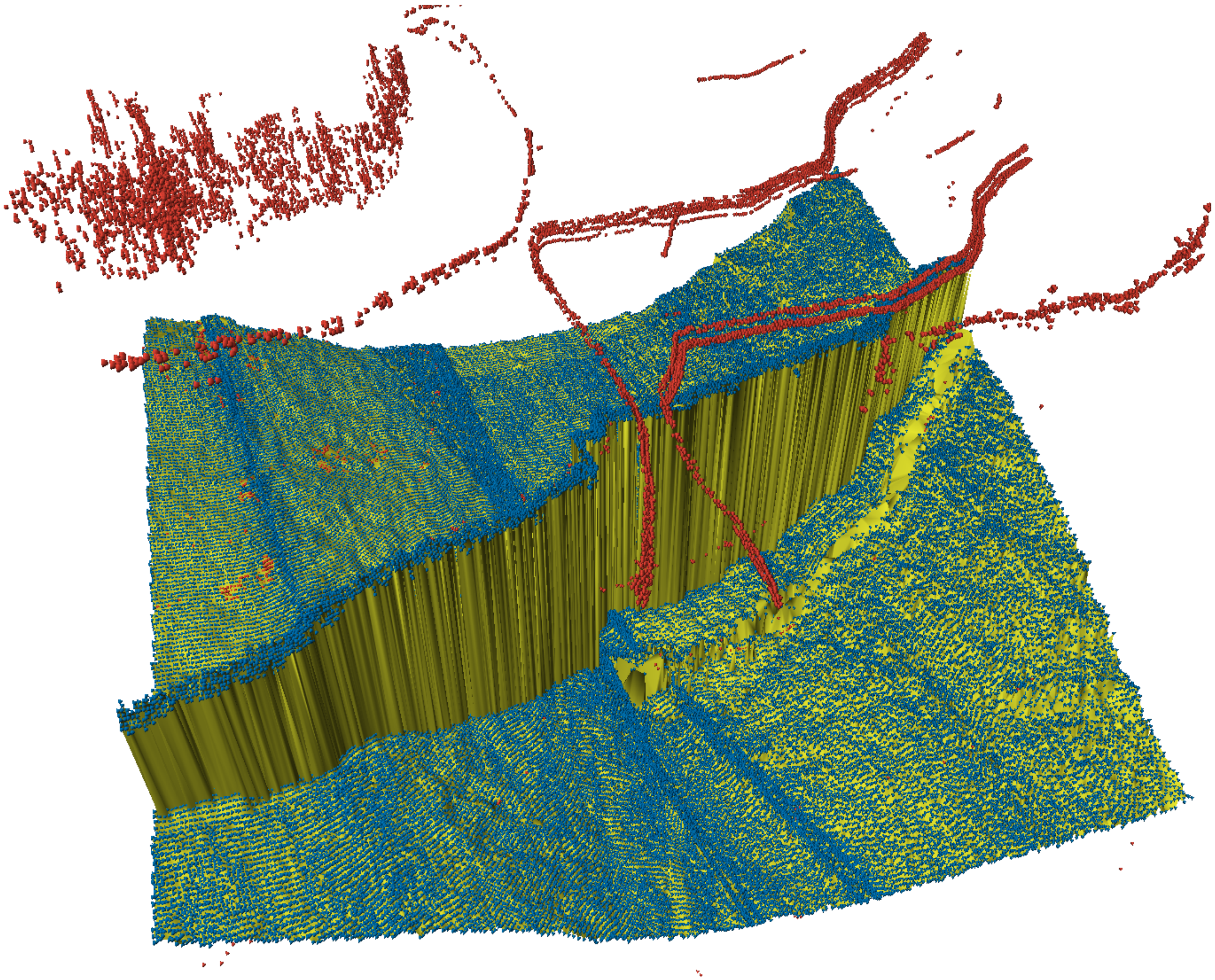
Results
type-3 noise

Data: StatoilHydro



Results
type-3 noise

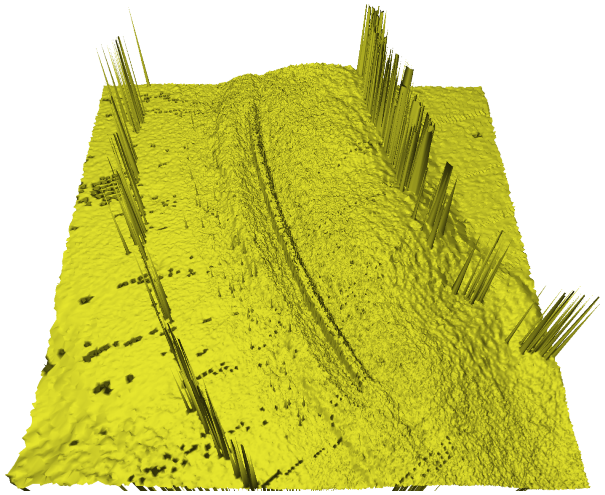
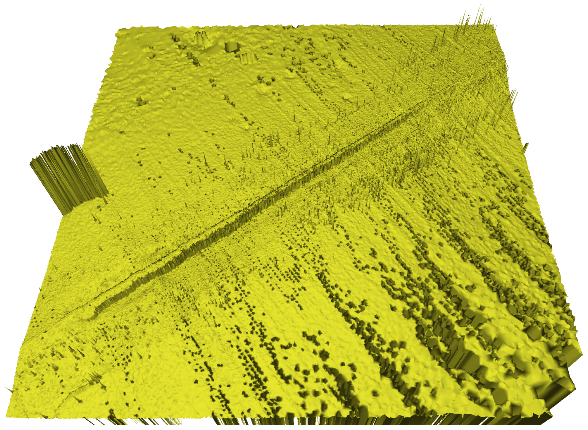
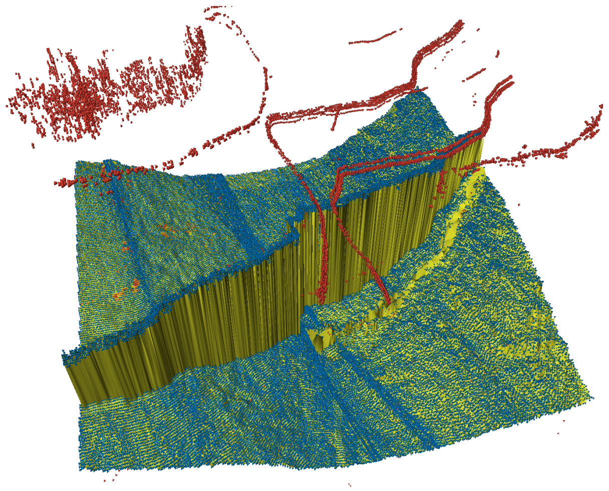
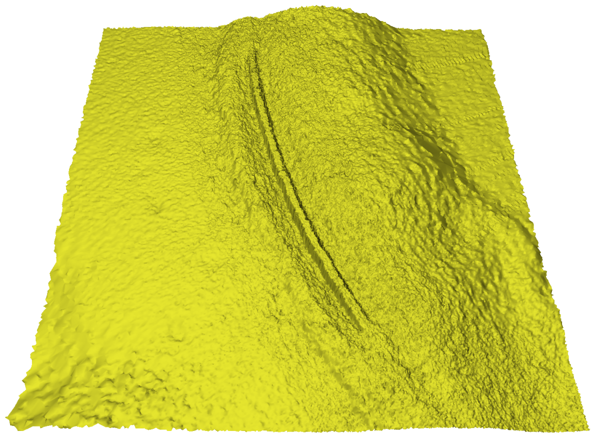
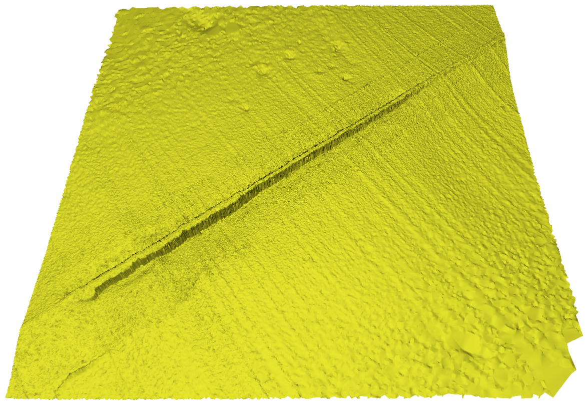
Data: StatoilHydro



Some numbers

Data: StatoilHydro, EIVA

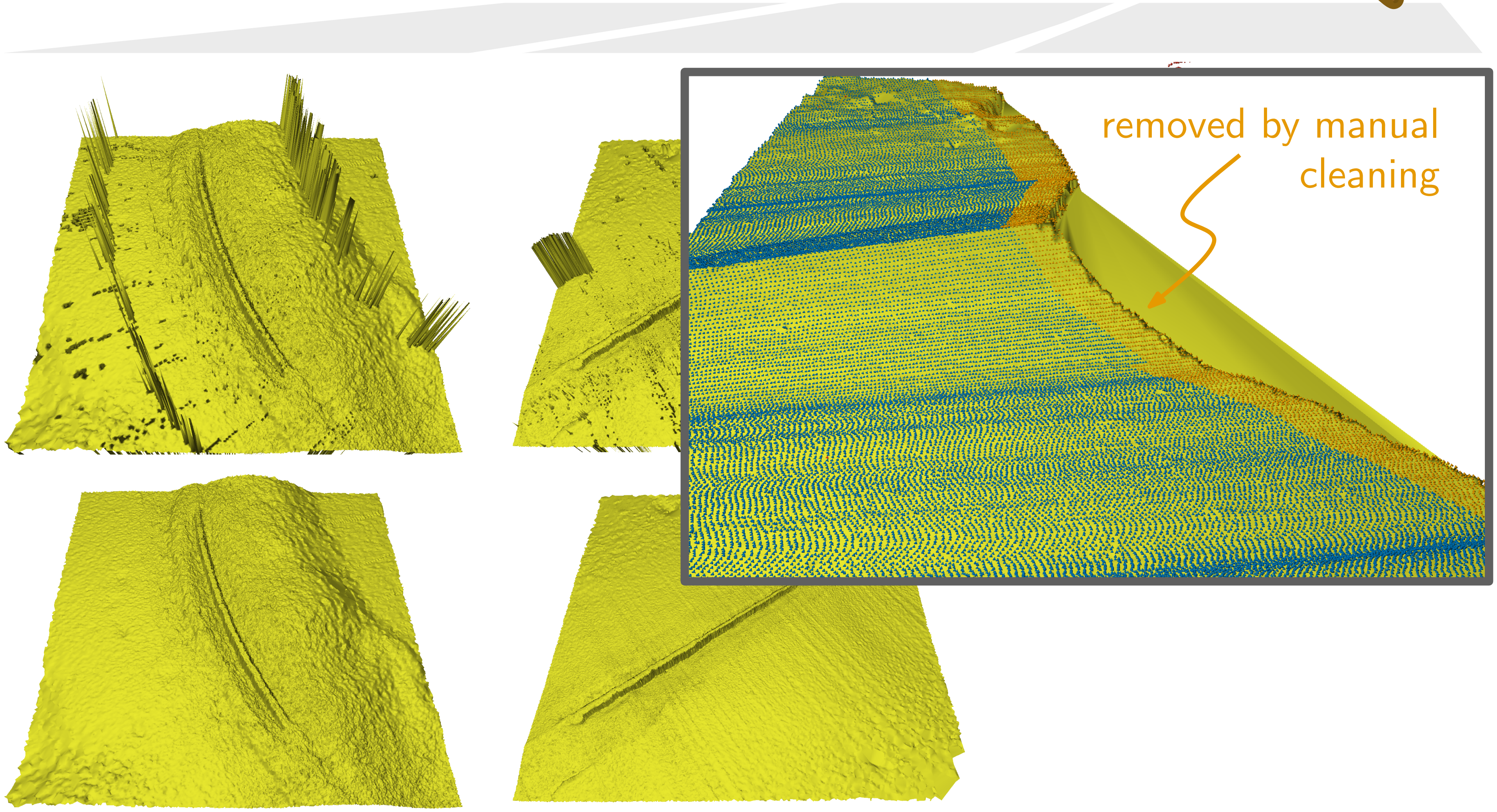
Noise	little	some	much
Threshold	5 cm	5 cm	35 cm
Manually removed: not auto.	0.4%	13%	18%
Not manual. removed: only auto.	0.4%	0.3%	0.8%

Some numbers

Data: StatoilHydro, EIVA

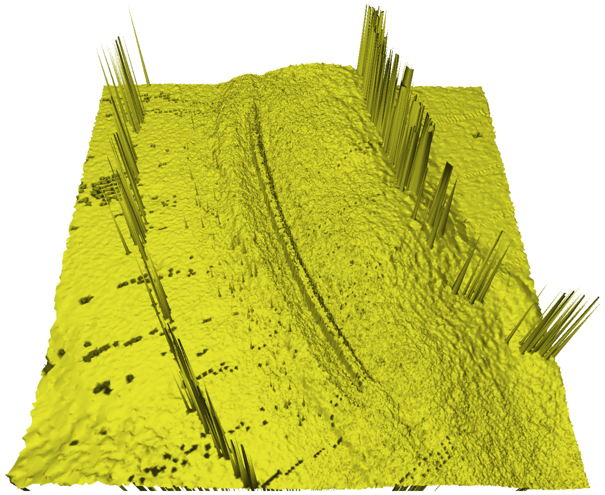
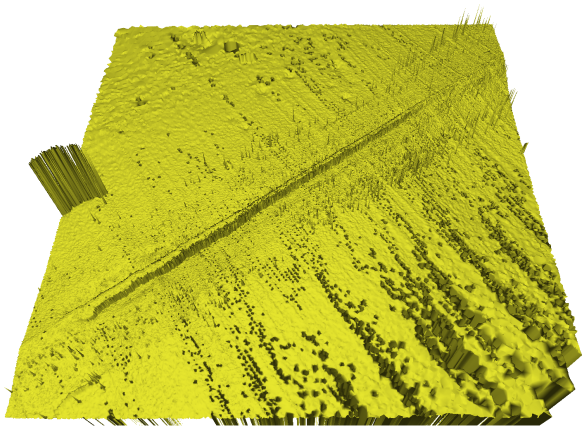
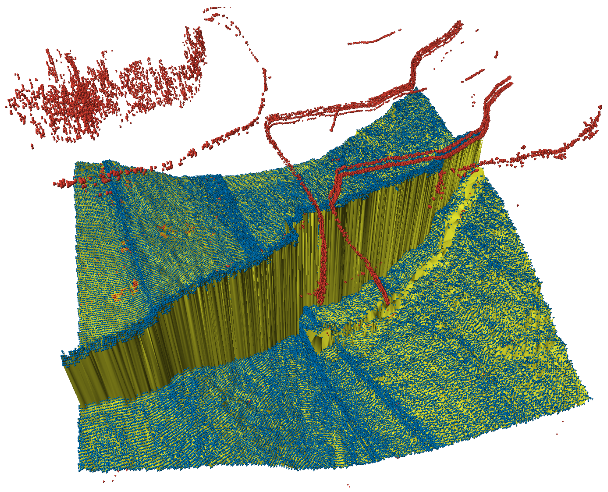
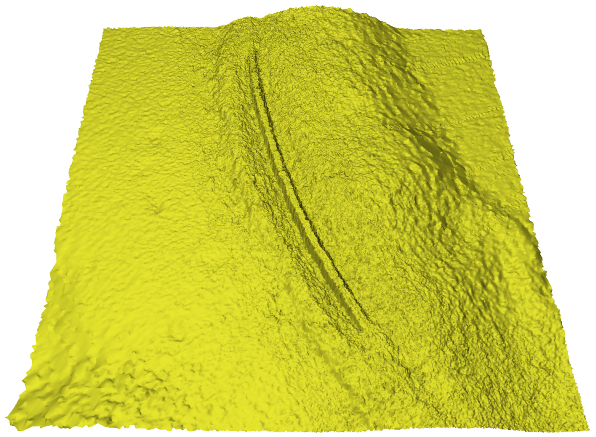
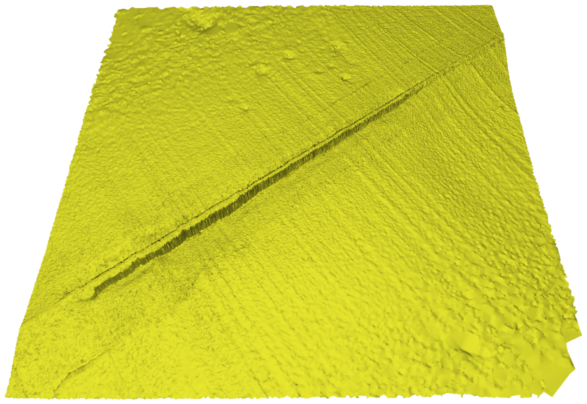
Noise	little	some	much
Threshold	5 cm	5 cm	35 cm
Manually removed: not auto.	0.4%	13%	18%
Not manual. removed: only auto.	0.4%	0.3%	0.8%



