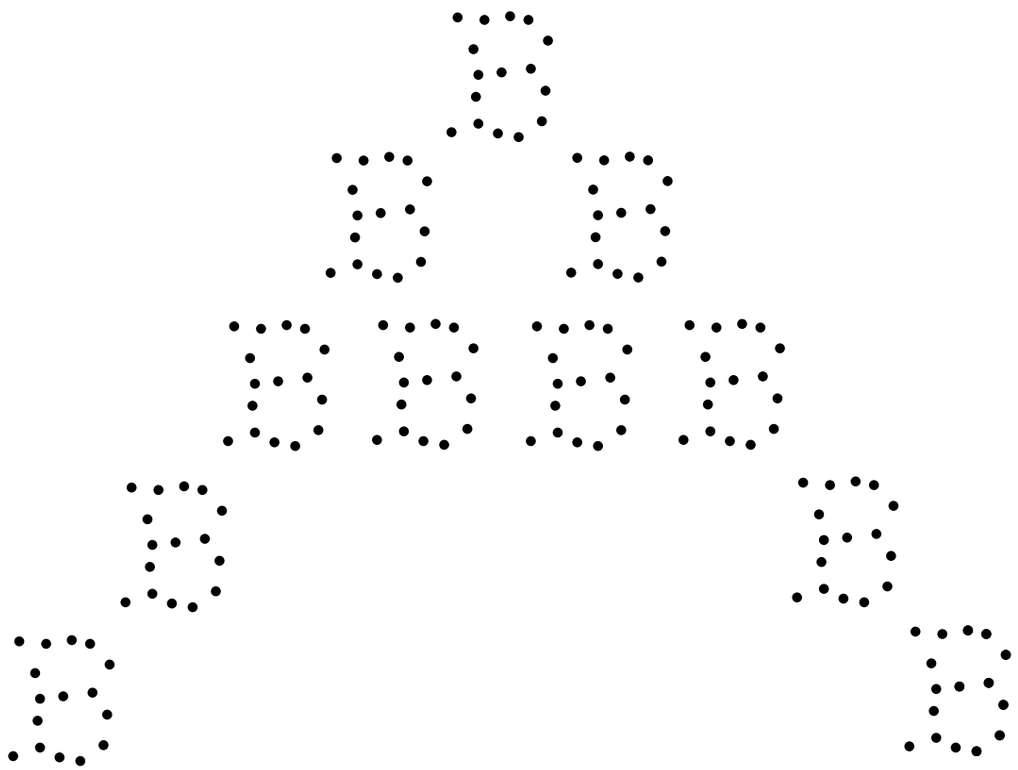
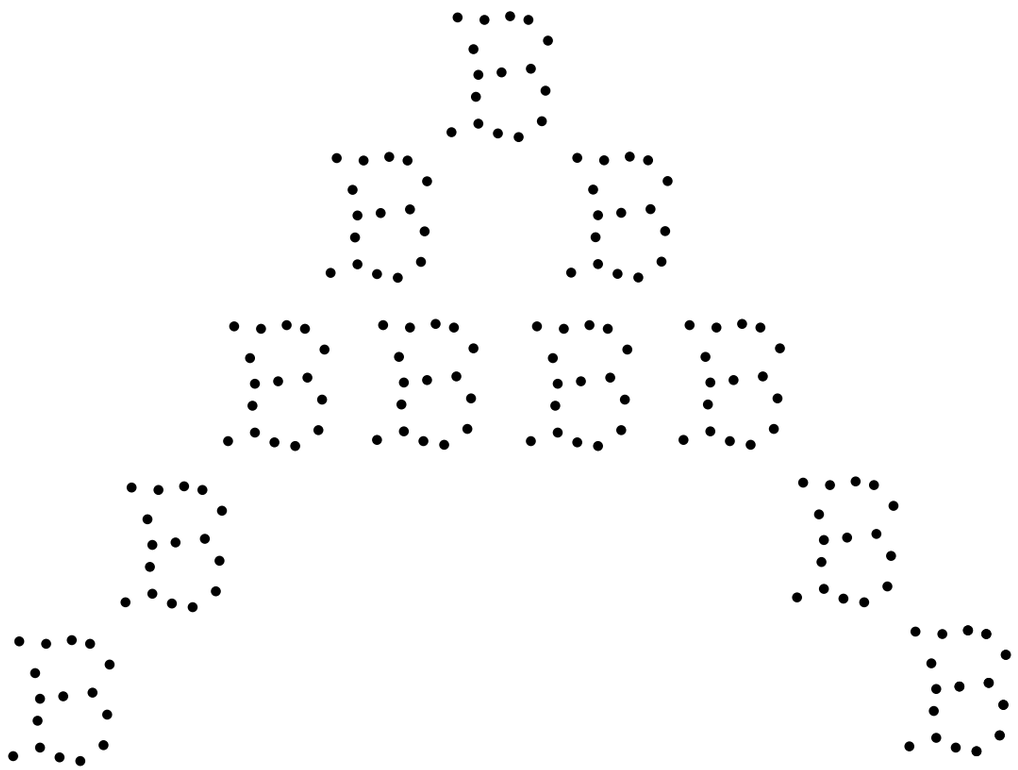
Problems with homology



First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Problems with homology

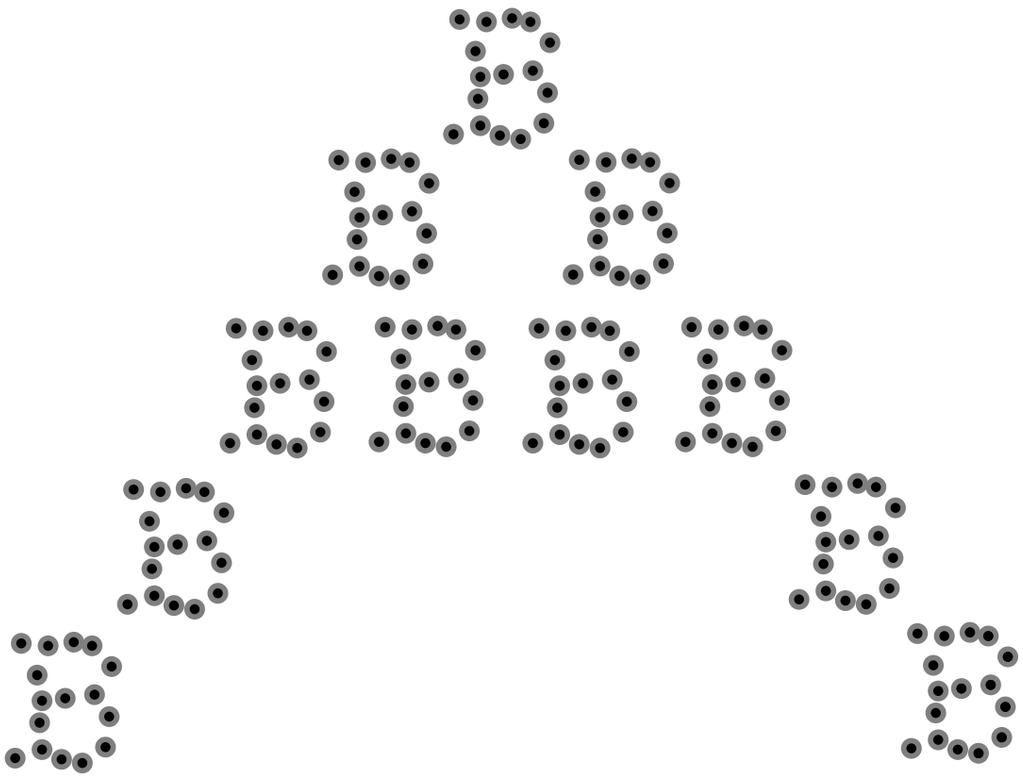


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology

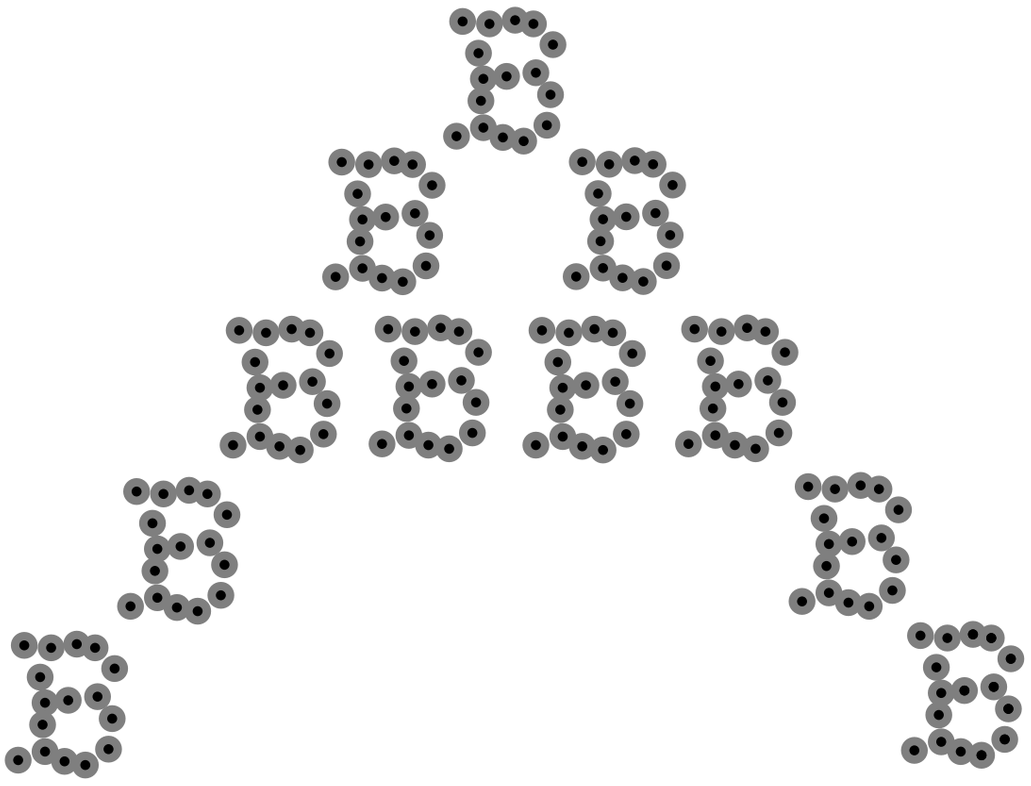


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology

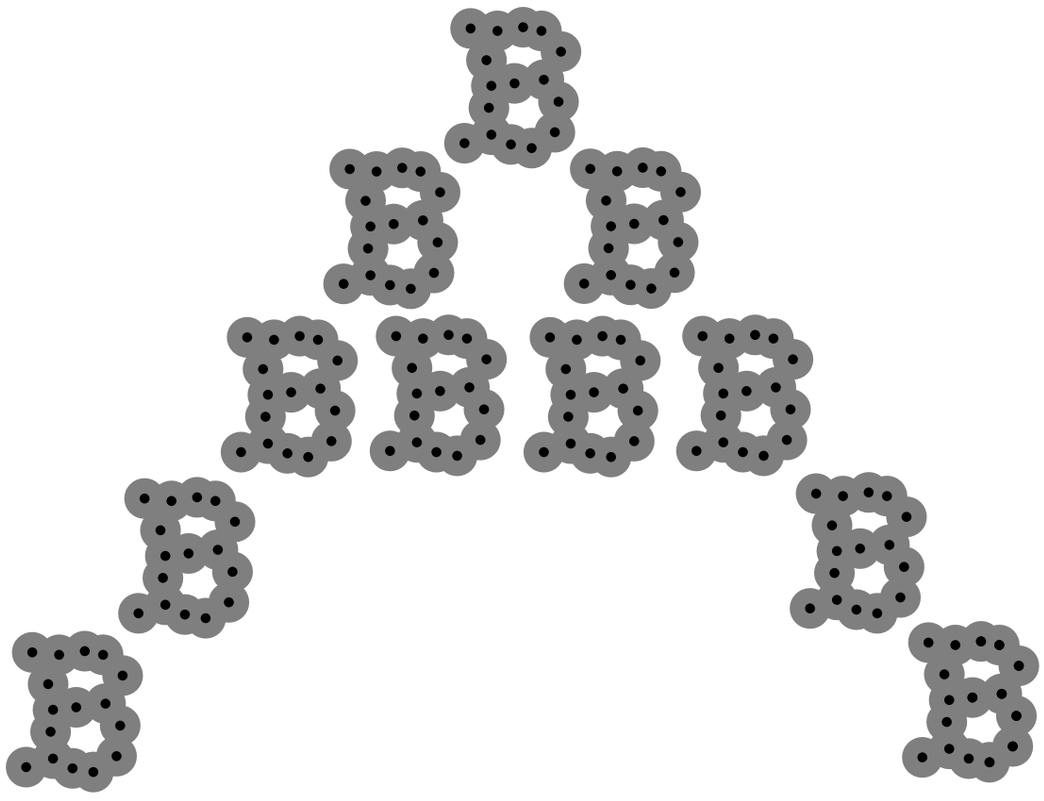


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology

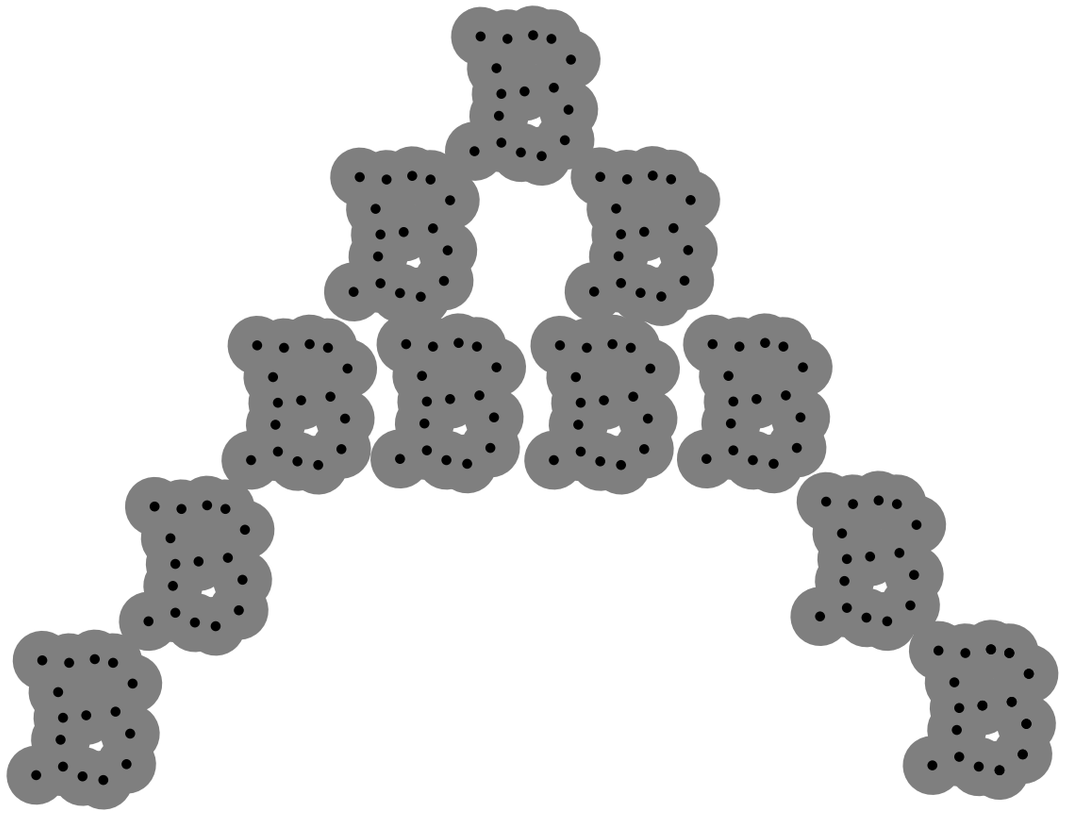


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology

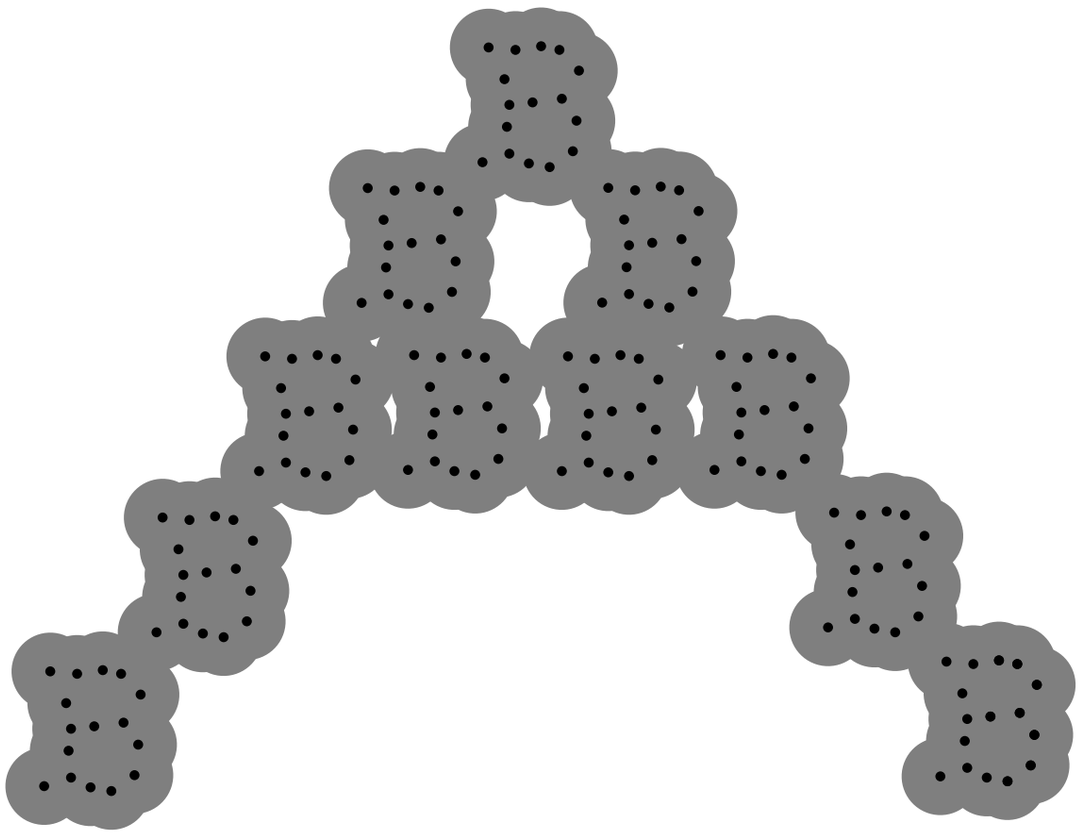


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology

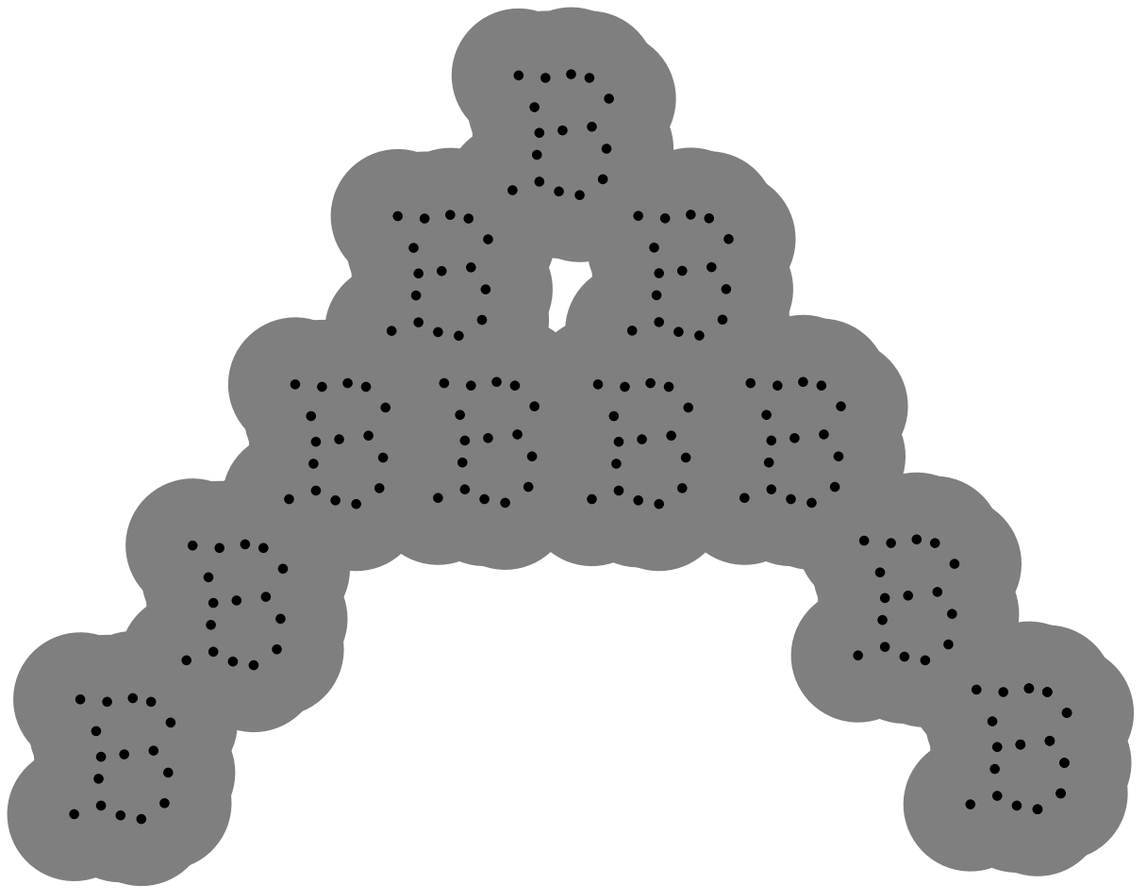


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology

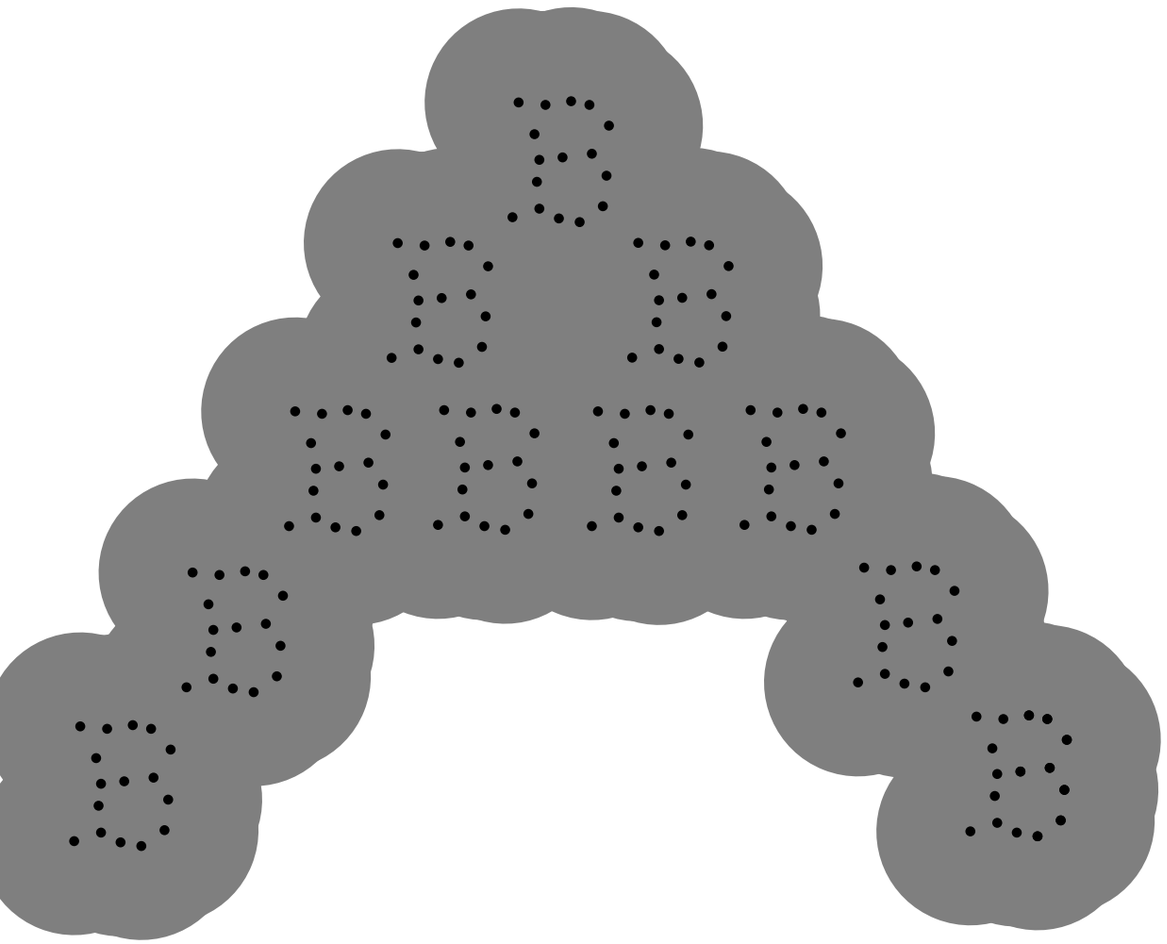


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology

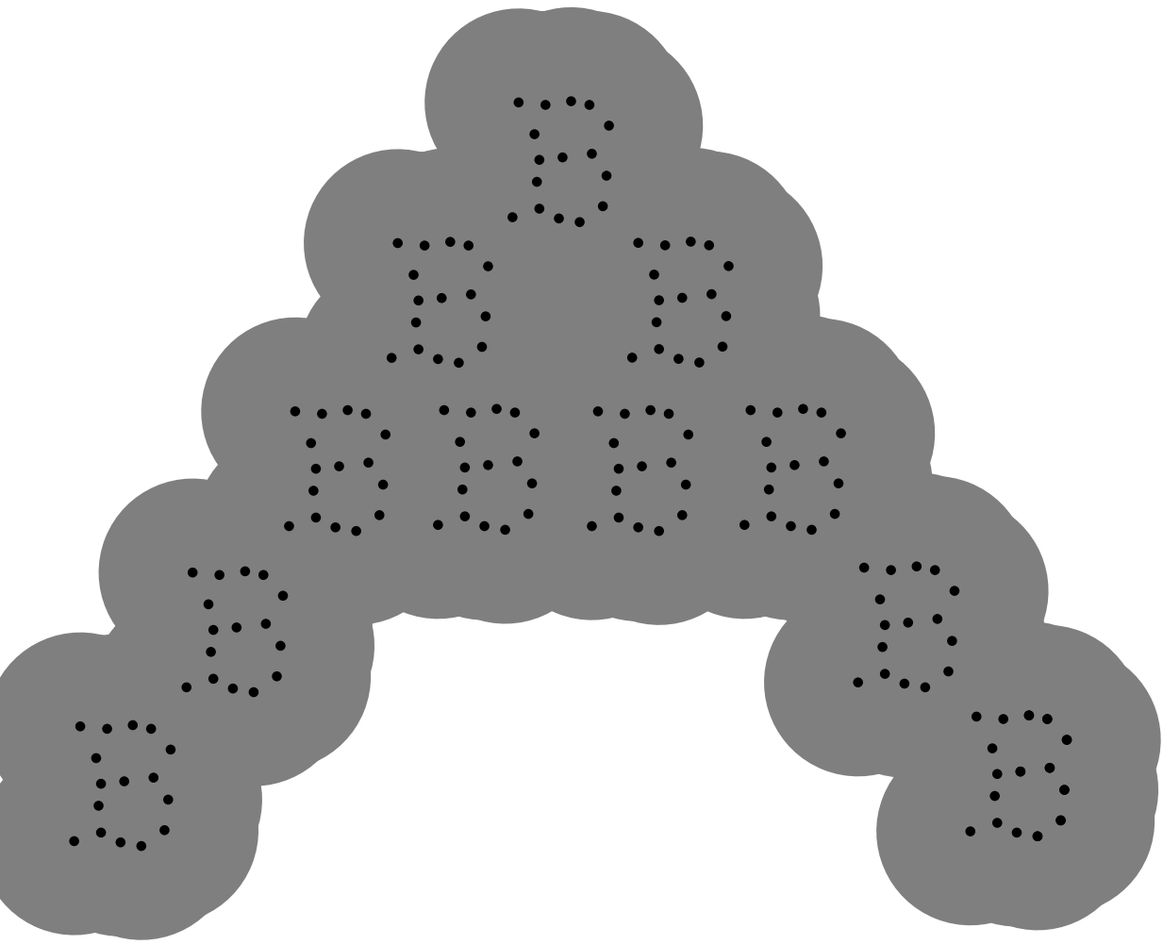


First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

Problems with homology



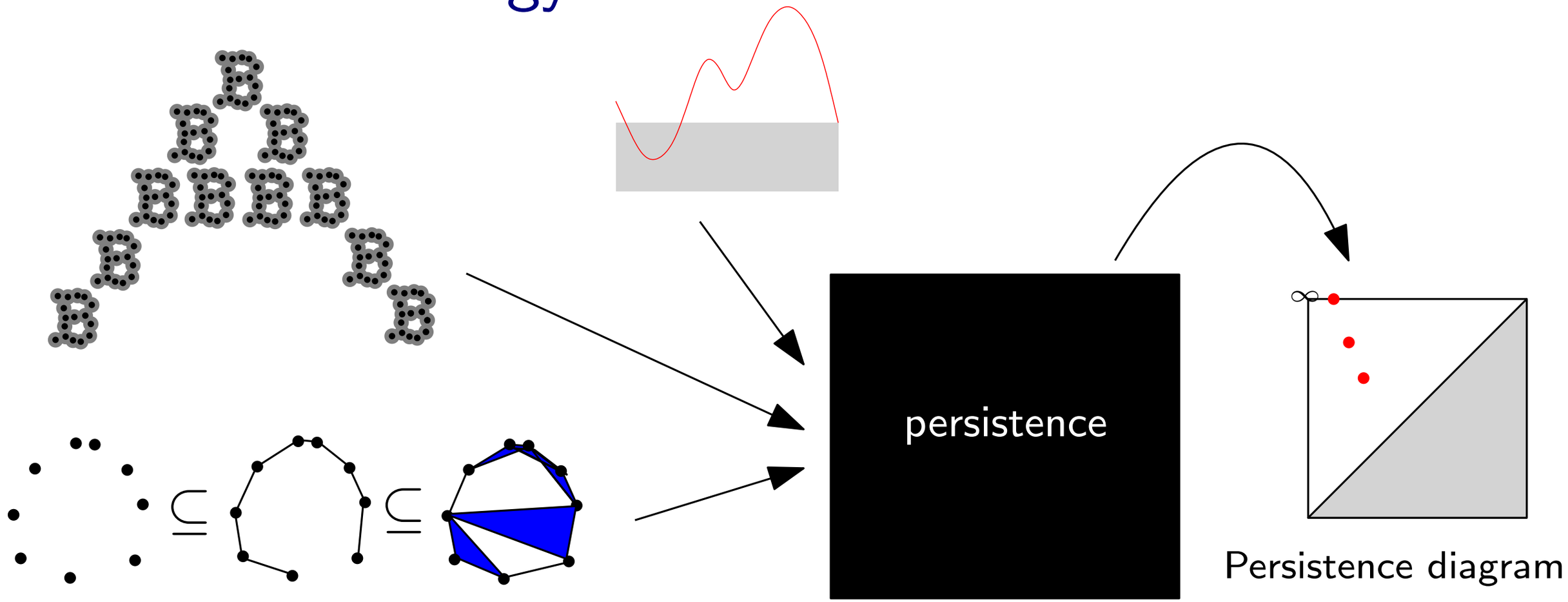
First, the algorithm for computing homology contains much more information than the mere homology of the last complex in the filtration.

Indeed, it contains the homology of *all* the subcomplexes in the filtration.

This is very interesting in the sense that data can be analyzed at *multiple* scales.

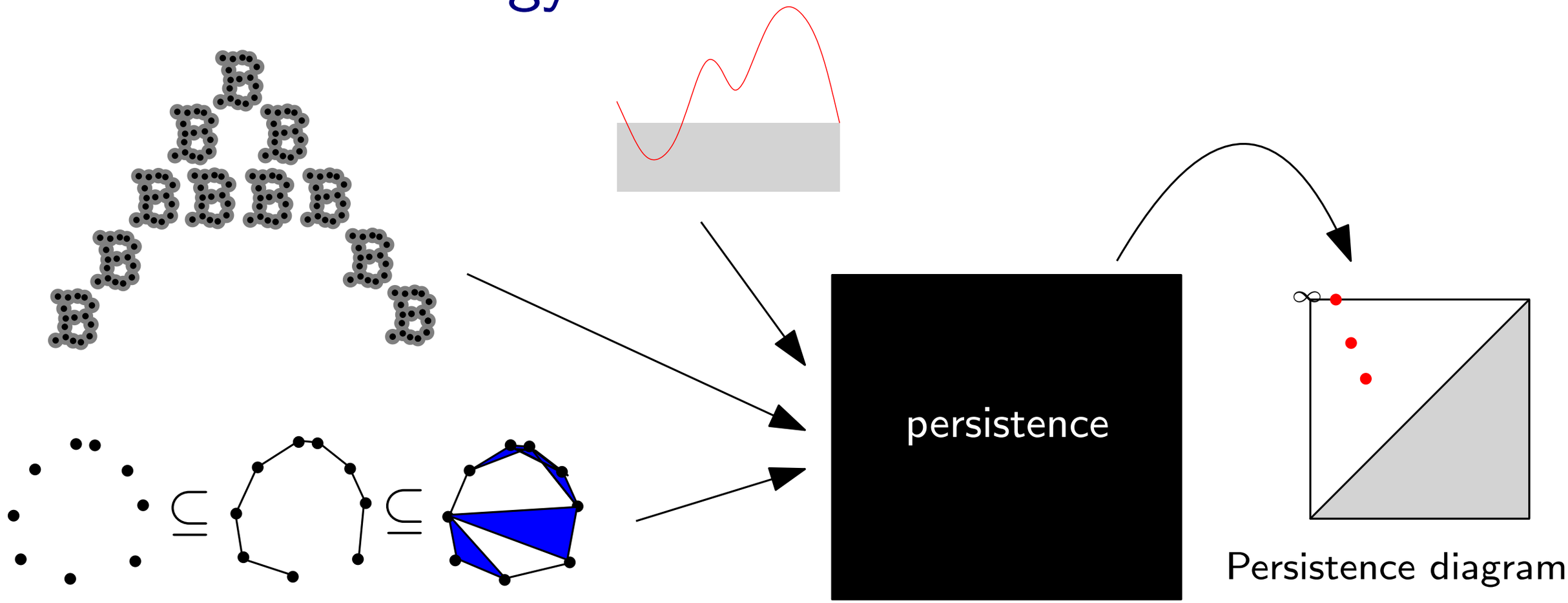
Persistent homology aims at encoding the homology of the complex at *all possible scales* into a compact descriptor.

Persistent homology



What is persistent homology?

Persistent homology

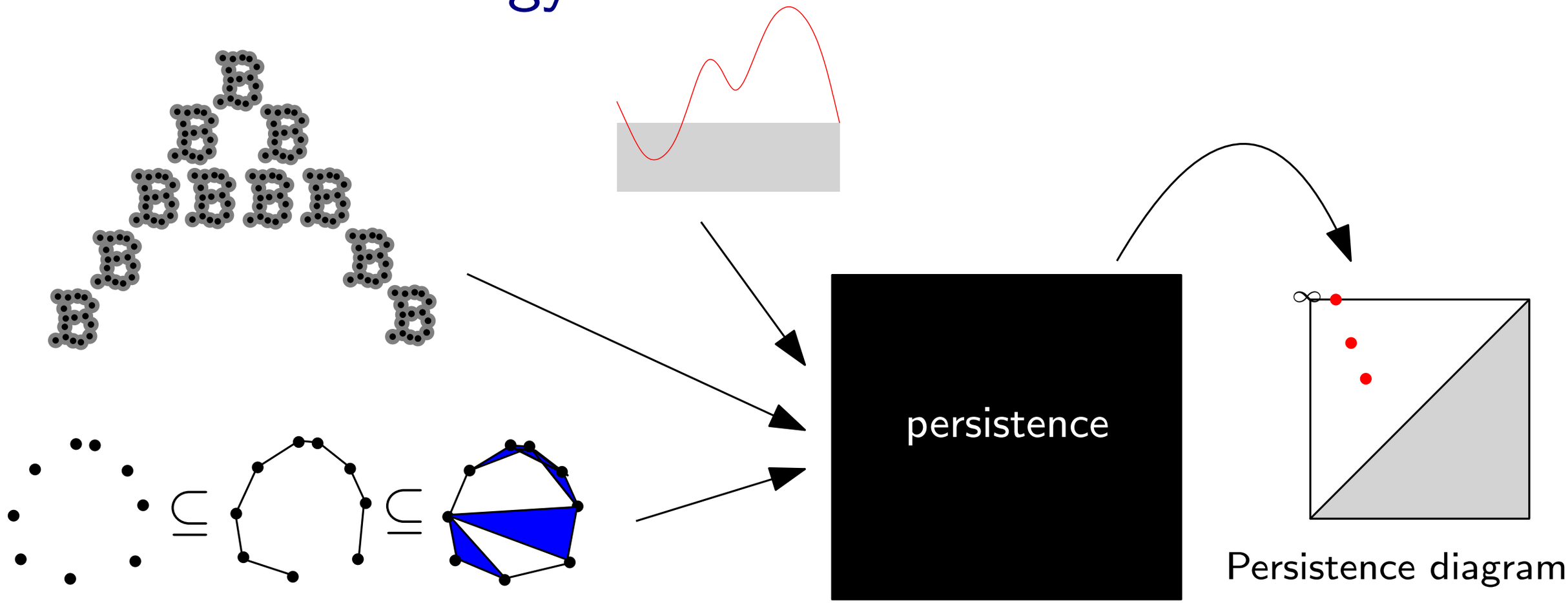


What is persistent homology?

$\rightarrow$ a mathematical framework for encoding the evolution of the homology of filtrations of simplicial complexes (it also works for general filtered spaces).

$\rightarrow$ formalized by H. Edelsbrunner et al. (2002) and G. Carlsson et al. (2005) with wide developments during the last decade.

Persistent homology



What is persistent homology?

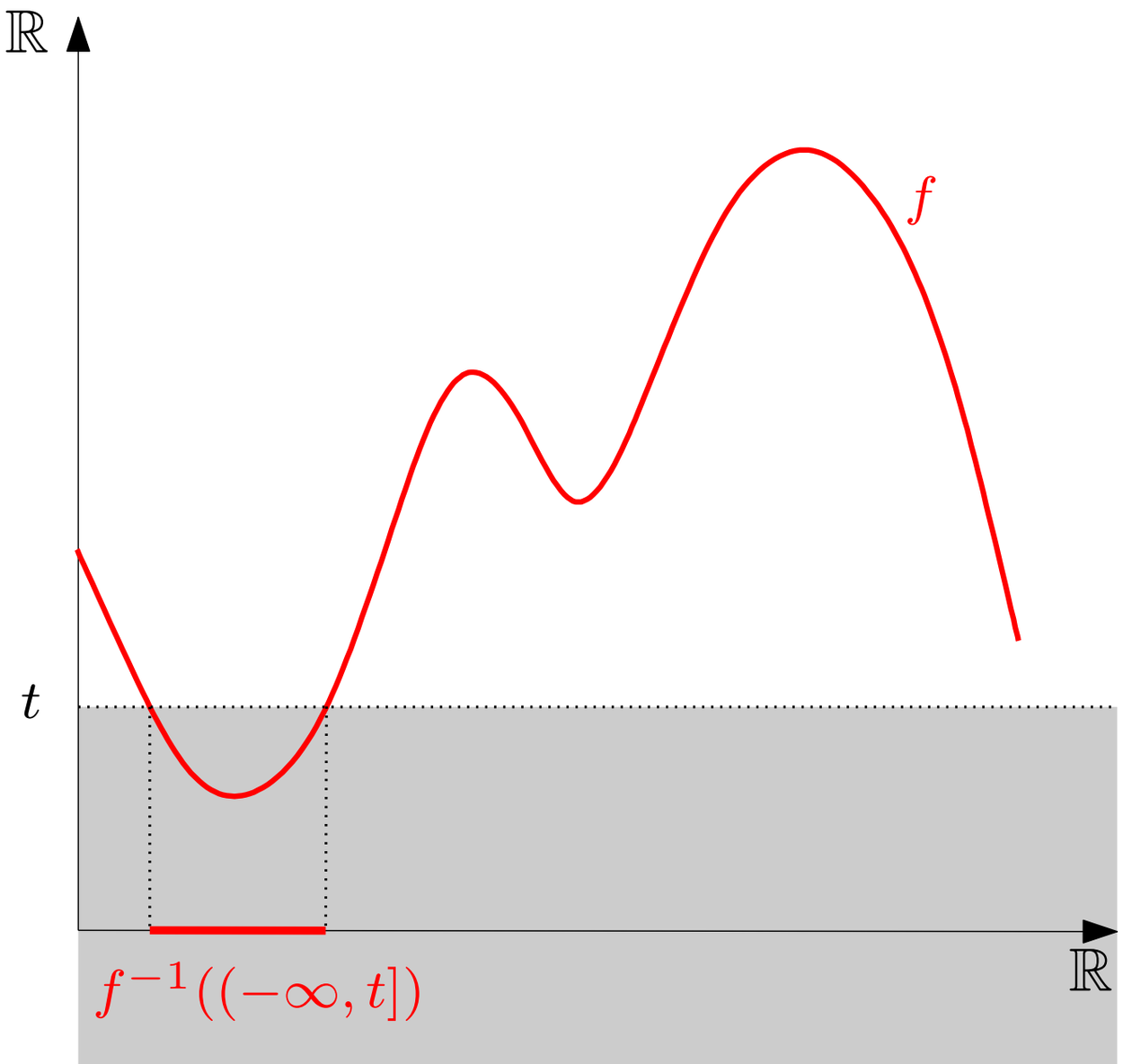
$\rightarrow$ barcodes/persistence diagrams can be efficiently computed.

$\rightarrow$ multiscale topological information.

$\rightarrow$ stability properties.

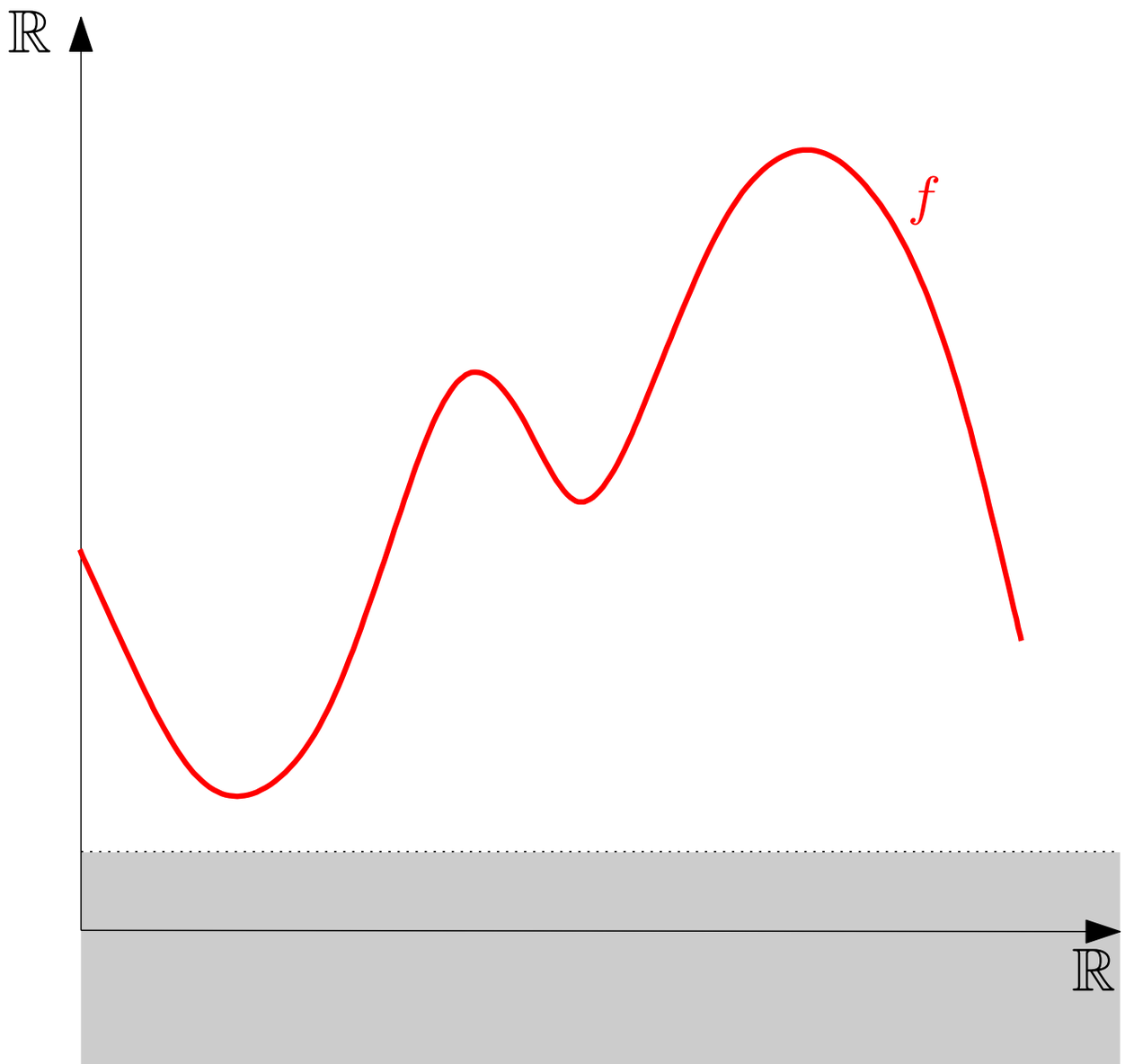
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



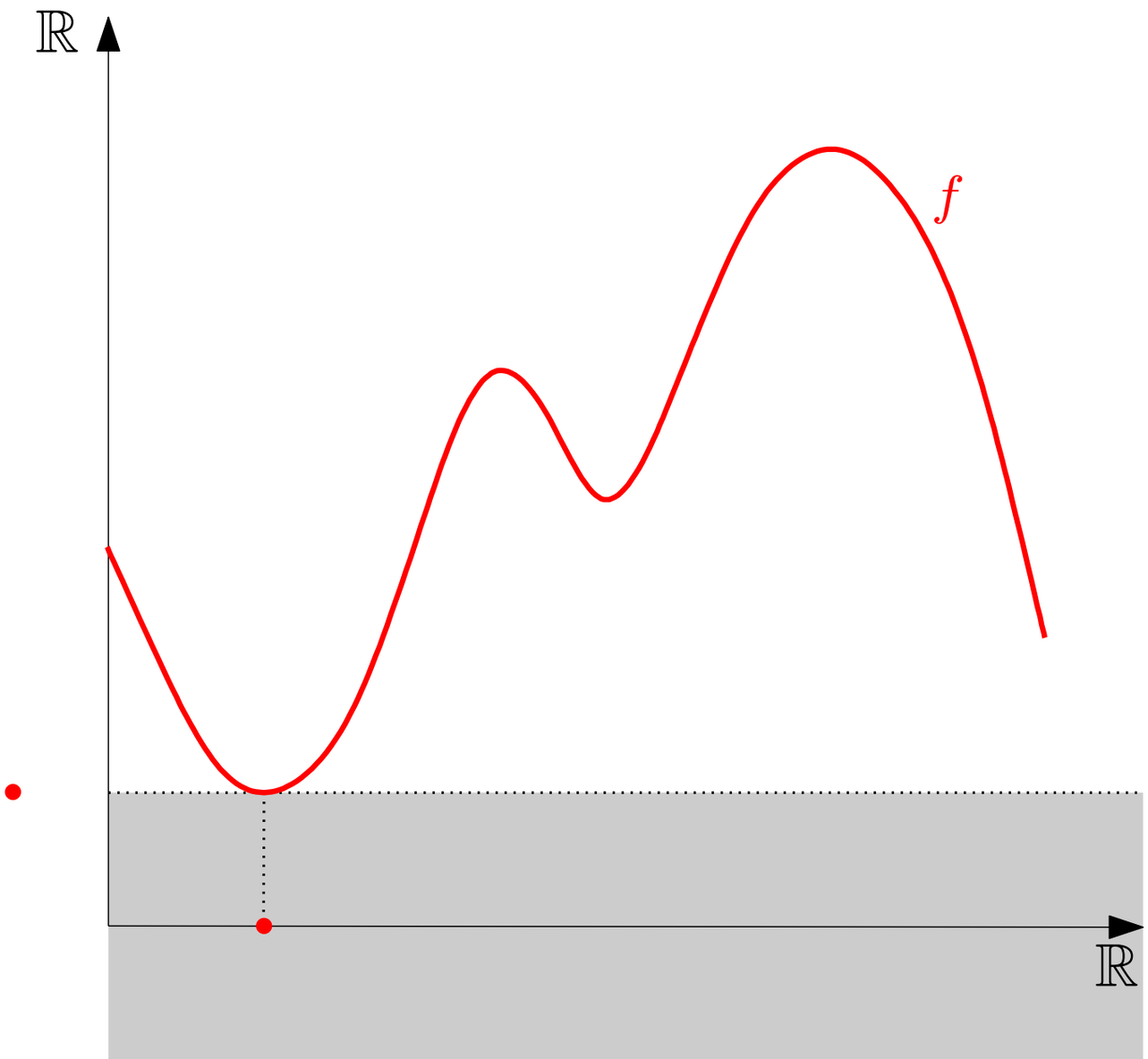
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



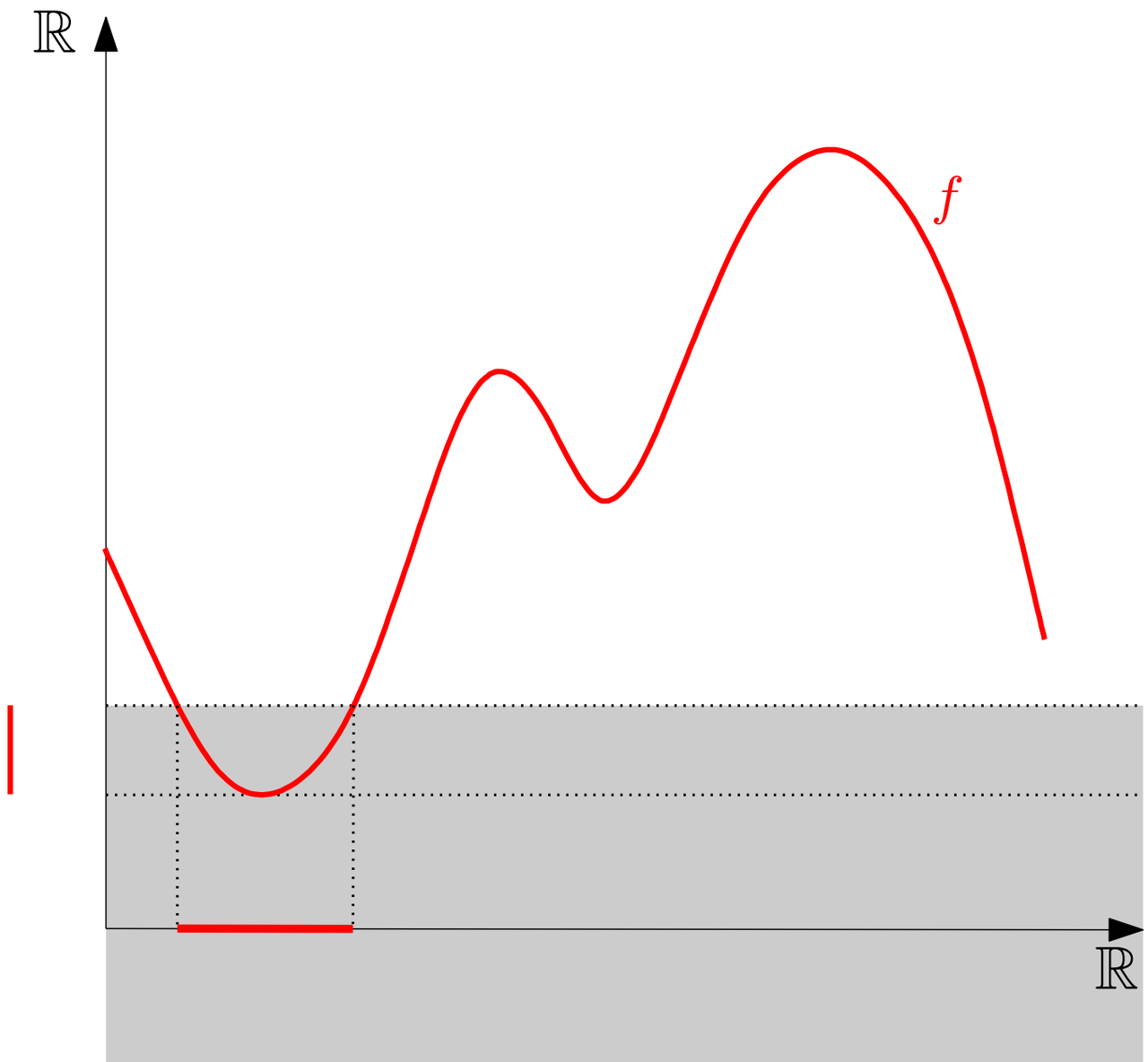
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



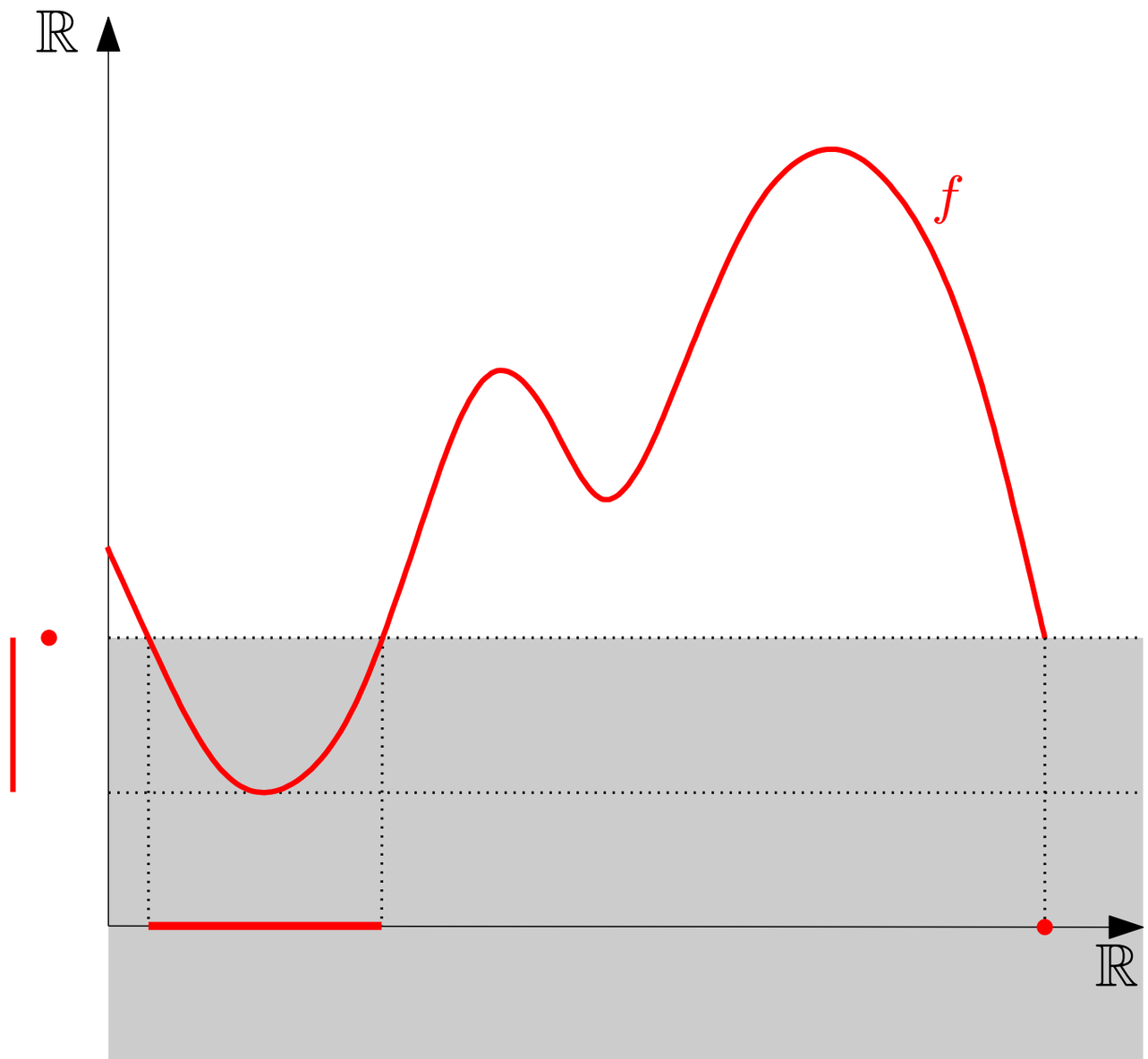
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



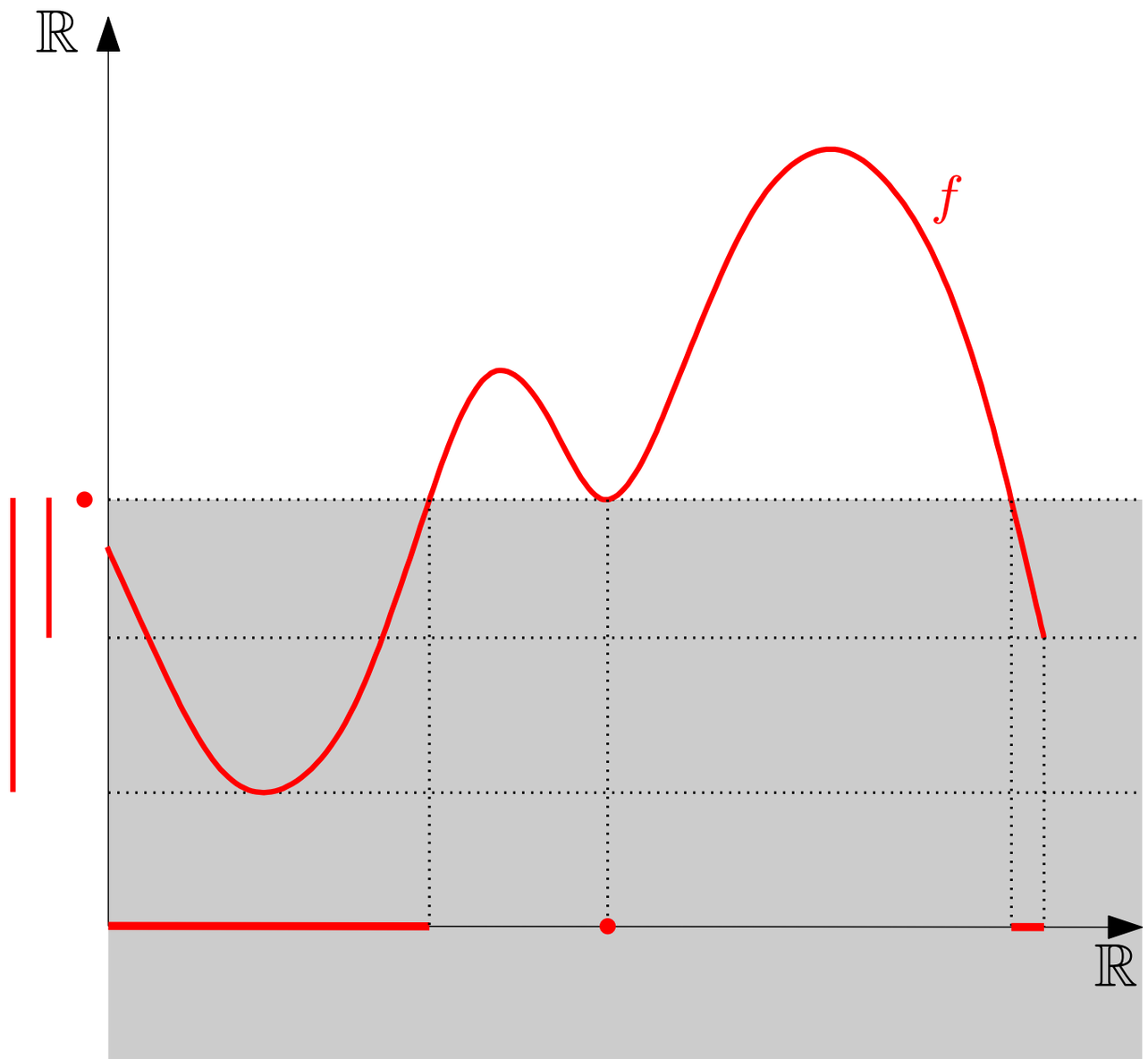
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



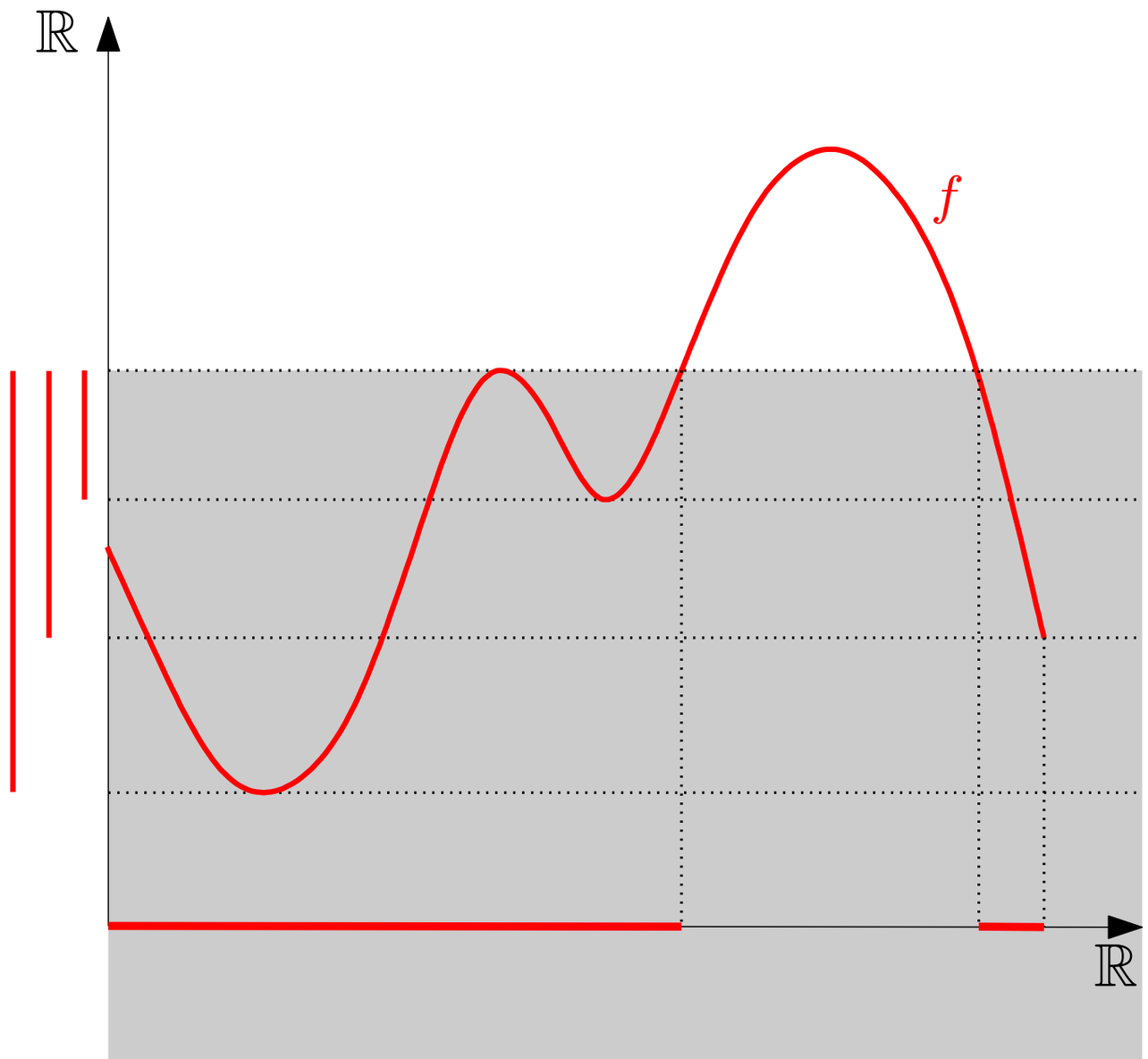
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



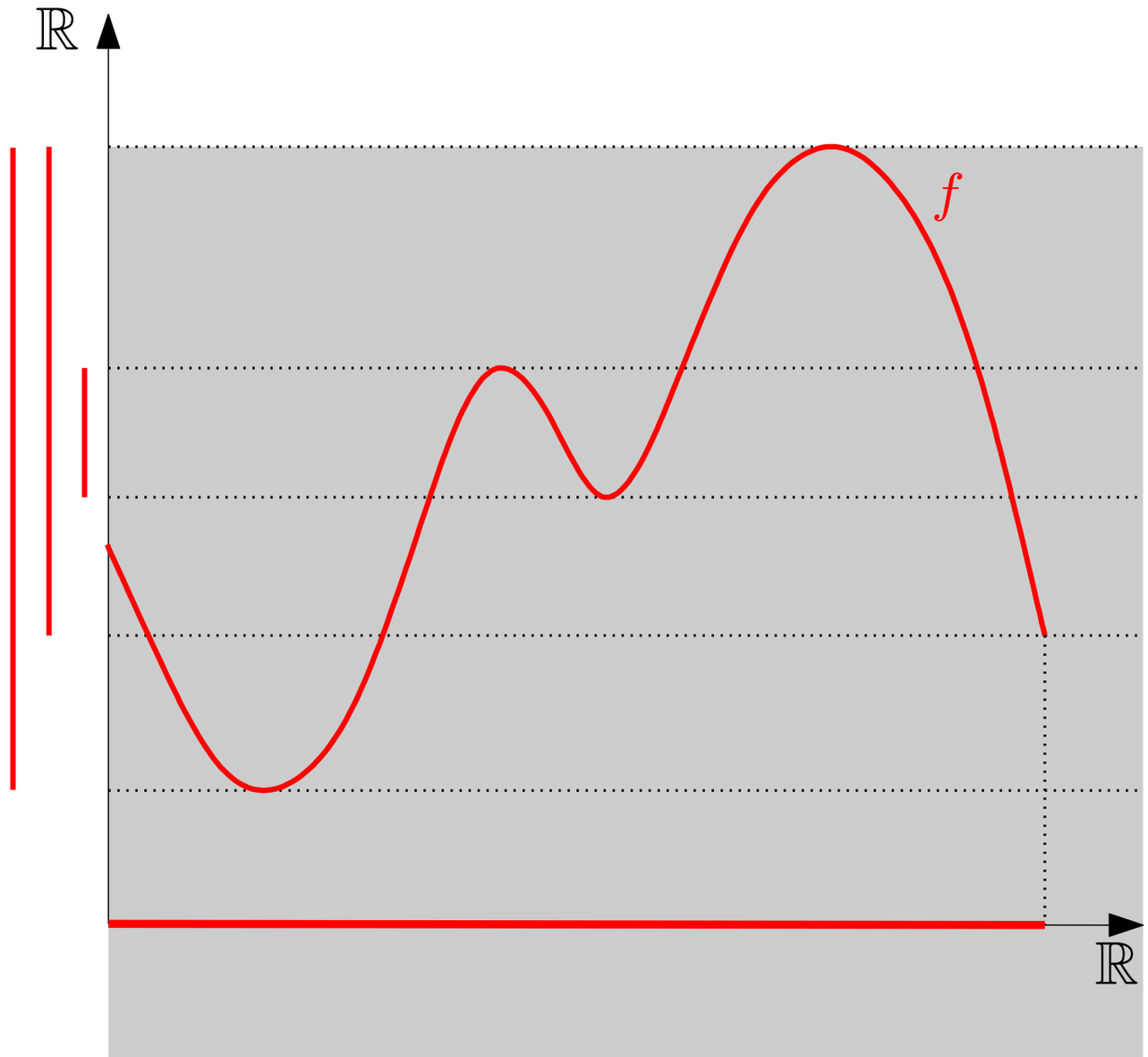
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



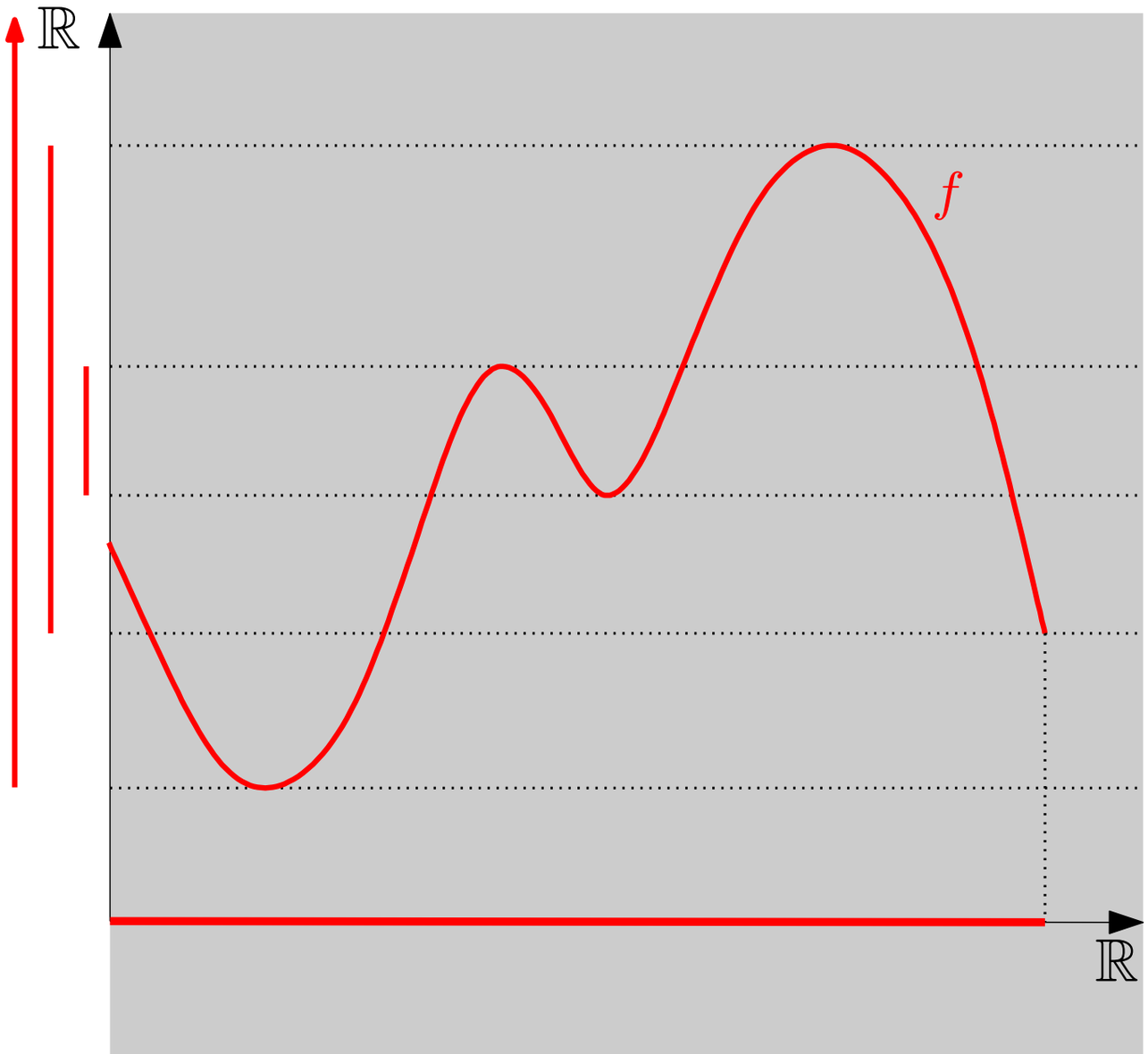
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family



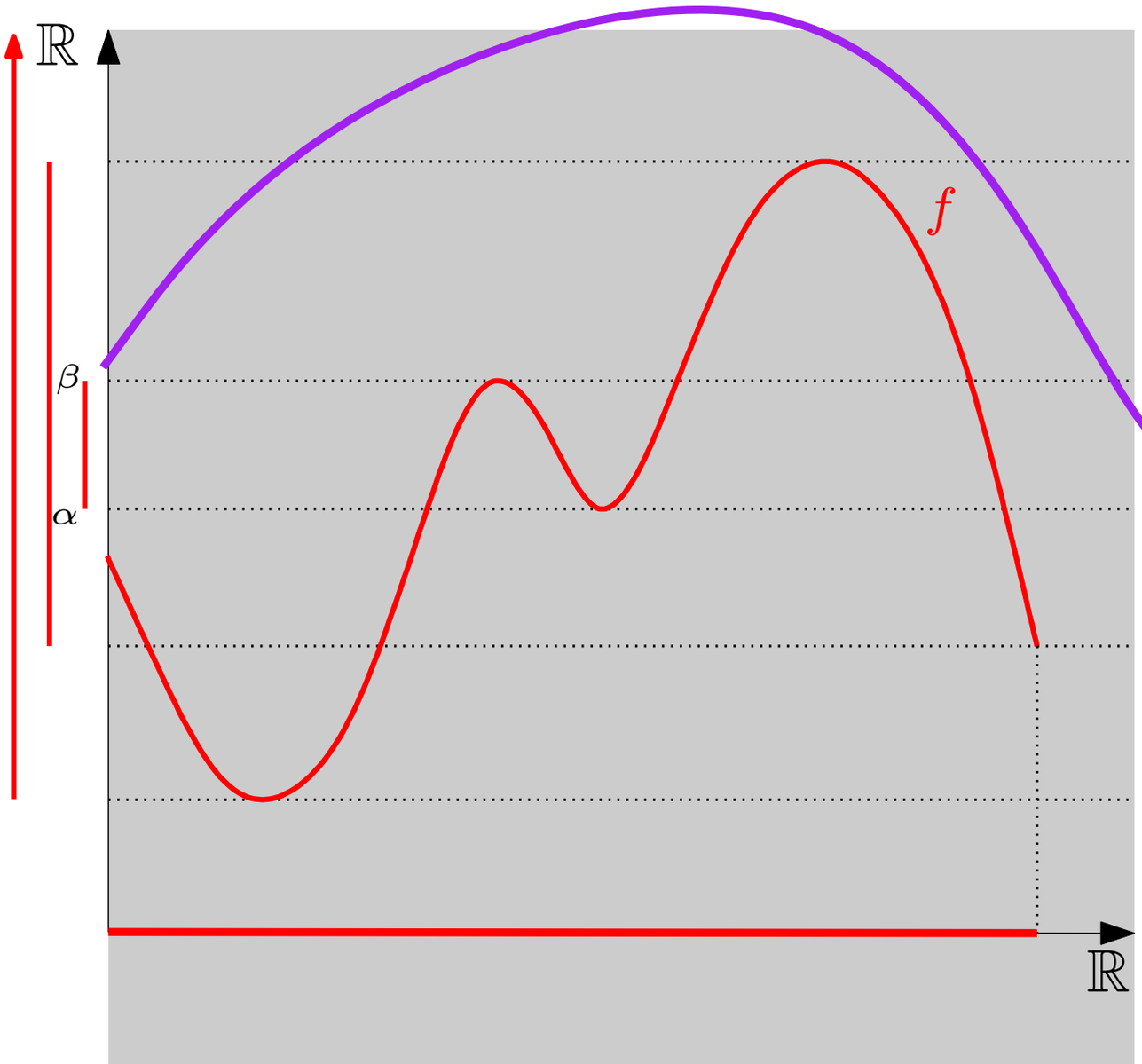
Example: persistence of sublevel sets of function

- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family
- finite set of intervals (barcode) encodes births/deaths of homology classes

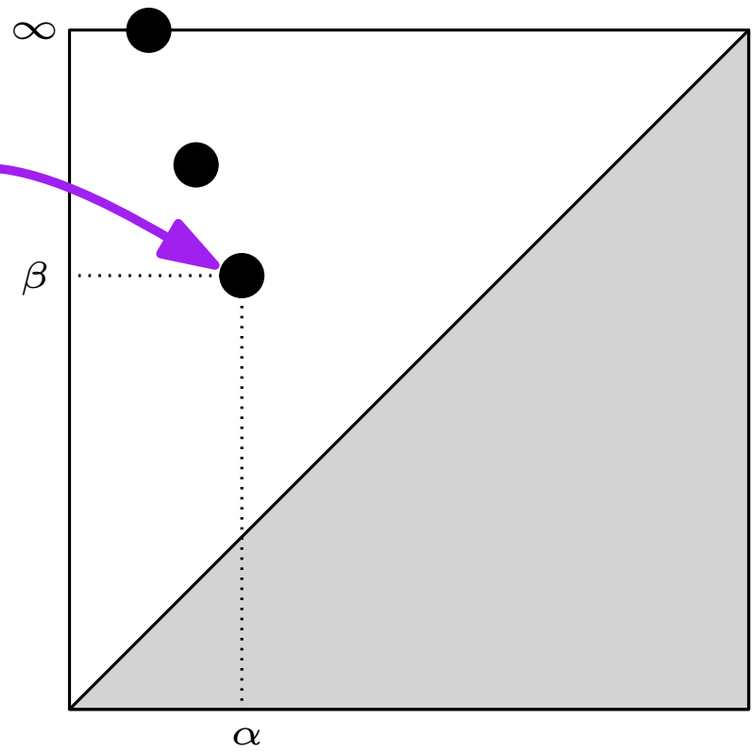


Example: persistence of sublevel sets of function

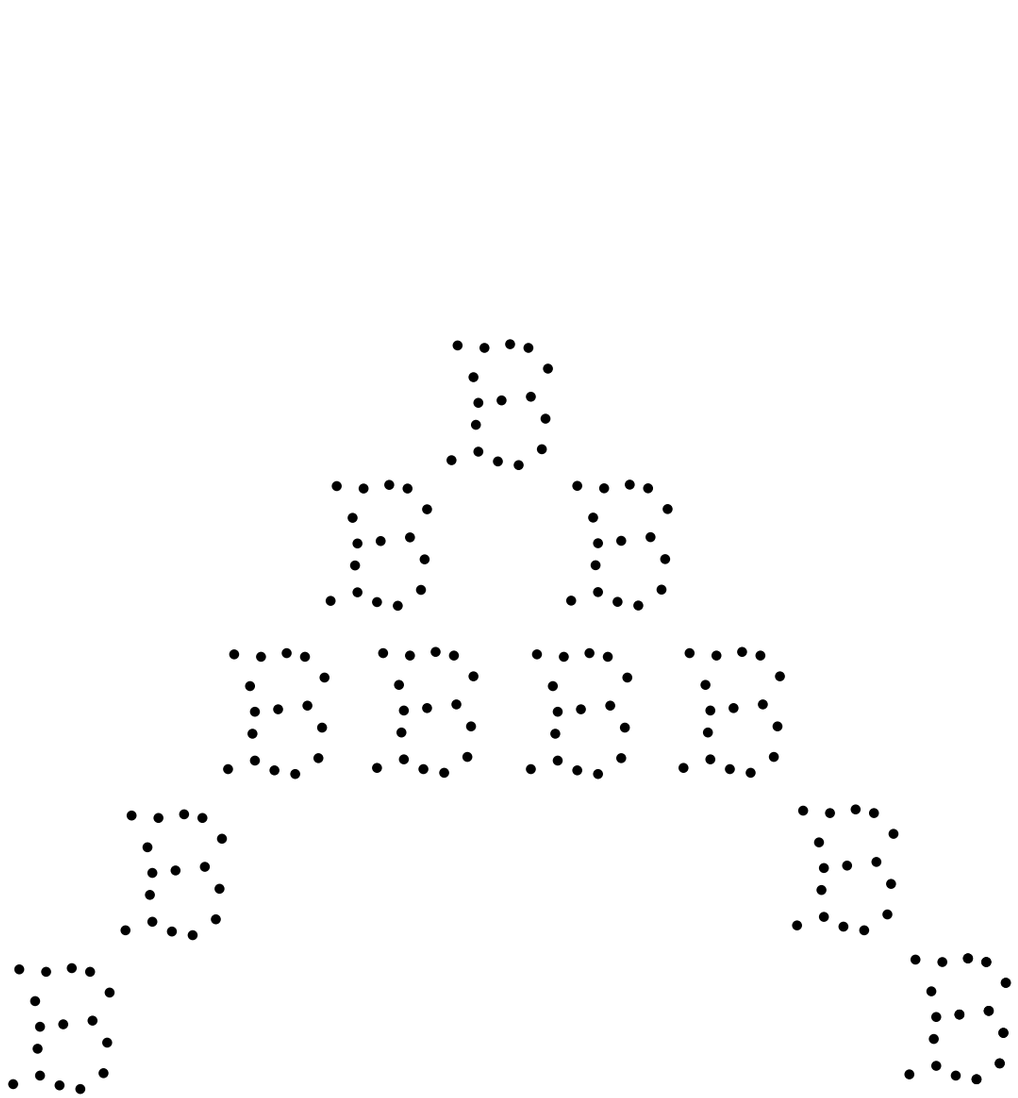
- input: *filtration* = nested family of sublevel-sets $f^{-1}((-\infty, t])$ for t ranging over $\mathbb{R}$
- track the evolution of the topology (homology) throughout the family
- finite set of intervals (barcode) encodes births/deaths of homology classes

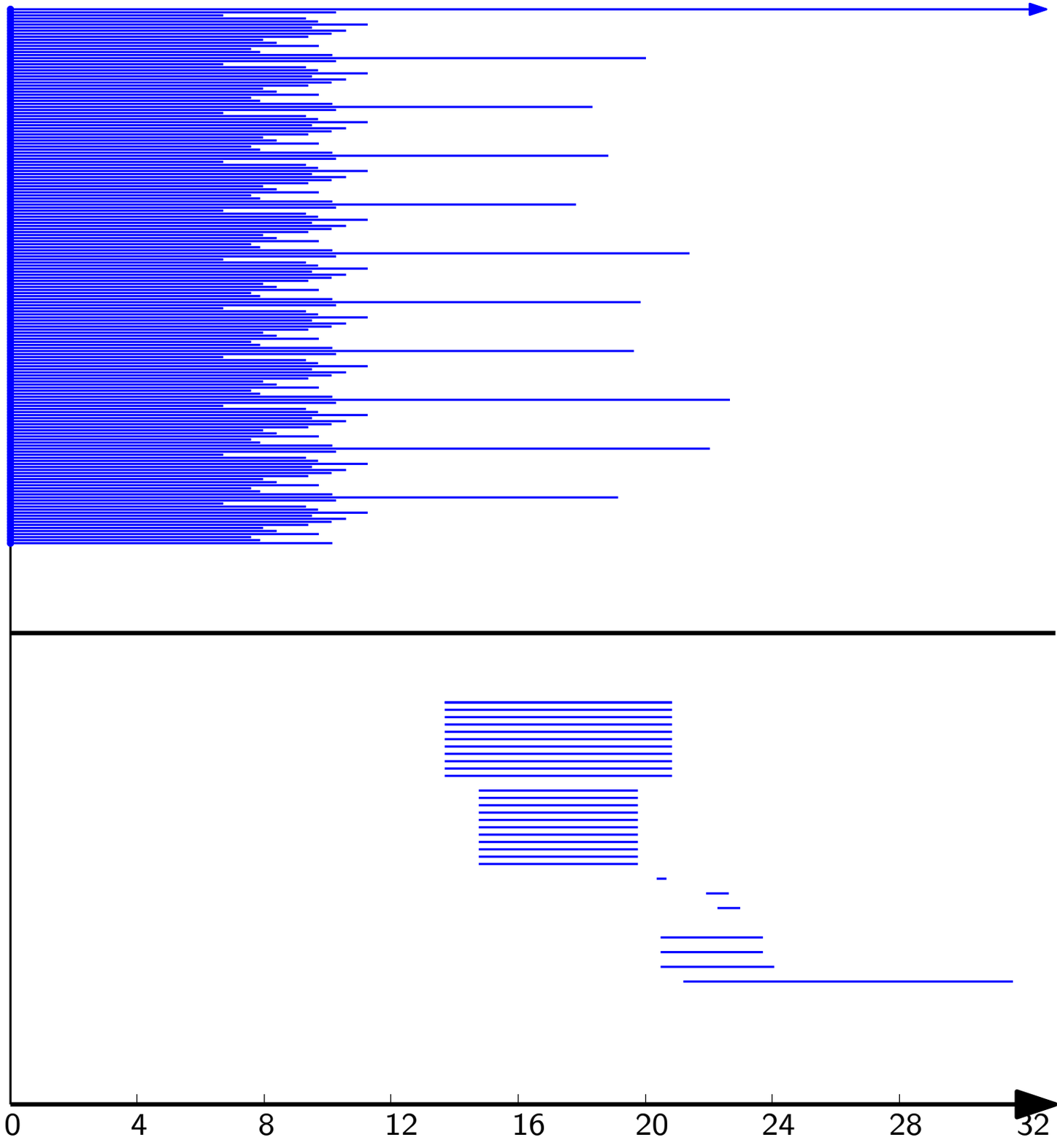


- alternate representation as a (multi-) set of points in the plane (*persistence diagram*).