Some numbers

Data: StatoilHydro, EIVA

Noise	little	some	much
Threshold	5 cm	5 cm	35 cm
Manually removed: not auto.	0.4%	13%	18%
Not manual. removed: only auto.	0.4%	0.3%	0.8%

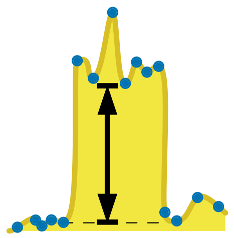
Conclusion, future work

Implemented in commercial product

Conclusion, future work

Implemented in commercial product

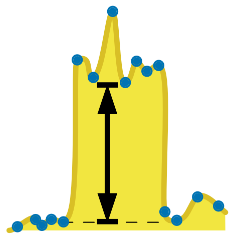
- Open problem: defining theoretical model of outlier noise
  - Objective theoretical performance analysis
  - Compare Delaunay triangulation with other neighbourhood graphs



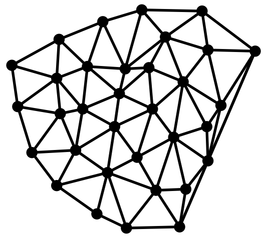
Conclusion, future work

Implemented in commercial product

- Open problem: defining theoretical model of outlier noise
  - Objective theoretical performance analysis
  - Compare Delaunay triangulation with other neighbourhood graphs



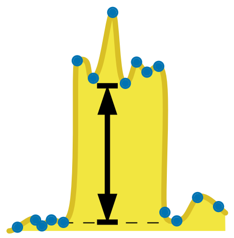
- Open problem: find easier alternative to Delaunay triangulation
  Requirements:
  - Good connectivity
  - Fast to compute



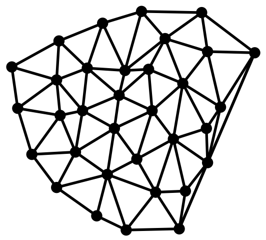
Conclusion, future work

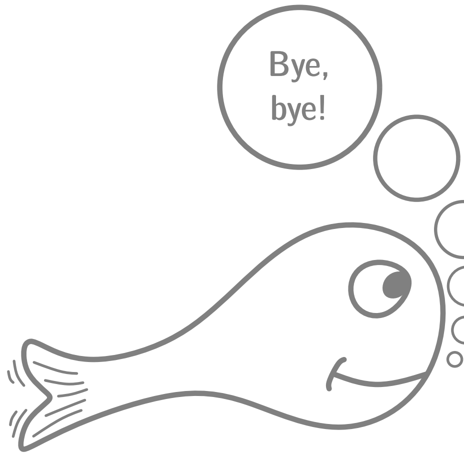
Implemented in commercial product

- Open problem: defining theoretical model of outlier noise
  - Objective theoretical performance analysis
  - Compare Delaunay triangulation with other neighbourhood graphs

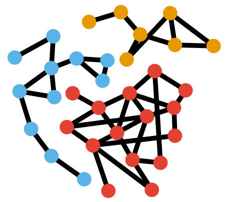


- Open problem: find easier alternative to Delaunay triangulation
  Requirements:
  - Good connectivity
  - Fast to compute



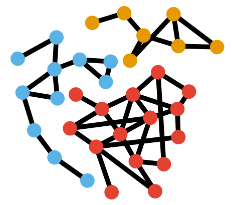


Connected component algorithm



- Compute connected component labelling:
  vertices have equal labels $\Leftrightarrow$ they are in the same connected component

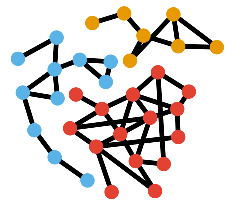
Connected component algorithm



- Compute connected component labelling:
  vertices have equal labels $\Leftrightarrow$ they are in the same connected component
- Algorithm: two phases, sweeping over edge & vertex lists
  - *Down phase*: augment some vertices with additional connectivity info.
  - *Up phase*: compute final component labels

  Assumption: edges intersecting sweep line always fit in main memory

Connected component algorithm

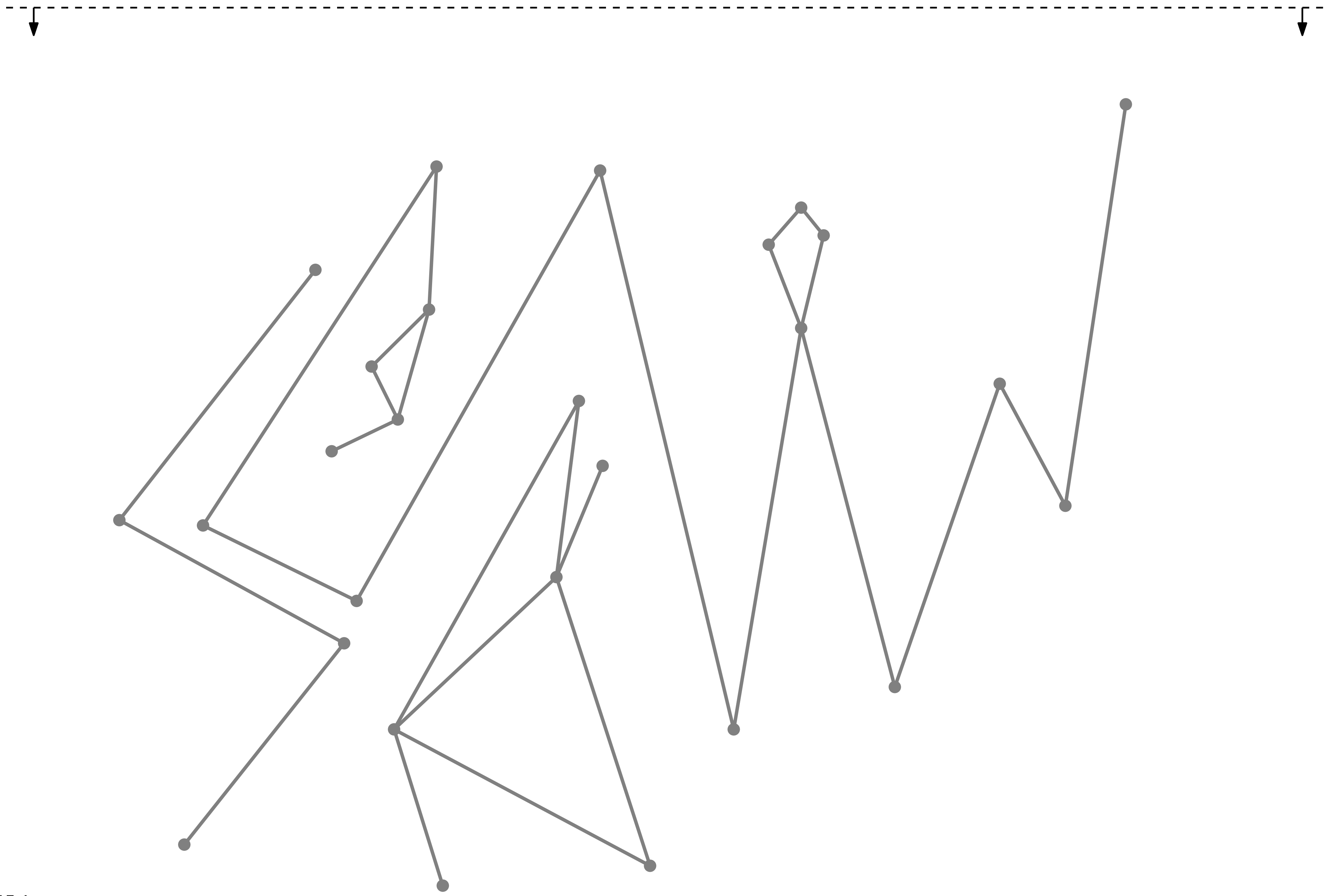


- Compute connected component labelling:
  vertices have equal labels $\Leftrightarrow$ they are in the same connected component
- Algorithm: two phases, sweeping over edge & vertex lists
  - *Down phase*: augment some vertices with additional connectivity info.
  - *Up phase*: compute final component labels

  Assumption: edges intersecting sweep line always fit in main memory
- Total number of I/Os necessary: $O(\mathtt{sort}(N))$

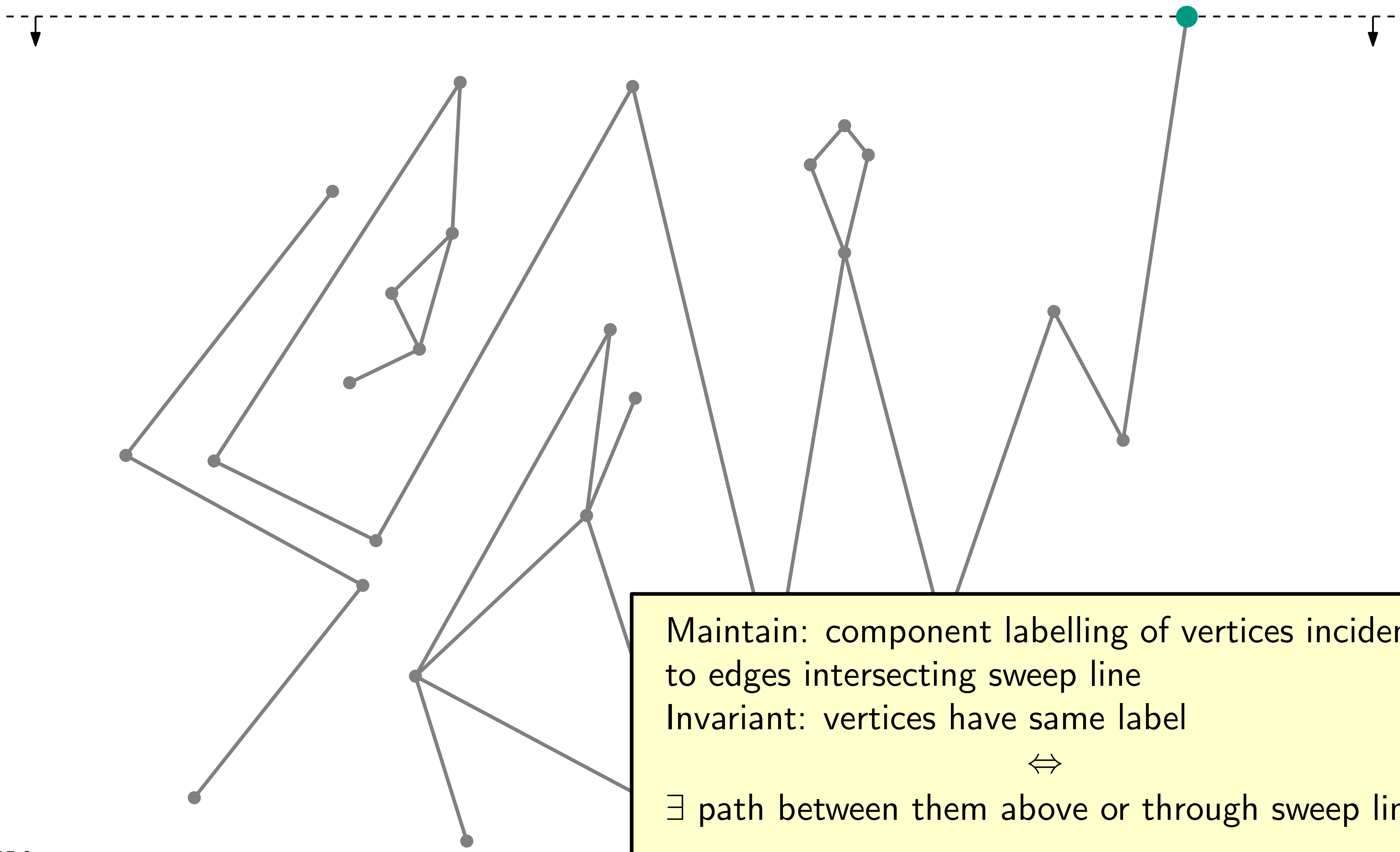
Connected component algorithm

down phase



Connected component algorithm

down phase



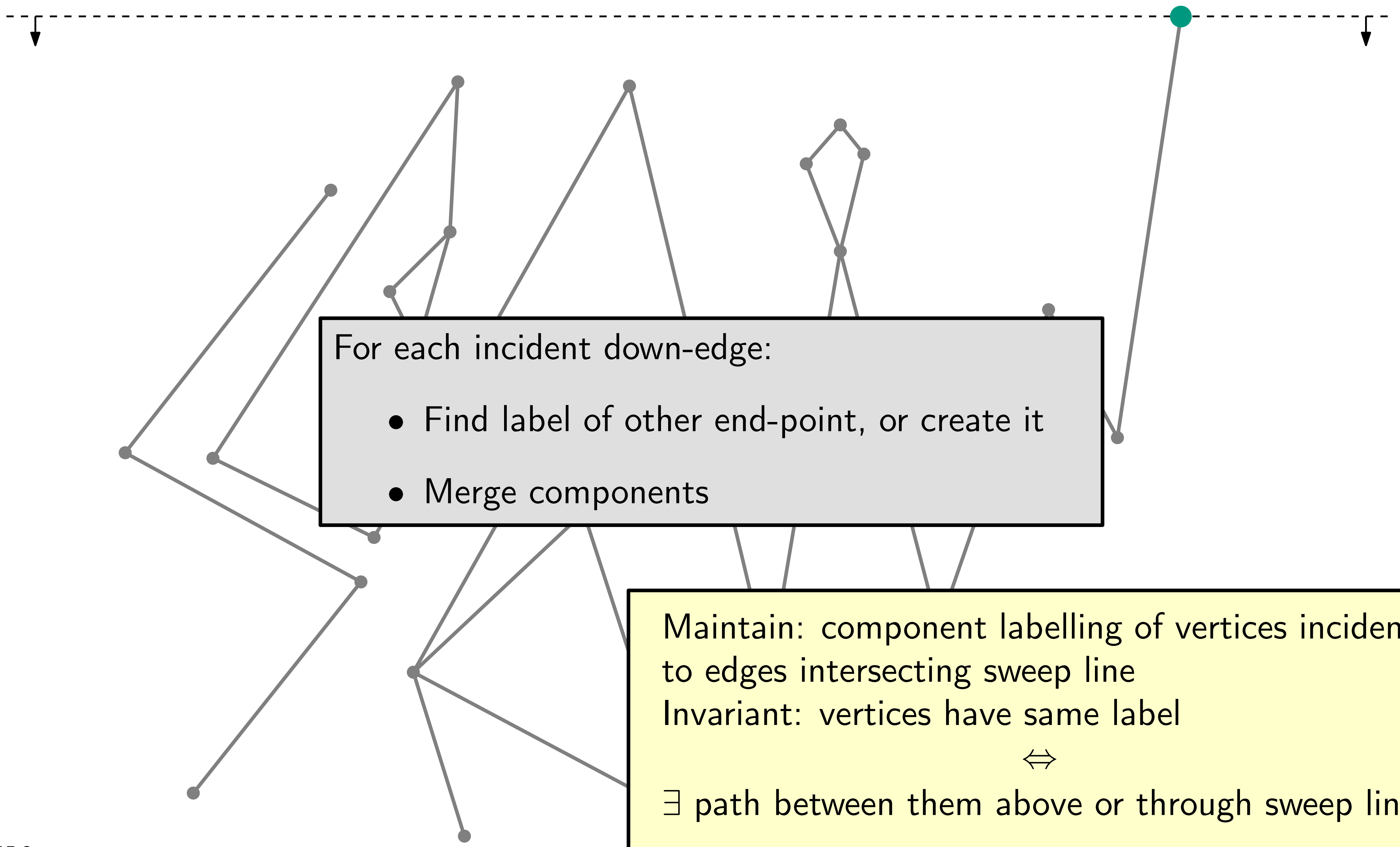
Maintain: component labelling of vertices incident to edges intersecting sweep line

Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase



For each incident down-edge:

- Find label of other end-point, or create it
- Merge components

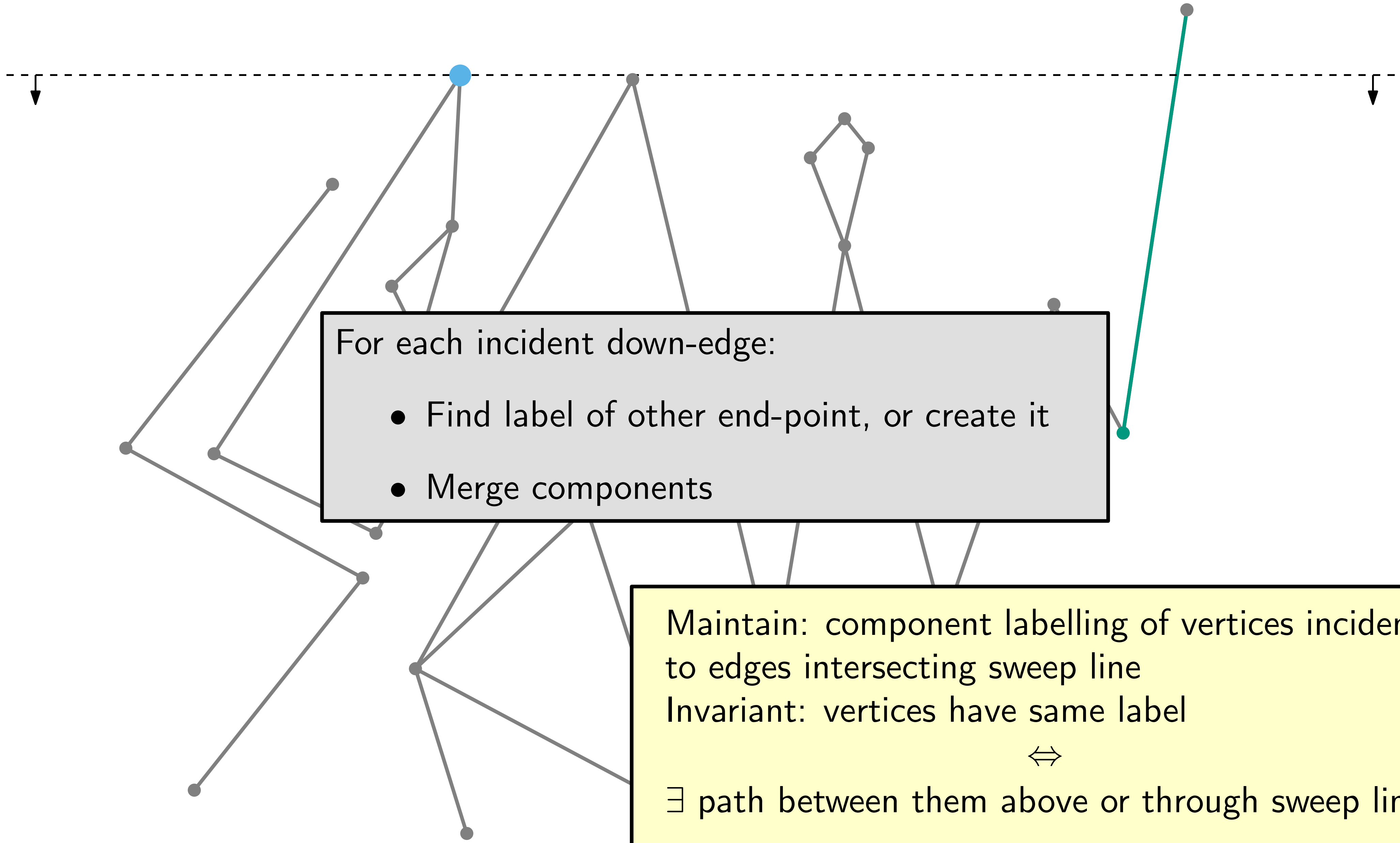
Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

Connected component algorithm

down phase



For each incident down-edge:

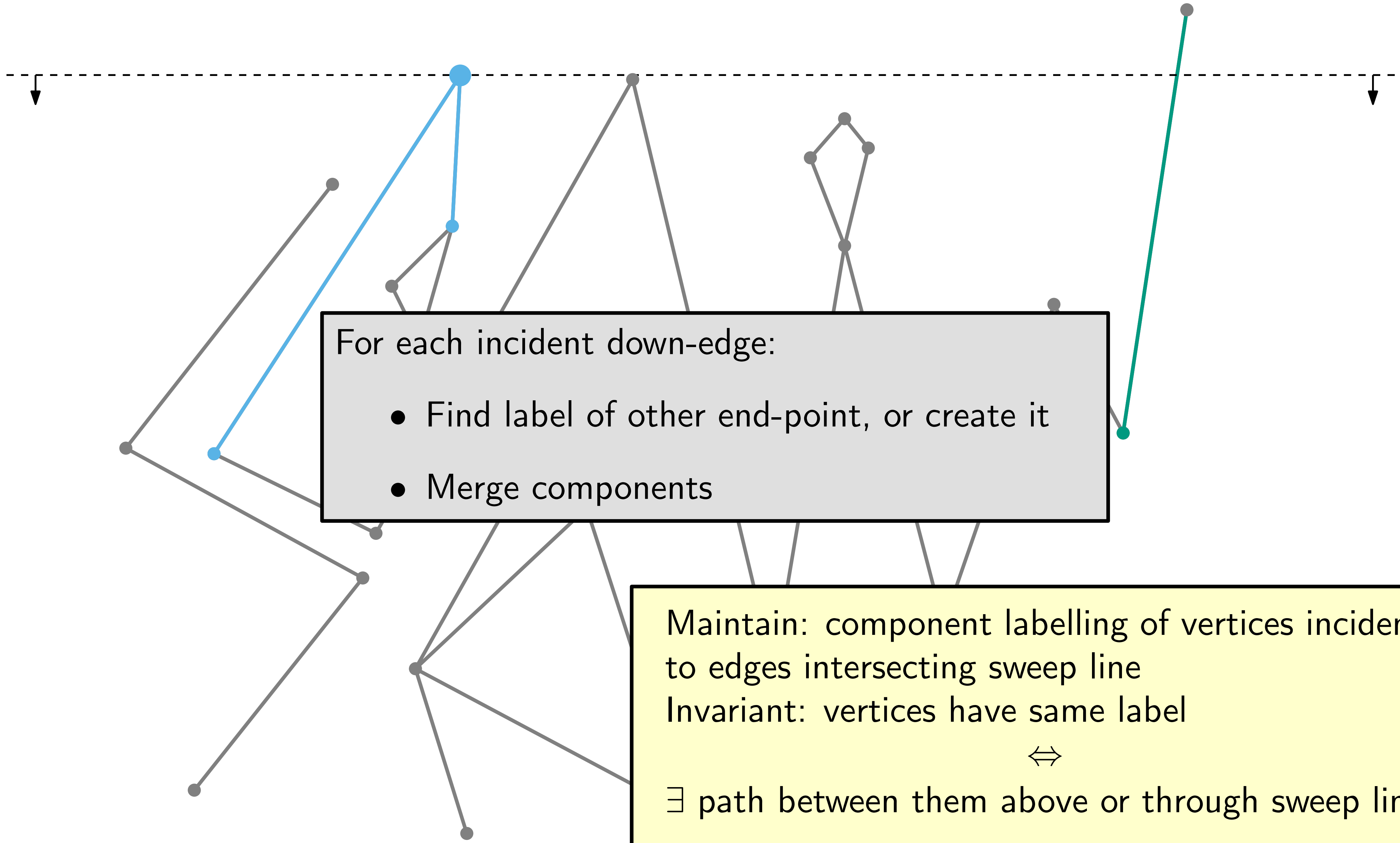
- Find label of other end-point, or create it
- Merge components

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase



For each incident down-edge:

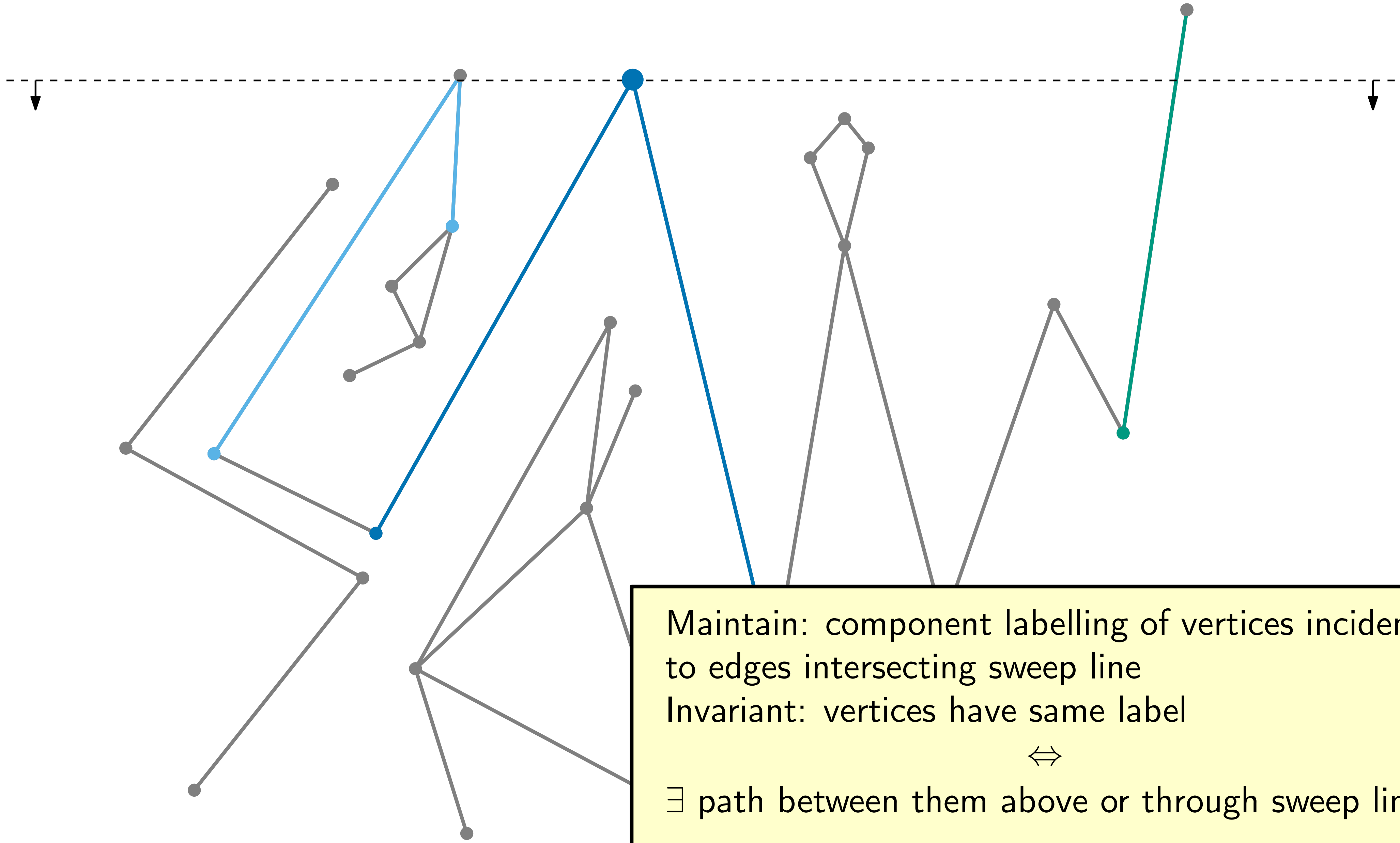
- Find label of other end-point, or create it
- Merge components

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase

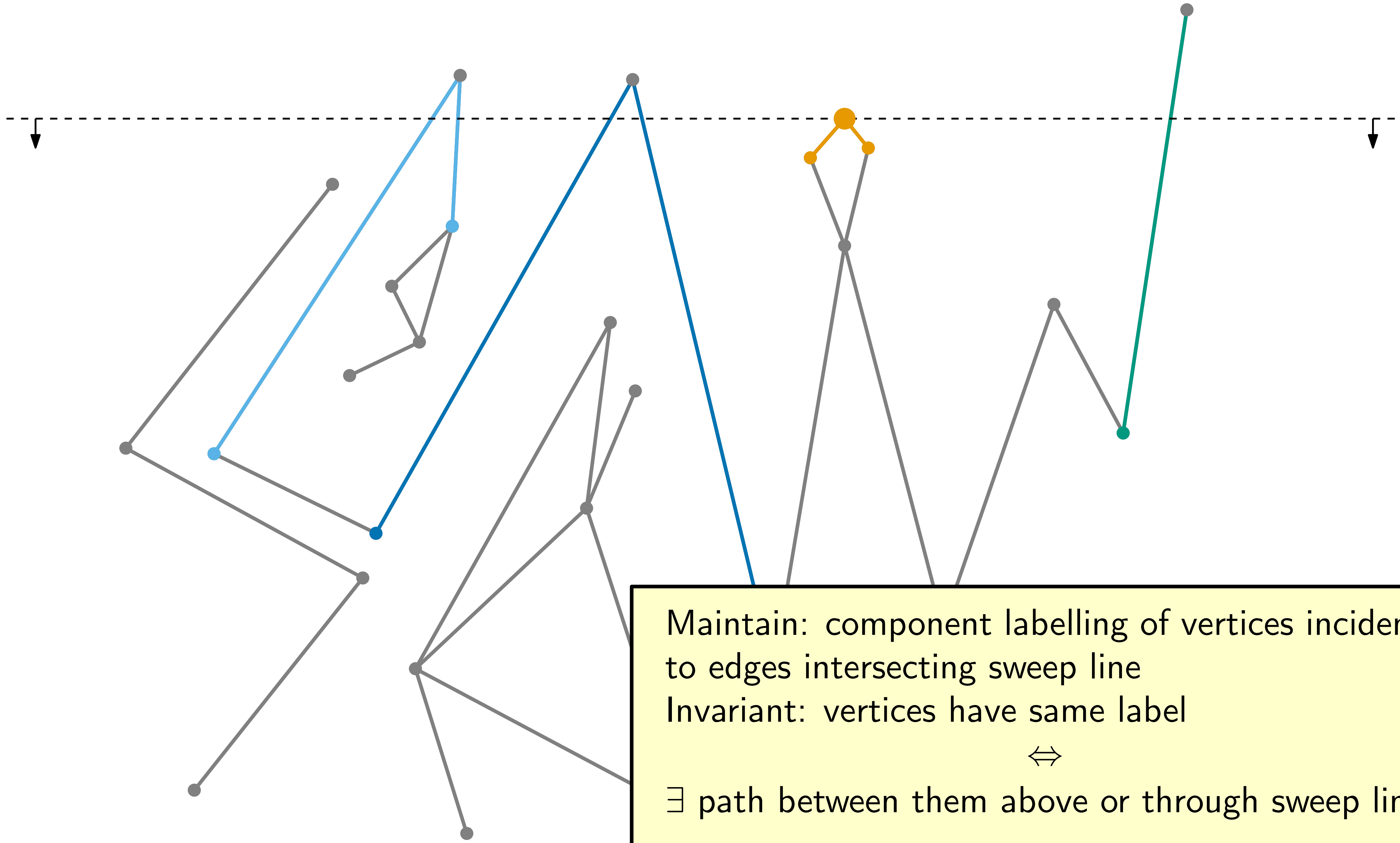


Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase



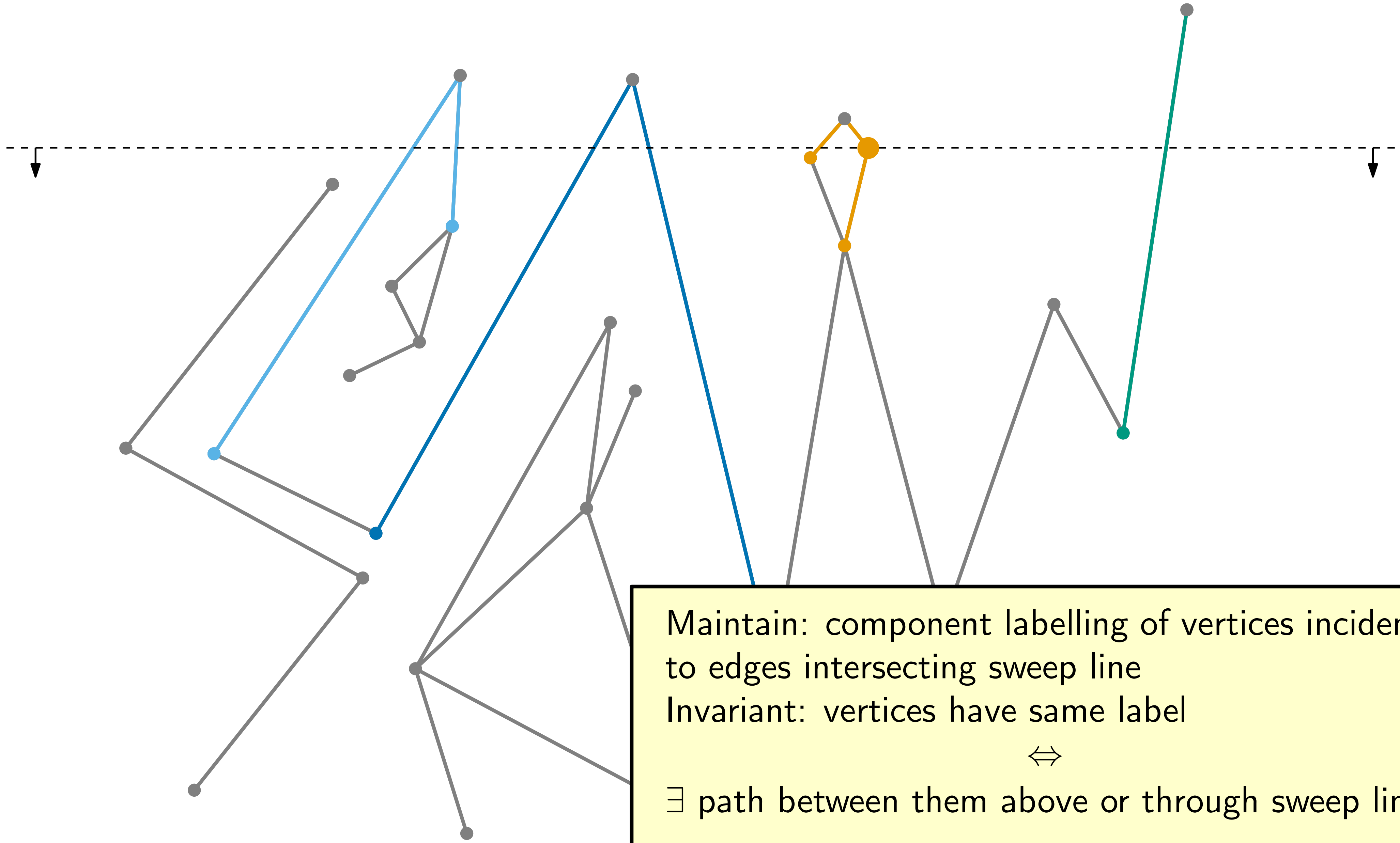
Maintain: component labelling of vertices incident to edges intersecting sweep line

Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase

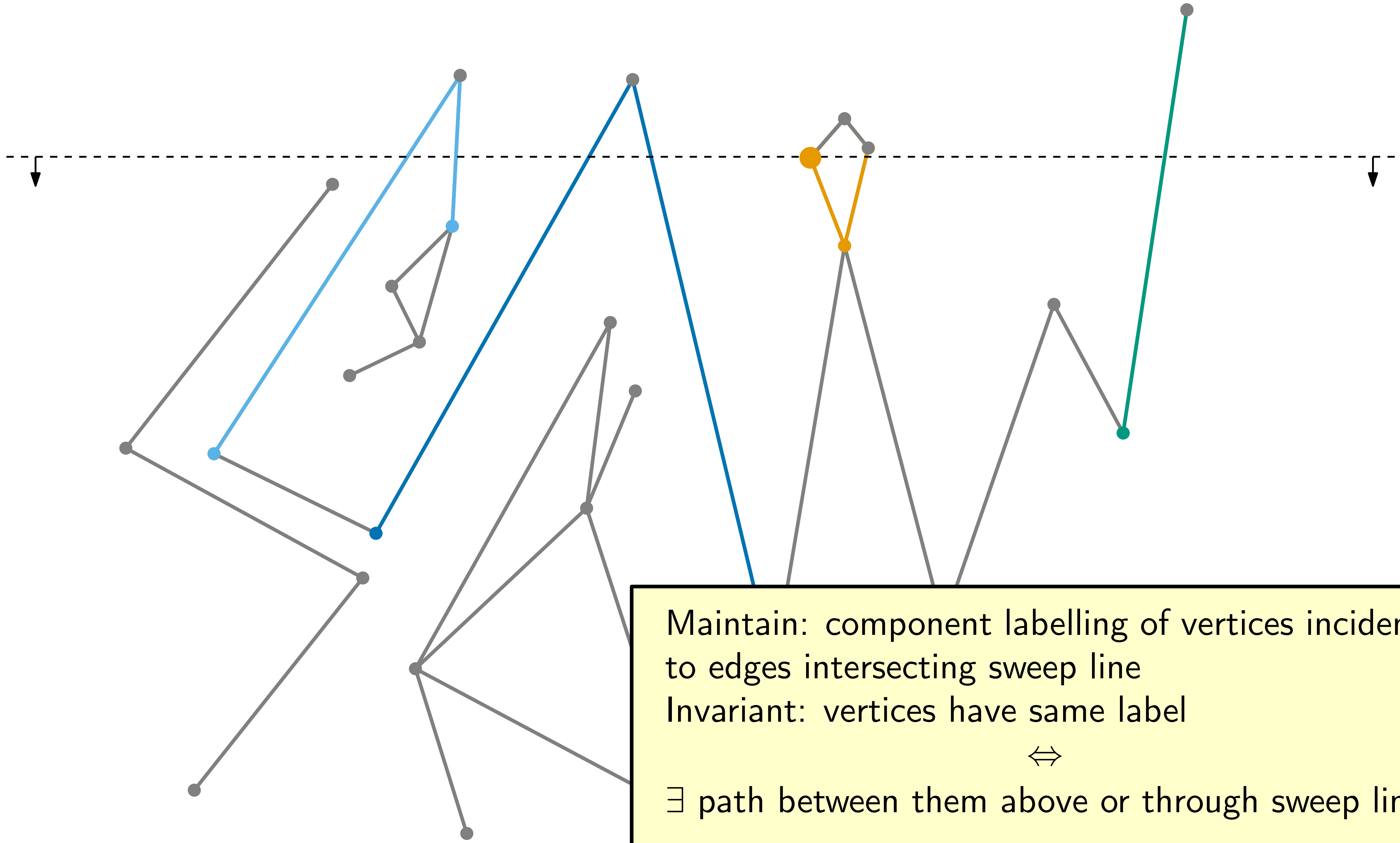


Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase

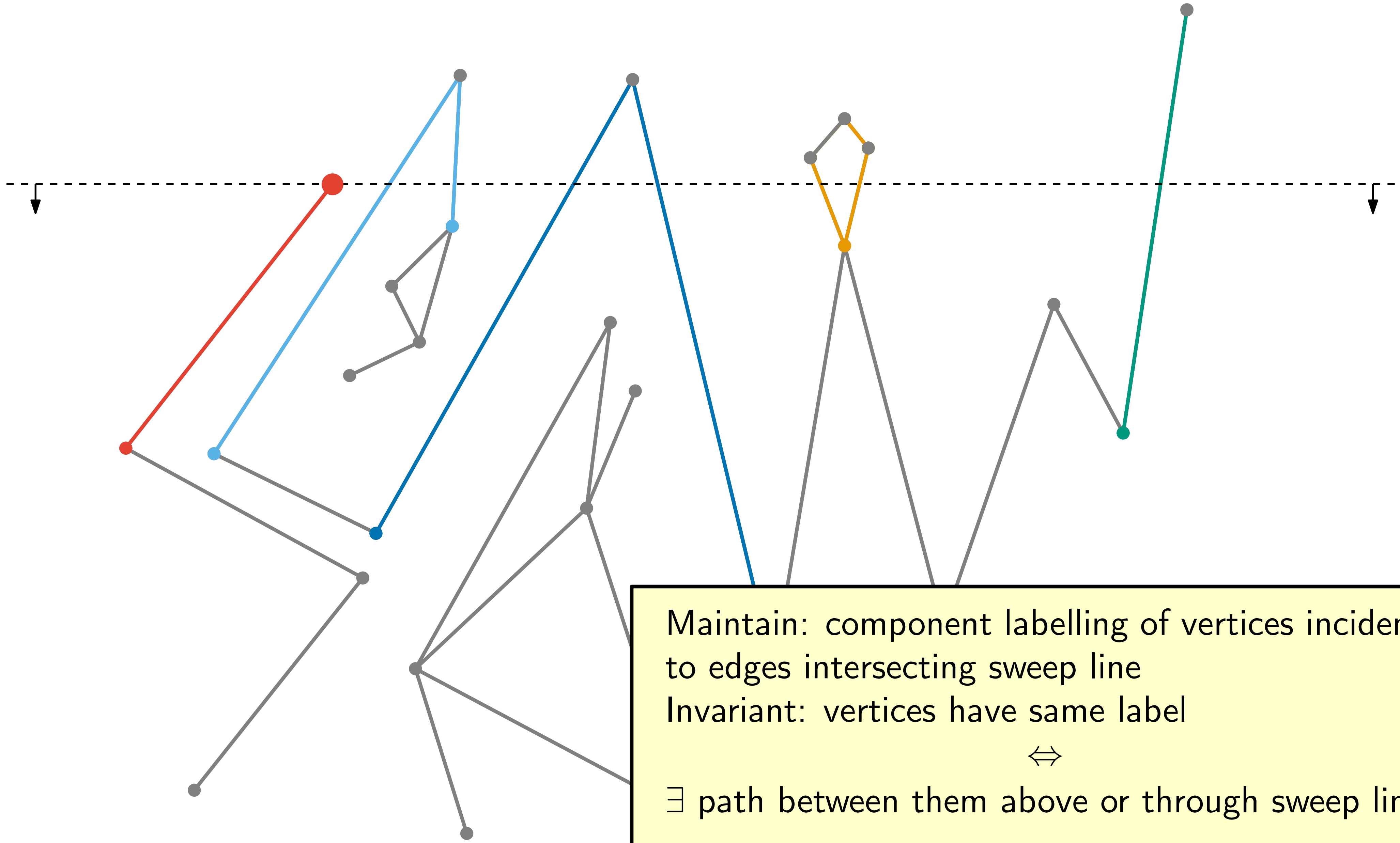


Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase

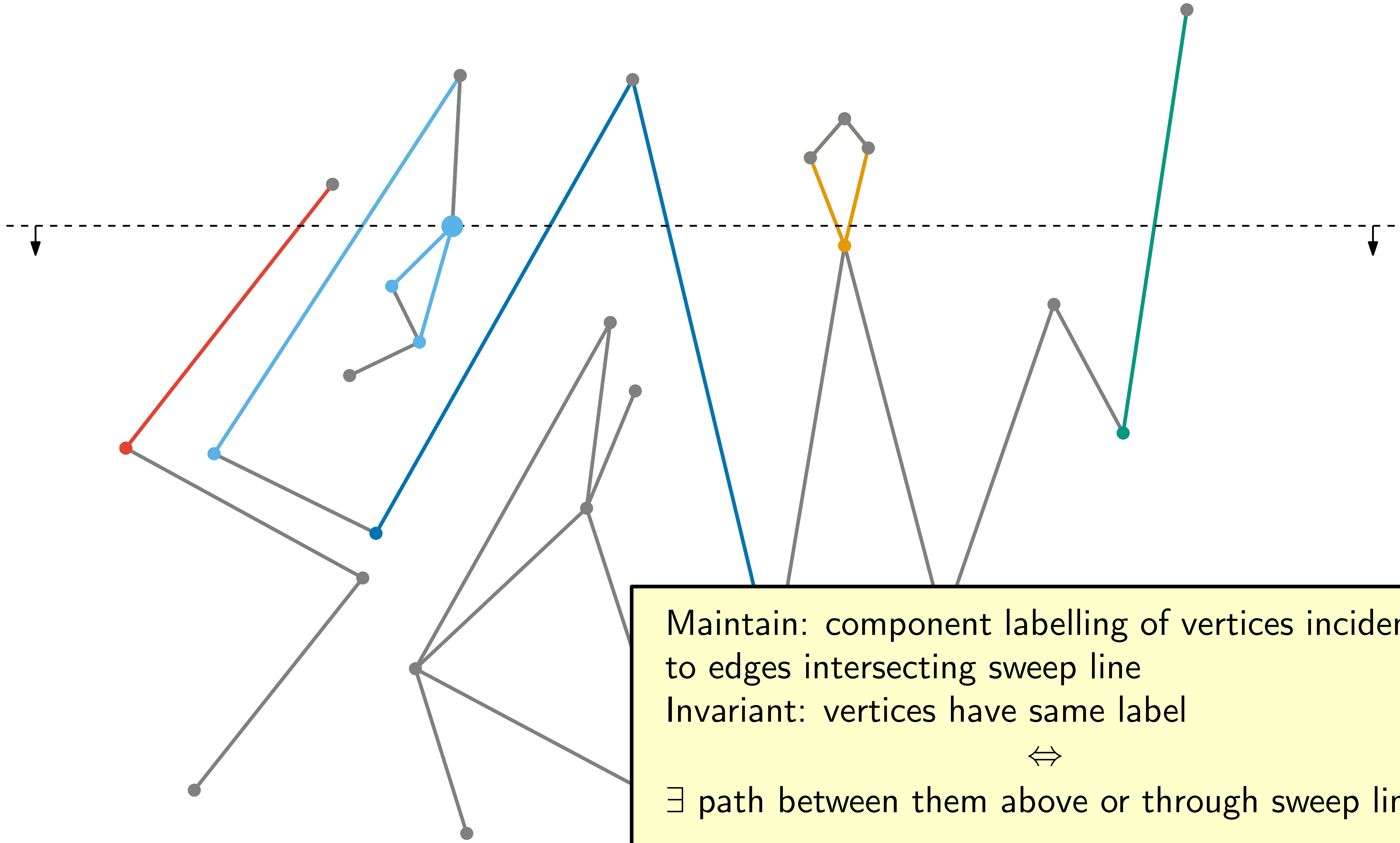


Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

Connected component algorithm
down phase



Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

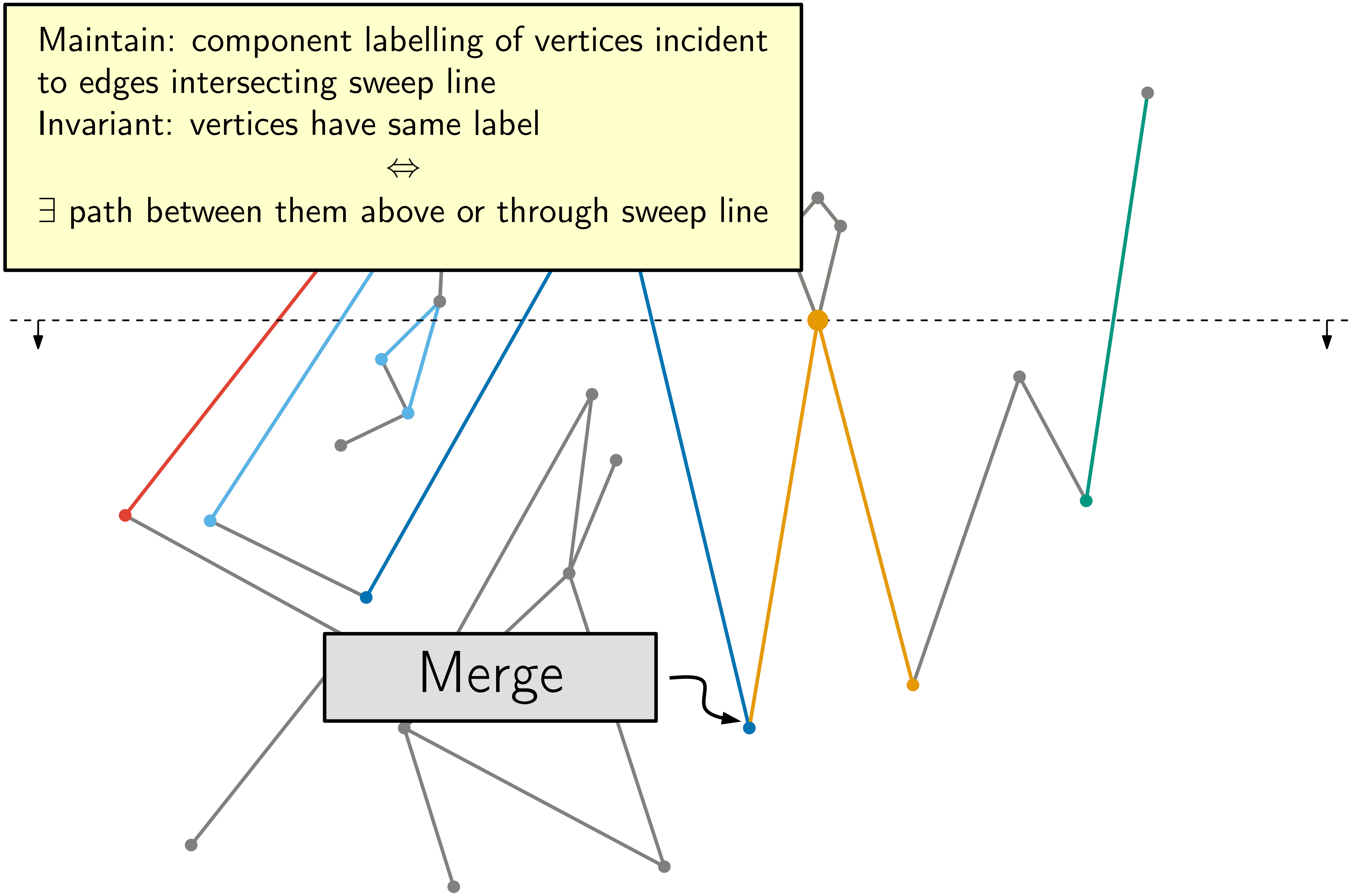
$\exists$ path between them above or through sweep line

Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line

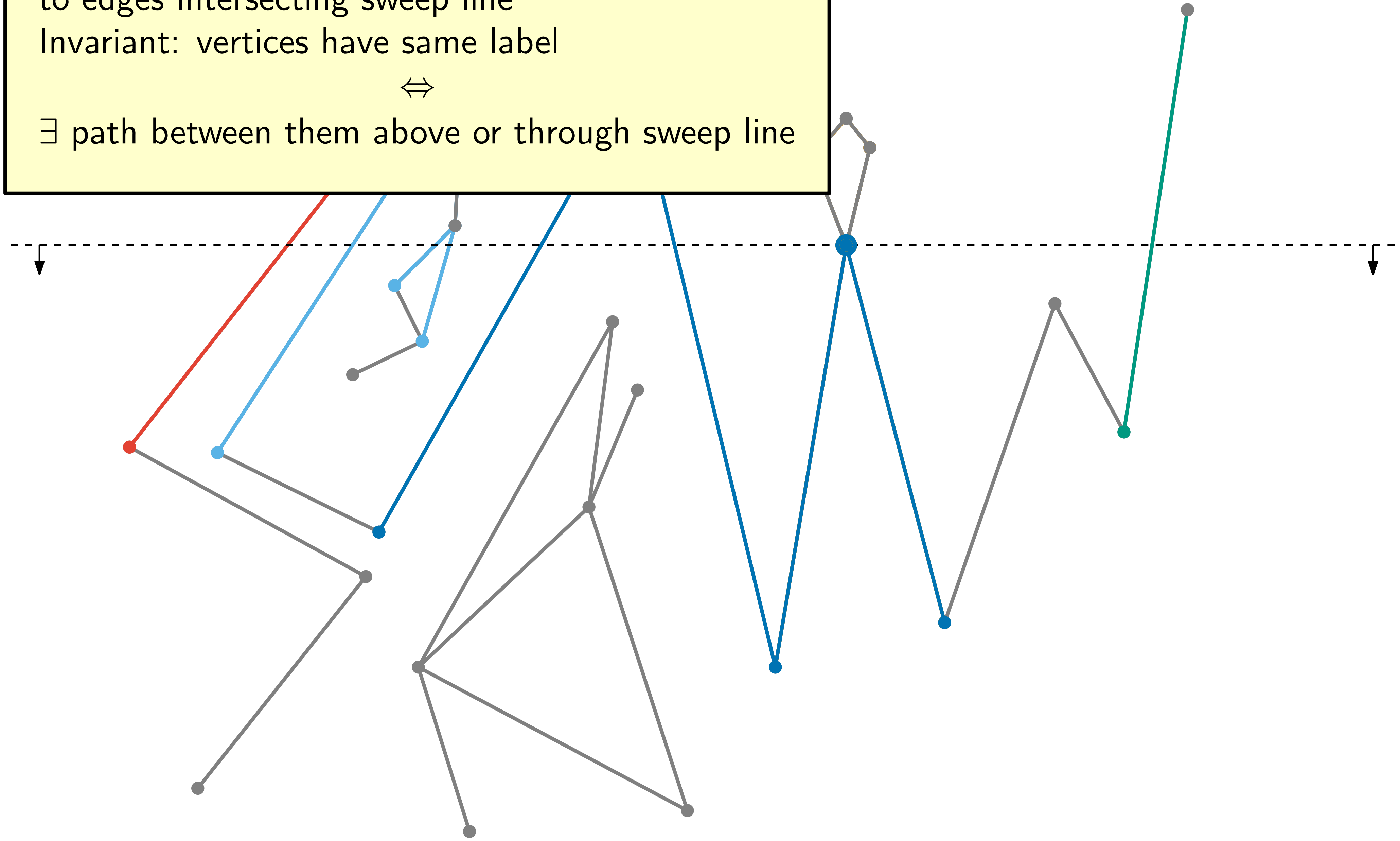


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

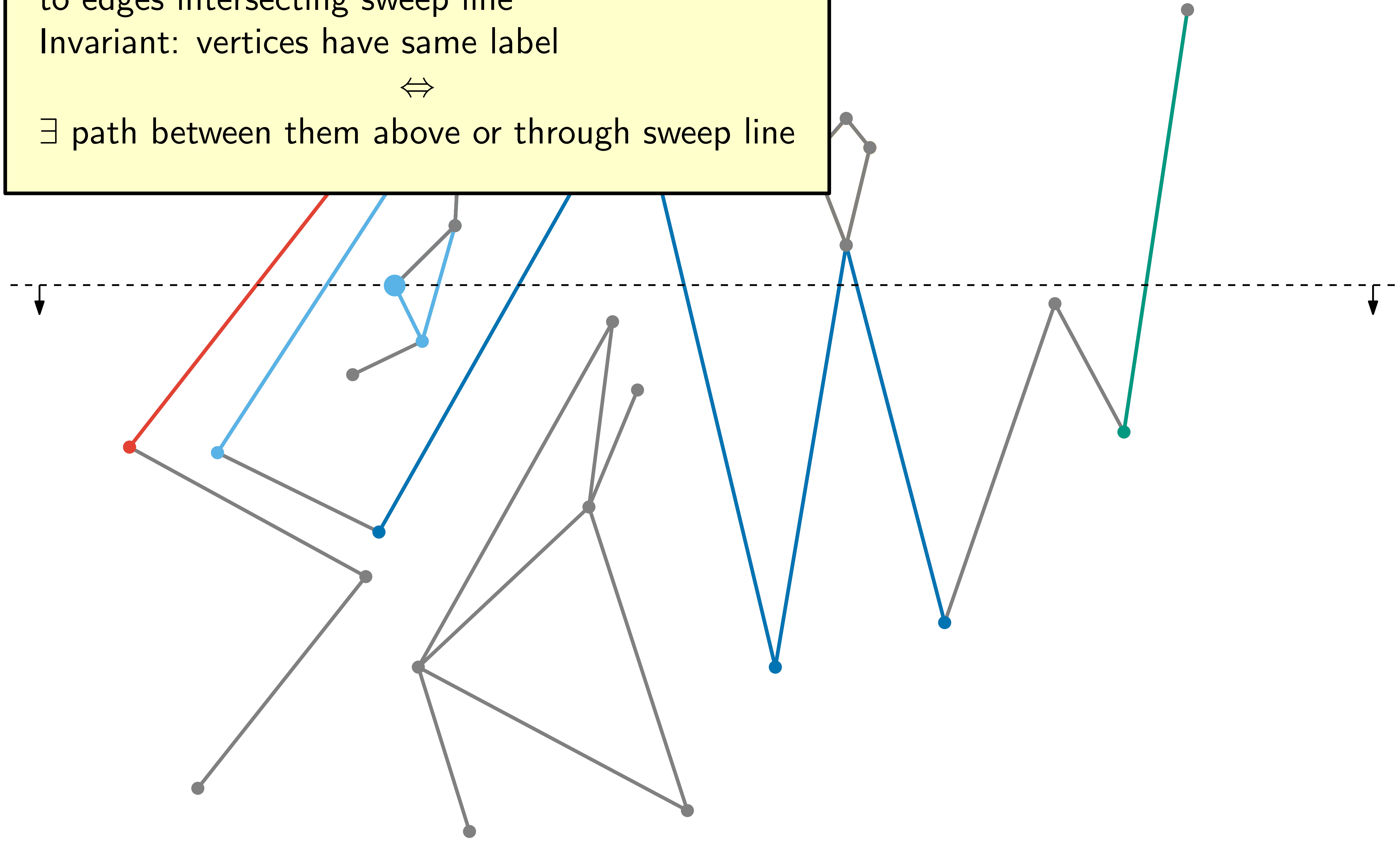


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line

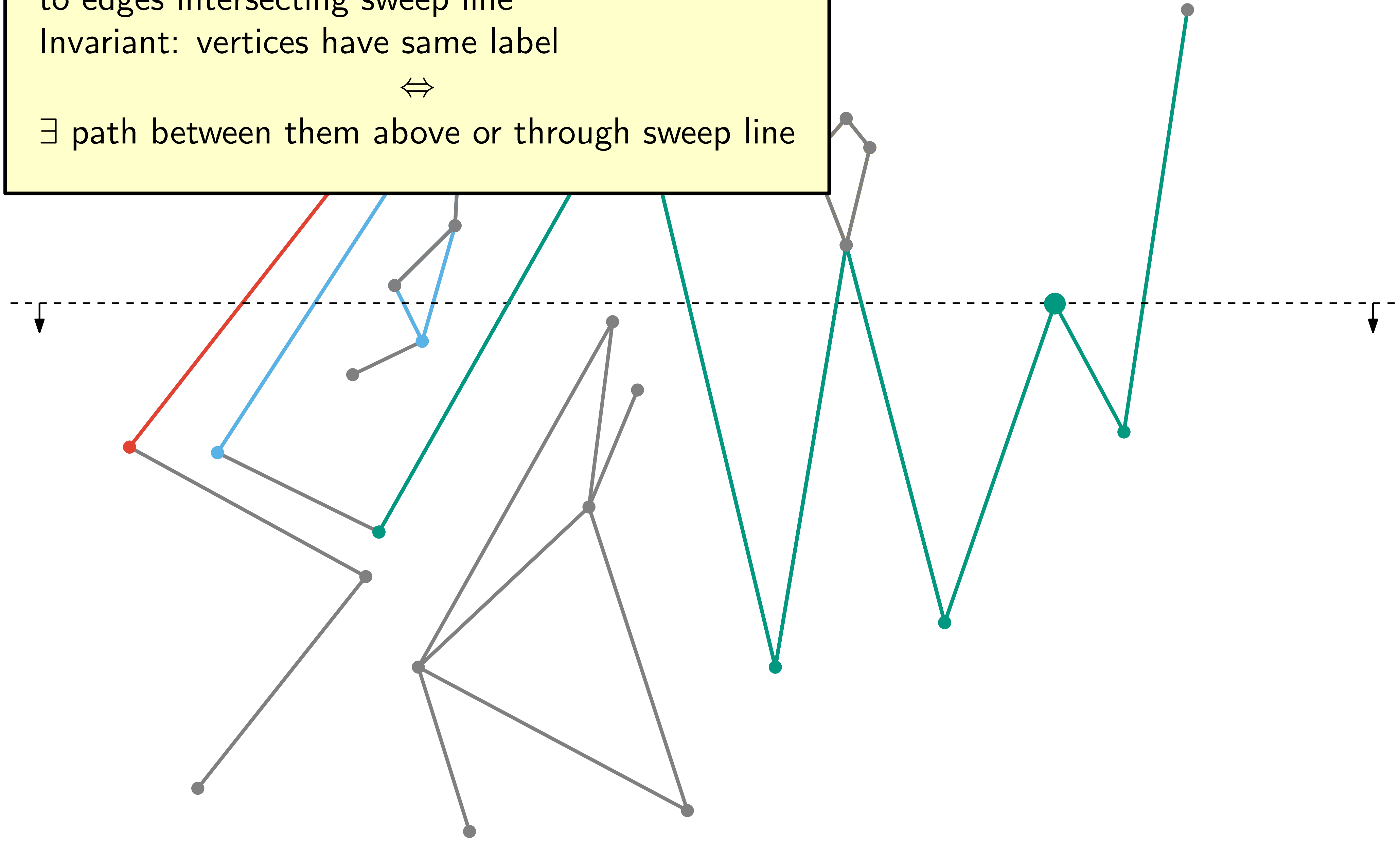


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

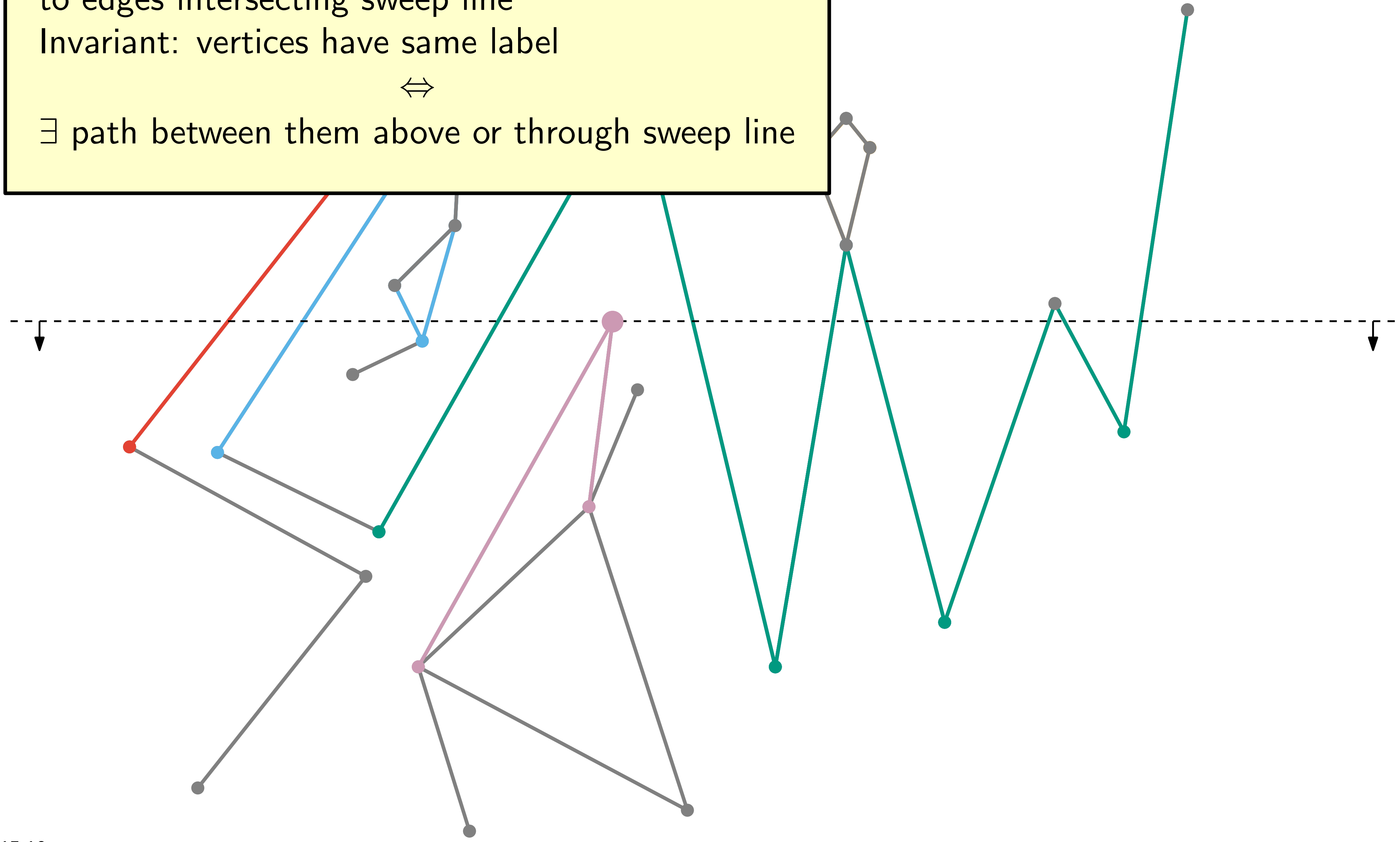


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line

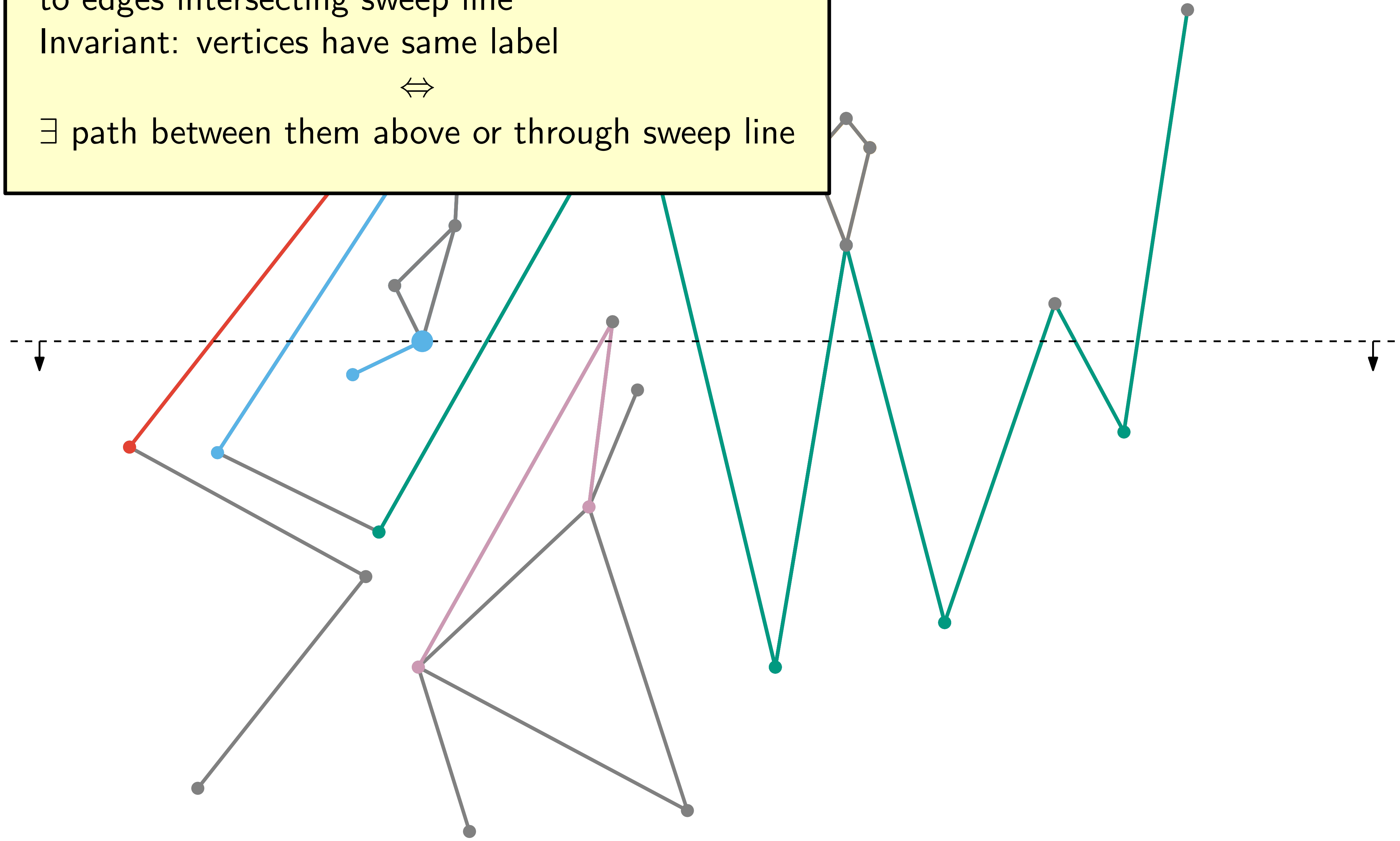