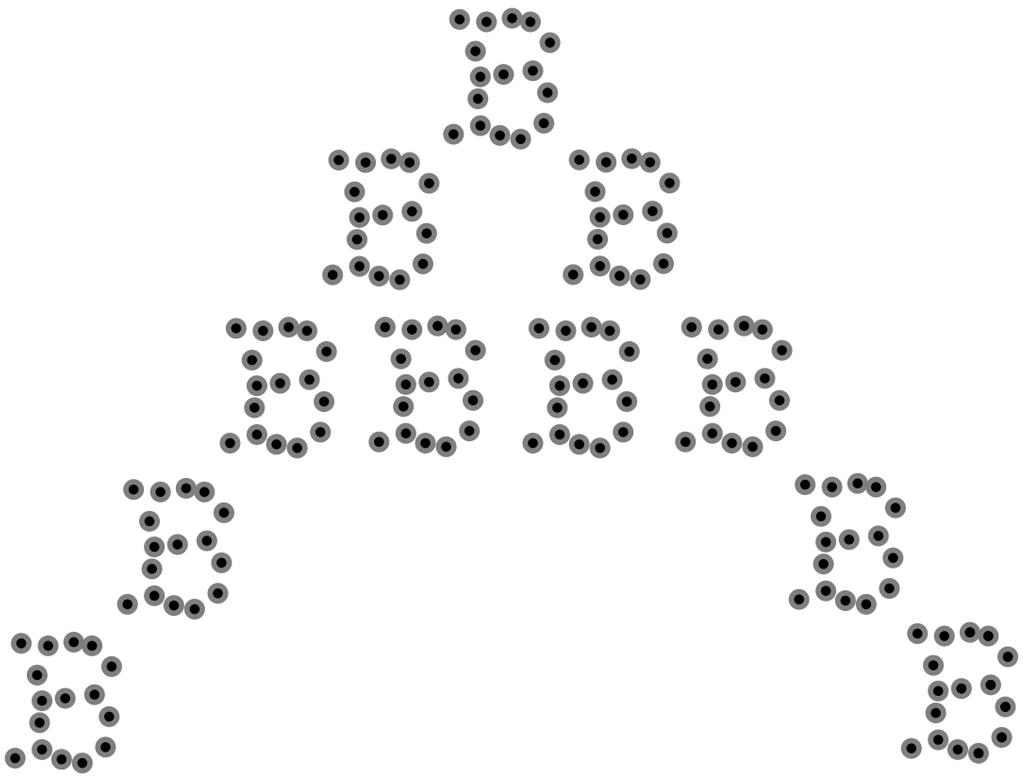


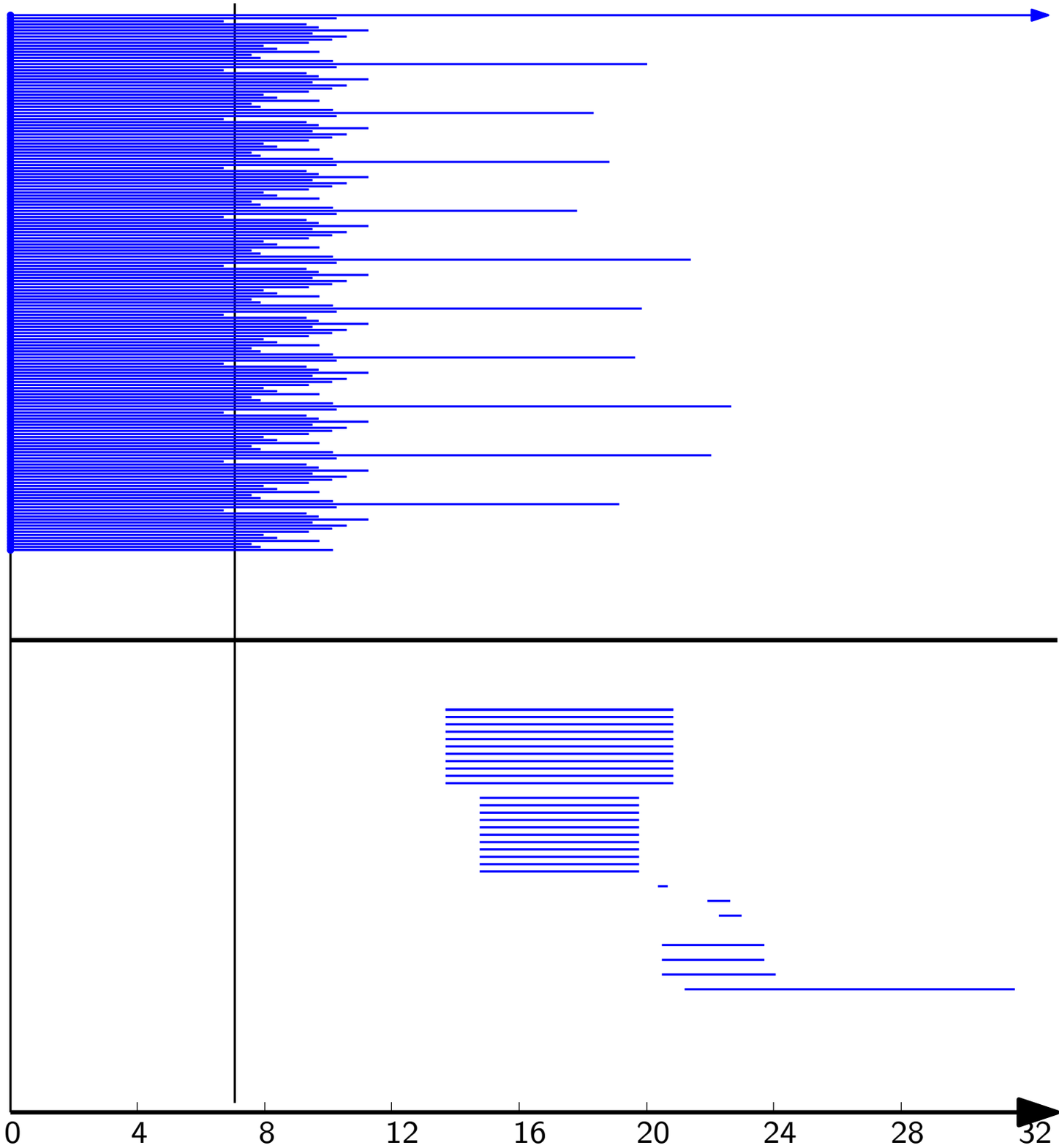
Example: persistence of Čech complexes



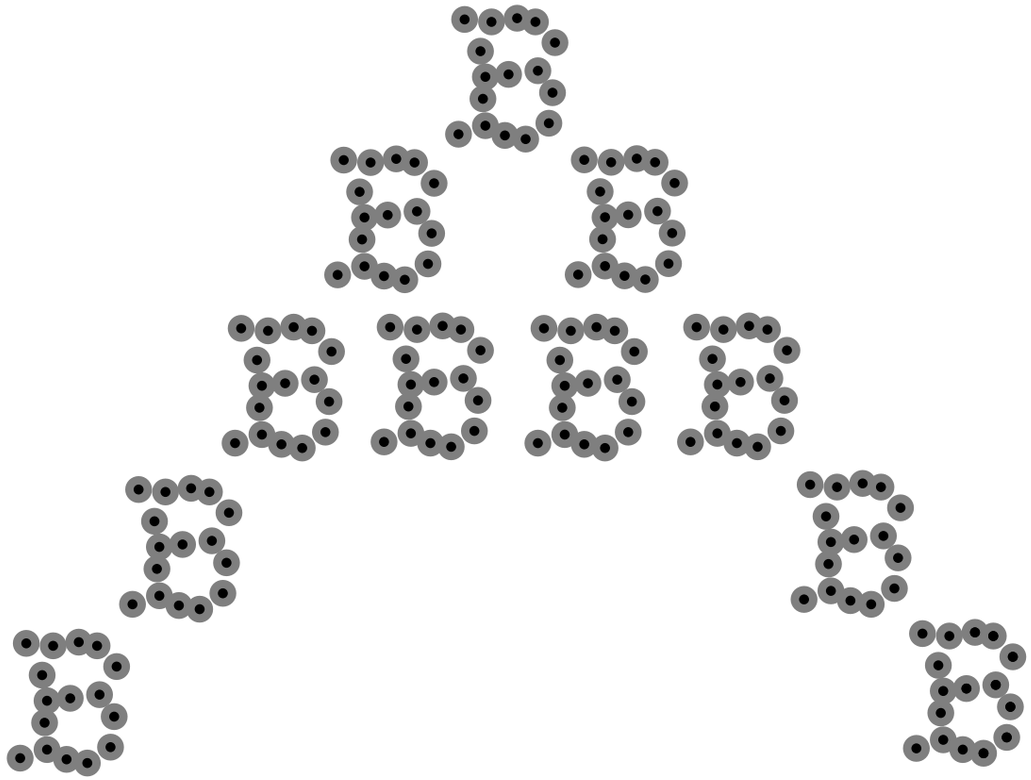


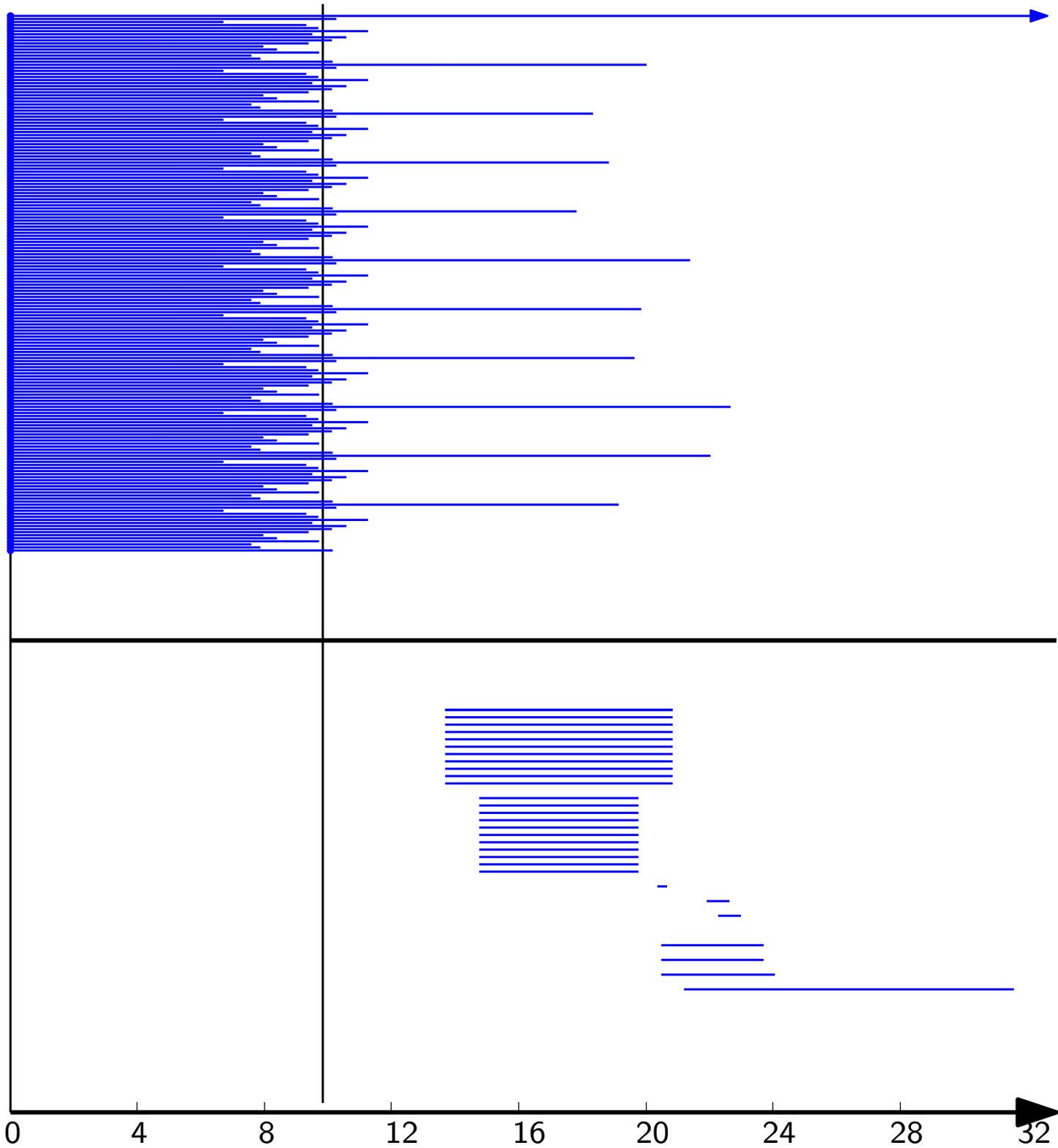
Example: persistence of Čech complexes



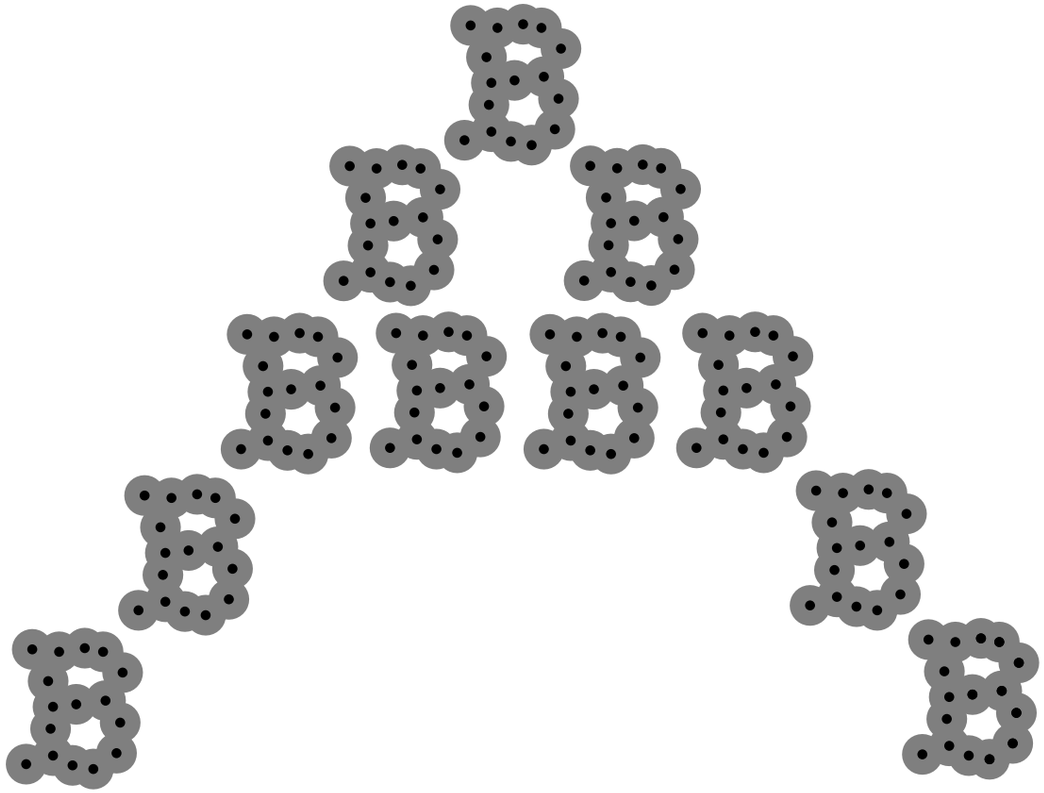


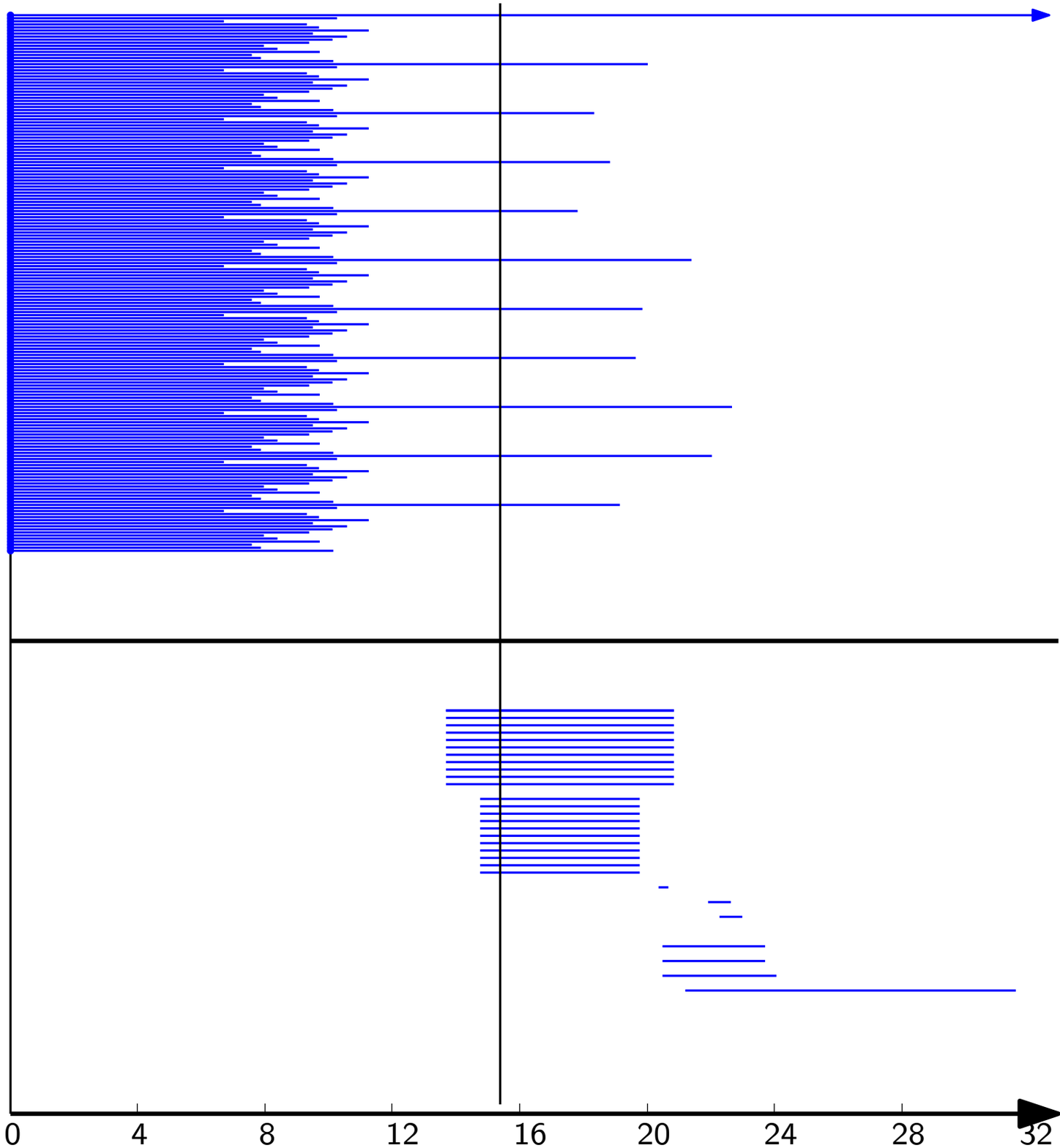
Example: persistence of Čech complexes



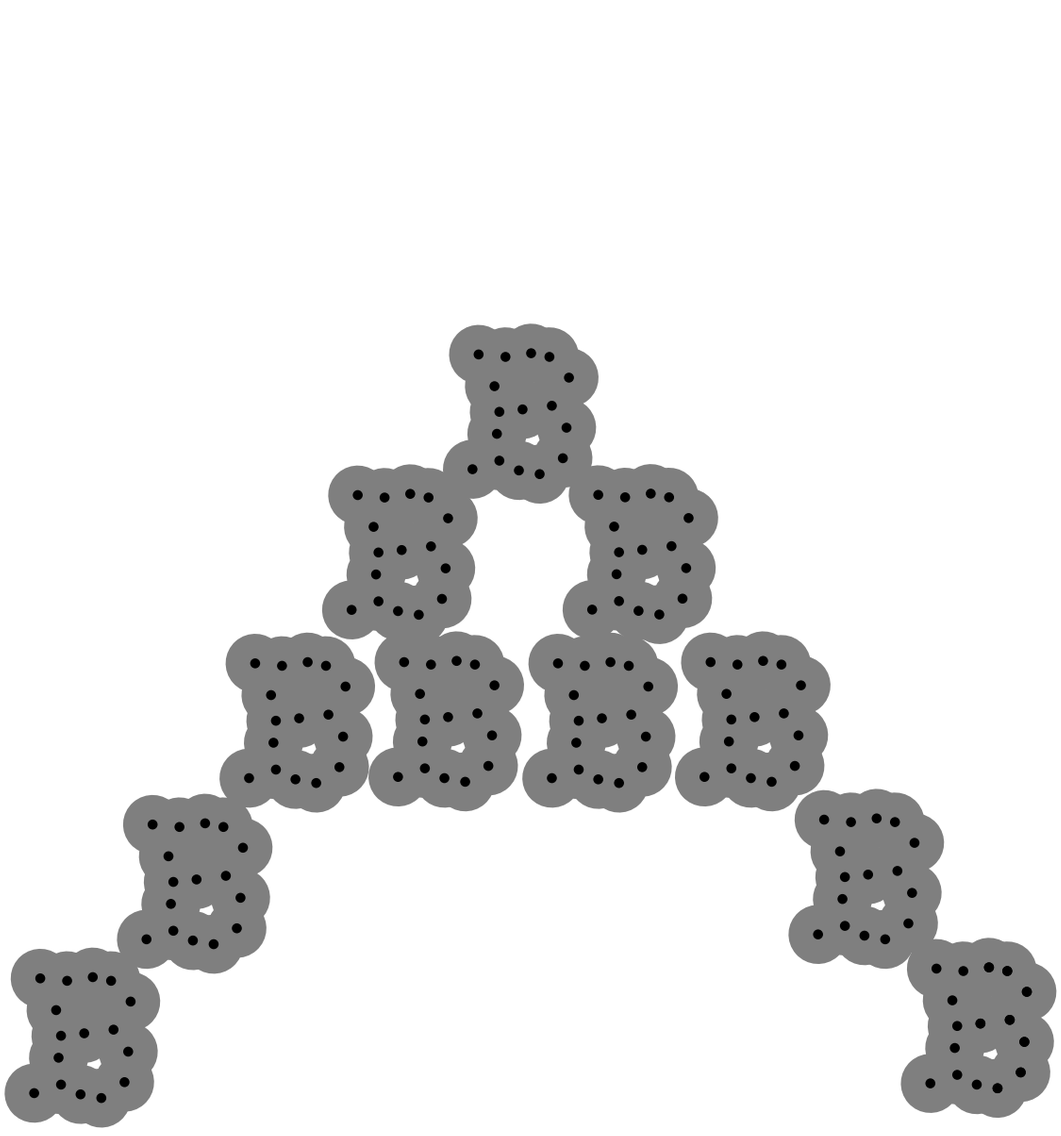


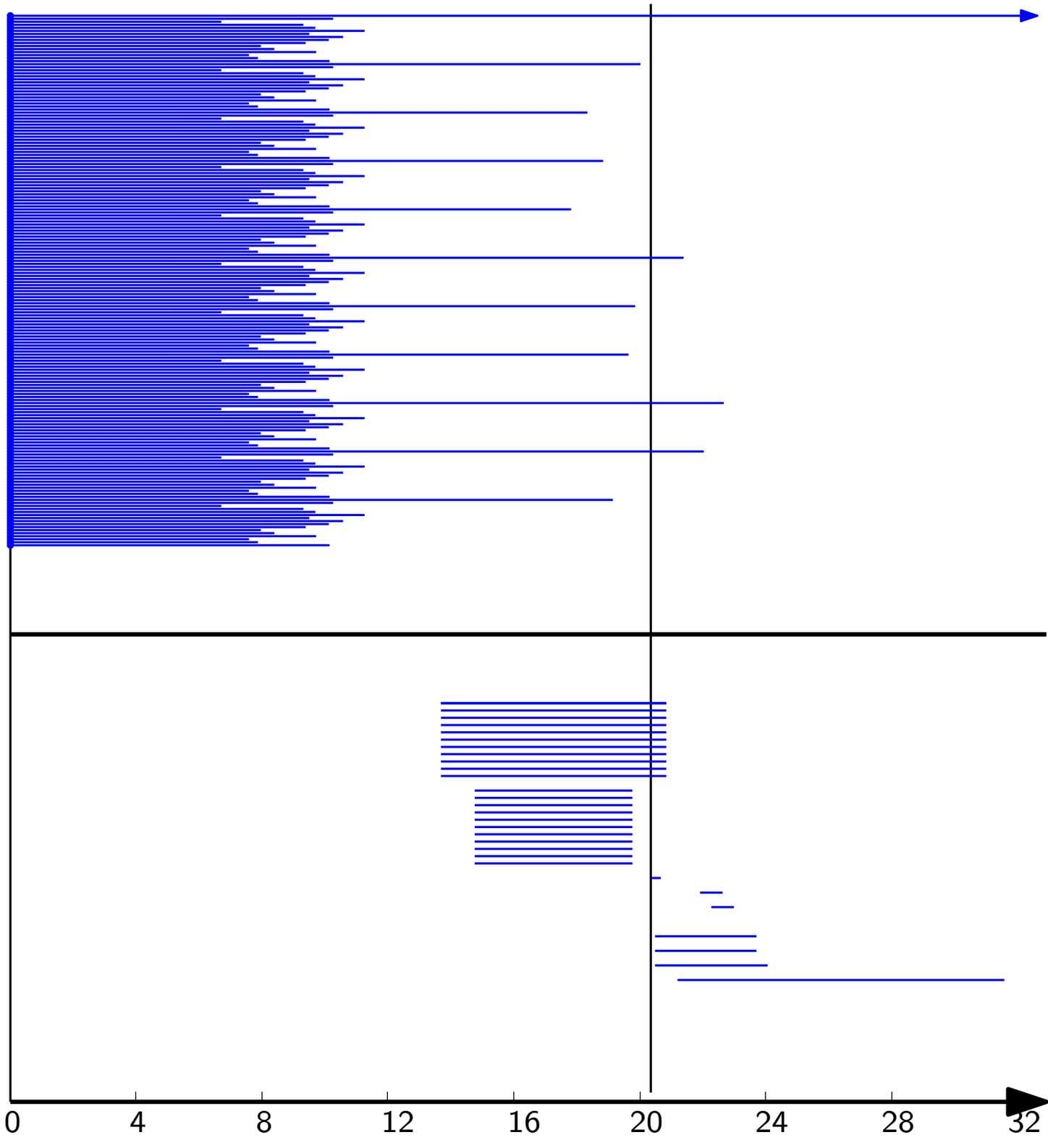
Example: persistence of Čech complexes



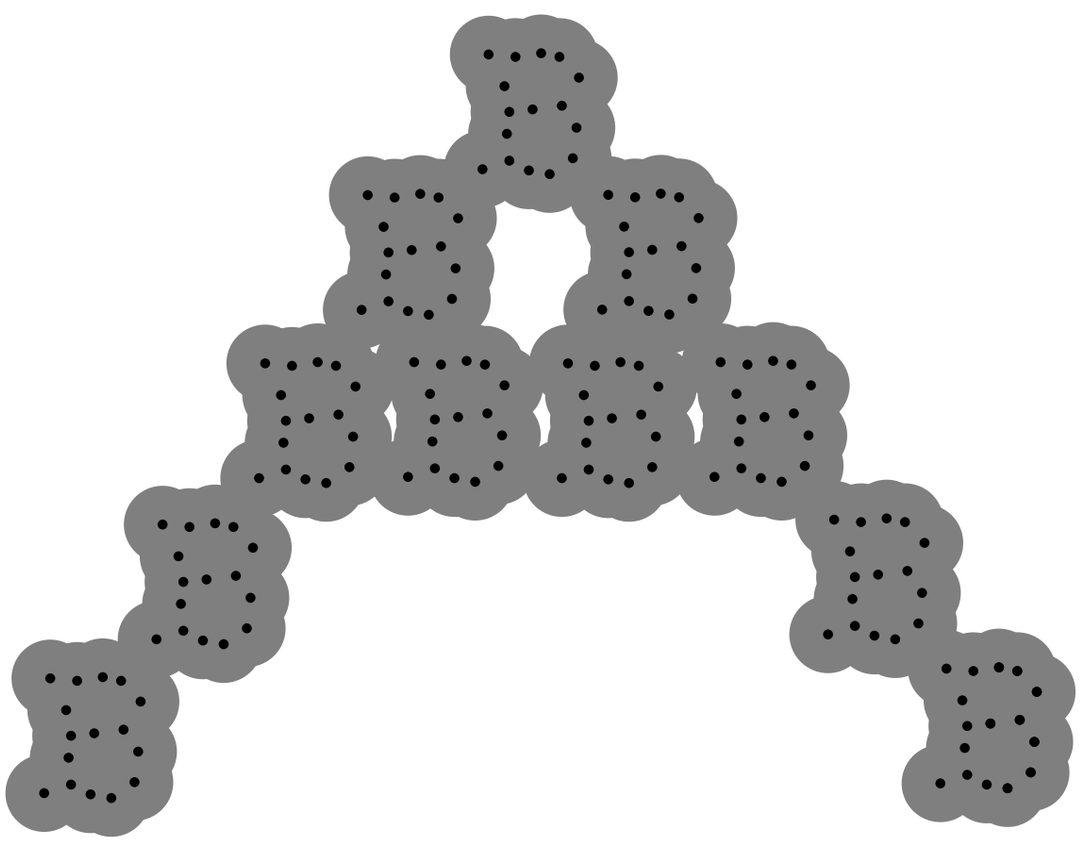


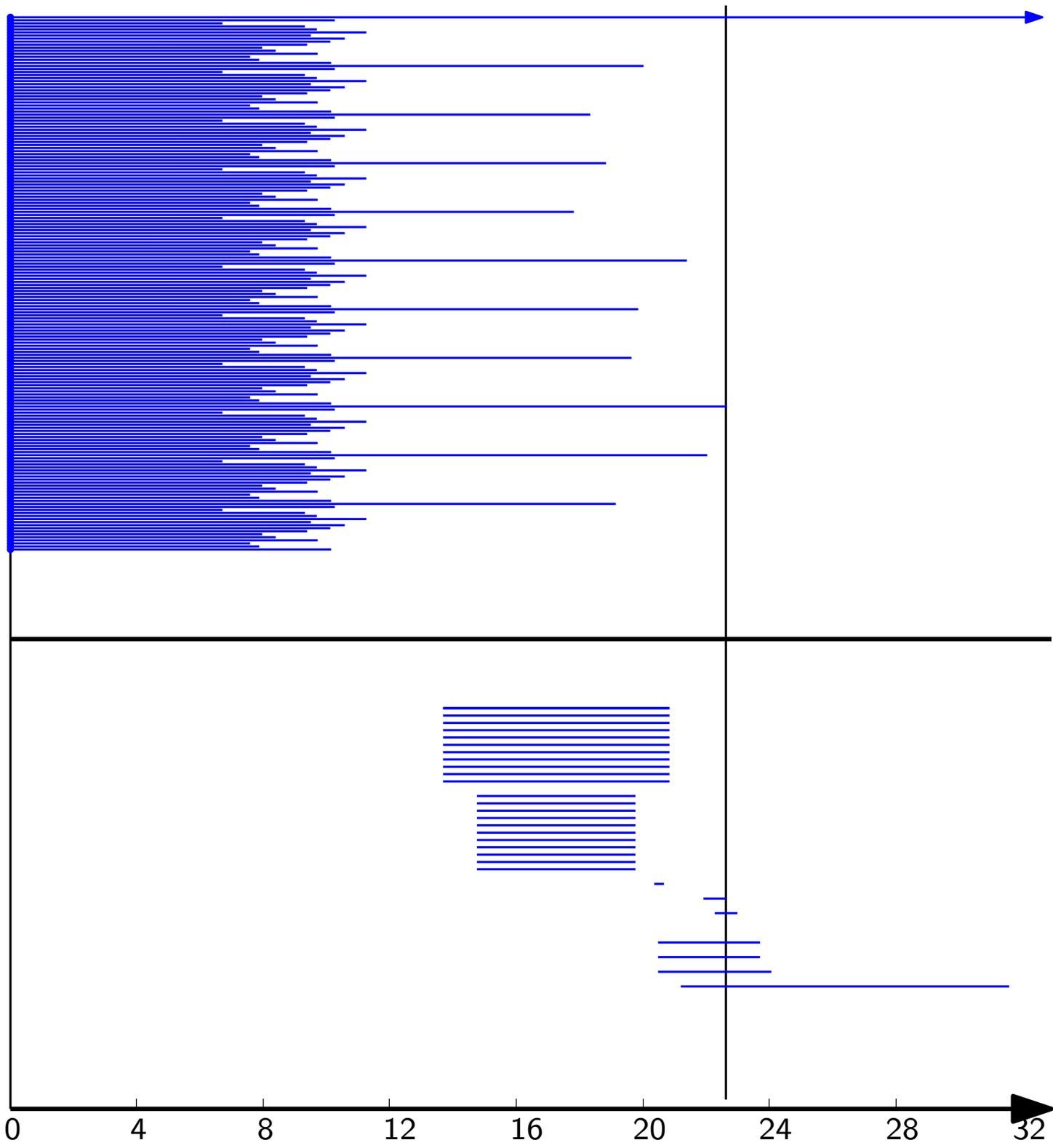
Example: persistence of Čech complexes



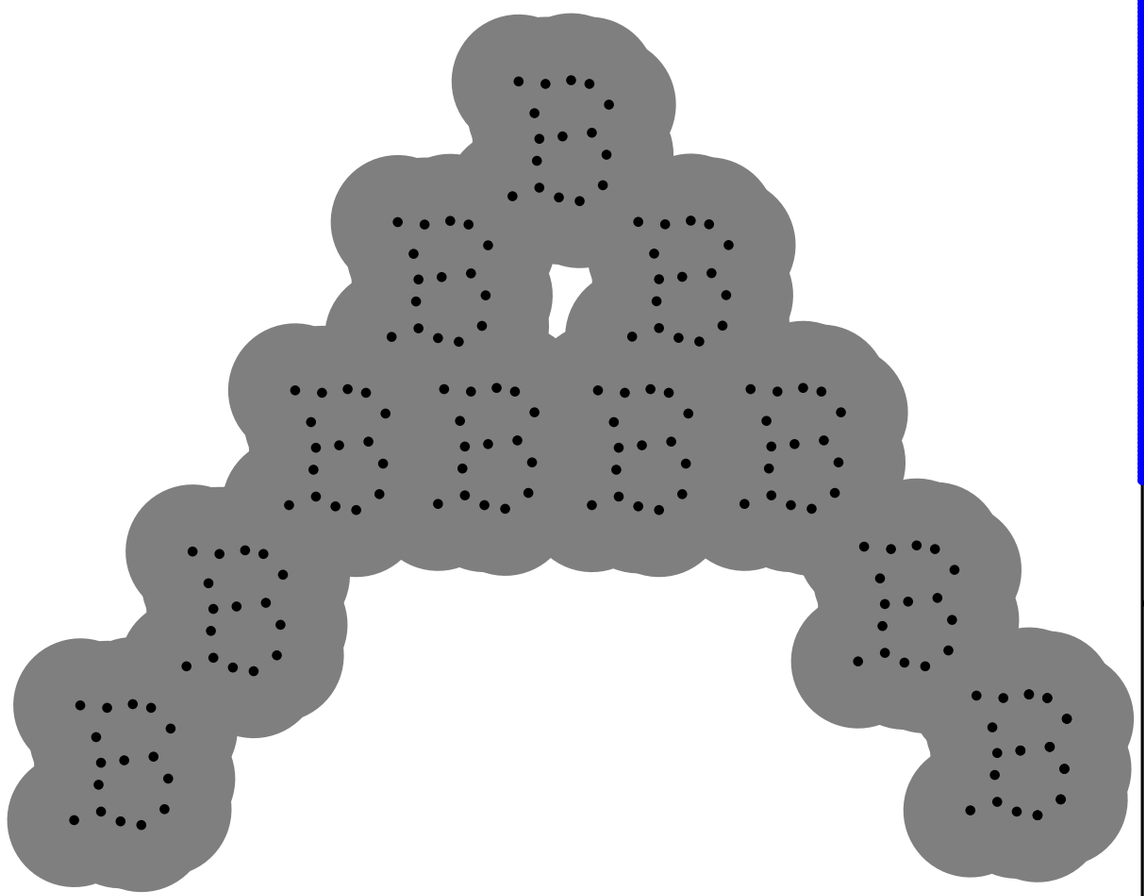


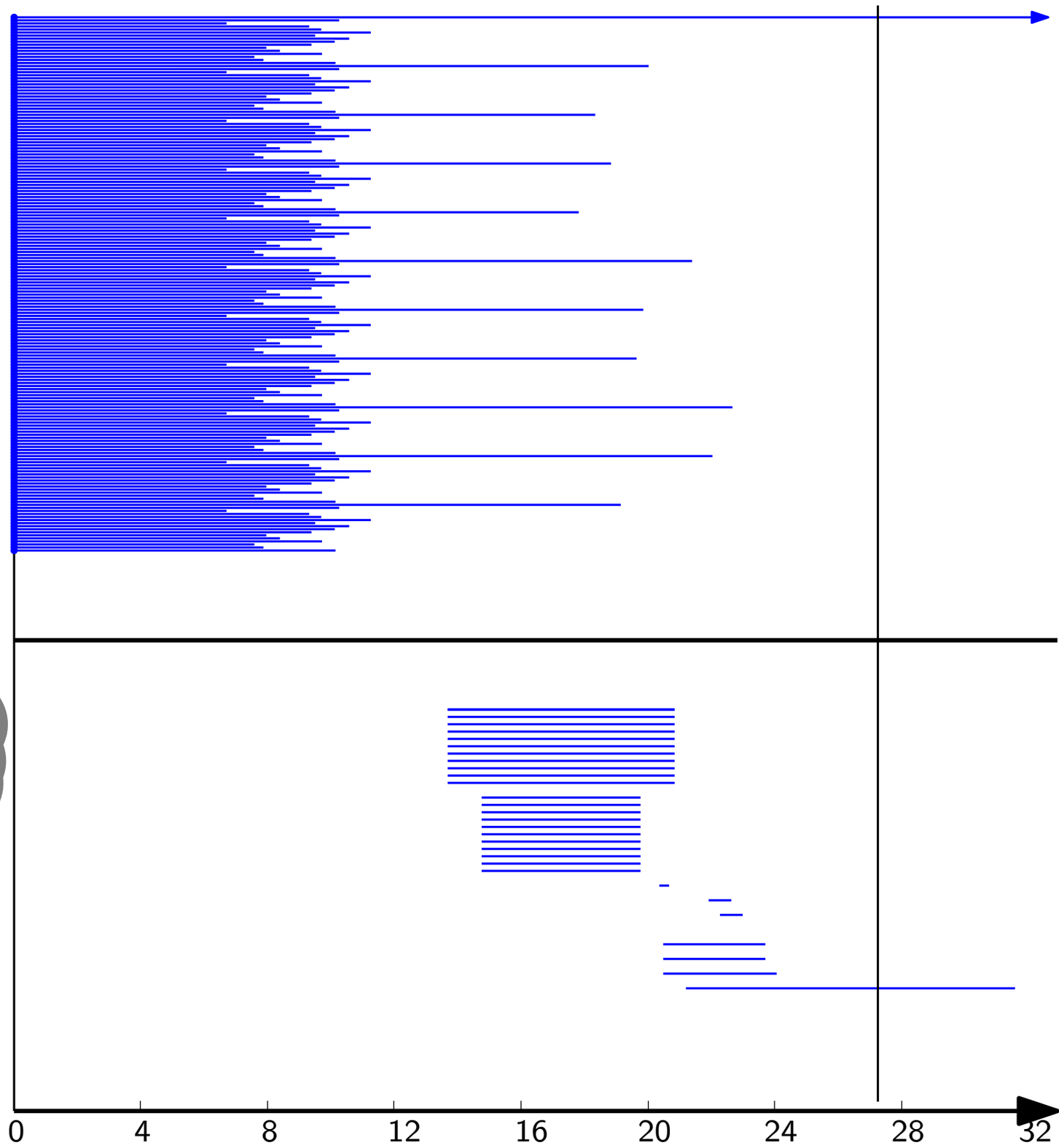
Example: persistence of Čech complexes



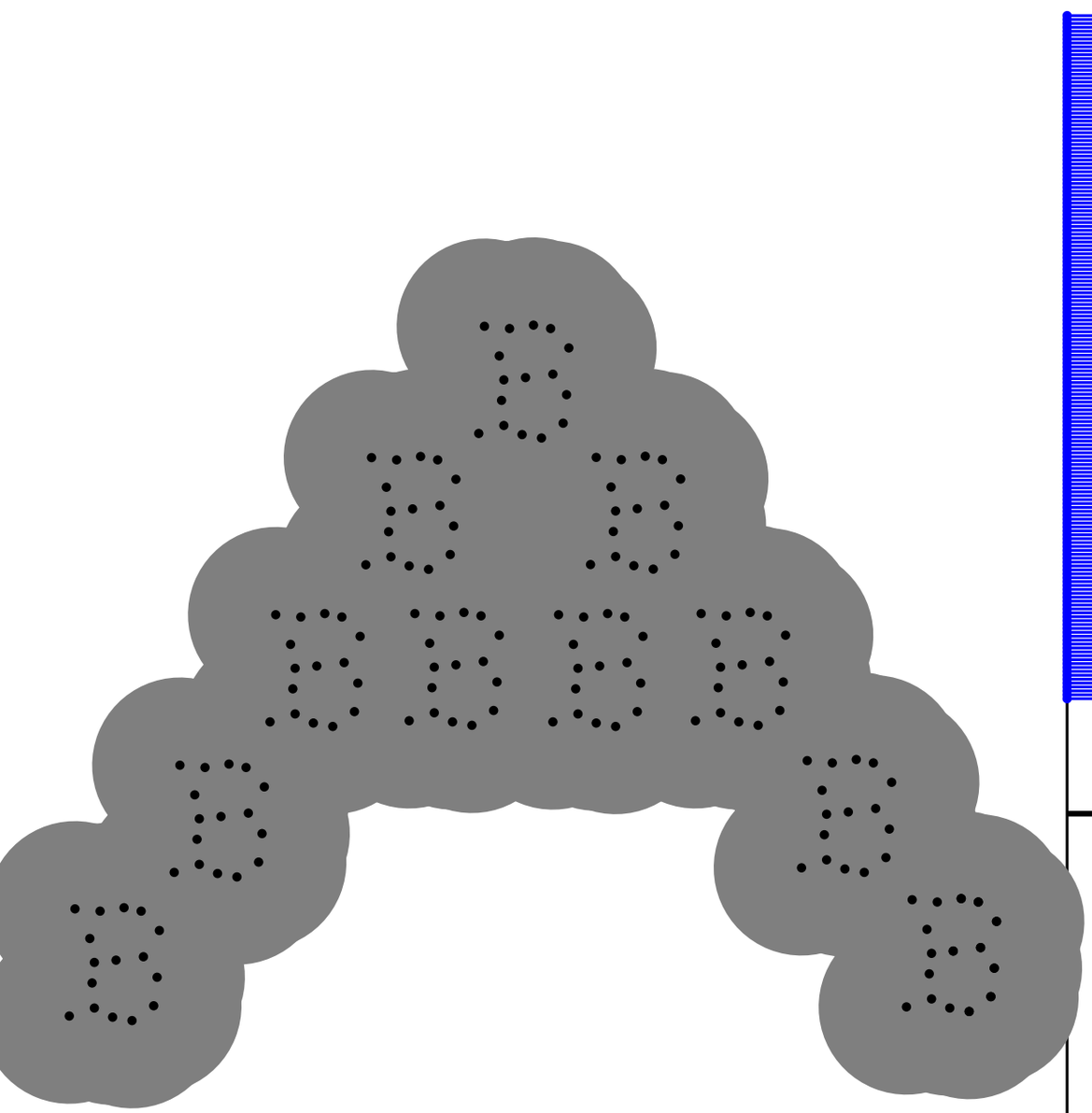


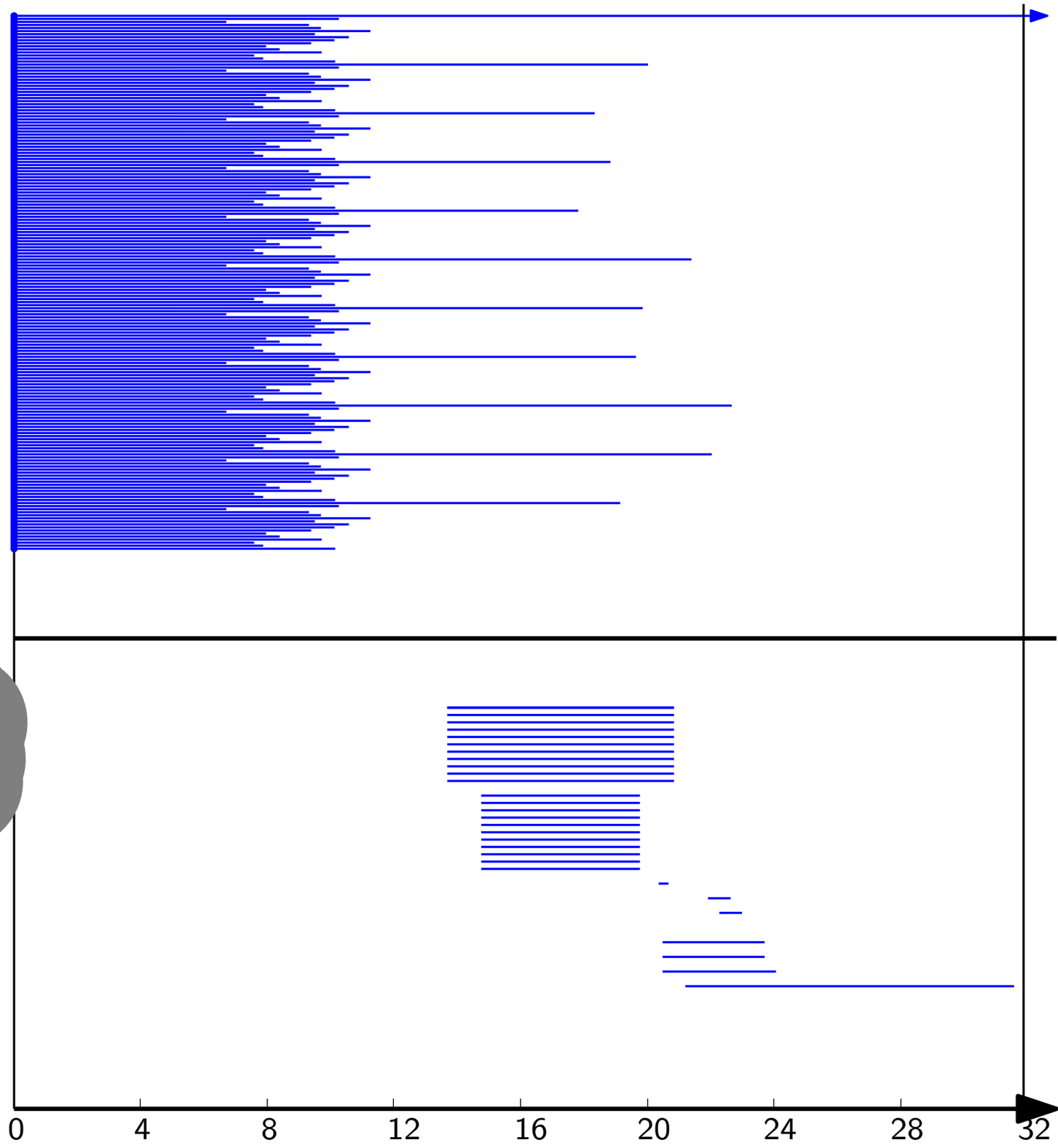
Example: persistence of Čech complexes



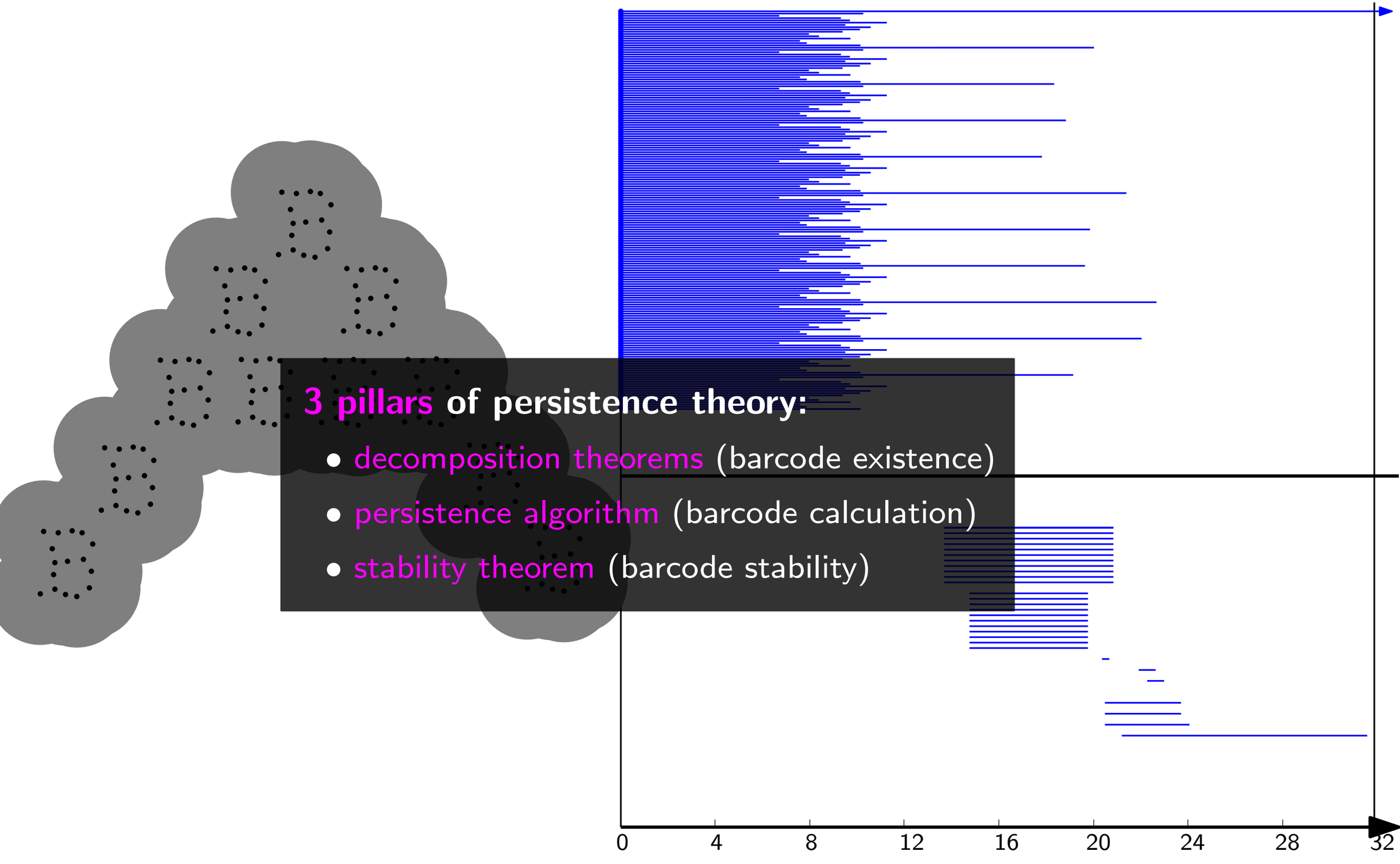


Example: persistence of Čech complexes





Example: persistence of Čech complexes

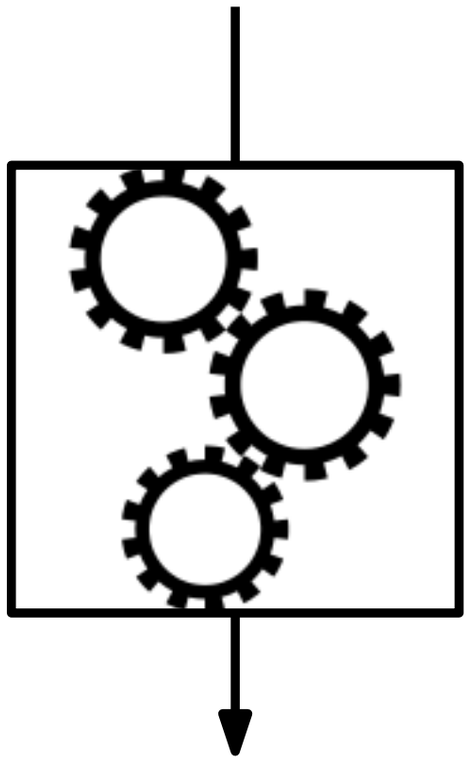


3 pillars of persistence theory:

- decomposition theorems (barcode existence)
- persistence algorithm (barcode calculation)
- stability theorem (barcode stability)

Mathematical foundations

Filtration: $F_1 \subseteq F_2 \subseteq F_3 \subseteq F_4 \subseteq F_5 \cdots$



(homology functor)

topological level

algebraic level

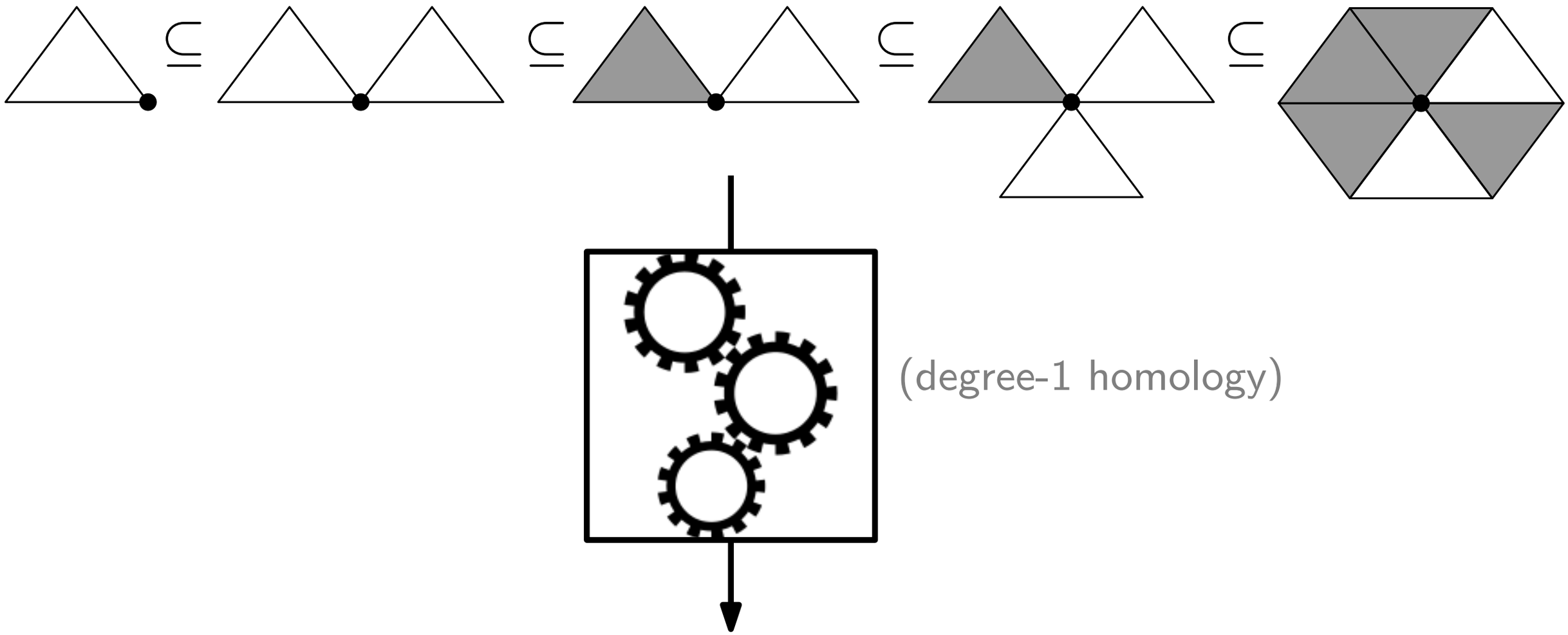
$$H_*(F_1) \to H_*(F_2) \to H_*(F_3) \to H_*(F_4) \to H_*(F_5) \to \cdots$$

Def: A *persistence module* is a sequence of vector spaces connected with linear maps:

$$H_*(F_1) \to H_*(F_2) \to H_*(F_3) \to H_*(F_4) \to \cdots$$

Mathematical foundations

Example:

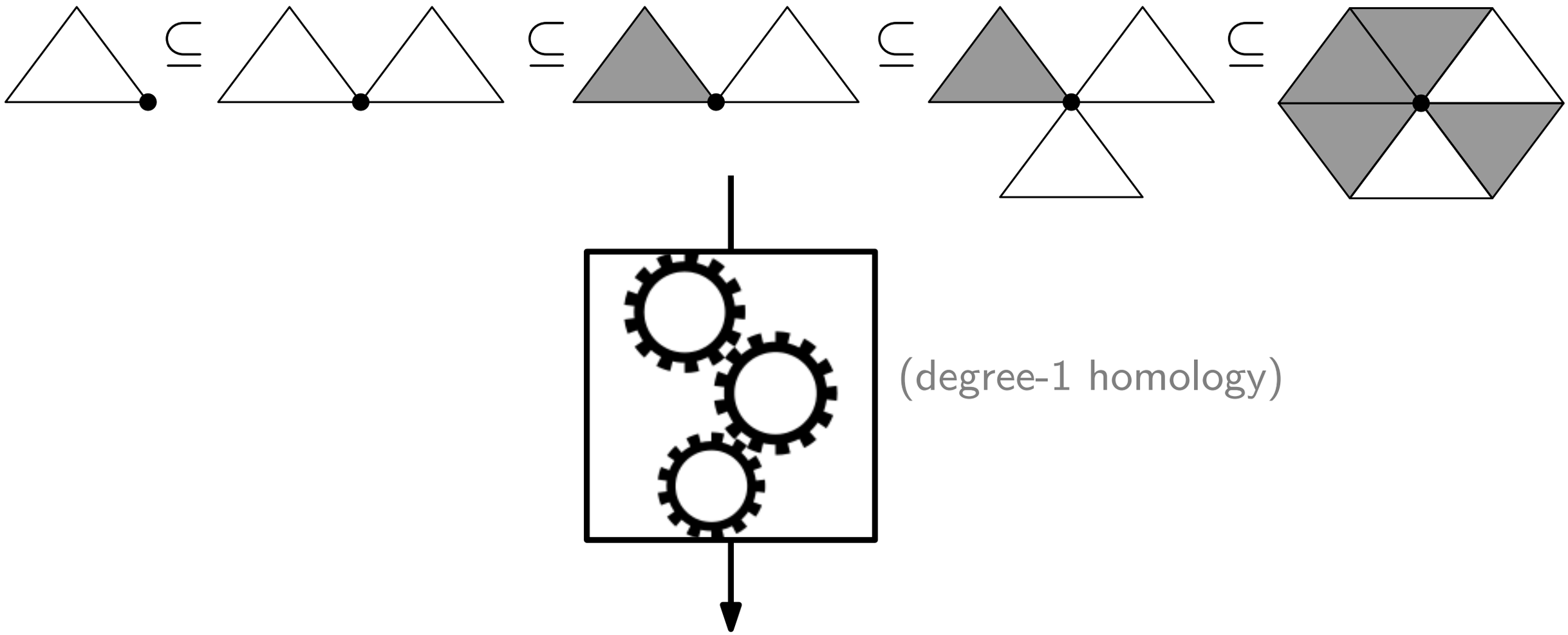


(degree-1 homology)

$$\boldsymbol{k} \xrightarrow{\binom{1}{0}} \boldsymbol{k}^2 \xrightarrow{(0\ 1)} \boldsymbol{k}$$

Mathematical foundations

Example:

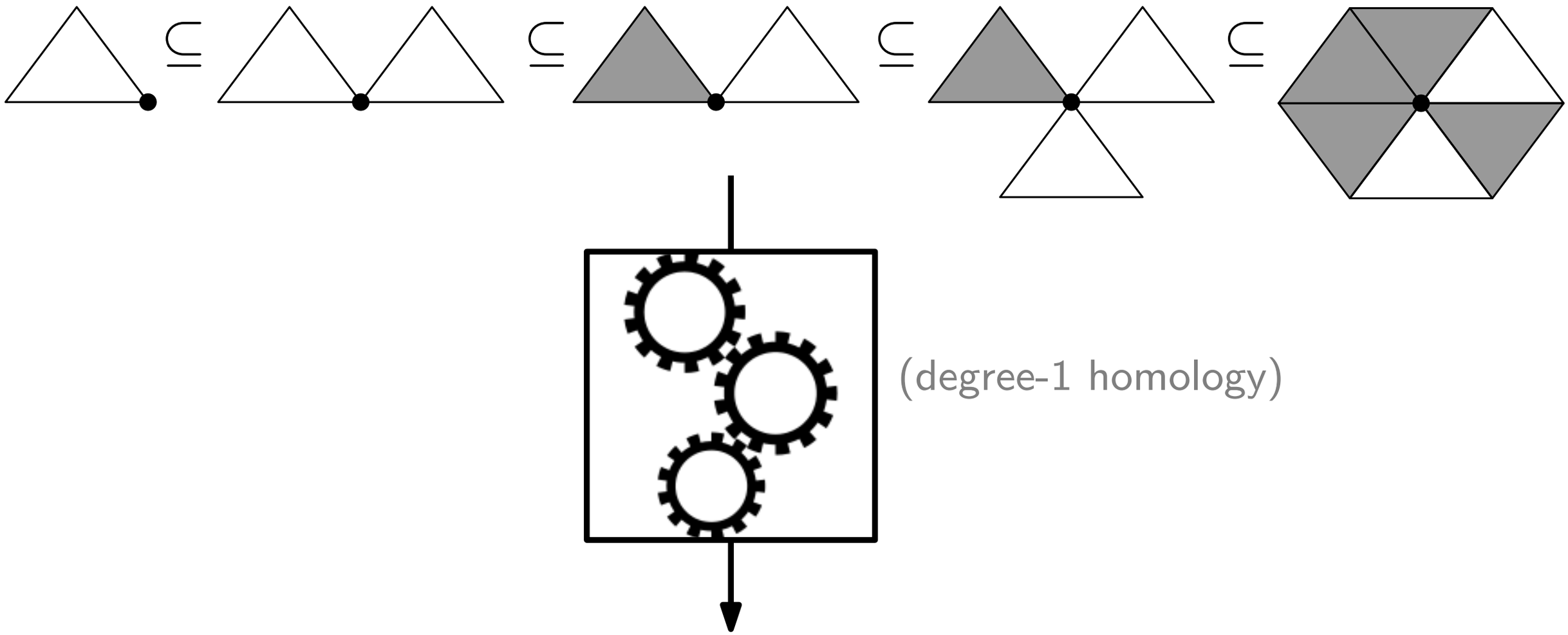


(degree-1 homology)

$$\boldsymbol{k} \xrightarrow{\binom{1}{0}} \boldsymbol{k}^2 \xrightarrow{(0\ 1)} \boldsymbol{k} \qquad \xrightarrow{\binom{0}{1}} \boldsymbol{k}^2$$

Mathematical foundations

Example:



(degree-1 homology)

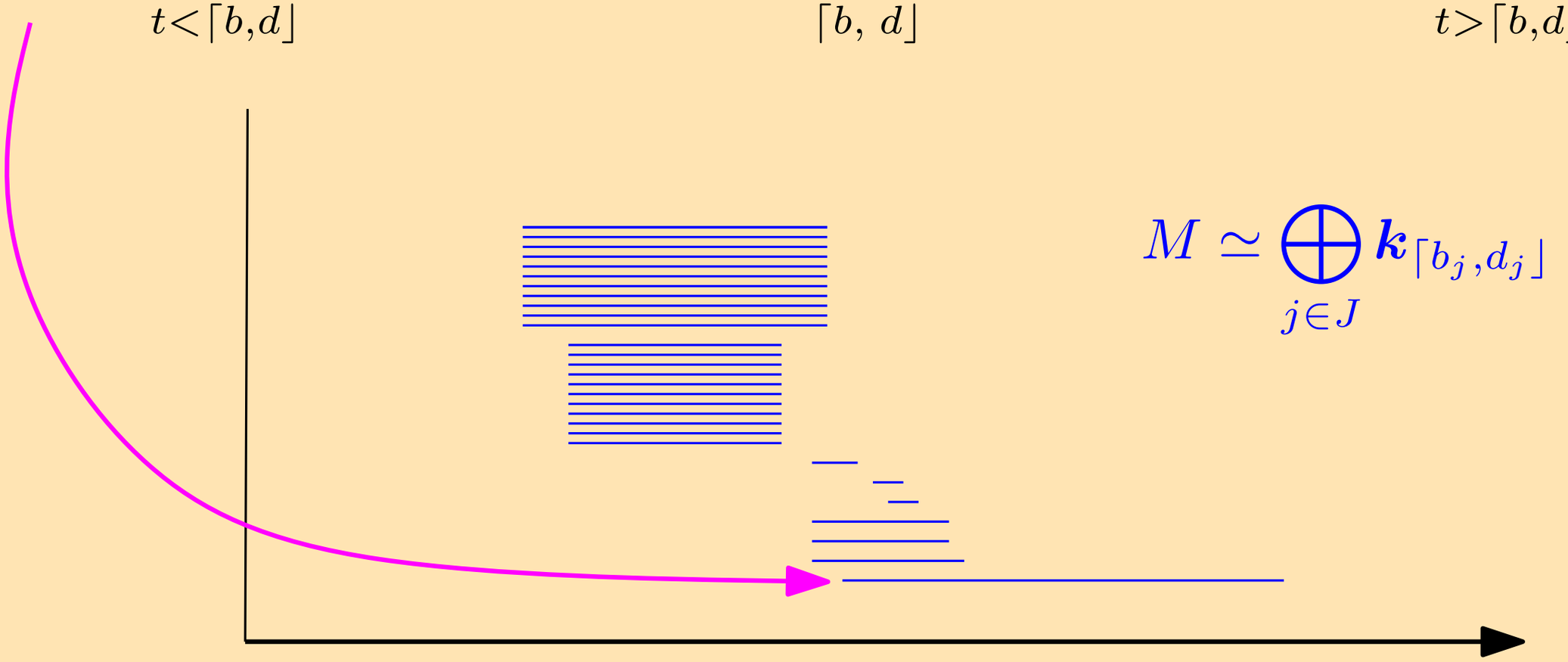
$$\boldsymbol{k} \xrightarrow{\binom{1}{0}} \boldsymbol{k}^2 \xrightarrow{(0\ 1)} \boldsymbol{k} \qquad \xrightarrow{\binom{0}{1}} \boldsymbol{k}^2 \xrightarrow{\binom{1\ 0}{0\ 1}} \boldsymbol{k}^2 \ \cdots$$

Mathematical foundations

[*The structure and stability of persistence modules*, Chazal, de Silva, Glisse, Oudot, Springer, 2016].

Thm: Let M be a persistence module over an index set $T \subseteq \mathbb{R}$. Then, M decomposes as a direct sum of *interval modules* $\boldsymbol{k}_{\lceil b,d \rfloor}$:

$$\underbrace{0 \xrightarrow{0} \cdots \xrightarrow{0} 0}_{t<\lceil b,d \rfloor} \xrightarrow{0} \underbrace{\boldsymbol{k} \xrightarrow{\mathbb{1}} \cdots \xrightarrow{\mathbb{1}} \boldsymbol{k}}_{\lceil b,\, d \rfloor} \xrightarrow{0} \underbrace{0 \xrightarrow{0} \cdots \xrightarrow{0} 0}_{t>\lceil b,d \rfloor}$$

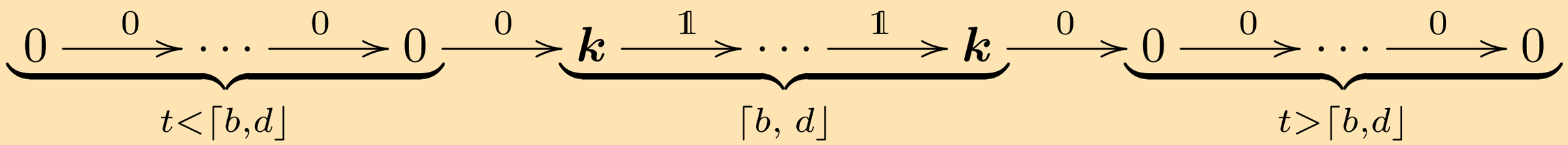


(the barcode is a complete descriptor of the algebraic structure of M)

Mathematical foundations

[*The structure and stability of persistence modules*, Chazal, de Silva, Glisse, Oudot, Springer, 2016].

Thm: Let M be a persistence module over an index set $T \subseteq \mathbb{R}$. Then, M decomposes as a direct sum of *interval modules* $\boldsymbol{k}_{\lceil b,d \rfloor}$:



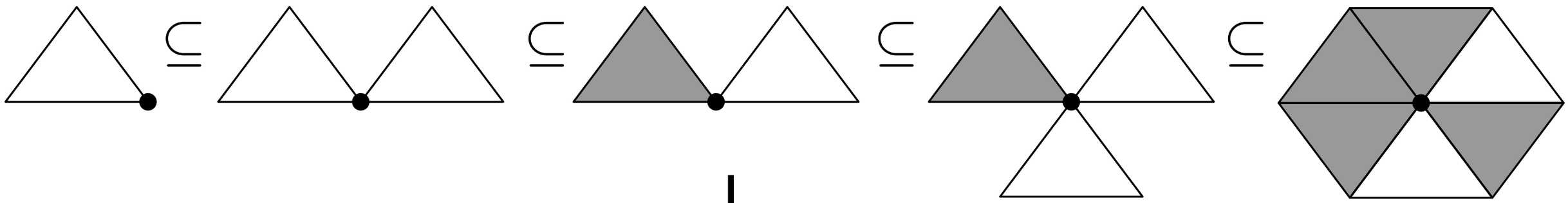
in the following cases:

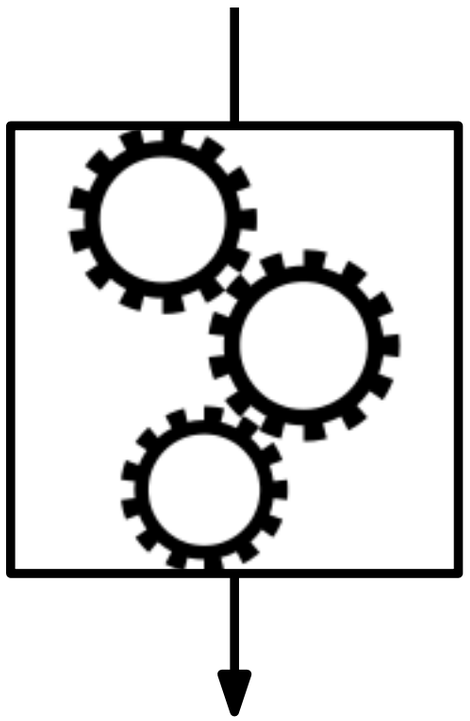
- T is finite,
- M is *pointwise finite-dimensional* (pfd), i.e., every space M_t has finite dimension.

Moreover, when it exists, the decomposition is **unique** up to isomorphism and permutation of the terms [Azumaya 1950].

Mathematical foundations

Example:

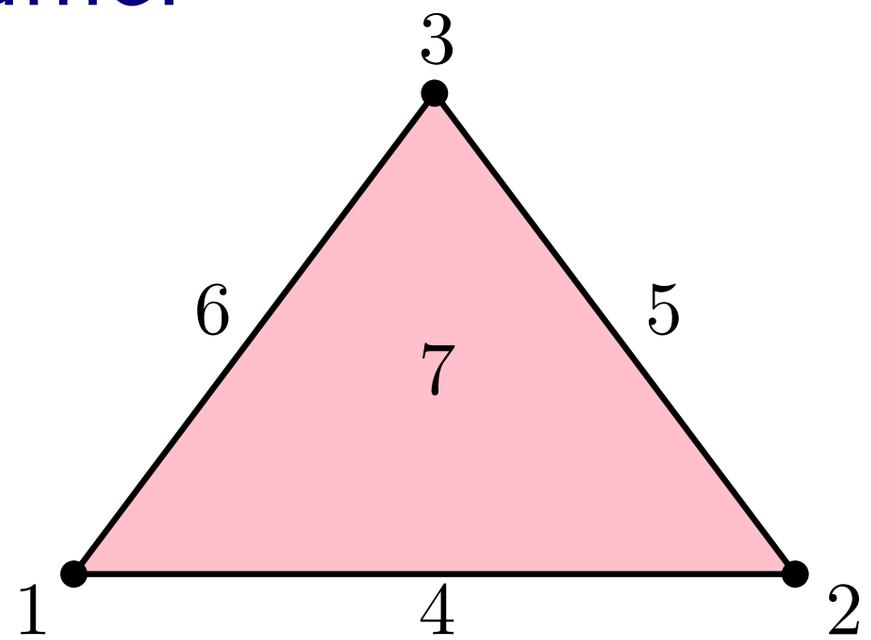




(degree-1 homology)

$$\boldsymbol{k} \xrightarrow{\binom{1}{0}} \boldsymbol{k}^2 \xrightarrow{(0\ 1)} \boldsymbol{k} \qquad \xrightarrow{\binom{0}{1}} \boldsymbol{k}^2 \xrightarrow{\binom{1\ 0}{0\ 1}} \boldsymbol{k}^2 \ \cdots$$

Good news: the algorithm is the same!



Input: simplicial filtration

Output: boundary matrix
reduced to column-echelon form

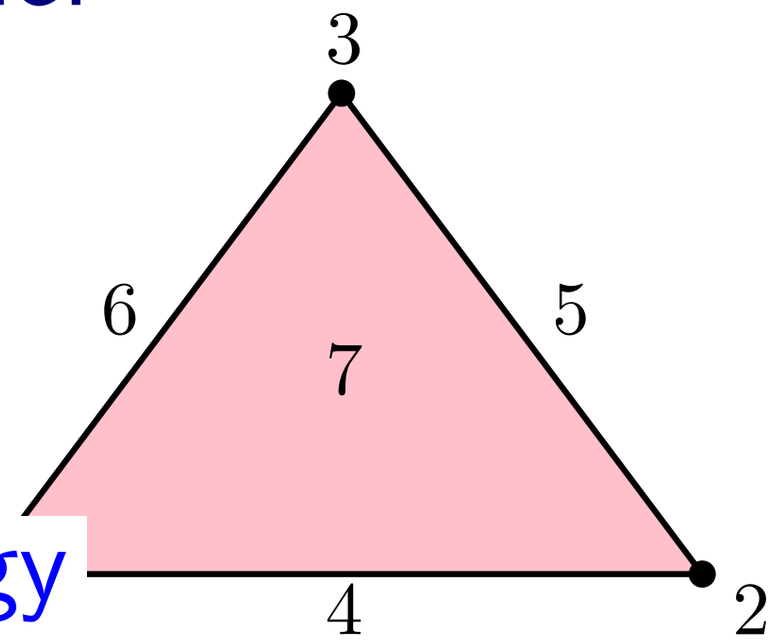
◯ simplex pairs give finite intervals:
$[2,4)$, $[3,5)$, $[6,7)$

□ unpaired simplices give infinite intervals: $[1,+\infty)$

	1	2	3	4	5	6	7
1				$*$		$*$	
2				$*$	$*$		
3					$*$	$*$	
4							$*$
5							$*$
6							$*$
7							

	1	2	3	4	5	6	7
1				$*$			
2				1	$*$		
3					1		
4							$*$
5							$*$
6							1
7							

Good news: the algorithm is the same!



Input: simplicial filtration

Output: boundary matrix
reduced to column-echelon form

◯ simplex pairs give Persistent homology
$[2,4)$, $[3,5)$, $[6,7)$

□ unpaired simplices give Regular homology

	1	2	3	4	5	6	7
1				$*$		$*$	
2				$*$	$*$		
3					$*$	$*$	
4							$*$
5							$*$
6							$*$
7							

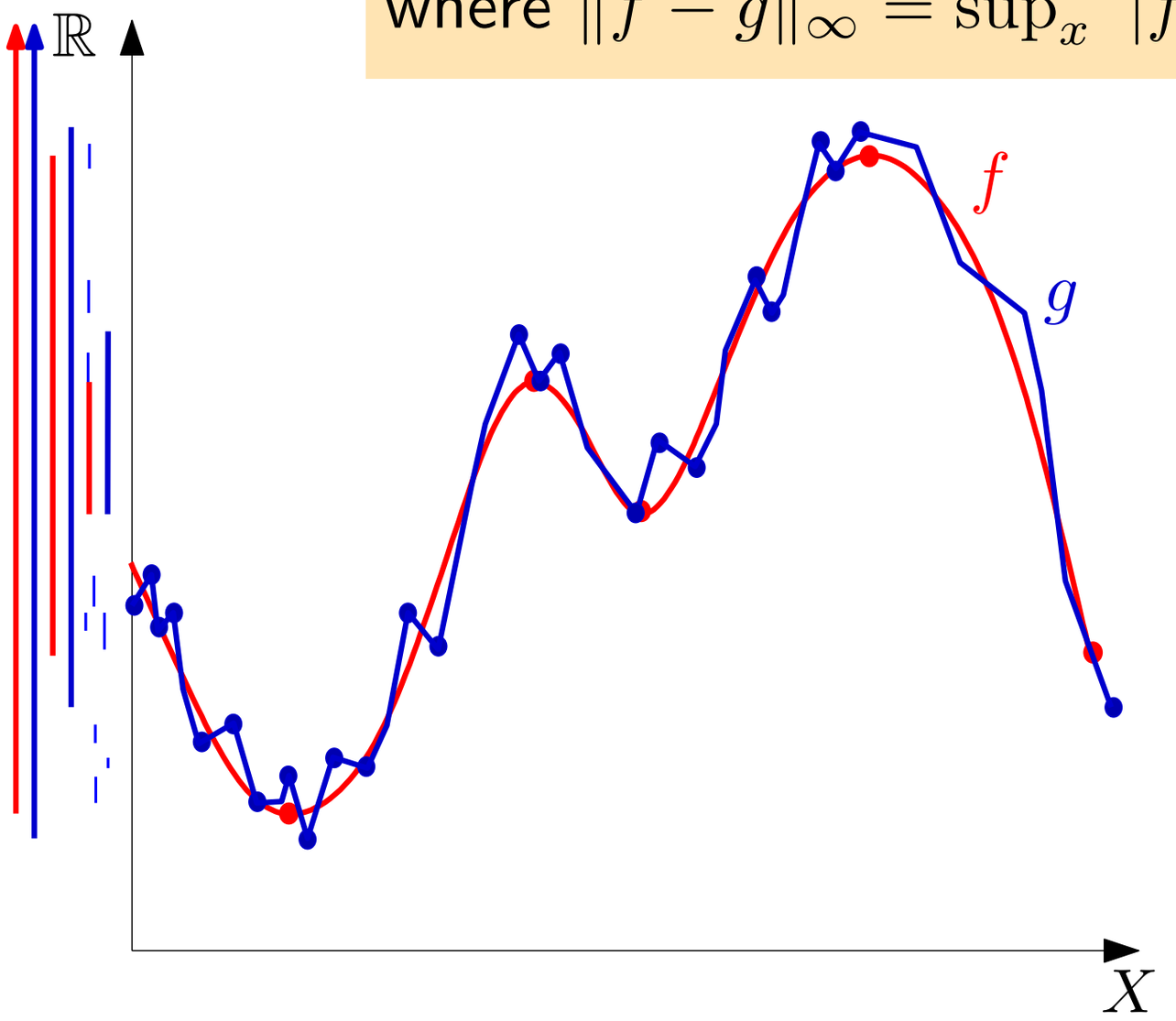
	1	2	3	4	5	6	7
1				$*$			
2				(1)	$*$		
3					(1)		
4							$*$
5							$*$
6							(1)
7							

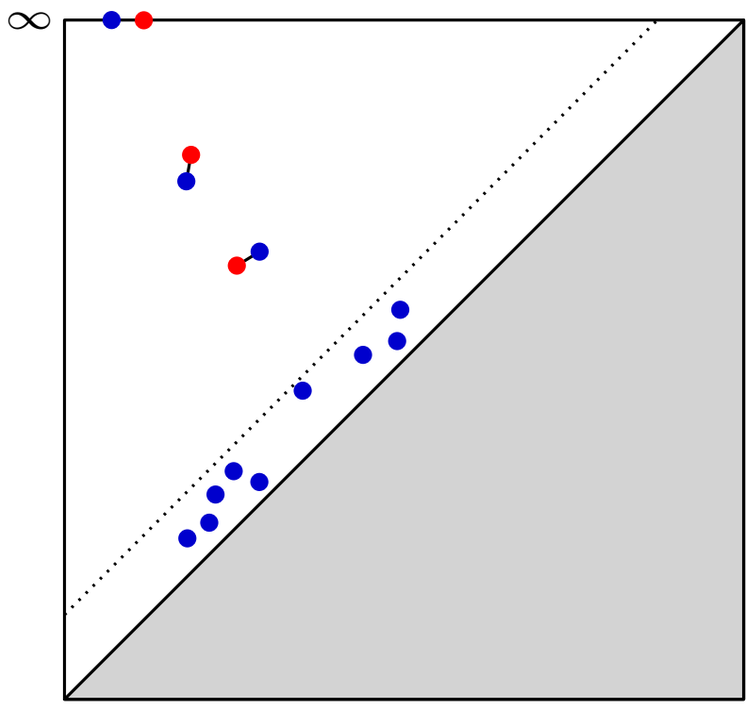
Stability properties

Thm: For any pfd functions $f, g : X \to \mathbb{R}$ and homological dimension k,

$$d_b(D_f, D_g) \leq \|f - g\|_\infty,$$

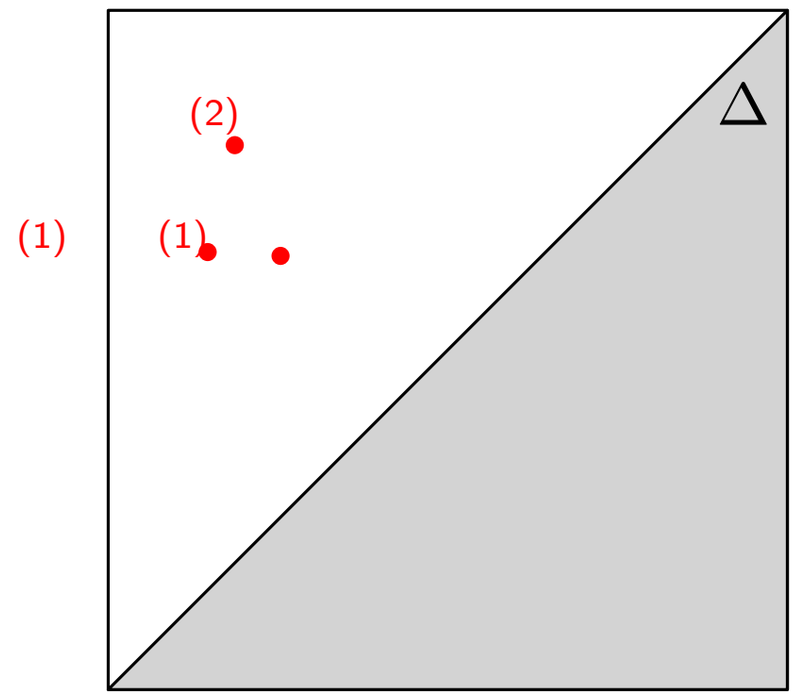
where $\|f - g\|_\infty = \sup_x \ |f(x) - g(x)|$.





Stability properties

Persistence diagram $\equiv$ **finite** multiset in the open half-plane $\Delta \times \mathbb{R}_{>0}$.



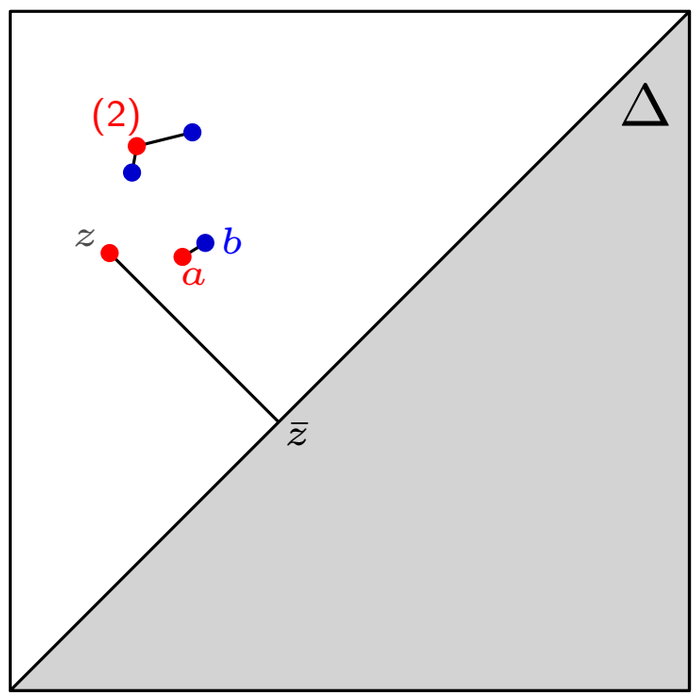
Stability properties

Persistence diagram $\equiv$ **finite** multiset in the open half-plane $\Delta \times \mathbb{R}_{>0}$.

Given a **partial matching** $M : D \leftrightarrow D'$:

- cost of a matched pair $(a, b) \in M$: $c_p(a, b) := \|a - b\|_\infty^p$,
- cost of an unmatched point $c \in D \sqcup D'$: $c_p(c) := \|c - \bar{c}\|_\infty^p$,
- **cost of** M:

$$c_p(M) := \left(\sum_{(a,\, b) \text{ matched}} c_p(a, b) \; + \sum_{c \text{ unmatched}} c_p(c) \right)^{1/p}$$



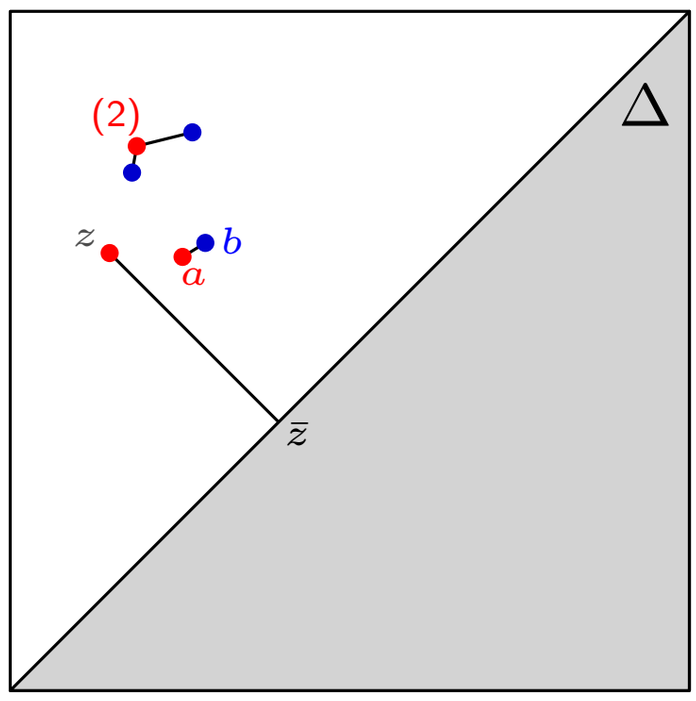
Stability properties

Persistence diagram $\equiv$ **finite** multiset in the open half-plane $\Delta \times \mathbb{R}_{>0}$.

Given a **partial matching** $M : D \leftrightarrow D'$:

- cost of a matched pair $(a, b) \in M$: $c_p(a, b) := \|a - b\|_\infty^p$,
- cost of an unmatched point $c \in D \sqcup D'$: $c_p(c) := \|c - \bar{c}\|_\infty^p$,
- **cost of** M:

$$c_p(M) := \left(\sum_{(a,\, b) \text{ matched}} c_p(a, b) \; + \sum_{c \text{ unmatched}} c_p(c) \right)^{1/p}$$



Def: p-th diagram distance (extended metric):

$$d_p(D, D') := \inf_{M : D \leftrightarrow D'} c_p(M)$$

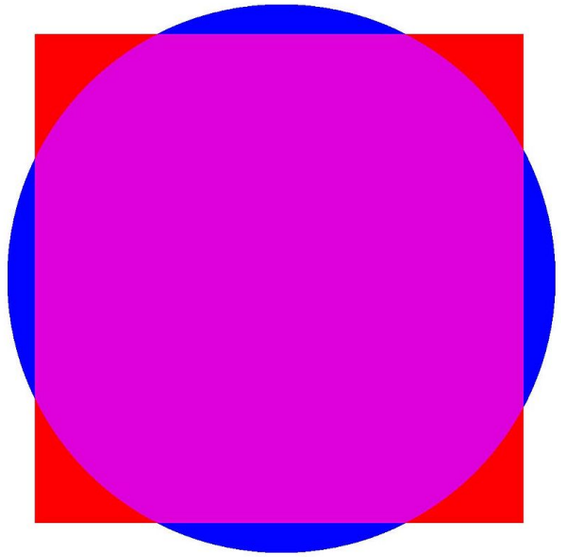
Def: bottleneck distance:

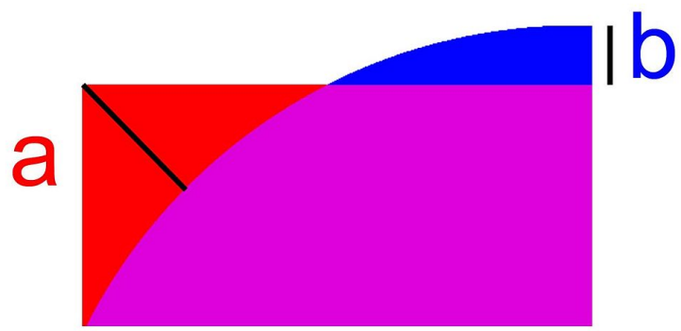
$$d_b(D, D') = d_\infty(D, D') := \lim_{p \to \infty} d_p(D, D')$$

Stability properties for point clouds

Def: The Hausdorff distance between two subspaces X, Y of a common metric space (Z, d) is:

$$d_H(X,Y) = \max\{\sup_{y\in Y} d(y,X), \sup_{x\in X} d(x,Y)\}$$
$$= \max\{\sup_{y\in Y}\inf_{x\in X} d(y,x), \sup_{x\in X}\inf_{y\in Y} d(x,y)\}$$





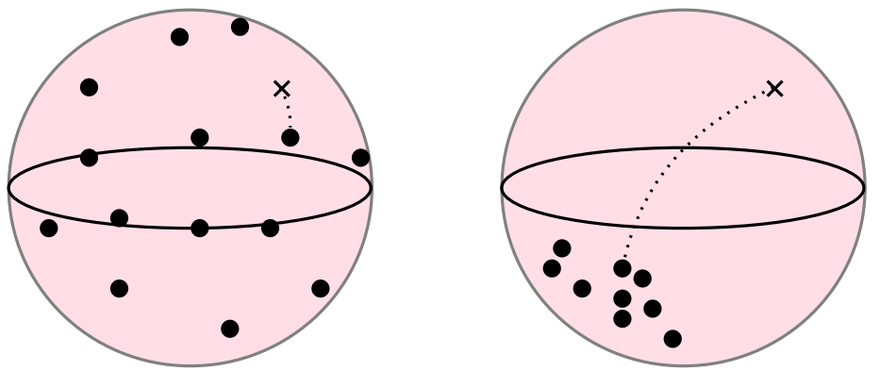
$$d_H(X,Y) = \max\{a,b\}$$

Stability properties for point clouds

Def: The Hausdorff distance between two subspaces X, Y of a common metric space (Z, d) is:

$$d_H(X,Y) = \max\{\sup_{y\in Y} d(y,X), \sup_{x\in X} d(x,Y)\}$$
$$= \max\{\sup_{y\in Y}\inf_{x\in X} d(y,x), \sup_{x\in X}\inf_{y\in Y} d(x,y)\}$$

Ex: Given a sampling $\hat{X}_n \subseteq X$, $d_H(\hat{X}_n, X)$ is a measure of sampling quality.



Q: Show that $d_H(X,Y) = \inf\{\epsilon > 0 : X^\epsilon \subseteq Y \text{ and } Y^\epsilon \subseteq X\}$, where $X^\epsilon = \{z : \exists x \in X \text{ s.t. } d(x,z) \leq \epsilon\}$.

Stability properties for point clouds

Def: The Hausdorff distance between two subspaces X, Y of a common metric space (Z, d) is:

$$d_H(X,Y) = \max\{\sup_{y\in Y} d(y,X), \sup_{x\in X} d(x,Y)\}$$
$$= \max\{\sup_{y\in Y}\inf_{x\in X} d(y,x), \sup_{x\in X}\inf_{y\in Y} d(x,y)\}$$

Def: The Gromov-Hausdorff distance between metric spaces $(X, d_X), (Y, d_Y)$ is the Hausdorff distance of the best common isometric embedding:

$$d_{GH}((X,d_X),(Y,d_Y)) = \inf_\gamma d_H(\gamma(X),\gamma(Y)),$$

where $d(\gamma(x),\gamma(x')) = d_X(x,x')$ and $d(\gamma(y),\gamma(y')) = d_X(y,y')$.

Stability properties for point clouds

Def: The Hausdorff distance between two subspaces X, Y of a common metric space (Z, d) is:

$$d_H(X,Y) = \max\{\sup_{y\in Y} d(y,X), \sup_{x\in X} d(x,Y)\}$$
$$= \max\{\sup_{y\in Y}\inf_{x\in X} d(y,x), \sup_{x\in X}\inf_{y\in Y} d(x,y)\}$$

Def: The Gromov-Hausdorff distance between metric spaces $(X, d_X), (Y, d_Y)$ is metric distortion of the best correspondence:

$$d_{GH}((X,d_X),(Y,d_Y)) = \inf_{\mathcal{C}} \sup_{(x,y),(x',y')\in\mathcal{C}} |d_X(x,x') - d_Y(y,y')|,$$

where $\mathcal{C} \subseteq X \times Y$ s.t. $\forall x$, $\exists y_x \in Y$ s.t. $(x, y_x) \in \mathcal{C}$ (and vice-versa).

Stability properties for point clouds

Def: The Hausdorff distance between two subspaces X, Y of a common metric space (Z, d) is:

$$d_H(X,Y) = \max\{\sup_{y\in Y} d(y,X), \sup_{x\in X} d(x,Y)\}$$
$$= \max\{\sup_{y\in Y}\inf_{x\in X} d(y,x), \sup_{x\in X}\inf_{y\in Y} d(x,y)\}$$

Def: The Gromov-Hausdorff distance between metric spaces $(X, d_X), (Y, d_Y)$ is metric distortion of the best correspondence:

$$d_{GH}((X,d_X),(Y,d_Y)) = \inf_{\mathcal{C}} \sup_{(x,y),(x',y')\in\mathcal{C}} |d_X(x,x') - d_Y(y,y')|,$$

where $\mathcal{C} \subseteq X \times Y$ s.t. $\forall x$, $\exists y_x \in Y$ s.t. $(x, y_x) \in \mathcal{C}$ (and vice-versa).

Thm: If X and Y are common subspaces of a common metric space (Z, d), then

$$d_b(D_{\text{Cech}}(X), D_{\text{Cech}}(Y)) \leq d_H(X,Y).$$

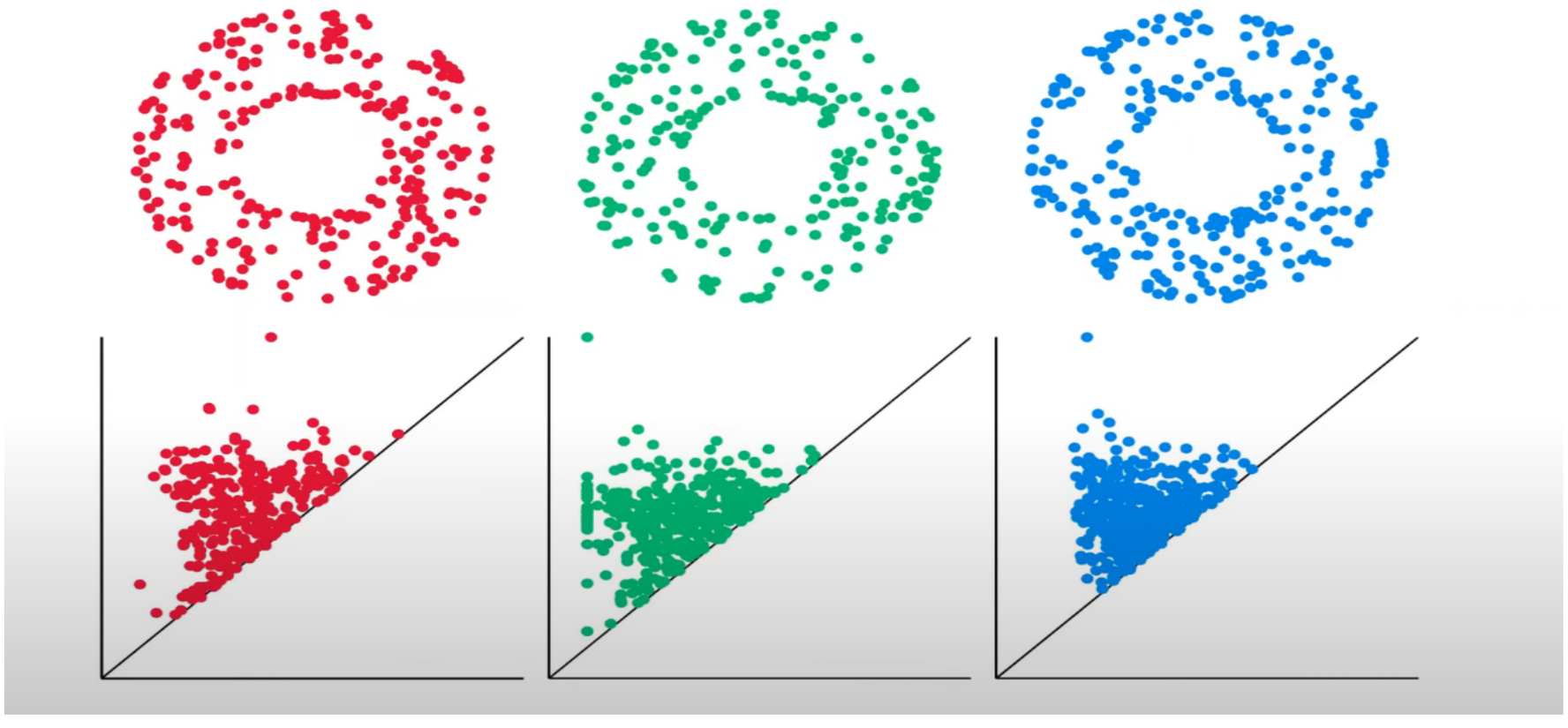
Q: Prove it.

Stability properties for point clouds

[*Persistence stability for geometric complexes*, Chazal, de Silva, Oudot, Geom. Dedicata, 2013].

Thm: If X and Y are pre-compact metric spaces, then

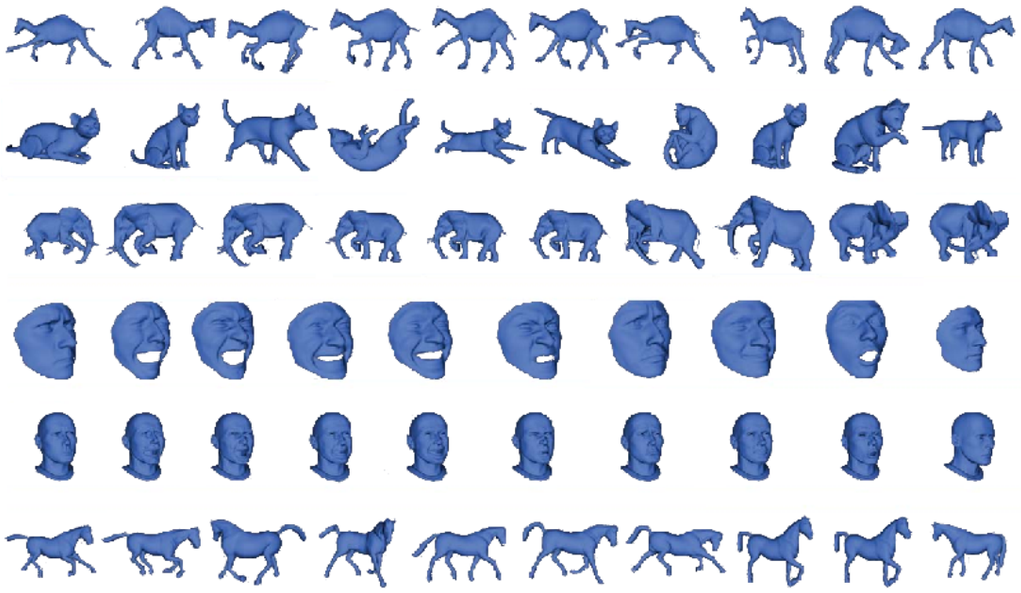
$$d_b(D_{\mathrm{Rips}}(X), D_{\mathrm{Rips}}(Y)) \leq d_{GH}(X, Y).$$

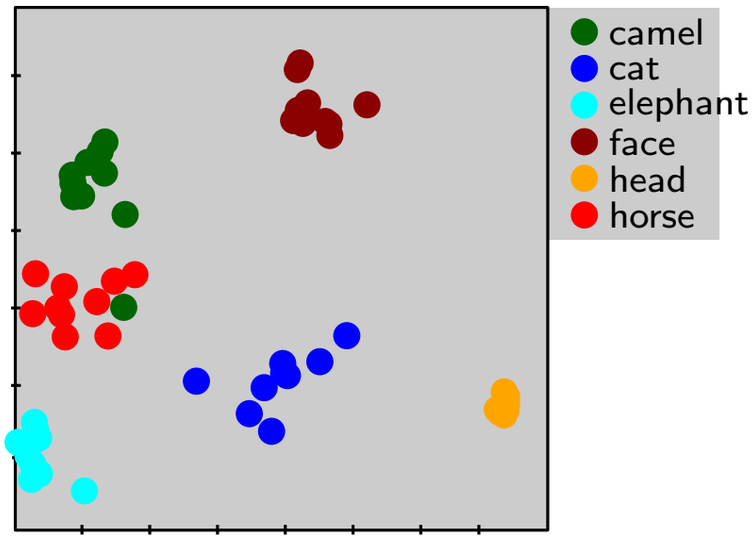


Rem: This result also holds for Čech and other families of filtrations (particular case of a more general theorem).

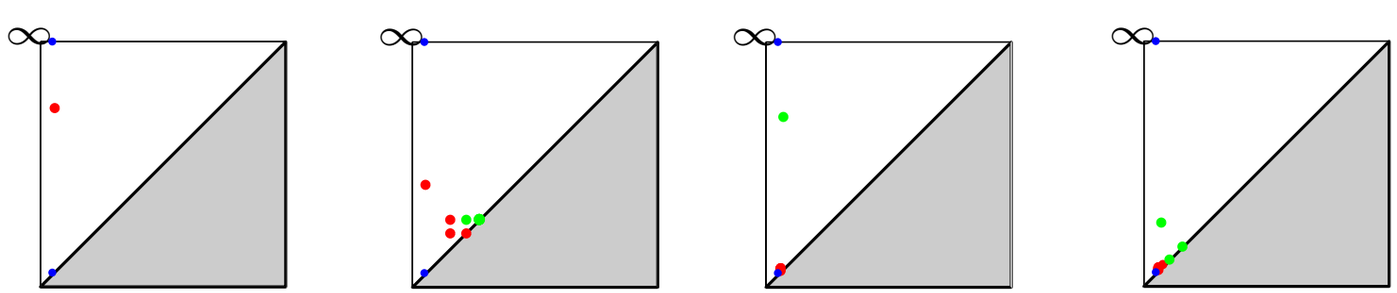
Application: non rigid shape classification

[*Gromov-Hausdorff Stable Signatures for Shapes using Persistence*, Chazal et al., Symp. Geom. Process., 2009]





MDS using bottleneck distance.



- Non rigid shapes in a same class are almost isometric, but computing Gromov-Hausdorff distance between shapes is extremely expensive.
- Compare diagrams of sampled shapes instead of shapes themselves.

Limitations

Thm: If X and Y are pre-compact metric spaces, then

$$d_b(D_{\mathrm{Rips}}(X), D_{\mathrm{Rips}}(Y)) \leq d_{GH}(X, Y).$$

$\rightarrow$ Vietoris-Rips (or Čech, witness) filtrations become quickly prohibitively large as the size of the data increases: $O(|X|^d)$, making the practical computation of persistence almost impossible.

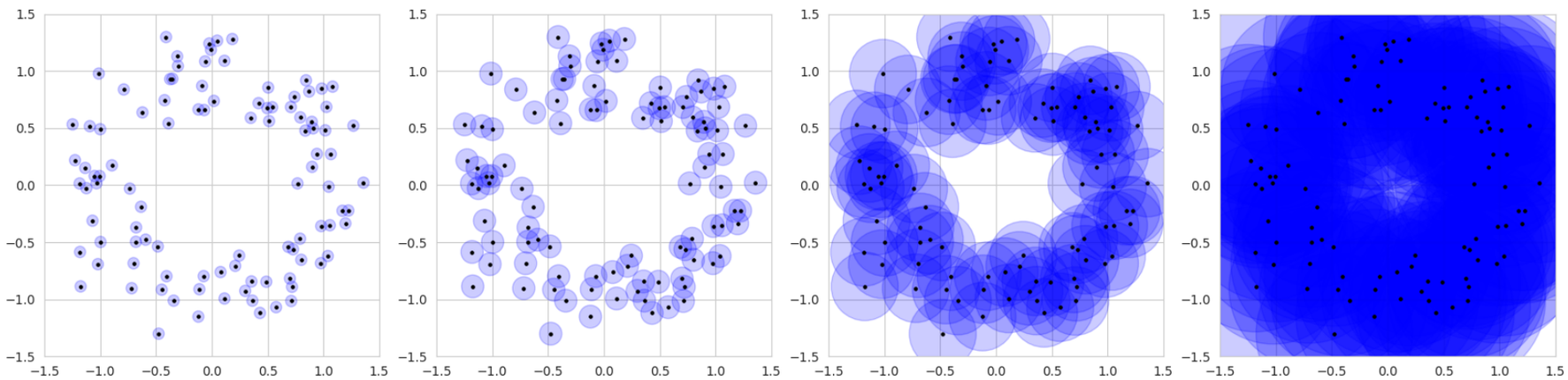
Limitations

Thm: If X and Y are pre-compact metric spaces, then

$$d_b(D_{\mathrm{Rips}}(X), D_{\mathrm{Rips}}(Y)) \leq d_{GH}(X, Y).$$

$\rightarrow$ Vietoris-Rips (or Čech, witness) filtrations become quickly prohibitively large as the size of the data increases: $O(|X|^d)$, making the practical computation of persistence almost impossible.

$\rightarrow$ Persistence diagrams of Vietoris-Rips (as well as Čech, witness,..) filtrations and Gromov-Hausdorff distance are very sensitive to noise and outliers.



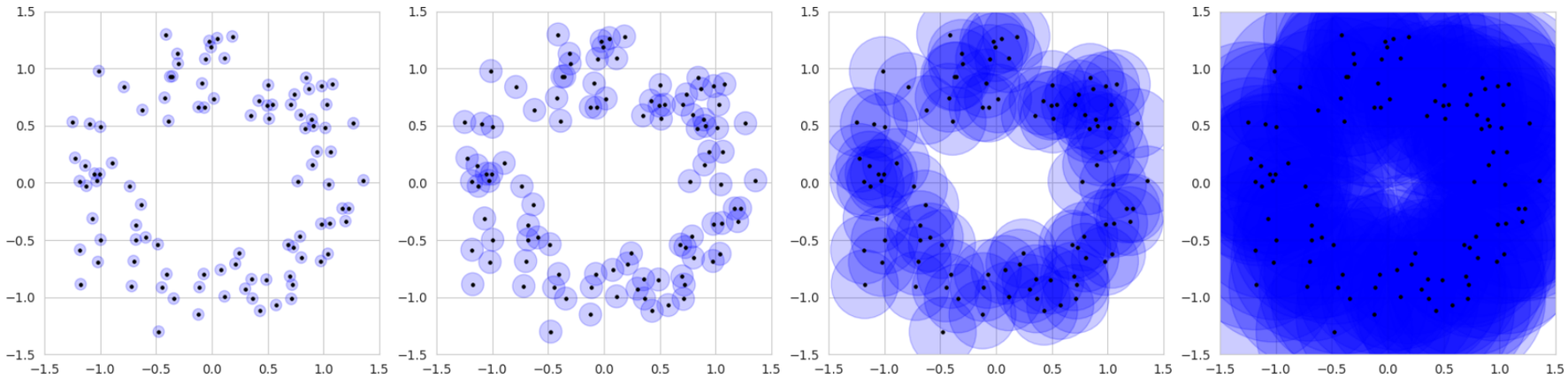
Limitations

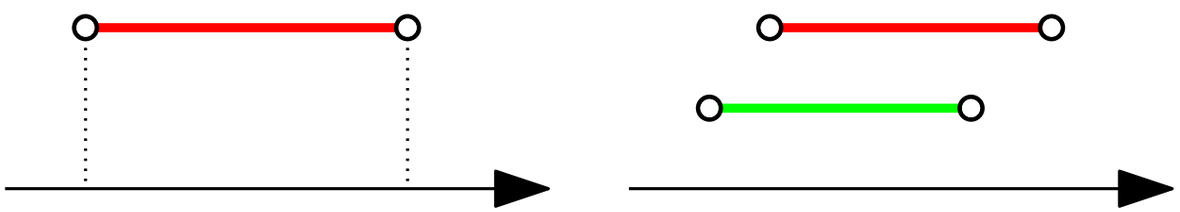
Thm: If X and Y are pre-compact metric spaces, then

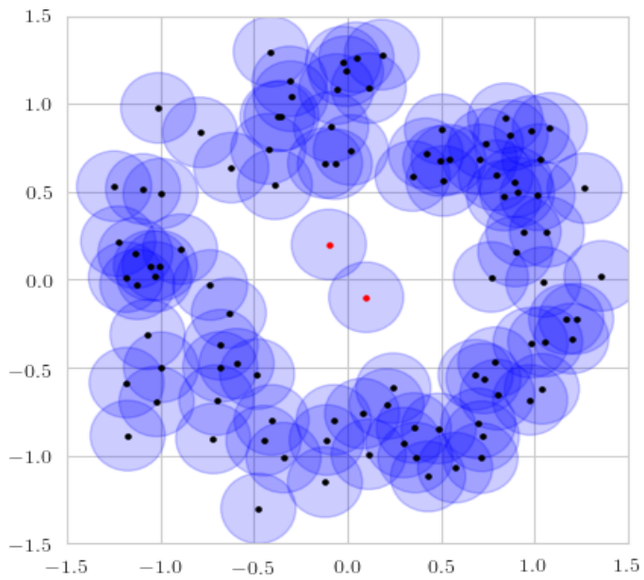
$$d_b(D_{\mathrm{Rips}}(X), D_{\mathrm{Rips}}(Y)) \leq d_{GH}(X, Y).$$

$\rightarrow$ Vietoris-Rips (or Čech, witness) filtrations become quickly prohibitively large as the size of the data increases: $O(|X|^d)$, making the practical computation of persistence almost impossible.

$\rightarrow$ Persistence diagrams of Vietoris-Rips (as well as Čech, witness,..) filtrations and Gromov-Hausdorff distance are very sensitive to noise and outliers.







The Distance To Measure (DTM)

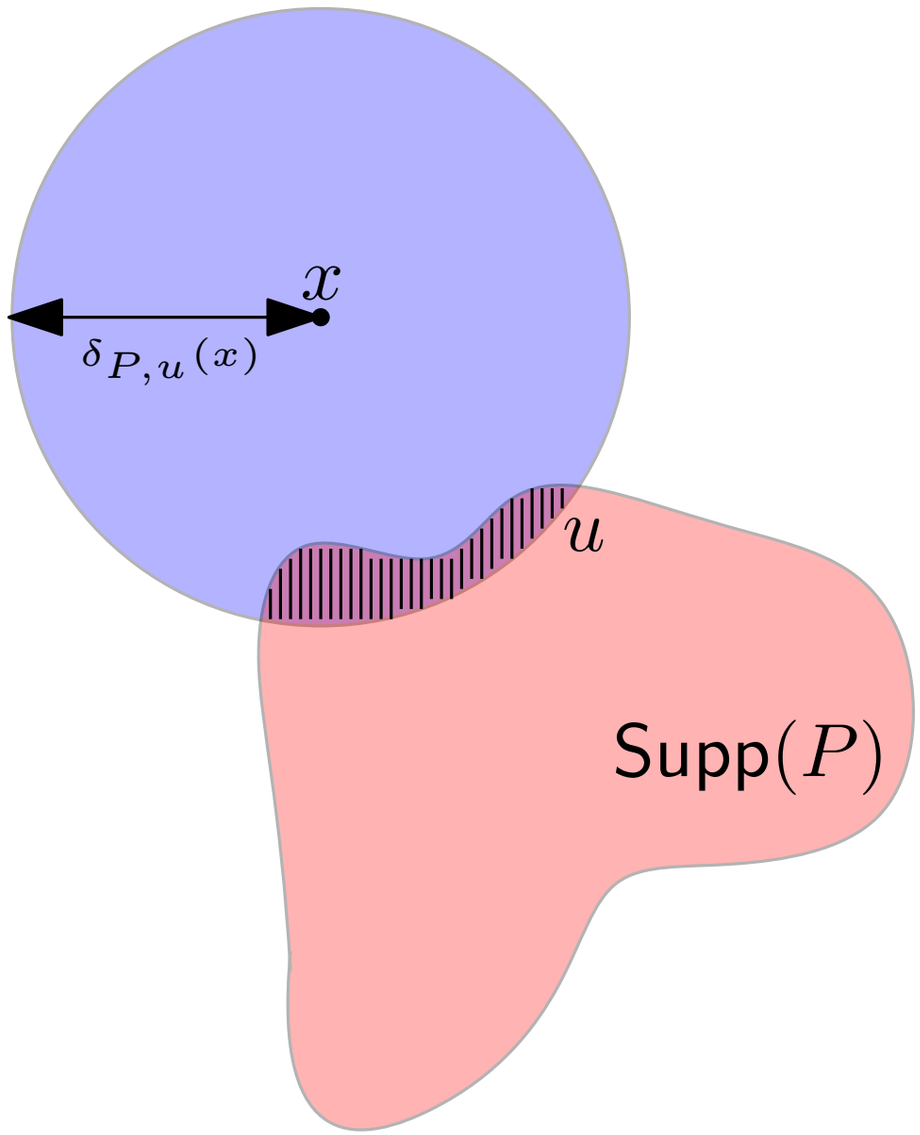
[*Geometric inference for probability measures*, Chazal, Cohen-Steiner, Mérigot, Found. Comput. Math., 2011]

The Distance To Measure (DTM)

[*Geometric inference for probability measures*, Chazal, Cohen-Steiner, Mérigot, Found. Comput. Math., 2011]

Preliminary distance function to a measure P: let $u \in]0,1[$ be a positive mass, and P a probability measure on $\mathbb{R}^d$:

$$\delta_{P,u}(x) = \inf\{r > 0 : P(B(x,r)) \geq u\}$$

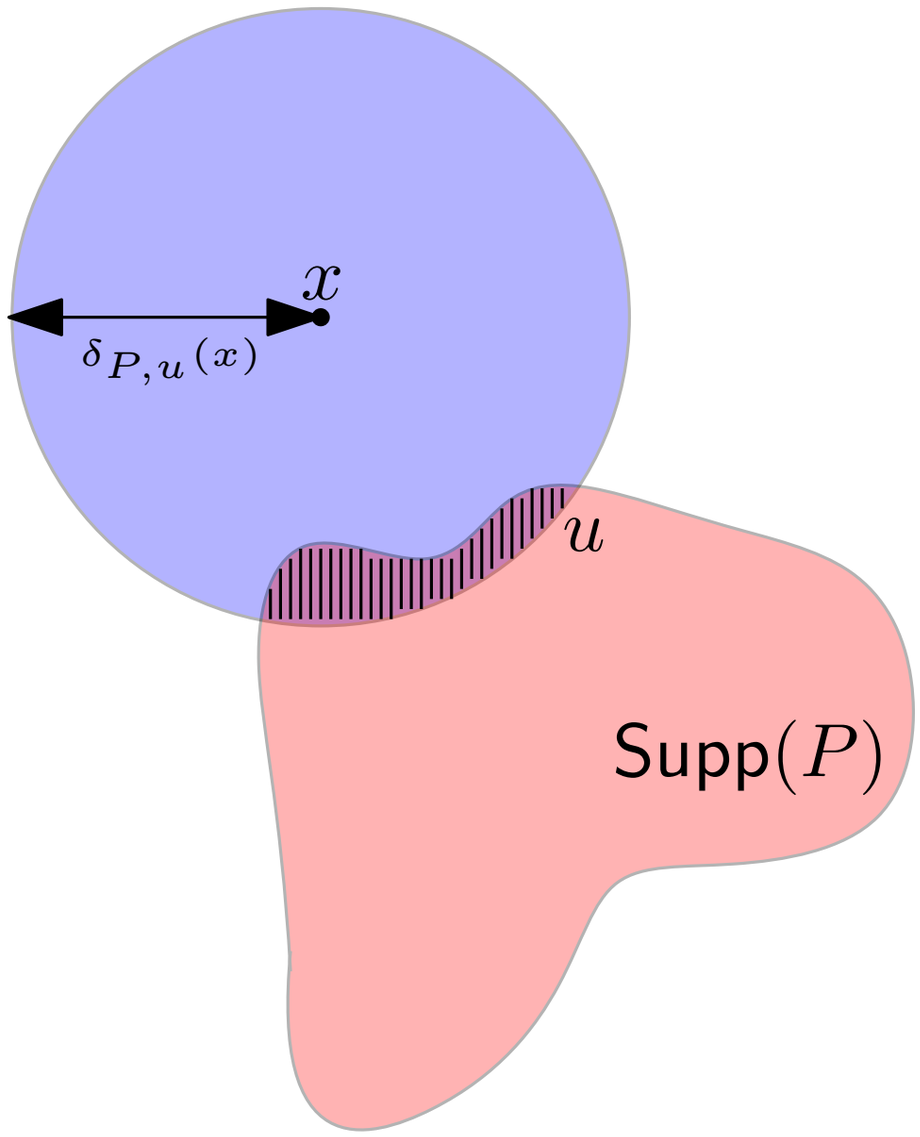


The Distance To Measure (DTM)

[*Geometric inference for probability measures*, Chazal, Cohen-Steiner, Mérigot, Found. Comput. Math., 2011]

Preliminary distance function to a measure P: let $u \in]0,1[$ be a positive mass, and P a probability measure on $\mathbb{R}^d$:

$$\delta_{P,u}(x) = \inf\{r > 0 : P(B(x,r)) \geq u\}$$



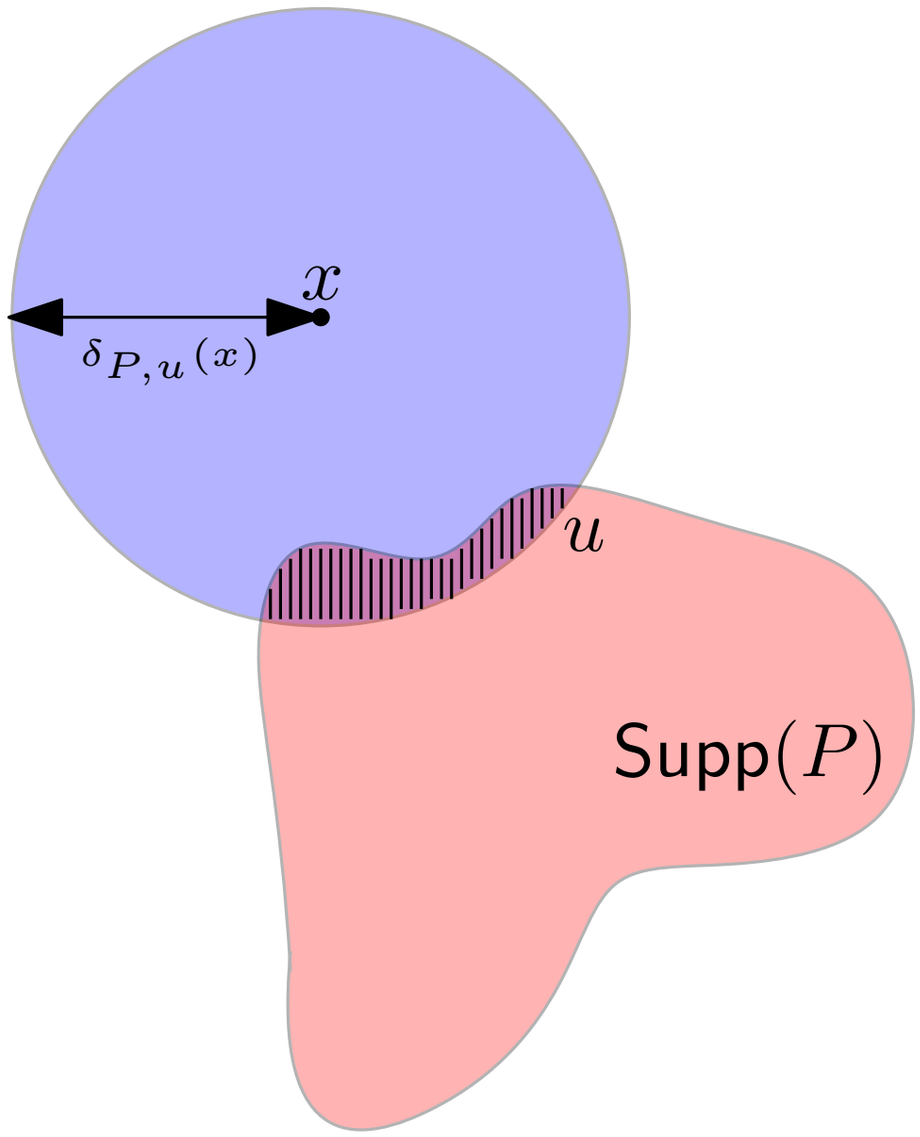
$\delta_{P,u}$ is the smallest distance needed to capture a mass of at least u.

The Distance To Measure (DTM)

[*Geometric inference for probability measures*, Chazal, Cohen-Steiner, Mérigot, Found. Comput. Math., 2011]

Preliminary distance function to a measure P: let $u \in]0,1[$ be a positive mass, and P a probability measure on $\mathbb{R}^d$:

$$\delta_{P,u}(x) = \inf\{r > 0 : P(B(x,r)) \geq u\}$$



$\delta_{P,u}$ is the smallest distance needed to capture a mass of at least u.

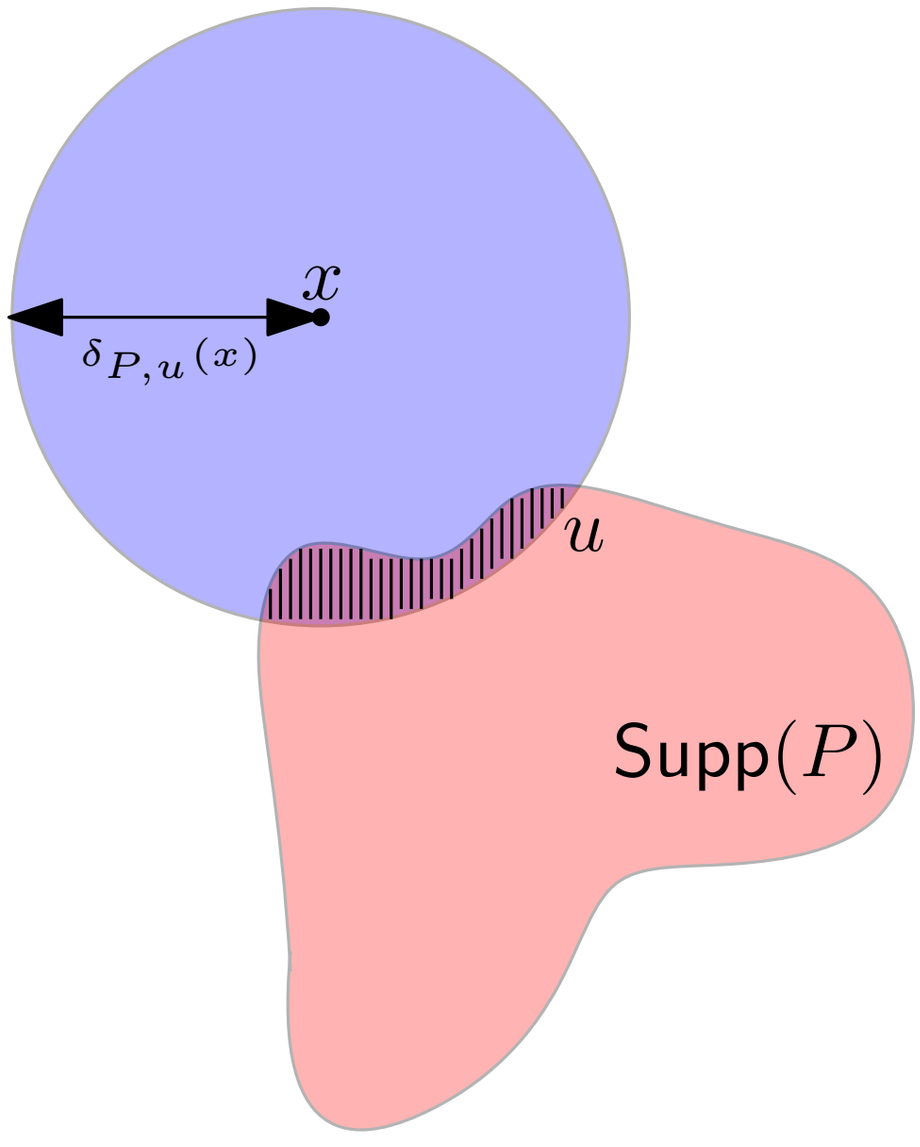
$\delta_{P,u}$ is the quantile function at u of the r.v. $\|x - X\|$ where $X \sim P$.

The Distance To Measure (DTM)

[*Geometric inference for probability measures*, Chazal, Cohen-Steiner, Mérigot, Found. Comput. Math., 2011]

Preliminary distance function to a measure P: let $u \in]0,1[$ be a positive mass, and P a probability measure on $\mathbb{R}^d$:

$$\delta_{P,u}(x) = \inf\{r > 0 : P(B(x,r)) \geq u\}$$



Def: Given a probability measure P on $\mathbb{R}^d$ and $m > 0$, the distance function to the measure P (DTM) is defined by

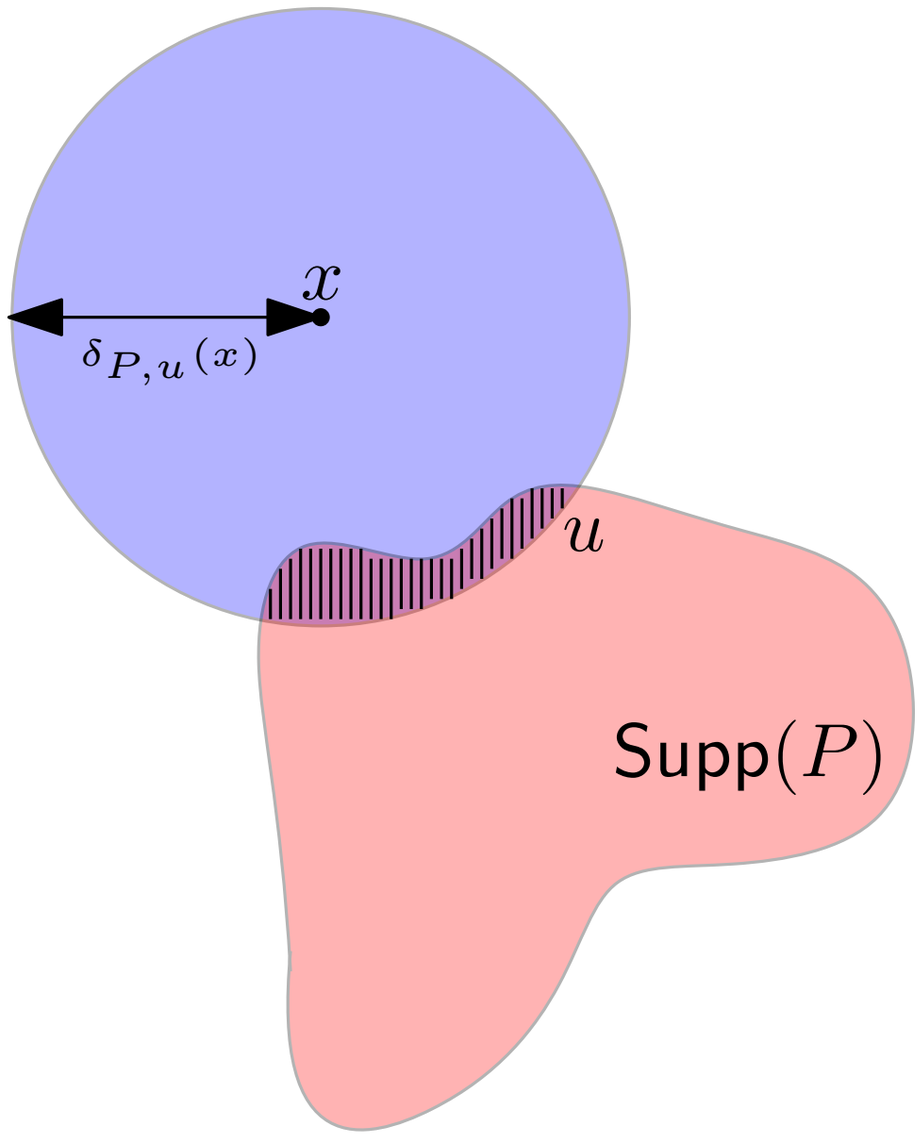
$$d_{P,m} : x \in \mathbb{R} \mapsto \left(\frac{1}{m}\int_0^m \delta_{P,u}^2(x)\mathrm{d}u\right)^{1/2}$$

The Distance To Measure (DTM)

[*Geometric inference for probability measures*, Chazal, Cohen-Steiner, Mérigot, Found. Comput. Math., 2011]

Preliminary distance function to a measure P: let $u \in]0,1[$ be a positive mass, and P a probability measure on $\mathbb{R}^d$:

$$\delta_{P,u}(x) = \inf\{r > 0 : P(B(x,r)) \geq u\}$$



Def: Given a probability measure P on $\mathbb{R}^d$ and $m > 0$, the distance function to the measure P (DTM) is defined by

$$d_{P,m} : x \in \mathbb{R} \mapsto \left(\frac{1}{m}\int_0^m \delta_{P,u}^2(x)\mathrm{d}u\right)^{1/2}$$

The DTM is robust, i.e., stable under Wasserstein perturbations:

$$\|d_{P,m} - d_{Q,m}\|_\infty \leq \frac{1}{\sqrt{m}} W_2(P,Q)$$

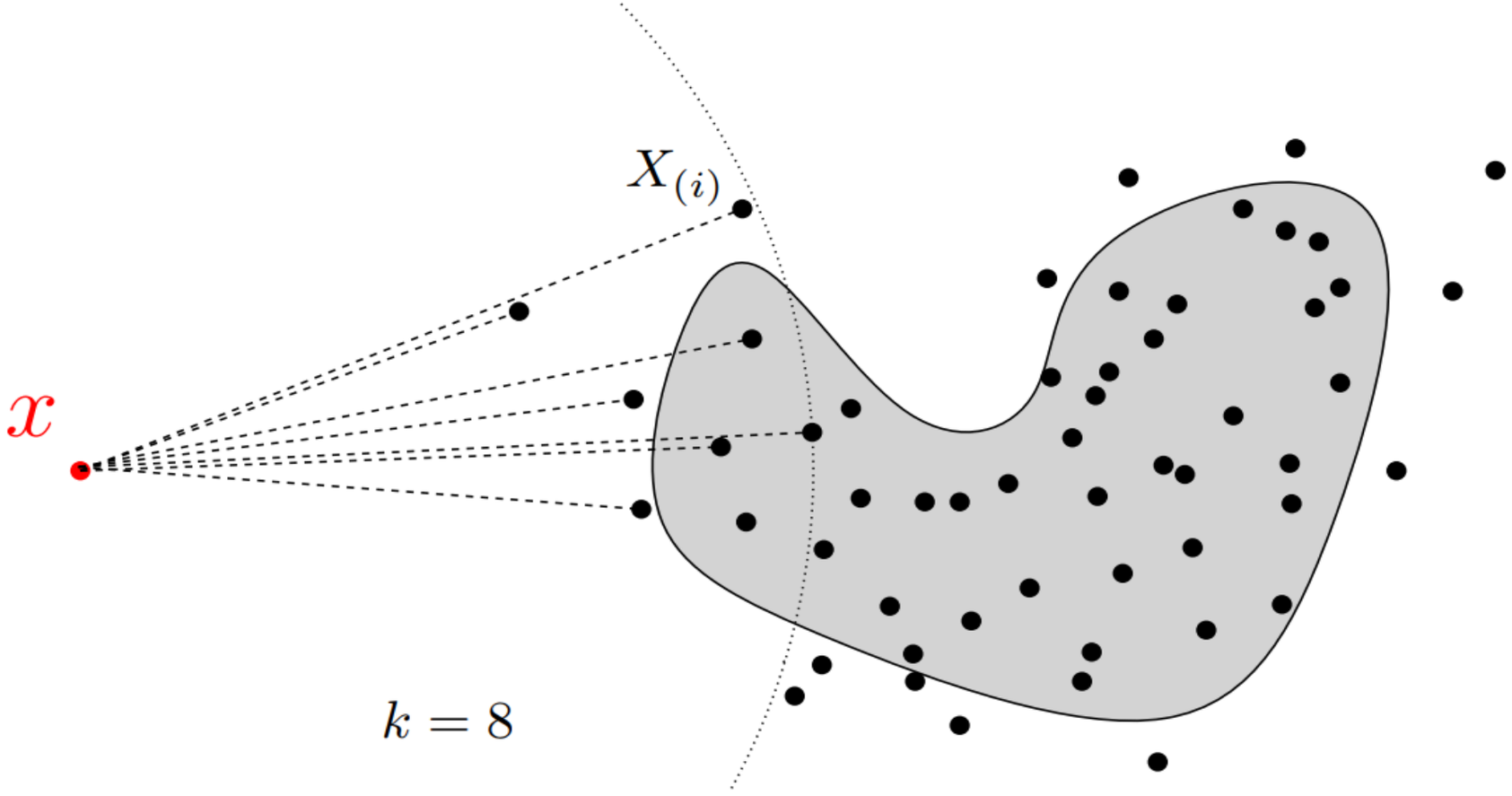
The Distance To Measure (DTM)

[*Geometric inference for probability measures*, Chazal, Cohen-Steiner, Mérigot, Found. Comput. Math., 2011]

Def: Let $X_1, \ldots, X_n$ sampled according to P and let P_n be the empirical measure. Then

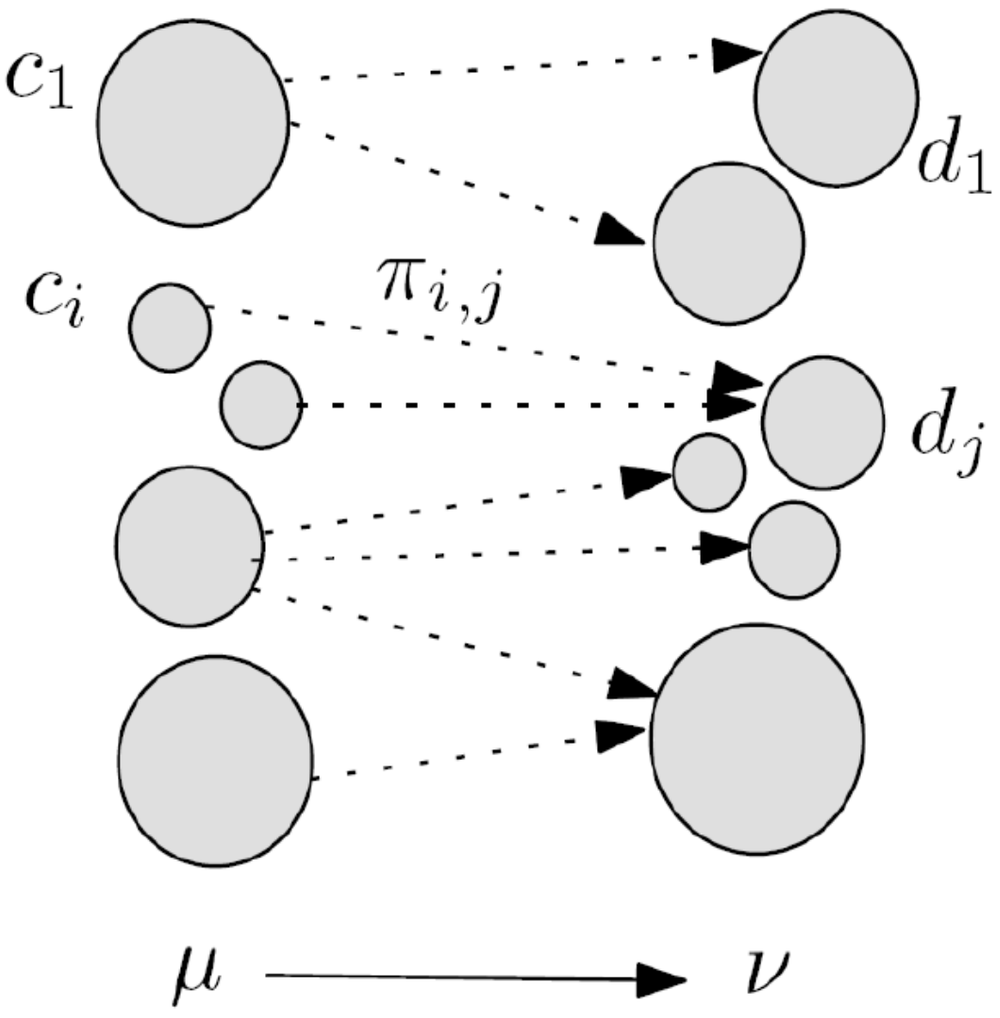
$$d_{P_n,k/n}(x) = \frac{1}{k} \sum_{i=1}^{k} \|x - X_{(i)}\|^2,$$

where $\|X_{(1)} - x\| \leq \|X_{(2)} - x\| \leq \cdots \leq \|X_{(k)} - x\| \leq \cdots \leq \|X_{(n)} - x\|$.



The Wasserstein distance

Let (X, d) be a metric space and let μ, ν be probability measures on X with finite p-moments ($p \geq 1$). The Wasserstein distance $W_p(\mu, \nu)$ quantifies the optimal cost of pushing μ onto ν, the cost of moving a small mass dx from x to y being $d(x, y)^p dx$.

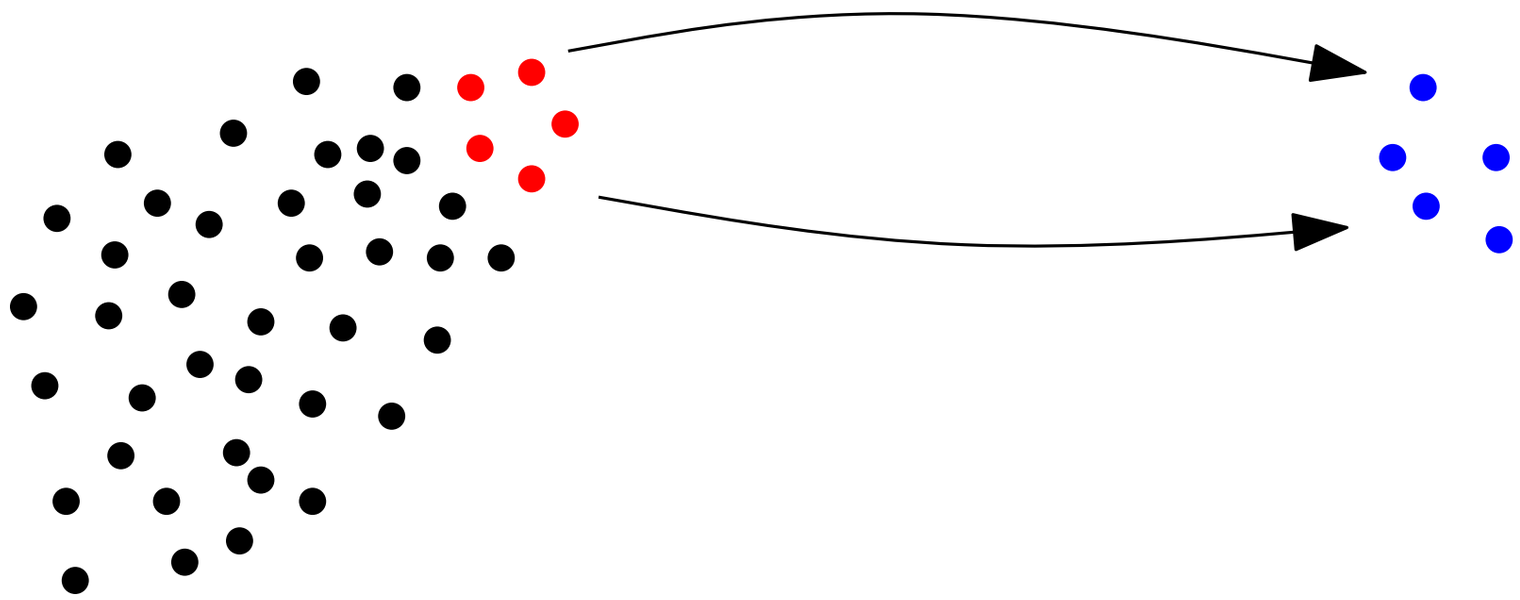


- Transport plan: Π a probability measure on $X \times X$ s.t. $\Pi(A \times \mathbb{R}^d) = \mu(A)$ and $\Pi(\mathbb{R}^d \times B) = \nu(B)$ for any borelian sets $A, B \subseteq X$.
- Cost of a transport plan:

$$C(\Pi) = \left(\int_{X \times X} d(x, y)^p d\Pi(x, y) \right)^{\frac{1}{p}}$$

- $W_p(\mu, \nu) = \inf_\Pi C(\Pi)$.

The Wasserstein distance



Ex: If $P = \{p_1, \ldots, p_n\}$ is a point cloud, and $P' = \{p_1, \ldots, p_{n-k-1}, o_1, \ldots, o_k\}$ with $d(o_i, P) = R$, then

$$d_H(P, P') \geq R \qquad \text{but} \qquad W_2(\mu_P, \mu_{P'}) \leq \sqrt{\frac{k}{n}}(R + \operatorname{diam}(P))$$

DTM-based filtrations

[*DTM-based filtrations*, Anai et al., Symp. Comp. Geom., 2019]

Def: Let V be a point cloud (in a metric space). The DTM-based complex $W(V)$ is the filtered simplicial complex indexed by $\mathbb{R}$ whose vertex set is V and whose other simplices are defined with

$$\sigma = [p_0, p_1 \ldots, p_k] \in W(V, \alpha) \iff \cap_{i=0}^{k} B(p_i, r_{p_i}(\alpha)) \neq \emptyset$$

where $r_p(\alpha) = 0$ if $\alpha \leq d_{P_n,k/n}(p)$ and $|\alpha^q - d_{P_n,k/n}(p)^q|^{1/q}$ otherwise.

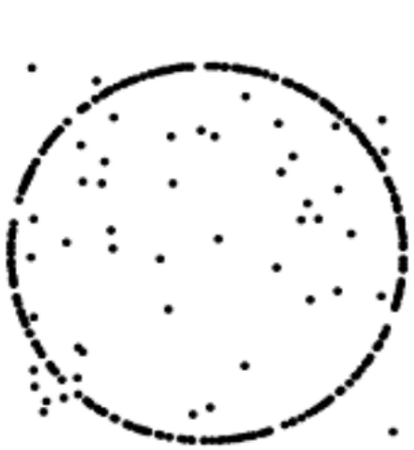
DTM-based filtrations

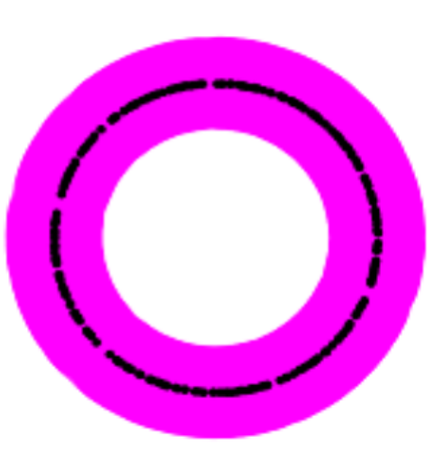
[*DTM-based filtrations*, Anai et al., Symp. Comp. Geom., 2019]

Def: Let V be a point cloud (in a metric space). The DTM-based complex $W(V)$ is the filtered simplicial complex indexed by $\mathbb{R}$ whose vertex set is V and whose other simplices are defined with

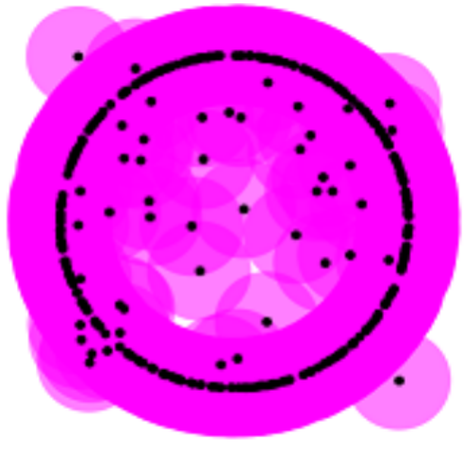
$$\sigma = [p_0, p_1 \ldots, p_k] \in W(V, \alpha) \iff \cap_{i=0}^{k} B(p_i, r_{p_i}(\alpha)) \neq \emptyset$$

where $r_p(\alpha) = 0$ if $\alpha \leq d_{P_n,k/n}(p)$ and $|\alpha^q - d_{P_n,k/n}(p)^q|^{1/q}$ otherwise.

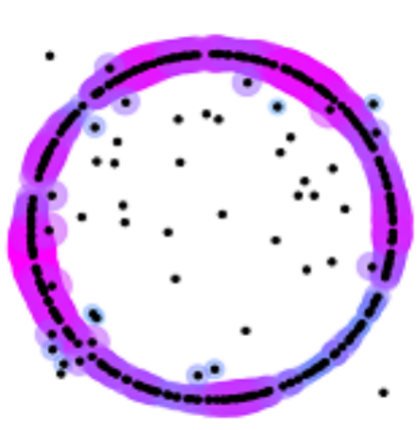




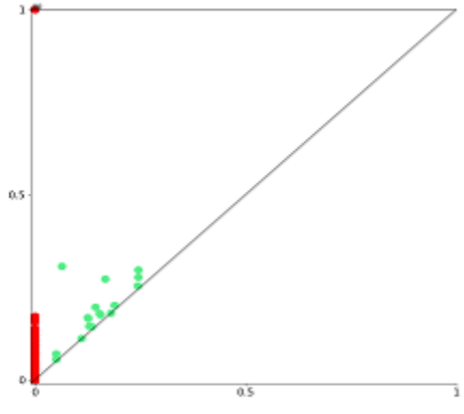
Rips

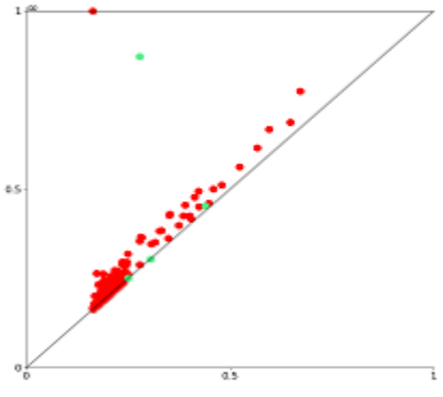


Rips



DTM-based





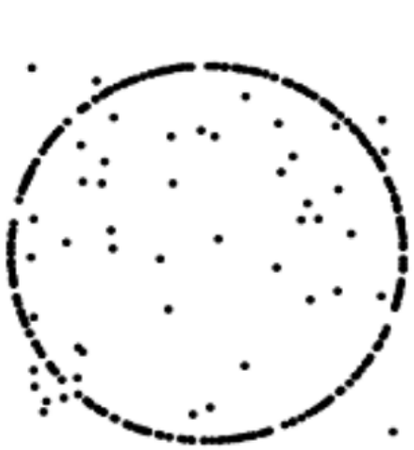
DTM-based filtrations

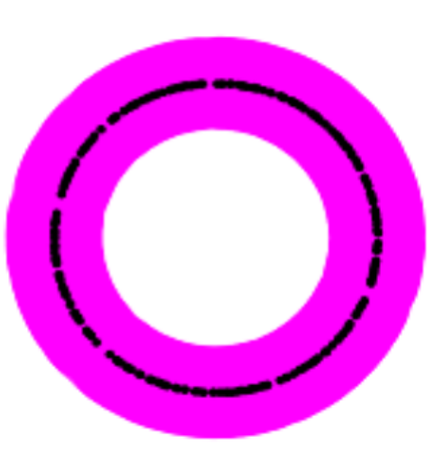
[*DTM-based filtrations*, Anai et al., Symp. Comp. Geom., 2019]

Def: Let V be a point cloud (in a metric space). The DTM-based complex $W(V)$ is the filtered simplicial complex indexed by $\mathbb{R}$ whose vertex set is V and whose other simplices are defined with

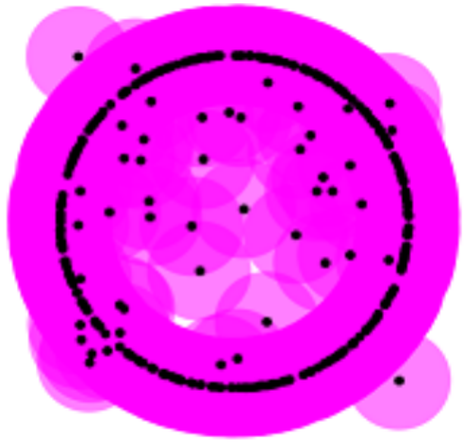
$$\sigma = [p_0, p_1 \ldots, p_k] \in W(V, \alpha) \iff \cap_{i=0}^{k} B(p_i, r_{p_i}(\alpha)) \neq \emptyset$$

where $r_p(\alpha) = 0$ if $\alpha \leq d_{P_n,k/n}(p)$ and $|\alpha^q - d_{P_n,k/n}(p)^q|^{1/q}$ otherwise.

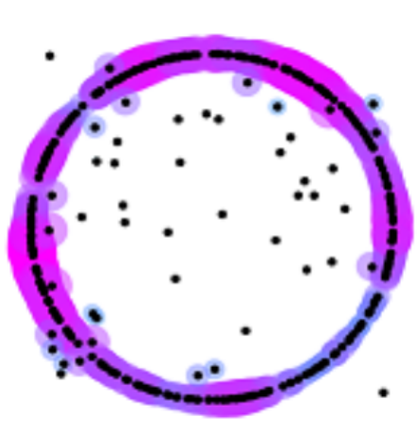




Rips

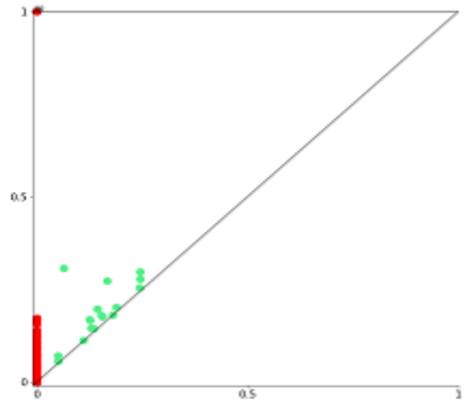


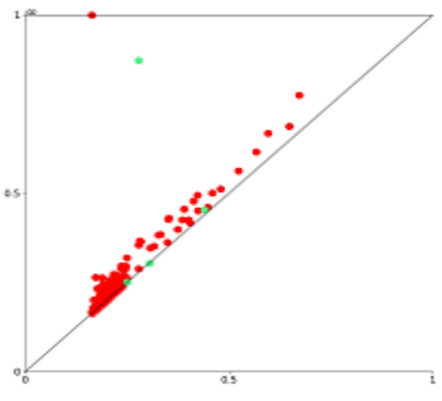
Rips



DTM-based

Thm: $d_b(D_W(X), D_W(Y)) \leq \sqrt{\frac{n}{k}} W_2(X, Y) + 2^{1/q} d_H(X, Y)$.