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

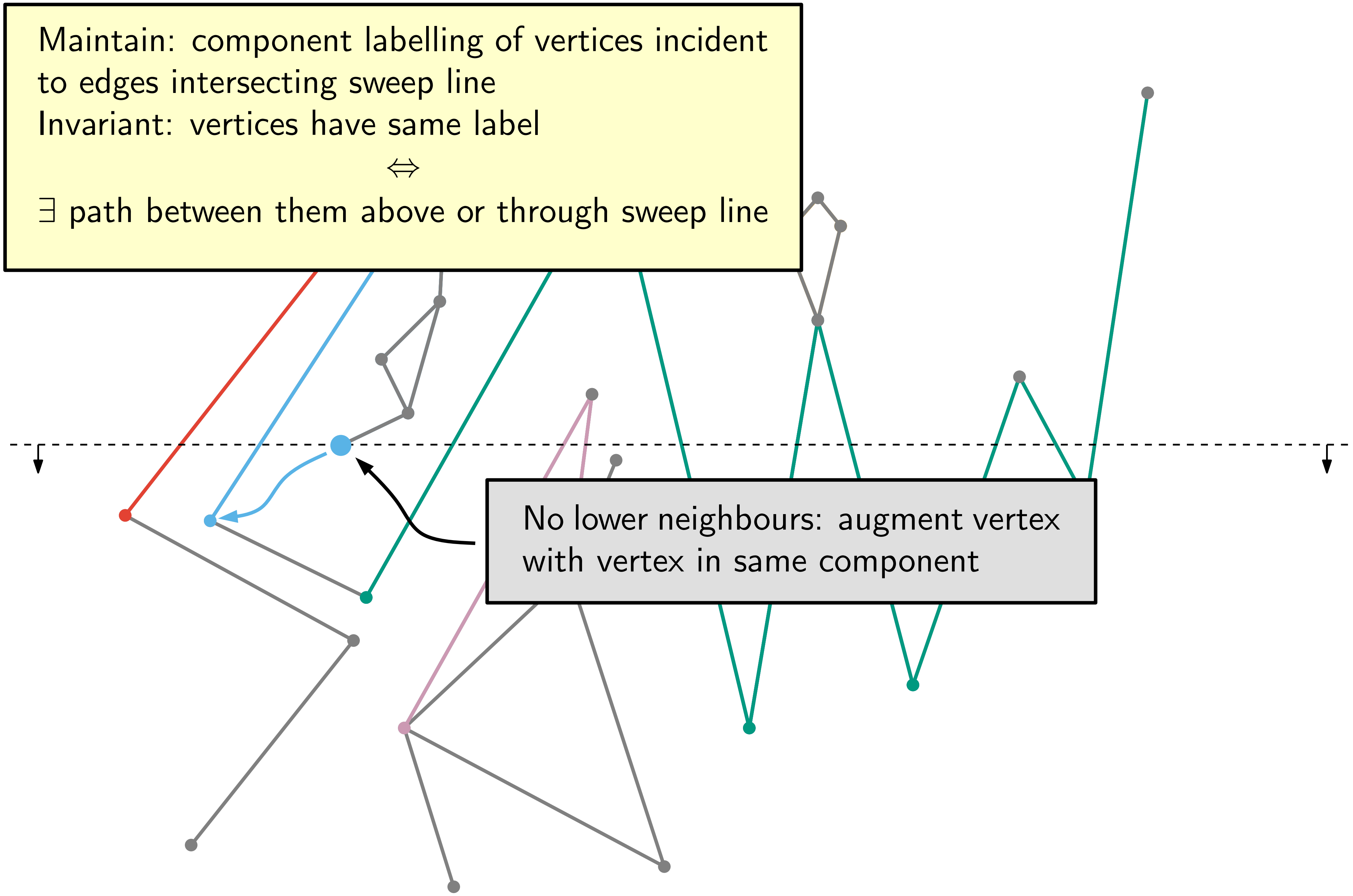


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line



No lower neighbours: augment vertex with vertex in same component

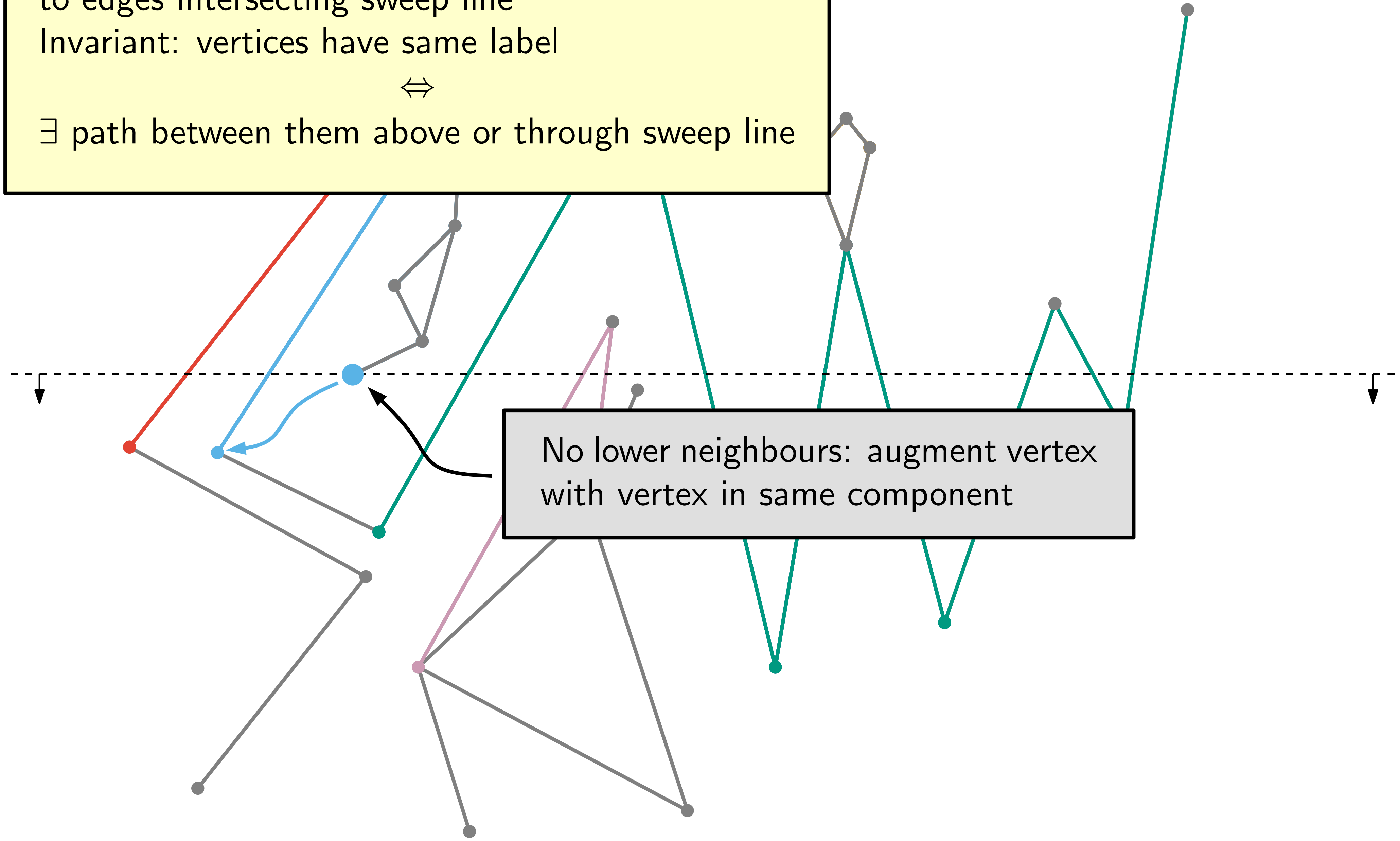
Connected component algorithm

down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line



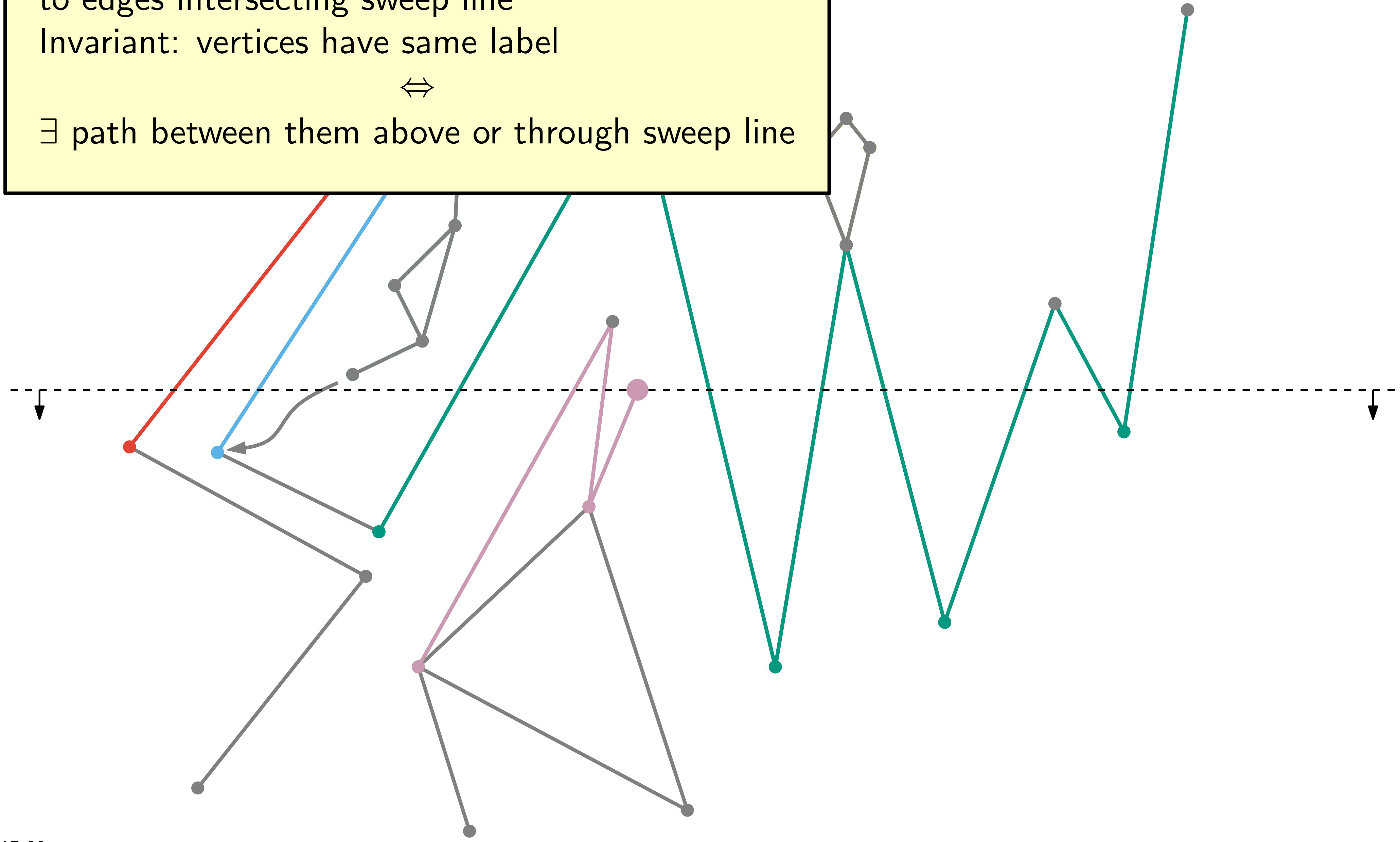
No lower neighbours: augment vertex with vertex in same component

Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line

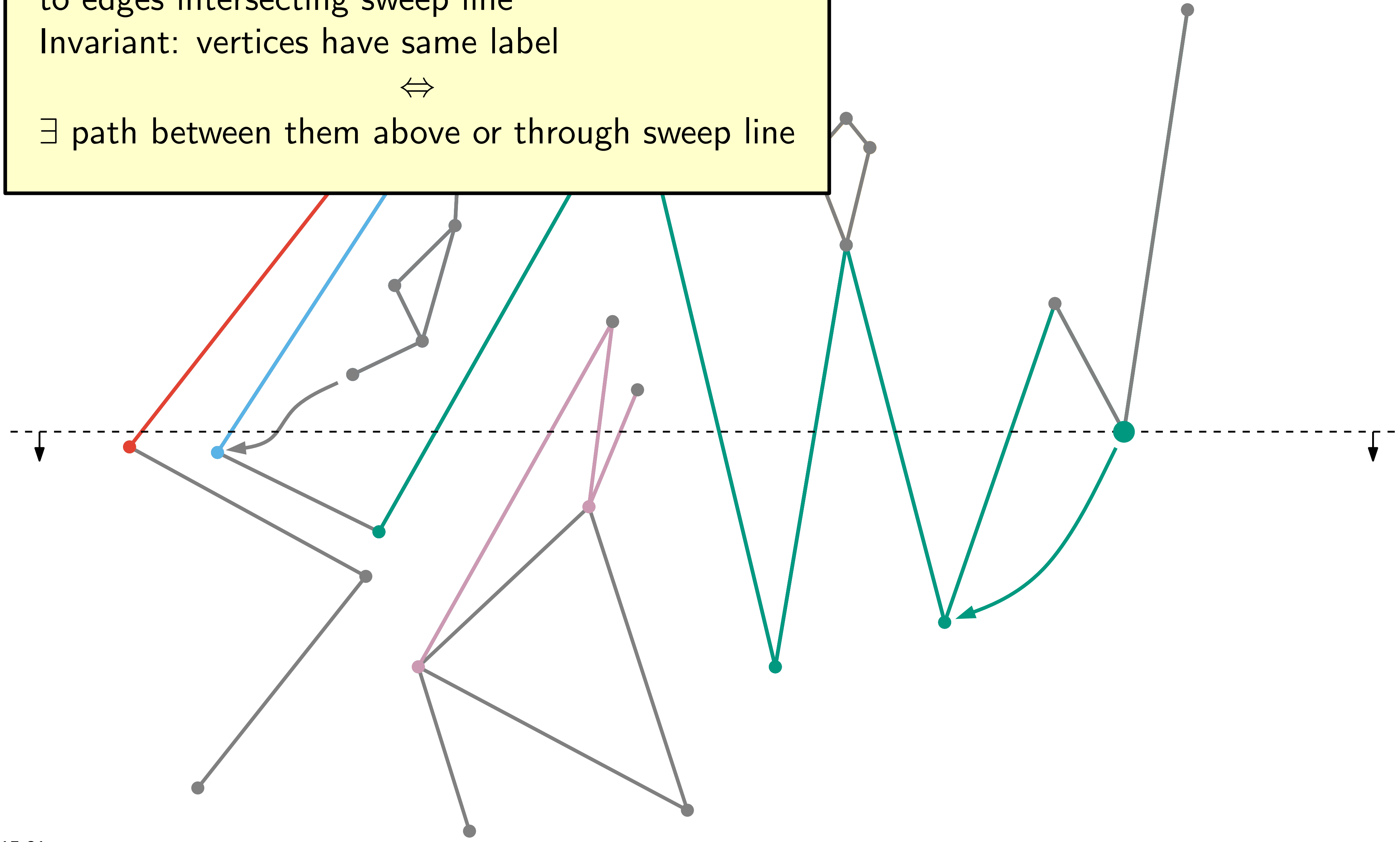


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

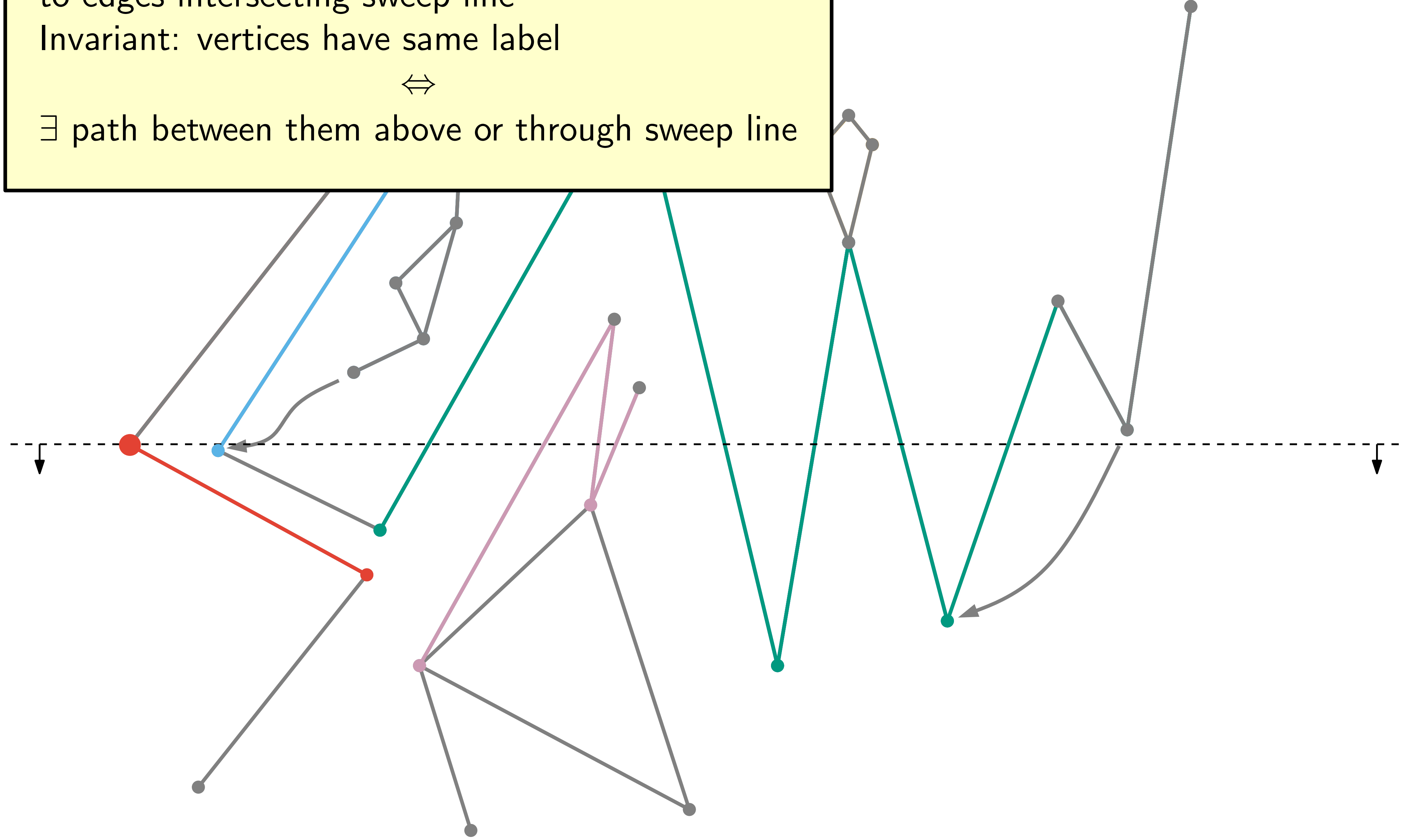


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

$\exists$ path between them above or through sweep line

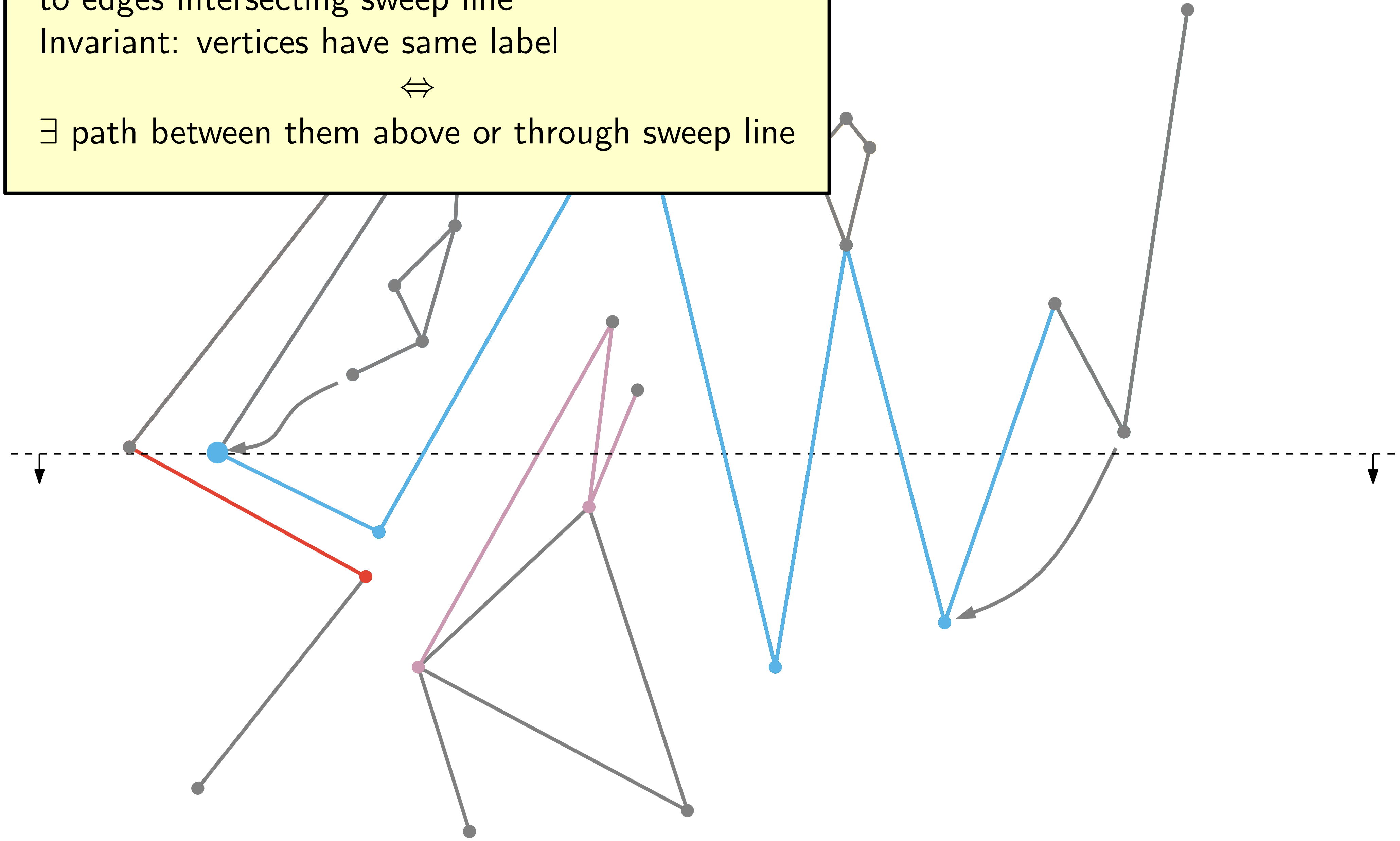


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

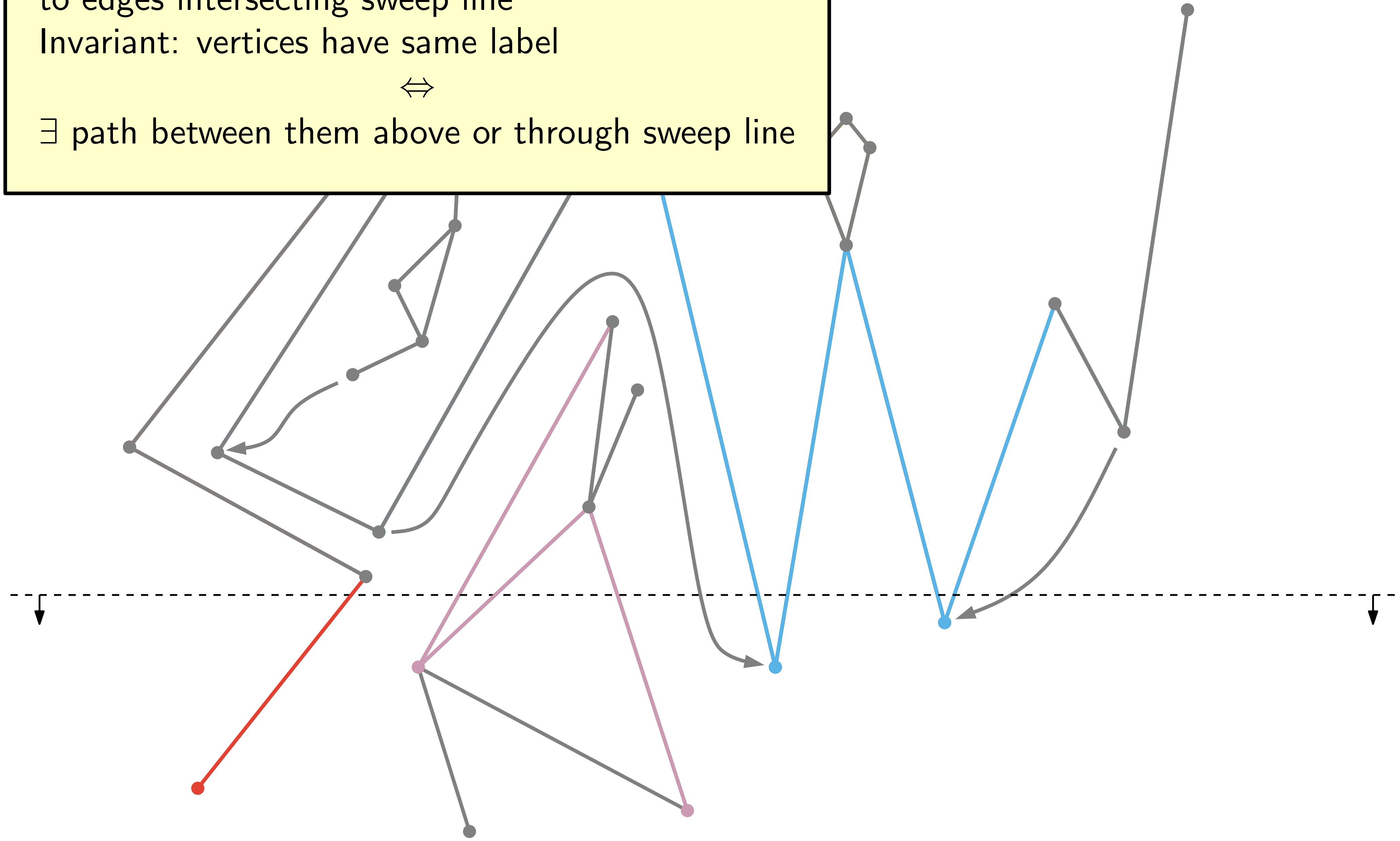


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$$\Leftrightarrow$$

$\exists$ path between them above or through sweep line

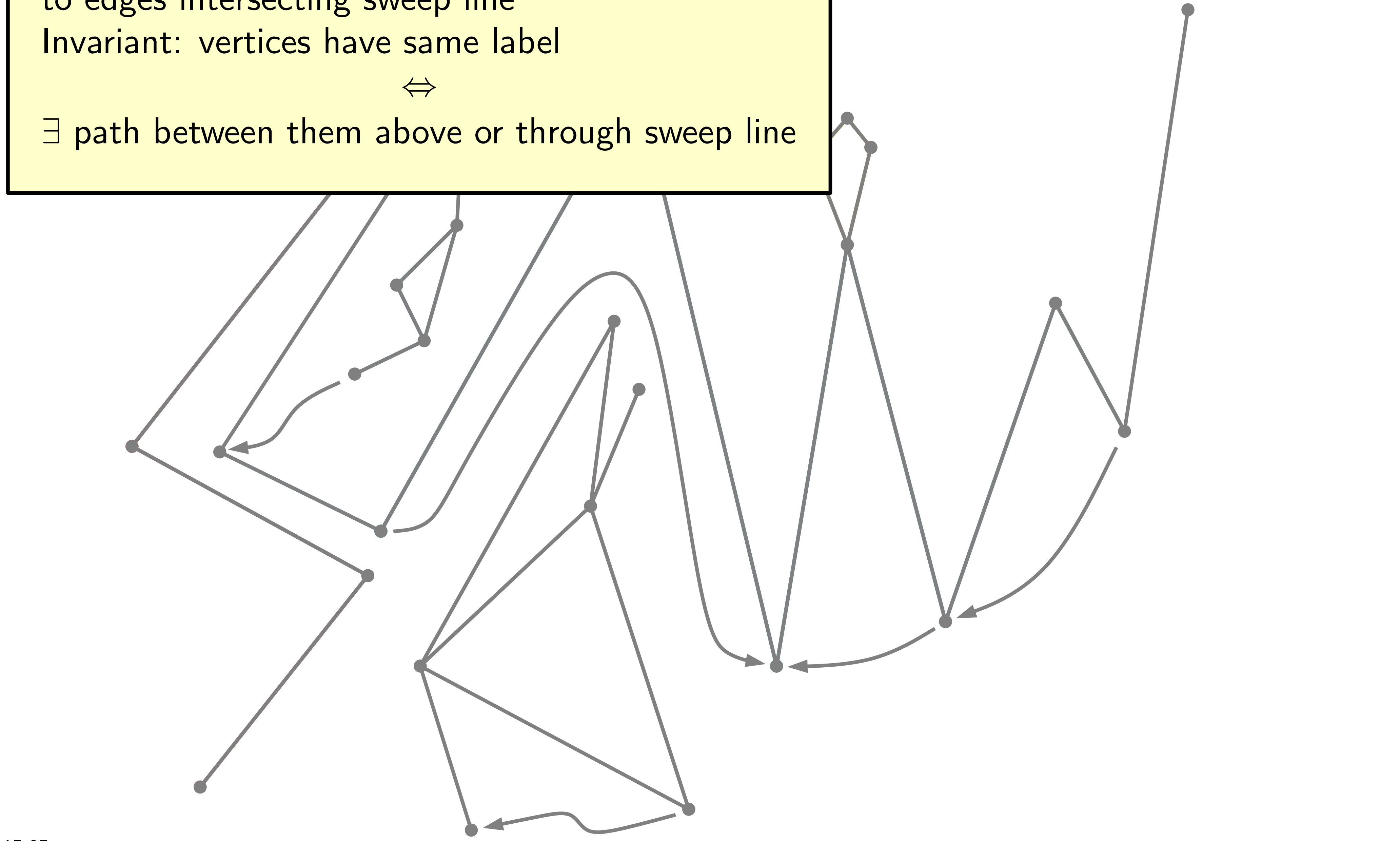


Connected component algorithm
down phase

Maintain: component labelling of vertices incident to edges intersecting sweep line
Invariant: vertices have same label

$\Leftrightarrow$

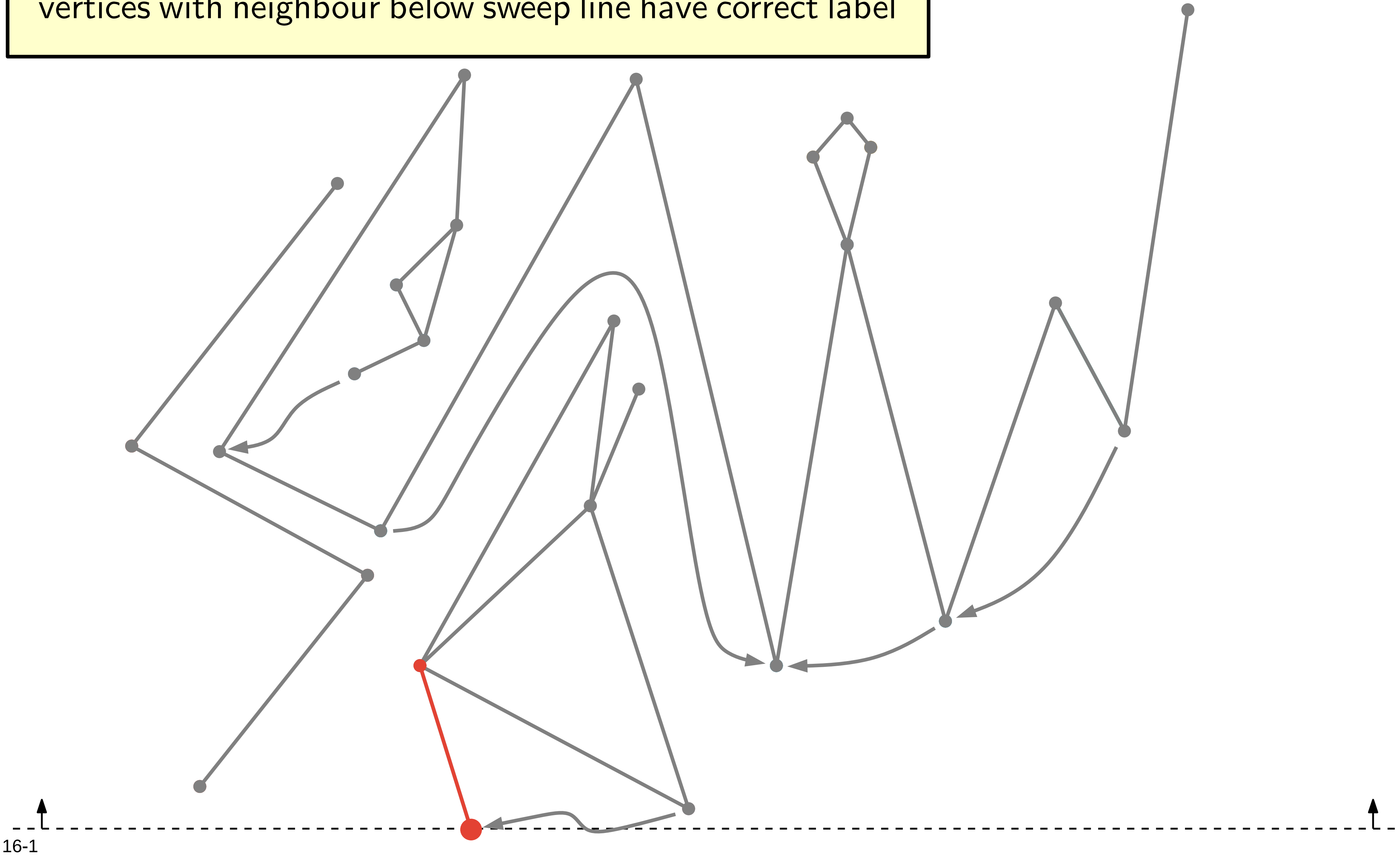
$\exists$ path between them above or through sweep line