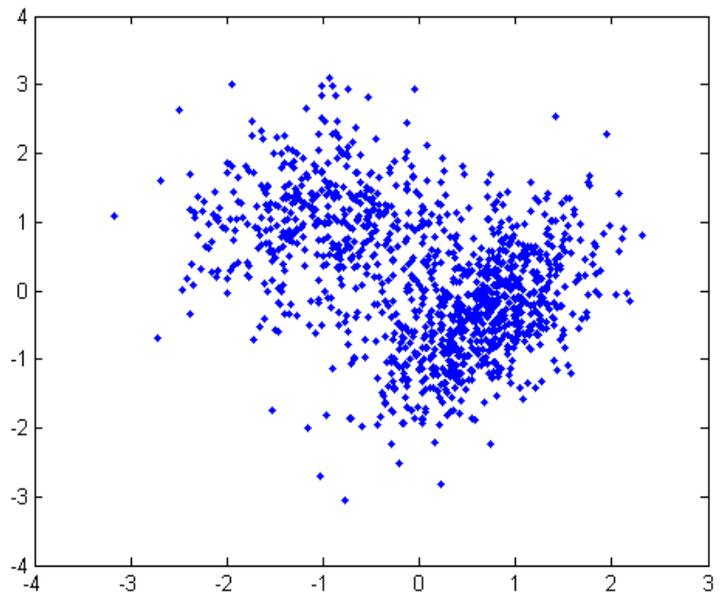


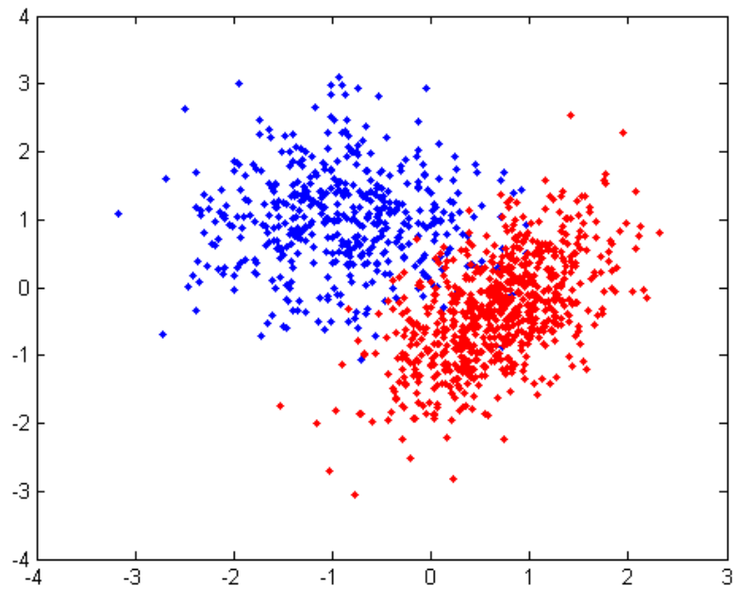


Clustering and 0-dimensional Persistent Homology

Clustering: A partition of data into groups of similar observations. The observations in each group (cluster) are similar to each other and dissimilar to observations from other groups.

Input: a finite set of observations: point cloud embedded in an Euclidean space (with coordinates) or a more general metric space (pairwise distance/similarity) matrix.





Goal: partition the data into a relevant family of subsets (clusters).

Clustering and 0-dimensional Persistent Homology

Clustering: A partition of data into groups of similar observations. The observations in each group (cluster) are similar to each other and dissimilar to observations from other groups.

Not a single/universal notion of cluster.

A wealth of approaches:

- Variational
- Spectral
- Density based
- Hierarchical
- etc...

Clustering and 0-dimensional Persistent Homology

Clustering: A partition of data into groups of similar observations. The observations in each group (cluster) are similar to each other and dissimilar to observations from other groups.

Not a single/universal notion of cluster.

A wealth of approaches:

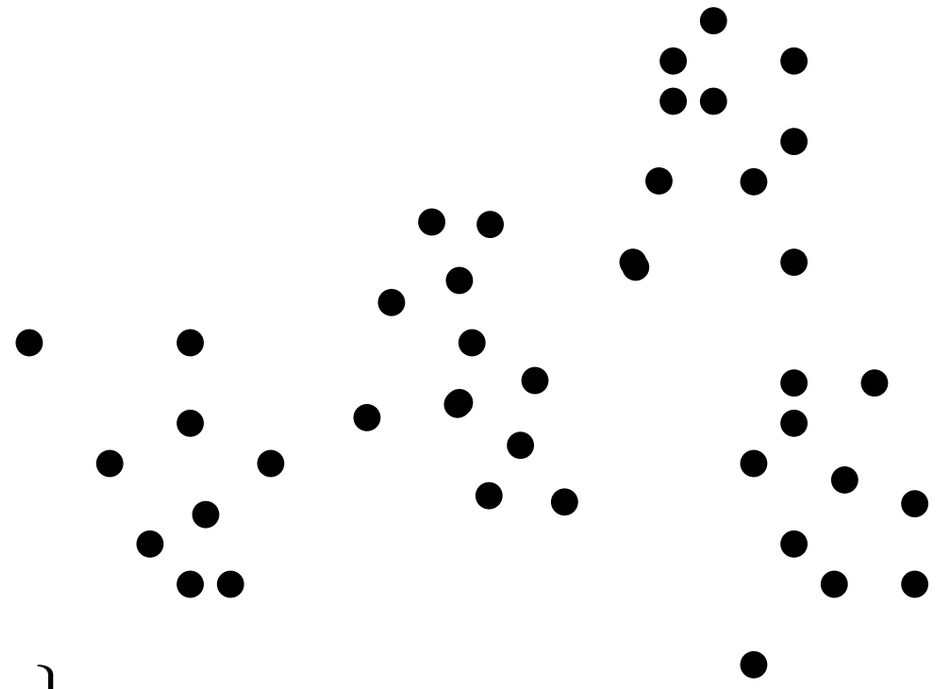
- Variational
- Spectral
- Density based
- Hierarchical
- etc...

In this lecture:

- a few basic classical "algorithms" motivating and bringing us to the introduction of persistent homology.

The k-means algorithm

Input: A (large) set of N points X and an integer $k < N$.

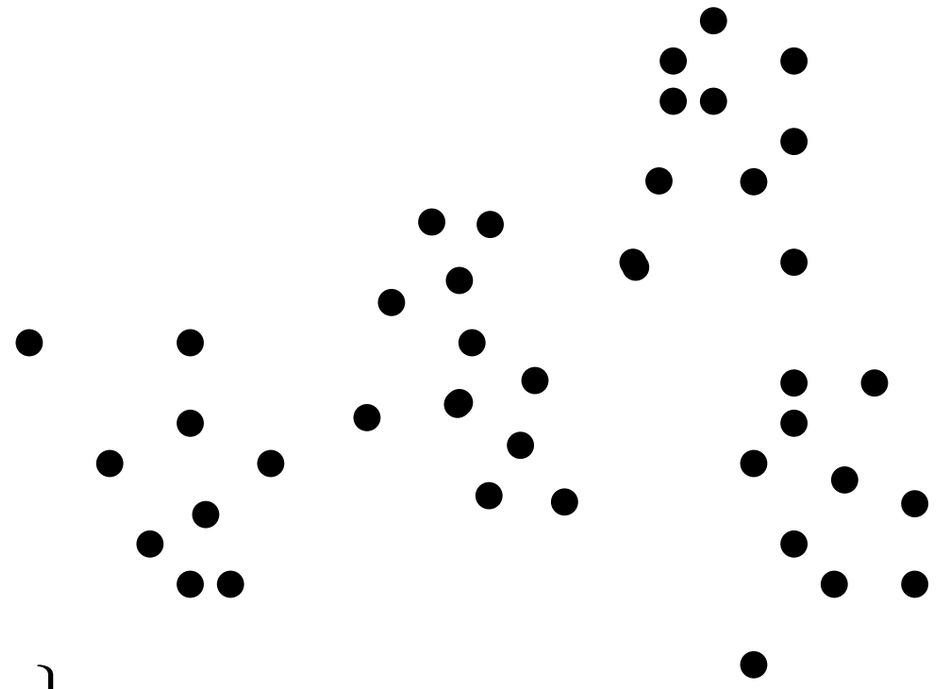


Goal: Find a set of k points $L = \{y_1, \ldots, y_k\}$ that minimizes

$$E = \sum_{i=1}^{N} d(x_i, L)^2$$

The k-means algorithm

Input: A (large) set of N points X and an integer $k < N$.



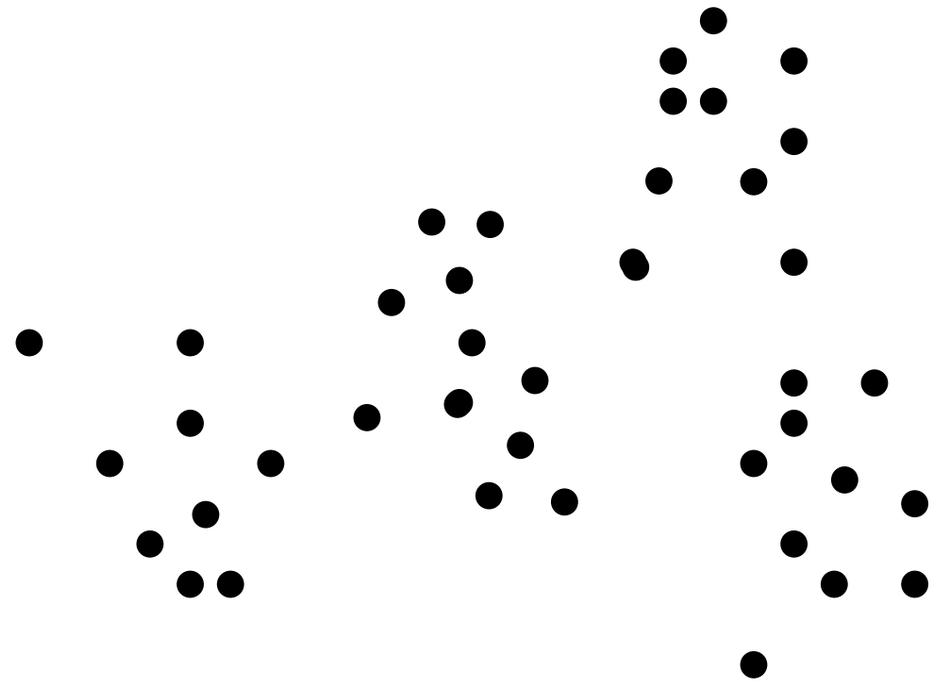
Goal: Find a set of k points $L = \{y_1, \ldots, y_k\}$ that minimizes

$$E = \sum_{i=1}^{N} d(x_i, L)^2$$

This is a NP hard problem!

Lloyd's algorithm: a very simple local search algorithm $\rightarrow$ local minimum.

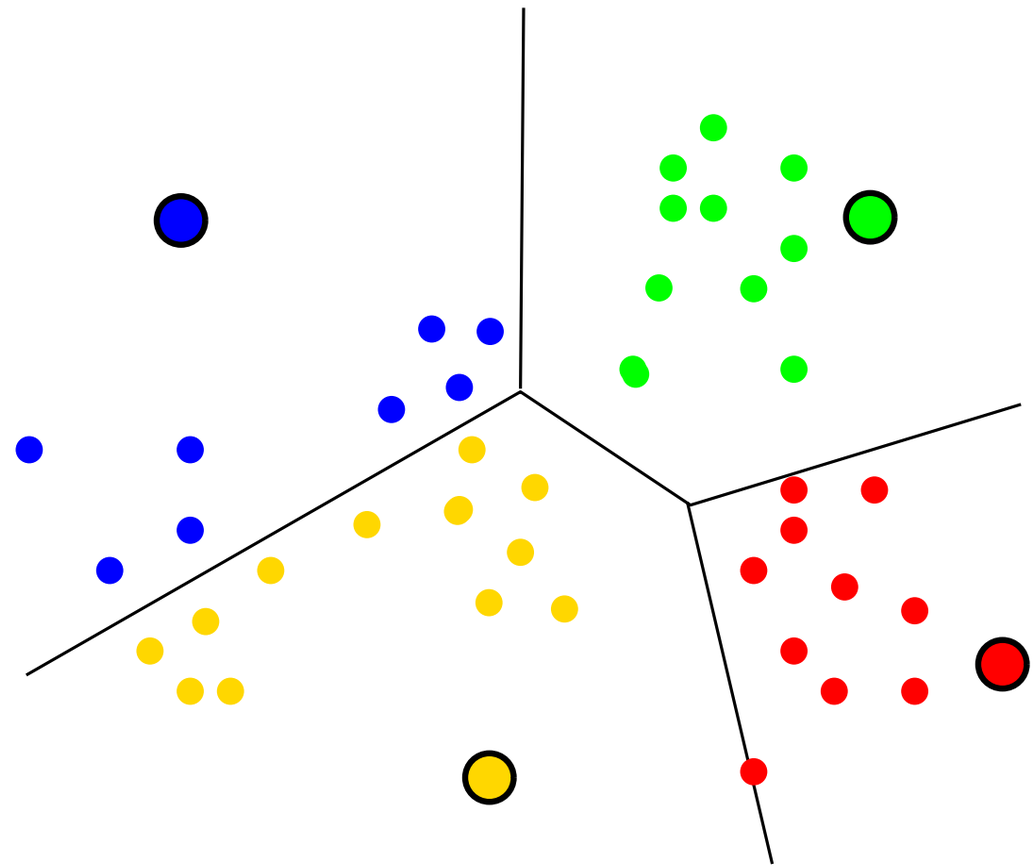
The k-means algorithm



Lloyd's algorithm

- Select $L^1 = \{y_1^1, \ldots, y_k^1\}$ initial 'seeds';
- $i = 1$;
- Repeat
  - For $(j = 1; j \leq k; j++)$ $S_j^i = \{x \in X : d(x, y_j^i) = d(x, L^i)\}$;
  - For $(j = 1; j \leq k; j++)$ $y_j^{i+1} = \frac{1}{|S_j^i|} \sum_{x \in S_j^i} x$;
  - $i++$;
- Until convergence

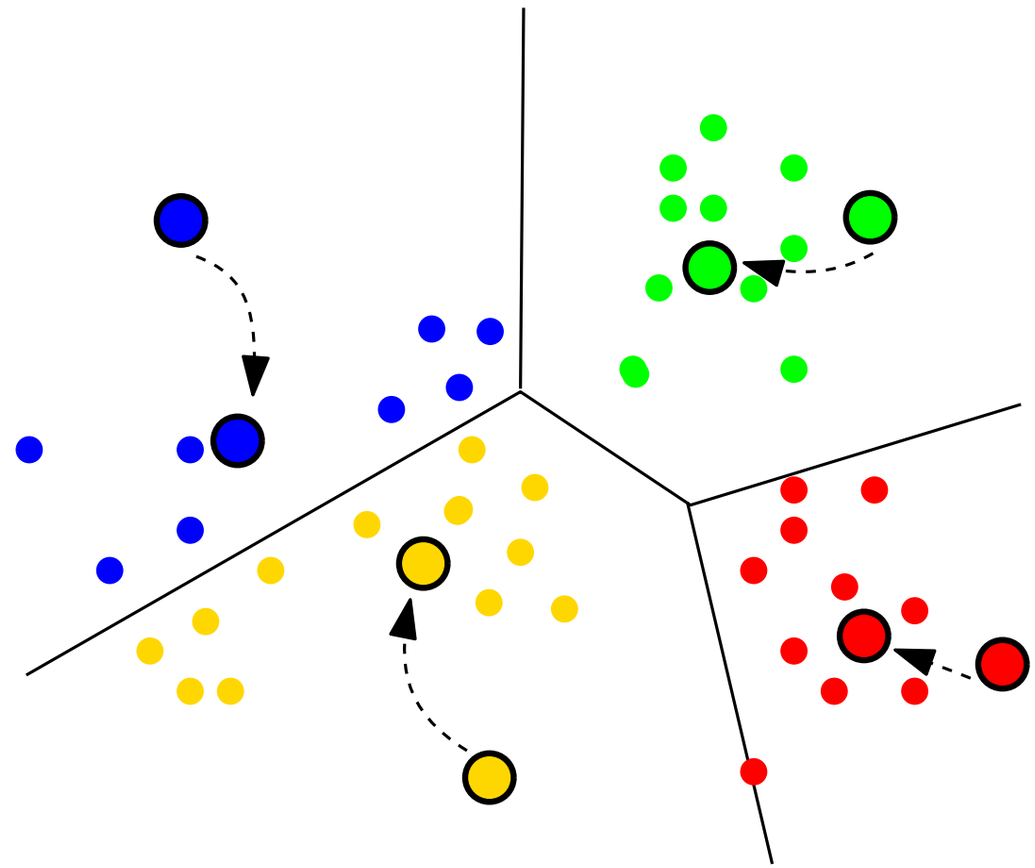
The k-means algorithm



Lloyd's algorithm

- Select $L^1 = \{y_1^1, \ldots, y_k^1\}$ initial 'seeds';
- $i = 1$;
- Repeat
  - For $(j = 1; j \leq k; j++)$ $S_j^i = \{x \in X : d(x, y_j^i) = d(x, L^i)\}$;
  - For $(j = 1; j \leq k; j++)$ $y_j^{i+1} = \frac{1}{|S_j^i|} \sum_{x \in S_j^i} x$;
  - $i++$;
- Until convergence

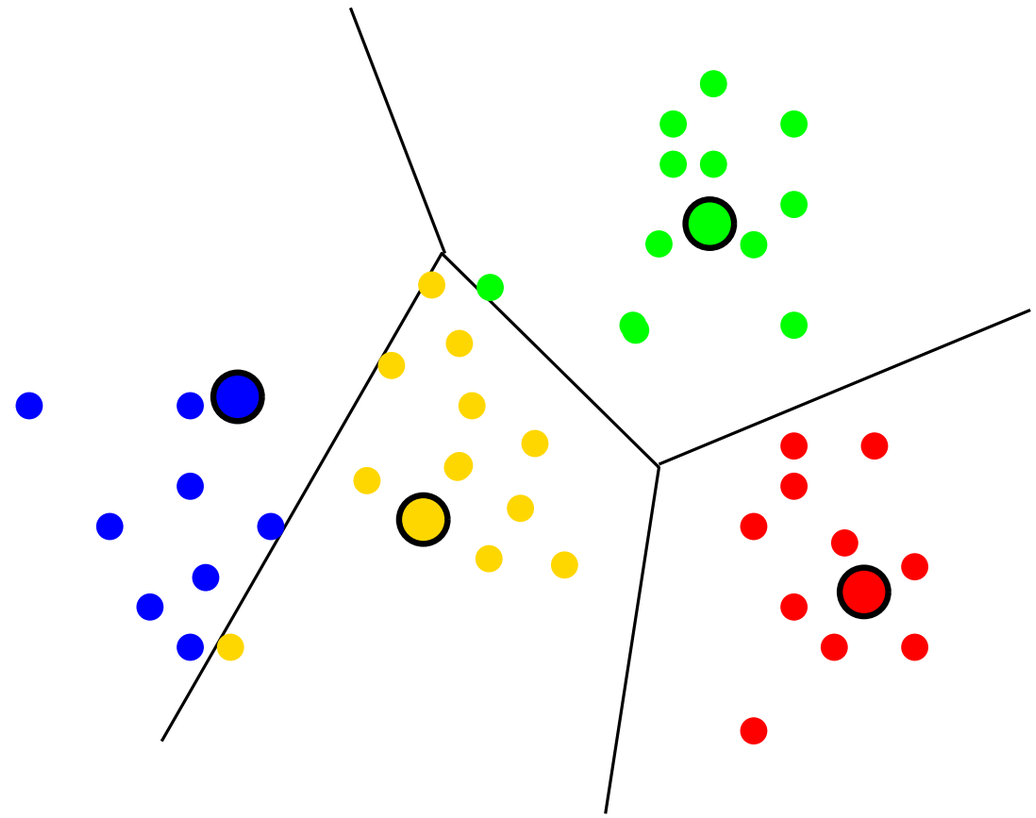
The k-means algorithm



Lloyd's algorithm

- Select $L^1 = \{y_1^1, \ldots, y_k^1\}$ initial 'seeds';
- $i = 1$;
- Repeat
  - For $(j = 1; j \leq k; j++)$ $S_j^i = \{x \in X : d(x, y_j^i) = d(x, L^i)\}$;
  - For $(j = 1; j \leq k; j++)$ $y_j^{i+1} = \frac{1}{|S_j^i|} \sum_{x \in S_j^i} x$;
  - $i++$;
- Until convergence

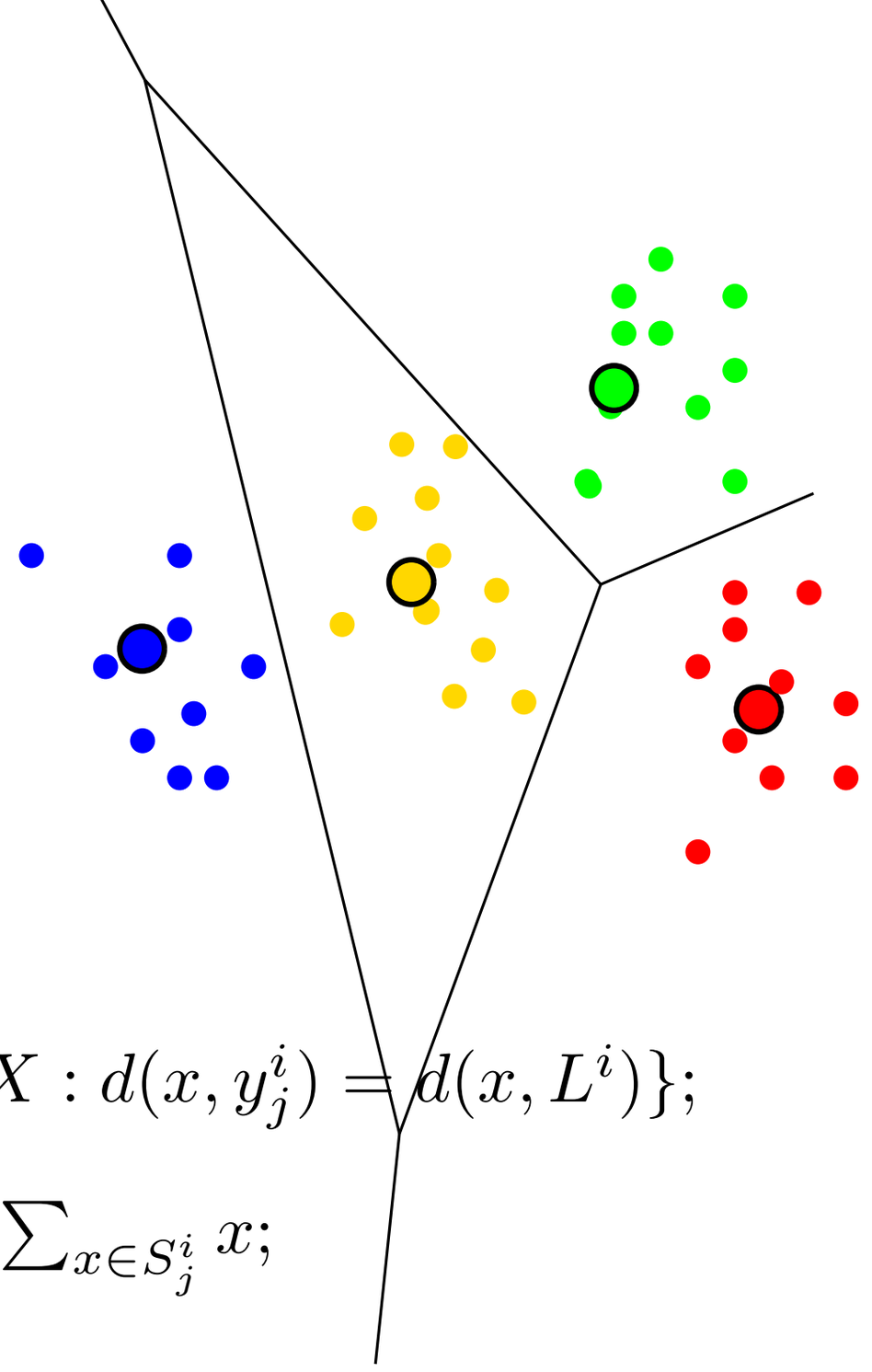
The k-means algorithm



Lloyd's algorithm

- Select $L^1 = \{y_1^1, \ldots, y_k^1\}$ initial 'seeds';
- $i = 1$;
- Repeat
  - For $(j = 1; j \leq k; j++)$ $S_j^i = \{x \in X : d(x, y_j^i) = d(x, L^i)\}$;
  - For $(j = 1; j \leq k; j++)$ $y_j^{i+1} = \frac{1}{|S_j^i|} \sum_{x \in S_j^i} x$;
  - $i++$;
- Until convergence

The k-means algorithm



Lloyd's algorithm

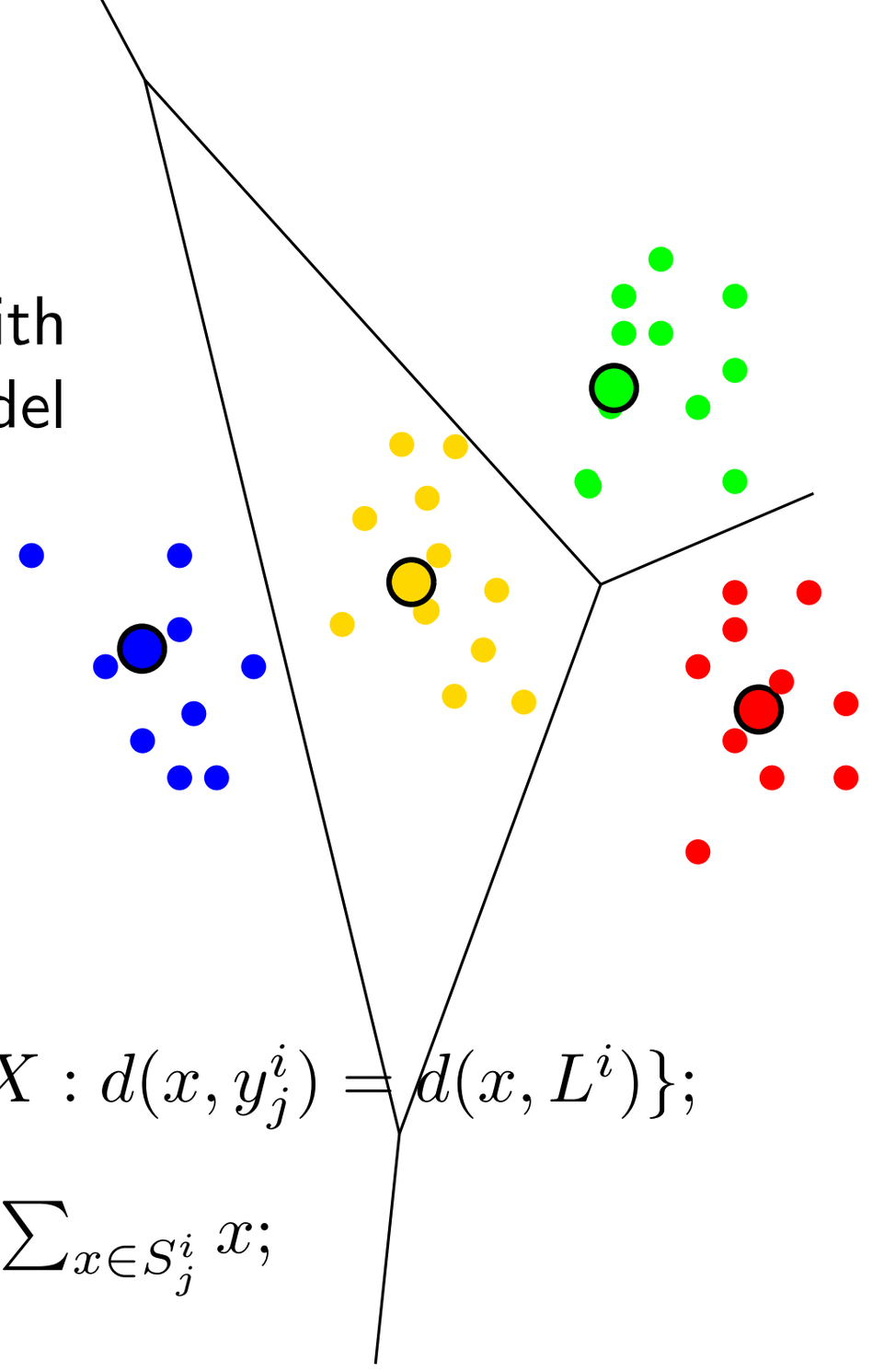
- Select $L^1 = \{y_1^1, \ldots, y_k^1\}$ initial 'seeds';
- $i = 1$;
- Repeat
  - For $(j = 1; j \leq k; j++)$ $S_j^i = \{x \in X : d(x, y_j^i) = d(x, L^i)\}$;
  - For $(j = 1; j \leq k; j++)$ $y_j^{i+1} = \frac{1}{|S_j^i|} \sum_{x \in S_j^i} x$;
  - $i++$;
- Until convergence

The k-means algorithm

Particular instance of the EM algorithm with (hard instead of soft) Gaussian mixture model with fixed bandwidths.

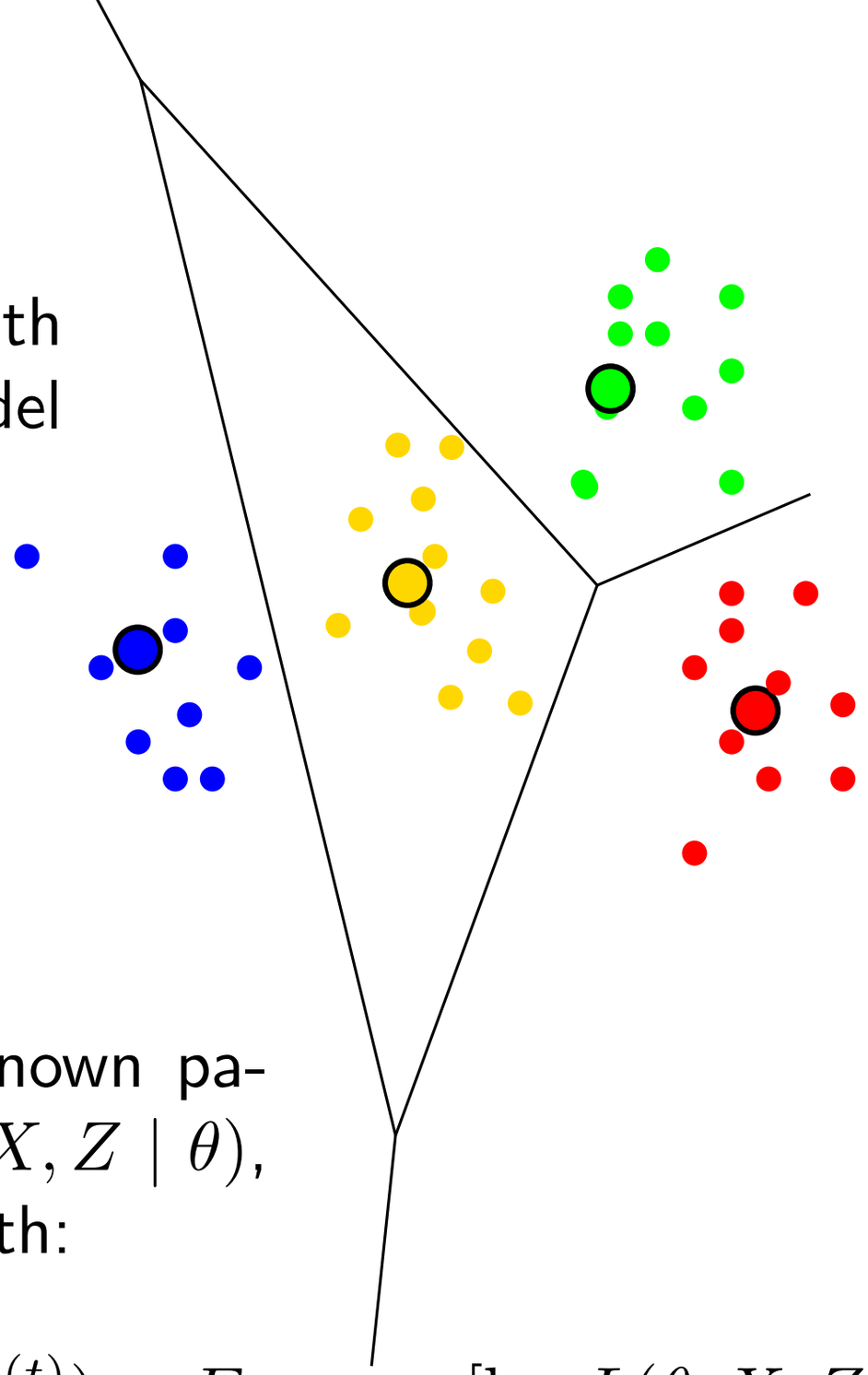
Lloyd's algorithm

- Select $L^1 = \{y_1^1, \ldots, y_k^1\}$ initial 'seeds';
- $i = 1$;
- Repeat
  - For $(j = 1; j \leq k; j++)$ $S_j^i = \{x \in X : d(x, y_j^i) = d(x, L^i)\}$;
  - For $(j = 1; j \leq k; j++)$ $y_j^{i+1} = \frac{1}{|S_j^i|} \sum_{x \in S_j^i} x$;
  - $i++$;
- Until convergence



The k-means algorithm

Particular instance of the EM algorithm with (hard instead of soft) Gaussian mixture model with fixed bandwidths.



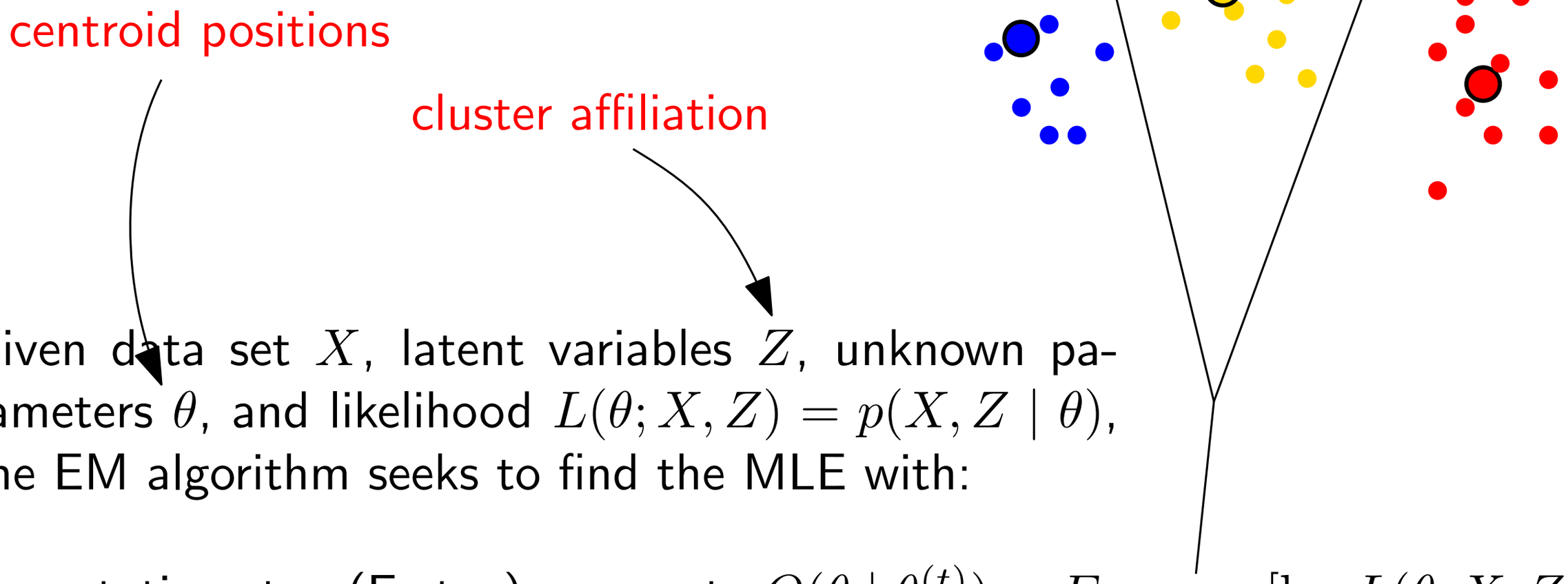
Given data set X, latent variables Z, unknown parameters θ, and likelihood $L(\theta; X, Z) = p(X, Z \mid \theta)$, the EM algorithm seeks to find the MLE with:

Expectation step (E step): compute $Q(\theta \mid \theta^{(t)}) = E_{Z|X,\theta^{(t)}} [\log L(\theta; X, Z)]$

Maximization step (M step): Find $\theta^{(t+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(t)})$

The k-means algorithm

Particular instance of the EM algorithm with (hard instead of soft) Gaussian mixture model with fixed bandwidths.



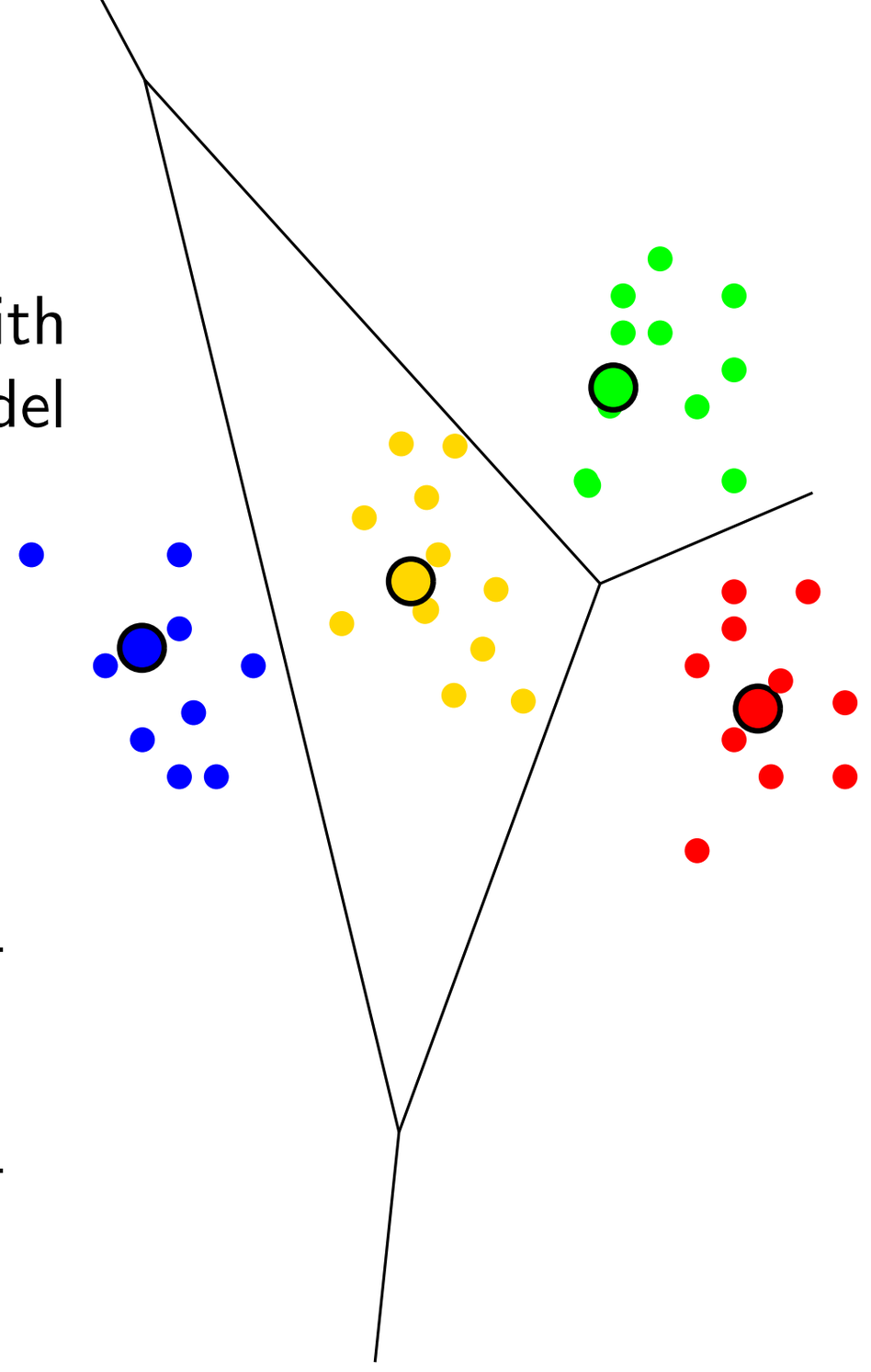
Given data set X, latent variables Z, unknown parameters θ, and likelihood $L(\theta; X, Z) = p(X, Z \mid \theta)$, the EM algorithm seeks to find the MLE with:

Expectation step (E step): compute $Q(\theta \mid \theta^{(t)}) = E_{Z|X,\theta^{(t)}} [\log L(\theta; X, Z)]$

Maximization step (M step): Find $\theta^{(t+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(t)})$

The k-means algorithm

Particular instance of the EM algorithm with (hard instead of soft) Gaussian mixture model with fixed bandwidths.



Warning:

- Lloyd's algorithm does not ensure convergence to a global minimum!
- The speed of convergence is not guaranteed.
- The output is very sensitive to the choice of initial seeds (but there exists some strategies $\rightarrow$ k-means++).

Hierarchical clustering algorithms

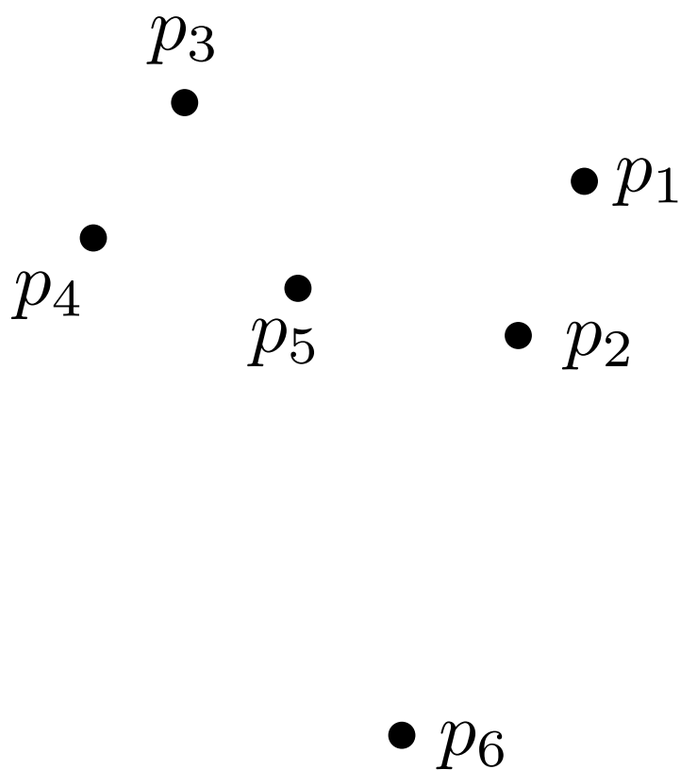
Build a hierarchy of clusters (nested family of clustering partitions)

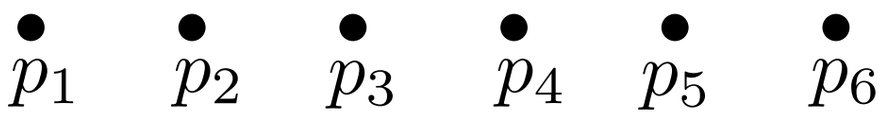
Hierarchical clustering algorithms

Build a hierarchy of clusters (nested family of clustering partitions)

Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).



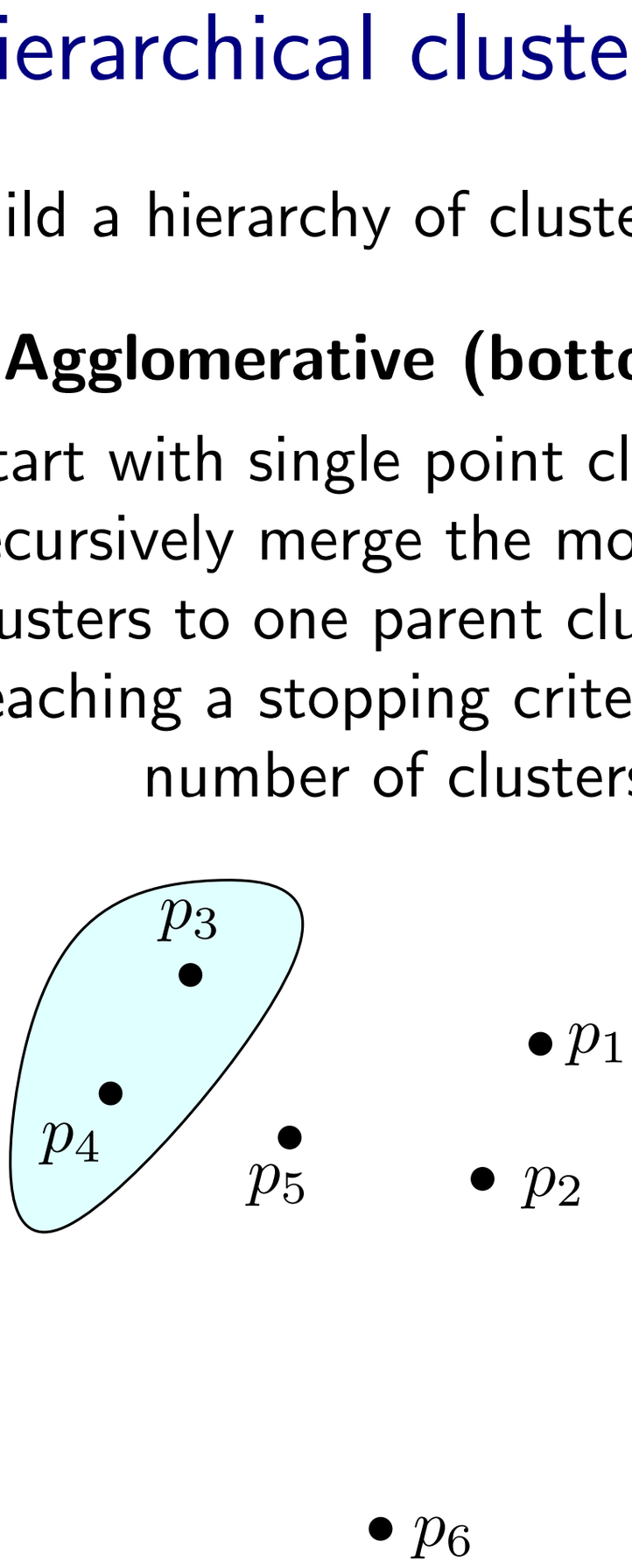


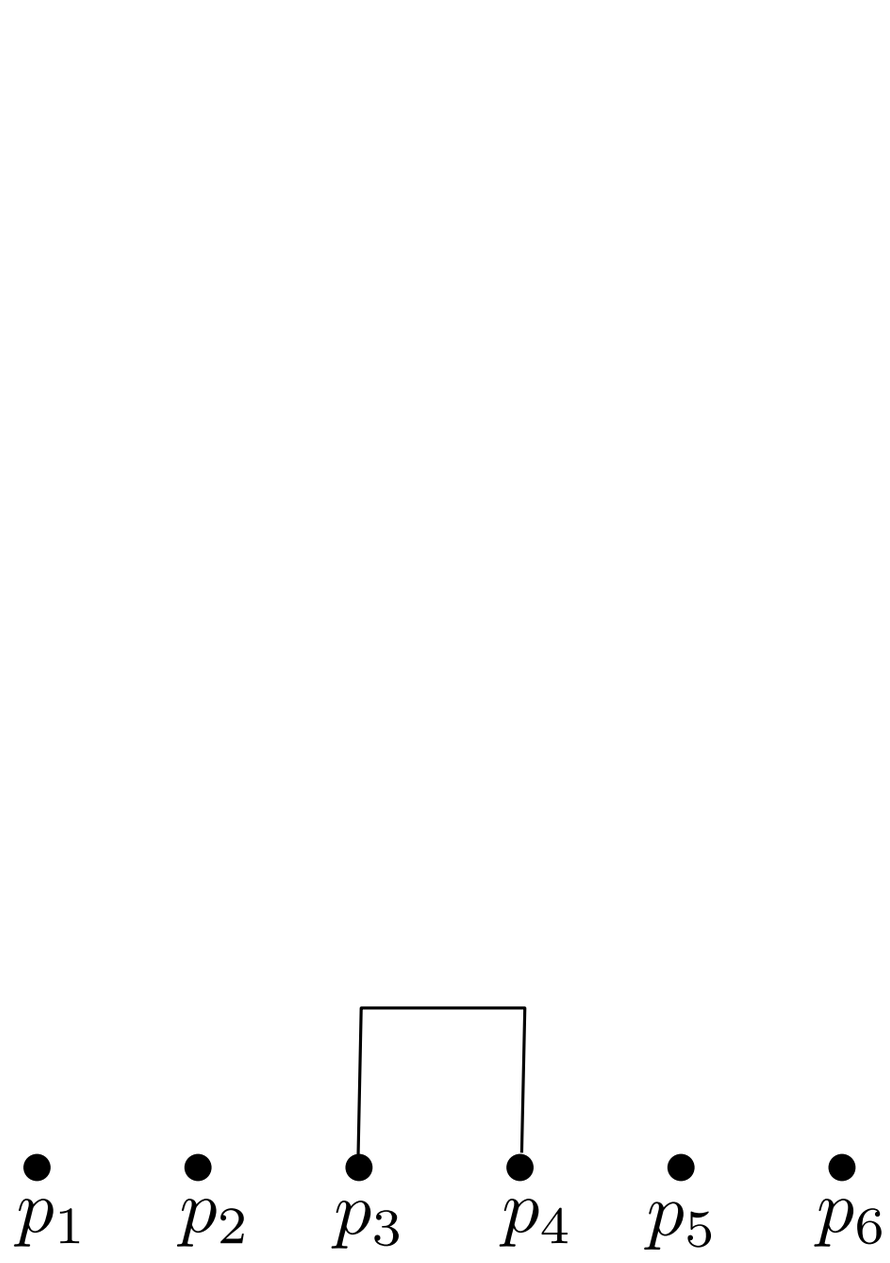
Hierarchical clustering algorithms

Build a hierarchy of clusters (nested family of clustering partitions)

Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).



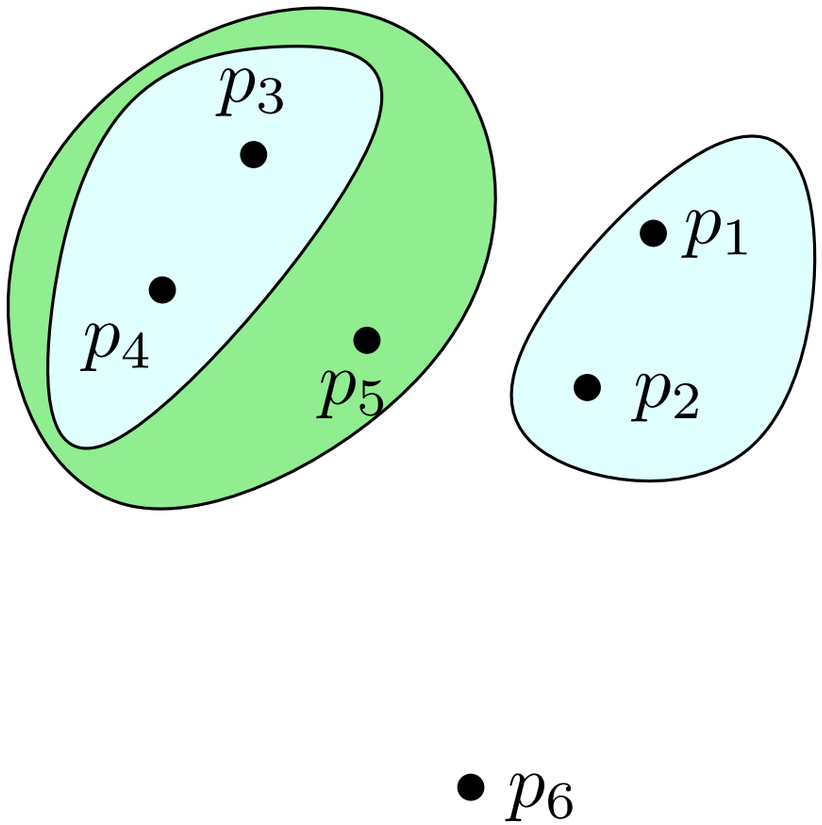


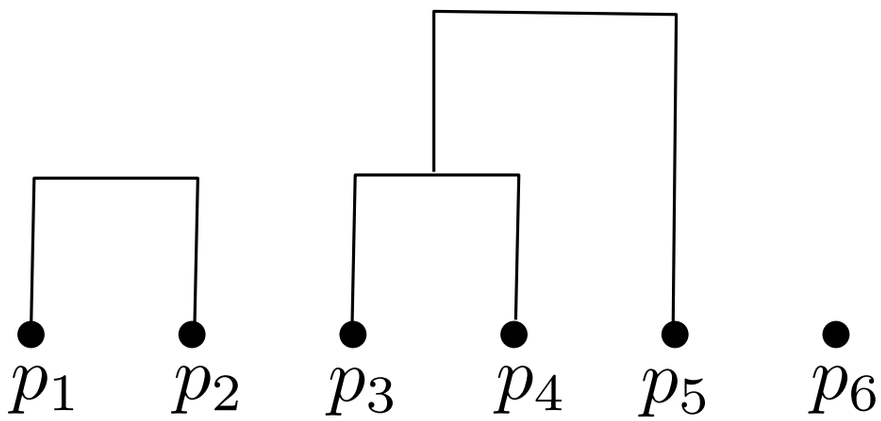
Hierarchical clustering algorithms

Build a hierarchy of clusters (nested family of clustering partitions)

Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).



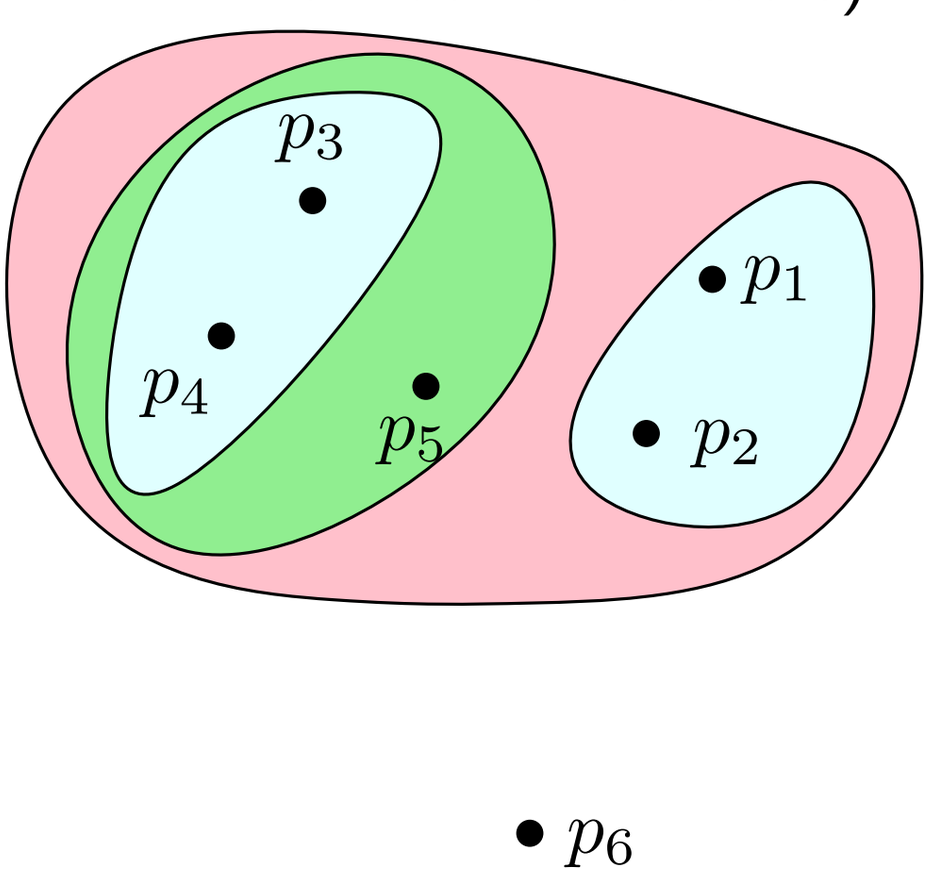


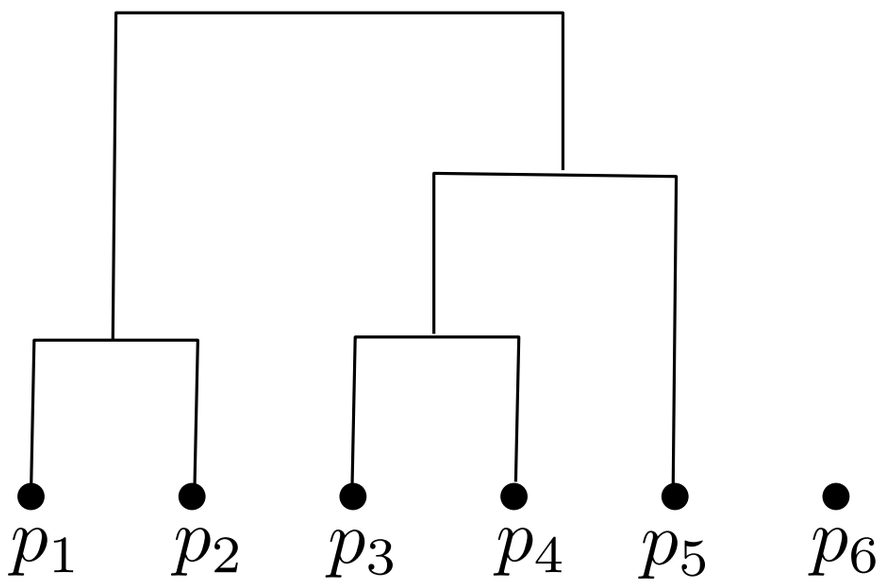
Hierarchical clustering algorithms

Build a hierarchy of clusters (nested family of clustering partitions)

Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).



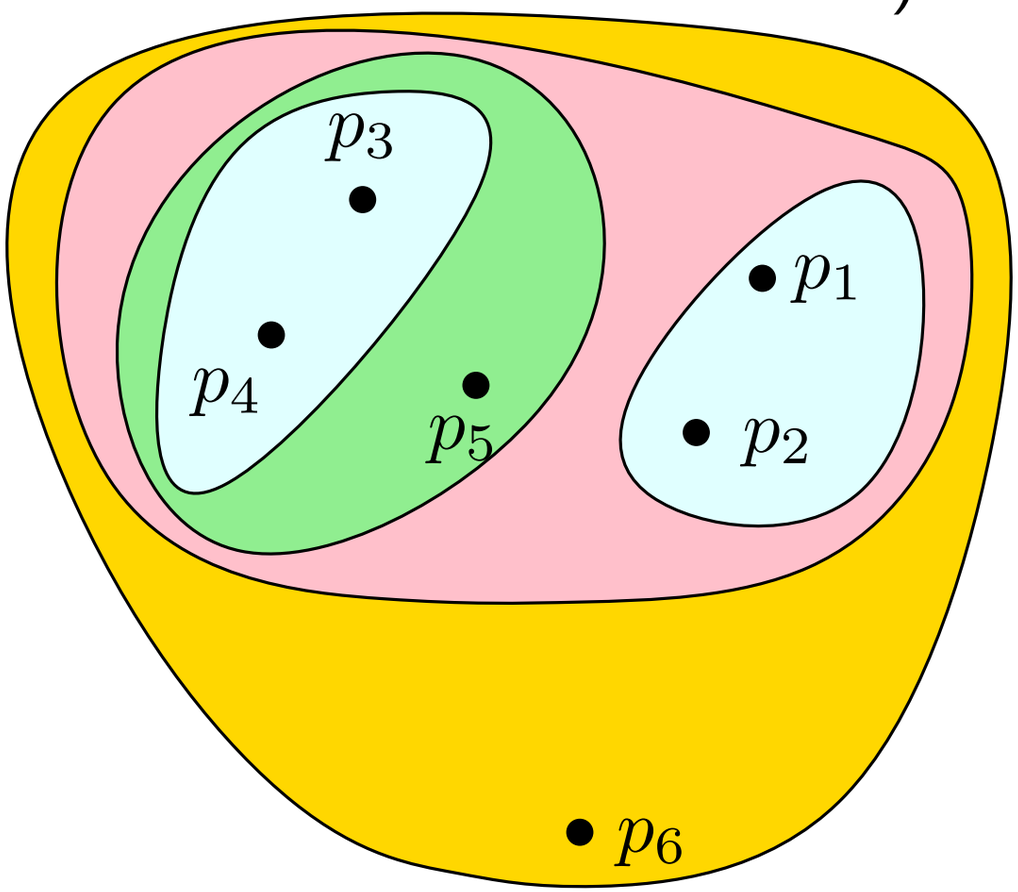


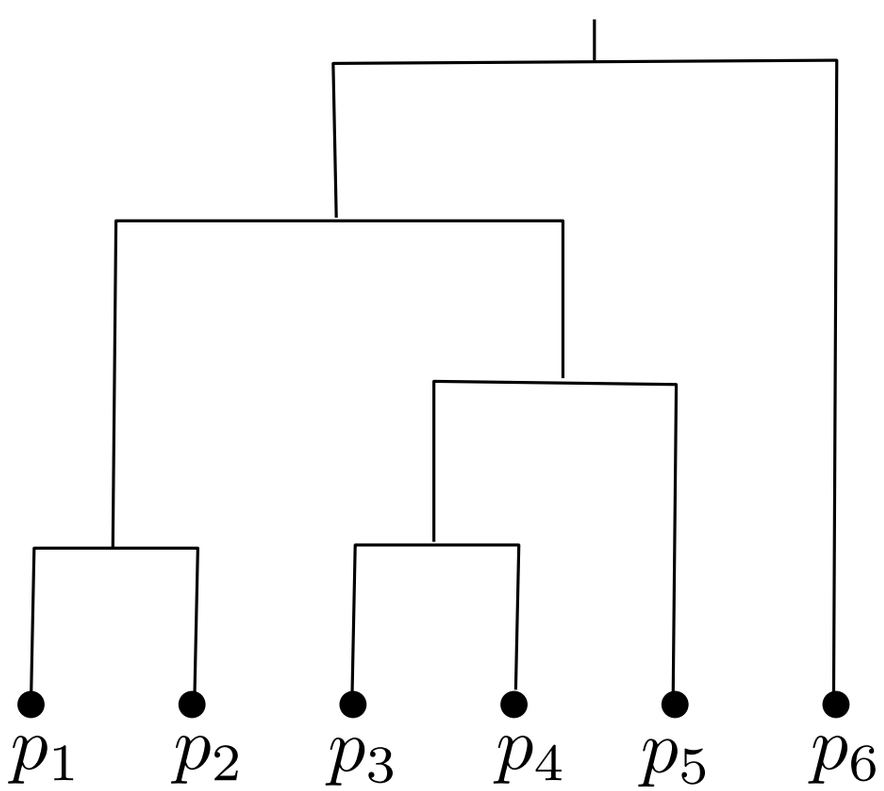
Hierarchical clustering algorithms

Build a hierarchy of clusters (nested family of clustering partitions)

Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).



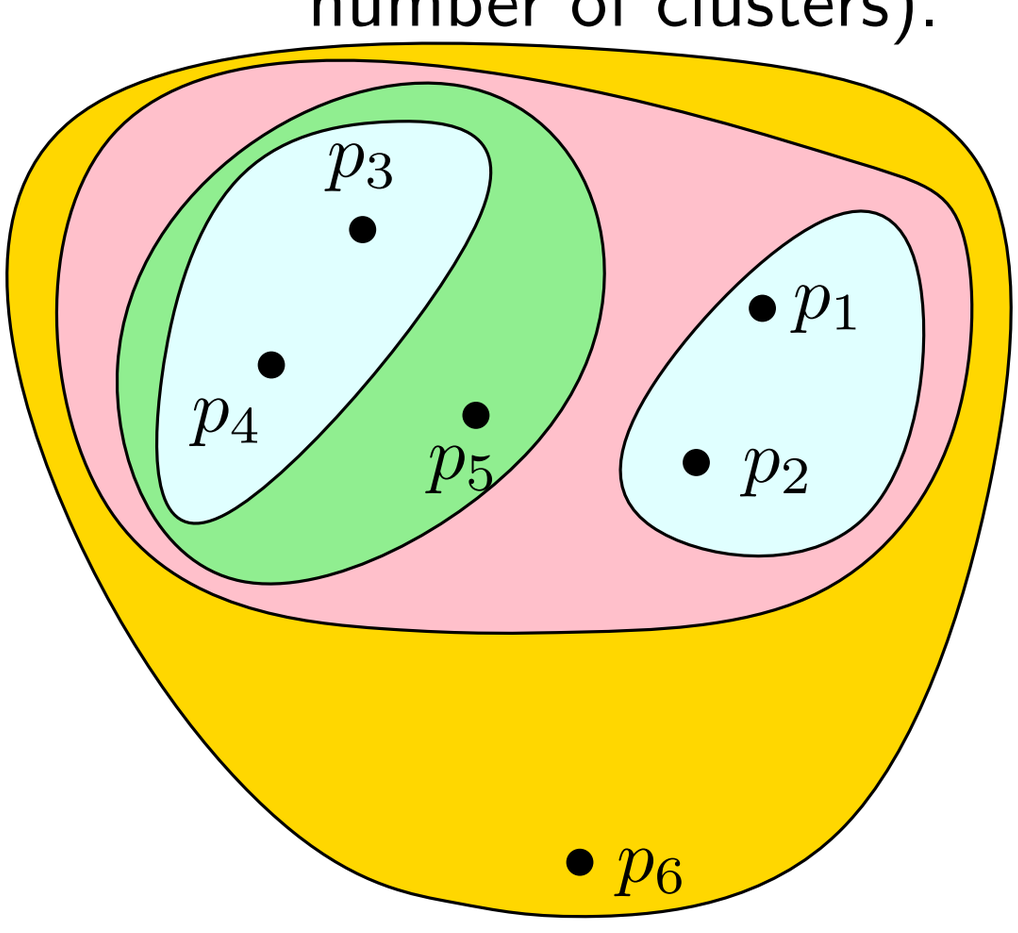


Hierarchical clustering algorithms

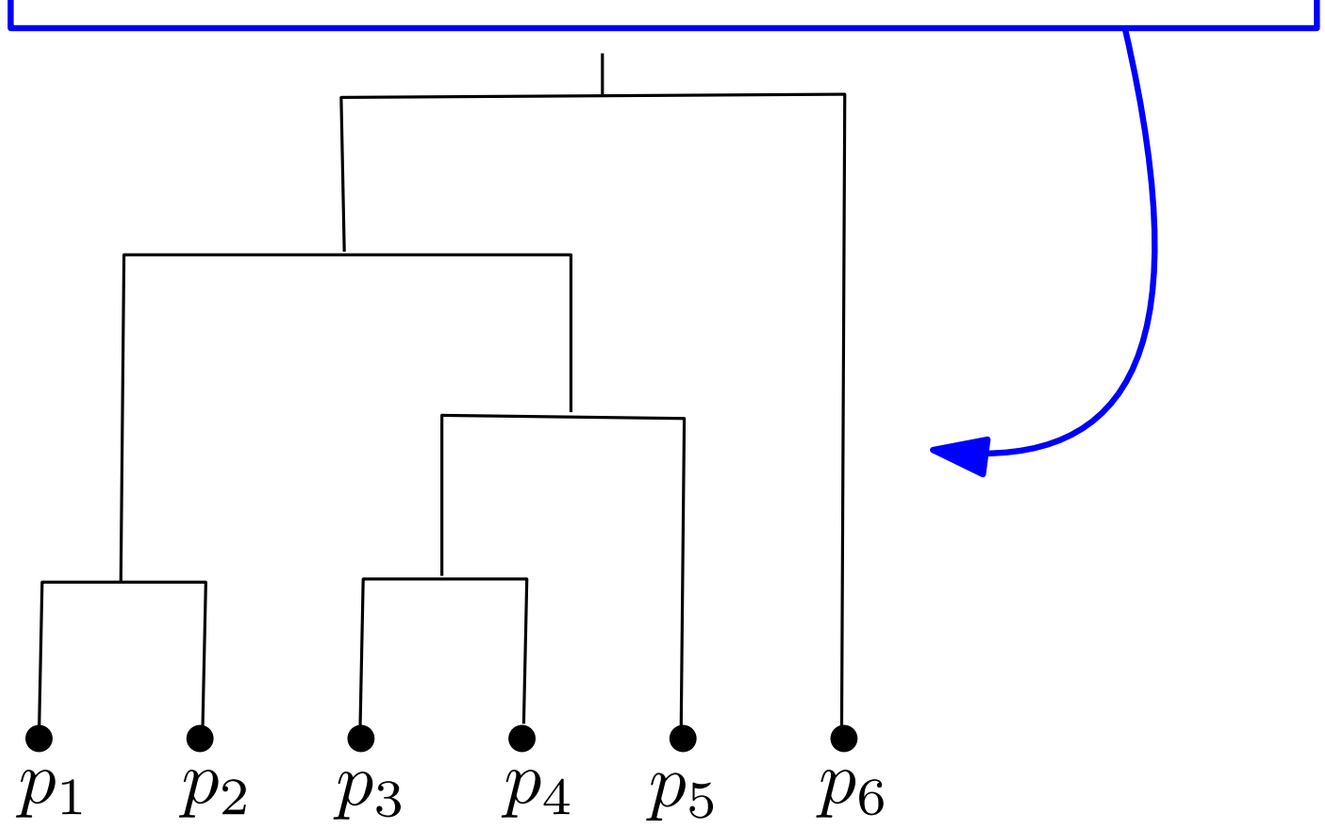
Build a hierarchy of clusters (nested family of clustering partitions)

Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).



Dendogram, i;e. a tree such that:
- each leaf node is a singleton,
- each node represent a cluster,
- the root node contains the whole data,
- each internal node has two daughters, corresponding to the clusters that were merged to obtain it.



Hierarchical clustering algorithms

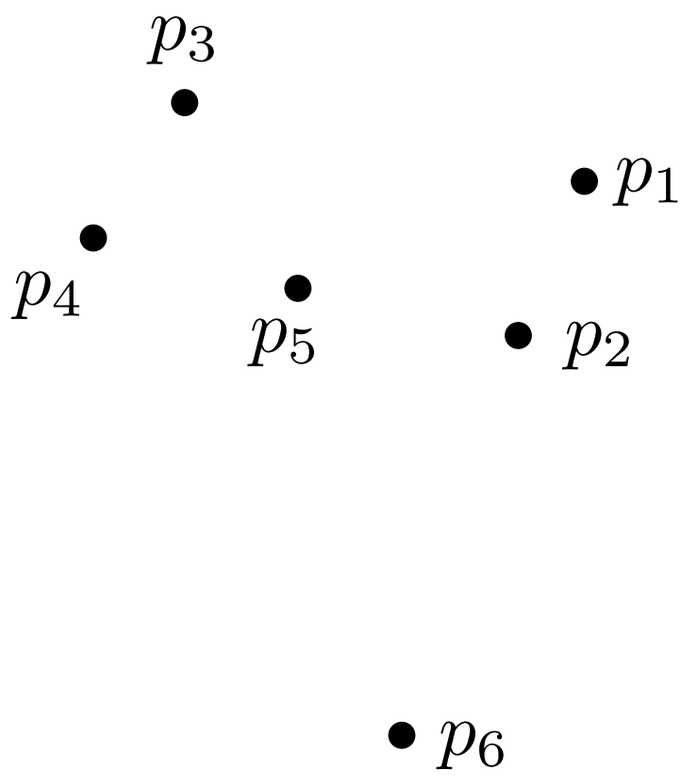
Build a hierarchy of clusters (nested family of clustering partitions)

Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).

Dividing (top-down)

Start with a single global cluster and recursively split each cluster until reaching a stopping criterion.



Hierarchical clustering algorithms

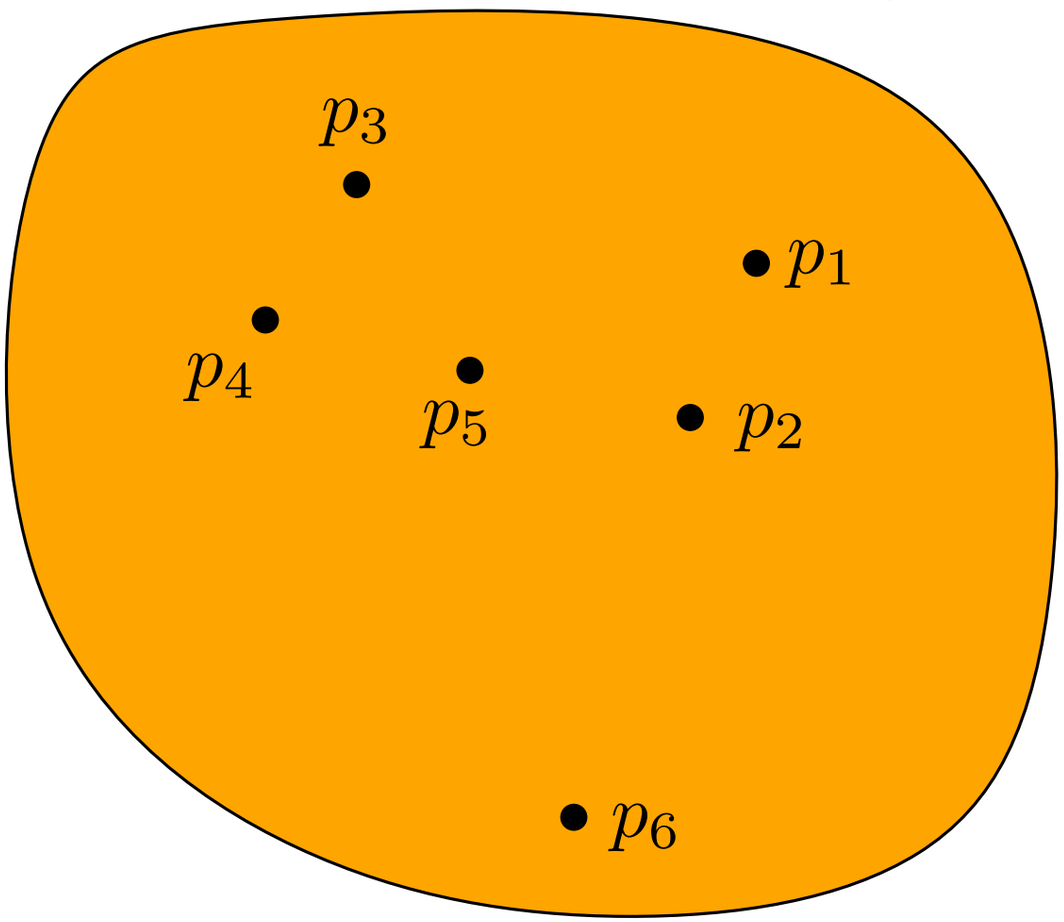
Build a hierarchy of clusters (nested family of clustering partitions)

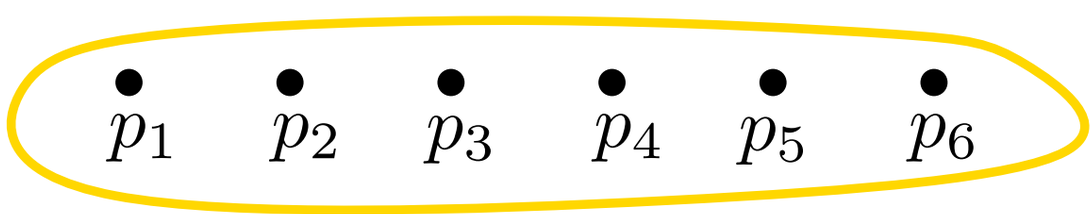
Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).

Dividing (top-down)

Start with a single global cluster and recursively split each cluster until reaching a stopping criterion.





Hierarchical clustering algorithms

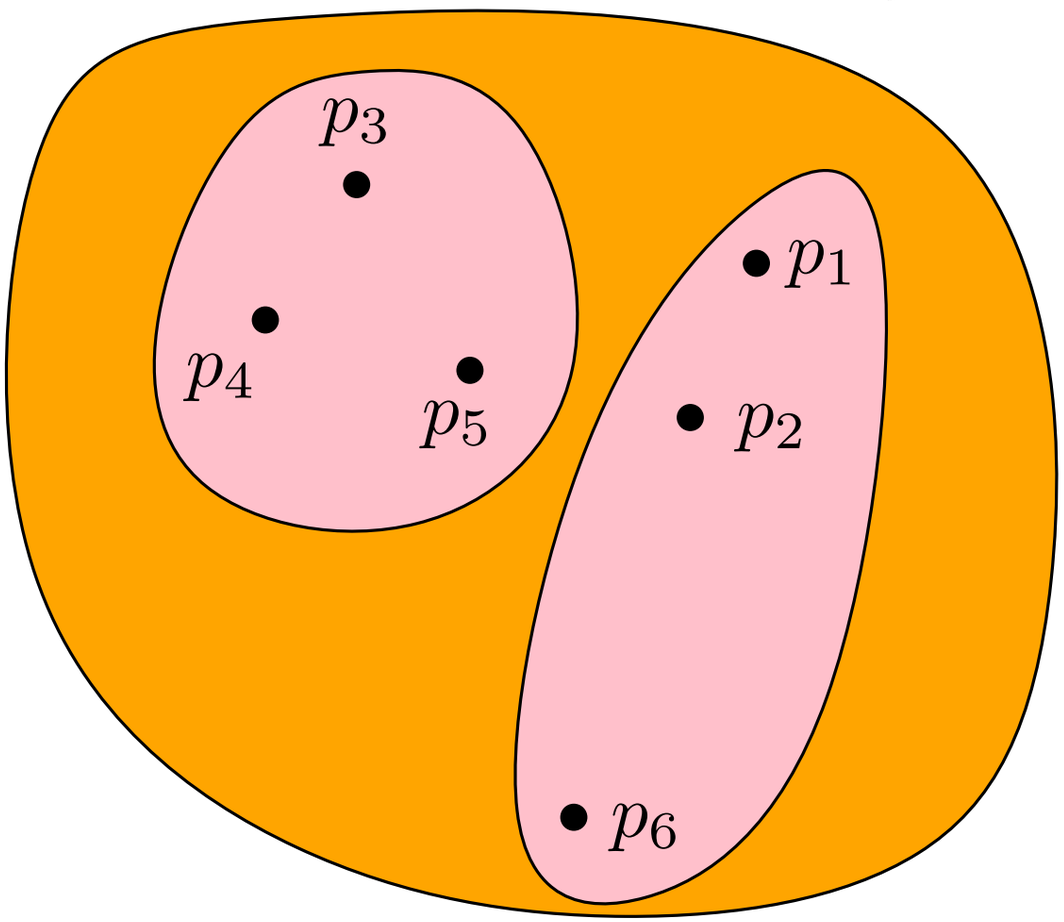
Build a hierarchy of clusters (nested family of clustering partitions)

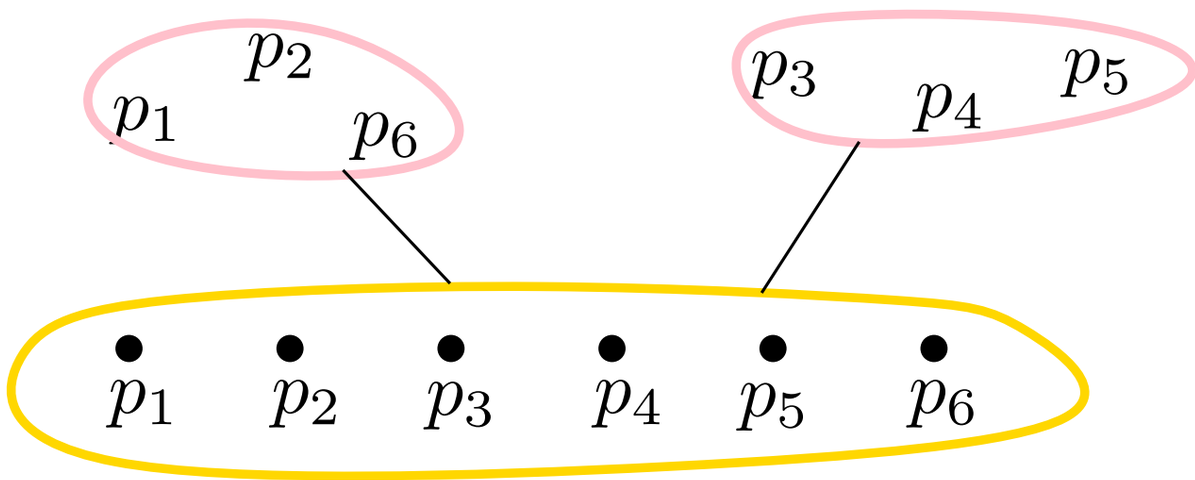
Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).

Dividing (top-down)

Start with a single global cluster and recursively split each cluster until reaching a stopping criterion.





Hierarchical clustering algorithms

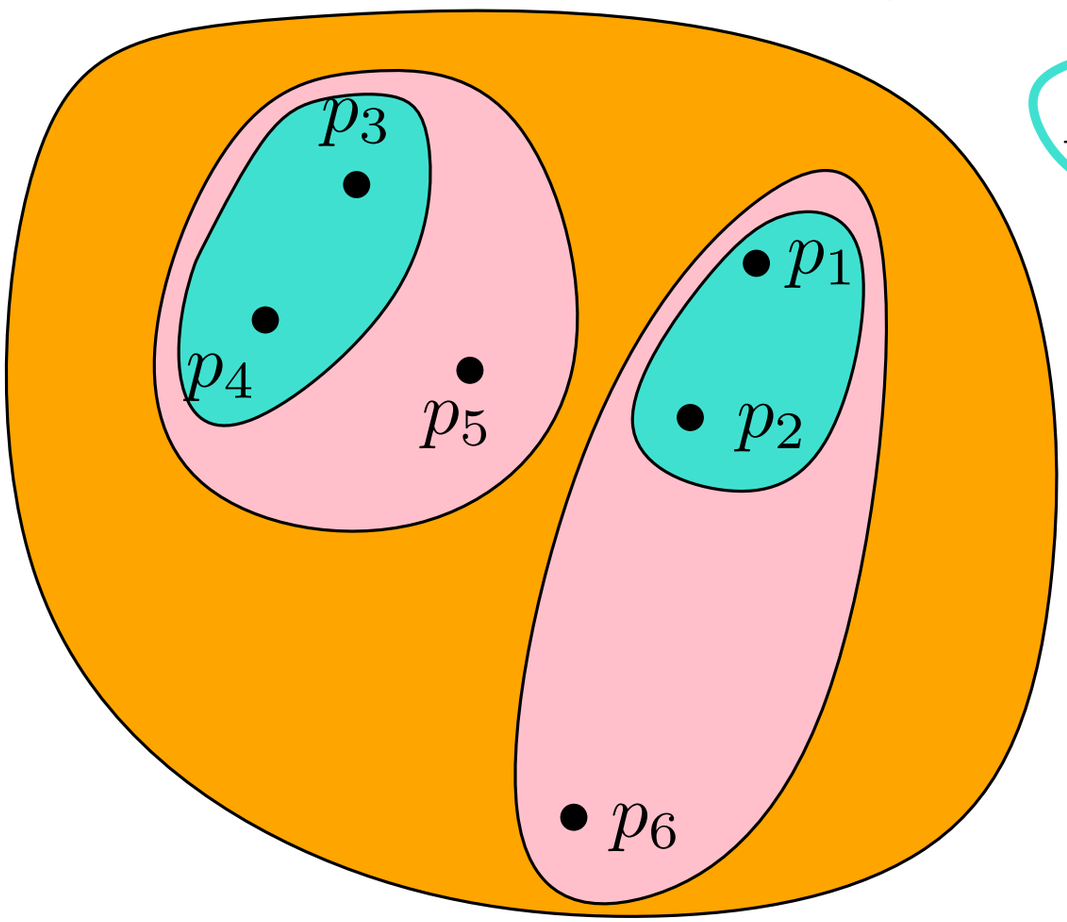
Build a hierarchy of clusters (nested family of clustering partitions)

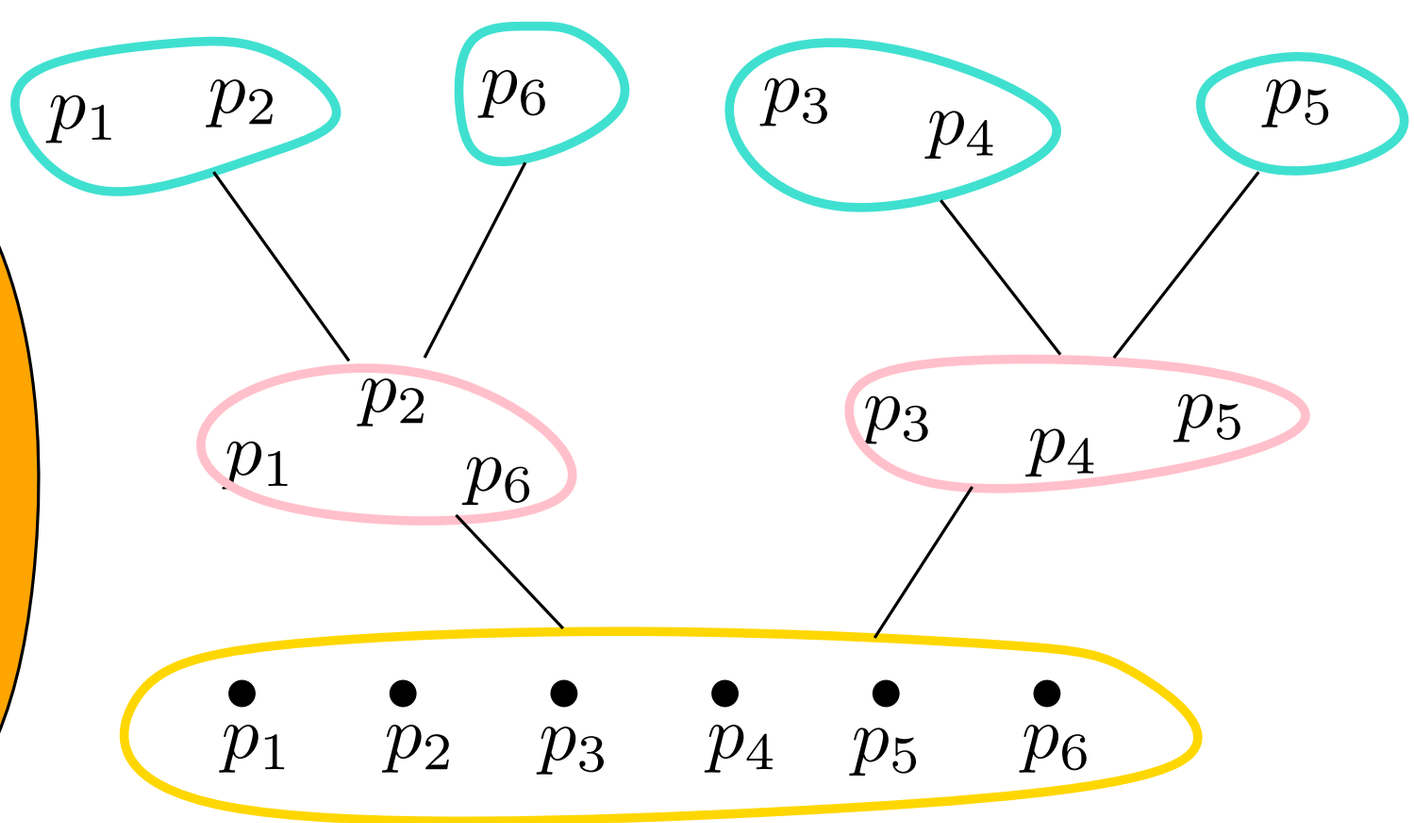
Agglomerative (bottom-up)

Start with single point cluster and recursively merge the most similar clusters to one parent cluster until reaching a stopping criterion (e.g. number of clusters).

Dividing (top-down)

Start with a single global cluster and recursively split each cluster until reaching a stopping criterion.





Single linkage clustering

Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).

Output: the resulting dendogram

Single linkage clustering

Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

sup: complete linkage

$\frac{1}{|C|\cdot|C'|}\sum$: average linkage

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).

Output: the resulting dendogram

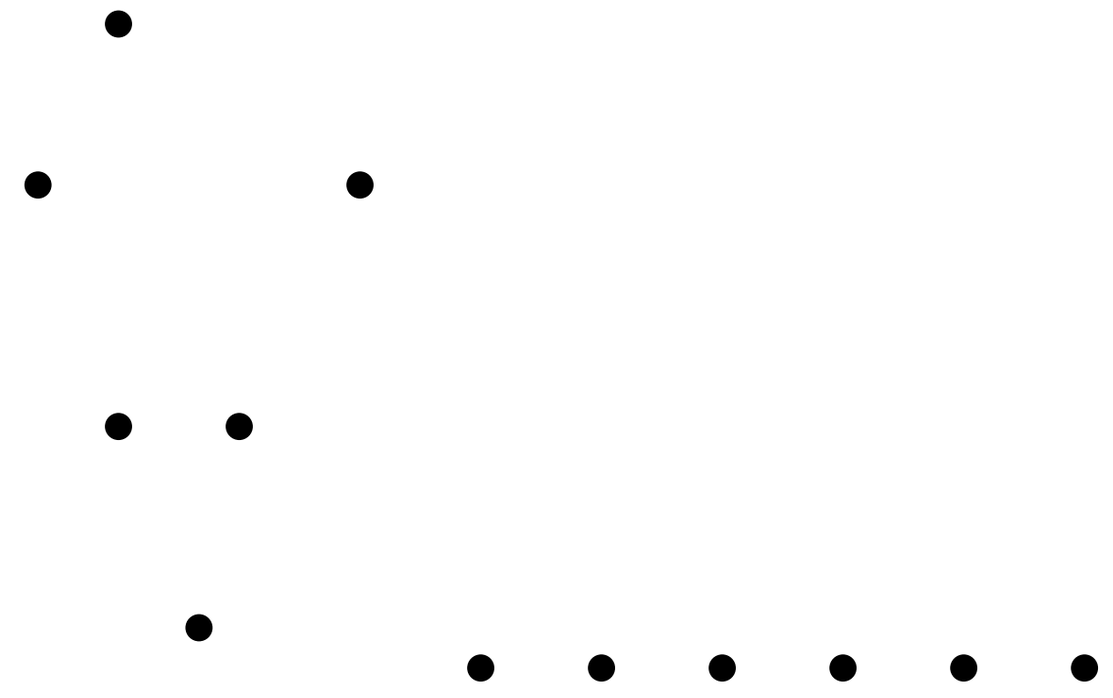
Single linkage clustering

Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).



Output: the resulting dendogram

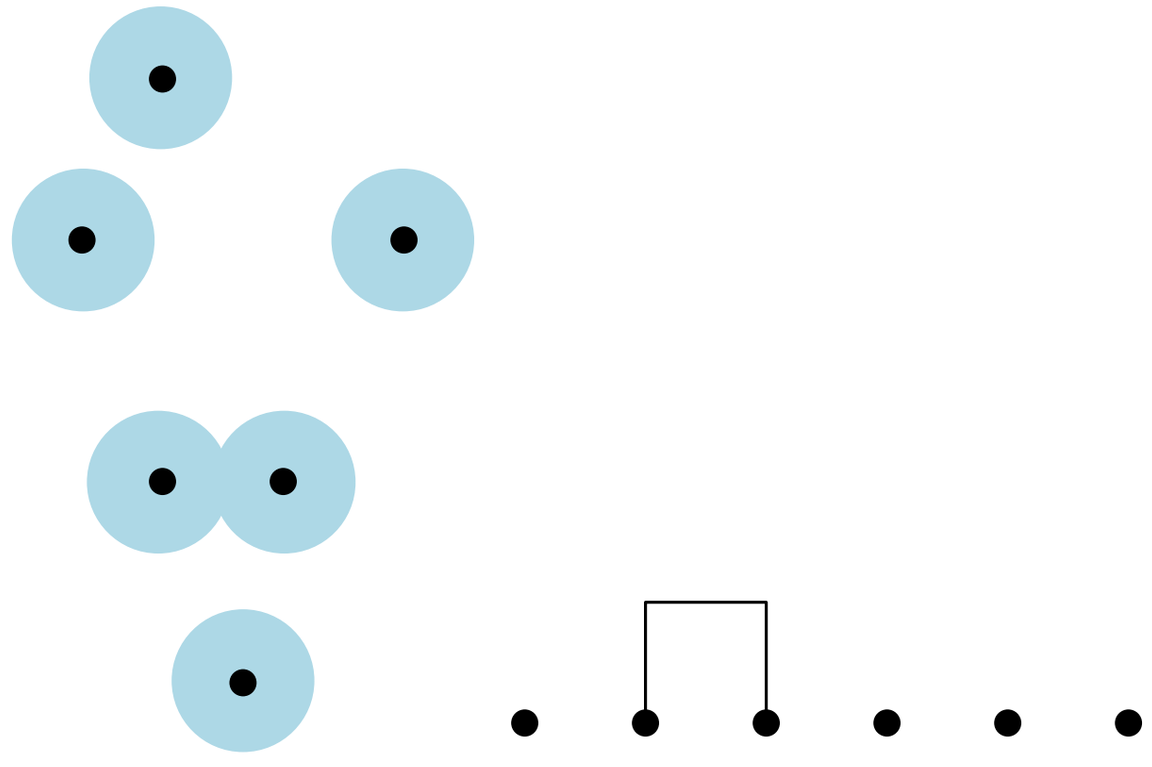
Single linkage clustering

Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).



Output: the resulting dendogram

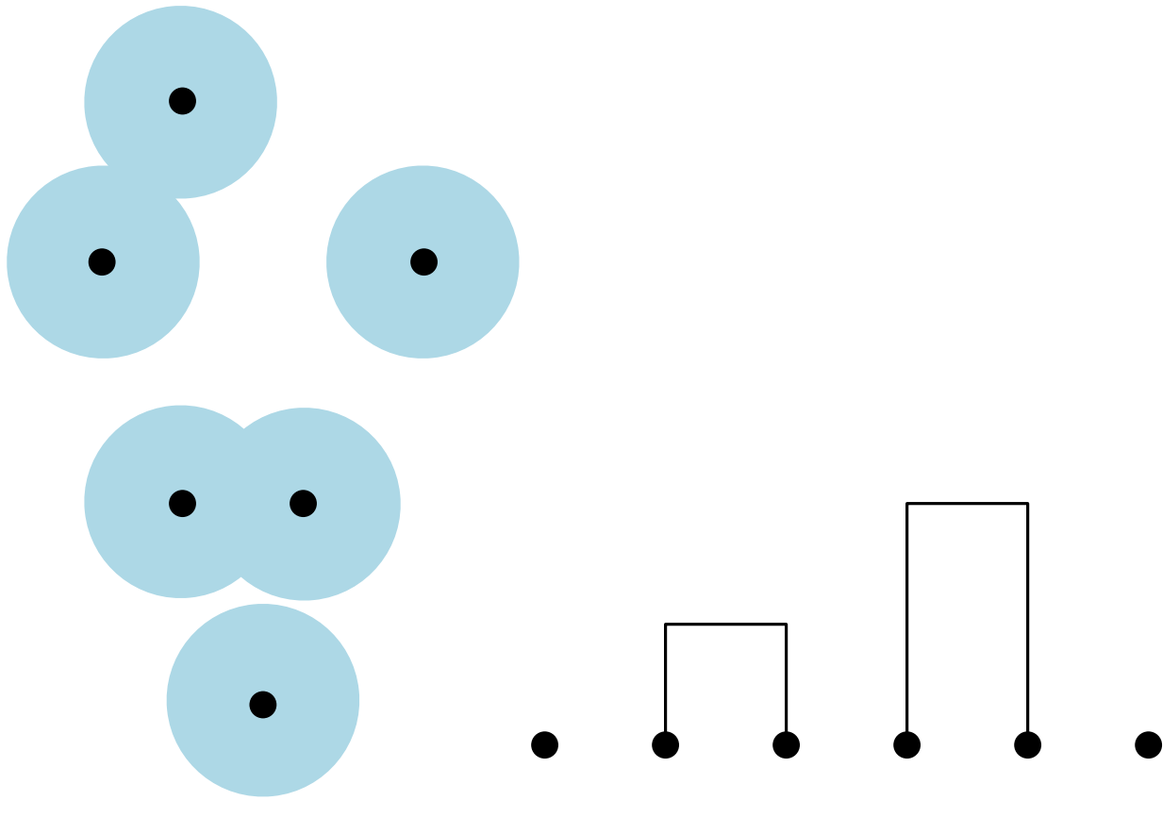
Single linkage clustering

Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).



Output: the resulting dendogram

Single linkage clustering

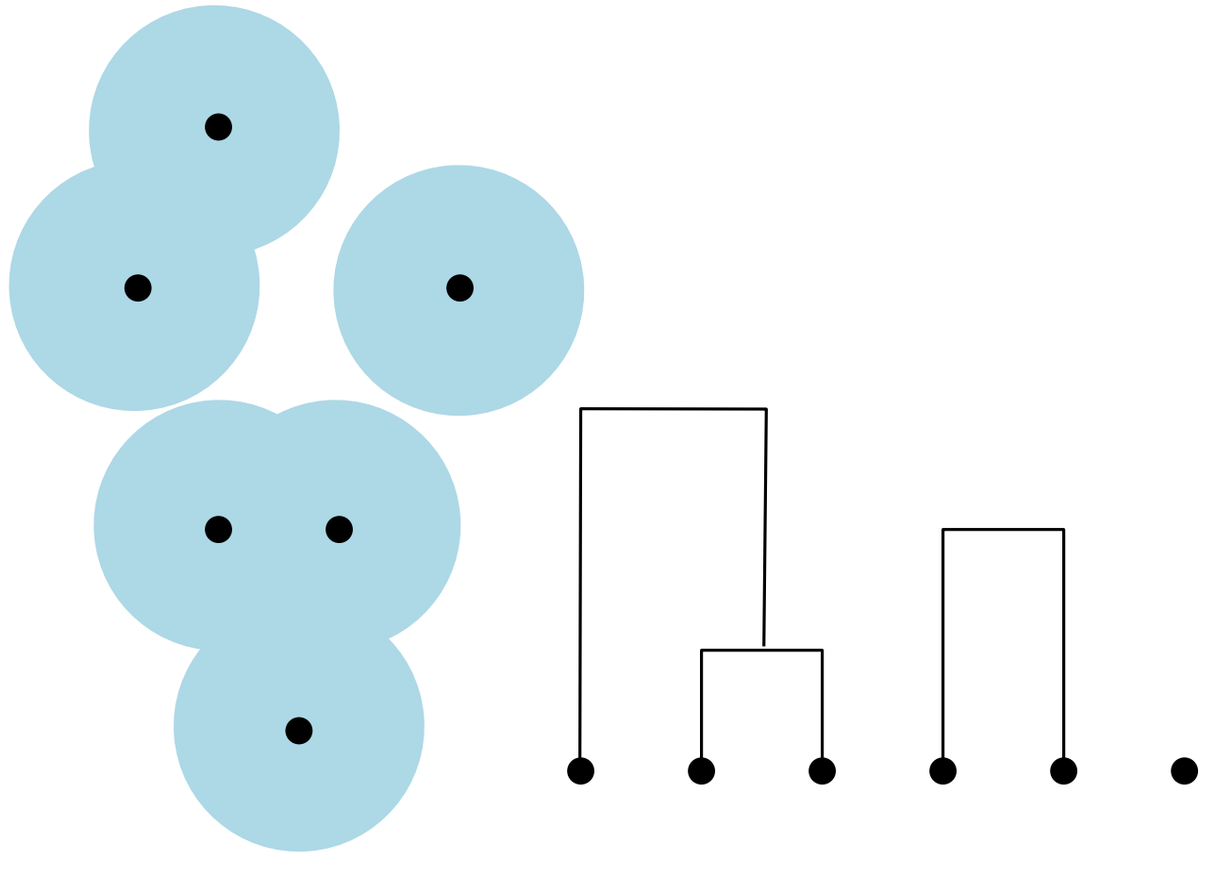
Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).

Output: the resulting dendogram



Single linkage clustering

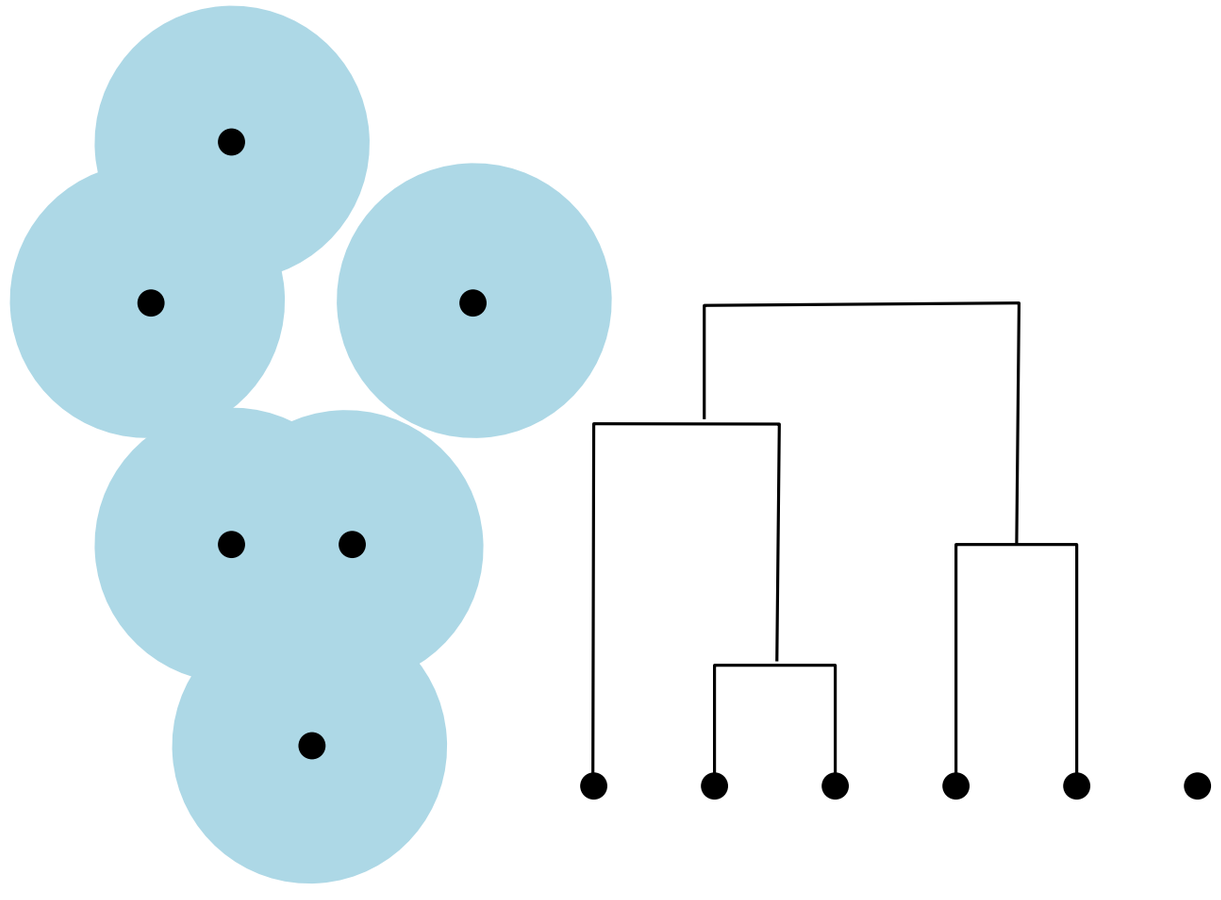
Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).

Output: the resulting dendogram



Single linkage clustering

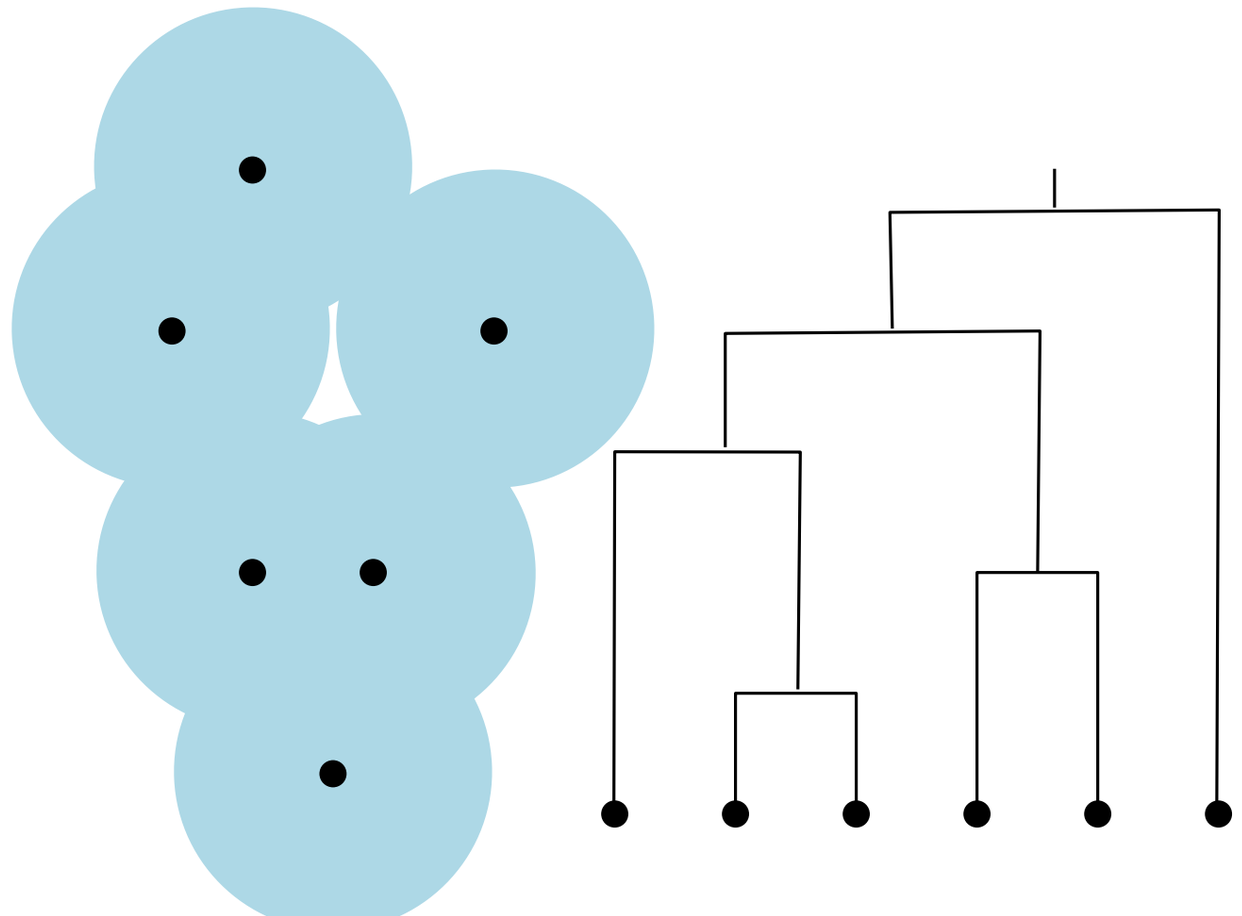
Input: A set $X_n = \{x_1, \ldots, x_n\}$ in a metric space (X, d) (or just a matrix of pairwise dissimilarities $(d_{i,j})$).

Given two clusters $C, C' \subseteq X_n$ let $d(C, C') = \inf_{x \in C, x' \in C'} d(x, x')$

1. Start with a clustering where each x_i is a cluster.

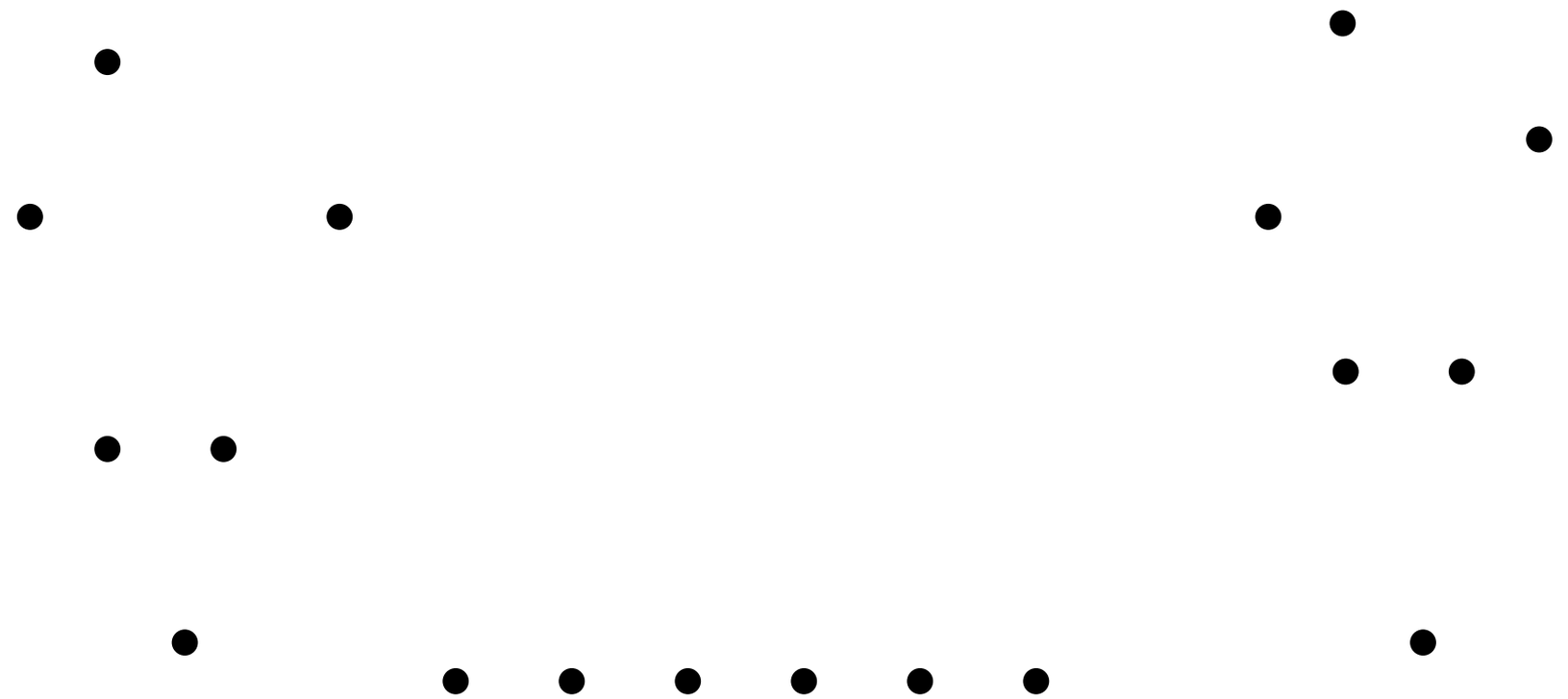
2. At each step, merge the two closest clusters until it remains a single cluster (containing all data points).

Output: the resulting dendogram

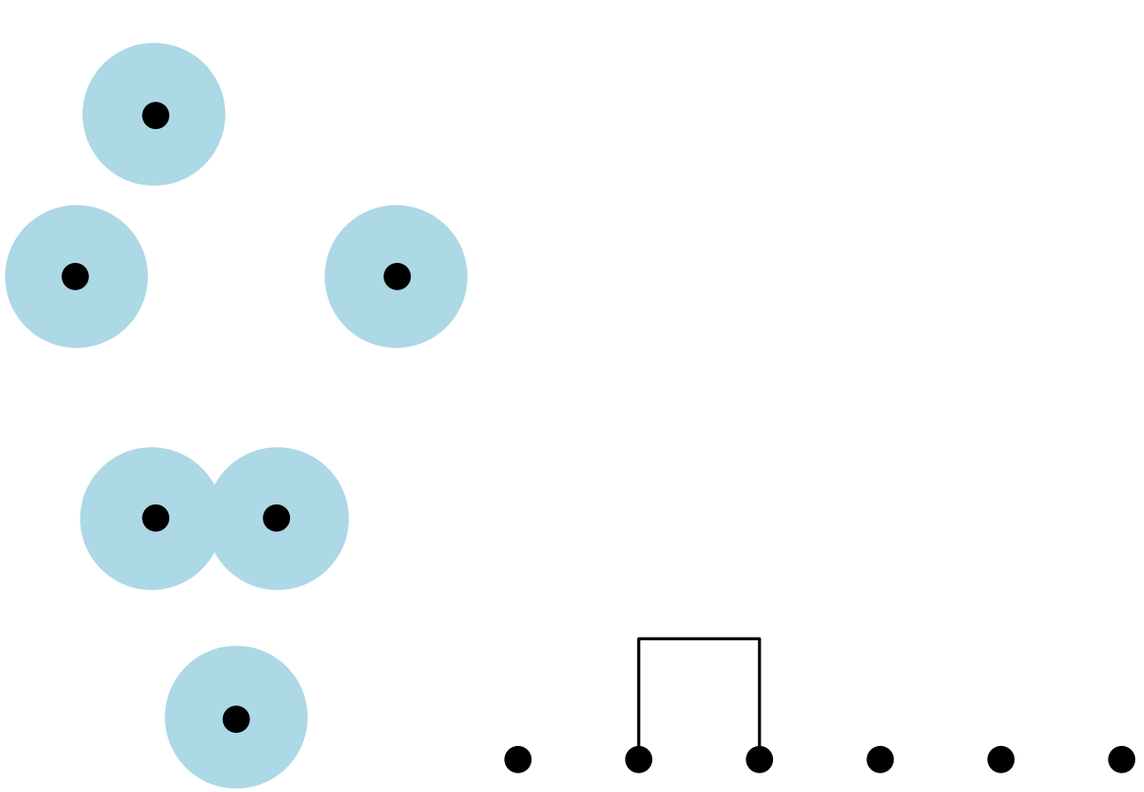


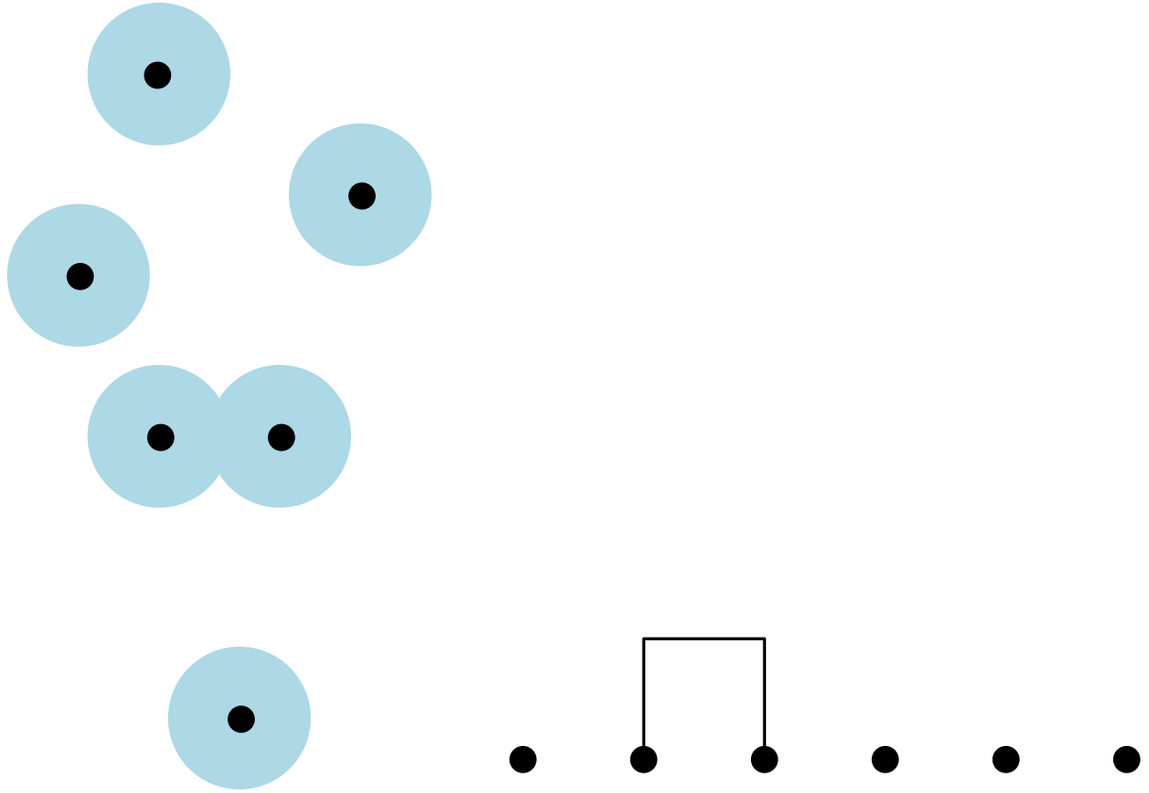
The (in)stability of dendograms

The (in)stability of dendograms

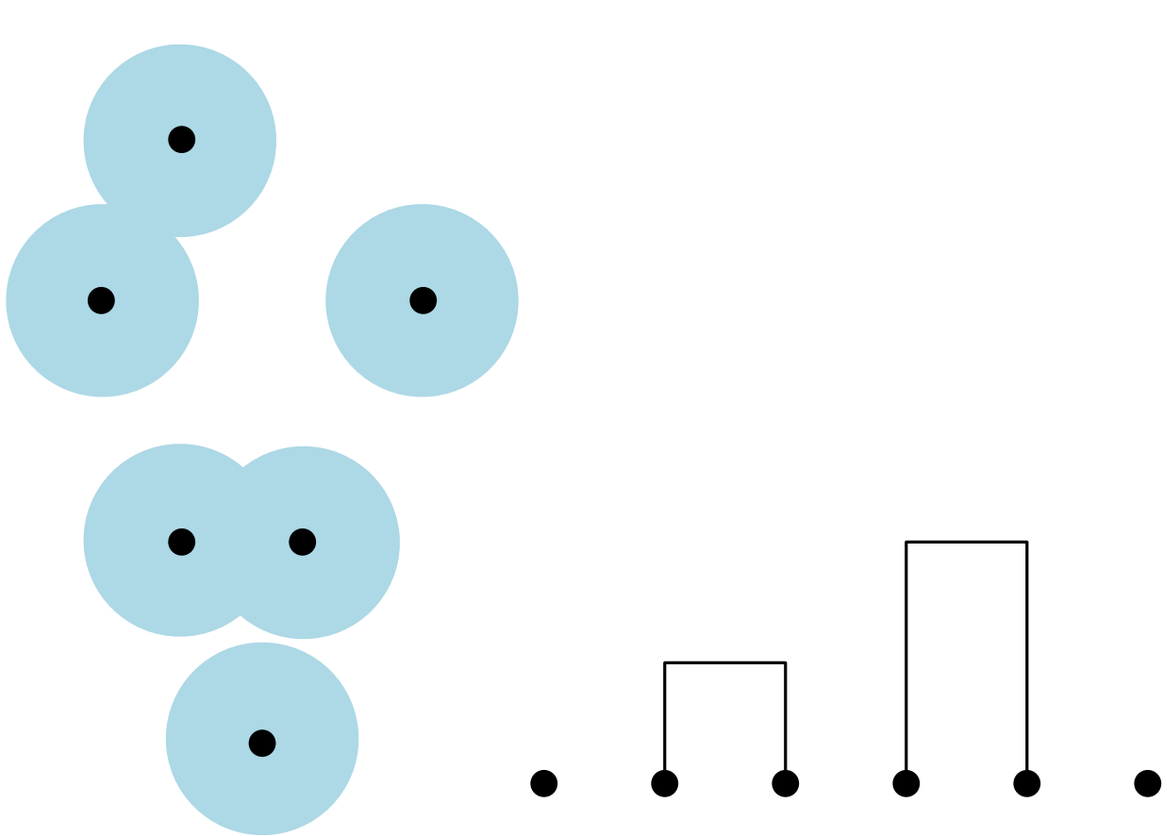


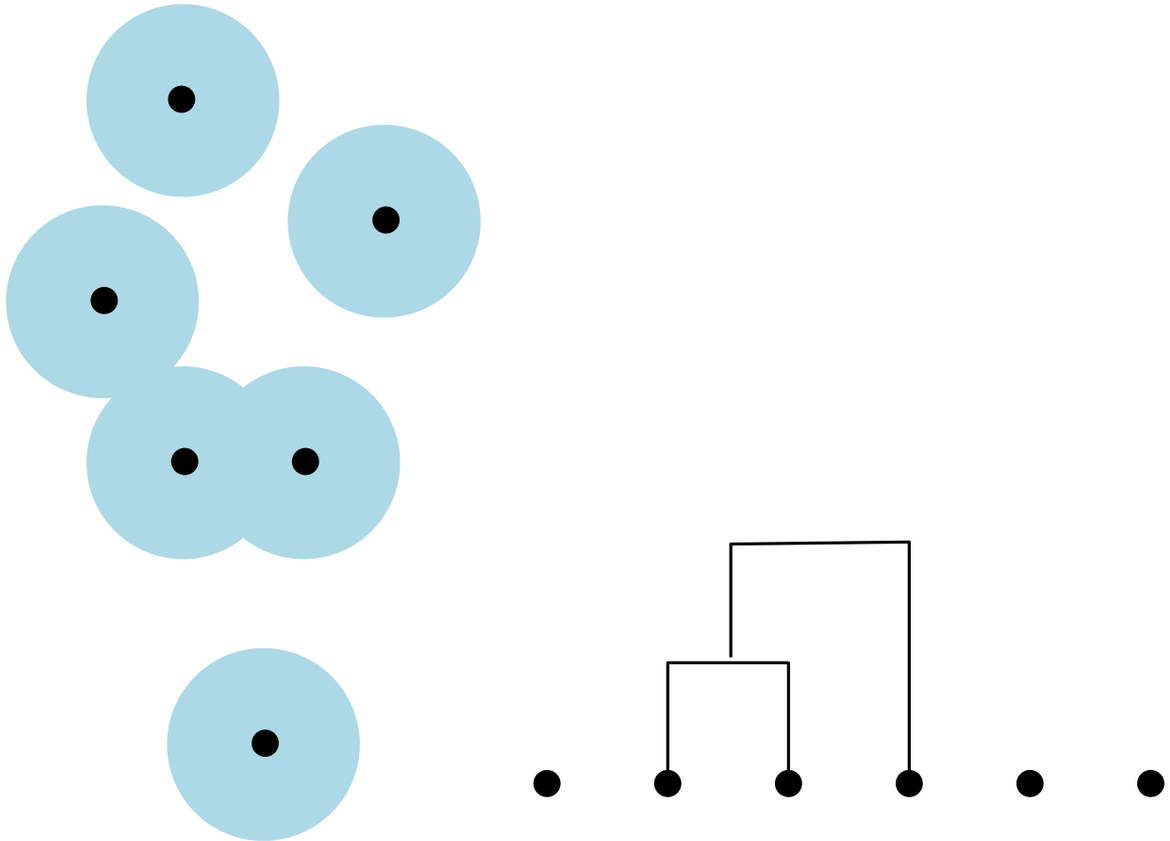
The (in)stability of dendograms



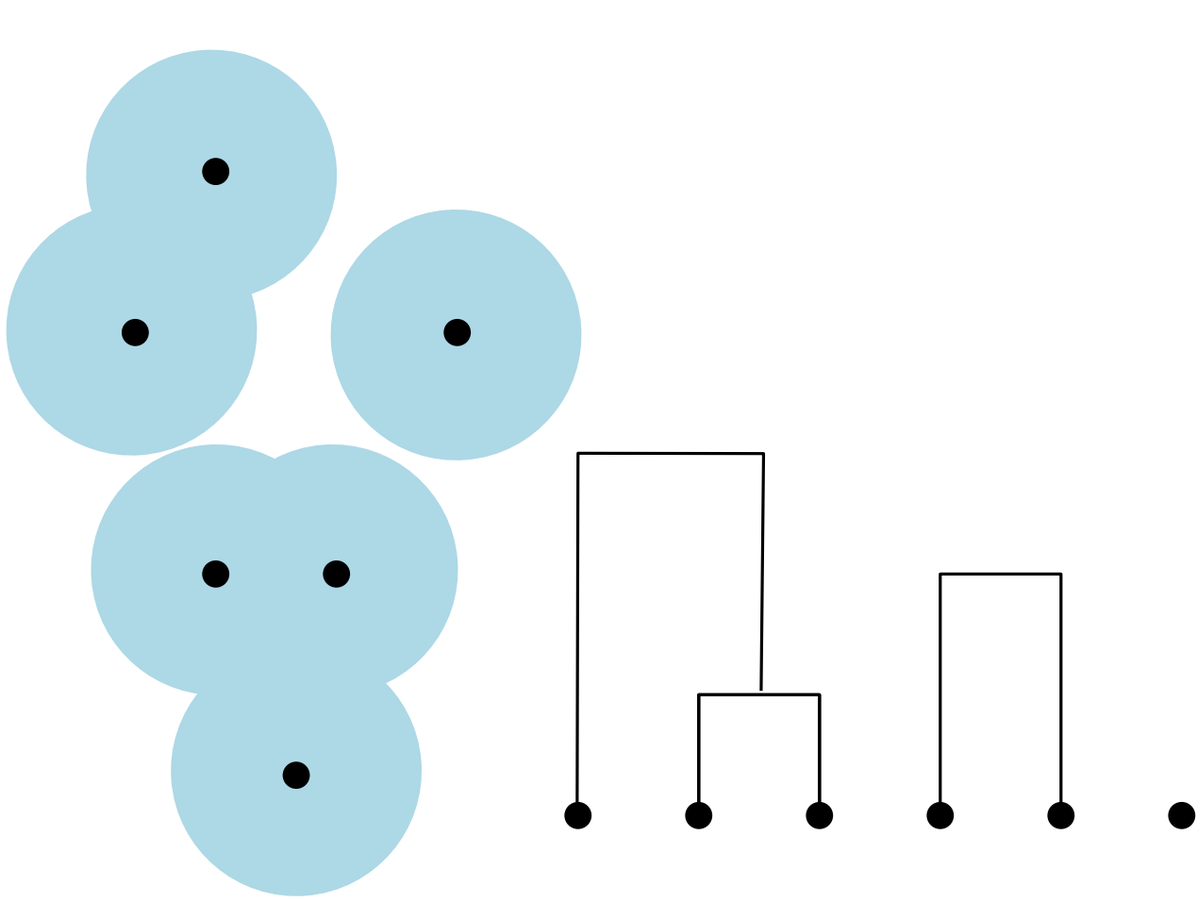


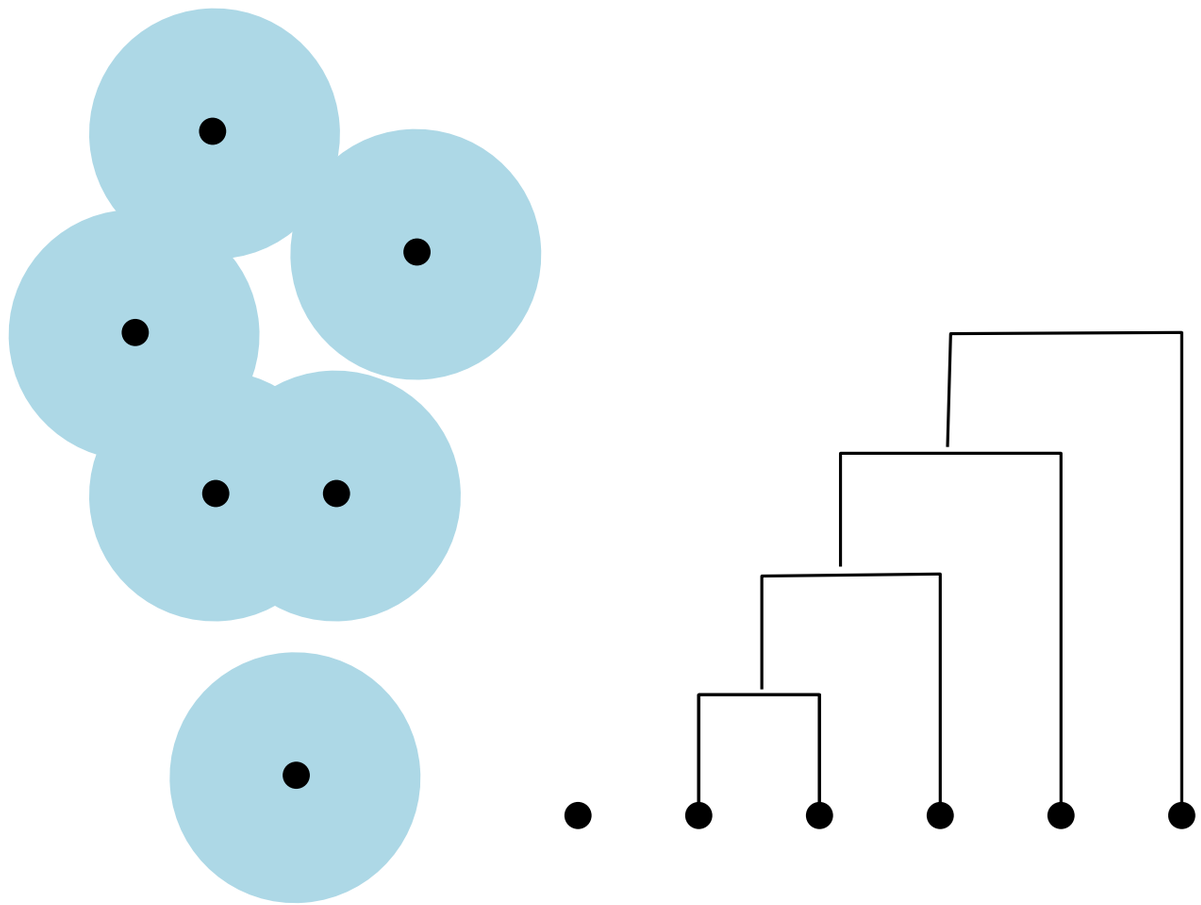
The (in)stability of dendograms



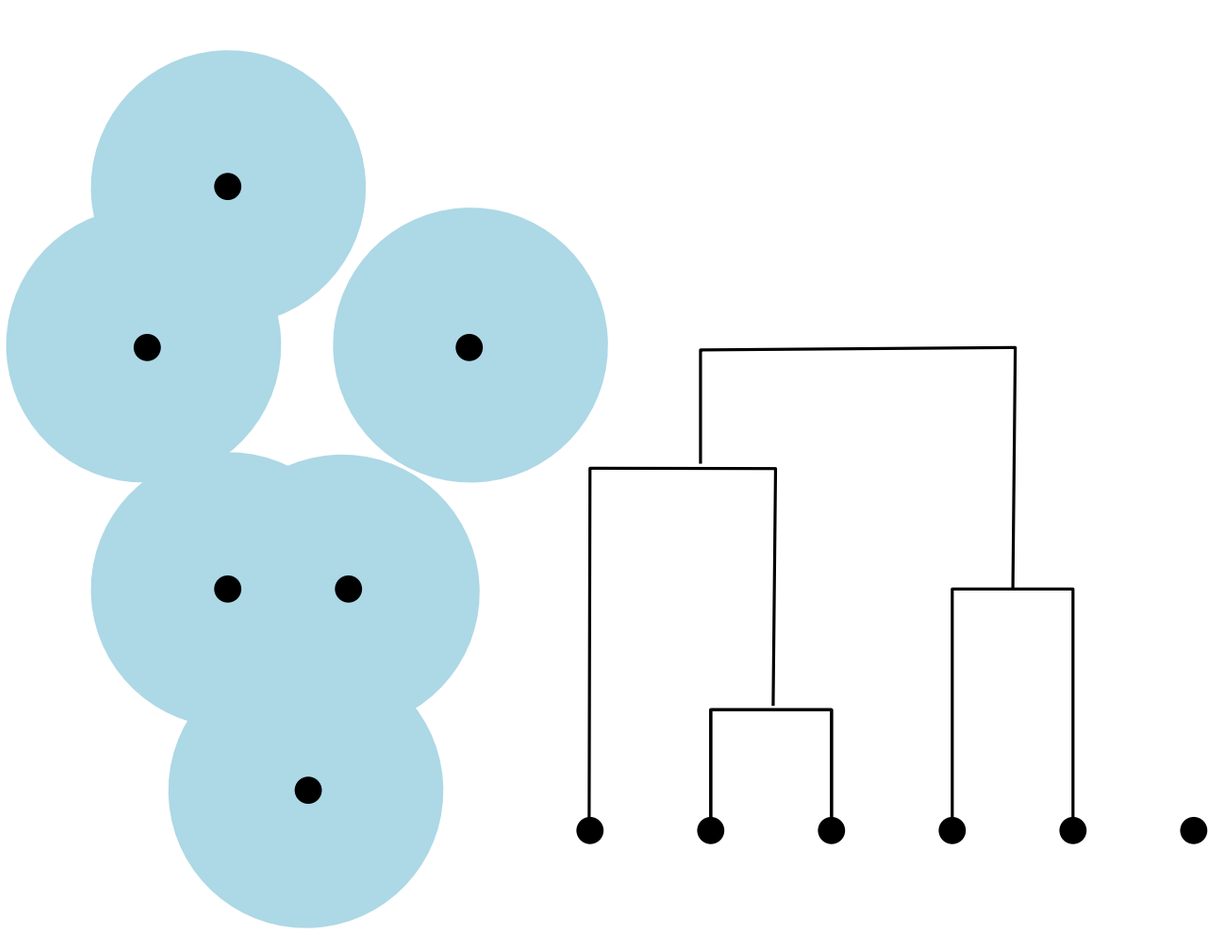


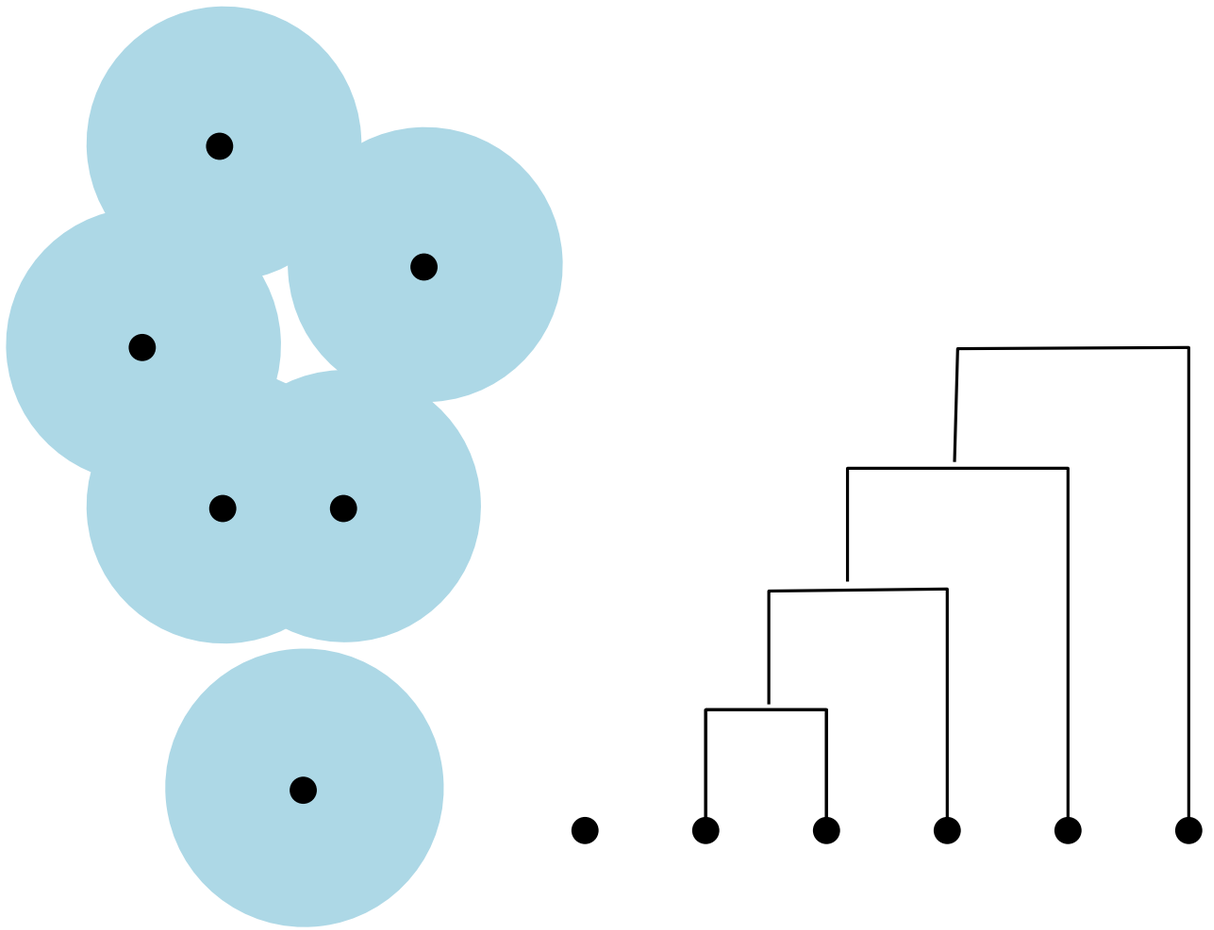
The (in)stability of dendograms



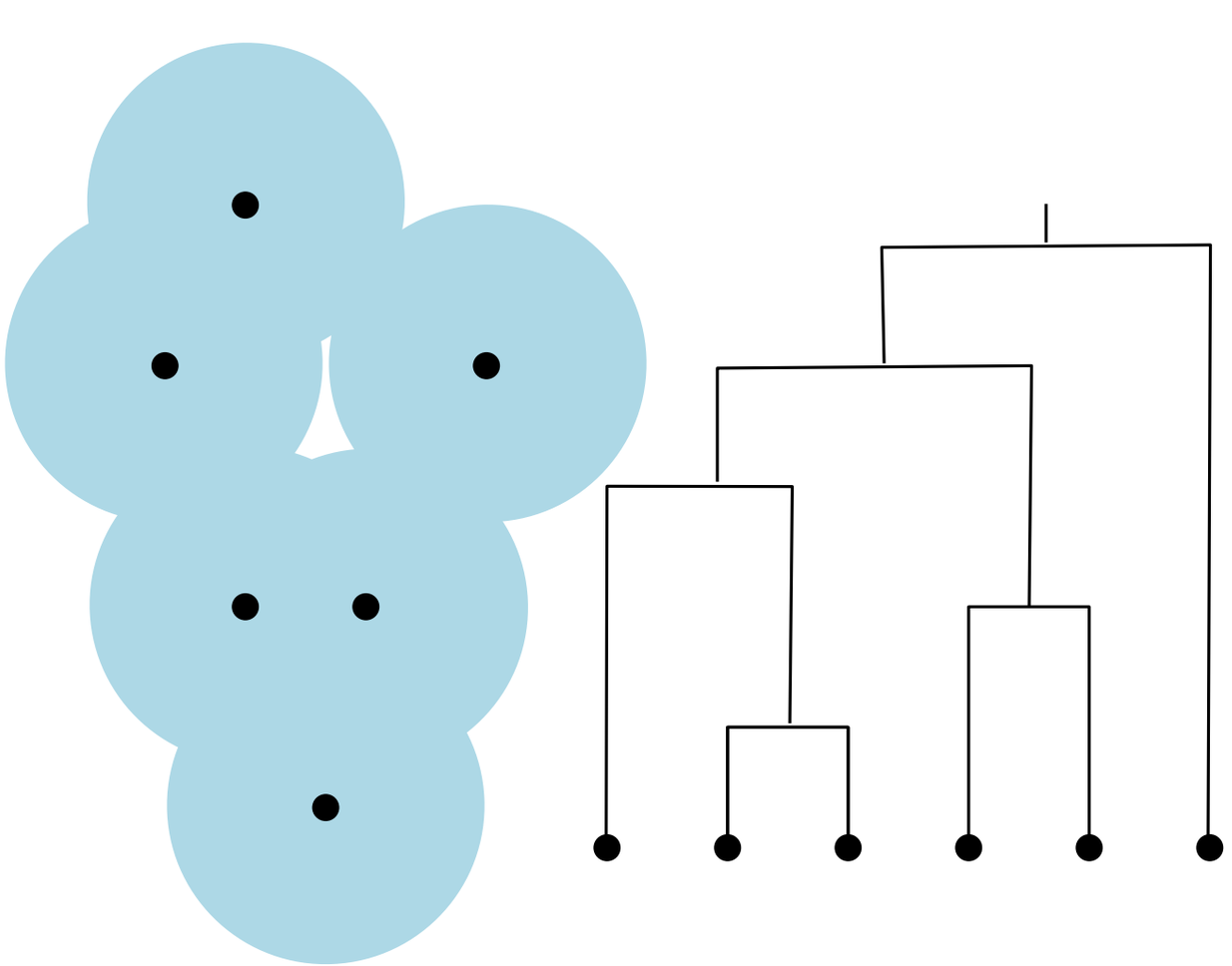


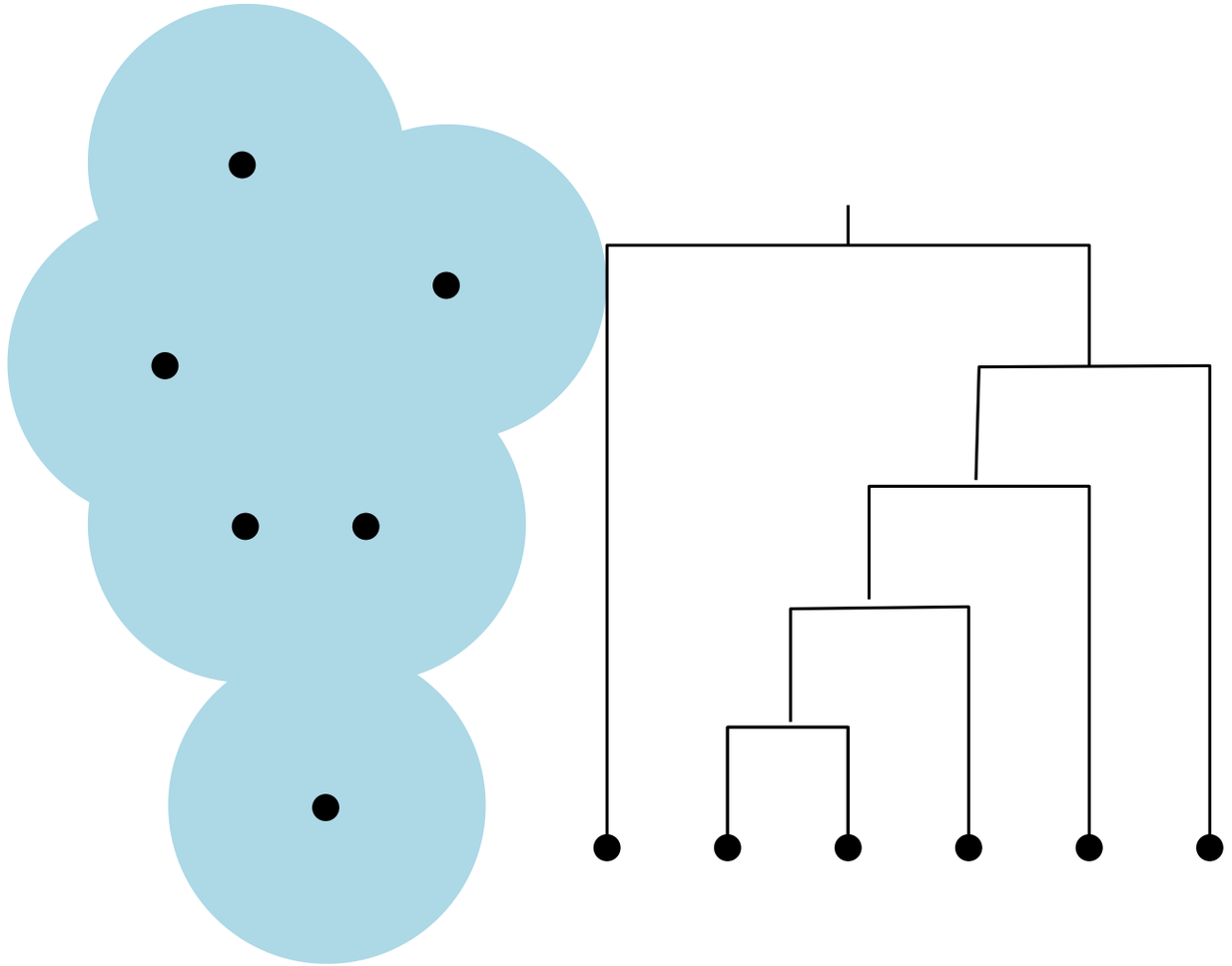
The (in)stability of dendograms





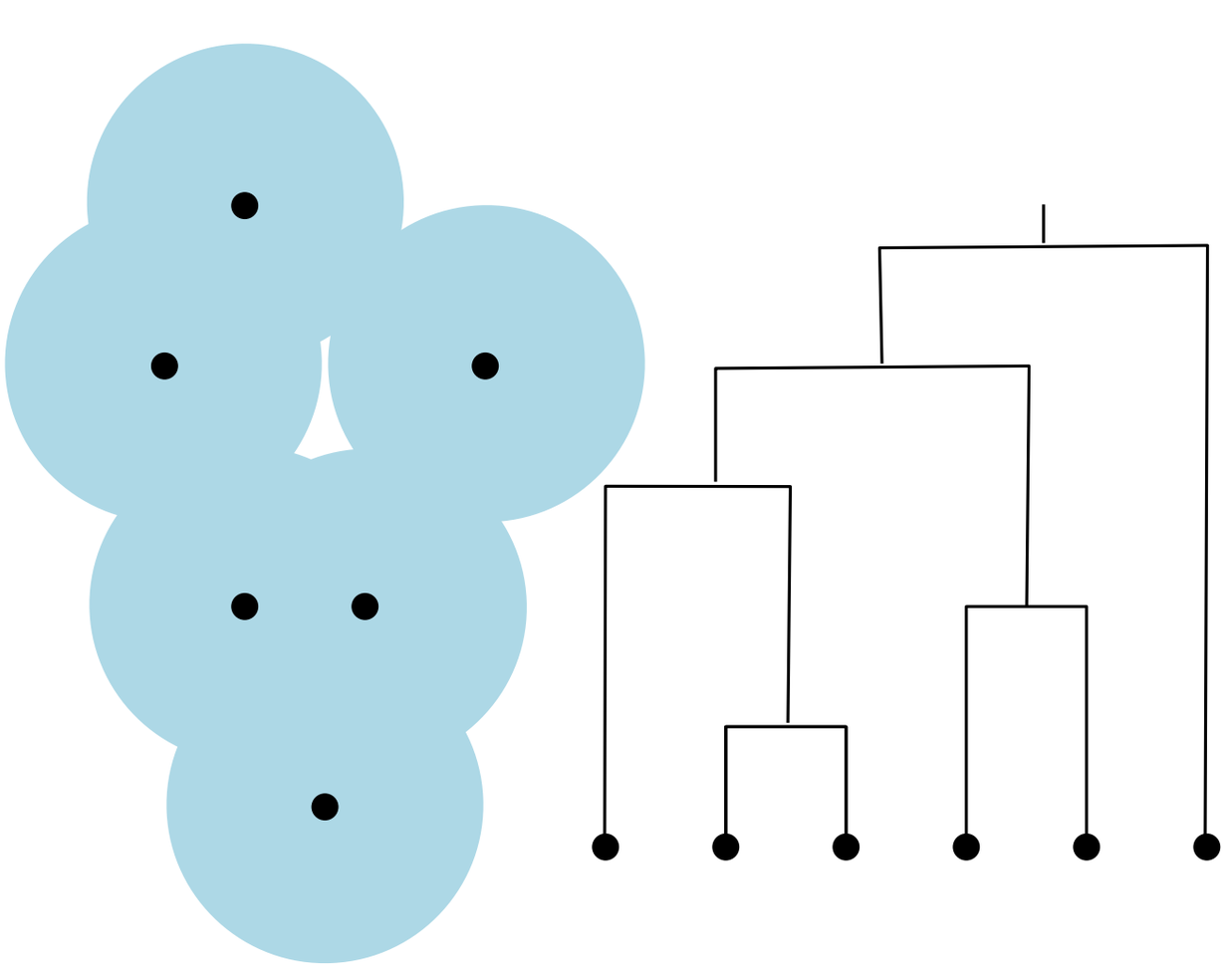
The (in)stability of dendograms

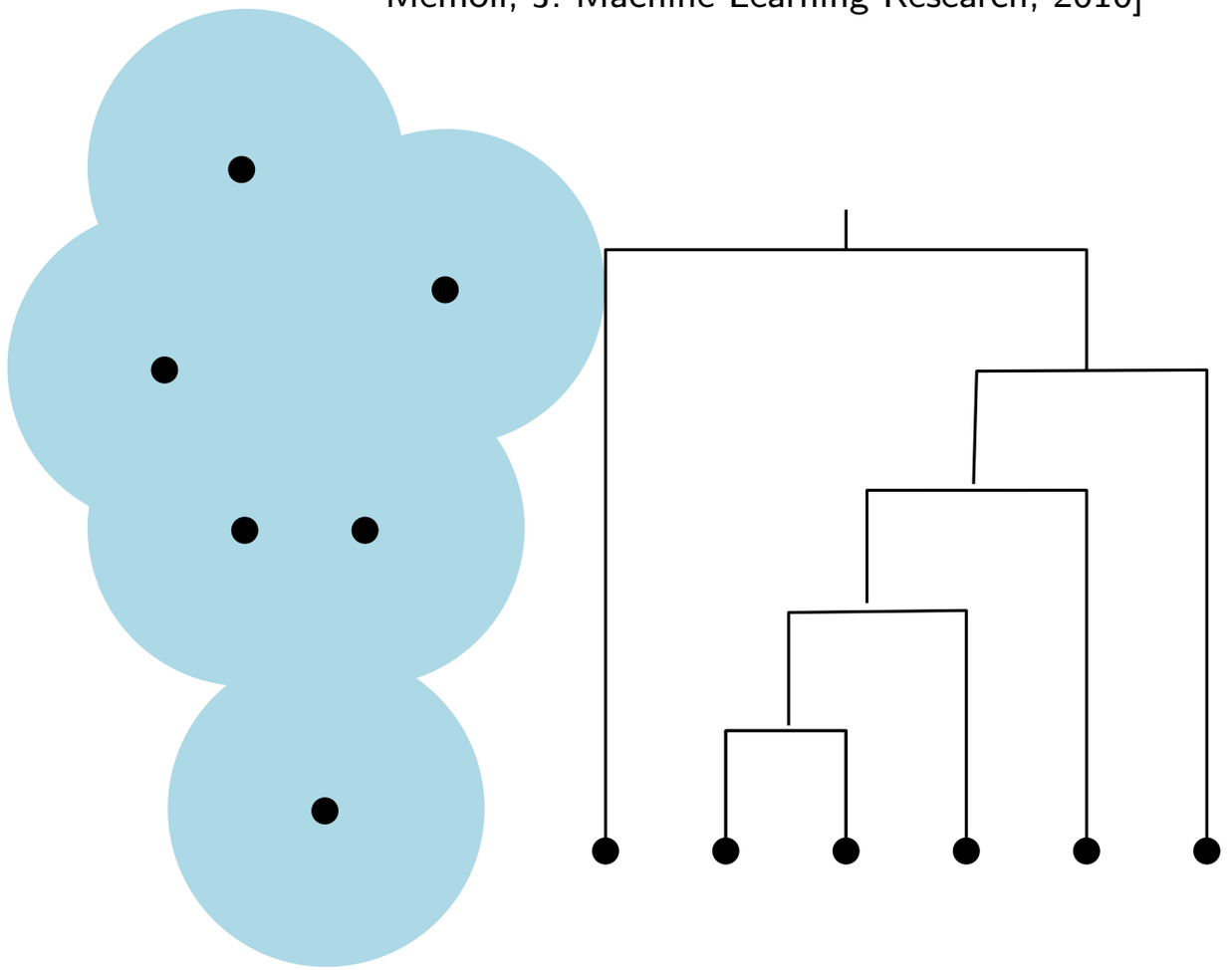




The (in)stability of dendograms

[*Characterization, Stability and Convergence of Hierarchical Clustering Methods*, Carlsson, Mémoli, J. Machine Learning Research, 2010]





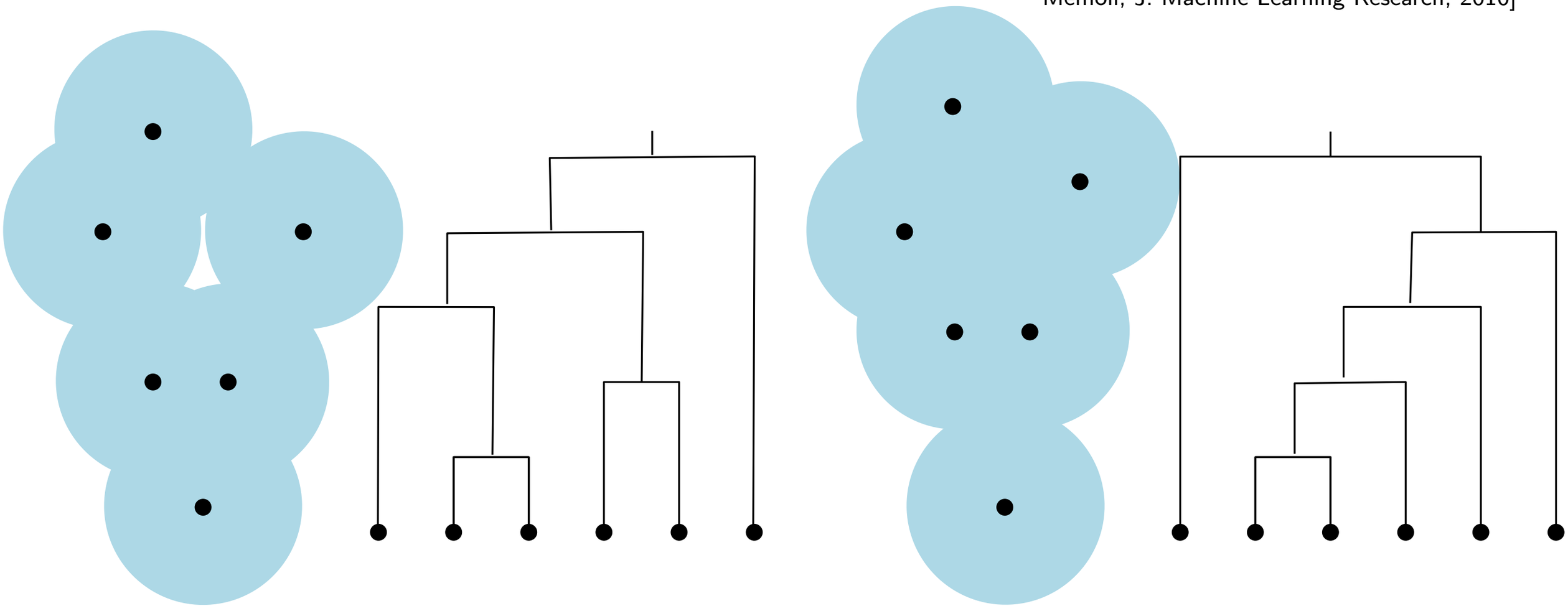
Let $d_{\mathcal{D}}(x, y) =$ height of lowest common ancestor of x, y in dendrogram $\mathcal{D}$.

ultrametric!

Thm: $d_{GH}((X, d_{\mathcal{D}}), ((Y, d_{\mathcal{D}})) \leq d_{GH}((X, d_X), (Y, d_Y))$.

The (in)stability of dendograms

[*Characterization, Stability and Convergence of Hierarchical Clustering Methods*, Carlsson, Mémoli, J. Machine Learning Research, 2010]



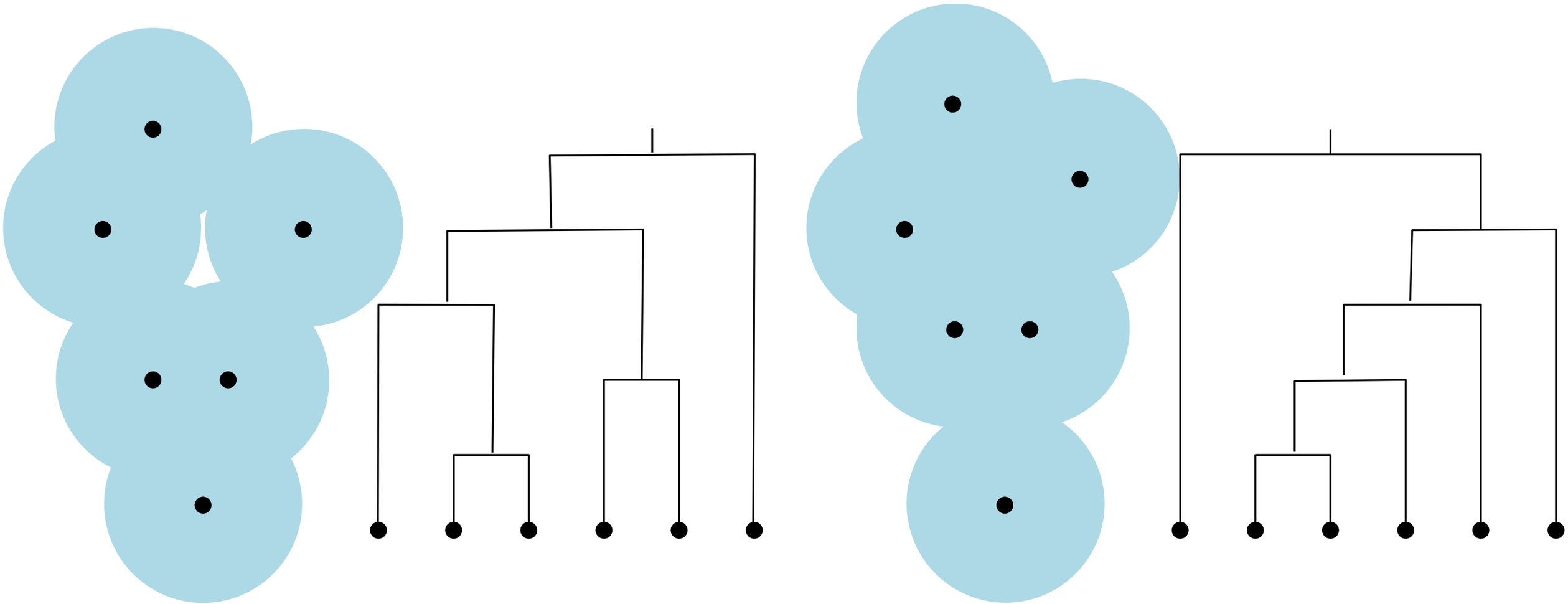
Let $d_{\mathcal{D}}(x, y) =$ height of lowest common ancestor of x, y in dendrogram $\mathcal{D}$.

ultrametric!