Connected component algorithm

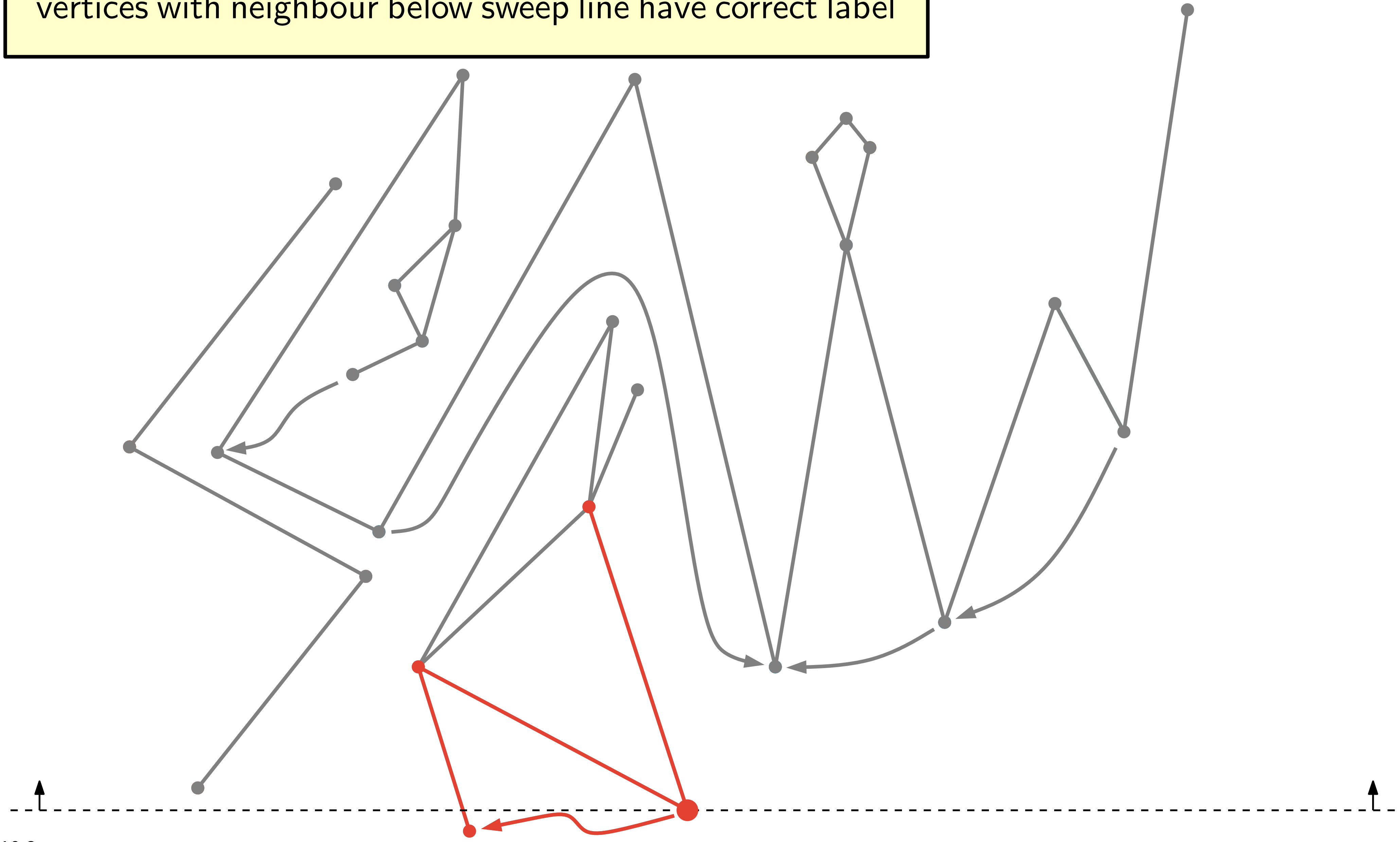
up phase

Invariant:
vertices with neighbour below sweep line have correct label



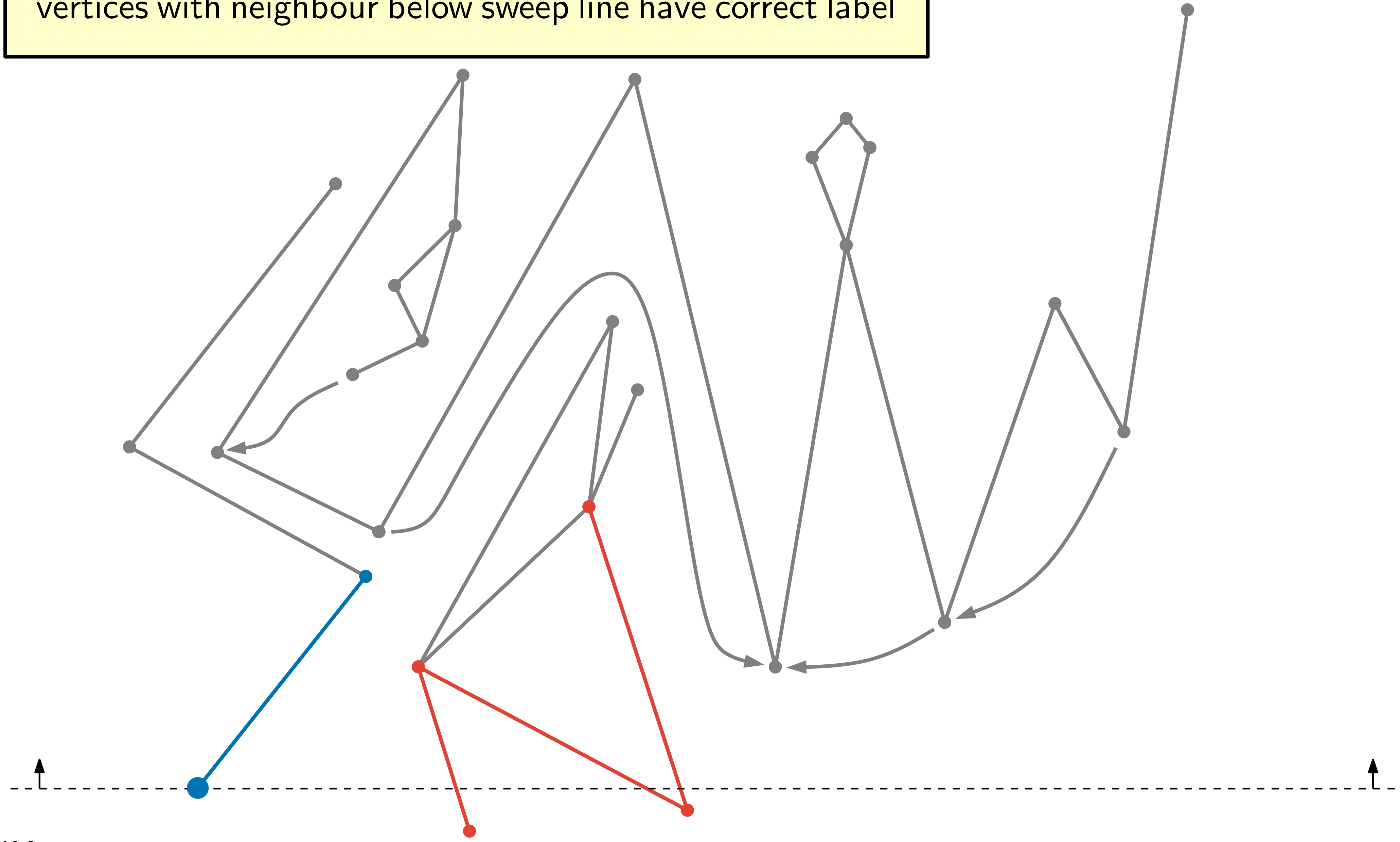
Connected component algorithm
up phase

Invariant:
vertices with neighbour below sweep line have correct label



Connected component algorithm
up phase

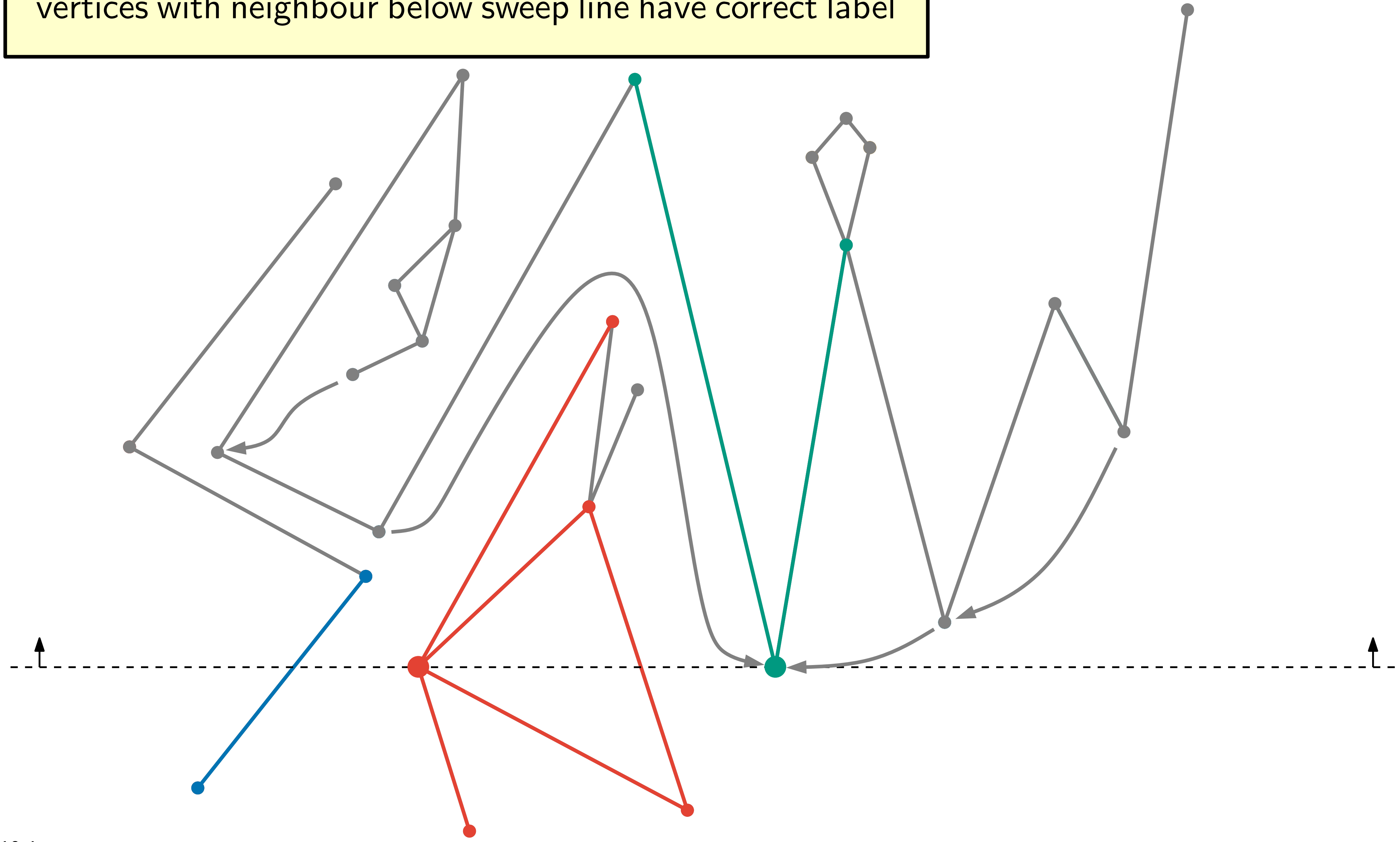
Invariant:
vertices with neighbour below sweep line have correct label



Connected component algorithm

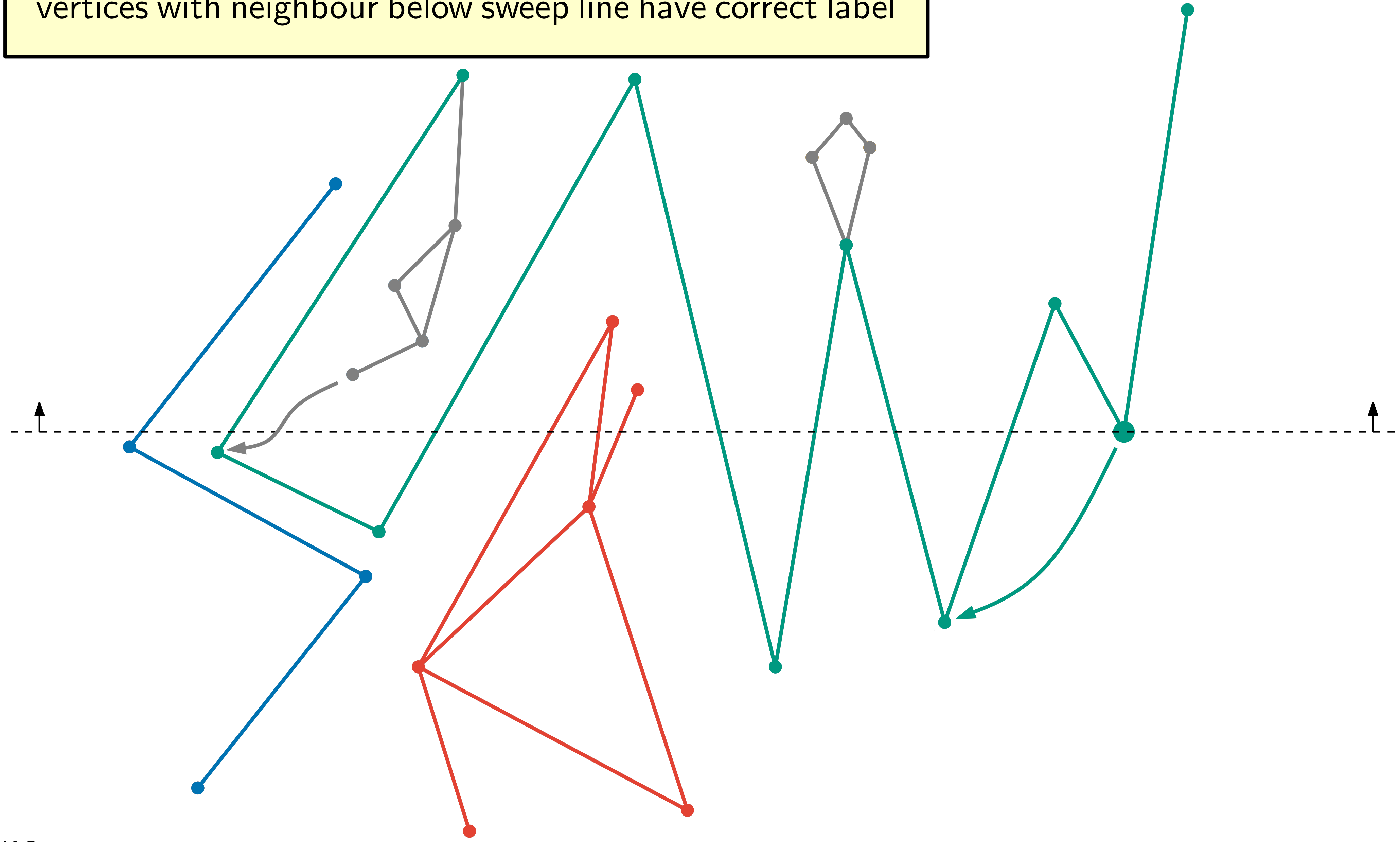
up phase

Invariant:
vertices with neighbour below sweep line have correct label



Connected component algorithm
up phase

Invariant:
vertices with neighbour below sweep line have correct label



Connected component algorithm

up phase

Invariant:
vertices with neighbour below sweep line have correct label