Thm: $d_{GH}((X, d_{\mathcal{D}}), ((Y, d_{\mathcal{D}})) \leq d_{GH}((X, d_X), (Y, d_Y))$.

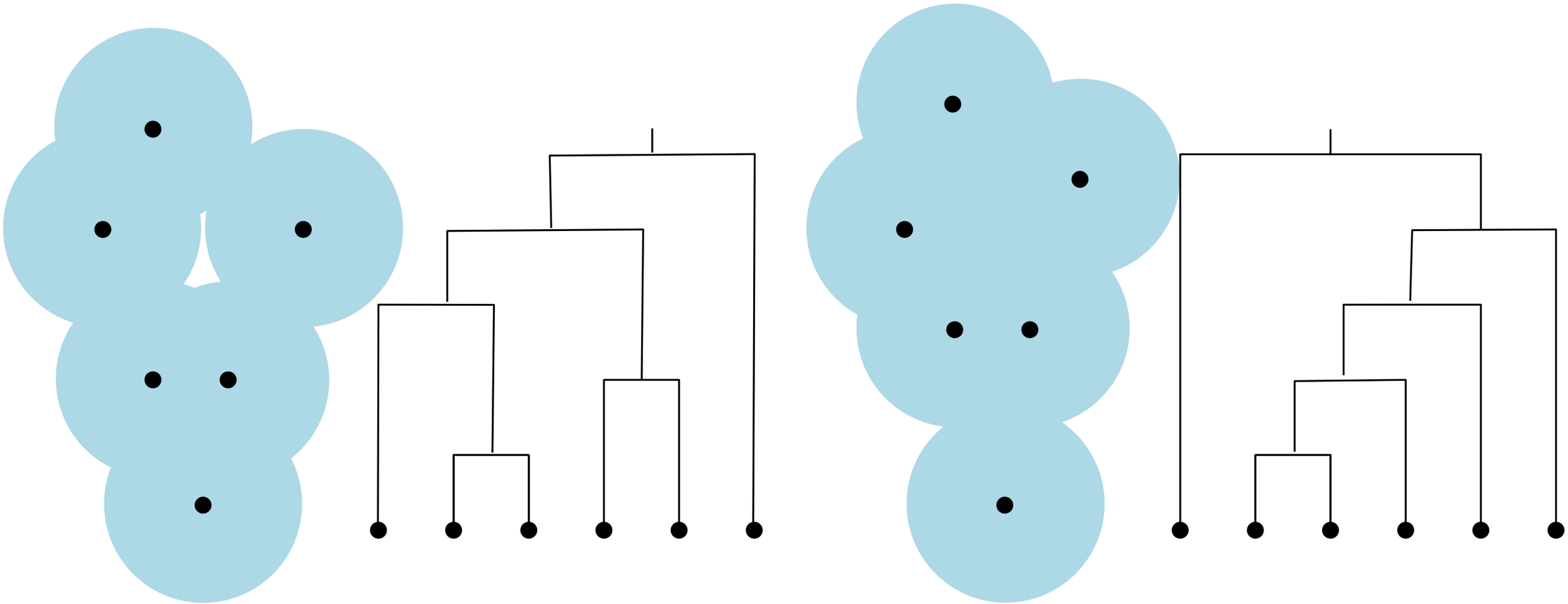
This is actually not true for complete and average clustering!

The (in)stability of dendograms



Small perturbations on the input data may lead to wide change in the structure of the trees.

The (in)stability of dendograms

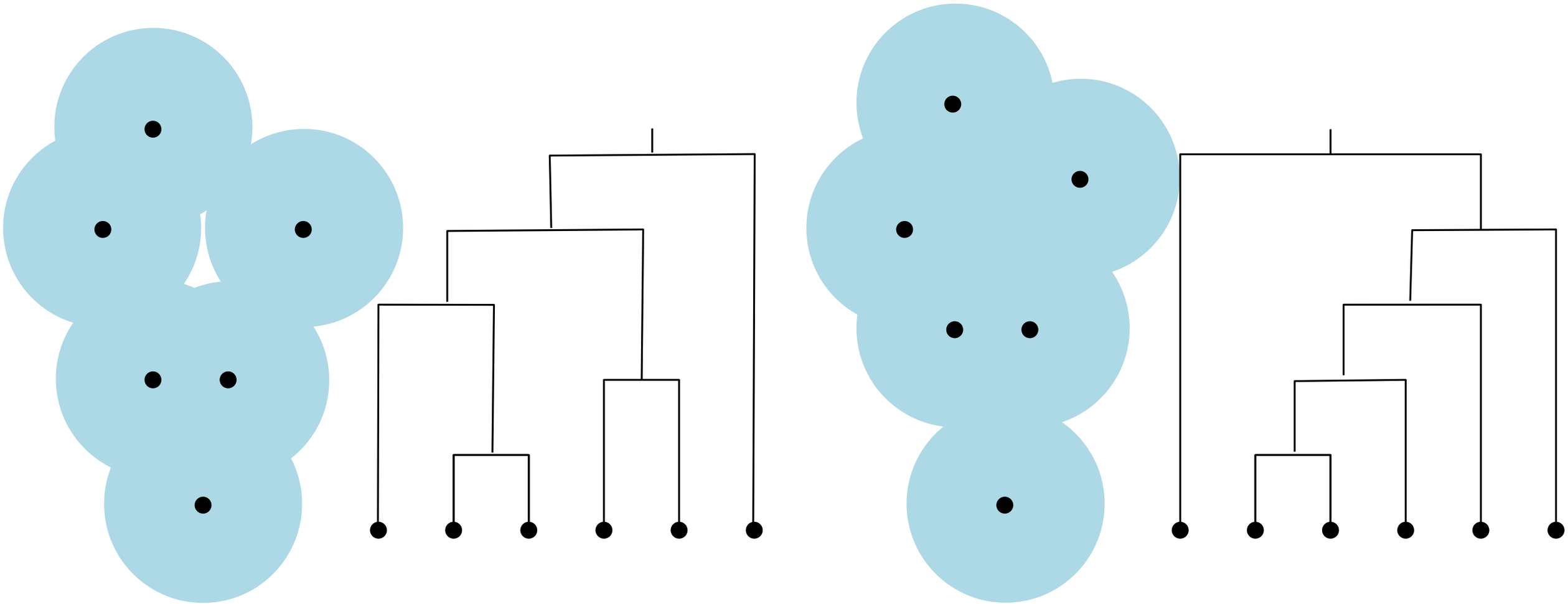


Small perturbations on the input data may lead to wide change in the structure of the trees.

However, the 'merging times' remain stable.

(For Euclidean data), single linkage clustering keeps track of the evolution of the connected components of the distance function to the data.

The (in)stability of dendograms



Small perturbations on the input data may lead to wide change in the structure of the trees.

Persistent homology!

However, the 'merging times' remain stable.

(For Euclidean data), single linkage clustering keeps track of the evolution of the connected components of the distance function to the data.

The Union-Find data structure for tracking clusters

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

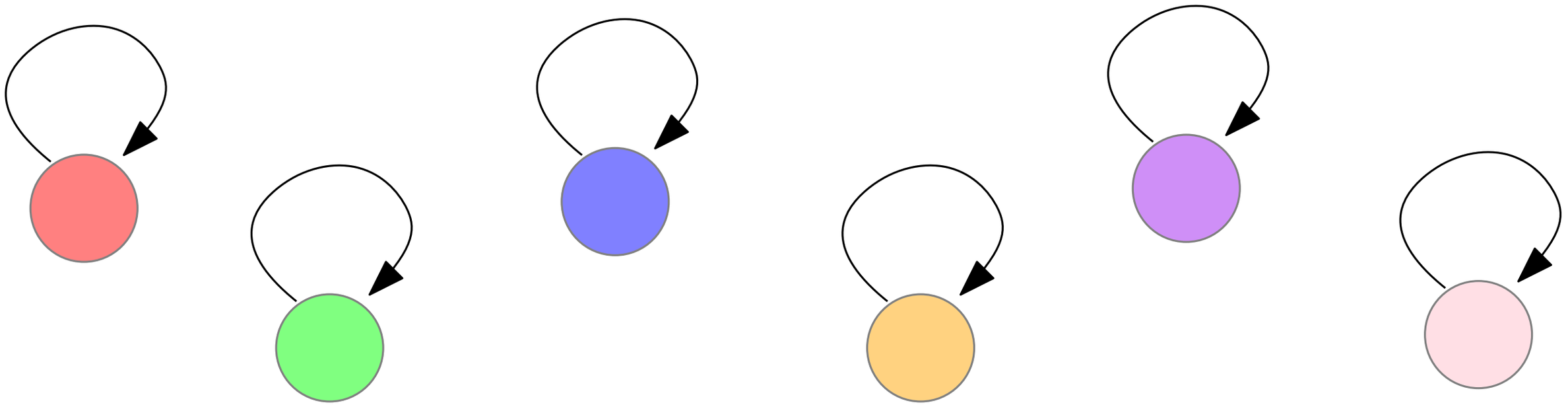
def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$



MakeSet( ) MakeSet( )

MakeSet( ) MakeSet( )

MakeSet( ) MakeSet( )

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

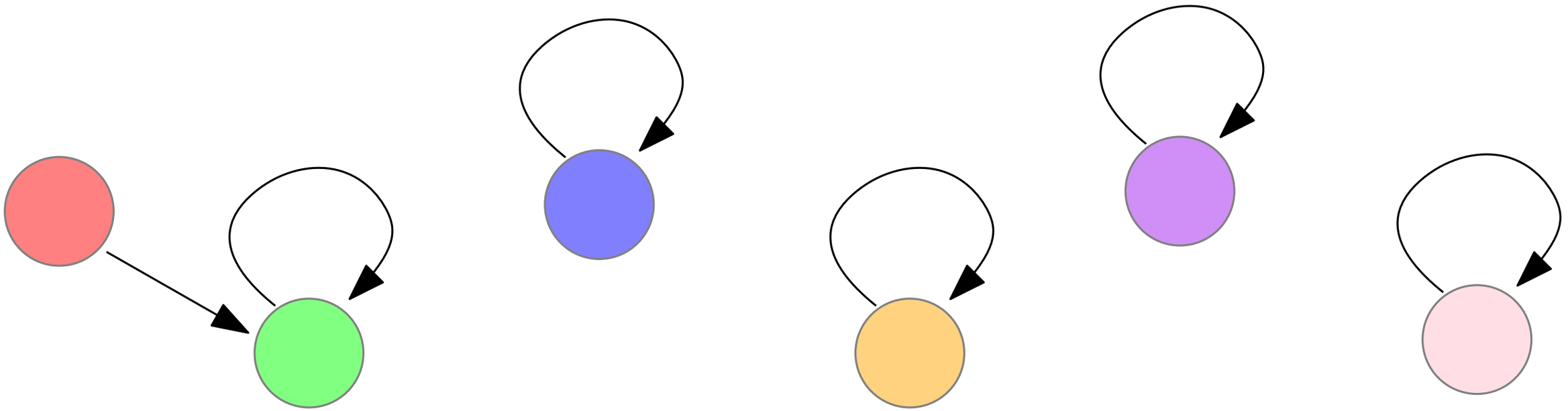
def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$



Union( )

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

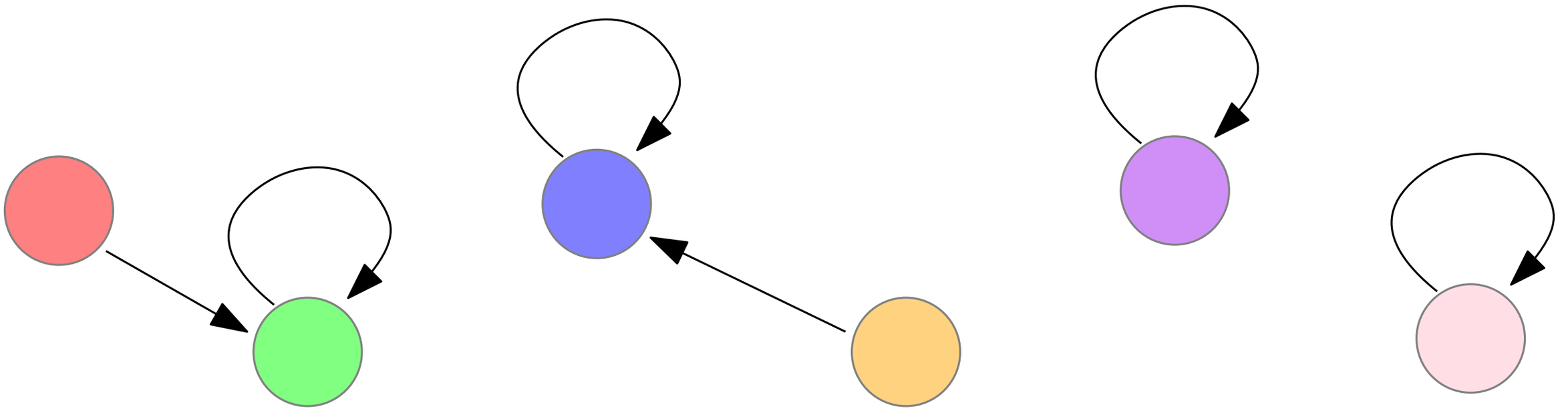
def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$



Union( )

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

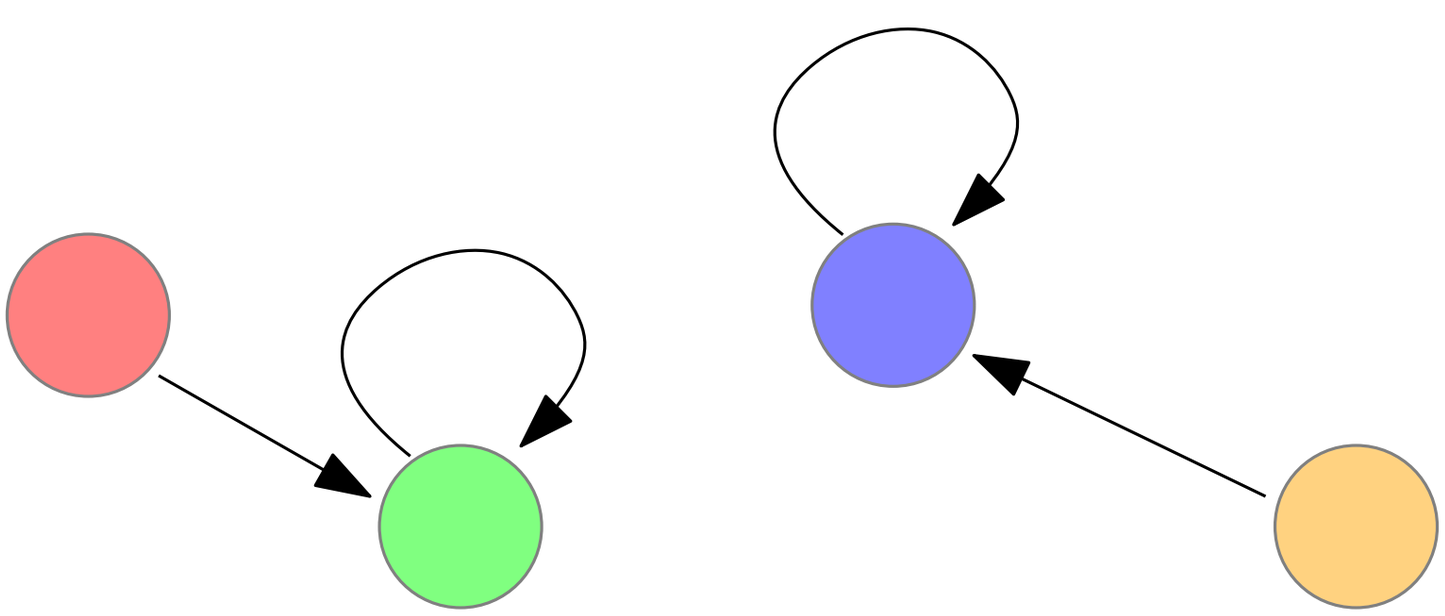
def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$



Union( )

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

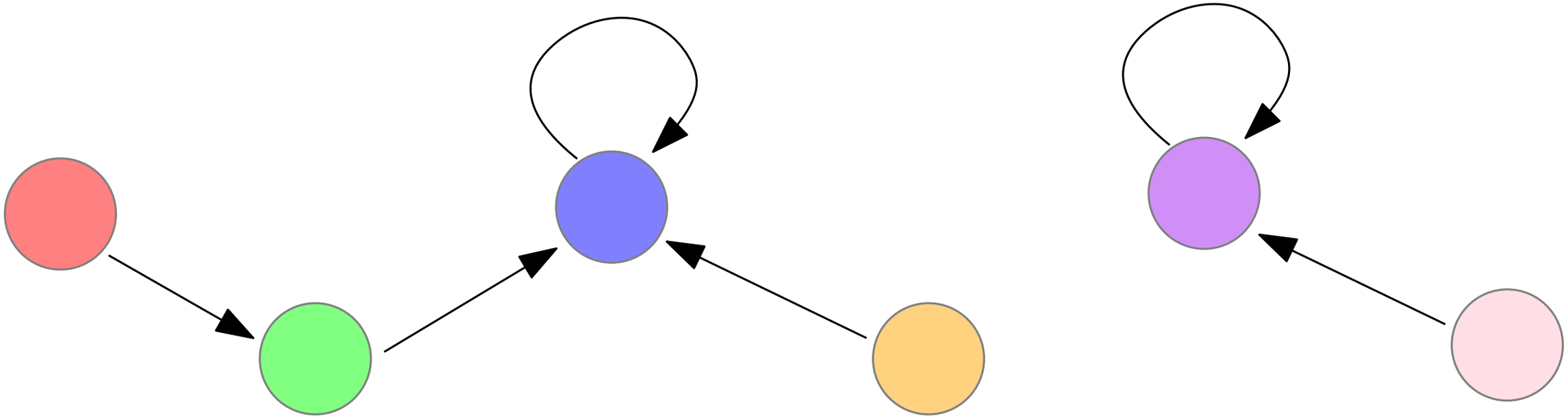
def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$



Union( )

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

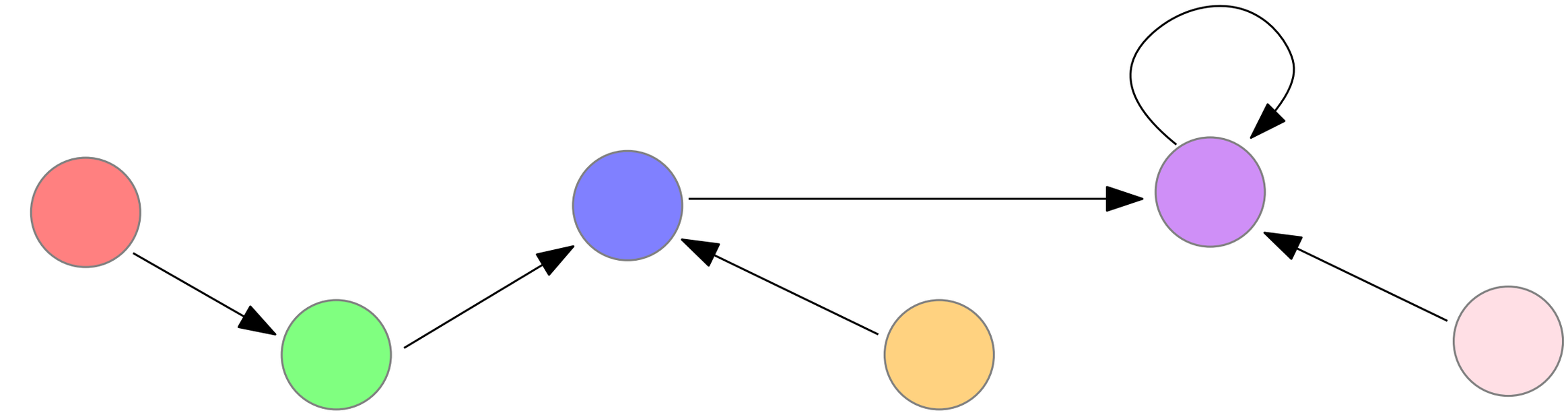
def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$



Union( )

The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

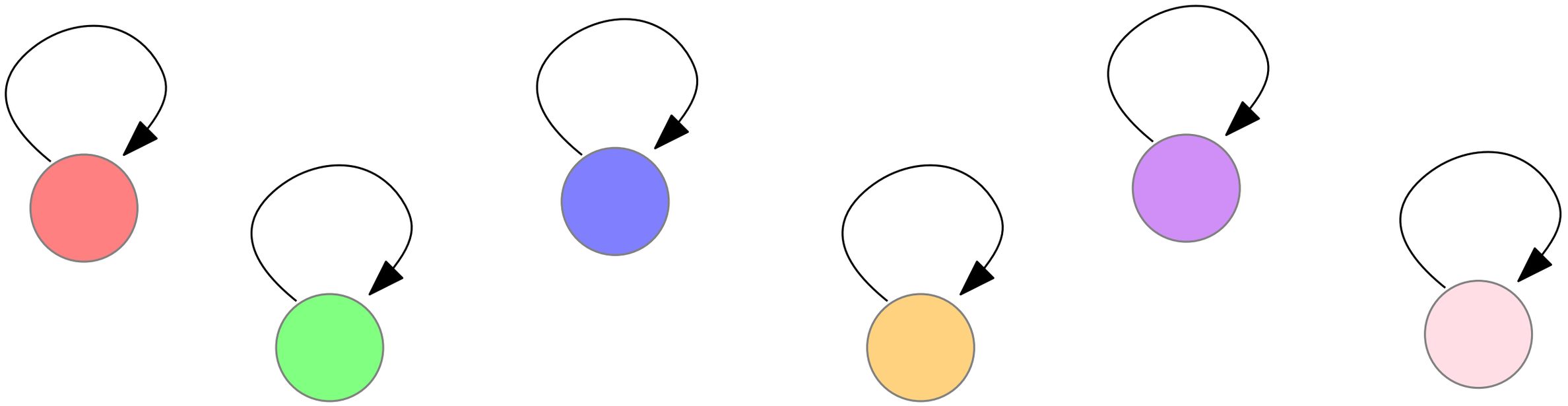
$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

Complexity is still not optimal



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

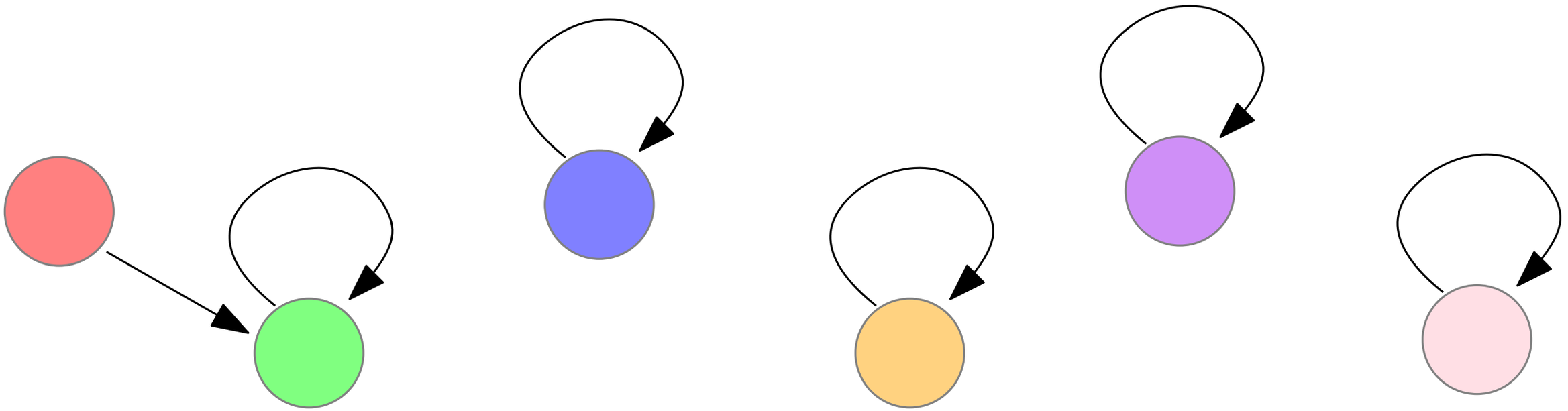
$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

Complexity is still not optimal



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

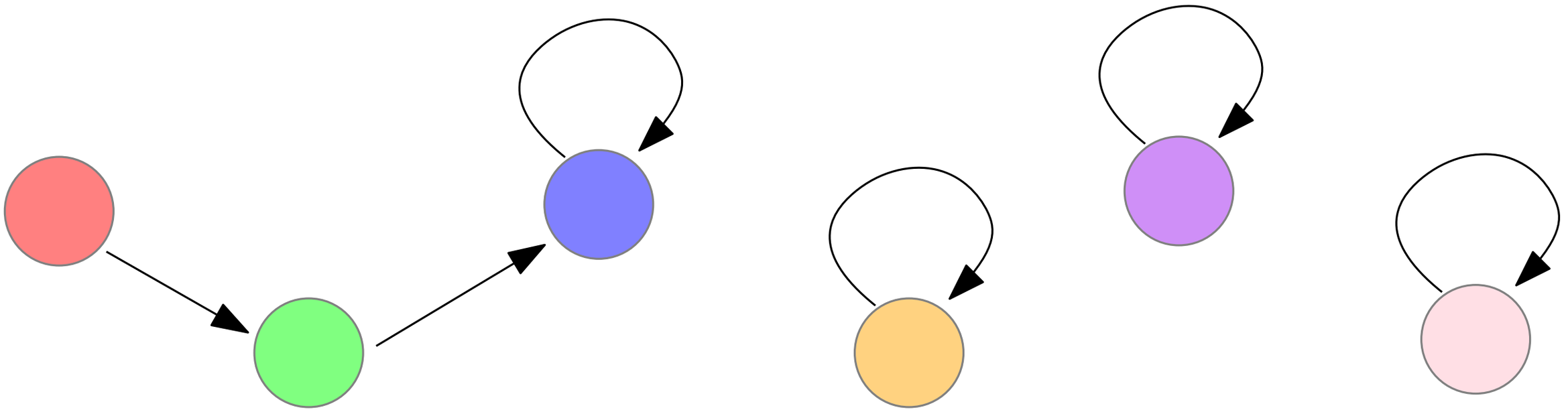
$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

Complexity is still not optimal



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

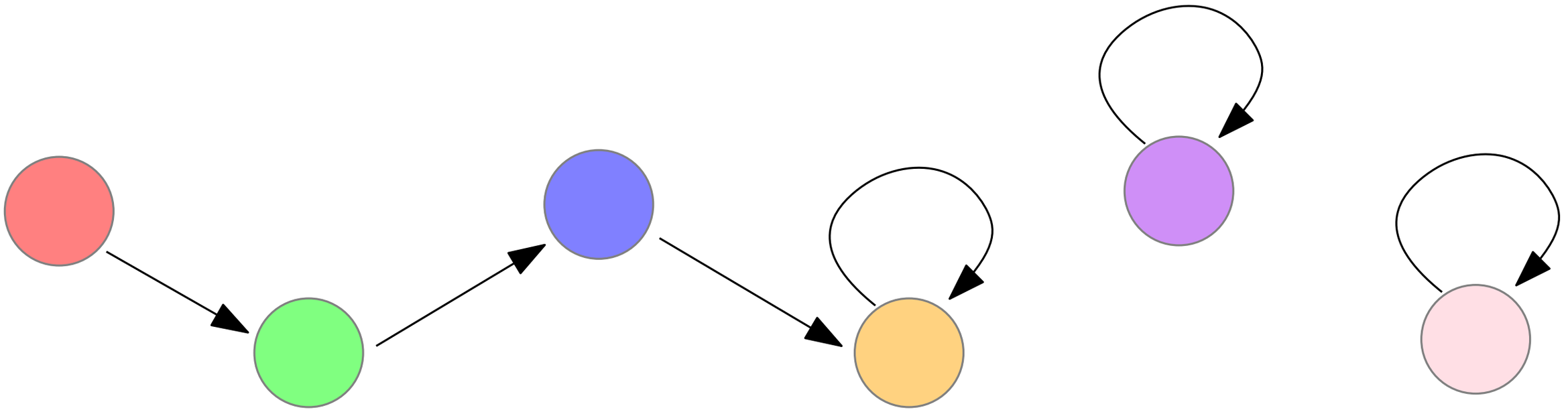
$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

Complexity is still not optimal



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

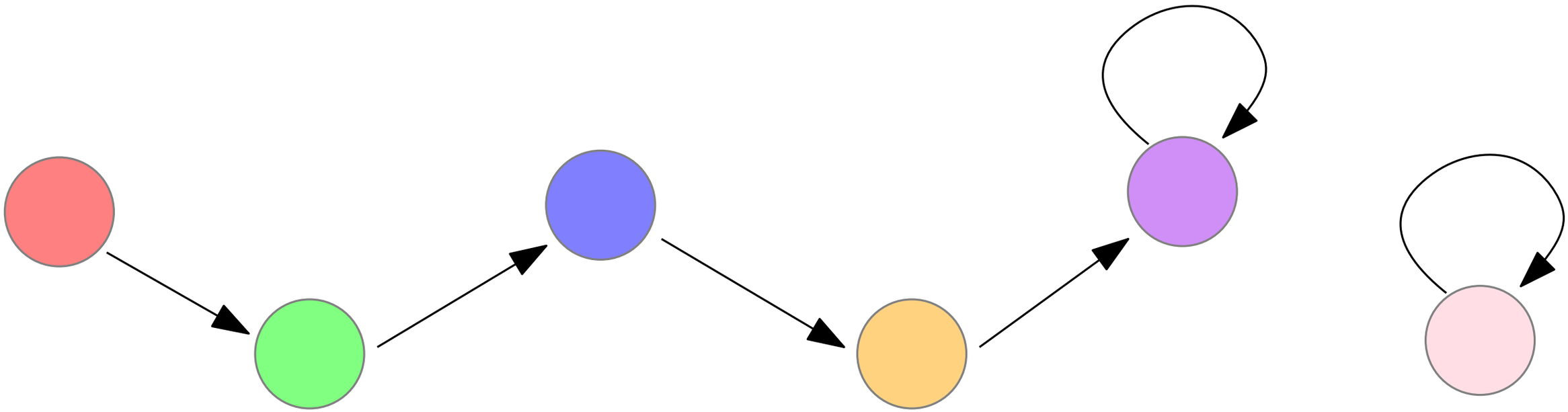
$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

Complexity is still not optimal



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

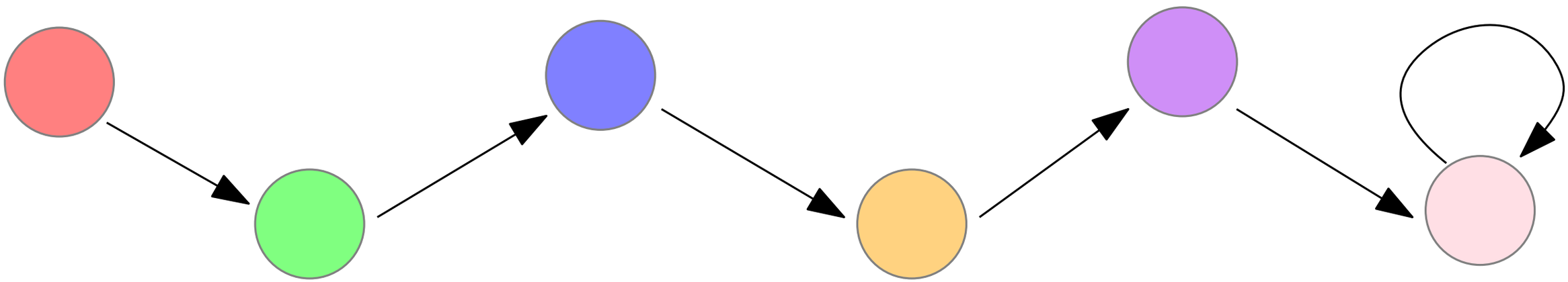
$p_x =$ Find(x)

$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

Complexity is still not optimal



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

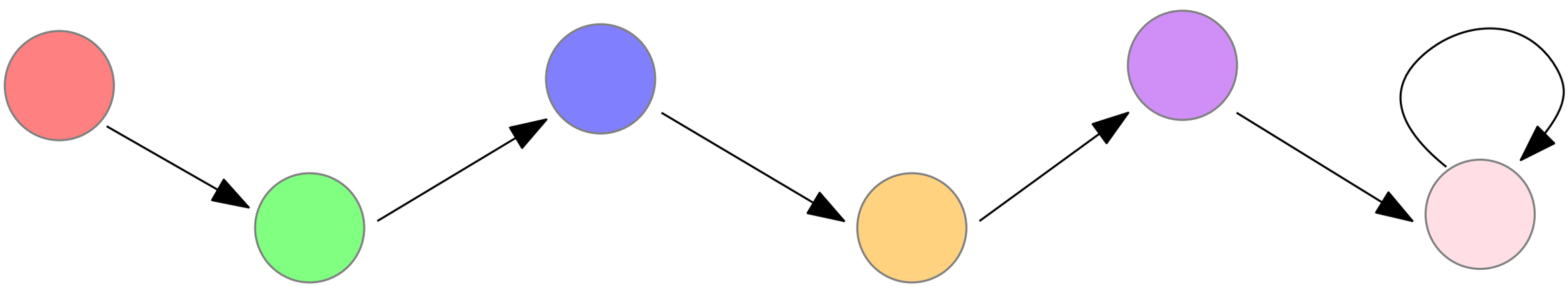
$p_y =$ Find(y)

if $p_x \neq p_y$:

parents$\{p_x\} \leftarrow p_y$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

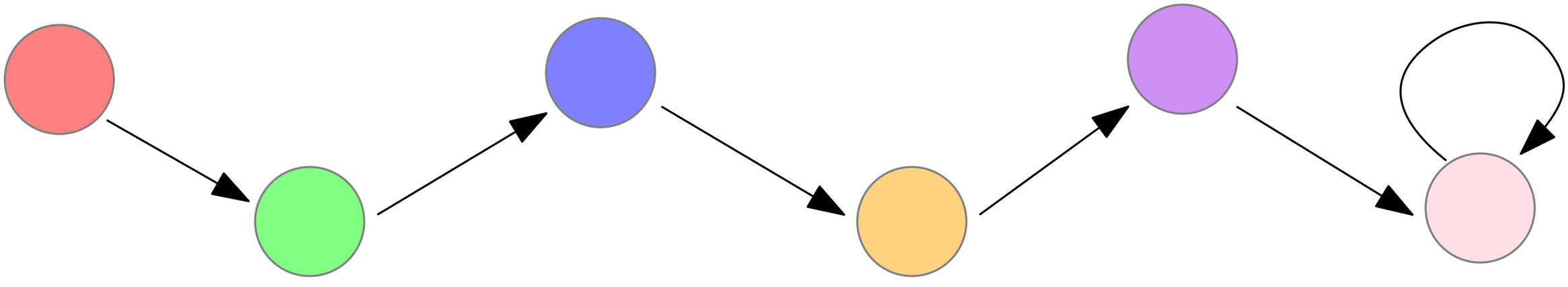
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

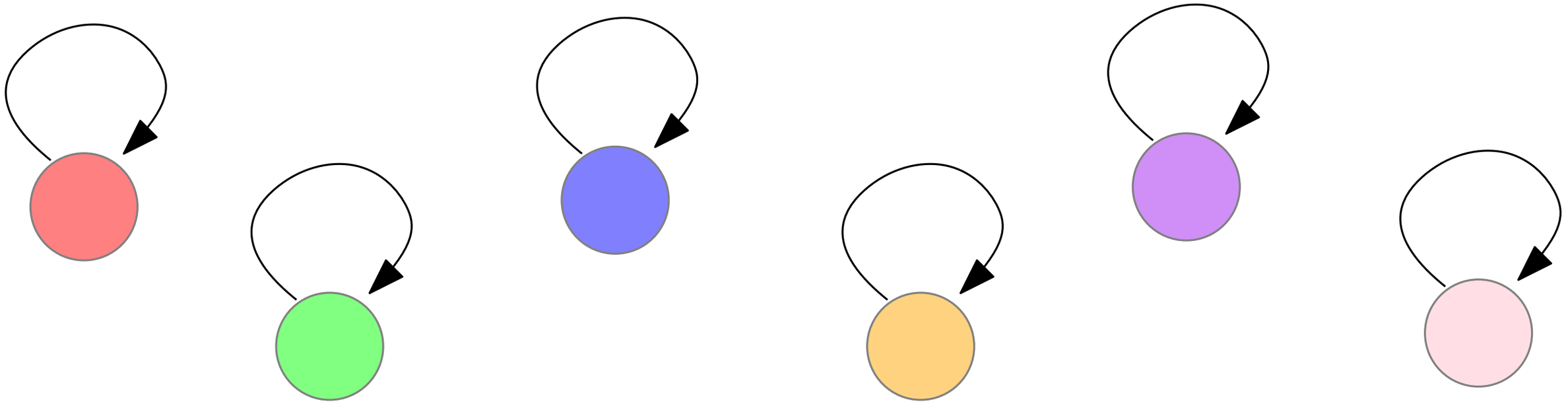
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

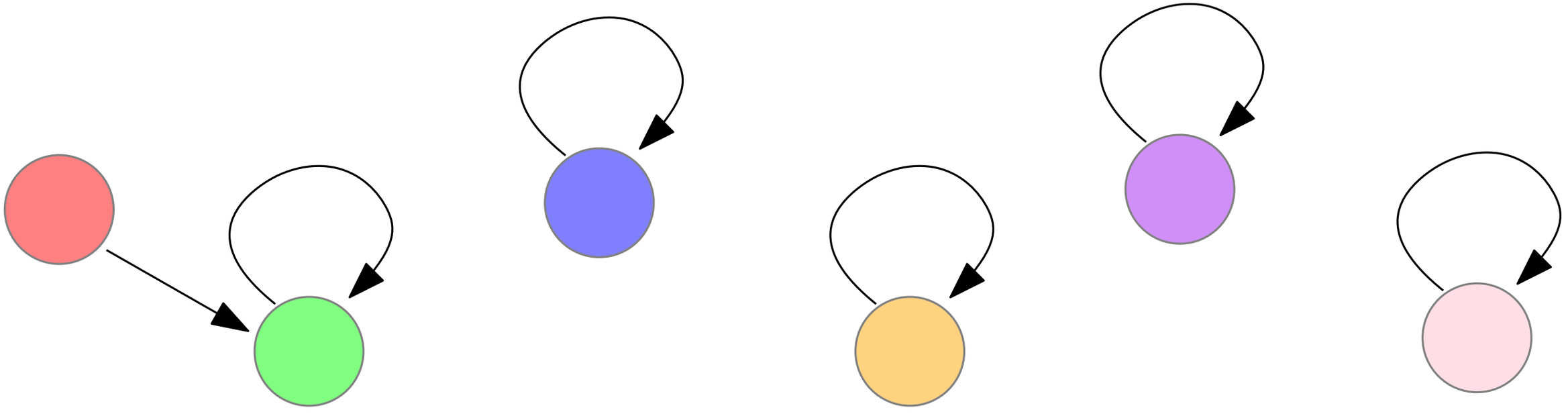
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

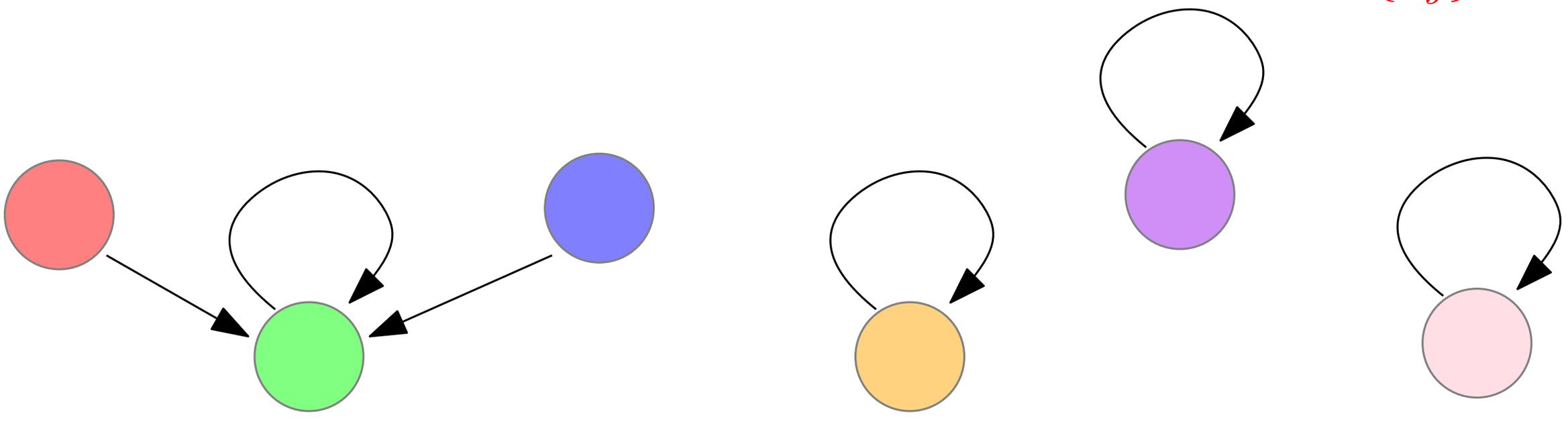
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

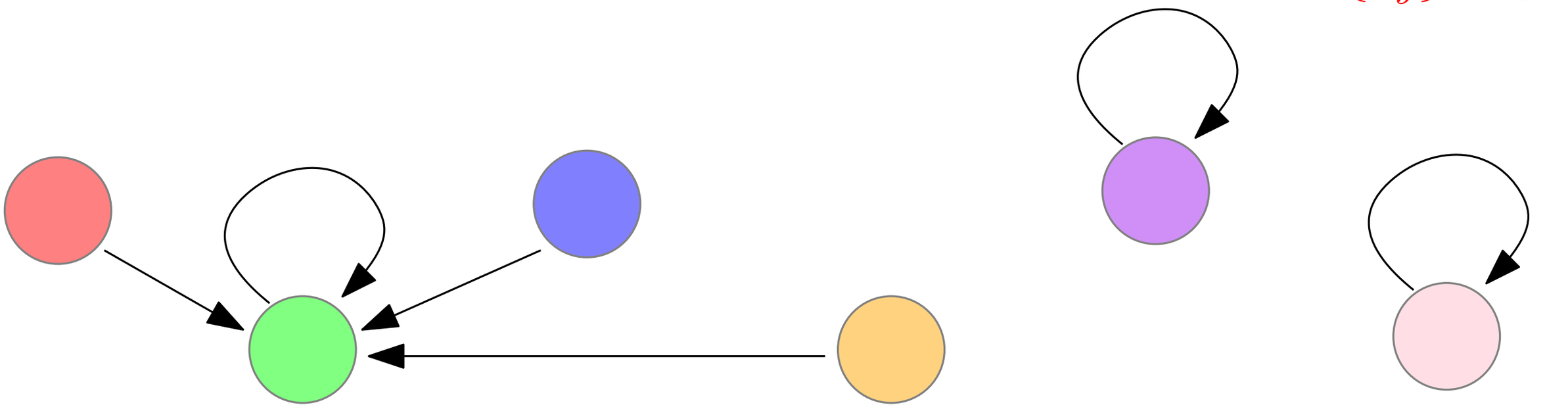
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

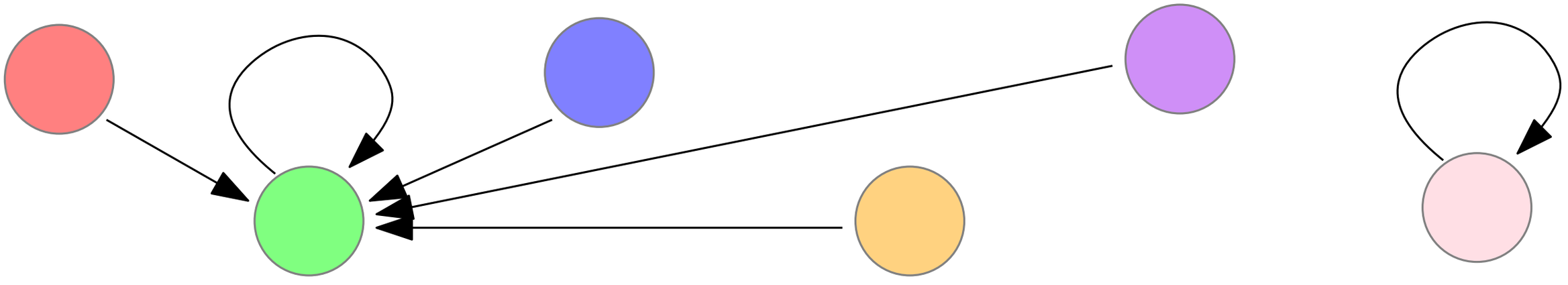
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

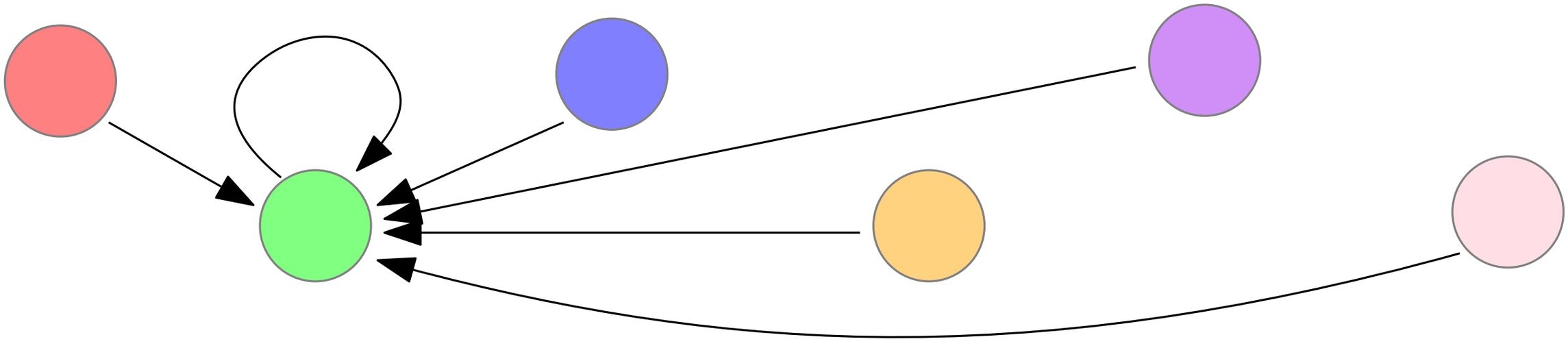
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes linear time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

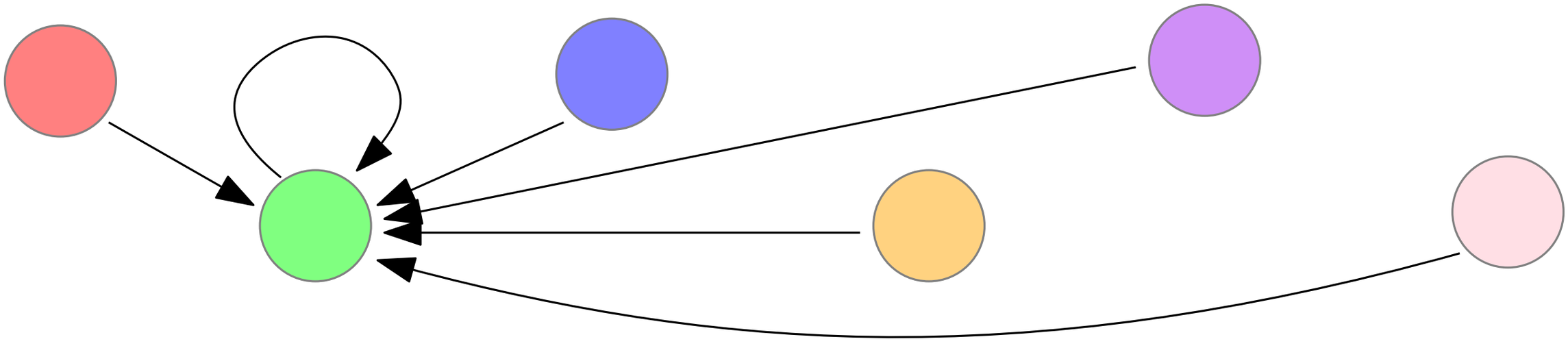
parents$\{p_x\} \leftarrow p_y$

else:

parents$\{p_y\} \leftarrow p_x$

Complexity is still not optimal

Find takes logarithmic time!



The Union-Find data structure for tracking clusters

def MakeSet(x):

parents$\{x\} \leftarrow x$

def Find(x):

if parents$\{x\} == x$:

return x

else:

return Find(parents$\{x\}$)

def Union(x,y):

$p_x =$ Find(x)

$p_y =$ Find(y)

if tree of p_x smaller than tree of p_y:

parents$\{p_x\} \leftarrow p_y$

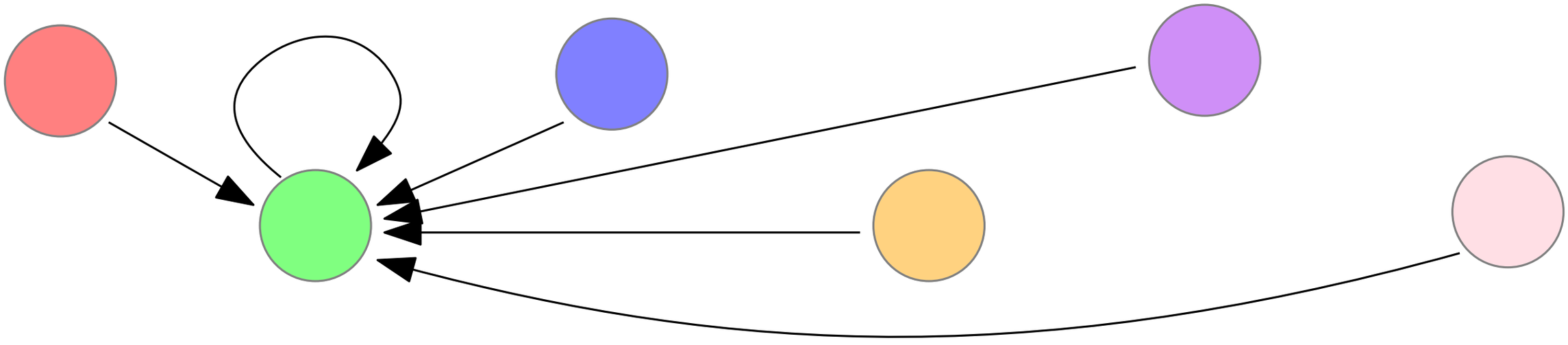
else:

parents$\{p_y\} \leftarrow p_x$

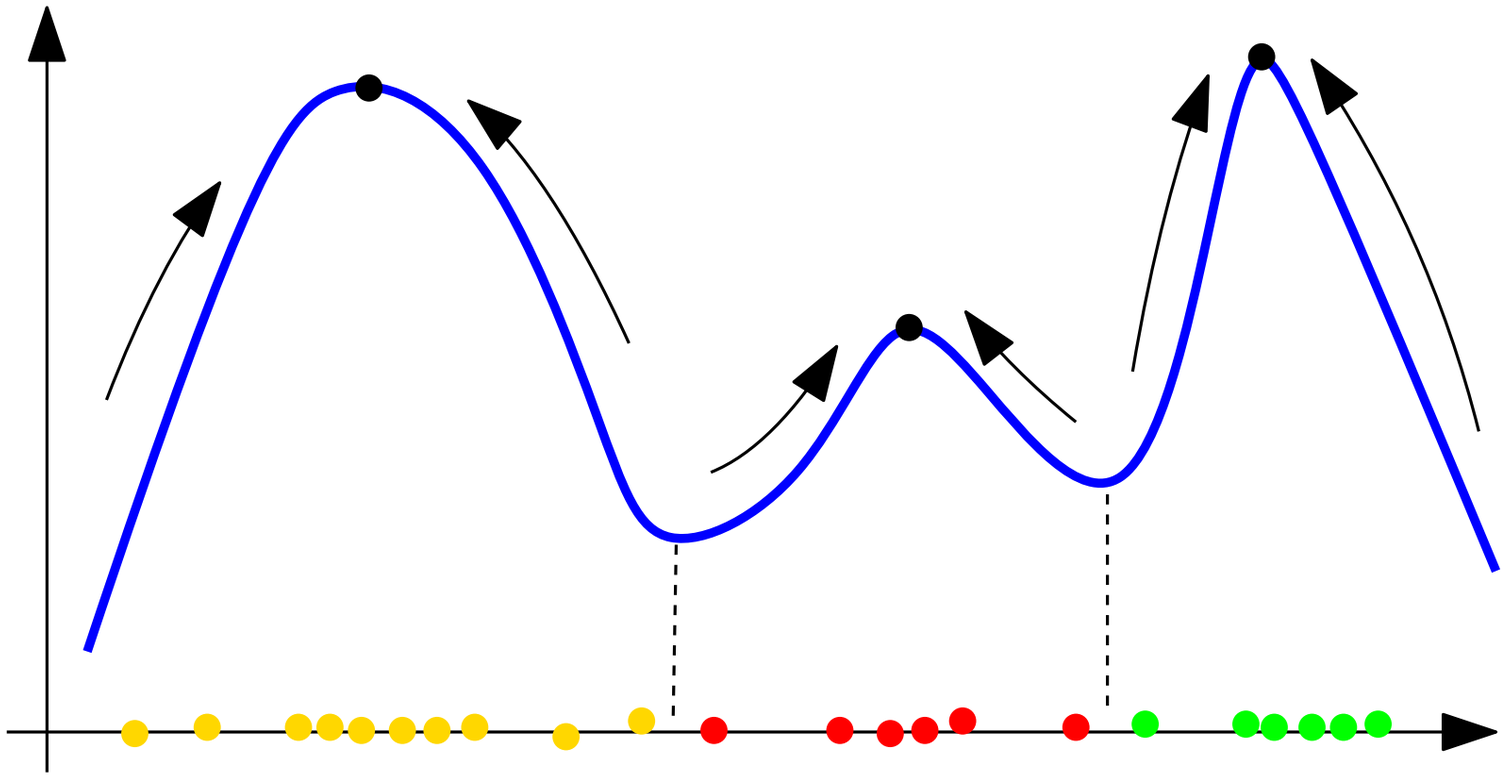
Complexity is still not optimal

$+$ path compression: Find takes $O(\alpha(n))$ time!

i.e., attach all nodes directly to their parents $\alpha =$ inverse of Ackermann function



Mode seeking clustering



- Data points are sampled according to some (unknown) probability density.
- Clusters = basins of attractions of the density.

Two approaches:

- **Iterative**, such as, e.g., Mean Shift. [*Mean shift: a robust approach toward feature space analysis*, Comaniciu et al., IEEE Trans. on Pattern Analysis and Machine Intelligence, 2002]
- **Graph-based**, such as, e.g., [*A Graph-Theoretic Approach to Nonparametric Cluster Analysis*, Koontz et al., IEEE Trans. on Computers, 1976].

Mean Shift

Mean Shift

1. Pick random guess $x \in X$.

Mean Shift

1. Pick random guess $x \in X$.
2. Compute $$M(x) = \frac{\sum_{x_i \in N(x)} K(x,x_i) \cdot x_i}{\sum_{x_i \in N(x)} K(x,x_i)},$$

where $N(x)$ is a neighborhood of x, and K is a kernel, e.g., Gaussian kernel $K(x,y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$.

Mean Shift

1. Pick random guess $x \in X$.
2. Compute $$M(x) = \frac{\sum_{x_i \in N(x)} K(x,x_i) \cdot x_i}{\sum_{x_i \in N(x)} K(x,x_i)},$$

where $N(x)$ is a neighborhood of x, and K is a kernel, e.g., Gaussian kernel $K(x,y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$.

3. Update $x \leftarrow M(x)$.

Mean Shift

1. Pick random guess $x \in X$.
2. Compute $$M(x) = \frac{\sum_{x_i \in N(x)} K(x,x_i) \cdot x_i}{\sum_{x_i \in N(x)} K(x,x_i)},$$

where $N(x)$ is a neighborhood of x, and K is a kernel, e.g., Gaussian kernel $K(x,y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$.

3. Update $x \leftarrow M(x)$.

Do that for many random guesses, postprocess and merge similar centroids, and use final centroid distances to assess clusters.

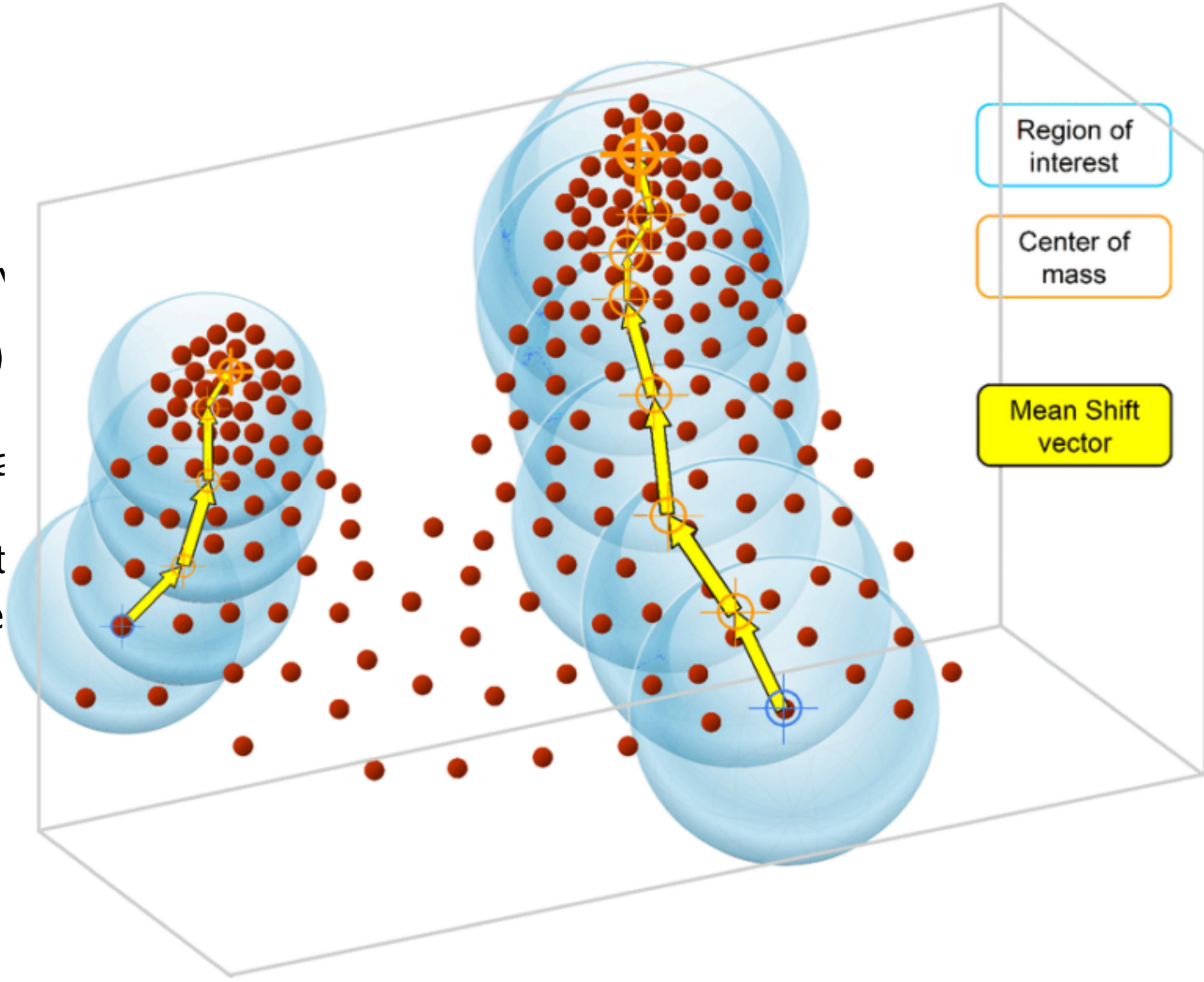
Mean Shift

1. Pick
2. Com

where M n kernel $K(x, y)$

3. Upda

Do that ntroids, and use



Mean Shift

1. Pick random guess $x \in X$.
2. Compute $$M(x) = \frac{\sum_{x_i \in N(x)} K(x,x_i) \cdot x_i}{\sum_{x_i \in N(x)} K(x,x_i)},$$

where $N(x)$ is a neighborhood of x, and K is a kernel, e.g., Gaussian kernel $K(x,y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$.

3. Update $x \leftarrow M(x)$.

Do that for K random guesses, and use final centroid distances to assess clusters.

Mean Shift

1. Pick random guess $x \in X$.
2. Compute $$M(x) = \frac{\sum_{x_i \in N(x)} K(x,x_i) \cdot x_i}{\sum_{x_i \in N(x)} K(x,x_i)},$$

where $N(x)$ is a neighborhood of x, and K is a kernel, e.g., Gaussian kernel $K(x,y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$.

3. Update $x \leftarrow M(x)$.

Do that for K random guesses, and use final centroid distances to assess clusters.

Given data set X, latent variables Z, unknown parameters θ, and likelihood $L(\theta; X, Z) = p(X, Z \mid \theta)$, the EM algorithm seeks to find the MLE with:

Expectation step (E step): compute $Q(\theta \mid \theta^{(t)}) = E_{Z|X,\theta^{(t)}}\left[\log L(\theta; X, Z)\right]$

Maximization step (M step): Find $\theta^{(t+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(t)})$

Mean Shift

1. Pick random guess $x \in X$.
2. Compute $$M(x) = \frac{\sum_{x_i \in N(x)} K(x,x_i) \cdot x_i}{\sum_{x_i \in N(x)} K(x,x_i)},$$

where $N(x)$ is a neighborhood of x, and K is a kernel, e.g., Gaussian kernel $K(x,y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$.

3. Update $x \leftarrow M(x)$.

cluster affiliation

centroid positions + bandwidths

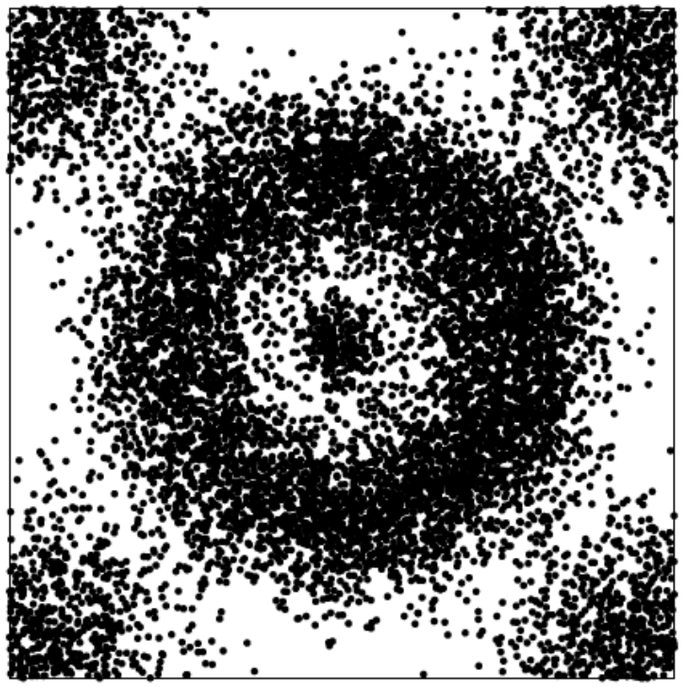
Do that for K random guesses, and use final centroid distances to assess clusters.

Given data set X, latent variables Z, unknown parameters θ, and likelihood $L(\theta; X, Z) = p(X, Z \mid \theta)$, the EM algorithm seeks to find the MLE with:

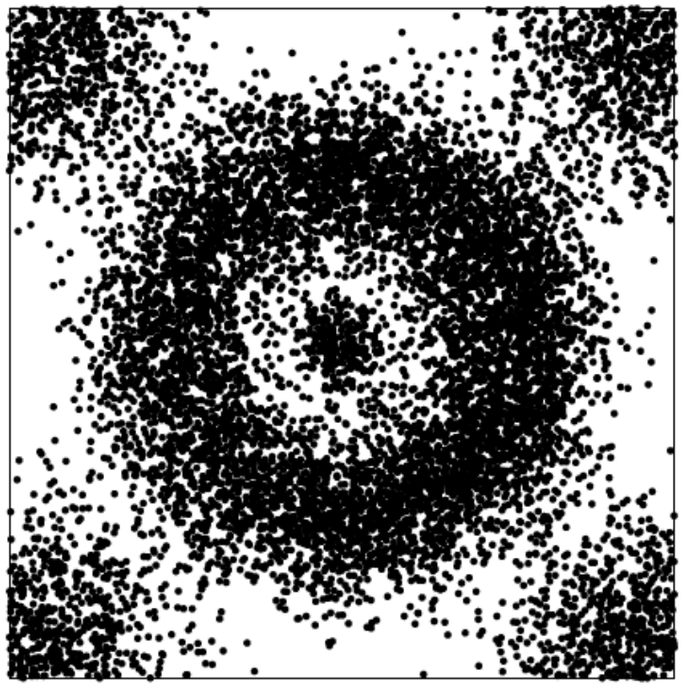
Expectation step (E step): compute $Q(\theta \mid \theta^{(t)}) = E_{Z|X,\theta^{(t)}}\left[\log L(\theta; X, Z)\right]$

Maximization step (M step): Find $\theta^{(t+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(t)})$

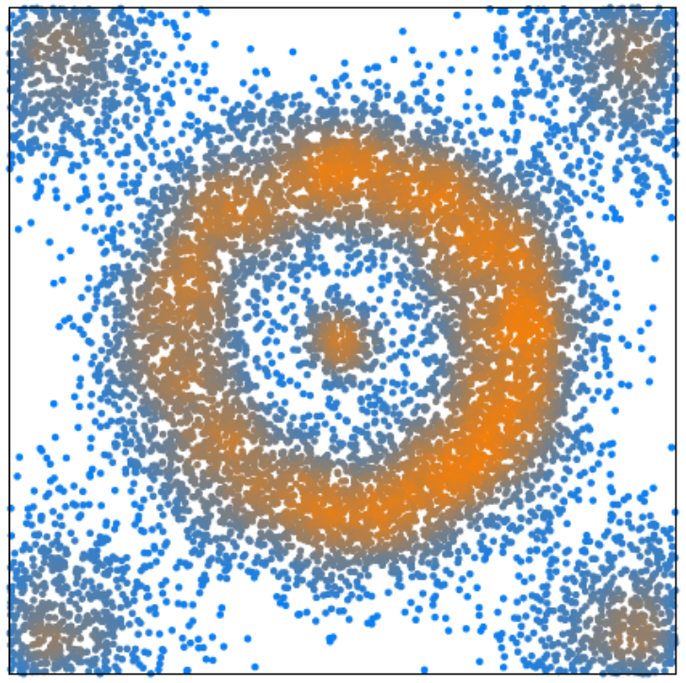
The Koonz, Narendra and Fukunaga algorithm (1976)



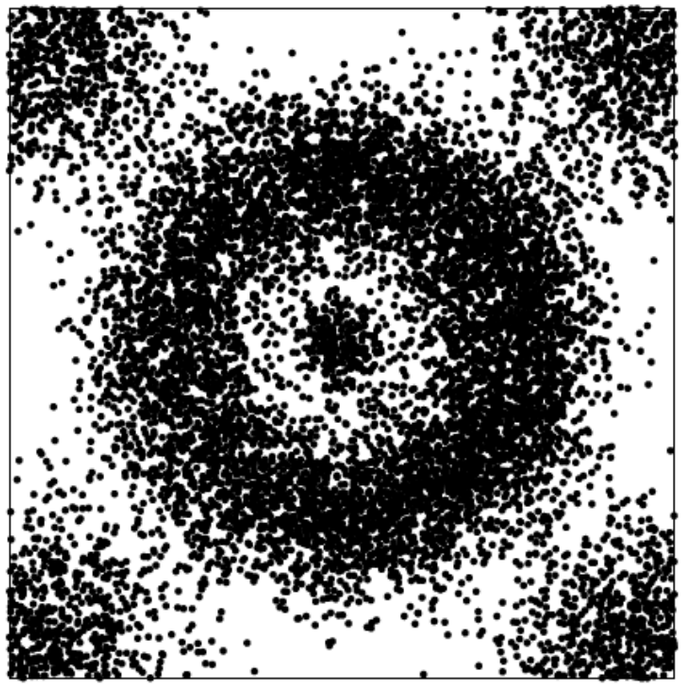
The Koonz, Narendra and Fukunaga algorithm (1976)



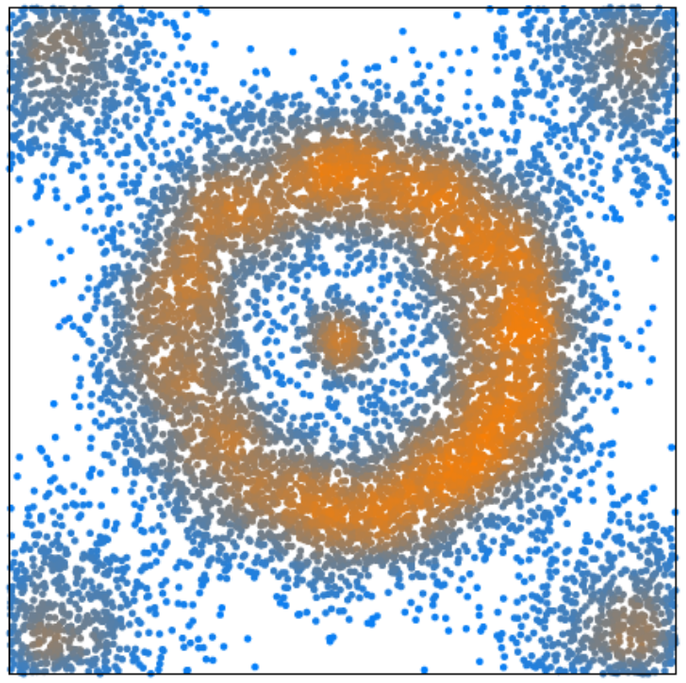
Density estimation



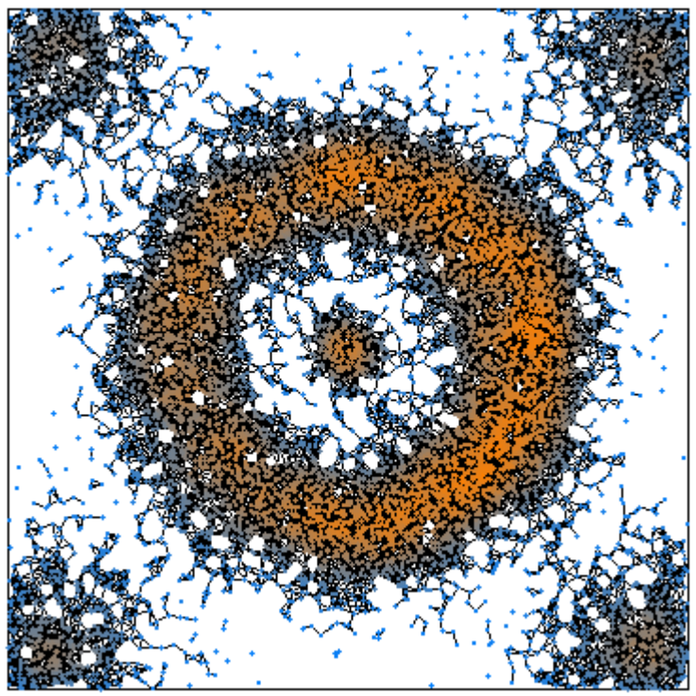
The Koonz, Narendra and Fukunaga algorithm (1976)



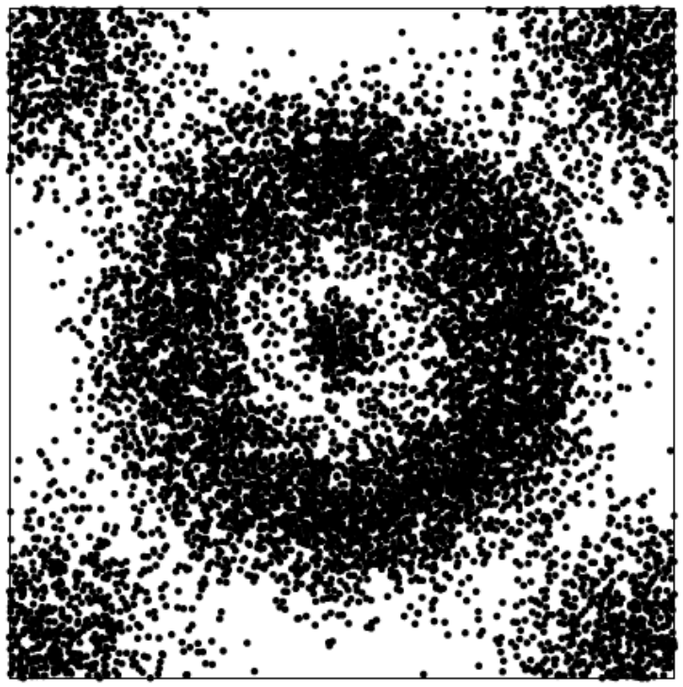
Density estimation



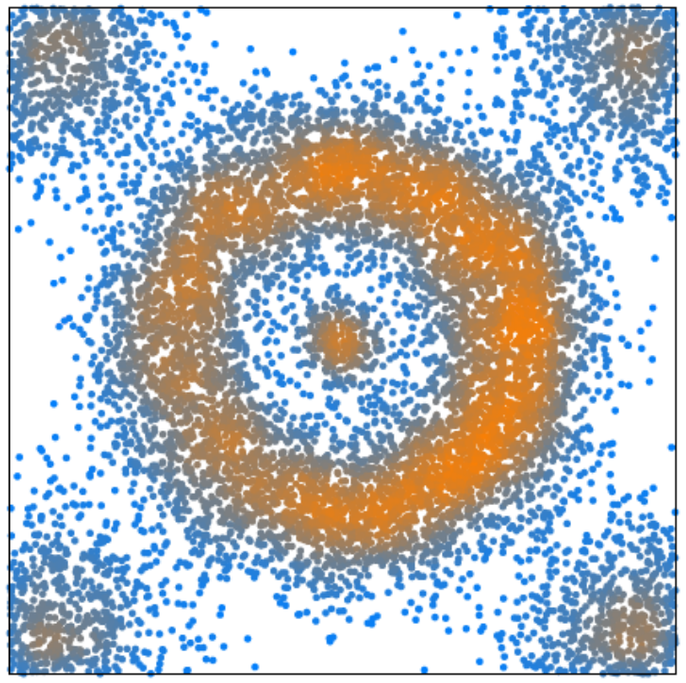
Neighborhood graph



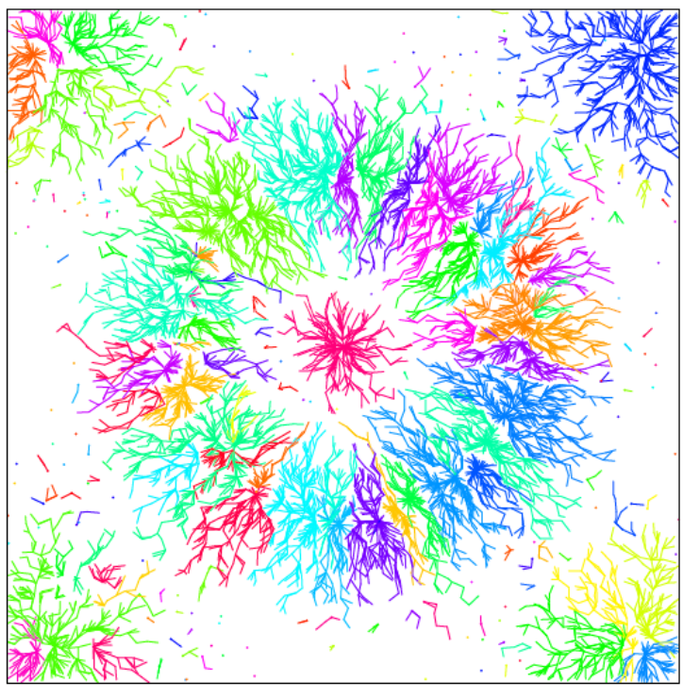
The Koonz, Narendra and Fukunaga algorithm (1976)



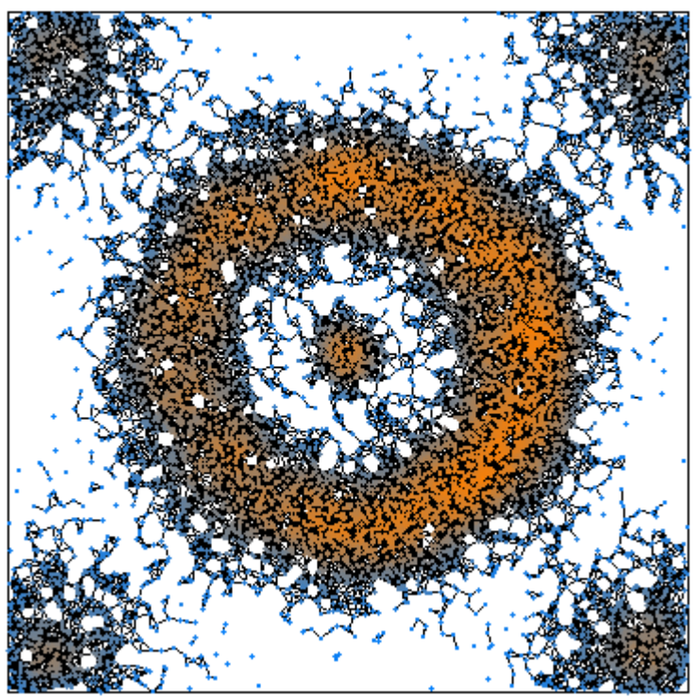
Density estimation



Neighborhood graph



Discrete approximation of the gradient; for each vertex v, a gradient edge is selected among the edges adjacent to v.



The Koonz, Narendra and Fukunaga algorithm (1976)

The algorithm:

Input: neighborhood graph G with n vertices (the data points) and a n-dimensional vector $\hat{f}$ (density estimate)

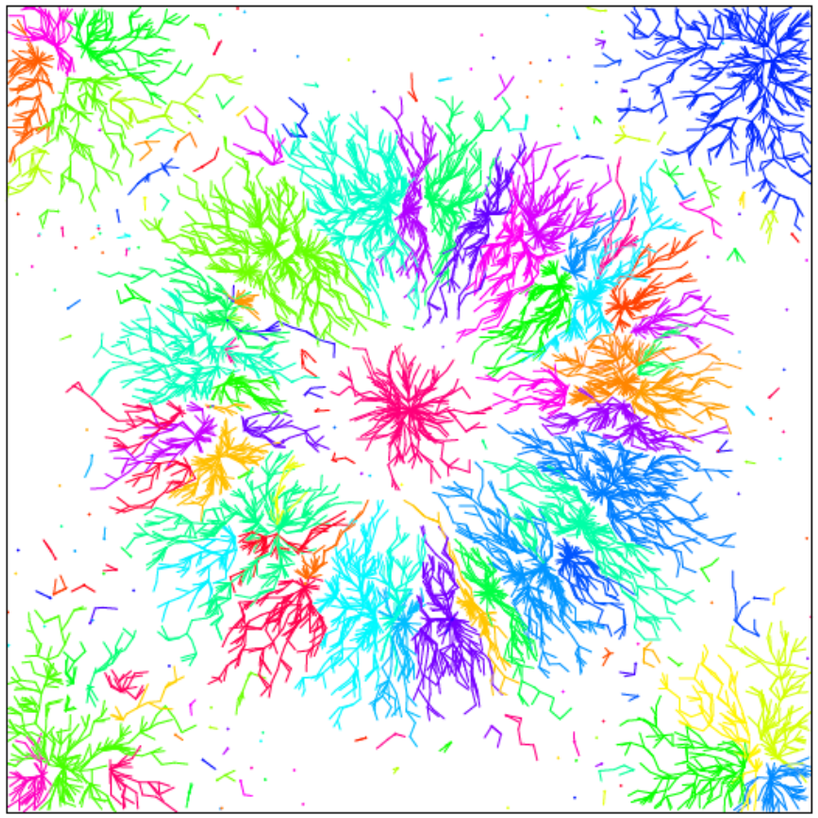
Sort the vertex indices $\{1, 2, \ldots, n\}$ in decreasing order: $\hat{f}(1) \geq \hat{f}(2) \geq \cdots \geq \hat{f}(n)$;
Initialize a union-find data structure (disjoint-set forest) $\mathcal{U}$ and two vectors g, r of size n;

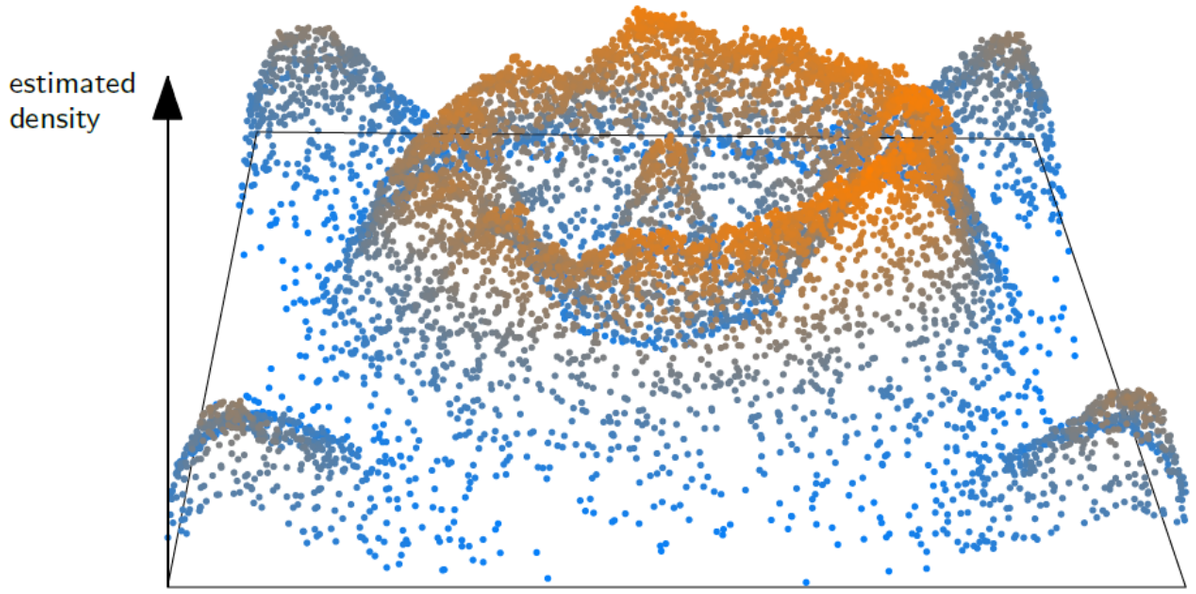
for $i = 1$ to n **do**
    Let N be the set of neighbors of i in G that have indices higher than i;
    if $N = \emptyset$
        Create a new entry e in $\mathcal{U}$ and attach vertex i to it: $\mathcal{U}$.`MakeSet`(i);
        $r(e) \leftarrow i$ *// $r(e)$ stores the root vertex associated with the entry e*
    else
        $g(i) \leftarrow \text{argmax}_{j \in \mathcal{N}} \hat{f}(j)$ *// $g(i)$ stores the approximate gradient at vertex i*
        $e_i \leftarrow \mathcal{U}$.`Find`$(g(i))$;
        Attach vertex i to the entry e_i: $\mathcal{U}$.`Union`(i, e_i);

Output: the collection of entries e in $\mathcal{U}$

The Koonz, Narendra and Fukunaga algorithm (1976)

Drawbacks:

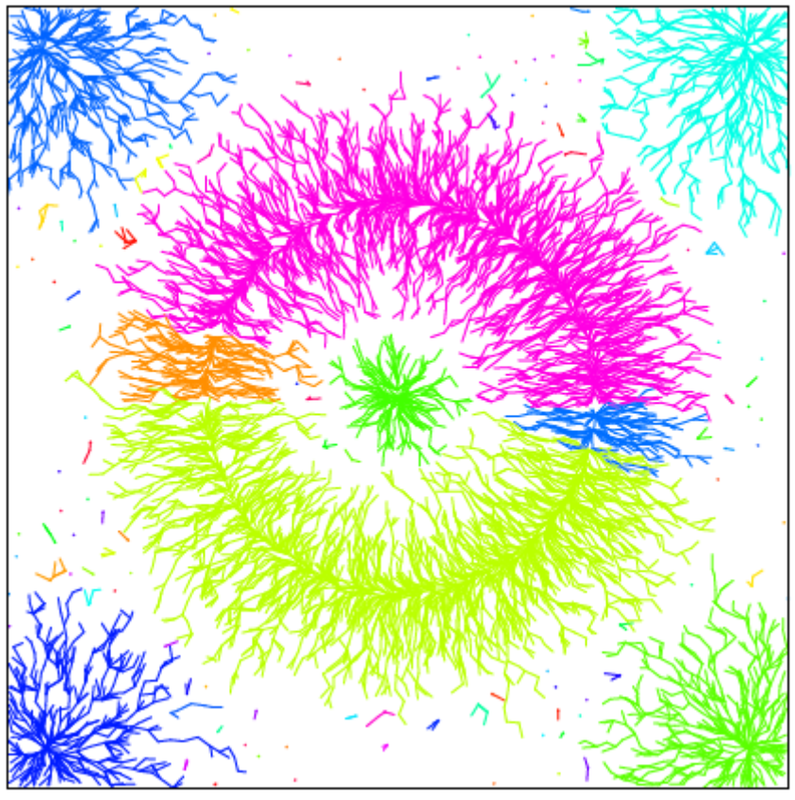


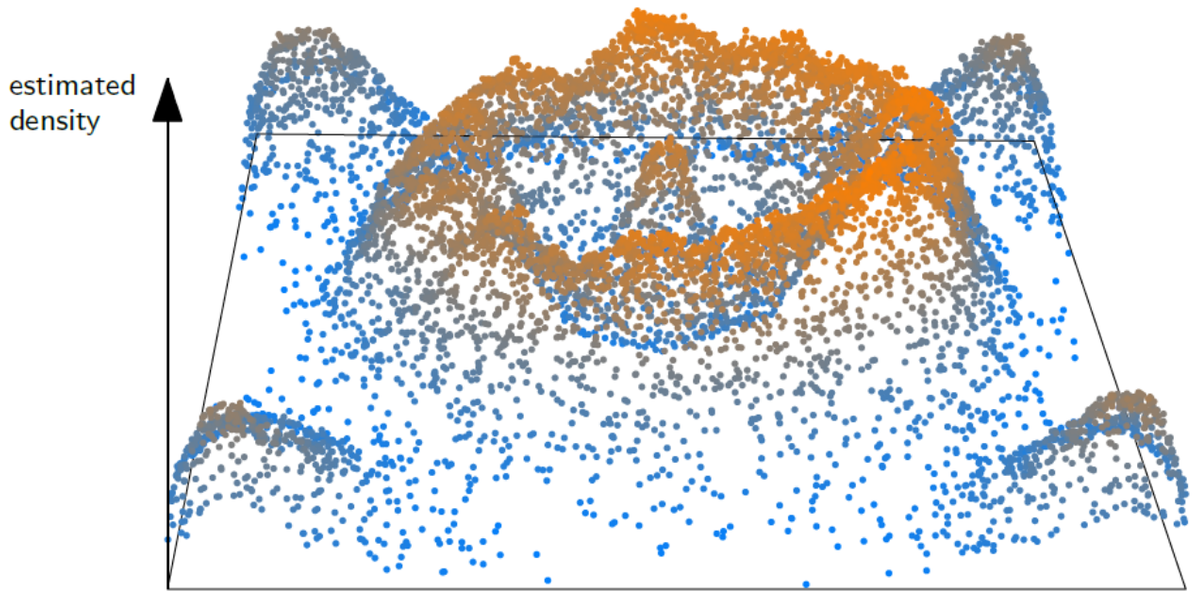


- As many clusters as local maxima of the density estimate $\rightarrow$ sensitivity to noise!

The Koonz, Narendra and Fukunaga algorithm (1976)

Drawbacks:





- As many clusters as local maxima of the density estimate $\rightarrow$ sensitivity to noise!
- The choice of the neighborhood graph may result in wide changes in the output.

The Koonz, Narendra and Fukunaga algorithm (1976)

Drawbacks:

- As many clusters as local maxima of the density estimate $\rightarrow$ sensitivity to noise!
- The choice of the neighborhood graph may result in wide changes in the output.

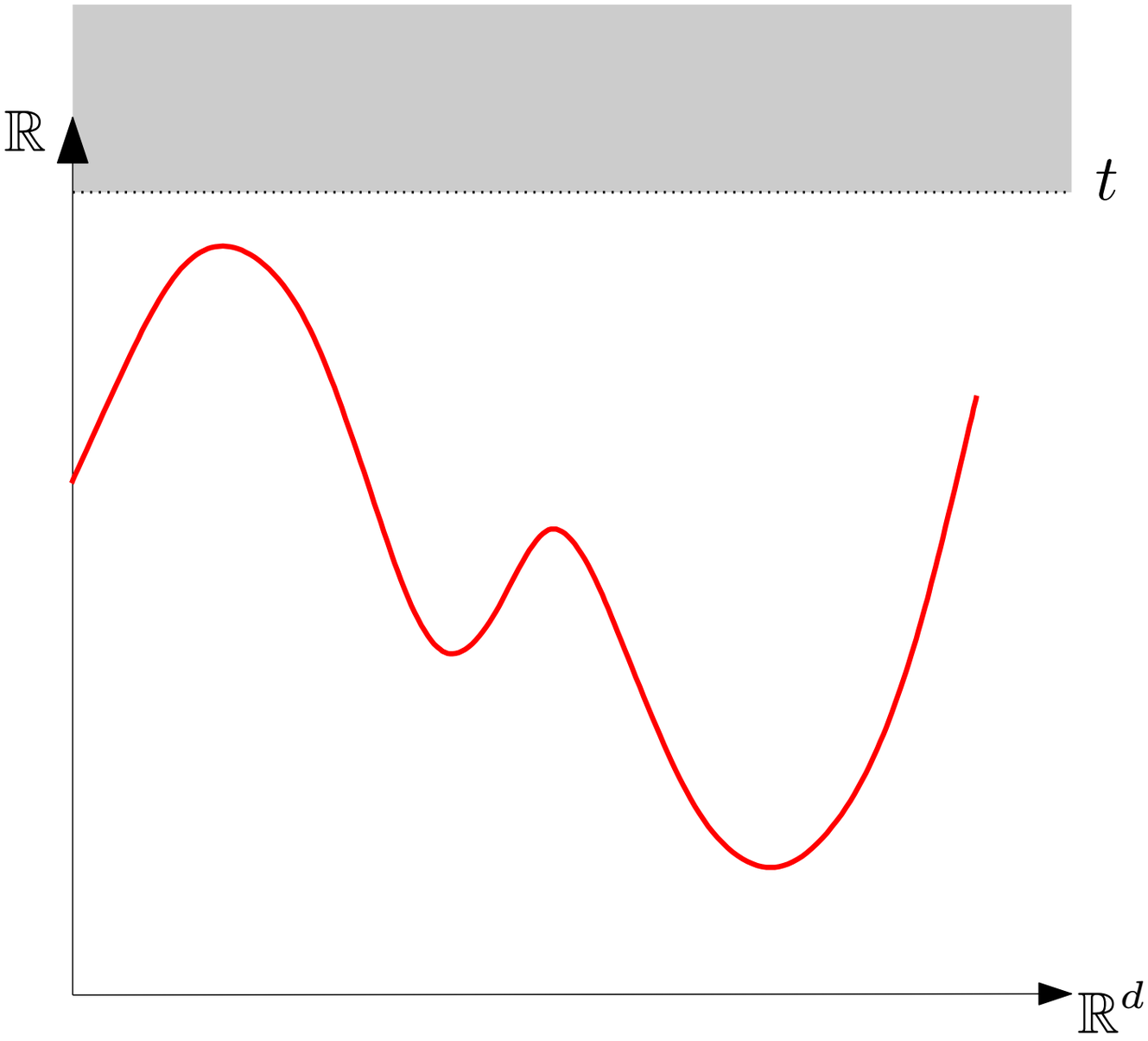
Approaches to overcome these issues:

- Smooth out the density estimate (e.g. mean-shift)... But how to choose the smoothing parameter?
- Merge clusters with 0-dimensional persistent homology!

0-dimensional PH of density

Given a probability density f:

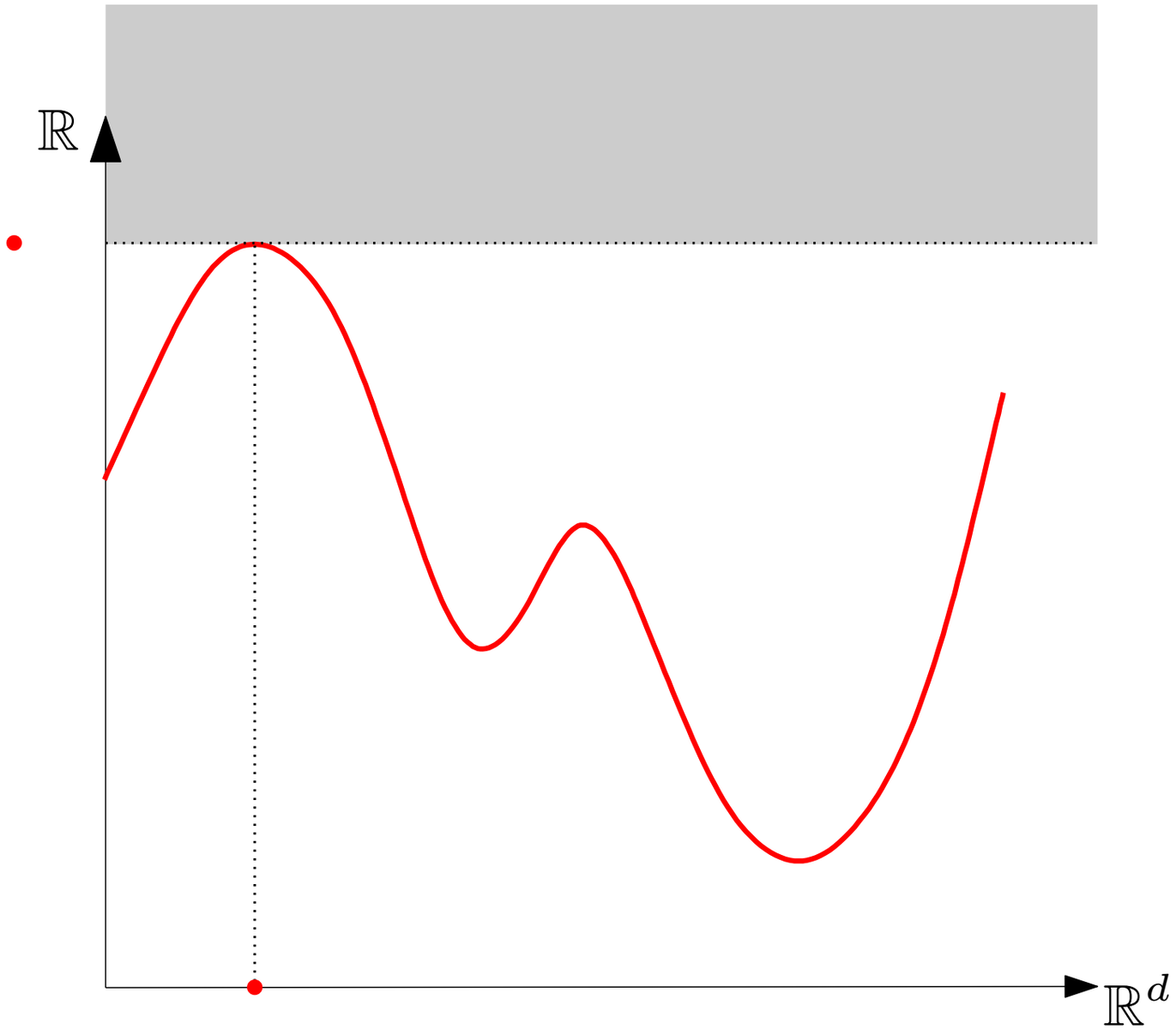
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

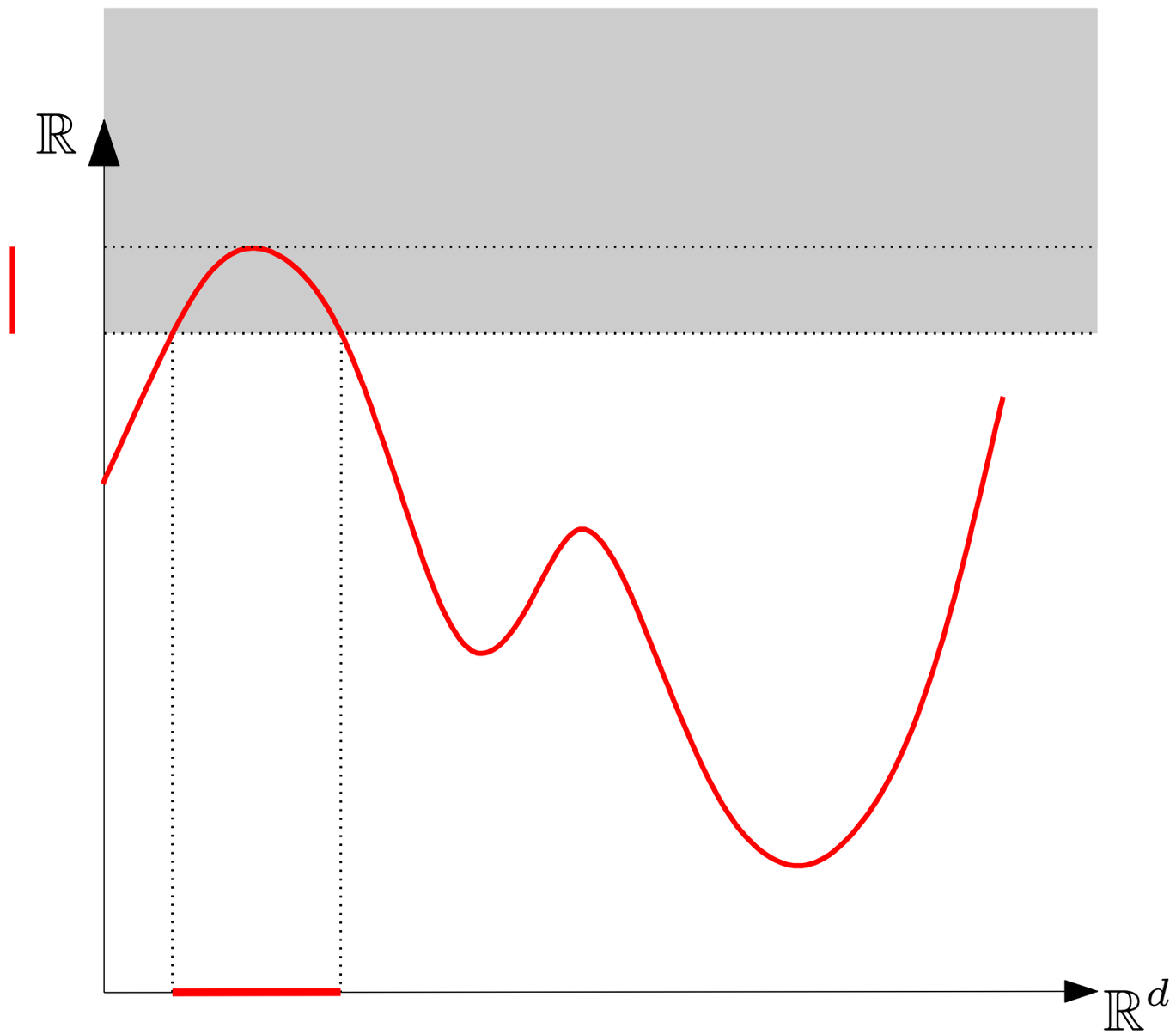
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

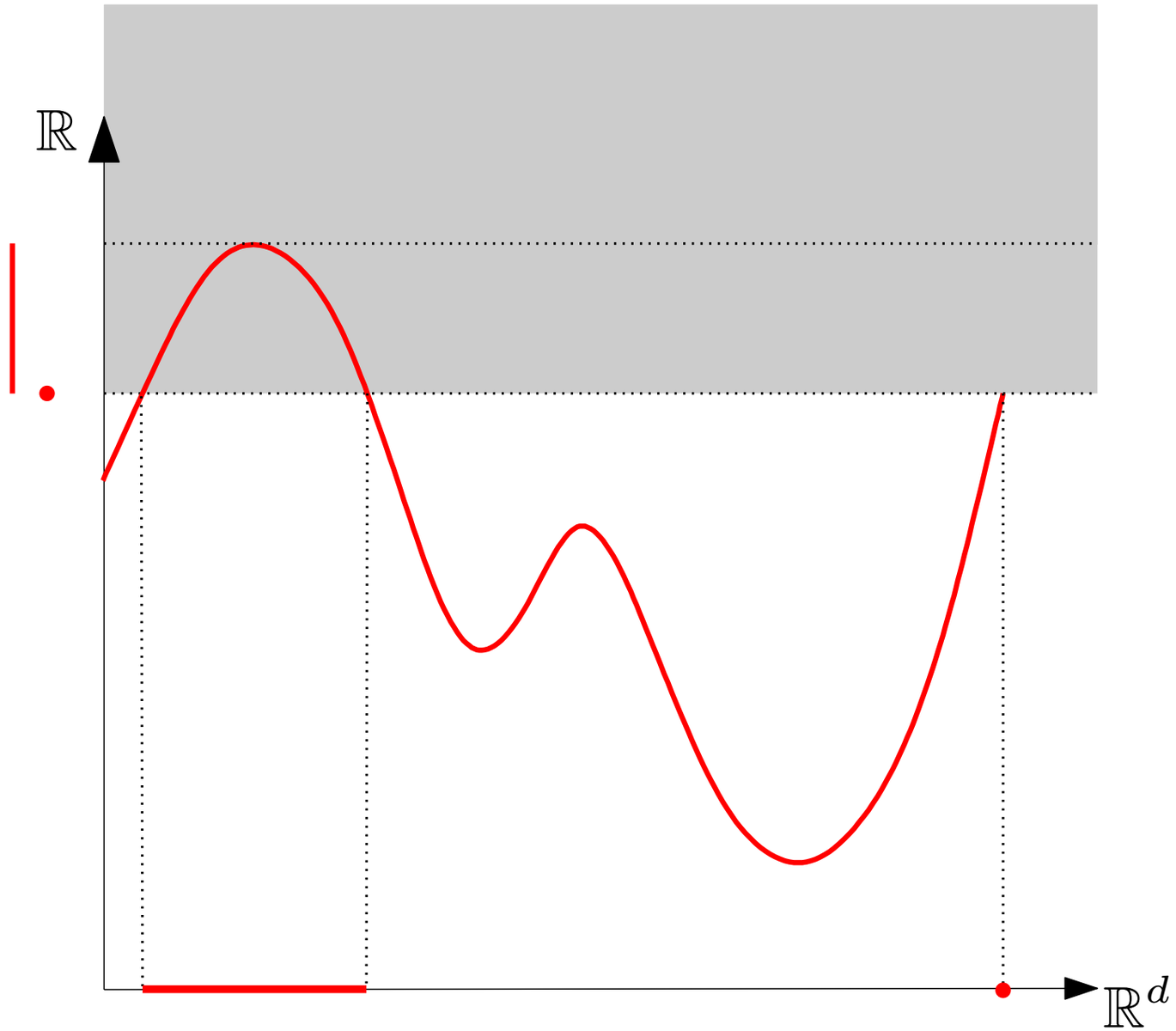
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

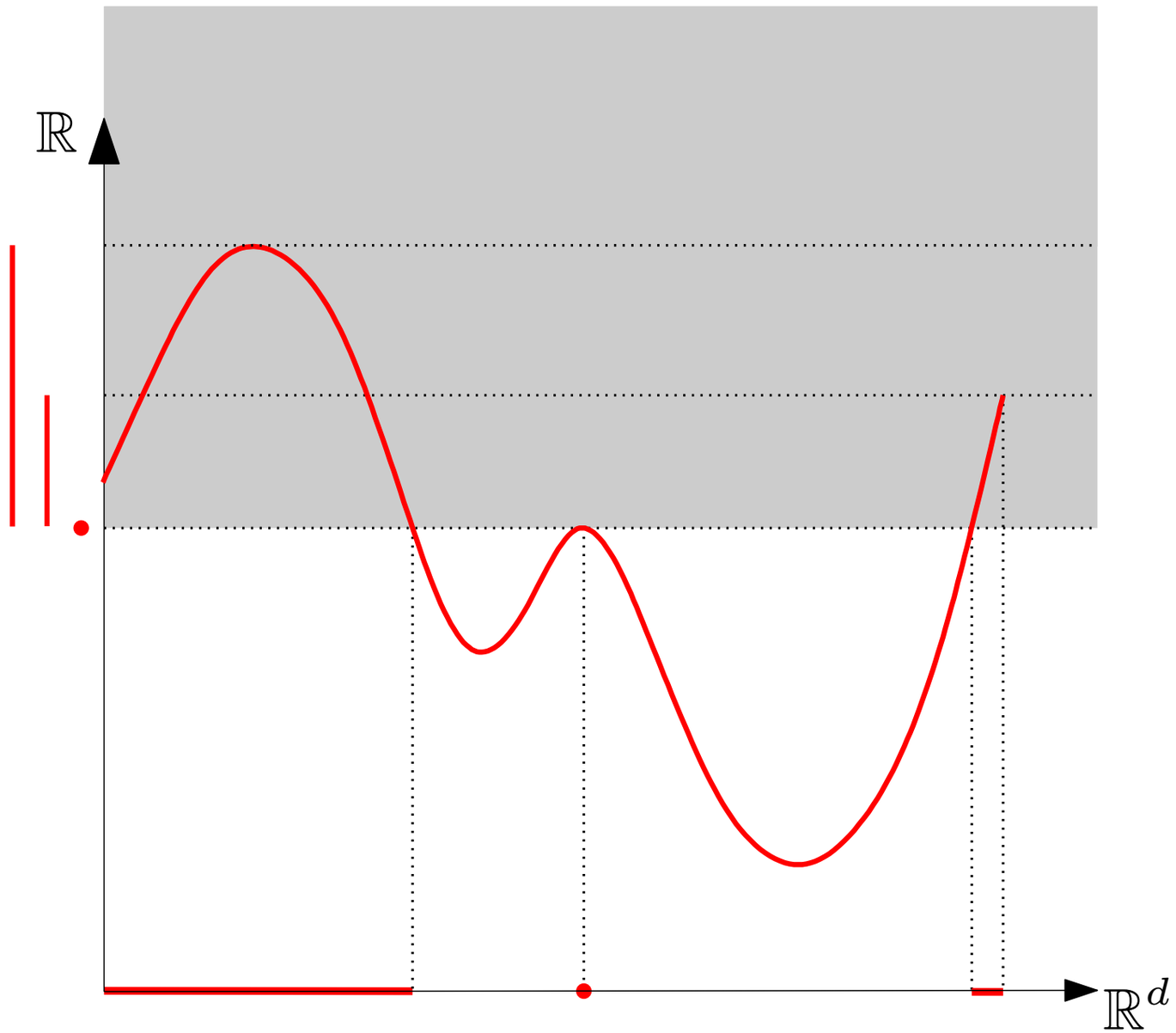
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

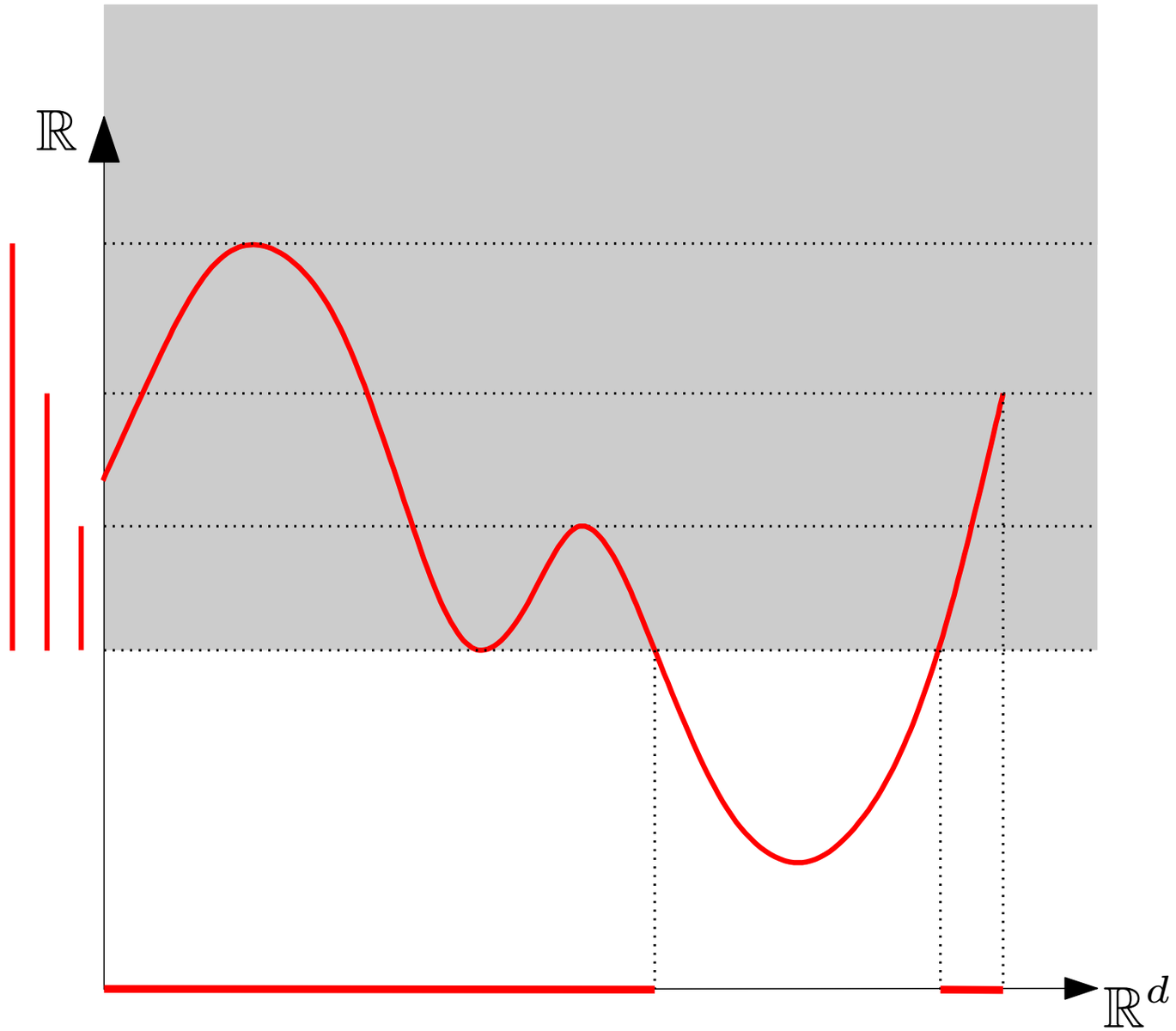
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

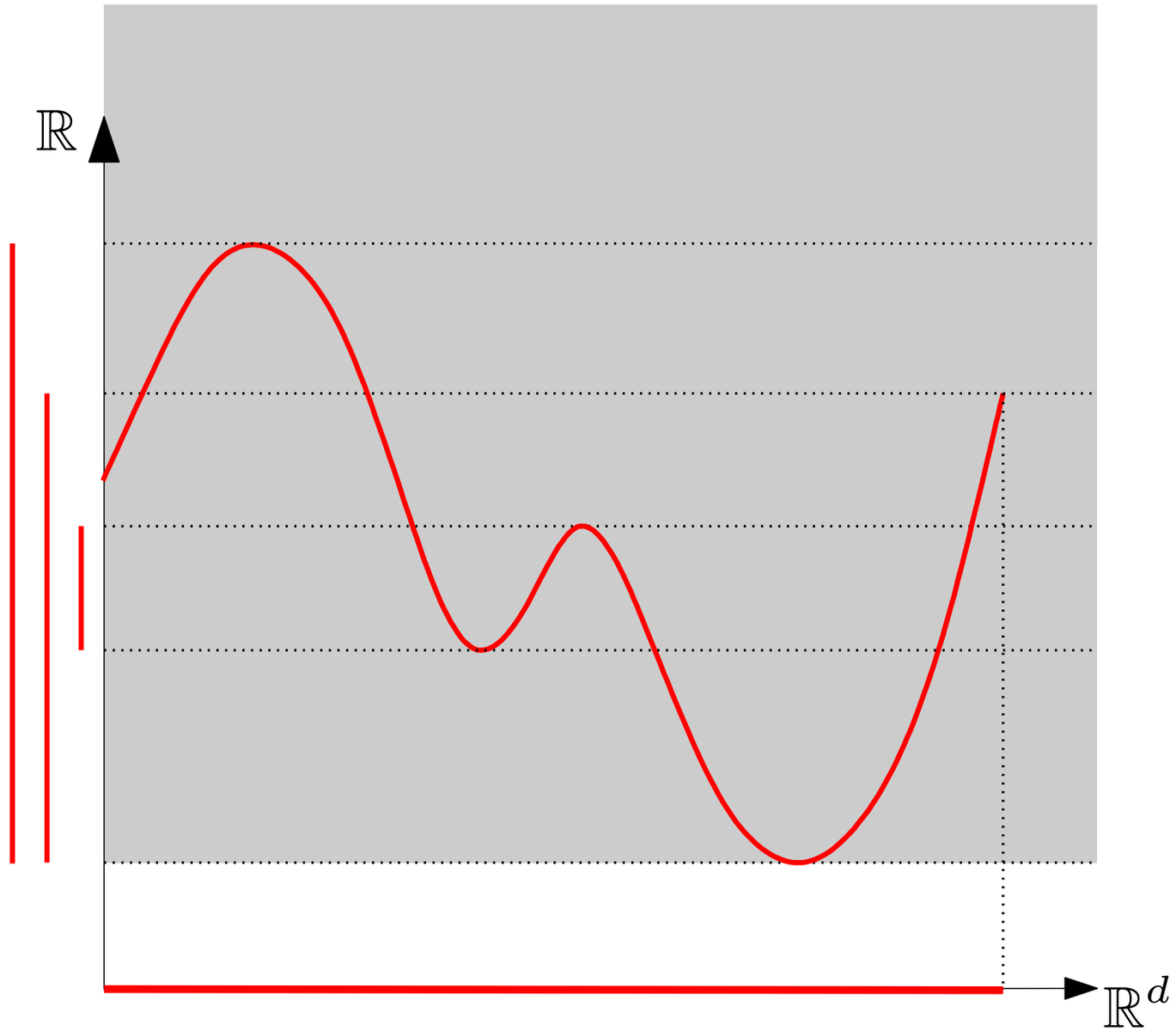
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

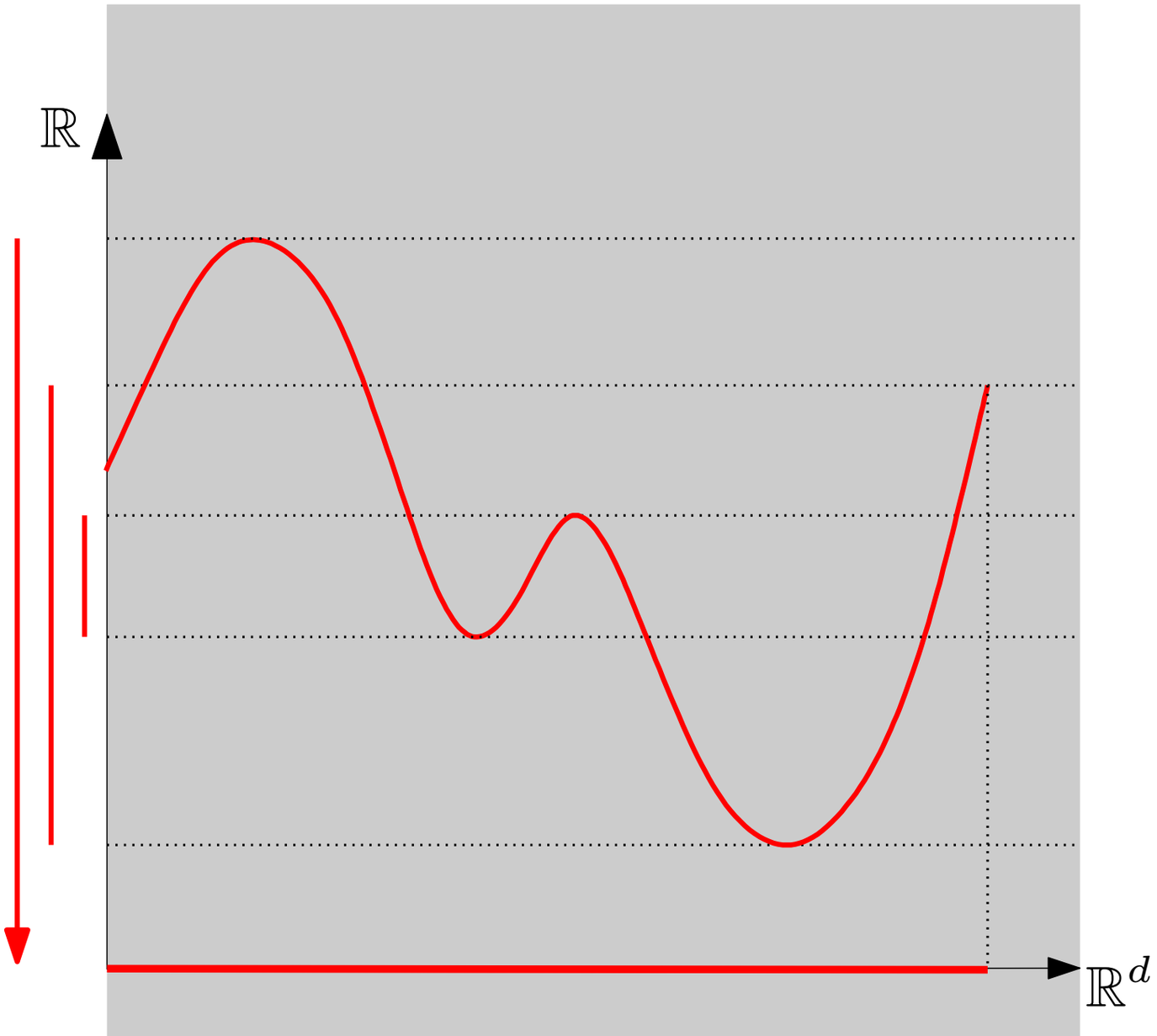
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

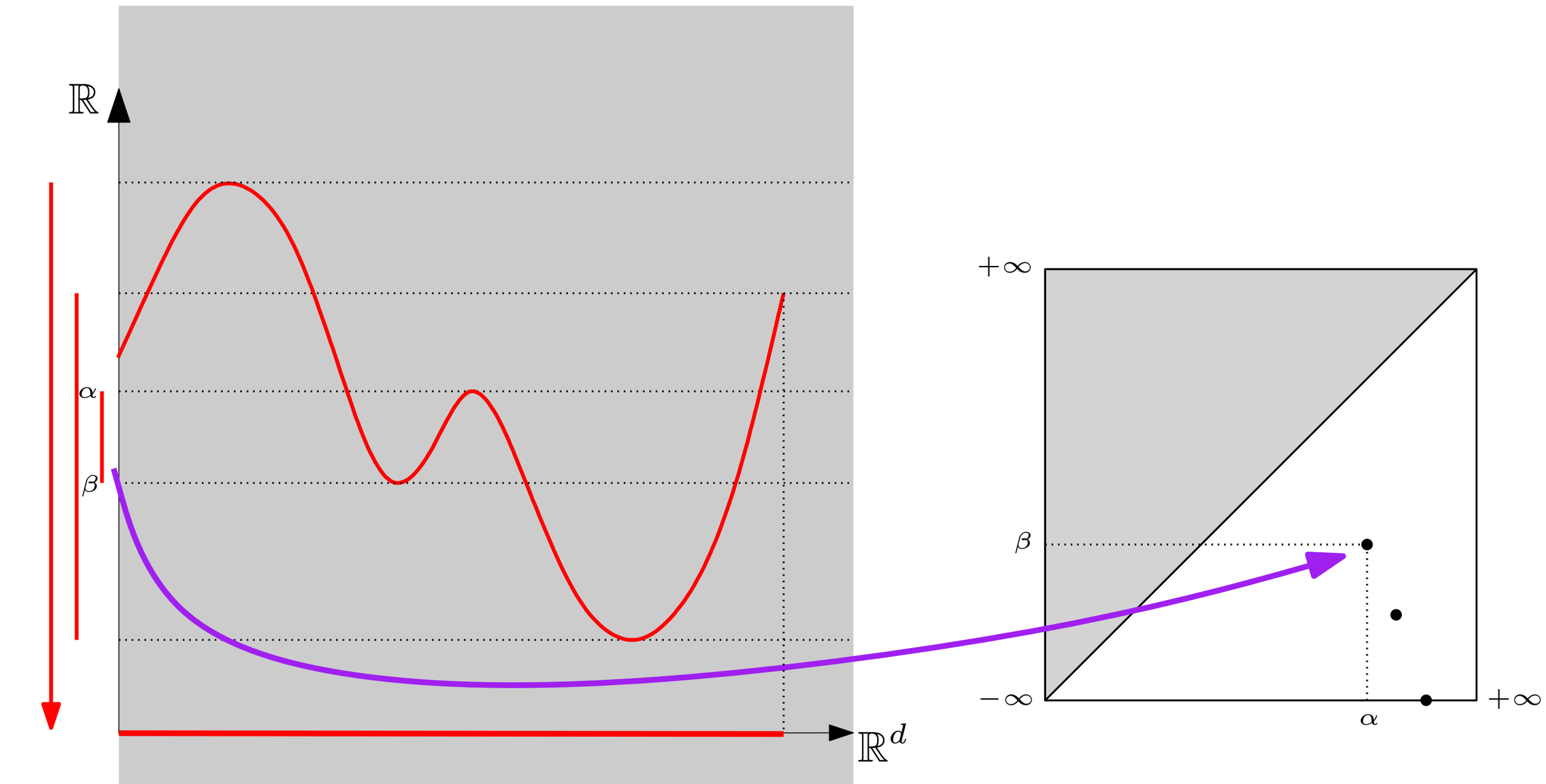
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way



0-dimensional PH of density

Given a probability density f:

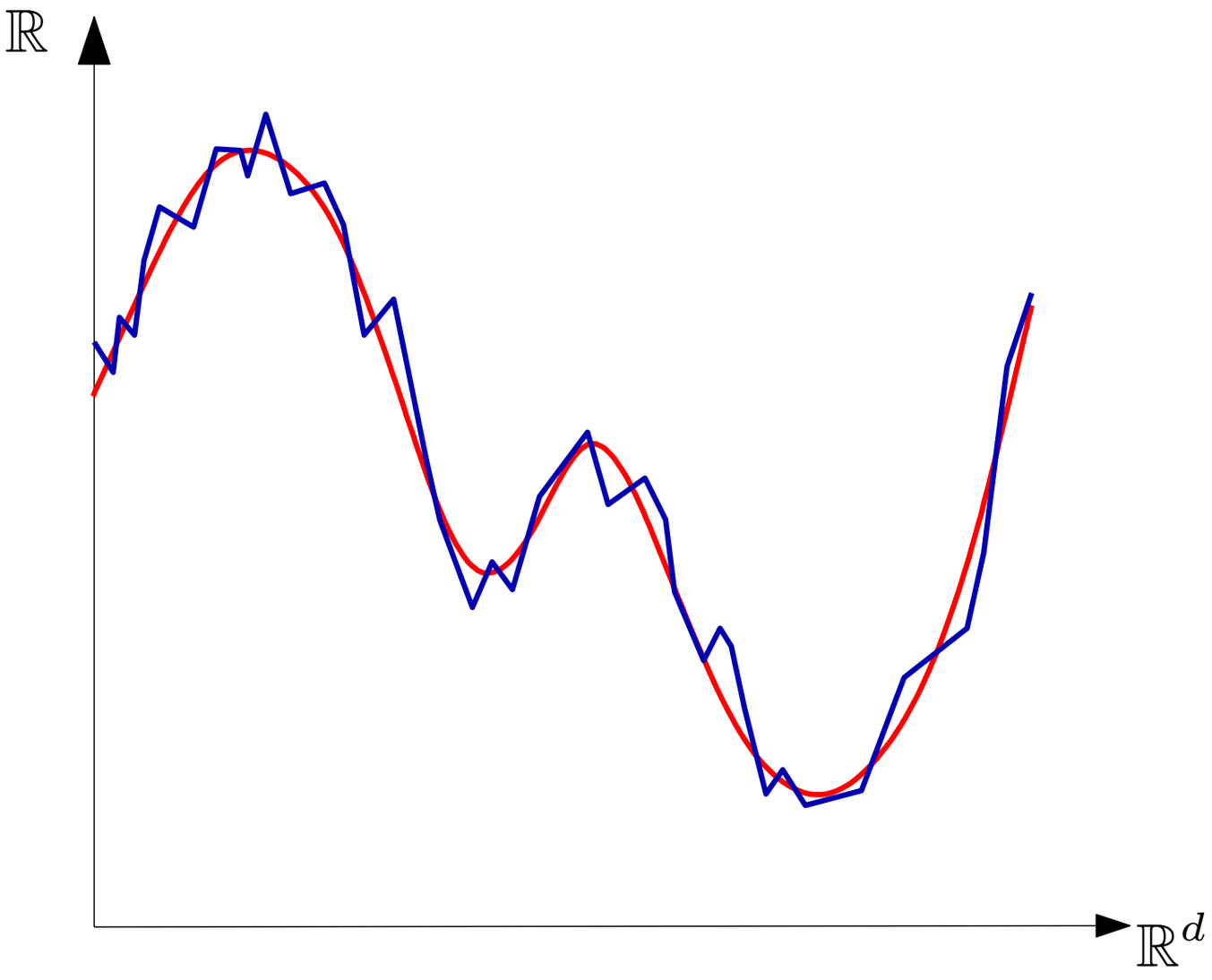
- Consider the superlevel-sets filtration $f^{-1}([t, +\infty))$ for t from $+\infty$ to $-\infty$, instead of the sublevel-sets filtration.
- Persistence is defined in the same way

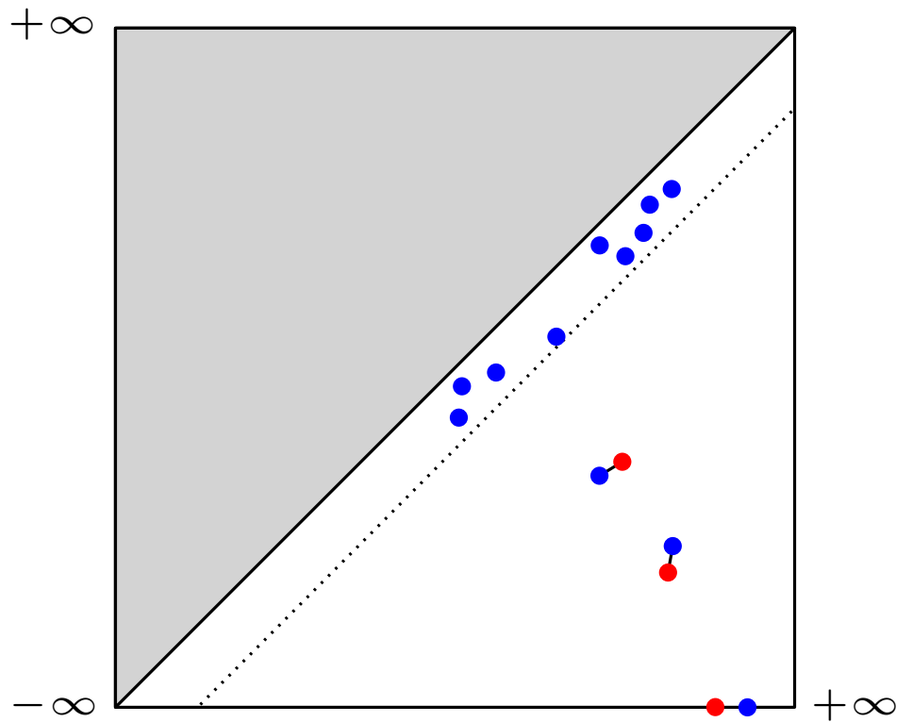


0-dimensional PH of density

Given an estimator $\hat{f}$:

Stability theorem $\Rightarrow d_b(D_f, D_{\hat{f}}) \leq \|f - \hat{f}\|_\infty$.



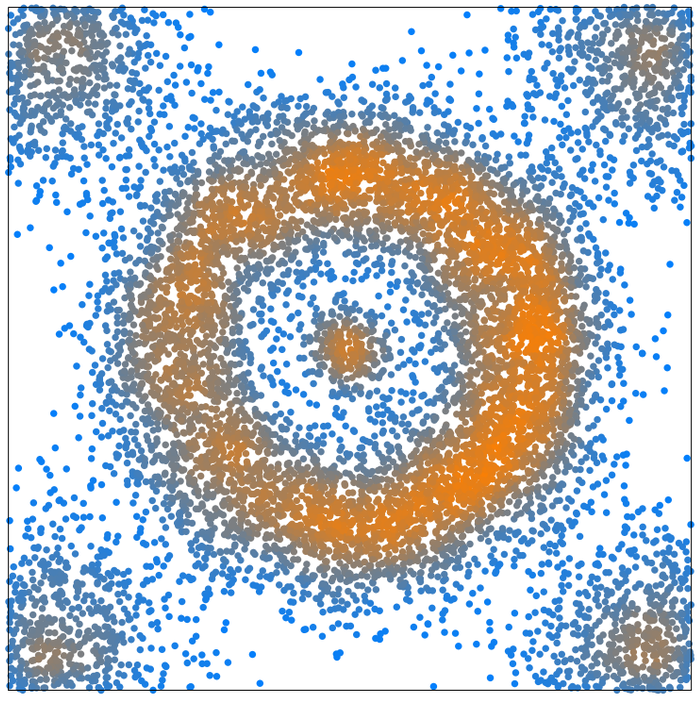


Persistence-based clustering

[*Persistence-Based Clustering in Riemannian Manifolds*, Chazal, Oudot, Skraba, Guibas, J. ACM, 2013]

- Density estimator $\hat{f}$ defines an order on the point cloud

  (sort data points by **decreasing** estimated density values)



Persistence-based clustering

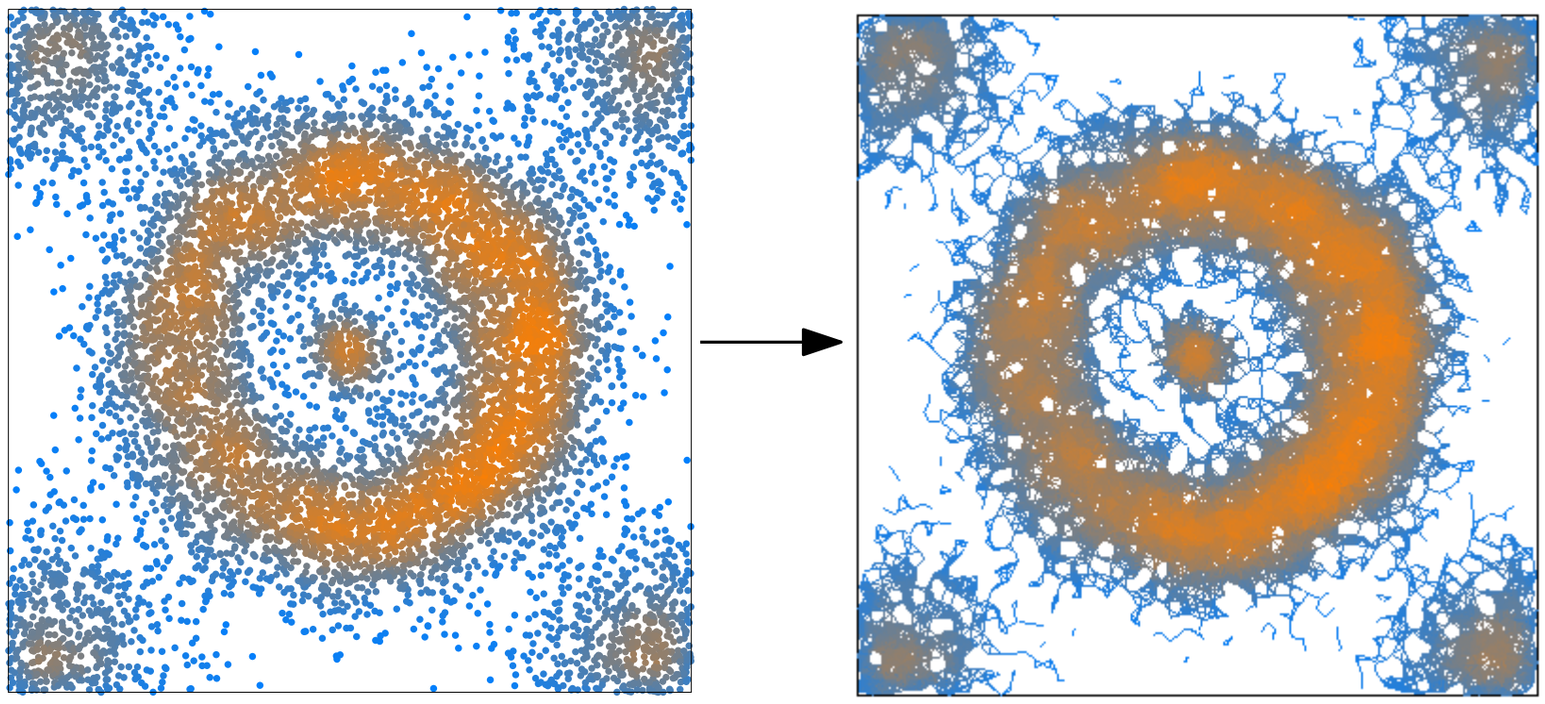
[*Persistence-Based Clustering in Riemannian Manifolds*, Chazal, Oudot, Skraba, Guibas, J. ACM, 2013]

- Density estimator $\hat{f}$ defines an order on the point cloud

  (sort data points by **decreasing** estimated density values)

- Extend order to the graph edges $\rightarrow$ *upper-star filtration*

  $(\hat{f}([u,v]) = \min\{\hat{f}(u),\ \hat{f}(v)\})$



Persistence-based clustering

[*Persistence-Based Clustering in Riemannian Manifolds*, Chazal, Oudot, Skraba, Guibas, J. ACM, 2013]

- Density estimator $\hat{f}$ defines an order on the point cloud

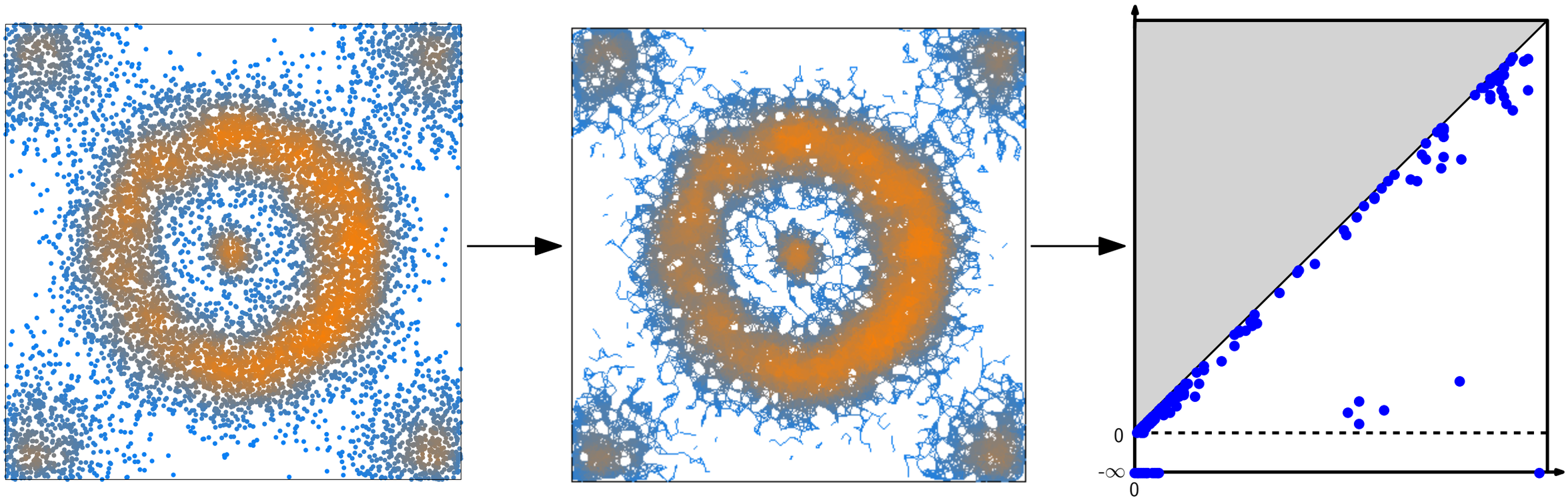
  (sort data points by **decreasing** estimated density values)

- Extend order to the graph edges $\rightarrow$ *upper-star filtration*

  $(\hat{f}([u,v]) = \min\{\hat{f}(u),\ \hat{f}(v)\})$

- Compute the 0-dimensional persistence diagram of this filtration

  (apply 0-dimensional persistence algorithm $\rightarrow$ union-find data structure)



Estimating the correct number of clusters

- Density estimator $\hat{f}$ defines an order on the point cloud

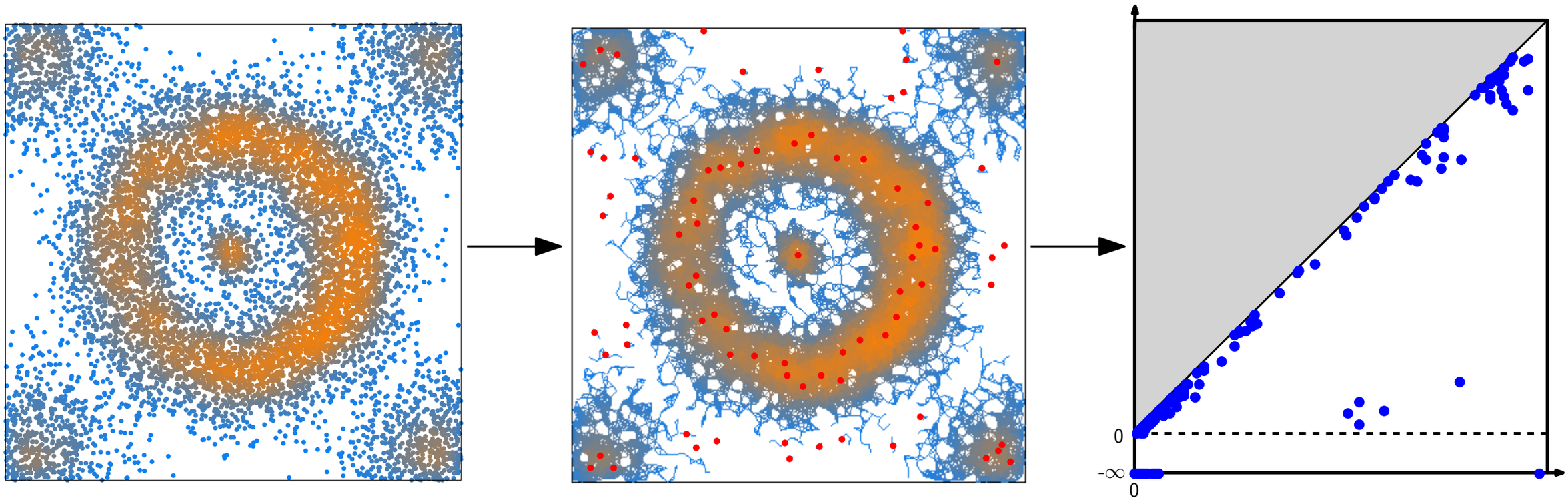
  (sort data points by **decreasing** estimated density values)

- Extend order to the graph edges $\rightarrow$ *upper-star filtration*

  $(\hat{f}([u,v]) = \min\{\hat{f}(u),\ \hat{f}(v)\})$

- Compute the 0-dimensional persistence diagram of this filtration

  (apply 0-dimensional persistence algorithm $\rightarrow$ union-find data structure)



Estimating the correct number of clusters

- Density estimator $\hat{f}$ defines an order on the point cloud

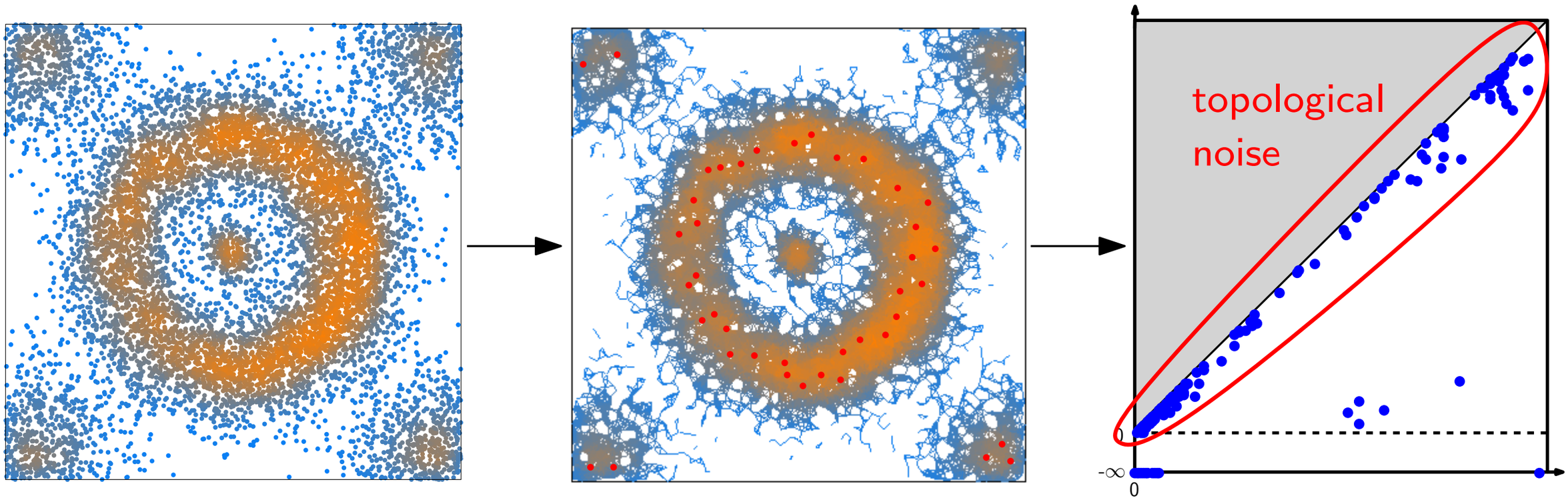
  (sort data points by **decreasing** estimated density values)

- Extend order to the graph edges $\rightarrow$ *upper-star filtration*

  $(\hat{f}([u,v]) = \min\{\hat{f}(u),\ \hat{f}(v)\})$

- Compute the 0-dimensional persistence diagram of this filtration

  (apply 0-dimensional persistence algorithm $\rightarrow$ union-find data structure)



Estimating the correct number of clusters

- Density estimator $\hat{f}$ defines an order on the point cloud

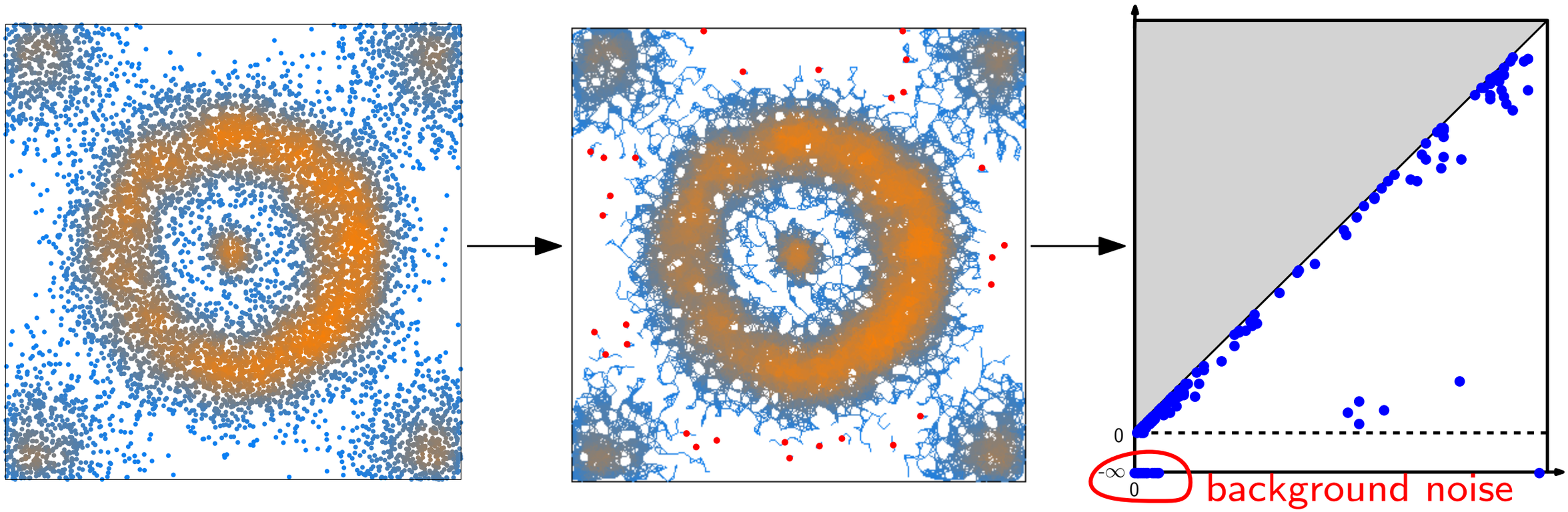
  (sort data points by **decreasing** estimated density values)

- Extend order to the graph edges $\rightarrow$ *upper-star filtration*

  $(\hat{f}([u,v]) = \min\{\hat{f}(u),\ \hat{f}(v)\})$

- Compute the 0-dimensional persistence diagram of this filtration

  (apply 0-dimensional persistence algorithm $\rightarrow$ union-find data structure)



Estimating the correct number of clusters

- Density estimator $\hat{f}$ defines an order on the point cloud

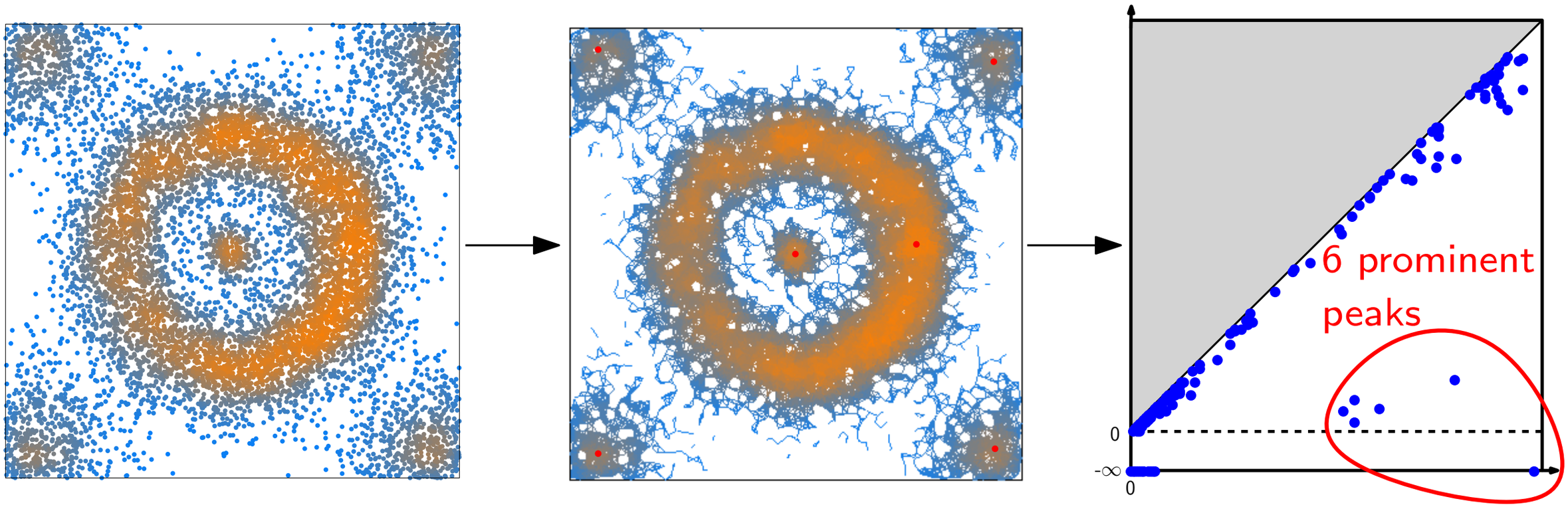
  (sort data points by **decreasing** estimated density values)

- Extend order to the graph edges $\rightarrow$ *upper-star filtration*

  $(\hat{f}([u,v]) = \min\{\hat{f}(u),\ \hat{f}(v)\})$

- Compute the 0-dimensional persistence diagram of this filtration

  (apply 0-dimensional persistence algorithm $\rightarrow$ union-find data structure)



Estimating the correct number of clusters

- Density estimator $\hat{f}$ defines an order on the point cloud

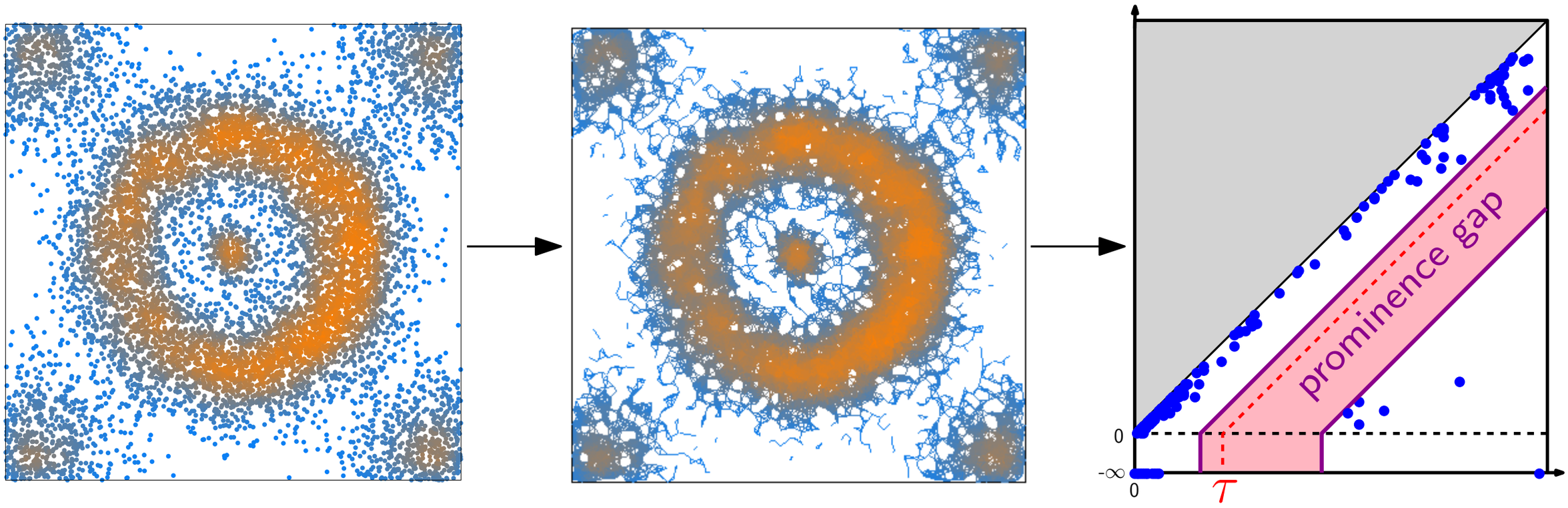
  (sort data points by **decreasing** estimated density values)

- Extend order to the graph edges $\rightarrow$ *upper-star filtration*

  $(\hat{f}([u,v]) = \min\{\hat{f}(u),\ \hat{f}(v)\})$

- Compute the 0-dimensional persistence diagram of this filtration

  (apply 0-dimensional persistence algorithm $\rightarrow$ union-find data structure)



Estimating the correct number of clusters

Hypotheses:

- $f : \mathbb{R}^d \to \mathbb{R}$ a c-Lipschitz probability density function,
- $P \subset \mathbb{R}^d$ a finite set of n points sampled i.i.d. according to f,
- $\hat{f} : P \to \mathbb{R}$ a density estimator such that $\eta := \max_{p \in P} |\hat{f}(p) - f(p)| < \Pi/5$,
- $G = (P, E)$ the δ-neighborhood graph for some positive $\delta < \frac{\Pi - 5\eta}{5c}$.

Note: Π is the prominence of the least prominent peak of f

Estimating the correct number of clusters

Hypotheses:

- $f : \mathbb{R}^d \to \mathbb{R}$ a c-Lipschitz probability density function,
- $P \subset \mathbb{R}^d$ a finite set of n points sampled i.i.d. according to f,
- $\hat{f} : P \to \mathbb{R}$ a density estimator such that $\eta := \max_{p \in P} |\hat{f}(p) - f(p)| < \Pi/5$,
- $G = (P, E)$ the δ-neighborhood graph for some positive $\delta < \frac{\Pi - 5\eta}{5c}$.

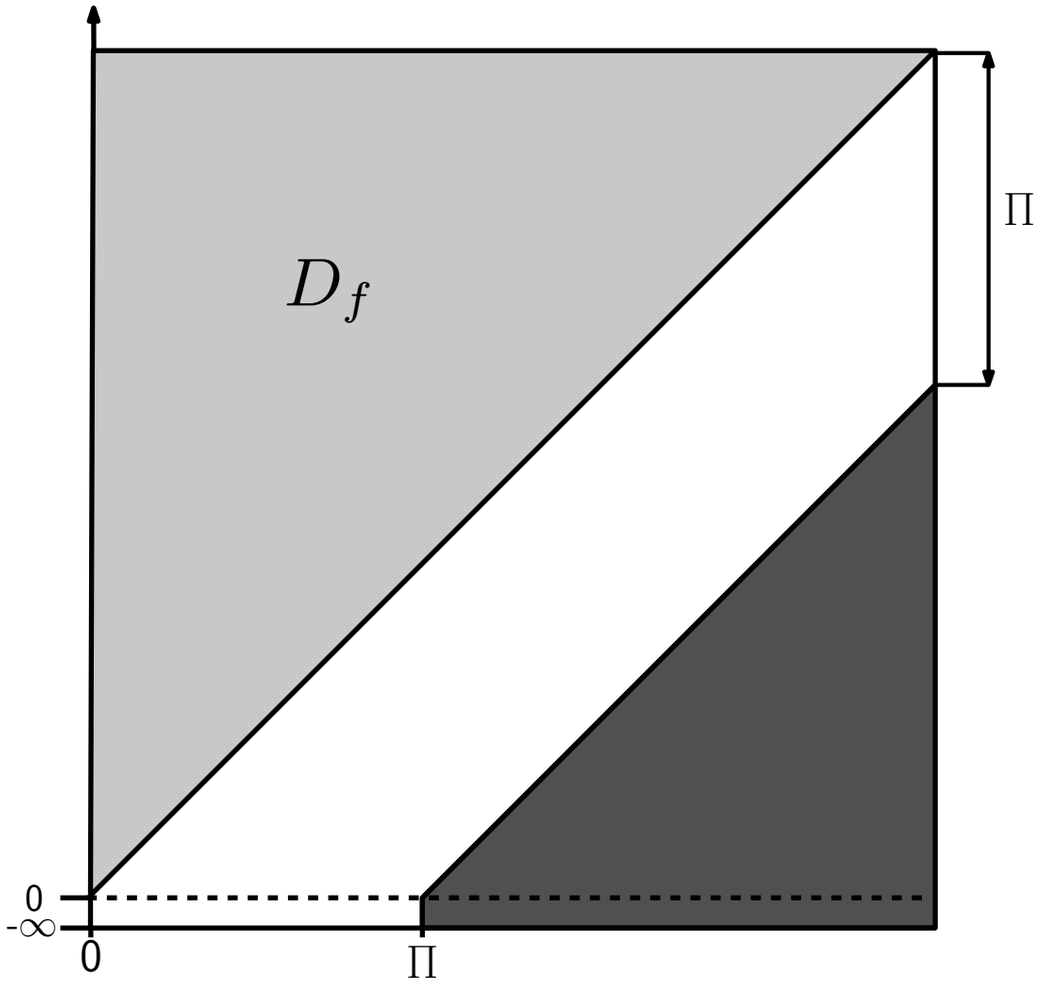
Note: Π is the prominence of the least prominent peak of f

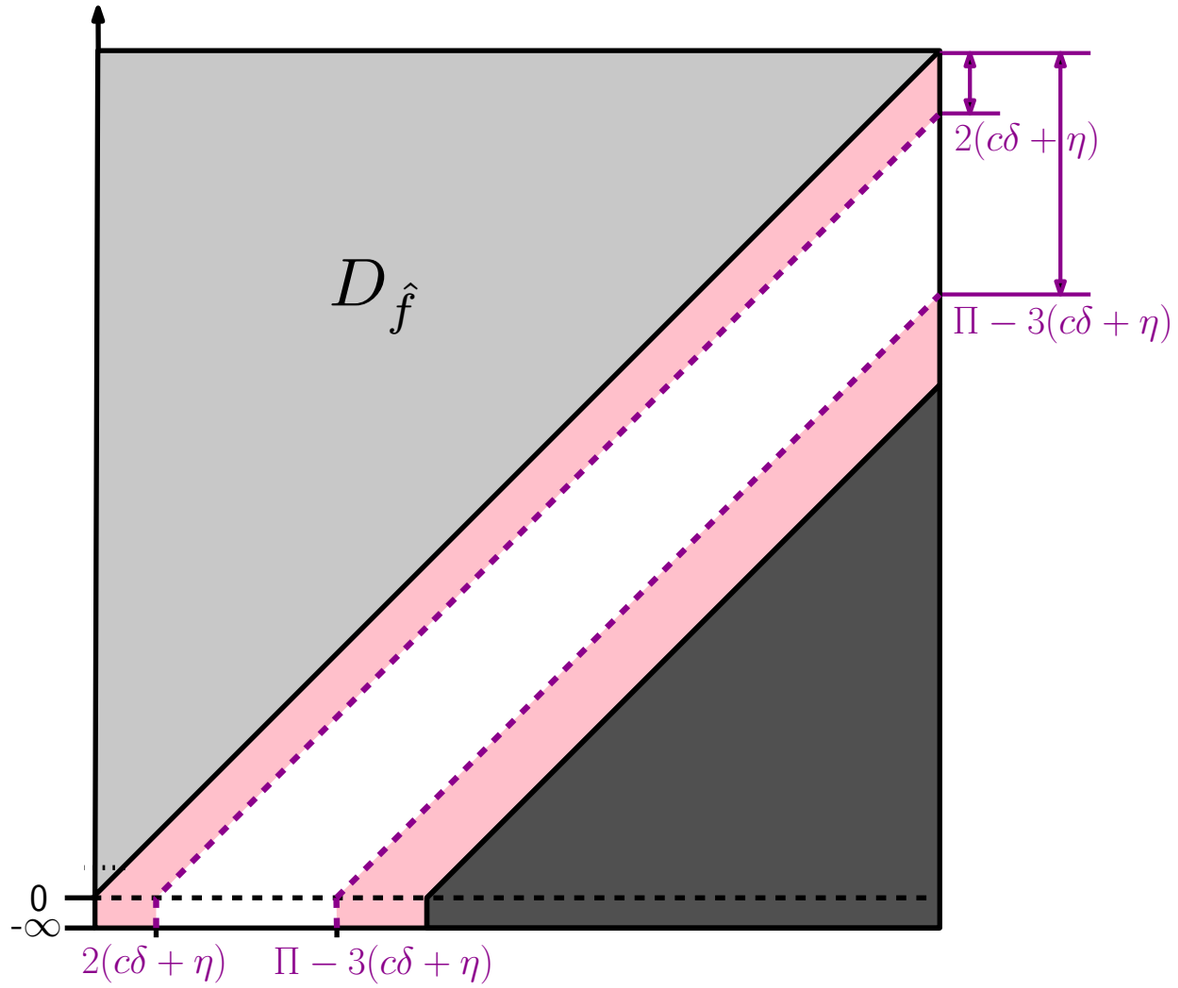
Conclusion:

For any choice of τ such that $2(c\delta + \eta) < \tau < \Pi - 3(c\delta + \eta)$, the number of clusters computed by the algorithm is equal to the number of peaks of f with probability at least $1 - e^{-\Omega(n)}$.

(the Ω notation hides factors depending on c, δ)

Estimating the correct number of clusters



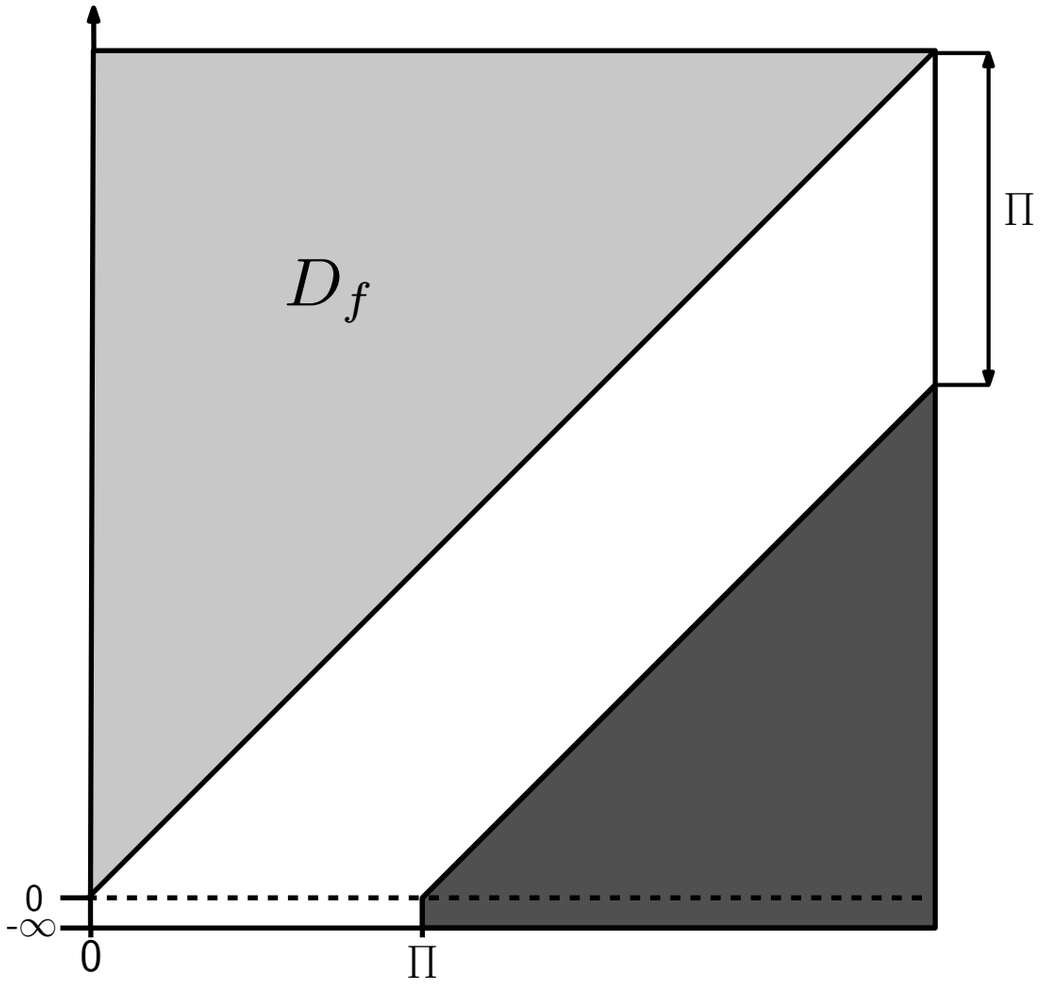


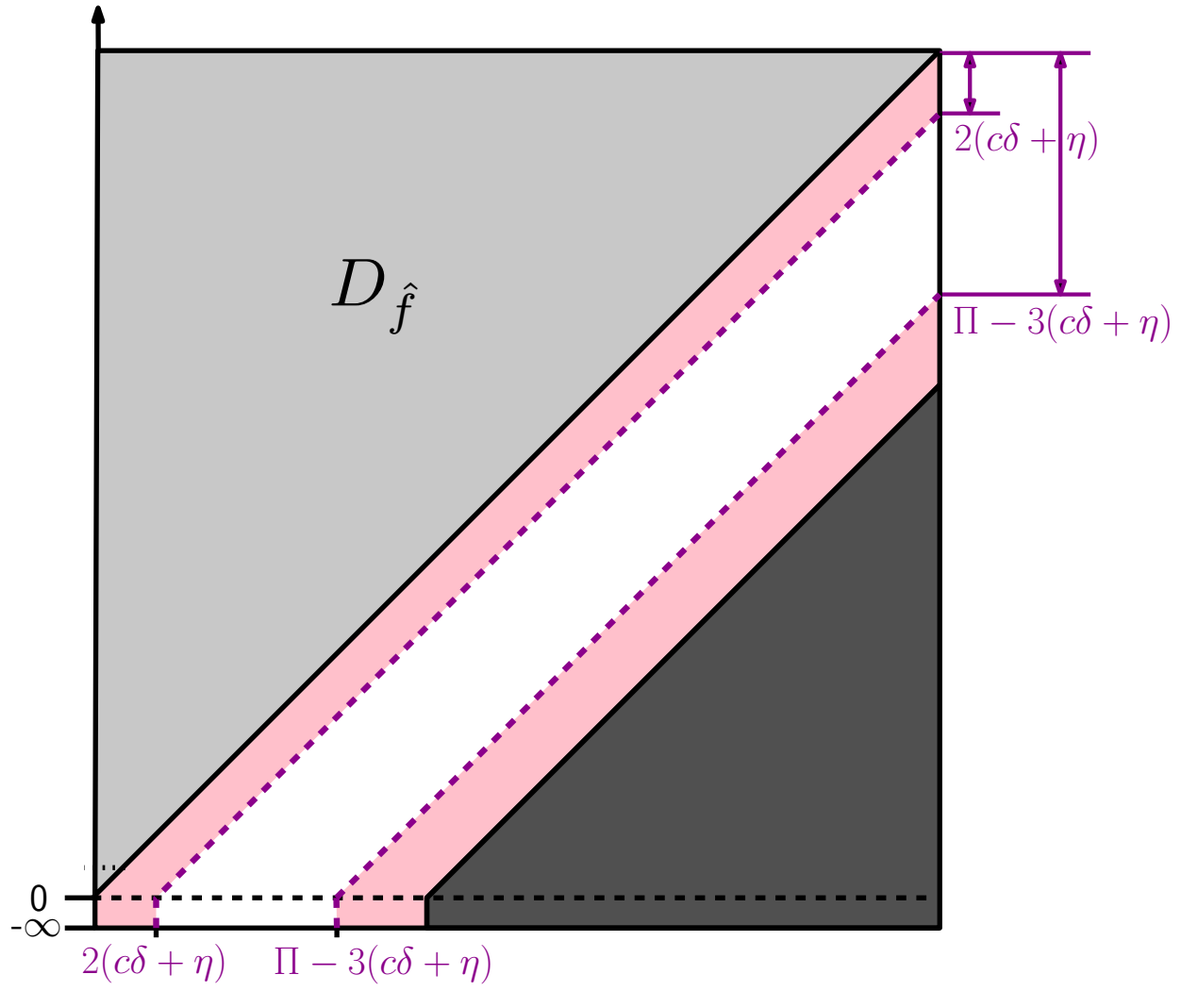
Conclusion:

For any choice of τ such that $2(c\delta+\eta) < \tau < \Pi - 3(c\delta+\eta)$, the number of clusters computed by the algorithm is equal to the number of peaks of f with probability at least $1 - e^{-\Omega(n)}$.

(the Ω notation hides factors depending on c, δ)

Estimating the correct number of clusters

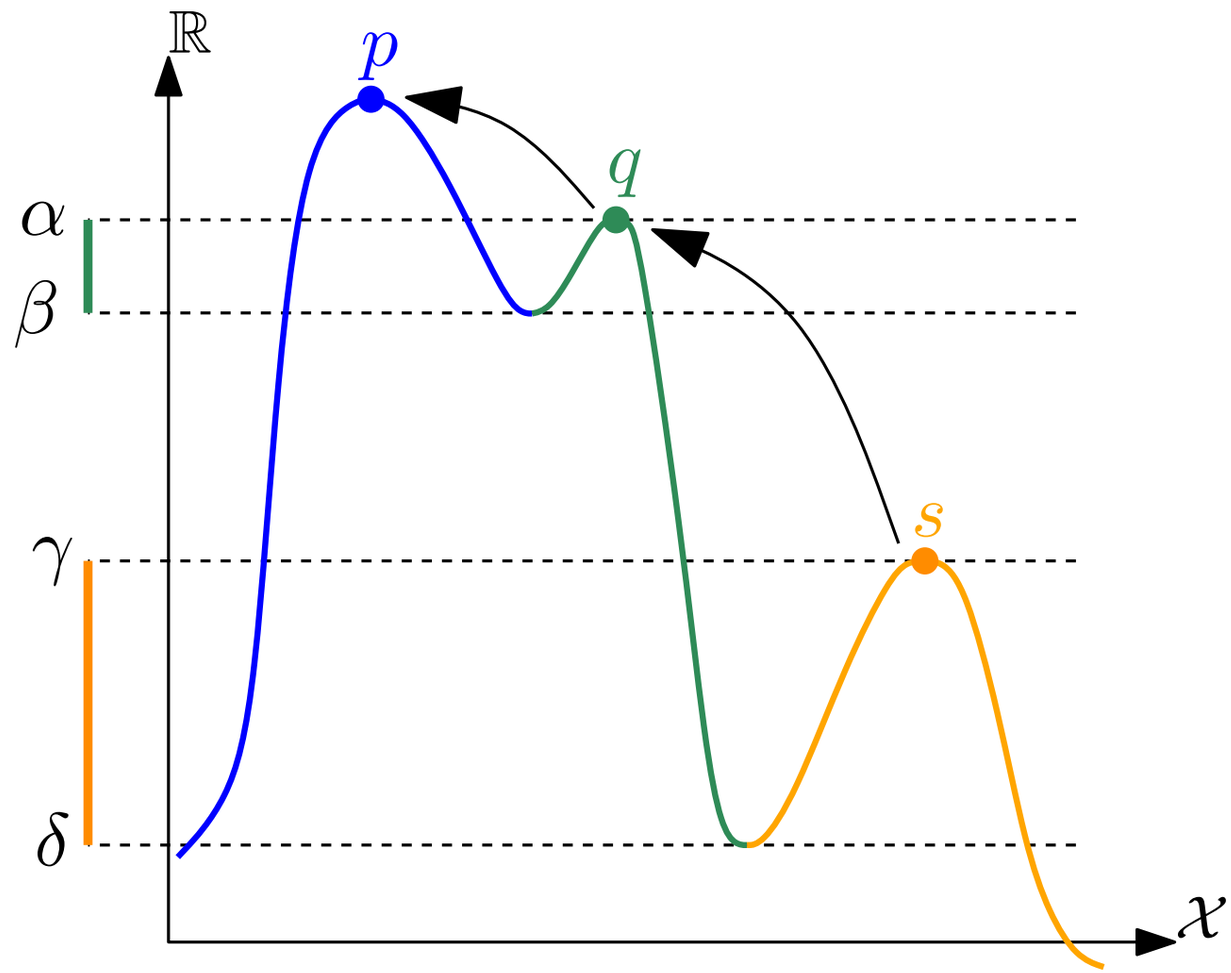




Proof's main ingredient: stability theorem for persistence diagrams

Merging clusters

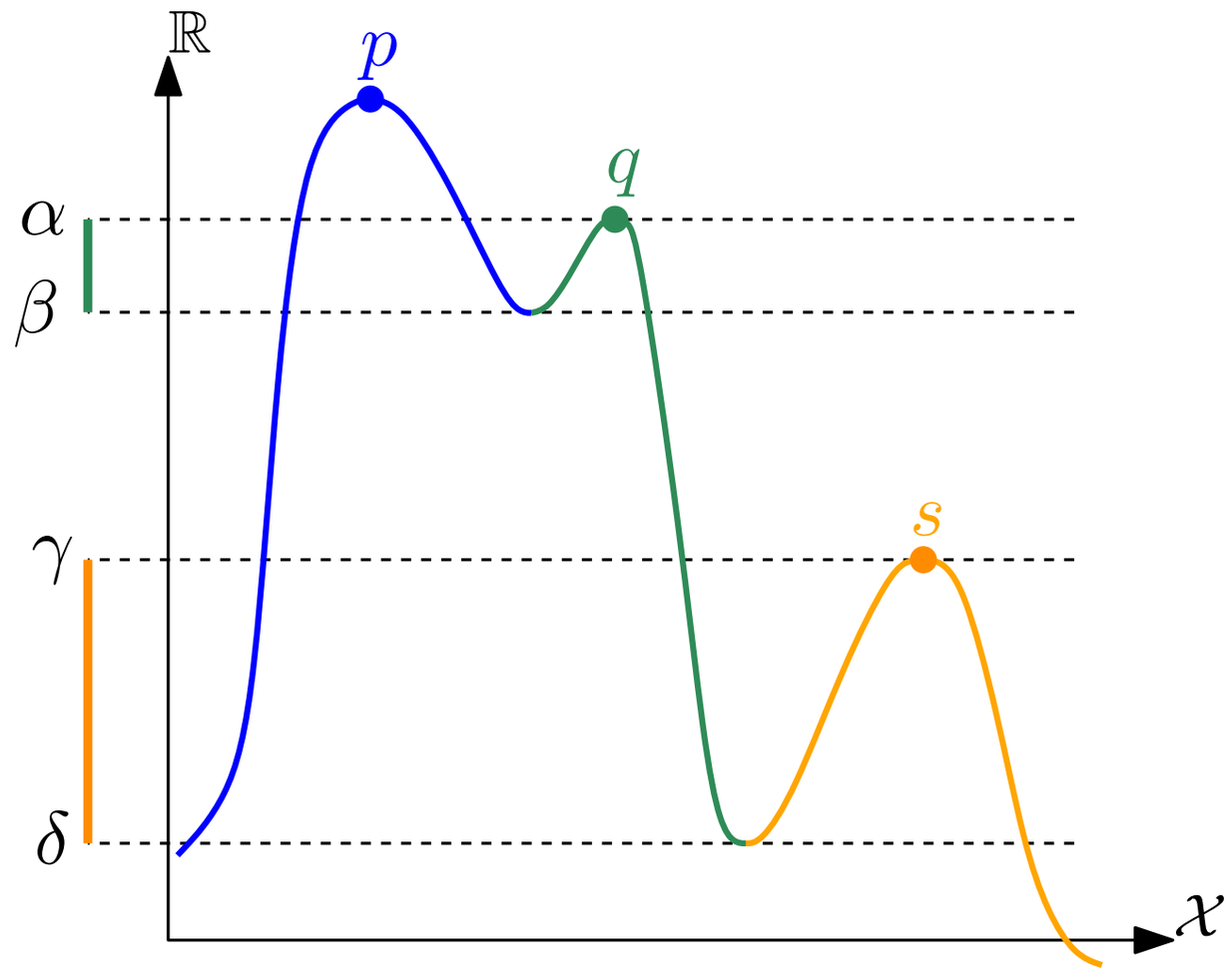
- degree-0 persistence algo. builds a hierarchy of the peaks of $\hat{f}$ (merge tree)
- merge clusters according to the hierarchy (merge each cluster into its parent)



Merging clusters

- degree-0 persistence algo. builds a hierarchy of the peaks of $\hat{f}$ (merge tree)
- merge clusters according to the hierarchy (merge each cluster into its parent)
- given a fixed threshold $\tau \geq 0$, only merge those clusters of prominence $< \tau$

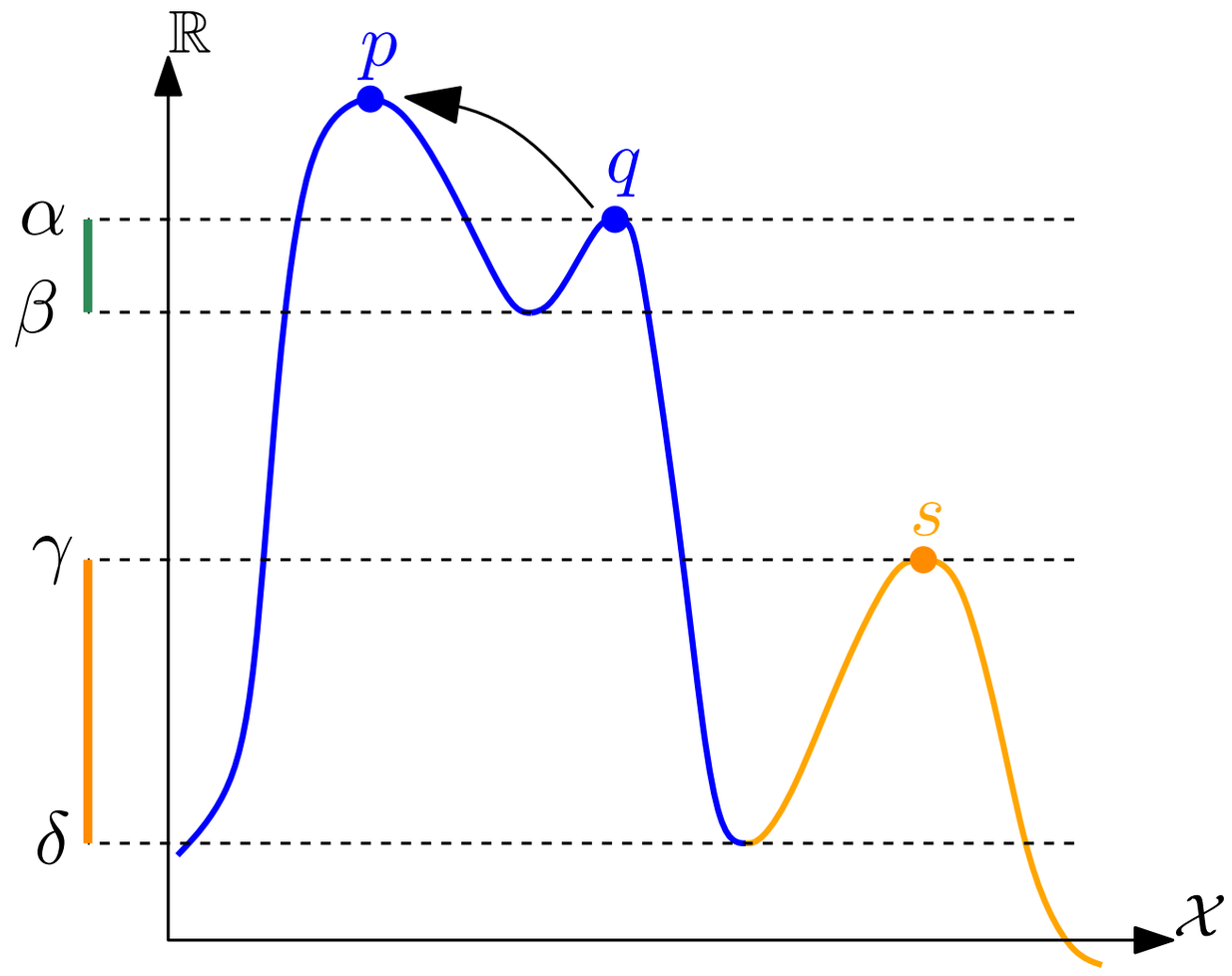
$$0 \leq \tau \leq \alpha - \beta$$



Merging clusters

- degree-0 persistence algo. builds a hierarchy of the peaks of $\hat{f}$ (merge tree)
- merge clusters according to the hierarchy (merge each cluster into its parent)
- given a fixed threshold $\tau \geq 0$, only merge those clusters of prominence $< \tau$

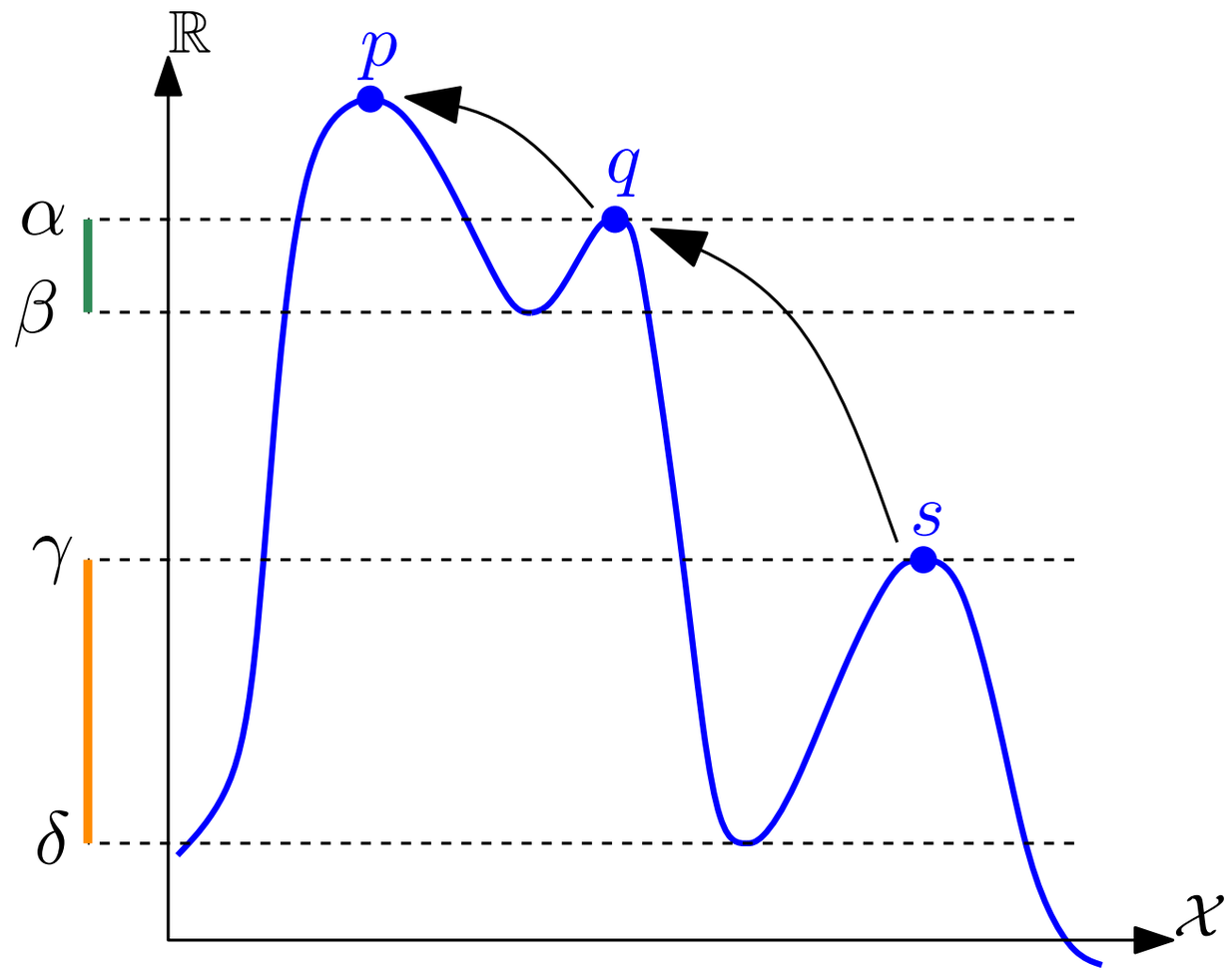
$$\alpha - \beta < \tau \leq \gamma - \delta$$



Merging clusters

- degree-0 persistence algo. builds a hierarchy of the peaks of $\hat{f}$ (merge tree)
- merge clusters according to the hierarchy (merge each cluster into its parent)
- given a fixed threshold $\tau \geq 0$, only merge those clusters of prominence $< \tau$

$$\gamma - \delta < \tau \leq +\infty$$



Pseudo-code

Input: simple graph G with n vertices, n-dimensional vector $\hat{f}$, real parameter $\tau \geq 0$.

Sort the vertex indices $\{1, 2, \ldots, n\}$ so that $\hat{f}(1) \geq \hat{f}(2) \geq \cdots \geq \hat{f}(n)$;
Initialize a union-find data structure $\mathcal{U}$ and two vectors g, r of size n;

for $i = 1$ to n **do**
    Let $\mathcal{N}$ be the set of neighbors of i in G that have indices lower than i;
    if $\mathcal{N} = \emptyset$ *// vertex i is a peak of $\hat{f}$ within G*
        Create a new entry e in $\mathcal{U}$ and attach vertex i to it: $\mathcal{U}.\texttt{MakeSet}(i)$;
        $r(e) \leftarrow i$ *// $r(e)$ stores the root vertex associated with the entry e*
    else *// vertex i is not a peak of $\hat{f}$ within G*
        $g(i) \leftarrow \operatorname{argmax}_{j \in \mathcal{N}} \hat{f}(j)$ *// $g(i)$ stores the approximate gradient at vertex i*
        $e_i \leftarrow \mathcal{U}.\texttt{Find}(g(i))$;
        Attach vertex i to the entry e_i: $\mathcal{U}.\texttt{Union}(i, e_i)$;
    for $j \in \mathcal{N}$ **do**
        $e \leftarrow \mathcal{U}.\texttt{Find}(j)$;
        if $e \neq e_i$ and $\min\{\hat{f}(r(e)),\ \hat{f}(r(e_i))\} < \hat{f}(i) + \tau$
            $\mathcal{U}.\texttt{Union}(e,\ e_i)$;
            $r(e \cup e_i) \leftarrow \operatorname{argmax}_{\{r(e),\ r(e_i)\}} \hat{f}$;
            $e_i \leftarrow e \cup e_i$;

graph-based hill-climbing (1976)

cluster merges with persistence (2013)

Output: the collection of entries e of $\mathcal{U}$ such that $\hat{f}(r(e)) \geq \tau$.

Complexity

Given a neighborhood graph with n vertices (with density values) and m edges:

1. the algorithm sorts the vertices by decreasing density values,

2. the algorithm makes a single pass through the vertex set, creating the spanning forest and merging clusters on the fly using a union-find data structure.

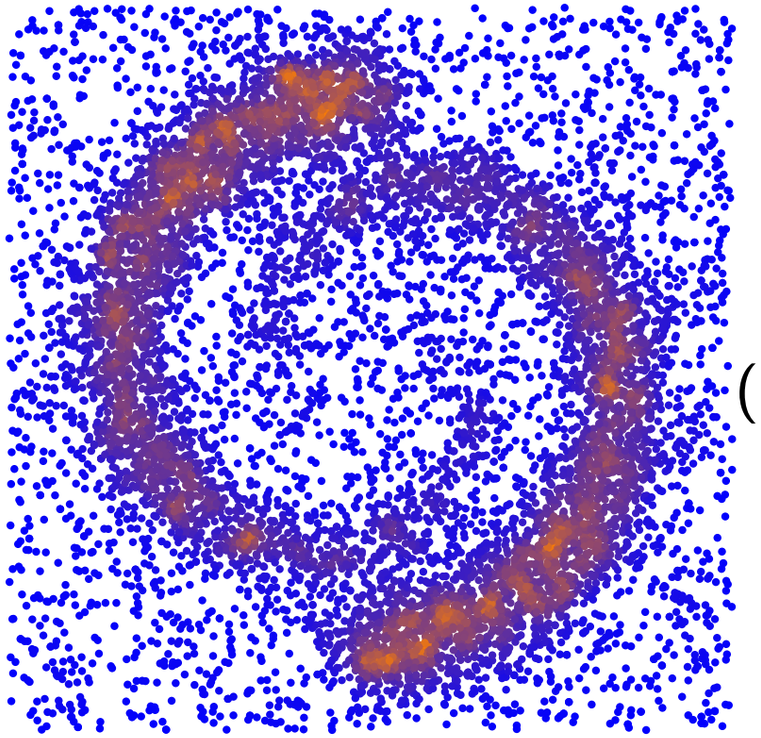
$\rightarrow$ Running time: $O(n \log n + (n + m)\alpha(n))$

$\rightarrow$ Space complexity: $O(n + m)$

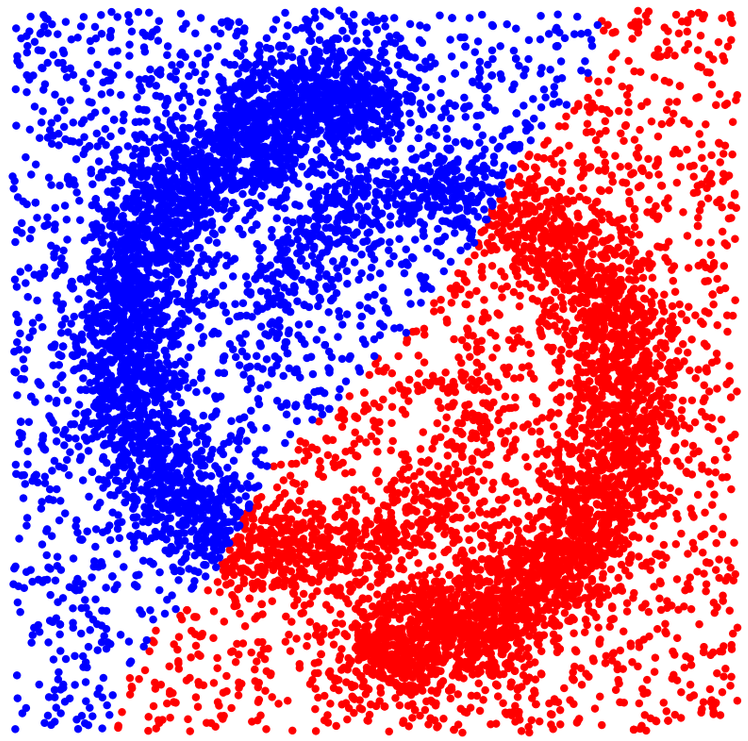
$\rightarrow$ Main memory usage: $O(n)$

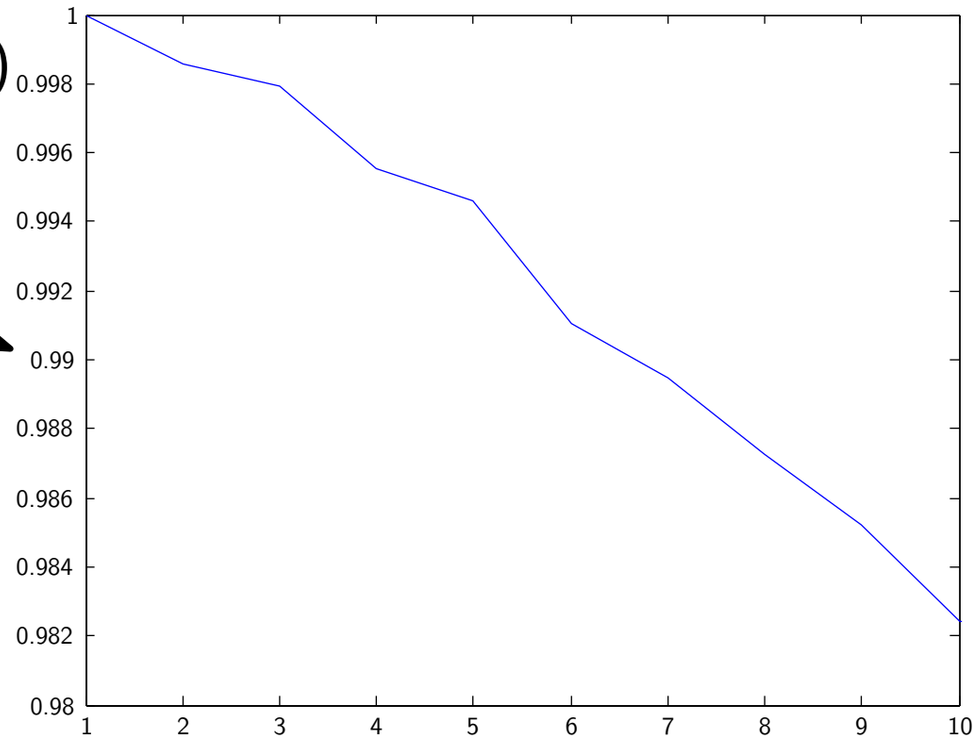
Experimental results

Synthetic Data



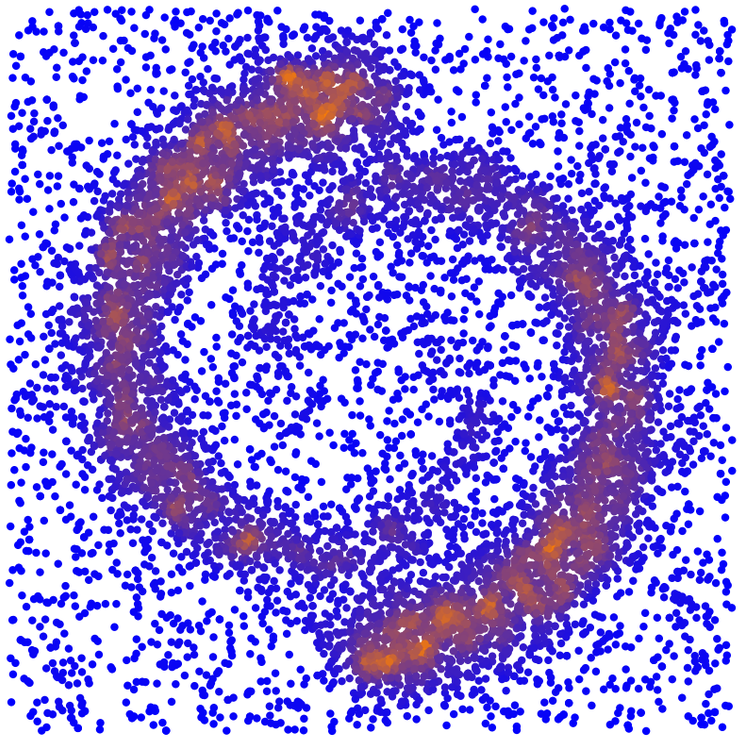
Spectral clustering
(k-means in eigenspace)





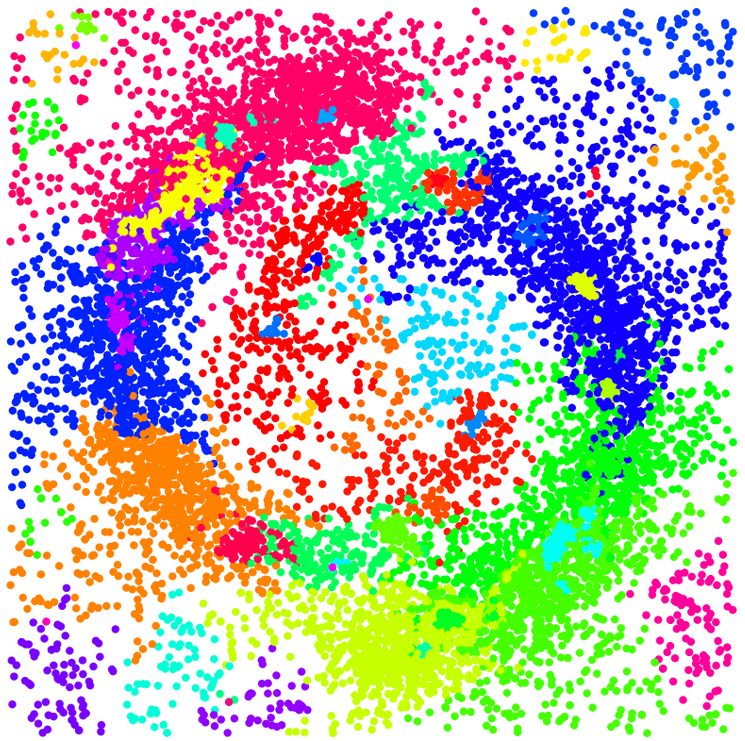
Experimental results

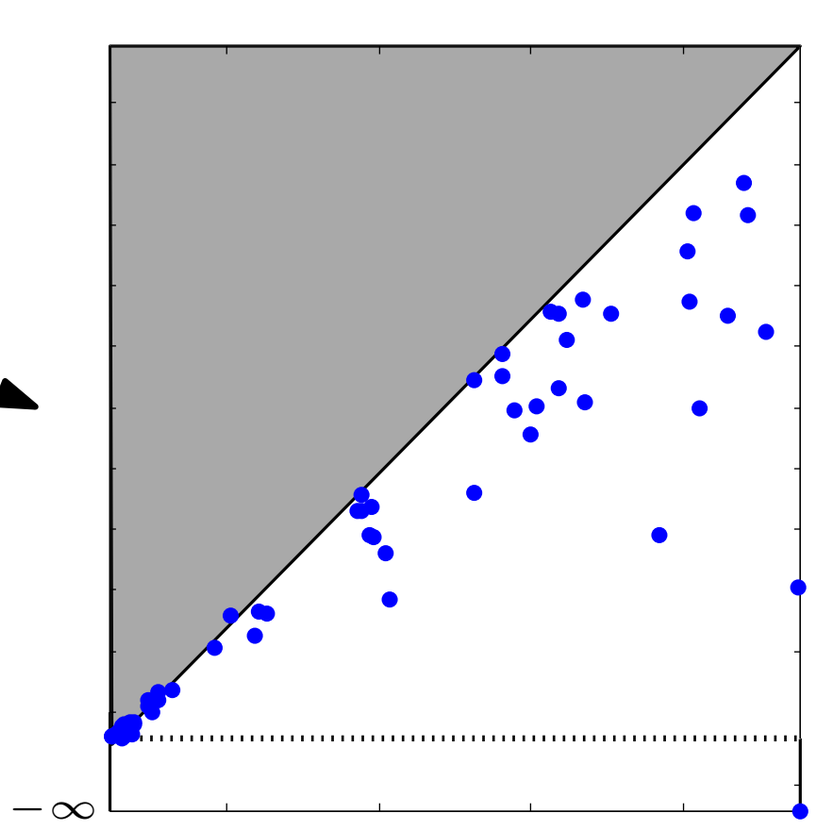
Synthetic Data



$\tau = 0$

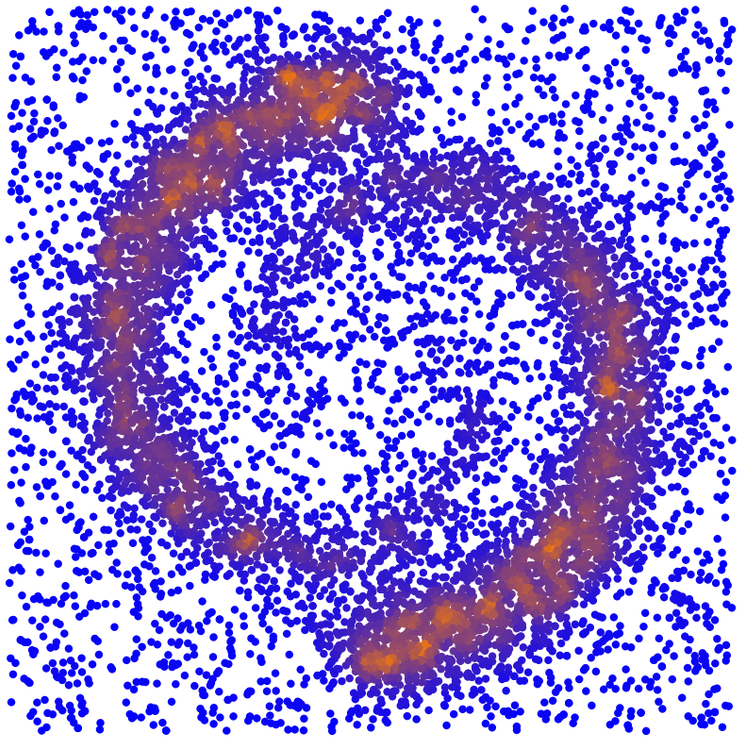
ToMATo



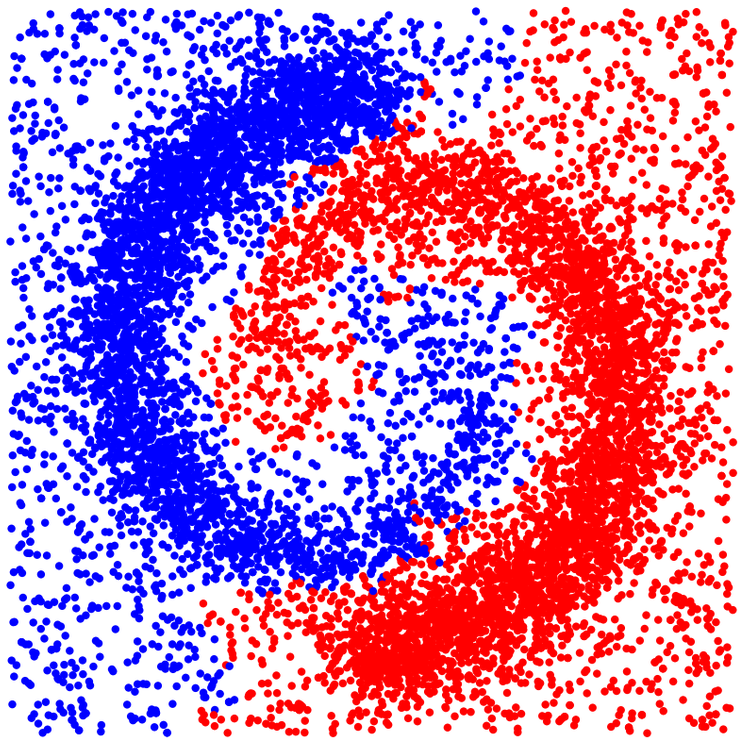


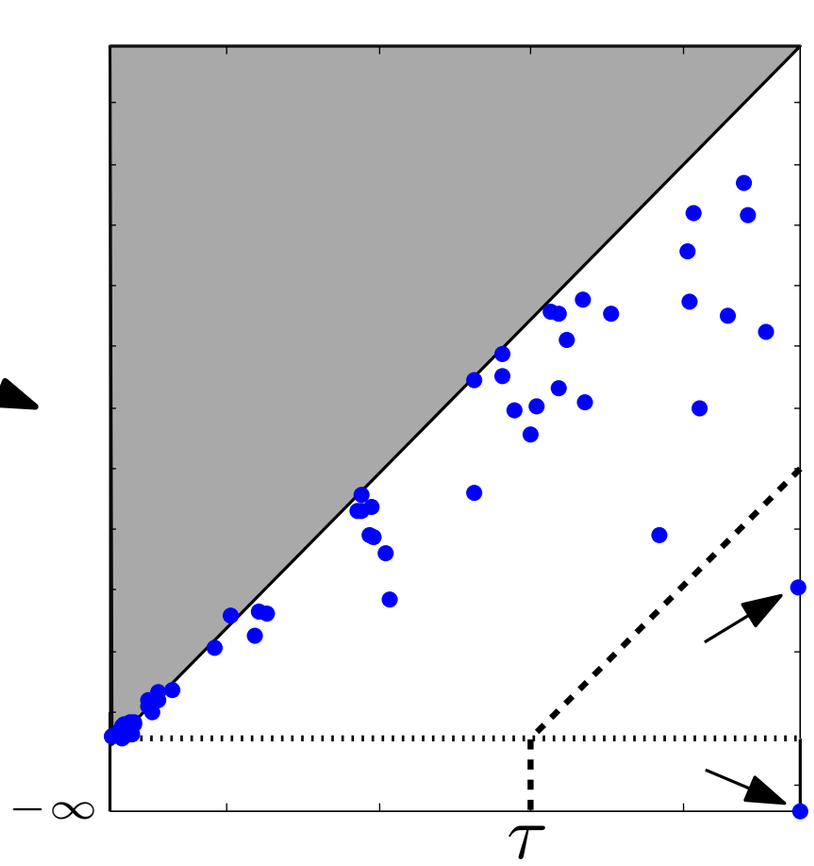
Experimental results

Synthetic Data



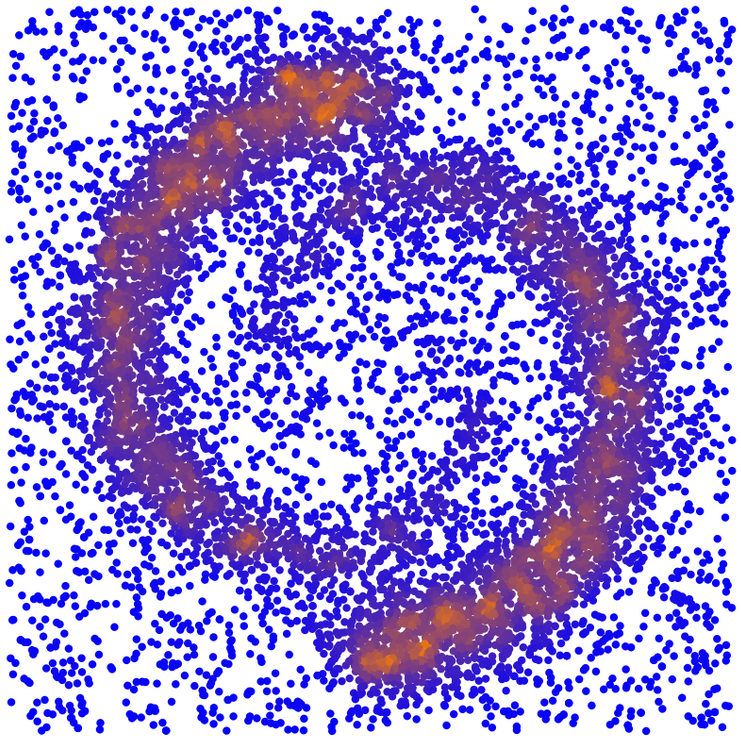
ToMATo

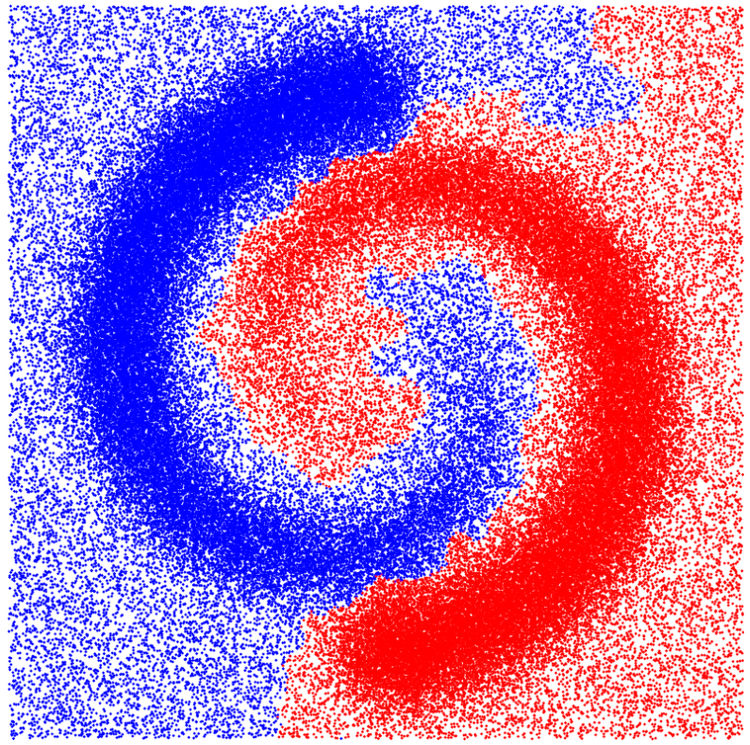


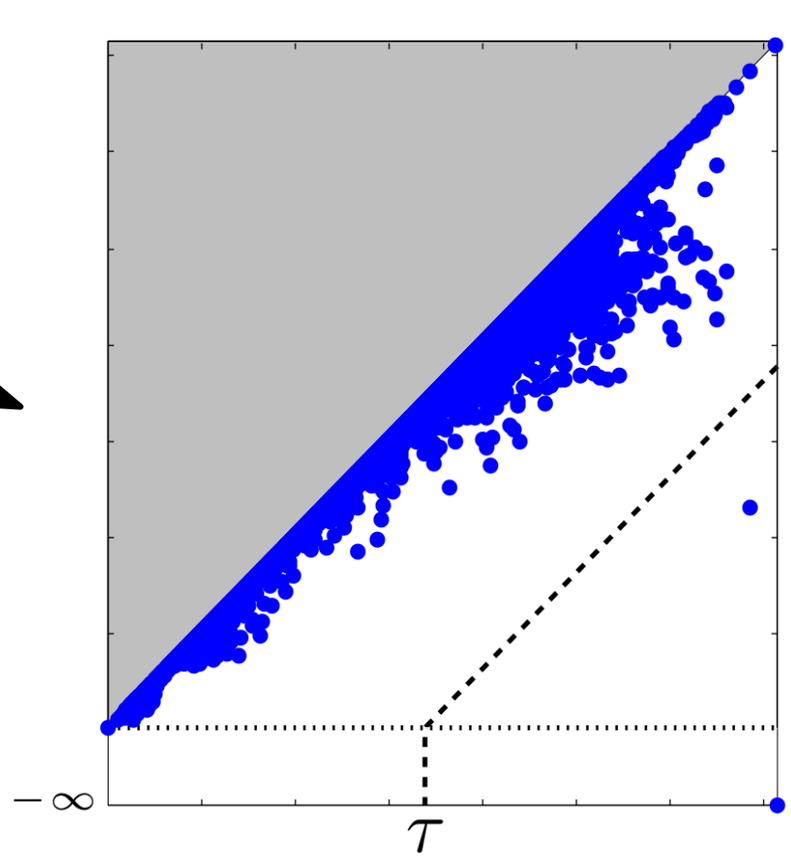


Experimental results

Synthetic Data





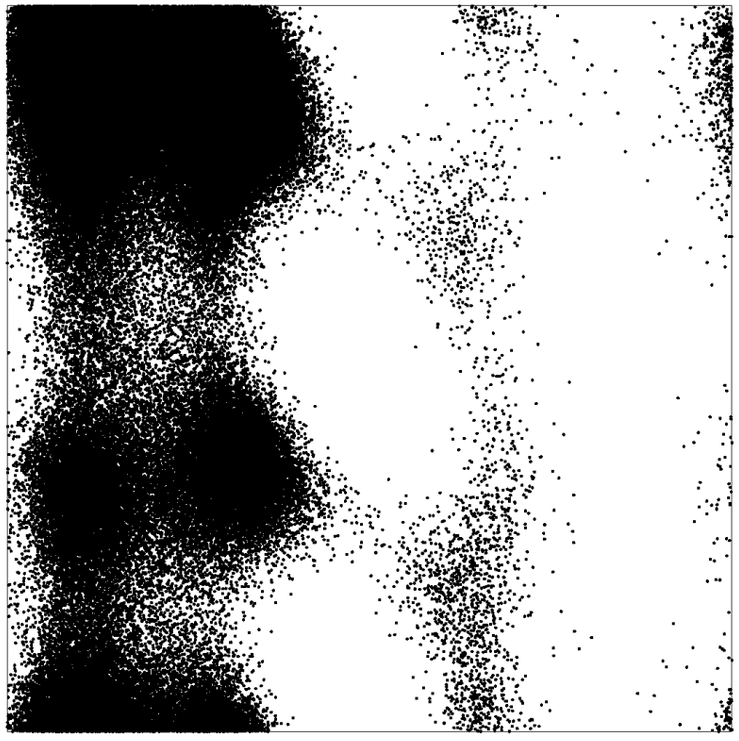


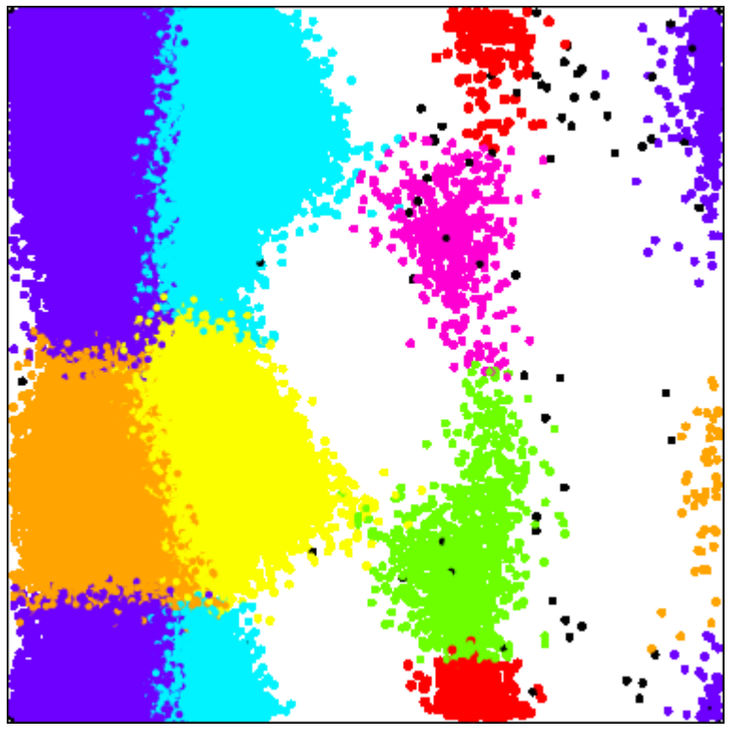
Experimental results

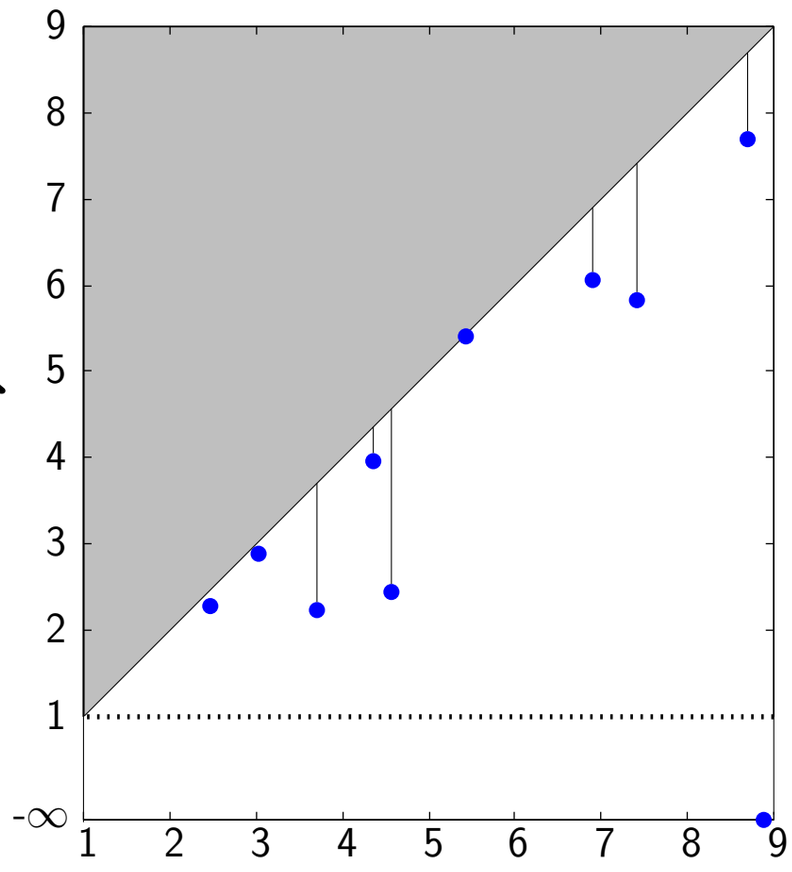
Biological Data

Alanine-Dipeptide conformations ($\mathbb{R}^{21}$)

RMSD distance (non-Euclidean)







Common belief: 6 metastable states

PD shows anywhere between 4 and 7 clusters

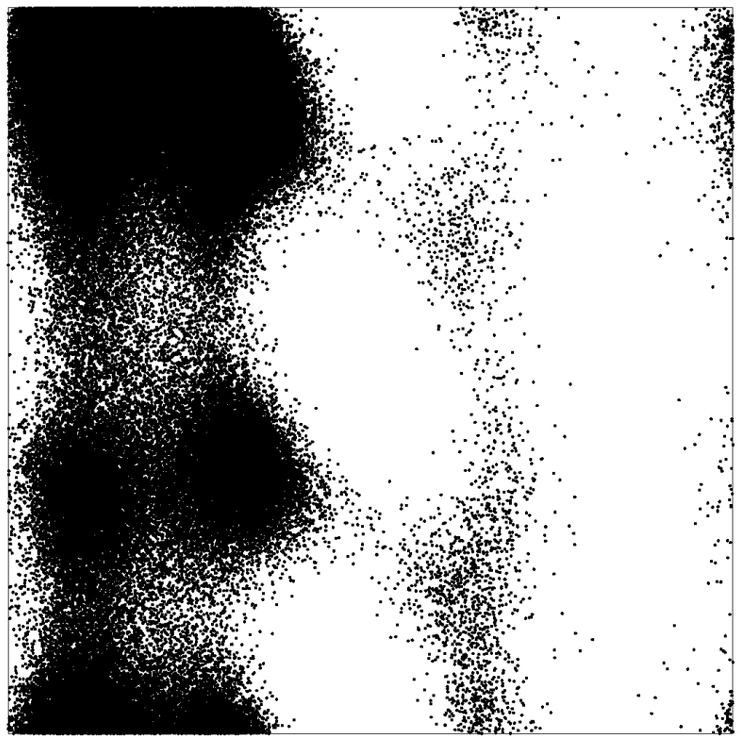
Experimental results

[*Topological methods for exploring low-density states in biomolecular folding pathways*, Yao, Sun, Huang, Bowman, Singh, Lesnick, Guibas, Pande, Carlsson, J. Chem. Phys., 2009]

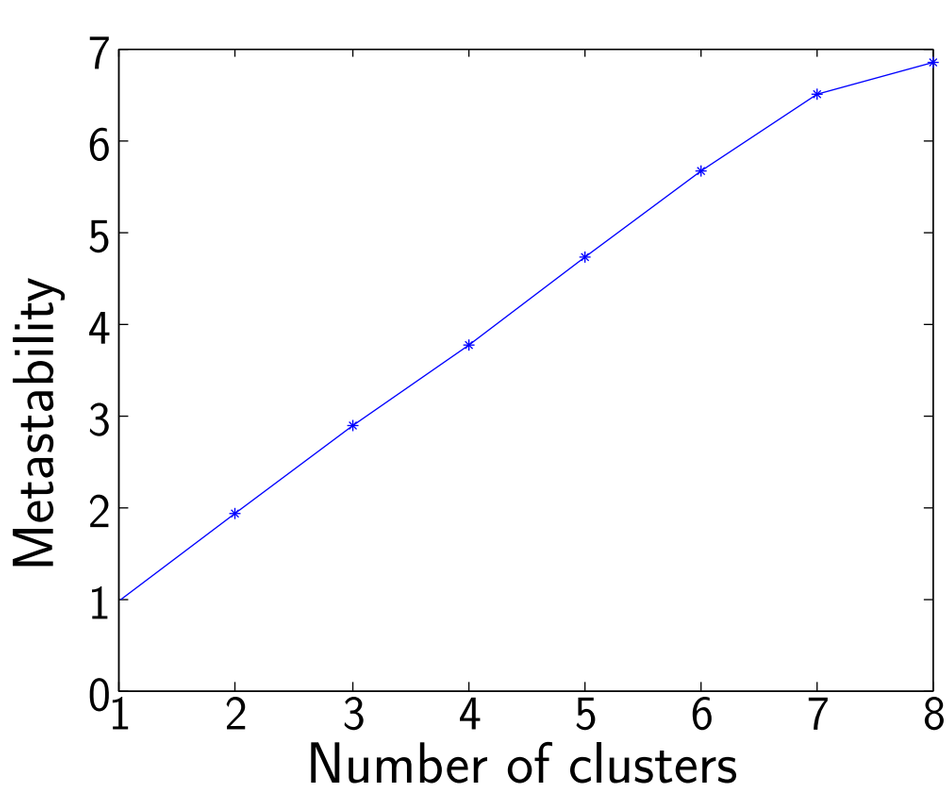
Biological Data

Alanine-Dipeptide conformations ($\mathbb{R}^{21}$)

RMSD distance (non-Euclidean)



Rank	Prominence	Metastability
1	$+\infty$	0.99982
2	3827	1.91865
3	1334	2.8813
4	557	3.76217
5	85	4.73838
6	32	5.65553
7	26	6.50757
8	7.2	6.8193
9	3.0	-
10	2.2	-



Common belief: 6 metastable states

PD shows anywhere between 4 and 7 clusters

Measures of metastability confirm this insight

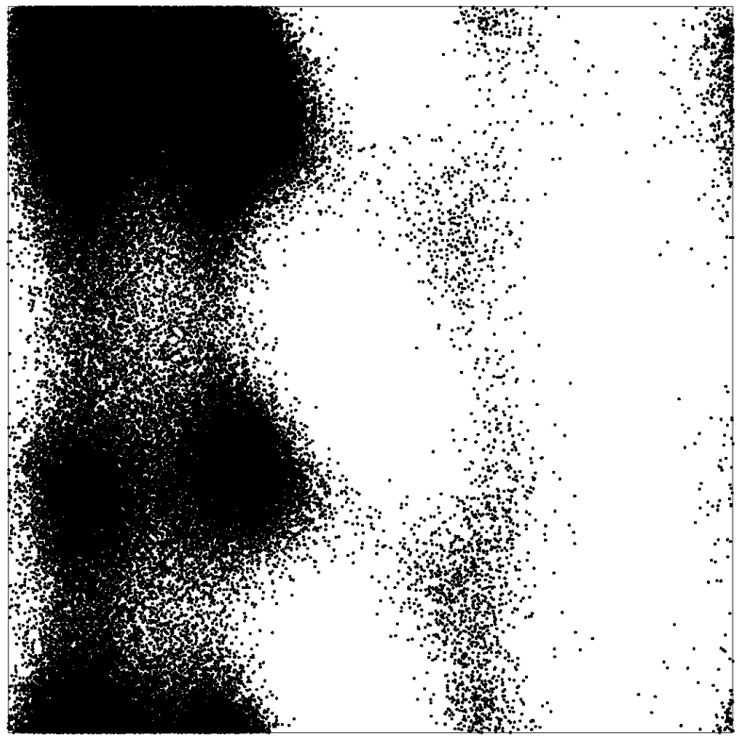
Experimental results

[*Topological methods for exploring low-density states in biomolecular folding pathways*, Yao, Sun, Huang, Bowman, Singh, Lesnick, Guibas, Pande, Carlsson, J. Chem. Phys., 2009]

Biological Data

Alanine-Dipeptide conformations ($\mathbb{R}^{21}$)

RMSD distance (non-Euclidean)



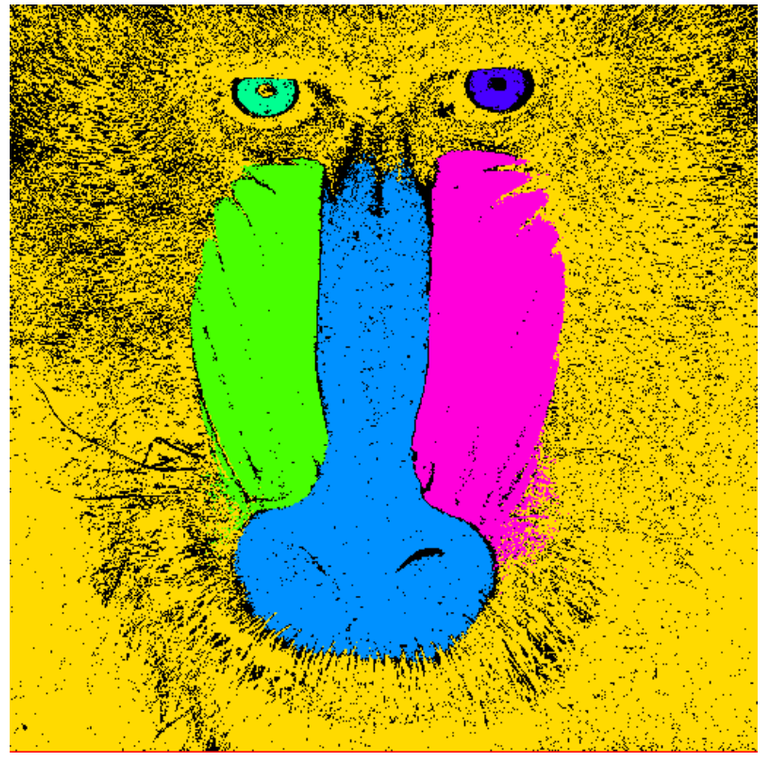
Note: Spectral Clustering takes a week of tweaking, while ToMATo runs out-of-the-box in a few minutes

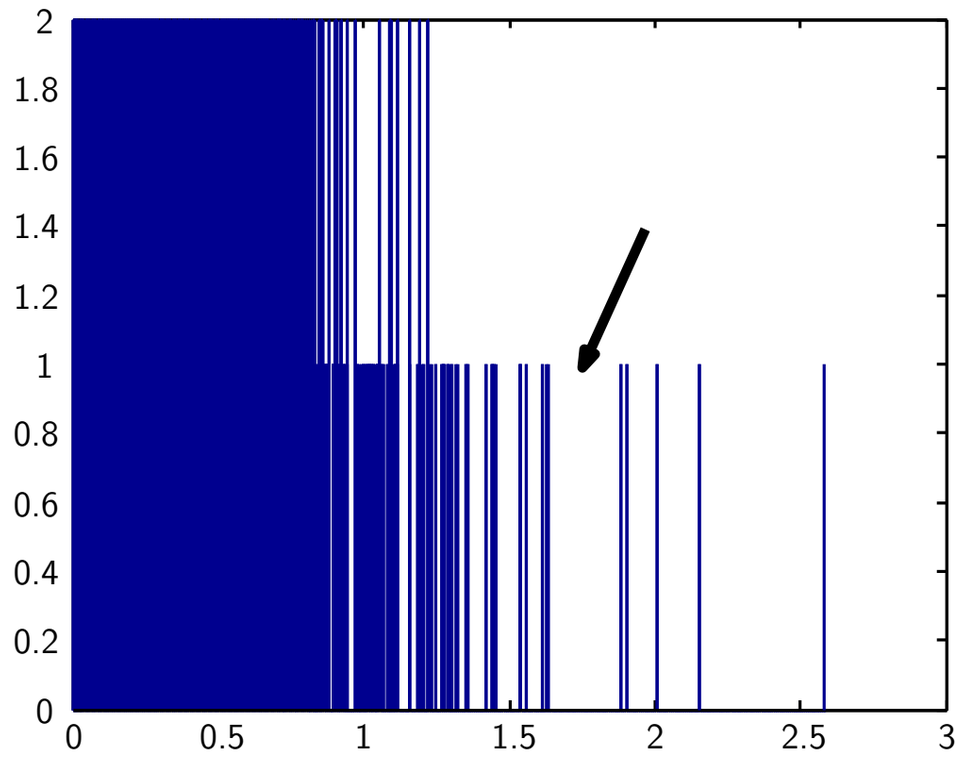
Experimental results

Image Segmentation

Density is estimated in 3D color space (Luv)

Neighborhood graph is built in image domain



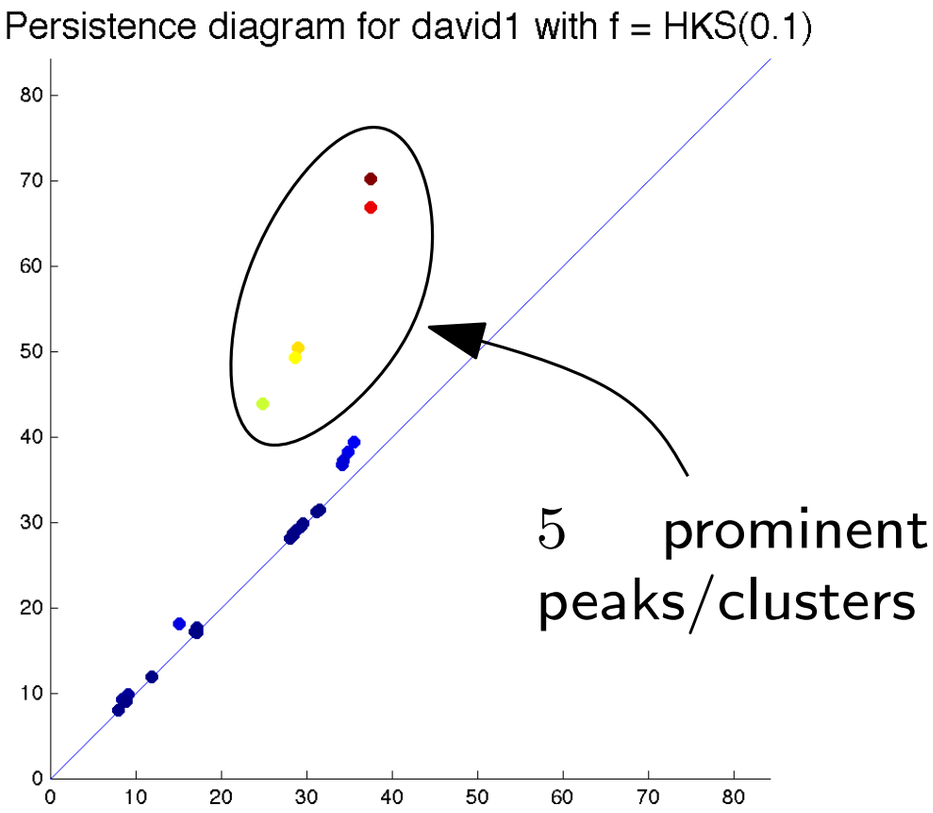


Distribution of prominences does not usually show a clear unique gap

Still, relationship between choice of τ and number of obtained clusters remains explicit

Application to non-rigid shape segmentation

[*Persistence-Based Segmentation of Deformable Shapes*, Skraba, Ovsjanikov, Chazal, Guibas, Proc. CVPR 2010]



X : a 3D shape

f = HKS function on X

Problem: some part of clusters are unstable $\rightarrow$ dirty segments

Application to non-rigid shape segmentation

[*Persistence-Based Segmentation of Deformable Shapes*, Skraba, Ovsjanikov, Chazal, Guibas, Proc. CVPR 2010]

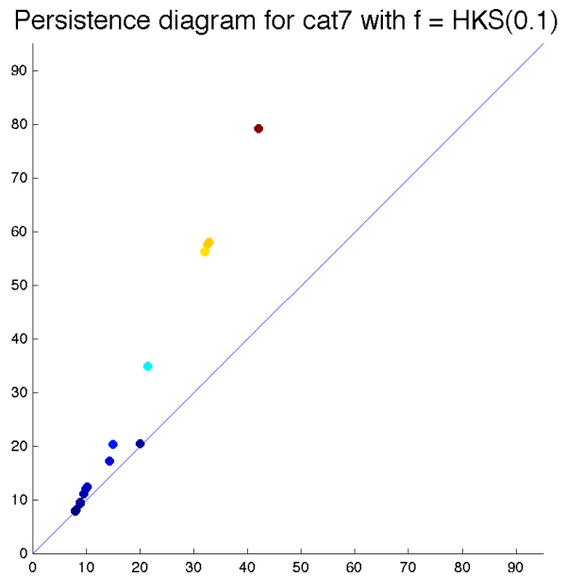
Problem: some part of clusters are unstable $\rightarrow$ dirty segments

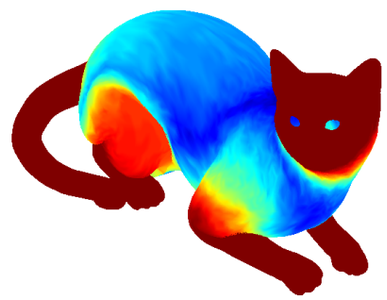
Idea:

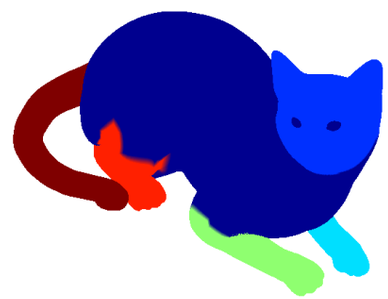
- Run the persistence based algorithm several times on random perturbations of f (size bounded by the "persistence" gap).
- Partial stability of clusters allows to establish correspondences between clusters across the different runs $\rightarrow$ for any $x \in X$, a vector giving the probability for x to belong to each cluster.

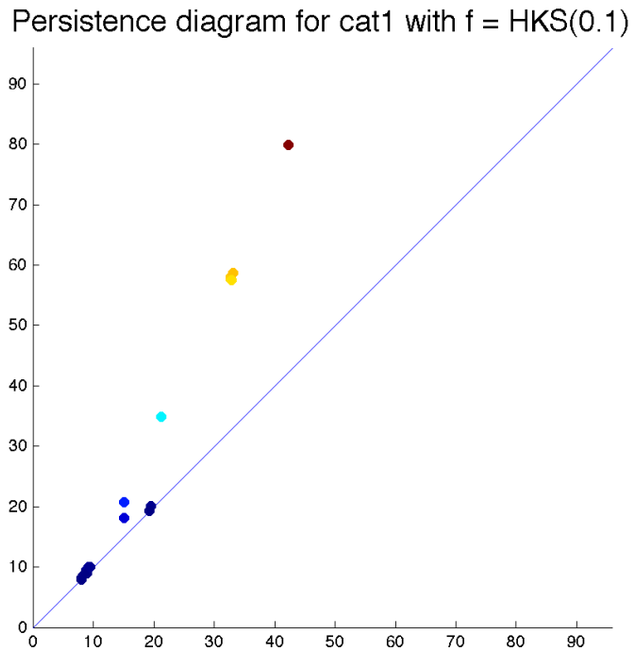
Application to non-rigid shape segmentation

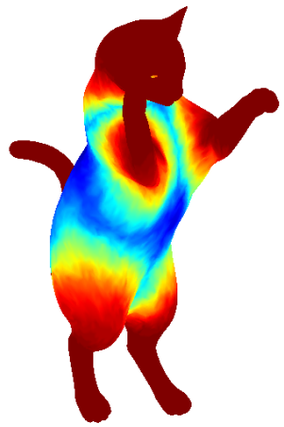
[*Persistence-Based Segmentation of Deformable Shapes*, Skraba, Ovsjanikov, Chazal, Guibas, Proc. CVPR 2010]









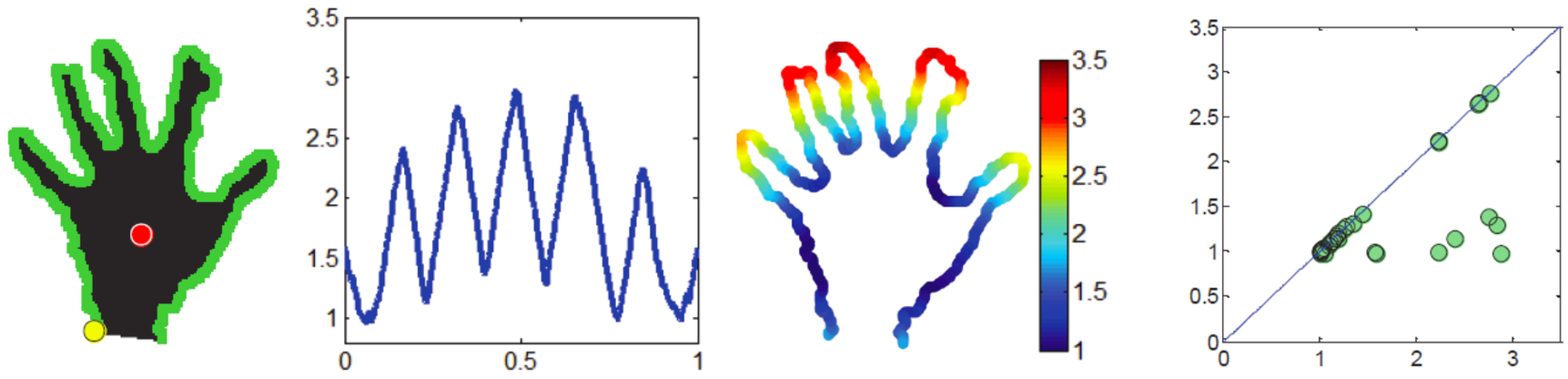


Other applications: classification, object recognition

Examples:

- Hand gesture recognition

[*Persistence-based structural recognition*, Li, Ovsjanikov, Chazal, Proc. CVPR, 2014]



- Persistence-based pooling for shape recognition

[*Persistence-based Pooling for Shape Pose Recognition*, Bonis, Ovsjanikov, Oudot, Chazal, 2015]